

*RAM on FPGA

前言

超标量 CPU 中有很多地方都需要用到多端口的存储

前言

超标量 CPU 中有很多地方都需要用到多端口的存储

- 缓存
- 寄存器文件
- 发射队列
-

前言

超标量 CPU 中有很多地方都需要用到多端口的存储

有一些模块具有特殊性

e.g. 发射队列可以实现为多个并行的 FIFO

每个 FIFO 只有一个读和一个写 → 双端口 LUTRAM

前言

超标量 CPU 中有很多地方都需要用到多端口的存储

有一些模块具有特殊性

e.g. 发射队列可以实现为多个并行的 FIFO

每个 FIFO 只有一个读和一个写 → 双端口 LUTRAM

通用的多端口 RAM 复杂一点

可以分 bank 来提供多端口

前言

超标量 CPU 中有很多地方都需要用到多端口的存储

有一些模块具有特殊性

e.g. 发射队列可以实现为多个并行的 FIFO

每个 FIFO 只有一个读和一个写 → 双端口 LUTRAM

通用的多端口 RAM 复杂一点

可以分 bank 来提供多端口

- 空间利用率高，没有冗余的存储

前言

超标量 CPU 中有很多地方都需要用到多端口的存储

有一些模块具有特殊性

e.g. 发射队列可以实现为多个并行的 FIFO

每个 FIFO 只有一个读和一个写 → 双端口 LUTRAM

通用的多端口 RAM 复杂一点

可以分 bank 来提供多端口

- 空间利用率高，没有冗余的存储
- 遇到 bank 冲突时无法在一个周期内完成多个写入
 - 注意这不是两个写端口同时写入同一个地址
bank 冲突时完全可以是写入不同的地址，属于合法的操作

前言

超标量 CPU 中有很多地方都需要用到多端口的存储

有一些模块具有特殊性

e.g. 发射队列可以实现为多个并行的 FIFO

每个 FIFO 只有一个读和一个写 → 双端口 LUTRAM

通用的多端口 RAM 复杂一点

可以分 bank 来提供多端口

- 空间利用率高，没有冗余的存储
- 遇到 bank 冲突时无法在一个周期内完成多个写入
 - 注意这不是两个写端口同时写入同一个地址
bank 冲突时完全可以是写入不同的地址，属于合法的操作
 - 有很多减少 bank 冲突的策略

前言

超标量 CPU 中有很多地方都需要用到多端口的存储

有一些模块具有特殊性

e.g. 发射队列可以实现为多个并行的 FIFO

每个 FIFO 只有一个读和一个写 → 双端口 LUTRAM

通用的多端口 RAM 复杂一点

可以分 bank 来提供多端口

- 空间利用率高，没有冗余的存储
- 遇到 bank 冲突时无法在一个周期内完成多个写入
 - 注意这不是两个写端口同时写入同一个地址
 - bank 冲突时完全可以是写入不同的地址，属于合法的操作
 - 有很多减少 bank 冲突的策略
 - 不适合用来做寄存器文件

前言

超标量 CPU 中有很多地方都需要用到多端口的存储

有一些模块具有特殊性

e.g. 发射队列可以实现为多个并行的 FIFO

每个 FIFO 只有一个读和一个写 → 双端口 LUTRAM

通用的多端口 RAM 复杂一点

可以分 bank 来提供多端口

- 空间利用率高，没有冗余的存储
- 遇到 bank 冲突时无法在一个周期内完成多个写入
 - 注意这不是两个写端口同时写入同一个地址
 - bank 冲突时完全可以是写入不同的地址，属于合法的操作
 - 有很多减少 bank 冲突的策略
 - 不适合用来做寄存器文件
- 仲裁逻辑和交互可能很复杂，对 FPGA 不友好

假设

为了方便接下来的讨论，在本幻灯片中，我们假设

假设

为了方便接下来的讨论，在本幻灯片中，我们假设

- 写端口之间有优先级
 - 同时写入同一个地址不会导致未定义行为

假设

为了方便接下来的讨论，在本幻灯片中，我们假设

- 写端口之间有优先级
 - 同时写入同一个地址不会导致未定义行为
- 写入和读取都和时钟上升沿同步
 - 读数据都是寄存器输出

假设

为了方便接下来的讨论，在本幻灯片中，我们假设

- 写端口之间有优先级
 - 同时写入同一个地址不会导致未定义行为
- 写入和读取都和时钟上升沿同步
 - 读数据都是寄存器输出
- “写优先模式”
 - 如果读地址和写地址相同，将读到写入的数据

假设

为了方便接下来的讨论，在本幻灯片中，我们假设

- 写端口之间有优先级
 - 同时写入同一个地址不会导致未定义行为
- 写入和读取都和时钟上升沿同步
 - 读数据都是寄存器输出
- “写优先模式”
 - 如果读地址和写地址相同，将读到写入的数据
 - 需要转发逻辑

假设

为了方便接下来的讨论，在本幻灯片中，我们假设

- 写端口之间有优先级
 - 同时写入同一个地址不会导致未定义行为
- 写入和读取都和时钟上升沿同步
 - 读数据都是寄存器输出
- “写优先模式”
 - 如果读地址和写地址相同，将读到写入的数据
 - 需要转发逻辑
- 不支持单周期 full reset
 - 如果确实有需求，可以多周期完成

接口

```
module RAM #(
    parameter int WIDTH    = 64,
    parameter int DEPTH   = 32,
    parameter int N_WRITE = 4,
    parameter int N_READ  = 8,
    localparam int ADDR_WIDTH = $clog2(DEPTH),
    localparam type word_t = logic [WIDTH-1:0],
    localparam type addr_t = logic [ADDR_WIDTH-1:0]
) (
    input logic clk, resetn,
    input logic [N_WRITE-1:0] wen,
    input addr_t [N_WRITE-1:0] waddr,
    input word_t [N_WRITE-1:0] wdata,
    input addr_t [N_READ-1:0] raddr,
    output word_t [N_READ-1:0] rdata
);
```

接口

```
module RAM #(
    parameter int WIDTH    = 64,
    parameter int DEPTH   = 32,
    parameter int N_WRITE = 4,
    parameter int N_READ  = 8,
    localparam int ADDR_WIDTH = $clog2(DEPTH),
    localparam type word_t = logic [WIDTH-1:0],
    localparam type addr_t = logic [ADDR_WIDTH-1:0]
) (
    input logic clk, resetn,
    input logic [N_WRITE-1:0] wen,
    input addr_t [N_WRITE-1:0] waddr,
    input word_t [N_WRITE-1:0] wdata,
    input addr_t [N_READ-1:0] raddr,
    output word_t [N_READ-1:0] rdata
);
```

上面的四个参数将是之后综合测试的参数

转发： BypassBuffer

source/ooo/src/util/BypassBuffer.sv

```
word_t [N_READ-1:0] next;

always_comb begin
    next = data;

    for (int i = 0; i < N_WRITE; i++)
        for (int j = 0; j < N_READ; j++) begin
            if (wen[i] && raddr[j] == waddr[i])
                next[j] = wdata[i];
        end
    end

always_ff @(posedge clk)
if (resetn) begin
    rdata <= next;
end else begin
    rdata <= '0;
end
```

存储

Xilinx 的 FPGA 上提供了

存储

Xilinx 的 FPGA 上提供了

- 寄存器

存储

Xilinx 的 FPGA 上提供了

- 寄存器
- Distributed RAM (LUTRAM) : 一个端口支持读写，另一个只读
 - 允许组合读取

存储

Xilinx 的 FPGA 上提供了

- 寄存器
- Distributed RAM (LUTRAM) : 一个端口支持读写，另一个只读
 - 允许组合读取
- Block RAM (BRAM) : 两个端口读写
 - 读数据至少一个周期延时

存储

Xilinx 的 FPGA 上提供了

- 寄存器
- Distributed RAM (LUTRAM) : 一个端口支持读写，另一个只读
 - 允许组合读取
- Block RAM (BRAM) : 两个端口读写
 - 读数据至少一个周期延时

建议使用 xpm_memory_* 模块来初始化

XPM_Library_Guide_2020.2_HDL_Templates.zip

存储

Xilinx 的 FPGA 上提供了

- 寄存器
- Distributed RAM (LUTRAM) : 一个端口支持读写，另一个只读
 - 允许组合读取
- Block RAM (BRAM) : 两个端口读写
 - 读数据至少一个周期延时

建议使用 xpm_memory_* 模块来初始化

XPM_Library_Guide_2020.2_HDL_Templates.zip

本幻灯片中，我们只需要双端口 RAM 即可
一个端口支持写入，另一个支持读取，二者不共享地址线

Flip-Flop RAM

完全使用寄存器作为存储

source/ooo/src/util/FFRAM.sv

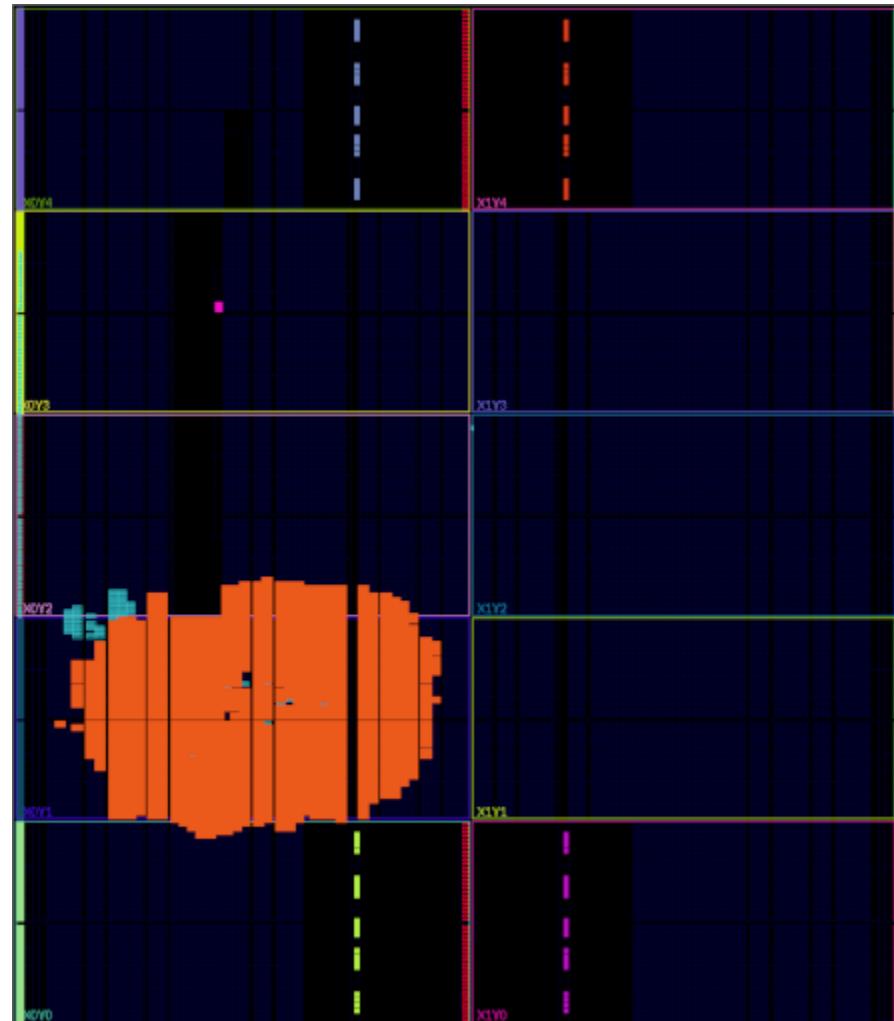
```
word_t [DEPTH-1:0] prev, next;

always_comb begin
    next = prev;

    for (int i = 0; i < N_WRITE; i++)
        if (wen[i]) begin
            for (int j = 0; j < DEPTH; j++) begin
                if (waddr[i] == addr_t'(j))
                    next[j] = wdata[i];
            end
        end
    end

always_ff @(posedge clk) begin
    prev <= next;
end
```

Flip-Flop RAM



频率: 146.84 MHz

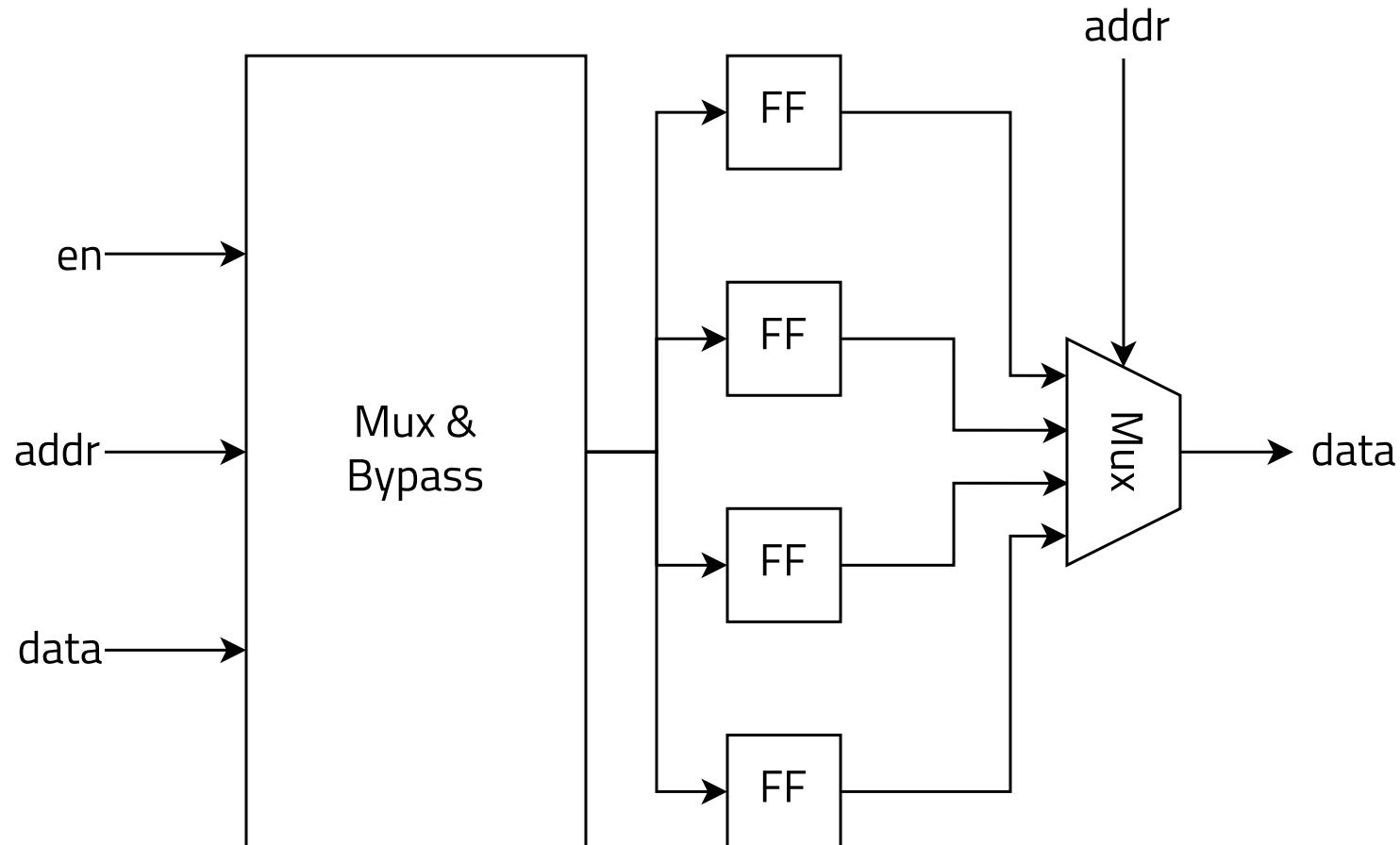
LUT / FF / Mux: 12754 / 2048 + 512 / 843

Flip-Flop RAM

虽然本身频率不低，但是浪费了大量 LUT 和 Mux 资源

Flip-Flop RAM

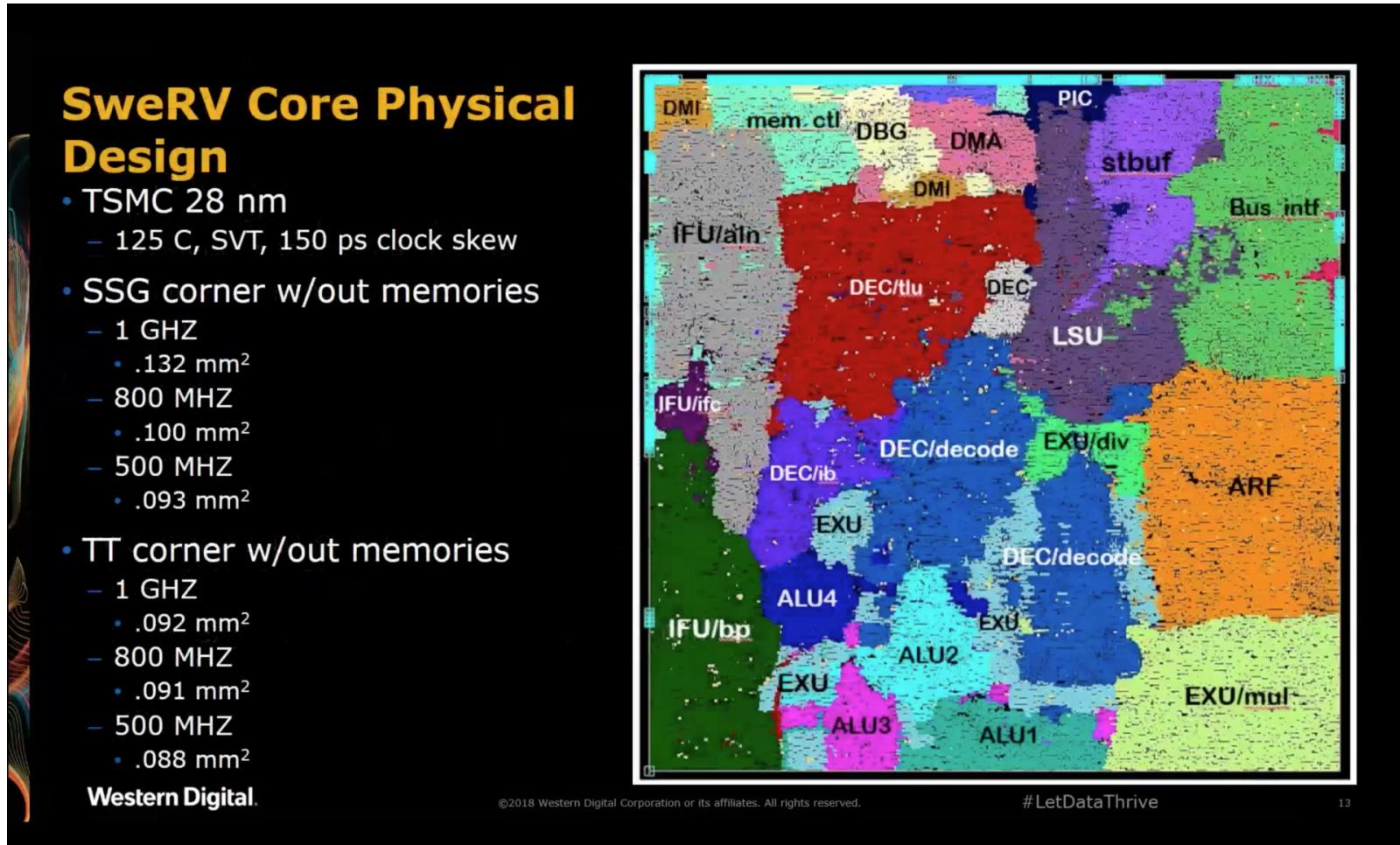
虽然本身频率不低，但是浪费了大量 LUT 和 Mux 资源



其中大量资源浪费在寻址和优先级转发电路上

Flip-Flop RAM

虽然本身频率不低，但是浪费了大量 LUT 和 Mux 资源

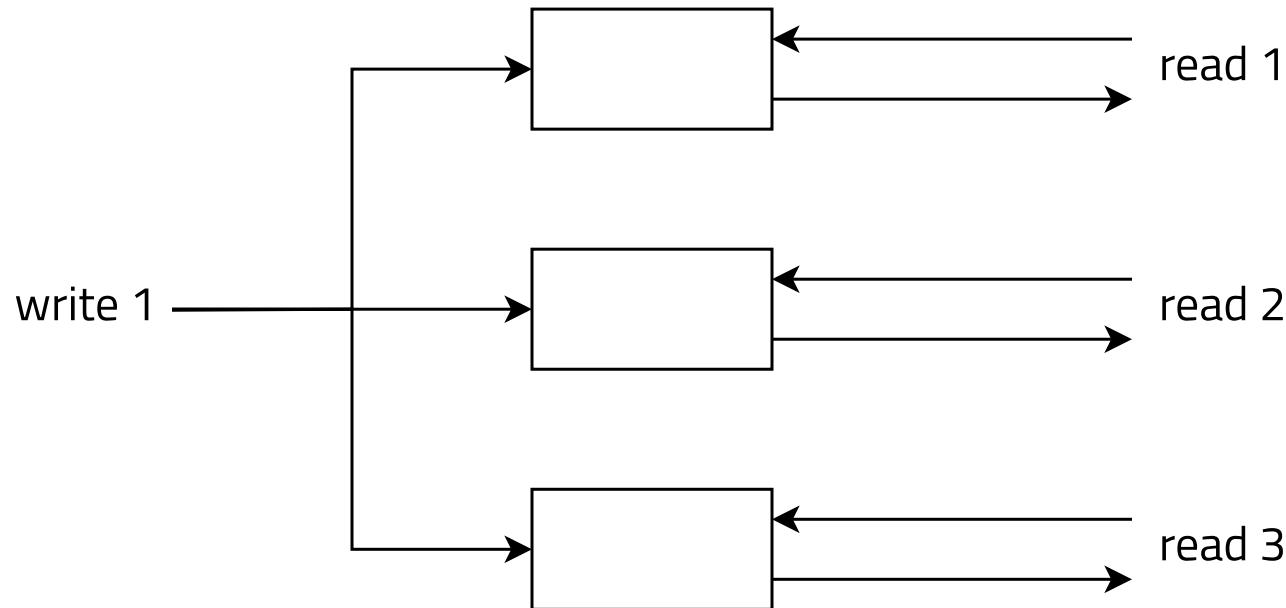


寄存器文件：ARF，2个写端口，4个读端口

只占到总面积的 9%，问题不大

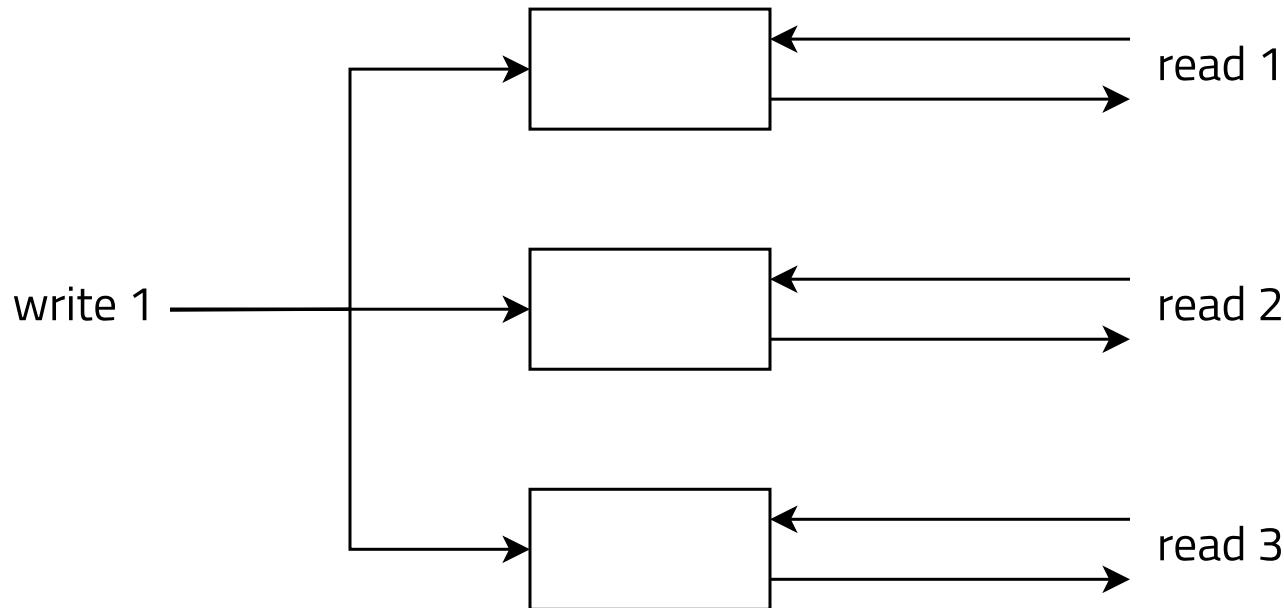
Single-Write Multiple-Read RAM

除了分 bank，还可以做 replica 来提供多端口



Single-Write Multiple-Read RAM

除了分 bank，还可以做 replica 来提供多端口

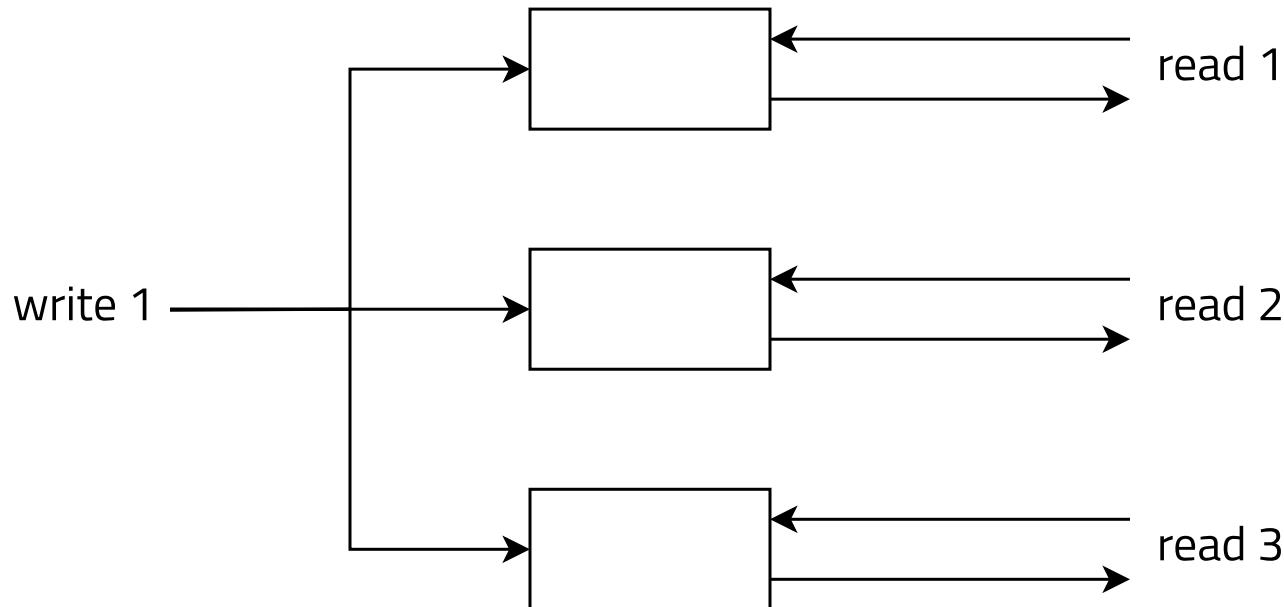


硬件一般还是比较靠谱的，可以认为写入不会出错

不用担心多个 replica 之间不一致的问题

Single-Write Multiple-Read RAM

除了分 bank，还可以做 replica 来提供多端口



硬件一般还是比较靠谱的，可以认为写入不会出错

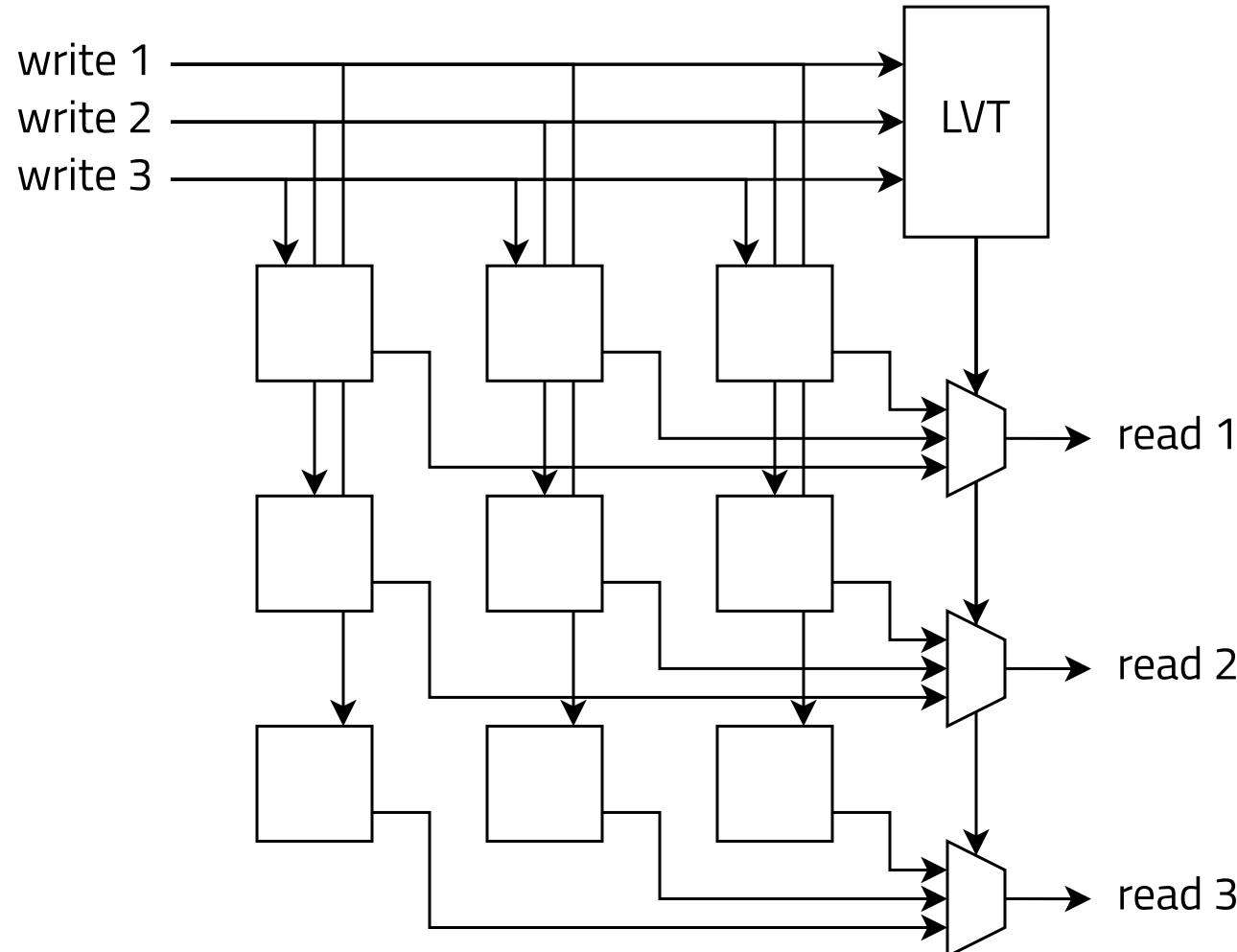
不用担心多个 replica 之间不一致的问题

增强容错：ECC

Live Value Table

每个写端口都有自己的 replica

使用 live value table 记录哪个 replica 是最新的



Live Value Table

source/ooo/src/util/LVTRAM.sv

```
for (genvar i = 0; i < N_READ; i++) begin: row
    for (genvar j = 0; j < N_WRITE; j++) begin: column
        /* verilator lint_off PINMISSING */
        DistributedRAM #(.WIDTH, .DEPTH) ram_inst (
            .clk,
            .wea(wen[j]),
            .addra(waddr[j]),
            .dina(wdata[j]),
            .addrb(raddr[i]),
            .doutb(data[i][j])
        );
        /* verilator lint_on PINMISSING */
    end
end
```

Live Value Table

source/ooo/src/util/LVTRAM.sv

```
typedef logic [ID_WIDTH-1:0] id_t;

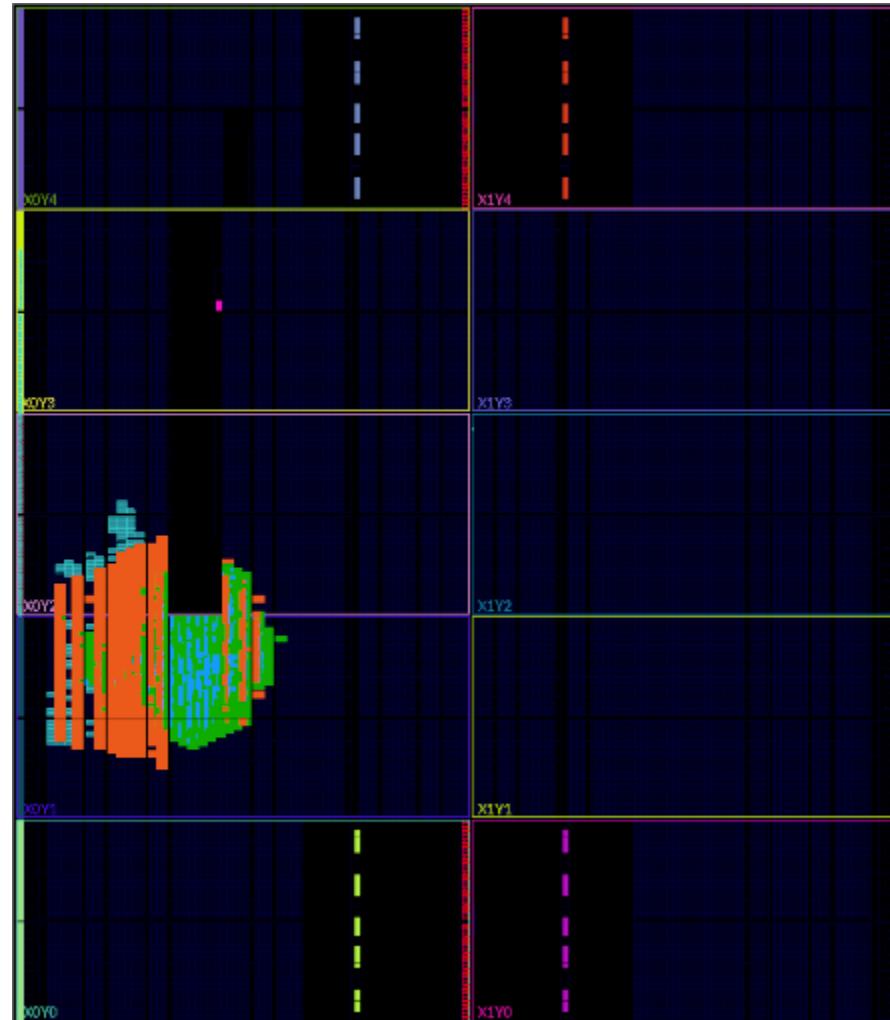
id_t [N_WRITE-1:0] wid;
id_t [N_READ-1:0] rid;

for (genvar i = 0; i < N_WRITE; i++) begin
    assign wid[i] = id_t'(i);
end

FFRAM #(
    .WIDTH(ID_WIDTH),
    .DEPTH, .N_WRITE, .N_READ,
    .LATENCY(0)
) lvt_inst (
    .clk, .resetn(1),
    .wen, .waddr, .wdata(wid),
    .raddr, .rdata(rid)
);

for (genvar i = 0; i < N_READ; i++) begin
    assign out[i] = data[i][rid[i]];
end
```

Live Value Table



频率: 133.32 MHz

LUT / FF / Mux: 4684 / 64 + 512 / 128

XOR-Based RAM

LVT 依然是一个小的多端口 RAM，通常就是 FF RAM

XOR-Based RAM

LVT 依然是一个小的多端口 RAM，通常就是 FF RAM

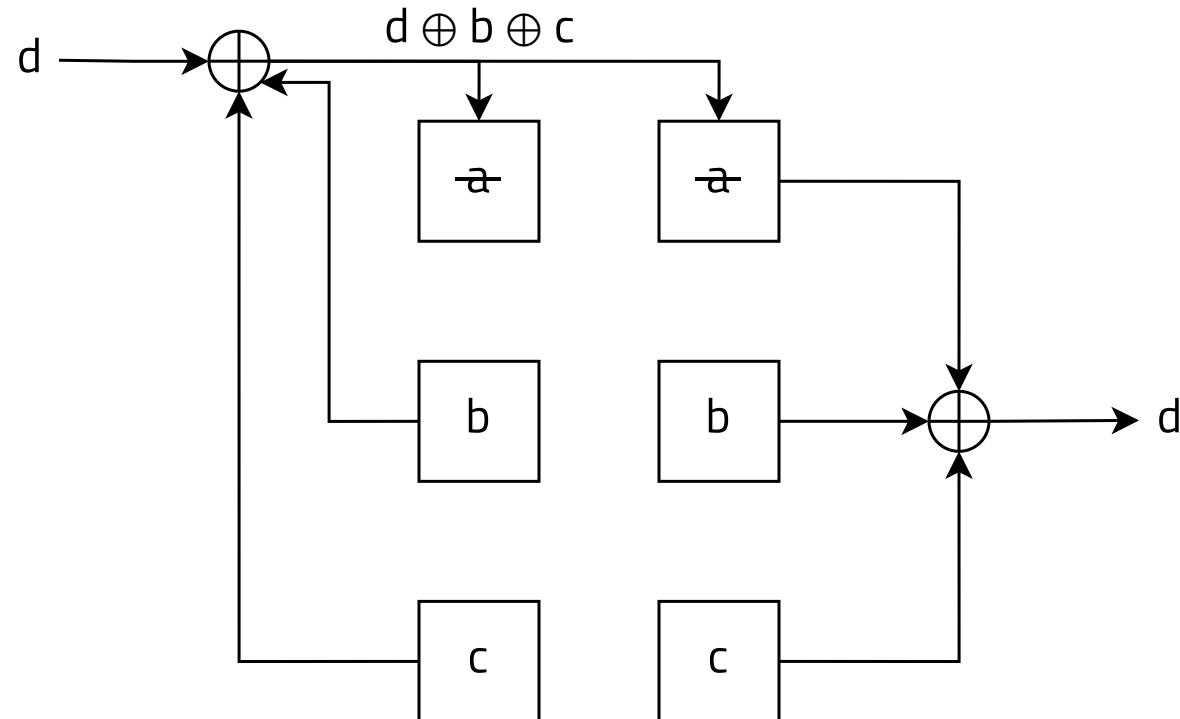
XOR-based RAM 完全消除了对 FF RAM 的依赖

XOR-Based RAM

关键点：利用 $a \oplus a = 0$ 的特性

写入时异或上其它 replica 的值

输出时再异或一下，从而消除其它 replica 的影响

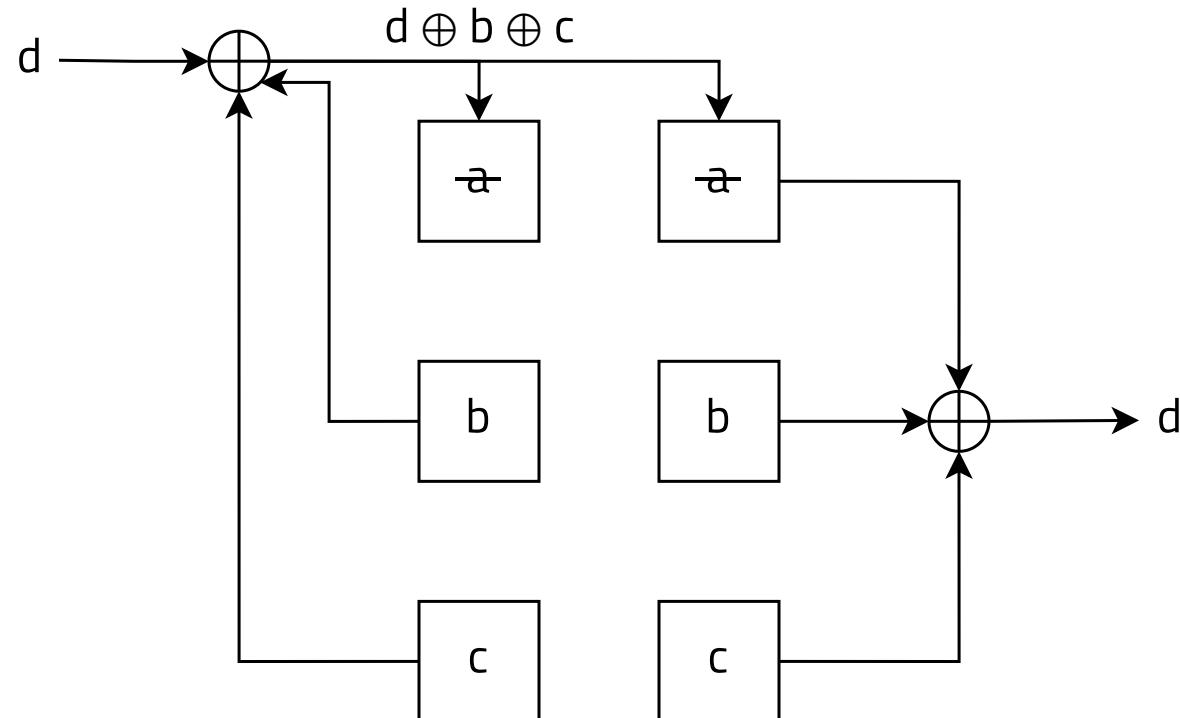


XOR-Based RAM

关键点：利用 $a \oplus a = 0$ 的特性

写入时异或上其它 replica 的值

输出时再异或一下，从而消除其它 replica 的影响



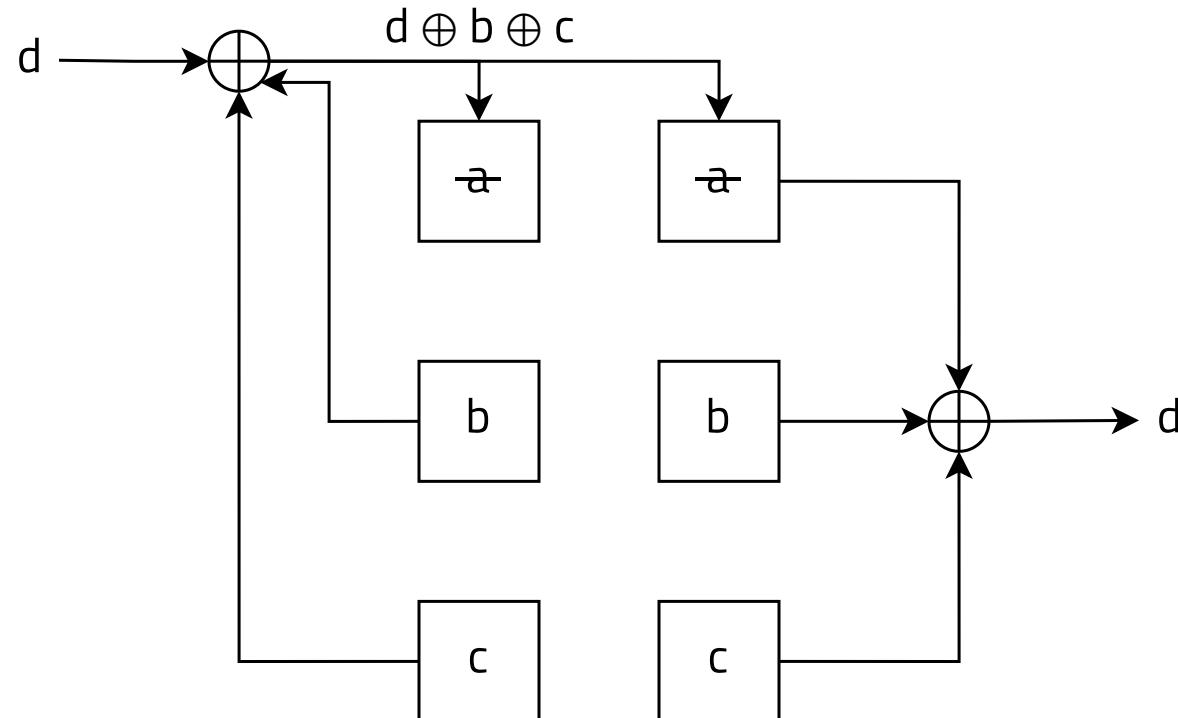
当多个写端口同时写入同一地址时，需要禁用优先级低的使能

XOR-Based RAM

关键点：利用 $a \oplus a = 0$ 的特性

写入时异或上其它 replica 的值

输出时再异或一下，从而消除其它 replica 的影响



当多个写端口同时写入同一地址时，需要禁用优先级低的使能
需要用到更多的 replica

XOR-Based RAM

source/ooo/src/util/XORRAM.sv

```
logic [N_WRITE-1:0] en;

for (genvar i = 0; i < N_WRITE; i++) begin
    always_comb begin
        en[i] = wen[i];
        for (int j = i + 1; j < N_WRITE; j++) begin
            en[i] &= !wen[j] || waddr[i] != waddr[j];
        end
    end
end
end
```

XOR-Based RAM

source/ooo/src/util/XORRAM.sv

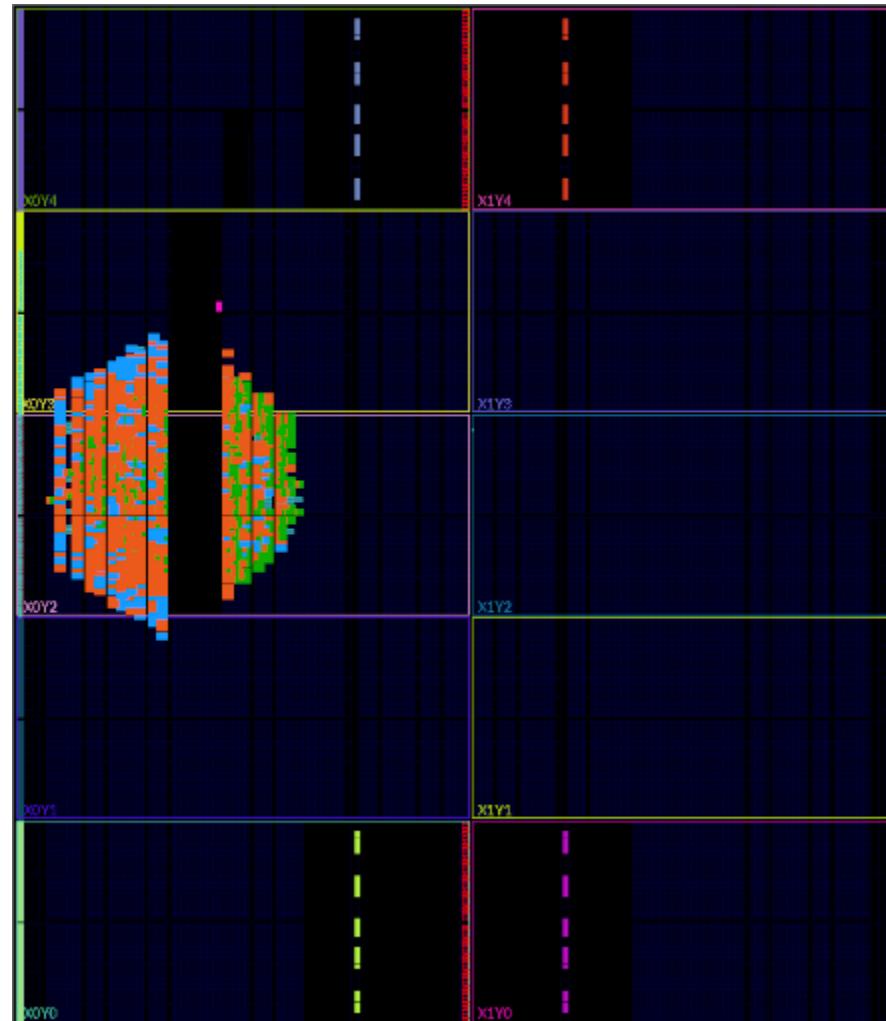
```
for (genvar i = 0; i < N_WRITE; i++) begin: w_row
    word_t [N_WRITE-1:0] vec;
    for (genvar j = 0; j < N_WRITE; j++) begin: w_column
        if (i == j) begin: self
            assign vec[i] = wdata[i];
        end else begin: others
            /* verilator lint_off PINMISSING */
            DistributedRAM #( .WIDTH, .DEPTH ) ram_inst (
                .clk,
                .wea(en[j]),
                .addrA(waddr[j]),
                .dina(sum[j]),
                .addrB(waddr[i]),
                .doutb(vec[j])
            );
            /* verilator lint_on PINMISSING */
        end
    end
    // sum[i] = vec[0] ⊕ vec[1] ⊕ ... ⊕ vec[N_WRITE-1]
end
```

XOR-Based RAM

source/ooo/src/util/XORRAM.sv

```
for (genvar i = 0; i < N_READ; i++) begin: r_row
    word_t [N_WRITE-1:0] vec;
    for (genvar j = 0; j < N_WRITE; j++) begin: r_column
        /* verilator lint_off PINMISSING */
        DistributedRAM #(.WIDTH, .DEPTH) ram_inst (
            .clk,
            .wea(en[j]),
            .addrA(waddr[j]),
            .dina(sum[j]),
            .addrB(raddr[i]),
            .doutb(vec[j])
        );
        /* verilator lint_on PINMISSING */
    end
    // out[i] = vec[0] ⊕ vec[1] ⊕ ... ⊕ vec[N_WRITE-1]
end
```

XOR-Based RAM



频率: 174.00 MHz

LUT / FF / Mux: 4874 / 0 + 512 / 64

更多对比

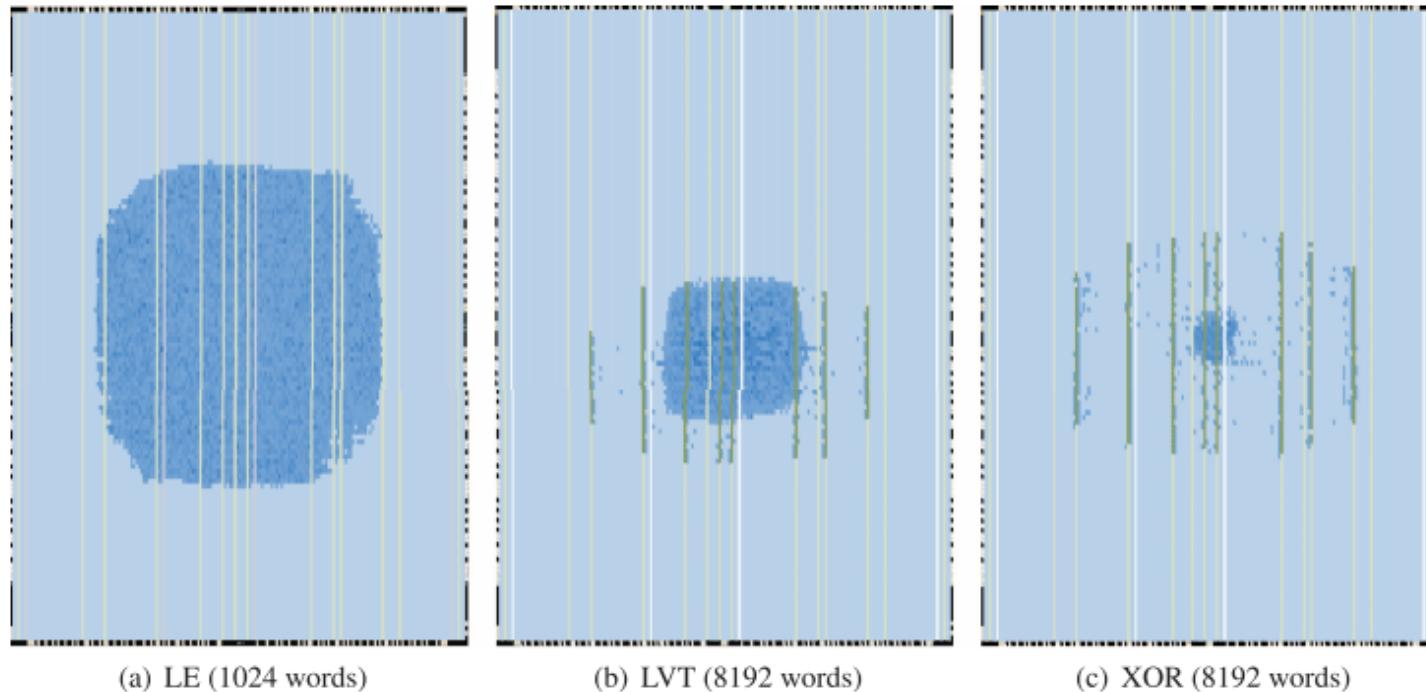


Fig. 5. Resource layout of (a) a 1024-deep 2W/4R pure LE memory and 8192-deep 2W/4R memories for (b) LVT, and (c) XOR designs, as rendered by Quartus. The thin columns represent BRAMs or DSPs (darkened indicates in-use), the dots and dot-clouds point out ALMs.

这个测试里面用的是 BRAM

可以看到 BRAM 在板上的分布对 RAM 布线的影响

更多对比

Depth	Design that minimizes:		
	Delay	Slices	BRAMs
2	LE	LVT	LVT
4	LE	LVT	LVT
8	LE	LVT/XOR	LVT
16	LE	XOR	LVT
32	LVT	XOR	LVT
64	LE	XOR	LVT
128	LVT	XOR	LVT
256	LVT	XOR	LVT
512	LVT	XOR	LVT
1K	LVT/XOR	XOR	LVT
2K	XOR	XOR	LVT
4K	XOR	XOR	LVT
8K	XOR	XOR	LVT

(a) 2W/4R

Depth	Design that minimizes:		
	Delay	Slices	BRAMs
4	LE	LE	LVT
8	LE	LE	LVT
16	LE/LVT	XOR	LVT
32	LVT	XOR	LVT
64	LVT	XOR	LVT
128	LVT	XOR	LVT
256	LVT	XOR	LVT
512	XOR	XOR	LVT
1K	XOR	XOR	LVT

(b) 4W/8R

Depth	Design that minimizes:		
	Delay	Slices	BRAMs
8	LE	XOR	LVT
16	LE	XOR	LVT
32	LVT	XOR	LVT
64	LVT	XOR	LVT
128	LVT	XOR	LVT
256	XOR	XOR	LVT
512	XOR	XOR	LVT
1K	XOR	XOR	LVT
2K	XOR	XOR	LVT

(c) 8W/16R

Fig. 9. Xilinx design space navigation: for each memory depth, we list the design that minimizes delay (i.e., has the highest F_{max}), the number of slices used, or the number of BRAMs used, for (a) 2W/4R; (b) 4W/8R; (c) 8W/16R memories. We consider results within 5% as equal and list multiple designs in such cases.

“LE” 指 “Logic Elements”，就是我们的 FFRAM

更多对比

Depth	Design that minimizes:		
	Delay	Slices	BRAMs
2	LE	LVT	LVT
4	LE	LVT	LVT
8	LE	LVT/XOR	LVT
16	LE	XOR	LVT
32	LVT	XOR	LVT
64	LE	XOR	LVT
128	LVT	XOR	LVT
256	LVT	XOR	LVT
512	LVT	XOR	LVT
1K	LVT/XOR	XOR	LVT
2K	XOR	XOR	LVT
4K	XOR	XOR	LVT
8K	XOR	XOR	LVT

(a) 2W/4R

Depth	Design that minimizes:		
	Delay	Slices	BRAMs
4	LE	LE	LVT
8	LE	LE	LVT
16	LE/LVT	XOR	LVT
32	LVT	XOR	LVT
64	LVT	XOR	LVT
128	LVT	XOR	LVT
256	LVT	XOR	LVT
512	XOR	XOR	LVT
1K	XOR	XOR	LVT

(b) 4W/8R

Depth	Design that minimizes:		
	Delay	Slices	BRAMs
8	LE	XOR	LVT
16	LE	XOR	LVT
32	LVT	XOR	LVT
64	LVT	XOR	LVT
128	LVT	XOR	LVT
256	XOR	XOR	LVT
512	XOR	XOR	LVT
1K	XOR	XOR	LVT
2K	XOR	XOR	LVT

(c) 8W/16R

Fig. 9. Xilinx design space navigation: for each memory depth, we list the design that minimizes delay (i.e., has the highest F_{max}), the number of slices used, or the number of BRAMs used, for (a) 2W/4R; (b) 4W/8R; (c) 8W/16R memories. We consider results within 5% as equal and list multiple designs in such cases.

“LE” 指 “Logic Elements”，就是我们的 FFRAM

可以看到并不是所有情况下都是 XOR-based RAM 最优

实践出真知 😊

更多对比：RSD

MULTIPORT-RAM-BASED COMPONENTS IN THE RSD.

	Component	RAM type
Rename Unit	PRF Free List	Banked RAM
	RMT	I-LVT
Dispatch Unit	IQ Free List	Banked RAM
	Ready Bit Table	I-LVT
	WAT	I-LVT
	Destination RAM	I-LVT
Issue Queue	Payload RAM	I-LVT
Reorder Buffer	ROB Meta Data	Banked RAM
	ROB Exec. State	I-LVT
Physical Register	PRF	I-LVT
Load-Store Unit	SDA	I-LVT

FPGA RESOURCES AND OPERATING FREQUENCY OF THE RSD WITH DIFFERENT MULTIPORT RAM IMPLEMENTATIONS.

	FF+Logic	I-LVT+Bank
LUTs	28166	15379 (54.6%)
Registers	13511	8584 (63.5%)
Freq. [MHz]	89.8	95.3 (106.1%)

参考资料

- “*Building Multiport Memories with Block RAMs*”
<https://tomverbeure.github.io/2019/08/03/Multiport-Memories.html>
- “*Composing Multi-Ported Memories on FPGAs*”
http://people.csail.mit.edu/ml/pubs/trets_multiport.pdf
- “*An Open Source FPGA-Optimized OoO RISC-V Soft Processor*”
<http://sv.rsg.ci.i.u-tokyo.ac.jp/pdfs/Mashimo-FPT'19.pdf>
<https://github.com/rsd-devel/rsd>
- “*UltraScale Architecture Libraries Guide*”
UG974 (v2020.2) December 4, 2020
<https://www.xilinx.com/support.html>