EECS151 : Introduction to Digital Design and ICs

Lecture 23 – SRAM

Bora Nikolić

Intel’s Process Roadmap to 2025: with 4nm, 3nm, 20A and 18A?
Ian Cutress, Anandtech, July 2021

- 2020, Intel 10nm SuperFin (10SF): Current generation technology in use with Tiger Lake and Intel’s Xe-LP discrete graphics solutions (DG1, Xe).
- The name stays the same.

- 2021 H2, Intel 7: Previously known as 10nm Enhanced Super Fin or 10ESF. Alder Lake and Sapphire Rapids will use this process. Intel's roadmap states a 20-30% performance per watt gain over 10SF due to transistor optimizations. Intel's 10nm is currently in volume production.

- 2022 H2, Intel 4: Previously known as Intel 7. Intel revealed this technology at the 2021 Intel Investor Day. The Alder Lake family will use this technology and will represent a 10-15% performance per watt gain over the previous generation. The first Intel 4 products are expected to be offered in late 2022. Intel has stated that the technology will use EUV and is expected to be used in the Broadwell product line. Intel has identified this process as Intel 4.

- 2023 H2, Intel 3: Previously known as Intel 7+. Increased use of EUV and new high-density libraries. Intel has a strategy of development in increments, but Intel's roadmap states that 7nm will have some features of Intel 4 and may be enough to achieve a new node. Intel has noted that Intel 3 will be a new full node, but enough will be new enough to be considered a new full node, in particular new high-performance libraries.

- 2024, Intel 20A: Previously known as Intel 5. Moving to double-digit naming, with the A standing for Ångström, or 10A is equal to 1nm. Few details, but Intel is expected to use high-NA EUV and high-aspect-ratio lithography. Intel has stated that Intel 20A will be the first process node to be based on EUV technology.

- 2025, Intel 18A: Not listed on the diagram above, but Intel is expected to use ASML's High-NA machines, known as High-NA EUV or High-NA lithography. Intel has stated that Intel 18A will be the first process node to be based on EUV technology.

Review

- Latches are based on positive feedback
- Clk-Q delay calculated similarly to combinational logic
- Setup, hold defined as D-Clk times that correspond to Clk-Q delay increases
- Flip-flop is typically a latch pair
Random Access Memory Architecture

- Conceptual: Linear array of addresses
  - Each box holds some data
  - Not practical to physically realize
    - millions of 32b/64b words

- Create a 2-D array
  - Decode Row and Column address to get data
Basic Memory Array

- Core
  - Wordlines to access rows
  - Bitlines to access columns
  - Data multiplexed onto columns

- Decoders
  - Addresses are binary
  - Row/column MUXes are 'one-hot' - only one is active at a time

Basic Static Memory Element

- If D is high, D will be driven low
  - Which makes D stay high

- Positive feedback
  - Same principle as in latches
Positive Feedback: Bi-Stability

- As in latches

\[ V_{o2} = V_{i1} \]
\[ V_{o1} = V_{i2} \]

Writing into a Cross-Coupled Pair

- This is a 5T SRAM cell
  - Access transistor must be able to overpower the feedback; therefore must be large
  - Easier to write a 0, harder to write 1
  - Can implement as a transmission gate as well; single-ended 6T cell
  - There is a better solution...
Since it is easier to write a 0 through NMOS, write only 0s, but on opposite sides!
When reading, measure the difference.

- Wordline (WL) enables read/write access for a row
- Data is written/read differentially through shared BL, BL
SRAM Operation

**Write**

![Write Diagram]

**Hold**

![Hold Diagram]

SRAM Operation

**Read**

![Read Diagram]

SRAM read in non-destructive
- Reading the cell should not destroy the stored value
Sizing SRAM Cell

- **Read stability**: Cell should not change value during read
  - $Q = 0$: $M_5$, $M_1$ both on
  - Voltage divider between $M_5$, $M_1$
  - $V_Q$ should stay low, not to flip $M_4$-$M_3$ inverter
  - $(W/L)_1 > (W/L)_5$
  - Typically $(W/L)_1 = 1.5 (W/L)_5$
  - In finFETs: $(W/L)_1 = 2(W/L)_5$
  - Read speed: Both $M_5$ and $M_1$

- **Writeability**: Cell should be writeable by pulling $BL$ low
  - $Q = 1$, $M_5$, $M_2$ both on
  - Voltage divider between $M_5$, $M_2$
  - $V_Q$ should pull below the switching point of $M_4$-$M_3$ inverter
  - $(W/L)_5 > (W/L)_2$
  - Typically $(W/L)_5 = (W/L)_2$ in planar
  - In finFETs: $(W/L)_5 = 2(W/L)_2$
  - 1:2:2 and 1:2:3 sizing
**SRAM Column: Write**

- BL discharged to GND by Write Driver
- High-node discharged through series stacked NFET devices. NFET effective device strength must overcome cell pull-up.
- "Weak" 1 written through source-follower NFET xfer device

**SRAM Column: Read**

- BLs precharged to VDD
- Long BLs
- Short BLs
- Positive feedback causes high-node to droop
- Pull-down / Transfer-device ratio (Beta ratio) determines how high the low node rises
**Administrivia**

- Homework 10 posted on Friday, due 11/22
  - No homework during Thanksgiving
- Project checkpoints #3 this week
**Multi-Ported Memory**

- **Motivation:**
  - Consider CPU core register file:
    - 1 read or write per cycle limits processor performance.
    - Complicates pipelining. Difficult for different instructions to simultaneously read or write regfile.
    - Single-issue pipelined CPUs usually needs 2 read ports and 1 write port (2R/1W).
    - Superscalar processors have more (e.g. 6R/3W)
  - I/O data buffering:
    - Dual-porting allows both sides to simultaneously access memory at full bandwidth.

**Dual-Ported Memory Internals**

- Add decoder, another set of read/write logic, bits lines, word lines:
  - Example cell: SRAM
    - Repeat everything but cross-coupled inverters.
    - This scheme extends up to a couple more ports, then need to add additional transistors.
• Dual-port read/write capability
• Single-cycle read and write, timed appropriately
• Often found in register files, first level (L1) of cache

True or False

1. Transistor leakage doesn’t affect SRAM read speed
2. One should write into an SRAM cell by pulling BL high
3. One can only write into some cells of a selected WL
Deciders

Intuitive architecture for $N \times M$ memory

- Too many select signals:
  - $N$ words = $N$ select signals

$K = \log_2 N$

Decoder reduces the number of select signals $K = \log_2 N$
Row Decoders

Collection of $2^m$ complex logic gates
Organized in regular and dense fashion

(N)AND Decoder

$$WL_0 = \overline{A_0A_1A_2A_3A_4A_5A_6A_7A_8A_9}$$
$$WL_{511} = \overline{A_0A_1A_2A_3A_4A_5A_6A_7A_8A_9}$$

NOR Decoder

$$WL_0 = A_0 + A_1 + A_2 + A_3 + A_4 + A_5 + A_6 + A_7 + A_8 + A_9$$
$$WL_{511} = A_0 + A_1 + A_2 + A_3 + A_4 + A_5 + A_6 + A_7 + A_8 + A_9$$

Decoder Design Example

- Look at decoder for 256x256 memory block (8KBytes)
**Possible Decoder**

- 256 8-input AND gates
  - Each built out of a tree of NAND gates and inverters
  - Need to drive a lot of capacitance (SRAM cells)
  - What’s the best way to do this?

**Possible AND8**

- Build 8-input NAND gate using 2-input gates and inverters
  - Is this the best we can do?
  - Is this better than using fewer NAND4 gates?
Problem Setup

• Goal: Build fastest possible decoder with static CMOS logic

• What we know
  • Basically need 256 AND gates, each one of them drives one word line

Problem Setup (1)

• Each wordline has 256 cells connected to it
  • $C_{WL} = 256 \times C_{cell} + C_{wire}$
  • Ignore wire for now
  • Assume that decoder input capacitance is $C_{address} = 4 \times C_{cell}$
Problem Setup (2)

- Each address bit drives $2^8/2$ AND gates
  - $A_0$ drives $\frac{1}{2}$ of the gates, $A_0_b$ the other $\frac{1}{2}$ of the gates

- Total fanout on each address wire is:

$$ F = \Pi B \frac{C_{\text{load}}}{C_{\text{in}}} = 128 \frac{256C_{\text{cell}}}{4C_{\text{cell}}} = 2^7 \frac{2^8C_{\text{cell}}}{2^2C_{\text{cell}}} = 2^{13} $$

Decoder Fan-Out

- $F$ of $2^{13}$ means that we will want to use more than $\log_4(2^{13}) = 6.5$ stages to implement the AND8

- Need many stages anyways
  - So what is the best way to implement the AND gate?
  - Will see next that it's the one with the most stages and least complicated gates
### 8-Input AND

- Using 2-input NAND gates
- 8-input gate takes 6 stages
- Total $G$ is $(3/2)^3 \approx 3.4$
- So $H$ is $3.4 \times 2^{13} = \text{optimal } N \text{ of } \sim 7.4$
Decoder So Far

- 256 8-input AND gates
  - Each built out of tree of NAND gates and inverters
- Issue:
  - Every address line has to drive 128 gates (and wire) right away
  - Forces us to add buffers just to drive address inputs

Look Inside Each AND8 Gate
Predecoders

- Use a single gate for each of the shared terms
  - E.g., from $A_0, \overline{A}_0, A_1,$ and $\overline{A}_1$, generate four signals: $A_0A_1, \overline{A}_0A_1, A_0\overline{A}_1, \overline{A}_0\overline{A}_1$

- In other words, we are decoding smaller groups of address bits first
  - And using the “predecoded” outputs to do the rest of the decoding
Predecode Options
• Larger predecode usually better:
  • More stages before the long wires
    • Decreases their effect on the circuit
  • Fewer number of long wires switches
    • Lower power
  • Easier to fit 2-input gate into cell pitch

Building Larger Arrays
Building Larger Custom Arrays

- Each subarray is 2-8kB
- Hierarchical decoding
- Peripheral overhead is 30-50%
- Delay is wire dominated
- Scratchpads, caches, TLBs

Cascading Memory-Blocks

How to make larger memory blocks out of smaller ones.

Increasing the width. Example: given 1Kx8, want 1Kx16
Cascading Memory-Blocks

How to make larger memory blocks out of smaller ones.

Increasing the depth. Example: given 1Kx8, want 2Kx8

Adding Ports to Primitive Memory Blocks

Adding a read port to a simple dual port (SDP) memory.

Example: given 1Kx8 SDP, want 1 write & 2 read ports.
Adding Ports to Primitive Memory Blocks

How to add a write port to a simple dual-port memory.

Example: given 1Kx8 SDP, want 1 read & 2 write ports.

Review

• Dense memories are built as arrays of memory elements
  • SRAM is a static memory
  • SRAM has unique combination of density, speed, power
  • SRAM cells sized for stability and writeability
  • SRAM and regfile cells can have multiple R/W ports
  • Memory decoding is done hierarchically
    • Wire-limited in large arrays