

DIPLOMSKI RAD

*Diplomski rad*

**Komunikacija između Linux operativnog sistema i  
hardvera realizovanog u FPGA**

Dejan Lukić 82 / 2014

Predmetni profesor:

**Lazar Saranovac**

Predmetni asistenti:

**Strahinja Janković**

Elektrotehnički fakultet

Univerzitet u Beogradu

Letnji semestar 2017/2018

# Contents

<b>1</b>	<b>Uvod</b>	<b>3</b>
1.1	Sistemi na čipu sa FPGA . . . . .	3
1.2	Opis DE1-SoC . . . . .	3
1.3	Opis Altera Cyclone 5 . . . . .	4
1.3.1	Konfigurisanje FPGA i pokretanje HPS . . . . .	4
1.3.2	HPS-FPGA interfejsi . . . . .	4
1.3.3	Proces pokretanja HPS (boot) . . . . .	5
1.4	Alati . . . . .	5
<b>2</b>	<b>Opis projektovanog sistema</b>	<b>7</b>
2.1	Memorijski mapiran interfejs ka LE diodama i tasterima u FPGA . . . . .	7
2.2	Preloader . . . . .	7
2.3	Bootloader . . . . .	7
2.4	Linuks kernel i Root File System . . . . .	8
2.5	Drajver . . . . .	8
<b>3</b>	<b>Uputstvo za repliciranje rezultata</b>	<b>9</b>
3.1	Projektovanje u Quartus-u . . . . .	9
3.2	Podešavanja okruženja i preuzimanje kompajlera . . . . .	15
3.3	Generisanje RBF fajla . . . . .	15
3.4	Generisanje i kompajliranje Preloader-a . . . . .	15
3.5	Generisanje Device Tree . . . . .	16
3.6	U-Boot . . . . .	16
3.7	Linuks kernel . . . . .	18
3.8	Kompajliranje drajvera . . . . .	19
3.9	Generisanje Root File System . . . . .	19
3.10	Priprema SD kartice . . . . .	20
3.11	Testiranje sistema . . . . .	21
<b>4</b>	<b>Zaključak</b>	<b>23</b>
<b>5</b>	<b>Dodatak</b>	<b>24</b>
5.1	Izvorni kod drajvera . . . . .	24
5.2	Skripta za U-Boot . . . . .	28

# 1 Uvod

U ovom radu je na DE1-SoC razvojnom sistemu implementiran jednostavan hardver u FPGA, portovan je Linux operativni sistem i napisan je drajver za pristup registrima i prihvatanje prekida iz FPGA.

## 1.1 Sistemi na čipu sa FPGA

Sa sve većim mogućnostima namenskih sistema došlo je do popularizacije sistema na čipovima (SoC - System on Chip) koji integrišu mikroprocesore sa više jezgara, memorije na čipu, mnogobrojne periferije i transivere, kao i FPGA (Field Programmable Gate Array).

Ova tehnologija daje dizajneru sistema veliku slobodu i mogućnosti, a zadržava se klasičan postupak projektovanja namenskih sistema. Uz to se ostvaruje veća integracija, manja potrošnja, manja površina štampane ploče (PCB - Printed Circuit Board) i veći protok podataka između procesora i FPGA dela.

Uobičajena primena ovih sistema je implementacija specifičnih akceleratora koji ubrzavaju izvršavanje algoritama i implementacija specifičnih programabilnih interfejsa ka spoljnom svetu. Nove tehnologije kao što su OpenCL, Vivado HLS, Matlab HDL Coder omogućavaju kompatibilnost dizajna softvera na visokom nivou i implementiranog hardvera na niskom nivou.

SoCFPGA sistemi najčešće sadrže ARM mikroprocesor. Aplikacije na mikroprocesoru bez operativnog sistema (baremetal) nude jednostavno pisanje koda i uštedu na resursima. Za kompleksnije aplikacije koriste se operativni sistemi (OS) i time se olakšava integrisanje mrežnih protokola, rad sa multimedijalnim sadržajima, kriptografskim bibliotekama kao i mnoge druge mogućnosti koje su dostupne kao open-source softver. Kada je potrebno garantovati reakciju u određenom vremenu na neki spoljni događaj veliki operativni sistemi nisu dobro rešenje i koriste operativni sistemi u realnom vremenu (RTOS).

Hardver u FPGA se projektuje upotrebom nekog dva popularna jezika za opis hardvera (Verilog i VHDL - Very High Speed Integrated Circuit Hardwer Description Language) i softverskih alata za specifični uređaj. Dodatno ovi alati olakšavaju dizajn upotrebom IP( Intellectual Property) blokova kao i generisanjem raznih izlaznih fajlova koji opisuju projektovani hardver na standardni način i koriste se prilikom razvoja softvera.

## 1.2 Opis DE1-SoC

U ovom radu korišćen je DE1-SoC razvojni sistem koji se vrlo često upotrebljava u edukativne svrhe. Razvojni sistem je zasnovan na čipu iz familije Cyclone V kompanije Intel (ranije Altera).

U nastavku su navedene samo osobine razvojnog sistema koje se tiču ovog rada, a detaljniji opis se može pronaći u dokumentu DE1-SoC User Manual [] dodati referencu: <https://www.terasic.com.tw/HOWTO-get-software-manual/User-Manual/rev.E/Board.html> SoC User Manual(rev.E Board) Terasic

- Sistem na čipu Cyclone V SoC 5CSEMA5F31
- Memorija 1GB (2x256Mx16) DDR3 SDRAM povezana na HPS
- Slot za Micro SD karticu povezan na HPS
- UART na USB (USB Mini-B konektor)
- 5 debaunsiranih tastera (FPGA x4, HPS x1)
- 11 LE dioda (FPGA x10, HPS x 1)
- 12V DC napajanje

## 1.3 Opis Altera Cyclone 5

Altera Cyclone 5 je SoC FPGA koji se sastoji od dva dela(slika): procesorski deo (HPS - Hard processor System) i programabilni FPGA deo. HPS se sastoji od MPU (Microprocessor unit) sa ARM Cortex-A9 MPCore sa dva jezgra i sledećih modula: kontroleri memorije, memorije, periferije, sistem interkonekcije, debug moduli, PLL moduli. FPGA deo se sastoji od sledećih delova: FPGA programabilna logika (look-up tabele, RAM memorije, množači i rutiranje), kontrolni blok, PLL, kontroler memorije.

Svaki pin kućista je povezan na samo jedan od ova dva dela sistema, tako da HPS deo i FPGA deo ne mogu međusobno razmenjivati pinove.

### 1.3.1 Konfigurisanje FPGA i pokretanje HPS

Pri pokretanju HPS (boot) može da učitava program iz FPGA dela, iz eksterne flash memorije ili preko JTAG. FPGA ima mogućnost da se konfiguriše softverski iz HPS korišćenjem periferije FPGA Manager ili spoljnim programatorom. Kombinacije ovih mogućnosti daju nekoliko scenarija:

- nezavisno konfigurisanje FPGA i pokretanje HPS
- konfigurisanje FPGA, zatim pokretanje HPS iz memorije koja se nalazi u FPGA
- pokretanje HPS, zatim konfigurisanje FPGA iz HPS

DE1-SoC razvojni sistem dolazi sa integrisanim programatorom kojem se pristupa preko USB porta. Moguće je podesiti konfigurisanje FPGA spolja ili iz HPS upotrebom prekidača MSEL, dok se HPS uvek pokreće iz flash memorije SD kartice. (dodati tabelu 3-2 iz de1soc user guide)

### 1.3.2 HPS-FPGA interfejsi

HPS-FPGA interfejsi su komunikacioni kanali između HPS i FPGA dela. U nastavku su nabrojani i opisani HPS-FPGA interfejsi:

- FPGA-to-HPS bridge - magistrala visokih performansi konfigurabilne širine od 32,64 ili 128 bita. Na ovoj magistrali je FPGA master. Ovaj interfejs otkriva FPGA masterima ceo adresni prostor HPS dela.
- HPS-to-FPGA bridge - magistrala visokih performansi konfigurabilne širine od 32,64 ili 128 bita. Na ovoj magistrali je HPS master a u FPGA se nalazi slave.
- Lightweight HPS-to-FPGA - magistrala širine 32 bita. HPS je master na ovoj magistrali. Ovaj interfejs manjeg protoka je namenjen za pristup statusnim i kontrolnim registrima periferijama implementiranim u FPGA delu.
- FPGA manager - HPS periferija koja komunicira sa FPGA delom prilikom konfiguracije ili pokretanja (boot)
- Prekidi - mogućnost povezivanja prekida iz FPGA na HPS kontroler prekida
- HPS debug interfejs - omogućava da se debug mogućnosti prošire i na FPGA deo

Interfejsi koji su produžetak AXI magistrale na FPGA deo su FPGA-to-HPS bridge, HPS-to-FPGA bridge i Lightweight HPS-to-FPGA. Za povezivanje na ovu magistralu sa strane FPGA koristi se Avalon magistrala, stoga je neophodan AXI-Avalon bridge.

### 1.3.3 Proces pokretanja HPS (boot)

Pokretanje HPS je proces koji se obavlja u više koraka. Nakon izvršavanja svakog koraka se učitava i pokreće sledeći. Ovo je proces je sličan kod svih ARM procesora, a u nastavku je ukratko opisan za konkretnu platformu.

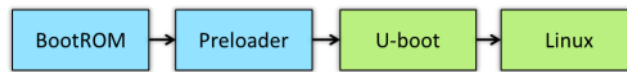


Figure 1: Tok pokretana sistema

Pri izlazu iz reset stanja procesor počinje izvršavanje sa reset vektora iz memorije na čipu. Na adresi reset vektora je upisan Boot ROM program. Ovo je prvi korak u pokretanju HPS. BootROM izvršava osnovna podešavanja procesora i dohvata Preloader iz NOR flash memorije, NAND flash memorije ili SD/MMC flash memorije. Očitavaju se BSEL pinovi na osnovu kojih se određuje gde je smešten Preloader, zatim se inicijalizuje taj interfejs i učitava i pokreće Preloader. Boot ROM softver proizvođača i ne može se menjati.

Preloader je prvi korak u pokretanju koji može da se konfiguriše. Preloader koji se koristi u ovom radu je zasnovan na SPL (Secondary Program Loader) framework koji je deo U-Boot projekta, što znači da Preloader i U-Boot dele dosta izvornog koda, kao što je mnoštvo pouzdanih drajvera. Preloader obično izvršava inicijalizaciju SDRAM, dodatna podešavanja sistema, inicijalizaciju flash kontrolera koji sadrži sledeći program (NAND, SD/MMC, QSPI) i zatim učitavanje programa u RAM memoriju i pokretanje.

Softver koji sledi nakon Preloader-a može biti baremetal aplikacija ili bootloader. Preloader i svi prethodni programi se izvršavaju na prvom jezgri procesora dok je drugo u reset stanju. Naredni koraci mogu inicijalizovati drugo jezgro.

Bootloader ima zadatak da podesi promenljive okruženja operativnog sistema, dohvati fajlove za pokretanje operativni sistem (sa flash memorije, putem Etherneta preko TFTP protokola ili USB), konfiguriše FPGA pruži konzolu za korisničke operacije. Neki od popularnih open-source bootloadera su U-Boot i Barebox.

## 1.4 Alati

U nastavku će ukratko biti opisani korišćeni alati sa izdvojenim najvažnijim mogućnostima:

- Quartus Prime 18.0 - alat za razvoj hardvera na FPGA. Deo paketa je Platform Designer (ranije Qsys) koji u dizajn uključuje HPS, IP blokove i definiše povezanost ovih delova
- Preloader Generator (bsp-editor alat iz SoC EDS) - Generise izvorni kod Preloader-a na osnovi izlaznih fajlova koji opisuju hardver
- Device Tree Generator (sopc2dts alat iz SoC EDS) - Generise Device Tree opis hardvera na osnovi izlaznih fajlova koji opisuju hardver
- DE1-SoC Builder - Generise prazan Quartus projekat za DE1-SoC razvojni sistem
- Linaro Toolchain - koristi se za kompajliranje softvera

Na slici 2 je grafički prikazan tok projektovanja jednog SoCFPGA sistema.

U nastavku su objašnjeni fajlovi koji se koriste pri projektovanju:

- .qpf - projektni fajl za Quartus. Ovaj fajl generiše DE1-SoC Builder
- .qsf - skripta za podešavanje pinova. Ovaj fajl generiše DE1-SoC Builder

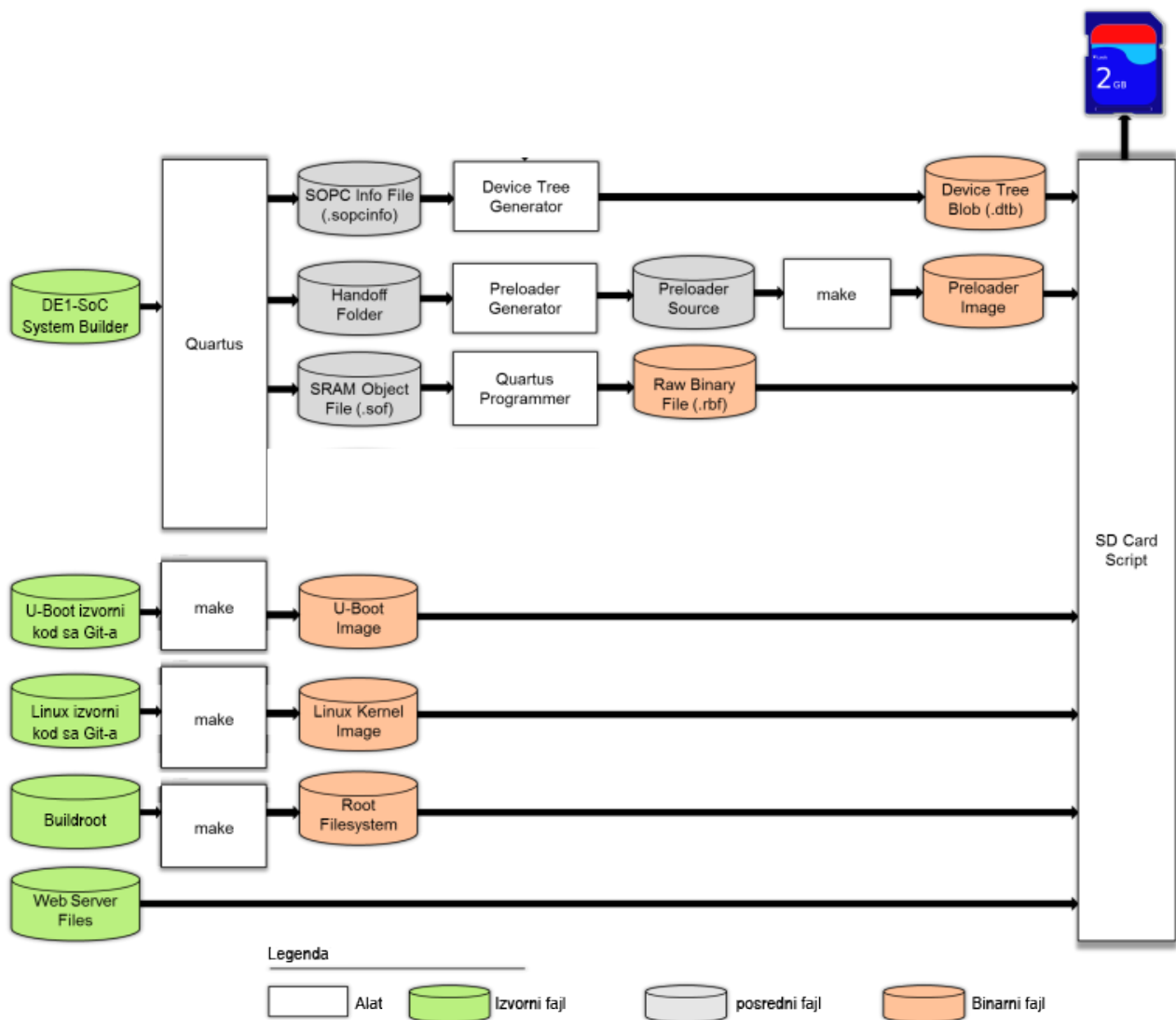


Figure 2: Tok projektovanja

- `.sdc` - skripta za podešavanje takta. Ovaj fajl generiše DE1-SoC Builder
- `.v` - Verilog HDL izvorni kod
- `.vhd` - VHDL izvorni kod
- `.sof` - SDRAM Object File - fajl za programiranje FPGA. Ovaj fajl generiše Quartus pri kompajliranju dizajna
- `.rbf` - Raw Binary File - fajl za programiranje FPGA. Ovaj fajl se dobija konverzijom `.sof` alatom `quartus_cpf`
- `.dts` - Device Tree Source - opis hardvera za Linuks kernel
- `.dtb` - Device Tree Blob - binarni fajl, kompajlirani opis hardvera za Linuks kernel
- `.sopcinfo` - sadrži opis hardvera na osnovu kog se generišu drugi fajlovi. Ovaj fajl generiše Platform Designer
- `.c` - izvorni kod u jeziku C
- `Makefile` - sadrži set direktiva za `make` buld system

## 2 Opis projektovanog sistema

U ovom radu je implementiran jednostavan sistem koji demonstrira osnovne mogućnosti u dizajniranju sistema na SoC FPGA. Na slici 7 je prikazan realizovani sistem (dodati novu sliku?)

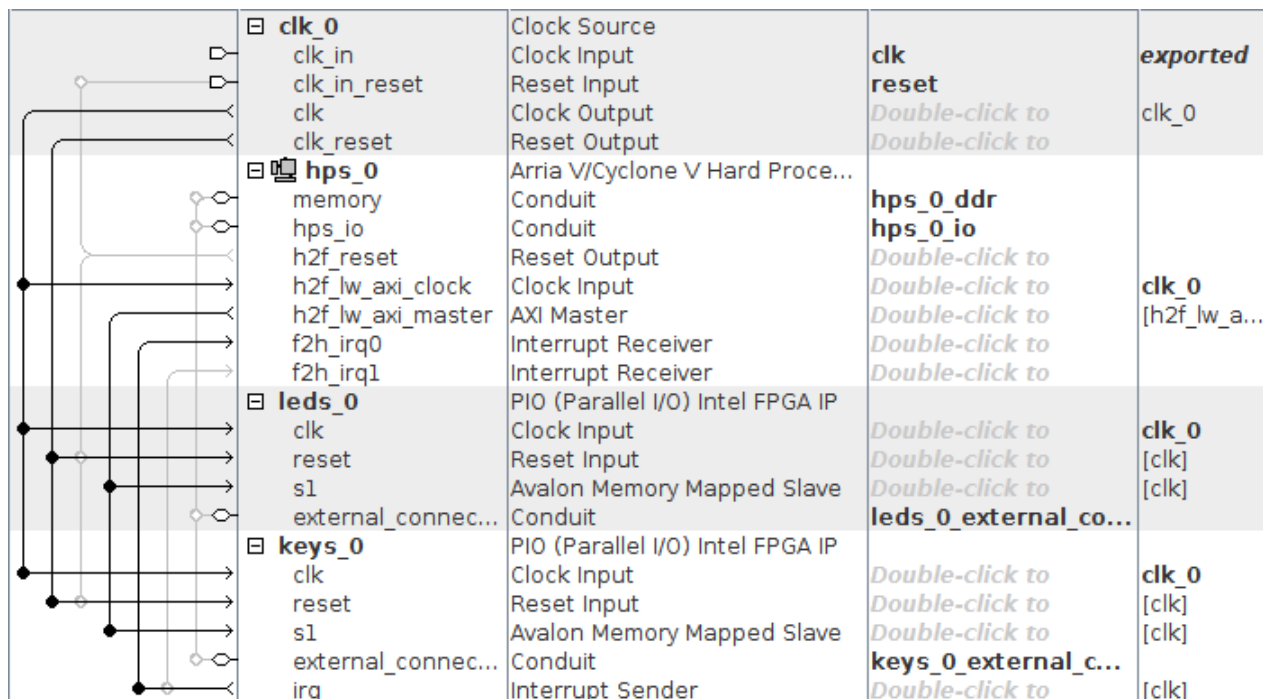


Figure 3: Povezivanje keys\_0 bloka

### 2.1 Memorijski mapiran interfejs ka LE diodama i tasterima u FPGA

U FPGA delu su postavljena dva PIO (Parallel I/O) Intel FPGA IP bloka. PIO IP blok `leds_0` je izlazni i koristi se za kontrolisanje LE dioda. PIO IP blok `keys_0` je ulazni i koristi se za očitavanje tastera. PIO IP blok `keys_0` takođe šalje prekidni zahtev HPS-u na svaku uzlaznu i silaznu ivicu.

PIO blokovi `keys_0` i `leds_0` su povezani kao Memory Mapped Slave na Avalon magistralu. Prilikom dizajna se automatski generiše AXI-Avalon bridge i tako su PIO blokovi `keys_0` i `leds_0` memorijski mapirani u adresni prostor HPS-a.

### 2.2 Preloader

Za pokretanje HPS sistema koristi se Preloader generisan Alterinim alatima na osnovu generisanih fajlova za opis harvera.

### 2.3 Bootloader

Uobicajeni izbor za bootloader je U-Boot. U-Boot učitava konfiguracioni fajl u `.rbf` formatu i konfigurise FPGA. Zatim učitava binarni fajl za opis hardverske pratforme (`.dtb` fajl) i binarni fajl kernela operativnog sistema (`zImage` fajl). Argumenti za podešavanje sistema se prosleđuju pri prepustanju toka izvršavanja kernelu.

## 2.4 Linuks kernel i Root File System

Izvorni kod za Linuks kernel je preuzet za Alterinog `git`-a. Root File System je generisan korišćenjem `buildroot`-a i upisan na SD karticu.

## 2.5 Drajver

Drajver je napisan kao modul kernela koji se uključuje odgovarajućom komandom. Nakon uključivanja drajver iz binarnog opisa harvdera (Device Tree Blob) učitava informacije o harvderu. Drajver pravi fajlove u `sysfs` fajl sistemu. Čitanje i upis u ove fajlove poziva odgovarajuće funkcije u drajveru koje upravljaju hardverom.

Sistemske fajlovi koje pravi drajver i njihov opis:

- `leds` - upisani broj se prikazuje na LE diodama u binarnoj predstavi
- `keys` - čitanje vraća binarnu predstavu stanja tastera
- `irq_flag` - pristup registru za flegove prekida (1 na n-tom bitu označava pristigli prekid na n-tom tasteru, upis 1 na n-ti bit čisti n-ti fleg)
- `irq_mask` - čitanje i upis u registar za maksiranje prekida (upis 1 na n-ti bit omogućava prekid na n-tom tasteru)

Ceo kod drajvera je dostupan u dodatku.

## Testiranje sistema

Binarni fajlovi za pokretanje sistema i root fajl sistem su napisani na SD karticu. HPS je povezan sa PC računaru preko UART-USB serijske veze. Otvaranjem konzole na PC računaru se pristupa sistemu.

U nastavku je dat kompleatan spisak koraka za realizovanje sistema.



### 3 Uputstvo za repliciranje rezultata

#### 3.1 Projektovanje u Quartus-u

1. U Microsoft Windows operativnom sistemu pokrenuti `DE1SoC_SystemBuilder.exe` (dostupan na [\[1\]](#))
2. Izabrati konfiguraciju kao na slici 4 i izabrati Generate

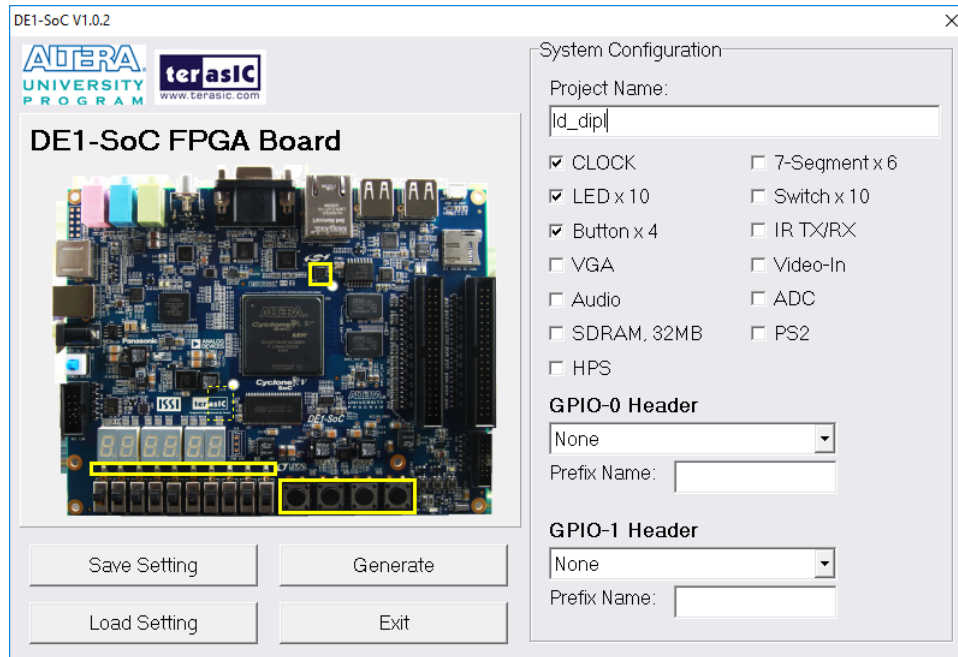


Figure 4: Podešavanja DE1.Soc Bulder-a

3. Izbristati `ld_dipl.v` fajl (ovaj fajl se ne koristi, kasnije će biti napravljen `ld_dipl.vhd` fajl za VHDL)
4. Kopirati generisane fajlove u Ubuntu u radni folder (u ovom radu je to `~/ld_dipl/hw/`)
5. Pokrenuti Quartus (Quartus Prime 18.0) Lite Edition
6. Otvoriti projekat komandom `File > Open Project...` i izabrati `~/ld_dipl/hw/ld_dipl.qpf`
7. U prozoru **Tasks** izabrati **Edit Settings**, u novom prozoru pod **Timing Analyser** ispod teksta **SDC files to include in the project** klikom na dugme `'...'` izabrati fajl `ld_dipl.sdc`
8. Pokrenuti Platform Designer klikom na `Tools > Platform Designer`
9. Iz prozora **IP Catalog** izabrati **Processors and Peripherals > Hard Processor Systems > ArriaV/Cyclone V Hard Processor System**. Ovim se otvara meni za podešavanje HPS modula
10. Pod tabom **FPGA interfaces** izvršiti sledeće izmene:
  - U opštim podešavanjima isključiti opciju `Enable MPU standby and event signals`

- U podešavanjima AXI Bridges podesiti FPGA-to-HPS interface i HPS-to-FPGA interface na unused, a Lightweight HPS-to-FPGA na 32-bit
- U podešavanjima FPGA-to-HPS SDRAM Interface izabrati f2h\_sdram0 i zatim isključiti pritiskom na dugme '-'
- U podešavanjima Interrupts uključiti opciju Enable FPGA-to-HPS interrupts

11. Pod tabom Peripheral Pins izvršiti sledeće izmene

- U podešavanjima SD/MMC Controller postaviti SDIO pin na HPS I/O Set 0 i SDIO mode na 4-bit Data
- U podešavanjima UART Controllers postaviti UART0 pin na HPS I/O Set 0 i UART mode na No Flow Control

*U ovom tabu je za potrebe nekog drugog projekta moguće uključiti ostale periferije: CAN Controller, Ethernet Media Access Controller, I2C Controller, SPI Controller, QSPI Flash Controller, NAND Flash Controller, Trace Port Interface Unit, GPIO za podešavanja pogledati []*

12. Pod tabom HPS Clocks ostaviti podešavanja na podrazumevanim vrednostima

13. Pod tabom SDRAM podesiti:

- SDRAM Protocol: DDR3
- PHY Settings:
  - Clocks:
    - \* Memory clock frequency: 400.0 MHz
    - \* PLL reference clock frequency: 25.0 MHz
  - Advanced PHY Settings:
    - \* Supply Voltage: 1.5V DDR3
- Memory Parameters:
  - Memory vendor: Other
  - Memory device speed grade: 800.0 MHz
  - Total interface width: 32
  - Number of chip select/depth expansion: 1
  - Number of clocks: 1
  - Row address width: 15
  - Column address width: 10
  - Bank-address width: 3
  - Uključiti DM pins
  - Uključiti DQS#
  - Memory Initialization Options:
    - \* Mirror Addressing: 1 per chip select: 0
    - \* Burst Length: Burst chop 4 or 8 (on the fly)
    - \* Read Burst Type: Sequential
    - \* DLL precharge power down: DLL off
    - \* Memory CAS latency setting: 11

- \* Output drive strength setting: RZQ/7
- \* ODT Rtt nominal value: RZQ/4
- \* Auto selfrefresh method: Manual
- \* Selfrefresh temperature: Normal
- \* Memory write CAS latency setting: 8
- \* Dynamic ODT (Rtt\_WR) value: RZQ/4
- Memory Timing:
  - tIS (base): 180 ps
  - tIH (base): 140 ps
  - tDS (base): 30 ps
  - tDH (base): 65 ps
  - tDQSQ: 125 ps
  - tQH: 0.38 cycles
  - tDQSCK: 255 ps
  - tDQSS: 0.25 cycles
  - tQSH: 0.4 cycles
  - tDSH: 0.2 cycles
  - tDSS: 0.2 cycles
  - tINIT: 500 us
  - tMRD: 4 cycles
  - tRAS: 35.0 ns
  - tRCD: 13.75 ns
  - tRP: 13.75 ns
  - tREFI: 7.8 us
  - tRFC: 260.0 ns
  - tWR: 15.0 ns
  - tWTR: 4 cycles
  - tFAW: 30.0 ns
  - tRRD: 7.5 ns
  - tRTP: 7.5 ns
- Board Settings:
  - Setup and Hold Derating:
    - \* Use Altera's default settings
  - Channel Signal Integrity:
    - \* Use Altera's default settings
  - Board Skews:
    - \* Maximum CK delay to DIMM/device: 0.03 ns
    - \* Maximum DQS delay to DIMM/device: 0.02 ns
    - \* Minimum delay difference between CK and DQS: 0.06 ns
    - \* Maximum delay difference between CK and DQS: 0.12 ns
    - \* Maximum skew within DQS group: 0.01 ns
    - \* Maximum skew between DQS groups: 0.06 ns
    - \* Average delay difference between DQ and DQS: 0.05 ns



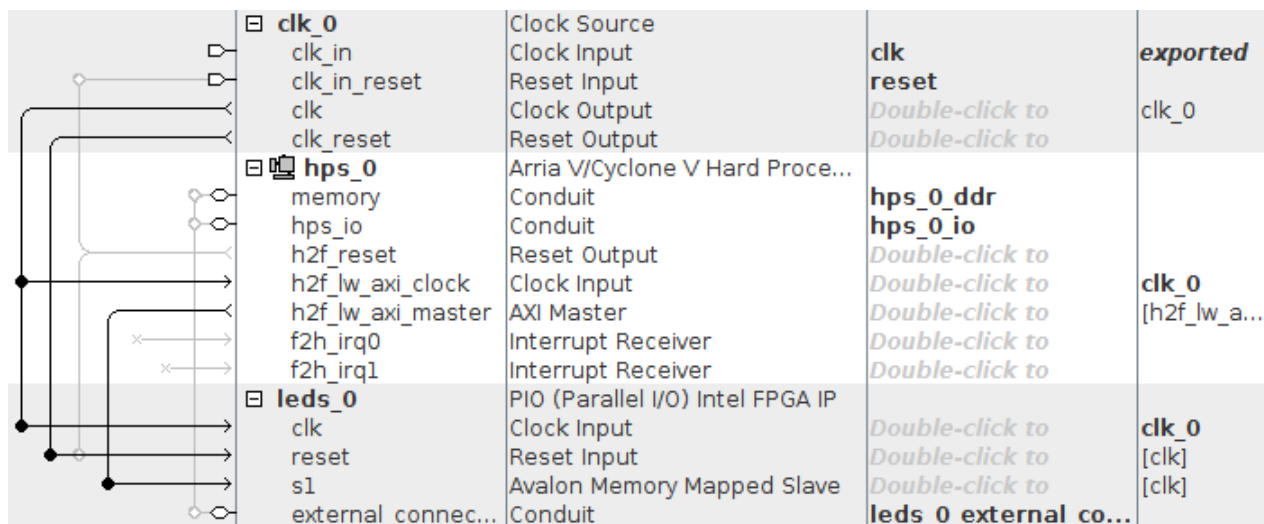


Figure 6: Povezivanje leds\_0 bloka

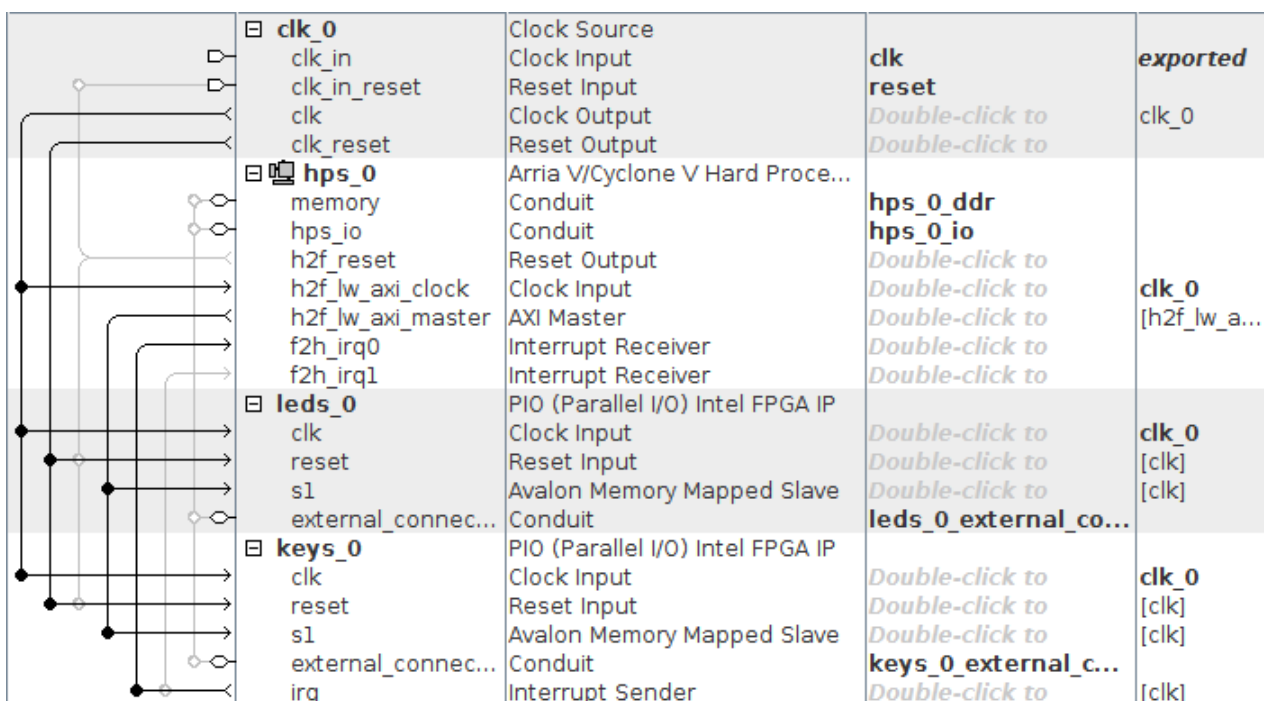


Figure 7: Povezivanje keys\_0 bloka

25. Povezati keys\_0 blok sa izvorom takta i resetom, zatim povezati Avalon Memory Mapped Slave pod imenom s1 sa hps\_0 interfejsom h2f\_lw\_axi\_master i na kraju povezati irq signal na f2h\_irq0 interfejs hps\_0 bloka, kao što je prikazano na slici 7
26. Duplim klikom u koloni Base podesiti adresu porta s1 bloka leds\_0 i porta s1 bloka keys\_0 kao na tabeli 1.
27. Sačuvati Platform Designer projekat izborom File > Save i sačuvati ga pod imenom ld\_dipl\_system.qsys
28. Trebalo bi da se pojavi obaveštenje Save System: completed successfully. Zatim odabrati iz menija Generate > Generate HDL... U novom prozoru podesiti Create HDL design files for synthesis: VHDL i isključiti opciju Create block symbol file (.bsf). Pokrenuti generisanje klikom na Generate. Proces bi trebalo da se završi bez

grešaka ali može imati upozorenja.

29. Zatvoriti Platform Designer. U prozoru Quartus-a izabrati Project > Add/Remove Files in Project... i u meniju klikom na '...' izabrati fajl ld\_dipl\_system/synthesis/ld\_dipl\_system.qip.
30. Izabrati File > New VHDL File i novi fajl nazvati ld\_dipl.vhd. Pod Project Navigator > Files desnim klikom na ld\_dipl.vhd izabrati Set as Top-Level Entity.
31. U ovom fajlu je potrebno instancirati HPS komponentu iz Platform Designer-a. Potrebno je ručno napiati ovaj fajl (primer za ovaj rad dat je u dodatku. Takođe među generisanim fajlovima nalazi se deklaracija komponente ld\_dipl\_system koja može biti od pomoći (fajl ~/ld\_dipl/hw/ld\_dipl\_system/ld\_dipl\_system.cmp).
32. Izabrati Processing > Start > Start Analysis and Synthesis
33. Izabrati Tools > Tcl Scripts...

**Važno:** Prozor koji se otvori mora da izgleda upravo kao na slici 8 (generisani fajlovi ne smeju biti duplirani). Ukoliko su fajlovi duplirani neophodno je zatvoriti Quartus i pokrenuti ponovo. Neke verzije Quartus-a imaju ovu grešku pri detekciji tcl skripti.

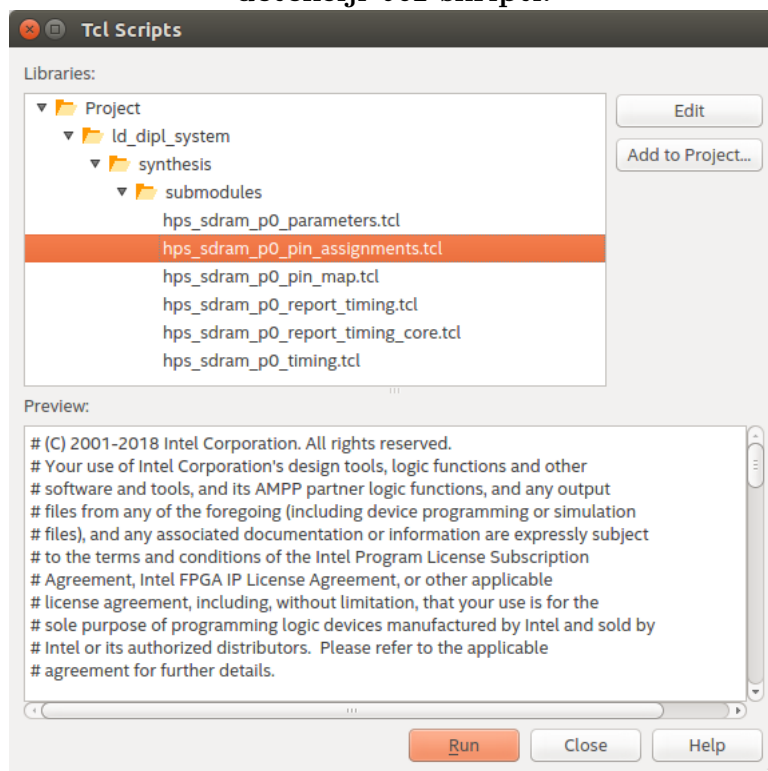


Figure 8: Ispravan izgled menija

Izabrati hps\_sdram\_p0\_pin\_assignments.tcl i kliknuti Run. Ukoliko dođe do grešaka proveriti da li je izvršen prethodni korak.

34. Pokrenuti kompajliranje projekta izborom Processing > Start Compilation

## 3.2 Podešavanja okruženja i preuzimanje kompajlera

35. Preuzeti arhivu sa Linaro toolchain-om  
`wget https://releases.linaro.org/components/toolchain/binaries/7.2-2017.11/arm-linux-gnueabihf/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabihf.tar.xz`
36. Otpakovati preuzeti toolchain:  
`tar -xf arm-linux-gnueabihf/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabihf.tar  
~/ld.dipl/sw/toolchain`
37. Dodati putanju za toolchain u promenljivu okruženja PATH.  
`export PATH=~/ld.dipl/sw/toolchain/bin:$PATH`
38. Podesiti promenljive okruženja  
`export ARCH=arm  
export CROSS_COMPILE=arm-linux-gnueabihf-`
39. Pokrenuti podešavanja Altera SOC EDS:  
`cd /intelFPGA/18.0/embedded  
source embedded_command_shell.sh`
40. Dodati putanju za Quartus alate u promenljivu okruženja PATH:  
`export PATH=~/intelFPGA_lite/18.0/quartus/bin/:$PATH`

## 3.3 Generisanje RBF fajla

41. `quartus_cpf -c -o bitstream_compression=on \  
~/ld.dipl/hw/ld.dipl.sof \  
~/ld.dipl/sdcard/fat32/socfpga.rbf`

## 3.4 Generisanje i kompajliranje Preloader-a

42. Pokrenuti Preloader generator komadnom:  
`bsp-editor`
43. Izabrati File > New HPS BSP...
44. U novom prozoru podesiti Preloader Settings Directory:  
`~/ld.dipl/hw/hps_isw_handoff/ld.dipl_system_hps_0`
45. Isključiti opciju: Use default locations i podesiti BSP target directory:  
`~/ld.dipl/sw/preloader`. Podešavanja bi trebalo da izgledaju kao na slici 9. Izabrati OK.
46. U podešavanjima pod `spl.boot` uključiti `FAT_SUPPORT` i ostala podešavanja ostaviti na podrazumevanim vrednostima. Izabrati **Generate** i po završenom generisanju zatvoriti program.
47. Izvršiti komande za kompajliranje Preloader-a  
`cd ~/ld.dipl/sw/preloader  
make`

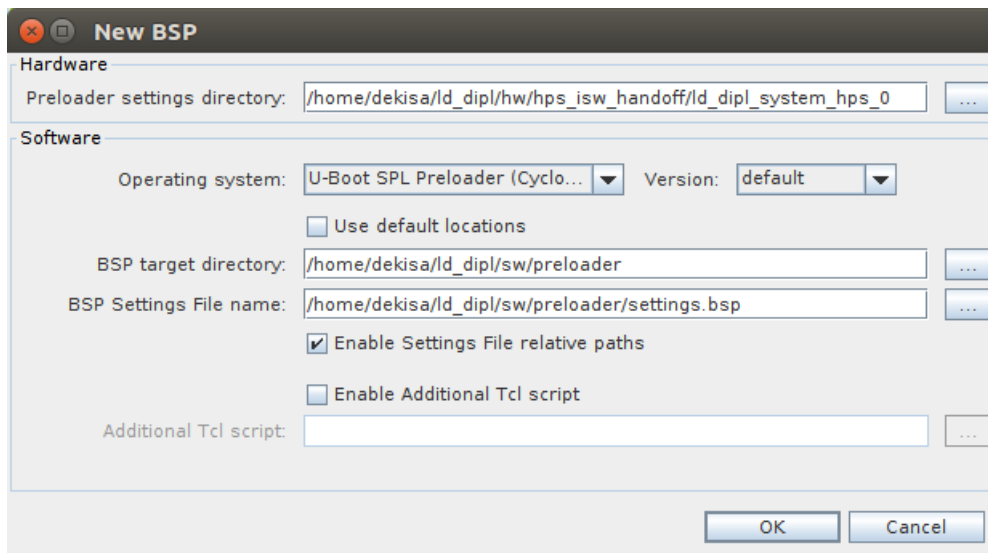


Figure 9: Podešavanja bsp-editor-a

48. Kopirati kompajlirani Preloader: `cp ~/ld_dipl/sw/preloader/preloader-mkpimage.bin ~/ld_dipl/sdcard/a2/`

### 3.5 Generisanje Device Tree

49. Uporebom Alterinog alata generisati Device Tree komandom:  
`sopc2dts -i /ld_dipl/hw/ld_dipl_system.sopcinfo`  
`-o /ld_dipl/hw/ld_dipl_generated.dts`

### 3.6 U-Boot

50. Preuzeti izvorni kod projekta U-boot komandom  
`git clone git://git.denx.de/u-boot.git ~/ld_dipl/sw/u-boot`
51. Promeniti radni direktorijum  
`~/ld_dipl/sw/u-boot/`
52. Napraviti novu granu na osnovu verzije v2018.01 komandom:  
`git checkout v2018.01 -b tmp`
53. Konfigurisati U-Boot za DE1-SoC razvojni sistem  
`make socfpga_de1_soc_defconfig`
54. Otvoriti meni za konfigurisanje U-Boot  
`make menuconfig`
55. U ovom meniju je potrebno podesiti da se pri pokretanju U-Boot pokrene naša skripta `callscript`. Izabrati opciju `Enable a default value for bootcmd` pritiskom tastera `space`.
56. U novoj opciji `bootcmd value` pritiskom tastera `enter` otvoriti novi meni i upisati tekst `run callscript`  
 Nakon ovog podešavanja bi prozor trebalo da izgleda kao na slici 10



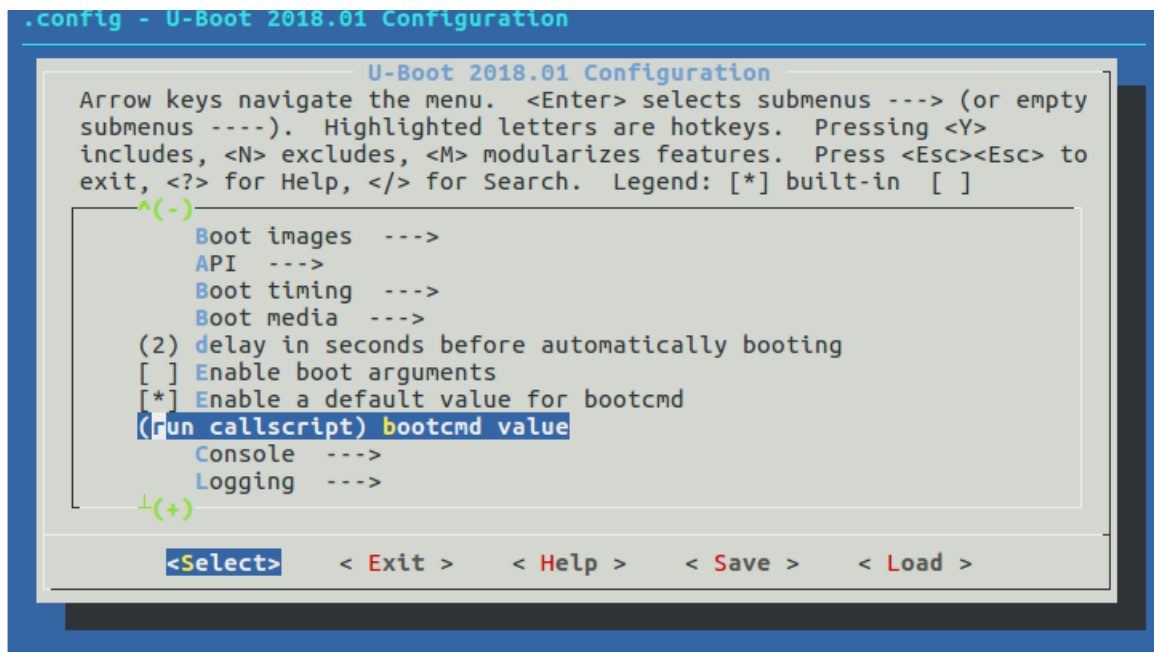


Figure 10: Konfiguracija U-Boot

57. Izaći iz konfiguracionog menija izborom opcije **exit** (strelica desno i zatim **enter**). Sačuvati podešavanja izborom **Yes**.
58. Otvoriti konfiguracioni **.h** fajl komandom:  
`vi include/configs/socfpga_de1_soc.h` U ovom fajlu je potrebno uneti izmene kao na slici 11

```
#define CONFIG_HW_WATCHDOG

/* Memory configurations */
#define PHYS_SDRAM_1_SIZE          0x40000000      /* 1GiB */

/* Booting Linux */
#define CONFIG_LOADADDR            0x01000000
#define CONFIG_SYS_LOAD_ADDR      CONFIG_LOADADDR

/* Ethernet on SoC (EMAC) */

/* The rest of the configuration is shared */
#include <configs/socfpga_common.h>

/* Custom env vars */
#define CONFIG_EXTRA_ENV_SETTINGS \
    "scriptfile=u-boot.scr" "\0" \
    "scraddr=0x20000000" "\0" \
    "callscript=fatload mmc 0:1 $scraddr $scriptfile;" \
    "source $scraddr" "\0"

#endif /* __CONFIG_TERASIC_DE1_SOC_H__ */
```

Figure 11: Izmena U-Boot konfiguracionog fajla

59. Pokrenuti kompajliranje U-Boot komandom  
`make`
60. Kopirati kompajlirani U-Boot  
`cp ~/ld_dipl/sw/u-boot/u-boot.img ~/ld_dipl/sdcard/fat32/u-boot.img`

61. Otvoriti novi tekstualni fajl:  
`vi u-boot.script`  
 U ovom fajlu je potrebno napisati skriptu za U-Boot. Ranije u radu je dato objašnjenje skripte, dok se ceo kod nalazi u dodatku rada.
62. Konvertovati napisanu skriptu u odgovarajući format komandom:  
`mkimage -A arm -O linux -T script -a 0 -e 0 -n Boot_script -d u-boot.script u-boot.scr`
63. Kopirati kompajlirani U-Boot  
`cp ~/ld_dipl/sw/u-boot/u-boot.scr ~/ld_dipl/sdcard/fat32/u-boot.scr`

### 3.7 Linuks kernel

64. Preuzeti izvorni kod Linuks kernela:  
`git clone git://github.com/torvalds/linux ~/ld_dipl/sw/linux`
65. Promeniti radni direktorijum  
`cd ~/ld_dipl/sw/linux`
66. Napraviti novu granu na osnovu verzije 4.6-rc2 komandom:  
`git checkout 4.6-rc2 -b tmp`
67. Konfigurisati linuks za Cyclone V arhitekturu  
`make socfpga_defconfig`
68. Pokrenuti kompajliranje linuksa  
`make zImage`
69. Kopirati kompajlirani kernel u pripremni folder  
`cp arch/arm/boot/zImage ~/ld_dipl/sdcard/fat32/zImage`

### Ručno pisanje Device Tree

70. Početi od Device Tree fajla  
`cp arch/arm/boot/dts/socfpga_cyclone5_socdk.dts arch/arm/boot/dts/socfpga_cyclone5_ld_dipl.dts`
71. Isključiti CAN podešavanjem parametra `status = 'disabled'`. Ovaj korak je neophodan za uspešno pokretanje sistema jer DE1-SoC nema CAN. Pogledati sliku ??
72. Iz Device Tree fajla generisanog Alterinim alatom izdvojiti samo najbitnije informacije i upisati u novi fajl. Pogledati sliku ??
73. Pokrenuti kompajliranje Device Tree fajla  
`make socfpga_cyclone5_ld_dipl.dtb`
74. Kopirati kompajlirani Device Tree Blob u pripremni folder  
`cp ~/ld_dipl/sw/linux/arch/arm/boot/dts/socfpga_cyclone5_ld_dipl.dtb ~/ld_dipl/sdcard/fat32/socfpga.dtb`

```

        regulator-max-microvolt = <3300000>;
    };
};
/ {
    soc {
        lddipl0: lddipl@0xff200000 {
            compatible = "ld,dipl";
            reg = <0xff200000 0x00000020>;
            interrupts = <0 40 1>;
        };
    };
};

&can0 {
    status = "disabled";
};

&gmac1 {
    status = "okay";
    phy-mode = "rgmii";
    rxd0-skew-ps = <0>;
};

```

Figure 12: Izmena Device Tree fajla

### 3.8 Kompajliranje drajvera

75. Promeniti radni folder  
`cd ~/ld_dipl/sw/device_driver`  
 U ovom folderu se nalazi izvorni fajl drajvera `ld_dipl.c` i odgovarajući `Makefile`
76. Podesiti promenljivu koja ukazuje na izvorni kod linuks kernela  
`export OUT=~/ld_dipl/sw/linux`
77. Pokrenuti kompajliranje komandom  
`make`
78. Kopirati kompajlirani drajver u pripremni direktorijum  
`cp ~/ld_dipl/sw/device_driver/ld_dipl.ko ~/ld_dipl/sdcard/ld_dipl.ko`

### 3.9 Generisanje Root File System

79. Preuziti Buildroot `git clone git://git.busybox.net/buildroot ~/ld_dipl/sw/buildroot`
80. Promeniti radni folder  
`cd ~/ld_dipl/sw/buildroot`
81. Napraviti novu granu na osnovu verzije 2017.08 komandom: `git checkout 2017.08 -b tmp`
82. Pokrenuti konfiguracioni meni komandom  
`make menuconfig`
83. U konfoguracionim meniju izvršiti sledeće izmene
  - (a) Target options —
  - (b) Target Architecture (ARM(little endian))

- (c) Target Architecture Variant (cortex-A9)
- (d) Target ABI (EABIhf)
- (e) Floating point strategy (VFPv3)

84. Toolchain —

85. Toolchain type (External toolchain)

86. Toolchain (Linaro ARM 2017.11)

87. Toolchain origin (Pre-installed toolchain) —

`~/ld_dipl/sw/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi`

Kernel —

Linux Kernel

Pokrenuti izvršavanje komandom

`make`

Kopirati fajl sistem u pripremni direktorijum

`cp ~/ld_dipl/sw/buildroot/output/images/rootfs.tar ~/ld_dipl/sdcard/rootfs.tar`

### 3.10 Priprema SD kartice

Ubaciti SD karticu u računar. Zaključiti pod kojim imenom se kartica pojavila u sistemskom folderu `/dev` (za primer je uzeto `/dev/sdx` )

Formatirati SD karticu komandom

`sudo dd if=/dev/zero of=/dev/sdx bs=512 count=1`

Particionisati SD karticu pokretanjem interaktivnog programa `fdisk`

`sudo fdisk /dev/sdx`

u kom treba uneti sledeće komande

`n p 3 <default> 4095 t a2`

`n p 1 <default> +32M t 1 b`

`n p 2 <default> +512M t 2 83`

`w`

Napraviti prazne fajl sisteme na odgovarajućim particijama

`sudo mkfs.vfat /dev/sdx1`

`sudo mkfs.ext3 -F /dev/sdx2`

Promeniti radni folder

`cd ~/ld_dipl/sdcard`

Napraviti foldere na koje će se mount-ovati SD kartica

`mkdir mountfat32`

`mkdir mountext3`

Mount-ovati odgovarajuće particije

`sudo mount /dev/sdx1 ~/ld_dipl/sdcard/mountfat32`

`sudo mount /dev/sdx2 ~/ld_dipl/sdcard/mountext3`

Napisati Preloader na odgovarajuću particiju

```
sudo dd if=~/.ld_dipl/sdcard/a2/preloader-mkpimage.bin of=/dev/sdx3 bs=64K seek=0
```

Kopirati binarne fajlove za pokretanje sistema na odgovarajuću particiju

```
sudo cp ~/.ld_dipl/sdcard/fat32/* ~/.ld_dipl/sdcard/mountfat32
```

Otpakovati fajlsistem na SD karticu

```
tar -xf rootfs.tar mounttext3
```

Kopirati modul drajvera na SD karticu

```
cp ld_dipl.ko mounttext3/root/home
```

Uveriti se da su svi fajlovi uspešno kopirani komandom

```
sync
```

Unmount-ovati sve particije

```
umount ~/.ld_dipl/sdcard/mountfat32 umount ~/.ld_dipl/sdcard/mounttext3
```

### 3.11 Testiranje sistema

Proverti da li su MSEL podešeni za konfigurisanje FPGA sa HSP (ima negde lepa slika)

Pruključiti DE1-SoC na napajanje

Povezati Mini USB kablom PC računar i port na DE1-SoC ploči koji je obležen sa UART

Ubaciti SD karticu u odgovarajući slot

Na PC računaru pokrenuti i podseiti `minicom`

```
sudo minicom -s
```

U podešavanjima `Serial port setup` uneti izmene kao na slici 13 (može se razlikovati ime `Setial device`)

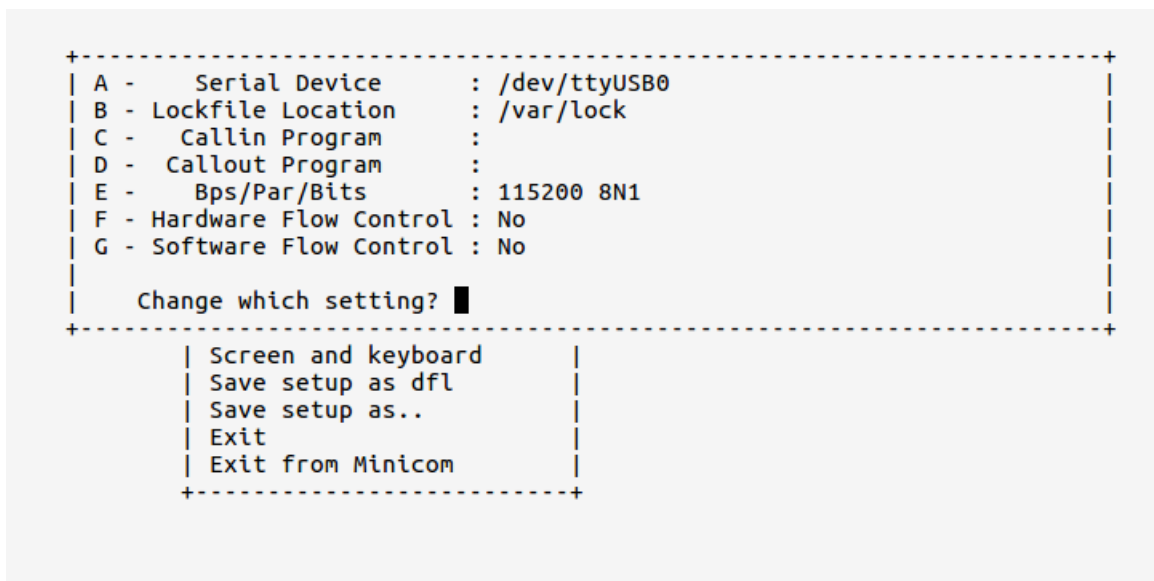


Figure 13: Podešavanje `minicom-a`

Pokrenuti DE1-SoC pritiskom crvenog tastera

Nakon pokretanja sistema potrebno je ulogovati se kao korisnik sa:q

Username:root

Password:root

Učitati modul drajvera u linux kernel komandom:

```
insmod /root/home/ld_dipl.ko
```

Uveriti se da su se pojavili odgovarajući sistemski fajlovi:

```
ls /sys/bus/platform/...
```

Uključiti LE diode

slika

Uključiti prekida

slika

## 4 Zaključak

Izrada ovog rada je motivisana željom da se upoznaju konkretni alati i postupak projektovanja sa SoCFPGA sistemima. Izveštaj o radu je napisan tako da pruži kratak pregled bitnih pojmova i detaljna uputsva za reprodukciju rezultata sa osvrtom na usputne probleme. Stečeno znanje može olakšati slušanje predmeta na master studijama, ili biti osnova za rešavanje konkretnog problema (npr. ubrzavanje algoritama kompresije i obrade slike), a autoru je olakšalo snalaženje na novom radnom mestu. Autor izražava veliku zahvalnost profesoru Lazaru Saranovcu i asistentu Strahinji Jankovicu za podršku i savete prilikom izrade diplomskog rada.

## 5 Dodatak

### 5.1 Izvorni kod drajvera

```
/* ld_dipl.c */

#include <linux/init.h>
#include <linux/module.h>
#include <linux/io.h>
#include <linux/interrupt.h>
#include <linux/uaccess.h>
#include <linux/platform_device.h>
#include <linux/of.h>

#define DEVICE_FILE_NAME      "ld_dipl"
#define DRIVER_NAME           "lddipldrv"

#define LED_OFFSET             0
#define KEYS_OFFSET            16
#define DATA_OFFSET           0
#define INTERRUPT_MASK_OFFSET  8
#define EDGE_CAPTURE_OFFSET    12

/* prototypes */
static struct platform_driver ld_dipl_driver;

/* globals */
static int g_ld_dipl_driver_irq;
static void *g_ld_dipl_driver_base_addr;
static int g_driver_mem_base_addr;
static int g_driver_mem_size;

static ssize_t irq_mask_show(struct device_driver *driver, char *buf)
{
    uint32_t value;
    value = ioread32(g_ld_dipl_driver_base_addr + KEYS_OFFSET
                    + INTERRUPT_MASK_OFFSET);
    return scnprintf(buf, PAGE_SIZE, "mask=%u\n", value);
}

static ssize_t irq_mask_store(struct device_driver *driver, const char *
    buf,
                                size_t count)
{
    uint32_t value;
    sscanf(buf, "%u", &value);
    iowrite32(value, g_ld_dipl_driver_base_addr + KEYS_OFFSET
              + INTERRUPT_MASK_OFFSET);
    return count;
}

DRIVER_ATTR(irq_mask, (S_IWUSR | S_IRUSR), irq_mask_show, irq_mask_store
);

static ssize_t irq_flag_show(struct device_driver *driver, char *buf)
{
    uint32_t value;
    value = ioread32(g_ld_dipl_driver_base_addr + KEYS_OFFSET
                    + EDGE_CAPTURE_OFFSET);
    return scnprintf(buf, PAGE_SIZE, "flags=%u\n", value);
}
```



```

}

static ssize_t irq_flag_store(struct device_driver *driver, const char *
    buf,
                                size_t count)
{
    uint32_t value;
    sscanf(buf, "%u", &value);
    iowrite32(value, g_ld_dipl_driver_base_addr + KEYS_OFFSET
        + EDGE_CAPTURE_OFFSET);
    return count;
}

DRIVER_ATTR(irq_flag, (S_IWUSR | S_IRUSR), irq_flag_show, irq_flag_store
    );

static ssize_t keys_show(struct device_driver *driver, char *buf)
{
    uint32_t value;
    value = ioread32(g_ld_dipl_driver_base_addr + KEYS_OFFSET
        + DATA_OFFSET);
    return scnprintf(buf, PAGE_SIZE, "keys=%u\n", value);
}

DRIVER_ATTR(keys, (S_IRUSR), keys_show, NULL);

static ssize_t leds_store(struct device_driver *driver, const char *buf,
    size_t count)
{
    uint32_t value;
    sscanf(buf, "%u", &value);
    iowrite32(value, g_ld_dipl_driver_base_addr + LED_OFFSET +
        DATA_OFFSET);
    return count;
}

DRIVER_ATTR(leds, (S_IWUSR), NULL, leds_store);

static irqreturn_t ld_dipl_isr(int irq, void *data)
{
    uint32_t value;

    value = ioread32(g_ld_dipl_driver_base_addr + KEYS_OFFSET
        + EDGE_CAPTURE_OFFSET);

    pr_info("irq received: %u\n", value);

    iowrite32(value, g_ld_dipl_driver_base_addr + KEYS_OFFSET
        + EDGE_CAPTURE_OFFSET);

    return IRQ_HANDLED;
}

static struct of_device_id ld_dipl_of_match[] = {
    {
        .compatible = "ld,dipl"
    },
    { /* end of table */ }
};

```

```

MODULE_DEVICE_TABLE(of, ld_dipl_of_match);

static int ld_dipl_probe(struct platform_device *pdev)
{
    int ret;
    struct resource *res;
    struct resource *driver_mem_region;

    pr_info("probe_enter\n");

    ret = -EINVAL;

    /* get memory resource */
    res = platform_get_resource(pdev, IORESOURCE_MEM, 0);
    if (res == NULL) {
        pr_err("IORESOURCE_MEM, 0 does not exist\n");
        return ret;
    }

    g_driver_mem_base_addr = res->start;
    g_driver_mem_size = resource_size(res);

    /* reserve our memory region */
    driver_mem_region = request_mem_region(g_driver_mem_base_addr,
                                           g_driver_mem_size,
                                           "demo_driver_hw_region");

    if (driver_mem_region == NULL) {
        pr_err("request_mem_region failed\n");
        return ret;
    }

    /* ioremap memory region */
    g_ld_dipl_driver_base_addr = ioremap(g_driver_mem_base_addr,
                                         g_driver_mem_size);
    if (g_ld_dipl_driver_base_addr == NULL) {
        pr_err("ioremap failed\n");
        goto bad_exit_release_mem_region;
    }

    /* get interrupt resource */
    g_ld_dipl_driver_irq = platform_get_irq(pdev, 0);
    if (g_ld_dipl_driver_irq < 0) {
        pr_err("invalid IRQ\n");
        goto bad_exit_iounmap;
    }

    pr_info("interrupt is: %d\n", g_ld_dipl_driver_irq);

    /* register interrupt handler */
    ret = request_irq(g_ld_dipl_driver_irq,
                     ld_dipl_isr,
                     0,
                     ld_dipl_driver.driver.name,
                     &ld_dipl_driver);

    if (ret < 0) {
        pr_err("unable to request IRQ\n");
        goto bad_exit_iounmap;
    }
}

```

```

/* create the sysfs entries */
ret = driver_create_file(&ld_dipl_driver.driver,
                        &driver_attr_irq_mask);

if (ret != 0) {
    pr_err("failed to create irq_mask sysfs entry");
    goto bad_exit_remove_irq_mask;
}

ret = driver_create_file(&ld_dipl_driver.driver,
                        &driver_attr_keys);

if (ret != 0) {
    pr_err("failed to create keys sysfs entry");
    goto bad_exit_remove_keys;
}

ret = driver_create_file(&ld_dipl_driver.driver,
                        &driver_attr_leds);

if (ret != 0) {
    pr_err("failed to create leds sysfs entry");
    goto bad_exit_remove_leds;
}

ret = driver_create_file(&ld_dipl_driver.driver,
                        &driver_attr_irq_flag);

if (ret != 0) {
    pr_err("failed to create irq_flag sysfs entry");
    goto bad_exit_remove_irq_flag;
}

pr_info("probe exit success\n");
return 0;

bad_exit_remove_irq_flag:
    driver_remove_file(&ld_dipl_driver.driver,
                      &driver_attr_irq_flag);
bad_exit_remove_leds:
    driver_remove_file(&ld_dipl_driver.driver,
                      &driver_attr_leds);
bad_exit_remove_keys:
    driver_remove_file(&ld_dipl_driver.driver,
                      &driver_attr_keys);
bad_exit_remove_irq_mask:
    driver_remove_file(&ld_dipl_driver.driver,
                      &driver_attr_irq_mask);
bad_exit_freeirq:
    free_irq(g_ld_dipl_driver_irq, &ld_dipl_driver);
bad_exit_iounmap:
    iounmap(g_ld_dipl_driver_base_addr);
bad_exit_release_mem_region:
    release_mem_region(g_driver_mem_base_addr, g_driver_mem_size);

pr_info("probe exit fail\n");
return ret;
}

static int ld_dipl_remove(struct platform_device *pdev)
{
    pr_info("remove enter\n");
    driver_remove_file(&ld_dipl_driver.driver,
                      &driver_attr_irq_flag);
    driver_remove_file(&ld_dipl_driver.driver,

```

```

        &driver_attr_leds);
driver_remove_file(&ld_dipl_driver.driver,
        &driver_attr_keys);
driver_remove_file(&ld_dipl_driver.driver,
        &driver_attr_irq_mask);
driver_remove_file(&ld_dipl_driver.driver,
        &driver_attr_irq_flag);
driver_remove_file(&ld_dipl_driver.driver,
        &driver_attr_leds);
driver_remove_file(&ld_dipl_driver.driver,
        &driver_attr_keys);
driver_remove_file(&ld_dipl_driver.driver,
        &driver_attr_irq_mask);
free_irq(g_ld_dipl_driver_irq, &ld_dipl_driver);

iounmap(g_ld_dipl_driver_base_addr);

release_mem_region(g_driver_mem_base_addr, g_driver_mem_size);

pr_info("remove_exit\n");
return 0;
}

static struct platform_driver ld_dipl_driver = {
    .probe = ld_dipl_probe,
    .remove = ld_dipl_remove,
    .driver = {
        .name = DRIVER_NAME,
        .of_match_table = ld_dipl_of_match,
    },
};

static int __init ld_dipl_init(void)
{
    int ret;

    pr_info("init_enter\n");

    ret = platform_driver_register(&ld_dipl_driver);
    if (ret != 0)
        pr_err("platform_driver_register returned %d\n", ret);
    pr_info("init_exit\n");

    return ret;
}

static void __exit ld_dipl_exit(void)
{
    pr_info("ld_dipl_exit_enter\n");
    platform_driver_unregister(&ld_dipl_driver);
    pr_info("ld_dipl_exit_exit\n");
}

module_init(ld_dipl_init);
module_exit(ld_dipl_exit);

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("ETF_Diploma_Thesis_FPGA_Driver_and_Device");
MODULE_AUTHOR("Dejan Lukic");

```

## 5.2 Skripta za U-Boot

```
echo --- Reseting env variables... ---

# reset environment variables to default
env default -a

echo --- Setting env variables... ---

# Set kernel image
setenv bootimage zImage;

# adress to wich the device tree will be loaded
setenv fdtaddr 0x00000100

# set device tree image
setenv fdtimage socfpga.dtb;

#set kernel boot arguments, boot kernel
setenv mmcboot 'setenv bootargs mem=1024M console=ttyS0,115200 root=${
    mmcroot}_rw_rootwait;_bootz_${loadaddr}_-${fdtaddr}';

#load linux kernel image and device tree to memory
setenv mmcload 'mmc_rescan;_${mmcloadcmd}_mmc_0:${mmcloadpart}_${
    loadaddr}_${bootimage};_${mmcloadcmd}_mmc_0:${mmcloadpart}_${fdtaddr
    }_${fdtimage}'

#command to be executed to read from sdcard
setenv mmcload fatload

# sdcard fat32 partition number
setenv mmcloadpart 1

# sdcard ext3 identifier
setnev mmcroot /dev/mmcblk0p2

#standard io
setenv stderr serial
setenv stdin serial
setenv stdout serial

#save environment to sdcard
saveenv

echo --- Programming FPGA...---

#load rbf from fat partition to memory
fatload mmc 0:1 ${fpgadata} socfpga.rbf;

# program fpga
fpga load 0 ${fpgadata} ${filesize}

# enable h2f, f2h, lw_h2f
bridge enable;

echo --- Booring Linux...---

#load kernel and device tree to memory
run mmcload;
```

```
# set bootargs, boot kernel  
run mmcboot;
```