Simulating Vehicular Traffic Flows using the Circal System

Gordon Russell   Alex Cowie   John McInnes

Department of Computer Science

University Of Strathclyde

26 Richmond Street

Glasgow, G1 1XH

Scotland, UK

Martin Bate   George Milne

School of Computer and Information Science

University of South Australia

The Levels

Adelaide

South Australia 5095

Abstract

Modern civil engineers have big problems on their hands; more cars use our road networks every year, leading to ever increasing congestion. Knowing how to change a road network to relieve congestion without spending millions of pounds doing so is a tricky business. Traffic simulators can go some way to helping the engineers know how flows will alter with changes in the network. Such changes could be a new road, or even a simple modification to a signalized junction's phasing. Unfortunately, accurate simulators typically available to engineers are slow to execute, especially if the network being simulated is the size of a city. Real-time simulations are available, running on parallel platforms, but these are costly.

This paper discusses the initial research into traffic simulation on programmable logic devices (PLDs). This promises simulations speeds which are faster than real time, but at a fraction of the cost of simulators based on parallel architecture. Rather that trying to map road networks directly onto the PLDs, we have used a hardware description language, XCircal, to model our hardware components. Each component corresponds to a part of a road network, such as a section of road or a one-lane to two-lane splitter. Using XCircal, full junctions can be modelled without touching the hardware. Results of a simulation are included, and these demonstrate that basic junction behaviour can be replicated in hardware.

1   Introduction

Road network simulation has in the past been a processor-intensive task. This has lead simulation down two separate tracks; microscopic and macroscopic. With microscopic simulation, a limited size of road network can be simulated, producing reasonably accurate information on the speeds and loadings of individual parts of the model. Macroscopic simulation greatly simplifies the road network, producing less accurate information on traffic flow statistics.

The work described here is focused upon microscopically simulating large urban networks (hundreds of junctions) at faster than real-time speeds. This requires orders of magnitude speed increases over any other microscopic simulators. An advantage of this greater speed is that through simulation of the network under various control strategies, traffic lights could be automatically phased to reduce congestion. Such solutions could be generated by the user in a time-scale of minutes.

The final execution platform for the system is the SPACE machine[3], developed at the University of Strathclyde. The SPACE machine is a scalable architecture based on a number of square boards, each comprising a 128 by 128 array of programmable logic cells. Each of these logic cells performs a simple boolean function, and can route data in and out of each of its four sides. All input and output connections from cell on the perimeter of the grid are taken off-board at the appropriate edge. When boards are connected edge to edge, the array of cells on each board are joined seamlessly. By connecting many boards, a vast array of cells can be build up. Since both the physical road network and the hardware are planar, a one-to-one mapping between an area of road and an area of the hardware can be made. This is done by tiling the hardware array with many finite state automata; each FSA corresponding to a particular section of road. If the area of hardware available is large enough to accommodate the road network being simulated, then a hour of real traffic flow could be simulated in around a millisecond.

Trying to implement traffic simulation directly onto the SPACE machine hardware is complicated, since it is difficult to identify problem components in a road network composed of thousands of simple functions. Instead, a software simulation for the simulation model was constructed. This simulator uses a hardware description for each of the components in the network, and allows us to build networks from the descriptions. When errors are detected, debugging can be performed in terms of the descriptions. We have used XCircal[8, 9] as the description language. XCircal has a highly concurrent structure, and has many powerful constructs available in its underlying process algebra. XCircal also allows us to compare boolean implementations of network components with the versions used in the simulator, assisting in verification of the SPACE machine circuits. The only disadvantage in this approach is that XCircal-based simulations are slower than a hardware implementation.

This paper begins with a discussion of traffic simulation requirements, commenting on two high-performance microscopic simulators[6, 7]. This is important, since it is vital that any mistakes made in previous attempts at rapid traffic simulators be avoided in our approach. We then give a general introduction to Circal, and then present the most important building-block in our design method; the road cell. A number of other components are also described, although at a higher level of abstraction. The simulator for Circal-based traffic models is then described. This simulator package supports hierarchical construction techniques, and this is demonstrated by an example design for a four-way junction. Hierarchical design allows us to build a junction our of less complex components, which are themselves formed from primitive components. This junction design can then be reused in other parts of the network. The simulation results for this junction design are then analysed with respect to a benchmark junction (a four-way two-phase intersection) and an average delay per car analysis produced. This was then compared against that produced by a PC-based micro-simulator.

2   Available Hardware Solutions

Microscopic simulations of large traffic networks (hundreds of junctions) at faster than real-time speeds are impossible to perform on currently available uniprocessor systems. A SPACE machine implementation, where each of its logic cells can be considered as a simple processor, allows find-grained parallelism to be used in the network design. In this way, each element of the junction (*e.g.* a roadcell) can be mapped onto a small number of the logic cells, and thus each cell can operate in parallel with all other network components. This type of implementation is unique for traffic network simulators.

Two other research teams have also considered the problems of high-speed simulation of

large traffic networks, one based at the University of Maryland [6], and the other in Edinburgh [7]. Both systems are based on a Connection machine[4] implementation. This architecture performs vector and array operations at high speed, and thus both simulators use arrays to represent road networks. An array for car position allows fast roadway processing, but road junctions are not easily mapped onto arrays. Road junctions are not identical to one another, with variations on approach road width, road gradient, light cycle times, junction capacity, number of arms, number of phases, *etc.*[13]. Junctions are therefore played-down in both simulators, replaced instead with simple links between roadways.

Even the links between roadways are not without problems. In the PARAMICS design, cars transferred between roadways can only be done one array element at a time, and this implies than only one car can move from a particular roadway to another in any one tick. Thus if a roadway represents three lanes, then only one of the lanes can pass a car into another roadway.

The comparative cost of a SPACE machine implementation and a Connection machine version is significant. We believe that a SPACE machine costing fifty thousand pounds would be sufficient for our simulation needs; a Connection machine will set you back many millions of pounds. Even then, the simulators based on the Connection machine struggle to maintain even twice as fast as real-time simulation, while we believe that the SPACE machine can perform orders of magnitude faster than real-time rates.

3   Cellular Automata Model of Traffic Flow

The model of a roadway is similar to that of the cellular automaton [14]. This is an array of identical or related simple "machines", each of whose behaviour is dependent on its own state and that of its near neighbours. The basic cell represents a length (*e.g.* 5 meters) of single carriageway which either contains a vehicle or is empty. Roadways are modelled as a connected series of such cells and vehicles are represented by full cells. The progress of vehicles in the model is governed by a protocol in which a full cell will empty itself if the next cell is empty and an empty cell will fill itself if the preceding cell is full. Junctions are modelled using different types of components which manipulate communications between road cells while respecting the protocol.

3.1   Process Algebras

Process algebras [5, 12] are mathematical formalisms used to represent concurrent systems as a set of processes whose behaviour is co-ordinated by synchronizing events. Each process has

a sort, which is the set of events through which the process communicates with its environment. The process algebras have a small number of operators which act on processes and events: guarding, choice, composition, abstraction and definition.

The guarding operator builds from a given event and process a new process which firstly can perform the guard event and then evolves as the given process.

The composition operator combines two processes to give a process representing the overall behaviour in which the component behaviours are synchronized.

The abstraction operator given a process and a set of events yields a process which is the original behaviour with specified events hidden.

The choice operator combines two guarded processes to give a new process whose behaviour is determined by the occurrence of one or other of the guard events in the process's environment. This is the common form of the choice operator and is referred to as deterministic choice, since the resultant process behaviour is completely determined by its environment. Another non-deterministic form of choice operation is defined in some process algebras, where the behaviour of the resultant process is dependent on events internal to the process. This operation can be used to represent incompletely specified processes, or may arise through the removal of external events through use of the abstraction operation.

The definition operator allows processes to be defined in terms of themselves, permitting descriptions of (potentially infinitely) repeating behaviours.

These operators are used in the definition of processes. There are also a number of laws which enable process definitions involving composition or abstraction to be expanded into equivalent processes which do not involve these operators.

3.2   The Circal Process Algebra

Circal [9, 10, 11] differs from other process algebras in permitting a process to be guarded by a set rather than just a single event. This simplifies the representation of simultaneous events. Circal also allows events to synchronize an arbitrary number of processes, unlike other algebras where synchronization occurs only between pairs of processes. This facilitates modelling of global signals such as timer ticks.

3.3   XCircal – an Implementation of Circal

The Circal algebra has been embedded into a programming framework, resulting in an implementation known as XCircal [8, 9]. This supports typing, control structures and definition of procedures which makes it better suited for handling large applications than the pure Circal

algebra. The ability to define a generic process which can have multiple instances is of particular value in the construction of hierarchical models. XCircal can thus be used to specify, expand and display system behaviours. In the following description the term Circal will be used to refer to both the process algebra and its implementation.

3.4   Use of Circal for Traffic Modelling

As indicated previously, roadways are divided up into discrete sections, *i.e.* roadcells. The evolution of the traffic system is represented as a sequence of 'snapshots' at discrete points in time, signified as ticks of a global clock. This model can be interpreted in various ways. The motion of vehicles could be regarded as continuous in time and sampled at each tick. It could also be viewed as vehicles instantaneously moving forward on each tick but remaining stationary between ticks.

By combining Circal specifications of roadways and junctions, complex road networks can be created. Process are joined together using the composition operator. Some events in the Circal description of the network correspond to the state of individual parts of the road network, such as the presence of a vehicle in a particular section of road, a vehicle's intention to turn left at a junction, or the display of a stop or proceed aspect by traffic signals. Other events may be used purely for communication between processes. These can be abstracted away so that they are unobservable at higher levels of the model hierarchy.

This style of modelling is akin to the so-called level-based style used for describing digital hardware [1], in which a pair of Circal events is used to represent the states of a Boolean logic signal. However in the current application a binary condition is represented by the occurrence or non-occurrence of a single Circal event. The recognition of the non-occurrence requires the presence of a 'carrier' event in the specification; the global timing tick here fulfills this role. This modelling style is extensible to multi-state conditions and provides a greater degree of abstraction than a description in terms of boolean logic components.

A simulation may be effected in Circal by composing together the Circal process which represents the behaviour of the system in question with one or more processes which represent constraints on the events which constitute the system's environment. The resultant process corresponds to the behaviour under the test conditions. In the case of the road network the test environment consists of the flow of vehicles across the boundary of the network, the control of traffic light timing, and vehicle routing at junctions. The signal timings are represented by recursive processes which generate stop or proceed events for the required number of ticks in a cycle and then repeat indefinitely. Other constraints can be imposed by generating processes consisting of a random generated stream of the appropriate types of events, possibly

combined with further processes coordinating the interaction with the relevant components of the network.

4   Model of a Roadcell

The key component of our simulation system is the roadcell. To simplify the design of our model, the roadcell is the only component to embody state. The roadcell can either contain a car unit or contain nothing. Communications between a car cell and other components is achieved via four lines; REQ_IN, ACK_IN, REQ_OUT, and ACK_OUT. All the signals are synchronized with a clock tick. REQ_IN are used when trying to store a car within the cell from the previous component, while REQ_OUT and ACK_OUT are used to pass the car onto the next component. This cell is also described in [2].

If the cell contains a car, then the REQ_OUT signal is generated. If no ACK_OUT event is received simultaneously, then the car is decreed to have moved onto the next component, and the state of the current cell is set to empty. Remember that events propagate instantaneously in Circal.

If the cell contains no car, then a REQ_IN event sets the internal state of the cell to that of containing a car. If the cell does contain a car, then the ACK_IN event occurs on every tick.

This design means that cars units normally travel along roadways with at least a single car space between them and other nearest cars units. If a car on a roadway is stopped for some reason, then a subsequent car will come to rest in the previous cell. When the first car starts to move, the second one will be delayed until there is a single space between it and the first. This action is not unlike natural traffic movement along a road.

In XCircal, a car cell can be described in the form shown in program 1. The format of the description is tabulated with a column per event. This is done for clarity, and it not a requirement of XCircal syntax.

5   Building Blocks

To assist in the construction of complex traffic models, graphical representations are used in place of Circal specifications. These components are each capable of performing only simple actions, but when combined together it is hoped that road elements of varying degrees of complexity can be produced.

The key element of the system is that of the roadcell, whose Circal description was depicted in the previous section. The graphical representation used for this component is shown in

```
Event t,REQ_IN,ACK_IN,REQ_OUT,ACK_OUT
Process Empty,Full
Empty <-  (t  REQ_IN                                  )FULL +
          (t  REQ_IN                       ACK_OUT )FULL +
          (t                               ACK_OUT )EMPTY +
          (t                                         )EMPTY

Full <-    (t           ACK_IN REQ_OUT              )EMPTY +
           (t  REQ_IN ACK_IN REQ_OUT              )EMPTY +
           (t           ACK_IN REQ_OUT ACK_OUT  )FULL +
           (t  REQ_IN ACK_IN REQ_OUT ACK_OUT  )FULL
```

Program 1: XCircal description of a car cell

figure 1. The REQ_IN and ACK_IN lines connect the roadcell to car-providing elements of the model, while REQ_OUT and ACK_OUT act as car-providing signals for the next component in the logical progression of elements (thus cars move from left to right in this case). The TICK line provides a global clock for the road network, and since this signal is consistent over all components in the model, it is not generally drawn.
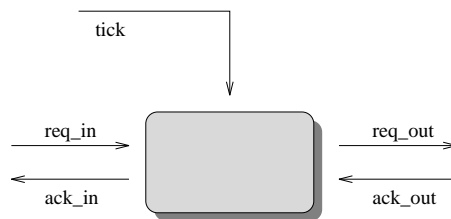

Figure 1: Graphical representation of a roadcell

When modelling a road network, Circal requires a closed system in order to provide deterministic events at the end of each tick cycle. This means that cars must be produced from within the model itself. Currently, a simple source element is provided to meet this criteria, shown in figure 2. This component is defined with a percentage, which is the percentage chance of producing a car on each tick event. Since a roadcell has a maximum throughput of one car per two ticks, a source connected to a roadcell will saturate that cell with cars whenever the percentage equals or exceeds 50%.

The sink component performs the inverse function to that of the source. The graphical representation for the sink is shown in figure 3. This element takes cars as inputs, and destroys
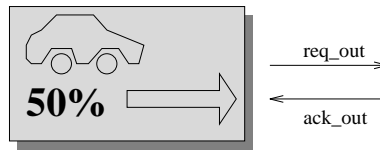
Figure 2: A source of cars into the network

the cars internally. The element is given a percentage probability of excepting a car on each tick, and a car is blocked until accepted.
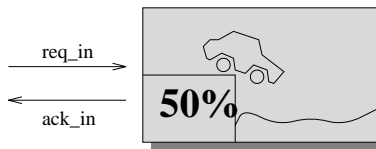
Figure 3: A sink to remove cars from the network

Many more complex constructs required in modelling even simple traffic flow require a mechanism to delay a car's progress through the network. This is provided by the blocking element (figure 4). If the GO event occurs at the same time as a car attempts to pass through a delay block, then the block appears transparent and the car's progress is unhindered. If the GO event is not present, then a car attempting to cross the delay element is blocked.
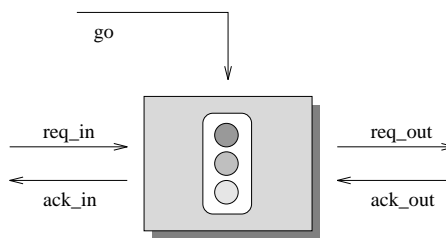
Figure 4: A single traffic light

With the elements described above, only cars which travel along a single route can be modelled. For junctions, it is also necessary to provide a means for cars to move from one road to another. This is provided by the splitter element, shown in figure 5. The inputs to this component can accept a single car at a time, coupled with two decision events; GO1 and GO2. If neither GO1 GO2 occur when a car attempts to pass through the splitter, then the car is blocked. If only GO1 is set, then the input lines are transparently connected to the REQ1_OUT ACK1_OUT part of the component. With only GO2 set, the input lines are connected to the other output port. One exception to this is if either of the output ports have their ACK line set. Under such circumstances any input car is blocked. Thus if a car cell is connected to each of

the outputs of the splitter, then it is guaranteed that only one of the cells can hold a car at any one time. It is illegal for both GO1 and GO2 to be set simultaneously.
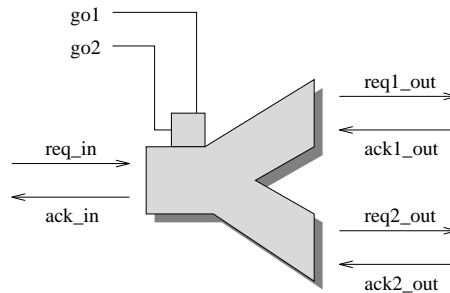

Figure 5: A one-cell to two-cell splitter component

Traffic from two sources of cars can be merged together into a single set of outputs via the arbitration component (figure 6). If a car event occurs at either of the two inputs, then they are transparently connected to the output lines. If both sets of input lines contain a car signal, then the highest numbered port is blocked, and the lowest numbered one connected to the output. If the output port is blocked, then both input ports are also signalled as blocked.
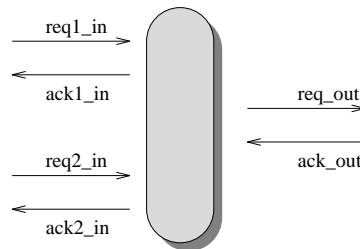

Figure 6: Two input arbitration component

Where traffic is crossing the path of another roadway, such as when turning right on a two-way road, then in reality cars would not attempt to cross until there was a reasonable space available in the oncoming traffic. This type of action can be simulated using the lookahead components, of which there are two (figure 7); lookahead and chained lookahead. Both components have a pair of probe lines, named PROBE and ACCEPT. Only when both these lines do not have an event occurring does the output GO event occur. The chained lookahead has in addition an extra input, named PREVIOUS GO. This allows n lookahead components to be constructed, where the first pair of probes connect to the standard lookahead element, with the remaining lookaheads being of the chained variety, with the previous lookahead's GO line wired into the PREVIOUS GO input.

The components described above form the majority of those used in constructing traffic
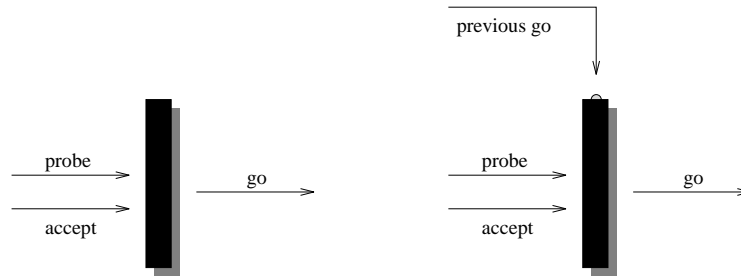
Figure 7: A single lookahead (left) and a chained lookahead component

models. Those which are missing are the ones involved in supplying random or sequenced event chains used for such things as sequencing the lights at signalized crossings, or supplying the probabilities to the splitters indicating which way a car should turn. For simplicity, these components are drawn as a circle containing a textual description.

6   The Simulation Engine

Each component used in constructing the road network has its own Circal definition. Each definition, or process, can be linked (or composed) together in the Circal semantics, allowing the state of each event in the design to be calculated.

The XCircal interpreter composes a road network together by composing pairs of components, producing a single pseudo component as an output. If two processes have n and m numbers of states respectively, then a composition can produce up to nm states in the resultant process. When the entire network has been composed, only one process remains, which contains all possible states of the network. If a network consisted of, say, 500 roadcells, then this final process would consist of 2^{500} states (each roadcell has two possible states). Since the memory required to hold a process is linear in the number of states, this type of composition is practically impossible to perform without running out of memory. In early experiments, networks consisting of a single four-way junction with approach and exit roads of 20 cells used more than 32 MBytes of memory. Increasing the network complexity produced exponential increasing memory requirements. This type of state expansion is necessary for equivalence checks between two processes to be performed, making XCircal a powerful verification tool [9].

In contrast to verification, simulation requires only information on a single state of the process to be present in memory at any particular time; it is only the state reached from the current state which is really of interest. Using this methodology, large memory savings can be achieved. This is the mechanism used by our new XCircal-based simulator, CTrace.

In performing a network simulation, a description of the network is created. Currently, this is done through a simple hierarchical assembly language, which in held in an ndf (network description format) file. Each line in the language refers either to a primitive component or a collection of components whose connections are defined in a separate ndf file. Every primitive component corresponds to a single Circal process. Each process description is then processed individually by circ [8, 9]. This produces a description of all possible states reachable by that component, along with the events acceptable for each state. The circ output is collated by the CTrace program, which composes the processes together. The network is described as a closed system, and thus no events are supplied by the simulator.

If during the simulation, composition of all primitive processes results in more than one possible next state, the simulator is halted and an error message produced. This message includes details of the number of choices left in the final process, and which events are not common to each pair of choices in turn. Since the network is a closed system, any such error must be a failure of the model. If a simulation step results in deadlock, the simulation is halted. Deadlock occurs when no valid next state can be calculated. One component, named *limit*, produces the tick events for a specified number of times, and then produces a deadlock condition. This allows the simulator to be limited to a certain time-span; without this, the simulation would continue infinitely. In general, only the limit component can generate deadlock conditions.

To conserve memory, CTrace uses bit vectors to represent events, where an event corresponds to a single bit at a unique index of an array (vector) of bits. This is used for both the sorts and the state guards in the process descriptions. This makes for efficient composition, where much of the work can be performed via bitwise AND, OR and XOR functions. In current simulations, the total memory requirement of the simulation is approximately 0.7 MBytes for the four-way junction with 20-cell roadways, rising to 1.5 MBytes for a simulation using 100-cell approaches and 3 cell exits.

In performance analysis of the first version of CTrace, almost 70% of the time was spent deciding which processes will be composed next. Incorrect choice in the order of composition can result in a state explosion, even in the single-step approach. This could occur when two processes are composed together, both containing a large number of states, with little (or no) overlap in the sorts. The resulting process could have a number of states equal to the product of the number of states present in both the original processes.

The selection algorithm used in CTrace to select the order of composition attempts to choose the two processes which will create the smallest resultant process, while also attempting to reduce the number of choices left in the process list. The current algorithm selects a

process with the lowest number of choices, and joins it with the process containing the lowest composition cost. The composition cost is derived from an equation developed through experimentation, and is based on three factors; the number of events in the overlap of the sorts, the number of events not in the overlap of the sorts, and the number of choices present in the second process.

To improve the performance, CTrace was revised to use the history of composition order collected during the first tick of the simulation. This does not generate as good a composition order as that obtained by recalculating the order on every tick, but usually comes close. This has improved our benchmark (a four-way junction with 100-cell approach roads) simulation time from 15 hours to approximately three (executed on a SPARC ELC).

Improvements in the performance of the simulation is certainly possible. The current code converts from linked lists of events to bit vectors and back again on every step. The next version of the software will be based entirely on bit vectors.

7   Hierarchical Construction

The ndf assembly language described above permits designers to make use of hierarchical construction techniques. This allows specific junction types to be constructed from the simple components available, with these junctions then being used as components at a higher-level. In this section this design strategy is shown by way of an example.

In order to demonstrate the simulation capabilities of the system, a four-way junction is to be constructed. To place all the junction components in a single ndf file would produce a file which was both difficult to read and difficult to update. Instead, it is decided to use macros to describe some of the components.

Firstly, cars in a four-way junction can either turn left, travel straight through, or turn right. This suggests the need for a three-way splitter. If duel two-way splitters were connected back to back, such a component could be created. This is shown in figure 8. Here, the left-hand exit from the first splitter is connected to the input of the second splitter component. The direction controller selects the left-hand arm of the first splitter whenever the car is not turning right, allowing the second splitter to control left and straight on directions. Three car cells are also needed, though due to the mutual exclusion of the splitters only one of the cells can be occupied at any one time.

Once a car has entered the splitter, the direction of its passage is effectively stored until it is able to leave the component. In this way a car which tries to go one way but is blocked does not try a different direction on the next tick.
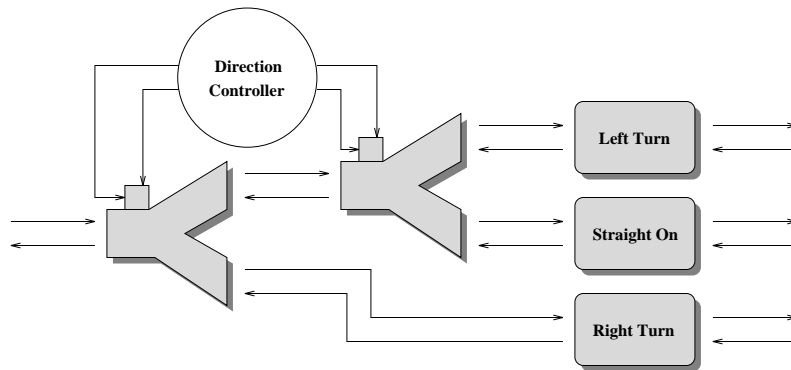
Figure 8: Construction of a three-way splitter

The four-way junction being designed is to be signalized, and so requires traffic light components. So that the direction of car passage through the junction is maintained even when the lights are at red, the traffic lights must occur after the splitter. This means that there must be three sets of lights for each approach road (one for each possible direction). In addition, cars turning right should check for oncoming cars before progressing. Both of these systems are contained within the component shown in figure 9. The actual lookahead signal is generated from the version of this component present on the opposing arm of the junction, and in turn this component generates the lookahead signal for that arm. As such a car waiting to turn across the route of cars using this component should be stopped when a competing car is either turning left or travelling straight on (it was decreed that cars in opposite directions both turning right would turn simultaneously without competing). The lookahead therefore connects to the left and straight on signals from the previous car cells, with the signal accepted only when the lights are at green. We name this macro a *threeway-lights* component.

By connecting the threeway-splitter to the inputs of the threeway-lights, one quarter of a junction is created. By cross-connecting the lookahead lines of four such combinations, a full junction can be routed. This produces three car-lanes on each exit from the junction. To funnel this into one lane, a third macro is created which is the opposite of the threeway splitter; a threeway arbiter. This uses two arbiter components, with the first one connected to one of the inputs of the second. The priorities are maintained such that a car travelling straight across the junction has the highest priority, with a left-turn entry second and the right-turn entry last. This complete four-way junction is shown in figure 10.
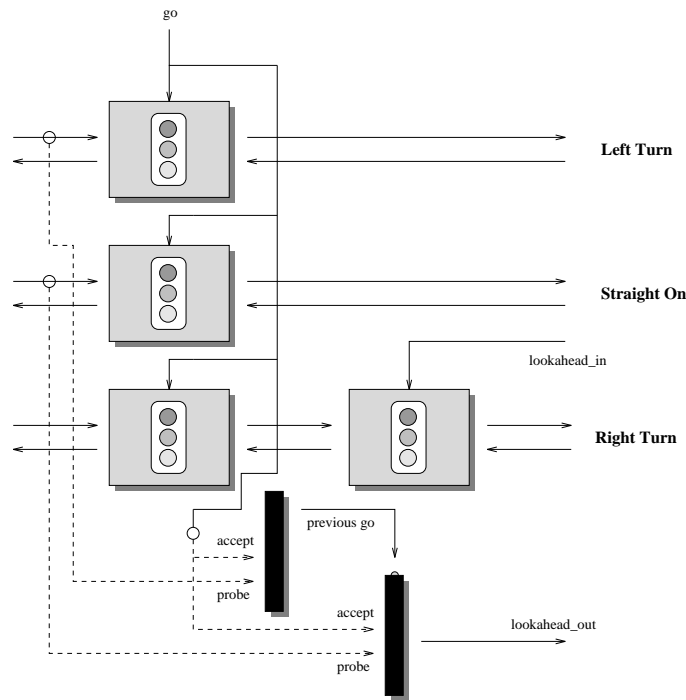
Figure 9: Three-way lights with on-coming traffic detection

8   Initial Results

Before the simulation system can be used to model real junctions, its accuracy with respect to the behaviour of actual physical junction must first be verified. As an initial step towards this, a modified version of the four-way junction described in the previous section was simulated. The junction was simulated for one hour, and the average delay per car unit recorded.

In attempting to match physical junction characteristics to that generated from our simulations, various parameters used in the simulator will have to be controlled. These parameters include the capacity of each roadcell (*e.g.* one roadcell could be equal to one passenger car unit), the correlation of ticks to real-time (*e.g.* each tick could be equal to one second of real-time), and the physical length of each roadcell (*e.g.* five metres of road per roadcell). These parameters are inter-related. The calibration of these (and other) parameters is the subject of a future technical report. Without calibration, the results of the simulation can only be compared to other simulators with respect to the shape of the resulting curve. This analysis used the example values for the parameters listed above, with the sources set to arbitrary values (22% and 42% creation rates, *i.e.* 22% being the probability out of 100 that a car will be produced from each source independently from previous ticks or other sources).

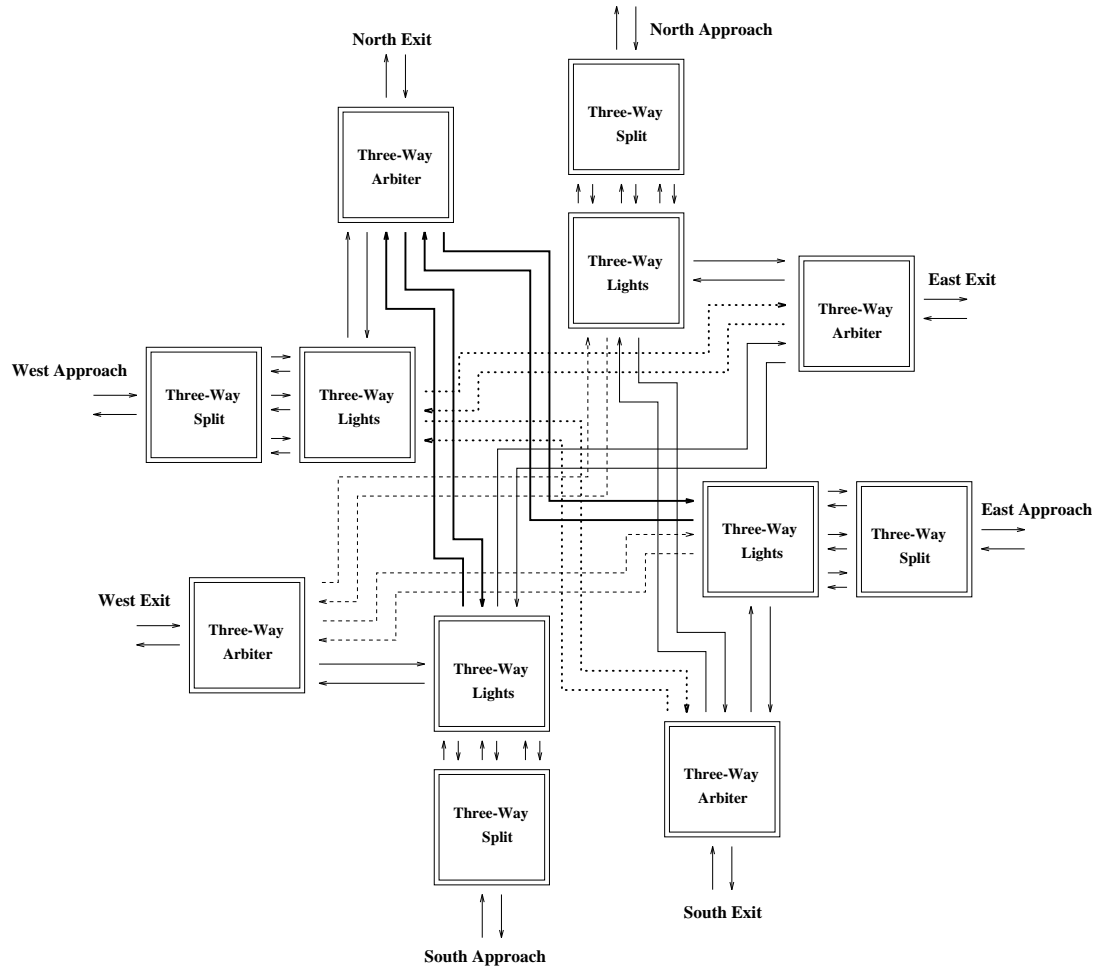The model was modified to use a greater lookahead than that described in the hierarchical

Figure 10: Example of a full four-way junction

construction section. Since cars move along the roads with at least one empty cell between each car, a lookahead of only one cell would indicate that no car was approaching at least one time in two. Instead, a lookahead of two car cells deep was implemented. The results of the simulation using this junction is shown in figure 11. The exit roads used are two cells long, coupled to perfect sinks. A perfect sink will always take cars from the connected roadway, and never blocks. This means that no tailbacks can ever occur on the exit roads, and thus we can use short exit roadways (the shorter the roads, the less components are used in the network, and thus the faster the simulation will run). The approach roads are 100 cells long, to allow space for sizable tailbacks to build.
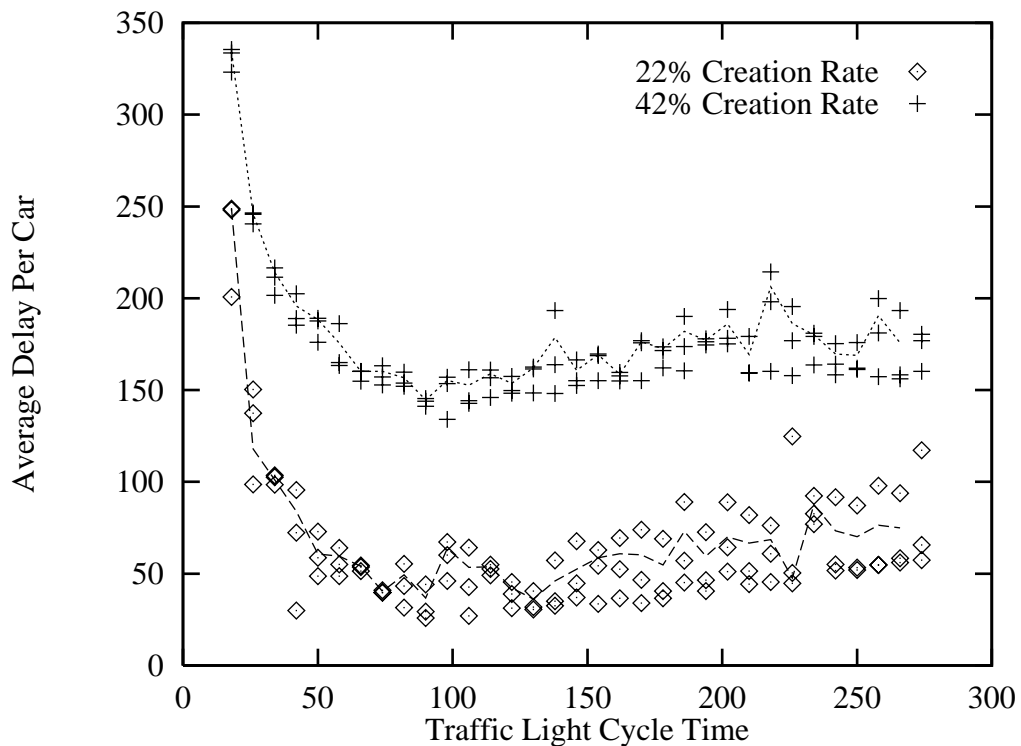

Figure 11: Average Blocking Delay per Car in Four-Way Junction

The top plot of the graph shows the 42% creation rate, with the bottom showing the 22% version. Each source, being probability based, has random characteristics. Thus every cycle time used in the junction was modelled three times to reduce these random effects. Time (both the cycle time and the delay per car) is measured in ticks. The inter-green period[1] was set to 4 seconds per cycle. The shape of the graph is similar to that predicted by theoretical modelling [13], but it is unclear how large a tick really is or if the car units of the y-scale is equivalent to pcu. The calibration of the network is one of the areas to be investigated next.

---

[1]The inter-green period it the time between one phase going red and the next phase going green.

9   Conclusions

The initial results gathered on this modelling approach demonstrate the possibilities of using simple hardware representations to simulate traffic networks. The actual graph displayed above closely correlates in shape to that found in theoretical studies.

The next step in this research is to calibrate the components with respect to theoretical models, including road capacity and tick to real-time conversion. Once this is achieved, a graphical construction tool will be created to aid in further research. This tool is needed to bridge the gap between physical representation of the road network and its logical implementation on the SPACE machine. Without a reasonable mapping between the two levels, simulation time will become swamped by place-and-route overheads, which is an *np* complete problem.

References

[1] Boole. *The Mathematical Analysis of Logic*. Cambridge, 1847.

[2] Paul Cockshott, George McCaskill, and Peter Barrie. Use of a high speed cellular automata machine to simulate road traffic. Technical Report HDV-27-93, University of Strathclyde, May 1993.

[3] Paul Cockshott, Paul Shaw, Peter Barrie, and George J. Milne. Scalable cellular array architecture. *Computing and Control*, 3(5), September 1992.

[4] W. Daniel Hillis. *The Connection Machine*. MIT Press, 1985.

[5] C. A. R. Hoare. *Comminicating Sequencial Processes*. International Series in Computer Science. Prentice Hall, 1985.

[6] Thanavat Junchaya and Gang-Len Chang. Exploring real-time traffic simulation with massively concurrent parallel computing architecture. *Transportation Research - C*, 1(1):57–76, 1993.

[7] Dr David McArthur, G. D. B. Cameron, M. D. White, and B. J. N. Wylie. Paramics: Parallel microscopic traffic simulator. Available from Dave McArthur, EPCC, Room 2412, JCMB, King's Buildings, Edinburgh, January 1994.

[8] G. A. McCaskill. The xcircal user guide and reference manual. Technical Report HDV-18-91, University of Strathclyde, October 1991.

[9] George A. McCaskill and George J. Milne. Hardware description and verification using the circal-system. Technical Report HDV-24-92, University of Strathclyde, June 1992.

[10] G. J. Milne. A calculus for circuit description integration. *The VLSI Journal*, 1(2,3):121–160, 1983.

[11] G. J. Milne and M. Pezzè. Circal:a high level framework for hardware verification. Technical Report HDV-1-88, University of Strathclyde, 1988.

[12] R. Milner. *Comminication and Concurrency*. Prentice Hall, 1989.

[13] R. J. Salter. *Highway Traffic Analysis and Design*. MacMillan Education, second edition, 1990.

[14] Stephen Wolfrom. *Theory and Applications of Cellular Automata*. World Scientific Publishing Co, 1986.