UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Matjaž Levstik

# Analiza načrtovalskega jezika SystemC

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: izr. prof. dr. Miha Mraz

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00065/2011
Datum: 03.02.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MATJAŽ LEVSTIK**

Naslov: **ANALIZA NAČRTOVALSKEGA JEZIKA SYSTEMC**
**ANALYSIS OF SYSTEMC DESIGN LANGUAGE**

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Kandidat naj v svojem delu analizira delovanje načrtovalskega jezika SystemC, ki je namenjen snovanju logičnih struktur na višjih nivojih abstrakcije. Pri tem naj kandidat preizkusi delovanje jezika v praksi in v omenjenem okolju realizira 4 bitni seštevalnik.

Mentor:

prof. dr. Miha Mraz

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

# IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Matjaž Levstik, z vpisno številko **63010085**, sem avtor diplomskega dela z naslovom:

*Analiza načrtovalskega jezika SystemC*

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Mihe Mraza,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 15. avgusta 2011 Podpis avtorja:

# Zahvala

Zahvaljujem se vsem, ki so mi na kakršen koli način pomagali pri izdelavi diplomskega dela. Posebna zahvala pa velja mentorju, izr. prof. dr. Mihi Mrazu, za odlično svetovanje in napotke pri pisanju diplomskega dela.

Ob uspešno opravljenem diplomskem delu se želim zahvaliti tudi svojim staršem, ki so verjeli vame.

# Kazalo

# Seznam uporabljenih kratic in simbolov

| | |
|---|---|
| ANSI: | Ameriški nacionalni inštitut za standardizacijo (angl. *American National Standards Institute*), |
| ASIC: | integrirana vezja za določen namen (angl. *application specific integrated circuit*), |
| BFM | model funkcijskega vodila (angl. *bus functional model*), |
| DSP: | digitalno procesiranje signalov (angl. *digital signal processing*), |
| ESL: | Načrtovanje elektronike na nivoju sistema (angl. *electronic system level design*), |
| FIFO: | prvi noter, prvi ven (angl. *first-in, first-out*), |
| FPGA: | programirljivo polje vrat (angl. *field programmable gate array*), |
| GDSII | grafični sistem zbirke podatkov II (angl. *graphic database system II*) |
| HDL: | jezik za opis strojne opreme (angl. *hardware description language*), |
| IC: | integrirano vezje (angl. *integrated curcuit*), |
| IEEE: | Inštitut inženirjev elektrotehnike in elektronike (angl. *Institute of Electircal and Electronics Engineers*), |
| LIFO | zadnji noter, prvi ven (angl. *last-in, first-out*), |
| LRM: | jezikovni referenčni priročnik (angl. *language reference manual*), |
| OS: | operacijski sistem (angl. *operating system*), |
| OSCI: | The Open SystemC Initiative, |
| PO: | programska oprema, |
| RTL: | model na nivoju registrov (angl. *register transfer level model*), |
| SAM: | arhitekturni model sistema (angl. *system arhitectual model*), |
| SNSPO: | sočasno načrtovanje strojne in programske opreme (angl. *HW/SW codesign*), |
| SO: | strojna oprema, |
| SoC: | sistem na integriranem vezju (angl. *system on chip*), |
| STL: | standardna knjižnica predlog (angl. *Standard template library),* |
| SVC: | verifikacijska knjižnica SystemC (angl. *SystemC verification library*), |
| TLM | modeliranje na nivoju transakcij (angl. *transaction level modeling*). |

# Povzetek

Diplomsko delo je osredotočeno na analizo jezika SystemC, ki se uporablja pri načrtovanju sistemov na višjih nivojih abstrakcije. Predstavljeni so koraki in prednosti sočasnega načrtovanja strojne in programske opreme, zahteve, ki morajo biti izpolnjene za tovrstno načrtovanje, in primeri nekaj jezikov, ki se uporabljajo pri načrtovanju sistemov na različnih nivojih abstrakcije. Pri načrtovanju sistemov na višjih nivojih uporabljamo višjenivojska načrtovalska orodja in jezike, ki interpretirajo obnašanje sistema z uporabo algoritmov ter ga prevedejo v izvršljiv programski model strojne opreme.

Jedro diplomskega dela predstavlja opis jezika SystemC in razlogi za njegovo smotrnost uporabe v praksi. Najprej se seznanimo z namestitvijo in delovanjem prevajalnika kode SystemC. Sledi opis razredov SystemC, ki so potrebni za načrtovanje strojne opreme. Ti omogočajo sočasno izvrševanje procesov modela, hierarhijo, upravljanje komunikacij in vsebujejo podatkovne tipe, potrebne za načrtovanje strojne opreme. Povzete so najpomembnejše lastnosti arhitekture SystemC, kot so moduli, metode, podatkovni tipi, dogodki, občutljivosti, priključki, vmesniki in kanali. Predstavljeno je tudi simulacijsko jedro SystemC, njegove faze in funkcionalnost. Na primeru vrat XOR je demonstrirana uporaba osnovnih gradnikov jezika, hierarhija in funkcionalnost. V zadnjem poglavju je predstavljena izdelava modela 4-bitnega seštevalnika na višjem nivoju abstrakcije.

## Ključne besede:

SystemC, načrtovanje sistema, abstrakcija, visokonivojska obravnava sistema, modeliranje, seštevalnik

# Abstract

The focus of the diploma thesis is set on analysing SystemC language which is used in design of systems at higher levels of abstraction. Procedures for hardware and software co-designing are described; the advantages and requirements of such a design are listed, along with some language examples used in designing the systems at different levels of abstraction. With higher levels of abstraction we use higher-level design tools and languages which interpret the system behaviour using algorithms and translating them into executable software model of hardware.

The core of the thesis represents the description of SystemC language and reasons for its wise use. At first we get acquainted with the installation and operation processes of the SystemC code compiler. In the following segment we turn to the description of the SystemC classes that are necessary for hardware design. They allow simultaneous processes execution of a model, hierarchy, communications management and contain data types needed for hardware design. The most important features of the SystemC architecture are summarized, i.e. modules, methods, data types, events, sensitivities, ports, interfaces and channels. The SystemC simulation kernel, its phases and functionality are also presented. In the case of XOR gate, the use of basic building blocks of the language, hierarchy and functionality is demonstrated. In the last chapter, the design of a 4-bit adder model on the higher level of abstraction is presented.

## Key words:

SystemC, system design, abstraction, high-level system design, modelling, adder

# 1 Uvod

Tema diplomske naloge je analiza jezika SystemC, ki je implementiran s programskim jezikom C++. Je enoten visokonivojski jezik za načrtovanje in verifikacijo strojne opreme. Načrtovalci lahko uporabljajo objektno orientirane zmožnosti za načrtovanje strojne opreme in potrebujejo le znanje iz programiranja v C++. SystemC dovoljuje načrtovanje na višjih nivojih abstrakcije, predvsem na sistemskem nivoju. To omogoča občutno hitrejšo arhitekturno analizo sistema in preoblikovanje sistema, kot je to mogoče na nižjem, detajlnem registrskem nivoju. Podpira opis komponent programske in strojne opreme v enotnem okolju ter opis strojne opreme na funkcionalnem nivoju in na nivoju registrov, kar omogoča sočasno načrtovanje strojne in programske opreme.

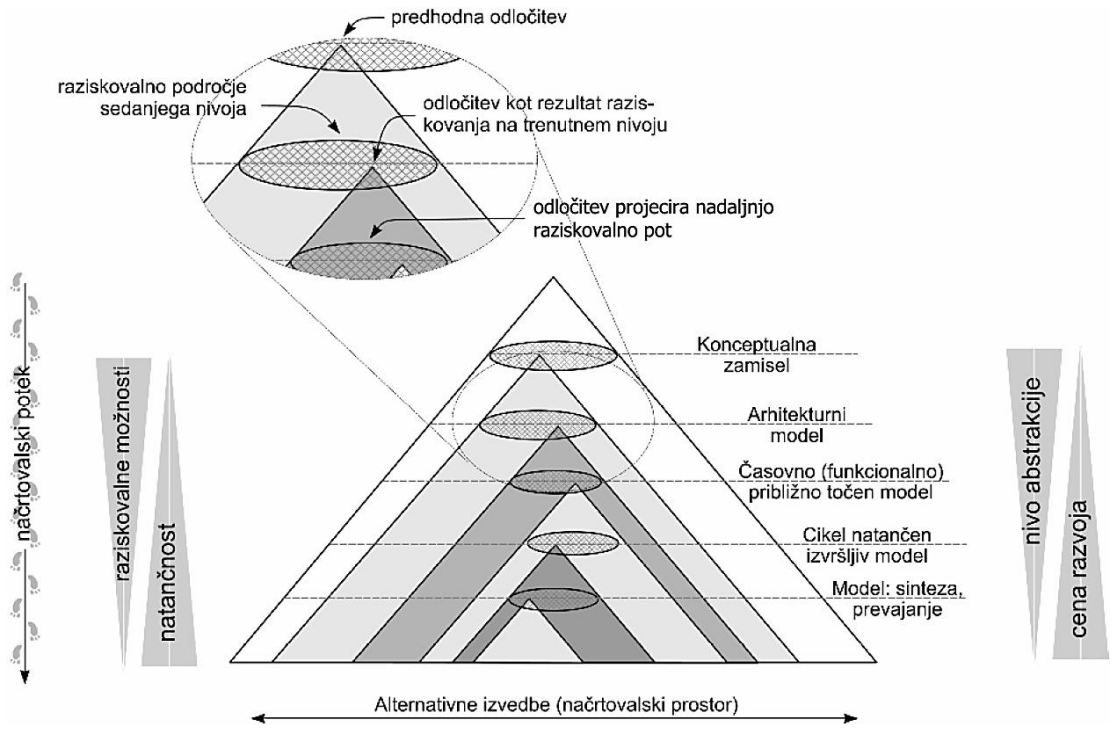
Medtem ko je C++ primeren za programiranje programske opreme, obstajajo trije razlogi, zakaj ni ustrezen za modeliranje komponent strojne opreme. C++ ne podpira sočasnosti, strojnih podatkovnih tipov in predstavitve časovnega zaporedja dogodkov. SystemC sestavljajo knjižnice razredov C++ in simulacijsko jedro, ki omogočajo modeliranje sistemov v C++. Z njimi lahko modeliramo sočasne procese, čas in dogodke ter vsebujejo bogate podatkovne tipe za strojno opremo. Opisovanje hierarhije pa dosežemo z uporabo modulov.

Opis vmesnika in funkcionalnosti modela je mogoča na več nivojih. Ti nivoji modelov so:

- časovno nedoločen model (angl. *untimed functional*), ki opisuje vmesnik in funkcionalnost brez modeliranja časa (izvajanje v ničelnem času),
- časovno določen model (angl. *timed functional*), pri katerem uporabljamo modeliranje zakasnitev pri računanju in pri prenosu podatkov oziroma komunikaciji,
- model, natančen na nivoju ciklov (angl. *cycle accurate*), kjer je delovanje vmesnika specificirano na nivoju ciklov sistemske ure,
- model, natančen na nivoju registrov (angl. *register transfer accurate*), kjer je funkcionalnost popolnoma časovno določena.

Vrste modelov po nivojih abstrakcije si od najvišje do najnižje sledijo takole:

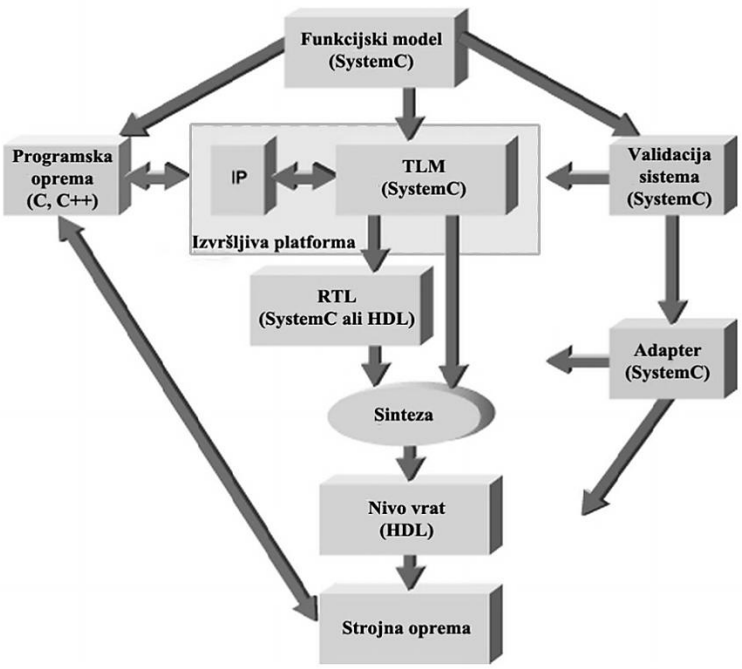
- Funkcionalni model (angl. *functional model*) je izvršljiva specifikacija strojne in programske opreme za raziskovanje algoritmov. Model je časovno nedoločen, le za oceno zmogljivosti je časovno določen. Poznamo arhitekturni in zmogljivostni model sistema.
- Model na nivoju transakcij (TLM, angl. *transaction level model*) se uporablja za modeliranje izvršljive platforme in tipično opisuje le strojno opremo. Obnašanje modeliramo s transakcijami. Vmesnik in funkcionalnost sta časovno določena.
- Modeli na nivoju sistema (angl. *system level model*) se imenujejo modeli na ravneh abstrakcije nad RTL.
- Model funkcijskega vodila (BFM, angl. *bus functional model*) je primeren za modeliranje in simulacijo procesorjev. Vmesnik je določen na nivoju ciklov.
- Model na nivoju registrov (RTL, angl. *register transfer level*) se uporablja za detajlni opis strojne opreme. Vmesnik je natančen na nivoju ciklov, funkcionalnost pa na nivoju registrov.
- Model na nivoju vrat (angl. *gate level model*) je opisan s shemo ali Boolovimi enačbami. Načrtovanje kompleksnih sistemov je zahtevno, simulacija pa počasna.
- Model na nivoju tranzistorjev vsebuje največ fizikalnih podrobnosti. Načrtovanje je zahtevno in zamudno, simulacija pa je počasna.




Slika 1: Piramida abstrakcije in načrtovalski potek [7].

SystemC se lahko uporablja na vseh višjih nivojih, vključno z nivojem logičnih vrat. Modele, ki so modelirani na višjih nivojih abstrakcije kot RTL, imenujemo tudi modeli na nivoju sistema. Najpogosteje se SystemC uporablja za izdelavo izvršljive specifikacije sistema, ki ga razvijamo. Prednosti takega višjenivojskega izvršljivega modela sistema sta hitrejši razvoj programske opreme ter hitrejši razvoj in verifikacija strojne opreme, s čimer se posledično izognemo različnim napakam v sistemu. Slika 1 prikazuje postopek načrtovanja sistema in alternativne možnosti njegove izvedbe. Ko se pomikamo navzdol po nivoju abstrakcije, cena razvoja narašča, poleg tega pa se zmanjšujejo alternativne možnosti izvedbe ter veča natančnost. Zato je zelo pomembno, da na višjih nivojih abstrakcije z uporabo sistemskega modela dobro preverimo funkcionalno in arhitekturno zasnovo sistema. Z uporabo enotnega okolja za modeliranje, ki se uporablja na večih nivojih abstrakcije, pa se lažje vračamo nazaj na predhodne nivoje, če ugotovimo, da nam trenutna izvedba sistema ne ustreza. Ker SystemC omogoča modeliranje v širokem nivojskem spektru, je zelo primeren za načrtovanje sistemov.

Slika 2 prikazuje tipičen načrtovalski potek z uporabo SystemC. Načrtovanje se začne s funkcijskim modelom, vendar večina kodiranja s SystemC poteka za model na nivoju transakcij. TLM največkrat služi kot izvršljiva platforma za programsko opremo. Ta platfoma je dovolj hitra in natančna, da na njej razvijajo in testirajo programsko opremo. Model TLM se uporablja tudi za verifikacijo sistema. Nadalje se model TLM uporabi za sintezo ali pa se ga prevede v model RTL, ki se ga prav tako lahko uporabi za sintezo.




Slika 2: Načrtovalski potek z uporabo SystemC [4].

# 2 Načrtovalski jezik SystemC

Pri načrtovanju sodobnih elektronskih sistemov in ob povečujoči se načrtovalski kompleksnosti se za potrditev sistemskih konceptov zahteva hitro izvršljivo specifikacijo sistema. Potreben je ustrezen opisni jezik, ki nudi zadostne nivoje abstrakcije, integracije strojne in programske opreme ter nenazadnje tudi zadostno zmogljivost. Da bi bila možna enovita uporaba orodij za načrtovanje sistemov, storitev in blokov intelektualne lastnine, je potreben tudi skupen načrtovalski jezik, ki bi izboljšal celotno produktivnost načrtovanja digitalnih elektronskih sistemov. Zaradi tega je bil razvit jezik SystemC. Ta je namenjen opisu sistemov na višjih nivojih abstrakcije. Z njegovo uporabo je na različnih nivojih abstrakcije mogoče predstaviti funkcionalnost, komunikacijo, strojno in programsko opremo. SystemC predstavlja standardiziran jezik za modeliranje na sistemskem nivoju in izmenjavi intelektualne lastnine med vsemi nivoji abstrakcije.

## 2.1 Paradigmi ESL in TLM

Današnji sistemi vsebujejo specifično strojno in programsko opremo za aplikacije, ki tečejo na takih sistemih. Strojna in programska oprema sta ponavadi načrtovana sočasno v kratkih časovnih rokih in z zahtevami po čim večji zmogljivosti in čim manjši porabi energije. Potrebno je temeljito funkcionalno in arhitekturno preverjanje, da se izognemo dragim in včasih katastrofalnim napakam v napravah ali sistemih. V nekaterih primerih take napake lahko rezultirajo v propad podjetja ali organizacije, ki je tak sistem razvila. Tak sočasen in multidisciplinaren pristop k razvoju kompleksnih sistemov imenujemo načrtovanje elektronike na nivoju sistema (ESL, angl. *electronic system level design*). Sočasno načrtovanje na nivoju sistema ima stranske učinke, ki ne vplivajo le na organizacijo načrtovanja podjetja, ampak tudi na celoten poslovni model in način, kako podjetje komunicira s strankami in dobavitelji.

ESL se uporablja pri modeliranju sistemov na višjih nivojih abstrakcije z uporabo orodij in metodologij, ki zvišujejo nivo abstrakcije modela iz logičnega nivoja na nivo argoritmov in

transakcij. Pri načrtovalskem procesu na višjem nivoju uporabljamo višjenivojska načrtovalska orodja in jezike, ki interpretirajo obnašanje sistema z uporabo algoritmov ter ga prevedejo v model strojne opreme. Nato tak programski model strojne opreme analiziramo, optimiziramo in arhitekturno preverjamo, šele nato izdelamo model na nivoju registrov z uporabo nižjenivojskih jezikov za opis strojne opreme (HDL, angl. *hardware description language*), kot sta na primer VHDL ali System Verilog. ESL zahteva, da programski jezik izpolnjuje določene zahteve, ki so drugačne od zahtev pri jeziku za opis strojne opreme. SystemC izpolnjuje zahteve, ki so potrebne za sočasno načrtovanje strojne in programske opreme na višjih nivojih abstrakcije.

### 2.1.1 Načrtovanje elektronike na nivoju sistema (ESL)

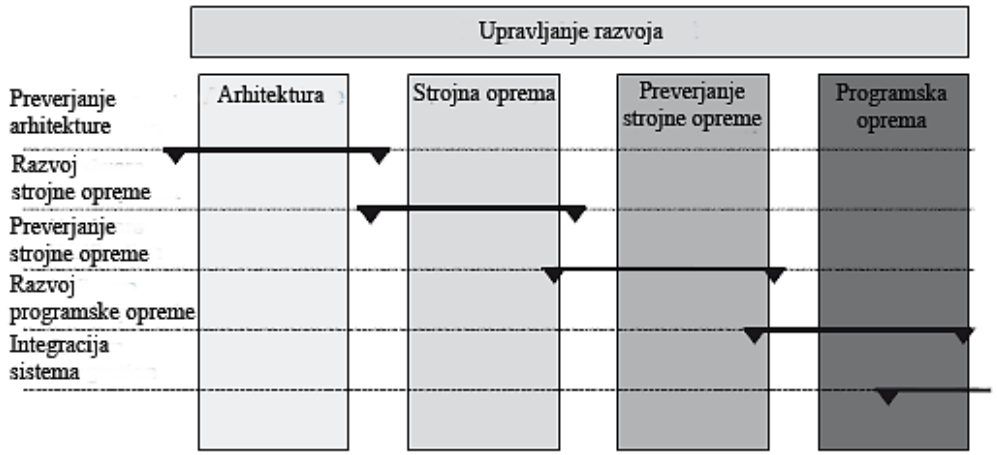
Tehnike ESL so se razvile zaradi vse večje kompleksnosti in krajših ciklov načrtovanja v industriji. Današnji sistemi, integrirana vezja (IC, angl. *integrated circuit*) in programirljivo logična vezja (FPGA, angl. *field programmable gate array*) postajajo večji in potrebno je veliko kompromisov pri optimizaciji sistemske in programske opreme ter celotne zmogljivosti sistema. Današnja integrirana vezja pogosto presežejo 10 milijonov vrat, kar pomeni več kot stotisoč vrstic kode modela na nivoju registrov. Vezja, ki jih razvijajo danes in jih bomo v prihodnosti uporabljali, pa bodo presegla sto milijonov vrat, kar bi pomenilo približno milijon vrstic RTL kode.

Eden izmed razlogov za vedno krajše cikle načrtovanja je tudi sam življenjski cikel izdelka na trgu. Če želimo danes v trgovinah najti mobilni telefon ali prenosni računalnik, ki je bil pred časom najnovejši in najboljši, ga bomo iskali zaman. Krajši cikel načrtovanja pomeni tudi več sredstev in stroškov. Projekt zahteva več komunikacije med razvojnimi skupinami, več orodij, ljudi itd. ESL in TLM pomagata k zmanjševanju stroškov razvoja z bolj učinkovito komunikacijo in z manjšim številom predelav.

Starejše metode načrtovanja so temeljile na programskem modelu, ki se je imenoval arhitekturni model sistema (SAM, angl. *system arhitectual model*), napisan v C, Java ali podobnem jeziku. Tak model je služil kot komunikacijsko orodje med razvojnimi skupinami (skupino za razvoj strojne opreme, skupino za razvoj programske opreme, skupino za razvoj algoritmov itd.).

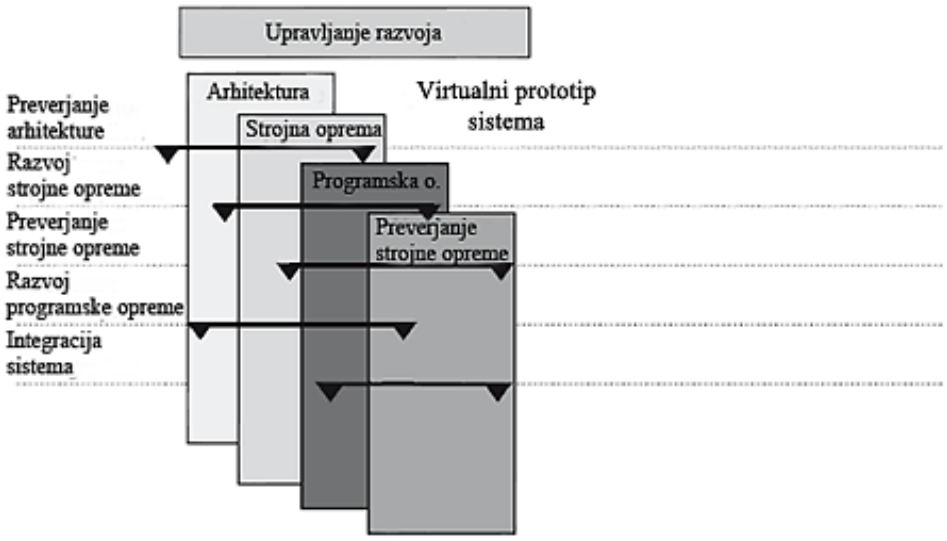
Tradicionalni pristopi načrtovanja sistemov so pogosto ločevali razvojne skupine, ki so delale serijsko. Taka skupina je papirnate specifikacije predala naslednji skupini »preko zida«, ki je

morala preučiti specifikacije sistema, da je lahko nadaljevala z razvojem na nižjih nivojih. Tak pristop se imenuje načrtovanje v slapu in je prikazan na sliki 3.




Slika 3: Tradicionalni pristop načrtovanja v slapu [1].

ESL pristop uporablja že obstoječe metode, ki jih dopolnjuje z virtualnim prototipom sistema ali TLM modelom sistema, kar različnim razvojnim skupinam omogoča, da sistem gradijo paralelno, kot je prikazano na sliki 4.




Slika 4: ESL pristop k paralelnemu načrtovanju [1].

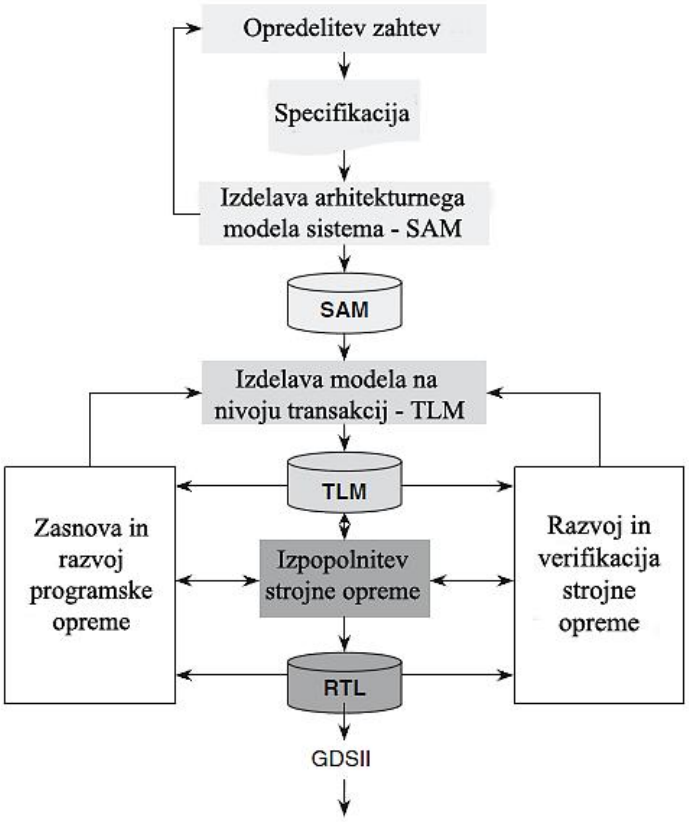
S paralelnim načrtovanjem sistemov končajo vse skupine razvoj v krajšem času. Glavni namen ESL pa je predvsem predčasno dokončanje programske opreme. Če lahko podjetje

končni produkt ponudi trgu predčasno, je to zanj velika finančna in strateška prednost. Poleg tega neuporaba metod ESL lahko privede do sistema, ki ima premajhno ali preveliko zasnovo. Pri premajhni zasnovi je produkt lahko brez napak, vendar morda ne bo zadovoljil uporabnikovih zahtev (ne bo dovolj hiter, ne bo imel funkcij, ki jih uporabnik potrebuje itd.). Prevelika zasnova pa vodi v večjo porabo sredstev in časa, poleg tega pa je lahko tak sistem zaradi večje kompleksnosti manj zanesljiv.

Virtualni prototip sistema tako omogoča, da različne skupine načrtovalcev razvijajo paralelno različne komponente modela na različnih nivojih abstrakcije. Na koncu sestavijo te komponente skupaj v celoten model sistema. Tak model sistema se uporablja kot izvršljiva platforma, ki tipično opisuje strojno opremo in se imenuje model na nivoju transakcij (TLM).

### 2.1.2 Modeliranje na nivoju transakcij (TLM)

TLM je osnovna tehnika načrtovanja, ki se uporablja za ESL. Osnovni koncept abstrakcijskih nivojev modela je, da je mogoče vsak sistem razdeliti na logične enote in jih neodvisno razvijati.




Slika 5: Metodologija TLM [1].

Metodologija TLM, ki jo pikazuje slika 5, se začne z uporabo tradicionalnih metod, s katerimi raziščemo zahteve in želje kupcev (raziskava trga), na podlagi katerih sestavimo specifikacijo sistema. Specifikacija je osnova za model TLM, ki vsebuje podroben opis sistema. Model TLM se ves čas izpopolnjuje, medtem ko se paralelno razvija programska in strojna oprema. Pri taki metodologiji je zelo pomembno, da če skozi celoten razvoj uporabljamo isti jezik in tehnike načrtovanja, lahko model TLM uporabimo večkrat in ga tudi izpopolnjujemo.

Modeliranje na nivoju transakcij se uporablja za:

- algoritmično modeliranje (enkripcija, video, procesiranje digitalnih signalov itd.),
- arhitekturno modeliranje (delitev programske in strojne opreme),
- virtualno izvršljivo platformo za razvijanje programske opreme (omogoča hitrejši razvoj programske opreme sistema),
- izpopolnjevanje strojne opreme,
- funkcijsko in arhitekturno verifikacijo (ali arhitektura sistema zadovolji željam kupcev).

Prednosti modeliranja na nivoju transakcij so sledeče:

- hitrejši razvoj programske opreme, ki jo lahko testriamo v zgodnji fazi, še predno je dostopna platforma sistema,
- hitrejši razvoj in verifikacija strojne opreme, na kateri lahko programsko opremo testiramo,
- ravna in neprekinjena pot načrtovanja od zahtev kupcev do končnega sistema.

### 2.1.3 Zahteve za načrtovanje na nivoju sistema

ESL in TLM zahtevata jezik, ki izpolnjuje naslednje pogoje:

- raztezanje abstrakcije skozi več nivojev,
- odprt in standardiziran jezik,
- razširjenost uporabe in znanj,
- primerne simulacijske zmogljivosti in funkcije,
- dodatna orodja,
- podpora konceptu načrtovanja TLM.

SystemC vsebuje vse elemente, ki so potrebni za ESL in TLM modeliranje sistemov, kar bom v nadaljevanju tudi opisal.

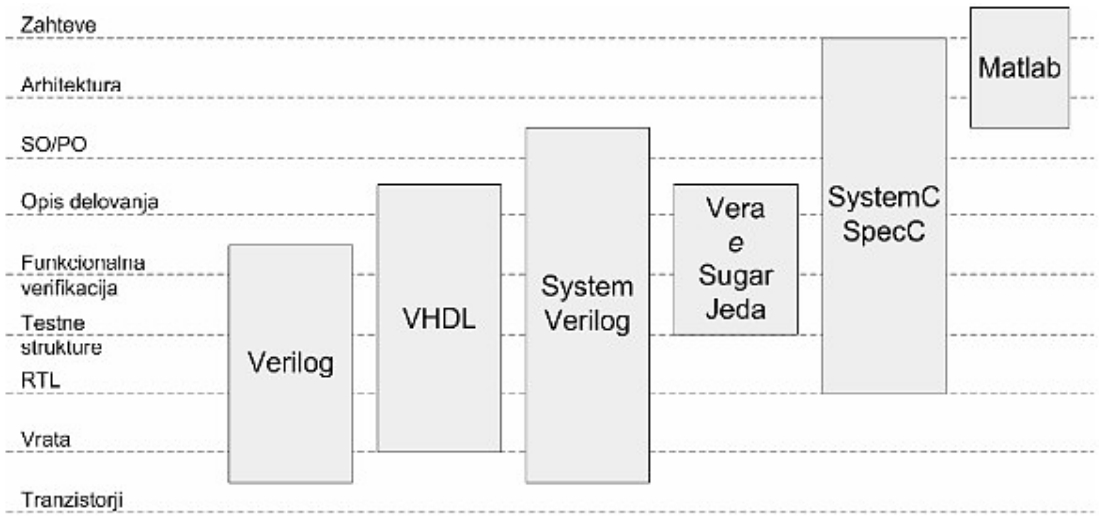
SystemC ni pravi programski jezik, ampak je knjižnica razredov znotraj zelo uveljavljenega jezika C++, ki v svojem jedru nudi podporo za različne računske modele in načrtovalske metodologije. Načrtovalske knjižnice in modeli, ki nudijo podporo za te metodologije, so obravnavane ločeno od standarda jedra jezika. SystemC je v celoti zgrajen na jeziku C++, izvorna koda referenčnega simulatorja pa je prosto dostopna na spletni strani OSCI [8], skladno z dogovorom o odprtem programiranju. Model sistema, opisanega v okolju SystemC, je mogoče prevesti in izvršiti na kateri koli platformi, za katero je na voljo ANSI skladen prevajalnik C++.

Jezik SystemC je razvilo neodvisno in neprofitno združenje OSCI, z namenom definirati napreden odprt indrustrijski standard za modeliranje na nivoju sistema, oblikovanje in verifikacijo. Izvorna koda je prosto dostopna na spletni strani OSCI [8]. Koda vsebuje več primerov implementacij s področji procesiranja signalov, podatkovnih komunikacij, simulacij procesorja (RISC) in topologijo vodila.

SystemC ni rešitev za vsak načrtovalski problem. Skupaj z verifikacijsko knjižnico SystemC (SVC, angl. *SystemC verification library*) pa nudi vse potrebne karakteristike, pomembne za načrtovanje sistemov, ki jih v drugih jezikih ne najdemo. Poleg tega SystemC lahko uporabljamo pri načrtovanju tako strojne kot programske opreme.

#### 2.1.3.1 Primerjava jezikov in nivoji abstrakcije

Pri načrtovanju sistemov se uporablja več jezikov in orodij. Pri današnjih vgrajenih sistemih se Java nadomešča s C/C++ (pri nižje nivojski programski opremi). Jezika za opis strojne opreme VHDL in Verilog se uporabljata pri simulacijah in sintezah digitalnih vezij, Vera, e in System Verilog se uporabljajo za funkcijsko verifikacijo kompleksnih integriranih vezij za določen namen (ASIC, angl. *application-specific integrated circuits*), MATLAB, Signal Processing Workbench in CoCentric System Studio pa so široko uporabljeni za razvoj algoritmov za obdelavo signalov. Slika 6 prikazuje primerjavo nekaterih jezikov in orodij za načrtovanje na nivoju sistema in njihovo občasno uporabo tudi zunaj njihove prvotne domene uporabe.




Slika 6: Nivoji uporabe nekaterih jezikov za načrtovanje sistemov [1].

#### 2.1.3.2 Standardiziran jezik

Standardiziran in industriji prosto dostopen jezik ima veliko prednosti, kot je na primer dostop do komercialnih in brezplačnih orodij ter podpore. SystemC je bil decembra 2005 standardiziran kot IEEE 1666-2005 [6] pri Inštitutu inženirjev elektrotehnike in elektronike. V letošnjem letu tečejo priprave za nadgradnjo in obnovo standarda.

#### 2.1.3.3 Razširjenost uporabe in znanj

Med sistemskimi inženirji, strojno ali programsko orientiranimi, SystemC še ni razširjen. Vendar ker temelji na C++ in objektno orientiranih tehnikah, ki jih poznajo vsi diplomanti inženirskih šol, so ta znanja dovolj za uporabo jezika SystemC.

#### 2.1.3.4 Simulacijske zmogljivosti in funkcije

Modeli sistemov se morajo izvršiti hitro, poleg tega mora jezik podpirati sočasno simulacijo in dodatne funkcije, kot na primer zmožnost upravljanja kompleksnosti modela preko hierarhije, iskanje in popravljanje napak itd. Lastnosti okolja SystemC, ki so združene z zelo dobrimi prevajalniki C++ in dobrimi tehnikami modeliranja, lahko proizvedejo model, ki ustreza hitrostnim zahtevam izvajanja.

Pri izvajanju kode modela je potrebno tudi sočasno ali paralelno izvrševanje. Med izvajanjem in delovanjem digitalnega sistema se veliko izračunov izvaja paralelno. SystemC ima simulacijsko jedro, ki daje iluzijo paralelnega izvajanja.

#### 2.1.3.5 Dodatna orodja

Ker SystemC temelji na C++, obstaja veliko dodatnih orodij in specifičnih knjižnic. Poleg tega je SystemC podprt z velikim in naraščajočim številom komercialnih ponudnikov. Večina popularnih orodij C++ je prosto dostopnih, kot na primer razvojna okolja, orodja za analizo delovanja itd., veliko pa jih je moč kupiti pri vodilnih podjetjih, ki se ukvarjajo s programsko opremo.

K večji produktivnosti pripomorejo knjižnice C++, ki so na voljo na internetu, na primer grafični algoritmi in algoritmi za digitalno procesiranje signalov. Izobilje takih knjižnic pa pripomore k lažjemu in hitrejšemu načrtovanju modela.

#### 2.1.3.6 Podpora konceptu načrtovanja TLM

Jezik za modeliranje na nivoju sistema potrebuje podporo konceptov načrtovanja TLM. Podpirati mora enostavno zamenjavo ene komunikacijske implementacije z drugo, ne da bi spreminjali vmesnik. C++ in koncept vmesnikov, ki so implementirani preko razreda virtualnih funkcij, združenih s stili kodiranja, omogočajo učinkovito načrtovanje po konceptu TLM.

## 2.2 Arhitektura načrtovalskega jezika SystemC

SystemC omogoča modeliranje strojne in tudi programske opreme z uporabo C++. Primarno se osredotoča na strojno opremo. Najpogostejše področje uporabe jezika SystemC je tako načrtovanje digitalnih elektronskih sistemov.

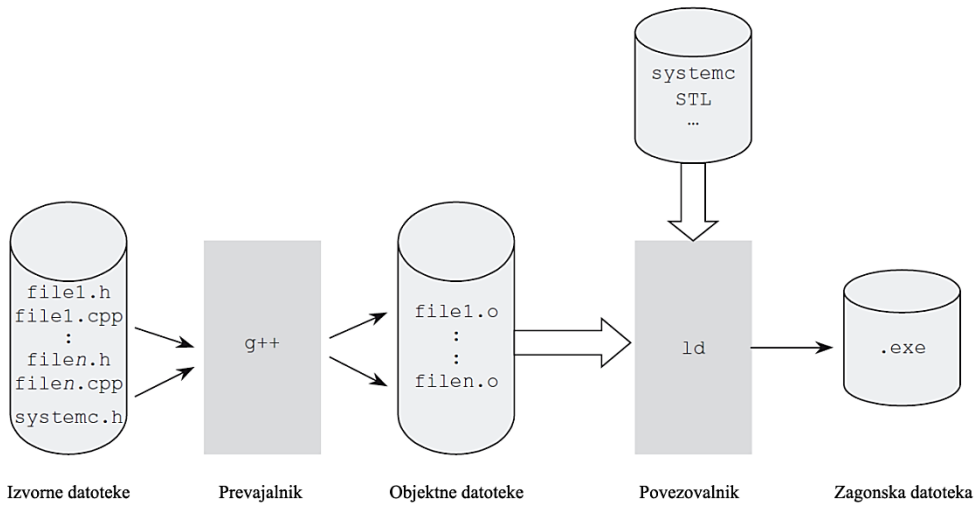
### 2.2.1 Prevajanje kode v okolju SystemC

Za prevajanje in zagon kode jezika SystemC potrebujemo prevajalnik C++ in razvojno okolje SystemC. Komponente razvojnega okolja SystemC vključujejo:

- platformo, ki podpira SystemC,
- prevajalnik C++, ki podpira SystemC,
- knjižnico SystemC,
- zaporedje ukazov za prevajalnik (npr. datoteko »make«).

Zadnja izdaja jezika SystemC (verzija 2.2) je na voljo brezplačno na spletni strani OSCI [8]. Prenos vsebuje skripte in datoteke »make« za namestitev knjižnice SystemC, izvorno kodo, primere in dokumentacijo. Namestitvene skripte so združljive s podprtimi operacijskimi sistemi in so relativno enostavne za namestitev, če sledimo priloženi dokumentaciji.

SystemC je podprt na različnih verzijah Sun Solaris, Linux, HP/UX, Windows in Mac OS X. Trenutna izdaja je podprta za prevajalnike C++, vključno z GNU C++, Sun C++, HP C++ in Visual C++. Zadnji podprti prevajalniki so navedeni v datoteki `INSTALL`.




Slika 7: Prevajanje kode v okolju SystemC [1].

Potek prevajanja programa oziroma modela SystemC je prikazan na sliki 7 za GNU C++. Prevajalnik C++ prebere vsak niz datotek kode SystemC ločeno in kreira objektno datoteko (običajno s končnico `.o`). Vsak niz datotek je sestavljen iz dveh datotek standardnih končnic `.h` (»header« datoteka) in `.cpp` (implementacijska datoteka).

Prevajalnik potrebuje informacijo, kje se nahajajo »header« datoteke SystemC, povezovalnik (angl. *linker*) ali nalagalnik (angl. *loader*) pa potrebuje informacijo o lokaciji prevedene knjižnice SystemC, ki jo ponavadi navedemo kot spremenljivko okolja z imenom `SYSTEMC`. Ko so kreirane objektne datoteke, jih povezovalnik poveže z objektnimi datotekami knjižnice SystemC (in ostalimi, kot so npr. standardne knjižnice s predlogami – STL, angl. *standard template library*). Rezultat je zagonska datoteka `.exe`, ki vsebuje simulacijsko jedro SystemC in funkcionalni model sistema. Tak model sistema se uporablja kot izvršljiva virtualna platforma, ki tipično opisuje strojno opremo. S tem modelom na nivoju transakcij lahko hitreje razvijamo in testiramo programsko opremo, funkcijsko in arhitekturno izpopolnjujemo model itd.

### 2.2.2 Koncept razredov za strojno opremo v SystemC

SystemC vsebuje mehanizme, potrebne za modeliranje strojne opreme, obenem pa uporablja jezikovno okolje, ki je namenjeno tudi razvoju programske opreme. Ponuja več strojno orientiranih konstruktov, ki ponavadi niso na voljo v jezikih za razvoj programske opreme. Vsi konstrukti so implementirani v okviru C++. Pomembnejše strojno orientirane funkcije, vgrajene v SystemC, so:

- časovni model,
- podatkovni tipi za strojno opremo,
- hierarhija modulov za prilagajanje struktur in povezljivosti,
- upravljanje komunikacij med paralelno izvršljivimi enotami in
- podpora paralelnemu izvajanju.

Eden izmed izzivov pri razvoju jezika za načrtovanje na nivoju sistema je podpora za širok nabor računskih modelov, nivojev abstrakcije in načrtovalskih metodologij. V ta namen je v SystemC vgrajeno majhno jedro, ki predstavlja osnovo za splošno namensko modeliranje, na katero se dodajo računski modeli, načrtovalske knjižnice in koraki načrtovanja, ki se zahtevajo pri specifični metodologiji. Majhno, splošnonamensko jedro jezika SystemC je označeno z izrazom jedro jezika (angl.*core language*) in predstavlja osrednji del standarda jezika SystemC.

#### 2.2.2.1 Časovni model

SystemC beleži čas s 64 bitno resolucijo z uporabo razreda `sc_time`. Ponuja mehanizme za pridobitev trenutnega časa in implementacijo specifičnih časovnih zamikov. Za modele, ki zahtevajo uro, je poskrbljeno z razredom `sc_clock`.

Za razliko od programske opreme je pri obravnavi strojne opreme čas pomemben. Pri programski opremi se je zavedanje časa skrčilo na število instrukcij, ki so potrebne za izvedbo določene funkcije. Podpora za realni čas v SystemC ne nudi neposredne obravnave časa, ampak le preko prioritete izvajanja opravil. To lahko razumemo kot nekakšno statistično obravnavo odziva sistema. Med simulacijo modela, zgrajenega v jeziku SystemC, se čas obravnava kot celoštevilska vrednost. Na podlagi tega časa lahko pridemo do sledi dogodkov v sistemu ter tekstovnih in slikovnih izpisov. Pravilno modeliranje signalov strojne opreme je izvedeno preko mehanizma »delta« korakov, ki ga lahko razumemo kot zelo majhno povečanje časa znotraj simulacije. Ta pa ne vpliva na povečanje simulacijskega časa, ki je viden od zunaj. V določenem časovnem koraku se tako lahko zgodi več »delta« korakov. Ko

se signalu dodeli določena vrednost, ostali procesi te vrednosti ne vidijo, dokler ne nastopi nov »delta« korak. Procesi, občutljivi na spremenjen signal, lahko v tem primeru nadaljujejo z izvajanjem. Prednost, ki jo ima jezik SystemC v primerjavi z ostalimi jeziki za opis strojne opreme, ki delujejo na podobnem principu simulacije, je hitrost izvajanja, saj je količina informacij na višjih nivojih abstrakcije lahko znatno manjša.

Globalni model časa je resda vezan na celoštevilski model, si pa načrtovalec lahko zgradi določene kanale, ki natančno odražajo pravila komunikacije med procesi, aktiviranja procesov in zaporedja dogodkov v celotnem sistemu. Modeli zveznega časa, ki se na primer uporabljajo v modeliranju analognih sistemov, trenutno še niso podprti znotraj simulacijskega jedra jezika SystemC. Podprto pa je modeliranje praktično katerega koli modela diskretnega časa.

#### 2.2.2.2 Podatkovni tipi za strojno opremo

Širokih spektrov podatkovnih tipov, ki jih zahtevajo digitalna vezja, ni zagotovljenih v samem C++, kjer so tipično 8-, 16-, 32- in 64-bitni. SystemC pa take podatkovne tipe vsebuje. Podpirajo tudi podatkovne tipe z izrecno določeno bitno širino in nebinarne podatkovne tipe, kot na primer logiko treh stanj ali nedoločena stanja. Strojna oprema pa ni vedno digitalna. SystemC direktno ne podpira načrtovanja analogne strojne opreme, vendar je to izvedljivo, če modeliramo analogne vrednosti s plavajočo vejico.

#### 2.2.2.3 Hierarhija in strukture

Veliki sistemi so skoraj vedno hierarhično razdeljeni na manjše dele zaradi lažjega obvladovanja in razumevanja. SystemC ponuja več konstruktov za implementacijo hierarhije. Modeli strojne opreme so največkrat sestavljeni iz blokov, povezanih z žicami ali s signali. Za modeliranje hierarhične strojne opreme SystemC uporablja module, povezane z drugimi moduli preko kanalov. Podobno kot pri jezikih za opis strojne opreme jezik SystemC omogoča strukturirano zgrajen opis sistema z uporabo modulov, priključkov in signalov. Module je mogoče uporabiti znotraj drugih modulov, s čimer je omogočen hierarhičen opis.

#### 2.2.2.4 Upravljanje komunikacij

Kanal v SystemC ponuja močan mehanizem za modeliranje komunikacij. Konceptualno je kanal (angl. *channel*) več kot le preprost signal ali žica. Kanali lahko predstavljajo kompleksne komunikacijske sheme, ki tvorijo pomembno strojno opremo, kot so lahko vodila, lahko pa predstavljajo zelo preproste komunikacije, kot na primer žico ali FIFO sklad. Zmožnost, da lahko za povezovanje modulov uporabljamo precej različne implementacije

kanalov, je zelo pomembna funkcija. Tako lahko preprosto implementacijo vodila nadomestimo z bolj detajlno strojno izvedbo, ki jo na koncu implementiramo z vrati. SystemC ima vgrajene kanale, ki se pogosto uporabljajo pri izdelavi strojne in programske opreme. Mogoče je na primer modeliranje signalov strojne opreme, skladov (FIFO, LIFO), semaforjev, spominov, vodil itd. Kanali so objekti, ki služijo za komunikacijo in sinhronizacijo. Kanali implementirajo enega ali več vmesnikov (angl. *interfaces*).

Priključki in signali omogočajo izmenjavo podatkov med moduli, oboje pa si lahko načrtovalec prilagodi po meri.

Dogodek je prilagodljiv, nizkonivojski sinhronizacijski mehanizem, ki predstavlja temelje za gradnjo ostalih oblik sinhronizacije. Z uporabo kanalov, vmesnikov in dogodkov lahko načrtovalec modelira široko področje komunikacijskih in sinhronizacijskih mehanizmov, uporabljenih v načrtovanju sistema.

#### 2.2.2.5 Sočasnost

V simulatorju je sočasnost le navidezna. Simulatorji izvršijo kodo na enem samem procesorju. Tudi če bi imeli več procesorjev, ki bi izvajali simulacijo, bi število enot, ki se izvajajo sočasno, v realni strojni opremi vedno presegalo število procesorjev, uporabljenih pri simulaciji za nekaj velikostnih razredov.

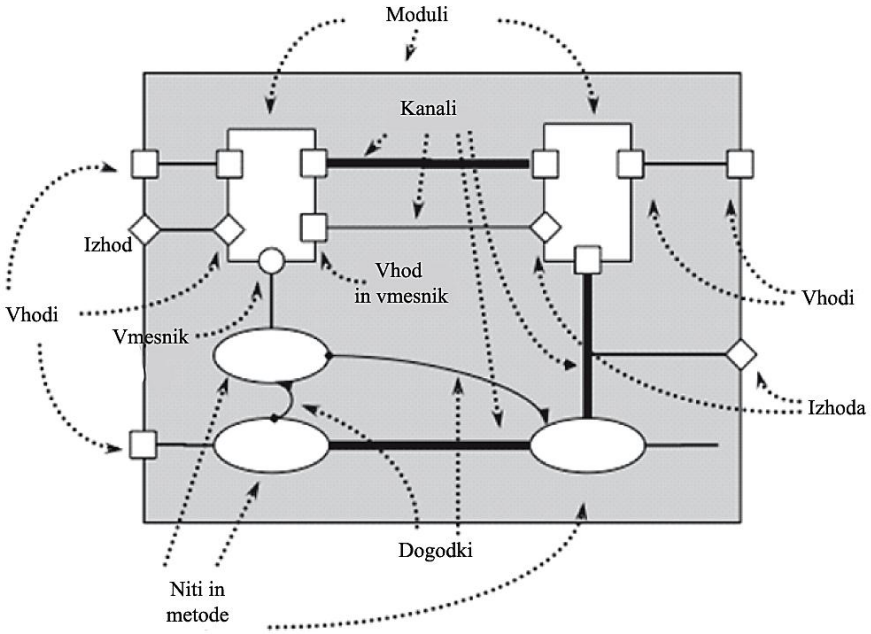
Simulacija sočasne izvedbe je dosežena s simuliranjem vsake sočasne enote. Vsaka enota lahko nadaljuje z izvajanjem, ko se spremenijo določeni signali. Izvajanje enote se začasno ustavi, ko se določena opravila zaključijo. Sama simulacijska koda določa, kdaj simulator preklopi med enotami. Tak način sočasnega izvajanja je enak pri SystemC, Verilog, VHDL ali katerem drugem jeziku za opis strojne opreme. Z drugimi besedami: simulator uporablja kooperativni večopravilni model in le priskrbi jedro, ki upravlja preklope med različnimi sočasnimi elementi, ki se imenujejo simulacijski procesi. Ker se procesi izvajajo sočasno in uporabnik njihovo izvajanje lahko prekine in nadaljuje kadar koli (v SystemC je tak tip označen kot `SC_THREAD`), zahteva uporaba procesov svoj neodvisen sklad izvajanja. Določeni tipi procesov, katerih izvajanje se prekine v določenih točkah, pa za svoje izvajanje ne potrebujejo neodvisnega sklada (v SystemC je tak tip označen kot `SC_METOD`). Uporaba tega drugega tipa procesov izredno izboljša učinkovitost simulacijskega jedra.

### 2.2.3 Pregled komponent SystemC

Slika 8 povzema najpomembnejše lastnosti arhitekture jezika SystemC. Najnovejša koda je prosto dostopna na strani OSCI [8], kjer je na voljo tudi ažurna spletna podpora za tehnične težave v obliki foruma. Želel bi še dodati, da v povezavi z jezikom SystemC poteka zelo veliko aktivnosti, tako akademske kot tudi industrijske narave. Zelo pomembno nadgradnjo predstavlja verifikacijska knjižnica SystemC (SVC, angl. *SystemC verification library*), ki omogoča združevanje jezika SystemC z mehanizmi, ki omogočajo učinkovito verifikacijo sistema.


Slika 8: Arhitektura jezika SystemC [7].

Slika 9 prikazuje koncept modula `sc_module`, ki vsebuje instance drugih `sc_module`, znotraj pa sta lahko definirana simulacijska procesa `SC_METHOD` ali `SC_THREAD`. Komunikacija med moduli in simulacijskimi procesi (`SC_METHOD` in `SC_THREAD`) je dosežena preko različnih kombinacij priključkov, vmesnikov in kanalov. Koordinacija med simulacijskimi procesi pa je zagotovljena preko dogodkov.




Slika 9: Komopnente jezika SystemC [1].

#### 2.2.3.1 Moduli in hierarhija

Strojna oprema je tipično sestavljena hierarhično zaradi zmanjšanja kompleksnosti. Vsak nivo hierarhije predstavlja blok. Pri VHDL so ti bloki entitetni pari, ki ločujejo specifikacijo vmesnika od jedra kode za vsak blok. SystemC podobno kot VHDL ločuje vmesnik in implementacijo. V C++ se »header« datoteka (`.h`) uporablja za entiteto, implementacijska datoteka (`.cpp`) pa za arhitekturo.

Komponente modela so vsebovane v modulih. To so razredi, podedovani iz osnovnega razreda `sc_module`. Zaradi poenostavitve uporabljamo makro `SC_MODULE`. Moduli lahko vsebujejo druge module, procese in kanale ter priključke za povezljivost.

#### 2.2.3.2 Niti in metode

Simulacijsko jedro SystemC določi izvršitev vseh simulacijskih procesov. Simulacijski procesi so funkcije razreda `sc_module`, ki so registrirane v simulacijskem jedru. Ker simulacijsko jedro kliče te funkcije, le-te ne potrebujejo argumentov in ne vračajo nobene vrednosti. Razred `sc_module` lahko vsebuje procese, ki jih simulacijsko jedro ne izvede. Ti procesi so klicani kot funkcijski klici znotraj simulacijskih procesov.

V elaboracijski fazi (med izvršitvijo konstruktorjev razreda `sc_module`) se procesi registrirajo v simulacijskem jedru z uporabo `SC_METHOD`, `SC_THREAD` ali `SC_CTHREAD`

makrov jezika SystemC. Osnovni tip simulacijskega procesa je `SC_METHOD`, ki je del `sc_module` razreda. Je funkcija C++ brez argumentov, ki ne vrača vrednosti in jo ponavljajoče kliče le simulacijsko jedro. Drugi simulacijski proces je `SC_THREAD`, ki se razlikuje od `SC_METHOD` v dveh stvareh. Prva je, da se `SC_METHOD` zažene večkrat, `SC_THREAD` pa le enkrat. Druga pa, da ima `SC_THREAD` možnost zadržanja izvajanja in ponovnega nadaljevanja po določenem času.

`SC_METHOD` in `SC_THREAD` sta osnovni enoti sočasnega izvajanja. Simulacijsko jedro kliče vsakega od teh procesov, zato ne moreta biti klicana s strani uporabnika. Ta lahko le indirektno nadzira izvajanje simulacijskih procesov z uporabo dogodkov, občutljivosti in obvestil.

#### 2.2.3.3 Dogodki, občutljivost in obvestila

Dogodki (angl. *event*), občutljivost (angl. *senstitivity*) in obvestila (angl. *notification*) so zelo pomembni koncepti za razumevanje implementacije sočasnosti v simulatorju SystemC. Dogodki so implementirani z razredoma SystemC `sc_event` in `sc_event_queue`. Sproženi so preko razreda dogodkov `notify`. Obvestila se pojavijo v simulacijskem procesu kot rezultat neke aktivnosti v kanalu. Simulacijsko jedro kliče `SC_METHOD` in `SC_THREAD`, kadar pride do dogodka, na katerega sta občutljiva.

SystemC ima dva tipa občutljivosti: statično in dinamično. Statična je implementirana z ukazom `sensitive` v `SC_METHOD` ali `SC_THREAD` ob elaboracijskem času (znotraj konstruktorja). Dinamično lahko simulacijski proces spreminja sproti. V `SC_METHOD` je dinamična občutljivost implementirana z ukazom `next_trigger(arg)`, v `SC_THREAD` pa z `wait(arg)`. Tako `SC_METHOD` kot `SC_THREAD` lahko med simulacijo preklapljata med dinamično in statično občutljivostjo.

#### 2.2.3.4 Podatkovni tipi v SystemC

SystemC ima veliko podatkovnih tipov za strojno opremo. Ker pa je zgrajen na C++, so na voljo tudi vsi podatkovni tipi C++. Dovoljuje pa tudi definiranje novih podatkovnih tipov za novo tehnologijo strojne opreme ali aplikacije.

Podatkovna tipa za matematične izračune, kot sta `sc_fixed<T>` in `sc_int<T>`, omogočata modeliranje kompleksnih kalkulacij. SystemC vsebuje vse potrebne metode za uporabo podatkovnih tipov za strojno opremo, vključno za pretvorbo med strojnimi in programskimi podatkovnimi tipi.

Nebinarni podatkovni tipi so podprti z logiko štirih stanj (ta stanja so: 0, 1, X, Z), na primer s tipom `sc_logic`. Podobni tipi, kot `sc_logic` in `sc_lv<T>`, so na voljo za razvijalce vezij RTL, ki potrebujejo predstavitev osnovnih logičnih vrednosti ali vektorjev logičnih vrednosti.

#### 2.2.3.5 Priključki, vmesniki in kanali v SystemC

Procesi med sabo komunicirajo lokalno in globalno med drugimi moduli. V tradicionalnih jezikih za opis strojne opreme procesi komunicirajo preko priključkov (angl. *port*) ali nožic ter signalov ali žic. V okolju SystemC procesi komunicirajo z uporabo kanalov (angl. *channel*). Procesi lahko komunicirajo tudi preko mej modula. Moduli so lahko povezani z uporabo kanalov in priključkov. SystemC uporablja konstrukt `sc_port<T>`, `sc_export<T>` ter osnovna razreda `sc_interface` in `sc_channel` za implementacijo povezljivosti.

SystemC ponuja nekatere standardne kanale in vmesnike (angl. *interface*), izpeljane iz osnovnih tipov. Kanali vključujejo sinhronizacijske primitive `sc_mutex` in `sc_semaphore` ter komunikacijske kanale `sc_fifo<T>`, `sc_signal<T>` in ostale. V teh kanalih so implementirani vmesniki SystemC: `sc_mutex_if`, `sc_semaphore_if`, `sc_fifo_in_if<T>`, `sc_fifo_out_if<T>`, `sc_signal_in_if<T>`, in `sc_signal_inout_if<T>`.

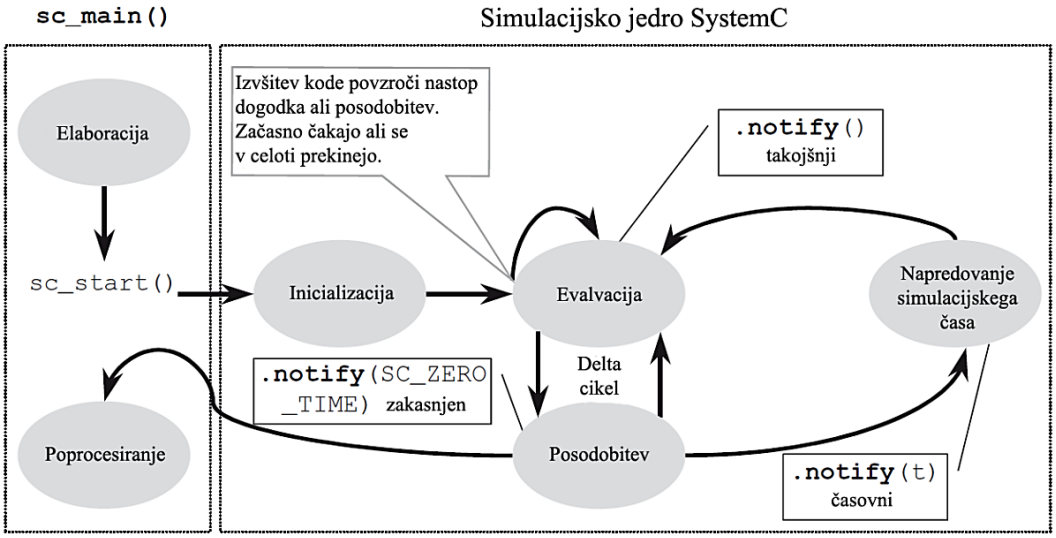
Moduli se medsebojno povežejo programsko med elaboracijo. Tak način povezovanja omogoča gradnjo običajnih struktur z uporabo zank in pogojnih stavkov. Programersko gledano je elaboracija čas, ko moduli kličejo svoje konstruktorske metode.

### 2.2.4 Simulacijsko jedro SystemC

Simulator SystemC ima dve pomembnejši fazi: *elaboracije* (angl. *elaboration*) in *izvršitve* (angl. *execution*). Tretjo, manj pomembno fazo, ki nastopi na koncu izvršitve, imenujemo *poprocesna* (angl. *post-processing*) ali *čistilna* (angl. *cleanup*) faza.

Izvršitve pred klicem funkcije `sc_star()` imenujemo elaboracijska faza. Za to fazo je značilno, da inicializira podatkovne strukture, vzpostavi povezave in pripravi model na drugo fazo, fazo izvršitve.

Izvršitvena faza preda kontrolo simulacijskemu jedru SystemC, ki nadzira izvrševanje procesov zaradi zagotavljanja sočasnosti.




Slika 10: Simulacijsko jedro SystemC [1].

Slika 10 prikazuje potek simulacije in simulacijsko jedro SystemC. Podobna simulacijska jedra imata tudi Verilog in VHDL. Po klicu funkcije `sc_start()`, to je med inicializacijo, se vsi simulacijski procesi (razen nekaj izjem) aktivirajo v nespecifičnem deterministčnem zaporedju.

Po inicializaciji se ob pojavu dogodkov, na katere so občutljivi, zaženejo simulacijski procesi. V simulator SystemC je vgrajeno kooperativno večopravilno okolje. V simulatorskem času se lahko istočasno zažene več simulacijskih procesov. Takrat se vsi procesi evalvirajo in se jim posodobijo izhodi. To imenujemo »delta« cikel. Če pri nobenem simulacijskem procesu ni potrebna evalvacija, se simulacijski čas nadaljuje. Simulacija se konča, ko ni več nobenega simulacijskega procesa, ki bi se moral zagnati.

V simulacijskem jedru je prisotna vsa potrebna funkcionalnost, ki predstavlja osnovo za zelo zmogljiv splošen računski model. Pomembno dejstvo je, da simulacijsko jedro SystemC nudi splošen računski model in tako v tem kontekstu praktično ne postavlja nobenih omejitev. V bodočih različicah jezika SystemC lahko pričakujemo veliko razširitev, na primer podporo za mehanizme, običajno prisotne v operacijskih sistemih, kot tudi modeliranje analognih sistemov. Že sedaj pa je modeliranje takih sistemov mogoče preko razširitev simulacijskega jedra.

## 2.3 Zgodovina

SystemC ima svoje korenine v programskem jeziku Scenic, ki so ga razvili pri Synopsys leta 1997. Scenic je načrtovalcem dovoljeval uporabo jezika C++ za razvoj strojne opreme s prevajalnikom C++ in majhno knjižnico Scenic, brez potreb po kompleksnih jezikih za opis strojne opreme.

Leta 1999 so podjetja ARM Ltd., CoWare, Synopsys in SynApps združeno izdala prvo verzijo SystemC. Takratna glavna konkurenca je bil prav tako odprtokoden jezik SpecC, razvit na kalifornijski univerzi. Ustanovljena je bila tudi organizacija Open SystemC Initiative (OSCI), in sicer zaradi zahtev po standardnem jeziku za oblikovanje sistemov na čipu (SoC, angl. *system-on-a-chip*) z uporabo C/C++. OSCI je neodvisna in neprofitna organizacija za razvoj SystemC kot odprtokodnega standarda za modeliranje na nivoju sistema, načrtovanje in verifikacijo.

Glavne naloge organizacije OSCI so:

- razvijanje jezika za opisovanje sistemov in njegovo odprtokodno implementacijo na osnovi knjižnice C++, imenovane SystemC,
- spodbujanje dostopnosti in sprejemanje intelektualne lastnine, orodij in metodologij, ki temeljijo na SystemC,
- zagotavljanje mehanizmov za nadaljnjo rast skupnosti SystemC,
- definiranje interoperabilnostnih meril za intelektualno lastnino in orodij, temelječih na SystemC,
- dostavljanje nadgradnje SystemC LRM in odprtokodno implementacijo,
- standardizacija jezika SystemC pri IEEE.

Člani OSCI so organizacije, ki se ukvarjajo z elektroniko. To so podjetja, ki izdelujejo sisteme na integriranih vezjih, prodajajo orodija, dobavljajo intelektualno lastnino in razvijajo vgrajene sisteme. Korporativni člani OSCI so ARM, Cadence Design Systems, Forte Design Systems, Intel, Mentor Graphics, NXP, ST Microelectronics, ST Ericsson in Synopsys. Izredni korporativni člani so Atrenta, Carbon Design Systems, CircuitSutra Technologies, Doulos, Fraunhofer Institute for Integrated Circuits, Kasura Technlologies, NEC, Offis Institue, Qualcomm, Texas Instruments in UPMC.

Razvoja jezika SystemC je potekal v sledečih fazah [1, 8]:

- september 1999 0.9 prva verzija; ciklični simulator,
- februar 2000 0.91 popravki,
- marec 2000 1.0 širše uporabna verzija,
- oktober 2000 1.0.1 popravki,
- februar 2001 1.2 različne izboljšave,
- februar 2002 2.0 dodani kanali in dogodki; preglednejša sintaksa,
- april 2002 2.0.1 popravki; široka uporaba,
- junij 2003 2.0.1 LRM,
- pomlad 2004 2.1 vložen LRM za IEEE standard,
- december 2005 2.1v1 ratificiran standard IEEE 1666-2005,
- julij 2006 2.2 popravki za boljšo implementacijo standarda.

# 3 Postavljanje modelov v okolju SystemC

V tem poglavju je predstavljena vzpostavitev okolja SystemC na sistemu Microsoft Windows in izgradnja vzorčnega primera modela, na katerem je demonstrirana uporaba elementov jezika SystemC, ki so pomembni pri modeliranju strojne opreme.

## 3.1 Vzpostavitev razvojnega okolja

Modeliral sem v operacijskem sistemu Microsoft Windows 7 in Microsoft Visual Studio 2005. Po namestitvi Visual Studia je potrebna registriracija na spletni strani OSCI [8], za prenos zadnje različice SystemC 2.2. Mapa »msvc71« vsebuje datoteke projekta in konfiguracijske datoteke za delovni prostor, s pomočjo katerega prevedemo knjižnico »systemc.lib«. To naredimo z zagonom projekta »SystemC.vcproj«. Zažene se Visual Studio s pravilno nastavljenimi stikali za prevajanje. Izberemo »Build SystemC« (F7) za izgradnjo »systemc.lib«. Prevajalnik in povezovalnik morata poznati lokacijo »header« datoteke in knjižnic datotek SystemC. To naredimo s spremenljivko okolja »SYSTEMC«, ki kaže na pot namestitve SystemC. Uspešno namestitev lahko preverimo s testnimi primeri, ki se nahajajo v mapi »examples«.

Podrobnejša navodila in sistemske zahteve za namestitev SystemC so napisana v datoteki »INSTALL«.

## 3.2 Izgradnja vzorčnega primera kode

V tem razdelku je na primeru demonstrirana uporaba modulov in procesov v okolju SystemC. Zaradi preprostosti je uporabljena nizkonivjska koda, kakršne pri jeziku za modeliranje na nivoju sistema ponavadi ne srečamo.

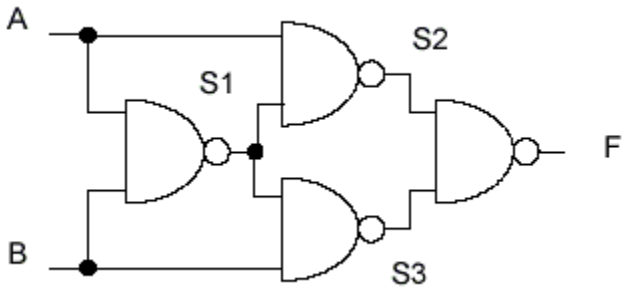
Predstavljeni so naslednji elementi okolja SystemC:

- kreiranje hierarhije,
- uporaba primitivnega kanala *sc_signal*,
- uporaba specializiranih priključkov,
- uporaba procesov,
- preprost predstavitveni test.

Procesi so manjši deli kode, ki tečejo sočasno z drugimi procesi. Vsa orodja za visoko nivojsko modeliranje na nivoju sistema uporabljajo mrežo procesov. SystemC zagotavlja podporo za gradnjo omrežij neodvisnih, paralelnih procesov. Ker pa se pri modeliranju na nivoju sistema srečujemo z velikimi modeli, potrebujemo hierarhijo. Hierarhija je v SystemC implementirana z uporabo modula. Modul je razred, ki se ga lahko poveže z drugimi moduli z uporabo priključkov. Z uporabo modulov lahko posamezne dele modela razvijamo ločeno. Moduli lahko vsebujejo procese in instance drugih modulov.

### 3.2.1 Primer modela vrat XOR z uporabo štirih vrat NAND

Model vrat XOR implementirmo z uporabo štirih vrat NAND, kot prikazuje slika 11.




Slika 11: Vrata XOR, realizirana s štirimi vrati NAND [4].

V prvem delu zmodeliramo vrata NAND. Vrata NAND so kombinatorno vezje, njihov izhod je povsem odvisen od vrednosti na vhodu. Nimajo spomina in ne potrebujejo ure. Zaradi tega lahko za model uporabimo enostavnejši proces SystemC `SC_METHOD`. `SC_METHOD` so preproste funkcije C++. Zanje knjižnice SystemC poskrbijo, da se obnašajo kot procesi in sicer:

- Simulacijsko jedro SystemC modelira potekanje časa in kliče funkcije, da izračuna njihove izhode ob spremembi vhodov.
- Funkcija mora biti deklarirana kot `SC_METHOD` in občutljiva na svoje vhode.

Koda za vrata NAND v datoteki `nand.h` je sledeča:

```
#include "systemc.h"

SC_MODULE(nand2)           // deklariranje nand2 sc_module
{
     sc_in<bool> A, B;          // vhodni signali priključkov
     sc_out<bool> F;            // izhodni signali priključkov

     void do_nand2()            // funkcija C++
     {
          F.write( !(A.read() && B.read()) ); // izračun vrednosti
     }

     SC_CTOR(nand2)             // konstruktor za nand2
     {
          SC_METHOD(do_nand2);  // registracija do_nand2 z jedrom
          sensitive << A << B;  // občutljivostni seznam
     }
};
```

Hierarhija v SystemC je kreirana z uporabo razreda `sc_module`, ki je lahko uporabljen direktno, ali pa skrit z uporabo makra `SC_MODULE`. Zgornji primer `SC_MODULE` kreira objekt razreda `sc_module`, imenovan `nand2`.

Nato deklariramo vhodne in izhodne priključke. V splošnem je priključek deklariran z uporabo razreda `sc_port`. Vhodni priključki bi bili z uporabo `sc_signal` deklarirani takole:

```
sc_port<sc_signal_in_if<bool>,1> A,B;
```

Lahko pa uporabimo specializirane priključke razreda `sc_signal`, kot na primer `sc_in`. Priključki so lahko tipa C++ ali SystemC. V primeru je uporabljen v C++ vgrajen tip `bool`.

Naslednja je deklarirana funkcija C++, ki izračuna vrednost NAND. Vhodni in izhodni specializirani priključki vsebujejo metode `read()` in `write()` za branje in pisanje na priključek. Preberemo `A` in `B`, izračunamo logično vrednost NAND in jo z uporabo metode `write()` zapišemo v `F`.

Namesto metod `read()` in `write()` bi lahko uporabili operator »=« :

```
F = !(A && B);
```

Za funkcijo `do_nand2` sledi konstruktor za `sc_module` instance `nand2`. To je makro `SC_CTOR`, ki kreira hierarhijo, v simulacijskem jedru registrira funkcije kot procese in deklarira občutljivostni seznam za procese. V našem primeru konstruktor deklarira, da je funkcija `do_nand2` simulacijski proces tipa `SC_METHOD` in da mora jedro po kakršnem koli dogodku na priključkih `A` in `B` znova zagnati funkcijo (znova izračunati vrednost `F`).

### 3.2.2 Hierarhija

Vrata XOR so zgrajena iz štirih kopij (instanc) vrat NAND. To dosežemo z uporabo konstruktorja vrat XOR, kjer povežemo instance vrat NAND. V nadaljevanju je navedena koda za vrata XOR v datoteki `xor2.h`:

```
#include "systemc.h"
#include "nand2.h"        // instanca vrat NAND

SC_MODULE(xor2)
{
    sc_in<bool> A, B;
    sc_out<bool> F;

    nand2 n1, n2, n3, n4; // deklariranje štirih instanc vrat NAND

    sc_signal<bool> S1, S2, S3;//deklariranje instanc signalov
    // oznake instanc nand2, ki jih konstruktor xor2 potrebuje
    SC_CTOR(xor2) : n1("N1"), n2("N2"), n3("N3"), n4("N4")
    {
        n1.A(A);          // povezova priključkov na štiri načine
        n1.B(B);
        n1.F(S1);

        n2 << A << S1 << S2;

        n3(S1);
        n3(B);
        n3(S3);

        n4.A(S2);
        n4.B(S3);
        n4.F(F);
    }
};
```

Začetek je zelo podoben kodi vrat NAND, s to razliko, da ta vsebuje še datoteko `nand2.h`. To omogoča dostop do modula, ki vsebuje vrata NAND. Nato kreiramo modul `xor2` in deklariramo priključke. Dovoljena je ponovna uporaba imen priključkov `A`, `B` in `F`, saj smo sedaj na drugem nivoju hierarhije.

Na sliki 11 vidimo žice, ki povezujejo vrata NAND. Te ustvarimo z deklaracijo `sc_signal S2`, `S2` in `S3`. Razred `sc_signal` vsebuje osnovne parametre, ki določajo tip podatka, ki ga signal lahko vsebuje. V našem primeru lahko vsebuje logično vrednost (`bool`). To je primer primitivnega signala, vgrajenega v knjižnico SystemC, ki se obnaša kot signal v VHDL.

Konstruktor za vrata XOR je bolj kompleksen, saj mora vsebovati instance `nand2`. Po deklaraciji priključkov deklariramo štiri instance `nand2`: `n1`, `n2`, `n3` in `n4` ter vsaki dodamo oznako. Te štiri oznake `"N1"`, `"N2"`, `"N3"` in `"N4"` se prenesejo konstruktorjem instanc `nand2` z uporabo inicializacijske liste na konstruktorju `xor2`.

Nazadnje v konstruktorju povežemo priključke z uporabo oklepajev ali operatorja <<. Z uporabo operatorja << lahko priključke povežemo po pozicijah, kot je primer za instanco `n2`, z uporabo oklepajev pa po pozicijah (`n3`) in imenih (`n1` in `n4`).

### 3.2.3 Meritveni test

Za testiranje modela potrebujemo »generator signalov«. To je modul, podoben zgornjim, le da ta uporablja nit (`SC_THREAD`), neke vrste proces, ki se lahko prekine. Zadnji klic `sc_stop()` zaustavi simulacijo. Koda modula v datoteki `stim.h` bi bila sledeča:

```
#include "systemc.h"

SC_MODULE(stim)
{
     sc_out<bool> A, B;
     sc_in<bool> Clk;

     void StimGen()         // funkcija C++
     {
           wait();
           A.write(false);
           B.write(false);
           wait();
           A.write(false);
           B.write(true);
           wait();
```

```
            A.write(true);
            B.write(false);
            wait();
            A.write(true);
            B.write(true);
            wait();

            sc_stop();       // prekinitev simulacije
      }

      SC_CTOR(stim)
      {
            SC_THREAD(StimGen);
            sensitive << Clk.pos();
      }
};
```

Koda »monitorja« izpisuje vrednosti `A`, `B` in `F` v določenem času. Koda v `mon.h` je sledeča:

```
#include "systemc.h"

SC_MODULE(mon)
{
      sc_in<bool> A, B, F;
      sc_in_clk Clk;

      void Monitor()
      {
            cout << "\n" << "Time\tA B F" << endl;
            while(1)
            {
                  wait();
                  cout << sc_time_stamp() << "\t" << A.read()
                  << " " << B.read() << " " << F.read() << endl;
            }
      }

      SC_CTOR(mon)
      {
            SC_THREAD(Monitor);
            sensitive << Clk.pos();
      }
};
```

Najvišji nivo je v datoteki `main.cpp` in vsebuje vse podmodele, opisane zgoraj. Koda v `main.cpp` je sledeča:

```
#include "systemc.h"
#include "stim.h"
#include "xor2.h"
#include "mon.h"

int sc_main(int argc, char* argv[])
{
    sc_signal<bool> ASig, BSig, FSig;
    sc_clock TestClk("TestClock", 10, SC_NS,0.5);

    stim Stim1("Stimulus");
    Stim1.A(ASig);
    Stim1.B(BSig);
    Stim1.Clk(TestClk);

    xor2 DUT("xor2");
    DUT.A(ASig);
    DUT.B(BSig);
    DUT.F(FSig);

    mon Monitor1("Monitor");
    Monitor1.A(ASig);
    Monitor1.B(BSig);
    Monitor1.F(FSig);
    Monitor1.Clk(TestClk);

    sc_start();     // zažene neprekinjeno simulacijo

    system("pause");

    return 0;
}
```

Vključimo vse header datoteke modulov, deklariramo signale na najvišjem nivoju, kreiramo uro `sc_clock` in zgradimo instance modulov ter jih povežemo. Nato klic `sc_start()` zažene neprekinjeno simulacijo oziroma se le-ta prekine ob pojavu klica `sc_stop()` v modulu `stim`. Dobimo naslednji izpis primera:

```
Time        A B F
0 s         0 0 1
10 ns       0 0 0
20 ns       0 1 1
30 ns       1 0 1
40 ns       1 1 0
```

Če pogledamo izpis, vidimo, da je v prvi vrstici ob času 0 sekund `F` enak 1 (*true*), medtem ko sta `A` in `B` enaka 0 (*false*), kar za vrata XOR ne velja. Kaj se dogaja ob času 0 sekund, je prikazano v naslednjem razdelku.

### 3.2.4 Simulacija

Jezik SystemC vsebuje simulacijsko jedro, ki odloča, kateri proces (programsko nit) bo izvrševal. Ob času 0 sekund se zaženejo vse `SC_METHOD` in `SC_THREAD` v nedefiniranem zaporedju, dokler se začasno ne ustavijo. Ko se pojavi fronta ure, pa simulacijski procesi tipa `SC_THREAD` tečejo.

Problem nastane zaradi kombinacije naslednjih okoliščin:

- Stavek `sc_clock` povzroči prehod ure ob času 0 sekund, kar pozroči, da se zaženeta monitor in generator signalov v nedefiniranem zaporedju (ne vemo, kateri se bo zagnal prvi).
- Spremenljivke v C++ nimajo vedno definirane začetne vrednosti. Začetna vrednost spremenljivke F je tako 1.
- Ob času 0 sekund se zažene `do_nand2 SC_METHOD` in določi `F` za posodobitev. Toda signal `F` se ne more v trenutku posodobiti, zato je ob zagonu »monitorja« vrednost 1 še vedno prisotna.

To lahko popravimo s spremembo navedbe `sc_clock` tako, da urino fronto premaknemo za 1 ns. Argument `1,SC_NS` določa zamudo za 1 nanosekundo, preden se pojavi prva fronta ure in tako preteče čas, da se `F` posodobi. Nova deklaracija ure je sledeča:

```
sc_clock TestClk("TestClock", 10, SC_NS,0.5, 1, SC_NS);
```

Dobimo naslednji izpis, kjer je `F` (izhod) vedno pravilen:

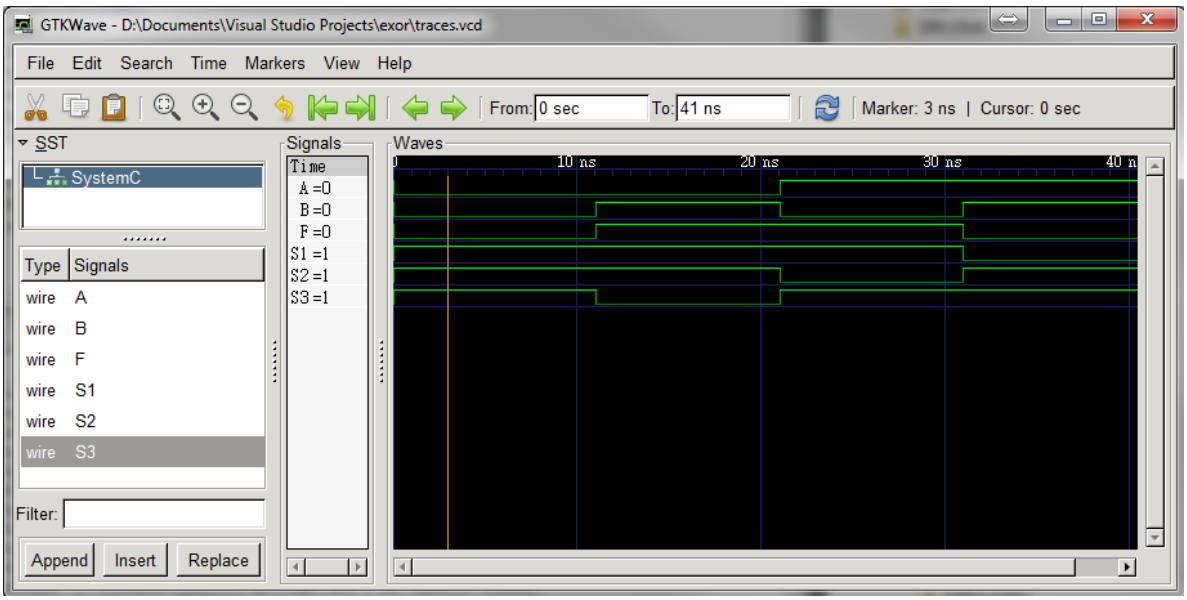
```
Time       A B F
1 ns       0 0 0
11 ns      0 0 0
21 ns      0 1 1
31 ns      1 0 1
41 ns      1 1 0
```

### 3.2.5 Sledenje modelu v valovni obliki

SystemC podpira sledenje v valovni obliki s pomočjo dodatnih ukazov v `sc_main`:

```
    ...                              // main.cpp
    sc_trace_file* Tf;               // deklariranje datoteke trace
    Tf = sc_create_vcd_trace_file("traces");// kreiranje datoteke
    ((vcd_trace_file*)Tf)->sc_set_vcd_time_unit(-9); //enota časa
    sc_trace(Tf, ASig  , "A" );      // registriranje signalov ali
    sc_trace(Tf, BSig  , "B" );      // spremenljivk, ki jim želimo
    sc_trace(Tf, FSig  , "F" );      // slediti
    sc_trace(Tf, DUT.S1, "S1");
    sc_trace(Tf, DUT.S2, "S2");
    sc_trace(Tf, DUT.S3, "S3");

    sc_start();                      // zagon simulacije

    sc_close_vcd_trace_file(Tf);     // zapiranje datoteke trace

    return 0;
}
```

Sledimo lahko celotni hierarhiji, na primer vmesnim signalom `DUT.S1`. Dobimo datoteko s končnico *.vcd (angl. *value change dump*), ki jo odpremo s pomočjo brezplačnega programa GTKWave Analyzer [5]. Rezultat prikaza je viden na sliki 12.




Slika 12: Prikaz delovanja modela v programu GTKWave Analyzer.

# 4 Realizacija 4-bitnega seštevalnika

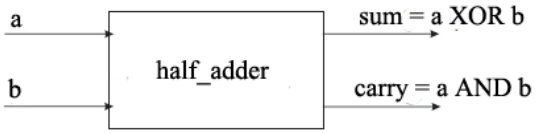
V okolju SystemC sem se odločil realizirati model 4-bitnega seštevalnika. Želel sem, da bi bil model višjega nivoja in ne na nivoju vrat, kot je vzorčni primer iz tretjega poglavja.

## 4.1 Seštevalnik

Seštevalnik je modeliran hierarhično. Osnovna najnižja enota v hierarhiji je enobitni polovični seštevalnik, ki sešteje dva bita in določi prenos. Dva polovična seštevalnika sta vezana v polni seštevalnik, ki predstavlja naslednjo stopnjo hierarhije. Polni seštevalnik vsebuje dodaten vhod za vhodni prenos. Štiri instance polnega seštevalnika, vezane zaporedno, pa tvorijo 4-bitni seštevalnik.

### 4.1.1 Polovični seštevalnik

Osnovni enobitni seštevalnik na sliki 13 je polovični seštevalnik (angl. *half adder*). Sešteje dva bita a in b ter vrne vsoto in prenos.




Slika 13: Polovični seštevalnik.

V okolju System C ga realiziramo v datoteki `half_adder.h`:

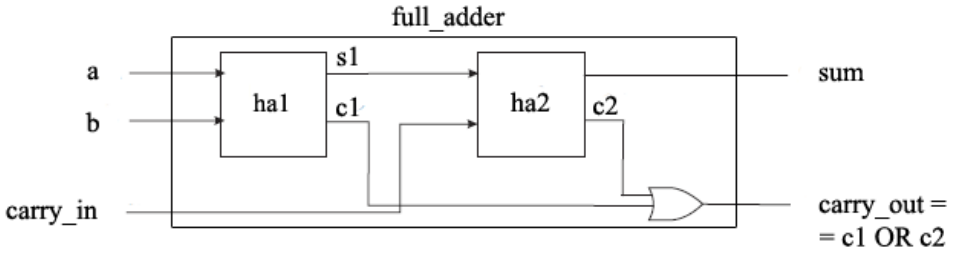
```
#include "systemc.h"

SC_MODULE(half_adder)     // modul polovičnega seštevalnika
{
    sc_in<sc_logic> a, b;           // vhoda tipa sc_logic
    sc_out<sc_logic> sum, carry;    // izhoda
```

```
    void do_half_adder()              // funkcija seštevanja
    {
        sum = a ^ b;                  // izračun sum = a XOR b
        carry = a & b;                // izračun carry = a AND b
    }

    SC_CTOR(half_adder)               // konstruktor za half_adder
    {
        SC_METHOD (do_half_adder);
        sensitive << a << b;          // občutljivost na vhoda a, b
    }
};
```

`SC_MODULE` kreira polovični seštevalnik, imenovan `half_adder`. Vhoda, tipa `sc_logic`, sta `a` in `b`, ki ju funkcija `do_half_adder()` sešteje. Izhoda pa sta vsota `sum` in prenos `carry`.

### 4.1.2 Polni seštevalnik

Enobitni seštevalnik, ki ga lahko uporabimo pri tvorbi večbitnega seštevalnika, je polni seštevalnik (angl. *full adder*). Ima tri enakovredne vhode, od katerih lahko enega uporabimo za vhodni prenos `carry_in` iz prejšnje stopnje.




Slika 14: Polni seštevalnik.

Realizacija polnega seštevalnika v datoteki `full_adder.h` je sledeča:

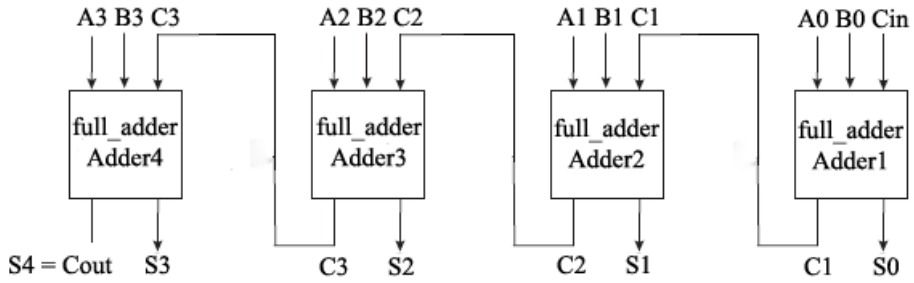
```
#include "systemc.h"
#include "half_adder.h"

SC_MODULE(full_adder)     // modul polnega seštevalnika
{
    sc_in<sc_logic> a, b, carry_in;       // vhodi
    sc_out<sc_logic> sum, carry_out;      // izhodi
    sc_signal<sc_logic> s1, c1, c2;       // signali
```

```
    void do_or()     // funkcija za izračun carry_out = c1 OR c2
    {
         carry_out = c1 | c2;
    }

    half_adder ha1, ha2;  // deklariranje dveh instand half_adder

    SC_CTOR(full_adder) : ha1("ha1"), ha2("ha2")
    {
         ha1.a(a);        // povezava priključkov na ha1
         ha1.b(b);
         ha1.sum(s1);
         ha1.carry(c1);

         ha2 << s1 << carry_in << sum << c2; // in h2

         SC_METHOD (do_or);
         sensitive << c1 << c2;
    }
};
```

Sama vezava dveh polovičnih seštevalnikov v polni seštevalnik je prikazana na sliki 14. V SystemC to naredimo z deklariranjem dveh instanc polovičnega seštevalnika, `ha1` in `ha2`, ter jima povežemo priključke.

### 4.1.3 4-bitni seštevalnik

S kaskadno vezavo enobitnih polnih seštevalnikov sestavimo večbitni seštevalnik. Na sliki 15 je prikazan 4-bitni seštevalnik, sestavljen iz štirih polnih seštevalnikov.




Slika 15: 4-bitni seštevalnik.

Deklaracija štirih instanc polnega seštevalnika in povezava v datoteki `main.cpp` je sledeča:

```
    full_adder Adder1("Adder1"); // instanca polnega seštevalnika
         Adder1 << A0 << B0 << Cin << S0 << C1; // in povezave
```

```
    full_adder Adder2("Adder2");
         Adder2 << A1 << B1 << C1 << S1 << C2;
    full_adder Adder3("Adder3");
         Adder3 << A2 << B2 << C2 << S2 << C3;
    full_adder Adder4("Adder4");
         Adder4 << A3 << B3 << C3 << S3 << Cout;
```

Za 4-bitni seštevalnik potrebujemo 4 instance enobitnih polnih seštevalnikov. Te povežemo, kot kaže slika 15. Vhodna 4-bitna vektorja A [A3, A1, A2, A0] in B [B3, B2, B, B0] razčlenimo na osem vhodnih enobitnih vhodov za posamezen polni seštevalnik. Bite vsote pa na koncu sestavimo v vektor SUM [Cout, S3, S2, S1, S0]. To naredimo z moduloma vector_in in vector_out. Koda za pretvorbo se nahaja v datoteki vector.h, ki je sledeča:

```
#include "systemc.h"

SC_MODULE(vector_in) // razčlemba vhodnega 4-bitnega vektorja
{
    sc_in<sc_lv<4> > I;
    sc_out<sc_logic> i3, i2, i1, i0;

    void do_vector()
    {
         sc_lv<4> temp = I.read();

         i3 = temp[3];
         i2 = temp[2];
         i1 = temp[1];
         i0 = temp[0];
    }

    SC_CTOR(vector_in)
    {
         SC_METHOD(do_vector);
         sensitive << I;
    }
};

SC_MODULE(vector_out) // sestavljanje bitov v vektor
{
    sc_in<sc_logic> o4, o3, o2, o1, o0;
    sc_out<sc_lv<5> > O;

    void do_vector()
    {
         sc_lv<5> temp;

         temp[4] = o4;
         temp[3] = o3;
         temp[2] = o2;
```

```
            temp[1] = o1;
            temp[0] = o0;

            O = temp;
      }

      SC_CTOR(vector_out)
      {
            SC_METHOD(do_vector);
            sensitive << o4 << o3 << o2 << o1 << o0 << O;
      }
};
```

## 4.2 Simulacija

Simulacijo izvedemo s pomočjo generatorja signalov `stim.h`, katerega koda je sledeča:

```
#include "systemc.h"

SC_MODULE(stim)
{
      sc_out<sc_lv<4> > A, B; // izhodna vektorja
      sc_out<sc_logic> Cin;
      sc_in<bool> Clk;
      char A_Ch [4], B_Ch [4];

      void StimGen()
      {
            cout << "\nVpisite dve 4 bitni binarni stevili:\n";
            cin >> A_Ch;
            A = A_Ch;
            cin >> B_Ch;
            B = B_Ch;
            Cin.write(SC_LOGIC_0); // vrenost 0 - false
            wait();

            A = "1000";
            B = "1001";
            Cin.write(SC_LOGIC_0);
            wait();
            A = "1111";
            B = "1001";
            Cin.write(SC_LOGIC_0);
            wait();
            A = "1000";
            B = "1011";
            Cin.write(SC_LOGIC_0);
            wait();
```

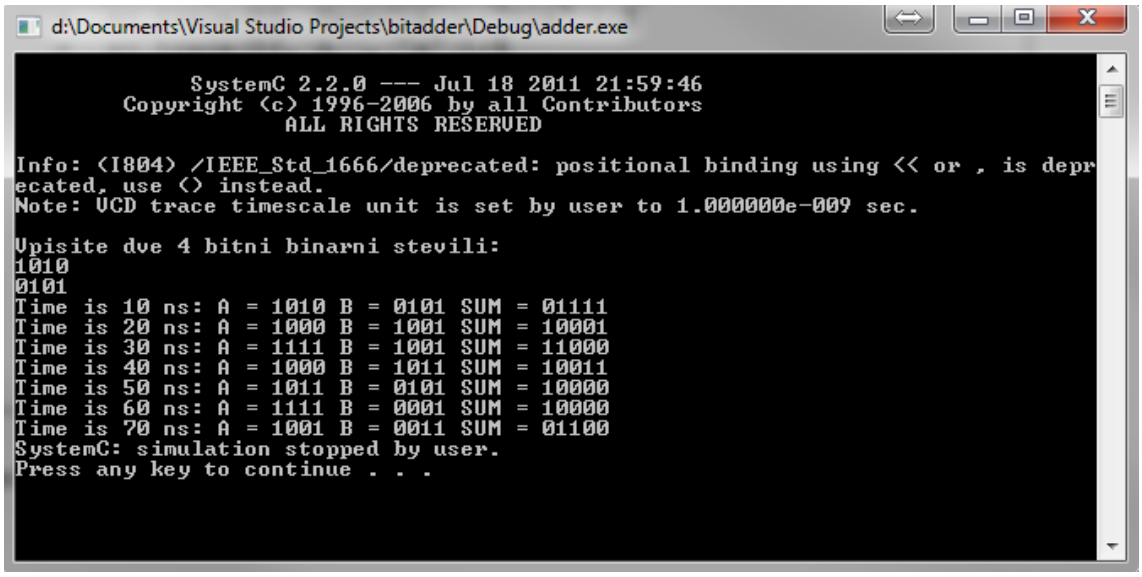
```
        A = "1011";
        B = "0101";
        Cin.write(SC_LOGIC_0);
        wait();
        A = "1111";
        B = "0001";
        Cin.write(SC_LOGIC_0);
        wait();
        A = "1001";
        B = "0011";
        Cin.write(SC_LOGIC_0);
        wait();

        sc_stop();
    }

    SC_CTOR(stim)
    {
        SC_THREAD(StimGen);
        sensitive << Clk.pos();
    }
};
```

Najvišji nivo v datoteki `main.cpp` vsebuje vse podmodele in zažene simulacijo. Vsebina datoteke `main.cpp` je sledeča:

```
#include "systemc.h"
#include "full_adder.h"
#include "stim.h"
#include "mon.h"
#include "vector.h"

int sc_main(int argc, char* argv[])
{
    sc_signal<sc_lv<4> > A, B;
    sc_signal<sc_lv<5> > SUM;
    sc_signal<sc_logic> A0,A1,A2,A3,B0,B1,B2,B3,S0,S1,S2,S3;
    sc_signal<sc_logic> Cin,C1,C2,C3,Cout;
    sc_clock TestClk("TestClock", 10, SC_NS,0.5, 10, SC_NS);

    stim Test1("Test1"); // generiranje vhodov A in B
        Test1 << A << B << Cin << TestClk;

    vector_in VectorA("VectorA"); // razčlemba vektorja A
        VectorA << A << A3 << A2 << A1 << A0;
    vector_in VectorB("VectorB");
        VectorB << B << B3 << B2 << B1 << B0;

    full_adder Adder1("Adder1"); // instance polnih seštevalnikov
        Adder1 << A0 << B0 << Cin << S0 << C1;
```

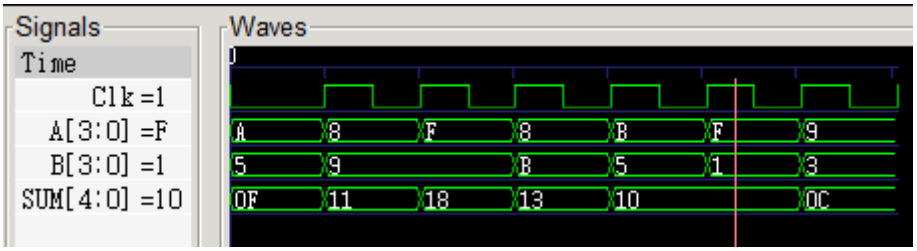
```
    full_adder Adder2("Adder2");
         Adder2 << A1 << B1 << C1 << S1 << C2;
    full_adder Adder3("Adder3");
         Adder3 << A2 << B2 << C2 << S2 << C3;
    full_adder Adder4("Adder4");
         Adder4 << A3 << B3 << C3 << S3 << Cout;

    vector_out VectorS("VectorS"); // sestavljanje bitov v vsoto
         VectorS << Cout << S3 << S2 << S1 << S0 << SUM;

    mon Monitor1("Monitor"); // »monitor« za izpis
         Monitor1 << A << B << SUM << Cin << Cout << TestClk;

    sc_trace_file* Tf; // sledenje v valovni obliki
    Tf = sc_create_vcd_trace_file("traces");
    ((vcd_trace_file*)Tf)->sc_set_vcd_time_unit(-9);
         sc_trace(Tf, TestClk  , "Clk" );
         sc_trace(Tf, A   , "A" );
         sc_trace(Tf, B   , "B" );
         sc_trace(Tf, SUM   , "SUM" );

    sc_start();

    sc_close_vcd_trace_file(Tf);

    system("pause");

    return 0;
}
```

Pri danem generatorju signalov dobimo izpis, prikazan na sliki 16.




Slika 16: Izpis seštevalnika.

Simulaciji lahko sledimo tudi v valovni obliki s pomočjo datoteke `*.vcd` in programa GTKWave Analyzer [5]. Rezultat je prikazan na sliki 17.




Slika 17: Izpis seštevalnika v GTKWave Analyzer.

## 4.3 Sklepne ugotovitve

Model 4-bitnega seštevalnika je realiziran na sistemskem nivoju abstrakcije. Kot je bilo že navedeno v uvodu, je pri višjenivojskem modelu obnašanje modelirano s transakcijami in v modelu ni uporabljenih logičnih vrat, vendar so operatorji modelirani z uporabo funkcij C++. Če bi želeli nižjenivojski model, bi operator XOR za izračun vsote nadomestili z nižjenivojskim vzorčnim primerom vrat XOR iz tretjega poglavja in prav tako bi operatorja AND in OR nadomestili z realiziacijo na nivoju vrat. Prednost višjenivojskega modela je tudi hitrost simulacije, ki je pri kompleksnih modelih, na katerih se testira programska oprema, zelo pomembna.

Seštevalnik lahko uporabimo za testiranje seštevanja dveh štiribitnih dvojiških števil, lahko pa z uporabo dveh instanc 4-bitnega seštevalnika zmodeliramo 8-bitni seštevalnik. Višjenivojski izvršljivi model lahko uporabimo za testiranje programske opreme, preden se načrtovalski postopek nadaljuje na nižjih nivojih abstrakcije. Tak model lahko dodatno nadgradimo, dodelamo in testiramo, s čimer bomo prihranili na času in sredstvih. Pri načrtovanju sistemov se velikokrat odločamo, katere dele sistema bomo realizirali programsko in katere strojno. Na višjih nivojih je prehod iz ene realizacije v drugo veliko bolj enostaven in hiter.

# 5 Zaključek

Z jezikom, ki opisuje strojno opremo, smo se prvič srečali pri predmetu Logične strukture in sistemi 2, in sicer je bil to VHDL, s katerim smo opisovali logična vezja. Pri predmetu Vrednotenje računalniške opreme pa smo se seznanili tudi z jezikom SystemC, ki je prav tako namenjen opisovanju in načrtovanju strojne opreme, vendar na višjem nivoju abstrakcije.

V diplomskem delu je analiziran jezik SystemC, predstavljene so prednosti sočasnega načrtovanja strojne in programske opreme ter pomen načrtovanja sistemov na višjem nivoju abstrakcije. Na vzorčnem primeru vrat XOR je prikazana uporaba osnovnih gradnikov jezika, hierarhija in funkcionalnost ter na sistemskem nivoju abstrakcije zmodeliran 4-bitni seštevalnik.

Uporaba visokonivojskih konceptov načrtovanja z uporabo SystemC omogoča učinkovitejše načrtovanje sistema in tako pripomore k večjemu obvladovanju načrtovalskega procesa, hitrejšo usmeritev načrtovalskega napora k detajlom sistema ter predvsem hitrejši in cenejši razvoj sistema. Poleg tega je SystemC dokaj enostaven za uporabo, saj temelji na C++, ki ga uporablja vsak inženir tehničnih šol. Iz poročila [9], ki ga je objavil OSCI, je razvidno, da uporaba jezika SystemC zelo narašča in da ima skupine uporabnikov po vsem svetu.

Danes poteka veliko raziskav v smeri prevajanja kode SystemC v kodo VHDL [3], kar bi načrtovalski potek še bolj poenostavilo in skrajšalo.

# Seznam slik

# Literatura


[1] D. C. Black, J. Donovan, B. Bunton, A. Keist, »SystemC: from the ground up«, 2010, Springer New York.

[2] Doulos. Dostopno na: http://www.doulos.com

[3] E.P.M. van Diggele, MSc Thesis, »Translation of SystemC to Synthesizable VHDL«, 2006, Delft University of Technology, Faculty of Electrical Engineering, Mathematics and Computer Science.

[4] Forte Design Systems. Dostopno na: http://www.forteds.com

[5] GTKWave Analyzer. Dostopno na: http://gtkwave.sourceforge.net/

[6] IEEE Standard SystemC Language Reference Manual. Dostopno na: http://standards.ieee.org/getieee/1666/download/1666-2005.pdf

[7] Jože Dedič, doktorska dizertacija, »Enovito razvojno okolje za sočasno načrtovanje strojne in programske opreme«, 2006, Univerza v Ljubljani, Fakulteta za elektrotehniko.

[8] Open SystemC Initiative. Dostopno na: www.systemc.org

[9] SystemC Users Group Survey Data Trend Report, April 2007. Dostopno na: http://www.systemc.org/community/user_groups/OSCI_2007_Survey_Data_Trends_Report.pdf