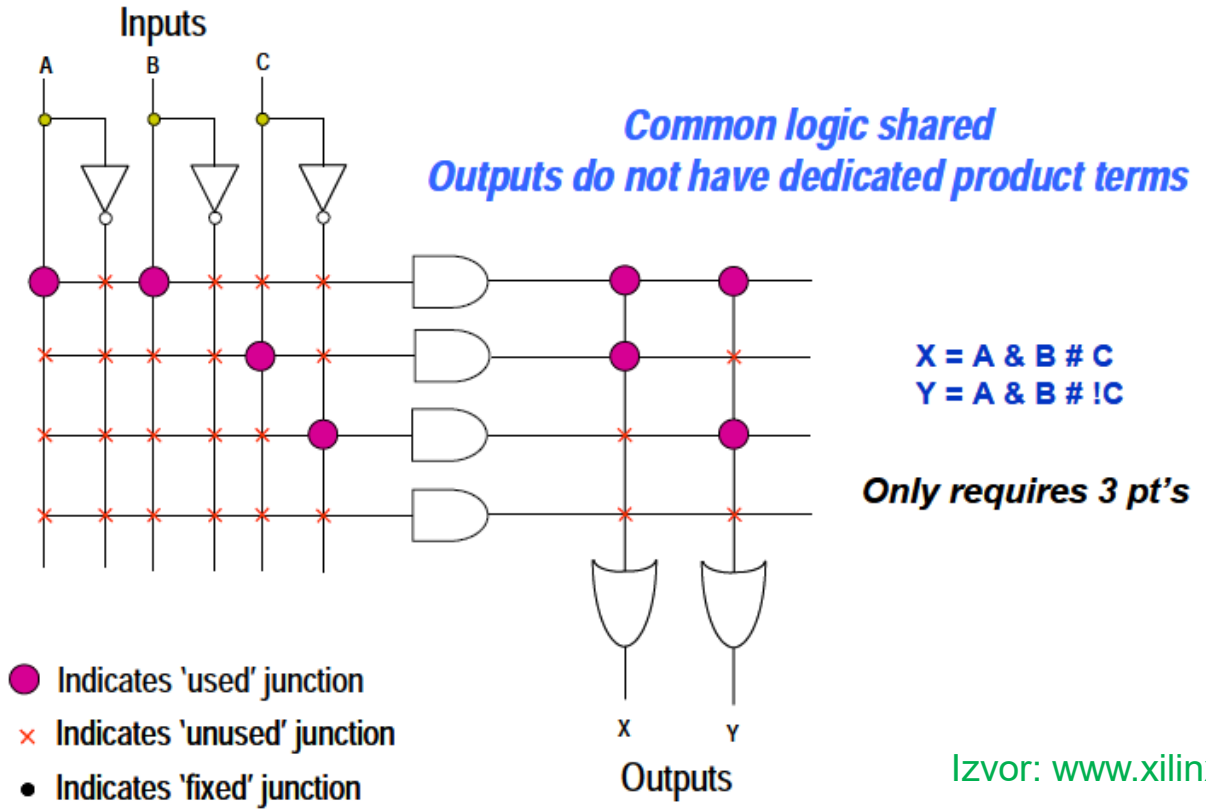
Dva pristupa u dizajnu hardvera (podsjećanje)

- Procesori opšte namjene (mikroprocesori)
  - ↑ Fleksibilni jer se mogu reprogramirati
  - ↓ Neefikasna potrošnja energije
  - ↓ Slabije performanse
- Integrisana kola za konkretnu namjenu (Application-Specific Integrated Circuits - ASICs)
  - ↓ Implementirana samo jedna funkcija - svaka promjena u funkcionalnosti zahtjeva ponovno dizajniranje kola od početka
  - ↑ Efikasnija potrošnja energije
  - ↑ Visoke performance
- Između ova dva ekstrema postoji nekoliko arhitektura koje pružaju najbolje od gore navedenih: bliži su hardveru, ali se mogu preprogramirati
  - PLA, PAL, PLD, CPLD, FPGA

Programmable Logic Array - PLA

- Da bi se postigla fleksibilnost u dizajniranju, došlo se do ideje da se naprave dvije progamabilne ravni: obezbeđuju sve moguće kombinacije AND i OR funkcija, kao i korišćenje jednog AND izraza kod više OR funkcija



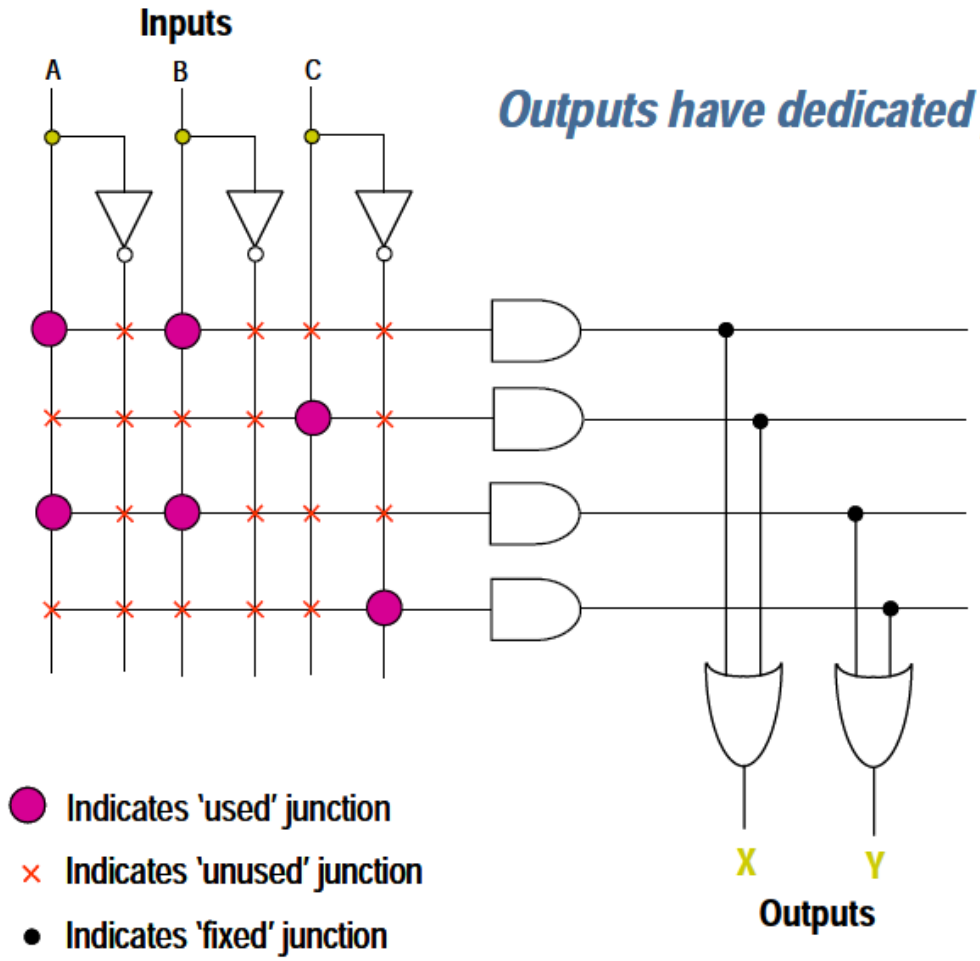
Izvor: www.xilinx.xom

Programmable Logic Array – PLA - nastavak

- Svaka funkcija se može prikazati u obliku zbira logičkih proizvoda
  - AND ravan realizuje logičke proizvode
  - OR ravan realizuje zbir logičkih proizvoda
- Tehnologija izrade je uzrokovala unošenje velikog propagacionog kašnjenja, pa su kola bila relativno spora
- Karakteristike PLA:
  - Dvije programabilne ravni
  - Moguće realizovati svaku kombinaciju AND i OR izraza
  - Jedan AND izraz se može koristiti kod više OR izraza
  - Programiranje se vrši „spaljivanjem osigurača” u ukrsnim tačkama polja
  - Veliki broj „osigurača”

Programmable Array Logic - PAL

- Razlikuje se od PLA u tome što je jedna programabilna ravan fiksirana: OR ravan



Izvor: www.xilinx.xom

Programmable Array Logic – PAL - nastavak

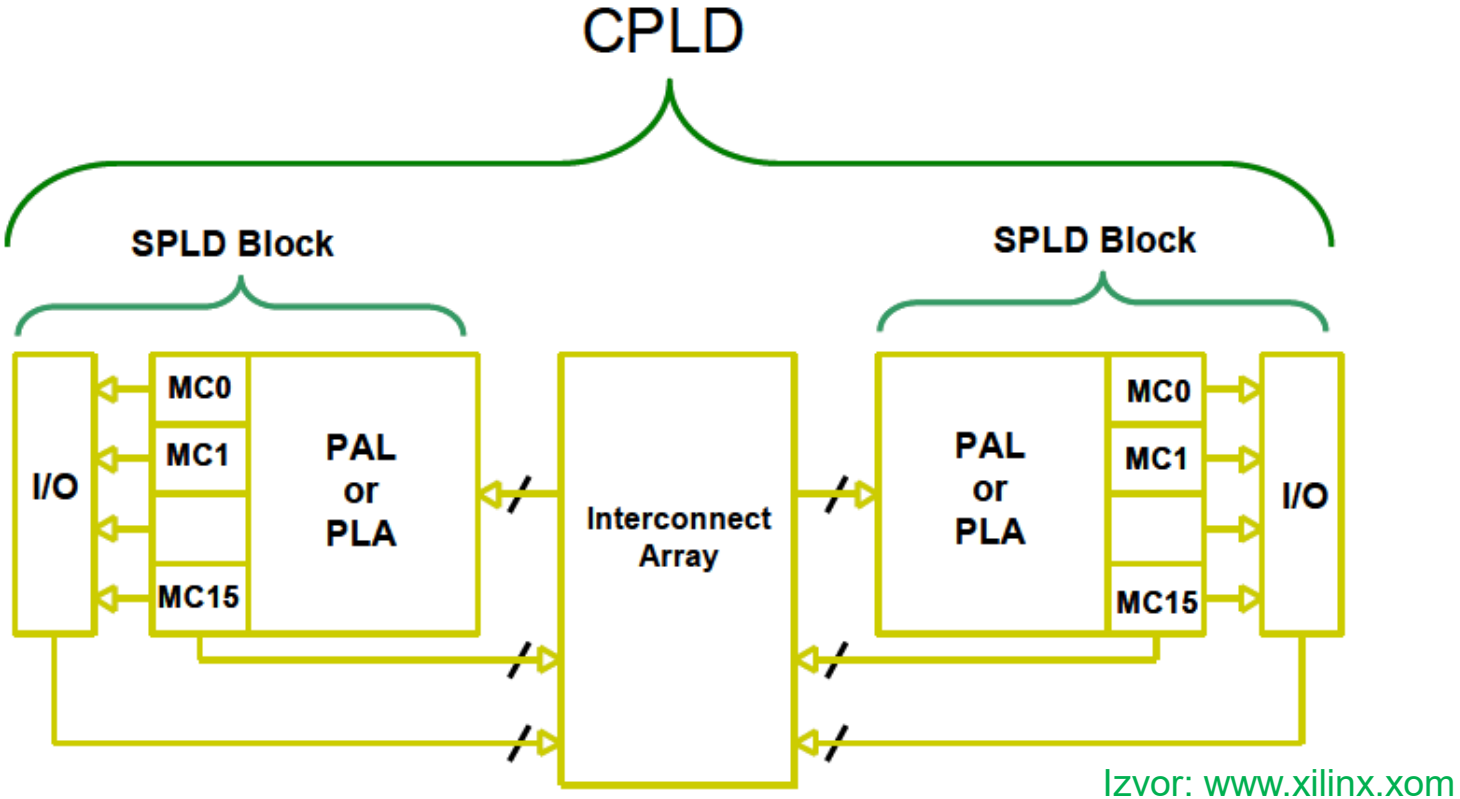
- Smanjena je fleksibilnost koju su imala PLA kola, ali je pojednostavljen softver za programiranje PAL kola
- Karakteristike PAL:
  - Jedna programabilna ravan (AND) – druga (OR) je fiksirana
  - Ograničen je broj kombinacija AND i OR izraza
  - Programiranje se vrši „spaljivanjem osigurača” u ukrsnim tačkama polja
  - Manji broj „osigurača” nego kod PLA, dobija se na brzini
- Slijede nove arhitekture programabilnih kola: PLD (Programmable Logic Devices)

Programmable Logic Device – PLD

- Arhitekture imaju mrežu vertikanih i horizontalnih interkonekcionih linija
- U svakoj ukrsnoj tački se nalazi „osigurač” (oslabljeni vod)
- Uz pomoć softverskih alata projektant odlučuje koje će linije biti povezane u ukrsnim tačkama tako što „pregori” osigurače u tačkama gdje kontakt ne treba da postoji
- Ulazni portovi se povezuju na vertikalne linije
- Izlazi OR kapija se (preko flip flopova) povezuju na izlazne portove
- Tehnologija dalje napreduje pojavom *flash* memorija: kolo se može više puta programirati i to tako što se električnim putem programira (nisu potrebni specijalni programatori) i „briše” (bez UV svjetlosti)

Complex Programmable Logic Device – CPLD

- Nekoliko PLD blokova ili makroćelija između kojih se nalaze interkonekcione linije
- Jednostavnije funkcije se realizuju unutar jednog bloka, a složenije zahtijevaju upotrebu više blokova i interkonekcija



Izvor: www.xilinx.xom

Problemi modelovanja (zašto HDL?)

- Suviše kompleksan dizajn: današnji čipovi imaju i preko 2.000.000.000 tranzistora
- Nemoguće ručno dizajnirati: po tranzistoru 10 sekundi => za dizajn čitavog sistema potrebno 1268 godina
- Nemoguće verifikovati na eksperimentalnoj pločici
- Savremeni dizajn je zasnovan na sofisticiranim alatima
  - Sistem se opisuje na visokom nivou apstrakcije (tekstualno ili grafički)
  - Nije potrebno unaprijed izabrati tehnologiju fabrikacije (alati za logičku sintezu će konvertovati dizajn za bilo koju tehnologiju fabrikacije)
  - U slučaju prelaska na novu tehnologiju nema potrebe za redizajnom
  - Alati za logičku sintezu će optimizovati kolo, i u smislu prostora i u smislu brzine rada, za novu tehnologiju

Problemi modelovanja (zašto HDL?) - nastavak

- Funkcionalna verifikacija se može obaviti u ranom stepenu razvoja čime se smanjuje vjerovatnoća pojave grešaka
- Alatima za sintezu se dobija opis na nižem nivou apstrakcije: logičke kapije (*gejtovi*), tranzistori
- Uz pomoć alata, u zavisnosti od tehnologije, sistem se implementira
- Eventualno se neki veoma mali kritični djelovi optimizuju ručno
- Tekstualni opis (sa komentarima) je najlakši način za razvoj i za otklanjanje neispravnosti
- Reprezentacija je veoma koncizna u poređenju sa šematskim prikazom – šematski prikaz je gotovo neupotrebljiv kod veoma složenih sistema

Zašto Verilog?

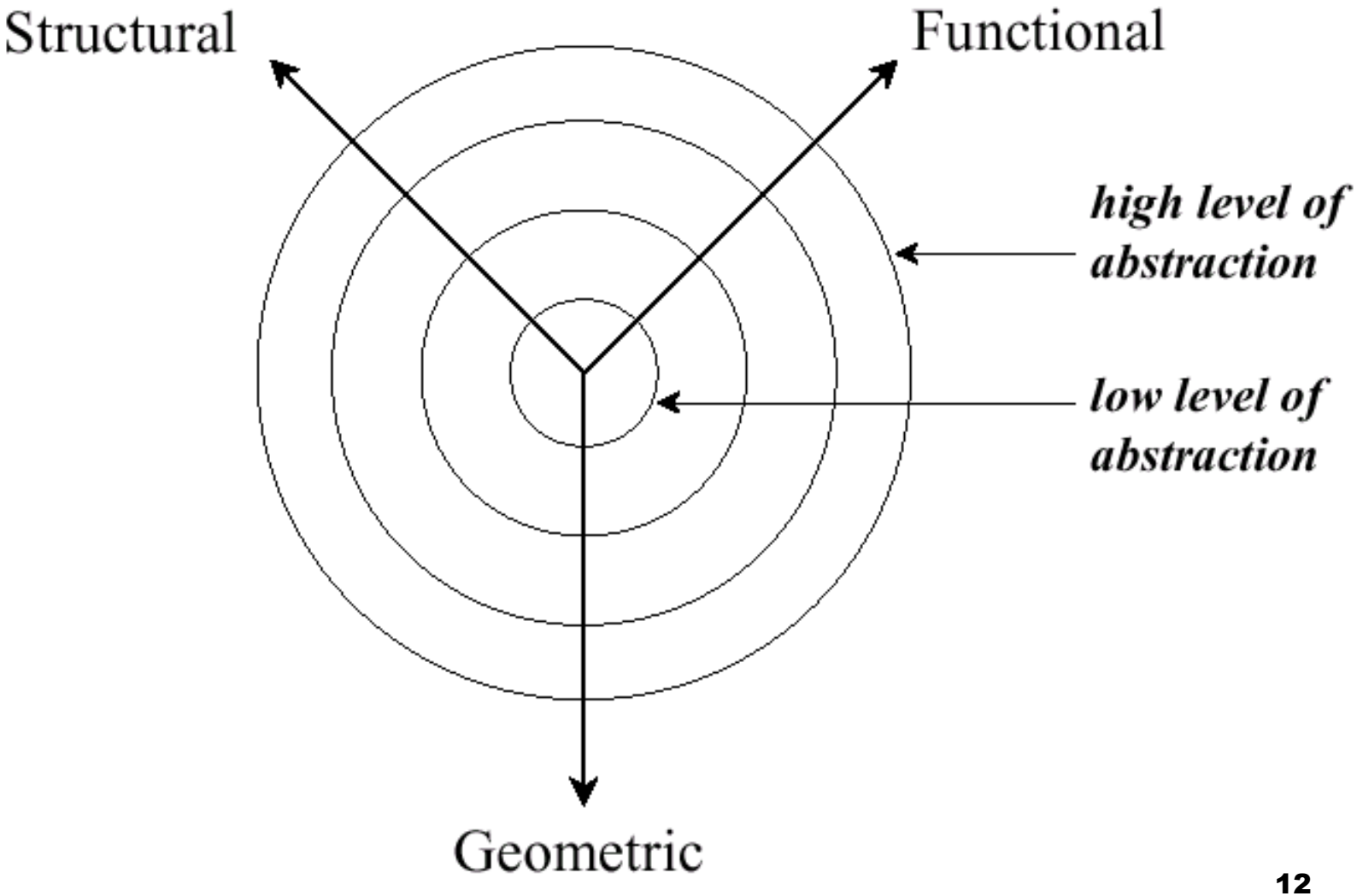
- Verilog je jednostavan i elegantan
- Njegove konstrukcije za opisivanje hardverskih elemenata su u konciznoj i čitljivoj formi
  - Za upoređenje: opis istog elementa u nekom drugom jeziku za opis hardvera može biti i duplo duži
- Sa Verilogom projektant mora da nauči samo jedan jezik za sve aspekte logičkog dizajna
- Omogućava da se koriste različiti nivoi apstrakcije, pomiješani u istom modelu
- Simulacija dizajna zahtijeva funkcionalne modele, hijerarhijske strukture, testne vektore, interakciju čovjek-uređaj...
  - U Verilogu je sve ovo omogućeno jednim jezikom

Zašto Verilog? - nastavak

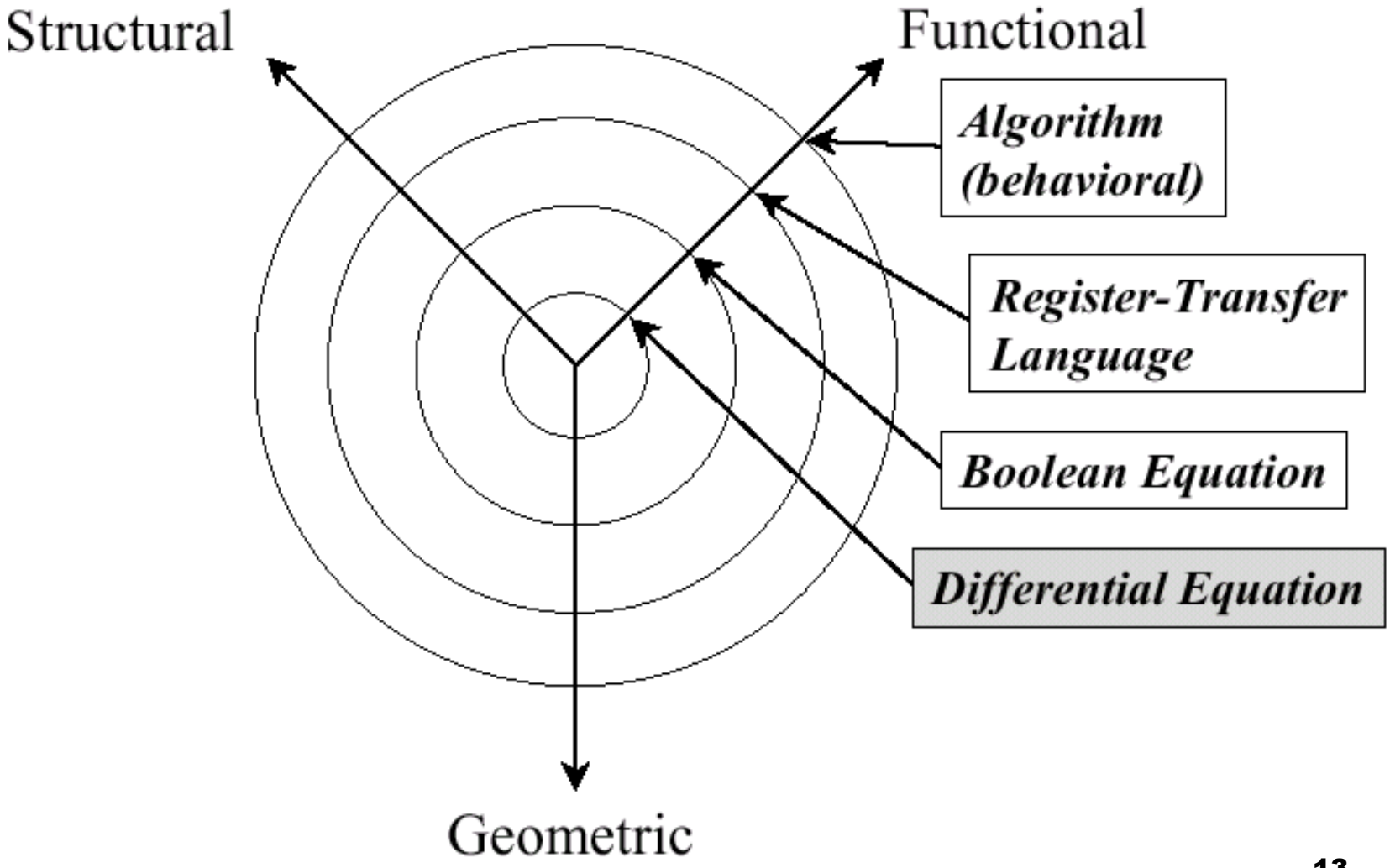
- Verilog se lako uči
  - Veoma je sličan programskom jeziku C
  - Pošto je C jedan od najviše korišćenih programskih jezika, većina projektanata bi trebalo da su familijarni sa njim i zato im je lako da nauče Verilog
- PLI (*Programming Language Interface*) je moćna funkcija koja omogućava korisniku da napiše prilagođeni C kod za interakciju sa internim strukturama podataka u Verilogu
- Dizajneri mogu da prilagodite Verilog HDL simulator njihovim potrebama koristeći PLI

Domeni modelovanja

Gajski-Kuhn (GK) Y-chart:



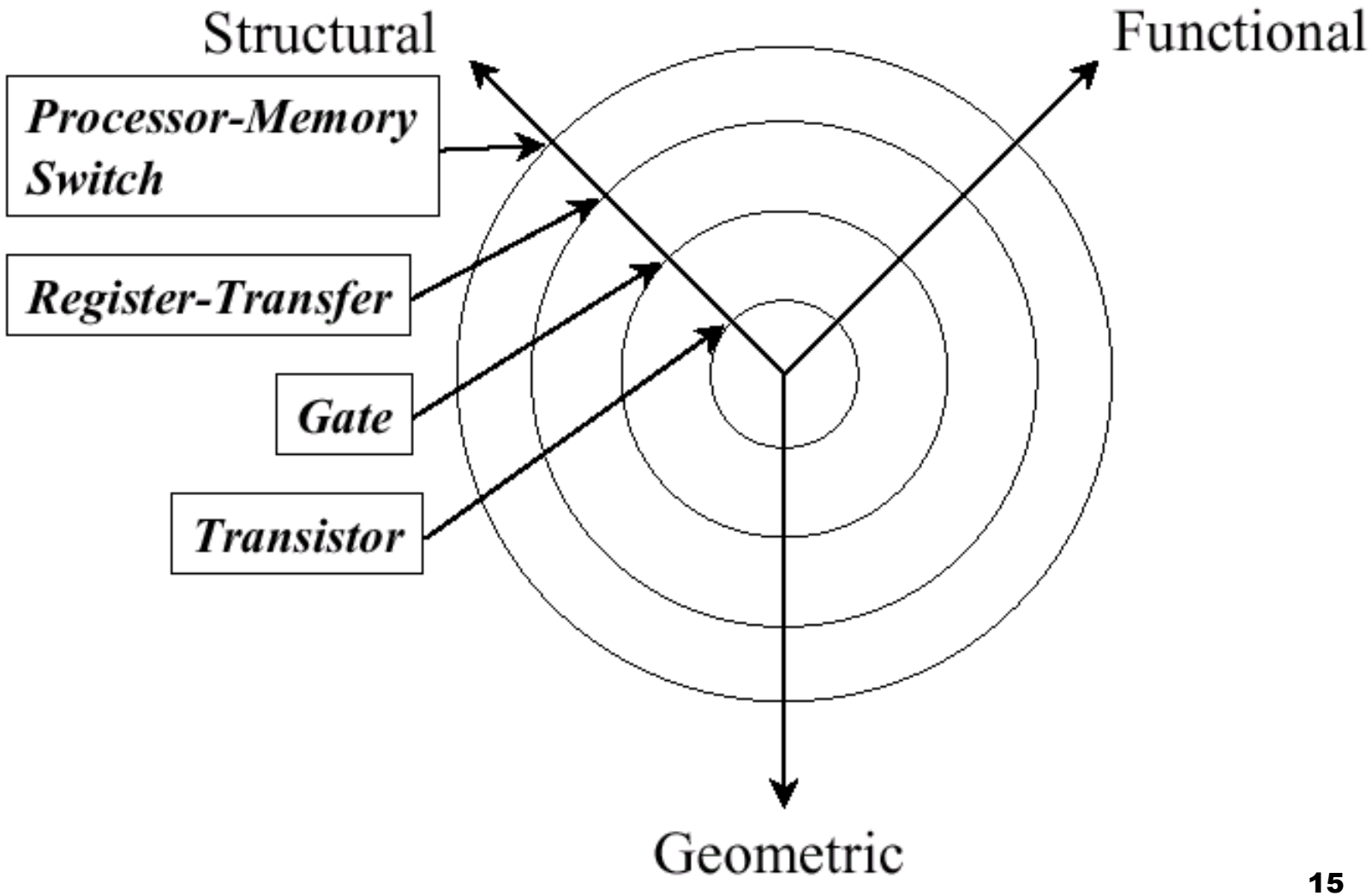
- Domeni modelovanja



Domeni modelovanja

- Funkcionalni (behavioral) domen
  - Opis ponašanja sistema u funkciji ulaza i proteklog vremena
  - Daje odgovor na pitanje: šta ili kako sistem radi?
  - Definiše algoritam rada sistema i interfejs između sistema i okruženja
  - Rezultat: funkcionalna reprezentacija u obliku dijagrama toka, programa u višem programskom jeziku, opisa u HDL-u, matematičke formule, grafikona, ...

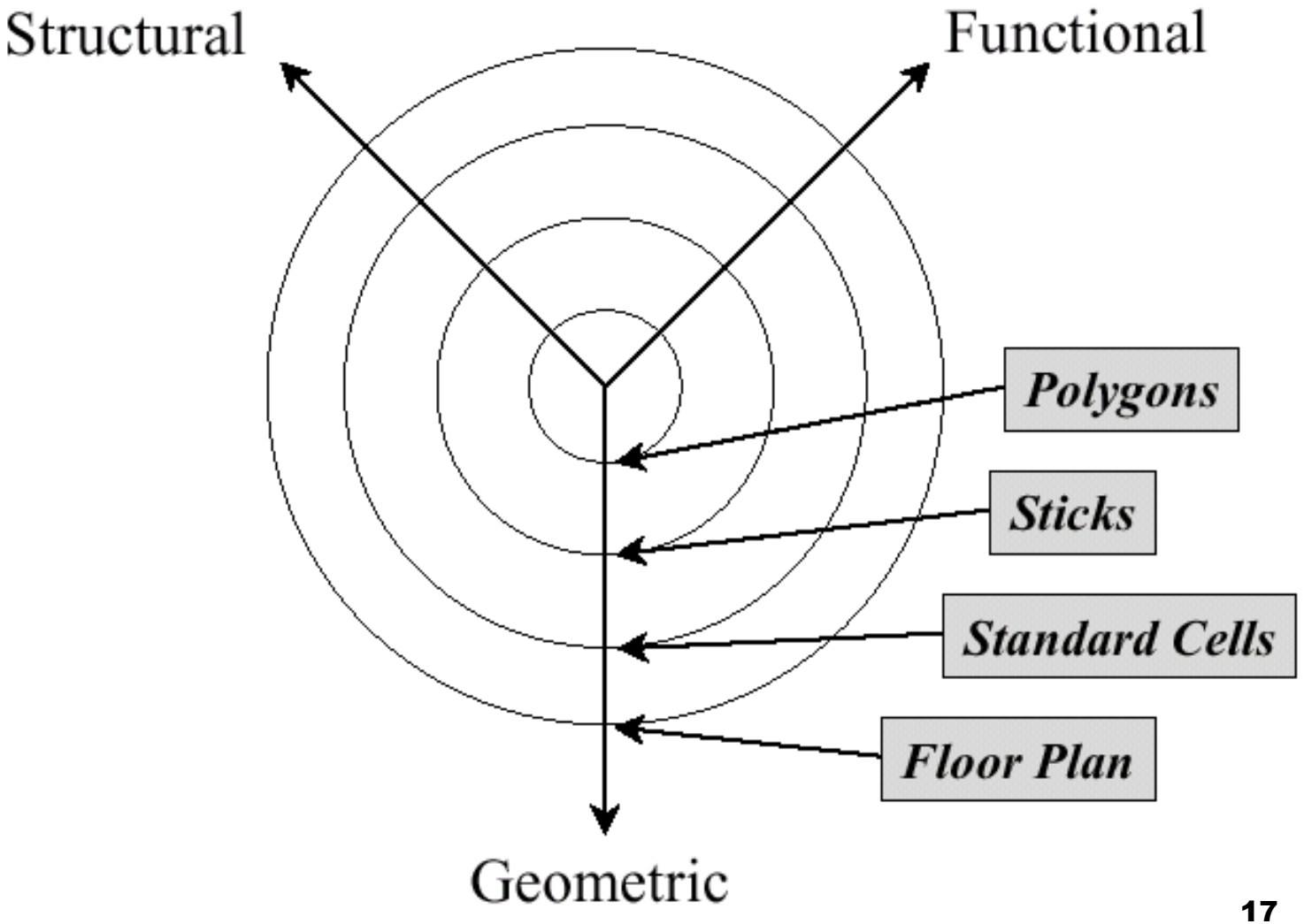
- Domeni modelovanja



Domeni modelovanja

- Strukturni domen
  - Razlaže sistem na skup komponenti i njihovih veza
  - Daje odgovor na pitanje: kako pomoću raspoloživih komponenti realizovati sistem zadate funkcije?
  - Rezultat: strukturna reprezentacija u vidu blok dijagrama, šematskog prikaza, netliste, ...

- Domeni modelovanja



Domeni modelovanja

- Fizički domen
  - Definiše fizičke karakteristike sistema (dimenzije i poziciju komponenata iz strukturnog opisa)
  - Bavi se fizičkim raspoređivanjem i povezivanjem komponenti
  - Rezultat: fizička reprezentacija u obliku *layout*-a čipa, crteža štampane ploče, ...
- Konačni proizvod projektovanja na osnovu koga se može direktno realizovati sistem ili fabrikovati čip

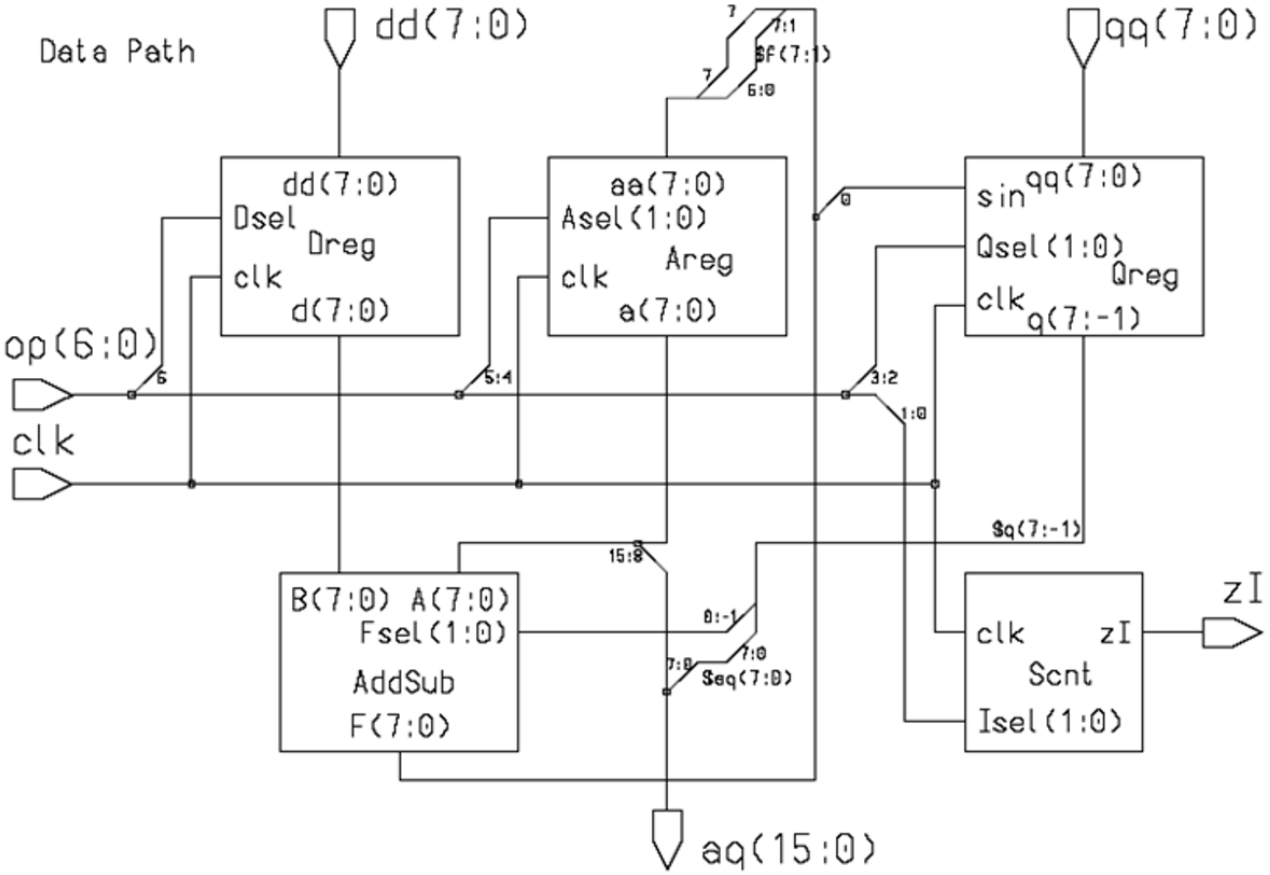
Domeni modelovanja - rezime

- Funkcionalni domen (šta sistem radi)

$$
\begin{aligned}
&P[0] = 0\,; \quad q_{-1} = 0\,; \\
&\textbf{for}\ (i = 0\,; \quad i < n\,; \quad i{+}{+}\,)\,\{ \\
&\qquad \hat{q}_i = -q_i + q_{i-1}\,; \\
&\qquad P[i+1] = (P[i] + \hat{q}_i \cdot D) \cdot 2^{-1}\,; \\
&\}
\end{aligned}
$$

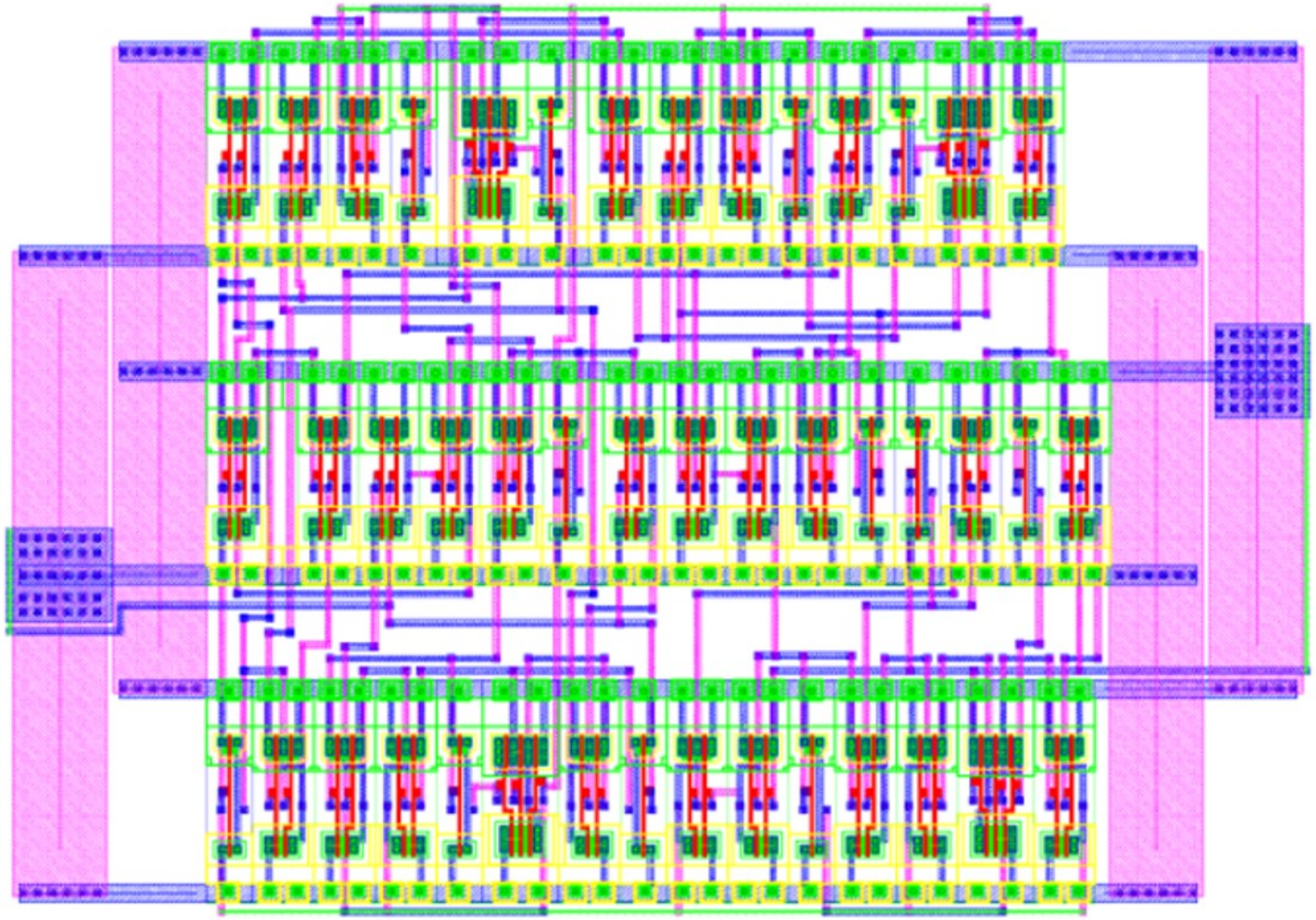
Domeni modelovanja - rezime

- Strukturni domen (sistem razložen na komponente)



Domeni modelovanja - rezime

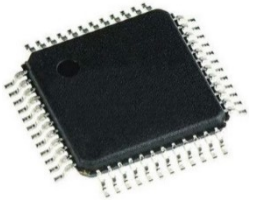
- Fizički domen (kako je sistem napravljen)



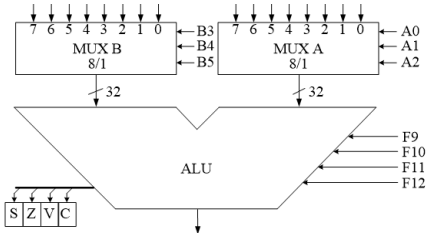
Nivoi apstrakcije

Visok

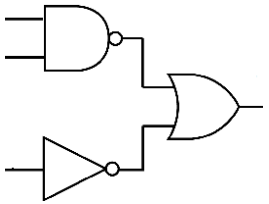
Sistemski nivo



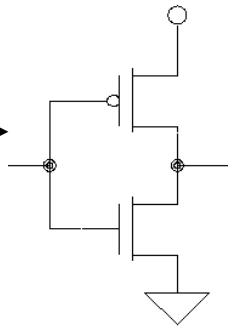
RTL nivo



Nivo logičkih kapija



Nivo tranzistora



Nizak

Nivoi apstrakcije

Nivo apstrakcije	Funkcionalna reprezentacija	Strukturne komponente	Fizički objekti
Nivo tranzistora	Diferencijalne jednačine, U/I dijagrami	Tranzistori, otpornici, kondenzatori	Analogne ćelije, digitalne ćelije
Nivo logičkih kapija	Logičke funkcije, automati (FSM)	Logička kola, flip flopovi	Moduli, funkcionalne jedinice
Registarski nivo (RTL)	Algoritmi, dijagrami toka, ASM dijagrami	Registri, multipleksori, dekoderi, sabirači	Integralna kola
Sistemski nivo	Specifikacija, programi	Procesori, kontroleri, memorije	Štampane ploče, moduli

FSM – Finite State Machine ASM – Algorithmic State Machine

Načini unosa dizajna

- Tekstualni
  - dovoljan obični tekstualni editor
  - fleksibilan
  - mogućnost opisa hardvera i na nivou algoritma
  - pregledan
  - pogodan za dokumentovanje
- Grafički
  - potrebni posebni alati
  - nije pogodan za opis na višem nivou apstrakcije

Testiranje - Verifikacija

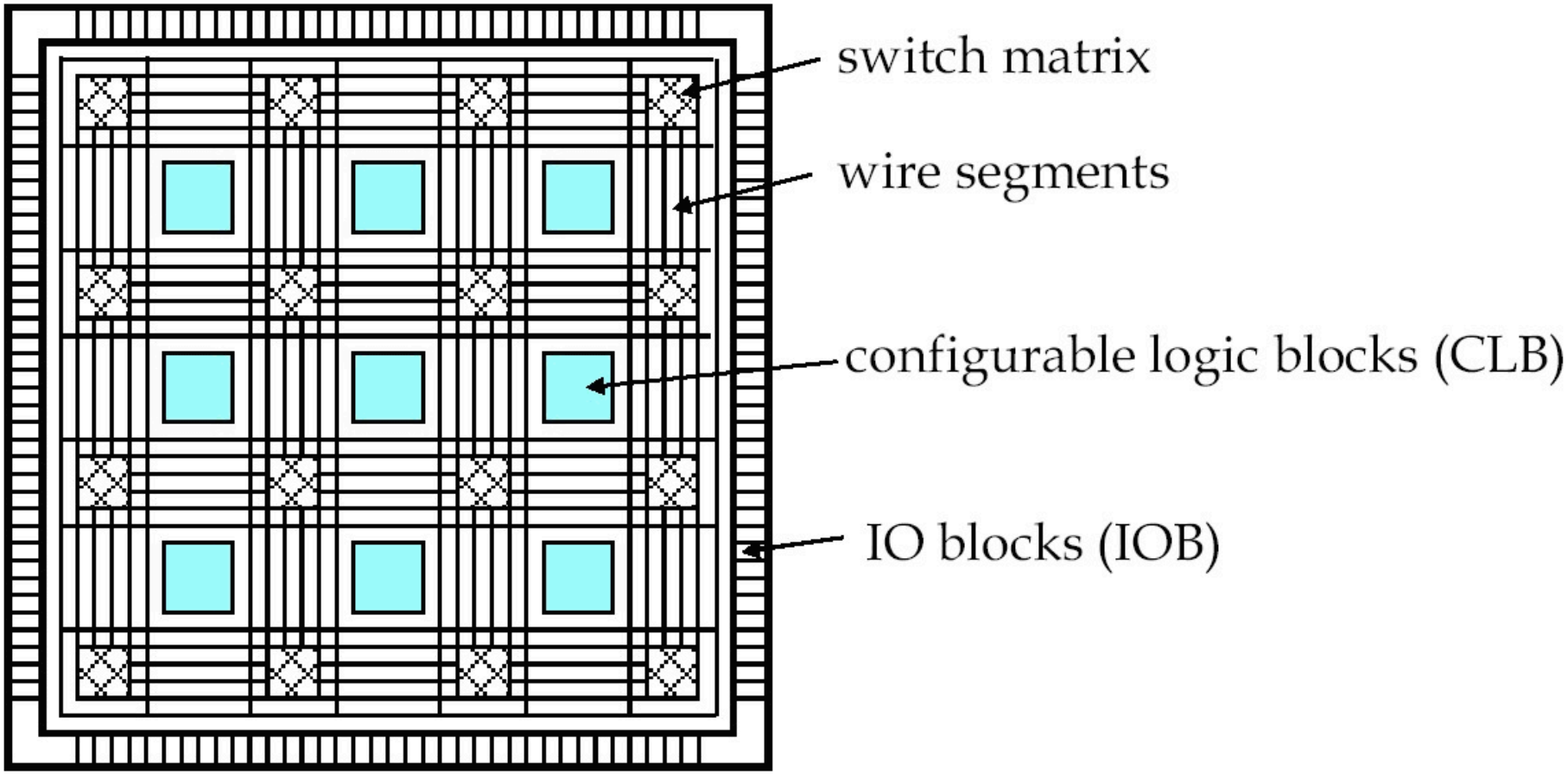
- Prije fabrikacije, sva testiranja se obavljaju simulacijom
- Koristi se ***Test bench*** model:
  - entitet bez portova
  - arhitektura koja sadrži: model koji se testira + dodatni procesi/entiteti koji:
    - generišu test vektore koji se dovode na ulaz testiranog kola
    - kontrolišu ispravnost izlaznih signala (ukoliko se provjera vrši automatizovano)
- Provjera ispravnosti izlaznih signala može da se obavlja i ručno, korišćenjem simulatora

Regresivno testiranje (regression test)

- Potrebno je potvrditi ispravnost (kasnijeg) dizajna na nižem nivou
  - strukturni model na niskom nivou apstrakcije treba da daje iste rezultate kao i funkcionalni model na višem nivou apstrakcije
- Test bench sadrži dvije instance koje se testiraju
  - strukturni i funkcionalni model
  - oba modela se paralelno simuliraju sa istim ulaznim test vektorima
  - porede se rezultati na izlazu
- Potrebno je voditi računa o vremenu (**tajming**)
  - funkcionalni model lako može da previdi određena kašnjenja koja su posljedica strukture modela

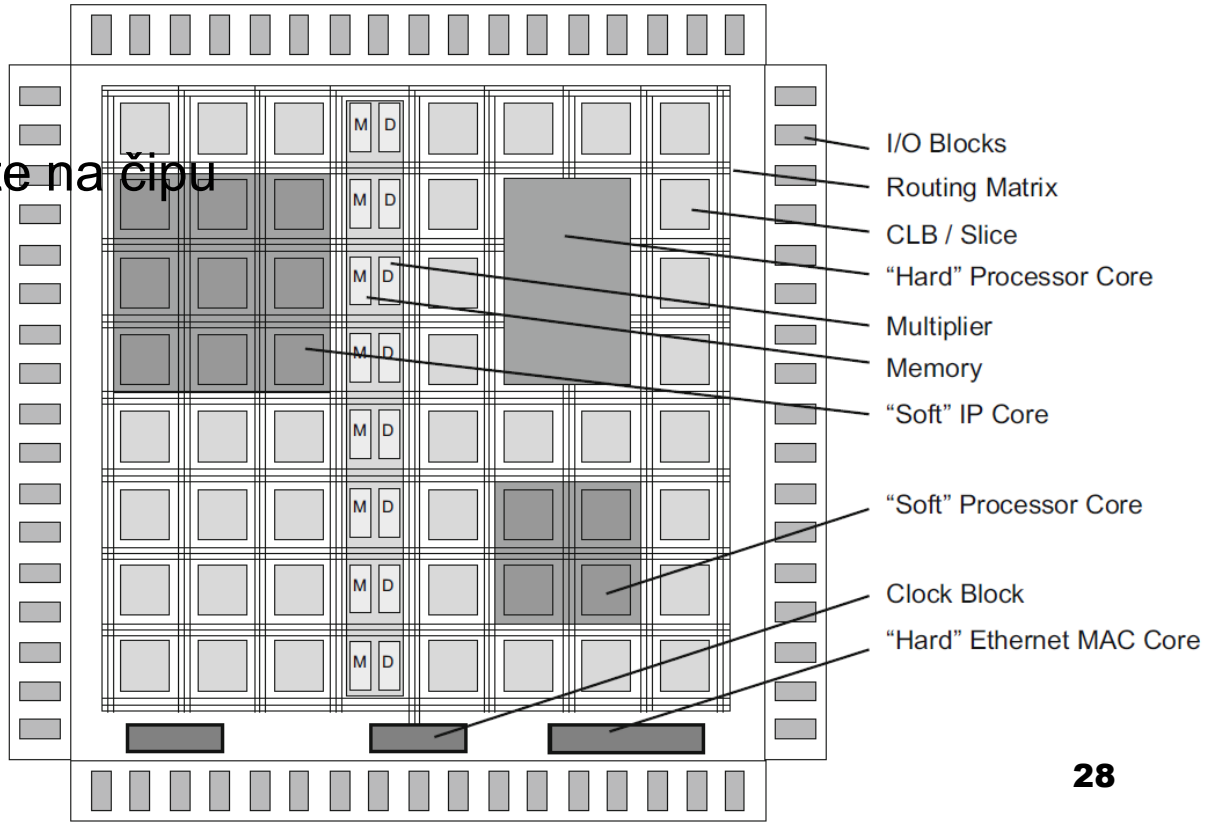
FPGA = Field Programmable Gate Arrays

- Svaki proizvođač ima svoju arhitekturu za FPGA čipove, ali je generalni koncept isti kod svih:



Šta je konfigurabilno?

- CLB
  - LUT
  - interno povezivanje
- IOB
- Veze između CLB
- Namjenske komponente na čipu
  - množači
  - memorija
  - kontrola takta
  - ALU ...



Kako se konfiguriše („programira“)?

- SRAM
  - za svaki programabilni element se uvodi SRAM bit
  - upisivanjem nule u taj bit isključije se prekidač, a upisivanjem jedinice se uključuje
  - prednost je što se koristi standardni proces fabrikacije; može se programirati više puta,...
  - mana je što naponski impuls može da izmijeni sadržaj nekog bita; velika su kašnjenja
- *Anti-fuses*
  - mikroskopski „anti-osigurači“ koji u početnom stanju ne provode (nema konekcije); protok određene struje tokom programiranja uzrokuje stvaranje konekcije između dva kraja osigurača
  - otporni na spoljašnje uticaje i brži, ali je kompleksniji proces fabrikacije, potreban specijalan programator i ne mogu se mijenjati

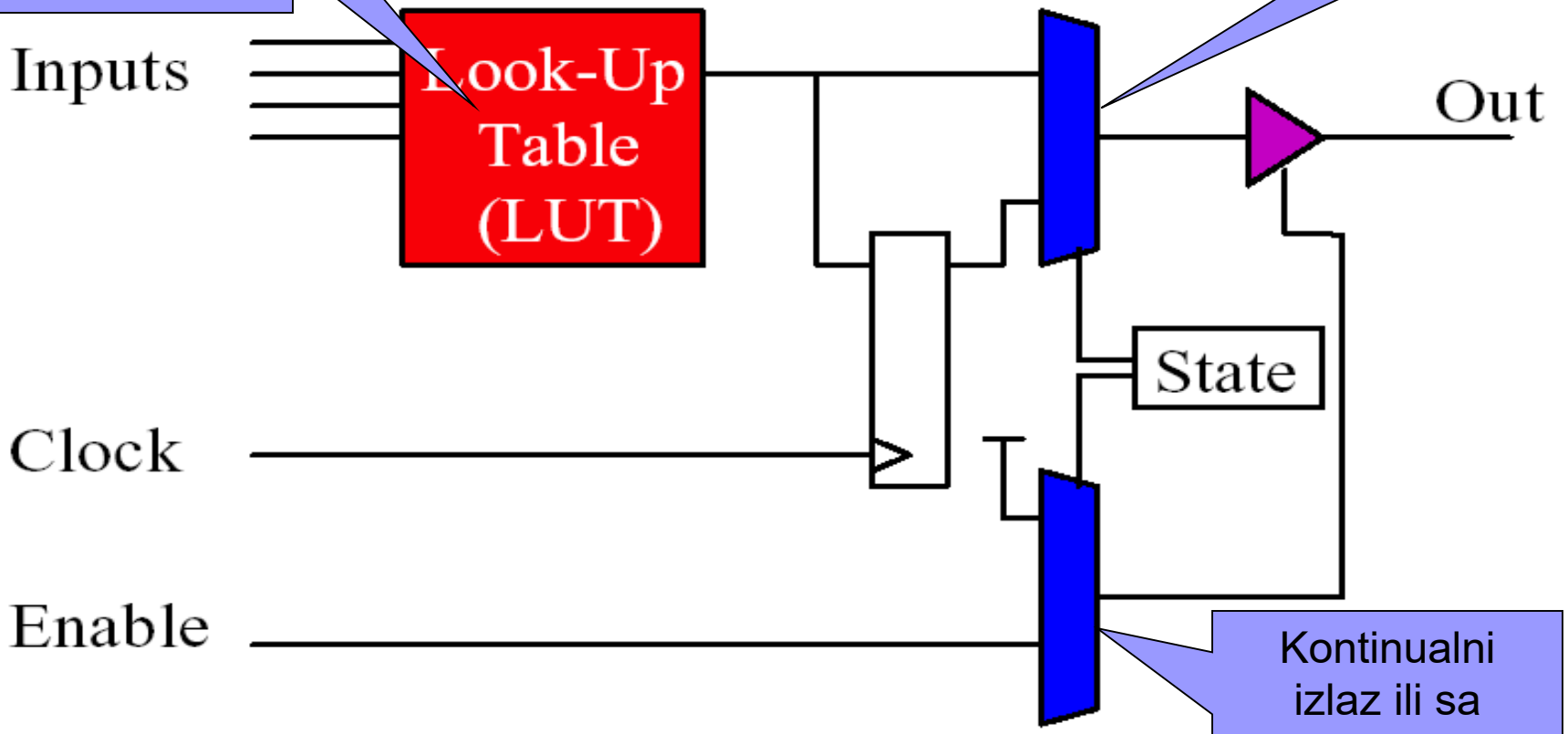
FPGA – prednosti i mane

Karakteristika	**FPGA**	**Full-Custom**
Vrijeme razvoja	Kratko	Dugo
Cijena proizvoda (u velikim serijama)	Visoka	Niska
Mogućnost izmjene nakon fabrikacije	Moguće	Nemoguće
Performanse	Srednje	Veoma visoke
Gustina pakovanja	Srednja	Veoma visoka
Potrošnja energije	Visoka	Niska
Minimalna veličina serije	Jedan	Velika
Kompleksnost dizajna	Srednja	Visoka
Kompleksnost testiranja	Srednja	Visoka
Vrijeme za ispravku greške	Sati	Mjeseci

Moguće formirati proizvoljnu logičku funkciju

rable Logic Block

Sinhroni ili asinhroni izlaz



Kontinualni izlaz ili sa signalom dozvole

CLB - nastavak

- U zavisnosti od konkretne implementacije mijenja se:
  - broj i veličina *LUT* tabele
  - broj i vrsta memorijskih elemenata
  - interna kombinaciona logika (mogućnost internog povezivanja)
  - dodatne funkcionalnosti

LUT – Look-Up Table

- 1 bitna memorija
- Ulazi su vezani na adresne linije
- Izlaz je sadržaj memorije sa adrese opisane tekućim vrijednostima na ulazu
- Konstantno kašnjenje

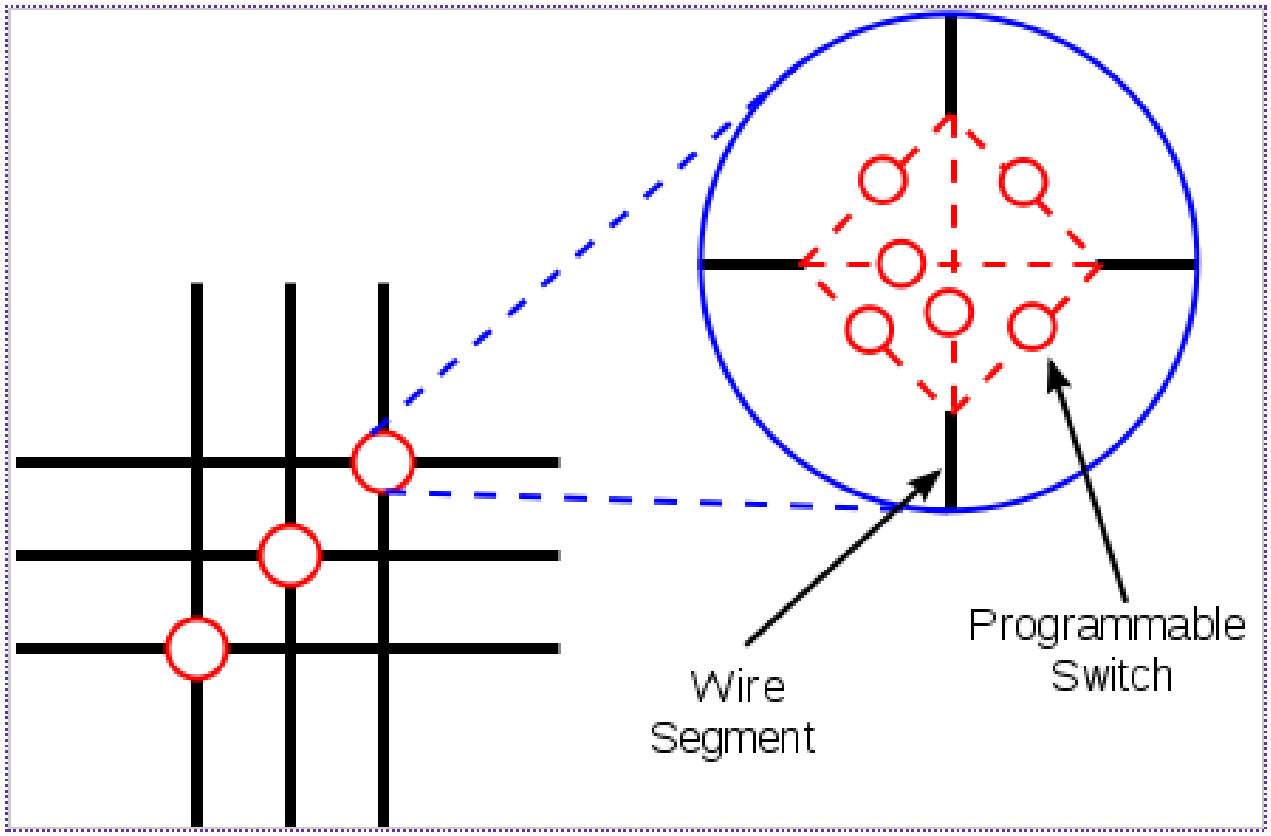
ads	A	B	C	D	F
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	0

Implementacija logičkih funkcija

- Pretpostavka: n-ulazni CLB
- Funkcije se raščlanjuju na n-ulazne funkcije
- Svaka n-ulazna funkcija se mapira na jedan CLB
- Povezivanjem CLB-ova se formira tražena funkcija

Povezivanje

- 2D matrica vodova koji u presjeku imaju konfigurabilne veze



Povezivanje - nastavak

- Zbog efikasnijeg povezivanja veoma često postoji više tipova veza:
  - one koje povezuju susjedne elemente
  - one koje povezuju prve nesusjedne elemente
  - one koje povezuju svakih n-ti, gdje je n cio broj veći od 2
  - krajnje elemente
  - ...

IOB – Input/Output Block

- I/O interfejs je implementiran u obliku I/O blokova
- I/O blokovi su djelovi FPGA arhitekture pozicionirani periferno i povezani sa I/O pinovima kao i sa unutrašnjim interkonekcijama
- I/O blokovi su grupisani u tzv. banke - grupa od susjednih pinova koji koriste isti ili kompatibilan I/O standard u isto vrijeme
- I/O blok obično sadrži:
  - Programabilne I/O bafere (da bi se prilagodili različitim standardima)
  - D - ff (koriste se kao registri ili za eventualno unošenje kašnjenja)
  - pull-up/pull-down otpornici (postavljaju pinove na logičku nulu odnosno jedinicu, a koji bi inače imali neodređeni nivo)
  - Polja za kašnjenje (obezbjeđuju programirano kašnjenje I/O signala)
  - Kola za zadržavanje (zadržavaju posljednje stanje na magistrali ako su svi njeni priključci u stanju visoke impedanse)

IOB – Input/Output Block - nastavak

- Mogućnost konfiguracije:
  - smjer:
    - ulazni
    - izlazni
    - bidirekcioni
  - sinhronizacija:
    - direktni
    - sinhronizovan sa signalom takta
  - niz drugih mogućnosti (pull-up otpornici, pull-down otpornici, ...)

Memorija

- Korišćenje CLB odnosno LUT kao registara ne obezbjeđuje dovoljno memorijskog prostora i dovoljno fleksibilnosti za mnoge upotrebe
- Vremenski zavisne aplikacije (npr. real-time) koje vrše mnogo računanja zahtijevaju ugrađenu memoriju
- Glavne prednosti ugrađene memorije su:
  - kratko vrijeme pristupa
  - veliki propusni opsjeg
  - velika fleksibilnost (može da se ponaša kao):
    - RAM
    - ROM
    - bafer (FIFO, LIFO, FILO, ...)
    - pomjerački registar
    - ...

Dodatni elementi unutar FPGA

- Procesorska jezgra
- Komunikaciona jezgra
  - Ethernet
  - PCIe
  - …
- Generatori takta (oscilatori)
- Komponente za digitalnu obradu signala (DSP slices)
- …

Trend u razvoju: System On a Chip (SOC)

- Slijedeći Murov zakon, unutar FPGA se može smjestiti sve veći broj ćelija i dodatnih elemenata
- Čitav sistem (procesor sa potrebnim periferijama) se smješta na jedan čip:
  - kompaktniji sistem
  - jednostavniji dizajn ostatka sistema
  - jeftiniji sistem
- Ovakva FPGA sve češće zamjenjuju ASIC

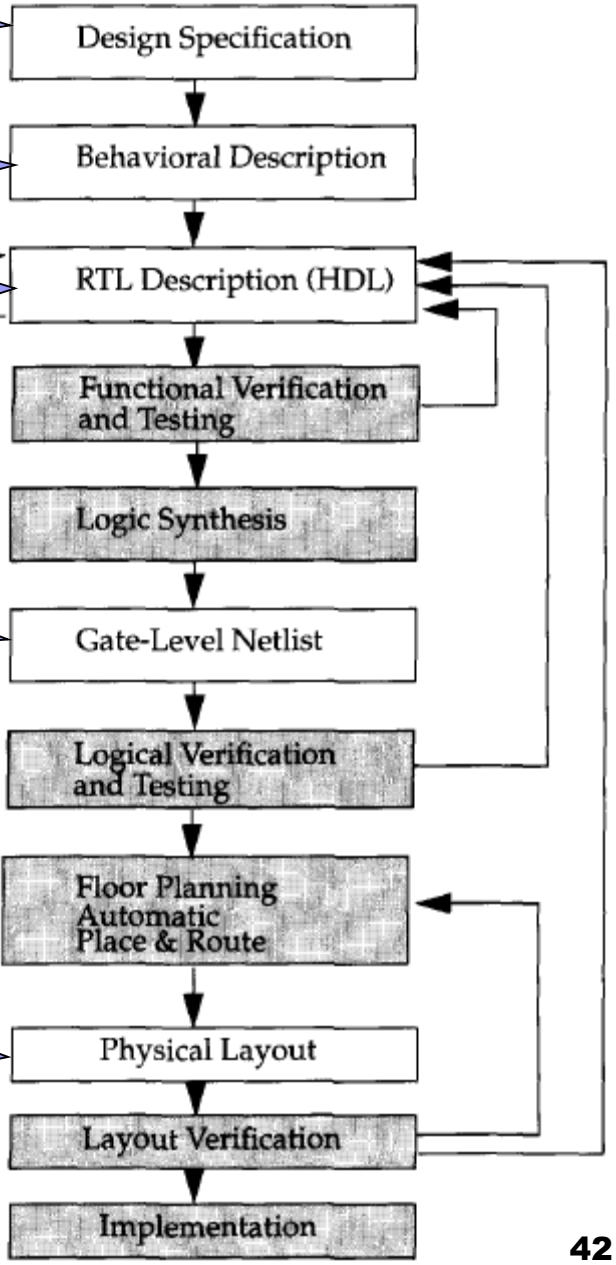
Apstraktno opisuje funkcionalnost, interfejse i opštu arhitekturu digitalnog sistema koji se

Analizira dizajn u smislu funkcionalnosti,

Behavioral opis se konvertuje u RTL opis u HDL-u. Dizajner mora da opiše tzv. *data-flow* koji će implementirati željeno digitalno kolo. Od ove tačke pa nadalje, proces dizajna se obavlja uz pomoć CAD alata.

Alati za logičku sintezu konvertuju RTL opis u *gate-level netlist*-u. Ona predstavlja opis kola u smislu logičkih kapija (*gate*) i veza između njih.

Gate-level netlist-a predstavlja ulaz u alate za *Automatic-Place-and-Route*, koji kreiraju konačni *layout*. On se verifikuje i fabrikuje na čipu.



Tok procesa dizajna - nastavak

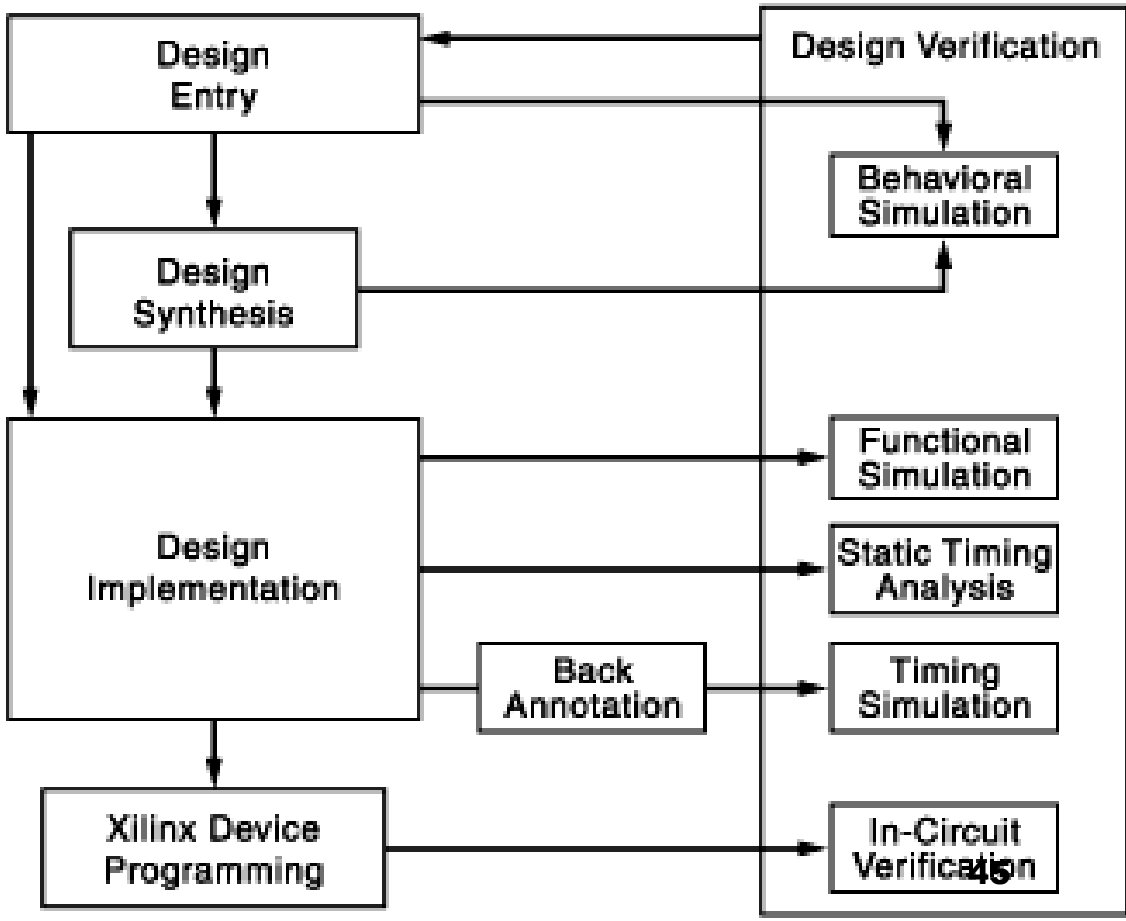
- Dakle, najveća aktivnost dizajnera je kod optimizovanja RTL opisa sistema
- U daljem procesu CAD alati rade najveći dio posla
- Dizajniranje na RTL nivou je značajno skratilo vrijeme potrebno za projektovanje
- U posljednje vrijeme se sve više koriste i alati za *behavioral* sintezu
  - Kreiraju RTL opis iz algoritamskog ili opisa ponašanja kola
  - Dizajn digitalnih kola postaje sličniji računarskom programiranju na jezicima višeg nivoa
- Nota bene: iako CAD alati omogućavaju automatizaciju procesa i skraćuju vrijeme dizajna, dizajner je i dalje onaj ko kontroliše kako će alati uraditi posao.
  - GIGO: Garbage IN => Garbage Out

GIGO: Garbage IN => Garbage Out

- Često je za inženjera veliko iskušenje da što prije počne sa pisanjem koda, pa čak iako nije do kraja sagledao i dobro razumio sve karakteristike finalnog proizvoda: pisanje koda je zabavno a planiranje nije!
- Bez obzira koliko je veliko iskušenje i koliko je zabavno pisati kod, ne počinjati sa kodiranjem dok se u potpunosti ne sagleda kakav treba da bude konačni rezultat!

Tok procesa dizajna (Xilinx)

- Razvoj FPGA-baziranog sistema se može podijeliti na sljedeće faze:
  - dizajn i sinteza sistema
  - implementacija dizajna
  - on-chip verifikacija



Faza dizajna sistema

- Počinje sa fazom ***unosa dizajna*** pomoću:
  - HDL – Hardware Description Language
  - schematic editor-a
- Softverski paketi obezbjeđuju integrisano okruženje za ovu fazu
- Na raspolaganju je širok dijapazon *biblioteka* sa razvijenim komponentama (jednostavni procesori, interfejsi, kontroleri, brojači, dekoderi, ..., sve do logičkih kola)
- Softver omogućava hijerarhijski unos dizajna
- Kada se završi unos dizajna, prelazi se na njegovu ***sintezu*** – proces koji ga transformiše iz HDL oblika u formu najnižih logičkih kola (tzv. RTL – Register Transfer Level)
- Ova faza je nezavisna od ciljne platforme – rezultujući RTL opis može da se smjesti u bilo koji FPGA čip

Faza implementacije

- Uobičajeno se naziva ***Place-And-Route*** faza
- Alati za implementaciju uzimaju ulaznu RTL *netlist*-u i mapiraju logiku unutar raspoloživih resursa konkretnog FPGA čipa
- Nakon toga se traže najbolje lokacije za kreirane blokove dizajna, na osnovu njihovih međusobnih veza i željenih performansi
- Na kraju se izvrši povezivanje i dodjeljuju se I/O pinovi odgovarajućim signalima
- Ova faza je zavisna od ciljne platforme jer se implementacija vrši unutar konkretnog FPGA čipa
- Zato se alati za *Place-And-Route* razvijaju i isporučuju od strane proizvođača FPGA čipova
- Razvijeni su da u potpunosti iskoriste prednosti konkretne FPGA arhitekture i da obezbijede optimalne performanse za dati dizajn
- Rezultat ove faze je konfiguracioni fajl koji se upisuje u FPGA – kao početna verzija dizajna

Faza on-chip verifikacije

- Ova se faza obavlja kad se dizajn upiše u FPGA čip
- Daje mogućnost dizajneru da provjeri dizajn (i traži greške) u realnom radu (okruženju)
- Koriste se posebni kablovi koji se dobijaju da razvojnim kompletima, za povezivanje FPGA sa računarom
- Pomoću njih se mogu pročitati sadržaji internih registara i memorije

"Intellectual Property" blokovi (IP)

- Kompletan dizajn nekih složenijih sistema, razvijen od strane proizvođača FPGA i optimizovan za rad na njihovim FPGA (npr: mikrokontroleri, mikroprocesori, ethernet interfejs, ...)

Napomene

- Verilog liči na C
- Međutim, jezici za opis hardvera nisu programski jezici
- U FPGA ne postoji mikroprocesor koji bi “izvršavao” Verilog kod
- Alati za sintezu na osnovu opisa pripremaju opis na nižem nivou
- Fajl koji se na kraju dobije, upisuje se u FPGA i na taj način se postiže konfiguracija čipa
- Za sintezu se koristi samo podskup čitavog jezika