A Case for Standard-Cell Based RAMs in Highly-Ported Superscalar Processor Structures

Sungkwan Ku, Elliott Forbes, Rangeen Basu Roy Chowdhury, Eric Rotenberg

NC STATE UNIVERSITY

This work is supported by NSF grant CCF-1218608. Any opinions, findings, and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the NSF.

BACKGROUND

Superscalar core has many highly ported memories

- Ports increase with Instruction fetch width and number of parallel execution lanes
- Many ports required for physical register file, reorder buffer, scheduler, and rename tables

Challenges for full-custom designs using multi-ported versions of 6T SRAM bitcell

- Design effort is high to reliably handle PVT variations and narrow noise margins at low voltages for sub-micron technology
- Limited port memory compilers are available for sub-micron technology

Microarchitectural alternatives

- Replication or banking of fewer ported memories
- Efficiency or performance drawbacks

MOTIVATION

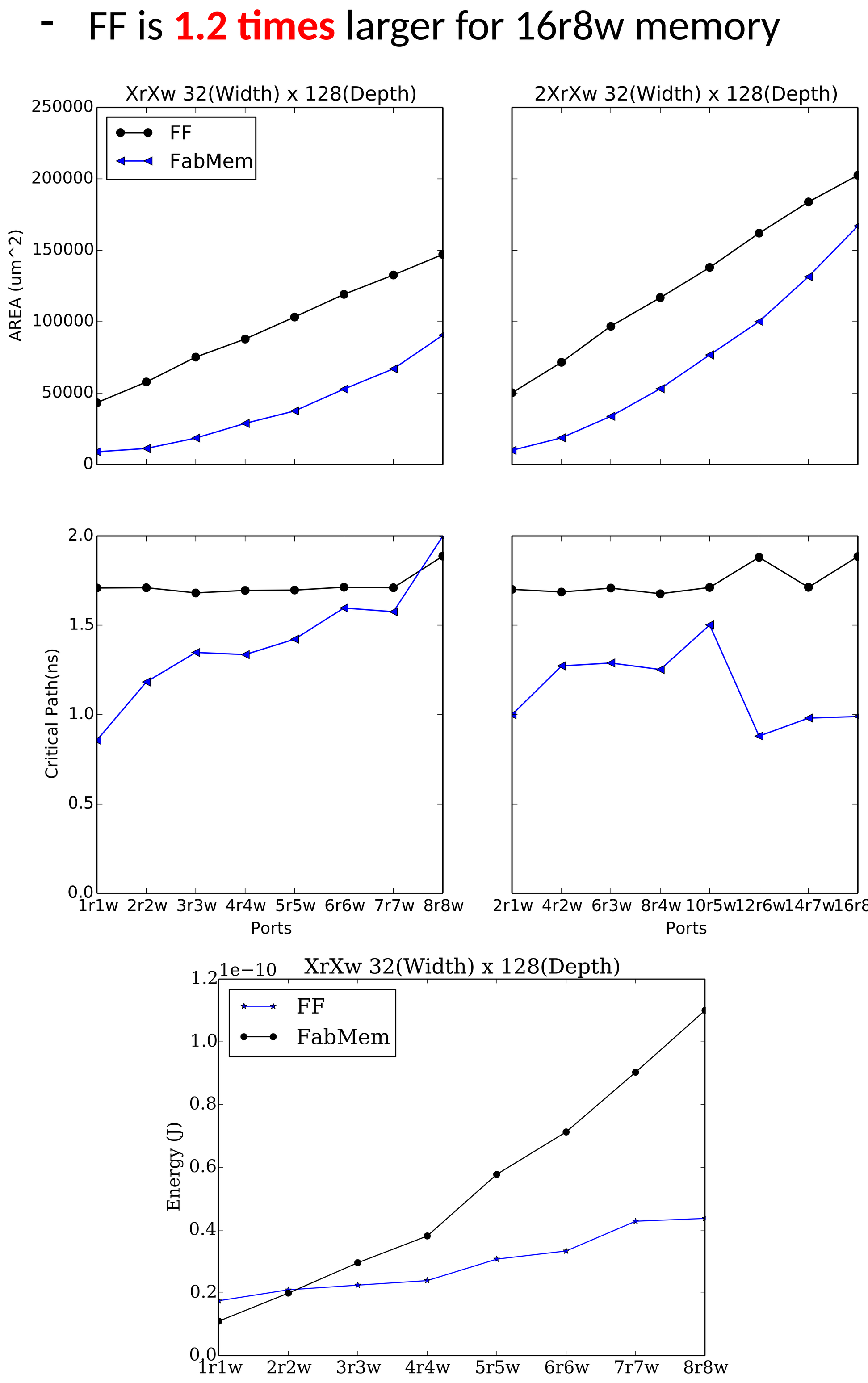
Study standard-cell based RAMs

- Traditionally, standard-cell based RAMs are used for small memories
- 24 transistors (4.522 um^2) per one flip-flop vs. 8 transistors (1.78 um^2) per 1r1w bitcell

Standard cell 45nm RAM (synthesis report) vs. FabMem 45nm SRAM (estimation tool report)

Area of FF:

- FF is 4.8 times larger for 1r1w memory
- FF is 1.62 times larger for 8r8w memory
- FF is **1.2 times** larger for 16r8w memory



Challenges in synthesized flip-flop based RAMs

- Placing and routing a large number of standard cells using automatic place-and-route tools results in a wide variation in area utilization
- Large increase in routing congestion with more rows and ports

CONTRIBUTION

Standard-cell based RAM compiler:

1. Per-row clock gating reduces both clock power and switching inside the D flip-flops.
2. A new tri-state based mux cell is added to the standard cell library. Employing it reduces total routing and further reduces switching inside the D flip-flops. The new mux cell presents the memory compiler with another choice for optimizing timing and power.
3. A modular layout strategy reduces total routing. A layout is generated for a smaller building block. A block's layout is made efficient by careful floorplanning. The modular layout allows stacking multiple blocks in series to compose the overall memory.

MODULAR SRAM COMPILER

Step1. Gate-level netlist generator

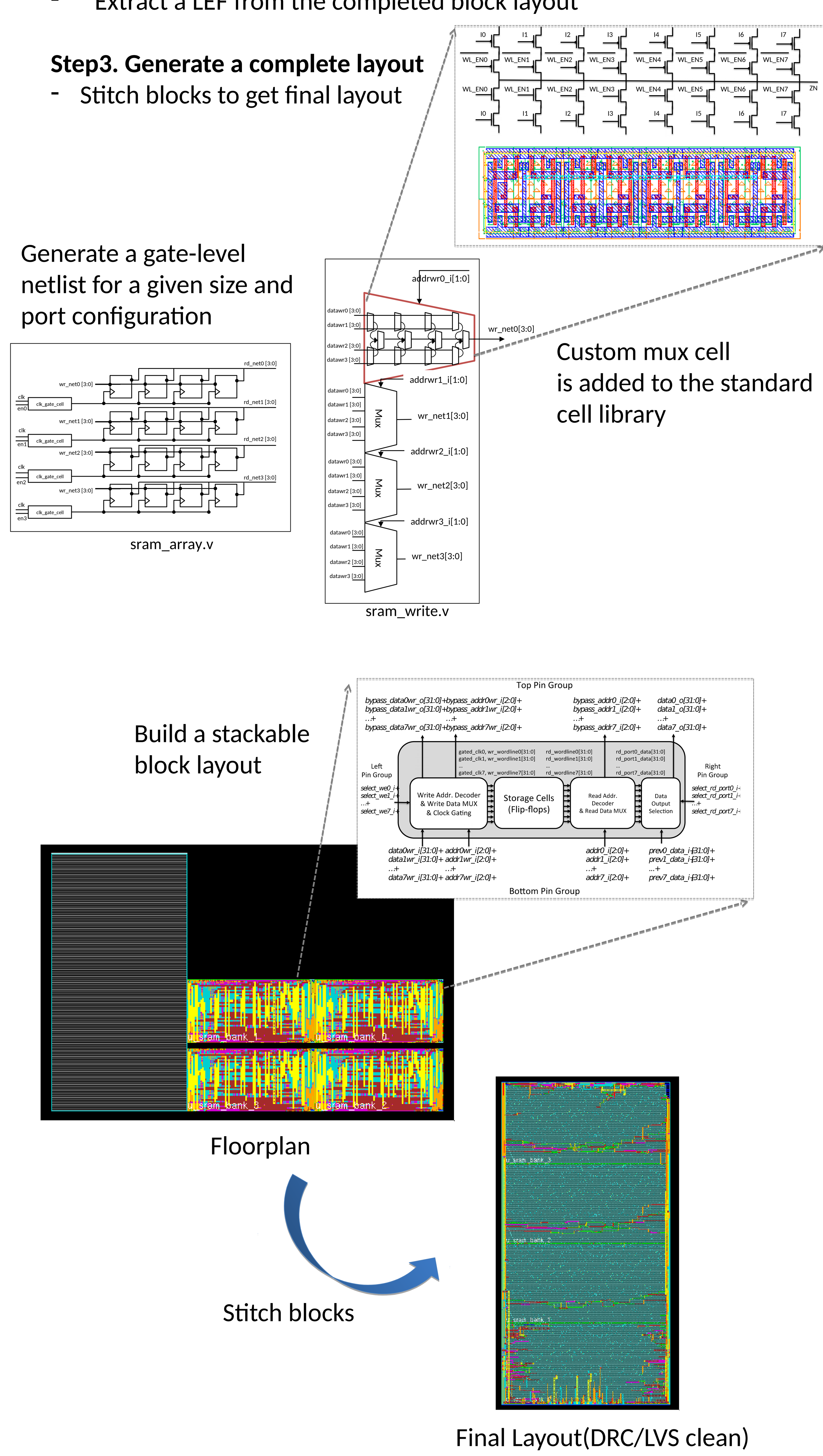
- For a given size and port configuration, gate-level netlist is generated
- Can select mux type (standard cell mux/custom cell mux), clock gating enable/disable, and the number of partitions

Step2. Build a stackable block layout

- If the number of partitions is defined, complete the stackable block layout first
- Extract a LEF from the completed block layout

Step3. Generate a complete layout

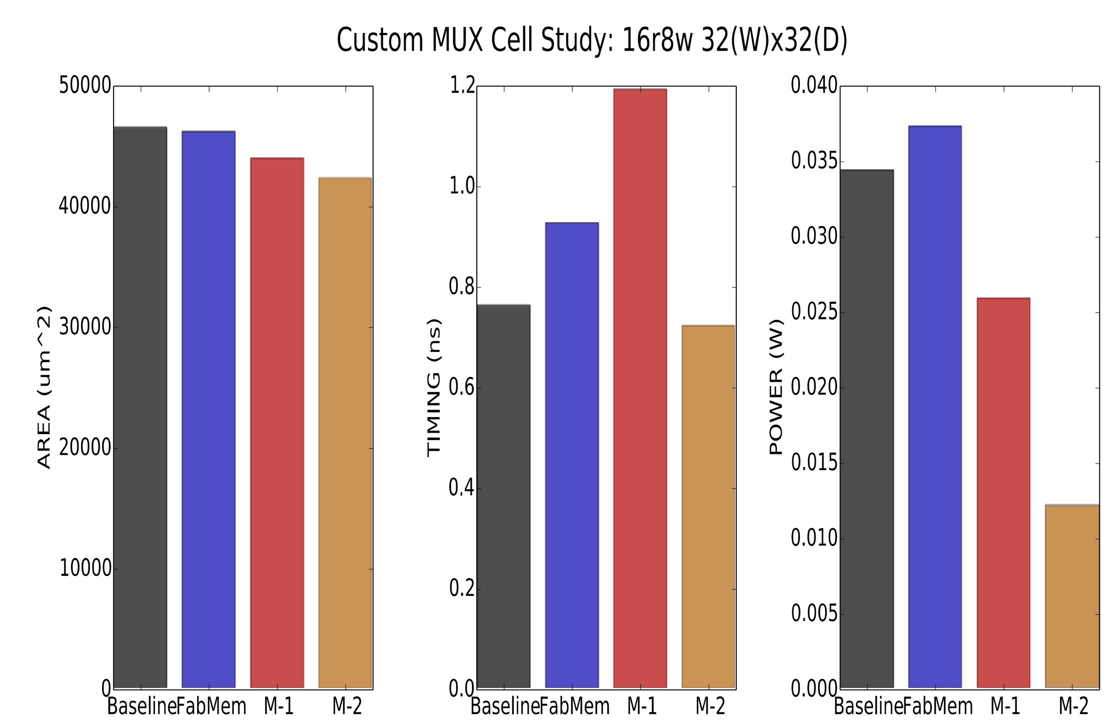
- Stitch blocks to get final layout



SENSITIVITY STUDY

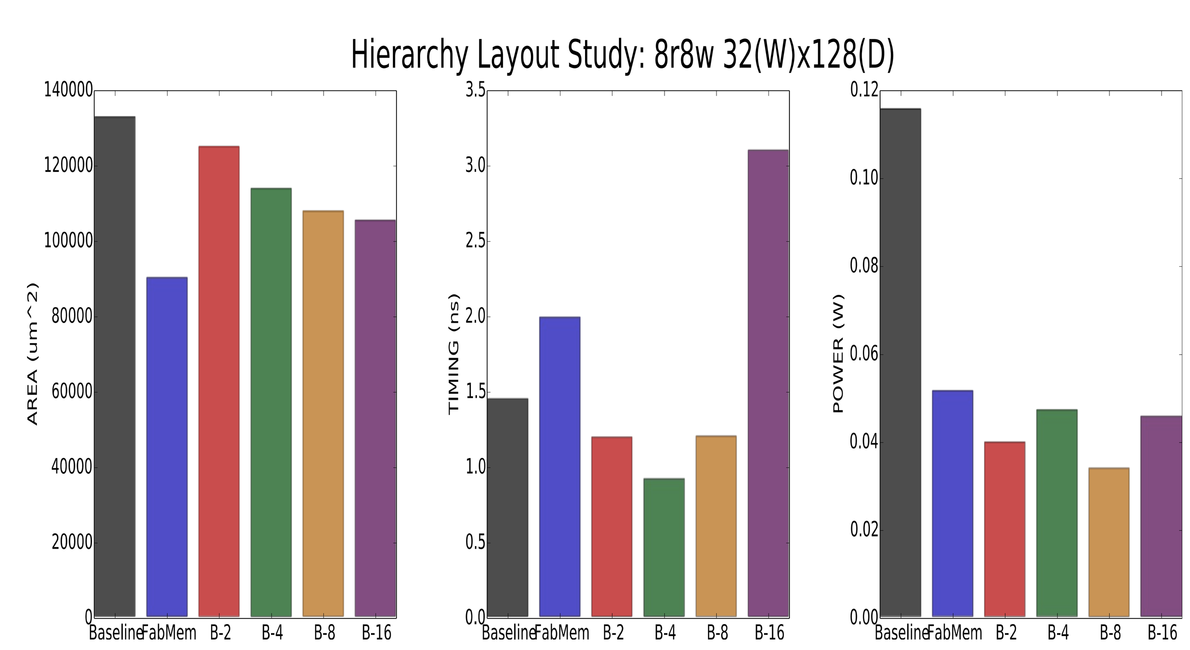
Effect of per-row clock gating and new mux cells (one partition)

- Baseline: synthesis and place-and-route
- M-1: baseline + per-row clock gating
- M-2: baseline + per-row clock gating + new mux cells
- FabMem: SRAM from FabMem



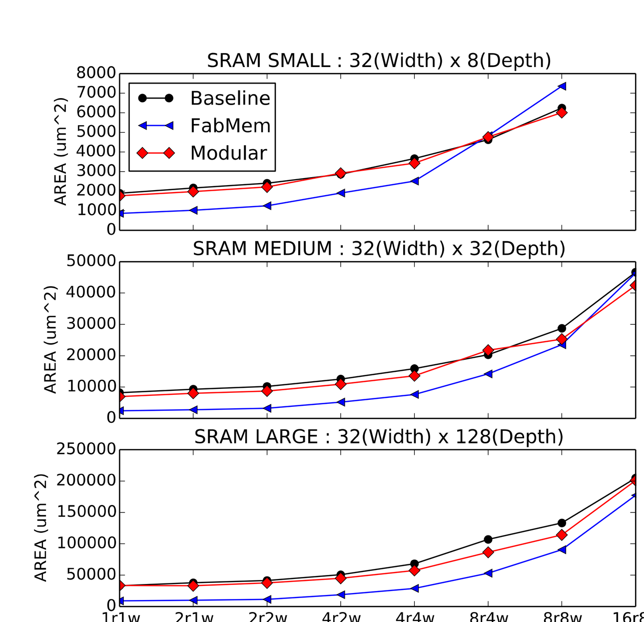
Effect of number of partitions for a fixed size

- B-2, B-4, B-8, and B-16, refer to number of blocks: 2, 4, 8, and 16 blocks
- Per-row clock gating enabled
- New mux cell *not* applied

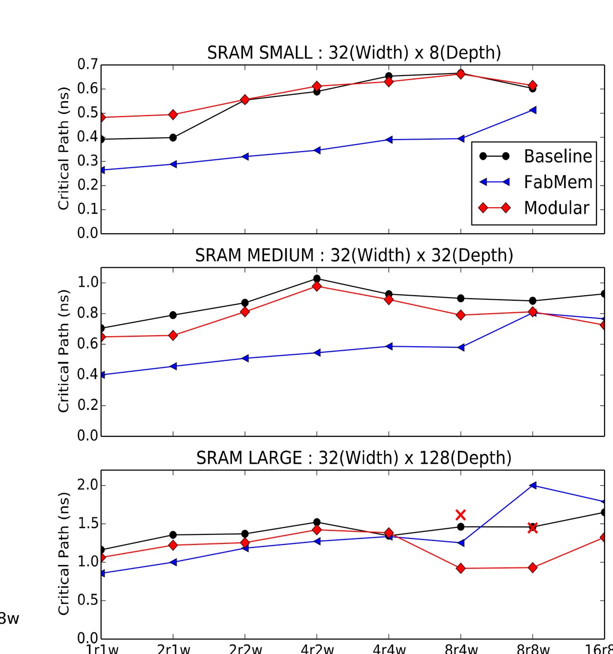


PRELIMINARY RESULTS

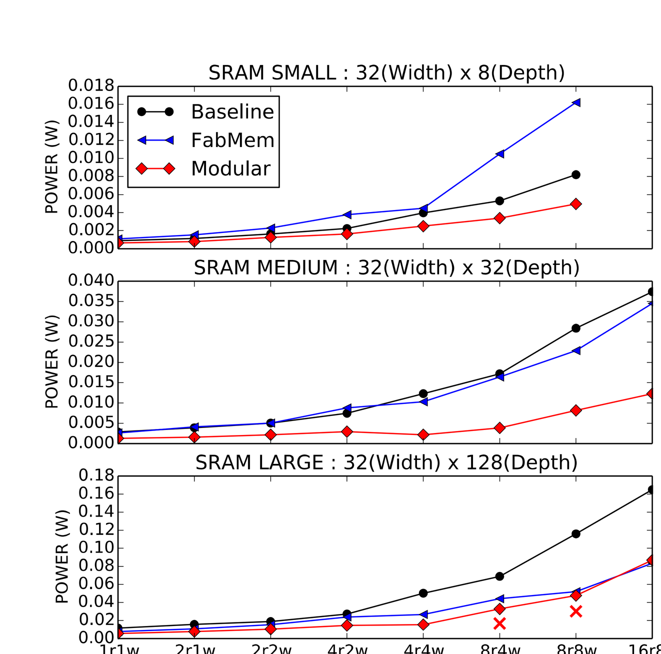
Baseline: synthesis and place-and-route
Modular: best configuration from our RAM compiler
FabMem: SRAM from FabMem



Area of *Modular*:
- Comparable or slightly better than *Baseline*
- Area overhead w.r.t. *FabMem* decreases with more ports



Timing of *Modular*:
- Better than *Baseline* for most cases
- Better than *FabMem* for large size and many ports



Power of *Modular*:
- Better than *Baseline* due to clock gating, tri-state MUXes, and minimized routing (modular layout).
- Better than *FabMem*