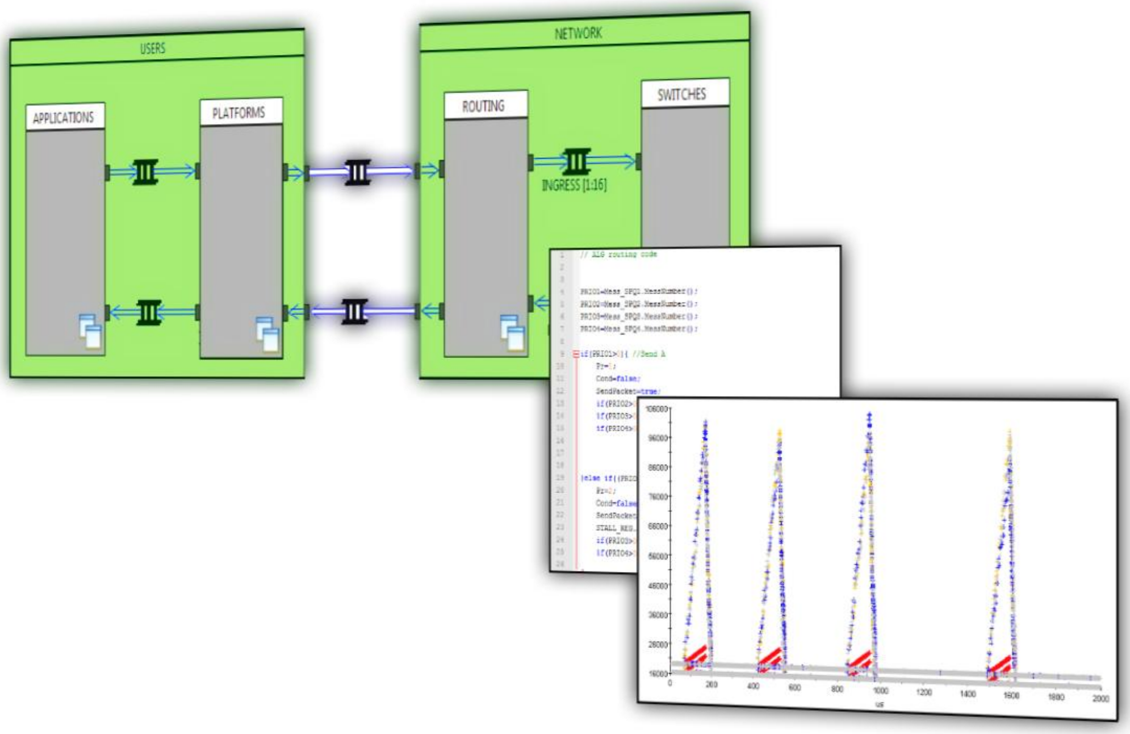


Baseband Inter-Chip Communication

Interface Simulation for Baseband ASIC Multi-Processor Chip

Master of Science Thesis in the department of Computer Science and Engineering

ROBIN ANDERSSON
NICLAS JACOBSSON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, October 2012

Baseband Inter-Chip Communication
Interface Simulation for Baseband ASIC Multi-Processor Chip

R. ANDERSSON,
N. JACOBSSON

Examiner: IOANNIS SOURDIS

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden October 2012

Abstract

In this thesis simulations of a baseband inter-chip communication interface are implemented in CoFluent Studio. This was done to investigate whether CoFluent Studio could be used to aid development process of baseband chips and also improve performance of these systems by using these models. The software models of the system included four Application Specific Integrated Circuits (ASIC) connected by a network consisting of four switches, where each individual model implements a specific scheduling discipline. Comparison between scheduling disciplines was performed. These consisted of strict priority (SP), round robin (RR), time-division multiplexing (TMD) and asynchronous latency guarantee (ALG) scheduling. Another addition to tested scheduling principles was a modification of ALG that was created by the authors which allowed for configuring bandwidth between priorities. This scheduling discipline was called Configurable Asynchronous Latency Guarantee (CALG).

The simulations were conducted using stimuli-data consisting of four different priorities which represents a typical traffic scenario within such baseband chip. The traffic data is transmitted over a Transmission Time Interval (TTI) of 2 ms where the highest priority data, representing time critical data, was sent as bursts while lower priorities were distributed uniformly over the remaining interval.

The results of RR and TDM showed that their properties yielded high priority latencies of 54.7 and 91.7 μs respectively. ALG resulted in 53.4 high priority latency and 19.4, 19.5 and 19.7 μs for the respective BE data. The default SP scheduling yielded results in which high priority data had a Guaranteed Latency (GL) of 19.2 μs while lower priorities made Best Effort (BE) latencies of 118.4, 126.7 and 139.3 μs respectively. By using CALG the GL was relaxed to a limit of 25 μs, which gave rise to lower BE latencies of 100.1, 104.1 and 98.1 μs for the lower priorities.

The results yielded from the CALG shows an latency improvement for low priority data of 18.3, 22.6 and 41.2 μs respectively, while maintaining a GL of 25 μs for high priority data. The original ALG has a 178.1% higher GL compared to the default SP scheduling. The configurable parameter of CALG showed that by relaxing GL to 25 μs its latency became 29.6% higher than the GL of default SP scheduling. This shows that the CALG is more suitable than ALG for bursty data traffic.

Keywords: Guaranteed latency, best effort, QoS, SoC, round robin, time-division multiplexing, ALG, CALG.

Acknowledgements

This work was carried out for the department of Computer Science at Chalmers University of Technology. We wish to express our gratitude to our supervisor Björn Johansson at Ericsson, who has been of great help during the course of this thesis, as well as Fredrik Wartenberg, Stefan Andersson, Urban Bergström and Henrik Carlsson who were the customers in this project and have been very engaged in the development of this thesis. We wish to offer our gratitude to our examiner Ioannis Sourdis who has given us many interesting ideas. We would like to thank Intel for providing us with CoFluent licenses. We also wish to thank Jerome Lemaitre from Intel for his excellent support regarding CoFluent and the design process of this thesis.

Robin Andersson and Niclas Jacobsson

Göteborg, Sweden 2012

Contents

List of Figures

Nomenclature

ALG - Asynchronous latency guarantee

ASIC - Application Specific Integrated Circuit

BP - Backpressure

CM - Common Memory

DSP - Digital Signal Processor

FPGA - Field-Programmable Gate Array

GL - Guaranteed Latency

GPP - General Purpose Processor

HSDPA - High-Speed Downlink Packet Access

MTU - Maximum Transfer Unit

Prio1 - Priority one data

Prio2 - Priority two data

Prio3 - Priority three data

Prio4 - Priority four data

QoS - Quality of Service

RR - Round Robin

SP - Strict priority

SPQ - Static Priority Queue

TDM - Time-division multiplexing

TTI - Transmission Time Interval

UE - User Equipment

VC - Virtual Channel

1

Introduction

Base stations are needed for transmitting and receiving radio signals to and from User Equipment (UE) such as mobile phones. The base station antennas are often placed high on a house rooftop or a radio tower to maximize coverage.

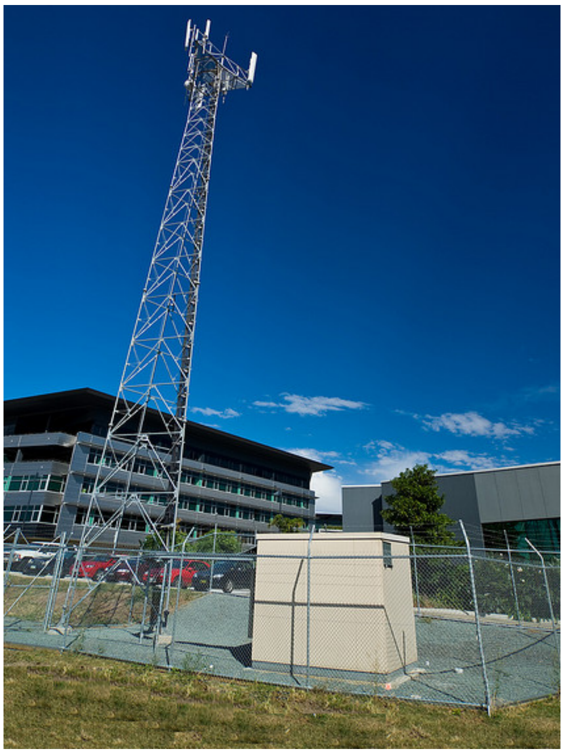


Figure 1.1: High baseband tower including cabinet

The data processing units or baseband cards of the base station are normally placed in a cabinet inside a building or inside a small closure next to the radio tower.

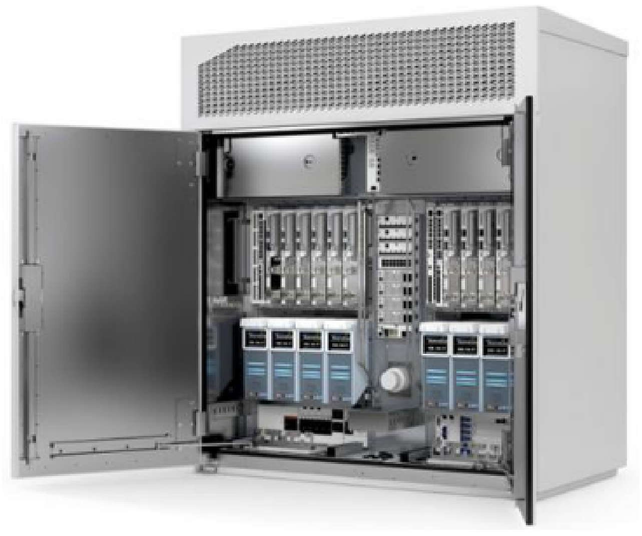


Figure 1.2: Cabinet containing multiple baseband cards

Since the number of mobile subscribers is growing and because of the demand for higher transmission speeds the requirements on the base station is constantly increasing. More users and higher throughput means that the base station will receive and transmit more data through the antennas that needs to be processed. Some of the functions that need to be performed on the receiving (or uplink side) is decoding the data from the air interface protocol, combine data from multiple streams for the same user and check if all data was received correctly. Data that was not received correctly can be requested to be retransmitted. In the transmitting side (the downlink) data is received from the core network and should be forwarded to the UE. In this direction data must be encoded into the air interface protocol so that it can be transmitted over the antenna.

The handling of these signals must be performed by some form of data processing elements on the base station. These elements can be General purpose processors (GPP), Digital Signal Processors (DSP), Application Specific Integrated Circuits (ASIC) or Field-Programmable Gate Arrays (FPGA). One GPP of sufficiently high capacity could perform all the signal processing tasks necessary and then the base station would be easy to construct with only one active processing component. However such a GPP capable of handling the requirements put on today's base stations is not possible to construct with currently available technology. Connecting many GPPs together would solve the processing capacity requirement. However such a solution would be unnecessary expensive, consume too much power and

generate too much heat. Therefore all base stations today are constructed by combining one or many of all mentioned techniques. Also each processing element executing software is in accordance to Moore's Law increasing the number of transistors on the same surface and then not only to make each core more capable but also to increase the number of cores on each element, see figure 1.3 below [1]. Functions that are power and time consuming to do in software are implemented directly into hardware blocks, for example decoding the incoming user data is done by hardware accelerators.
The data processing units or baseband cards of the base station are normally placed in a cabinet inside a building or inside a small closure next to the radio tower.

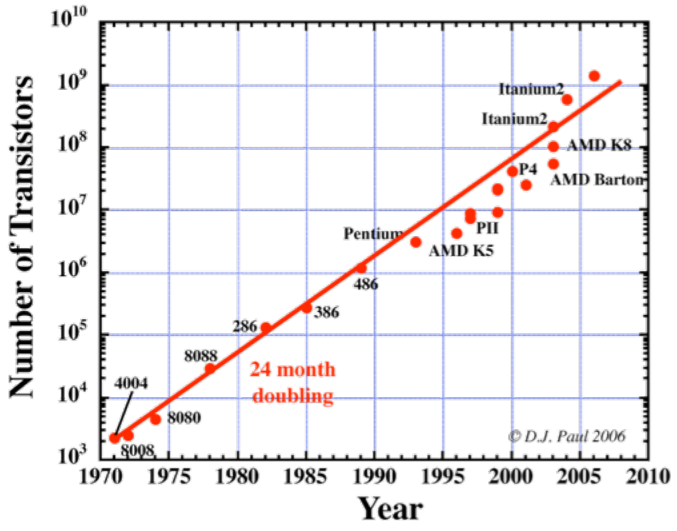


Figure 1.3: Moore's law vs. CPU transistors

Even if the complexity and performance is increased by having multiple cores and hardware accelerators on each chip several processing elements are still needed in each base station. Data must flow to and from the core network via these chips and to and from the antennas. Also the processing elements will need to talk with each other to synchronize and collaborate. In some cases specific hardware accelerators are only present in some of the processing elements and the input data must be moved to that element before the work can be performed. There is also a need to schedule the handling of specific data-flows to some of the elements. These requirements implies that there must be communication channels between the processing elements. Technically there will be copper wires imprinted on the base band card. For each copper wire connecting a pair of terminating entities an agreed upon protocol must exist so that the electrical signals sent on the wire can be understood by the receiver. There exist several industrial standard protocols for such short distance communication. One example is the protocols defined by the Rapid I/O organization. Another example is the PCI family of protocols. Yet another is the Ethernet standard which is also possible to use for communication within an embedded system. The connections between the chips need to be of many-to-many type since potentially every element needs to communicate with any other element. However it would be impractical to connect each chip with every other by dedicated wires. Thus it is necessary to use aggregating type of components capable of allowing multiple elements to use the same wire. A switch is such a component which is capable of receiving data packets on one or several connections and transmitting them out one by one on another connection.

The switch must calculate where to forward each packet and that is normally done by inspecting some part of the data within the packet. In the case of Ethernet packets the destination MAC address in the Ethernet protocol header can be used for this purpose. The switch will be configured with a routing-table mapping destination MAC addresses to outgoing connections. The connections are normally referred to as ports where the incoming connection ports are called ingress ports and the outgoing connection ports are called egress ports.

When designing such an embedded system it is important to know if the available communication properties such as data transport latency and bandwidth is sufficient for the communication requirements of all entities using the communication network. It is important that enough capacity is available but it is also important not to construct a network with too efficient components since that would be unnecessarily expensive and capacity would be left unused. Data flows through the network will have different requirements for latency and bandwidth. Also data flows will exist with different priorities. A low priority flow should interfere as little as possible when a high priority flow is requested. The switch algorithms for how to prioritize flows of different priorities will also impact the overall behaviour of the system.

It is difficult to calculate what impact different flows and scheduling algorithms have in a network of communicating entities. Switches with flows of different priorities and intensities, appearing by what can be described as stochastic processes. Since it is not possible to build the system and see if the requirements where fulfilled, calculations must be done beforehand. The best way to do this is to as accurately as possible simulate the system. It would not be feasible to simulate on component level so a higher abstraction level must be found.

There exist simulation tools on the market today which can simulate interconnected systems on bit level. This thesis work will evaluate the best such tools on the market and also evaluate the impact of different scheduling algorithms within the switching components.

1.1 Purpose

The purpose of this thesis is to investigate and evaluate how software modelling can aid the development of baseband chip communication. The obtained model is further used to investigate possible improvements in Quality of Service (QoS) with respect to latency, see [2] for more information regarding QoS. The model is specified to use SP which yields a GL of 19.2 μs for time critical data. The thesis investigates how BE latencies can be reduced by relaxing the GL requirements to 25 μs.

1.2 Objective

The main objective of the project is to obtain a model which corresponds to the theoretical behaviour of a future system, with respect to latency. The model should have high visibility of parameters, such as memory usage, latency and throughput. Using this model a comparison of scheduling principles will be performed to see if it can improve the overall performance. The thesis will cover Strict Priority scheduling (SP), Round Robin (RR), Time Division Multiplexing (TDM) as well as Asynchronous Latency guarantee (ALG) scheduling proposed in [3] which has been modified for this thesis. The thesis will mostly focus on the ALG scheduling in order to investigate if its adaptiveness could gain performance in the system.

1.3 Scope

Since access to real hardware is unavailable, due to the fact it has not been manufactured yet, the accuracy of the model compared to reality will not be investigated in this thesis. A model will be created in an effort to simulate the hardware. Changes to the scheduling will be made in the model and differences in results will be compared. Making corresponding changes in the real hardware should yield the same differences in results.

1.4 Division of effort

To increase productivity during this thesis work the work load was divided. While both participants were involved in all parts of the thesis, Robin was more involved with defining and implementing the scheduling disciplines whereas Niclas was more involved with routing and generating data traffic used for simulations.

1.5 Thesis outline

The thesis begins by introducing the background for the technical aspects. The background includes the given specifications, which are presented at the beginning of the thesis, these concerns network specifications and traffic characteristics, as well as comparison of different scheduling disciplines.

After background the Method chapter follows, here the conduction of the thesis is introduced. The chapter introduces software tools what were used, how the simulation models were implemented and at last details of how data stimuli is constructed and interpreted during simulation.

Once the background and method have been introduced a chapter for yielded results can be presented. The results chapter starts with an analysis of vitals parts and functions of the conducted models which has been yielded from the modelling software. This is followed by the simulation results for all scheduling disciplines regarding latencies and throughput.

Finally a conclusion chapter will summarise the content of this thesis. Important results are highlighted and discussed here which from these, conclusions are made in order to answer the questions that have been raised in the beginning of the thesis.

1.6 Pre-study phase

This chapter addresses the theoretical research done initial to this thesis. The aim of this section is to describe the foundation for the ideas that evolved this thesis from simply building simulation models to comparing schedule disciplines.

1.6.1 Preparative knowledge

In order to understand the description of the Ethernet switch system model the pre-studies were initialized by reading the book Principles and practices of interconnection networks [4]. This literature gave more insight in details regarding congestion control and deadlock, which are common issues discussed when speaking of real networks with finite capacities. Further investigations regarding scheduling disciplines and algorithms has been done by reading Computer Networking A Top Down Approach [5].

1.6.2 Investigating scheduling disciplines

With the intention of not only building the desired model but also find some ways of improving the performance regarding the traffic load the thesis investigates alternative schedule disciplines for various scenarios. Which scheduling discipline to be used becomes significant since data is sent with different distribution, using a suitable scheduling can minimize congestion created by certain types of data. The network was specified using a strict priority scheduling, this was compared to commonly used disciplines like Round Robin, Time-division Multiplexing. It was also interesting to investigate Asynchronous Latency Guarantee, proposed in [3], which led to the variation of the scheduling discipline used in the thesis.

2

Background

This chapter presents an overview of the system containing switches and the different scheduling disciplines, used to arbitrate data to be sent. The switches specified by Ericsson uses SP scheduling which was implemented in the models, together with RR, TDM and ALG scheduling.

2.1 Specifications

The modelled system is a Ethernet network consisting of four ASIC's connected to four switches with a linkrate of 10 Gbit/s, sending 64 byte packets. According to the specifications given the ASIC's will have a delay of 5 μs between the generation of data and sending it out to the initial switch.

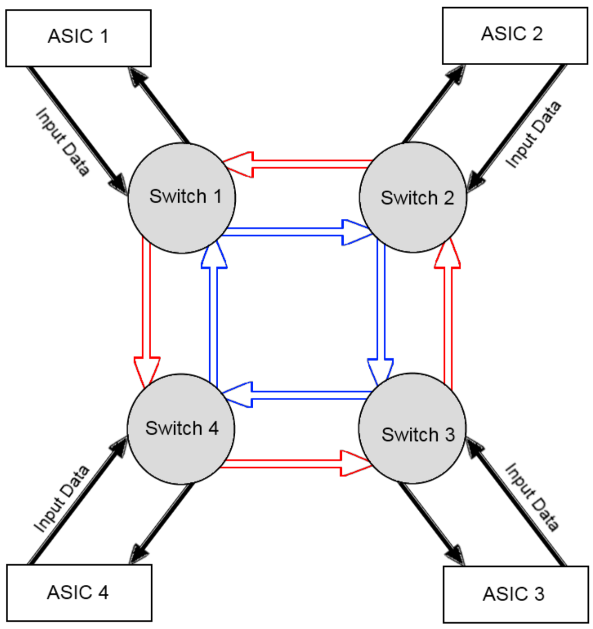


Figure 2.1: Ericsson switch model

Each switch is connected to an ASIC as can be seen in figure 2.1, as well as two adjacent switches. Thus packets which need to be sent across the network will have to be sent through a total of three switches.

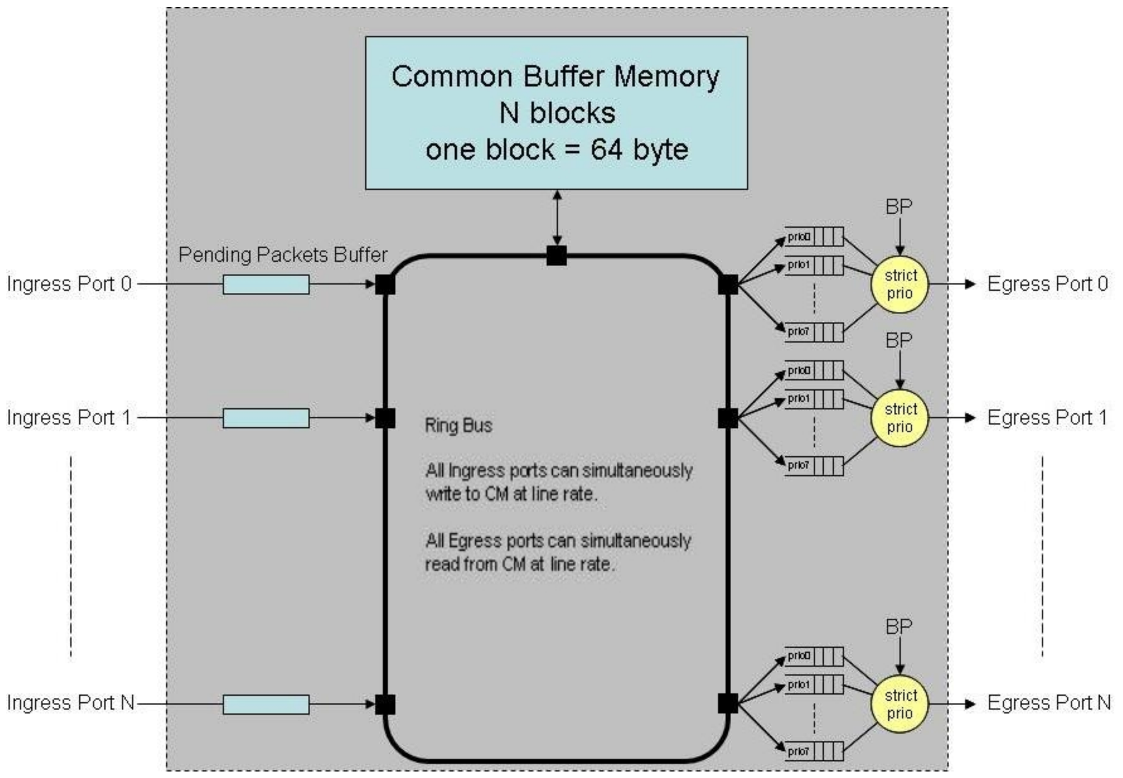


Figure 2.2: Specification for the internal workings of the switches

Each switch includes a common memory (CM) which is shared for all incoming data. The memory is divided into parts which are dedicated to each level of data priority. The required time for writing data to the memory should be packetsize/linkrate, additionally a fixed delay of 3 μs is required for processing data through each switch. The maximum time between sending two packets should be Maximum Transfer Unit (MTU) divided by the linkrate, for this thesis the MTU will be the same as packetsize/linkrate due to the fixed packetsize [6].

Due to the specified delays the lowest possible latency will thus become:

$$2 * 5\mu s + X * 3\mu s + (X + 1) * packetsize/linkate \qquad (2.1)$$

where X is the amount of switches.

When data is sent through two switches the lowest possible latency becomes 16,1536 μs and 19.2048 μs for three switches. If blocking occurs in the network however, the latency will be larger.

Data will be sent periodically over a Transmission Time Interval (TTI) of 2 ms which is due to the High-Speed Downlink Packet Access (HSDPA) scheduler located on the chip. The traffic is divided into four priority data levels, in which every ASIC can both send and receive.

The highest priority data will be sent in a bursts which will last a total of 20% of the interval, i.e. 400 μs. All lower priorities are sent a total of roughly 20% each. Lower priorities are spread out over the full interval except during the high priority burst interval [6].

2.1.1 Memory

Each switch in the network contains a common memory which enables received data to be stored until it is ready to be sent. The CM is divided into dedicated parts for each priority, these define virtual channels (VC) [4]. Using virtual channels allows for several connections to make use of the same physical channel, thus preventing blockage and wasted bandwidth. When the CM gets full while data is still being sent, the switch implements a backpressure function (BP). The BP functions by preventing any further data to be sent towards a congested switch. This is accomplish by letting every switch monitor the load of its CM. In order to prevent any further transmission towards a full switch an off-signal will be sent to its adjacent switches. This function is useful in systems were packets are not allowed to be dropped [6].

2.2 Related work

The subject of improving trade-offs between GL versus BE latency is discussed in the article [7]. The paper introduces a parallel scheme that detects the absence of GT packets and relaxes or underutilize reservation of GT of time-critical traffic, were bandwidth is distributed dynamically. The presented scheme for this thesis utilises a similar solution regarding dynamic bandwidth distribution but for a serial link implementation.

The articles [8] and [9] describes hardware complexity vs. hardware efficiency when mixing GT and BE services. The authors suggest a hardware design that combines architectures of both BE- and GT routers which yield efficient sharing of resources. This combined architecture shows significant advantages when using a single algorithm for both GL and BE services, very similar to the implementation used for this thesis were a single algorithm makes up for both services.

The paper [10] suggests a high speed and low area system design which implements a mix of GT and GL services. The authors also introduce adaptiveness of the system due to a memory-mapped configuration port that makes the network re-configurable during runtime. The authors describe their method as "To implement guarantees, we use contention-free routing, which is based on a time-division multiplexed circuit-switching approach, where one or more circuits are set up for a connection". The authors of the paper create guarantees by routing whereas this thesis uses scheduling between stored packets in memory to implement guarantees.

Regarding the subject of alternative ways for obtaining GL service, the article [11] introduces a dynamic TDM-based algorithm called unified single-objective algorithm called Unified MApping, Routing, and Slot allocation (UMARS+). UMARS+ allocates bandwidth through dynamic controlled timeslot invoked by reading a bandwidth requirement estimation function.

2.3 Scheduling disciplines

Here the different scheduling disciplines of this thesis will be described.

2.3.1 Strict Priority

When utilizing SP the highest priority data in the VC will always be preferred. The SP scheduling does not take fairness into account meaning it will always attempt to guarantee a latency for the highest priority data in the CM. When the traffic load is high, e.g. during a high priority burst, it will cause a build-up of lower priority data waiting to be sent. Figure 2.3 shows the strict priority queue architecture and figure 2.4 displays an example of how this queue operates [12] [13].

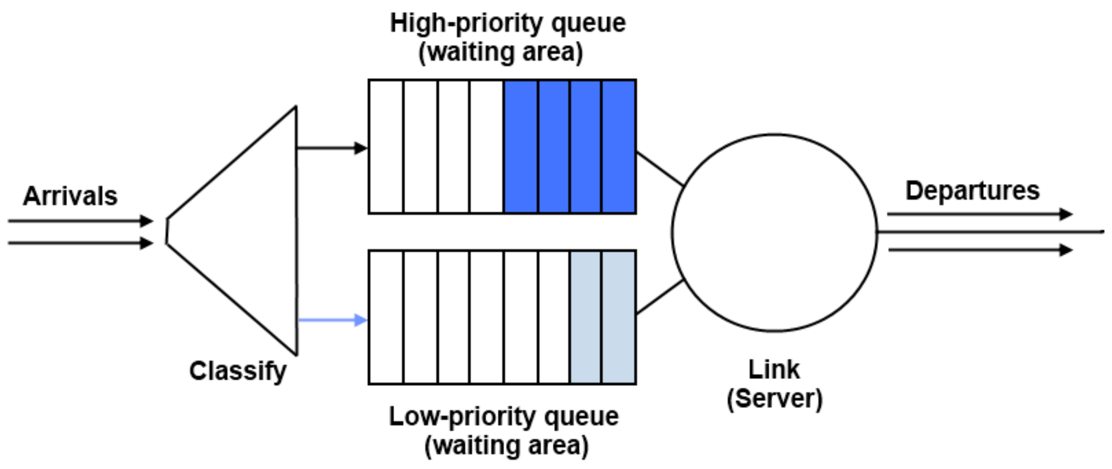

High-priority queue (waiting area)
Arrivals
Classify
Low-priority queue (waiting area)
Link (Server)
Departures

Figure 2.3: Strict priority scheduling

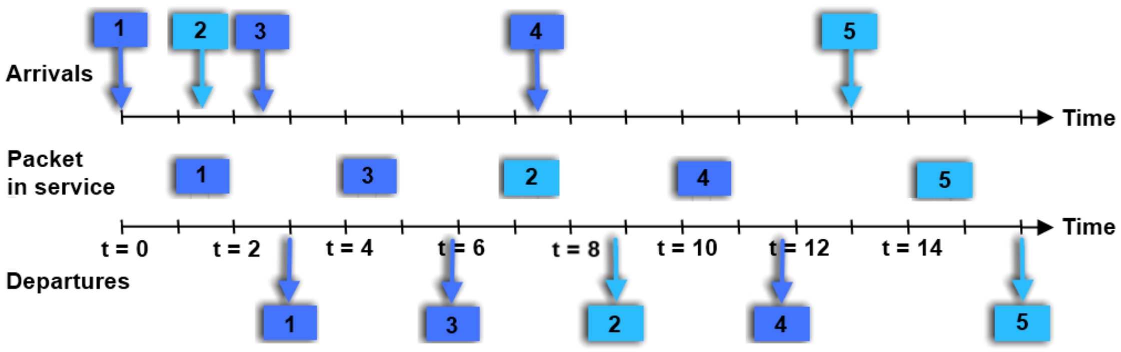


Figure 2.4: Strict priority operation where Prio1 packets is recognized by dark blue color and Prio2 turquoise

2.3.2 Round Robin

RR is a scheduling principle where the highest priority will be shifted like a token between the different priority levels after every sent packet. Once it has been sent the current channel will become the lowest priority and the next channel will become the highest priority. This means that all priorities will receive equal bandwidth, making it most suitable for non time critical systems where data is sent at a equal rate [14]. Figure 2.5 shows the Round Robin queue architecture and figure 2.6 displays an example of how this queue operates [15] [16].

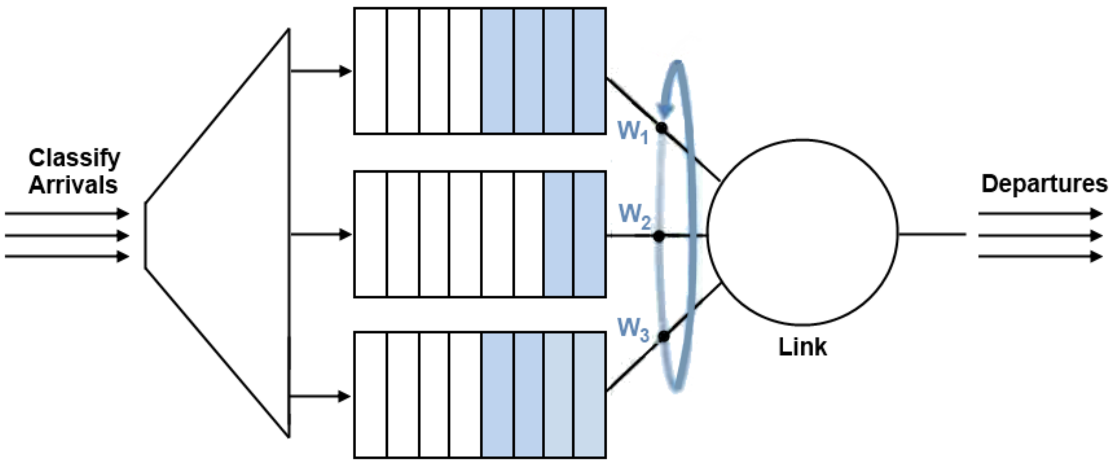


Figure 2.5: Round Robin scheduling

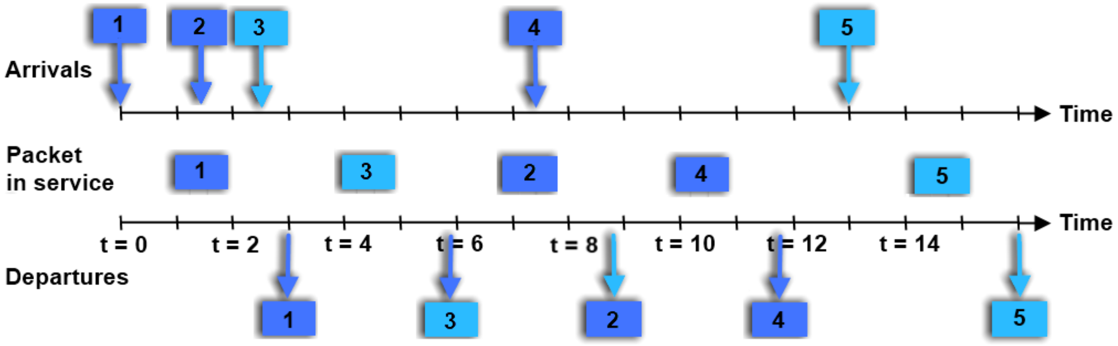


Figure 2.6: Round Robin scheduling, where Prio1 packets is recognized by dark blue color and Prio2 turquoise

2.3.3 Time-division multiplexing

Time-division multiplexing is a scheduling in which each channel will have a dedicated timeslot where its data can be sent. This discipline have a more flexible prioritization than scheduling like RR and SP since the size of the timeslot can be adjusted for each channel. The big disadvantage of this scheduling is the low use of spectral efficiency [17][18]. If, for example, the queue of Prio1 is empty during its timeslot bandwidth is going to be wasted, which instead could have been utilized for other priorities. Figure 2.7 displays how TDM operate its dedicated timeslots [5] [19].

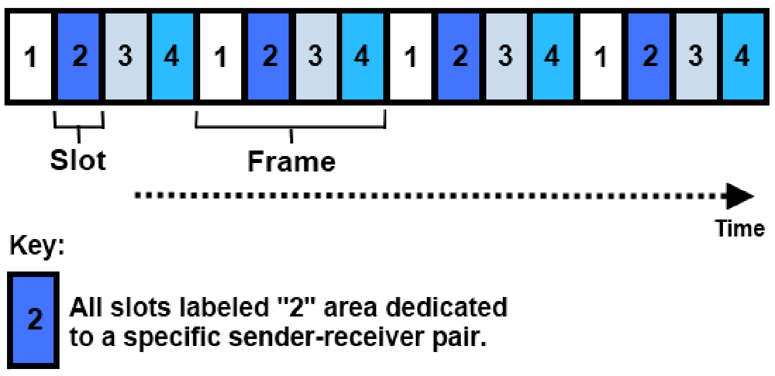

1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
Slot
Frame
Time
Key:
2
All slots labeled "2" area dedicated to a specific sender-receiver pair.

Figure 2.7: TDM operation

2.3.4 Asynchronous Latency Guarantee

ALG is a scheduling discipline proposed in [3]. The ALG is implemented in three steps; virtual channel, Admission control and Static Priority Queue (SPQ) which can be viewed in figure 2.8.

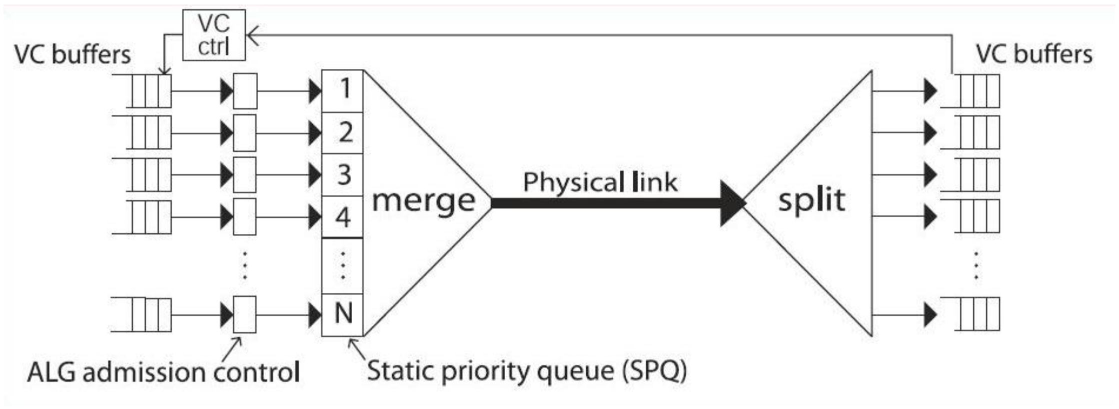


Figure 2.8: Detailed breakdown of the ALG scheduling

The SPQ is a parallel one-block memory dedicated to handling one data packet from each VC. The SPQ is meant as a buffer for outgoing messages whilst waiting for access to the network channel. The outgoing data is granted access to the channel by a static order,i.e. SP scheduling with one packet. An example of this is when Prio1 (the highest priority data) will be sent prior to all other SPQ data, Prio2 will wait for Prio1 but sent prior to Prio3 and so forth. Therefore data will wait proportionally to its priority.

In order to implement a latency guarantee of the lower priority VC's, higher priority data will delay any lower priority data in the SPQ once. This is handled by the intermediate admission control block as can be seen in figure 2.8 above. The amount of data in the SPQ is sampled while packets are transmitted on the physical link.

For example if data of a given priority, say Prio1, is the last to be transmitted on to the physical link, any other waiting priority will have to be transmitted before any further Prio1 data can be re-admitted into the SPQ. For further explanation see figure 2.9 below [3].

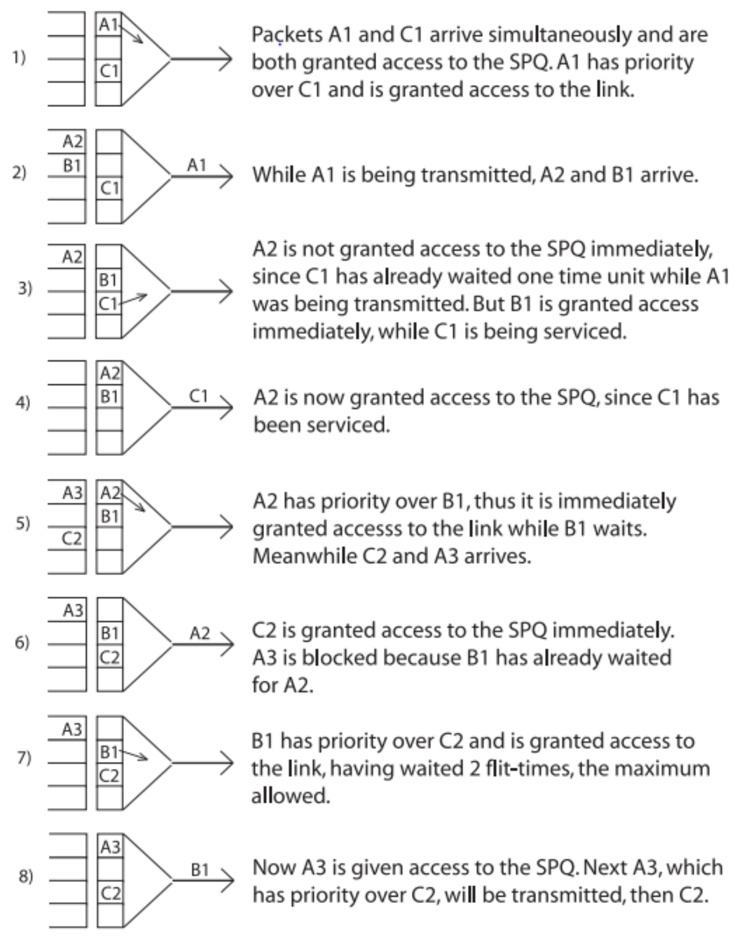

1)
A1
C1
Packets A1 and C1 arrive simultaneously and are both granted access to the SPQ. A1 has priority over C1 and is granted access to the link.
2)
A2
B1
C1
A1
While A1 is being transmitted, A2 and B1 arrive.
3)
A2
B1
C1
A2 is not granted access to the SPQ immediately, since C1 has already waited one time unit while A1 was being transmitted. But B1 is granted access immediately, while C1 is being serviced.
4)
A2
B1
C1
A2 is now granted access to the SPQ, since C1 has been serviced.
5)
A3
C2
A2
B1
A2 has priority over B1, thus it is immediately granted accesss to the link while B1 waits. Meanwhile C2 and A3 arrives.
6)
A3
B1
C2
A2
C2 is granted access to the SPQ immediately. A3 is blocked because B1 has already waited for A2.
7)
A3
B1
C2
B1 has priority over C2 and is granted access to the link, having waited 2 flit-times, the maximum allowed.
8)
A3
C2
B1
Now A3 is given access to the SPQ. Next A3, which has priority over C2, will be transmitted, then C2.

Figure 2.9: An execution example of ALG scheduling principle. Packets are named by priority and sequential order, e.g. the first and highest priority data is called A1.

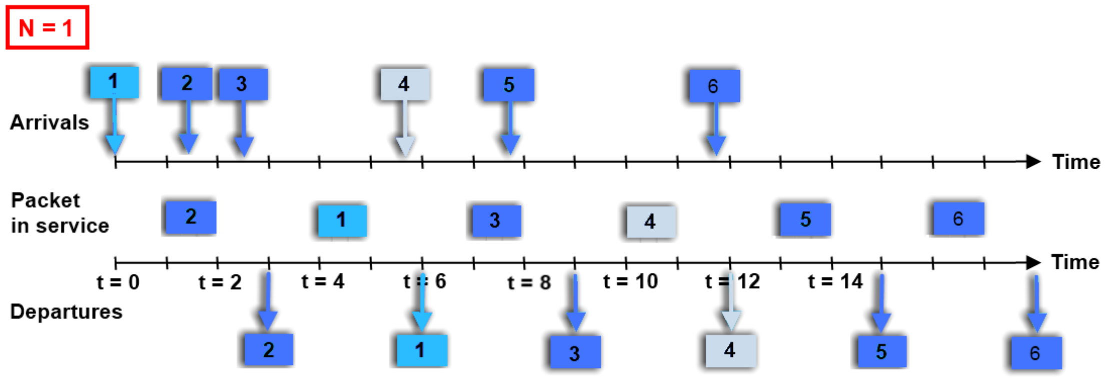


Figure 2.10: Scheduling of incoming packets using ALG where Prio1 packets is recognized by dark blue color, Prio2 turquoise, Prio3 gray.

2.3.5 Configurable Asynchronous Latency Guarantee

The ALG scheduling principle was modified for this thesis so that the number of times low priority data has to wait for higher priorities was adjustable. The authors have chosen to call this scheduling Configurable Asynchronous Latency Guarantee (CALG). This scheduling will allow the bandwidth to be distributed between the different priorities in order to improve performance of time-critical data. By adjusting the amount of times waiting, N, the results yields from a very fair scheduling, to a scheduling in which the highest priority allocates more bandwidth. Thus it is possible to find values of N where the latency will not exceed desired limits of latency. Figure 2.11 below shows an example of how high priority packets are handled when N = 2.

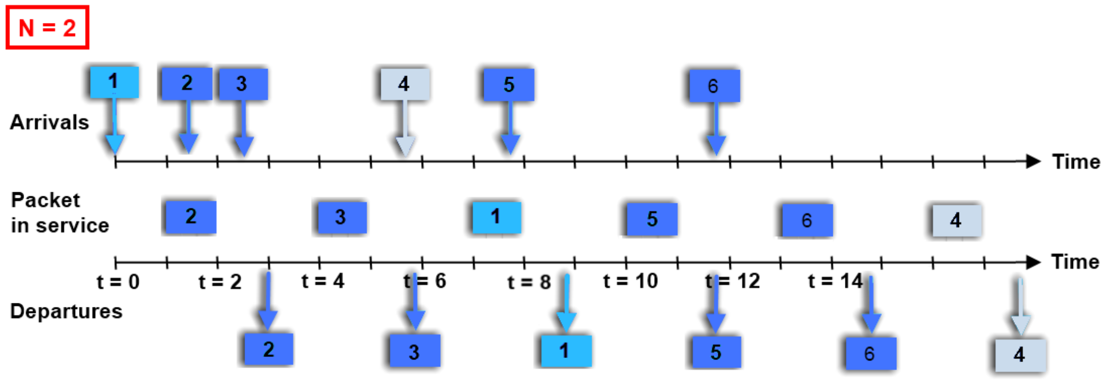


Figure 2.11: CALG operation with waiting parameter N=2 where Prio1 packets is recognized by dark blue color, Prio2 turquoise, Prio3 gray.

3

Method

This chapter describes the software used in this thesis, i.e. Ptolemy II and CoFluent. Further on the implemented software models are described as well as their characteristics. Finally the stimuli of Matlab generated input-files gets explained, such as construction and data distribution.

3.1 Implementation

As mentioned above the implementation of the simulation is done in two different modelling software's: Ptolemy II and CoFluent Studio.

3.1.1 Ptolemy II: using Vergil

Ptolemy II is an open-source simulation software which was specifically designed for simulating embedded systems. Ptolemy II is Java based software using actor-oriented design. Actors are software components which execute concurrently and communicate via ports. It is possible to create composite actors which include a set of pre-constructed actors, thus giving it a hierarchical design. [20]

Ptolemy II uses directors to handle the actors. These directors have different topologies which determines how actors will be interpreted. These topologies includes process networks (PN), discrete-events (DE), dataflow (SDF), synchronous/ reactive(SR), rendezvous-based models, 3-D visualization, and continuous-time models. Different directors can be used in different hierarchies, giving the ability of simulate for different data interpretations, e.g. both continuous and discrete events. [20]

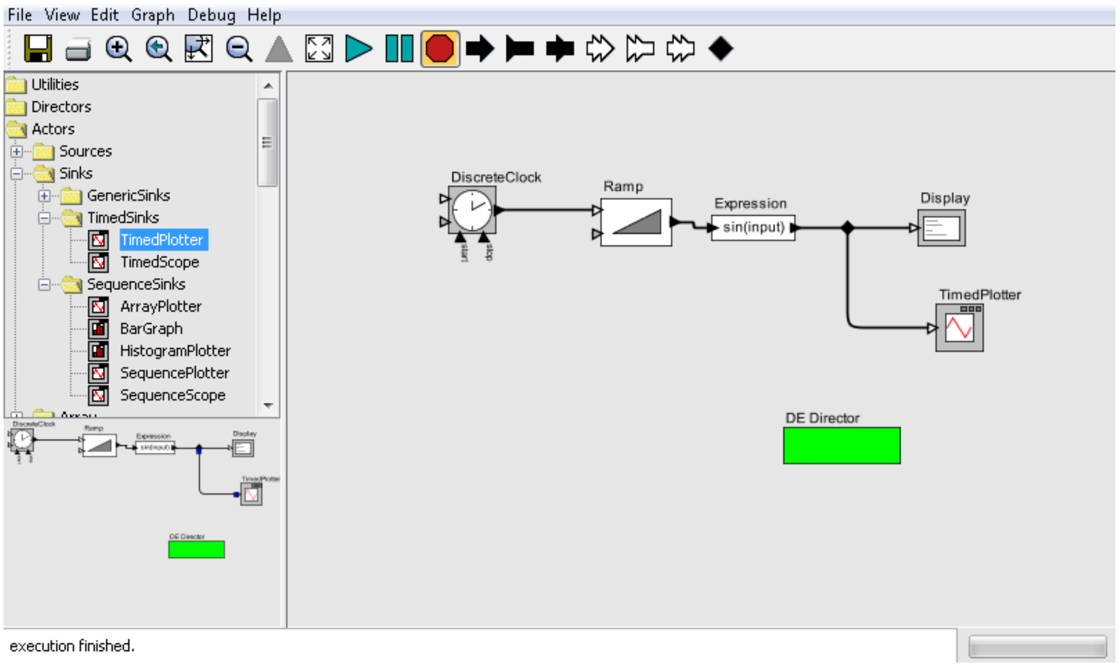


Figure 3.1: Figure of the Vergil interface

In Ptolemy it is possible to create modal models, these consists of state machines which are handled by the directors. This can be advantageous when modelling discrete systems as opposed to creating corresponding hardware.

The design can be divided into data generation, switching, buffers, prioritizing data and flow control. Data generation was the first step of the design when we decided to send information as packets to simplify simulation and reduce events in the model, increasing simulation speed. Prio1 data was modelled as a constant stream which would run for 200 μs or 10% of TTI. This data is generated by a sequential source which is controlled by a clock that make it run once for every TTI.

Prio2 data source was constructed in a similar fashion as Prio1 but with different parameters that use a random data distribution of 10% over a full TTI.

The switches inside the model have main implementations such as CM for use of virtual channels and BP for congestion control. These implementations were modelled primarily by the use of queue functions, logic blocks and counters. The logic blocks control the flow of data while the counters is essential for constructing the BP function.

As the work progressed Ptolemy II became very tedious to use due to problems of creating custom actors. The purpose of a custom actor is to execute custom code that represents a chain of operation within a single block. These blocks would help minimize the visual complexity of the model, but it turned out that this functionality were incomplete. The model became large, complex and obscure with wires spread out all over the screen. Some attempts were made to resolve this by using the hierarchical views, that in turn resulted in bugs and severe crashes. A final decision were made to use CoFluent studio and abandon further development in Ptolemy II.

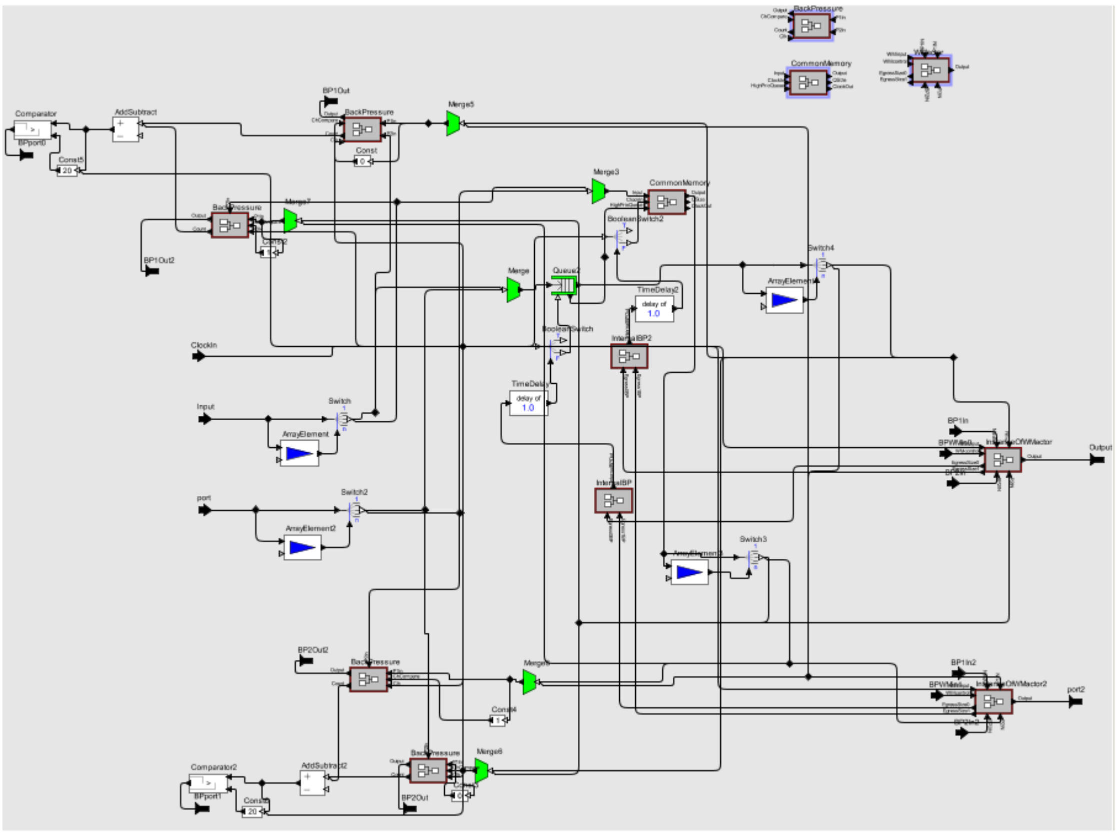


Figure 3.2: Switches designed in Ptolemy

Figure 3.2 shows the switch-design. As can be seen in the figure it became very messy to look at and was not helpful for designing components.

3.1.2 CoFluent Studio

CoFluent Studio is a software for modelling and simulation of electrical systems. CoFluent uses a top-down design process in which a graphical model of the system is implemented and is complemented with C++ code. CoFluent uses a three step design, the first of which graphical connections are designed between functions, variables and memories. Next, attributes such as time and size are modified in the model. Finally, code is added to the operations of the functions to implement algorithms, though it is possible to implement the code before modifying attributes if this has higher priority. [21]

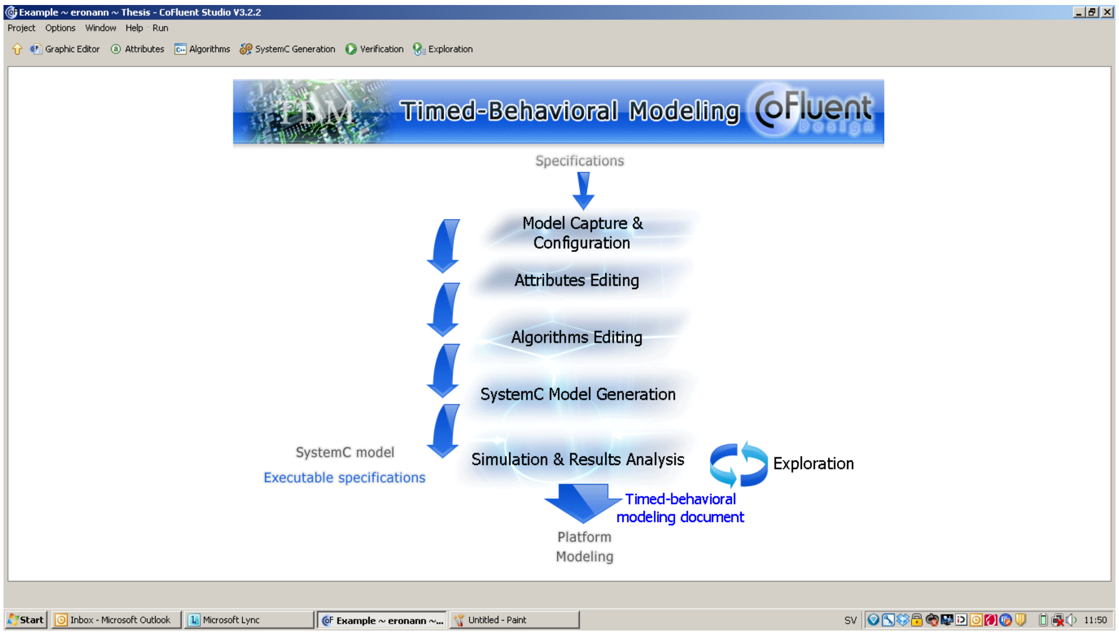


Figure 3.3: CoFluent project start-up screen

Figure 3.3 shows the design flow of CoFluent.

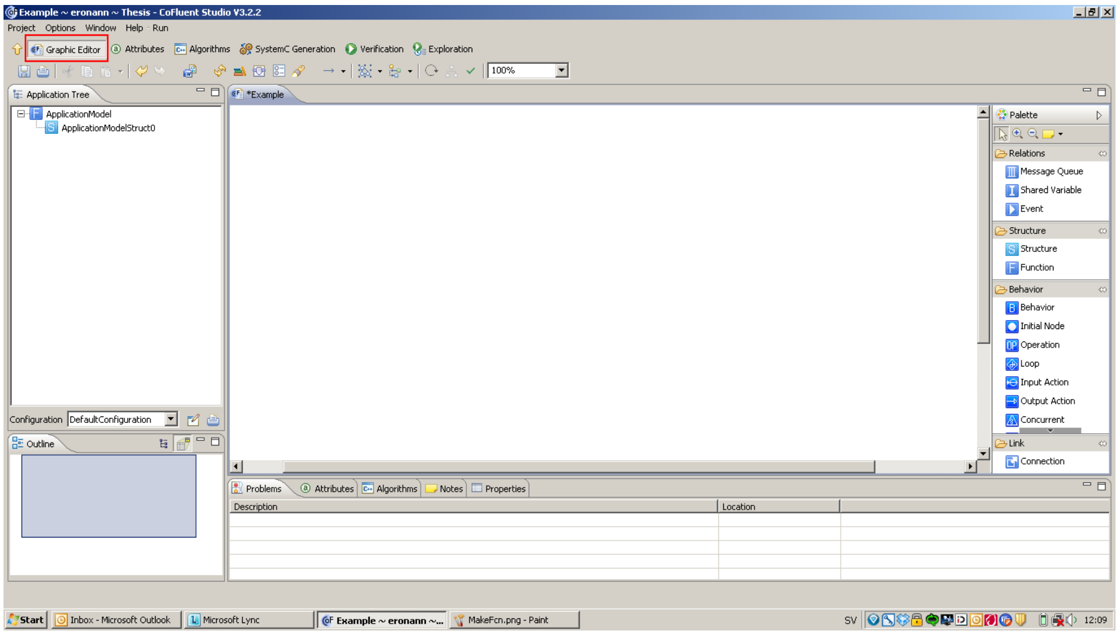


Figure 3.4: CoFluent graphic editor.

Figure 3.4 shows the Graphic editor, where the model outline is defined.

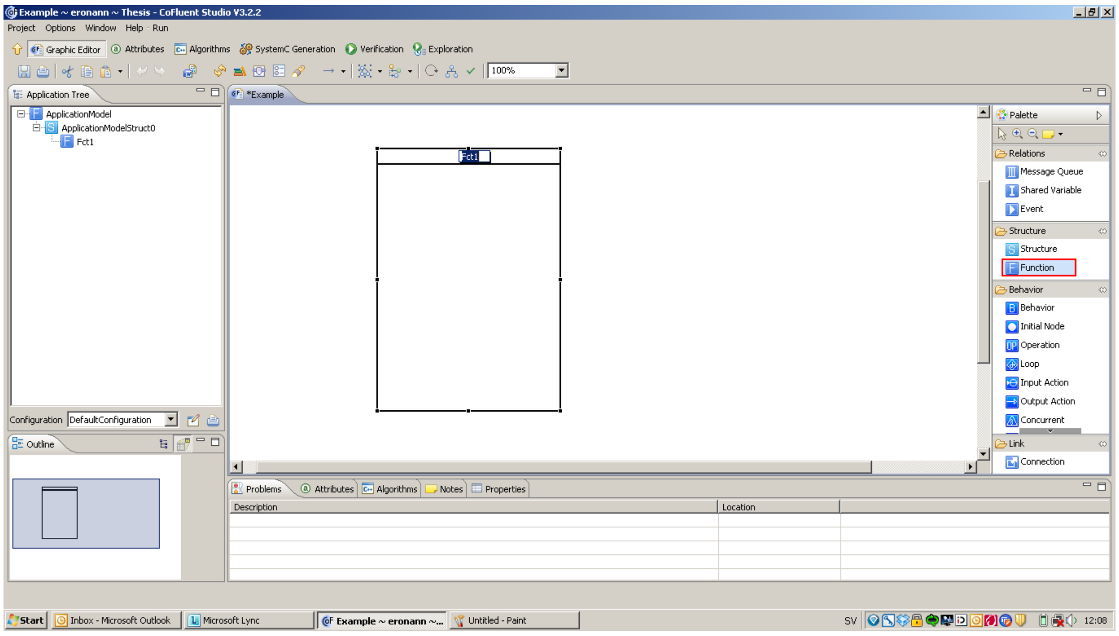
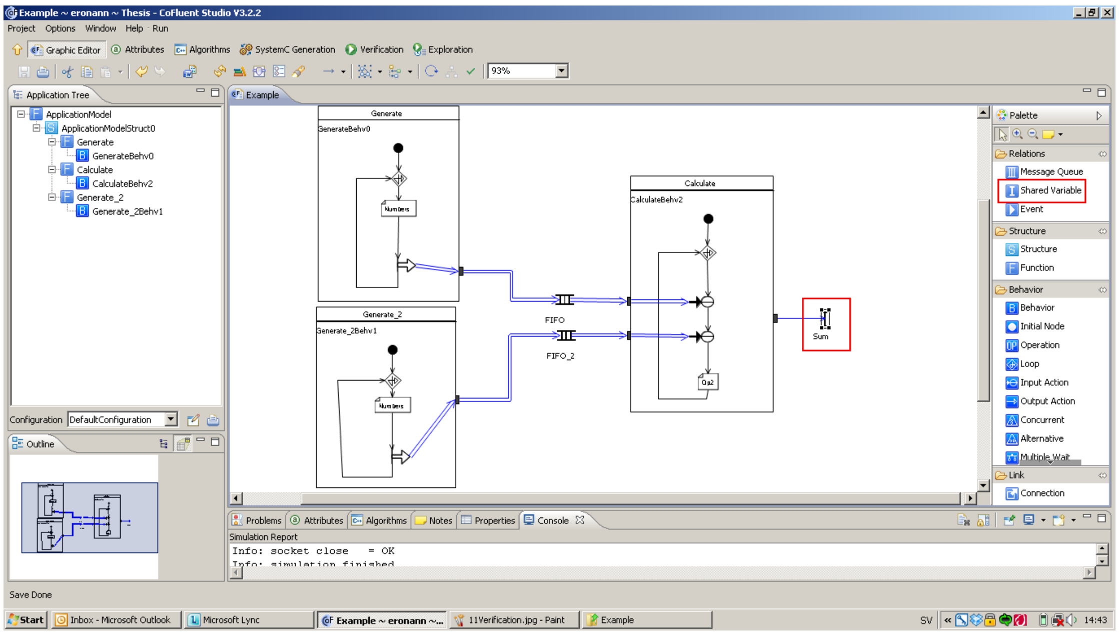


Figure 3.5: Function is added in graphic editor

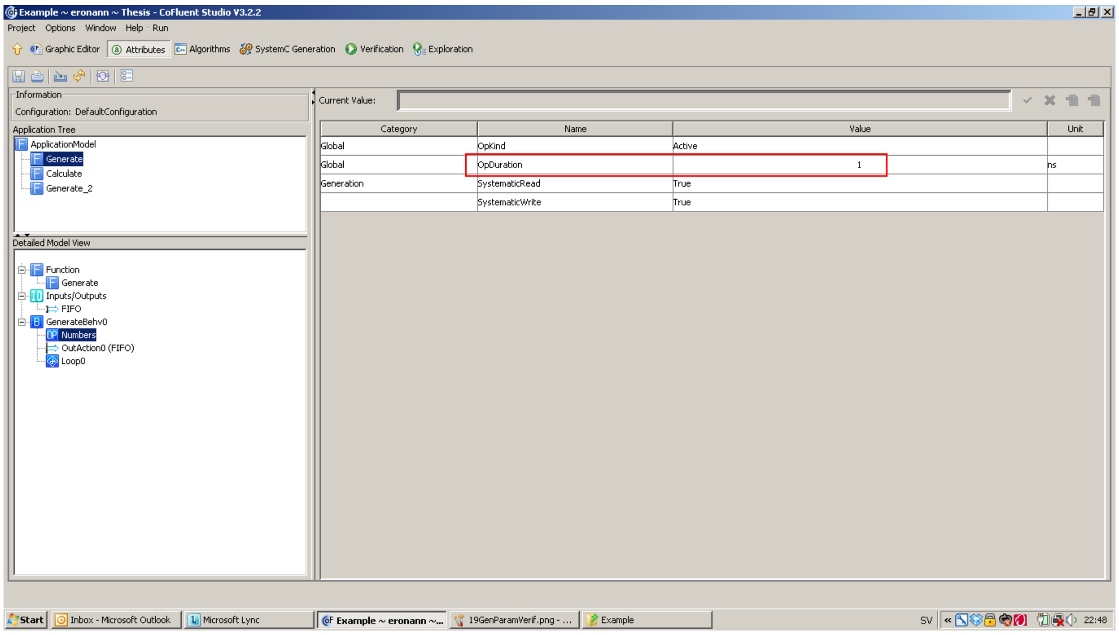
Functions are added as windows in CoFluent. As a behaviour is defined, components are added into it. Functions defined in CoFluent consist of loops in which a token is sent. When the token reaches a block the corresponding operation will be performed. This system is used to be able to implement code into operation blocks and make them able to interact with other functions.

Calculated data can be stored in a Shared Variable or printed to the console window. Shared variables as well as model parameters can be displayed in graphs in the verification view, giving high visibility of the internal processes of the model. This can be used in conjunction with general parameters which make it possible make changes of the model as simulation is performed. The exploration view further makes it possible to do parameter sweeps to see the result of different simulation parameters.



```
Info: socket close   = OK
Info: simulation finished
```

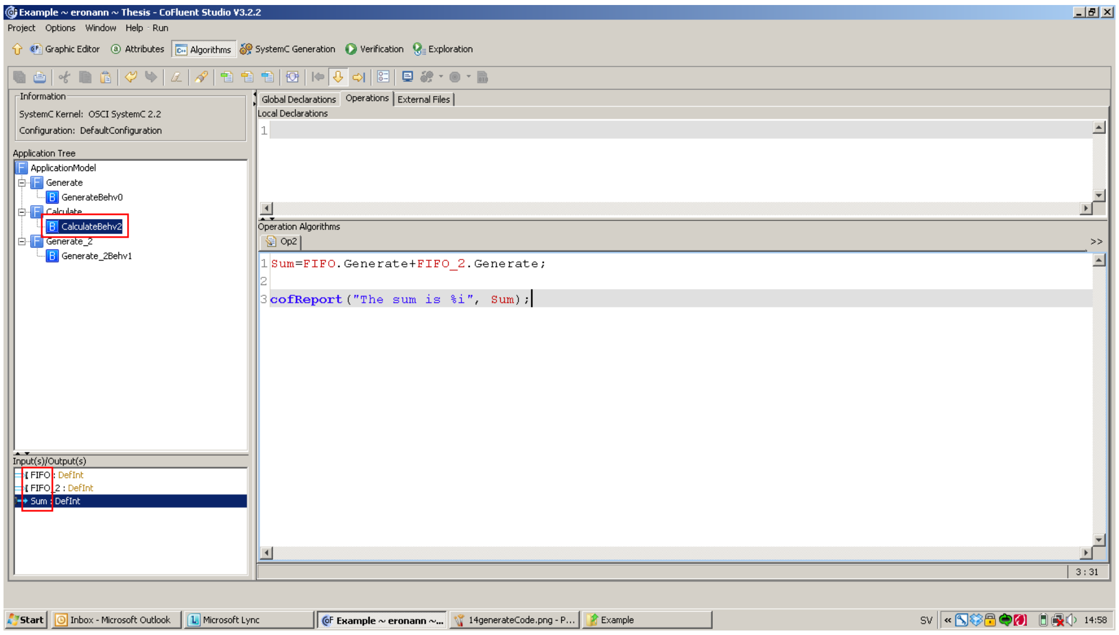
Figure 3.6: The figure shows a set of functions in CoFluent which sends data and stores it on a shred variable.


Example ~ eronann ~ Thesis - CoFluent Studio V3.2.2
Project Options Window Help Run
Graphic Editor Attributes Algorithms SystemC Generation Verification Exploration
Information
Configuration: DefaultConfiguration
Application Tree
ApplicationModel
Generate
Calculate
Generate_2
Detailed Model View
Function
Generate
Inputs/Outputs
FIFO
GenerateBehv0
Numbers
OutAction0 (FIFO)
Loop0
Current Value:

Category	Name	Value	Unit
Global	OpKind	Active	
Global	OpDuration	1	ns
Generation	SystematicRead	True	
	SystematicWrite	True	

Figure 3.7: Customise duration of an operation.

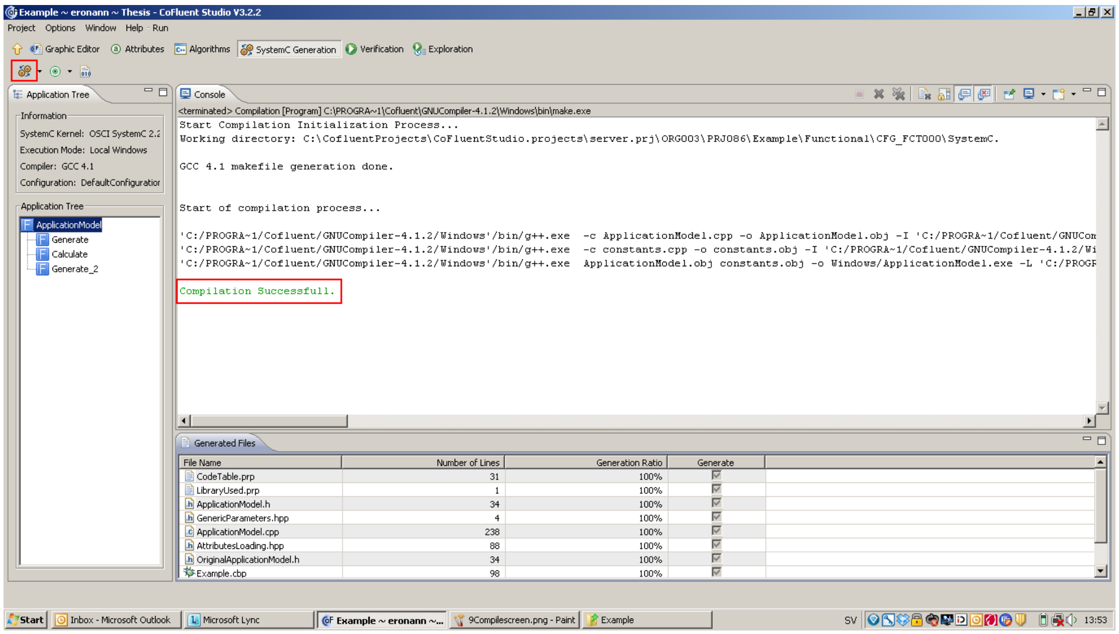
In the Attributes view changes to attributes of the blocks such as timing information, buffer size and behaviour can be made. In figure 3.7 the operation time of the operation ”Numbers” is changed from the default 10 ns to 1 ns.



```
Sum=FIFO.Generate+FIFO_2.Generate;

cofReport("The sum is %i", Sum);
```

Figure 3.8: Figure shows how code is added into the operations

When a graphical model is complete, using the Algorithms view it is possible to add C++ code into the operations to add a behaviour into the model.

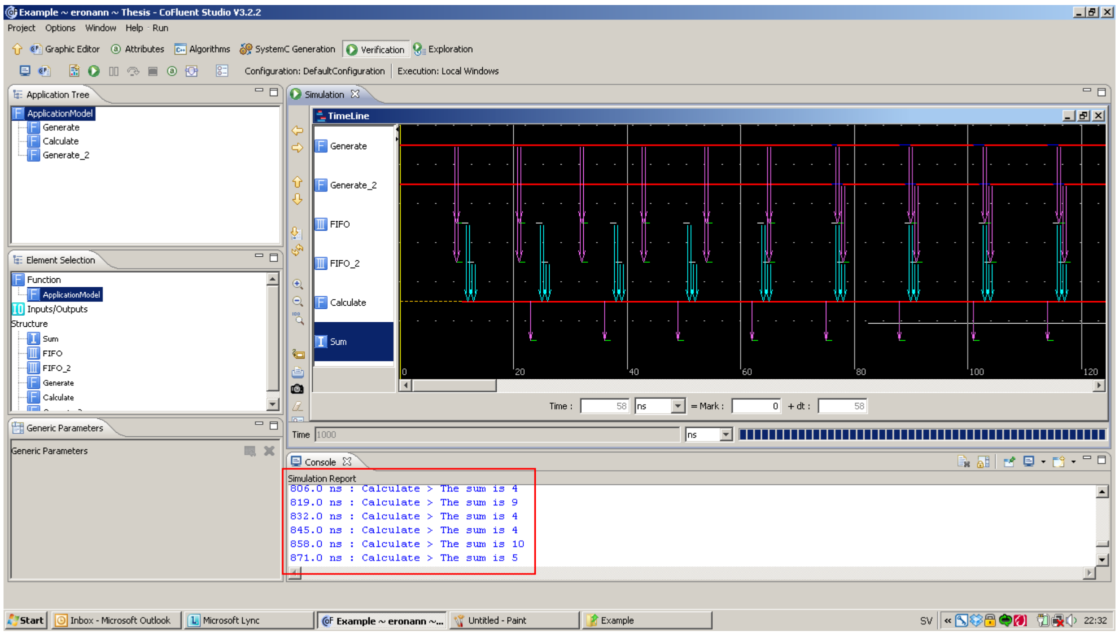


```
<terminated> Compilation [Program] C:\PROGRA~1\Cofluent\GNUCompiler-4.1.2\Windows\bin\make.exe
Start Compilation Initialization Process...
Working directory: C:\CofluentProjects\CoFluentStudio.projects\server.prj\ORG003\PRJ086\Example\Functional\CFG_FCT000\SystemC.

GCC 4.1 makefile generation done.


Start of compilation process...

'C:/PROGRA~1/Cofluent/GNUCompiler-4.1.2/Windows'/bin/g++.exe  -c ApplicationModel.cpp -o ApplicationModel.obj -I 'C:/PROGRA~1/Cofluent/GNUCom
'C:/PROGRA~1/Cofluent/GNUCompiler-4.1.2/Windows'/bin/g++.exe  -c constants.cpp -o constants.obj -I 'C:/PROGRA~1/Cofluent/GNUCompiler-4.1.2/Wi
'C:/PROGRA~1/Cofluent/GNUCompiler-4.1.2/Windows'/bin/g++.exe  ApplicationModel.obj constants.obj -o Windows/ApplicationModel.exe -L 'C:/PROGR

Compilation Successfull.
```

File Name	Number of Lines	Generation Ratio	Generate
CodeTable.prp	31	100%	✓
LibraryUsed.prp	1	100%	✓
ApplicationModel.h	34	100%	✓
GenericParameters.hpp	4	100%	✓
ApplicationModel.cpp	238	100%	✓
AttributesLoading.hpp	88	100%	✓
OriginalApplicationModel.h	34	100%	✓
Example.cbp	98	100%	✓

Figure 3.9: Figure shows the code generation view of CoFluent

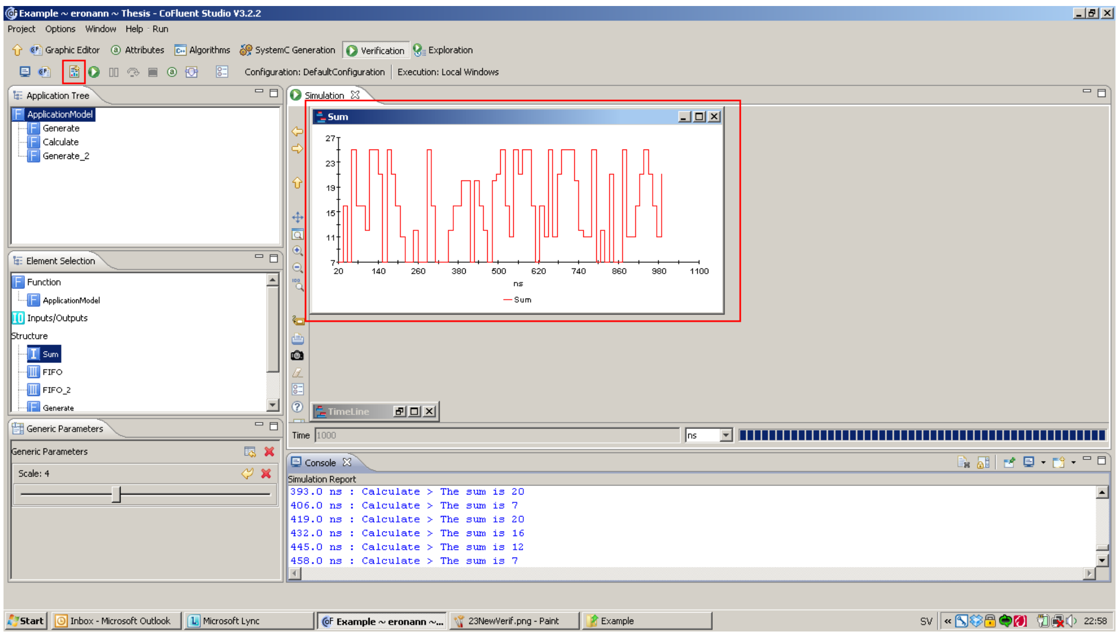
When a model is defined without error the model will need to be compiled. In the System Generation view CoFluent compiles the model and generates System C code to be run in the verification window.



```
Simulation Report
806.0 ns : Calculate > The sum is 4
819.0 ns : Calculate > The sum is 9
832.0 ns : Calculate > The sum is 4
845.0 ns : Calculate > The sum is 4
858.0 ns : Calculate > The sum is 10
871.0 ns : Calculate > The sum is 5
```

Figure 3.10: Results in verification view.

Figure 3.10 shows the verification view in which a timeline of the sent signals is displayed. The result of calculations are displayed in the Console window.



```
393.0 ns : Calculate > The sum is 20
406.0 ns : Calculate > The sum is 7
419.0 ns : Calculate > The sum is 20
432.0 ns : Calculate > The sum is 16
445.0 ns : Calculate > The sum is 12
458.0 ns : Calculate > The sum is 7
```

Figure 3.11: Observation of the variable Sum in time chart.

Alternatively it is possible to view the simulated values as a graph of the values in the y-axis and simulated time on the x-axis.

3.1.3 Model design

The simulated network contains three main component blocks: producers/consumers, routing and switches. As the focus of the thesis was mainly related to arbitration of data inside the switches the producer/consumer and routing was kept as simple as possible. The producer/consumer send data which is read from a pre-generated input-file. The consumers task is to receives data, measure latency and calculate throughput. The producer and consumer functions can be viewed in figure 4.2.

The routing is based on two vital functions. The first function forwards data from the producer towards its associated switch. The second function receives data from one switch and routes this to either another switch or to the consumer (end-point), depending on if it reached its destination or not. The routing functions can be viewed in figure 4.3.

The switches were divided into several functions, these were mainly implemented for receiving and sending signals and can be viewed in figure 4.4 in the results chapter.

In order to have the model receive packets a function called Receive_Interface was introduced. The function does not only receive packets but also sorting them into the corresponding VC, the outcome is based on the packet priority. This function was also expanded to handle BP in order to prevent packets to be sent into the CM when its memory occupation has exceeded a certain limit.

Since this CoFluent model was designed to not drop any packets the BP function resulted in that other sources could no longer send packets towards the affected switch, therefore an indirect BP was achieved. The BP was implemented this way to reduce complexity of the model since this narrows down the amount of variables that had to be shared between different switches.

Inside the switch block a forward function was made, its main task was to choose which packets to be sent based on a implemented algorithm. This function executes the scheduling principles and selects which VC to send data from. The most complex part of this function is to design it to work with the pre-existing parts of the model. The function had to be able to hold the sending token until it receives an incoming packet or else the token would loop empty for an extreme amount of times, which caused the simulation duration to increase more than 10 times. This was solved by introducing a queue which contained the amount of messages occupying the VC. The token was stuck in the input action until there were messages in the VC.

For the CALG scheduling additional functions and queues required to implement admission into the SPQ. These functions had variables which counted the amount of messages sent and decided if the VC could send messages to the SPQ.

3.1.4 Data stimuli

As the information content of the input data, aside from routing information, is irrelevant to the goals of the simulation it was decided traffic would consist of generic packets. Each packet was sent carrying information of priority, destination and size to be used for routing and measuring. Sending packets alleviates the process of sending and receiving data as compared to sending individual bits. The empty packet slots is reserved for CoFluent and these are already included in order to maintain a fixed packet size.

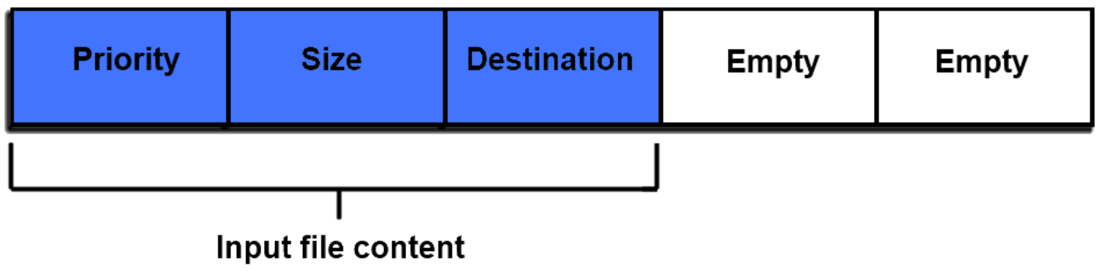


Figure 3.12: Elements of packet from Matlab input file

When packets gets read into the CoFluent model the empty slots get processed with routing information regarding source and timestamp. The timestamp is necessary when measuring transmission time from end to end.

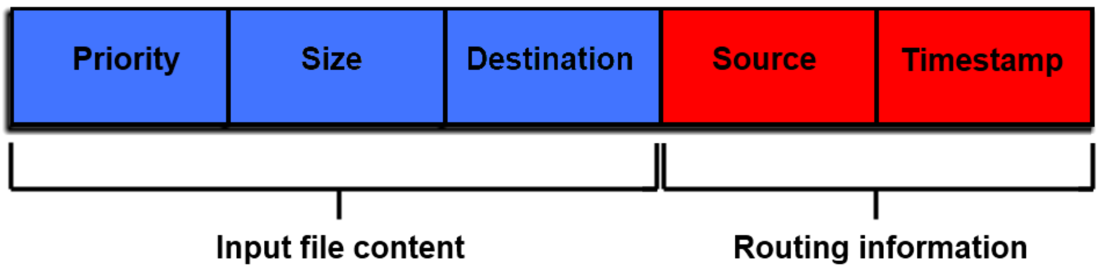


Figure 3.13: Elements of processed packet in CoFluent

The four input-data files or stimuli read by the CoFluent models were generated in Matlab. These files were coded to represent a fictive data spectrum over one TTI, i.e the information sent from UE to each ASIC:s during the period of 2 ms. A generated data file consists of a given amount of integer arrays, each representation information for one packet.

Initially, in Matlab these packet arrays are filled with zeros which then gets overwritten by four levels of priority data. Prio1, the high priority data, occupies a random generated placement inside the spectrum and its characteristic is made up by a consistent burst of given length. The Prio2-Prio4 data, each of same length as Prio1, are spread out as a uniformly mix over the remaining non-occupied space [22] of the spectrum. The generated packets for any priority have a random destination node, except for its individual node. Finally, inside the CoFluent model the packet arrays for each file are read row-wise by an iterating loop inside the corresponding producer-block. All zeros, those packets that never got overwritten, are processed as empty loop iterations with duration of one packet time.

In order to construct the stimuli some calculations had to be done. Given from the specifications:

- Bandwidth: 10 Gbit/s
- Fixed packet size: 64 bytes or 512 bits
- TTI: 2 ms

The maximum amount of packets to be sent (or to be fitted in to the packets spectrum) during one TTI is given by:

$$\frac{Bandwidth * Periodtime}{Packetsize} = 39062 \, packets \tag{3.1}$$

In the main model specified to have stimuli of 40% bandwidth occupation, this yields that every priority gets 10% spectrum occupation each. In order to apply more stress to the system a 80% occupation stimuli was also made. As mentioned before, the distribution for high priority data burst have a random distribution while low priorities have uniform distribution over the non-occupied parts of the spectrum. Table 3.1 and 3.2 below sums up the stimuli properties of the two different test cases.

	BW occupation[%]	Tot. no Packets	Spectrum distr.	Destination distr.
Prio1	40	15624	Random, Burst	Random
Prio2	40	15624	Uniform	Random
Prio3	40	15624	Uniform	Random
Prio4	40	15624	Uniform	Random

Table 3.1: Stimuli properties: 40%

	BW occupation[%]	Tot. no Packets	Spectrum distr.	Destination distr.
Prio1	80	31248	Random, Burst	Random
Prio2	80	31248	Uniform	Random
Prio3	80	31248	Uniform	Random
Prio4	80	31248	Uniform	Random

Table 3.2: Stimuli properties: 80%

Figure 3.14 and 3.15 are samples from the spectrum during 40% respectively 80% bandwidth usage. I can be seen in figure 3.15 it has more of a dense traffic and a wider burst.

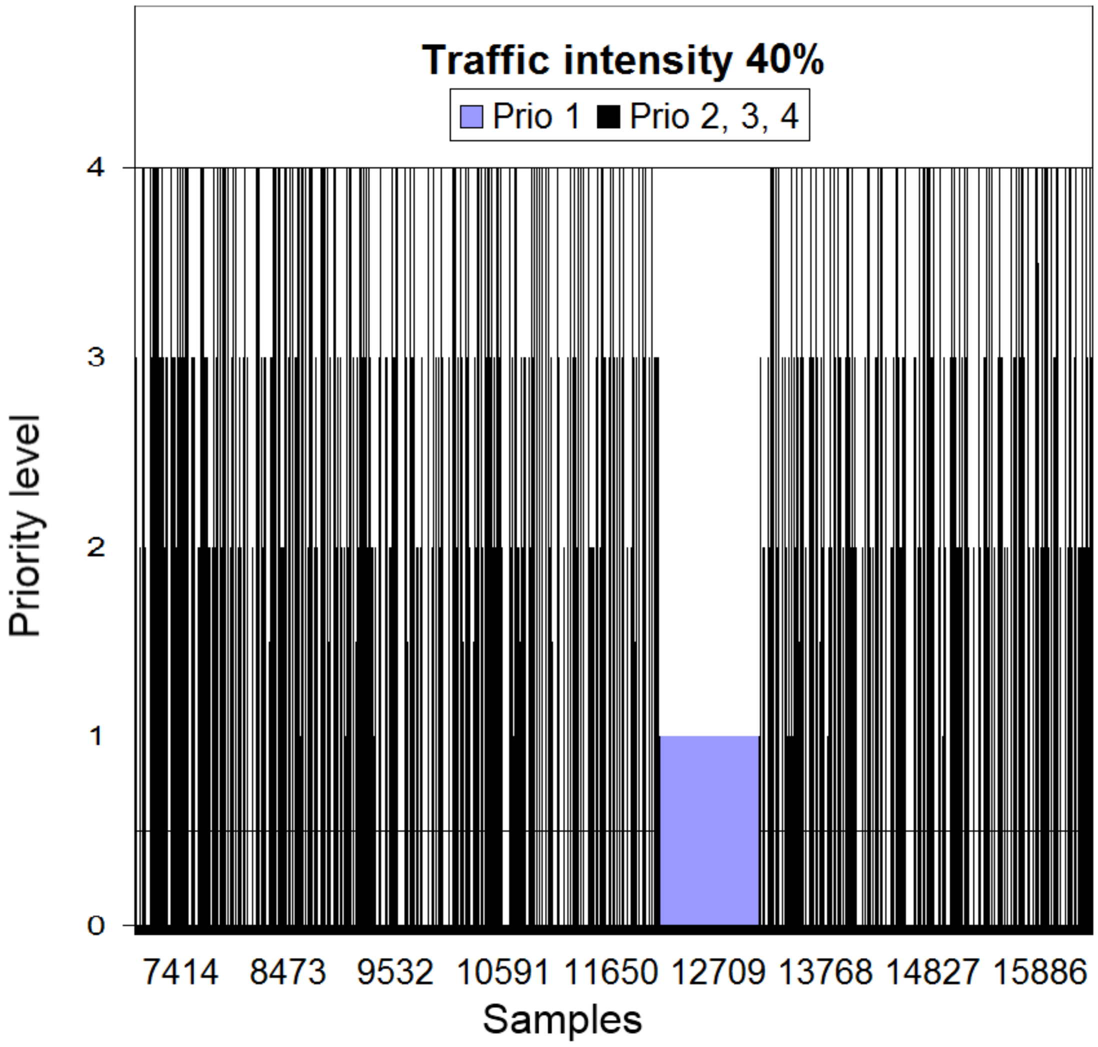


Figure 3.14: Sample of spectrum: 40% bandwidth usage.

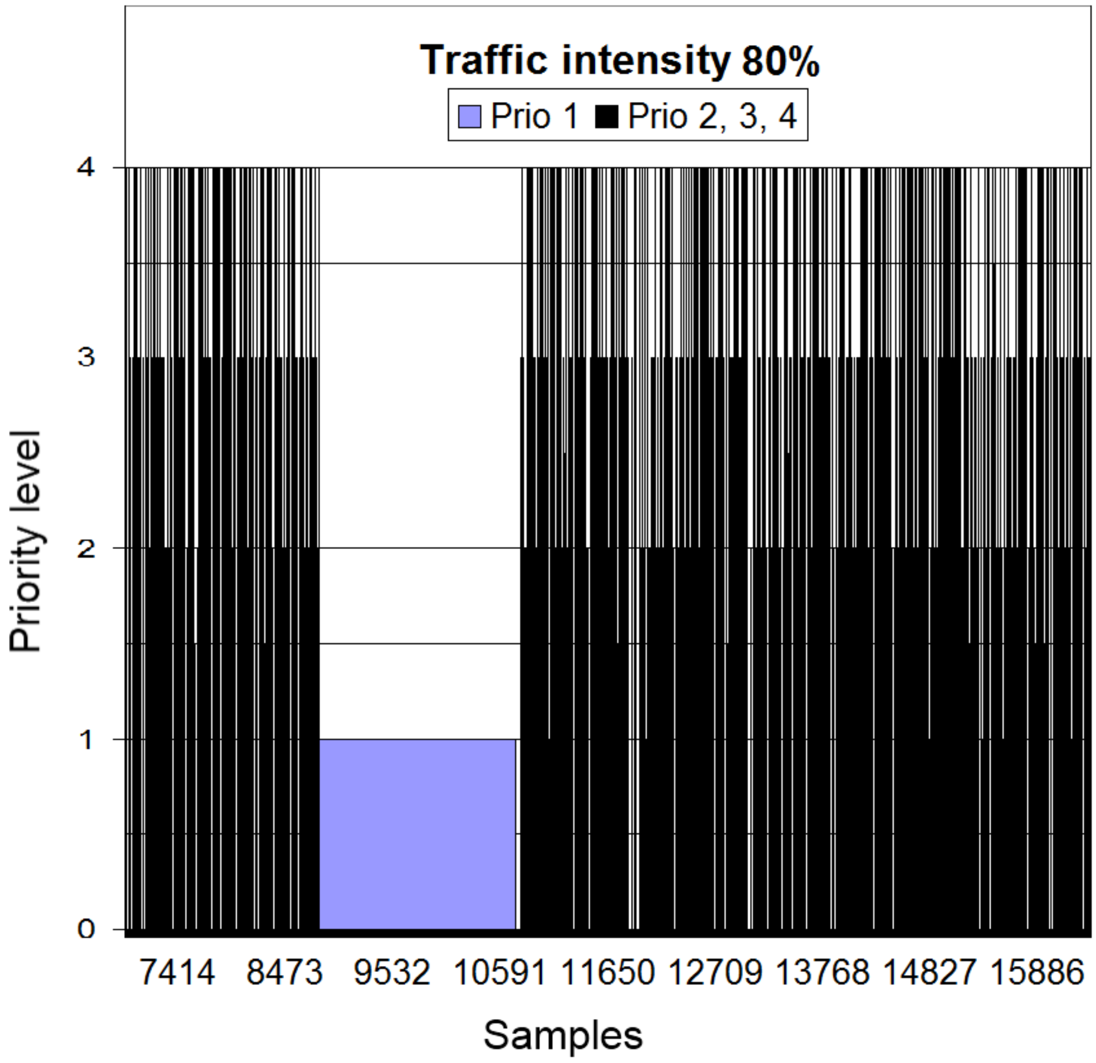


Figure 3.15: Sample of spectrum: 80% bandwidth usage.

4

Results and Discussion

This chapter presents the results achieved from constructing and simulating the model in this thesis. First the vital components created in the thesis will be presented, followed by the results simulated in CoFluent using this model with different scheduling types as discussed in chapter 2.

4.1 Implementation

The model was implemented using three major components:
A end-point component for Producer/Consumer functionality (applications block), a routing component and finally a switch component.

4.1.1 Overview

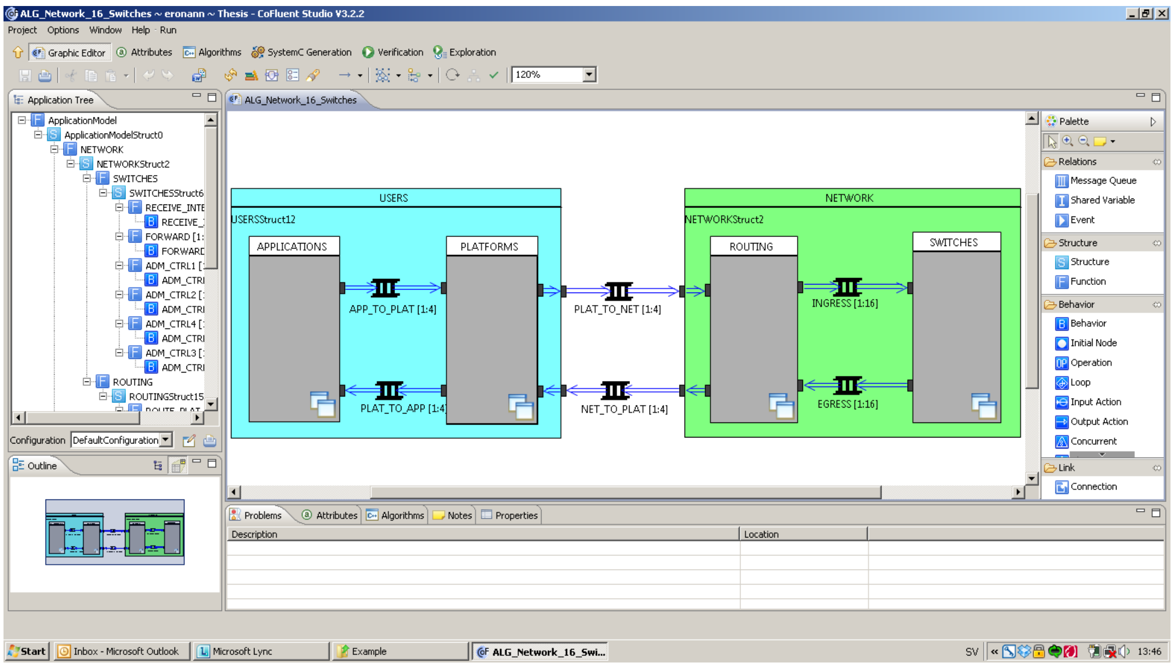


Figure 4.1: An overview of the model main components in the graphical environment of CoFluent studio

The figure 4.1 shows a fourth component called PLATFORMS, its purpose is to add the fixed delay of 5μs to the applications block. Both functions to the left will generate data and transmit it to the routing function on the right, were decisions being made of which switch data will be sent to. The data is sent back and forth between routing and switch component (representing number of hops required) until it is ready for the consumer end-point in the applications block. The Consumer function will calculate throughput and time of arrival for every packet.

4.1.2 Endpoints

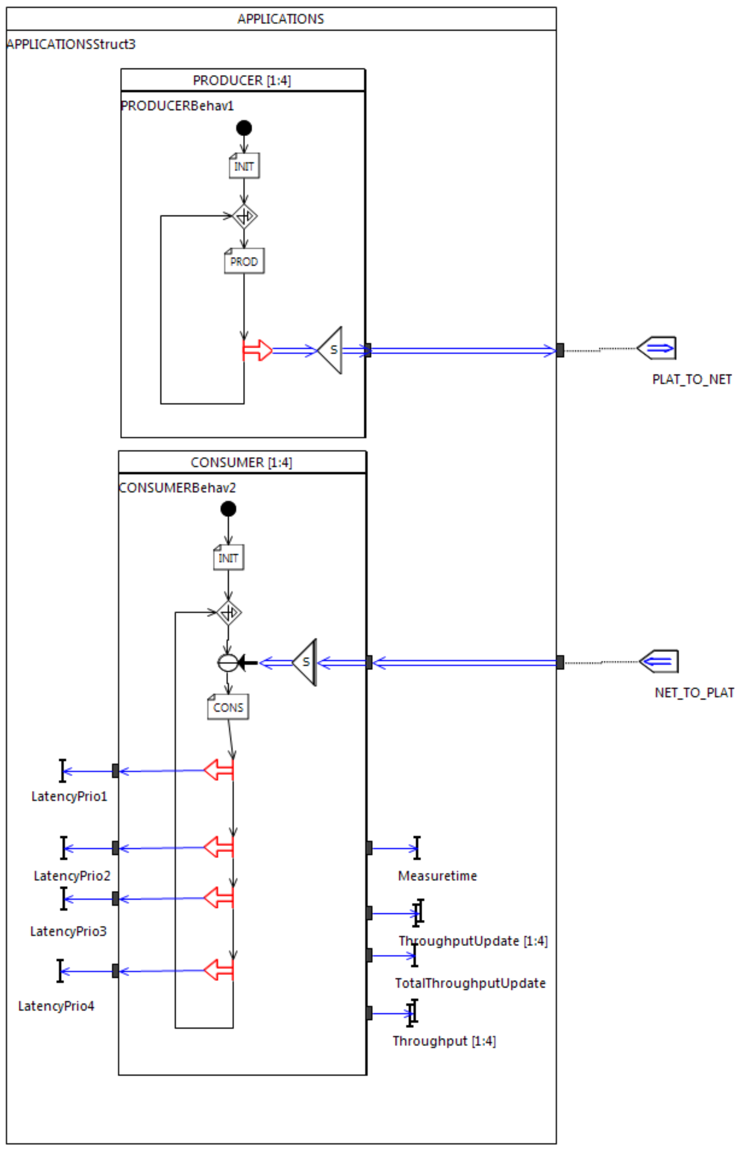


Figure 4.2: The figure shows the data end-point functions.

Figure 4.2 shows the two end-point functions. The upper one generates the data while the lower one receives, calculates and finally writes it to variables.

4.1.3 Routing

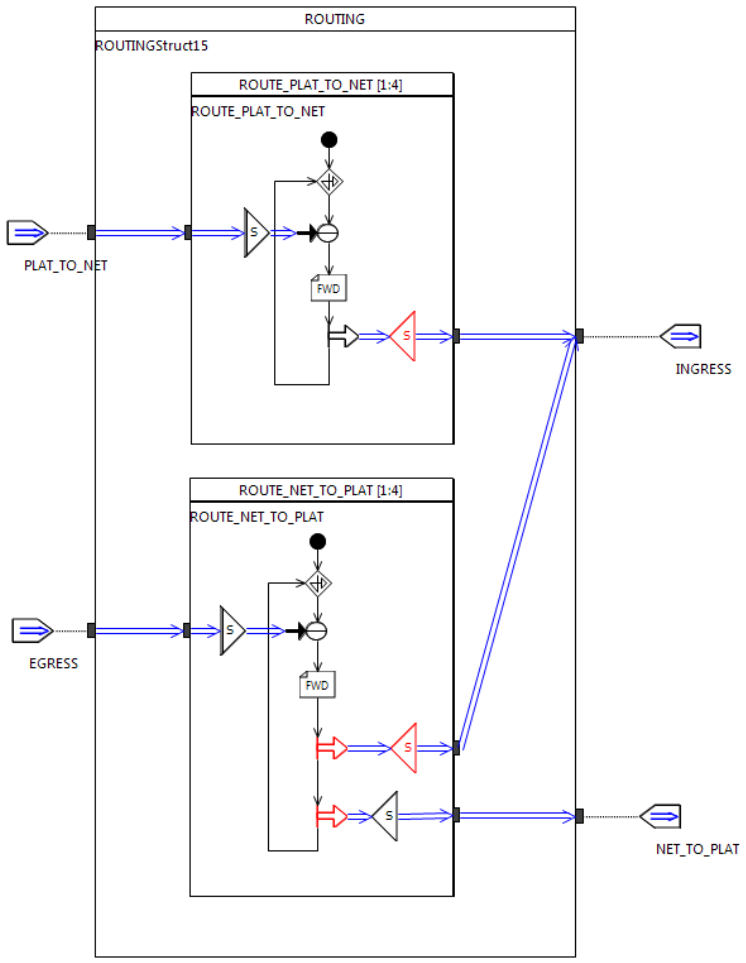


Figure 4.3: Figure of the routing block.

Figure 4.3 shows the routing block. The upper function sends data from the producer to corresponding switch. The lower function chooses whether data is sent from one switch to another or if it is ready to be sent to its destination, aka. the consumer.

4.1.4 Switch functions

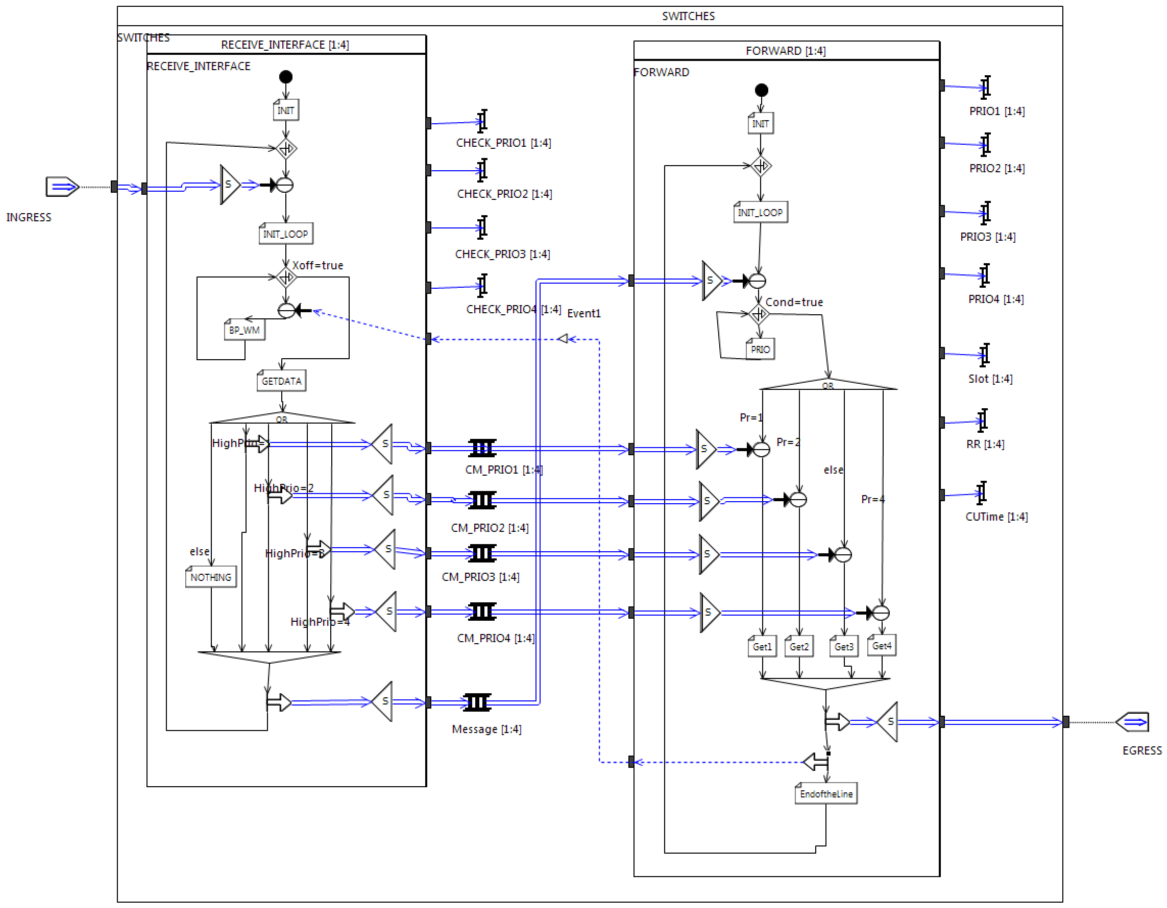


Figure 4.4: Figure shows the functions included in the switch block

Figure 4.4 shows an overview of the switch functions. The function to the left receives data and distributes it to the four intermediate queues, which represents the VC:s. The right hand block incorporates the scheduling functionality, where decisions are being made for what VC to admit data from. Figure 4.4 can be compared to figure 3.2 as it corresponds to the same component.

4.2 Simulation results

The simulations was done using four test cases, each case representing a different scheduling scenario SP, RR, TDM, ALG and CALG as discussed in chapter 2. All simulations were done using static input-files discussed in chapter 3.1.4 in order to achieve comparable results. The data-files represents a full TTI consisting of four non-overlapping high priority burst surrounded by a mix of low priority data.

Figures 4.5 to 4.11 show the latency of received packets after they have been sent through the network. The y-axis shows the latency of each packet in μs and the x-axis shows the elapsed time in μs.

Figures 4.12 to 4.18 show the throughput of received packets over a timeslot of 10 μs. The y-axis shows the throughput in Gbit/s and the x-axis shows the elapsed time in μs.

Figures 4.19 and 4.20 show parameter sweeps of the TDM where the timeslot window for Prio1 is increased, up from 200 ns where all timeslots are equal.

Figures 4.21 and 4.22, show parameter sweeps in which the N amount of times Prio1 data is sent is increased during simulation. This value ranges from one where all priorities are equally scheduled, to 300, where 300 packets of Prio1 are sent for every one of Prio2, Prio3, Prio4.

Red points shows the latency of Prio1 data, yellow Prio2 data, Prio3 data and gray Prio4 data. The y-axis shows the latency in μs and the x-axis shows time of arrival.

4.2.1 Latency

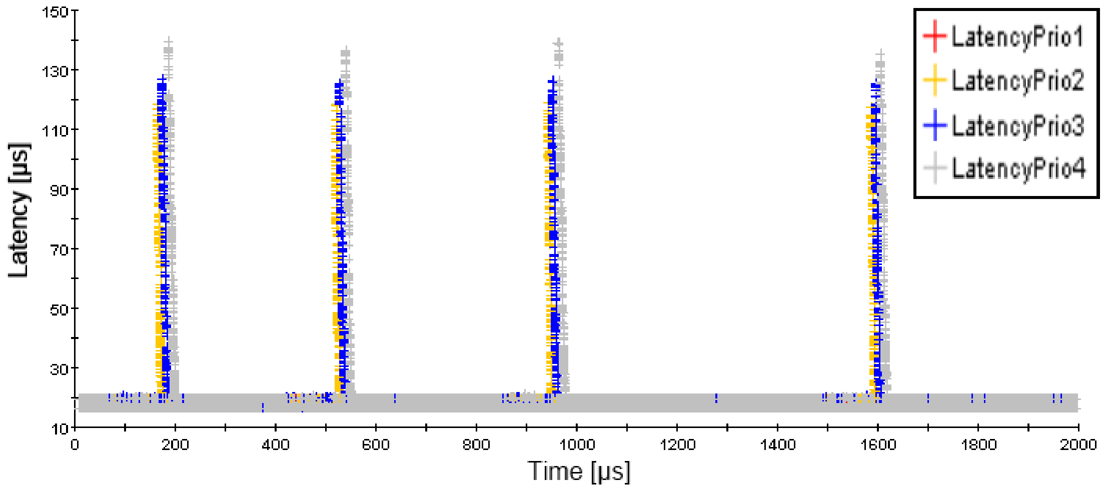


Figure 4.5: Simulation showing latency measurements for SP scheduling

In figure 4.5 high priority latency stays very low for the whole interval which makes it very hard to distinguish. The highest Prio1 latency is measured 19.2 μs, this can be seen more clearly in table 4.1. Lower priority data has a maximum latency 118.4 μs for Prio2 data, 127.8 μs for Prio3 data and 139.3 μs for Prio4 data.

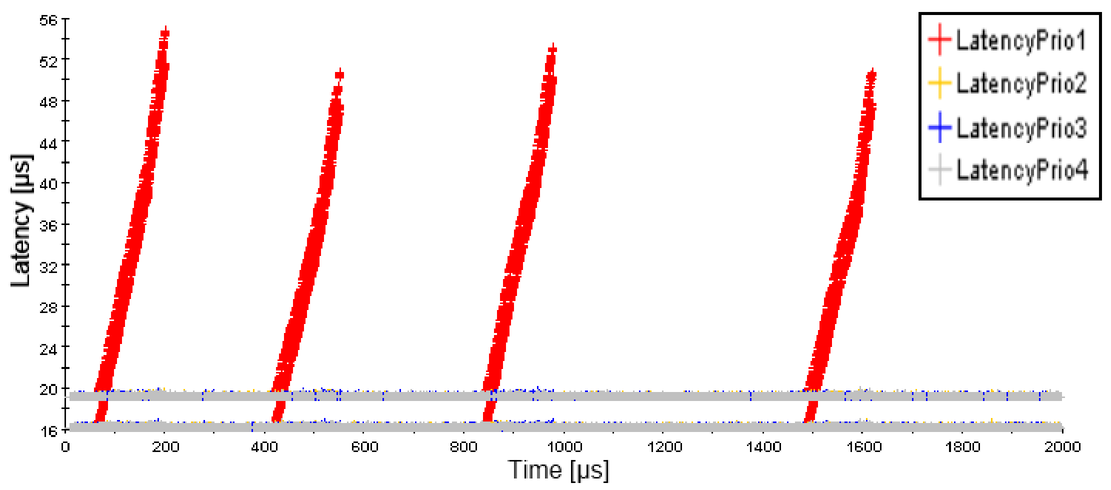


Figure 4.6: Simulation showing latency measurements for RR scheduling

Figure 4.6 shows a simulation of RR scheduling which yields high latency for Prio1 data due to the traffic bursts. Prio1 data has a maximum latency of 54.7 μs. Lower priority data have a maximum latency close to 19.5 μs.

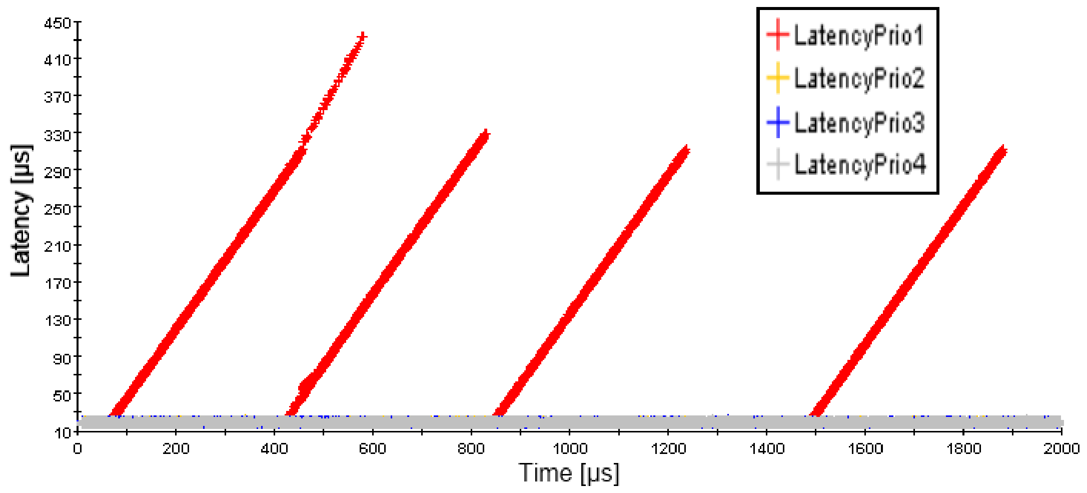


Figure 4.7: Simulation showing latency measurements for TDM scheduling with 200 ns for each timeslot

Figure 4.7 shows the latency of received packets when using TDM scheduling for timeslots of 200 ns each. Prio1 data has a maximum latency of 432.6 μs. Lower priority data have a maximum latency close to 21.8 μs.

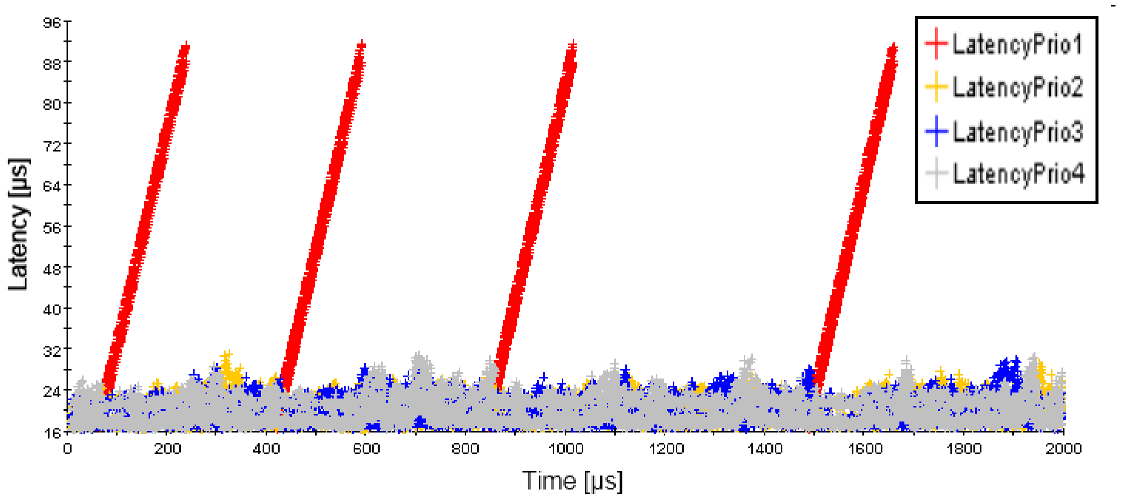


Figure 4.8: Simulation showing latency measurements for TDM scheduling with a 800 ns Prio1 timeslot

In figure 4.8 Prio1 timeslot has been increased from 200 ns to 800 ns. The result shows a decreased latency of 341.2 μs of Prio1 (compared to the initial simulation in 4.7) while the lower priorities increased close to 10 μs in average.

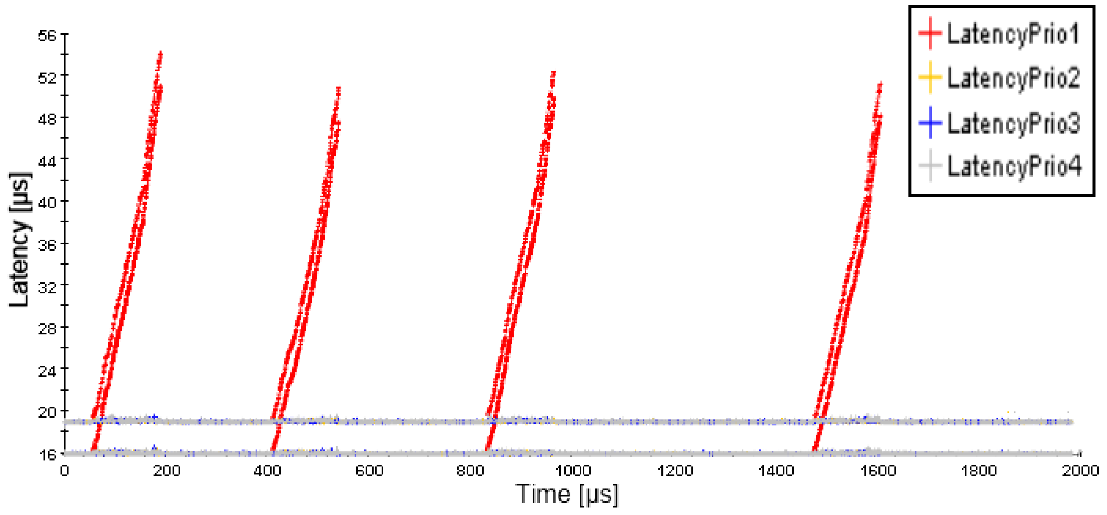


Figure 4.9: Simulation showing latency measurements for ALG scheduling

Figure 4.10 shows ALG scheduling. The result yields a maximum latency of 53.4 μs for Prio1, 19.4 μs for Prio2, 19.5 μs for Prio3 and 19.7 μs for Prio4. This result is roughly equivalent with RR scheduling.

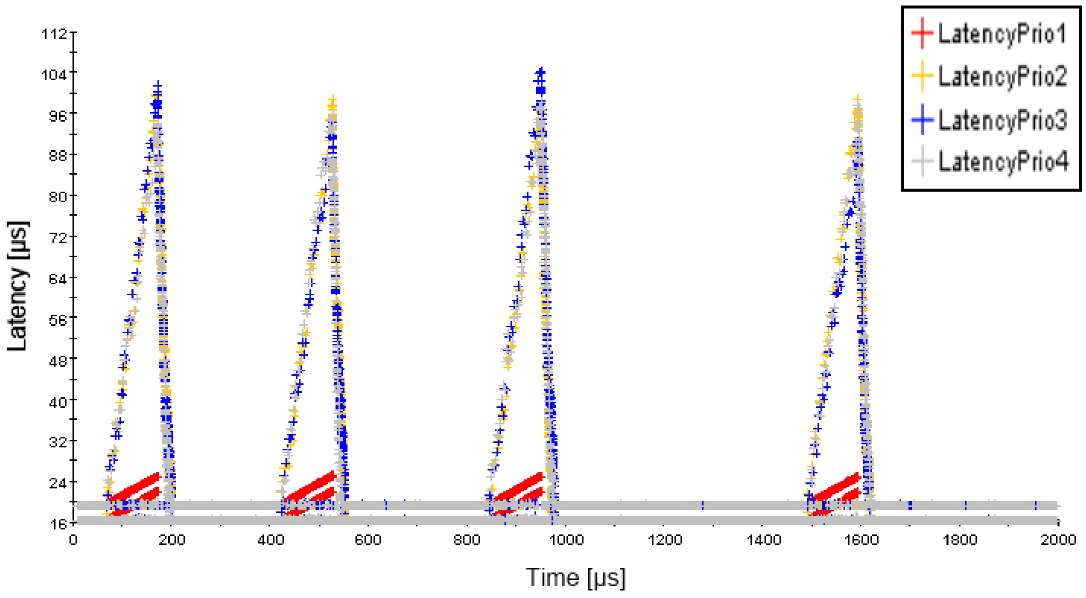


Figure 4.10: Simulation showing latency measurements for CALG scheduling with 25 μs maximum limit for Prio1

Figure 4.10 shows CALG scheduling with a modified maximum threshold for Prio1 latency of 25 μs. The result yields a maximum latency of 24.9 μs for Prio1, 100.1 μs for Prio2, 104.1 μs for Prio3 and 98.1 μs for Prio4. Compared to the SP scheduling the Prio1 latency has increased by 5.7 μs, while the lower priority latencies are decreased by 18.3, 22.6 and 41.2 μs respectively.

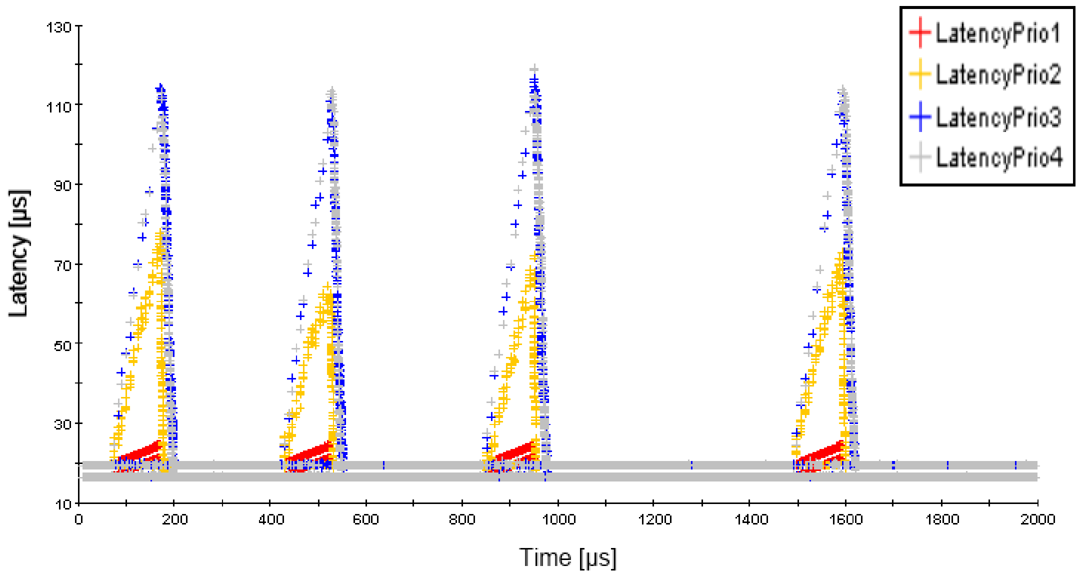


Figure 4.11: Simulation showing latency measurements for CALG scheduling with 25 μs maximum threshold for Prio1 and 80 μs for Prio2.

Figure 4.11 shows the latency of received packets when using CALG scheduling with modified maximum thresholds for both Prio1 and Prio2 latency. The result yields a maximum latency of 24.2 μs for Prio1, 77.7 μs for Prio2, 116.2 μs for Prio3 and 119.0 μs for Prio4.

By comparing to SP scheduling Prio1 latency has increased by 5.0 μs, while the lower priority latencies are decreased by 40.7, 10.5 and 20.3 μs respectively. When comparing to CALG with a singel threshold, figure 4.10, one can see that when introducing a second threshold of 80 μs for Prio2 the Prio3 and Prio4 increased by 12.1 and 19.8 μs respectively.

4.2.2 Discussion for latency measurements

The initial SP scheduling measurements shows high latency spikes for lower priority data while keeping significantly lower latencies for the Prio1 data. SP scheduling strives to minimize the latency of Prio1 bursts by giving it precedence to the physical channel, meanwhile all lower priority data gets delayed in their corresponding VC's while the burst gets processed. The spikes shown in the figures are then caused when the VC's are getting emptied out to physical channel.

Measurements of RR shows that the dense bursts of Prio1 have significantly higher latency compared to all lower priorities, this was expected since RR distributes bandwidth equally among all priorities. It can also be seen that by having equally distributed bandwidth the latencies of lower priorities obtains a lower average in general compared to SP. However, Since Prio1 data is time-critical this is not a suitable scheduling discipline for these traffic characteristics [14].

From the section of the TDM scheduling it can be seen that two simulations were made; One using equally long timeslots of 200 ns for every priority; the second using 800 ns for Prio1 timeslot. In general it can be seen that TDM yields high latencies for Prio1, this is mainly caused by the long length and high density of these bursts in proportion to the dedicated timeslot. Even for the second simulation, where Prio1 utilizes a four times larger timeslot, the results yields improvements but these are not sufficient. One can conclude that TDM scheduling is not preferable for the given traffic case due to its low efficiency [17].

The measurements of the CALG were done in three stages; firstly a ALG was performed; secondly the CALG was modified met a GL for Prio1 of 25 μs; lastly the CALG was modified to meet a GL of 25 μs for Prio1 and a GL of 80 μs for Prio2.

The results from the ALG shows performance that is very close to RR, this could be expected since unchanged ALG and RR are both very fair scheduling disciplines.

For the second simulation the CALG parameter N1=50 in order to obtain a GL for Prio1 of 25 μs. As expected, the results of lowering the time-critical Prio1 data yielded an increase of BE latencies for the lower priorities. Comparing with SP scheduling, the results show relaxing the GL of Prio1 a significant gain in BE latency. One can conclude from the results above that for bursty traffic characteristics CALG has better performance than ALG regarding GL. For these traffic characteristics this is a more suitable scheduling the ALG

The third simulation displays a test for two time critical priorities where a GL of 25 μs for Prio1 and a GL of 80 μs for Prio2 is performed. Compared to having only one threshold, the results show even further increase of BE latencies in the lower priorities. This, however, allows for a GL service for the Prio1 and Prio2.

4.2.3 Throughput

The figures show the throughput related to previous simulations. Red points shows the throughput of ASIC one, yellow points shows the throughput of ASIC two, blue points shows the throughput of ASIC three, gray points shows the throughput of

ASIC four. The y-axis shows the throughput in Gbit/s and the x-axis shows time of arrival.

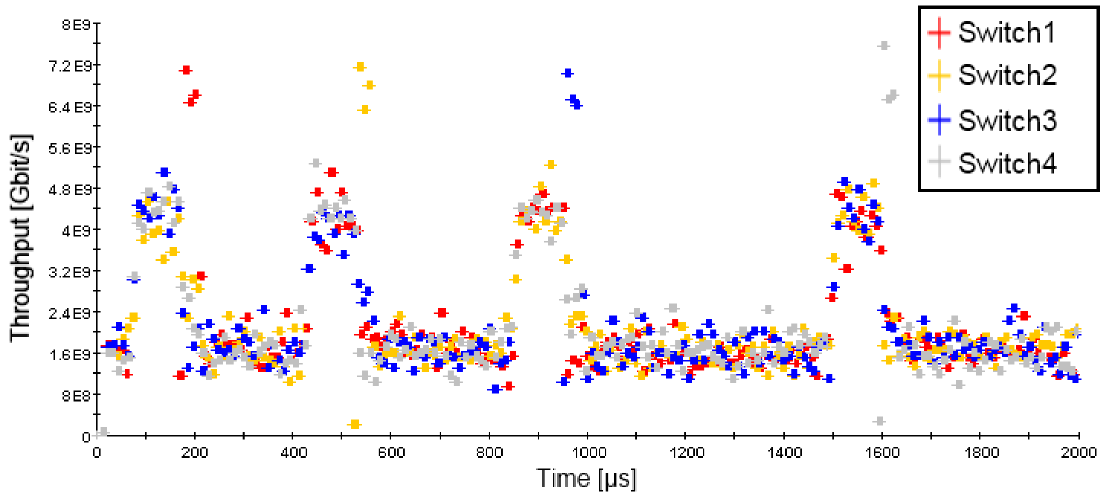


Figure 4.12: Simulation showing throughput for strict priority scheduling

Figure 4.12 shows the throughput of each ASIC while using SP scheduling.

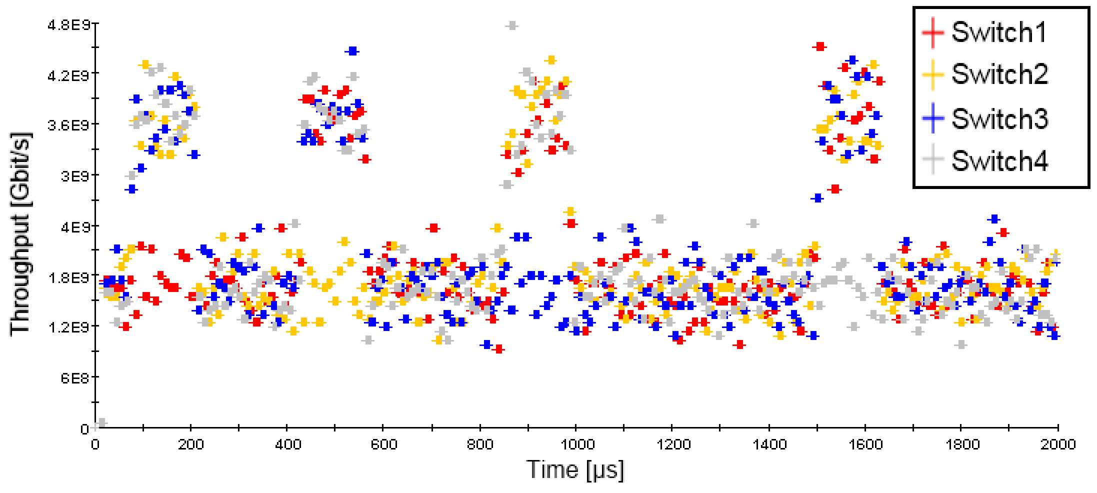


Figure 4.13: Simulation of Round Robin scheduling

Figure 4.13 shows the throughput of each ASIC while using RR scheduling.

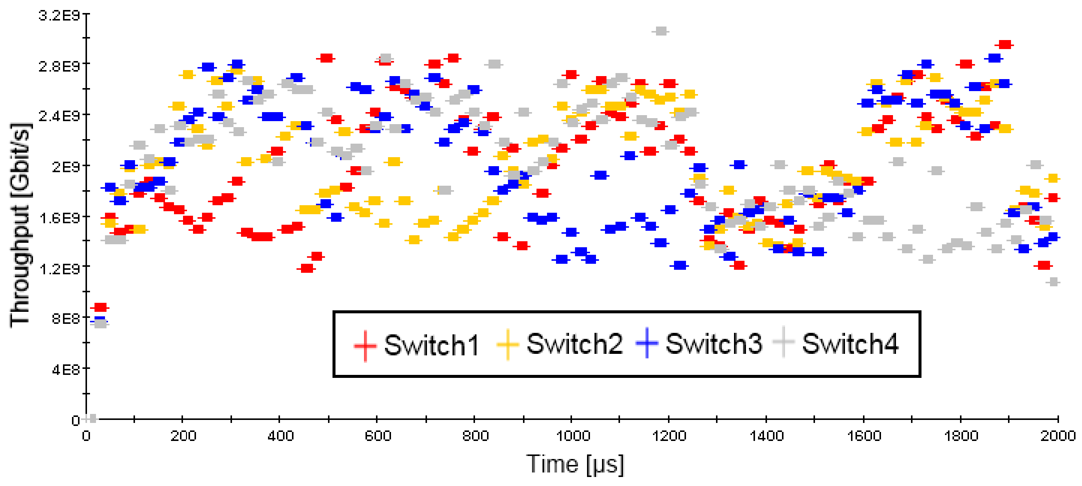


Figure 4.14: Simulation of TDM scheduling, showing throughput

Figure 4.14 shows the throughput of each ASIC while using TDM scheduling. All timeslots are 200 ns long.

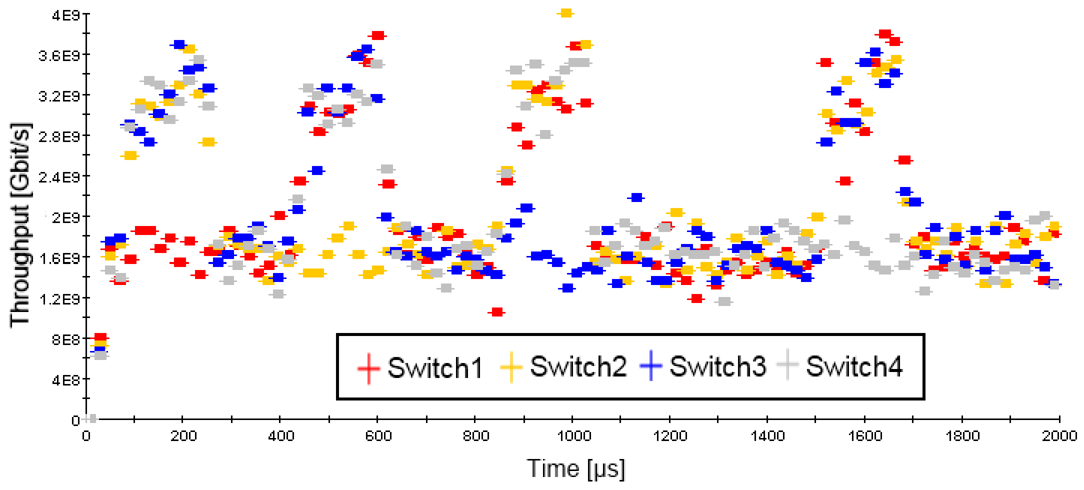


Figure 4.15: Simulation of TDM scheduling with an 800 ns timeslot, showing throughput

Figure 4.15 shows the throughput of each ASIC while using TDM scheduling. Prio1:s timeslot is 800 ns long while other priorities have a 200 ns timeslot. Red points shows the throughput of ASIC one, yellow points shows the throughput of ASIC two, blue points shows the throughput of ASIC three, gray points shows the throughput of ASIC four.

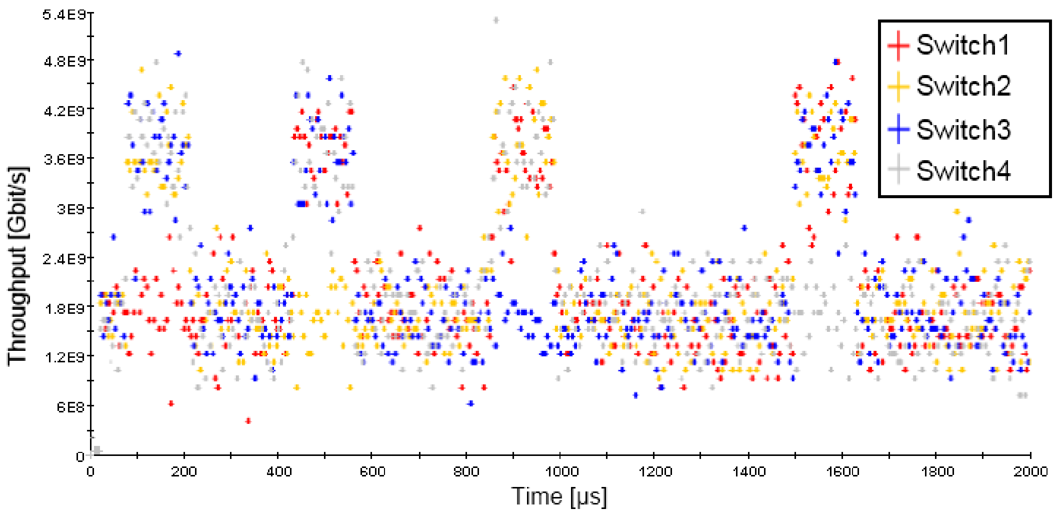


Figure 4.16: Simulation of ALG scheduling, showing throughput

Figure 4.17 shows the throughput of each ASIC while using ALG scheduling.

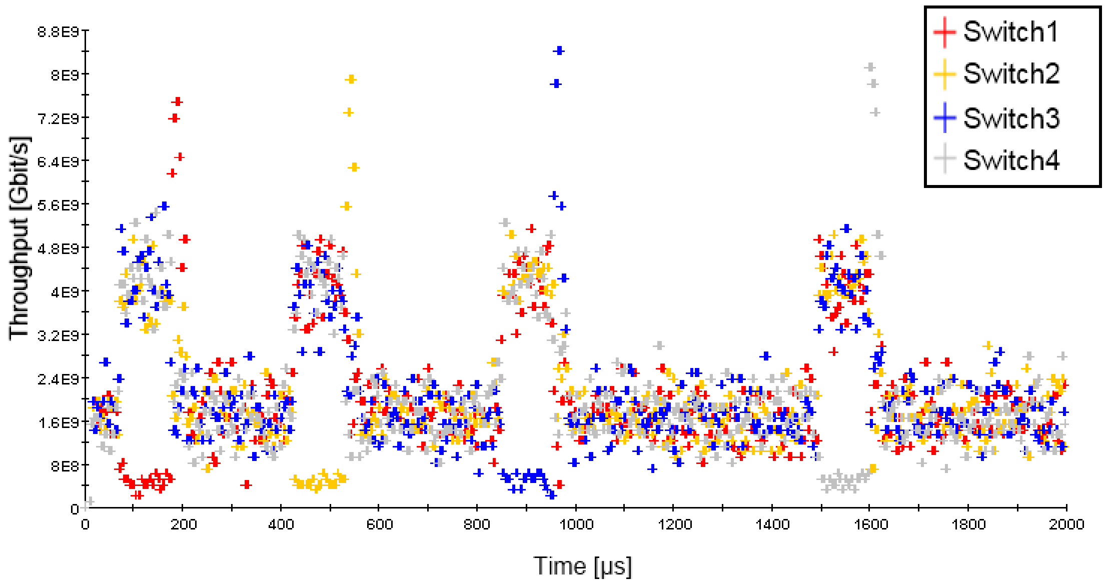


Figure 4.17: Simulation of CALG scheduling with a 25 μs limit of Prio1, showing throughput

Figure 4.17 shows the throughput of each ASIC while using CALG scheduling.

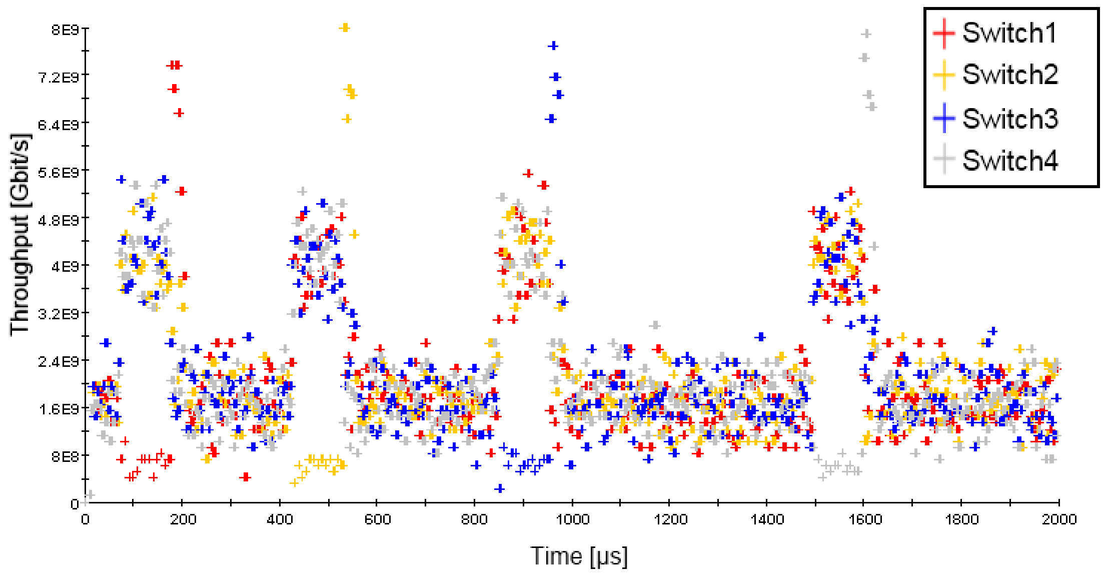


Figure 4.18: Simulation of modified CALG scheduling with a 25 μs and 80 μs limit of Prio1, showing throughput

Figure 4.18 shows the throughput of each ASIC while using CALG scheduling.

To conclude, by examining the peaks during Prio1 burst interval for each scheduling type one can see that the average throughput does not vary to a great extent. The only exception is for the significantly low throughput of TDM scheduling which is known to have low spectral efficiency [REF].

It can also be seen in figure 4.12, 4.17 and 4.18 there is an increase in bandwidth after the Prio1 burst has ended. This is explained by the traffic sent to ASIC from which the burst originates. Since Prio1 is prioritized lower priority data will remain in the VC buffer. Once the burst is complete this data is sent, meaning the ASIC will receive traffic at almost line rate.

4.2.4 Parameter sweep

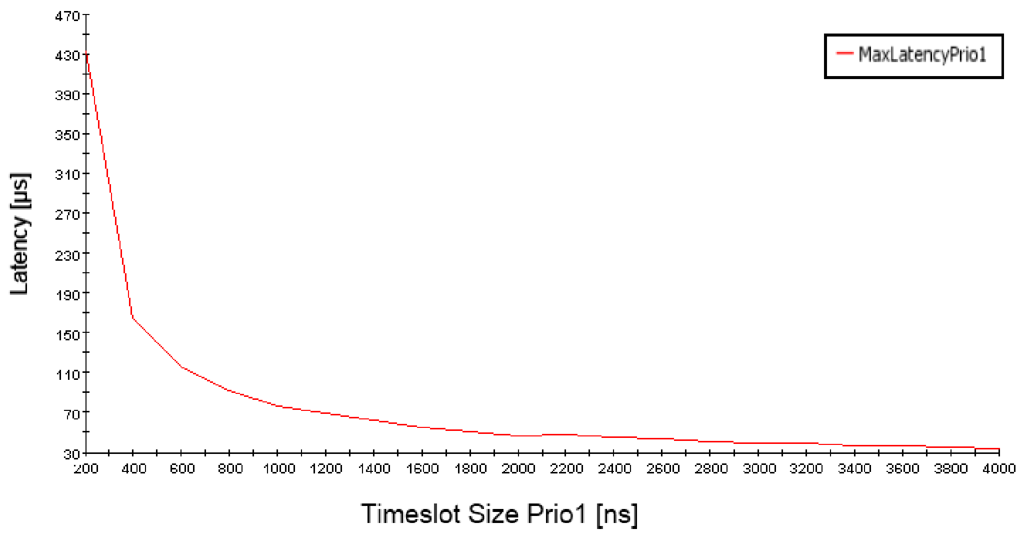


Figure 4.19: Sweep of Prio1 timeslot.

Figure 4.19 shows a parameter sweep of the Prio1 timeslot. The y-axis shows the latency in μs and the x-axis shows the Prio1 timeslot in ns.

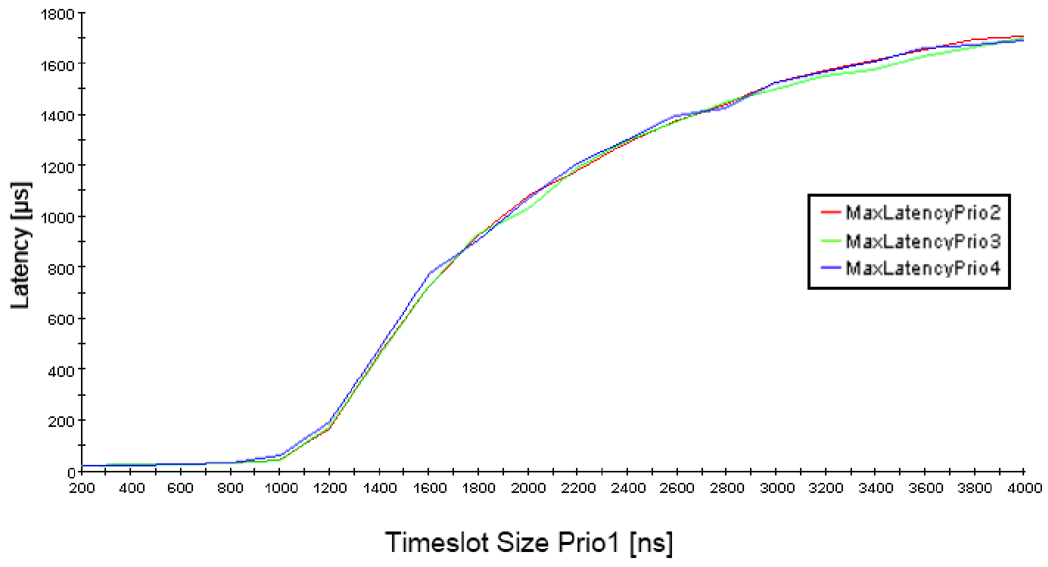


Figure 4.20: Sweep of Prio2-Prio4 timeslot.

Figure 4.20 shows a parameter sweep of the Prio2-Prio4 timeslot. The y-axis shows the latency in μs and the x-axis shows the Prio1 timeslot in ns.

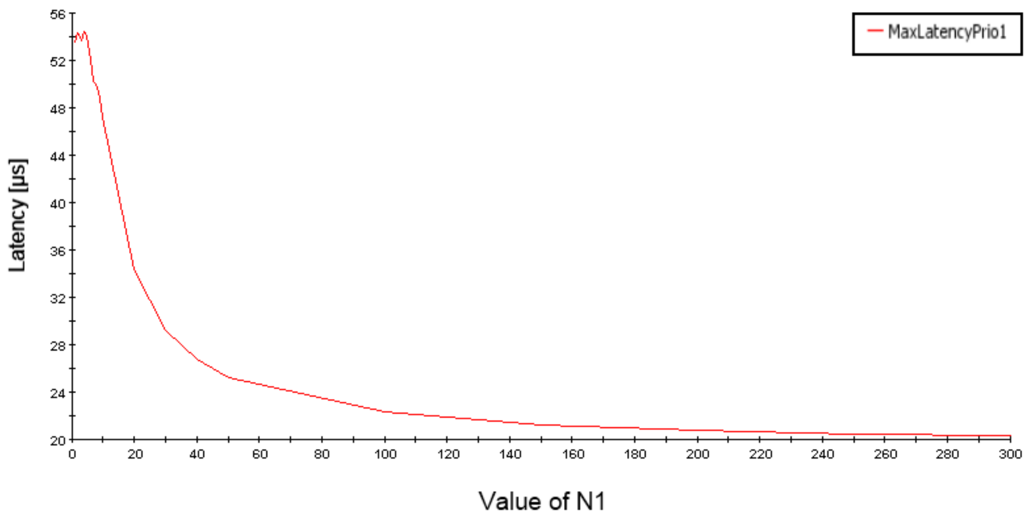


Figure 4.21: Sweep of ALG, showing Prio1 latency. X-axis shows the relative bandwidth of Prio1 data.

Figure 4.21 shows a parameter sweep of the N variable where the amount of times Prio1 packets are sent per one packet of lower priority data. The y-axis shows the Prio1 latency and the x-axis shows the N variable.

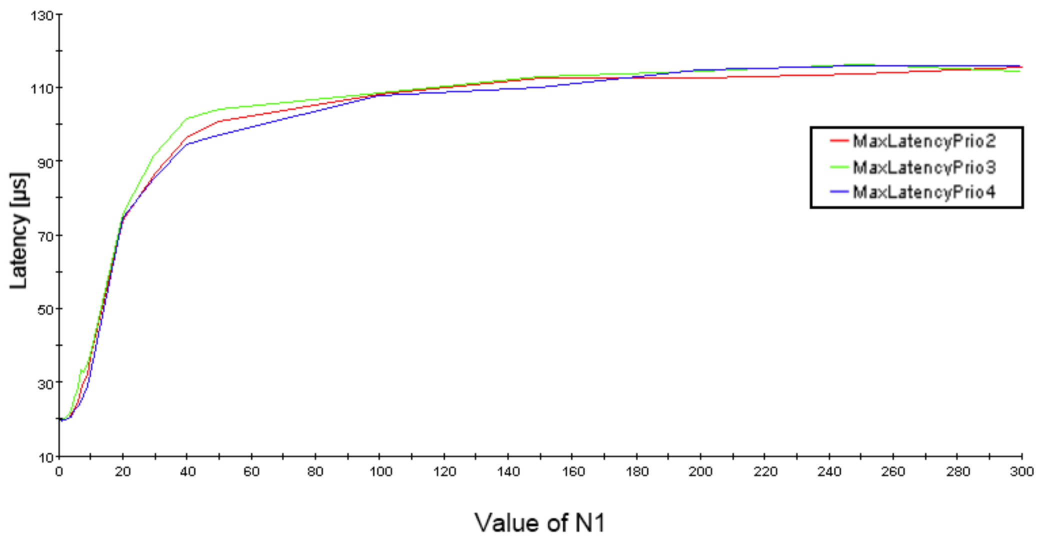


Figure 4.22: Sweep of CALG, showing Prio2-Prio4 latency. X-axis shows the relative bandwidth of Prio1 data.

Figure 4.22 shows a parameter sweep of the N variable where the amount of times Prio1 packets are sent per one packet of lower priority data. The y-axis shows the Prio2-Prio4 latency and the x-axis shows the N variable. This ability to configure the value of N shows a clear advantage over the original ALG which is equivalent to N=1.

Figure 4.21 to 4.22 shows the highest measured latency of each data priority when Prio1 data gains increasing bandwidth. From these results a suitable value of N can be selected in order to achieve a latency within requirements while improving BE latencies. A maximum latency of 25 μs was desired. A value of N was chosen from figure 4.21. Figure 4.10 shows the results of using an N of 50.

4.2.5 Tables

Table 4.1 and 4.2 display the highest latencies and throughputs attained from the figures in this chapter.

Results	ALG	RR	TDM	CALG 25 μ limit	SP
Max latency Prio1 [μs]	53.4	54.7	91.4	24.9	19,2
Max latency Prio2 [μs]	19.4	19.5	30.9	100.1	118,4
Max latency Prio3 [μs]	19.5	19.6	29.6	104.1	126,7
Max latency Prio4 [μs]	19.7	19.6	30.6	98.1	139,3
Burst throughput [Gbit/s]	5.4	4.8	4.0	5.5	5.2

Table 4.1: Table of results comparing ALG, RR and TDM

Table 4.1 shows the highest attained values from ALG, RR and TDM scheduling, viewable in figures 4.9, 4.6 and 4.7.

Results	SP	CALG 25μ limit	CALG 25μ/80μ limits	ALG
Max latency Prio1 [μs]	19.2	24.9	24.2	53.4
Max latency Prio2 [μs]	118.4	100.1	77.7	19.4
Max latency Prio3 [μs]	126.7	104.1	116.2	19.5
Max latency Prio4 [μs]	139.3	98.1	119.0	19.7
Burst throughput [Gbit/s]	5.2	5.5	5.6	5.4

Table 4.2: Table of results comparing SP, CALG and ALG

Table 4.2 shows the highest attained values from SP scheduling and CALG, which can be viewed in figures 4.5, 4.10 and 4.11. Here the changed distribution in latency is clearly visible.

Measurements were also done using a total of 40% of linerate per TTI rather than 80%. The results of these measurements are displayed in table 4.3.

Results	SP	RR	TDM	CALG 24μ limit	ALG
Max latency Prio1 [μs]	19.2	26.5	55.0	23.6	26.2
Max latency Prio2 [μs]	68.2	19.2	22.9	37.8	19.2
Max latency Prio3 [μs]	68.9	19.2	23.4	39.0	19.2
Max latency Prio4 [μs]	72.4	19.2	23.1	45.4	19.3
Burst throughput [Gbit/s]	4.9	4.4	2.3	4.4	4.4

Table 4.3: Table of results 40%

5

Conclusions

This thesis work has proposed an configurable scheduling discipline based on the existing scheduler named Asynchronous Latency Guarantee (ALG). The proposed Configurable Asynchronous Latency Guarantee (CALG) allows adjusting bandwidth of prioritized data. The CALG was implemented in a software model made from specifications given by Ericsson along with strict priority, Round Robin and Time-division Multiplexing and Asynchronous Latency Guarantee. The specifications requested to utilize a strict priority scheduling in a network consisting of four switches, each connected to an ASIC.

A model of the system started to be implemented in Ptolemy II. This model was however discontinued in favour of Cofluent Studio. Ptolemy II was very difficult for implementing advanced systems and very confusing for anyone who was unfamiliar with the project.

The model was tested with four different scheduling types using an input stimuli of uniformly distributed data with bursts of time-critical data. The measurements focused on the highest measured latency and bandwidth. Measurements using SP scheduling yielded a latency of 19.2, 118.4, 127.8, 139.3 μs respective to each data priority.

When corresponding simulations were done using RR scheduling Prio1 data had a maximum latency of 54.7 μs. Lower priority data had a maximum latency close to 19.5 μs. This was an improvement in low priority data, but unsuitable for the setup where priority one data is time-critical. TDM measurements yielded a latency of 91.4 μs for highest priority data and roughly 30 μs for lower priority data. The low efficiency of TDM made it unsuitable for burst traffic which resulted in higher latency in all respects. ALG simulations yielded a latency of 53.5 μs for Prio1 and 19.4 μs, making it roughly equivalent with RR scheduling.

CALG which is a modification of ALG by the authors was simulated with a GL. The first simulation was done with a 25 μs threshold on Prio1 data. This limit was met with a latency of 24.9 μs and 100.1 μs for Prio2, 104.1 μs for Prio3 and 98.1 μs for Prio4. Compared to SP scheduling, a loss of 5.7 μs resulted in a gain of 18.3, 22.6 and 41.2 μs, which means the loss is 20.8% of the average gain.

A second threshold was added onto Prio2 of 80 μs. This yielded results of 24.2 μs for Prio1, 77.7 μs for Prio2, 116.2 μs for Prio3 and 119.0 μs for Prio4. This allowed for Prio2 data to have a reduced latency but at the cost of Prio3 and Prio4 latency. This also implies that setting more strict limitations GL will decrease the gain in BE latency.

The original ALG has a 178.1% higher GL compared to the default SP scheduling. The configurable parameter of CALG showed that by relaxing GL to 25 μs its latency became 29.6% higher than the GL of default SP scheduling. This shows that the CALG is more suitable than ALG for bursty data traffic.

This thesis aimed to create a model of the specified network and investigate improvements in QoS. A model was created in Cofluent studio and different scheduling disciplines have been compared to SP scheduling which was specified in the network. By using CALG the authors found that it was possible to relax GL of time-critical data in order to receive a comparably large gain in BE latency.

5.1 Future work

Since the models created are entirely designed by the authors, it would be useful to make an implementation of the modelled system in real hardware. This would verify the accuracy of the model and verify the theory. An alternative would be to compare it to measurements from previous versions of the system to see how accurate the base model is.

In order to make a fair comparison between scheduling disciplines input stimuli came from an input file. This was a very static approach which made it difficult to make significant changes to the input between tests. A different solution is to implement the stimuli directly in CoFluent, creating data with probability or variables. This allows a more dynamic ability to manipulate parameters when simulating.

Our simulations only include two flows of data. Including more flows would make the simulations more realistic compared to baseband chips which include several different flows. Furthermore the ASIC models are very simplistic. The ASIC model is simply producing data and queues it up when there is no space. The ASIC could be modelled more in depth to make them interact better with the network.

Ericsson have design tools which is used for simulating data and setting requirements in models using UML. Making the CoFluent model able to communicate with these design tools would make it possible to verify communication requirements. It could be possible to use this to optimize component placement on chips [23].

Bibliography

[1] E. Mollick, Establishing moore's law, Annals of the History of Computing, IEEE 28 (3) (2006) 62–75.

[2] E. Bolotin, I. Cidon, R. Ginosar, A. Kolodny, Qnoc: Qos architecture and design process for network on chip, Journal of Systems Architecture 50 (2) (2004) 105–128.

[3] T. Bjerregaard, J. Sparso, Scheduling discipline for latency and bandwidth guarantees in asynchronous network-on-chip, in: Asynchronous Circuits and Systems, 2005. ASYNC 2005. Proceedings. 11th IEEE International Symposium on, IEEE, 2005, pp. 34–43.

[4] W. Dally, B. Towles, Principles and practices of interconnection networks, Morgan Kaufmann, 2004.

[5] J. Kurose, K. Ross, Computer networking, Pearson/Addison Wesley, 2008.

[6] F. Wartenberg, DUS switch model, Ericsson AB.

[7] D. Andreasson, S. Kumar, On improving best-effort throughput by better utilization of guaranteed-throughput channels in an on-chip communication system, in: Proceeding of 22th IEEE Norchip Conference, 2004.

[8] E. Rijpkema, K. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, E. Waterlander, Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip, in: Computers and Digital Techniques, IEE Proceedings-, Vol. 150, IET, 2003, pp. 294–302.

[9] E. Rijpkema, K. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, E. Waterlander, Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip, Philips Research Laboratories, Eindhoven, The Netherlands, 2003.

[10] J. Dielissen, A. Radulescu, K. Goossens, E. Rijpkema, Concepts and implementation of the philips network-on-chip, in: IP-Based SoC Design, 2003, pp. 1–6.

[11] A. Hansson, K. Goossens, A. Rădulescu, A unified approach to mapping and routing on a network-on-chip for both best-effort and guaranteed service traffic, VLSI design 2007.

[12] Y. Qian, Z. Lu, Q. Dou, Qos scheduling for nocs: Strict priority queueing versus weighted round robin, in: Computer Design (ICCD), 2010 IEEE International Conference on, IEEE, 2010, pp. 52–59.

[13] L. Kleinrock, Queueing systems volume 2: Computer applications, Recherche 67 (1976) 02.

[14] L. Ji, T. Arvanitis, S. Woolley, Fair weighted round robin scheduling scheme for diffserv networks, Electronics Letters 39 (3) (2003) 333–335.

[15] A. Parekh, R. Gallager, A generalized processor sharing approach to flow control in integrated services networks: the single-node case, IEEE/ACM Transactions on Networking (TON) 1 (3) (1993) 344–357.

[16] S. S. A. Demers, S. Keshav, Analysis and simulation of a fair queuing algoithm, Internetworking: Research and Experience 1 (1) (1990) 3–26.

[17] R. Chipalkatti, Z. Zhang, A. Acampora, Protocols for optical star-coupler network using wdm: Performance and complexity study, Selected Areas in Communications, IEEE Journal on 11 (4) (1993) 579–589.

[18] H. Stern, S. Mahmoud, Communication Systems: Analysis and Design, Prentice-Hall, Inc., 2003.

[19] J. Proakis, M. Salehi, N. Zhou, X. Li, Communication systems engineering, Vol. 2, Prentice-Hall, 1994.

[20] Ptolemy ii faq (02-10-2012).
URL `http://ptolemy.eecs.berkeley.edu/ptolemyII/ptIIfaq.htm`

[21] Cofluent Design, CoFluent Studio user's guide (August 2010).

[22] R. Durrett, Probability: theory and examples, Cambridge University Press, 2010.

[23] IBM, Rational software architect family of products information centers (02-10-2012).
URL `http://www-01.ibm.com/support/docview.wss?uid=swg27010908`

[24] D. J. Paul, Moore's law (01-08-2012).
URL `http://www.sp.phy.cam.ac.uk/~SiGe/Moore's%20Law.html`

[25] T. Bjerregaard, J. Sparso, A router architecture for connection-oriented service guarantees in the mango clockless network-on-chip, in: Design, Automation and Test in Europe, 2005. Proceedings, IEEE, 2005, pp. 1226–1231.

[26] U. Bergström, WCDMA calculation sheet, Ericsson AB.