HASE: A Flexible Toolset for Computer Architects

R. N. Ibbett, P. E. Heywood and F. W. Howell

Computer Systems Group, Department of Computer Science, University of Edinburgh, Edinburgh, EH9 3JZ UK

HASE is a Hierarchical computer Architecture design and Simulation Environment (HASE) which allows for the rapid development and exploration of computer architectures at multiple levels of abstraction, encompassing both hardware and software. The components of a computer system lend themselves naturally to being modelled as objects, so HASE has been implemented in an object-oriented language. Within HASE there are graphical entity design and edit facilities, entity library creation and retrieval mechanisms, an animator, and statistical analysis and experimentation tools for deriving system performance metrics. HASE uses an object-oriented database management system (ObjectStore) to make the design objects and the entity library persistent. For each architecture model HASE allows many experiments with varying parameters to be performed. The database facilities provided through HASE manage not only the results of each experiment, but also their relationship to the state of the architecture model that produced these results, including all input and output parameters and their values during the experiment. This paper describes the design of HASE, some of the varied projects which have used it, and the future direction of the system.

Received July 26, 1995, accepted November 8, 1995

1. INTRODUCTION

The Purdue Workshop on Grand Challenges in Computer Architecture for the Support of High Performance Computing [1] identified four 'Grand Challenge Problems in Computer Architecture'. HASE, the Hierarchical computer Architecture design and Simulation Environment developed at the University of Edinburgh, is a tool which addresses the fourth of these: 'to develop sufficient infrastructure to allow rapid prototyping of hardware ideas and the associated software in a way that permits realistic evaluation'. Sophisticated VLSI design tools have been in existence for a number of years but it is only recently that attention has been focused on providing higher level simulation and animation tools for computer architects. Thus the HASE project has aimed to address two major problem areas: high level simulation and visualization of computer architectures, and simulation of parallel systems.

The hierarchical nature of computer architecture and design has been well understood for many years, e.g. Bell and Newell's PMS, ISP and RTL levels [2]. HASE allows the designer to move freely between these levels and to select the appropriate simulation level for different parts of the system in order to strike a balance between simulation accuracy and processing time. To meet all the aims for the environment, however, attention also had to be focused in the area between the domain of hardware simulators and general purpose simulation packages. Hardware simulators are typically inappropriate for dealing with software layers and general purpose simulation packages are not normally designed with hardware in mind. The usual approach to this problem is to write project-specific simulators in a language such as C + +. This provides a high degree of flexibility, but also an amount of wheel re-invention.

Many commercial CAD tools are moving progressively towards higher levels of abstraction, and the use of hardware description languages such as VHDL and Verilog for hardware system simulation is becoming widespread. Since much effort has been invested in developing these toolsets it would be convenient to extend them to higher levels of simulation. However, most are not particularly suited to this task at present. In [3], for example, external C routines were written to compensate for VHDL's deficiencies in this respect.

Specialized tools include Ptolemy [4] at Berkeley which defines a framework for simulating and prototyping heterogeneous systems, and work at the University of Florida has involved simulating microprocessor-based parallel computers using processor libraries [5]. At UMIST the SES/workbench [6], a general queuing model tool, has been adapted to simulate the ARM processor [7]. At the Illinois Institute of Technology Chicago a prototype version of MIES [8] has been developed to visualize Register Transfer Level descriptions and a newer version is currently being implemented in an object oriented programming language.

At the same time, there has also been interest in developing mathematical formulations for modelling discrete event systems, most notably Zeigler's DEVS formalism [9] together with its primarily non-graphical implementation, DEVS-Scheme.

The ideas for HASE grew from a simulator built for an MC88000 system [10], written in occam and run on a Meiko Computing Surface at the Edinburgh Parallel

Computing Centre. However, since the components of a computer can be treated very naturally as objects, HASE itself has been developed using object oriented simulation languages, the first prototype [11] using DEMOS [12] and the current version Sim + + [13]. Sim + + is essentially a superset of C + + which includes a set of library routines to provide for process oriented discrete event simulation and a run time system for multi-threading many objects in parallel and keeping track of simulation time.

In the same vein, HASE now also uses an object-oriented database management system, ObjectStore [14]. The environment includes a design editor and object libraries appropriate to each level of abstraction in the hierarchy, plus instrumentation facilities to assist in the validation of the model. HASE also provides *model exploitation facilities* based on [15] and [16] allowing performance measurements to be derived from simulation runs. The system can thus be set up to return event traces and statistics which provide information at the PMS level, for example, about synchronization, communication and memory latencies.

The user interface to HASE is via an X-Windows/Motif graphical interface. Many complex systems of interacting components can be more easily understood as a picture rather than as words. In computer architecture the dynamic behaviour of systems is frequently of interest and HASE allows users to view the results of simulation runs through animation of the design window.

The first sections of this paper present an overview of HASE, the database organization and the HASE libraries. Then follows a description of the design of a system within HASE including test software to execute on the model architecture. This is followed by a description of the Sim + + code generated by HASE, and of the way a simulation is run. Later sections describe the various ways to view the results of a simulation, gather statistics and perform experimentation on the model architecture. Finally we present some of the projects which have used HASE and consider possible future developments.

2. OVERVIEW OF HASE

Figure 1 shows an overview of the HASE system. The core of each project undertaken using HASE is the *Architecture Description*. In the case of a multiprocessor, for example, this description consists of a collection of Processor, Memory and Interconnection Network *entities*. Each entity is a multi-faceted object having an instance name, an icon (usually a bitmap), a textual description, a list of its parameters, a list of ports and a pointer to its Sim + + simulation code. The design phase of a project involves selecting the appropriate entities from a *library*, or alternatively creating them, and linking them together to form the required system. Each entity's behaviour is described in the corresponding Sim + + method (the *body*). Once the design is complete the

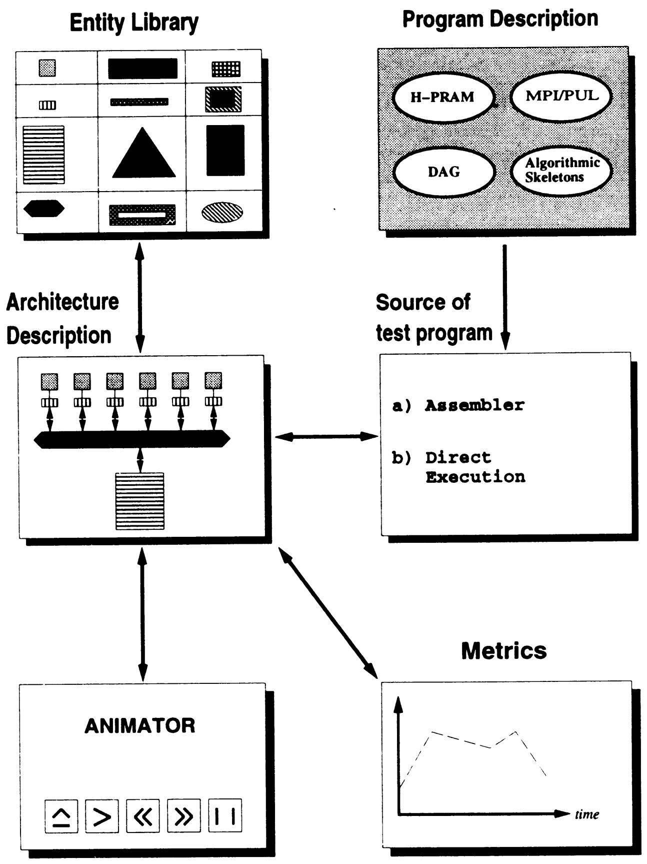


FIGURE 1. Overview of the HASE system.

description is compiled to produce the simulation code for that system.

To run a simulation, it is necessary to provide appropriate inputs. Thus test programs for the architecture can be written (in assembler, or any HLL with a compiler for the chosen processor), and the architecture, along with its program, can then be simulated. The output from the simulation run can then be used to animate the design, and thus provide visual feedback data to the designer, or to obtain performance metrics.

3. DATABASE ORGANIZATION

HASE includes an object-oriented database management system based on ObjectStore in order to allow all architecture design projects and the entity libraries to be maintained persistently. A major advantage of this approach is that in addition to its purely repository functionality, ObjectStore can also be used to manage the relationships and connectivity between objects. Furthermore, the use of object-oriented database technology provides the opportunity to exploit advanced transaction processing capabilities, such as nested transactions and rollback, and to facilitate the exploration of alternative paths while fine-tuning a model.

The database management system also allows versions of simulation models and experiments to be maintained so that an experimental program can proceed on an existing version of the model while subsequent versions are under development. By integrating a C++-based object-oriented database management system with the existing HASE environment, most of the problems associated with impedance mismatch have been avoided. All HASE environment utilities are C++ based and all relevant classes are coherent throughout the environment.

Figure 2 shows how the databases are organized. The *user startup file* contains environment variable definitions for the default library databases and the user specific project directory containing subdirectories for all individual projects. Each user can have a number of project databases, each holding a number of projects, and a number of personal entity libraries in addition to having access to the public HASE entity library.

A project in HASE is interpreted to be the set of all entities, ports, links, parameters, etc., comprising the simulation model, together with their associated Sim++ code, bitmaps, etc., and the set of all experiments performed on the model. For each architecture model, a set of experiments may also be stored. Experiments typically involve changing the value of one of the parameters of a component of the architecture and running the same simulation for each parameter value. The database supports this experimentation facility by storing not only the results of each experiment, but also their relationship to the state of the architecture model that produced the results, including all input and output parameters and their values during the experiment.

4. LIBRARIES

Libraries in HASE are repositories for entities, the basic components of the Architecture Description. Each modular, reusable entity can be archived to a library for shared or later use or retrieved from a library for inclusion in an architectural model. The storage of predesigned (and pre-tested) entities in the library means that HASE offers a reliable and convenient method for rapid prototyping.

The HASE Entity Library is a global read-only library, selected from possibly many shared libraries containing related entities. As a means of initially populating and supplementing this library for a specific site, entities from all projects migrate to a temporary holding area where the site database administrator determines which entities merit inclusion into a particular HASE Entity Library. The User Entity Library is a user's personal catalogue of entities. The number of libraries is virtually limitless, with the library in use defined as the most recently selected library.

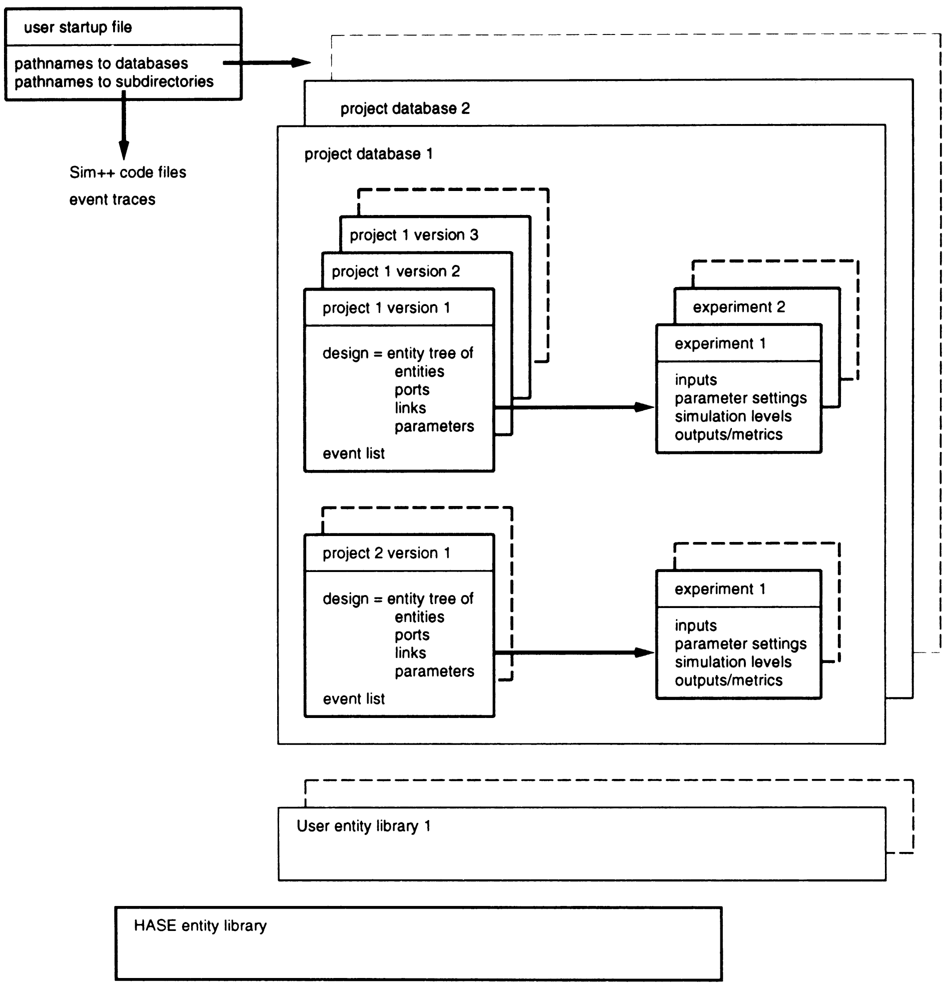


FIGURE 2. HASE database organization.

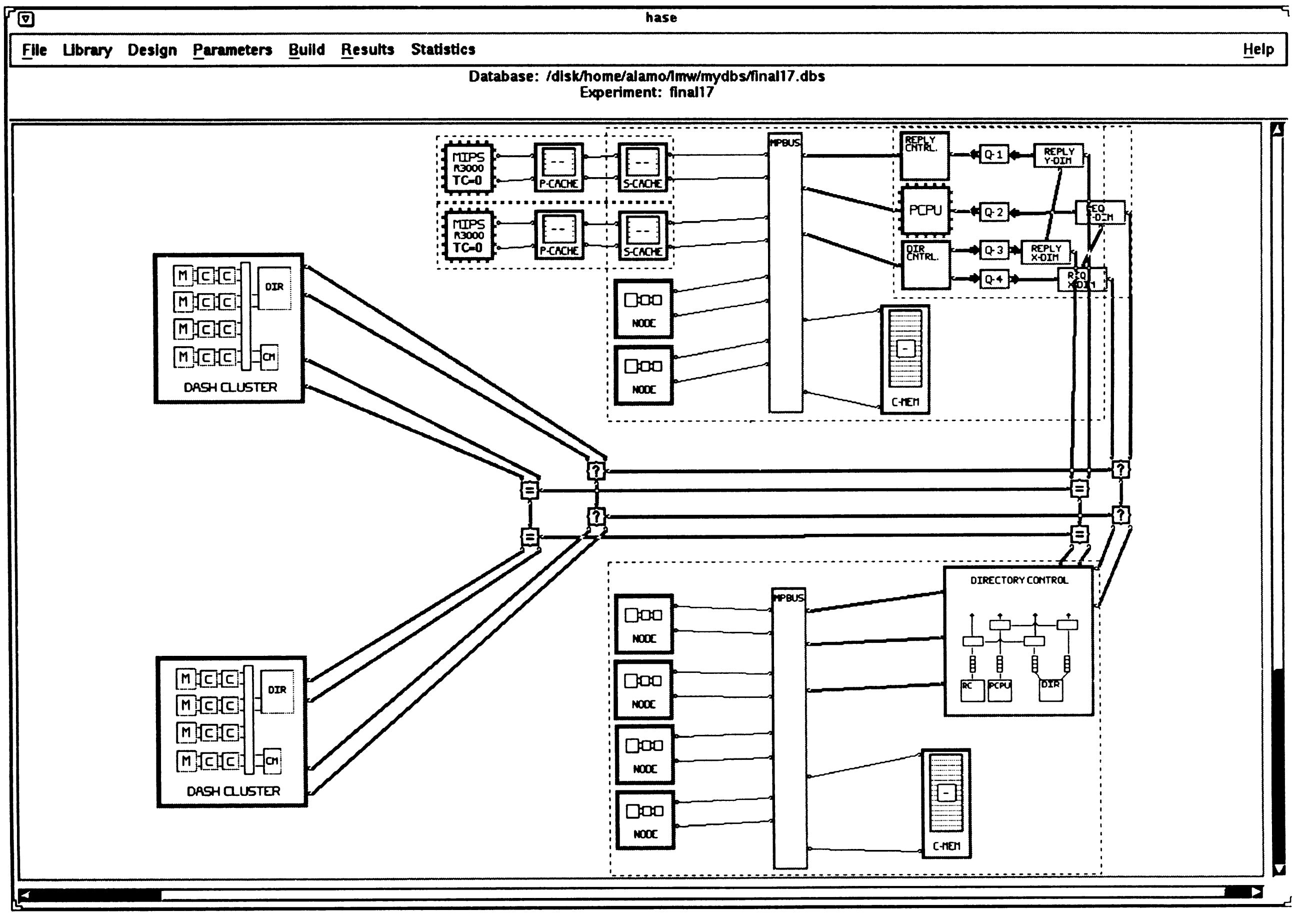
The design of a HASE Entity Library corresponds to the different levels of abstraction for the archived entities, e.g. PMS or RTL. An entity's hierarchy can be easily perused from within the library and can be included in the architectural model at various levels. The utilities are flexible enough to allow the user to map to other decomposition methodologies when creating the User Entity Library.

5. DESIGN

The basic constituent of the project is the Architecture Description which is a collection of entities with ports for data transmission across links to other entities or levels of decomposition. The architecture can be designed either *top down* by subdividing an entity into its constituent components or *bottom up* by grouping a set of components into a compound component. An example of a compound entity is a multiprocessor array, for which several different *templates* are available as library components. Currently available are one-dimensional array templates, several two-dimensional array templates with differing (pre-defined) indexing schemes, a three-dimensional torus and an Omega Network which can be instantiated for simulation at any hierarchical level. Indeed any entity can in principle be simulated at any specified hierarchical level. Figure 3 shows an example design window taken from an M.Sc. project [17] which has modelled the Stanford DASH architecture and its cache coherency protocols [18].

The DASH architecture consists of sets of processing nodes, grouped together in clusters of four and connected together via a common bus. Each node consists of an R3000 processor with a primary and secondary cache. As well as connecting the nodes together, the bus also provides access to memory which is shared between the processors and which forms part of the global address space of the system as a whole. Clusters are themselves connected together by a dual interconnection network. Figure 3 shows a four-cluster network in which the bold interconnection lines represent the request (?) and response (=) networks. The system is modelled as a three-level entity hierarchy. On the left of the figure two clusters are shown represented at the highest level, while the lower right hand cluster has been expanded to show the middle level (the dotted lines around a cluster of entities indicates expansion from a higher-level entity). The top right cluster has been further expanded to show the lowest level design of two of the nodes and also the lowest level design of the Directory Control logic, the subentities of which are responsible for ensuring inter-cluster cache coherence.

The HASE Architecture Description created in the design window describes the physical composition of the



Database: /disk/home/alamo/lmw/mydbs/final17.dbs
Experiment: final17

FIGURE 3. An example architecture in the HASE design window.

architecture. The behavioural aspects of the architecture are described by the corresponding Sim++ body methods. Sim++ is essentially C++ with functions to communicate events between entities. These events are sent to and received from ports which are the entity's link with the rest of the simulation. Entities may also have a variable number of parameters, which can be strings, integers, floating point numbers, enumerated types, structures, ranges, instructions or arrays.

To simulate a multi-processor system (say), the first task is to create an Architecture Description for it. The required entities (processors, memories, interconnection networks) are selected from a library, from a pre-defined HASE template or are custom designed. All entities can be further customized or modified from within HASE, including, but not limited to, additional subdivision (decomposition), grouping, and adding parameters. The required ports, including the link parameter are also added. The link parameter represents the data packet, message, instruction, etc., to be transmitted to/from the port over the link.

An entity may be defined at several different abstraction levels. The external interface (the set of ports) at each level must be identical, but at the lower level the entity may be composed of a set of interconnected 'sub-entities'. The abstraction level to be used for each entity is chosen at simulation time. Particular entities may be simulated at a lower level while leaving the rest at a higher level.

The entities are linked together to create the system to be simulated and each entity's behaviour is described in the Sim++ *body* method.

Global parameters can be defined for the system to be modelled. As the term implies, global parameters are accessible to all entities, for instance, the dimensions of a compound entity array.

The evaluation of an architecture normally involves the execution of test programs. An interface between simulations and a 'software level' is also needed for parallel programming or for investigating computational models. Several different approaches can be used within HASE; interpreting assembler, execution driven simulation, and interpreting a simple higher level language.

5.1. Instruction sets

One of the uses of HASE parameters is to store instruction sets. Instructions are typically divided into several different 'classes', such as load/store instructions, ALU operations, branches, etc. To deal with the resulting variety of instruction formats, HASE provides a special type of parameter, TInstrParam. For example, in

```
struct TInstr {
  TIClass iclass;
  union {
    char Name [20];
    Tmem_format mfield;
    Tbra_format ffield;
    Topr_format ofield;
    Tfopr_format ffield;
    int Word;
  };
};
```

the instruction class, iclass, is an enumerated type that indicates the type of the operands. The appropriate operand format (one of Name, mfield, bfield, ofield, ffield, Word) is used. The simulation code can then access the parsed instruction by checking the instruction class and referring to the elements of the relevant field. HASE can automatically produce a parser to load in data types which have been defined, e.g. to initialize a memory.

Higher level instruction formats can also be defined as HASE TInstrParam parameters. For example, a simple

```
if (stopping == 0) switch (Instr.OpCode)
{
  case COMPUTE:
      sim_hold( Instr.time, ev );
      break;
  case SEND:
      send_DATAPKT( TO_NET, ''TO_NET'', Instr.Pkt );
      sim_hold( SendTime, ev );     /* 'ev' = 'event' */
      break;
  case RECV:
      sim_wait( ev );
      SIM_GET(DataPacket,pkt,ev);
      sim_hold( RecvTime, ev );
      break;
  case STOP:
      stopping = 1;
      printf( "%s executed STOP instruction\n", sim_name() );
      break;
}
```

FIGURE 4. Sim++ switch statement.

```
$class_decls
    // Headers for extra functions
    int MPI_Send(void *,           /* data buffer */
             int,                  /* number of data elements */
             MPI_Datatype,         /* type of each data element */
             int,                  /* destination of message */
             int,                  /* message tag */
             MPI_Comm);            /* communicator */
    int MPI_Recv(...);
    // Any other function calls in the interface
$class_defs
    // Implementation of the extra functions
$body
    #include ''theactualcode.c''
```

FIGURE 5. Message passing interface code.

language might have **compute**, **send**, **receive** and **stop** constructs. These could be stored in memory and interpreted by a simple processor entity.

The simulation code can perform a *switch* statement on this field to determine the format of the commands. The Sim + + code segment in Figure 4 illustrates this last point.

Externally created code can be linked in with a HASE simulation. This enables the functionality of test programs on the simulated architecture to be used. Typically an interface is defined for the HASE object so the simulation can trap the desired function calls. Example applications include message passing interfaces and low level I/O on a simple computer system.

The basic form of the interface is as shown in Figure 5. In this example, the file *theactualcode.c* is standard MPI code making use of the functions. 'MPL_Send' and 'MPI_Recv'. In the simulation, these call the member functions which can be implemented in terms of the Sim + + primitives. In this way, standard code may be linked in with the simulation and can be used as a realistic workload.

Other ways of implementing this are possible, such as making the functions globally linked in rather than making them methods of the Sim + + object. It is even possible to link in Fortran 77 functions.

6. CODE GENERATION

Sim + + breaks down the simulation into an initialization and an execution phase. For inclusion in the code pertaining to both phases, HASE generates a Sim + + header file called ⟨*ProjectName*⟩*.h*. For the initialization phase, HASE generates the Sim + + initialization file *init.c*; for the execution phase, it generates the Sim + + entity constructors and bodies.

The behavioural specification for each entity at any given level of simulation is provided by the user in the ⟨*entityName*⟩*.sim* files. From these files and the Architecture Description HASE generates the Sim + + code required for the simulation. HASE also generates the makefile for compiling and linking the various component files.

- **The Sim + + body method**. Each entity in the model architecture needs a Sim + + *body* method for the specified level of simulation. If the entity is a compound entity, the default simulation level can be toggled. It is necessary for HASE to know at which level of decomposition the simulation will occur. HASE will then use the Architecture Description and the corresponding set of ⟨*entityName*⟩*.sim* files to generate the appropriate Sim + + code. The body can be constructed and edited off-line (external to HASE), or within the *Design Window* **Edit Body** function.
- **The project header file**. HASE generates a Sim + + header file for the Sim + + program (⟨*ProjectName*⟩*.h*) which contains parameter and event declarations, global constants, entity initialization structures, class definitions and declarations.
- **The initialization file**. HASE generates the Sim + + initialization file for the Sim + + program, called *init.c*, which contains the instances of the entities and allocates and initializes global data. The auxiliary functions for writing states to the trace file also reside in this file.
- **The Sim + + code**. HASE generates the ⟨*EntityName*⟩*.c* file for each type of entity in the Architecture Description based on information extracted from the entities themselves and the user defined ⟨*entityName*⟩*.sim* files.
- **The SMakefile**. The Sim + + makefile *SMakefile* used to compile and link the Sim + + code is also generated by HASE.

Dependency lists and compilation directives are constructed for *init.c*, the additional global functions file *global_ fns.c* (if it exists), and all ⟨*entityName*⟩*.c* files. The link directive to form the executable is also written to SMakefile.

7. SIMULATION

Running a simulation involves the execution of the Sim + + program produced by HASE in conjunction with the user specified ⟨*entityName*⟩*.sim* files.

Menu options under *Build* (Figure 3) within HASE trigger the generation of Sim + + code, compilation, execution of the simulation and reading of the trace file.

A **debug** version of the simulation may also be compiled. This version of the code includes a simple routine which inserts commands to trace the current line number into the ⟨*entityName*⟩*.sim* file prior to compilation. Selecting this option pops up a window displaying the ⟨*entityName*⟩*.sim* file. This allows the program execution to be viewed alongside the animation.

8. DISPLAYING THE RESULTS

HASE provides two tools for viewing the results of a single simulation execution, an **Animator** and a **Timing Diagram**. Both assist in verifying the validity of the Architecture Description.

The **Animator** uses the event sequence held in an event trace file produced during a simulation run (normally the most recent) to provide the user with a visual display of activity in the system. It allows the data flowing between components to be visualized in a variety of ways, e.g. through moving icons showing individual instructions flowing down the stages of a pipeline or changes to the contents of a register bank when an instruction is executed. The important benefit of the animator is that it lets the user check that the model produces correct results. It is also useful as a presentation aid.

Animation is produced *automatically* from the simulation model with no need for the user to write explicit animation code. The simulation primitives for sending messages between components generate the trace information needed by the animator. It is also possible to animate a component's icon by providing different bitmaps for the different states. If a component has a

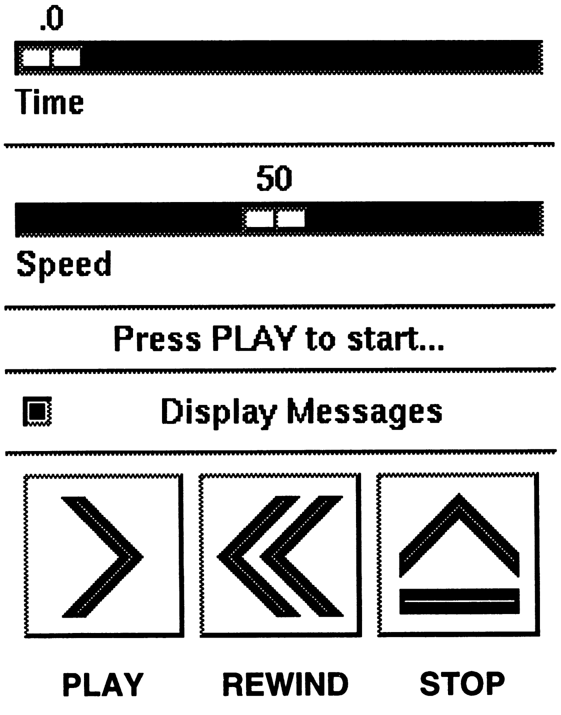


FIGURE 6. The animation controller.

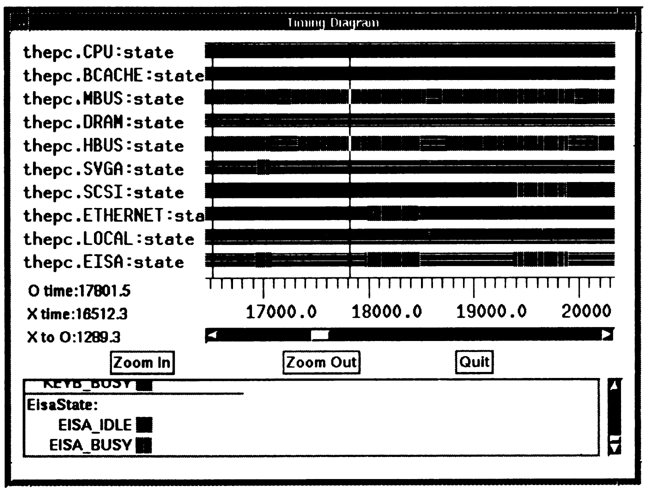


FIGURE 7. Example of a timing diagram.

state defined by the enumerated parameter **BUSY**, **ROUTING**, **IDLE**, animation is achieved by providing X bitmap files *BUSY.btm*, *ROUTING.btm*, *IDLE.btm*. Any number of a component's state variables may be displayed in this way. Variables may be 'dragged' onto the screen display using the component editor (or alternatively they may be left out of the animation altogether). Enumerated variables can be displayed either as the text value, or using bitmaps. The values of integer and string parameters are displayed as a text label. These values are updated whenever the user's simulation code calls the built-in function **dump_state( )**.

Manipulation of the animation of the architecture is handled through an Animation Controller (Figure 6) where time, speed and message display are handled as well as the standard 'tape' functions of PLAY, REWIND and STOP.

The **Timing Diagram** display (Figure 7) shows how the states of the *currently displayed* entities vary over the course of the simulation run. Only the enumerated parameters of each entity are regarded as the state. Different colours/patterns are allocated for each different state. Devotees of project management will recognize the display as a Gantt chart. Time measurements may be taken with two measuring bars, O and X and marked regions can be expanded to show finer detail.

There are additional single run *Data Collection Utilities* available through Sim + + that are not currently integrated with HASE, but still available to the user for manual inclusion in the simulation code.

9. METRICS

In general, the behaviour and validity of the project model are verified by single run results and the performance of the model is observed for subsequent tuning through experimentation with the model.

An experiment comprises repeated simulation runs varying input parameter values to produce output parameters from which performance statistics and other metrics may be gathered. General facilities are provided for monitoring the values of particular state variables but more complex metrics may be obtained by explicitly writing Sim + + code.

Within the HASE environment the architect of the model defines the set of input parameters and also specifies a number of output parameters that could be monitored during the experiment. Users of the model can then determine which input parameters to assign values to in order to make certain observations regarding the performance of the model. The set of input or free parameters is a subset of the parameters of the model, chosen as being either external stimuli or interesting variable factors. The set of output parameters is the results obtained after applying the input parameters to the model. From the set of input parameters, single, sets of or a range of values can be specified.

The experimenter must decide what kind of statistical analysis should be performed on the partial results and view the final results to observe the performance of the model for the defined experimental state. HASE includes facilities for selection from a set of statistical functions and input of the confidence coefficient, interval width and maximum number of iterations. HASE also allows for and manages multiple experiments per model.

10. PROJECTS USING HASE

10.1. The ALAMO project

The ALAMO project (Algorithms, Architectures & Models of Computation: Simulation Experiments in Parallel Systems Design) aims to address the first of the Purdue Grand Challenges [1], 'to identify one "universal" or a small number of "fundamental" models of parallel computation that serve as a natural basis for programming languages and that facilitate high performance hardware implementations'. The project involves an investigation of the use of the H-PRAM model of computation [14] as a bridging model for parallel computation, i.e. an interaction platform for parallel software and hardware, via simulation. Algorithmic skeletons are written in an H-PRAM notation, compiled on to simulation models of parallel architectures created in HASE, and the performance metrics of various hardware architectures investigated. The goal is to determine the properties of cost-effective (cheapest possible) systems based on scalable architectures to provide efficient support of the H-PRAM model.

10.2. Parallel performance prediction

As an approach to satisfying the need for appropriate tools for developing concurrent applications for multiprocessors, HASE has been applied to parallel program development based on the MPI message passing interface. The ease of interfacing software layers to simulation models has made it straightforward to link actual code to models of an architecture. This approach to software development has also been investigated elsewhere using Proteus [20] and the WWT [21]. The advantages of using a simulation model for software development include repeatability, availability, variety, removal of Heisenbugs, ease of visualization and generality. At the design stage of parallel software it is better to have a simple method which is reasonably accurate than an accurate one which is unusable. Because of this, models have been calibrated using an MPI characterization routine which measures the performance of all MPI operations on an architecture. The focus has been on obtaining a quick first-cut design rather than on developing perfect models.

An interesting spin-off benefit of this project is that because Sim + + uses simple co-routines rather than Unix processes, the performance of a parallel MPI program running under HASE can be three to four times better than the same program running under a standard MPI distribution on the same workstation. The absolute improvement depends on the amount of context switching the program causes (since the context switch time for co-routines is faster than for processes).

10.3. An on-line teaching system for computer architecture

This project has produced a demonstration to aid students in the understanding of computer architecture. The demonstration involves playing back a pre-run simulation of the DLX architecture [22] which both animates the diagram of the architecture and displays a sequence of text windows which explain what is happening in the simulation. The simulation deals with hazards, multicycle operations, scoreboarding, etc. There is also a facility to enable students to create and animate their own programs.

11. CONCLUSIONS AND FUTURE DIRECTIONS

This paper has described a flexible environment for computer architects which has the following characteristics:

- **Hierarchy**: each part of a system can be both designed and simulated at the appropriate abstraction level.
- **Flexibility**: no system can anticipate all the needs of potential application areas; HASE has therefore been designed to be as flexible as the most common alternative—writing a simulation from scratch in a programming language.
- **Software support**: a simulation in HASE may involve both the hardware and software aspects of the systems under investigation i.e. HASE facilitates *software/hardware codesign* [23]; this is possible because software libraries can be linked into a simulation model.
- **Component reuse**: a major aim has been to make it easy to construct components which can be used in many different projects.
- **Graphics interface**: The X-Windows/Motif graphical

interface allows the user to view the results of simulation runs through animation of the design drawings.

HASE has already been used for a number of projects and users have commented on the relative ease with which they have been able to create their architectures. Further projects are in progress or are about to start. These include an extension of the on-line teaching system for computer architecture, simulation of a sparse vector processor and investigations of cache performance. Work on multiprocessor systems will include investigations of multiprefix algorithms and dynamic routing algorithms on mesh interconnection networks, and the evaluation of multiprocessor interconnection networks. In this project each of the different networks under investigation will be instantiated in a testbed consisting of a set of processor and/or memory components attached to the network. The processors will be relatively simple models, limited to generating network activity. The output from the various simulation runs will be used to visualize the effects of hotspots, for example, and to produce overall performance measures such as throughput and latency.

A number of possibilities for expanding the capabilities of HASE are also being considered. These include the incorporation of VHDL definitions and formal specifications as additional facets of HASE entities, and the incorporation of a flexible compiling system to allow experimentation with new instruction sets on meaningful example programs. In its simplest form such a compiler would offer modular flexibility in its back-end for generating code targetted at a pre-defined set of instruction sets. The ultimate in flexibility would be a compiler capable of compiling to an arbitrary instruction set, given the specification of that instruction set. Tools to support experimental compilation at some point on the spectrum between these two extremes will be investigated as part of related compiler research currently being undertaken at Edinburgh.

As well as providing a powerful tool for architecture research, HASE is also proving to be a valuable testbed for new ideas in modelling support environments. So far this work has concentrated on adding features to allow experiments involving replicated runs, and thus the exploration of parameter spaces, to be automated [24]. This is proving attractive in increasing the productivity of users, removing the need to repeat runs and collect results manually.

Further use of HASE is required before its run-time performance can be fully assessed, but it seems likely that improvements will be needed. One technique which is already being explored involves exploitation of concurrency mechanisms in the database to deliver results from multiple runs *in parallel* from a network of work-stations [25]. This should increase the speed of the system and allow more extensive simulations of more detailed models to be undertaken.

ACKNOWLEDGEMENTS

HASE is being developed as part of the ALAMO project supported by the UK EPSRC under grant GR/J43295. The authors would like to thank a number of colleagues, especially Paul Coe, Rob Pooley, Peter Thanisch, Nigel Topham and Lawrence Williams, who have provided constructive comments and input to this paper, and also all the students who have used HASE throughout its development for their comments and contributions. F.W.H. has been supported by an EPSRC Postgraduate Studentship.

REFERENCES

[1] Siegel, H. J., Abraham, S. *et al.* (1992) Report of the Purdue Workshop on Grand Challenges in Computer Architecture for the Support of High Performance Computing. *J. Parallel and Distributed Comput.*, **16**, 199–211.
[2] Bell, C. G. and Newell, A. (1971) *Computer Structures: Readings and Examples*. McGraw-Hill, New York.
[3] McHenry, J. T. and Midkiff, S. F. (1994) VHDL modelling for the performance evaluation of multicomputer networks. In *Proc. MASCOTS-94*. IEEE Computer Society Press, New York.
[4] Buck J, Ha, S., Lee, E. A. and Messerschmitt, D. G. (1992) 'Ptolemy: a framework for simulating prototyping heterogeneous systems. *Int. J. Comp. Sim.*, 155–182.
[5] George, A. D. (1993) Simulating microprocessor-based parallel computers using processor libraries. *Simulation*, **60**, 129–134.
[6] Sheehan, K. and Esslinger, M. (1989) *The SES/sim Modelling Language*, Society for Computer Simulation, San Diego, CA.
[7] Evans, D. G. and Morris, D. (1992) Applying modelling to computer systems. In *Proc. IFIP 'CODES Workshop'*, Munich
[8] Nestor, J. A. (1993) Visual register-transfer description of VLSI microarchitectures. *IEEE Trans. VLSI*, **1**, 72–76.
[9] Zeigler, B. P. (1990) Object-Oriented Simulation with Hierarchical, Modular Models. Academic Press, San Diego CA.
[10] Robertson, A. R. and Ibbett, R. N. (1991) Simulation of the MC88000 Microprocessor System on a Transputer Network. In *Proc. EDMCC2*. Springer-Verlag, Berlin.
[11] Robertson, A. R. and Ibbett, R. N. (1994) HASE: a flexible high performance architecture simulator. In *Proc. HICSS-27*, Hawaii.
[12] Birtwistle, G. M. (1985) *Demos: Discrete Event Modelling On Simula*. Prentice-Hall, Englewood Cliffs, NJ.
[13] *Sim + + User Manual* (1992) Jade Simulations International Corp., Calgary, Canada.
[14] *ObjectStore Release 3.0 User Guide* (1993) Object Design Incorporated, Burlington, MA.
[15] Hillston, J. E. (1992) A tool to enhance model exploration. In *Proc. Sixth Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, Edinburgh.
[16] Pooley, R. J. (1991) The integrated modelling support environment a new generation of performance modelling tools. In *Computer Performance Evaluation Modelling Techniques and Tools*, Elsevier Science Publishers, Amsterdam.
[17] Williams, L. M. (1995) *Simulating DASH in HASE*, M.Sc. Dissertation, Department of Computer Science, University of Edinburgh.
[18] Lenoski, D. E., Laudon, J., Joe, T., *et al.* (1992) The

DASH prototype: implementation and performance. In *Proc. 19th Int. Symp. on Computer Architecture*, Queensland, Australia.

[19] Heywood, T. and Ranka, S. (1992) A practical hierarchical model of parallel computation I: the model. *J. Parallel and Distributed Comput.*, **16**, 212–232.

[20] Brewer, E. A. and Weihl, W. E. (1993) Developing parallel applications using high-performance simulation. In *Proc. IEEE Workshop on Parallel and Distributed Debugging*, San Diego, CA.

[21] Burger, D. C. and Wood, D. A. (1995) Accuracy vs. performance in parallel simulation of Interconnection Networks. In *Proc. ACM/IEEE Int. Parallel Processing Symp. (IPPS)*, Santa Barbara, CA.

[22] Hennessy, J. and Patterson, D. (1990) *Computer Architecture, A Quantitative Approach*. Morgan Kaufmann, San Mateo, CA.

[23] Razenblit, J. and Buchenreider, K. (eds) (1995) *Codesign: Computer-aided software/hardware engineering*. IEEE Press, New York

[24] Heywood, P.E., Pooley, R. J. and Thanisch, P. (1995) Object-oriented database technology for simulation experiments. In *Proc. Second United Kingdom Simulation Society Conf.*, North Berwick.

[25] Heywood, P.E., MacKechnie, G., Pooley, R. J. and Thanisch P. (1995) Object-oriented database technology applied to distributed simulation. In *Proc. EUROSIM Congr.*, Vienna..

[26] Lomow, G., Cleary, J. *et al.* (1988) A performance study of time warp. *Distributed Sim.*, **19**, 50–55.