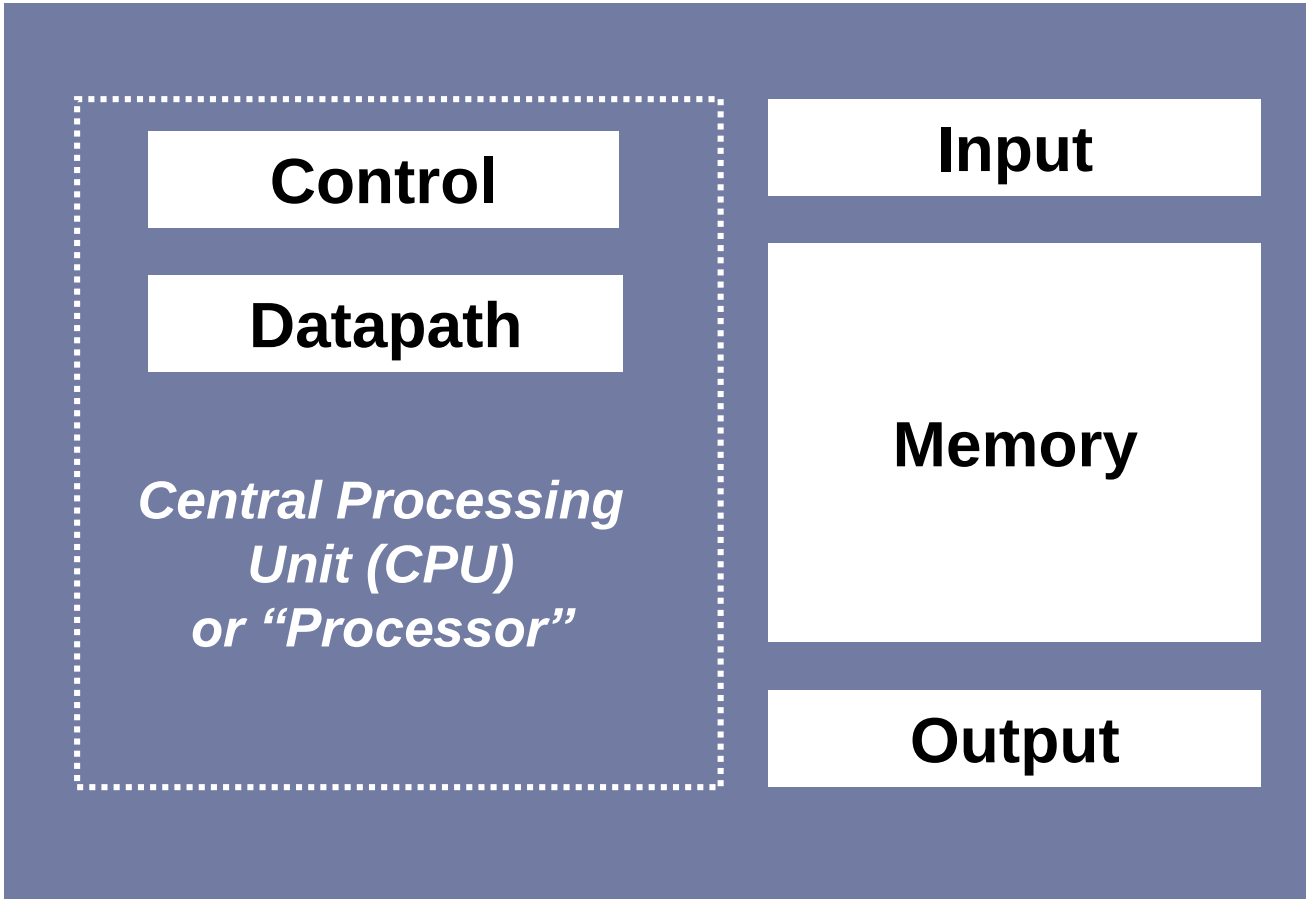
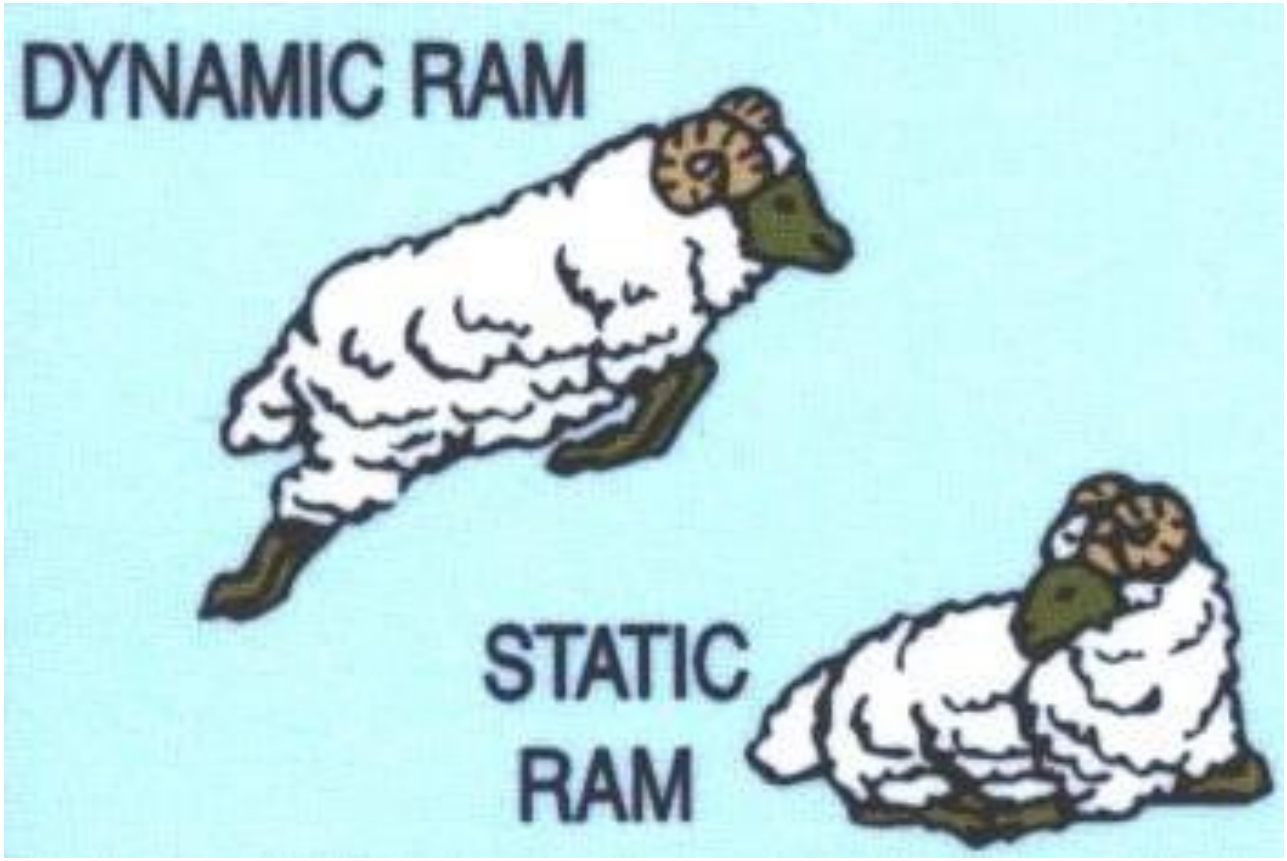
Lecture 6 - Modeling of Memory and Register Files

A typical Computing System

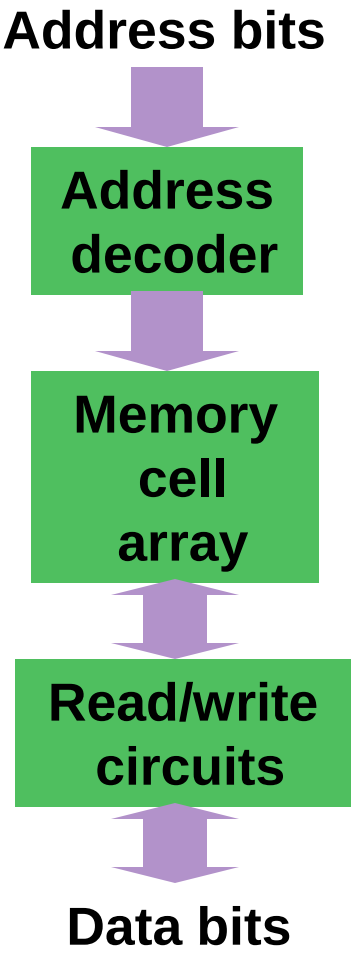


Types of Computer Memories

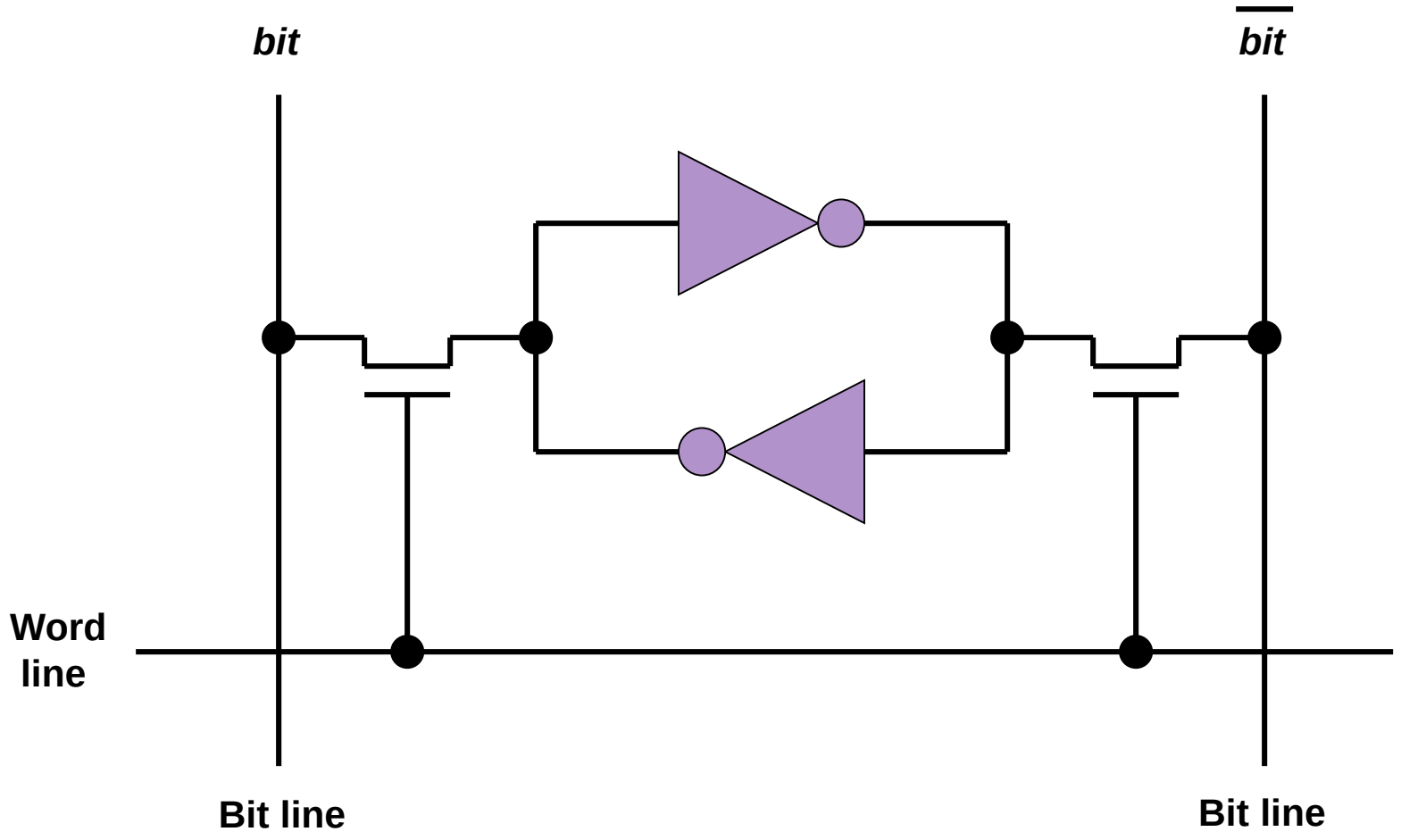


From the cover of:
A. S. Tanenbaum, *Structured Computer Organization, Fifth Edition*, Upper Saddle River, New Jersey: Pearson Prentice Hall, 2006.

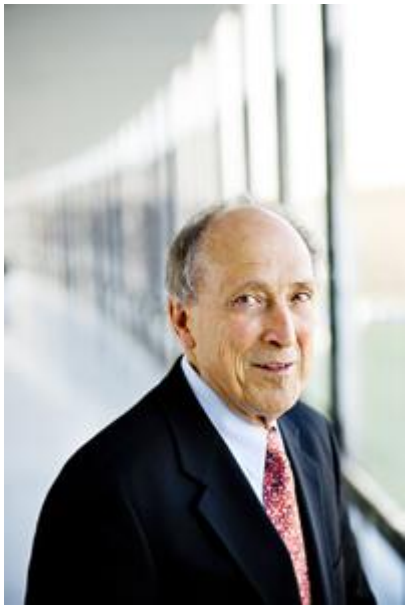
Random Access Memory (RAM)



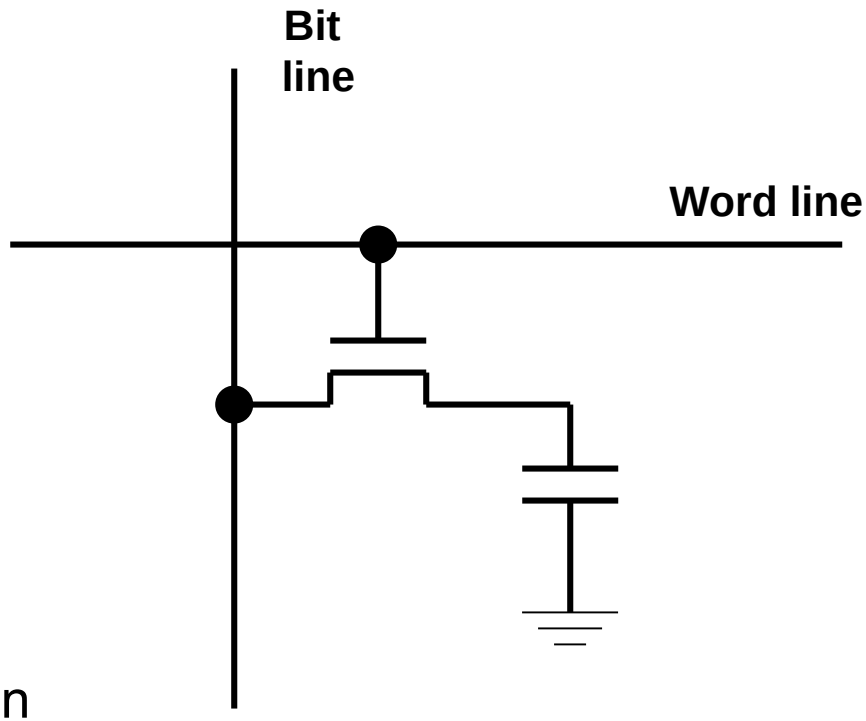
Six-Transistor SRAM Cell



Dynamic RAM (DRAM) Cell



"Single-transistor DRAM cell"
Robert Dennard's 1967 invention



Electronic Memory Devices

Memory technology	Typical access time	$ per GiB in 2012
SRAM semiconductor memory	0.5–2.5 ns	$500–$1000
DRAM semiconductor memory	50–70 ns	$10–$20
Flash semiconductor memory	5,000–50,000 ns	$0.75–$1.00
Magnetic disk	5,000,000–20,000,000 ns	$0.05–$0.10

For more on memories:

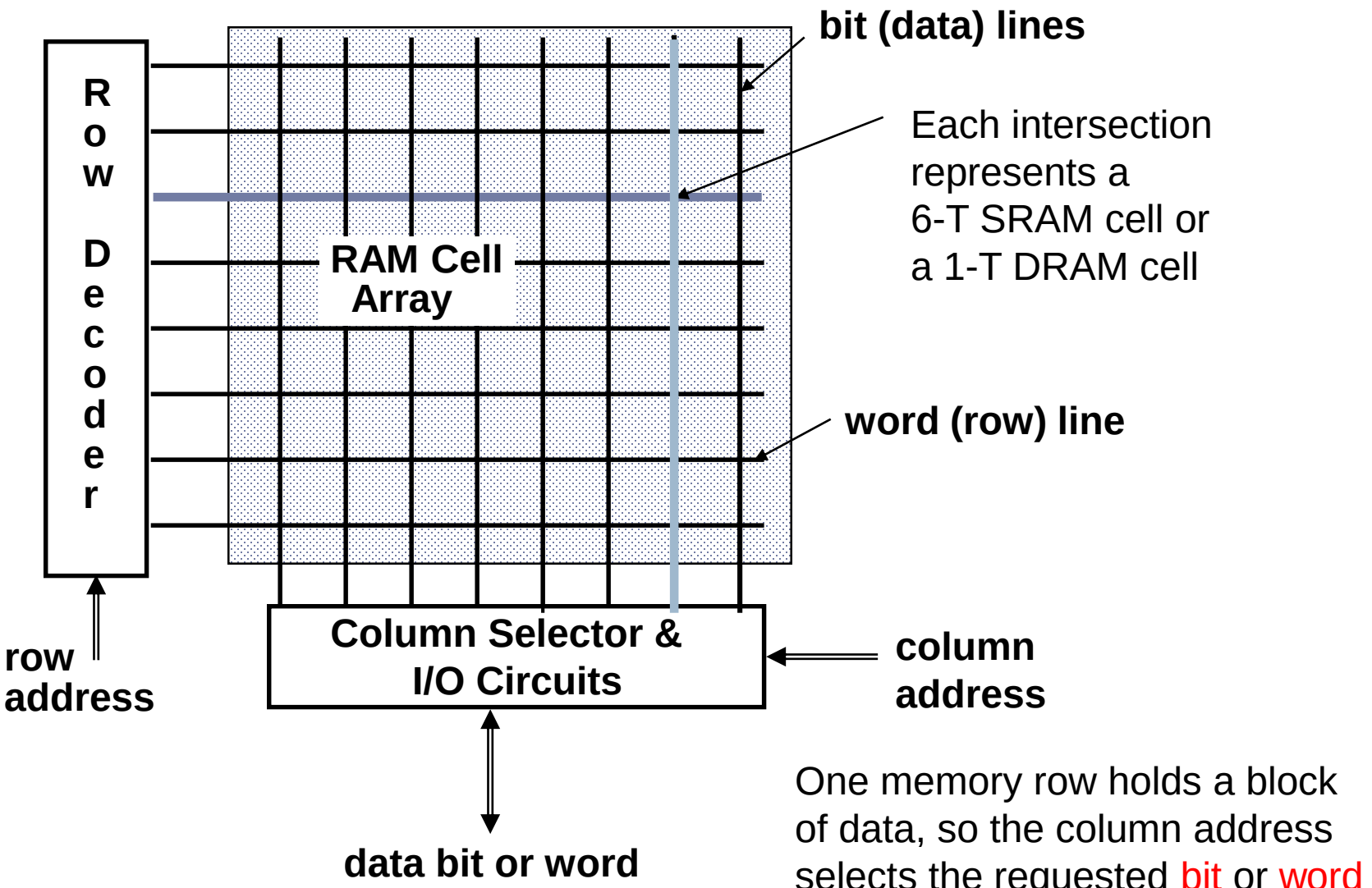
Semiconductor Memories: A Handbook of Design, Manufacture and Application, by Betty Prince, Wiley 1996.

Emerging Memories: Technologies and Trends**,** by Betty Prince, Springer 2002.

DRAM Evolution

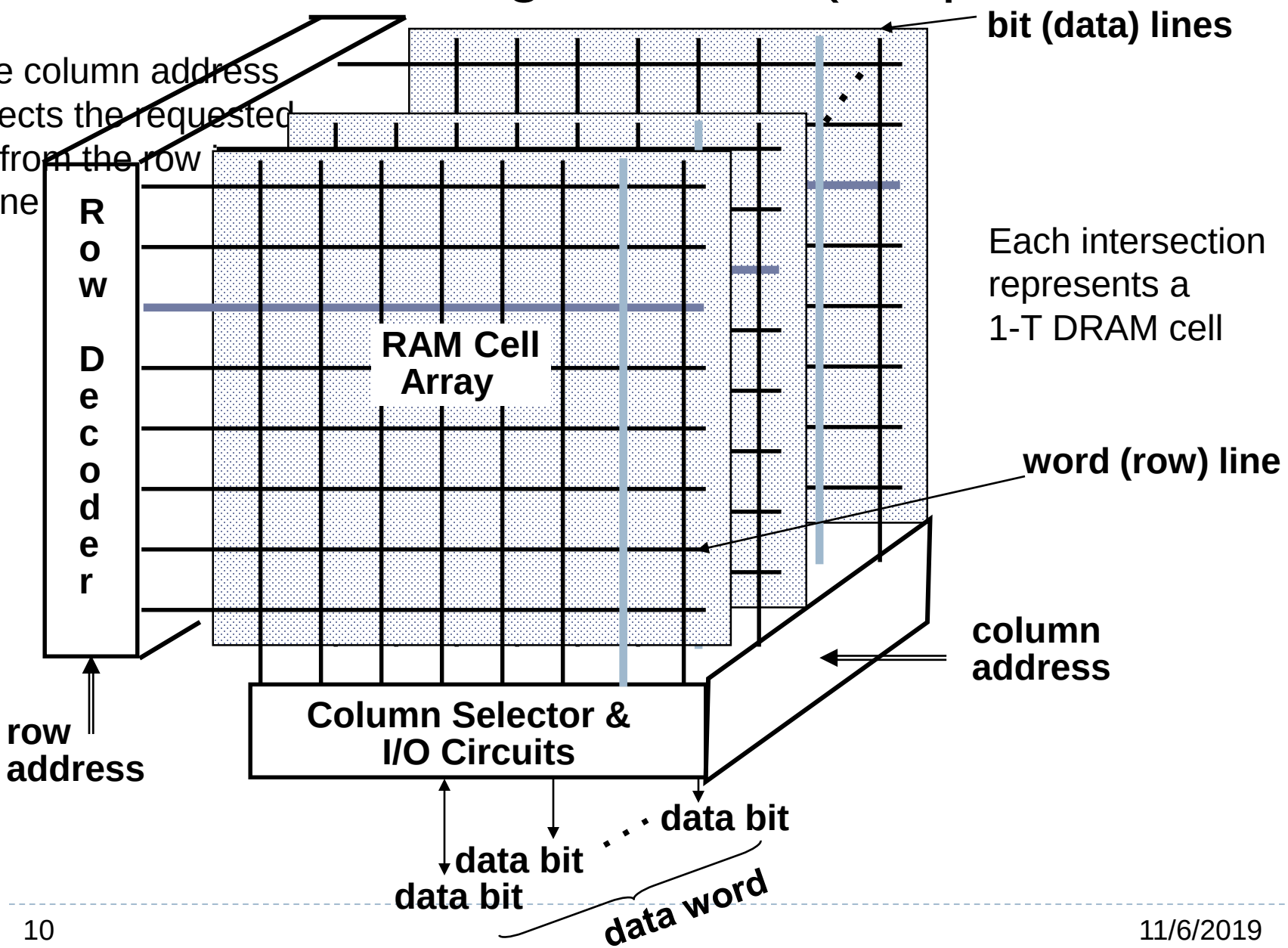
Year introduced	Chip size	$ per GiB	Total access time to a new row/column	Average column access time to existing row
1980	64 KiB	$1,500,000	250 ns	150 ns
1983	256 KiB	$500,000	185 ns	100 ns
1985	1 MeB	$200,000	135 ns	40 ns
1989	4 MeB	$50,000	110 ns	40 ns
1992	16 MeB	$15,000	90 ns	30 ns
1996	64 MeB	$10,000	60 ns	12 ns
1998	128 MeB	$4,000	60 ns	10 ns
2000	256 MeB	$1,000	55 ns	7 ns
2004	512 MeB	$250	50 ns	5 ns
2007	1 GiB	$50	45 ns	1.25 ns
2010	2 GiB	$30	40 ns	1 ns
2012	4 GiB	$1	35 ns	0.8 ns

Classical RAM Organization (~Square)



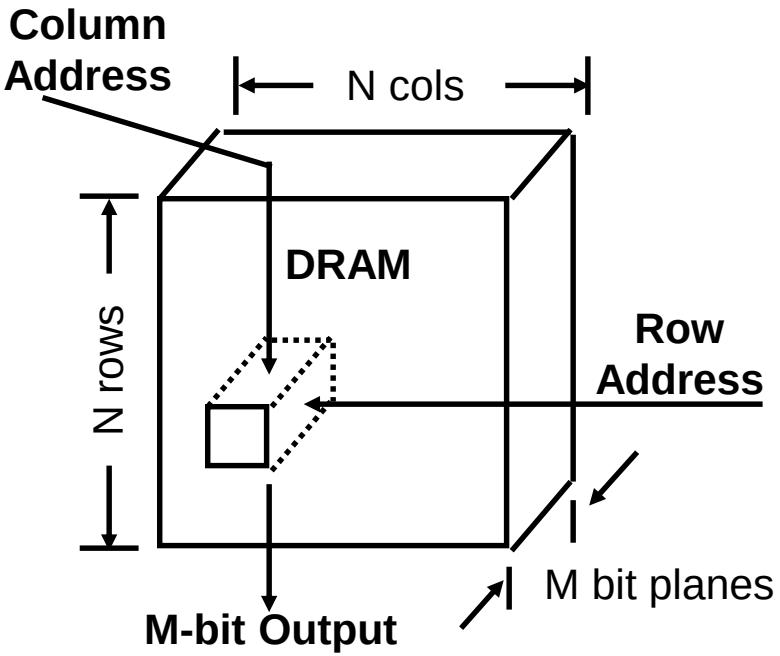
One memory row holds a block of data, so the column address selects the requested bit or word from that block

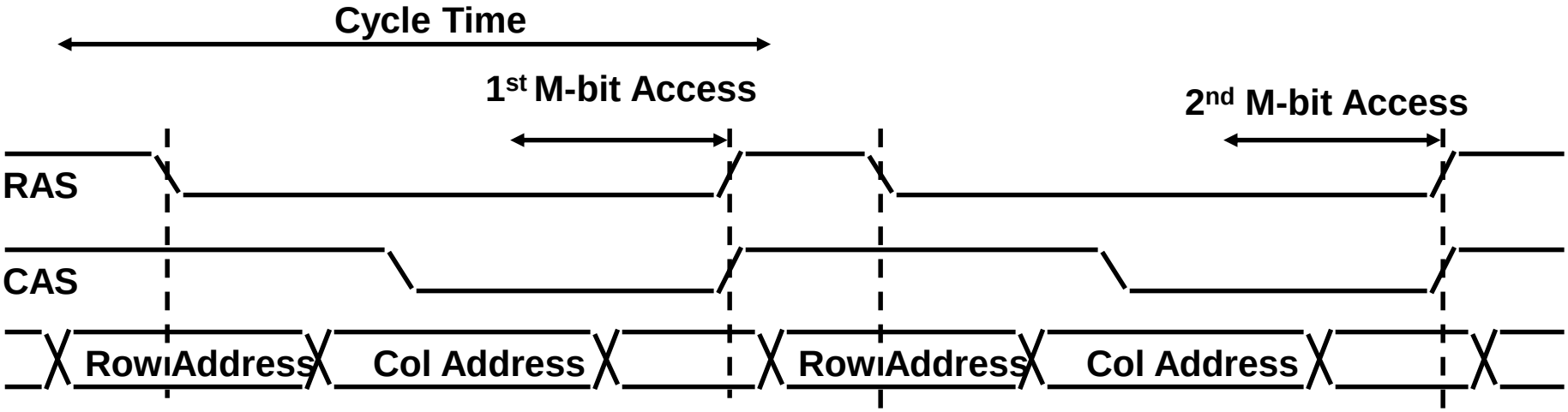
Classical DRAM Organization (~Square Planes)



Classical DRAM Operation

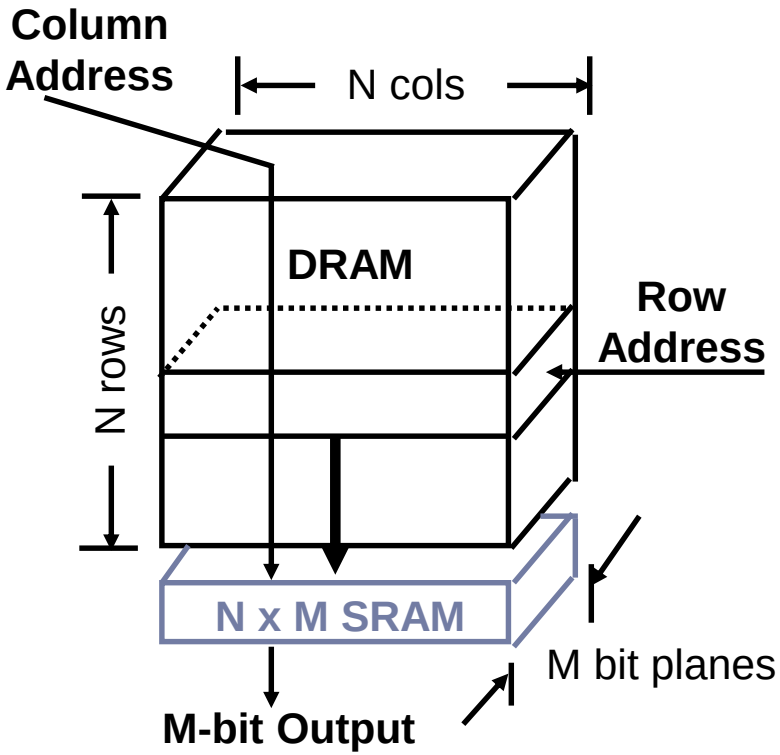
- DRAM Organization:
  - N rows x N column x M-bit
  - Read or Write M-bit at a time
  - Each M-bit access requires a RAS / CAS cycle

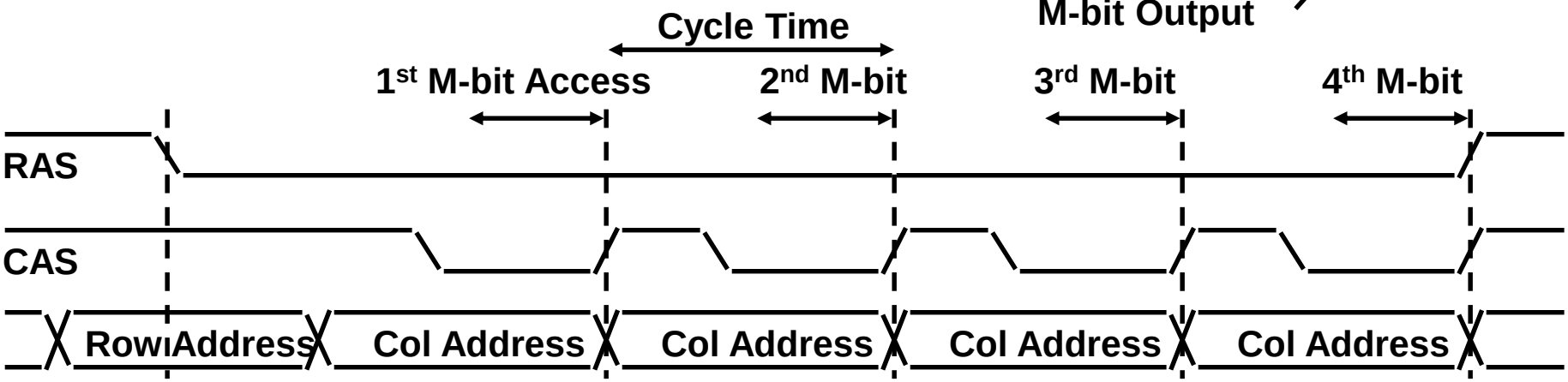




Page Mode DRAM Operation

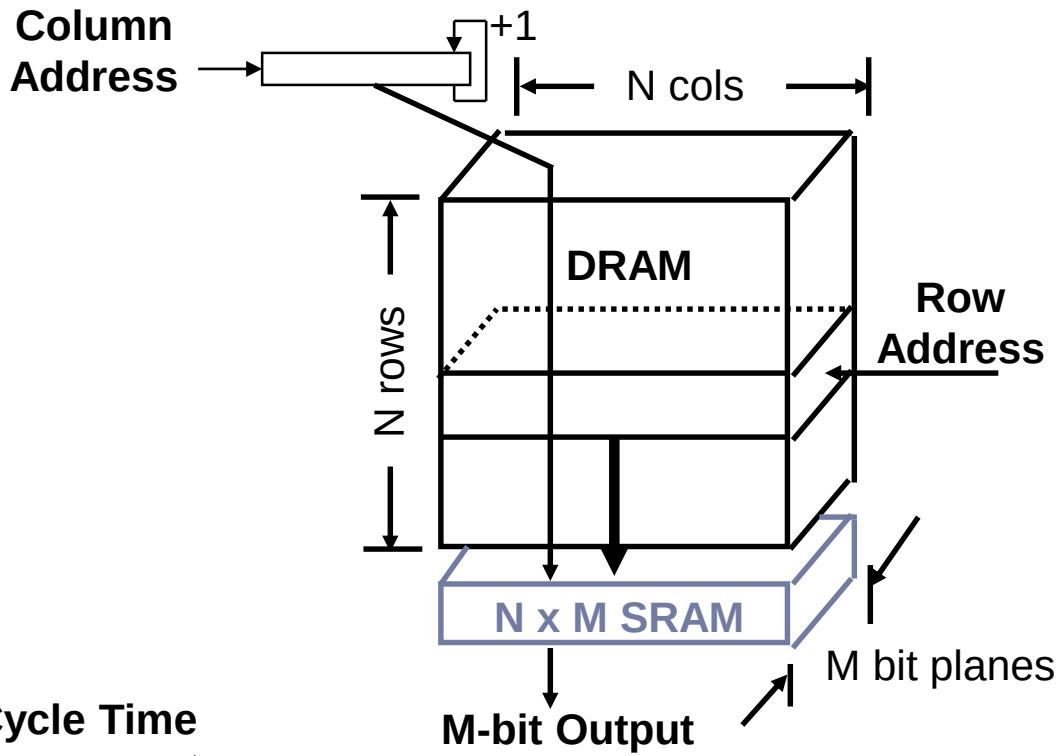
- Page Mode DRAM
  - N x M SRAM to save a row
- After a row is read into the SRAM “register”
  - Only CAS is needed to access other M-bit words on that row
  - RAS remains asserted while CAS is toggled

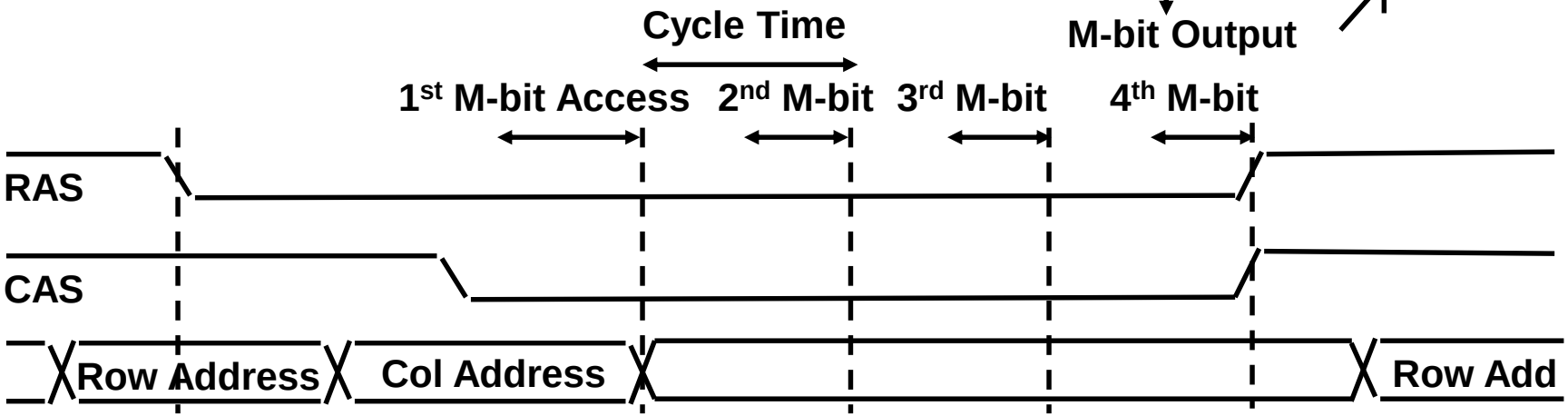




Synchronous DRAM (SDRAM) Operation

- After a row is read into the SRAM register
  - Inputs CAS as the starting "burst" address along with a burst length
  - Transfers a burst of data from a series of sequential addresses within that row
    - A **clock** controls transfer of successive words in the burst – 300MHz in 2004





Other SDRAM Architectures

- Double Data Rate SDRAMs – DDR-SDRAMs
  - Double data rate because they transfer data on both the rising and falling edge of the clock
    - Most widely used form of SDRAMs
  - For DDR memory, 2n prefetch architecture means
    - Internal bus width is twice of external bus width
    - Hence, internal column access freq can be half of external data rate
  - For users, 2n prefetch means that data access occurs in pairs
    - A single READ fetches two data words
    - A single WRITE, two data words must be provided

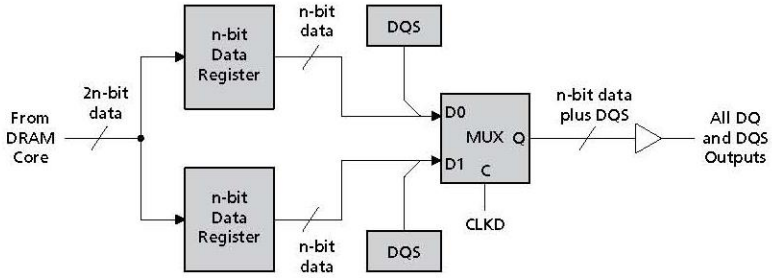


Figure 1. Simplified Block Diagram of 2n-Prefetch READ

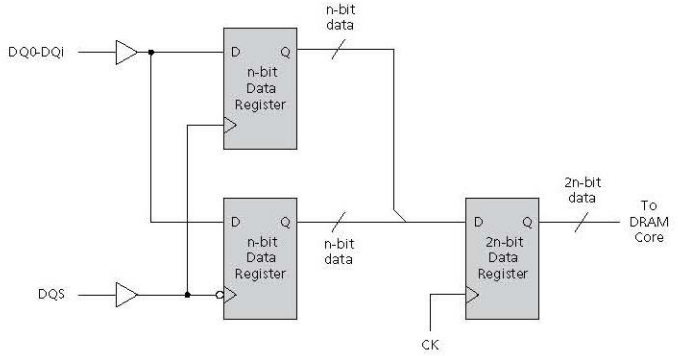
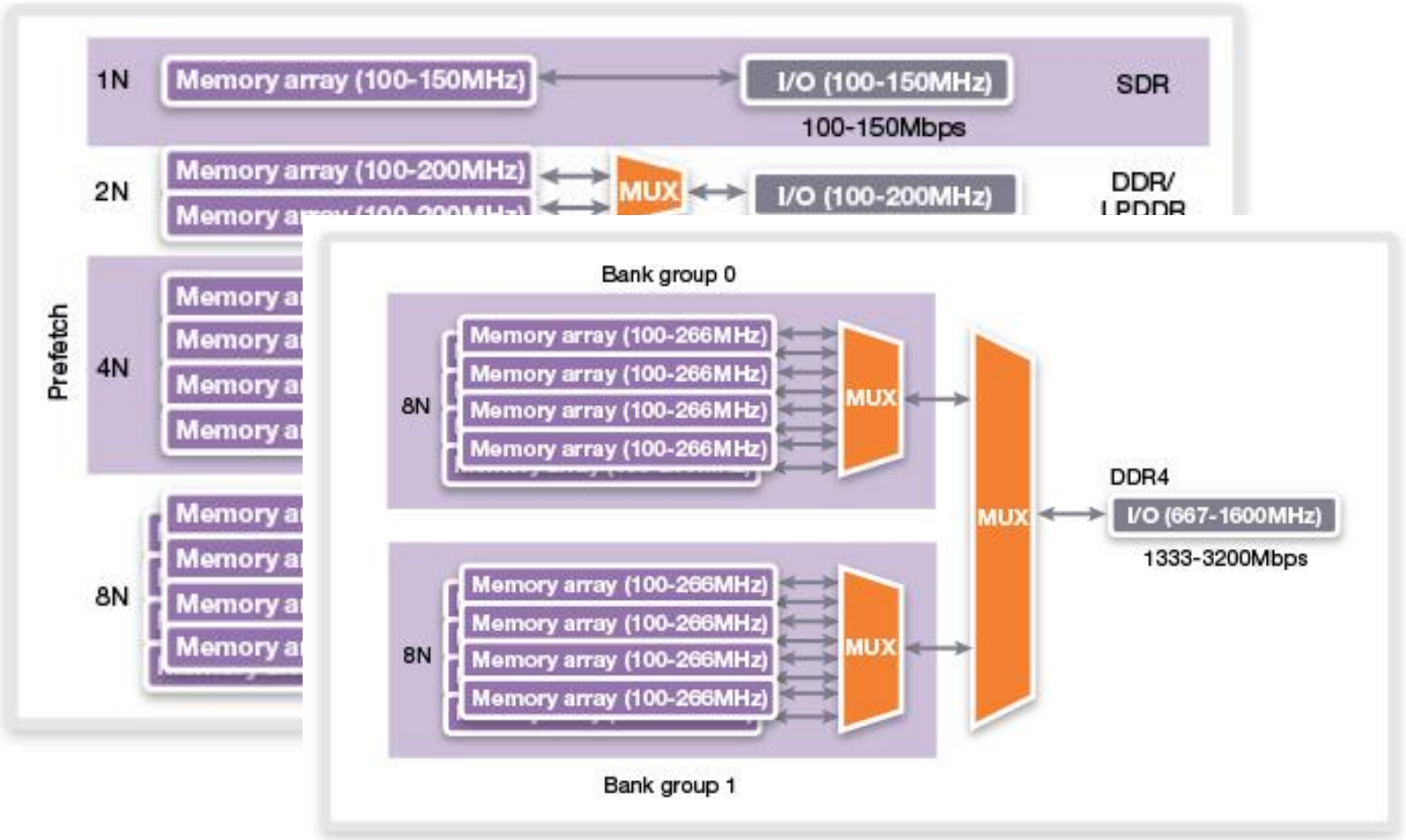


Figure 2. Simplified Block Diagram of 2n-Prefetch WRITE

Other SDRAM Architectures- Cont.

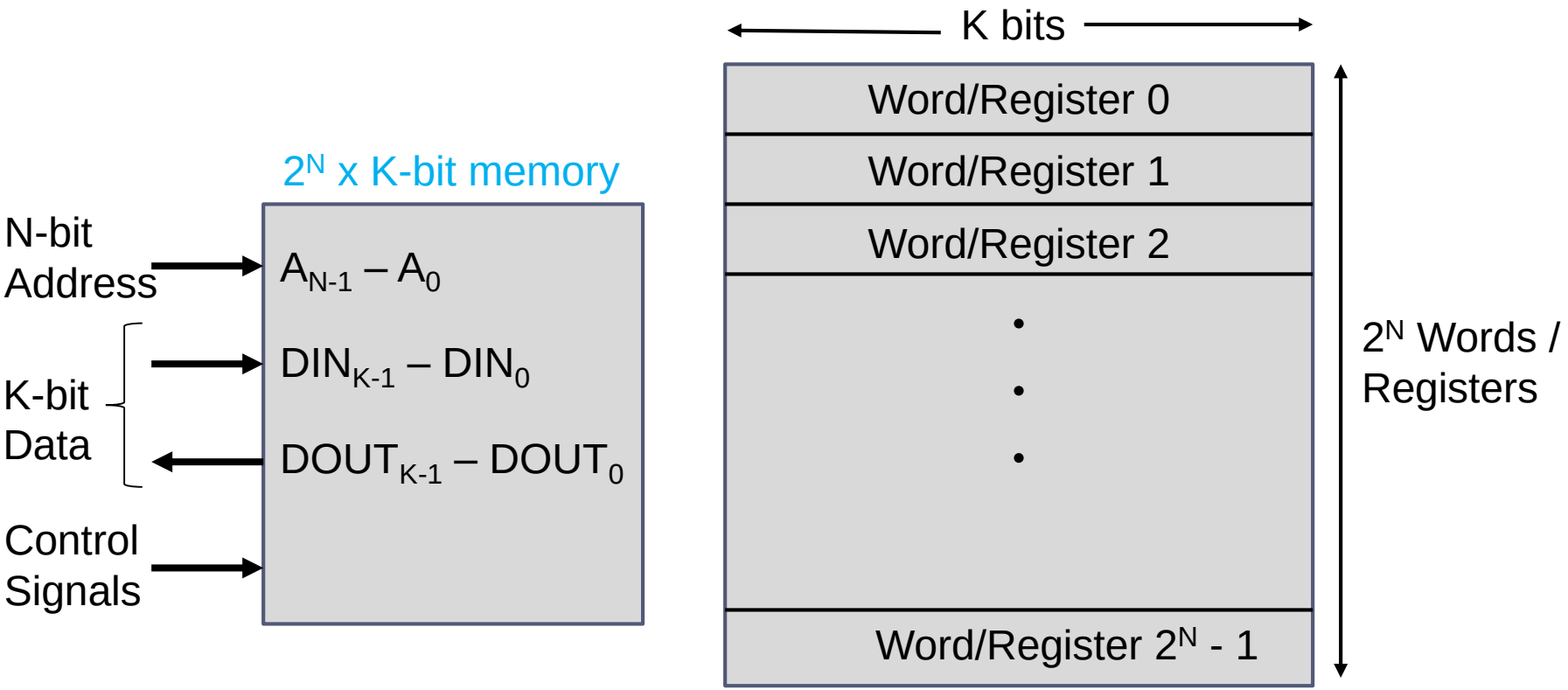


https://www.synopsys.com/Company/Publications/DWTB/Pages/dwtb-ddr4-bank-groups-2013Q2.aspx

Memory Synthesis

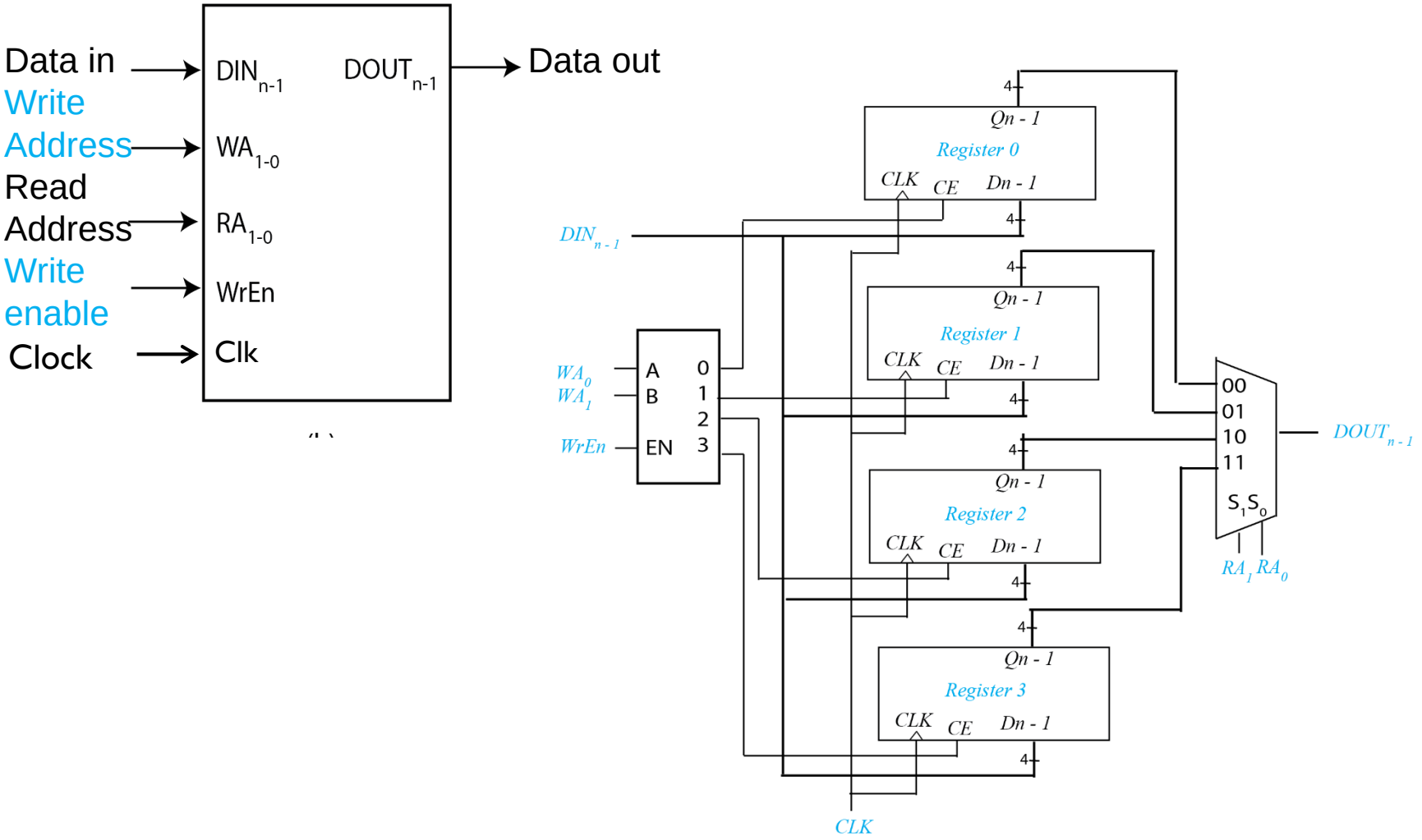
- Approaches:
  - Random logic using flip-flops or latches
  - Register files in data paths
  - RAM standard components
  - RAM compilers
- Computer “register files” are often just multi-port RAMs
  - ARM CPU: 32-bit registers R0-R15 => 16 x 32 RAM
  - MIPS CPU: 32-bit registers R0-R31 => 32 x 32 RAM
- Communications systems often use dual-port RAMs as transmit/receive buffers
  - FIFO (first-in, first-out RAM)

Basic memory/register array



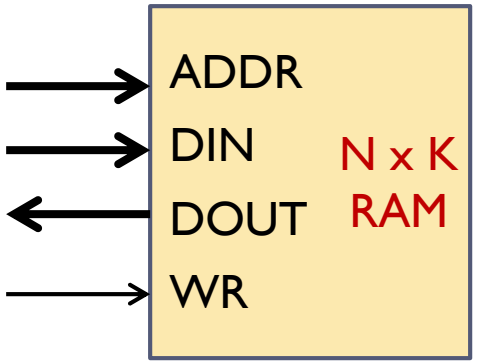

```
-- 2^N x K-bit memory VHDL struture
signal MemArray: array (0 to 2**N – 1) of std_logic_vector(K-1 downto 0);
-- ARM register file is 16 32-bit registers
signal ARMregisterFile: array (0 to 15) of std_logic_vector(31 downto 0);
```

Example: 4 x n-bit register file



Technology-independent RAM Models

```
-- N x K RAM is 2-dimensional array of N K-bit words
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_numeric_std.all;

entity RAM is
        generic (K: integer:=8;             -- number of bits per word
                  W: integer:=8);           -- number of address bits; N = 2^W
        port (
              WR:    in std_logic;          -- active high write enable
              ADDR: in std_logic_vector (W-1 downto 0);     -- RAM address
              DIN:    in std_logic_vector (K-1 downto 0);   -- write data
              DOUT: out std_logic_vector (K-1 downto 0));   -- read data
end entity RAM;
```



RAM Models in VHDL

```
architecture RAMBEHAVIOR of RAM is
    subtype WORD  is std_logic_vector ( K-1 downto 0);      -- define size of WORD
    type MEMORY is array (0 to 2**A-1) of WORD;             -- define size of MEMORY
    signal RAM256:  MEMORY;                                 -- RAM256 as signal of type MEMORY

begin
   process (WR, DIN, ADDR)
       variable RAM_ADDR_IN: natural range 0 to 2**W-1;  -- translate address to integer
   begin
               RAM_ADDR_IN := to_integer(UNSIGNED(ADDR));        -- convert address to integer
               if (WR='1') then                                   -- write operation to RAM
                       RAM256 (RAM_ADDR_IN) <= DIN ;
               end if;
               DOUT <= RAM256 (RAM_ADDR_IN);                      -- continuous read operation
    end process;
end architecture RAMBEHAVIOR;
```

Multi-port RAM (two parallel outputs):

```
DOUT1 <= RAM256(to_integer(UNSIGNED(ADDR1));
DOUT2 <= RAM256(to_integer(UNSIGNED(ADDR2));
```

Initialize RAM at start of simulation

```
process (WR, DIN, ADDR)
   variable RAM_ADDR_IN:  natural range 0 to 2**W-1;  -- to translate address to integer
   variable STARTUP:  boolean := true;                -- temp variable for initialization

begin
    if (STARTUP = true) then     -- for initialization of RAM during start of simulation
            RAM256 <=  (0 => "00000101",          -- initializes first 4 locations in RAM
                        1 => "00110100",          -- to specific values
                        2 => "00000110",          -- all other locations in RAM are
                        3 => "00011000",          -- initialized to all 0s
                        others => "00000000");
            DOUT <= "XXXXXXXX";       -- force undefined logic values on RAM output
            STARTUP :=false;  -- now this portion of process will only execute once
     else
          -- “Normal” RAM operations
```

RAM with bidirectional data bus

FIGURE 8-14: Block Diagram of Static RAM

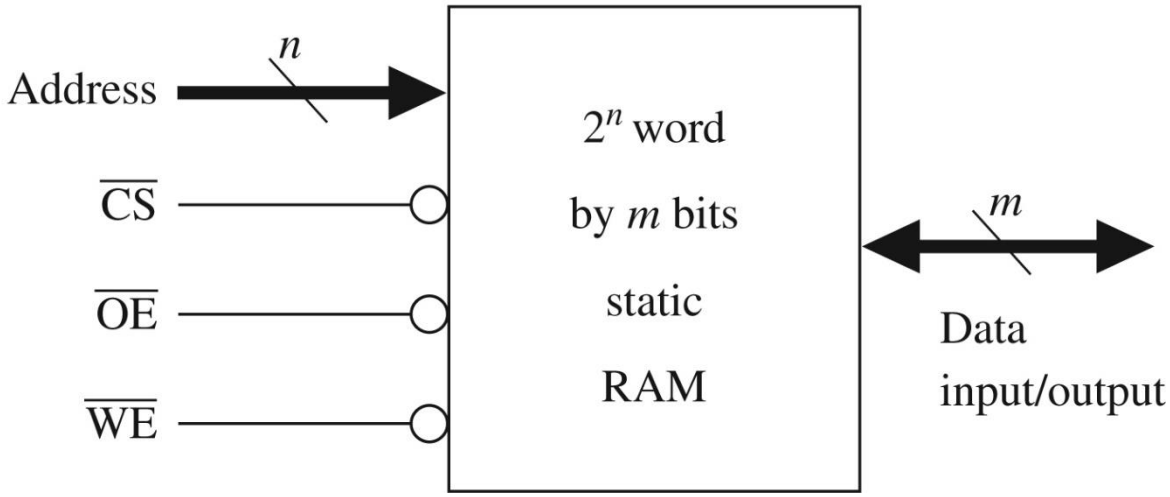
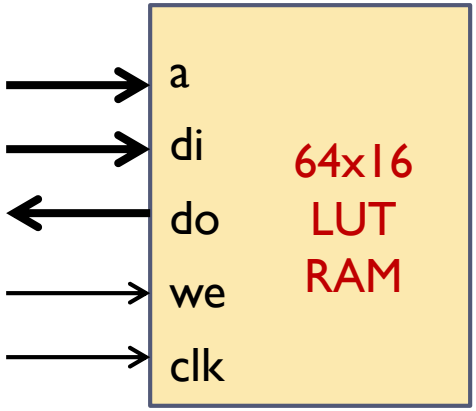


TABLE 8-7: Truth Table for Static RAM

$\overline{CS}$	$\overline{OE}$	$\overline{WE}$	Mode	I/O pins
H	X	X	not selected	high-Z
L	H	H	output disabled	high-Z
L	L	H	read	data out
L	X	L	write	data in

Single-port distributed RAM

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

entity rams_04 is
        port (      clk : in std_logic;
                    we : in std_logic;
                    a : in std_logic_vector(5 downto 0);
                    di : in std_logic_vector(15 downto 0);
                    do : out std_logic_vector(15 downto 0));
end rams_04;

architecture syn of rams_04 is
        type ram_type is array (63 downto 0) of std_logic_vector (15 downto 0);
        signal RAM : ram_type;
begin
        process (clk)
        begin
                if (clk'event and clk = '1') then
                        if (we = '1') then
                                RAM(conv_integer(a)) <= di;
                        end if;
                end if;
        end process;
        do <= RAM(conv_integer(a));
end syn;
```



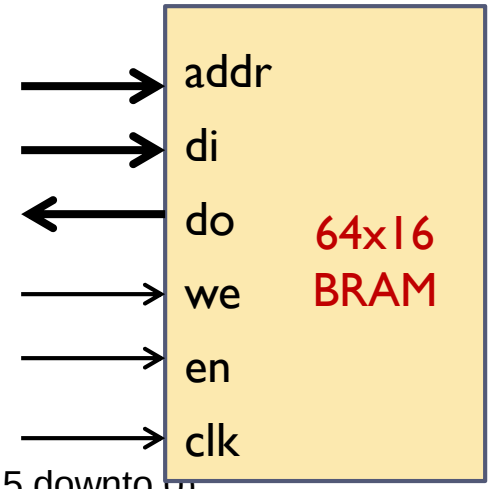
From Xilinx "Synthesis and Simulation Design Guide"

FIGURE 6-18: Behavioral VHDL Code That Typically Infers LUT-Based Memory

```
library IEEE;
use IEEE.numeric_bit.all;

entity Memory is
  port(Address: in unsigned(6 downto 0);
       CLK, MemWrite: in bit;
       Data_In: in unsigned(31 downto 0);
       Data_Out: out unsigned(31 downto 0));
end Memory;

architecture Behavioral of Memory is
type RAM is array (0 to 127) of unsigned(31 downto 0);
signal DataMEM: RAM;  -- no initial values
begin
  process(CLK)
  begin
    if CLK'event and CLK = '1' then
      if MemWrite = '1' then
        DataMEM(to_integer(Address)) <= Data_In;  -- Synchronous Write
      end if;
    end if;
  end process;

  Data_Out <= DataMEM(to_integer(Address));  -- Asynchronous Read
end Behavioral;
```

Block RAM inferred

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

entity rams_01 is
        port (      clk : in std_logic;
                    we : in std_logic;
                    en : in std_logic;
                    addr : in std_logic_vector(5 downto 0);
                    di : in std_logic_vector(15 downto 0);
                    do : out std_logic_vector(15 downto 0));
end rams_01;

architecture syn of rams_01 is
        type ram_type is array (63 downto 0) of std_logic_vector (15 downto 0);
        signal RAM: ram_type;
begin
        process (clk)
        begin
                if clk'event and clk = '1' then
                        if en = '1' then
                                if we = '1' then
                                        RAM(conv_integer(addr)) <= di;
                                end if;
                                do <= RAM(conv_integer(addr)) ;      -- read-first operation
                        end if;
                end if;
        end process;
end syn;
```

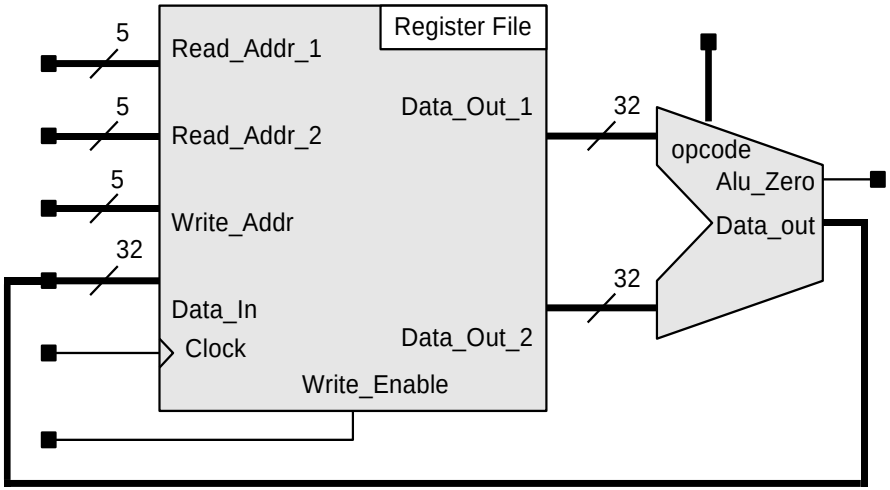


From Xilinx "Synthesis and Simulation Design Guide"

FIGURE 6-19: **Behavioral VHDL Code That Typically Infers Dedicated Memory**

```
library IEEE;
use IEEE.numeric_bit.all;

entity Memory is
  port(Address: in unsigned(6 downto 0);
       CLK, MemWrite: in bit;
       Data_In: in unsigned(31 downto 0);
       Data_Out: out unsigned(31 downto 0));
end Memory;

architecture Behavioral of Memory is
type RAM is array (0 to 127) of unsigned(31 downto 0);
signal DataMEM: RAM;  -- no initial values
begin
  process(CLK)
  begin
    if CLK'event and CLK = '1' then
      if MemWrite = '1' then
        DataMEM(to_integer(Address)) <= Data_In;  -- Synchronous Write
      end if;
      Data_Out <= DataMEM(to_integer(Address));   -- Synchronous Read
    end if;
  end process;
end Behavioral;
```

Register File in Verilog



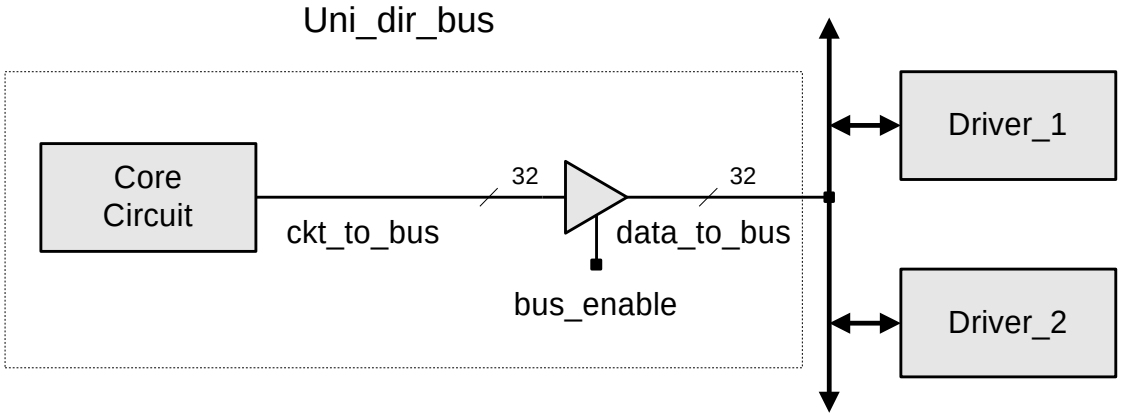

```
module Register_File (Data_Out_1,Data_Out_2,Data_in,
                      Read_Addr_1,Read_Addr_2,Write_Addr,Write_Enable,Clock);
  output   [31: 0]    Data_Out_1, Data_Out_2;
  input    [31: 0]    Data_in;
  input    [4: 0]     Read_Addr_1, Read_Addr_2, Write_Addr;
  input               Write_Enable, Clock;
  reg      [31: 0]    Reg_File [31: 0];      // 32bit  x32 word memory declaration

  assign Data_Out_1 = Reg_File[Read_Addr_1];
  assign Data_Out_2 = Reg_File[Read_Addr_2];
  always @ (posedge Clock) begin
    if (Write_Enable) Reg_File [Write_Addr] <= Data_in;
  end
endmodule
```

“Concept of Memory” in Verilog

- Memory
  - Declaration an array of words
  - E.g. reg [31:0] data_out; // one 32-bit word
    reg [31:0] Reg_file [31:0]; // 32x32 bit word memory
- Verilog does not support 2-dimensional array
  - However, a word in a Verilog memory can be addressed directly
    - E.g., Reg_file [12]
  - A cell bit in a word can also be addressed indirectly by first loading the word into a buffer register then addressing the bit of the word
    - E.g. Data_out = Reg_file [12];
      Data_out [1:0]
- Decoder are synthesized automatically by synthesis tool in Reg_file[] to decode the address which locates a specific register

Bus Interface

- Unidirectional Bus

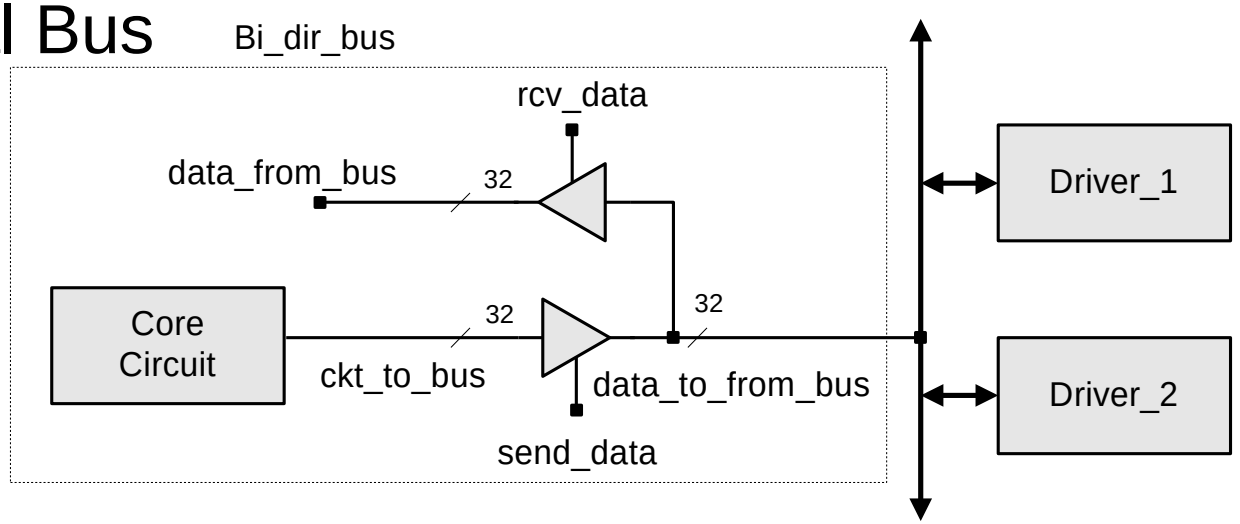



```
module Uni_dir_bus ( data_to_bus, bus_enable);
Input                bus_enable;
output [31: 0]       data_to_bus;
reg [31: 0]          ckt_to_bus;
assign data_to_bus = (bus_enable)? ckt_to_bus: 32'bz;
// Description of core circuit goes here to drive ckt_to_bus
endmodule
```

Michael D. Ciletti, *Advanced digital design with the Verilog HDL*.

Bus Interface

- Bidirectional Bus




```
module Bi_dir_bus (data_to_from_bus, send_data, rcv_data);
inout    [31: 0]   data_to_from_bus;
Input              send_data, rcv_data;
wire     [31: 0]   ckt_to_bus;
wire     [31: 0]   data_to_from_bus, data_from_bus;
assign data_from_bus = (rcv_data)? data_to_from_bus: 32'bz;
assign data_to_from_bus = (send_data)? ckt_to_bus: data_to_from_bus;
// Behavior using data_from_bus and generating
// ckt_to_bus goes here
endmodule
```