Nebojša Šabović

PINUSD – simulator digitalnih kola opisanih u jeziku C++

Realizovan je simulator koji svojom organizacijom olakšava opis kompleksnog hardvera i grupisanje u familije. Koncipiran je tako da je lako nadogradiv za rad sa analognim kolima. Simulacija je asinhrona, što znači da u okviru simulacije vreme teče kontinualno. Problem povratne sprege je rešen tako što nije dozvoljena promena izlaza u nultom vremenu, već svako kolo mora imati kašnjenje, koja imaju sva realna kola zbog fizičkih karaktristika materijala od kojih su načinjeni. Program je realizovan u dve celine - samu simulaciju i monitore koji mogu čitati i menjati parametre simulacije. Za komunikaciju se koristi TCP protokol.

***Ključne reči**. elektronska kola: simulacija; objektna orijentacija*

Problem simulacije elektronskih kola

Simulator kola je jedan od alata koji se u procesu projektovanja hardvera često koriste. On treba da bude sposoban da relativno brzo i unutar dozvoljenih tolerancija predvidi kako će se kolo ponašati u realnim situacijama. Takođe, služi da oslobodi projektanta vremenski i materijalno zahtevnih radnji kao što su proračun tolerancija, provera konflikata u kolu pre testiranja i proizvodnja prototipa. Simulatorom se problem podiže na viši nivo apstrakcije i time omogućava veće usredsređenje na suštinu problema.

Pri projektovanju simulatora treba obratiti posebnu pažnju upravo na nivo apstrakcije, jer je to osnovna karakteristika po kojoj se simulatori razlikuju. Detaljan model simuliranog objekta može iziskivati previše vremena za simulaciju, a kao rezultat dati opis kola koji za potrebe simulacije sadrži suvišne informacije. Kod specijalizovanih simulatora, kao što su simulatori digitalnih kola, model elemenata simulacije može se tako uprostiti, da se posmatraju samo tri nivoa napona, a funkcije prelaza elemenata se opisuju jezikom matematičke logike. Ako posmatramo signale, to je do-

Nebojša Šabović (1980), Beograd, Vladetina 3/31, učenik 3. razreda Pete beogradske gimnazije

Predrag Minić (1978), Jagodina, Kneza Miloša 108/20, student prve godine Elektrotehničkog fakulteta

voljna tačnost, tako da se kod simulatora digitalnih kola, za glavni činilac greške uzima rezolucija vremena sa kojom mogu da rade. Ovakve simulacije nailaze na probleme kod opisa kompleksnijih kola. Iako se teorijski svaka logička funkcija može opisati, kod velikog broja ulaza kola Booleove funkcije postaju suviše složene i za projektanta i za računar. U tom slučaju, pogodno je opisati ponašanje jezikom koji je specijalizovan za takav vid simulacije.

S obzirom da su ovakvi problemi prisutni već duže vreme, primenjena su razna rešenja. Među najpoznatijim rešenjima, u obliku jezika za opis, ističu se VHDL i Verilog.

VHDL (VHSIC Hardware Description Language – jezik za opisivanje VHSIC hardvera; VHSIC – Very High Speed Integrated Circuit, odnosno integrisana kola velike brzine), projekat vlade SAD iz 1980. godine, je robustan jezik prihvaćen od strane IEEE kao standard za opis digitalnih kola (Ashenden 1980). Karakterišu ga komplikovana i neproduktivna sintaksa, te nije podesan za opisivanje složenog hardvera. Verilog je takođe jezik za opis digitalnih kola i konceptualno je sličan VHDL-u. Ovi jezici predstavljaju standardne strukturalne programske jezike, koji ne mogu elegantno rešiti problem opisa elektronskih kola.

Kao alternativa, može se upotrebiti objektno orijentisan jezik, čime se obezbeđuje mnogo lakši opis kola zbog integracije strukture opisa sa funkcijom kola. Jedna takva realizacija jezika za opisivanje kola jeste PINUSD.

Objektna orijentacija

Simulator digitalnih kola PINUSD je implementiran u objektno orijentisanom jeziku C++. Klase se formiraju i organizuju tako da odgovaraju modelima elemenata. Osnovne klase predstavljaju opšte definicije elemenata kola, i njihovim nasleđivanjem kreiraju se konkretni elementi. Objektnom orijentacijom je postignut prirodniji opis elemenata kola, a polimorfizmom se omogućava kreiranje familija elektronskih komponenti.

Jedna od osnovnih klasa u PINUSD je *Value*. Ona sadrži opis vrednosti koja može biti izmerena u simulaciji. Primer za value bi bile vrednosti napona ili struje, ali i stanje bita ili registra procesora. Klasa je apstraktna, za svaku vrstu vrednosti se nasleđuje u nove klase. Time se simulator ne ograničava samo na simulaciju digitalnih kola, već je moguća nadgradnja do simulatora analognih kola.

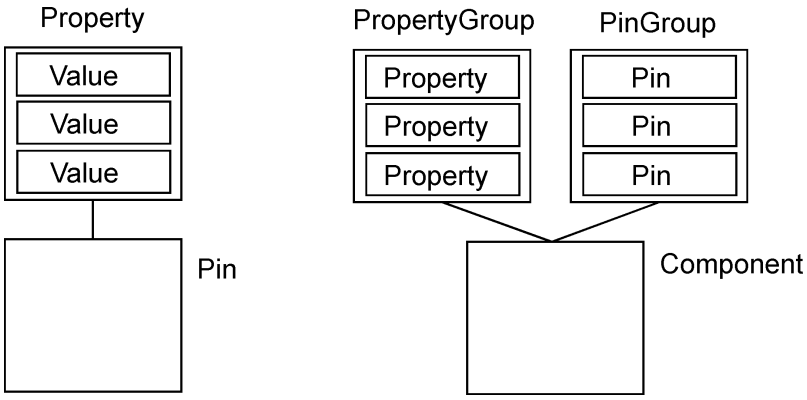
Merljiva tačka u simulaciji može sadržati više vrednosti. Takvu tačku predstavlja klasa *Property* (na primer, pin na integrisanom kolu; u simulaciji analognih kola, jedan property bi sadržao vrednosti i napona i struje). Kako je PINUSD za sada okrenut isključivo digitalnim kolima, osnovna merna klasa je *PropertyTriState*, nasleđena od klase property, za koju su

definisana tri logička stanja. To su stanja logičke nule, jedinice, ili stanje visoke impedanse. Neki autori definišu i nivoe snage (najčešće tri nivoa) za poboljšan rad sa žičnim kolima (Ashenden 1980, Litovski *et al*. 1992). Međutim, u PINUSD, nivoi snage nisu definisani, već postoje tri načina ponašanja žičnih veza (čvorišta). Pri konfliktu na čvorovima, preovladava logička nula ili jedinica, ili se prijavljuje greška u simulaciji, što se određuje opisom kola. Potpuno realnu simulaciju žičnih veza moguće je ostvariti proširenjem klase PropertyTriState, tako što bi se u nju implementirali nivoi snage.

Svako kolo u simulaciji ima vrednosti koje sa njega mogu da se mere. To su razni propertyji, kao što su registri procesora, intenzitet svetlosti na LE-diodi, ili sadržaj memorijskih elemenata. U opisu kola definišu se propertyji koji mogu biti posmatrani na kolu.

Posebni propertyji su pinovi. Klasa *Pin* nasleđuje klasu property i omogućava povezanost sa nekim drugim pinom. Promena na jednom pinu menja vrednost i na pinu sa kojim je povezan.

Na kraju, komponenta (klasa *Component*) nasleđuje grupu pinova (klasa *PinGroup*) i grupu property-ja (klasa *PropertyGroup*) (slika 1).

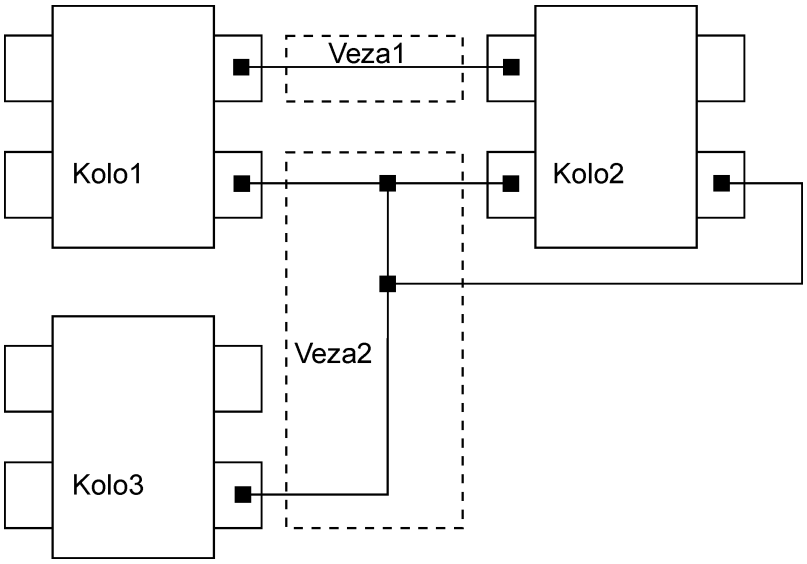


Slika 1. Hijerarhija osnovnih klasa.

Figure 1. Hierarchy of essential classes.

Šema nema čvorove kao posebne celine, već su čvorovi komponente koje se ponašaju kao što bi se ponašao čvor u šemi. U tim komponentama se definiše ponašanje žičnih kola. Dve komponente treba da budu povezane čvorom, da bi se pravilno simulirale situacije kada dva susedna pina postave različite vrednosti (slika 2).

Svaka komponenta ima svoju *execute()* funkciju koja izračunava naredne vrednosti njenih mernih tačaka, imajući u vidu prethodna stanja. Ta funkcija se izvršava svaki put kada dođe do promene na nekom propertyju ili pinu komponente. Njom se definiše ponašanje komponente.



Slika 2.
Čvorovi kao komponente.

Figure 2.
Nodes as components.

Vreme

Tok vremena se u simulatorima digitalnih kola može definisati na više načina. Dva osnovna načina simulacije vremena su sinhrona i asinhrona simulacija. Kod sinhrone, kolo se posmatra u pravilnim intervalima i tu, u zavisnosti od dužine intervala, može doći do različitih grešaka u rezultatu simulacije. U PINUSD je rađena asinhrona simulacija, i primenjuje se princip narednog događaja (Litovski *et al*. 1992), tj. stanja na izlazima se izračunavaju samo kada se promeni neko od stanja na ulazima.

Sva integrisana kola imaju kašnjenja, zbog osobina materijala od kojih su napravljena. Kola u PINUSD nemaju mogućnost promene stanja u nultom vremenu (odmah nakon promene na ulazu), već imaju obavezno kašnjenje. To kašnjenje rešava problem povratne sprege, i čini simulaciju realnijom. Svako kolo može definisati kašnjenje za svaki svoj izlaz. Prilikom dodele vrednosti pinu, u opisu se navodi kada ta promena stupa na snagu. Po uzoru na ključnu reč VHDL-a, uvedena je funkcija *after()* (slika 3).

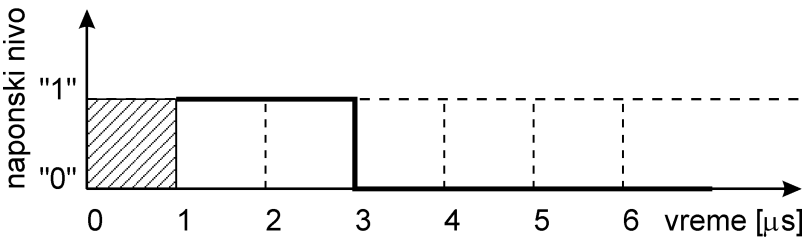
Ukoliko se vreme ne navede, podrazumeva se usvojena vrednost od 10 μs (mikrosekunde su u PINUSD označene kao uSecond).

Primer: **`Pin[2] = Hi`**

Osim što omogućava definisanje kašnjenja, funkcija after() se može iskoristiti za stvaranje vremenski zavisnih kola (takt-generatore).

Primer (slika 3):

```
Pin[1].after (uSecond(1)) = Hi
Pin[1].after (uSecond(2)) = Lo                    (1)
```



Slika 3.
Izlaz generisan naredbama (1).

Figure 3.
Output generated by statements (1).

U evaluate() funkciji kola ne može se direktno menjati stanje na pinu. Posebna celina, *EventManager*, je zadužena za to. Svaka promena vrednosti pina vrši se tako što se EventManageru zakaže vreme promene vrednosti (t_0 + kašnjenje), a on će tu vrednost postaviti nakon zadatog kašnjenja.

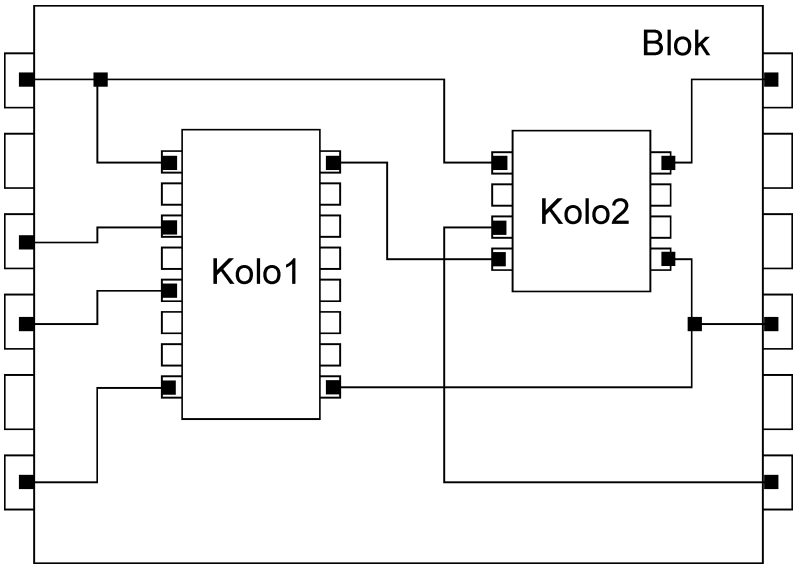
Pošto se postave početni uslovi simulacije, pozove se execute() svih komponenata. EventManager će tada imati listu zakazanih promena na pinovima. Sledeći korak simulacije biće u momentu kada je zakazana najskorija promena. Tada će EventManager promeniti zadate pinove, i pozvati execute() promenjenih komponenti, koji će zakazati nove promene. Property svake komponente se takođe ponaša kao pin, pa se i promene na njemu prijavljuju EventManageru.

Moguće je funkcijom after() zakazati uzastopne promene, ali ukoliko se zakaže promena na određenom pinu, koja treba da se desi pre već zakazanih promena na istom pinu, ove kasnije promene postaju nevažeće.

Organizacija

Radi podele šeme u organizacione celine, postoje blokovi kola, koji u sebi sadrži jedno ili više kola i njihove međusobne veze. Blok se takođe ponaša kao komponenta (component), s tim što u svojoj evaluate() funkciji poziva evaluate() funkcije svih članova (slika 4).

Blok nikako ne menja ponašanje kola u njemu, već služi samo kao organizaciona celina. On olakšava opis kola, čini ga očiglednijim, ali i ubrzava simulaciju. Takođe, omogućava i lakše deljenje simulacije na više procesa.

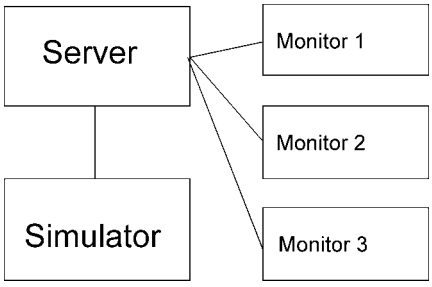


Slika 4.
Blokovi komponenta.

Figure 4.
Component blocks.

Implementacija i dalji razvoj

C++ nema mogućnost dinamičkog učitavanja klasa, pa dodavanje nove komponente zahteva i ponovno kompajliranje. Zato je simulator rešen kao klijent program koji prima komande od servera. Server, nakon svakog dodavanja komponente u sistem, kompajlira izvorne fajlove koji sadrže opis komponenti, linkuje i ponovo startuje klijenta. Server prima i prosleđuje poruke od jednog ili više monitora (programa koji postavljaju uslove simulacije i očitavaju stanje mernih tačaka) (slika 5).



Slika 5.
Interprocesna komunikacija.

Figure 5.
Interoprocess communication.

Monitori rade kao posebni procesi paralelno sa simulatorom, šalju komande koje menjaju stanje simulacije i kontrolišu tok vremena u simulacije. Komunikacija se vodi po TCP protokolu, što znači da se simulator, monitori i server ne moraju obavezno izvršavati na istom računaru. Komande su u tekstualnom formatu (pogledati prilog).

Glavni port na kome radi simulator je 28000. Ukoliko postoji više simulatora na jednoj mašini, oni se raspoređuju na slobodne UNIX por-

tove, i prijavljuju se simulatoru na glavnom portu. Svaki monitor može dobiti spisak svih simulatora na računaru ispitujući simuslator koji je na glavnom portu.

PINUSD je rađen kao otvoren sistem, sa osnovnom idejom da bude lako nadgradiv. Nasleđivanjem osnovnih klasa za vrednosti i dodavanjem integracionih algoritama u čvorove mogao bi se napraviti simulator analognih kola. Pošto se koristi C++ za koga postoje kompajleri za skoro sve platforme, PINUSD je lako portabilan. Zbog IPC-a (interprocesna komunikacija), moguće je napraviti editore šema i programe za nadgledanje simulacije. Ovim bi se zaokružio paket simulacije digitalnih elektronskih kola. Sledeći korak u samom simulatoru bi bilo omogućivanje distrubuiranosti simulacije.

Literatura

Litovski, V.C., Petković, P.M., Milovanovic, D.P., Milenković, S.Lj. Damjanović, M.S. 1992. *CADEC 2: Simulacija i projektovanje topologije integrisanih kola*. Beogad: Nauka

Ashenden, P.J. 1980. *The VHDL Cookbook*. Adelaide: University of Adelaide, Dept. Computer Science.

Nebojša Šabović and Predrag Minić

PINUSD – Simulator of Digital Electrinic Circuits in C++

PINUSD is a simulator of digital electronic circuits, which are described in object-oriented language (C++). It's organisation simplifies the description of complex hardware and grouping into families of circuits. The base classes used to describe values measured in simulation are abstract, which makes PINUSD easily upgradable for work with analog circuits. The connections are watched as separate circuits and their behaviour can be changed. The passage of time is continual (the simulation is asynchronous), and implemented on the 'next event' principle. As in real circuits, all simulated circuits must define delays on their outputs, which solves the problem with feedback and makes simulation more realistic. The simulation consists of the simulator and monitors that can read and modify the parameters of simulation and control time in simulation. Simulator and monitors communicate by TCP protocol, which enables the simulation to work on several computers. The choice of C++ as a commonly used language makes for good portability of PINUSD.

Prilog: Komande

`ADDC Ime Tip [Parametri]`

Dodaje komponentu tipa *Tip*, po imenu *Ime*. Moguće je komponenti u konstruktoru proslediti parametre, što koriste blokovi da bi znali koje komponente da uključe u svoj sastav.

`RMC Ime`

Briše komponentu po imenu *Ime*

Ove dve funkcije presreće server i potrebi rekompajlira klijenta

`CONNECT Komponenta1.BrPin Komponenta2.BrPin`

Kreira vezu između datih pinova. Ukoliko je neki od pinova bio već povezan, otkačiće ga i vratiti upozorenje.

`NEXT [Vreme]`

Izvršava simulaciju do vremena ako se navede, stopirajući na prvoj promeni, ako se ona desi pre zadatog vremena. Ukoliko se ne navede, izvršava sledeći vremenski interval. Vraća vreme nakon kraja simulacije.

`GET Komponenta.Property[.Value]`

Vraća tekstualnu vrednost valuea po datom imenu komponente. Ako se konkretno ime valuea ne navede, vraća vrednost samog propertyja, što je najčešće tekst svih valueova tog propertyja.

`SET  Komponenta.Property[.Value]`

Postavlja vrednost property-ja ili nekog valuea tog propertyja.

`INFO [Ime[.Property]]`

Ukoliko se *Ime* ne navede, vraća spisak komponenata. Inače vraća spisak propertyja ili valueova date komponente.

`SAVE`

Vraća trenutno stanje celokupne simulacije. To predstavlja vrednost svih promenjivih – informacije pomoću kojih se simulacija kasnije može dovesti u isto stanje. Monitor može snimiti vraćenu vrednost u fajl i kasnije je učitati.

`RESTORE [Podaci]`

Postavlja simulaciju u stanje koje je opisano podacima. Format podataka je isti kao kod SAVE naredbe – fajl sa kompajliranim komponentama pre linkovanja sa simulatorom praćen stanjima svih komponenata. Stanja komponenata se dobijaju pomoću funkcija SaveState/RestoreState koje svaka komponenta ima. Ukoliko klasa komponente sadrži pointere, da bi ih učitala/snimila mora da nasledi te funkcije i omogući snimanje tih podataka.

CLONE

Klonira simulator i server, vraćajući adresu novog servera. Ako jedan monitor promeni simulaciju, svi ostali će biti obavešteni o tome. Ali, ukoliko monitor želi da izvrši neke preglede koji će uticati na stanje simulacije, a da pri tome ne utiče na glavnu simulaciju, funkcija CLONE omogućava kloniranje servera.

QUIT

Šalje zahtev za gašenje serveru. Server će poslati poruku svim monitorima i ako se svi slože, ugasiti server.

Nekoliko komandi omogućuju komunikaciju među serverima:

LIST

Vraća listu svih servera koji su prijavljeni. Server koji nije na glavnom portu vraća grešku ukoliko mu se prosledi ova komanda.

REGISTER Adresa Ime

Ovu komandu koristi server prilikom startovanja, da bi se prijavio serveru na glavnom portu. *Adresa* je adresa servera koji se prijavljuje, *Ime* je ime simulacije i zadaje se prilikom startovanja simulatora. Server koji nije na glavnom portu vraća grešku ukoliko mu se prosledi ova komanda.

VER

Vraća ime i verziju startovanog simulatora. Glavni server može proveriti da li se na toj adresi nalazi server koji mu je prijavljen.

Pošto simulatori ne šalju jedni drugima informacije osim REGISTER-a, simulator na glavnom portu periodično proverava status prijavljenih servera što ih pita za verziju i proverava da li se ona slaže sa prijavljenom. Ostali simulatori takođe povremeno proveravaju glavni port. Prvi koji ustanovi da je glavni simulator odsutan (glavni port slobodan) zauzima glavni port, a ostali se prijavljuju ponovo.

Kao odgovor, simulator može poslati:

ERROR [Tekst]

Greška u izvršavanju komande.

OK [WARN Upozorenje]
[UPDATED Ime[.Property[.Value]]] ...] [DATA Odgovor]

OK označava da je komanda izvršena. WARN označava upozorenje – grešku koja nije onemogućila izvršavanje komande. Ukoliko je UPDATED prisutan, komanda je promenila stanje simulacije. Ukoliko su navedena imena komponenti, ili njihovih propertyja, znači da je komanda izmenila samo stanje navedenih objekata. Ukoliko je komanda zahtevala odgovor, iza DATA stoji tekst odgovora.