Comparison of SpecSyn and Workbench Modeling

Chu-Yi Huang

Technical Report# 94-59
November 23, 1994

Dept. of Information and Computer Science
University of California, Irvine
Irvine, CA 92717-3425
Phone: (714)824-8059
FAX: (714)824-4056
Email: chu-yih@ics.uci.edu

Abstract

SpecSyn is a system modeling and synthesis tool designed in University of California, Irvine. Workbench is a commercial system modeling tool of Scientific Engineering Software. SpecSyn is based on Program-State-Machine (PSM) model while Workbench is based on Queuing Network model. In this report, we compare these two models by the feature of modeling a system and the application of these two tools. Two examples, answering machine controller and flash file system, are given to demonstrate the difference of modeling systems using SpecSyn and Workbench. The comparison shows that SpecSyn has more good features and is more suitable in the whole design process than Workbench. Improvements for the weakness of SpecSyn, such as pipeline behavior and queuing mechanism, are proposed to make SpecSyn applicable to wider variety of applications.

Contents

1 Model of SpecSyn and Workbench

SpecSyn is a system modeling and synthesis tool based on Program-State-Machine (PSM) model [1] while Workbench is a system modeling tool based on queuing network model [4] [5]. Workbench is a product of Scientific Engineering Software. In this report, we are going to compare PSM and queuing network model by using these two tools as examples.

In SpecSyn, user models system behavior in SpecChart language, a front end for VHDL. After the behavior was correctly described, user can select different implementation configurations and get a refined behavior description and some design quality estimations. In Workbench, user models a system implementation in queuing network and C programming language. The output of Workbench is input dependent statistic performance information to help user locate system's bottleneck. Following are brief introduction to the model, input, output, and user interface of these two tools.

1.1 Model

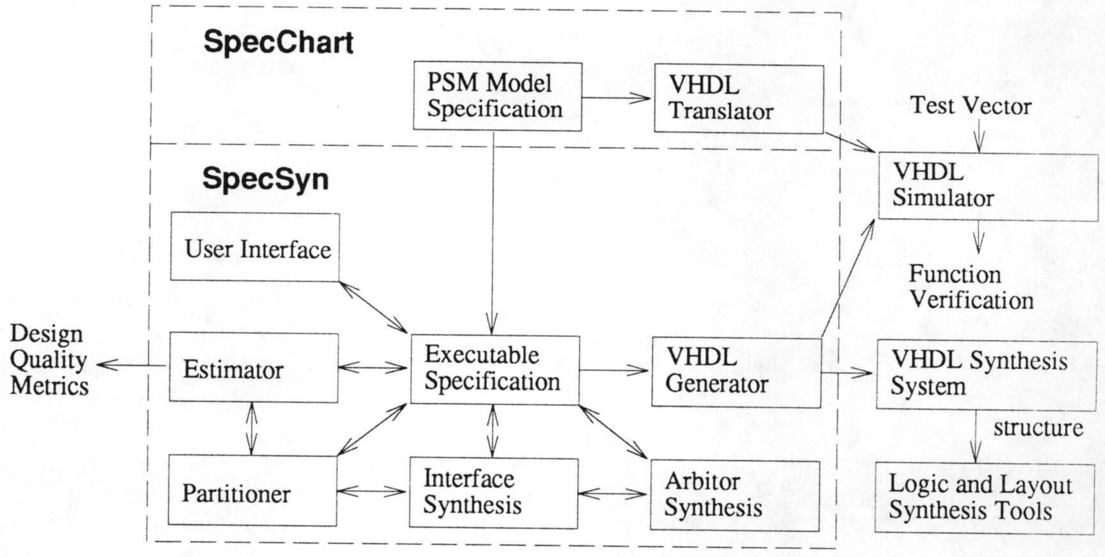


Fig. 1.1: SpecSyn Design Environment.

Fig. 1.1 is the SpecSyn design environment[2]. SpecChart is a PSM based modeling language. PSM model basically consists of a hierarchy of program-states, in which each program-state represents a distinct mode of computation. Each program-state is either a composite program-state or a leaf program-state. A composite program state can be further decomposed into either concurrent or sequential program-substates. If they are concurrent, all program-substates will be active if the program-state is active, whereas if they are sequential, only one program-substate will be active at a time when the program-state is active. There are two types of transition arcs connecting program states: transition-on-completion (TOC) arc and transition-immediately (TI) arc. TOC arcs traverse only when the source program-state complete its computation and the associated arc condition is true, but TI arcs traverse immediately when the arc condition becomes true. In SpecChart, the computation of leaf program-state was expressed in VHDL code.

Fig. 1.2 is a SpecChart specification example. **X1**, **X2**, **Y** and **Z** are leaf program-state. **X1** and **X2** are sequential program-substates of **X**. **X**, **Y** and **Z** are concurrent program-substates of **B**. Fig. 1.3 is the equivalent graphic representation of Fig. 1.2. Concurrent program-states are separated by a dotted line. A bold inverted triangle indicates the first sequential program-substate. A bold square represents a completion point. Arcs originating from a bold square are TOC arcs, for example the **e2** and **e3** arc. On the other hand, arcs originating from the perimeter of the source program-state are TI arcs, for example the **e1** arc.

PSM is an event-driven computation. Events are first generated by changing input values. These events stimulate active program-states to respond. In processing these events, the set of active program-states may change and new events are generated which may stimulate the new set of active program-states. This stimuli-response process keeps going until no active program-state (computation complete) exists.

Besides modeling behavior, SpecSyn help user decide implementation configuration by refining system behavior and estimating design quality.

```
entity E is
     port (P: in integer, Q: out integer);
end E
architecture A of E is
begin
          behavior B type concurrent subbehaviors is
               type int_array is array (natural range<>) of integer;
               signal M: int_array(15 downto 0);
          begin
               X: (TOC, true, complete);
               Y: (TOC, e3, complete);
               Z: ;
               behavior X type sequential subbehaviors is
               begin
                    X1: (T1, e1, X2);
                    X2: (TOC, e2, complete);
                    behavior X1 type code is .....
                    behavior X2 type code is .....
               end
               behavior Y type code is
                    variable max:integer;
               begin
                    max:=0;
                    for J in 0 to 15 loop
                         if (M(J)¿max) then
                              max:=M(J);
                         end if;
                    end loop;
               end Y;
               behavior Z type code is .....
          end B;
end A;
```

Fig. 1.2: A sample SpecChart specification.

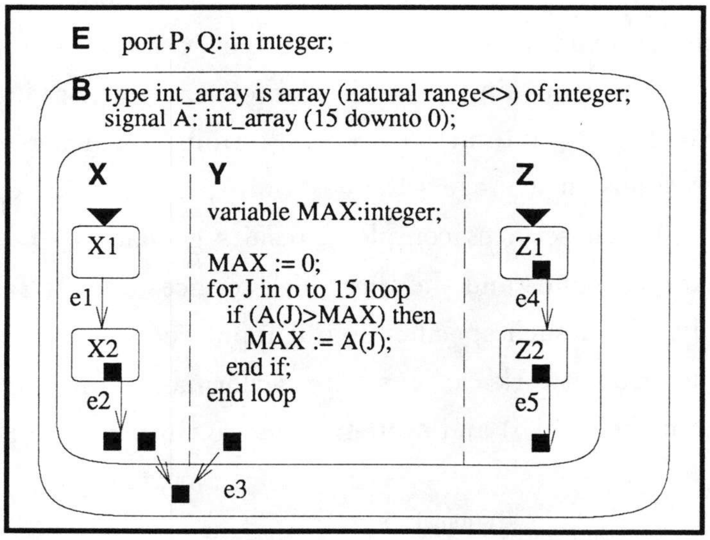


Fig. 1.3: Graphic Representation of SpecChart.

After user allocates hardware resources and specifies performance/cost constraints, SpecSyn partitions and binds behavior to meet the constraints as much as possible. When given an allocation, partition and binding, SpecSyn refines behavior to maintain correct functionality. Variables partitioned among memories require memory address translation. Behaviors separated among components must be modified to maintain correct communication. Channels mapped to buses require interface synthesis to determine communication protocols, and arbiter synthesis to resolve any simultaneous bus requests. SpecSyn generates VHDL specification for these added function. The partition/binding and refine process are guided by estimation of design quality metrics, such as area, performance, and pin number. The estimation are based on library information. The performance estimation are input independent, no input data profile are needed.

Queuing network represents a system as a network of queues and servers. Arriving requests are stored in queue while waiting for processing by servers. Edges, connecting queues and servers in network, can associate a condition to divert input requests. Queuing network was originally and typically used for performance analysis. In this case, user only specified the arriving rate of input request and process time of each server. The modeling tool calculated the system throughput by using queuing theory or simulation.

Fig. 1.4(a) is a queuing network characterizing a computer system [3]. There are one to fifty interactive users on terminals which generate requests every 30 seconds in average. Requests are processed by CPU and disks iteratively. When services complete, results go back to terminals. CPU service takes 3 seconds and the three disk services take 1, 2, and 4 seconds respectively. With such specification of input request and server's service time, a queuing network can generate performance analysis such as CPU utilization in Fig. 1.4(b) and system response time in Fig. 1.4(c).

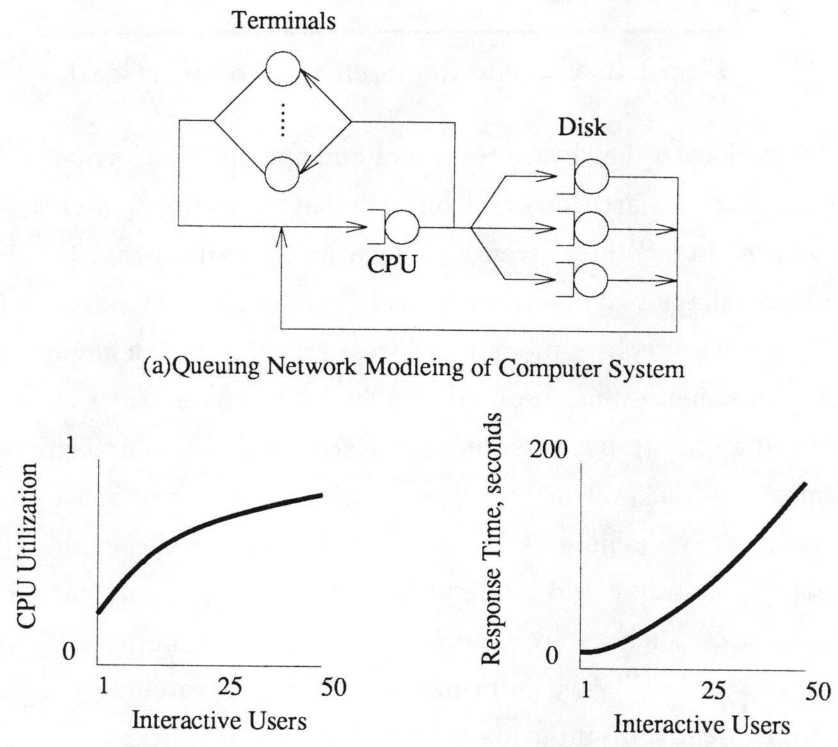


Fig. 1.4: Queuing Network Model.

In order to enhance queuing network for modeling functionality, Workbench associates each server with a function, such as delay and lock resource, and provides different kinds of queue such as first-in-first-out and last-in-first-out. Workbench implements a stream-flow computation. **Generation** nodes which are the start points of computation generate input requests. When a input request flows into a server, the function associated with this server is executed. After processing, the request flows along network edges to next server. The request keep flowing in the network until it flow into a **Termination** node which terminates input requests. All servers processing input requests are active at that instance. The computation completes when no more input requests are generated.

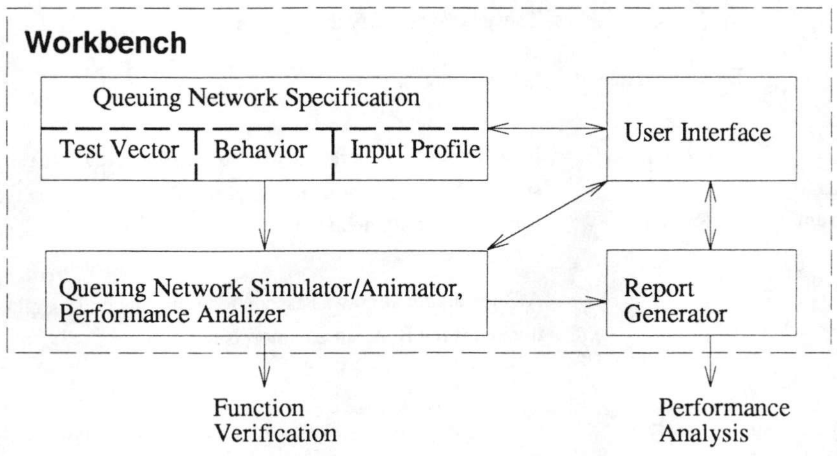
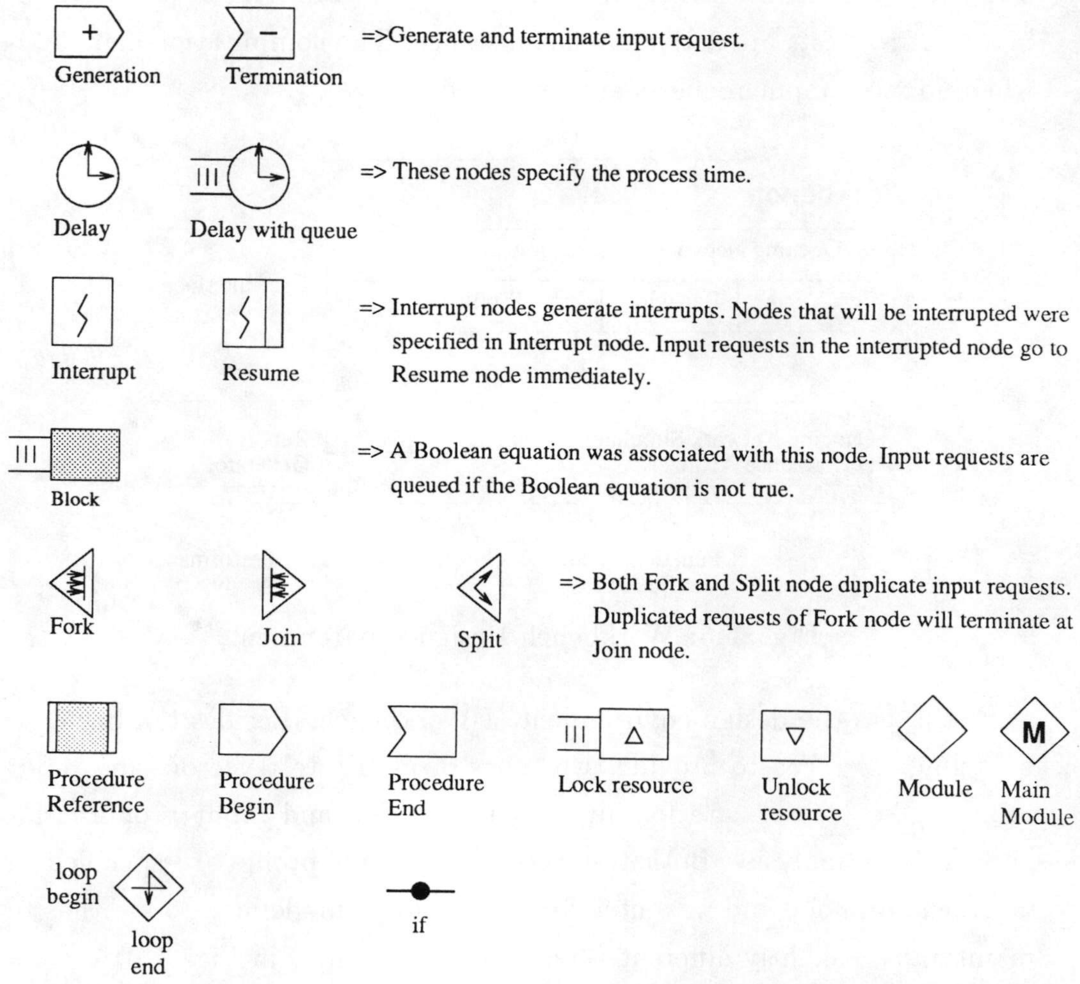


Fig. 1.5: Workbench Design Environment.

Fig. 1.5 is the design environment of Workbench. Besides the behavior modeling, user has to provide two types of input: test vectors and input profile. Test vectors are for function verification and input profile is for performance analysis. Both test vector and input profile are specified in **Generation** node and was integrated in behavior modeling. So user has to maintain two slightly different version, as the example in Fig. 3.10.

In Workbench, queue and server are combined and represented as node. Some node types represent a queue attached in front of a server while the others are servers only. Workbench dose not provide independent queue that is a queue which dose not associate with any server. Beside the built-in functionality, user can associate each node with a C program to extend that node's function. Fig. 1.6 shows Workbench's major node types.



=>Generate and terminate input request.

=> These nodes specify the process time.

=> Interrupt nodes generate interrupts. Nodes that will be interrupted were specified in Interrupt node. Input requests in the interrupted node go to Resume node immediately.

=> A Boolean equation was associated with this node. Input requests are queued if the Boolean equation is not true.

=> Both Fork and Split node duplicate input requests. Duplicated requests of Fork node will terminate at Join node.

Fig. 1.6: Workbench's Node Types.

1.2 Input

SpecChart language is an augmented VHDL program as shown in Fig. 1.2. Besides standard VHDL statement, user can use several augmented statements to define and describe program states and their relationships. The input of SpecSyn, besides the system behavior, is system level hardware components and performance/cost constraints for the proposed system implementation. User selects hardware components such as memory, CPU and ASIC of different area, pin number, monetary cost, performance etc. from library. The input of Workbench are nodes connected by edges that are all represented in graphics as shown in Fig. 1.5. Attributes of nodes and condition of edges are expressed in text.

1.3 Output

There are two outputs of SpecSyn: a refined VHDL program and design quality estimation. The refined VHDL program composed of the VHDL translation of SpecChart modeling, the component interface specification, and partition/binding information. The design quality metrics include performance, area, monetary cost, bus width and pin number.

The output of Workbench is the statistic performance data. Statistic performance data means the maximal value, minimal value, sample count, standard variance etc. of some performance metrics such as execution time and queue waiting time. They were calculated by simulation and categorized by different input types. Workbench can produce maximal value, average value, minimal value, and standard variance of queue length, queue waiting time, execution time for different execution path, inter arrival time and resource/node utilization. Fig. 1.7 is an example of Workbench output.

1.4 User Interface

SpecChart supports two user interfaces. User can key in the whole description in a text editor or use a simple graphic user interface to capture

```
Utilization:
     Resource CPU::   min:0.01    ave:0.08     max:0.13
                      variance:0.1             count: 150
                      category: read

                      min:0.02    ave:0.14     max:0.23
                      variance:0.12            count: 124
                      category: write

                      min:0.01    ave:0.1      max:0.23
                      variance:0.11            count: 274
                      category: all
```

Fig. 1.7: Example of Workbench output.

and traverse the hierarchy relation and use the integrated text editor to key in transition statements and VHDL codes. SpecSyn provides graphic user interface for entering constraints, selecting hardware components and invoking partition, binding and estimating functions.

Workbench provides a graphic user interface for selecting and placing node and connecting nodes with lines. A simple text editor was integrated for keying node attributes, line conditions, and C codes. Workbench also provides a graphic interface animating input requests flowing in network for verification use.

2 Goal of SpecSyn and Workbench

An ideal system-level design process consists of four steps:

1. model the pure behavior and verify its correctness,
2. select an implementation,
3. evaluate design quality of the selected implementation and check it against user constraints,
4. find the bottleneck if constraints are violated; Then refine the implementation by repeating step 2, 3, 4 and sometimes step 1.

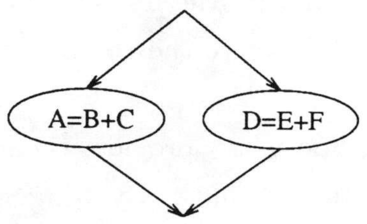
This process can restrict the side effect of modification in each step to minimal. Modification in latter steps did not cause modification in prior steps.

Sometimes user combined step 1 and 2, particular in modeling an existed system. They take the implementation characteristic such as hardware allocation, behavior partition, binding, and scheduling as part of system behavior. If this design dose not meet user requirement, the refinement may change the hardware configuration or behavior partition. This kind of refinement should modify the whole modeling, not only the implementation characteristic. This waste much time and also increase the complexity of modeling.

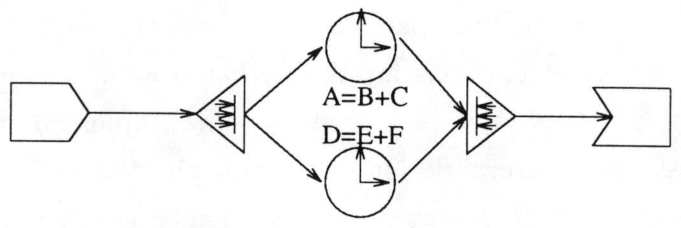
SpecSyn support this ideal design process. SpecChart provides the functionality needed for step 1 and SpecSyn is for step 2, 3 and 4. By using SpecChart, user can easily describe their concept which can be a pure behavior or a behavior bound to some hardware implementation. Besides providing hardware allocation user interface for specifying implementation configuration, SpecSyn generates glue functions for the allocated components to keep the correct function. This releases user from some implementation detail. SpecSyn also estimates several design quality metrics for constraint checking and design refinement. So SpecSyn is ideal for most applications.

Because SpecSyn do not provide input-dependent performance analysis, they can not provide enough useful performance metrics for those systems whose performances are affected significantly by input values. Without those input dependent statistic performance information as provided by queuing network, user can not locate system's bottleneck easily. As in flash file system example, read, write (set 0 to 1), and erase (set 1 to 0) take about 100 nanoseconds, 10 to 20 microseconds and 2 to 3 seconds respectively. Improving erase command did not significantly improve the whole system because less than 10% of total execution time was spent in erase commands due to its rare occurrence. Write commands cost more than 40% of total execution time. In this case, input dependent statistic performance analysis categorized by input types is more helpful for locating system bottlenecks and tuning the system for certain input profile.

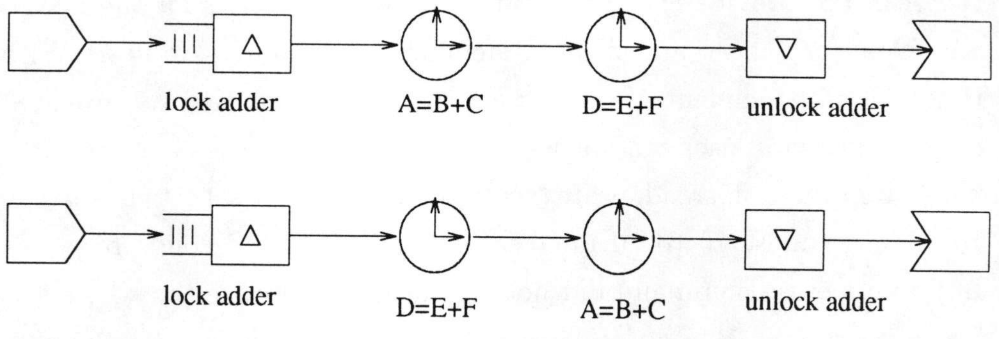
Workbench dose not support this design process. It combined step 1 and 2 into one task. Hardware resources were declared in the same modeling of behavior. Behavior partition and binding was done by user and specified by using resource lock/unlock nodes. So user models not only the system behavior but also implementation characteristics. If the refinement changes the implementation, user has to rewrite the modeling as the example in Fig. 2.1.



(a) original behavior in data-flow-diagram



(b) modeling in Workbench for two adders



(c) Two possible modelings in Workbench for one adder

Fig. 2.1: Modeling modification due to hardware allocation in Workbench.

Performance is the only one design metric that Workbench supports. So user can not check the violation about area, pin count etc.. Workbench analizes the system performance based on the delay of each subbehavior that was provided by user. Because Workbench dose not estimate the performance, Workbench is not ideal for modeling a new system. User can not measure performance of each subbehavior for a new system. Based on user's roughly estimated performance information, Workbench can not provide any precise design metric. The modeling is only for documentation. No further operation can be done. For modeling existed system, the complex performance analysis based on queuing model can provide more useful information for systems whose performance are affected significantly by input profile.

3 Examples

Here we present two modeling examples: answering machine [1] and flash file system. Answering machine was first modeled in SpecChart and then translated into Workbench. Flash file system was first modeled in Workbench and then translated into SpecChart. The goal of modeling answering machine is to create a complete modeling which can be used as a reference of lower level design or as an input for synthesis tool. The goal of modeling flash file system is to locate system's bottleneck. Many behaviors were abstracted because they are performance irrelevant. In this two examples, we want to show the difference in modeling capability of SpecChart and Workbench.

3.1 Answering Machine

Fig. 3.1 is the environment for a telephone answering machine controller available on the market. The **Announcement unit** records and plays a short outgoing announcement. The **Tape unit** records messages on a tape and plays them back. Standard **Line circuitry** answers a call, detects a ring, detects a hangup, decodes a button tone, and produce a beep. The

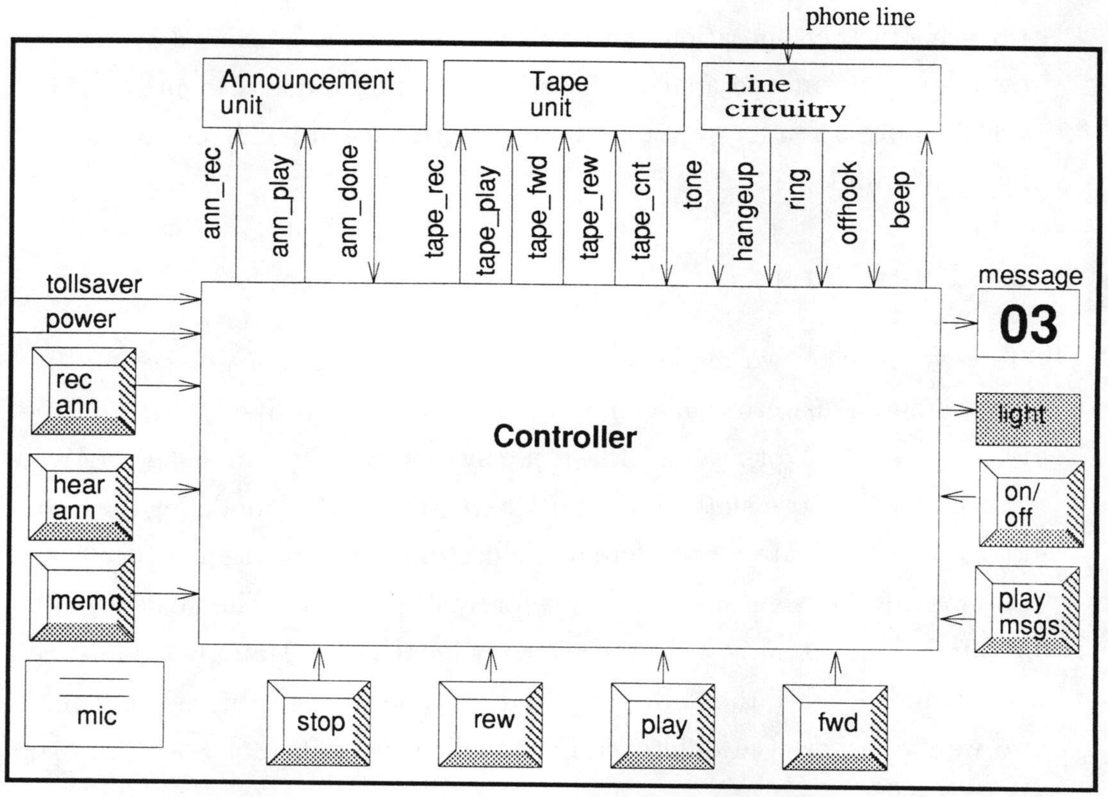


Fig. 3.1: The Answering Machine Controller's Environment.

Display shows the current number of messages and the on/off state of the machine. Nine touch-sensitive buttons allow the machine's user to edit the announcement and hear message. The machine also has two **switches** for switching machine's on/off state and the number of rings (two or four) before recording messages respectively. Fig. 3.2 to Fig. 3.8 shows the modeling of **Controller** in SpecChart and Workbench.

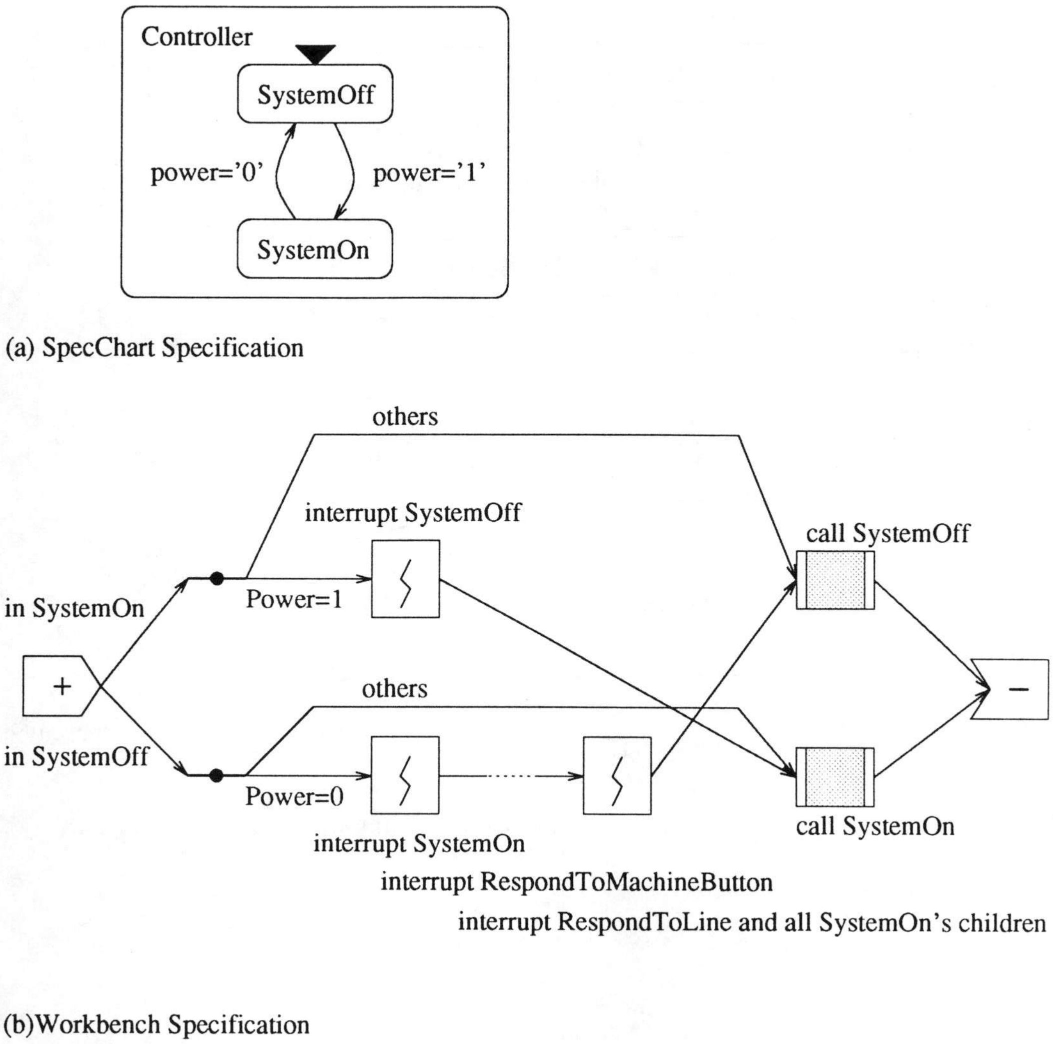


Fig. 3.2: Highest-level View of the Controller.

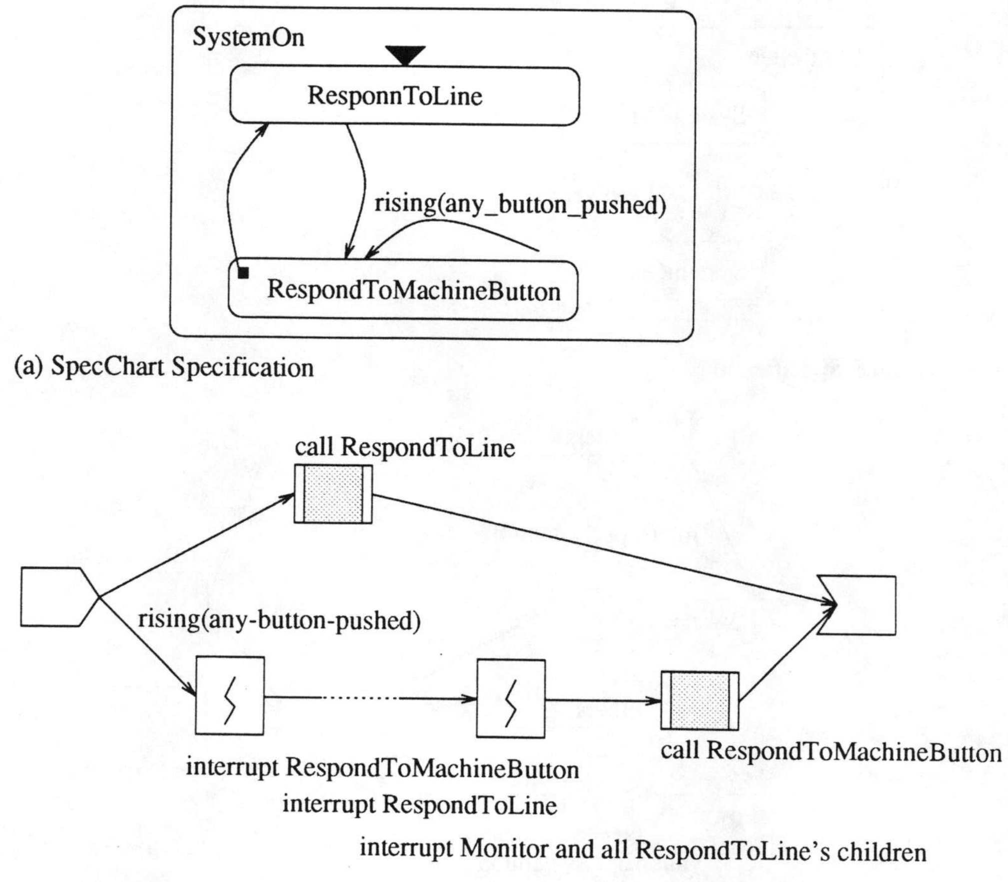
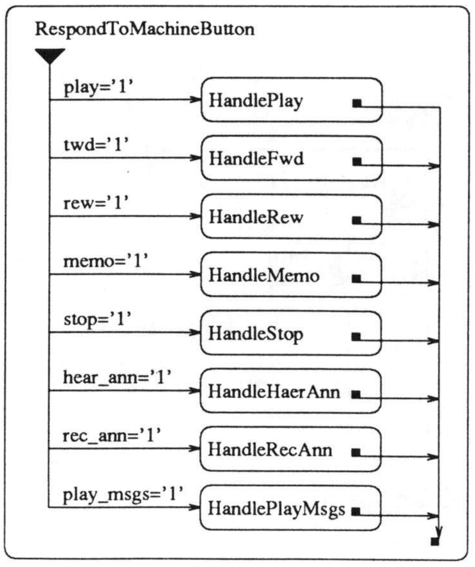
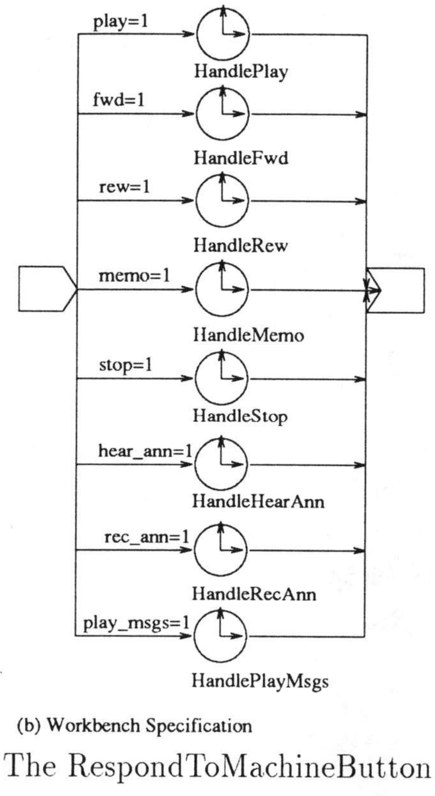

SystemOn
ResponnToLine
rising(any_button_pushed)
RespondToMachineButton
(a) SpecChart Specification
call RespondToLine
rising(any-button-pushed)
interrupt RespondToMachineButton
interrupt RespondToLine
interrupt Monitor and all RespondToLine's children
call RespondToMachineButton
(b)Workbench Specification

Fig. 3.3: The SystemOn Behavior.

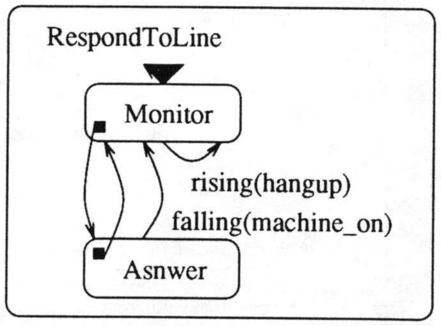


(a) SpecChart Specification

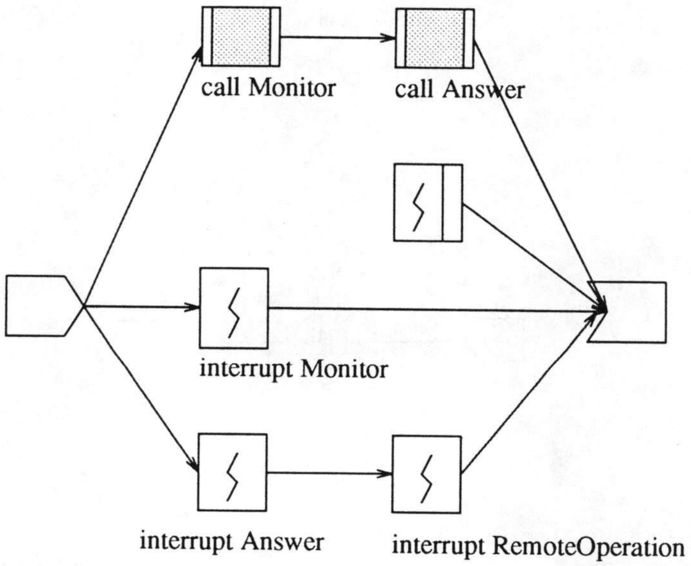


(b) Workbench Specification

Fig. 3.4: The RespondToMachineButton Behavior.

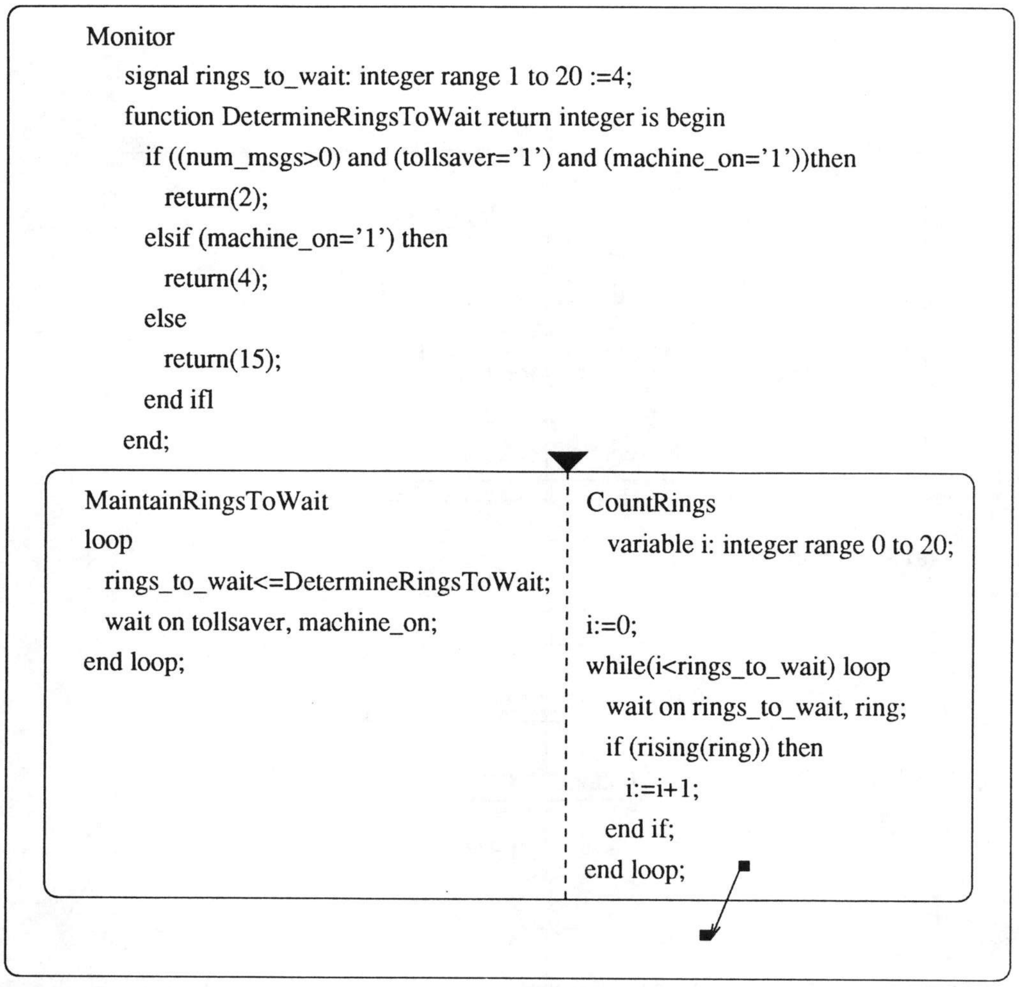


(a) SpecChart Specification



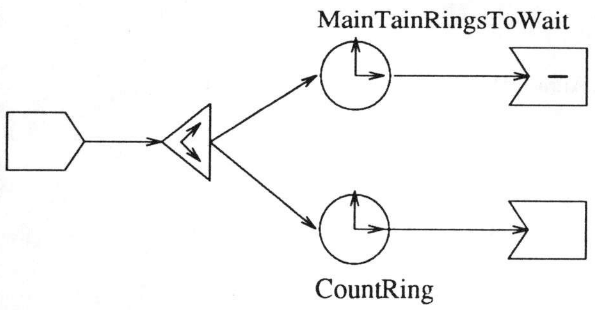
(b) Workbench Specification

Fig. 3.5: The RespondToLine Behavior.



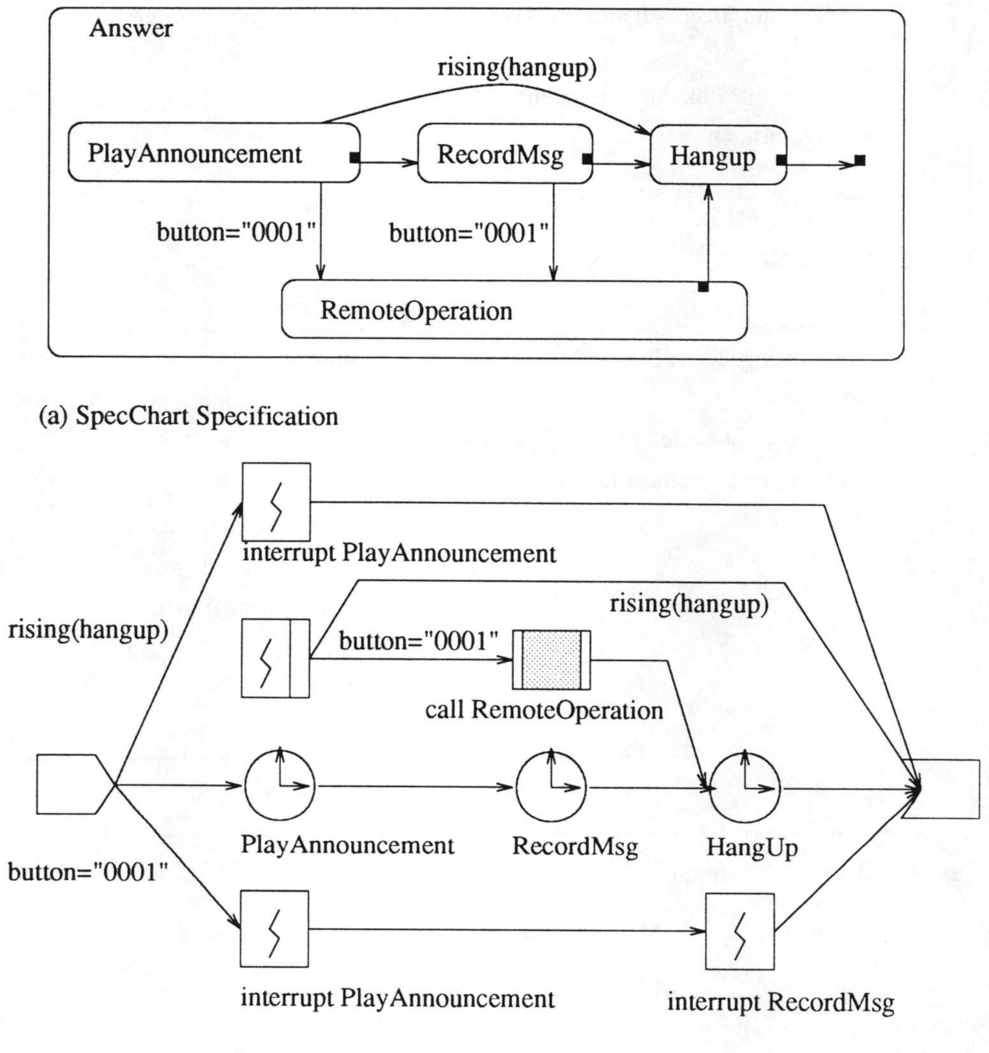
```
Monitor
    signal rings_to_wait: integer range 1 to 20 :=4;
    function DetermineRingsToWait return integer is begin
      if ((num_msgs>0) and (tollsaver='1') and (machine_on='1'))then
        return(2);
      elsif (machine_on='1') then
        return(4);
      else
        return(15);
      end ifl
    end;

MaintainRingsToWait
loop
  rings_to_wait<=DetermineRingsToWait;
  wait on tollsaver, machine_on;
end loop;

CountRings
  variable i: integer range 0 to 20;

i:=0;
while(i<rings_to_wait) loop
  wait on rings_to_wait, ring;
  if (rising(ring)) then
    i:=i+1;
  end if;
end loop;
```

(a) SpecChart Specification



(b) Workbench Specification

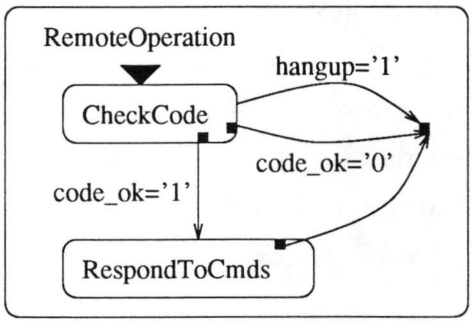
Fig. 3.6: The Monitor Behavior.



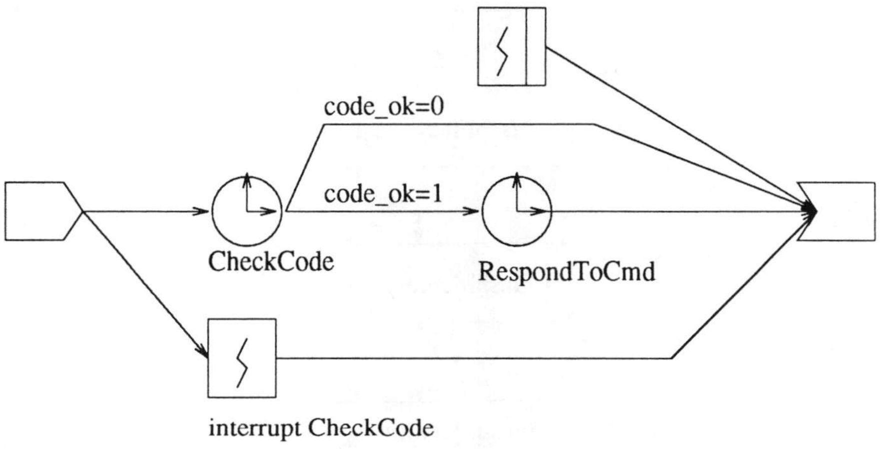
(a) SpecChart Specification

(b) Workbench Specification

Fig. 3.7: The Answer Behavior.



(a) SpecChart Specification



(b) Workbench Specification

Fig. 3.8: The RemoteOperation Behavior.

3.2 Flash File System

Flash File System (FFS) is a file system implemented on flash memory, not on traditional harddisk. Flash memory is a non-volatile memory that it can keep data even when power was disconnected. There are three basic operations for flash memory: read, write and erase. Read operation reads one byte/word. Write operation bitwisely changes value from 0 to 1 of one byte/word. Erase operation bitwisely changes value from 1 to 0 of a whole sector which may be 16K, 64K or some other value according to different products. Read, write, and erase operation take about 100 nanoseconds, 10 microseconds and 2 seconds respectively. Because erase operation has a very large delay, several features such as concurrent sector erase and erase suspend for read are provided to improve performance.

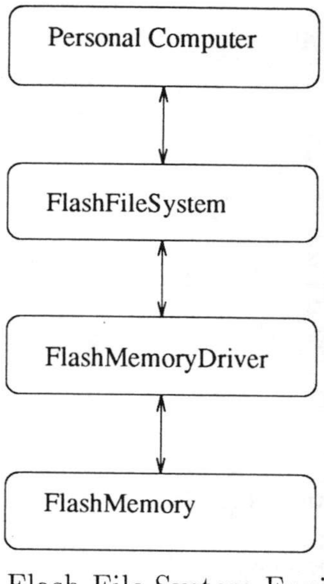


Fig. 3.9: Flash File System Enviornment.

To implement a FFS, file system commands such as read and write a record should be translated into read, write and erase commands for flash memory. In the translation, the special features mentioned above should be utilized according to flash memory status for a better performance. Fig. 3.9 shows the FFS environment. **FlashFileSystem** and **Flash-MemoryDriver** are software running on PC. **FlashFileSystem** generates read/write/erase commands for PC application programs. **FlashMemory-Driver** takes care of the special features of different flash memory products. **FlashMemory** is the modeling of flash memory chip. Fig. 3.10 to Fig. 3.16 shows the Workbench and SpecChart modeling of **FlashMemoryDriver** and **FlashMemory**.

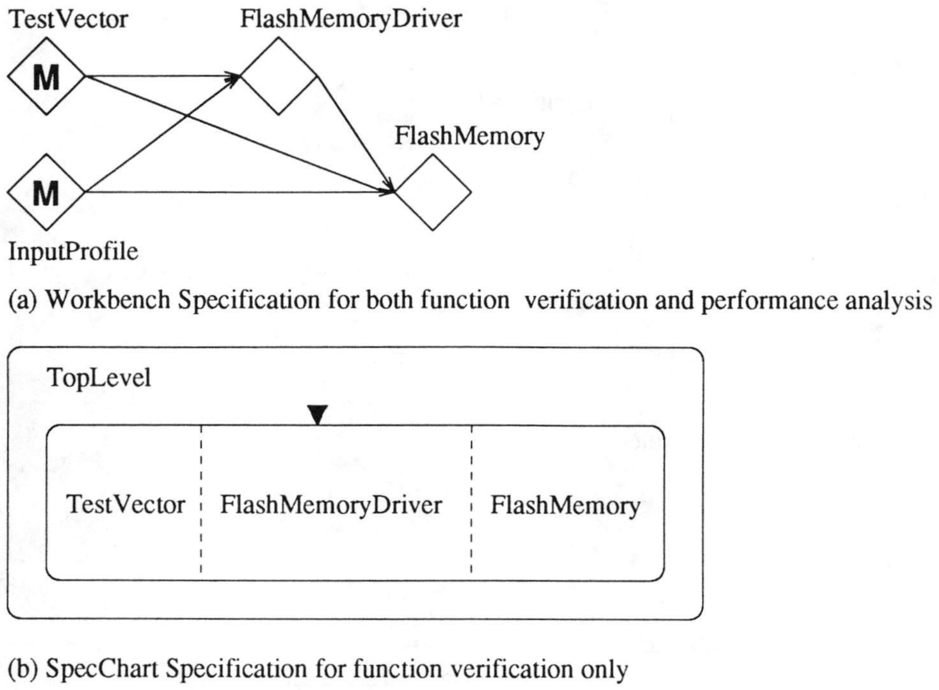
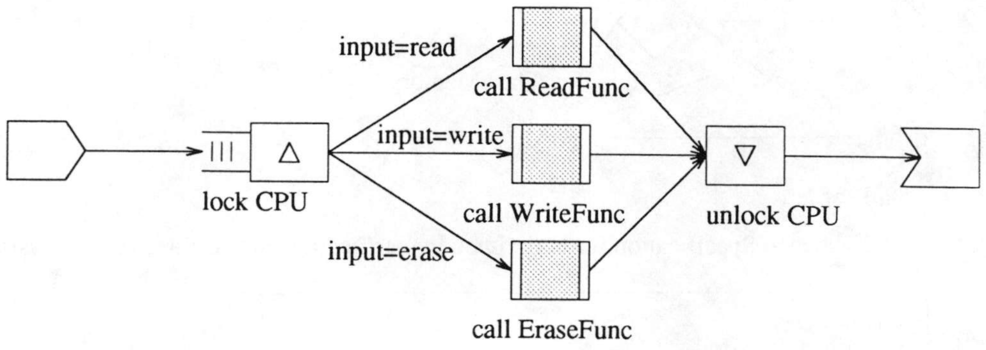
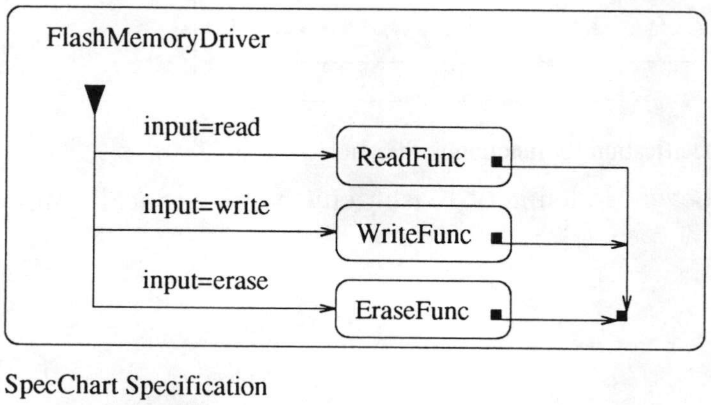

TestVector
FlashMemoryDriver
M
FlashMemory
M
InputProfile
(a) Workbench Specification for both function verification and performance analysis
TopLevel
TestVector
FlashMemoryDriver
FlashMemory
(b) SpecChart Specification for function verification only

Fig. 3.10: Top Level Modeling of FlashMemoryDriver and FlashMemory.

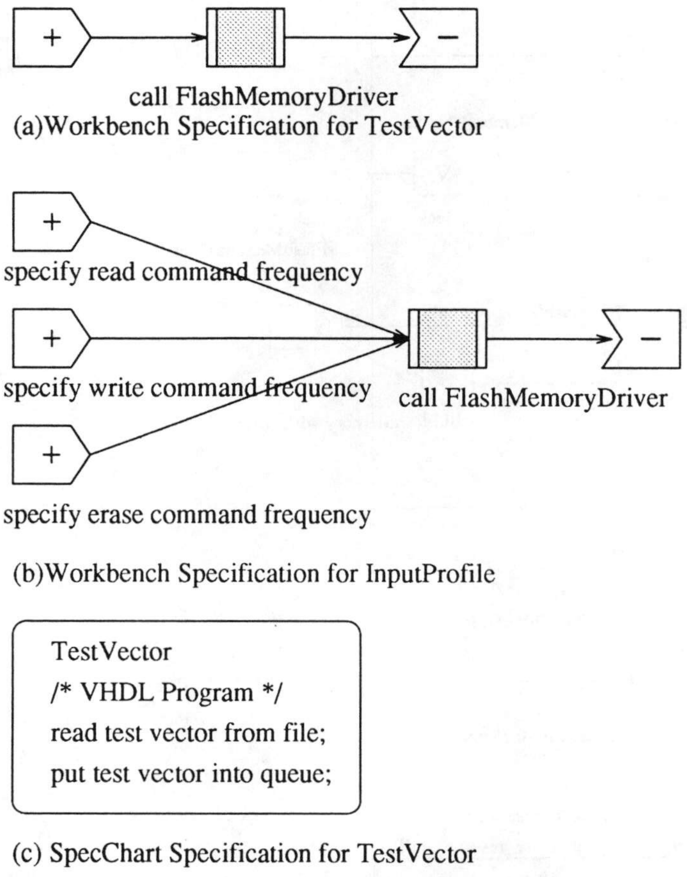


(a) Workbench Specification



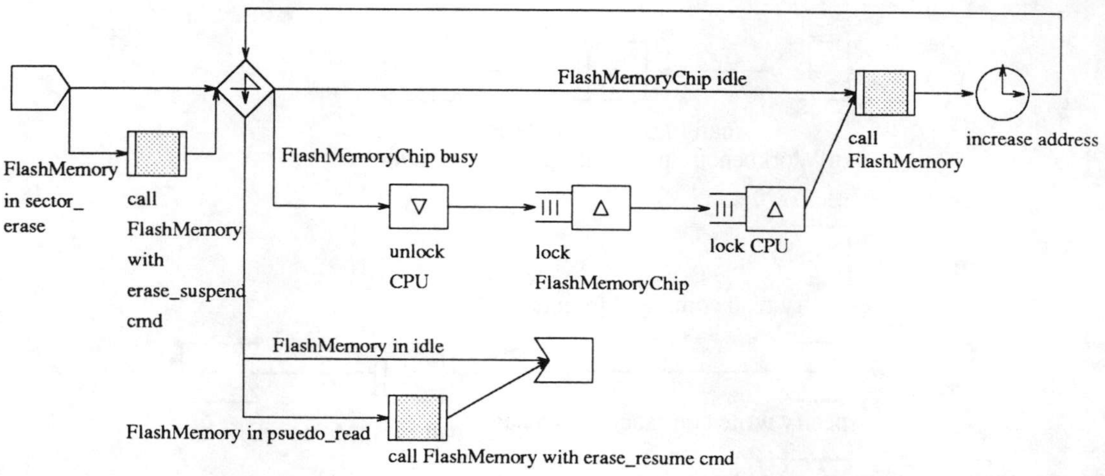
(b) SpecChart Specification

Fig. 3.11: FlashMemoryDriver Behavior.


+
−
call FlashMemoryDriver
(a)Workbench Specification for TestVector
+
specify read command frequency
+
specify write command frequency
+
specify erase command frequency
call FlashMemoryDriver
−
(b)Workbench Specification for InputProfile
(c) SpecChart Specification for TestVector

```
TestVector
/* VHDL Program */
read test vector from file;
put test vector into queue;
```

Fig. 3.12: TestVector and InputProfile Behavior.

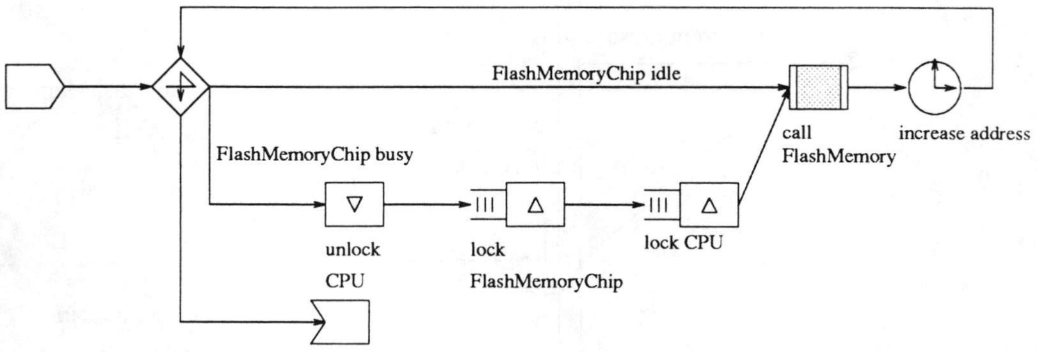


(a) Workbench Specificatiom

```
ReadFunc
/* VHDL Program */
if FlashMemory in sector_erase then
  call FlashMemory with erase_suspend cmd;
loop for every address
  wait until FlashMemoryChip=0;
  FlashMemoryChip=1;
  send write command to FlashMemory;
  increase address;
end loop
if FlashMemory in psuedo_read then
  call FlashMemory with erase_resume cmd;
```

(b) SpecChart Specification

Fig. 3.13: ReadOperation Behavior.

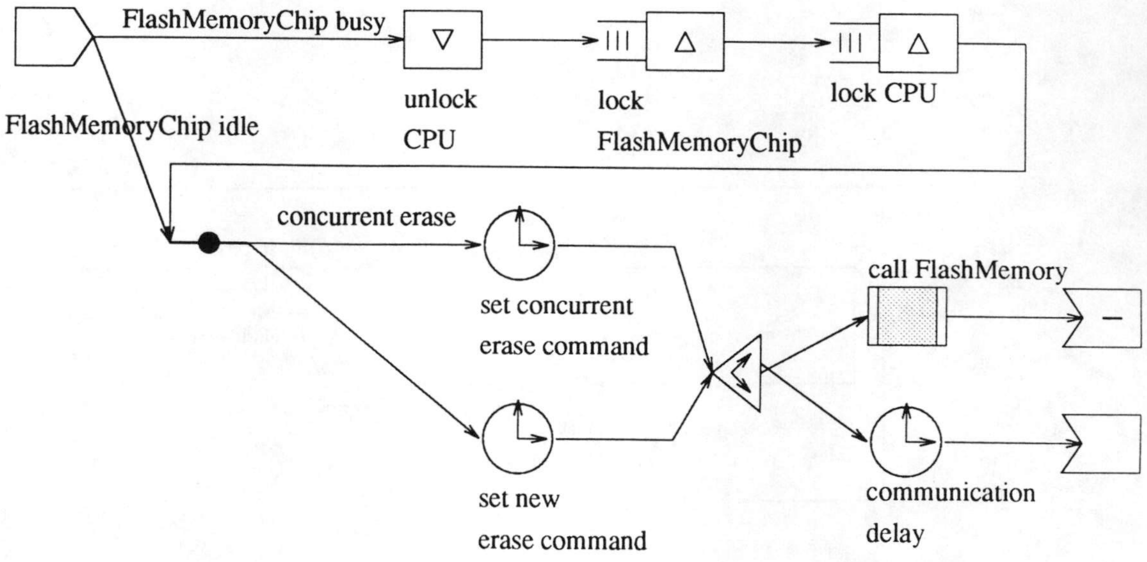


(a) Workbench Specificatiom

```
WriteFunc
/* VHDL Program */

loop for every address
  wait until FlashMemoryChip=0;
  FlashMemoryChip=1;
  send write command to FlashMemory;
  increase address;
end loop
```

(b) SpecChart Specification

Fig. 3.14: WriteOperation Behavior.

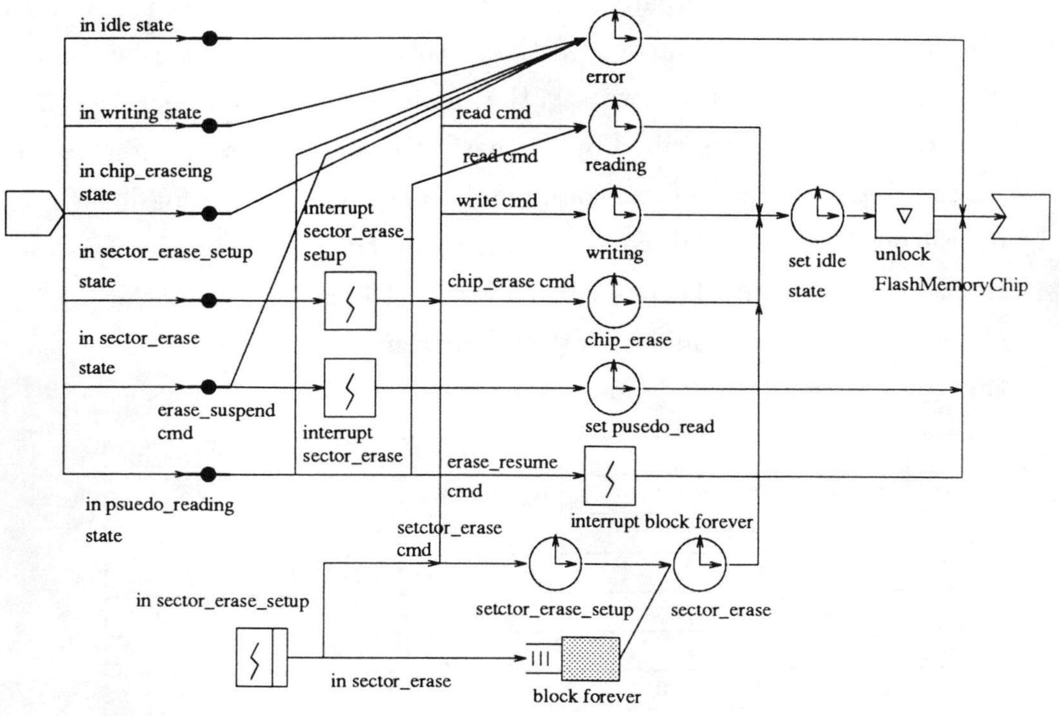


(a) Workbench Specificatiom

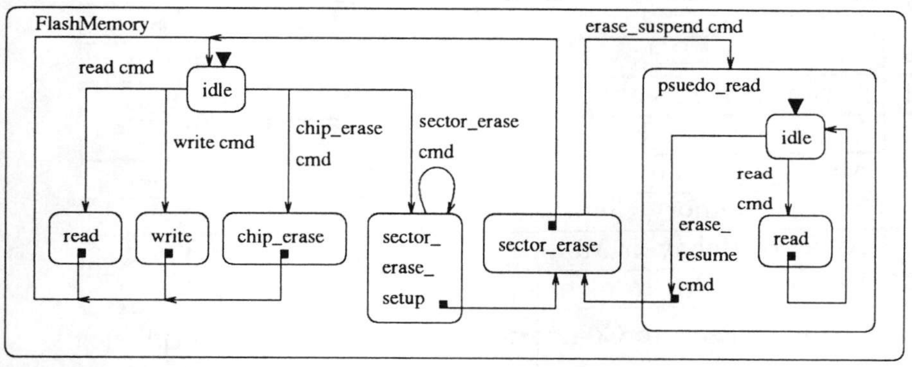
```
EraseFunc
/* VHDL Program */
wait until FlashMemoryChip=0;

if a concurrent erase command
  set concurrent erase command;
else
  set new erase command;
send erase command to FlashMemory;
```

(b) Workbench Specification

Fig. 3.15: EraseOperation Behavior.



(a) Workbench Specification



(b) SpecChart Specification

Fig. 3.16: FlashMemory Behavior.

4 Comparison

In this section we compare features of SpecSyn and Workbench model. Table 1 shows the comparison. If the feature can be described by the basic elements and operations, we say the model support this feature. If the feature can not be described by the basic elements and operations but can be described by a predefined compound object composed of basic elements, operations and C/VHDL code, the model partially support this feature. If the feature can only be described by a customized combination of basic elements, operations, and C/VHDL program or can not be described by this tool, we say the model dose not support this feature.

	Feature	SpecSyn	Workbench
1	State Transition	Yes	No
2	Behavior Hierarchy	Yes	No
3	Concurrency	Yes	Yes
4	Serialization	Yes	Partial
5	Pipeline	Partial	Yes
6	Program Construct	Yes	Yes
7	Exception	Yes	Yes
8	Behavior Completion	Yes	No
9	Queue	Partial	Yes
10	Non-queued Behavior	Yes	Partial
11	Performance Analysis	Partial	Yes
12	Design Quality Estimation	Yes	No
13	Message Passing Comm.	Partial	Partial
14	Shared Memory Comm.	Yes	Yes
15	Array of Behavior Module	No	Yes
16	Refinement for Implementation	Yes	No

Table 1: Feature Comparison of SpecSyn and Workbench.

(1) State Transition. SpecChart represents states by program-states and state transitions by TI and/or TOC arcs. Workbench dose not support state transition. Though Workbench can build a network look like a state transition diagram, input requests start at the same nodes every time. It dose not behave like state transition in which transition starts from current state, not a fixed state. State transition must be translated into data flow and expressed in C code or queuing network as the example in Fig. 3.16.

(2) Behavior Hierarchy. Workbench dose not support behavior hierarchy. Though Workbench provide **Procedure Reference** node, it represents a procedure call not a hierarchy relation. The called function is not a subbehavior of the calling function. In one execution path, process X called process Y, but process Y can call process X in other execution pathes. This causes some inconvenience as the interrupt specification in Fig. 3.3.

(3) Concurrency. Both tools support concurrent behavior. SpecChart defines the concurrency when decomposing behavior into concurrent program-substates. The concurrency is explicitly expressed and is static. In Workbench, nodes on different pathes which starts from a same fork node, a same split node, or different generation nodes can be concurrent. Fork node, split node, and generation can generate multiple requests. When these requests flow into different nodes, they become concurrent behaviors. This concurrency is dynamic and implicit. They are decided by the flow of input requests in run-time. As examples of Fig. 4.1, though the networks in (a), (b), and (c) look similar, they represent concurrent, pipelined, and sequential behaviors respectively. Because conditions associated with network edges decide requests' flow direction, requests flowing to same path are pipeline or sequential and requests flowing to different pathes are concurrent. In Fig. 4.1(a), reading odd and even byte are served by different servers. So requests for reading odd and even byte go to different path and are concurrent. In Fig. 4.1(b), one path is serving for address larger than 64K and the other is for address less than 64K, so reading word 100 will be decomposed as reading byte 100 and 101. These two requests will go to the lower path. After reading address 100, translating byte 100 execute at the same time with reading byte 101. This is a pipeline behavior. In Fig. 4.1(c), reading address 100 lock resource **x** while it is in processing. Reading address 101 was servered after byte 100 was translated and resource x was released. So reading address 100 and 101 execute sequentially.

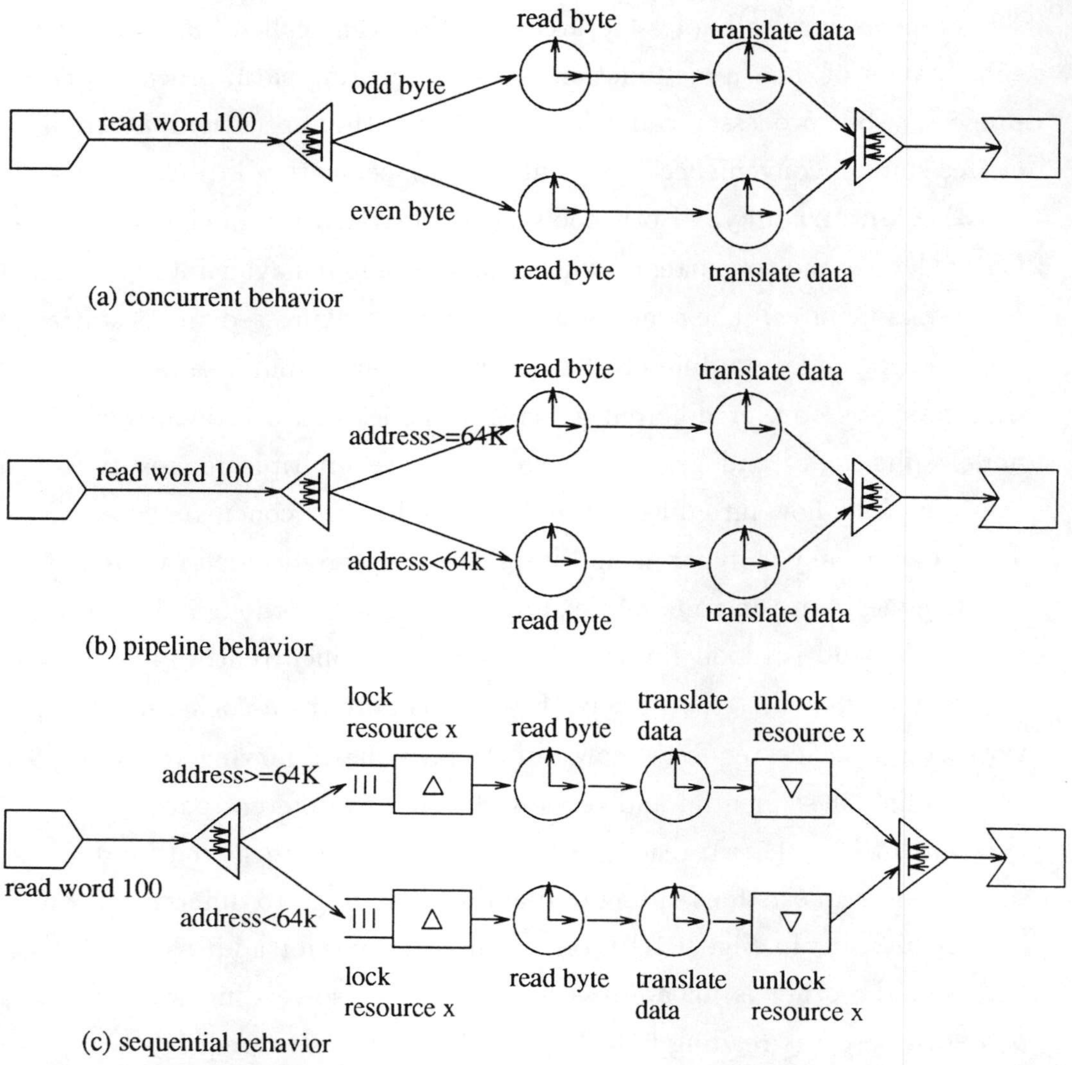


Fig. 4.1: Concurrent and Sequential Representation in Workbench.

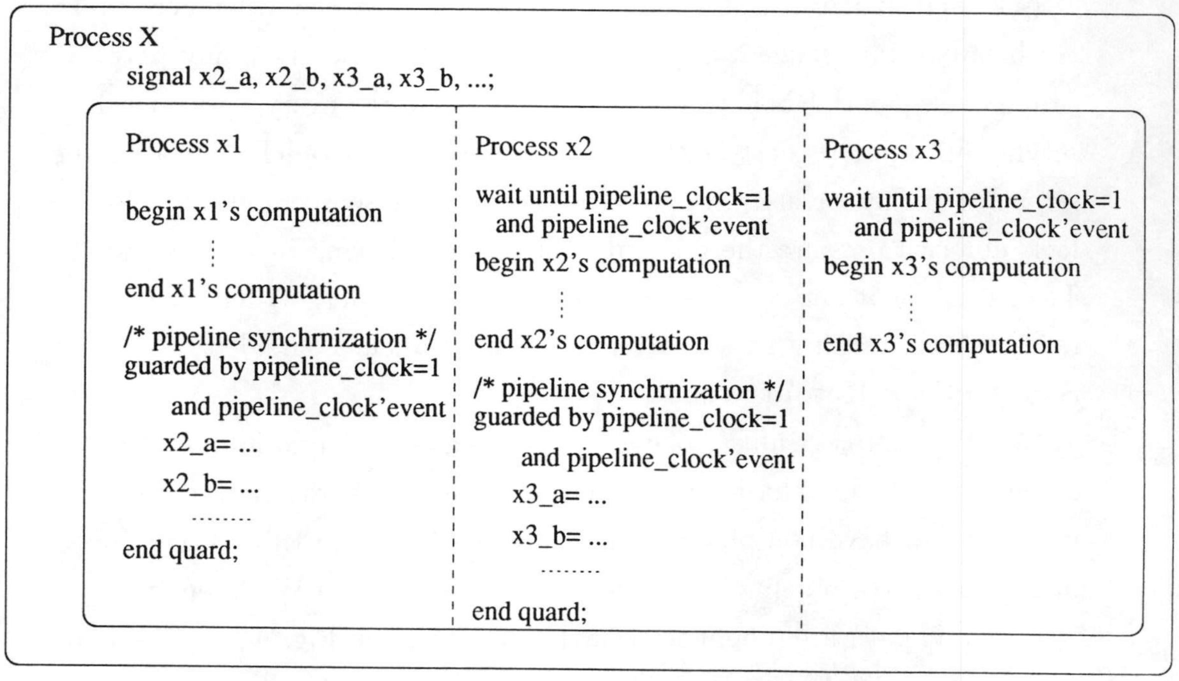
(4) Serialization. SpecChart specifys the serialization when decomposing behavior into sequential program-substates. Workbench only partially supports sequential behavior. In queuing network, every node which is serving an input request at the same time execute parallelly. To describe sequential behavior, user has to create a dummy resource and use resource lock/unlock to ensure there is only one request flowing in a set of nodes. Then this set of nodes will execute in sequential order. When there is already one request in this set, latter requests are queued outside this set that is in the **Lock Resource** node.

(5) Pipeline. Unlike concurrent and sequential relation, pipeline is an implementation characteristic, not a behavior characteristic. A same behavior can have non-pipeline implementation, two pipeline stage implementation, three pipeline stage implementation etc.. In Workbench, nodes in the same path are pipelined behaviors as shown in Fig. 4.1(b). Pipeline computation should be represented as concurrent program-states in SpecChart and ensure pipeline relation by using synchronization mechanism as the pipeline program-substates **x1**, **x2**, and **x3** in Fig. 4.2.

(6) Program Construct. Both tools support program construct. SpecChart uses VHDL language and Workbench uses C language.

(7) Exception. Both tools support exception. SpecChart uses a TI arc starting from the interrupted behavior to the following behavior to describe an interrupt. Workbench uses interrupt node and resume node to describe interruption. When a request flow through an interrupt node, an interrupt was generated. Nodes that will be interrupted were specified in the interrupt node. All requests executing and/or waiting in these nodes go to the resume node immediately. Because the exception was specified in the source side, the interrupted node sometimes did not know which interrupt occurred when there were multiple interrupts. In this case, the interrupted node will have a wrong reaction. Fig. 3.3 shows an example of modeling interrupt.

(8) Behavior Completion. SpecChart has an explicit completion point for every program-state. Workbench has no behavior completion expression. When there is no input request, the behavior stop. So it is decided by



```
Process X
        signal x2_a, x2_b, x3_a, x3_b, ...;

    Process x1
    begin x1's computation
            ⋮
    end x1's computation
    /* pipeline synchrnization */
    guarded by pipeline_clock=1
          and pipeline_clock'event
        x2_a= ...
        x2_b= ...
          .......
    end quard;

    Process x2
    wait until pipeline_clock=1
       and pipeline_clock'event
    begin x2's computation
            ⋮
    end x2's computation
    /* pipeline synchrnization */
    guarded by pipeline_clock=1
          and pipeline_clock'event
        x3_a= ...
        x3_b= ...
          .......
    end quard;

    Process x3
    wait until pipeline_clock=1
       and pipeline_clock'event
    begin x3's computation
            ⋮
    end x3's computation
```

Fig. 4.2: Pipeline Representation in SpecChart.

the test vector specified in **Generation** nodes. **ProcedureEnd** and **Termination** nodes are not completion point. When requests flow through **ProcedureEnd** node, they go to next procedure. When requests flow into **Termination** node, requests terminate. But the system is still active if there are some other requests in the network. **Termination** node should only be placed at the end of whole system behavior, it can not be used as a completion point in one procedure to replace **ProcedureEnd** node. Otherwise, requests terminate in one procedure and followed procedures will not be executed. Computation was only partially performed.

(9) Queue. Workbench provides many types of queue attaching to different kinds of server. They can express queue behavior easily. In SpecChart, queue behavior can be written in VHDL code and used as a library function call. So queue functions can become a standard library, not a customized function for each different behavior.

(10) Non-queue Behavior. In Workbench, when one server is busy, the incoming requests are always queued. If an input value change twice in that period, there are two input requests in the queue. If the latter value should overwrite the previous value, it is hard to describe in Workbench. In SpecChart, latter updated value always overwrite the previous one.

(11) Input Dependent Statistic Performance Analysis. SpecSyn did not support input dependent performance analysis. User has to insert VHDL codes for probing the current execution time and performing statistic calculation. To generate analysis result in Workbench, user only has to turn on switch of each node, procedure, and resource.

(12) Design Quality Estimation. SpecSyn provides five design metrics: performance, area, pin number, monetary cost, and bus width. They are estimated by using library information. The only design metric provided by Workbench is performance. The metric was calculated by simulation, not by estimation. User has to provide basic performance information of each subbehavior. They were specified by **Delay** nodes.

(13, 14) Communication. Both tools provide shared memory communication by using global variable in C and VHDL language. But message passing communication was only partially supported. User can write a message passing library in C or VHDL that is basically a queue.

(15) Array of Behavior Module. In SpecChart, user has to duplicate the description of program-state by editor. After duplicating program-state description, arcs among these program-states will also change. A node in Workbench can represent only one instance of queue/server, one queue with multiple servers, or multiple queues with multiple servers by specifying a node's **server** and/or **dimension** attribute. If the **server** attribute of a node is larger than one, these servers share a same queue. These servers have the same function. If the **dimension** of a node is larger than one, this node represents more than one queue and each queue has **server** number of servers. These queues are treated as an array. Each queue can be referenced by different index. A rule was associated with each incoming edge of this node for how to select the array element for every arriving request. Fig. 4.3 illustrate this feature.

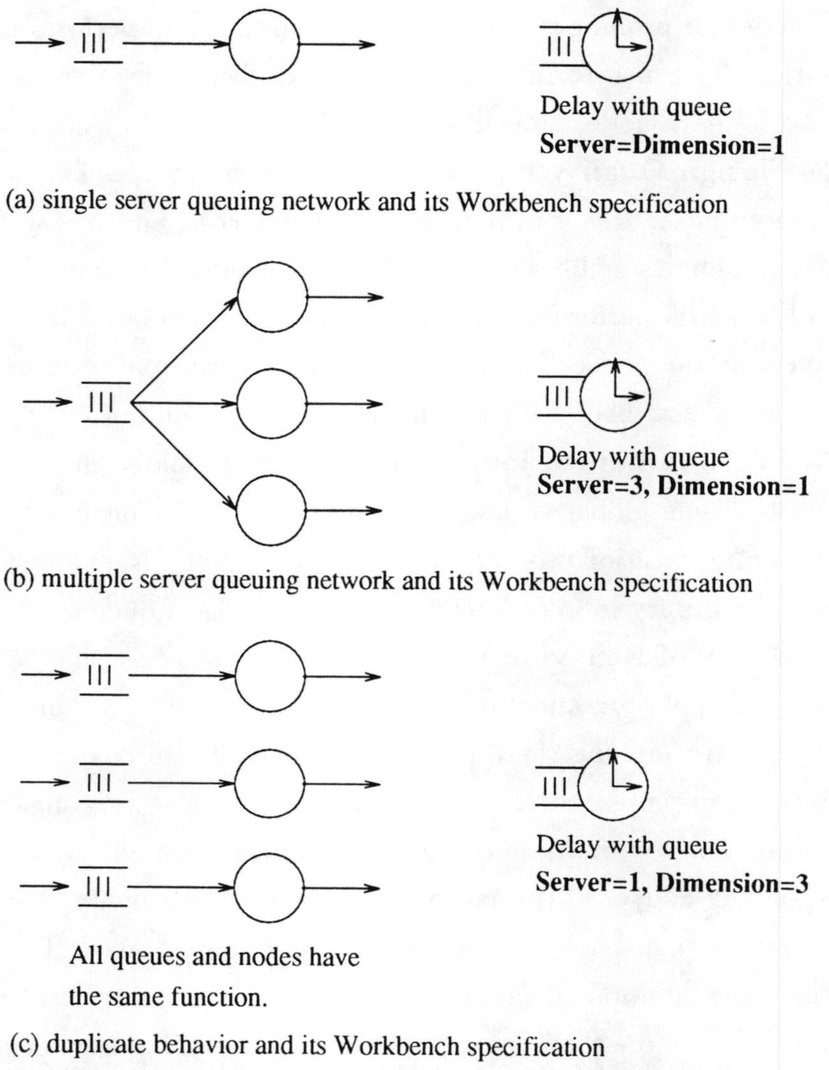

Delay with queue
Server=Dimension=1
(a) single server queuing network and its Workbench specification
Delay with queue
Server=3, Dimension=1
(b) multiple server queuing network and its Workbench specification
Delay with queue
Server=1, Dimension=3
All queues and nodes have the same function.
(c) duplicate behavior and its Workbench specification

Fig. 4.3: Multiple queues/servers representation in Workbench

(16) Refinement for Implementation. In Workbench, the concurrent, sequential, scheduling, and partition relations were specified by the fork/join/split nodes, lock/unlock nodes, and delay nodes in the modeling. They are fixed. Workbench can not change it. In SpecChart, user only specify the behavior requirement for concurrency and serialization. Delay information is optional. SpecSyn will schedule and partition the behavior according to hardware resources. So concurrent behavior can executed sequentially due to hardware constraint without changing the modeling. Component interface after partition was automatically generated.

5 Enhancement of SpecSyn

From the comparison, SpecSyn are weak in five features: pipeline, queue, input dependent statistic performance analysis, message passing communication, and array of behavior module. Message passing communication is basically a queue implementation. Here we propose the enhancement of SpecSyn over these features.

5.1 Pipeline

There are two possible ways to describe pipeline in SpecChart. The first way is to provide pipelined program-state in SpecChart besides concurrent and sequential program-states. The second way is to specify pipeline stage number for each program-state in SpecSyn and let SpecSyn refine the behavior automatically. Because pipeline is an implementation characteristic, the second way fit design process better than the first one. But automatic pipeline generation is more difficult and requires further research.

Here we propose the implementation of the first approach. Users partition behavior into pipeline program-states as in Fig. 5.1(a). Pipeline program-states are separated by double dot line. When translating pipeline program-state into current SpecChart specification, SpecChart can find out variables that were used in latter pipeline stages as variable **a**, **b**, **c**, and **d** and generate temporary variables for them. Instead of updating original variables in computation, translated code update the temporary variables. At the end of computation, guarded statements are appended to update original variables that will be used at latter pipeline stages. Fig. 5.1(b) is the translation of Fig. 5.1(a).

5.2 Message Passing Communication and Queue

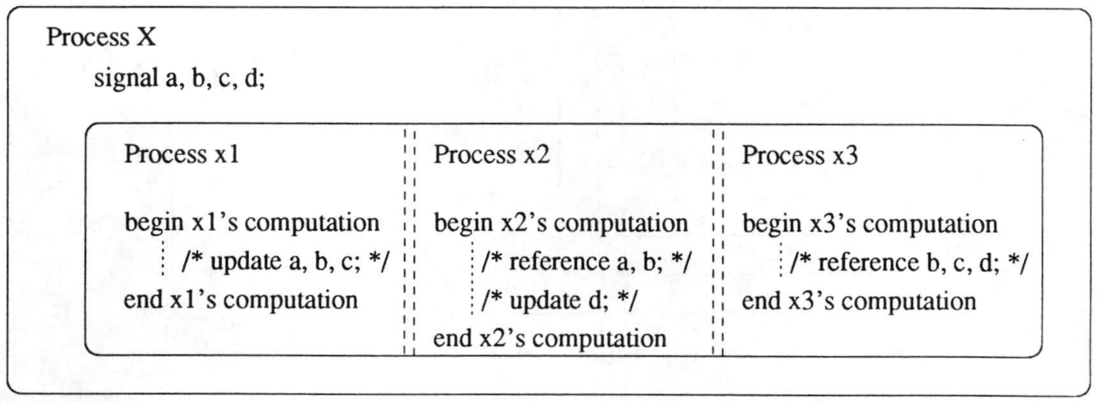
Message passing communication is done by calling **SendMessage** and **ReceiveMessage**. They are equivalent to put queue and get queue. So SpecChart can provide SendMessage and ReceiveMessage function call in its standard library. The parameter for these two function calls are channel name and message. Each communication channel has a unique name. User can use these two function call to pass message.

In SpecChart, there are concurrent and sequential program-states. Because only one sequential program-substates can be active at any time, so no queue is needed for sequential program-substates except the first one in the toppest level of a concurrent program-state. As in Fig. 5.2, program-state **A**, **D**, **E**, **F** and **G** may need queues and the others do not. All concurrent behaviors and the sequential behavior mentioned above may have or may not have a queue. It depends on system behavior. If a program-state needs a queue, SpecChart can automatically insert a **GetQueue** function call at the beginning of that program-state. The only extra effort for user is to insert **PutQueue** function call at the place where data were generated because transitions in SpecChart are control flow not data flow. The more queue mechanisms SpecChart provides such as FIFO queue, FILO queue and priority queue, the less chance user has to write a queue in VHDL.

With library of queue functions, SpecChart can provide message passing communication and queued subbehavior. User only has to insert **SendMessage**, **ReceiveMessage** and **PutQueue** in proper place in the modeling.

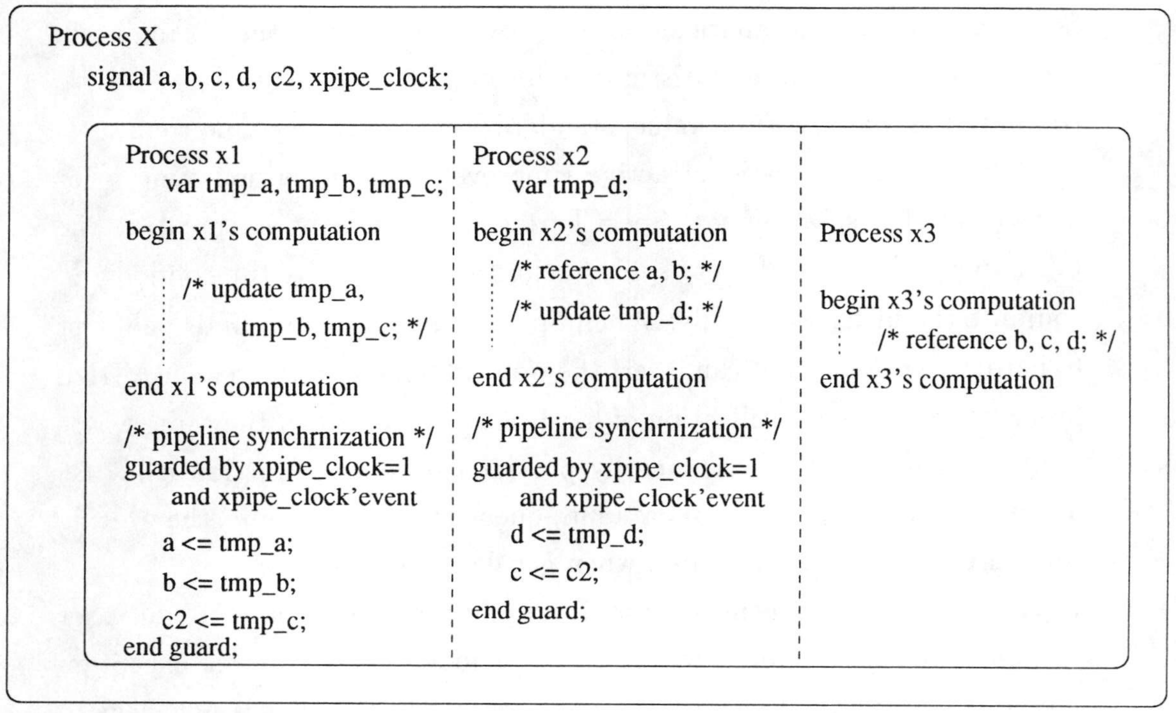
5.3 Input Dependent Statistic Performance Analysis

The major statistic data got from queuing model are inter arrival time, queue length, queue waiting time, utilization and response time. Fig. 5.3



```
Process X
    signal a, b, c, d;

    Process x1                     | Process x2                     | Process x3

    begin x1's computation         | begin x2's computation         | begin x3's computation
        /* update a, b, c; */      |     /* reference a, b; */      |     /* reference b, c, d; */
    end x1's computation           |     /* update d; */            | end x3's computation
                                   | end x2's computation           |
```

(a) Proposed Pipeline Specification.



```
Process X
    signal a, b, c, d,  c2, xpipe_clock;

    Process x1                        | Process x2                       |
        var tmp_a, tmp_b, tmp_c;      |     var tmp_d;                   |
    begin x1's computation            | begin x2's computation           | Process x3
                                      |     /* reference a, b; */        |
        /* update tmp_a,              |     /* update tmp_d; */          | begin x3's computation
                tmp_b, tmp_c; */      |                                  |     /* reference b, c, d; */
                                      |                                  | end x3's computation
    end x1's computation              | end x2's computation             |
    /* pipeline synchrnization */     | /* pipeline synchrnization */    |
    guarded by xpipe_clock=1          | guarded by xpipe_clock=1         |
        and xpipe_clock'event         |     and xpipe_clock'event        |
        a <= tmp_a;                   |     d <= tmp_d;                  |
        b <= tmp_b;                   |     c <= c2;                     |
        c2 <= tmp_c;                  | end guard;                       |
    end guard;                        |                                  |
```

(b) Translation of Pipeline Specification.

Fig.5.1: SpecChart Enhancement for Pipeline.

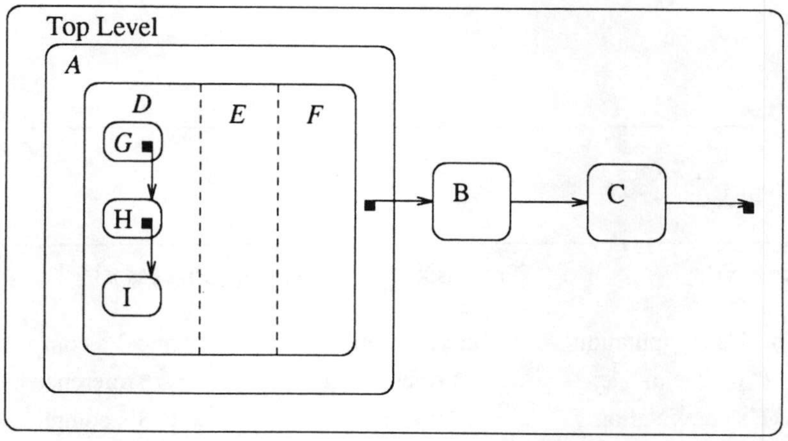


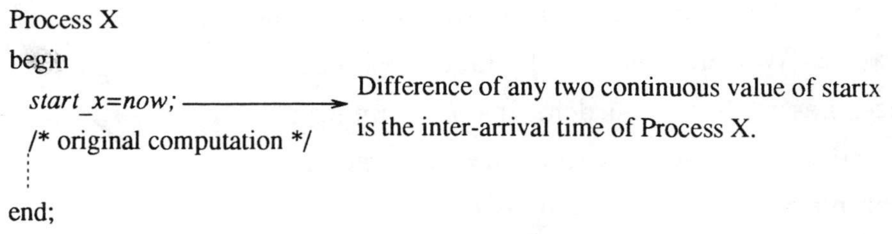
Fig. 5.2: Possible Queue Insertion in SpecChart.

shows how to add these functions in SpecChart. To get the inter arrival time, we need to know the every starting time of the probed subbehaviors. So SpecChart can automatically insert **now** statement in the beginning of every subbehavior and a statistic function call after the **now** statement to compute the max/ave/min value, standard variance and sample count.

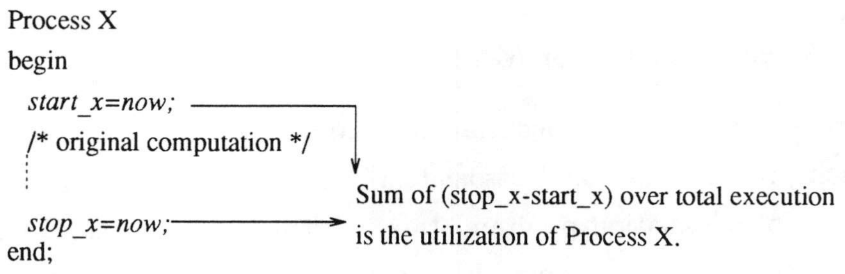
Utilization is the ratio of active time over total execution time. For utilization of a subbehavior, SpecChart can insert **now** statment at the beginning and ending of every subbehavior to get the active time period and compute the utilization easily. When SpecSyn partions and binds behavior to hardware resources, it can insert the **now** statement in the refined VHDL program to calculate utilization of hardware component in the same way.

Queue length can be very easily provided by queue function call. By inserting time stamp and real data into queue at the same time, the waiting time in queue can be calculated when get data out of queue.

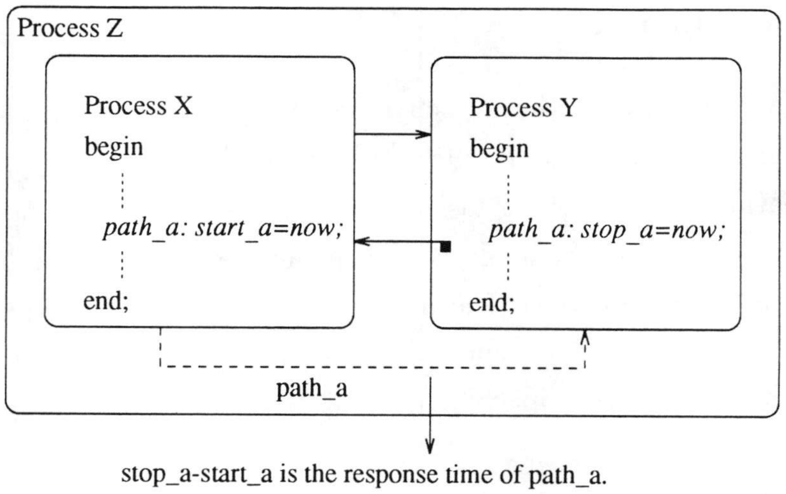
To calculate the execution time from one point to another point, user can insert **now** statement in these two points. They can get execution time in smaller granularity that is in statement level but it is not clear to user which execution path is probing. If SpecChart shows program-states in graphic, user can select the probing path more easily. This is clearer but it is in larger granularity that is program-state level. User can both select the probing path and insert **now** statement to get statement level


Process X
begin
start_x=now;
/* original computation */
end;
Difference of any two continuous value of startx
is the inter-arrival time of Process X.

(a) Inserted Statement for Getting Inter-arrival Time.


Process X
begin
start_x=now;
/* original computation */
stop_x=now;
end;
Sum of (stop_x-start_x) over total execution
is the utilization of Process X.

(b) Inserted Statement for getting Utilization.



(c) Enhancement for getting response time.

Fig. 5.3: SpecChart Enhancement for Performance Analysis.

execution time. Pathes and **now** statements can associate with each other by *label* name. In this way, SpecChart can get the response time in an easy to capture way.

SpecChart can provide table for specifying how to categorize the statistic data as Workbench dose. If there are input or internal signal for this use, user has no heavy burden. If the grouping rule is not easy to describe by these signals, user has to create some extra signal for this purpose. It will complex the behavior modeling.

After adding these functions, user can get the input dependent statistic performance analysis by running VHDL simulator.

5.4 Array of Behavior Module

To provide this feature, SpecChart must allow user to specify the dimension of each program-state and associate the transition a rule for chosing the instance of destination program-state. When translating to VHDL, each program-state instance should have a unique name.

6 Conclusion

This paper has shown that SpecSyn has many good modeling feature over Workbench. User can save much time in the design process and get more information. For modeling tasks need some special feature such as input dependent performance analysis and queue mechanism, user has to write VHDL programs for these features in current SpecChart implementation. With the proposed enhancement, SpecSyn will be suitable for these applications, too. This makes SpecSyn applicable to wider variety of applications.

References

[1] D.D. Gajski, F. Vahid, S. Narayan and J. Gong, *Specification and Design of Embedded Systems*, 1994

[2] D.D. Gajski, F. Vahid and S. Narayan, *"A System-Design Methodology: Executable-Specification Refinement"*, in Proceed-

ings of European Conference on Design Automation (EDAC), 1994

[3] E.D. Lazowska, J. Zahorjan, G.S. Graham and K.C. Sevcik. *Quantitative System Performance*, Prentice-Hall, 1984

[4] *SES/Workbench User Manual*

[5] *SES/Workbench Reference Manual*