

UVOD

Osmisliti laboratorijske vežbe iz određene oblasti računarskih nauka nije uvek jednostavan zadatak. Kao efikasnu pomoć savremena nastava koristi razne softverske sisteme. Na vežbama iz oblasti arhitekture i organizacije računara u velikoj meri se koriste simulatori na skoro svim poznatijim svetskim univerzitetima.

Tokom svog školovanja na Elektrotehničkom fakultetu pohađajući laboratorijske vežbe iz raznih predmeta sreo sam se sa različitim simulatorima. Oni se najviše koriste u edukativne svrhe. Njihova osnovna namena je vizuelno prikazivanje načina rada hardverskih komponenti da bi studenti što lakše shvatili kako funkcionišu. Osim toga, simulatori se mogu koristiti u mnogim drugim sferama ljudskog života kao što su elektrotehnika, arhitektura, mašinstvo, tehnologija, matematika, medicina, ekonomija i druge.

Simulacija je imitacija neke realne stvari ili procesa. Srž simulacije je predstavljanje ključnih karakteristika ili ponašanja odabranog fizičkog ili apstraktnog sistema. Simulacije se koriste u razne svrhe. Na primer za modelovanje prirodnih ili ljudskih sistema, da bi se shvatila srž njihovog funkcionisanja. Ili simulacija tehnoloških pronalazaka, da bi se poboljšale performanse, povećala sigurnost, otklonile greške ili zbog vežbanja ili učenja. Fizička simulacija je simulacija u kojoj fizički objekti menjaju neke realne stvari. Ovi fizički objekti se često biraju zato što su manji ili jeftiniji od stvarnih objekata ili sistema. Interaktivna simulacija je posebna vrsta fizičke simulacije. Ona uključuje i čoveka – operatera koji upravlja simulacijom. Računarska simulacija je pokušaj imitacije stvarne ili hipotetičke situacije na računaru, tako da se može proučavati kako neki sistem radi. Menjanjem vrednosti promenljivih može se menjati situacija u kojoj se sistem nalazi.

Za nas je zanimljiva računarska simulacija računara. Softver koji vrši ovu vrstu simulacije se zove simulator arhitekture računara. Ovaj tip softvera se često koristi da bi izvršio program koji nije kompatibilan sa računarom na kojem se izvršava. Može se koristiti i za potrebe testiranja, npr. mikroprograma procesora, kao i u edukativne svrhe. Simulatori arhitekture računara se mogu svrstati u razne kategorije u zavisnosti od vrste podele. U zavisnosti od predmeta simulacije, podela je na simulatore mikro – arhitekture i simulatore kompletnog sistema. Prvi simulira samo rad mikroprocesora a drugi ceo računarski sistem. Po detaljnosti simulacije simulatori se dele na funkcionalne i simulatore performansi. Prvi tip prikazuje način na koji sistem radi dok se drugi bavi njegovim mogućnostima. Po trećoj podeli postoje simulatori koji rade uvek na isti način i oni čiji rad zavisi od ulaznih podataka.

POSTOJEĆA REŠENJA

Laboratorijske vežbe obuhvataju različite koncepte i predznanje na različitim nivoima studija. Početne treba da demonstriraju karakteristike i ponašanje osnovnih komponenti računarskog sistema. U okviru kasnijih kurseva potrebno je raditi sa kompleksnijim tehnikama i komponentama. Njihov sastavni deo su i neke od osnovnih komponenti sa ranijih kurseva. Zato bi bilo pogodno da se one mogu što lakše iskoristiti u drugim simulatorima.

Simulatori arhitekture računara se uglavnom izrađuju ručno, odnosno projektovanjem na papiru i direktnim pisanjem koda u ciljnom programskom jeziku. Kada se pogleda kôd različitih simulatora može se uočiti da oni imaju dosta sličnosti. Međutim, ako bi programer hteo da iskoristi parče kôda iz simulatora koji se bavi sličnim problemom imao bi težak posao da ga prilagodi svom.

Neki univerziteti u svetu za ovu svrhu koriste poznate simulatore kao što su SimpleCPU, ESCAPE, DLXview, M5, JHDL i HASE. Međutim, svi oni predstavljaju fiksne računarske sisteme i konstantan skup problema. Osim toga, studenti na različitim kursevima koriste različite simulatore pa pre upotrebe moraju da se upoznaju sa njihovim osnovnim funkcijama.

Sve ovo navodi na činjenicu da je potreban nov, kompletan softverski sistem koji bi posedovao sledeće funkcionalnosti: mogućnost projektovanja i unošenja simulacije, lako ponovno korišćenje isprojektovanih komponenti i okruženje za izvršavanje simulacije, što bi rezultovalo uvek istim komandama za rukovanje sistemom koji se simulira.

PROBLEM IZ UGLA KORISNIKA

U ovom poglavlju će biti objašnjeno kakav bi pogled na sistem koji projektuje trebao da ima korisnik. Njemu je potreban simulator nekog logičkog kola. U zavisnosti od potrebe kolo može biti velike složenosti, te se ne može isprojektovati odjednom. Zbog toga je korisniku jako potrebna mogućnost da svoj sistem podeli na delove – module. Napredniji korisnik bi želeo da svoje module iskoristi i u drugim projektima.

Pošto korisnik podeli svoj sistem na delove i napravi jedan deo želeo bi da proveri da li se taj deo ponaša prema očekivanjima. Kao najbolje nameće se rešenje da se simulira ponašanje samo tog dela a korisnik da aktivno učestvuje u simulaciji postavljanjem odgovarajućih vrednosti na ulaze. Naravno, i mogućnost aktivnog učešća korisnika u simulaciji krajnjeg sistema je od izuzetne važnosti, jer i sistemi iz realnog sveta zahtevaju korisnikovu aktivnost.

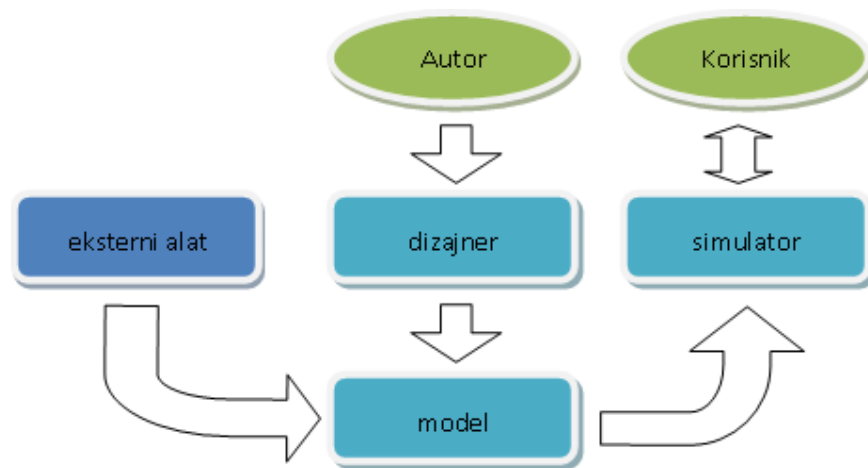
Grafičko korisničko okruženje bi ubrzalo proces pravljenja simulatora. Komponente bi mogle da se postavljaju na radnu površinu pomoću miša. Njihovi izlazi i ulazi bi se jednostavno povezivali. Komponente i veze bi mogle da se pomeraju ili brišu. Sve to bi olakšalo projektovanje, a i uputstvo za upotrebu se delom može integrisati u GUI.

Bilo bi ugodno da neka osnovna logička kola (kao što su I, ILI, NI i NILI) već postoje, jer su ona uvek potrebna. Neki korisnici bi poželeli da imaju i neka složenija kola „pri ruci“, ali pošto je nemoguće zadovoljiti potrebe svih, idealna bi bila mogućnost da korisnik na jednostavan način može napraviti složeno logičko kolo. To se odnosi na kola kod kojih nije bitna unutrašnja struktura za potrebe simulacije već samo treba da „odrade“ svoj posao. To je omogućeno u SAGS-u, a način na koji se unosi ponašanje komponente je programerski – odnose između izlaznih i ulaznih signala unose se u programskom jeziku Java.

To nameće novi problem – sintaksu. Korisnik mora biti siguran da je njegov Java kôd sintaksno ispravan. U slučaju greške korisnik mora dobiti obaveštenje na osnovu kojeg može jasno zaključiti gde je pogrešio.

ORGANIZACIJA SISTEMA

U ovom poglavlju će biti objašnjeno kako je organizovan sistem SAGS. On se sastoji iz dva programa: *Dizajner* i *Simulator*. Oba programa imaju grafičko korisničko okruženje. *Dizajner* služi za pravljenje dok *Simulator* služi za izvršavanje simulacije. Inače, *Dizajner* je tema drugog rada i ovde će ukratko biti objašnjene njegove osnovne mogućnosti. Program *Simulator* će ovde biti detaljno objašnjen.



Slika 1 Organizacija sistema

Oba programa se oslanjaju na *model* – mozak simulacije. U modelu je implementirana perzistencija simulacije na disku. On poseduje i kompletan interfejs za dizajniranje i izvršavanje simulacije. To daje mogućnost relativno lakog konvertovanja i importovanja šema iz popularnih alata kao što su VHDL i Visio.

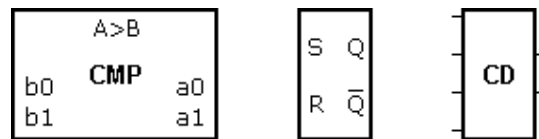
Pod pravljenjem simulacije spada definisanje komponenti od kojih je sistem koji se simulira sastavljen i njihovo povezivanje. Komponente mogu biti proste i složene. Proste komponente se nazivaju elementi a složene moduli. Pri tome se ne misli na složenost komponente u realnom svetu. I veoma složene komponente se mogu jednostavno napraviti pomoću SAGS-a.

- Element

Element je ona komponenta kojoj je definisano samo ponašanje a unutrašnja struktura nije. Može biti kombinacionog i sekvencijalnog tipa. Tu spadaju osnovni elementi kao što su I, ILI, NI, NILI i NE kola, razni flip-flopovi, zatim nešto složenije komponente kao što su koderi, dekoderi, multiplekseri, komparatori, registri i brojači. To mogu biti i komponente višeg nivoa kao na primer memorije i ALU jedinice za koje u datom problemu nije potrebno prikazati unutrašnju strukturu ali moraju biti prisutne u sistemu i obavljati svoju funkciju.



Slika 2 Jednostavna logička kola (I, ILI, NI, NILI)

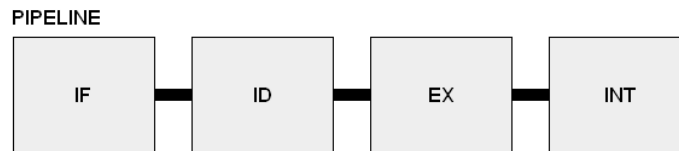


Slika 3 Logička kola (komparator, RS flop-flop, koder)

Svatom elementu treba definisati spoljašnji izgled i ponašanje. Spoljašnji izgled predstavlja slika, pozicija komponente i njen interfejs prema ostatku sistema odnosno ulazni i izlazni signali uključujući i njihove pozicije. Ponašanje elementa se definiše u programskom jeziku Java tako što se definiše zavisnost izlaznih od ulaznih signala. Izlaznim signalima se može dodeliti a iz ulaznih signala se može čitati vrednost. Ostale mogućnosti programskog jezika koje se mogu koristiti su aritmetičke operacije, pomoćne promenljive i funkcije.

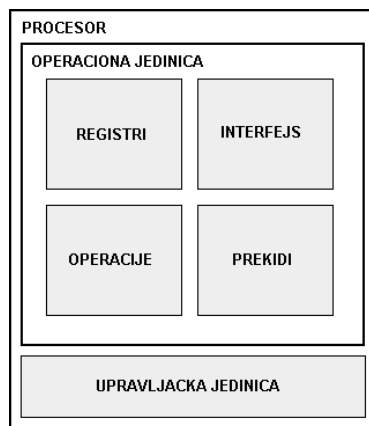
- Modul

Moduli predstavljaju logička kola za koja projektant želi da prikaže unutrašnju strukturu. Zapravo, preporuka za veće projekte je da se ceo digitalni uređaj podeli na module tako da svaki ima jasno definisanu funkciju. Jedan od često primenjivanih postupaka je da se izvršavanje operacije podeli na logičke celine ili faze i da za izvršavanje svake faze u modulu postoji posebna jedinica ili podmodul. Na primer, izvršavanje instrukcije procesora se deli na faze očitavanje instrukcije, dekodovanje instrukcije i čitanje podataka, izvršavanje instrukcije i fazu opsluživanje prekida (slika 4).



Slika 4 Pipeline organizacija procesora (IF - dohvaćanje instrukcije ili instruction fetch, ID - dekodovanje instrukcije ili instruction decode, EX - izvršavanje instrukcije ili execute, INT - opsluživanje prekida ili interrupt)

Za komplikovanje uređaja potrebno je da neki delovi rade u paraleli. Primer za to je opet procesor u računaru koji se sastoji od upravljačke i operacione jedinice, koja se opet sastoji od registara, interfejsa prema ostatku sistema, dela za izvršavanje operacija i dela za opsluživanje prekida (slika 5).

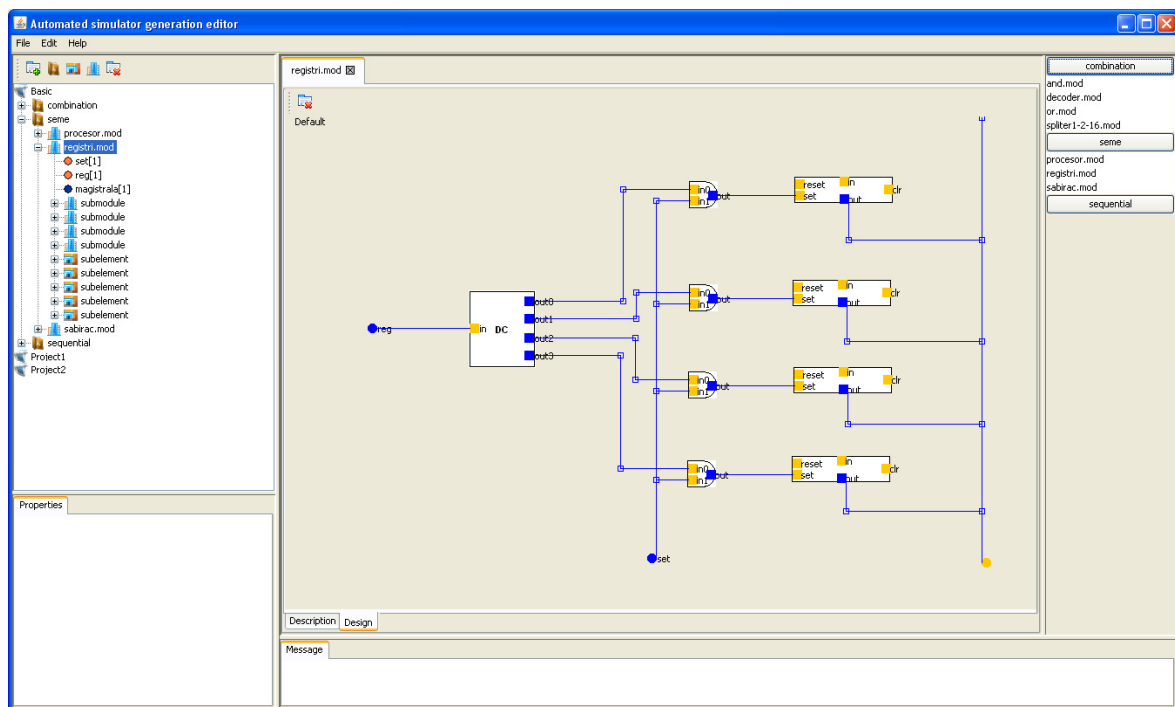


Slika 5 Procesor

Modulu je takođe potrebno definisati spoljašnji izgled i ponašanje. Spoljašnji izgled se definiše na isto kao kod elementa. Međutim ponašanje se definiše na potpuno drugi način – kroz unutrašnju strukturu. Prvo treba izabrati sve komponente koje postoje u modulu. To mogu biti kombinacioni i sekvencijalni elementi ali i drugi moduli. Onda treba definisati kako su komponente međusobno povezane tako što im se povežu ulazni i izlazni pinovi. Linija koja predstavlja vezu se može lomiti i pomerati proizvoljno.

- Program Dizajner

Kao što je već napomenuto, za pravljenje simulacije se koristi program *Dizajner*. Pomoću njega se mogu praviti novi i menjati postojeći elementi i moduli. Sa komponentom se radi na radnoj površini. Obe vrste komponente imaju dva izgleda koje je potrebno definisati – unutrašnji i spoljašnji. Stoga radna površina ima ova dva môda. Môd spoljašnjeg izgleda je isti kod oba tipa komponente. Tu je, kao i što sama reč kaže, potrebno definisati kako komponenta izgleda „spolja“. Pod spoljašnjim izgledom spadaju slika, ulazni i izlazni signali, kao i njihove pozicije i širine. U môdu unutrašnjeg izgleda treba definisati ponašanje komponente. Radna površina se u ovom môdu razlikuje za elemente i module.



Slika 6 Program Dizajner

Radna površina u slučaju môda unutrašnjeg izgleda elementa ima četiri dela: *imports*, *variables*, *functions* i *simulate*. Najvažniji deo je *simulate*. To u stvari predstavlja telo funkcije koja će se izvršiti kada dođe do promene nekog signala na ulazu. Kao predefinisane promenljive mogu se koristiti imena pinova, s tim da se ulaznim pinovima može samo čitati a izlaznim dodeliti vrednost. Za obradu vrednosti mogu se koristiti svi operatori iz programskog jezika Java. U slučaju potrebe za dodatnim funkcionalnostima omogućeno je da se uvezu dodatne klase (koje se navode u delu *import*), definišu pomoćne promenljive (u delu *variables*) i funkcije (u delu *functions*). SAGS poseduje mogućnost provere sintaksne ispravnosti unetog kôda, tako da nije moguće sačuvati sintaksno neispravan element.

Radna površina u slučaju môda unutrašnjeg izgleda modula je grafički orijentisana. Glavni alat u ovom môdu je miš. Potrebno je samo naređati željene komponente na radnu površinu, rasporediti ih po svojoj želji i povezati ih. Neke osnovne komponente su predefinisane (kao što su ulazni i izlazni signali, I, NI, ILI i NILI kola) a mogu se koristiti i komponente koje je korisnik sam napravio. Ovde treba imati u vidu jednu izuzetno korisnu činjenicu – iste komponente se mogu koristiti bez ograničenja u različitim simulacijama.

Finalna simulacija koja će kasnije biti učitana i izvršena u *Simulatoru* je takođe modul.

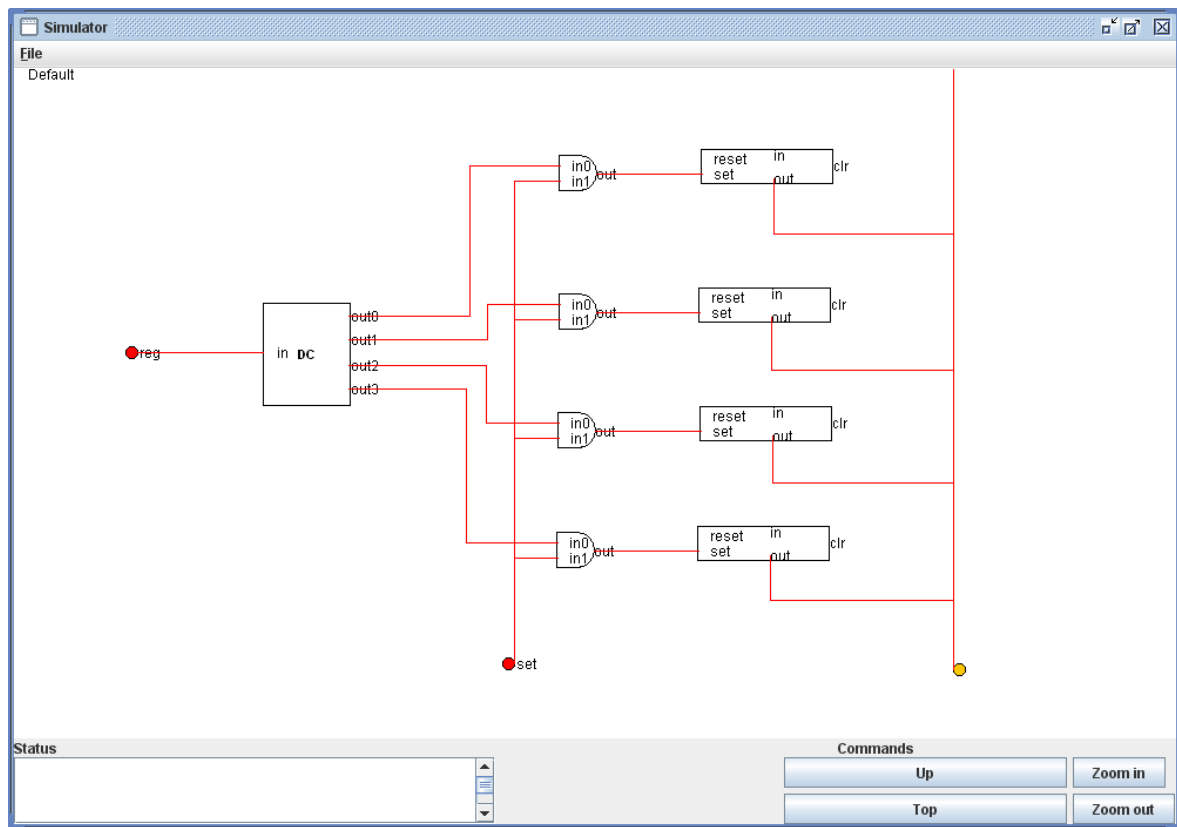
- Program *Simulator*

U ovom poglavlju će biti objašnjeno kako izgleda i kako se koristi program za simulaciju. On se sastoji iz tri dela: meni, panel za simulaciju i panel sa komandama. Program izgleda kao na slici 7.

Prvo je potrebno učitati simulaciju odnosno izabrati modul koji se simulira. Pošto je simulacija učitana na panelu za simulaciju će biti prikazana, kao najviši nivo apstrakcije, unutrašnja struktura modula koji je izabran. Moguće je kretati se kroz strukturu sistema koji se simulira. Za prikaz unutrašnje strukture neke komponente potrebno je kliknuti na nju. Prikaz višeg nivoa apstrakcije od trenutnog omogućuje komanda „Up“. Prikaz najvišeg nivoa apstrakcije postiže se izborom komande „Top“. Unutrašnju strukturu komponente moguće je videti samo ako je ona napravljena u *Dizajneru* kao modul. Tako da je moguće ići duboko u detalje do onog nivoa dok na ekranu ne budu sve prikazane komponente elementi.

Korisnik može menjati stanje sistema koji se simulira menjanjem ulaznih signala i signala takta koji su ponuđeni. Ulazni signali se prikazuju na panelu za simulaciju i to onako kako je dizajner simulacije definisao u programu *Dizajner*. Signali takta (najčešće jedan) su prikazani na panelu sa komandama kao dugme. Broj taktova koji će se desiti po pritiskanju dugmeta ne mora biti jedan. To se može podesiti u susednom polju.

Da bi korisnik imao veću udobnost prilikom korišćenja programa *Simulator* reaguje na neke događaje miša. Najvažniji je naravno klik. Ukoliko korisnik klikne na ulazni signal njegova vrednost će se promeniti. Ukoliko klikne na modul sadržaj panela za simulaciju će biti promenjen i biće prikazana unutrašnja struktura izabranog modula. Osim toga implementirane su i neke dodatne mogućnosti. Okretanjem točkića miša možete vršiti zumiranje. Okrećite točkić unapred (od sebe) za uvećavanje, a unazad (prema sebi) za umanjivanje šeme simulacije. I na panelu sa komandama postoje komande za zumiranje. Osim približavanja i udaljavanja nivo zumiranja se može vratiti i u početni položaj. Šemu simulacije možete pomerati prevlačenjem miša (eng. dragging). Šema se može centrirati pritiskanjem dugmeta „Center“ koje se nalazi na panelu sa komandama.



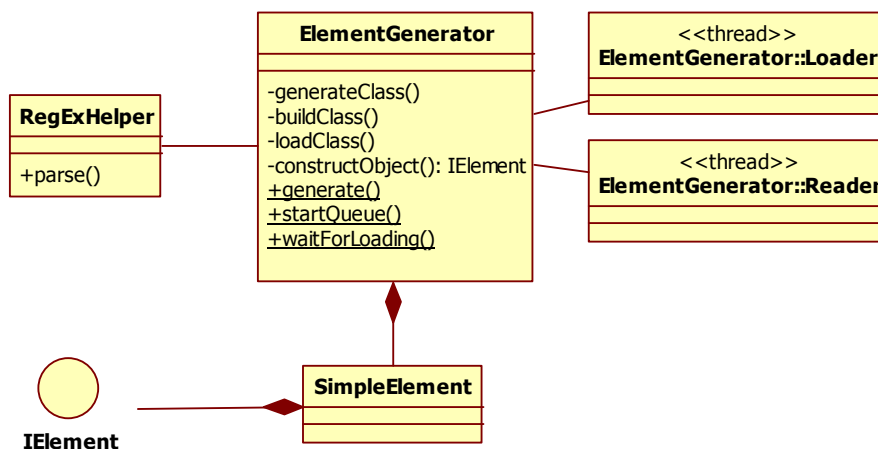
Slika 7 Program Simulator

IMPLEMENTACIJA

SAGS je implementiran u programskom jeziku Java. Sastoji se od tri projekta: *model*, *Dizajner* i *Simulator*. U prvom je implementiran model podataka, tj. učitavanje, čuvanje, pamćenje trenutnog stanja i upravljanje simulacijom. U drugom projektu je implementiran program *Dizajner* a u trećem program *Simulator*.

- Učitavanje

Učitavanje se vrši iz fajla u kojem je definisana kompletna simulacija. Format fajla je ZIP arhiva koja sadrži XML fajl i resurse. U XML fajlu su sve potrebne definicije, a resursi su slike koje predstavljaju spoljašnji izgled komponente. Za svaku vrstu objekta postoji klasa u *modelu* u čijoj metodi *load* je implementirano učitavanje. Vrste objekata su ulazni i izlazni pinovi, elementi, moduli i veze između njih. Za sve objekte osim elementata potrebno je samo učitati podatke iz ulaznog fajla. U slučaju elementa potreban je još jedan korak – prevesti Java kôd. Prevođenje se poziva iz posebnih niti zbog povećanja performansi, tako da se fajlovi prevode u paraleli. Prilikom prevođenja kreiraju se nove Java klase u privremenom direktorijumu čiji se sadržaj briše prilikom pokretanja programa.



Slika 8 UML dijagram klasa koje učestvuju u učitavanju elementa

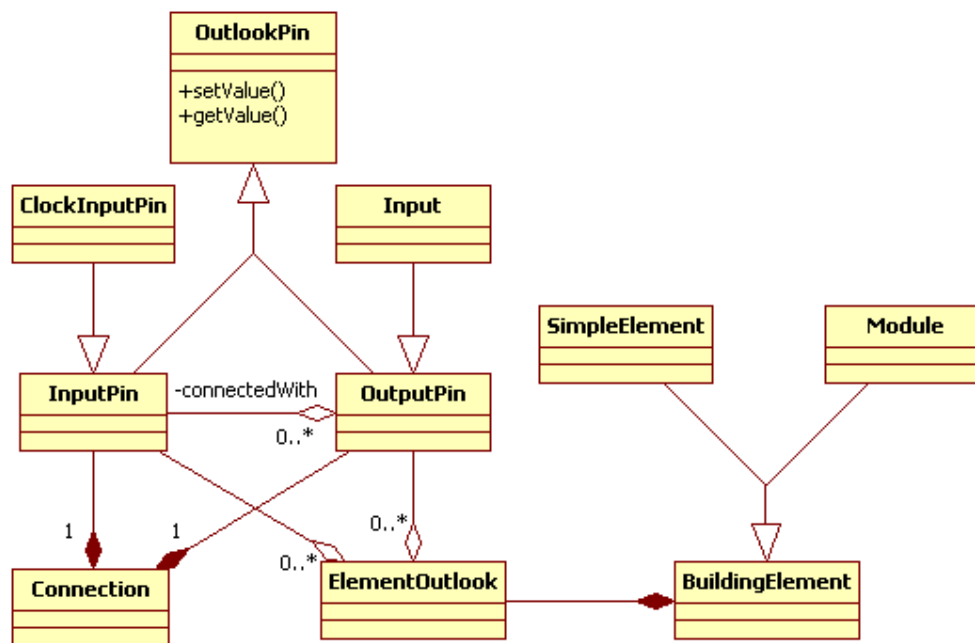
- Učitavanje elemenata

Za potpuno učitavanje elementa potrebno je prevesti Java kôd koji je autor simulacije uneo. Za to se mogu koristiti klase sa dijagrama na slici 8. A prevođenje je podeljeno u četiri faze: generisanje, prevođenje, učitavanje Java klase i kreiranje objekta. Klasa koja se generiše izvodi postojeći interfejs `IElement`, koji poseduje metodu `simulate`. Ova metoda će biti pozvana kada na ulazu elementa dođe do promene vrednosti nekog pina. Za pokretanje postupka prevođenja potrebno je prvo pozvati funkciju `generate` klase `ElementGenerator`. Ona će element koji joj je dat kao parametar staviti u red čekanja. Pošto su svi elementi na ovaj način smešteni u red potrebno je pozvati funkciju `startQueue` iz iste klase. Ona će pokrenuti niti koje će izvršiti prevođenje za elemente koji se prvi put javljaju, odnosno učitavanje za elemente koji su već prevedeni. Nit iz koje je ovaj postupak pokrenut može se blokirati dok se svi elementi ne učitaju pozivom funkcije `waitForLoading`. Ukoliko neki element ima grešku u sintaksi metoda `hasError` će vratiti rezultat `true`, a metoda `getError` tekst poruke o grešci, obe iz klase `SimpleElement`.

- Stanje sistema

Stanje nekog elementa definisano je vrednostima njegovih izlaznih signala. Stanje celog sistema definisano je šemom sistema i stanjima svakog pojedinačnog elementa. Stoga se u modelu pamte samo vrednosti na pinovima. Vrednosti ulaznih pinova se pamte da bi se detektovala promena vrednosti. Osim toga moguće je da elementi pamte svoja stanja pošto je autor simulacije imao slobodu da ih tako implementira.

Promena stanja se dešava kada se promeni neki od ulaznih signala ili signala takta. Ulazni signal je u modelu predstavljen klasom `Input` koja je izvedena iz klase `OutputPin`, a ona predstavlja izlazni pin. Signal takta je predstavljen klasom `ClockInputPin` koja izvodi `InputPin`, koja predstavlja ulazni pin. Pinovi su povezani vezama, koje su predstavljene klasom `Connection`, a elementi i moduli poseduju pinove. Detekcija promene stanja je implementirana korišćenjem projektnog uzorka Posmatrač. Ulazni pinovi posmatraju vrednosti izlaznih pinova sa kojima su povezani. Prilikom kreiranja objekta `Connection`, koji predstavlja vezu, izlaznom pinu se zada ulazni pin koji treba da obavesti o svojoj promeni. Po promeni vrednosti na ulaznom pinu, ukoliko pin pripada elementu, osvežiće se stanje ovog elementa. To znači da će se pozvati `simulate` metoda datog elementa. To opet može izazvati promenu vrednosti na izlaznim pinovima elementa, tj. dalju propagaciju signala kroz sistem.



Slika 9 UML dijagram povezanosti pinova

- Prikazivanje

Simulacija se prikazuje u panelu za simulaciju koji zauzima najveći deo ekrana. Prikazivanje je centralizovano u klasi `DrawingManager`. U njoj za svaki objekat simulacije postoji funkcija u kojoj je definisano njegovo iscrtavanje. U projektu `model` postoji ova klasa koja se koristi kao bazna za klase sa istom funkcijom u ostalim projektima. Takav način organizacije daje mogućnost skinova. Za sada i *Dizajner* i *Simulator* imaju po jedan skin.

Do poziva odgovarajuće funkcije iz klase `DrawingManager` stiže se na sledeći način. Svaki objekat u simulaciji ima metodu `paintElement` koja poziva odgovarajuću metodu iz klase `DrawingManager`. U baznoj klasi `DrawingManager` postoji statičko polje koje definiše skin, odnosno to je instanca određenog `DrawingManager`-a. U metodi `paint` panela na kom je potrebno iscrtati komponentu prvo se podesi skin. U ovom slučaju to je skin predstavljen klasom `DrawingManager` iz projekta `Simulator`. Potom se pozove funkcija `paintElement` modula koji se trenutno prikazuje.

- Obrada događaja

Osluškivanje svih događaja je implementirano u klasi `MouseListener`. Događaji na koje se reaguje su `mouseClicked`, `mouseWheelMoved`, `mousePressed`, `mouseReleased` i `mouseDragged`. Prvi događaj se koristi za promenu vrednosti ulaznih pinova ukoliko je korisnik kliknuo na ulazni pin, odnosno za prikaz unutrašnje strukture modula ukoliko je korisnik kliknuo na modul. Drugi događaj se koristi za zumiranje. Poslednja tri događaja se koriste za pomeranje šeme. Događaj `mousePressed` se koristi da se odredi tačka u kojoj je počelo pomeranje, `mouseDragged` da se šema pomera zajedno sa mišem, a `mouseReleased` da se označi kraj pomeranja.

MESTA ZA NADOGRADNJU

Postoji nekoliko mesta u trenutnoj realizaciji koji bi u značajnoj meri povećali upotrebljivost opisanog sistema.

U SAGS-u ne postoji podrška za povezivanje sa drugim sistemima ove vrste. Međutim, u projektu `model` je implementiran interfejs koji bi to olakšao. Na jednostavan način se programskim putem mogu kreirati i povezivati elementi i moduli, tako da postoje dobri osnovi za importovanje poznatih fajl formata. Tako bi korisnici koji imaju iskustva u radu sa popularnim alatima za projektovanje logičkih kola kao što su VHDL i Visio mogli da nastave da rade sa njima, a dobili bi i dodatnu mogućnost da testiraju svoju šemu i eksperimentalno provere njenu ispravnost.

Sledeća stvar koja bi korisnicima olakšala posao je jezik za definisanje ponašanja elemenata. Trenutna implementacija uslovljava korišćenje programskog jezika Java što može da dovede do zabune ako korisnik ne vlada dobro ovim jezikom. Problemi koji najviše bune ovakve korisnike su ne poznavanje tipova i operatora i ne razlikovanje aritmetičkih operatora od logičkih. Osim toga novi jezik koji bi se uveo doveo bi do bolje čitljivosti unetog kôda.

Kao još jedna korisna stvar nameće se izvršavanje simulacije kao internet stranice. Program bi tada bio dostupan svim korisnicima Interneta. Svako bi mogao jednostavno da proba ovaj program bez potrebe da preuzima dodatni softver, što je velikoj većini ljudi dosadan postupak. A danas se na svakom računaru nalazi bar jedan veb pregledač.

ZAKLJUČAK

Rezultat ovog rada predstavlja deo sistema za projektovanje logičkih kola i simulaciju njihovog ponašanja. Deo koji je ovde rađen je izvršno okruženje simulacije, deo učitavanja komponenti koji se odnosi na ponašanje elemenata. U drugom radu je urađen dizajner simulacije, učitavanje ostalih komponenti i njihovo čuvanje u definisanom fajl formatu.

Ciljni korisnici ove aplikacije su profesori i studenti na tehničkim fakultetima. Profesorima će biti olakšana priprema laboratorijskih vežbi. Studenti dobijaju uvek isto okruženje za rad, a i jedni i drugi će dobiti sa novom mogućnosti u nastavi – studenti će moći sami da projektuju logička kola i da posmatraju njihovo ponašanje.