Логичко пројектовање

Предавање 7/10

Меморије

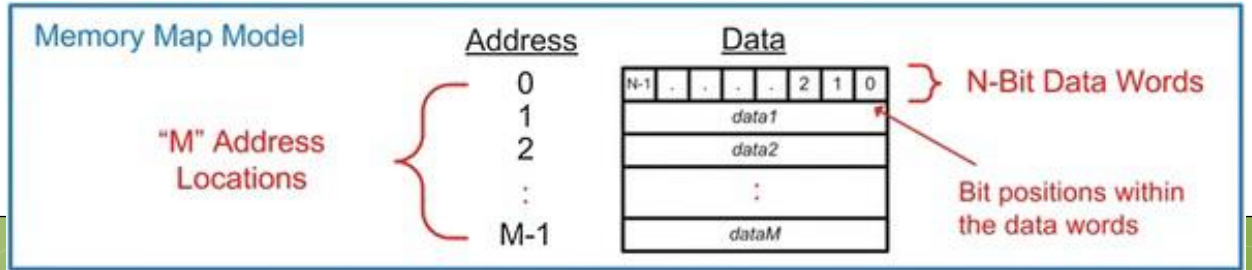
Меморије

Увод

- Термин меморија се користи за описивање система са могућношћу складиштења дигиталних података.
- Ове врсте система складиште дигиталне информације користе транзисторе, осигураче и/или кондензаторе на једном супстрату полупроводника.
- Меморија се такође може имплементирати помоћу других технологијс осим полупроводника.
- Диск јединице чувају информације променом поларитета магнетних поља на кружном супстрату. Два магнетна поларитета (север и југ) се користе за представљање различитих логичких вредности (тј. 0 или 1).
- Оптички дискови употребљавају ласере за спаљивање удубљења у рефлектујуће супстрате. Бинарна информација је представљено светлошћу која се одбија (нема удубљења) или се не рефлектује (присутно удубљење).
- Полупроводничка меморија нема покретне делове, па се назива SSD (solid-state memory) и може да садржи више информација по јединици површине од диск меморије.
- Без обзира на технологију која се користи за складиштење бинарних података, сва меморија има заједничке атрибуте и терминологију.
-

Модел мапирања меморије

- Информације складиштене у меморији називају се подацима.
- Када је информација смешта у меморију, та операција се назива "упис". Када се информације преузимају из меморија, та операција се назива "читање". Да би се приступило подацима у меморији, користити се "адреса".
- Док се подацима може приступити и као појединачним битовима, како би се смањио број потребних адресних локација, подаци се обично групишу у N-битне речи.
- Ако меморијски систем има N=8, то значи да је на свакој адреси складиштено 8 битова података.
- Број адресних локација се описује помоћу променљиве M. Укупна величина меморије се обично наводи речима „M×N". На пример, ако имамо меморијски систем 16×8, што значи да постоји 16 адресних локација, свака способна да складишти 1 бајт података. Ова меморија би имала капацитет од 16×8 = 128 бита.
- Пошто је адреса имплементирана као бинарни код, број линија у адресној магистрали (n) ће диктирати број адресних локација које меморијски систем има ($M = 2^n$).
- Следећа слика приказује како се подаци смештају у меморији. Ово се назива модел мапирања меморије.
-



Класификација меморија (1)

- 3. врсте класификације:
  - Volatile – Nonvolatile меморија,
  - ROM – R/W меморија,
  - RAM – SAM мемрија.

- Меморија је класификована у две категорије у зависности од тога да ли може да складишти информације када је напајање искључено или не.
- Термин постојана (nonvolatile) се користи за опис меморије која чува информације и када се искључи напајање, док се термин привремена (volatile) користи за описивање меморије која губи информације када се напајање уклони.
- Историјски гледано, привремена меморија је у стању да ради већим брзинама у поређењу са постојаном меморијом, па се користи као примарни механизам за складиштење док ради дигитални систем.
- Постојана меморија је неопходна да би се чувале потребне операције за рад дигиталног система, као што су покретање, упутства, оперативни систем и апликације.

Класификација меморија (2)

- Меморија се такође може класификовати у две категорије с обзиром на то како се подацима приступа.
- Меморија само за читање (ROM- Read Access Memory) је меморија у коју се не може уписивати током нормалног рада система.
- Ова врста меморије је корисна за чување критичних системских информација или програма који се не мењају док у току рада система.
- Меморија за читање/писање (R/W – Read/Write Memory) се односи на меморију у која се може читати и у коју се може уписивати у току нормалног рада система и користи се за чување привремених података и променљивих.

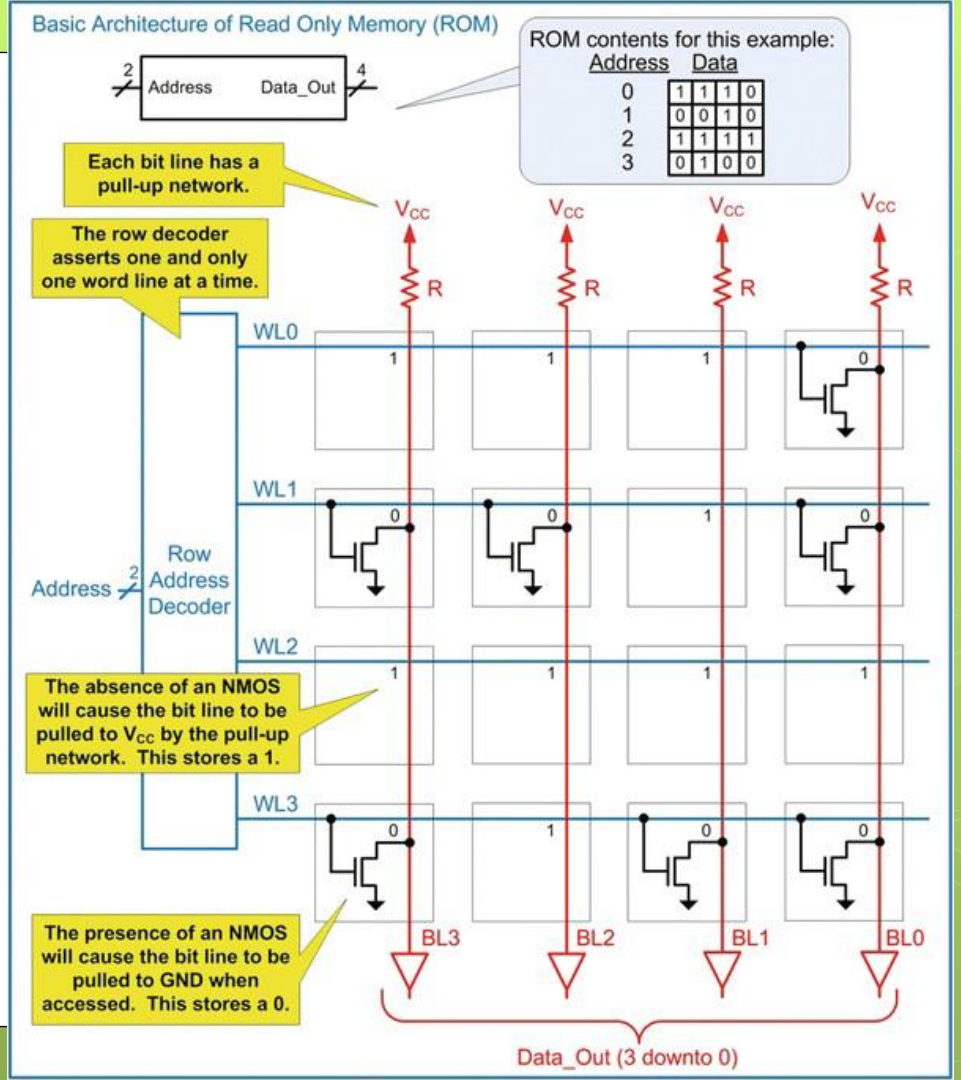
- Меморија са случајним приступом (RAM – Random Access Memory) описује меморију у којој се било којој локацији у систему може приступити у било ком тренутку.
- Супротно овоме меморија (SAM – Sequential Access Memory) има секвенцијални приступ, у којој нису одмах доступне све локације адреса. Пример система меморије са секвенцијалним приступом је магнетна трака. Да би се приступило жељеној адреси у овом систему, калем траке се мора окретати до адресе где се налазе подаци.
- Већина полупроводничке меморије у модерним системима има случајни приступ.

- Термини RAM и ROM су усвојени, донекле нетачно, да опишу групе меморија са различитим понашањем. Док термин ROM технички описује систем у који не може да се врши упис, попримио је додатну асоцијацију да буде термин за описивање трајне меморије. Док термин RAM технички описује како се подацима приступа, он је попримио додатну асоцијацију да буде термин за описивање меморије за читање/писање.

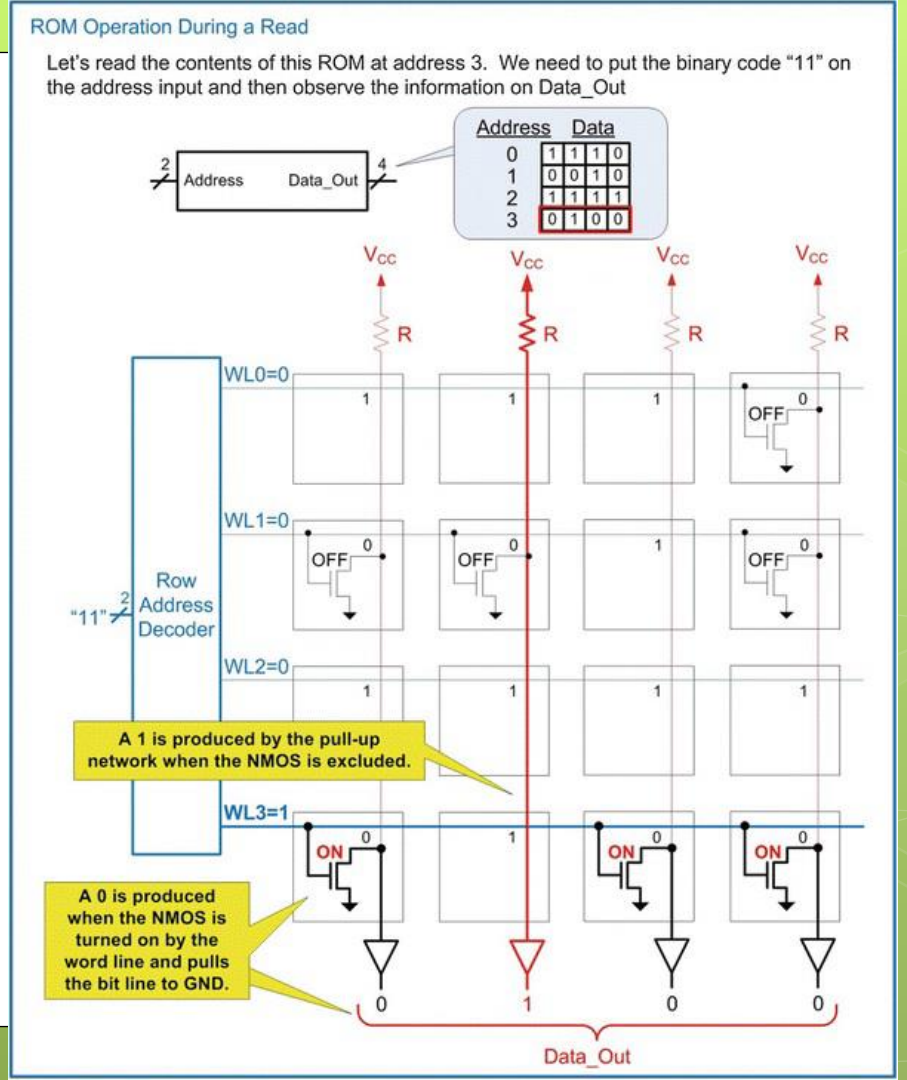
ROM архитектура (1)

- Једна од најчешћих технологија која се користе за складиштење дигиталних информација.
- Адресни декодер се користи за приступ појединачним речима података унутар меморијског система. Декодер адресе издваја једну и само једну линију речи (WL) за сваку јединствену бинарну адресу која се доводи на његов улаз. Ова операција је идентична "binary-to-one-hot" декодеру.
- За n-битну адресу, декодер може приступити 2^n или M речи у меморији.
- Редови речи се историјски издвајају хоризонтално преко меморијског низа и често се називају линије редова, а декодер речи често се назива декодер реда.
- Линије битова (BL) се издвајају вертикално у односу на линије речи како би се обезбедили појединачни приступи меморији битова на пресеку линија бита и речи. Ове линије обично пролазе вертикално кроз меморијски низ; тако се често називају колоне линија. Излаз меморијског система (тј. Data_Out) се добија помоћу адресе и затим читање речи из бафера са бит линијама.
- Када систем обезбеди појединачни приступ биту, реду или приступ више речи података које деле линију реда, декодер колоне се користи за рутирање одговарајуће битне линије до излазног порта података.
- У низу ROM-ова, свака линија битова садржи pull-up мрежу ка V_{CC}-у. Ово пружа могућност складиштења логичке 1 на свим локацијама унутар низа.
- Ако је логичка 0 пожељна на одређеној локацији, користи се NMOS транзистор. Гејт NMOS-а је повезан са одговарајућом линијом речи и дрејн NMOS-а је повезан са бит-ском линијом. Приликом читања, један ред речи се издваја и укључује NMOS транзистор. Ово повлачи битну линију на GND и производи логичку 0 на излазу.
- Када је NMOS транзистор искључен, бит линија остаје на логици 1 због pull-up мреже.
- На следећој слици је приказана основна архитектура ROM-а.

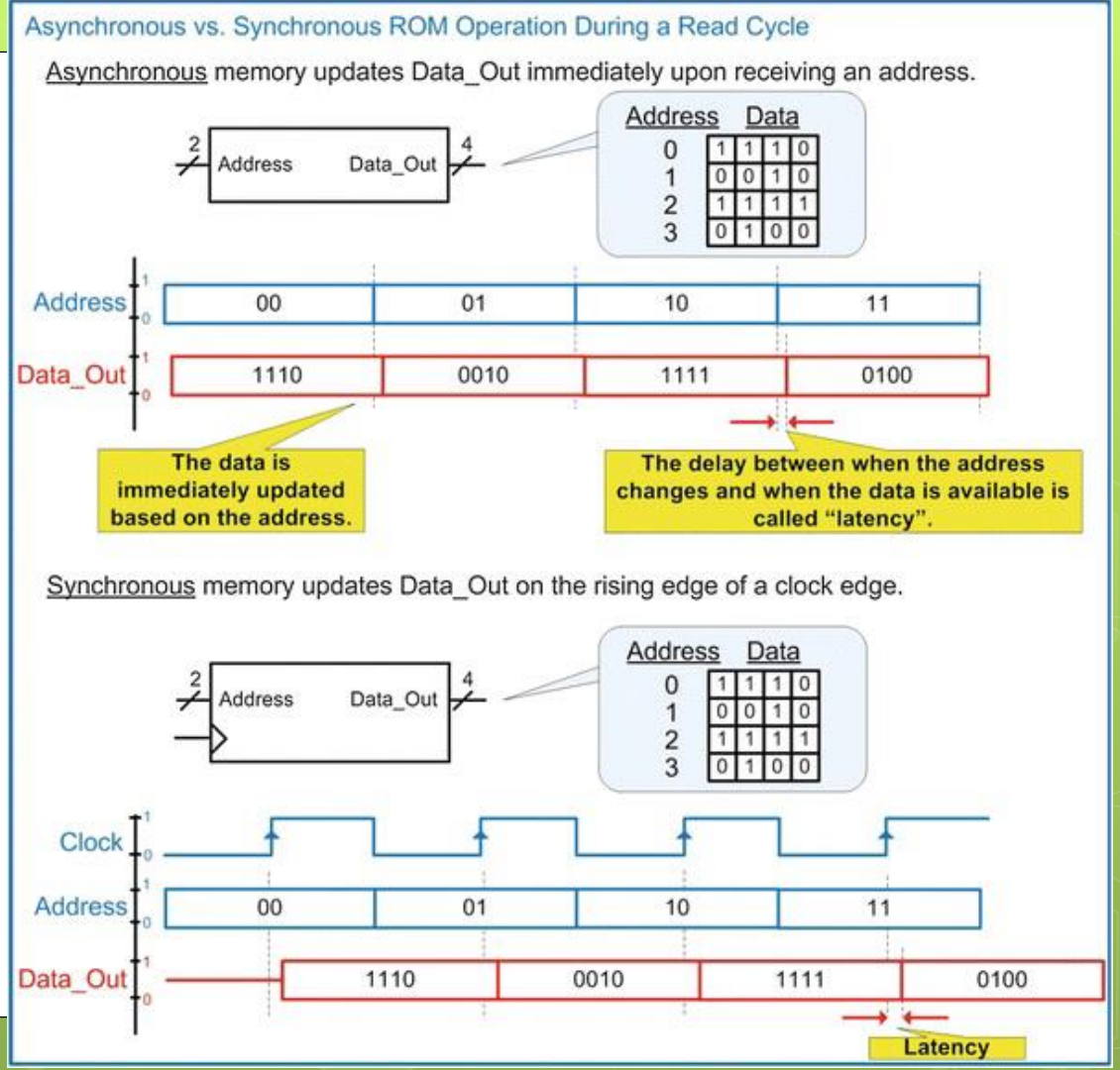
ROM архитектура (2)



ROM архитектура (3)



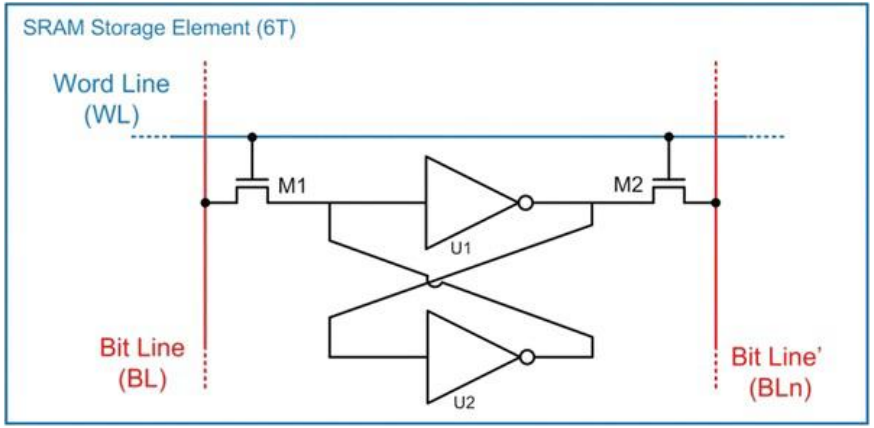
ROM архитектура (4)



SRAM

- Статичка меморија са случајним приступом (SRAM) је полупроводничка технологија која складишти информације помоћу унакрсне повратне спреге претварача. Слика показује шему за основну SRAM ћелију за складиштење.
- У овој конфигурацији, два транзистора за приступ (M1 и M2) се користе за читање и писање из меморијске ћелије. Ћелија има два комплементарна порта која се зову Bit Line (BL) и Bit Line' (BLn). Због функције инвертовања повратне спреге, ова два порта ће увек бити допуњавање једна друге. Овакво понашање је корисно јер се ове две линије могу поредити једна са другом да би се одредила вредност података.
- Ово омогућава да се нивои напона који се користе у ћелији снизе, а да се и даље може детектовати сачувана вредност податка.
- Линије речи се користе за контролу приступних транзистора. Овај елемент за складиштење користи 6 CMOS транзистора за имплементацију и често се назива 6 T конфигурација.
- Предност ове меморијске ћелије је што има веома брзе перформансе у поређењу са другим подсистемима због његове основне технологије као једноставни CMOS транзистори.
- SRAM ћелије се обично имплементирају на исти IC супстрат као и остатак система, што омогућава потпуну интеграцију са системом. SRAM ћелије се користе за кеш меморију у рачунарском систему.

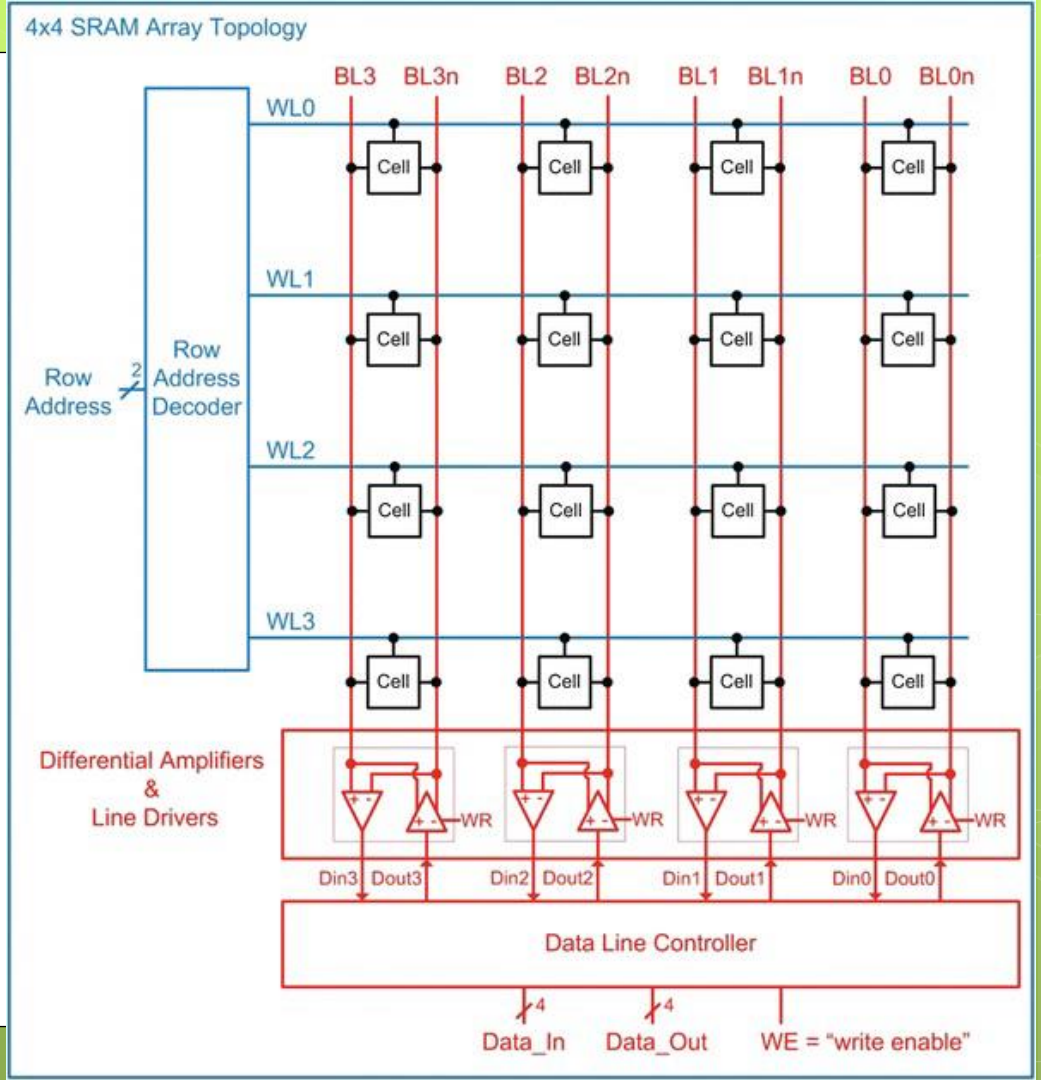
SRAM елемент за складиштење



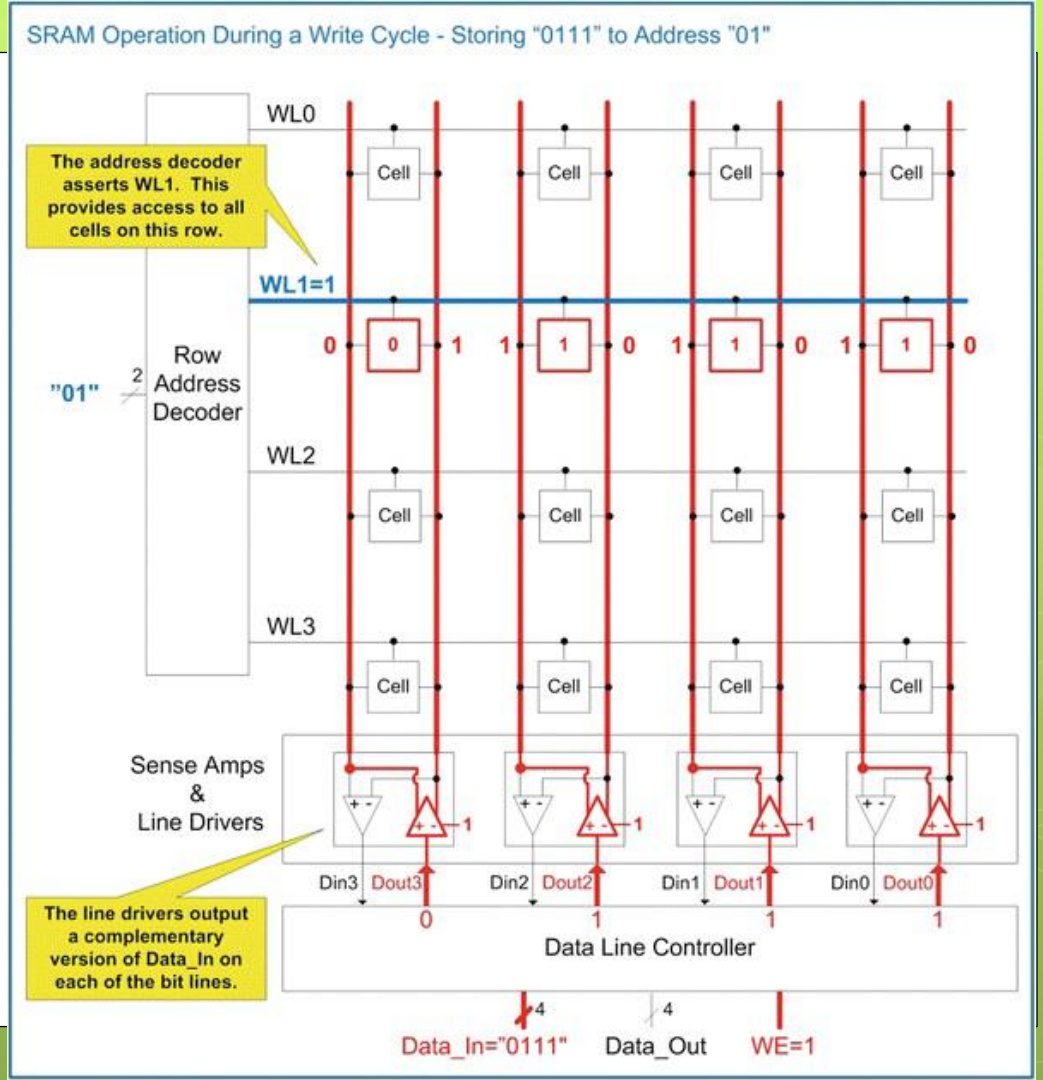
SRAM меморијски систем

- Да би се изградио SRAM меморијски систем, ћелије су распоређене у низу.
- Слика приказује топологију 4 × 4 SRAM низа. У овој конфигурацији, линије речи се деле хоризонтално преко низа како би се обезбедило адресирање. Декодер адресе се користи за претварање бинарно кодиране адресе у одговарајуће вредности линија речи.
- N ћелија за складиштење је повезано за ред речи да обезбеди жељену ширину речи података. Битне линије се деле вертикално преко низа да би се обезбедио приступ подацима (или читање или писање).
- Контролер линије података управља да ли се подаци читају или уписују у ћелије на основу екстерног сигнала за омогућавање писања (WE). Када се постави WE (WE = 1), подаци ће бити уписани у ћелије. Када се WE поништи (WE = 0), подаци ће бити прочитани из ћелија.
- Контролер линије података такође управља одређивањем исправно прочитане логичке вредности из ћелија упоређивањем BL са BLn.
- Како се више ћелија додаје битним линијама, величина сигнала коју покрећу ћелије за складиштење смањује се због додатног оптерећења осталих ћелија. Овде имамо комплементарне податке сигнала (BL и BLn) је предност јер ово ефективно удвостручује величина излаза ћелије за складиштење.
- Обрада поређења BL са BLn се врши коришћењем диференцијалног појачавача који производи потпуни логички ниво излаза чак и када су долазни сигнали веома мали.

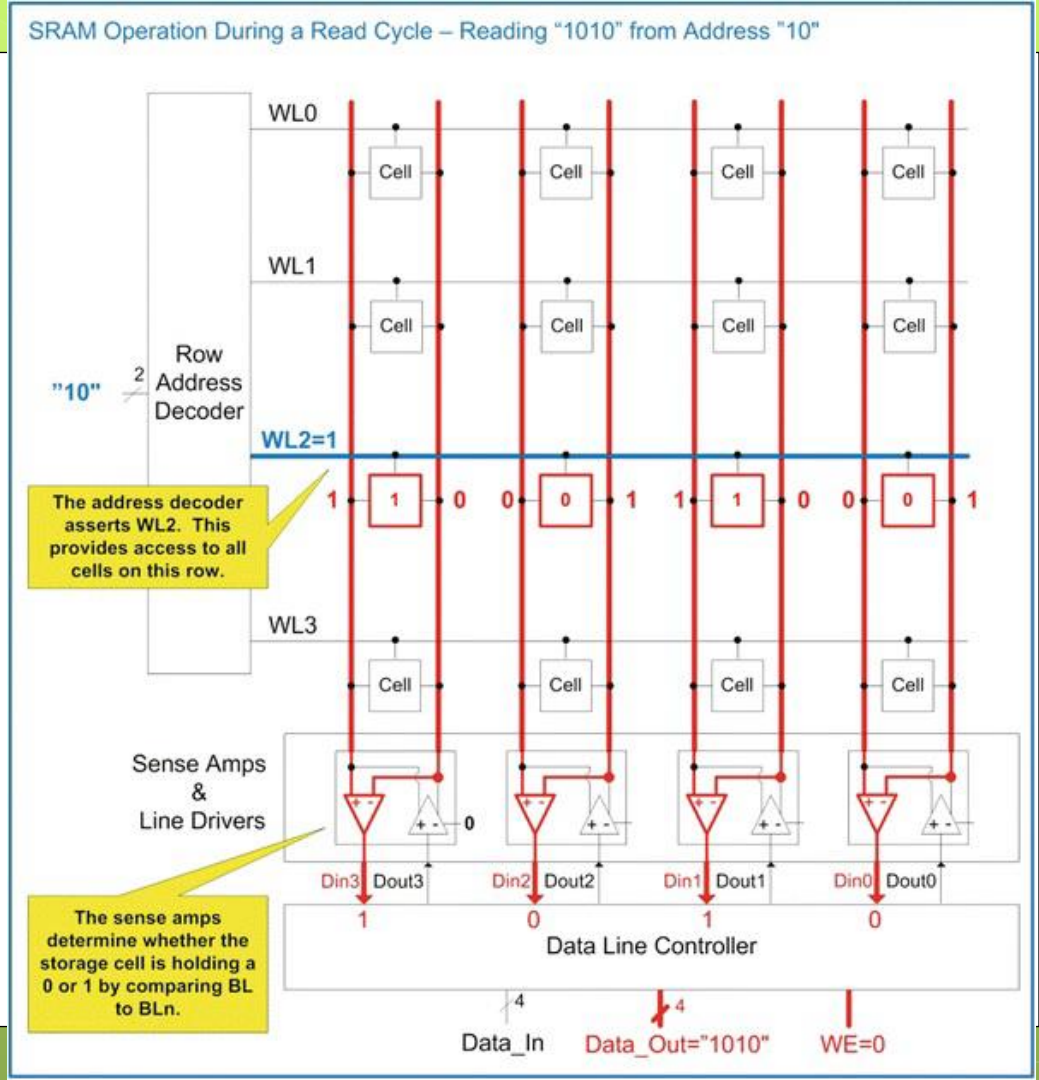
SRAM топологија



SRAM - упис

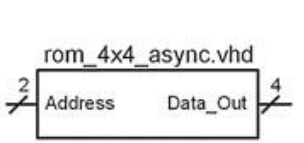

SRAM Operation During a Write Cycle - Storing "0111" to Address "01"
The address decoder asserts WL1. This provides access to all cells on this row.
WL0
WL1=1
WL2
WL3
Cell
"01"
2
Row Address Decoder
Sense Amps & Line Drivers
The line drivers output a complementary version of Data_In on each of the bit lines.
Din3 Dout3
Din2 Dout2
Din1 Dout1
Din0 Dout0
Data Line Controller
4
4
Data_In="0111"
Data_Out
WE=1

SRAM - читање


SRAM Operation During a Read Cycle – Reading "1010" from Address "10"
WL0
WL1
WL2=1
WL3
Cell
"10"
2
Row Address Decoder
The address decoder asserts WL2. This provides access to all cells on this row.
Sense Amps & Line Drivers
The sense amps determine whether the storage cell is holding a 0 or 1 by comparing BL to BLn.
Din3
Dout3
Din2
Dout2
Din1
Dout1
Din0
Dout0
Data Line Controller
Data_In
Data_Out="1010"
WE=0

Асинхрони ROM у VHDL-у

Example: Behavioral Model of a 4x4 Asynchronous Read Only Memory in VHDL



ROM contents for this example:

Address	Data
0	1 1 1 0
1	0 0 1 0
2	1 1 1 1
3	0 1 0 0

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity rom_4x4_async is
  port      (address  : in  std_logic_vector(1 downto 0);
             data_out : out std_logic_vector(3 downto 0));
end entity;

architecture rom_4x4_async_arch of rom_4x4_async is

  type ROM_type is array (0 to 3) of std_logic_vector(3 downto 0);

  constant ROM : ROM_type := (0      => "1110",
                              1      => "0010",
                              2      => "1111",
                              3      => "0100");

  begin

    data_out <= ROM( to_integer(unsigned(address) ) );

end architecture;
```

The numeric_std package is required to provide a type conversion between std_logic_vector and unsigned.

A VHDL "array" is used to define the MxN memory size.

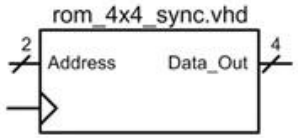
A constant is declared that is of size MxN and is initialized

Since the ROM constant requires indices of type integer but the address is in std_logic_vector, a type conversion is required. The std_logic_vector is first converted to unsigned using a conversion from the numeric_std package. The unsigned value is then converted to an integer using the to_integer cast.

data_out is updated immediately when the address is changed.

Синхрони ROM у VHDL-у

Example: Behavioral Model of a 4x4 Synchronous Read Only Memory in VHDL



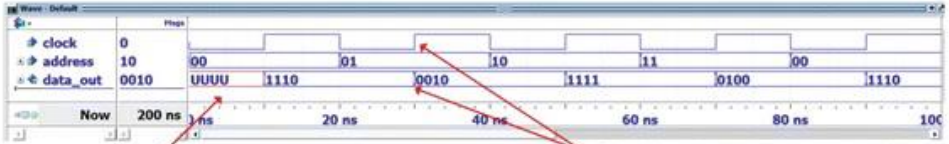
ROM contents for this example:

Address	Data
0	1 1 1 0
1	0 0 1 0
2	1 1 1 1
3	0 1 0 0

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity rom_4x4_sync is
  port     (clock    : in  std_logic;
            address  : in  std_logic_vector(1 downto 0);
            data_out : out std_logic_vector(3 downto 0));
end entity;

architecture rom_4x4_sync_arch of rom_4x4_sync is

  type ROM_type is array (0 to 3) of std_logic_vector(3 downto 0);

  constant ROM : ROM_type := (0      => "1110",
                              1      => "0010",
                              2      => "1111",
                              3      => "0100");
  begin

   MEMORY : process (clock)
     begin
       if (clock'event and clock='1') then
          data_out <= ROM( to_integer(unsigned(address) ) );
       end if;
   end process;

end architecture;
```

To model synchronous behavior, the clock is placed in the sensitivity list, and a rising edge condition is used to trigger the assignment.

When there is not a clock edge, the memory will hold its last output on data_out.

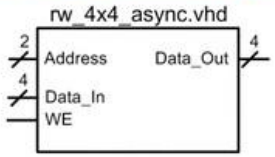


Before the first clock edge, the simulator doesn't know what the output should be.

The data does not appear on the output until a rising edge of clock.

Асинхрони R/W у VHDL-у

Example: Behavioral Model of a 4x4 Asynchronous Read/Write Memory in VHDL



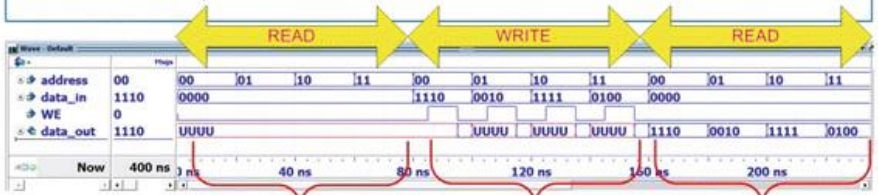
Contents to be written:

Address	Data
0	1 1 1 0
1	0 0 1 0
2	1 1 1 1
3	0 1 0 0

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity rw_4x4_async is
  port     (address  : in  std_logic_vector(1 downto 0);
            data_in  : in  std_logic_vector(3 downto 0);
            WE       : in  std_logic;
            data_out : out std_logic_vector(3 downto 0));
end entity;

architecture rw_4x4_async_arch of rw_4x4_async is

  type RW_type is array (0 to 3) of std_logic_vector(3 downto 0);

  signal RW : RW_type;

  begin

   MEMORY: process (address, WE, data_in)
     begin
        if (WE = '1') then
          RW(to_integer(unsigned(address))) <= data_in;
        else
          data_out <= RW(to_integer(unsigned(address)));
        end if;
   end process;

end architecture;
```

A signal is used since the read/write memory is uninitialized until it is written to.

Type conversions are needed for both reads and writes to RW.



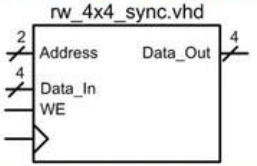
On start-up, the memory is empty so the reads from the four addresses yield "uninitialized".

Data is then written to the four addresses.

When reads are performed again, the data that was written appears.

Синхрони R/W у VHDL-у

Example: Behavioral Model of a 4x4 Synchronous Read/Write Memory in VHDL



Contents to be written:

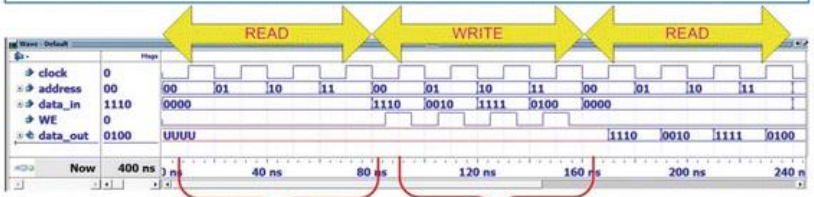
Address	Data
0	1 1 1 0
1	0 0 1 0
2	1 1 1 1
3	0 1 0 0

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity rw_4x4_sync is
  port      (clock    : in  std_logic;
             address  : in  std_logic_vector(1 downto 0);
             data_in  : in  std_logic_vector(3 downto 0);
             WE       : in  std_logic;
             data_out : out std_logic_vector(3 downto 0));
end entity;

architecture rw_4x4_sync_arch of rw_4x4_sync is

  type RW_type is array (0 to 3) of std_logic_vector(3 downto 0);

  signal RW : RW_type;

  begin

   MEMORY : process (clock)
     begin
        if (clock'event and clock='1') then
           if (WE = '1') then
             RW(to_integer(unsigned(address))) <= data_in;
           else
             data_out <= RW(to_integer(unsigned(address)));
           end if;
        end if;
   end process;

end architecture;
```

Synchronous behavior is modeled by listing clock in the sensitivity list and using a rising edge condition.

The WE control signal dictates whether information is read or written to the RW array.

Type conversions are needed for both reads and writes to RW.



Reads are performed on the rising edge of clock when WE=0.

Data is written on the rising edge of clock when WE=1.