A Multilevel Simulation Study Linking Parallel Applications to Executable and Realisable Hardware Models

Lasse Natvig* and Pauline C. Haddow

Group for Computer Architecture and Design
Department of Computer and Information Systems (IDI)
Norwegian University of Science and Technology (NTNU)
N-7034, Trondheim, Norway, E-mail: {lasse,pauline}@idi.ntnu.no

Abstract: *To improve performance in parallel computers, a major challenge is to reduce the communication bottleneck by decreasing latency and increasing throughput. However, these performance goals must also be traded of against cost which is not, in general, included in traditional performance models. In addition it is important to consider the type of applications that will be run on such a computer by including realistic traffic characteristics in such models.*

In this work we present three models of parallel computers, each of which meets a subset of the above goals. We discuss the advantages and disadvantages of each of these along with an analysis of the effect of a combination of all three. The first, BSPlab is a simulation environment created to study parallel applications, written in the Bulk Synchronous Parallel programming style, on a variety of parallel computers. The second is a behavioural simulation of a specific parallel architecture at the system-level, providing performance results for a variety of traffic patterns. The third approach is a generalised model of the hardware level in a parallel computer's communication system. It's aim is to provide performance and cost results at the gate-level.

Keywords: Performance modelling, simulation, executable hardware models, HW/SW codesign, Verilog, VHDL, PDL.

1. Introduction and Motivation

The art of both software and hardware modelling over the years has led to the development of more and more complex or detailed models. However the question is, is a complex model a better model? In architectural terms, the level of complexity of a model is said to increase as we move from system-level to gate-level. A model may represent a complete system or just part of it, in such a way as to meet the goals of the model. Hardware Description Languages (HDLs) provide us with the freedom to model systems at various levels of detail.

Simulators may be written in HDLs to model large complex designs at the system-level. Computer simulation of simplified models and prototypes of these complex systems is an established technique to unveil design errors at an early stage, as well as improving the co-operation with the end user during development. HDLs such as VHDL and Verilog, provide simulation as an integrated part of the language.

The most common use of HDLs is the development of hardware designs described at the RTL level (Register Transfer Level). The description may be structural — an interconnection of a set of components, or behavioral — an algorithmic style description. Both of these descriptions are technology

* This work was partly supported by the Norwegian Research Council, and was done while the author was at Nordic VLSI, Trondheim.

independant. If a hardware implementation i sought, the code is restricted to the subset of the language implemented in the logic synthesis tool chosen. The process of logic synthesis maps the code level description into hardware patterns within the technology chosen for synthesis. The following steps in the design process can often be completed with the aid of CAD tools which adheres to the design rules of the chosen technology. The architecture chosen for our study is the Torus Routing Chip (TRC), a multicomputer router, designed by Dally and Seitz [DS86]. A multicomputer router offers a sufficiently complex architecture providing a reasonable case study for our modelling techniques. The asynchronous nature of this architecture also gives an interesting element to this work.

This paper presents two modelling approaches using HDLs — the first using the simulation features of HDL to provide system-level modelling and the second focusing on logic synthesis to provide gate-level modelling. Both approaches model the same architecture, a multicomputer router. The first can be said to be an algorithmic or behavioural approach to modelling the functionality of the router chips and the overall performance of the interconnection network at the system level. The second is a structural approach based on a collection of modular building blocks. It models the same functionality, but is closer to the actual hardware, and hence gives more accurate figures for latency.

We also present *BSPlab* which is a simulation environment created to study parallel applications on a variety of parallel architectures — following the Bulk Synchronous Parallel programming style [DU97]. This is followed by a discussion of how these three simulations may be linked together, and how they may benefit from each other. The paper ends by discussing the strengths and weaknesses of the individual and combined models.

2. The Torus Routing Chip (TRC)

Traditionally, parallel computers have been categorised into either non-shared distributed memory computers based on message passing or those based on shared memory. Nowadays, shared memory machines are more and more often distributed and use some form of message passing, a crucial component regarding performance. It is quite common to use a dedicated processor or a dedicated routing chip e.g. the TRC, that reduces the load on the local processor and thus improves performance.

The TRC provides routing of packets along two dimensions called x and y in this paper. Every TRC is connected to a processor. The processors are connected through the TRCs in an arbitrary *k*-ary *n*-cube (torus) interconnection network [Hwa93]. In this work we concentrate on a 2-ary *n*-cube i.e. a two dimensional mesh structure with *n* by *n* nodes and wrap-around connections. Simulations have been undertaken for a 4 x 4 system with 16 processors and thus 16 TRC chips. The simulation at the system level is parametrised allowing for variation in network size.

A TRC receives outgoing packets from and forwards packets to its local processor. In addition, it will transfer incoming packets destined for other nodes to neighbouring nodes. A *packet* is a sequence of bytes containing the *relative address* followed by a sequence of non-zero data bytes and is terminated by a tail-byte (zero). In this paper, we often use the term *flit (flow control digit)* to denote a byte in a packet. In general, a flit need not to be the same as one byte [Hwa93].

The relative address is adjusted in each TRC a packet traverses on its way to the destination. In a two-dimensional mesh, the TRCs are given a unique (x, y) address. A TRC sends a packet along the x-dimension to the neighbouring TRC with a higher x-address. The nodes with the highest x-addresses are connected to the TRCs with the same y-address but zero x-address, i.e. wrap around connections. The y-dimension has a similar strcuture. This structure is shown in Figure 1. The x and y connections are 12 bits wide. Each consists of an eight bit channel and two pairs of request (REQ) / acknowledge (ACK) lines used to control the two virtual channels. To simplify the figure, only the data-transport direction is shown. The ACK lines oppose the data direction.

There are two uni-directional connections to the processor, one for each direction. These are 10 bits wide, 8 bits of data and one pair of REQ/ACK control lines. The structure is regular and may easily be changed to other *2*-ary *n*-cubes. It is also possible to vary the number of TRCs in the x dimension with respect to those in the y-direction. The maximum size of the structure is limited by the fact that the relative x and y-addresses are stored in a byte. Thus we may have up to 256 x 256 processors connected in a TRC network.

The TRC uses the *wormhole routing* switching technique where, as long as the transmission route is free, flits are forwarded towards their destination in a pipelined or "flit-train" fashion. The main benefit over store and forward routing is reduced latency as a flit may be transferred to the next node as soon as the channel is available rather than having to wait until all flits have arrived at the present node. A deterministic routing scheme based on the relative x and y address, termed x-y routing, is used. As long as the relative x-address is non-zero, it is decremented and the packet is forwarded to the next TRC-chip along the x-dimension. When the relative-x-address is zero the packet is routed similarly along the y-connections (if it has not reached its final destination). *Typically* many packets will coexist in the structure. If a TRC discovers that a required channel is being used by another packet, it blocks the in-coming packet until the channel is free.

Blocking of packets, combined with a simple deterministic x-y-routing scheme could give rise to dead-lock caused by a channel dependency cycle (circular wait situation). The TRC uses virtual channels to avoid this problem. Each physical channel provides two virtual channels (VCs), referred to as VC0 and VC1 in this paper. The VCs prevent deadlock by converting possible channel dependency cycles into spirals [DS86]. For a single physical channel, both the virtual channels may transmit data concurrently multiplexed on a flit basis.

Figure 1 shows a 4 by 4 mesh of TRC modules connected along the x and y dimensions. The connection between a TRC and its local processor is not shown. It should be noted that the connections, termed channels, are unidirectional.

3. A multilevel modelling approach; simulations at three different levels

There is much disagreement in the HDL community as to the choice between VHDL and Verilog. A comparison of these languages and C may be found in a paper by Douglas Smith [Smi95]. However, what is important for this work is that HDL languages contain the features required to meet the models' requirements. Therefore we chose to implement each model in the language currently available in the organisations of the two developers — Verilog for the behavioural model (Section 3.1) and VHDL for the synthesisable structural model (Section 3.2). The BSPlab (Section 3.3) environment was implemented in Visual C++.

3.1 Architecture simulation of the TRC

A system level Verilog simulation model of the interconnection network of TRCs and their associated processors has been developed. The model and the experience gained with using Verilog at the system level was reported at the 6'th International Verilog HDL Conference [Nat97].

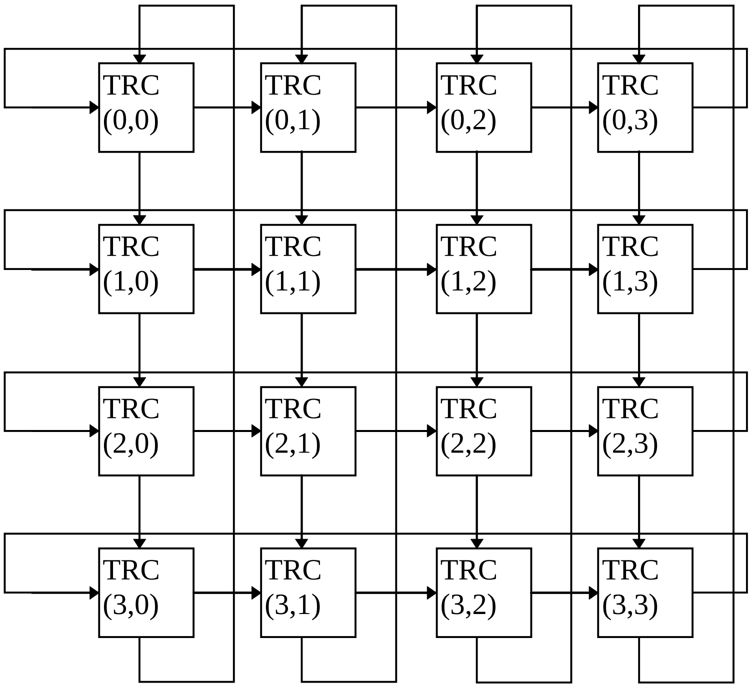


***Figure 1**: 4 x 4 mesh of TRC modules with (y,x) numbering*

The Verilog model provides an environment to test the functionality and performance of the TRC interconnection network. The top level module, system, defines the TRCs, processors and channels (or wires) of the system. Both control and data lines are defined at this level. The processors contain tasks for sending and receiving packets. It is also an ideal place for locating test programs that provide data stimulus (traffic) to the mesh of TRCs and for system performance data gathering. A detailed model of the processor connected to each TRC-chip is not included as the interconnection network and the TRC itself is the focus of the model.

The central part of the TRC chip model is three concurrent processes, each listening on one of the three input channels x-in, y-in and proc-in. Each process describes the functionality within the TRC which handles both the receipt of data, routing decisions and forwarding of the data either to the local processor or towards the destination i.e. transmission to the relevant neighbouring node. The pseudocode in Figure 2 on the next page describes the processing of flits arriving at the x-input channel (task process_xin). It should be noted that the value of the relative x-address denotes the number of hops remaining in the x-dimension and similarly for the relative y-address. When a packet is sent from a processor, both the relative x and y addresses are included in the packet even if one or both of them are zero. The processing of incoming flits on the y-input is similar, except that the relative x-address has been stripped away so that the packet starts with the relative y-address. The relative x and y-addresses of a packet are calculated by the sending processor. As such, a packet arriving at the processor input of a TRC can be handled in the same way as a packet arriving on the x-input.

The model also handles the self-timed control signals, blocking and resuming of transmission and implementation of the virtual channels. More details can be found in [Nat97].

3.2 Modelling a TRC using synthesised HDL building blocks

A generalised hierarchical framework for modelling a variety of router chips and their interconnection networks is under development. Within this framework it is chosen to represent a router as an interconnection of a set of modules. To limit the number of modules required, a module is not necessarily a single router component but may be made up of a number of components. As such, the hierarchy consists of the interconnection level, router level and the router modules themselves.

To provide flexibility in the model, many router features — referred to as decision issues, are provided as parameters to the model. Static parameters provide information regarding design choices made and dynamic parameters allow traffic patterns to be fed into the model.

Each module provides a frame into which a gate-level design is included. The frame includes a list of those parameters which effect the design of the particular module and mathematical expressions which, when executed, provide performance data from the module to the next higher level in the hierarchy.

```
wait for new flit on x_input
read relative_x_address from x_input

if relative_x_address is zero then
  strip it // i.e. do not pass it
  wait for relative_y_address on x_input

  if relative_y_address is zero then
    strip it // i.e. do not pass it

    wait for and read flit-data from x-input
    send flit_data to own processor

    while (flit_data <> zero) do
      wait for and read flit_data from x-input
      send flit_data to own processor

  else // relative_y_address is not zero
    decrement relative_y_address
    send relative_y_address along y_dimension

    pass all flit_data in the packet arriving
      on x-input along the y-dimension until
      the tail byte has been sent

else // relative_x_address is not zero
  decrement relative _x_address
  send relative_x_address along x-dimension

  wait for relative y-address (may be zero) on
    the x-input and pass it along the x-dimension

  pass all flit_data in the packet arriving at
    the x-input along the x-dimension until
    the tail byte has been sent
```

Figure 2: *Pseudocode for task process_xin*

Performance analysis equations consist of expressions for the three key performance features: latency, throughput and cost. Mathematical expressions to support these features depend on a number of parameters given to the model and as such, a selection of expressions are available to support different router issues. Technology data, expressing technology features, also provides the model with the ability to conduct cost analysis.

A modular representation of the TRC is described in the Performance Description Language (PDL) developed by Vemuri et.al. at the University of Cincinatti [VMM96]. This describes both the modules required and their interconnection. Each module is described in VHDL and synthesised to a gate level design. The resulting net-list is then simulated to test for correct behaviour. To enable the program to understand the structure and behaviour of the module the net-list is converted into PDL net-list by a perl program.

To be able to develop a *general* framework, a survey of a number of router designs has been undertaken to build up the set of router modules. These routers were chosen to represent both deterministic and adaptive, synchronous and asynchronous designs. It is assumed, that only point-to-point communication takes place in the network and therefore all multi-cast and broadcast features of these routers are ignored, at present. The resulting set of 19 modules, as well as further description of the PDL modelling, is described in [Had97a]. The router modules required to describe the TRC router are as follows:

AC The *address comparator* module represents either a zero checker, as in this case where relative addressing is implemented or an address checker in the case of absolute addressing. It is often placed at the input of a router where it checks if the incoming message is destined for the local processor or not. The complexity of an AC depends on the length of the word to be compared. In this case, it is the length of an address byte or in other words the flit size.

AD The *address decrementer* is used in deterministic routers with relative addressing to decrement the relative address. Implementation of an AD is often achieved by some form of adder. The size of the adder is dependent on the flit size, where a flit holds the relative address in the current direction. As such, the design of the decrementer AD is parametrised by the flit size.

AR The purpose of an *arbiter* is to make a decision about the assignment of a shared resource. This decision is based on both the arbitration policy or method and the routing algorithm. The arbitration policy describes the priority of the competing inputs and the routing algorithm limits the choice of outputs for a given input. The delay through the arbiter is due to both the amount of traffic competing, the ease of winning (arbitration method) and the choice of outputs (routing algorithm). The complexity of the implementation is also dependent on whether control of the router is synchronous or asynchronous.

CB A *crossbar* unit handles part or all of the switching functionality of a router with the aid of an arbiter. In general, for an n x n crossbar where there are n inputs and n outputs, $2n$ buses are required for the input and output data and n^2 crossing points for switching between these buses. Therefore, implementation is dependent on the number of inputs to the crossbar, equivalent to the number of data paths traversing it and the size of the arbiter itself.

RD The *routing decision* module includes a routing table which implements the routing algorithm. For deterministic routing, generation of the output channel is based on information regarding the input channel and the destination address. The complexity is determined by the routing algorithm and the number of outputs, or in other words the network degree.

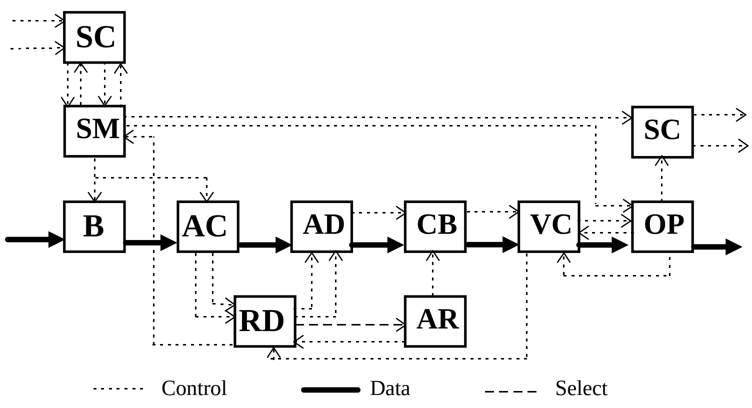


Figure 3: *Router-level description of a path through the TRC using generic router modules.*

SC The *signal converter* handles conversion between two different signalling conventions. In the case of the TRC, 2-phase signalling is used between neighbouring routers and 4-phase signalling within the router. The complexity of the module is therefore dependent on the number of signal lines to be converted and the technology of implementation.

SM The *signal manager* controls the flow of data between two neighbouring routers by request/ acknowledgement lines or control codes. A sender must have control of the channel, the receiver must be ready to receive and in the case of virtual channels, a virtual channel must have won use of the channel before a request can be raised and data sent. In this case req/ack signalling is implemented.

VC The *Virtual channel controller* includes both queuing for and multiplexing between the virtual channels of a physical channel. In the case of virtual channels it is more common to use output buffering than input buffering so that data destined for a particular virtual channel is not blocked behind data for other virtual channels on the same or different physical channels. The parameters to this component are therefore the number of virtual channels per physical channel, the amount of output buffering provided and the arbitration method to arbitrate between the virtual channel queues.

Figure 3 illustrates the structural description given in the PDL code to connect these modules in a single path through the Torus Routing Chip. Similar paths are required for each of the input channels.

3.3 A short introduction to BSPlab

BSPlab is an environment for experimenting with BSP programs on different computer architectures. Parallel applications are written in Visual C++ using the BSPlib standard [BSP96] for communication and synchronisation. The BSPlib is the result of the standardisation effort by the BSP World Wide organisation (`http://www.bsp-worldwide.org/`). An introduction to the BSP model of computing by Bill McColl can be found in [McC95].

In BSPlab, the user may select among various predefined computer architectures. Currently, these are multiprocessors with shared memory or distributed shared memory, network of workstations, and multicomputers with message passing organised in various network topologies. The user may also define her own architectures. The BSP programs are then debugged and executed in BSPlab to achieve (simulated) performance measures and hopefully a better understanding of the interplay between the selected BSP-architecture and the executed BSP-program. BSPlab is primarily an environment for studying BSP computations focusing on the impact of algorithms and architectures on BSP program performance. However, it can also be used as a programming environment for BSP applications developed for real parallel computers running BSPlib.

BSPlab was developed as the diploma work of Haakon Dybdahl and Ivan Uthus from August 1996 to February 1997. A web-site, "`http://www.idi.ntnu.no/bsplab`", is currently being developed to offer the BSPlab to interested researchers.

One of the architectures available in BSPlab is a multicomputer where the nodes are interconnected in a 2-dimensional torus. Several alternative strategies for message passing are supported. This makes it possible to run different parallel BSP applications on a parallel architecture with processor nodes that are interconnected with routers such as the Torus Routing Chip. Such applications will be used to provide the lower level simulation models with realistic message traffic patterns.

4. Linking the simulations together, advantages and status

4.1 Advantages

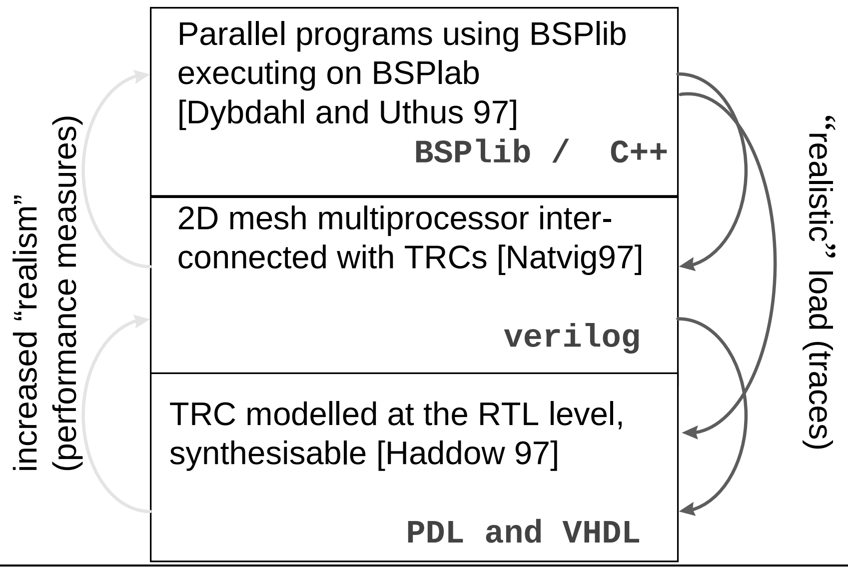
There is a classical trade-off between model complexity and performance accuracy. In general, the performance models used in BSPlab rely heavily on abstraction to avoid becoming too complex and thus the accuracy of the performance data is reduced. However, we believe that the lower level models, such as those described in Section 3.1 and 3.2 may be used to calibrate the abstract performance models used in BSPlab. On the other hand, we believe that these models will (directly or indirectly) benefit from using realistic traffic patterns from traces generated by BSPlab. Thus, an integrated use of the three simulation models should give rise to improved performance results by providing more realistic load (top-down) and by providing performance measures from models closer to the actual hardware (bottom up). This is summarised in Figure 4.

4.2 Status

4.2.1 BSPlab

The BSPlab is up and running. We have collected several parallel applications, *e.g.* sorting and numerical linear algebra [Bis96], that have been tested on various BSP architectures. A lot of performance data is, by default, logged by BSPlab but the selection of data to be logged can easily be modified by the user. What remains to be implemented to integrate BSPlab with the other simulation models is to decide upon a simple format for outputting the message traces that later can be read by the architectural simulation. This is a straightforward task, estimated to take only a few working days.

Figure 4: *Advantages of coupling the three simulation models.*



4.2.2 The Verilog-model

The architectural model of the TRC written in behavioural Verilog simulates the behaviour of each individual TRC within the structure of the network. Tests have been made for correct behaviour, that is that the router responds to incoming messages in the manner described in the specification of the router. The tests are typically written as a separate task (somewhat similar to a procedure) that is included in the Verilog model of the processors. The same task is thus executed by all processors. However, the test programs typically use conditional statements and the unique processor number known by every processor *instance* to control which processor(s) should send the given packet(s) at the given time. This corresponds to the SPMD (Single Program Multiple Data) programming style.

Each TRC has three input channels and three output channels being handled concurrently, with two virtual channels on each of the x- and y-channels. The processors may also send and receive packets concurrently. With 16 processors and TRCs we have close to 200 parallel activities in the system being debugged. In such a system it is important to start the debugging with simple tests to avoid being drowned in debugging information. The first set of tests typically transmitted a single packet between a given pair of TRCs. The next step was to send a single packet to a random destination, and let the receiver forward the message to another random destination. This "random walk test" of a single packet for a large number of iterations proved to be a good test of the routing algorithm since it identified many errors. The correctness of the routing implementation was automatically checked by the inclusion (in the debug version) of the destination-address in every packet being sent. Correct receipt at a destination is then easily verified by checking of the addresses received.

To test that the virtual channels prevented deadlock it was necessary to let all the processors send and receive packets concurrently in a random manner. Before the virtual channels were properly implemented, such tests produced deadlock situations. The current version of the model has been successfully tested by letting each of the 16 processors send 1000 packets of random length concurrently at random times.

A small set of test cases to be used to test the behaviour of a single TRC have been specified. Similar test cases will be used in the VHDL model such that identical behaviour of the two implementations may be ascertained. Creation of a Verilog task to be included in the processor model for reading the tracefile produced by BSPlab has, as yet, not been undertaken.

4.2.3 The VHDL & PDL model

The network module, generic router module and associated generic modules have been implemented in the Performance Description Language (PDL) [VMM96]. PDL is designed specifically for generic performance modelling enabling design instances to be compiled and analysed.

The individual modules are being implemented in *synthesisable VHDL* code. Synthesisable VHDL code is chosen since the code may be synthesised into more than one technology and may easily be adapted to reflect simple design changes. Therefore, it provides flexibility in design, a goal of this work. Each net-list generated from logic synthesis of VHDL code is then converted into a PDL-compatible net-list. These net-lists will provide a library of component designs available to the PDL model.

The synthesisable VHDL model for each generic router is an example of a *virtual component*. It is a non-physical entity (described in HDL), but may be automatically transformed into a physical component by CAD tools. Synthesis tools support only a subset of the HDL language and it is in general significantly more time-consuming and challenging to write HDL code when logic synthesis is required. However, the main advantage is the ability to automatically produce hardware. Synthesisable designs are based on RTL which is inherently a synchronous design methodology [Rus95]. As such, it does not preclude the logic synthesis of asynchronous designs but makes the task somewhat more complicated. However, since this approach first breaks the design into modules for synthesis, both creation and testing of the design is more manageable.

Efficiency of the RTL-code will of course effect the synthesised results, however in this work, the aim is to develop and test a useful set of generic virtual components for router design, not an exercise in making optimised HDL code.

An HDL simulator is used to check correct behaviour of the generic modules. In addition, the flow of signals through the TRC module itself is tested within PDL.

The aim of this model is not to represent an exact copy of the TRC but a significantly accurate copy such that cost and performance data at the gate level may be obtained. As such, assumptions have been made regarding implementation details which are unavailable where those assumptions do not affect the behavioural characteristics and principle design issues of the TRC.

We are currently working on extending the VHDL model to read trace files produced by either the BSPlab or Verilog models, network data or single router data respectively. The current implementation is restricted to unloaded networks. In addition, the Verilog model is being adapted to accept the calibration figures produced by the VHDL model. Finally at the top level, this will be very useful for calibrating the architecture specifications of a 2D torus used in BSPlab (Figure 4). The interaction of these three models will therefore improve the realism at each level of the modelling process.

5. Discussion

Both simulation of parallel applications on the BSPlab and the architectural Verilog model of the system of 16 TRCs have been extensively tested. The synthesisable VHDL generic router modules have also been tested, and currently testing of the TRC router module is in progress. The work on the models will continue, and we expect that experience with the integrated use of the three models will result in adjustments (calibrating) at all three levels. Although this will give new insight, we feel that it is possible to give *preliminary judgements* of weaknesses and strengths of the models. This is summarised in the table below.

In the table, the short names provided in the headings are used for ease of reference to describe the three models. Under the heading *"combined"* we have listed our expectations about the integrated use of the three models *after* calibration. The judgement of each single model is *before* eventual calibrations.

In this context, it should be noted that the Verilog model and the VHDL/PDL model have different goals;

> *The high level model in Verilog*: Behavioural simulation enabling system level performance results to be obtained for different traffic patterns. The key performance goal being throughput.
>
> *The low level model in VHDL:* Flexible detailed model allowing as near as possible the actual design to be realised with the opportunity to study the effect of design changes on the performance. The key performance goals being latency and cost. Currently latency being limited to an unloaded system.

Throughput may be described as the number of messages that can be transported through the system pr. time unit. It is a typical *system level* performance measure that assumes a *loaded system i.e.* a system being stressed with (realistic) traffic patterns. *Unloaded latency* measures the time it takes to transport a message from source to destination in an *unloaded system* i.e. a system serving only this message.

Remarks regarding the table:
(a) The latency for a message through one TRC is a typical example of a parameter that would benefit from calibration using the results of the lower level model.

(b) An important feature of the PDL model and the generic router modules is that they are parametrised. This implies the possibility of looping through various design alternatives for exploring (parts of) the space of possible designs in search for an optimal solution. It should be noted that BSPlab also is made for exploring the design space (of possible BSP computers), but this feature is not used in this work.

(c) The development time for and the size of BSPlab are greater than the others because it is a much more general system that may be used for other purposes not described in this paper.

(d) Combining the simulation models does not increase complexity as integration is realised by trace files and simple interfaces.

(e) TRCsim models message blocking, virtual channels and resource sharing down to the flit level, and is therefore good at measuring system throughput. The trace files will make it possible to study the effect of various traffic patterns on system throughput.

Model property	BSPlab	TRCsim	PDL	Combination
Abstraction level	application & system architecture	system architecture and TRC functional behaviour	TRC architecture, & design space (b)	application, system, TRC architecture & design space
Model language & size (no of lines)	C++ (≈ 8000)	Verilog (≈ 2000)	VHDL & PDL (≈ 3000)	Those to the left plus perl & shell-programming
Development time	≈ 10 man months (c)	≈ 3 man months	≈ 5 man months (i)	≈ additional 1 MM.
Model complexity	medium/high	medium	medium/high	medium (d)
HW-realisability	unknown (poor?)	medium	very good	good/medium (g)
Model scalability (h)	very good	good	poor (h)	good
Model execution	fast	fast	fast	fast
TRC latency, unloaded system	medium (a)	medium	very good	very good
TRC latency, loaded system	medium (a) (unknown)	medium	currently restricted, difficult	very good
TRC throughput in loaded system	medium/good (f) (unknown)	good (e)	currently restricted, difficult	good
Representative of HW cost	poor	medium/poor	very good	medium/good

(f) It is anticipated that BSPlab will provide realistic performance figures for a loaded system. However, this part of BSPlab has not yet been evaluated.

(g) Although the PDL model is "very good" for realisability in hardware, the combined model is judged as "good" since it aims at modelling a whole system of TRCs where other parts are not modelled at the same HW-realisable level.

(h) With model scalability we mean the possibility of using the model for a range of computing systems of different size in number of processors (and TRCs). The weakness of the PDL model here is due to limitations in the PDL tools currently available.

(i) Asynchronous control particularly within the router has given rise to increased development time in the PDL model. It is believed that a synchronous example would have reduced the development time. In TRCsim the choice of synchronous or asynchronous control does not adversely effect the development time.

References

[Bis96] R. H. Bisseling, Basic Techniques for Numerical Linear Algebra on Bulk Synchronous Parallel Architectures, Preprint 964, Department of Mathematics, Utrecht University, June 1996. To appear in Springer lecture Notes in Computer Science, Vol. 1196.

[BSP96] M. Goudreau, J.M.D. Hill, K. Lang, B. McColl, S.B. Rao, D.C Stefanescu, T. Suel and T. Tsantilas, *A Proposal for the BSP Worldwide Standard Library (preliminary version)*, July 1996. Available from "`http://www.bsp-worldwide.org/`"

[DS86] Dally, William J. and Seitz, Charles L., *The torus routing chip*, Distributed Computing (1986) 1:187-196.

[DU97] Dybdahl, Haakon and Uthus, Ivan, *Simulation of the BSP Model on Different Computer Architectures*, Diploma Thesis, *Department of Computer and Information Systems (IDI), Norwegian University of Science and Technology (NTNU),Trondheim, Norway. February 1997. Partly available from* "`http://www.idi.ntnu.no/bsplab`".

[Had97a] Haddow, Pauline C. , *A Generalisation of Router Chip Design,* In proceedings of 5th Euromicro Workshop on Parallel and Distributed Processing, London, January 1997. Pages 307-313.

[Had97b] Haddow, Pauline, *Modelling Communication in Message Passing, MIMD Computers* (preliminary title), Ph.D. thesis, IDI, NTNU. Work in progress.

[Hwa93] Hwang, Kai, *Advanced Computer Architecture, Parallelism, Scalability, Programmability*. McGraw Hill, 1993

[McC95] McColl, William F., *Bulk Synchronous Parallel Computing*, In Abstract Machine Models for Highly Parallel Computers, Editors J. R. Davy and P. M. Dew, Oxford Science Publications, pages 41-63, 1995.

[Nat97] Lasse Natvig, *High-level Architectural Simulation of the Torus Routing Chip*, In Proceedings of 6'th Int'l Verilog HDL conference (IVC'97) pages 48-55, april 1997, Santa Clara, California.

[Pal96] Palnitkar, Samir, *Verilog HDL: A Guide to Digital Design and Synthesis*, SunSoft Press, Sun Microsystems Inc. 1996.

[Rus95] A. Rushton. *VHDL for Logic Synthesis: An Introductory Guide for Achieving Design Requirements.* McGraw Hill 1995.

[Smi95] Smith, Douglas J, *VHDL & Verilog Compared & Contrasted - Plus Modelled Example Written in VHDL, Verilog and C*, from 33rd Design Automation Conference, 1995.

[VMM96] Ranga Vemuri, Ram Mandayam and Vijay Meduri, *Performance Modeling Using PDL*, IEEE Computer april 1996, pages 44-53. See also "`http://www.ece.uc.edu/~ddel/pdl.html`"