Register Files and Memories

ECE 554

Digital Engineering Laboratory

C. R. Kime and M.J. Schulte and M. Lipasti

Modified: Kewal K. Saluja

Register Files and Memories

- Register Files
  - Issues and Objectives
  - Register File Concepts
  - Implementation of Register Files
  - Workarounds For Xilinx FPGAs
  - Bottom Line
- Memories
  - Timing Issues
  - Width Expansion

Issues and Objectives

- Issues
  - ECE 554 projects require a broad range of register file and memory configurations
  - ECE 554 lab boards provide very limited structures for implementing register files and memories.
- Objectives:
  - To develop techniques for implementing a broad range of register file and memory configurations by using available lab board structures

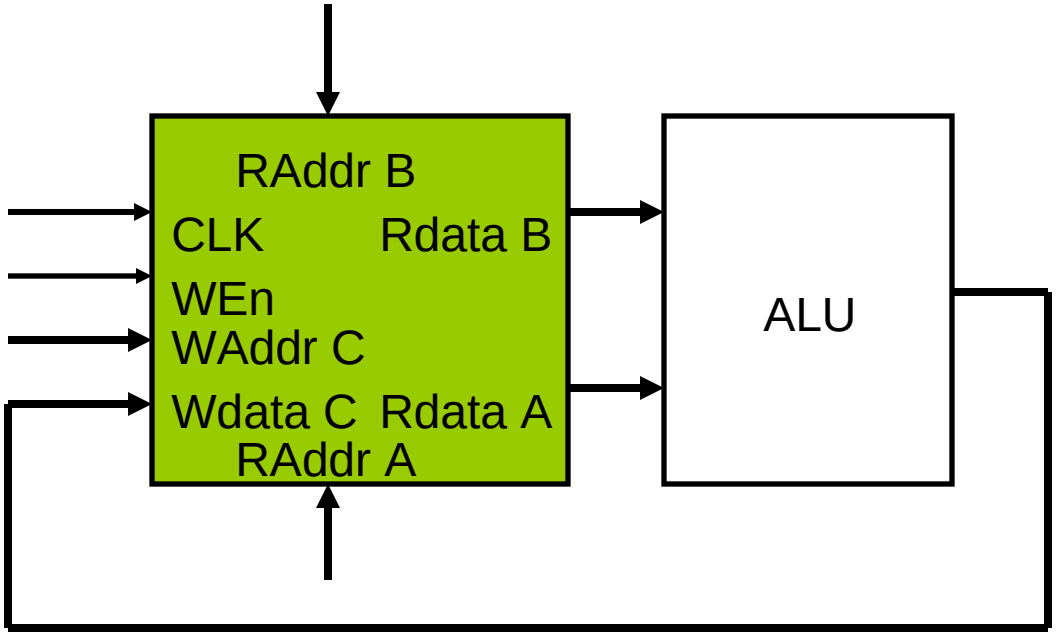
Register File Concepts

- Register file environments
  - Non-Pipelined
  - Pipelined
- Register File Ports
  - Address Ports
  - Data Ports
  - Control Ports
- Register File Implementations

Register File Ports

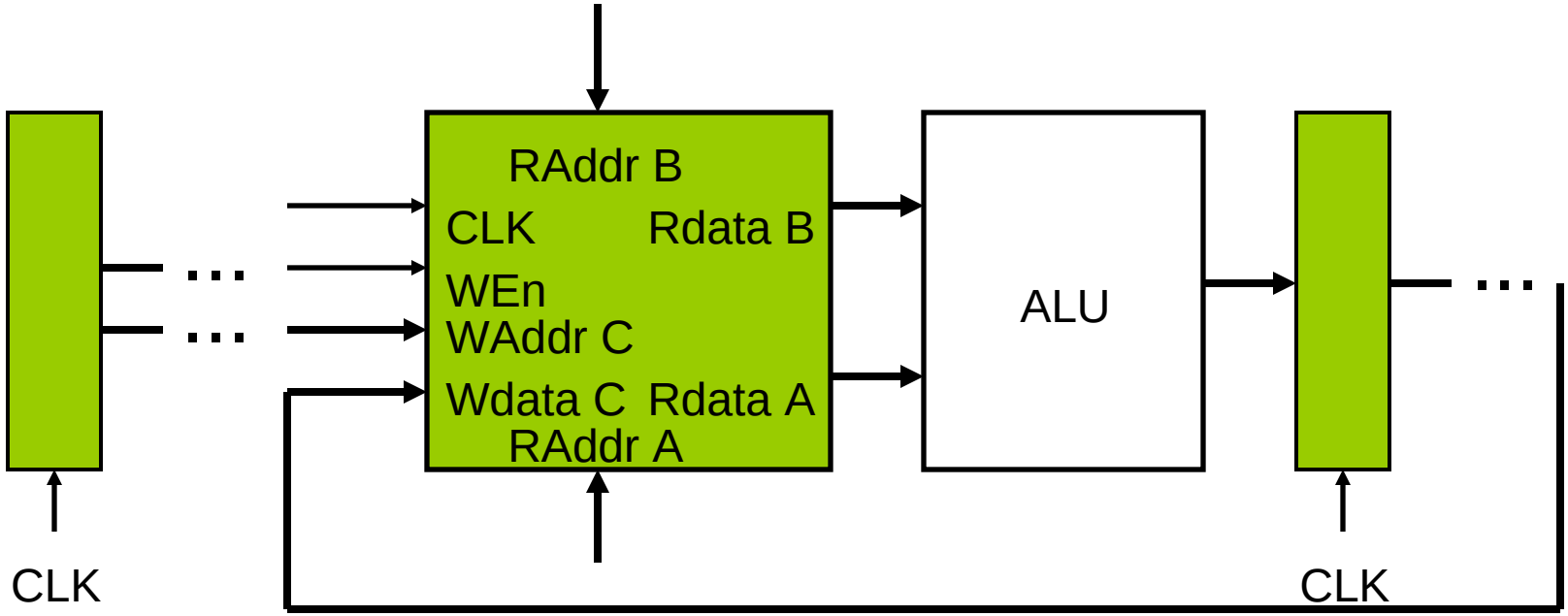
- Address
  - Read
  - Write
  - Shared
- Data
  - Input
  - Output
  - Bidirectional
- Control
  - Write Enable, Read/$\overline{\text{Write}}$, Enable, Read, Write, CLK

Environment - Non-Pipelined



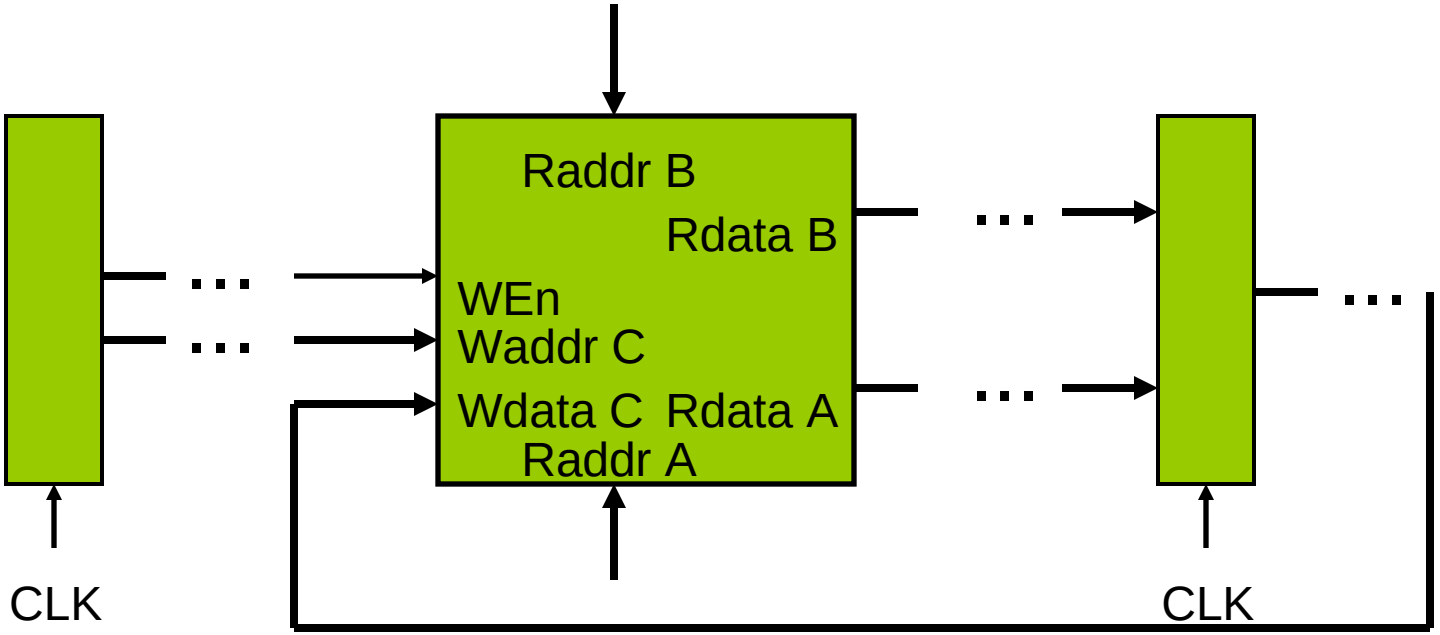
- Input Wdata C not registered outside of Register File
  - Clock controls when Wdata C is written
- Inputs WEN and Waddr C may or may not be registered

Environment - Pipelined 1



- Register File is part of the pipeline platform
- Inputs may or may not be registered
  - Clock controls when Wdata C is written

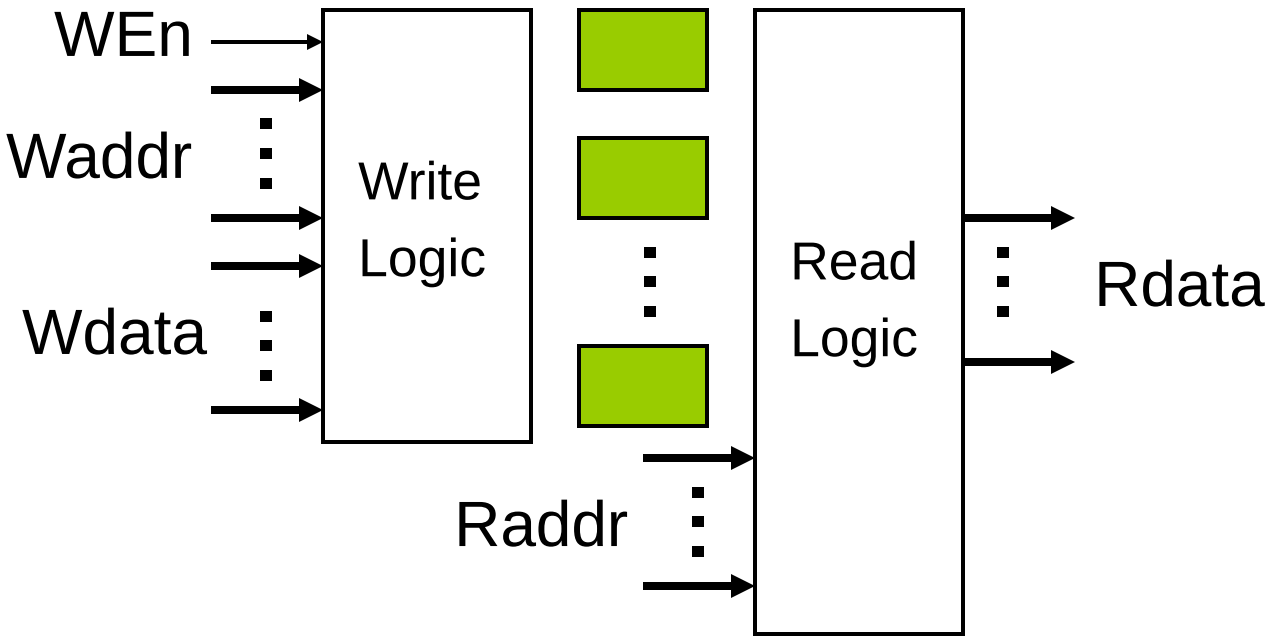
Environment - Pipelined 2



- Register File
  - is between pipe stages
  - is not clocked - WEN controls latches => SRAM
- Inputs
  - may or may not be registered, but
  - register must be between Rdata A, Rdata B, and Wdata C

Latch-Based

- Latch/bit of file
- Latch control can be Write Enable and addresses or some combination of other signals and addresses

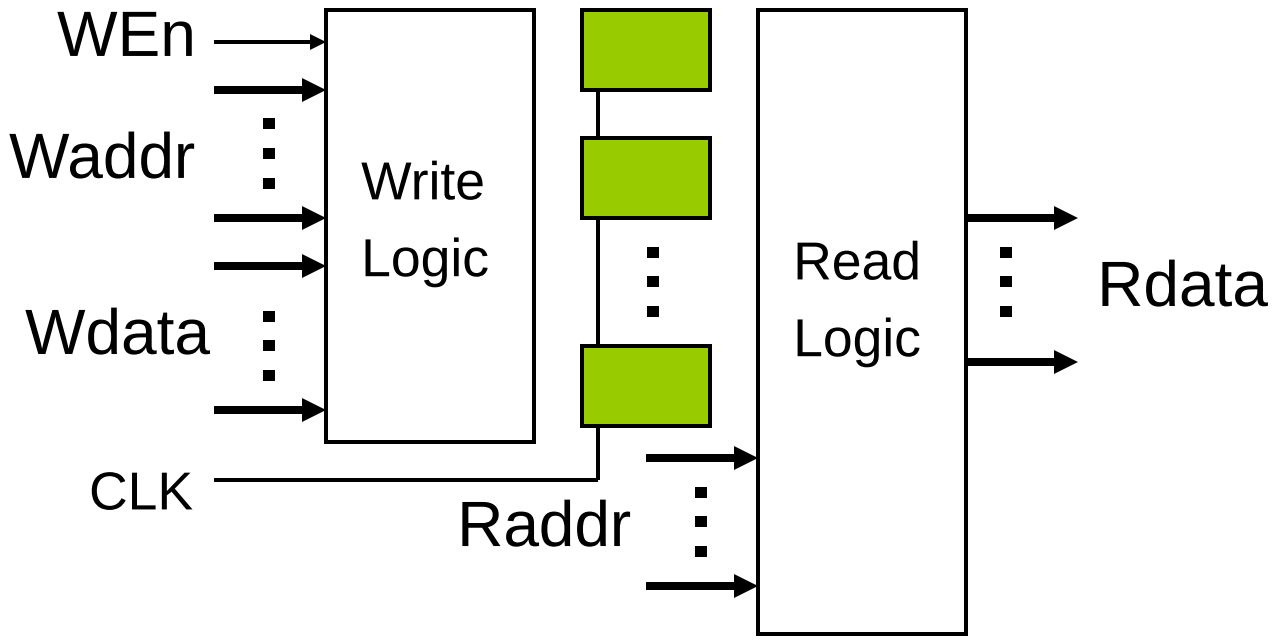


Latched-Based

- Level-sensitive write (assume positive level)
- Setup time on write address relative to leading edge of Wen
- Hold time on write address relative to trailing edge of Wen
- Setup and hold time on write data relative to trailing edge of Wen
- Latches cannot be in closed loop without:
  - Additional latch on different clock in loop, or
  - Flip-flop in loop

Flip-flop (Latch Pair)-Based

- Flip-flop/bit of file
- Flip-flop is clocked by CLK or some combination of CLK and other signal and enabled by addressing logic and combination of other signals

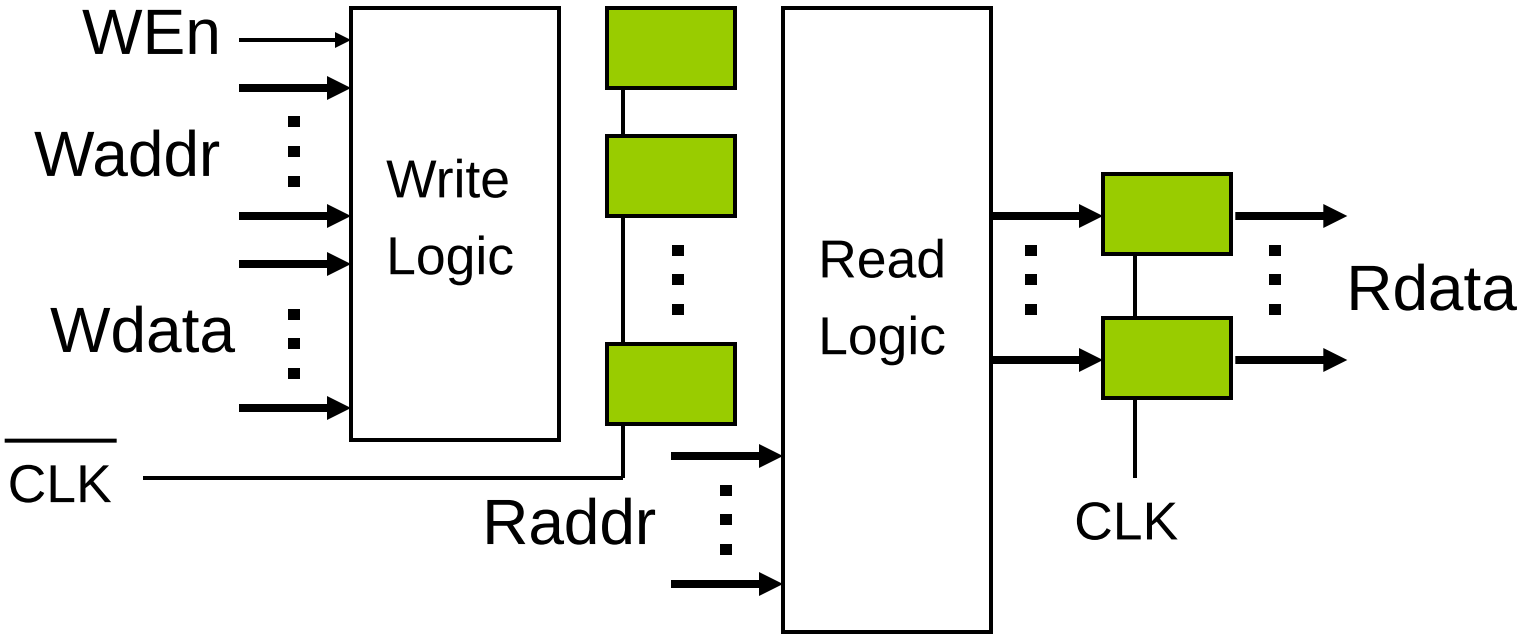


Flip-flop (Latch Pair)-Based

- Write Logic adds setup-time to that for flip-flops
- Read Logic adds propagation delay to that for flip-flops
- Acts as edge triggered flip-flop register file with above delays added

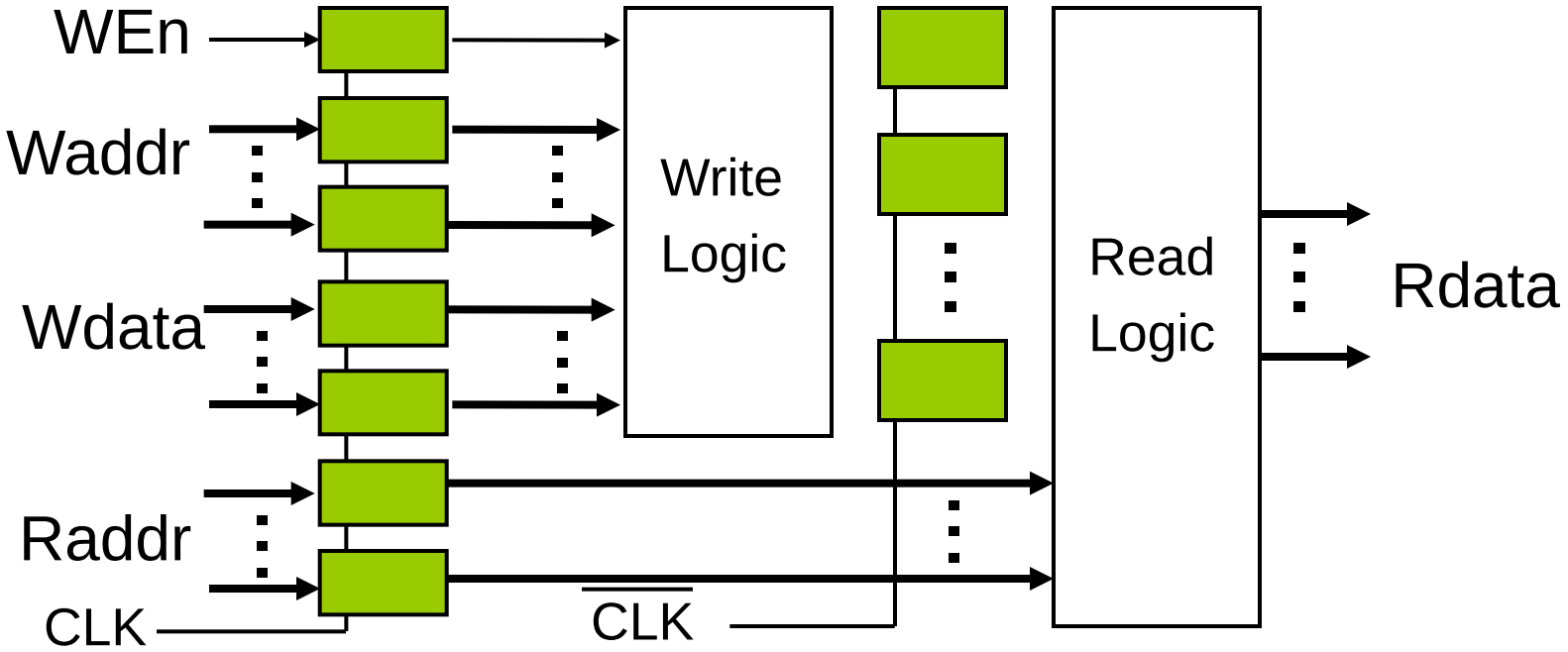
Flip-flop (Shared-Slave)-Based

- Latch/bit of file plus latch/bit of output
- Master latches are clocked by $\overline{CLK}$ or some combination of $\overline{CLK}$ and other signal and enabled by addressing logic and combination of other signals; slave latches clocks by CLK



Flip-flop (Shared-Master)-Based

- Latch/bit of file plus latch/bit of input
- Master latches are clocked by $\overline{CLK}$ some combination of $\overline{CLK}$ and other signal and enabled by addressing logic and combination of other signals; slave latches clocks by CLK



Implementation of Register Files

- Custom VLSI SRAM
- Classic SRAM
- Xilinx Virtex SRAM
  - Specifications
  - Shortcomings

Custom VLSI SRAM

- Is the most flexible of all implementation techniques
- Can be used to implement any combination of variants discussed
  - Latch-based straightforward; needs additional rank of latches to do flip-flop-based
  - Short of performance issues due to capacitance, can implement any port configuration in a singe storage element array.

Classic SRAM

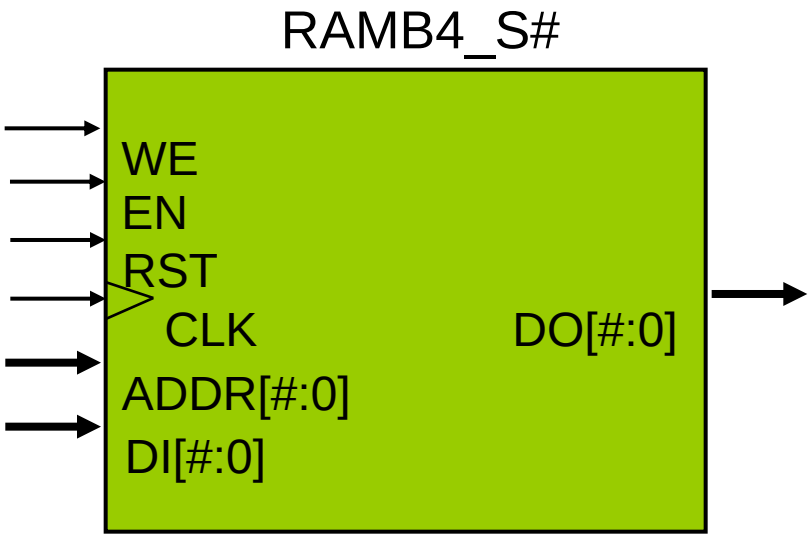
- Has single RWaddr port, single Wdata port, and single Rdata port and is latch-based.
  - Due to single address port, can handle only one R or W access per clock cycle
- Expansion to *n* R address/data ports
  - Place *n* SRAMs in parallel with the write accomplished by:
    - Applying same address to all RWaddr, and
    - Wiring together all Wdata ports
- Expansion to *m* W address/data ports
  - Add an m-way multiplexer to address port
  - Use a clock that is *m* times CLK and multiplex the writes over m clocks

Classic SRAM (Continued)

  - Addresses must be switched on positive clock edge
  - WEn must be generated from negative clock edge and positive clock edge
- Expansion to *m* W address/data ports and *n* R address/data ports
  - Doing both expansions above
    - Using (m +1)-way multiplexer, and
    - A clock that is (m + 1) times CLK
- Virtex Distributed SelectRAM
  - The SRAM capability provided in CLBs
  - Can be used with expansion methods here in classic asynchronous SRAM mode or some synchronous modes
  - Getting reliable timing is tricky - may require more complex clocking!

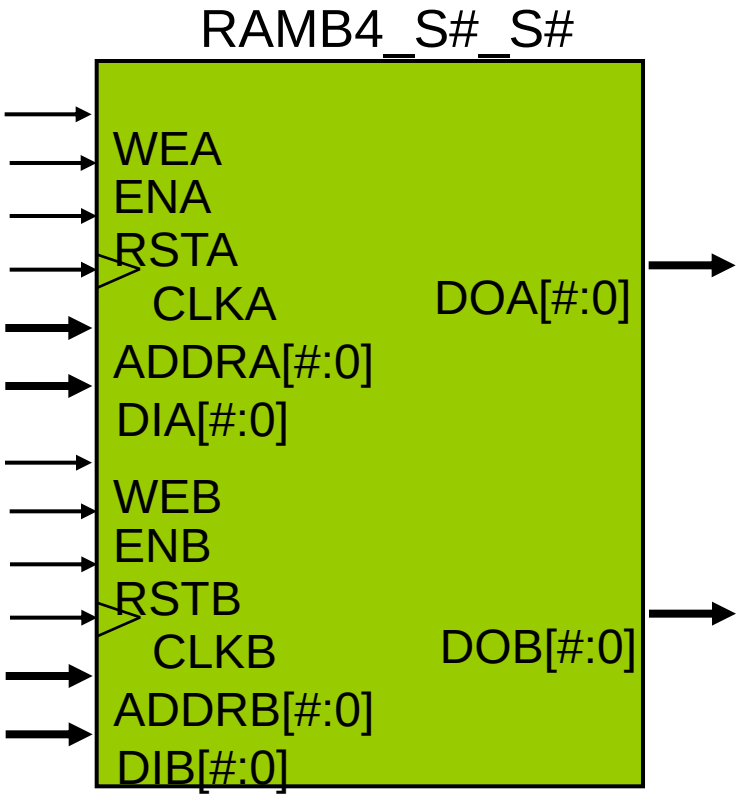
Virtex Block SRAM Specifications

- Symbol - Single Port



Virtex Block SRAM Specifications

- Symbol - Dual Port



Virtex Block SRAM Specifications

- Functionality
  - A WRITE operation of data DI to address ADDR occurs for WE = 1, EN = 1, RST = 0 and a positive edge on CLK. DI can also be read on DO after a delay.
  - A READ operation from address ADDR occurs for WE = 0, EN = 1, RST = 0 and a positive edge on CLK.
  - A RESET operation occurs on the DO latches only for EN = 1, RST = 1, and a positive edge on CLK

Virtex Block SRAM Specifications

- Functionality
  - CLK, EN, WE, and RST can also be programmed to be active low
  - Conflicts for Dual Port SRAM
    - Simultaneous WRITEs to same location give invalid data
    - A simultaneous READ on the alternate port of a location being written gives invalid READ data
    - A READ on the alternate port of a location being written may not be performed until after a clock-to-clock setup window

Virtex Block SRAM Specifications

- Functionality - Timing
  - EN, WE, RST, ADDR, DI are captured on the positive edge of CLK in registers
  - WRITEs into the SRAM latch array occur later due to internal timing logic
  - READs (including those associated with writes) occur later due to internal timing logic

Virtex Block SRAM Shortcomings

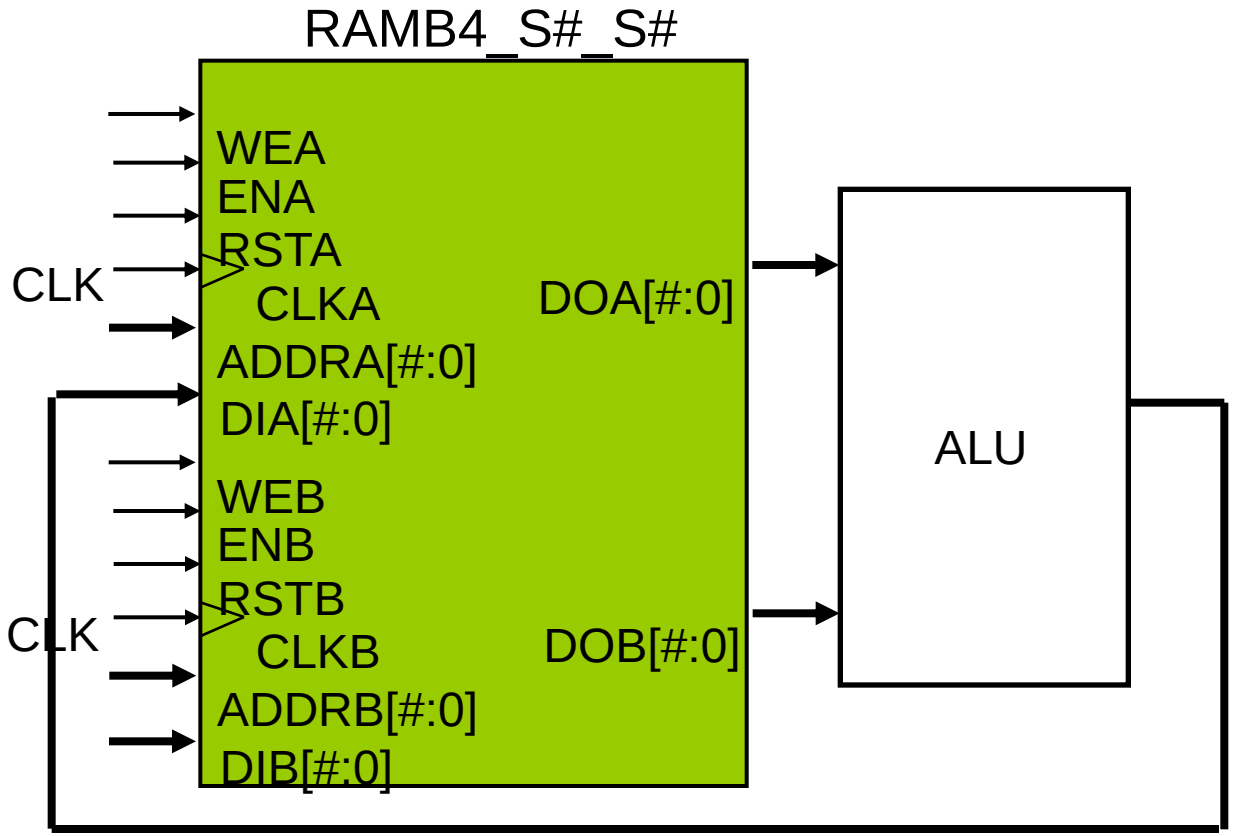
- Positive edge-triggered storage of inputs to SRAM places an implicit register in front of the SRAM
  - Combinational READs with address changing, for example, on both the leading and trailing edge of clock, impossible (i.e. dual-porting with single clock)
  - Feeding the SRAM array directly from combinational logic impossible
- Latching of outputs
  - Combinational READs impossible

Workarounds for Virtex FPGAs

- Absorbing input registers
- READ-after-alternate port-WRITE
- READ port expansion
- Inter-operation address dependency removal
- WRITE port expansion
- Absorbing output latches

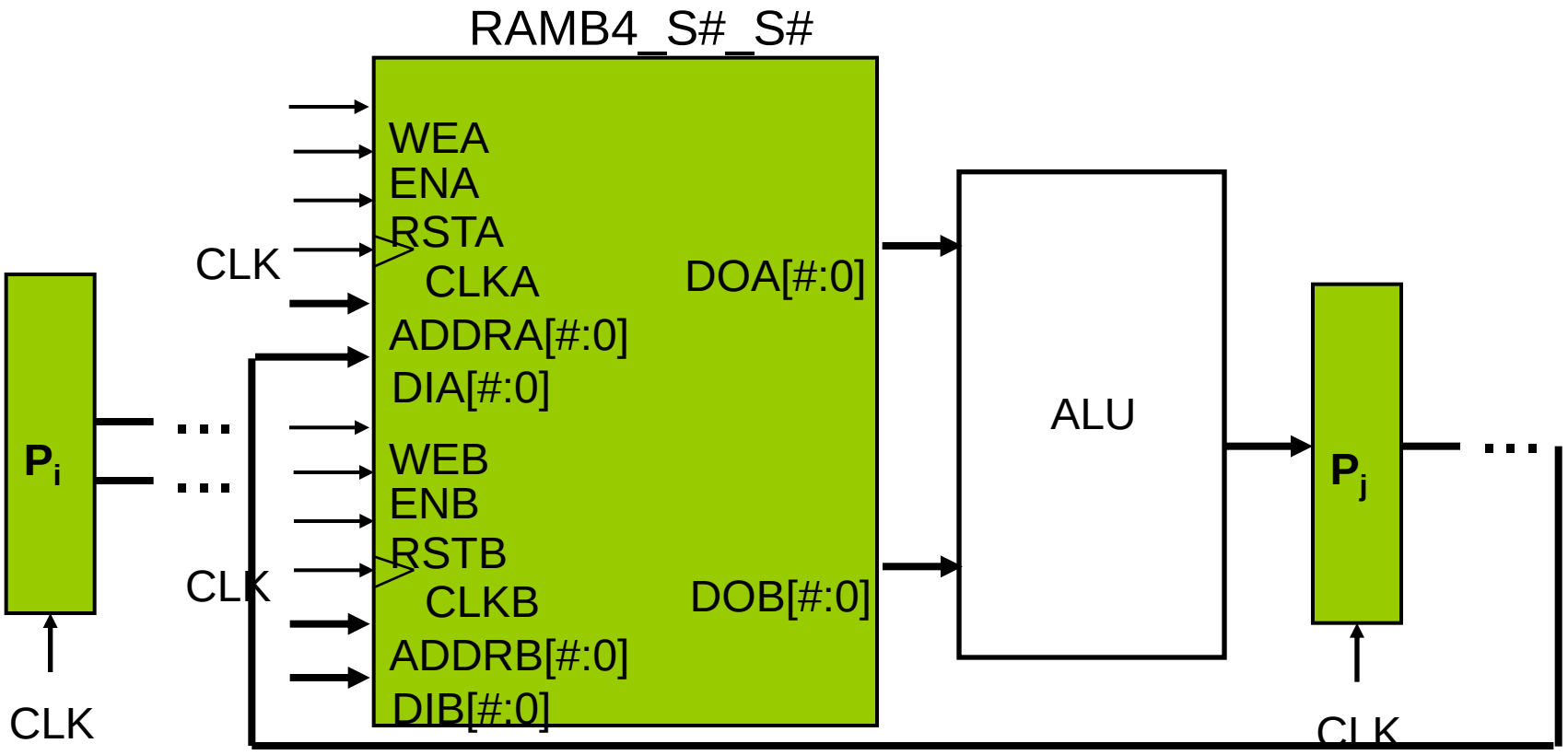
Absorbing Input Registers

- Non-Pipelined - looks like flip-flop-based file - no absorbing needed!



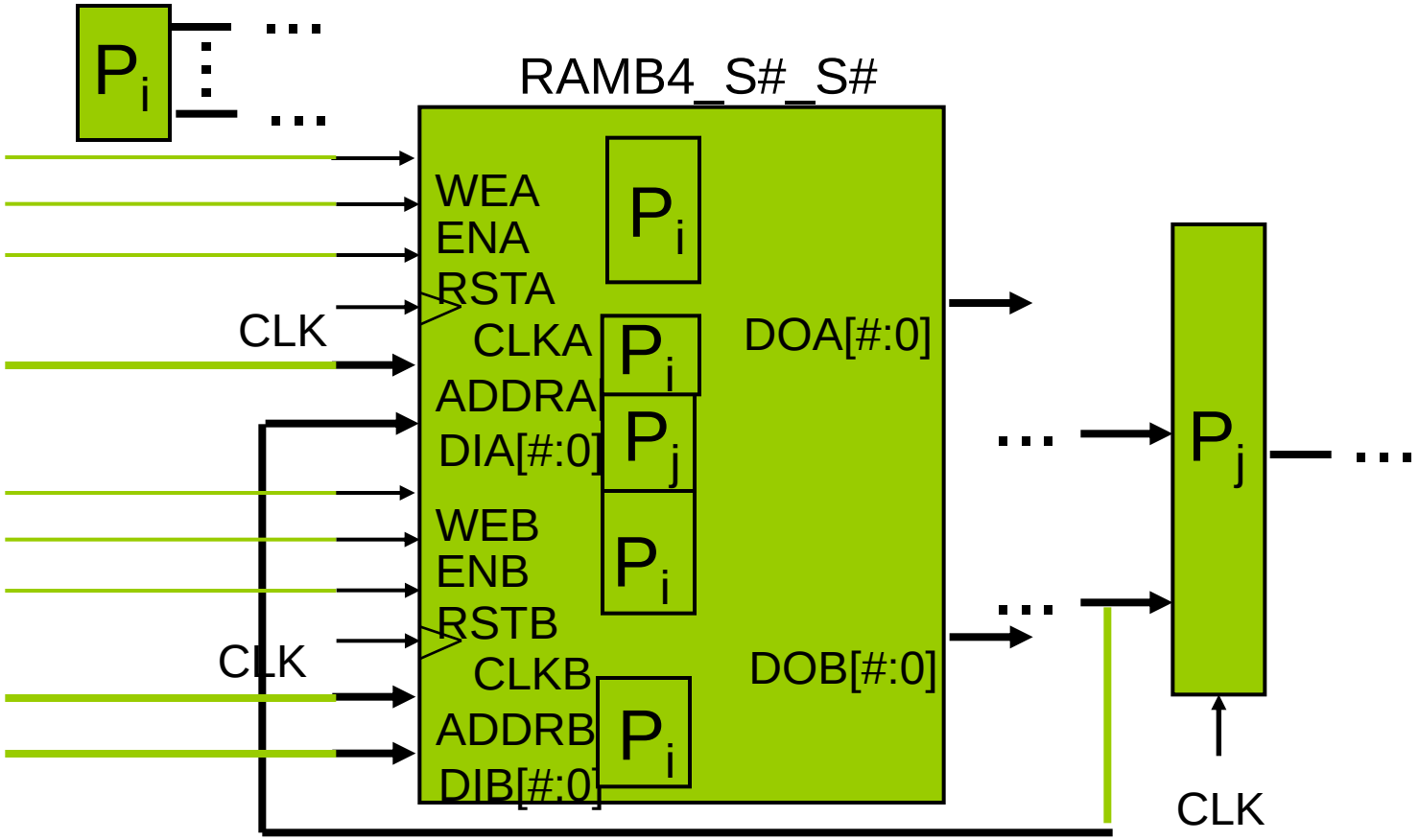
Absorbing Input Registers

- Pipelined 1 - Register file part of pipeline platform - looks like flip-flop-based file - no absorbing needed!



Absorbing Input Registers

- Pipelined 2 - Register file as SRAM between pipeline platforms - input registers give unwanted platform - must absorb into Pi and Pj platforms
- Combinational logic between $\mathbf{P_i}$ and SRAM now placed before $\mathbf{P_i}$



Absorbing Input Registers

- Summary
- Non-pipelined - No problem
- Pipelined 1 - No problem
- Pipelined 2 - Problem
  - Handle by moving pipeline platform pieces
  - Handle by converting to Pipeline 1 form
  - Affects combinational delay distribution between stages and hence may affect pipeline performance

Virtex Block SRAM Shortcomings

- Additional implication of conditions on prior page:
  - Since the Virtex Block SRAM has two addresses, it should support operands for a binary operation:

    R[ADDRA] <= R[ADDRA] op R[ADDRB]

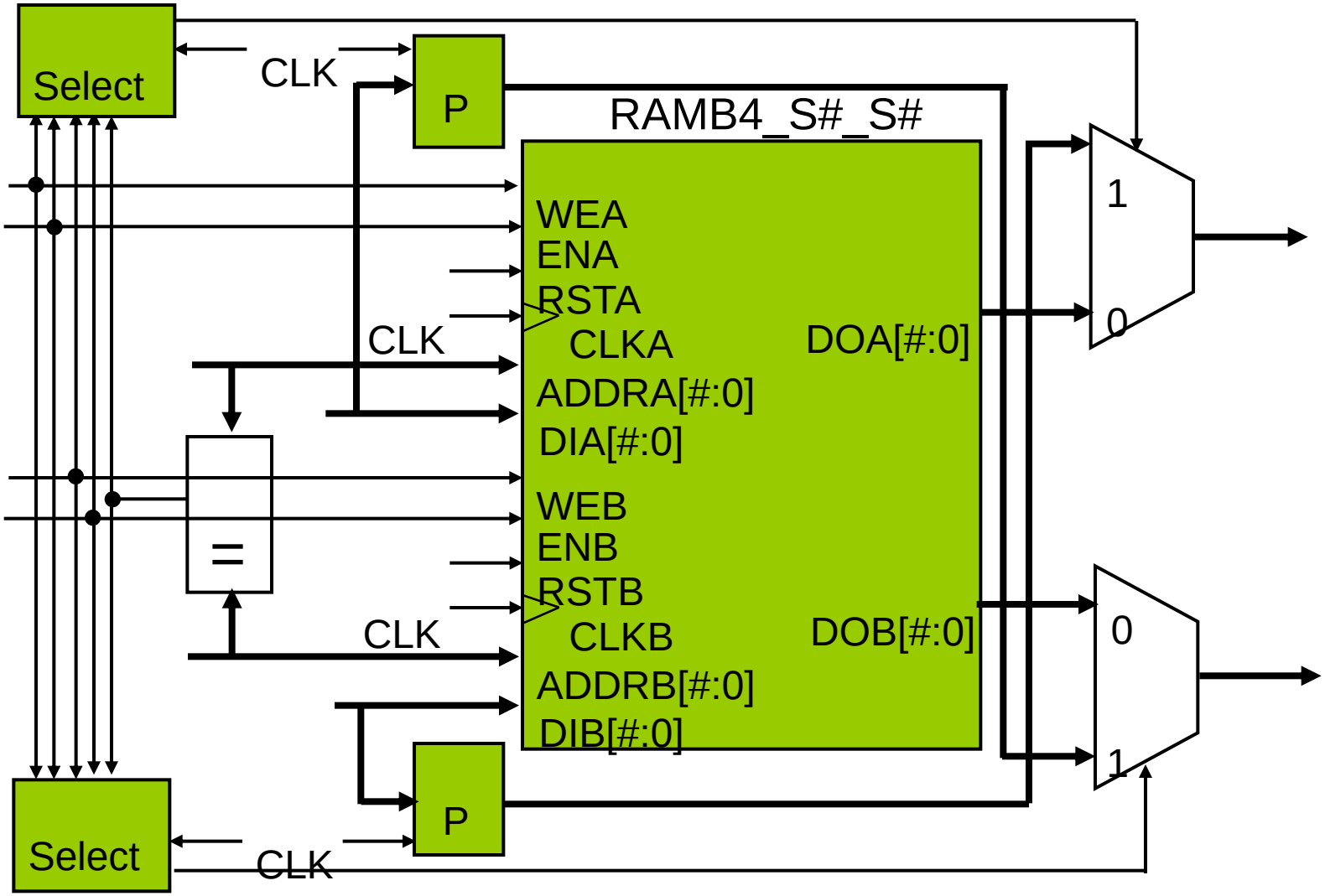
    for arbitrary ADDRA and ADDRB on each clock cycle.
  - But, it does not!
    - Since it is READ-after-WRITE, the right hand side operands are read in clock cycle i and the left hand side result is written in clock cycle i+1. One of the two addresses on the right hand side for cycle i must be the same as the write address on the left hand side for cycle i. Hazard logic would have to enforce this or insert a stall cycle (would almost always stall).
    - Further, the READ-after-alternate port-WRITE problem causes the transfer R[ADDRy] <= R[ADDRx] op R[ADDRx] to be impossible to execute after a write to ADDRx. Would cause frequent stalls, once again.

Virtex Block SRAM Shortcomings

- Using Dual Port Virtex Block SRAM with custom VLSI SRAM used as the standard for comparison
- On a single clock cycle:
  - Maximum of two independent READ or WRITE operations
  - Maximum of two READbacks of written value from WRITE operation on same port possible
  - READback of written value from WRITE on alternate port not possible

READ-after-alternate port-WRITE

- Add bypass logic outside of Virtex Block SRAM:

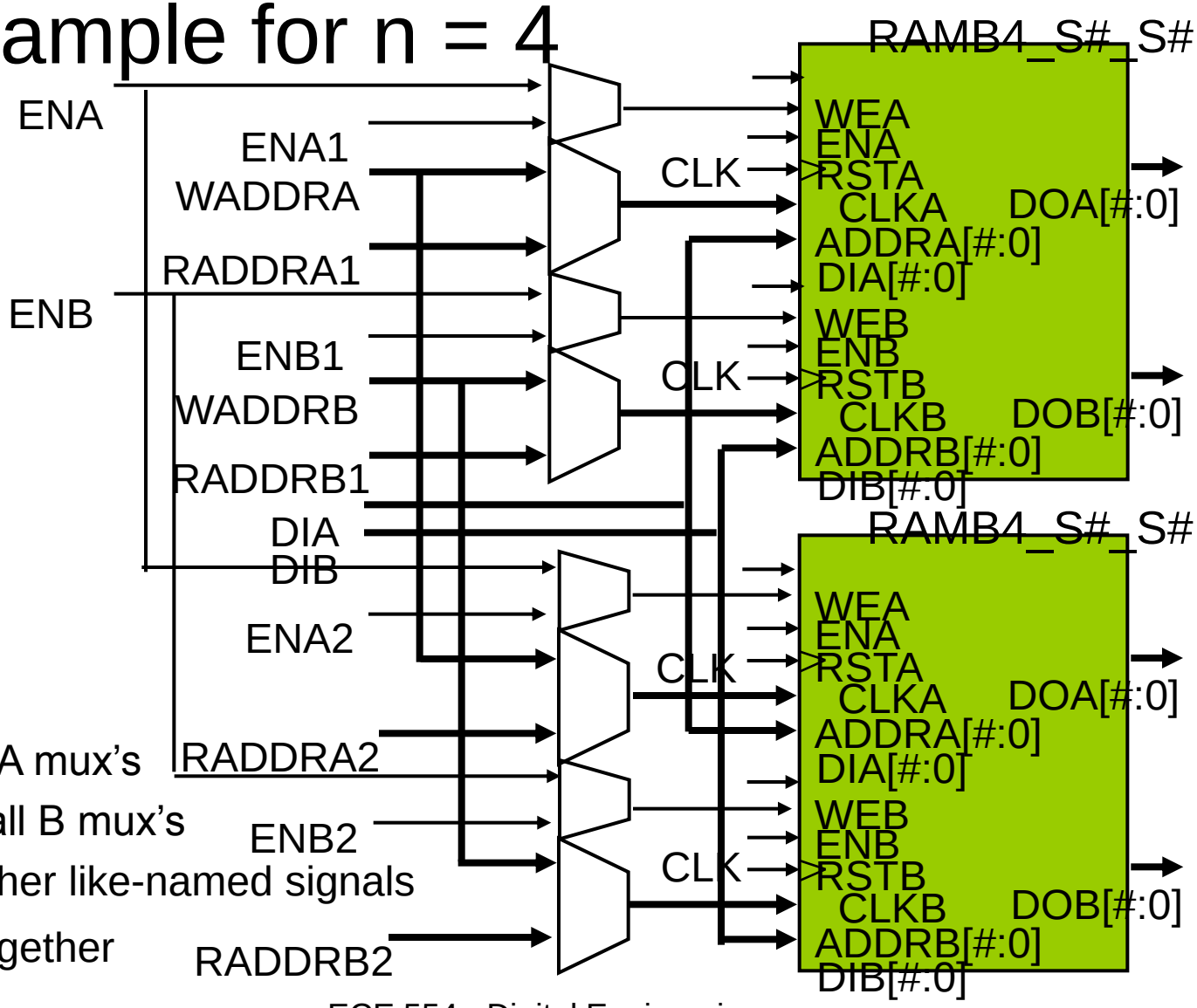


Read Port Expansion

- Expansion to *n* R address/data ports
  - Place ceiling(*n/2)* SRAMs in parallel with the two writes accomplished by:
    - Applying same address to all ADDRA and the same address to all ADDRB, and
    - Wiring together all DIA ports and all DIB ports

Read Port Expansion

- Example for n = 4



Select for all A mux's is WEA and all B mux's is WEB All other like-named signals connected together

Inter-operation Address Dependency

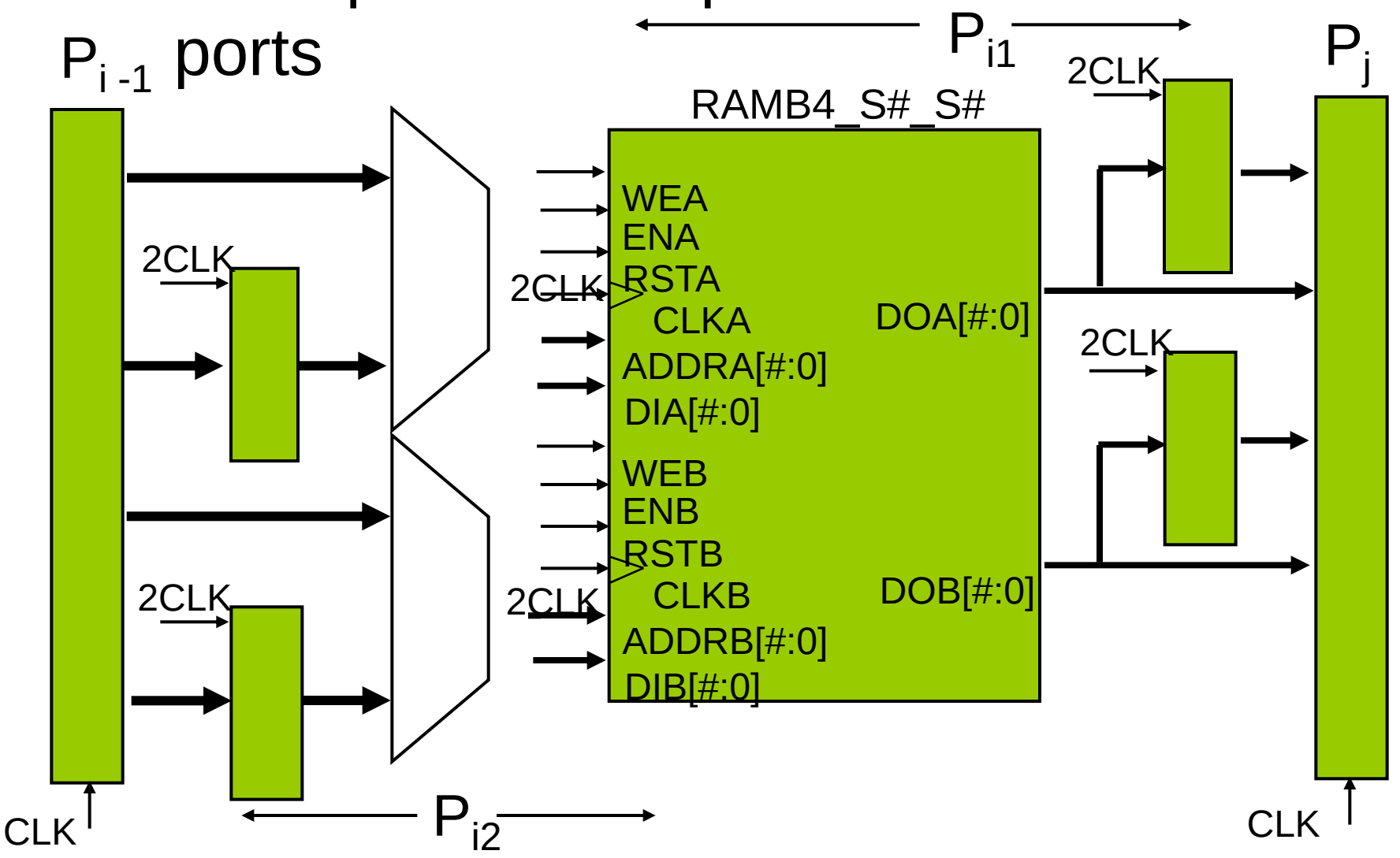
- READ-after-WRITE - Can be done for one WRITE - two READs with two parallel Dual Port Block SRAMs with READ-after-alternate port-WRITE logic added to READ side of both.
  - Parallel WRITE on A ports
  - Independent parallel READs on B-ports
- Each additional parallel Dual Port Block SRAM adds one more READ port
- Cannot accomplish WRITE-after-READ
- Cannot be done for more than one active WRITE port without using WRITE Port Expansion

Write Port Expansion

- Requires “super-clocking,” in which a clock having a multiple of the frequency of the fundamental operational clock is used to serialize Block SRAM operations.
- Requires additional registers to locally enter into and return from serialized operations
- Muxes required that are switched by the a flip-flop driven by the faster clock

Write Port Expansion

- Example - Non-Pipelined - 4 WRITE Max



Absorbing Output Latches

- The output latch is a part of the attempt at a “flip-flop” appearance for the SRAM operation.
- As such, there appears to be no way to explicitly work around it
- Other workarounds handle its effects

The Bottom Line

- Overall, it appears that the best approach is to:
  - Use a Non-Pipelined or Pipeline 1 structure
  - Use the Interoperation Dependency solution to achieve multiple dependency-free READs
  - Use WRITE Port Expansion for multiple WRITEs
  - Use the READ-after-alternate port-WRITE to get READ-after-WRITE capability
  - Use WRITE Port Expansion with READs on early subcycles to get WRITE-after-READ capability
  - Be cognizant of substantial setup times and delays for the synchronous operations
- Feel free to experiment with other approaches and apply ideas given to other Virtex Block SRAM uses

Memories

- Timing Issues
- Width Expansion

Timing Issues

- The off-chip SRAMs are asynchronous and have typical signal timing requirements
- See AS7C4096 Datasheets for timing parameters
  - Address controlled READ is easy
  - $\overline{WE}$-controlled WRITE has zero setup and hold times which look easy, but read on
- Due to unpredictable FPGA timing, timing of memory signals, particularly for WRITE should be verified.
- In worst case, may need to use “super clocking” to get reliable timing

Width Expansion

- Width expansion can be achieved by using “super clocking” with implementation similar to that for register file write expansion.
- To expand a 16-bit word to a 16 *n* bit word requires “super clocking” at *n* times the fundamental rate.

Width Expansion

- Implementation
  - For address-controlled READs, straight-forward
  - Challenging for WRITEs:
    - Must be trailing edges on, for example, $\overline{WE}$, for each of the super clock cycles
    - This will require changes on negative as well as positive super clock edges

Postscript

- The workarounds do not consider:
  - Multiple clock edge use instead of super-clocking
  - Different clock edges on the two ports on a dual port SelectRAM
- These techniques can potentially be beneficial to the degree that:
  - the resulting constructs are synthesizable, and
  - do not adversely affect performance

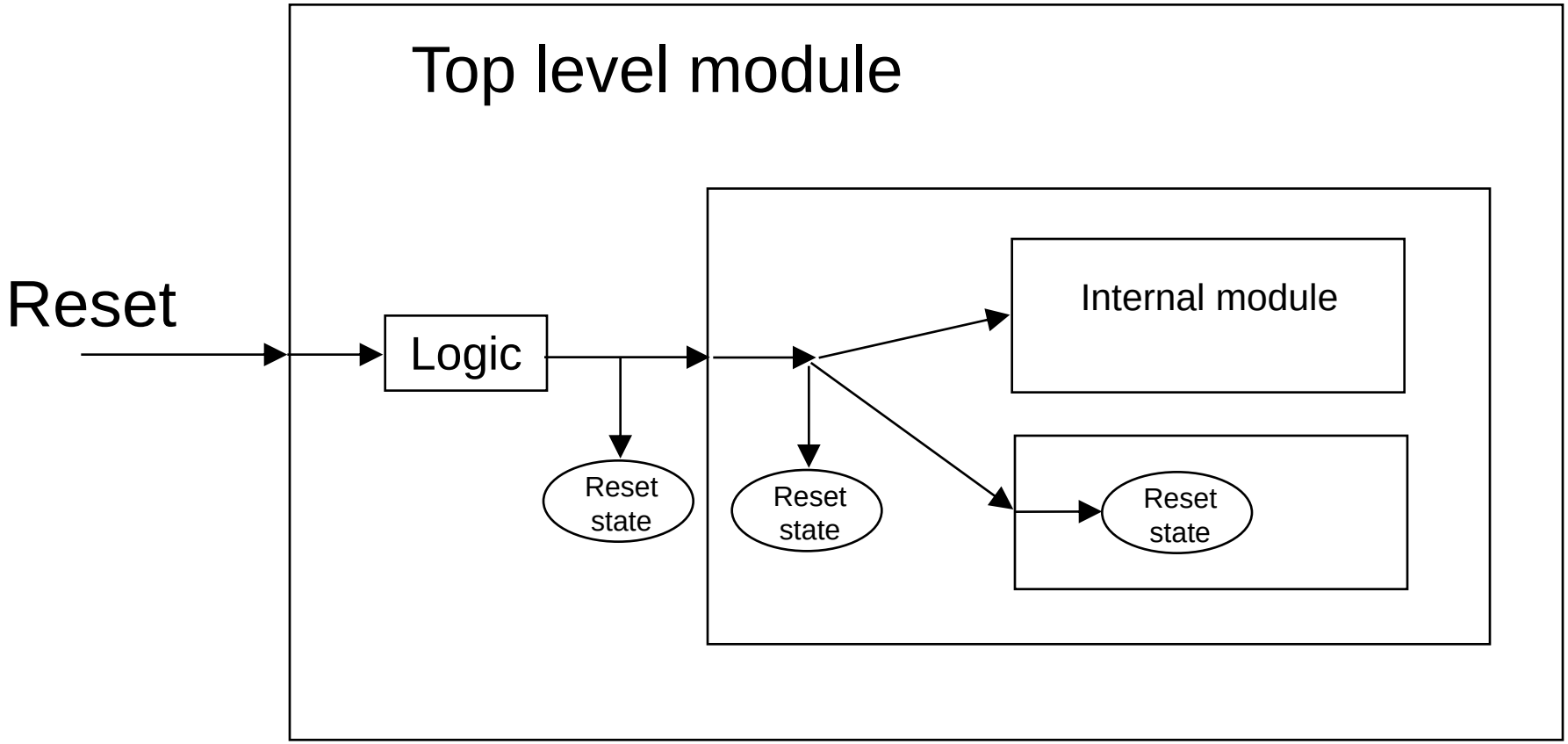
Clocking Issues

- Limits of hardware and synthesis tools constrain clock and reset signal use
- Known problems and their workarounds
  - Generating reset signals
  - Limitations of the global clock routing
  - Using enable signals instead of clocks

Reset Signals

- Generate reset signals only once, in top level module
- Pass signal down to lower level modules, but do not change it
- Both synchronous and asynchronous reset are OK, but be consistent.

Reset Signals (cont.)



Clocking Limitations

- Signals on the global clock routing cannot be used as inputs to combinational logic.
- Using a clock signal as a combinational input will pull that clock signal off the global routing, severely impacting performance.

Clocking Examples

- BAD:
  - Signal_1 <= Signal_2 & clk;
  - Gated_clk <= Signal & clk;
- Neither clk or gated_clk will be on the global clock routing
- GOOD:
  - Use enable signals instead of clocks
  - Don’t use clock gating

Enable Vs. Clock

- Use enables instead of clocks:
  - Reg enable;

```
always @ (posedge clk or negedge clk)
    begin
        enable <= ~enable;
    end
```

- Same waveform, but only uses clock as input to flip-flop – doesn't pull it off global clock routing.