SOC Design

Design with SRAM

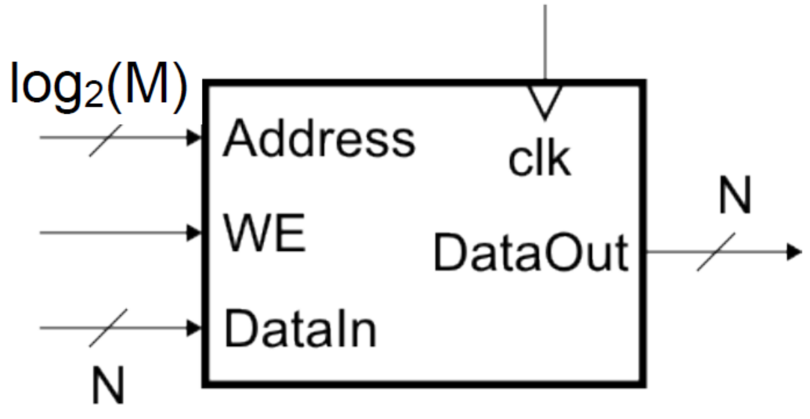
Jiin Lai

Topics

- FPGA Memory
- Memory Inference (FPGA, ASIC)
- Design Example to use SRAM (spiflash.v)

SRAM Usage

- data & program storage
- general purpose registers
- data buffering
- table lookups
- Combinational Logic implementation



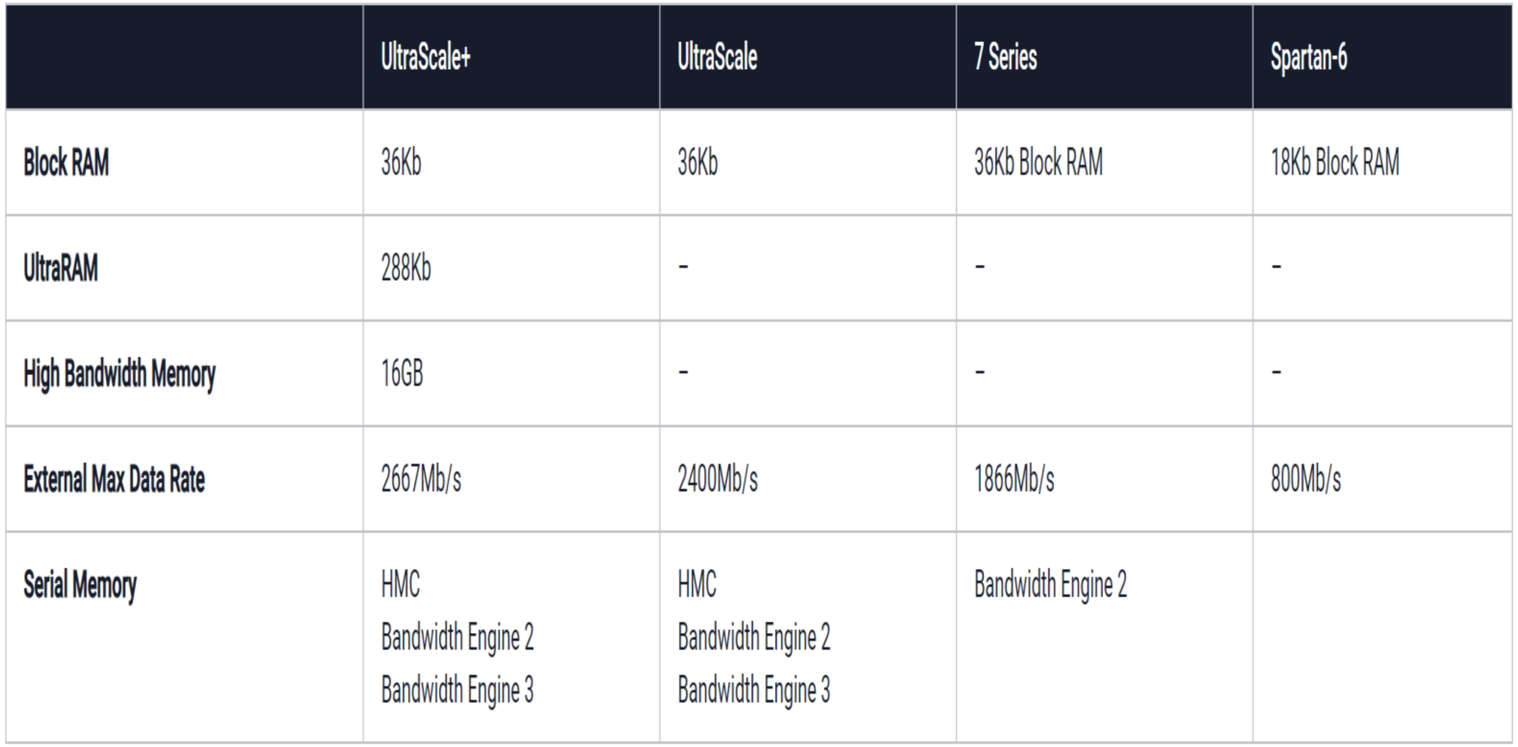
M X N memory:

Depth = M, Width = N.

M words of memory, each word N bits wide.

FPGA Memory

- CLB/LUT: Distributed RAM (64x1b or 32x2b)
- Block RAM - FIFO
- Ultra RAM



	UltraScale+	UltraScale	7 Series	Spartan-6
Block RAM	36Kb	36Kb	36Kb Block RAM	18Kb Block RAM
UltraRAM	288Kb	-	-	-
High Bandwidth Memory	16GB	-	-	-
External Max Data Rate	2667Mb/s	2400Mb/s	1866Mb/s	800Mb/s
Serial Memory	HMC Bandwidth Engine 2 Bandwidth Engine 3	HMC Bandwidth Engine 2 Bandwidth Engine 3	Bandwidth Engine 2	

CLB Resources

- 8 x 6-input LUT
- 16 x flip-flops
- 8-bit carry-chain CARRY8
- Wide-multiplexers combine LUTs to FMUX- 7, 8, 9 inputs, 55 inputs
- A LUT in SLICEM
  - A loop-up table
  - 64-bit distributed RAM
  - 32-bit shift registers

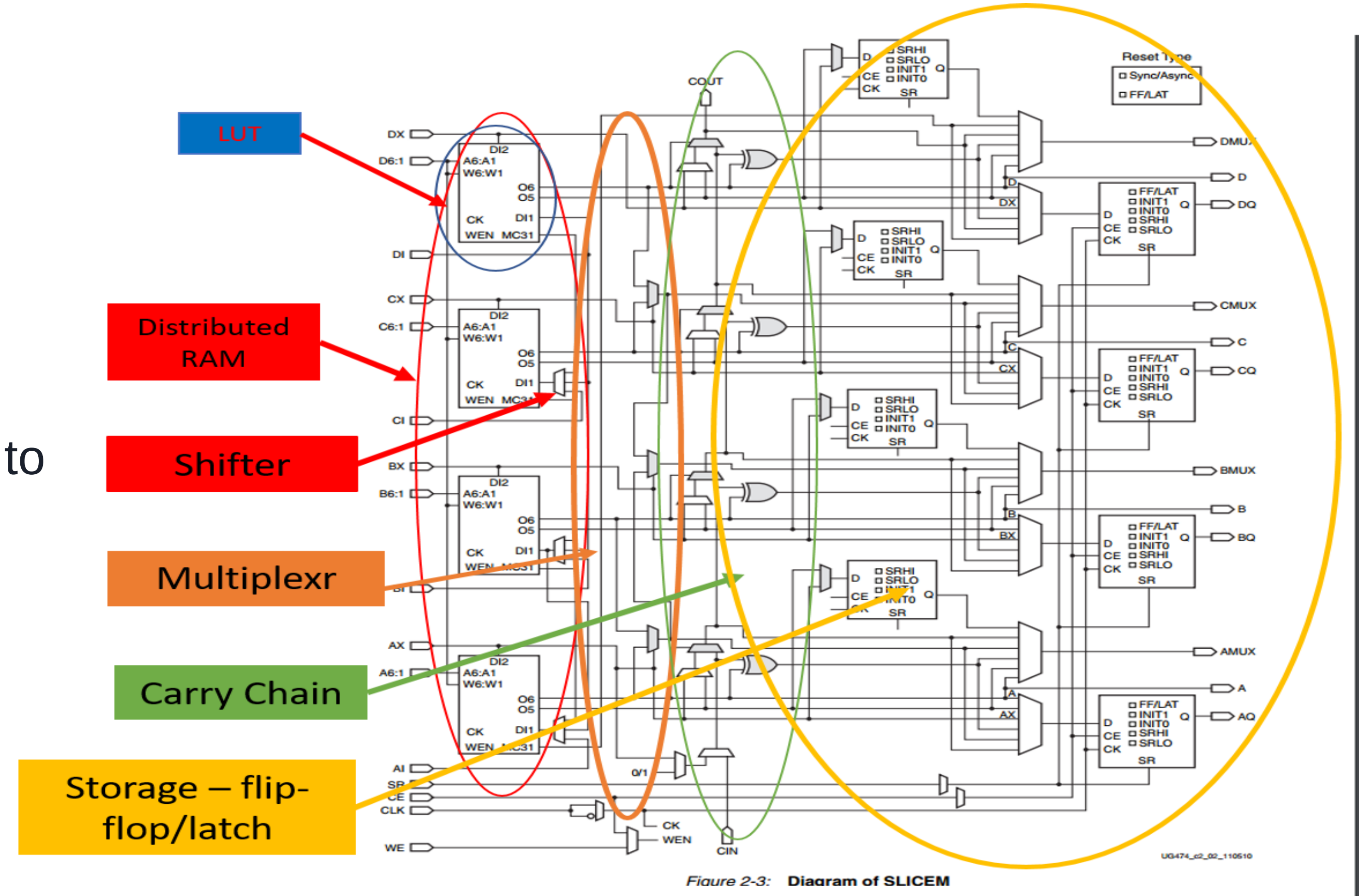
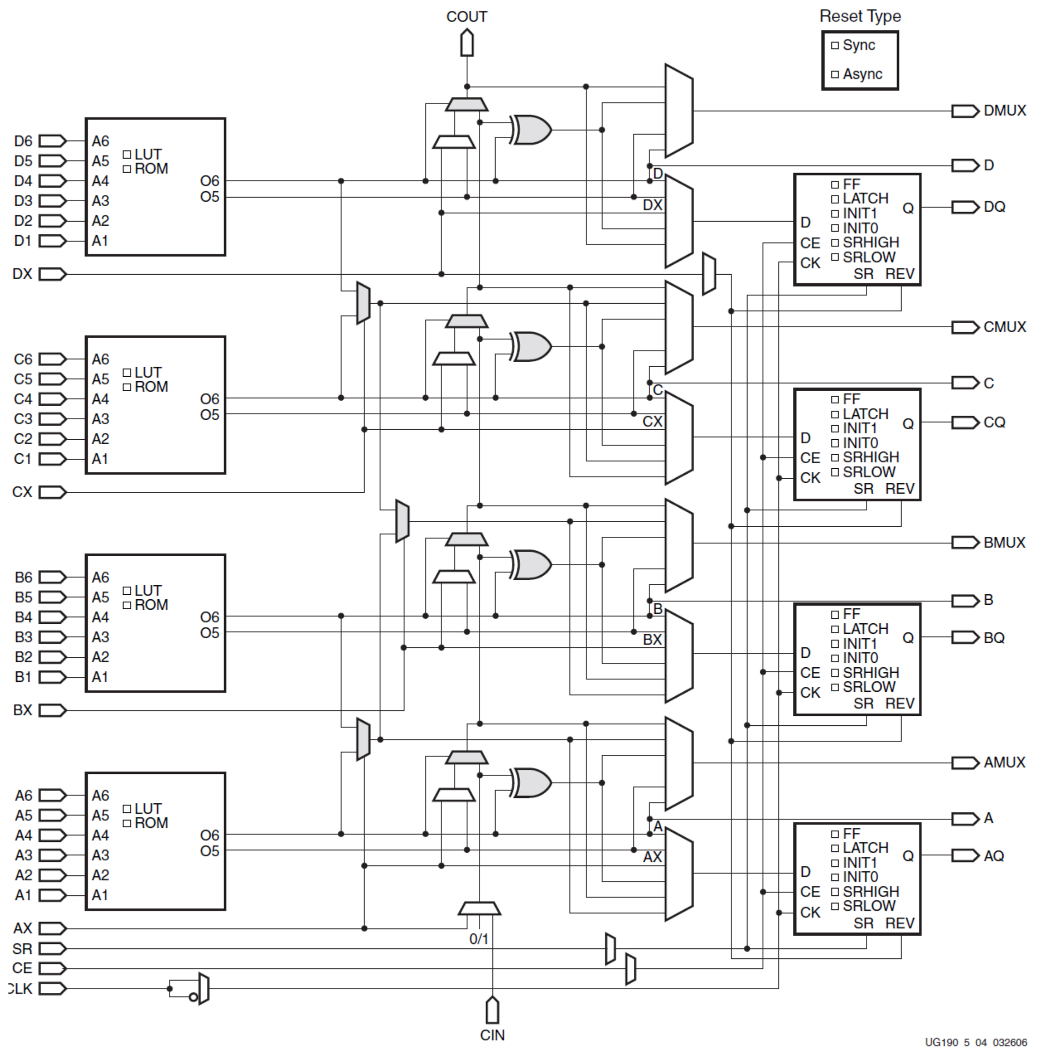


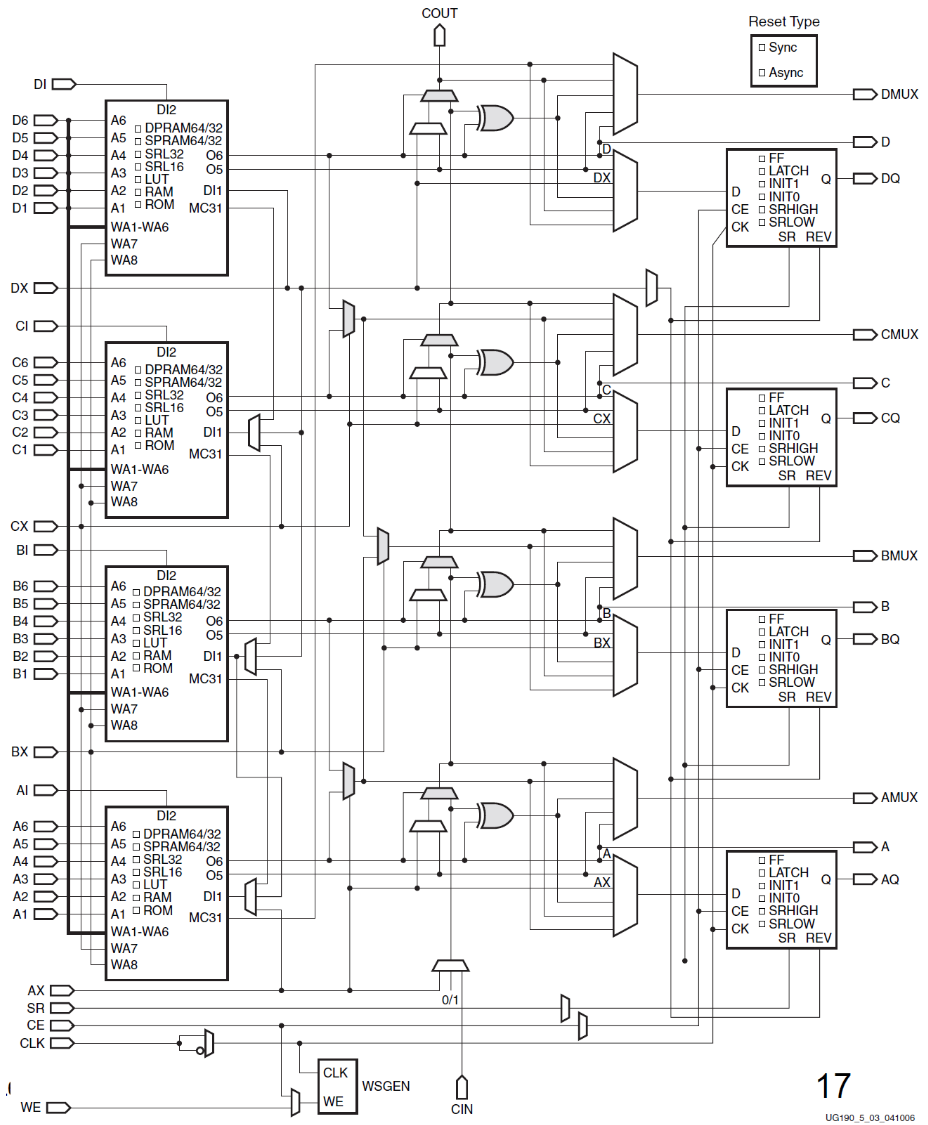
Figure 2-3: Diagram of SLICEM

CLB Slice	LUTs	Flip-Flops	Arithmetic and Carry Chains	Wide Multiplexers	Distributed RAM	Shift Registers
SLICEL	8	16	1	F7, F8, F9	N/A	N/A
SLICEM	8	16	1	F7, F8, F9	512 bits	256 bits

SLICEL



SLICEM



LUT as Distributed RAM, ROM

- One LUT - 64x1 or 32x2
- SliceM = 8 LUT => 512bit
- Combine multiple Slice-M

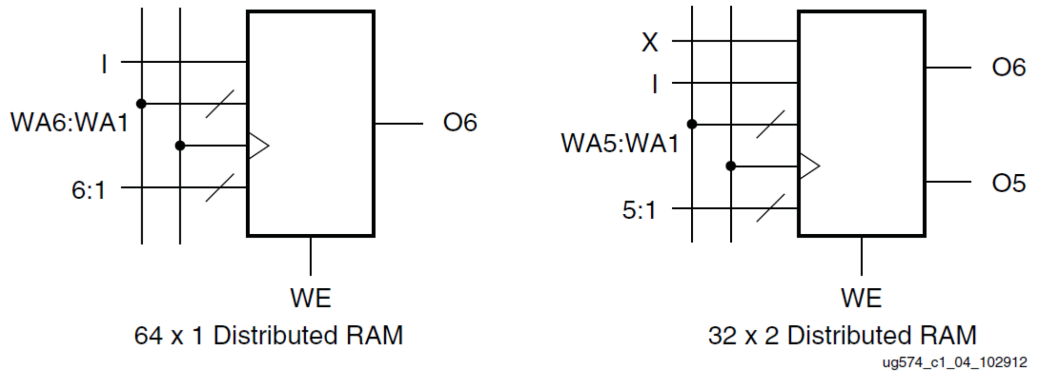


Figure 1-4: **Distributed RAM Using Look-Up Tables**

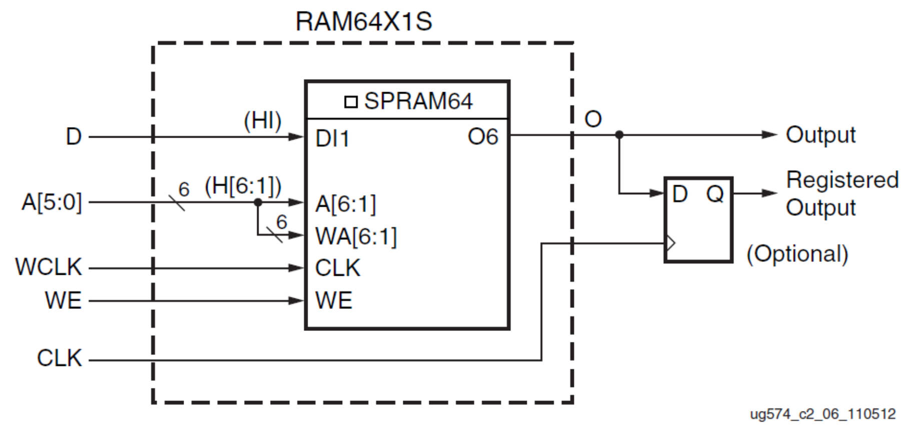
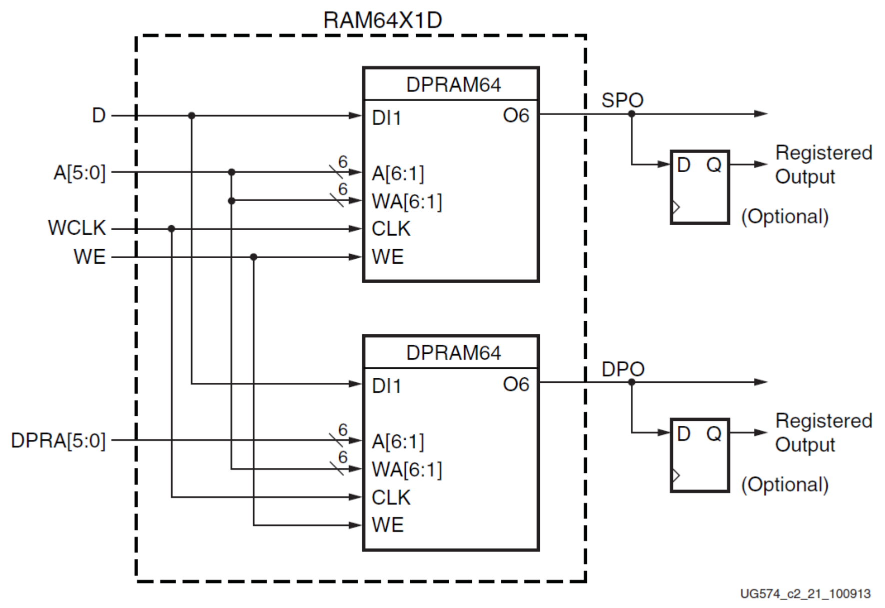


Figure 2-5: **Distributed RAM (RAM64X1S)**

Table 2-2: **ROM Configuration**

ROM	Number of LUTs
64 x 1	1
128 x 1	2
256 x 1	4
512 x 1	8



Distributed RAM Configurations

Distributed RAM Configurations (SLICEM): port * depth * width <= 512 bit

- Single-port 32 x (1 to 16)-bit RAM
- Dual-port 32 x (1 to 8)-bit RAM
- Quad-port 32 x (1 to 4)-bit RAM
- Simple dual-port 32 x (1 to 14)-bit RAM
- Single-port 64 x (1 to 8)-bit RAM
- Dual-port 64 x (1 to 4)-bit RAM
- Quad-port 64 x (1 to 2)-bit RAM
- Octal-port 64 x 1-bit RAM
- Simple dual-port 64 x (1 to 7)-bit RAM
- Single-port 128 x (1 to 4)-bit RAM
- Dual-port 128 x 2-bit RAM
- Quad-port 128 x 1-bit RAM
- Single-port 256 x (1 to 2)-bit RAM
- Dual-port 256 x 1-bit RAM
- Single-port 512 x 1-bit RAM

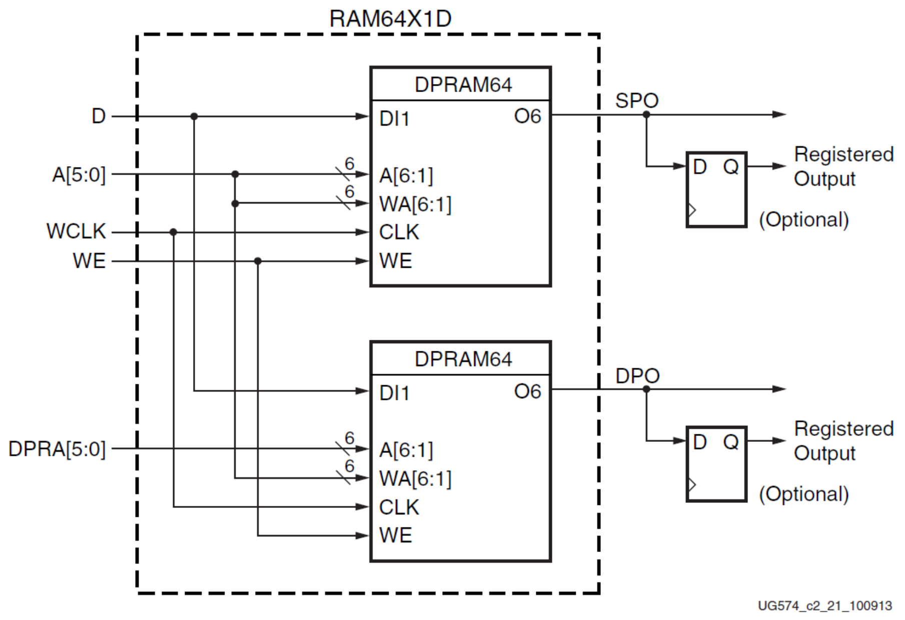


Figure 2-6: **64x1 Dual-Port Distributed RAM (RAM64X1D)**

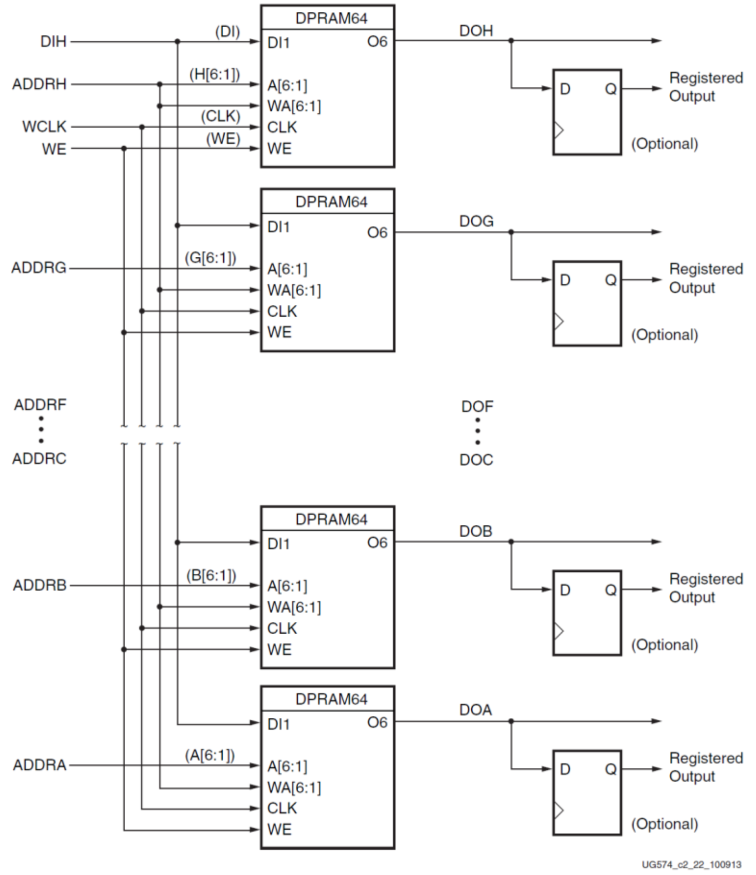
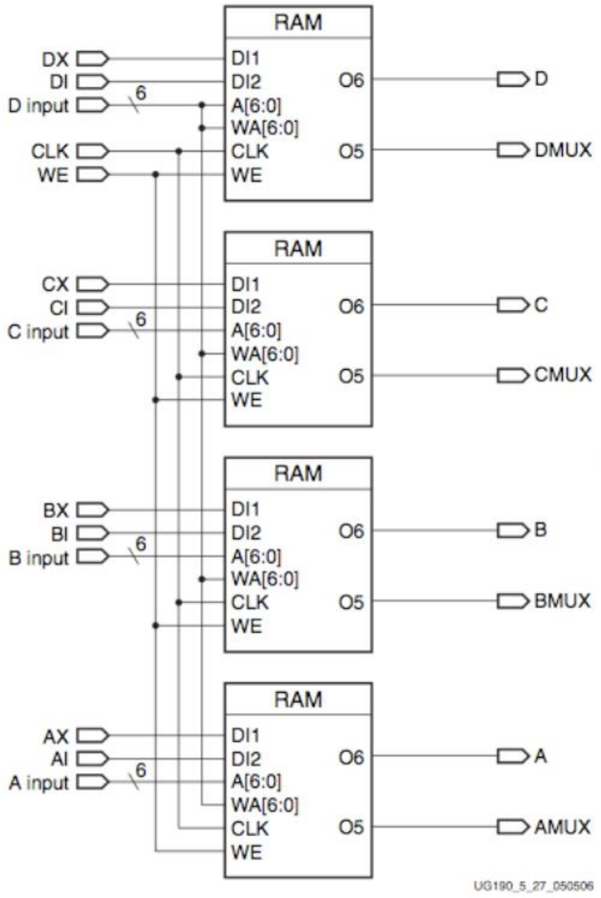
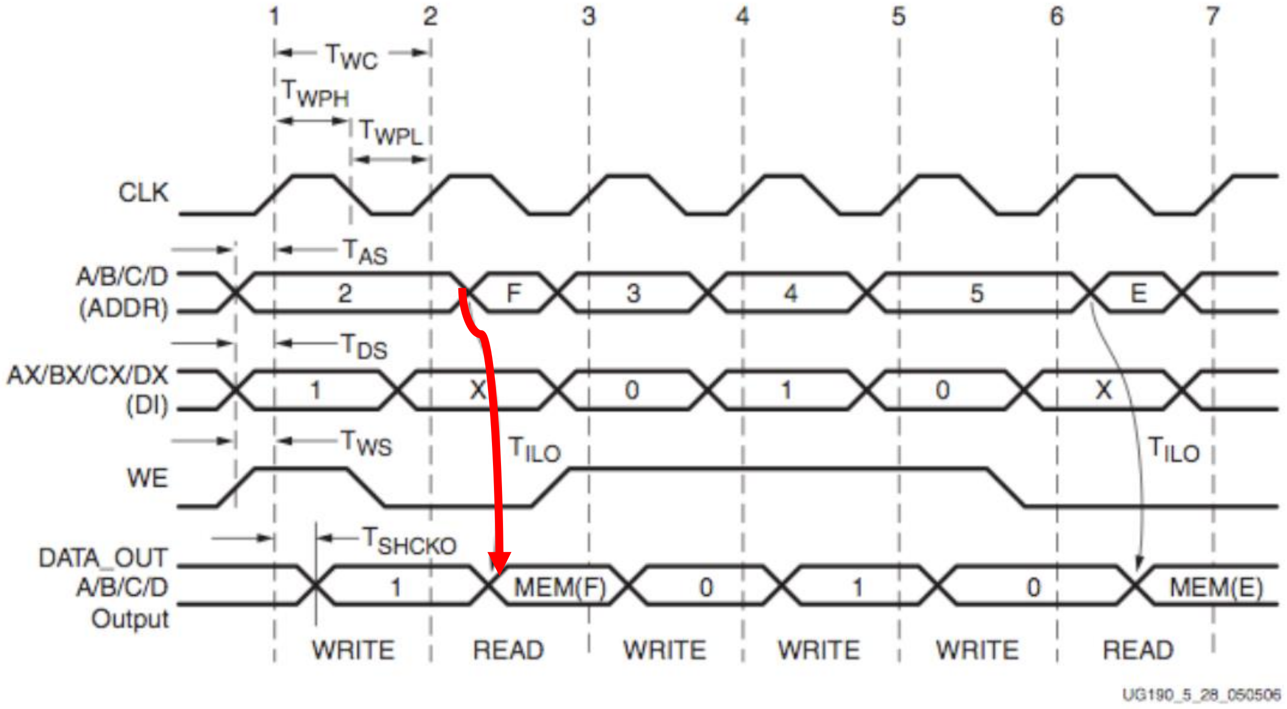


Figure 2-7: **64x1 Octal Port Distributed RAM**

Distributed RAM Access Timing



Synchronous write / Asynchronous read

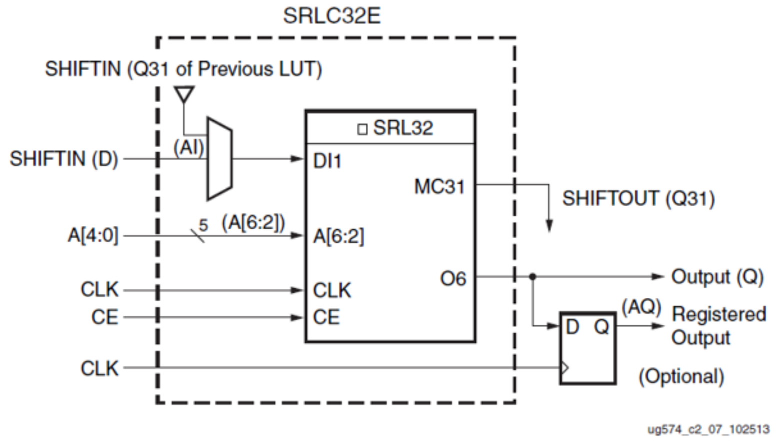


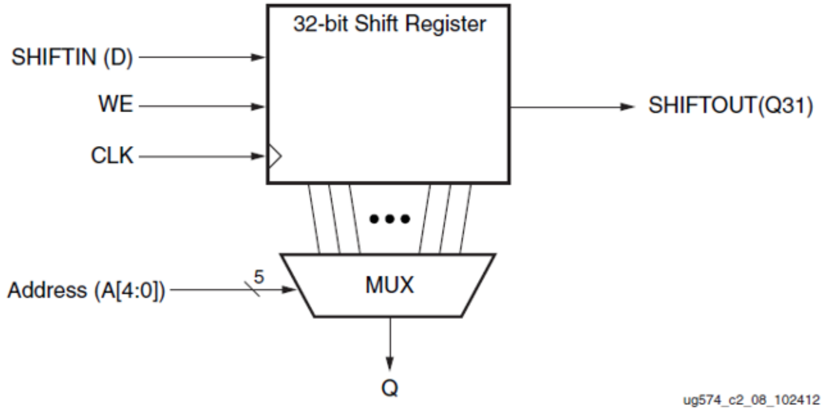
Shift Registers (SLICEM only)

- SLICEM = 256bit

Usages

- Synchronous FIFO
  - dataflow stream channel
- Delay or Latency compensation
  - balance latency difference for re-converged paths
- Content-Addressable Memory
- Cascade to form larger shift registers





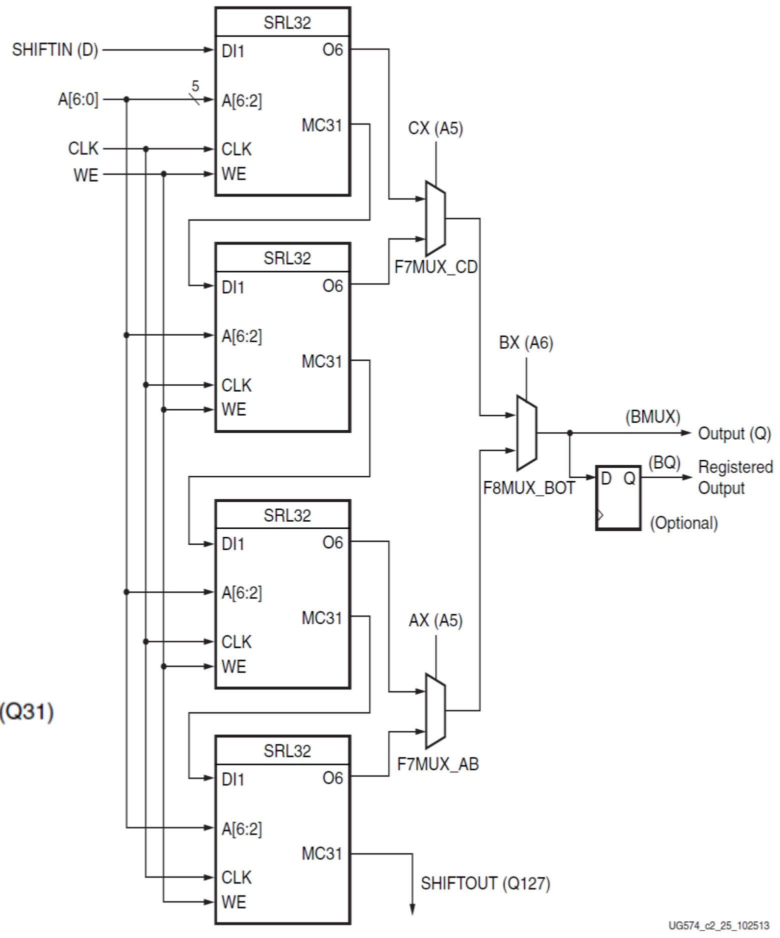


Figure 2-13: **128-bit Shift Register Configuration**

https://www.academia.edu/6839238/Implementation_and_Design_of_High_Speed_FPGA_based_Content_Addressable_Memory

Block RAM

- **36Kb storage area + 2 completely independent access ports (each 18kb)**
- 36Kb: Width: 36K x 1, 16K x 2, 8K x 4, 4K x 9, 2K x 18, 1K x 36, 512 x 72 (if simple Dual-port)
- 18Kb: Width: 16K x 1, 8K x 2, 4K x 4, 2K x 9, 1K x 18, 512 x 36 (used as SDP)
- Memory content can be initialized by configuration bitstream
- Optional Output Registers (better timing)
- **Write/Read are synchronous operation**
- Independent Read and Write Port Width Selection
- **Simple Dual-Port BlockRAM, True Dual port**
- Cascadable Block RAM (more depth)
- Byte-Wide Write Enable (for MCU)
- Error Correction (reliability)
- FIFO (built in sequencing, flag … save CLB resource)
- Read-during-write function

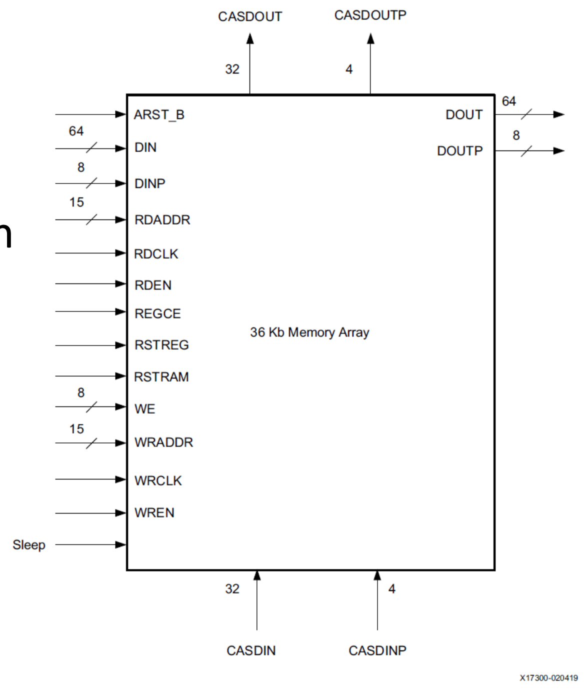


Figure 1-6: **RAMB36 Usage in a Simple Dual-Port Data Flow**

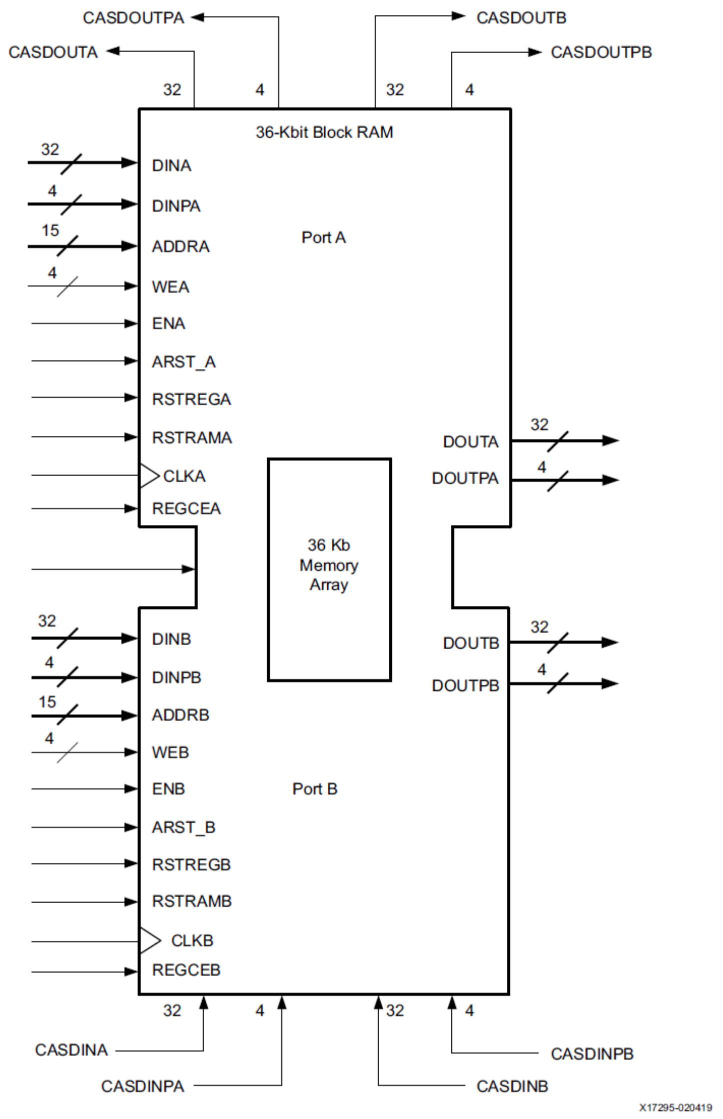
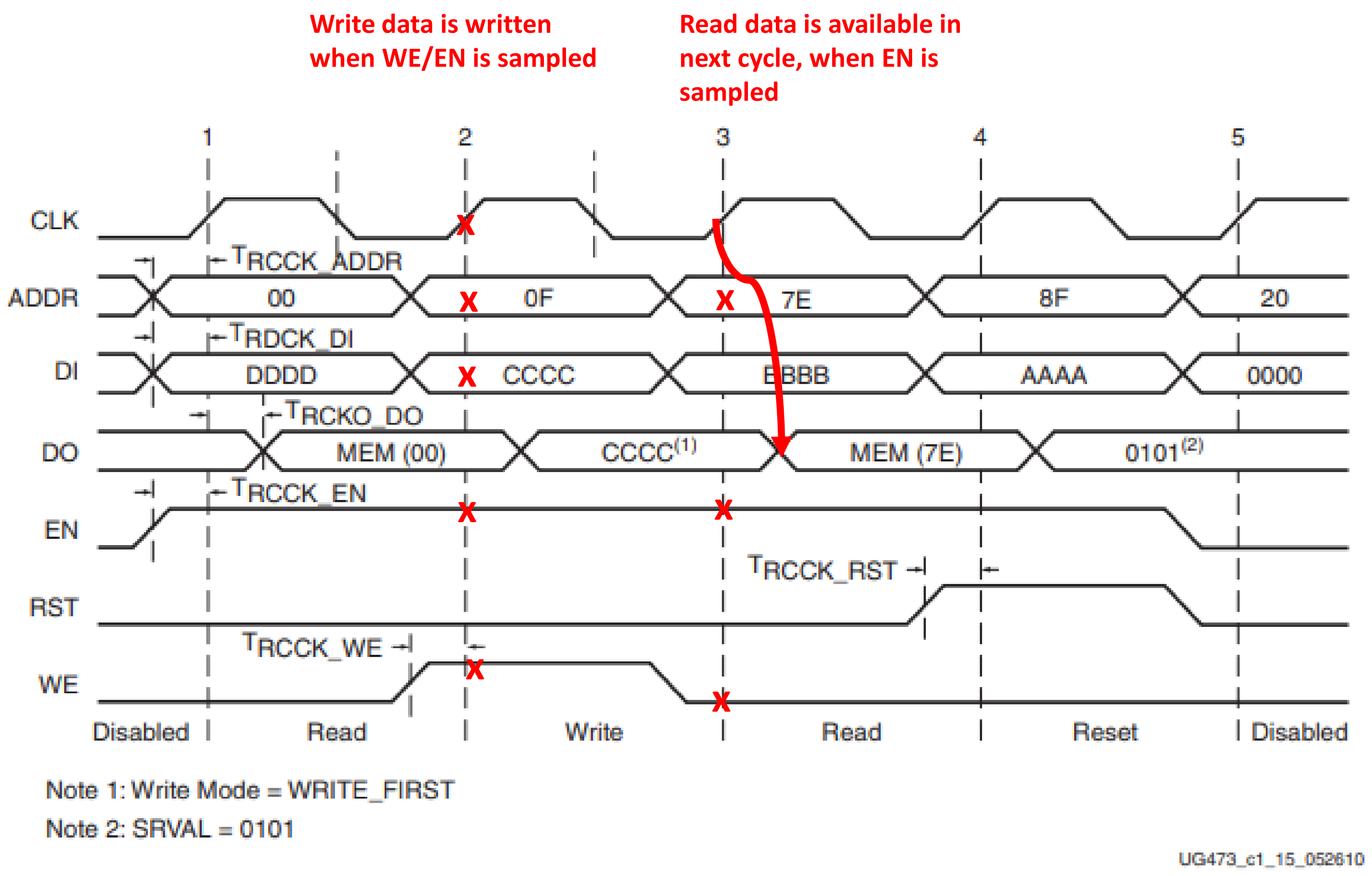


Figure 1-1: **RAMB36 Usage in a True Dual-Port Data Flow**

https://www.xilinx.com/support/documentation/user_guides/ug573-ultrascale-memory-resources.pdf

BlockSRAM Access Timing – Synchronous Read/Write



SRAM access has different modes, refer to https://docs.xilinx.com/r/en-US/am007-versal-memory/Read-Operation?tocId=VRYu0HURA1U147fufYDMNQ

Optional Output Registers

```
#pragma HLS bind_storage variable=xxx type=RAM_1P impl=bram latency=2
```

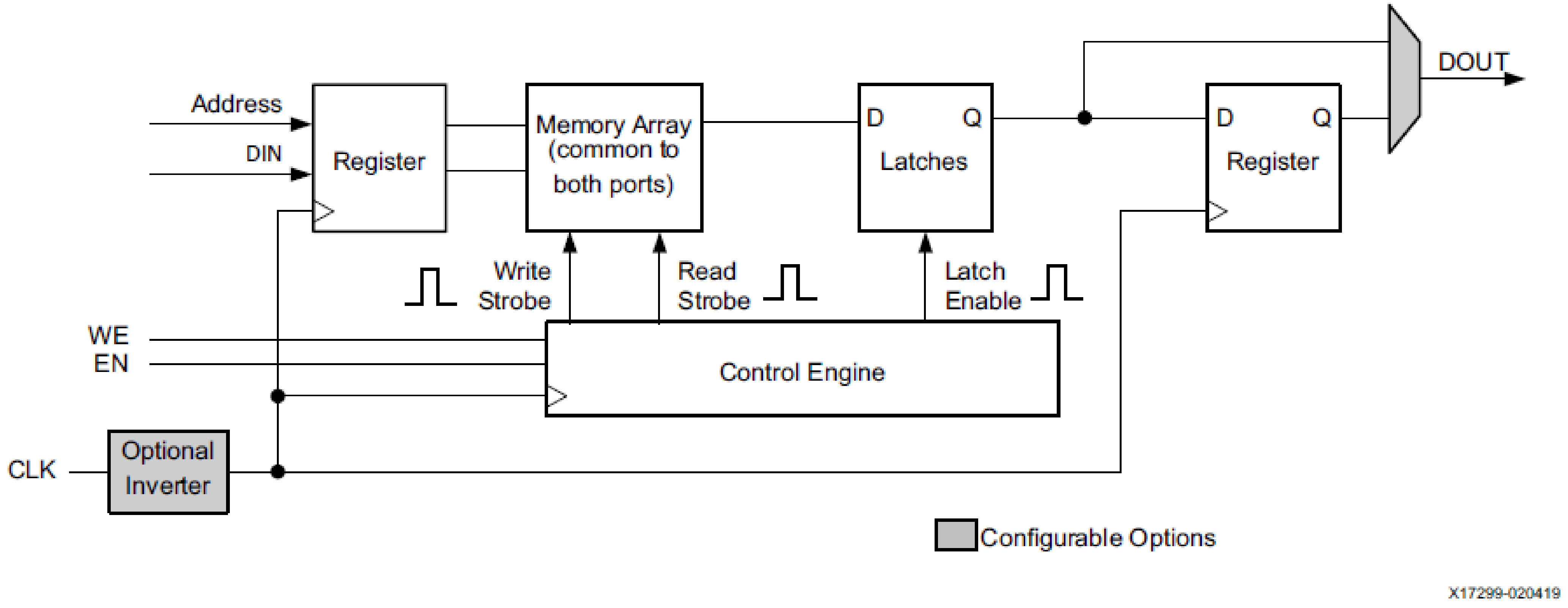


Figure 1-5: **Block RAM Logic Diagram (One Port Shown)**

Cascadable Block RAM

- Standard Data Out Cascade
  - Impact final clock-to-out performance
- Data Out cascade in Pipeline Mode
  - Last stage select register out
- Cascaded signals have dedicated interconnects within a block RAM column

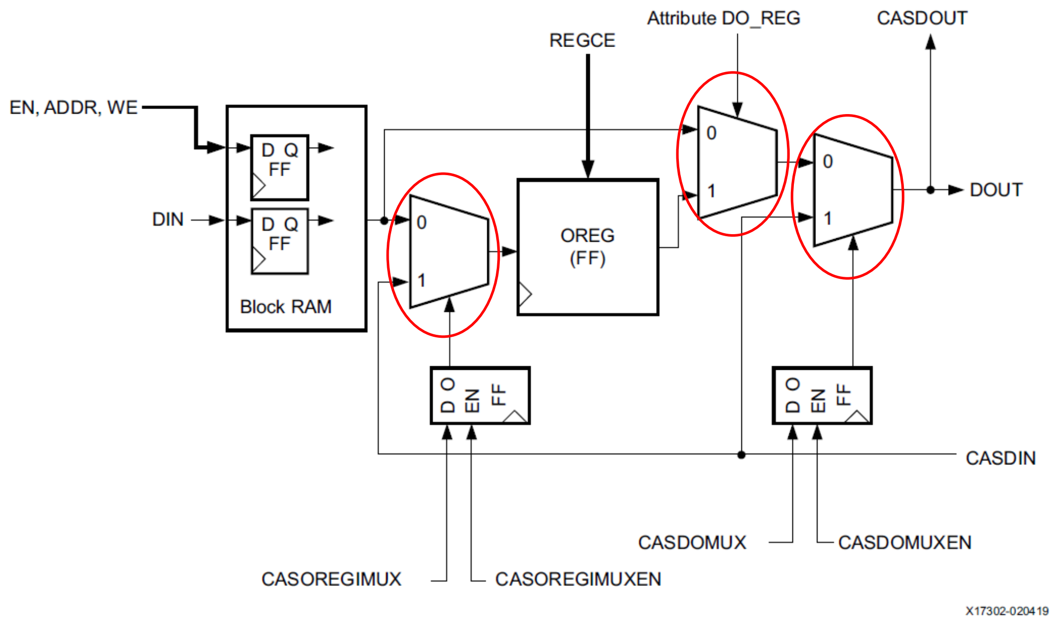


Figure 1-8: **Cascade Functional Diagram**

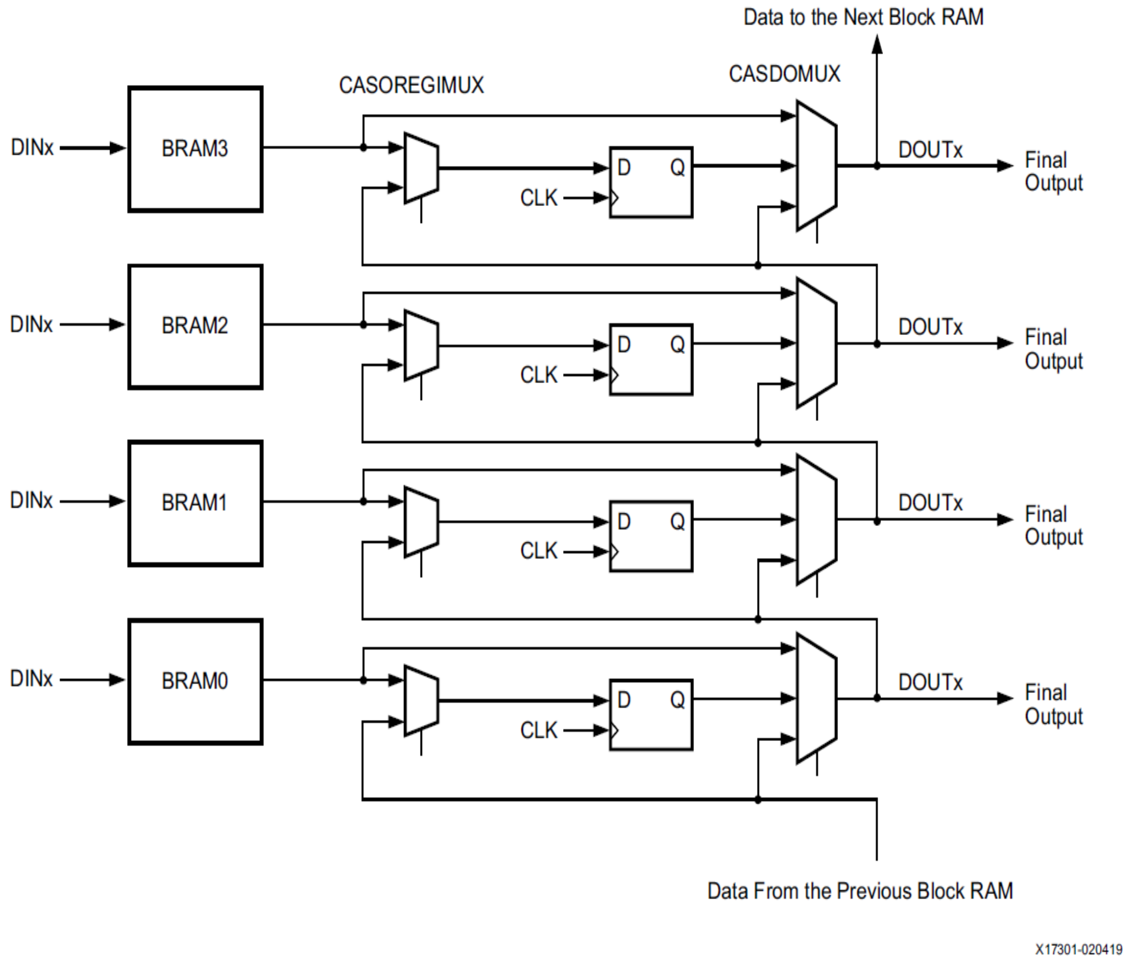


Figure 1-7: **High-level View of the Block RAM Cascade Architecture**

Error Correction Code- Single bit correction , Double-bit detection

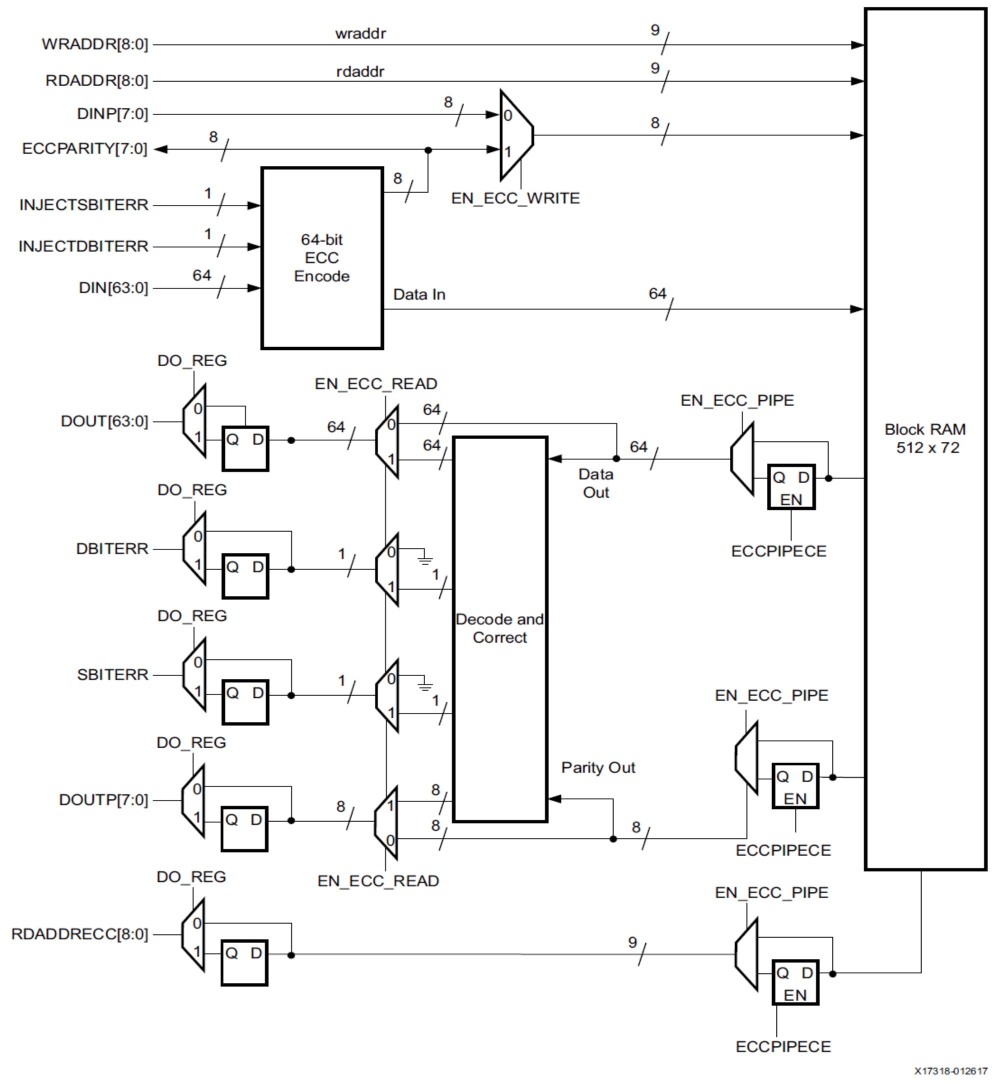


Figure 1-32: **Top-Level View of Block RAM ECC**

FIFO

- Common-clock or independent-clock
  - Avoid ambiguity, glitches, meta-stability
  - Pass data across different clock domains
- Standard or first-word fall-through (FWFT)
- Port A as FIFO read port, Port B as FIFO write port
- 18Kb (FIFO18E2): 4Kx4, 2Kx9, 1Kx18, 512 x36
- 36Kb (FIFO36E2): 8Kx4, 4Kx9, 2Kx18, 1Kx35, 512 x72

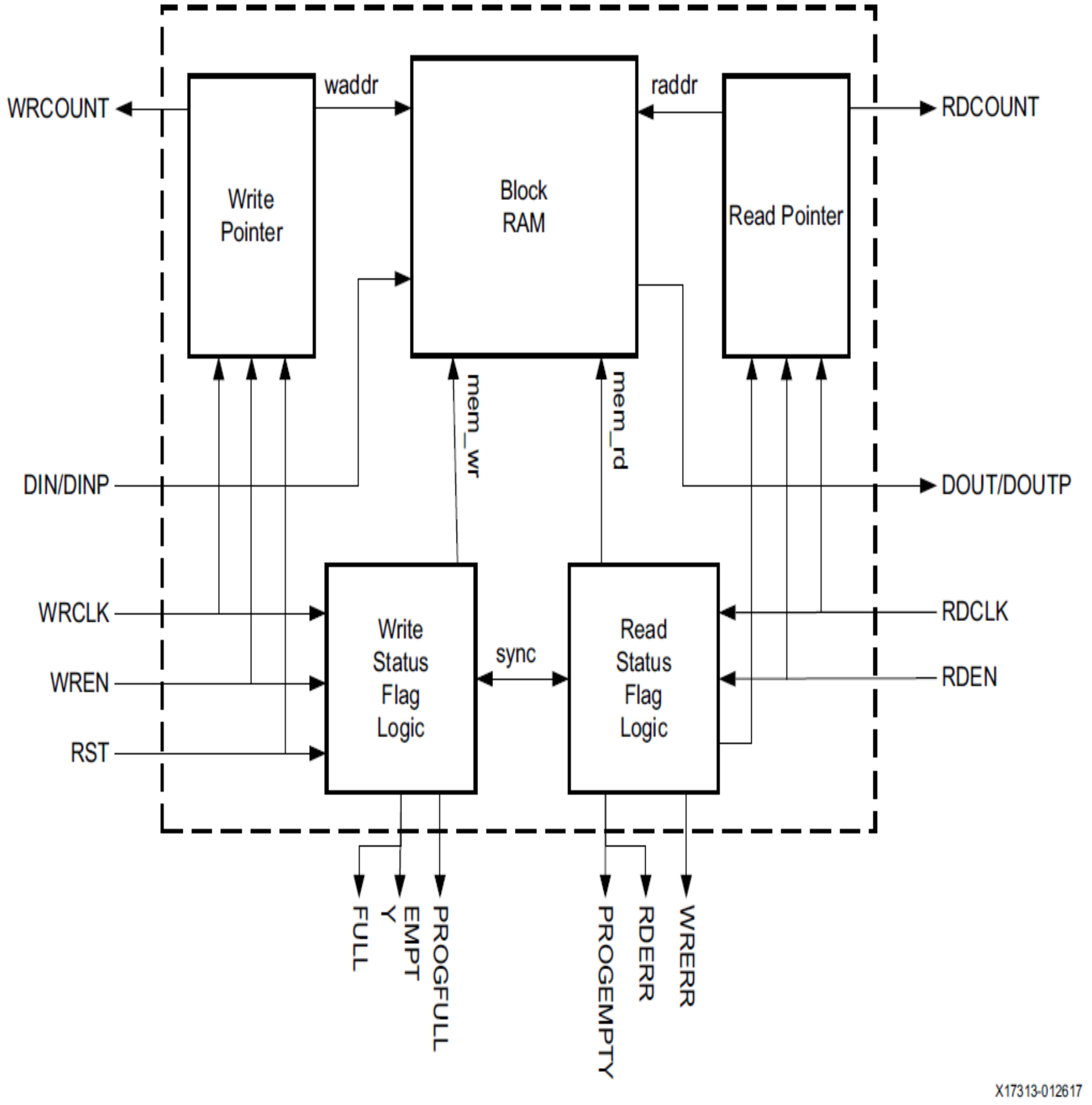


Figure 1-20: **Top-Level View of FIFO in Block RAM**

Cascade FIFO

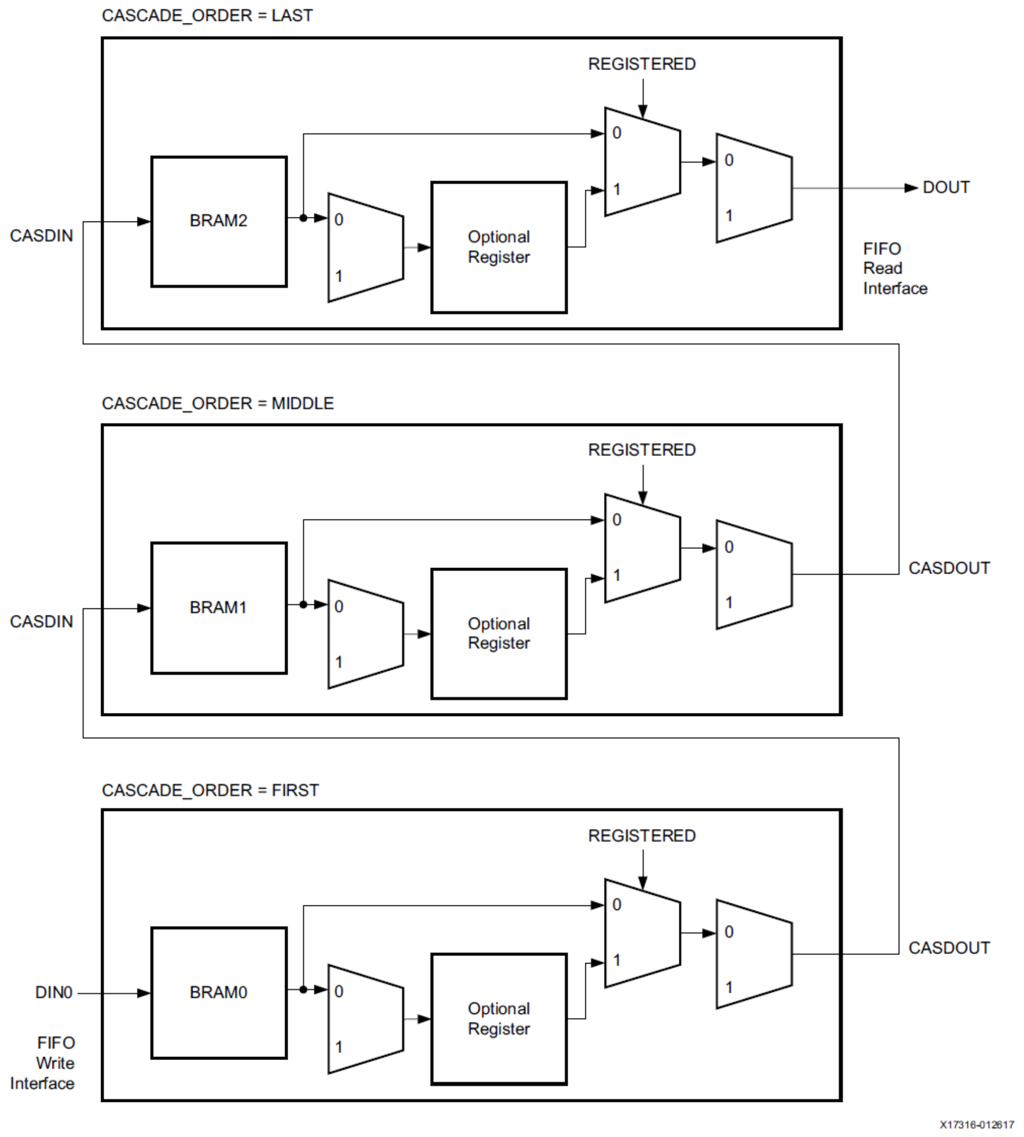
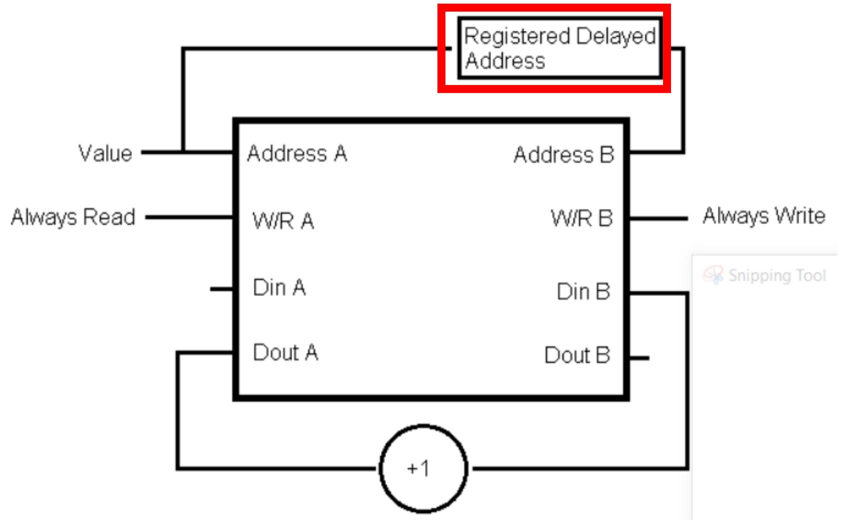


Figure 1-30: **FIFO Serial Cascade**

Creative Use of BlockRAM

- Read-Modify-Write, One Operation Per Clock
  - Dual-port – Port-A as the read port, Port-B as the write port
  - Use one common clock for both ports

- Shift Registers (FIFO)



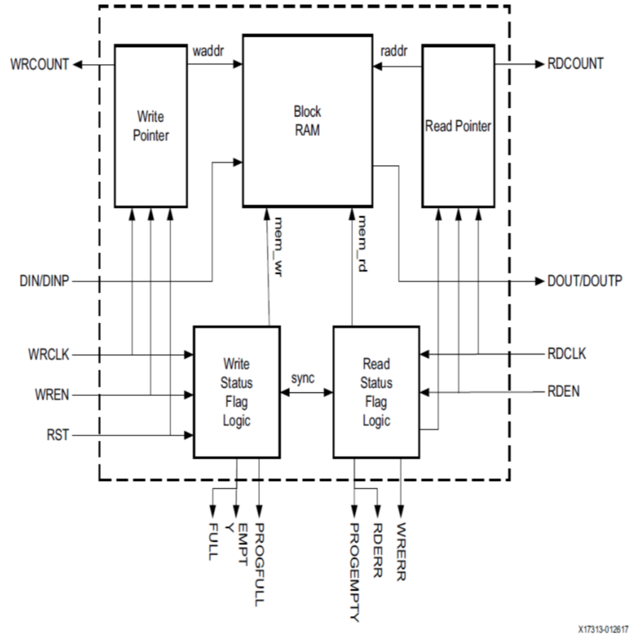


Figure 1-20: Top-Level View of FIFO in Block RAM

wp335 - https://docs.xilinx.com/v/u/en-US/wp335

Creative Use of BlockRAM : State Machines

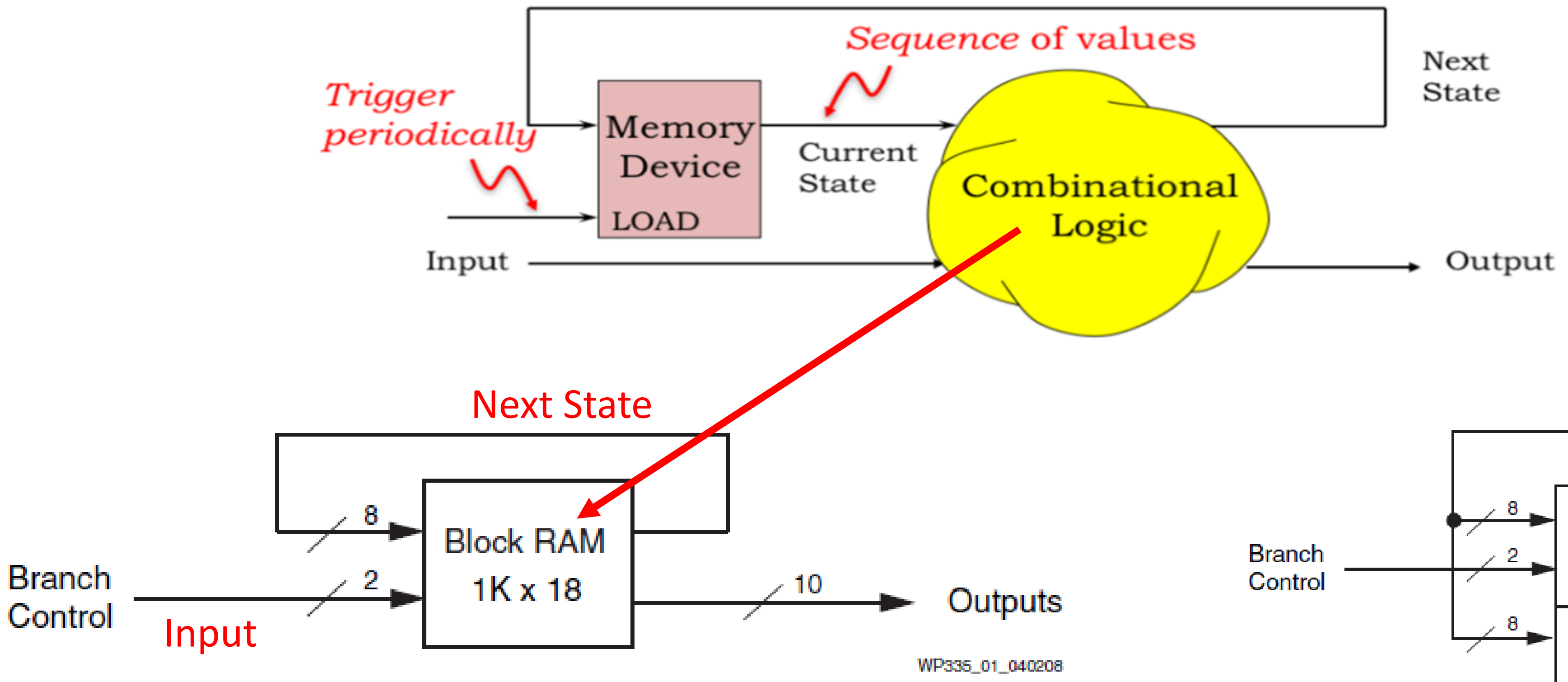


Figure 1: **Simple Finite State Machine**

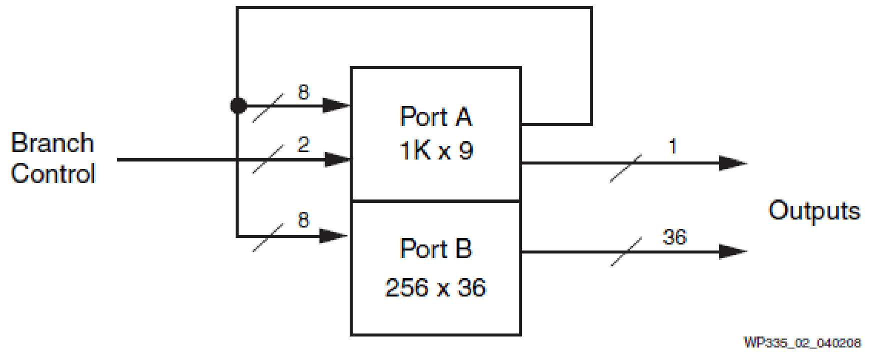


Figure 2: **FSM with Additional Outputs**

wp335 - https://docs.xilinx.com/v/u/en-US/wp335

UltraRAM

- 288kb (8xBRAM) – Single Dual port 4K x 72
- No FIFO
- Single clock – 2 ports
  - Each port can operate read/write independently
  - Port A take precedence over Port B
- Cascadable from 288kb to 300Mb – replace external SRAM
- ECC
- Optional pipeline flip-flops (input, output)
- Initialized to all 0's

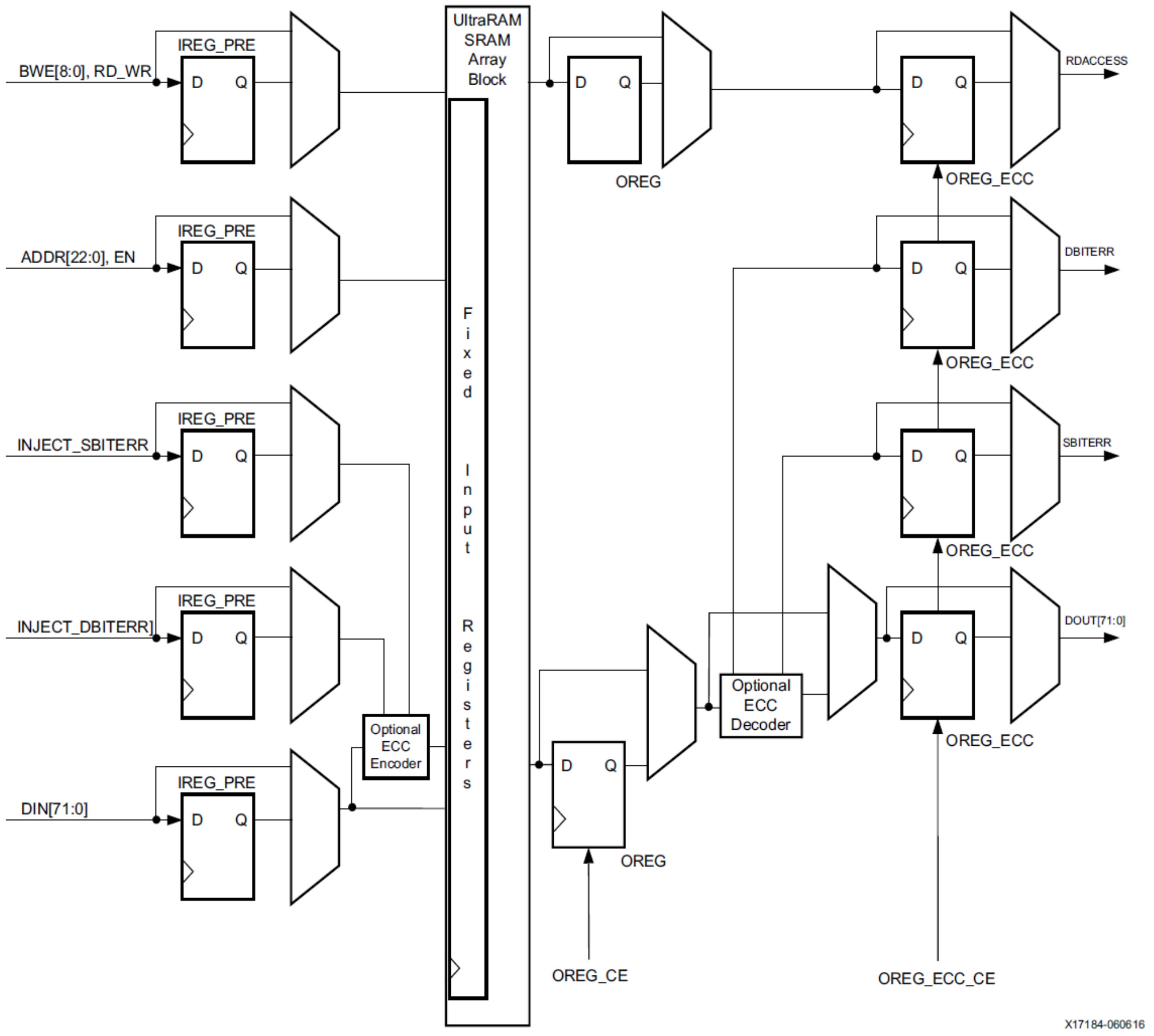


Figure 2-3: **Simplified Single UltraRAM Block Diagram without Cascade (One Port shown)**

[UltraRAM https://www.xilinx.com/support/documentation/white_papers/wp477-ultraram.pdf](https://www.xilinx.com/support/documentation/white_papers/wp477-ultraram.pdf)
https://www.xilinx.com/products/technology/memory.html

Cascade UltraRAMs to build deep memory

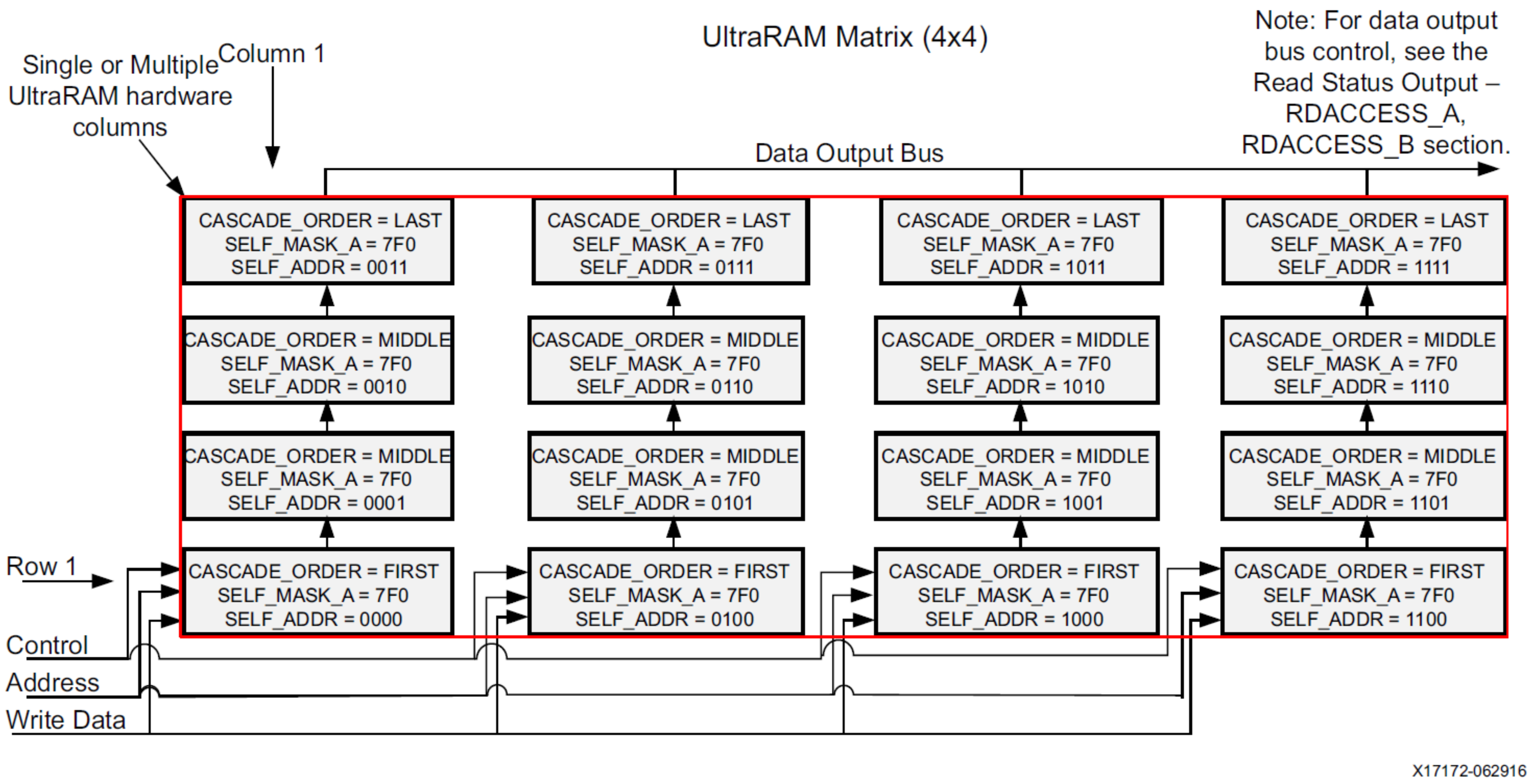


Figure 2-7: **4x4 UltraRAM Matrix**

Memory Inference – FPGA & ASIC

FPGA Memory Inference

- To use Block RAMS and LUT RAMS,
  - Instantiate the memory
  - Use inference

- Verilog coding determines the inference of BlockRAM, LUTRAM, or flip-flops.
  - Size, read style (synchronous v.s. asynchronous).

- See **XST User Guide** for examples.

https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/xst.pdf

Distributed RAM Inference

```
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

```
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.

BlockRAM Inference an Initialization

```
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)

True Dual-Port BlockRAM Inference

```
module test (data0,data1,waddr0,waddr1,we0,we1,clk0, clk1, q0, q1);

   parameter d_width = 8;  parameter addr_width = 8; parameter mem_depth = 256;

   input [d_width-1:0] data0, data1;
   input [addr_width-1:0] waddr0, waddr1;
   input we0, we1, clk0, clk1;

   reg [d_width-1:0] mem [mem_depth-1:0]
   reg [addr_width-1:0] reg_waddr0, reg_waddr1;
   output [d_width-1:0] q0, q1;

   assign q0 = mem[reg_waddr0];
   assign q1 = mem[reg_waddr1];

   always @(posedge clk0)
     begin
       if (we0)
         mem[waddr0] <= data0;
         reg_waddr0 <= waddr0;
     end

   always @(posedge clk1)
     begin
       if (we1)
         mem[waddr1] <= data1;
         reg_waddr1 <= waddr1;
     end

   endmodule
```

Memory Usage in Processor

- **Register File: Consider distributed RAM (LUT RAM)**
  - Size is close to what is needed: distributed RAM primitive configurations are 32 or 64 bits deep. Extra width is easily achieved by parallel arrangements.
  - LUT-RAM configurations offer multi-porting options - useful for register files.
  - Asynchronous read, might be useful by providing flexibility on where to put register read in the pipeline.
- **Instruction / Data Caches : Consider Block RAM**
  - Higher density, lower cost for large number of bits
  - A single 36kbit Block RAM implements 1K 32-bit words.
  - Configuration stream based initialization, permits a simple “boot strap” procedure.
- **Other Memories? FIFOs? Video “Frame Buffer”? How big?**

Memory Inference in ASIC

- ASIC Synthesis tool does not infer memories from RTL in the way FPGA synthesis tools do.

- Use Memory Compiler
  - Generate memory block with the specification (bitwidth, depth, # of port)
  - Generate verilog behavior model with timing check in specify block
  - Generate the .lib file with timing data to synthesis timing

- In RTL code, explicitly instantiate the memory, and design its control signals, e.g. Enable, read/write, address, input/output data

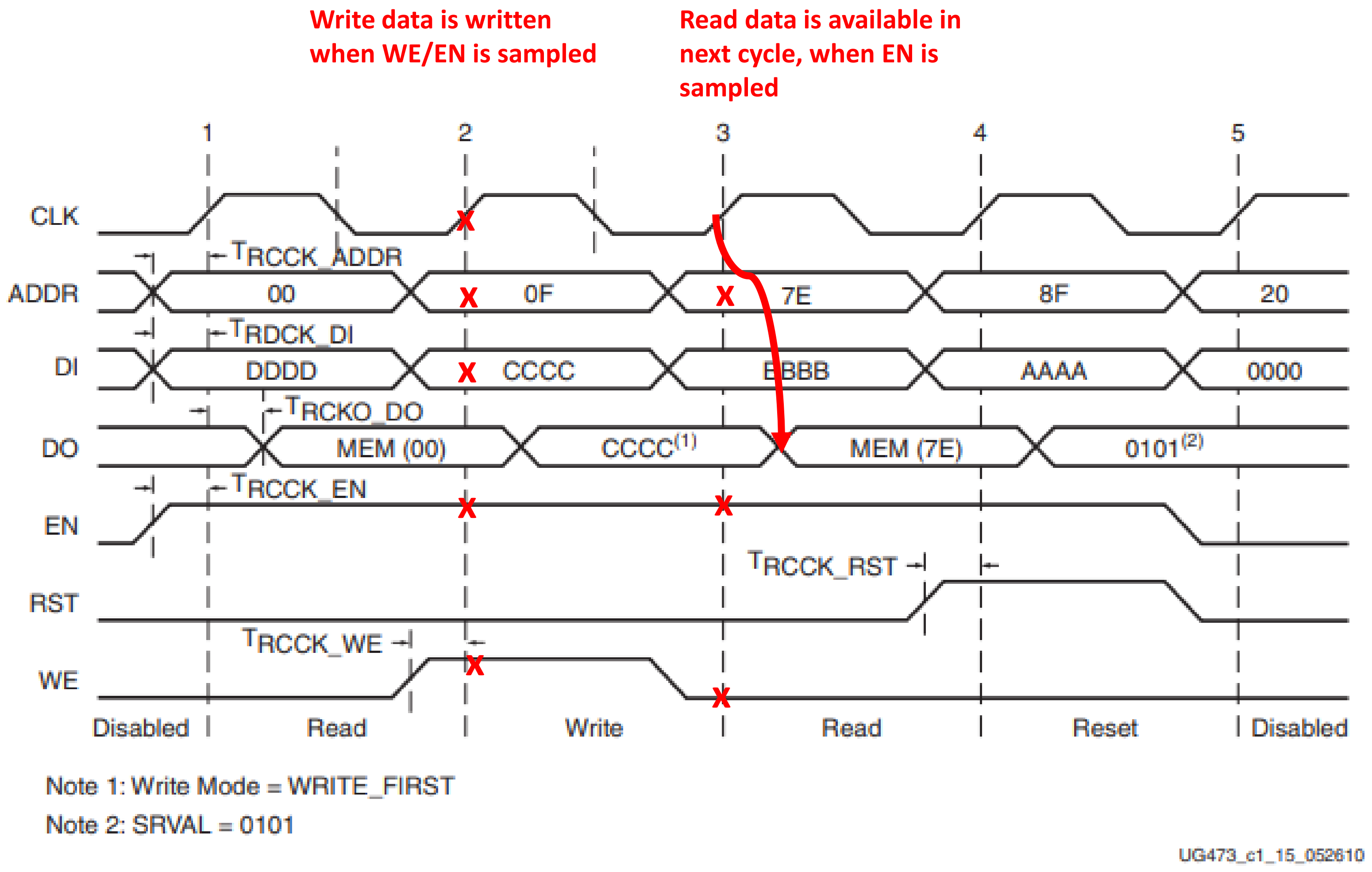
SRAM Access in behavior model and Synthesizable Hardware Design
ref : spiflash-vip.v   v.s. spiflash.v

Example: spiflash design

- spiflash-vip.v - spiflash behavior model
  - access sram as an model
- bram.v – BlockRAM behavior model
- spiflash.v
  - Adapt from spiflash-vip.v, synthesizable verilog design
  - Generate bram interface signal to access data

Reference code:
https://github.com/bol-edu/caravel-soc_fpga-lab/tree/main/spiflash

SRAM Access Timing



SRAM access has different modes, refer to https://docs.xilinx.com/r/en-US/am007-versal-memory/Read-Operation?tocId=VRYu0HURA1U147fufYDMNQ

BRAM Model

@(posedge CLK) if EN0 is sampled, output its memory content to Do0

@(posedge CLK) if WE[3:0] is sampled, RAM is written with Di0 per byte

```
module bram #( parameter FILENAME = "firmware.hex")
(
   input  wire   CLK;
   input  wire   [3:0]  WE0;
   input  wire   EN0;
   input  wire   [31:0]  Di0;
   output  reg   [31:0]  Do0;
   input  wire   [31:0]  A0
)
reg [7:0] RAM[0:4*1024*1024-1];   // Declare Memory Storage
```

```
always @(posedge CLK)
    if(EN0) begin
        Do0 <= {RAM[{A0[31:2],2'b11}],
                RAM[{A0[31:2],2'b10}],
                RAM[{A0[31:2],2'b01}],
                RAM[{A0[31:2],2'b00}]} ;
        if(WE0[0]) RAM[{A0[31:2],2'b00}] <= Di0[7:0];
        if(WE0[1]) RAM[{A0[31:2],2'b01}] <= Di0[15:8];
        if(WE0[2]) RAM[{A0[31:2],2'b10}] <= Di0[23:16];
        if(WE0[3]) RAM[{A0[31:2],2'b11}] <= Di0[31:24];
    end
    else
        Do0 <= 32'b0;
```

```
initial begin
            $display("Reading %s",  FILENAME);
            $readmemh(FILENAME, RAM);
            $display("%s loaded into memory", FILENAME);
            $display("Memory 5 bytes = 0x%02x 0x%02x 0x%02x 0x%02x 0x%02x",
                     RAM[0], RAM[1], RAM[2], RAM[3], RAM[4]);
end
```

spiflash-vip – behavior model to access RAM

Memory defined and initialization

```
reg [7:0] memory [0:16*1024*1024-1]; // 16MB
initial begin    // memory content loaded data from file
  $readmemh(FILENAME, memory);
end
```

Memory data is available the same time address is supplied. This is not feasible in the actual memory system.

memory read access

```
buffer = memory[spi_addr]; // memory read
```

memory write access

```
memory[spi_addr] = buffer;
```

spiflash.v – Synthesizable hardware design

- Generate SRAM interface signals
  - Addr, EN, WEN, Din, Dout
  - **The interface signal is generated from internal control logic (FSM)** Interface signals follows the interface timing specification, e.g. read data is available in next clock cycle

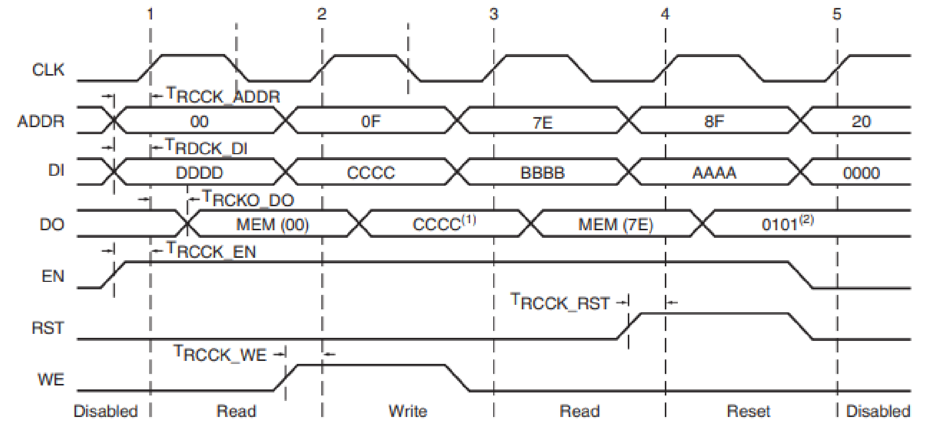
BRAM RAM

Addr
EN
WEN
Din
Dout

Clk
Rst

```
// BRAM Interface
  assign romcode_Addr_A = {8'b0, spi_addr};
  assign romcode_Din_A = 32'b0;
  assign romcode_EN_A = (bytecount >= 4);
  assign romcode_WEN_A = 4'b0;
  assign romcode_Clk_A = ap_clk;
  assign romcode_Rst_A = ap_rst;

wire [7:0] memory;
assign memory =
            (spi_addr[1:0] == 2'b00) ? romcode_Dout_A[7:0] :
            (spi_addr[1:0] == 2'b01) ? romcode_Dout_A[15:8] :
            (spi_addr[1:0] == 2'b10) ? romcode_Dout_A[23:16] :
                                          romcode_Dout_A[31:24] ;
```

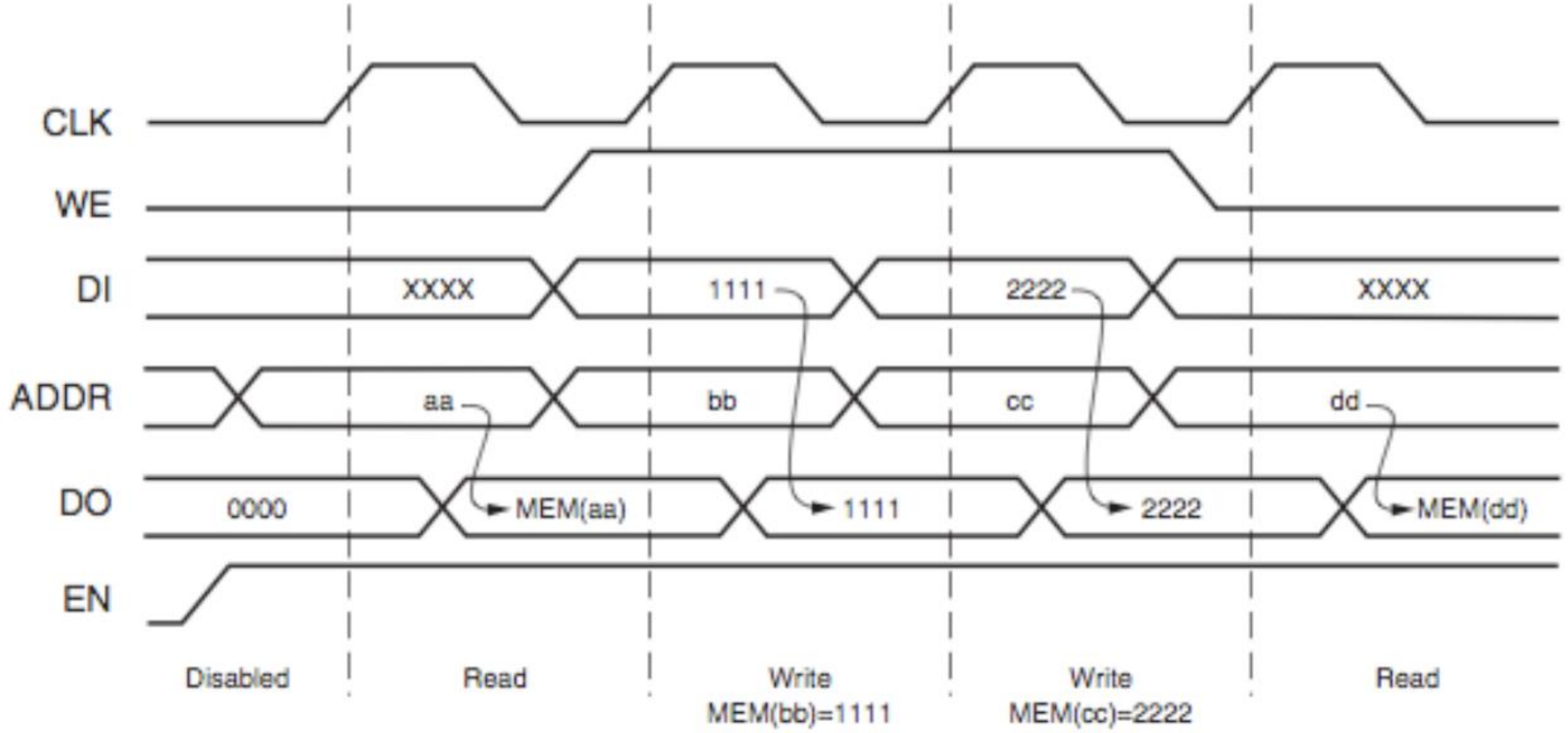


Note 1: Write Mode = WRITE_FIRST
Note 2: SRVAL = 0101

UG473_c1_15_052610

Supplement

BlockRAM Access Timing – Synchronous Read/Write


CLK
WE
DI
XXXX
1111
2222
XXXX
ADDR
aa
bb
cc
dd
DO
0000
MEM(aa)
1111
2222
MEM(dd)
EN
Disabled
Read
Write
MEM(bb)=1111
Write
MEM(cc)=2222
Read