5 Programirljiva vezja in sistemi

Tehnologija programirljivih vezij omogoča spreminjanje delovanja že izdelanih elektronskih naprav s spremembami v programski in strojni opremi. Najbolj znani so mikroprocesorski sistemi, pri katerih spreminjamo programsko opremo. V programirljivih napravah z oznakami PLA, CPLD in FPGA pa lahko spreminjamo zgradbo elektronskega vezja oz. strojno opremo.

5.1 Programirljive naprave

Pred pojavom programirljivih vezij, so uporabljali pomnilnike ROM za izvedbo poljubni kombinacijskih logičnih funkcij. S pomnilnikom naredimo pravilnostno tabelo logične funkcije, tako so vhodni signali vezani na pomnilniške naslove. Število izhodnih signalov je omejeno s številom podatkovnih bitov oz. širino pomnilniške besede. Pri izvedbi kombinacijskih funkcij s pomnilniki smo zato precej omejeni, sekvenčnih vezij pa ne moremo narediti.

Boolova algebra pravi, da je kombinacijske funkcije možno pretvoriti v obliko, kjer je vsaj izhodni signal zapisan kot vsota (vrata OR) produktov (vrata AND) vhodov oz. negiranih vhodov. Programirljive naprave z oznako PLD (angl. Programmable Logic Device) vsebujejo množico večvhodnih logičnih vrat **AND**, ki so z izhodi vezana na vrata **OR**.
Logično funkcijo za avtomobilski alarm:

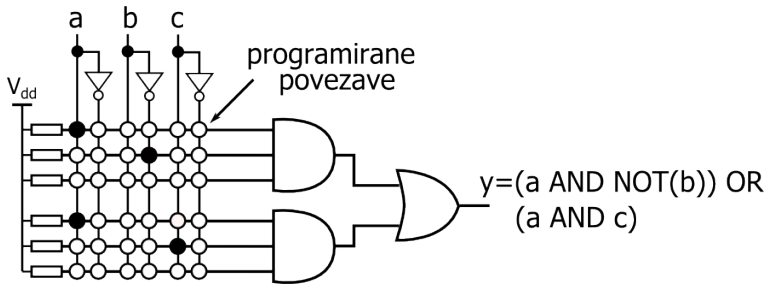
$$alarm = vklop \text{ AND } (NOT(vrata) \text{ OR } gib)$$

pretvorimo v obliko:

$$alarm = (vklop \text{ AND } NOT(vrata)) \text{OR} (vklop \text{ AND } gib)$$

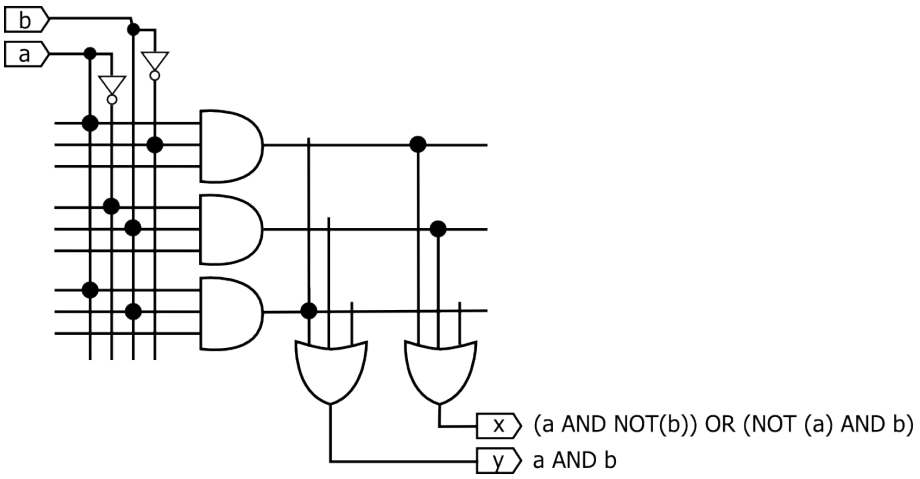
Za izvedbo te funkcije potrebujemo dve vrati AND, ki sta vezani na vrata OR.

Programirljiva matrika PAL (angl. Programmable Array Logic) je sestavljena iz večvhodnih vrat AND, ki jim v postopku programiranja določimo povezave na vhodne oz. negirane vhodne signale. Slika 5.1 prikazuje majhno matriko PAL s katero je narejeno vezje za avtomobilski alarm.



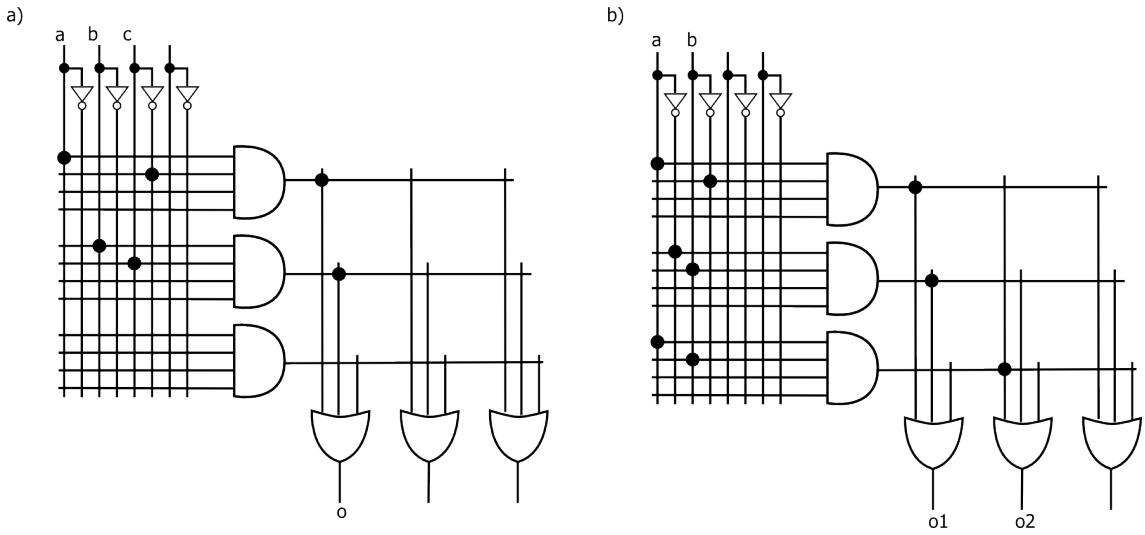
Slika 5.1: Programirljiva matrika PAL

Na shemi matrike so vzpostavljene povezave označene s polnim krožcem, prazen krožec pa označuje prekinjeno programirljivo povezavo. Na teh povezavah so tudi upori proti Vdd, ki zagotavljajo logično 1 kadar so vse povezave prekinjene. Logična 1 na posameznem vhodu vrat AND ne vpliva na funkcijo ostalih vhodov, tako npr. 3-vhodna vrata AND uporabljamo kot 2-vhodna vrata.



Slika 5.2: Programirljiva matrika PLA

Programirljive matrike z oznako PLA (angl. Programmable Logic Array) imajo možnost programiranja povezav na vhodih AND in vhodih vrat OR, kot prikazuje slika 5.2. Matrika na sliki ima 3 produktne člene ter 2 vhodna in 2 izhodna signala. V programirljivih integriranih vezjih so matrike precej večje. Vezja Coolrunner-II proizvajalca Xilinx vsebujejo matrike s 56 produktnimi členi in 16 izhodnih signalov.



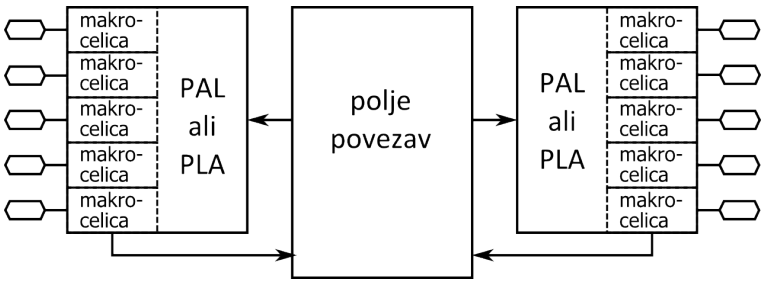
Slika 5.3: Primer izvedbe kombinacijskih gradnikov v programirljivi matriki

Slika 5.3 prikazuje izvedbo kombinacijskih gradkikov v programirljivi matriki. Na podlagi vzpostavljenih povezav na sliki 5.3a napišemo logični izraz: $o = a$ AND NOT(c) OR b AND c, ki predstavlja izbiralnik 2-1, kjer sta a in b vhodna podatkovna signala, c pa izbirni vhod. Zapišimo še logični izraz za gradnik s slike 5.3b, ki predstavlja enobitni seštevalnik:

$$o1 = a \text{ AND NOT}(b) \text{ OR NOT}(a) \text{ AND } b$$
$$o2 = a \text{ AND } b$$

5.1.1 Programirljive naprave - CPLD

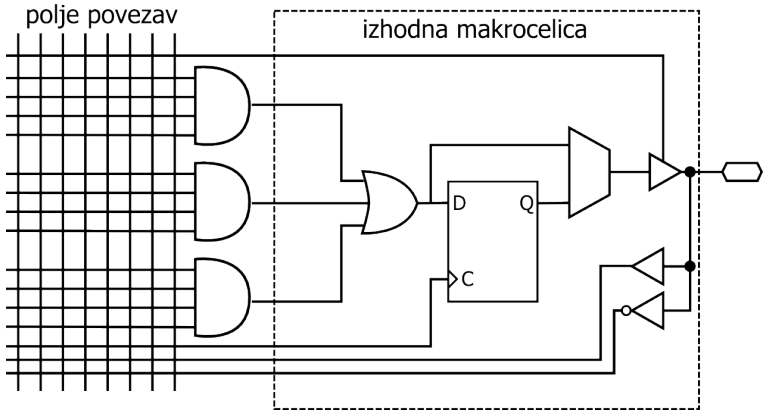
Programirljiva vezja PLD vsebujejo več matrik PLA, ki so med seboj povezane s programirljivim povezovalnim poljem. Vezja z oznako CPLD (angl. Complex Programmable Logic Device) so zgrajena iz osnovnih PLD blokov, ki vsebujejo polje povezav, matriko PAL ali PLA in izhodne makrocelice, kot prikazuje slika 5.4



Slika 5.4: Blokovna shema vezja CPLD.

Kombinacijski izhodi matrike PLA so vezani na makrocelice (MC) znotraj katerih so pomnilni gradniki, s katerimi naredimo sekvenčna vezja. Signali so povezani preko vhodno/izho-

dnih (I/O) celic na zunanje priključke ali pa na povezovalno mrežo (angl. Interconnect Array), ki med seboj povezuje posamezne PLD bloke. Kompleksne gradnike, ki jih ne moremo narediti v eni matriki PLA, razdelimo in naredimo z več matrikami. Preslikavo logične sheme v strukturo PLA opravlja programska oprema, tako da za njihovo uporabo ni potrebno podrobno poznavanje zgradbe vezij PLD.



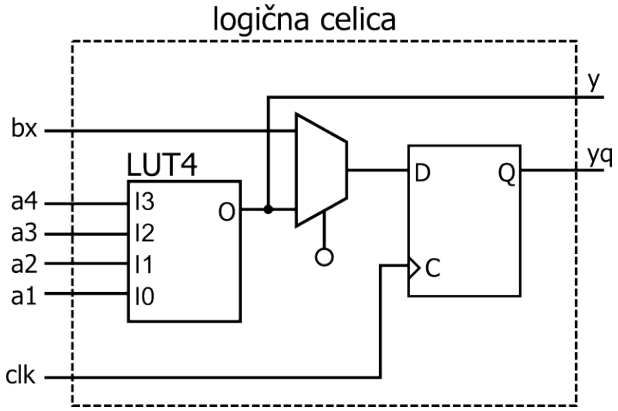
Slika 5.5: Izhodna makrocelica vezja CPLD.

Pri programiranju vezja se določijo povezave v matriki PLA, povezave v makrocelicah in I/O blokih ter povezovalni matriki. S takšno strukturo lahko naredimo poljubna digitalna vezja, omejeni smo le z velikostjo CPLD gradnika. Vezja CPLD omogočajo izdelavo logičnih vezij z nekaj 1000 logičnimi vrati in nekaj 100 flip-flopi, ki delujejo pri frekvencah ure do okoli $200\,\mathrm{MHz}$. Imajo programski pomnilnik vrste FLASH, ki ga lahko večkrat zapišemo in ohrani vsebino ob izklopu napajanja. Naštejmo nekaj prednosti programirljivih vezij CPLD pred ostalimi tehnologijami:

- enostaven razvoj prototipa vezja, ki ga opišemo na računalniku, prevedemo in naložimo na razvojni sistem,
- nizki stroški razvoja v primerjavi z integriranimi vezji, ki bi jih ob vsaki spremembi na novo naredili v tovarni polprevodnikov,
- omogočajo hiter razvoj izdelka, saj je danes zelo pomembno da nov izdelek čim hitreje spravimo na tržišče,
- zmanjšanje komponent na tiskanem vezju - namesto množice logičnih gradnikov v obliki posameznih integriranih vezij lahko vse skupaj naredimo z enim vezjem CPLD (najmanjše med vezji CPLD ima 32 priključkov v $5 \times 5\,\mathrm{mm}$ ohišju),
- proizvajalci naprav ne potrebujejo velikih zalog komponent v življenjski dobi naprave, saj lahko načrt vezja hitro prenesemo v drugo programirljivo vezje, ki se bo obnašalo enako.

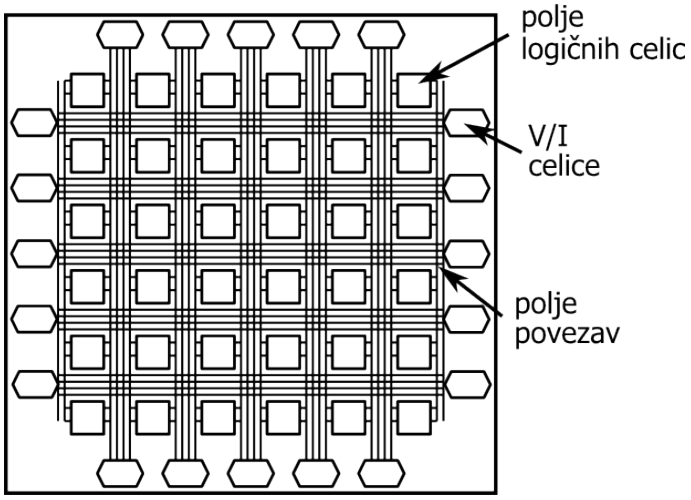
5.1.2 Programirljiva polja vrat - FPGA

Vezja z oznako FPGA (angl. Field Programmable Gate Array) so narejena iz množice programirljivih logičnih celic in polja povezav v katerem lahko med seboj povežemo poljubne celice. Logične celice vsebujejo vpogledne tabele za izvedbo majhnih kombinacijskih gradnikov in flip-flope. Slika 5.6 prikazuje poenostavljeno zgradbo logične celice:



Slika 5.6: Blokovna shema logične celice.

Okoli programirljive matrike so še vhodno/izhodne celice, ki povezujejo signale na zunanje priključke (slika 5.7). Vezja FPGA vsebujeo veliko število celic in največja med njimi omogočajo izdelavo vezij z več kot 10 milijoni logičnih vrat! Poznamo različne tehnološke izvedbe, največ pa jih je v standardnem CMOS procesu, kjer se programski podatki zapišejo v zapahe. Zapah ob izklopu napajanja izgubi shranjeno stanje, zato imamo poleg vezja FPGA na plošči še pomnilnik iz katerega se ob startu naloži vsebina.



Slika 5.7: Blokovna shema vezja FPGA.

Logične celice so univerzalni gradniki programirljivih vezij FPGA, s katerimi lahko nare-

dimo enobitne gradnike za izvedbo več registrskih operacij. Elementarne enobitne gradnike iz kombinacijske logike in flip-flopa D imenujemo *registrske celice*. S povezavo n registrskih celic dobimo n-bitni register z logiko za izvedbo ene ali več mikrooperacij.

Izvedba logičnih operacij

Registrska celica za izvedbo dveh logičnih mikrooperacij:

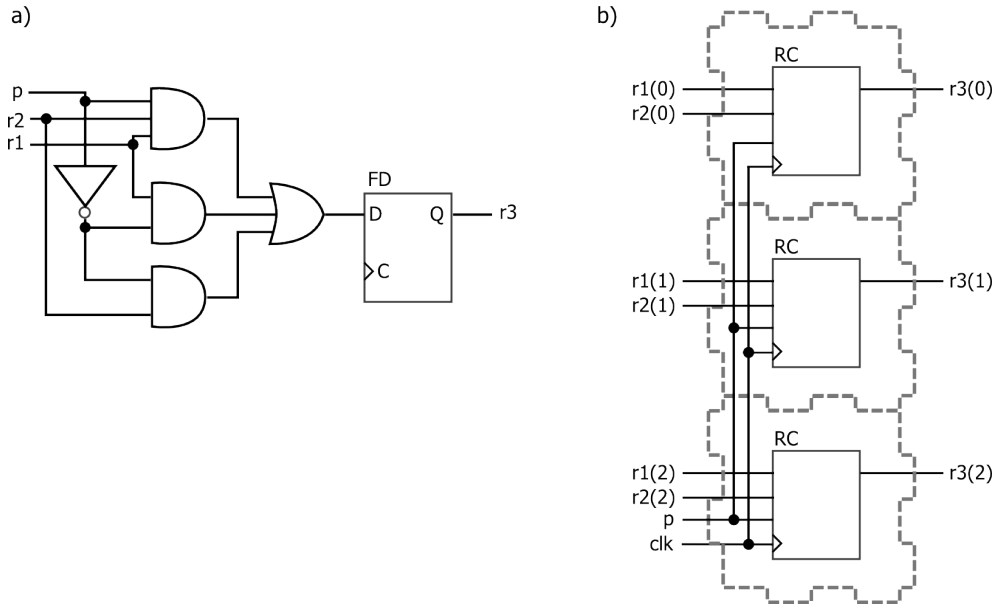
$$\begin{array}{ll} r3 \leftarrow r1 \text{ AND } r2 & | \; p = 1 \\ r3 \leftarrow r1 \text{ OR } r2 & | \; p = 0 \end{array}$$

Parameter p določa vrsto mikrooperacije; kadar je p=1 se izvede logična AND, pri $p = 0$ pa OR. Za načrtovanje registrske celice privzemimo, da so $r1$, $r2$ in $r3$ enobitni signali. Signal $r3$ ima vrednost 1 v primeru ko so $r1$ in $r2$ in p enaki 1 ali pa ko sta $r1$ ali $r2$ enaka 1 pri $p = 0$, kar zapišemo z enačbo:

$$r3 \leftarrow (r1 \text{ AND } r2 \text{ AND } p) \text{ OR } ((r1 \text{ OR } r2) \text{ AND NOT}(p))$$

Enačbo nekoliko preuredimo, da bo primerna za AND - OR izvedbo, kot prikazuje slika 5.8 Z *vzporedno* vezavo registrskih celic dobimo univerzalni gradnik za dve logični mikrooperaciji.

$$r3 \leftarrow (r1 \text{ AND } r2 \text{ AND } p) \text{ OR } (r1 \text{ AND NOT}(p)) \text{ OR } (r2 \text{ AND NOT}(p))$$



Slika 5.8: Shema a) registrske celice in b) gradnika za izvedbo operacij AND ali OR.

Izvedba aritmetičnih operacij

Registrska celica za izvedbo aritmetičnih mikrooperacij z dvema parametroma:

$$\begin{array}{ll} r3 \leftarrow r1 & \mid p1 = 0,\ p0 = 0 \text{ (prenesi } r1) \\ r3 \leftarrow r1 + r2 & \mid p1 = 0,\ p0 = 1 \text{ (seštej)} \\ r3 \leftarrow r1 + (\text{ NOT } r2) & \mid p1 = 1,\ p0 = 0 \\ r3 \leftarrow r1 - 1 & \mid p1 = 1,\ p0 = 1 \text{ (zmanjšaj)} \end{array}$$

Osnova aritmetične registrske celice je seštevalnik z dodatno logiko na vhodu. Seštevalnik ima tudi vhodni prenos ci, s katerim lahko naredimo dodatne operacije:

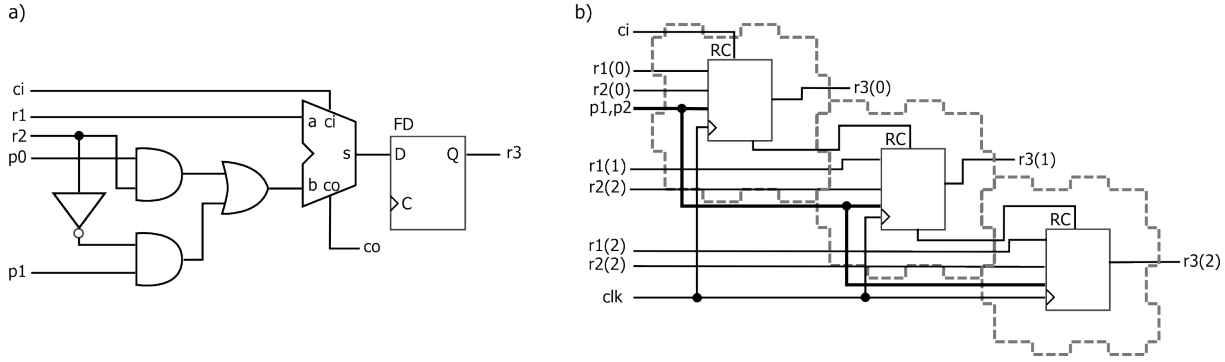
$$\begin{array}{ll} r3 \leftarrow r1 + ci & \mid p1 = 0,\ p0 = 0,\ ci = 1 \text{ (povečaj)} \\ r3 \leftarrow r1 + r2 + ci & \mid p1 = 0,\ p0 = 1,\ ci = 1 \text{ (seštej s prenosom)} \\ r3 \leftarrow r1 + (\text{ NOT } r2) + ci & \mid p1 = 1,\ p0 = 0,\ ci = 1 \text{ (odštej)} \\ r3 \leftarrow r1 - 1 + ci & \mid p1 = 1,\ p0 = 1,\ ci = 1 \text{ (prenesi } r1) \end{array}$$

Logično funkcijo za registrsko celico zapišemo v obliki:

$$r3 \leftarrow r1 + y + ci$$

$$\begin{array}{ll} y = 0 & \mid p1 = 0,\ p0 = 0 \\ y = r2 & \mid p1 = 0,\ p0 = 1 \\ y = (\text{ NOT } r2) & \mid p1 = 1,\ p0 = 0 \\ y = 1 & \mid p1 = 1,\ p0 = 1 \end{array}$$

Uvedli smo nov signal y, ki predstavlja kombinacijsko logiko z izhodom odvisnim od parametrov.
Z razvijanjem logičnih enačb pridemo do sheme registrske celice, ki jo vidimo na sliki 5.9a. Aritmetične celice povezujemo v večbitno enoto na podoben način kot gradimo večbitni seštevalnik; izhodne prenose (co) nižjih bitov *zaporedno* povežemo z vhodnimi prenosi (ci) višjih bitov, kot vidimo na primeru 5.9b.



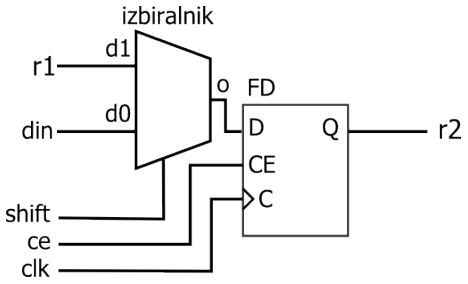
Slika 5.9: Shema a) registrske celice in b) gradnika za izvedbo aritmetičnih operacij.

Izvedba operacij pomikanja

Registrska celica za izvedbo mikrooperacije pomika in prenosa:

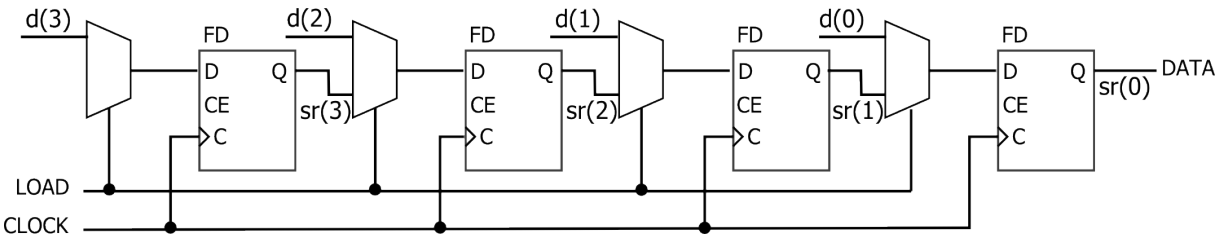
$$\begin{array}{ll} r2 \leftarrow din \ : \ r1[3..1] & \mid shift = 0 \text{ and } ce = 1 \\ r2 \leftarrow r1 & \mid shift = 1 \text{ and } ce = 1 \end{array}$$

Signal $shift$ določa, ali naj se ob pogoju za nalaganje ($ce = 1$) v register $R2$ naloži pomaknjena vrednost ali pa vrednost iz registra $R1$. Registrska celica je sestavljena iz izbiralnika in flip-flopa D, kot prikazuje slika 5.10.



Slika 5.10: Shema registrske celice izvedbo pomika in prenosa.

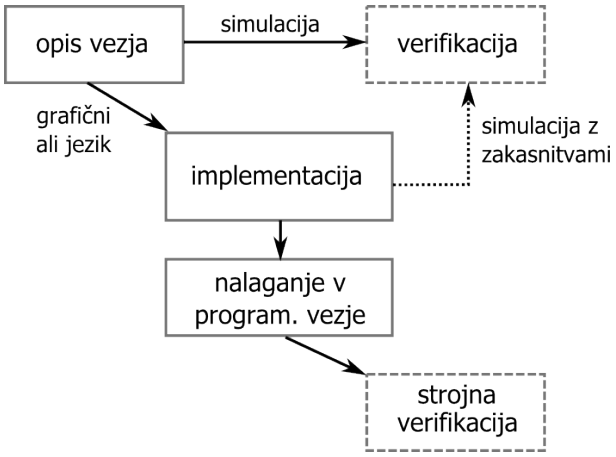
Slika 5.11 prikazuje izvedbo paralelno serijskega pomikalnega registra z registrskimi celicami za pokanje bitov. Takšen register uporabljamo v serijskih vmesnikih za pretvorbo in prenos podatkov.



Slika 5.11: Paralelno serijski 4-bitni pomikalni register.

5.2 Računalniška orodja za programirljive naprave

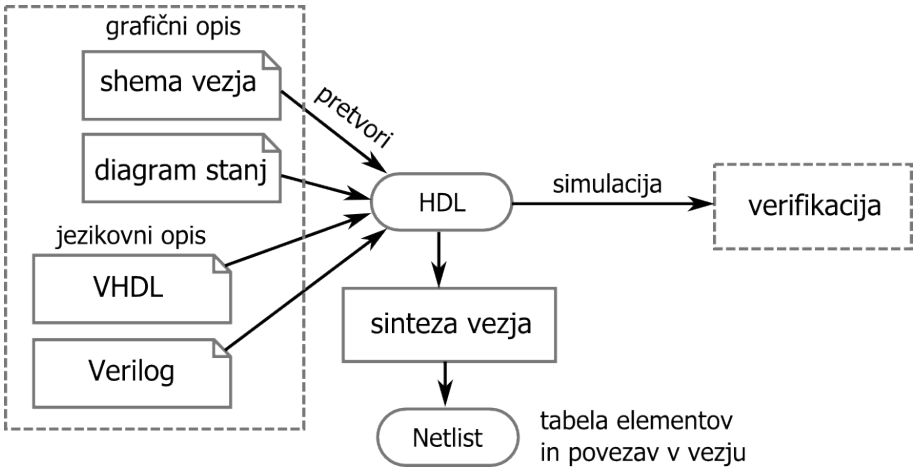
Postopek načrtovanja vezja začnemo z opisom vezja v grafični obliki (npr. shema) ali v nekem jeziku za opis strojne opreme. Delovanje opisanega vezja lahko preverimo oz. verificiramo s simulacijo. Naslednji korak pri načrtovanju s programirljivimi vezji je implementacija, s katero pripravimo datoteko za nalaganje v programirljivo vezje, kot prikazuje slika 5.12. Ko vezje naložimo, preverimo delovanje na strojni opremi (npr. na razvojnem sistemu). Ta postopek imenujemo strojna verifikacija.



Slika 5.12: Osnovni koraki načrtovanja digitalnih vezij.

Delo razvojnega inženirja je predvsem opis vezja in priprava verifikacije. Precej zapleten postopek implementacije vezja, s katero logično vezje prevedemo, prilagodimo in pretvorimo v programsko datoteko, je na srečo avtomatiziran. Programska oprema za računalniško načrtovanje vezij zahteva le nekaj nastavitev, da se implementacija izvede pod želenimi pogoji. Rezultat implementacije lahko ponovno verificiramo s simulacijo, ki tokrat vsebuje tudi ocenjene zakasnitve gradnikov vezja. Postopek nalaganja programske datoteke v programirljivo vezje je odvisen od strojne opreme in izvedbe komunikacije z računalnikom. Običajno ta korak ni zahteven, saj moramo le izbrati pravilne nastavitve in datoteko, ki jo programska oprema naloži v vezje. Postopek strojne verifikacije je odvisen predvsem od razvite aplikacije (nalog, ki jih vezje izvaja). V najbolj preprosti obliki zadošča ročno nastavljanje signalov in vizualni pregled delovanja v zahtevnejših primerih pa potrebujemo posebno merilno opremo.

Programska oprema za računalniško načrtovanje vezij pozna različne načine vnosa vezja, ki jih v grobem razdelimo v grafični in jezikovni opis. Primer grafičnega opisa je shema vezja. Shemo narišemo v grafičnem urejevalniku s postavljanjem elementov iz knjižnice digitalnih gradnikov in risanjem povezav. V tem poglavju bomo spoznali še en grafični opis vezja v obliki diagrama stanj. Jezikovni opis logičnega vezja predstavljajo npr. Boolove enačbe. Jezik za opis strojne opreme HDL (angl. Hardware Description Language) določa pravila takšnega opisa. V današnjem času se uporabljata predvsem jezika VHDL in Verilog, ki omogočata opis vezja z enačbami ali pa v obliki algoritma.



Slika 5.13: Načini opisa digitalnega vezja.

Programska oprema datoteke z grafičnim opisom avtomatsko prevede v jezikovno obliko (HDL), ki je osnova za izvedbo simulacije in ostalih korakov prevajanja, kot prikazuje slika 5.13. Jezikovni opis vezja v koraku sinteze pretvorimo v datoteko, ki vsebuje vse elemente končnega vezja in povezave med njimi (angl. netlist). Ta datoteka je osnova za implementacijo vezja.

5.2.1 Opis števca

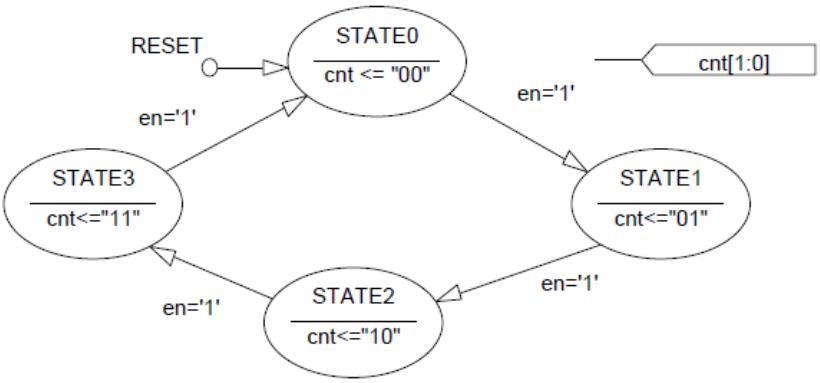
Prikazali bomo različne načine opisa vezja 2-bitnega števca, ki ima vhod za omogočanje en in uro clk. Ob prednji fronti ure naj se vrednost na izhodnem signalu cnt poveča za 1.

Listing 5.1: Opis 2-bitnega števca v jeziku VHDL

```
entity stevec is
    Port ( clk : in  STD_LOGIC;
           en : in  STD_LOGIC;
           cnt : buffer  STD_LOGIC_VECTOR(1 downto 0));
end stevec;
architecture opis of stevec is
begin
 p: process(clk)
 begin
   if rising_edge(clk) and en='1' then
     cnt <= cnt + 1;
   end if;
 end process;
end opis;
```

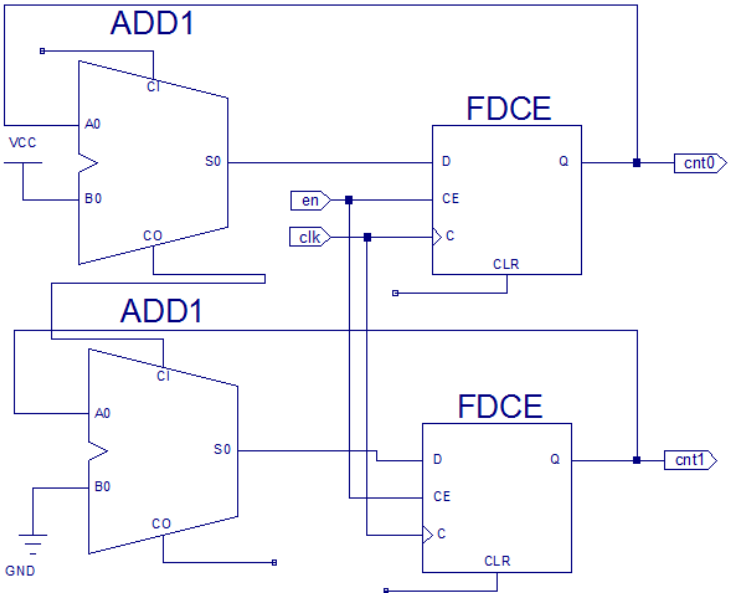
Jezikovni opis števca v jeziku VHDL je sestavljen iz deklaracije priključkov (port) in opisa arhitekture, v kateri je sekvenčni proces za povečevanje vrednosti števca cnt ob naraščajoči fronti ure in pogoju $en = 1$.

Delovanje sekvenčnega vezja, kot je 2-bitni števec, lahko opišemo z diagramom stanj. Diagram prikazuje stanja v registrih vezja in pogoje za prehode. Slika 5.14 predstavlja diagram stanj s štirimi stanji: STATE0, STATE1, STATE2 in STATE3, ki predstavljajo kombinacije na izhodu števca od ”00”do ”11”. Puščice med stanji predstavljajo prehode. Pogoj za prehod v novo stanje je fronta ure in pogoj na puščici en = ’1’. Signal $reset$ določa začetno stanje.



Slika 5.14: Diagram stanj 2-bitnega števca.

Slika 5.15 prikazuje grafično shemo števca, ki je sestavljen iz dveh enobitnih seštevalnikov in flip-flopov. Programska oprema pretvori shemo vezja v strukturni opis v jeziku VHDL ali Verilog. Primer strukturnega opisa v jeziku Verilog prikazuje izpis 5.2.



Slika 5.15: Shematski opis 2-bitnega števca.

Listing 5.2: Strukturni opis sheme števca v jeziku Verilog

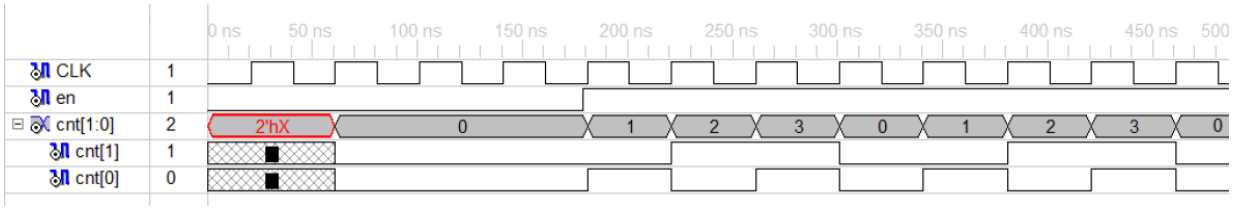
```
module cnt(clk, en, cnt0, cnt1);
   input clk, en;
   output cnt0, cnt1;

   wire XLXN_1, XLXN_2, XLXN_3, XLXN_8, XLXN_9;
   wire cnt0_DUMMY, cnt1_DUMMY;

   assign cnt0 = cnt0_DUMMY;
   assign cnt1 = cnt1_DUMMY;

   ADD1_MXILINX_cnt XLXI_1 (.A0(cnt0_DUMMY),
                            .B0(XLXN_9),
                            .CI(),
                            .CO(XLXN_1),
                            .S0(XLXN_2));

   ADD1_MXILINX_cnt XLXI_2 (.A0(cnt1_DUMMY),
                            .B0(XLXN_8),
                            .CI(XLXN_1),
                            .CO(),
                            .S0(XLXN_3));

   FDCE XLXI_3 (.C(clk),
                .CE(en),
                .CLR(),
                .D(XLXN_2),
                .Q(cnt0_DUMMY));

   FDCE XLXI_4 (.C(clk),
                .CE(en),
                .CLR(),
                .D(XLXN_3),
                .Q(cnt1_DUMMY));

   VCC XLXI_5 (.P(XLXN_9));
   GND XLXI_6 (.G(XLXN_8));
endmodule
```

V strukturnem opisu sheme so najprej definirani vhodni in izhodni signali, ter notranje povezave. Nato je za vsak gradnik vezja določena povezava s signali v vezju.

5.2.2 Verifikacija

Računalniška simulacija modela vezja je osnovni postopek preverjanja oz. verifikacije vezja. Simulacijo pripravimo tako, da določimo časovni potek spreminjanja signalov na vhodu vezja. Časovni potek vhodnih signalov določimo v testni strukturi (angl. test bench) in je zapisan v grafični obliki, v jeziku HDL ali pa v obliki simulacijskih makrojev. Ko je testna struktura pripravljena, poženemo simulator in pregledamo rezultate na časovnem diagramu signalov (angl. waveform).

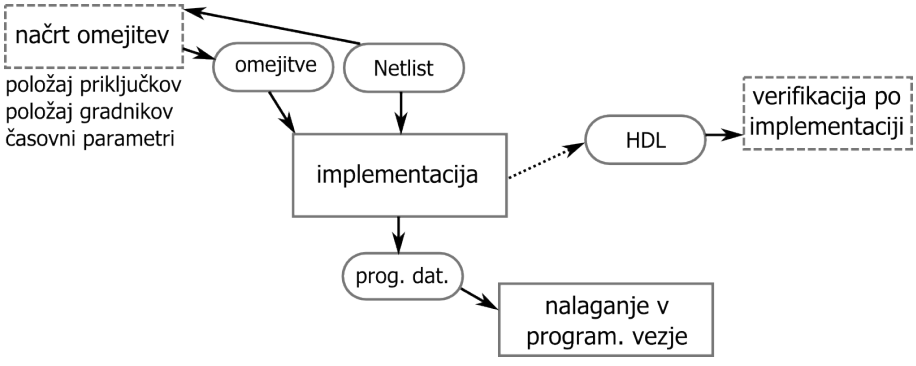
Slika 5.16 prikazuje časovni diagram simulacije 2-bitnega števca. V testni strukturi smo nastavili uro in signal en, opazujemo pa vrednost izhoda cnt. Večbitne signale, kot je $cnt(1:0)$, lahko opazujemo v obliki dvojiških, desetiških ali šestnajstiških številskih vrednosti, možen pa je tudi prikaz posameznih bitov $cnt(1)$ in $cnt(0)$.



Slika 5.16: Rezultat simulacije 2-bitnega števca.

S pregledom časovnega diagrama ugotovimo, ali se vezje obnaša tako, kot smo pričakovali. V testni strukturi lahko tudi vnaprej predpišemo pričakovane vrednosti izhodov ali pa določimo pravila spreminjanja signalov in tako avtomatiziramo postopek verifikacije. Če želimo z verifikacijo res preveriti delovanje vezja, moramo dobro poznati signale, ki jih lahko pričakujemo na vhodu realnega vezja in pripraviti testne strukture za veliko različnih primerov.

5.2.3 Tehnološka preslikava



Slika 5.17: Priprava implementacije vezja.

Pred avtomatsko implementacijo moramo podati zahteve oz. datoteko z omejitvami (angl. constraints file) v kateri določimo položaje zunanjih priključkov, lahko pa tudi notranjih gra-

dnikov ali želene časovne parametre končnega vezja. Rezultat implementacije je programska datoteka, ki jo naložimo v programirljivo vezje. Delni rezultat implementacije je lahko tudi ponoven jezikovni opis vezja (HDL), ki pa sedaj vsebuje še ocenjene zakasnitve in ga uporabimo za bolj natančno verifikacijo vezja po implementaciji.

5.2.4 Nivoji opisa vezij

Programska oprema za računalniško načrtovanje digitalnih vezij omogoča opisovanje vezij na različnih nivojih. Čeprav so vezja fizično narejena iz osnovnih stikalnih gradnikov - transistorjev, digitalna vezja raje obravnavamo na nivoju logičnih vrat ali pa na višjih nivojih, kot so prikazani v tabeli 5.1. Načini opisa so grafični v obliki sheme ali diagrama ali pa jezikovni v obliki enačb ali HDL kode. Orodja za analizo in obravnavo vezij so simulatorji, na višjih nivojih opisa pa tudi orodja, ki izvajajo sintezo opisa na nivo logičnih vrat.

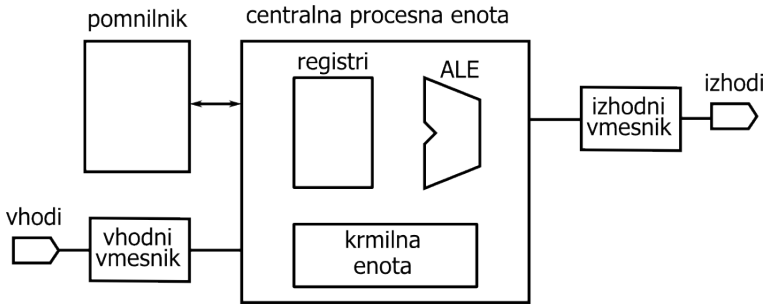
Nivo opisa	**transistorji**	**logična vrata**	**gradniki**	**registri - RTL**
primer	PMOS	AND, OR	izbiralnik	števec, avtomat
način opisa	shema stikal	shema, HDL	blok. shema, HDL	diagram, HDL
orodja	simulator vezij	log. simulator	HDL sim. in sinteza	HDL sim. in sinteza
velikost	20 stikal	20 vrat	10 gradnikov	10 registrov, stanj
št. log. vrat	5	20	200	2000

Tabela 5.1: Nivoji opisa digitalnih sistemov.

Višji nivoji opisa uporabljajo manj podrobne, *poenostavljene* (abstraktne) *modele*, ki omogočajo opis večjih vezij z manj elementi. Tipično število elementov, ki jih opišemo na enem listu je nekaj 10, z uporabo modelov na višjem nivoju pa lahko hitro opišemo vezje z nekaj 1000 logičnimi vrati. Za večja vezja uporabljamo *hierarhičen* način načrtovanja, pri katerem kompleksno vezje razdelimo na več sklopov, ki jih ločeno opišemo in kasneje povežemo med seboj.

5.3 Mikroprocesorski sistemi

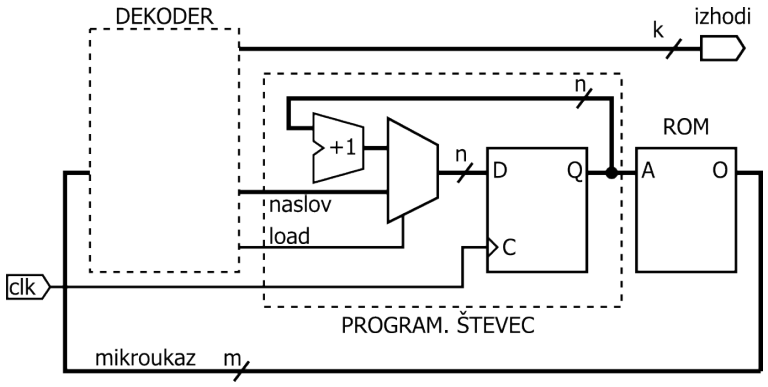
Mikroprocesorski sistemi so univerzalni programirljivi sistemi, ki sprejemajo in obdelujejo digitalne podatke po po programu iz pomnilnika. Izraz *mikroprocesor* označuje miniaturno tehnološko izvedbo procesorja v integriranem vezju. Mikroprocesorski sistem vsebuje najmalj eno centralno procesno enoto (CPE) sestavljeno iz registrov za shranjevanje podatkov, aritmetično-logične enote za obdelavo in krmilne enote, kot prikazuje slika 5.18. Podatki se prenašajo sistem preko vhodnih in izhodnih vmesnikov. Računalništvo se ukvarja s teorijo in orodji s katerimi algoritme razstavimo na aritmetične in logične operacije, ki jih izvajajo mikroprocesorji.



Slika 5.18: Mikroprocesorski digitalni sistem.

5.3.1 Krmilna enota

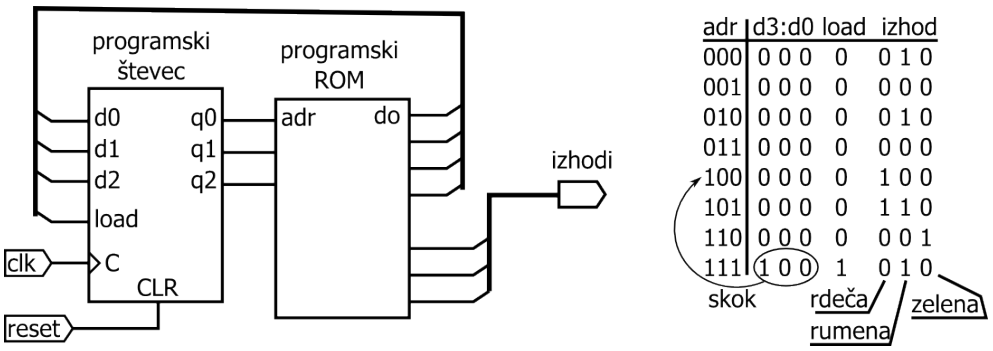
Krmilna enota je sekvenčno vezje, ki nadzira izvajanje ukazov iz pomnilnika in pripravlja krmilne signale za ostale enote. Krmiljenje digitalnih sistemov lahko naredimo s sekvenčnim strojem, vendar le za izvajanje vnaprej določenih sekvenc. *Mikroprogramirana* krmilna enota ali mikrosekvenčnik ima krmilni postopek zapisan v programskem pomnilniku.



Slika 5.19: Mikroprogramirana krmilna enota - mikrosekvenčnik

Mikroprogramirano enoto sestavlja programski števec za določanje naslovov pomnilnika in dekodirnik. Dekodirnik razdeli pomnilniške mikroukaze na signale za krmiljenje programskega števca (load, naslov) in krmilne izhodne signale.

Za ilustracijo si bomo ogledali delovanje enostavnega mikrosekvenčnika, ki je sestavljen iz 4-bitnega programskega števca in pomnilnika vrste ROM. Mikrosekvenčnik ob vsakem urnem ciklu prebere nov mikroukaz iz pomnilnika. Programski števec določa zaporedje mikroukazov: ko je signal load na 0 se berejo mikroukazi po vrsti, pri 1 pa se izvede skok na poljuben naslov.



Slika 5.20: Mikrosekvenčnik s 3-bitnim izhodom in tabela za izvedbo semaforja.

Tabela na sliki 5.20 prikazuje vsebino programskega pomnilnika z osmimi mikroukazi za krmiljenje luči semaforja. Ob resetu se začnejo po vrsti izvajati ukazi, ki najprej povzročijo utripanje rumene luči, potem pa prižiganje luči od rdeče do zelene. Zadnji mikroukaz ima signal load postavljen na 1 in povzroči, da se izvajanje nadaljuje na ukazu z naslovom 100, kar sproži ponoven cikel prižiganja luči.

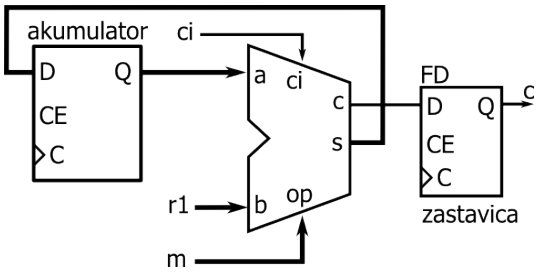
5.3.2 Aritmetično logična enota

Aritmetično logična enota (ALE) je srce mikroprocesorja v katerem se odvija obdelava podatkov. Osem bitni mikroprocesor ima ALE z 8-bitnim podatkovnim vhodom in izhodom, 32-bitni mikroprocesor pa ima 32-bitno ALE. Tipična ALE ima dva podatkovna vhoda a in b, podatkovni izhod s, kontrolni vhod za izbiro operacij op, vhodni ci in izhodni prenos c. Enota izvaja različne mikrooperacije, kot so naprimer:

$s \leftarrow a$	$c \leftarrow 0$	(prenos podatka)
$s \leftarrow a + b$	$c \leftarrow co$	(seštevanje)
$s \leftarrow a + b + ci$	$c \leftarrow co$	(seštevanje s prenosom)
$s \leftarrow a - b$	$c \leftarrow co$	(odštevanje)
$s \leftarrow a - (b + ci)$	$c \leftarrow co$	(odštevanje z izposojo)
$s \leftarrow a$ AND b	$c \leftarrow 0$	(logični in)
$s \leftarrow a$ OR b	$c \leftarrow 0$	(logični ali)
$s \leftarrow 0 \ : \ a[3..1]$	$c \leftarrow a[0]$	(pomik v desno)
$s \leftarrow a[2..0] \ : \ 0$	$c \leftarrow a[3]$	(pomik v levo)

Izhodni prenos uporabimo pri kombiniranju operacij, npr. naredimo seštevanje s prenosom iz prejšnje vsote. Prenos dobimo tudi pri operacijah pomikanja, pri katerih shranimo v prenos tisti bit, ki pri pomikanju vrednosti izpade.

Rezultat operacije ALE shranimo v registru, izhodni prenos pa v flip-flopu imenovanem prenosna zastavica. V praksi ima ALE še druge zastavice: ničelna zastavica se postavi na 1, kadar je rezultat operacije enak 0, negativnostna zastavica pa označuje, da je rezultat negativno število.

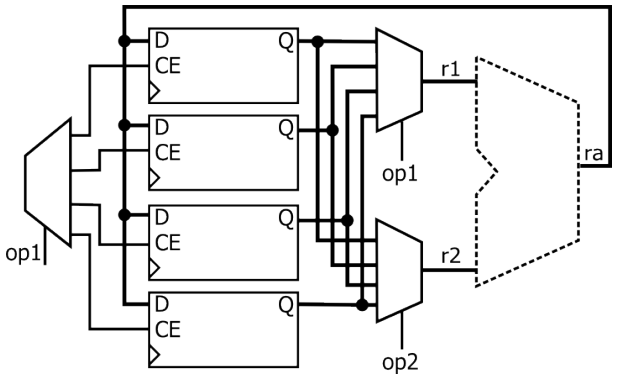


Slika 5.21: Aritmetično logična enota z akumulatorjem.

Aritmetično logična enota je lahko v vezavi s povratno zanko, pri kateri se rezultat vedno shrani v register z imenom akumulator, kot prikazuje slika 5.21. Vezava ALE z akumulatorjem je najbolj preprosta, ni pa najbolj optimalna, ker mora mikroprocesor ob izvajanju algoritma pogosto prenašati vrednost iz in v akumulator.

5.3.3 Registri

Centralna procesna enota vsebuje običajno večje število registrov v katerih shranjuje podatke za izvedbo operacij. Slika 5.22 prikazuje vezavo štirih registrov na aritmetično-logično enoto. Na izhodu registrov sta dva izbiralnika za izbiro prvega operanda $op1$ in drugega operanda $op2$ za aritmetično logično enoto.

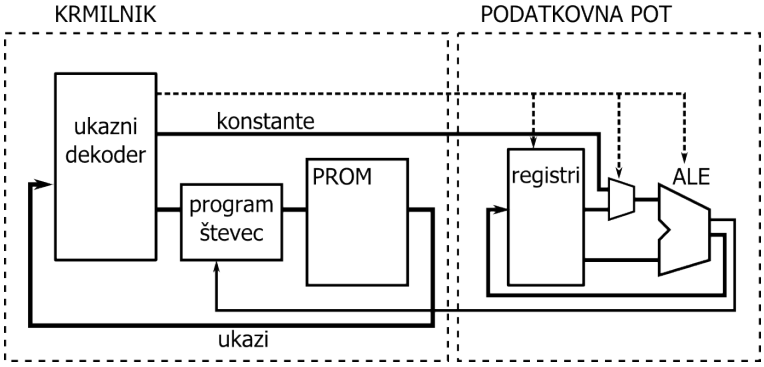


Slika 5.22: Vezava štirih registrov na ALE.

Izhod ALE je vezan na podatkovno vodilo, na katerega so priključeni vsi podatkovni vhodi registrov. Z binarnim dekodirnikom pa določimo v kateri register se bo shranil rezultat operacije. Običajno je to kar eden izmed registrov operandov (npr. $op1$).

5.3.4 Centralna procesna enota

Slika 5.23 prestavlja povezavo med krmilno enoto mikroprocesorja in podatkovno potjo v centralno procesni enoti procesorja Picoblaze. Pomnilnik s programskimi ukazi (PROM) je po svoji funkciji del krmilne enote. Dekodirnik sprejema mikroukaze in nastavlja krmilne signale in konstantne vrednosti za podatkovno pot v CPE. Na podatkovni poti so registri in ALE, zastavice iz ALE pa vplivajo na programski števec ob izvedbi skočnih ukazov.



Slika 5.23: Poenostavljena zgradba centralne procesne enote procesorja Picoblaze.

V centralno procesni enoti se izvajajo ukazi v obliki *strojne kode*, ki določajo vrsto mikrooperacije. Ker je strojna koda za nas težko berljiva jo zapišemo raje v obliki programa v zbirniku (angl. assembler), ki ga prevajalnik prevede v strojno kodo. Primer ukazov v zbirniku:

mikrooperacija	**zbirnik**	**pomen**
$ra \leftarrow rb$	LOAD ra, rb	prenos podatka med registri
$ra \leftarrow k$	LOAD ra, K	prenos konstante K iz pomnilnika
$ra \leftarrow ra + rb$	ADD ra, rb	vsota dveh registrov
$ra \leftarrow ra + k$	ADD ra, K	vsota registra in konstante
$ra \leftarrow ra$ OR rb	OR ra, rb	logični ali med registri
$ra \leftarrow 0 \ : \ ra[3..1]$	SR0 ra	pomik registra v desno

Za komunikacijo z okolico ima procesor vhodne in izhodne enote. Vhodni vmesnik procesorja Picoblaze prenaša podatke iz vrat IN v izbrani register, izhodni vmesnik pa iz registra v vrata OUT z ukazi:

mikrooperacija	**zbirnik**	**pomen**
$ra \leftarrow IN$	IN ra, ID	prenos iz vhodne enote ID
$OUT \leftarrow rb$	OUT rb, ID	prenos iz registra na izhodno enoto ID

Izhodne enote krmilijo vmesnike, npr. analogno-digitalni pretvornik, serijski vmesnik za prenos podatkov po eni povezavi, pulzno-širinski modulator za pulzno krmiljene enote. Do različnih vmesnikov dostopa CPE z izbiro identifikacijskega naslova (ID), ki je del strojnega ukaza IN ali OUT.

5.4 Obdelava digitalnih signalov

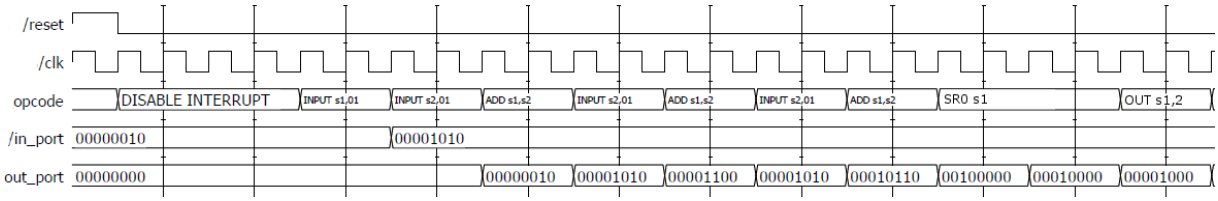
Digitalni sistemi izvajajo obdelavo signalov v digitalni obliki: obdelavo podatkov, digitaliziranega zvoka, slike ipd. Mikroprocesorski sistemi izvajajo obdelavo signalov z zaporednimi ukazi v centralni procesni enoti.

Poglejmo primer algoritma, ki iz vhodnega vmesnika prebere 4 vrednosti, izračuna njihovo povprečje in ga pošlje na izhod. Program prebere prva dva podatka, naredi njuno vsoto, nato pa prebere in prišteje še tretji in četrti podatek. Povprečje dobimo tako, da skupno vsoto delimo s 4, kar storimo z dvema zaporednima pomikoma vrednosti za eno mesto v desno.

Listing 5.3: Program v zbirniku za izračun povprečja štirih vrednosti

```
start:  IN    s1, VHD  ; beri 1. podatek
        IN    s2, VHD  ; 2. podatek
        ADD   s1, s2   ; vsota
        IN    s2, VHD  ; 3. podatek
        ADD   s1, s2   ; vsota
        IN    s2, VHD  ; 4. podatek
        ADD   s1, s2   ; vsota
        SR0   s1       ; deli z 2
        SR0   s1       ; deli z 2
        OUT   s1, IZH  ; na izhod
```

Centralno procesna enota iz pomnilnika nalaga ukaze, jih dekodira in pripravi kontrolne signale, ki izvršijo mikrooperacijo v podatkovni podatkovni poti ter shrani rezultat v izbrani register. Izvajanje ukazov poteka v več korakih, ki jih lahko opazujemo na simulatorju procesorja na sliki 5.24. Iz simulacije razberemo, da je minilo od prvega ukaza za branje podatka do zadnjega ukaza za izpis rezultata 20 urnih ciklov. Procesor Picoblaze potrebuje za vsak ukaz dva urina cikla.



Slika 5.24: Simulacija izvajanja kode na 8-bitnem procesorju Picoblaze.

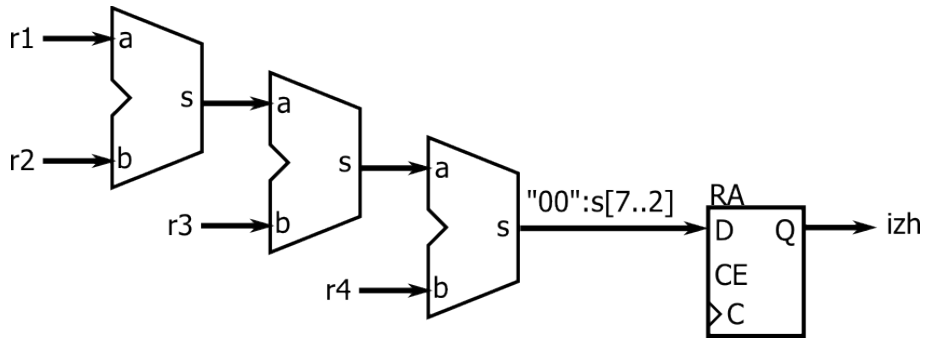
Mikroprocesorski sistem s centralno obdelavo podatkov je zelo fleksibilen, saj lahko njegovo delovanje hitro spremenimo s spremembo programa, ki ga naložimo v pomnilnik. Takšni sistemi se zato vgrajujejo v veliko različnih naprav in predstavljajo pomemben del digitalnih elektronskih vezij. Narejeni so v obliki mikroprocesorjev na integriranih vezjih, ki jih programiramo tako, da naložimo nov program v pomnilnik.

Glavna slabost sistemov s centralno obdelavo podatkov je v tem, da potrebujejo veliko urnih ciklov za izvajanje algoritma, ki ga razstavimo na osnovne mikrooperacije. Frekvence ure vgrajenih mikroprocesorjev so nekaj 10 ali nekaj 100 MHz, kar povsem zadostuje za veliko aplikacij.

Novejša tehnologija integriranih vezij omogoča doseganje višjih frekvenc delovanja, vendar pa z višjo frekvenco narašča tudi poraba energije. V prenosnih, baterijsko napajanih napravah, zaradi tega ne moremo uporabljati najhitrejših in najzmogljivejših procesorjev.

5.4.1 Porazdeljena obdelava signalov

Kadar imamo v vezju zelo hitre signale, kot so npr. hitri podatkovni tok, video signal, moramo uporabiti vezja s porazdeljeno oz. vzporedno obdelavo signalov, saj zaporedna obdelava na CPE ni dovolj hitra. Vzemimo primer izračuna povprečja iz prejšnjega poglavja, ki ga lahko najdemo v aplikaciji za digitalno obdelavo video slike. Vezje naredimo iz treh seštevalnikov in registra, kot prikazuje slika 5.25.



Slika 5.25: Vezje za vzporedni izračun povprečja štirih vrednosti.

Če imamo hkrati na voljo vse štiri vrednosti vhodnih podatkov (signali $r1$, $r2$, $r3$ in $r4$), dobimo rezultat v enem urnem ciklu. Tudi v primeru, ko potrebujemo več ciklov, da na vhode pripeljemo vse vrednosti, je vezje hitrejše od mikroprocesorja, saj ne potrebujemo ciklov za shranjevanje vmesnih rezultatov, pa tudi pomikanje je narejeno le z ustreznimi povezavami. Takšno vezje lahko pri manjši frekvenci ure obdela hiter tok video podatkov v realnem času.

Slabost porazdeljene obdelave signalov je, da potrebujemo več računskih enot in s tem večje vezje, ter da moramo za vsako spremembo algoritma razviti novo vezje. Kadar uporabljamo programirljive digitalne sisteme z vezji CPLD ali FPGA, je tudi takšna rešitev dovolj prilagodljiva, ker omogoča poljubno spreminjanje in nalaganje novega vezja. Glavna slabost je višja cena vezja v primerjavi z mikroprocesorji in omejitev velikosti porazdeljenega vezja, ki pa jo nekoliko kompenzira hiter razvoj tehnologije integriranih vezij.