Flow Specification of Cache Coherence Protocol in GEM5

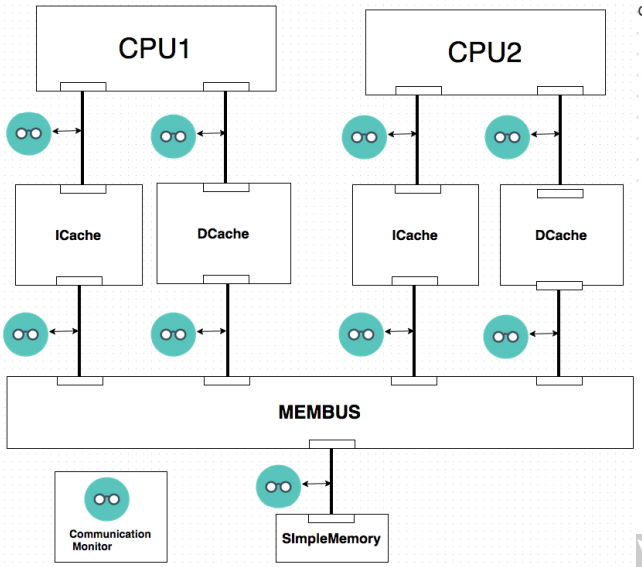


Fig. 1. SoC platform structure.

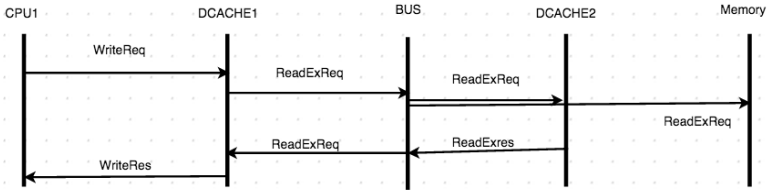


Fig. 2. Flow sequence chart of write operation when requested data is not included in Dcache. ReadExRes can also be sent from Memory if Dcache2 doesn't have requested data. This sequence chart is symmetric for CPU2.

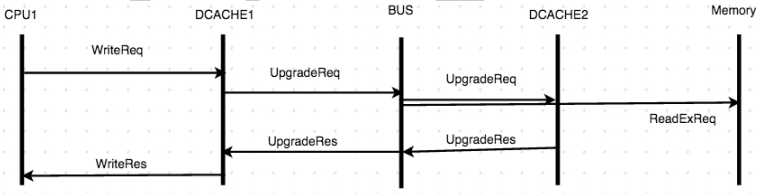


Fig. 3. Flow sequence chart of write operation when requested data is shared by another component. UpgradeRes can also be sent from Memory if Dcache2 doesn't have requested data. This sequence chart is symmetric for CPU2.

***Abstract*—This paper will go through a brief overview for the implemented instances of given GEM5 flow specifications. These flow specifications capture how messages are exchanged for different use cases.**

The flow specifications are implemented in a SoC model using GEM5. The architecture of this SoC model is shown in Fig. 1. This SoC model consists of two ARM Cortex-A9 cores, each of which contains two separate 16KB data and instruction caches. The caches are connected to a 1GB memory through a memory bus model. .

In this model, components communicate with each other by sending and receiving various request and response messages. In order to observe and trace communications occurring inside this model during execution, monitors are attached to links connecting the components. These monitors record the messages flowing through the links they are attached to, and store them into output trace files. The recorded message is defined in the following format, (Src,Dest,Cmd), where Src and Dest refer to the source and destination components of messages, Cmd refers to the operations that the destination component should perform.

I. Write Flow Specification

In this section, we will take CPU1 as an example to explain how write flow specification works in our SoC system.

Coherence protocol for write operation can be triggered by CPU1 initiating a memory write request to its data cache. Next, depends on whether the required data is included in cache1, and the status of requested data, cache1 can generate three possible responses.

First, the cache1 will send read exclusive request message msg2 to interconnect if data is not present in cache, as shown in Fig. 2.

Second, if the data is shared by CPU2, it will send upgrade request message msg16 to interconnect request disabling CPU2's ownership of the data, as shown in Fig. 3.

Last, if CPU1 has exclusive right of required data, ICache1 will perform the write operation and sent an response message msg15 to CPU1, as shown in Fig. 4.

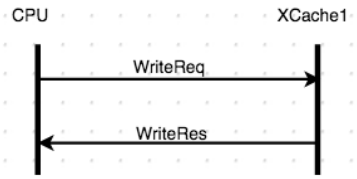
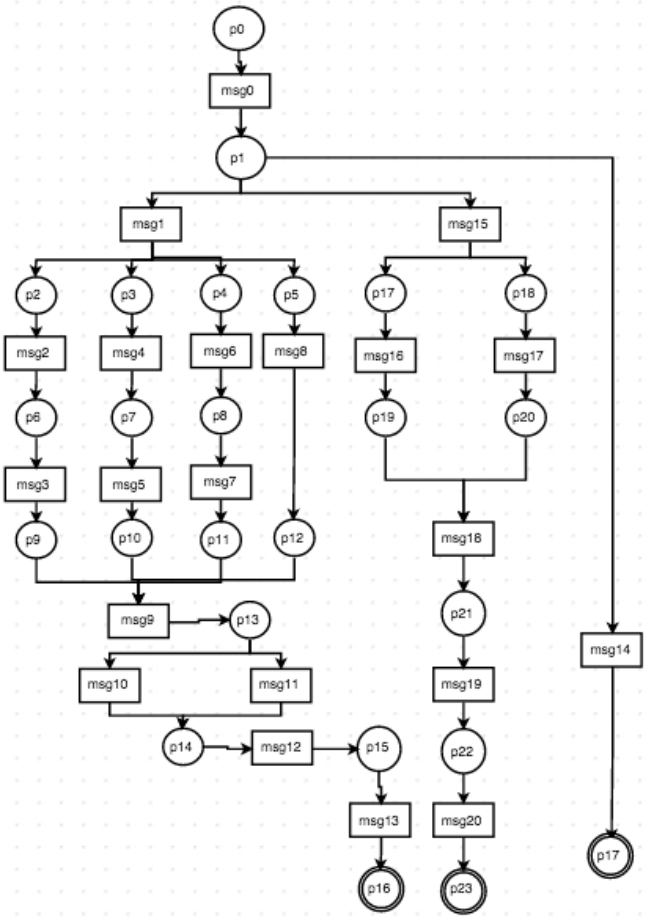


Fig. 4. Flow sequence chart of write operation when XCache has the exclusive right of requested data. XCache can be instruction cache or data cache. This sequence chart is symmetric for CPU2.



msg_0 : (CPU1, writeReq, icache1)
msg_1 : (dcache1, readExreq , Bus)
msg_2 : (Bus, readExreq, dcahce2)
msg_3 : (dcache2, readExreq, cpu2)
msg_4 : (Bus, readExreq, icahce2)
msg_5 : (icache2, readExreq, cpu2)
msg_6 : (Bus, readExreq, icahce1)
msg_7 : (dcache1, readExreq, cpu1)
msg_8 : (Bus, readExreq, Memory)
msg_9 : (true)
msg_{10} : (Memory, readExres, Bus)
msg_{11} : (icache2, readExres, Bus)
msg_{12} : (Bus, readExres, dcache1)
msg_{13} : (icache1, writeRes, CPU1)
msg_{14} : (icache1, writeRes, CPU1)
msg_{15} : (dcache1, UpgradeReq, Bus)
msg_{16} : (Bus, UpgradeReq, icahce2)
msg_{17} : (Bus, UpgradeReq, Memory)
msg_{18} : (icache2, UpgradeRes, Bus)
msg_{19} : (Bus, UpgradeRes, icache1)
msg_{20} : (icache1, WriteRes, CPU1)

Fig. 5. Flow specification of a cache coherent write operation initiated from CPU1. This flow is symmetric for CPU2.

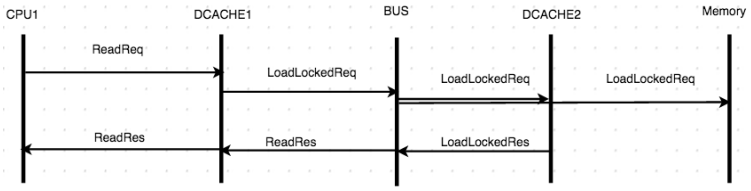


Fig. 6. Flow sequence chart of read operation when requested data is shared by another component. LoadLockedRes can also be sent from Memory if Dcache2 doesn't have requested data. This sequence chart is symmetric for CPU2.

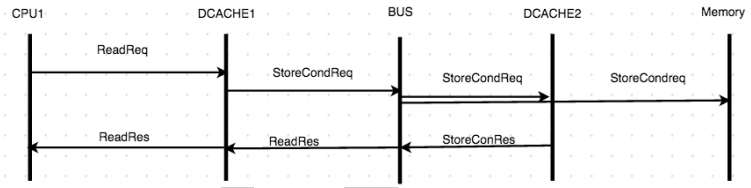


Fig. 7. Flow sequence chart of read operation when requested data is not present. StoreCondRes can also be sent from Memory if Dcache2 doesn't have requested data. This sequence chart is symmetric for CPU2.

Afterwards, interconnect will sent received request to all connected component (data cache and instruction cache of each CPU and memory). Once the interconnect obtains the response from either CPU2 or memory, it will generate a response message to cache1, cache1 will generate a write response message msg14 (or msg 21) to CPU1. These three flow sequence charts are implemented together in one flow specification in GEM5, as specified in Fig. 5.

II. READ FLOW SPECIFICATION

In this section, we will also take CPU1 as an example to explain how read flow specification works in our SoC system.

Read coherence protocol can be triggered by CPU1 initiating a memory read request to its cache. Depending on whether the required data is included in its cache, the status of requested data, and which cache the request is sent to, cache can generate three possible responses.

First, for both instruction cache and data cache, if they have the exclusive right of requested data, they can generate a read response carrying the data back to CPU. Detailed flow sequence chart is displayed in Fig. 8.

Second, when ICache1 decide that the requested data is not present, it will issue an StoreCondReq to memory bus asking for data.

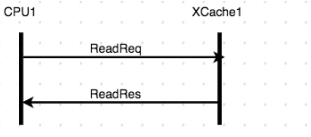


Fig. 8. Flow sequence chart of read operation when XCache has the exclusive right of requested data. XCache can be instruction cache or data cache. This sequence chart is symmetric for CPU2.

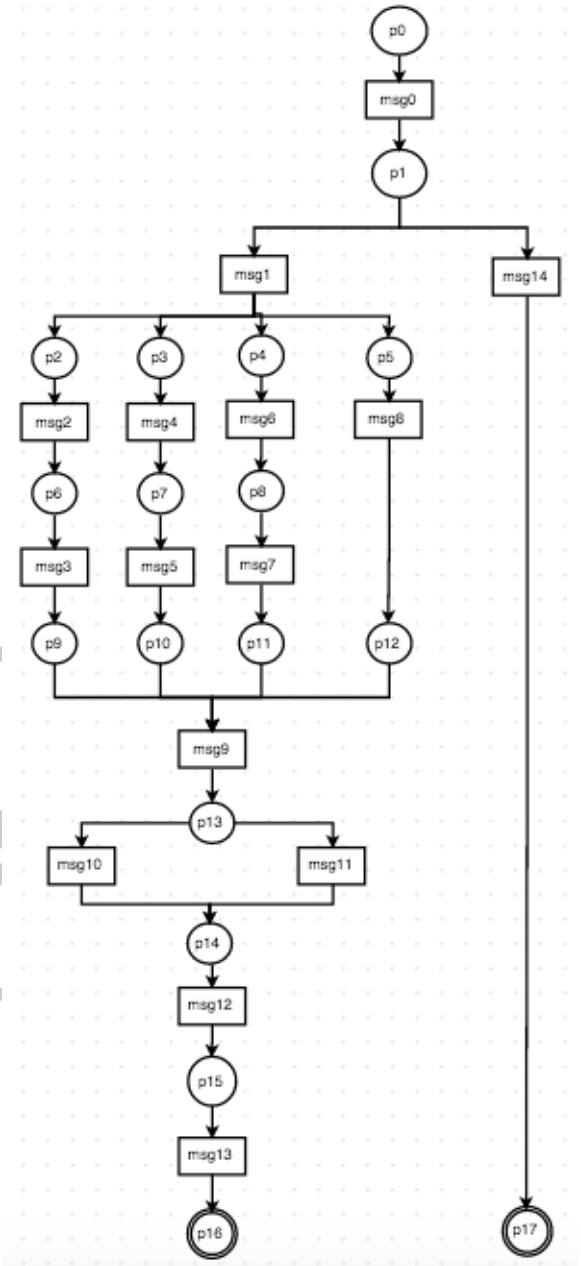
Finally, if DCache1 has the data, but it's shared, it will issue an LoadLockedReq to memory bus requesting exclusive right of the data. Flow sequence charts for these two possibility are shown in Fig.r̃eread1 and Fig.r̃eread.

Since instruction cache and data cache have different coherence protocol, there are two flow specifications for read operation generated by CPU1. The LPN specification as shown in Fig. 9 captures the system flow where CPU1 initiate a memory read to instruction cache. And Fig. 10 specifies read operation to data cache from CPU1.

III. FAILED READ SPECIFICATION

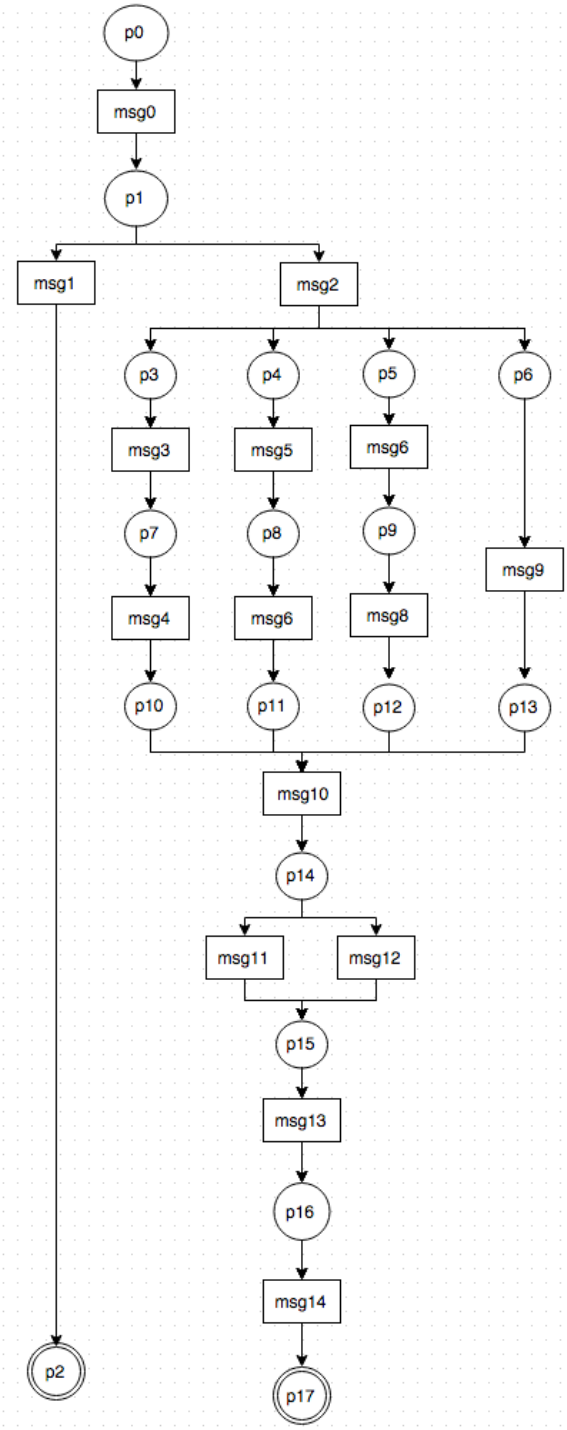
Another flow implemented in GEM5 is when CPU initiate a store failed request to instruction cache, with a simple store failed response from instruction cache. Detailed flow specification is specified in Fig. 11

This three specifications are also symmetric for CPU2.



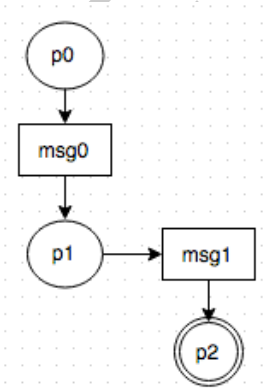
$msg0$: (CPU1, ReadReq, icache1)
$msg1$: (dcache1, StoreCondreq , Bus)
$msg2$: (Bus, StoreCondreq, icahce2)
$msg3$: (icache2, StoreCondreq, cpu2)
$msg4$: (Bus, StoreCondreq, dcahce2)
$msg5$: (dcache2, StoreCondreq, cpu2)
$msg6$: (Bus, StoreCondreq, dcahce1)
$msg7$: (icache1, StoreCondreq, cpu1)
$msg8$: (Bus, StoreCondreq, Memory)
$msg9$: (true)
$msg10$: (Memory, ReadRes, Bus)
$msg11$: (icache2, ReadRes, Bus)
$msg12$: (Bus, ReadRes, dcache1)
$msg13$: (icache1, ReadRes, CPU1)
$msg14$: (icache1, ReadRes, CPU1)

Fig. 9. Flow specification of a cache coherent read operation initiated from CPU1 to instruction cache.This flow is symmetric for CPU2.



$msg0$: (CPU1, ReadReq, dcache1)
$msg1$: (dcache1, ReadRes, CPU1)
$msg2$: (icache1, LoadLockedreq , Bus)
$msg3$: (Bus, LoadLockedreq, dcahce2)
$msg4$: (dcache2, LoadLockedreq, cpu2)
$msg5$: (Bus, LoadLockedreq, icahce2)
$msg6$: (icache2, LoadLockedreq, cpu2)
$msg7$: (Bus, LoadLockedreq, dcahce1)
$msg8$: (icache1, LoadLockedreq, cpu1)
$msg9$: (Bus, LoadLockedreq, Memory)
$msg10$: (true)
$msg11$: (Memory, ReadRes, Bus)
$msg12$: (icache2, ReadRes, Bus)
$msg13$: (Bus, ReadRes, icache1)
$msg14$: (dcache1, ReadRes, CPU1)

Fig. 10. Flow specification of a cache coherent read operation initiated from CPU1 to data cache. This flow is symmetric for CPU2.



$msg0$: (CPU1, StoreCondFailedReq, ICache1)
$msg1$: (ICache1, StoreCondFailedRes, CPU1)

Fig. 11. Flow specification of failed read. This flow is symmetric for CPU2.