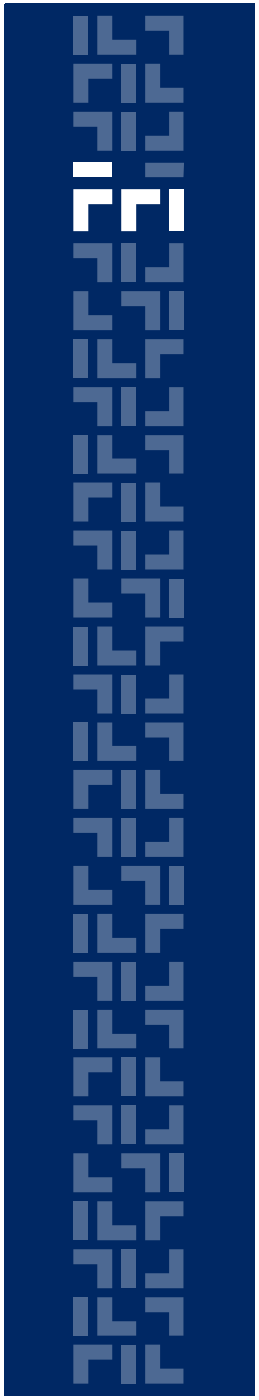
Univerza v Ljubljani
Fakulteta
za računalništvo
in informatiko

Metode logičnega snovanja
Programabilne logične naprave in VHDL

Miha Moškon



Programabilne logične naprave

Programabilna logična vezja

V času proizvodnje je njihova logična funkcija nedefinirana.

Uporabnik jih prilagodi svojim potrebam s programiranjem.

2 tipa programiranja:

- neponovljivo,
- ponovljivo.

Programabilna logična vezja

Delimo jih tudi glede na notranjo organizacijo:

- PLD ali programabilne logične naprave (sestavljene so iz polja AND vrat in iz polja OR vrat),
- CPLD ali kompleksna PLD (sestavljena so iz PLD celic in iz povezovalne mreže),
- FPGA (sestavljena so iz nastavljivih logičnih blokov (CLB), povezovalnih kanalov in vhodno/izhodnih blokov).

Primerjava z ostalimi izvedbami

Komercialni logični čipi:

- potrebujemo veliko različnih enot (prostor),
- ob spremembi funkcionalnosti novo tiskano vezje,
- nizka cena.

Čipi po naročilu:

- so optimalno prirejeni za aplikacijo,
- majhna cena na čip (za velike serije),
- dolgotrajen in drag razvoj.

Programabilne logične naprave

Sestavljena so iz polja AND vrat in iz polja OR vrat.

Delimo jih glede na zmožnost programiranja posameznega polja:

- ROM (programabilno OR polje, fiksno AND polje),
- PAL (programabilno AND polje, fiksno OR polje),
- PLA (programabilno AND in OR polje).

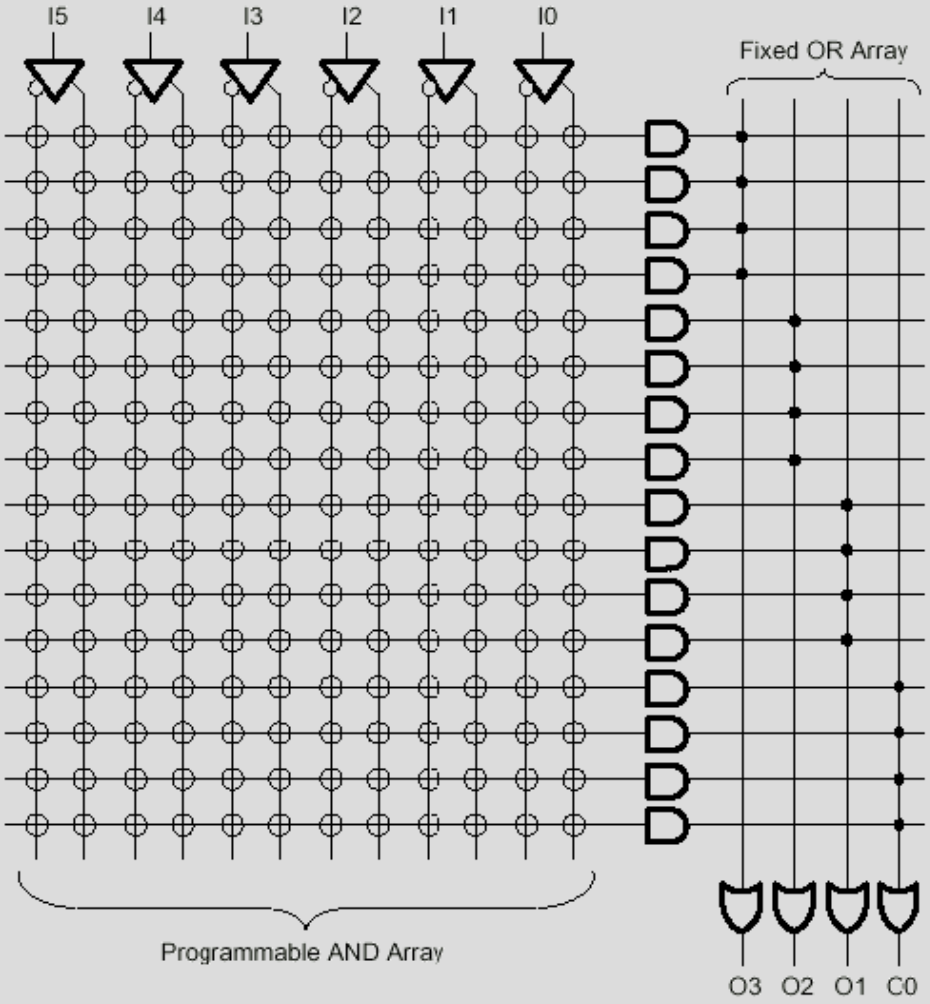
GAL vezja

Vezja tipa GAL spadajo v skupino PAL vezij – programabilno polje AND vrat.

GAL vezja spadajo v skupino reprogramabilnih vezij (GAL 16V8 – EECMOS, 20 let ohranja stanje, do 100 brisanj/pisanj)

Z njimi lahko realiziramo preklopne funkcije v disjunktivni normalni obliki (DNO)

GAL vezja

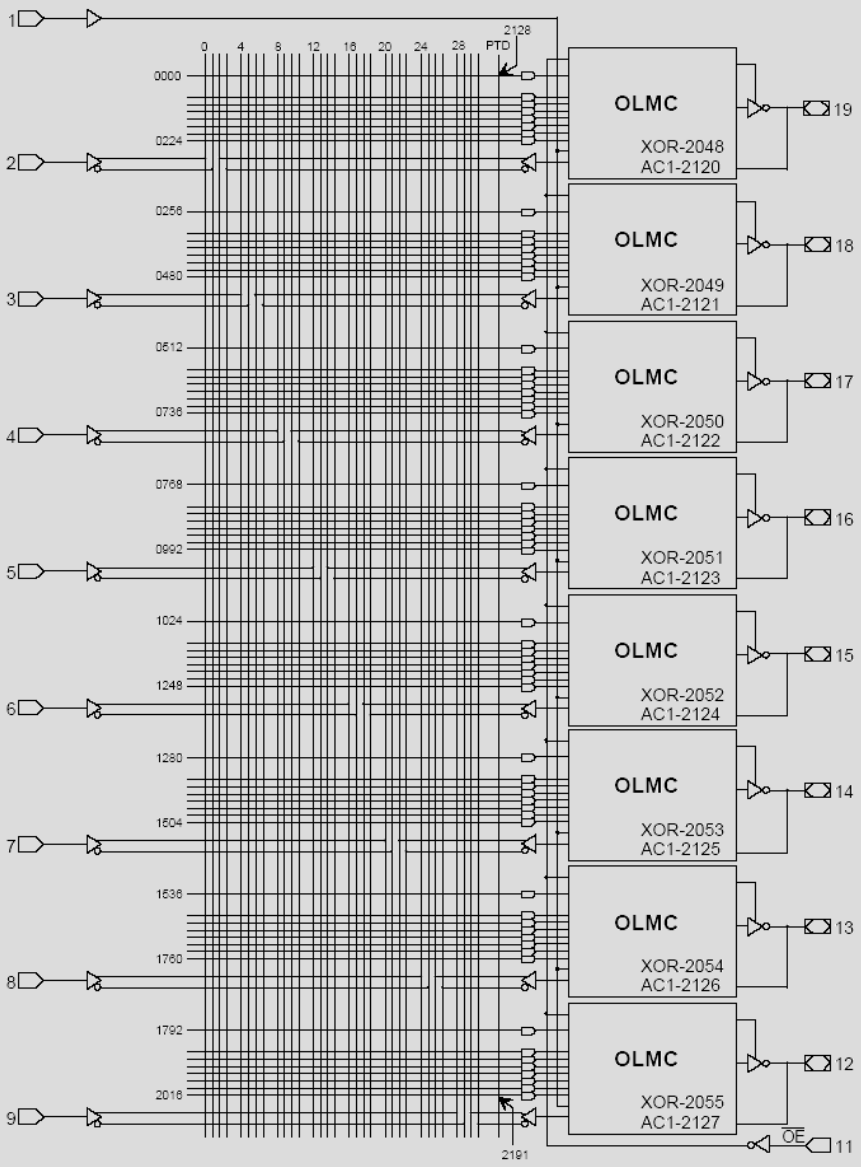

I5
I4
I3
I2
I1
I0
Fixed OR Array
Programmable AND Array
O3
O2
O1
C0

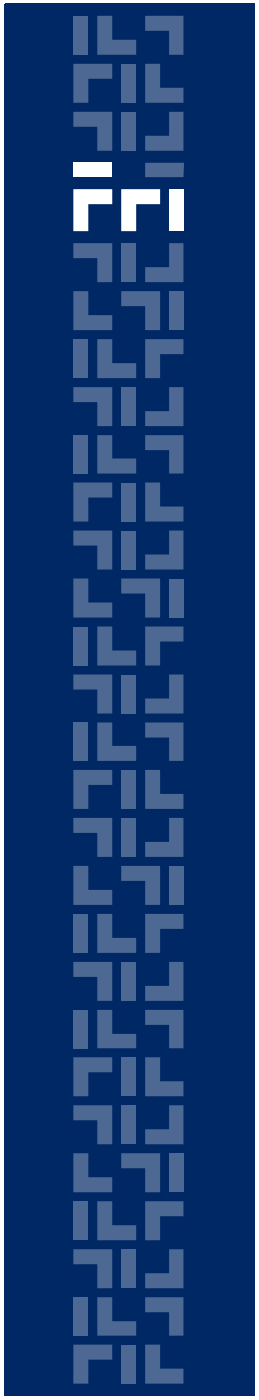
GAL 16V8

Do 16 vhodov, urin signal in signal za aktiviranje izhodov.

8 izhodnih logičnih makrocelic (OLMC).

Programiramo tako logiko, kot tudi makrocelice.





VHDL

VHDL

Very High Speed Integrated Circuit **H**ardware **D**escription **L**anguage

- industrijski standard za opisovanje, modeliranje in sintezo digitalnih vezji in sistemov (vsebuje tudi ukaze, ki jih ne moremo sintetizirati – časovne zakasnitve),
- dovolj je da poznamo obnašanje sistema, ki ga programiramo,
- programiranje je neodvisno od ciljnega vezja (paziti moramo le na omejitve le-tega – npr. število vhodov),
- omogoča simuliranje delovanja vezja pred sintezo,
- stavki se izvajajo **paralelno** (z izjemo procesov),
- HDL jeziki: VHDL, ABEL, Verilog.

VHDL sintaksa

Imena spremenljivk se morajo začeti s črko, ne smejo se končati s podčrtajem, zaporedoma le en podčrtaj.

Case Insensitive.

Komentar: “--” na začetku vrstice.

Struktura programa

Vsak program je sestavljen iz treh delov:

- **knjižnice**: v njih so definirani podatkovni tipi, funkcije, komponente,...
- **entitete**: vsebuje definicijo priključkov vezij (definira povezave z zunanjim svetom),
- **arhitekture**: opisuje delovanje vezja na tri načine:
  - vzporedno (stavki se izvajajo paralelno),
  - zaporedno (stavki se izvajajo zaporedno – procesi),
  - modularno (uporabimo vnaprej sprogramirane komponente).

Struktura programa

```
-- definicija knjiznic
library ime_knjiznice
use     ime_knjiznice.ime_komponent.all

-- entiteta
entity ime_entitete is
   port (ime_signala1: nacin_delovanja podatkovni_tip;...
         ime_signalaN: nacin_delovanja podatkovni tip);
end ime_entitete;

-- arhitektura
architecture ime_arhitekture of ime_entitete is
   -- definicije tipov, pomoznih signalov in konstant
   signal pomozni_signal1: podatkovni_tip;
begin
   -- opis delovanja vezja
end ime_arhitekture;
```

Knjižnice

Če želimo uporabiti določen podatkovni tip moramo v naš program vključiti knjižnico, v kateri je definiran.

Primer uporabe podatkovnega tipa `std_logic`, ki je definiran v knjižnici IEEE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

Entiteta

Služi definiciji naprave, ki jo bo realizirala arhitektura.

Definiranje priključkov:

- način delovanja: `in` (vhod), `out` (izhod), `buffer` (povratni izhod), `in/out`,
- podatkovni tipi:
  - `bit`: `'0'` ali `'1'`,
  - `std_logic`: `'0'`, `'1'`, `'U'`, `'Z'`,...
  - `std_logic_vector`: navedemo območje: `(0 to 5);` `(6 downto 0);` referenca z `ime_vektorja(i)`.

Arhitektura

Vzporedno opisovanje delovanja vezja:

- prireditev konstante:
  - za navaden signal: `ime_signala <= '1';`
  - za vektor: `ime_signala <= "00110";`
- prireditev z logičnimi enačbami:
  - `ime_signala <= logicna_enacba;`
  - na desni strani enačbe so **operatorji** in **spremenljivke** (signali),
  - operatorji: AND, OR, NOT, XOR,... (vsi razen NOT imajo enako prioriteto).
  - primer: `izhod <= vhod1 OR (vhod2 AND NOT vhod3 );`

Arhitektura

Vzporedno opisovanje:

- stavek `when-else`:
  - ```
    ime_signala <= log_enacba1 when pogoj1 else
                   log_enacba2 when pogoj2 else
                   ...
                   log_enacbaN;
    ```
  - pogoji so sestavljeni iz spremenljivk (signalov) in primerjalnih operatorjev `(<, =, >, <=, >=, /=)`.
  - primer: ```
    izhod <= '0' when vhod1 = '0' else
             '1' when vhod2 = '0' else
             'Z';
    ```

Primer programa MUX 2/1

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity MUX is
   port (I: in std_logic_vector(1 downto 0); -- podat. vhoda
         A: in std_logic; -- adresni vhod
         O: out std_logic); -- izhod MUX-a
end MUX;

architecture behavioral of MUX is
begin
   -- jedro programa (glej naslednjo prosojnico)
end behavioral;
```

Primer programa MUX 2/1

Delovanje MUX 2/1 lahko opišemo na več načinov:

- z logično enačbo:

```
O <= (I(0) AND NOT(A)) OR (I(1) AND A);
```

- s stavkom `when-else`:

```
O <= I(0) WHEN A = '0' ELSE
     I(1);
```

- s stavkom `with-select`:

  Prihodnjič!

  Sinteza (končno vezje) je v vseh treh primerih enaka!

Postopek izdelave logičnega vezja

- **definiranje problema** (določimo željeno preklopno funkcijo),
- **izbiranje programabilne enote** (glede na naše zahteve),
- **zapis preklopne funkcije** (programiranje v VHDL-u, simuliranje in zapis v JEDEC datoteko),
- **programiranje** (zapis v JEDEC datoteki prenesemo v programabilno enoto),
- **testiranje.**

Delovno okolje

Za programiranje uporabljamo Xilinx ISE WebPack paket, ki je na voljo na Xilinxovi domači strani.

Že sprogramirane komponente v VHDL-u (npr. MicroBlaze Soft Processor Core).

Xilinx ISE WebPack

Postopek izdelave projekta:

- File → New Project
- Top-Level Source Type: **HDL**
- Device properties:
  - Family: Spartan3E
  - Device: XC3S500E
  - Package: FG320
  - Speed: -4

Xilinx ISE WebPack

Dodajanje VHDL datotek:

- Project → New Source → VHDL modul

Prevajanje projekta:

- Processes → Synthesize – XST

Simuliranje

Avtomatsko generiranje testnih signalov (s tbw datoteko):

- Project → New Source → Test Bench WaveForm,
- kombinatorno ali sekvenčno vezje,
- nastavimo želene potek signalov,
- izpis poteka simulacije – izberemo:
  - Sources for: Behavioral simulation (levo zgoraj), kjer mora biti izbrana naša datoteka s testnimi signali,
  - Processes (levo spodaj),
  - Simulate Behavioral Model (levo na sredini).

Simuliranje

Ročno generiranje testnih signalov (z vhd datoteka):

- Project → New Source → VHDL Test Bench
- določanje poteka signalov v dveh procesih (en proces za urin signal, en proces za ostale signale)
- izpis poteka simulacije – izberemo:
  - Sources for: Behavioral simulation (levo zgoraj), kjer mora biti izbrana naša datoteka s testnimi signali,
  - Processes (levo spodaj),
  - Simulate Behavioral Model (levo na sredini).
- Opozorilo za uporabo simulatorja na lastnem računalniku: v Windows 7 je potrebno omogočiti WebClient Service (Start → services.msc → Enable WebClient).

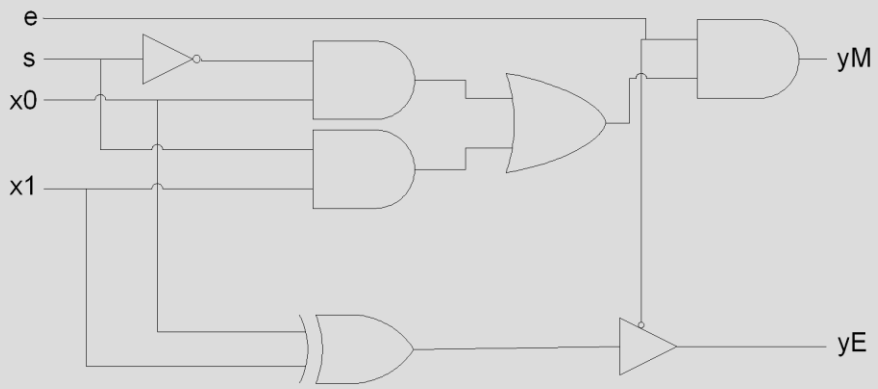
Xilinx ISE WebPack

Sinteza in programiranje vezja:

- Sources for Synthesis/Implementation,
- Generate Programming File → Configure Device (iMPACT),
- Configure devices using Boundary-Scan (JTAG) – auto,
- pri xc3s500e izberemo *.bit datoteko, v drugih dveh primerih damo Bypass,
- desni gumb na napravo xc3s500e → Program → OK.

Nalogi

1. Realizirajte logično vezje na sliki in ga simulirajte. Pri tem izhod yM napišite z logičnimi enačbami, yE pa s stavkom `when-else`.



Nalogi

2. Naredite dekodirnik 2/4, ki je podan s tabelo in ga simulirajte (prvič ga napišite z logičnimi enačbami, drugič pa z uporabo stavka `when-else`). Uporabite vektorske tipe.

A1	A0	Z3	Z2	Z1	Z0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0