Kushal Gourikrishna
Jeremy Kaish
Jessica Li

**Design:**

For a 4 KB cache with 64 byte line size, 2 way set associative with LRU replacement, we used the bit assignments shown below in Figure 1. The valid bit tells us whether the block is in use, the dirty bit tells us whether it has been written to and needs to be written to memory when the block is evicted, and LRU tells us which block in the set is least recently used and should be evicted on a conflict miss. The tag is 23 bits to match the 23-bit tag after splitting the address into offset, index, and tag as shown below in Figure 2. After finding the indexed cache line and checking for matching tag, the offset is used to access the appropriate word within the 64 bytes.

Figure 1. Diagram of one cache line

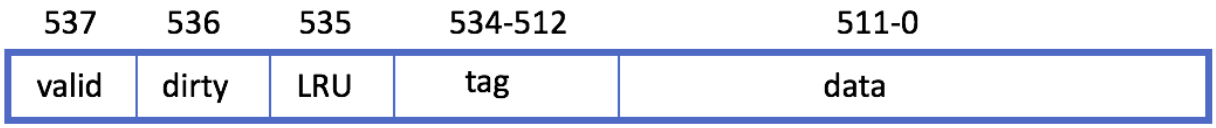
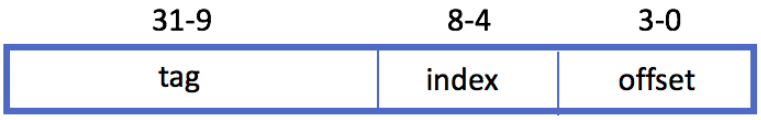



Figure 2. Diagram of one address




To take care of the timing between components and signals, we used a Finite State Machine. We used flip-flops to store states and combinational logic of different signals to compute the next state. There are four states and a variety of sequences for different events such as read/write hits and read/write misses; its summary is shown below in Figure 3.

Figure 3. Summary of Finite State Machine

Kushal Gourikrishna
Jeremy Kaish
Jessica Li


**Problems Encountered:**

CpuReady Signal:

The main issue we had with timing was the CpuReady signal. Since this signal controlled when the next request came in, it was important to synchronize it with the rest of the system. We initially had it un-clocked, but we eventually synchronized it with the rest of the system. However, we had an overall issue with timing which will be discussed in the coming sections.

General Timing Issues:

Synchronizing the entire system on a clock cycle was a big problem for our group. We made a mistake by only allowing one cycle for each stage and this caused some stages to overlap and addresses to be skipped over. This also messed with the counters and prevented them from working correctly as well as the writing to cache feature.

Kushal Gourikrishna
Jeremy Kaish
Jessica Li

Writing to Cache:

As mentioned above the timing problems messed with some of the writing features. However, the main issue was the way we approached our cache. We made the csram with large 512 bit blocks instead of making individual csrams for each word. This made writing difficult as we had to write over the entire block with the specific word change instead of just inputting the 32 bit value and changing the word. We had to then first access the entire block with a read request and then write over that value which was difficult to figure out timing wise.

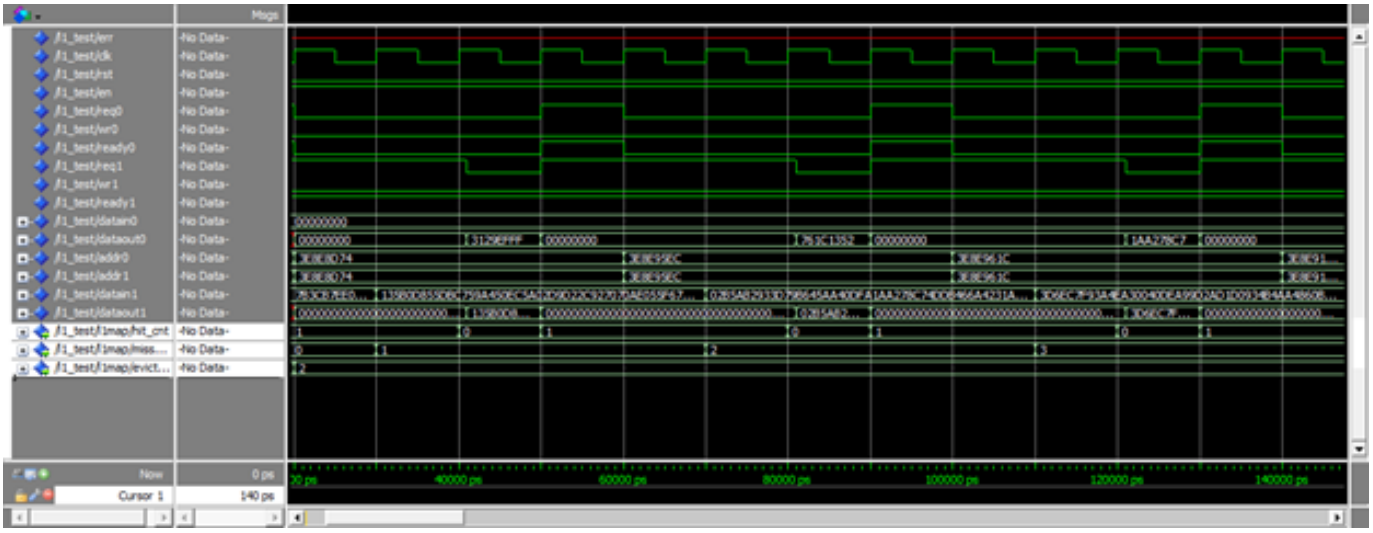
Counters:

The three counters (hit_cnt, miss_cnt, evict_cnt) all had issues. Due to our state machine only being 4 states, we had an issue with determining what the hit count logic would be. We only kept track of two states at a time, but we needed 3 states to determine if it was hit. The sequence of states, Idle -> Compare Tag -> Idle meant that we hit. To work around this we incremented the counter whenever the transition of Compare Tag to Idle occurred and decremented the counter whenever it went to the Allocate state which meant it missed. However, again due to timing the counter did not align with the rest of the design. The evict count was wrong due to reading the dirty bit wrong from the csram. The miss count did not align with the rest of the counters as well due to timing.

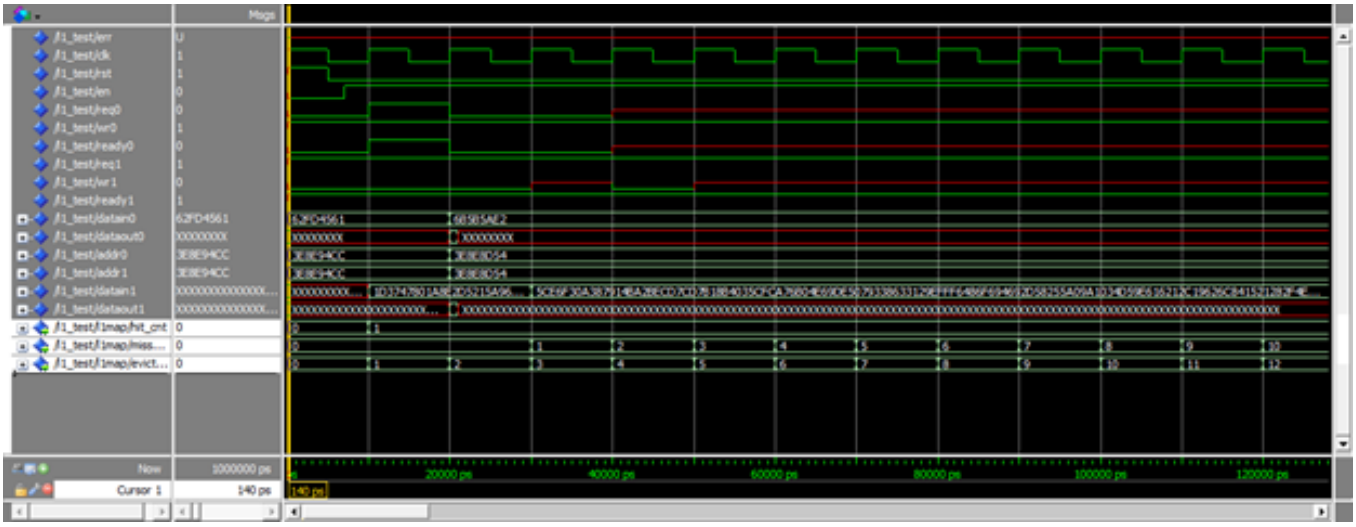
**Waveforms:**

Readonly:

Kushal Gourikrishna
Jeremy Kaish
Jessica Li

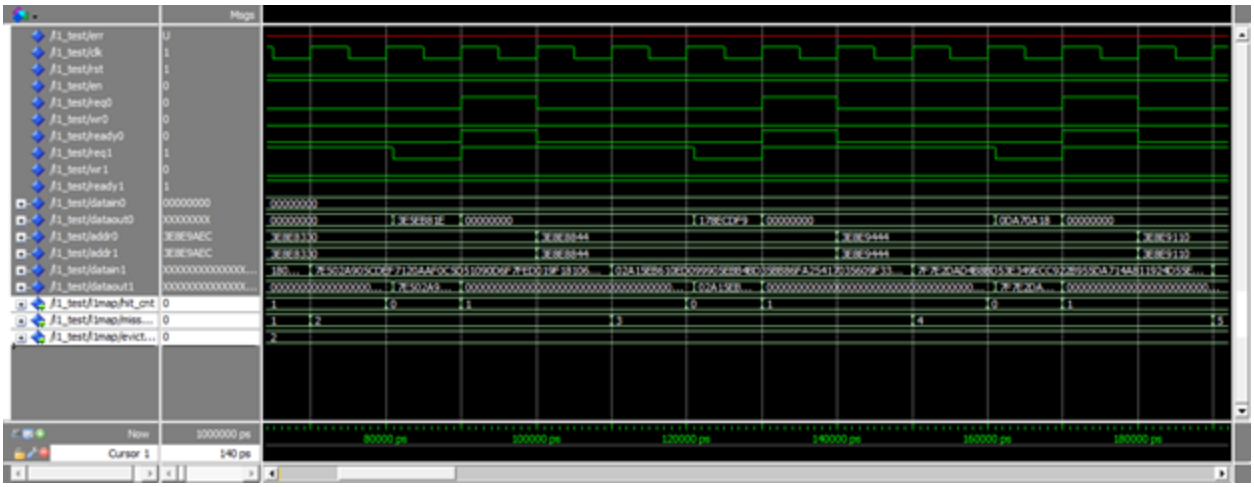


Write:



Random:

Kushal Gourikrishna
Jeremy Kaish
Jessica Li



You can see that the write waveform does not cycle through all of the addresses due to issues with our write functionality. However, the readonly and random waveforms go through the entire test file and at least output the correct read values.