OPENRAM

AN OPEN-SOURCE MEMORY COMPILER

VLSI-DA at University of California Santa Cruz
VLSIARCH at Oklahoma State University

Contributors

- Presenters
  - Prof. Matthew R. Guthaus (mrg@ucsc.edu)
  - Samira Ataei (ataei@ostatemail.okstate.edu)
- Authors
  - Prof. Matthew R. Guthaus, Prof. James E. Stine, Samira Ataei, Brian Chen, Bin Wu, Mehedi Sarwar
- Other student contributors
  - Jeff Butera, Tom Golubev, Seokjoong Kim, Matthew Gaalswyk, and Son Bui

Outline

- **Background on Memory Compilers**
- OpenRAM Features
- OpenRAM Architecture and Circuits
- OpenRAM Usage
- OpenRAM Development
- How to port OpenRAM to a New Technology
- How to change the OpenRAM Circuits/Modules
- Conclusion

Why an Open Source Memory Compiler is needed?

- SRAMs have become a standard component in SoC, ASIC, and microprocessor designs and play a significant role in overall system performance and costs.
- Regular structure of memories leads to automation, but developing this with multiple technologies and tool methodologies is challenging.
- Most academic ICs design are limited by the availability of memories.
- Custom memory design can be a tedious and time-consuming task and may not be the intended purpose of the research.
- The lack of a customizable compiler makes it difficult for researchers to prototype and verify circuits beyond a single row or column of memory cells.

Memory Compiler Options

Contemporary memory compilers usually allow customers to view front-end simulation and not back-end features unless with a licensing fee.

- **Globalfoundries:** offers front-end PDKs for free, but not back-end views.
- **Virage Logic:** provides a dashboard compiler that selects from a pre-designed configurations.
- **Faraday Technologies:** provides a black box design kit for UMC technologies.
- **Dolphin Technology:** offers closed-source compilers for TSMC, UMC, and IBM.
- **FabMem:** (NCSU research groups) is able to create small arrays, but it is highly dependent on the Cadence design tools.
- **Synopsys Generic Memory Compiler:** is not recommended for fabrication since the supported technologies are not real.

Dolphin Technology

SYNOPSYS®

OpenRAM Motivation

- We believe in OpenRAM
  - It is free and open-source
  - Helpful to community
  - Integrates into computer architecture and digital systems easily.
  - Allows researchers to modify and use for existing SRAM architectures and any memory design (even memresistors).
- We also believe in community
  - We want circuits and system research as well as EDA to prosper
  - Academic research in memory is important!

Outline

- Background on Memory Compilers
- **OpenRAM Features**
- OpenRAM Architecture and Circuits
- OpenRAM Usage
- OpenRAM Development
- How to port OpenRAM to a New Technology
- How to change the OpenRAM Circuits/Modules
- Conclusion

OpenRAM Features

- OpenRAM is implemented in the Python programming language.
- OpenRAM provides reference circuit and physical implementations in a non-fabricable generic 45nm technology (FreePDK45) and fabricable Scalable CMOS (SCMOS) by non-confidential MOSIS foundry services.
- OpenRAM includes a characterizer to generate the timing/power results in a Liberty (lib) file.
- OpenRAM generates GDSII layout data, SPICE netlist, Verilog model, DRC/LVS verification reports and .lef file for place and route.
- OpenRAM is independent of any specific commercial tool.
- OpenRAM is completely user-modifiable since all source code is open source.

What is Python?

- Object-oriented rapid prototyping language
  - Not just a scripting language
  - Not just another Perl
- Rich set of libraries
  - numpy, SciPy, etc.
- Easy to learn, read, and use
- Extensible (add new modules)
  - C/C++/Fortran/whatever
  - Java (through Jython)
- Embeddable in applications

Engineering properties of Python

- Open Source (OSI Certified)
  - Copyrighted but use not restricted
  - No "viral" license
  - Owned by independent non-profit, PSF
- Mature (10+ years old)
- Supportive user community
  - Plenty of good books and references
- Simple design and *easy* to learn
  - Reads like “pseudo-code”
  - Suitable as first and last language

High-Level Properties

- Extremely portable
  - Unix/Linux, Windows, Mac, PalmOS, WindowsCE, RiscOS, VxWorks, QNX, OS/2, OS/390, AS/400, PlayStation, Sharp Zaurus, BeOS, VMS…
- Compiles to interpreted byte code
  - Compilation is implicit and automatic
- Memory management automatic
  - Reference counting for most situations
  - Garbage Collection added for cycle detection
- “Safe”: no core dumps due to your bugs

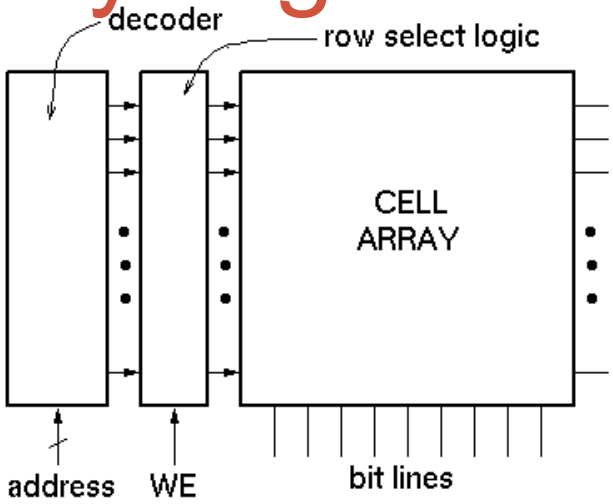
Language Properties

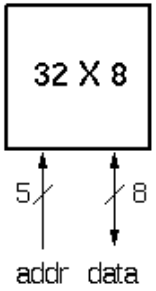
- Everything is an object
- Packages, modules, classes, functions
- Exception handling
- Dynamic typing, polymorphism
- Static scoping
- Operator overloading
- Indentation for block structure
  - Otherwise conventional syntax

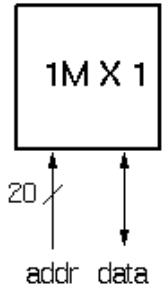
Outline

- Background on Memory Compilers
- OpenRAM Features
- **OpenRAM Architecture and Circuits**
- OpenRAM Usage
- OpenRAM Development
- How to port OpenRAM to a New Technology
- How to change the OpenRAM Circuits/Modules
- Conclusion

Memory Organization

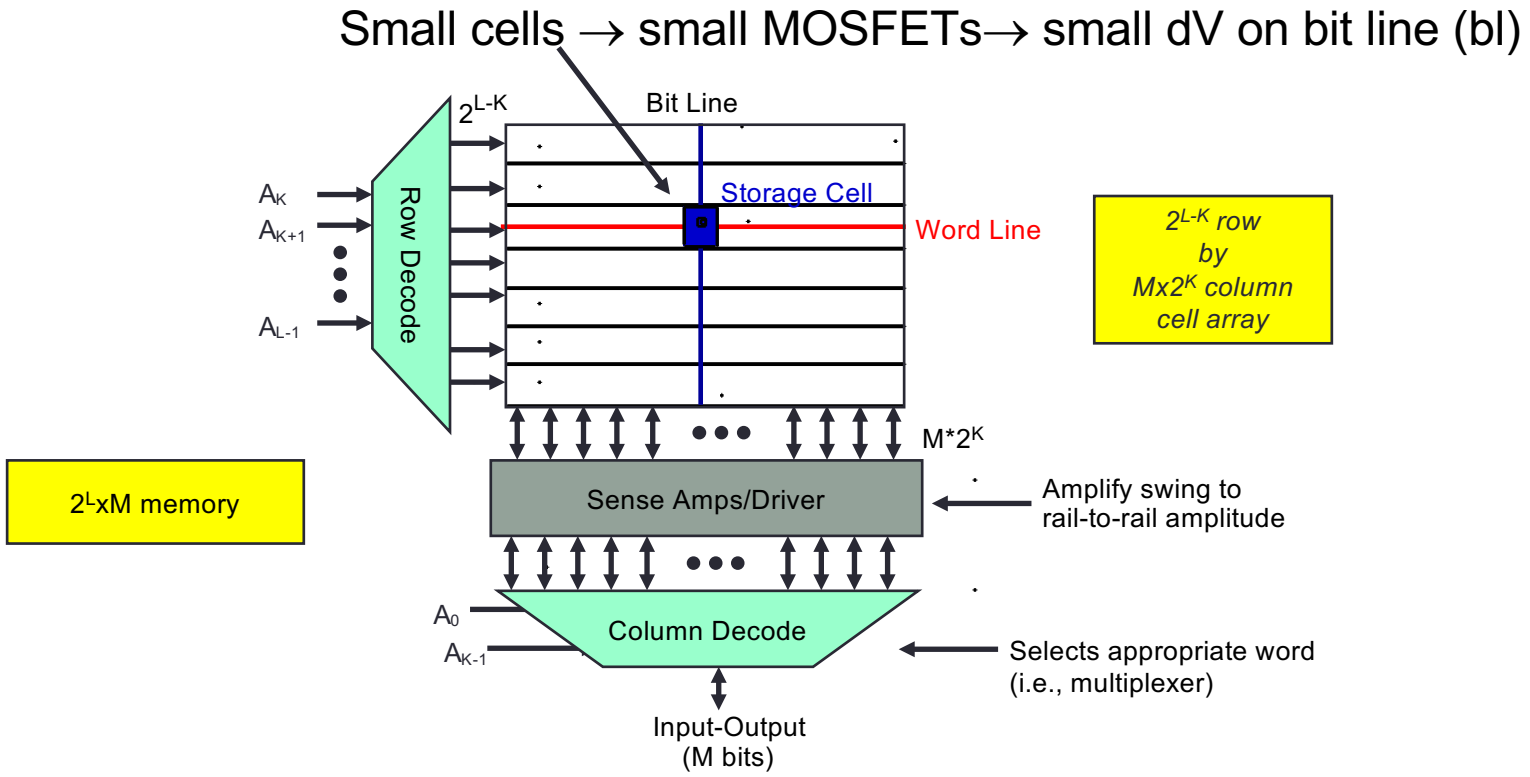




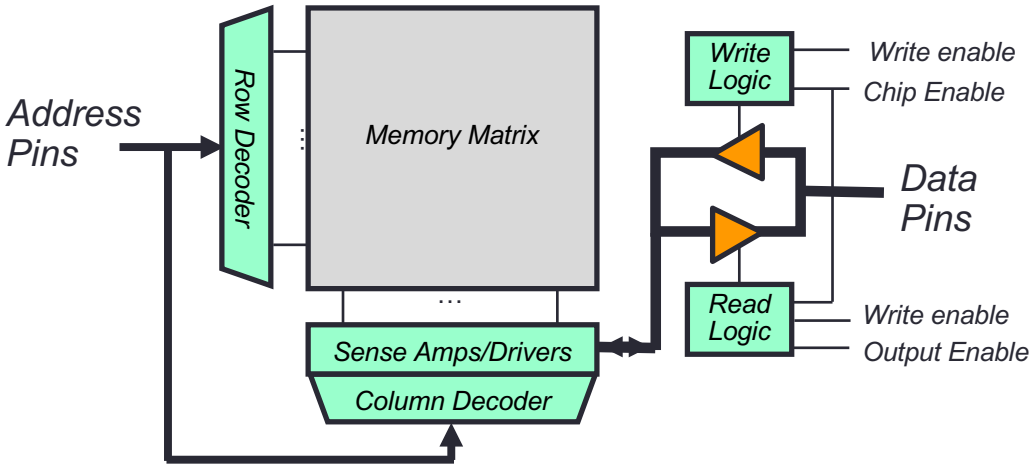


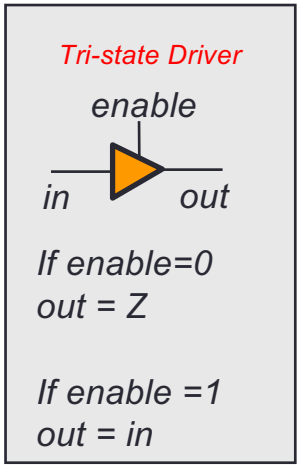
- Special circuit tricks are used for the cell array to improve storage density.
- RAM/ROM naming convention:
  - examples: 32 X 8, "32 by 8" => 32 8-bit words
  - 1M X 1, "1 meg by 1" => 1M 1-bit words
- Standard address, data and control signal names

Memory Array Architecture



Using External Memories





- Address pins drive row and column decoders
- Data pins are bidirectional, shared by reads and writes

- Output Enable gates the chip's tristate driver
- Write Enable sets the memory's read/write mode
- Chip Enable/Chip Select acts as a "master switch"

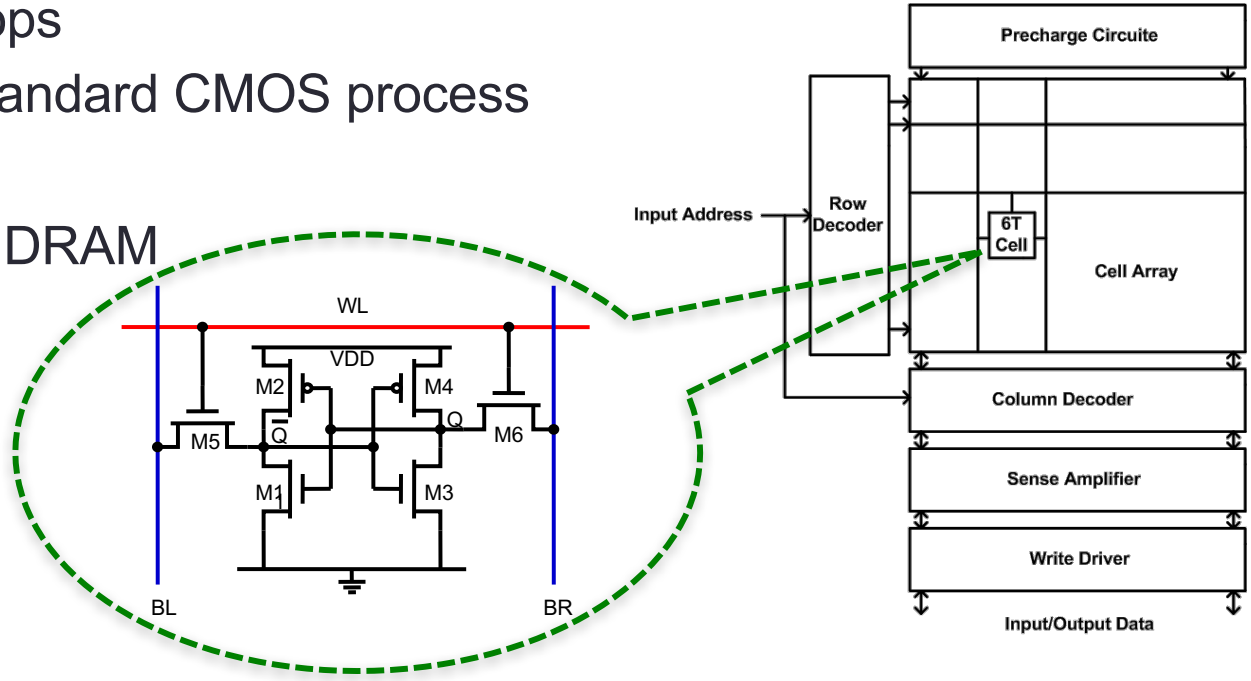
What's right/wrong with SRAM?

- It is currently everywhere and will be everywhere.
  - It is fast and efficient
  - Top of the food chain
  - Small footprint

- What's wrong with it?
  - It is not that easy to build without time and experience.
  - It is part of every computer system but usually absent from courses because of time to implement within course.
    - This signifies meaningful research is sometimes absent in this area, because we cannot teach people how to use it!

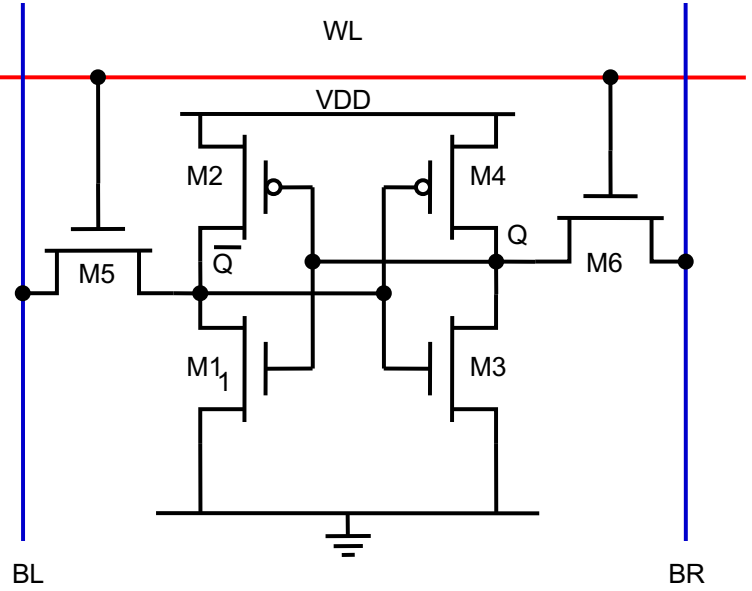
Static Random Access Memory (SRAM)

- SRAM is the most widely used form of on-chip memory RAMs.
- Denser than flip-flops
- Compatible with standard CMOS process
- Faster than DRAM
- Easier to use than DRAM

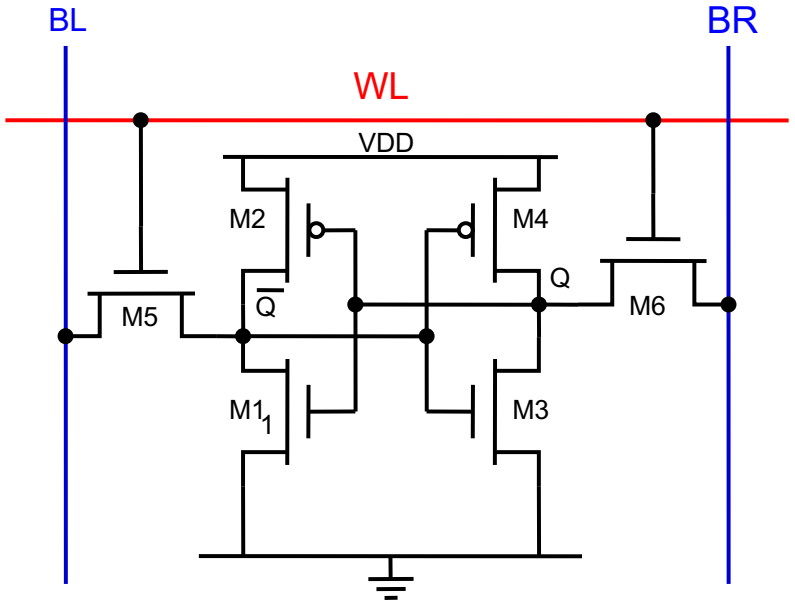


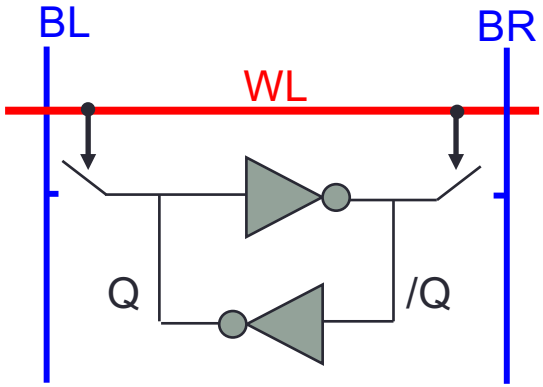
6T SRAM Cell

- ✓ Simple with Differential Structure
- ✓ Small Layout Area
- ✓ Fast Read Operation
- X High Leakage
- X Write Half-Select Disturbance
- X Low Stability at Low Voltages



SRAM Cell (The 6T Cell)





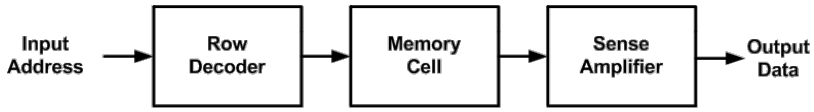
Write: *1)* Set BL, BR to (0,V_{DD}) or (V_{DD},0)
2) Enable WL (= V_{DD})

Read: 1) Enable WL (=V_{DD}).
2) Sense a small differential swing on BL/BR

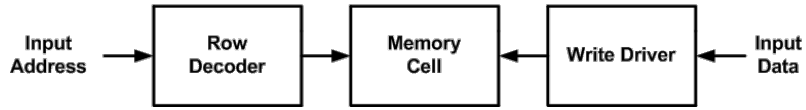
- State held by cross-coupled inverters (M1-M4)
- Retains state as long as power supply turned on
- Feedback must be overdriven to write into the memory

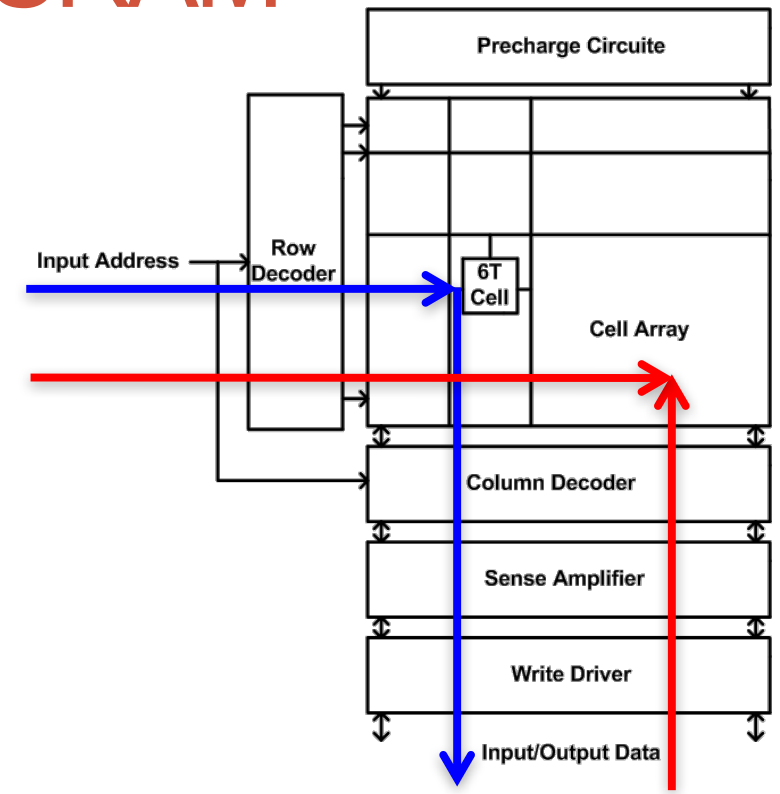
Read/Write Path in SRAM

- Data path for read operation



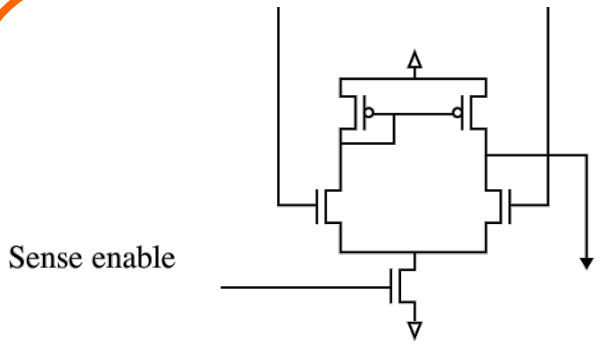
- Data path for write operation





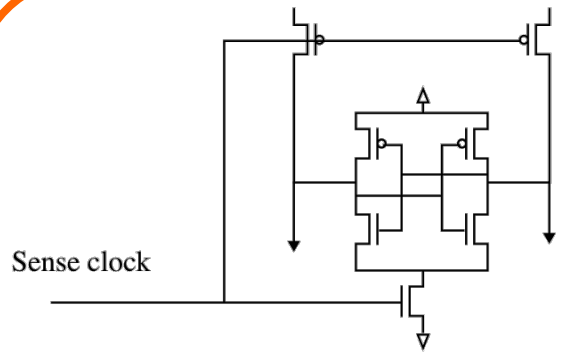
Latch-Based Sense Amplifier

- Bitline capacitors and resistance is significant for large array.
- Cannot easily change R, C, but can change bitline swing



Differential Pair Sense Amplifier

✓ Requires no Sense clock.
x Always dissipates static Power.

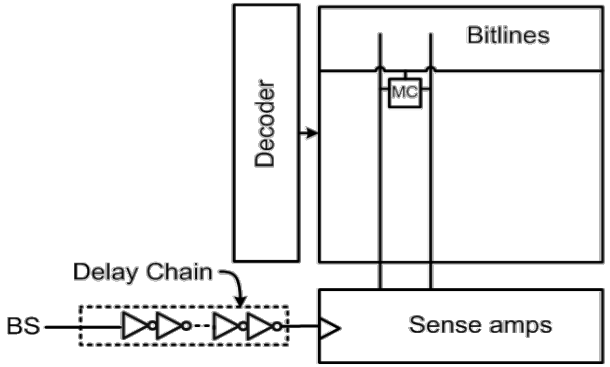


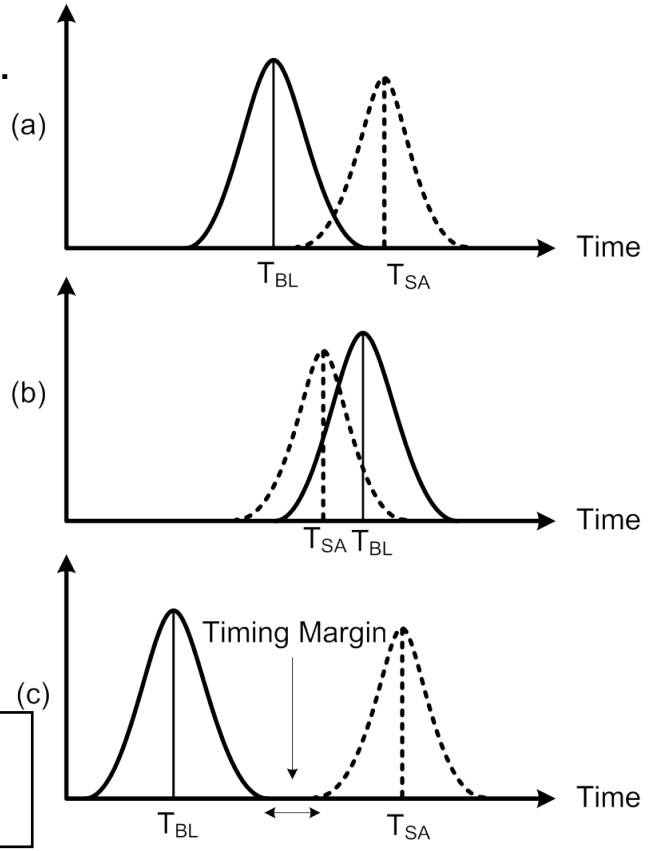
Latch-based Sense Amplifier

✓ Saves power by using Sense clock.
✓ Has isolation transistors.

Sense Amplifier Timing

T_{BL} : Time that bitline Voltage is sufficient for sensing.
T_{SA} : Time that sense-amp activates.
(a) Correct sensing when $T_{BL} < T_{SA}$.
(b) Wrong sensing when $T_{BL} > T_{SA}$.
(c) Timing margin in SRAM design.

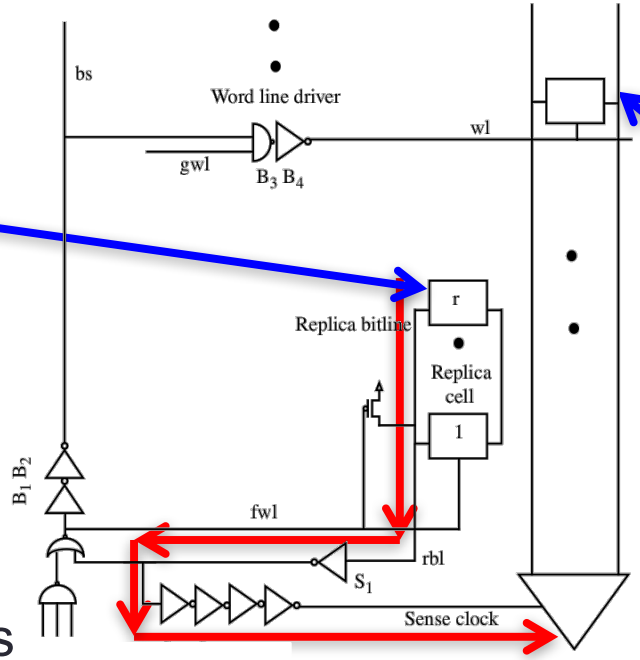




X The inverter delay does not track the memory cell delay over all process/environment conditions.

Replica Bitline (RBL) Technique

A Replica Technique for Wordline and Sense Control in Low-Power SRAMs" B. Amrutur and M. Horowitz, IEEE JSSC 1998



Memory cells drive delay of replica bitline.

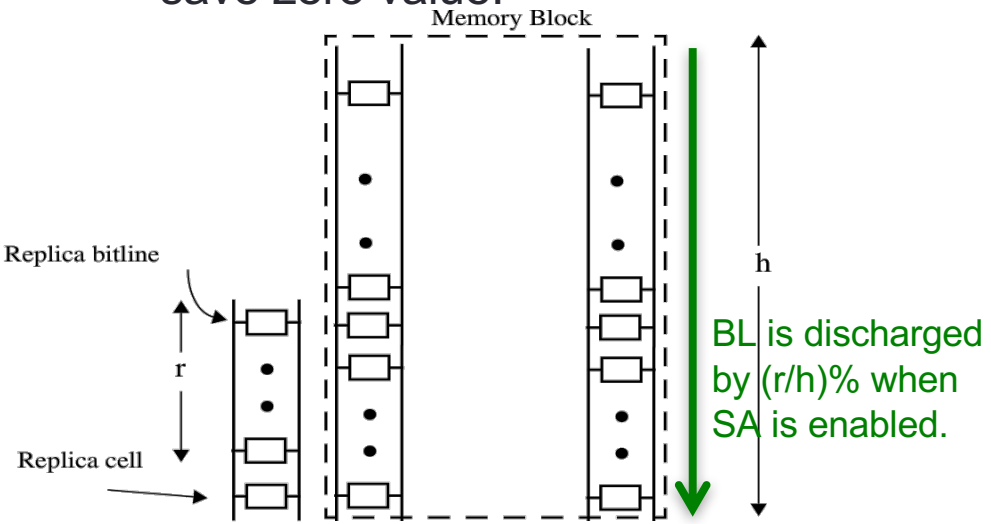
Memory cells drive delay of main bitlines.

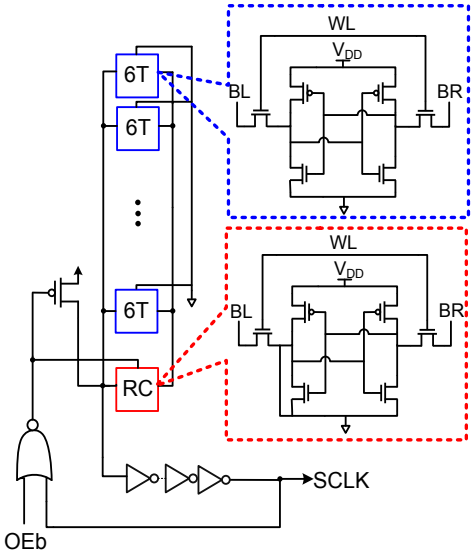
- ✓ PVT variation delay shift is same for bitlines and replica bitline.

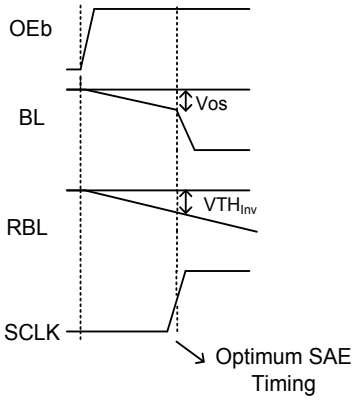
RBL is fully discharged to generate the SA enable signal.

Replica Bitline (RBL) Technique

- Height of replica bitline (r) is a fraction of main bitline height (h) and this fraction is determined by the required bitline swing for proper sensing.
- Replica Cell (RC) in RBL is same as memory cell and is hardcoded to always save zero value.



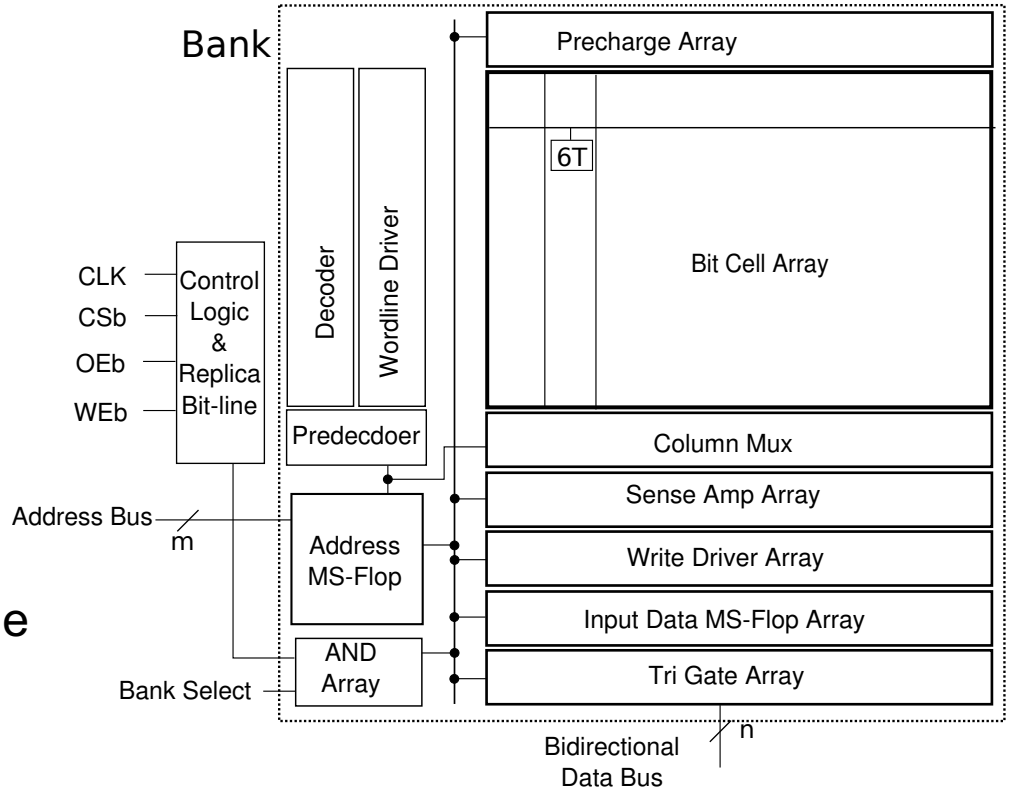




OpenRAM SRAM Architecture

SRAM Major blocks:

- Bit-cell Array (6T SRAM Cell)
- Hierarchical Address Decoder
- Wordline Driver
- Column Multiplexer
- Bitline Precharger
- Latch-type Sense Amplifier
- Tri-state Write Driver
- Control Logic with Replica Bitline



Outline

- Background on Memory Compilers
- OpenRAM Features
- OpenRAM Architecture and Circuits
- **OpenRAM Usage**
- OpenRAM Development
- How to port OpenRAM to a New Technology
- How to change the OpenRAM Circuits/Modules
- Conclusion

Getting OpenRAM

- https://github.com/mguthaus/OpenRAM
  - git clone https://github.com/mguthaus/OpenRAM.git
- https://openram.soe.ucsc.edu/
- License: GNU General Public License v3

Dependencies

- Python 2.7
  - numpy: http://www.numpy.org/
- SPICE (one or more)
  - Ngspice: http://ngspice.sourceforge.net/
  - HSPICE: from Synopsys
- DRC/LVS (necessary for porting technologies)
  - Calibre nmDRC nmLVS: From Mentor Graphics
  - Future: magic and netgen
- Technology PDK (one or more)
  - FreePDK45: http://www.eda.ncsu.edu/wiki/FreePDK
  - SCMOS: https://www.mosis.com/pages/design/flows/design-flow-scmos-kits
- Layout viewer/editor (optional)
  - LayoutEditor: http://www.layouteditor.net/
  - GLADE: http://www.peardrop.co.uk/
  - Magic: http://opencircuitdesign.com/magic/

OpenRAM Directory Structure

- **compiler** : OpenRAM compiler (OPENRAM_HOME)
  - Main source code of compiler in this directory.
  - **compiler/characterizer** : timing characterization code
  - **compiler/gdsMill** : gds reader/writer
  - **compiler/tests** : unit tests
- **technology** : Technology libraries (OPENRAM_TECH)
  - **technology/freepdk45** : Library for freepdk45 technology
  - **technology/scn3me_subm** : Llibrary SCMOS technology
  - **technology/setup_scripts** - setup scripts for your PDK

Basic Environment Setup

- OpenRAM has two environment variables:
  - OPENRAM_TECH points to a directory where all of your technology files reside. This allows proprietary technologies in a separate location with access control.
  - OPENRAM_HOME points to the directory of your OpenRAM source code. This allows a read-only installation if desired.
- Your PDK for a specific technology
  - This is set up in $OPENRAM_TECH/setup_scripts/setup_openram_<tech>.py for OpenRAM.

Running OpenRAM as a User

- Main executable is openram.py
  - Usage: openram.py [options] <config file>
  - Example:

```
openram.py -n -o testsram -p /designdir/testsram -v
example_config.py
```

- Uses a configuration file for an SRAM instance: example_config.py
- Options to custom name the SRAM (-o testsram), specify a design directory (-p /designdir/testsram), disable DRC/LVS (-n), specify SPICE simulator (-s ngspice), increase verbosity (-v), override technology (-t scn3me_subm), etc. -h for help.

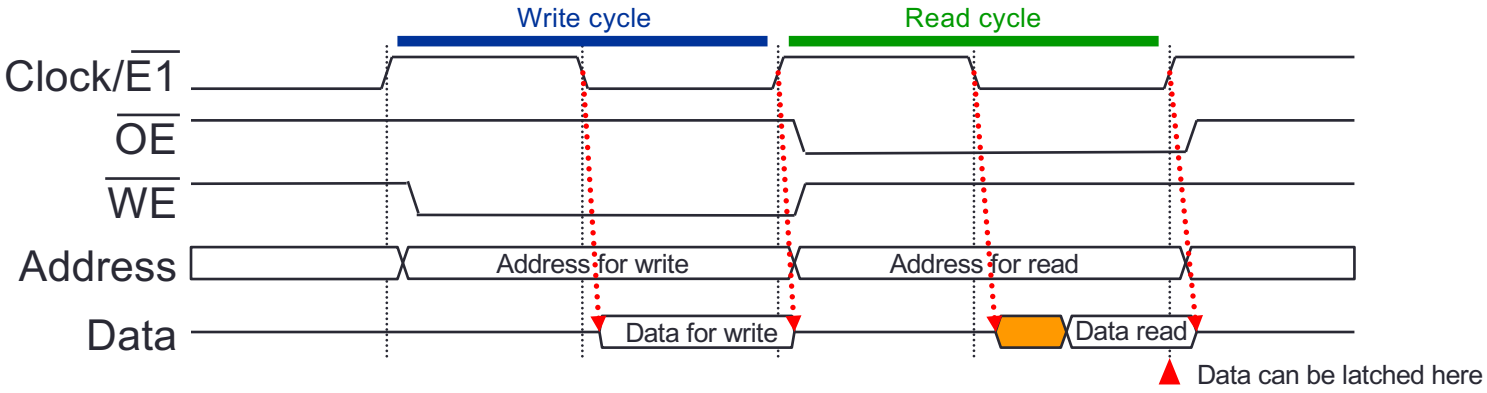
What is an SRAM configuration file?

- Each SRAM you generate will contain a configuration file that determines:
  - The technology name (tech_name)
  - The SRAM sizes:
    - word size (word_size)
    - number of words (num_words)
    - number of banks (num_banks)
  - Defines which modules to use if there are multiple alternatives for architecture components. For example, 6T bitcell or 8T bitcell?

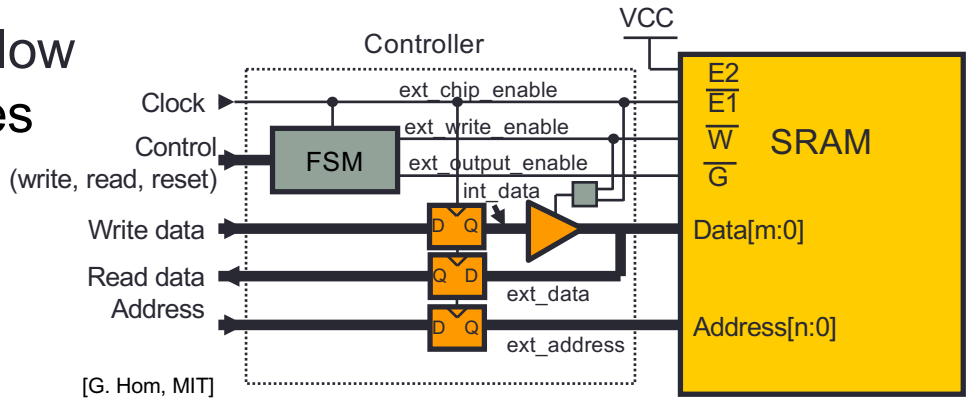
Example SRAM Configuration File (example_config.py)

```
word_size = 1
num_words = 16
num_banks = 1

tech_name = "freepdk45"

decoder = "hierarchical_decoder"
ms_flop = "ms_flop"
ms_flop_array = "ms_flop_array"
…
tri_gate_array = "tri_gate_array"
wordline_driver = "wordline_driver"
replica_bitcell = "replica_bitcell"
bitcell = "bitcell"
delay_chain = "logic_effort_dc"
```

Sample Memory Interface Logic



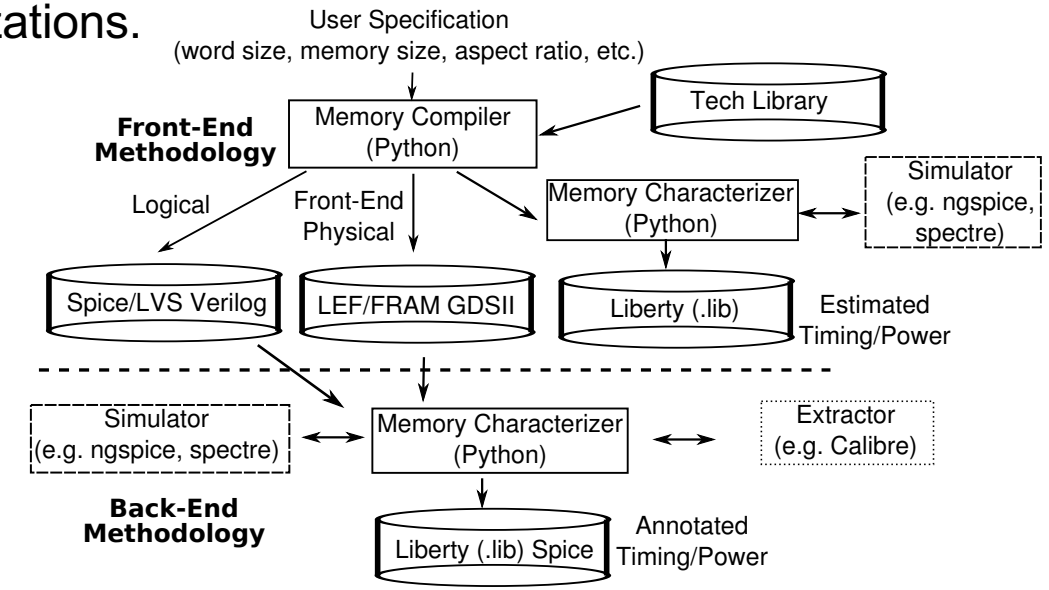
- Drive data bus only when clock is low
- Ensures address is stable for writes
- Prevents bus contention
- Minimum clock period is twice memory access time



OpenRAM Design Flow

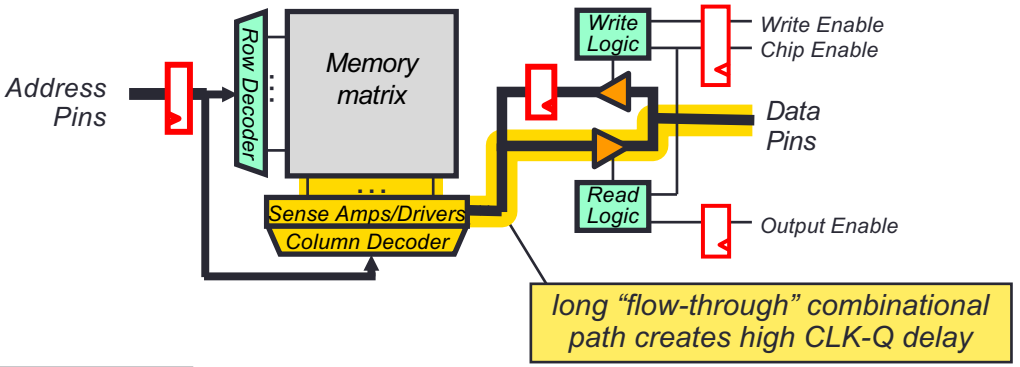
OpenRAM's framework is divided into front-end and back-end methodologies:

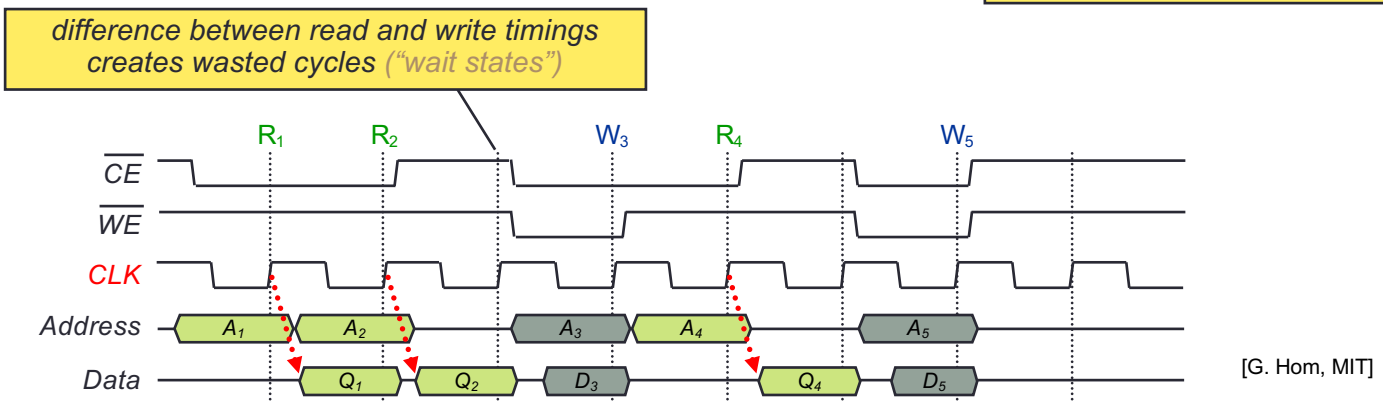
- Front-end has the compiler and the characterizer.
- Back-end generate annotated timing/power models using back-annotated characterizations.



Synchronous SRAM Memories

Clocking provides input synchronization and encourages more reliable operation at high speeds

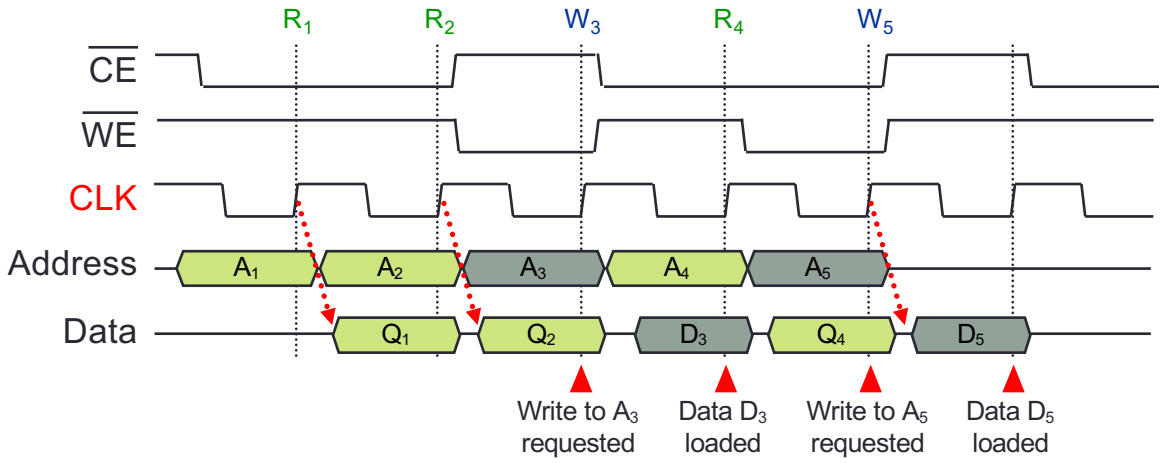




[G. Hom, MIT]

Zero Bus Turnaround (ZBT)

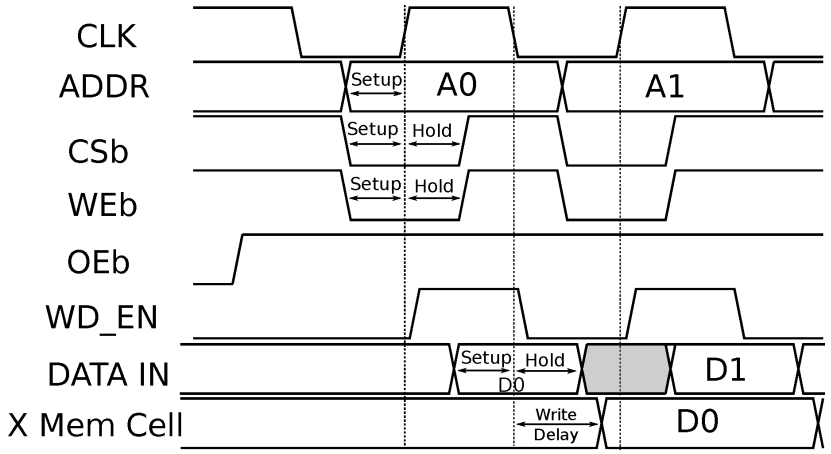
- The wait state occurs because:
  - On a read, data is available *after* the clock edge
  - On a write, data is set up *before* the clock edge
- ZBT ("zero bus turnaround") memories change the rules for writes
  - On a write, data is set up after the clock edge (so that it is read on the following edge)
  - Result: no wait states, higher memory throughput



[G. Hom, MIT]

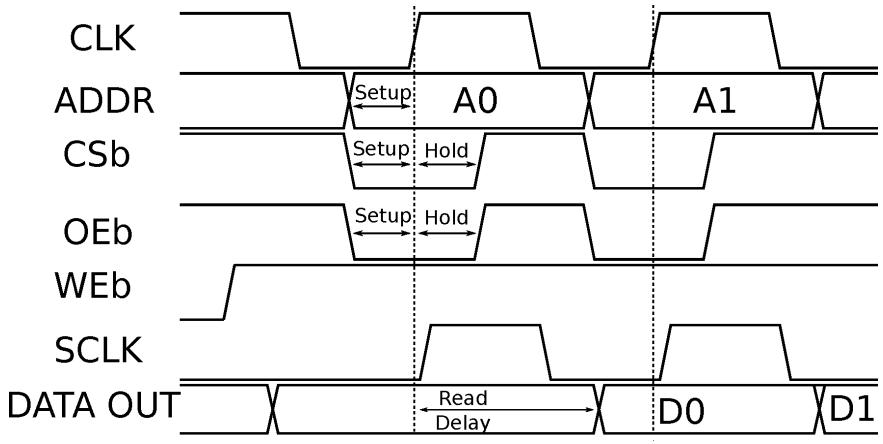
SRAM Timing in Write Mode

- OpenRAM SRAM is a synchronous memory with system clock (clk).
- Externally provided control signals are:
  - Output Enable (OEb)
  - Chip Select (CSb)
  - Write Enable (WEb
- OpenRAM uses Zero Bus Turn-around (ZBT) technique in timing.
- The ZBT enables higher memory throughput since there are no wait states.



SRAM Timing in Read Mode

- OpenRAM uses Replica Bit-Line (RBL) structure for timing of the sense amplifiers.
- The RBL turns on the sense amplifiers at the exact time in presence of process variations.
- Read occurs after the negative edge, but read delay is measured relative to positive edge.

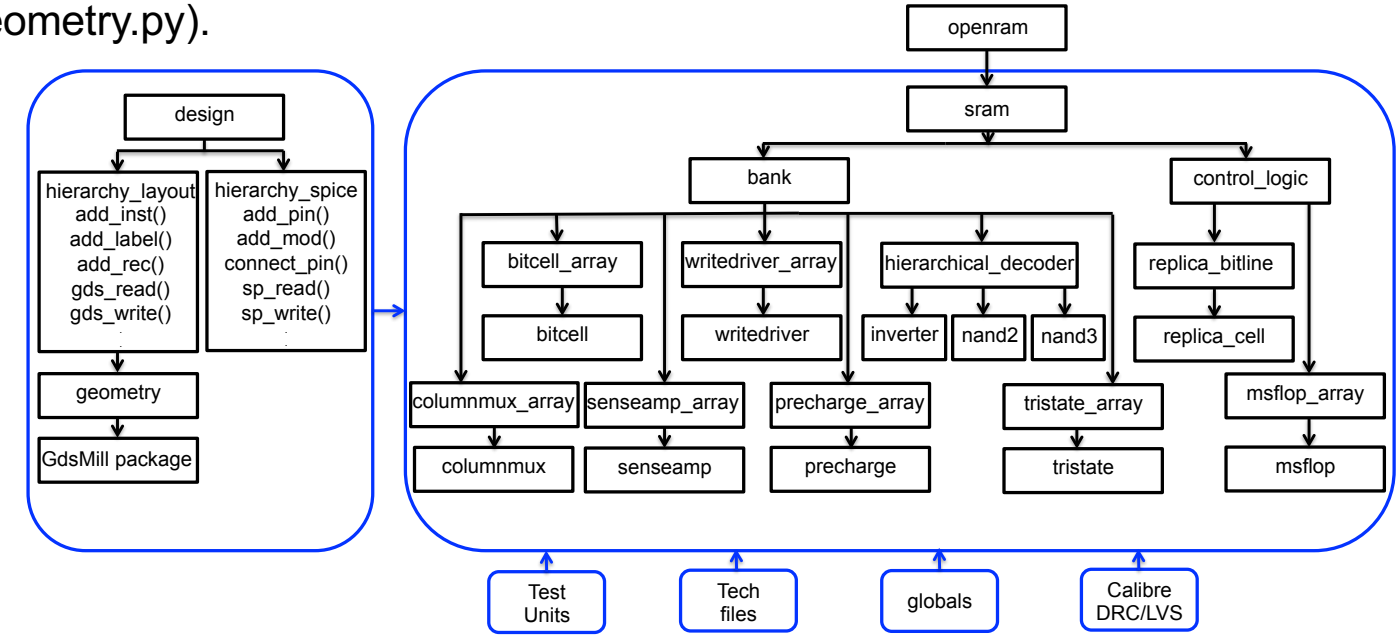


Outline

- Background on Memory Compilers
- OpenRAM Features
- OpenRAM Architecture and Circuits
- OpenRAM Usage
- **OpenRAM Development**
- How to port OpenRAM to a New Technology
- How to change the OpenRAM Circuits/Modules
- Conclusion

OpenRAM Structure

- OpenRAM has an integrated, custom GDSII library (gdsMill) to read, write, and manipulate GDSII files.
- To make the interfacing easier, OpenRAM implements a geometry wrapper class (geometry.py).



Design Class (design.py)

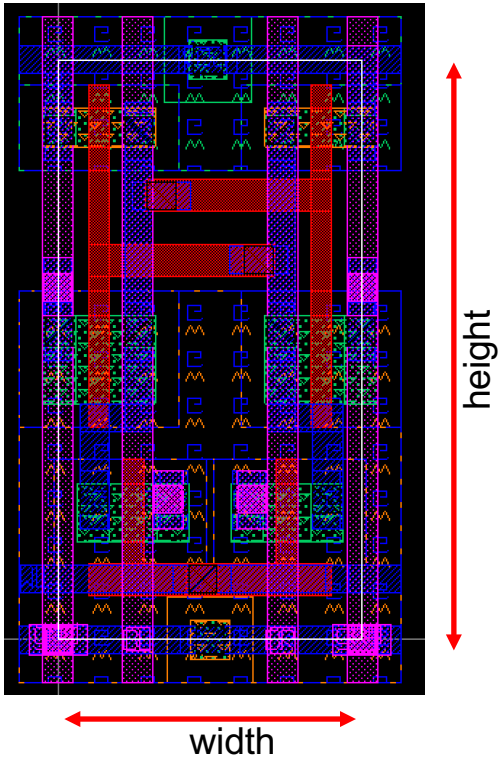
- All of the previous classes are derived from the “design” class. A design class has:
  - Layout (hierarchy_layout.py)
    - Instances, objects (shapes), width, height, pin locations
  - Netlist (hierarchy_spice.py)
    - Modules (other design classes), pins, connections
- A design class has numerous utility functions (e.g., write GDS/SPICE, add wires, add pins, etc.)
- A design class can run DRC and LVS on itself.

Types of Design Cells

- **Pre-made Designs**
  - Simplest but also the least technology portable.
  - Must be hand-made for each new technology.
- **Low-Level Designs**
  - Parameterized transistor class (ptx).
  - Parameterized logic gate classes (inverter, nand2, nand3, nor2).
- **High-Level Designs**
  - SRAM class instantiates the control logic module and the SRAM banks.
  - Bank class does the bulk memory layout. It instantiates bit-cell arrays, address decoders along with their precharge, sense amplifiers, and input/output data flops.

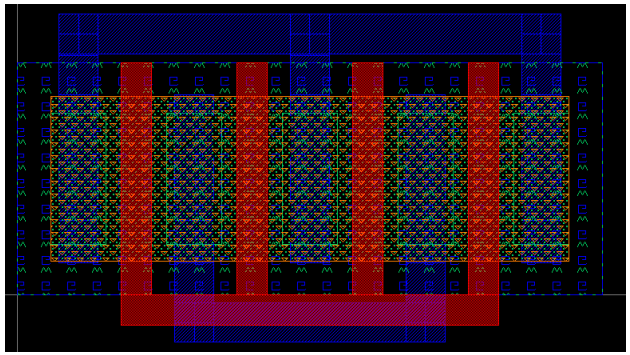
Pre-made Cell Example: 6t cell

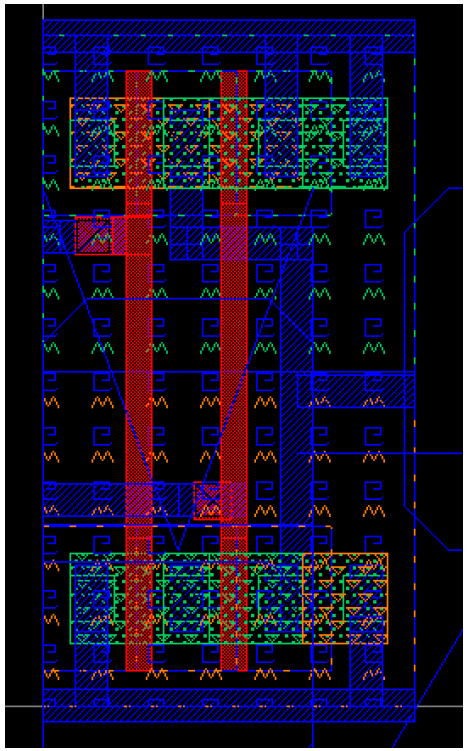
- From fab or self-made
- Layout is read from GDS file
  - Boundary determines the size for placement.
  - Pins are parsed from labels and shapes.
    - Some pins may be connected by abutment!
- Netlist is read from SPICE file
  - Pins are parsed from subckt
- Cells that are premade in OpenRAM
  - 6t cell
  - replica 6t cell
  - sense amplifier
  - flip-flop
  - write driver
  - tristate gate



Low-Level Class Example: ptx, nand2

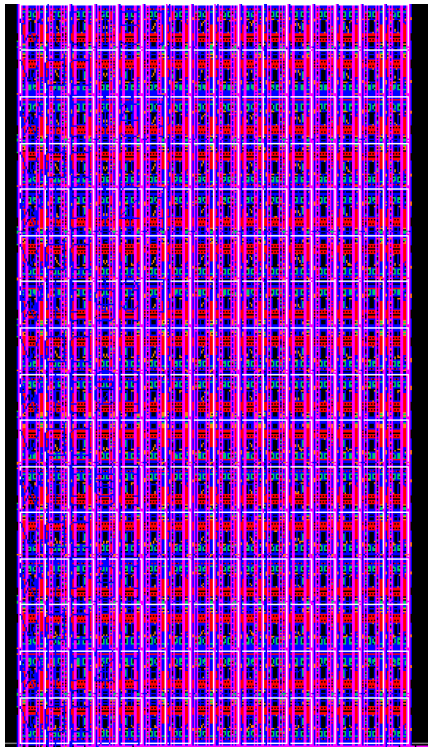
- Parameterized classes
  - Size, fingers, transistor type
  - Ptx has optional contacted source/drain
- Subset of design rules are needed in the tech.py file





High-Level Class Example: bitcell_array

- Can place instances of other design classes.
- Must generate its own layout and netlist.
- Uses utility functions:
  - add_mod
  - add_inst/connect_inst
  - add_rect
  - add_via/add_contact
  - add_wire (auto vias)
  - add_path (single layer wire)
  - add_layout_pin



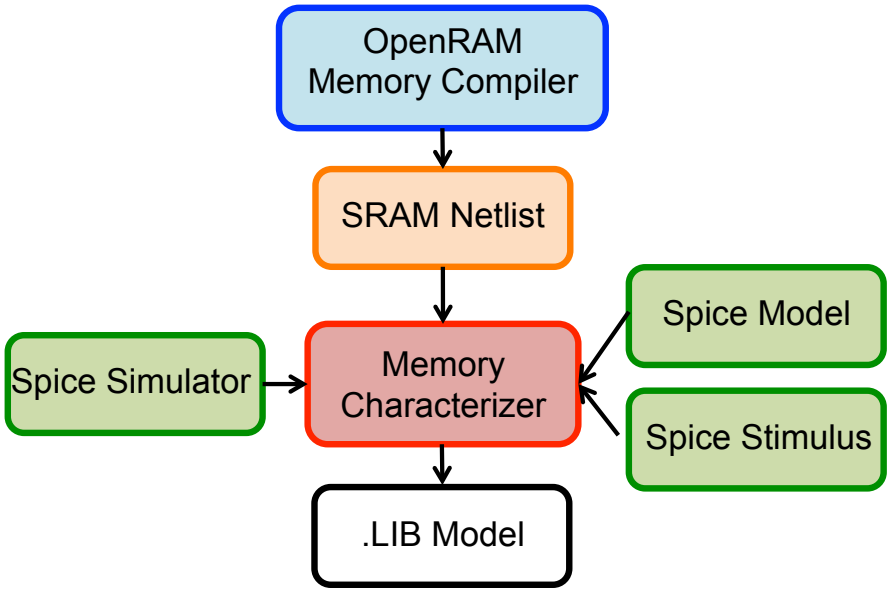
Technology and Tool Portability

- OpenRAM is technology independent by using a technology directory that includes:
  - Technology's specific information
  - Subset of technology's DRC rules and the GDS layer map
  - Premade library cells (6T, sense amp, ..) to improve the SRAM density.
- For technologies that have specific design requirements, such as specialized well contacts, the user can include helper functions in the technology directory. These will be called through technology call-back mechanisms.
- OpenRAM provides a wrapper interface with DRC/LVS tools that allow flexibility of any DRC/LVS tool, the default is Calibre nmDRC and nmLVS.
  - DRC/LVS are performed at all levels of the design hierarchy to enhance bug tracking.
  - DRC/LVS can be disabled for improved run-time or if tool is not available.

OpenRAM Characterizer

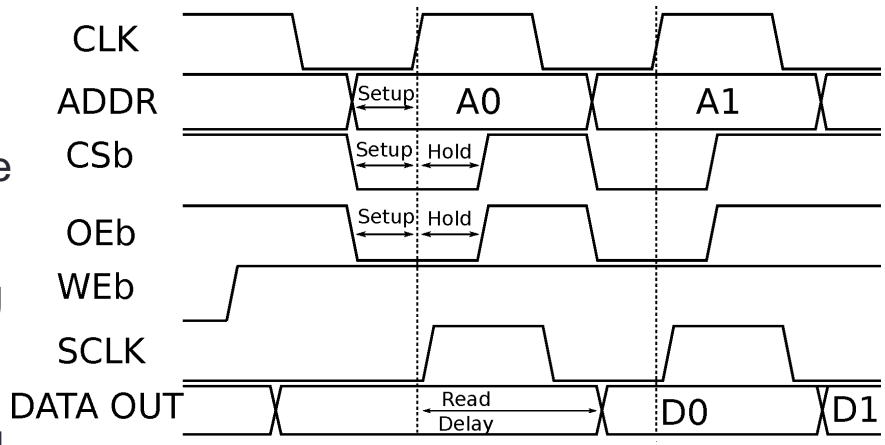
Characterizer measures the timing/power characteristics through SPICE simulation in 4 main steps:

- Generating the SPICE stimulus
- Running the circuit simulations
- Parsing the simulator's output
- Producing the Liberty (.lib) file.



Characterization

- Setup/Hold for Rise/Fall
  - Bisection search using generic SPICE syntax
  - First, finds a feasible period by doubling the time
  - Starts with a “hint” from the technology file but not required.
- Min Period/Delay for Rise/Fall
  - Measures data out delay while minimizing the period
    - First half of period is address decoding
    - Second half of period is access time
  - Performs a bisection search on the period
- Power Characterization
  - Measures read and write during rise/fall delay



OpenRAM Unit Tests

OpenRAM has the set of thorough regression tests implemented with the Python unit test framework:

- Unit tests allow users to add features without worrying about breaking functionality.
- Unit tests guide users when porting to new technologies. Unit tests pass in both FreePDK45 and SCMOS.
- Every sub-module has its own regression test.
- There are also regression tests for memory functionality, library cell verification, timing verification, and technology verification.

Unit Test Organization

- Grouped bottom-up for porting:
  - **01 and 02** test library cells (DRC & LVS of predesigned cells).
  - **03 and 04** test parameterized gate cells (inv, nand, nor, contact, ptx…).
  - **05 – 20** test the modules (bitcell-array, control-logic, sram…).
  - **21 – 30** test characterization, liberty, .lef, .v files.
- Run all the tests in a technology with **regress.py -t freepdk45**
- Regression daemon script **regress_daemon.py** checks out from svn/git, runs regression, and emails results.

Running Unit Tests

- Can specify command-line options:
  - Increase verbosity (-v)
- Must specify technology
  - -t scn3me_subm or -t freepdk45
- Can run individual tests while debugging
- Results in OK, FAIL or ERROR
- Example:
  - python tests/16_replica_bitline_test.py –t freepdk45

```
|=============================================================================|
|=========                  Running Test for:                        =========|
|=========                      freepdk45                            =========|
|=========            tests/16_replica_bitline_test.py               =========|
|=============================================================================|
.
----------------------------------------------------------------------
Ran 1 test in 5.282s

OK
```

Outline

- Background on Memory Compilers
- OpenRAM Features
- OpenRAM Architecture and Circuits
- OpenRAM Usage
- OpenRAM Development
- **How to port OpenRAM to a New Technology**
- How to change the OpenRAM Circuits/Modules
- Conclusion

Port OpenRAM to a New Tech – 1st step

Setup the technology file in a tech directory (e.g. freepdk45/tech/tech.py):

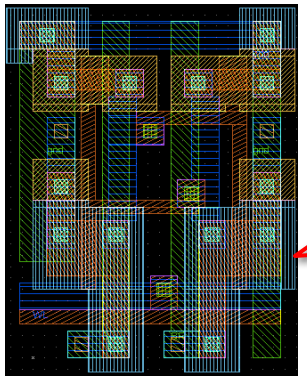
- Technology name (e.g. freepdk45)
- GDS layer map
- GDS library files (bitcell, sense-amp, write-driver,..)
- Design rules (DRC/LVS test setup)
- SPICE info (transistor name and model,..)
- SPICE stimulus variable (voltage, frequency,…)
- SRAM signal names (DATA, ADDR, clk,…)

Port OpenRAM to a New Tech – 2nd step

- Setup the GDS library by custom designing library cells (bitcell.gds, sense-amp.gds, …).
- Setup the SP library by creating the SPICE netlist for designed cells (bitcell.sp, sense-amp.sp, …).
- Unit test **01_drc** checks the DRC for layouts in GDS library.
- Unit test **02_lvs** checks the LVS for layouts in GDS library versus their SPICE netlist in SP directory.

6T cell SPICE netlist in SP library

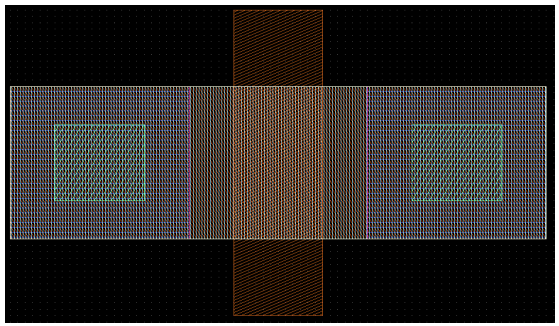
```
.SUBCKT cell_6t bl br wl vdd gnd
M1 1 2 vdd vdd pmos  W=0.9u L=1.2u
M2 2 1 vdd vdd pmos  W=0.9u L=1.2u
M3 br wl 2 gnd nmos  W=1.2u L=0.6u
M4 bl wl 1 gnd nmos  W=1.2u L=0.6u
M5 2 1 gnd gnd nmos W=2.4u L=0.6u
M6 1 2 gnd gnd nmos W=2.4u L=0.6u
.ENDS $ cell_6t
```



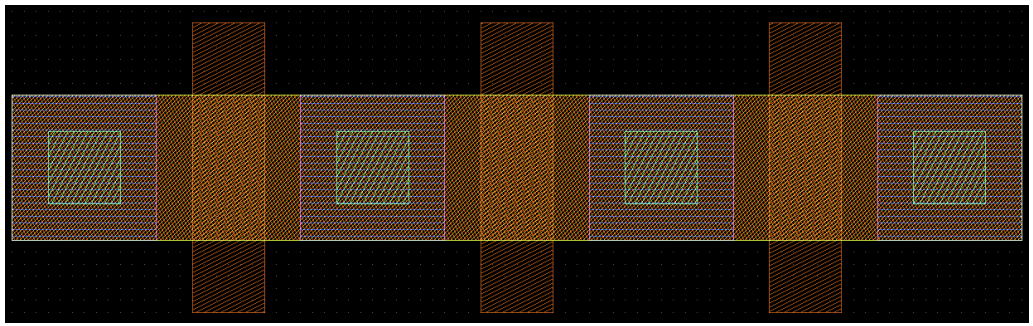
6T cell gds layout in GDS library

Port OpenRAM to a New Tech – 3rd step

- Make sure following tests generate clean DRC/LVS outputs:
- **03_ptx:** generates a single/multi finger transistor (nmos/pmos) based on the technology information.



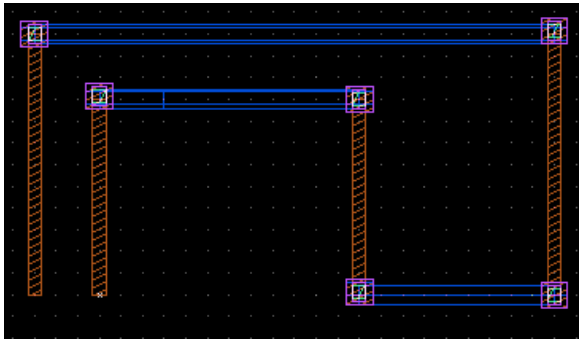
Single-finger NMOS



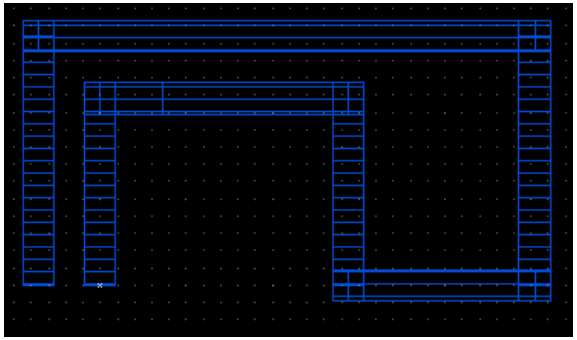
Triple-finger PMOS

Port OpenRAM to a New Tech – 3rd step

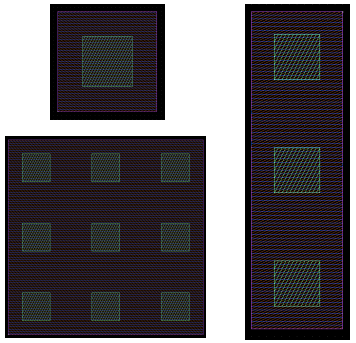
- **03_wire:** generates a metal wire between a set of points for a given layer set.
- **03_path:** generates a metal path between a set of points for a given layer type.
- **03_contact:** generate contact array in different size and type (poly-contact, via,..).



Wire for (M2, Via1, M1)



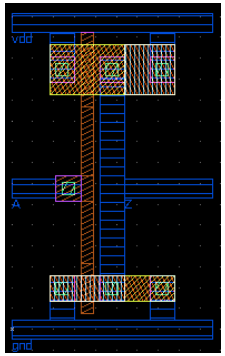
Path for M1



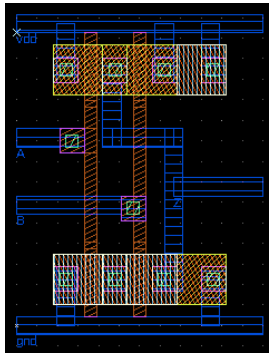
Contacts in different sizes

Port OpenRAM to a New Tech – 3rd step

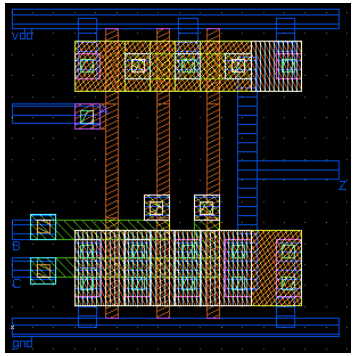
- **04_pinv:** generates a parametrically sized inverter using ptx.
- **04_nand_2:** generates a parametrically sized 2 input nand gate using ptx.
- **04_nand_3:** generates a parametrically sized 3 input nand gate using ptx.
- **04_nor_2:** generates a parametrically sized 2 input nor gate using ptx.



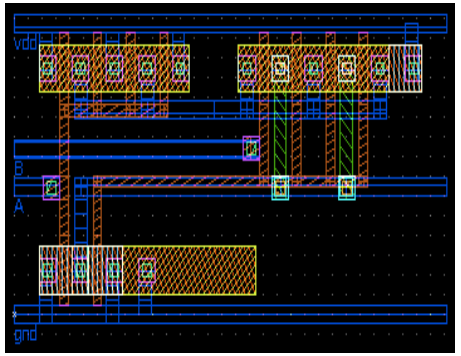
Inverter



NAND2



NAND3

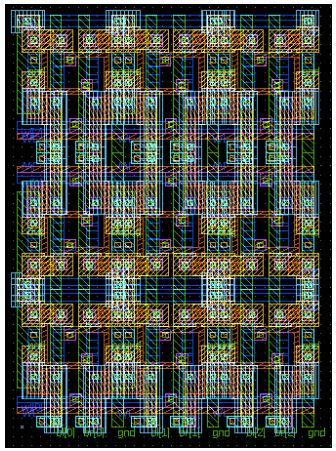


NOR2

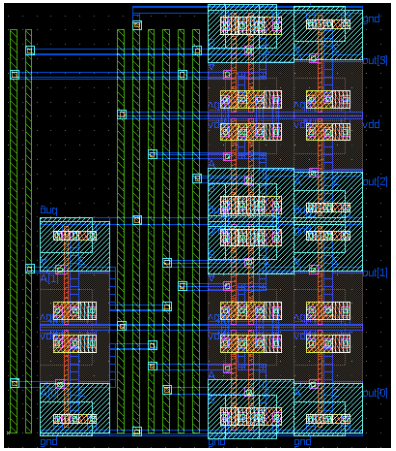
Port OpenRAM to a New Tech – 4th step

Following tests generate array of provided layouts in GDS library or parameterized cells:

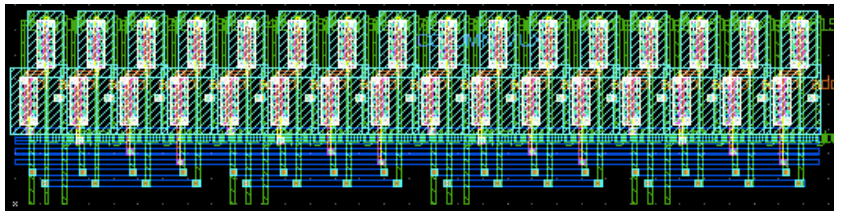
- **05_bitcell_array:** generates a *mxn* bitcell array.
- **06_hierarchical_decoder:** generates a hierarchical decoder with inverter and nand gates.
- **07_column_mux_array:** generates a column multiplexer with bit interleaving structure.



3x3 bitcell_array



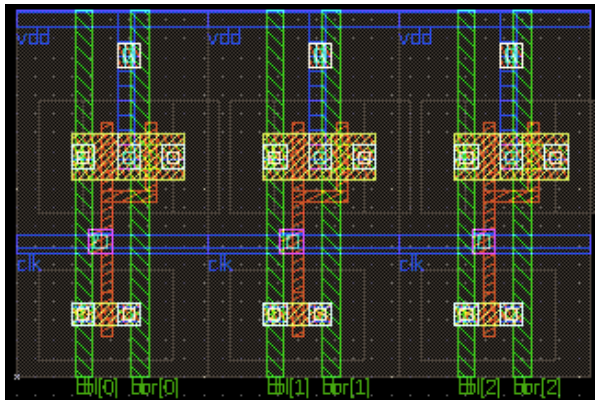
2:4 decoder



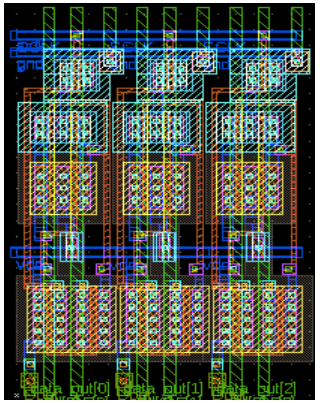
4:1 column_multiplexer

Port OpenRAM to a New Tech – 4th step

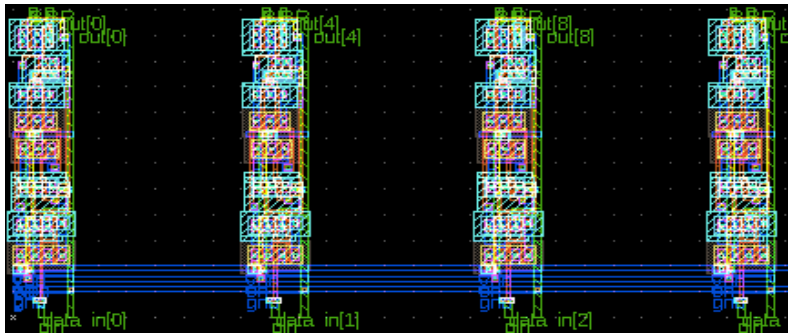
- **08_prechage_array:** generates a *1xm* parameterized precharge cell array.
- **09_ sense_amp_array:** generates a *1xm* sense amp array.
- **10_write_driver_array:** generates a *1xm* write driver array.
- **11_ms_flop_array:** generates master-slave flipflop arrays for data, address and controls.
- **15_tri_gate_array:** generates a *1xm* tri_gate array for bidirectional data bus.



1x3 precharge_array



1x3 sense_amp_array

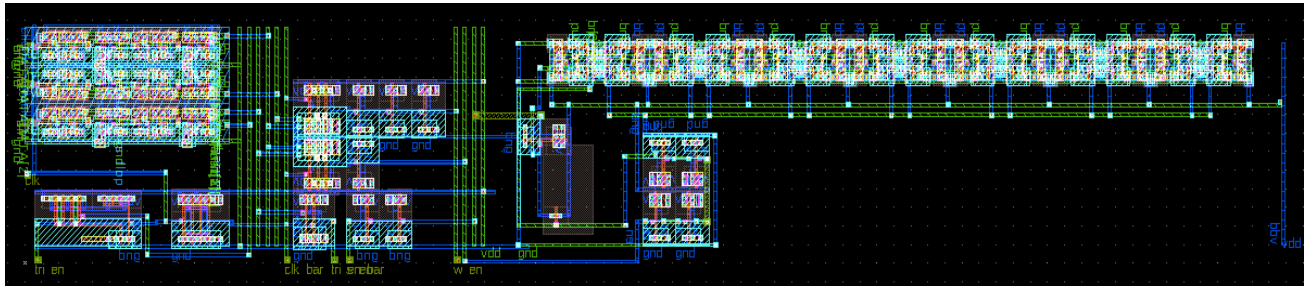


1x4 write_driver_array
(there is column_mux in array)

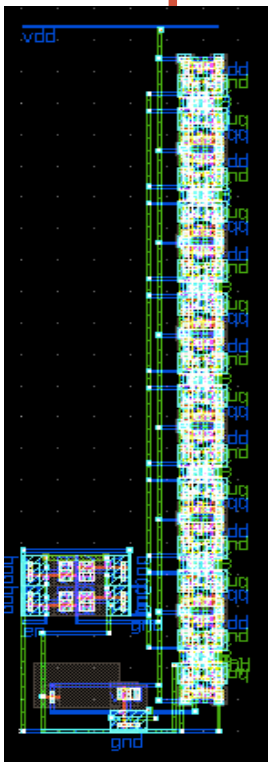
Port OpenRAM to a New Tech – 5th step

Following tests generate the control logic:

- **14_logic_effort:** generates chain of inverters.
- **16_replica_bitline:** generates the replica bitline using bitcell, replica cell and inverter chain.
- **15_control_logic:** generates the final control logic.



Control_logic (master-slave FF, Nand, Nor, Inverter and RBL)

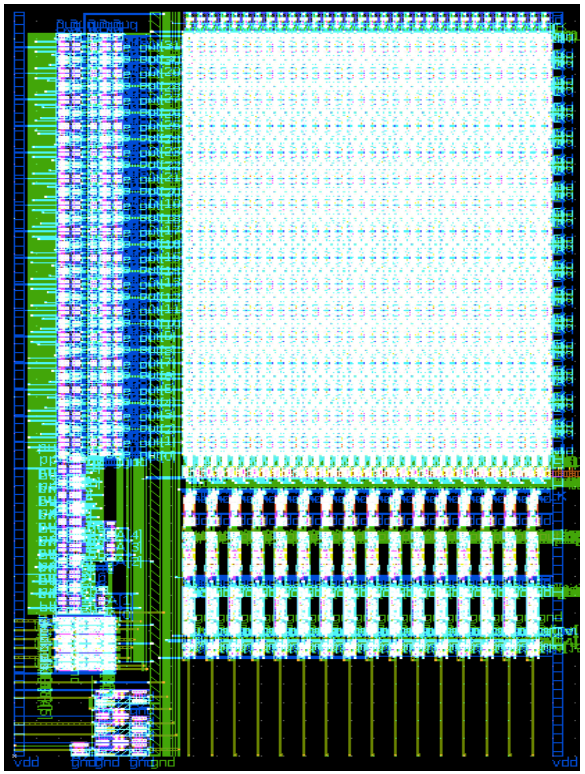


Replica Bitline

Port OpenRAM to a New Tech – 6th step

Following test generate the bank array:

- **19_bank:** generates a non-control memory bank by connecting and routing following modules:
  - Bitcell-array
  - Sense-amp-array
  - Write-driver-array
  - Hierarchical-decoder
  - Wordline-driver
  - Precharge-array
  - Column-multiplexer (if needed)
  - MS-flipflop-array
  - Trigate-array

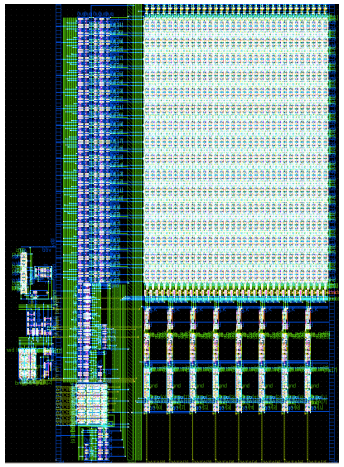


32x32 bit bank with a 2:1 column_multiplexer

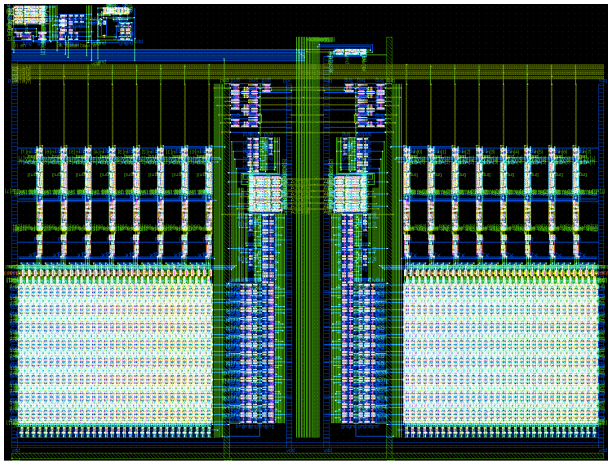
Port OpenRAM to a New Tech – 6th step

Following tests generate SRAM array.

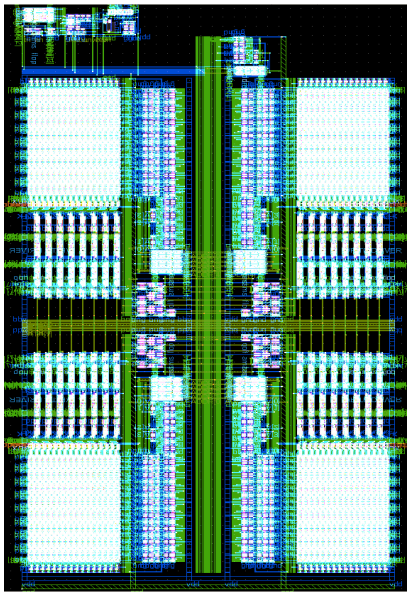
- **20_sram_1bank:** connects and route one memory bank to control logic
- **20_sram_2bank:** generates a double bank SRAM array.
- **20_sram_4bank:** generates a quad bank SRAM array.



Single bank SRAM



Dual bank SRAM

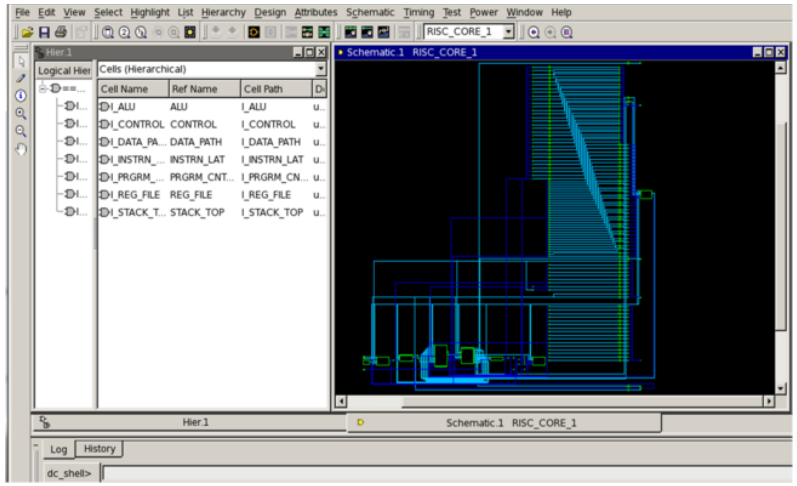


Quad bank SRAM

Port OpenRAM to a New Tech – 7th step

Use characterizer and a SPICE simulator to ensure that characterization works:

21_timing_delay: measures the timing delay of the SRAM.
21_timing_hold/setup: measures the setup/hold timing of flip-flops.
23_lib_sram: generates a lookup-table liberty file for synthesis.

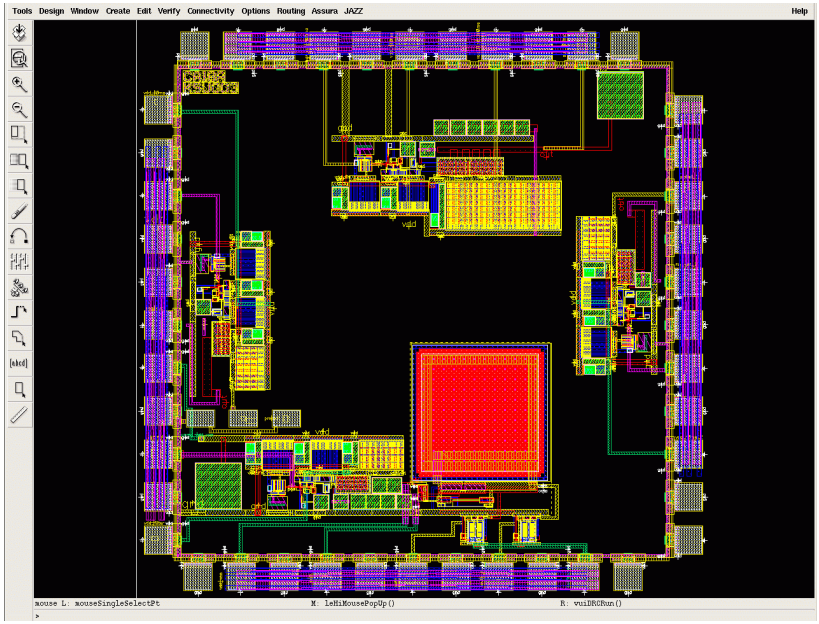


Port OpenRAM to a New Tech – 7th step

Confirm that .lef file and verilog models generate correctly:

24_lef_sram: generates a LEF file of SRAM for place and route.

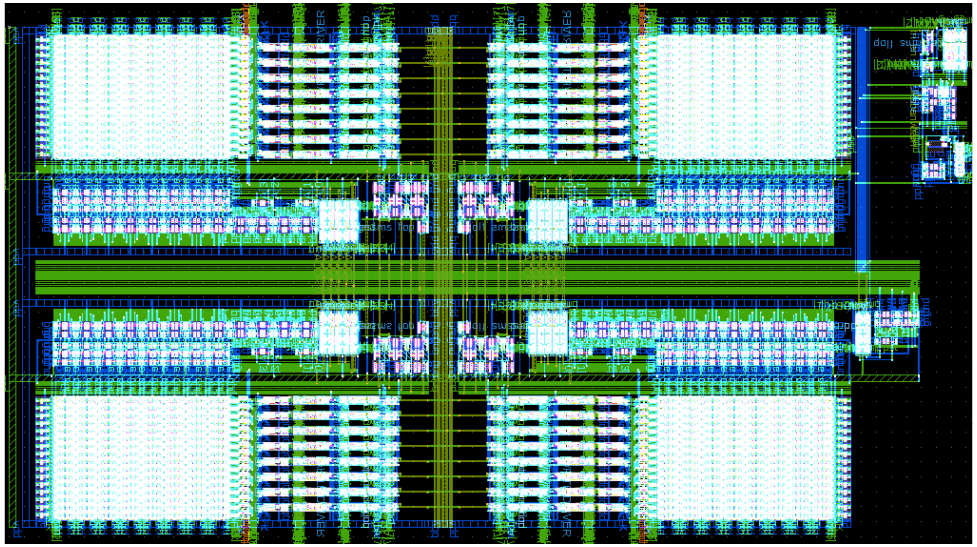
25_verilog_sram: generates a Verilog model for synthesis and place & route.



Note: The LEF should be manually inspected in your P&R tool.

Port OpenRAM to a New Tech – 8th step

- If all the previous tests pass you can now make “any” size of SRAM array and generate gds, SPICE, lib, LEF, Verilog, … by running openram.py.
- **Copy the config_20_freepdk45.py for each size memory and modify the size and technology:**
  - tech_name
  - word_size
  - num_words
  - num_banks



Outline

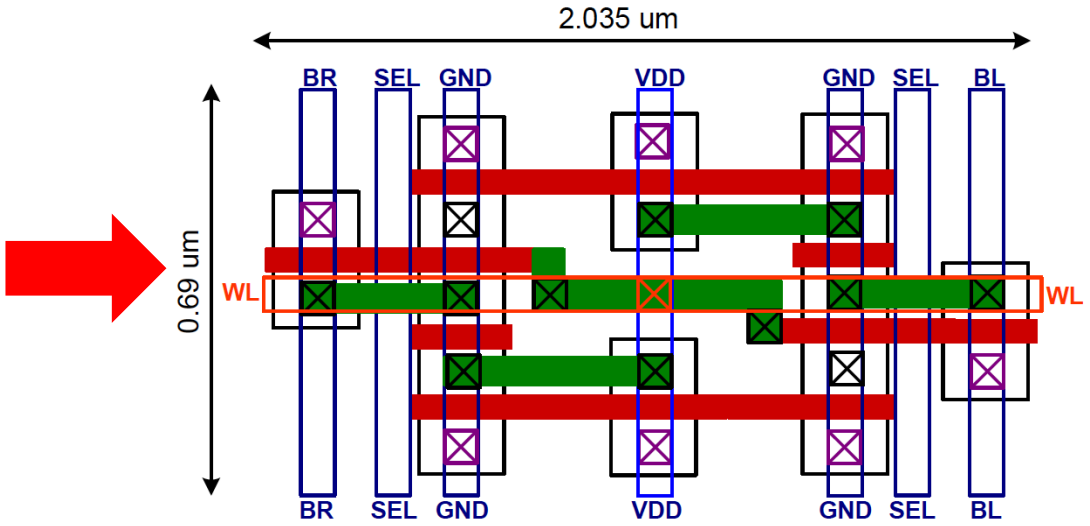
- Background on Memory Compilers
- OpenRAM Features
- OpenRAM Architecture and Circuits
- OpenRAM Usage
- OpenRAM Development
- How to port OpenRAM to a New Technology
- **How to change the OpenRAM Circuits/Modules**
- Conclusion

Customize OpenRAM Circuits – 1st step

Want to generate an 12T SRAM array instead of a 6T?!!

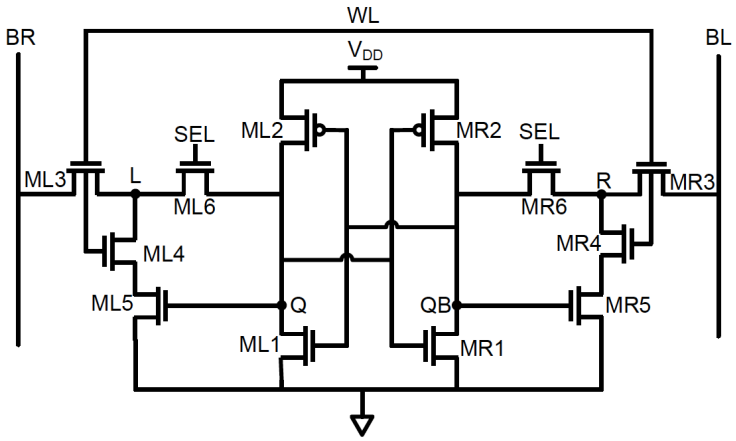
- First, add the 12T bitcell layout to GDS library: technology/freepdk45/gds_lib/cell_12t.gds

Make sure it has a Boundary layer and pin labels.



Customize OpenRAM Circuits – 2nd step

- Add the 12T SPICE netlist to SP library: technology/freepdk45/sp_lib/cell_12t.sp




```
******* "cell_12t" **********
.SUBCKT cell_12t BL BR WL sel vdd gnd

M1 Q QB vdd vdd pfet L=0.04u W=0.1u
M2 QB Q vdd vdd pfet L=0.04u W=0.1u
M3 Q QB gnd gnd nfet L=0.04u  W=0.1u
M4 QB Q gnd gnd nfet L=0.04u  W=0.1u
M5 L Q gnd gnd nfet L=0.04u  W=0.1u
M6 R QB gnd gnd nfet L=0.04u  W=0.1u
M7 L WL vdd vdd pfet L=0.04u  W=0.1u
M8 R WL vdd vdd pfet L=0.04u  W=0.1u
M9 Q sel L gnd nfet L=0.04u  W=0.1u
M10 QB sel R gnd nfet L=0.04u  W=0.1u
M11 L WL BL gnd nfet L=0.04u  W=0.1u
M12 R WL BR gnd nfet L=0.04u  W=0.1u

.ENDS cell_12t
```

Customize OpenRAM Circuits – 3rd step

- Create a new bitcell module that uses the 12T cell: cell_12t.py

```
import design
from tech import cell
import debug

class cell_12t:
    """
    A single bit 12T cell.
    """
    pins = ["BL", "BR", "WL", "vdd", "gnd"]
    chars = utils.auto_measure_libcell(pins, "cell_12t", GDS["unit"],layer["boundary"])

    def __init__(self, name="cell_12t"):
        design.design.__init__(self, name)
        debug.info(2, "Create bitcell object")
        self.width = bitcell.chars["width"]
        self.height = bitcell.chars["height"]
```

Customize OpenRAM Circuits – 4th step

- Change an SRAM configuration file to refer to your new 12T module as the bitcell: **config_example_12t.py**

```
word_size = 32
num_words = 128
num_banks = 1

…
tri_gate_array = "tri_gate_array"
wordline_driver = "wordline_driver"
replica_bitcell = "replica_cell_12t"
bitcell = "cell_12t"
delay_chain = "logic_effort_dc"
```

Oh, replica too!!

Customize OpenRAM Circuits – 5th step

- Repeat for **replica_bitcell.py**.
  - Add the 12T replica bitcell layout to GDS library: technology/freepdk45/gds_lib/replica_cell_12t.gds
  - Add the 12T SPICE replica bitcell netlist to SP library: technology/freepdk45/sp_lib/replica_cell_12t.sp
  - Create a new replica bitcell module that uses the 12T cell: replica_cell_12t.py
  - Change an SRAM configuration file to refer to your new 12T replica module as the replica_bitcell

- Then, make sure all the unit tests all pass!

Customize OpenRAM Modules – 1st Step

- What if you want to implement a new module? Let's look at the bitcell_array.py class.

```
import debug
import design
from tech import bitcell
from vector import vector
from globals import OPTS


class bitcell_array(design.design):
    """
    Creates a rows x cols array of memory cells. Assumes bit-lines
    and word line are connected by abutment.
    Connects the word lines and bit lines.
    """
```

Customize OpenRAM Modules – 2st Step

- Top level: loads sub-modules then creates netlist/layout and verifes.

```
def __init__(self, name, cols, rows):
        design.design.__init__(self, name)
        debug.info(1, "Creating {0} {1} x {2}".format(self.name, rows, cols))
        self.column_size = cols
        self.row_size = rows

        c = reload(__import__(OPTS.config.bitcell))
        self.mod_bitcell = getattr(c, OPTS.config.bitcell)

        self.add_pins()
        self.create_layout()
        self.add_labels()
        self.DRC_LVS()
```

Customize OpenRAM Modules – 3rd Step

- Dynamically add all the pins to the netlist based on the size.

```
def add_pins(self):
        for col in range(self.column_size):
            self.add_pin("bl[{0}]".format(col))
            self.add_pin("br[{0}]".format(col))
        for row in range(self.row_size):
            self.add_pin("wl[{0}]".format(row))
        self.add_pin("vdd")
        self.add_pin("gnd")
```

Customize OpenRAM Modules – 4th Step

- Place all the instances of the cell module and logically connect them.

Note: In this cell, all connections are physically made by abutment!

```
self.cell = self.mod_bitcell()
self.add_mod(self.cell)
…
xoffset = 0.0
for col in range(self.column_size):
        for row in range(self.row_size):
                name = "bit_r{0}_c{1}".format(row, col)

…
self.add_inst(name=name,
              mod=self.cell,
              offset=[xoffset, tempy],
              mirror=dir_key)
self.connect_inst(["bl[{0}]".format(col),
                   "br[{0}]".format(col),
                   "wl[{0}]".format(row),
                   "vdd", "gnd"])
```

Customize OpenRAM Modules – 5th Step

- Add labels for LVS and store pin offsets for higher level

```
offset = vector(0.0, 0.0)
for col in range(self.column_size):
     offset.y = 0.0
     self.add_label(text="bl[{0}]".format(col),
                    layer="metal2",
                    offset=offset + vector(cell_6t["BL"][0],0))
     self.add_label(text="br[{0}]".format(col),
                    layer="metal2",
                    offset=offset + vector(cell_6t["BR"][0],0))
     self.BL_positions.append(offset + vector(cell_6t["BL"][0],0))
     self.BR_positions.append(offset + vector(cell_6t["BR"][0],0))
```

Customize OpenRAM Modules – 6th Step

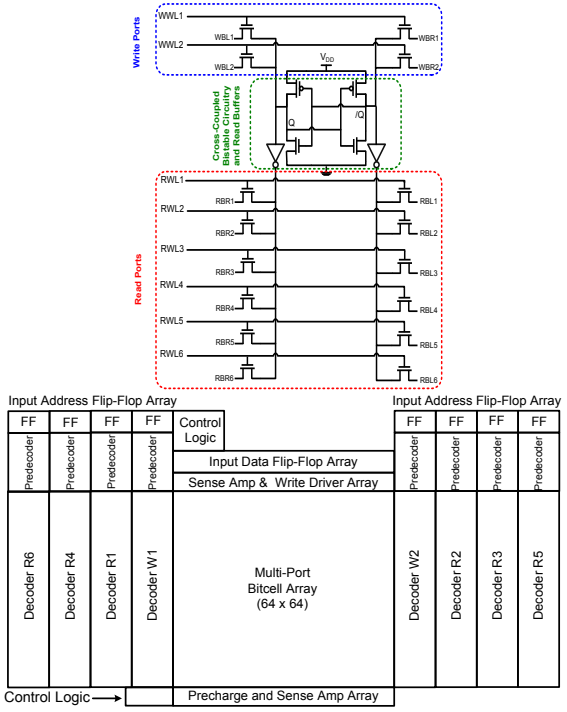
- Compute your height and width

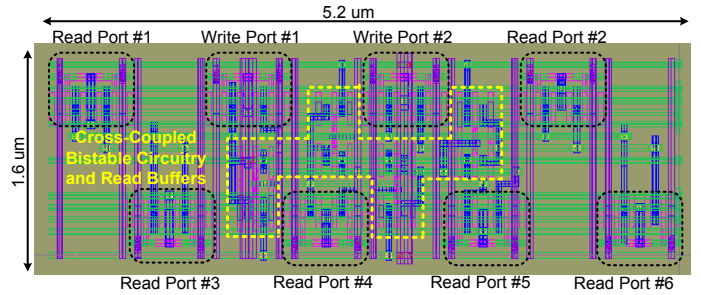
```
self.height = self.row_size * self.cell.height
self.width = self.column_size * self.cell.width
```

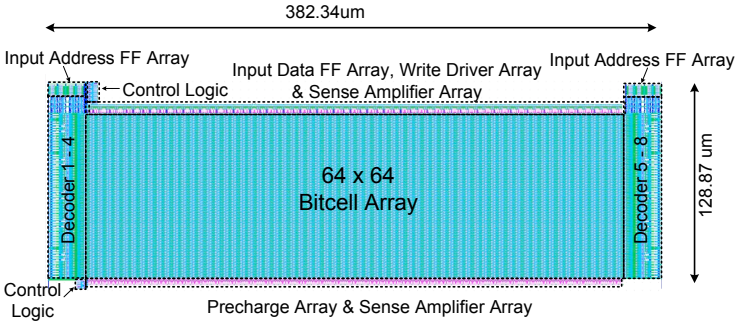
Outline

- Background on Memory Compilers
- OpenRAM Features
- OpenRAM Architecture and Circuits
- OpenRAM Usage
- OpenRAM Development
- How to port OpenRAM to a New Technology
- How to change the OpenRAM Circuits/Modules
- **Conclusion**

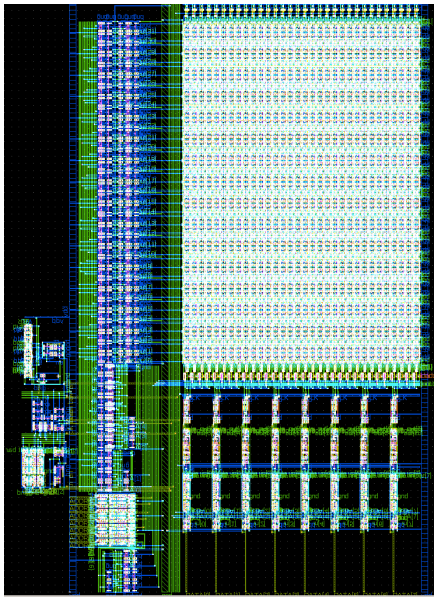
Generated Layout by OpenRAM for a Multiport (6R/2W) SRAM in 32 nm SOI CMOS Technology



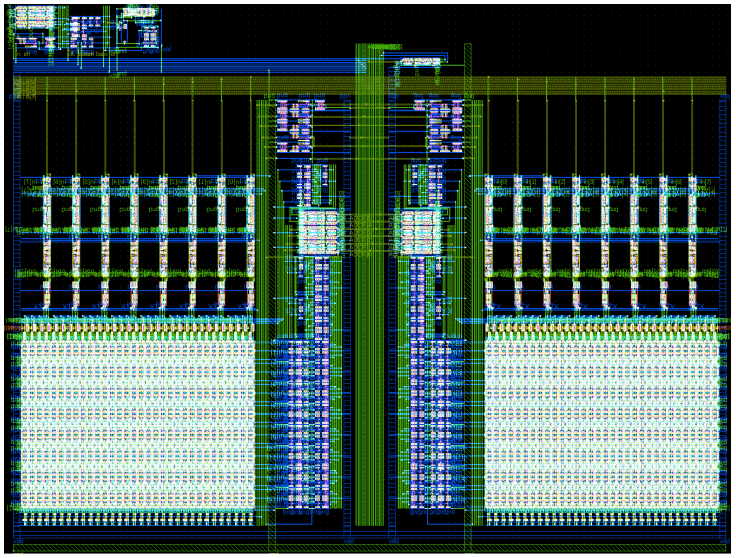




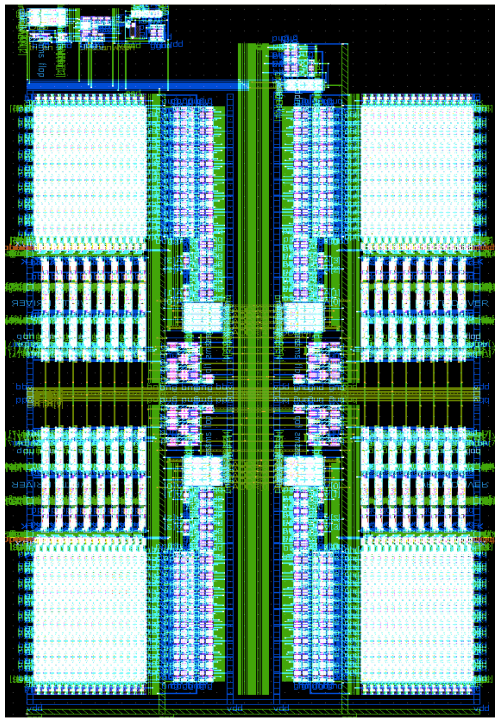
Multiple Bank Results



Single bank SRAM

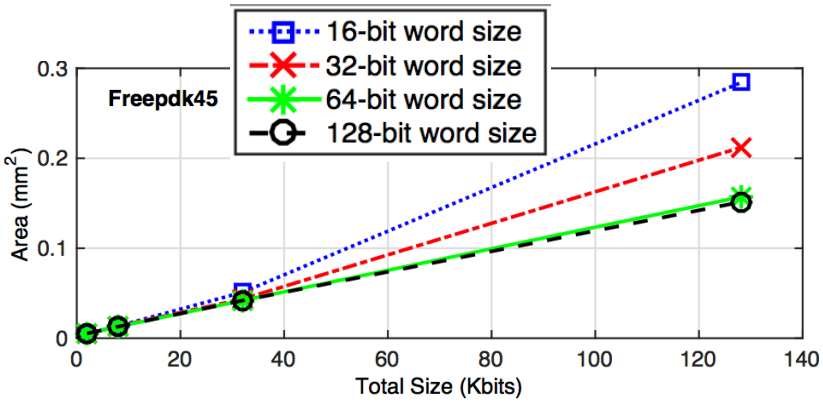


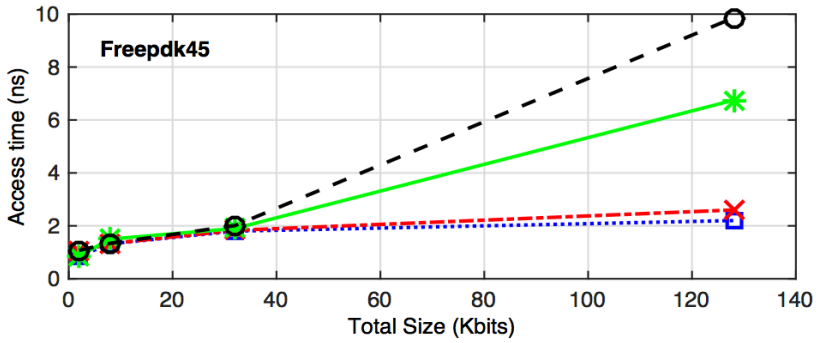
Dual bank SRAM

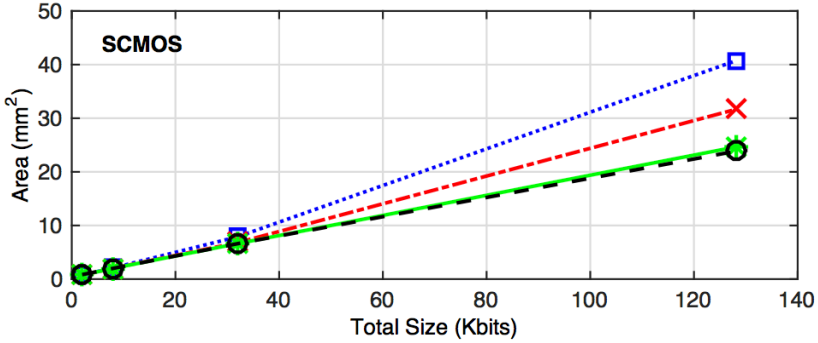


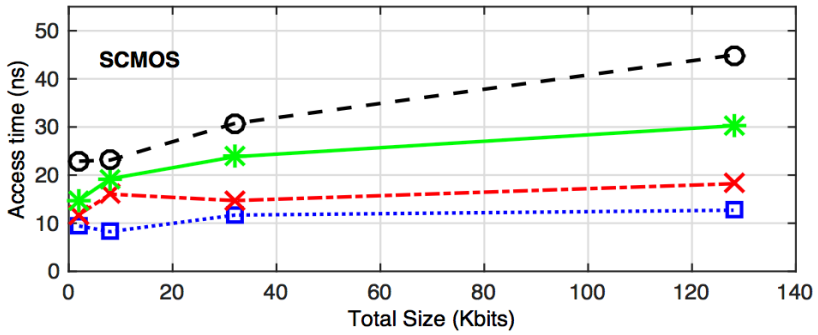
Quad bank SRAM

Timing and Density Results









Comparison with Fabricated SRAMs

Reference	Feature Size	Technology	Density [Mb/mm2]
IEEE-VLSI'08	65 nm	CMOS	0.7700
JSSC'11	45 nm	CMOS	0.3300
JSSC'13	40 nm	CMOS	0.9400
OpenRAM	45 nm	FreePDK45	0.8260
JSSC'92	0.5 um	CMOS	0.0036
JSSC'94	0.5 um	BICMOS	0.0020
JSSC'99	0.5 um	CMOS	0.0050
OpenRAM	0.5 um	SCMOS	0.0050

Conclusions

- The main motivation behind OpenRAM is to promote memory-related research in academia and provides a platform to implement and test new memory designs.
- OpenRAM is open-sourced, flexible, and portable and can be adapted to various technologies.
- OpenRAM generates the circuit, functional model, and layout of variable-sized SRAMs and provides a memory characterizer for synthesis timing/power models.
- We are also continuously introducing new features, such as non-6T memories, variability characterization, word-line segmenting, characterization speed-up, and a graphical user interface (GUI).
- We hope to engage an active community in the future development of OpenRAM.

Future Work

- Porting to FreePDK15
- Multi-port memories
- Internal clock buffers
- Variability analysis
- Create a maze router
- Alternate cells: 8T? 10T?

Acknowledgment

- Many thanks to National Science Foundation
  - This material is based upon work supported by the National Science Foundation under Grant No. CNS-1205685.
- We give thanks to other students who have contributed to the project, as well.
- We hope many will use and contribute to project!

Reminder: Getting OpenRAM

- https://github.com/mguthaus/OpenRAM
  - git clone https://github.com/mguthaus/OpenRAM.git
- https://openram.soe.ucsc.edu/
- Contact:
  - Prof. Matthew Guthaus (mrg@ucsc.edu)
  - Samira Ataei (ataei@okstate.edu)
  - Prof. James Stine (james.stine@okstate.edu)

OPENRAM

AN OPEN-SOURCE MEMORY COMPILER

Matthew R. Guthaus[1], James E. Stine[2], Samira Ataei[2]
Brian Chen[1], Bin Wu[1], Mehedi Sarwar[2]

[1] VLSI Design and Automation Group at UCSC

[2] VLSI Architecture Research Group at Oklahoma State University

UNIVERSITY OF CALIFORNIA
SANTA CRUZ