Универзитет у Крагујевцу
Факултет инжењерских наука


Назив студијског програма: Машинско инжењерство
Ниво студија: Мастер академске студије
Модул: Примењена механика и аутоматско управљање
Предмет: Индустријски рачунарски системи
Број индекса: 328/2014

Никола Д. Јовић

Пројектовање WEB лабораторије за образовне потребе програмирања и употребе FPGA уређаја

Мастер рад

Комисија за преглед и одбрану:

1. Проф. др Милан Матијевић, ментор
2.
3.

Датум одбране:

Оцена:

Захвалница

Желим да се захвалим доц. др Владимиру Цвјетковићу и проф. др Милану Матијевићу који су ме укључили у реализацију SCOPES пројекта из кога је и проистекао овај рад.

Велику захвалност дугујем проф. др Владимиру М. Стојановићу и Ненаду Бабајићу, које нисам упознао, али сам користио ишоде њиховог залагања око трансфера предмета МИТ 6.111 на Факултет инжењерских наука.

Захваљујем се свим члановима Комисије за оцену и усмену одбрану мог дипломског рада на пажљивом читању и датим сугестијама.

Највећу захвалност дугујем својој породици на пруженој љубави, разумевању и подршци током мог школовања.

Аутор

У оквиру овог рада кандидат треба да:

Пројектује **WEB** Лабораторију за образовне потребе програмирања и употребе *FPGA*. У конкретном случају то подразумева реализацију WEБ портала који ће кориснику омогућити приступ лабораторијској опреми путем Интернета ради савладавања предмета *MIT 6.111 Introductory Digital Systems Laboratory* (*http://web.mit.edu/6.111*). Наставна методологија предмета (дата кроз 6.111 - *MIT OpenCourseWare* стране и *http://web.mit.edu/6.111*) подразумева израду 8 домаћих задатака и 5 лабораторијских вежби програмирања *FPGA* (које је у конкретном случају могуће реализивати на *Digilent Nexys2* платформи). Рад ће понудити решење ових задатака, и пропратне садржаје, са идејом да се може надаље користити у наставне сврхе на српском говорном подручју. Такође, **WEB** стране које ће омогућити програмирање и тестирање програма на *FPGA* путем Интернета, обухватиће типичне сервисе **WEB** Лабораторија и омогући ће проширивост на већи број експеримената и/или модуле експеримената из других области.

Препоручена литература:

[1] Интернет портал предмета *MIT 6.111 Introductory Digital Systems*, *http://web.mit.edu/6.111/*

[2] MIT OpenCourseWare, *ocw.mit.edu*, портал предмета *MIT 6.111 Introductory Digital Systems Laboratory*

[3] Владимир М. Стојановић, *Увод у пројектовање и контролу интегрисаних кола за комуникације, сензоре и актуаторе, http://moodle.mfkg.rs/course/view.php?id=250*, *Moodle* портал предмета докторских студија Факултета инжењеских наука Универзитета у Крагујевцу , школска 2010/2011 година

[4] Ненад Бабајић, Решења са образложењем домаћих задатака и лабораторијских вежби предмета *Увод у пројектовање и контролу интегрисаних кола за комуникације, сензоре и актуаторе, http://moodle.mfkg.rs/course/view.php?id=250*, Факултет инжењерских наука у Крагујевцу, 2011.

Садржај

Списак слика

Списак табела

Резиме

У овом мастер раду је представљена имплементација лабораторијског модела и **WEB** лабораторије за програмирање и контролу *FPGA* симулатора, са намером коришћења као наставно средство за курс *MIT 6.111 Introductory Digital Systems Laboratory* (*http://web.mit.edu/6.111*). Циљеви које треба остварити у оквиру овог рада су: Омогућавање лабораторијске наставе путем Интернета у области програмирања и примене *FPGA*, развој лабораторијских материјала на српском језику за предмет *MIT 6.111 Introductory Digital Systems Laboratory*, да се омогући примена наставне методологије предмета *MIT 6.111 Introductory Digital Systems Laboratory* у пуном капацитету (да се превазиђе проблем недоступности хардверске и софтверске подршке за обуку студената и других корисника) и да се допринесе општем софтверском решењу **WEB** лабораторије која се развија на Универзитету у Крагујевцу у оквиру *SCOPES* пројекта IЗ74З0_160454 / 1: "Enabling Web-based Remote Laboratory Community and Infrastructure" of Swiss National Science Foundation Кроз рад ће бити описан хардвер и софтвер коришћен за прављење овог модела, интеграција лабораторијског модела у направљено **WEB** окружење, интеграција модела у постојеће глобално окружење за **WEB** лабораторије (*Go Lab*) и дати решени примери вежби на **WEB** окружењу.

Кључне речи: Програмирање FPGA, GO LAB, WEB лабораторија, MIT 6.111

Abstract

This master's thesis will present implementation of laboratory model and **WEB** based laboratory designed for programming and control of an FPGA device, which will serve as teaching material for the *MIT 6.111 Introductory Digital Systems Laboratory* (*http://web.mit.edu/6.111*) course. Aims that must be achieved through this work are: Enabling laboratory lectures through Internet, development of laboratory material for *MIT 6.111 Introductory Digital Systems Laboratory* in Serbian language, enabling teaching methodology of *MIT 6.111 Introductory Digital Systems Laboratory* course in full capacity, and contribution to the overall software solution for Web based laboratory in University in Kragujevac, developed within the scope of the *SCOPES* project IЗ74З0_160454 / 1: "Enabling Web-based Remote Laboratory Community and Infrastructure" of Swiss National Science Foundation. Through this work, hardware and software of a laboratory will be described, as well as integration of laboratory model in two **WEB** laboratory services, namely *Go Lab* service and Laboratory Aggregator service made within the scope of this master's thesis. Solved laboratory exercises are also included in this work.

Keywords: FPGA programming, GO LAB, Web laboratory, MIT 6.111

1 Увод

WEB лабораторије представљају физичку опрему и пропратну апаратуру која може даљински да се контролише путем Интернета посредством Интернет прегледача, који омогућава промену конфигурације опреме и преглед релевантних информација у реалном времену са било ког места у било које време. Историјски развој **WEB** лабораторија је описан у прегледном раду [1], док су у [2] анализирани неки од позитивних утицаја **WEB** лабораторија. Истиче се чињеница да земље у развоју коришћењем **WEB** лабораторија могу да повећају образовне капацитете без губитка квалитета наставе [3]. Други потенцијални бенефити су и могућност наставника да демонстрира реалне физичке експерименте *ex cathedra* приступом [4], већа доступност лабораторијских капацитета студентима у односу на дефинисане термине непосредног рада студената на опреми за спровођење лабораторијских вежби, доступност опреме студентима са посебним потребама, као и наставницима и студентима матичних институција које не поседују одговарајућу опрему, а значајан бенефит може бити и повећана безбедност корисника лабораторије приликом обављања експеримената (високи напон, температура, опасне хемикалије, итд.). Посебно се истиче повољан однос цене коришћења опреме наспрам броја корисника лабораторије, као и опремљеност лабораторије наставним материјалима који подржавају проблемски орјентисано учење [1, 5]. Особине које би **WEB** лабораторија требало да поседује су:

- Приступност 24 часа дневно, седам дана у недељи
- Клијентска страна би требало да буде напишена коришћењем *HTML 5* стандарда, односно да се не користе нестандардни софтвери (*Adobe Flash, Java Applet*)
- Минимална администрација (без резервација, приступ добија клијент који први приступи)
- Максимална безбедност (сигурност хардвера на првом месту, немогућност хаковања)
- Систем би требало да се врати на првобитно стање после завршетка експеримента
- Експеримент би требало да траје пар минута

Циљеви овог дипломског рада су следећи:

- Да се омогући лабораторијска настава путем Интернета у области програмирања и примене *FPGA*
- Да се допринесе развоју наставних лабораторијских материјала на српском језику предмета *MIT 6.111 Introductory Digital Systems Laboratory* који је намењен оспособљавању студената за примену *FPGA* развојне платформе
- Да се омогући примена наставне методологије предмета *MIT 6.111 Introductory Digital Systems Laboratory* у пуном капацитету (да се превазиђе проблем недоступности хардверске и софтверске подршке за обуку студената и других корисника)
- Да се допринесе општем софтверском решењу **WEB** лабораторије која се развија на Универзитету у Крагујевцу у оквиру *SCOPES* пројекта IZ74Z0_160454 / 1: "Enabling Web-based Remote Laboratory Community and Infrastructure" of Swiss National Science Foundation

Прва верзија развијеног софтверског решења **WEB** лабораторије (у контексту поменутог *SCOPES* пројекта) је тестирана на Факултету инжењерских наука Универзитета у Крагујевцу у току зимског семестра школске 2015/16, у оквиру предмета Мерење и управљање. Омогућено је да 190 студената уради 4 лабораторијске вежбе путем Интернета, при чему је физички било расположиво 2-4 лабораторијска модела у истом временском тренутку [6].

Примена WEБ лабораторије помогла је у решавању следећих проблема:

- Превазилажење ограничења у ресурсима (у обученом наставном кадру, опреми и простору)
- Држане су аудиторне демонстрационе вежбе у складу са постојећим ограничењима, а студентима је омогућен приступ лабораторији путем Интернета у флексибилном термину који њима највише индивидуално одговара и када су спремни.
- Студенти су имали могућност и обавезу даураде индивидуалне лабораторијске вежбе упркос огрничењу од 20 лабораторијских места по студентској групи за дати ниво студија.
- Лабораторијска настава је утицала на исходе знања студената.

Развијено софтверско решење **WEB** лабораторије поседује агрегатор, који омогућава додавање произвољног броја нових лабораторијски модела у постојећи садржај **WEB** лабораторије. Сервер **WEB** лабораторије се тренутно налази у Центру за примењену аутоматику Факултета инжењерских наука Универзитета у Крагујевцу. Почетна **WEB** страна је на адреси [7].

Овај рад описује софтверску и хардверску инфраструктуру дела WEБ лабораторије у функцији лабораторијске подршке наставним садржајима предмета **MIT 6.111** (који на странама *http://web.mit.edu/6.111/* садржи десетогодишњу архиву студентских пројеката у програмирању и употреби *FPGA*). Приступ **WEB** лабораторији решава проблем недостатка опреме у праћењу предмета **MIT 6.111**, и на српском језику упућује студента како да се посвети савладавању градива (којим редоследом које домаће задатке и лабораторијске вежбе да решава, и ако не може самостално да дође до решења, објашњава решавање истих). **WEB** лабораторијски приступ предмету **MIT 6.111** је већ спроведен у [8], тако да идеја није нова. Међутим, недостатаци приступа примењеног у [8], сада, после 10 година, су коришћење скупе апаратуре за мерење сигнала и управљање *FPGA* уређајем, и серверска страна писана у **.NET** језицима, који раде само у *Windows* окружењу. Један од задатака овог рада јесте превазилажење горе наведених проблема у смислу прављења портабилне серверске стране која може да ради на широком дијапазону оперативних система и хардверских архитектура.

У овом раду су наведени недостаци приступа у [8] превазиђени, тј. направљана је портабилна серверске стране која може да ради у широком дијапауону оперативних система и хардверских архитектура. Лабораторијски модели WEБ лабораторије за подршку реализацији образовног садржаја у **MIT 6.111**, у овом раду, користе *Digilent Nexys2 FPGA* развојну платформу која поседује *Xilinx Spartan 3E-500* уређај. Избор за *Digilent Nexys2 FPGA* је направио проф. др Владимир М. Стојановић који је у својству професора *MIT*-а на Универзитету у Крагујевцу предавао овај предмет школске 2010/11 на нивоу докторских студија. Приступачна цена и мноштво улаза и излаза су очигледно били фактори који су утицали на наведени избор.

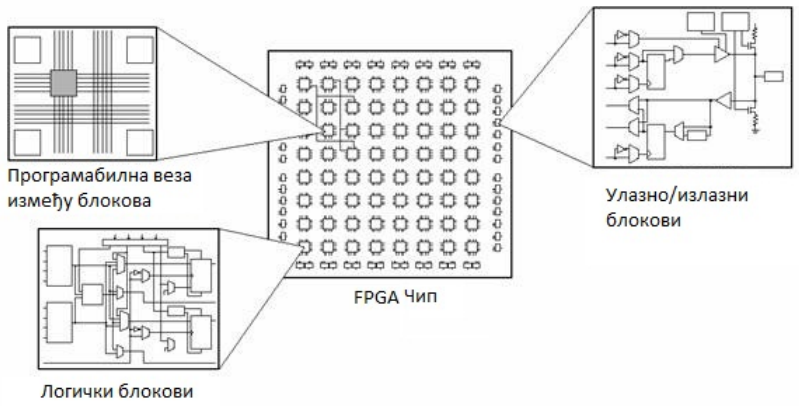
У другом поглављу овог рада описана је структура и примена *FPGA* чипа. Треће поглавље се фокусира на пројектовање **WEB** лабораторије и лабораторијског модела за програмирање и примену *FPGA* развојне платформе, како у хардверском, тако и

у софтверском смислу. У четвртом поглављу, описана је интеграција направљеног лабораторијског модела у агрегатор **WEB** лабораторија на Факултету инжењерских наука Универзитета у Крагујевцу, и у сервис *Go Lab*. У петом поглављу је дат преглед курса који треба овај мастер рад да покрије, док је шесто поглавље написано са сврхом да служи као својеврсни увод у језик за описивање хардвера *Verilog*.

2 *FPGA* чип

FPGA (*Field programmable gate array, енгл. Низ логичких кола са могућношћу програмирања на терену*) чип представља интегрисано коло које је могуће програмирати после процеса производње. За разлику од микропроцесора и микроконтролера који имају фиксну архитектуру способну за извршавање софтверског алгоритма који се налази у меморији, *FPGA* чипови немају фиксну архитектуру. Уместо тога, *FPGA* чипови поседују одређен број логичких кола организованих у блокове, где се два или више блокова могу произвољно повезивати један са другим. Оваква архитектура пружа високу флексибилност при пројектовању дигиталних кола, јер омогућава прављење високо паралелизованих кола, која могу бити вишеструко бржа у односу на конвенционалне процесоре у зависности од задатка који је потребно извршити. *FPGA* чип је сачињен од три типа основних компоненти, и то (слика 2.1):

- Програмабилни логички блок који имплементира основне логичке функције
- Програмабилне прекидаче који повезују различите логичке блокове
- Улазно-излазне блокове за комуникацију са спољним светом



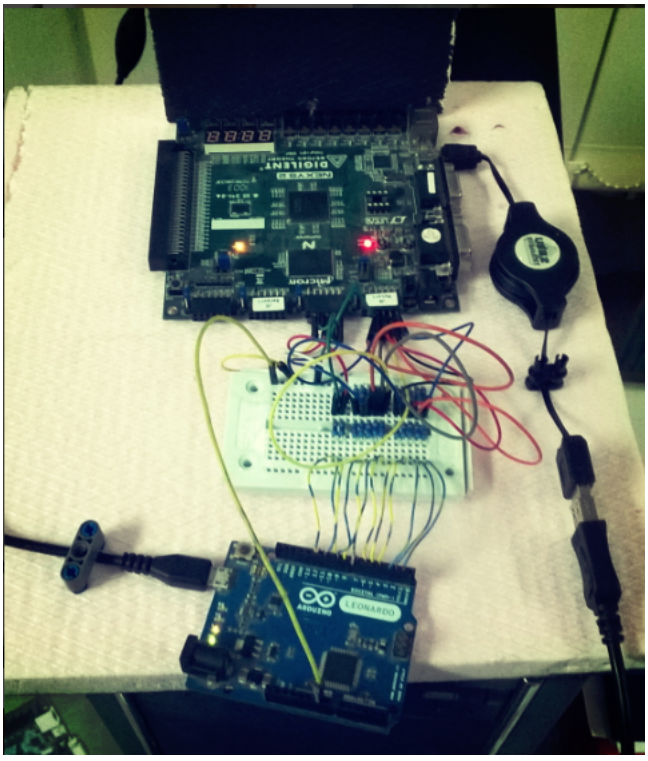
Сл. 2.1: Упрошћени дијаграм *FPGA* чипа

Програмабилни логички блокови у себи садрже одређен број логичких кола способних да обаве основне логичке функције. Та кола су најчешће мултиплексери или *RAM* меморије, које се понашају као универзална логичка кола. Адресна линија служи као улаз, док подаци који се налазе на датој адреси представљају излазе. Тако је могуће имплементирати таблице истинитости унутар меморије које ће садржати у себи све вредности улаза и излаза операције која се извршава. На излазу се налазе регистри у виду флип-флопова који чувају последњи резултат операције.
Програмабилни прекидачи се налазе између програмабилних логичких блокова и служе да споје блокове у одређену целину. Постоји неколико метода имплементације ових прекидача (променљиви и непроменљиви). Променљиви програмабилни прекидачи су сачињени од транзистора који се понаша као прекидач и меморијске ћелије повезане на капију транзистора. Када је у меморијској ћелији уписана логичка јединица, напон се доводи на капију транзистора, који преспаја везу између два блока, и тиме их спаја. Ако је уписана логичка нула, транзистор не проводи, те блокови нису спојени. Непроменљиви програмабилни прекидачи представљају низ осигурача или антиосигурача, који се, довођењем високог напона, прекидају и тиме трајно спајају или одвајају логичке блокове. Разлика између осигурача и антиосигурача је у томе што осигурачи, приликом довођења напона, одвајају два логичка блока, док их антиосигурачи спајају [9].

3 Пројектовање WEB лабораторије за програмирање и примену *FPGA* развојне платформе

3.1 Лабораторијски модели WEB лабораторије за програмирање и примену *FPGA* развојне платформе

Експерименти на даљину **WEB** лабораторије, намењени лабораторијској подршци у програмирању и примени *FPGA* развојне платформе, су засновани на програмирању *Digilent Nexys2-500k* плоча и мониторингу ефеката написаних програма. При том, улазни сигнали *FPGA* развојне платформе се генеришу путем *Arduino Leonardo* развојне плоче и/или *Raspberry Pi* минирачунара. На располагању су два лабораторијска модела за програмирање и примену *FPGA (Digilent Nexys2-500k)* у оквиру **WEB** лабораторије. Лабораторијски модел интегрише и *FPGA* и хардвер за генерисање улаза у *FPGA*, и детекцију излаза из *FPGA*. У случају првог лабораторијског модела користи се Arduino Leonardo развојна плоча са низом отпорничких делитеља напона за емулацију физичких прекидача и тастера који се налазе на *FPGA* развојној платформи. На тај начин је могуће контролисати тастере *FPGA* развојне платформе путем рачунара. Лабораторијски модел је приказан на слици 3.1 (на слици није приказан рачунар који омогућава програмирање *FPGA* развојне платформе која је повезана на **WEB**). Други лабораторијски модел има исту функцију, подједнако ефикасан као и претходни, али је јефтинији. Ради се о томе, да је јефтини минирачунар *Raspberry Pi* заменио рачунар и *Arduino Leonardo* који су коришћени у претходном лабораторијском моделу.



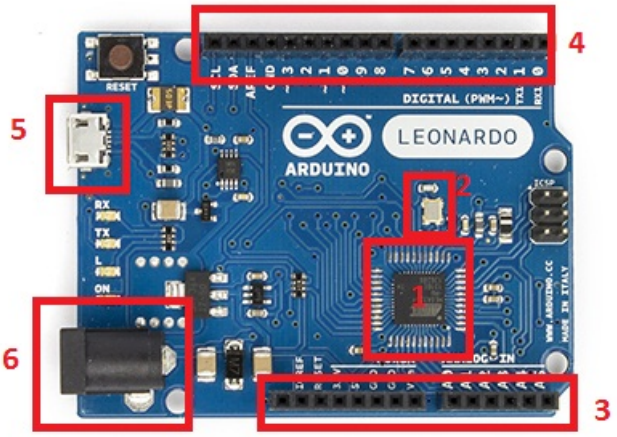
Сл. 3.1: Лабораторијски модел **WEB** лабораторије за програмирање и примену *FPGA* развојне платформе *Digilent Nexys2-500k*

Лабораторијски модели интегришу и **WEB** камере путем којих је могуће пратити извршење експеримената путем графичког интерфејса **WEB** лабораторије.

3.1.1 *Arduino Leonardo* развојна платформа

Arduino Leonardo (Сл. 3.2) је развојна платформа базирана на *Atmel AVR ATmega32u4* (1) микроконтролеру са следећим карактеристикама [10]:

- Радни такт: 16 MHz (2)
- Радна меморија: 2.5 KB статичке *RAM* меморије
- Програмска меморија: 28 KB *FLASH*
- Корисничка меморија: 1 KB *EEPROM*
- Број аналогних улаза: 12 (3)
- Резолуција АД конвертера: 10 бита
- Број дигиталних улаза/излаза: 20 (4)
- Број *PWM* канала: 7
- Тип конекције: *USB* (5)
- Радни напон: 7 V - 20 V (6)



Сл. 3.2: *Arduino Leonardo* развојна платформа

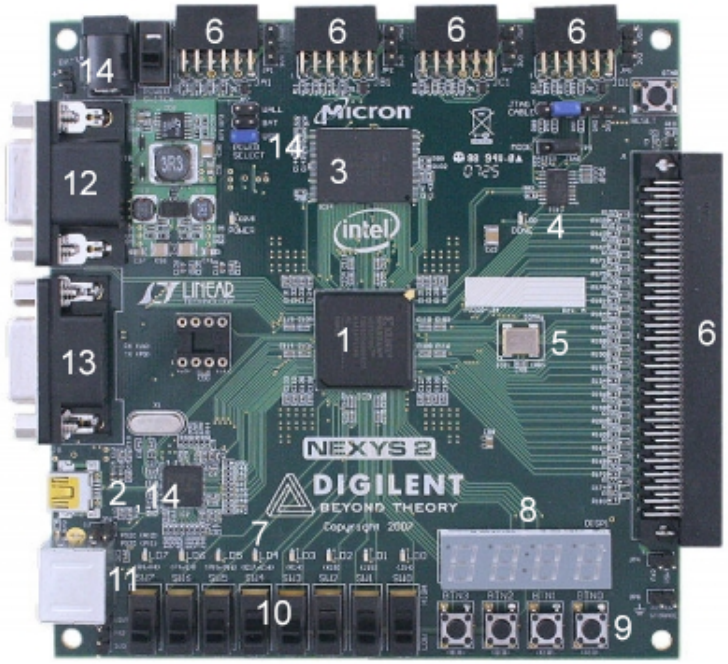
Arduino Leonardo развојна платформа се програмира помоћу програмског језика *C*, уз помоћ *AVR-GCC* компајлера који је интегрисан у *Arduino IDE* који се бесплатно може преузети са [11].

3.1.2 *Digilent Nexys2-500k FPGA* развојна платформа

Digilent Nexys2-500k FPGA (Сл. 3.3) развојна платформа се заснива на *Xilinx Spartan 3E FPGA* интегрисаном колу. Спецификације ове развојне плоче су [12]:

- *Xilinx Spartan 3E FPGA* интегрисано коло (1)
- Конфигурисање путем *USB2* протокола (2)
- 16 MB *Micron PSDRAM* и 16 MB *Intel StrataFlash ROM* меморије (3)

- *Xilinx Platform Flash* за конзервирану конфигурацију (4)
- 50 MHz осцилатор са могућношћу додавања произвољног осцилатора (5)
- 60 дигиталних улазно-излазних пинова (3.3 V) (6)
- 8 ЛЕ диода (7), четвороцифрени седмосегментни екран (8), 4 тастера (9), 8 прекидача (10)
- *PS/2* конектор (11), *VGA* конектор (12), *RS-232* конектор (13)
- Напајање помоћу *USB* прикључка, батерија или адаптера (14)



Сл. 3.3: *Digilent Nexys2-500k* развојна платформа

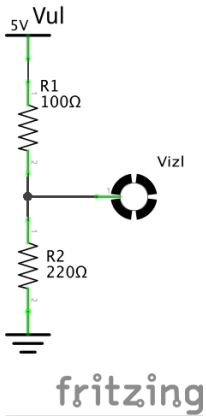
Ова *FPGA* развојна плоча се програмира у језицима за описивање хардвера, првенствено у језику **VHDL** и језику **Verilog**. Језик **Verilog** је коришћен у овом раду. Предност *FPGA* развојне платформе наспрам микропроцесора и микроконтролера се огледа у томе што корисник може да имплементира произвољну логику и није ограничен конкретном архитектуром. Друга предност јесте што *FPGA* развојне платформе обављају процесе паралелно, за разлику од процесора и микроконтролера који инструкције извршавају секвенцијално, што може бити погодно за брже извршавање специфичних алгоритама.

3.1.3 Хардверска интеграција лабораторијског модела: Повезивање *FPGA* развојне платформе са микроконтролером. Капацитет могућности лабораторијског модела.

Да би се контролисали тастери и прекидачи *FPGA* развојне платформе даљински (путем Интернета), потребно је направити интерфејс са рачунаром који ће да контролише тастере. За успешно савладавање курса [13], потребно је да кориснику на располагању буду четири тастера и осам прекидача. Да би се постигла контрола укупно дванаест улаза (осам прекидача и четири тастера), у даљем тексту је описано спајање *FPGA* развојне

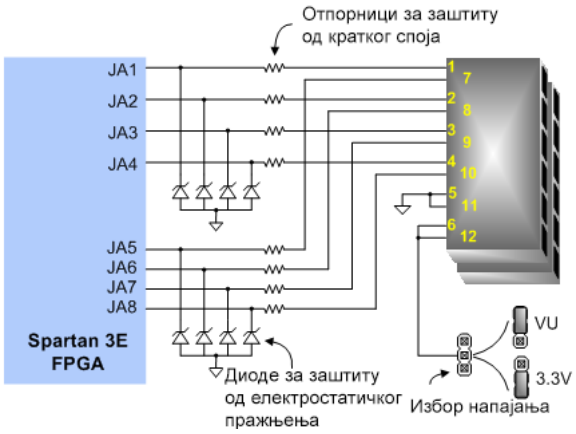
платформе са *Arduino Leonardo* развојном платформом. Будући да *FPGA* није толерантан на напон од 5 V, делитељи напона су обавезни. Делитељ напона се састоји од два редно повезана отпорника. Слободан крај првог отпорника је везан за улазни напон, слободан крај другог за нулти потенцијал, а чвор између отпорника даје подељен напон по формули (1) (Сл. 3.4.).

$$V_{izl} = V_{ul} \frac{R_2}{R_1 + R_2} \tag{1}$$



Сл. 3.4: Отпорнички делитељ напона

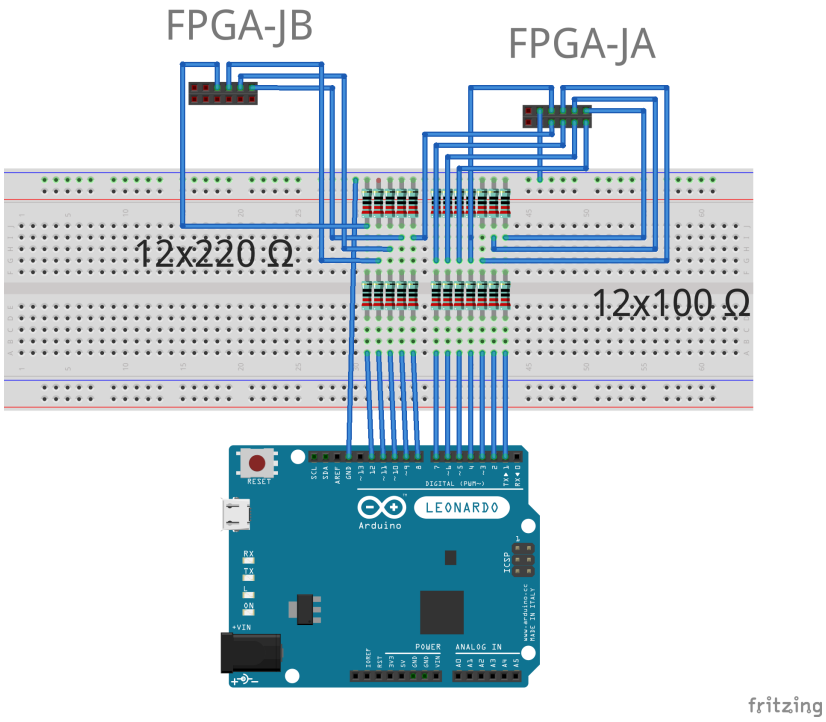
Digilent Nexys2 ради са напоном на дигиталним улазима од 3.3 V, док *Arduino Leonardo* ради са 5 V, па је коришћењем једначине (1) добијено да су отпорници у односу $R_2 = 2R_1$. Коришћени отпорници имају номиналну вредност $R_1 = 100\ \Omega$ и $R_2 = 220\ \Omega$. *Digilent Nexys2* поседује 4 *PMod* интерфејса у облику дванаестопинских конектора намењених за качење периферија. Сваки од њих поседује 8 пинова који по потреби могу бити или улази или излази. Шема *PMod* конектора дата је на слици 3.5 [1] [12].



Сл. 3.5: Шема *PMod* конектора

Дигитални излази *Arduino Leonardo* платформе *(D1-D12)* су спојени на дигиталне улазе *Digilent Nexys2* платформе *(JA1-JA8, JB1-JB4)* посредством отпорничке мреже. Шема везивања је дата на слици 3.6.

[1] Слика је власништво компаније *Digilent, Inc* и постављена је уз дозволу

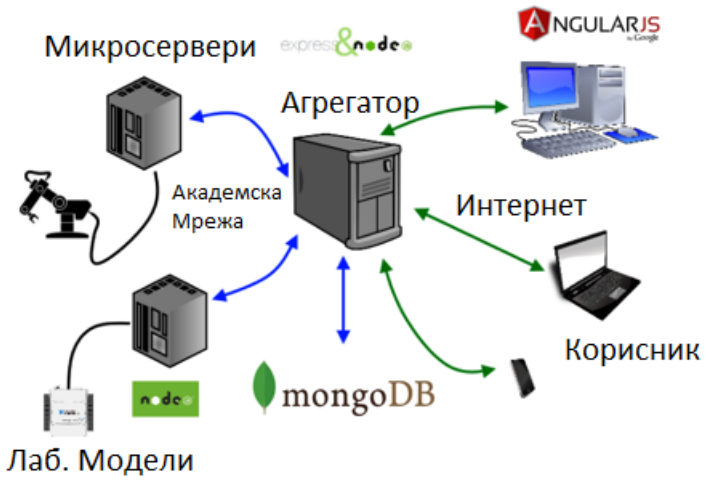


Сл. 3.6: Шема везивања *Arduino Leonardo* платформе са *Digilent Nexys2* платформом

Оваква лабораторијска поставка је ограничена у смислу да корисник има контролу над дванаест улаза у *FPGA* уређај (осам прекидача и четири тастера). Број доступних улаза је сасвим довољан за експерименталну верификацију задатака предмета *MIT 6.111 Introductory Digital Systems Laboratory* чији наставни материјали су дати на [13].

3.2 Софтверско решење и сервиси WEB лабораторије. Имплементација лаб. модела са *FPGA* уређајем у WEB лабораторију

SCOPES пројекат IZ74Z0_160454 / 1: "Enabling Web-based Remote Laboratory Community and Infrastructure" of Swiss National Science Foundation је подстакао потребу за евентуалном ревизијом техничког решења [14] *WEB* лабораторије на Универзитету у Крагујевцу. У овом раду је предложено ново софтверско решење *WEB* лабораторије које функционише на серверу који се налази у Центру за примењену аутоматику на адреси [7]. Приницпијелна шема *WEB* лабораторије је приказана на слици 3.7.



Сл. 3.7: Поједностављен дијаграм удаљене лабораторије

Предложено софтверско решење *WEB* лабораторије је написано у програмсом језику *JavaScript* коришћењем *Node.JS* окружења. Састоји се из два дела: Централног сервера који се понаша као агрегатор имплементираних лабораторијских модела у *WEB* лабораторију. У њему се налази база података свих корисника и свих доступних лабораторијских модела. Клијент се конектује на централни сервер, где може изабрати лабораторијски модел на ком хоће да изврши експеримент. Други део представљају појединачни лабораторијски модели интегрисани у *WEB* лабораторију. Појединачни експерименти над појединачним лабораторијским моделима се реализују посредством микросервера. Један микросервер опслужује један експеримент. Корисник се повезује на централни сервер **WEB** лабораторије, у коме се налази листа свих постојећих лабораторијских модела. Кориснички интерфејс *WEB* лабораторије је тако осмишљен да се приликом приступа Интернет странама *WEB* лабораторије посредством различитих уређаја (паметних телефона, таблета, рачунара), димензије Интернет страна прилагођавају екрану уређаја са кога се приступа *WEB* лабораторији. Централни сервер узима податке из базе података, стилизује их, и показује кориснику у виду **HTML** страница или линкова. Уласком на линк конкретног лабораторијског модела *WEB* лабораторије, централни сервер повлачи са микросервера лабораторијског модела страницу експеримента, интегрише је у своју **WEB** страну, и такву је сервира кориснику.

3.2.1 Централни сервер WEB лабораторије

Централни сервер **WEB** лабораторије (агрегатор лабораторијских модела у оквиру *WEB* лабораторије) је писан уз помоћ *MEAN Stack*-а, што је целина коју чине серверска компонента **Node.JS**, рутер Express.JS, **NoSQL** база података **MongoDB** и клијентска компонента **Angular.JS**. Централни сервер служи да комуницира са базом података у којој се налазе колекције које чувају све информације о сваком лабораторијском моделу интегрисаном у *WEB* лабораторију, као и информације о корисницима. Такође, централни сервер поседује **RESTful API** за комуникацију између клијентске и серверске стране, где у духу *WEB 2.0*, клијент не мора по захтевању новог садржаја да тражи комплетну страну од сервера, већ само освежава део стране у коме се налазе потребни подаци. Овај **API** поседује све команде које је могуће извршити на агрегатору, а то су:

- Креирање новог корисничког налога
- Мењање корисничког налога
- Додавање нових микросервера

- Додавање описа лабораторијских вежби
- Преглед свих постојећих вежби
- Преглед корисника
- Додавање и измена лабораторијских модела

Приликом прављења новог налога, корисник на своју *e-mail* адресу добија верификациони линк на који треба да оде да би потврдио свој налог. Налози који нису верификовани се бришу у року од 24 часа. База података поседује две колекције. Прва колекција је колекција **labs** и она садржи релевантне параметре о лабораторијским моделима интегрисаним у **WEB** лабораторију. Поља у овој колекцији су следећа:

- **_id** Уникатни идентификациони број
- **name** Име лабораторијског модела и редни број, уколико је више лабораторијских модела истог типа
- **short** Кратко име
- **setupDescript** Опис лабораторијског модела, **HTML** страна, низ који садржи онолико описа колико има лабораторијских поставки
- **assignments** Могући задаци, **HTML** страна
- **skeleton** Егземпларни и скелетон кодови, **HTML** страна
- **guide** Упутство за коришћење **WEB** лабораторије, **HTML** страна
- **labLinks** Линкови ка конкретним микросерверима, низ који садржи онолико линкова колико има лабораторијских модела, чини уређен пар са **setupDescript**
- **MaPbackground** Опис теоријске позадине осмишљене лабораторијске вежбе и карактеристика лабораторијског модела, **HTML** страна
- **CaMapparatus** Опис опреме која је коришћена за мерење и аквизицију у датом експерименту, односно на датом лабораторијском моделу
- **experimentStructure** Опис структуре експеримента, **HTML** страна
- **dateCreated** Време и датум интегрисања лабораторијског модела у *WEB* лабораторију
- **thumb** Мала слика лабораторијског модела, за насловну страну агрегатора
- **description** Кратак опис лабораторијског модела интегрисаног у *WEB* лабораторију

Друга колекција садржи информације о корисницима, и то:

- **password** Лозинка, енкриптована са **SHA-256** енкрипцијом
- **email** Корисников *e-mail* налог
- **institution** Факултет
- **labs** Листа лабораторија које је корисник посетио, кад их је посетио, и колико се задржао

- **role** Чува информације да ли је корисник студент, администратор или модератор

Корисничка страна је рађена помоћу **HTML5**, **Angular.JS** и **jQuery** технологија. За визуелни садржај је коришћена **Bootstrap css** библиотека. У зависности од позване путање, кориснику се приказују различити погледи (**Views**). Списак истих је дат испод.

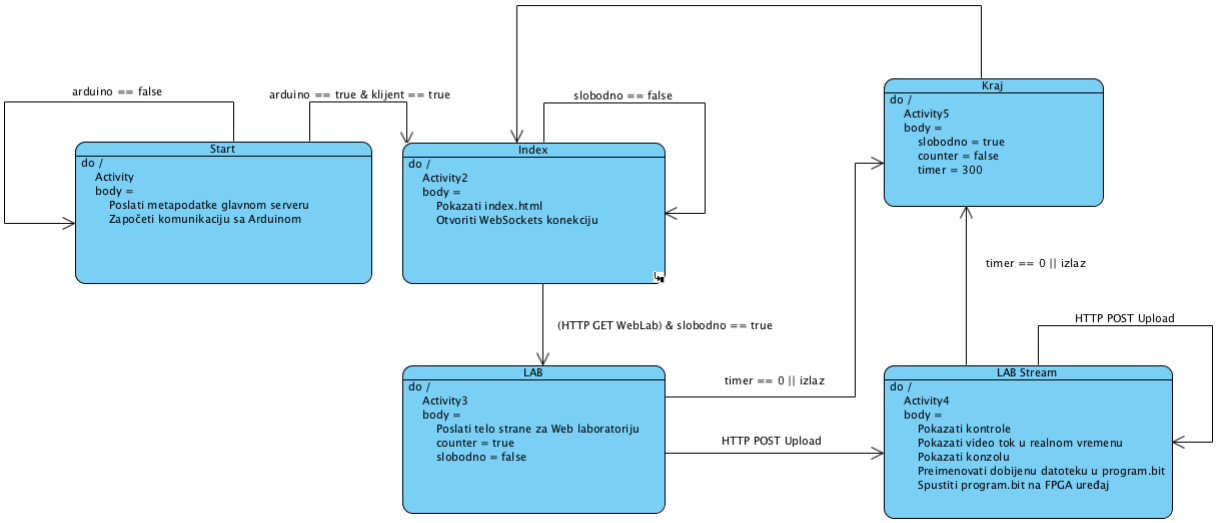
- **formaLogin** Форма за приступ
- **infoView** Преглед садржаја лабораторијског модела, повлачи податке путем **API**ја из колекције лабораторијских модела у оквиру *WEB* лабораторије.
- **labAdd** Графички интерфејс за додавање лабораторијских модела, поседује уграђен **HTML** едитор, као и могућност слања мултимедијалног садржаја
- **labList** Приказује све доступне лабораторијске моделе у оквиру *WEB* лабораторије, иначе основни поглед када се уђе на агрегатор. Поседује могућност претраге. Сваком лабораторијском моделу се додељује слику (**Thumb**), име и кратки опис.
- **navbar** Навигациона трака
- **webLabView** Конкретна **WEB** страна лабораторијског модела, користи **iFrame** за приказ **HTML** документа микросервера

Сваки поглед има свој одговарајући контролер, у коме се налази сва програмска логика. Овиме је постигнута **MVC** (*Model-View-Controller*) филозофија, где је модел садржај базе података, поглед оно што корисник види, а контролер логика која повезује модел и поглед.

3.2.2 Микросервер

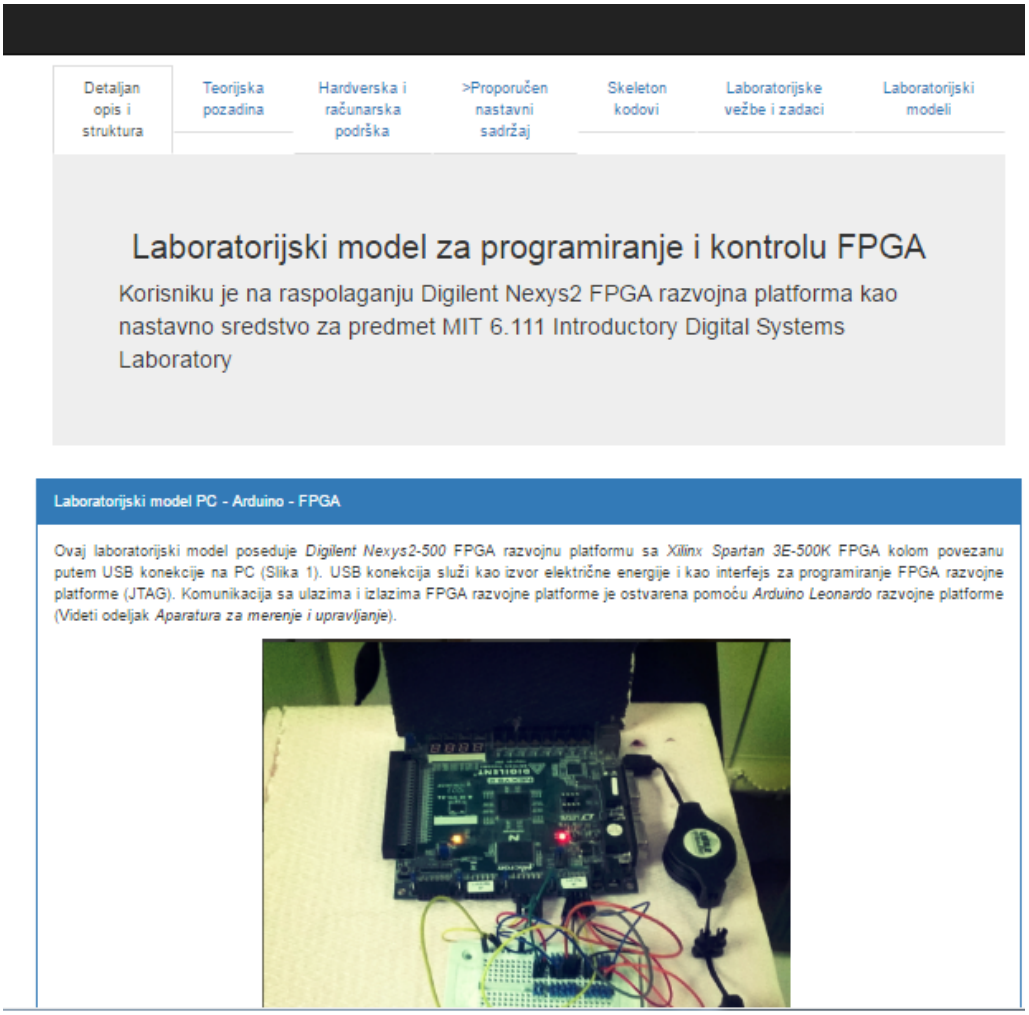
Микросервер је развијен уз помоћ програмског језика *Javascript* користећи *Node.JS* радно окружење. Ако лабораторијски модел није заузет, сервира се **HTML** страница са контролама за управљање експерименталном поставком, као и два видео преноса (један показује индикацију улаза и излаза на *FPGA* уређају, а други *LCD* монитор који је повезан на *FPGA* уређај). **WEB** лабораторија је осмишљена тако да се од корисника очекује да постави или имплементира сопствени програм који путем *FPGA* или неког другог уређаја управља објекат управљања и мери ефекте управљања. Тај програм не би смео да узрокује евентуалне штетне ефекте. Из тог разлога, али и да би се олакшало кориснику да се не бави специфичним деловима програма, прогам се проверава да би се пронашли потенцијални безбедносни ризици и коначно се убацује у постојећи скелетон код са функцијама нижег нивоа за управљање објектом и стартује се експеримент где корисник види визуализацију улаза и излаза на графику који се освежава у реалном времену посредством *Socket.IO* библиотеке. Такође, корисник види и конзолу из које може да добије све релевантне параметре и упозорења. У случају овог рада, уместо програма за контролу објекта управљања, корисник шаље *bitstream* датотеку коју је добио компајлирањем *Verilog* кода, која затим стартује програм *Digilent Adept* који исту спушта на *FPGA* уређај. Корисник потом може да управља уређајем помоћу тастера који се налазе на **WEB** страници лабораторије, и да добија видео у реалном времену *FPGA* развојне платформе и монитора (у случају да ради са *VGA* излазом развојне платформе). Да би се остварила комуникација између *FPGA* развојне платформе и корисника у реалном времену, потребно је направити посредника. У овом случају је коришћен Ардуино Леонардо на кога је стављен *Firmata* протокол и минирачунар *Raspberry Pi*. Овај протокол служи за контролу улаза и излаза Ардуино развојне платформе помоћу серијске везе путем

рачунара. Библиотека која омогућава серијску везу између Ардуина и **WEB** лабораторије је *Johnny-Five*. Извршење експеримента на микросерверу лабораторијске поставке се моделира путем коначне машине стања, где су окидачи за прелазак у следеће стање унутрашњи догађаји направљени посредством библиотеке *EventEmitter*. Дијаграм стања је могуће видети на слици 3.8.

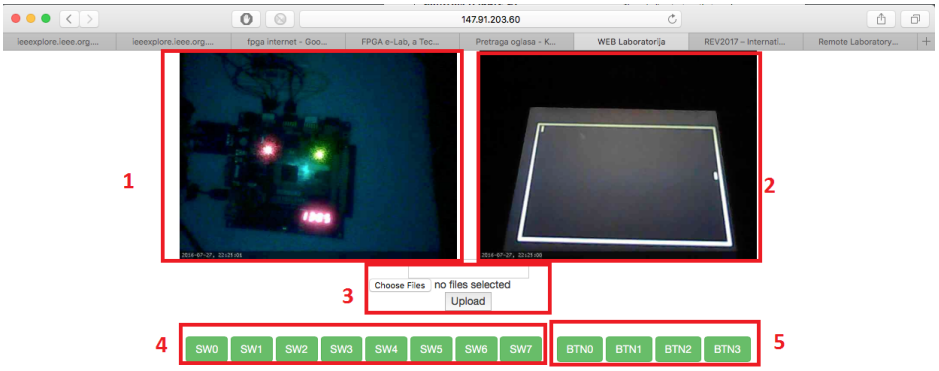


Сл. 3.8: Дијаграм стања извршења експеримената на FPGA(Лабораторијски модел са Arduino Leonardo платформом за постављање улаза на FPGA)

Приликом укључивања сервера лабораторије, покреће се комуникација са Ардуином посредством библиотеке *Johnny-Five*. Када се комуникација оствари, покреће се **WEB** сервер и лабораторијском моделу *WEB* лабораторије је могуће приступити. Када корисник приступи лабораторисјком моделу на коме ће се експериментисати путем **WEB** лабораторије, долази до активирања бројача који гарантује кориснику одређено време током којег може да користи лабораторијски модел. За то време, други корисници не могу да експериментишу на истом лабораторијском моделу јер је заузет, док је могуће посматрати тренутни ток експеримента који се извршава на лабораторијском моделу. По истеку времена, кориснику се одузима контрола, и други корисници могу да приступе лабораторијском моделу. Изглед почетне стране приступа лабораторијском моделу *WEB* лабораторије на агрегатору је могуће видети на слици 3.9, док је извршење експеримената могуће пратити путем *WEB* камера, како је то показано на слици 3.10. Камера која је уперена у *FPGA* развојну платформу је означена бројем 1, камера уперена у монитор бројем 2. Корисник програм уноси у поље означено бројем 3, и за контролу истог су му доступни осам прекидача (4), и четири тастера (5).



Сл. 3.9: Уводна *WEB* страна лабораторијског модела за програмирање и коришћење *FPGA*



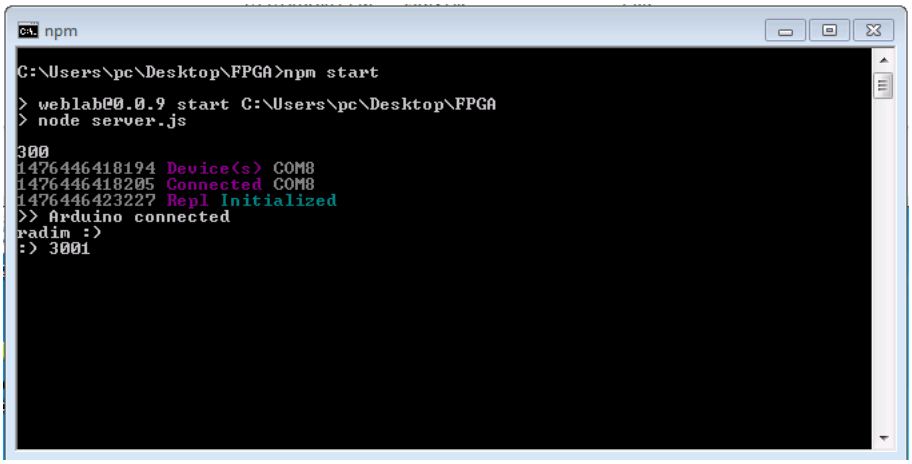
Сл. 3.10: Графички кориснички интерфејс са приказом *WEB* камера при извршењу експеримента на лабораторијском моделу

Клијентска страна лабораторије је писана уз помоћ *HTML5* и *CSS3* технологија. Контролер који је задужен за интеракцију корисник-лабораторијски модел је писан у језику *JavaScript* коришћењем *jQuery* библиотеке. Стилизовање елемената странице лабораторијског модела је писано у *CSS3* језику коришћењем библиотеке *Twitter Bootstrap*. Постоји више отворених комуникационих линија између клијента и сервера. Прва отворена линија је за почетну страну, где се са сервера клијенту шаљу информације о заузећу лабораторијског модела у тренутку приступа. Друга линија је за саму лабораторију, где се

шаљу корисникове команде (притиснути тастери или прекидачи) серверу. Програмирање *FPGA* развојне платформе се покреће када корисник пошаље програм притиском на тастер *Upload*, који потом шаље помоћу *HTTP POST* упита програм серверу. Сервер такође шаље два *MJPEG* видео тока клијенту на којима се налази *FPGA* уређај и монитор повезан на *FPGA* уређај. Изворни код микросервера се налази у додатку овог рада.

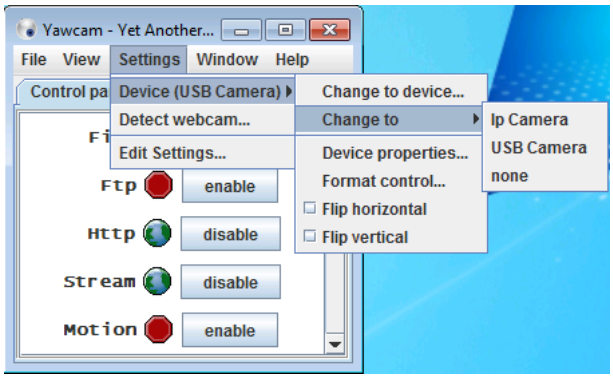
4 Конфигурација микросервера

Да би се покренуо микросервер лабораторијског модела, потребно је отићи у директоријум где се налази микросервер, и покренути програм **StartRL**. После неколико секунди, приказаће се информација да је сервер покренут, и у конзоли ће бити исписан порт на коме се налази (сл. 4.1).



Сл. 4.1: Испис информација на конзоли сервера

Потом, потребно је у датотеци *index3.html* у истом директоријуму променити *IP* адресу и порт на ону на којем се налази дат сервер (*Додатак 2, 65. линија*). У овом раду за потребе видео преноса са камере коришћена је апликација *YawCam*, али могуће је користити било коју другу апликацију која то омогућава. Покренути апликацију *YawCam*, одабрати камеру која се налази изнад *FPGA* развојне платформе, и притиснути *Http* и *Stream* дугме (сл. 4.2).



Сл. 4.2: Избор камере у апликацији *YawCam*

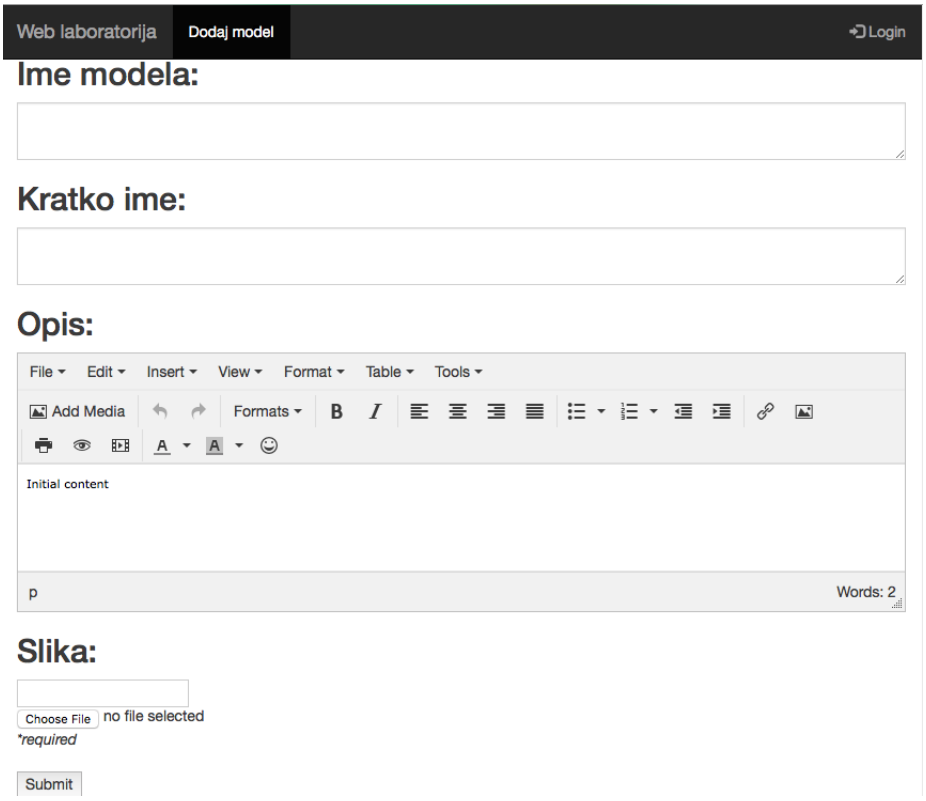
Покренути још једну инстанцу *YawCam* апликације, изабрати камеру која показује на монитор и притиснути исту дугмад као за прву инстанцу. У датотеци *index3.html* у линији 23 написати адресу на којој се налази укључен *YawCam*. У линији 27 написати исту адресу, а порт променити на 8082. Овиме је завршена конфигурација микросервера.

5 Интеграција лабораторијског модела у агрегатор

Да би се интегрисао лабораторијски модел у агрегатор, потребно је отићи на сајт агрегатора [7] (*cpa.fin.kg.ac.rs/weblab/index*). У навигационом панелу, потребно је пиритиснути дугме *Dodaj model* (сл. 5.1).

Сл. 5.1: Навигациони панел агрегатора

Отвара се нова страна у којој власник лабораторијског модела треба да попуни информације да би се тај лабораторијски модел нашао у агрегатору (сл. 5.2).

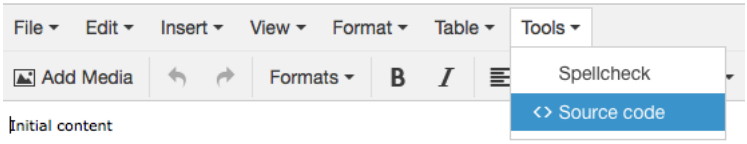


Сл. 5.2: Поља за унос имена и описа лабораторијског модела

У пољу *Ime modela* потребно је унети име лабораторијског модела. Ово поље мора да буде уникатно, што значи да два модела не могу да носе исто име. У пољу *Kratko ime* потребно је унети име као једну реч, и то поље служи за базу података и не приказује се нигде на страни. У пољу *Opis* потребно је унети кратак опис поставке који ће се појавити на насловној страни агрегатора испод слике модела. Форма за унос описа је рађена уз помоћ *TinyMCE WYSIWYG HTML* [2] едитора, што значи да је у њу могуће додавати разни мултимедијални садржај који ће бити интерпретиран као *HTML* документ. По жељи,

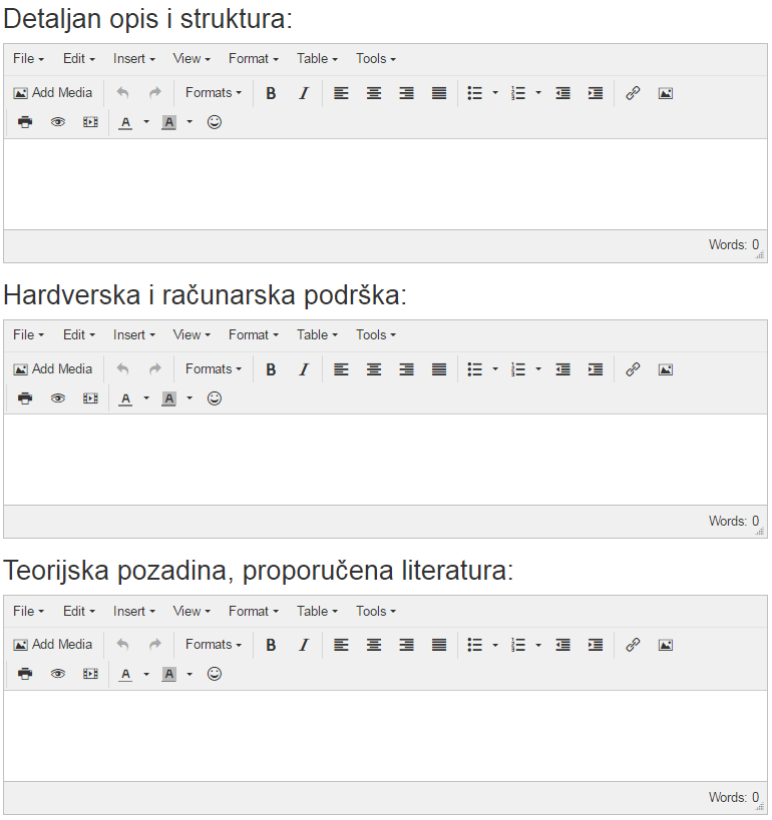
[2] *What You See Is What You Get, енгл. Добијаш шта видиш, нап. аут.*

корисник може додати *HTML* код и сам, притиском на тастер *Tools*, и затим тастер *Source Code* (сл. 5.3).



Сл. 5.3: Мени за одабир уноса *HTML* кода

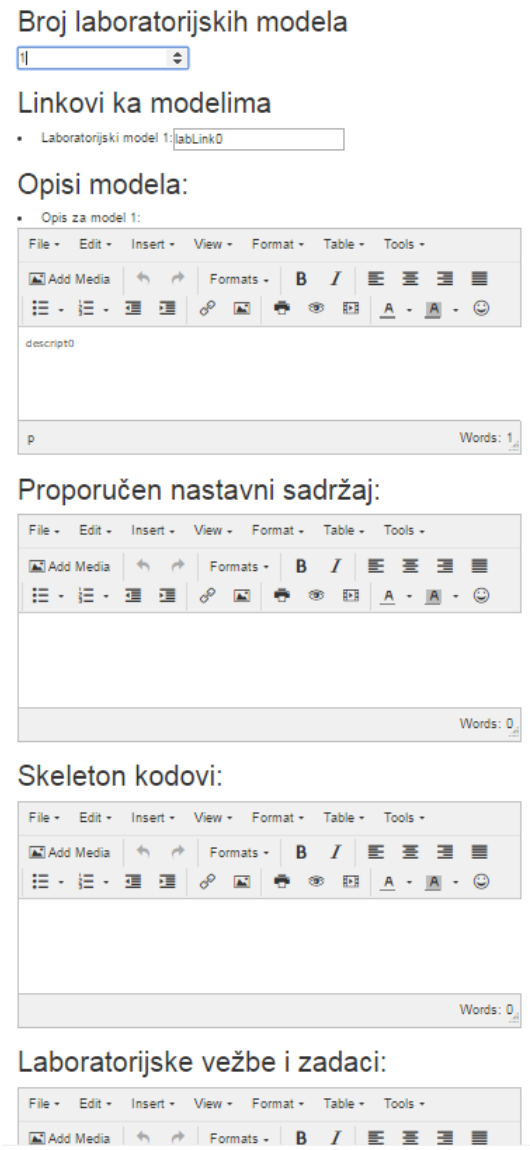
Будући да је за графички интерфејс коришћен *Twitter Bootstrap*, у *HTML* коду је могуће унети било коју класу из те библиотеке, и она ће бити интегрисана у крајњи интерфејс. Поље *Slika* треба да садржи слику модела, ширине 300, а висине 200 пиксела. Следећа поља које је потребно унети су: детаљан опис и структура лабораторијског модела, хардверска и рачунарска подршка и теорисјка позадина експеримента са препорученом литературом (сл. 5.4).



Сл. 5.4: Поља за унос детаљног описа и сврхе лабораторијског модела

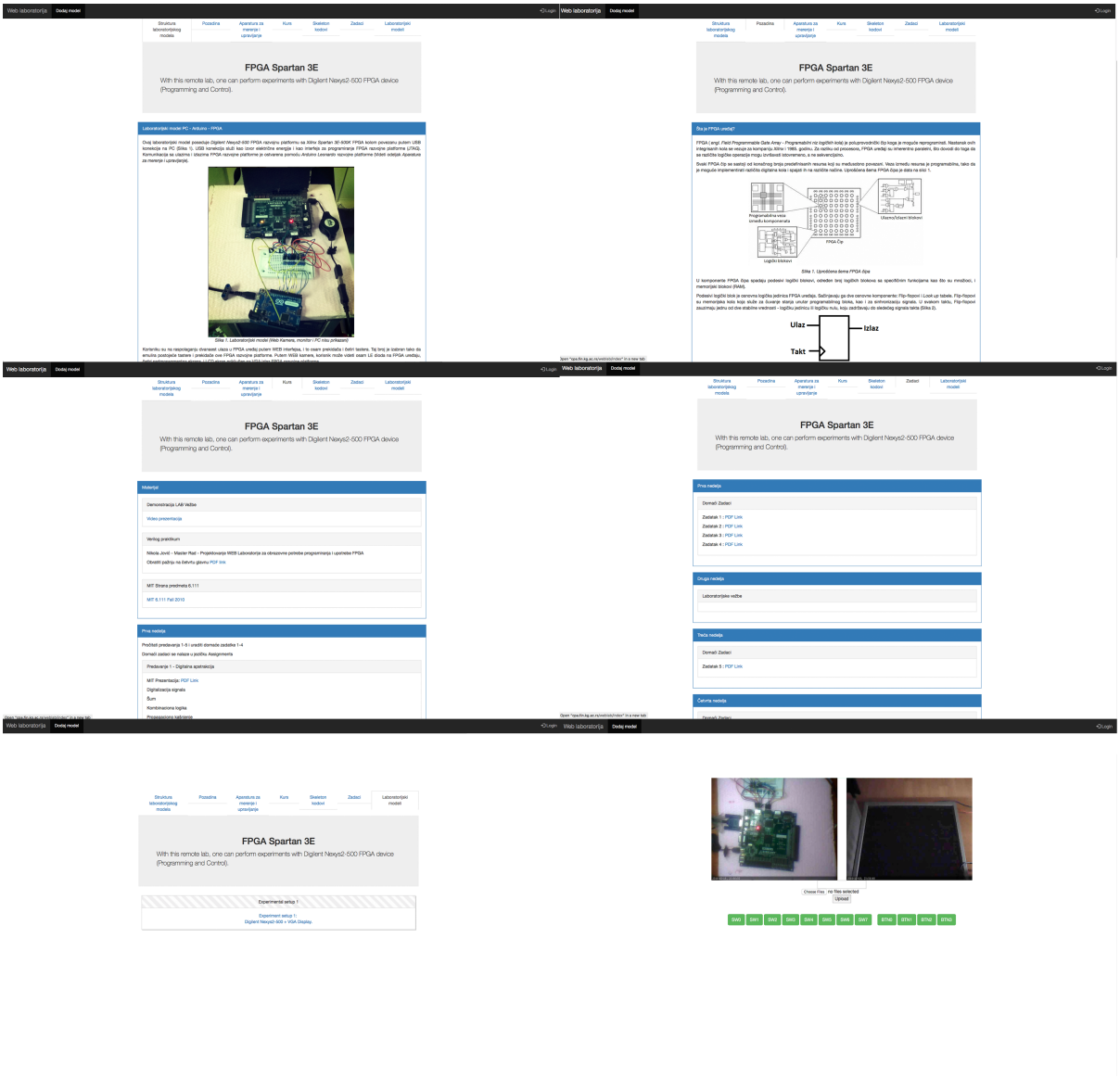
У пољу структура лабораторијског модела, потребно је унети детаљан опис целине лабораторијског модела, предвиђене сврхе и капацитета истог. Затим описати структуру полазећи од функционалне шеме модела, па завршно до шема прилагодне електронике за спајање сензора и актуатора. Кориснику треба омогућити увид у токове енергије и информација у оквиру лабораторијског модела и како је то физички остварено. Даје се техничко решење лабораторијског модела у мери у којој је то доступно, могуће и има сврху. У пољу апаратура за мерење и управљање, потребно је описати како је објекат управљања повезан са уређајем за аквизицију и контролу (ад/да конвертер, на пример), који уређај је коришћен, како је написан софтверски интерфејс, итд. У пољу теоријска позадина,

описати феномен на којем је модел или експеримент базиран. На крају, потребно је дати информацију колико је истих лабораторијских модела у функцији, спецификације сваког модела, уколико има специфичности, препоручени наставни садржај и програм вежбања који се изводи над лабораторијским моделом. (сл. 5.5).



Сл. 5.5: Поља за унос броја модела, линкова ка моделима, описа, наставног садржаја, примера и задатака

У пољу број лабораторијских модела, потребно је унети колико има истих лабораторијских модела који су повезани на Интернет. Аутоматски се прави онолико поља за унос линкова колико има модела. У тим пољима је потребно ставити линк до стране лабораторијског модела. Такође, прави се онолико поља за опис колико има модела. У тим пољима се пишу информације које су специфичне за дат модел. У пољу курс се оставља наставни материјал везан за лабораторијске моделе, у пољу скелетон кодови се остављају примери програма или модули који су потребни за успешно извршење експеримента. У пољу задаци се остављају могући задаци које је потребно реализовати на датим моделима. Да би дати подаци били уписани у базу података и приказани на сајту, потребно је притиснути дугме *Dodaj* на дну странице. На слици 5.6 могуће је видити релевантне стране постављене за експериментисање са *FPGA* уређајем.



Сл. 5.6: Приказ страна на сајту. а) Детаљан опис и структура б) Теоријска позадина и препоручена литература в) Препоручен наставни садржај г) Препоручен програм вежбања д) Списак лабораторијских модела ђ) Кориснички интерфејс за извођење експеримента на лабораторијском моделу

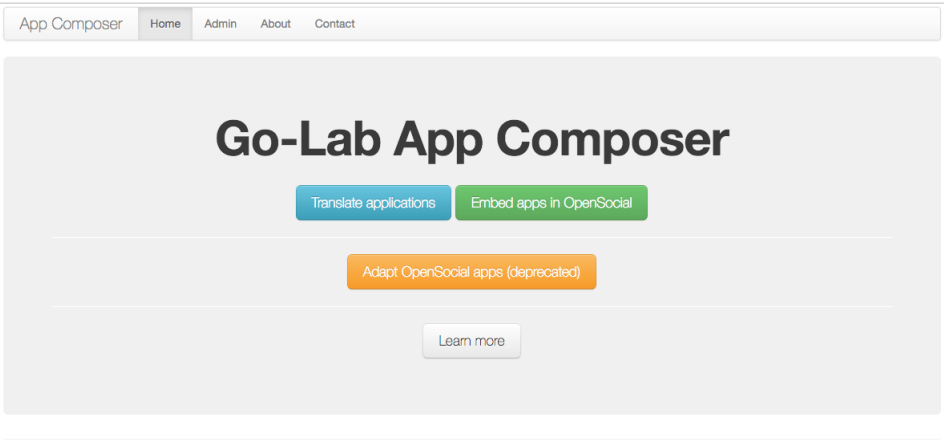
Поступак покретања експеримента је дат у [7], на адреси (*http://cpa.fin.kg.ac.rs/weblab/public/courses/fpga/video.webm*).

6 Интеграција лабораторијског модела у *Go-Lab* окружење

Go-Lab портал обједињује проблемски орјентисане наставне садржаје, засноване на физичким или виртуелним (анимираним) експериментима, а који се могу постављати и користити са ма ког места у свету за потребе образовања ученика и студената у природним наукама. *Go-Lab* обједињује портале виртуелних лабораторија и физичких **WEB** лабораторија. Ове лабораторије су основа за наставу засновану на демонстрацији физичких феномена (који се испољавају реализацијом експеримената), као и основа за проблемски орјентисано учење и учење засновано на истраживању. Ова методологија подразумева да се лабораторије, било да су засноване на физичким или виртуелним експериментима, комбинују са мултимедијалним садржајем и дидактичким апликацијама, а све у циљу подршке процесу учења код студента. *Go-Lab* пројекат је финансиран од стране Европске Комисије унутар оквира *"7th Framework Programme (FP7)"*. *Go-Lab* је скраћеница за Global Online Science Labs for Inquiry Learning at School *(енгл. Глобално умрежене лабораторије за проблемски орјентисано учење у основним и средњим школама)*. Учење засновано на истраживању је приступ у коме студенти прате своја питања и хипотезе, стварајући знање истраживањем и експериментисањем. Типични процес учења заснованом на истраживању се састоји од неколико фаза учења организованих у циклус истраживања. Карактеристичне фазе су:

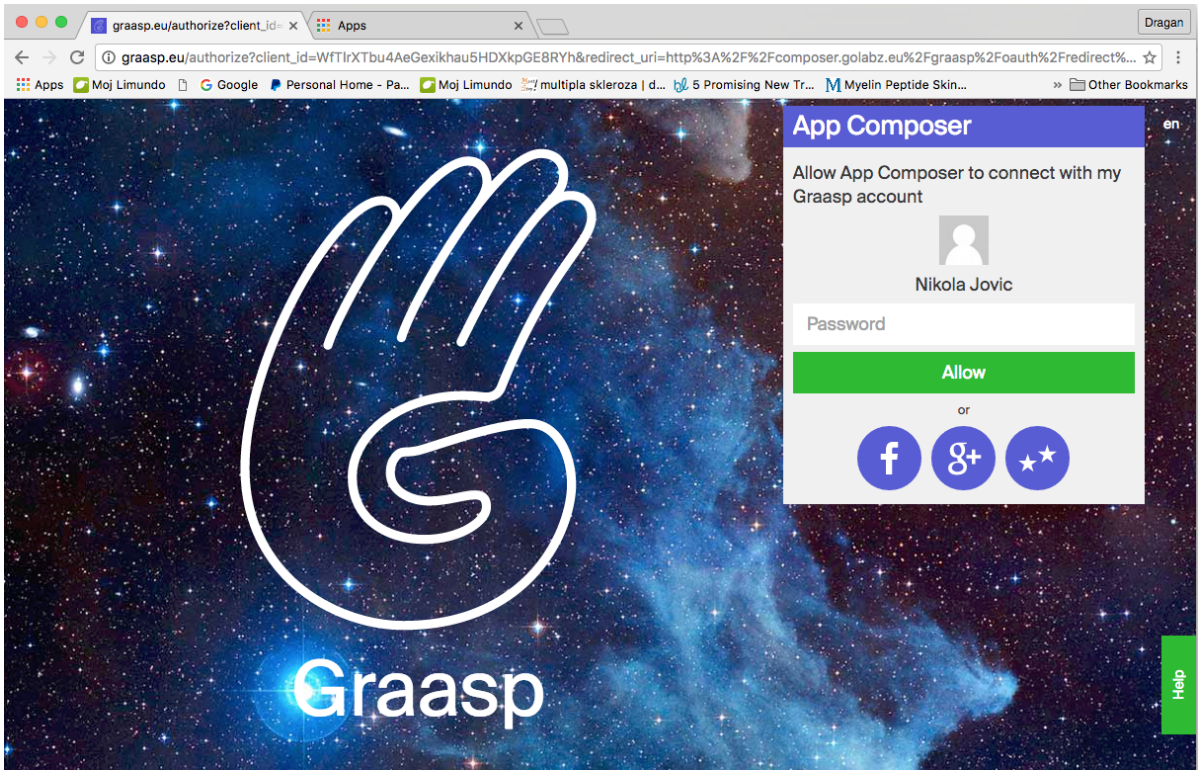
- Орјентација (Дефинисање предмета истраживања)
- Концептуализација (Апстраховање реалности у мислима)
- Истраживање
- Дискусија
- Закључак

Да би учење засновано на истраживању било ефективно, саме лабораторије нису довољне, већ студенти морају бити вођени задацима, истраживачким алатима и материјалима. Помоћу *Go-Lab* окружења, наставници могу да направе виртуелни простор за учење засновано на истраживању који ће пружити потребне наставне ресурсе и алате својим студентима [15]. У даљем тексту биће описан поступак постављања направљеног лабораторијског модела у *Go-Lab*.
Да би се страна лабораторијског модела поставила у *Go-Lab*, потребно је да она буде *OpenSocial* апликација. Будући да се не ради о таквој апликацији, могуће је уградити страницу у *OpenSocial* апликацију помоћу портала *Go-Lab App Composer*(*http://composer.golabz.eu*), слика 6.1.



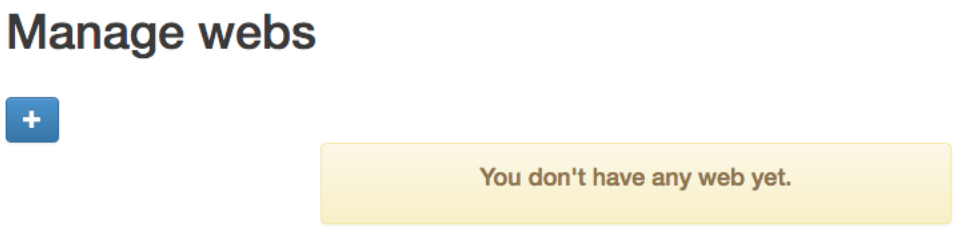
Сл. 6.1: Почетна страна портала *Go-Lab App Composer*

По уласку на страну, потребно је притиснути дугме *Embed apps in OpenSocial*, након чега се отвара *Graasp* страна на којој се корисник пријављује или, ако нема кориснички налог, региструје (слика 6.2).



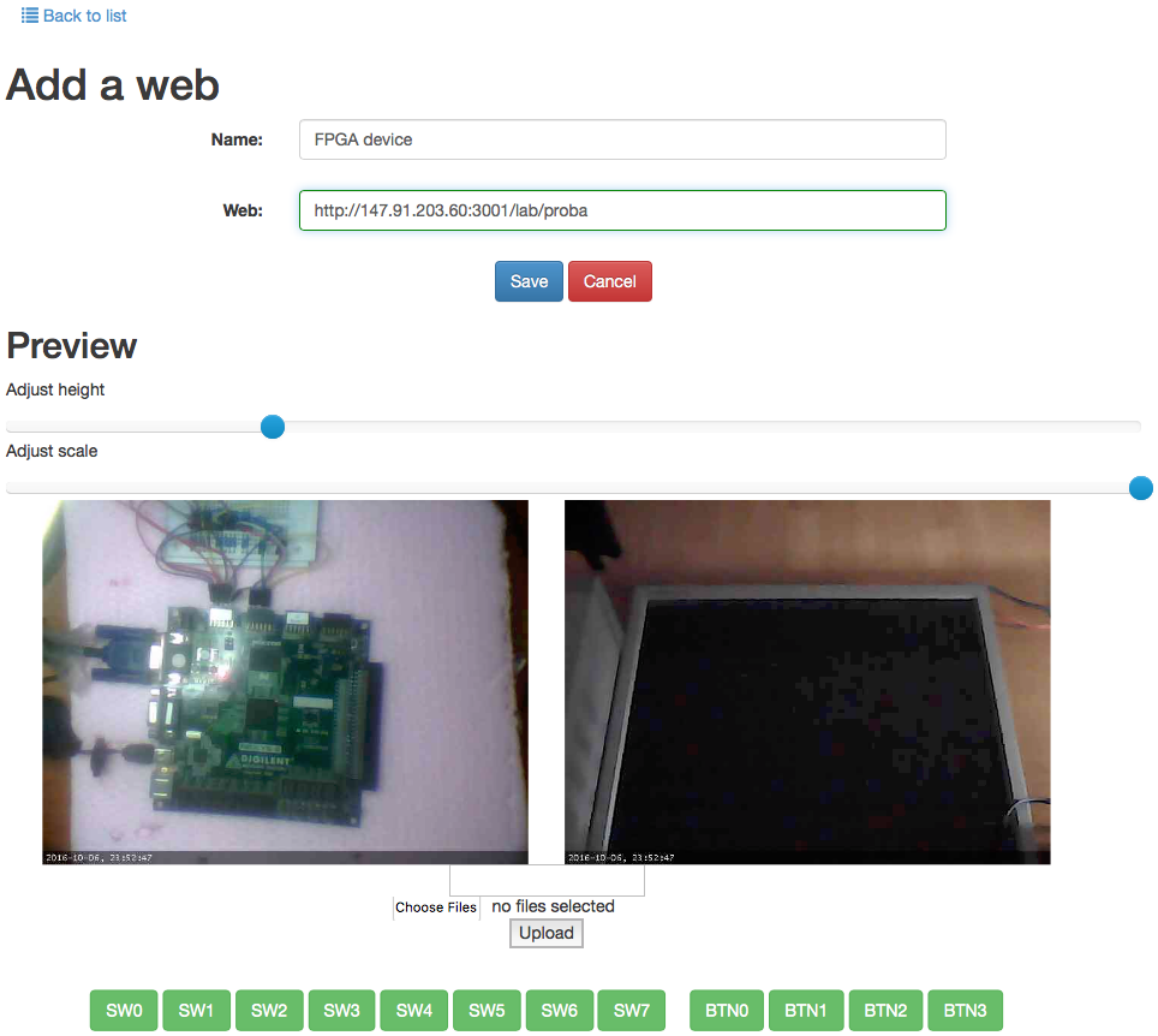
Сл. 6.2: Страна на којој је потребно унети корисничке податке

Пошто се тражени подаци унесу, отвара се страница за администрацију апликација које је корисник направио, слика 6.3.



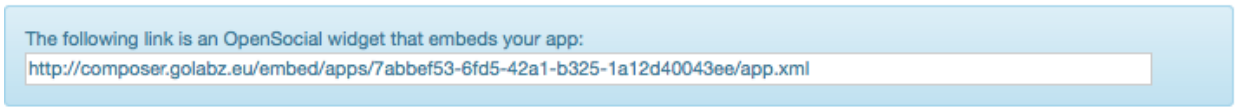
Сл. 6.3: Страна за администрацију корисникових апликација

За интеграцију новог лабораторијског модела, притиснути дугме које има ознаку плус са слике 6.3. Отвориће се страница на којој је потребно унети име лабораторијског модела и линк ка Интернет страни лабораторијског модела. Када се унесе линк, на страници ће се аутоматски приказати садржај који се налази на том линку (*Preview*). Потом, потребно је притиснути дугме *Save* (слика 6.4).



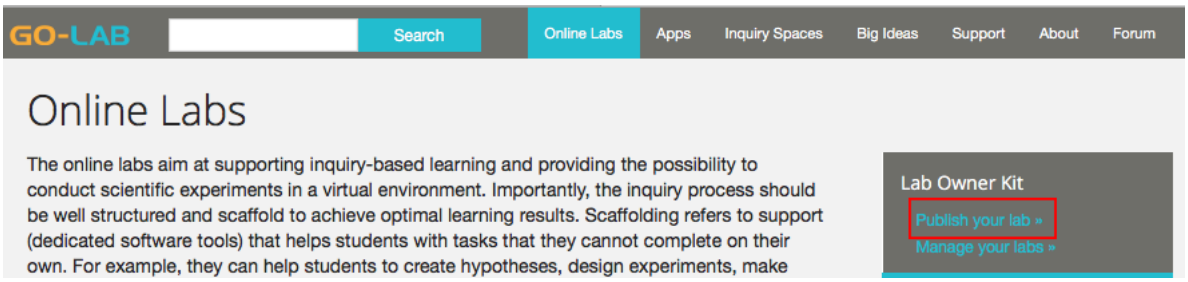
Сл. 6.4: Страна за интеграцију лабораторијског модела у *OpenSocial* апликацију

По завршетку уноса података, на страници ће се приказати линк ка новонасталој апликацији (слика 6.5).



Сл. 6.5: Исписан линк новонастале *OpenSocial* апликације спремне за регистровање у *Go-Lab*

Садржај је потребно ископирати, јер ће линк ка апликацији бити неопходан у предстојећим корацима. Сада, потребно је отићи на страницу *Go-Lab* интернет презентације за онлајн лабораторије *http://golabz.eu/labs* и на десној страни странице, у пољу *Lab Owner Kit* притиснути линк *Publish your lab* уоквирен на слици 6.6 црвеном бојом.

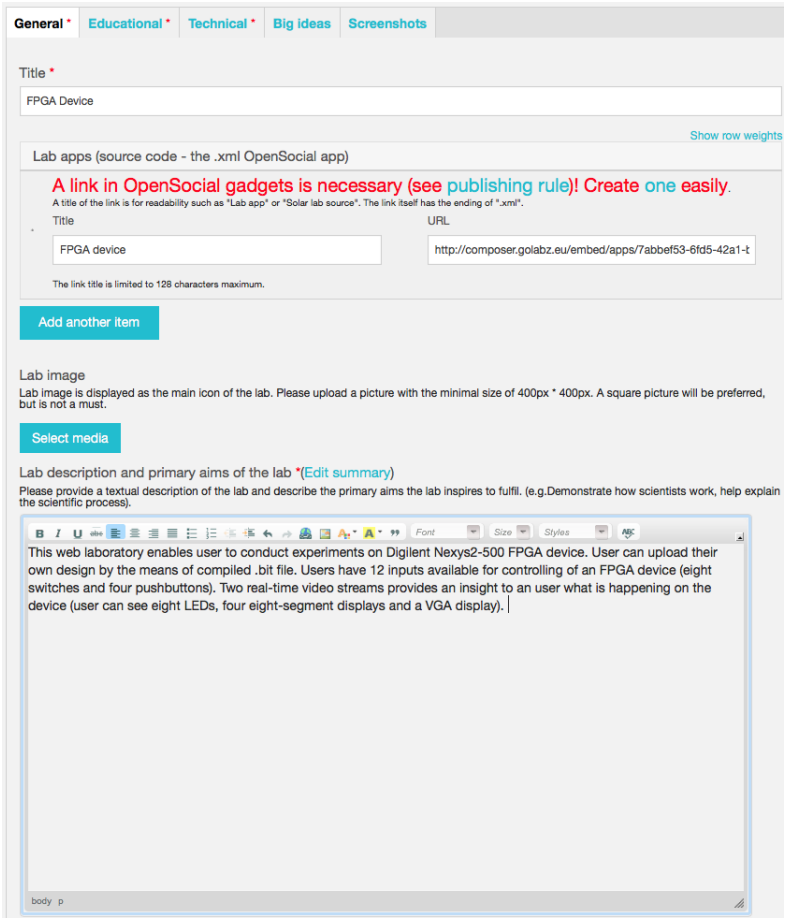


Сл. 6.6: Поље за публикацију лабораторијског модела

Потом, на страници која се отворила, потребно је притиснути дугме *Publish your lab* (сл. 6.7), након чега ће се отворити формулар који је потребно попунити да би се страна лабораторијског модела завела у базу података сервиса *Go-Lab*.

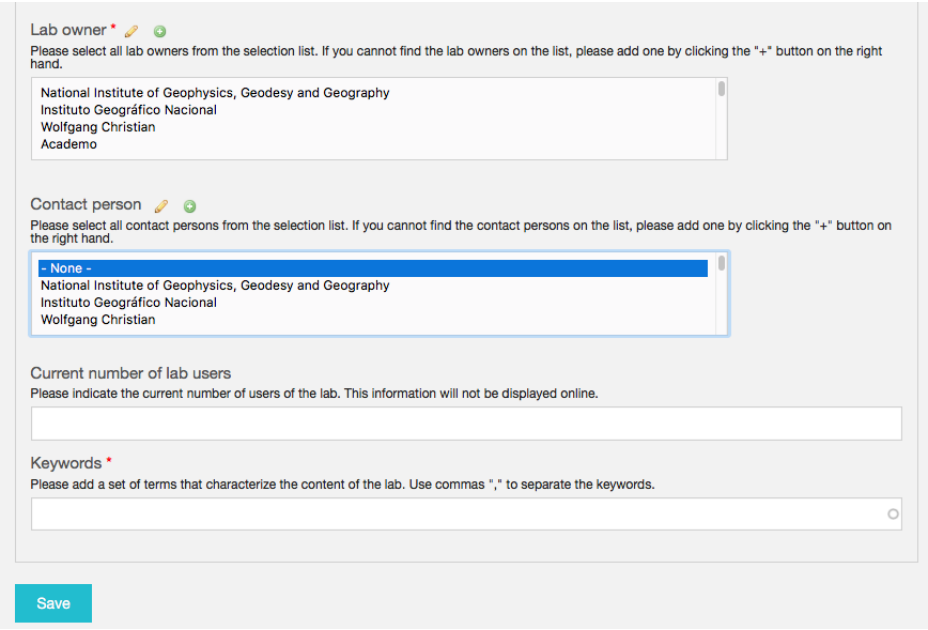
Сл. 6.7: Дугме за отварање форме за унос лабораторијског модела

Формулар је подељен са неколико језичака, где се у првом (*General*) језичку налазе поља за информације као што су име лабораторијског модела, поље за унос линка (добијен изнад) и опис лабораторијског модела (сл. 6.8).



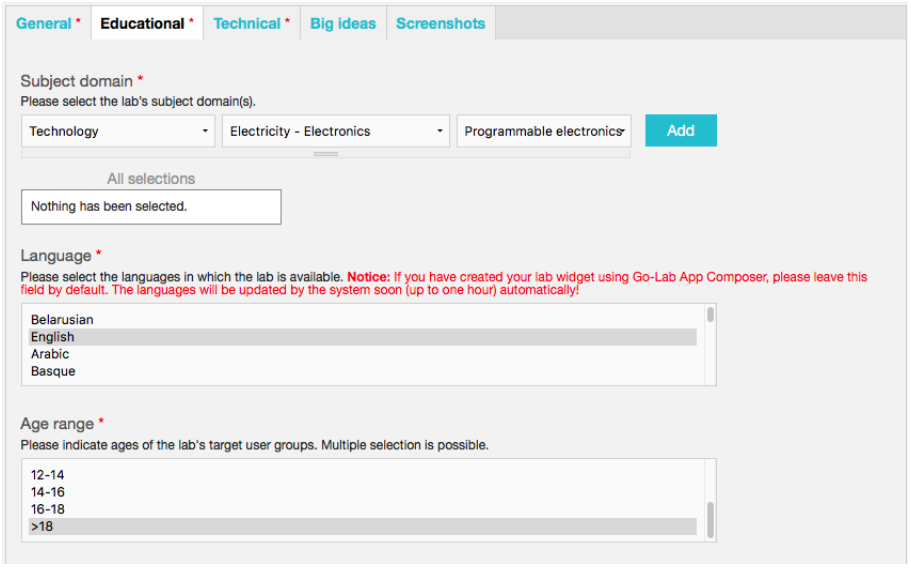
Сл. 6.8: Садржај језичка *General*

У пољу *Title* на врху странице (слика 6.8) потребно је именовати лабораторијски модел који ће бити постављен на *Go-Lab* сервис. Потом, у пољу *Lab apps (source code - the .xml OpenSocial app)*, потребно је написати име направљене апликације у *Title*, и налепити линк до апликације у пољу *URL*. У пољу *Lab description and primary aims of the lab* је потребно написати кратки опис лабораторијског модела, и намену истог. Испод (сл. 6.9) је потребно изабрати власника лабораторије и особу за контактирање. Ако се лабораторија поставља први пут, притиснути зелено дугме са плусом, и унети личне податке. У пољу *Keywords* унети неколико кључних речи по којима ће корисници моћи да претражују.



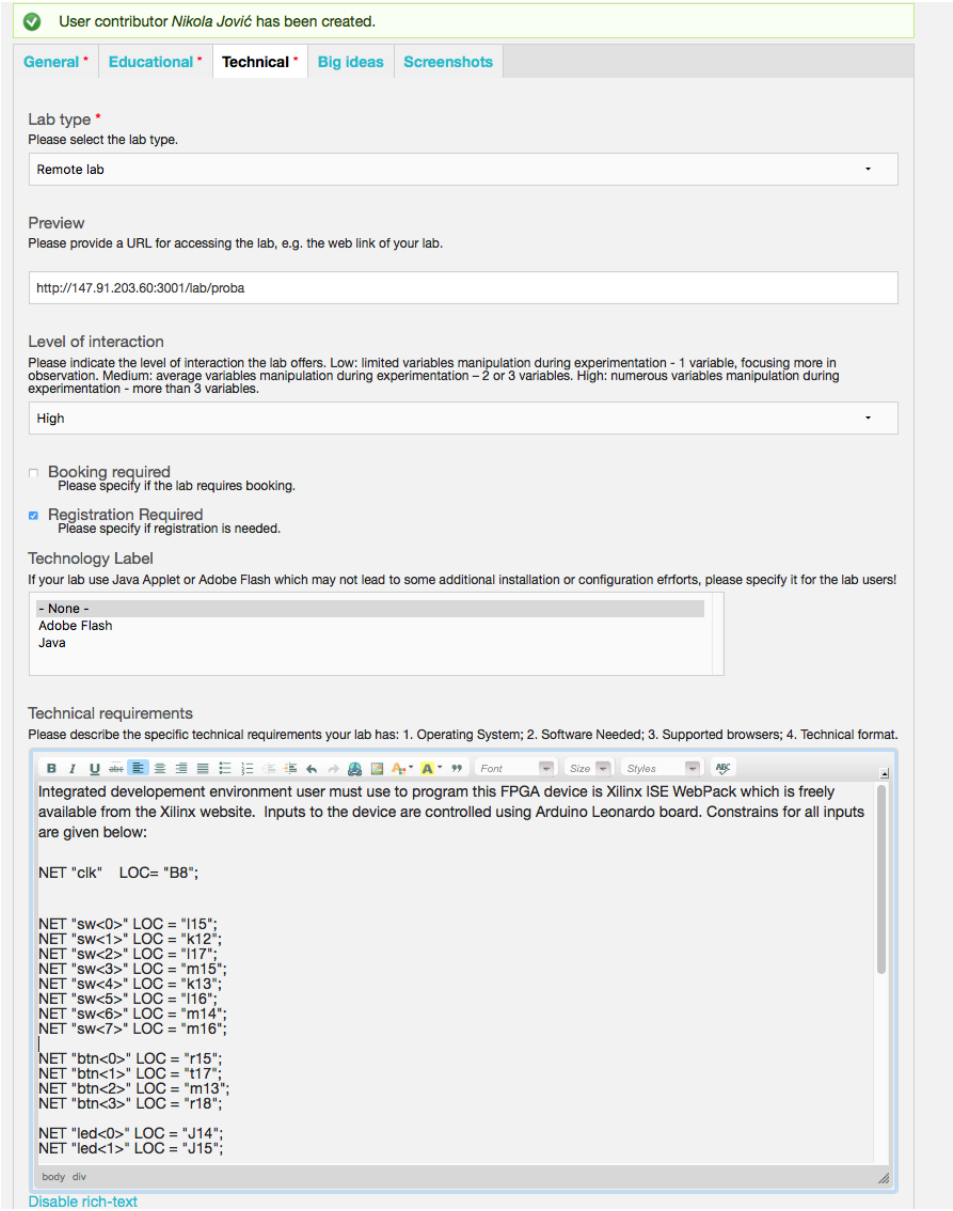
Сл. 6.9: Формулар за власнике лабораторије и кључне речи

По завршетку попуњавања података, прелази се на језичак *Educational*, слика 6.10.



Сл. 6.10: Садржај језичка *Educational*

У овом језичку, потребно је унети податке о области експеримента (*Subject domain*, слика 6.10), језик на ком је експеримент (*Language*, слика 6.10) и старост корисника којима је намењен експеримент (*Age range*, слика 6.10). Последњи језичак који је обавезан да се попуни јесте *Technical* (сл. 6.11).



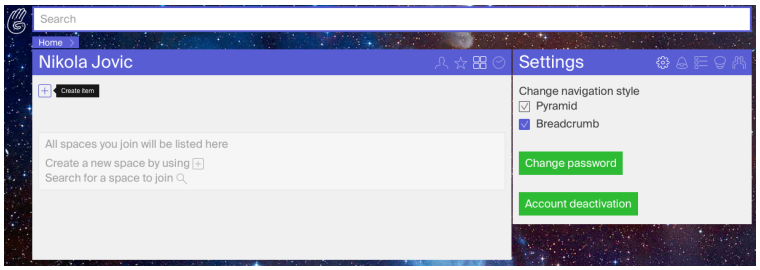
Сл. 6.11: Садржај језичка *Technical*

У пољу *Lab type* (сл. 6.11), унети да је у питању удаљена лабораторија (*Remote lab*), у пољу *Preview* унети линк до странице лабораторијског модела, који омогућава контролу и надзор извршења експеримената. У пољу *Level of interaction* треба назначити ниво интерактивности лабораторије. Ако је могуће контролисати једну промењљиву, ставити *Low*. У случају да је могуће контролисати две или три промењљиве, ставити *Medium*. Ако је могуће контролисати више од три промењљиве, ставити *High*. Означити поље *Booking required* уколико је потребно извршити резервацију лабораторијског модела, а поље *Registration required* уколико је потребно да се корисник региструје да би приступио лабораторијском моделу. У пољу *Technology Label* се уноси технологија која је коришћена при изради удаљене лабораторије. Ако је коришћен само *HTML* и *JavaScript*, назначити - *None* -. Уколико је коришћена нестандардна технологија у виду *Adobe Flash* пакета или *Java Applet-a*, назначити нека од та два поља. У пољу *Technical requirements* навести техничке захтеве које клијентов рачунар мора да испуњава за успешно извршавање експеримента, у виду програмских пакета, оперативног система, као и скелетон кодове ако су потребни. По завршетку уноса неопходних информација, притиснути дугме *Save* на дну странице. Сада, у бази лабораторија на сервису *Go-Lab* могуће је видети постављену лабораторију,

слика 6.12.

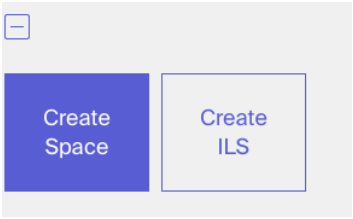
Сл. 6.12: Имплементирана страна лабораторијског модела на сервису *Go-Lab*

Поред евидентирања **WEB** лабораторијских модела, *Go Lab* окружење нуди и могућност организације предмета, делова предмета или курсева који се фокусирају око **WEB** лабораторија ради примене нових методологија наставе и учења попут проблемски орјентисаног учења и учења заснованог на истраживању. Да би се направио виртуални простор за учење, потребно је отићи на сајт *graasp.eu*, пријавити се на сајт, и притиснути дугме плус у горњем левом углу (слика 6.13).



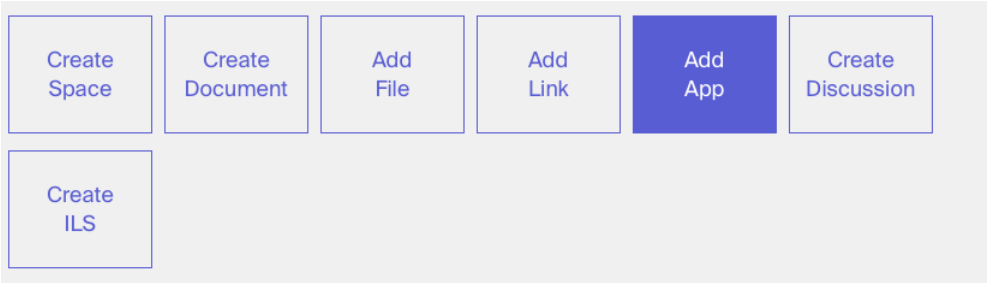
Сл. 6.13: Страна за додавање виртуалног простора за постављање наставних садржаја за учење на сајту *graasp.eu*

Потом, притиснути дугме *Create Space*, слика 6.14, и у пољу *Name* уписати име простора за постављање наставних садржаја за учење.



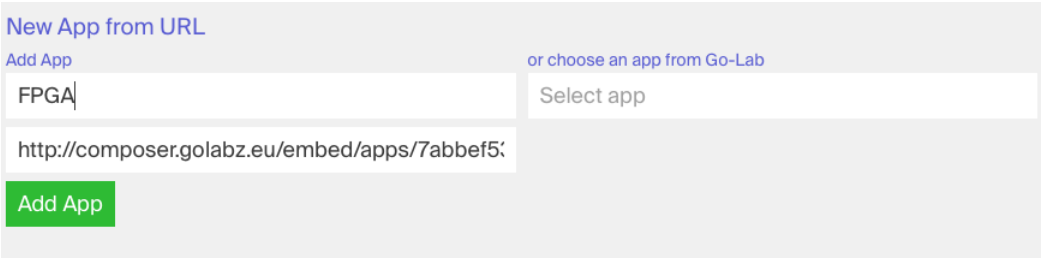
Сл. 6.14: Поље за креирање простора за наставне садржаје

По задавању имена, могуће је на страну додавати мултимедијални садржај по жељи, притиском на дугме плус у горњем левом углу. Да би се додала **WEB** лабораторијска вежба, потребно је притиснути дугме са именом креираног простора, и притиснути дугме плус у горњем левом углу. Појавиће се списак садржаја који је могуће додати, и потребно је притиснути дугме *Add App* (слика 6.15).



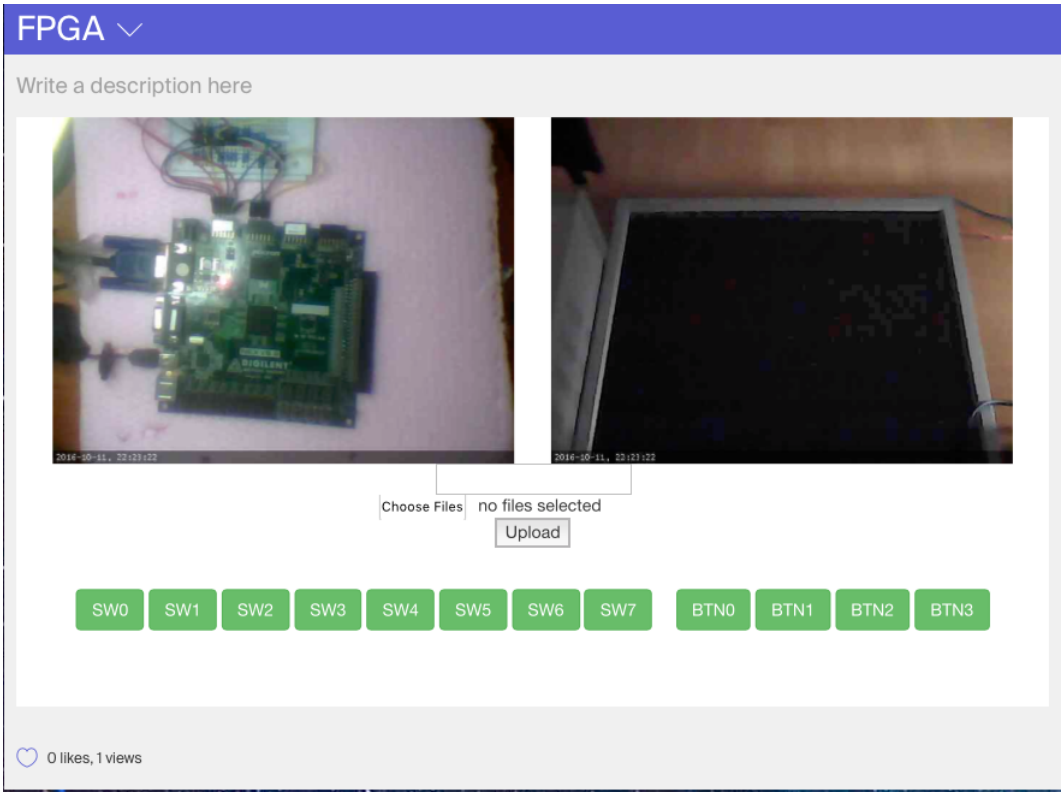
Сл. 6.15: Избор садржаја који је могуће поставити у простор за креирање наставног материјала

Потом, у пољу *Name* уписати име лабораторијског модела, а у пољу *App URL* уписати линк до *OpenSocial* апликације направљене на почетку овог поглавља. Потом, притиснути дугме *Add App* (слика 6.16).



Сл. 6.16: Додавање лабораторијског модела у виртуелни простор за креирање наставног садржаја

Тиме је успешно извршено додавање лабораторијског модела у виртуелни простор за креирање наставног садржаја. Резултат је могуће видети на слици 6.17.



Сл. 6.17: Лабораторијски модел локалне **WEB** лабораторије интегрисан у *GO LAB* окружење

7 Организација наставног садржаја за учење програмирања и употребе *FPGA*

Један од циљева овог рада јесте да омогући реализацију образовне методологије предмета **MIT 6.111** и у случају корисника који немају приступ организованим лабораторијским ресурсима (немају претходна предзнања, стручну помоћ, приступ хардверу - *FPGA* уређајима, немају потребне софтверске инсталације, немају приступ оргранизованим лабораторијским задацима, припремним наставним материјалимам нити решењима лабораторијских задатака). Наставна технологија предмета **MIT 6.111** за основу има да студент пре свега треба да самостално уради осам домаћих задатака и пет лабораторијских вежби - што није могуће спровести без програмирања и тестирања програма на *FPGA* и придруженој опреми. Управо, текстови домаћих задатака и лабораторијских вежби из [13], [16] су искоришћени као основа обуке за програмирање и употребу *FPGA* путем **WEB** лабораторије. При том, вежбе (оригинални домаћи задаци и лабораторијске вежбе [13], [16]) су прилагођене **WEB** окружењу.Курс се бави проблематиком програмирања *FPGA* развојне платформе, и састављен је од осам домаћих задатака и пет лабораторијских вежби. Домаћи задаци и вежбе су изведене из [13]. Вежбе су прилагођене **WEB** окружењу. Кратак *Verilog* курс, као и поставке задатака су постављене на интернет страници **WEB** лабораторије [7]. Задаци и темпо израде лабораторијских вежби су:

- Израда бинарног бројача *74LS163*
- Израда декодера из четворобитног бинарног броја у хексдецимални, приказ на седмосегментном екрану
- Програмирање рада аларма за кола (на основу самостално осмишљене машине стања)

- Програмирање видео игре *Pong*

Релевантне стране **WEB** лабораторије су опремљене и поставкама и решењима лабораторијских задатака (у овом случају поготово, јер емелементарни сет задатака преузет из [13] има функцију подучавања корисника који немају предходна предзнања у овој области). Отуда је **WEB** лабораторија опремљена и наставним материјалима. У осмом поглављу овог рада, дати су урађени примери прва четири задатка, док је пети задатак заправо скуп знања стечених у изради прва четири (бројање импулса, генерисање *PWM* сигнала, машина стања). Курс везан за *FPGA* се састоји из слајдова који су доступни јавности из курса **MIT 6.111**, упутсвима за коришћење потребног софтвера, као и овог мастер рада где су објашњени принципи *Verilog* језика за описивање хардвера. Потребан наставни садржај у вези програмирања и употреба *FPGA* развојне платформе је структуиран према [13], [17] на следећи начин:

1. **Дигитална Апстракција**
   - Дигитализација
   - Шум
   - Комбинаторна логика
   - Пропагационо кашњење
2. **Логичка синтеза**
   - Таблице истинитости
   - Основна логичка кола, универзална логичка кола
   - Минимизација логичких функција
   - Карноове мапе
   - Имплементационе технике (мултиплексери, **Look-up** таблице)
3. **Увод у *Verilog***
   - Модули, инстанце
   - Континуална додела вредности (комбинаторна логика)
   - Секвенцијално понашање, **always** блок
4. **Секвенцијална логика**
   - Дигитална стања, **D** регистри
   - Временске карактеристике регистра
   - Иницијализација регистра у *Verilog* језику
   - Блокирајућа и неблокирајућа додела вредности
5. **Машине стања**
   - Методологија дизајна секвенцијалне логике
   - Идентификовање појединачних стања
   - Прављење дијаграма стања
   - Одабир начина обележавања стања
   - Писање комбинаторног *Verilog* кода за логику следећег стања

- Писање комбинаторног *Verilog* кода за излазне сигнале

6. **Интеграција система**
   - Комуникација унутар машине стања
   - Спровођење сигнала такта
7. **Меморије**
   - Меморије у *Verilog* језику
   - Меморије у *FPGA* уређају
   - Екстерне меморије
     – Синхрона и асинхрона статичка *RAM* меморија
     – Динамичка *RAM* меморија
     – *Flash* меморија
8. **Аритметичка кола**
   - Представљање бројевних вредности
   - Сабирање, одузимање
   - Пренос (*Carry*)
9. **Множиоци**
   - Комбинаторни множилац
   - Множилац комплемента броја два
   - Брзи множиоци
10. **Аналогни блокови**
    - Теорема одабирања
    - Антиалајзинг
    - Дигитални филтери са коначним импулсним одзивом
    - Шум кватнизације
    - Операциона појачала, Дигитално-аналогни и Аналогно-дигитални претварачи
11. **Видео**
    - Генерисање синхронизационих сигнала
    - Генерисање пиксела
      – Тест обрасци
      – Приказивање карактера
      – Видео игре на бази спрајтова

Динамика предмета по недељама је дата у табели 7.1

Табела 7.1: Динамика предмета по недељама

Време	Припрема	Задатак	Рок/Исход
Недеља 1	Предавања 1-5	Домаћи задаци 1-4, Лаб 1	7 Дана
Недеља 2	Лаб 2 упутство	Лаб 2	7 дана
Недеља 3	Предавања 5-6, Лаб 3 упутство	Домаћи задатак 5, Лаб 3, Први део	7 дана, Скица машине стања
Недеља 4	Предавања 7-8	Домаћи задатак 6, Лаб 3, Други део	7 Дана
Недеља 5	Предавања 9-11, Лаб 4 упутство	Домаћи задатак 7, Лаб 4	7 Дана
Недеља 6	Предавања 12-14,	Домаћи задатак 8	7 Дана
Недеља 7-12		Пројекат	4-6 Недеља

8 Програмирање и примена *FPGA* - Практикум решених примера

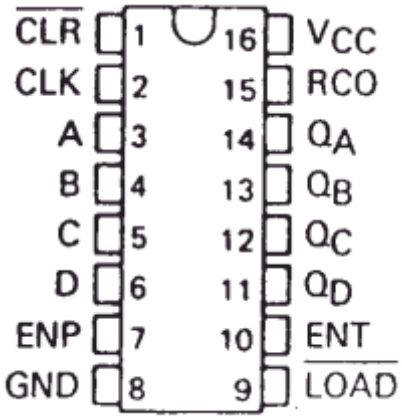
Промовишући принцип проблемски орјентисаног учења, и полазећи од наставне методологије [13], [16] која подразумева израду осам домаћих задатака (прва четири у првој недељи, пети у трећој недељи, шести у четвртој недељи, седми у петој недељи, и осми у шестој недељи) и пет лабораторијских вежби (прва у првој недељи, друга у другој недељи, први део треће у трећој недељи, други део треће у четвртој недељи, четврта у петој недељи и пета у шестој недељи), и на крају самостални студентски пројекат који се спроводи у времену од четири до шест недеља, овде ће бити дато детаљно решавање лабораторијских вежби. Сврха је и да овај рад буде од користи у савладавању увода у програмирање и примену *FPGA* развојне платформе. Зато је посебна пажња посвећена и упутствима о коришћењу *Verilog* језика. Садржај главе која следи представља својеврстан практикум за програмирање и коришћење *FPGA* развојне платформе. Као помоћ при изради овог поглавља, коришћена је литература из [17], [18].

8.1 Задатак 1

8.1.1 Поставка и основни подаци

Написати *Verilog* модул који имплементира логичко коло 74LS163.

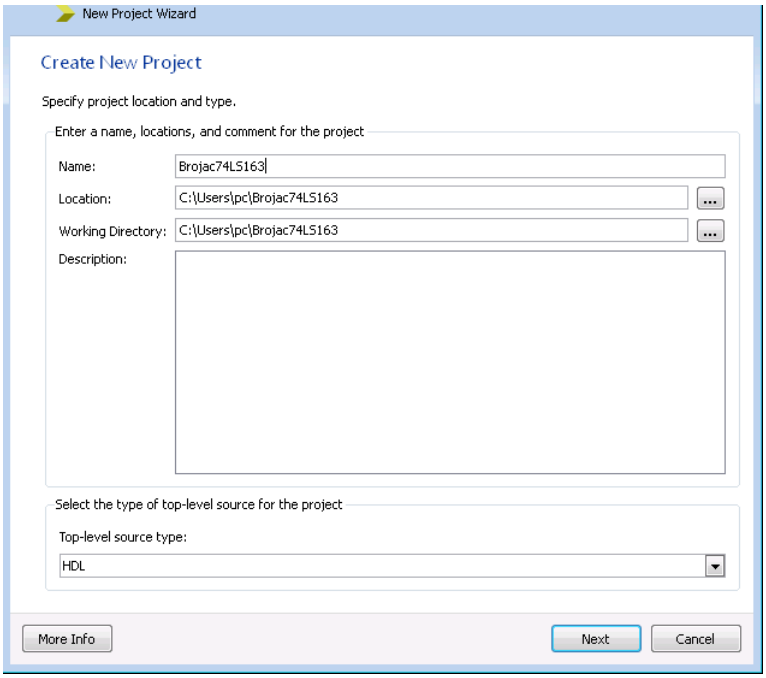
Логичко коло 74LS163 је четворобитни програмабилни бинарни бројач са каскадним излазом за повезивање других бројача (Сл. 8.1). Поседује четири паралелна излаза, као и каскадни излаз. Ради програмирања, користе се четири улаза чија вредност се чува у регистре, и то се узима као почетни број бројача. Бројач може да броји од нуле, или од вредности која му је дата на улазина. Чување улазне вредности се врши спуштањем извода *LOAD* на логичку нулу. Бројач броји када се на извод *CLK* доведе позитивна ивицу такта, и само у случају да су изводи *ENP* и *ENT* на логичкој јединици. Спуштањем извода *CLR* на логичку нулу, бројач се рестартује. Каскадни излаз даје сигнал само ако су сва четири излаза на логичкој јединици (хексдецимални број F), и ако је извод *ENP* на логичкој јединици [19].



Сл. 8.1: 74LS163 четворобитни бинарни бројач

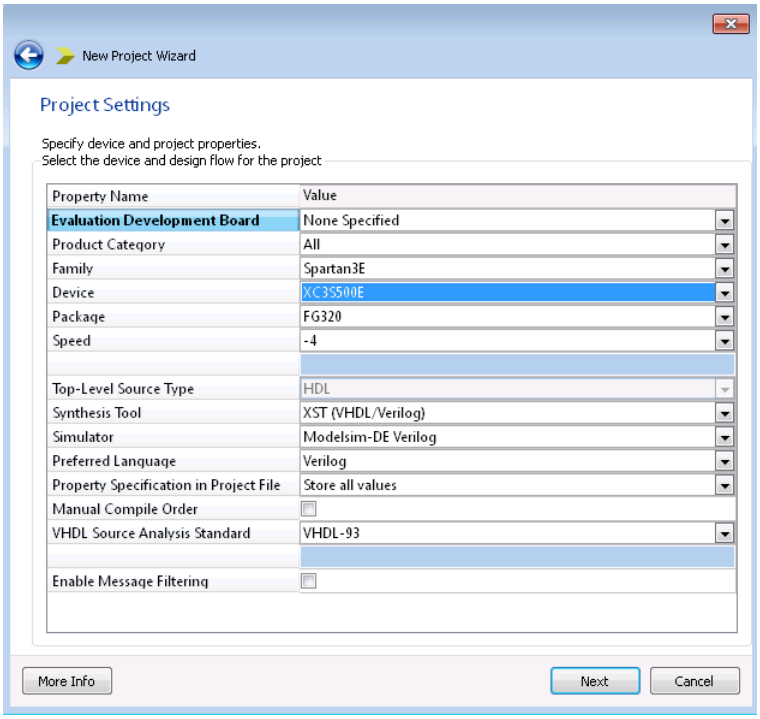
8.1.2 Креирање новог пројекта у пакету *Xilinx ISE WebPack*

Отворити апликацију *Xilinx ISE WebPack*. Када се апликација отвори, направите нови пројекат притиском на *File* > *New Project*. Отвориће се дијалог приказан на слици 8.2.



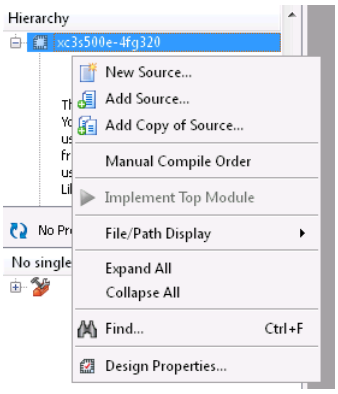
Сл. 8.2: Дијалог за креирање новог пројекта

У пољу *Name* напишите име пројекта (у овом примеру је коришћено име *Brojac74LS163*). Име пројекта не може почињати бројевима или специјалним карактерима. Пројекат ће бити сачуван на подразумеваној локацији која пиче у пољу *Location*. Ако желите да промените локацију, то можете урадити притиском на три тачке десно од поља *Location*. По завршетку давања имена пројекту, притисните тастер *Next*. Отвара се нови дијалог (слика 8.3.) који од вас тражи да унесете податке о *FPGA* уређају који користите.



Сл. 8.3: Дијалог за креирање новог пројекта

За *Evaluation Development Board* изаберите *None Specified*, а за *Product Category* изаберите *All*. Компанија *Xilinx* производи неколико различитих породица *FPGA* развојне платформе, које се разликују по комплексности, цени, перформансама, потрошњи и осталим параметрима (*Artix, Spartan, Kintex, Virtex*). Развојна платформа коју користимо садржи *FPGA* коло *Spartan-3E*, тако да за *Family* је потребно ставити *Spartan3E*. На датој платформи, тачна ознака кола је *XC3S500E*, и то коло треба изабрати у пољу *Device*. У пољу *Package* се бира тип паковања кола, и обично се паковања разликују по броји физичких извода које коло поседује. Ово коло се налази у паковању *FG320*. Притиснути дугме *Next*, на следећем прозору још једном притиснути *Next*, и потом *Finish*. Овиме је креиран нови пројекат. Сада је потребно креирати датотеку у којој ће да се налази изворни код који описује бинарни бројач. На левој страни прозора у панелу *Hierarchy* (слика 8.4.) притиснути десни тастер миша на имену *FPGA* развојне платформе.



Сл. 8.4: Контекстни мени

Притиснути тастер *New Source*, и затим изабрати *Verilog Module*. У пољу *Name* уписати име жељеног модула (*Brojac*). Отвара се едитор у коме је могуће писати код.

8.1.3 Решење задатка и објашњење

У листингу испод је дат изворни код у *Verilog* језику за описивање хардвера.

```
1  module counter(clr, load, in, clk, enp, ent, out, rco);
2          input clr;
3          input load;
4          input[3:0] in;
5          input clk;
6          input enp;
7          input ent;
8          output[3:0] out;
9          reg[3:0] out;
10         output rco;
11
12         assign rco = &{out[3:0], ent};
13
14         always @(posedge clk)
15         begin
16                 if (~load)
17                         out <= in;
18                 else if (&{enp, ent})
19                         out <= out + 1'b1;
20                 else if (~clr)
21                         out <= 4'b0;
22         end
23 endmodule
```

Шта се овде дешава?
Прва линија кода служи за декларисање модула. Посматрајмо модуле у *Verilog* језику као интегрисана кола. Свако интегрисано коло има своје излазе и улазе, и може да се повезује са другим колима или физичким улазима и излазима. Сваки модул се декларише у посебној датотеци, и декларишу се на следећи начин:

module *име_модула*(*листа улаза и излаза одвојених зарезом*)**;**

После завршеног описивања понашања датог модула, ставља се кључна реч **endmodule**. Након декларисања модула, потребно је рећи шта су улази, а шта излази из модула. Улази у модул се дефинишу на следећи начин:

input *име_улаза* **;**

У четвртој линији се могу видети угласте заграде које садрже бројеве *3:0* у себи. То означава да је улаз низ од четири бита, где је први број бит највеће тежине, а други број је бит најмање тежине. Можемо то да посматрамо као да тај улаз има четири физичка извода (погледати слику 8.1, улазе *A,B,C,D*). Било који низ у језику *Verilog* се дефинише на тај начин. Постоје два главна типа излазних променљивих у верилогу:

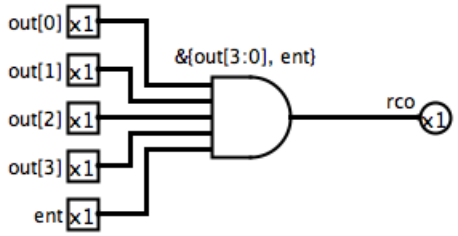
- net
- reg

Тип излазних променљивих *net* не поседује меморију, и понаша се као жица. Излаз овог типа ће увек бити једнак тренутној вредности улаза у њега. Најкоришћенији подтип је **wire**, који је основна жица. Када се декларише излаз са

output *име* **;**

подразумева се да је он типа *wire*. Тип података *reg* је нешто другачији. Он поседује

меморију, и он задржава у себи последњу вредност која је у њега унешена. Пошто се дефинише излазна променљива, потребно је да се нагласи да је она типа *reg* (видети линије 8 и 9). Да би биле јасније разлике између ова два типа, погледајмо линију 12. Променљива *RCO* је типа *Wire*. Кључна реч **assign** служи за додељивање вредности променљивим типа *Wire*. Променљива увек узима вредност израса са десне стране знака једнакости. Будући да је ово језик за описивање хардвера, једна од битних разлика у односу на програмске језике је што доста ствари могу да се дешавају истовремено. Тако и овде, било кад да се промени вредност са десне стране једнакости, променљива ће одмах имати ту вредност (за разлику од програмских језика где се инструкције извршавају секвенцијално, и променљива би попримила дату вредност тек када дође ред на њу да се изврши). Са десне стране једнакости се могу видети две новости, где је прва знак **&**. Овај симбол представља логичку операцију "И", и могуће је да се користи на два различита начина. Први начин јесте као бинарни оператор, у смислу *променљива1* **&** *променљива2*. Када би обе променљиве биле низ, резултат би био низ где је n-ти члан резултат логичке операције између n-тог члана прве променљиве и n-тог члана друге променљиве. Други начин је као редукциони оператор. У том случају синтакса је **&***променљива*. Подразумева се да је променљива низ, где се сваки члан низа може посматрати као један улаз у логичко коло "И". Како смо онда дали две променљиве (низ *out[3:0]* и променљиву *ent*) као аргументе редукционог оператора који прихвата само један низ као аргумент? Ту ступају на сцену велике заграде **{ }** које се називају концентратори. Између њих се наводи листа променљивих, одвојена зарезима, а концентратори од свих тих промељивих праве један низ. Шематски еквивалент дванаесте линије изведен са логичким колом "И" је дат на слици 8.5.



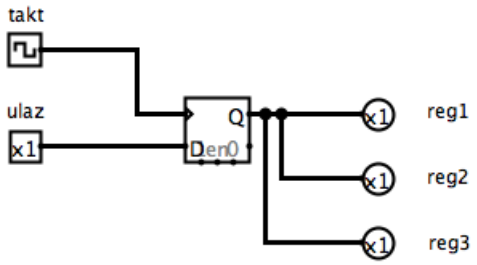
Сл. 8.5: Дванаеста линија кода изведена као шема

Излазне променљиве типа **reg** се понашају као физички *D* флип-флопови, што имплицира да се податак у њима чува само када се доведе неки сигнал на извод за омогућавање уписа података. Сходно томе, подаци у излазној променљивој типа *reg* се не могу уписивати командом **assign**. Блок наредби који се користи за уписивање и манипулацију података над излазним *reg* променљивама је дат на линији 14. Команда **always @(***списак сигнала на које треба да се реагује***)**. Сигнали су улазне променљиве на које треба да се реагује. Могуће је бити више сигнала на које треба да се реагује, одвојени су кључном речју **or**. У том случају, када се неки услов испуни, могу се уписати вредности у регистар. Кључна реч **posedge** говори да систем треба да реагује када дође позитивна ивица сигнала, супротно томе, постоји и кључна реч **negedge** где се реагује при негативној ивици сигнала. Списак сигнала се највише користи код комбинаторне логике, где било који сигнал да се промени, резултат ће да се промени. Реаговање на ивицу се користи код секвенцијалне логике, где постоји такт, и где се вредности уписују у регистар у правилним интервалима (када се појави сигнал такта). Све акције које треба да се десе после доласка сигнала се налазе унутар блока оивиченог са командама **begin** и **end**. Из поставке задатка смо видели да постоји неколико услова који када се испуне, дешавају се различите ствари. У *Verilog*-у, услове пишемо као у већини програмских језика, са командом **if**. Синтакса је

следећа:
if (*услов*) *команда*;
else *команда ако услов није испуњен*;
или, у случају да има више команди које треба да се изврше ако се услов (не)испуни:
if (*услов*) **begin**
команда1;
команда2;
end
Услови могу да садрже неке од стандардних логичких оператора, као што је једнакост (*променљива1* == *променљива2*), компарација (*променљива1* > *променљива2* или *променљива1* < *променљива2*) и неједнакост (*променљива1* !== *променљива2*). Из текста задатка видимо да у бројач може да се запише произвољна четворобитна вредност ако је извод *LOAD* спуштен на логичку нулу, те нам први услов (линија 16) управо то и говори. Будући да се услови извршавају када су тачни, а *LOAD* је активан када је на логичкој нули (вредност - нетачно), потребно је извшити инверзију, што се ради тако што се испред променљиве (*LOAD*, у овом случају) поставља тилда(~). Када се овај услов испуни, потребно је напунити регистар са улазним подацима. Постоје две методе за записивање података у регистре. Прва метода је блокирајуће писање (*излазниРегистар* = *променљива*). Ова метода ради на следећи начин: прво се израчуна израз који се налази са десне стране једнакости, и онда се одмах додели излазном регистру, потом се прелази на нов израз (ако га има) и ради се исто. Друга метода је неблокирајуће писање (*излазниРегистар* <= *променљива*) и суштинска разлика у односу на прву методу се огледа у томе да се код сваког неблокирајућег израза прво израчуна десна страна једнакости, и тек на крају циклуса се додељују ведности излазним регистрима. Покажимо то на једном примеру. Рецимо да имамо три регистра, *reg1*, *reg2* и *reg3*. Регистар *reg1* на сваком такту узима вредност улаза, *reg2* узима стару вредност првог регистра, а трећи регистар узима стару вредност другог регистра (тробитни померачки регистар). Ако бисмо то урадили са блокирајућим уписивањем на следећи начин:

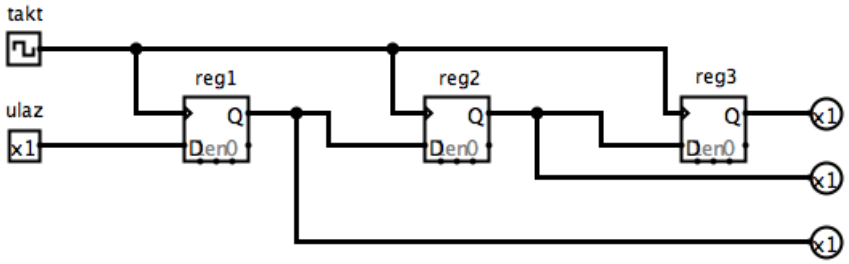
```
1  module pomerackiRegistar(reg1, reg2, reg3, ulaz, takt);
2          output reg reg1;
3          output reg reg2;
4          output reg reg3;
5          input ulaz;
6          input takt;
7          always @(posedge takt)
8          begin
9                  reg1 = ulaz;
10                 reg2 = reg1;
11                 reg3 = reg2;
12         end
13 endmodule
```

Дешава се следећа ствар: Рецимо да је тренутни улаз једнак логичкој нули. Први регистар у себе записује логичку нулу. Други регистар у себе записује вредност првог, који је нула, и трећи записује вредност другог који је такође нула. Тиме добијамо ситуацију да било који улаз да се доведе у први регистар, и остала два регистра дају на излазу исту вредност као и први (слика 8.6.).



Сл. 8.6: Погрешан начин за дизајнирање померачког регистра

Ако бисмо горњи код променили, тако да уместо блокирајућег ставимо неблокирајуће писање, резултат би био следећи. Претпоставимо да на првом такту на улазу имамо логичку јединицу. Памтимо ту вредност, и прелазимо на десету линију која каже да у други регистар треба да се упише садржај првог. Памтимо и то, и прелазимо на једанаесту линију која каже да садржај другог регистра ставимо у трећи. После једанаесте линије се излази из блока. По изласку из блока у трећи регистар се записује запамћена вредност другог (0), у други запамћена вредност првог (0) и у први се записује улаз (1). На излазу, дакле, имамо секвенцу *reg1* = 1, *reg2* = 0, *reg3* = 0. У следећем такту се доводи на улаз логичка нула. Понавља се горњи поступак, и сада имамо излаз *reg1* = 0, *reg2* = 1, *reg3* = 0, и тако се поступак понавља *ad infinitum*. Шематски еквивалент исправног начина се може видети на слици 8.7. Најлакши начин на који може да се закључи који од ове две врсте доделе вредности користити је то да треба да се користи блокирајуће додељивање у случају да моделујемо понашање комбинаторне логике, а неблокирајуће додељивање када моделујемо понашање секвенцијалне логике. Будући да померачки регистар спада у секвенцијалну логику, исправно је користити неблокирајуће додељивање вредности регистру.



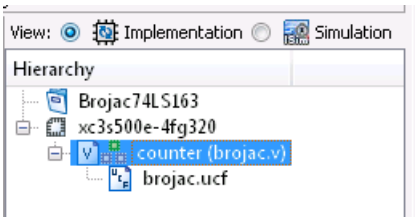
Сл. 8.7: Исправан начин за дизајнирање померачког регистра

Ако извод *LOAD* није на логичкој нули, прелазимо на следећи услов, који посматра да ли су изводи *ENT* и *ENP* на логичкој јединици. Из поставке се види да бројач броји када су ова два извода на логичкој јединици. Бројање можемо да дефинишемо као додавање јединице та тренутну вредност четворобитног регистра. То нам говори линија 19. Израз 1'b1 говори да треба да додамо једнобитну цифру која је логичка јединица. Први број говори дужину речи (битова), потом следи апостроф, и затим тип података који користимо (**b** - бинарни, **h** хексдецимални), после чега следи сама вредност. Ако, на пример, желимо у неки регистар да упишемо вредност 5 као бинарну вредност, то би урадили са командом 3'**b**101. Лева цифра је цифра највеће тежине. Последњи услов јесте, да када се извод *CLR* спусти на логичку нулу, у регистар треба да се упишу нуле.

8.1.4 Тестирање кола

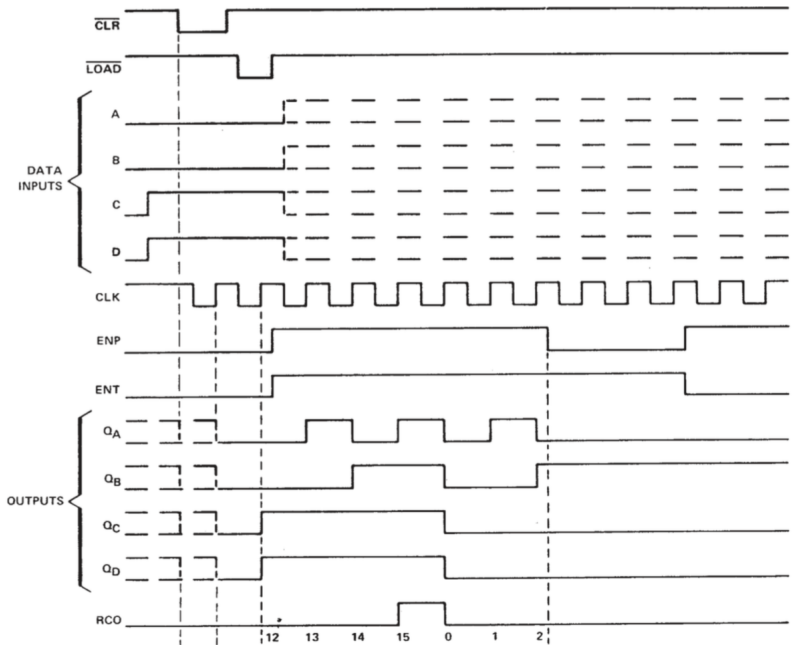
После дефинисања понашања кола, потребно је истестирати да ли се коло понаша

онако како желимо. *Xilinx ISE WebPack* долази са симулатором *ISim* за симулирање понашања кола. Да би се коло тестирало, потребно је направити тест окружење. У тест окружењу се дефинише на ком временском интервалу и кој сигнал треба дати колу, и онда се преко анализатора сигнала може пратити временски дијаграм. Поступак за прављење тест окружења и тестирање кола је следећи: Притиснути дугме *Simulation* у пољу *Design* (слика 8.8.).



Сл. 8.8: Поље за бирање између имплементације и симулације

Потом, потребно је направити нови документ, истим поступком као што смо направили главни *Verilog* документ. Уместо *Verilog Module*, сада бирамо *Verilog test fixture*. Назвати документ *test*, притиснути два пута *next*, и затим *finish*. Новонаправљени документ ће у себи већ садржати основна подешавања, као што су иницијализација модула који се тестира, и улази и излази из тог модула. Потребно је дизајнирати тест којим ће се утврдити исправност дизајна. У случају овог бројача, могуће је узети сигнални дијаграм који даје произвођач [19], и проверити да ли ће наше коло да се понаша на исти начин. Дијаграм је дат на слици 8.9.



Сл. 8.9: Сигнални дијаграм логичког кола *74LS163*

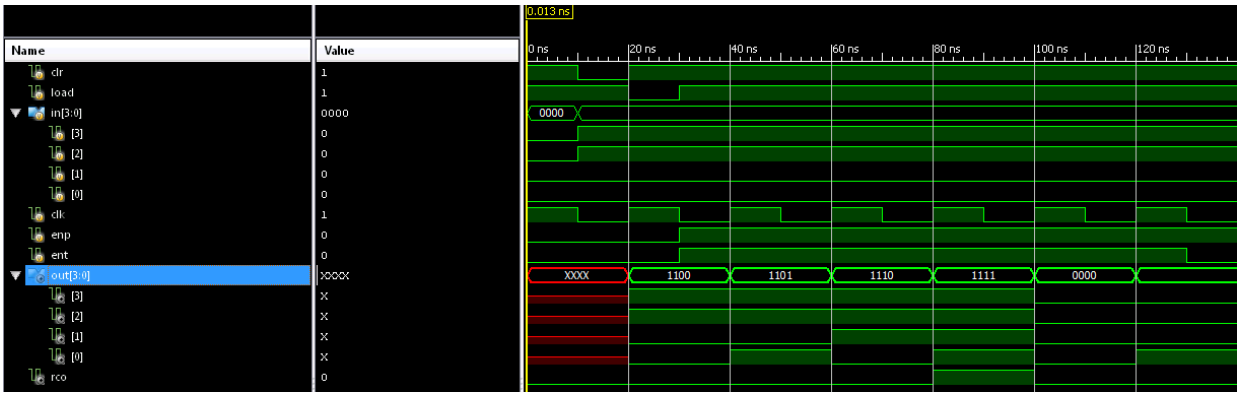
Са слике 8.9 можемо да видимо неколико битних података, које ћемо да реплицирамо у тесту кола. У првом такту пинове **CLR** и **LOAD** стављамо на логичку јединицу. У следећем такту, спуштамо извод **CLR**, и тиме бришемо сав садржај бројача. У следећем такту, враћамо **CLR** на логичку јединицу и исто време, на улазе *C* и *D* стављамо логичку јединицу. Потом, спуштамо извод *LOAD* на логичку нулу, и тиме уписујемо почетну вредност бројача у регистар. Враћамо *LOAD* на логичку јединицу, и изводе **ENT** и **ENP**

стављамо на логичку јединицу, и тиме активирамо бројач. После неког времена, спуштамо изводе **ENT** и **ENP**, и тиме прекидамо бројање. Листинг теста је дат испод.

```
1  `timescale 10ns / 1ps
2
3
4  module test;
5
6          // Ulazi
7          reg clr;
8          reg load;
9          reg [3:0] in;
10         reg clk;
11         reg enp;
12         reg ent;
13
14         // Izlazi
15         wire [3:0] out;
16         wire rco;
17
18         // Inicijalizacija modula
19         counter uut (
20                 .clr(clr),
21                 .load(load),
22                 .in(in),
23                 .clk(clk),
24                 .enp(enp),
25                 .ent(ent),
26                 .out(out),
27                 .rco(rco)
28         );
29
30         initial begin
31                 // Inicijalizacija ulaza
32                 clr = 1;
33                 load = 1;
34                 in = 0;
35                 clk = 1;
36                 enp = 0;
37                 ent = 0;
38
39                 // Pauza 10 ns
40                 #1;
41                 clr = 0;
42                 in = 4'b1100;
43                 #1;
44                 clr = 1;
45                 load = 0;
46                 #1;
47                 load = 1;
48                 ent = 1;
```

```
49                  enp = 1;
50                  #10;
51                  ent = 0;
52          end
53      always
54                  #1 clk = ~clk;
55  endmodule
```

Прва линија кода говори да је временски корак теста 10 наносекунди, а резолуција једна пикосекунда. Програм је аутоматски направио тест модул, одредио илазе и излазе и иницијализовао бројач. Наредба **initial** говори да следи блок команди које се извршавају једном по старту симулације. У почетном тренутку стављамо улазе на вредности који су описани изнад листинга. Команда $\#n$ говори да је потребно направити паузу дужине n временских тренутака (у нашем случају, један временски тренутак је 10 *ns*). Блок **always** се извршава све време док симулација траје. У њему смо описали сигнал такта, који на сваки временски корак мења вредност између јединице и нуле. Потребно је напоменути да се у *Verilog* језику ствари извршавају паралелно, те ће блок **always** бити активан и док још траје извршавање блока **initial**. Притиском десног тастера миша на *Simulate behavioural model* и потом на дугме *Run* покрећемо симулацију. Резултат се може витети на слици 8.10.



Сл. 8.10: Резултат симулације логичког кола *74LS163*

Поређењем слике 8.10 и слике 8.9 може се видети да се резултати симулације поклапају са дијаграмом који је дао произвођач.

8.1.5 Програмирање *FPGA* развојне платформе и тестирање помоћу WEB лабораторије

После тестирања нашег дизајна, време је да се исти упрограмира у *FPGA* уређај. Премда су улази и излази нашег дизајна дефинисани, не постоји ништа што говори компајлеру на које физичке изводе развојне платформе треба да доведе или да слуша сигнал. Тај проблем се решава текстуалним описивањем где се свака променљива која је улаз или излаз мапира на физички извод чипа. Таква текстуална датотека се зове *Universal Constraint File*, или скраћено *UCF*. Постоји више типова ограничења, а за овај случај се користе само ограничења имплементације. Ограничења имплементације су ограничења код којих је могуће мапирати локалну променљиву (улаз или излаз) на физички извод развојне платформе (**LOC**) и ограничења код којих је могуће управљати тајмингом компоненти у уређају (**PERIOD**) [20]. У овом случају, потребно је само мапирање извода. Потребно је прећи назад у поље за имплементацију (дугме *Implementation*), и потом

направити датотеку са ограничењима (*New Source -> Implementation Constraints File*). На удаљеној лабораторији, изводи који мењају прекидаче и тастере су прикључени преко *PMod* конектора на *Arduino*. За прекидаче је коришћен први конектор (JA1-JA10), а за тастере други (JB1-JB4). У табели 8.1 су дати физички изводи и одговарајуће компоненте [12].

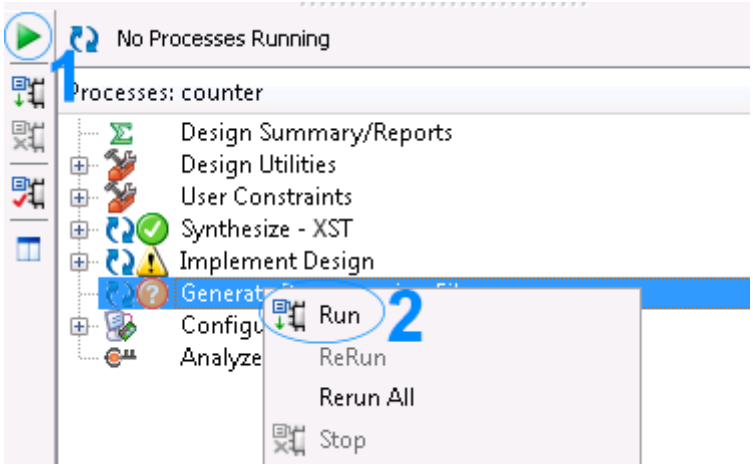
Табела 8.1: Изводи за *LE* диоде и прекидаче

Компонента	Извод	Компонента	Извод
Led0	j14	Btn2	r15
Led1	j15	Btn3	t17
Led2	k15	Sw0	l15
Led3	k14	Sw1	k12
Led4	e17	Sw2	l17
Led5	p15	Sw3	m15
Led6	f4	Sw4	k13
Led7	r4	Sw5	l16
Btn0	m13	Sw6	m14
Btn1	r18	Sw7	m16

За излазе ћемо користити од прве четири *LE* диоде, и за **RCO** пету *LE* диоду. Прва четири прекидача ће бити за уношење броја у бројач, пети прекидач за **CLR** сигнал, шести за, седми и осми за **LOAD**, **ENP** и **ENT**, респективно. На први тастер је доведен сигнал такта. Листинг *UCF* датотеке је дат испод.

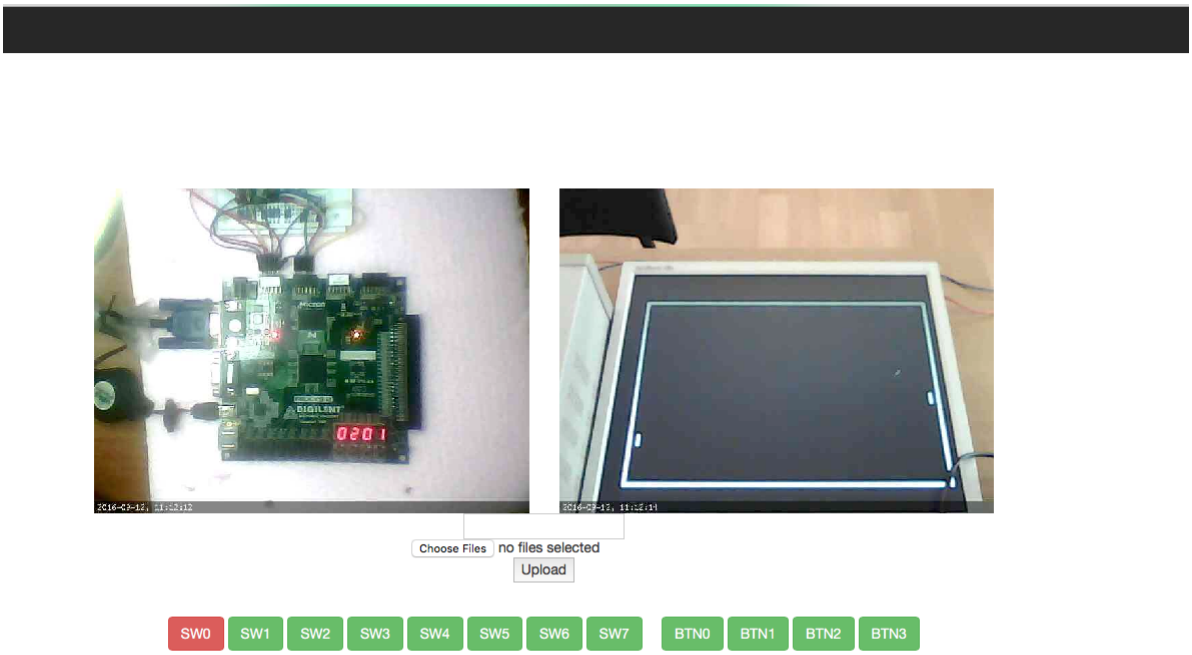
```
net "out[0]" loc = "j14";
net "out[1]" loc = "j15";
net "out[2]" loc = "k15";
net "out[3]" loc = "k14";
net "rco"    loc = "e17";
net "in[0]"  loc = "l15";
net "in[1]"  loc = "k12";
net "in[2]"  loc = "l17";
net "in[3]"  loc = "m15";
net "clr"    loc = "k13";
net "load"   loc = "l16";
net "enp"    loc = "m14";
net "ent"    loc = "m16";
net "takt"   CLOCK_DEDICATED_ROUTE = TRUE;
net "takt"   loc = "m13" ;
```

Кључна реч **NET** праћена именом променљиве прослеђује компајлеру коју променљиву хоћемо да повежемо, док **LOC** прослеђује компајлеру на коју физичку локацију хоћемо да је повежемо. Кључна реч **CLOCK_DEDICATED_ROUTE** говори да се тактни сигнал не узима са интерног такта, већ се доводи споља. Постављајући променљиву на **TRUE** омогућавамо то. Треба имати у виду да се ово ради само у специјалним случајевима, и није добра пракса да се користи. Када смо све дефинисали, потребно је извршити синтезу (компајлирање) дизајна, и исти упрограмирати у *FPGA* уређај. Да би се извршила синтеза дизајна, потребно је притиснути зелени троуглић (1, сл. 8.11), и по завршетку синтезе потребно је притиснути десни тастер миша на пољу *Generate programming file* и притиснути *Run* (2, сл. 8.11).



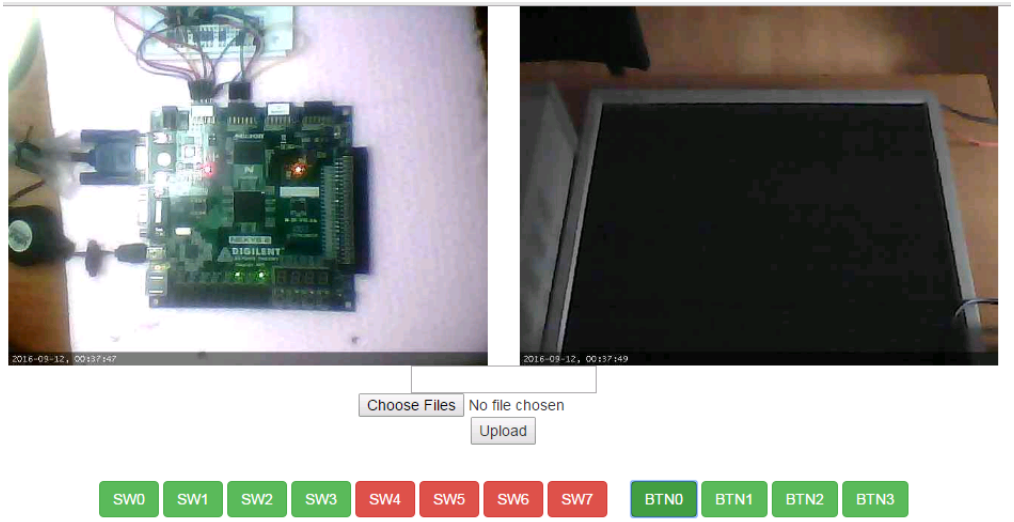
Сл. 8.11: Синтеза дизајна и генерисање датотеке за програмирање *FPGA*

По завршетку процеса синтезе и генерисања датотеке за програмирање, потребно је исту окачити на сервер удаљене лабораторије. Приликом уласка у *FPGA* удаљену лабораторију, појавиће се **WEB** страна дата на слици 8.12.



Сл. 8.12: Почетни екран удаљене лабораторије

Притиснути тастер *Choose files*, и потом изабрати датотеку са наставком *.bit* која се налази у фолдеру где је сачуван пројекат. Потом, притиснути тастер *Upload*. За неколико тренутака, уређај ће бити испрограмиран, и моћиће да се контролише. Да би се бројање започело, потребно је да изводи **ENP**, **ENP**, **CLR** и **LOAD** буду на логичкој јединици. То су прекидачи **SW4**, **SW5**, **SW6**, **SW7**. Када се дати тастери притисну, бројач се повећава за један сваки пут када наиђе сигнал такта **BTN0**. На слици 8.13 је приказан број 9.



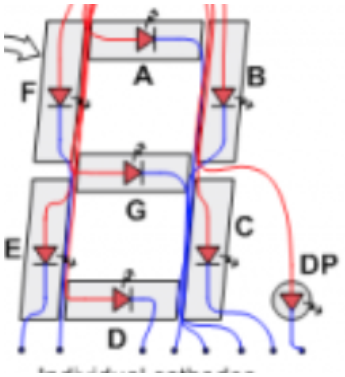
Сл. 8.13: Приказивање броја 9 на *LE* диодама *FPGA* развојне платформе

8.2 Задатак 2

8.2.1 Поставка и основни подаци

а) Написати *Verilog* модул који бинарни број унет на прекидачима конвертује у хексдецимални број који се приказује на седмосегментном екрану
б) Спојити направљени модул са модулом направљеним у претходном задатку у целину, тако да бинарни бројач показује тренутни број на седмосегментном екрану

У овом задатку је потребно наћи шта треба послати седмосегментном екрану да би се добила хексдецимална представа бинарног броја. Из [12] се може добити распоред извода појединачних сегмената седмосегментног екрана (сл. 8.14). У табели 8.2 могуће је видети одговарајуће изводе са *FPGA* развојне платформе за сваки сегмент.



Сл. 8.14: Изводи седмосегментног екрана

Табела 8.2: Сегменти дисплеја и одговарајући изводи

Сегмент	Извод
A	l18
B	f18
C	d17
D	d16
E	g14
F	j17
G	h14
DP	c17

Будући да на развојној плочи постоје четири седмосегментна екрана, а контролишу се преко истих извода, постоје одређени изводи (*F17, H17, C18, F15*) који служе за омогућавање приказа броја на одређеном седмосегментном екрану (активан је екран чији је извод спуштен на логичку нулу). *LE* диоде у седмосегментном екрану су повезане тако да им је заједничка анода, а спуштањем напона на катоди се одређени сегмент искључује. У задатку **a)** четворобитна цифра ће се уносити помоћу прва четири прекидача, док ће се на седмосегментни екран исписивати њена вредност у хексдецималном систему притиском на први тастер.

8.2.2 Решење задатка и објашњење

У листингу испод је дат изорни код првог дела задатка.

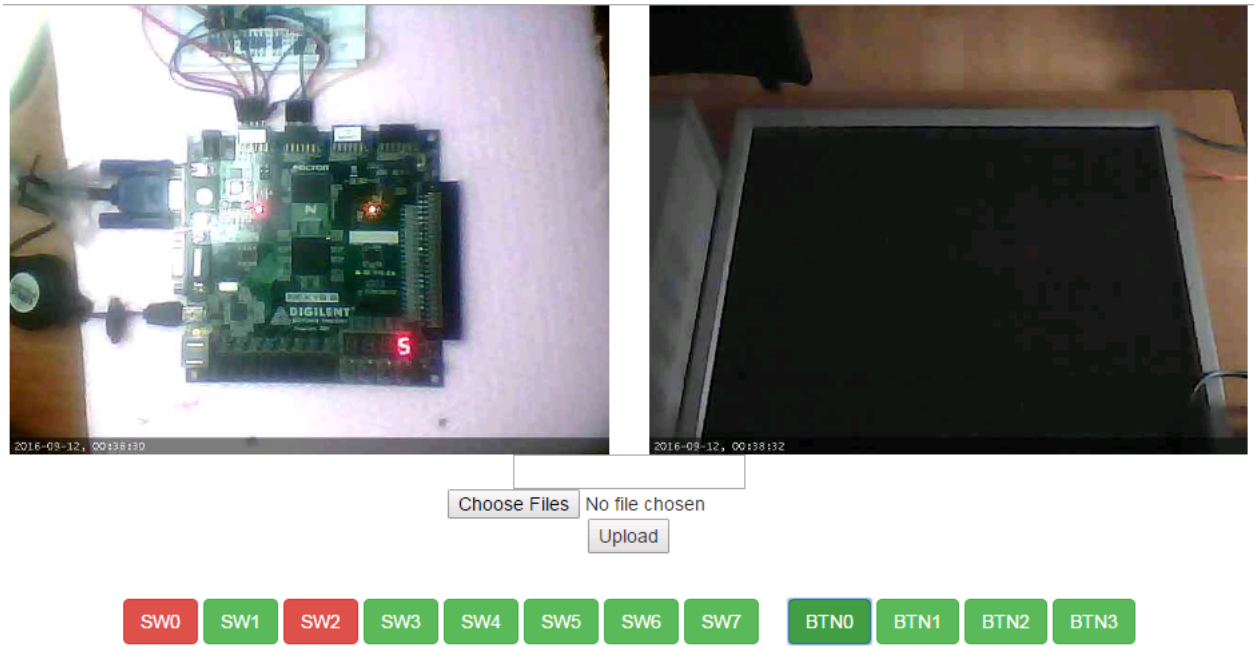
```
1  module displej(takt, cifra, disp, cifre);
2          input [3:0] cifra;
3          input takt;
4          output [6:0] disp;
5          reg [6:0] disp;
6          output wire[3:0] cifre;
7          parameter bnula          = 7'b100_0000;
8          parameter bjedan         = 7'b111_1001;
9          parameter bdva           = 7'b010_0100;
10         parameter btri           = 7'b011_0000;
11         parameter bcetiri        = 7'b001_1001;
12         parameter bpet           = 7'b001_0010;
13         parameter bsest          = 7'b000_0010;
14         parameter bsedam         = 7'b111_1000;
15         parameter bosam          = 7'b000_0000;
16         parameter bdevet         = 7'b001_0000;
17         parameter bcrta          = 7'b011_1111;
18         parameter ba             = 7'b000_1000;
19         parameter bb             = 7'b000_0011;
20         parameter bc             = 7'b100_0110;
21         parameter bd             = 7'b010_0001;
22         parameter be             = 7'b000_0110;
23         parameter bf             = 7'b000_1110;
24         assign cifre = 4'b1110;
25                 always @(posedge takt)
26                 case (cifra)
```

```
27                          0:      disp = bnula;
28                          1:      disp = bjedan;
29                          2:      disp = bdva;
30                          3:      disp = btri;
31                          4:      disp = bcetiri;
32                          5:      disp = bpet;
33                          6:      disp = bsest;
34                          7:      disp = bsedam;
35                          8:      disp = bosam;
36                          9:      disp = bdevet;
37                          10:     disp = ba;
38                          11:     disp = bb;
39                          12:     disp = bc;
40                          13:     disp = bd;
41                          14:     disp = be;
42                          15:     disp = bf;
43                          default: disp = bcrta;
44                  endcase
45  endmodule
```

У овом листингу се могу запазити неке нове кључне речи. Прва нова кључна реч је **parameter**, што у *Verilog* језику за описивање хардвера означава константу. Константе се не могу мењати у коду, и оне служе да замене записивање бројева који се често користе, зарад читљивости. Константе се пишу на следећи начин:
parameter *Име_константе* = *вредност*;.
Друга нова конструкција јесте **case** исказ. Он је исказ вишеструког гранања, и за разлику од исказа **if** он може да садржи више услова. У овом случају када је променљива *cifra* вредности било ког броја који се налази у **case** блоку, извршава се наредба која иде после двотачке. Ако променљива није једнака ни једној бројној вредности са леве стране, извршава се исказ **default**. Из приложеног кода се може видети да су услови у **case** исказу писани у декадном бројевном систему, јер *Verilog* аутоматски пребацује из бинарног у декадни или хексдецимални систем, у зависности од потребе. Датотека са ограничењима има облик:

```
1  NET "disp[0]"   LOC      = L18;
2  NET "disp[1]"   LOC      = F18;
3  NET "disp[2]"   LOC      = D17;
4  NET "disp[3]"   LOC      = D16;
5  NET "disp[4]"   LOC      = G14;
6  NET "disp[5]"   LOC      = J17;
7  NET "disp[6]"   LOC      = H14;
8  NET "cifre[0]"  LOC      = F17;
9  NET "cifre[1]"  LOC      = H17;
10 NET "cifre[2]"  LOC      = C18;
11 NET "cifre[3]"  LOC      = F15;
12 NET "cifra[0]"  LOC      = L15;
13 NET "cifra[1]"  LOC      = K12;
14 NET "cifra[2]"  LOC      = L17;
15 NET "cifra[3]"  LOC      = M15;
16 NET "takt"      CLOCK_DEDICATED_ROUTE = TRUE;
17 NET "takt"      LOC      = M13;
```

После синтезе дизајна, и после спуштања дизајна на *FPGA* уређај, могуће је исти контролисати посредством **WEB** лабораторије. На слици 8.15. приказан је резултат са приказаним бројем 5.



Сл. 8.15: Приказ резултата на **WEB** лабораторији

Други део задатка је повезивање претходно направљених модула (задатак 1 и 2) у једну целину. Будући да су модули већ направљени, потребно их је дорадити да би радили као једна целина. По креирању новог пројекта, десним тастером миша отворити мени у *Hierarchy* прозору, и потом притиснути *Add source*. Учитати модул из првог задатка (*brojac.v*). Потом, на исти начин, учитати модул из првог дела другог задатка. Будући да је главни модул бројач, притиском десног тастера миша на њега, у менију селектовати *Set as Top module*. Будући да је декодер имплементиран у комбинационој логици, из њега је могуће избацити сигнал такта. Такође, укључивање екрана ћемо померити у главни модул. У главном модулу је потребно иницијализовати декодер. Главни модул сада има облик:

```
module counter(clr, load, in, takt, enp, ent, out, rco, sseg, cifre
    );
        input clr;
        input load;
        input[3:0] in;
        input takt;
        input enp;
        input ent;
        output[3:0] out;
        reg[3:0] out;
        output rco;
        output[6:0] disp;
        output[3:0] cifre;
        assign rco = &{out[3:0], ent};

        assign cifre = 4'b1110;
        always @(posedge takt)
```

```
18          begin
19                  if (~load)
20                          out <= in;
21                  else if (&{enp, ent})
22                          out <= out + 1'b1;
23                  else if (~clr)
24                          out <= 4'b0;
25          end
26          displej a(out, disp);
27  endmodule
```

За аргументе главног модула, потребно је навести све улазе и излазе, а будући да сада користимо и екран, потребно је увести и излазе за исти (*sseg*). У 26. линији кода се иницијализује декодер, где се за улаз узима излаз бинарног бројача, а за излаз се узимају регистри који су повезани са седмосегментним екраном. Модул декодера сада изгледа:

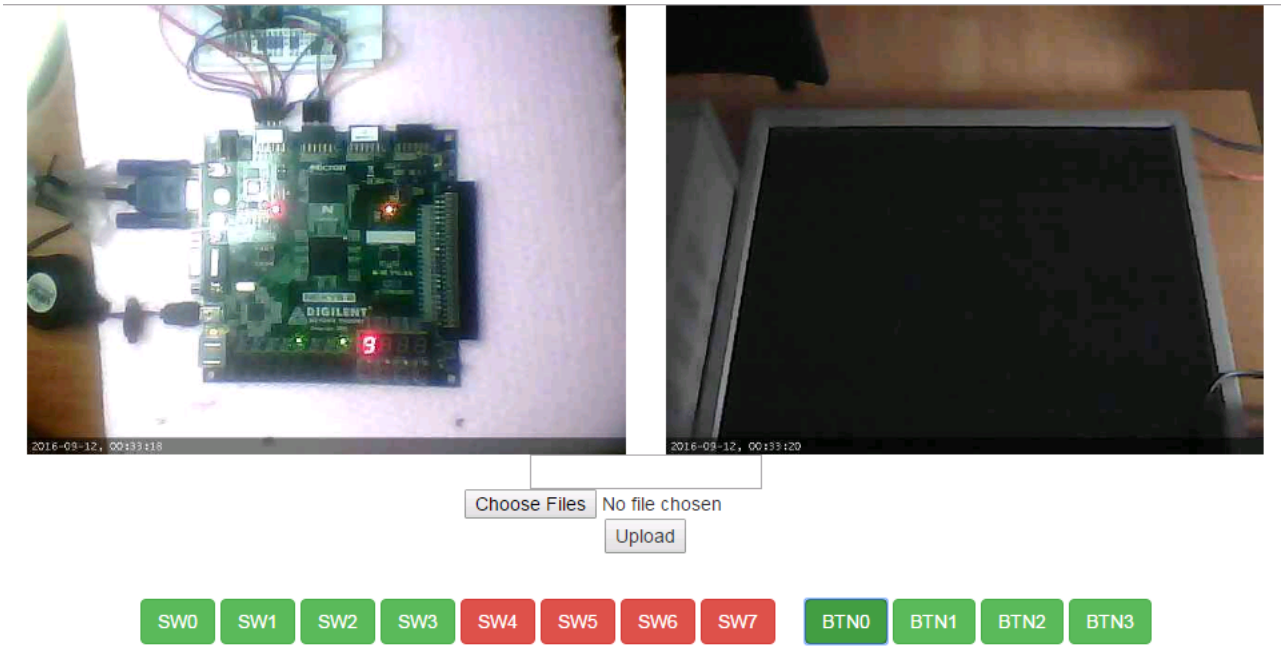
```
1   module displej(cifra, disp);
2           input [3:0] cifra;
3           output [6:0] disp;
4           reg [6:0] disp;
5           parameter bnula         = 7'b100_0000;
6           parameter bjedan        = 7'b111_1001;
7           parameter bdva          = 7'b010_0100;
8           parameter btri          = 7'b011_0000;
9           parameter bcetiri       = 7'b001_1001;
10          parameter bpet          = 7'b001_0010;
11          parameter bsest         = 7'b000_0010;
12          parameter bsedam        = 7'b111_1000;
13          parameter bosam         = 7'b000_0000;
14          parameter bdevet        = 7'b001_0000;
15          parameter bcrta         = 7'b011_1111;
16          parameter ba            = 7'b000_1000;
17          parameter bb            = 7'b000_0011;
18          parameter bc            = 7'b100_0110;
19          parameter bd            = 7'b010_0001;
20          parameter be            = 7'b000_0110;
21          parameter bf            = 7'b000_1110;
22                  always @(cifra)
23                  case (cifra)
24                          0:      disp = bnula;
25                          1:      disp = bjedan;
26                          2:      disp = bdva;
27                          3:      disp = btri;
28                          4:      disp = bcetiri;
29                          5:      disp = bpet;
30                          6:      disp = bsest;
31                          7:      disp = bsedam;
32                          8:      disp = bosam;
33                          9:      disp = bdevet;
34                          10:     disp = ba;
35                          11:     disp = bb;
```

```
36                          12:     disp = bc;
37                          13:     disp = bd;
38                          14:     disp = be;
39                          15:     disp = bf;
40                          default:disp = bcrta;
41                  endcase
42  endmodule
```

Будући да сада излази иду и на *LE* диоде и на дисплеј, потребно је да датотека са ограничењима садржи оба. Облик датотеке је сада:

```
1   net "out[0]"    loc     = "j14";
2   net "out[1]"    loc     = "j15";
3   net "out[2]"    loc     = "k15";
4   net "out[3]"    loc     = "k14";
5   net "rco"       loc     = "e17";
6   net "in[0]"     loc     = "l15";
7   net "in[1]"     loc     = "k12";
8   net "in[2]"     loc     = "l17";
9   net "in[3]"     loc     = "m15";
10  net "clr"       loc     = "k13";
11  net "load"      loc     = "l16";
12  net "enp"       loc     = "m14";
13  net "ent"       loc     = "m16";
14  net "takt"      CLOCK_DEDICATED_ROUTE = TRUE;
15  net "takt"      loc     = "m13" ;
16  NET "disp[0]"   LOC     = L18;
17  NET "disp[1]"   LOC     = F18;
18  NET "disp[2]"   LOC     = D17;
19  NET "disp[3]"   LOC     = D16;
20  NET "disp[4]"   LOC     = G14;
21  NET "disp[5]"   LOC     = J17;
22  NET "disp[6]"   LOC     = H14;
23  NET "cifre[0]"  LOC     = F17;
24  NET "cifre[1]"  LOC     = H17;
25  NET "cifre[2]"  LOC     = C18;
26  NET "cifre[3]"  LOC     = F15;
```

По синтези и покретању дизајна на **WEB** лабораторији, могуће је утврдити исправност дизајна. На слици 8.16. је приказан бројач са бинарним приказом броја девет на *LE* диодама, и екран који показује исти број у хексдецималном систему.



Сл. 8.16: Приказ резултата на **WEB** лабораторији

8.3 Задатак 3

8.3.1 Поставка задатка

Задатак је имплементирати аларм за аутомобил користећи машину стања. Систем се аутоматски укључује пошто возач искључи аутомобил, изађе из возила (возачева врата се отворе, па затворе) и време T_ARM_DELAY прође. Ако је сувозач у аутомобилу, и оба врата су отворена, систем се укључује након што се сва врата затворе и време T_ARM_DELAY прође. Активација се рестартује ако се врата отворе пре него што време T_ARM_DELAY прође. Ако се врата отворе после активације аларма, аларм почиње одбројавање. Ако се возило не упали пре истека интервала T_DRIVER_DELAY, оглашава се сирена. Сирена остаје укључена све док су врата отворена, и по затварању још T_ALARM_ON времена. После тог времена, аларм искључује сирену и враћа се у активирано стање. Ако се сувозачева врата прва отворе, одбројавање траје $T\_PASSENGER\_DELAY$ времена. На инструмент табли возила се налази LE диода која има функцију индикатора статуса аларма. Ако је систем активан, LE диода трепери са периодом од две секунде. Ако је одбројавање почело, или се огласила сирена, LE диода сија константно. Као додатна мера заштите, постављена је блокада на пумпи за гориво. Када се аутомобил угаси, блокада се активира. Блокада се деактивира само у случају када се у исто време држи скривени прекидач и да контакт за паљење возила.
Времена кашњења се базирају на четири параметара изражена у секундама: времена између изласка из аутомобила и активирања аларма T_ARM_DELAY, дужине одбројавања при отварању возачевих врата T_DRIVER_DELAY, дужине одбројавања при отварању сувозачевих врата $T\_PASSENGER\_DELAY$ и дужине трајања сирене T_ALARM_ON. У табели 3 су дате подразумеване вредности сваког од наведених параметара, али сваки параметар се може наместити по жељи користећи сигнале $Time\_Parameter\_Selector$, $Time\_Value$ и $Reprogram$. $Time\_Parameter\_Selector$ наводи који параметар треба променити, $Time\_Value$ чине четири прекидача која представљају време (0-15 s), а тастер $Reprogram$ служи за чување подешавања. Имена параметара, основне вредности, и бројеви параметара су дати у табели 8.3.

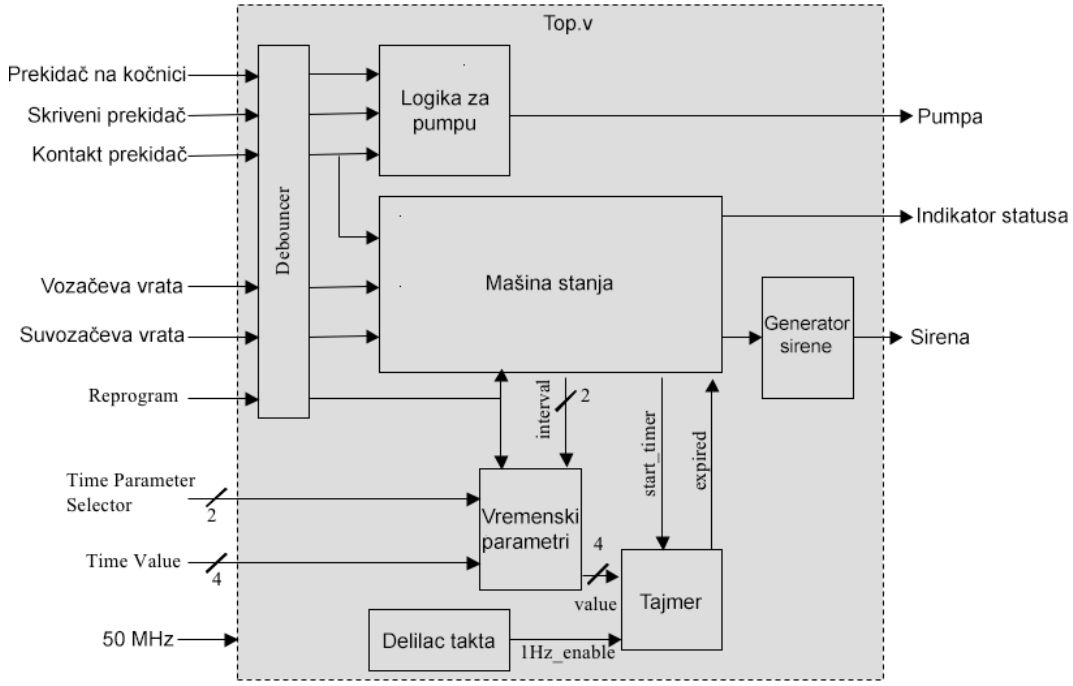
Табела 8.4: Списак улаза и излаза

Улаз/Излаз	**Физички извод**
Скривени прекидач	sw[6]
Прекидач кочнице	btn2
Врата возач	btn0
Врата сувозач	btn1
Контакт	sw7
Селектор параметра	sw[5:4]
Време	sw[3:0]
Репрограм	btn3
Пумпа	Led[1]
Статус	Led[0]
Сирена	Led[2]

Табела 8.3: Параметри времена

Интервал	Ознака	Параметар	Време	Бин. број
Активационо кашњење	*T_ARM_DELAY*	00	6	0110
Одбројавање, возач	*T_DRIVER_DELAY*	01	8	1000
Одбројавање, сувозач	*T_PASSENGER_DELAY*	10	15	1111
Време трајања сирене	*T_ALARM_ON*	11	10	1010

На слици 8.17. је приказан дијаграм повезивања модула који ће се правити у овом задатку.



Сл. 8.17: Блок дијаграм алармног система

У табели 8.4 се налази списак улаза и излаза који ће бити коришћени.

Будући да систем ради са унутрашњим сигналом такта, сваки улаз који корисник притисне се на систем шаље асинхроно. У тренутку када *FPGA* прими сигнал, могуће је

да се он још није попео на логичку јединицу, или спустио на логичку нулу, већ се напон налази негде између. То може да изазове метастабилност у регистрима који примају сигнал. Метастабилност је феномен где излаз регистра није ни на јединици ни на нули, већ плута између те две вредности. Такво стање може да изазове проблеме унутар развојне платформе. Решење јесте да се спољни, асинхрони сигнал синхронизује са тактом који се користи унутар развојне платформе. То се може постићи са каскадним везивањем два регистра, где се у првом такту, спољни сигнал записује у први. Излаз првог регистра може бити метастабилан, али у други се прима тек по наиласку на нови тактни сигнал. У том тренутку, излаз првог се стабилизује и прослеђује другом регистру [21].Други проблем који се може десити у реалности јесте учестано мењање вредности улазног сигнала приликом притиска на тастер/прекидач. То се дешава због несавршеног контакта у прекидачу, тако да приликом притиска он узастопно прекида сигнал. После одређеног времена, сигнал се смирује. То се може превазићи тако што, када се прекидач прочита, направи се одређено кашњење, и поново се прочита (треба напоменути да се други случај не може десити на **WEB** лабораторији, јер на *FPGA* нису прикључени физички прекидачи). Модул **Debouncer** служи да реши оба горе наведена проблема.
Модул **Vremenski_parametri** чува вредности временских параметара. Понаша се као меморија са четири локације која се иницијализује са подразумеваним вредностима приликом укључивања система, али оне се могу изменити у било ком тренутку од стране корисника. Користећи двобитни **Interval** сигнал, главна машина стања бира један од четири параметара и шаље их модулу **Timer**. Када год се неки параметар репрограмира, машина стања би требало да се врати у стање **ARMED**, након чега се одмах враћа у одговарајуће стање у зависности од улаза.
Модул **Delilac takta** служи да обезбеди такт фреквенције 1 Hz, који се добија дељењем такта осцилатора *FPGA* развојне платформе фреквенције 50 MHz.
Модул **Mašina stanja** представља машину коначних стања која се понаша као секвенцер за цео систем. Систем има четири главна мода управљања:

- **ARMED** Индикатор статуса треба да трепери у интервалу од две секунде, сирена је искључена. Ако се прекидач за контакт укључи, прећи у стање **DISARMED**, у супротном, ако су врата отворена, покренути прикладно одбројавање (у зависности да ли су отворена сувозачева или возачева врата) и прећи у стање **TRIGGERED**. Ово је почетно стање по укључивању система.

- **TRIGGERED** Индикатор статуса је константно укључен, сирена је искључена. Ако је прекидач за контакт укључен, отићи у стање **DISARMED**. Ако се одбројавање заврши пре укључивања контакта, отићи у стање **SOUND ALARM**.

- **SOUND ALARM** Индикатор статуса и сирена треба да буду константно укључени. Потребно је да се сирена оглашава **T_ALARM_OFF** секунди пошто се сва врата затворе (после чега треба да се пређе у стање **ARMED**) или док се не укључи контакт прекидач (тада се прелази у **DISARMED** стање).

- **DISARMED** Индикатор статуса и сирена су искључени. Чекати док се контакт прекидач не искључи, возачева врата не буду отворена па затворена, и после **T_ARM_DELAY** прећи на **ARMED** стање.

Модул **Logika za pumpu** поседује једноставну машину стања која управља пумпом за гориво. Пумпа се искључује када се прекине контакт прекидач, а поново се укључује тек када се укључи контакт прекидач, а скривени прекидач и кочница се притисну у исто време.

Модул **Generator sirene** генерише сигнал за сирену. Будући да на **WEB** лабораторији не постоји звук, уместо звучника ће се користити *LE* диода. У овом модулу ће се генерисати *PWM* сигнал који ће утицати на интензитет светла.

8.3.2 Модул *Debounce*

У следећем листингу је дат изворни код модула **Debounce**

```
1  module Debounce (reset, clock, noisy, clean);
2  input reset;
3  input clock;
4  input noisy;
5  output reg clean;
6  parameter DELAY=500000
7  reg [18:0] count;
8  reg new;
9
10 always @(posedge clock)
11      if (reset)
12      begin
13          count <= 0;
14          new <= noisy;
15          clean <= new;
16      end
17      else if (noisy != new)
18      begin
19          new <= noisy;
20          count <= 0;
21      end
22      else if (count == DELAY)
23          clean <= new;
24      else
25          count <= count+1;
26
27 endmodule
```

Константа **DELAY** са вредношћу 500 000 служи за поређење са тренутном вредношћу бројача. Када бројач дође до 500 000, протекло је време од десет милисекунди ($50000000Hz/500000 = 100Hz$). Дакле, по окидању прекидача, чека се десет милисекунди док се вредност не упише у регистар, тиме избегавајући комешање вредности на излазу прекидача. Коришћењем два регистра решава се проблем метастабилности.

8.3.3 Модул *Vremenski parametri*

У листингу испод је дат изворни код модула **Vremenski parametri**

```
1  module vremenski_parametri(
2              input reprogram,
3              input [1:0] izbor_vremena,
4              input [3:0] vrednost_vremena,
5              input [1:0] interval,
6              input clk,
7              output reg [3:0] vrednost
```

```
8           );
9
10  // podrazumevane vrednosti
11  parameter T_ARM_DELAY_INIT=4'b0110;
12  parameter T_DRIVER_DELAY_INIT=4'b1000;
13  parameter T_PASSANGER_DELAY_INIT=4'b1111;
14  parameter T_ALARM_ON_INIT=4'b1010;
15
16  reg [3:0] T_ARM_DELAY;
17  reg [3:0] T_DRIVER_DELAY;
18  reg [3:0] T_PASSANGER_DELAY;
19  reg [3:0] T_ALARM_ON;
20
21  initial
22  begin
23      T_ARM_DELAY=T_ARM_DELAY_INIT;
24      T_DRIVER_DELAY=T_DRIVER_DELAY_INIT;
25      T_PASSANGER_DELAY=T_PASSANGER_DELAY_INIT;
26      T_ALARM_ON=T_ALARM_ON_INIT;
27  end
28
29  always @(posedge clk)
30  begin
31      case(interval)
32          2'b00: vrednost=T_ARM_DELAY;
33          2'b01: vrednost=T_DRIVER_DELAY;
34          2'b10: vrednost=T_PASSANGER_DELAY;
35          2'b11: vrednost=T_ALARM_ON;
36      endcase
37  end
38
39  always @(posedge reprogram)
40  begin
41      case(izbor_vremena)
42          2'b00: T_ARM_DELAY=vrednost_vremena;
43          2'b01: T_DRIVER_DELAY=vrednost_vremena;
44          2'b10: T_PASSANGER_DELAY=vrednost_vremena;
45          2'b11: T_ALARM_ON=vrednost_vremena;
46      endcase
47  end
48
49  endmodule
```

У овом модулу се сусрећемо са новом кључном речју у дизајну, а то је **initial**. У првом задатку, кључна реч **initial** је коришћена за блок наредби које се шаљу симулацији по покретању симулације. Она се може користити и у дизајну, и све што се налази у њеном блоку, извршава се на уређају по укључењу истог.

8.3.4 Модул *Delilac*

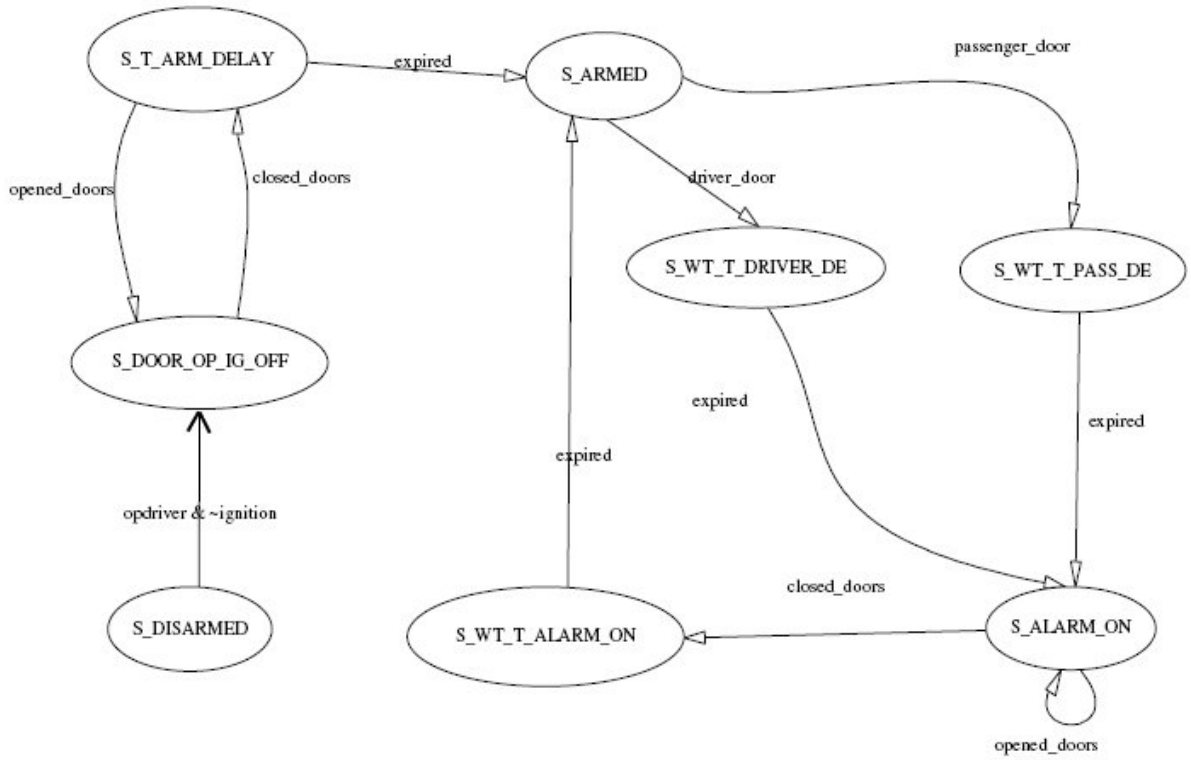
Изворни код овог модула је:

```
1  module divider (input clk,
2                  output hz_enable,
3                  input start_timer
4                 );
5
6  reg [26:0] brojac;
7
8  always@(posedge clk)
9  begin
10     if (start_timer)
11         brojac <= 0;
12     else if (brojac == 49_999_999)
13         brojac <= 0;
14     else
15         brojac <= brojac + 1;
16 end
17 assign hz_enable = (brojac == 49_999_999);
18 endmodule
```

Потребно је направити такт учестаности 1 Hz. Будући да је такт који се генерише на *FPGA* плочи учестаности 50 MHz, потребно је направити бројач који броји до 50 000 000, и тада пусти један импулс. У овом модулу то је решено тако што постоји двадесетшестобитни бројач ($2^{26} = 67108864$), који на сваких 50 000 000 откуцаја избацује један импулс.

8.3.5 Модул *Masina stanja*

У овом модулу је дефинисана главна логика система. Дијаграм стања система приказан је на слици 8.18. У табели 8.5 је дат опис сваког стања.



Сл. 8.18: Блок дијаграм алармног система

Табела 8.5: Опис стања

	Ознака стања	**Опис**	**Функција**
1	**S_T_ARM_DELAY**	Возач напушта ауто (затворена врата) и почиње одбројавање T_ARM_DELAY секунди. Ако се врата у међувремену отворе, одбројавање почиње поново. Када се одбројавање заврши, иде се у стање укљученог аларма S_ARMED.	**start_timer = 1** **period = T_ARM_DELAY** **sirena = 0** **status_ind = OFF**
2	**S_ARMED**	Означава да је аларм спреман. Ако се отворе врата, прелази се у одбројавање времена T_DRIVER_DELAY или T_PASSENGER_DELAY. Уколико се стави кључ у браву (контакт), бројање се прекида и систем је деактивиран.	**status_ind = BLINK** **sirena = 0**

3	**S_WT_T_DRIVER_DE**	Након отварања врата возача почиње одбројавање. Једино се прекида стартовањем мотора (контакт прекидач). Када се одбројавање заврши, активира се сирена.	**start_timer=1** **period=** **T_DRIVER_DELAY** **status_ind=ON** **sirena=0**
4	**S_WT_T_PASS_DE**	Након отварања врата сувозача почиње одбројавање.Једино се прекида стартовањем мотора(контакт прекидач). Када се одбројавање заврши, активира се сирена.	**start_timer=1** **period=** **T_PASSENGER_DELAY** **status_ind=OFF** **sirena=0**
5	**S_ALARM_ON**	Аларм је активиран и оглашава се сирена. Остаје активна док год се не затворе сва врата.	**sirena=1** **status_ind=ON**
6	**S_WT_T_ALARM_ON**	Када се сва врата затворе, сирена је активна још T_ALARM_ON времена. Када се бројање заврши, систем поново прелази у активно стање - S_ARMED	**start_timer=1** **sirena=1** **period=** **T_ALARM_ON** **status_ind=OFF**
7	**S_DISARMED**	У ово стање се прелази из свих осталих уколико се стартује мотор (контакт). То је иначе почетно стање. Када возач искључи мотор и отвори врата прелази се у стање провере да ли сва врата затворена.	**sirena=0** **status_ind=OFF**
8	**S_DOOR_OP_IG_OFF**	Ово стање служи као припрема за улазак у активно стање и у њега се прелази након што возач отвори врата и искључи мотор. Када се затворе сва врата, прелази се у стање за одбројавање периода T_ARM_DELAY	**sirena=0** **status_ind=OFF**

Изворни код машине стања је дат испод.

```
1  module alarm_KA(
2              input kontakt,
3              input vrata_vozaca,
4              input vrata_suvozaca,
5              input reprogram,
6              input reset_sync,
7              input isteklo,
8              input clk,
9              output reg [1:0] interval,
10             output reg Start_timer,
11             output reg sirena,
12             output reg [1:0] status_indikator, //00-OFF, 01-ON, 10 -
                   BLINK
13             output  [2:0] trenutno_stanje
14         );
15
16 parameter S_DISARMED                                          =0;
17 parameter S_DOOR_OPEN_IGNITION_OFF       =1;
18 parameter S_WAIT_T_ALARM_DELAY           =2;
19 parameter S_ARMED
               =3;
20 parameter S_WAIT_T_PASSANGER_DELAY       =4;
21 parameter S_WAIT_T_DRIVER_DELAY          =5;
22 parameter S_ALARM_ON                                          =6;
23 parameter S_WAIT_T_ALARM_ON                          =7;
24
25 reg [3:0] stanje, sledece_stanje;
26 reg [1:0] skip_one_clk;
27
28 always @(*)
29 begin
30     case(stanje)
31         S_DISARMED: //0
32         begin
33             //*********************************************
34             //Odredjivanje funkcionalnosti - signala
35             //*********************************************
36             Start_timer=0;
37             interval=2'bxx;
38             status_indikator=2'b00;
39             sirena=0;
40
41             //*********************************************
42             //Definisanje stanja
43             //*********************************************
44
45             if(kontakt)
46                 sledece_stanje=S_DISARMED;
47             else if(reprogram)
48                 sledece_stanje=S_ARMED;
```

```
49              else if(vrata_vozaca & ~kontakt)
50                  sledece_stanje=S_DOOR_OPEN_IGNITION_OFF;
51              else
52                  sledece_stanje=S_DISARMED;
53
54          end
55          S_DOOR_OPEN_IGNITION_OFF: //1
56          begin
57              //***********************************************
58              //Odredjivanje funkcionalnosti - signala
59              //***********************************************
60              Start_timer=0;
61              interval=2'bxx;
62              status_indikator=2'b00;
63              sirena=0;
64
65              //***********************************************
66              //Definisanje stanja
67              //***********************************************
68              if(kontakt)
69                  sledece_stanje=S_DISARMED;
70              else if(reprogram)
71                  sledece_stanje=S_ARMED;
72              else
73              begin
74                  if(~vrata_vozaca & ~vrata_suvozaca)
75                  begin
76                      sledece_stanje=S_WAIT_T_ALARM_DELAY;
77                  end
78                  else
79                      sledece_stanje=S_DOOR_OPEN_IGNITION_OFF;
80              end
81          end
82          S_WAIT_T_ALARM_DELAY: //
83          begin
84              //***********************************************
85              //Odredjivanje funkcionalnosti - signala
86              //***********************************************
87              Start_timer=1;
88              interval=2'b00;
89              status_indikator=2'b00;
90              sirena=0;
91
92              //***********************************************
93              //Definisanje stanja
94              //***********************************************
95
96              if(kontakt)
97                  sledece_stanje=S_DISARMED;
98              else if(reprogram)
```

```
99                  sledece_stanje=S_ARMED;
100             else if(isteklo)
101                 sledece_stanje=S_ARMED;
102             else if (vrata_vozaca | vrata_suvozaca)
103                 sledece_stanje=S_DOOR_OPEN_IGNITION_OFF;
104             else
105                 sledece_stanje=S_WAIT_T_ALARM_DELAY;
106
107         end
108         S_ARMED: //
109         begin
110             //*********************************************
111             //Odredjivanje funkcionalnosti - signala
112             //*********************************************
113             Start_timer=0;
114             interval=2'bxx;
115             status_indikator=2'b10;
116             sirena=0;
117
118             //*********************************************
119             //Definisanje stanja
120             //*********************************************
121
122             if(kontakt)
123                 sledece_stanje=S_DISARMED;
124             else if(reprogram)
125                 sledece_stanje=S_ARMED;
126             else if(vrata_suvozaca)
127                 sledece_stanje=S_WAIT_T_PASSANGER_DELAY;
128             else if (vrata_vozaca)
129                 sledece_stanje=S_WAIT_T_DRIVER_DELAY;
130             else
131                 sledece_stanje=S_ARMED;
132         end
133         S_WAIT_T_PASSANGER_DELAY: //4
134         begin
135             //*********************************************
136             //Odredjivanje funkcionalnosti - signala
137             //*********************************************
138             Start_timer=1;
139             interval=2'b10;
140             status_indikator=2'b01;
141             sirena=0;
142
143             //*********************************************
144             //Definisanje stanja
145             //*********************************************
146
147             if(kontakt)
148                 sledece_stanje=S_DISARMED;
```

```
149             else if(reprogram)
150                 sledece_stanje=S_ARMED;
151             else if (isteklo)
152                 sledece_stanje=S_ALARM_ON;
153             else
154                 sledece_stanje=S_WAIT_T_PASSANGER_DELAY;
155         end
156         S_WAIT_T_DRIVER_DELAY: //5
157         begin
158             //*********************************************
159             //Odredjivanje funkcionalnosti - signala
160             //*********************************************
161             Start_timer=1;
162             interval=2'b01;
163             status_indikator=2'b01;
164             sirena=0;
165 
166             //*********************************************
167             //Definisanje stanja
168             //*********************************************
169             if(kontakt)
170                 sledece_stanje=S_DISARMED;
171             else if(reprogram)
172                 sledece_stanje=S_ARMED;
173             else if (isteklo)
174                 sledece_stanje=S_ALARM_ON;
175             else
176                 sledece_stanje=S_WAIT_T_DRIVER_DELAY;
177         end
178         S_ALARM_ON: //6
179         begin
180             //*********************************************
181             //Odredjivanje funkcionalnosti - signala
182             //*********************************************
183             Start_timer=0;
184             interval=2'bxx;
185             status_indikator=2'b01;
186             sirena=1;
187 
188             //*********************************************
189             //Definisanje stanja
190             //*********************************************
191             if(kontakt)
192                 sledece_stanje=S_DISARMED;
193             else if(reprogram)
194                 sledece_stanje=S_ARMED;
195             else if(~vrata_vozaca & ~vrata_suvozaca)
196                 sledece_stanje=S_WAIT_T_ALARM_ON;
197             else
198                 sledece_stanje=S_ALARM_ON;
```

```
199
200             end
201             S_WAIT_T_ALARM_ON: //7
202             begin
203                 //*********************************************
204                 //Odredjivanje funkcionalnosti - signala
205                 //*********************************************
206                 Start_timer=1;
207                 interval=2'b11;
208                 status_indikator=2'b01;
209                 sirena=1;
210
211                 //*********************************************
212                 //Definisanje stanja
213                 //*********************************************
214                 if(kontakt)
215                     sledece_stanje=S_DISARMED;
216                 else if(reprogram)
217                     sledece_stanje=S_ARMED;
218                 else if(isteklo)
219                     sledece_stanje=S_ARMED;
220                 else
221                     sledece_stanje=S_WAIT_T_ALARM_ON;
222             end
223             default: sledece_stanje=S_DISARMED;
224         endcase
225
226 end
227
228 always @(posedge clk)
229 begin
230     if (reset_sync) stanje<=S_DISARMED;
231     else
232         stanje<=sledece_stanje;
233 end
234
235 assign trenutno_stanje=stanje;
236 endmodule
```

Машина стања је једноставна, стања се дефинишу преко **case** исказа, где у сваком исказу постоје услови где се прелази у ново стање ако се исти испуне. Следеће стање се памти у регистру **sledece_stanje**. На сваком такту, у регистар за тренутно стање се стављају подаци из регистра за следеће стање, и тако је обезбеђен прелаз (ако се стање не мења, у регистар **sledece_stanje** се једноставно стави тренутно стање).

8.3.6 Модул *Logika pumpe*

Овај модул представља једноставну машину коначних стања. Постоје два стања, где пумпа може бити или укључена или искључена. Приликом рестартовања модула, она је искључена. Испод се налази листинг овог модула.

```
1  module pumpa_goriva(
2              input clk,
3              input reset_sync,
4              input kocnica,
5              input skriveno_dugme,
6              input kontakt,
7              output reg napajanje_pumpe
8          );
9
10 reg stanje,sledece_stanje;
11 parameter S_PUMPA_RADI    =1'b1;
12 parameter S_PUMPA_NERADI =1'b0;
13 initial stanje=S_PUMPA_NERADI;
14
15 always@(*)
16 begin
17     if (reset_sync==1) begin
18         napajanje_pumpe=0;
19         sledece_stanje=S_PUMPA_NERADI;
20     end
21     else
22     begin
23         case (stanje)
24             S_PUMPA_RADI:begin
25                 napajanje_pumpe=1;
26                 if (~kontakt) sledece_stanje=S_PUMPA_NERADI;
27                 else sledece_stanje=S_PUMPA_RADI;
28             end
29
30             S_PUMPA_NERADI:begin
31                 napajanje_pumpe=0;
32                 if (kontakt & kocnica & skriveno_dugme)
                       sledece_stanje=S_PUMPA_RADI;
33                 else sledece_stanje=S_PUMPA_NERADI;
34
35             end
36             default : begin napajanje_pumpe=0;
37                 sledece_stanje=S_PUMPA_NERADI;
38             end
39         endcase
40     end
41 end
42
43 always @(posedge clk)
44 begin
45     if (reset_sync==1) stanje <=S_PUMPA_NERADI;
46     else
47         stanje<=sledece_stanje;
48 end
49 endmodule
```

8.3.7 Модул *Tajmer*

Овај модул је задужен за одбројавање једног од четири интервала, и по завршетку одбројавања, шаље сигнал машини стања. Изворни код је дат испод.

```
module timer(input clk,
             input clk1hz,
             input reset_sync,
             input [3:0] vrednost,
             output  isteklo,
             input start_timer,
             output reg brojanje,
             output reg [3:0] broj_ciklusa,
             output reg [3:0] brojac
            );

initial brojanje=0;

always@(posedge clk)
begin

    if (reset_sync)
        brojanje <= 0;
    else if (start_timer)
    begin
        broj_ciklusa<=vrednost;
        brojac <= 0;
        brojanje <= 1;
    end
    else if (brojanje && clk1hz)
    begin
        brojac <=brojac+ 1;
    end
    else if (isteklo)
        brojanje <= 0;
end

assign isteklo = (brojanje && (brojac == broj_ciklusa));

endmodule
```

Из услова **start_timer** могуће је ивући закључак да се бројач рестартује колико год трајао сигнал на том улазу. Да би се тај проблем решио, потребно је направити коло за генерисање импулса који траје тачно један такт. Коло је дато у наставку.

```
module signal_u_impuls(
            input in,
            input clk,
            output out
        );

reg q1,q2;

```

```
9  initial
10 begin
11      q1=0;
12      q2=0;
13 end
14
15 always @(posedge clk)
16 begin
17      q1<=in;
18      q2<=q1;
19 end
20
21 assign out=(q1 && ~q2);
22 endmodule
```

8.3.8 Модул *Status Indikator*

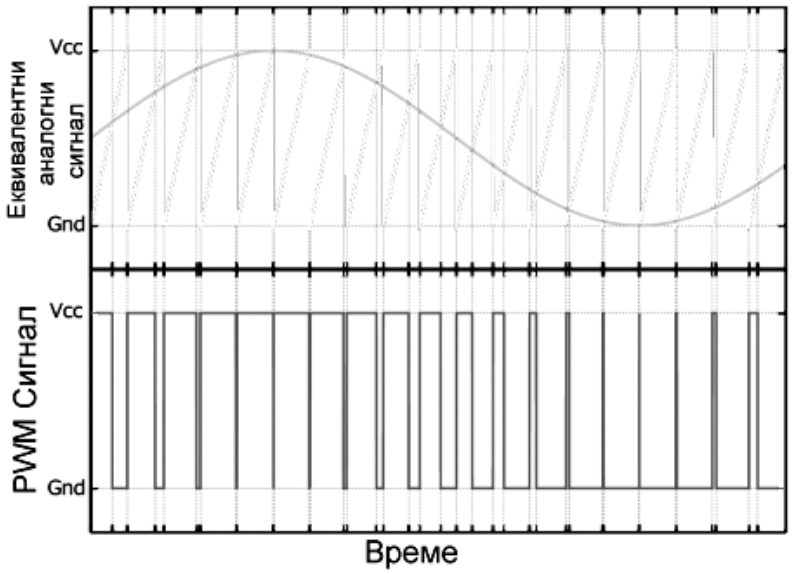
Овај једноставан модул је задужен за индикацију статуса. Будући да он контролише једну *LE* диоду, потребно је само дефинисати када је та диода укључена и искључена. Трећа могућност јесте да диода треба да трепери, то је решено трећим стањем које на сваки сигнал такта инвертује излаз ка диоди. Овај модул користи спор такт учестаности 1 Hz. Код овог модула је дат у наставку.

```
1  module prikazi_status(input clk,
2                        input [1:0] status,
3                        output reg led
4                       );
5
6  always@(posedge clk )
7  begin
8      case(status)
9          2'b00: led=0;
10         2'b01: led=1;
11         2'b10:
12         begin
13             if(led==1'bz)
14                 led=1;
15             led=~led;
16         end
17         default:led=0;
18     endcase
19
20 end
21
22 endmodule
```

У 13. линији кода могуће је видети вредност **z**. Поред бинарних вредности **1** и **0**, *Verilog* поседује још две вредности: **z** - висока импенданса (физички искључено) и **x** - неодређена вредност (метастабилност).

8.3.9 Модул *Sirena*

Сигнал сирене се генерише модулацијом ширине импулса. Модулација ширине импулса је техника контроле аналогних развојне платформе дигиталним системом, где контролни сигнал није аналогна вредност, већ серија импулса различитих ширина које одговарају жељеном аналогном сигналу. Што је импулс дужи, амплитуда еквивалентног аналогног сигнала је већа (слика 8.19).



Сл. 8.19: Поређење аналогног и *PWM* сигнала

PWM се лако генерише. Потребно је направити бројач (дужина бројача утиче на резолуцију излазног сигнала) и регистар где се чува жељена вредност који је исте ширине као и бројач. Бројач се на сваком такту повећава за један, и потребно је упоредити вредност бројача са жељеном вредношћу. Ако је вредност у бројачу мања или једнака жељеној вредности, на излаз се шаље логичка јединица. У супротном, шаље се логичка нула. Оваквом врстом управљања ћемо контролисати интензитет *LE* диоде. Резолуција бројача је осам бита, тако да је могуће контролисати интензитет са 256 различитих нивоа. Фреквенција *PWM*-а ће бити 1000 Hz, те је потребно направити делитељ фреквенције. Интензитет светла ће расти у корацима од по 10 десет пута у секунди, тако да је потребан још један делилац. Изворни код модула је дат испод.

```
1  module sirena(input enable, input clk, output reg LED);
2  reg[7:0] pwm;
3  reg[6:0] brojacIntenzitet;
4  reg[15:0] brojac;
5  reg[7:0] vrednost;
6  initial begin
7    vrednost = 0;
8    pwm = 0;
9    brojac = 0;
10   brojacIntenzitet = 0;
11 end
12 always @(posedge clk) begin
13   if (enable) begin
14     brojac <= brojac + 1;
15     if (brojac == 50_000) begin //1000 Hz PWM
```

```
16          brojac <= 0;
17          pwm <= pwm + 1;
18          brojacIntenzitet <= brojacIntenzitet + 1;
19          if (brojacIntenzitet == 100) begin//menjanje intenziteta
                svetla 10 puta u sekundi
20            vrednost <= vrednost + 10;
21            brojacIntenzitet <= 0;
22          end
23          if (pwm <= vrednost) LED <= 1;
24          else LED <= 0;
25        end
26      end
27      else LED <= 0;
28  end
```

8.3.10 Главни модул

У овом модулу се инстанцирају сви модули, и повезују се са одговарајућим променљивама. Овај модул ће бити главни модул пројекта. Развијени декодер у другом задатку се може позвати да би се на седмосегментном екрану исписивало тренутно стање. Комплетан код је дат испод.

```
1  module top(
2              input [7:0] sw,
3              input [3:0] btn,
4              input clk_50mhz,
5              output [7:0] led,
6              output dp,
7              output [3:0] an,
8              output [6:0] seg,
9              output Sirena
10         );
11
12 reg reset_sync =1'b0;
13
14 assign dp = 1;                                  // gasimo decimalnu tacku -
       ACTIVE LOW
15 assign an = 4'b1110;                  // samo prva cifra ce biti prikazana
       , ACTIVE LOW
16
17
18 //interne varijable
19 wire kocnica_Debounce;
20 wire skriveno_dugme_Debounce;
21 wire kontakt_Debounce;
22 wire vrata_vozaca_Debounce;
23 wire vrata_suvozaca_Debounce;
24 wire reprogram_Debounce;
25 wire Hz_enable;
26 wire Start_timer;
27 wire [1:0] interval;
```

```
28 wire [3:0] vrednost;
29 wire isteklo;
30 wire Start_timer_puls;
31
32 wire Siren;
33 wire [2:0] Watch_states;
34 wire [1:0] prikaz_status_indikatora;
35
36
37 //Radimo debounce za sve prekidace i tastere
38 Debounce Debounce_kocnica(.reset(reset_sync),.clock(clk_50mhz),.
       noisy(btn[2]),.clean(kocnica_Debounce));
39 Debounce Debounce_skriveni_prekidac(.reset(reset_sync),.clock(
       clk_50mhz),.noisy(sw[6]),.clean(skriveno_dugme_Debounce));
40 Debounce Debounce_kontakt(.reset(reset_sync),.clock(clk_50mhz),.
       noisy(sw[7]),.clean(kontakt_Debounce));
41 Debounce Debounce_vrata_vozaca(.reset(reset_sync),.clock(clk_50mhz)
       ,.noisy(btn[0]),.clean(vrata_vozaca_Debounce));
42 Debounce Debounce_vrata_suvozaca(.reset(reset_sync),.clock(clk_50mhz
       ),.noisy(btn[1]),.clean(vrata_suvozaca_Debounce));
43 Debounce Debounce_reprogram(.reset(reset_sync),.clock(clk_50mhz),.
       noisy(btn[3]),.clean(reprogram_Debounce));
44
45
46 pumpa_goriva pumpa_goriva(
47                     .clk(clk_50mhz),
48                     .kocnica(kocnica_Debounce),
49                     .skriveno_dugme(skriveno_dugme_Debounce),
50                     .kontakt(kontakt_Debounce),
51                     .napajanje_pumpe(led[1]));
52
53 divider divider(
54                .clk(clk_50mhz),
55                .hz_enable(Hz_enable),
56                .start_timer(Start_timer_puls));
57
58 vremenski_parametri vremenski_parametri_1(
59                             .reprogram(reprogram_Debounce),
60                             .izbor_vremena(sw[5:4]),
61                             .vrednost_vremena(sw[3:0]),
62                             .interval(interval),
63                             .clk(clk_50mhz),
64                             .vrednost(vrednost));
65
66 //                Osnovni modul - konacni automat koji obradjuje
       vecinu ulaznih i izlaznih signala
67
68 alarm_KA alarm_KA(
69                .kontakt(kontakt_Debounce),
70                .vrata_vozaca(vrata_vozaca_Debounce),
```

```
71                 .vrata_suvozaca(vrata_suvozaca_Debounce),
72                 .reprogram(reprogram_Debounce),
73                 .reset_sync(reset_sync),
74                 .isteklo(isteklo),
75                 .clk(clk_50mhz),
76                 .interval(interval),
77                 .Start_timer(Start_timer),
78                 .sirena(Siren),
79                 .status_indikator(prikaz_status_indikatora),
80                 .trenutno_stanje(Watch_states));
81
82 timer timer_1(
83             .clk(clk_50mhz),
84             .clk1hz(Hz_enable),
85             .reset_sync(reset_sync),
86             .vrednost(vrednost),
87             .isteklo(isteklo),
88             .start_timer(Start_timer_puls));
89
90 sirena sirena_1(
91              .enable(Siren),
92              .clk(clk_50mhz),
93              .LED(led[2]));
94 //
95 //                start_timer      mora biti kratak, samo impuls koji
       ce startovati
96 //                brojanje. Na ovaj nacin se dobija kratak impuls
97 signal_u_impuls  signal_u_impuls(
98                       .in(Start_timer),
99                       .clk(clk_50mhz),
100                      .out(Start_timer_puls));
101
102 //                pokazivanje stanja preko diode
103 prikazi_status prikazi_status_1(
104                    .clk(Hz_enable),
105                    .status(prikaz_status_indikatora),
106                    .led(led[0]));
107
108 //       Prikaz trenutnog stanja na 7 segmentnom pokazivacu
109 hex7segment displej(.sw({1'b0,Watch_states}),.sseg(seg));
110
111 endmodule
```

8.3.11 Датотека са ограничењима

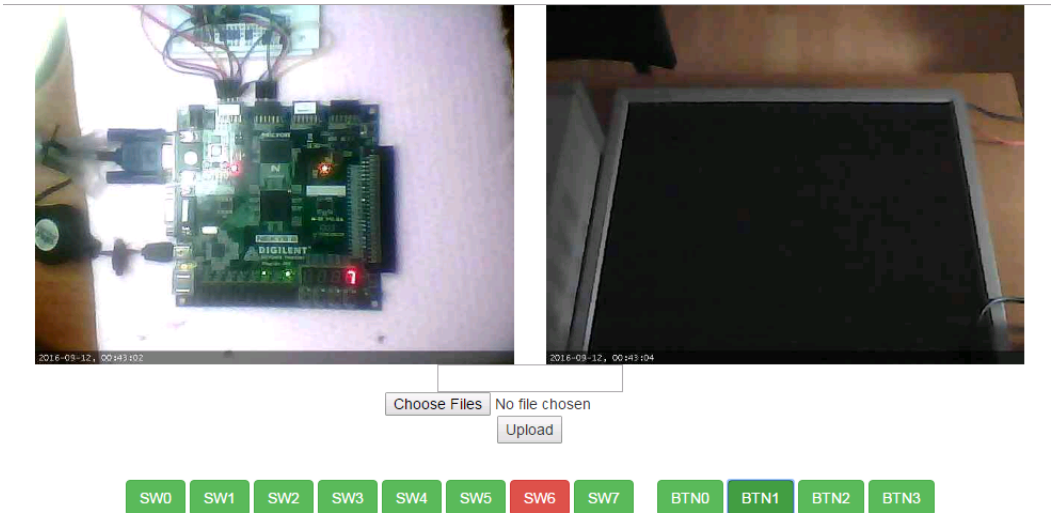
Датотека са мапирањем извода (*UCF*) је дата у листингу испод. Једини извод који до сада није коришћен јесте генератор такта који се налази на плочи. Његова локација је **B8**.

```
1 NET "clk_50mhz" LOC="B8";
2 NET "LED<0>" LOC="J14";
```

```
3  NET "LED<1>" LOC="J15";
4  NET "LED<2>" LOC="K15";
5  NET "sw<0>" LOC = "l15";
6  NET "sw<1>" LOC = "k12";
7  NET "sw<2>" LOC = "l17";
8  NET "sw<3>" LOC = "m15";
9  NET "sw<4>" LOC = "k13";
10 NET "sw<5>" LOC = "l16";
11 NET "sw<6>" LOC = "m14";
12 NET "sw<7>" LOC = "m16";
13 NET "dp" LOC="C17";
14 NET "an<0>" LOC="F17";
15 NET "an<1>" LOC="H17";
16 NET "an<2>" LOC="C18";
17 NET "an<3>" LOC="F15";
18 NET "seg<0>" LOC = "L18";
19 NET "seg<1>" LOC = "F18";
20 NET "seg<2>" LOC = "D17";
21 NET "seg<3>" LOC = "D16";
22 NET "seg<4>" LOC = "G14";
23 NET "seg<5>" LOC = "J17";
24 NET "seg<6>" LOC = "H14";
25 NET "btn<0>" LOC = "m13";
26 NET "btn<1>" LOC = "r18";
27 NET "btn<2>" LOC = "r15";
28 NET "btn<3>" LOC = "t17";
```

8.3.12 Резултати

Када се изврши синтеза дизајна, и спусти се на развојну плочу, потребно је проверити функционалност целог дизајна. Треба запамтити да машина стања има укупно седам стања, и тренутно стање је приказано на седмосегментном екрану. На слици 8.20. је дат приказ стања где је укључена сирена. На екрану се види тачан број стања, и такође се види ефекат управљања модулацијом ширине импулса (трећа *LE* диода са десне стране).



Сл. 8.20: Резултат трећег задатка

9 Закључак

Дипломски рад је настао као последица ангажмана на *SCOPES* пројекту IЗ74З0_160454 / 1: "Enabling Web-based Remote Laboratory Community and Infrastructure" of Swiss National Science Foundation и рада на проблематици **WEB** лабораторија у периоду од јуна 2015. године.

Циљеви рада су били:

- Да се омогући лабораторијска настава путем Интернета у области програмирања и примене *FPGA*
- Да се допринесе развоју наставних лабораторијских материјала на српском језику предмета *MIT 6.111 Introductory Digital Systems Laboratory* који је намењен оспособљавању студената за примену *FPGA* развојне платформе
- Да се омогући примена наставне методологије предмета *MIT 6.111 Introductory Digital Systems Laboratory* у пуном капацитету (да се превазиђе проблем недоступности хардверске и софтверске подршке за обуку студената и других корисника)
- Да се допринесе општем софтверском решењу **WEB** лабораторије која се развија на Универзитету у Крагујевцу у оквиру *SCOPES* пројекта IЗ74З0_160454 / 1: "Enabling Web-based Remote Laboratory Community and Infrastructure" of Swiss National Science Foundation

У раду је представљен портал, објашњење његове функционалности, дато упутство за примену, представљена опремљеност релевантних страна **WEB** лабораторије за програмирање и примену *FPGA* развојне платформе, и тај део кључних задатака које треба урадити је детаљно представљен са решењима.

Такође, рад је понудио ново техничко решење софтвера за **WEB** лабораторију.

10 Литература

[1] Ruben Heradio, Luis de la Torre, Daniel Galan, Francisco Javier Cabrerizo, Enrique Herrera-Viedma, and Sebastian Dormido. Virtual and remote labs in education: A bibliometric analysis. *Computers & Education*, 98:14–38, 2016.

[2] Mohammad Reza Farrokhnia and Ayoub Esmailpour. A study on the impact of real, virtual and comprehensive experimenting on students' conceptual understanding of dc electric circuits and their skills in undergraduate electricity laboratory. *Procedia-Social and Behavioral Sciences*, 2(2):5474–5482, 2010.

[3] Jesus García-Guzmán, Farah H Villa-L-pez, Francisco H Silva-Del-Rosario, Alfredo Ramírez-Ramírez, Jorge V élez Enríquez, and Ervin J Álvarez-Sánchez. Virtual environment for remote access and automation of an ac motor in a web-based laboratory. *Procedia Technology*, 3:224–234, 2012.

[4] M. Abdulwahed and Z.K. Nagy. Systematic evaluation of the use of remote and virtual laboratories in engineering education. *21th European Symposium on Computer aided Process engineering - ESCAPE21*, 2011.

[5] Milan S Matijević, Vladimir M Cvjetković, Vojislav Ž Filipović, and Nikola D Jović. Basic concepts of automation and mechatronics with LEGO mindstorms NXT. *Tehnika*, 69(4):653–660, 2014.

[6] Radomir Mitrović, Nikola Jović, Vladimir Cvjetković, Miloš Nedeljković, and Milan Matijević. Internet Mediated Laboratories in Engineering Education. *Proc. of XXII Conference on Development Trends: "New Technologies in Teaching", TREND 2016*, February 16-19 2016.

[7] Никола Јовић. *Web Laboratorije, http://cpa.fin.kg.ac.rs/index*, 2016.

[8] Judy Tsai. Design and Implementation of an Online Laboratory for Introductory Digital Systems. Master's thesis, Massachusetts Institute of Technology, August 2005.

[9] Umer Farooq, Zied Marrakchi, and Habib Mehrez. *Tree-based Heterogeneous FPGA Architectures: Application Specific Exploration and Optimization*. Springer Science & Business Media, 2012.

[10] Arduino. Arduino Leonardo, *https://www.arduino.cc/en/main/arduinoboardleonardo*, 2016.

[11] Arduino. Arduino IDE, *https://www.arduino.cc/en/main/software*, 2016.

[12] Inc. Digilent. Digilent Nexys2 reference manual, *https://reference.digilentinc.com/nexys/nexys2/refmanual*, 2016.

[13] 6.111 Introductory Digital Systems Laboratory, *http://web.mit.edu/6.111/www/f2010/*, MIT, Fall 2010., 2010.

[14] Драган Лазић, Милан Матијевић, Миладин Стефановић, Милан Ристановић, Владимир Цвијетковић, and Милан Ерић. *Техничко решење "Софтвер за управљање web лабораторијама"*, 2010.

[15] Go-lab, *http://golabz.eu*, 2016.

[16] Stojanović Vladimir and Hom Gim. *Introductory Digital Systems Laboratory*. Massachusetts Institute of Technology: MIT OpenCourseWare, http://ocw.mit.edu License: Creative Commons BY-NC-SA, Fall 2010.

[17] Ненад Бабајић. *Решења са образложењем домаћих задатака и лабораторијских вежби предмета Увод у пројектовање и контролу интегрисаних кола за комуникације, сензоре и актуаторе, http://moodle.mfkg.rs/course/view.php?id=250*. Факултет инжењерских наука, Универзитет у Крагујевцу, 2011.

[18] Владимир М. Стојановић. *Увод у пројектовање и контролу интегрисаних кола за комуникације, сензоре и актуаторе*. Moodle портал Факултета инжењерских наука Универзитета у Крагујевцу, школска 2010/2011 година.

[19] Texas Instruments. *74LS163 Synchronous 4-bit counter*, 3 1988. Rev. 1.

[20] Xilinx. *Constrains Guide*, 12 2009.

[21] Altera. Understanding metastability, July 2009.

11 Додатак

11.1 Изворни код микросервера

```
var express        =    require("express"),
    app            =    express(),
    ruta           =    "proba",
    dostupno       =    true,
    events         =    require("events"),
    wsevent        =    new events.EventEmitter(),
    formidable     =    require("formidable"),
    fs             =    require("fs"),
    io             =    require("socket.io")(),
    odbroj         =    undefined,
    odg            =    true,
    podesavanje    =    require('./proba.json'),
    vreme          =    podesavanje.vreme,
    clientTimeout  =    0,
    radi           =    false,
    proc           =    require('child_process'),
    child          =    undefined,
    cookieParser   =    require('cookie-parser'),
    bodyParser     =    require('body-parser'),
    session        =    require('express-session'),
    pwd            =    require('password-hash'),
    ime            =    undefined,
    indeks         =    undefined,
    arduino        =    false,
    sw0            =    undefined;
var five = require("johnny-five");
// Socket.IO komunikacija
var board = new five.Board();
var pinovi = [0,1,2,3,4,5,6,7,8,9,10,11,12];
console.log(podesavanje.vreme)
board.on("ready", function() {
  console.log('Arduino connected');
  sw = new five.Leds(pinovi);
  komunikacija();
}
);

var komunikacija = function(){
  app.set('view engine', 'ejs');
  app.use('/public',express.static(__dirname + '/public'));
  app.use(cookieParser());
  app.use(bodyParser.urlencoded({extended:true}));
  app.use(session({
        secret: 'WebLabL3L4ExperimentalnaPostavka2MIU',
        resave: true,
        saveUninitialized:true,
```

```
47          expires: new Date(Date.now() + 3600000)
48    }));
49
50
51    app.get('/home/:ime/:indeks',function(req,res){
52      if (!req.session.auth){
53        req.session.auth = true;
54        req.session.ime = req.params.ime;
55        req.session.indeks = req.params.indeks;
56      }
57      res.render('desc',require('./langs/srpski.json'));
58      delete require.cache[require.resolve('./langs/srpski.json')];
59    });
60
61    app.get('/lab/:id',function(req,res){
62      if (req.params.id == ruta){
63        res.sendFile(__dirname + "/index3.html");
64        wsevent.emit('zauzeto');
65      } else {
66        res.sendFile(__dirname + "/index2.html");
67      }
68    });
69    app.post('/lab/upload/:id',function(req,res){
70      if (req.params.id == ruta){
71
72        var form = new formidable.IncomingForm();
73        form.uploadDir = __dirname + '/script/';
74        form.parse(req, function(err, fields, files) {
75          res.sendFile(__dirname + "/index3.html");
76          odb = true;
77          fs.unlinkSync(form.uploadDir  + "program.bit")
78          fs.renameSync(files.upload.path, form.uploadDir  + "program.
              bit");
79          if (radi){
80            child.kill();
81            radi = false;
82          };
83          ime = req.session.ime;
84          indeks = req.session.indeks;
85          child = proc.spawn('run.bat');
86          radi = 1;
87          child.stdout.on('data',function(data){
88            console.log(data.toString('utf8'));
89          });
90          child.stderr.on('data', function(data){
91            console.log(data.toString('utf8'));
92          });
93        });
94      } else {
95        res.sendFile(__dirname + "/index2.html");
```

```
96      }
97    });
98
99
100   //Unutrasnji eventovi u serveru
101
102   wsevent.on('kraj', function(){
103     vreme = podesavanje.vreme;
104     clearInterval(odbroj);
105     dostupno = true;
106     clientHomeNamespace.emit('slobodno',ruta);
107     clientLabNamespace.emit('kraj');
108   });
109   wsevent.on('zauzeto', function(){
110     dostupno = false;
111     if (odbroj){
112       clearInterval(odbroj);
113     };
114     odbroj = setInterval(tajmer, 1000);
115   })
116
117
118   //Startovanje servera
119
120   var server = app.listen(podesavanje.port,function(){
121       console.log(":) " + podesavanje.port);
122   });
123
124
125
126   io.listen(server);
127
128
129   var clientLabNamespace = io.of('/client-lab');
130   console.log('radim :)');
131   // tajmer
132   function tajmer(){
133     clientHomeNamespace.emit('zauzeto', vreme);
134     if (odg){
135       odg = false;
136       clientLabNamespace.emit('Jel si tu?');
137       clientTimeout = 0;
138     } else {
139       clientTimeout++;
140       if (clientTimeout == podesavanje.vreme) {
141         wsevent.emit('kraj');
142         clientTimeout = 0;
143       }
144     }
145     if (!vreme--){
```

```
146         wsevent.emit('kraj');
147       }
148       clientLabNamespace.emit('vreme', vreme);
149     }
150
151
152     clientLabNamespace.on('connection', function(socket){
153       odg = true;
154       socket.emit('w', 'jeeej :)');
155       socket.emit('ssw0', sw[0].io.pins[0]);
156       socket.emit('ssw1', sw[0].io.pins[0]);
157       socket.on('ruta', function(rt){
158         ruta = rt;
159       });
160       socket.on('Jesam', function(){
161         odg = true;
162       });
163       socket.on('sw', function(data){
164         if (!(sw[0].io.pins[data].value==0)){
165           sw[data].off();
166         } else {
167           sw[data].on();
168         }
169       socket.emit('ssw', {"value" : sw[0].io.pins[data].value, "dugme"
              : data-1} );
170       console.log(data);
171       console.log(sw[0].io.pins[data].value)
172       })
173     });
174
175 var clientHomeNamespace = io.of('/client-home');
176
177 clientHomeNamespace.on('connection',function(socket){
178   if (dostupno){
179     clientHomeNamespace.emit('slobodno',ruta);
180   }
181 })
182 }
```

11.2 Изворни код клијентске стране

```
1 <html>
2 <head>
3 <meta charset="UTF-8">
4 <title>WEB Laboratorija</title>
5 <script src="https://cdn.socket.io/socket.io-1.3.7.js"></script>
6 <link rel="stylesheet" href="http://maxcdn.bootstrapcdn.com/
    bootstrap/3.3.6/css/bootstrap.min.css">
7
8 <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/
    jquery.min.js"></script>
```

```
9
10 <script src="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/
     bootstrap.min.js"></script>
11 <style>
12 .img-responsive{
13   margin: 0 auto;
14 }
15 </style>
16 </head>
17 <body>
18   <div class="container-fluid">
19     <div class="row">
20       <div class="col-md-12 text-center">
21         <div id='proba'></div>
22         <div id='sifra'></div>
23         <div class="col-md-6 col-sm-6 col-xs-6 align-left">
24           <img src="http://147.91.203.60:8081/video.mjpg?q=30&fps=33
               " class="img-responsive left-block"></img>
25         </div>
26         <div class="col-md-6 col-sm-6 col-xs-6 align-right">
27           <img src="http://147.91.203.57:8081/video.mjpg?q=30&fps=33
               " class="img-responsive right-block"></img>
28         </div>
29       </div>
30     </div>
31         <div class="row">
32
33           <div class="col-md-12 text-center">
34           <div class="center-block">
35             <form  id = "uploadForma" action ="/lab/upload/proba"
                 enctype="multipart/form-data" method="post">
36               <input type="text" name="title">
37               <input type="file" class = "center-block" name="upload
                   " multiple="multiple">
38               <input type="submit" value="Upload">
39             </form>
40           </div>
41           <br>
42           <div class="center-block">
43             <button id = "sw0" type="button" class="btn btn-success"
                 >SW0</button>
44             <button id = "sw1" type="button" class="btn btn-success"
                 >SW1</button>
45             <button id = "sw2" type="button" class="btn btn-success"
                 >SW2</button>
46             <button id = "sw3" type="button" class="btn btn-success"
                 >SW3</button>
47             <button id = "sw4" type="button" class="btn btn-success"
                 >SW4</button>
48             <button id = "sw5" type="button" class="btn btn-success"
```

```
                >SW5</button>
49              <button id = "sw6" type="button" class="btn btn-success"
                >SW6</button>
50              <button id = "sw7" type="button" class="btn btn-success"
                >SW7</button>
51              <span>   </span>
52              <button id = "pb0" type="button" class="btn btn-success"
                >BTN0</button>
53              <button id = "pb1" type="button" class="btn btn-success"
                >BTN1</button>
54              <button id = "pb2" type="button" class="btn btn-success"
                >BTN2</button>
55              <button id = "pb3" type="button" class="btn btn-success"
                >BTN3</button>
56            </div>
57            </div>
58
59
60
61      </div>
62
63  </div>
64  <script type="text/javascript">
65  var socket = io.connect('http://147.91.203.60:3001/client-lab');
66  socket.on('ssw', function(data){
67    if (!data.value){
68        if ($('#sw' + data.dugme.toString()).hasClass('btn-danger')){
69          $('#sw' + data.dugme.toString()).removeClass('btn-danger');
70        }
71      } else {
72        $('#sw' + data.dugme.toString()).addClass('btn-danger');
73      }
74  });
75
76  $('#sw0').click(function(dugme){
77      socket.emit('sw', 1);
78  });
79  $('#sw1').click(function(dugme){
80      socket.emit('sw', 2);
81  });
82  $('#sw2').click(function(dugme){
83      socket.emit('sw', 3);
84  });
85  $('#sw3').click(function(dugme){
86      socket.emit('sw', 4);
87  });
88  $('#sw4').click(function(dugme){
89      socket.emit('sw', 5);
90  });
91  $('#sw5').click(function(dugme){
```

```
92      socket.emit('sw', 6);
93  });
94  $('#sw6').click(function(dugme){
95      socket.emit('sw', 7);
96  });
97  $('#sw7').click(function(dugme){
98      socket.emit('sw', 8);
99  });
100 $('#pb0').mousedown(function(dugme){
101     socket.emit('sw', 9);
102     $( this ).toggleClass( "btn-danger" )
103 })
104 $('#pb1').mousedown(function(dugme){
105     socket.emit('sw', 10);
106     $( this ).toggleClass( "btn-danger" )
107 })
108 $('#pb2').mousedown(function(dugme){
109     socket.emit('sw', 11);
110     $( this ).toggleClass( "btn-danger" )
111 })
112 $('#pb3').mousedown(function(dugme){
113     socket.emit('sw', 12);
114     $( this ).toggleClass( "btn-danger" )
115 })
116 $('#pb0').mouseup(function(dugme){
117     socket.emit('sw', 9);
118     $( this ).toggleClass( "btn-danger" )
119 })
120 $('#pb1').mouseup(function(dugme){
121     socket.emit('sw', 10);
122     $( this ).toggleClass( "btn-danger" )
123 })
124 $('#pb2').mouseup(function(dugme){
125     socket.emit('sw', 11);
126     $( this ).toggleClass( "btn-danger" )
127 })
128 $('#pb3').mouseup(function(dugme){
129     socket.emit('sw', 12);
130     $( this ).toggleClass( "btn-danger" )
131 })
132 </script>
133 </body>
134 </html>
```