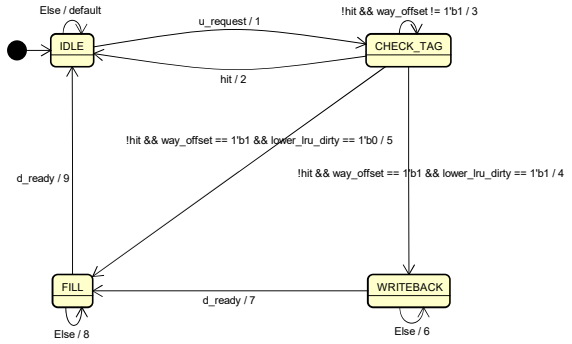
stm L1



default:
state = state;
way_offset = way_offset;
accessed_way = accessed_way;

u_ready = 1'b0;
u_dout = 8'hxx;

c_we = 1'b0;
c_dirty = 1'bx;
c_addr = {(INDEX_BITS+1){1'bx}};

d_request = 1'b0;
d_addr = 6'hxx;
d_we = 1'bx;
d_din = {32{1'bx}};

1:
way_offset = 1'b0;

c_addr = {r_index, way_offset};

state = CHECK_TAG;

2:
accessed_way = way_offset;

c_addr = {r_index, way_offset};
c_we = r_we;
c_dirty = 1'b1;

u_dout = f_data[r_block_offset];
u_ready = 1'b1;

state = IDLE;

This is implementing both read and write, by setting c_we = r_we and setting the dirty to 1. The u_dout will still output trash data on the write, but the processor shouldn't store the data coming from a write.

This could be changed by adding an if(r_we), which would effectively create another transition, one for (hit & r_we) and another for (hit & !r_we).

3:
way_offset = way_offset + 1'b1;

c_addr = {r_index, way_offset};

4:
accessed_way = lower_lru_way_offset;

d_addr = {tag_lower_lru_way, r_index};
d_we = 1'b1;
d_din = block_lower_lru_way;
d_request = 1'b1;

state = WRITEBACK;

6:
d_addr = {tag_lower_lru_way, r_index};
d_we = 1'b1;
d_din = block_lower_lru_way;

Transitions 6 and 8 are simmilar to the transitions that get into that state, except removing the d_request = 1'b1; Keeping this signal would result in the down hierarchy acknowledging two requests, which could mess sequential requests from the up hierarchy, as the d_ready would signal the end of an unwanted request, skipping the real request.

5:
accessed_way = lower_lru_way_offset;

c_we = 1'b1;
c_dirty = r_we;
c_addr = {r_index, lower_lru_way_offset};

d_addr = {r_tag, r_index};
d_we = 1'b0;
d_request = 1'b1;

state = FILL;

7:
c_we = 1'b1;
c_dirty = r_we;
c_addr = {r_index, lower_lru_way_offset};

d_addr = {r_tag, r_index};
d_we = 1'b0;
d_request = 1'b1;

state = FILL;

8:
c_we = 1'b1;
c_dirty = r_we;
c_addr = {r_index, lower_lru_way_offset};

d_addr = {r_tag, r_index};
d_we = 1'b0;

9:
u_dout = f_data[r_block_offset];
u_ready = 1'b1;

state = IDLE;

The L2 cache also updates the d_ready and d_dout on the posedge. The controller only recognizes the u_ready one cycle after u_ready and u_dout have been set. Since the c_din is set to the d_dout, the block has already been filled during the last negedge.

This is also implementing for both read and write.