1.2 Purpose of this document

This document is intended to break the chain of simulator initiation. This

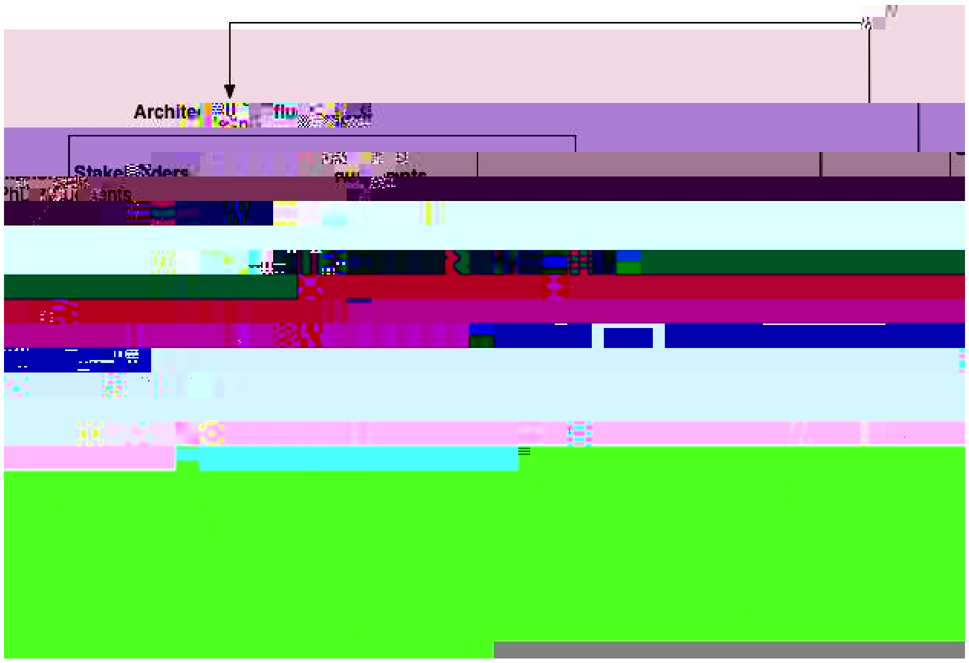

Figure 1: The Architectural Business Cycles of SESC.

2.4 Stakeholders

the users are usually commited once the code has been verified to work. Other

Conference's
Call for Papers
Student X's
Proposal
version N
proposal
proposal
Student X's
Paper
Student
Paper
integration
Conference's
Submission
Subm
each
area tools
version N+1

Figure 2: The ABC of the i-acoma group.

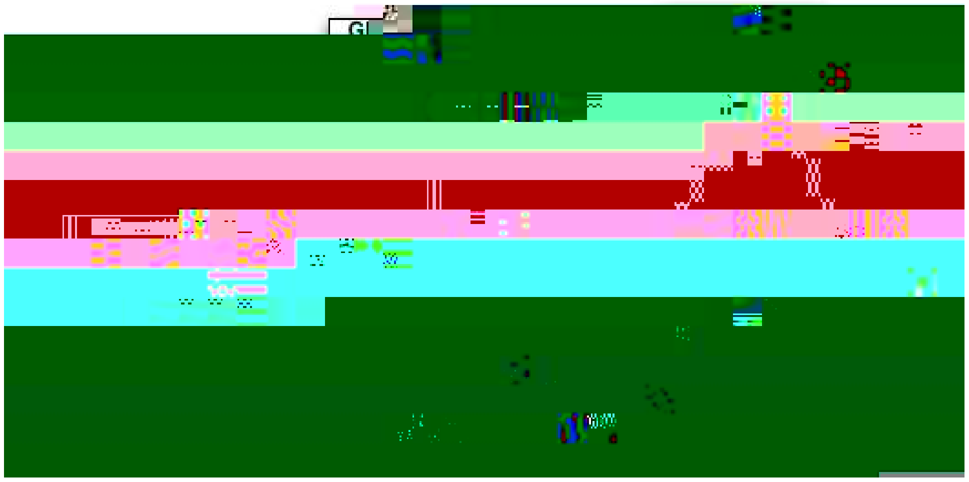

Figure 3: The class interactions that model the pipeline.

Justification

It is very important that emulation without modeling timing be very fast. In many benchmarks there are lengthy initialization sections, which may be longer than the main portion of the program.

In "rabbit mode," in which Instructions are only emulated and no timing simulation is performed, the simulator executes instructions about 1000 times faster than in full simulation mode.

3.3 Pipeline view

In SESC, the GProcessor (generic processor) object type coordinates interactions between the di erent pipeline stages. The upper-level interface to the GProcessor object is the advanceClock() function. The advanceClock() function advances each stage in the pipeline one clock cycle. It does this by first calling a function to fetch instructions into the instruction queue. It then calls a function to issue instructions from the queue into a scheduling window. There are two *clusters* that schedule and execute instructions, one for integer and one for floating-point instructions, and each has its own scheduling window. Instruction scheduling and execution is handled in other parts of the simulator. Finally, a function is called to retire already-executed instructions from the reorder bu er.

All of the class interactions that model the pipeline are shown in Figure 3.

queue for each cluster. The issue() function calls addInst() for each instruction. If addInst() fails for an instruction, issue() returns, and issue() will try again in the next cycle to issue that instruction.

The addInst() function checks several things before it confirms that the instruction can be issued. First, it checks that there is space in the reorder buffer. Second, it checks that there is a free destination register. Third, it checks that there is space in the scheduling window for the cluster. Finally, based on the specific resource that an instruction uses, it performs other checks. For example, for loads, it will check that the maximum number of loads has

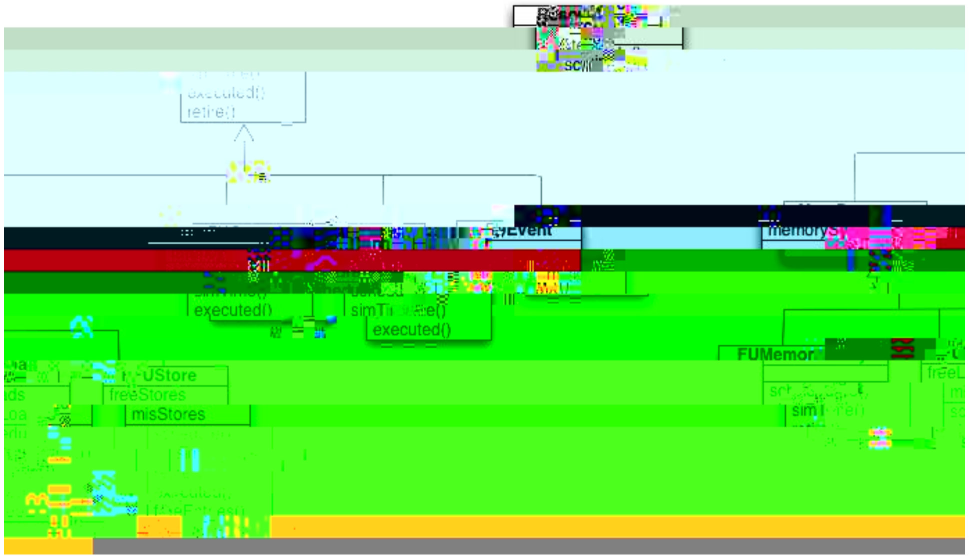

Figure 4: The Resource class hierarchy.

are ready for execution in a specific Resource, as each Resource can only execute a small fixed number of instructions per cycle.

referred to as L1 caches. Below this is a larger, slower L2 cache. In many configurations, there is also an o-die L3 cache that is even larger and slower than the L2 cache. Caches have many parameters. SESC models d(s)-erent cache:

- S(s)-zes
- Hit & Miss latencies
- Replacement policies
- Cache-line sizes
-

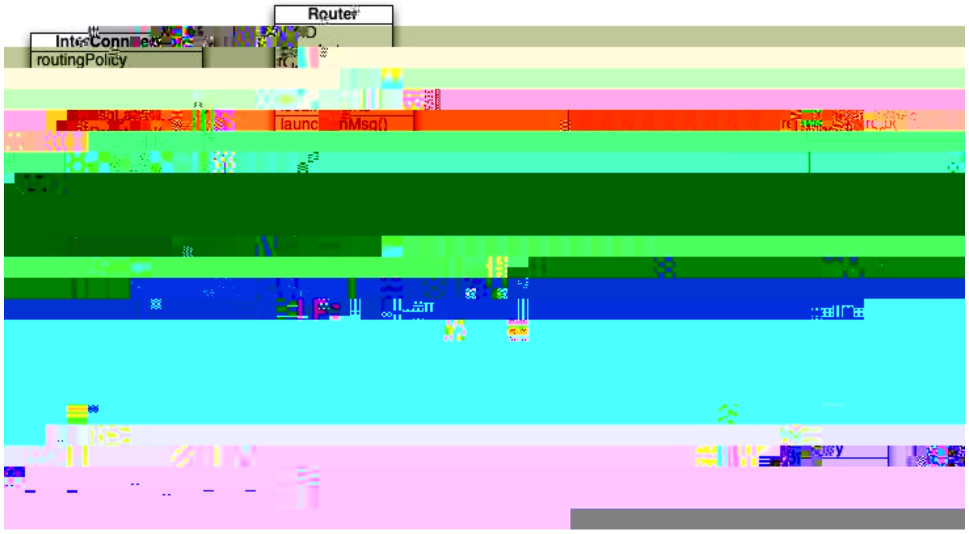

Figure 5: Interconnection network class organization.

The important thing is that all types of Caches and Buses inherit from a common class, MemObj, which defines a common interface consisting of access(), returnAccess(), and other less important functions. (ReturnAccess() is called by goUp() when an access returns to a higher-level cache from a lower-level cache.) This common interface allows fully-configurable cache hierarchies. Each Cache subtype can have a different manner of handling requests internally, as long as it conforms to this interface to upper and lower-level caches.

In a multiprocessor system, at the lowest level, each processor's caches and the main memory are connected. The manner in which they are connected is described now.

3.5 Interconnection network

The interconnection network refers to the communication channel between processors in a large multiprocessor system. Examples are a bus or a hyper-cube.

The job of an interconnection network in a parallel machine is to transfer data from any source node to any desired destination node. The network is composed of switches that route the packages from the source to the target. Each network node contains a routing table, which stores network path and status information and is used to select the most appropriate route to forward the packages along.

Figure 5 shows a UML representation of the classes that compose the network module. The InterConnection class represents the whole network layout. An InterConnection object is defined by two components:

- A set of Router objects. The Router class represents a router in an interconnection network. It decides where to send the packages it receives according to the routing table and the ports traffic flow. Each Router

In this example, all intermediate compilation files and the final binary will be stored in the build_

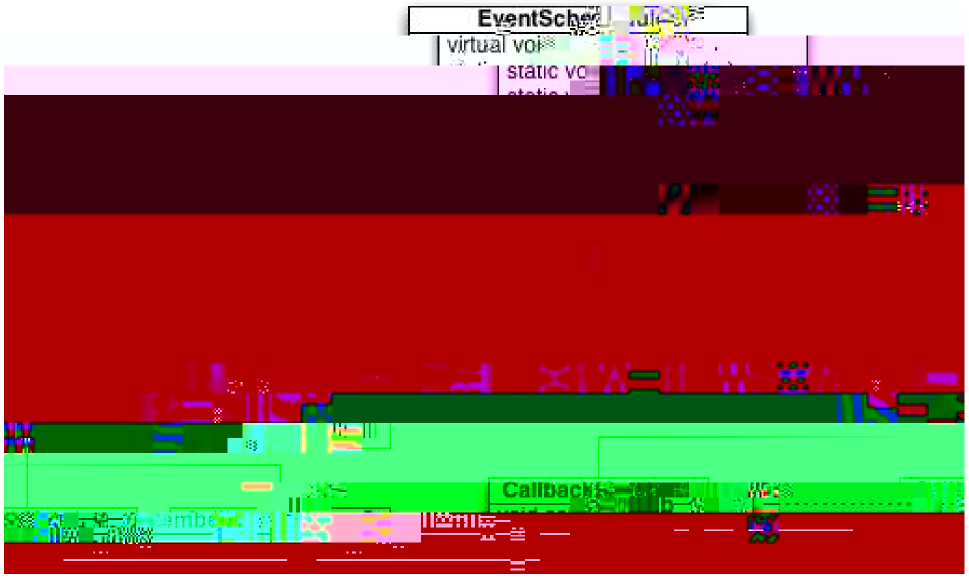

Figure 6: The callback class hierarchy.

- `virtual void call()=0:`. Each concrete class will implement it.
-

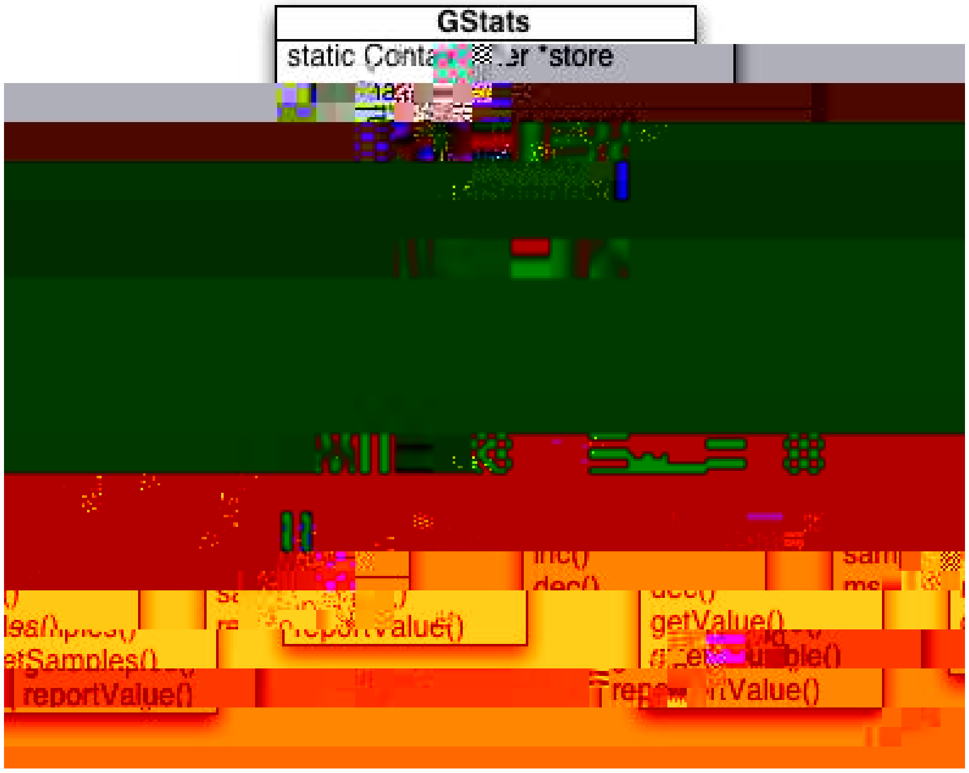

Figure 7: T0e GStats class hierarchy.

been instantiated. Every `GStats` object *subscribes* itself to that global list of statistics in its constructor.

Three classes derive from `GStats`:

- `GStatsCntr`: A simple counter.
- `GStatsAvg`: An average counter.
- `GStatsMax`: Stores the max value of all the Tiven values.

All of them must implement the `reportValue()` function, which prints the value it stores or calculates. `reportValue()` will be called from the `GStats` static `report()` function, which traverses the Tlobal list of `GStats` objects.

Figure 7 shows a simplified UML diagram of t0e GStats 0ierarchy.

4.3 Pools

Some objects in SESC are allocated once and used until the simulation is fin-