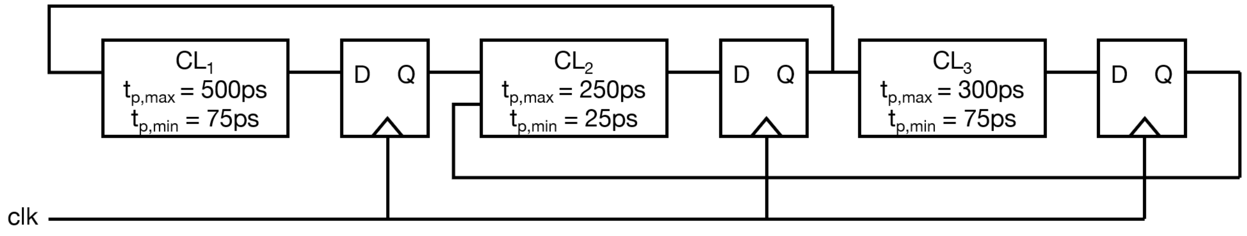
EECS 151/251A Homework 10

Due Friday, Dec 4th, 2020

Problem 1: Timing [16 points]

Consider the following sequential circuit. The minimum and maximum logic delays are annotated on the figure. The flip-flops have the following properties: $t_{clk-q} = 50ps$, $t_{setup} = 50ps$, and $t_{hold} = 25ps$.



a) Let's first assume that the clock has no jitter. What is the minimum clock cycle time for this circuit?

Solution:

The longest delay is on path CL_1:

$$T_{clk} \geq t_{clk_q} + t_{p,max,CL_1} + t_{setup}$$
$$T_{clk} \geq 50ps + 500ps + 50ps$$
$$T_{clk} \geq 600ps$$

Note: strict or non-strict inequalities are acceptable.

b) Under the conditions established so far, does the circuit meet all hold time requirements? Explain.

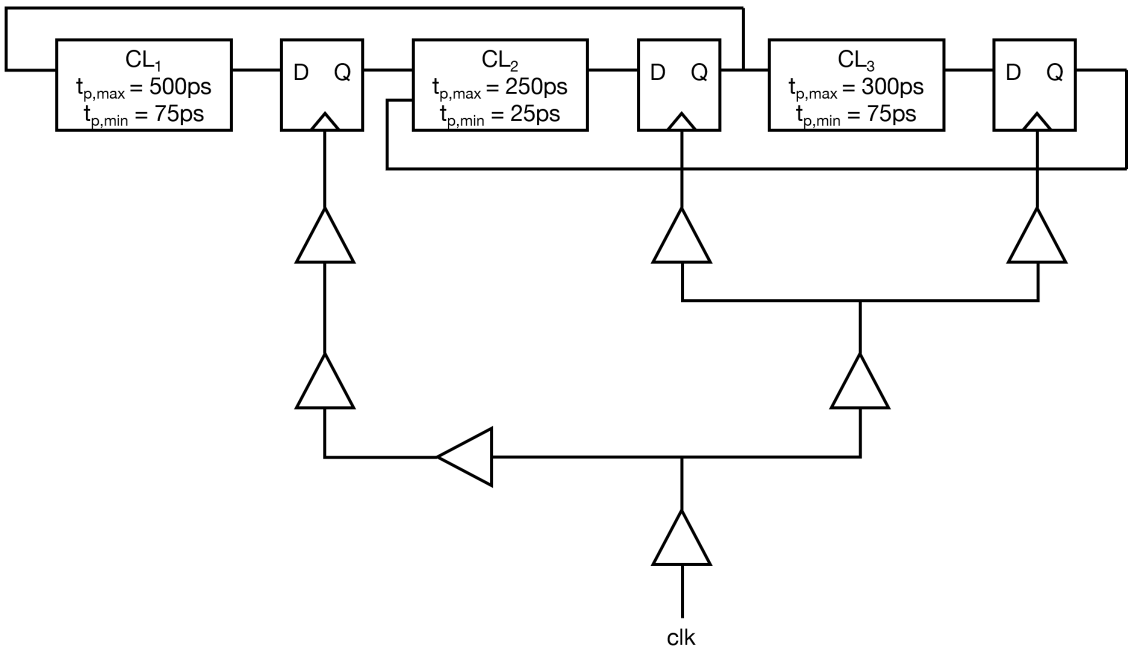
Solution:

Yes. The shortest delay path is on CL_2:

$$t_{hold} \leq t_{clk-q} + t_{p,min,CL_2}$$
$$25ps \leq 50ps + 25ps$$
$$25ps \leq 75ps$$

Note: strict or non-strict inequalities are acceptable.

c) Now let's include a clock distribution network for this circuit, as shown below. Assume the delay of each clock buffer has an delay of 50ps (unaffected by fanout, etc.) and has no variation. Continue to assume no jitter. What is the minimum clock cycle time?



Solution:

The clock arrives at the flip-flop after CL_1 50ps later than the the flip-flop before CL_1 (positive clock skew due to an unbalanced clock tree). Using the same analysis as part a) but subtracting a clock skew term to the RHS, we can see that the clock cycle time can now be **550ps**.

$$\begin{aligned} T_{clk} &\geq t_{clk_q} + t_{p,max,CL_1} + t_{setup} - t_{skew} \\ T_{clk} &\geq 50ps + 500ps + 50ps - 50ps \\ T_{clk} &\geq 550ps \end{aligned}$$

Note: strict or non-strict inequalities are acceptable.

d) Under the same conditions as c), do we have any hold time violation?

Solution:

The clock skew introduced by this network improves the margin for the path in part b) (CL_2). However, there is a new path we should analyze: CL_1. It ends up still having no violation by the same margin as part b).

$$\begin{aligned} t_{hold} + t_{skew} &\leq t_{clk-q} + t_{p,min,CL_1} \\ 25ps + 50ps &\leq 50ps + 75ps \\ 75ps &\leq 125ps \end{aligned}$$

Note: strict or non-strict inequalities are acceptable.

e) Now let's introduce 50ps of maximum cycle-to-cycle clock jitter. Do your answers from c) and d) change? Explain.

Solution:

Yes. The clock jitter is at the root of the clock tree, which means it doesn't cause any additional clock skew for a common clock edge. But, because it is cycle-to-cycle jitter, it but reduces the clock cycle time for setup time analysis.

For minimum cycle time, we need to add the jitter to the RHS (treating it as negative skew), causing the min. cycle time to increase:

$$\begin{aligned} T_{clk} &\geq t_{clk_q} + t_{p,max,CL_1} + t_{setup} + t_{jitter} - t_{skew} \\ T_{clk} &\geq 50ps + 500ps + 50ps + 50ps - 50ps \\ T_{clk} &\geq 600ps \end{aligned}$$

For hold time, there are two cases to consider:

Case 1: the jitter is assumed to be at the root of the clock tree. As a result, all of the registers experience the jitter equally at the same clock edge for hold time analysis, so we still have the same margin as d) of **50ps**.

Case 2: the jitter is assumed to be created by the clock tree buffers. In this case, we must consider it as additional skew, which lowers the margin to **0ps** (violation if non-strict inequality).

f) Repeat parts c) and d), but now with the condition where each clock buffer's delay varies randomly by ±20%. Analyze the timing first with no clock jitter, followed by with clock jitter (same amount as part e)).

Solution:

Without clock jitter:

- We now need to take into account the worst case skew between the flip-flops, given that clock buffer delay can vary by ±10ps.
- Since the flip-flops around CL_3 only vary by 1 clock buffer each (±20ps max), so it doesn't create any new setup/hold violation.
- The flip-flops around CL_1 vary by much more, however, because the common branching point is further back in the clock network. In the worst case, the variation is $\pm 50ps$. Because of this, the min. cycle time is analyzed similarly to part e). However, delay variation affects hold time, and we *just* meet the hold time constraint through CL_1, as shown below (if the inequality is not strict, otherwise it's a violation).

$$\begin{aligned} t_{hold} + t_{skew} &\leq t_{clk\text{-}q} + t_{p,min,CL_1} - t_{variation} \\ 25ps + 50ps &\leq 50ps + 75ps - 50ps \\ 75ps &\leq 75ps \end{aligned}$$

With clock jitter:

- The minimum clock cycle time would increase by a further 50ps to **650ps**.

- If considering Case 1 for jitter, hold time is unaffected by cycle-to-cycle jitter, so we still just meet the constraint (or violate if the inequality is strict). If considering Case 2, then we will violate by 50ps.

g) (251A students only) As you have (hopefully) analyzed, this clock distribution network was designed to increase performance, but only in the absence of both clock buffer delay variation and clock jitter. In practice, both clock distribution networks and combinational paths are adjusted to meet setup and hold timing constraints. Your task is to improve on the circuit in part c) given the following rules:

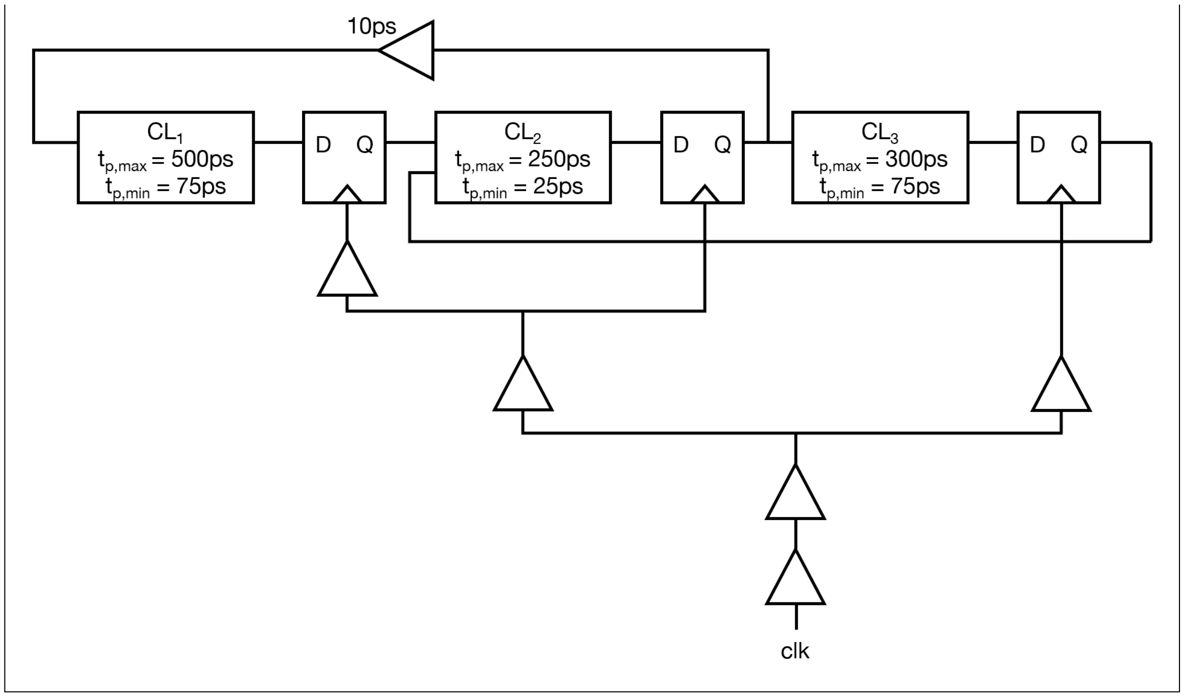
- The combinational logic blocks as given cannot be modified (fixed min./max. delay, logic configuration).
- You may **add combinational logic buffers** of any amount of delay. You can assume these buffers have no delay variation.
- The depth of the clock distribution network must be at least 3 buffers deep.
- The maximum fanout of clock buffers is 2. The load can be clock buffers or flip-flop clock pins, which are assumed equivalent for this problem. Fanout doesn't affect clock buffer delay for this problem.
- Clock buffer delay variation and cycle-to-cycle clock jitter is now present, with the same values as above.

Propose a new circuit configuration that will have at least **50ps hold time margin** (we're *really paranoid!*) while **minimizing the cycle time** in the presence of delay variation and jitter. Include a diagram and explain.

Solution:

There may be multiple solutions (some with larger min. cycle time – only the analysis matters), but here is one:

- We know that delaying the first flip-flop's clock relative to the second one is helpful for minimizing cycle time, so let's keep 4 clock buffers to the first and 3 buffers to the second.
- Since the clock distribution network doesn't affect the contribution of clock jitter, we want to try to minimize the clock skew as much as possible. This means we want to make the last clock branching point as close to the flip-flops as possible. To achieve this, a clock buffer should drive both the second flip-flop and the remaining clock buffer for the first flip-flop, reducing the delay variation to just ±10ps.
- For Case 1 jitter assumption, this results in just 40ps of hold time margin. To get the remaining 10ps, we will need to add a logic buffer to the path to increase its delay of 10ps. For Case 2 jitter assumption, this results in a 60ps logic buffer delay.
- Therefore, the min. clock cycle will be **620ps** for Case 1 jitter assumption, (50ps jitter + 10ps delay variation + 10ps hold buffer on top of 550ps from part c)), or **680ps** for Case 2 jitter assumption.


10ps
CL$_1$
$t_{p,max}$ = 500ps
$t_{p,min}$ = 75ps
D Q
CL$_2$
$t_{p,max}$ = 250ps
$t_{p,min}$ = 25ps
D Q
CL$_3$
$t_{p,max}$ = 300ps
$t_{p,min}$ = 75ps
D Q
clk

Problem 2: SRAM [14 points]

The following 7T SRAM was once proposed and claimed to save power as compared to a standard 6T SRAM. Like the 6T SRAM, there is only 1 pair of bitlines, a read word line (R), and a write word line (WL). There is a new signal, W, that cuts off the feedback connection between the cross-coupled inverters before a write operation. W is logically equivalent to $\overline{WL}$ during a write operation, and high otherwise. R and WL are both high during read, while only WL is high during a write.

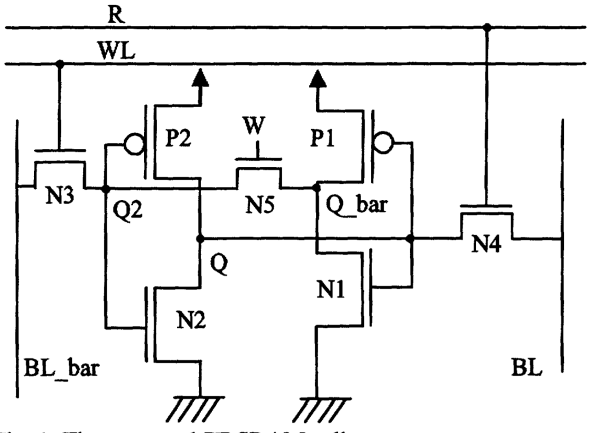


Fig. 1. The proposed 7T SRAM cell.

a) Determine which transistors are involved in a Write operation. How is this different from a 6T SRAM cell? Comment on how sizing can be used to overcome the difference between writing a "0" (BL=0) and a "1" (BL=1). Do we have the same sizing concerns as a 6T SRAM for write?

Solution:

The write operation is done from only a single bitline, and essentially uses an inverter chain before the feedback is re-established:

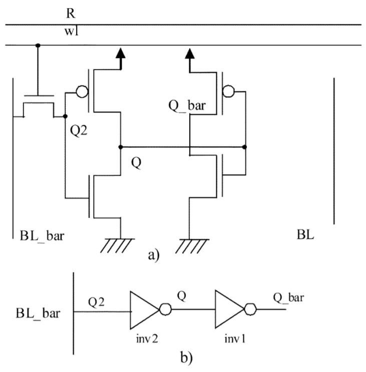


Fig. 3. (a) Proposed 7T SRAM cell during a write operation. (b) Equivalent circuit.

When writing a "1", the N3 NMOS passes a good 0 - no issue. Sizing is unclear here because there's no contention between what's already stored and what's trying to be

written compared to a 6T SRAM.
When writing a "0", N3 passes a bad 1, so it needs to have a skewed inverter (N2 stronger than P2) as well as a strong N3 to reduce delay relative to writing a 1. It is equally valid to say that sizing is also unclear here since you did not learn about skewing gates.

b) Determine which transistors are involved in a Read operation. How is this different from a 6T SRAM cell? Comment on how sizing can be used to overcome the difference between reading a "0" and a "1". Do we have the same sizing concerns as a 6T SRAM for read?
(251A students only) Are there any additional problems with this cell compared to a 6T SRAM cell?

Solution:

The read operation is different depending on whether the cell stored a "0" or "1":

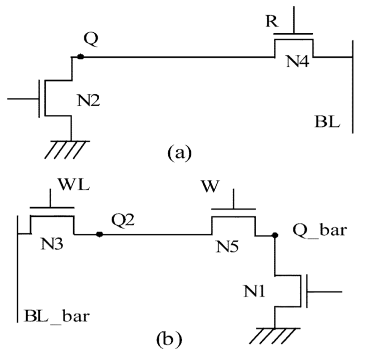


Fig. 4. Read path when (a) Q = "0" and (b) Q = "1".

When reading a "0", it is very similar to a 6T SRAM. N2 should be sized larger than N4. When reading a "1", there is an extra transistor to go through. N1 should be sized larger than N3. N5 also probably needs to be large too.
(251A) The problem is the weaker feedback path for when the cell is storing a "0" as compared to the 6T SRAM. Q2 can't be pulled up to Q_bar well, causing the inverter with N2/P2 to probably leak because P2 isn't as fully off. The solution would be to boost the W signal above the supply voltage or make P2 quite weak (which is required by the write operation anyway).

c) Qualitatively explain how this scheme saves power. What might be the penalty of this? For what applications might this be useful?

Solution:

This has the potential to save power because only one of the bitlines needs to be pulled low after precharge for a write operation. So, writing a "0" (BL=0, BL_bar=1) would not require the bitlines to change at all, thereby saving switching power. The power saving improves the more "0"s are stored in the SRAM array, which could be useful for some sparse data applications or data caches.
Penalties: Because writing is an inverter cascade, it will be slower to write than a 6T SRAM cell. It also is larger than a 6T SRAM cell due to the extra transistor and sizing constraints, in addition to requiring extra wires for R and W without adding additional

port functionality (still 1R1W).

d) Qualitatively explain how reducing the cell supply voltage without changing signal voltage levels external to the cell affects the read stability, read access time, writability, and write delay of the cell.
(251A students only) Does this help solve any additional problems from part b)?

Solution:

- Decreasing the supply voltage decreases the cell's ability to retain data (reduced read stability).
- The read access time is decreased.
- Writability is mostly unchanged for writing a "1" since there's no contention. However, it actually somewhat improves when writing a "0" since the issue of N3 passing a bad 1 is lessened if the bitline's voltage is higher than the cell voltage, requiring less sizing weirdness.
- Write delay of the cell would increase because the inverter cascade is slower.
- (251A) The leakage can be reduced because the W signal is boosted over the cell voltage.

Problem 3: Cache [10 points]

a) Let's review cache associativity. For a direct-mapped cache, a 4-way set associative cache, and a fully-associative cache, rank them (1st, 2nd, or 3rd) on the following metrics. Explain your ordering for each metric.

Solution:

Metric	**Direct-mapped**	**4-way set associative**	**Fully-associative**
Hardware simplicity of tag checking	1st	2nd	3rd
Cache hit rate	3rd	2nd	1st
Cache hit time (for a given size)	1st	2nd	3rd
Cache placement flexibility	3rd	2nd	1st
Cache replacement policy flexibility	3rd	2nd	1st

- For all metrics, 4-way set-associative is always in between because its a tradeoff between direct-mapped and fully-associative.
- For tag checking, direct-mapped is the simplest because only 1 tag needs to be checked, In contrast, fully-associative needs to compare against all the tag bits across all cache lines, so it is the most complex.
- For cache hit rate, direct-mapped is lowest because there is only 1 line per set, increasing the probability of conflict misses. Fully-associative by definition has none, and we can use the placement/replacement policy to maximize cache hits!
- For cache hit time, the more ways we have, the more inputs we need to have in a mux that selects between the ways for the output data.
- For cache replacement policy flexibility, direct-mapped has no flexibility while we can implement many different policies for fully-associative since we can place data in any cache line. As for cache placement flexibility, this only affects cold start, but with more associativity we have more freedom to choose how to order the ways within a set.

b) For the LRU replacement policy, we need to keep track of all the relative ages of each block within a set in order to determine which one is the least recently used. For a 4-way set-associative cache, what is the minimum number of age bits per set needed to implement LRU?

Solution:

We need 6 bits (this is just a combinatorics problem):
bit 0: block[1] more recently used than block[0]

bit 1: block[2] more recently used than block[0]
bit 2: block[2] more recently used than block[1]
bit 3: block[3] more recently used than block[0]
bit 4: block[3] more recently used than block[1]
bit 5: block[3] more recently used than block[2]
(Note: "more" above can be replaced by "less" and ordering doesn't matter)

c) There is an approximation of LRU that uses a binary decision tree for a 4-way set-associative cache. In this algorithm, each node of the tree denotes which half of the lines in the set are older (or newer). Draw a diagram of what this binary tree looks like and then two truth tables: one of how the decision tree bits translate to which line to replace, and the other of what the next decision tree bits would be given a reference to each line (you may need to use X's and notation for unchanged). How would this scale with larger set-associativity, and how does it compare with exact LRU?

Solution:

This replacement policy is commonly called pseudo-LRU.
Let a 1 bit denote that the lower half has been used more recently than the other half (opposite encoding is also valid).

```
                are all 4 lines valid?
                 /       \
               yes        no, use an invalid line
                |
                |
                |
          bit_0 == 0?           state | replace     ref to | next state
           /       \            ------+--------     -------+-----------
          y         n            00x  |  line_0     line_0 |    11_
         /           \           01x  |  line_1     line_1 |    10_
  bit_1 == 0?    bit_2 == 0?     1x0  |  line_2     line_2 |    0_1
    /    \          /    \       1x1  |  line_3     line_3 |    0_0
   y      n        y      n
  /        \      /        \       ('x' means         ('_' means
line_0  line_1  line_2  line_3     don't care)        unchanged)
```

This scales with $log_2(N)$ where N is the degree of associativity. This is smaller than exact LRU (which using the scheme from b) is $\binom{N}{2}$)).

d) Nowadays, processors use multi-level caches to improve the average memory access time (AMAT). The AMAT is defined as: $AMAT = hit\ time + miss\ rate * miss\ penalty$. Given a 3-level cache with the following specs, where a miss in a higher-level (i.e. lower number) cache goes to the next lower-level cache, calculate the overall AMAT.

Level	**Access Time**	**Miss Rate**
$L1	1ns	10%
$L2	5ns	2%
$L3	10ns	1%
Main memory	50ns	N/A

Table 1: Memory hierarchy parameters

Solution:

The key is knowing that hit time is the access time for the current level cache and miss penalty is the access time for the next lower-level cache. $AMAT = 1ns + (0.1 * (5ns + 0.02 * (10ns + 0.01 * 50ns))) = 1.521ns$

Problem 4: DRAM [Ungraded! Just for your own fun :)]

As you have learned, DRAM is significantly more dense than SRAM. Specifically, it has a footprint of 1 transistor/cell and has a trench capacitor under one terminal (look it up!). However, DRAM requires refreshing due to those leaky trench capacitors. A commonly cited characterization for DDR3 retention time is 64 ms @ 85°C. The retention time is highly inversely proportional to temperature and has cell-to-cell variation.

a) The refreshing is normally accomplished row-by-row. The average time between refreshes is specified at 7.8μs to minimally impact access latency (remember, the array is inaccessible during a refresh!). To the nearest power of 2, what is the maximum number of rows we can have in our SRAM array to ensure we never lose data at 85°C?

Solution:

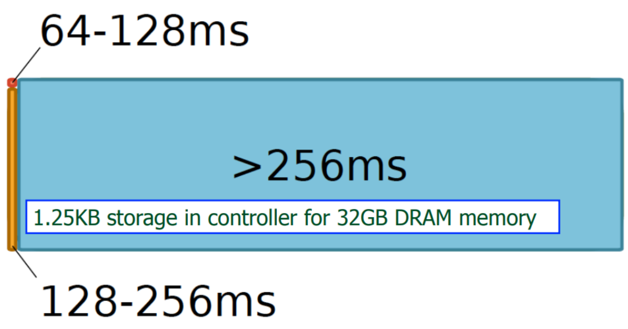
64ms/7.8μs = 8205. The closest lower power of 2 is 8192 rows.

b) If we're limited by how many rows we can have in the array, we can make larger DRAM capacity by growing the number of columns (essentially the word size). However, the DRAM interface is only 64bits for DDR3. How should we partition the DRAM so that we don't unnecessarily destructively read and then refresh cells?

Solution:

We need to break up the array further column-wise. At the extreme, we only read 64bits at a time and bank our DRAM accordingly. In practice, however, bank widths are much greater than this (usually DRAM reads are done in bursts anyway) and so you pipeline the data out to improve throughput.

c) Liu et al. [ISCA 2012] found that 64 ms is highly pessimistic, due to it being at the tail end of a manufacturing process variation distribution. It was found retention times are more like the following:



Propose as many techniques as you wish to reduce on average how often we need to refresh our DRAM, given the retention time profile and the temperature dependence described earlier. Briefly explain how they would work and what overhead they would have. The thought process matters more than accuracy for this problem!

Solution:

Here are a few techniques:

- Characterize which rows have longer retention times (at manufacture or in situ), and schedule those refreshes less often. Requires more complex refresh logic.
- Use redundant rows. Characterize which rows have the shortest retention times at manufacture, disable them, then replace them by the redundant rows. Then, refresh the whole array less often. Requires reconfigurable decode logic.
- Measure the DRAM temperature, and follow a pre-characterized curve of retention time. Requires a temperature sensor and more complex refresh logic.
- Use redundant columns and use error detection and correction. This can allow perhaps 1+ cell per byte/word to leak away but still have a recoverable read. Requires extra ECC en/decoding logic.