Универзитет у Београду

Електротехнички факултет



Конфигурабилни преводилац и симулатор архитектура рачунара

Мастер рад

|  |  |
| --- | --- |
| Ментор: | Кандидат: |
| доц. др Милош Цветановић | Бојан Јелача 3106/2014 |

Београд, Август 2016.

Садржај

[Садржај 2](#_Toc459472145)

[1. Увод 4](#_Toc459472146)

[2. Преводилац и симулатор - дефиниције и значења 5](#_Toc459472147)

[2.1. Преводилац 5](#_Toc459472148)

[2.1.1. Лексичка анализа 5](#_Toc459472152)

[2.1.2. Синтаксна анализа 6](#_Toc459472153)

[2.1.3. Семантичка анализа 6](#_Toc459472154)

[2.1.4. Генерисање кода 6](#_Toc459472155)

[2.2. Симулатор 6](#_Toc459472156)

[3. Преглед постојећих решења 7](#_Toc459472157)

[3.1. Превиодиоци 7](#_Toc459472158)

[3.1.1. А-0 систем 7](#_Toc459472159)

[3.1.2. Турбо Паскал 7](#_Toc459472160)

[3.1.3. ГНУ колекција преводилаца 8](#_Toc459472161)

[3.1.4. LLVM 8](#_Toc459472162)

[3.1.5. Visual Studio 9](#_Toc459472163)

[3.2. Симулатори 10](#_Toc459472164)

[3.2.1. Proteus Design Suite 10](#_Toc459472166)

[4. Преглед функционалности 12](#_Toc459472167)

[4.1. Дефинисање архитектуре рачунарских компоненти 12](#_Toc459472168)

[4.1.1. Дефинисање архитектуре меморије 12](#_Toc459472169)

[4.1.2. Дефинисање архитектуре компоненти општег типа 14](#_Toc459472170)

[4.2. Дефинисање рада рачунарских компоненти 15](#_Toc459472171)

[4.3. Дефинисање изгледа рачунарских компоненти 16](#_Toc459472172)

[4.3.1. Подразумевани начин дефинисања изгледа рачунарске компоненте 17](#_Toc459472173)

[4.3.2. Дефинисање изгледа рачунарске компоненте кроз графички интерфејс 17](#_Toc459472174)

[4.3.3. Дефинисање изгледа рачунарских компоненти писањем наредби у коду 18](#_Toc459472175)

[4.4. Пројектовање рачунарског система 19](#_Toc459472176)

[4.4.1. Додавање компоненти 20](#_Toc459472177)

[4.4.2. Дефинисање веза између компоненти 20](#_Toc459472178)

[4.4.3. Додавање магистрале 20](#_Toc459472179)

[4.4.4. Meњање фреквенције рада система 21](#_Toc459472180)

[4.4.5. Чување пројектованог система 21](#_Toc459472181)

[4.4.6. Учитавање претходно сачуваног система 21](#_Toc459472182)

[4.5. Симулирање рада рачунарског система 21](#_Toc459472183)

[5. Реализација 23](#_Toc459472184)

[1. Стил писања текста 24](#_Toc459472185)

[5.1. Организација текста тезе 24](#_Toc459472186)

[5.2. Стил писања, скраћенице, преводи са енглеског 24](#_Toc459472187)

[5.3. Слике, табеле, променљиве у тексту 25](#_Toc459472188)

[5.4. Мерне јединице 27](#_Toc459472189)

[5.5. Програмски код 27](#_Toc459472190)

[5.6. Референце 28](#_Toc459472191)

[5.7. Списак скраћеница, слика, табела 28](#_Toc459472192)

[5.8. Захвалница 28](#_Toc459472193)

[6. Форматирање текста 29](#_Toc459472194)

[6.1. Форматирање текста и стилови у алату *Microsoft Word* 29](#_Toc459472195)

[6.2. Форматирање насловне стране 30](#_Toc459472196)

[6.3. Форматирање текста тезе 31](#_Toc459472197)

[6.3.1. Хијерархија наслова текстуалних целина 31](#_Toc459472198)

[6.3.2. Форматирање текста у оквиру текстуалних целина 32](#_Toc459472199)

[6.3.3. Слике и табеле 32](#_Toc459472200)

[6.3.4. Једначине 32](#_Toc459472201)

[6.3.5. Променљиве 32](#_Toc459472202)

[6.3.6. Набрајање 32](#_Toc459472203)

[6.4. Листа референци (списак литературе) 33](#_Toc459472204)

[6.5. Списак скраћеница, слика и табела 34](#_Toc459472205)

[6.6. Прилози 34](#_Toc459472206)

[6.7. Садржај 34](#_Toc459472207)

[7. Литература 36](#_Toc459472208)

[8. Списак скраћеница 37](#_Toc459472209)

[9. Списак слика 38](#_Toc459472210)

[10. Списак табела 39](#_Toc459472211)

[Подешавање стилова текста за *Word* 2003 40](#_Toc459472212)

[Подешавање изгледа странице 40](#_Toc459472213)

[Форматирање основног текста и наслова хијерархијски уређених текстуалних целина 40](#_Toc459472214)

[Основни текст 42](#_Toc459472215)

[Наслов поглавља 44](#_Toc459472216)

[Наслов потпоглавља 46](#_Toc459472217)

[Наслов одељка 47](#_Toc459472218)

[Наслов пододељка 47](#_Toc459472219)

[Слике 48](#_Toc459472220)

[Позиционирање слике 48](#_Toc459472221)

[Назив и нумерација слике 49](#_Toc459472222)

[Табеле 49](#_Toc459472223)

[Позиционирање и изглед табеле 49](#_Toc459472224)

[Назив и нумерација табеле 49](#_Toc459472225)

[Једначине 49](#_Toc459472226)

[Набрајање 50](#_Toc459472227)

[Садржај 54](#_Toc459472228)

[Списак литературе 55](#_Toc459472229)

[Форматирање наслова за садржај, списак литературе (скраћеница, слика, табела) 55](#_Toc459472230)

[Наслови хијерархијски уређених целина у прилозима централном тексту тезе 56](#_Toc459472231)

[Први ниво наслова у прилозима 56](#_Toc459472232)

[Други ниво наслова у прилозима 56](#_Toc459472233)

[Трећи ниво наслова у прилозима 57](#_Toc459472234)

[Четврти ниво наслова у прилозима 57](#_Toc459472235)

1. Увод

Тема овог рада је самостална израда софтверске апликације чија је сврха да кориснику омогући дефинисање произвољног рачунарског система за који онда добија преводилац и симулатор.

У поглављу 2 је дат кратак теоријски осврт на основне елементе теме рада. Дате су кратке дефиниције и теоријска објашњења појмова *преводилац* и *симулатор*.

У области преводилаца постоје бројна, већ развијена решења. Исто важи и за област симулатора. Међутим, оно што ово решење издваја од осталих јесте могућност да кориcник сам дефинише целокупан систем, као и да на различите начине прати извршавање кода на свом систему. Детаљнији преглед постојећих решења и разлике реализованог решења у односу на постојећа дати су у поглављу 3.

Сам рад представља надоградњу дипломског рада !Наслов рада! који је кориснику омогућавао да дефинише само архитектуру рачунара. Апликација која је предмет и циљ овог рада омогућава кориснику да, поред архитектуре рачунара, дефинише и архитектуре осталих рачунарских компоненти, начине на који компоненте међусобно интерагују, као и изглед сваке од компоненти. Након што то уради, корисник добија преводилац за језик дефинисане архитектуре рачунара и симулатор на коме може да покреће програмски код. Приликом покретања кода, корисник може да прати стања свих компоненти система, да прати вредности сигнала и магистрала које повезују компоненте, да дефинише фреквенцију рада система итд. Детаљан преглед свих корисничких функционалности заједно са корисничким упуством дат је у поглављу 4.

Основна намена апликације је едукативна и експериментална. У едукативном смислу, апликација је намењена наставницима како би лакше пренели знање студентима везано за неку конкретну архитектуру рачунарског система, као и како би омогућили студентима лакше разумевање теоријских основа рачунарских система. Такође, едукативна сврха апликације је и обучавање студената за пројектовање рачунарских система. У експерименталном смислу, апликација се може примењивати за тестирање нових идеја у пројектовању рачунарских система, пре него што се она хардверски реализују, како би се одмах виделе основне предности и мане идеја. О овој теми биће више речи у поглављу 5.

Апликација је реализована коришћењем програмског језика Ц# (енг. *С#*), радног оквира *.NET 4.5* компаније Мајкрософт (енг. *Microsoft*). За реализацију графичког интерфејса коришћено је окружење *Windows forms.* За реализацију функционалног дела апликације, коришћено је неколико спољашњих библиотека. Детаљан опис целокупне реализације апликације са примерима програмског кода налази се у поглављима !Списак поглавља!.

У последњем поглављу, изведен је закључен и дате смернице како се реализована апликација може проширивати и надограђивати.

1. Преводилац и симулатор - дефиниције и значења

У овом поглављу дат је кратак теоријски осврт на термине *преводилац* и *симулатор* у домену рачунарске технике. Дефинисана су њихова значења, наведене намене и истакнуте разлике.

Иако се ова два појма често спомињу и реализују заједно, суштински су то два потпуно различита и независна термина. Стога, преводилац може да постоји потпуно независно од симулатора, и обрнуто. Имајући то у виду, софтверску реализацију преводиоца је могуће извести потпуно независно и за постојање преводиоца није неопходно постојање симулатора. Наравно, важи и обрнуто - реализацију симулатора је могуће извести потпуно независно и за постојање симулатора није неопходно постојање преводиоца.

* 1. Преводилац

Преводилац представља компјутерски програм или низ програма који трансформише код написан у неком програмском језику у машински код који се након тога може извршавати на циљној машини. Улаз преводиоца је датотека или скуп датотека које садрже код написан у програмском језико. Преводилац као излаз даје датотеку која се може извршити на циљној машини.

Постоји неколико фаза превођења кода написаног у програмском језику у машински код:

* лексичка анализа
* синтаксна анализа
* семантичка анализа
* генерисање кода

2. 1. 1. Лексичка анализа

Лексичка анализа представља анализирање програмског кода током којег се код трансформише у листу токена. Сваки токен представља низ карактера коме се додаје специфично значење. Тако генерисани токени представљају почетно стање за синтаксну анализу. Програм или део програма који обавља лексичку анализу обично се назива лексер.

Током лексичке анализе, програмски код пролази кроз низ провера на основу предефинисаног скупа правила, односно граматике програмског језика. На основу граматике, лексер, на основу граматике, групише програмски код на тај начин креирајући токене. Тако креирани токени су делови програмског кода који представљају смислену целину.

У случају да се у оквиру лексичке анализе пронађу делови кода који ни на који начин не могу да се уклопе у граматику програмског језика, лексер обично прекида рад преводиоца. Типичне грешке које се у овој фази могу открити су употреба непостојећих кључних речи, употреба непостојећих симболаи слично.

* + 1. Синтаксна анализа

Друга фаза превођења програмског кода је синтаксна анализа или парсирање. Програм или део програма који извршава синтаксну анализу се најчешће назива парсер.

У овој фази, низ токена добијен од лексера се трансформише у синтаксно стабло, које представља улаз семантичке анализе. Типична репрезентација синтаксног стабла је стабло чији сваки чвор дефинише операцију коју треба извршити, а деца тог чвора представљају аргументе операције.

Током синтаксне анализе могу пронаћи грешке као што су некомплетне наредбе, погрешан распоред симбола операција и операнада и слично. Уобичајено је да се, у случају појаве синтаксни грешака не прелази на фазу семантичке анализе, већ се превођење зауставља.

* + 1. Семантичка анализа

У процесу семантичке анализе користи се синтаксно стабло да би се проверила семантичка доследност програмског кода са дефинисаним програмским језиком. Такође, прикупљају се информације о типовима које су од користи за генерисање кода.

Битан део семантичке анализе је проверавање типове. У овом делу, преводилац проверава да се тип сваког операнда поклапа са операцијом. На пример, многи програмски језици захтевају да индекс низа буде цео број. У случају да се као индекс низа не користи цео број.

* + 1. Генерисање кода

Последња фаза превођења је генерисање кода. У овој фази се на основу информација из синтаксног стабла и знања о машинском језику генерише машински код који се онда може извршавати на циљној машини. Тако генерисани код се уписује у меморију и спреман је за извршавање.

* 1. Симулатор

Симулатор је рачунарски програм који симулира извршавање програма за друге рачунаре или рачунарске системе.

Основна сврха симулатора јесте евалуација и валидација неке новодефинисане архитуктере рачунара или рачунарског система. Такође, симулатори се могу користити и у процесима обучавања будућих корисника неког рачунарског система.

1. Преглед постојећих решења

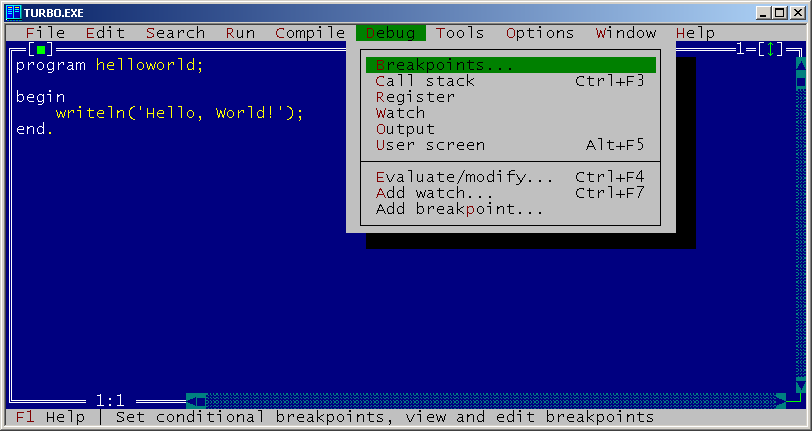
Постоји велики број рачунарских преводилаца и симулатора. У овом поглављу дат је кратак преглед неколико историјски најважнијих преводилаца и симулатора.

* 1. Превиодиоци
     1. А-0 систем

А-0 систем (аритметички језик - верзија 0) написан је од стране америчке научнице Грејс Хопер 1951. године. Циљни рачунар био је *UNIVAC I*. Ово је први преводилац икада развијен. Чешћа намена овог система је било уписивање програма у меморију и повезивање програма. Програми су се састојали од потпрограма и аргумената за потпрограме и по томе је језик највише личио на асемблерске језике. Овај преводилац је тако дефинисане програме преводио директно у машински код.

* + 1. Турбо Паскал

Турбо Паскал је развојни систем који укључује преводилац и графичко развојно окружење за програмски језик Паскал. Развила га је компанија Борланд 1983. године. Овај алат је значајан јер је један од првих који је укључивао графичко развојно окружење, које је данас уобичајени део сваког озбиљног система за развој софтвера. На наредној слици приказан је изглед графичког окружења алата Турбо Паскал.



Слика 3.1.1. Графичко окружење алата Турбо Паскал

* + 1. ГНУ колекција преводилаца

ГНУ колекција преводилаца је систем преводилаца развијен од стране компаније *GNU Project* који подржава различите програмске језике. Настао је 1987. године. Систем је написан у програмском језику Ц, а касније верзије су написане у језику Ц++. Одиграо је веома битну улогу у развоју бесплатног софтвера.

Иницијална верзија је подржавала само језик Ц, а врло брзо развијена је и верзија за Ц++. Данас подржава велики број језика, међу којима:

* Ц
* Ц++
* Објектни Ц
* Објектни Ц++
* Фортран
* Јава
* Ада
* *Go*

Подржава још већи број архитектура, од којих су најбитније:

* *ARM*
* *AVR*
* *SPARC*
* *x86*
* *x64*
* *MIPS*
* *Intel i960*
  + 1. LLVM

*LLVM додати у табелу скраћеница (Low Level Virtual Machine)* јe колекција модуларних и вишеструко употребљивих преводилаца. Писан је у језику Ц++ и подржава оптимизацију у време превођења, повезивања и извршавања. Настао је 2003. године на Универзитету Илиној.

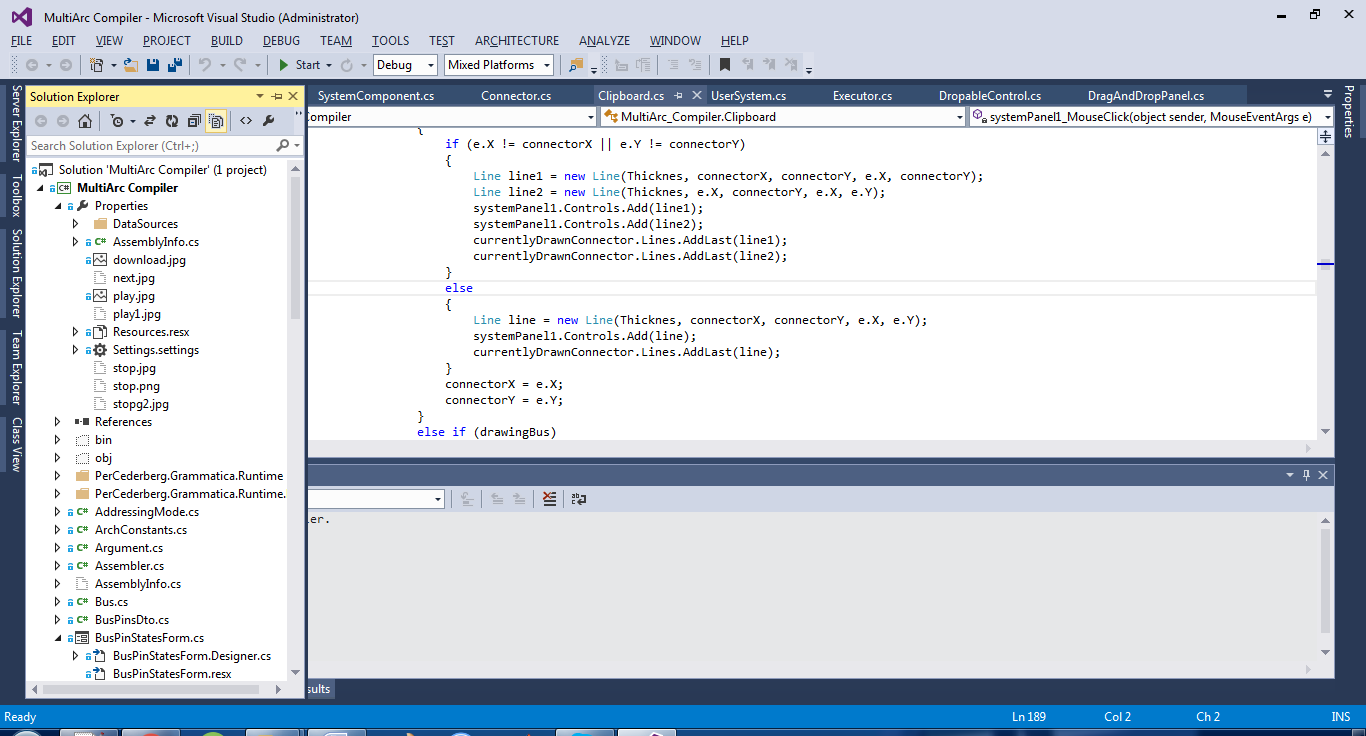
Између осталих, *LLVM* подржава следеће језике:

* *ActionScript*
* Ада
* Ц#
* Делфи
* Фортран
* Јава
* Пајтон
* Руби
* Скала
  + 1. Visual Studio

*Visual Studio* је интегрисано развојно окружење развијено од стране компаније Мајкрософт. Превасходна намена овог окружења је развој апликација за оперативни систем *Microsoft Windows*, као веб сајтова, веб апликација и веб сервиса. Прва верзија објављена је 1997. године, а тренутно је актуелна верзија 2015.

Укључује едитор који подржава аутоматско завршавање наредби као и аутоматско преуређивање кода. Подржава дебаговање на нивоу програмског кода и на нивоу машинског кода. Такође, укључује многе уграђене графичке алате за развој корисничког интерфејса, веб дизајн, дијаграме класа и дизајн шема база података.

На слици 3.1.2. приказан је типичан изглед окружења *Visual Studio*.



Слика 3.1.2. Типичан изглед екрана у алату *Visual Studio*

Ово окружење подржава различите програмске језике као што су:

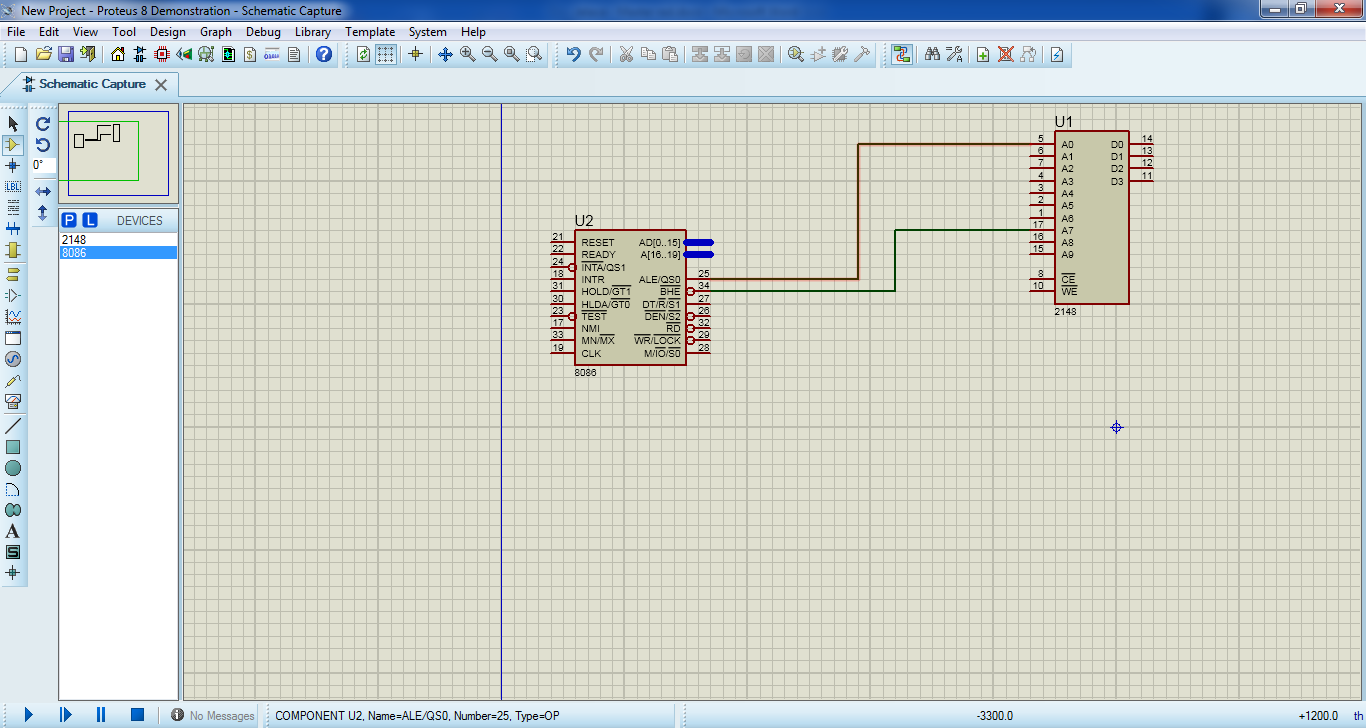
* Ц
* Ц++
* *Visual Basic*
* Ц#
* Ф#
* *XML*
* *HTML*
* *JavaScript*
* *CSS*
  1. Симулатори

У овом делу пажња неће бити посвећена прегледу већег броја симулатора развијених кроз исторују, јер је њихов број заиста огроман. Исто тако, сви симулатори развијани до сада су имали најразличитије намене, од симулирања рада авиона све до симулирања рада специфичног рачинарског система. Уместо тога, пажња ће бити посвећена окружењу *Proteus Design Suite*, развијеног од стране британске компаније *Labcenter Electronics Ltd*, које је било један од основних узора апликације која је тема овог рада.

* 1. 1. Proteus Design Suite

*Proteus Design Suite* је алат који је развила британска компанија *Labcenter Electronics Ltd.* 1988. године. Алат се покреће на оперативном систему *Microsoft Windows.*

Ово је алат за аутоматизацију електронског дизајна који укључује дефинисање шеме система, симулацију рада микроконтролера и бројне модуле штампаних плоча. На слици 3.2.1. приказан је део алата *Proteus Design Suite* за дефинисање шеме система.



Слика 3.2.1. Дефинисање шеме система у алату *Proteus Design Suite*

Симулација рада микроконтролера се рализује тако што као улаз добије датотеку са машинским кодом. Након тога, симулација се обавља паралелно са било којом електронском компонентом повезаном било аналогним, било дигиталним везама. То омогућује да се користи за широк спектар прототипова из области:

* контрола мотора
* контрола температуре
* дизајн корисничког интерфејса

Пошто не захтева постојање никаквог хардвера, има веома широку примену као алат за учење и вежбање.

Покрива широк спектар технологија:

* *Microchip Technologies*
* *Arduino (Atmel AVR)*
* *NXP*
* *Texas Instruments*

Алат *Proteus Design Suite* подржава и дизајн штампаних плоча. То је могуће урадити тако што се прво дефинише унутрашња шема штампане плоче, а затим и спољашњи изглед и пинови.

1. Преглед функционалности

У овом поглављу дат је детаљан преглед свих функционалности које апликација која је предмет овог рада укључује, заједно са упутством за реализовање ових функционалности. Пошто је део функционалности развијен као тема дипломског рада !наслов рада! у овом поглављу пажња се посвећује искључиво функционалностима које нису биле део дипломског рада.

Функционалности које ова апликација додаје на функционалности апликације која је развијена као тема дипломског рада могу се поделити на:

* дефинисање архитектуре рачунарских компоненти
* дефинисање рада рачунарских компоненти
* дефинисање изгледа рачунарских компоненти
* пројектовање рачунарског система
* симулирање рада рачунарског система
  1. Дефинисање архитектуре рачунарских компоненти

Апликација разликује три типа рачунарских компоненти:

* процесор
* меморија
* компоненте општег типа

Процесор је централна компонента рачунарског система која може покретати машински код добијен превођењем програмског кода. Овом типу компоненте се неће посвећивати већа пажња, пошто је детаљно описан у дипломском раду.

У наредним одељцима, пажња ће бити посвећена меморији и компонентама општег типа.

* + 1. Дефинисање архитектуре меморије

Меморија је компонента чија је намена складиштење веће количине података. Она не може извршавати програмски код и нема регистарски фајл, већ само меморијске локације. Корисник може дефинисати протокол по коме се подаци дохватају из меморије и протокол по коме се подаци уписују у меморију. Детаљан опис дефинисања овог протокола дат је у потпоглављу 4.2.

Услов за рализацију функционалности дефинисања архитектуре меморије је претходно отворен пројекат.

Да би дефинисао архитектуру меморије, корисник треба из главног менија апликације да изабере опцију *Architecture*, из менија који се отвори опцију *Load* и затим опцију *Memory*. Све ово се може остварити и истовременим притиском тастера *CTRL* и *М* на тастатури. Након тога, отвара се дијалог за избор датотеке у којој се налази опис архитектуре меморије.

Уколико већ није, корисник може ван апликације да дефинише датотеку која описује архитектуру меморије. Ради се о датотеци написаној у *XML* језику чији је детаљан опис дат у наставку текста.

Цела датотека треба да садржи један главни *XML* чвор *memory.*Унутар овог чвора налазе се чворови који дефинишу својства меморије:

* **Чвор *name* -** Вредност овог чвора дефинише име меморијске компоненте
* **Чвор *filename* -** Вредност овог чвора дефинише локацију и име датотеке у којој се налази код који дефинише рад меморије, о чему ће више речи бити у потпоглављу 4.2. Уколико се овај чвор не наведе, узима се датотека са именом истим као име меморијске компоненте, са екстензијом *.cs* и смешта у директоријум */Data/Memories* унутар пројектног фолдера.
* **Чвор *size* -** Вредност овог чвора дефинише укупну величину меморије у бајтовима
* **Чвор *au*** *-* Вредност овог чвора дефинише величину адресне јединице
* **Чвор *rom\_start* -** Вредност овог чвора дефинише почетну адресу РОМ меморије
* **Чвор *rom\_end* -** Вредност овог чвора дефинише крајњу адресу РОМ меморије
* **Чвор *ram\_start*** *-* Вредност овог чвора дефинише почетну адресу РАМ меморије
* **Чвор *ram\_end* -** Вредност овог чвора дефинише крајњу адресу РАМ меморије
* **Чвор *init\_file*** **-** Вредност овог чвора дефинише локацију датотеке у којој се налази иницијални садржај меморије који корисник може да дефинише
* **Чвор *storage\_file*** **-** Вредност овог чвора дефинише локацију датотеке у којој се чува садржај меморије током извршавања програма
* **Чвор *dimensions* -** Вредност овог чвора дефинише величину визуелне репрезентације меморије. Више о овоме биће у потпоглављу 4.3.
* **Чвор *ports* -** Вредност овог чвора дефинише портове које меморија садржи. Овај чвор има своје подчворове од којих сваки представља назив порта. Подчворови сваког овог чвора дефинишу његова својства:
  + **Чвор *name* -** Вредност овог подчвора дефинише име порта
  + **Чвор *side* -** Вредност овог подчвора дефинише страну на којој се налази порт. Може имати вредности *left, rigth, up* и *down.*
  + **Чвор *type* -** Вредност овог подчвора дефинише тип порта. Може имати вредности *in, out* и *inout.*
  + **Чвор *number* -** Вредност овог подчвора дефинише величину порта у битовима, тј. број пинова порта.
* **Чвор *design* -** Овај чвор се користи при дефинисању изгледа меморије. О овоме ће више речи бити у потпоглављу 4.3.

Након што је учитао датотеку која описује архитектуру меморије, апликација сама генерише датотеку специфицирану чвором *filename*, уколико она већ није генерисана. Ова датотека дефинише код који описује рад меморије. Више речи о томе биће у потпоглављу 4.2.

* + 1. Дефинисање архитектуре компоненти општег типа

Компоненте општег типа су компоненте чија је намена обављање посла који корисник дефинише. Оне не могу извршавати програмски код и немају меморијске локације, већ само регистарски фајл. Детаљнији опис дефинисања посла који компонента обавља дат је у потпоглављу 4.2.

Услов за рализацију функционалности дефинисања архитектуре компоненти општег типа је претходно отворен пројекат.

Да би дефинисао архитектуру компоненте општег типа, корисник треба из главног менија апликације да изабере опцију *Architecture*, из менија који се отвори опцију *Load* и затим опцију *Other Component*. Све ово се може остварити и истовременим притиском тастера *CTRL* и *T* на тастатури. Након тога, отвара се дијалог за избор датотеке у којој се налази опис архитектуре компоненте општег типа.

Уколико већ није, корисник може ван апликације да дефинише датотеку која описује архитектуру компоненте општег типа. Ради се о датотеци написаној у *XML* језику чији је детаљан опис дат у наставку текста.

Цела датотека треба да садржи један главни *XML* чвор *othercomponent.*Унутар овог чвора налазе се чворови који дефинишу својства компоненте општег типа:

* **Чвор *name* -** Вредност овог чвора дефинише име компоненте
* **Чвор *filename* -** Вредност овог чвора дефинише локацију и име датотеке у којој се налази код који дефинише рад компоненте општег типа, о чему ће више речи бити у потпоглављу 4.2.Уколико се овај чвор не наведе, узима се датотека са именом истим као име компоненте, са екстензијом *.cs* и смешта у директоријум */Data/Other* унутар пројектног фолдера.
* **Чвор *dimensions* -** Вредност овог чвора дефинише величину визуелне репрезентације компоненте. Више детаља о овоме биће изнето у потпоглављу 4.3.
* **Чвор *registers* -** Вредност овог чвора дефинише регистре које компонента садржи. Сваки подчвор представља дефиницију једног регистра. Његови подчворови дефиншу својства регистра:
  + **Чвор *size* -** Вредност овог подчвора дефинише величину регистра у битовима.
  + **Чвор *name* -** Вредност овог подчвора дефинише име регистра
  + **Чвор *value* -** Вредност овог подчвора дефинише почетну вредност регистра
  + **Чвор *part* -** Ово је посебан тип подчвора који служи за дефинисање делова регистра, односто подрегистара. Сваки подчвор овог типа дефинише један део регистра који се може користити у програмском коду.Дефинисање дела неког регистра се обавља кроз подчворове подчвора *part*:
    - **Чвор *start* -** Овај подчвор дефинише бит од кога део регистра почиње
    - **Чвор *end* -** Овај подчвор дефинише бит на којем се завршава део регистра
    - **Чвор *name* -** Овај подчвор дефинише име, односно псеудоним дела регистра који се може користити у програмском коду
* **Чвор *ports* -** Вредност овог чвора дефинише портове које компонента садржи. Овај чвор има своје подчворове од којих сваки представља назив порта. Подчворови сваког овог чвора дефинишу његова својства:
  + **Чвор *name* -** Вредност овог подчвора дефинише име порта
  + **Чвор *side* -** Вредност овог подчвора дефинише страну на којој се налази порт. Може имати вредности *left, right, up* и *down.*
  + **Чвор *type* -** Вредност овог подчвора дефинише тип порта. Може имати вредности *in, out* и *inout.*
  + **Чвор *number* -** Вредност овог подчвора дефинише величину порта у битовима, тј. број пинова порта.
* **Чвор *design* -** Овај чвор се користи при дефинисању изгледа компоненте. О овоме ће више речи бити у потпоглављу 4.3.

Након што је учитао датотеку која описује архитектуру меморије, апликација сама генерише датотеку специфицирану чвором *filename*, уколико она већ није генерисана. Ова датотека дефинише код који описује рад меморије. Више речи о томе биће у потпоглављу 4.2.

* 1. Дефинисање рада рачунарских компоненти

Да би корисник могао да дефинише рад неке рачунарске компоненте, треба да отвори пројекат и да учита архитектуру те компоненте.

Након што учита архитектуру компоненте, у директоријуму *Data/Memories* или *Data/Other* отвори датотеку са именом компоненте и суфиксом *.cs*, ако није другачије специфицирао својством *filename.* У овој датотеци, постоји предефинисана метода *Cycle,* која као аргумент прихвата компоненту чији рад се дефинише. У случају да се ради о меморији, компонента ће бити типа *Memory,* ау супротном ће бити типа *OtherComponent*. Дефинисање рада процесора је описано у дипломском раду и овде се томе неће посвећивати пажња.

Корисник може да измени тело ове методе и тиме дефинише скуп наредби које компонента циклично обавља. Како би се кориснику олакшао посао, објекат компоненте који је прослеђен као аргумент пружа следеће методе:

* ***GetPort* -** Ова метода прихвата аргумент типа стринг и враћа порт са именом истим као прослеђени аргумент, уколико такав порт постоји
* ***GetPin* -** Ова метода прихвата аргумент типа стринг и враћа пин са именом истим као прослеђени аргумент, уколико такав пин постоји
* ***GetAllPorts* -** Ова метода враћа низ свих портова компоненте
* ***GetAllPins* -** Ова метода враћа низ свих пинова компоненте
* ***Wait*** **-** Ова метода реализована је на три начина. Уколико корисник проследи аргументе типа стринг и цео број, онда ова метода блокира извршавање док порт са именом истим као први аргумент не добије вредност другог аргумента. У случају да корисник проследи аргументе типа стринг и *PinValue*, метода блокира извршавање док пин са именом истим као први аргумент не добије вредност другог аргумента, која може бити из опсега *TRUE, FALSE* или *HIGHZ.*Ако корисник проследи један целобројни аргумент, метода блокира извршавање онолико откуцаја системског сата колико је специфицирано артументом методе
* ***WaitForRisingEdge* -** Ова метода прихвата аргумент типа стринг који представља име пина. Метода блокира извршавање док се не појави узлазна ивица на пину специфицираном са аргументом методе, ако такав постоји
* ***WaitForFallingEdge* -** Ова метода прихвата аргумент типа стринг који представља име пина. Метода блокира извршавање док се не појави силазна ивица на пину специфицираном са аргументом методе, ако такав постоји
* **Дохватање податка са меморијске локације** *-* Уколико се дефинише рад меморије, корисник може прочитати вредност са било које локације у меморији. То може учинити са *memory[address],* где је*memory* аргумент прослеђен методи *Cycle*, а *address* представља адресу меморијске локације са које се жели дохватити податак
* ***GetRegister* -** Ова метода прихвата стринг као аргумент и враћа регистар специфициран аргументом, уколико такав постоји. Може се позвати само у случају да се дефинише рад компоненте општег типа.

Након што је дефинисао рад компоненте, корисник треба да поново учита архитектуру те компоненте, на начин описан у потпоглављу 4.1, да би апликација поново учитала код који описује рад компоненте. Алтернативни начин за остваривање овога је избором опције *Architecture* из главног менија, а затим кликом на опцију *Recompile.* То се може остварити и притиском тастера *F3* на тастатури.

* 1. Дефинисање изгледа рачунарских компоненти

Кориснику апликације пружена је могућност да дефинише визуелну репрезентацију сваке компоненте. Услов за реализацију ове функционалности је отворен пројекат и дефинисана архитектура компоненте.

Дефисање изгледа компоненте може се остварити на три начина:

* **подразумевани начин** - корисник дефинише само димензије и позиције портова
* **кроз графички интерфејс** - корисник користи графички интерфејс специјално за намену дефинисања изгледа компоненте
* **писањем наредби у коду** - корисник дефинише код који се извршава при , исцртавању компоненте
  + 1. Подразумевани начин дефинисања изгледа рачунарске компоненте

За овај начин, већина својстава је предефинисана.

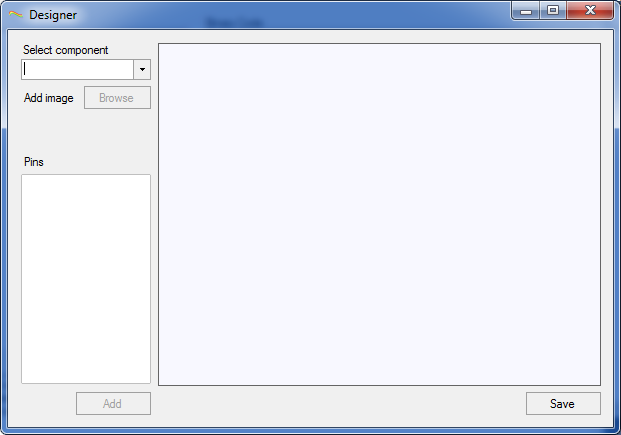
Корисник треба да, при дефиницији архитектуре компоненте (као што је описано у потпоглављу 4.1), дефинише локацију сваког порта. Поред тога, корисник треба да дефинише чвор *dimensions*, којим су обухваћена два подчвора:

* **Чвор *height* -** Вредност овог подчвора дефинише висину компоненте у пикселима
* **Чвор *width* -** Вредност овог подчвора дефинише ширину компоненте у пикселима

Када корисник на овај начин дефинише позиције портова и димензије компоненте, компонента ће се аутоматски исцртати. Компонента ће бити беле боје, са црним оквирима и портовима. Уколико се, при дефиницији система, о чему ће више речи бити у потпоглављу 4.4, изабере овако дефинисана компонента, њени оквири ће постати зелени.

* + 1. Дефинисање изгледа рачунарске компоненте кроз графички интерфејс

За дефинисање изгледа графичке компоненте, постоји посебан графички интерфејс. Да би се отворио овај графички интерфејс, корисник треба да изабера опцију *View* из главног менија апликације, а затим опцију *Designer*. Услов за ово је претходно отворен пројекат и дефнисана архитектура компоненте. На слици 4.3.1. приказан је поменути интерфејс.



Слика 4.3.1. Графички интерфеjс за дефинисање изгледа компоненти

Да би изабрао компоненту чији изглед жели да дефинише, корисник треба да из падајућег менија са леве стране изабере компоненту. Тада се, у листи пинова са леве стране појављују сви пинови изабране компоненте. Корисник може изабрати пин и кликом на дугме *Add*, пин се појављује на панелу за дефинисање изгледа компоненте и корисник га може притиском и померањем миша превлачити по панелу.

Такође, може се додати слика по жељи тако што се притисне дугме *Browse*. Тада се отвара дијалог за избор слике. По одабиру слике, корисник треба да изабере да ли жели да слика буде транспарентна или не. Уколико одабере да слика буде транспарентна, уклања се сва позадина беле боје. Додавање слике се може поновити више пута. Свака слика се може уклонити са панела десним кликом миша на њену површину и одабиром опције *Remove* у менију који се појави. У случају да постоји више слика које се преклапају, могуће је послати одређену слику иза свих осталих или је довести испред свих осталих слика. То се постиже десним кликом на слику и одабиром опције *Bring to Front* за довођење слике испред осталих слика, односно *Send to Back,* за слање слике иза осталих.

Када је завршено дефинисање изгледа компоненте, иста се може сачувати притиском на дугме *Save*. Тада се изглед компоненте чува и он представља унију свих додатих слика заједно са пиновима. Уколико се, при дефиницији система, о чему ће више речи бити у потпоглављу 4.4, изабере овако дефинисана компонента, цела компонента ће добити јачу нијансу зелене боје.

* + 1. Дефинисање изгледа рачунарских компоненти писањем наредби у коду

Уколико жели да има потпуну слободу при дефинисању изгледа рачунарске компоненте, корисник то може остварити писањем кода који се извршава при исцртавању компоненте.

При дефинисању архитектуре компоненте, корисник треба да дода чвор *design* и да му дода атрибут *code* чију ће вредност поставити на *true.* Када то уради, при учитавању архитектуре компоненте, аутоматски се генерише и датотека са именом компоненте и суфиксом *Design.cs*. У овој датотеци, налази се метода *DrawComponent* која као прихвата два аргумента. Први аргумент је типа компоненте чије се исцртавање дефнише (*CPU, Memory, OtherComponent*). Други аргумент је типа *Graphics*, из библиотеке *System.Drawing .NET* радног оквира и он се користи за исцртавање компоненте.

Поред свих метода описаних у потпоглављу 4.2, на располагању је још и својство *DefaultPen* које пружа информацију о боји којом се исцртавају линије. Ова боја се мења у зависности да ли је компоненте изабрана или не, при пројектовању рачунарског система, о чему ће више речи бити у потпоглављу 4.4.

У наставку је приказан пример исцртавања компоненте у коду.

**public** static void DrawComponent**(**MultiArc\_Compiler**.**OtherComponent component**,** System**.**Drawing**.**Graphics graphics**)**

**{**

component**.**Height **=** 20**;**

component**.**Width **=** 25**;**

Rectangle rectangle **=** **new** Rectangle**(**0**,** 0**,** 20**,** 20**);**

graphics**.**FillRectangle**(new** SolidBrush**(**Color**.**White**),** rectangle**);**

graphics**.**DrawRectangle**(**component**.**DefaultPen**,** rectangle**);**

component**.**GetPin**(**"CLK0"**).**Location **=** **new** Point**(**20**,** 10**);**

**}**

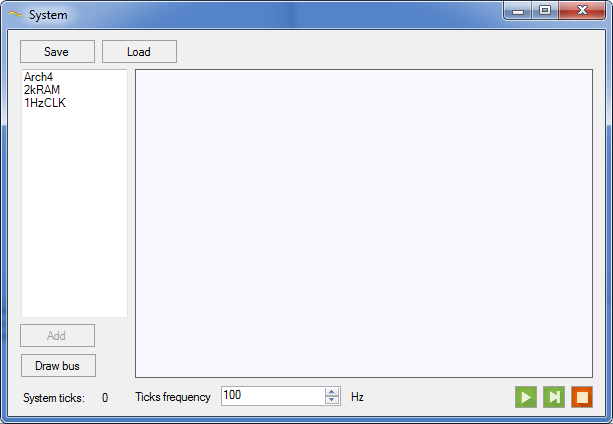
Након што је дефинисао исцртавање компоненте, корисник треба да поново учита архитектуру те компоненте, на начин описан у потпоглављу 4.1, да би апликација поново учитала код који описује рад компоненте. Алтернативни начин за остваривање овога је избором опције *Architecture* из главног менија, а затим кликом на опцију *Recompile.* То се може остварити и притиском тастера *F3* на тастатури.

* 1. Пројектовање рачунарског система

Корисник апликације има могућност да пројектује рачунарски систем. Ово овухвата дефинисање компоненти које ће систем користити, као и веза између њих.

Услов за реализовање ове функционалности је претходно отворен пројекат и учитана архитектура барем једне компоненте.

Да би започео дефинисање система, корисник треба у главном менију апликације да изабере опцију *View,* а затим из подменија који се отвори опцију *System.* Након тога, отвара се нови прозор, приказан на слици 4.4.1.



Слика 4.4.1. Прозор за пројектовање система

Пројектовање система је комплексан процес и може се поделити у неколико мањих функционалности, као што су:

* Додавање компоненти
* Дефинисање веза између компоненти
* Додавање магистрале
* Мењање фреквенције рада система
* Чување пројектованог система
* Учитавање претходно сачуваног система

У наставку текста, свака од ових функционалности је кратко објашењена.

* + 1. Додавање компоненти

Да би додао компоненту, корисник треба из листе постојећих компоненти, односно компоненти чију је архитектуру дефинисао, изабере на једну и затим кликне на дугме *Add.* Изабрана компонента се тада додаје на панел системски панел и корисник је може померати по системском панелу превлачењем миша.

Свака компонента, осим компоненти типа процесора, се може додавати више пута. Такође, све компоненте које нису процесор се могу копирати. То се може постићи десним кликом на компоненту која је додата на системски панел, затим одабиром опције *Copy*, а онда десним кликом на празан део системског панела и одабиром опције *Paste*.

Свака компонента се може и уклонити из система, десним кликом на њу, а затим одабиром опције *Remove*.

* + 1. Дефинисање веза између компоненти

Све компоненте се могу повезивати сигналима који повезују њихове пинове.

Левим кликом миша на пин компоненте, почиње се исцртавање сигнала. Док исцртавање траје, левим кликом миша се може направити промена правца исцртавања. Сигнал се може повезати са другим пином, левим кликом на њега, а може се и оставити неповезан са другим, што се чини десним кликом миша.

За сваки сигнал, може се везати други сигнал. То се може урадити именовањем сигнала. Десним кликом миша на сигнал, отвара се мени на коме треба изабрати опцију *Manage names.* Отвара се дијалог у коме се са леве стране види листа свих актуелних имена сигнала. У тој листи, бирањем било ког имена, а затим кликом на дугме *Remove*, изабрано име се уклања из листе имена. Укуцавањем имена у поље *New name,* а затим кликом на дугме  *Add,* ново име се додаје у листу имена. Кликом на дугме *Аpply,* извршене промене се чивају, док се кликом на дугме *Cancel* сачуване промене одбацују. Уколико се дода име које је већ додељено неком другом сигналу, та два сигнала се онда спајају. Новодобијени сигнал има листу имена која одговара унији претходно спојених сигнала.

Сваки додати сигнал се може и уклонити. То се чини десним кликом миша на њега, а затим одабиром опције *Remove.*

* + 1. Додавање магистрале

Корисник може додати произвољан број магистрала, а свака магистрала може имати произвољан број битова, тј. садржати произвољан број сигнала.

Да би се додала магистрала, треба кликнути на дугме *Draw bus*. Тада се прелази на мод цртања магистрале, из кога се излази поновним кликом на исто дугме, које је сада постало *Stop*. Док се налази у моду цртања магистрале, левим кликом на било који празан део системског панела, почиње цртање магистрале. Левим кликом миша може се направити промена правца исцртавања. Десним кликом, цртање магистрале се завршава и отвара се дијалог у коме се од корисника тражи да изабере ширину магистрале. Након тога, магистрала је направљена и излази се из мода цртања магистрале. Овако направљена магистрала садржи онолико сигнала колико је корисник изабрао за њену ширину и има аутоматски генерисано име. Сваки од њених сигнала има име исто као и магистрала, са суфиксом [*brojSignala*], где *brojSignala* представља редни број сигнала на магистрали.

Свакој магистрали се може мењати име на исти начин као што се то може чинити и са сигналом, при чему важе иста правила за спајање сигнала као и раније.

Магистрала се може уклонити десним кликом на њу, па одабиром опције *Remove*.

* + 1. Meњање фреквенције рада система

Да би променио фреквенцију, корисник треба да у одељак испод системског панела унесе вредност која представља фрекфенцију рада система у Hz.

* + 1. Чување пројектованог система

У сваком тренутку, корисник може сачувати пројектовани систем. Да би то учинио, треба да притисне дугме *Save*, а затим у дијалогу који се отвори да одабере локацију где жели да сачува систем.

* + 1. Учитавање претходно сачуваног система

Да би учитао претходно сачувани систем, корисник треба да притисне дугме *Load* и у дијалогу који се отвори одабере систем који жели да учита. Када ово уради, корисник губи све несачуване измене које је урадио на систему који је до тада пројектовао.

* 1. Симулирање рада рачунарског система

Корисник може симулирати рад рачунарског система. Услов за остваривање ове функционалности јесте претходно дефинисан пројекат и пројектован рачунарски систем.

Да би симулирао рад рачунарског система, корисник треба прво да напише програмски код у едитору кода и да га преведе у машински код. Овај код треба да одговара спецификацији језика који је дефинисао при дефинисању архитектуре процесора. Након што је код написан и преведен у машински код, он се може покретати на процесору који припада дефинисаном рачунарском систему. То се постиже притиском на дугме *Execute*, у доњем десном углу прозора за пројектовање система. Овакво покретање машинског кода извршава програм док не дође до краја или до прве следеће тачке заустављања, које се могу дефинисати у едитору. Други начин покретања је покретање у режиму такт-по-такт. Ово се може постићи притиском на дугме *Execute next step,* одмах поред *Execute* дугмета. Сваки притисак на ово дугме помера извршавање симулације за један такт системског времена, дефинисаног фреквенцијом рада система. Ова два режима се могу комбиновати, па се тако, на пример, првих 5 инструкција може извршити одједном, а затим се код може покретати такт-по-такт, да би се могли пратити неки специфични детаљи рада целог система.

У сваком тренутку, корисник може променити фреквенцију рада система, што се одражава на брзину рада система.

Током симулације, корисник на системском панелу може да прати вредности сигнала, пратећи боје сигнала. Такође, у сваком тренутку корисник може да види који пинови постављају вредности сигнала, као и које вредности постављају. То може урадити тако што направи десни клик на сигнал, а затим изабере опцију *See pin states*. Кориснику се приказује листа свих пинова и вредности које они постављају на изабрани сигнал.

У сваком тренутку, корисник може проверити или изменити вредности регистара компоненти које имају регистре. То може урадити десним кликом на жељену компоненту, а затим одабиром опције *Registers*. Тада се отвара дијалог са приказом вредности свих регистара компоненте, које је могуће изменити.

Корисник може у сваком тренутку проверити или изменити вредности меморијских локација компонената које имају меморијске локације. То може урадити десним кликом на жељен компоненту, а затим одабиром опције *Memory dump*. Тада се отвара дијалог са приказом вредности меморијских локације, које је такође могуће изменити.

1. Реализација

У овом поглављу, биће изнети детаљи имплементације апликације која представља предмет овог рада. Пошто је део апликације развијен као тема дипломског рада !наслов рада! у овом поглављу пажња се посвећује искључиво деловима који нису били део дипломског рада.

Апликација је реализована у програмском језилу Ц# и радном оквиру *.NET*, компаније *Microsoft.* За реализацију графичког интерфејса, коришћена је библиотека *Windows Forms*, из радног оквира *.NET.* Како би се поједноставила реализација појединих функционалности, коришћена је библиотека *LINQ (Language Integrated Query).* У целој апликацији коришћене су парадигме објектно-оријентисаног програмирања.

Реализација се може поделити у две основне целине, реализацију графичког интерфејса и реализацију логичких функционалности, а свака од њих се даље може поделити на више мањих целина, према функционалностима које имплементирају.

* 1. Реализација графичког интерфејса

Највећи део графичког интерфејса реализованог у оквиру овог рада тиче се имплементације прозора за дефинисање изгледа компоненти и прозора за пројектовање система. У наставку овог поглавља, објашњена је имплементација ова два прозора.

* + 1. Реализација прозора за дефинисање изгледа компоненти

Главни део прозора за дефинисање изгледа компоненти реализован је у оквиру класе *Designer* која имплементира класу *Form* из библиотеке *WindowsForms*. Објекат ове класе апстрахује прозор који служи за интеракцију са корисником преко корисничких контрола као штп су дугме, клизач, мени итд. Поред ове класе, користе се и друге специфичне контроле као што су *DragAndDropPanel* и *DropableControl,* о којима ће више речи бити у наставку. Дијаграм класа релевантних за имплементацију прозора за дефинисање изгледа компоненти приказан је на слици 5.1.1.



Слика 5.1.1. Дијаграм класа за имплементацију дефинисања изгледа компоненти

У наставку текста ће укратко бити описане класе приказане на претходној слици.

Класа Designer

Као што је већ речено, ова класа садржи највећи део кода који имплементира функционалност дефинисања изгледа компоненти и на неки начин представља централну класу овог дела апликације. Креирањем објекта ове класе, кориснику се приказује прозор за дефинисање изгледа компоненти, приказан на слици 4.3.1. У овој класи дефинисане су методе које представљају ослушкиваче догађаја који се одигравају при интеракцији корисника са прозором за дефинисање изгледа компоненти. Списак ових догађаја, као и начин реаговања на сваки од догађаја, дат је у табели 5.1.1.

|  |  |
| --- | --- |
| **Догађај** | **Реакција** |
| Промена изабране компоненте | Дозвољава се кориснику да кликне на дугме *Browse*, сви пинови изабране компоненте се додају у листу пинова и дозвољава се кориснику да кликне на дугме *Save*. Такође, брише се интерна листа додатих слика. |
| Одабир слике из дијалога | Кориснику се приказује дијалог за избор да ли жели да изабрана слика постане транспарентна. Након одлуке корисника, изабрана слика се додаје на позицију (0,0) на панелу за дизајн компоненте и у интерну листу додатих слика. |
| Клик на дугме *Save* | Листа додатих слика се копира у листу слика компоненте, а затим се чува дизајн компоненте у XML датотеку у којој је дефинисана архитектура компоненте. |
| Одабир *Send to Back* или *Bring to Front* | Изабрана слика се шаље иза или испред свих компоненти и ажурира се њено својство *Level,* како би се при поновном исцртавању компоненте могла задржати дубина сваке слике. |

Табела 5.1.1. Преглед најважнијих догађаја класе *Designer*

Од значаја је детаљније објаснити како се реализује чување дизајна компоненте у XML датотеку у којој је дефинисана архитектура компоненте. Када корисник кликне на дугме *Save,* у изабраној компоненти се прво обришу све постојеће слике, ако таквих има. Затим се пролази кроз све додате слике да би се идентификовале границе свих слика, односно пронашао правоугаоник минималне површине који садржи све слике. Ово је потребно да би се могле одредити нове димензије компоненте. Након тога се све слике транслирају тако да се налазе у горњем левом углу панела за дефинисање изгледа компоненте, али да се њихове међусобне позиције не промене, а затим и ажурира регион који ће компонента од сада заузимати. Овај регион се добија као унија региона свих слика и пинова. Овим се омогућује да компонента има и неправилан облик, уколико је сликама тако дефинисано. На самом крају се дизајн компоненте чува тако што се чувају сва релевантна својства у чвору *design* датотеке у којој је дефинисана архитектура компоненте. Код који реализује проналажење граница компоненте приказан је у наставку текста.

int upperBorder **=** DesignPanel**.**Height**;**

int lowerBorder **=** 0**;**

int leftBorder **=** DesignPanel**.**Width**;**

int rightBorder **=** 0**;**

**foreach** **(**var c **in** \_addedImages**)**

**{**

**if** **(**c**.**Location**.**X **<** leftBorder**)**

**{**

leftBorder **=** c**.**Location**.**X**;**

**}**

**if** **(**c**.**Location**.**X **+** c**.**Width **>** rightBorder**)**

**{**

rightBorder **=** c**.**Location**.**X **+** c**.**Width**;**

**}**

**if** **(**c**.**Location**.**Y **<** upperBorder**)**

**{**

upperBorder **=** c**.**Location**.**Y**;**

**}**

**if** **(**c**.**Location**.**Y **+** c**.**Height **>** lowerBorder**)**

**{**

lowerBorder **=** c**.**Location**.**Y **+** c**.**Height**;**

**}**

**}**

**foreach** **(**var p **in** \_addedPins**)**

**{**

**if** **(**p**.**Location**.**X **<** leftBorder**)**

**{**

leftBorder **=** p**.**Location**.**X**;**

**}**

**if** **(**p**.**Location**.**X **+** p**.**Width **>** rightBorder**)**

**{**

rightBorder **=** p**.**Location**.**X **+** p**.**Width**;**

**}**

**if** **(**p**.**Location**.**Y **<** upperBorder**)**

**{**

upperBorder **=** p**.**Location**.**Y**;**

**}**

**if** **(**p**.**Location**.**Y **+** p**.**Height **>** lowerBorder**)**

**{**

lowerBorder **=** p**.**Location**.**Y **+** p**.**Height**;**

**}**

**}**

\_selectedComponent**.**Size **=** **new** Size**(**rightBorder **-** leftBorder **+** 1**,** lowerBorder **-** upperBorder **+** 1**);**

Класа DragAndDropPanel

Класа *DragAndDropPanel* наслеђује класу *Panel* из библиотеке *WindowsForms.* Понашање ове класе у потпуности наслеђује понашање класе *Panel* и наме објеката ове класе јесте постављање на прозор како би корисник могао да додаје компоненте. Ова класа додаје и неке специфичности које класа *Panel* не дефинише.

Класа *DragAndDropPanel* имплементира специфичне ослушкиваче догађаја који се одигравају у оквиру реализоване апликације. У табели 5.1.2. приказани су сви имплементирани ослушкивачи и кратко описане акције које се изврше да би се реаговало на догађај.

|  |  |
| --- | --- |
| **Догађај** | **Реакција** |
| Спуштање компоненте на панел | Израчунава се нова локација компоненте. Ова локација се добија као локација на коју је извршено спуштање померена по *х* и *у* оси онолико колико је миш померен у односу на компоненту при подизању компоненте. |
| Притисак тастер миша | Деселектују се све компоненте на панелу и започиње исцртавање правоугаоника којим се селектују компоненте. То се чини тако што се у приватне променљиве запамти тачка од које се почело селектовање и поставе логичке променљиве које прате да је селектовање у току. |
| Померање миша | Уколико је селектовање у току, исцртава се правоугаоник између тачке у којој је селектовање почело и тачке у којој се налази миш. |
| Пуштање тастера миша | Завршава се процес селектовања и пролази се кроз све компоненте и проверава која од њих је обухваћена правоугаоником за селектовање. На крај се обухваћене компоненте селектују. |
| Одабир опције *Remove* у менију или притисак тастера *Delete* на тастатури | Пролази се кроз листу свих компоненти и селектоване компоненте уклањају са панела и из система, уколико је панел део прозора за пројектовање система. |
| Одабир опције *Copy* на некој компоненти или притисак тастера *CTRL* и *C* | Пролази се кроз листу свих компоненти и селектоване компоненте се пребацују у интерну листу копираних компоненти |

Табела 5.1.2. Преглед најважнијих догађаја класе *DragAndDropPanel*

Списак слика

[Слика 3.1.2. Типичан изглед екрана у алату *Visual Studio* 9](#_Toc459836153)

[Слика 3.2.1. Дефинисање шеме система у алату *Proteus Design Suite* 10](#_Toc459836154)

[Слика 4.3.1. Графички интерфеjс за дефинисање изгледа компоненти 17](#_Toc459836155)

[Слика 4.4.1. Прозор за пројектовање система 19](#_Toc459836156)

[Слика 5.1.1. Дијаграм класа за имплементацију дефинисања изгледа компоненти 24](#_Toc459836157)