FIGURE e5.12.1 Type declarations in SystemVerilog for the cache tags and data. The tag field is 18 bits wide and the index field is 10 bits wide, while a 2-bit field (bits 3–2) is used to index the block and select the word from the block. The rest of the type declaration is found in the following figure.

FIGURE e5.12.2 Type declarations in SystemVerilog for the CPU-cache and cache-memory interfaces. These are nearly identical except that the data is 32 bits wide between the CPU and cache and is 128 bits wide between the cache and memory.

FIGURE e5.12.3 Block diagram of the simple cache using the Verilog names. Not shown are the write enables for the cache tag memory and for the cache data memory, or the control signals for multiplexors that supply data for the Data Write variable. Rather than have separate write enables on every word of the cache data block, the Verilog reads the old value of the block into Data Write and then updates the word in that variable on a write. It then writes the whole 128-bit block.

FIGURE e5.12.4 Cache data and tag modules in SystemVerilog. These are nearly identical except that the data is 32 bits wide between the CPU and cache and is 128 bits wide between the cache and memory. Both only write on positive clock edges if the write enable is set.

FIGURE e5.12.5 FSM in SystemVerilog, part I. These modules instantiate the memories according to the type definitions in the previous figure.

FIGURE e5.12.6 FSM in SystemVerilog, part II. This section describes the default value of all signals. The following figures will set these values for one clock cycle, and this Verilog will reset it to these values for the following clock cycle.

FIGURE e5.12.7 FSM in SystemVerilog, part III. Actual FSM states via case statement in this figure and the next. This figure has the Idle state and most of the Compare Tag state.

FIGURE e5.12.8 FSM in SystemVerilog, part IV. Actual FSM states via the case statement in the prior figure and this one. This figure has the last part of the Compare Tag state, plus Allocate and Write-Back states.

FIGURE e5.12.9 The cache coherence mechanism receives requests from both the processor and the bus and responds to these based on the type of request, whether it hits or misses in the cache, and the state of the cache block specified in the request. The fourth column describes the type of cache action as normal hit or miss (the same as a uniprocessor cache would see), replacement (a uniprocessor cache replacement miss), or coherence (required to maintain cache coherence); a normal or replacement action may cause a coherence action depending on the state of the block in other caches. For read misses, write misses, or invalidates snooped from the bus, an action is required only if the read or write addresses match a block in the cache and the block is valid. Some protocols also introduce a state to designate when a block is exclusively in one cache but has not yet been written. This state can arise if a write access is broken into two pieces: getting the block exclusively in one cache and then subsequently updating it; in such a protocol this “exclusive unmodified state” is transient, ending as soon as the write is completed. Other protocols use and maintain an exclusive state for an unmodified block. In a snooping protocol, this state can be entered when a processor reads a block that is not resident in any other cache. Because all subsequent accesses are snooped, it is possible to maintain the accuracy of this state. In particular, if another processor issues a read miss, the state is changed from exclusive to shared. The advantage of adding this state is that a subsequent write to a block in the exclusive state by the same processor need not acquire bus access or generate an invalidate, since the block is known to be exclusively in this cache; the processor merely changes the state to modified. This state is easily added by using the bit that encodes the coherent state as an exclusive state and using the dirty bit to indicate that a block is modified. The popular MESI protocol, which is named for the four states it includes (modified, exclusive, shared, and invalid), uses this structure. The MOESI protocol introduces another extension: the “owned” state.

FIGURE e5.12.10 A write-invalidate, cache-coherence protocol for a write-back cache, showing the states and state transitions for each block in the cache. The cache states are shown in circles, with any access permitted by the processor without a state transition shown in parentheses under the name of the state. The stimulus causing a state change is shown on the transition arcs in regular type, and any bus actions generated as part of the state transition are shown on the transition arc in bold. The stimulus actions apply to a block in the cache, not to a specific address in the cache. Hence, a read miss to a block in the shared state is a miss for that cache block but for a different address. The left side of the diagram shows state transitions based on actions of the processor associated with this cache; the right side shows transitions based on operations on the bus. A read miss in the exclusive or shared state and a write miss in the exclusive state occur when the address requested by the processor does not match the address in the cache block. Such a miss is a standard cache replacement miss. An attempt to write a block in the shared state generates an invalidate. Whenever a bus transaction occurs, all caches that contain the cache block specified in the bus transaction take the action dictated by the right half of the diagram. The protocol assumes that memory provides data on a read miss for a block that is clean in all caches. In actual implementations, these two sets of state diagrams are combined. In practice, there are many subtle variations on invalidate protocols, including the introduction of the exclusive unmodified state, as to whether a processor or memory provides data on a miss.

FIGURE e5.12.11 Cache coherence state diagram with the state transitions induced by the local processor shown in black and by the bus activities shown in gray. As in Figure e5.12.10, the activities on a transition are shown in bold.