

EECS151 : Introduction to Digital Design and ICs

Lecture 20 – Memory and Clock

Bora Nikolić and Sophia Shao



This Maglev Heart Could Keep Cardiac Patients Alive

Aug. 22, 2019. Inside Bivacor's artificial heart, a levitating disk spins 2000 times per minute to keep blood flowing

IEEE Spectrum: <https://spectrum.ieee.org/biomedical/devices/this-maglev-heart-could-keep-cardiac-patients-alive>



Review

- SRAM and regfile cells can have multiple R/W ports
- Memory decoding is done hierarchically
 - Wire-limited in large arrays
- Multiple cache levels make memory appear both fast and big
- Direct mapped and set-associative cache



ASIC Memories

ASIC Memory Compilers

Artisan[®]
ARM[®] Physical IP

sram_sp_hdc_svt_rvt_hvt Compiler
TSMC cln65gplus 65nm Process, 0.525um² Bitcell

GENERIC PARAMETERS

Instance Name: sram_sp_hdc_svt_rvt_hvt

Number of Words: 2048

Number of Bits: 16

Frequency <MHz>: 1

Multiplexer Width: ☐ 4 ☐ 8 ☒ 16 ☐ 32

Pipeline: ☐ on ☒ off

Word-Write Mask: ☐ on ☒ off

Write-thru: ☒ on ☐ off

Top Metal Layer: ☒ m5-m9

Power Type: ☒ ArtiGrid

Redundancy: ☐ on ☒ off

BIST MUXes: ☐ on ☒ off

Soft Error Repair (SER): ☒ none ☐ 1bd1bc ☐ 2bd1bc

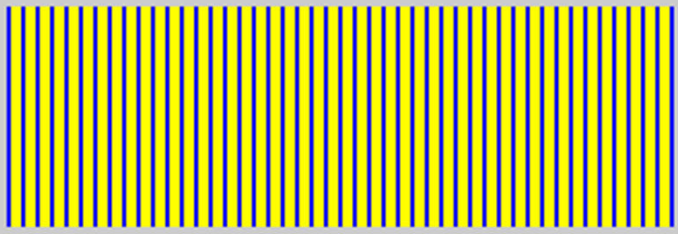
Back Biasing: ☐ on ☒ off

Power Gating: ☐ on ☒ off

Retention: ☒ on

Default Update

RELATIVE FOOTPRINT



ASCII DATATABLE

name	tt_1p00v...	ss_0p90v...	ss_0p90v...	ff_...
geomx	317.790	317.790	317.790	317.790
geomy	111.350	111.350	111.350	111.350
volt	1.000	0.900	0.900	1.000
temp	25.000	-40.000	125.000	125.000
tcyc0	0.984	1.658	1.581	0.984
tcyc1	1.042	1.761	1.684	1.042
tcyc2	1.124	1.941	1.826	1.124
tcyc3	1.150	1.990	1.874	1.150
tcyc4	999.000	999.000	999.000	999.000
tcyc5	999.000	999.000	999.000	999.000
tcyc6	999.000	999.000	999.000	999.000
tcyc7	999.000	999.000	999.000	999.000
ta0	0.736	1.213	1.142	0.736
ta1	0.794	1.316	1.244	0.794
ta2	0.876	1.496	1.387	0.876
ta3	0.901	1.545	1.435	0.901
ta4	999.000	999.000	999.000	999.000
ta5	999.000	999.000	999.000	999.000
ta6	999.000	999.000	999.000	999.000
ta7	999.000	999.000	999.000	999.000
tas	0.226	0.389	0.356	0.226
tah	0.133	0.206	0.213	0.133
tcs	0.152	0.221	0.230	0.152
tch	0.121	0.209	0.194	0.121
tws	0.121	0.235	0.224	0.121
twb	0.160	0.262	0.264	0.160

VIEWS

Verilog Model

PostScript Datasheet

ASCII Datable

Verilog Model

Synopsys Model

VCLEF Footprint

GDSII Layout

LVS Netlist

TetraMax Model

- Memory compiler produces front-end views (similar to standard cells, but really large ones)



FPGA Memories

Verilog RAM Specification

```
//  
// Single-Port RAM with Asynchronous Read  
//  
module ramBlock (clk, we, a, di, do);  
    input  clk;  
    input  we;           // write enable  
    input  [19:0] a;      // address  
    input  [7:0] di;      // data in  
    output [7:0] do;      // data out  
    reg    [7:0] ram [1048575:0]; // 8x1Meg  
    always @(posedge clk) begin // Synch write  
        if (we)  
            ram[a] <= di;  
    assign do = ram[a];           // Asynch read  
endmodule
```

What do the synthesis tools do with this?

Verilog Synthesis Notes (FPGAs)

- Block RAMS and LUT RAMS all exist as primitive library elements. However, it is much more convenient to **use inference**.
- Depending on how you write your Verilog, you will get either a collection of block RAMs, a collection of LUT RAMs, or a collection of flip-flops.
- The synthesizer uses size, and read style (sync versus async) to determine the best primitive type to use.
- It is possible to force mapping to a particular primitive by using synthesis directives.
Ex: `(* ram_style = "distributed" *) reg myReg;`
- The synthesizer has limited capabilities (eg., it can combine primitives for more depth and width, but is limited on porting options). Be careful, as you might not get what you want.
- See **User Guide** for examples.
- CORE generator memory block has an extensive set of parameters for explicitly instantiated RAM blocks.

Inferring RAMs in Verilog (FPGA)

// 64X1 RAM implementation using distributed RAM

```
module ram64X1 (clk, we, d, addr, q);  
input clk, we, d;  
input [5:0] addr;  
output q;
```

```
    reg [63:0] temp;  
    always @ (posedge clk)  
        if (we)  
            temp[addr] <= d;  
    assign q = temp[addr];
```

```
endmodule
```

Verilog reg array used with
“always @ (posedge ... infers
memory array.

Asynchronous read infers LUT
RAM

Dual-read-port LUT RAM (FPGA)

```
//  
// Multiple-Port RAM Descriptions  
//  
module v_rams_17 (clk, we, wa, ra1, ra2, di, do1, do2);  
    input  clk;  
    input  we;  
    input  [5:0] wa;  
    input  [5:0] ra1;  
    input  [5:0] ra2;  
    input  [15:0] di;  
    output [15:0] do1;  
    output [15:0] do2;  
    reg    [15:0] ram [63:0];  
    always @(posedge clk)  
    begin  
        if (we)  
            ram[wa] <= di;  
    end  
    assign do1 = ram[ra1];  
    assign do2 = ram[ra2];  
endmodule
```

Multiple reference to same
array.

Block RAM Inference (FPGA)

```
//  
// Single-Port RAM with Synchronous Read  
//  
module v_rams_07 (clk, we, a, di, do);  
    input  clk;  
    input  we;  
    input  [5:0] a;  
    input  [15:0] di;  
    output [15:0] do;  
    reg    [15:0] ram [63:0];  
    reg    [5:0] read_a;  
    always @(posedge clk) begin  
        if (we)  
            ram[a] <= di;  
        read_a <= a;  
    end  
    assign do = ram[read_a];  
endmodule
```

**Synchronous read (registered
read address) infers Block RAM**

FPGA Block RAM initialization (FPGA)

```
module RAMB4_S4 (data_out, ADDR, data_in, CLK, WE);
    output[3:0] data_out;
    input [2:0] ADDR;
    input [3:0] data_in;
    input CLK, WE;
    reg [3:0] mem [7:0];
    reg [3:0] read_addr;

    initial
        begin
            $readmemb("data.dat", mem);
        end

    always@(posedge CLK)
        read_addr <= ADDR;

    assign data_out = mem[read_addr];

    always @(posedge CLK)
        if (WE) mem[ADDR] = data_in;

endmodule
```

“data.dat” contains initial RAM contents, it gets put into the bitfile and loaded at configuration time.

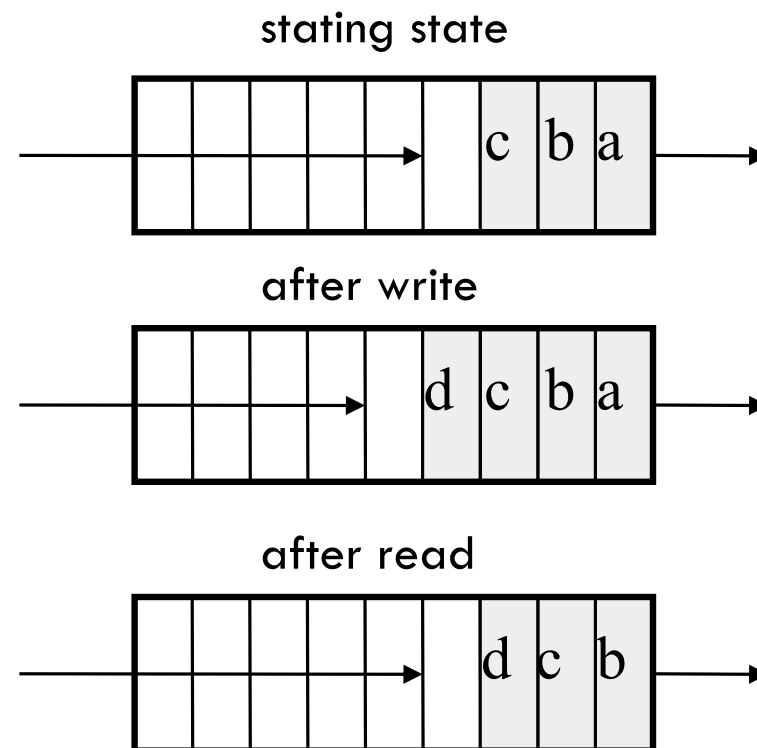
(Remake bits to change contents)



FIFOs

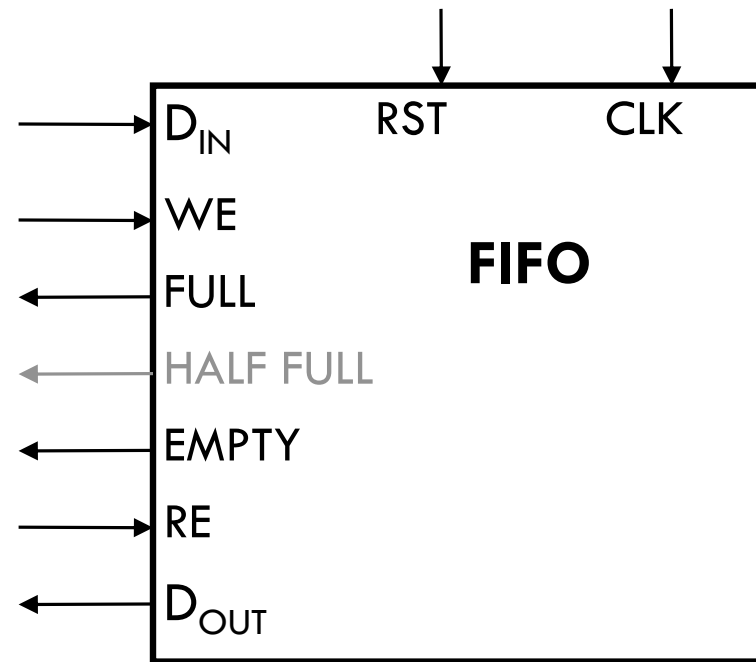
First-in-first-out (FIFO) Memory

- Used to implement *queues*.
- These find common use in processor and communication circuits.
- Generally, used to “decouple” actions of producer and consumer:



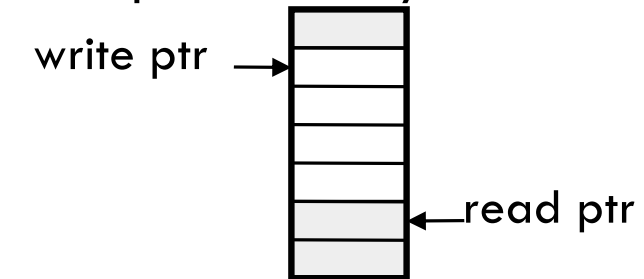
- Producer can perform many writes without consumer performing any reads (or vice versa). However, because of finite buffer size, on average, need equal number of reads and writes.
- Typical uses:
 - interfacing I/O devices. Example network interface. Data bursts from network, then processor bursts to memory buffer (or reads one word at a time from interface). Operations not synchronized.
 - Example: Audio output. Processor produces output samples in bursts (during process swap-in time). Audio DAC clocks it out at constant sample rate.

FIFO Interfaces

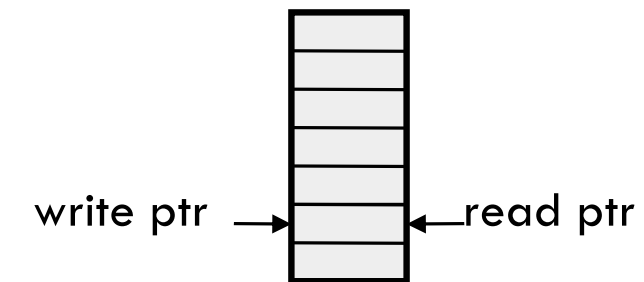


- After write or read operation, $FULL$ and $EMPTY$ indicate status of buffer.
- Used by external logic to control own reading from or writing to the buffer.
- FIFO resets to $EMPTY$ state.
- $HALF\ FULL$ (or other indicator of partial fullness) is optional.

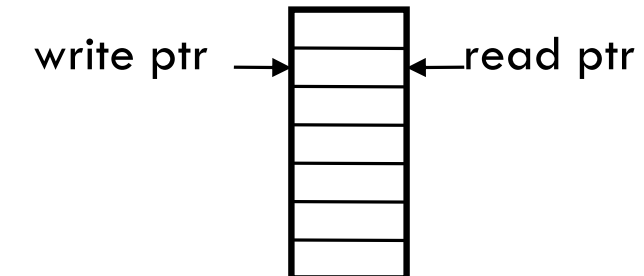
- Address pointers are used internally to keep next write position and next read position into a dual-port memory.



- If pointers equal after write $\Rightarrow FULL$:



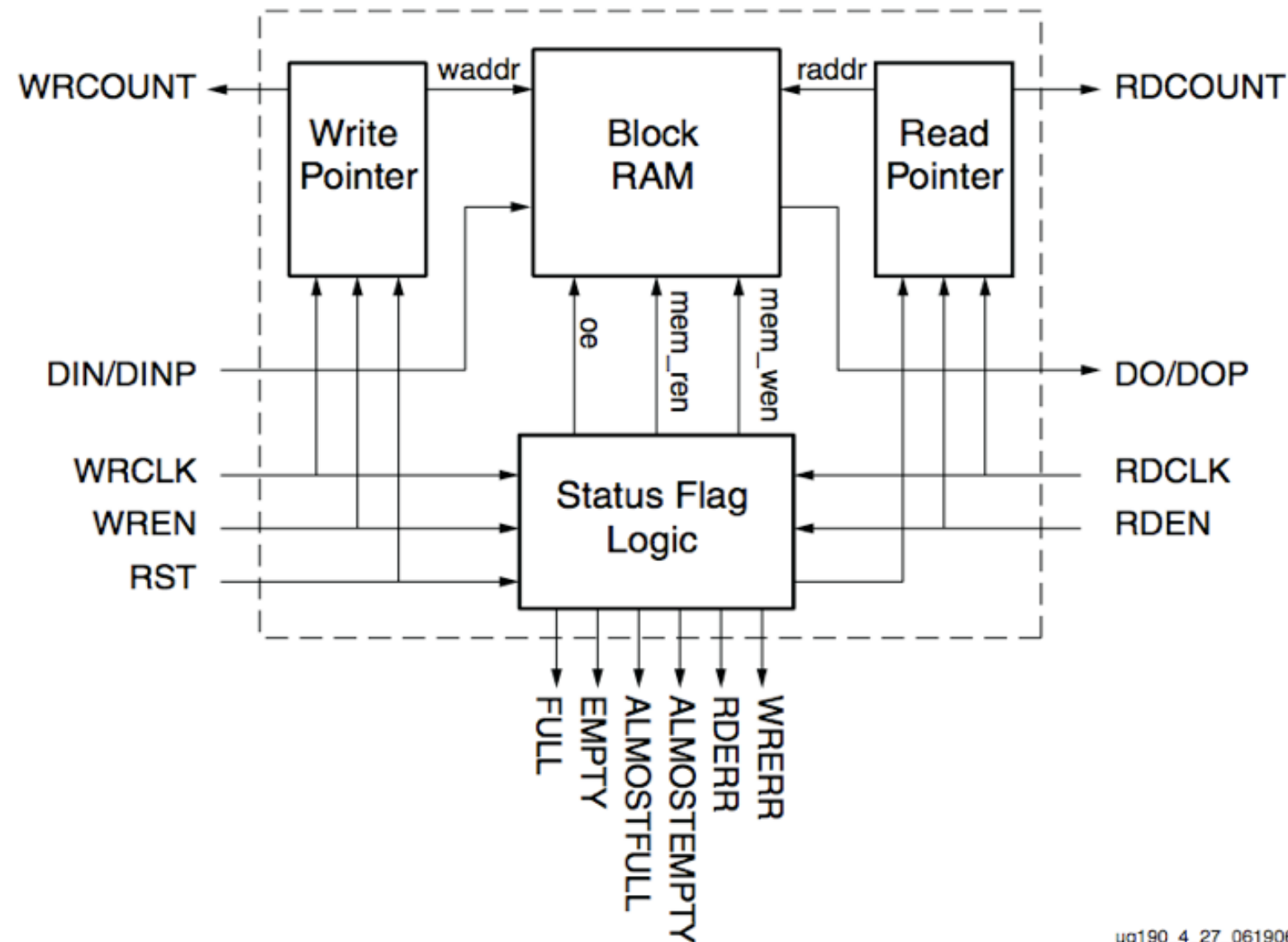
- If pointers equal after read $\Rightarrow EMPTY$:



Note: pointer incrementing is done "mod size-of-buffer"

Xilinx Virtex5 FIFOs

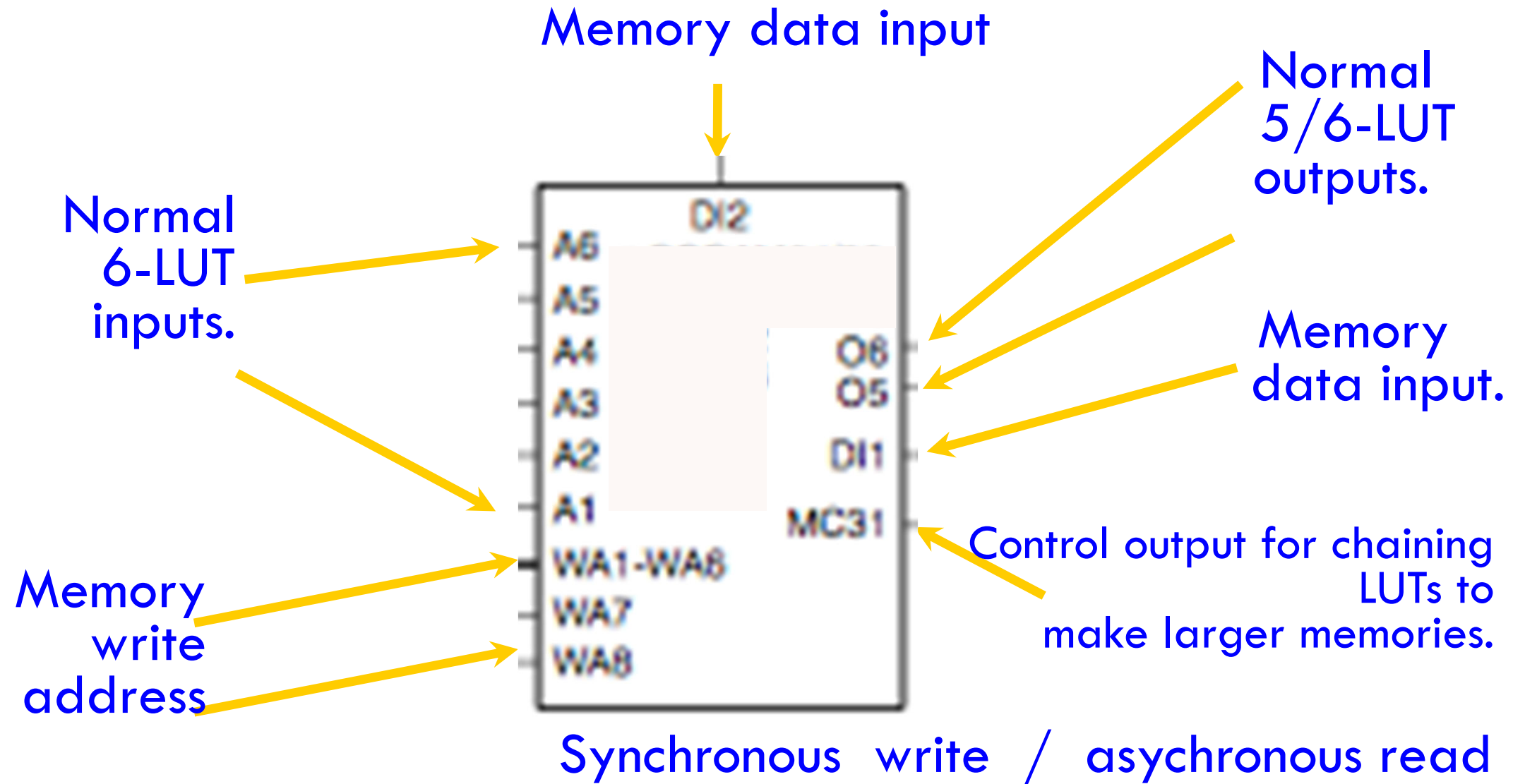
- Virtex5 BlockRAMS include dedicated circuits for FIFOs.
- Details in User Guide (ug190).
- Takes advantage of separate dual ports and independent ports clocks.





FPGA Memory Blocks

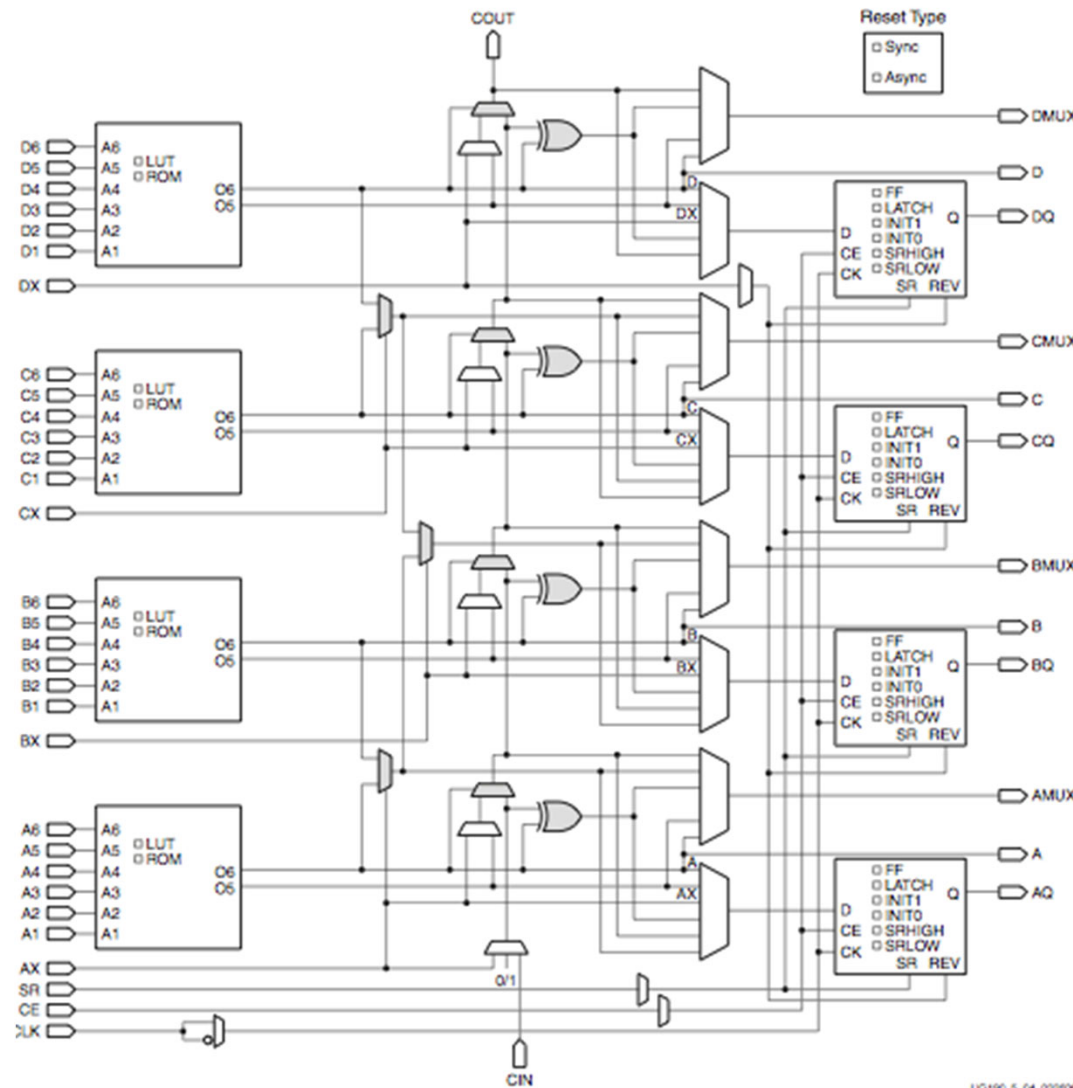
A SLICEM 6-LUT ...



A 1.1 Mb distributed RAM can be made if all SLICEMs of an LX110T are used as RAM.

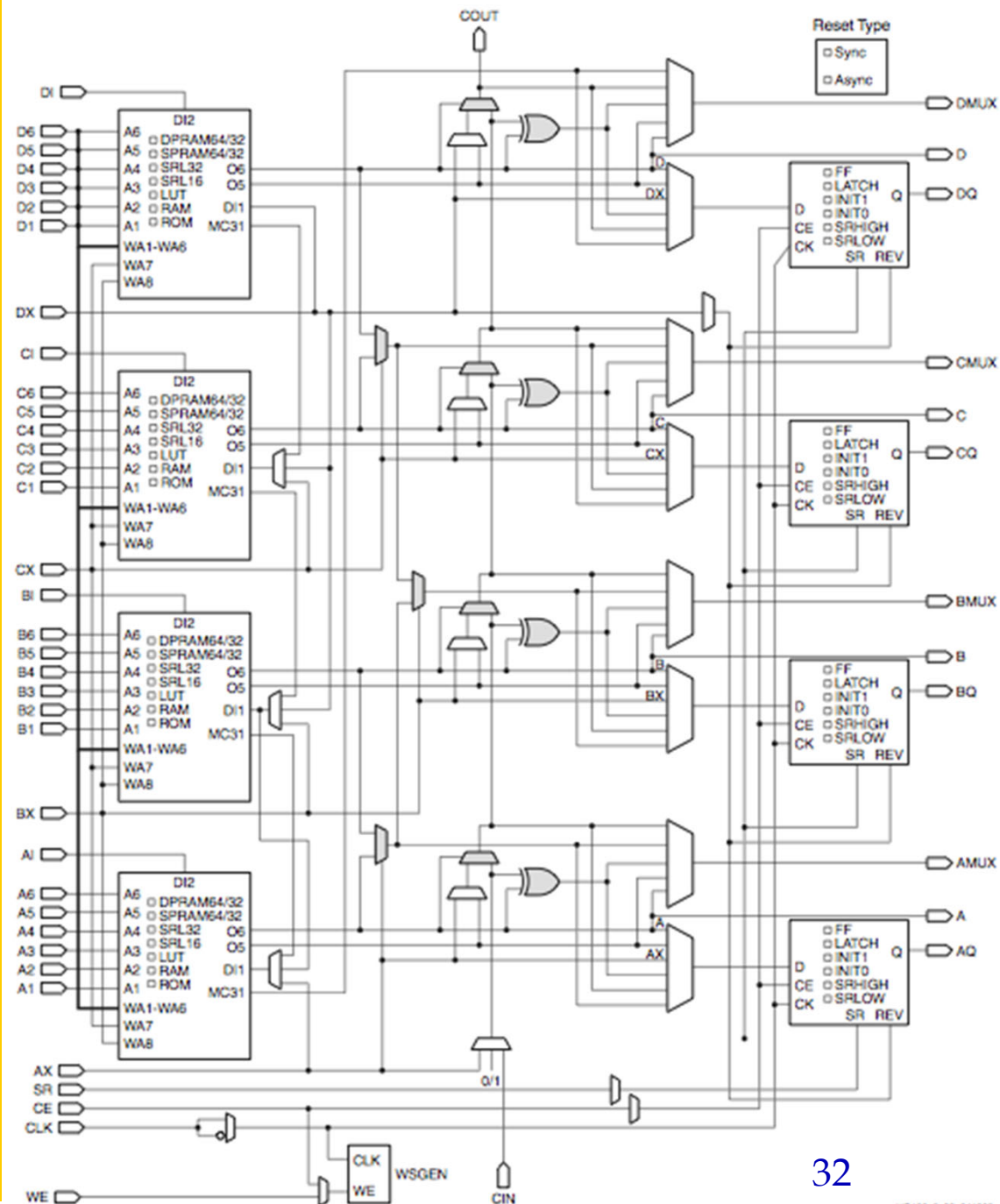
SLICEL vs SLICEM ...

SLICEL



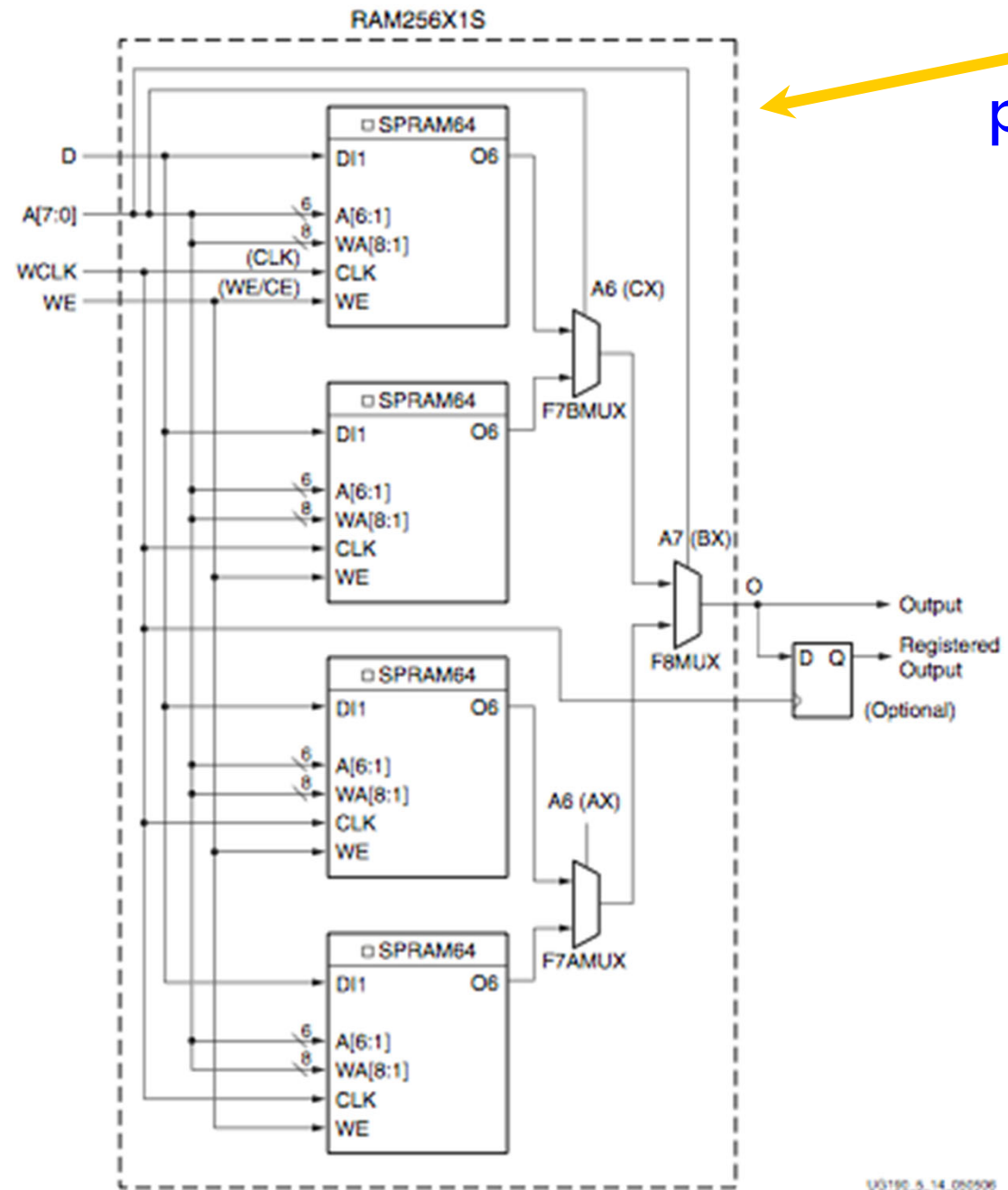
SLICEM adds memory features to LUTs, + muxes.

SLICEM



32

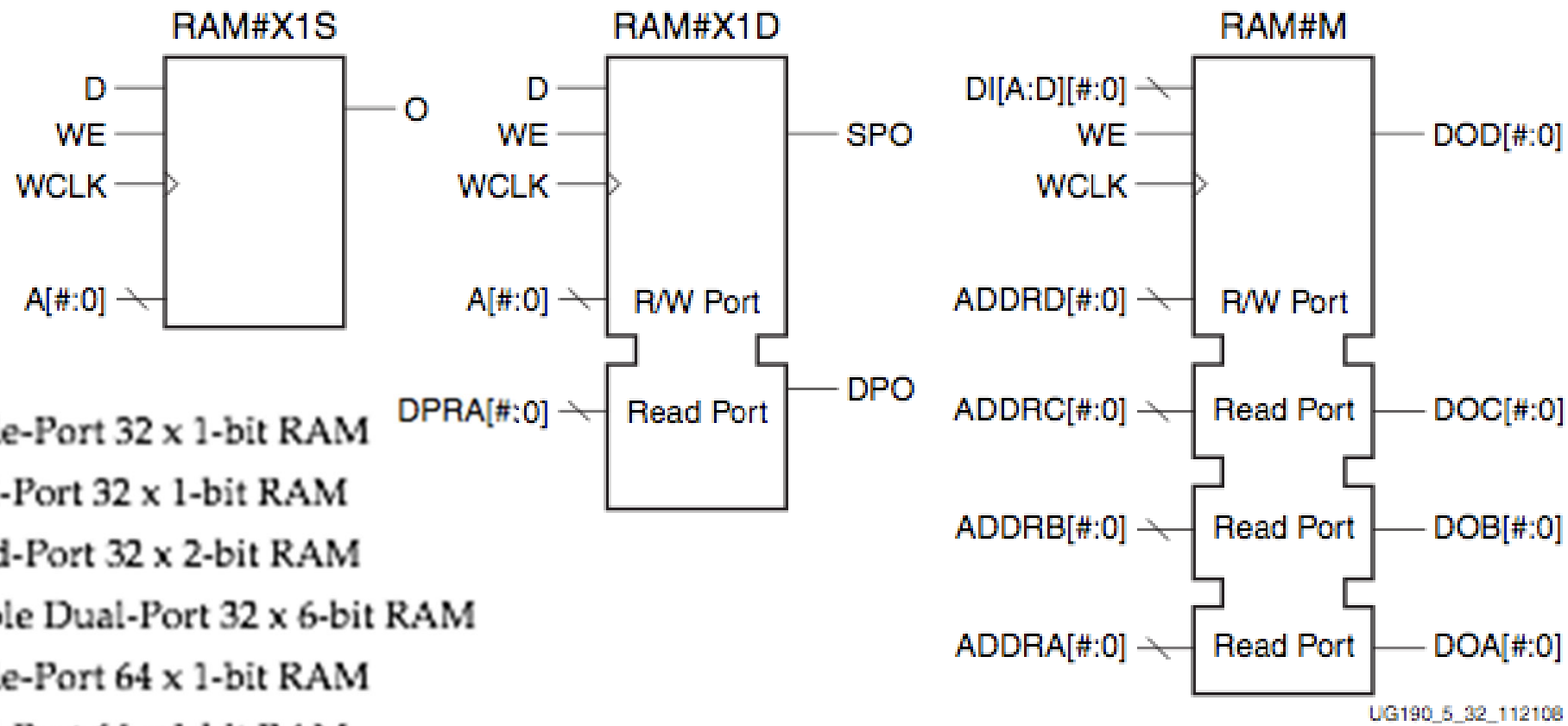
Example Distributed RAM (LUT RAM)



Example configuration: Single-port 256b x 1, registered output.

Figure 5-14: Distributed RAM (RAM256X1S)

Distributed RAM Primitives

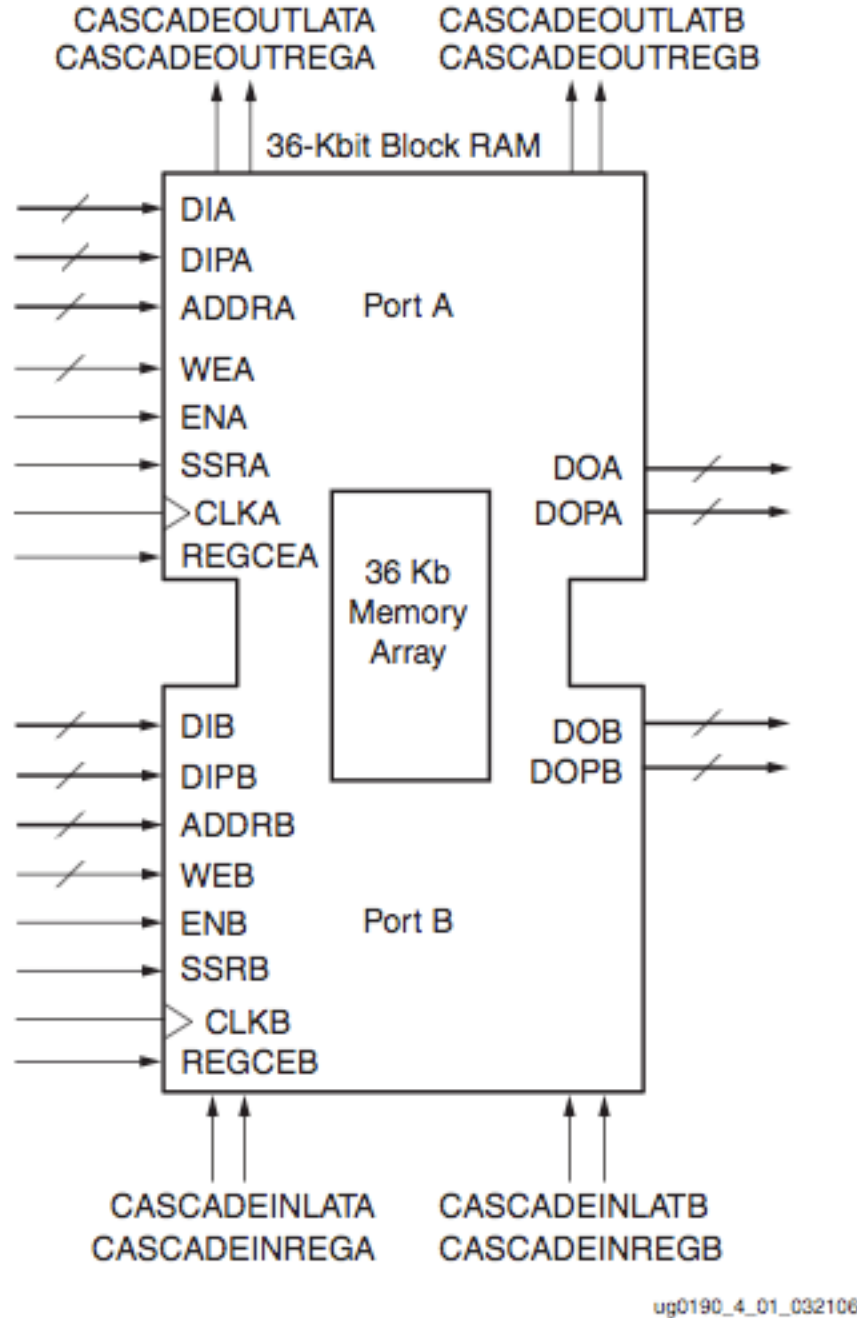


- Single-Port 32 x 1-bit RAM
- Dual-Port 32 x 1-bit RAM
- Quad-Port 32 x 2-bit RAM
- Simple Dual-Port 32 x 6-bit RAM
- Single-Port 64 x 1-bit RAM
- Dual-Port 64 x 1-bit RAM
- Quad-Port 64 x 1-bit RAM
- Simple Dual-Port 64 x 3-bit RAM
- Single-Port 128 x 1-bit RAM
- Dual-Port 128 x 1-bit RAM
- Single-Port 256 x 1-bit RAM

All are built from a single slice or less.

Remember, though, that the SLICEM LUT is naturally only 1 read and 1 write port.

Block RAM Overview



- 36K bits of data total, can be configured as:
 - 2 independent 18Kb RAMs, or one 36Kb RAM.
- Each 36Kb block RAM can be configured as:
 - 64Kx1 (when cascaded with an adjacent 36Kb block RAM), 32Kx1, 16Kx2, 8Kx4, 4Kx9, 2Kx18, or 1Kx36 memory.
- Each 18Kb block RAM can be configured as:
 - 16Kx1, 8Kx2, 4Kx4, 2Kx9, or 1Kx18 memory.
- Write and Read are synchronous operations.
- The two ports are symmetrical and totally independent (can have different clocks), sharing only the stored data.
- Each port can be configured in one of the available widths, independent of the other port. The read port width can be different from the write port width for each port.
- The memory content can be initialized or cleared by the configuration bitstream.

UltraRAM Blocks

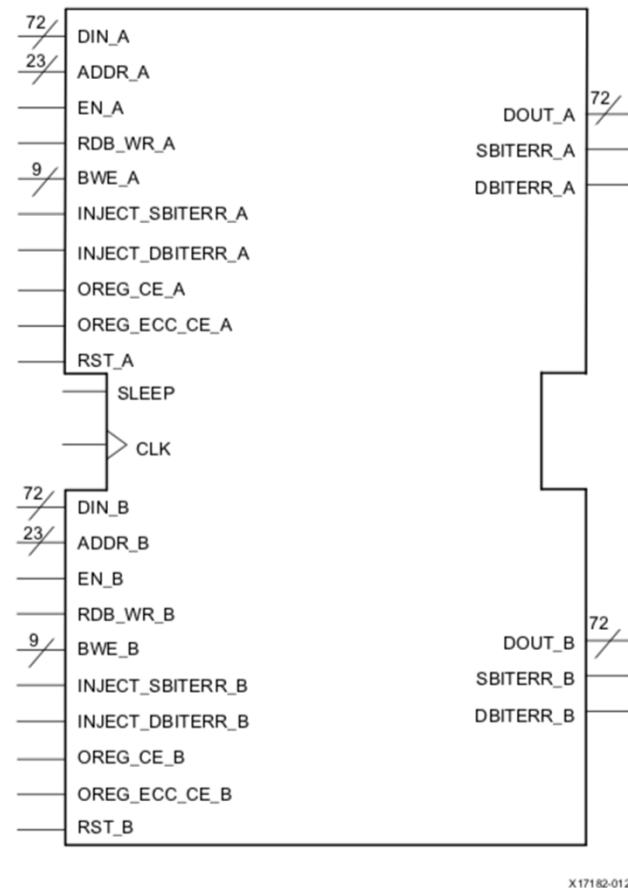


Figure 2-1: UltraRAM URAM288_BASE Primitive

Table 2-1: Block RAM and UltraRAM Comparison

Feature	Block RAM	UltraRAM
Clocking	Two clocks	Single clock
Built-in FIFO	Yes	No
Data width	Configurable (1, 2, 4, 9, 18, 36, 72)	Fixed (72-bits)
Modes	SDP and TDP	Two ports, each can independently read or write (a superset of SDP)
ECC	64-bit SECDED Supported in 64-bit SDP only (one ECC decoder for port A and one ECC encoder for port B)	64-bit SECDED One set of complete ECC logic for each port to enable independent ECC operations (ECC encoder and decoder for both ports)
Cascade	<ul style="list-style-type: none"> Cascade output only (input cascade implemented via logic resources) Cascade within a single clock region 	<ul style="list-style-type: none"> Cascade both input and output (with global address decoding) Cascade across clock regions in a column Cascade across several columns with minimal logic resources
Power savings	One mode via manual signal assertion	One mode via manual signal assertion

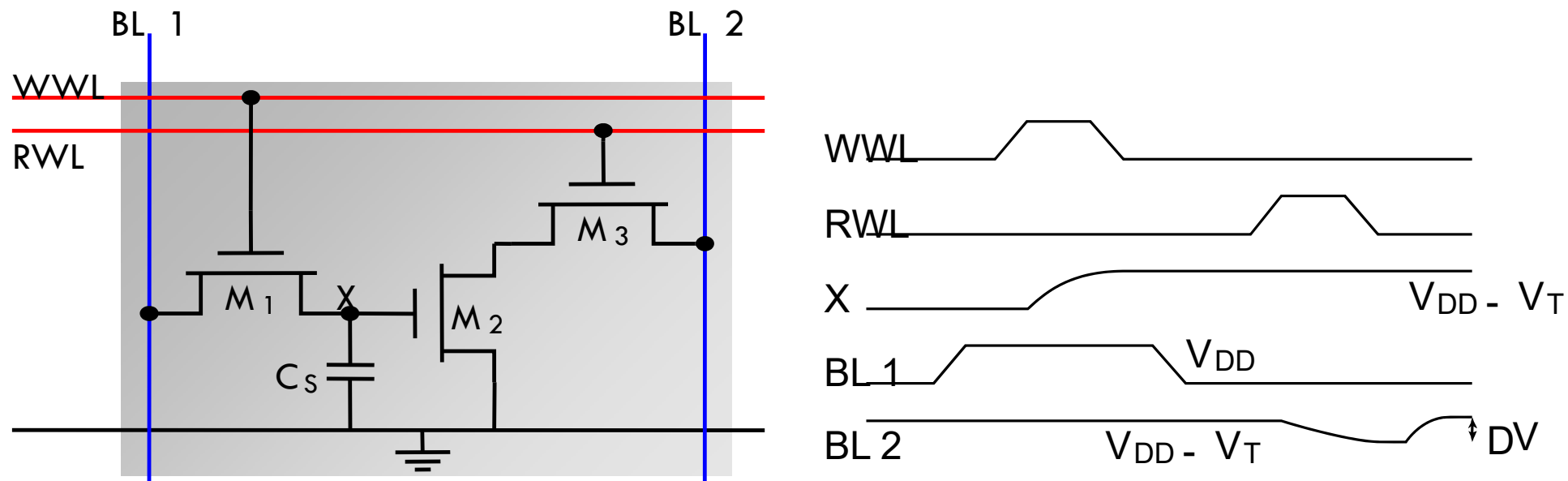
Administrivia

- Projects
 - Checkpoints 2 are this Friday
- Homework 9 due on *Monday, Nov. 25*
- No class on *Wednesday, Nov. 27*
- Last day of classes is Dec. 6



DRAM

3-Transistor DRAM Cell



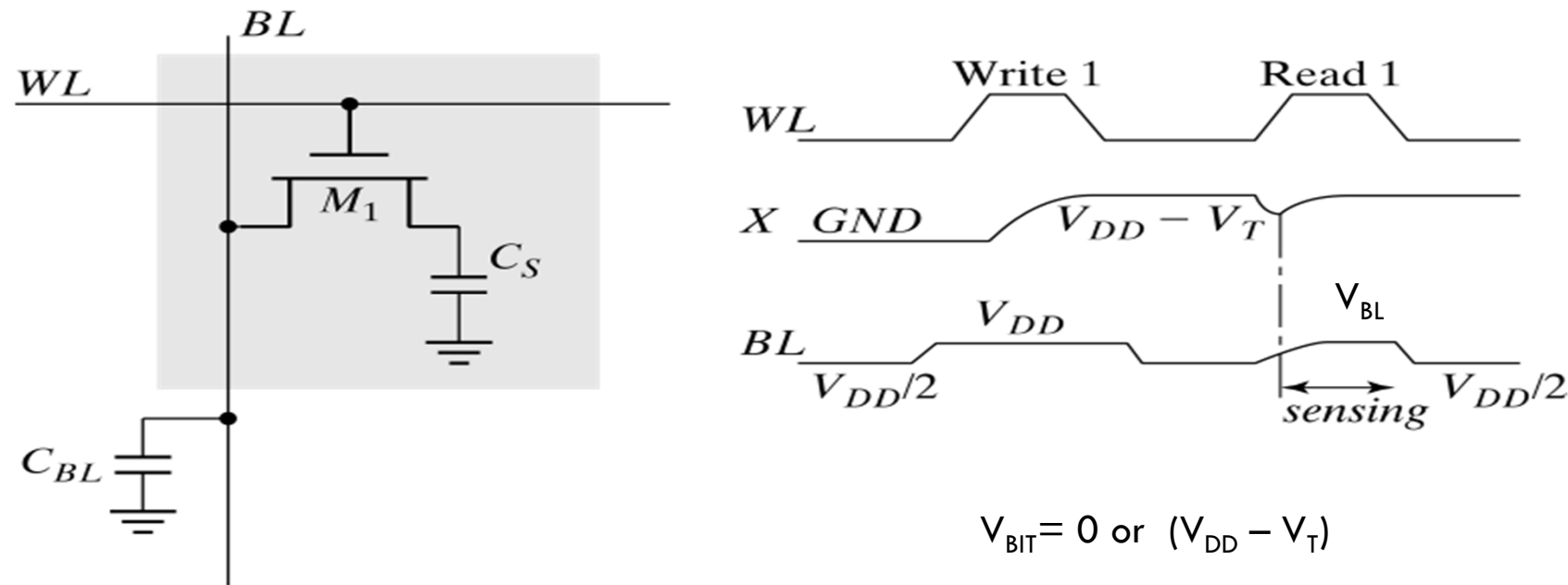
No constraints on device ratios

Reads are non-destructive

Value stored at node X when writing a "1" = $V_{WWL} - V_{Th}$

Can work with a logic IC process

1-Transistor DRAM Cell



Write: C_S is charged or discharged by asserting WL and BL.
Read: Charge redistribution takes places between bit line and storage capacitance

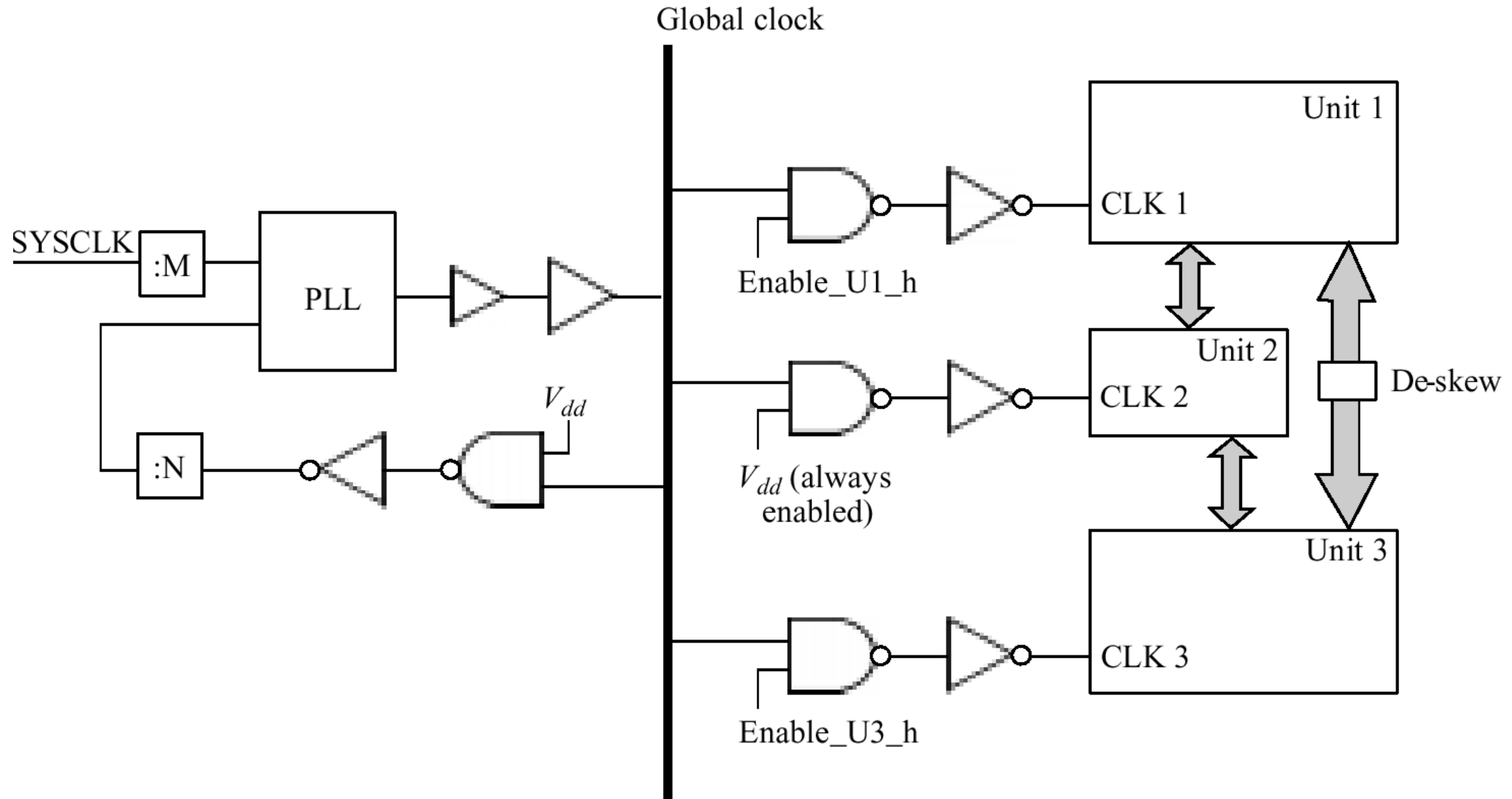
$C_S \ll C_{BL}$ Voltage swing is small; typically hundreds of mV.

- To get sufficient C_S , special IC process is used
- Cell reading is destructive, therefore read operation always is followed by a write-back
- Cell loses charge (leaks away in ms - highly temperature dependent), therefore cells occasionally need to be “refreshed” - read/write cycle



Clocking

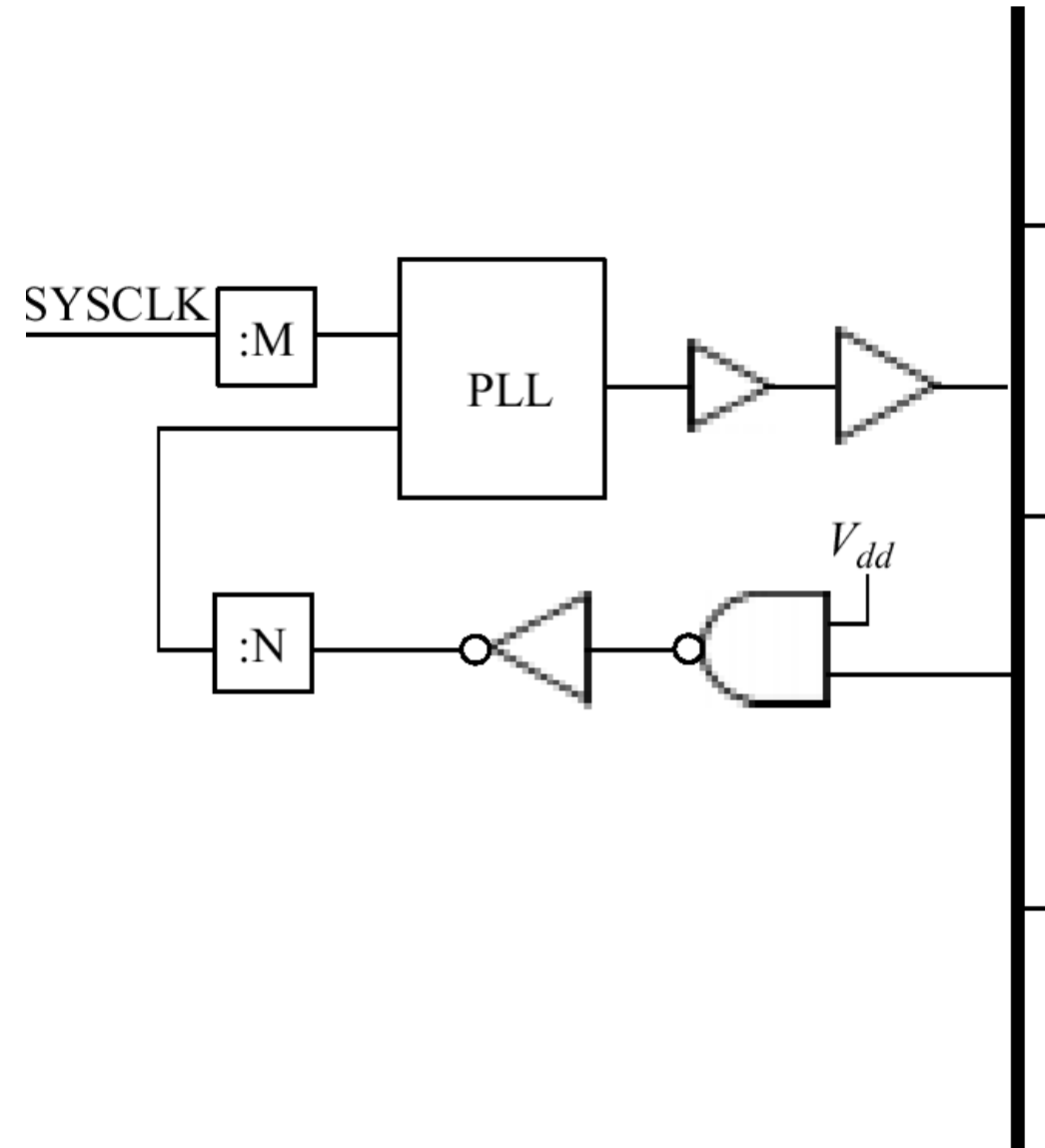
Example Clock System



Courtesy of IEEE Press, New York. © 2000

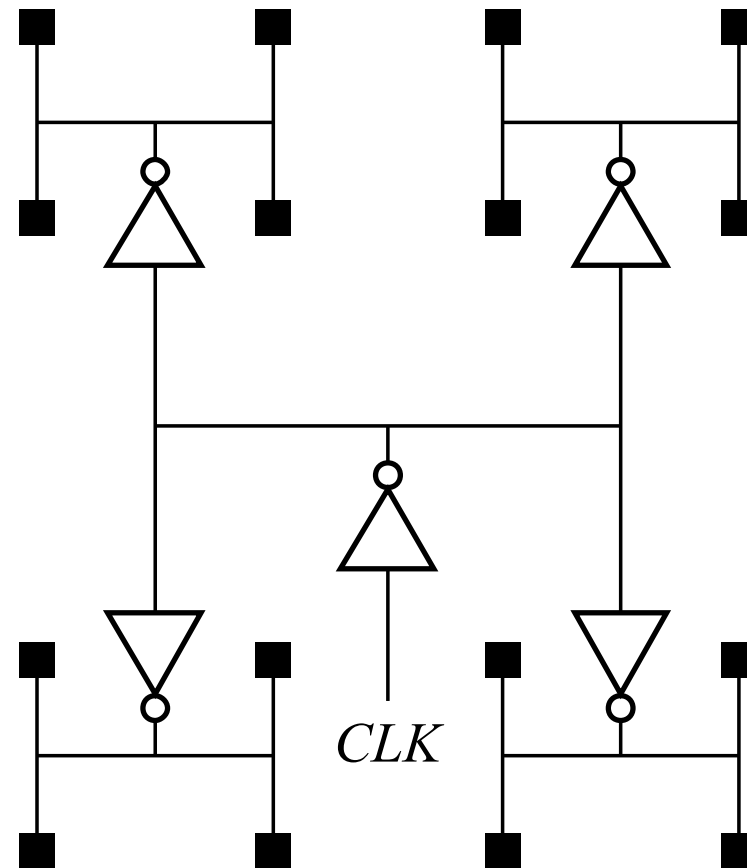
Clock Generation: Phase-Locked Loop

- Phase-locked loop (PLL)
 - Used for clock frequency synthesis
 - $N/M \times F_{\text{SYSCLK}}$
- Locks output clock phase to input clock phase
 - Can multiply clock frequency
- Multiple PLLs on a SoC
 - One per core/clock domain
- Can be analog or digital
 - Low phase noise (jitter)



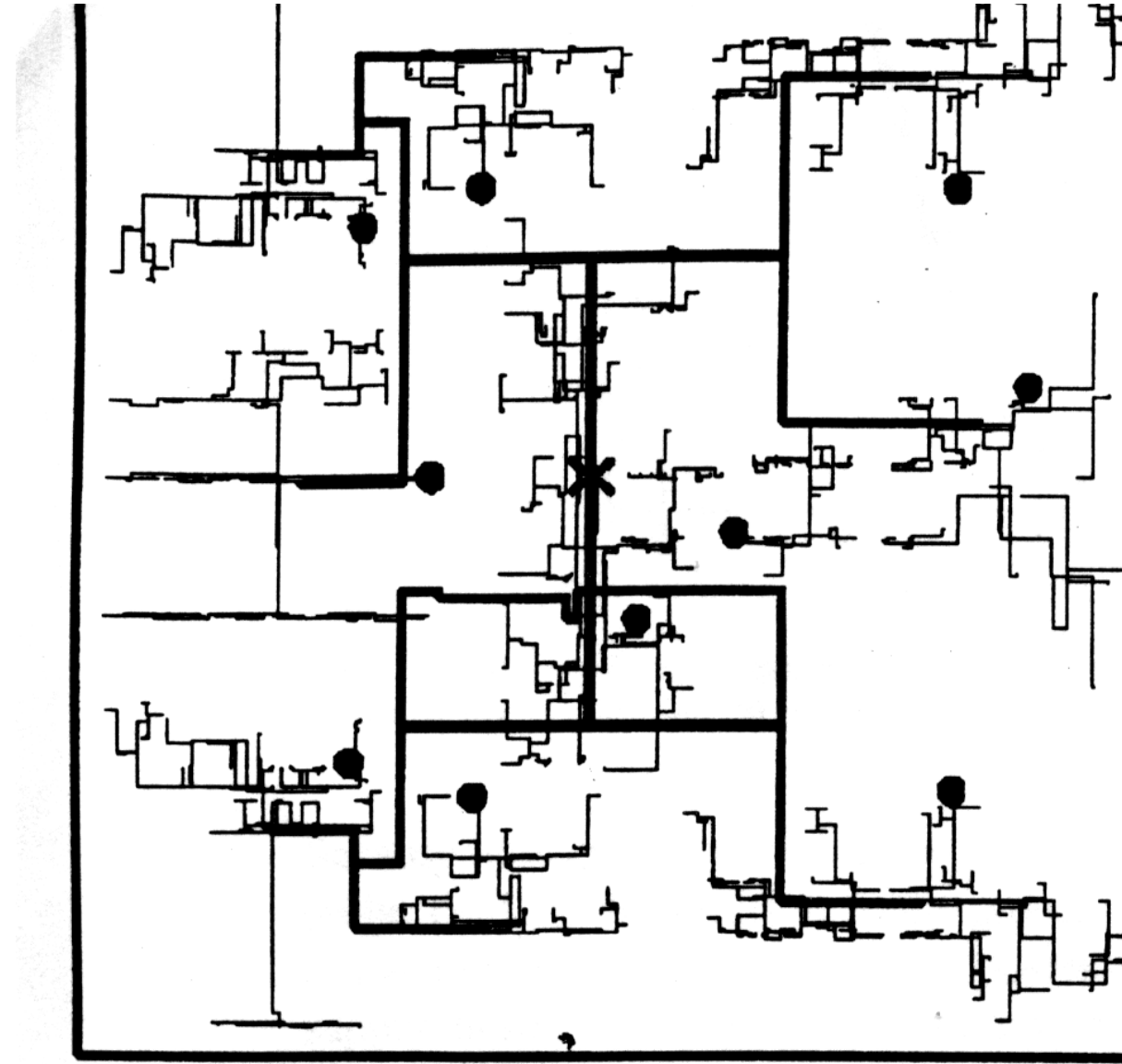
Clock Distribution

H-tree



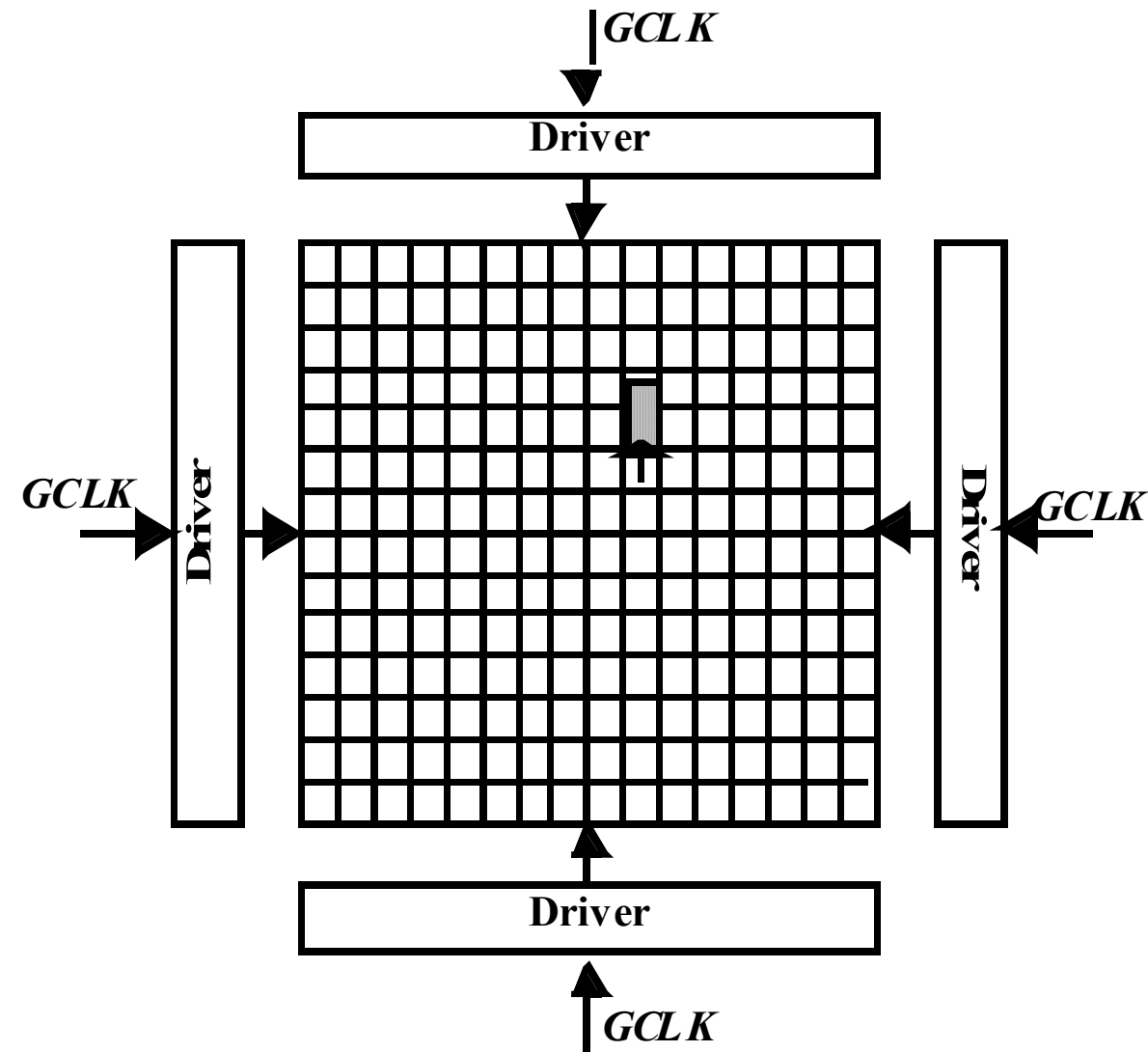
Clock is distributed in a tree-like fashion
Large chips (blocks) – many levels of buffers
Goal minimize skew, supply-induced jitter

More realistic H-tree



[Restle98]

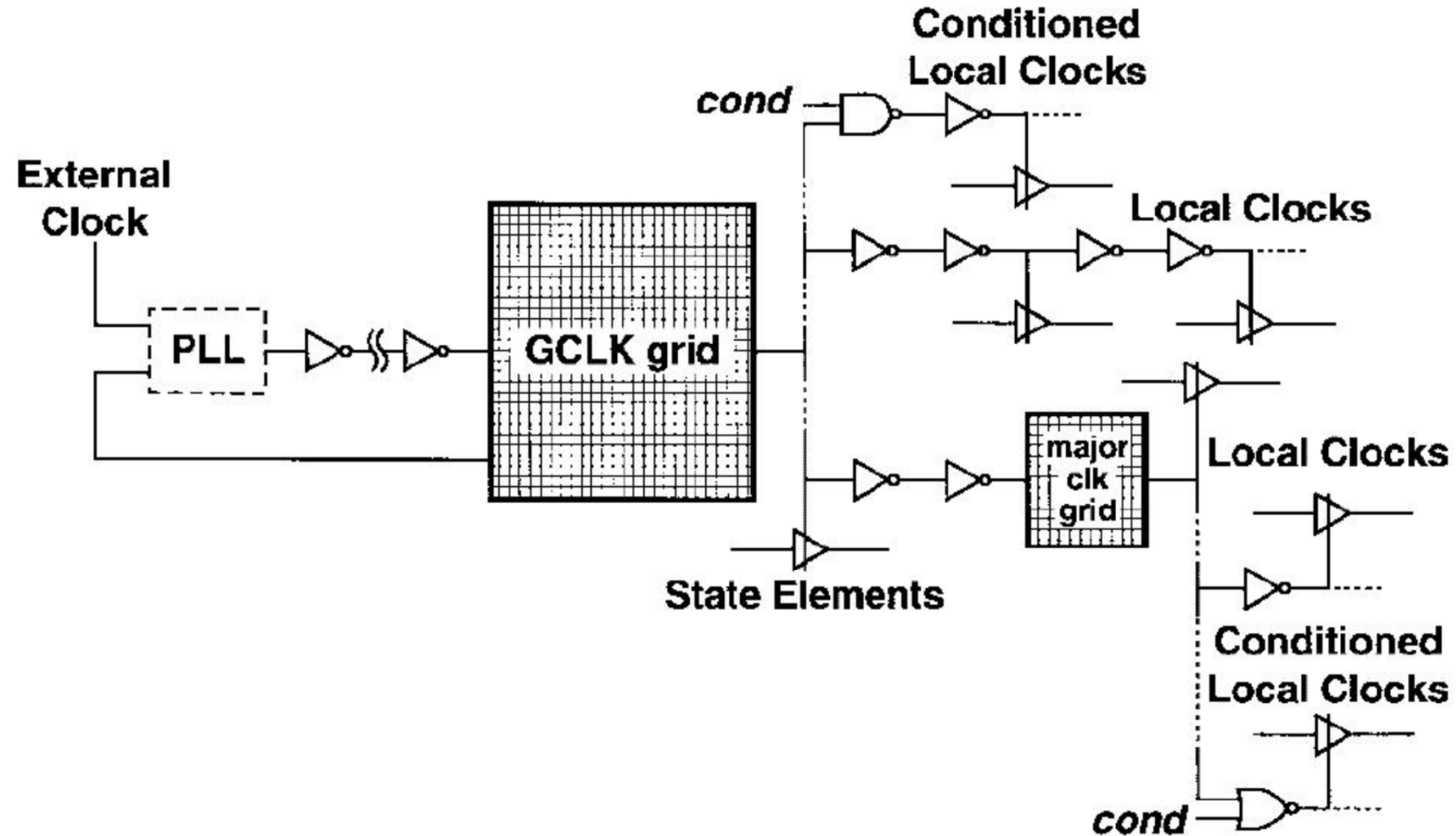
The Grid System



- Relaxed design constraints for low skew
- Large power

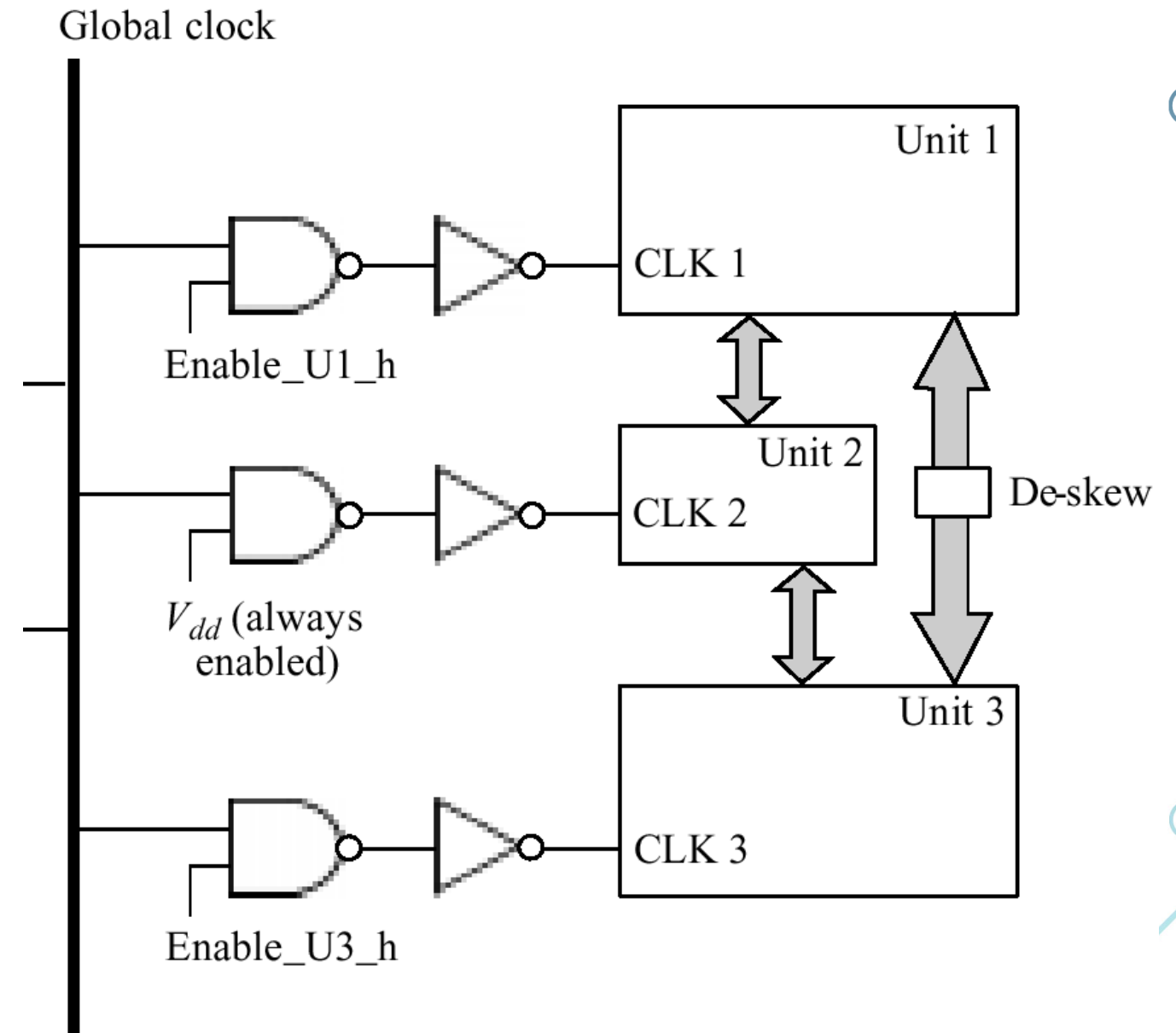
DEC Alpha 21264 Clocking

- Highest-performance processor of its time



Clock Gating

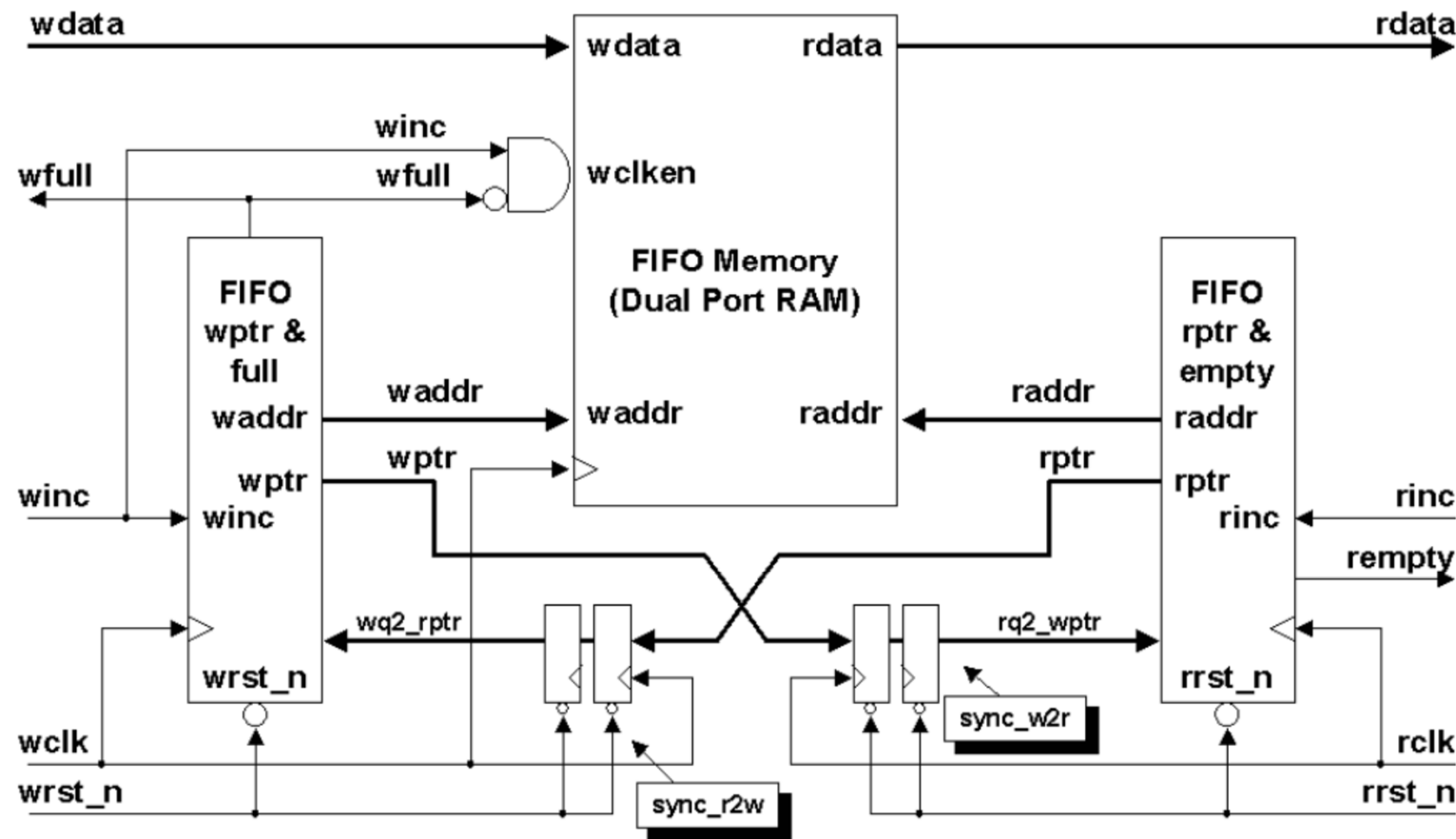
- Clock consumes 20-40% of power in a chip
 - Gate off in inactive blocks
- Multiple levels of clock gating



What Happens When We Un-Gate Clock?

Crossing Clock Domains

- Two domains at different frequencies exchange wdata, rdata
- FIFO with two clocks



http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO1.pdf

Summary

- Memory compilers generate SRAM blocks
- Several options for memory on FPGAs: Distributed, BlockRAM, UltraRAM
- Clock generation and distribution is a major part of digital system design
 - We just touched on it