EITF35: Introduction to Structured VLSI Design

Part 3.2.1: Memories

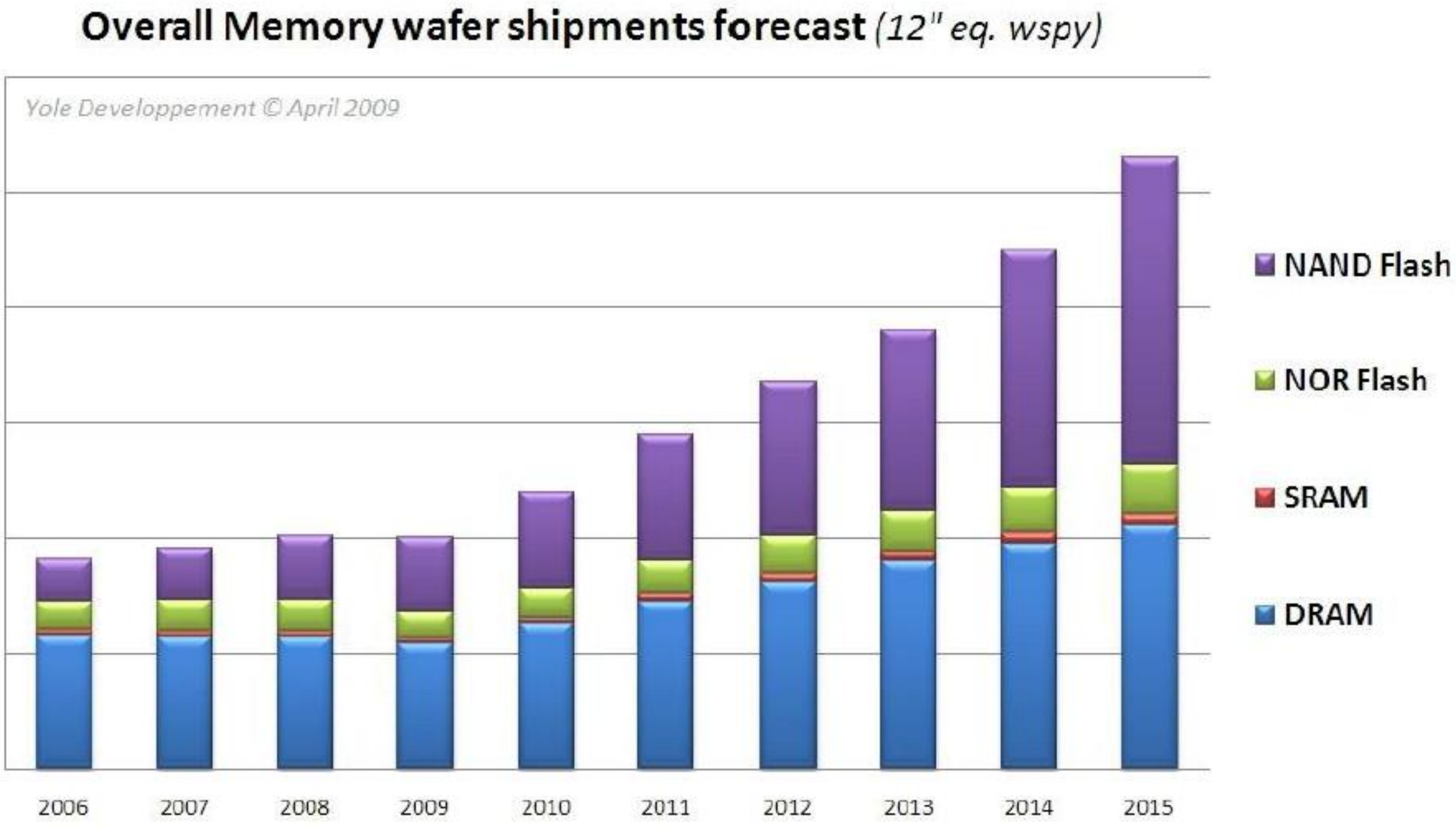
Liang Liu
liang.liu@eit.lth.se

Outline

- □ **Overview of Memory**
  - Application, history, trend
  - Different memory type
  - Overall architecture
- □ **Registers as Storage Element**
  - Register File
  - FIFO
- □ **Xilinx Storage Elements**

Memory is Everywhere

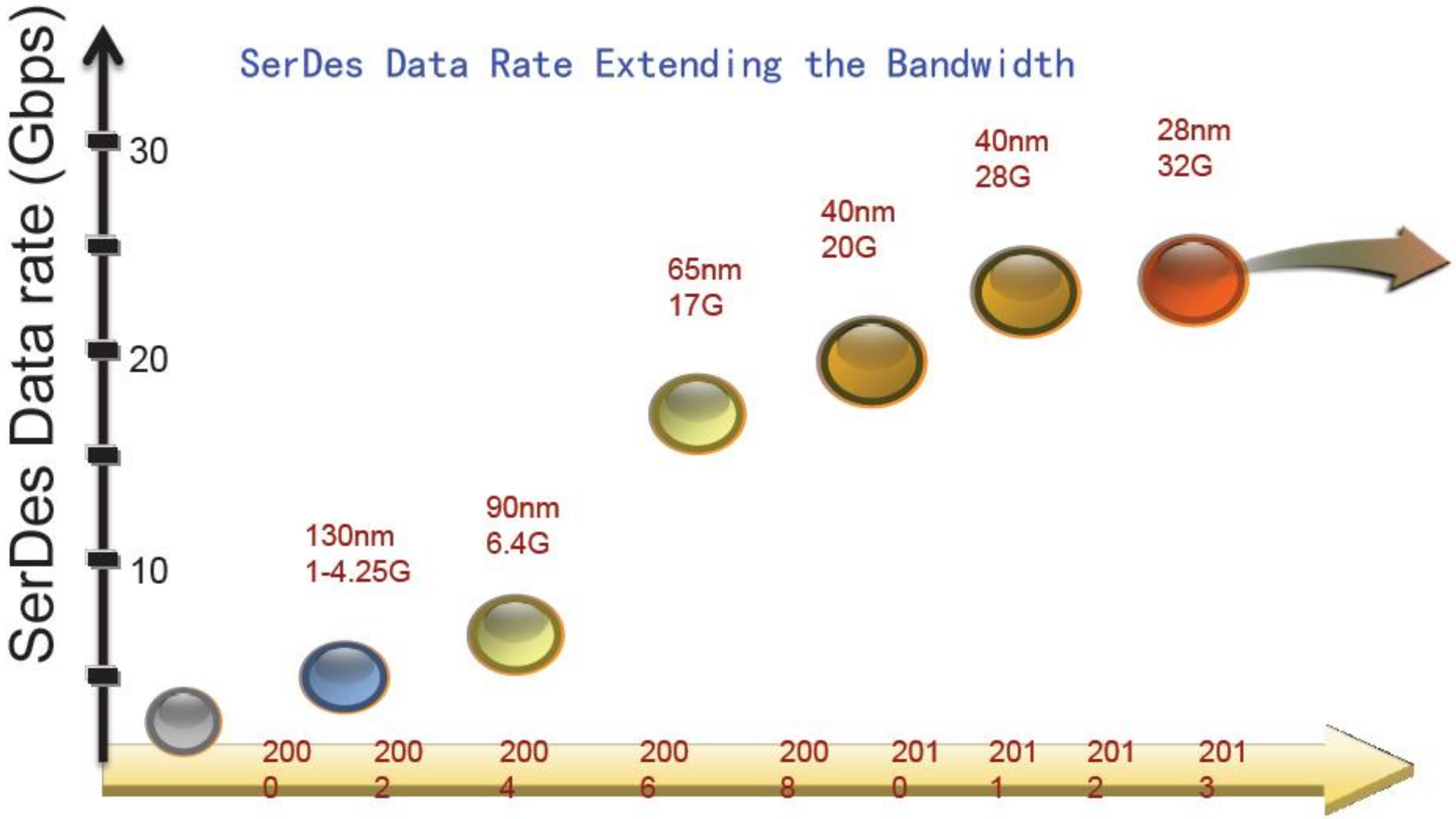
Memory Wafer Shipments Forecast



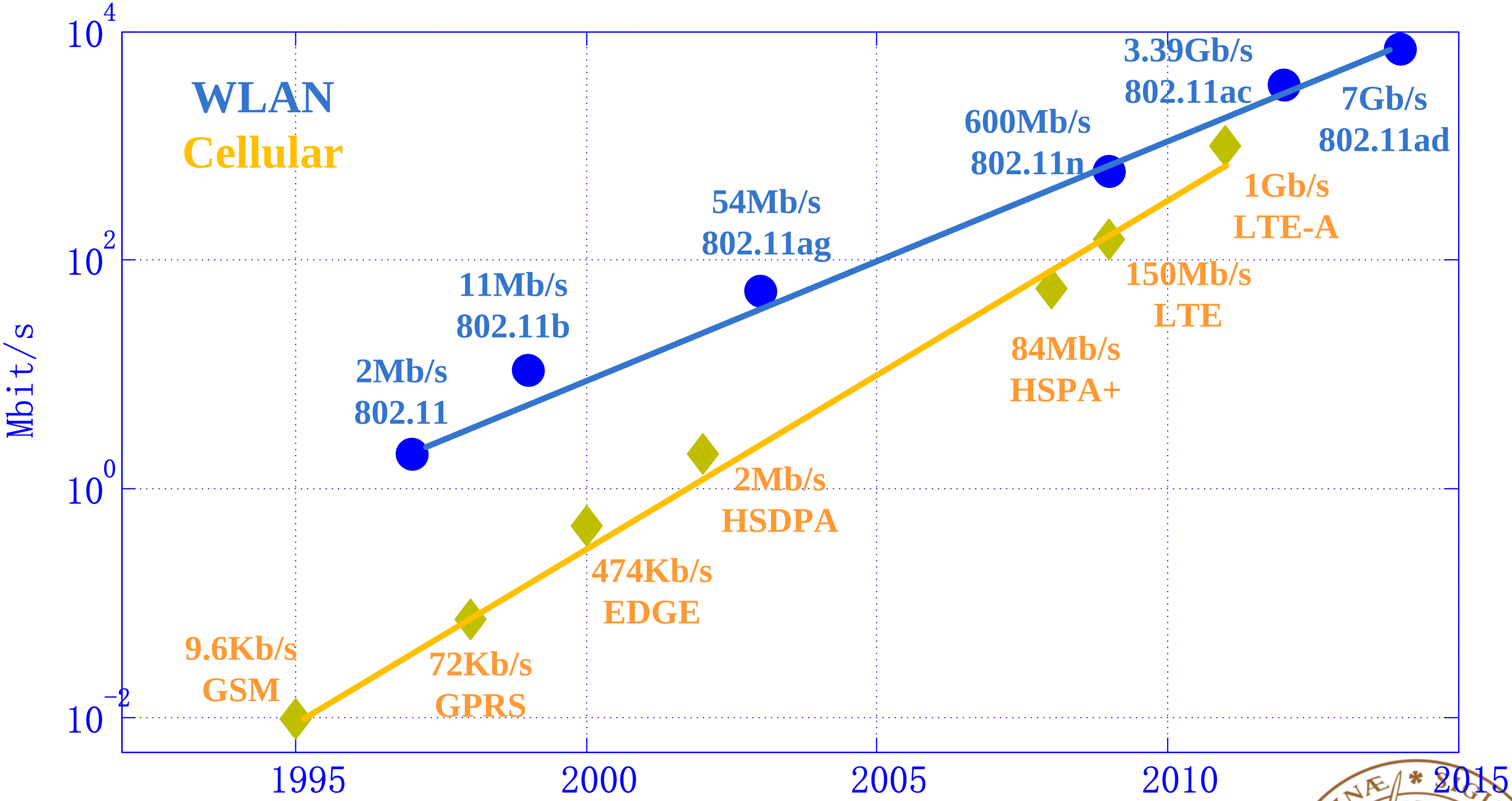
Faster-Than-Moore?

Bits shipped routinely doubles-to-triples year-over-year

Bandwidth


SerDes Data Rate Extending the Bandwidth
SerDes Data rate (Gbps)
30
20
10
130nm
1-4.25G
90nm
6.4G
65nm
17G
40nm
20G
40nm
28G
28nm
32G
200
0
200
2
200
4
200
6
200
8
201
0
201
1
201
2
201
3

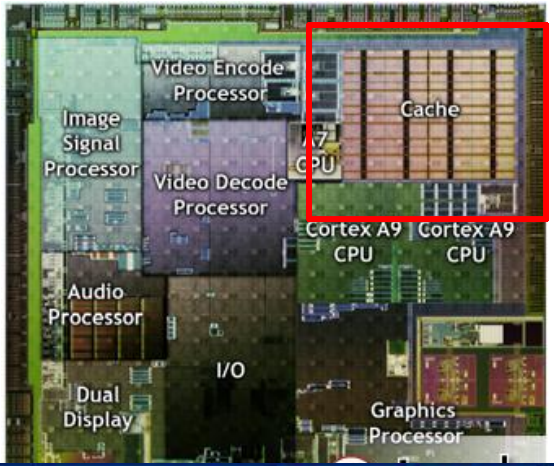
Bandwidth (cont'd.)

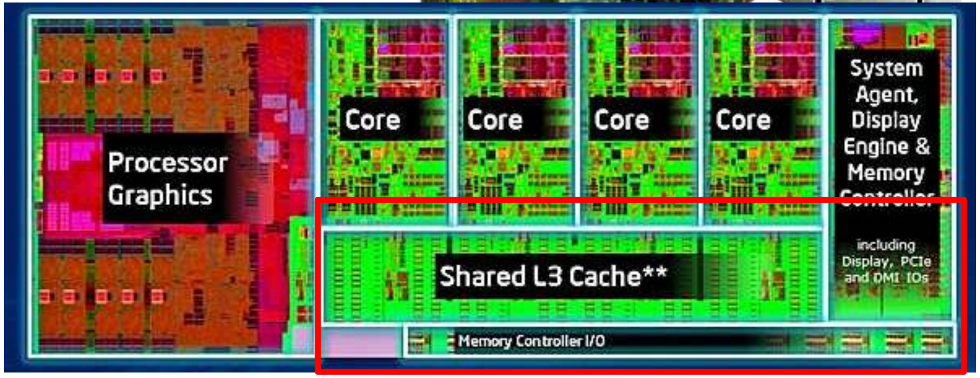


Memories, on chip

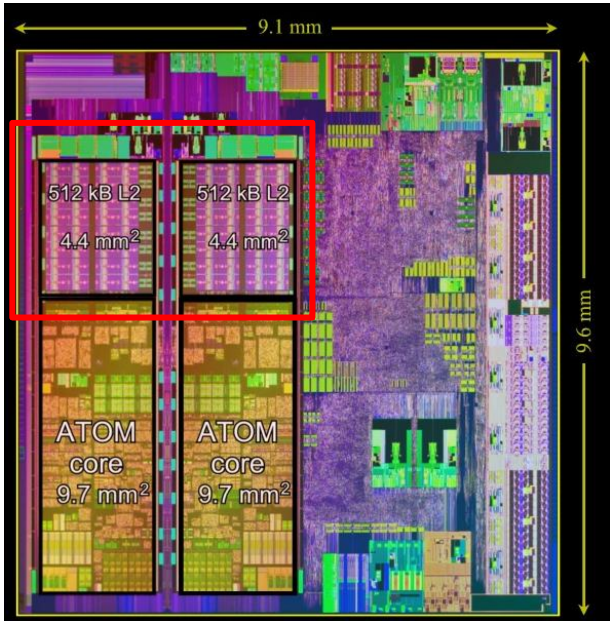
- Power and Bandwidth becomes bottleneck
- Everything is pointing to more and more “local” memory/storage at the device level

Nvidia Tegra 2




Processor Graphics
Core
Core
Core
Core
System Agent, Display Engine & Memory Controller
including Display, PCIe and DMI IOs
Shared L3 Cache**
Memory Controller I/O

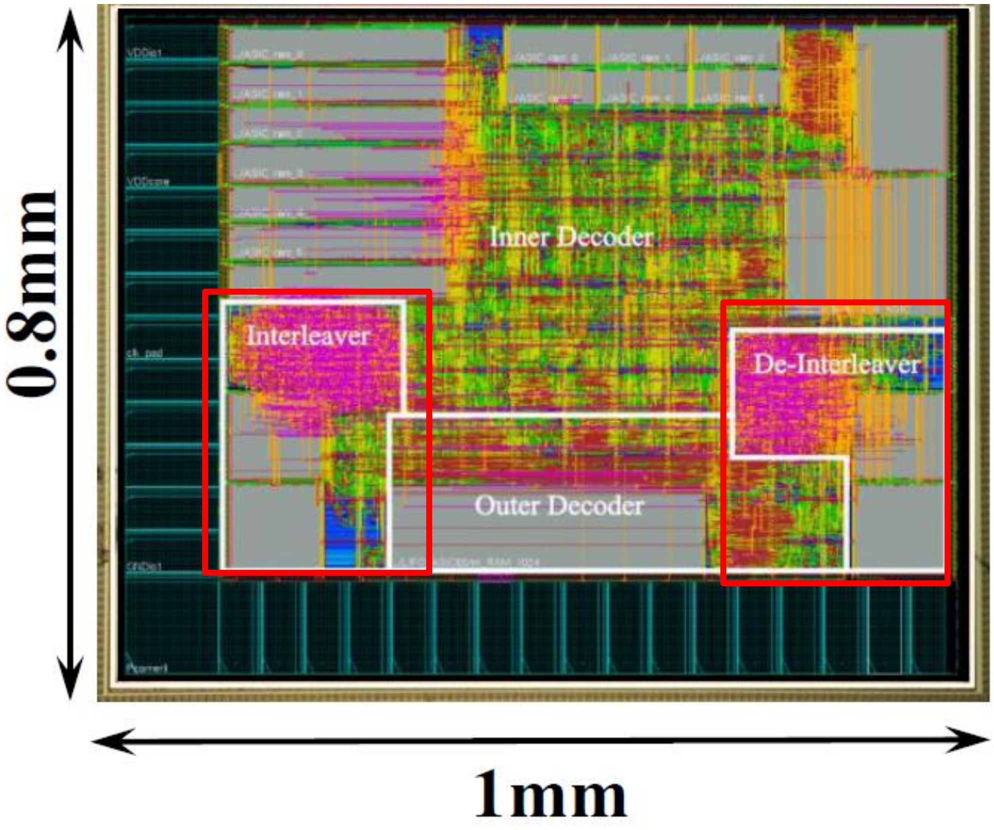
Intel Haswell

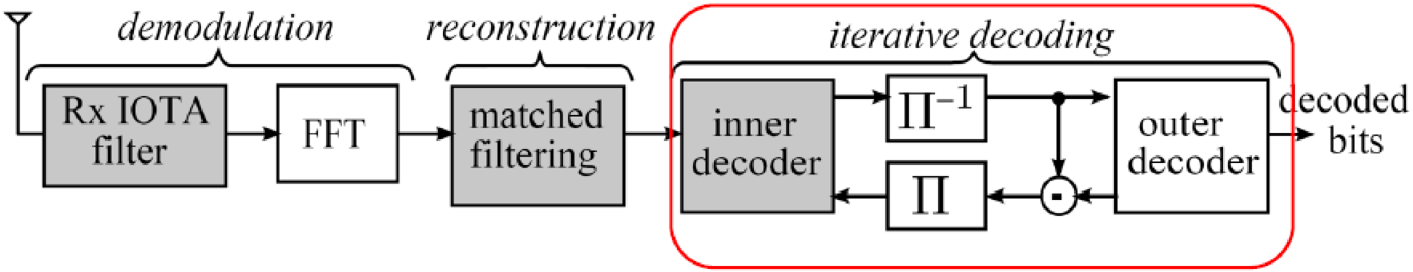


Intel ATOM

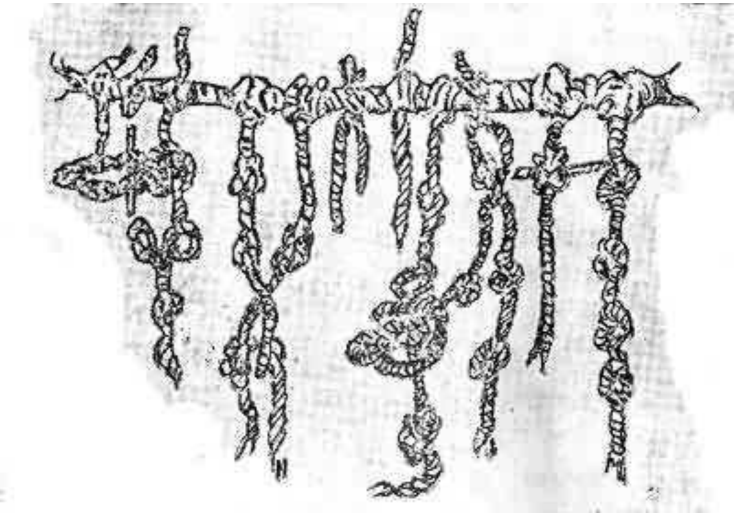
Memories, on chip

□One of our chip for wireless communication system (iterative decoder+interleaver)





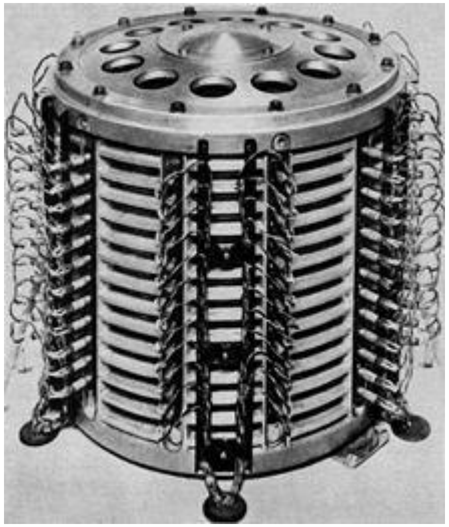
Memories, History



□**First Storage?**

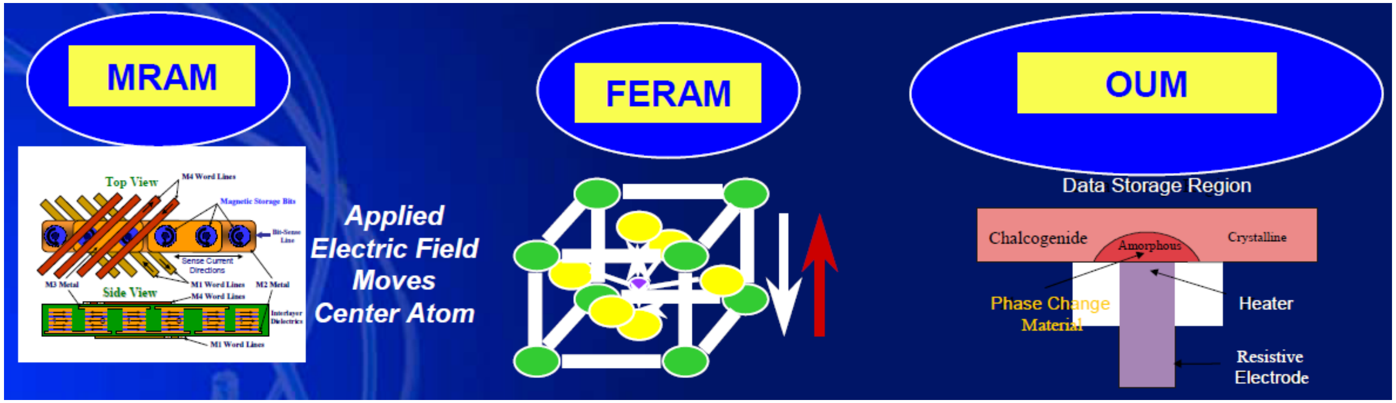
□**Early Memory**

- Drum memory: magnetic data storage device.
- Gustav Tauschek (1932)
- Widely used in the 1950s and into the 1960s as computer memory



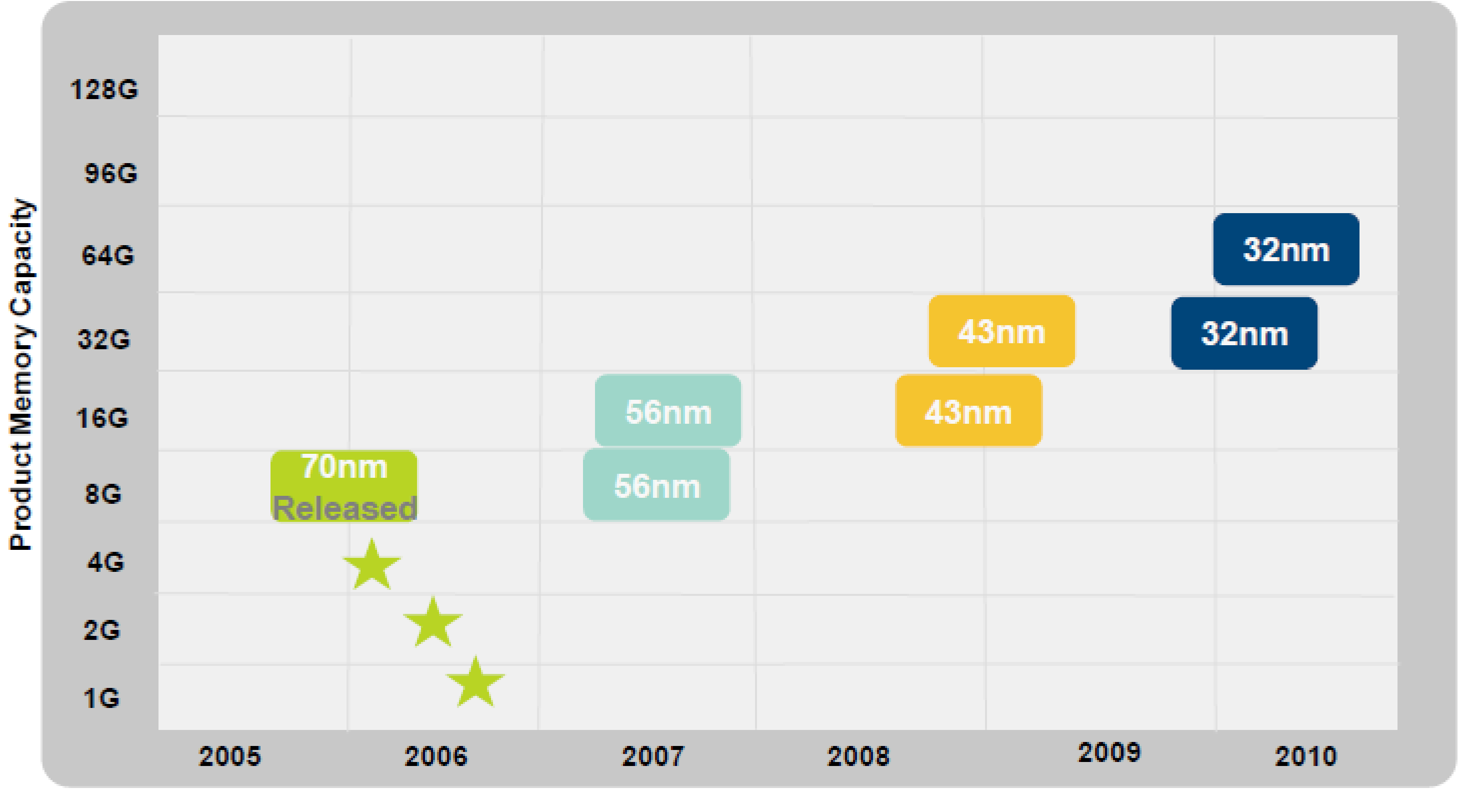
Memory, current state

- □ **Yesterday:**
  - •RAM memories are historically driven by computing applications
  - •NOR/NAND Flash is used in most of consumer devices (cell-phone, digital camera, USB stick …)
- □ **Today:**
  - •New generation memories
    - □ ***PRAM, FeRAM, MRAM..***
  - •“Solid State” memory is the killer application for NAND Flash in volume:
    - □ ***SSDs to replace HDD (hard disk magnetic drives)***
  - •RAM (SRAM / DRAM)
    - □ ***DDR3 / DDR4 DRAM***



Memory, leading the semiconductor tech.

NAND Memory Product Roadmap

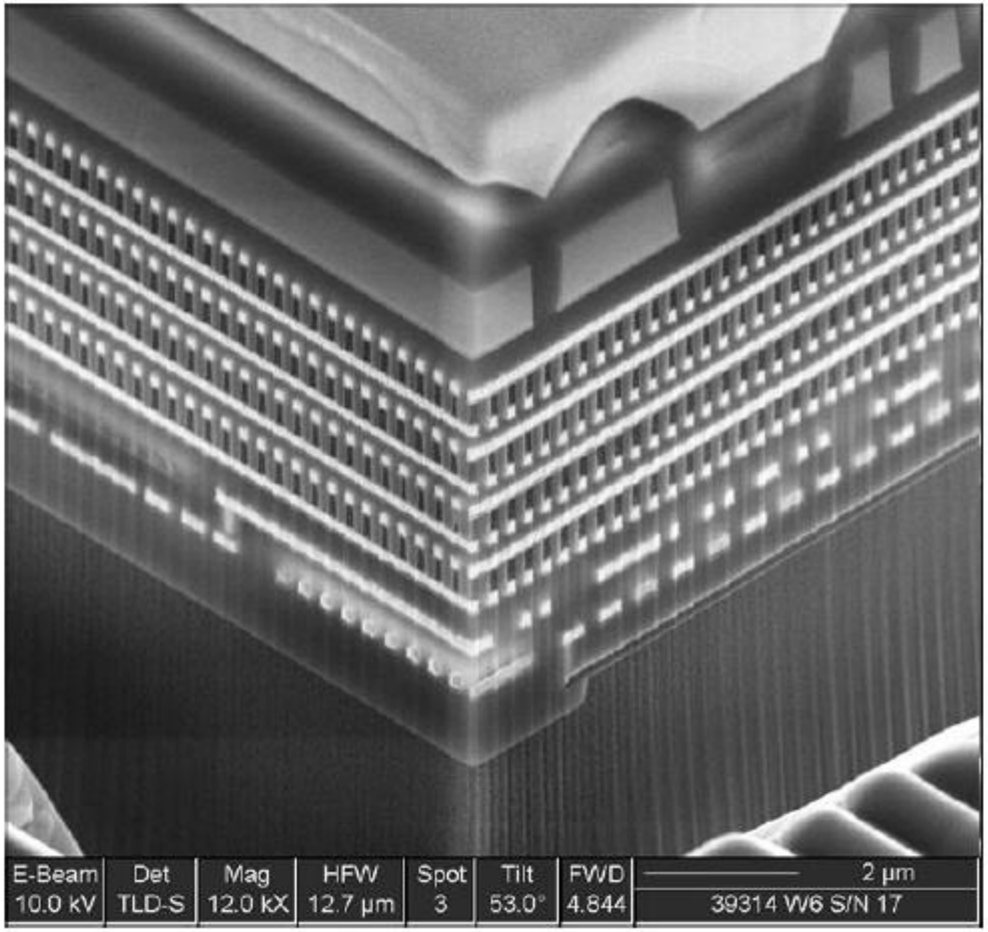


Source: SanDisk

First 32nm NAND Flash memory, 2009, Toshiba

First 32nm CPU released, 2010, Intel Core i3

Memory, leading the semiconductor tech.

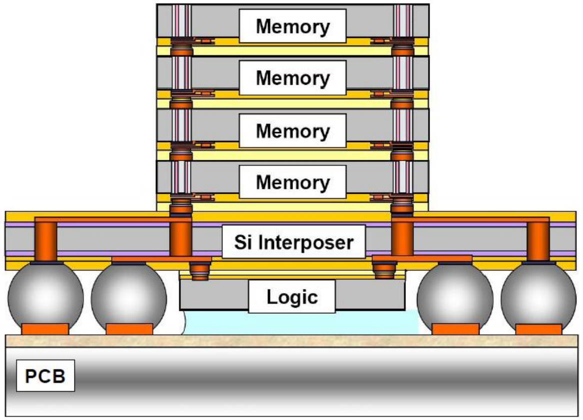


Al top metal

4 layers of memory cells + tungsten interconnect

2 levels of tungsten routing

LV + HV CMOS logic



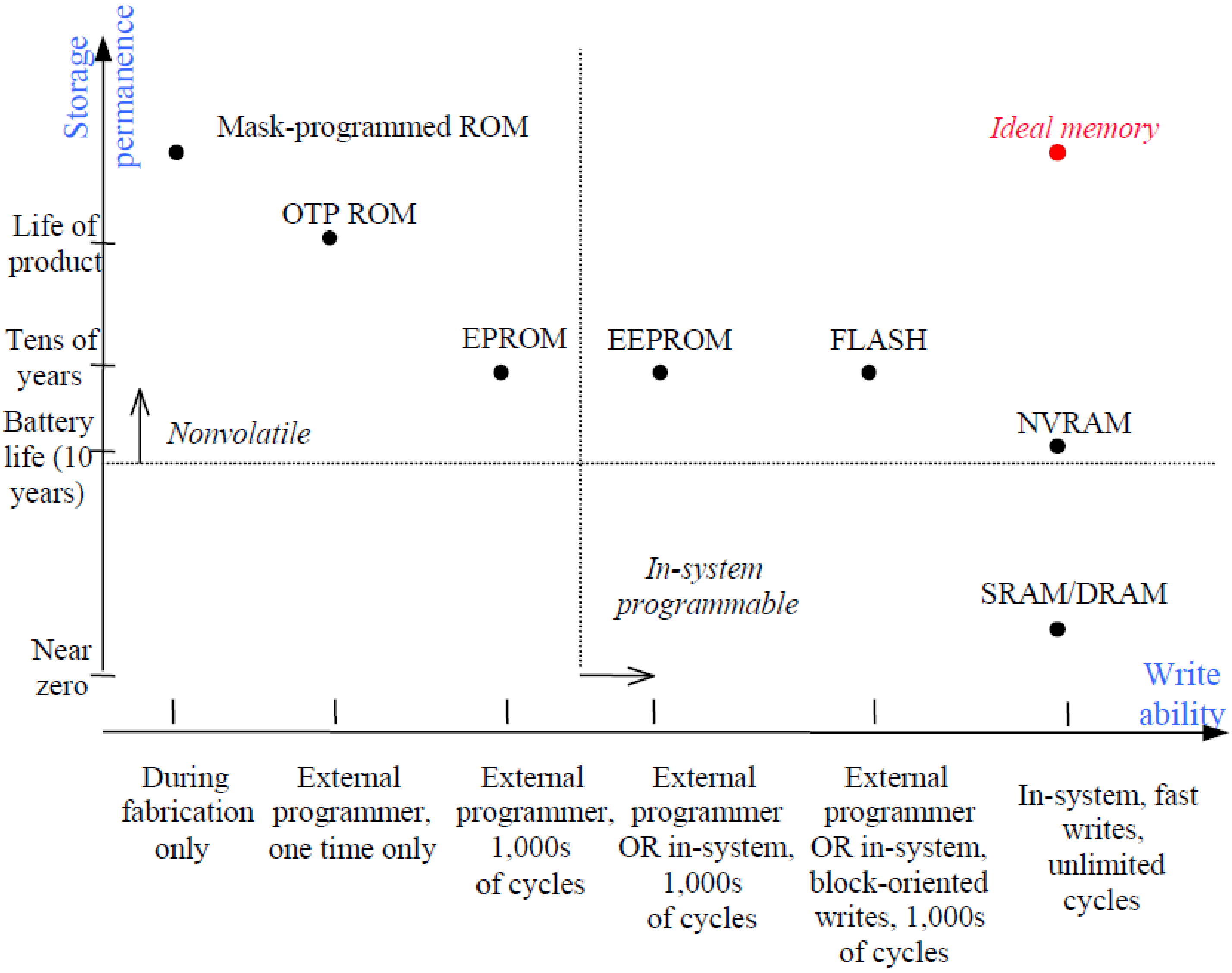
Example of 3-D integrated construction *(Image courtesy of DuPont Electronics)*

First 22-nm SRAMs using Tri-Gate transistors, in Sept.2009
First 22-nm Tri-Gate microprocessor (Ivy Bridge), released in 2013

Memory Classification

Read-Write Memory		Non-Volatile Read-Write Memory	Read-Only Memory
Random Access	Non-Random Access	EPROM E^2PROM FLASH	Mask-Programmed Programmable (PROM)
SRAM DRAM 	FIFO LIFO Shift Register CAM		

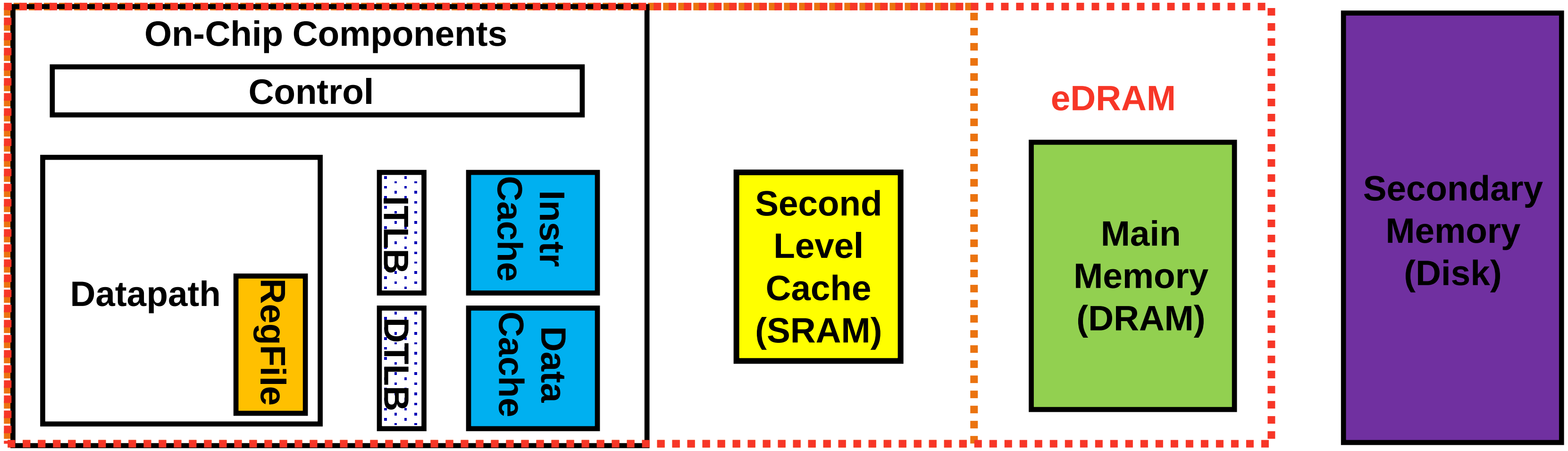
Memory Classification


Storage permanence
Mask-programmed ROM
Ideal memory
Life of product
OTP ROM
Tens of years
EPROM
EEPROM
FLASH
Battery life (10 years)
Nonvolatile
NVRAM
In-system programmable
SRAM/DRAM
Near zero
Write ability
During fabrication only
External programmer, one time only
External programmer, 1,000s of cycles
External programmer OR in-system, 1,000s of cycles
External programmer OR in-system, block-oriented writes, 1,000s of cycles
In-system, fast writes, unlimited cycles

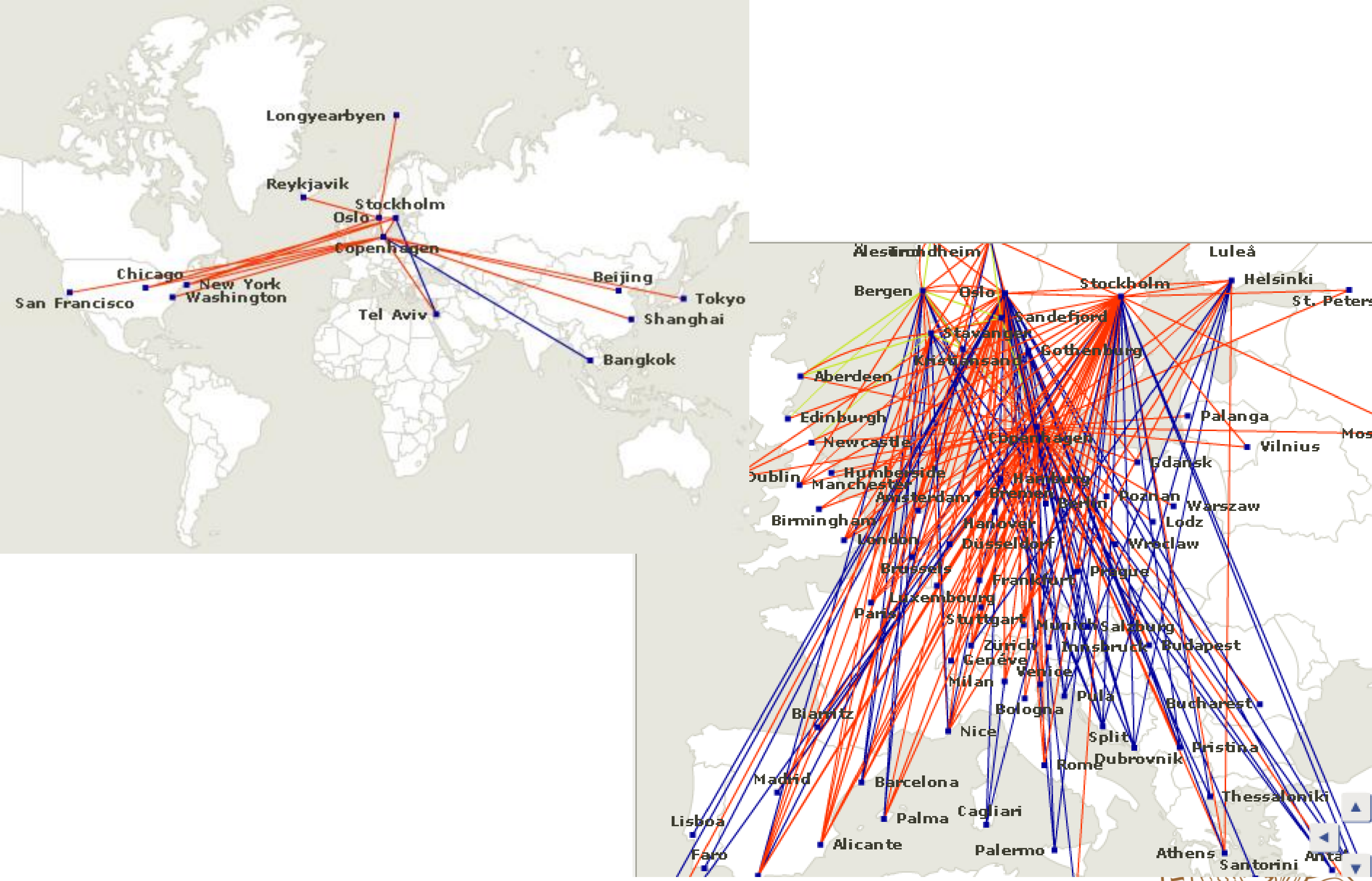
Picture from Embedded Systems Design: A Unified Hardware/Software Introduction

Memory Hierarchy

Speed (ns):	.1's	1's	10's	100's	1,000's
Size (bytes):	100's	K's	10K's	M's	T's
Cost:	highest				lowest



Hierarchy, Heterogeneous



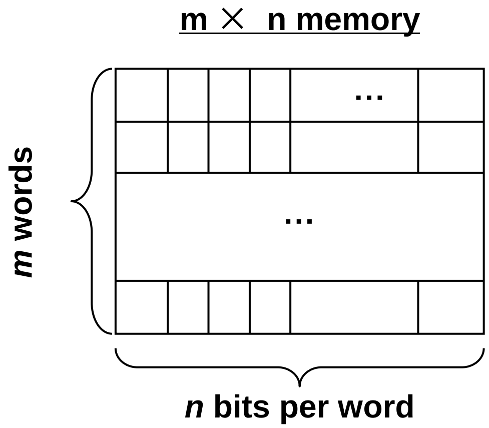
Memory Basic Concept

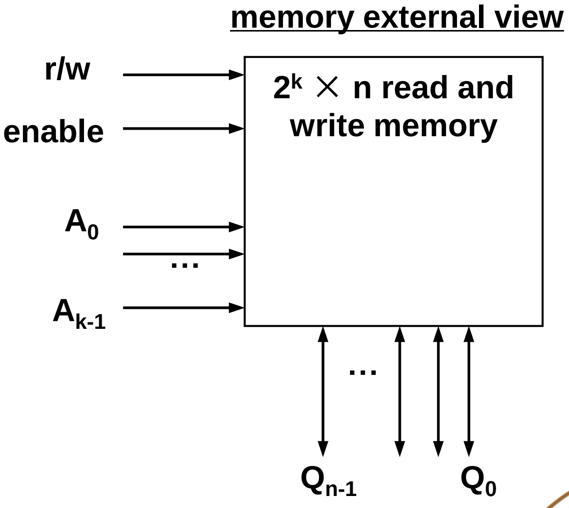
□ Stores large number of bits

- m x n: m words of n bits each
- k = Log_2(m) address input signals
- or m = 2^k words
- e.g., 4096 x 8 memory:
  - □ ***32,768 bits***
  - □ ***12 address input signals***
  - □ ***8 input/output data signals***

□ Memory access

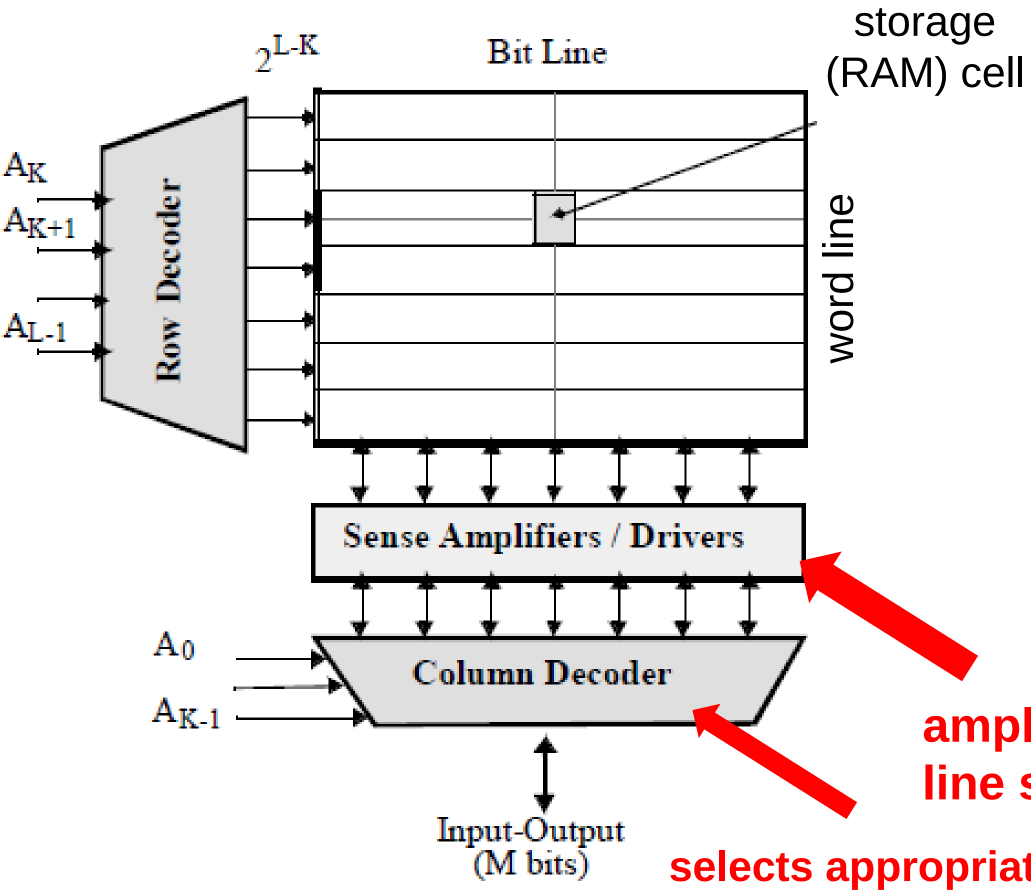
- r/w: selects read or write
- enable: read or write only when asserted
- Address
- Data-port

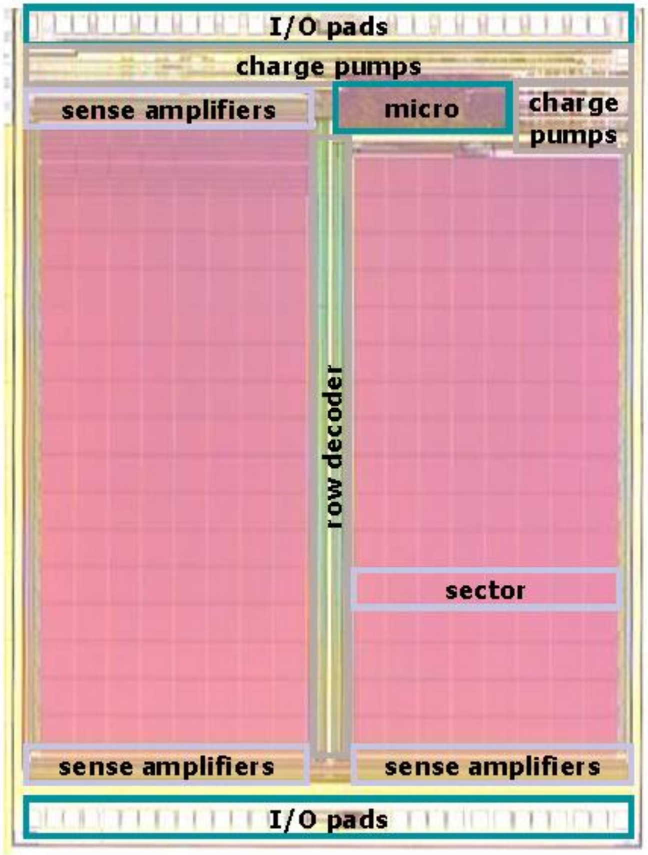




We stay at higher-level, gate-level view of memory will be taught at Digital IC Design

Memory Architecture





amplifies bit line swing

selects appropriate word from memory row

Outline

□Overview of Memory

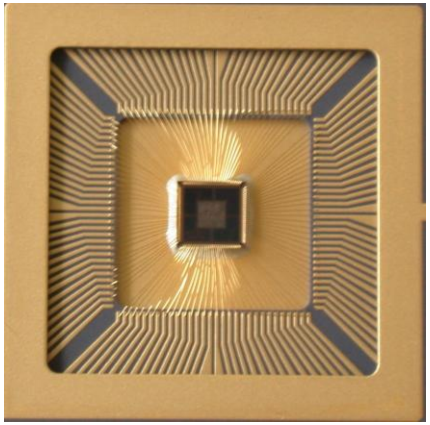
- Application, history, trend
- Different memory type
- Overall architecture

□Registers as Storage Element

- Register File
- FIFO

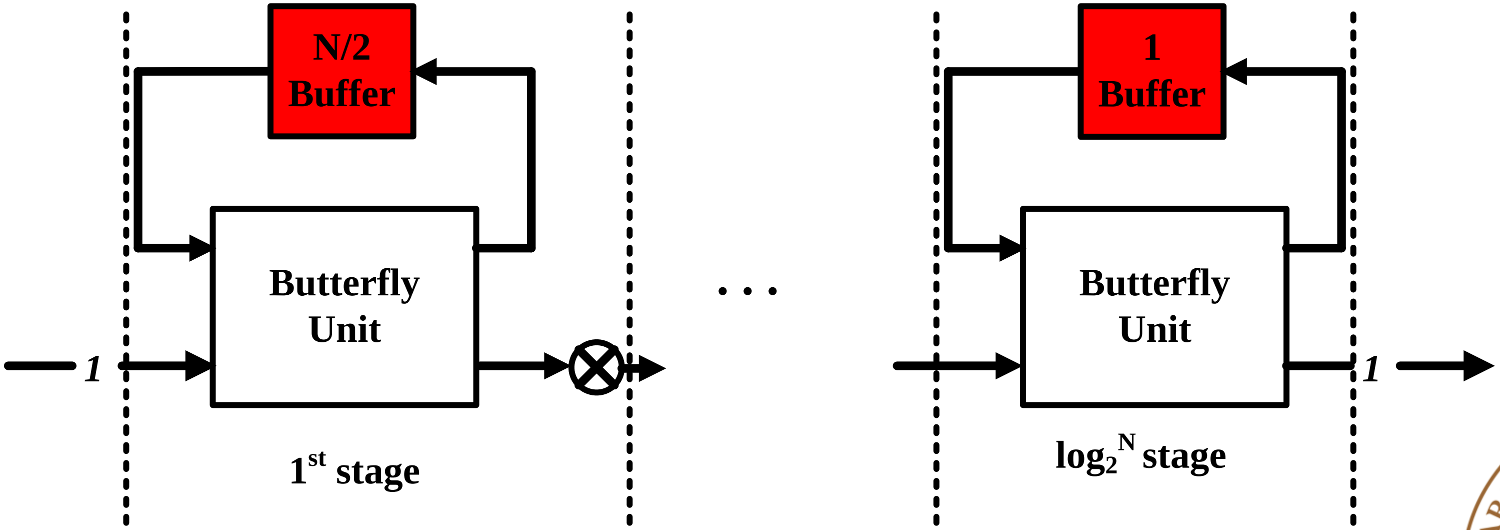
□Xilinx Storage Elements

Storage Examples 1



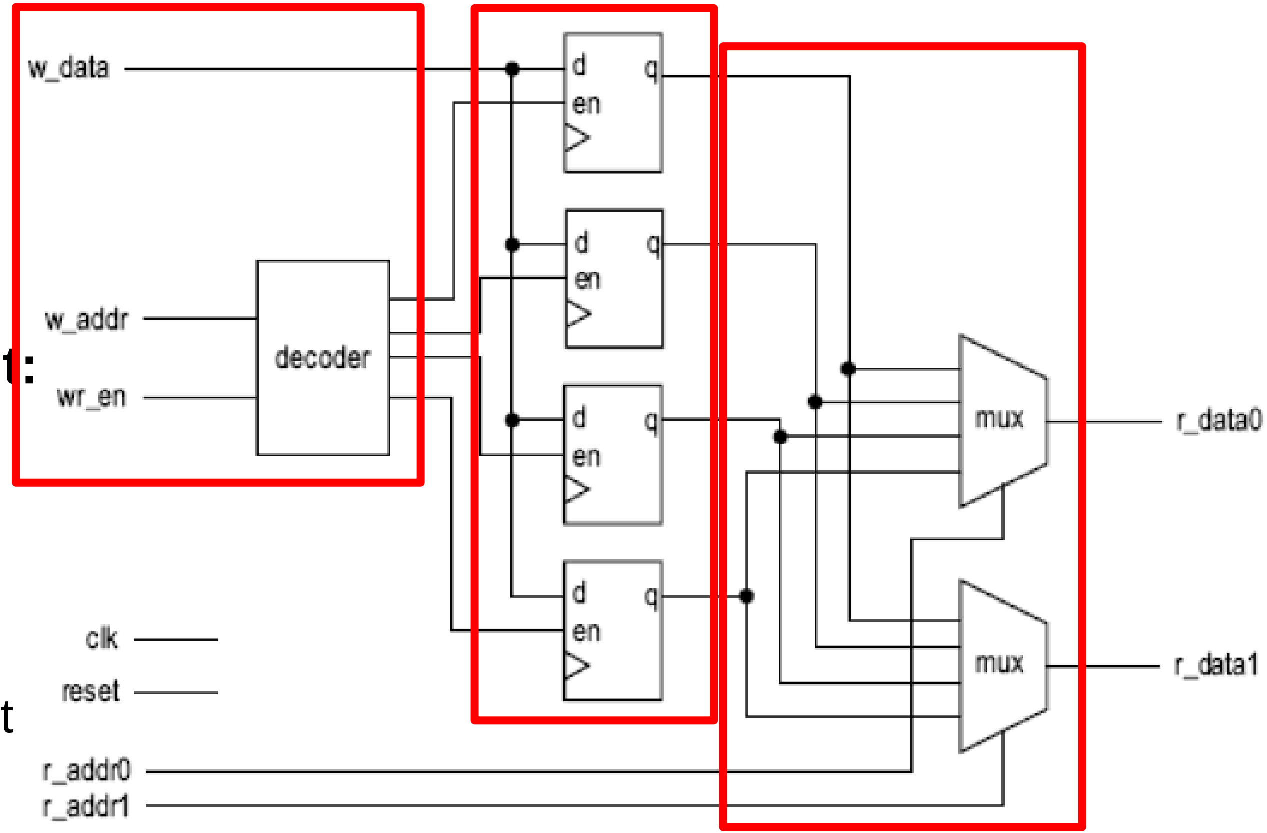
□Register File

- Used as ***fast temporary*** storage
- Registers arranged as ***array***
- Each register is identified with an address
- Normally has 1 write port (with write enable signal)
- Can has multiple read ports



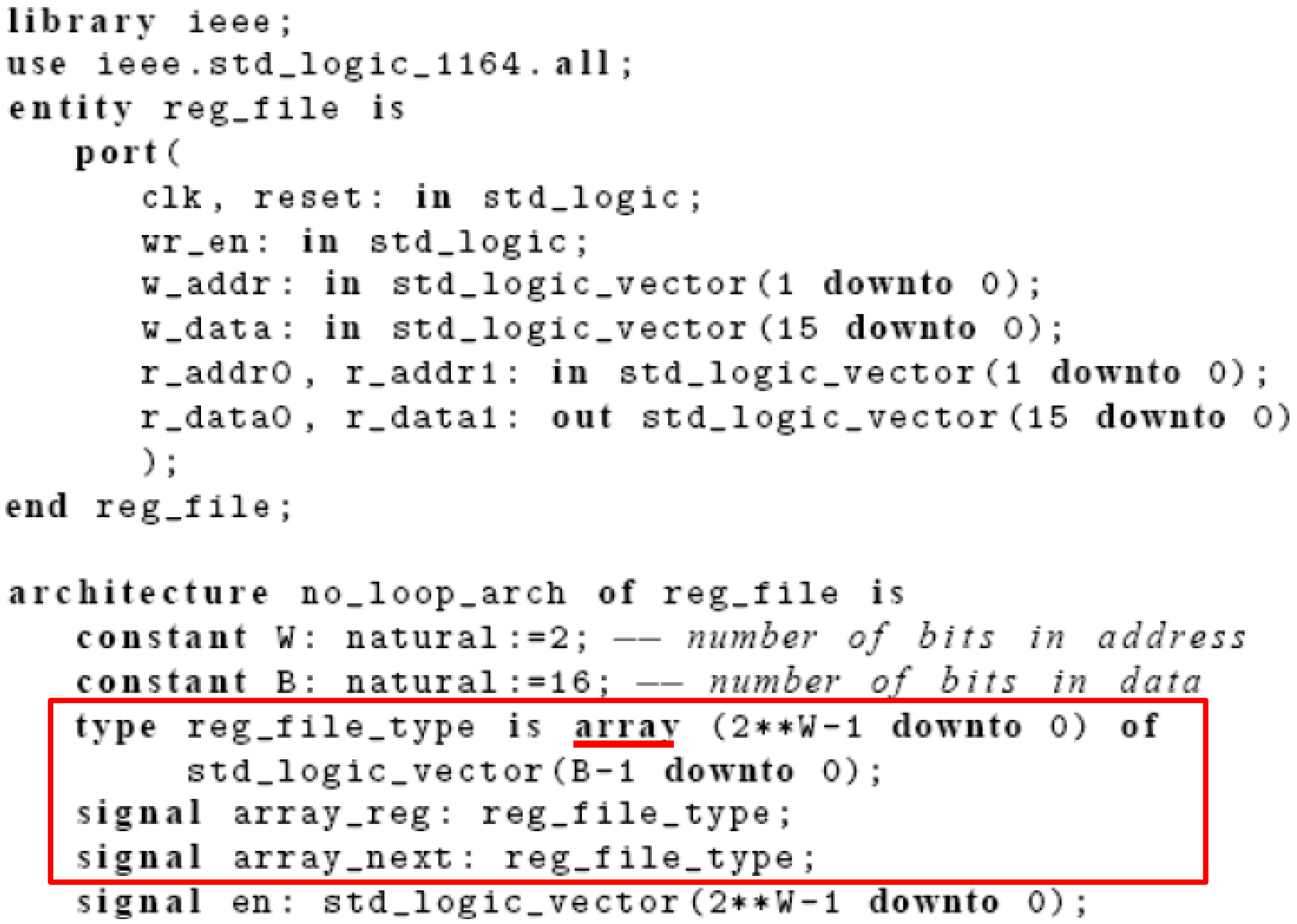
Register File

- **Example:** 4-word register file with 1 write port and two read ports



- **Register array:**
  - 4*16bit registers
  - Each register has an enable signal
- **Write decoding circuit:**
  - 0000 if wr_en is 0
  - 1 bit asserted according to w_addr if wr_en is 1
- **Read circuit:**
  - A mux for each read port

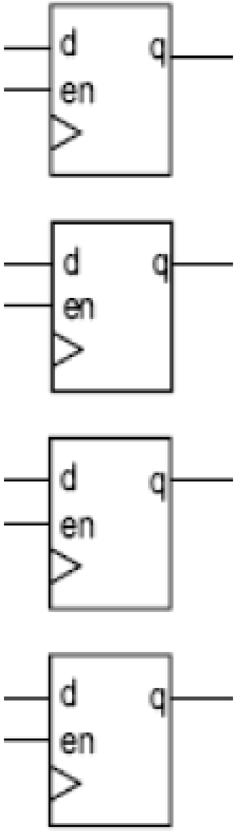
VHDL: a parameterized 2^W-by-B register file

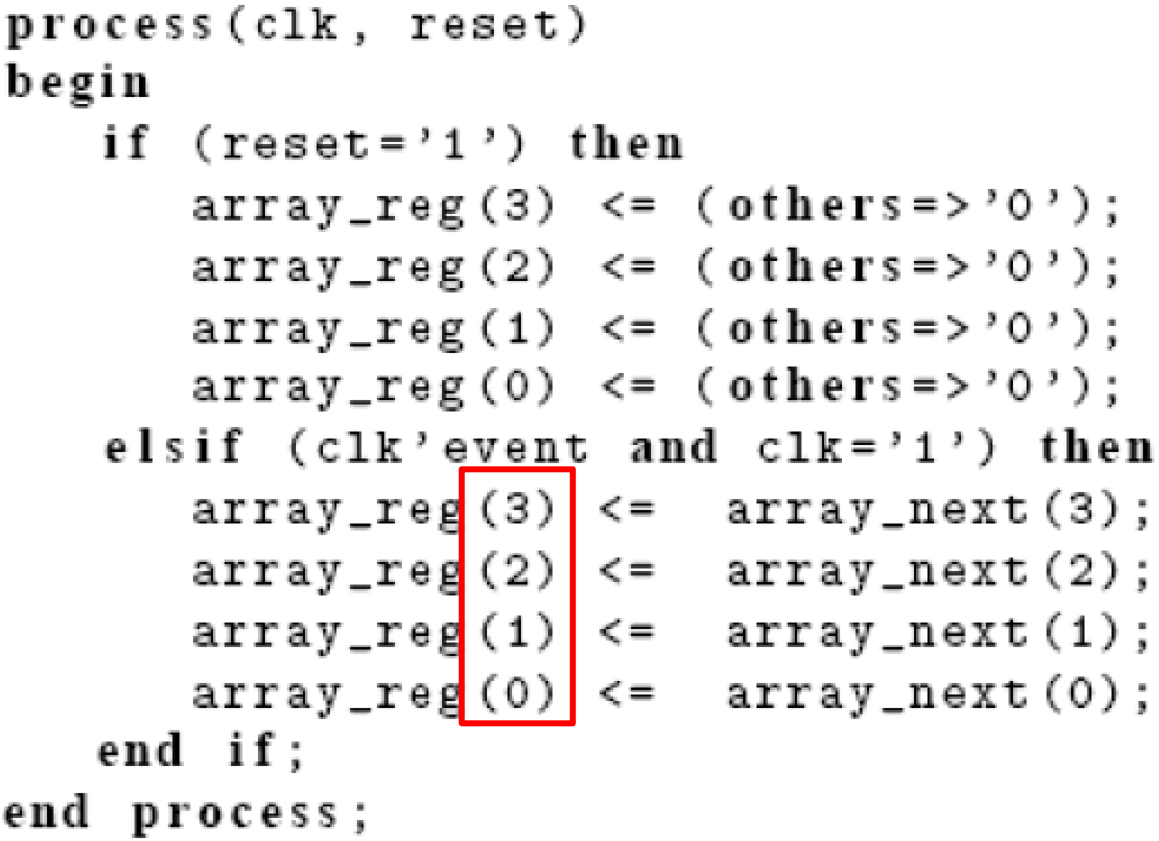


```
library ieee;
use ieee.std_logic_1164.all;
entity reg_file is
   port(
      clk, reset: in std_logic;
      wr_en: in std_logic;
      w_addr: in std_logic_vector(1 downto 0);
      w_data: in std_logic_vector(15 downto 0);
      r_addr0, r_addr1: in std_logic_vector(1 downto 0);
      r_data0, r_data1: out std_logic_vector(15 downto 0)
      );
end reg_file;

architecture no_loop_arch of reg_file is
   constant W: natural:=2; -- number of bits in address
   constant B: natural:=16; -- number of bits in data
   type reg_file_type is array (2**W-1 downto 0) of
        std_logic_vector(B-1 downto 0);
   signal array_reg: reg_file_type;
   signal array_next: reg_file_type;
   signal en: std_logic_vector(2**W-1 downto 0);
```

A user-defined array-of-array data type is introduced

VHDL: a parameterized 2^W-by-B register file





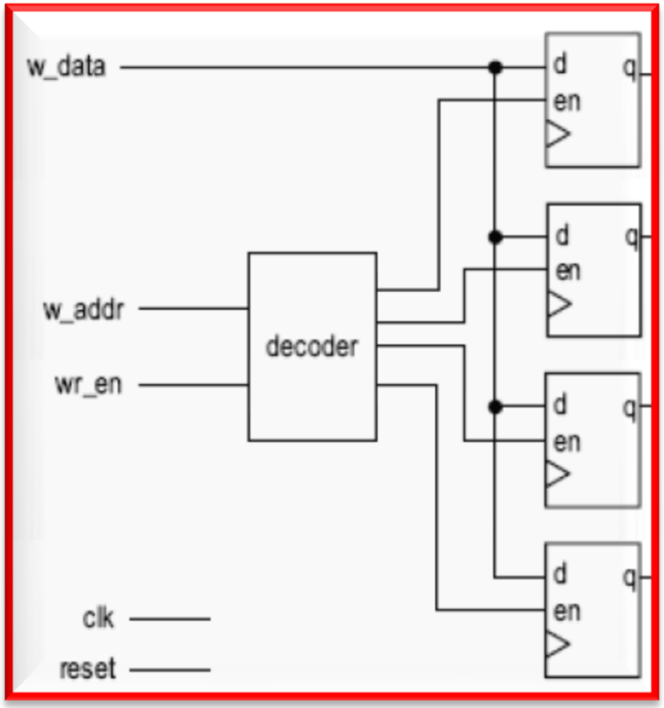
```
process(clk, reset)
begin
    if (reset='1') then
        array_reg(3) <= (others=>'0');
        array_reg(2) <= (others=>'0');
        array_reg(1) <= (others=>'0');
        array_reg(0) <= (others=>'0');
    elsif (clk'event and clk='1') then
        array_reg(3) <=  array_next(3);
        array_reg(2) <=  array_next(2);
        array_reg(1) <=  array_next(1);
        array_reg(0) <=  array_next(0);
   end if;
end process;
```

Index to access an element in the array

- *s(i)* to access the ith row of the array s
- *S(i)(j)* to access the jth element of ith row in the array

VHDL: a parameterized 2^W-by-B register file

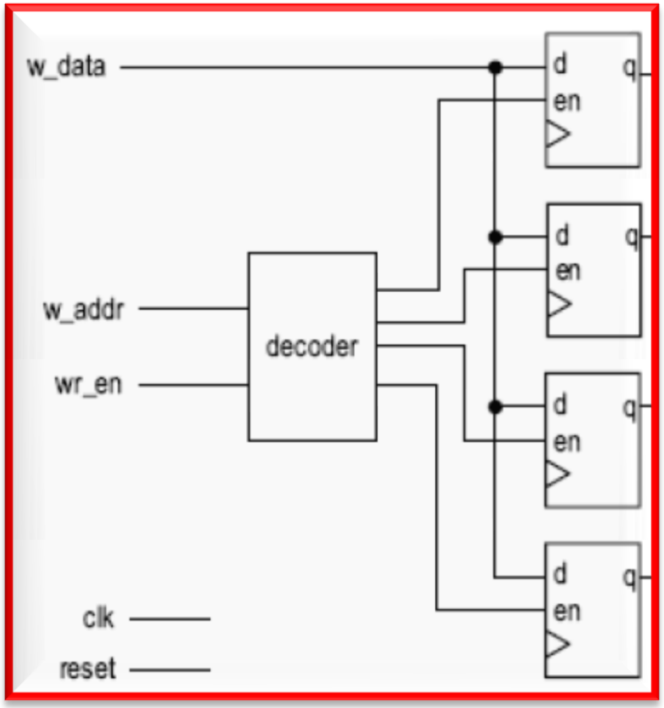
Enable logic for register

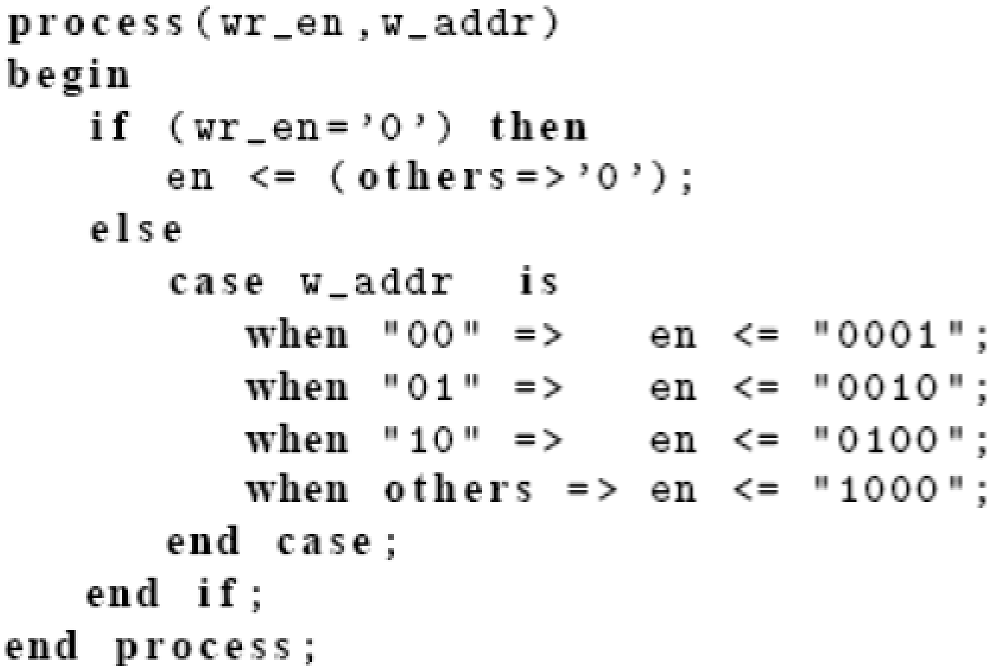


```
process(array_reg, en, w_data)
begin
   array_next(3) <= array_reg(3);
   array_next(2) <= array_reg(2);
   array_next(1) <= array_reg(1);
   array_next(0) <= array_reg(0);
   if en(3)='1' then
      array_next(3) <= w_data;
   end if;

   if en(2)='1' then
      array_next(2) <= w_data;
   end if;
   if en(1)='1' then
      array_next(1) <= w_data;
   end if;
   if en(0)='1' then
      array_next(0) <= w_data;
   end if;
end process;
```

VHDL: a parameterized 2^W-by-B register file

Enable logic for register (Cont.)

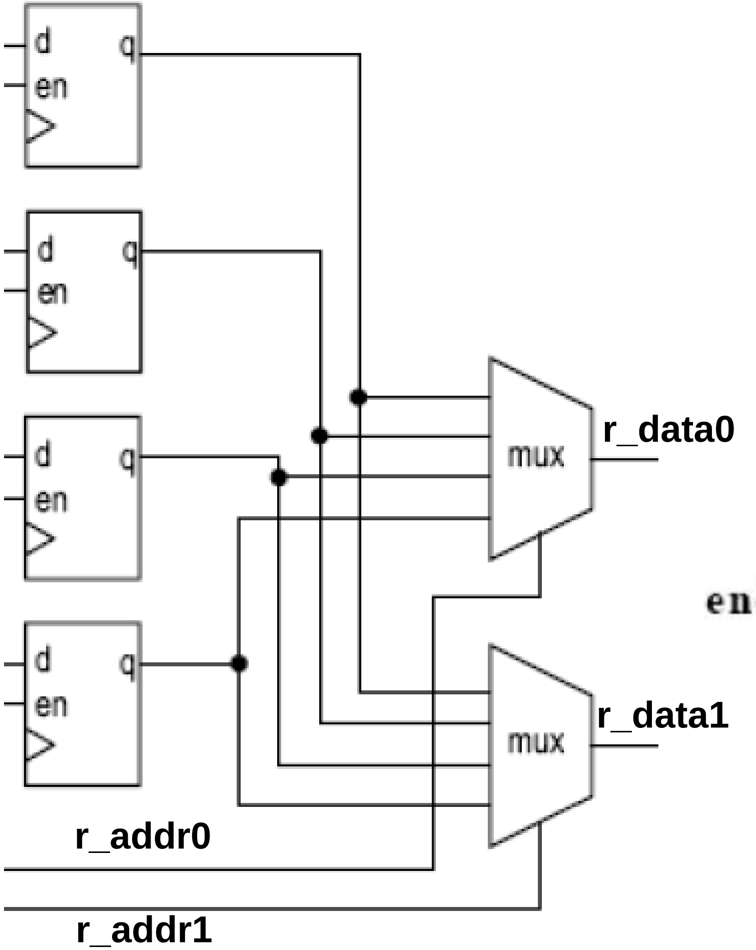




```
process(wr_en,w_addr)
begin
   if (wr_en='0') then
      en <= (others=>'0');
   else
      case w_addr  is
         when "00" =>   en <= "0001";
         when "01" =>   en <= "0010";
         when "10" =>   en <= "0100";
         when others => en <= "1000";
      end case;
   end if;
end process;
```

VHDL: a parameterized 2^W-by-B register file

Read Multiplexing

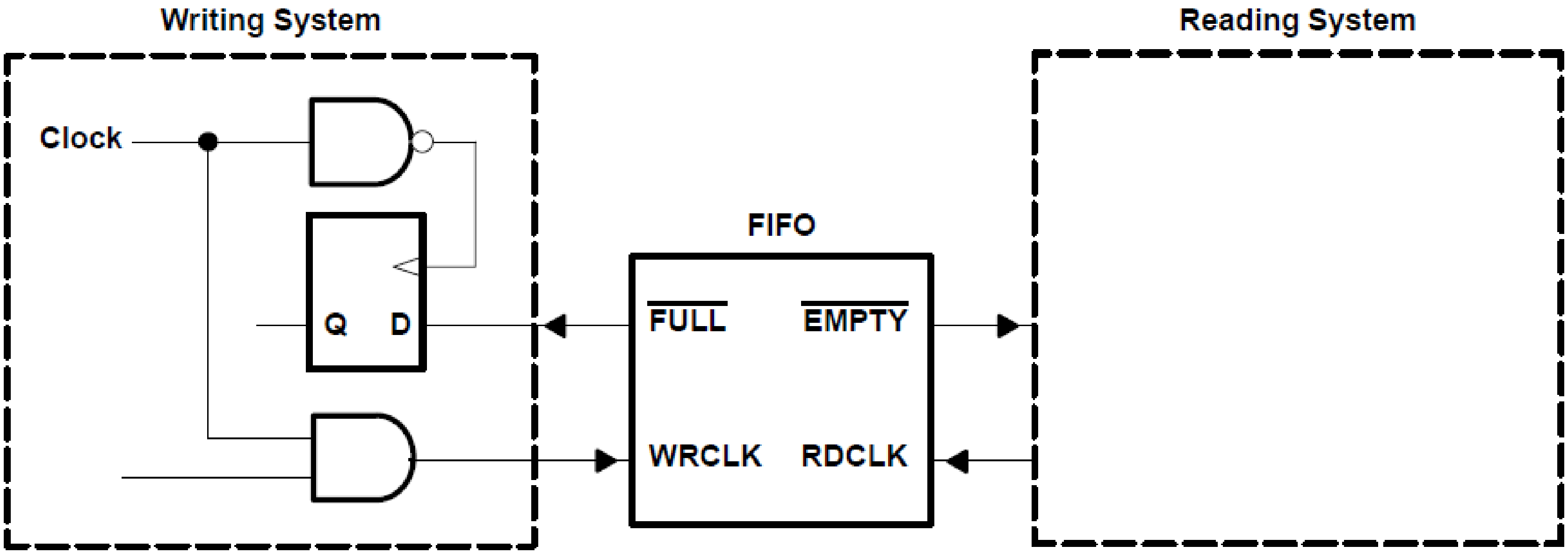


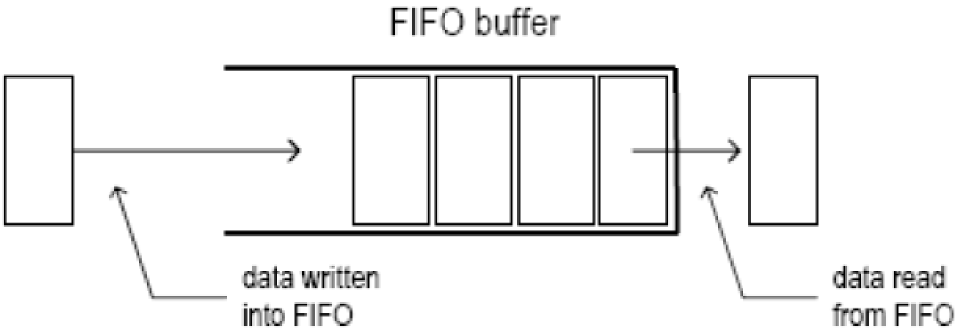
```
with r_addr0 select
   r_data0 <=  array_reg(0) when "00",
               array_reg(1) when "01",
               array_reg(2) when "10",
               array_reg(3) when others;
with r_addr1 select
   r_data1 <=  array_reg(0) when "00",
               array_reg(1) when "01",
               array_reg(2) when "10",
               array_reg(3) when others;
end no_loop_arch;
```

Storage Examples 2

FIFO (first in first out) Buffer

- “Elastic” storage between two subsystems

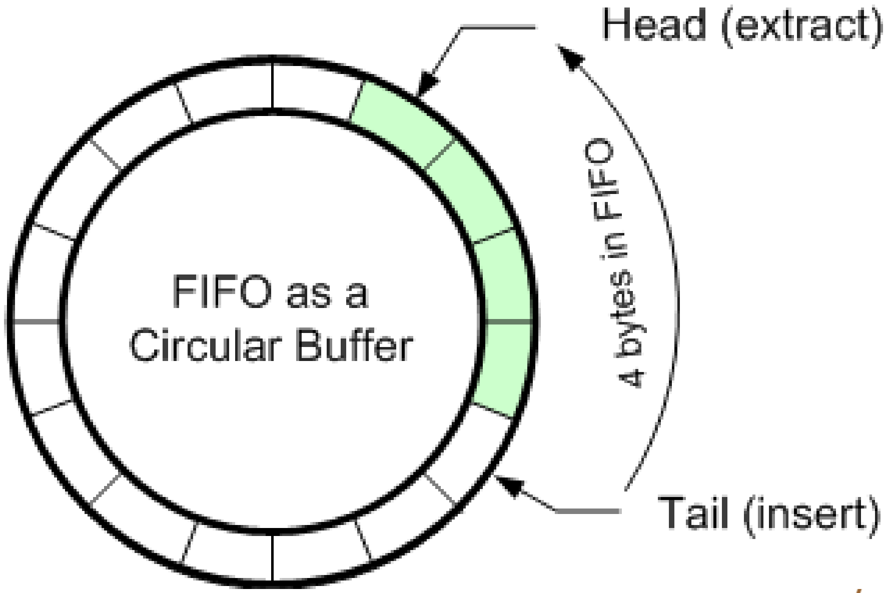




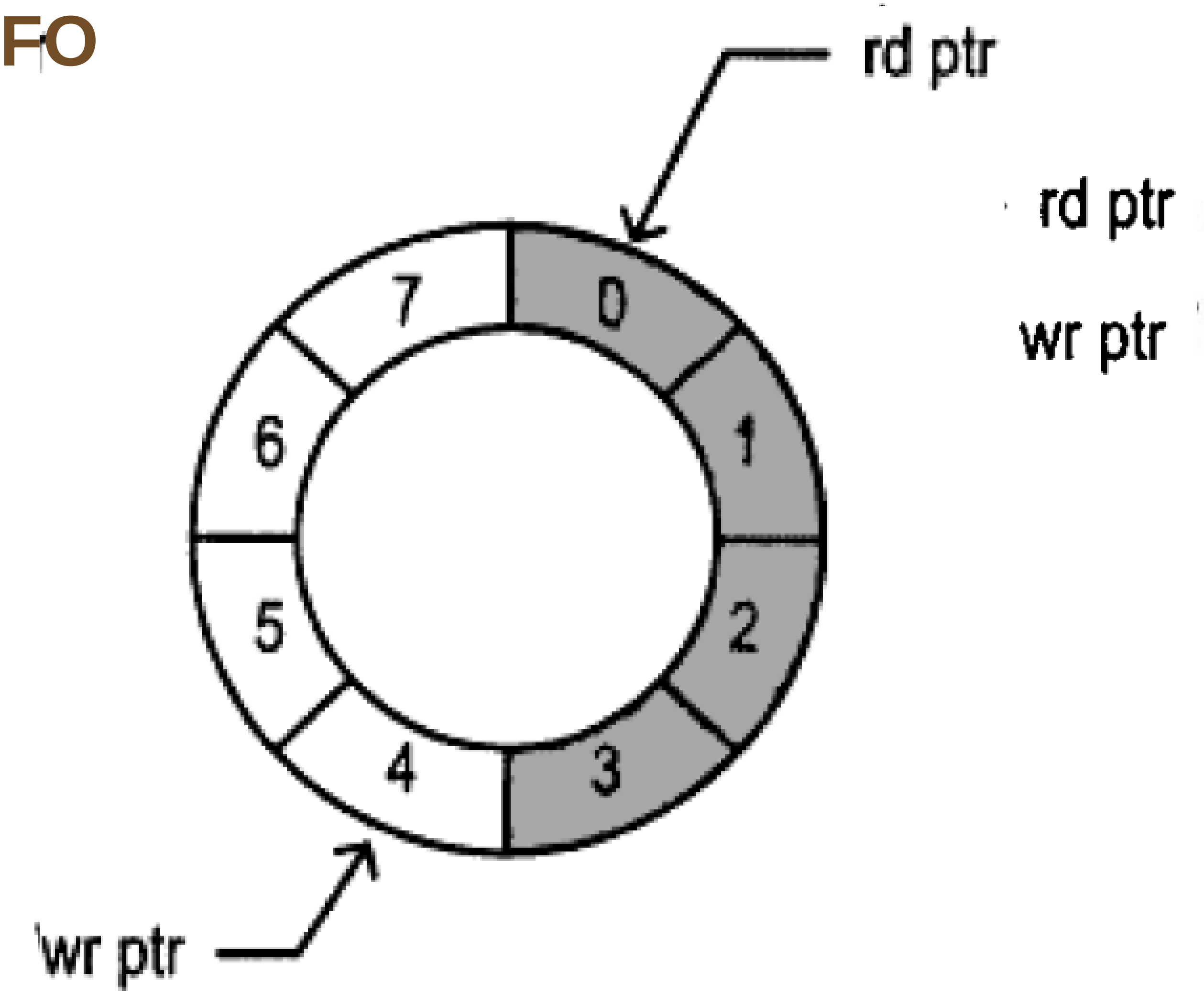
Circular FIFO

□ How to Implement a FIFO?

- Circular queue implementation
- Use two pointers and a “generic storage”
  - □ ***Write pointer*: point to the empty slot before the *head* of the queue**
  - □ ***Read pointer*: point to the *tail* of the queue**



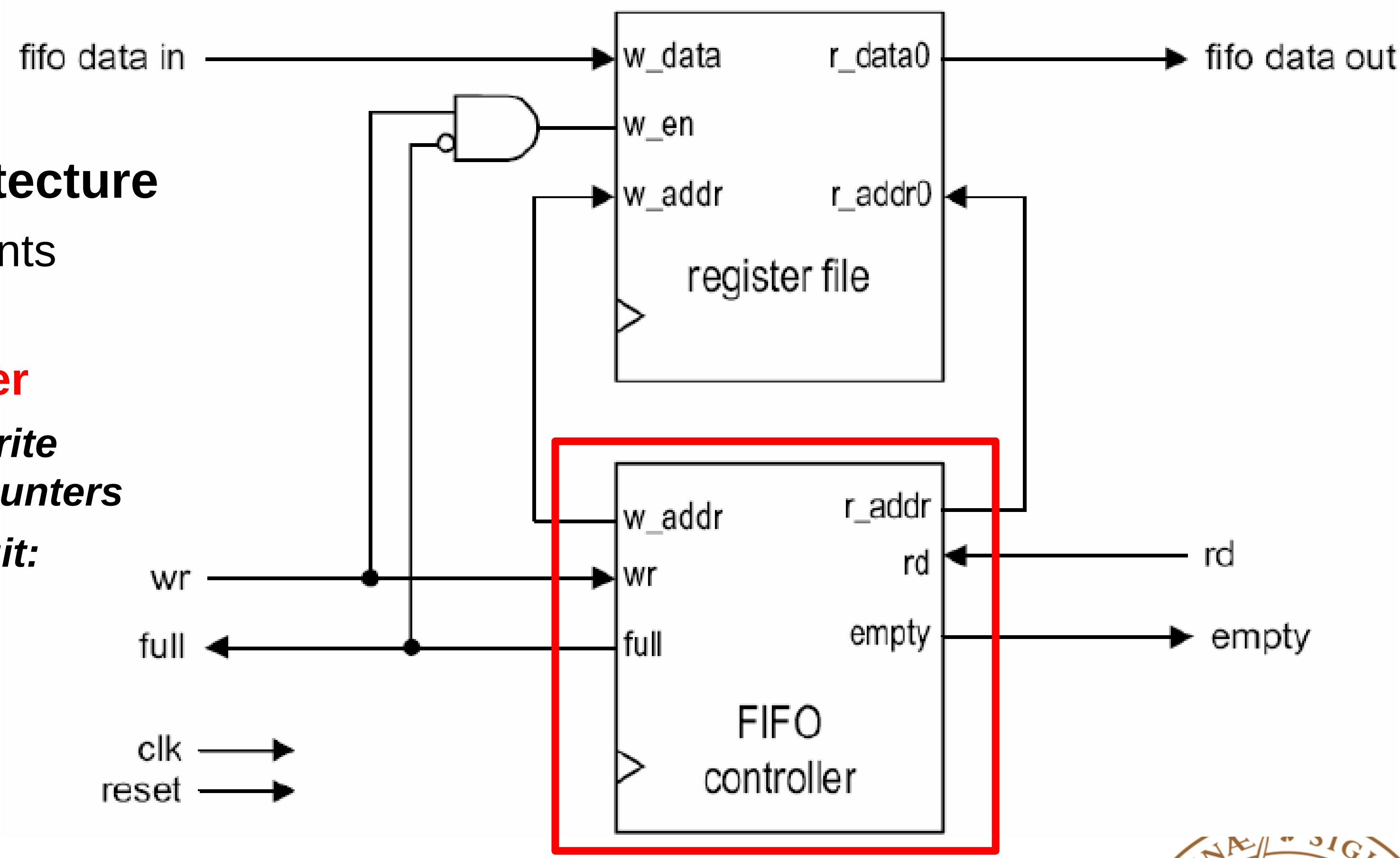
Circular FIFO



(c). 4 more writes

FIFO Implementation

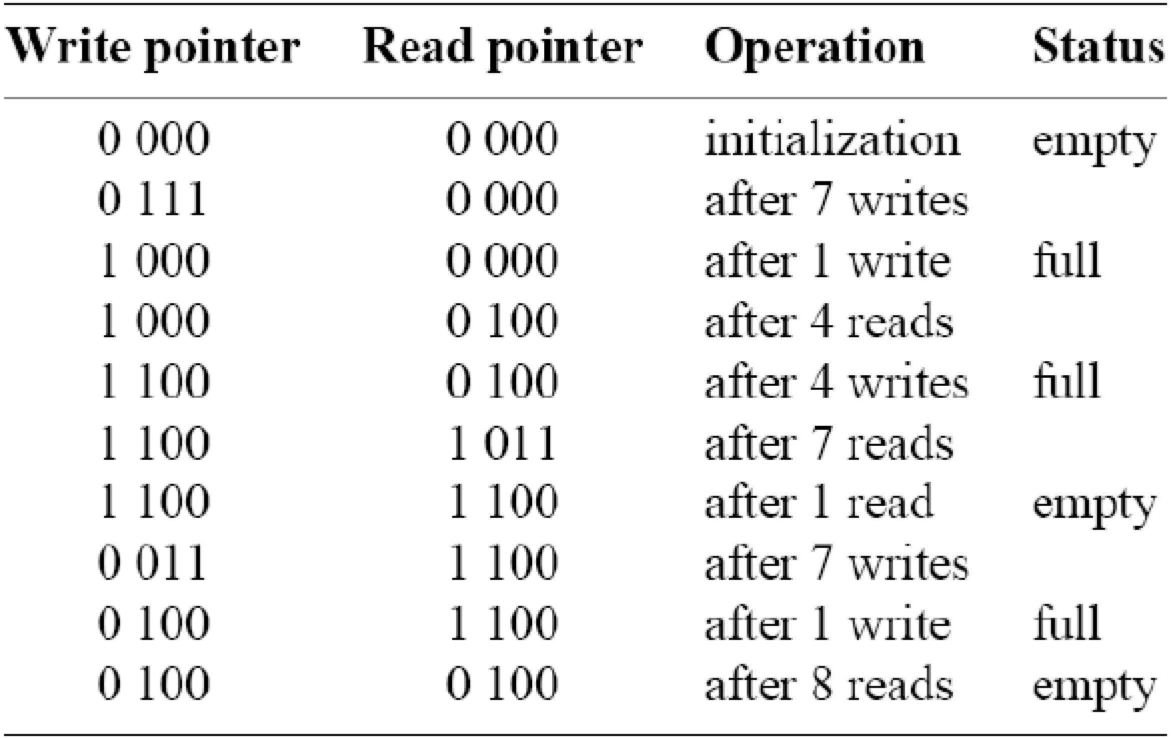
- □ **Overall Architecture**
  - Storage Elements
    - □ ***Reg. file***
  - FIFO Controller
    - □ ***Read and write pointers: 2 counters***
    - □ ***Status circuit:*** ***full, empty***



FIFO Implementation: Controller

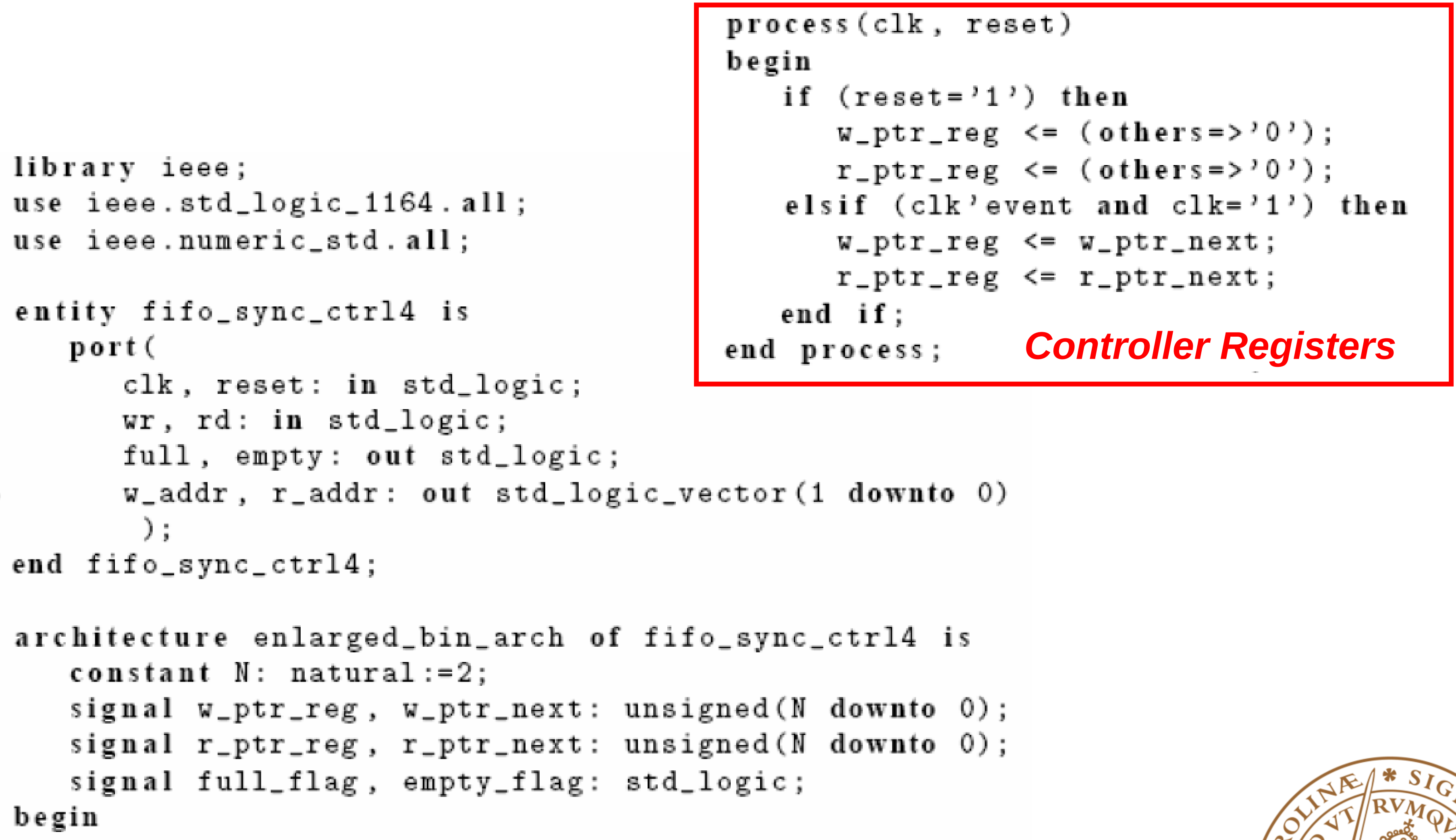
Augmented binary counter:

- Increase the counter by 1 bits
- Use LSBs for as register address
- Use **MSB** to distinguish full or empty



Write pointer	Read pointer	Operation	Status
0 000	0 000	initialization	empty
0 111	0 000	after 7 writes	
1 000	0 000	after 1 write	full
1 000	0 100	after 4 reads	
1 100	0 100	after 4 writes	full
1 100	1 011	after 7 reads	
1 100	1 100	after 1 read	empty
0 011	1 100	after 7 writes	
0 100	1 100	after 1 write	full
0 100	0 100	after 8 reads	empty

FIFO Implementation: VHDL



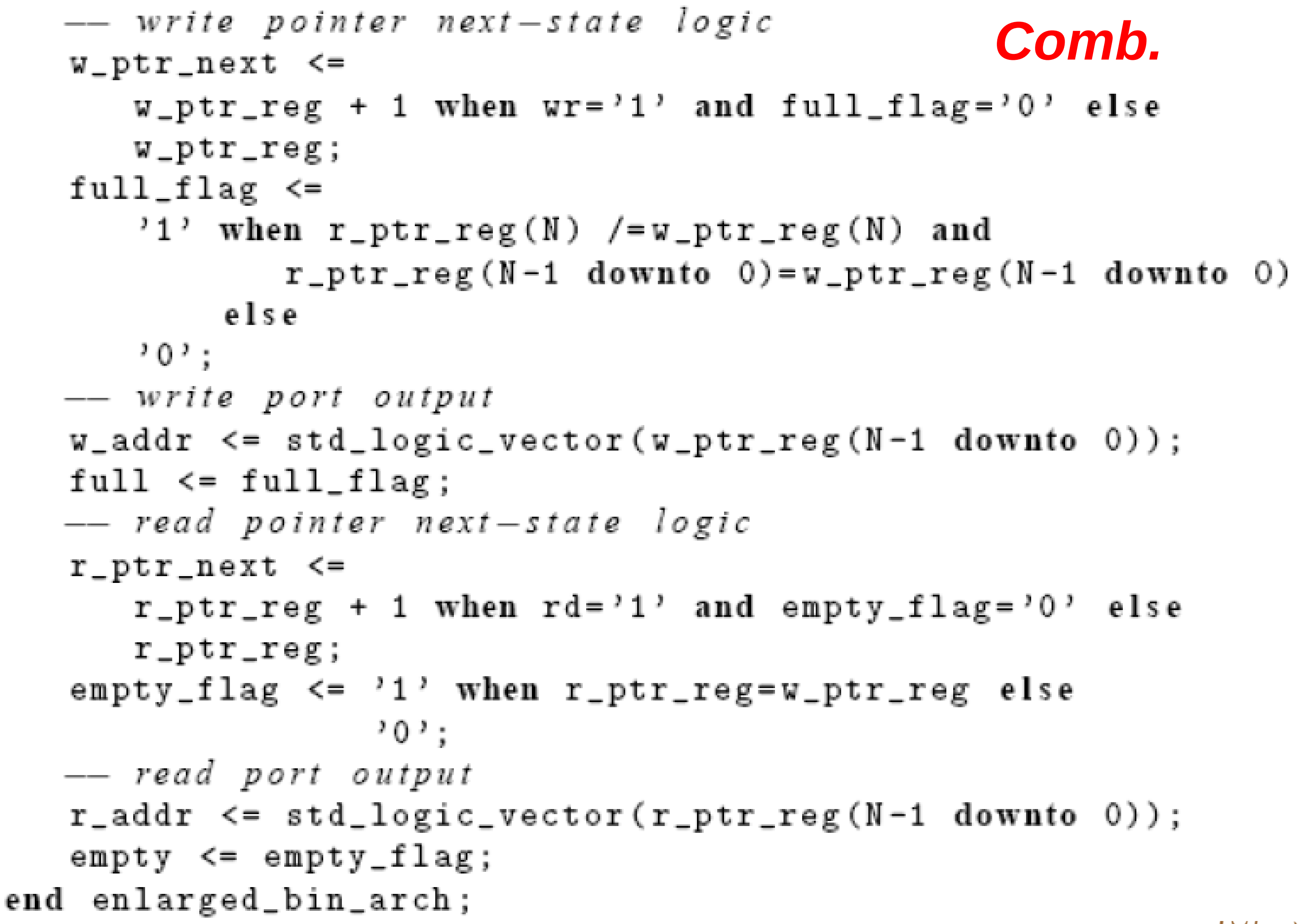
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity fifo_sync_ctrl4 is
   port(
      clk, reset: in std_logic;
      wr, rd: in std_logic;
      full, empty: out std_logic;
      w_addr, r_addr: out std_logic_vector(1 downto 0)
       );
end fifo_sync_ctrl4;

architecture enlarged_bin_arch of fifo_sync_ctrl4 is
   constant N: natural:=2;
   signal w_ptr_reg, w_ptr_next: unsigned(N downto 0);
   signal r_ptr_reg, r_ptr_next: unsigned(N downto 0);
   signal full_flag, empty_flag: std_logic;
begin
```

```
process(clk, reset)
begin
   if (reset='1') then
      w_ptr_reg <= (others=>'0');
      r_ptr_reg <= (others=>'0');
   elsif (clk'event and clk='1') then
      w_ptr_reg <= w_ptr_next;
      r_ptr_reg <= r_ptr_next;
   end if;
end process;
```

Controller Registers

FIFO Implementation: VHDL

Controller Comb.



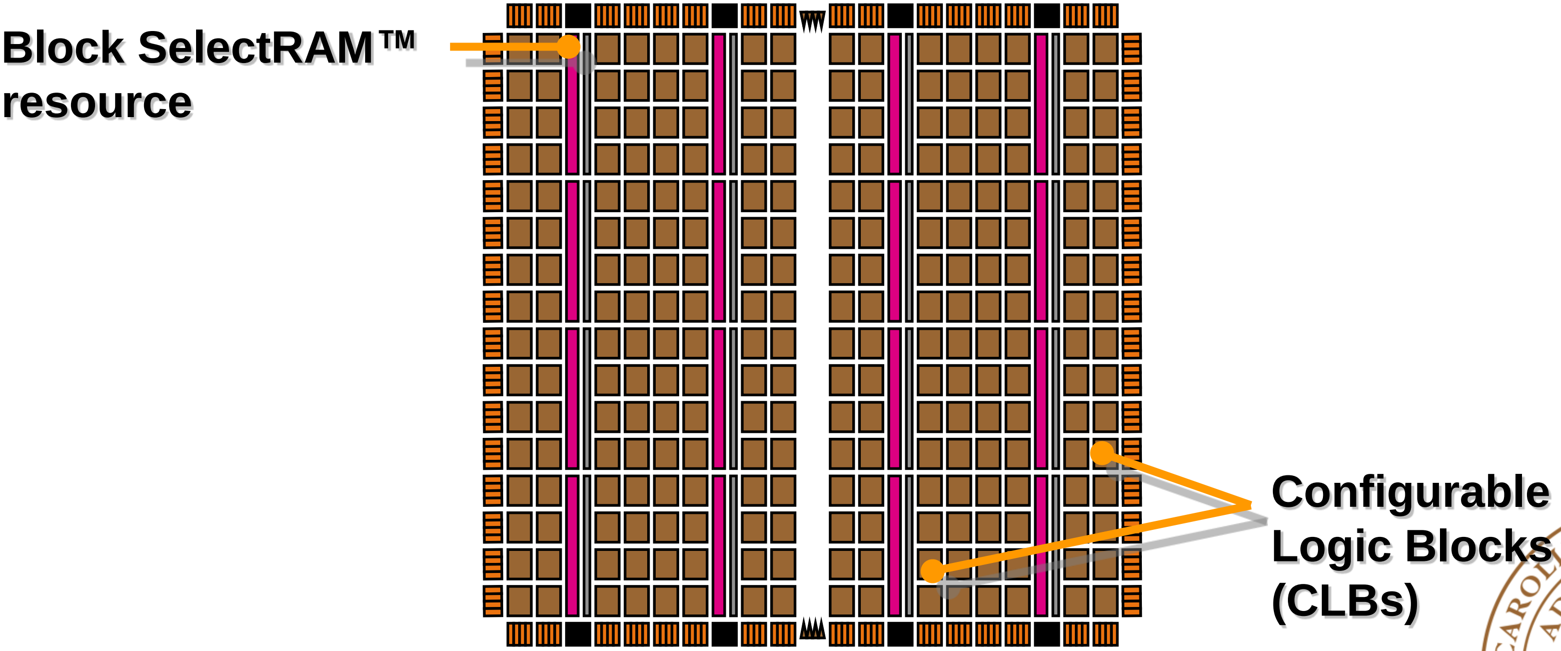
```vhdl
    -- write pointer next-state logic
    w_ptr_next <=
       w_ptr_reg + 1 when wr='1' and full_flag='0' else
       w_ptr_reg;
    full_flag <=
       '1' when r_ptr_reg(N) /=w_ptr_reg(N) and
              r_ptr_reg(N-1 downto 0)=w_ptr_reg(N-1 downto 0)
           else
       '0';
    -- write port output
    w_addr <= std_logic_vector(w_ptr_reg(N-1 downto 0));
    full <= full_flag;
    -- read pointer next-state logic
    r_ptr_next <=
       r_ptr_reg + 1 when rd='1' and empty_flag='0' else
       r_ptr_reg;
    empty_flag <= '1' when r_ptr_reg=w_ptr_reg else
                  '0';
    -- read port output
    r_addr <= std_logic_vector(r_ptr_reg(N-1 downto 0));
    empty <= empty_flag;
 end enlarged_bin_arch;
```

Outline

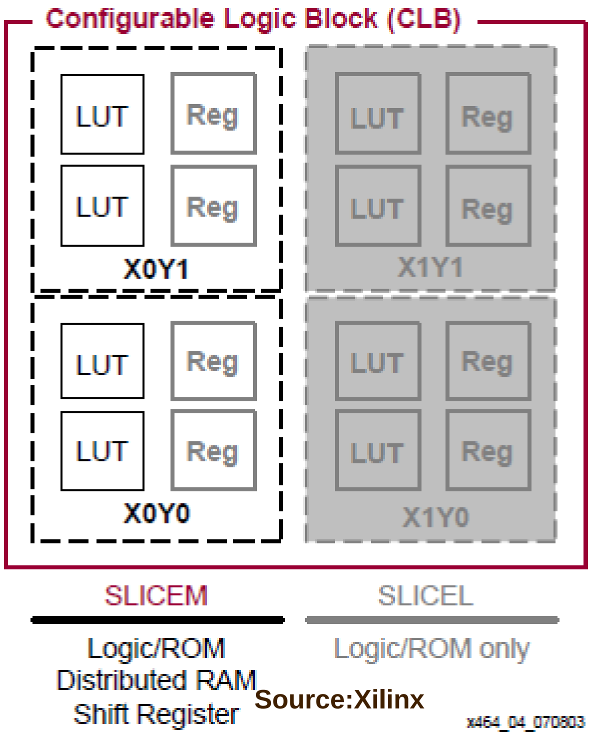
- □ **Overview of Memory**
    - Application, history, trend
    - Different memory type
    - Overall architecture
- □ **Registers as Storage Element**
    - Register File
    - FIFO
- □ **Xilinx Storage Elements**
- □ **Memory Generator**

Storage Components in a Spartan-3 Device

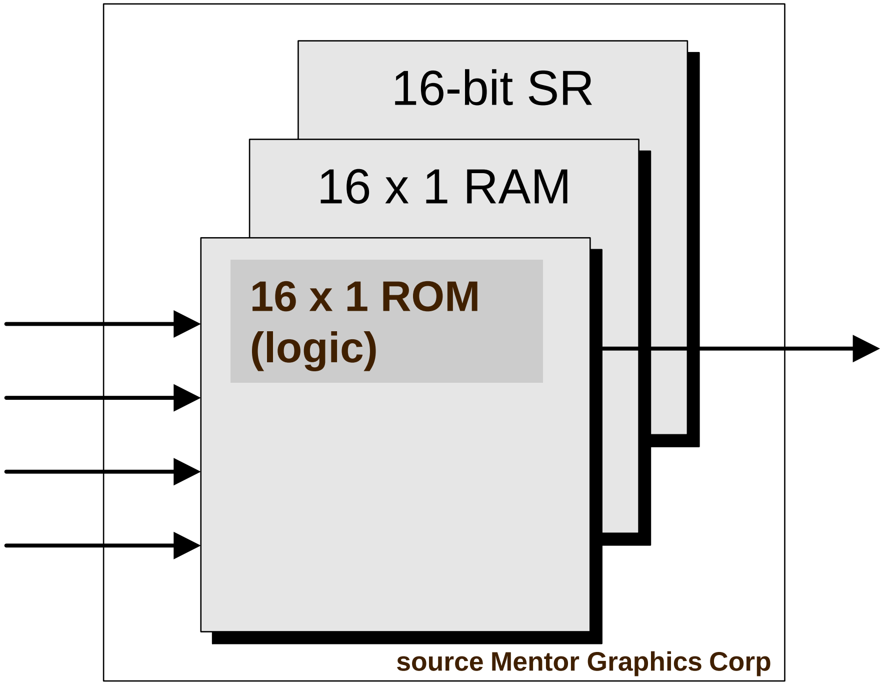
- **Distributed RAM**
  - Fast, localized
  - ideal for small data buffers, FIFOs, or register files
- **Block RAM**
  - For applications requiring large, on-chip memories



Spartan-3 Distributed Memory



Source:Xilinx

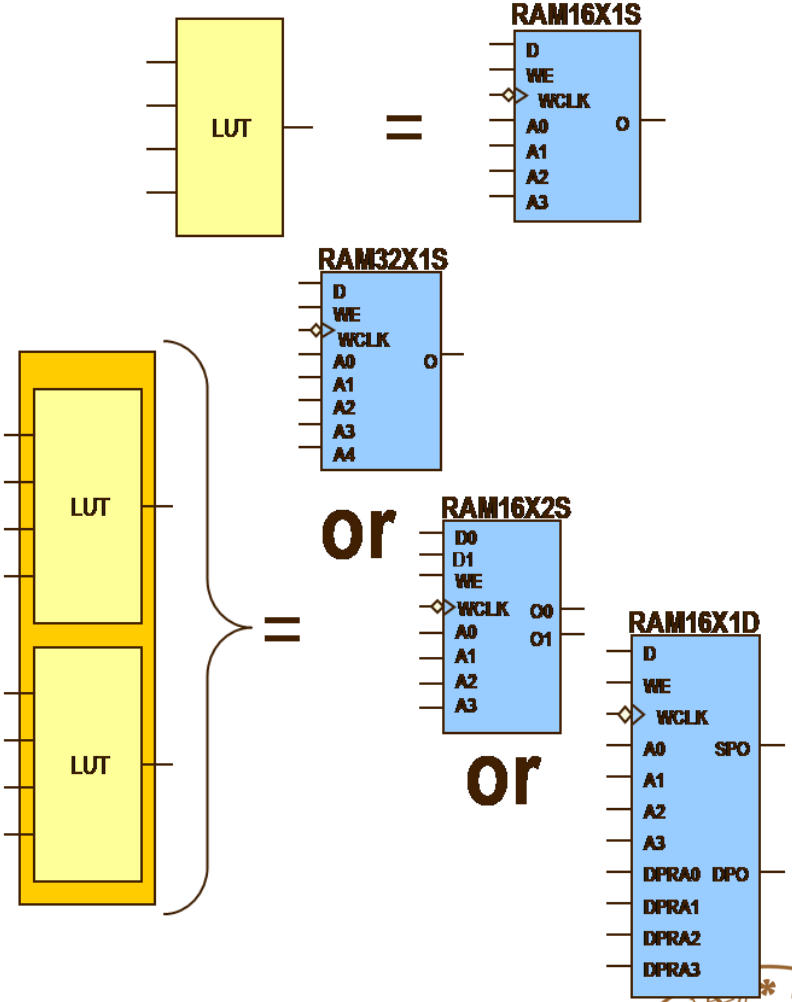


source Mentor Graphics Corp

- One CLB has four slices: SLICEM & SLICEL
- Each LUT in SLICEM has RAM16 × 1S

Spartan-3 Distributed Memory

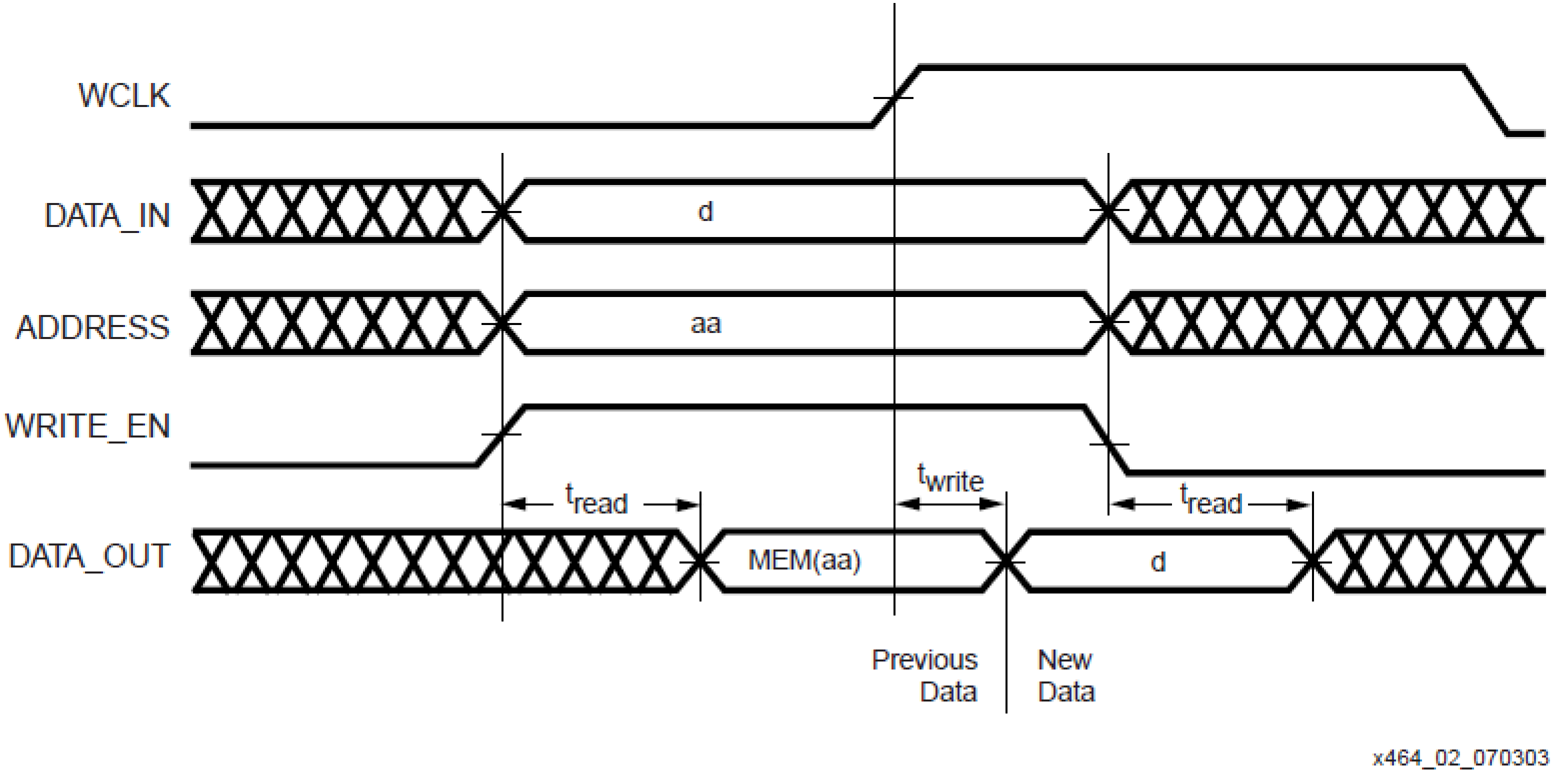
- **Uses a LUT in a slice as memory**
  - An LUT equals 16x1 RAM
  - Cascade LUTs to increase RAM size
- **Two LUTs can make**
  - 32 x 1 single-port RAM
  - 16 x 2 single-port RAM
  - 16 x 1 dual-port RAM
- **Synchronous write**
- **Asynchronous read**
  - Accompanying flip-flops can be used to create synchronous read
- **RAM and ROM are initialized during configuration**
  - Data can be written to RAM after configuration



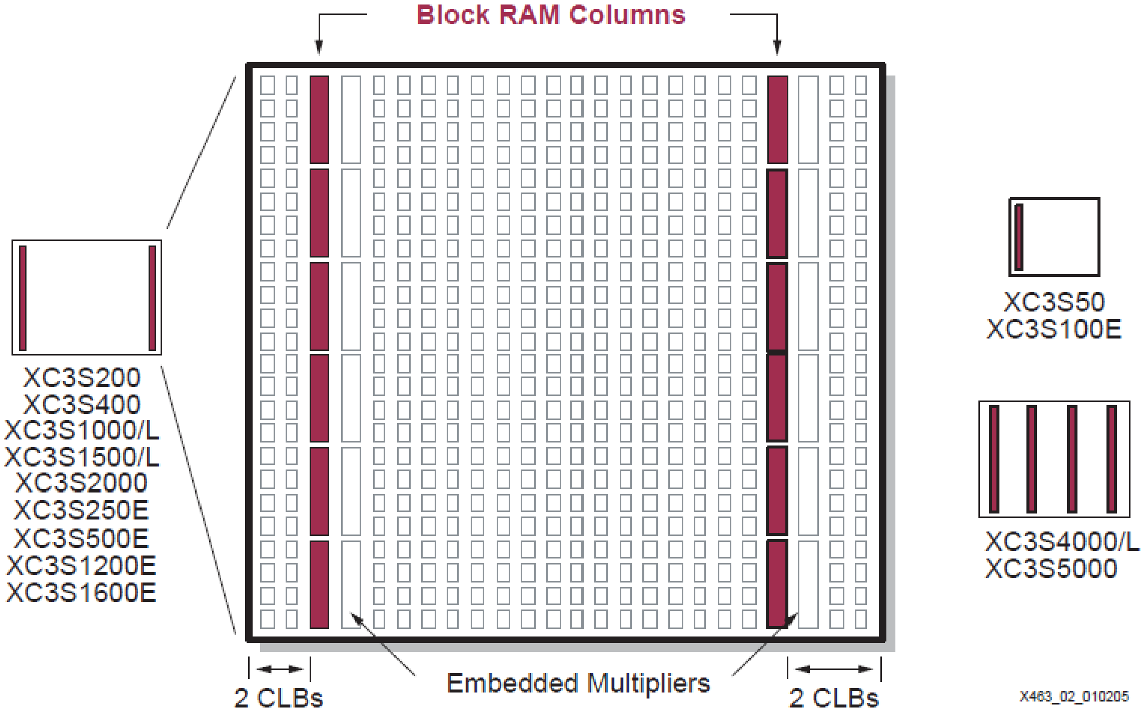
Spartan-3 Distributed Memory

Timing

- Synchronous write
- Asynchronous read

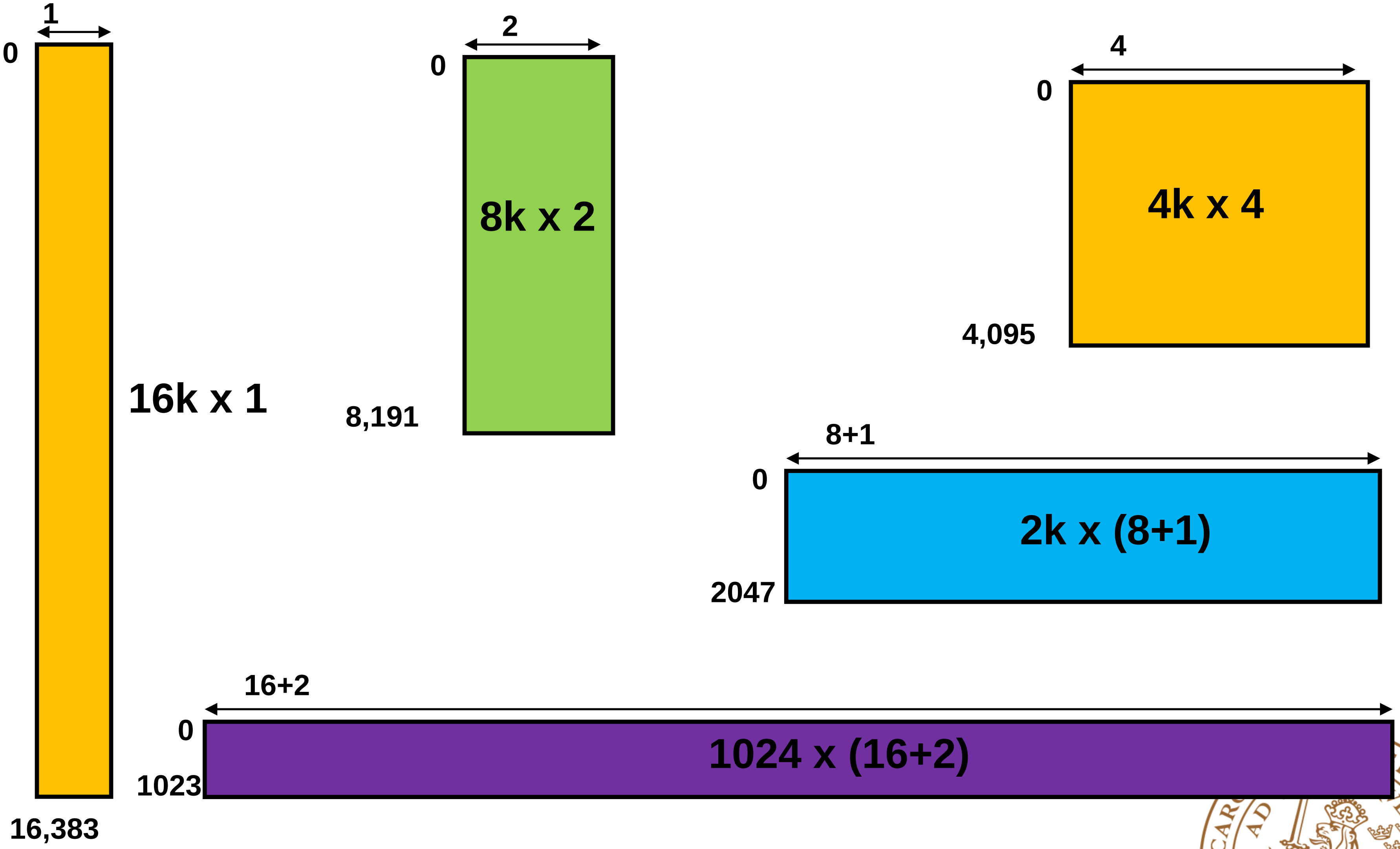


Spartan-3 Block Memory

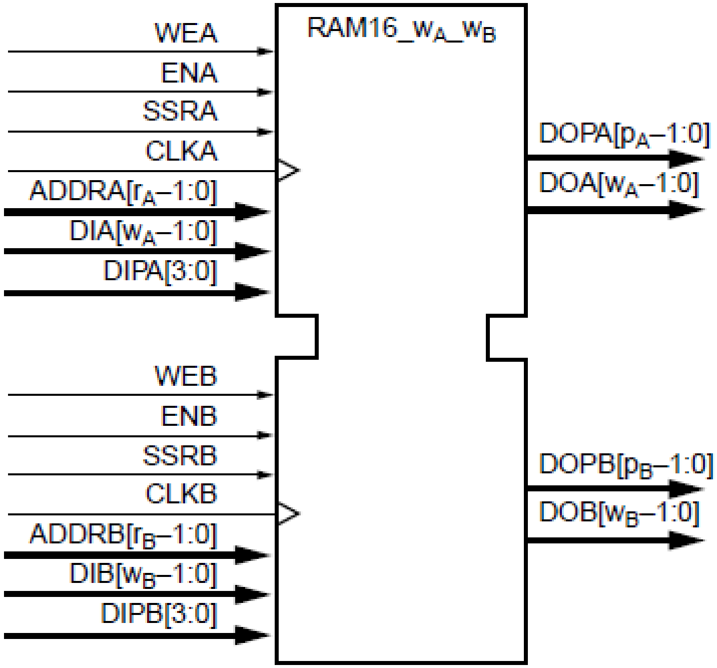


- **Most efficient memory implementation**
  - Dedicated blocks of memory
  - 18 kbits = 18,432 bits per block (16 k without parity bits)
- **Builds both single and true dual-port RAMs**
- **Synchronous write and read (different from distributed RAM)**

Block RAM Configuration (port aspect ratios)

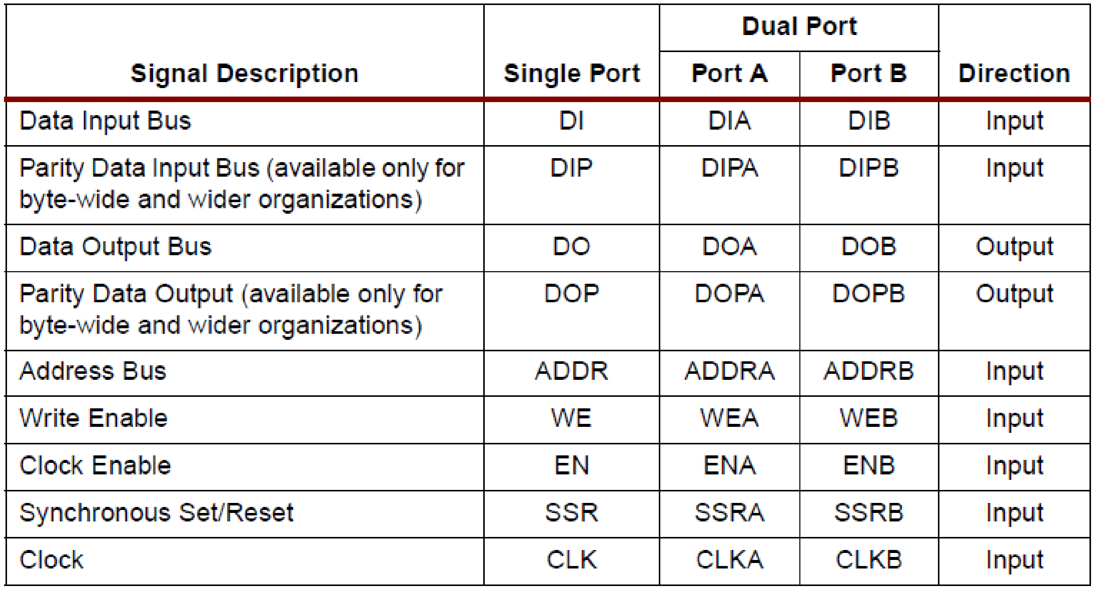


Block RAM Ports



(a) Dual-Port

Table 4: **Block RAM Interface Signals**



Signal Description	Single Port	Dual Port Port A	 Port B	Direction
Data Input Bus	DI	DIA	DIB	Input
Parity Data Input Bus (available only for byte-wide and wider organizations)	DIP	DIPA	DIPB	Input
Data Output Bus	DO	DOA	DOB	Output
Parity Data Output (available only for byte-wide and wider organizations)	DOP	DOPA	DOPB	Output
Address Bus	ADDR	ADDRA	ADDRB	Input
Write Enable	WE	WEA	WEB	Input
Clock Enable	EN	ENA	ENB	Input
Synchronous Set/Reset	SSR	SSRA	SSRB	Input
Clock	CLK	CLKA	CLKB	Input

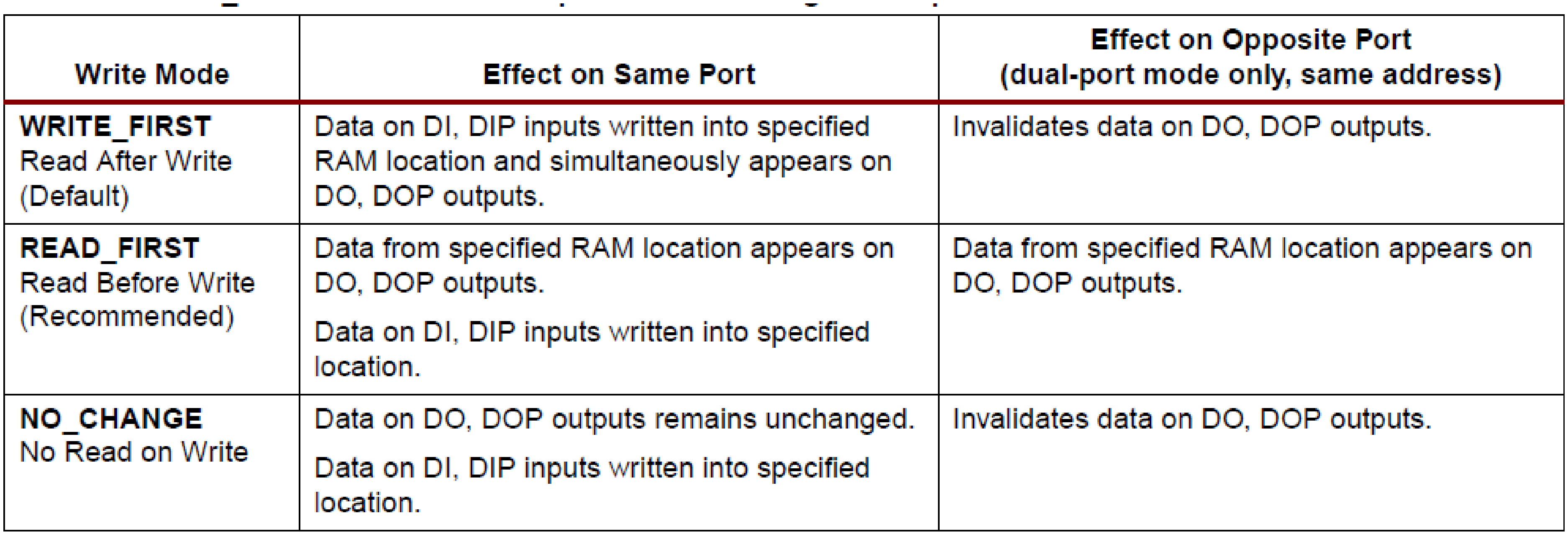
□$w_{A,B}$: the data path width at ports A,B.

□$p_{A,B}$: the number of data path lines serving as parity bits.

□$r_{A,B}$: the address bus width at ports A, B

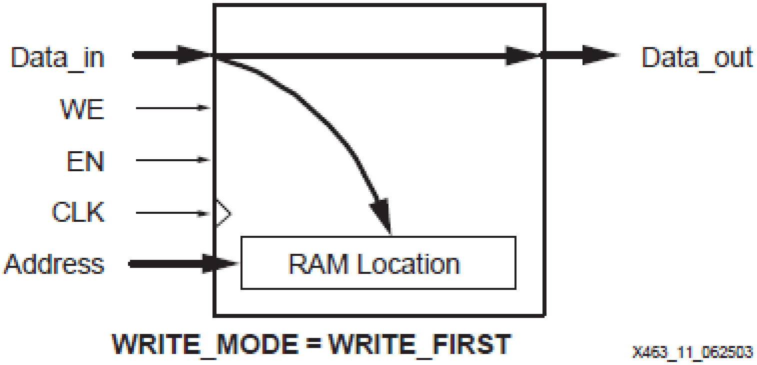
□The control signals CLK, WE, EN, and SSR on both ports have the option of inverted polarity.

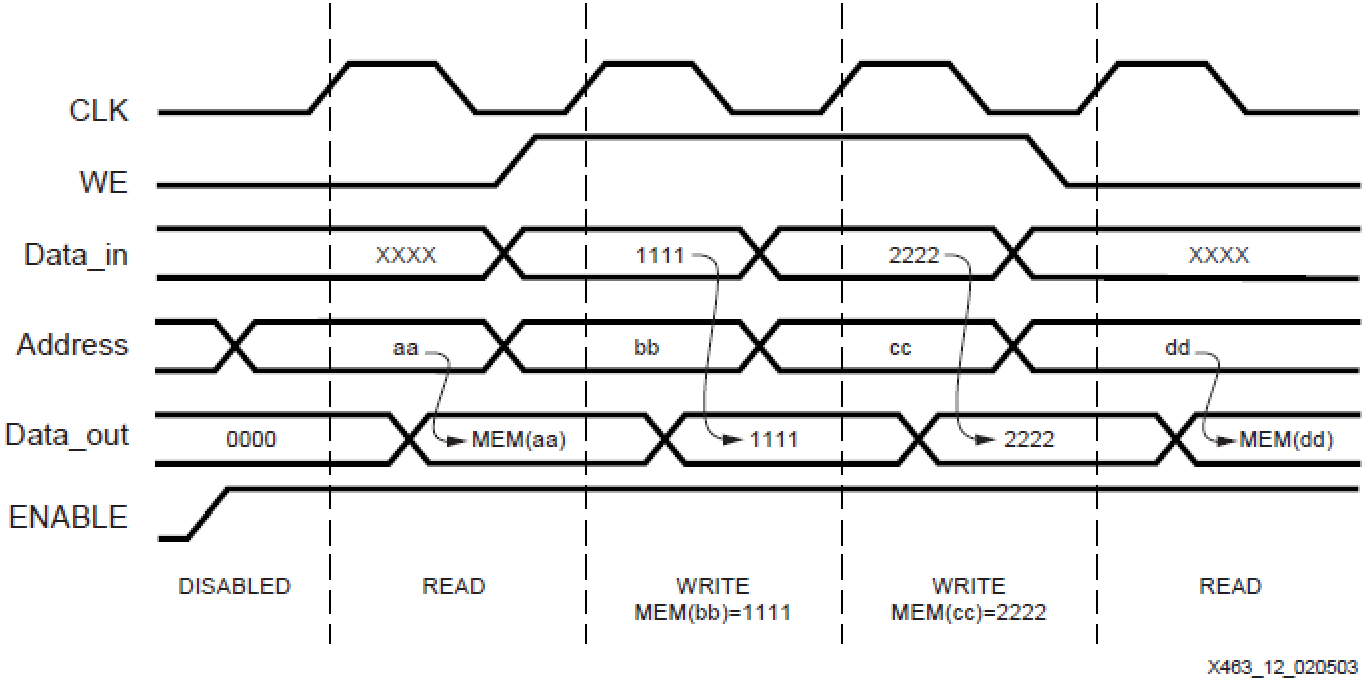
Block RAM: Operation Modes



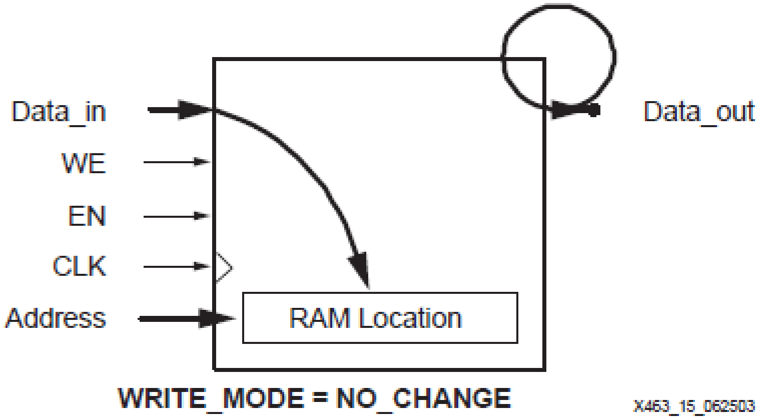
Write Mode	Effect on Same Port	Effect on Opposite Port (dual-port mode only, same address)
WRITE_FIRST Read After Write (Default)	Data on DI, DIP inputs written into specified RAM location and simultaneously appears on DO, DOP outputs.	Invalidates data on DO, DOP outputs.
READ_FIRST Read Before Write (Recommended)	Data from specified RAM location appears on DO, DOP outputs. Data on DI, DIP inputs written into specified location.	Data from specified RAM location appears on DO, DOP outputs.
NO_CHANGE No Read on Write	Data on DO, DOP outputs remains unchanged. Data on DI, DIP inputs written into specified location.	Invalidates data on DO, DOP outputs.

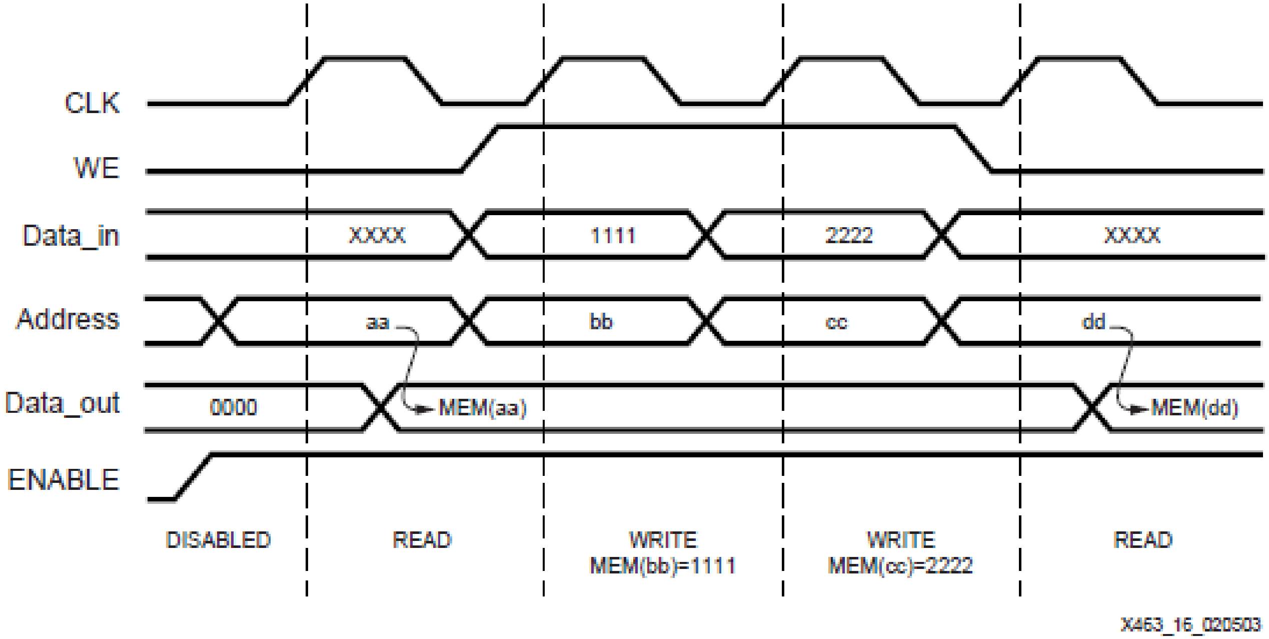
Block RAM: WRITE_FIRST



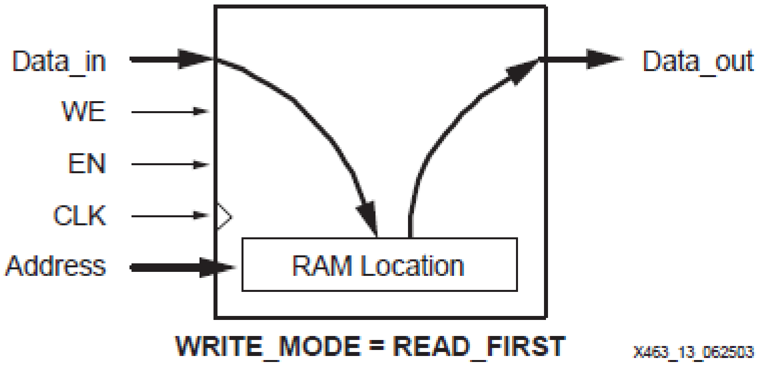


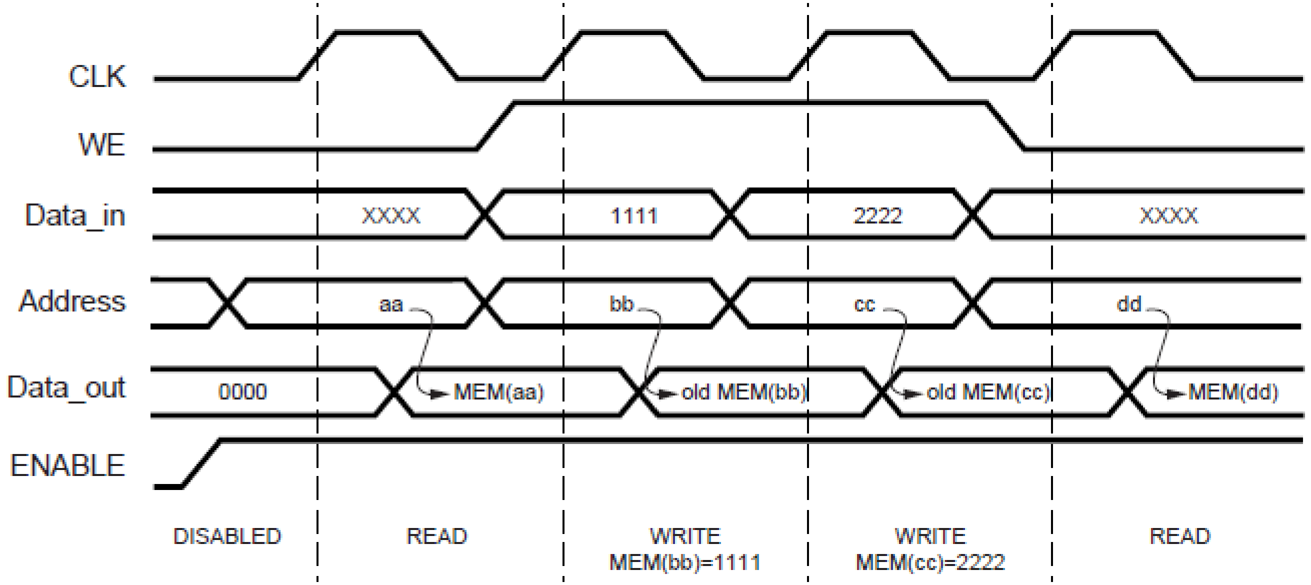
Block RAM: NO_CHANGE





Block RAM: READ_FIRST (Recomm.)





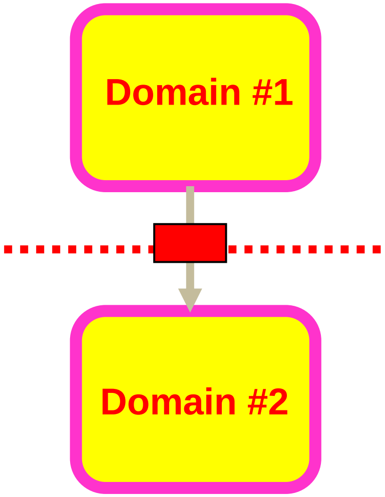
Reading Advice

- □RTL Hardware Design Using VHDL: P276-P292
- □XAPP463 Using Block RAM in Spartan-3 Generation FPGAs *(Google search: XAPP463)*
- □ XAPP464 Using Look-Up Tables as Distributed RAM in Spartan-3 Generation FPGAs *(Google search: XAPP464)*
- □ XST User Guide, Section: RAMs and ROMs HDL Coding Techniques *(Google search: XST User Guide (PDF))*
- □ ISE In-Depth Tutorial, Section: Creating a CORE Generator Software Module *(Google search: ISE In-Depth Tutorial)*

□Why two DFFs?

Crossing clock domain

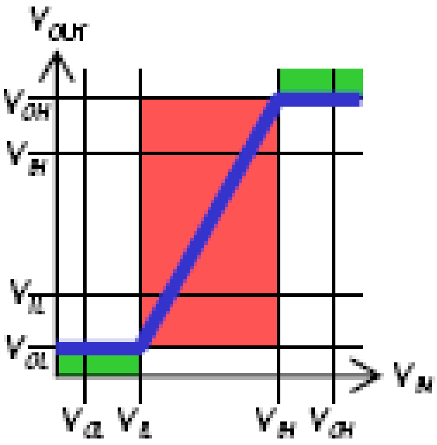
- **Multiple clock is needed in case:**
  - Inherent system requirement
    - ***Different clocks for sampling and processing***
  - Chip size limitation
    - ***Clock skew increases with the # FFs in a system***
    - ***Current technology can support up to 10^4 FFs***
  - Low power design
    - ***Clock gating***



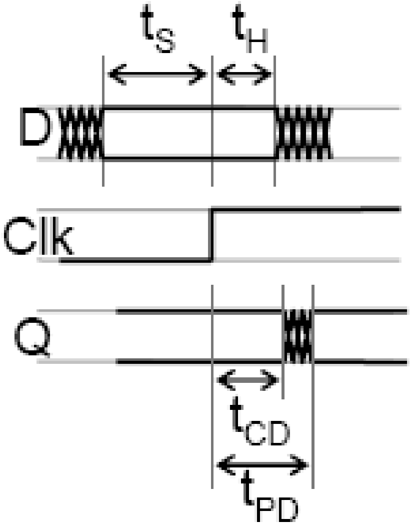
Multiple Clocks: Problems

- We have been setting very strict rules to make our digital circuits safe: using a forbidden zone in both voltage and time dimensions

Digital Values: distinguishing voltages representing “1” from “0”

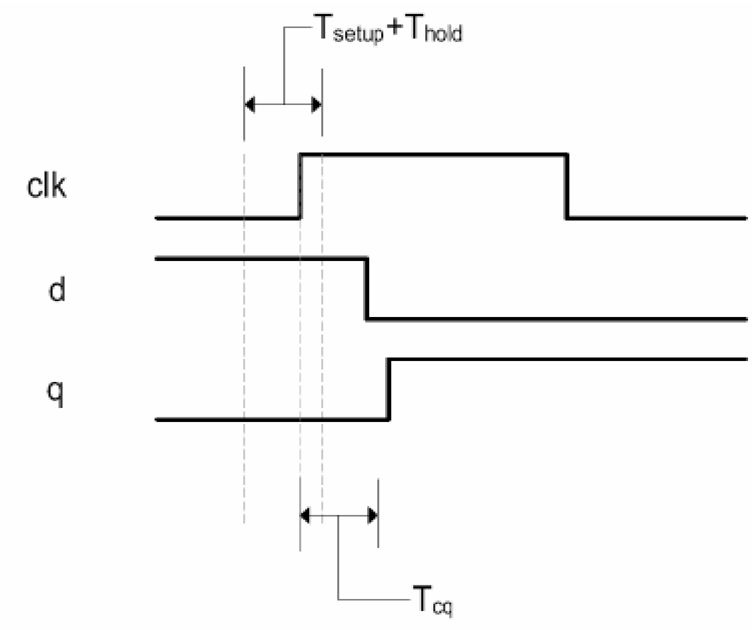


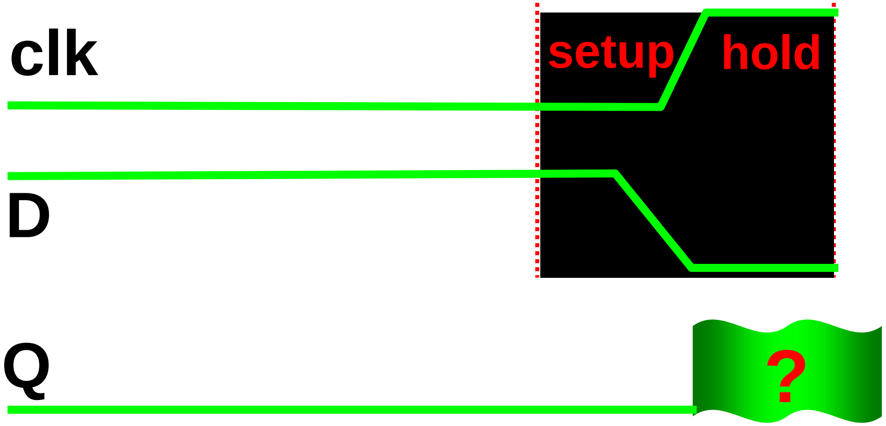
Digital Time: setup and hold time rules



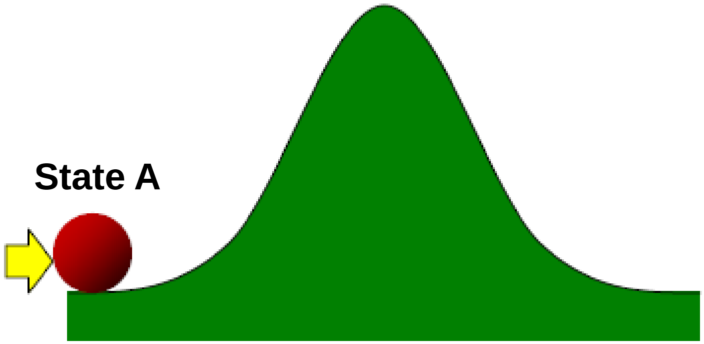
Metastability

- With asynchronous inputs, we have to break the rules: ***we cannot guarantee that setup and hold time requirements are met at the inputs!***
- What happens after timing violation?



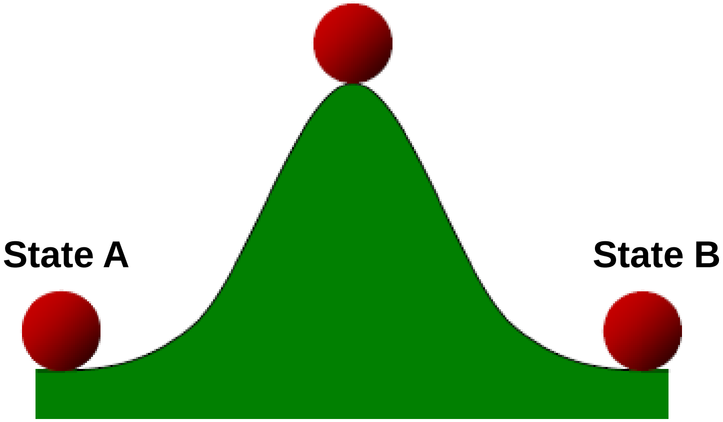


Mechanical Metastability



□ **Launch a golf up a hill, 3 possible outcomes:**

- Hit lightly: Rolls back
- Hit hard: Goes over
- Or: Stalls at the apex



□ **That last outcome is not stable:**

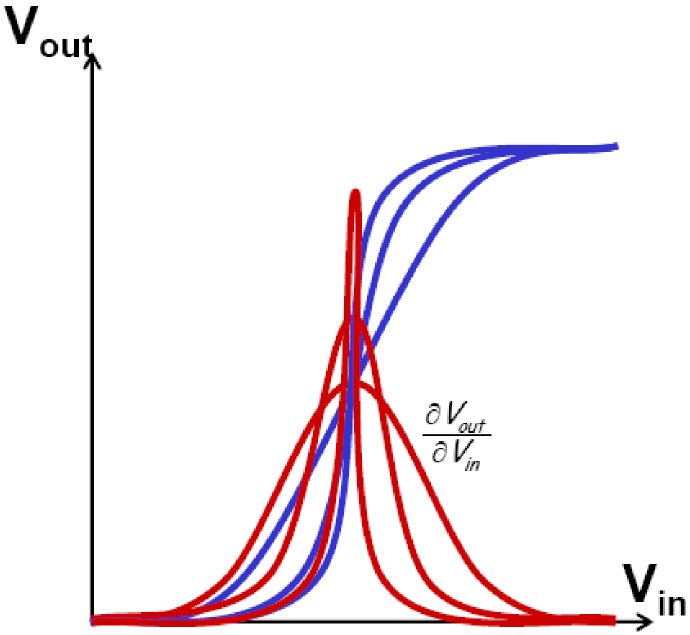
- A gust of wind
- Brownian motion
- ***Can you tell the eventual state?***

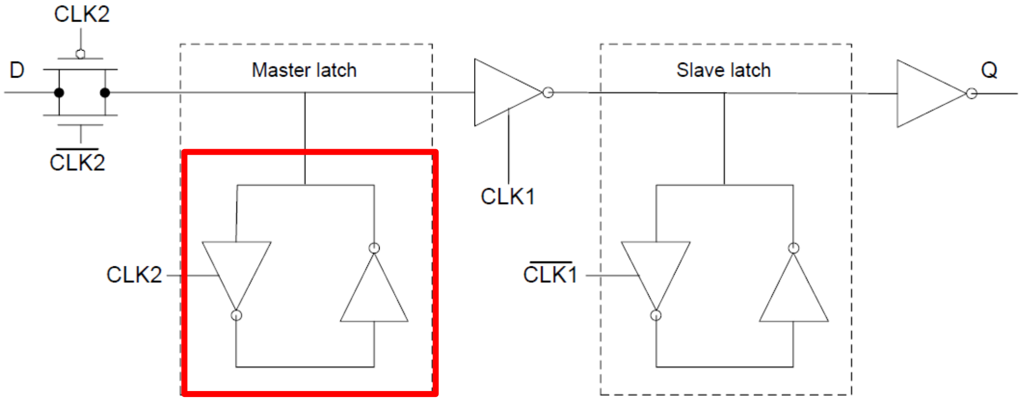
Metastability in Digital Logic

□Our hill is related to the VTC (Voltage Transfer Curve).

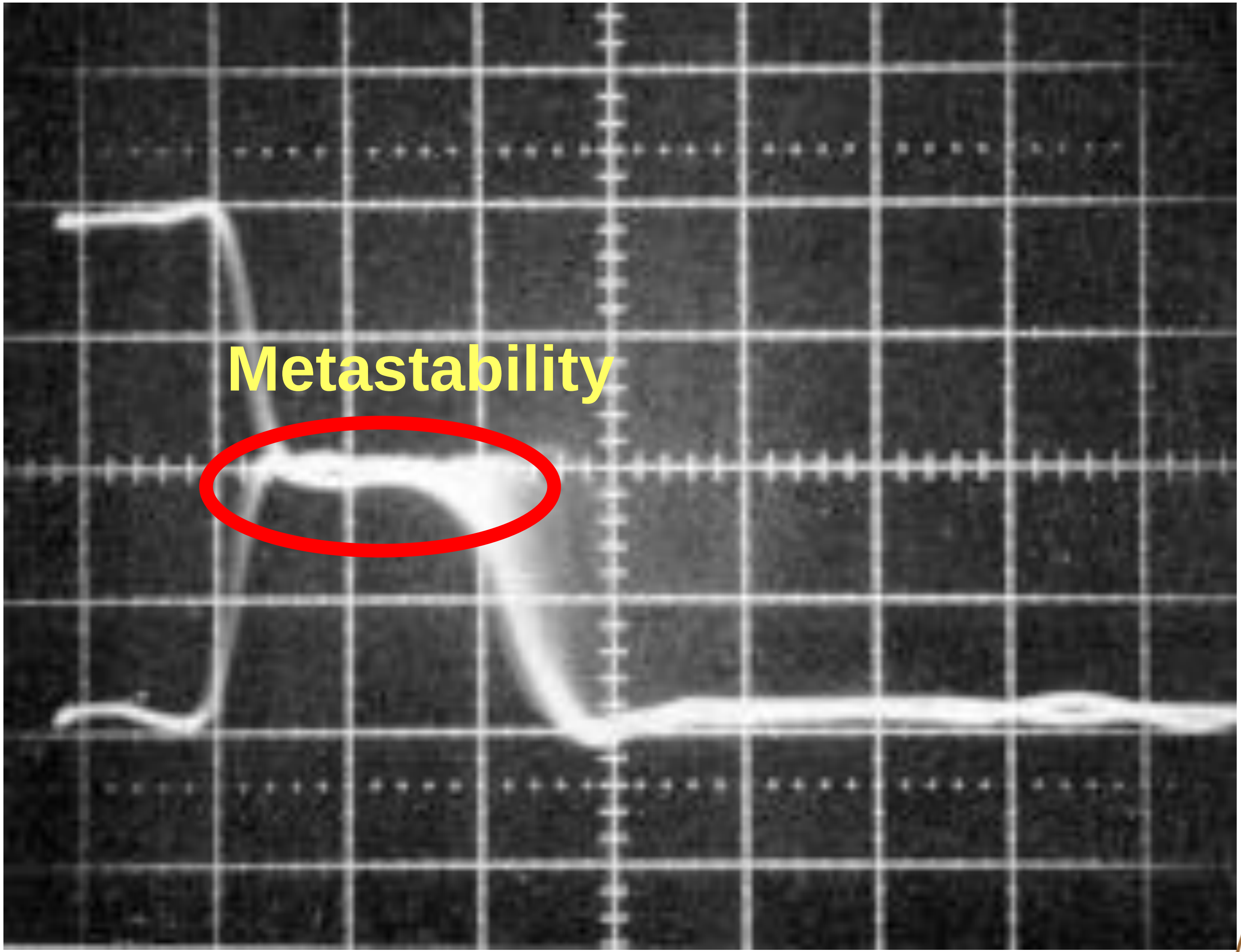
□The higher the gain thru the transition region, the steeper the peak of the hill, the harder to get into a metastable state.

□We can decrease the probability of getting into the metastable state, but we can’t eliminate it…

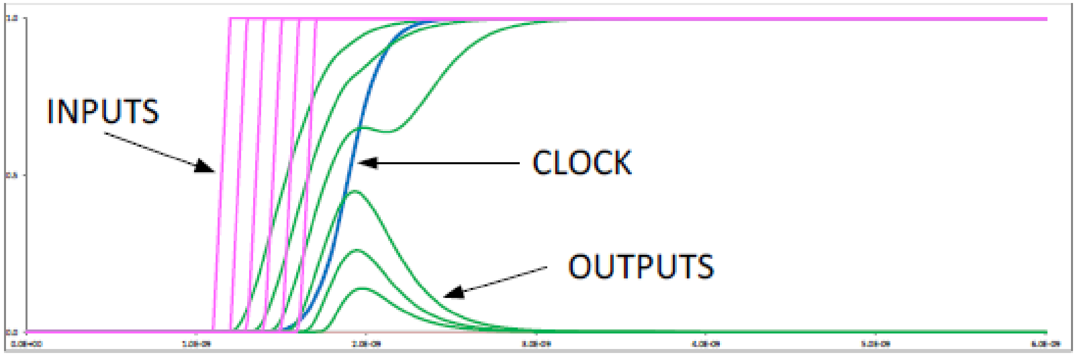


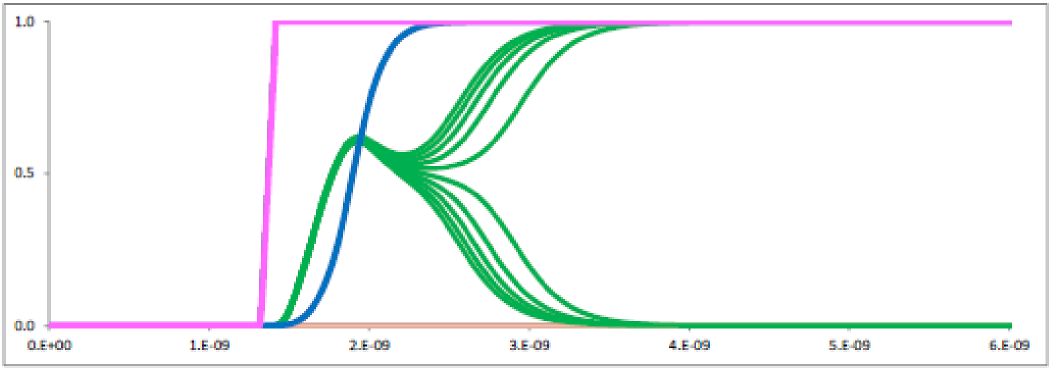


Metastability in Digital Logic



Metastability in Digital Logic



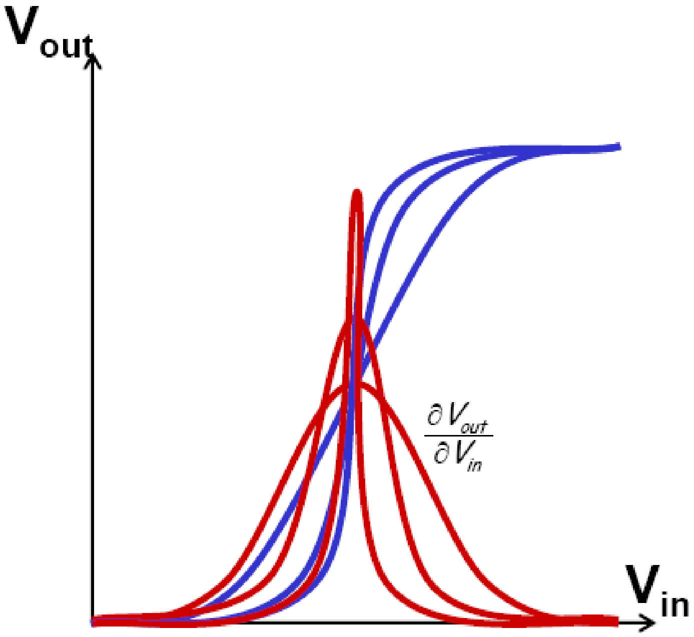


- **Fixed clock edge**
- **Change the edge of inputs**
- **The input edge is moved in steps of 100ps and 1ps**
- **The behavior of outputs**
  - ‘Three’ possible states
  - Will exit metastability

How long it takes to exit Metastability?

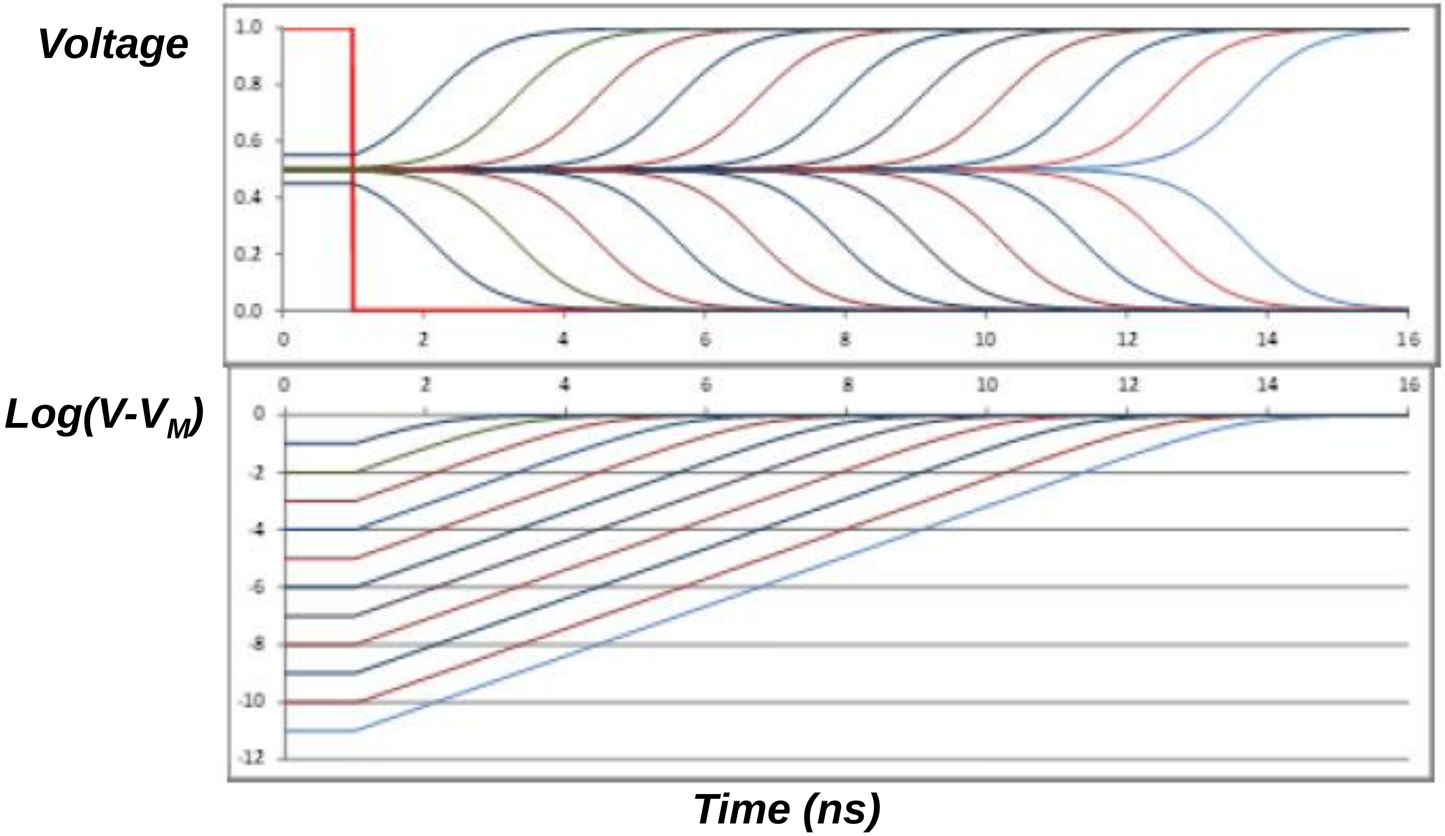
Exit Metastability

- □Define a fixed-point voltage, ***V_M***, (always have) such that $V_{IN} = V_M$ implies $V_{OUT} = V_M$
- □Assume the device is sampling at some voltage V_0 near V_M
- □The time to settle to a stable value depends on ***$(V_0 - V_M)$***; its theoretically infinite for $V_0 = V_M$



Exit Metastability

- □The time to exit metastability depends ***logarithmically on*** $(V_0 - V_M)$
- □The ***probability*** of remaining metastable at time T is $e^{-T/\tau}$



MTBF: The probability of being metastable at time S?

- Two conditions have to be met concurrently
  - An FF enters the metastable state
  - An FF cannot resolve the metastable condition within S
- The rate of failure $p(failure) = p(enter\ MS) \times p(time\ to\ exit > S)$

$$\boxed{Rate(failures) = T_W F_C F_D \times e^{-S/\tau}}$$

  - T_W: time window around sampling edge incurring metastability
  - F_C: clock rate (assuming data change is uniformly distributed)
  - F_D: input change rate (input may not change every cycle)
- Mean time between failures (MTBF)

$$\boxed{MTBF = \frac{e^{S/\tau}}{T_W F_C F_D}}$$

MTBF (Mean Time Between Failure)

□ **Let’s calculate an ASIC for 28nm CMOS process**

- τ: 10ps (different FFs have different τ)
- T_W=20ps, F_C=1GHz
- Data changes every ten clock cycles
- Allow 1 clock cycle to resolve metastability, **$S=T_C$**

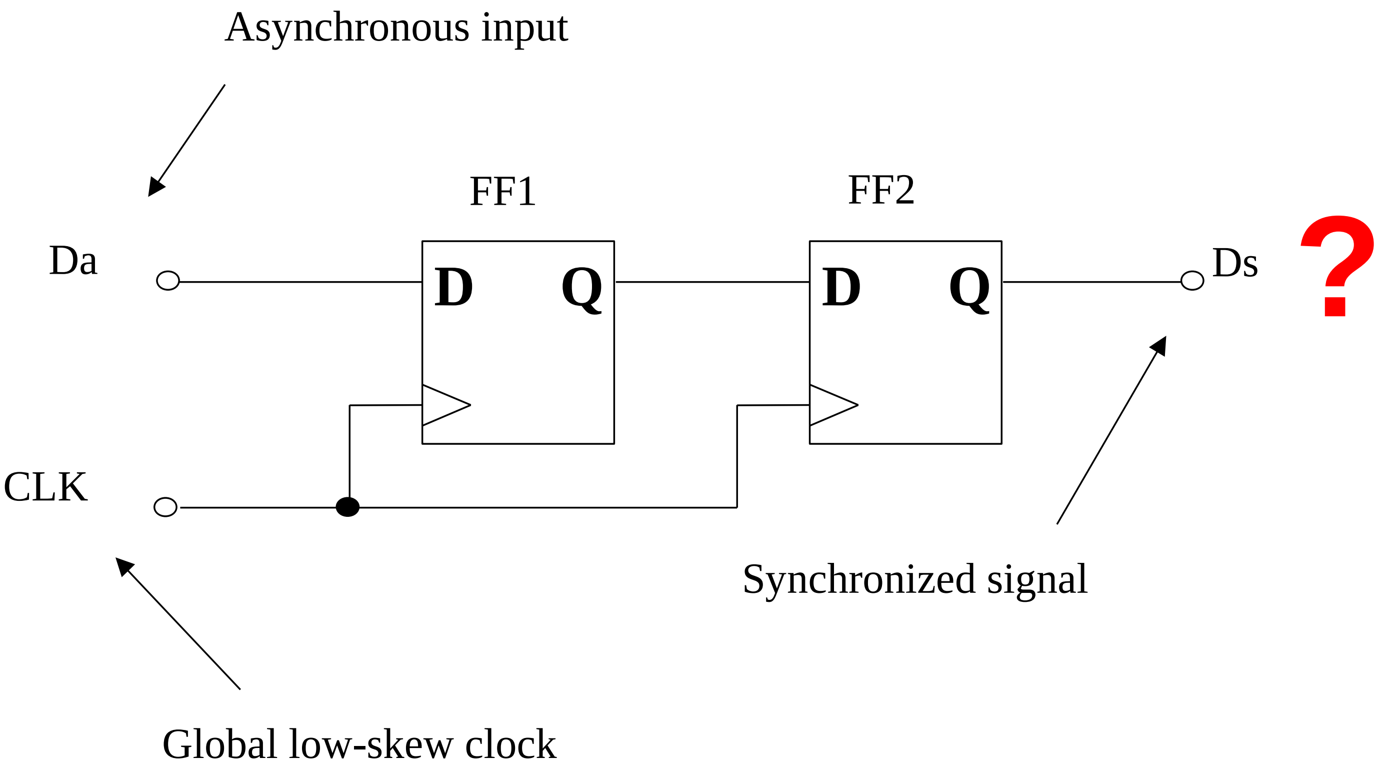
MTBF=4×10^{29} year !

[For comparison:

Age of oldest hominid fossil: 5x10^6 years

Age of earth: 5x10^9 years]

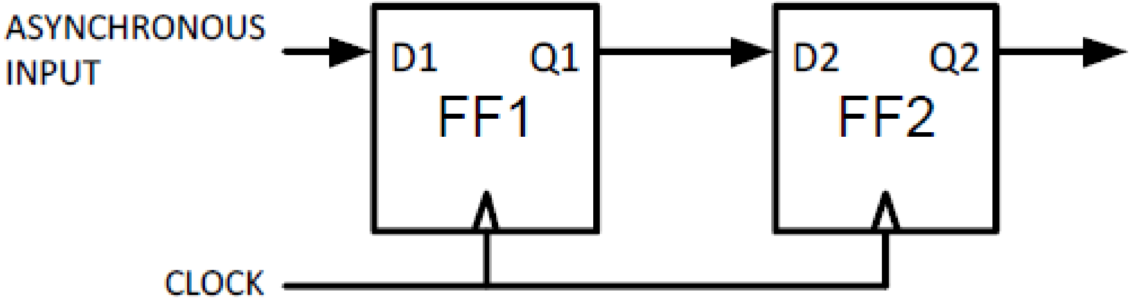
The Two-Flip-Flop Synchronizer

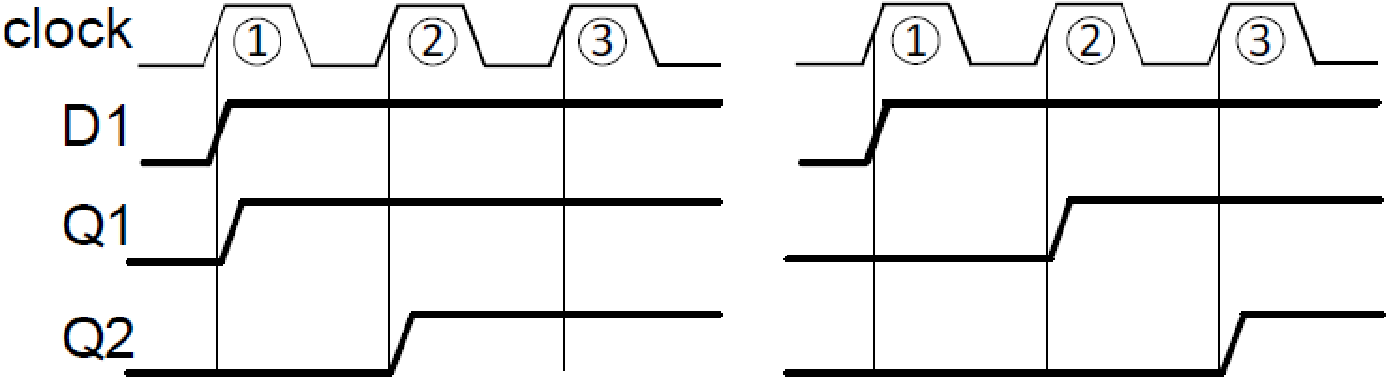


$S = T_C$

The Two-Flip-Flop Synchronizer

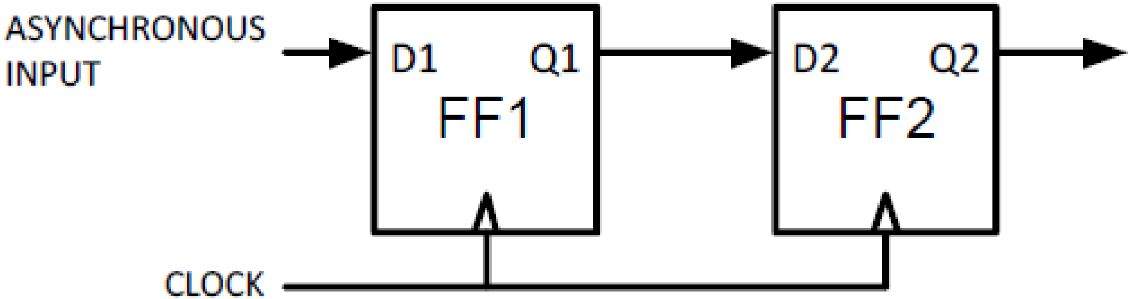
□Possible Outcomes

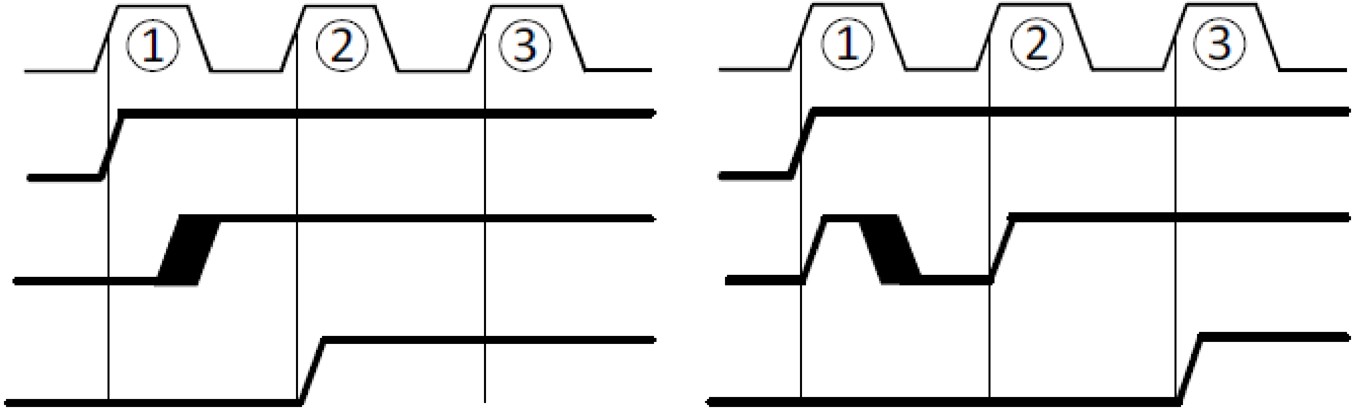




The Two-Flip-Flop Synchronizer

- Possible Outcomes

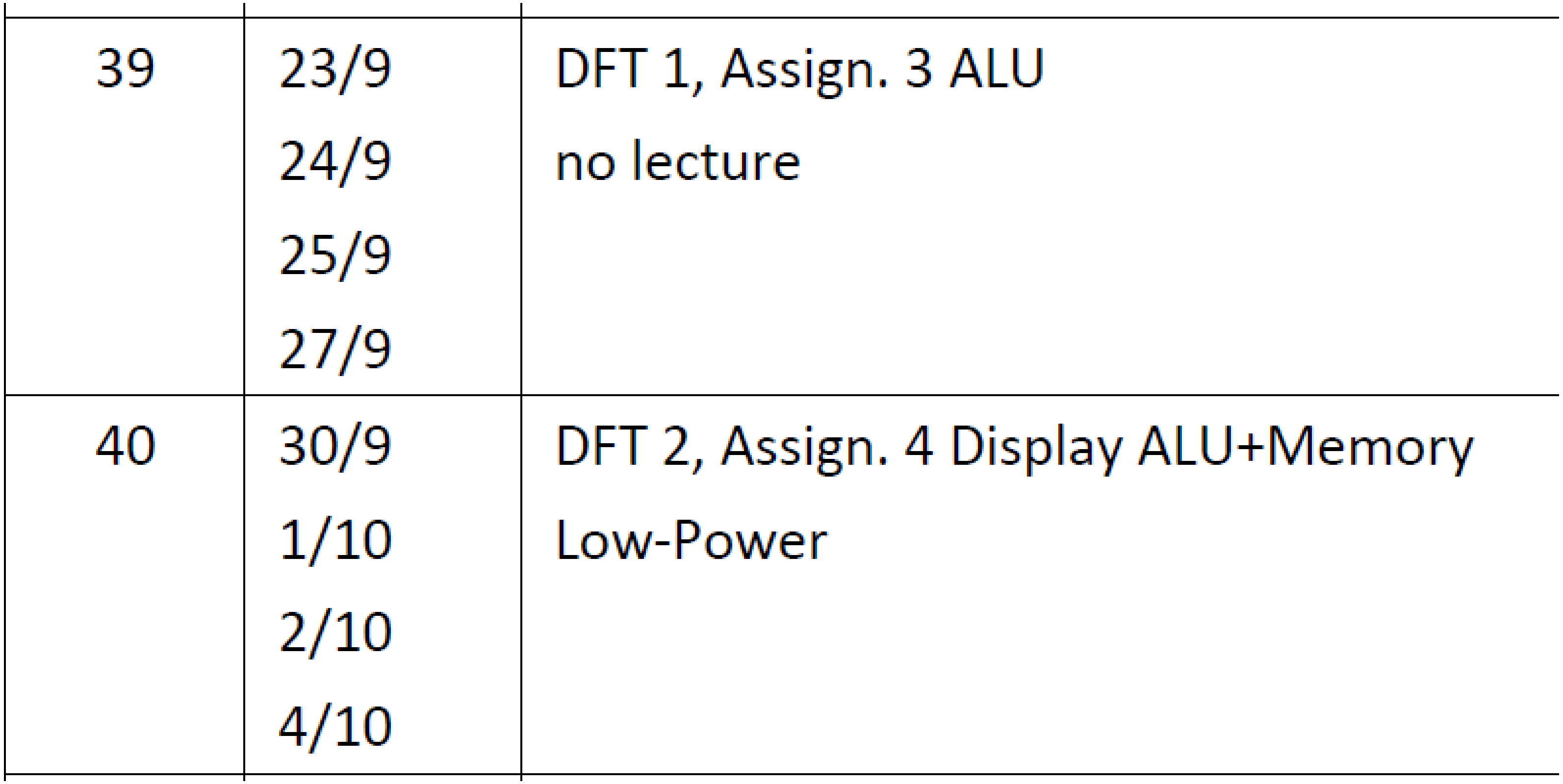




Open Question: What is the limitation?

Reading Advice

“Metastability and Synchronizers: A Tutorial”, Ran Ginosar, VLSI Systems Research Center, Israel Institute of Technology



39	23/9	DFT 1, Assign. 3 ALU
	24/9	no lecture
	25/9	
	27/9	
40	30/9	DFT 2, Assign. 4 Display ALU+Memory
	1/10	Low-Power
	2/10	
	4/10	