Modeling Cache Memory in Multiprocessor Systems Using the DEVS Framework

Kyou H. Lee†, Won H. Lee†† and Tag G. Kim†††

†Processor Architecture Section, ETRI, Daejon, Korea
††Supercomputing Reserach Center, Chung-Ang University, Seoul, Korea
†††Department of Electrical Engineering, KAIST, Daejon, Korea

ABSTRACT

It is known that throughput of a tightly coupled multiprocessor system largely depends on performance of its memory subsystem. Employing cache improves performance of such a multiporcessor system by having a copy of most frequently used data for quick reference. This paper describes performance modeling of a shared bus, a shared memory multiprocessor system with physical and virtual cache. We employ Zeigler's DEVS (Discrete Event Systems Specification) formalism for modeling such multiprocessor systems in a hierarchical, modular manner.

1. INTRODUCTION

It is known that throughput of a tightly coupled multiprocessor system largely depends on performance of its memory subsystem. Employing cache improves performance of such a multiprocessor system by having a copy of most frequently used data for quick reference. Among many parameters in cache design, addressing scheme is one of most important factors[Smith 82]. There are two types of addressing scheme: physical cache, operated on physical address, and virtual cache, operated on virtual address. Physical cache obtains addresses from the output of the TLB(Table Lookaside Buffer), a specialized cache for address translation, and virtual cache from the processor's address lines. Performance of a multiprocessor system with cache depends not only on the above cache addressing schemes but on bus contention associated with cache miss and data coherency. This paper describes performance modeling of a shared bus, a shared memory multiprocessor system with physical and virtual cache. Since a multiprocessor system can be viewed as in hierarchical structure, we employ a hierarchical, modular DEVS(Discrete Event Systems Specification) formalism[Zeigler 84] in modeling the multiprocessor systems. The DEVS formalism, developed by Zeigler, offers a powerful means to model discrete event systems in a systematic manner. Models developed here can be easily coded in realizations of the formalism, such as DEVS-Scheme[Kim 90] or DEVSIM++[Kim 92].

This paper is organized as follows. Section 2 briefly describes the DEVS formalism. Section 3 discusses the arguments of multiprocessor system considered in this paper and modeling assumptions. In Section 4, the development of hierarchical coupled models of multiprocessor systems is described. Finally, Section 5 offers concluding remarks and discusses the future work.

2. THE DEVS FORMALISM IN BRIEF

A set-theoretic formalism, the DEVS formalism [Zeigler 84], specifies discrete event models in a hierarchical, modular form. Within the formalism, one must specify 1) the basic models from which larger ones are built, and 2) how these models are connected together in hierarchical fashion. A basic model, called an atomic model (or atomic DEVS), has specification for dynamics of the model. An atomic model M is specified as [Zeigler 84]:

$$M = < X, S, Y, ext, int, out, ta >$$

where X: the input event set;
S: the sequential state set;
Y: the output event set;
ext: the external transition function;
int: the internal transition function;
out: the output function;
ta: the time advanced function.

Details of the functions above including constraints can be found in [Zeigler 84]. The second form of the model, called a coupled model (or coupled DEVS), tells how to couple (connect) several component models together to form a new model. This latter model can itself be employed as a component in a larger coupled model thus giving rise to construction of complex models in hierarchical fashion. A coupled model DN is defined as in [Zeigler 84]:

$$DN = < D, \{Mi\}, \{Ii\}, \{Zi,j\}, SELECT >$$

where D: the component names set;
for each i in D
Mi: DEVS for component i in D;
Ii: influencees set of i;
for each j in Ii
Zi,j: i-to-j output translation function;
SELECT: the tie breaking selector.

Detail descriptions for the formalism can be found in [Zeigler 84].

3. MODEL ISSUES

3.1 Multiprocessor systems modeling

Physical cache has a disadvantage compared with virtual cache in time delay associated with address translation through TLB before accessing cache. It can be sped up by partially overlapping the cache access with address translation. However, such a method has a limitation in cache size because the only part of the address bits available can be used to access cache[Hill 86]. Virtual cache in nature has no shortcomings discussed above. However, virtual cache has a serious problem, so-called "synonym problem". There are a number of possible ways to deal with this synonym problem[Hill 85][Hill 86][Goodman 87]. One way to detect synonyms is using IMT(Inverse Mapping Table)[Hwang 84][Lee 88]. Goodman[Goodman 87] proposed the organization for dual-directory cache, which also can be one implementation of such an inverse mapping table, to detect synonyms. Having the advantage of decreasing the cache access time without the limitation of cache size on a hit, virtual cache must tolerate the cost of increased hardware complexity.

Performance of a multiprocessor system with cache depends not only on the efficiency of cache access but on the contention of bus. The bus requests are typically occurs when one of three issues is required: cache entry update on cache miss, data coherency issues including main memory update, and prefetch of cache entry to fill. The dynamics of requesting bus in a multiprocessor system with physical cache is much different from that with virtual cache. The processor that cannot have a bus grant must wait. Such a contention in shared bus is affecting on the total throughput of the multiprocessor system. It may be of great interest to compare the total system performances between two configurations.

3.2 Modeling assumptions

The systems considered in this paper are tightly coupled multiprocessor systems with shared buses and shared memories. Followings are assumed to develop the system models:

1. A multiprocessor system separates instruction cache from data cache to efficiently support the pipeline operation of a processor. The simulation model of the processor is developed to support fully separate transactions both for instruction fetches and for data references.

2. For the data coherency, it is assumed that the simple "write through" main memory update policy and the bus snooping mechanism are adopted. An inverse mapping table is adopted for virtual cache to detect synonyms.

3. If a TLB miss is accepted from the transaction of data references, then all transactions currently under processing are discarded. After then, the processor switches to begin a new transaction for the TLB miss handling. If a TLB miss is accepted from the transaction of instruction fetches on data reference currently under processing, then the processor switches to a TLB miss handling rouitne after finishing the current transaction of data reference. On the routine, it is assumed to access the main memory N times periodically.

4. It is assumed that there are no request issues for the prefetch of cache entries to fill. This may rarely affect on performances because the same assumptions are adopted for each case.

5. Two or more processing elements simultaneously request the same bus, each of which has separate instruction and data caches that independently request the same bus.

3.3 The operation flows

Two multiprocessor systems considered in this paper are shown in Figure 3.1 and Figure 3.2. The brief descriptions of their operations are as follows:

(1) Multiprocessor system with physical caches

When virtual address is generated in Processor in Figure 3.2 for external references, namely instruction fetch and/or data reference, it is sent to P_TLB for address transaltion. P_TLB retains copies of recently used virtual to physical translation pairs. If a TLB miss occurs, P_TLB signals a TLB miss trap to the Processor. The Processor, on the occurrence of a TLB miss, switches to handle the TLB miss, and reattempts. If the physical address is available(TLB hit), it is presented to the P_Cache. If the corresponding cache entry is found(cache hit), and the attempted transaction is for a read, it is then returned to the Processor. On a write in the write-through policy, if the necessary entry is in the P_Cache(cache write hit), the P_Cache updates the entry. At the same time, the original one in main memory is also updated.

When a cache write miss in the P_Cache occurs, there are two possibilities. Not only the original one in main memory are updated, but the missed entry is fetched from main memory. Otherwise, no fetch on a write hit is performed. Instead, only update in main memory is performed. However, in the simulation modeling view point, both are required the same processing time. When a write miss occurs, it issues a global processing via bus. If a cache miss occurs or if a write in write through policy is issued, P_Cache requests bus and waits until a bus grant. Upon receiving a bus grant, the physical address of the necessary information is sent to main memory, and the information is returned to be loaded into the P_Cache.

Being requested with data coherency or a main memory update, bus signals to all other Snoopers. Then, bus snooping is enabled in P_Cache. This ensures that data in the local P_Cache is updated or invalidated when that data is written to by another P_Cache. Several types of bus snooping protocols are proposed[Katz 84][Cho 86]. However, their simulation models can be the same. If a snoop hit occurs after receiving the input from bus, Snooper invalidates or updates the associated entry of P_Cache. But the Snooper discards the input if a snoop miss occurs.

(2) Multiprocessor system with virtual cache

When the virtual address is available from Processor in Figure 3.1, it is presented to the V_Cache. If the corresponding cache entry is found(cache hit), and the attempted transaction is for a read, then it is returned to the Processor.

If a cache miss occurs or a write with write through policy is issued, the given virtual address is transferred to V_TLB to translate into physical address. After that, the V_Cache waits until a bus grant. If the physical address is available(TLB hit), it is presented to request bus. At the same time, the given physical address is sent to IMT in V_Snooper, which can search all the synonyms in the V_Cache under the different names. If a V_Snoop hit occurs on a read miss, then it is returned to the Processor and the request of bus is discarded; if a V_Snoop hit occurs on a write, then the entry that hit is invalidated or updated. If a V_snoop miss occurs, then no actions occurs. when a write miss or hit with the write through policy occurs upon receiving a bus grant, V_Cache attempts to access main memory and updates the original one in it. For a read miss, the necessary information returns to be loaded into V_Cache when that is a read miss. If a TLB miss occurs, V_TLB signals a TLB miss trap to the Processor. When a TLB miss occurs, the Processor switches to handle the TLB miss, and reattempts.

When the transaction for data coherency or main memory update is requested, bus signals to all other V_Snoops. V_Snoop lets the given physical address to be translated to virtual address via IMT. All synonyms in V_Cache under the different virtual addresses can be searched. Bus snooping and address inverse mapping are enabled in V_Cache. This ensures that data in the local V_Cache is updated or invalidated when that data is written to by another V_Cache. If a snoop hit occurs after receiving the input from bus, Snooper invalidates or updates the associted entry of V_Cache. But Snooper discards the input if a snoop miss occurs.

4. MODELING

4.1 Broadcast Model

All the processing elements in this multiprocessor system have the same behavior characteristics in an abstract level. They can be treated as the isomorphic copies of an init-cell of the kernel model[Kim 90]. The broadcast model is suitable to model this processing element set. Each members of a broadcast model is a processing element described in Section 4.2. All members of a broadcast coupled model communicate directly with each other and with the outside world. The prob-in ports of all members are coupled together and coupled to the broadcast model's input port prob-in. The same is true for the other input and output ports. Refer to [Dai 91] for details.

4.2 The hierarchical coupling

Simulation models are built up hierarchically. The coupled model of a shared bus multiprocessor system consists of three components: PES, INTERNET, and MM. PES and INTERNET are coupled models while MM is an atomic model. Figure 4.1 is the overall coupling scheme for the shared bus multiprocessor system.

PES is a broadcast coupled model. Each PE is constructed, shown in Figure 4.2, by coupling of three coupled models: Processor, CM_I and CM_D, and one atomic model Snooper. Processor, shown in Figure 4.3, has three atomic models: Dispatcher, PROC_D, and PROC_I. The same models of CM_I and CM_D are used for both virtual cache and physical cache under the different names: VCM_I(VCM_D) and PCM_I(PCM_D). VCM_I(VCM_D), shown in Figure 4.4, consists of two atomic models: VCache_I(VCache_D) and VTLB_I(VTLB_D). PCM_I(PCM_D), shown in Figure 4.5, compries of two atomic models of PCache_I(PCache_D) and PTLB_I(PTLB_D).

A multiprocessor with virtual cache can be similarly modeled. Instead of the atomic model Snooper for a multiprocessor with physical cache, V_Snoop for one with virtual cache is a coupled model consisting of two atomic model: IMT and Snooper. The INTERNET has the broadcast model Buffers and the atomic model G_Bus.

4.3 Atomic model development

As discussed in Section 2, the behavioral specification includes definitions of the external transition function, the internal transition function, and the output function for atomic models.

Figure 4.6 and Figure 4.7 show atomic models specification of V_Cache and P_Cache, respectively. Let us briefly explain the Figures. After receiving an input on port "ref_in", V_Cache goes into one of three phases, CACHE_HIT, CACHE_WT, and CACHE_MISS, under the condition of input. This condition is created by a random variables under the given rate of cache hit. If a read is issued and a hit occurs, then cache goes into the phase CACHE_HIT, and returns to a passive state after holding for *comp_read_time* time unit. At the same time the ouput is generated on port "cache_done", which is sent to Processor. If it is issued a write and a hit occurs, then cache goes into the phase CACHE_WT, and transits to phase WAITING_WT after holding for *comp_write_time* time unit. At the same time the output is generated on port "bus_req_wt". This is done to update the corresponding location of main memory under the "Write-Through" policy. At the phase WAITING_WT, it waits for a bus grant on port "bus_connect". When receiving the input on port "bus_connect" from INTERNET, it goes into the phase MM_SERVICE. After holding for *mm_service_time* time unit, V_Cache generated the output on port "cache_done" to be sent to Processor, and returns to a passive state. If a cache miss occurs in a read and a write operations, then cache goes into CACHE_MISS, and transits to the phase WAITING after holding for *comp_time* time unit. At the same time the output is generated on port "bus_req" and sent to the input port "in" of V_TLB. At the phase WAITING, it waits until a bus grant on port "bus_connect". It then goes into phase MM_SERVICE on receiving a bus grant, and stays for *mm_service_time* time unit. It transits to the phase PASSIVE. At the same time it put the output on port "cache_done", which is sent to Processor.

The operation sequence of a write hit with the write through policy is very similar to that of the case above. The only difference is that the INTERNET must broadcast the signal to all inputs of Snoopers if a write hit with the write through policy is accepted, so that Snoopers can search their

entries shared. If the input is received on port "inv" from Snooper, V_Cache goes into phase INVALIDATE. After holding for *inv_time* time unit, it goes into a passive state.

The phase transition diagram for V_Cache atomic model is shown in Figure 4.8.

The atomic models of P_Cache is similarly specified. The input on the port "ref_in" of P_Cache is sent from the ouput port "out" of P_TLB, instead of Processor on V_Cache. The outputs on ports "bus_req" and "bus_req_wt" send directly to INTERNET, instead of through V_TLB on V_Cache. The output on port "tlb_miss" or "cache_done" sends to the input port of INTERNET or Processor, which is the same destination for V_TLB or V_Cache.

5. CONCLUDING REMARKS

A methodology for performance modeling of a shared bus, a shared memory multiprocessor system with physical and virtual cache has been presented. We employed Zeigler's DEVS (Discrete Event Systems Specification) formalism for modeling such multiprocessor systems in a hierarchical, modular manner. The systems considered in this paper were tighly coupled multiprocessor systems with shared buses and shared memories. Several assumptions have been made to simplify modeling.

These simulation models can be efficiently extended to many other cases of performance analysis: having copy-back or other policies for updating main memory, having any special protocols for cache data coherency, having multilayered cache memory structures, and so forth. The methodology employed here is a hierarchical modular one, thereby being able to easily adding models as well as replacing one by the other without affecting other model components.

Future research will employ actual simulation of the developed models using DEVS based simulation environments for DEVS-Scheme or DEVSIM++. Such simulation will provide the designer of multiprocessor systems with important design parameters such as virtual or physical cache, bus traffic dynamics, and so forth.

REFERENCES

[Cho 86] J. Cho, A. J. Smiths, and H. Sachs, *The Memory Architecture and the Cache and Memory Management Unit for the Fairchild CLIPPER Processor,* Technical Report No. UCB/CSD 86/289, EECS, UC Berkeley, April 1986.

[Dai 91] B. Dai, *A Hierarchical, Modular Methodology for Multiprocessor System Modelling and Simulation,* MS Thesis, Dept of Electrical and Computer Engineering, The Univ of Kansas, 1991.

[Goodman 87] J. R. Goodman, "Coherency for Multiprocessor Virtual Address Caches," *Proc. 2nd Int. Conf. on Architectural Support for Programming and Operating Systems(ASPLOS-II),* Oct. 1987, pp. 72-81.

[Hawng 84] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing,* McGraw Hill Book Company, 1984, pp. 111-112.

[Hill 85] M. D. Hill, et. al., "SPUR: A VLSI multiprocessor workstation," *IEEE Computer,* Nov. 1985.

[Hill 86] M. D. Hill, S. Eggers, et. al., "Design Decisions in SPUR," *IEEE Computer,* Nov. 1986, pp.8-22.

[Katz 84] R. H. Katz, S. J. Eggers, D.A. Wood, C. L. Perkins, and R. G. Sheldon, *Implementing a Cache Consistency Protocol,* Technical Report No. UCB/CSD 84/207, EECS, UC Berkeley, Oct. 1984.

[Kim 90] T. G. Kim and B. P. Zeigler, "The DEVS-Scheme Simulation and Modelling Environment," *Knowledge Based Simulation: Methodology and Applications*, Springer Verlag, New York, NY, 1990.

[Kim 92] T. G. Kim and S. B. Park, "The DEVS Formalism: Hierarchical Modular Systems Specification in C++", *Proc. in ESM'92*, York, UK, June 1992.

[Lee 88] K. H. Lee, "A study of the Data Coherency for HARP(High-performance Architecture for the Risc Processor)," *The ETRI Journal,* Vol. 10, No. 3, Sep. 1988, pp. 62-72(in Korean).

[Smith 82] A. J. Smith, "Cache Memories," *Computing Surveys,* Vol. 14, No. 3, Sep. 1982, pp. 473-530.

[Zeigler 84] B. P. Zeigler, *Multifacetted Modeling and Discrete Event Simulation,* Orlando, FL, Academic Press, 1984.

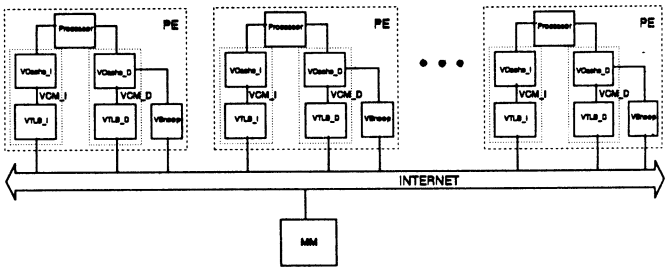

Figure 3.1. The multiprocessor system with virtual cache

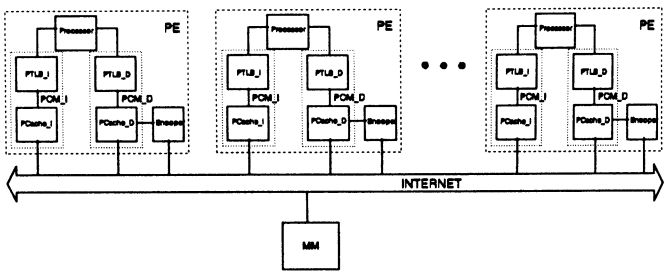

Figure 3.2. The multiprocessor system with physical cache

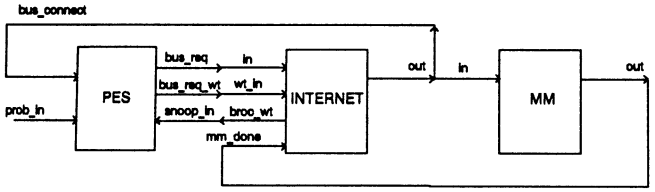

Figure 4.1. Overall system model

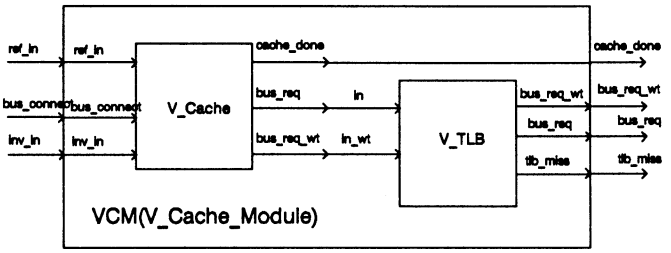

Figure 4.4. The coupled model of VCM

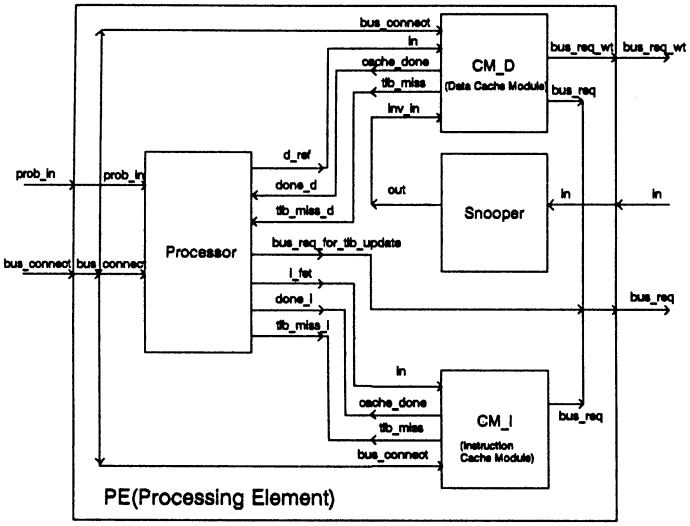

Figure 4.2. The coupled model of PE

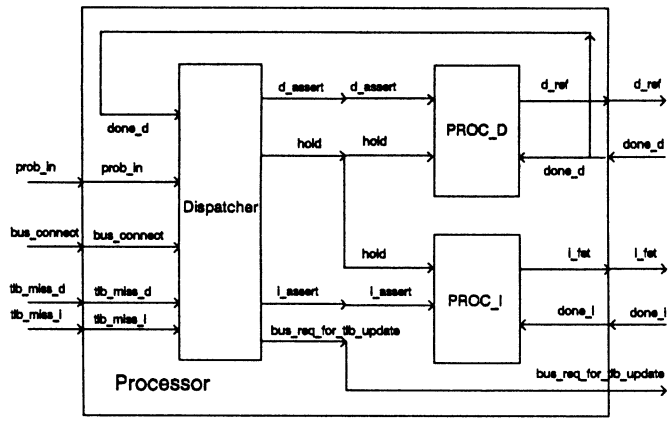

Figure 4.3. The coupled model of Processor

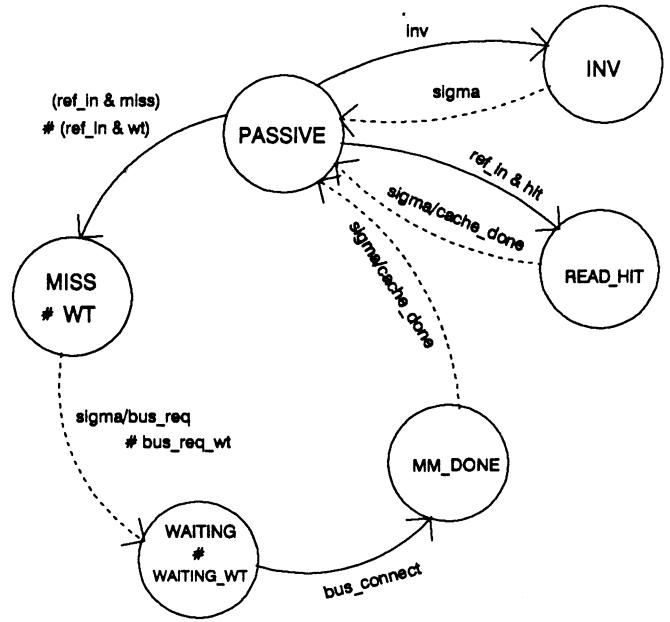

Figure 4.5. The coupled model of PCM

Figure 4.8. The phase transition diagram of V_Cache

```
/*   Input port: in, bus_connect, invalidate
     Output port: bus_req, cache_done, bus_req_wt
     State variables: sigma, phase */

     /* External Transition Function */
     ----------------------------
     when receive input x at port in
          case of:
          "hit & read"   sigma := comp_read_time
                         phase := CACHE_HIT
          "hit & write"  sigma := comp_write_time
                         phase := CACHE_WT
          "miss"         sigma := comp_time
                         phase := CACHE_MISS
     end when

     when receive x at port bus_connect
          if proc_id := proc_id(prob) then
               if phase := WAITING then
                    sigma := common_time
                    phase := CM_SERVICE
               else  if phase := WAITING_WT then
                          sigma := common_wt_time
                          phase := CM_SERVICE_WT
                     else
                          continue
                     endif
               endif
          else
               continue
          endif
     end when

     when receive x at port inv
          sigma := inv_time
          goto phase INVALIDATE
     end when

     /* Internal Transition Function */
     ----------------------------
     case phase of:
     CACHE_HIT      hold(sigma)
                    phase := passivate
     CACHE_WT       hold(sigma)
                    sigma := inf
                    phase := WAITING_WT
     CACHE_MISS     hold(sigma)
                    sigma := inf
                    phase := WAITING
     WAITING_WT     hold(sigma)
                    phase := passivate
     WAITING        hold(sigma)
                    phase := passivate
     CM_SERVICE_WT  hold(sigma)
                    phase := passivate
     CM_SERVICE     hold(sigma)
                    phase := passivate
     INVALIDATE     hold(sigma)
                    phase := passivate

      /* Output Function */
      ---------------
     case of:
     phase := CACHE_HIT
          send output to port cache_done
     phase := CACHE_WT
          send output to port bus_req_wt
     phase := CACHE_MISS
          send output to port bus_req
     phase := CM_SERVICE_WT
          send output to port cache_done
     phase := CM_SERVICE
          send output to port cache_done
```

Figure 4.6. DEVS Pseudo-code for V_Cache

```
/* Input port: in, bus_connect, invalidate
   Output port: bus_req, cache_done
   State variables: sigma, phase */

   /* External Transition Function */
   ----------------------------
   when receive input at port in
        case of:
        "hit & read"   sigma := comp_read_time
                       phase := CACHE_HIT
        "hit & write"  sigma := comp_write_time
                       phase := CACHE_WT
        "miss"         sigma := comp_time
                       phase := CACHE_MISS
   end when

   when receive x at port bus_connect
        if proc_id := proc_id(prob) then
             if phase := WAITING then
                  sigma := common_time
                  phase := CM_SERVICE
             else
                  continue
             endif
        else
             continue
        endif
   end when

   when receive x at port inv
        sigma := search_time
        goto phase TAG_SEARCH
   end when

   /* Internal Transition Function*/
   ----------------------------
   case phase of:
   CACHE_HIT      hold(sigma)
                  phase := passivate
   CACHE_WT       hold(sigma)
                  sigma := inf
                  phase := WAITING
   CACHE_MISS     hold(sigma)
                  sigma := inf
                  phase := WAITING
   WAITING        hold(sigma)
                  phase := passivate
   CM_SERVICE     hold(sigma)
                  phase := passivate
   TAG_SEARCH     hold(sigma)
                  if "hit" then
                       sigma := inv_time
                       goto phase INVALIDATE
                  else  if "miss" then
                             goto phase PASSIVE
                        else
                             continue
                        endif
                  endif
   INVALIDATE     hold(sigma)
                  passivate

    /* Output Function*/
    ---------------
   case of:
   phase := CACHE_HIT
        send output to port cache_done
   phase := CACHE_WT
        send output to port bus_req_wt
   phase := CACHE_MISS
        send output to port bus_req
   phase := CM_SERVICE
        send output to port cache_done
```

Figure 4.7. DEVS Pseudo-code for P_Cache