Memory

- Overview of Memories
- Memories on the FPGA
- Memories in Verilog
- External Memories
  - Flash
  - DRAM

Administrative

- Proposal Presentations are next week (scheduling is out!...figure it out!)
- Lab 05 is due tonight

Memories: a practical primer

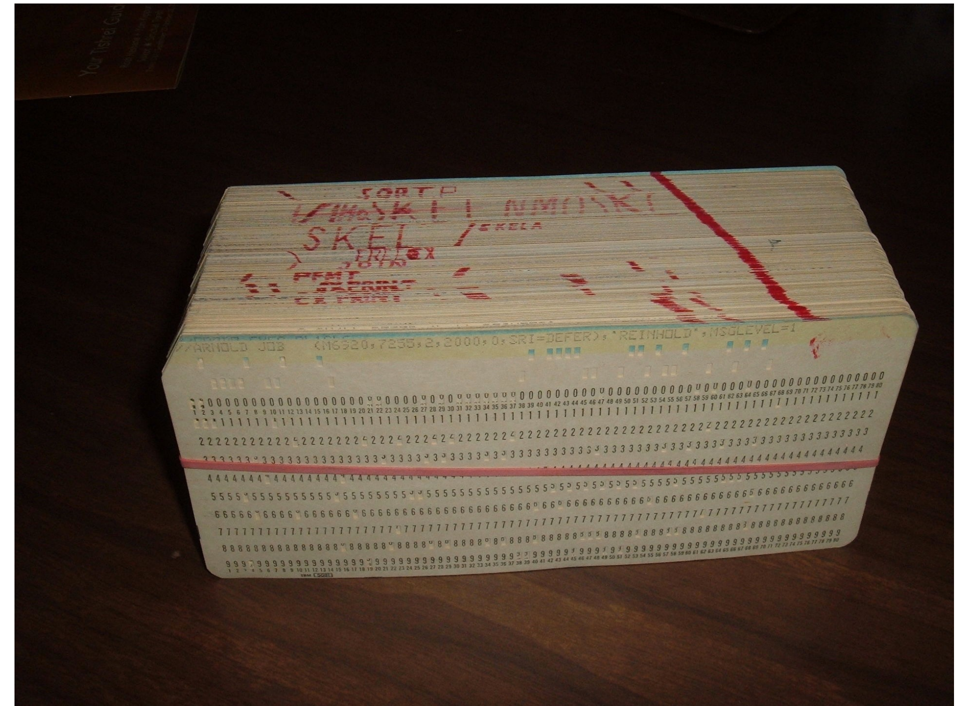
- The good news: huge selection of technologies
  - Small & faster vs. large & slower
  - Every year capacities go up and prices go down
  - Almost cost competitive with hard disks: high density, fast flash memories
    - Non-volatile, read/write, no moving parts! (robust, efficient)
- The bad news: perennial system bottleneck
  - Latencies (access time) haven’t kept pace with cycle times
  - Often a separate technology from logic, so must communicate between silicon, so physical limitations (# of pins, R’s and C’s and L’s) limit bandwidths
    - New hopes: capacitive interconnect, 3D IC’s, FRAMs, etc…
  - Likely one of the limiting factor in cost & performance of many digital systems: designers spend a lot of time figuring out how to keep memories running at peak bandwidth

How do we Electrically Remember Things?

- We can convey/transfer information with voltages that change over time
- How can we store information in an electrically accessible manner?
- Store in either:
  - Electric Field
  - Magnetic Field

Early attempts:

- Punched Cards have existed as electromechanical program storage since ~1800s
- Switches would sense holes in card and interpret as 1's and 0's
- We're mostly concerned with rewritable storage mechanisms today (cards were true ROMs)



Computer program in punched card format

https://en.wikipedia.org/wiki/Computer_programming_in_the_punched_card_era

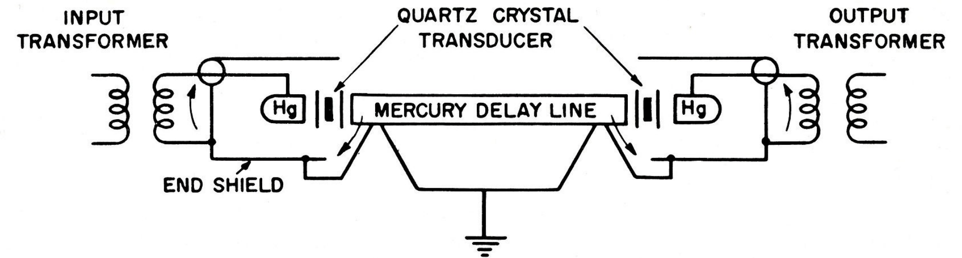
Electronic Memories in History

http://www.computerhistory.org/timeline/memory-storage/

- Drum Memory:
  - Information stored magnetically on large rotating metallic cylinder
  - Could read/write to it
- Did not require periodic refresh

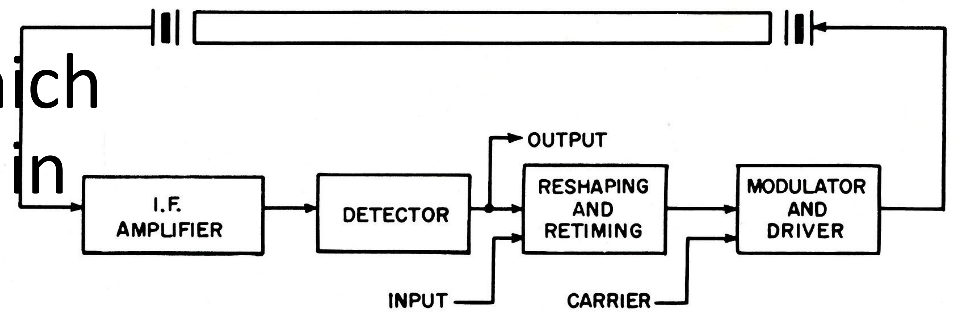
- Non-volatile (lasted after power cycles off)

Delay Line Memory

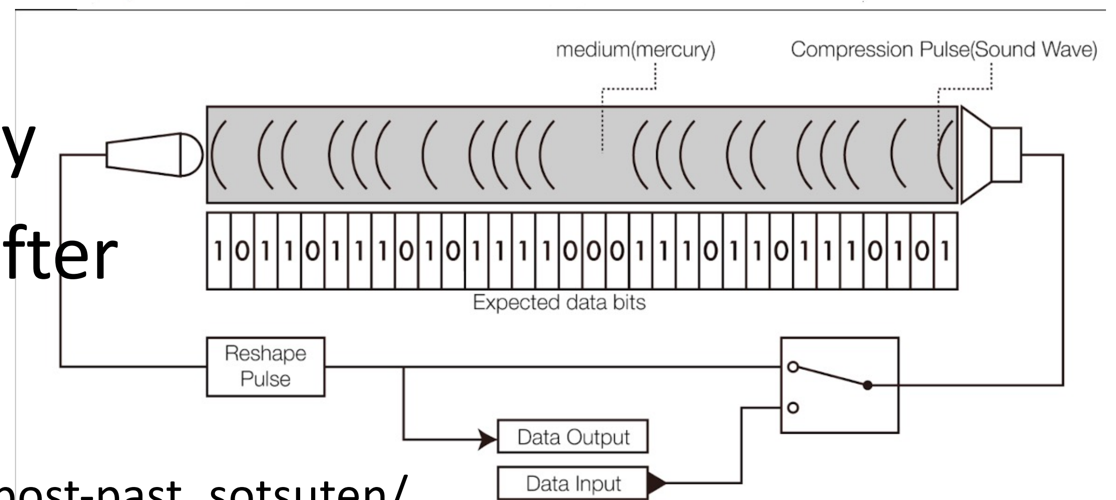


Schematic diagram of circuit connections to the acoustic delay line used in NBS mercury memory.

- Early form of FIFO memory (talk about later)
- Generate a wave pattern which exists for a few milliseconds in mercury
- Recover on the other end and either reload or use
- Requires refresh circuitry
- Volatile (info lost soon after power cut)



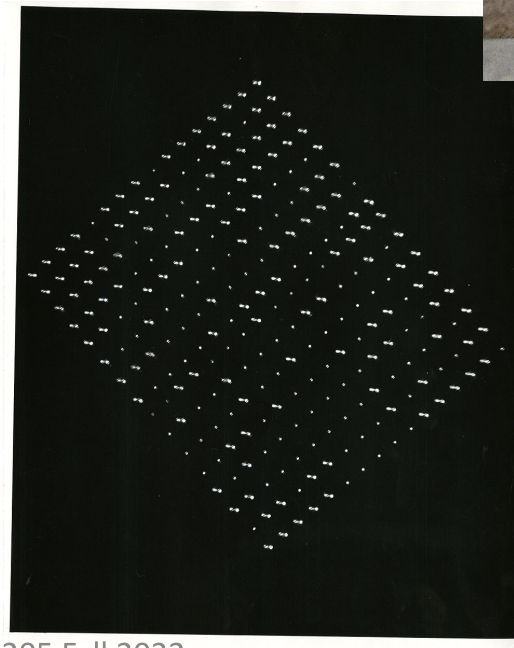
Block diagram of the mercury memory system.



https://matsuuratomoya.com/en/works/post-past_sotsuten/

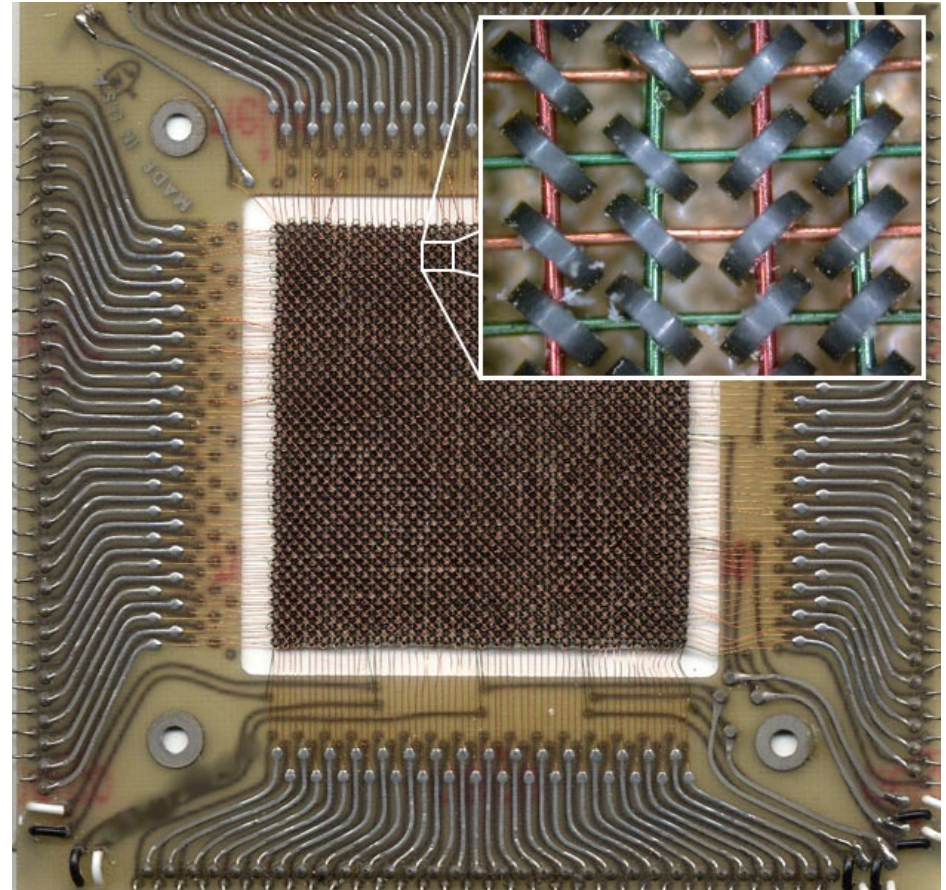
William’s Tube

- Take advantage of non-negligible decay time of phosphors on CRT to store data
- Project data image
- Little bit later (milliseconds) recover it .
- Either use it or re-project it for later use
- Requires periodic refresh



Core Memory

- MIT!
- Store 1’s and 0’s in the magnetic field of small torroids (magnetic cores)
- Where the term “core dump” comes from.
- Used up until mid 70’s
- Few on display in fourth floor of 38
- Non volatile!



https://en.wikipedia.org/wiki/Magnetic-core_memory#/media/File:KL_Kernspeicher_Makro_1.jpg

More Modern Memory

- Most modern memory uses some form of transistor-based object to maintain data in either a long or short term
- How is it done?

Memory Classification & Metrics

<table>
<tr><th colspan="2">Volatile Read-Write Memory</th><th rowspan="2">Non-Volatile Read-Write Memory</th><th rowspan="2">Non-Volatile Read-Only Memory</th></tr>
<tr><th>Random Access</th><th>Sequential Access</th></tr>
<tr><td>SRAM
DRAM</td><td>FIFO</td><td>EPROM
E^2PROM
FLASH</td><td>Mask-Programmed ROM</td></tr>
</table>

Key Design Metrics:

1. Memory Density (number of bits/mm^2) and Size
2. Access Time (time to read or write) and Throughput
3. Power Dissipation

Memory Classification & Metrics

Volatile Read-Write Memory		Non-Volatile Read-Write Memory	Non-Volatile Read-Only Memory
Random Access	Sequential Access		
SRAM DRAM	FIFO	EPROM E^2PROM FLASH	Mask-Programmed ROM

- **Random Access:** Give any address, get corresponding data. Access to memory need not be in a certain order
- **Sequential Access:** Put in values in an order, get them out in same order. Can't get or modify values at your desire…must wait for appropriate value to appear at ordered output (FIFO is an example)

Memory Classification & Metrics

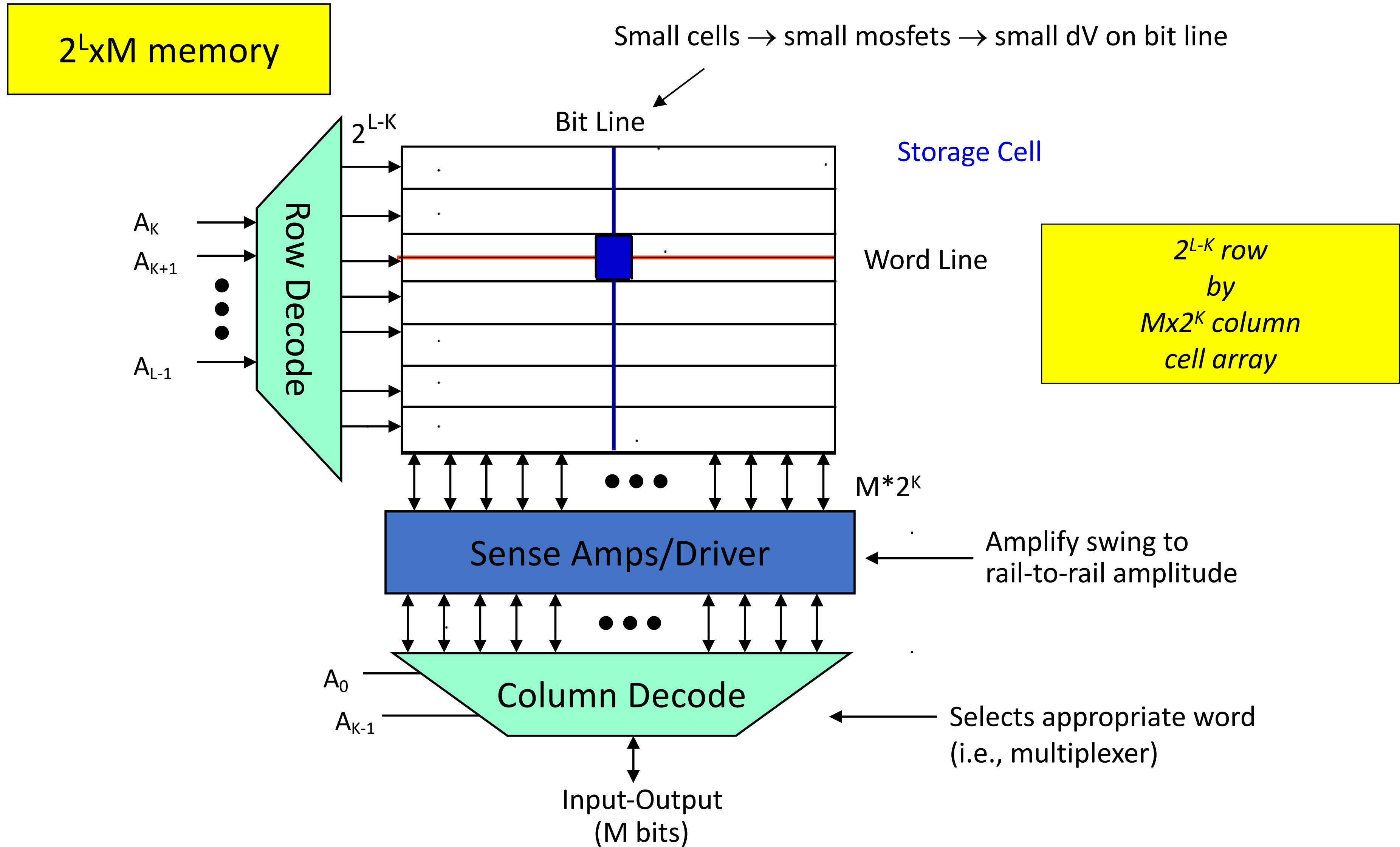
Volatile Read-Write Memory		Non-Volatile Read-Write Memory	Non-Volatile Read-Only Memory
Random Access	Sequential Access		
SRAM DRAM	FIFO	EPROM E^2PROM FLASH	Mask-Programmed ROM

- **Volatile:** Maintains data only as long as power is applied
- **Non-Volatile:** Maintains data after power is applied!

Memory Density

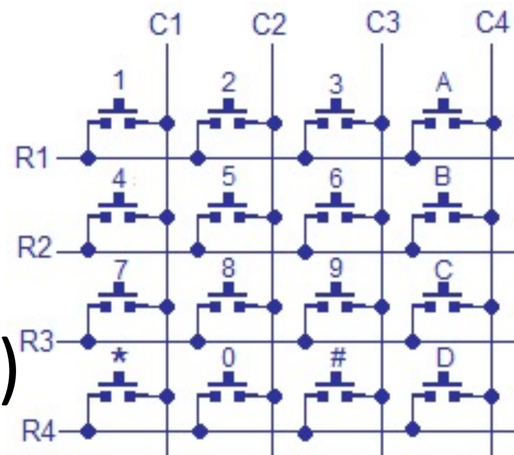
- High-density memory technologies rarely enable "direct" access to anything inside of them.
- There's just too many wires that would be needed and you wouldn't be able to be very dense.
- Instead the memory-storage technology (transistors or whatever) are usually built into large grids which are accessed in a row-column format.
- This has implications for reading and writing!!!

Memory Array Architecture (SRAM, Flash, DRAM)



Memory Array’s (Inspiration in Switches)

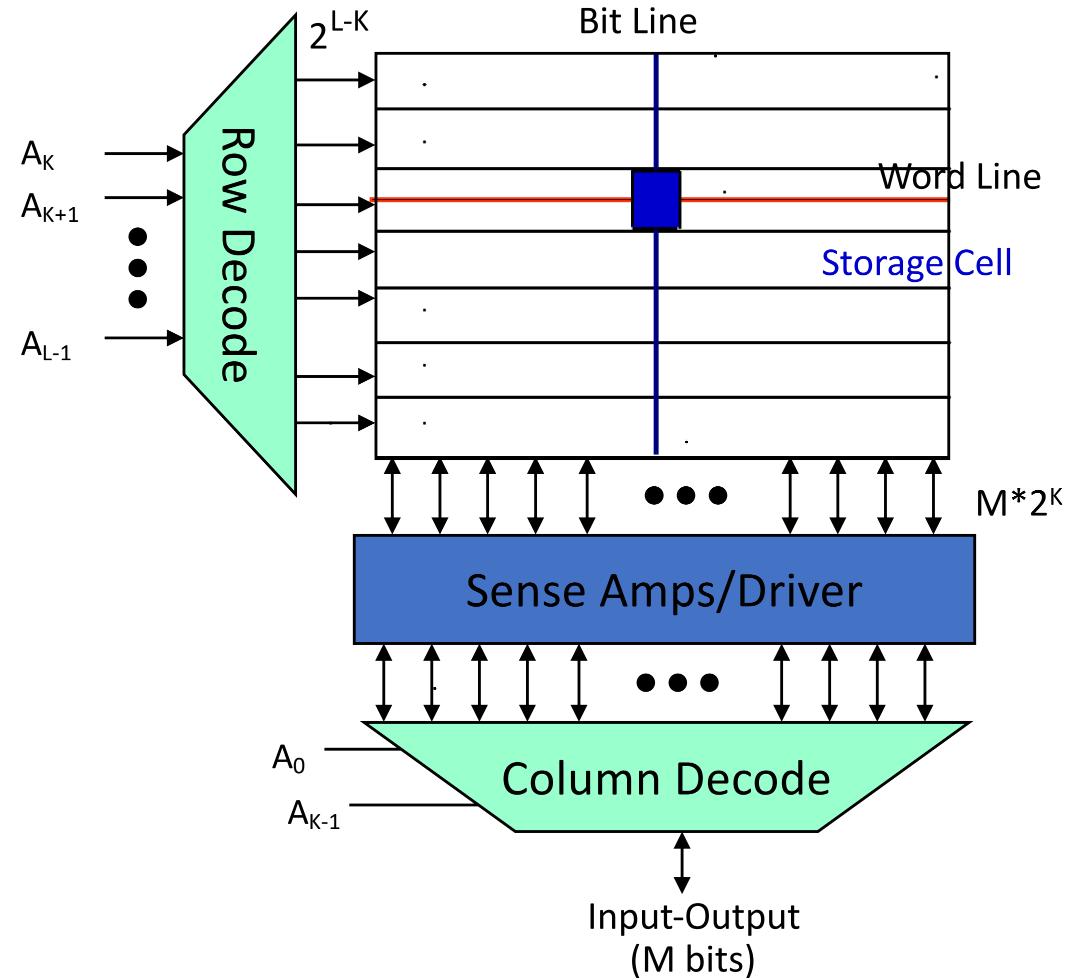
- If you have 16 switches, you can convey that using 16 independent wires (one-hot encoding)
- Alternatively if you assemble in an array/matrix, you can do with 8 wires (if you add some interfacing circuitry)
- Same situation in memory



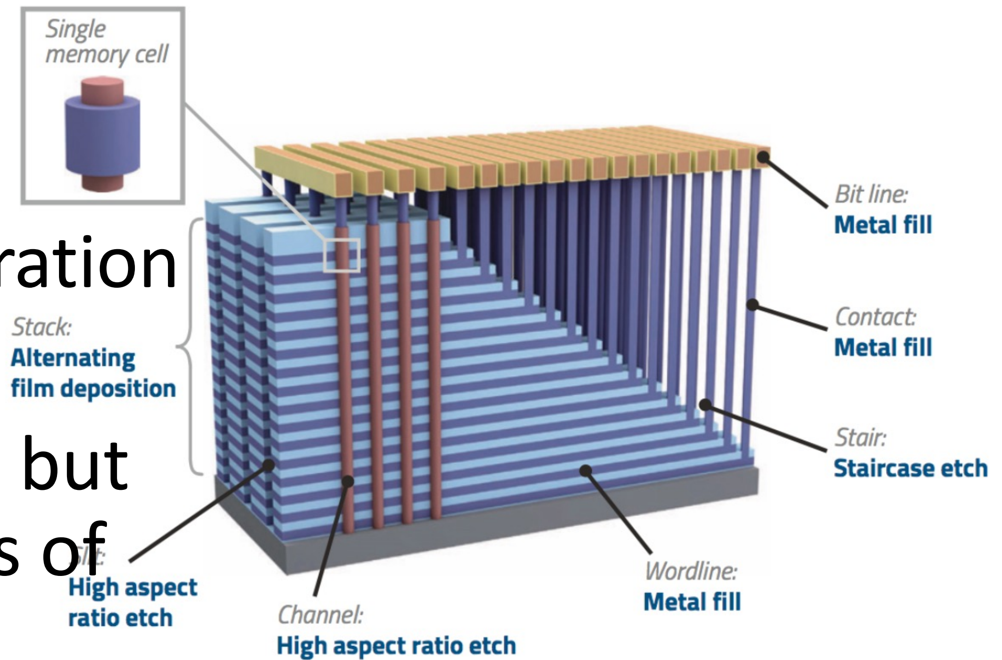
With correct interfacing you can still think of this as a 16X1 array of switches!!! Even though it isn’t

As a result…

- Can't simultaneously access multiple locations.
- In most technologies you can access one (or maybe two) entries at any point in time!
- In some layouts reading out two nearby addresses is easier/faster than reading out two addresses in different spots.



3D Memory



- Last decade has seen proliferation of 3D memory architectures.
- Same rough technology idea, but instead of planes, go to cubes of memory.
- Much higher densities.
- Still can only access a few spots at one time

https://sst.semiconductor-digest.com/2017/07/overcoming-challenges-in-3d-nand-volume-manufacturing/

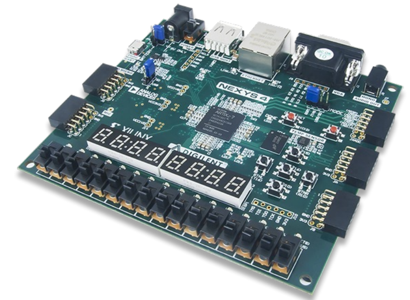
Nexys4 DDR Memory

Inside the FPGA

- Regular registers in logic blocks
  - Operates at system clock speed, expensive (CLB utilization)
  - Configuration set by Verilog design (eg FIFO, single/dual port, etc)
- FPGA Distributed memory
  - Operates at system clock speed
  - Uses LUTs (64 bits) for implementation, expensive (CLB utilization)
  - Requires significant routing for implementation
  - Configured using IP
  - Theoretical maximum: ~1Mbit
- FPGA Block RAM:
  - 4,860K bits total (in 135/270 chunks)

Outside the FPGA

- DDR2 SDRAM
  - 128MiB (Megabytes)
  - Requires MIG (Memory Interface Generator) Wizard
- Flash memory
  - 16MiB
  - Slow read access, even slower write access time!
- microSD port
  - Tested with 2GB (Windows 7, FPGA)



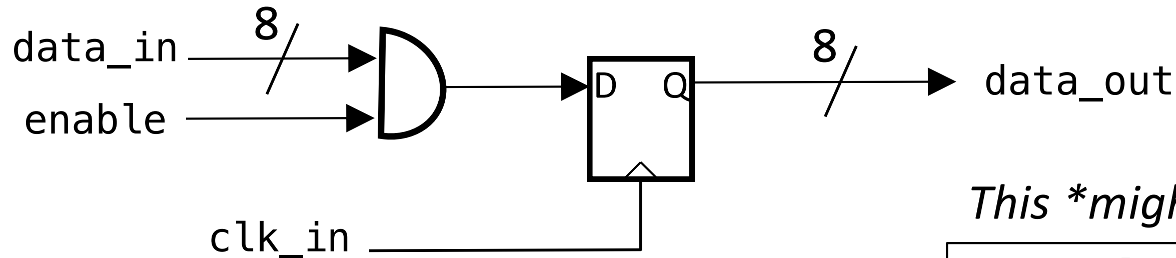
Memory <u>IN</u> the FPGA

FPGA Memory: Two Types

- The FPGA has two dedicated sets of resources (other than Flipflops) for storing information. All are comprised of SRAM (Static Random-Access Memory)

Hold on…Aren't Flipflops Memory?

- Yes, they are memory and you can use them like this:

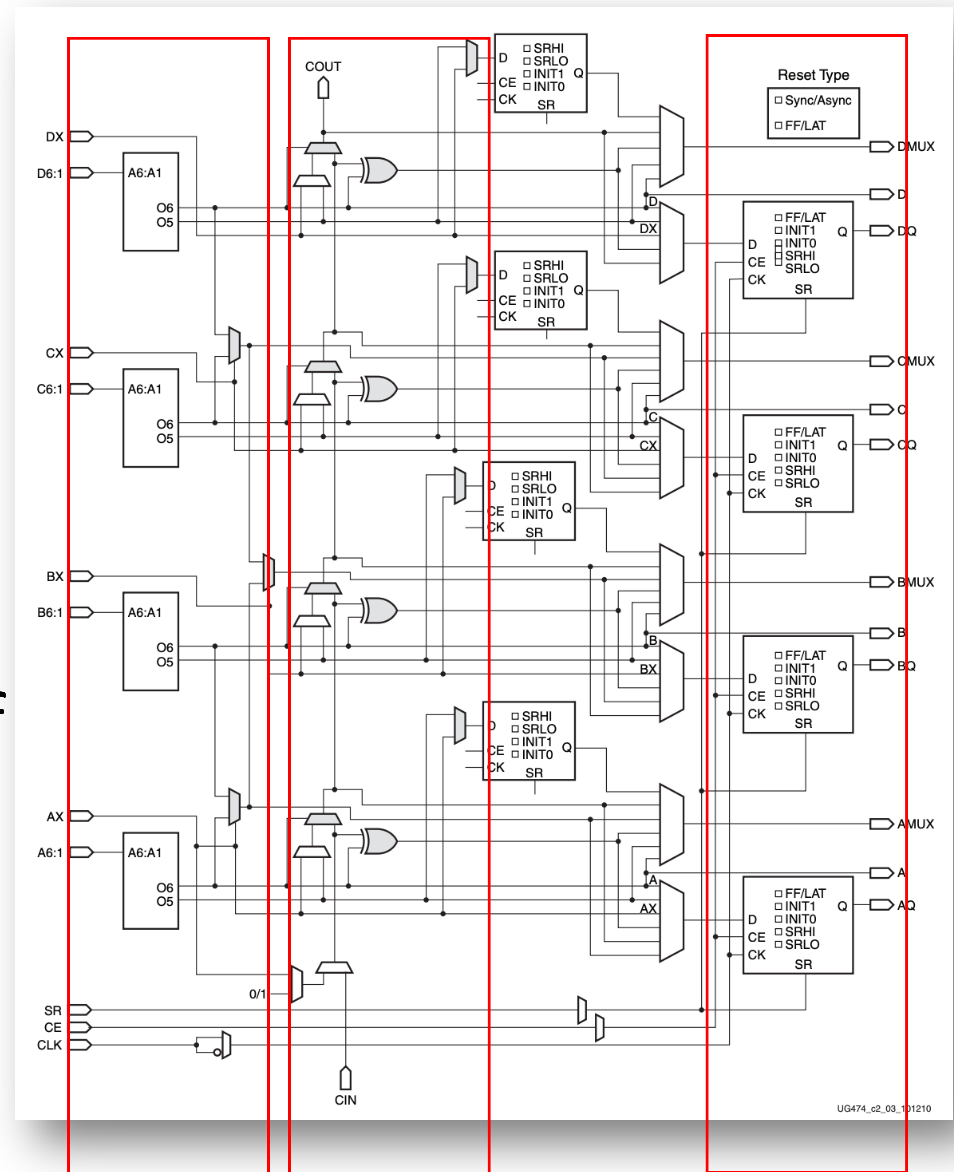


*This *might* synthesize using flip flops*

```
logic [7:0] storage;
logic [7:0] data_in;
logic enable;
logic [7:0] data_out;
assign data_out = storage;
always_ff @(posedge clk_in)begin
  if (enable)begin
    storage <= data_in;
  end
end
```

Flip Flops

- Flip flops are distributed all over the board in the logic cells
- Nearby for convenience
- Are meant for holding smaller temporary chunks of data
- Flip flops are not meant for bulk storage… (an image, for example)



LUTs

Used to synthesize all combinational stuff

Fast carry chain

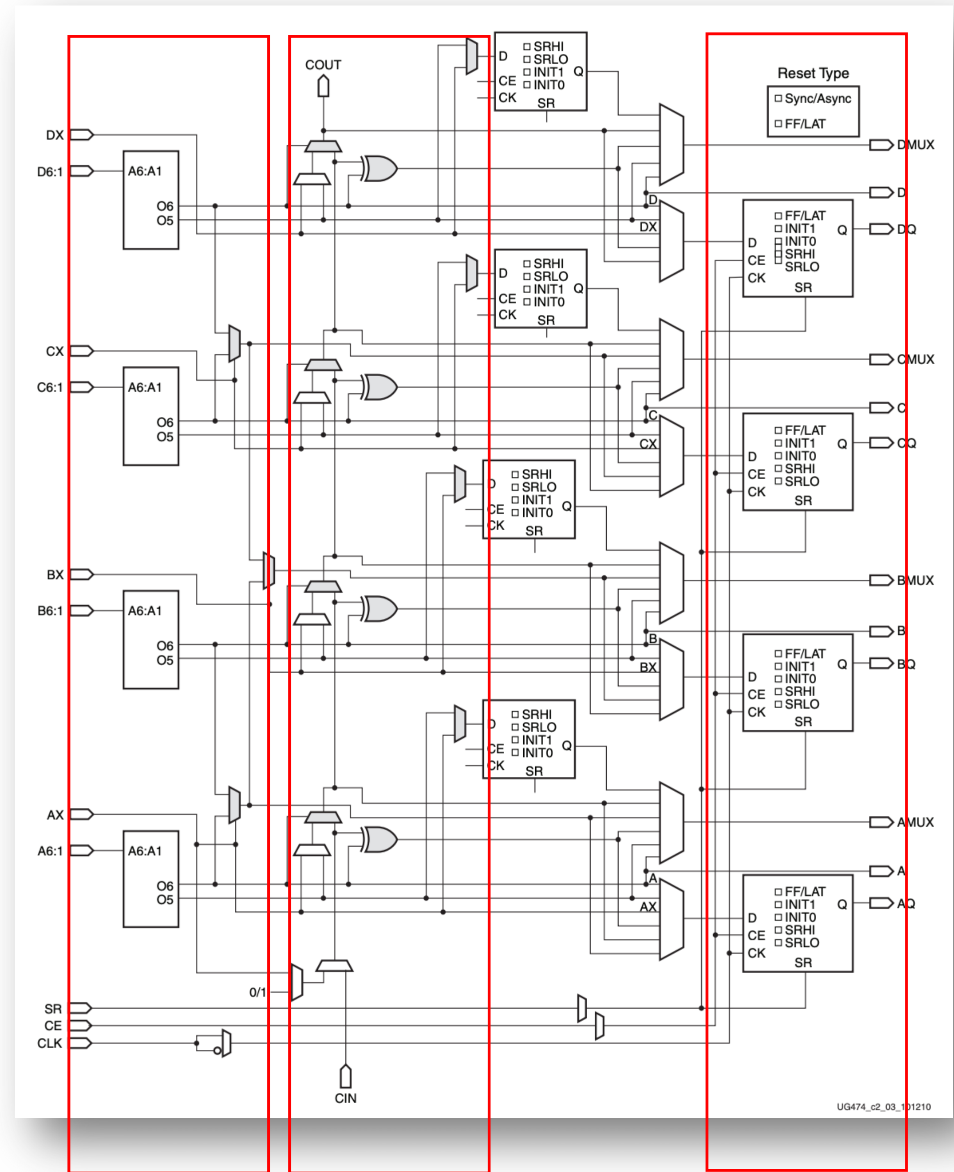
For mult-slice logic (addition, etc)

FF/Latches

Route through these for registers. Else bypass for purely combinational

Flip Flops

- Think of nearby flipflops as the registers you see in a processor
- Quick and relatively small memory access units
- Nearby so easy to route to
- Immediately accessible (not living in dense piles in which only one entry can be read at a time)
- But what about more memory?



LUTs

Used to synthesize all combinational stuff

Fast carry chain

For mult-slice logic (addition, etc)

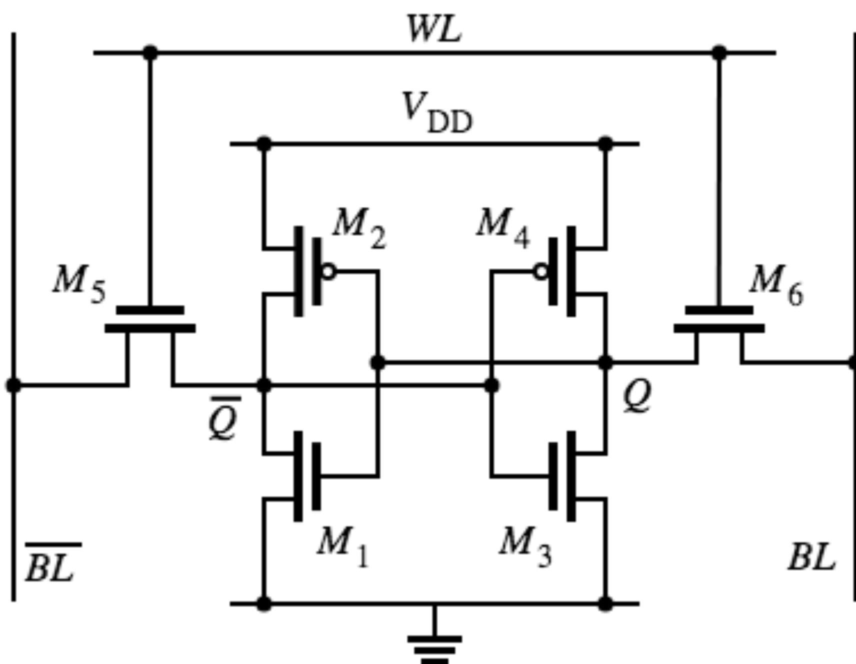
FF/Latches

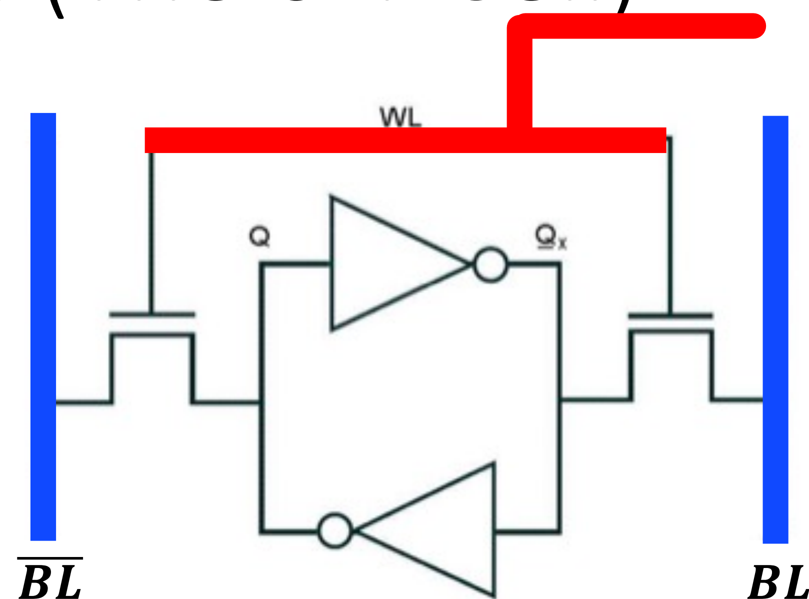
Route through these for registers. Else bypass for purely combinational

FPGA Internal Memory: Two Types

- The FPGA has two dedicated sets of resources for storing information in larger quantities
  - Block RAM
  - Distributed RAM
- All are comprised of SRAM (Static Random-Access Memory)

Static RAM (SRAM) Cell (The 6-T Cell)





Write: Set BL, $\overline{\text{BL}}$ to (0,V_{DD}) or (V_{DD},0) then enable WL (= V_{DD})

Read: Disconnect drivers from BL and $\overline{\text{BL}}$, then enable WL (=V_{DD}). Sense a small change in BL or $\overline{\text{BL}}$

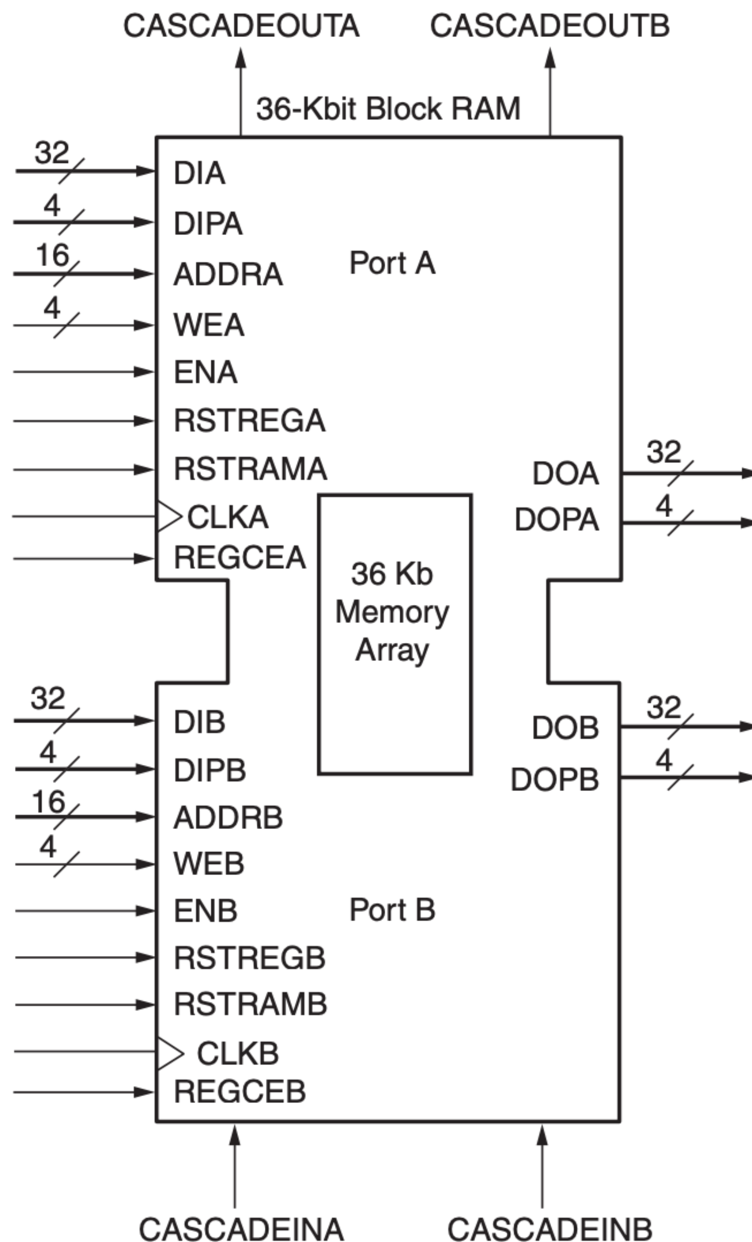
- State held by cross-coupled inverters (M1-M4)
- Retains state as long as power supply turned on
- Feedback must be overdriven to write into the memory

FPGA Memory: Two Types

- The SRAM in our FPGA (Xilinx 7A100T) is organized into two types (meant for using as memory explicitly):
- **<u>Block RAM (BRAM):</u>**
  - Large continuous chunks of SRAM
  - 36 kbits a piece
  - 135 of these on our particular FPGA
- **<u>Distributed RAM:</u>**
  - Of the ~15,000 Logic Slices on the FPGA, about 5000 have 256 bits of SRAM in them that is usable
  - Can use this spread-out RAM as well (to squeeze another ~150 KBytes out of chip…but this takes away resources from your logic so you should use as last resort!

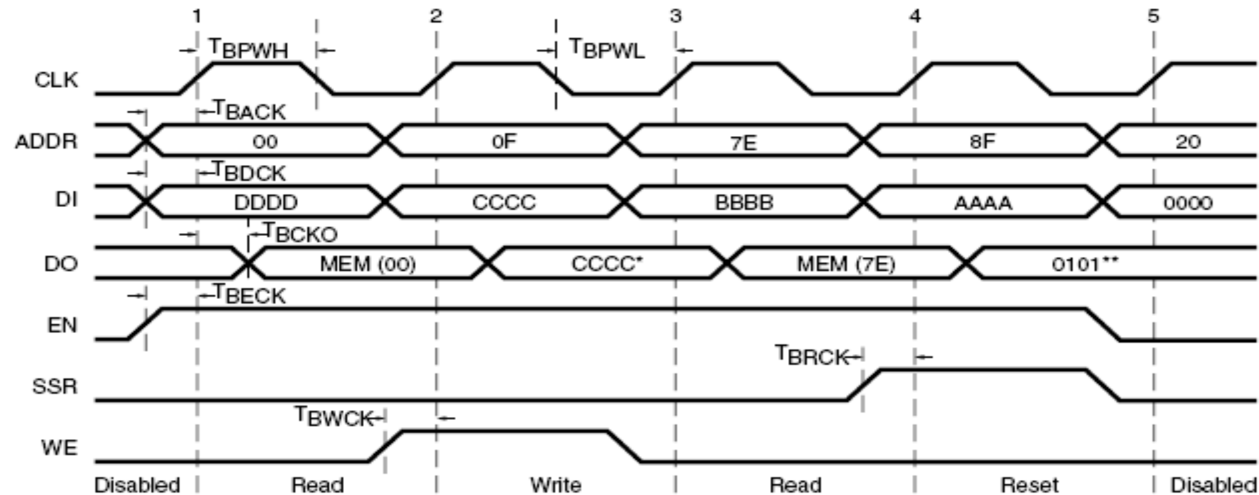
Block Memories (BRAMs)

There's 135 of these 36Kx1 bit SRAM arrays



- Our FPGA has 135 dual-port SRAM modules
- Can write-to and lookup values using these two ports as needed

BRAM Timing



Block SelectRAM Timing Diagram

Block SelectRAM Switching Characteristics

Description	Symbol	Speed Grade -6	Speed Grade -5	Speed Grade -4	Units
Sequential Delays					
Clock CLK to DOUT output	T_{BCKO}	2.10	2.31	2.65	ns, Max
Setup and Hold Times Before Clock CLK					
ADDR inputs	T_{BACK}/T_{BCKA}	0.29/ 0.00	0.32/ 0.00	0.36/ 0.00	ns, Min
DIN inputs	T_{BDCK}/T_{BCKD}	0.29/ 0.00	0.32/ 0.00	0.36/ 0.00	ns, Min
EN input	T_{BECK}/T_{BCKE}	0.95/–0.46	1.04/–0.50	1.20/–0.58	ns, Min
RST input	T_{BRCK}/T_{BCKR}	1.31/–0.71	1.44/–0.78	1.65/–0.90	ns, Min
WEN input	T_{BWCK}/T_{BCKW}	0.57/–0.19	0.63/–0.21	0.72/–0.25	ns, Min
Clock CLK					
CLKA to CLKB setup time for different ports	T_{BCCS}	1.0	1.0	1.0	ns, min
Minimum Pulse Width, High	T_{BPWH}	1.17	1.29	1.48	ns, Min
Minimum Pulse Width, Low	T_{BPWL}	1.17	1.29	1.48	ns, Min

BRAM Operation

*Similar to what we did earlier with SRAM (BRAM is just a **B**lock of **SRAM** on the FPGA)*

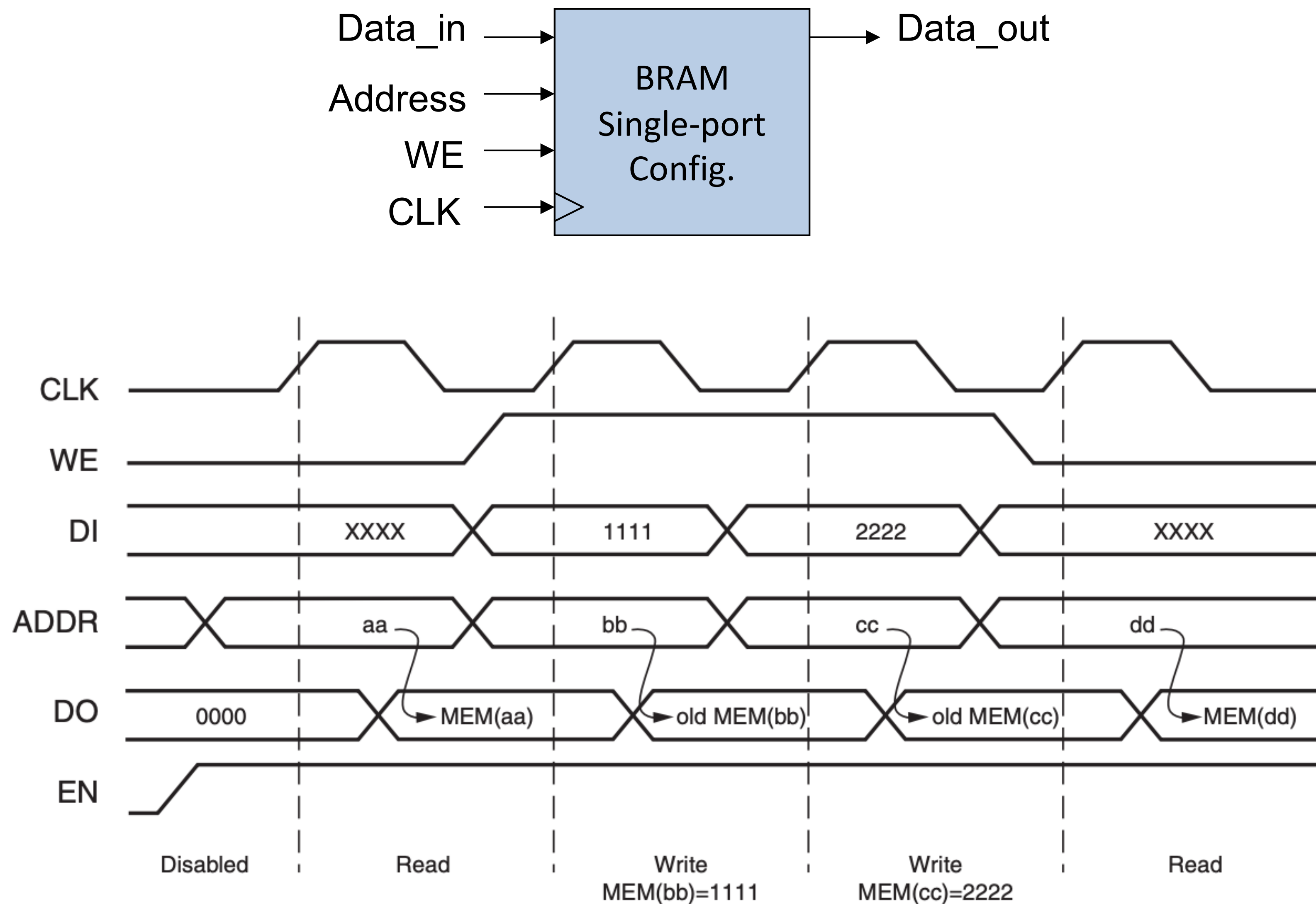
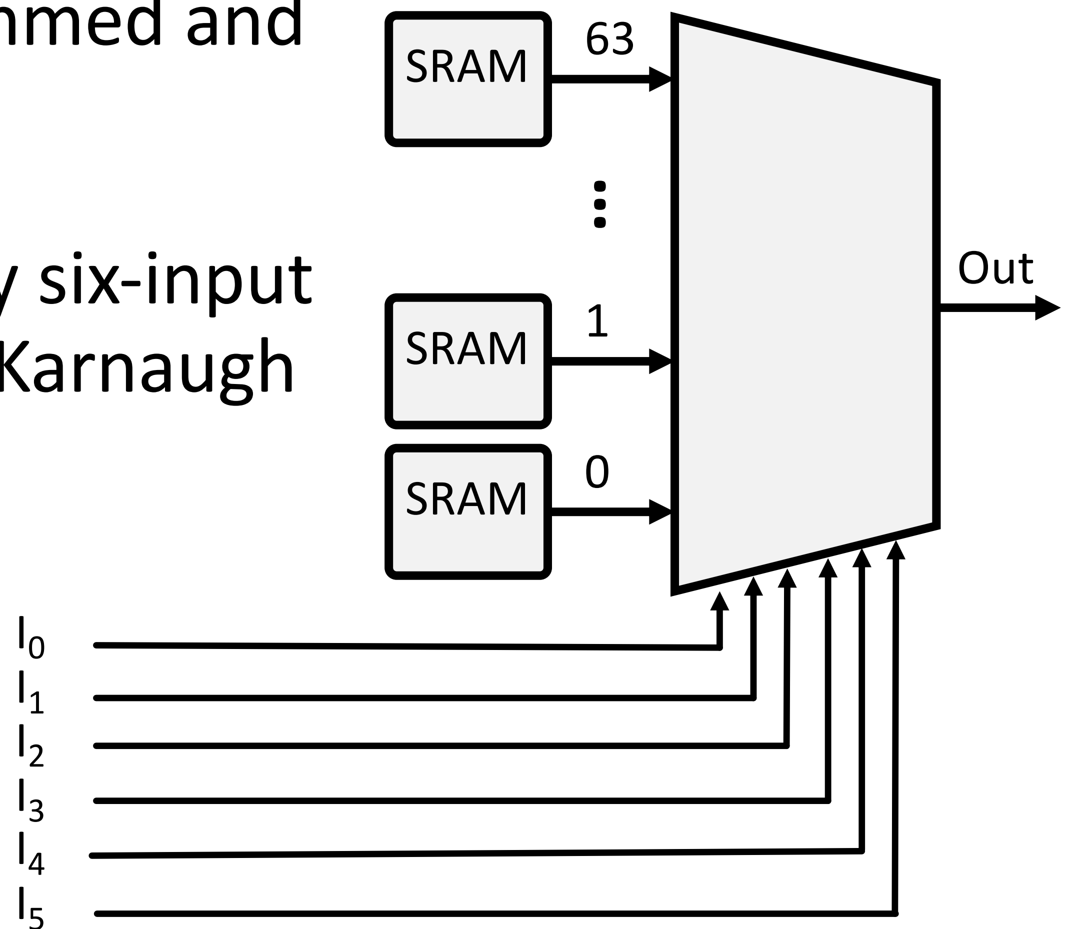


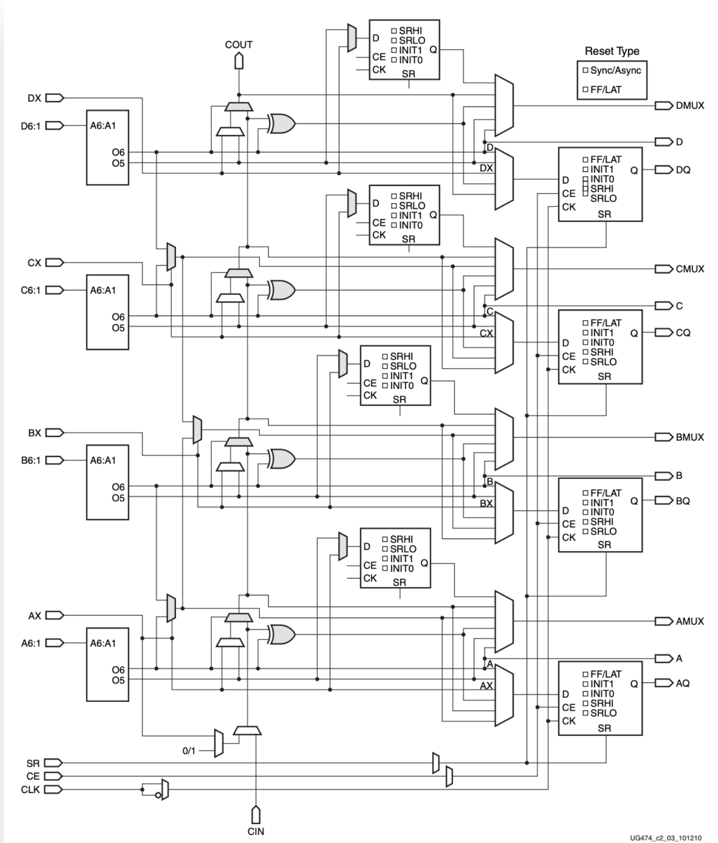
Figure 1-3: **READ_FIRST Mode Waveforms**

Distributed RAM: Each Logic Cell is made of Four Six-Input Lookup Tables with inputs that can be set

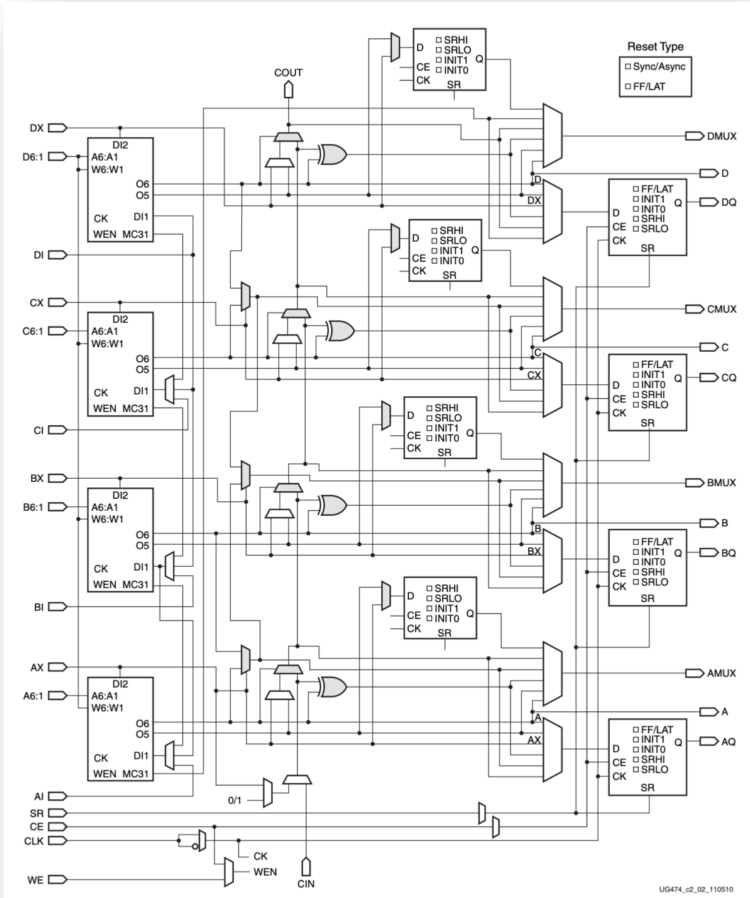
- These LUTs are programmed and that program is set in SRAM...they can thereforesynthesize any six-input lookup-table/function/Karnaugh Map
- In some logic cells, you can alternatively use this SRAM for memory! …change it over time



SliceL vs. Slice M



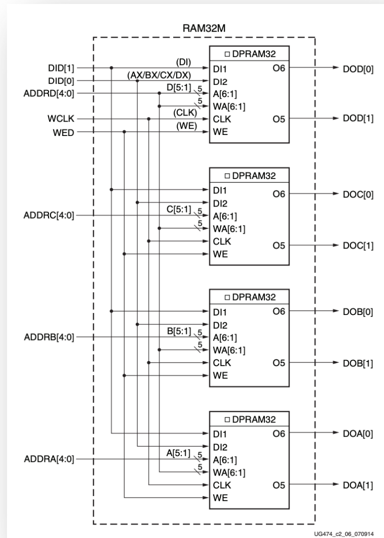
SliceL
LUTs programmed when bitfile written



SliceM
Memory used in LUT programming broken out and available

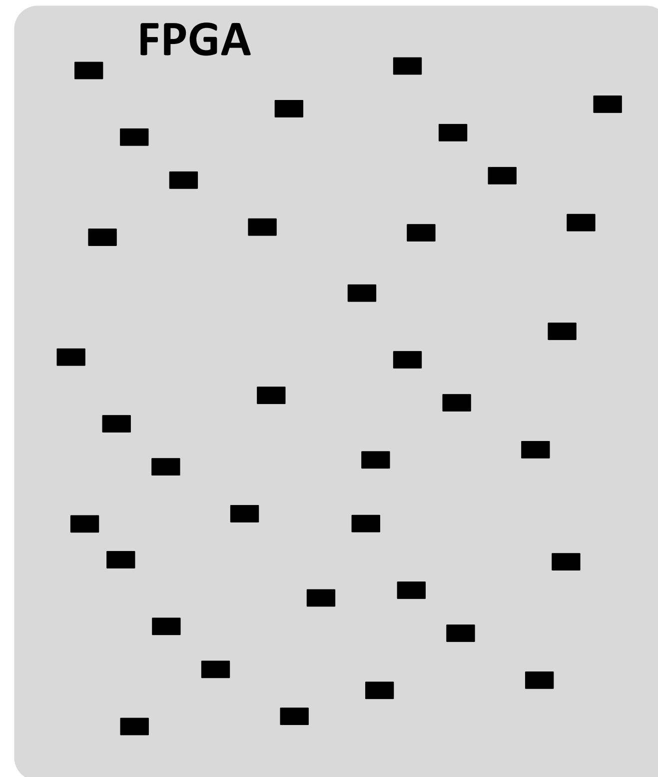
Distributed RAM is Distributed

- Each 64 bits of LUT specification is broken out…so each Slice (with Four LUT6's) has 256 bits of RAM



Four of these per slice

SliceM's are *distributed all over the FPGA:*



Distributed RAM vs. Block RAM

- Distributed RAM:
  - More flexible:
    - Smaller unit size (256 bits)
    - read multiple (>2) values at once
    - Single-cycle reads/writes
    - Single-cycle resets (bulk reset)
    - Often a lot of things we “think” we want when designing!
- Block RAM:
  - Less flexible:
    - Bigger unit size (18kbits)
    - Dual-port ONLY (<=2)
    - Risky single-cycle reads/writes
    - No single-cycle bulk reset

Distributed RAM vs. Block RAM

- Distributed RAM:
  - Trading logic for memory
  - Occupies a logic cell whenever you use it (more you use, the fewer logic cells you have for logic)
  - For large data structures/memories, needs to use lots of memory units (can be hard to route)
  - Too much usage can make it hard to place and route (long builds!)
- Block RAM:
  - Meant to be memory and nothing else
  - Relatively dense!
  - Use it or lose it. It is there…if you don't use it, you don't get free logic instead so you should use it.
  - For large things, will outperform distributed RAM in speed/latency
  - Generally won't constrain place and route like distributed RAM usage will!

Distributed RAM vs. Block RAM

- Conclusions:
- Distributed RAM:
  - Small things (16 bit shift register), small memories of a few bytes, few entries, etc… are all best implemented with Distributed RAM:
- Block RAM:
  - Large things (images, large audio files, etc), large buffers, all best implemented with Block RAMs

How do We Specify Which to Use?

- We do it with how we write Verilog.
- The code below specifies a small, low-latency 8X8 memory:

- This will get implemented using Distributed RAM most likely
- In particular these:
  - Instant lookup
  - Bulk reset
- Push it to be done with Dist RAM

Instant lookup

Bulk reset

```
logic [7:0] storage [7:0];
logic [2:0] addr;
logic [7:0] data_in;
logic enable;
logic [7:0] data_out;
assign data_out = storage[addr];
always_ff @(posedge clk_in)begin
  if (rst_in)begin
    for(int i=0; i<8; i=i+1)begin
      storage[i]<=0;
    end
  end else if (enable)begin
    storage[addr] <= data_in;
  end
end
```

Building a Block RAM

- Use Verilog in a very particular way, Vivado can confidently “infer” block RAM usage
- Can do with Verilog natively, however it is very easy to ask for too much flexibility in which case distributed RAM will get used.
- Instead use pre-provided templates:

```
module xilinx_true_dual_port_read_first_2_clock_ram #(
parameter RAM_WIDTH = 18, // Specify RAM data width
parameter RAM_DEPTH = 1024, // Specify RAM depth (number of entries)
parameter RAM_PERFORMANCE = "HIGH_PERFORMANCE", // Select "HIGH_PERFORMANCE" or "LOW_LATENCY"
parameter INIT_FILE = "" // Specify name/location of RAM initialization file if using one (leave blank
if not)
) (
input [clogb2(RAM_DEPTH-1)-1:0] addra, // Port A address bus, width determined from RAM_DEPTH
input [clogb2(RAM_DEPTH-1)-1:0] addrb, // Port B address bus, width determined from RAM_DEPTH
input [RAM_WIDTH-1:0] dina, // Port A RAM input data
input [RAM_WIDTH-1:0] dinb, // Port B RAM input data
input clka, // Port A clock
input clkb, // Port B clock
input wea, // Port A write enable
input web, // Port B write enable
input ena, // Port A RAM Enable, for additional power savings, disable port when not in use
input enb, // Port B RAM Enable, for additional power savings, disable port when not in use
input rsta, // Port A output reset (does not affect memory contents)
input rstb, // Port B output reset (does not affect memory contents)
input regcea, // Port A output register enable
input regceb, // Port B output register enable
output [RAM_WIDTH-1:0] douta, // Port A RAM output data
output [RAM_WIDTH-1:0] doutb // Port B RAM output data
);
```

Building a Block RAM

- If we use our Verilog in a very particular way, Vivado can confidently “infer” block RAM usage

```
xilinx_true_dual_port_read_first_2_clock_ram #(
.RAM_WIDTH(32),
.RAM_DEPTH(1024))
frame_buffer (
  //Write Side (16.67MHz)
  .addra(addr_count+3),
  .clka(axi_clk), //NEW FOR LAB 04B
  .wea(sqrt_valid),
  .dina({8'b0,sqrt_data}),
  .ena(1'b1),
  .regcea(1'b1),
  .rsta(btnd),
  .douta(),
  //Read Side (65 MHz)
  .addrb(draw_addr),
  .dinb(16'b0),
  .clkb(pixel_clk),
  .web(1'b0),
  .enb(1'b1),
  .rstb(btnd),
  .regceb(1'b1),
  .doutb(amp_out)
);
```

This (lec 10), leads to this usage:

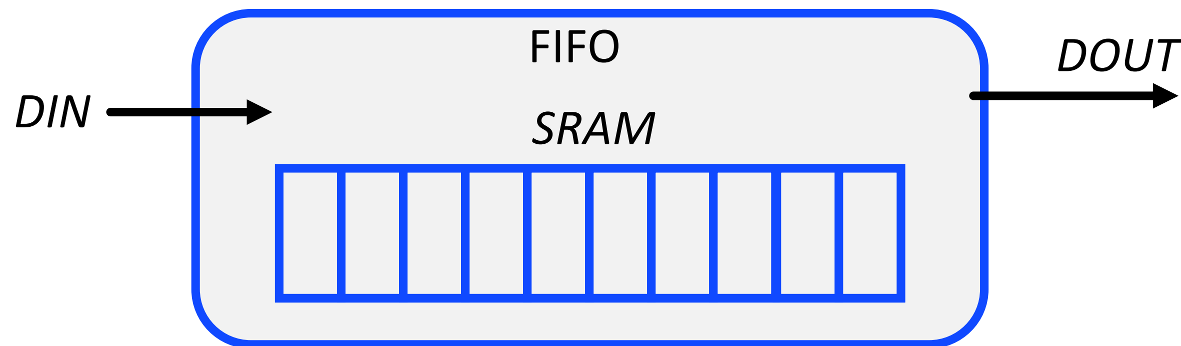
```
3. Memory
---------

+-------------------+------+-------+-----------+-------+
|     Site Type     | Used | Fixed | Available | Util% |
+-------------------+------+-------+-----------+-------+
| Block RAM Tile    |    5 |     0 |       135 |  3.70 |
|   RAMB36/FIFO*    |    2 |     0 |       135 |  1.48 |
|     RAMB36E1 only |    2 |       |           |       |
|   RAMB18          |    6 |     0 |       270 |  2.22 |
|     RAMB18E1 only |    6 |       |           |       |
+-------------------+------+-------+-----------+-------+
* Note: Each Block RAM Tile only has one FIFO logic available and
therefore can accommodate only one FIFO36E1 or one FIFO18E1.
However, if a FIFO18E1 occupies a Block RAM Tile, that tile can
still accommodate a RAMB18E1
```

RAM Uses

- Store Images (Lab03)
- Store Audio
- Store Video (Lab 4/4b) for frame buffer (two-port RAM)
- Clock Domain Crossing:
  - With dual-port RAM, you can write with one clock and read with another (can specify some settings to prevent race conditions, though no 100% guarantee)
- Store *anything you want! (exciting)*

FIFO (First-In-First-Out)

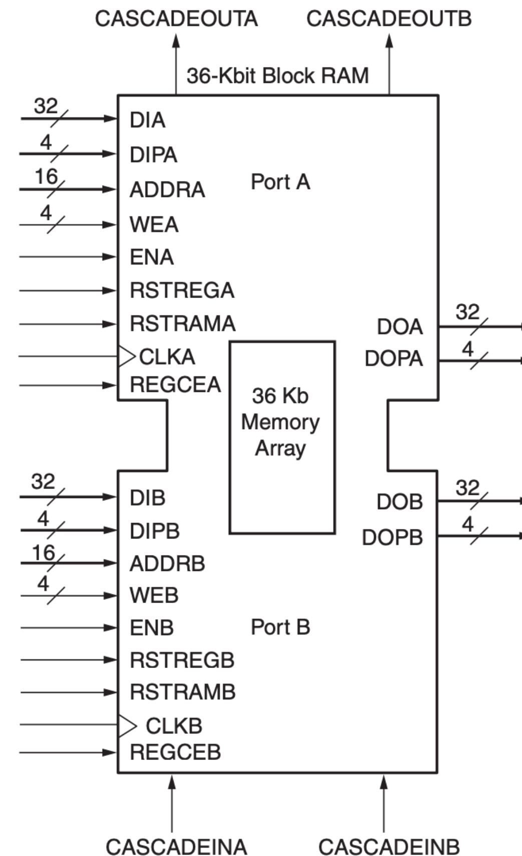
- Basically a Queue like you see in Python or something, but we can’t dynamically allocate storage space ahead of time at our low level!



- Data is not randomly accessed, but instead is accessed in the order it was provided
- Can generate either using Dist RAM or BRAM

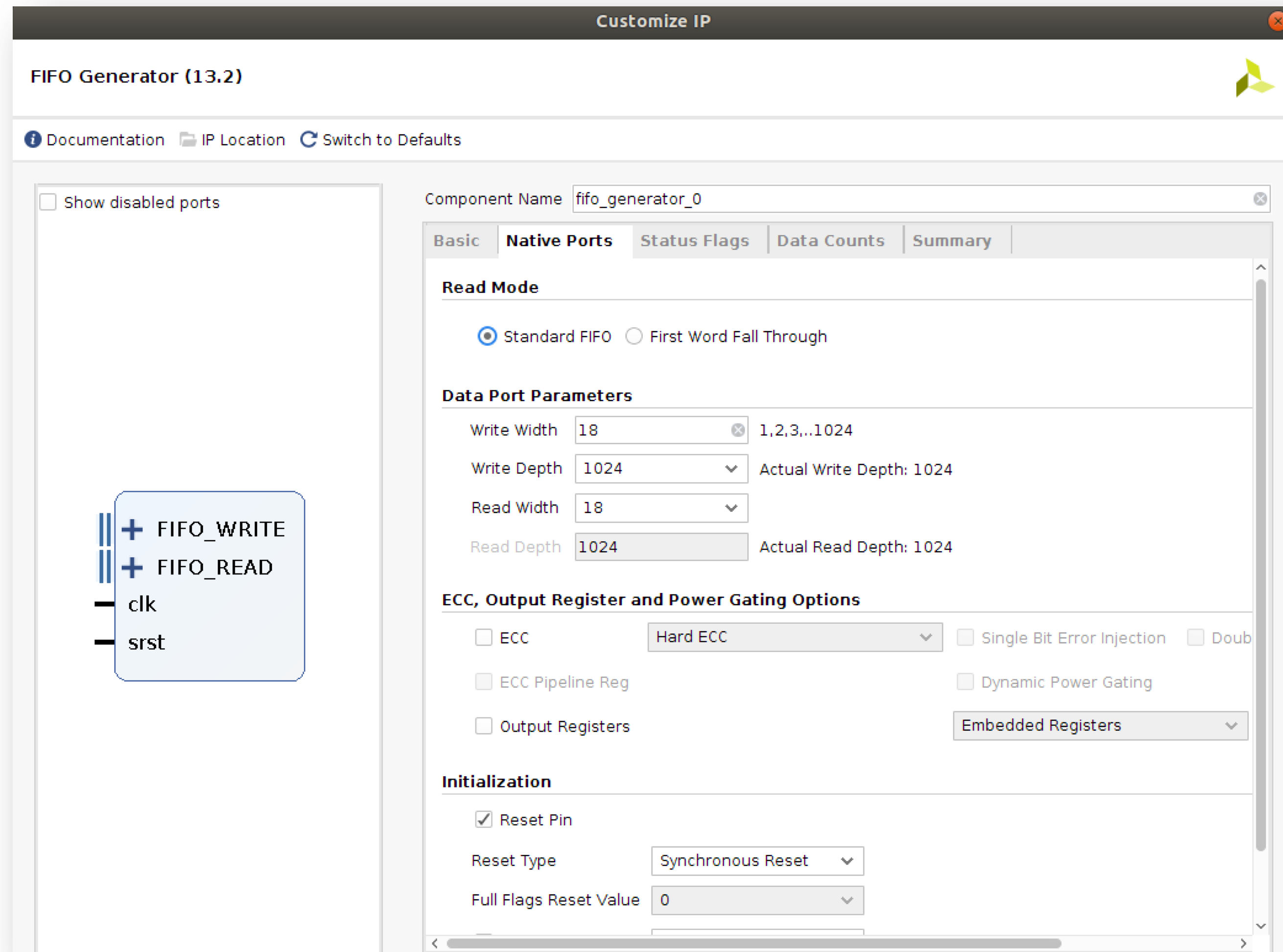
FIFO Implemented with BRAM:

- Remember structure of BRAM:
- Dual Port allows us to simultaneously read and write to different SRAM cells
- Add some logic around it to store and autoincrement the memory addresses and you've got a FIFO

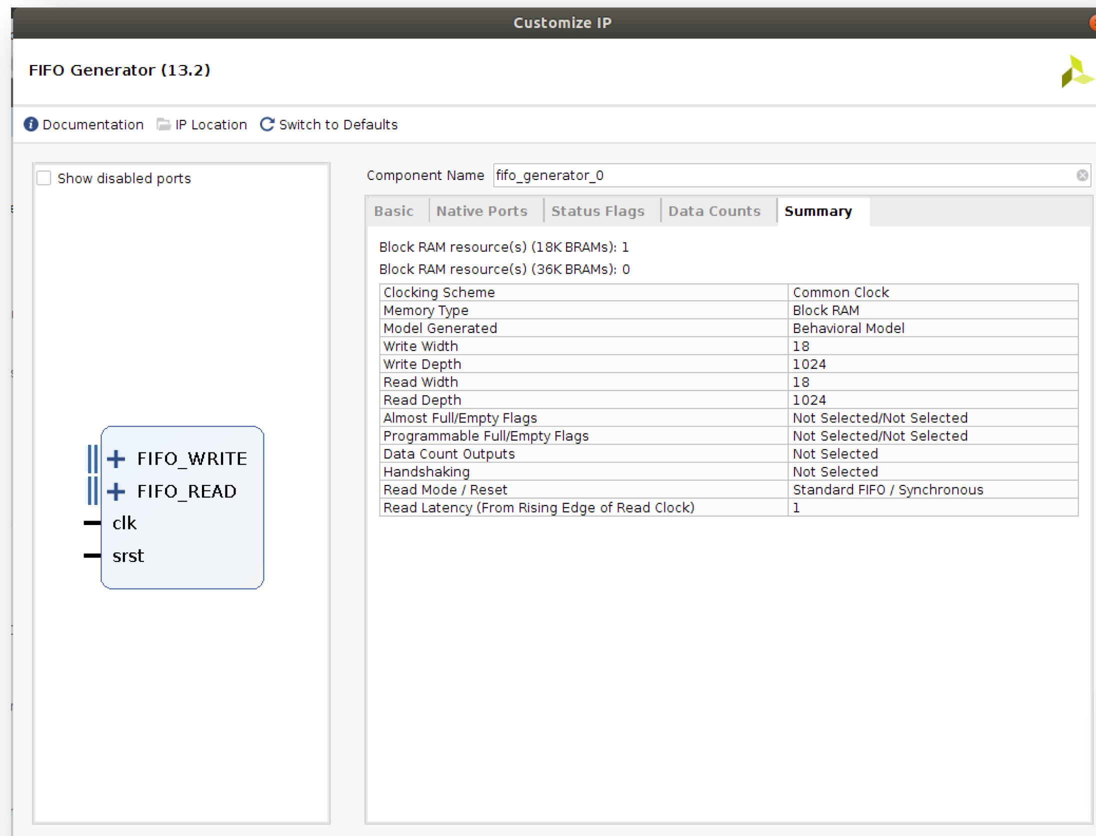


FIFO

- From menus: IP Catalog → FIFO Generator



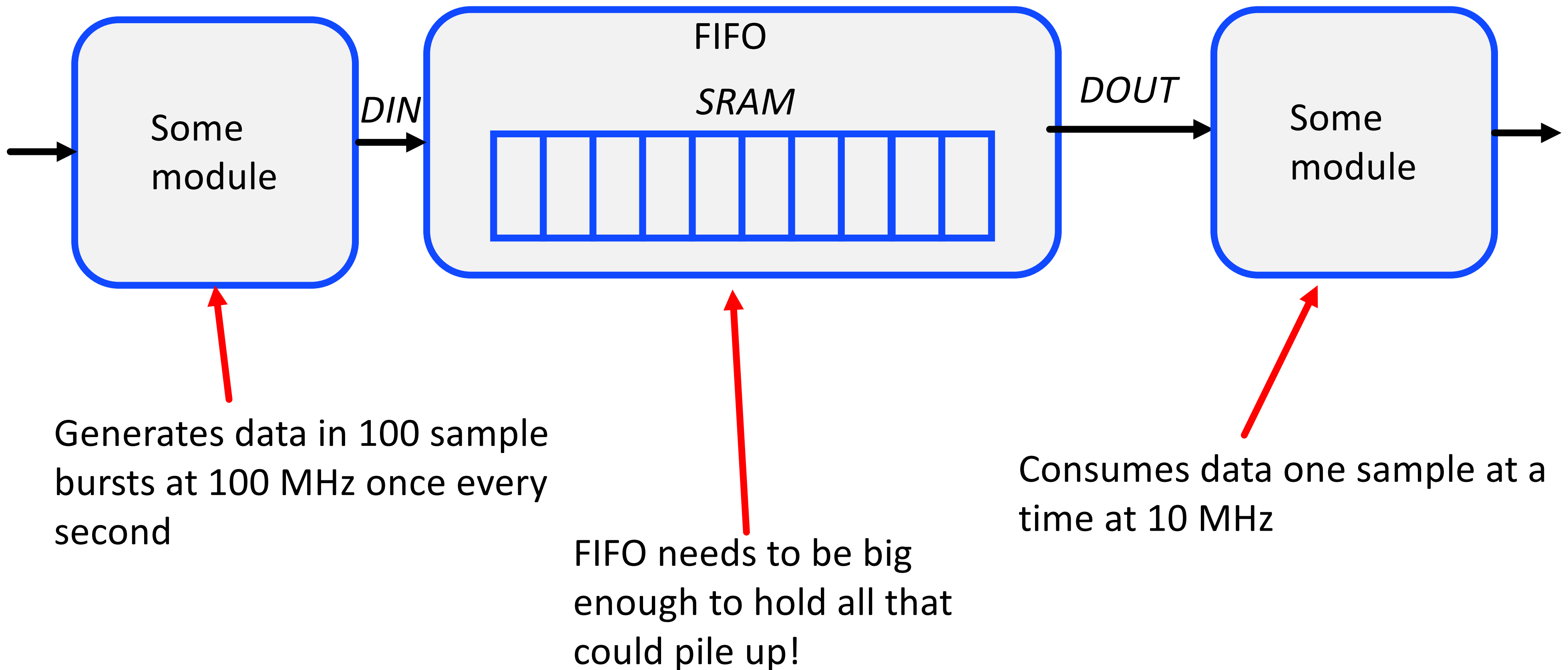
FIFO



- This FIFO can store can store up to 1024 18 bit values in order!

Where to Use FIFOs?

- Anytime you have two modules sharing data (one providing data to another) and they may be producing/consuming in differing patterns



More Reading

- Resource utilization on an FPGA is not easy nor simple (but is really cool). Some resources:
- Series 7 Memory Resources:
  - https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf
- Series 7 Configurable Logic Blocks Resources:
  - https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf
- Series 7 Libraries Docs (How to build almost anything):
  - https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug953-vivado-7series-libraries.pdf
- Good paper on What Gets Inferred from Verilog:
  - https://www.xilinx.com/support/documentation/white_papers/wp231.pdf

SRAM Summary

- Block RAM (and less so Distributed RAM) should be your first choice in storing information
- Quick and reliable:
  - Want measurement? Ask for it and get it:
    - One cycle later (some Distributed RAM configs)
    - One or two cycles later from BRAM
- Limited amounts of it on FPGA (0.5 Mbyte of BRAM, ~amount of Dist RAM)

Memory <u>OFF</u> the FPGA

Two-ish Major Options

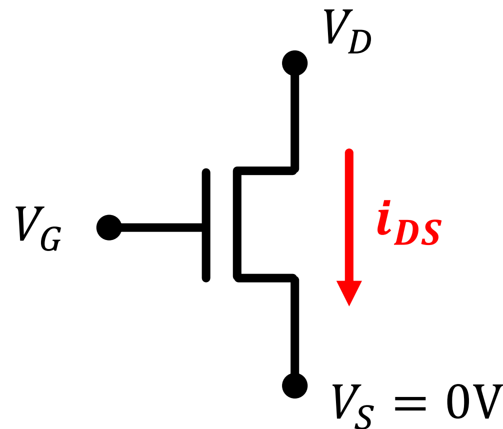
- Flash/EEPROM:
  - Many different form factors, very slow to read/write, but non-volatile, meaning it will last beyond power cycles
- Dynamic Random Access Memory (DRAM):
  - Potentially very high read-write rates
  - Needs to be constantly refreshed (dynamic)
  - Volatile…~100 ms after poweroff, memory lost

EPROM Families

- Includes EPROM, EEPROM, Flash memory, (and SSDs)
- Utilize **Floating Gates**
- Different from SRAM!
- Instead of ~6 transistors per bit, you can do about 1!
- Acts sorta like SRAM from outside but *Non-Volatile* and writes are *much* slower than reads
- Invented by Dov Frohman while at Intel ~1970ish

An early EPROM.
You'd program electrically and then shine UV onto it to erase it…don't use these anymore

Quick Review on MOSFETs



In sub-threshold mode:

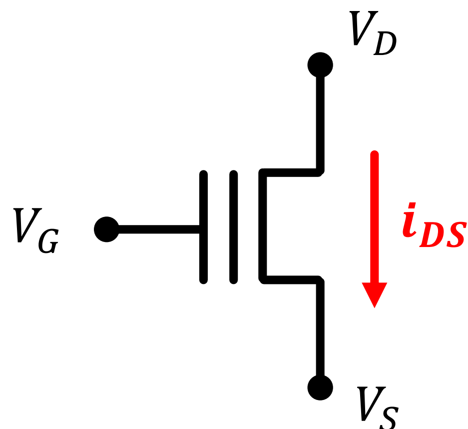
$$i_{DS} = K\left((V_G - V_T)V_D - \frac{V_D^2}{2}\right)$$

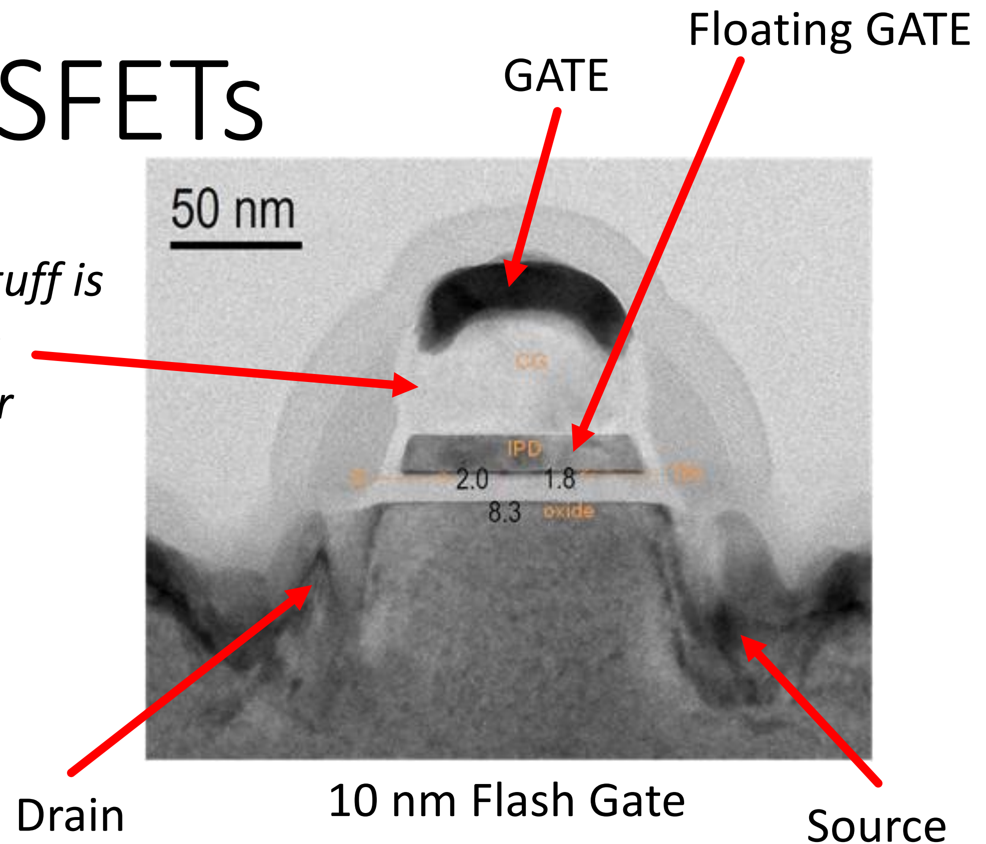
Above Threshold (saturation)

$$i_{DS} = K(V_G - V_T)^2$$

- Basically:
  - If V_G is > V_T you conduct (are ”on”)
  - If V_G is < V_T you do not conduct (are ”off”)
- Traditionally V_T is a function of doping, transistor dimensions, etc…
- BUT!....

Floating Gate MOSFETs





Presence or absence of carriers on floating gate affects the threshold voltage of MOSFET

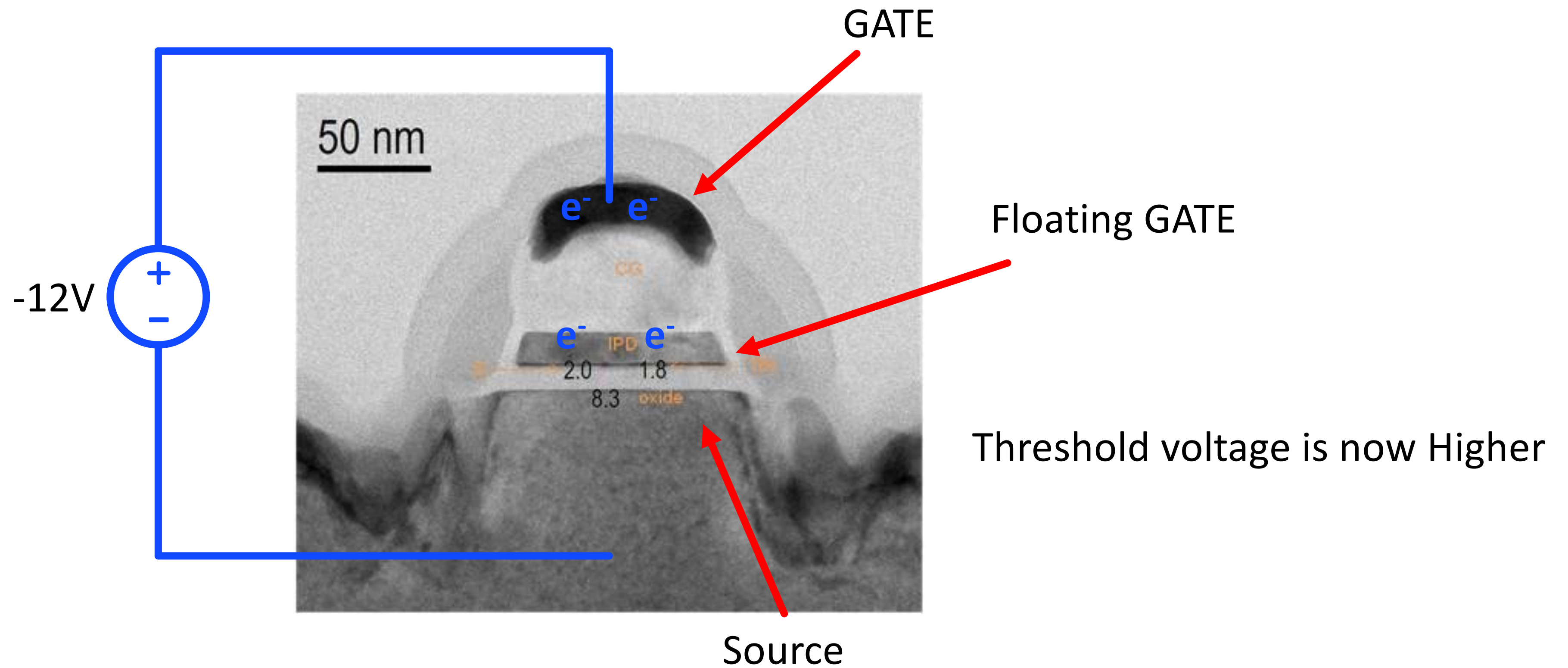
- Default ("binary 1")…Threshold voltage is lower V_{TL}
  - (no electrons trapped in gate)
- Programmed bit ("binary 0")…threshold voltage is higher V_{TH}
  - (electrons trapped in gate)

https://www.electronicsweekly.com/news/research-news/device-rd/iedm-hybrid-floating-gate-scales-flash-to-10nm-2012-12/

Hot Carrier Injection/Tunneling to Program/Reprogram

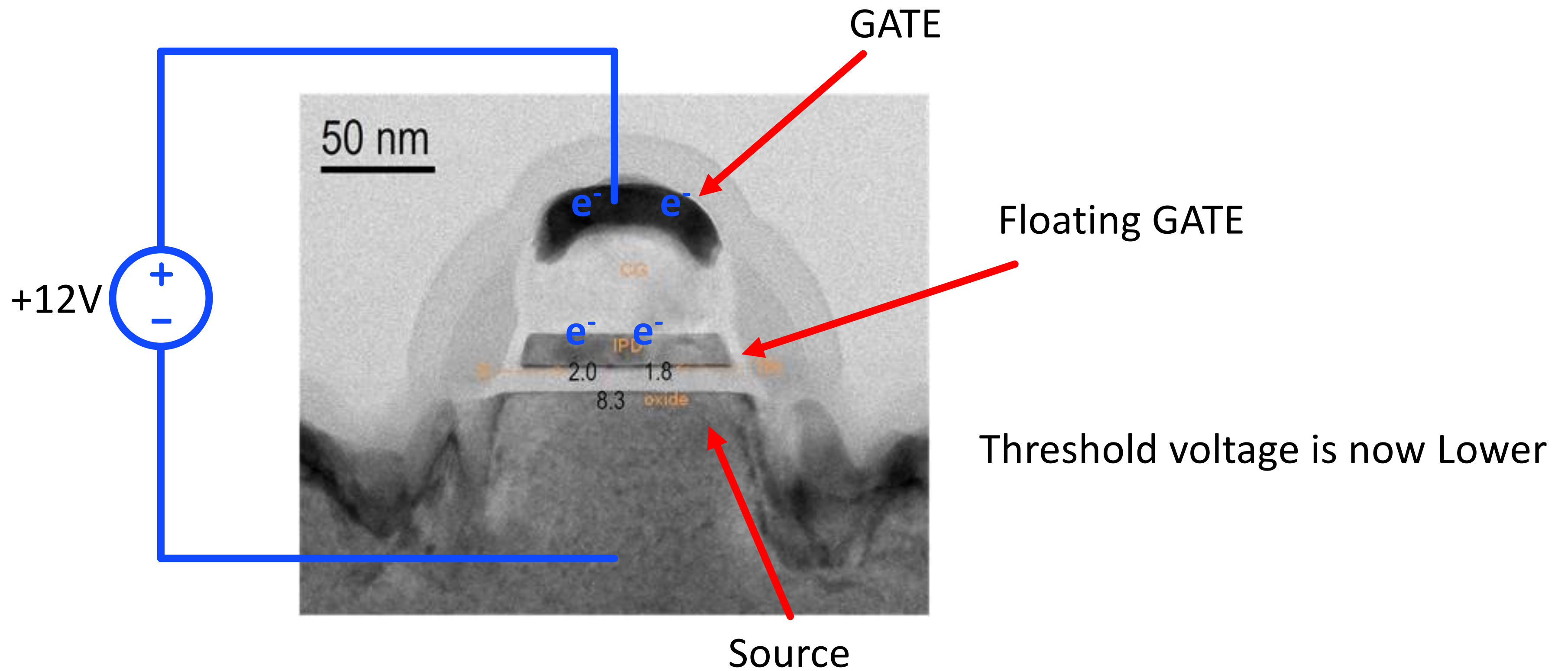
- To add or remove electrons to the floating gate you use a quantum tunneling phenomenon
- High voltage (~12V over 100's of Angstroms) is used to force electrons to tunnel into floating gate… the term "hot" refers to high energies on electrons.
- A similar process is used in reverse to tunnel them out again
- High voltage is a potentially destructive process and will eventually ruin the device. Flash traditionally therefore has limits of ~ several 100,000's of program/erase cycles
- Mitigate issues by wear-leveling (try to spread out usage across all of device…like rotating tires on a car)

To Program a 0



Threshold voltage is now Higher

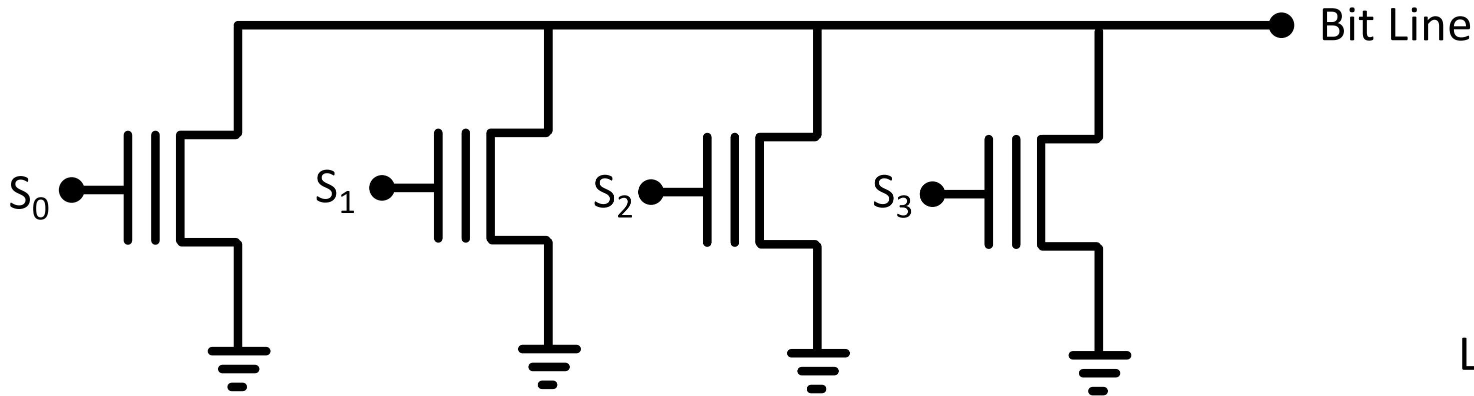
To Program a 1 aka erase



NOR Flash

1 on the select bits means voltage:

- greater than V_{TL} (low threshold from no trapped carriers)
- less than V_{TH} (high threshold from trapped carriers)

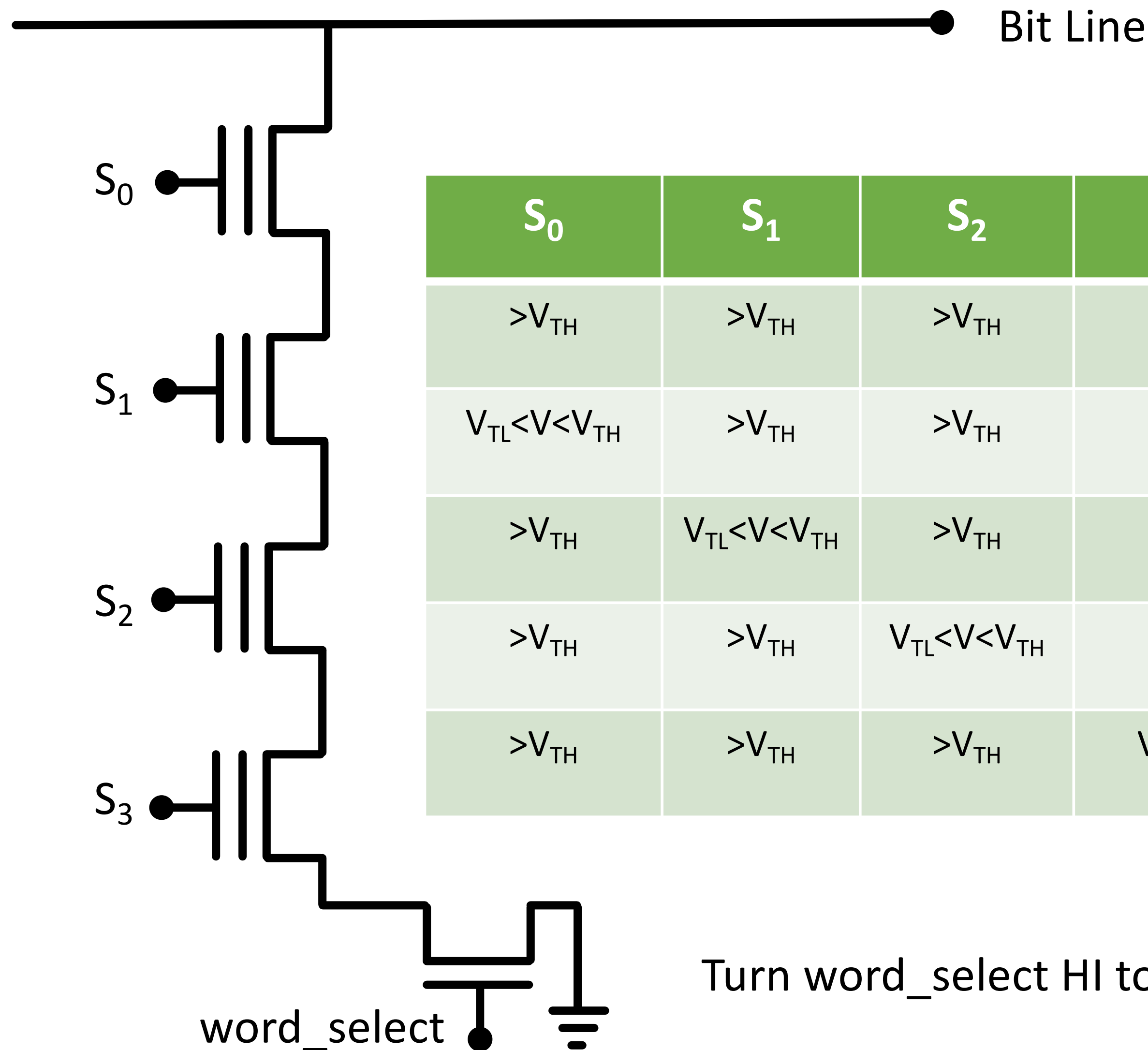


S_0	S_1	S_2	S_3	Out
0	0	0	0	1
1	0	0	0	0 if B_0==1, 1 if B_0==0
0	1	0	0	0 if B_1==1, 1 if B_1==0
0	0	1	0	0 if B_2==1, 1 if B_2==0
0	0	0	1	0 if B_3==1, 1 if B_3==0

Like a **NOR** gate since when all bits are 1, bit line goes low if any input is turned high…hence called **NOR Flash**

NAND Flash

Like a **NAND** gate since when all bits are 1, bit line goes low if tested input is also 1…hence called **NAND Flash**



S_0	S_1	S_2	S_3	Out
$>V_{TH}$	$>V_{TH}$	$>V_{TH}$	$>V_{TH}$	0
$V_{TL}<V<V_{TH}$	$>V_{TH}$	$>V_{TH}$	$>V_{TH}$	0 if B_0==1, 1 if B_0==0
$>V_{TH}$	$V_{TL}<V<V_{TH}$	$>V_{TH}$	$>V_{TH}$	0 if B_1==1, 1 if B_1==0
$>V_{TH}$	$>V_{TH}$	$V_{TL}<V<V_{TH}$	$>V_{TH}$	0 if B_2==1, 1 if B_2==0
$>V_{TH}$	$>V_{TH}$	$>V_{TH}$	$V_{TL}<V<V_{TH}$	0 if B_3==1, 1 if B_3==0

Turn word_select HI to enable whole word

NAND vs. NOR Flash?

- Have Pros cons related to r/w time, size, etc.

Table 1: Major Differences between NOR and NAND

	DiskOnChip (NAND-Based)	NOR	NAND
Capacity	8MB-1024MB	1MB-16MB	8MB-128MB
XIP capabilities (code execution)	XIP boot block	Yes	None
Performance	Fast erase (3msec) Fast write Fast read	VERY SLOW erase (5 sec) Slow write Fast read	Fast erase (3msec) Fast write Fast read
Reliability	Extremely high: Built-in EDC/ECC solves bit-flipping. Bad block management supplied by TrueFFS.	Standard: Bit-flipping issues reported Less than 10% the life span of NAND.	Low: Requires at least one bit for error management (bit-flipping issue). Bad block management required.
Erase Cycles	100,000 – 1,000,000	10,000 – 100,000	100,000 – 1,000,000
Life Span	At least as high as NAND. Usually much better thanks to TrueFFS.	Less than 10% the life span of NAND.	Over 10 times more than NOR
Interface	SRAM-like	Full memory interface	I/O only, Requires toggling both CLE and ALE signals.
Access	Random on code area,	Random	Sequential

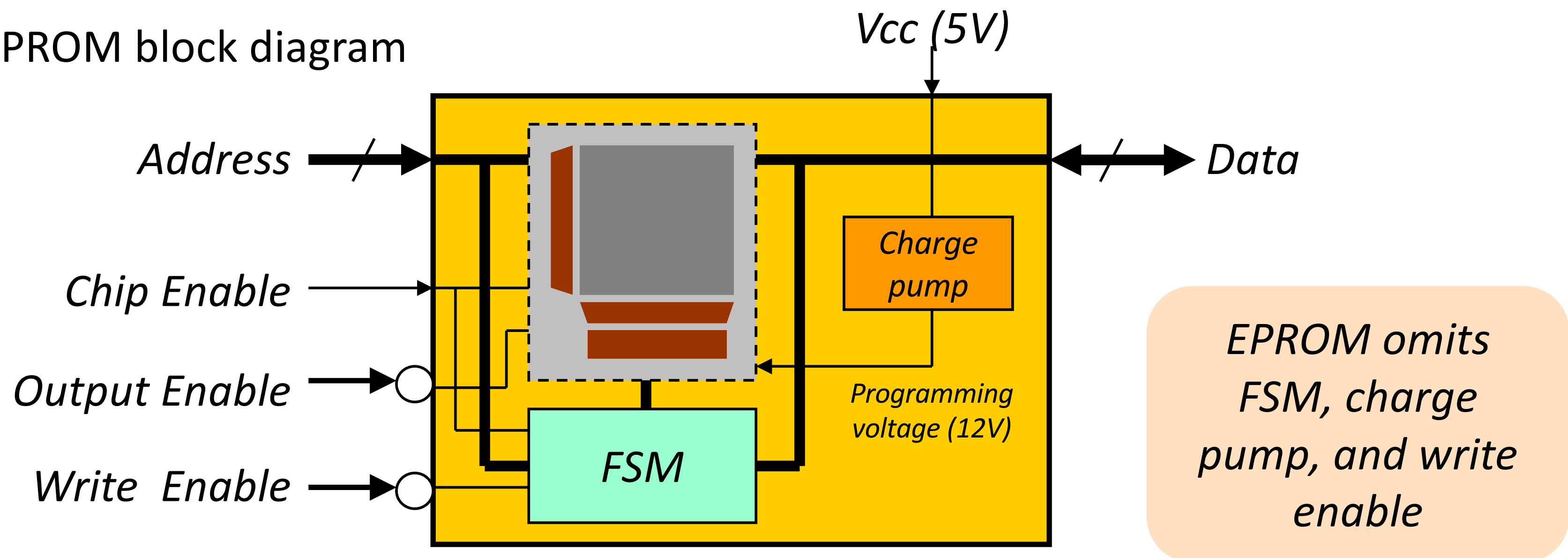
91-SR-012-04-8L 2

https://www.semanticscholar.org/paper/White-Paper-Two-Flash-Technologies-Compared-%3A-NOR-Tal/52f7d974a7be1911b33cb64c26ba4d7f5b337d9e/figure/0

Interacting with Flash and (E)EPROM

- Reading from flash or (E)EPROM is the same as reading from SRAM
- Vpp: input for programming voltage (12V)
  - EPROM: Vpp is supplied by programming machine
  - Modern flash/EEPROM devices generate 12V using an on-chip charge pump
- EPROM lacks a write enable (unless you work for missile defense, you'll not interact with EPROM)
  - Not in-system programmable (must use a special programming machine)
- For **flash** and **EEPROM**, write sequence is controlled by an internal FSM
  - Writes to device are used to send signals to the FSM
  - Although the same signals are used, one can't write to flash/EEPROM in the same manner as SRAM

Flash/EEPROM block diagram

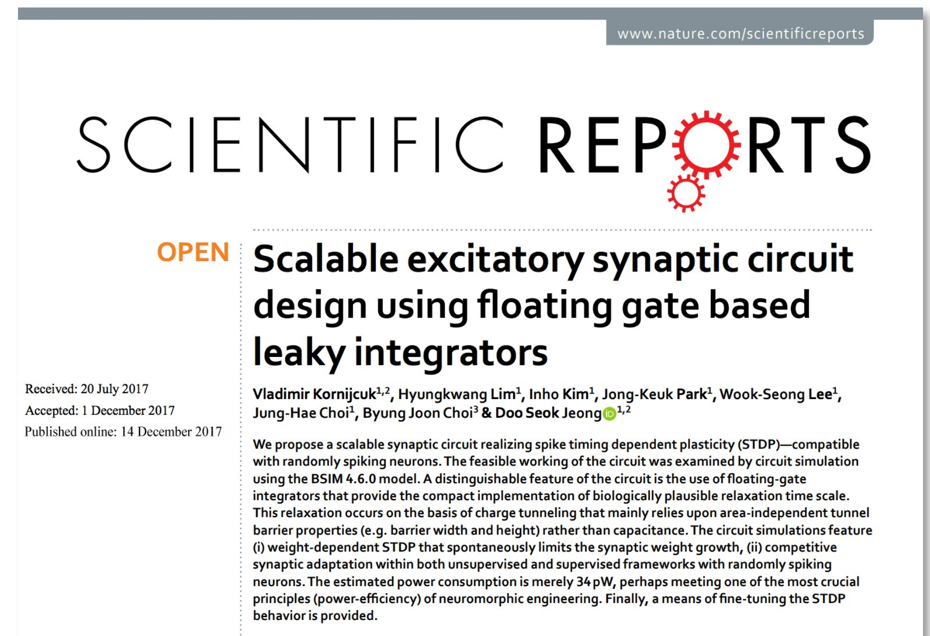


Using Flash with Nexys 4 DDR Board

- You can use the 16MBits of Quad-SPI Flash to permanently "program" the board
- About 75% is unused with full binary so you could use this for permanent storage (I never have, but it is doable)
- Can also interface directly to a 2 GB SD card (which is itself Flash)

Floating Gates

- Some neat recent work using floating gates and their adjustable threshold capabilities
- Result is ability to adjust/teach a single transistor when to fire based on input signals!



www.nature.com/scientificreports

SCIENTIFIC REPORTS

OPEN

Scalable excitatory synaptic circuit design using floating gate based leaky integrators

Received: 20 July 2017
Accepted: 1 December 2017
Published online: 14 December 2017

Vladimir Kornijcuk[1,2], Hyungkwang Lim[1], Inho Kim[1], Jong-Keuk Park[1], Wook-Seong Lee[1], Jung-Hae Choi[1], Byung Joon Choi[3] & Doo Seok Jeong[1,2]

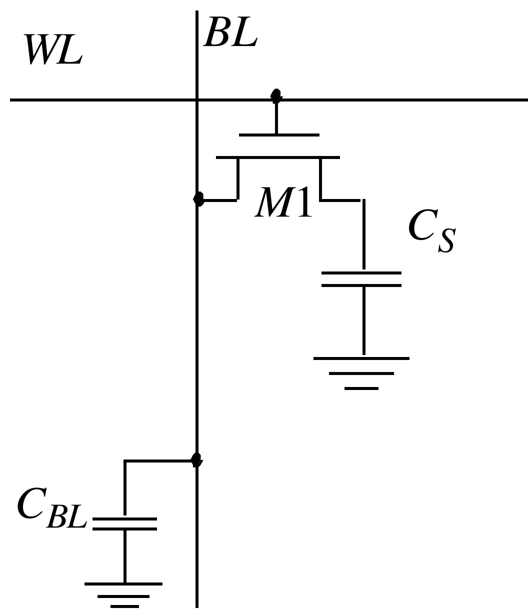
We propose a scalable synaptic circuit realizing spike timing dependent plasticity (STDP)—compatible with randomly spiking neurons. The feasible working of the circuit was examined by circuit simulation using the BSIM 4.6.0 model. A distinguishable feature of the circuit is the use of floating-gate integrators that provide the compact implementation of biologically plausible relaxation time scale. This relaxation occurs on the basis of charge tunneling that mainly relies upon area-independent tunnel barrier properties (e.g. barrier width and height) rather than capacitance. The circuit simulations feature (i) weight-dependent STDP that spontaneously limits the synaptic weight growth, (ii) competitive synaptic adaptation within both unsupervised and supervised frameworks with randomly spiking neurons. The estimated power consumption is merely 34 pW, perhaps meeting one of the most crucial principles (power-efficiency) of neuromorphic engineering. Finally, a means of fine-tuning the STDP behavior is provided.

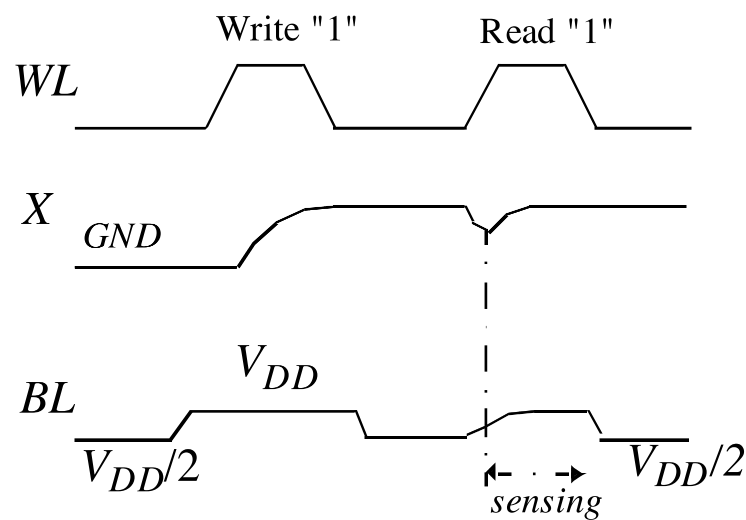
Floating Gate neural-like implementations, 2017

DRAM

- Dynamic Random Access Memory!
- Single transistor and capacitor per bit (capacitor does the storage)
- Capacitors decay rather quickly (especially since DRAM capacitors are about 10 femtoFarads) so need to be refreshed
- Can be made extremely dense and therefore economical
- Are fast-ish:
  - SRAM will have access time of down to 10ns or less (consistent)
  - **DRAM** will have access time from 50-250ns (variable)
  - EEPROM/Flash way slower (esp for writes)

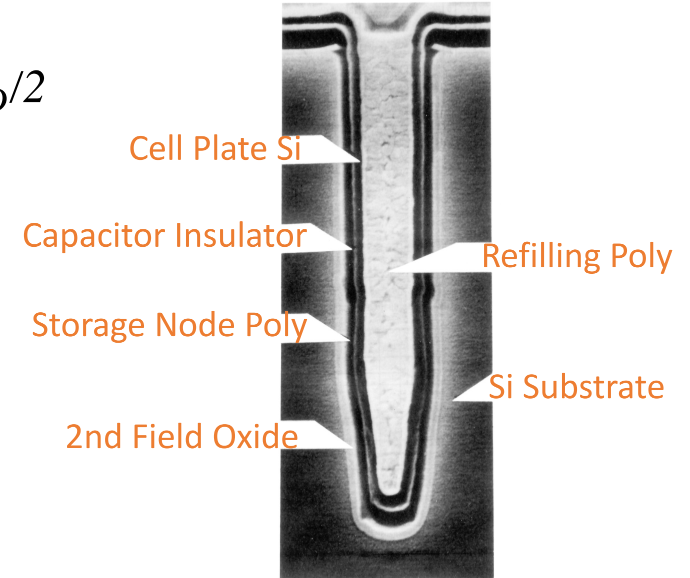
Dynamic RAM (DRAM) Cell





[Rabaey03]

DRAM uses Special Capacitor Structures

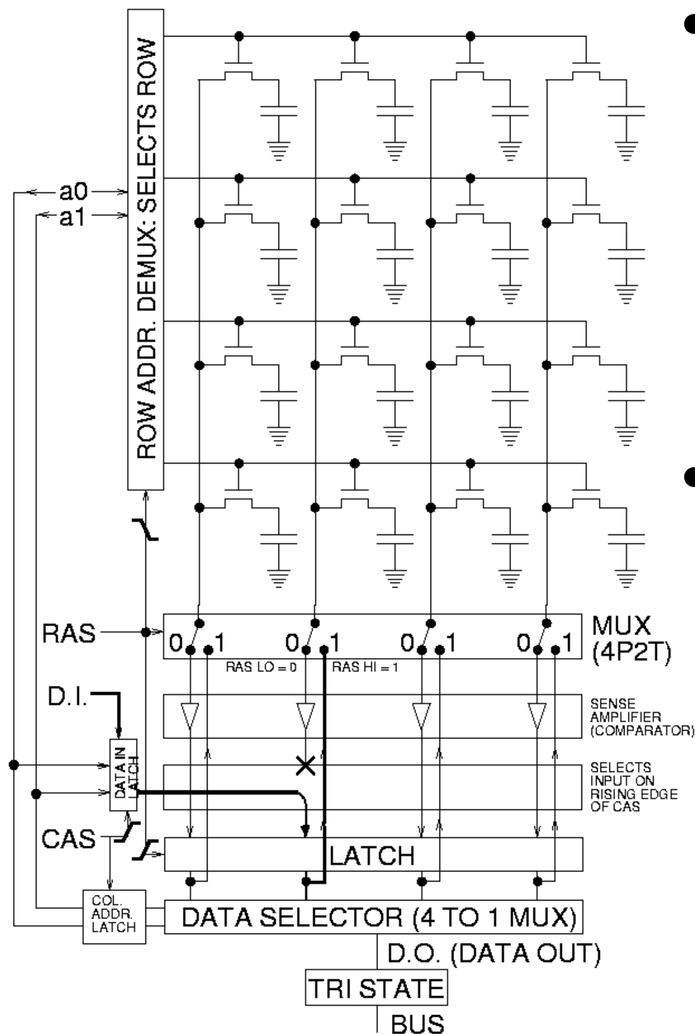


To Write: set Bit Line (BL) to 0 or V_{DD}
& enable Word Line (WL) (i.e., set to V_{DD})

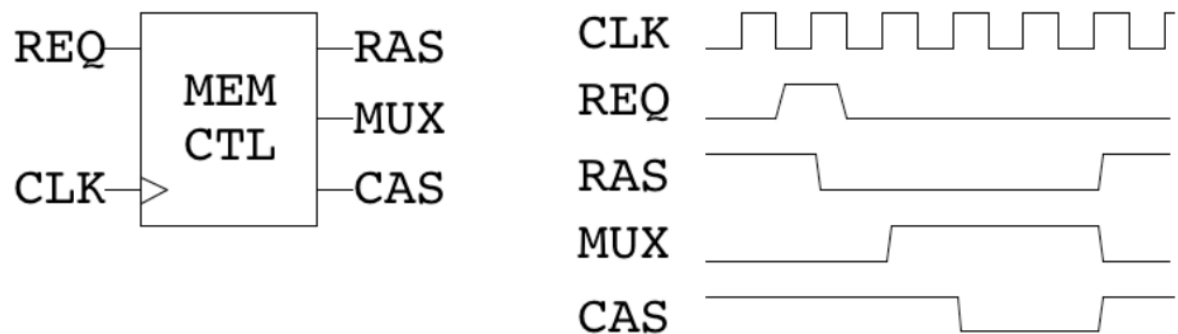
To Read: set Bit Line (BL) to V_{DD} /2
& enable Word Line (i.e., set it to V_{DD})

- DRAM relies on charge stored in a capacitor to hold state
- Found in all high density memories (one bit/transistor)
- Must be “refreshed” or state will be lost – high overhead

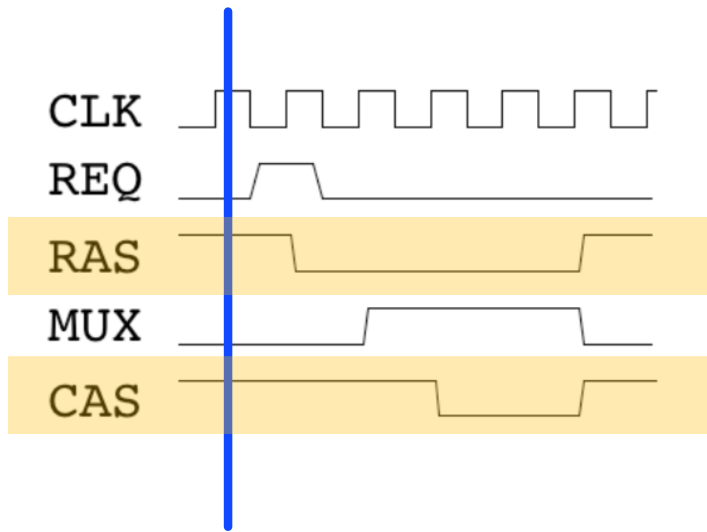
DRAM Memory and Controller



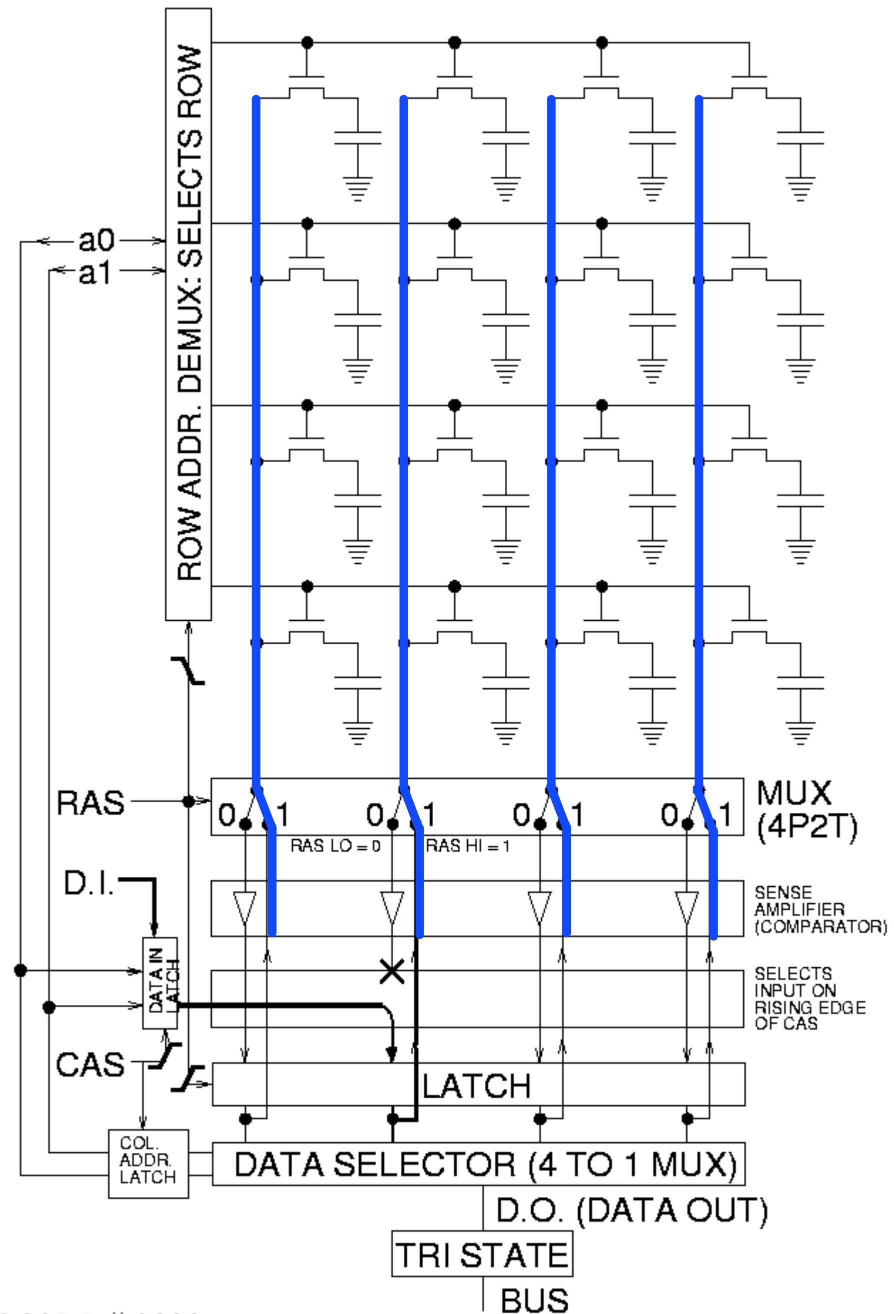
- Reading is destructive!
  - Data stored on small capacitor
  - To read it we must bleed the capacitor off
  - Therefore need to refresh
- Need to refresh even when not reading (every 100 ms)



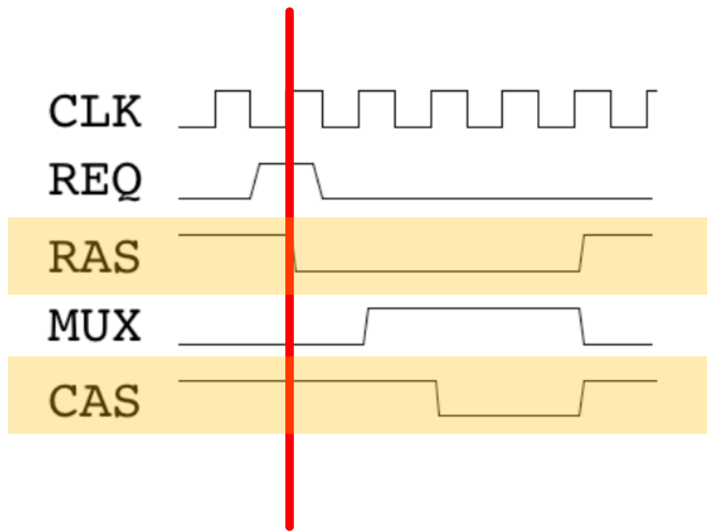
Asynchronous DRAM



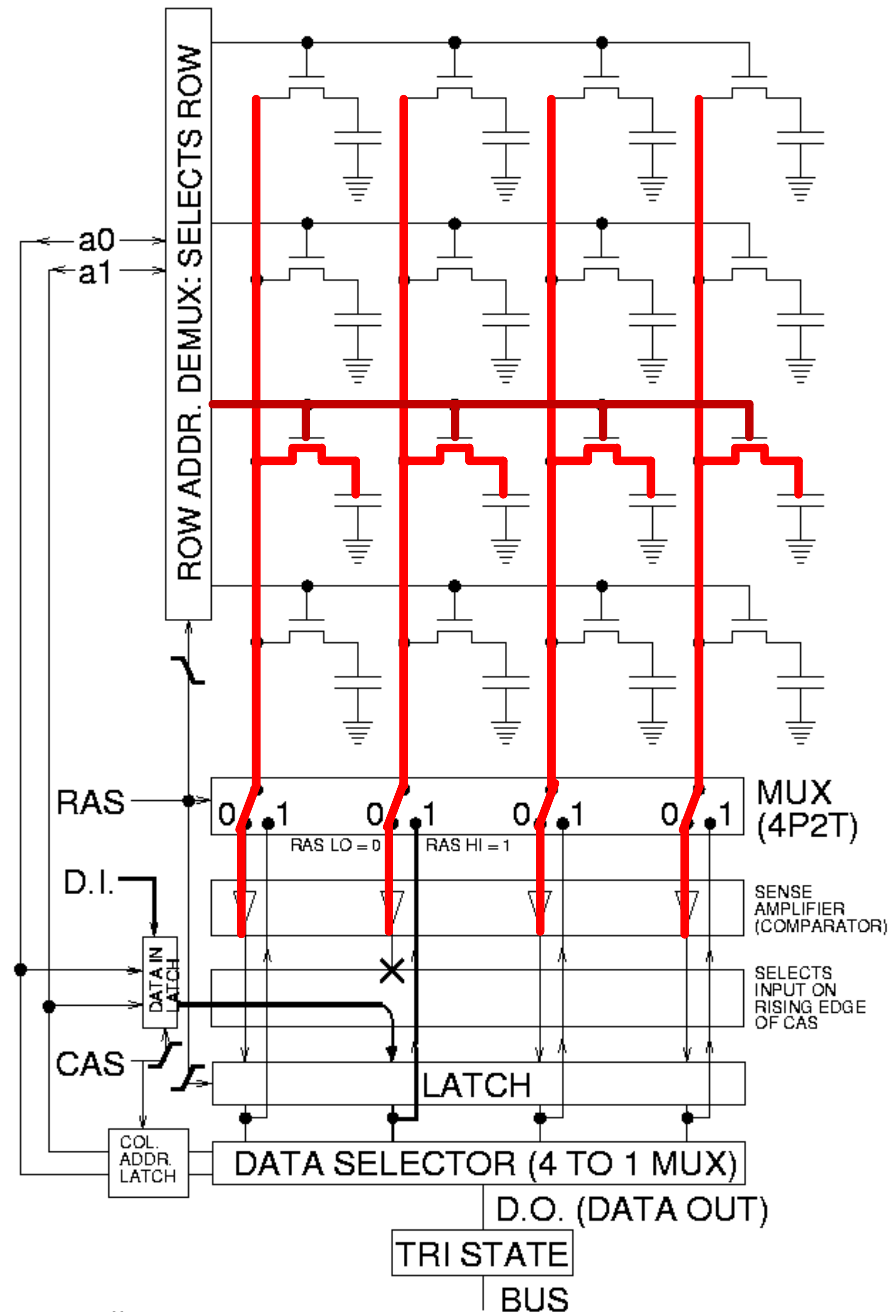
Column lines start charged to mid-voltage (RAS=1)



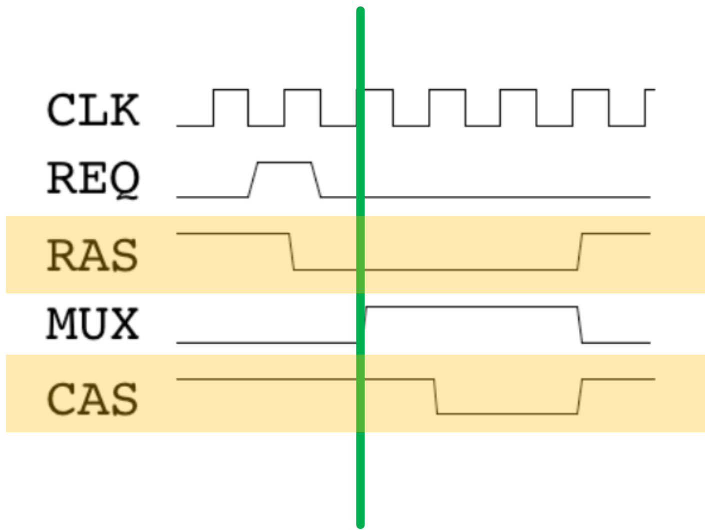
Asynchronous DRAM



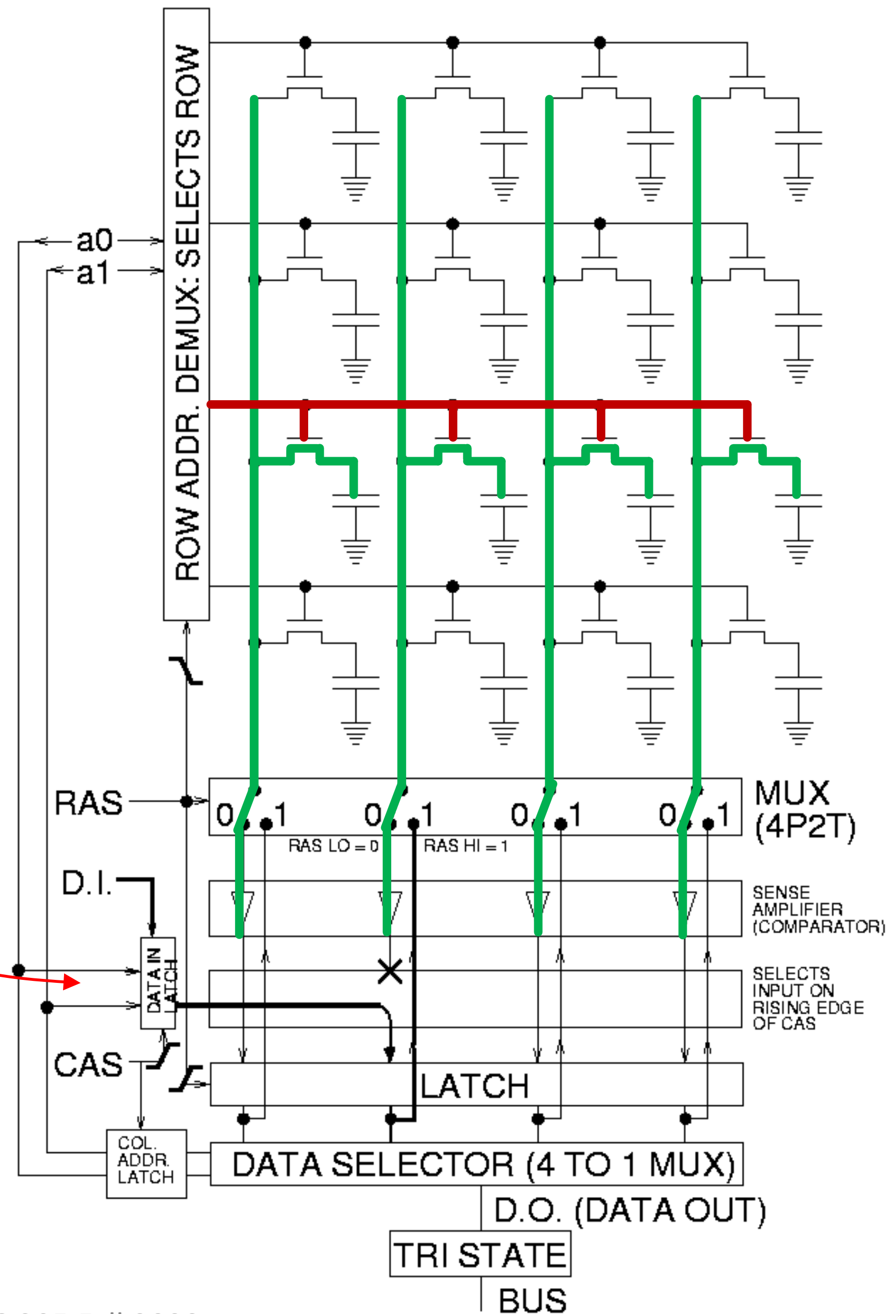
Select Row and feed output of each column into feedback amplifiers to sense/regenerate 1s and 0s



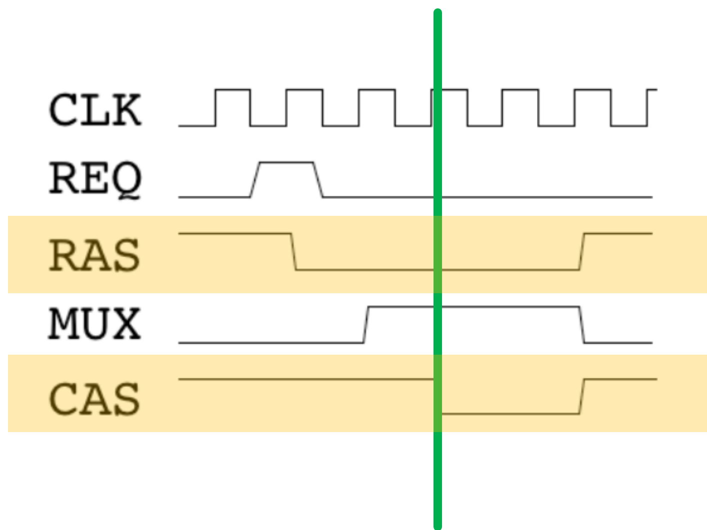
Asynchronous DRAM



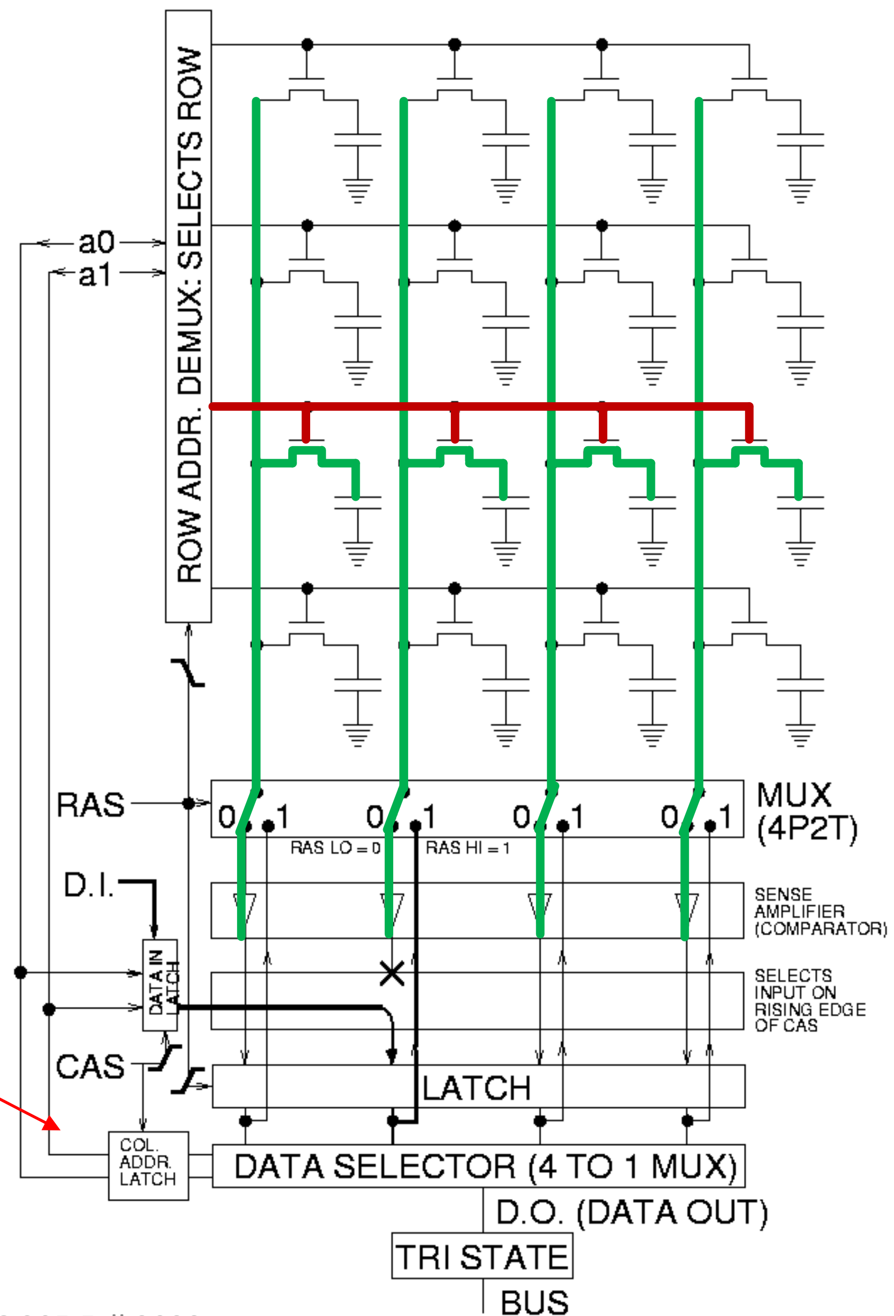
Read/Write by adjusting some input/output selector (associated with MUX signal)



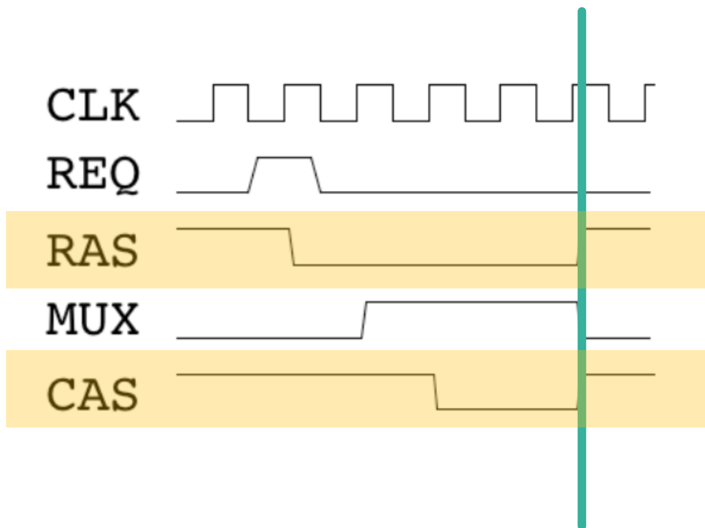
Asynchronous DRAM

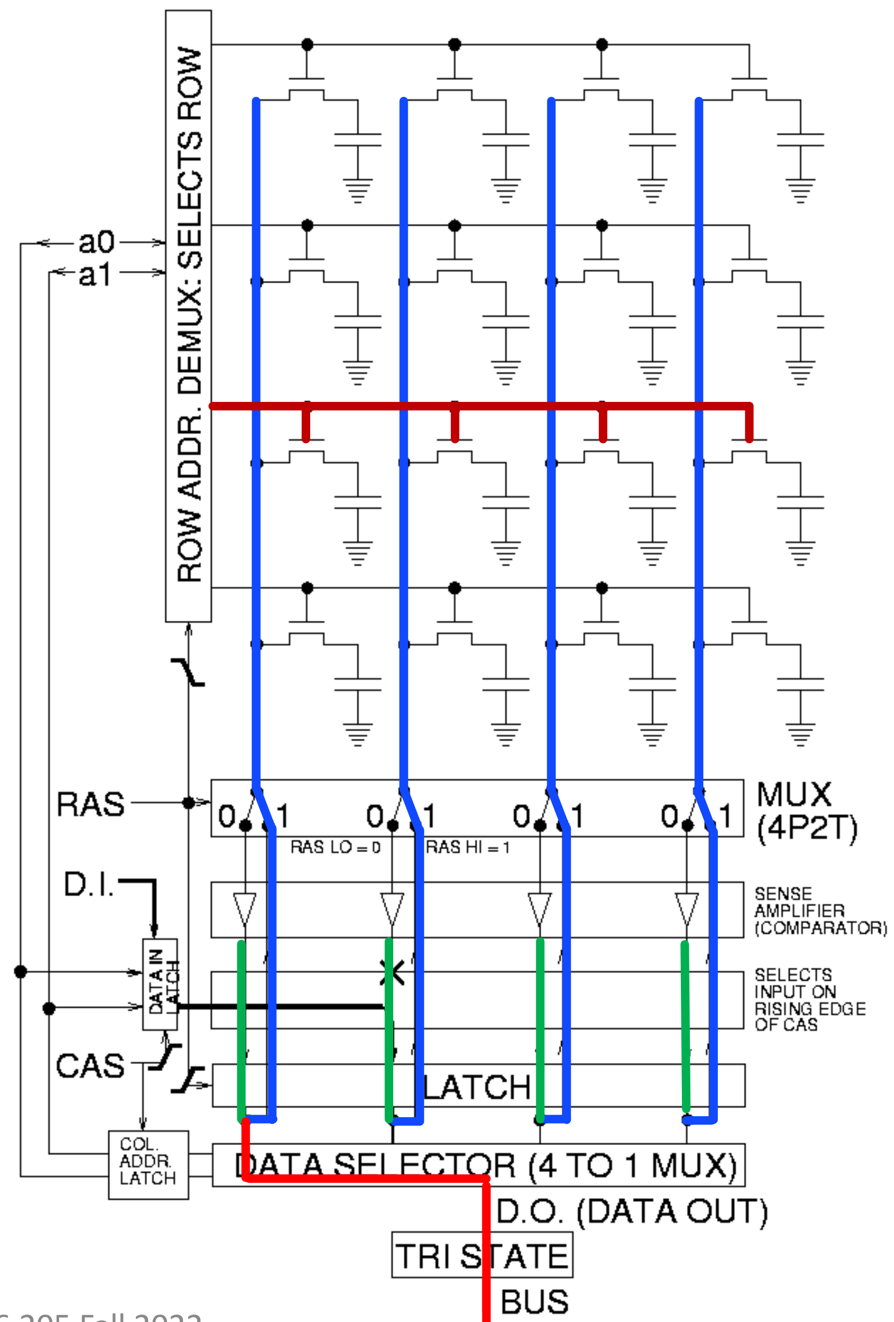


Select which Column to route out



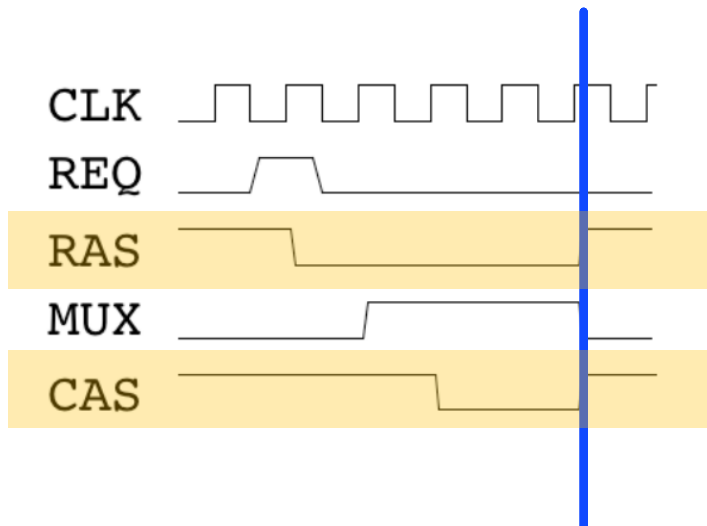
Asynchronous DRAM





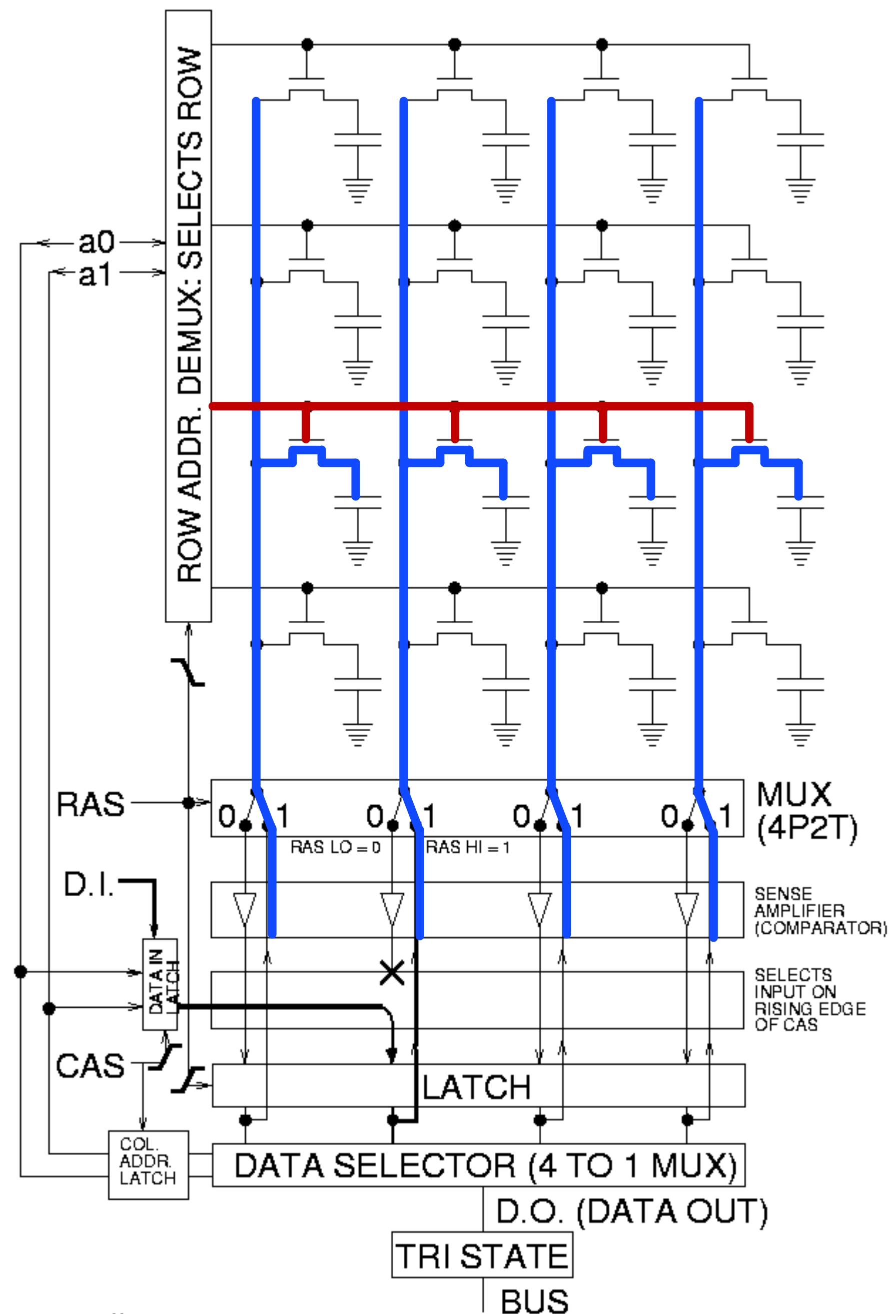
Route out desired bits
ALSO
Redirect read out columns back up to recharge appropriate columns

Asynchronous DRAM



Recharge capacitors fully

Then you're back to beginning



Asynchronous DRAM

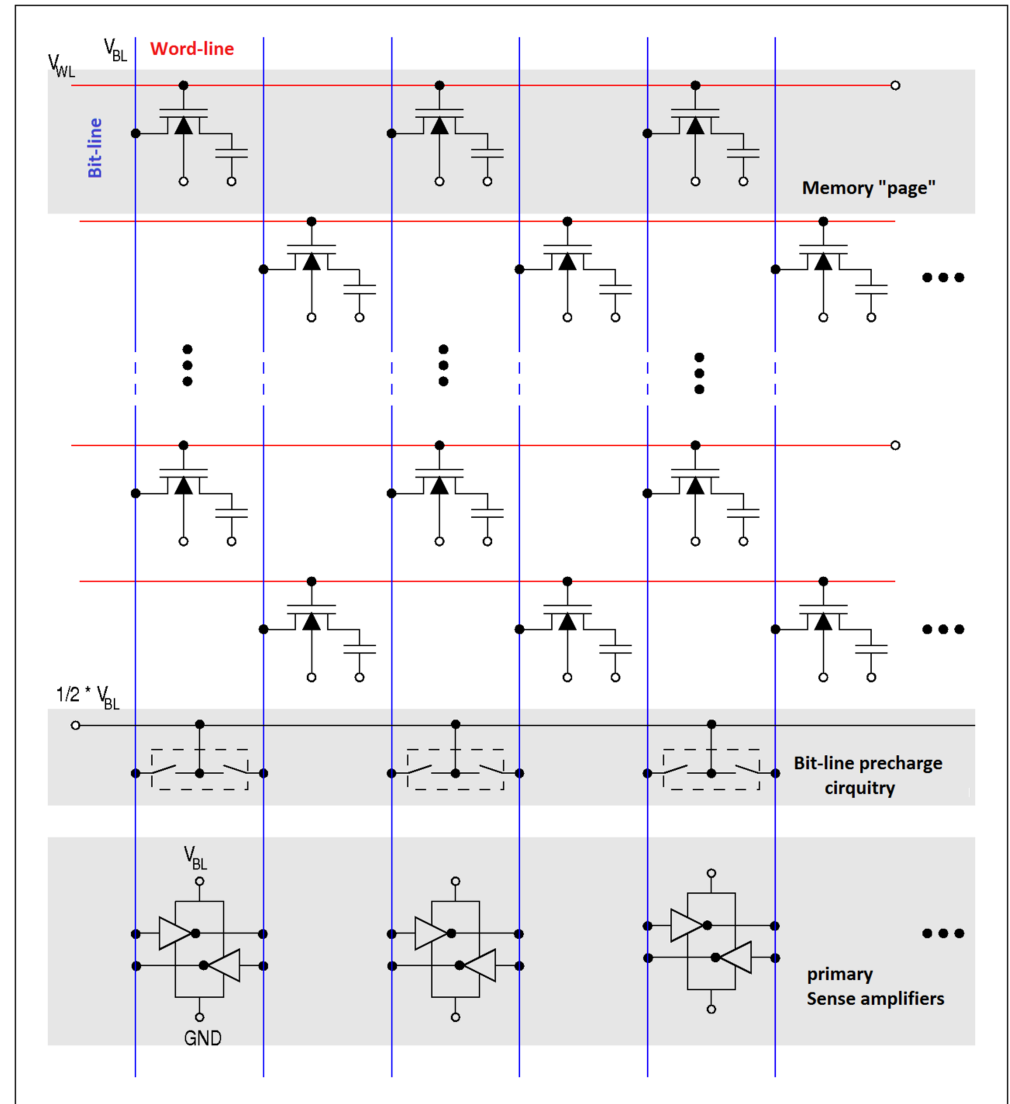
CLK
REQ
RAS
MUX
CAS

Then you're back to beginning

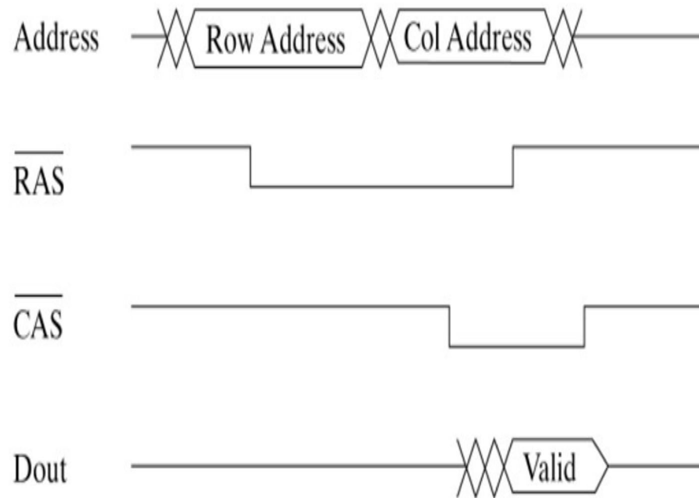
ROW ADDR. DEMUX: SELECTS ROW
a0
a1
RAS
D.I.
DATA IN LATCH
CAS
COL. ADDR. LATCH
MUX (4P2T)
RAS LO = 0
RAS HI = 1
SENSE AMPLIFIER (COMPARATOR)
SELECTS INPUT ON RISING EDGE OF CAS
LATCH
DATA SELECTOR (4 TO 1 MUX)
D.O. (DATA OUT)
TRI STATE
BUS

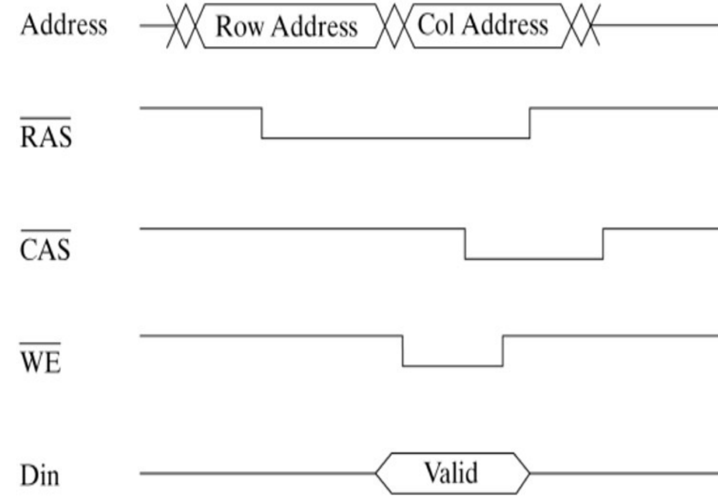
DRAM Cells are Staggered Physically

- The sense amplifiers use two parallel bit lines (one active and one for reference) to detect the slight perturbation when you discharge the capacitor



READ PATTERN **WRITE PATTERN**





Even though we can run DRAM very fast, because of all the maintenance involved in it (we have to clean up after ourselves every time we do something), there's a lot of downtime on the data bus. Compare that to the SRAM from earlier!

https://pubweb.eng.utah.edu/~cs7810/pres/dram-cs7810-protocolx2.pdf

Many Flavors of DRAM

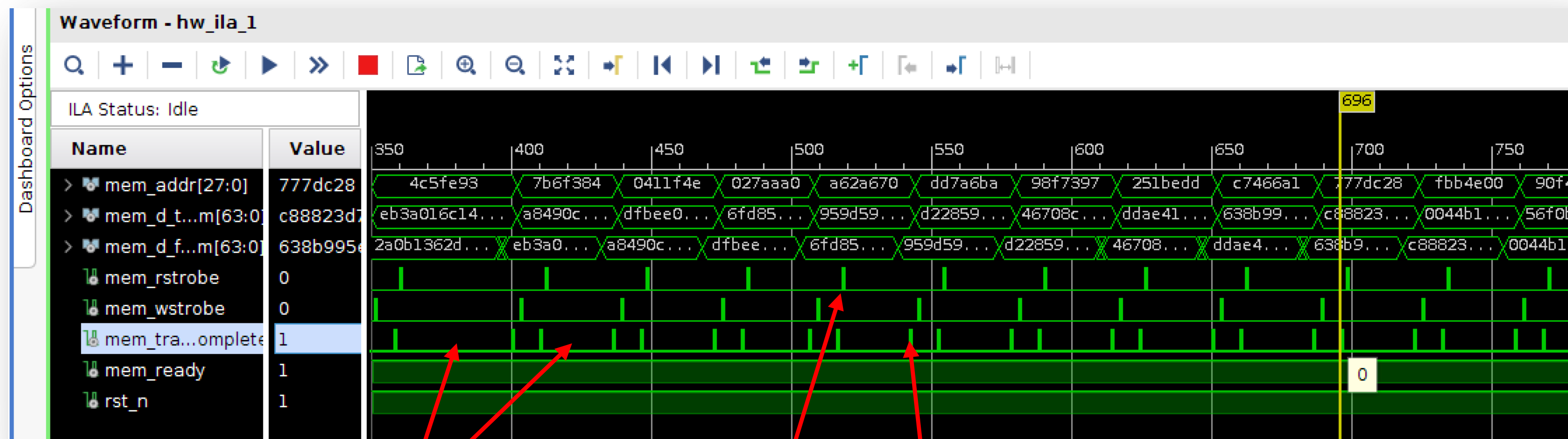
- DRAM (Asynchronous)
- SDRAM (Synchronous DRAM)
  - (one clock cycle per operation)
- Single Data Rate SDRAM (SDR SDRAM):
  - One R/W per clock Cycle
- Double-Data Rate SDRAM (DDR SDRAM)
  - Two R/W per clock cycle (called double pumping)
- Faster Double-Data Rate SDRAM (DDR2 SDRAM)
  - And DDR3 and DDR4

Using DDR2 on Nexys 4 DDR

- Nexys board has ~128MB of DDR2
- There's a "easy" wrapper but the data read/write time is something awful like ~200 ns…however if you tweak it that can be 200 ns for 8 bytes…which can be sufficient for audio throughputs (not video)
- The Memory Interface Generator (MIG), a Xilinx IP, can allow us to interact with the DDR2 more reliably, however, it is much more complicated to work with…
  - I have some working examples for you if you'd like

Using DDR2 on Nexys 4 DDR

- SDRAM is *FAST*, BUT:
  - A lot of delay from a read request to the data out
  - Response time can be variable since DRAM will be taken offline periodically to internally refresh its values…
  - if you're trying to access something that is in a bank getting refreshed, your readout (or write) will be delayed



Readout is variable

Read request

Data appears (~230 ns later)

Using DDR2 on Nexys 4 DDR

- SDRAM is good for:
  - LOTs of data needed in random access at low data rates
    - (~32 MBytes/s average R/W)
  - LOTS and LOTs of data needed in short high speed bursts
    - ~1.2 GBytes/s in bursts

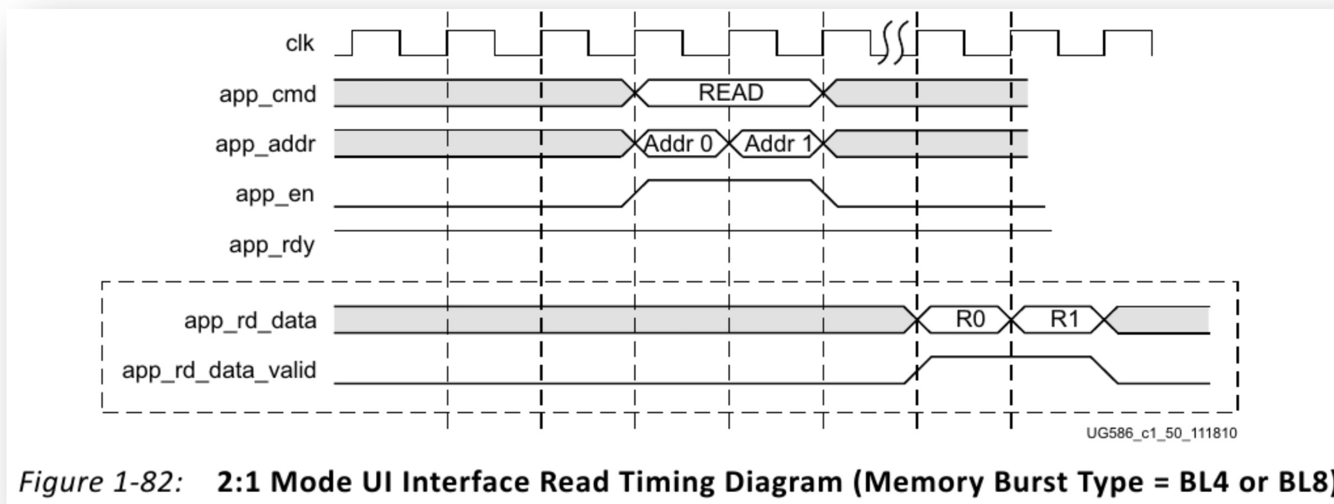


Figure 1-82: **2:1 Mode UI Interface Read Timing Diagram (Memory Burst Type = BL4 or BL8)**

That’s it