|  |  |
| --- | --- |
| |  | | --- | | **УНИВЕРЗИТЕТ У НОВОМ САДУ**  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**  **НОВИ САД**  **Департман за рачунарство и аутоматику**  **Одсек за рачунарску технику и рачунарске комуникације**  **ЗАВРШНИ (BACHELOR) РАД**  **Кандидат: Филип Унчанин**  **Број индекса: РА159/2020**  **Тема рада: Аутоматизација генерисања System Verilog својстава и јединичних тестова на основу предефинисанe VHDL архитектуре употребом Вештачке интелигенције**  **Ментор рада: проф. др Небојша Пјевалица**  **Нови Сад, септембар, 2024.** | |

|  |  |  |  |
| --- | --- | --- | --- |
| Редни број, **РБР**: | |  | |
| Идентификациони број, **ИБР**: | |  | |
| Тип документације, **ТД**: | | Монографска документација | |
| Тип записа, **ТЗ**: | | Текстуални штампани материјал | |
| Врста рада, **ВР**: | | Завршни (Bachelor) рад | |
| Аутор, **АУ**: | | Филип Унчанин | |
| Ментор, **МН**: | | проф. др Небојша Пјевалица | |
| Наслов рада, **НР**: | | Аутоматизација генерисања System Verilog својстава и јединичних тестова на основу предефинисанe VHDL архитектуре употребом Вештачке интелигенције | |
| Језик публикације, **ЈП**: | | Српски / ћирилица | |
| Језик извода, **ЈИ**: | | Српски | |
| Земља публиковања, **ЗП**: | | Република Србија | |
| Уже географско подручје, **УГП**: | | Војводина | |
| Година, **ГО**: | | 2024 | |
| Издавач, **ИЗ**: | | Ауторски репринт | |
| Место и адреса, **МА**: | | Нови Сад; трг Доситеја Обрадовића 6 | |
| Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога) | | 8/42/0/1/27/0/0 | |
| Научна област, **НО**: | | Електротехника и рачунарство | |
| Научна дисциплина, **НД**: | | Рачунарска техника | |
| Предметна одредница/Кључне речи, **ПО**: | | VHDL, SystemVerilog, аутоматско прегледање испитних задатака, тестирање, јединични тестови, вештачка интелигенција, LLM, GPT Api | |
| **УДК** | |  | |
| Чува се, **ЧУ**: | | У библиотеци Факултета техничких наука, Нови Сад | |
| Важна напомена, **ВН**: | |  | |
| Извод, **ИЗ**: | | У оквиру дипломског рада реализовано је једно решење аутоматизованог генерисања испитних случајева, односно јединичних тестова у оквиру SVUnit окружења на бази постојећег референтног VHDL решења употребом комерцијалног GPT Api LLM-a | |
| Датум прихватања теме, **ДП**: | |  | |
| Датум одбране, **ДО**: | | 26.09.2024. | |
| Чланови комисије, **КО**: | Председник: | проф. др Мирослав Поповић |
|  | Члан: | проф. др Иван Каштелан | Потпис ментора |
|  | Члан, ментор: | проф. др Небојша Пјевалица |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Accession number, **ANO**: | |  | |
| Identification number, **INO**: | |  | |
| Document type, **DT**: | | Monographic publication | |
| Type of record, **TR**: | | Textual printed material | |
| Contents code, **CC**: | | Bachelor Thesis | |
| Author, **AU**: | | Filip Unčanin | |
| Mentor, **MN**: | | prof. Nebojša Pjevalica, PhD | |
| Title, **TI**: | | Automation of System Verilog assertions generation and unit tests based on predefined VHDL architecture using Artificial Intelligence | |
| Language of text, **LT**: | | Serbian | |
| Language of abstract, **LA**: | | Serbian | |
| Country of publication, **CP**: | | Republic of Serbia | |
| Locality of publication, **LP**: | | Vojvodina | |
| Publication year, **PY**: | | 2024 | |
| Publisher, **PB**: | | Author’s reprint | |
| Publication place, **PP**: | | Novi Sad, Dositeja Obradovica sq. 6 | |
| Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes) | | 8/42/0/1/27/0/0 | |
| Scientific field, **SF**: | | Electrical Engineering | |
| Scientific discipline, **SD**: | | Computer Engineering, Engineering of Computer Based Systems | |
| Subject/Key words, **S**/**KW**: | | VHDL, SystemVerilog, automatic exam review, testing, unit tests, artificial intelligence, LLM, GPT Api | |
| **UC** | |  | |
| Holding data, **HD**: | | The Library of Faculty of Technical Sciences, Novi Sad, Serbia | |
| Note, **N**: | |  | |
| Abstract, **AB**: | | As part of the thesis, a solution for automated generation of test cases, specifically unit tests within the SVUnit environment, was implemented based on the existing reference VHDL solution using a commercial GPT API LLM. | |
| Accepted by the Scientific Board on, **ASB**: | |  | |
| Defended on, **DE**: | | 26.09.2024 | |
| Defended Board, **DB**: | President: | prof. Miroslav Popović, PhD |
|  | Member: | prof. Ivan Kaštelan, PhD | Menthor's sign |
|  | Member, Mentor: | prof. Nebojša Pjevalica, PhD |  |

**Захвалност**

Желим да изразим дубоку захвалност свом ментору, проф. др Небојши Пјевалици, чија подршка и стручност су били од непроценљиве важности за израду овог дипломског рада. Такође, захваљујем докторанткињи Теодори Новковић на њеном присуству и спремности да буде део овог процеса. Велику захвалност дугујем и својим колегама Александру Обрадовићу и Јовану Гашпару за њихов увид и конструктивне коментаре. На крају, захваљујем се својој породици и пријатељима на сталној подршци, разумевању и мотивацији током целокупног процеса. Без њих, овај рад не би био могућ.

**Садржај**

[1. Увод 1](#_Toc177926552)

[2. Задатак дипломског рада 3](#_Toc177926553)

[3. Теоријске основе 4](#_Toc177926554)

[3.1 HDL језици 4](#_Toc177926555)

[3.2 VHDL 6](#_Toc177926556)

[3.2.1 Основне структуре и концепти 6](#_Toc177926557)

[3.2.2 Предности и примена 6](#_Toc177926558)

[3.2.3 Значај у савременом развоју интегрисаних кола 7](#_Toc177926559)

[3.3 SystemVerilog 7](#_Toc177926560)

[3.3.1 Интеграција и напредне карактеристике 7](#_Toc177926561)

[3.3.2 Основне структуре и концепти 8](#_Toc177926562)

[3.3.3 Предности и примена 8](#_Toc177926563)

[3.3.4 Значај у савременом развоју интегрисаних кола 8](#_Toc177926564)

[3.4 Поређење VHDL-а и SystemVerilog-а 9](#_Toc177926565)

[3.5 Проблем верификације 10](#_Toc177926566)

[3.5.1 Изазови у верификацији 11](#_Toc177926567)

[3.5.2 Методе верификације 11](#_Toc177926568)

[3.5.3 Аутоматизоване методе 11](#_Toc177926569)

[3.5.4 SystemVerilog Тврдње (Assertions) и њихова примена 11](#_Toc177926570)

[3.5.4.1 Основне карактеристике тврдњи 12](#_Toc177926571)

[3.5.4.2 Врсте тврдњи 12](#_Toc177926572)

[3.5.4.3 Примена тврдњи 12](#_Toc177926573)

[3.5.5 SVUnit - основе 14](#_Toc177926574)

[3.6 Утицај вештачке интелигенције на верификацију дигиталних система 14](#_Toc177926575)

[4. Поставка проблема 16](#_Toc177926576)

[5. Програмско решење 18](#_Toc177926577)

[5.1 Поставка ChatGPT API-ја 18](#_Toc177926578)

[5.2 Припрема SVUnit oкружења 20](#_Toc177926579)

[5.3 Припрема основних датотека 21](#_Toc177926580)

[5.4 Креирање wrapper и pack датотека 22](#_Toc177926581)

[5.5 Генерисање SystemVerilog тврдњи 24](#_Toc177926582)

[5.5.1 Текстуална интерпретација VHDL кода 24](#_Toc177926583)

[5.5.2 Генерисање SystemVerilog тврдњи уз помоћ ChatGPT API-ја 26](#_Toc177926584)

[5.5.3 Провера синтаксне исправности тврдњи 26](#_Toc177926585)

[5.6 Генерисање SVUnit Unit Test датотека 29](#_Toc177926586)

[5.6.1 Учитавање SVUnit Unit Test костура датотеке 29](#_Toc177926587)

[5.6.2 Уметање сигнала 30](#_Toc177926588)

[5.6.3 Уметање генерисаних тврдњи 32](#_Toc177926589)

[5.6.4 Уметање поворке улазних сигнала 32](#_Toc177926590)

[5.7 Провера логичке исправности тврдњи 33](#_Toc177926591)

[5.7.1 Покретање Questa симулације 33](#_Toc177926592)

[5.7.2 Анализа резултата симулације 33](#_Toc177926593)

[5.7.3 Уклањање логички неисправних тврдњи 34](#_Toc177926594)

[5.8 Креирање табеле за оцењивање 34](#_Toc177926595)

[5.9 Додела бодова тврдњама унутар SVUnit тестова 35](#_Toc177926596)

[5.10 Извршавање целокупног процеса 36](#_Toc177926597)

[5.10.1 Извршавање скрипте run\_first.bsh 36](#_Toc177926598)

[5.10.2 Извршавање скрипте run\_second.bsh 37](#_Toc177926599)

[5.11 Преглед директоријума пре и након извршавања процеса 37](#_Toc177926600)

[5.11.1 Директоријум пре извршавања 37](#_Toc177926601)

[5.11.2 Директоријум након извршавања 38](#_Toc177926602)

[5.12 Дијаграм функционалности 39](#_Toc177926603)

[6. Резултати 40](#_Toc177926604)

[7. Закључак 42](#_Toc177926605)

[8. Литература 43](#_Toc177926606)

**Списак слика**

[*Слика 5.1 Добијање API кључа на OpenAI платформи 19*](#_Toc177663349)

[*Слика 5.2 Уплата средстава на OpenAI налог 19*](#_Toc177663350)

[*Слика 5.3 svunit директоријум 21*](#_Toc177663351)

[*Слика 5.4 Припрема основних датотека 21*](#_Toc177663352)

[*Слика 5.5 Почетна VHDL датотека 22*](#_Toc177663353)

[*Слика 5.6 top\_wrapper датотека 23*](#_Toc177663354)

[*Слика 5.7 top\_tb\_wrapper датотека 23*](#_Toc177663355)

[*Слика 5.8 pack датотека 24*](#_Toc177663356)

[*Слика 5.9 Захтев за текстуалном интерпретацијом 25*](#_Toc177663357)

[*Слика 5.10 Текстуална интерпретација 25*](#_Toc177663358)

[*Слика 5.11 Захтев за генерисање тврдњи 26*](#_Toc177663359)

[*Слика 5.12 SystemVerilog костур за превођење 27*](#_Toc177663360)

[*Слика 5.13 Превођење тврдњи 28*](#_Toc177663361)

[*Слика 5.14 Извршавање скрипте за генерисање тврдњи 28*](#_Toc177663362)

[*Слика 5.15 Пример генерисаних синтаксно исправних тврдњи 29*](#_Toc177663363)

[*Слика 5.16 Костур SVUnit тестне датотеке 30*](#_Toc177663364)

[*Слика 5.17 Сигнали унутар SVUnit тестне датотеке 31*](#_Toc177663365)

[*Слика 5.18 Поворка улазних сигнала из top\_all\_tb.vhd датотеке 32*](#_Toc177663366)

[*Слика 5.19 Поворка улазних сигнала у SVUnit тестној датотеци 33*](#_Toc177663367)

[*Слика 5.20 Табела својстава са бодовима 35*](#_Toc177663368)

[*Слика 5.21 run\_first.bsh скрипта 36*](#_Toc177663369)

[*Слика 5.22 Бодови додељени тврдњама 37*](#_Toc177663370)

[*Слика 5.23 Садржај директоријума пре извршавања 37*](#_Toc177663371)

[*Слика 5.24 Садржај директоријума након извршавања 38*](#_Toc177663372)

[*Слика 5.25 Дијаграм функционалности 39*](#_Toc177663373)

[*Слика 6.1 Резултати коришћењем претходног решења 40*](#_Toc177663374)

[*Слика 6.2 Резултати коришћењем тренутног решења 40*](#_Toc177663375)

**Списак табела**

[*Табела 1 Поређење VHDL-а и SystemVerilog-а 10*](#_Toc176976636)

**Скраћенице**

**HDL** – *Hardware Description Language*, уопштен назив за језике за описивање хардвера

**VHDL** – *Very High-Speed Integrated Circuit Hardware Description Language*, језик за описивање хардвера

**SVA** – *SystemVerilog Assertions*, систем верилог тврдњe

**LLM** – *Large Language Models*, врста алгоритма вештачке интелигенције који користи технике дубоког учења за разумевање, генерисање и предвиђање новог садржаја

**DUT** – *Device under-test*

# Увод

Развој интегрисаних кола и дигиталних система довео је до значајних промена у начину на који се ти системи пројектују, анализирају и верификују. Од самих почетака развоја електронских компоненти, методе и технике описа хардверских ентитета еволуирале су како би се прилагодиле све већем нивоу сложености система. Почевши од ручних шематских приказа, који су доминирали у раним фазама развоја, индустрија је усвојила језике за опис хардвера (HDL) као средство за апстракцију функционалности дигиталних система. Међу овим језицима, два најзаступљенија су VHDL и Verilog, који се данас користе широм света у развоју и верификацији интегрисаних кола.

Значај описа дигиталних система у HDL језицима огледа се у могућности да се дигитални системи опишу на високом нивоу апстракције. Ови језици омогућавају пројектантима да се фокусирају на функционалност система, без потребе за детаљним бављењем физичком реализацијом сваког логичког блока. VHDL и Verilog су развијени са циљем да олакшају процес пројектовања, али и да омогуће ефикаснију верификацију хардверских решења. Овај корак је био кључан у преласку са шематског на текстуални опис хардвера, што је уједно и омогућило већу флексибилност у моделовању и модуларност у пројектовању система.

Проблем верификације HDL решења постаје све важнији како расте сложеност интегрисаних кола. Савремени дигитални системи, као што су процесорска језгра, меморијски подсистеми и периферне јединице, захтевају детаљну верификацију свих својих делова како би се осигурала исправност и поузданост целокупног система. У индустријским условима, процењује се да верификација учествује са више од 70% укупних трошкова развоја интегрисаних кола. Ово чини верификацију једним од најкритичнијих аспеката у процесу развоја. Грешке у развоју интегрисаних кола могу бити изузетно скупе, како у смислу новчаних губитака, тако и у погледу времена потребног за исправљање тих грешака.

Аутоматизација верификације HDL решења представља решење за многе проблеме са којима се сусрећу инжењери. Са порастом сложености система, ручна верификација постаје све тежа и дуготрајнија. Због тога су развијени алати који аутоматизују процес верификације. Један од најзначајнијих доприноса на овом пољу је развој језика SystemVerilog, који је проширио могућности језика Verilog укључивањем објектно оријентисаних метода и функционалности за верификацију. Ови нови приступи омогућавају стварање свеобухватних верификационих платформи које могу покрити све аспекте сложених дигиталних система.

# Задатак дипломског рада

Аутоматизација процеса прегледања и оцењивања студентских радова писаних у језику за опис хардвера VHDL представља важан корак ка унапређењу ефикасности и тачности у верификацији ових решења. Студенти, као део својих испитних задатака, пишу VHDL модуле и основни део тест-бенча који служи као демонстрација њиховог знања. Међутим, ручно проверавање тих решења може бити веома захтеван процес, јер захтева детаљну верификацију рада сваког модула.

У оквиру претходног истраживања показало се да System Verilog језик, односно у њему развијени концепт јединичног тестирања (SVUnit testing), базиран на тврдњама (System Verilog Assertions – SVA) пружа широк спектар могућности за проверу функционалности дигиталних система.

Ослањајући се на наведене резултате потребно је испитати технику употребе великих језичких модела из домена Вештачке интелигенције (Large Language Model - LLM) у правцу креирања потребних тврдњи и јединичних тестова за проверу функционалности испитиваних VHDL модула. Фокусирати употребу ChatGPT комерцијалне платформе као једне од водећих у датом домену.

# Теоријске основе

## HDL језици

Hardware Description Languages (HDL) представљају срце савременог описа дигиталних система, омогућавајући инжењерима да опишу, моделују и анализирају комплексне електронске уређаје и интегрисана кола. Ови језици су неопходни за стварање детаљних спецификација које превазилазе границе физичких компоненти, омогућавајући развој хардверских решења са високим степеном прецизности и ефикасности.

У основи, HDL језици нуде различите нивое апстракције за моделовање дигиталних система. Структурална апстракција омогућава пројектантима да јасно дефинишу како су компоненте система међусобно повезане. Ово укључује детаље о логичким блоковима као што су регистри, мултиплексери и логичке капије, као и њихове међусобне везе. Овакав приступ омогућава креирање физички верних приказа дигиталних кола, што је кључно за разумевање како ће систем радити у стварном свету.

Са друге стране, апстракција описа понашања фокусира се на описивање функционалности система, занемарујући детаље физичке имплементације. Ова врста апстракције користи изразе високог нивоа за спецификацију како систем реагује на улазе и производи излазе. Ово омогућава инжињерима да креирају моделе који су лакши за разумевање и бржи за развој, док се детаљи имплементације остављају за касније фазе пројектовања.

Временска апстракција игра кључну улогу у моделовању и верификацији дигиталних система. Ова врста апстракције помаже инжењерима да анализирају како различита кашњења и периоди утичу на особине система. Разумевање времена које је потребно за обраду сигнала и извршење операција омогућава прецизно прилагођавање решења и осигурање да ће систем радити у складу са захтевима.

Једна од највећих предности HDL језика је њихова способност да омогуће аутоматску синтезу. Алати за синтезу могу интерпретирати HDL опис и генерисати физичке компоненте и њихово повезивање, што значајно убрзава процес развоја и смањује потребу за мануелним радом. Ово не само да штеди време, већ и смањује могућност људских грешака.

Осим тога, HDL модели се користе за симулацију и тестирање пре него што се физички реализују. Симулација омогућава пројектантима да уоче грешке и оптимизују решење, чиме се значајно побољшава поузданост и ефикасност коначног производа. Флексибилност HDL језика омогућава лака прилагођавања и проширења, што је кључно за рад у брзо мењајућем технолошком окружењу.

У процесу развоја дигиталних система помоћу HDL језика, обично се пролази кроз неколико кључних фаза. Почетна фаза је спецификација, у којој се дефинишу захтеви и функционалности система. Након тога следи моделовање, где се развијају HDL модели који описују структуру и понашање система. Симулација омогућава тестирање ових модела, док синтеза преводи HDL опис у физичке компоненте. Коначна фаза је верификација која осигурава да решење задовољава све захтеве и функционалне спецификације.

Hardware Description Languages (HDL) су, дакле, основни алат у модерном развоју дигиталних система, пружајући инжењерима моћне методе за моделовање, симулацију и верификацију комплексних кола. Разумевање основних карактеристика и предности ових језика пружа чврсту основу за даља истраживања и примену у развоју дигиталних система, као и за разумевање специфичних језика као што су VHDL и SystemVerilog.

## VHDL

VHDL, што је акроним за VHSIC Hardware Description Language, представља кључни језик у области развоја дигиталних система. Развијен током 1980-их година као део пројекта VHSIC (Very High Speed Integrated Circuit) у Сједињеним Државама, VHDL је направљен да омогући инжењерима тачно и детаљно описивање сложених електронских уређаја и интегрисаних кола. Његова свестрана природа и прецизност учинили су га основним алатом у савременом развоју дигиталних система.

### Основне структуре и концепти

VHDL нуди богат сет конструкција које омогућавају прецизно моделовање дигиталних система. Основне структуре у VHDL-у укључују ентитете и архитектуре. Ентитет представља спољашњи интерфејс дигиталног уређаја, укључујући све улазне и излазне сигнале. Ово је основа на којој се гради, а представља спољашњи изглед и функционалност компонената. Архитектура, с друге стране, описује унутрашњу структуру ентитета, укључујући како се различити делови повезују и функционишу. Ова подела омогућава пројектантима да одвоје опис спољашњег интерфејса од унутрашњег рада компоненте, чиме се побољшава јасноћа и управљивост кода.

Још један важан аспект VHDL-а су сигнали и варијабле. Сигнали у VHDL-у представљају средства за комуникацију између различитих делова система, док варијабле служe као привремени носиоци података који се користе унутар једног модела. Ова разлика омогућава ефикасно управљање подацима и њихову обраду у оквиру различитих делова система.

### Предности и примена

VHDL је посебно цењен због своје способности да омогући детаљно описивање сложених система и пружи јасну документацију о свим аспектима решења. Једна од кључних предности VHDL-а је његова способност да омогући верификацију решења пре него што се физички реализује. Користећи симулацију, инжињери могу тестирати и анализирати како ће систем функционисати под различитим условима, што помаже у идентификацији потенцијалних проблема и оптимизацији решења. Овај процес значајно смањује ризик од грешака у каснијим фазама развоја и омогућава уштеду времена и ресурса.

Током пројектовања дигиталних система, VHDL се користи за развој одређених модела и прототипова који се могу тестирати и усавршавати пре него што се крене у физичку имплементацију. Ова могућност омогућава инжењерима да лакше управљају комплексним пројектима и обезбеде да сви аспекти система буду адекватно покривени и проверени.

### Значај у савременом развоју интегрисаних кола

Вредност VHDL-а у савременом развоју дигиталних система не може се потценити. Његова способност да пружи прецизно и свеобухватно описивање система чини га непроцењивим алатом за инжењере који се баве развојем сложених електронских уређаја. Чврста структура и детаљно описивање компоненти и њихових веза омогућавају развој висококвалитетних решења и осигурање да сви аспекти система функционишу у складу са очекивањима.

Укратко, VHDL је основни језик који омогућава прецизно и ефикасно описивање дигиталних система, пружајући инжењерима моћне алате за развој, тестирање и верификацију сложених уређаја.

## SystemVerilog

SystemVerilog представља један од најновијих и најсавременијих језика за описивање дигиталних система, који је развијен као надоградња на ранији Verilog. Од свог увођења, SystemVerilog је постао стандард у индустрији за пројектовање и верификацију сложених електронских уређаја, због своје способности да комбинује могућности описивања хардвера са напредним функцијама верификације.

### Интеграција и напредне карактеристике

SystemVerilog је осмишљен да уједини најбоље карактеристике из различитих језика за описивање хардвера. Његов главни циљ је да побољша развој и верификацију дигиталних система, чинећи их ефикаснијим и прецизнијим. Једна од кључних карактеристика SystemVerilog-а је његова способност да комбинује описивање хардвера на ниском нивоу са моћним алатима за тестирање и верификацију.

### Основне структуре и концепти

SystemVerilog нуди широк спектар конструкција и концепата који су осмишљени да побољшају и прошире функционалности основног Verilog језика. Основне структуре у SystemVerilog-у укључују:

* Модули и Интерфејси: Модули у SystemVerilog-у представљају основне градивне блокове система, слично као у Verilog-у. Интерфејси, с друге стране, омогућавају лакше управљање сложеним сигналима и њиховим везама између различитих компоненти система. Интерфејси омогућавају груписање сродних сигнала и пружају јаснији начин управљања и комуникације између модула.
* Класе и Објектно-Оријентисано Програмирање: SystemVerilog уводи концепте објектно-оријентисаног програмирања, као што су класе и наслеђивање, што омогућава развој напредних модела и верификационих средстава. Класе омогућавају креирање сложених структура података и дефинисање метода који могу манипулисати тим подацима, чиме се побољшава могућност управљања и тестирања система.
* Верификација и тврдње: Једна од најзначајнијих надоградњи SystemVerilog-а је његова способност да укључи напредне функције верификације. Ово укључује употребу тврдњи (Assertions) које омогућавају дефинисање правила и услова који морају бити испуњени током симулације. Тврдње помажу у идентификацији и отклањању грешака и неправилности у систему пре него што се приступи физичкој имплементацији.

### Предности и примена

SystemVerilog доноси бројне предности у развоју и верификацији дигиталних система. Његова способност да интегрише описивање хардвера са напредним верификацијским алатима чини га изузетно корисним за развој сложених уређаја и система. Примена SystemVerilog-а омогућава брже и прецизније тестирање система, што смањује време развоја и побољшава квалитет коначних производа.

### Значај у савременом развоју интегрисаних кола

SystemVerilog игра кључну улогу у савременом развоју дигиталних система, посебно у контексту верификације сложених интегрисаних кола. Његова напредна функција верификације и објектно-оријентисаног програмирања чини га неопходним алатом за инжењере који се баве развојем и тестирањем висококвалитетних електронских уређаја. Његова способност да уједини синтезу и верификацију у један оквир омогућава боље управљање комплексношћу и повећава ефикасност у процесу развоја.

Укратко, SystemVerilog представља значајан корак напред у језицима за описивање хардвера, нудећи напредне функције које омогућавају прецизније и ефикасније пројектовање и тестирање дигиталних система. Његова примена у савременом инжењерингу електронских уређаја чини га кључним алатом у развоју сложених и система високих перформанси.

## Поређење VHDL-а и SystemVerilog-а

|  |  |  |
| --- | --- | --- |
| **Фактор** | **VHDL** | **SystemVerilog** |
| **Главна сврха** | Примарно се користи за описивање дигиталних система, са нагласком на формалне спецификације и симулације. | Спецификација, верификација и синтеза дигиталних система. Пружа напредне технике за верификацију и подршку за сложене пројекта. |
| **Структура језика** | Заснован је на структурама као што су ентитети (спољни интерфејси) и архитектуре (унутрашња имплементација), што омогућава јасну поделу између спољног и унутрашњег дела. | Користи модуле као основне грађевинске блокове, али уводи и интерфејсе за лакшу комуникацију међу компонентама, као и класе за напредне објекте и структуре података. |
| **Модуларност** | VHDL нуди високу модуларност кроз јасну сепарацију ентитета и архитектура, што омогућава лако раздвајање и поновну употребу компоненти. | Такође омогућава модуларност, али са додатним предностима интерфејса и објектно-оријентисаних принципа. Лакше је управљати великим системима и комплексним комуникацијама међу модулима. |
| **Верификација** | Верификација у VHDL-у је често базирана на писању тестбенчева који се користе за симулацију. Ова метода може бити ограничена за комплексне системе. | Има напредне функције верификације кроз SystemVerilog Assertions (SVA), као и уграђену подршку за аутоматизовано тестирање. |
| **Типови података** | VHDL има богат сет типова података, укључујући стандардне типове (integer, real, bit, итд.) и кориснички дефинисане типове. Типови података су строго типизовани. | SystemVerilog проширује типове података из Verilog-а, додајући нове типове као што су randc за случајне бројеве и struct за сложене податке. Пружа и подршку за недељиво типизовање и напредне врсте података. |
| **Синтакса** | Синтакса VHDL-а је строга и формална, често комплексна за новајлије, али омогућава прецизну спецификацију. | Синтакса SystemVerilog-а је флексибилнија и сличнија Verilog-у, али са додатним проширењима и поједностављењима која омогућавају лакше коришћење. |
| **Објектно-оријентисано програмирање** | VHDL не пружа директну подршку за објектно-оријентисано програмирање. | SystemVerilog укључује подршку за ООП са могућностима као што су класе, наслеђивање, енкапсулација и полиморфизам, што омогућава напредне технике синтезе и верификације. |
| **Библиотеке и модули** | Користи пакете за груписање типова података, функција и компоненти, али пакети нису толико флексибилни као модули у SystemVerilog-у | Користи модуларност кроз богате и флексибилне модуле и интерфејсе, који омогућавају ефикаснију организацију великих пројеката |
| **Подршка за симулацију** | Има добру подршку за симулацију, али често се ослања да додатне алате за верификацију сложенијих система. | Има интегрисану подршку за симулацију и верификацију, укључујући аутоматско генерисање тест сценарија кроз алате као што је UVM |
| **Синтеза** | VHDL је добро подржан за синтезу, посебно за структуриране и формално описане пројекте. | SystemVerilog такође пружа добру подршку за синтезу, али је фокусиран на сложенија системе и напредне могућности верификације. |
| **Подршка за вишеструке тактове (clock domains)** | VHDL подржава вишеструке тактове, али може захтевати додатне технике за синхронизацију и управљање различитим тактовима. | SystemVerilog нуди бољу подршку за вишеструке тактове и асинхроне системе, са напредним техникама за синхронизацију и управљање clock domains. |
| **Интероперабилност** | VHDL се може користити заједно са Verilog-ом, али интеграција може бити изазовна и захтева посебне алате или приступе. | SystemVerilog је развијен да буде компатибилан са Verilog-ом, што омогућава лакшу интероперабилност и интеграцију у mixed-language пројектима. |
| **Симулацијски модели** | VHDL омогућава писање различитих нивоа симулацијских модела, али може бити сложен за имплементацију напредних тестбенчева. | SystemVerilog пружа напредне функционалности за писање симулацијских модела и тестбенчева, укључујући могућност писања функционалних и покретачких модела у оквиру исте синтаксе. |
| **Подршка за хардверске описе и паралелизам** | VHDL омогућава паралелно извршавање компоненти кроз описивање процеса у различитим архитектурама. Међутим, управљање паралелизмом може бити компликовано. | SystemVerilog нуди јаснију и лакшу подршку за паралелизам и конкурентно извршавање, укључујући и контекстуалне задатке и променљиве. |
| **Подршка за интелигентно тестирање** | VHDL нуди основне могућности за тестирање, али напредне технике као што су случајне провере нису део стандардног језика. | SystemVerilog пружа напредне технике за интелигентно тестирање, укључујући случајне генераторе тестова и констрентно задовољавање. |
| **Будући развој** | VHDL је стабилан, али његов развој је спорији у поређењу са новијим језицима. Стандарди се мењају споро. | SystemVerilog је активно развијан, са редовним ажурирањима и додатним функционалностима. Развија се како би задовољио потребе модерне верификације и синтезе. |

*Табела 1 Поређење VHDL-а и SystemVerilog-а*

## Проблем верификације

Верфикација представља један од најкритичнијих аспеката у развоју дигиталних система и интегрисаних кола. Као процес који осигурава да решење испуњава све спецификације и захтеве, верфикација је кључна за осигурање функционалности и поузданости електронских уређаја пре него што се они физички реализују. Овај процес се бави провером исправности и ефикасности решења, што укључује отклањање грешака, оптимизацију перформанси и осигурање компатибилности са спецификацијама.

### Изазови у верификацији

Верфикација је сложен и захтеван процес, који се суочава са бројним изазовима. Један од главних изазова је управљање комплексношћу система. Како системи постају све сложенији, са великим бројем компоненти и интеракција, тако и процес верификације постаје компликованији. Комплексна решења често укључују велики број сигнала и функционалних блокова, што отежава тестирање свих могућих сценарија и услова. Овај проблем се додатно компликује када се узму у обзир различите врсте грешака које могу настати, од функционалних грешака до временских проблема.

### Методе верификације

Да би се успешно решили ови изазови, различите методе верификације се користе у индустрији. Традиционално, процес верфикације укључује ручно тестирање и симулацију решења. Ове методе могу бити успешне, али су често временски интензивне и подложне људским грешкама.

Симулација је један од основних алата у процесу верификације, где се решење моделује и тестира у виртуелном окружењу. Овај процес омогућава инжењерима да испитају како решење функционише под различитим условима и оптерећењима. Међутим, симулација сама по себи није довољна, јер не може увек у потпуности покрити све могуће сценарије и комбинације улазних услова.

### Аутоматизоване методе

Успех у верификацији често зависи од примене аутоматизованих метода. Употреба напредних алата и техника, као што су верификационе библиотеке, генеричке тест стратегије и тврдње, помажу у убрзању и побољшању процеса тестирања. Аутоматизоване методе омогућавају детаљније и свеобухватније тестирање решења, узимајући у обзир велики број могућих сценарија и услова.

### SystemVerilog Тврдње (Assertions) и њихова примена

Тврдње су један од кључних елемената у процесу верификације дигиталних система, посебно у контексту тестирања и осигуравања исправности решења. Оне представљају механизам који омогућава дефинисање и праћење критичних услова и правила у систему, како би се осигурало да решење функционише у складу са спецификацијама. Тврдње су посебно важне, јер помажу у идентификацији и корекцији грешака у раној фази развоја, чиме значајно побољшавају квалитет и поузданост електронских уређаја.

#### Основне карактеристике тврдњи

Тврдње су конструкције у језицима за описивање хардвера које дефинишу одређене услове који морају бити испуњени током симулације. Оне се користе за проверу тачности и исправности функционалности система. Основне карактеристике тврдњи укључују:

* Условна Провера: Тврдње омогућавају дефинисање услова који морају бити испуњени у одређеним временским интервалима или у одговору на специфичне догађаје. Када услови нису испуњени, тврдња ће покренути обавештење о грешци, чиме се омогућава брза идентификација проблема.
* Проактивно откривање грешака: Тврдње омогућавају проактивно откривање грешака и неправилности у систему. Оне могу бити укључене у различите делове развоја и симулације, чиме се повећава могућност уочавања проблема пре него што дођу до физичке имплементације.
* Документација и разумевање: Тврдње пружају јасан начин за документовање очекиваног понашања система. Оне могу служити као важан део документације и помоћи у разумевању решења и његових очекивања.

#### Врсте тврдњи

У SystemVerilog-у, тврдње се могу класификовати у две категорије:

* Тренутне тврдње (Immediate assertions): Ове тврдње се извршавају одмах када се наиђе на њих током симулације. Користе се за проверу услова који морају бити испуњени у том тренутку. Ако услов није испуњен, тренутна тврдња одмах генерише грешку. Тренутне тврдње су погодне за проверу услова који се односе на тренутну вредност сигнала.
* Конкурентне тврдње (Concurrent assertions): Ове тврдње се користе за праћење услова током времена и проверавају односе између сигнала у више симулационих циклуса. Конкурентне тврдње омогућавају проверу сложених секвенци догађаја и временских услова. Погодне су за верификацију понашања система које се одвија у више временских тачака или током више стања у систему.

#### Примена тврдњи

Тврдње могу бити примењене на различите начине у процесу развоја и верфикације. Њихова примена укључује:

* Верификација функционалности: Тврдње се користе за проверу функционалности система, како би се осигурало да све компоненте и модули функционишу у складу са спецификацијама.
* Откривање грешака у раној фази: Укључивање тврдњи у симулације помаже у раном откривању грешака и неправилности. Овакав приступ омогућава брзо идентификовање и исправљање проблема пре него што решење пређе у физичку имплементацију.
* Оптимизација решења: Тврдње могу помоћи у оптимизацији решења тако што пружају повратне информације о томе како се систем понаша под различитим условима и оптерећењем.
* Документација и опис решења: Тврдње служe као део документације која помаже у разумевању решења и његових функционалности. Оне могу пружити јасан опис очекиваног понашања система.

Пример једноставне тврдње у SystemVerilog-у може бити:

Ова тврдња проверава да ли су сигнал *a* и *b* једнаки на сваком позитивном рубу сигнала *clk*. Ако услов није испуњен, тврдња ће генерисати грешку.



Комплексније тврдње могу укључивати временске услове и комбинације више сигнала, као што је:



Ова тврдња провера да ли се сигнал *b* активира у интервалу од једног до три такта након што је сигнал *a* активиран, под условом да сигнал *reset\_n* није активан.

Тврдње су од кључне важности у савременом развоју дигиталних система, јер пружају моћан механизам за проверу и осигурање исправности решења. Њихова способност да дефинишу и провере услове и правила у систему омогућава инжењерима да идентификују и исправе грешке у раној фази развоја, побољшавајући тако квалитет и поузданост електронских уређаја. Уз напредне функције и разноврсне примене, тврдње играју кључну улогу у модерном процесу верфикације и развоја дигиталних система.

### SVUnit - основе

SVUnit је оквир за верификацију који омогућава инжењерима да пишу и извршавају тестове за решења у језику SystemVerilog. Фокусиран је на тестирање јединица (unit testing), што значи да омогућава тестирање појединачних модула у изолацији од остатка решења. SVUnit поједностављује писање тестова коришћењем стандардних SystemVerilog метода и структура као што су тврдње, интерфејси и task-ови.

Једна од главних предности SVUnit-а је његова подршка за аутоматизовано тестирање, што омогућава покретање великог броја тестова без потребе за ручним управљањем сваким појединачним тестом. Такође, подржава генерисање XML извештаја у формату који је компатибилан са алатима за континуирану интеграцију, чиме се олакшава интеграција са софтверским развојним токовима. Једна од највећих предности SVUnit-а је интеграција са алатима за симулацију као што су ModelSim и Questa, омогућавајући корисницима да брзо покрену и провере своје тестове.

SVUnit користи једноставну хијерархију која укључује јединицу под тестирањем (Device Under Test, DUT) и тест јединицу (Unit Test), која садржи SystemVerilog тврдње, task-ове за покретање и валидацију различитих сценарија. Развијен је тако да се лако интегрише у алате као што су Questa и VCS.

Овај оквир посебно је користан у ситуацијама када је потребно брзо креирати и извршавати мале, фокусиране тестове, што га чини идеалним за тестирање решења у раним фазама развоја, али и за регресивно тестирање приликом увођења измена у постојећи код.

## Утицај вештачке интелигенције на верификацију дигиталних система

Утицај вештачке интелигенције (LLM) на верификацију дигиталних система огледа се у значајном унапређењу ефикасности и аутоматизацији процеса тестирања и потврђивања исправности хардверских решења. У традиционалним приступима верификацији, инжењери су ручно развијали тестове, писали верификационе сценарије и анализирали резултате симулација. Овај процес је био временски захтеван, а људске грешке су често утицале на коначне резултате. С појавом LLM-а, многе од ових активности могу се аутоматизовати и убрзати.

LLM модели омогућавају брже генерисање верификационих тестова, аутоматизацију исправљања кода, као и анализу симулационих резултата. Захваљујући могућности обраде великог броја података и анализе сложених шема кода, LLM може да идентификује потенцијалне грешке у решењу, предложи оптимизације, па чак и аутоматски креира нове тестове на основу пројектних захтева. Овај процес значајно смањује потребу за ручним уносом и анализом, што доприноси скраћивању времена потребног за комплетну верификацију.

Једна од кључних предности примене LLM-а у верификацији је могућност анализе кода на више нивоа, од функционалног тестирања појединачних модула до анализе комплексних сценарија у симулацији интегрисаних система. LLM модели могу аутоматски генерисати тест сценарије на основу спецификација или чак на основу анализе претходних грешака у систему. Ово омогућава да се раније идентификују потенцијални пропусти у развоју и осигура да систем испуњава све потребне стандарде квалитета.

Друга важна карактеристика је могућност аутоматизованог креирања верификационих извештаја. Раније су инжењери морали ручно да припремају ове извештаје, што је захтевало значајно време и труд. Са применом LLM-а, системи могу аутоматски генерисати извештаје који укључују анализу резултата, идентификацију грешака и препоруке за побољшања. Овај процес је не само бржи, већ и прецизнији јер елиминише могућност људских грешака у анализи и интерпретацији података.

Осим тога, LLM може бити коришћен и за подршку у комуникацији и сарадњи међу тимовима који раде на верификацији. Модели обраде природног језика могу аутоматски креирати документацију која је лако разумљива и која може бити корисна за различите стручњаке, од инжењера до менаџера пројеката. Ово значајно побољшава проток информација и омогућава бољу сарадњу међу тимовима.

Уз све наведено, LLM модели су посебно корисни у контексту сложених дигиталних система где ручна верификација постаје скоро немогућа због обима података који морају бити анализирани. Вештачка интелигенција може да обради ове податке у реалном времену и предложи решења која би иначе захтевала недеље анализе. Ово не само да убрзава процес, већ и омогућава да се пројекти заврше у краћем временском року уз већу поузданост и квалитет коначног производа.

На крају, вештачка интелигенција и LLM модели не само да побољшавају ефикасност и тачност верификације, већ и мењају начин на који се верификација обавља, доносећи нови ниво аутоматизације који ће постати стандард у будућности развоја дигиталних система.

# Поставка проблема

У области образовања и технолошког развоја, ефикасно и тачно оцењивање студентских радова представља значајан изазов. Овај проблем постаје нарочито изражен у техничким дисциплинама као што су дигитални системи, где студенти демонстрирају своје способности кроз писање VHDLкода и test-bench-а. Традиционални процес прегледања и бодовања ових радова је сложен и временски захтеван, јер захтева детаљну анализу сваког појединачног решења, што може довести до грешака и одлагања.

Студенти на испиту добијају задатак да напишу VHDLрешење и test-bench који покрива део функционалности на основу текстуалног описа. Процес ручне провере и бодовања ових радова подразумева велико време и труд за особу задужену за оцењивање. Овакав приступ може довести до значајних проблема у временској ефикасности и прецизности оцењивања, посебно када је у питању велики број студената.

Интеграција SystemVerilog-а и његових тврдњи (SVA) представља иновацију у решавању овог проблема. Креирање SVUnit Unit Test-ова који се покрећу над сваким студентским решењем омогућава аутоматизовану процену тачности и функционалности кода. Тврдње (SVA), као специјализоване конструкције у SystemVerilog-у, могу дефинисати услове који морају бити испуњени током симулације. Ова метода омогућава брзо откривање и идентификацију грешака без потребе за ручним прегледањем.

Један од значајних проблема је велики обим времена који се троши на ручно писање тврдњи и креирање unit test-ова. Писање тврдњи (SVA) захтева дубоко разумевање сваког аспекта система и пажљиво дефинисање услова и правила, што може довести до значајног оптерећења за особу задужену за овај задатак. Ово не само да одузима време, већ и представља ризик од људских грешака, које могу утицати на квалитет оцењивања.

Решење које се предлаже укључује коришћење вештачке интелигенције за аутоматско генерисање SystemVerilogтврдњи. Коришћењем напредних алгоритама и техника машинског учења, наш систем може аутоматски креирати тврдње на основу задатих спецификација и примера. Ово не само да убрзава процес, већ и смањује могућност грешака, чиме се осигурава већа тачност и поузданост у процени студенских решења.

Такође, систем ће аутоматски креирати SVUnit Test-ове на основу референтних датотека и генерисаних тврдњи. Ово значајно смањује потребу за ручним радом и омогућава брзу и ефикасну оцену великог броја студенских радова.

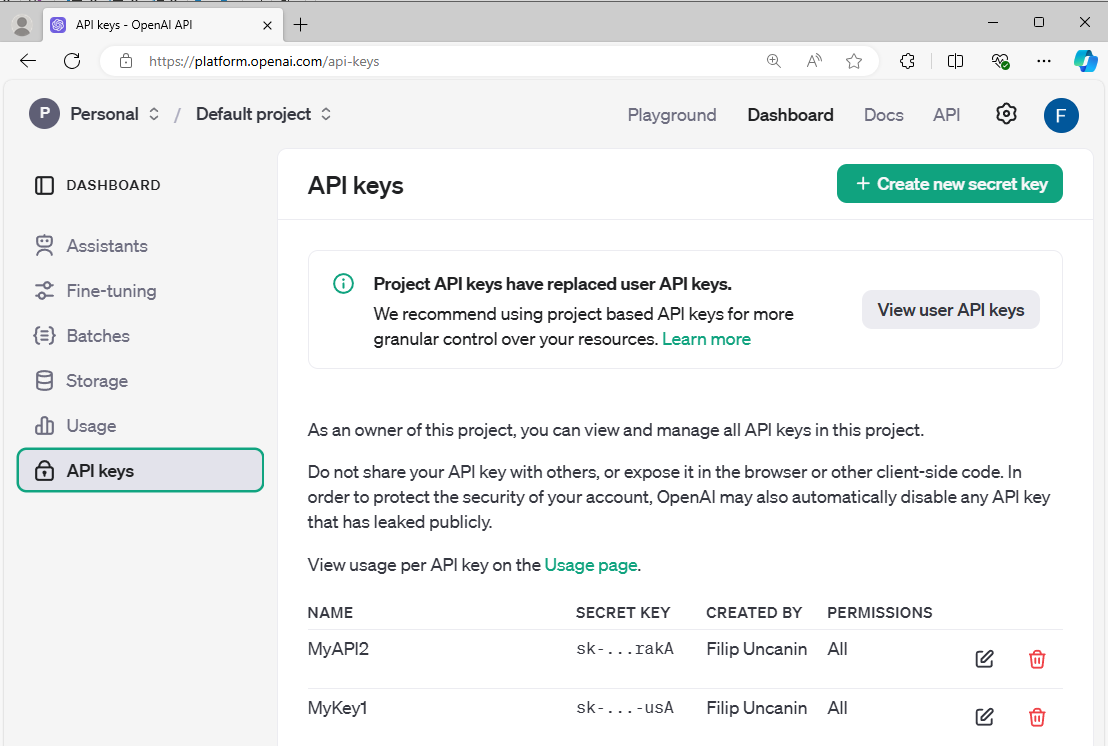
# Програмско решење

У оквиру овог поглавља описан је процес аутоматизације генерисања SystemVerilog тврдњи и креирања SVUnit Unit Test-ова. Ово решење се заснива на употреби вештачке интелигенције са циљем олакшавања и убрзавања процеса верификације студентских VHDL модула. Свака од фаза развоја програмског решења, од постављања API-ја до интеграције у симулационо окружење, детаљно је описана у наредним секцијама.

## Поставка ChatGPT API-ја

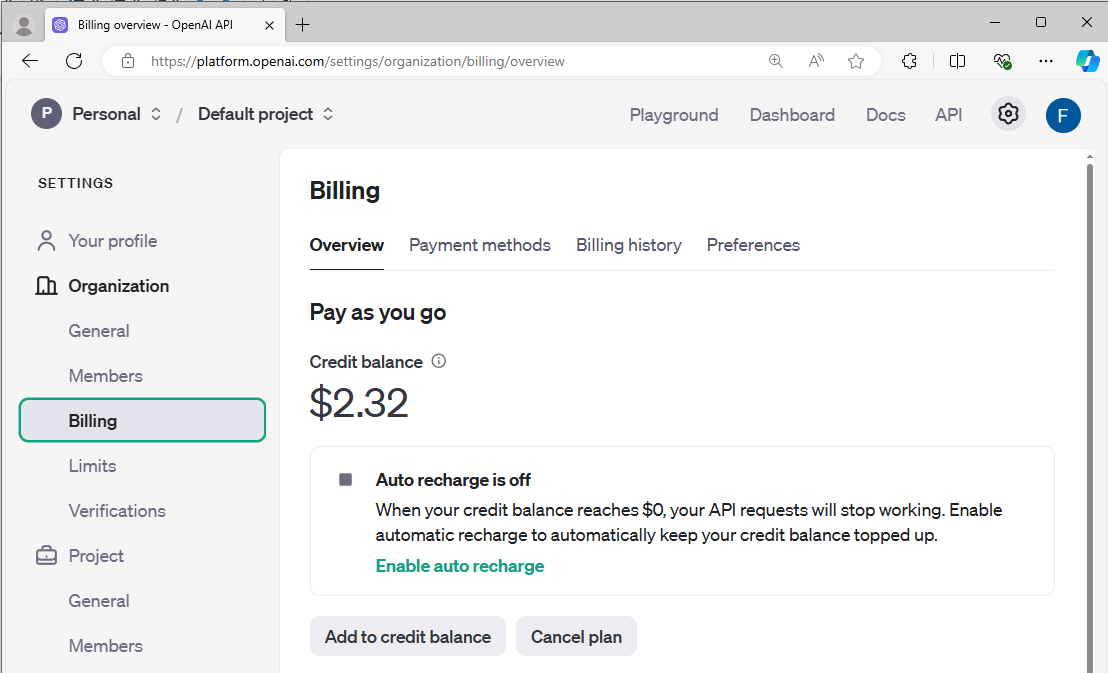
Први корак у коришћењу ChatGPT API-ја подразумева постављање програмског окружења које омогућава приступ и комуникацију са API-јем. Овај процес укључује следеће кораке:

1. Креирање налога на OpenAI платформи: За приступ ChatGPT API-ју потребно је прво креирати налог на OpenAI платформи. Ово се може урадити посећивањем званичне OpenAI веб странице (<https://platform.openai.com/>) и праћењем упутстава за регистрацију. Важно је користити важећу имејл адресу ради потврде налога и управљања подешавањима.
2. Добијање API кључа за аутентификацију: Након креирања и верификације налога, потребно је генерисати API кључ који ће се користити за аутентификацију приликом сваког позива API-ју. Овај кључ служи као јединствени идентификатор за остваривање комуникације са OpenAI услугама. API кључ се може пронаћи у секцији „API Keys“ на OpenAI платформи, а важно је чувати га на сигурном месту.



*Слика 5.1 Добијање API кључа на OpenAI платформи*

1. Уплата средстава на налог: Пошто сваки позив ка ChatGPT API-ју захтева одређена средства, неопходно је извршити уплату на налог. Наплата се врши на основу броја обрађених токена по позиву, па је потребно обезбедити довољно кредита како би сви захтеви били успешно обрађени. Плаћање се врши кредитном картицом или другим подржаним методама директно преко OpenAI платформе



*Слика 5.2 Уплата средстава на OpenAI налог*

1. Инсталација неопходних библиотека: За интеграцију ChatGPT API-ја у Python окружење, потребно је инсталирати openai библиотеку. Ово се постиже коришћењем Python-овог менаџера пакета pip. Команда за инсталацију је:

> pip install openai

1. Чување API кључа у *.env* датотеци: За безбедно складиштење API кључа, препоручује се његово чување у *.env* датотеци која је смештена на путањи *automation\_script/scripts/.env*. Унутар те датотеке, API кључ се дефинише на следећи начин: *OPENAI\_API\_KEY=генерисани\_апи\_кључ* .

Да би се кључ учитао у Python скриптама, потребно је користити библиотеку python-dotenv, која се може инсталирати помоћу следеће команде:

> pip install python-dotenv

Након тога, у Python коду се користи следећи начин за учитавање API кључа:



Овај приступ обезбеђује да се API кључ не налази директно у коду, чиме се повећава безбедност пројекта.

## Припрема SVUnit oкружења

Да би се припремило окружење за рад са SVUnit алатом, неопходно је извршити следеће кораке:

1. Клонирање репозиторијума са GitHub-а: Први корак у подешавању SVUnit окружења јесте клонирање званичног SVUnit репозиторијума са GitHub-а. Ово се постиже коришћењем git алата, који преузима комплетан садржај репозиторијума на локални рачунар. Команда за клонирање репозиторијума изгледа овако:

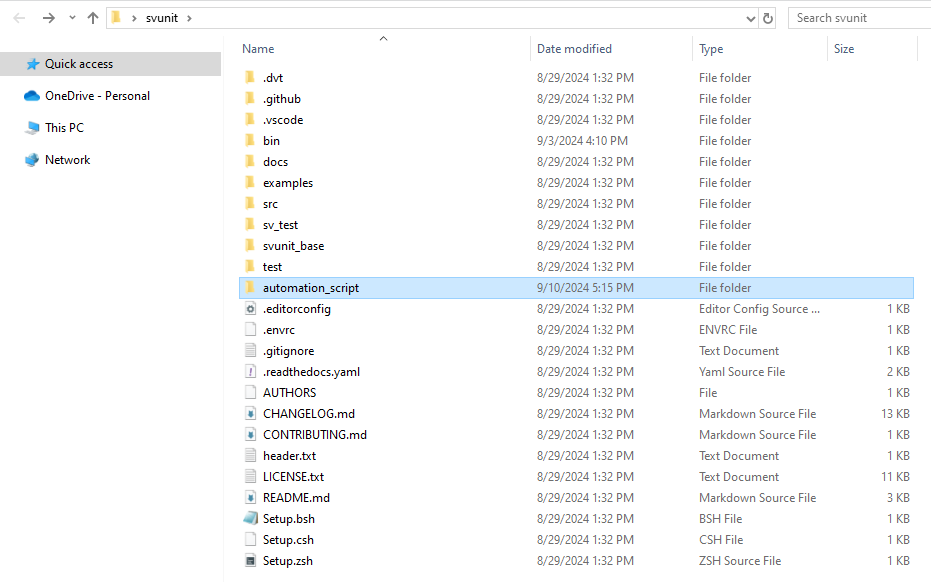
> git clone https://github.com/svunit/svunit.git

Након извршења ове команде, комплетан код и ресурси за SVUnit биће преузети у директоријум под називом svunit

1. Позиционирање и подешавање окружења: Након успешног клонирања, потребно је позиционирати се унутар преузетог директоријума и покренути скрипту за подешавање. Ово се постиже следећом командом:

> source Setup.bsh

1. Интеграција радног директоријума: Следећи корак је премештање постојећег директоријума *automation\_script* унутар SVUnit директоријума. На овај начин, све потребне скрипте и ресурси ће бити доступни унутар SVUnit окружења.



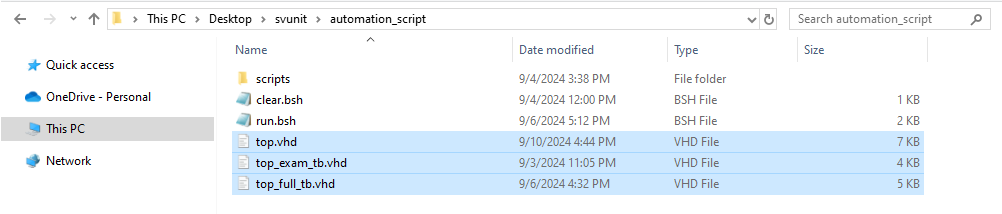
*Слика 5.3 svunit директоријум*

## Припрема основних датотека

Креатор испитног задатка обезбеђује три основне датотеке: *top.vhd*, *top\_tb.vhd* и *top\_full\_tb.vhd*.

* *top.vhd* представља исправно решење задатка, које служи као основа за процену студенских радова.
* *top\_tb.vhd* је тест-бенч који представља пример тачно написаног тест-бенча који се захтева од студента.
* *top\_full\_tb.vhd* обухвата све функционалности датотеке top.vhd, пружајући комплетан тест-бенч који тестира све аспекте решења.

Наведене датотеке треба да буду премештене у радни директорјум *automation\_script*, где ће бити доступне за даље активности.

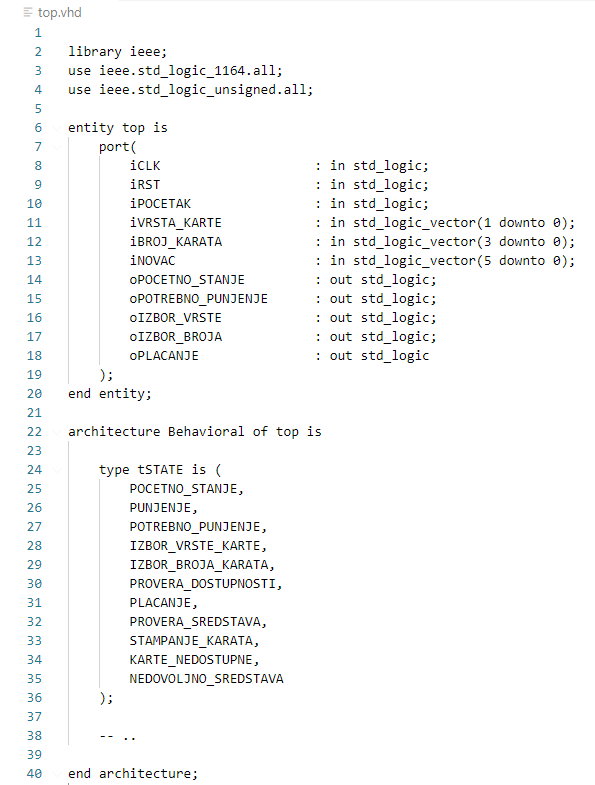


*Слика 5.4 Припрема основних датотека*

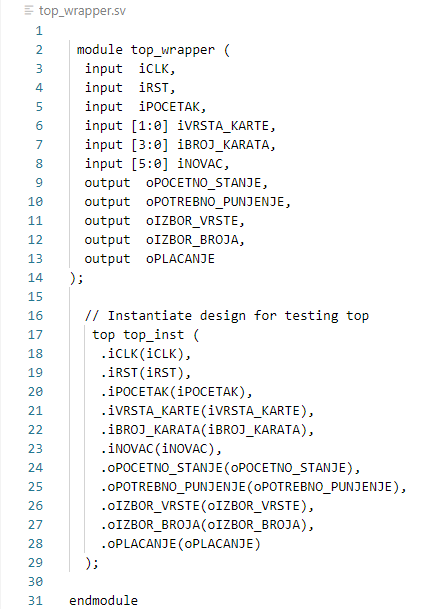
## Креирање wrapper и pack датотека

Следећи корак у процесу укључује позивање Perl скрипти *wrap\_vhdl.pl* и *modify\_vhdl.pl*. Ове скрипте играју кључну улогу у интеграцији VHDL кода са SystemVerilog оквиром:

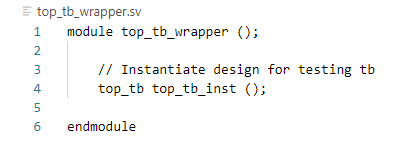
* *wrap\_vhdl.pl*: Ова скрипта генерише датотекe *top\_wrapper.sv* и *top\_tb\_wrapper.sv*, које служе као спрега између VHDL кода и SystemVerilog-а и омогућавају укључивање и комуникацију VHDL компоненти у SystemVerilog окружењу.



*Слика 5.5 Почетна VHDL датотека*

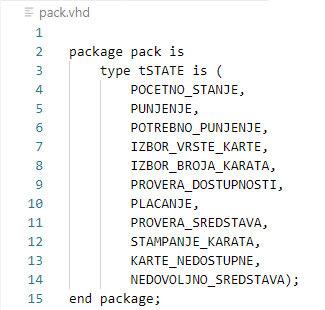


*Слика 5.6 top\_wrapper датотека*



*Слика 5.7 top\_tb\_wrapper датотека*

* *modify\_vhdl.pl*: Скрипта креира датотеку *pack.vhd*, која садржи издвојену дефиницију пребројивих типова из *top.vhd* уколико они постоје. Овај корак је важан како би се обезбедило да се већ дефинисани пребројиви типови из VHDL-а могу користити и у оквиру SystemVerilog хијерархије.



*Слика 5.8 pack датотека*

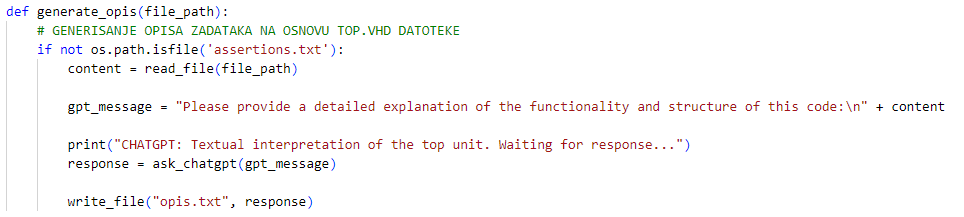
## Генерисање SystemVerilog тврдњи

Процес генерисања SystemVerilog тврдњи представља кључан корак у аутоматизацији прегледања студенских радова. Овај процес укључује неколико основних фаза: текстуалну интерпретацију VHDL кода, генерисање тврдњи на основу те интерпретације и коначну проверу синтаксне исправности генерисаних тврдњи.

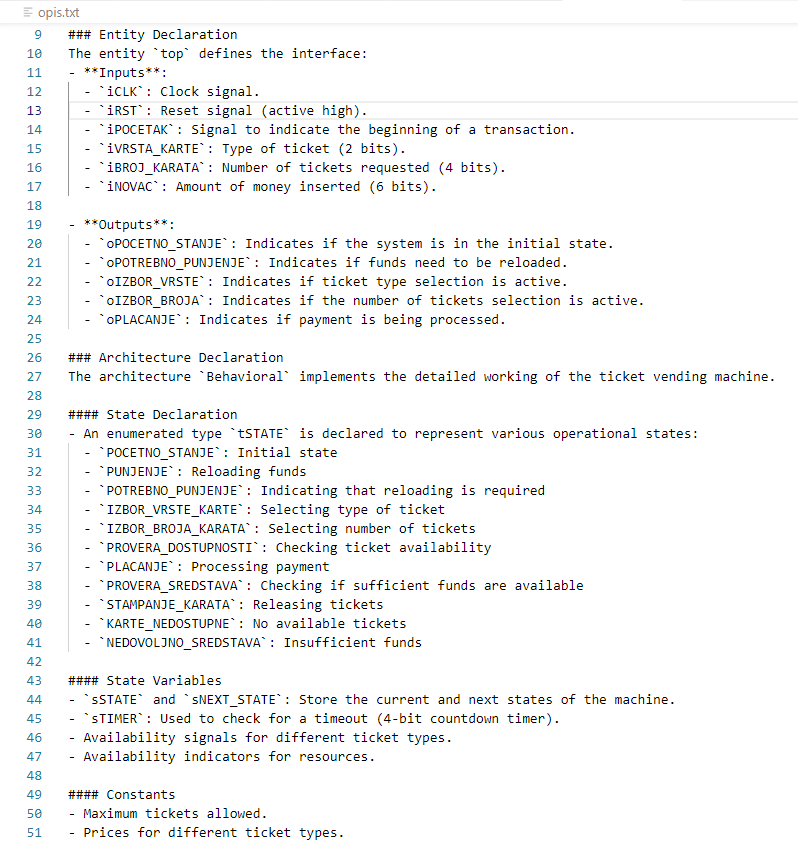
Све операције у вези са генерисањем SystemVerilog тврдњи, укључујући текстуалну интерпретацију VHDL кода, само генерисање тврдњи и проверу њихове синтаксне исправности, извршава Python скрипта *create\_assertions.py*. Ова скрипта се налази на путањи *automation\_script\scripts\* и аутоматски управља свим корацима, од почетне интерпретације кода до финалне верификације тврдњи, обезбеђујући да су све генерисане тврдње синтаксно исправне и спремне за употребу у тестирању.

### Текстуална интерпретација VHDL кода

Први корак у процесу је захтев за текстуалном интерпретацијом датотеке *top.vhd* преко ChatGPT API-ја. Овај корак је важан јер омогућава систему да разуме структуру и функционалност VHDL кода, што је критично за последњи корак генерисања тврдњи. Након вишеструких тестирања, утврђено је да текстуална интерпретација значајно доприноси прецизности и релевантности захтева које ће систем касније обрадити.



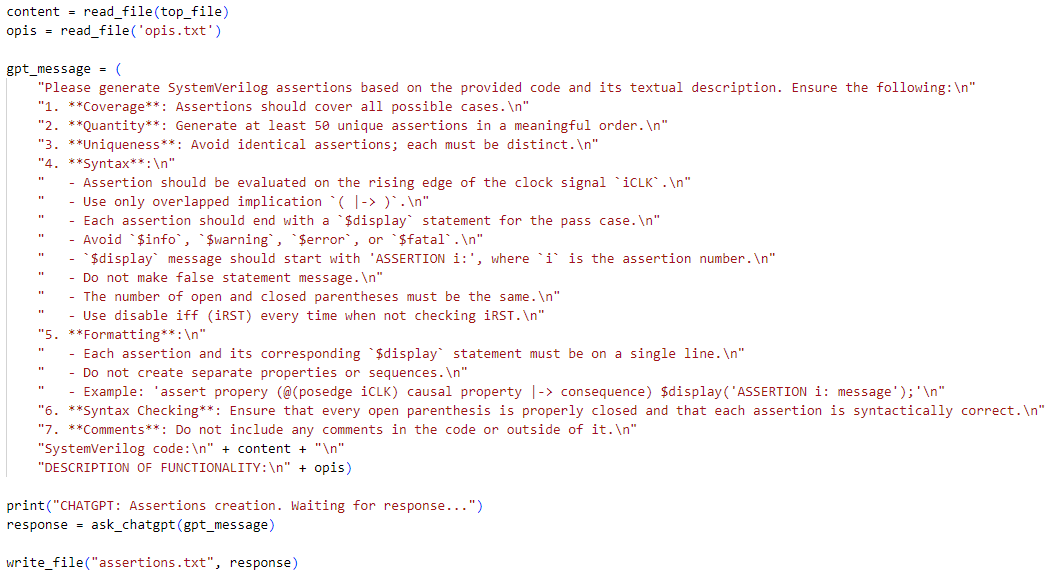
*Слика 5.9 Захтев за текстуалном интерпретацијом*



*Слика 5.10 Текстуална интерпретација*

### Генерисање SystemVerilog тврдњи уз помоћ ChatGPT API-ја

Након што је добијена текстуална интерпретација, следећи корак је захтев за генерисање SystemVerilog тврдњи на основу те интерпретације и оригиналне датотеке *top.vhd*. Ова фаза укључује употребу ChatGPT API за креирање тврдњи које одговарају функционалности описаној у VHDL коду.



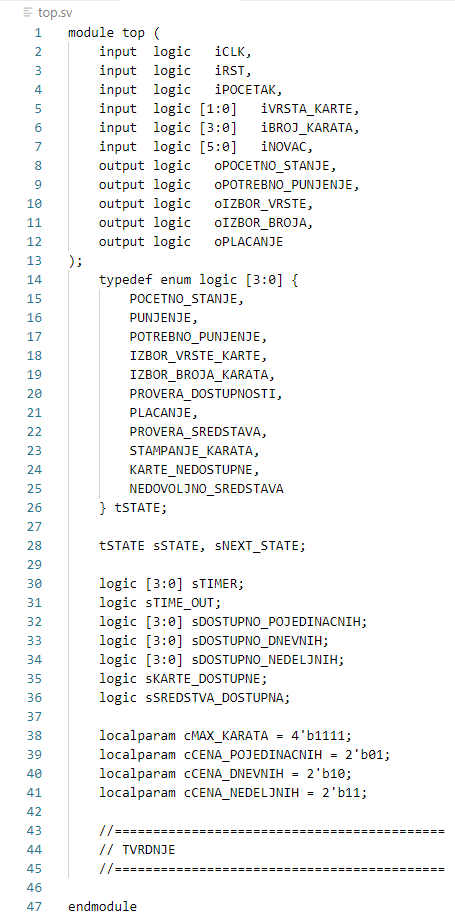
*Слика 5.11 Захтев за генерисање тврдњи*

### Провера синтаксне исправности тврдњи

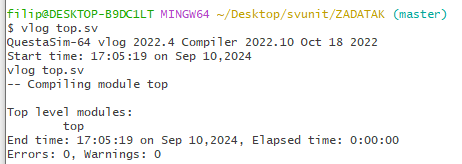
Да би се обезбедила синтаксна исправност генерисаних тврдњи, креирамо SystemVerilog костур који садржи све потребне улазe, излазе, међу-сигнале и константе из VHDL кода. Овај костур представља основну структуру у којој се уграђују генерисане тврдње.

Током овог корака, генерисане тврдње се убацују у SystemVerilog костур и извршава се компајлирање. Ако компајлирање није успешно, поступак генерисања се понавља све док компајлирање не буде успешно. Овај процес осигурава да су све тврдње синтаксно исправне и спремне за употребу у тестирању.

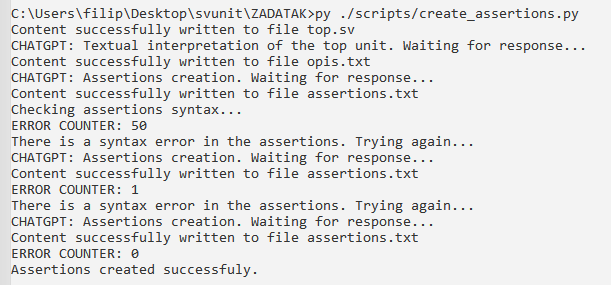
Овако структуиран приступ осигурава да су сви аспекти генерисања тврдњи тачно обрађени и интегрисани, што омогућава ефикасно и поуздано тестирање студенских решења.



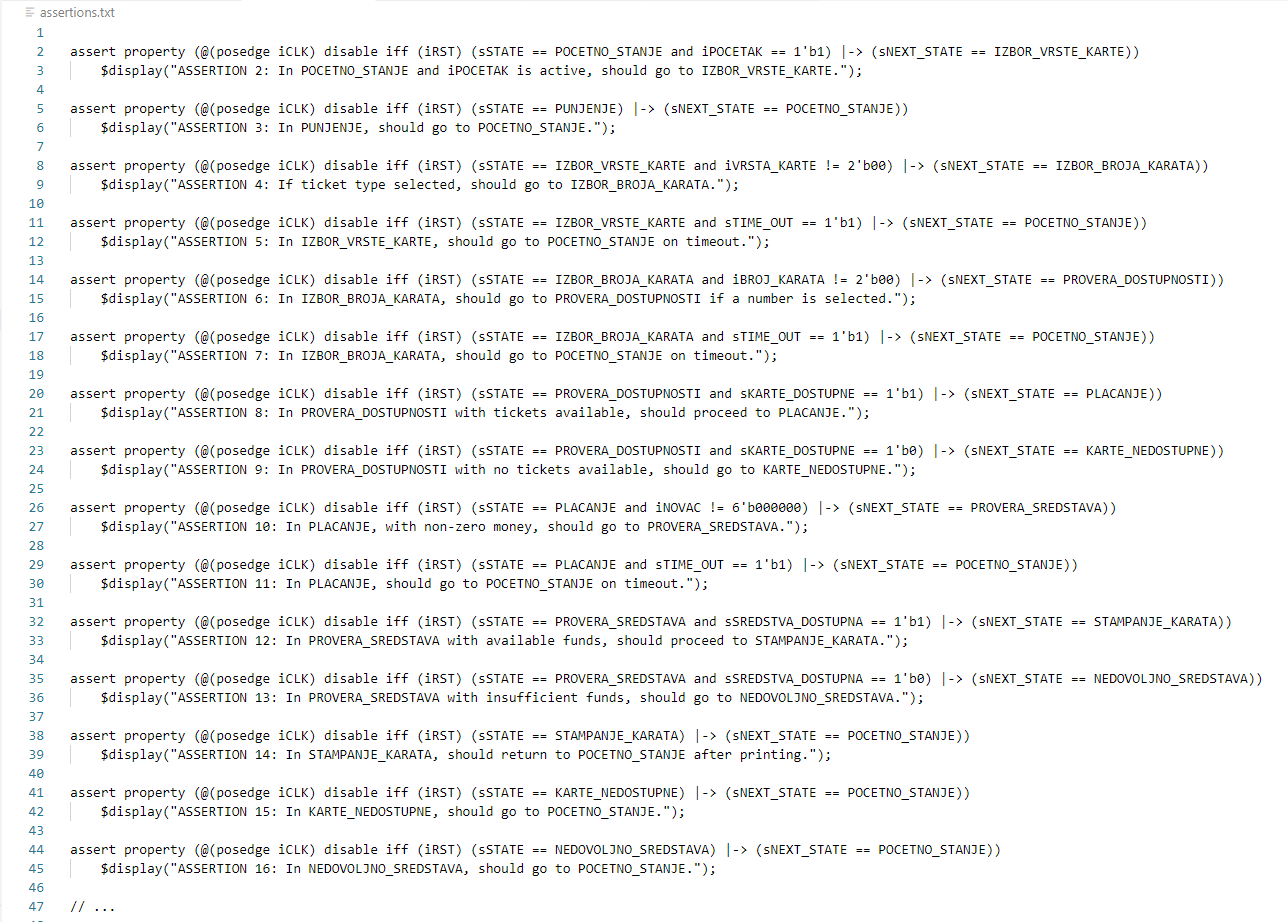
*Слика 5.12 SystemVerilog костур за превођење*



*Слика 5.13 Превођење тврдњи*



*Слика 5.14 Извршавање скрипте за генерисање тврдњи*



*Слика 5.15 Пример генерисаних синтаксно исправних тврдњи*

## Генерисање SVUnit Unit Test датотека

Након што су успешно генерисане SystemVerilog тврдње, следећи корак је креирање SVUnit Unit Test датотека које ће омогућити аутоматизовано тестирање студенских решења.

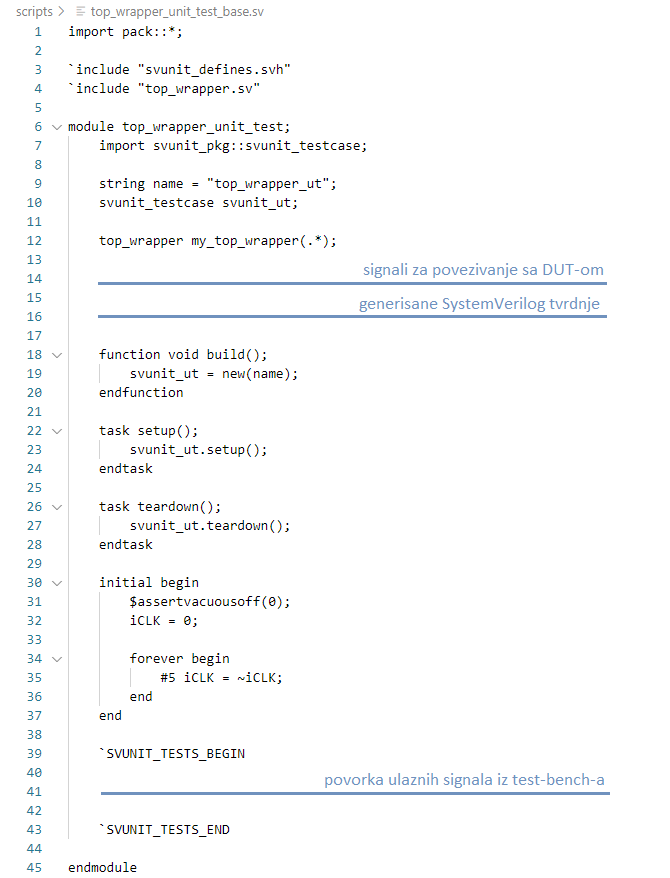
Све операције везане за креирање SVUnit Unit Test датотека, укључујући учитавање костура тестних датотека, уметање сигнала, тврдњи и поворке улазних сигнала, извршава Python скрипта *create\_unit\_test.py* која се налази на путањи automation\_script\scripts\. Након што се датотеке припреме, оне су спремне за даље тестирање у SVUnit окружењу.

### Учитавање SVUnit Unit Test костура датотеке

Први корак је учитавање шаблонских датотека које служе као основа за тестове. Ове датотеке се налазе на путањи *automation\_script/scripts/* и садрже неопходне сигнале и структуре за правилно функционисање тест окружења.

Учитавају се следеће датотеке:

* top\_wrapper\_unit\_test\_base.sv – садржи основни костур теста за DUT.
* top\_tb\_wrapper\_unit\_test\_base.sv – садржи основи костур теста за TestBench.

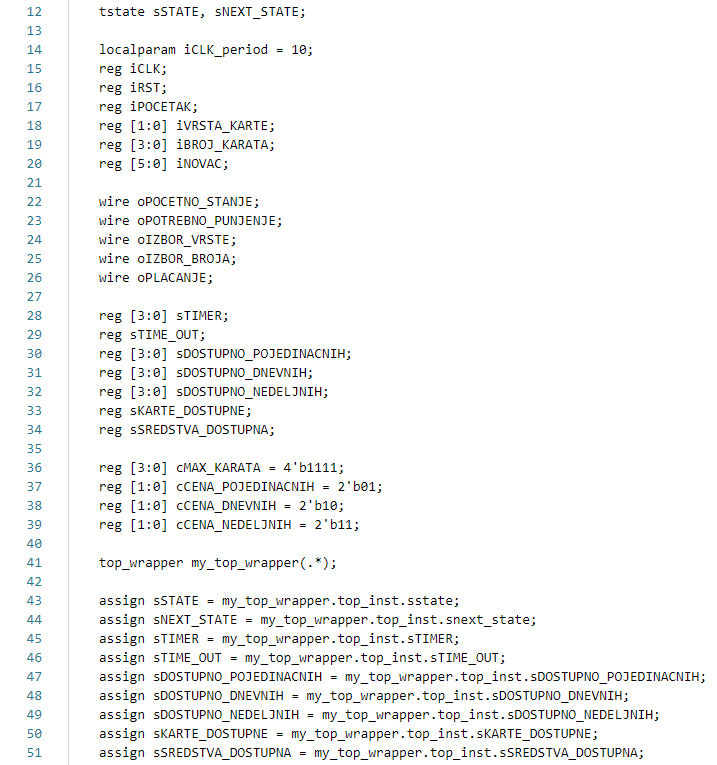


*Слика 5.16 Костур SVUnit тестне датотеке*

### Уметање сигнала

Након учитавања костура, следећи корак је уметање сигнала неопходних за функционалност теста. Ови сигнали повезују SystemVerilog тест јединицу са главним модулом, омогућавајући интеракцију између тестних компоненти и функционалности главног модула.

Уметање сигнала у SVUnit тестну датотеку врши се тако што се отвара главни модул, из кога се узимају референтни параметри и сигнали. Ови параметри се затим трансформишу у одговарајући облик који је потребан за тестирање, и након тога се убацују у тестну структуру. Овај процес је исти за обе датотеке, *top\_wrapper\_unit\_test\_base.sv* и *top\_tb\_wrapper\_unit\_test\_base.sv*, обезбеђујући да сви улазни и излазни сигнали буду коректно повезани са модулом који се тестира, као и да је целокупна комуникација унутар теста исправна.



*Слика 5.17 Сигнали унутар SVUnit тестне датотеке*

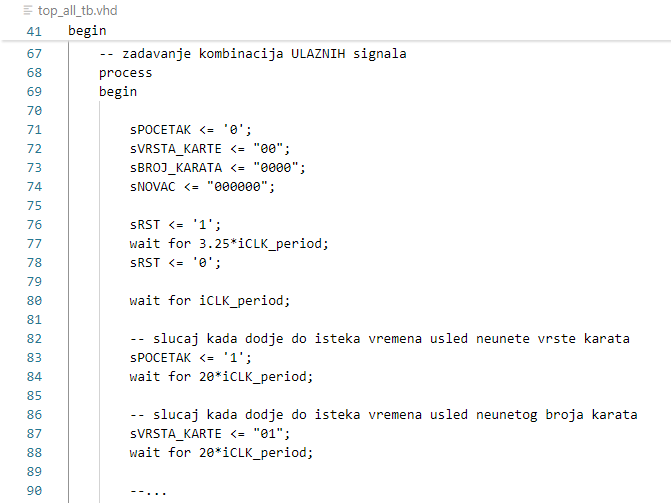
### Уметање генерисаних тврдњи

Када су сигнали постављени, у SVUnit Unit Test датотеку се убацују се раније генерисане SystemVerilog тврдње. Ове тврдње су пресудне за тестирање исправности решења које су студенти предали, јер се на основу њих врши провера тачности функционалности модула и тест бенча. Процес уметања тврдњи је исти за обе датотеке, *top\_wrapper\_unit\_test\_base.sv* и *top\_tb\_wrapper\_unit\_test\_base.sv*.

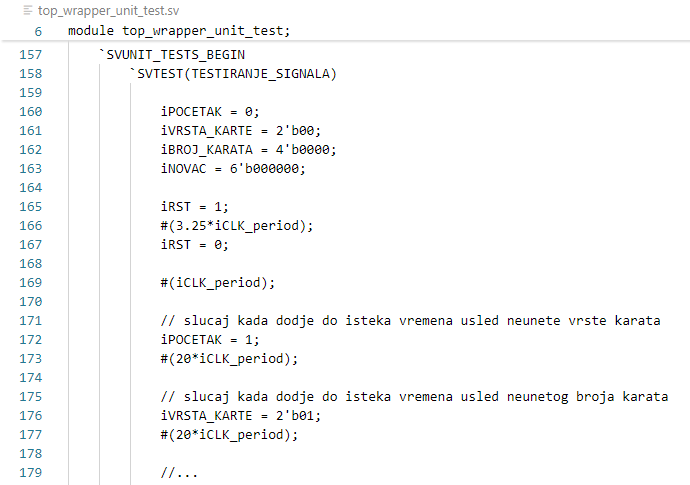
### Уметање поворке улазних сигнала

Последњи корак у процесу је уметање поворке сигнала. Поворка сигнала из датотеке *top\_tb.sv* убацује се у тест структуру *top\_tb\_wrapper\_unit\_test\_base.sv*, док се поворка из датотеке *top\_all\_tb.sv* убацује у одговарајућу тест структуру *top\_wrapper\_unit\_test\_base.sv*.

Како су ови сигнали учитани из VHDL форме, они се прво конвертују у одговарајућу SystemVerilog форму пре уметања у тестне датотеке. Овај приступ осигурава да сваки тест обухвата исправно форматиране улазне услове за оба типа тестних датотека.



*Слика 5.18 Поворка улазних сигнала из top\_all\_tb.vhd датотеке*



*Слика 5.19 Поворка улазних сигнала у SVUnit тестној датотеци*

## Провера логичке исправности тврдњи

### Покретање Questa симулације

Да би се проверила логичка исправност тврдњи у оквиру развијеног пројекта, први корак је покретање Questa симулације над датотеком *top.vhd*, која садржи функционалност модула. Симулација обухвата два претходно генерисана SVUnit теста: *top\_wrapper\_unit\_test.sv* и *top\_tb\_unit\_test.sv* и изводи се позивањем скрипте *runSVUnitNP*. Ова скрипта представља адаптацију за Windows окружење скрипте *runSVUnit*, која је део стандардног SVUnit окружења и служи за покретање тестова.

### Анализа резултата симулације

Након завршетка симулације, генерише се лог датотека *run.log*, која садржи све битне податке о резултатима оба SVUnit теста. За обраду и анализу ове датотеке користи се Python скрипта *invalid\_assertion\_filter.py*, која је смештена у директоријуму *automation\_script/scripts/*. Скрипта филтрира податке из лог датотеке, издваја делове који се односе на сваки тест посебно, те бележи успешне тврдње које могу бити коришћене у будућим тестовима.

### Уклањање логички неисправних тврдњи

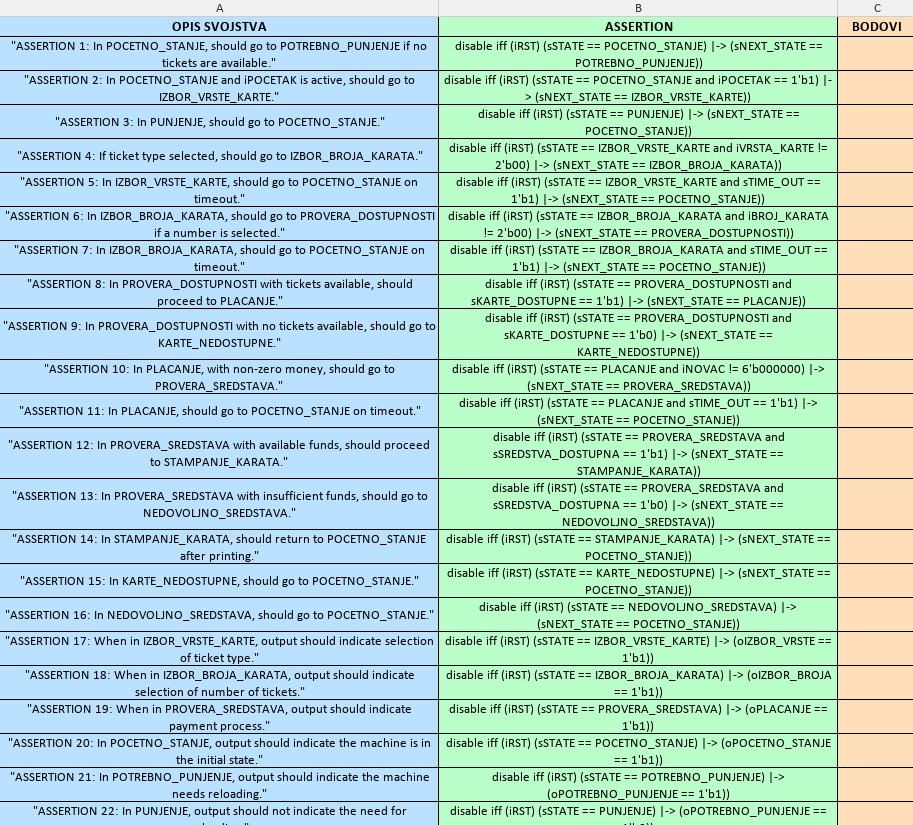
Тврдње које нису прошле симулацију и које су детектоване као логички неисправне се уклањају из SVUnit Unit Test датотека. Ово се врши коришћењем већ поменуте скрипте *create\_unit\_test.sv*, која у овом случају ажурира постојеће тестне датотеке, уклањајући тврдње које су детектоване као неисправне. Овим процесом се осигурава да ажуриране SVUnit Unit Test датотеке: *top\_wrapper\_unit\_test.sv* и *top\_tb\_wrapper\_unit\_test.sv* садрже само тврдње које су логички исправне и могу бити примењене за будуће тестирање и евалуацију студентских радова.

## Креирање табеле за оцењивање

Скрипта *invalid\_assertion\_filter.py*, поред тога што одбацује логички неисправне тврдње на основу анализе симулационог лога, аутоматски генерише и две Excel табеле које садрже логички исправне тврдње. Ове табеле служе за евалуацију тврдњи које су успешно прошле симулацију.

* Табела под називом *tabela\_svojstava\_dut.xlsx* садржи логички исправне тврдње из теста *top\_wrapper\_unit\_test.sv* и односи се на DUT.
* Табела под називом *tabela\_svojstava\_tb.xlsx* садржи логички исправне тврдње из теста *top\_tb\_wrapper\_unit\_test.sv* и односи се на студентски test-bench

Након што су табеле генерисане, свакој тврдњи је потребно ручно доделити одређени број бодова. Оцењивање се врши у складу са значајем сваке тврдње у контексту функционалности коју проверава. Тврдње које покривају критичне аспекте функционалности могу носити више бодова, док тврдње које су мање значајне или проверавају мање важне аспекте могу носити мањи број бодова.



*Слика 5.20 Табела својстава са бодовима*

## Додела бодова тврдњама унутар SVUnit тестова

Додела бодова тврдњама представља последњи корак у процесу аутоматизације оцењивања студентских радова. Овај задатак извршава Python скрипта *load\_scores\_from\_excel.py*, која се налази у директоријуму *automation\_script/scripts/.*

Скрипта користи попуњене Excel табеле које садрже тврдње и њихове одговарајуће бодове (*tabela\_svojstava\_dut.xlsx* и *tabela\_svojstava\_tb.xlsx*) и на основу тих табела, свакој тврдњи унутар SVUnit тестова додељује се број бодова који је дефинисан и унет у табеле.

Овај корак је од суштинске важности јер омогућава интеграцију са скриптом за оцењивање, која користи додељене бодове како би проценила и рангирала исправност решења студената.

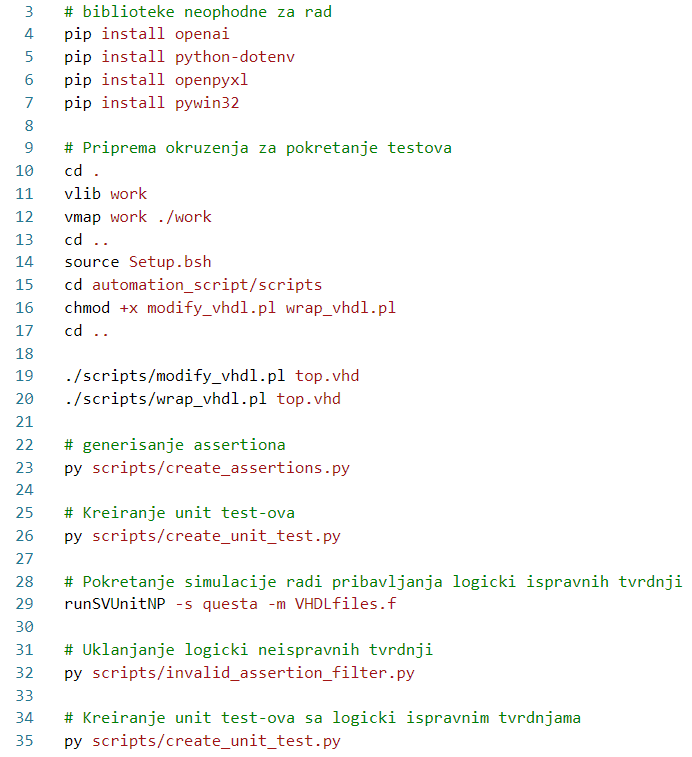
## Извршавање целокупног процеса

Ово поглавље представља демонстрацију корисничке интеракције са целокупним процесом. Све скрипте које аутоматизују генерисање, проверу и евалуацију тврдњи интегрисане су у једну Bash скрипту под називом *run\_first.bsh*. Ова скрипта покреће све потребне кораке редоследом којим се осигурава исправност процеса и генерисање свих потребних датотека.

### Извршавање скрипте run\_first.bsh

Скрипта *run\_first.bsh* покреће се као први корак у овом процесу. Она редом извршава:

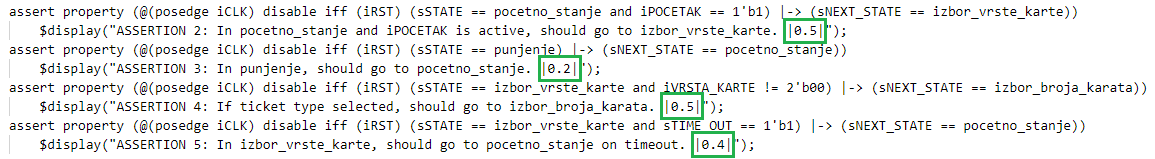
* Преузимање библиотека неопходних за рад
* Припрему окружења
* Генерисање тврдњи
* Креирање Unit Test-ова
* Покретање симулације
* Уклањање логички неисправних тврдњи и креирање табела
* Креирање Unit Test-ова са логички исправним тврдњама



*Слика 5.21 run\_first.bsh скрипта*

### Извршавање скрипте run\_second.bsh

Након евалуације тврдњи унутар табела и њиховог бодовања, последни корак је позивање скрипте *run\_second.bsh* која унутар себе позива Python скрипту *load\_scores\_from\_excel.py*.



*Слика 5.22 Бодови додељени тврдњама*

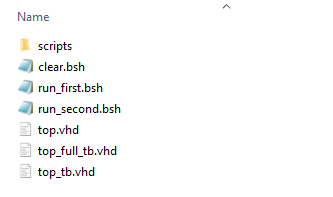
## Преглед директоријума пре и након извршавања процеса

На крају корисно је приказати визуелну разлику између стања директоријума пре и после извршења свих корака. Овај део демонстрира како су датотеке организоване и који су резултати генерисани у току поступка.

### Директоријум пре извршавања

На следећој слици представљен је директоријум пре почетка извршавања свих корака процеса. У овом директоријуму се налазе само изворне датотеке и скрипте које су потребне за иницијалну припрему. Значајне су:

* Изворне датотке као што су *top.vhd*, *top\_tb.vhd*, и *top\_full\_tb.vhd*
* Све скрипте које се користе унутар директоријума *scripts*

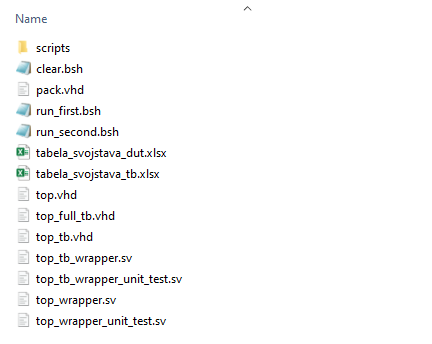


Слика . Садржај директоријума пре извршавања

### Директоријум након извршавања

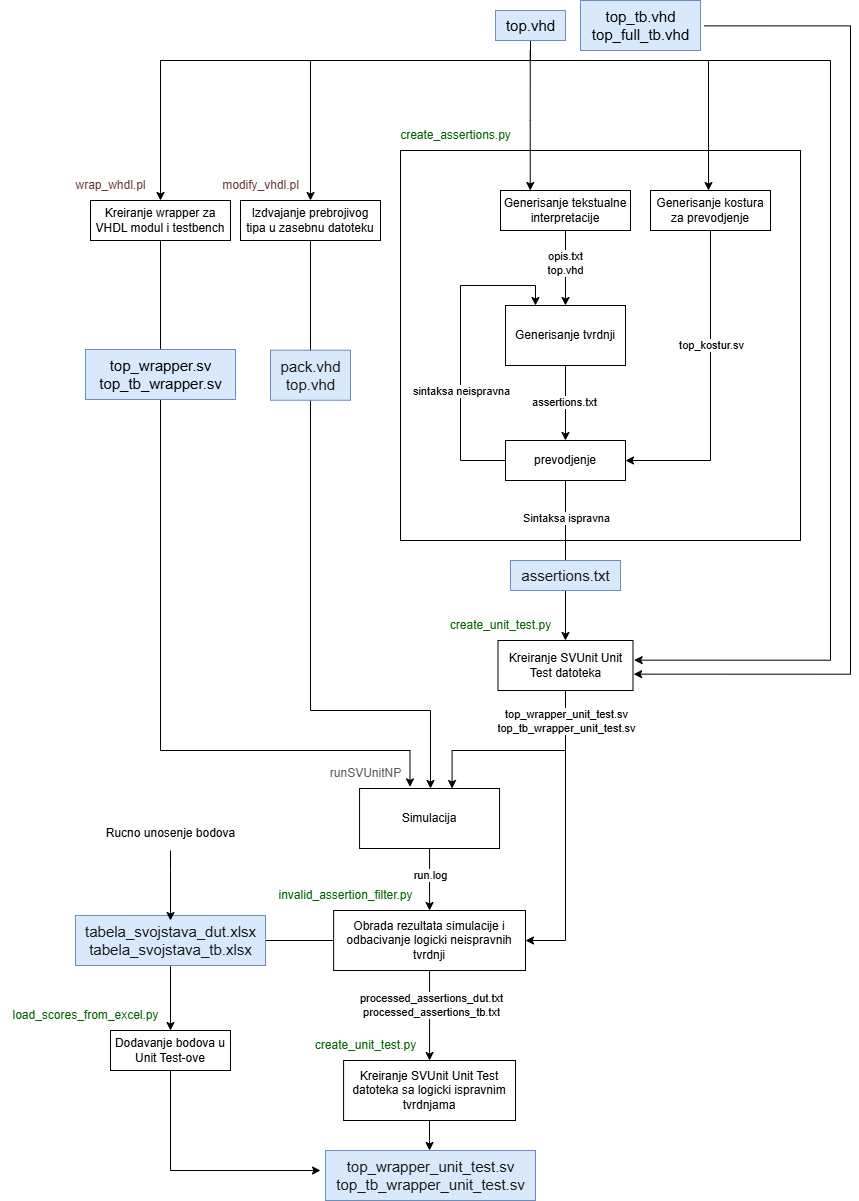
Након што су сви кораци завршени, директоријум садржи генерисане датотеке које су резултат аутоматизације процеса. У овом директоријуму налазе се:

* Генерисане припремне датотеке *pack.vhd*, *top\_wrapper.vhd* и *top\_tb\_wrapper.vhd*
* Текстуалне датотеке који садрже тврдње *assertions\_dut.txt* и *assertions\_tb.txt*
* Unit Test датотеке *top\_wrapper\_unit\_test.sv* и *top\_tb\_wrapper\_unit\_test.sv*
* Табеле са додељеним бодовима за тврдње *tabela\_svojstava\_dut.xlsx* и *tabela\_svojstava\_tb.xlsx*



*Слика 5.24 Садржај директоријума након извршавања*

## Дијаграм функционалности



*Слика 5.25 Дијаграм функционалности*

# Резултати

Најначајнији резултат овог рада јесте успешна употреба ChatGPT комерцијалне LLM платформе за креиранје базичних SV тврдњи са фокусом на проверу функционалности коначних аутомата писаних у VHDL језику. С обзиром на чињеницу да у оквиру провера знања на предмету ЛПРС1 значајан проценат задатака покривају коначни аутомати, овакав вид аутономног креирања испитних тврдњи представља олакшицу која доприноси објективности провера.

Током развоја програмског решења сагледано је да ChatGPT платформа повремено креира синтаксно неисправне тврдње, што је кориговано додатним позивањем симулационог алата.

При анализи синтаксно исправног скупа тврдњи уочено је да постоје тврдње које логички нису исправне, па је стога програмско решење проширено са додатним позивом симулатора у ком се оне проналазе и бришу из коначног скупа.

На крају, битно је истаћи да скуп логички исправних тврдњи захтева валоризацију, односно постављање бодовне тежине за сваку, што излази ван оквира овог рада.

Слике 6.2 и 6.1 илуструју употребу овде изложеног решења, без дубље анализе проблема валоризације тврдњи. Разлике у бодовима последица су пре свега различитих критеријума наслеђених из претходног периода са релативно високом пролазношћу на проверама знања, са једне стране и релативно унифорнмне дистрибуције бодова по тврдњама у овде анализираном решењу.

|  |  |
| --- | --- |
|  |  |
| *Слика 6.1 Резултати коришћењем претходног решења* | *Слика 6.2 Резултати коришћењем тренутног решења* |

# Закључак

У овом дипломском раду предложен је систем за аутоматско генерисање SystemVerilog тврдњи и SVUnit тестова користећи вештачку интелигенцију, са циљем олакшавања и убрзавања процеса верификације студентских решења писаних у VHDL језику. Кроз интеграцију ChatGPT API-ја и аутоматизацију поступка, значајно су смањени напори који су до сада били потребни за ручно писање тврдњи и формирање тестова. Имплементирано решење омогућава брже, прецизније, и објективније оцењивање студентских радова.

Иако предложено решење доноси значајна унапређења, и даље постоје одређена ограничења. Најпре, иако је систем ефикасан у генерисању тврдњи и тестова, квалитет генерисаних тврдњи зависи од сложености и квалитета улазних података, што може утицати на тачност тестирања у неким случајевима. Такође, систем захтева пажљиву припрему улазних датотека и почетну конфигурацију, што може представљати изазов корисницима без искуства у раду са оваквим алатима.

Са даљим развојем вештачке интелигенције, очекује се да ће се значајно повећати квалитет генерисаних тврдњи, што ће омогућити још прецизнију и поузданију верификацију система. Напреднији модели вештачке интелигенције би могли боље разумети сложену структуру кода и самим тим генерисати сложеније тврдње које покривају већи број функционалних сценарија.

Такође, с порастом комплексности испитиваних система, процес генерисања тврдњи може постати све захтевнији и детаљнији. У том контексту, једна од могућих идеја за будући развој јесте увођење механизама за самостално унапређивање система. Ови механизми би омогућили да систем учи из претходних процеса генерисања тврдњи и оптимизује своје алгоритме, чиме би се убрзао целокупан процес.

# Литература

1. Небојша Пјевалица: *Верификација дигиталних интегрисаних кола System Verilog са основама UVM-а*, Нови Сад, ФТН 2023
2. Verification Academy, [Verification Academy: Tools and training for functional verification | Siemens Verification Academy](https://verificationacademy.com/), (приступљено септембар 2024.)
3. Questa Simulator, [Questa Advanced Simulator | Siemens Software](https://eda.sw.siemens.com/en-US/ic/questa/simulation/advanced-simulator/), (приступљено септембар 2024.)
4. SVUnit, [SVUnit | AgileSoC](http://agilesoc.com/open-source-projects/svunit/), (приступљено септембар 2024.)
5. Python, [Welcome to Python.org](https://www.python.org/), (приступљено септембар 2024.)
6. Perl, [The Perl Programming Language - www.perl.org](https://www.perl.org/), (приступљено септембар 2024.)
7. OpenAI, [OpenAI](https://openai.com/), (приступљено септембар 2024.)