EECS 151/251A Homework 8

Instructor: Prof. John Wawrzynek, TAs: Christopher Yarp, Arya Reais-Parsi

Due Monday, Apr 15th, 2019

Problem 1: Power Distribution [10pts]

Suppose you are designing an ASIC that draws 10 Watts peak power at 1 Volt V_{dd}. You plan to use a process that has the following resistance values for the metal layers:

M1-M3	1.2 Ω/square
M4-M5	0.77 Ω/square
M6-M7	0.50 Ω/square
M9	0.36 Ω/square

To feed the power grid you plan to use 500 wires that are each 0.1mm long and would like to keep the voltage variation to under 10%. In microns, what total width (across all wires) would you need to use for each of the metal layer choices?

Solution:

First, we need to determine the current flowing through the feed lines into the power grid. The peak power if 10 W on a system that operates at a V_{DD} of 1 V.

$$P = IV \tag{1}$$

$$I_{peak} = \frac{P_{peak}}{V_{DD}} = \frac{10W}{1V} = 10A \tag{2}$$

The voltage after the power grid feed lines must be within 10% of the nominal V_{DD} therefore, the voltage drop across the feed lines must be $\leq 0.1 \cdot 1V = 0.1V$.

We can now calculate the allowed resistance of the feed lines into the power grid:

$$V = IR \leq 0.1V \tag{3}$$

$$10A \cdot R \leq 0.1V \tag{4}$$

$$R \leq 0.01\Omega \tag{5}$$

If we assume all of the feed wires have the same resistance, the resistance required per wire is:

$$R = \frac{1}{\sum_{i=1}^{500} \frac{1}{R_{wire}}} \tag{6}$$

$$R = \frac{1}{\frac{500}{R_{wire}}} \tag{7}$$

$$R = \frac{R_{wire}}{500} \tag{8}$$

$$R_{wire} = 500R \tag{9}$$

$$R_{wire} \leq 5\Omega \tag{10}$$

The resistance of an individual wire can be computed with:

$$R_{wire} = R_{\square}\frac{l}{w} \leq 5\Omega \tag{11}$$

$$\frac{R_{\square}l}{5} \leq w \tag{12}$$

$$\frac{R_{\square}0.1mm}{5} \leq w \tag{13}$$

$$\frac{R_{\square}0.1mm}{5} \leq w \tag{14}$$

$$w \geq R_{\square}0.02mm \tag{15}$$

For the total width across all feed wires (assuming all are identical):

$$w_{total} = \sum_{i=1}^{500} w = 500 \cdot w \tag{16}$$

$$w_{total} \geq R_{\square}10mm \tag{17}$$

Now, we simply need to substitute $R_{\square}$ for each series of metal layers.

Layers	$w_{total} \geq$
M1-M3	12 mm
M4-M5	7.7 mm
M6-M7	5 mm
M9	3.6 mm

Problem 2: Power Distribution [10pts]

For the ASIC design above, you plan to supply V_{dd} and GND to the chip using solder bump connections, each with 25pH of inductance. 50% of the peak power is dynamic switching power. Assume that the peak current rushes into the chip at the beginning of each clock cycle over 250ps. Again, you would like to keep the voltage variation to within 10%. How many bump connections do you need?

Solution:

We first need to capture some parameters from the previous problem. This design runs with a V_{DD} of 1 V, a peak power of 10 W.

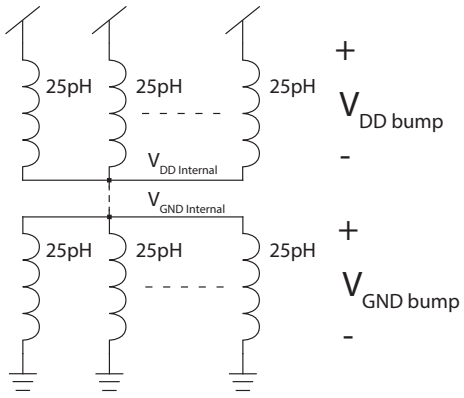
In this problem, we are given that 50% of the peak power is dynamic, resulting in 5 W peak dynamic power.

The peak current can then be calculated as $I = \frac{P}{V} = \frac{5W}{1V} = 5A$.

The maximum voltage variation allowed in this is 10%. The voltage variation is allowed to be $V_{drop} = 0.1 \cdot 1V = 0.1V$.

In this problem, we are asked to consider both V_{DD} and ground pins. In the worst case, the max current flows from V_{DD} into the on chip capacitance and, simultaneously, the same magnitude current flows out charged on-chip capacitors to ground. This current going into/out of the chip also includes any short circuit current.

A diagram of the package solder bumps is shown below. $V_{drop} = V_{DDbump} + V_{GNDbump} \leq 0.1V$.



The voltage drop across an inductor is $L\frac{dI}{dt}$.

Recall that inductors in parallel have an effective inductance of $L = \frac{1}{\frac{1}{L_1}+\frac{1}{L_2}+\cdots}$. When n parallel inductors have the same inductance L_0, the effective inductance is $L = \frac{L_0}{n}$.

Using these properties, we can find the following expression:

$$V_{drop} = V_{DDbump} + V_{GNDbump} \tag{18}$$

$$= L_{DD}\frac{dI}{dt} + L_{GND}\frac{dI}{dt} \tag{19}$$

$$= \frac{L_{bump}}{n_{DD}}\frac{dI}{dt} + \frac{L_{bump}}{n_{GND}}\frac{dI}{dt} \tag{20}$$

$$= \left(\frac{1}{n_{DD}} + \frac{1}{n_{GND}}\right) L_{bump}\frac{dI}{dt} \tag{21}$$

We want to minimize the total number of pins, $n_{tot} = n_{DD} + n_{GND}$ that are devoted to power and ground. This minimization occurs when $n = n_{DD} = n_{GND}$. To prove this is the case, you can solve for n_{GND} in the V_{drop} expression, substitute that into the n_{tot} expression, take the first derivative, and set to zero. Alternatively, you can ask Mathematica or WolframAlpha to minimize the expression for n_{tot} given the V_{drop} constraint, $n_{DD} > 0$, and $n_{GND} > 0$.

With this observation, we can further reduce the V_{drop} expression:

$$V_{drop} = \left(\frac{1}{n_{DD}} + \frac{1}{n_{GND}}\right) L_{bump} \frac{dI}{dt} \tag{22}$$

$$= \frac{2L_{bump}}{n} \frac{dI}{dt} \leq 0.1V \tag{23}$$

$$n \geq \frac{2L_{bump}}{0.1V} \frac{dI}{dt} \tag{24}$$

$$n \geq \frac{2 \cdot 25 \times 10^{-12}}{0.1V} \left(\frac{5A - 0}{250 \times 10^{-12} s}\right) \tag{25}$$

$$n \geq 10 \tag{26}$$

Therefore, at least 10 solder bumps must be devoted to V_{DD} and at least 10 solder bumps devoted to GND.

Problem 3: Clock Uncertainty [12pts]

For each of the 7 sources of clock uncertainty listed in the lecture notes, explain how each can lead to clock skew, clock jitter, or both.

Solution:

1. Clock Generation: [Primarily Jitter] Clock generation can introduce jitter into the clock signal due to noise injected into the PLL. Noise can come from several sources including the power supply and unintentionally resonant circuitry. There isn't much opportunity for skew at this point in since the clock signal has not been split and distributed yet.

2. Device Variation: [Primarily Skew] This primarily contributes to clock skew. Device variations may cause different devices to have different intrinsic delay, thus injecting skew. Contributions to jitter could occur if the device variations unintentionally caused resonant circuits that inject noise into the clock. Device variations may make other sources of jitter more noticeable (ex. through increased coupling).

3. Interconnect: [Primarily Skew] The interconnect primarily contributes to skew (with the exception of coupling which is treated separately). The primary causes are length differences which cause capacitance load differences and speed-of-light propagation differences.

4. Power Supply: [Both] The power supply is one of the principle components which injects jitter into the clock signal. Noise in the power supply can cause random delay differences in devices, thus introducing random jitter in the clock distribution. In addition, power supply difference to different parts of the chip (ex. IR drop) can cause deterministic changes in device delay for different parts of the chip.

5. Temperature: [Both] Many device parameters vary based on temperature. These variations can cause device delays to change. For relatively stable temperatures, these delay variations can introduce skew. For rapidly changing temperatures, this same effect can introduce jitter. Increased temperature also increases thermal noise which may be injected into the clock signal causing jitter.
6. Capacitive Load [Both]: Differences in capacitive load cause it to take more or less time to charge/discharge a given net. With static capacitive loads, this contributes to clock skew. The capacitive load is actually voltage and data dependent which introduces jitter.
7. Coupling to Adjacent Lines [Both, Depends]: Coupling to adjacent lines can cause cross talk between the lines. This can inject noise into the clock signal resulting in jitter. Coupling to lines that generally held a constant value can also introduce a deterministic clock skew.

Problem 4: Clock Load [8pts]

You are designing an ASIC with a total clock load of 10pF based on a clock grid with minimal resistance. With your 7nm Finfet process, a single fin has an R_{eff} of 12KΩ. How many fins would you need in the final stage of the clock driver to achieve a clock rise time (to 50%) of 10ps?

Solution:

In this problem, we need to compute the RC time constant of the clock network.

Recall that the expression for the transient voltage across a capacitor while charging is:

$$V = V_{DD} e^{\frac{-t}{RC}} \tag{27}$$

and the equation for the transient voltage across a capacitor when discharging is:

$$V = V_{DD}\left(1 - e^{\frac{-t}{RC}}\right) \tag{28}$$

For this problem, we are interested in the time to charge/discharge to $0.5V_{DD}$. Solving the above equations for time yeilds the same equation:

$$t = -\ln(0.5)RC \tag{29}$$

In order to achieve a clock rise time of 10ps with a capacitive load of 10 pF, the resistance in the final stage of the clock driver must be:

$$t = -\ln(0.5)RC \leq 10ps \tag{30}$$
$$-R\ln(0.5)10 \times 10^{-12} \leq 10 \times 10^{-12} \tag{31}$$
$$-\ln(0.5)R \leq 1 \tag{32}$$
$$R \leq 1.4427 \tag{33}$$

Now that we know the required resistance, we need to know the number of fins required to attain that resistance. A common technique in FINFET technologies to reduce the effective R_{DS} is for a transistor to have multiple, parallel, fins. This is equivalent to having multiple parallel transistors with their sources, drains, and gates shared. Using multiple fins is a result of the design constraints on individual fins. These multiple fins provide multiple parallel path that effective reduce the effective R_{DS} across the transistor.

Recall that resistance in parallel leads to an effective resistance of $R = \frac{1}{\frac{1}{R_1}+\frac{1}{R_2}+\cdots}$. If n resistors have the same value R_0, the effective resistance is $R = \frac{R_0}{n}$.

Therefore, to have an effective resistance of $R \leq 1.4427$:

$$R \leq 1.4427 \tag{34}$$

$$\frac{R_{fin}}{n} \leq 1.4427 \tag{35}$$

$$\frac{12000}{1.4427} \leq n \tag{36}$$

$$n \geq 8317.8 \tag{37}$$

Therefore, at least 8318 fins are required in the final clock driver.

Problem 5: FIFO Design [20pts]

The FIFO block presented in lecture has the following inputs D_{in}, WE (write enable), RST (reset), CLK, RE (read enable), and the following outputs, FULL, EMPTY, D_{out}. Your job is to design a 8-bit wide FIFO based on a simple dual port memory that is 1K by 8-bits. Assume there are two counters that you can use for the read and write pointers. The counters increment on the rising edge of the clock when their clock enable (CE) signal is true. The counters are 10-bits wide and wrap-around to 0 after they reach their max value. You also can use simple flip-flops as flags to hold the state (FULL/EMPTY) of the FIFO. For simplicity, you may assume that the external producer will never write when full and the consumer will never read when empty.

Design the control logic for the FIFO and draw a block diagram of its implementation using the counters, the memory block, and your control logic.

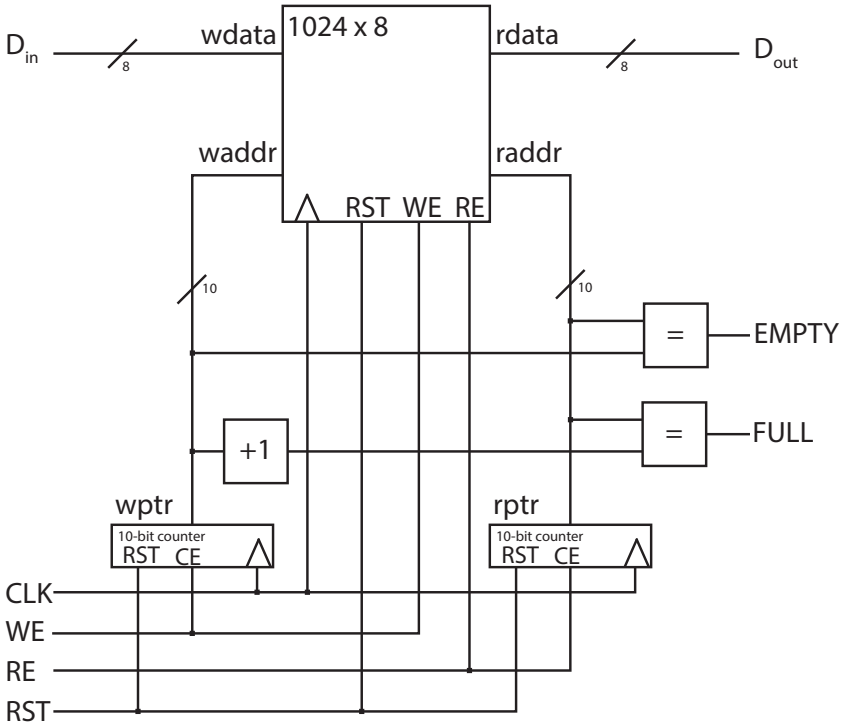
Solution:

We keep two counters, `wptr` and `rptr`. Initially, and on every reset `RST`, these are both 0. A 1K memory needs a $\log_2(1024) = 10$ bit address. Since the counters are both the 10-bit counters described in the problem, we can immediately use them as the read and write addresses to our dual-port memory (`raddr` and `waddr` respectively). When the counters are equal, as they are just after a reset, the FIFO is empty. Whenever adding 1 to the `wptr` would make it equal to the `rptr`, the FIFO is full.

Whenever a read is issued (`RE` high), the memory block is enabled for read and `rptr` is allowed to be incremented. Likewise, whenever a write is issued (`WE` is high), `wptr` is incremented and the memory has write enabled.

Since we can assume that external circuitry does not attempt to read from the FIFO when it is empty or write to the FIFO when it is full, this is the only control circuitry we need.

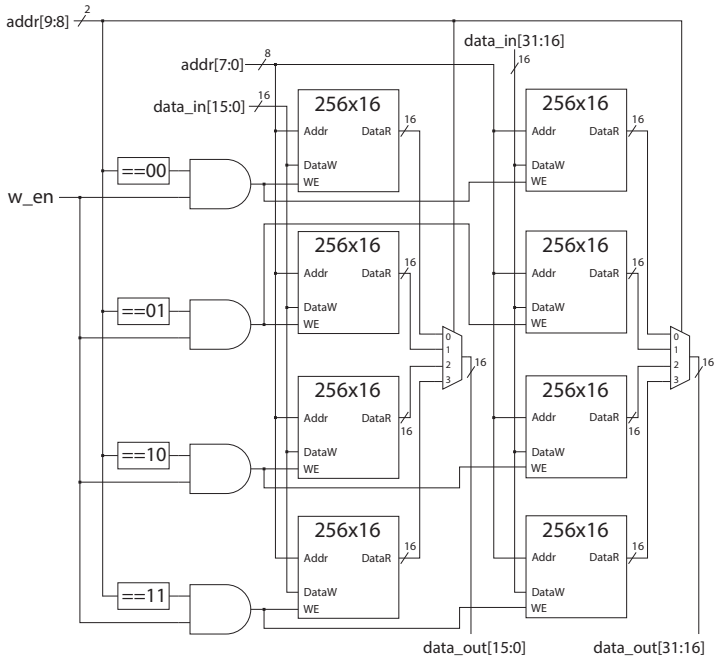
Note however that the assumption of external coordination between full and empty states means we can't be sure what happens when, for example, the FIFO is full and an external circuit both reads and writes at the same time. It's possible for us to support this, but we currently don't. Since behaviour in this case is not specified, it is omitted.



Problem 6: Memory Blocks [10pts]

You are given a memory block that is 256x16. Show how you would use multiple instances to design a memory that is 1Kx32.

Solution:



Problem 7: Memory Blocks [8pts]

Write the Verilog code for a synchronous read memory block that is 4Kx32 with two read ports.

Solution:

```
module RAM(
  input clk,
  input [11:0] addr_rd_a,
  input [11:0] addr_rd_b,
  input [11:0] addr_wr,
  input [31:0] din,
  input w_en,
  output [31:0] dout_a,
  output [31:0] dout_b
);

  reg [31:0] data [4095:0];
  reg [11:0] addr_rd_a_reg;
  reg [11:0] addr_rd_b_reg;

  always @(posedge clk) begin
    //Write
    if(w_en) begin
      data[addr_wr] <= din;
    end
```

```
    //Address register
    addr_rd_a_reg <= addr_rd_a;
    addr_rd_b_reg <= addr_rd_b;
  end

  //Read based on registered addr (syncronous)
  assign dout_a = data[addr_rd_a_reg];
  assign dout_b = data[addr_rd_b_reg];

endmodule
```

You can verify that the RAM is inferred as a synchronous read RAM by inspecting the output of Vivado. By looking at the synthesized design, this RAM is converted into 8 RAMB36E1 blocks. Each is configured with the synchronous property enabled. Note fro the Xilinx *7 Series FPGA Memory Resources* manual (`https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf`) that RAMB36E1 modules can be used in several configurations including one that is 4Kx9. For 32 bits, 4 of these RAMB36E1 modules are required. The two read ports are implemented by creating a copy of the RAM for a total of 8 RAMB36E1 blocks. The write address and din are fed to all RAMB36E1 modules.

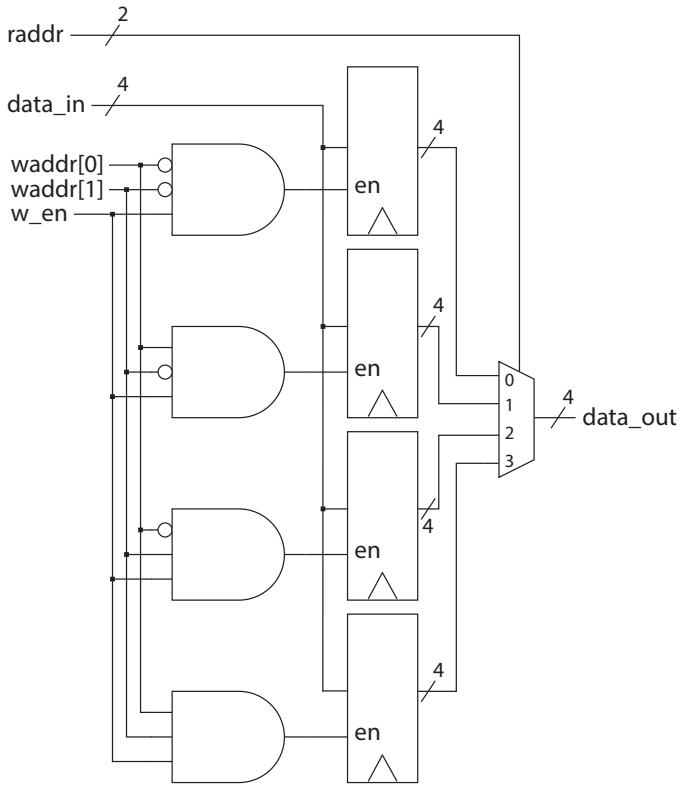
Problem 8: Memory Implementation

(a) [15pts] Consider the design of a (very) small asynchrous-read memory block of 4 words by 4-bits each. You want to implement the memory cells as positive edge-triggered flip-flops. Draw the circuit diagram for your design using the flip-flop cells, multiplexers, and logic gates.

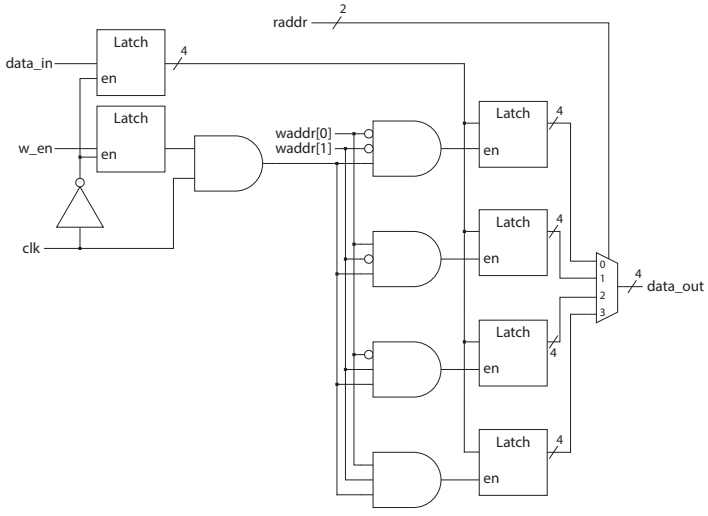
(b) **251A only**. [15pts] Now consider the redesign of the memory from part a) using latches instead of flip-flops. For this design, as above, the write operation occurs on the positive edge of the clock, but now the output data on a read become available after the failing edge of the clock.

Solution:

8.1 Part a



8.2 Part b



Note in this solution that left most latch is disabled when the positive clock edge occurs, saving the value of `data_in`. The `w_en` signal is also latched and saved on the positive clock edge.

While the clock is high, the latch associated with that address is allowed to go transparent, passing the latched value of `data_in`. On the negative edge of the clock, this latch becomes opaque and saves the value. Note that this structure looks very similar to how we construct Flip-Flops out of latches - this is not a coincidence. We effectively share the first latch in a Flip-Flop across all of the addresses.

There is a similar solution to what was presented here which uses a latch after the multiplexer instead of at `data_in`. This design has an added requirement that `w_en` and the write address must be stable before the preceding negative clock edge to avoid erroneously allowing one of the memory latches to become transparent.

Problem 9: Cache Implementation [12pts]

Consider the design of a 32KB direct mapped cache memory, with a block size of 8-Bytes, and a valid bit for each cache block. This cache will be used in a system with a 32 bit address space. How many total bits of memory would be needed to implement the cache? Sketch the block diagram of the cache implementation.

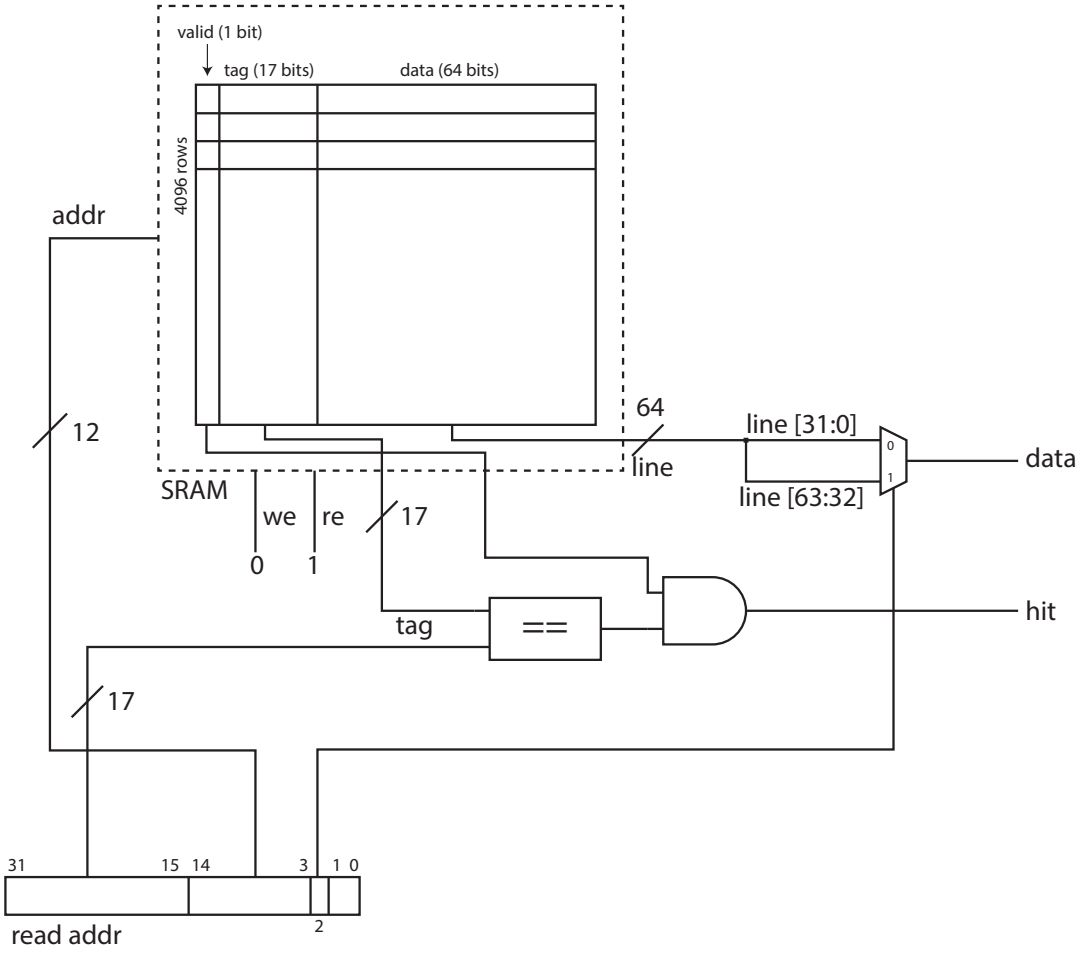
Solution:

The cache is 32 KB $= 32 \times 1024 = 32768$ bytes. Since each block (a.k.a. "line") is 8 bytes, the cache fits $32768/8 = 2^{5+10}/2^3 = 2^{12} = 4096$ blocks. We're told that each block needs 1 valid bit. A *modified* bit isn't necessary, presumably because the cache uses a write-through update policy.

To address each byte in a block takes $\log_2(8) = 3$ bits. To index each of 4096 blocks takes $\log_2(4096) = 12$ bits. In a 32-bit address space, that leaves $32 - 3 - 12 = 17$ of the most significant bits as identifying tags. (Another way to think about this is that addresses will map to the same byte in a cache block every 2^{12+3} addresses, and the upper 17 bits of address are needed to disambiguate such collisions.) We need to store 17 bits of tag information to uniquely associate blocks to their source addresses.

In all, that means we need $4096 \times (8 \times 8 + 17 + 1) = 335872$ bits for the cache: 4096 blocks and 64 bits of data per block, 17 bits of tag information, and 1 valid bit.

The cache implementation is as follows. A read operation is shown. On a read, `re` is enabled and `we` is disabled. Bits 3 to 14 (inclusive) of the CPU request address are used as the index into the cache. The upper 17 bits of the CPU request address are used as the tag: these are compared to the existing tag in the indexed set for equality. If they are equal, and the valid bit for that line is set, there is a hit. One bit of the request address selects the upper or lower word within the CPU line to return.



On a write, assuming a hit, the `re` input would be disabled and `we` would be enabled. The SRAM input data (not shown) would have the new contents of a the destination block present with the corresponding word within that block modified according to the CPU request (64 bits in total).