ECE 4514
Digital Design II
Spring 2008

Lecture 16:
Synthesis of Memories in FPGA

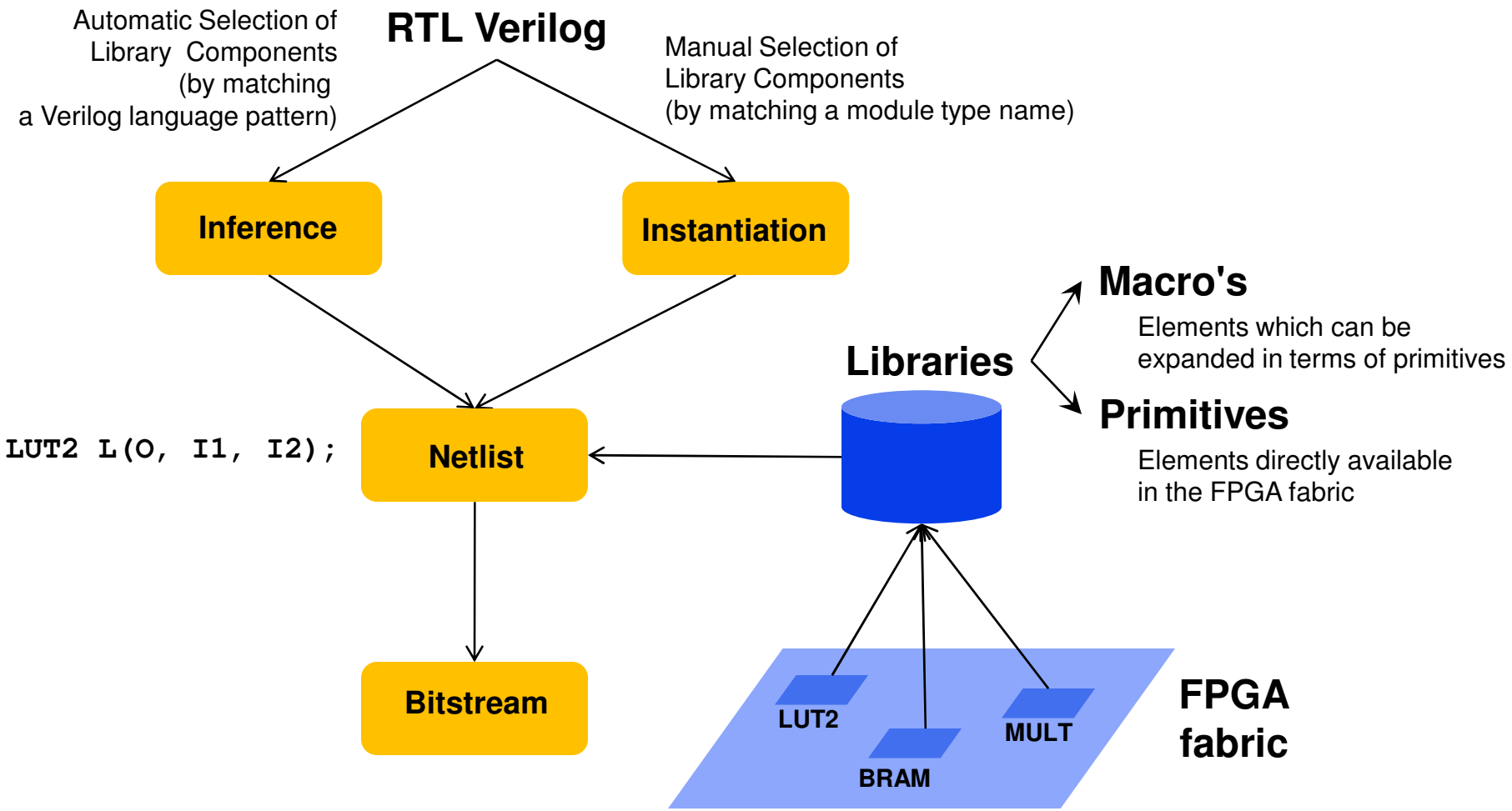
A *Design* Lecture

Patrick Schaumont

Overview

- Design flow for Datapath and Memory Elements
  - Inference and instantiation
  - Integrated synthesis and simulation
- Memory Elements
  - Register Files
  - Memory Arrays: SRAM and DRAM
  - Operation of SRAM and DRAM
  - SRAM in Spartan 3E FPGA
    - Configuration: single/dual port, read/write ordering
    - Inferring SRAM with Verilog
    - Initializing SRAM, $readmemb and $readmemh
- Overview of Project 4 & 5

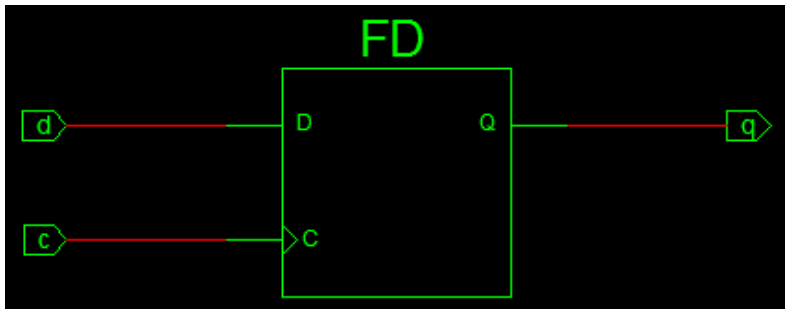
Design flow for datapath & memory elements



Example 1 (primitive): Flip-flop

- This piece of code *infers* a flip-flop

```
module my_flipflop_i(q, c, d);
  output q;
  reg    q;
  input  c;
  input  d;

  always @(posedge c)
    q <= d;

endmodule
```

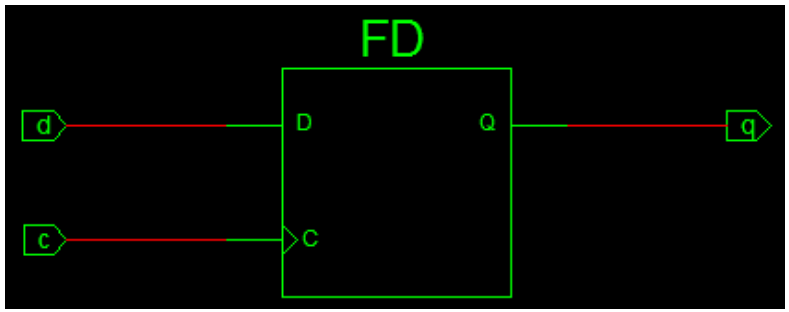


This is a *primitive* library element

Example 1 (primitive): Flip-flop

- This piece of code *instantiates* a flip-flop

```
module my_flipflop_l(q, c, d);
  output q;
  input  c;
  input  d;

  FD FD_instance (q, c, d);

endmodule
```



This is a primitive library element

How do we know what primitives are available?

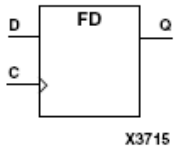
- "Xilinx FPGA Libraries Guide"
  - lists all available primitive elements
  - available parameters, availability over FPGA families, ...

FD

D Flip-Flop

Architectures Supported

FD	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FD is a single D-type flip-flop with data input (D) and data output (Q). The data on the D inputs is loaded into the flip-flop during the Low-to-High clock (C) transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

How do we know what primitives are available?

- "Xilinx FPGA Libraries Guide"
  - FD entry shows how to instantiate it in Verilog or VHDL

Verilog Instantiation Template

```
FD FD_instance_name (.Q (user_Q),
                     .C (user_C),
                     .D (user_D));

defparam FD_instance_name.INIT = bit_value;
```

Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

Example 2 (macro): 4-bit adder

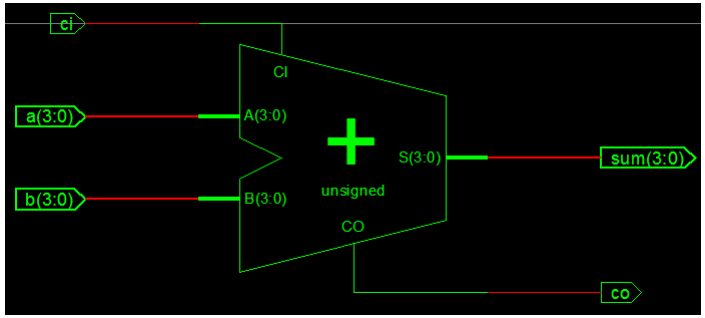
- This code infers a 4-bit adder

```
module a1(sum, co, a, b, ci);
  output [3:0] sum;
  reg [3:0]    sum;
  output       co;
  reg          co;
  input [3:0]  a, b;
  input        ci;

  always @(a or b or ci)
  begin
    {co, sum} <= a + b + ci;
  end
endmodule
```

Example 2 (macro): 4-bit adder

- 4-bit adder synthesizes as a *macro* (not a primitive)

```
=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# Adders/Subtractors                                   : 1
 4-bit adder carry in/out                              : 1
```



Since this is a *macro*, it will be expanded into FPGA primitives (LUTs, XORchain, registers,...)

Inference vs Instantiation

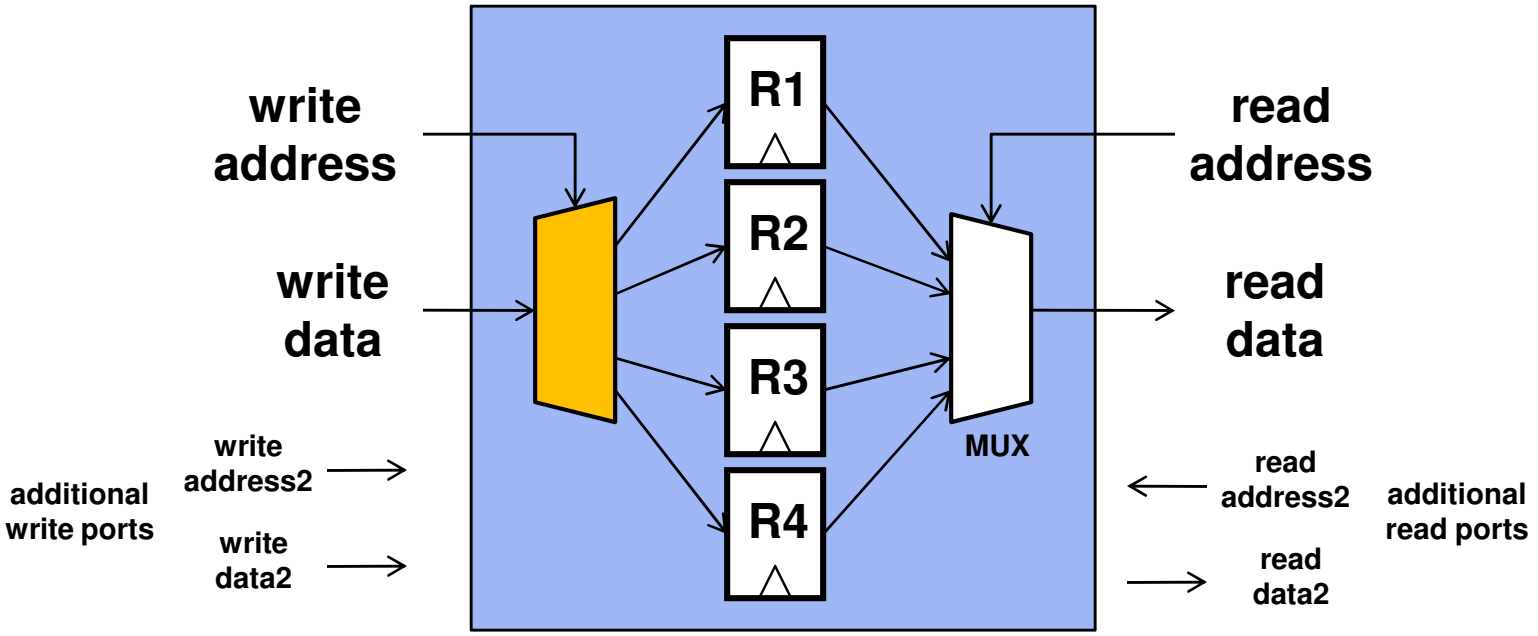
- Standard design practice (for any type of digital design: FPGA, ASIC) will *mix* inference and instantiation
- Inference:
  - Portable code which compiles on multiple targets
  - Portable code which simulates with generic simulator
  - Provides maximal freedom to synthesis tools
- Instantiation:
  - Nonportable code which makes specific assumption on target
  - Nonportable code which requires simulation library to simulate
  - Gives designer maximum control over synthesis process

Integrated Synthesis and Simulation

- Once we use design elements from a library, we need to add a simulation library specific to the target technology
  - Simulation library may provide a behavioral model of primitives and macro's, to implement the leaf cells of your design
- In addition, after synthesis, additional design details help to estimate precisely the delay incurred by each component and each net. This can also be included in the simulation.
  - Simulation library may provide detailed structural model of primitives, which are decorated with delay figures from tools.
- See lecture 20: Timing Analysis & Simulation

Memory Elements: Register Files

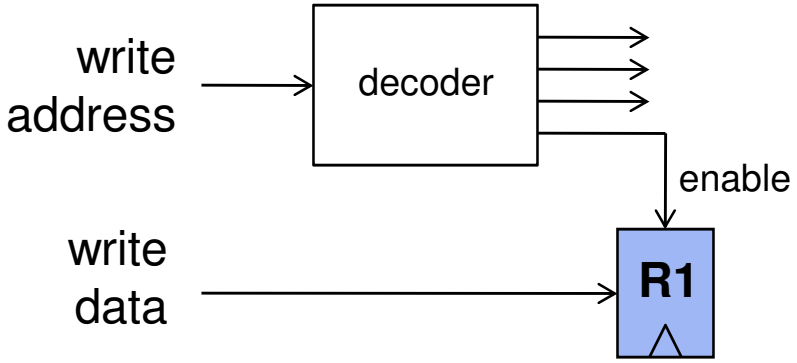
- Used when multiple registers are attached to single bus
- May be multi-ported and/or concurrent read/write



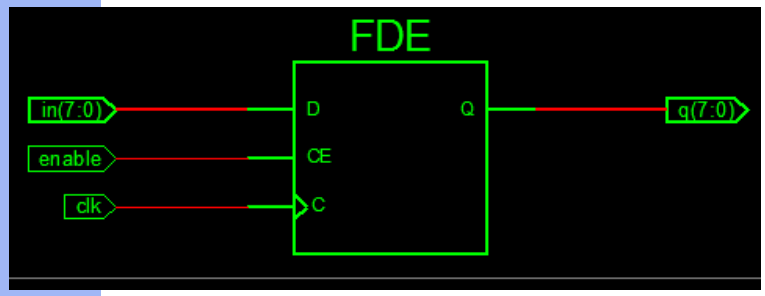
How to implement this ?

Memory Elements: Register Files

- Single write-port : load-enable edge-triggered reg

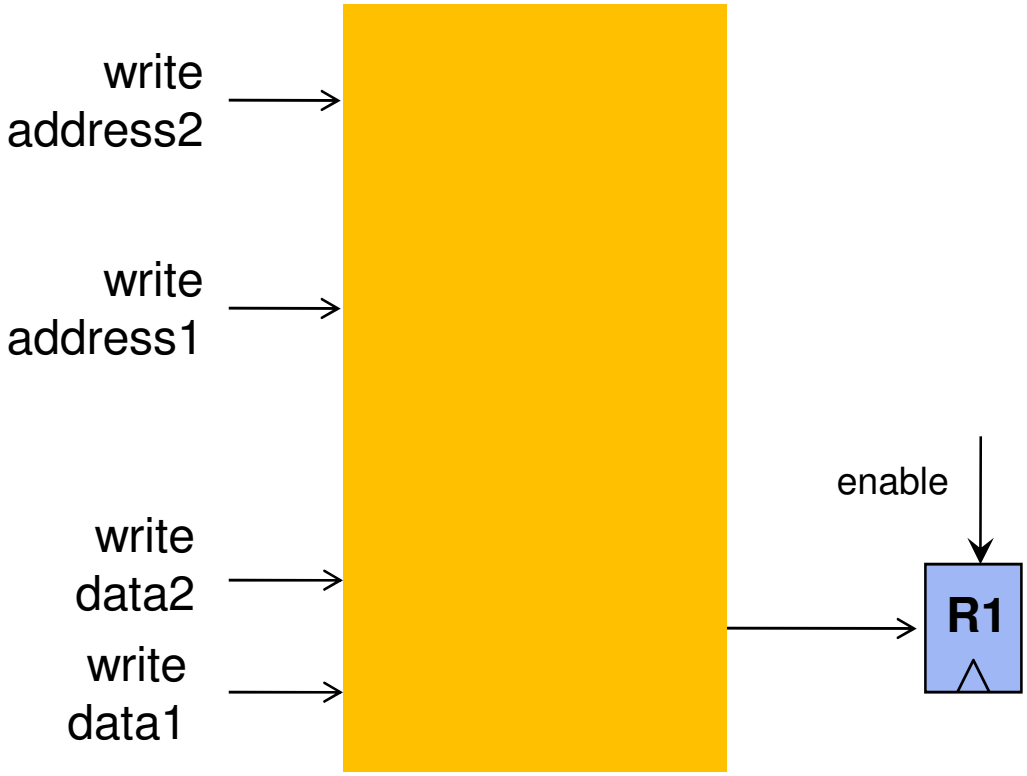



```
module r1(q, clk, enable, in);
  output [7:0] q;
  reg    [7:0] q;
  input  clk, enable;
  input  [7:0] in;

  always @(posedge clk)
    if (enable)
      q <= in;

endmodule
```



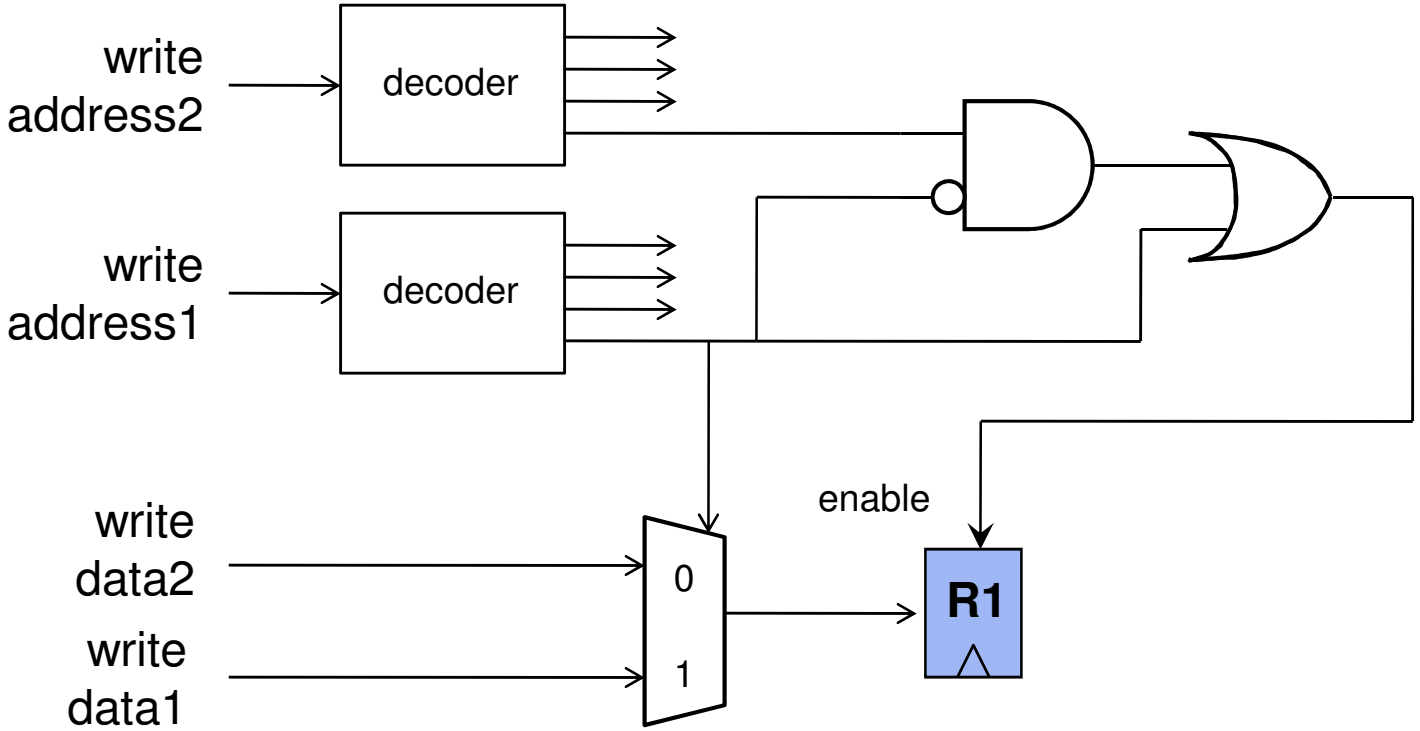
Memory Elements: Register Files

- How to implement dual write-port ?



Memory Elements: Register Files

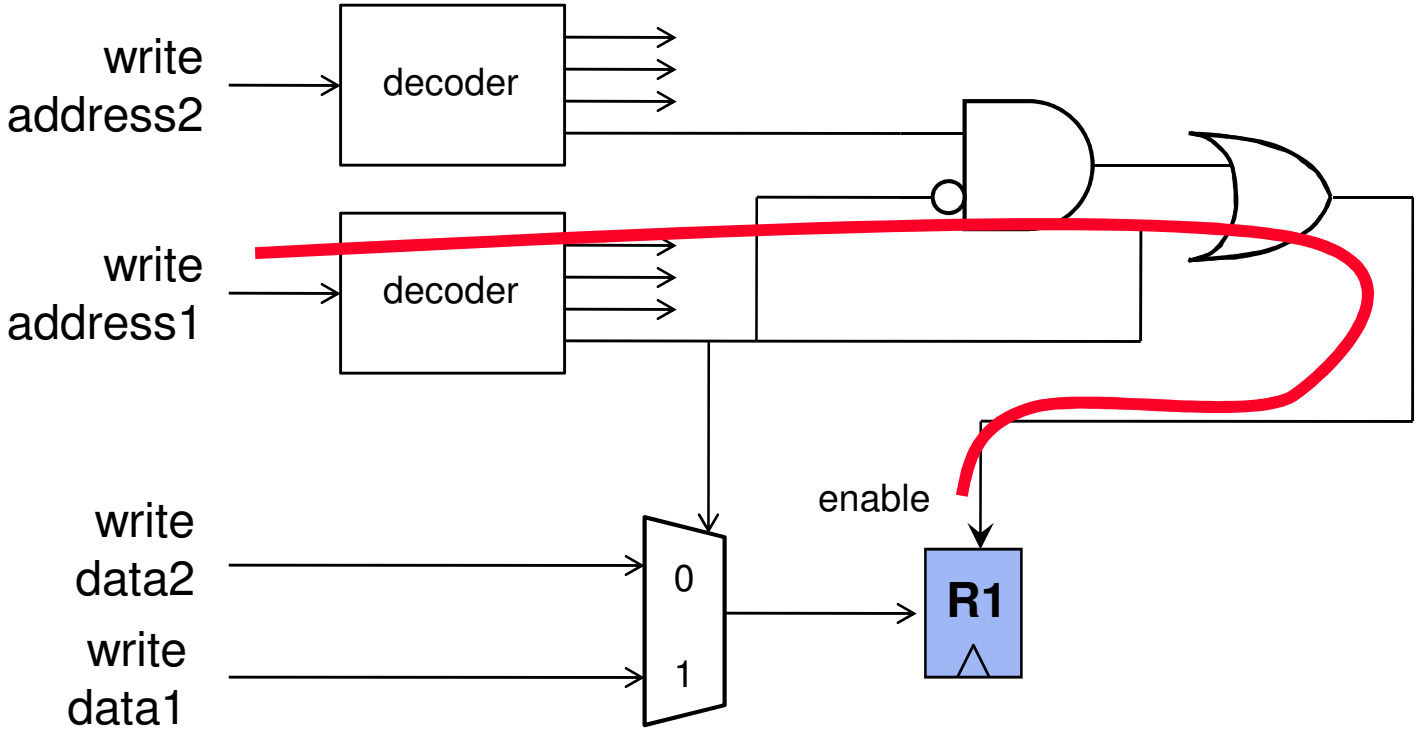
- Dual write-port : load-enable edge-triggered reg, mux and priority



What is the (most likely) critical path of this design ?

Memory Elements: Register Files

- Dual write-port : load-enable edge-triggered reg, mux and priority



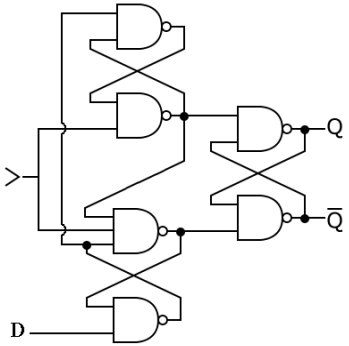
What is the (most likely) critical path of this design ?

Multi-port registers are slower then single-port registers

Memory Arrays

- Flip-flop based on gates is area- and power-hungry
- More dense solutions arrange bit cells in an array
  - static random access memory (SRAM)
  - dynamic random access memory (DRAM)

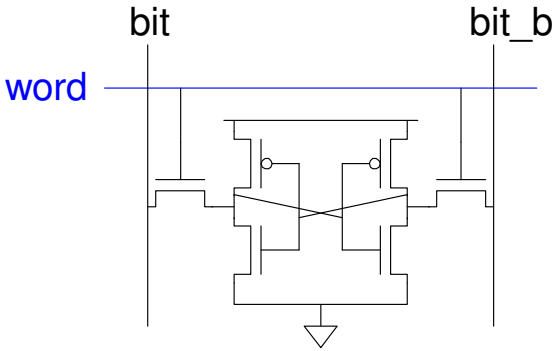
D flip-flop



>20 transistors/bit

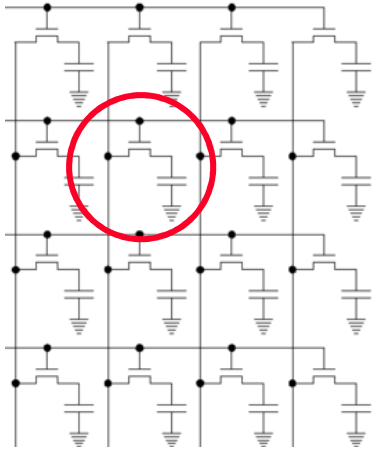
Basic SRAM Cell

[D. Harris]



6 transistors/bit

Basic DRAM Cell

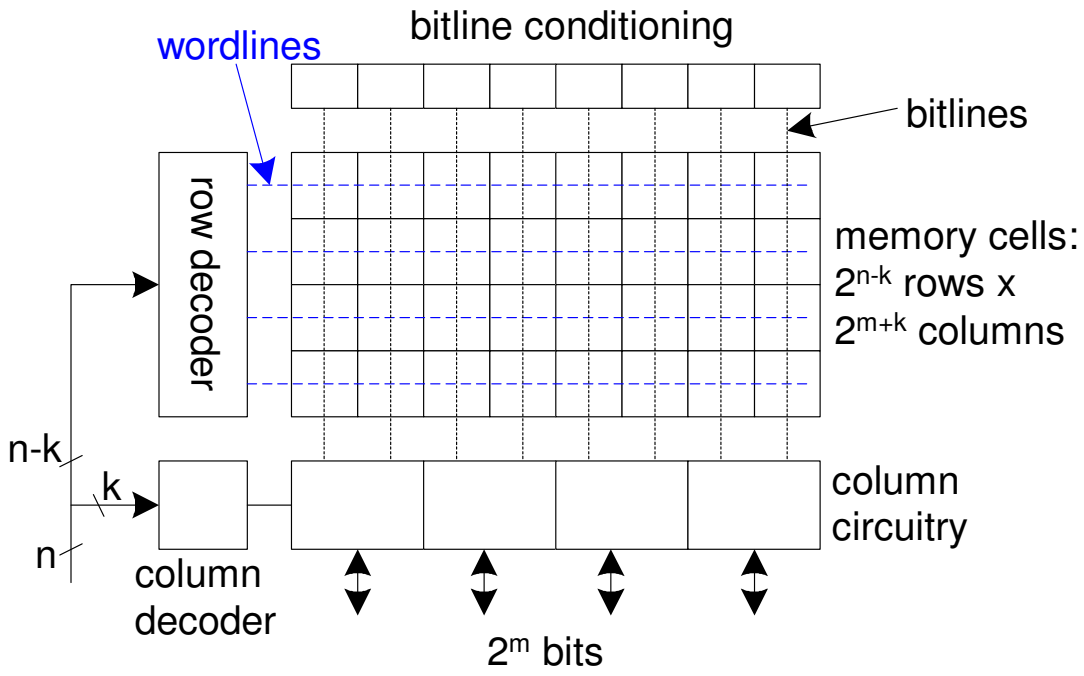


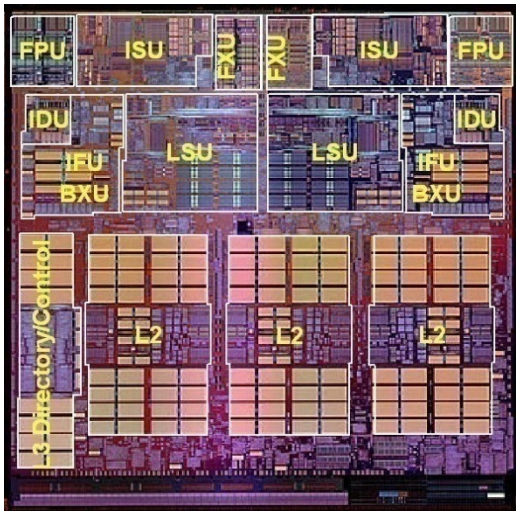
1 transistor/bit

Generic SRAM Architecture

- m data bits per address, 2^n addresses

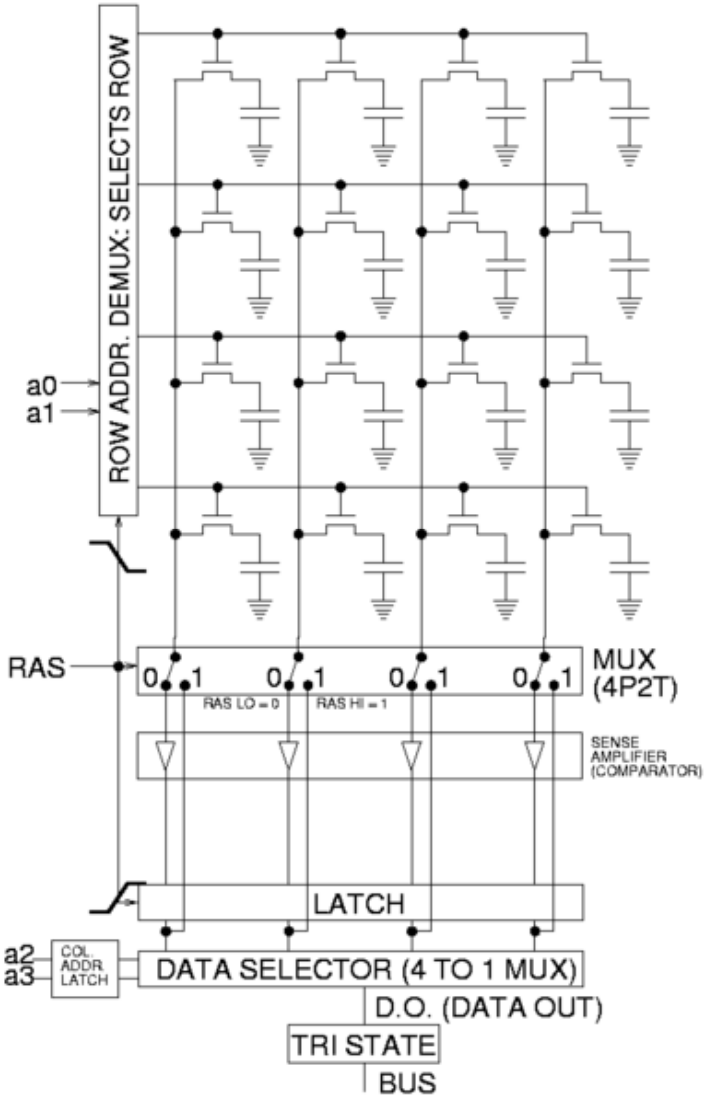
[D. Harris]





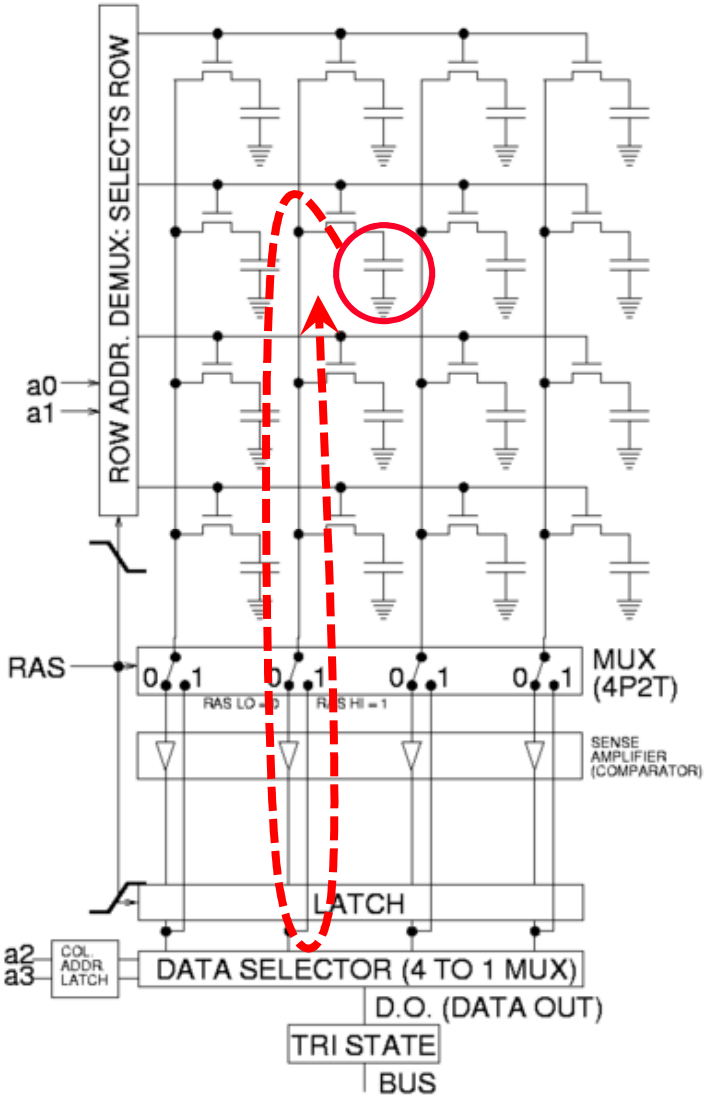
IBM Power4
(LSU, L2) Caches have regular SRAM structures, which are easy to spot

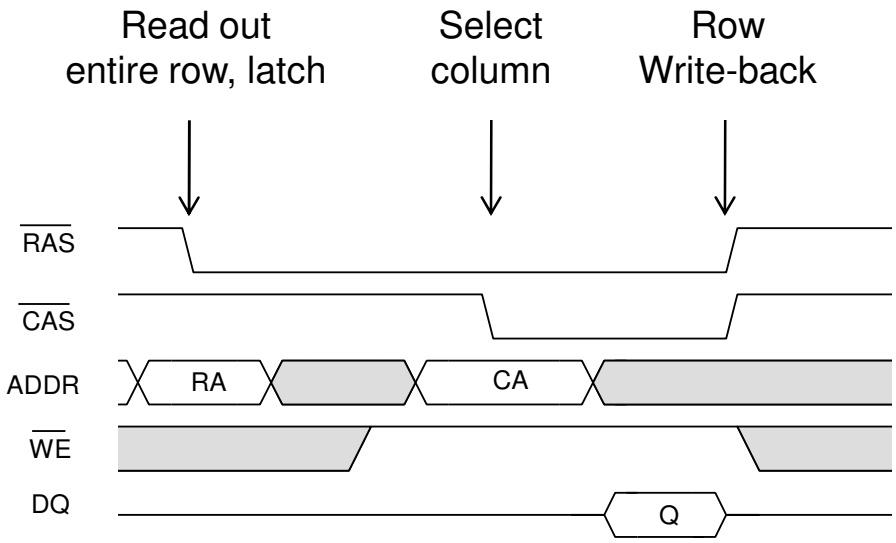
Dynamic RAM Architecture



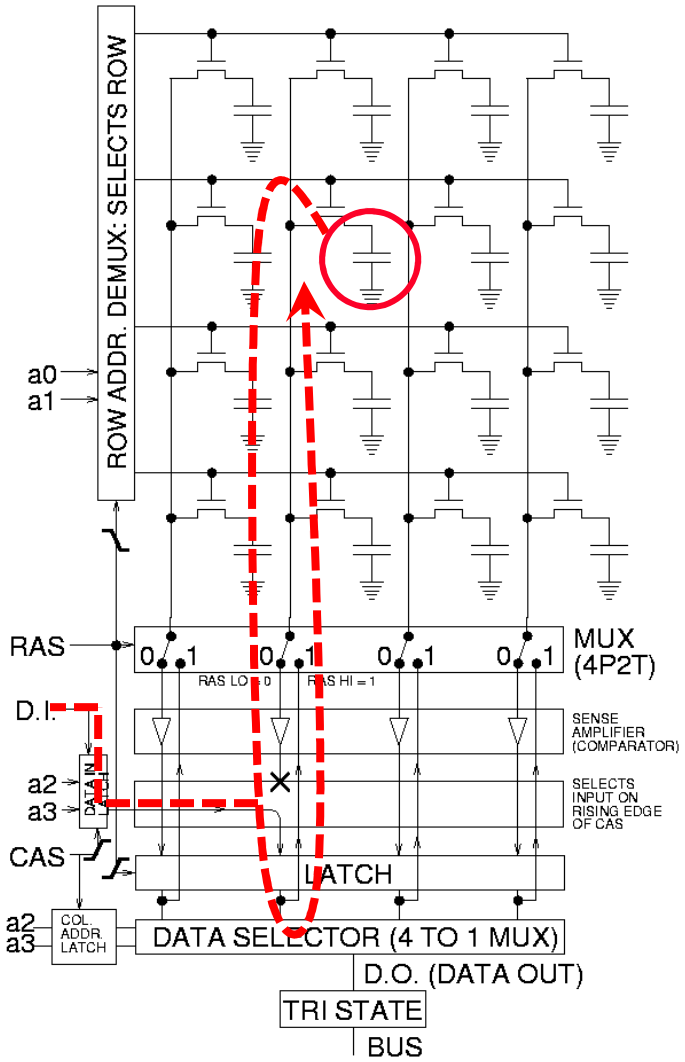
- One transistor/ capacitor per bit
- Multiplexed Address Bus
  - RAS = Row Address Select
  - CAS = Column Address Select
- Due to leakage current, capacitor discharges in a few ms
  - DRAM memory needs to be continuously rewritten. This process is called *refresh*
- Data is always read/written an *entire* row at a time

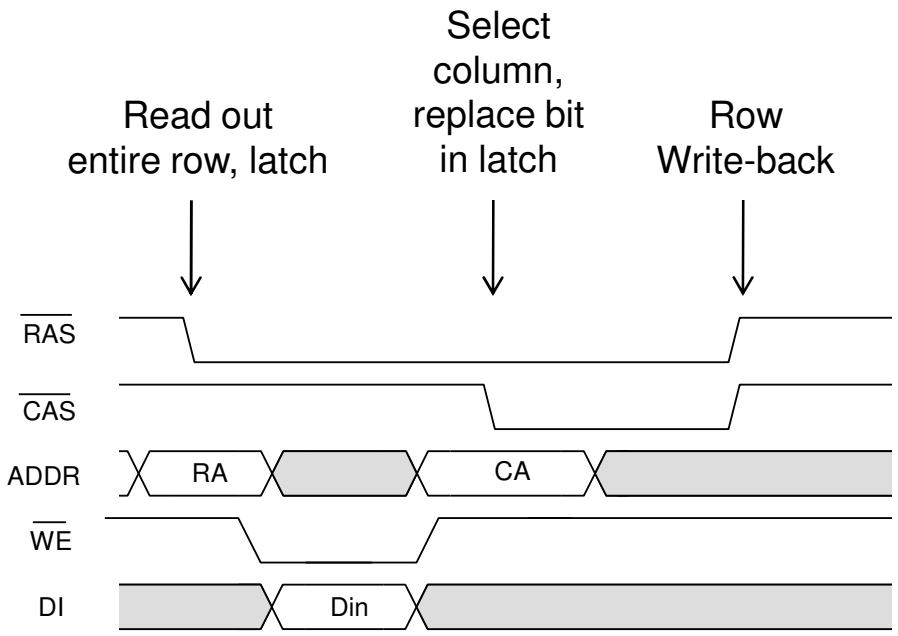
Dynamic RAM Architecture - Read





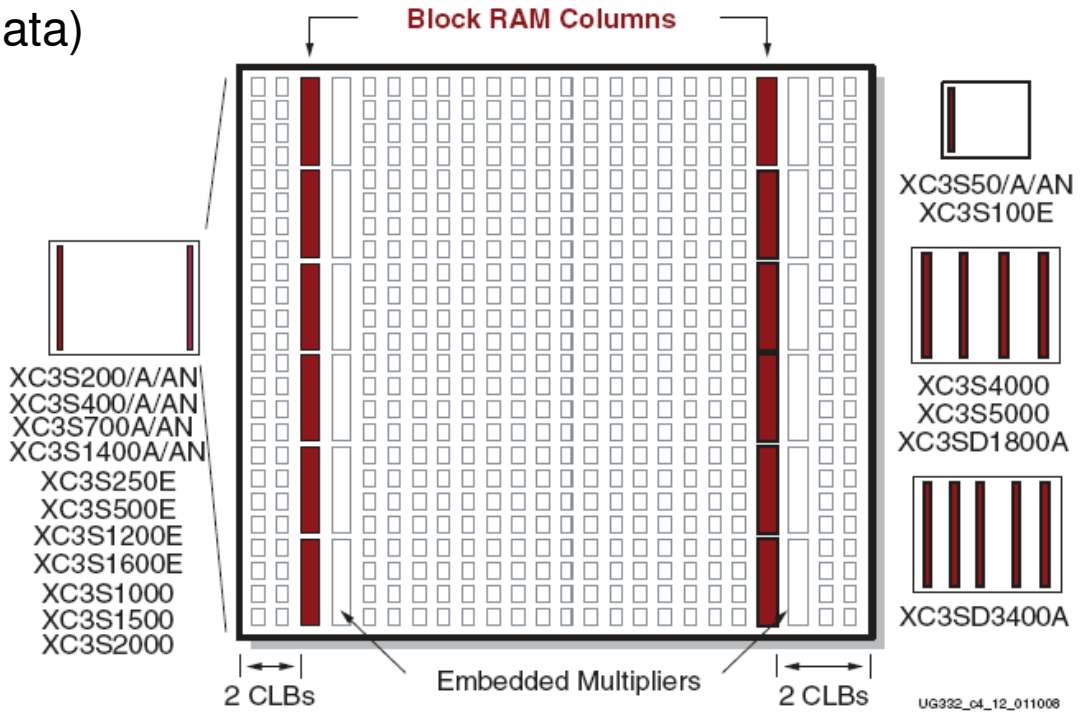
Dynamic RAM Architecture - Write



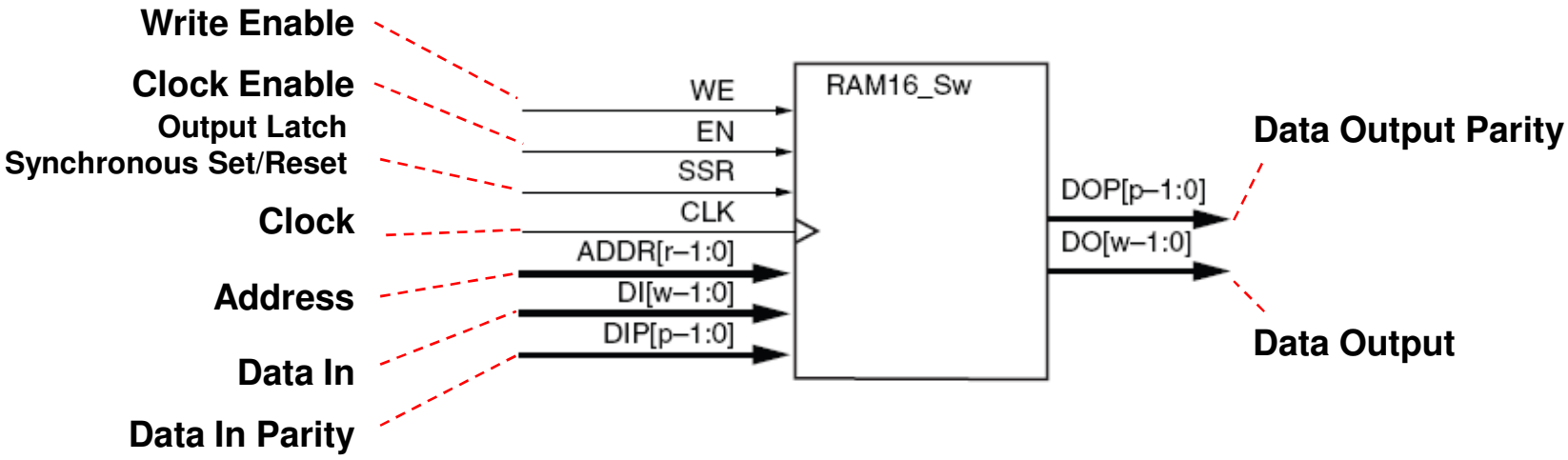


SRAM in FPGA

- In Spartan 3ES500, 20 BlockRAM modules of 18Kbit each
- 18Kbit = (16 data + 2 parity) Kbit
- Single or Dual Port
  - a 'Port' contains a data-input port, a data-output port, an address bus, and a read/write command signal
- Reconfigurable to multiple data/address combinations
  - 16K (address) x1 (data)
  - 8Kx2
  - 4Kx4
  - ...
  - 512x36



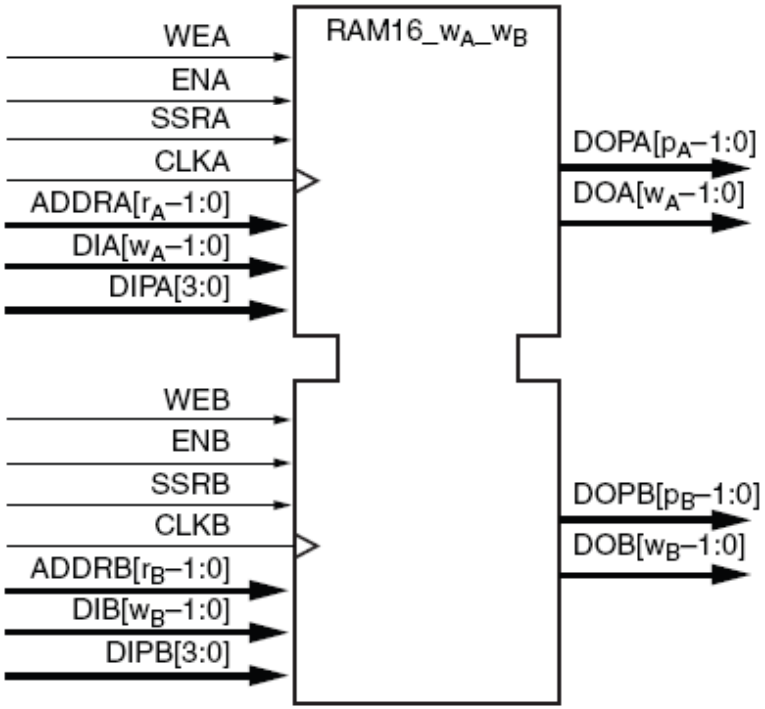
Single Port Configuration



EN	WE	CLK	Function
0	X	X	Block RAM disabled. No operation.
1	0	↑	Block RAM enabled but no write operation.
1	1	↑	As appropriate for the block RAM data organization, write data from the DI and DIP input ports to the currently addressed RAM location.

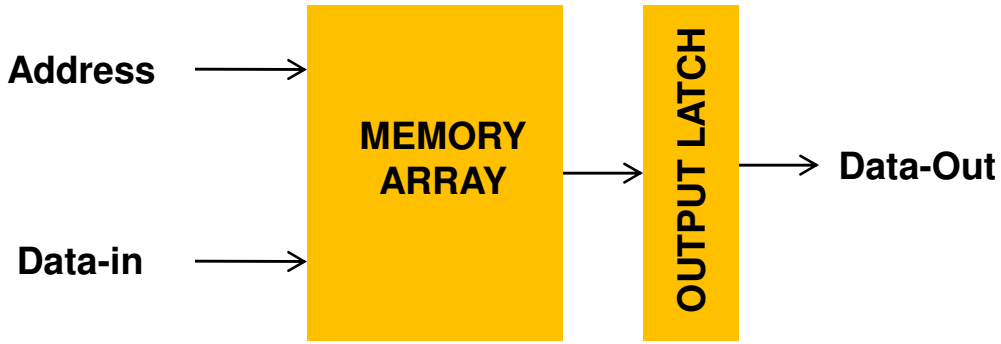
Dual Port Configuration

- One memory array with dual write/read ports: supports two writes, two reads per clock cycle
- Dual port does not offer priority resolution
  - After write collision, memory location becomes undefined
- Each port can be configured separately to desired (Adr x Data) width



Read/write ordering on BlockRAM

- If we *write* data, what data appears on the *data-output port*?
- Three write modes: WRITE_FIRST, READ_FIRST, NO_CHANGE
  - WRITE_FIRST: `Data-out = Data-in; Mem[Addr] = Data-in`
  - READ_FIRST: `Data-out = Mem[Addr]; Mem[Addr] = Data-in`
  - NO_CHANGE: `Mem[Addr] = Data-in`



Inferring RAM in Verilog

```
module v_rams_01 (clk, en, we, addr, di, do);
    input  clk;
    input  we;
    input  en;
    input  [5:0] addr;
    input  [15:0] di;
    output [15:0] do;
    reg    [15:0] RAM [63:0];  // 16-bit, 64 locations
    reg    [15:0] do;

    always @(posedge clk)
    begin
        if (en)
        begin
            if (we)
              RAM[addr]<=di;
            do <= RAM[addr];
        end
    end

endmodule
```

Read First Mode
(non-blocking assignment)

Always verify the feedback from the tools

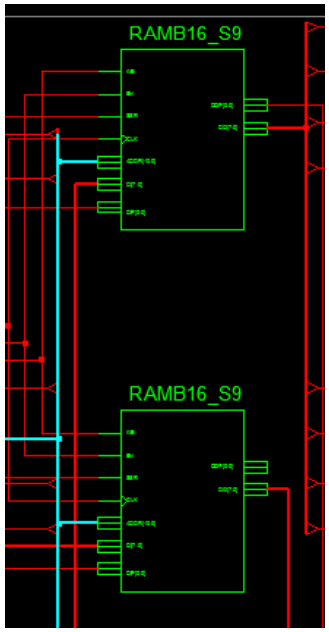
- Verify your assumptions

```
=========================================================================
*                        Advanced HDL Synthesis                         *
=========================================================================

Loading device for application Rf_Device from file '5vlx30.nph' in environment
C:\Xilinx91i.
INFO:Xst:2691 - Unit <v_rams_01> : The RAM <Mram_RAM> will be implemented as a BLOCK
RAM, absorbing the following register(s): <do>.
    ---------------------------------------------------------------------
    | ram_type           | Block                                        |          |
    ---------------------------------------------------------------------
    | Port A                                                            |          |
    |     aspect ratio   | 64-word x 16-bit                             |          |
    |     mode           | read-first                                   |          |
    |     clkA           | connected to signal <clk>                    | rise     |
    |     enA            | connected to signal <en>                     | high     |
    |     weA            | connected to signal <we>                     | high     |
    |     addrA          | connected to signal <addr>                   |          |
    |     diA            | connected to signal <di>                     |          |
    |     doA            | connected to signal <do>                     |          |
    ---------------------------------------------------------------------

=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# RAMs                                                : 1
  64x16-bit single-port block RAM                     : 1
```

Inferring RAM in Verilog

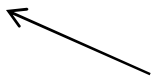
```
module v_rams_01 (clk, en, we, addr, di, do);
    input  clk;
    input  we;
    input  en;
    input  [10:0] addr;
    input  [15:0] di;
    output [15:0] do;
    reg    [15:0] RAM [2047:0];  // 16-bit, 2048 locations
    reg    [15:0] do;

    always @(posedge clk)
    begin
        if (en)
        begin
            if (we)
              RAM[addr]<=di;
            do <= RAM[addr];
        end
    end

endmodule
```

When a single BlockRAM is too small, tools will allocate multiple BlockRAM



Inferring RAM in Verilog

```
module v_rams_01 (clk, en, we, addr, di, do);
    input  clk;
    input  we;
    input  en;
    input  [5:0] addr;
    input  [15:0] di;
    output [15:0] do;
    reg    [15:0] RAM [63:0];  // 16-bit, 64 locations
    reg    [15:0] do;

    always @(posedge clk)
    begin
        if (en)
        begin
            if (we)
              RAM[addr]<= di;
            do <= di;
        end
    end

endmodule
```



Write First Mode

Always verify the feedback from the tools

- Verify your assumptions

```
=========================================================================
*                        Advanced HDL Synthesis                         *
=========================================================================

Loading device for application Rf_Device from file '3s500e.nph' in environment
C:\Xilinx91i.
INFO:Xst:2691 - Unit <v_rams_02a> : The RAM <Mram_RAM> will be implemented as a BLOCK
RAM, absorbing the following register(s): <do>.
    ---------------------------------------------------------------------
    | ram_type            | Block                                 |           |
    ---------------------------------------------------------------------
    | Port A                                                              |
    |     aspect ratio    | 64-word x 16-bit                      |           |
    |     mode            | write-first                           |           |
    |     clkA            | connected to signal <clk>             | rise      |
    |     enA             | connected to signal <en>              | high      |
    |     weA             | connected to signal <we>              | high      |
    |     addrA           | connected to signal <addr>            |           |
    |     diA             | connected to signal <di>              |           |
    |     doA             | connected to signal <do>              |           |
    ---------------------------------------------------------------------
```

Verify your assumptions - typical mistake

```
module v_rams_01 (clk, en, we, addr, di, do);
    input  clk;
    input  we;
    input  en;
    input  [5:0] addr;
    input  [15:0] di;
    output [15:0] do;
    reg    [15:0] RAM [127:0];  // 16-bit, 128 locations
    reg    [15:0] do;

    always @(posedge clk)
    begin
        if (en)
        begin
            if (we)
              RAM[addr]<= di;
            do <= di;
        end
    end

endmodule
```

What is the problem with this code ?

Verify your assumptions - typical mistake

```
module v_rams_01 (clk, en, we, addr, di, do);
    input  clk;
    input  we;
    input  en;
    input  [5:0] addr;
    input  [15:0] di;
    output [15:0] do;
    reg    [15:0] RAM [127:0];  // 16-bit, 128 locations
    reg    [15:0] do;

    always @(posedge clk)
    begin
        if (en)
        begin
            if (we)
              RAM[addr]<= di;
            do <= di;
        end
    end

endmodule
```

What is the problem with this code ?

128 locations, but only 6 address bits

Always verify the feedback from the tools

- Warnings *may* (not: *will*) suggest cause

```
=========================================================================
*                            HDL Analysis                               *
=========================================================================
Analyzing top module <v_rams_02a>.
INFO:Xst:1607 - Contents of array <RAM> may be accessed with an index that
does not cover the full array size.
INFO:Xst:1607 - Contents of array <RAM> may be accessed with an index that
does not cover the full array size.
Module <v_rams_02a> is correct for synthesis.

. . .

    ---------------------------------------------------------------------
    | ram_type            | Block                                 |           |
    ---------------------------------------------------------------------
    | Port A                                                              |
    |     aspect ratio    | 64-word x 16-bit                      |           |
    |     mode            | write-first                           |           |
    |     clkA            | connected to signal <clk>             | rise      |
    |     enA             | connected to signal <en>              | high      |
    |     weA             | connected to signal <we>              | high      |
    |     addrA           | connected to signal <addr>            |           |
    |     diA             | connected to signal <di>              |           |
    |     doA             | connected to signal <do>              |           |
    ---------------------------------------------------------------------
```

RAM with asynchronous read port

```
module v_rams_04 (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];

    always @(posedge clk) begin
        if (we)
            ram[a] <= di;
    end

    assign do = ram[a];

endmodule
```

← **Asynchronous Read**

What would the synthesis tools do?
Remember: BlockRAM is a synchronous module. Address is only captured on clock edge.

EN	WE	CLK	Function
0	X	X	Block RAM disabled. No operation.
1	0	↑	Block RAM enabled but no write operation.
1	1	↑	As appropriate for the block RAM data organization, write data from the DI and DIP input ports to the currently addressed RAM location.

Asynchronous read not possible on BRAM

```
=========================================================================
*                        Advanced HDL Synthesis                         *
=========================================================================

Loading device for application Rf_Device from file '3s500e.nph' in
environment C:\Xilinx91i.
INFO:Xst:2664 - HDL ADVISOR - Unit <v_rams_04> : The RAM <Mram_ram> will be
implemented on LUTs either because you have described an asynchronous read
or because of currently unsupported block RAM features. If you have
described an asynchronous read, making it synchronous would allow you to
take advantage of available block RAM resources, for optimized device usage
and improved timings. Please refer to your documentation for coding
guidelines.
    ------------------------------------------------------------------------
    | ram_type           | Distributed                           |          |
    ------------------------------------------------------------------------
    | Port A                                                               |
    |     aspect ratio   | 64-word x 16-bit                      |          |
    |     clkA           | connected to signal <clk>             | rise     |
    |     weA            | connected to signal <we>              | high     |
    |     addrA          | connected to signal <a>               |          |
    |     diA            | connected to signal <di>              |          |
    |     doA            | connected to signal <do>              |          |
    ------------------------------------------------------------------------
```

64-word x 16-bit 'Distributed RAM' occupies 82 LUTs …

Dual-port RAM

```
module v_rams_13 (clk, en, we, addra, addrb, di, doa, dob);
    input  clk;
    input  en;
    input  we;
    input  [5:0] addra;
    input  [5:0] addrb;
    input  [15:0] di;
    output [15:0] doa;
    output [15:0] dob;
    reg    [15:0] ram [63:0];
    reg    [5:0] read_addra;
    reg    [5:0] read_addrb;

    always @(posedge clk) begin
        if (en)
        begin
            if (we)
                ram[addra] <= di;
            read_addra <= addra;
            read_addrb <= addrb;
        end
    end

    assign doa = ram[read_addra];
    assign dob = ram[read_addrb];
endmodule
```

1) One write-port, two read-port
2) write-first mode

Synchronous Read
(because address is clocked)

Instantiating RAM Library Blocks

- Previous examples illustrate how a RAM can be *inferred*
- We can also *instantiate* a RAM as a library module
- The primitive BlockRAM modules in Spartan 3E S500 are of the form RAMB16_S*n*, where *n* is a number that selects the configuration

Component	Data Cells	
	Depth	Width
RAMB16_S1	16384	1
RAMB16_S2	8192	2
RAMB16_S4	4096	4
RAMB16_S9	2048	8
RAMB16_S18	1024	16
RAMB16_S36	512	32

Example Instantiation

Selects type (address/data configuration)

↓

```
RAMB16_S9 RAMB16_S9_inst4 (
    .DO(ram_data_out4),        // 8-bit Data Output
    .DOP(parity4),
    .ADDR(ram_address[10:0]), // 11-bit Address Input
    .CLK(clk),                 // Clock
    .DI(out_port),             // 8-bit Data Input
    .DIP (1'b0),
    .EN(cs_ram4),              // RAM Enable Input
    .SSR(1'b0),                // Synchronous Set/Reset Input
    .WE(write_to_ram)       ); // Write Enable Input

defparam RAMB16_S9_inst4.WRITE_MODE = "WRITE_FIRST";
defparam RAMB16_S9_inst4.INIT0 = 256'h000000000000000000000000\
                                 000000000000000000000000\
                                 0000000000000000;
```

↑

Selects additional synthesis parameters: mode, initial values, …

Templates and parameters are defined as part of the 'HDL Libraries Guide'

If you can infer RAM, don't bother instantiating a specific module

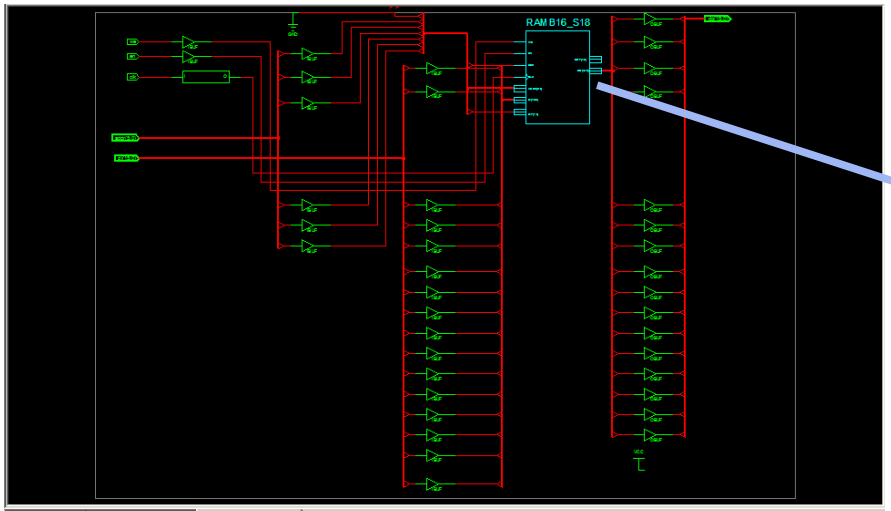
Initialized RAM

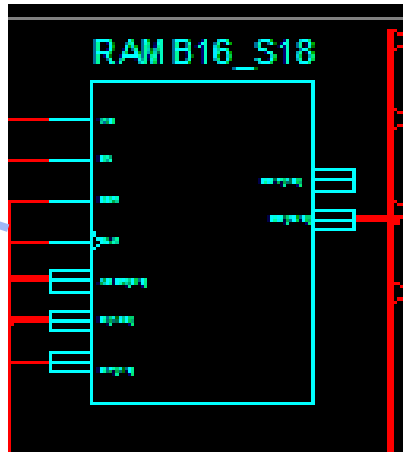
- An initialized RAM can be used as a lookup table or as a ROM
- In FPGA, bitstream specifies the initial contents of the BRAM
- In Verilog, we may specify the initial contents of the RAM in diffferent ways
  - using *initial* block (inference) or *synthesis constraints* (instantiation)
  - using $readmemb or $readmemh system calls
- The tools will then ensure that the RAM will be initialized to the specified state after configuration

Inferring Initialized RAM in Verilog

```
module v_rams_01 (clk, en, we, addr, di, do);
    input  clk;
    input  we;
    input  en;
    input  [5:0] addr;
    input  [15:0] di;
    output [15:0] do;
    reg    [15:0] RAM [63:0];  // 16-bit, 64 locations
    reg    [15:0] do;
    reg    [6:0] i;

    initial
      for (i=0; i<64; i++)
        RAM[i] = i;

    always @(posedge clk)
    begin
        if (en)
        begin
            if (we)
              RAM[addr]<=di;
            do <= RAM[addr];
        end
    end

endmodule
```

Result of Inferred Initialized RAM





rams_01.v | Synthesis Report | v_rams_01.ngc

Design Objects of v_rams_01

Signals
addr(5:0)
clk
clk_BUFGP
di(15:0)
di_15_IBUF,di_14_IBUF,di_13_IBUF,di_...

Properties of Instance Mram_RAM

Name	Value
INIT_00	000F000E000D000C000B000A000900080007000600050004000300020001 0000
INIT_01	001F001E001D001C001B001A001900180017001600150014001300120011 0010
INIT_02	002F002E002D002C002B002A002900280027002600250024002300220021 0020
INIT_03	003F003E003D003C003B003A003900380037003600350034003300320031 0030
InstName	Mram_RAM

Find in Files

Properties of Instance
Mram_RAM

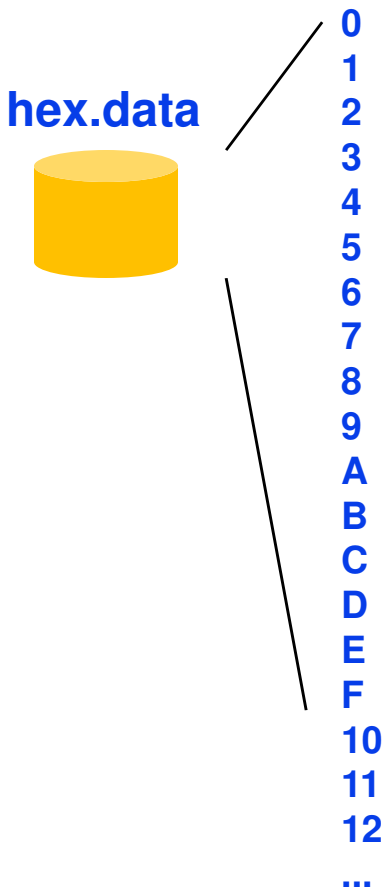
Name	Value
INIT_00	000F000E000D000C000B000A000900080007000600050004000300020001 0000
INIT_01	001F001E001D001C001B001A001900180017001600150014001300120011 0010
INIT_02	002F002E002D002C002B002A002900280027002600250024002300220021 0020
INIT_03	003F003E003D003C003B003A003900380037003600350034003300320031 0030
InstName	Mram_RAM
Type	ramb16_s18
WRITE_MODE	READ_FIRST

Synthesis attributes for BlockRAM Library Module

$readmemh

```
module v_rams_01 (clk, en, we, addr, di, do);
    input  clk;
    input  we;
    input  en;
    input  [5:0] addr;
    input  [15:0] di;
    output [15:0] do;
    reg    [15:0] RAM [63:0];
    reg    [15:0] do;

    initial
      $readmemh("hex.data", RAM, 0, 63);

    always @(posedge clk)
    begin
        if (en)
        begin
            if (we)
              RAM[addr]<=di;
            do <= RAM[addr];
        end
    end

endmodule
```

Filename | Array | Initial Index | Final Index



Synthesis tools will read hex file to initialize memory

Similar command ($readmemb) for binary radix

Summary

- Design flow for Dapath and Memory Elements
  - Inference vs Instantiation
  - Integrated synthesis and simulation
- Memory Elements
  - Register Files
  - Memory Arrays: SRAM and DRAM
  - Operation of SRAM and DRAM
  - SRAM in Spartan 3E FPGA
    - Configuration: single/dual port, read/write ordering
    - Infering SRAM with Verilog
    - Initializing SRAM, $readmemb and $readmemh