Graehme Blair, Drew Hasse
EECS 361
Group Project 2: Cache
Project Report

Component Descriptions

cache.vhd:

Input/Output:

Cache takes in clk, ovrwr, addr, LRU, din and we signals. It outputs new_LRU, dout, miss and dirty signals. Clk is used to synchronize the we of the cache memory to the low half of the clock cycle to give the address time to reach the memory. Ovrwr is used to tell the cache that the allocate state is currently overwriting a block and that it does not matter if the tag of the block matches. Addr is the 26 bit tag and index of the block we are interested in. LRU is 1 bit representing whether or not the top line or the bottom line was the least recently used. Din is the data that would be written to the block at addr if we is high. New_LRU is 1 bit representing what line will be the least recently used after the current operation finishes. Dout is the data read of from the block at addr. Miss and dirty are 1 bit signals indicating whether addr misses or whether the hit block is dirty, respectively.

Function:

Cache maintains two csram instances with index width 5 and bit width 535. One each for the top and bottom lines of each index location. Cache takes addr and din and generates would be replacements for the top and bottom lines found at the index indicated by addr as if they were going to be written. The din to the top and bottom line csrams are determined by muxxes that choose either the top or the bottom as the would be writes and the other as a loopback. The csrams only write when clk is low, we is high, and there is a valid hit. On a read the output of the csrams is compared to the tag from addr using cmp_n components. If neither match, miss is set high. If there is a match, miss is set low and the dirty bit is set based on the dirty bit of the matched block. If there is a valid hit, dout is set the matched block, otherwise dout is set the the block indicated by LRU.

L1cache.vhd:

Input/Output:

L1cache takes in clk, rst, en, cpuReq, cpuWr, cpuAddr, cpuDin, l2Din and l2Ready signals. It ouputs cpuDout, cpuReady, l2req, l2Wr, l2Addr, l2Dout, hit_cnt, miss_cnt and evict_cnt signals. Clk, rst and enable are your typical control inputs. The input cpu signals get registered and are used in the state machine to inform the components as to what they are reading or writing and where. Cpu outputs are just the registered outputs from the cache. The l2 outputs are control for the lower level memory. The l2 inputs are the output of memory and its ready signal. (hit/miss/evict)_cnt are signals that keep track of how many hits, misses and evictions there have been, respectively.

Function:

L1cache maintains NextState_ctrl, RegWE_ctrl and cache instances and has registers for the current state, the last state, cpuWr, cpuAddr, cpuDin, cpuReady, cpuDout, the block-to-be-replaced's address, the block-to-be-replaced's data, the LRU bit of each index location and (hit/miss/evict)_cnt. The write enables of each of these registers is determined in the RegWe_ctrl component. The input of the current state register is determined by the NextState_ctrl component. The input of the cpu registers is just the input signals of the same name. the input of the cpu ready register is the cpuReady_we signal generated by the RegWE_ctrl component. The input of the cpuDout register is determined by a special mux that selects the 4 byte segment of the cache line corresponding to the index and offset of the input

addr. The block-to-be-replaced registers' inputs are determined by the cache component. The input of the LRU register is also determined by the cache component. The input of the (hit/miss/evict)_cnt registers is just the output + 1.

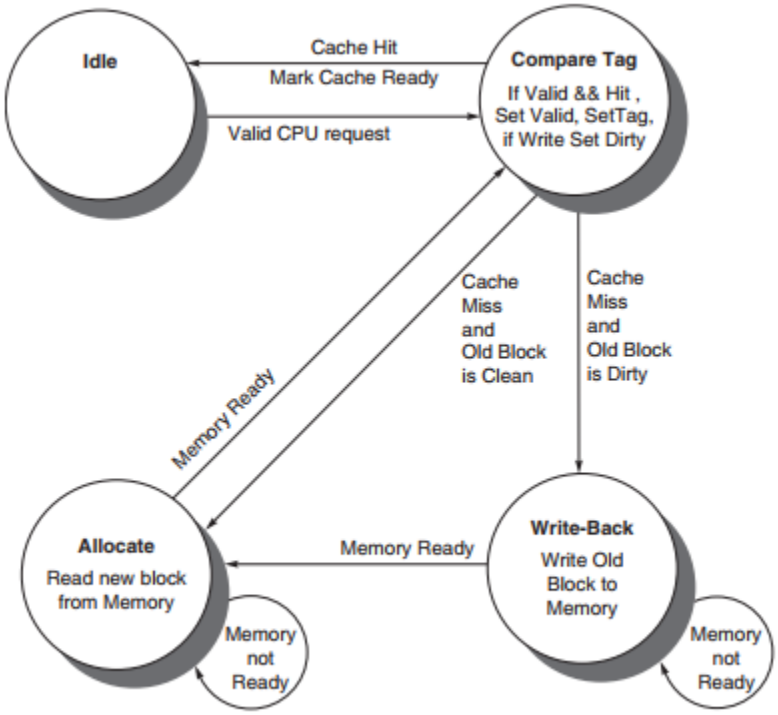


FIGURE 5.40 Four states of the simple controller.

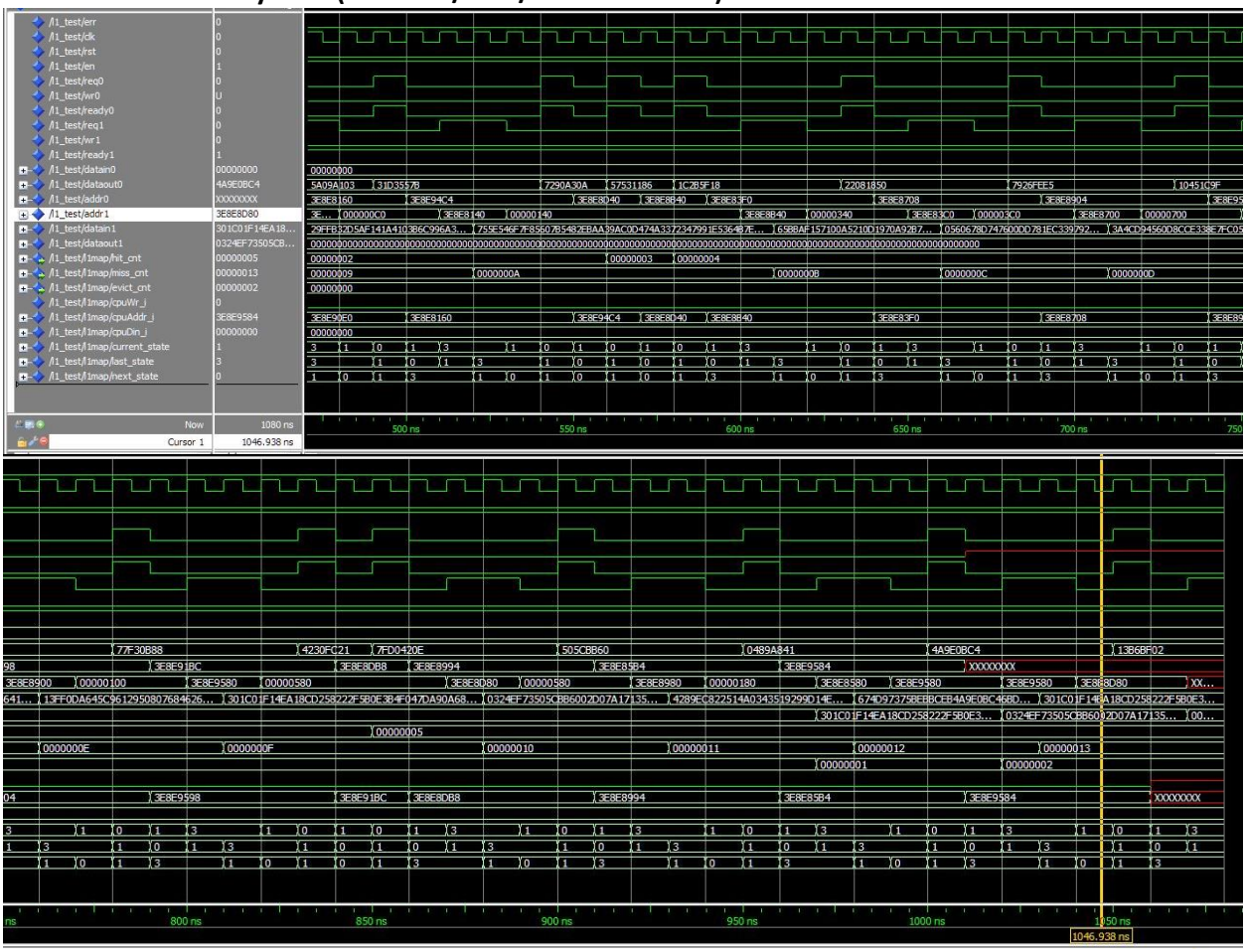
NextState_ctrl.vhd and RegWE_ctrl:

The control consists of two main components, a register write enable control unit, and a next state control unit. The register write control unit outputs the write enable signals for all the internal register based on the inputs which are: miss, cpuReq, dirty, current_state, and last state. The output signals from the register write enable unit are: cpuWr_we, cpuAddr_we, cpuDin_we, cpuDout_we, cpuReady_we, L2Addr_we, L2Dout_we, LRU_we, prevState_we, repAddr_we, repData_we, hit_we, miss_we, evict_we. The cpuWr_we, cpuAddr_we, and cpuDin_we signals all need to be high when there is a CPU request and it is in the idle state, because at the beginning of a new CPU request we need to register if it is a write, the data being fed in, and the address. The cpuDout_we and cpuReady_we signals need to be high when there is hit and it is in the compare tag state, because a hit in the compare tag state means that the data is now in the cache and another request can be issued. L2Addr_we is high when the current state is either writeback or allocate and L2Dout is high when the current state is allocate. This is because the allocate and writeback states interface with memory so we need to provide the address to memory at these states. The memory outputs a value after the allocate state which is why L2Dout_we is high in the allocate state. The repAddr_we and repData_we are high when there is a miss and it is in the compare tag state. We have these registers for the replacement address and data because we may need to write the data back that is being replaced if it is dirty. The hit_we signal is high when the previous state is idle, the current state is compare tag, and there is not a miss. This is because we only want the hit to count once, and there is only a transition from idle to compare tag once per request. The miss_we signal is determined by the last state being compare tag and the current state being either writeback or allocate. On a miss, the state machine transitions from compare tag to either writeback or allocate, so counting miss on these transitions ensures that a miss is counted only once.

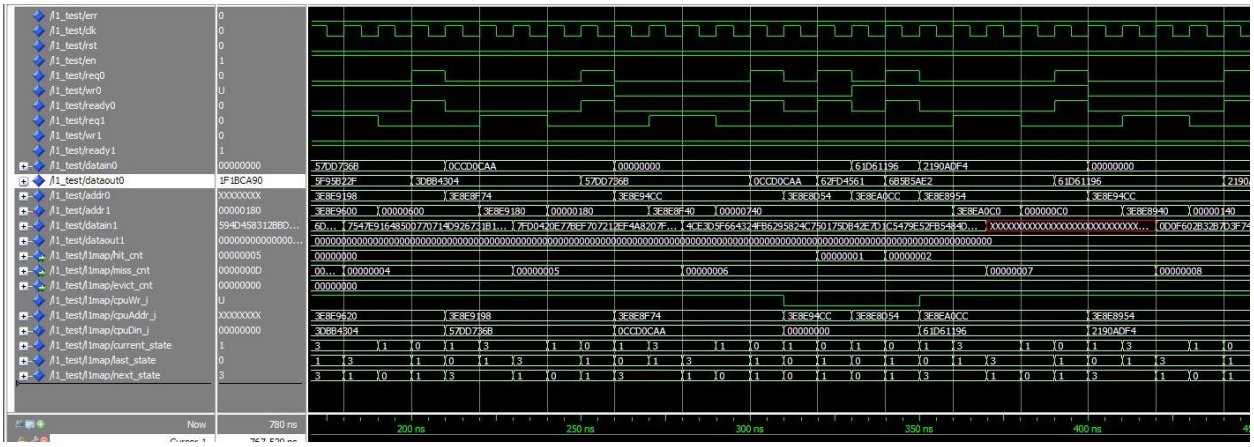
The other control unit is the next state control outputs the next state and has inputs: current_state, miss, cpuReq, L2Ready, dirty, prev_state. This control unit follows the image provided in the book and uses the same states: idle, compare tag, writeback and allocate. The next state is idle when the current state is compare tag and there is a hit because this means the data is in the cache. The next state is compare tag when the current state is allocate, the previous state is allocate and the memory is ready. We used the previous state because we wanted the FSM to stay in allocate for at least one cycle before transitioning to compare tag. The FSM goes to writeback when the current state is compare tag, there is a miss, and the evicted data is dirty. If these three things happen, that means the evicted data has changed in the cache, so it needs to be written back in memory. Finally, the next state is allocate when: the current state is writeback, the last state was writeback, and the memory is ready, or when the current state is compare tag, there is a miss, and the evicted data is not dirty, or when the current state is allocate and the memory is not ready.

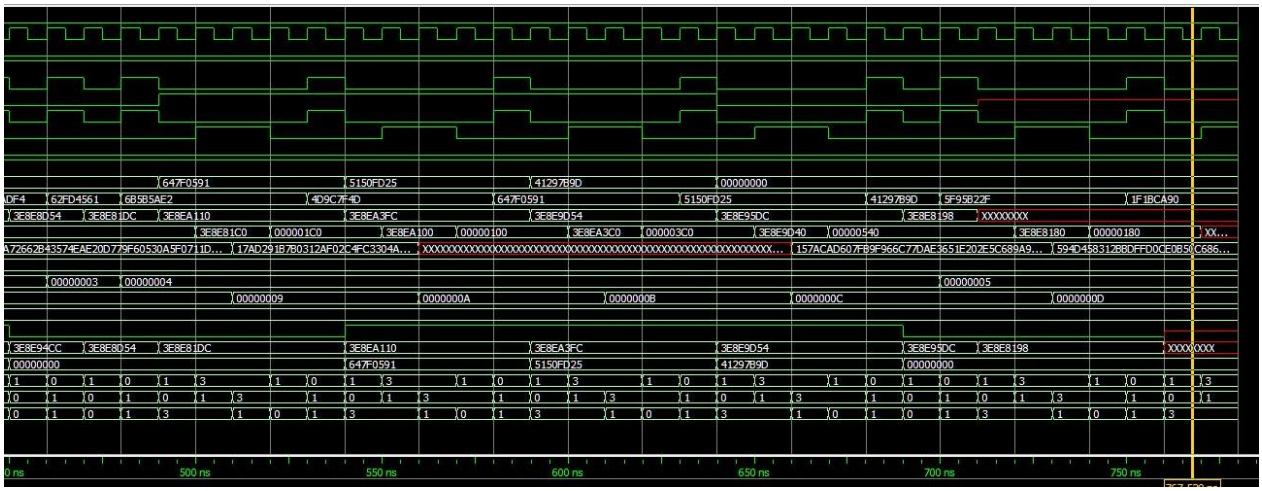
Traces (Next Page)

Trace of the read only test: (Note: hit/miss/evict are in hex)



Trace of the write only test: (Note: hit/miss/evict are in hex)





Trace of the random RW test: (Note: hit/miss/evict are in hex)