The Mic-1IO Architecture

Modification of the Mic-1 architecture to support peripheral devices

Written by
Esben Rugbjerg & Anders Markussen
esrug@ehp.dk & amark@diku.dk

Contents

List of Figures

Abstract

This report discusses an extension of the Mic-1 architecture in a way that enables I/O interaction between the CPU and peripheral devices on a bus. Furthermore the IJVM instruction set described by Tannenbaum [Tan99] is extended to implement some simple I/O instructions. The peripherals are connected in a daisy chain with the CPU. A simple protocol to control the data communication on the bus has been developed as well. The new architecture has been implemented in the hardware description language Gezel and the Mic-1 simulator made by Ray Ontko has been modified to support the I/O extension and to include two peripherals.

Preface

This report summarises the work of a bachelor thesis (i.e. the polytechnical midway project on DTU). The counsellors on the project was professor Jan Madsen and associate professor Jørgen Stensgaard-Madsen which we would like to thank. We would also like to thank Patrick Schaumont, the developer of the Gezel hardware description language, for his quick responses to our questions concerning issues on the Gezel hardware description language.

CHAPTER 1

Introduction

In the book Structured Computer Organisation [Tan99], Andrew S. Tanenbaum proposes a simple architecture called the Mic-1 with the ability to execute programs written in a subset of the Java Virtual Machine (JVM) instruction set. This subset is called Integer Java Virtual Machine (IJVM) since it only contains integer instructions. The IJVM instruction set contains no I/O instructions and the description of the Mic-1 architecture contains no precise description of ways the CPU could interact with any external devices except from a note telling that I/O could be done through method calls [Tan99, p. 226; first paragraph].

The primary goal of this report is to end up with a solution to the problem, both simple and flexible, meeting the requirements by Jan Madsen and Jørgen Stensgaard-Madsen enabling them to use it while teaching the student at course 02130 - Introduction to Codesign.

One of the main obstacles in making the IJVM support I/O interaction is the memory model which uses a relative addressing scheme where the precise memory addresses are hidden from the programmer in the micro code layer. This fact does not prevent I/O based on the memory, but limits the opportunities of using the memory as the interface for communicating with peripheral devices.

In order to make the CPU able to communicate with external devices in a simple manner, an extension of the IJVM instruction set is proposed. These extensions include the addition of some input and output instructions to the IJVM Instruction set. These instructions should give the IJVM programmer the ability to communicate with peripheral devices connected to the CPU executing the IJVM code doing absolute addressing of the peripherals.

In addition to the extension of the IJVM instruction set, the Mic-1 architecture described by Tannenbaum [Tan99] is modified with some extra I/O registers making it able to connect to devices on a developed I/O bus using a protocol described later on in this report.

The decisions made in this report are based on a number of points of view.

First of all the solution to the problem should be easily understandable for inexperienced users and developers of instruction sets and micro architectures (i.e. students on the DTU course: 02130 - Introduction to Codesign). This is achieved by proposing a solution where the Mic-1 architecture is changed as little as possible, or at least in such a way that the major concepts in the architecture are maintained. This consideration is of course also applied to the extension of the IJVM instruction set architecture.

Furthermore the decisions made should end up in a relatively reasonable design considering the ways similar tasks have been done in practice and considering the limitations given by the above and the Gezel hardware description language.

CHAPTER 2

I/O model and analysis

This chapter describes major issues concerning the design of I/O systems for microcomputers. The different approaches discussed include port based I/O, memory mapped I/O and daisy chain I/O.

The pros and cons of the different designs are discussed, leading to our final design decision. Both the required extensions of the Mic-1 architecture and the IJVM ISA are discussed. Furthermore the consequences of changes in the Mic-1 architecture concerning the IJVM ISA and vice versa will be discussed.

The structure of this chapter will be as follows: First the Three major concepts memory mapped I/O, daisy chain based I/O and port based I/O are introduced. Next possible designs for the I/O system is discussed based on the three concepts. Finally our solution is proposed and described. The IJVM and Mic-1 architecture are discussed separately in two sections.

2.1 Memory-mapped I/O

The concept of memory-mapped I/O is summarised in this section. Further reading on the subject can be found in the book Computer Organization and Design [PH98, Page 675].

The main concept of memory mapped I/O is to route I/O traffic through certain memory addresses or sub areas of the memory space to the external devices. This means that data written into this area or to these addresses is not stored in the memory but in some way handled by some device not being a part of the memory.

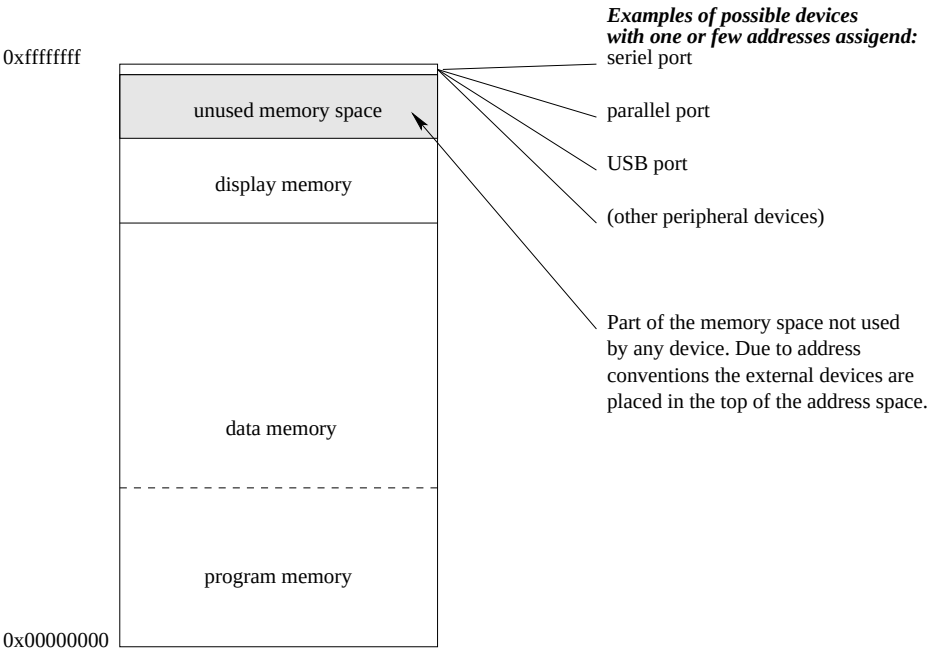

Figure 2.1: *Example of memory map of system with memory mapped I/O. The memory space is divided into several subsections where one subsection is used to address the real memory and some subsections are used to connect external devices. To the left the address limits in the address space are written. The example is constructed for illustration only and do not refer to a specific system.*

The CPU is connected to the memory and the peripherals on a bus and the individual devices react to the requests depending on whether the address which is sent out on the bus is a part of the address space belonging to the device in question.

Arbitration of a memory mapped I/O system can be done in many ways. However this topic will not be discussed further in this text. Tannenbaum gives a short introduction on this topic in Structured Computer Organization [Tan99, Section 3.4.5].

The major advantage of memory mapped I/O is that the communication between the CPU and the I/O devices is done using the same instructions (i.e. I/O are done using regular memory read/write instructions). This simplifies the instruction set since no special instructions for I/O communication are needed.

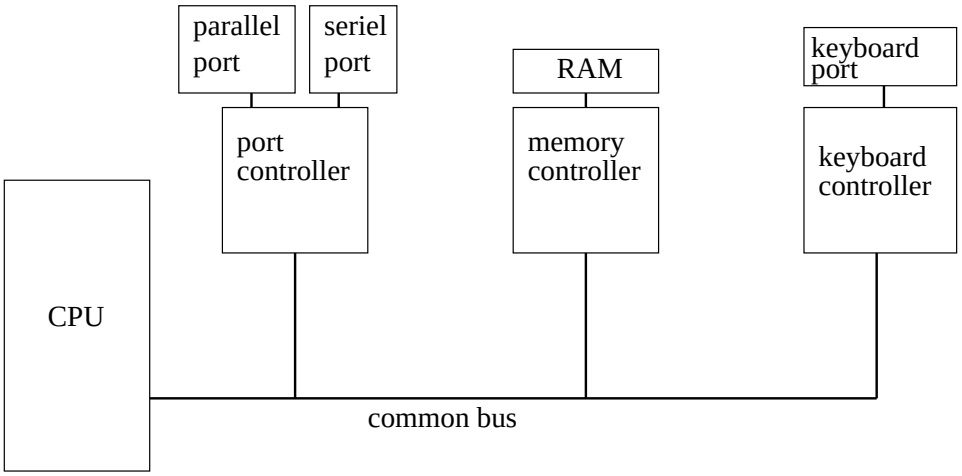

Figure 2.2: *Figure showing the schematic of a system where the bus is shared among the different devices which the CPU should be able to connect to. Of course other devices could be connected too.*

2.2 Daisy-chain I/O

The concept of daisy-chained I/O is summarised in this section. Further reading on the subject can be found in the book Structured Computer Organization [Tan99, Pages 165-167].

In a daisy chain, the devices are connected in a chain. Only one device is connected to the CPU and the second device is connected to the device connected to the CPU but not directly to the CPU. This means that a request sent to a device in the chain has to propagate through the other devices located between the CPU and the device.

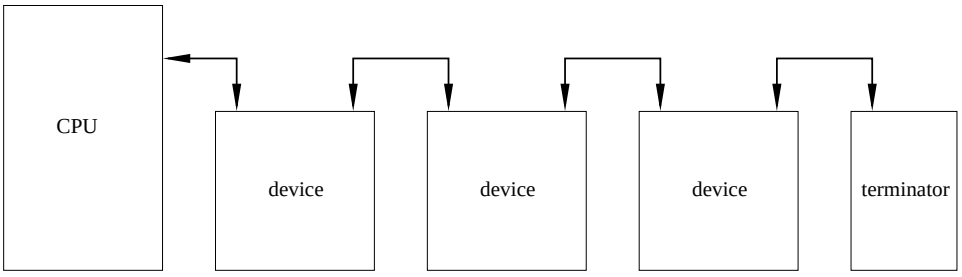

Figure 2.3: *A daisy chain topology based bus.*

This property is exploited in the arbitration of the system. If the devices are connected in a chain based on their individual priority with the highest prioritised device connected directly to the CPU and the least prioritised device connected as the last link in the chain, the communication protocol

for the chain could be made such that the devices are allowed to answer requests when a grant signal is raised by the CPU and this propagates out through the devices until it reaches a device which is to write something back to the CPU. This device does not forward the signal, but writes its data back instead. By placing the highest prioritised device closest to the CPU it is insured that this device always receives the grant signal first and therefore gets highest write priority.

One of the advantages of the daisy chain is the simple arbitration scheme described above. Besides that it is easy to make the system scalable, adding and removing devices from the chain.

One of the drawbacks of the system is the fact that all data and requests sent between the most distant device and the CPU have to propagate forth and back through all the other devices.

2.3 Port based I/O

Finally a port based I/O system is used in some architectures. The main idea in this concept is that the CPU has a number of ports which it is possible to read from and write to. One peripheral should be connected to each port. The 8051 architecture (and derived architectures) is known to have this architecture but the concept is not described in either [PH98] or [Tan99], which leads to the assumption that the concept is not seen as a general design concept despite its use in the popular 8051 CPUs. It is taken into consideration in the project due to its presence in the 8051. A full description of the port based I/O in the 8051 can be found in [Ste93, sec 1.4 and 2.6]

An advantage of this system is that the addressing of the peripherals is done simply by choosing the port it is connected to. A port can be represented by a register in the data path and if the computer is directly programmed as it is the case in the 8051, I/O operation can be performed simply by reading and writing from and to the register. A disadvantage is that the number of ports on the CPU limits the number of peripherals that can be connected, making scalability hard to achieve.

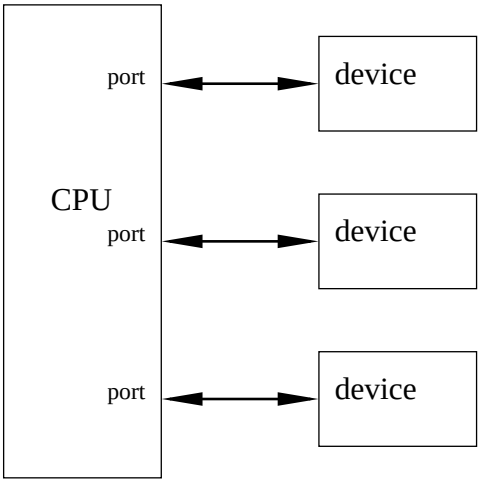

Figure 2.4: *A port based I/O system. No real bus is present since every connected device has its own connection to the CPU.*

2.4 Requirements of I/O system

Several issues should be considered during design of the I/O system. The following should all be taken into consideration in our final design:

- The types of peripherals which it should be possible to connect.
- The limitations given by the IJVM ISA.
- The possibilities and limitations given by the Mic-1 architecture.
- The limitations given by the Gezel hardware description language.
- The efficiency of different solutions.
- Ease of understanding for unexperienced designers and undergraduate students.

2.4.1 Peripheral types

The I/O system proposed should supply a common interface for all the kind of peripherals that could be connected to the Mic-1. This means that it should be possible to connect both I/O devices and various memory devices as well as ASICs and coprocessors.

Some of the mentioned types of peripherals demand only one address, for example a keyboard which sends one character at a time.

The memory devices need an address referring to the target of the operation besides the address of the device on the chain. The communication

can be made more efficient if the address of the target is sent either at the same time as the data is sent or if it is coded into the address of the device, which is possible if the architecture allows one device to refer to more than one address i.e. a part of the address space of the bus.

These two different kinds of demands point in different directions since a memory mapped I/O system gives the ability to let one device refer to a large part of the address space. In conjunction to that some buses bind one address to each device in the chain. As an example the SCSI bus could be mentioned [Tan99, Section 2.3.5].

Timing issues concerning different kinds of peripherals

A way to divide the peripherals into groups is by their reply time. Three groups can be formed.

Fixed delay This group contains the devices that have a fixed delay length. This could be RAM memories or devices doing simple operations which take the same amount of cycles every time it operates.

Varying finite delay This group contains the devices which have a varying delay but where the result is always returned after a finite number of cycles. This could be all sorts of devices where the delay is data dependent. An example is a GCD-calculating ASIC where different combinations of input values lead to different computation times.

Possibly infinite delay The group contains devices which could have a possibly infinite delay on its reply. The reply of these devices will normally depend on interaction with or an event happening in the physical environment and if this fail to appear the device will never answer.

Since all three groups of devices could be connected to the Mic-1 possibly through a common interface, i.e. the I/O extension of the Mic-1, mechanisms to handle these different delays should be considered. Especially the presence of devices which could have an infinite delay is problematic since, it prevents solutions based on some kind of standard time out period.

A possible solution is to let the device handle the delay and let it send some status information after a certain period of time if no real answers have been returned. This status information should be interpreted by the program running on the ISA level.

2.4.2 A memory mapped I/O system in Mic-1

This section will discuss the possibility and the pros and cons regarding the extention of the Mic-1 architecture with a memory mapped I/O system.

A memory mapped I/O system demands the ability to directly manipulate values of memory addresses since a certain device has a certain address in the memory space.

The use of memory mapped I/O has two main obstacles:

- The IJVM architecture has no specific memory read and write instructions which refer to a specific address in the memory space (i.e the IJVM ISA only supply memory access through a limited amount of memory pointers).
- The Mic-1 architecture uses a simple memory model where it is expected that the answer to a request is ready in the next cycle in all situations. This assumption is reflected in the microprogram and no mechanism to handle delays in answers to memory request are available.

Several solutions could be proposed to overcome these hurdles, some of which will be presented here.

Constructing memory addresses in memory-mapped I/O

Since it is impossible to manipulate values in registers directly from the ISA level due to the nature of the IJVM's architecture, an extension of the instruction set is necessary to be able manipulate the registers to form specific memory addresses.

This leads to a proposal of an extension of the IJVM. At least two new instructions are necessary since it should be possible to choose between a read and a write operation.

Timing issues in memory-mapped I/O

The other major concern in the process of making the Mic-1 capable of communicating with its environment when the solution should be based on its present memory system, is the earlier mentioned timing issue.

The assumption about the memory used by the Mic-1 is that it is almost as fast as a register file and therefore answers to requests are always ready in the next cycle. Because of this assumption, no mechanism to handle delays in the communication has been implemented in the microprogram. The consequence of this is that an I/O system based on the present memory system should support the present timing assumptions, and therefore always reply in the next cycle.

This solution is very unrealistic since some of the connected devices could demand the ability to do several iterations of its calculations for example a GCD calculator. This is a clocked device which would normally be clocked by the global system clock which clock the CPU as well.

An even more problematic group of devices are devices with possibly infinite delays since these cannot be combined with the one cycle reply assumption unless the clock clocking the CPU is stopped every time a request is sent to a device.

The solution to all these problems is to add a mechanism which handles delays in replies. This includes:

- a mechanism in the CPU to stall its operation when a request is sent if it cannot continue operation before it has received a reply to the request.
- a protocol which allows the devices connected to the memory bus to signal to the CPU when it has an answer ready.

A standard solution for stalling a CPU is to put it in polling mode when it has sent a request. In polling mode it checks a status flag and when it changes, the CPU continues running the program. In the Mic-1 this mechanism should at least be implemented in the microprogram since the ISA level program cannot read flags or registers directly.

As mentioned in section 2.4.1 the polling mechanism implemented in the microprogram makes it possible to use polling on the ISA level too. This can be achieved if the device returns status information periodically to the CPU until the real answer is returned.

A solution to the protocol requirement could be a run/done protocol with dedicated run and done signals as a part of the bus protocol. An indirect run/done protocol could also be implemented. This could be done by sending specific values to registers or alter specific bits in a register.

Limitations introduced by the Gezel hardware description language

An additional problem connected to a memory mapped I/O system concerns the requirement to the implementation since it is required that it should be possible to implement the chosen solution in the Gezel hardware description language.

A major problem in the Gezel language is that it does not have a construct that works as a tri-state buffer. Therefore it does not allow more than one driver on a signal. Several receivers are allowed but not more than one driver. Therefore communication lines cannot be shared between several devices if they should be allowed to write to the communication lines.

This limitation hinder construction of a normal bus. A solution addressing this problem is explained in section 2.4.3.

Conclusion

It is not possible to implement a pure memory mapped I/O system based on a common bus connecting the devices to the CPU due the limitations

in the Gezel language. Despite this, several of the concepts in memory mapped I/O are suitable for an I/O system to the Mic-1 and can be implemented in the Gezel hardware description language. This includes mainly the ability to assign parts of the memory space to certain devices which gives room for connecting all sorts of memory devices to the Mic-1. The scalability of the memory mapped system also makes it attractive as foundation for the solution.

The timing issues in the present memory system in the Mic-1 also give rise to difficulties and therefore a solution should not be based on the memory port formed by the MAR and MDR registers.

It is also concluded that an extension of the IJVM ISA is necessary since it does not include any instructions handling I/O operations or operations reading and writing registers due to its stack based architecture.

2.4.3 Daisy chain based I/O in the Mic-1

In the discussion of extending the Mic-1 with a daisy chain based I/O system, the daisy chain is first discussed as a topology for the bus and later the use of the arbitration method is mentioned. This is different from how the notion daisy chain is used in [Tan99] and [PH98] where daisy chaining is used purely as an arbitration scheme.

In the discussion of designing an I/O system for the Mic-1 based on the daisy chain topology two major issues have to be addressed:

- The Mic-1 has no hardware supporting connections of any kind except the memory interface as discussed in section 2.4.2.
- A bus needs some kind of protocol. This should be developed and implemented in the microprogram. A modification is necessary in all circumstances, since the IJVM does not have any functions which supports any kind of I/O operation.

Extending the datapath of the Mic-1

The Mic-1 is a "closed" system in its basic form and has no I/O ports. Therefore an extension of the architecture with a daisy chain requires that the hardware, i.e. the data path, is modified.

This extension should be some kind of port composed of several sub signals. At least two sub signals are needed: a single bit read/write signal, and a wide address/data signal. In this configuration a write sequence would be composed by a cycle where an address is sent to the peripherals first and in the next cycle the data is sent out. This is called a multiplexed bus (see [Tan99, p. 160]) and is known from a lot of buses for example backplane buses like the PCI bus as mentioned in [PH98, figure 8.15]. A more efficient design would have more signals and therefore the ability to

send the address and data simultaneously. This also simplifies the protocol since only one clock cycle is needed to send both things.

In the data path the port should be represented by registers containing the values written to the port and in that sense work as a buffer for the port. On the physical level this will ensure stable values of the signals. This approach should also be used on input ports so data sent to the CPU are stored in a register connected to the port until the data is needed in the CPU.

In the Mic-1 there is only one general purpose register which is the `OPC` register. The rest of the registers has a certain function and can therefore not be used to compose the I/O port buffer. Therefore the data path should be extended by a group of registers, forming a buffer for the daisy chain. The number of registers needed depends on the protocol which is chosen for the communication on the bus. At least two are needed, corresponding to data/address and wr/rd signals mentioned above. These registers should of course be placed in the data path in the same way as the rest of the registers in it. This will also simplify the implementation, reduce the derived changes of the controller and ease the understanding of the system in general.

Requirement of the protocol

In general the protocol for a daisy chain should at least fulfill the same requirements as described for the protocol handling the memory mapped I/O bus described in section 2.4.2. In addition to these requirements the arbitration scheme described in section 2.2 could be implemented.

Conclusion

It is possible to extend the Mic-1 architecture with a bus based on the daisy chain topology. This solution requires an extension of the data path.

It also requires an extension of the IJVM instruction set together with a development of a protocol for communication on the daisy chain. The extension of the data path can be done by a small number of additional registers.

A daisy chain based topology will meet the demands of scalability and is easy to understand for undergraduate students.

2.4.4 Port based I/O in the Mic1

This section will discuss the pros and cons of an I/O system for the Mic-1 designed as a port based I/O system.

A major issue in a port based I/O system is the number ports and thereby the number of devices that could be connected.

Two main designs for a port based I/O system should be considered:

- A system based on the architecture of the present Mic-1 data path where the registers added to represent the ports are placed in parallel to the present registers in the data path.
- A system where an I/O controller is added to the system outside the data path.

Ports added directly to the data path

For each register in the data path, at least two control signals are needed: a load signal and a `b-bus` selection signal. For each added register a load signal is added to the micro instruction. The `b-bus` selection signal has room for more register without changing the width of signal. It is possible to add $16 - 9 = 7$ more registers before the b-bus selection signal has to be made wider. This gives room for 7 ports.

In general this solution does not provide a good scalability. The amount of ports cannot be changed but has to be decided in the design process. This leads to either unused hardware if not all ports are used, or lack of ports if all are used.

I/O controller added

This design is not mentioned in the general description but is closely related to the port based system. In this design a controller is added outside the data path. The controller works as an advanced multiplexor and demultiplexor where data from the data path is routed to a certain port on the controller. This solution was not considered further because it would be a major change in the Mic-1 and would also represent a solution with poor scalability.

Conclusion

A port based I/O system could be implemented in the Mic-1. In general this solution would be quite simple concerning the hardware extensions, but it has some drawbacks where its scalability is the major concern.

In general it must be concluded that a port based solution does not fit the requirements to an I/O system for the Mic-1.

2.5 The new I/O instructions

As menioned earlier one of our primary goals in this project is to achieve an easily understandable solution to the given problem. This should also

be taken into consideration when creating the new I/O instructions.

The intructions could be formed in different ways to achieve the best compromise between the wanted flexibility and the ease of use.

Three formats for the instructions are proposed. The “zero address”, “one address” and “two address” instructions.

Details about these solutions are relatively implementation specific and are therefore discussed no further in this section. Refer to section 3.1 for further details on the subject.

CHAPTER 3

The IJVM instruction set and the Mic-1IO architechture

After having discussed possible solutions to our problem regarding design and implementation of a flexible I/O environment to the IJVM instruction set and Mic-1 architecture described by Tannenbaum [Tan99], it is now time to decide which approaches we should implement and which approaches will be left out of the final design of the IJVM Instruction set and the Mic-1IO architecture.

Until now, we have assumed that no previous I/O instructions were available. However there are two views on this.

If we look at the IJVM instruction set defined in Tannenbaum's book [Tan99], no sort of I/O instructions have been included. If we on the other hand look at the Mic-1 implementation by Ray Ontko and Dan Stone [OS], two simple I/O instructions have been added to the IJVM instruction set and implemented in the microprogram and the Mic-1 simulator.

The discussions in this chapter will be based on the original IJVM instruction set by Tannenbaum unless otherwise specified. This mean that no I/O instructions are available in the IJVM instruction set discussed in this chapter.

Taking an overview of the IJVM instruction set and Mic-1 architecture implemented by Ontko and Stone (i.e. the IJVM and Mic-1 simulator featuring simple I/O capabilities), it is seen that the `IN` and `OUT` instructions are zero address instructions basing their actions on the values written to the stack instead of specified arguments to the instructions. When the `IN` and `OUT` instructions are interpreted by the microprogram, memory mapped I/O is used, addressing specific memory addresses for input and output. This implementation gives the possibility to perform simple I/O, but provide limited opportunity for adding new and more sophisticated de-

vices to the Mic-1 architecture without having to do complex rewriting of the memory system.

3.1 Modifications to the IJVM instruction set

The IJVM Instruction Set Architecture (ISA) defined by Tannenbaum, consists of a collection of integer based instructions from the Java Virtual Machine (JVM) ISA [Tan99, Section 4.2.3]. As stated above there are no input or output instructions available in the IJVM ISA.

This section of the report will cover this problem and propose a new IJVM instruction set containing I/O instructions to solve the problem of supplying a flexible I/O environment to the IJVM programmer.

The subjects discussed in this section have been briefly mentioned in section 2.5 but will be taken into more thorough consideration in this section of the report.

3.1.1 Needed I/O instructions

The primary motivation to implement the I/O instructions in the IJVM instruction set and Mic-1 architecture is to supply a simple yet flexible environment to perform simple I/O operations.

One way to do this is by supplying one input and one output instruction that takes care of all I/O operations.

In the following discussion of the IJVM instruction set, we chose to add the instructions IN and OUT. The discussion will therefore focus on how to make this approach flexible and easy to use.

3.1.2 Construction of the I/O instructions

Considering the construction of the added I/O instructions, two things should be taken into consideration. First of all the instructions should provide a high level of flexibility regarding the I/O devices tied to the relevant micro architecture.

Second it should be as simple as possible to perform basic I/O without preventing the use of complex I/O operations (i.e. drivers running on the ISA level or above performing complex communication with a device connected to the relevant architecture).

In the following, three strategies of implementing the I/O instructions are introduced and discussed. The discussion should result in the decision made regarding how the I/O instructions should be implemented in the final IJVM instruction set.

The discussion on zero, one and two address instructions are based on the `OUT` instruction. However the points discussed apply to the `IN` instruction as well with the exception of the two address version which only needs to specify one input device and therefore only needs one argument when looking at the `IN` instruction.

Zero address instructions

A zero address instruction consists of an instruction without any arguments. The instruction bases its actions on data already present in the system (i.e data already present in memory, registers or similar).

If we take the `OUT` instruction as an example, a zero address version of the instruction would appear as shown in figure 3.1.

```
...
OUT
...
```

Figure 3.1: *A zero address version of the `OUT` instruction*

As it is seen, no arguments are given. The instruction is written into the code basing its actions on the preceding code.

This version of the `OUT` instruction gives the IJVM programmer the power to write a program that modifies all the data given to the `OUT` instruction before it is executed. If we assume that the `OUT` instruction uses two data values, a device value containing which device is being addressed and a data value containing the data being transferred to the device, it is therefore possible to do calculations on both device and data values in the zero address schema.

However this version of the instruction forces the IJVM programmer to keep track on how the data is being loaded into the program to ensure correct function of the `OUT` instruction.

One address instructions

The one address version of the `OUT` instruction takes one argument. An example is shown in figure 3.2.

```
...
OUT <argument>
...
```

Figure 3.2: *A one address version of the `OUT` instruction*

Under the assumption that the argument given to the instruction is the device value (i.e the value identifying which device is being addressed),

the use of the OUT instruction is simplified for the IJVM programmer. The programmer no longer needs to concentrate on remembering where and how the device address should be stored when using the instruction. The device value is now just given as an argument to the instruction.

However in this schema, the programmer is prevented from doing calculations on which device should be addressed. The need for this depend on the which devices are connected to the micro architecture executing the code and how the devices have been attached to the micro architecture. In some cases the ability to do these calculations on both data and device values would be preferable (i.e. devices reserving a large chunk of I/O addresses, for example some kind of storage media).

Two address instructions

The two address version of the OUT instruction is similar to the one address version but takes two arguments instead of one. An example of the two address version of the OUT instruction is shown in figure 3.3.

```
...
OUT <argument 1> <argument 2>
...
```

Figure 3.3: *A two address version of the OUT instruction*

The two address version of the OUT instruction takes both device and data values as arguments. This has one obvious advantage. It is very easy for the programmer to control which arguments are given to the instruction.

However the two address approach completely eliminates the possibility of doing any calculation on the supplied data. The weight of this disadvantage is very dependent on the need and wishes for flexibility.

3.1.3 Flexibility versus ease of use

As it is seen from the previous discussion, ease of use regarding the number of arguments limits the flexibility of the new IJVM instruction set. Since the goal of this discussion is to chose a simple yet flexible solution to this problem, we need to decide on an appropriate solution.

The fact that the two address solution is very easy to use and that it creates compact code makes it interesting. However the fact that it prevents the possibility of doing output of results from calculations and only permits static I/O operations, makes it unacceptable for our solution due to the flexibility requirements.

Looking at the zero address and one address instruction schemas both of them need to supply data to the system before calling the OUT instruction

this could for example be done using the `BIPUSH` instruction. The difference of complexity when having to do two `BIPUSH` instructions instead of one is limited. We therefore chose to use the zero address schema resulting in the simplest use of the `OUT` instruction being similar to the code shown in figure 3.4.

```
...
BIPUSH <argument 1>
BIPUSH <argument 2>
OUT
...
```

Figure 3.4: *Simple output operation using the zero address schema*

As it is seen from figure 3.4 it is still relatively easy to do simple I/O, while maintaining full flexibility in the IJVM instruction set.

3.1.4 The new IJVM instruction set

After the discussion above, we chose to use the zero address version of the I/O instructions. It maintains a full level of flexibility without being complex to use. If simple I/O is needed only three instructions are needed as shown above. This is simple enough to form an acceptable solution regarding the ease of use.

After having decided how to represent the new IJVM instructions, we decided to add one extra level of flexibility. The feature we wish to add is the possibility of doing debugging on the I/O devices. It is therefore decided that both I/O instructions write data or status information back to stack.

The debugging possibility adds one instruction to the simplest use of the `OUT` instruction. This means that the simplest use of the instruction will be similar to the use shown in figure 3.5.

```
...
BIPUSH <argument 1>
BIPUSH <argument 2>
OUT
POP
...
```

Figure 3.5: *Simple output operation using the zero address schema after adding debugging information*

Finally it is necessary to note that the need for calculations on the device value heavily depends on the implementation of the I/O bus and the implementation of the I/O devices.

The possibility of doing calculations would for example be an advantage or even a demand if one device owns several addresses on the I/O bus and addressing of the device blocks are done through these addresses.

To sum up this discussion the IJVM instruction set defined by Tannenbaum [Tan99, Figure 4-11] is extended with the IJVM instructions shown in figure 3.6.

Hex	Mnemonic	Meaning
0xFD	OUT	Pop two words from stack to use as data and device address. Write status back to stack.
0xFC	IN	Pop one word from stack to use as device address. Write data or status back to stack.

Figure 3.6: *The new IJVM instructions*

3.2 Overview of the Mic-1IO design

After the analysis performed in chapter 2, a hybrid solution composed primarily from principles found in memory-mapped and in daisy chain I/O systems was developed.

The main characteristics of the solution can be summarised as follows:

- The data path of the Mic-1 architecture are extended by one port including two buffer registers called `IOA` and `IOD`.
- The topology of the bus connected to the CPU is a daisy chain looking topology.
- The address scheme for the devices on the bus is continuous, featuring a 32-bit wide address space.
- The bus is terminated by an active terminator which signals address overflow on the bus.
- A protocol implementing blocking I/O has been developed to control the communications taking place on the bus.

The above mentioned characteristics will be described in further detail in the following sections.

3.3 Modifications to the hardware of the Mic-1

Since one of our main goals in making this extension to the Mic-1 is to keep it as simple and easily understandable as possible, we wish to make the changes to the architecture as isolated as possible (i.e. we wish to keep as much of the original architecture intact and limit our modifications to a small attachment to the architecture without affecting the way the original architecture works). This goal should result in an easily understandable Mic-1IO architecture for a person already familiar with the Mic-1 since the architectures are almost identical with the exception of a few minor additions to the architecture.

Taking this into consideration, the way to implement I/O with the smallest adjustments to the architecture would be using memory mapped I/O described in section 2.1. However the fact that JVM and IJVM does not support direct access to memory addresses complicate the use of memory mapped I/O. Memory access is only available through the model described by Tannenbaum [Tan99, Section 4.2.2]. Implementing memory mapped I/O using the available memory access registers would require the development of a relatively complicated I/O protocol to provide the level of flexibility that we wish from our new I/O system. These facts make it undesirable for us to use this schema since it would require a relatively abstract understanding of the IJVM ISA and the Mic-1 architecture complicating things for inexperienced users and developers (i.e. students doing the course related assignments in the Introduction to Codesign course at DTU).

It could be considered to redefine the IJVM ISA to support memory access in a new area with the use of a new register added to the architecture. Memory mapped I/O through this part of the memory could then be made possible. However this would complicate the understanding of the IJVM ISA and is therefore undesirable.

A separate I/O bus is now considered. The fact that we want a certain degree of scalability is met by using this approach since new devices can be added to the I/O bus without having to modify the CPU itself.

As earlier mentioned, a way of implementing the I/O bus approach would be through the daisy chain approach described in section 2.2. However the daisy chain approach does fit our wishes completely since it is built on top of an interrupt system that enables the devices to send an interrupt to the CPU when they are ready resulting in grant token being sent through the daisy chain and caught by the highest prioritised device ready for operation.

In our version of the I/O bus, we wish the devices to be addressed from the CPU. Furthermore we wish to keep the design scalable and minimise

the modifications of the original Mic-1 architecture . Looking at the discussion above, we find that this could be achieved combining features from the memory mapped I/O schema and the daisy chain schema.

The actual modification to the Mic-1 architecture will be discussed in a moment after this description of the bus topology.

To be able to implement the I/O bus without extending the CPU with a port for each I/O device present while still keeping within the limitations of the Gezel hardware description language, we use the topology of the daisy chain schema, making all I/O data and addresses propagate through all the devices on the I/O bus until it reaches its destination device. However we make all devices on the chain able to catch data request to a certain address or set of addresses similar to the way it is done in the memory mapped I/O schema.

The remaining part of this chapter will primarily focus on the actual modifications to the Mic-1 architecture and the communications protocol. Further detailed description of the bus topology will follow in section 5.2.1.

Based on the above discussion, our first strategy for extending the Mic-1 architecture was based on extending the architecture with three new registers. A data register, an address register and a status register. The data register should contain the data transmitted to and from the I/O bus. The address register should contain the address of the device being addressed. The status register should be used by the devices on the I/O bus to signal the CPU, telling it that the device had finished its operation and execution of the program could continue.

After considering this solution it is seen that the status register is unnecessary and that the needed status information can be given using only the data and address registers. Due to our goal of making the least amount of modifications to the Mic-1 architecture , we chose to use the two register version instead of the one containing the status register. The two register version communications protocol will be a little more complex than the three register version. However it should still be relatively easy to understand. A protocol taking care of the communication in the two register version of the I/O strategy is described in section 3.4.

After having chosen the two register approach to the I/O communication interface, we need to determine how it should be connected to the Mic-1 architecture .

It should be possible to write to the I/O registers from both the `c-bus` and from the I/O bus. Furthermore the I/O registers should be able to output to both the `b-bus` and the I/O chain. The new I/O registers implementing the required features are shown in figure 3.7.

Besides the added registers the necessary control lines are of course added and the micro instruction register is extended accordingly. This extension is described in section 4.3.2.

In addition to the control signals added to the architecture, a signal used

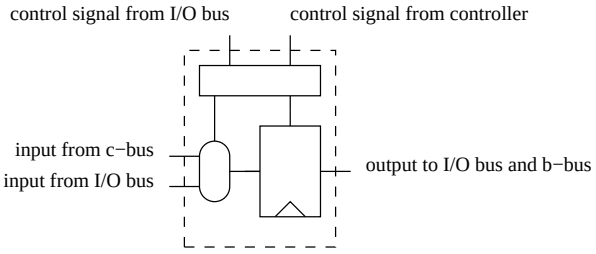

Figure 3.7: *I/O register taking input from both* `c-bus` *and I/O bus*

to tell the I/O devices whether to function as an input or an output device is added to the system. This signal will be described in further detail in section 4.1.

A figure showing the new Mic-1IO architecture is shown in figure 3.8.

The I/O interface composed by the I/O registers supply a 32 bit address space. This address space is very large and would be reduced substantially if the architecture should be implemented in practice. For example five bits would support 32 device addresses which in many cases would be sufficient.

Furthermore one would probably construct the I/O interface from one register in practice since multiplexing between address and data would make I/O possible and save a number of bits on the I/O bus making it more cost efficient. However this approach would make the protocol needed more complex thus conflicting with our goal of simplicity.

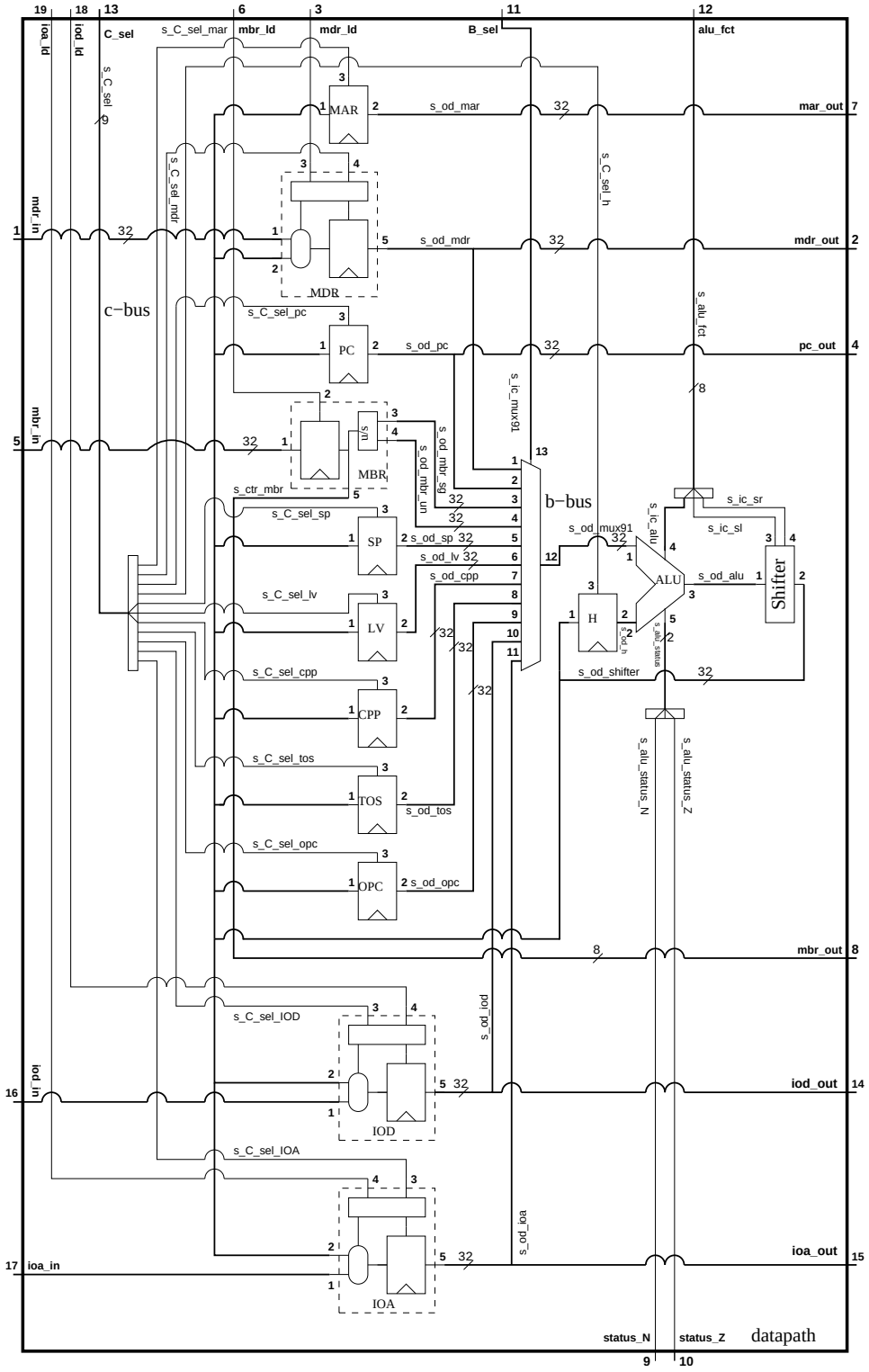

Figure 3.8: *The Mic-1IO datapath*

3.4 I/O communications protocol

After having chosen the version of the Mic-1IO having two registers and no status register, a protocol to control the I/O data flow is needed. This section of the report will focus on describing this protocol.

The I/O protocol states a set of rules for the communication on the I/O bus. The main purpose is to secure a simple and efficient handling of I/O calls, read and write operations, to and from the devices on the bus. The protocol should also support mechanisms which gives the programmer ability to detect errors in the communication.

Figure 3.9 shows a finite state machine describing a protocol satisfying the needs specified above.

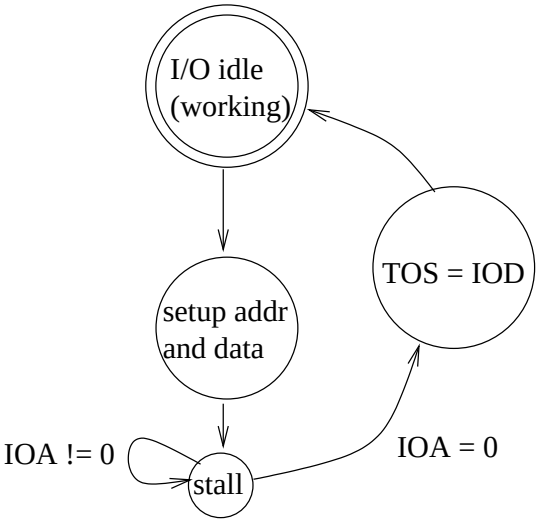

Figure 3.9: *Finite state machine showing the behavior of the CPU when handling I/O instructions*

Figure 3.10 shows a timing diagram of the values present on the ports of the Mic-1IO architecture.

The following two subsections interpret this protocol, describing an actual `IN` and `OUT` instruction and its effects on the CPU and I/O bus.

3.4.1 A read operation

This is a description of a successful read operation:

1. The CPU sets up the `iord` signal and the address line by writing the address to the `IOA` register.

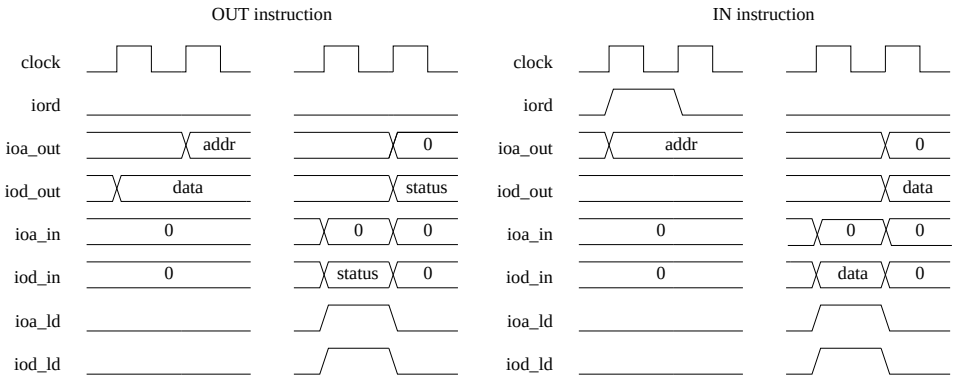

Figure 3.10: *Timing daigram for the* `IN` *and* `OUT` *instructions*

2. In the following cycle the address is sent out on the bus and is forwarded by the devices until the device which "owns" the address receives it. At the same time the device detects the value of the `iord` signal to be able to determine if the requested operation is a read or a write operation.

3. In the same cycle the CPU goes into polling mode where it does not do anything until the requested data is received.

4. When the device has the data ready, it writes the data out on data line (including raising the `IOD` load signal) and write zeros on the address line (including raising the `IOA` load signal) to signal that the data is ready.

5. When the CPU detects the zero in `IOA` register, it writes the value in data register to the stack and continues the execution of the program.

3.4.2 A write operation

This is a description of a successful write operation:

1. The CPU load the data into the `IOD` register.

2. The CPU load the device address into the `IOA` register

3. In the following cycle the address and data is send out on the bus and is forwarded by the devices until the device which "owns" the address receives it. At the same time the value of `iord` is read by the device to determine if the requested operation is a read or a write operation.

4. In the same cycle the CPU goes into polling mode where it does not do anything until the `IOA` is changed to zeros.

5. When the device has written the data, it writes zeros back in the `IOD` and in the `IOA` . Alternatively the device might write some other status value to the `IOD` register.

6. When the CPU detects the zeros in the `IOA` register, it writes the value in data register to the stack and continues execution of the program.

3.4.3 Error detection and handeling

The error detection is done by having the terminator return the value 0xfeedbeef to the CPU if no peripherals catch the operation before it reaches the terminator. This makes the programmer or CPU able to detect the error since it, depending on the way it is treated when returned to the CPU, can be used to identify the error. Figure 3.11 shows an example of handling an address overflow.

```
BIPUSH 1
BIPUSH 42
OUT
IFEQ succes
... code for error handling continues here
```

Figure 3.11: *Example code showing simple address overflow detection during an output instruction. If zero are written back to stack it is interpreted as a succesful operation and program jumps to the label called "succes"*

In the general case this error handling schema will be sufficient to detect most errors. However, some errors will not be detected when using this type of error handling on this CPU.

Since the CPU enters polling mode, polling the `IOA` register every cycle to see if the request has been processed,it will go into an infinite loop if the peripheral catches the request but never returns a value.

However one would probably seek to implement some sort of device drivers in the IJVM programs polling for status information from the devices instead of waiting infinitely for the device. This would allow program execution to continue on parts of the program that does not rely on the I/O device being polled periodically.

If the device is implemented to send back status information to the CPU only one error could cause the CPU to stall completely. This error would be a complete breakdown of one of the I/O devices after having caught the I/O request, but before being able to send status information to the CPU. This error would cause the CPU to enter an infinite loop that would never terminate.

However this error could be avoided by implementing some sort of timeout on the I/O system. For reasons of simplicity this has been left

out of this protocol since it would complicate communication with slow devices, without implementing a sophisticated device driver in the IJVM program for slow devices like a keyboard and similar.

The implementation of device drivers would be a nontrivial task but the flexibility of the new IJVM ISA does not leave out the possibility.

CHAPTER 4

Microprogram, micro-assembler and IJVM assember

To be able to make use of the modified IJVM instruction set and the Mic-1IO architecture it is necessary to develop a microprogram capable of interpreting the IJVM instruction set and capable of controlling the Mic-1IO architecture in the intended way when running IJVM programs on the architecture.

As mentioned in chapter 3, a simulator for the Mic-1 architecture has already been implemented by Ontko and Stone [OS]. This simulator includes a microprogram, a micro-assembler, an IJVM assembler and of course the Mic-1 simulator.

Given the nature of the modifications to the Mic-1 architecture where we have been trying to limit the modifications to the original Mic-1, the programs supplied by Ontko and Stone need limited modifications to implement the features of the new Mic-1IO architecture.

The modifications of the microprogram, micro assembler and IJVM assembler for the IJVM instruction set and the Mic-1IO described in chapter 3 will be described in the following sections.

4.1 The modified Micro Assembly Language (MAL)

To be able to support the extensions to the Mic-1 architecture, the microprogram has to be modified. However the MAL in its original state is unable to describe the proposed modifications to the Mic-1 architecture, since new reserved keywords to describe the newly added registers and signals need to be added to the MAL.

To make the MAL able to describe the actions of the new I/O system, the reserved keywords `IOD` , `IOA` and `iord` have been added to the MAL.

The keywords `IOD` and `IOA` are added to make it possible to read from and write to the two new registers, `IOA` and `IOD` . The usage of these keywords is identical to the ones used to address the other registers of the Mic-1.

The keyword `iord` is used to set a signal telling the peripheral device being addressed that the I/O request given should be interpreted as an input instruction. For example a peripheral storage device such as a flash disk, able to do both read and write operations, could use this signal to interpret whether the requested operation is a read or a write operation since the signal will be set during read operations.

4.2 Modification of the microprogram

Implementing the new I/O instructions leaves us with some choices on how to implement the new I/O instructions in the MAL.

The new microcode should implement the I/O communications protocol described in section 3.4.

The implementation of the new microinstructions can be seen in appendix A.1 line 228-239. The commented code should supply a detailed description of the modifications made compared to the I/O protocol described.

Both IN and OUT instructions result in data being written to the stack. As suggested in section 3.4.3 this value can be used to detect errors during the operation of the I/O bus.

The `iord` signal are unbuffered when accessing the I/O bus. This means that it needs to be set at the same time as the I/O address is present on the output of `IOA` . This is why the `iord` signal has its own cycle when looking at the microcode for the `IN` instruction.

4.3 Modification of the micro-assembler

Having modified the MAL and the microprogram to implement the new I/O instructions; modifications of the micro-assembler are now needed to make use of the new microprogram.

4.3.1 Grammar and parser generator

First of all the grammar of the MAL must be modified to implement the new reserved keywords introduced in section 4.1. Figure 4.1 shows the

modified nonterminals of the grammar and their corresponding terminals. To view the entire original grammar without modifications, refer to the MAL definition described on Ray Onko's website [OS, Micro-Assembly Language (MAL) Specification].

io_statement	$\mapsto$	*rd*	
io_statement	$\mapsto$	*wr*	
io_statement	$\mapsto$	*fetch*	
io_statement	$\mapsto$	*iord*	$\leftarrow$ New terminal
c_register	$\mapsto$	*MAR*	
c_register	$\mapsto$	*MDR*	
c_register	$\mapsto$	*PC*	
c_register	$\mapsto$	*SP*	
c_register	$\mapsto$	*LV*	
c_register	$\mapsto$	*CPP*	
c_register	$\mapsto$	*TOS*	
c_register	$\mapsto$	*OPC*	
c_register	$\mapsto$	*H*	
c_register	$\mapsto$	*IOD*	$\leftarrow$ New terminal
c_register	$\mapsto$	*IOA*	$\leftarrow$ New terminal
b_term	$\mapsto$	*MDR*	
b_term	$\mapsto$	*PC*	
b_term	$\mapsto$	*MBR*	
b_term	$\mapsto$	*MBRU*	
b_term	$\mapsto$	*SP*	
b_term	$\mapsto$	*LV*	
b_term	$\mapsto$	*CPP*	
b_term	$\mapsto$	*TOS*	
b_term	$\mapsto$	*OPC*	
b_term	$\mapsto$	*IOD*	$\leftarrow$ New terminal
b_term	$\mapsto$	*IOA*	$\leftarrow$ New terminal

Figure 4.1: *The modified nonterminals of the MAL grammar.*

The actual implementation of the grammar is done using the CUP Parser Generator for the generation of a parser for the actual assembler. Refer to the `Mic1Parser.cup` source code in appendix A.2 to view the actual modifications to the CUP definition file originally supplied with the Mic-1 implementation by Ray Ontko. In particular, refer to these lines: 39, 55-56, 143-145, 208-212 and 301-305. A description of the modifications is supplied below.

`IOA` , `IOD` and `iord` are defined as terminals in the grammar with the purpose of making the keywords known to the parser. Furthermore `IOD` and `IOA` are added as `c_register` and `b_term` to make it accessible from the B-bus and writable from the C-bus. `iord` are added as an

`io_statement` making it work similar to the I/O instructions for communicating with the RAM module. For an overview of the B- and C-bus layout of the original Mic-1 architecture refer to [Tan99, Figure 4-1].

4.3.2 The micro-assembler

Since our modification of the Mic-1 architecture results in some extra control signals, the micro-assembler must be modified to meet these requirements. Therefore the original microinstruction format defined in [Tan99, Figure 4-5] must be modified to implement the new signals and afterwards the micro-assembler must be modified to produce this output when run on a microprogram written in the MAL.

The microinstruction format

The new microinstruction format defined to implement the new signals is shown in figure 4.2.

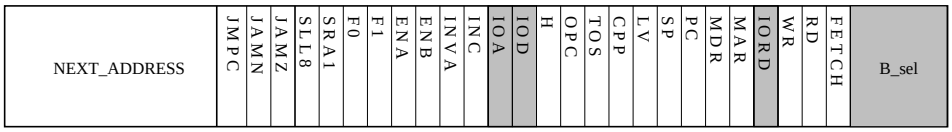

Figure 4.2: *The microinstruction format for the Mic-1IO architecture. Added and redefined fields in the instruction format are greyed out.*

As it is seen, four fields have been added or redefined.

The signals `IOA` and `IOD` have been added to make the micro-controller able to write the value on the `c-bus` into the newly added I/O registers in the same way as the signals for the other registers are present.

The signal `iord` has been added to function as the signal telling a peripheral device that the operation given should be interpreted as an input instruction. As mentioned earlier this signal will be set during the appropriate clock cycles of input instructions.

As shown in figure 4.2 the signals contained by `B_sel` have been redefined as well. This is done to make the micro-controller able to read the values of the I/O registers and put them onto the `b-bus`. The number of bits in this signal remains the same due to the fact that the 4 bit signal can be decoded to a maximum of 16 different signals and only 11 are needed. This means that no actual modification of the signals are made; Only two new interpretations of unused 4 bit values are defined.

The micro-assembler

The micro-assembler supplied by Ray Ontko and Dan Stone [OS] consists of numerous files. Some parts have been generated using the CUP parser generator described in section 4.3.1, i.e. the files related to the parsing of the microprogram. The rest of the files have been written to interpret and assemble the output of the generated parser. These files need to be modified to support the new output from the parser after having modified the grammar.

After closer examination of the files mentioned, it turns out that only two files need modification. These files are `Mic1Instruction.java` see appendix A.3 and `Mic1Scanner.java` see appendix A.4.

The modifications to `Mic1Instruction.java` can be summarised to the following lines of appendix A.3: 64-65, 80-81, 90, 117-146, 159-189, 224-227, 241-242 and 452-454.

The modifications made to `Mic1Scanner.java` can be summarised to the following lines of appendix A.4: Lines 58 and 75-76.

The comments in the modified code should provide sufficient information about the nature of the modifications.

4.4 The IJVM assembler

Since we have chosen to base our microprogram, micro-assembler and IJVM assembler on the programs supplied by Ontko and Stone, the implementation of the IJVM assembler is straight forward.

The IJVM assembler supplied by Ontko and Stone does not need any modification to function correctly on the new IJVM instruction set and the Mic-1IO architecture. This is due to the fact that we have chosen to implement the I/O of the new IJVM instruction set through the zero-address instructions `IN` and `OUT`. These instructions are already present in the IJVM assembler supplied by Ontko and Stone. The only difference is the interpretation made by the microprogram.

CHAPTER 5

Implementing the Mic-1IO using Gezel

In this chapter major issues concerning the implementation of the Mic-1IO in the Gezel hardware description language will be discussed.

As a part of the project, the Mic-1 architecture should be implemented from scratch. This led to some design decisions which will discussed in the first section.

Afterwards issues concerning the extensions leading from the Mic-1 architecture to the Mic-1IO architecture are discussed. This includes both extensions to the data path, controller and the bus itself.

Of course the discussions in the section dealing with the basic Mic-1 architecture also applies to the CPU part of Mic-1IO architecture.

5.1 The Mic-1 architecture

As mentioned above a reference design used to do comparisons between the Mic-1 and Mic-1IO was implemented. Appendix D.1 shows the Gezel sourcecode for the Mic-1 reference. This design also worked as basis for the extensions done to design the Mic-1IO.

5.1.1 The B-bus

In [Tan99, figure 4-6 on page 214], Tanenbaum suggests a design where the B-bus in the CPU is a real bus shared between the registers. His design implies bus driving signals in the registers controlling whether the individual registers drives the bus.

Unfortunately this design cannot be implemented in the Gezel hardware description language since it does not allow more than one driver on each signal.

Our solution to the problem is to use a multiplexor instead of the shared bus. In this solution the registers' outputs are connected to individual inputs on the multiplexor. The decoder, decoding the B-bus selector signal, is removed from the design and the B-bus selector signal is connected directly to the multiplexor.

This solution also leads to a simpler register design which works more like edge-triggered registers used in hardware implementations on the circuit level. A short description can be found in [RCN03, section 7.1].

The structure of the data path for the reference system can be seen in figure 5.1

5.1.2 The registers

The memory model and microprogram of the Mic-1 leads to a demand for some registers with a special property. These registers are `MAR`, `MDR`, and `PC`. The property could be described as transparency which means that the value loaded into the register is reflected on the output port in the same cycle as it is loaded into the register.

The implementation is showed in figure 5.2.

5.1.3 Hierarchical use of ip-blocks

In the version of Gezel available when we started working on this project it was not possible to use embedded ip-blocks. This has fortunately been changed in newer versions, which has allowed a more straightforward structure of the system, avoiding the connection of all ip-blocks at the top level of the fdl-file.

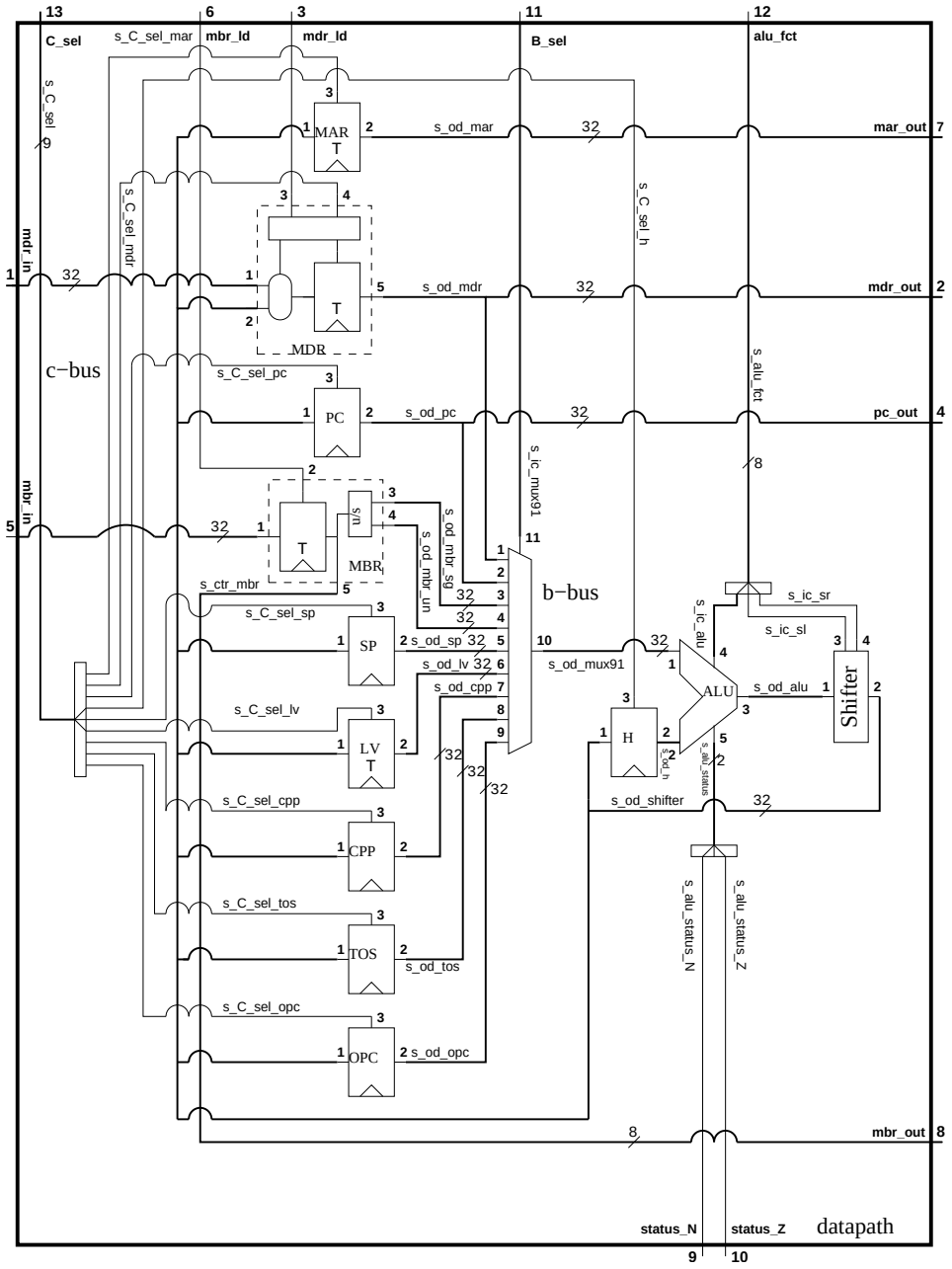

Figure 5.1: *The data path of the Mic-1 system.*

```
dp marR (
    in id_x : tc(32);
    out od_q : tc(32);
    in ic_ld : ns(1)
    ) {
        reg rx : tc(32);

        sfg exec {
             rx = ic_ld ? id_x : rx;
             od_q = ic_ld ? id_x : rx;
        }
    }
hardwired ctrl(marR) {exec;}
```

Figure 5.2: *Implementation of transparency in registers. The figure shows the* `MAR` *but the same technique is used for the other registers.*

5.2 The Mic-1IO architecture

The Mic-1IO architecture includes extensions to the data path, controller and addition of a bus connected to the CPU.

The extension of the CPU is the addition of two registers. The registers are both able to choose input from two different sources.

5.2.1 The I/O bus

The bus is connected to the CPU through a port composed of seven signals: Data in, data out, data register load, address in, address out, address register load and read/write.

The implementation of the bus has been done by composing the daisy chain topology bus from standard bus blocks called `daisy_chain_block`. These blocks are connected to each other and the last is connected to the terminator. The topology of the system is shown in figure 5.3.

The purpose of the `daisy_chain_blocks` is to send the communicated data and address to the connected device if the address is in the subspace assigned to that device or otherwise forward the communication to the next `daisy_chain_block`.

The upper and lower limit of the subspace are set by assignments to two registers at the time of initialisation. The code is shown in figure 5.4. If the device has only one address assigned, the upper and lower limits should of course have the same value.

The full `daisy_chain_block` implementation can be investigated between line 2 and 106 in appendix D.2.

The `daisy_chain_blocks` are implemented as combinatorial logic. and the data communicated through blocks are not buffered in the blocks. This has two implications:

First this could lead to problems concerning the simulation in the Gezel environment but this is not certain and in any circumstances it does not lead to any problems when simulating in the Gezel environment by Patrick Schaumount.

Secondly the present design could lead to a long critical path in case of many peripherals being attached to the I/O bus. This is due to the fact that the signal has to propagate through all the `daisy_chain_blocks` in the I/O bus to reach the terminator. This could be an argument for doing some buffering in `daisy_chain_blocks`.

Unfortunately the developed protocol would need to be modified if buffers were implemented in the `daisy_chain_blocks` due to some timing conflicts that would arise because of the buffers.

This subject will not be discussed further, but more investigation should be carried out if buffers were implemented into the `daisy_chain_blocks`.

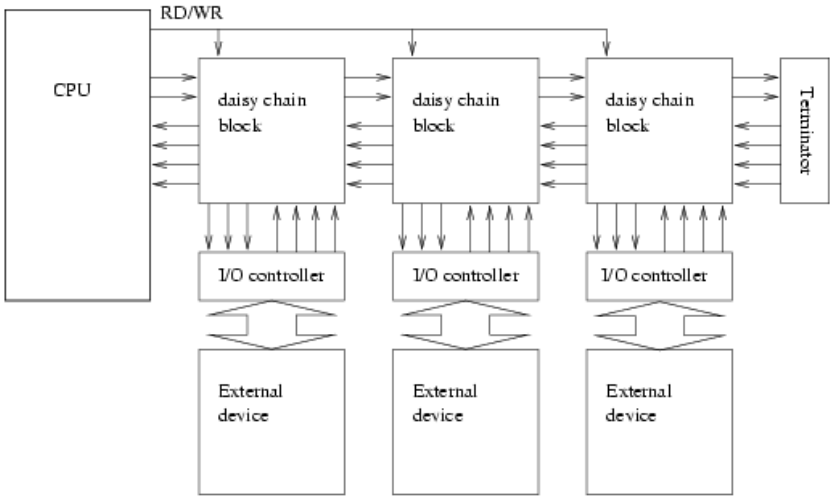

Figure 5.3: *The topology of the I/O bus. The purpose of the I/O controllers is handle the communication between device and the bus so that the device does not need to have the protocol of the I/O bus implemented.*

```
    reg lower_bound, upper_bound : tc(32);

    sfg init {
        device_iord = iord;
        adress_to_mic2 = 0;
        data_to_mic2 = 0;
        iod_ld_to_mic2 = 0;
        ioa_ld_to_mic2 = 0;
        adress_to_terminator = 0;
        data_to_terminator = 0;
        adress_to_device = 0;
        data_to_device = 0;

        //sets upper and lower
        //address bound for this daisy chain block
        lower_bound = 1;
        upper_bound = 30;
}
```

Figure 5.4: *code assigning upper and lower limit to the registers in a* `daisy_chain_block`*. It is the two last lines doing the job.*

Another issue in implementing the `daisy_chain_block` was the fact that every `daisy_chain_block` needs to be written individually in the fdl-file. This is due to a limitation in the Gezel language's `use` construct which does not have the possibility to give arguments to the data path reused through `use` construct. If that was possible the address limits for the individual `daisy_chain_blocks` could be given as arguments.

The I/O controllers

The purpose of the I/O controllers in figure 5.3 is to handle the communication between the device and the I/O bus. This controller should be designed for every single device which should be connected to the bus.

An example of the use of this concept is given together with the description of the GCD-calculator in section 6.2.

CHAPTER 6

Description of the example peripherals

The XiLinx FPGA lab boards are used in course 02130 to test the synthesis of the Mic-1 implemented by the students. In order to extend the capabilities of the system and exploit the peripherals connected on the DIO 5 expansion board, ip-blocks emulating some of the peripherals on the DIO 5 board have been implemented.

Besides that a simple GCD calculator and a simple output device has been implemented in Gezel. The last device is meant for debugging and test purposes.

6.1 Developed ip-blocks

Two ip-block have been implemented to emulate two I/O devices available on the DIO 5 extension board. These ip-blocks will be briefly described in this section of the report.

It is important to note that the ip-blocks are not considered a central part of this project but have been implemented as simple examples to be able to simulate some of the surroundings of the FPGA board. This means that no thorough testing of these devices will be performed.

For details about implementation of the ip-blocks, please refer to the commented source code in appendix B.

6.1.1 The NumPad ip-block

The NumPad ip-block is meant to emulate the numerical keyboard available on the DIO 5 board.

The NumPad ip-block is relatively simple. Its function is to be able to feed the requesting Gezel program with 4 digit decimal numbers. When called during simulation, the program prompts the user for a number between 0 and 9999. When this is given, it is passed on to the reguesting Gezel program.

It is important to note that the numpad ip-block is created as a simple example and that no fault tolerance has been implemented (i.e. giving it input other than an integer between 0 and 9999 will result in incorrect output).

The DIO 5 board does not provide any specific functionality of the onboard numpad. However the FPGA could be programmed to perform the described functionality of the numpad.

To include the numpad ip-block in a Gezel program, use the Gezel codeshown in figure 6.1.

```
ipblock numpad(
    in in_addr : ns(32);
    in in_data : ns(32);
    in read : ns(1);
    out out_addr : ns(32);
    out out_data : ns(32);
    out ioa_ld : ns(1);
    out iod_ld : ns(1)
    ) {
        iptype "numpad";
    }
```

Figure 6.1: *Gezel code used to include the numpad ip-block*

6.1.2 The lcd ip-block

The lcd ip-block is meant to emulate the LCD display available on the DIO 5 extension board.

Since the ip-block is meant to have the same functionality as the LCD display of the I/O board connected to the FPGA board, the lcd ip-block is relatively complex.

The usage and functionality of the ip-block is described in the following paragraphs, for a more detailed description of the functionality of the LCD display, check the detailed data sheet supplied by the FPGA board manufacturer or the description [Dig] of the Samsung LCD driver [Sam].

The ip-block is activated by changing the value of the `in_addr` port.

To include the lcd ip-block in a Gezel program, use the Gezel code shown in figure 6.2.

```
ipblock lcd(
    in in_addr : ns(32);
    in in_data : ns(32);
    in read : ns(1);
    out out_addr : ns(32);
    out out_data : ns(32);
    out ioa_ld : ns(1);
    out iod_ld : ns(1)
    ) {
    iptype "lcd";
    }
```

Figure 6.2: *Gezel code used to include the lcd ip-block*

The lcd ip-block will, when called, print a textual representation of the character memory and display on the the FPGA board. The representation is done using three text lines. One line showing the position of the visible area and two lines showing the data in memory. When initialised the representation will look as shown in figure 6.3.

```
"B               E                          "
"                                           "
"                                           "
```

Figure 6.3: *The lcd ip-block after initialisation*

The first line represents the physical displays position in the character memory. The "B" represents the beginning of the display and the “E”, the end of the display. If the visible area of the display is moved to far to either side it will wrap to the other end of the character memory.

The second and third line of the display represents the contents of the character memory. This means that the display representation shown in figure 6.4 would display “abc” in the beginning of the first line and “def” in the end of the second line. The text “ghi” will not be shown since it is in the character memory but not in the visible display area.

```
"B               E                          "
"abc                                        "
"              def          ghi             "
```

Figure 6.4: *The lcd ip-block containing some example data*

When the ip-block is activated, changing the address on the `addr_in`

port, the ip-block will print what instruction has been called, the display and character memory, after the instruction has been applied.

6.2 The GCD block

The GCD block is implemented to illustrate connection of an ASIC on the bus. The GCD calculator unit is composed from a simple data path with two registers, an ALU, a multiplexor and a demultiplexor and a controller performing Euclid's algorithm. This GCD calculator is used as an introductory example of hardware design in the co-design course. The controller of the GCD unit is extended with support for a run/done protocol.

The GCD calculator unit is connected through an I/O controller to a daisy chain block. This I/O controller implements the protocol described in section 3.4.

This design is chosen to illustrate that an already designed circuit can be connected if a dedicated I/O controller is designed for the purpose.

The buffer register and the I/O controller in figure 6.5 fill out function of the I/O controller of figure 5.3.

The Gezel interface to connect the GCD calculator to the I/O bus is shown in figure 6.6.

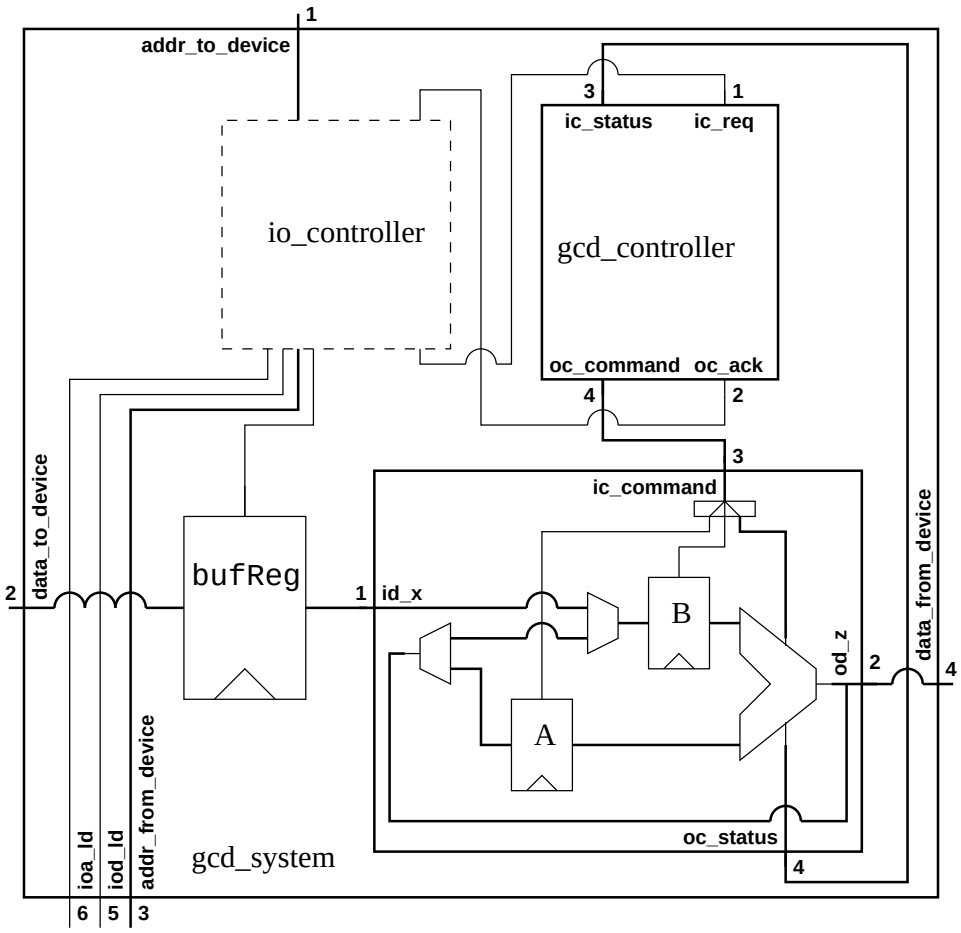

Figure 6.5: *The GCD system.*

```
dp gcd_system(
    in addr_to_device : tc(32);
    in data_to_device : tc(32);

    out addr_from_device : tc(32);
    out data_from_device : tc(32);

    out iod_ld : ns(1);
    out ioa_ld : ns(1);

    in iord : ns(1)
)
```

Figure 6.6: *The Gezel code to connect the gcd system to I/O bus. As it is seen it is a standard connection to a daisy chain block.*

6.3 A simple IO device

To test the I/O bus' basic functionality and the protocol, a simple device which can receive and return a data word is developed. It does not do any calculations on the received data except from writing out the data on the screen along with other debug information.

CHAPTER 7

The modification of the Mic-1 simulator

To make it easier to verify the correctness of an implementation of the Mic-1IO, the Mic-1 simulator developed by Ray Ontko has been modified to support the extensions made to the original Mic-1 architecture.

To be able to use the extensions, two peripherals have been developed as well. These two peripherals have been made to be identical to the ones developed as the ipblocks to be used with the gezel implementation.

7.1 The Original Mic-1 Simulator

The Mic-1 simulator has been implemented by Ray Ontko and Dan Stone to follow the design guidelines described by Tanenbaum [Tan99].

The simulator has been implemented using the JAVA programming language using the Abstract Window Toolkit (AWT) to supply a graphical simulation and debugging environment for the program run on the simulator.

The original simulator has been written to be a functional simulator for the Mic-1 architecture. However no intentions of making the simulator easy to extend and modify has been present during the development of the simulator.

When evaluating the source code of the original Mic-1 simulator, it is seen that no separation of the Graphical User Interface (GUI) and the actual functional parts of the program has been made. For example the registers are implemented, extending the AWT TextField, combining the graphical features of the TextField with the register capabilities of the simulator.

This fact combined with the lack of program documentation and source

code comments makes the understanding and modification of the simulator relatively complex.

7.2 The Mic-1IO Simulator

As described the original simulator was not implemented with the purpose of making it easy to modify it. This leaves us with two choices. Either we should implement the entire simulator again, making it more flexible and easier to extend in the future. Or we should make the modifications to the simulator as it is and keep the present layout and structure of the original simulator.

As discussed earlier in section 3, a main goal of the Mic-1IO architecture is, if possible, to keep the modifications made separated from the original Mic-1 parts of the architecture.

The I/O parts of the architecture will be connected to the original parts of the architecture, but due to the above mentioned fact, a limited amount of connections are needed. This means that the actual modifications to the original simulator will be relatively limited.

As a consequence of this we have chosen to modify the original simulator instead of starting over, implementing a new simulator from scratch. The principle of extending the graphical classes when implementing the functionality of the simulator has been followed in the extension of the simulator since this would keep the code homogeneous in its structure.

7.2.1 The I/O Registers

The two I/O registers, `IOA` and `IOD` , are implemented in the simulator through the developed `IO_reg` Java class. This class implements the IO-register functionality which is an extended register with two data input ports and the ability to choose between these two.

The functionality of the registers is a hybrid between the registers defined in Tanenbaum's book [Tan99] where a register only drives the bus if it is signalled, and a real register where a value is always present on the output port. This is done to meet the demands which apply in the implementation of the simulator and to make an implementation which comes as close to the implementation of the Mic-1IO in Gezel as possible.

Another issue in these registers is the decision whether to read the value from the c-bus or the I/O bus. The solution is adapted from the MDR register. The solution is to give an input for the I/O bus priority over the c-bus. This is done by first reading from the c-bus and then from I/O bus. In both cases it is only done if the read og load signal is raised. This solution is not robust in the sense that it is impossible to write from both busses in

the same cycle and lose the data from the c-bus. It must be expected though that this situation will never occur. If it did it would be an error in the micro code.

7.2.2 The I/O bus

Since the simulator developed by Ontko and Stone connects all components at the same level, we continue to do so. This makes the code harder to understand but starting to use another way of structuring the code would probably be even more confusing.

In the Gezel model of the system each peripheral is connect to the bus through a data path called `daisy_chain_block` which handles the communication on the bus together with the other `daisy_chain_blocks` and the terminator. This structure is not copied in the simulator.

Instead the functionality of the `daisy_chain_block` is embedded in the device classes `Mic1LCDDisplay` and `Mic1NumPad`. The terminator functionality is contained in the `Terminator` class.

7.2.3 The Peripheral Devices

The keyboard

The keyboard is an implementation of a numpad which should have the same functionality as the numpad on the DIO5 expansion board made for the XiLinx FPGA boards. It is possible to construct a number from four digits. The number constructed is showed in seven segment display.

The numpad is implemented through the `Mic1NumPad` class which handles the communication on the bus and instantiates the `NumGrid` class which handles the graphic representation and the collection of input from the user through the buttons.

The LCD display

The goal of the implementation of the LCD display is to implement a model of the LCD display mounted on the DIO5 expansion board [1]. The model should support as much of the features and behavior of the real display as possible. The only two features which the model should not support are the possibility for the user to design her own characters for use in the display and the blinking cursor feature. Timing issues cannot be handled, but except from these it should be possible for the student to read the data sheet for the LCD driver chip and from that set up and operate the LCD display.

[1] circuit board for use together with the XiLinx FPGA boards

The LCD display is implemented through the following three classes: The `Mic1LCDDisplay` class contains the I/O bus control functionality and facilitate `LCDCharacterDisplay` which render the actual display from an array containing the text which should be displayed. The `LCDDriver` class is also facilitated by the `Mic1LCDDisplay`. This class emulates the protocol for setting up the display and communicate with it.

CHAPTER 8

Testing of the Mic-1IO Architecture

After having developed the architecture described in the previous sections of this report, we now need to make a thorough test of the developed programs to verify the functionality of the developed architecture.

This chapter will primarily focus on verifying the correctness of the developed Mic-1IO architecture. This means that the testing will focus on testing the actual design and not the extended simulator and the developed ip-blocks.

The test strategy is to test the data paths individually by connecting them in dedicated test benches.

The test of the CPU itself will be carried out by running a modified version of the standard test program and by running a simple test program using `IN` and `OUT` instructions.

8.1 Test of the terminator

The test of terminator is carried out by sending zeros to it and checking that zeros are returned. After that an address different from zero is sent to it and it is checked that `0xfeedbeef` is returned.

This test is run four times in a row to control that the system is robust and does not deadlock in a certain state and to check that the terminator does not get influenced by the value on the data input signal. In the first two test cases only zeros are sent on the data signal and and in the last two `0x25` are sent. This ensures that both `IN` and `OUT` instructions will be handled correctly. The test cases is listed in table 8.1.

The Gezel code of the test system can be seen in appendix E.1.

Test	input		expected output				comments
	addr.	data.	addr.	data	iod_ld	ioa_ld	
1	0	0	0	0	0	0	Test that nothing is returned when $addr = 0$
2	0x25	0	0	0	1	1	Test that $0xfeedbeef$ is returned when $addr \neq 0$
3	repetition of test no. 1						
4	repetition of test no. 2						
5	0	0	0	0	0	0	Test that nothing is returned when $addr = 0$
6	0x25	0x25	0	0	1	1	Test that $0xfeedbeef$ is returned when $addr \neq 0$
7	repetition of test no. 5						
8	repetition of test no. 6						

Table 8.1: *List of test cases of basic test of functionality of terminator.*

The result of the test is shown in figure 8.1.The result indicates that no errors occurred during the test.

```
> Cycle 10
testTerm: testTerm.s9 -> testTerm.finish
==============================
Terminator: Cycle: 9
Terminator: Address from micIO: 0
Terminator: Data from micIO: 0
Terminator: Address to micIO: 0
Terminator: Data to micIO: 0
Terminator: Data to micIO: 0
*****************************
Terminator test succeeded!
*****************************
finish reached !
```

Figure 8.1: *Result from terminator test.*

8.2 Test of daisy_chain_block

The `daisy_chain_block` should be tested to see if it is able to route the signals correctly based on the address sent to it, and the address limits written into it.

It should also be tested that the timing of the system is correct since the correct function of the protocol is based on a certain timing on the bus.

Both tests can be carried out at the same time since the finite state machine of the test bench is synchronised with the block.

The timing issue refers to the fact that the block should forward the data and addresses received in the same cycle as it receives them.

In table 8.2 the performed test are summarized. The test names refer to names on `sfg`'s in the source code of the test bench. It can be viewed in appendix E.2.

Test name	description
silent	If nothing is received on input nothing is written out.
forwardRD	signals are forward from cpu to the direction of the terminator.
forwardWR	same as above with diffenrent RD/WR signal value.
return	Values sent from terminator is forwarded in direction of the cpu.
catchDataWR	data sent to address in assigned address space are forwarded to the device.
catchDataRD	same as above with diffenrent RD/WR signal value.
dataFromDev	forwarding signals from device to cpu.
lowerbound	test if request to lower bound address is handled correct.
upperbound	same as above for upper bound.
underRange	are requests sent to addresses below lower limit in address space handled correct.
overRange	same as above for address above upper limit.

Table 8.2: *Summary of tests performed to control correct functioning of the daisy_chain_block. Test names refer to names on the signal flow graphs in the* `DCBtestbench` *data path which set the signals.*

In all tests output from all ports on the `daisy_chain_block` are examined also the ports which one wouldn't expect any activity on during a certain test. This is done to ensure that the `daisy_chain_block` does not "confuse" its surroundings by sending out unnecessary output.

It can be seen from the listing in figure 8.2 that a successful test was performed.

```
CYCLE: 12
Daisy chain: Data from mic2: 0
addr_to_cpu: 0/0
data_to_cpu: 0/0
addr_to_term: 6/0
data_to_term: 8/0
addr_to_dev: 0/0
data_to_dev: 0/0
iod_ld_to_cpu: 0/0
ioa_ld_to_cpu: 0/0
dev_iord: 0/0
********************************
Daisy_Chain_Block test succeeded!
********************************
finish reached !
```

Figure 8.2: *Listing of output from test documenting a successful test.*

8.3 Test of the system containing the CPU

The components of the I/O chain have now been tested individually why it is now possible to test the CPU together with them.

The test of the CPU can be divided into two parts: A test of the CPU's computational functionality. The second test should test the CPU's interaction with the I/O bus. These two test are performed through running a test program on the CPU.

The first part of the test includes testing of the IJVM instruction set. This test is carried out by running a modified version of the standard test program found on [OS]. This is modified such that it does not use several `IN` and `OUT` instruction in the end to write "OK" or "error". Instead a part is added which test the I/O chain.

The purpose of this test is to verify that the CPU interact correctly with the peripherals connected to the I/O bus. The main focus of this test is whether the CPU is able to go into polling mode and react on the replies it receives from the peripherals connected on the I/O bus. The modfied version of the program can be found in appendix E.3.

The addition to the test program by Ontko is a simple test program that communicates with the GCD calculator: The CPU sends two numbers to the GCD calculator (see appendix D.2 for the source code) which will have a latency of more than one cycle calculating the answer. In this way the CPU's ability to handle longer latencies can be verified.

Finally it is signaled whether the hole test went well. This is done by

writing a certain address on the I/O port depending on whether the test succeeded or failed. This address is collected by a modified version of the terminator and the result is written out on the screen so it can be detected.

```
ijvm: set file ./ijvmtest.ijvm
---------------- (ijvm)
      Cycle: 31883
Terminator received address: 23/0
*******************************
 Test was successful
*******************************
finish reached !
```

Figure 8.3: *Listing showing the result of the test of the CPU.*

8.4 Conclusion

It can be concluded that the system works as expected in the given test cases. No errors were found during the systematic test.

CHAPTER 9

Synthesis of the design in VHDL

This chapter gives a short summary of the efforts put into translating the Gezel code into VHDL and synthesize this code onto an FPGA.

The goal was only partly achieved since it was only possible to complete the synthesis of the reference Mic-1 design without the I/O extensions.

9.1 The Mic-1 reference version

The reference version of the Mic-1 was programmed into a Spartan 2E FPGA and the standard IJVM test program was run on this system with a successful result.

9.2 The Mic-1IO

Several attempts were performed to construct an extended system which could be programmed into the FPGA. Unfortunately none of these attempts turned out successfully.

The general problem was a too large design compared to the resources on the FPGA. The demand of LUTs was between 50 and 100 percent larger than the number of LUTs available on the FPGA. Several different designs were tried. The designs were made smaller and simpler to try to reduce the number of LUTs used. Furthermore several optimization options in the synthesis tool were tried, but these did not succeed either.

It was tried to write VHDL code that would "encourage" the synthesis tool to store the ram in block rams instead of implementing the memory in LUTs. This approach did not give a useful result except for an observation: It was possible to reduce the use of LUTs dramatically if almost the entire microprogram storage was commented out so that only a very small amount of memory were needed. This could be an indication of a lack of use of block-ram when the design is synthesized, leading to the large amount of LUTs used to synthesize the design.

In the synthesis of the reference Mic-1 almost all the ram blocks of the FPGA were used by the design. Since this is the case in the simple version but not in the extended version it is expected that this could be the reason for our problems. The only difference between the micro storage in the reference Mic-1 and the extended version is the size of the data words. The size is changed from 36 bits to 40 bits. We expect that a further investigation of the block-ram and a modification of the ROM generation tool F.1 could make the synthesis succeed. However the lack of time prevents us from going into this problem analysis.

CHAPTER 10

Conclusion

This report describes the analysis and work related to the development of an I/O environment for the Mic-1 architecture described by Tannenbaum [Tan99, Chapter 4]. The resulting architecture is called the Mic-1IO.

The I/O environment is derived partly from memory-mapped I/O and partly from daisy-chained I/O, resulting in an addressable bus connected in a daisy-chain looking fashion. The I/O environment has been constructed in this way to meet a set of simplicity requirements while maintaining a large level of flexibility.

To be able to use the developed data path, two I/O instructions have been implemented in the IJVM instructions set. The instructions have been added to the microprogram to provide the functionality of the I/O instructions in the Mic-1IO.

The developed Mic-1IO has been implemented and tested in the Gezel hardware description language. The tests performed has been completed successfully, telling us that the Mic-1IO has the capabilities described in this report.

A protocol to control the data flow on the I/O bus has been developed and tested with the Mic-1IO. The protocol has been developed to support our requirements of simplicity and flexibility.

Furthermore an existing Mic-1 simulator has been modified to function as a simulator for the new Mic-1IO.

To be able to simulate and use the developed architecture, an existing set of tools have been modified to support the features of the Mic-1IO. These tools consists of a micro-assembler and an IJVM assembler.

Bibliography

[Dig] Digilent. *Digilent DIO5 Peripheral Board - Reference Manual*. URL: `http://www.digilentinc.com/Data/Products/DIO5/DIO5-rm.pdf`.

[Lor04] Andreas V. Lorentzen. Using gezel for hardware design. IMM, DTU, 2800 Lyngby, Denmark, 2004.

[OS] Ray Ontko and Dan Stone. The mic-1 architecture. Website: `http://www.ontko.com/mic1`.

[PH98] David A. Patterson and John L. Hennesey. *Computer Organization and Design, The Hardware/Software Interface*. Morgan Kaufmann Publishers, Inc., second edition, 1998.

[RCN03] Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nikolic. *Digital Integrated Circuits - A design perspective*. Prentice Hall Electronic and VLSI series. Pearson Education International, second edition edition, 2003.

[Sam] Samsung Electronics. *Data sheet for KS0066U*. Description of 40 segment driver and controller for dot matrix lcd. URL: `http://www.eio.com/ks0066u.pdf`.

[SC04] Patrick Schaumont and Doris Ching. Gezel user manual. UCLA Electrical Engineering Department, 420 Westwood Plaza, P.O.box 951594, Los Angeles, April 2004.

[Ste93] James W. Stewart. *The 8051 Microcontroller Hardware, Software and Interfacing*. Regents/Prentice Hall, 1993.

[Tan99] Andrew S. Tanenbaum. *Structured Computer Organization*. Prentice-Hall International, fourth edition edition, 1999.

APPENDIX A

Micro assembler

A.1 mic1ioijvm.mal

```
1  // note that this is nearly identical to the example
2  // given in Tanenbaum.  Note:
3  //
4  // 1) SlashSlash-style ("//") comment characters have been added.
5  //
6  // 2) "nop" has been added as a pseudo-instruction to indicate that
7  //    nothing should be done except goto the next instruction.  It
8  //    is a do-nothing sub-instruction that allows us to have MAL
9  //    statements without a label.
10 //
11 // 3) instructions are "anchored" to locations in the control
12 //    store as defined below with the ".label" pseudo-instruction
13 //
14 // 4) a default instruction may be specified using the ".default"
15 //    pseudo-instruction.  This instruction is placed in all
16 //    unused locations of the control store by the mic1 MAL assembler.
17 //
18 // Note 2:
19 //
20 // 5) IN and OUT instructions has been modified to use the I/O registers
21 //    IOA and IOD.
22
23 // labeled statements are "anchored" at the specified control store address
24 .label    nop1        0x00
25 .label    bipush1       0x10
26 .label    ldc_w1       0x13
27 .label    iload1       0x15
28 .label    wide_iload1    0x115
29 .label    istore1       0x36
30 .label    wide_istore1    0x136
31 .label    pop1        0x57
32 .label    dup1        0x59
33 .label    swap1        0x5F
34 .label    iadd1        0x60
35 .label    isub1        0x64
36 .label    iand1        0x7E
37 .label    iinc1        0x84
38 .label    ifeq1        0x99
39 .label    iflt1        0x9B
40 .label    if_icmpeq1    0x9F
41 .label    goto1        0xA7
42 .label    ireturn1    0xAC
43 .label    ior1        0xB0
44 .label    invokevirtual1    0xB6
45 .label    wide1        0xC4
46 .label    halt1        0xFF
```

```
47  .label    err1        0xFE
48  .label    out1        0xFD
49  .label    in1        0xFC
50
51  // default instruction to place in any unused addresses of the control store
52  .default    goto err1
53
54  Main1    PC = PC + 1; fetch; goto (MBR)    // MBR holds opcode; get next byte;
55                       // dispatch
56
57  nop1    goto Main1            // Do nothing
58
59  iadd1    MAR = SP = SP - 1; rd        // Read in next-to-top word on stack
60  iadd2    H = TOS                // H = top of stack
61  iadd3    MDR = TOS = MDR + H; wr; goto Main1    // Add top two words; write
62                           // to top of stack
63
64  isub1    MAR = SP = SP - 1; rd        // Read in next-to-top word on stack
65  isub2    H = TOS                // H = top of stack
66  isub3    MDR = TOS = MDR - H; wr; goto Main1    // Do subtraction; write to
67                           //top of stack
68
69  iand1    MAR = SP = SP - 1; rd        // Read in next-to-top word on stack
70  iand2    H = TOS                // H = top of stack
71  iand3    MDR = TOS = MDR AND H; wr; goto Main1    // Do AND; write to new top
72                           // of stack
73
74  ior1    MAR = SP = SP - 1; rd        // Read in next-to-top word on stack
75  ior2    H = TOS                // H = top of stack
76  ior3    MDR = TOS = MDR OR H; wr; goto Main1    // Do OR; write to new top
77                           // of stack
78
79  dup1    MAR = SP = SP + 1        // Increment SP and copy to MAR
80  dup2    MDR = TOS; wr; goto Main1    // Write new stack word
81
82  pop1    MAR = SP = SP - 1; rd        // Read in next-to-top word on
83                       //stack
84  pop2                     // Wait for new TOS to be read
85                       //from memory
86  pop3    TOS = MDR; goto Main1        // Copy new word to TOS
87
88  swap1    MAR = SP - 1; rd        // Set MAR to SP - 1; read 2nd
89                       // word from stack
90  swap2    MAR = SP            // Set MAR to top word
91  swap3    H = MDR; wr            // Save TOS in H; write 2nd word
92                       // to top of stack
93  swap4    MDR = TOS            // Copy old TOS to MDR
94  swap5    MAR = SP - 1; wr        // Set MAR to SP - 1; write as 2nd
95                       // word on stack
96  swap6    TOS = H; goto Main1        // Update TOS
97
98  bipush1    SP = MAR = SP + 1        // MBR = the byte to push onto stack
99  bipush2    PC = PC + 1; fetch        // Increment PC, fetch next opcode
100 bipush3    MDR = TOS = MBR; wr; goto Main1    // Sign-extend constant and push on
101                      // stack
102
103
104 iload1    H = LV                // MBR contains index; copy LV to H
105 iload2    MAR = MBRU + H; rd        // MAR = address of local variable
106                      // to push
107 iload3    MAR = SP = SP + 1        // SP points to new top of stack;
108                      // prepare write
109 iload4    PC = PC + 1; fetch; wr        // Inc PC; get next opcode; write top
110                      // of stack
111 iload5    TOS = MDR; goto Main1        // Update TOS
112
113 istore1    H = LV                // MBR contains index; Copy LV to H
114 istore2    MAR = MBRU + H            // MAR = address of local variable to
115                      // store into
116 istore3    MDR = TOS; wr            // Copy TOS to MDR; write word
117 istore4    SP = MAR = SP - 1; rd        // Read in next-to-top word on stack
118 istore5    PC = PC + 1; fetch        // Increment PC; fetch next opcode
119 istore6    TOS = MDR; goto Main1        // Update TOS
120 wide1    PC = PC + 1; fetch; goto (MBR OR 0x100)    // Multiway branch with high
121                      //bit set
122
123 wide_iload1    PC = PC + 1; fetch    // MBR contains 1st index byte;
124                      // fetch 2nd
125 wide_iload2    H = MBRU << 8        // H = 1st index byte shifted left
126                      // 8 bits
127 wide_iload3    H = MBRU OR H        // H = 16-bit index of local variable
128 wide_iload4    MAR = LV + H; rd; goto iload3    // MAR = address of local
129                      //variable to push
130
```

```
131 wide_istore1    PC = PC + 1; fetch    // MBR contains 1st index byte;
132                     // fetch 2nd
133 wide_istore2    H = MBRU << 8        // H = 1st index byte shifted left
134                     // 8 bits
135 wide_istore3    H = MBRU OR H        // H = 16-bit index of local variable
136 wide_istore4    MAR = LV + H; goto istore3    // MAR = address of local
137                         //variable to store into
138
139 ldc_w1    PC = PC + 1; fetch        // MBR contains 1st index byte;
140                     // fetch 2nd
141 ldc_w2    H = MBRU << 8            // H = 1st index byte << 8
142 ldc_w3    H = MBRU OR H            // H = 16-bit index into constant pool
143 ldc_w4    MAR = H + CPP; rd; goto iload3    // MAR = address of constant in pool
144
145 iinc1    H = LV                // MBR contains index; Copy LV to H
146 iinc2    MAR = MBRU + H; rd        // Copy LV + index to MAR; Read variable
147 iinc3    PC = PC + 1; fetch        // Fetch constant
148 iinc4    H = MDR                // Copy variable to H
149 iinc5    PC = PC + 1; fetch        // Fetch next opcode
150 iinc6    MDR = MBR + H; wr; goto Main1    // Put sum in MDR; update variable
151
152 goto1    OPC = PC - 1            // Save address of opcode.
153 goto2    PC = PC + 1; fetch        // MBR = 1st byte of offset;
154                     // fetch 2nd byte
155 goto3    H = MBR << 8            // Shift and save signed first byte in H
156 goto4    H = MBRU OR H            // H = 16-bit branch offset
157 goto5    PC = OPC + H; fetch        // Add offset to OPC
158 goto6    goto Main1            // Wait for fetch of next opcode
159
160 iflt1    MAR = SP = SP - 1; rd        // Read in next-to-top word on stack
161 iflt2    OPC = TOS            // Save TOS in OPC temporarily
162 iflt3    TOS = MDR            // Put new top of stack in TOS
163 iflt4    N = OPC; if (N) goto T; else goto F    // Branch on N bit
164
165 ifeq1    MAR = SP = SP - 1; rd        // Read in next-to-top word of stack
166 ifeq2    OPC = TOS            // Save TOS in OPC temporarily
167 ifeq3    TOS = MDR            // Put new top of stack in TOS
168 ifeq4    Z = OPC; if (Z) goto T; else goto F    // Branch on Z bit
169
170 if_icmpeq1    MAR = SP = SP - 1; rd    // Read in next-to-top word of stack
171 if_icmpeq2    MAR = SP = SP - 1    // Set MAR to read in new top-of-stack
172 if_icmpeq3    H = MDR; rd        // Copy second stack word to H
173 if_icmpeq4    OPC = TOS        // Save TOS in OPC temporarily
174 if_icmpeq5    TOS = MDR        // Put new top of stack in TOS
175 if_icmpeq6    Z = OPC - H; if (Z) goto T; else goto F    // If top 2 words are
176                             // equal, goto T, else
177                             // goto F
178
179 T    OPC = PC - 1; fetch; goto goto2    // Same as goto1; needed for target
180                     // address
181
182 F    PC = PC + 1            // Skip first offset byte
183 F2    PC = PC + 1; fetch        // PC now points to next opcode
184 F3    goto Main1            // Wait for fetch of opcode
185
186 invokevirtual1    PC = PC + 1; fetch    // MBR = index byte 1; inc. PC,
187                       // get 2nd byte
188 invokevirtual2    H = MBRU << 8        // Shift and save first byte in H
189 invokevirtual3    H = MBRU OR H        // H = offset of method pointer from CPP
190 invokevirtual4    MAR = CPP + H; rd    // Get pointer to method from CPP area
191 invokevirtual5    OPC = PC + 1        // Save Return PC in OPC temporarily
192 invokevirtual6    PC = MDR; fetch        // PC points to new method; get param count
193 invokevirtual7    PC = PC + 1; fetch    // Fetch 2nd byte of parameter count
194 invokevirtual8    H = MBRU << 8        // Shift and save first byte in H
195 invokevirtual9    H = MBRU OR H        // H = number of parameters
196 invokevirtual10    PC = PC + 1; fetch    // Fetch first byte of # locals
197 invokevirtual11    TOS = SP - H        // TOS = address of OBJREF - 1
198 invokevirtual12    TOS = MAR = TOS + 1    // TOS = address of OBJREF (new LV)
199 invokevirtual13    PC = PC + 1; fetch    // Fetch second byte of # locals
200 invokevirtual14    H = MBRU << 8        // Shift and save first byte in H
201 invokevirtual15    H = MBRU OR H        // H = # locals
202 invokevirtual16    MDR = SP + H + 1; wr    // Overwrite OBJREF with link pointer
203 invokevirtual17    MAR = SP = MDR;        // Set SP, MAR to location to hold old PC
204 invokevirtual18    MDR = OPC; wr        // Save old PC above the local variables
205 invokevirtual19    MAR = SP = SP + 1    // SP points to location to hold old LV
206 invokevirtual20    MDR = LV; wr        // Save old LV above saved PC
207 invokevirtual21    PC = PC + 1; fetch    // Fetch first opcode of new method.
208 invokevirtual22    LV = TOS; goto Main1    // Set LV to point to LV Frame
209
210 ireturn1    MAR = SP = LV; rd    // Reset SP, MAR to get link pointer
211 ireturn2                // Wait for read
212 ireturn3    LV = MAR = MDR; rd    // Set LV to link ptr; get old PC
213 ireturn4    MAR = LV + 1        // Set MAR to read old LV
214 ireturn5    PC = MDR; rd; fetch    // Restore PC; fetch next opcode
```

```
ireturn6    MAR = SP        // Set MAR to write TOS
ireturn7    LV = MDR        // Restore LV
ireturn8    MDR = TOS; wr; goto Main1    // Save return value on original
                         // top of stack

halt1    goto halt1

err1    OPC=H=-1
        OPC=H+OPC
        MAR=H+OPC            // compute IO address
    OPC=H=1                 // 1
    OPC=H=H+OPC             // 10
    OPC=H=H+OPC             // 100
    OPC=H=H+OPC             // 1000
    OPC=H=H+OPC+1             // 10001
    OPC=H=H+OPC             // 100010
    MDR=H+OPC+1;wr             // 1000101 'E'
    OPC=H=1                 // 1
    OPC=H=H+OPC             // 10
    OPC=H=H+OPC+1             // 101
    OPC=H=H+OPC             // 1010
    OPC=H=H+OPC             // 10100
    OPC=H=H+OPC+1             // 101001
    MDR=H+OPC;wr             // 1010010 'R'
        nop
    MDR=H+OPC;wr             // 1010010 'R'
    OPC=H=1                 // 1
    OPC=H=H+OPC             // 10
    OPC=H=H+OPC             // 100
    OPC=H=H+OPC+1             // 1001
    OPC=H=H+OPC+1             // 10011
    OPC=H=H+OPC+1             // 100111
    MDR=H+OPC+1;wr             // 1001111 'O'
    OPC=H=1                 // 1
    OPC=H=H+OPC             // 10
    OPC=H=H+OPC+1             // 101
    OPC=H=H+OPC             // 1010
    OPC=H=H+OPC             // 10100
    OPC=H=H+OPC+1             // 101001
    MDR=H+OPC;wr             // 1010010 'R'
    goto halt1

out1    MAR = SP = SP - 1; rd              // Read in next-to-top word
                         // on stack
out2    IOD = TOS                  // Transfer TOS to IOD
out3    IOA = MDR                  // Activate OUT by setting
                         // IOA = MDR
out4    Z = IOA; if (Z) goto out5; else goto out4 // Loop until IOA is changed
                         // to zero
out5    MDR = TOS = IOD; wr              // Make error detection
                         // possible (0xFEEDBEEF)
out6    goto Main1                 // Return to Main

in1    IOA = TOS                 // Transfer TOS to IOA
in2    iord                     // Supply read signal to
                         // peripherals
in3    Z = IOA; if (Z) goto in4; else goto in3      // Loop until IOA = zero
in4    MDR = TOS = IOD; wr              // Make error detection
                         // possible (0xFEEDBEEF)
in5    goto Main1                 // Return to Main
```

A.2 Mic1Parser.cup

```
1   // CUP specification for mic1asm.
2
3   // added IOD, IOA and iord
4
5
6   import java_cup.runtime.*;
7   import java.lang.*;
8   import java.util.*;
9
10  /* Preliminaries to set up and use the scanner.  */
11  action code
12  {:
13  private Mic1Parse mic1parse = new Mic1Parse() ;
14  private Mic1Statement s = new Mic1Statement() ;
15  private int instructionCount = 0 ;
16  :} ;
17
```

```
18  // init with {: Mic1Scanner.init( );              :};
19  scan with {: return Mic1Scanner.next_token(); :};
20
21  /* Terminals (tokens returned by the scanner). */
22  terminal EOL ;
23  terminal PLUS ;
24  terminal MINUS ;
25  terminal ZERO ;
26  terminal ONE ;
27  terminal EIGHT ;
28  terminal LESSTHAN ;
29  terminal EQUALS ;
30  terminal GREATERTHAN ;
31  terminal LPAREN ;
32  terminal RPAREN ;
33  terminal SEMI ;
34  terminal Integer address ;
35  terminal String label ;
36  terminal rd ;
37  terminal wr ;
38  terminal fetch ;
39  terminal iord ; // Modification
40  terminal IF ;
41  terminal ELSE ;
42  terminal GOTO ;
43  terminal N ;
44  terminal Z ;
45  terminal MBR ;
46  terminal MAR ;
47  terminal MDR ;
48  terminal PC ;
49  terminal SP ;
50  terminal LV ;
51  terminal CPP ;
52  terminal TOS ;
53  terminal OPC ;
54  terminal H ;
55  terminal IOD ; // Modification
56  terminal IOA ; // Modification
57  terminal MBRU ;
58  terminal OR ;
59  terminal AND ;
60  terminal NOT ;
61  terminal NOP ;
62  terminal DOTLABEL ;
63  terminal DOTDEFAULT ;
64
65  /* Non terminals */
66  non terminal Mic1Parse       program ;
67  non terminal                 line_sequence ;
68  non terminal                 line ;
69  non terminal                 instruction ;
70  non terminal                 directive ;
71  non terminal                 statement_sequence ;
72  non terminal                 statement ;
73  non terminal                 io_statement ;
74  non terminal                 control_statement ;
75  non terminal                 if_statement ;
76  non terminal                 condition ;
77  non terminal                 multiway_branch_statement ;
78  non terminal                 mb_expr ;
79  non terminal                 goto_statement ;
80  non terminal                 nop_statement ;
81  non terminal                 assignment_statement ;
82  non terminal                 target ;
83  non terminal                 c_register ;
84  non terminal                 expr ;
85  non terminal                 operation ;
86  non terminal                 a_term ;
87  non terminal                 b_term ;
88
89  /* The grammar */
90
91  start with program ;
92
93  program ::= line_sequence
94  {: RESULT = mic1parse ; :}
95          ;
96
97  line_sequence ::= line line_sequence
98                      |
99                      ;
100
101 line ::=  instruction EOL
```

```
102 {: instructionCount ++ ; :}
103      | directive EOL
104      | EOL
105      ;
106
107 instruction ::= label:l statement_sequence
108 {: s.label = l ;
109    mic1parse.add_statement( s ) ; s = new Mic1Statement() ; :}
110               | statement_sequence
111 {: s.label = new String( "%" + instructionCount ) ;
112    mic1parse.add_statement( s ) ; s = new Mic1Statement() ; :}
113               | label:l
114 {: s.label = l ;
115    mic1parse.add_statement( s ) ; s = new Mic1Statement() ; :}
116             ;
117
118 directive ::= DOTLABEL label:l address:a
119 // we should verify that the label has not already been defined
120 {: mic1parse.add_label( l , a ) ; :}
121            | DOTDEFAULT statement_sequence
122 // we should verify that a default has not already been defined
123 {: mic1parse.defaultStatement = s ; s = new Mic1Statement() ; :}
124            ;
125
126 statement_sequence ::= statement SEMI statement_sequence
127                      | statement SEMI
128                      | statement
129                      ;
130
131 statement ::= io_statement
132            | control_statement
133            | assignment_statement
134            | nop_statement
135            ;
136
137 io_statement ::= rd
138 {: s.i.READ = true ; :}
139               | wr
140 {: s.i.WRITE = true ; :}
141               | fetch
142 {: s.i.FETCH = true ; :}
143           | iord    // Modification
144 {: s.i.IORD = true ; :}  // Modification
145               ;     // Modification
146
147 control_statement ::= if_statement
148 {: s.isIf = true ; :}
149                     | multiway_branch_statement
150 {: s.isMultiway = true ; :}
151                     | goto_statement
152 {: s.isGoto = true; :}
153                     ;
154
155 if_statement ::= IF LPAREN condition RPAREN GOTO label:g SEMI ELSE GOTO label:e
156 {: s.gotoLabel = g ; s.elseLabel = e ; :}
157             ;
158
159 condition ::= N
160 {: s.i.JAMN = true ; :}
161            | Z
162 {: s.i.JAMZ = true ; :}
163            ;
164
165 multiway_branch_statement ::= GOTO LPAREN mb_expr RPAREN
166 {: s.i.JMPC = true ; :}
167                           ;
168
169 mb_expr ::= MBR OR address:a
170 {: s.i.NEXT_ADDRESS = a.intValue() ; :}
171          | MBR
172 {: s.i.NEXT_ADDRESS = 0 ; :}
173        ;
174
175 goto_statement ::= GOTO label:l
176 {: s.gotoLabel = l ; :}
177                ;
178
179 assignment_statement ::= target EQUALS assignment_statement
180                        | expr
181                        ;
182
183 target ::= c_register
184 // Note that N and Z aren't really targets; they're place-holders.
185 // We don't need to modify the instruction if they're used.
```

```
186         | N
187         | Z
188         ;
189
190   c_register ::= MAR
191   {: s.i.MAR = true ; :}
192               | MDR
193   {: s.i.MDR = true ; :}
194               | PC
195   {: s.i.PC = true ; :}
196               | SP
197   {: s.i.SP = true ; :}
198               | LV
199   {: s.i.LV = true ; :}
200               | CPP
201   {: s.i.CPP = true ; :}
202               | TOS
203   {: s.i.TOS = true ; :}
204               | OPC
205   {: s.i.OPC = true ; :}
206               | H
207   {: s.i.H = true ; :}
208         | IOD       // Modification
209   {: s.i.IOD = true ; :} // Modification
210         | IOA       // Modification
211   {: s.i.IOA = true ; :} // Modification
212               ;           // Modification
213
214   expr ::= operation
215         | operation LESSTHAN LESSTHAN EIGHT
216   {: s.i.SLL8 = true ; :}
217         | operation GREATERTHAN GREATERTHAN ONE
218   {: s.i.SRA1 = true ; :}
219         ;
220
221   operation ::= a_term AND b_term
222   {: s.i.F0 = false ; s.i.F1 = false ;
223      s.i.ENA = true ; s.i.ENB = true ; s.i.INVA = false ; s.i.INC = false ; :}
224               | b_term AND a_term
225   {: s.i.F0 = false ; s.i.F1 = false ;
226      s.i.ENA = true ; s.i.ENB = true ; s.i.INVA = false ; s.i.INC = false ; :}
227               | a_term OR b_term
228   {: s.i.F0 = false ; s.i.F1 = true ;
229      s.i.ENA = true ; s.i.ENB = true ; s.i.INVA = false ; s.i.INC = false ; :}
230               | b_term OR a_term
231   {: s.i.F0 = false ; s.i.F1 = true ;
232      s.i.ENA = true ; s.i.ENB = true ; s.i.INVA = false ; s.i.INC = false ; :}
233               | NOT b_term
234   {: s.i.F0 = true ; s.i.F1 = false ;
235      s.i.ENA = false ; s.i.ENB = true ; s.i.INVA = false ; s.i.INC = false ; :}
236               | NOT a_term
237   {: s.i.F0 = false ; s.i.F1 = true ;
238      s.i.ENA = true ; s.i.ENB = false ; s.i.INVA = true ; s.i.INC = false ; :}
239               | b_term PLUS a_term
240   {: s.i.F0 = true ; s.i.F1 = true ;
241      s.i.ENA = true ; s.i.ENB = true ; s.i.INVA = false ; s.i.INC = false ; :}
242               | a_term PLUS b_term
243   {: s.i.F0 = true ; s.i.F1 = true ;
244      s.i.ENA = true ; s.i.ENB = true ; s.i.INVA = false ; s.i.INC = false ; :}
245               | b_term PLUS ONE
246   {: s.i.F0 = true ; s.i.F1 = true ;
247      s.i.ENA = false ; s.i.ENB = true ; s.i.INVA = false ; s.i.INC = true ; :}
248               | a_term PLUS ONE
249   {: s.i.F0 = true ; s.i.F1 = true ;
250      s.i.ENA = true ; s.i.ENB = false ; s.i.INVA = false ; s.i.INC = true ; :}
251               | b_term MINUS a_term
252   {: s.i.F0 = true ; s.i.F1 = true ;
253      s.i.ENA = true ; s.i.ENB = true ; s.i.INVA = true ; s.i.INC = true ; :}
254               | MINUS a_term
255   {: s.i.F0 = true ; s.i.F1 = true ;
256      s.i.ENA = true ; s.i.ENB = false ; s.i.INVA = true ; s.i.INC = true ; :}
257               | b_term MINUS ONE
258   {: s.i.F0 = true ; s.i.F1 = true ;
259      s.i.ENA = false ; s.i.ENB = true ; s.i.INVA = true ; s.i.INC = false ; :}
260               | b_term PLUS a_term PLUS ONE
261   {: s.i.F0 = true ; s.i.F1 = true ;
262      s.i.ENA = true ; s.i.ENB = true ; s.i.INVA = false ; s.i.INC = true ; :}
263               | a_term PLUS b_term PLUS ONE
264   {: s.i.F0 = true ; s.i.F1 = true ;
265      s.i.ENA = true ; s.i.ENB = true ; s.i.INVA = false ; s.i.INC = true ; :}
266               | b_term
267   {: s.i.F0 = false ; s.i.F1 = true ;
268      s.i.ENA = false ; s.i.ENB = true ; s.i.INVA = false ; s.i.INC = false ; :}
269               | a_term
```

```
{: s.i.F0 = false ; s.i.F1 = true ;
   s.i.ENA = true ; s.i.ENB = false ; s.i.INVA = false ; s.i.INC = false ; :}
            | MINUS ONE
{: s.i.F0 = false ; s.i.F1 = true ;
   s.i.ENA = false ; s.i.ENB = false ; s.i.INVA = true ; s.i.INC = false ; :}
            | ZERO
{: s.i.F0 = false ; s.i.F1 = true ;
   s.i.ENA = false ; s.i.ENB = false ; s.i.INVA = false ; s.i.INC = false ; :}
            | ONE
{: s.i.F0 = false ; s.i.F1 = true ;
   s.i.ENA = false ; s.i.ENB = false ; s.i.INVA = false ; s.i.INC = true ; :}
            ;

b_term ::= MDR
{: s.i.B = Mic1Instruction.B_MDR ; :}
            | PC
{: s.i.B = Mic1Instruction.B_PC ; :}
            | MBR
{: s.i.B = Mic1Instruction.B_MBR ; :}
            | MBRU
{: s.i.B = Mic1Instruction.B_MBRU ; :}
            | SP
{: s.i.B = Mic1Instruction.B_SP ; :}
            | LV
{: s.i.B = Mic1Instruction.B_LV ; :}
            | CPP
{: s.i.B = Mic1Instruction.B_CPP ; :}
            | TOS
{: s.i.B = Mic1Instruction.B_TOS ; :}
            | OPC
{: s.i.B = Mic1Instruction.B_OPC ; :}
        | IOD              // Modification
{: s.i.B = Mic1Instruction.B_IOD ; :} // Modification
        | IOA                    // Modification
{: s.i.B = Mic1Instruction.B_IOA ; :} // Modification
            ;                  // Modification

a_term ::= H
         ;

nop_statement ::= NOP
                ;
```

A.3 Mic1Instruction.java

```
/*
*
*  Mic1Instruction.java
*
*  mic1 microarchitecture simulator
*  Copyright (C) 1999, Prentice-Hall, Inc.
*
*  This program is free software; you can redistribute it and/or modify
*  it under the terms of the GNU General Public License as published by
*  the Free Software Foundation; either version 2 of the License, or
*  (at your option) any later version.
*
*  This program is distributed in the hope that it will be useful, but
*  WITHOUT ANY WARRANTY; without even the implied warranty of
*  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
*  Public License for more details.
*
*  You should have received a copy of the GNU General Public License along with
*  this program; if not, write to:
*
*    Free Software Foundation, Inc.
*    59 Temple Place - Suite 330
*    Boston, MA 02111-1307, USA.
*
*  A copy of the GPL is available online the GNU web site:
*
*    http://www.gnu.org/copyleft/gpl.html
*
*/

/*
Mic1Instruction

This class represents an instruction which might appear
in the Mic-1 control store.
```

```
36
37  To Do
38
39  We should probably create a separate class for Mic1Reader
40  and move the read method from here to the Mic1Reader class.
41
42  Modification History
43
44  Name             Date       Comment
45  ---------------- ---------- ----------------------------------------
46  Ray Ontko        1998.09.01 Created
47
48  */
49
50  import java.io.* ;
51  import java.lang.* ;
52
53  public class Mic1Instruction
54  {
55  public static final int B_MDR = 0 ;
56  public static final int B_PC = 1 ;
57  public static final int B_MBR = 2 ;
58  public static final int B_MBRU = 3 ;
59  public static final int B_SP = 4 ;
60  public static final int B_LV = 5 ;
61  public static final int B_CPP = 6 ;
62  public static final int B_TOS = 7 ;
63  public static final int B_OPC = 8 ;
64  public static final int B_IOD = 9 ; // Decoder value for b-bus access from IOD register (B_sel)
65  public static final int B_IOA = 10 ;// Decoder value for b-bus access from IOA register (B_sel)
66
67  public int NEXT_ADDRESS = 0 ;
68  public boolean JMPC = false ;
69  public boolean JAMN = false ;
70  public boolean JAMZ = false ;
71  public boolean SLL8 = false ;
72  public boolean SRA1 = false ;
73  public boolean F0 = false ;
74  public boolean F1 = false ;
75  public boolean ENA = false ;
76  public boolean ENB = false ;
77  public boolean INVA = false ;
78  public boolean INC = false ;
79  public boolean H = false ;
80  public boolean IOD = false ; // IOD signal value of instruction format
81  public boolean IOA = false ; // IOA signal value of instruction format
82  public boolean OPC = false ;
83  public boolean TOS = false ;
84  public boolean CPP = false ;
85  public boolean LV = false ;
86  public boolean SP = false ;
87  public boolean PC = false ;
88  public boolean MDR = false ;
89  public boolean MAR = false ;
90  public boolean IORD = false ; // IORD signal value of instruction format
91  public boolean WRITE = false ;
92  public boolean READ = false ;
93  public boolean FETCH = false ;
94  public int B = 0 ;
95
96
97  public Mic1Instruction()
98  {
99  }
100
101
102 public int read( InputStream in ) throws IOException
103 // public int read( Reader in ) throws IOException
104 {
105 int b0 ;
106 int b1 ;
107 int b2 ;
108 int b3 ;
109 int b4 ;
110
111 if ( ( b0 = in.read() ) == -1 ) return( -1 ) ;
112 if ( ( b1 = in.read() ) == -1 ) return( -1 ) ;
113 if ( ( b2 = in.read() ) == -1 ) return( -1 ) ;
114 if ( ( b3 = in.read() ) == -1 ) return( -1 ) ;
115 if ( ( b4 = in.read() ) == -1 ) return( -1 ) ;
116
117 // The following section of code is used by the simulator to read/decode
118 // an instruction read from memory.
119 NEXT_ADDRESS = ( b0 << 1 ) | ( ( b1 & 0x80 ) >> 7 ) ;
```

```
120   if ( ( b1 & 0x40 ) > 0 ) JMPC = true ;
121   if ( ( b1 & 0x20 ) > 0 ) JAMN = true ;
122   if ( ( b1 & 0x10 ) > 0 ) JAMZ = true ;
123   if ( ( b1 & 0x08 ) > 0 ) SLL8 = true ;
124   if ( ( b1 & 0x04 ) > 0 ) SRA1 = true ;
125   if ( ( b1 & 0x02 ) > 0 ) F0 = true ;
126   if ( ( b1 & 0x01 ) > 0 ) F1 = true ;
127   if ( ( b2 & 0x80 ) > 0 ) ENA = true ;
128   if ( ( b2 & 0x40 ) > 0 ) ENB = true ;
129   if ( ( b2 & 0x20 ) > 0 ) INVA = true ;
130   if ( ( b2 & 0x10 ) > 0 ) INC = true ;
131   if ( ( b2 & 0x08 ) > 0 ) IOA = true ;
132   if ( ( b2 & 0x04 ) > 0 ) IOD = true ;
133   if ( ( b2 & 0x02 ) > 0 ) H = true ;
134   if ( ( b2 & 0x01 ) > 0 ) OPC = true ;
135   if ( ( b3 & 0x80 ) > 0 ) TOS = true ;
136   if ( ( b3 & 0x40 ) > 0 ) CPP = true ;
137   if ( ( b3 & 0x20 ) > 0 ) LV = true ;
138   if ( ( b3 & 0x10 ) > 0 ) SP = true ;
139   if ( ( b3 & 0x08 ) > 0 ) PC = true ;
140   if ( ( b3 & 0x04 ) > 0 ) MDR = true ;
141   if ( ( b3 & 0x02 ) > 0 ) MAR = true ;
142   if ( ( b3 & 0x01 ) > 0 ) IORD = true ;
143   if ( ( b4 & 0x80 ) > 0 ) WRITE = true ;
144   if ( ( b4 & 0x40 ) > 0 ) READ = true ;
145   if ( ( b4 & 0x20 ) > 0 ) FETCH = true ;
146   B = ( b4 << 3 & 0xF0 ) >> 4 ;
147
148   return( 1 ) ;
149   }
150
151   public void write( OutputStream out ) throws IOException
152   {
153   int b0 = 0 ;
154   int b1 = 0 ;
155   int b2 = 0 ;
156   int b3 = 0 ;
157   int b4 = 0 ;
158
159   // The following section of code does the actual coding of the
160   // parsed instructions.
161   b0 = ( NEXT_ADDRESS & 0x1FE ) >> 1 ;
162   b1 = ( NEXT_ADDRESS & 0x01 ) << 7 ;
163   if ( JMPC ) b1 = b1 | 0x40 ;
164   if ( JAMN ) b1 = b1 | 0x20 ;
165   if ( JAMZ ) b1 = b1 | 0x10 ;
166   if ( SLL8 ) b1 = b1 | 0x08 ;
167   if ( SRA1 ) b1 = b1 | 0x04 ;
168   if ( F0 ) b1 = b1 | 0x02 ;
169   if ( F1 ) b1 = b1 | 0x01 ;
170   if ( ENA ) b2 = b2 | 0x80 ;
171   if ( ENB ) b2 = b2 | 0x40 ;
172   if ( INVA ) b2 = b2 | 0x20 ;
173   if ( INC ) b2 = b2 | 0x10 ;
174   if ( IOA ) b2 = b2 | 0x08 ;
175   if ( IOD ) b2 = b2 | 0x04 ;
176   if ( H ) b2 = b2 | 0x02 ;
177   if ( OPC ) b2 = b2 | 0x01 ;
178   if ( TOS ) b3 = b3 | 0x80 ;
179   if ( CPP ) b3 = b3 | 0x40 ;
180   if ( LV ) b3 = b3 | 0x20 ;
181   if ( SP ) b3 = b3 | 0x10 ;
182   if ( PC ) b3 = b3 | 0x08 ;
183   if ( MDR ) b3 = b3 | 0x04 ;
184   if ( MAR ) b3 = b3 | 0x02 ;
185   if ( IORD ) b3 = b3 | 0x01 ;
186   if ( WRITE ) b4 = b4 | 0x80 ;
187   if ( READ ) b4 = b4 | 0x40 ;
188   if ( FETCH ) b4 = b4 | 0x20 ;
189   b4 = b4 | ( ( B & 0x0F ) << 1 ) ;
190
191   out.write( b0 ) ;
192   out.write( b1 ) ;
193   out.write( b2 ) ;
194   out.write( b3 ) ;
195   out.write( b4 ) ;
196   }
197
198   public String toString()
199   {
200   String s = "" ;
201   String a = "H" ;
202   String b = "" ;
203   String c = "" ;
```

```
204
205   // decode the C-bus bits
206   if ( H )
207     s = s + "H=" ;
208   if ( OPC )
209     s = s + "OPC=" ;
210   if ( TOS )
211     s = s + "TOS=" ;
212   if ( CPP )
213     s = s + "CPP=" ;
214   if ( LV )
215     s = s + "LV=" ;
216   if ( SP )
217     s = s + "SP=" ;
218   if ( PC )
219     s = s + "PC=" ;
220   if ( MDR )
221     s = s + "MDR=" ;
222   if ( MAR )
223     s = s + "MAR=" ;
224   if ( IOD )
225     s = s + "IOD=" ;
226   if ( IOA )
227     s = s + "IOA=" ;
228
229   // decode the B-bus bits
230   switch ( B )
231   {
232   case B_MDR: b = "MDR" ; break ;
233   case B_PC: b = "PC" ; break ;
234   case B_MBR: b = "MBR" ; break ;
235   case B_MBRU: b = "MBRU" ; break ;
236   case B_SP: b = "SP" ; break ;
237   case B_LV: b = "LV" ; break ;
238   case B_CPP: b = "CPP" ; break ;
239   case B_TOS: b = "TOS" ; break ;
240   case B_OPC: b = "OPC" ; break ;
241   case B_IOD: b = "IOD" ; break ;
242   case B_IOA: b = "IOA" ; break ;
243   // are the rest of these really no-ops?
244   case 11: b = "R0B" ; break ;
245   case 12: b = "R0C" ; break ;
246   case 13: b = "R0D" ; break ;
247   case 14: b = "R0E" ; break ;
248   case 15: b = "R0F" ; break ;
249   }
250
251   // decode the ALU operation
252   if ( F0 == false && F1 == false ) // a AND b
253     {
254     if ( ENA )
255       {
256       if ( ENB )
257         {
258         if ( INVA )
259           if ( INC )
260             s = s + "((NOT " + a + ") AND " + b + ")+1";
261           else
262             s = s + "(NOT " + a + ") AND " + b ;
263         else
264           if ( INC )
265             s = s + "(" + a + " AND " + b + ")+1";
266           else
267             s = s + a + " AND " + b ;
268         }
269       else
270         {
271         if ( INC )
272           s = s + "1";
273         else
274           s = s + "0" ;
275         }
276       }
277     else
278       {
279       if ( ENB )
280         {
281         if ( INVA )
282           if ( INC )
283             s = s + b + "+1";
284           else
285             s = s + b ;
286         else
287           if ( INC )
```

```
288             s = s + "1";
289           else
290             s = s + "0" ;
291         }
292       else
293         {
294         if ( INC )
295           s = s + "1";
296         else
297           s = s + "0" ;
298         }
299       }
300   }
301 else if ( F0 == false && F1 == true ) // a OR b
302   {
303   if ( ENA )
304     {
305     if ( ENB )
306       {
307       if ( INVA )
308         if ( INC )
309           s = s + "((NOT " + a + ") OR " + b + ")+1";
310         else
311           s = s + "(NOT " + a + ") OR " + b ;
312       else
313         if ( INC )
314           s = s + "(" + a + " OR " + b + ")+1";
315         else
316           s = s + a + " OR " + b ;
317       }
318     else
319       {
320       if ( INVA )
321         if ( INC )
322           s = s + "-" + a ;
323         else
324           s = s + "NOT " + a ;
325       else
326         if ( INC )
327           s = s + a + "+1";
328         else
329           s = s + a ;
330       }
331     }
332   else
333     {
334     if ( ENB )
335       {
336       if ( INVA )
337         if ( INC )
338           s = s + "0";
339         else
340           s = s + "-1" ;
341       else
342         if ( INC )
343           s = s + b + "+1";
344         else
345           s = s + b ;
346       }
347     else
348       {
349       if ( INVA )
350         if ( INC )
351           s = s + "0";
352         else
353           s = s + "-1" ;
354       else
355         if ( INC )
356           s = s + "1";
357         else
358           s = s + "0" ;
359       }
360     }
361   }
362 else if ( F0 == true && F1 == false ) // NOT b
363   {
364   if ( ENB )
365     if ( INC )
366       s = s + "-" + b ;
367     else
368       s = s + "NOT " + b ;
369   else
370     if ( INC )
371       s = s + "-1" ;
```

```
372       else
373         s = s + "0" ;
374     }
375   else if ( F0 == true && F1 == true ) // a + b
376     {
377     if ( ENA )
378       {
379       if ( ENB )
380         {
381         if ( INVA )
382           if ( INC )
383             s = s + b + "-" + a ;
384           else
385             s = s + b + "-" + a + "-1" ;
386         else
387           if ( INC )
388             s = s + a + "+" + b + "+1" ;
389           else
390             s = s + a + "+" + b ;
391         }
392       else
393         {
394         if ( INVA )
395           if ( INC )
396             s = s + "-" + a ;
397           else
398             s = s + "-" + a + "-1" ;
399         else
400           if ( INC )
401             s = s + a + "+1" ;
402           else
403             s = s + a ;
404         }
405       }
406     else
407       {
408       if ( ENB )
409         {
410         if ( INVA )
411           if ( INC )
412             s = s + b ;
413           else
414             s = s + b + "-1" ;
415         else
416           if ( INC )
417             s = s + b + "+1" ;
418           else
419             s = s + b ;
420         }
421       else
422         {
423         if ( INVA )
424           if ( INC )
425             s = s + "0" ;
426           else
427             s = s + "-1" ;
428         else
429           if ( INC )
430             s = s + "1" ;
431           else
432             s = s + "0" ;
433         }
434       }
435     }
436
437   // decode the shifter operation
438   if ( SRA1 )
439     s = s + ">>1" ;
440   if ( SLL8 )
441     s = s + "<<8" ;
442
443
444   // decode the mem bits
445   if ( WRITE )
446     s = s + ";wr" ;
447   if ( READ )
448     s = s + ";rd" ;
449   if ( FETCH )
450     s = s + ";fetch" ;
451
452   //decode I/O bit
453   if ( IORD )
454     s = s + ";iord" ;
455
```

```
456 // decode the JAM bits/addr
457 if ( JAMN )
458   s = "N=" + s +
459       ";if (N) goto 0x" + Integer.toHexString( NEXT_ADDRESS | 256 ).toUpperCase() +
460       "; else goto 0x" + Integer.toHexString( NEXT_ADDRESS ).toUpperCase() ;
461 else if ( JAMZ )
462   s = "Z=" + s +
463       ";if (Z) goto 0x" + Integer.toHexString( NEXT_ADDRESS | 256 ).toUpperCase() +
464       "; else goto 0x" + Integer.toHexString( NEXT_ADDRESS ).toUpperCase() ;
465 else if ( JMPC )
466   {
467   if ( NEXT_ADDRESS == 0 )
468      s = s + ";goto (MBR)" ;
469   else
470      s = s + ";goto (MBR OR 0x" + Integer.toHexString( NEXT_ADDRESS ).toUpperCase() + ")" ;
471   }
472 else
473   s = s + ";goto 0x" + Integer.toHexString( NEXT_ADDRESS ).toUpperCase() ;
474
475 if ( s.equals( "0" ) )
476   s = "nop" ;
477 else if( s.startsWith( "0;" ) )
478   s = s.substring( 2 ) ;
479
480 return( s ) ;
481 }
482
483 public static void main( String args[] )
484 {
485 Mic1Instruction i = null ;
486
487 i = new Mic1Instruction() ;
488 i.WRITE = true ;
489 System.out.println( i.toString() ) ;
490 }
491
492 }
```

A.4 Mic1Scanner.java

```
1  /*
2  *
3  *  Mic1Scanner.java
4  *
5  *  mic1 microarchitecture simulator
6  *  Copyright (C) 1999, Prentice-Hall, Inc.
7  *
8  *  This program is free software; you can redistribute it and/or modify
9  *  it under the terms of the GNU General Public License as published by
10 *  the Free Software Foundation; either version 2 of the License, or
11 *  (at your option) any later version.
12 *
13 *  This program is distributed in the hope that it will be useful, but
14 *  WITHOUT ANY WARRANTY; without even the implied warranty of
15 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
16 *  Public License for more details.
17 *
18 *  You should have received a copy of the GNU General Public License along with
19 *  this program; if not, write to:
20 *
21 *    Free Software Foundation, Inc.
22 *    59 Temple Place - Suite 330
23 *    Boston, MA 02111-1307, USA.
24 *
25 *  A copy of the GPL is available online the GNU web site:
26 *
27 *    http://www.gnu.org/copyleft/gpl.html
28 *
29 */
30
31 // Simple mic1asm scanner class
32
33 import java_cup.runtime.*;
34 import java.io.* ;
35
36 public class Mic1Scanner {
37
38 private static StreamTokenizer st = null ;
39
40 public static void init( StreamTokenizer st )
```

```
41  {
42  Mic1Scanner.st = st ;
43  }
44
45  public static Symbol next_token() throws java.io.IOException
46  {
47  int t = st.nextToken() ;
48  if ( t ==  StreamTokenizer.TT_WORD )
49    {
50    // System.out.println( "token = " + st.sval ) ;
51    // we can replace much of the following with a hash table if desired
52    if ( st.sval.equals( "0" ) ) return new Symbol(Mic1Symbol.ZERO) ;
53    if ( st.sval.equals( "1" ) ) return new Symbol(Mic1Symbol.ONE) ;
54    if ( st.sval.equals( "8" ) ) return new Symbol(Mic1Symbol.EIGHT) ;
55    if ( st.sval.equals( "rd" ) ) return new Symbol(Mic1Symbol.rd) ;
56    if ( st.sval.equals( "wr" ) ) return new Symbol(Mic1Symbol.wr) ;
57    if ( st.sval.equals( "fetch" ) ) return new Symbol(Mic1Symbol.fetch) ;
58    if ( st.sval.equals( "iord" ) ) return new Symbol(Mic1Symbol.iord) ;    // iord symbols
59                                    // are created
60    if ( st.sval.equals( "if" ) ) return new Symbol(Mic1Symbol.IF) ;
61    if ( st.sval.equals( "else" ) ) return new Symbol(Mic1Symbol.ELSE) ;
62    if ( st.sval.equals( "goto" ) ) return new Symbol(Mic1Symbol.GOTO) ;
63    if ( st.sval.equals( "nop" ) ) return new Symbol(Mic1Symbol.NOP) ;
64    if ( st.sval.equals( "N" ) ) return new Symbol(Mic1Symbol.N) ;
65    if ( st.sval.equals( "Z" ) ) return new Symbol(Mic1Symbol.Z) ;
66    if ( st.sval.equals( "MBR" ) ) return new Symbol(Mic1Symbol.MBR) ;
67    if ( st.sval.equals( "MAR" ) ) return new Symbol(Mic1Symbol.MAR) ;
68    if ( st.sval.equals( "MDR" ) ) return new Symbol(Mic1Symbol.MDR) ;
69    if ( st.sval.equals( "PC" ) ) return new Symbol(Mic1Symbol.PC) ;
70    if ( st.sval.equals( "SP" ) ) return new Symbol(Mic1Symbol.SP) ;
71    if ( st.sval.equals( "LV" ) ) return new Symbol(Mic1Symbol.LV) ;
72    if ( st.sval.equals( "CPP" ) ) return new Symbol(Mic1Symbol.CPP) ;
73    if ( st.sval.equals( "TOS" ) ) return new Symbol(Mic1Symbol.TOS) ;
74    if ( st.sval.equals( "OPC" ) ) return new Symbol(Mic1Symbol.OPC) ;
75    if ( st.sval.equals( "H" ) ) return new Symbol(Mic1Symbol.H) ;
76    if ( st.sval.equals( "IOD" ) ) return new Symbol(Mic1Symbol.IOD) ;// IOD symbols
77                                    // are created
78    if ( st.sval.equals( "IOA" ) ) return new Symbol(Mic1Symbol.IOA) ;// IOA symbols
79                                    // are created
80    if ( st.sval.equals( "MBRU" ) ) return new Symbol(Mic1Symbol.MBRU) ;
81    if ( st.sval.equals( "OR" ) ) return new Symbol(Mic1Symbol.OR) ;
82    if ( st.sval.equals( "AND" ) ) return new Symbol(Mic1Symbol.AND) ;
83    if ( st.sval.equals( "NOT" ) ) return new Symbol(Mic1Symbol.NOT) ;
84    if ( st.sval.equals( ".label" ) ) return new Symbol(Mic1Symbol.DOTLABEL) ;
85    if ( st.sval.equals( ".default" ) ) return new Symbol(Mic1Symbol.DOTDEFAULT) ;
86    if ( st.sval.startsWith( "0x" ) || st.sval.startsWith( "0X" ) )
87       {
88       return new Symbol(Mic1Symbol.address, Integer.valueOf(st.sval.substring(2),16));
89       }
90    // else
91    return new Symbol(Mic1Symbol.label,new String(st.sval));
92    }
93  if ( t == StreamTokenizer.TT_EOL ) return new Symbol(Mic1Symbol.EOL) ;
94  if ( t == StreamTokenizer.TT_EOF ) return new Symbol(Mic1Symbol.EOF) ;
95  // System.out.println( "char = " + t ) ;
96  switch ( (char)t )
97    {
98    case ';': return new Symbol(Mic1Symbol.SEMI);
99    case '+': return new Symbol(Mic1Symbol.PLUS);
100   case '-': return new Symbol(Mic1Symbol.MINUS);
101   case '<': return new Symbol(Mic1Symbol.LESSTHAN);
102   case '=': return new Symbol(Mic1Symbol.EQUALS);
103   case '>': return new Symbol(Mic1Symbol.GREATERTHAN);
104   case '(': return new Symbol(Mic1Symbol.LPAREN);
105   case ')': return new Symbol(Mic1Symbol.RPAREN);
106   default :
107      System.out.println( st.lineno() + ": unexpected character " + t ) ;
108      break ;
109   }
110 return new Symbol(Mic1Symbol.error) ;
111 }
112
113 }
```

APPENDIX B

Ip-blocks

B.1 ipio.h

```
1  //-----------------------------------------------------------------
2  // Copyright (C) 2005 Esben Rugbjerg and Anders Markussen
3  // Tecnical University of Denmark
4  //
5  // This program is free software; you can redistribute it and/or
6  // modify it under the terms of the GNU Lesser General Public License
7  // as published by the Free Software Foundation; either version 2 of
8  // the License, or (at your option) any later version.
9  //
10 // This program is distributed in the hope that it will be useful, but
11 // WITHOUT ANY WARRANTY; without even the implied warranty of
12 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
13 // Lesser General Public License for more details.
14 //
15 // You should have received a copy of the GNU Lesser General Public
16 // License along with this program; if not, write to the Free Software
17 // Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
18 // USA
19 //
20 // $Id: ipdtu.h,v 1.1 2004/03/18 19:52:04 schaum Exp $
21 //-----------------------------------------------------------------
22 #ifndef IPIO_H
23 #define IPIO_H
24
25 #include "ipblock.h"
26
27 // ---------- Numerical Input Device ----------
28 class numpad : public aipblock {
29     int value;
30     public:
31     numpad(char *);
32     void setparm(char *_name);
33     void run();
34     bool checkterminal(int n, char *tname, aipblock::iodir d);
35     bool cannotSleepTest();
36     int getValue();
37 };
38
39 // ---------- LCD Output Device ----------
40 class lcd : public aipblock {
41     string line1;
42     string line2;
43     string display;
44     int data;
45     int write_addr;
46     int cursor_pos;
```

```
47      int lower_bound;
48      int bit0,
49          bit1,
50          bit2,
51          bit3,
52          bit4,
53          bit5,
54          bit6,
55          bit7,
56          bit8,
57          bit9;
58      bool id;
59      bool sh; // No shift is done if false
60      bool b; // Blinking cursor
61      bool c; // Cursor
62      bool d; // Display
63      bool dl; // Number of bits in input: true = 8 bit
64      bool f; // Font type:  5x8 = false
65      bool n; // Number of lines in display: true = 2
66      bool setFunction;
67      bool dispControlSet;
68      bool firstDisplayClear;
69      bool db0,
70           db1,
71           db2,
72           db3,
73           db4,
74           db5,
75           db6,
76           db7,
77           rw,
78           rs;
79      public:
80      lcd(char *);
81      void setparm(char *_name);
82      void run();
83      bool checkterminal(int n, char *tname, aipblock::iodir d);
84      bool cannotSleepTest();
85      void parseInput(int i);
86      void initDisplay();
87      void printDisplay();
88      void clearDisplay();
89      void returnHome();
90      void shiftDisplay(bool s, bool d);
91      void writeChar(char c, bool w, bool d);
92      void movePosInRam(int n, bool d);
93      void assignWriteAddr(int a);
94      void setEntryMode();
95      void dispOnOff();
96      void functionSet();
97      int readAddr();
98      char readData(bool d);
99      bool checkIni();
100 };
101
102 #endif
```

B.2 ipio.cxx

```
1  //-------------------------------------------------------------
2  // Copyright (C) 2005 Esben Rugbjerg and Anders Markussen
3  // Tecnical University of Denmark
4  //
5  // This program is free software; you can redistribute it and/or
6  // modify it under the terms of the GNU Lesser General Public License
7  // as published by the Free Software Foundation; either version 2 of
8  // the License, or (at your option) any later version.
9  //
10 // This program is distributed in the hope that it will be useful, but
11 // WITHOUT ANY WARRANTY; without even the implied warranty of
12 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
13 // Lesser General Public License for more details.
14 //
15 // You should have received a copy of the GNU Lesser General Public
16 // License along with this program; if not, write to the Free Software
17 // Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
18 // USA
19 //
20 // $Id: ipio.cxx,v 1.6 2004/09/16 19:17:23 schaum Exp $
21 //-------------------------------------------------------------
```

```
22
23
24  #include "ipio.h"
25  #include <errno.h>
26  #include <stdio.h>
27  #include <unistd.h>
28  #include <stdlib.h>
29  #include <cstring>
30
31  // ---------- Numerical Input Device ----------
32  numpad::numpad(char *inst)
33      : aipblock(inst), value(0) {}
34
35  void numpad::setparm(char *_name) {
36      printf("Error: numpad ipblock does not take any parameters");
37  }
38
39  bool numpad::checkterminal(int n, char *tname, aipblock::iodir dir) {
40      switch (n) {
41      case 0:
42          return (isinput(dir) && isname(tname, "in_addr"));
43          break;
44      case 1:
45          return (isinput(dir) && isname(tname, "in_data"));
46          break;
47      case 2:
48          return (isinput(dir) && isname(tname, "read"));
49          break;
50      case 3:
51          return (isoutput(dir) && isname(tname, "out_addr"));
52          break;
53      case 4:
54          return (isoutput(dir) && isname(tname, "out_data"));
55          break;
56      case 5:
57          return (isoutput(dir) && isname(tname, "ioa_ld"));
58          break;
59      case 6:
60          return (isoutput(dir) && isname(tname, "iod_ld"));
61          break;
62      }
63      return false;
64  }
65
66  bool numpad::cannotSleepTest() {
67      return false; // numpad can always cause 'sleep' mode
68  }
69
70  void numpad::run() {
71      //cout << "Test: " << (unsigned) ioval[0]->toulong() << "\n";
72      if (ioval[0]->toulong() != 0) {
73      ioval[3]->assignulong(0);
74      ioval[4]->assignulong(getValue());
75      ioval[5]->assignulong(1);
76      ioval[6]->assignulong(1);
77      }
78      else {
79      ioval[4]->assignulong(0);
80      ioval[6]->assignulong(0);
81      ioval[6]->assignulong(0);
82      }
83  }
84
85  int numpad::getValue() {
86      cout << "Enter decimal value (from 0 - 9999): ";
87      cin >> value;
88      if (value < 10000 && value >= 0) {
89      return value;
90      }
91      else {
92      cout << value << " is not between 0 and 9999.\n";
93      return getValue();
94      }
95  }
96
97  // ---------- LCD Output Device ----------
98  lcd::lcd(char *inst) : aipblock(inst), id(true), sh(false), b(false),
99                 c(false), d(false), dl(false), f(true), n(false),
100                lower_bound(0), write_addr(0), bit0(1), bit1(2),
101                bit2(4), bit3(8), bit4(16), bit5(32), bit6(64),
102                bit7(128), bit8(256), bit9(512), setFunction(false),
103                dispControlSet(false), firstDisplayClear(false) {
104 initDisplay();
105 }
```

```
106
107 void lcd::setparm(char *_name) {
108     printf("Error: lcd ipblock does not take any parameters");
109 }
110
111 bool lcd::checkterminal(int n, char *tname, aipblock::iodir dir) {
112     switch (n) {
113     case 0:
114         return (isinput(dir) && isname(tname, "in_addr"));
115         break;
116     case 1:
117         return (isinput(dir) && isname(tname, "in_data"));
118         break;
119     case 2:
120         return (isinput(dir) && isname(tname, "read"));
121         break;
122     case 3:
123         return (isoutput(dir) && isname(tname, "out_addr"));
124         break;
125     case 4:
126         return (isoutput(dir) && isname(tname, "out_data"));
127         break;
128     case 5:
129         return (isoutput(dir) && isname(tname, "ioa_ld"));
130         break;
131     case 6:
132         return (isoutput(dir) && isname(tname, "iod_ld"));
133         break;
134     }
135     return false;
136 }
137
138 bool lcd::cannotSleepTest() {
139     return false; // lcd can always cause 'sleep' mode
140 }
141
142 void lcd::run() {
143     if (ioval[0]->toulong() != 0) {
144     parseInput(ioval[1]->toulong());
145
146     data = ioval[1]->toulong();
147
148     printf("data: %d\n",data);
149     printf("Bin data: %d%d %d%d%d%d%d%d%d%d\n", rs, rw, db7,
150         db6, db5, db4, db3, db2, db1, db0);
151
152     if (rs) {
153         if (rw) {
154         ioval[4]->assignulong(readData(id));
155         }
156         else {
157         writeChar((char) ((data << 24) >> 24), true, !id);
158         }
159     }
160     else if (!rs && rw){
161         ioval[4]->assignulong(readAddr());
162     }
163     else if (!rs && !rw && db7) {
164         assignWriteAddr(((unsigned int) data << 25) >> 25);
165     }
166     else if (!rs && !rw && !db7 && !db6 && db5) {
167         functionSet();
168     }
169     else if (!rs && !rw && !db7 && !db6 && !db5 && db4) {
170         shiftDisplay(db3, !db2);
171         writeChar(' ', false, !db2);
172     }
173     else {
174         switch(data) {
175         case 1:
176             clearDisplay();
177             break;
178         case 3:
179             returnHome();
180             break;
181         case 7:
182             setEntryMode();
183             break;
184         case 15:
185             dispOnOff();
186             break;
187         default:
188             cout << "In data not cought by ip-block\n";
189             break;
```

```
190         }
191     }
192     ioval[3]->assignulong(0);
193     ioval[5]->assignulong(1);
194     ioval[6]->assignulong(1);
195     }
196     else {
197     ioval[4]->assignulong(0);
198     ioval[6]->assignulong(0);
199     ioval[6]->assignulong(0);
200     }
201     printDisplay();
202 }
203
204 void lcd::parseInput(int i) {
205     db0 = (bool) ((i & bit0) == bit0);
206     db1 = (bool) ((i & bit1) == bit1);
207     db2 = (bool) ((i & bit2) == bit2);
208     db3 = (bool) ((i & bit3) == bit3);
209     db4 = (bool) ((i & bit4) == bit4);
210     db5 = (bool) ((i & bit5) == bit5);
211     db6 = (bool) ((i & bit6) == bit6);
212     db7 = (bool) ((i & bit7) == bit7);
213     rw = (bool) ((i & bit8) == bit8);
214     rs = (bool) ((i & bit9) == bit9);
215 }
216
217 void lcd::initDisplay() {
218     display = "B              E                        ";
219     line1 = "                                         ";
220     line2 = "                                         ";
221 }
222
223 void lcd::printDisplay() {
224     cout << display << "\n";
225     cout << line1 << "\n";
226     cout << line2 << "\n";
227 }
228
229 void lcd::clearDisplay() {
230     initDisplay();
231     write_addr = 0;
232     lower_bound = 0;
233     id = true;
234     cout << "Instruction: Clear Display\n";
235 }
236
237 void lcd::returnHome() {
238     display = "B              E                        ";
239     write_addr = 0;
240     lower_bound = 0;
241     cout << "Instruction: Return Home\n";
242 }
243
244 void lcd::shiftDisplay(bool shift, bool direction) {
245     display[lower_bound] = ' ';
246     display[(lower_bound + 15) % 40] = ' ';
247     if (shift) {
248     if (direction) {
249         if (lower_bound == 0) {
250         lower_bound = 39;
251         }
252         else {
253         lower_bound--;
254         }
255         display[lower_bound] = 'B';
256         display[(lower_bound + 15) % 40] = 'E';
257     }
258     else {
259         if (lower_bound == 39) {
260         lower_bound = 0;
261         }
262         else {
263         lower_bound++;
264         }
265         display[lower_bound] = 'B';
266         display[(lower_bound + 15) % 40] = 'E';
267     }
268     }
269 }
270
271 void lcd::writeChar(char c, bool doWrite, bool direction) {
272     printf("Direction: %d\n",direction);
273     printf("write_addr before: %d\n",write_addr);
```

```
274         if (doWrite) {
275         if (write_addr >= 0 && write_addr <= 39) {
276             line1[write_addr] = c;
277         }
278         if (write_addr >= 64 && write_addr <= 103) {
279             line2[write_addr - 64] = c;
280         }
281         }
282         if (direction) {
283         write_addr--;
284         if (write_addr == 40) {
285             write_addr = 64;
286         }
287         if (write_addr == 104) {
288             write_addr = 0;
289         }
290         }
291         else {
292         write_addr++;
293         if (write_addr == -1) {
294             write_addr = 103;
295         }
296         if (write_addr == 63) {
297             write_addr = 39;
298         }
299         }
300         printf("write_addr after: %d\n",write_addr);
301     }
302
303     void lcd::assignWriteAddr(int a) {
304         printf("Address: %d\n", a);
305         write_addr = a;
306         cout << "Instruction: Assign Address\n";
307     }
308
309     int lcd::readAddr() {
310         cout << "Instruction: Read Address\n";
311         return write_addr;
312     }
313
314     char lcd::readData(bool direction) {
315         int data = 0;
316         if (write_addr >= 0 && write_addr <= 39) {
317         data = line1[write_addr];
318         }
319         if (write_addr >= 64 && write_addr <= 103) {
320         data = line2[write_addr - 64];
321         }
322         if (direction) {
323         write_addr--;
324         if (write_addr == 40) {
325             write_addr = 64;
326         }
327         if (write_addr == 104) {
328             write_addr = 0;
329         }
330         }
331         else {
332         write_addr++;
333         if (write_addr == -1) {
334             write_addr = 103;
335         }
336         if (write_addr == 63) {
337             write_addr = 39;
338         }
339         }
340         cout << "Instruction: Read Data\n";
341         return data;
342     }
343
344     void lcd::setEntryMode() {
345         if (checkIni()) {
346         id = db1;
347         sh = db0;
348         }
349         cout << "Instruction: Set Entry Mode\n";
350     }
351
352     void lcd::dispOnOff() {
353         if (setFunction) {
354         d = db2;
355         c = db1;
356         b = db0;
357         dispControlSet = true;
```

```
358         }
359         cout << "Instruction: Display on/off\n";
360     }
361
362     void lcd::functionSet() {
363         if (db4) dl = true;
364         else dl = false;
365         if (db3) n = true;
366         else n = false;
367         if (db2) f = true;
368         else f = false;
369         setFunction = (dl && n && !f);
370         cout << "Instruction: Set Function\n";
371     }
372
373     bool lcd::checkIni() {
374         return (setFunction && dispControlSet && firstDisplayClear && d);
375     }
```

APPENDIX C

Mic-1IO simulator

C.1 mic1sim.java

```
1  /*
2  *
3  *  mic1sim.java
4  *
5  *  mic1 microarchitecture simulator
6  *  Copyright (C) 1999, Prentice-Hall, Inc.
7  *
8  *  This program is free software; you can redistribute it and/or modify
9  *  it under the terms of the GNU General Public License as published by
10 *  the Free Software Foundation; either version 2 of the License, or
11 *  (at your option) any later version.
12 *
13 *  This program is distributed in the hope that it will be useful, but
14 *  WITHOUT ANY WARRANTY; without even the implied warranty of
15 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
16 *  Public License for more details.
17 *
18 *  You should have received a copy of the GNU General Public License along with
19 *  this program; if not, write to:
20 *
21 *    Free Software Foundation, Inc.
22 *    59 Temple Place - Suite 330
23 *    Boston, MA 02111-1307, USA.
24 *
25 *  A copy of the GPL is available online the GNU web site:
26 *
27 *    http://www.gnu.org/copyleft/gpl.html
28 *
29 */
30
31 import java.awt.*;
32 import java.io.*;
33 import java.util.*;
34
35
36
37 /**
38  * Main component of the mic1 simulator.  All components are
39  * created in this class, and displayed in the mic1sim frame.
40  * mic1sim also handles all keyboard and mouse events, loads
41  * micro and macro programs, and coordinates the sequence of
42  * activities for each clock cycle.
43  *
44  * @author
45  *   Dan Stone (<a href="mailto:dans@ontko.com"><i>dans@ontko.com</i></a>),
46  *   Ray Ontko & Co,
```

```
47   *   Richmond, Indiana, US
48   */
49  public class mic1sim extends Frame implements Mic1Constants {
50
51      public static TextArea stdout = new TextArea(5, 50);
52      public static boolean debug = false;
53      public static boolean run = false;
54      public static boolean halt = false;
55      public static Vector key_buffer = new Vector();
56
57      DebugFrame debug_frame = null;
58      MenuBar menubar = null;
59      Menu file = null;
60      GridBagLayout gridbag = new GridBagLayout();
61      RunThread run_thread = null;
62
63      private Button reset_button = null;
64      private Button run_button = null;
65      private Button stop_button = null;
66      private Button step_button = null;
67      private Label mic1_status = null;
68      private Label ijvm_status = null;
69      private TextField next_micro = null;
70
71      private int cycle_count;
72      private boolean null_array[] = {false, false, false, false, false, false};
73      private boolean true_array[] = {true};
74      private boolean mp_loaded = false;
75
76      // Components
77      private ControlStore control_store = null;
78      private MainMemory main_memory = null;
79      private ALU alu = null;
80      private Shifter shifter = null;
81      private Decoder decoder = null;
82      private O o = null;
83      private HighBit high_bit = null;
84      private Mic1LCDDisplay lcd = null; //Mic1IO Lcd display peripheral
85      private Mic1NumPad keyboard = null; //Mic1IO Keyboard peripheral
86      private Terminator terminator = null; //Mic1IO terminator of I/O bus
87
88
89      // Registers
90      public Register
91          sp = null,
92          lv = null,
93          cpp = null,
94          tos = null,
95          opc = null,
96          h = null;
97      private MAR mar = null;
98      private PC pc = null;
99      private MDR mdr = null;
100     private MBR mbr = null;
101     private MPC mpc = null;
102     private MIR mir = null;
103     private IO_reg ioa = null; //Mic1IO reg storing address for I/O operations
104     private IO_reg iod = null; //Mic1IO reg storing data for I/O operations
105
106     // Busses
107     private Bus
108         byte_address_bus = null,
109         word_address_bus = null,
110         byte_data_bus = null,
111         word_data_bus = null,
112         a_bus = null,
113         b_bus = null,
114         c_bus = null,
115         alu_bus = null;
116     private ObjectBus mir_bus = null;
117
118     // Component control lines
119     private IntControlLine
120         control_store_cl = null,
121         decoder_cl = null,
122         o_addr_cl = null,  // Carries next microinstruction address to O
123         o_mbr_cl = null,   // Carries MBR value to O
124         mpc_cl = null;     // Carries O value to MPC
125
126     private IntControlLine //Mic1IO
127         io_addr_out = null,
128         io_addr_in = null,
129         io_data_in = null,
130         io_data_out = null,
```

```
131         keyb_data_term = null, //Keyb
132         term_data_keyb = null, //Keyb
133         keyb_addr_term = null, //Keyb
134         term_addr_keyb = null, //Keyb
135         lcd_data_keyb = null, //LCD
136         keyb_data_lcd = null, //LCD
137         lcd_addr_keyb = null, //LCD
138         keyb_addr_lcd = null; //LCD
139
140
141
142     private ControlLine
143         memory_cl = null,
144         alu_cl = null,
145         shifter_cl = null,
146         o_jmpc_cl = null,    // Carries JMPC from MIR to O
147         jam_cl = null,       // Carries JAMN/JAMZ from MIR to High bit
148         high_bit_cl = null,  // Carries bit from High bit to MPC
149         n_cl = null,         // Negative flag from ALU to High bit
150         z_cl = null;         // Zero flag from ALU to High bit
151
152     private ControlLine //Mic1IO
153         ioa_ld_cl = null,
154         iod_ld_cl = null,
155         io_rd_cl = null,
156         term_iod_ld_keyb = null, //Keyb
157         term_ioa_ld_keyb = null, //Keyb
158         keyb_iod_ld_lcd = null, //LCD
159         keyb_ioa_ld_lcd = null; //LCD
160
161     // Register control lines
162     //   Each register has 2 control lines, one to indicate whether to store the value
163     //   on the IN bus, the other to indicate whether to put its value onto the OUT
164     //   bus.  Two control lines are used instead of one because the control signals
165     //   come from two different sources--STORE from the MIR, and PUT from the decoder--
166     //   and the two operations occur at different times in the clock cycle
167     private ControlLine
168         mar_store_cl = null,
169         mar_put_cl = null,
170         mdr_store_cl = null,
171         mdr_put_cl = null,
172         mdr_mem_cl = null,
173         pc_store_cl = null,
174         pc_put_cl = null,
175         mbr_store_cl = null,
176         mbr_put_cl = null,  // MBR stores only on a memory fetch, so it does not have a
177         mbru_put_cl = null, // store control line.  It has 2 put control lines, signed
178                 // and unsigned
179         sp_store_cl = null,
180         sp_put_cl = null,
181         lv_store_cl = null,
182         lv_put_cl = null,
183         cpp_store_cl = null,
184         cpp_put_cl = null,
185         tos_store_cl = null,
186         tos_put_cl = null,
187         opc_store_cl = null,
188         opc_put_cl = null,
189         h_store_cl = null,
190         h_put_cl = null;  // In the dual-bus data path, this control line will always
191               // be asserted
192
193     private ControlLine //Mic1IO
194         ioa_store_cl = null,
195         ioa_put_cl = null,
196         iod_store_cl = null,
197         iod_put_cl = null;
198
199
200         //*************************************************************************
201         /**   Constructor.  Creates all components, busses, and control lines
202          */
203         public mic1sim() {
204         super("Mic-1 Simulator");
205         init();
206     }
207
208     public mic1sim(String filename) {
209         super("Mic-1 Simulator");
210         init();
211         loadMicroprogram(filename);
212     }
213
214     public mic1sim(String file1, String file2) {
```

```
215         super("Mic-1 Simulator");
216         init();
217         loadMicroprogram(file1);
218         loadProgram(file2);
219     }
220
221     private void init() {
222         stdout.setFont(new Font("Courier",Font.PLAIN,12));
223         menubar = new MenuBar();
224         setMenuBar(menubar);
225         file = new Menu("File");
226         file.add(new MenuItem("Load Microprogram"));
227         file.add(new MenuItem("Load Macroprogram"));
228         file.add(new MenuItem("Open Debug Window"));
229         file.add(new MenuItem("Exit"));
230         menubar.add(file);
231         next_micro = new TextField();
232         next_micro.setEditable(false);
233
234         // First, create busses and control lines
235
236         // Busses
237         byte_address_bus = new Bus();
238         word_address_bus = new Bus();
239         byte_data_bus = new Bus();
240         word_data_bus = new Bus();
241         a_bus = new Bus();
242         b_bus = new Bus();
243         c_bus = new Bus();
244         alu_bus = new Bus();
245         mir_bus = new ObjectBus();
246
247         // Control lines
248         control_store_cl = new IntControlLine();
249         memory_cl = new ControlLine();
250         alu_cl = new ControlLine();
251         shifter_cl = new ControlLine();
252         decoder_cl = new IntControlLine();
253         o_mbr_cl = new IntControlLine();
254         o_addr_cl = new IntControlLine();
255         o_jmpc_cl = new ControlLine();
256         mpc_cl = new IntControlLine();
257         jam_cl = new ControlLine();
258         high_bit_cl = new ControlLine();
259         n_cl = new ControlLine();
260         z_cl = new ControlLine();
261         ioa_ld_cl = new ControlLine(); //Mic1IO
262         iod_ld_cl = new ControlLine(); //Mic1IO
263         io_rd_cl = new ControlLine(); //Mic1IO
264         io_addr_in = new IntControlLine(); //Mic1IO
265         io_addr_out = new IntControlLine(); //Mic1IO
266         io_data_in = new IntControlLine(); //Mic1IO
267         io_data_out = new IntControlLine(); //Mic1IO
268         keyb_data_term = new IntControlLine(); //Keyb
269         term_data_keyb = new IntControlLine(); //Keyb
270         keyb_addr_term = new IntControlLine(); //Keyb
271         term_addr_keyb = new IntControlLine(); //Keyb
272
273         lcd_data_keyb = new IntControlLine(); //Lcd
274         keyb_data_lcd = new IntControlLine(); //Lcd
275         lcd_addr_keyb = new IntControlLine(); //Lcd
276         keyb_addr_lcd = new IntControlLine(); //Lcd
277
278         mar_store_cl = new ControlLine();
279         mar_put_cl = new ControlLine();
280         mdr_store_cl = new ControlLine();
281         mdr_put_cl = new ControlLine();
282         mdr_mem_cl = new ControlLine();
283         pc_store_cl = new ControlLine();
284         pc_put_cl = new ControlLine();
285         mbr_store_cl = new ControlLine();
286         mbr_put_cl = new ControlLine();
287         mbru_put_cl = new ControlLine();
288         sp_store_cl = new ControlLine();
289         sp_put_cl = new ControlLine();
290         lv_store_cl = new ControlLine();
291         lv_put_cl = new ControlLine();
292         cpp_store_cl = new ControlLine();
293         cpp_put_cl = new ControlLine();
294         tos_store_cl = new ControlLine();
295         tos_put_cl = new ControlLine();
296         opc_store_cl = new ControlLine();
297         opc_put_cl = new ControlLine();
298         h_store_cl = new ControlLine();
```

```
299         h_put_cl = new ControlLine();
300         h_put_cl.setValue(true_array);
301
302         ioa_store_cl = new ControlLine(); //Mic1IO
303         ioa_put_cl = new ControlLine(); //Mic1IO
304         iod_store_cl = new ControlLine(); //Mic1IO
305         iod_put_cl = new ControlLine(); //Mic1IO
306         term_iod_ld_keyb = new ControlLine(); //keyb
307         term_ioa_ld_keyb = new ControlLine(); //keyb
308         keyb_iod_ld_lcd = new ControlLine(); //Lcd
309         keyb_ioa_ld_lcd = new ControlLine(); //Lcd
310
311
312
313         // Next, create components and registers
314
315         // Registers
316         mar = new MAR(c_bus, word_address_bus, mar_store_cl, memory_cl);
317         sp = new Register(c_bus, b_bus, "SP", sp_store_cl, sp_put_cl);
318         lv = new Register(c_bus, b_bus, "LV", lv_store_cl, lv_put_cl);
319         cpp = new Register(c_bus, b_bus, "CPP", cpp_store_cl, cpp_put_cl);
320
321         tos = new Register(c_bus, b_bus, "TOS", tos_store_cl, tos_put_cl);
322         opc = new Register(c_bus, b_bus, "OPC", opc_store_cl, opc_put_cl);
323         h = new Register(c_bus, a_bus, "H", h_store_cl, h_put_cl);
324         pc = new PC(c_bus, b_bus, byte_address_bus, pc_store_cl, pc_put_cl, memory_cl);
325         mdr = new MDR(c_bus, b_bus, word_data_bus, mdr_store_cl, mdr_put_cl, mdr_mem_cl,
326                       memory_cl);
327         mbr = new MBR(byte_data_bus, b_bus, mbr_store_cl, mbr_put_cl, mbru_put_cl,
328                       o_mbr_cl);
329         mir = new MIR(o_addr_cl, o_jmpc_cl, jam_cl, shifter_cl, alu_cl, mar_store_cl,
330                       mdr_store_cl, pc_store_cl, sp_store_cl, lv_store_cl, cpp_store_cl,
331                       tos_store_cl, opc_store_cl, h_store_cl, iod_store_cl, ioa_store_cl,
332                       io_rd_cl, memory_cl, decoder_cl, mir_bus);
333         ioa = new IO_reg(c_bus, b_bus, io_addr_in, io_addr_out, ioa_store_cl, ioa_put_cl,
334                          ioa_ld_cl, "IOA");  //Mic1IO
335         iod = new IO_reg(c_bus, b_bus, io_data_in, io_data_out, iod_store_cl, iod_put_cl,
336                          iod_ld_cl, "IOD"); //Mic1IO
337
338         // Components
339         control_store = new ControlStore(control_store_cl, mir_bus);
340         mpc = new MPC(mpc_cl, high_bit_cl, control_store_cl, control_store, next_micro);
341         main_memory = new MainMemory(byte_address_bus, word_address_bus,
342                                      byte_data_bus, word_data_bus, memory_cl,
343                                      mbr_store_cl, mdr_mem_cl, mdr);
344         alu = new ALU(a_bus, b_bus, alu_bus, alu_cl, n_cl, z_cl);
345         shifter = new Shifter(alu_bus, c_bus, shifter_cl);
346         decoder = new Decoder(decoder_cl, mdr_put_cl, pc_put_cl, mbr_put_cl, mbru_put_cl,
347                               sp_put_cl, lv_put_cl, cpp_put_cl, tos_put_cl, opc_put_cl,
348                               iod_put_cl, ioa_put_cl);
349         o = new O(o_addr_cl, o_mbr_cl, o_jmpc_cl, mpc_cl);
350         high_bit = new HighBit(n_cl, z_cl, jam_cl, high_bit_cl);
351
352         lcd = new Mic1LCDDisplay(io_data_out, io_data_in, io_addr_out, io_addr_in,
353                                  iod_ld_cl, ioa_ld_cl, keyb_data_lcd, lcd_data_keyb,
354                                  keyb_addr_lcd, lcd_addr_keyb, keyb_iod_ld_lcd,
355                                  keyb_ioa_ld_lcd, io_rd_cl, 1); //Lcd
356
357         keyboard = new Mic1NumPad(lcd_data_keyb, keyb_data_lcd, lcd_addr_keyb,
358                                   keyb_addr_lcd, keyb_iod_ld_lcd, keyb_ioa_ld_lcd,
359                                   term_data_keyb, keyb_data_term, term_addr_keyb,
360                                   keyb_addr_term, term_iod_ld_keyb, term_ioa_ld_keyb,
361                                   io_rd_cl, 2); //Mic1IO
362
363         terminator = new Terminator(keyb_data_term, term_data_keyb, keyb_addr_term,
364                                     term_addr_keyb, term_iod_ld_keyb, term_ioa_ld_keyb); //Mic1IO
365
366         initializeFrame();
367
368     }
369
370     //*****************************************************************************************
371
372     private void clearControlLines() {
373         mbr_store_cl.setValue(null_array);
374         mdr_mem_cl.setValue(null_array);
375         memory_cl.setValue(null_array);
376         alu_cl.setValue(null_array);
377         shifter_cl.setValue(null_array);
378         o_jmpc_cl.setValue(null_array);
379         jam_cl.setValue(null_array);
380         high_bit_cl.setValue(null_array);
381         n_cl.setValue(null_array);
382         z_cl.setValue(null_array);
```

```
383         mar_store_cl.setValue(null_array);
384         mar_put_cl.setValue(null_array);
385         mdr_store_cl.setValue(null_array);
386         mdr_put_cl.setValue(null_array);
387         pc_store_cl.setValue(null_array);
388         pc_put_cl.setValue(null_array);
389         mbr_put_cl.setValue(null_array);
390         mbru_put_cl.setValue(null_array);
391         sp_store_cl.setValue(null_array);
392         sp_put_cl.setValue(null_array);
393         lv_store_cl.setValue(null_array);
394         lv_put_cl.setValue(null_array);
395         cpp_store_cl.setValue(null_array);
396         cpp_put_cl.setValue(null_array);
397         tos_store_cl.setValue(null_array);
398         tos_put_cl.setValue(null_array);
399         opc_store_cl.setValue(null_array);
400         opc_put_cl.setValue(null_array);
401         h_store_cl.setValue(null_array);
402
403         ioa_ld_cl.setValue(null_array); //Mic1IO
404         iod_ld_cl.setValue(null_array); //Mic1IO
405         ioa_store_cl.setValue(null_array); //Mic1IO
406         ioa_put_cl.setValue(null_array); //Mic1IO
407         iod_store_cl.setValue(null_array); //Mic1IO
408         iod_put_cl.setValue(null_array); //Mic1IO
409         io_addr_in.setValue(0); //Mic1IO
410         io_addr_out.setValue(0); //Mic1IO
411         io_data_in.setValue(0); //Mic1IO
412         io_data_out.setValue(0); //Mic1IO
413         io_rd_cl.setValue(null_array); //Mic1IO
414
415         keyb_data_term.setValue(0); //keyb
416         term_data_keyb.setValue(0); //keyb
417         keyb_addr_term.setValue(0); //keyb
418         term_addr_keyb.setValue(0); //keyb
419
420         lcd_data_keyb.setValue(0); //Lcd
421         keyb_data_lcd.setValue(0); //Lcd
422         lcd_addr_keyb.setValue(0); //Lcd
423         keyb_addr_lcd.setValue(0); //Lcd
424
425
426         term_iod_ld_keyb.setValue(null_array); //keyb
427         term_ioa_ld_keyb.setValue(null_array); //keyb
428
429         keyb_iod_ld_lcd.setValue(null_array); //Lcd
430         keyb_ioa_ld_lcd.setValue(null_array); //Lcd
431
432
433     }
434
435
436     //*********************************************************************************************
437
438
439     public void run() {
440         run = true;
441         halt = false;
442         run_button.disable();
443         step_button.disable();
444         reset_button.disable();
445         stop_button.enable();
446         run_thread = new RunThread(this);
447         run_thread.start();
448     }
449
450     public void stop() {
451         run = false;
452         if (halt)
453             run_button.disable();
454         else
455             run_button.enable();
456         step_button.enable();
457         reset_button.enable();
458         stop_button.disable();
459         refresh();
460     }
461
462     public void step() {
463         cycle();
464     }
465
466     public void reset() {
```

```
467         if (mp_loaded) {
468             halt = false;
469             if (debug)
470                 debug_frame.text.setText("");
471             stdout.setText("");
472             cycle_count = 0;
473             control_store_cl.setValue(0);
474             mir.reset();
475             clearControlLines();
476             mpc.forceValue(0);
477             pc.forceValue(-1);
478             mbr.forceValue(0);
479             mar.forceValue(0);
480             mdr.forceValue(0);
481             opc.forceValue(0);
482             tos.forceValue(0);
483             h.forceValue(0);
484             iod.forceValue(0); //Mic1IO
485             ioa.forceValue(0); //Mic1IO
486             sp.forceValue(SP);
487             lv.forceValue(LV);
488             cpp.forceValue(CPP);
489             key_buffer.removeAllElements();
490             run_button.enable();
491             step_button.enable();
492             reset_button.enable();
493             stop_button.disable();
494             main_memory.reset();
495 
496             control_store_cl.setValue(0);
497             decoder_cl.setValue(0);
498             o_mbr_cl.setValue(0);
499             o_addr_cl.setValue(0);
500             mpc_cl.setValue(0);
501 
502             byte_address_bus.setValue(0);
503             word_address_bus.setValue(0);
504             byte_data_bus.setValue(0);
505             word_data_bus.setValue(0);
506             a_bus.setValue(0);
507             b_bus.setValue(0);
508             c_bus.setValue(0);
509             alu_bus.setValue(0);
510 
511 
512         }
513         else {
514             run_button.disable();
515             step_button.disable();
516             reset_button.disable();
517             stop_button.disable();
518         }
519     }
520 
521     public void cycle() {
522         cycle_count++;
523         if (debug)
524             debug_frame.text.appendText("-----------------Start cycle "
525                 + cycle_count + "----------------------\n");
526         clearControlLines();
527         // Drive next microinstruction onto control lines
528         control_store.poke();
529         mir.poke();
530 
531         // Data path cycle
532         decoder.poke();
533         mdr.put();
534         pc.put();
535         mbr.put();
536         sp.put();
537         lv.put();
538         cpp.put();
539         tos.put();
540         opc.put();
541         h.put();
542 
543         ioa.put(); //Mic1IO
544         iod.put(); //Mic1IO
545 
546         lcd.poke(); //Lcd
547         keyboard.poke(); //Keyb
548         terminator.poke(); //Mic1IO
549         keyboard.poke(); //Keyb
550         lcd.poke(); //Lcd
```

```
551
552             alu.poke();
553             shifter.poke();
554
555             mar.store();
556             mdr.store();
557             pc.store();
558             sp.store();
559             lv.store();
560             cpp.store();
561             tos.store();
562             opc.store();
563             h.store();
564
565             ioa.store(); //Mic1IO
566             iod.store(); //Mic1IO
567
568             pc.mem();
569             mar.mem();
570             mdr.mem();
571             main_memory.poke();
572             mbr.store();
573
574             // Control path cycle
575             high_bit.poke();
576             o.poke();
577             mpc.poke();
578             if (!run)
579                 refresh();
580         }
581
582         public static void halt() {
583             run = false;
584             halt = true;
585             stdout.appendText("\nEnd of run.\n");
586         }
587
588         public void refresh() {
589             mdr.refresh();
590             pc.refresh();
591             mbr.refresh();
592             sp.refresh();
593             lv.refresh();
594             cpp.refresh();
595             tos.refresh();
596             opc.refresh();
597             h.refresh();
598             mar.refresh();
599             ioa.refresh(); //Mic1IO
600             iod.refresh(); //Mic1IO
601         }
602
603         public void setControlStore(Mic1Instruction instructions[]) {
604             control_store.setStore(instructions);
605         }
606
607         private void setMemory(byte[] bytes) {
608             main_memory.setMemory(bytes);
609         }
610
611         private void programDialog() {
612             FileDialog fd = new FileDialog(this, "Load Macroprogram", FileDialog.LOAD);
613             fd.setFile("*.ijvm");
614             fd.show();
615             // fd.paintAll(fd.getGraphics());
616             if (fd.getFile() != null) {
617                 loadProgram(fd.getDirectory() + fd.getFile());
618             }
619         }
620
621         public void loadProgram(String filename) {
622             ErrorDialog err = null;
623             try {
624                 IJVMLoader loader = new IJVMLoader(filename);
625                 if (loader.isValid() == null) {
626                     setMemory(loader.getProgram());
627                     reset();
628                 }
629                 else
630                     err = new ErrorDialog("Error loading program", loader.isValid());
631             }
632             catch (FileNotFoundException fnfe) {
633                 err = new ErrorDialog("Error opening program", "File not found: " + filename);
634             }
```

```
635         catch (IOException ioe) {
636             System.out.println("Error loading macroprogram");
637         }
638     }
639
640     private void microprogramDialog() {
641         FileDialog fd = new FileDialog(this, "Load Microprogram", FileDialog.LOAD);
642         fd.setFile("*.mic1");
643         fd.show();
644         // fd.paintAll(fd.getGraphics());
645         if (fd.getFile() != null) {
646             loadMicroprogram(fd.getDirectory() + fd.getFile());
647         }
648     }
649
650     public void loadMicroprogram(String filename) {
651         ErrorDialog err = null;
652         Mic1Instruction microprogram[] = new Mic1Instruction[612];
653         try {
654             InputStream in = new FileInputStream(filename);
655             boolean eof = false;
656             if (in.read() != mic1_magic1 ||
657                 in.read() != mic1_magic2 ||
658                 in.read() != mic1_magic3 ||
659                 in.read() != mic1_magic4) {
660                 err = new ErrorDialog("Error opening file",
661                                       "Error loading Microprogram: invalid file format: "
662                       + filename);
663                 mp_loaded = false;
664             }
665             else {
666                 int i = 0;
667                 while (!eof) {
668                     microprogram[i] = new Mic1Instruction();
669                     eof = (microprogram[i].read(in) == -1);
670                     i++;
671                 }
672                 setControlStore(microprogram);
673                 mp_loaded = true;
674             }
675         }
676         catch (FileNotFoundException fnfe) {
677             err = new ErrorDialog("Error opening file",
678                                   "Error loading Microprogram: file not found: "
679                   + filename);
680             mp_loaded = false;
681         }
682         catch (IOException ioe) {
683             err = new ErrorDialog("Error reading file",
684                                   "Error loading Microprogram: error reading file: "
685                   + filename);
686             mp_loaded = false;
687         }
688         reset();
689     }
690
691     public boolean handleEvent(Event event) {
692         switch (event.id) {
693         case Event.ACTION_EVENT :
694             if (event.target instanceof MenuItem) {
695                 if (((String)event.arg).equals("Load Microprogram"))
696                     microprogramDialog();
697                 if (((String)event.arg).equals("Load Macroprogram"))
698                     programDialog();
699                 if (((String)event.arg).equals("Open Debug Window")) {
700                     debug_frame = new DebugFrame();
701                     debug = true;
702                 }
703                 if (((String)event.arg).equals("Exit")) {
704                     System.exit(0);
705                 }
706             }
707             else if (event.target == run_button)
708                 run();
709             else if (event.target == stop_button)
710                 stop();
711             else if (event.target == step_button)
712                 step();
713             else if (event.target == reset_button)
714                 reset();
715             break;
716         case Event.KEY_ACTION:
717         case Event.KEY_PRESS :
718             key_buffer.addElement(new Character((char)event.key));
```

```
719             //System.out.println("Key pressed: " + (char)event.key);
720             break;
721         case Event.WINDOW_DESTROY:
722             System.exit(0);
723             break;
724         }
725         return true;
726     }
727
728     private void initializeFrame() {
729         setLayout(gridbag);
730         GridBagConstraints c = new GridBagConstraints();
731         c.fill = GridBagConstraints.BOTH;
732         c.insets = new Insets(2,5,1,5);
733
734         c.weightx = 1.0; c.weighty = 0.0;
735
736         c.gridx = 0; c.gridy = 0; c.gridwidth = 2; c.gridheight = 1;
737         c.gridwidth=2; constrain(new Label("Registers"),c);
738         c.gridwidth=1;
739         c.gridy=1; constrain(new Label("MAR"),c);
740         c.gridy=2; constrain(new Label("MDR"),c);
741         c.gridy=3; constrain(new Label("PC"),c);
742         c.gridy=4; constrain(new Label("MBR"),c);
743         c.gridy=5; constrain(new Label("SP"),c);
744         c.gridy=6; constrain(new Label("LV"),c);
745         c.gridy=7; constrain(new Label("CPP"),c);
746         c.gridy=8; constrain(new Label("TOS"),c);
747         c.gridy=9; constrain(new Label("OPC"),c);
748         c.gridy=10; constrain(new Label("H"),c);
749         c.gridy=11; constrain(new Label("IOD"),c);//Mic1IO
750         c.gridy=12; constrain(new Label("IOA"),c);//Mic1IO
751         c.gridx=1;
752         c.gridy=1; constrain(mar,c);
753         c.gridy=2; constrain(mdr,c);
754         c.gridy=3; constrain(pc,c);
755         c.gridy=4; constrain(mbr,c);
756         c.gridy=5; constrain(sp,c);
757         c.gridy=6; constrain(lv,c);
758         c.gridy=7; constrain(cpp,c);
759         c.gridy=8; constrain(tos,c);
760         c.gridy=9; constrain(opc,c);
761         c.gridy=10; constrain(h,c);
762         c.gridy=11; constrain(iod,c); //Mic1IO
763         c.gridy=12; constrain(ioa,c); //Mic1IO
764
765         c.gridx=2; c.gridy=0; c.gridwidth=2;
766         constrain(new Label("Components"),c);
767         c.gridwidth=1;
768         c.gridy=1; constrain(new Label("Microinstruction"),c);
769         c.gridy=2; constrain(new Label("NextMicroinstruction"),c);
770         c.gridy=3; constrain(new Label("MPC"),c);
771         c.gridy=4; constrain(new Label("ALU"),c);
772         c.gridwidth=2;
773         c.gridy=5; constrain(new Label("Standard out"),c);
774
775         c.gridwidth=1; c.gridx=3;
776         c.gridy=1; constrain(mir,c);
777         c.gridy=2; constrain(next_micro,c);
778         c.gridy=3; constrain(mpc,c);
779         c.gridy=4; constrain(alu,c);
780         c.gridwidth=2;
781         c.gridx=2;
782         c.gridheight=7;
783         c.gridy=6; constrain(stdout,c);
784
785         Panel button_panel = new Panel();
786         button_panel.setLayout(new GridLayout(1,4,5,5));
787
788         run_button = new Button("Run");
789         run_button.enable(false);
790         stop_button = new Button("Stop");
791         stop_button.enable(false);
792         step_button = new Button("Step");
793         step_button.enable(false);
794         reset_button = new Button("Reset");
795         reset_button.enable(false);
796
797         button_panel.add(run_button);
798         button_panel.add(stop_button);
799         button_panel.add(step_button);
800         button_panel.add(reset_button);
801
802         c.gridx = 0; c.gridy = 13; c.gridwidth = 5; c.gridheight = 1;
```

```
803         c.weightx = 1.0;
804         c.weighty = 0.0;
805         constrain(button_panel, c);
806
807         show();
808         resize(preferredSize());
809         paintAll(getGraphics());
810     }
811
812     private void constrain(Component component, GridBagConstraints constraints) {
813         ((GridBagLayout)getLayout()).setConstraints(component, constraints);
814         add(component);
815     }
816
817     public static void main(String args[]) {
818         mic1sim s = null;
819         stdout.setEditable(false);
820         if (args.length == 1)
821             s = new mic1sim(args[0]);
822         else if (args.length == 2)
823             s = new mic1sim(args[0], args[1]);
824         else
825             s = new mic1sim();
826     }
827 }
```

C.2 Terminator.java

```
1  /*********************************************************/
2  /* Author: Anders Markussen and Esben Rugbjerg           */
3  /* Date: 20050103                                        */
4  /* The 'Terminator' class implements the functionality of   */
5  /* the terminator in the I/O chain proposed in the Mic-1IO  */
6  /* architecture.                                         */
7  /*********************************************************/
8
9  import java.io.*;
10 import java.awt.*;
11
12 public class Terminator extends TextField {
13
14     private int
15         in_data = 0,
16         in_addr = 0;
17     private IntControlLine
18         data_in = null,
19         data_out= null,
20         addr_in = null,
21         addr_out= null;
22     private ControlLine
23         iod_ld = null,
24         ioa_ld = null;
25     //used to signal the load signal to the IOD and IOA registers.
26     private boolean true_array[] = {true};
27
28
29     public Terminator(IntControlLine data_in, IntControlLine data_out,
30                       IntControlLine addr_in, IntControlLine addr_out,
31                       ControlLine iod_ld, ControlLine ioa_ld) {
32
33         this.in_data = in_data;
34         this.in_addr = in_addr;
35         this.data_in = data_in;
36         this.data_out = data_out;
37         this.addr_in = addr_in;
38         this.addr_out = addr_out;
39         this.iod_ld = iod_ld;
40         this.ioa_ld = ioa_ld;
41     }
42
43     public void poke() {
44         //            System.out.println("terminator:" + addr_in.getValue());
45
46         if(addr_in.getValue() != 0) {
47
48             data_out.setValue(0xfeedbeef);
49             iod_ld.setValue(true_array);
50             ioa_ld.setValue(true_array);
51
52
```

```
53          }
54
55      }
56
57  }
```

C.3 IO_reg.java

```
1   /*********************************************************/
2   /* Author: Anders Markussen and Esben Rugbjerg           */
3   /*                                                       */
4   /* Purpose of this register is to contain the value      */
5   /* which should be written out on the I/O bus.           */
6   /* This register is an extension of the Mic-1 architechture.*/
7   /* The I/O bus is implemented in this simulator with     */
8   /* ControlLines and IntControlLines because approach is  */
9   /* closest to the posibilities in the Gezel HDL.         */
10  /*********************************************************/
11
12  import java.awt.*;
13
14  public class IO_reg extends TextField {
15
16      private int value;
17      private String name;
18      protected Bus
19          in = null,
20          out = null;
21      protected ControlLine
22          put_cl = null,
23          store_cl = null,
24          io_cl = null; /* controls whether the register reads from the bus */
25      protected IntControlLine
26          io_in = null,
27          io_out = null;
28
29      public IO_reg(Bus in, Bus out, IntControlLine io_in, IntControlLine io_out,
30          ControlLine store_cl,ControlLine put_cl, ControlLine io_cl,
31          String name) {
32
33          super(10); /*construct a textfield through */
34          this.in = in;
35          this.out = out;
36          this.put_cl = put_cl;
37          this.store_cl = store_cl;
38          this.io_cl = io_cl;
39          this.io_in = io_in;
40          this.io_out = io_out;
41          this.name = name;
42
43          setEditable(false);
44          refresh();
45      }
46
47
48      public void store() {
49          if (store_cl.getValue()[0]) {
50              value = in.getValue();
51              if(mic1sim.debug)
52                  DebugFrame.text.appendText(name + ": Store 0x" +
53              Integer.toHexString(value).toUpperCase()
54              + "\n");
55              if(!mic1sim.run)
56                  refresh();
57          }
58          if(io_cl.getValue()[0]) {
59              value = io_in.getValue();
60              if (mic1sim.debug)
61                  DebugFrame.text.appendText(name + ": Read 0x" +
62              Integer.toHexString(value).toUpperCase()
63              + "\n");
64              if(!mic1sim.run)
65                  refresh();
66          }
67      }
68
69      public void put() {
70          if (put_cl.getValue()[0]) {
71              System.out.println("Der puttes af " + name);
72              out.setValue(value);
```

```
73              if (mic1sim.debug)
74                  DebugFrame.text.appendText(name + ": Put 0x" +
75              Integer.toHexString(value).toUpperCase()
76              + "\n");
77
78              if (mic1sim.debug)
79                  DebugFrame.text.appendText(name + ": Put 0x" +
80              Integer.toHexString(value).toUpperCase()
81              + "on IO-bus \n");
82
83              if (!mic1sim.run)
84                 refresh();
85          }
86              io_out.setValue(value);
87      }
88
89      public void refresh() {
90          setText("0x" + Integer.toHexString(value).toUpperCase());
91      }
92
93      public void forceValue(int value) {
94          this.value = value;
95          refresh();
96      }
97  }
```

C.4 Mic1NumPad.java

```
1   /*********************************************************/
2   /* Author: Anders Markussen and Esben Rugbjerg           */
3   /* Date: 20050103                                        */
4   /*********************************************************/
5
6   import java.awt.*;
7   import java.io.*;
8   import java.util.*;
9   import ti.swing.*;
10  import javax.swing.*;
11
12  public class Mic1NumPad { //extends Frame {
13
14      private NumGrid numgrid;
15
16      //GridBagLayout gridbag = new GridBagLayout();
17
18      private IntControlLine
19          data_in_c = null,
20          data_in_t = null,
21          data_out_c = null,
22          data_out_t = null,
23          addr_in_c = null,
24          addr_in_t = null,
25          addr_out_c = null,
26          addr_out_t = null;
27
28      private ControlLine
29          iod_ld_c = null,
30          iod_ld_t = null,
31          ioa_ld_c = null,
32          ioa_ld_t = null,
33          io_rd = null;
34
35      private int
36          address = 2;
37
38      private boolean true_array[] = {true};
39
40      private boolean inCycle = false;
41
42      public Mic1NumPad(IntControlLine data_in_c, IntControlLine data_out_c,
43                        IntControlLine addr_in_c, IntControlLine addr_out_c,
44                        ControlLine iod_ld_c, ControlLine ioa_ld_c,
45                        IntControlLine data_in_t, IntControlLine data_out_t,
46                        IntControlLine addr_in_t, IntControlLine addr_out_t,
47                        ControlLine iod_ld_t, ControlLine ioa_ld_t,
48                        ControlLine io_rd, int address) {
49
50          this.data_in_c = data_in_c;
51          this.data_out_c = data_out_c;
52          this.addr_in_c = addr_in_c;
```

```
53          this.addr_out_c = addr_out_c;
54          this.iod_ld_c = iod_ld_c;
55          this.ioa_ld_c = ioa_ld_c;
56          this.data_in_t = data_in_t;
57          this.data_out_t = data_out_t;
58          this.addr_in_t = addr_in_t;
59          this.addr_out_t = addr_out_t;
60          this.iod_ld_t = iod_ld_t;
61          this.ioa_ld_t = ioa_ld_t;
62          this.io_rd = io_rd;
63
64          this.address = address;
65
66          numgrid = new NumGrid();
67      }
68
69      public void poke(){
70
71      if( addr_in_c.getValue() == address ) {
72              /* do what this device should do*/
73
74
75          if(inCycle)
76          inCycle = false;
77          else {
78          data_out_c.setValue(getInput());
79          addr_out_c.setValue(0);
80          iod_ld_c.setValue(true_array);
81          ioa_ld_c.setValue(true_array);
82          inCycle = true;
83          }
84          }
85          else {
86              /*forward what ever comes on the ports*/
87
88
89
90              data_out_t.setValue(data_in_c.getValue());
91              addr_out_t.setValue(addr_in_c.getValue());
92              data_out_c.setValue(data_in_t.getValue());
93              addr_out_c.setValue(addr_in_t.getValue());
94              iod_ld_c.setValue(iod_ld_t.getValue());
95              ioa_ld_c.setValue(ioa_ld_t.getValue());
96
97          }
98      }
99
100     private int getInput() {
101
102     return numgrid.getValue();
103
104     }
105 }
```

C.5 NumGrid.java

```
1   /*********************************************************/
2   /* Author: Anders Markussen and Esben Rugbjerg           */
3   /* Date: 20050103                                        */
4   /*********************************************************/
5
6   import java.awt.*;
7   import java.awt.event.*;
8   import java.lang.*;
9
10  public class NumGrid extends Thread implements ActionListener { //Panel {
11
12      private Frame frame = new Frame();
13
14      private int value = 0, digits = 0;
15
16      protected boolean number_commited = false;
17
18      private Button but0,
19             but1,
20             but2,
21             but3,
22             but4,
23             but5,
24             but6,
```

```
25              but7,
26              but8,
27              but9,
28              butC,
29              butE;
30
31      private Panel button_panel = new Panel();
32
33      private Label number = new Label("0",Label.CENTER);
34
35      private NumGrid parent_thread;
36
37      protected Semaphore sem = new Semaphore();
38
39      public NumGrid() {
40      super("NumPad");
41      init();
42      }
43
44      public NumGrid(NumGrid parent_thread) {
45      this.parent_thread = parent_thread;
46      }
47
48      public void run() {
49      while (!parent_thread.number_commited) {
50          try {
51          sleep(1);
52          }
53          catch (Exception e) {
54          System.out.println("Exception in NumPad");
55          }
56      }
57      parent_thread.sem.release();
58      }
59
60      public void init() {
61      frame.setLayout(new GridLayout(2,1));
62
63      but0 = new Button("0");
64      but0.setActionCommand("0");
65      but0.addActionListener(this);
66      but1 = new Button("1");
67      but1.setActionCommand("1");
68      but1.addActionListener(this);
69      but2 = new Button("2");
70      but2.setActionCommand("2");
71      but2.addActionListener(this);
72      but3 = new Button("3");
73      but3.setActionCommand("3");
74      but3.addActionListener(this);
75      but4 = new Button("4");
76      but4.setActionCommand("4");
77      but4.addActionListener(this);
78      but5 = new Button("5");
79      but5.setActionCommand("5");
80      but5.addActionListener(this);
81      but6 = new Button("6");
82      but6.setActionCommand("6");
83      but6.addActionListener(this);
84      but7 = new Button("7");
85      but7.setActionCommand("7");
86      but7.addActionListener(this);
87      but8 = new Button("8");
88      but8.setActionCommand("8");
89      but8.addActionListener(this);
90      but9 = new Button("9");
91      but9.setActionCommand("9");
92      but9.addActionListener(this);
93      butC = new Button("Clear");
94      butC.setActionCommand("Clear");
95      butC.addActionListener(this);
96      butE = new Button("Enter");
97      butE.setActionCommand("Enter");
98      butE.addActionListener(this);
99
100     button_panel.add(but7);
101     button_panel.add(but8);
102     button_panel.add(but9);
103     button_panel.add(but4);
104     button_panel.add(but5);
105     button_panel.add(but6);
106     button_panel.add(but1);
107     button_panel.add(but2);
108     button_panel.add(but3);
```

```
109         button_panel.add(but0);
110         button_panel.add(butC);
111         button_panel.add(butE);
112
113         button_panel.setLayout(new GridLayout(4,3));
114
115         frame.add(number,"North");
116         frame.add(button_panel,"South");
117
118         frame.show();
119         frame.resize(frame.preferredSize());
120         frame.paintAll(frame.getGraphics());
121         }
122
123         public int getValue() {
124         NumGrid child = new NumGrid(this);
125         child.start();
126         sem.acquire();
127         //while (!number_commited) {
128         //    try {
129         //    sleep(1);
130         //    }
131         //    catch (Exception e) {
132         //    System.out.println("Exception in NumPad");
133         //    }
134         //}
135         return value;
136         }
137
138         //public boolean checkReady() {
139         //return number_commited;
140         //}
141
142         public void actionPerformed(ActionEvent e) {
143         String button = e.getActionCommand();
144         if (button.equals("0")) addDigit(0,number);
145         else if (button.equals("1")) addDigit(1,number);
146         else if (button.equals("2")) addDigit(2,number);
147         else if (button.equals("3")) addDigit(3,number);
148         else if (button.equals("4")) addDigit(4,number);
149         else if (button.equals("5")) addDigit(5,number);
150         else if (button.equals("6")) addDigit(6,number);
151         else if (button.equals("7")) addDigit(7,number);
152         else if (button.equals("8")) addDigit(8,number);
153         else if (button.equals("9")) addDigit(9,number);
154         else if (button.equals("Enter")) number_commited = true;
155         else if (button.equals("Clear")) {
156             number_commited = false;
157             value = 0;
158             digits = 0;
159             number.setText("0");
160         }
161         }
162
163         private void addDigit(int digit, Label display) {
164         if (digits < 4) {
165             value = ((10 * value) + digit);
166             digits++;
167             display.setText(String.valueOf(value));
168         }
169         }
170
171
172 }
```

C.6 Mic1LCDDisplay

```
1  /*********************************************************/
2  /* Author: Anders Markussen and Esben Rugbjerg           */
3  /* Date: 20050103                                        */
4  /*********************************************************/
5
6  import java.awt.*;
7  import java.io.*;
8  import java.util.*;
9  import ti.swing.*;
10 import javax.swing.*;
11
12 public class Mic1LCDDisplay extends Frame {
13
```

```
14      private static LCDCharacterDisplay display = new LCDCharacterDisplay(2, 16,
15              Color.black,new Color(165,181,66),new Color(148,156,66) );
16
17      GridBagLayout gridbag = new GridBagLayout();
18
19      private IntControlLine
20          data_in_c = null,
21          data_in_t = null,
22          data_out_c = null,
23          data_out_t = null,
24          addr_in_c = null,
25          addr_in_t = null,
26          addr_out_c = null,
27          addr_out_t = null;
28
29      private ControlLine
30          iod_ld_c = null,
31          iod_ld_t = null,
32          ioa_ld_c = null,
33          ioa_ld_t = null,
34          io_rd = null;
35
36      private int
37          address = 0;
38
39      private boolean true_array[] = {true};
40
41      private boolean inCycle = false;
42
43  //      private LCDDriver driver = new LCDDriver(data_in_c, data_out_c, addr_in_c,
44  //                                               addr_out_c, io_rd);
45
46      private LCDDriver driver = new LCDDriver();
47
48      public Mic1LCDDisplay(IntControlLine data_in_c, IntControlLine data_out_c,
49                            IntControlLine addr_in_c, IntControlLine addr_out_c,
50                            ControlLine iod_ld_c, ControlLine ioa_ld_c,
51                            IntControlLine data_in_t, IntControlLine data_out_t,
52                            IntControlLine addr_in_t, IntControlLine addr_out_t,
53                            ControlLine iod_ld_t, ControlLine ioa_ld_t,
54                            ControlLine io_rd, int address ) {
55
56          super("LCD display");
57
58          this.data_in_c = data_in_c;
59          this.data_out_c = data_out_c;
60          this.addr_in_c = addr_in_c;
61          this.addr_out_c = addr_out_c;
62          this.iod_ld_c = iod_ld_c;
63          this.ioa_ld_c = ioa_ld_c;
64          this.data_in_t = data_in_t;
65          this.data_out_t = data_out_t;
66          this.addr_in_t = addr_in_t;
67          this.addr_out_t = addr_out_t;
68          this.iod_ld_t = iod_ld_t;
69          this.ioa_ld_t = ioa_ld_t;
70          this.io_rd = io_rd;
71
72          if(data_in_c == null)
73              System.out.println("på Mic1LCDDisplay niveau er data_in_c lig 'null'");
74          if(data_in_c instanceof IntControlLine)
75              System.out.println("på Mic1LCDDisplay niveau er data_in_ korrekt instantieret");
76
77
78          this.address = address;
79
80          init();
81
82
83      }
84
85      public void init() {
86          setLayout(gridbag);
87          GridBagConstraints c = new GridBagConstraints();
88          c.fill = GridBagConstraints.BOTH;
89          c.insets = new Insets(2,5,1,5);
90          c.weightx = 1.0; c.weighty = 0.0;
91
92          c.gridx = 0; c.gridy = 0; c.gridwidth = 1; c.gridheight = 2;
93          ((GridBagLayout)getLayout()).setConstraints(display,c);
94          add(display);
95          System.out.println("har tilføjet det hele");
96          c.gridy = 1;
97          show();
```

```
98          resize(preferredSize());
99          paintAll(getGraphics());
100
101         char[][] testText= {{' ',' ',' ',42,'M','i','c','1',' ','I','/','O',42,' ',' ',' '},
102             {' ','A','n','d','e','r','s',' ','&',' ','E','s','b','e','n',' '}};
103
104         display.updateScreen(testText);
105     }
106
107
108
109     public void poke(){
110 //         if(data_in_c instanceof IntControlLine)
111 //             System.out.println("på poke niveau er data_in_ korrekt instantieret");
112
113         if( addr_in_c.getValue() == address ) {
114             /* do what this device should do*/
115
116
117             if(inCycle)
118                 inCycle = false;
119             else {
120         display.updateScreen(driver.update(data_in_c, data_out_c, addr_in_c,
121                                            addr_out_c, io_rd));
122         iod_ld_c.setValue(true_array);
123         ioa_ld_c.setValue(true_array);
124         inCycle = true;
125             }
126
127         }
128         else {
129             /*forward what ever comes on the ports*/
130
131             data_out_t.setValue(data_in_c.getValue());
132             addr_out_t.setValue(addr_in_c.getValue());
133             data_out_c.setValue(data_in_t.getValue());
134             addr_out_c.setValue(addr_in_t.getValue());
135             iod_ld_c.setValue(iod_ld_t.getValue());
136             ioa_ld_c.setValue(ioa_ld_t.getValue());
137
138         }
139
140     }
141
142
143 }
144
```

C.7 LCDDriver.java

```
1   /*************************************************************/
2   /* Author: Anders Markussen and Esben Rugbjerg               */
3   /* Date: 20050103                                            */
4   /*************************************************************/
5
6   public class LCDDriver {
7
8
9
10      //Ports
11      public static IntControlLine
12          data_in = null,
13          data_out = null,
14          addr_in = null,
15          addr_out = null;
16
17
18      private ControlLine
19          io_rd = null;
20
21      //input signals
22      private boolean
23          db0 = false,
24          db1 = false,
25          db2 = false,
26          db3 = false,
27          db4 = false,
28          db5 = false,
29          db6 = false,
30          db7 = false,
```

```
31          rs = false,
32          rw = false,
33          e = false;
34
35      //status bits
36      private boolean
37          //entry mode increment control
38          //high/true: increment ram addr, moving cursor to the right
39          //low/false: decrement ram addr, moving cursor to the left
40          id = true,
41          //shift of entire display
42          //high/true: shifting of entire display is performed in each write operation to
43          //the DDram, according to the id-bit
44          //low/false: no shift is done.
45          sh = false,
46          d = false, //display on or off
47          c = false, // cursor on or off
48          b = false, // blinking cursor on or off
49          dl = false, // number of bits in input: true = 8 bit
50          n = false, // number of lines on display: true = 2 lines
51          f = true, //type of font: false for 5x8 font.
52          setFunction = false, //if true 'function set' step in initialization completed
53          dispControlSet = false, //true if 'display Control set' step of initialization completed
54          firstDisplayClear = false; //true after first 'display clear' instruction
55
56      private char[][] output = new char[2][16];
57
58      private int
59          ramAddr = 0, //addr in ddram
60          data = 0, //input data from data line - all 32 bit
61          charValue = 0, // value of first byte
62          addrValue = 0,  // value of first six bits.
63          outData = 0, // value that is returned on data line
64          outAddr = 0, // value that is returned on address line
65          cursorPos = 0, //position of cursor
66          offSet = 0; // offset of display from startposition (home = 0x0 and 0x40)
67
68
69
70      private final int
71          bit0 = 1,
72          bit1 = 2,
73          bit2 = 4,
74          bit3 = 8,
75          bit4 = 16,
76          bit5 = 32,
77          bit6 = 64,
78          bit7 = 128,
79          bit8 = 256,
80          bit9 = 512,
81          bit10 = 1024;
82
83
84      private char
85          ramData = 0x20;
86
87      private char[][] ddram = new char[2][40];
88
89
90  //      public LCDDriver(IntControlLine data_in, IntControlLine data_out,
91  //                       IntControlLine addr_in, IntControlLine addr_out,
92  //                       ControlLine io_rd) {
93
94
95  //          this.data_in = data_in;
96  //          this.data_out = data_out;
97  //          this.addr_in = addr_in;
98  //          this.addr_out = addr_out;
99  //          this.io_rd = io_rd;
100
101 //      }
102
103     public LCDDriver() {
104
105         ramData = 0x20;
106
107     }
108
109
110     //The externaly called method which performs the control of the display
111     public char[][] update(IntControlLine data_in, IntControlLine data_out,
112                            IntControlLine addr_in, IntControlLine addr_out,
113                            ControlLine io_rd) {
114
```

```
115         System.out.println("------------------- begyndelse på cycle -------------");
116 //         if(data_in instanceof IntControlLine) //debug
117 //             System.out.println("data_in er instantieret");
118 //         if(data_in == null)
119 //             System.out.println("data_in er stadig 'null'");
120          data = data_in.getValue();
121         //        addr = addr_in.getValue();
122          //         System.out.println("sh i begyndelsen af cycle: "+ sh);
123          System.out.println("data = " + data); //debug
124 
125         parseInput();
126 
127 
128         if(rs){
129             System.out.println("I 'rs' afsnit");
130             if(rw)
131                 readRam();
132             else{
133                 System.out.println("før writeRam() i rs-afsnit");
134                 writeRam();
135             }
136         }
137         else if(!rs && rw){
138             System.out.println("I '!rs && rw' afsnit");
139             returnBusyFlag();
140         }
141         else if(!rs && !rw && db7){
142             System.out.println("I '!rs && !rw && db7' afsnit");
143             setDDramAddress();
144         }
145         else if(!rs && !rw && !db7 && !db6 && db5){
146             System.out.println("I '!rs && !rw && !db7 && !db6 && db5' afsnit");
147             functionSet();
148         }
149         else if(!rs && !rw && !db7 && !db6 && !db5 && db4 ){
150             System.out.println("I '!rs && !rw && !db7 && !db6 && !db5 && db4' afsnit");
151             dispShift();
152         }
153         else{
154             switch(data)
155                 {
156 
157                 case 1: //First instruction
158                     clearDisp();
159                     break;
160                 case 2:
161                 case 3:
162                     returnCursor();
163                     break;
164                 case 4:
165                 case 5:
166                 case 6:
167                 case 7:
168                     System.out.println("før SetEntryMode");
169                     setEntryMode();
170                     break;
171                 case 8:
172                 case 9:
173                 case 10:
174                 case 11:
175                 case 12:
176                 case 13:
177                 case 14:
178                 case 15:
179                     dispOnOff();
180                     break;
181                 }
182         }
183 
184         generateOutput();
185         data_out.setValue(outData);
186         addr_out.setValue(outAddr);
187         System.out.println("------------------- Slut på cycle ----------------");
188         return output;
189     }
190 
191     private void parseInput(){
192 
193         db0 = (boolean)((data & bit0) == bit0);
194         //        System.out.println("db0 er:" + db0);
195         db1 = (boolean)((data & bit1) == bit1);
196         //        System.out.println("db1 er:" + db1);
197         db2 = (boolean)((data & bit2) == bit2);
198         //        System.out.println("db2 er:" + db2);
```

```
199         db3 = (boolean)((data & bit3) == bit3);
200         //        System.out.println("db3 er:" + db3);
201         db4 = (boolean)((data & bit4) == bit4);
202         //        System.out.println("db4 er:" + db4);
203         db5 = (boolean)((data & bit5) == bit5);
204         //        System.out.println("db5 er:" + db5);
205         db6 = (boolean)((data & bit6) == bit6);
206         //        System.out.println("db6 er:" + db6);
207         db7 = (boolean)((data & bit7) == bit7);
208         //        System.out.println("db7 er:" + db7);
209         rw = (boolean)((data & bit8) == bit8);
210         System.out.println("rw er:" + rw);
211         rs = (boolean)((data & bit9) == bit9);
212         System.out.println("rs er:" + rs);
213         e = (boolean)((data & bit10) == bit10);
214         System.out.println("e er:" + e);
215
216         charValue = data;
217         //        System.out.println("charValue initialt = " + charValue);
218         charValue = charValue << 24;
219         //        System.out.println("charValue eft V-shift = " + charValue);
220         charValue = charValue >>> 24;
221         //        System.out.println("charValue eft H-shift = " + charValue);
222         addrValue = ((data << 25) >>> 25);
223
224     }
225
226
227     /*======================*/
228     /*instructions          */
229     /*======================*/
230
231     //Clear Display
232     private void clearDisp(){
233         if(d) {
234             System.out.println("clearDisp()");
235             ramAddr = 0;
236
237             while(ramAddr <= 0x27){
238                 ddram[0][ramAddr] = ramData;
239                 ramAddr++;
240
241             }
242
243             ramAddr = 0;
244
245             while(ramAddr <= 0x27){
246                 ddram[1][ramAddr] = ramData;
247                 ramAddr++;
248
249             }
250             firstDisplayClear = true;
251             ramAddr = 0;
252             cursorPos = 0;
253             id = true;
254             System.out.println("id= "+ id);
255             offSet = 0;
256
257             outData = 0;
258             outAddr = 0;
259         }
260     }
261
262     //Return Home
263     private void returnCursor(){
264         System.out.println("returnCursor()");
265         if(checkIni()){
266             ramAddr = 0;
267             offSet = 0;
268             cursorPos = 0;
269         }
270
271         outData = 0;
272         outAddr = 0;
273
274     }
275
276     //Entry mode set
277     private void setEntryMode(){
278         System.out.println("setEntryMode()");
279         if(checkIni()) {
280             id = db1;
281             sh = db0;
282
```

```
283             System.out.println("id = "+id);
284             System.out.println("sh = "+sh);
285 
286             outData = 0;
287             outAddr = 0;
288         }
289     }
290 
291     //Display ON/OFF control
292     private void dispOnOff() {
293         System.out.println("dispOnOff()");
294         if(setFunction){
295             d = db2 ; //display on or off
296             c = db1 ; //cursor on or off
297             b = db0 ; //blink cursor on or off
298             dispControlSet = true;
299 
300             System.out.println("d er:" + d);
301             System.out.println("c er:" + c);
302             System.out.println("b er:" + b);
303         }
304 
305         outData = 0;
306         outAddr = 0;
307     }
308 
309     //Cursor or Display shift
310     private void dispShift() {
311         System.out.println("dispShift()");
312         if(checkIni()){
313             if(db3){
314                 offSet = db2 ? (offSet+1) : (offSet-1);
315             }
316             else{
317                 cursorPos = movePosInRam(cursorPos, db2);
318                 ramAddr = movePosInRam(ramAddr, db2);
319                 }
320         }
321         outData = 0;
322         outAddr = 0;
323     }
324 
325     //Function set
326     private void functionSet() {
327         System.out.println("functionSet()");
328         dl = db4 ? true : false ; // data length setting, correct = true
329         n = db3 ? true : false ; // number of lines, correct = true
330         f = db2 ? true : false ; // font selection, correct = false
331         setFunction = (dl & n & !f); //check if values is correct
332 
333             System.out.println("dl= "+ dl);
334             System.out.println("n= " + n);
335             System.out.println("f= " + f);
336 
337         outData = 0;
338         outAddr = 0;
339     }
340 
341     //Set DDRAM address
342     private void setDDramAddress(){
343         System.out.println("setDDramAddress, addrValue= " + addrValue);
344         if(checkIni())
345             ramAddr = addrValue;
346 
347         outData = 0;
348         outAddr = 0;
349     }
350 
351     //Read busy flag/address
352     private void returnBusyFlag(){
353         System.out.println("returnBusyFlag, addrRam= " + ramAddr);
354         if(checkIni())
355             outData = ramAddr;
356         else
357             outData = 0;
358 
359         outAddr = 0;
360     }
361 
362     //Write data of RAM
363     private void writeRam(){
364         System.out.println("writeRam()");
365         if(checkIni()) {
366             setRamData();
```

```
367             if(sh){
368                 offSet = id ? (offSet+1) : (offSet-1);
369             }
370         }
371         else
372             outData = 0;
373 
374         outAddr = 0;
375     }
376 
377     //Read data from RAM
378     private void readRam(){
379         System.out.println("readRam()");
380         if(checkIni())
381             outData = getRamData();
382         else
383             outData = 0;
384         outAddr = 0;
385     }
386 
387     /*************************/
388     /* Helper functions      */
389     /*************************/
390 
391 
392     //helper method for read and write operation
393     private int movePosInRam(int num, boolean upDown){
394         int out = 0;
395 
396         if(num == 0x27){
397             System.out.println("præmise: num == 0x27");
398             out = upDown ? 0x40 : (num-1);
399         }
400         else if (num == 0x67) {
401             System.out.println("præmise: num == 0x67");
402             out = upDown ? 0x0 : (num-1);
403         }
404         else if(num == 0x40){
405             System.out.println("præmise: num == 0x40");
406             out = upDown ? (num+1) : 0x27;
407         }
408         else if(num == 0x0){
409             System.out.println("præmise: num == 0x0");
410             out = upDown ? (num+1) : 0x67;
411             System.out.println("out efter opskrivning" + out);
412         }
413         else {
414             System.out.println("standardtilfælde i movePosInRam");
415             out = upDown ? (num+1) : (num-1);
416         }
417 
418         return out;
419     }
420 
421     //helper method for readRam()
422     private char getRamData()
423     {
424         char out;
425 
426         if(ramAddr <= 0x27){
427             out = ddram[0][ramAddr];
428             movePosInRam(ramAddr, id);
429         }
430         else
431             {
432             out = ddram[1][ramAddr - 0x40];
433             movePosInRam(ramAddr, id);
434             }
435         System.out.println("getRamData: ramAddr= " + ramAddr + ", out= " +out);
436         return out;
437     }
438 
439     //helper method for writeRam
440     private void setRamData()
441     {
442         System.out.println("SetRamData: ramAddr= " + ramAddr + ", charValue= " +charValue);
443         if(ramAddr <= 0x27){
444             ddram[0][ramAddr] = (char)charValue;
445             ramAddr = movePosInRam(ramAddr, id);
446         }
447         else
448             {
449             ddram[1][ramAddr - 0x40] = (char)charValue;
450             ramAddr = movePosInRam(ramAddr, id);
```

```
451                 }
452         }
453
454         //helper function to investigate if the display has been correctly
455         //initialised and is turned on
456         private boolean checkIni(){
457             return (setFunction & dispControlSet & firstDisplayClear & d);
458         }
459
460         //Method which generate the output which are sent to the display
461         //This method handles offsets between the initial position in the DDram and an
462         //eventual shifted position.
463         private void generateOutput(){
464             System.out.println("generateOutput, offSet= " + offSet );
465             int count = 0;
466             int pos = 0; //position in array DDram array
467             while(count < 16) {
468                 pos = offSet + count;
469                 if(pos <= 0x27 && pos >= 0){
470                     output[0][count] = ddram[0][pos];
471                     output[1][count] = ddram[1][pos];
472                 }
473                 else if(pos > 0x27){
474                     pos = pos - 0x28;
475                     output[0][count] = ddram[0][pos];
476                     output[1][count] = ddram[1][pos];
477                 }
478                 else{
479                     pos = pos + 0x28;
480                     output[0][count] = ddram[0][pos];
481                     output[1][count] = ddram[1][pos];
482                 }
483                 count++;
484             }
485         }
486
487
488 }
```

C.8 SevenSegment.java

```
1  import java.awt.*;
2  import java.awt.image.ImageProducer;
3
4
5  /** Seven Segment Led Class
6   @author Simon Horman
7   *
8   * <p>(c) 1996 Verge Systems International<br>
9   * The SevenSegment Led Class was consieved, designed and implimented by
10  * Simon Horman of Verge Systems International and remains the property
11  * of Laura Morris.
12  * </p>
13  *
14  * <p>Permision is herby granted to freely use this class for any purpose as
15  * long a the author is acknowladged.
16  * </p>
17  *
18  * <b>Features</b>
19  * <ul>
20  *        <li> Hexadecimal digits 0-F are directly supported.
21  *        <li> Individual Segments can be made active and inactive.
22  *        <li> Can be created in any size.
23  *        <li>  Backround, Led Avtive and Led Inactive coulours can be changed at any time.
24  *        <li> Entire seven segment display can be turned on and off, activating and
25  *        <li> deactivationg the displaying
26  *        <li> of inactive as well as actice leds.
27  * </ul>
28  *
29  * <b>The Segments</b><br>
30  * <pre>
31  *      A
32  *    ----
33  *  F|    |B
34  *   |  G |
35  *    ----
36  *  E|    |C
37  *   |    |
38  *    ----
39  *      D
```

```
40   * </pre>
41   */
42
43  public class SevenSegment extends Canvas{
44     /* The currnet hex value displayed on the Display
45      * 0x10 if segmants are being indivitually manip[ulated. */
46     private int value;
47     // The Image we draw on
48     private Image image;
49     // Our Dimensions
50     private Dimension size;
51     /* The Polygons dor the segments.
52      * We need only render these once */
53     private Polygon pA, p[];
54     /* The state of each segment
55      * 0 inactive, 1 active */
56     private int segment_state[];
57     // The Background Colour
58     private Color background_colour;
59     // The colour of active segments.
60     private Color foreground_colour;
61     // The colour of inactive segments.
62     private Color inactive_colour;
63     // Is the whole display on
64     boolean ison;
65     // The Graphics
66     Graphics g;
67
68     /** Create a Seven Segment display with all segments inactive.
69      @param parent The parent component usually an applet
70      @param width the width for the Seven Segment
71      @param height the height for the Seven Segment
72      */
73     public SevenSegment(Component parent, int width, int height){
74        this(parent, width, height, 0x10);
75     }
76
77     /** Create a Seven Segment display displaying a hexadecimal value.
78      @param parent The parent component usually an applet
79      @param width the width for the Seven Segment
80      @param height the height for the Seven Segment
81      @param value the hexadecimnal value form 0 to F to display
82      */
83     public SevenSegment(Component parent, int width, int height, int value){
84        background_colour = Color.black;
85        foreground_colour = Color.red;
86        inactive_colour =new Color(0x400000);
87
88        ison = true;
89
90        size = new Dimension(width, height);
91
92        image = parent.createImage(size.width, size.height);
93        g = image.getGraphics();
94
95        createSegments();
96        segment_state = new int[7];
97        setValue(value);
98     }
99
100    /** Change the background color. Redraws interbally the Seven Segment
101     * Display but does not paint it onto the parent compotents graphic.
102     @param background the new background colour
103     */
104    public void setBackground(Color background){
105       background_colour = background;
106       draw();
107    }
108
109    /** Change the foreground color. This is the colour that active segments
110     * are painted in Redraws interbally the Seven Segment Display but does
111     * not paint it onto the parent compotents graphic.
112     @param foreground the new foreground colour
113     */
114    public void setForeground(Color foreground){
115       foreground_colour = foreground;
116       draw();
117    }
118
119    /** Change the inactive color. Redraws internally the Seven Segment
120     * Display but does not paint it onto the parent compotents graphic.
121     @param background the new background colour
122     */
123    public void setInactiveColour(Color inactive){
```

```
124         inactive_colour = inactive;
125         draw();
126      }
127
128      /** Set a segment to be active or inactive.
129       @param segment the segment to set 0 corresponds to segment A
130       through to  7 corresponding to segment G.
131       @param value State to set the segment to. 0 inactive 1 active.
132       */
133      public void setSegment(int segment, int value){
134         this.value = 0x10;
135         segment_state[segment] = value;
136      }
137
138      /** Set a segment to be active or inactive.
139       @param segment the segment. A through G are valid.
140       @param value State to set the segment to. 0 inactive 1 active.
141       */
142      public void setSegment(char segment, int value){
143         setSegment((int)(segment-'@'), value);
144      }
145
146      /** Set the Hexadecimal digit displayed by the Seven Segment.
147       @param value The Hexcadegimal digit to display. Valued form 0 to 0xF
148       are valid.
149       */
150      public void setValue(int value){
151         int a, b, c, d;
152         int na, nb, nc, nd;
153
154         if((this.value = value)<0x10){
155        a = value & 1;
156        b = (value & 2) >>> 1;
157        c = (value & 4) >>> 2;
158        d = (value & 8) >>> 3;
159
160        na = ~a & 1;
161        nb = ~b & 1;
162        nc = ~c & 1;
163        nd = ~d & 1;
164
165        //A
166        segment_state[0] = (na&nc)|(na&d)|(b&nd)|(nb&nc&d)|(a&c&nd)|(b&c&d);
167        //B
168        segment_state[1] = (nc&nd)|(na&nc)|(na&nb&nd)|(a&nb&d)|(a&b&nd);
169        //C
170        segment_state[2] = (nb&nd)|(a&nd)|(c&nd)|(nc&d)|(a&nb);
171        //D
172        segment_state[3] = (na&nc&nd)|(a&b&nc)|(a&nb&c)|(na&nb&d)|(na&b&c);
173        //E
174        segment_state[4] = (na&b)|(c&d)|(b&d)|(na&nc);
175        //F
176        segment_state[5] = (na&nb)|(nc&d)|(b&d)|(nb&c&nd)|(na&c&nd);
177        //G
178        segment_state[6] = (na&b)|(nc&d)|(nb&c&nd)|(b&nc&nd)|(a&c&d);
179         }
180      }
181
182      /** Get the colour of inactive leds
183       */
184      public Color getInactiveColour(){
185         return inactive_colour;
186      }
187
188      /** Get the foreground colour. This is the colour used for
189       * active leds
190       */
191      public Color getForeground(){
192         return foreground_colour;
193      }
194
195      /** Get the background Colour
196       */
197      public Color getBackground(){
198         return background_colour;
199      }
200
201      /** Get the state of a segment.
202       @param segment the segment to set 0 corresponds to segment A
203       through to  7 corresponding to segment G.
204       @return 0 corresponds to active and 1 to inactive.
205       */
206      public int getSegment(int segment){
207         return segment_state[segment];
```

```
208     }
209
210     /** Get the state of a segment.
211      @param segment the segment. Valid segments are A to G.
212      @return 0 corresponds to active and 1 to inactive.
213      */
214     public int getSegment(char segment){
215        return getSegment((int)(segment-'@'));
216     }
217
218     /** Get the hexadecimal value currently displayed by the Seven Segment.
219      * A return value of 0x10 signifies that segments are beign manipulated.
220      */
221     public int getValue(){
222        return value;
223     }
224
225     /** An ascii aoutput of the Seven Segment
226      */
227     public String toString(){
228        String string;
229
230        string  = " "              + segment_state[0]
231                                + segment_state[0]                    + "\n";
232        string += segment_state[5] + "  "             + segment_state[1] + "\n";
233        string += segment_state[5] + "  "             + segment_state[1] + "\n";
234        string +=" "               + segment_state[6]
235                                + segment_state[6]                    + "\n";
236        string += segment_state[4] + "  "             + segment_state[2] + "\n";
237        string += segment_state[4] + "  "             + segment_state[2] + "\n";
238        string +=" "               + segment_state[3]
239                                + segment_state[3]                    + "\n";
240
241        return string;
242     }
243
244     // Render the segments
245     private void createSegments(){
246        p = new Polygon[7];
247
248        int gap = (int)(0.01*Math.min(size.width, size.height));
249        if(gap==0){
250       gap = 1;
251        }
252
253        //A
254        p[0] = new Polygon();
255        p[0].addPoint((int)(0.05*size.width),(int)(0.05*size.height)-gap);
256        p[0].addPoint((int)(0.95*size.width),(int)(0.05*size.height)-gap);
257        p[0].addPoint((int)(0.85*size.width),(int)(0.15*size.height)-gap);
258        p[0].addPoint((int)(0.15*size.width),(int)(0.15*size.height)-gap);
259        p[0].addPoint((int)(0.05*size.width),(int)(0.05*size.height)-gap);
260
261        //B
262        p[1] = new Polygon();
263        p[1].addPoint((int)(0.95*size.width)+gap,(int)(0.05*size.height));
264        p[1].addPoint((int)(0.95*size.width)+gap,(int)(0.5*size.height));
265        p[1].addPoint((int)(0.85*size.width)+gap,(int)(0.45*size.height));
266        p[1].addPoint((int)(0.85*size.width)+gap,(int)(0.15*size.height));
267        p[1].addPoint((int)(0.95*size.width)+gap,(int)(0.05*size.height));
268
269        //C
270        p[2] = new Polygon();
271        p[2].addPoint((int)(0.95*size.width)+gap,(int)(0.5*size.height));
272        p[2].addPoint((int)(0.95*size.width)+gap,(int)(0.95*size.height));
273        p[2].addPoint((int)(0.85*size.width)+gap,(int)(0.85*size.height));
274        p[2].addPoint((int)(0.85*size.width)+gap,(int)(0.55*size.height));
275        p[2].addPoint((int)(0.95*size.width)+gap,(int)(0.5*size.height));
276
277        //D
278        p[3] = new Polygon();
279        p[3].addPoint((int)(0.05*size.width),(int)(0.95*size.height)+gap);
280        p[3].addPoint((int)(0.95*size.width),(int)(0.95*size.height)+gap);
281        p[3].addPoint((int)(0.85*size.width),(int)(0.85*size.height)+gap);
282        p[3].addPoint((int)(0.15*size.width),(int)(0.85*size.height)+gap);
283        p[3].addPoint((int)(0.05*size.width),(int)(0.95*size.height)+gap);
284
285        //E
286        p[4] = new Polygon();
287        p[4].addPoint((int)(0.05*size.width)-gap,(int)(0.5*size.height));
288        p[4].addPoint((int)(0.05*size.width)-gap,(int)(0.95*size.height));
289        p[4].addPoint((int)(0.15*size.width)-gap,(int)(0.85*size.height));
290        p[4].addPoint((int)(0.15*size.width)-gap,(int)(0.55*size.height));
291        p[4].addPoint((int)(0.05*size.width)-gap,(int)(0.5*size.height));
```

```
292
293         //F
294         p[5] = new Polygon();
295         p[5].addPoint((int)(0.05*size.width)-gap,(int)(0.05*size.height));
296         p[5].addPoint((int)(0.05*size.width)-gap,(int)(0.5*size.height));
297         p[5].addPoint((int)(0.15*size.width)-gap,(int)(0.45*size.height));
298         p[5].addPoint((int)(0.15*size.width)-gap,(int)(0.15*size.height));
299         p[5].addPoint((int)(0.05*size.width)-gap,(int)(0.05*size.height));
300
301         //G
302         p[6] = new Polygon();
303         p[6].addPoint((int)(0.05*size.width)+gap,(int)(0.5*size.height));
304         p[6].addPoint((int)(0.15*size.width)    ,(int)(0.45*size.height)+gap);
305         p[6].addPoint((int)(0.85*size.width)    ,(int)(0.45*size.height)+gap);
306         p[6].addPoint((int)(0.95*size.width)-gap,(int)(0.5*size.height));
307         p[6].addPoint((int)(0.85*size.width)    ,(int)(0.55*size.height)-gap);
308         p[6].addPoint((int)(0.15*size.width)    ,(int)(0.55*size.height)-gap);
309         p[6].addPoint((int)(0.05*size.width)+gap,(int)(0.5*size.height));
310
311      }
312
313      /** Draw the Seven Segment internally
314       */
315      public void draw(){
316         g.setColor(background_colour);
317         g.fillRect(0, 0, size.width, size.height);
318
319         if(ison){
320        g.setColor(foreground_colour);
321        for(int i=0; i<7;i++){
322           if(segment_state[i]==1){
323              g.fillPolygon(p[i]);
324           }
325        }
326        g.setColor(inactive_colour);
327        for(int i=0; i<7;i++){
328           if(segment_state[i]==0){
329              g.fillPolygon(p[i]);
330           }
331        }
332         }
333      }
334
335      /** Paint the Seven Segment
336       @param graphics the graphics to paint on
337       */
338      public void paint(Graphics g){
339         try{g.drawImage(image, 0, 0, this);}
340         catch(NullPointerException e){}
341      }
342
343      /** Preferred Size for Layout
344       */
345      public Dimension preferredSize(){
346         return size;
347      }
348
349      /** Minimum size for layout
350       */
351      public Dimension minimumSize(){
352         return size;
353      }
354
355      /** Turn the Seven Segment on.
356       * When on active and inactive segments are displayed as expected.
357       * This is the default state for the Seven Segment.
358       */
359      public void turnOn(){
360         ison = true;
361         draw();
362      }
363
364      /** Turn off Seven Segment.
365       * When off the Seven Segment is dawn as a rectangle in its background
366       * colour.
367       */
368      public void turnOff(){
369         ison = false;
370         draw();
371      }
372
373      /** Is the Seven Segment on or off
374       @return true if Seven Segment is on.
375       */
```

```
376     public boolean isOn(){
377        return ison;
378     }
379  }
380
381
382
```

APPENDIX D

Mic-1 and Mic-1IO in Gezel

D.1 The Mic-1 reference version

```
1  //------------------------------------------------
2  // IPBlocks
3  //------------------------------------------------
4  ipblock mic1_rom(
5      in address : ns(9);           // address of control instruction
6      in rd : ns(1);                // read request
7      out odata : ns(36)            // control instruction
8      ) {
9      iptype "rom";                // to be initialised from a *.mic1 file
10     ipparm "size=512";
11     ipparm "wl=36";
12     ipparm "file = mic1ijvm.mic1";
13     ipparm "startbyte=4";        // magic number, 4 by default!
14     }
15
16 ipblock raw_ram(
17     in address : ns(32); // word address
18     in wr : ns(1);
19     in rd : ns(1);
20     in idata : tc(32); // data to be written
21     out odata : tc(32); // data read out
22
23     in bytes : ns(32); // word (of byte) address 0..size-1
24     in fetch : ns(1); // get byte request
25     out byteval : ns(32) // data read out
26     ) {
27     iptype "ijvm";
28     ipparm "size=0x16000";
29     ipparm "file=ijvmtest.ijvm"; // Path to "object program file"
30     }
31 //------------------------------------------------
32 // End of IPBlocks
33 //------------------------------------------------
34
35 //------------------------------------------------
36 // Mic-1 controller
37 //------------------------------------------------
38 dp mic1_ctr(
39     out ctr_rd   : ns(1);    // request to read
40     out ctr_wr   : ns(1);    // request to write
41     out ctr_fetch: ns(1);    // request for byte
42
43     in mbr_out   : ns(8);    // data from MBR register
44
45     in status_N,             // ALU result negative
46        status_Z  : ns(1);    // ALU result zero
```

```
47      out B_sel    : ns(4);    // register to feed B-bus
48      out alu_fct  : ns(8);    // request for ALU function
49      out C_select : ns(9);    // registers to load from C-bus
50
51      out rom_adr  : ns(9);    // virtual MPC 'register'
52      out rom_rd   : ns(1);    // request for next control instruction
53      in  rom_in   : ns(36)    // data from control memory
54      ){
55
56      sig jamz, jamn, jmpc: ns(1);
57      sig mpc: ns(9);
58      reg mir: ns(36);
59
60      sfg c_0 { // action list
61          ctr_rd    = mir[5];
62          ctr_wr    = mir[6];
63          ctr_fetch = mir[4];
64          jamz      = mir[24];
65          jamn      = mir[25];
66          jmpc      = mir[26];
67          mpc       = (jmpc ? mbr_out | mir[27:35] : mir[27:35]);  // CHANGE
68          B_sel     = mir[0:3];
69          alu_fct   = mir[16:23];
70          C_select  = mir[7:15];
71          rom_adr   = mpc | ((status_Z & jamz | status_N & jamn)<<8);  // CHANGE
72          rom_rd    = 1;
73          mir       = rom_in;
74
75          // $display($bin, mir);
76      }
77      }
78  hardwired mic1_ctr_seq(mic1_ctr){ c_0; }
79
80  //-----------------------------------------------
81  // CONTROLLER
82  //-----------------------------------------------
83  dp controller(
84      out ctr_rd   : ns(1);    // request to read
85      out ctr_wr   : ns(1);    // request to write
86      out ctr_fetch: ns(1);    // request for byte
87
88      in mbr_out   : ns(8);    // data from MBR register
89
90      in status_N,             // ALU result negative
91         status_Z  : ns(1);    // ALU result zero
92      out B_sel    : ns(4);    // register to feed B-bus
93      out alu_fct  : ns(8);    // request for ALU function
94      out C_select : ns(9)    // registers to load from C-bus
95      ) {
96
97      sig s_rom_adr : ns(9); // address in microprogram
98      sig s_rom_rd  : ns(1); // read in microprogram
99      sig s_rom_odata : ns(36); // instruction from microprogram
100
101     use mic1_rom(s_rom_adr, s_rom_rd, s_rom_odata);
102     use mic1_ctr(ctr_rd, ctr_wr, ctr_fetch, mbr_out, status_N, status_Z,
103         B_sel, alu_fct, C_select, s_rom_adr, s_rom_rd, s_rom_odata);
104
105     sfg exec {
106
107     }
108     }
109 hardwired ctrl(controller) {exec;}
110
111 //-----------------------------------------------
112 // End of CONTROLLER
113 //-----------------------------------------------
114
115 //-----------------------------------------------
116 // RAM
117 //-----------------------------------------------
118 dp ram_dp (
119     // ports to ijvm ipblock
120     out ram_idata : tc(32);
121     in ram_odata : tc(32);
122     out ram_adr : ns(32);
123     out ram_wr : ns(1);
124     out ram_rd : ns(1);
125     out ram_bytes : ns(32);
126     out ram_fetch : ns(1);
127     in ram_byteval : ns(32);
128
129     // recieved requests
130     in ctr_rd : ns(1);
```

```
131       in ctr_wr : ns(1);
132       in ctr_fetch : ns(1);
133
134       // register connections
135       out mdr_in : ns(32);
136       in mdr_out : tc(32);
137       out mdr_ld : ns(1);
138
139       in pc_out : ns(32);
140       out mbr_in : ns(8);
141       out mbr_ld : ns(1);
142
143       in mar_out : ns(32)
144       ) {
145
146       reg del_mdr : ns(1);
147       reg del_mbr :  ns(1);
148       reg rw_buf : tc(32);
149       reg rb_buf : ns(32);
150       reg byte : ns(2);
151
152       sfg c_0 {
153
154           // Delayed load-signals for elements of the register file
155           del_mdr = ctr_rd;
156           del_mbr = ctr_fetch;
157           byte = (ns(2))pc_out; // Byte-in-word address
158
159           // Read 32-bit words from ijvm-ram
160           ram_adr = ctr_rd | ctr_wr ? mar_out : 0;
161           ram_idata = ctr_wr ? mdr_out : 0;
162           ram_rd = ctr_rd;
163           ram_wr = ctr_wr;
164           rw_buf = ctr_wr ? mdr_out : ram_odata;
165
166           // Read four operation bytes from ijvm-ram
167           ram_bytes = ctr_fetch ? pc_out >> 2 : 0;
168           ram_fetch = ctr_fetch;
169           rb_buf = ctr_fetch ? ram_byteval : rb_buf;
170
171           // set load signals for elements of the register file
172           mdr_ld = del_mdr;
173           mdr_in = del_mdr ? rw_buf : 0;
174           mbr_ld = del_mbr;
175           mbr_in = del_mbr ? (ns(8)) (rb_buf >> ((3-byte) << 3)) : 0;
176
177       }
178       }
179   hardwired ram_dp_seq(ram_dp) {c_0;}
180
181   //-----------------------------------------------
182   // memory_1
183   //-----------------------------------------------
184   dp memory_1(
185       in ctr_rd : ns(1);
186       in ctr_wr : ns(1);
187       in ctr_fetch : ns(1);
188
189       out mdr_in : tc(32);
190       in mdr_out : tc(32);
191       out mdr_ld : ns(1);
192
193       out mbr_in : ns(8);
194       out mbr_ld : ns(1);
195
196       in pc_out : tc(32);
197
198       in mar_out : ns(32)
199       ) {
200
201       sig data_rd, data_wr : ns(1);
202       sig data_address : ns(32);
203       sig data_to_mem, data_from_mem : tc(32);
204
205       sig code_rd : ns(1);
206       sig code_address : ns(32);
207       sig code_from_mem : ns(32);
208
209       use ram_dp(data_to_mem, data_from_mem, data_address, data_wr, data_rd,
210           code_address, code_rd, code_from_mem, ctr_rd, ctr_wr, ctr_fetch,
211           mdr_in, mdr_out, mdr_ld, pc_out, mbr_in, mbr_ld, mar_out);
212       use raw_ram(data_address, data_wr, data_rd, data_to_mem, data_from_mem,
213           code_address, code_rd, code_from_mem);
214
```

```
215      sfg exec {
216
217      }
218      }
219  hardwired ctrl(memory_1) {exec;}
220
221  //-------------------------------------------------
222  // End of memory_1
223  //-------------------------------------------------
224
225  //-------------------------------------------------------
226  // ALU
227  //-------------------------------------------------------
228  dp alu(
229      in  id_x, id_y : tc(32);
230      out od_z       : tc(32);
231      in  ic_sel     : ns(6);
232      out oc_stat    : ns(2)
233      ) {
234      sig s_N, s_Z : ns(1);
235
236      sfg exec {
237          od_z = (ic_sel==0b011000)? id_x :
238             (ic_sel==0b010100)? id_y :
239             (ic_sel==0b011010)? ~id_x :
240             (ic_sel==0b101100)? ~id_y :
241             (ic_sel==0b111100)? id_x + id_y :
242             (ic_sel==0b111101)? id_x + id_y + 1 :
243             (ic_sel==0b111001)? id_x + 1 :
244             (ic_sel==0b110101)? id_y + 1 :
245             (ic_sel==0b111111)? id_y - id_x :
246             (ic_sel==0b110110)? id_y - 1 :
247             (ic_sel==0b111011)? -id_x :
248             (ic_sel==0b001100)? id_x & id_y :
249             (ic_sel==0b011100)? id_x | id_y :
250             (ic_sel==0b010000)? 0 :
251             (ic_sel==0b010001)? 1 :
252             (ic_sel==0b010010)? -1 :
253             0 ;
254          s_N = (od_z < 0);  // flag N
255          s_Z = (od_z == 0); // falg Z
256          oc_stat = s_N#s_Z;
257      }
258      }
259  hardwired ctrl(alu) {exec;}
260  //-------------------------------------------------------
261  // End of ALU
262  //-------------------------------------------------------
263
264  //-------------------------------------------------------
265  // BusB - MUX 9:1
266  //-------------------------------------------------------
267  dp busB(
268      in id_MDR, id_PC, id_MBR1, id_MBR2, id_SP, id_LV, id_CPP, id_TOS, id_OPC : tc(32);
269      out od_z : tc(32);
270      in ic_sel : ns(4)
271      ) {
272
273      sfg exec {
274          od_z = (ic_sel==0) ? id_MDR :
275             (ic_sel==1) ? id_PC :
276             (ic_sel==2) ? id_MBR1 :
277             (ic_sel==3) ? id_MBR2 :
278             (ic_sel==4) ? id_SP :
279             (ic_sel==5) ? id_LV :
280             (ic_sel==6) ? id_CPP :
281             (ic_sel==7) ? id_TOS :
282             (ic_sel==8) ? id_OPC :
283             0;
284      }
285      }
286  hardwired ctrl(busB) {exec;}
287  // -------------------------------------------------
288  // End of BusB - MUX 9:1
289  //-------------------------------------------------
290
291  // -------------------------------------------------
292  // Shifter
293  //-------------------------------------------------
294  dp shifter (
295      in  id_x  : tc(32);
296      out od_z  : tc(32);
297      in  ic_sll8, ic_sra1  : ns(1)
298      ) {
```

```
299
300         sfg exec {
301             od_z = (ic_sll8==1) ? id_x << 8 :
302                (ic_sra1==1) ? id_x >> 1 :
303                id_x;
304         }
305         }
306     hardwired ctrl(shifter) {exec;}
307     // -----------------------------------------------
308     // End of Shifter
309     //-----------------------------------------------
310
311     //-----------------------------------------------
312     // MAR register
313     //-----------------------------------------------
314     dp marR (
315         in id_x : tc(32);
316         out od_q : tc(32);
317         in ic_ld : ns(1)
318         ) {
319         reg rx : tc(32);
320
321         sfg exec {
322              rx = ic_ld ? id_x : rx;
323              od_q = ic_ld ? id_x : rx;
324
325              // $display("--------------------------");
326              // $display("CYCLE: ", $cycle);
327              // $display("MAR = ", rx);
328         }
329         }
330     hardwired ctrl(marR) {exec;}
331
332     dp MAR : marR
333     //-----------------------------------------------
334     // End of MAR register
335     //-----------------------------------------------
336
337     //-----------------------------------------------
338     // MDR register
339     //-----------------------------------------------
340     dp mdrR(
341         in id_x, id_y : tc(32);
342         in ic_x, ic_mem : ns(1);
343         out od_z, od_mem : tc(32)
344         ) {
345         reg rx : tc(32);
346
347         sfg exec {
348
349             rx = ic_x ? id_y : ic_mem ? id_x : rx;
350             od_z = rx;
351             od_mem = ic_x ? id_y : rx;
352
353             // $display("MDR = ", rx);
354         }
355         }
356     hardwired ctrl(mdrR) {exec;}
357
358     dp MDR : mdrR
359     //-----------------------------------------------
360     // End of MDR register
361     //-----------------------------------------------
362
363     //-----------------------------------------------
364     // PC register
365     //-----------------------------------------------
366     dp pcR(
367         in id_x : ns(32);
368         out od_z, od_mem : ns(32);
369         in ic_ld : ns(1)
370         ) {
371         reg rx : ns(32);
372         sfg init {
373             rx = 0xFFFFFFFF;
374             od_z = 0xFFFFFFFF;
375             od_mem = 0;
376
377             // $display("PC = ", rx);
378         }
379         sfg exec {
380             od_z = rx;
381             od_mem = ic_ld ? id_x : rx;
382             rx = ic_ld ? id_x : rx;
```

```
383
384             // $display("PC = ", rx);
385         }
386         }
387     fsm ctrl(pcR) {
388         initial s0;
389         state s1;
390
391         @s0 (init) -> s1;
392         @s1 (exec) -> s1;
393     }
394
395     dp PC : pcR
396     //-----------------------------------------------
397     // End of PC register
398     //-----------------------------------------------
399
400     //-----------------------------------------------
401     // MBR register
402     //-----------------------------------------------
403     dp mbrR(
404         in id_x : ns(8);
405         in ic_x : ns(1);
406         out od_sg : tc(32);
407         out od_un : ns(32);
408         out od_c : ns(8)
409         ){
410
411         // keeps the datavalue
412         reg rx : ns(8);
413
414         sfg init{
415             rx = 0x00;
416
417             //It is assumed that Gezel uses intelligent type casting
418             od_un = (ns(32))rx;
419             od_sg = (tc(32))rx;
420             od_c = rx;
421         }
422
423         sfg exec {
424             // import new data in every cycle
425             rx = ic_x ? id_x : rx;
426
427             //It is assumed that Gezel uses intelligent type casting
428             od_un = (rx >> 7) ? (0-rx) : (ns(32)) rx;
429             od_sg = (rx >> 7) ? (0xFFFFFF00 + rx) : (tc(32)) rx;
430             od_c = rx;
431
432             // $display("MBR = ", rx);
433         }
434         }
435     fsm ctrl(mbrR) {
436         initial s0;
437         state s1;
438
439         @s0(init) -> s1;
440         @s1(exec) -> s1;
441     }
442
443     dp MBR : mbrR
444     //-----------------------------------------------
445     // End of MBR register
446     //-----------------------------------------------
447
448     //-----------------------------------------------
449     // SP register
450     //-----------------------------------------------
451     dp spR(
452         in id_x : tc(32);
453         out od_z : tc(32);
454         in ic_ld : ns(1)
455         ) {
456         reg rx : tc(32);
457
458         sfg init {
459             rx = 0x8000;
460             od_z = rx;
461
462             // $display("SP = ", rx);
463         }
464
465         sfg exec {
466             od_z = rx;
```

```
467             rx = ic_ld ? id_x : rx;
468
469             // $display("SP = ", rx);
470         }
471         }
472     fsm ctrl(spR) {
473         initial s0;
474         state s1;
475
476         @s0 (init) -> s1;
477         @s1 (exec) -> s1;
478     }
479
480     dp SP : spR
481     //-----------------------------------------------
482     // End of SP register
483     //-----------------------------------------------
484
485     //-----------------------------------------------
486     // LV register
487     //-----------------------------------------------
488     dp lvR(
489         in id_x : tc(32);
490         out od_z : tc(32);
491         in ic_ld : ns(1)
492         ) {
493         reg rx : tc(32);
494
495         sfg init {
496             rx = 0xC000;
497             od_z = rx;
498
499             // $display("LV = ", rx);
500         }
501         sfg exec {
502             od_z = rx;
503             rx = ic_ld ? id_x : rx;
504
505             // $display("LV = ", rx);
506         }
507         }
508     fsm ctrl(lvR) {
509         initial s0;
510         state s1;
511
512         @s0 (init) -> s1;
513         @s1 (exec) -> s1;
514     }
515
516     dp LV : lvR
517     //-----------------------------------------------
518     // End of LV register
519     //-----------------------------------------------
520
521     //-----------------------------------------------
522     // CPP register
523     //-----------------------------------------------
524     dp cppR(
525         in id_x : tc(32);
526         out od_z : tc(32);
527         in ic_ld : ns(1)
528         ) {
529         reg rx : tc(32);
530
531         sfg init {
532             rx = 0x4000;
533             od_z = rx;
534
535             // $display("CPP = ", rx);
536         }
537         sfg exec {
538             od_z = rx;
539             rx = ic_ld ? id_x : rx;
540
541             // $display("CPP = ", rx);
542         }
543         }
544     fsm ctrl(cppR) {
545         initial s0;
546         state s1;
547
548         @s0 (init) -> s1;
549         @s1 (exec) -> s1;
550     }
```

```
551
552 dp CPP : cppR
553 //-------------------------------------------------
554 // End of CPP register
555 //-------------------------------------------------
556
557 //-------------------------------------------------
558 // TOS register
559 //-------------------------------------------------
560 dp tosR(
561     in id_x : tc(32);
562     out od_z : tc(32);
563     in ic_ld : ns(1)
564     ){
565     reg rx : tc(32);
566
567     sfg exec {
568         od_z = rx;
569         rx = ic_ld ? id_x : rx;
570
571         // $display("TOS = ", rx);
572     }
573     }
574 hardwired ctrl(tosR) {exec;}
575
576 dp TOS : tosR
577 //-------------------------------------------------
578 // End of TOS register
579 //-------------------------------------------------
580
581 //-------------------------------------------------
582 // OPC register
583 //-------------------------------------------------
584 dp opcR(
585     in id_x : tc(32);
586     out od_z : tc(32);
587     in ic_ld : ns(1)
588     ){
589     reg rx : tc(32);
590
591     sfg exec {
592         od_z = rx;
593         rx = ic_ld ? id_x : rx;
594
595         // $display("OPC = ", rx);
596     }
597     }
598 hardwired ctrl(opcR) {exec;}
599
600 dp OPC : opcR
601 //-------------------------------------------------
602 // End of OPC register
603 //-------------------------------------------------
604
605 //-------------------------------------------------
606 // H register
607 //-------------------------------------------------
608 dp hR(
609     in id_x : tc(32);
610     out od_z : tc(32);
611     in ic_ld : ns(1)
612     ){
613     reg rx : tc(32);
614
615     sfg exec {
616         od_z = rx;
617         rx = ic_ld ? id_x : rx;
618
619         // $display ("H = ", rx);
620     }
621     }
622 hardwired ctrl(hR) {exec;}
623
624 dp H : hR
625 //-------------------------------------------------
626 // End of H register
627 //-------------------------------------------------
628
629 dp ALU : alu
630 dp Shifter : shifter
631 dp BusB : busB
632 dp Ctrl : mic1_ctr
633 //-------------------------------------------------------
634 //               DATAPATH
```

```
635 //-------------------------------------------------------
636 dp datapath(
637     in  mdr_in:  tc(32);     // data from memory
638     out mdr_out: tc(32);     // data to memory
639     in  mdr_ld:  ns(1);      // load MDR from memory
640
641     out pc_out:  tc(32);     // address to memory
642     in  mbr_in:  ns(8);      // byte from memory
643     in  mbr_ld:  ns(1);      // load MBR from memory
644
645     out mar_out: ns(32);     // address to memory
646     out mbr_out: ns(8);      // byte to mic1_ctr
647
648     out status_N, status_Z : ns(1);
649
650     in B_sel: ns(4);         // select data for the B-bus
651     in alu_fct: ns(8);       // select alu function AND shifter function
652     in C_sel: ns(9)          // select target registers
653     ) {
654
655     sig s_od_mar, s_od_mdr, s_od_pc, s_od_mbr_un, s_od_mbr_sg, s_od_sp,
656         s_od_lv, s_od_cpp, s_od_tos, s_od_opc, s_od_h : tc(32);
657     sig s_C_sel_mar, s_C_sel_mdr, s_C_sel_pc, s_C_sel_sp, s_C_sel_lv,
658         s_C_sel_cpp, s_C_sel_tos, s_C_sel_opc, s_C_sel_h : ns(1);
659     sig s_od_mux91 : tc(32);
660     sig s_od_alu : tc(32);
661     sig s_ic_alu : ns(6);
662     sig s_od_shifter : tc(32);
663     sig s_ic_sl, s_ic_sr : ns(1);
664     sig s_ic_mux91 : ns(4);
665     sig s_ctr_rd, s_ctr_wr, s_ctr_fetch :ns(1);
666     sig s_ctr_mbr : ns(8);
667     sig s_alu_status_N, s_alu_status_Z : ns(1);
668     sig s_alu_fct : ns(8);
669         sig s_alu_status : ns(2);
670     sig s_rom_adr : ns(9);
671     sig s_rom_rd : ns(1);
672     sig s_rom_in : ns(36);
673
674     use MAR(s_od_shifter, s_od_mar, s_C_sel_mar);
675     use MDR(mdr_in, s_od_shifter, s_C_sel_mdr, mdr_ld, s_od_mdr, mdr_out);
676     use PC(s_od_shifter, s_od_pc, pc_out, s_C_sel_pc);
677     use MBR(mbr_in, mbr_ld, s_od_mbr_sg, s_od_mbr_un, s_ctr_mbr);
678     use SP(s_od_shifter, s_od_sp, s_C_sel_sp);
679     use LV(s_od_shifter, s_od_lv, s_C_sel_lv);
680     use CPP(s_od_shifter, s_od_cpp, s_C_sel_cpp);
681     use TOS(s_od_shifter, s_od_tos, s_C_sel_tos);
682     use OPC(s_od_shifter, s_od_opc, s_C_sel_opc);
683     use H(s_od_shifter, s_od_h, s_C_sel_h);
684     use ALU(s_od_h, s_od_mux91, s_od_alu, s_ic_alu, s_alu_status);
685     use Shifter(s_od_alu, s_od_shifter, s_ic_sl, s_ic_sr);
686     use BusB(s_od_mdr, s_od_pc, s_od_mbr_sg, s_od_mbr_un, s_od_sp, s_od_lv,
687         s_od_cpp, s_od_tos, s_od_opc, s_od_mux91, s_ic_mux91);
688
689     sfg exec{
690         mar_out = s_od_mar;
691         mbr_out = s_ctr_mbr;
692         status_N = s_alu_status[1];
693         status_Z = s_alu_status[0];
694
695         s_C_sel_mar = C_sel[0];
696         s_C_sel_mdr = C_sel[1];
697         s_C_sel_pc = C_sel[2];
698         s_C_sel_sp = C_sel[3];
699         s_C_sel_lv = C_sel[4];
700         s_C_sel_cpp = C_sel[5];
701         s_C_sel_tos = C_sel[6];
702         s_C_sel_opc = C_sel[7];
703         s_C_sel_h = C_sel[8];
704
705         s_ic_mux91 = B_sel;
706         s_ic_alu = alu_fct[5:0];
707         s_ic_sr = alu_fct[6];
708         s_ic_sl = alu_fct[7];
709     }
710     }
711 hardwired ctrl(datapath) {exec;}
712 //-------------------------------------------------------------
713 // End of datapath
714 //-------------------------------------------------------------
715
716 //-------------------------------------------------------------
717 // System
718 //-------------------------------------------------------------
```

```
719 system mic1_system {
720     controller(ctr_rd, ctr_wr, ctr_fetch, mbr_out, status_N, status_Z,
721     B_sel, alu_fct, C_sel);
722     memory_1(ctr_rd, ctr_wr, ctr_fetch, mdr_in, mdr_out, mdr_ld, mbr_in,
723     mbr_ld, pc_out, mar_out);
724     datapath(mdr_in, mdr_out, mdr_ld, pc_out, mbr_in, mbr_ld, mar_out,
725     mbr_out, status_N, status_Z, B_sel, alu_fct, C_sel);
726 }
727 //-------------------------------------------------------------
728 // End of System
729 //-------------------------------------------------------------
```

D.2 The Mic-1IO with GCD calculator

```
1
2  //-----------------------------------------------
3  // Daisy chain block
4  //-----------------------------------------------
5  dp daisy_chain_block(
6
7      in iord : ns(1);
8
9      in adress_from_mic2 : tc(32);
10     in data_from_mic2 : tc(32);
11
12     out adress_to_mic2 : tc(32);
13     out data_to_mic2 : tc(32);
14
15     out iod_ld_to_mic2 : ns(1);
16     out ioa_ld_to_mic2 : ns(1);
17
18     out adress_to_terminator : tc(32);
19     out data_to_terminator : tc(32);
20
21     in adress_from_terminator : tc(32);
22     in data_from_terminator : tc(32);
23
24     in iod_ld_from_terminator : ns(1);
25     in ioa_ld_from_terminator : ns(1);
26
27     out adress_to_device : tc(32);
28     out data_to_device : tc(32);
29
30     in adress_from_device : tc(32);
31     in data_from_device : tc(32);
32
33     in iod_ld_from_device : ns(1);
34     in ioa_ld_from_device : ns(1);
35
36     out device_iord : ns(1)
37     ) {
38     reg lower_bound, upper_bound : tc(32);
39
40     sfg init {
41                 device_iord = iord;
42                 adress_to_mic2 = 0;
43                 data_to_mic2 = 0;
44                 iod_ld_to_mic2 = 0;
45                 ioa_ld_to_mic2 = 0;
46                 adress_to_terminator = 0;
47                 data_to_terminator = 0;
48                 adress_to_device = 0;
49                 data_to_device = 0;
50                 lower_bound = 1;
51                 upper_bound = 1;
52              }
53
54     sfg exec {
55
56 //        $display("---------------------------");
57 //        $display("CYCLE: ", $cycle);
58 //        $display("Daisy chain: Address from mic2: ", $hex, adress_from_mic2);
59 //        $display("Daisy chain: Data from mic2: ", $hex, data_from_mic2);
60 //        $display("Daisy chain: Address to mic2: ", $hex, adress_to_mic2);
61 //        $display("Daisy chain: Data to mic2: ", $hex, data_to_mic2);
62 //          $display("Daisy chain: Address to device: ", $hex, adress_to_device);
63 //          $display("Daisy chain: Address to terminator: ", $hex, adress_to_terminator);
64 //        $display("Daisy chain: IOD load to cpu: ", iod_ld_to_mic2);
65 //        $display("Daisy chain: IOA load to cpu: ", ioa_ld_to_mic2);
66
```

```
67
68          adress_to_terminator = (adress_from_mic2 > upper_bound) ?
69                                 adress_from_mic2 :
70                    (adress_from_mic2 < lower_bound) ?
71                    adress_from_mic2 : 0;
72          data_to_terminator = (adress_from_mic2 > upper_bound) ?
73                               data_from_mic2 :
74                   (adress_from_mic2 < lower_bound) ?
75                   data_from_mic2 : 0;
76          adress_to_device = (adress_from_mic2 > upper_bound) ?
77                          0 : (adress_from_mic2 < lower_bound) ?
78                        0 : adress_from_mic2 - (lower_bound - 1);
79          data_to_device = (adress_from_mic2 > upper_bound) ?
80                   0 :
81                   (adress_from_mic2 < lower_bound) ?
82                   0 : data_from_mic2;
83          adress_to_mic2 = (adress_from_mic2 > upper_bound) ?
84                           adress_from_terminator :
85                   (adress_from_mic2 < lower_bound) ?
86                   adress_from_terminator : adress_from_device;
87          data_to_mic2 = (adress_from_mic2 > upper_bound) ?
88                 data_from_terminator :
89                 (adress_from_mic2 < lower_bound) ?
90                 data_from_terminator : data_from_device;
91          iod_ld_to_mic2 = iod_ld_from_terminator | iod_ld_from_device;
92          ioa_ld_to_mic2 = ioa_ld_from_terminator | ioa_ld_from_device;
93          device_iord = iord;
94      }
95      }
96      fsm ctrl(daisy_chain_block) {
97      initial s0;
98      state s1;
99
100     @s0 (init) -> s1;
101     @s1 (exec) -> s1;
102 }
103
104 //-----------------------------------------------
105 // End of Daisy chain block
106 //-----------------------------------------------
107
108 //-----------------------------------------------
109 // Daisy chain terminator
110 //-----------------------------------------------
111 dp daisy_chain_terminator(
112     in adress_from_micIO : tc(32);
113     in data_from_micIO : tc(32);
114
115     out adress_to_micIO : tc(32);
116     out data_to_micIO : tc(32);
117
118     out iod_ld : ns(1);
119     out ioa_ld : ns(1)
120     ) {
121     reg address : ns(32);
122
123     sfg init {
124
125 //        $display("Terminator: Address from micIO: ", $hex, adress_from_micIO);
126 //        $display("Terminator: Data from micIO: ", $hex, data_from_micIO);
127 //        $display("Terminator: Address to micIO: ", $hex, adress_to_micIO);
128 //        $display("Terminator: Data to micIO: ", $hex, data_to_micIO);
129
130         address = adress_from_micIO;
131         adress_to_micIO = 0;
132         data_to_micIO = (adress_from_micIO != 0) ? 0xfeedbeef : 0;
133         iod_ld = (adress_from_micIO != 0) ? 1 : 0;
134         ioa_ld = (adress_from_micIO != 0) ? 1 : 0;
135     }
136
137     sfg success {
138         $display("      Cycle: ", $cycle);
139         $display("Terminator received address: ", address);
140         $display("*********************************");
141         $display("       Test was successful      ");
142         $display("*********************************");
143         $finish;}
144
145     sfg error {
146         $display("      Cycle: ", $cycle);
147         $display("Terminator received address: ", address);
148         $display("*********************************");
149         $display("           Test failed           ");
150         $display("*********************************");
```

```
151         $finish;}
152
153
154
155 }
156 fsm ctrl(daisy_chain_terminator) {
157 initial running;
158
159 @running if(address == 35) then (success,init) -> running;
160  else if(address == 40) then (error,init) -> running;
161  else (init) -> running;
162
163
164 }
165 //-----------------------------------------------
166 // End of Daisy chain terminator
167 //-----------------------------------------------
168
169 //-----------------------------------------------
170 // IPBlocks
171 //-----------------------------------------------
172
173 ipblock micIO_rom(
174     in address : ns(9);           // address of control instruction
175     in rd : ns(1);                // read request
176     out odata : ns(39)            // control instruction
177     ) {
178     iptype "rom";                // to be initialised from a *.mic1 file
179     ipparm "size=512";
180     ipparm "wl=39";
181     ipparm "file = ../../Microcode/mic1IOijvm.mic1";
182     ipparm "startbyte=4";        // magic number, 4 by default!
183     }
184
185 ipblock raw_ram(
186     in address : ns(32); // word address
187     in wr : ns(1);
188     in rd : ns(1);
189     in idata : tc(32); // data to be written
190     out odata : tc(32); // data read out
191
192     in bytes : ns(32); // word (of byte) address 0..size-1
193     in fetch : ns(1); // get byte request
194     out byteval : ns(32) // data read out
195     ) {
196     iptype "ijvm";
197     ipparm "size=0x16000";
198     ipparm "file=../../AssTestPrograms/ModifiedOntko/ijvmtest.ijvm"; // Path to "object program file"
199     }
200 //-----------------------------------------------
201 // End of IPBlocks
202 //-----------------------------------------------
203
204
205
206 //-----------------------------------------------
207 // Mic-1IO controller
208 //-----------------------------------------------
209 dp mic1IO_ctr(
210     out ctr_rd   : ns(1);    // request to read
211     out ctr_wr   : ns(1);    // request to write
212     out ctr_fetch: ns(1);    // request for byte
213
214     in mbr_out   : ns(8);    // data from MBR register
215
216     in status_N,             // ALU result negative
217        status_Z  : ns(1);    // ALU result zero
218     out B_sel    : ns(4);    // register to feed B-bus
219     out alu_fct  : ns(8);    // request for ALU function
220     out C_select : ns(11);    // registers to load from C-bus
221
222     out rom_adr  : ns(9);    // virtual MPC 'register'
223     out rom_rd   : ns(1);    // request for next control instruction
224     in  rom_in   : ns(39);    // data from control memory
225
226     out io_read : ns(1) // Read/Write signal for peripheral units
227     ){
228
229     sig jamz, jamn, jmpc: ns(1);
230     sig mpc: ns(9);
231     reg mir: ns(39);
232
233     sfg c_0 { // action list
234         ctr_rd    = mir[5];
```

```
235           ctr_wr    = mir[6];
236           ctr_fetch = mir[4];
237           jamz      = mir[27];
238           jamn      = mir[28];
239           jmpc      = mir[29];
240           mpc       = (jmpc ? mbr_out | mir[30:38] : mir[30:38]);
241           B_sel     = mir[0:3];
242           alu_fct   = mir[19:26];
243           C_select  = mir[8:18];
244           rom_adr   = mpc | ((status_Z & jamz | status_N & jamn)<<8);
245           rom_rd    = 1;
246           mir       = rom_in;
247           io_read = mir[7];
248
249   //        $display($bin, mir);
250   //        $display("mic1IO_ctr: iord: ", $dec, io_read);
251       }
252       }
253   hardwired mic1IO_ctr_seq(mic1IO_ctr){c_0;}
254
255   //-----------------------------------------------
256   // CONTROLLER
257   //-----------------------------------------------
258   dp controller(
259       out ctr_rd   : ns(1);    // request to read
260       out ctr_wr   : ns(1);    // request to write
261       out ctr_fetch: ns(1);    // request for byte
262
263       in mbr_out   : ns(8);    // data from MBR register
264
265       in status_N,             // ALU result negative
266          status_Z  : ns(1);    // ALU result zero
267       out B_sel    : ns(4);    // register to feed B-bus
268       out alu_fct  : ns(8);    // request for ALU function
269       out C_select : ns(11);    // registers to load from C-bus
270
271       out io_read : ns(1) // Read/Write signal for peripheral units
272       ) {
273
274       sig s_rom_adr : ns(9); // address in microprogram
275       sig s_rom_rd  : ns(1); // read in microprogram
276       sig s_rom_odata : ns(39); // instruction from microprogram
277
278       use micIO_rom(s_rom_adr, s_rom_rd, s_rom_odata);
279       use mic1IO_ctr(ctr_rd, ctr_wr, ctr_fetch, mbr_out, status_N,
280           status_Z, B_sel, alu_fct, C_select, s_rom_adr, s_rom_rd,
281           s_rom_odata, io_read);
282
283       sfg exec {
284
285   //        $display("Controller: iord: ", $dec, io_read);
286
287       }
288       }
289   hardwired ctrl(controller) {exec;}
290
291   //-----------------------------------------------
292   // End of CONTROLLER
293   //-----------------------------------------------
294
295   //-----------------------------------------------
296   // RAM
297   //-----------------------------------------------
298   dp ram_dp (
299       // ports to ijvm ipblock
300       out ram_idata : tc(32);
301       in ram_odata : tc(32);
302       out ram_adr : ns(32);
303       out ram_wr : ns(1);
304       out ram_rd : ns(1);
305       out ram_bytes : ns(32);
306       out ram_fetch : ns(1);
307       in ram_byteval : ns(32);
308
309       // recieved requests
310       in ctr_rd : ns(1);
311       in ctr_wr : ns(1);
312       in ctr_fetch : ns(1);
313
314       // register connections
315       out mdr_in : ns(32);
316       in mdr_out : tc(32);
317       out mdr_ld : ns(1);
318
```

```
319         in pc_out : ns(32);
320         out mbr_in : ns(8);
321         out mbr_ld : ns(1);
322
323         in mar_out : ns(32)
324         ) {
325
326         reg del_mdr : ns(1);
327         reg del_mbr :  ns(1);
328         reg rw_buf : tc(32);
329         reg rb_buf : ns(32);
330         reg byte : ns(2);
331
332         sfg c_0 {
333
334             // Delayed load-signals for elements of the register file
335             del_mdr = ctr_rd;
336             del_mbr = ctr_fetch;
337             byte = (ns(2))pc_out; // Byte-in-word address
338
339             // Read 32-bit words from ijvm-ram
340             ram_adr = ctr_rd | ctr_wr ? mar_out : 0;
341             ram_idata = ctr_wr ? mdr_out : 0;
342             ram_rd = ctr_rd;
343             ram_wr = ctr_wr;
344             rw_buf = ctr_wr ? mdr_out : ram_odata;
345
346             // Read four operation bytes from ijvm-ram
347             ram_bytes = ctr_fetch ? pc_out >> 2 : 0;
348             ram_fetch = ctr_fetch;
349             rb_buf = ctr_fetch ? ram_byteval : rb_buf;
350
351             // set load signals for elements of the register file
352             mdr_ld = del_mdr;
353             mdr_in = del_mdr ? rw_buf : 0;
354             mbr_ld = del_mbr;
355             mbr_in = del_mbr ? (ns(8)) (rb_buf >> ((3-byte) << 3)) : 0;
356
357         }
358         }
359     hardwired ram_dp_seq(ram_dp) {c_0;}
360
361     //-----------------------------------------------
362     // memory_1
363     //-----------------------------------------------
364     dp memory_1(
365         in ctr_rd : ns(1);
366         in ctr_wr : ns(1);
367         in ctr_fetch : ns(1);
368
369         out mdr_in : tc(32);
370         in mdr_out : tc(32);
371         out mdr_ld : ns(1);
372
373         out mbr_in : ns(8);
374         out mbr_ld : ns(1);
375
376         in pc_out : tc(32);
377
378         in mar_out : ns(32)
379         ) {
380
381         sig data_rd, data_wr : ns(1);
382         sig data_address : ns(32);
383         sig data_to_mem, data_from_mem : tc(32);
384
385         sig code_rd : ns(1);
386         sig code_address : ns(32);
387         sig code_from_mem : ns(32);
388
389         use ram_dp(data_to_mem, data_from_mem, data_address, data_wr, data_rd,
390             code_address, code_rd, code_from_mem, ctr_rd, ctr_wr, ctr_fetch,
391             mdr_in, mdr_out, mdr_ld, pc_out, mbr_in, mbr_ld, mar_out);
392         use raw_ram(data_address, data_wr, data_rd, data_to_mem, data_from_mem,
393             code_address, code_rd, code_from_mem);
394
395         sfg exec {
396
397         }
398         }
399     hardwired ctrl(memory_1) {exec;}
400
401     //-----------------------------------------------
402     // End of memory_1
```

```
403 //-------------------------------------------------
404
405 //-------------------------------------------------------
406 // ALU
407 //-------------------------------------------------------
408 dp alu(
409     in  id_x, id_y : tc(32);
410     out od_z       : tc(32);
411     in  ic_sel     : ns(6);
412     out oc_stat    : ns(2)
413     ) {
414     sig s_N, s_Z : ns(1);
415
416     sfg exec {
417         od_z = (ic_sel==0b011000)? id_x :
418            (ic_sel==0b010100)? id_y :
419            (ic_sel==0b011010)? ~id_x :
420            (ic_sel==0b101100)? ~id_y :
421            (ic_sel==0b111100)? id_x + id_y :
422            (ic_sel==0b111101)? id_x + id_y + 1 :
423            (ic_sel==0b111001)? id_x + 1 :
424            (ic_sel==0b110101)? id_y + 1 :
425            (ic_sel==0b111111)? id_y - id_x :
426            (ic_sel==0b110110)? id_y - 1 :
427            (ic_sel==0b111011)? -id_x :
428            (ic_sel==0b001100)? id_x & id_y :
429            (ic_sel==0b011100)? id_x | id_y :
430            (ic_sel==0b010000)? 0 :
431            (ic_sel==0b010001)? 1 :
432            (ic_sel==0b010010)? -1 :
433            0 ;
434         s_N = (od_z < 0);  // flag N
435         s_Z = (od_z == 0); // falg Z
436         oc_stat = s_N#s_Z;
437     }
438     }
439 hardwired ctrl(alu) {exec;}
440 //-------------------------------------------------------
441 // End of ALU
442 //-------------------------------------------------------
443
444 //-------------------------------------------------------
445 // BusB - MUX 9:1
446 //-------------------------------------------------------
447 dp busB(
448     in id_MDR, id_PC, id_MBR1, id_MBR2, id_SP, id_LV, id_CPP, id_TOS,
449     id_OPC, id_IOD, id_IOA : tc(32);
450     out od_z : tc(32);
451     in ic_sel : ns(4)
452     ) {
453
454     sfg exec {
455         od_z = (ic_sel==0) ? id_MDR :
456            (ic_sel==1) ? id_PC :
457            (ic_sel==2) ? id_MBR1 :
458            (ic_sel==3) ? id_MBR2 :
459            (ic_sel==4) ? id_SP :
460            (ic_sel==5) ? id_LV :
461            (ic_sel==6) ? id_CPP :
462            (ic_sel==7) ? id_TOS :
463            (ic_sel==8) ? id_OPC :
464            (ic_sel==9) ? id_IOD :
465            (ic_sel==10) ? id_IOA :
466            0;
467     }
468     }
469 hardwired ctrl(busB) {exec;}
470 // -----------------------------------------------
471 // End of BusB - MUX 9:1
472 //-----------------------------------------------
473
474 // -----------------------------------------------
475 // Shifter
476 //-----------------------------------------------
477 dp shifter (
478     in  id_x  : tc(32);
479     out od_z  : tc(32);
480     in  ic_sll8, ic_sra1  : ns(1)
481     ) {
482
483     sfg exec {
484         od_z = (ic_sll8==1) ? id_x << 8 :
485            (ic_sra1==1) ? id_x >> 1 :
486            id_x;
```

```
487         }
488         }
489     hardwired ctrl(shifter) {exec;}
490     // ------------------------------------------------
491     // End of Shifter
492     //-------------------------------------------------
493
494     //-------------------------------------------------
495     // MAR register
496     //-------------------------------------------------
497     dp MAR(
498         in id_x : tc(32);
499         out od_q : tc(32);
500         in ic_ld : ns(1)
501         ) {
502         reg rx : tc(32);
503
504         sfg exec {
505              rx = ic_ld ? id_x : rx;
506              od_q = ic_ld ? id_x : rx;
507
508     //         $display("MAR = ", rx);
509         }
510         }
511     hardwired ctrl(MAR) {exec;}
512     //-------------------------------------------------
513     // End of MAR register
514     //-------------------------------------------------
515
516     //-------------------------------------------------
517     // MDR register
518     //-------------------------------------------------
519     dp MDR(
520         in id_x, id_y : tc(32);
521         in ic_x, ic_mem : ns(1);
522         out od_z, od_mem : tc(32)
523         ) {
524         reg rx : tc(32);
525
526         sfg exec {
527
528             rx = ic_x ? id_y : ic_mem ? id_x : rx;
529             od_z = rx;
530             od_mem = ic_x ? id_y : rx;
531
532     //        $display("MDR = ",$dec, rx);
533         }
534         }
535     hardwired ctrl(MDR) {exec;}
536     //-------------------------------------------------
537     // End of MDR register
538     //-------------------------------------------------
539
540     //-------------------------------------------------
541     // PC register
542     //-------------------------------------------------
543     dp PC(
544         in id_x : ns(32);
545         out od_z, od_mem : ns(32);
546         in ic_ld : ns(1)
547         ) {
548         reg rx : ns(32);
549         sfg init {
550             rx = 0xFFFFFFFF;
551             od_z = 0xFFFFFFFF;
552             od_mem = 0;
553
554     //        $display("PC = ", rx);
555         }
556         sfg exec {
557             od_z = rx;
558             od_mem = ic_ld ? id_x : rx;
559             rx = ic_ld ? id_x : rx;
560
561     //        $display("PC = ", rx);
562         }
563         }
564     fsm ctrl(PC) {
565         initial s0;
566         state s1;
567
568         @s0 (init) -> s1;
569         @s1 (exec) -> s1;
570     }
```

```
571 //-----------------------------------------------
572 // End of PC register
573 //-----------------------------------------------
574
575 //-----------------------------------------------
576 // MBR register
577 //-----------------------------------------------
578 dp MBR(
579     in id_x : ns(8);
580     in ic_x : ns(1);
581     out od_sg : tc(32);
582     out od_un : ns(32);
583     out od_c : ns(8)
584     ){
585
586     // keeps the datavalue
587     reg rx : ns(8);
588
589     sfg init{
590         rx = 0x00;
591
592         //It is assumed that Gezel uses intelligent type casting
593         od_un = (ns(32))rx;
594         od_sg = (tc(32))rx;
595         od_c = rx;
596     }
597
598     sfg exec {
599         // import new data in every cycle
600         rx = ic_x ? id_x : rx;
601
602         //It is assumed that Gezel uses intelligent type casting
603         od_un = (rx >> 7) ? (0-rx) : (ns(32)) rx;
604         od_sg = (rx >> 7) ? (0xFFFFFF00 + rx) : (tc(32)) rx;
605         od_c = rx;
606
607 //        $display("MBR = ", rx);
608     }
609     }
610 fsm ctrl(MBR) {
611     initial s0;
612     state s1;
613
614     @s0(init) -> s1;
615     @s1(exec) -> s1;
616 }
617 //-----------------------------------------------
618 // End of MBR register
619 //-----------------------------------------------
620
621 //-----------------------------------------------
622 // SP register
623 //-----------------------------------------------
624 dp SP(
625     in id_x : tc(32);
626     out od_z : tc(32);
627     in ic_ld : ns(1)
628     ) {
629     reg rx : tc(32);
630
631     sfg init {
632         rx = 0x8000;
633         od_z = rx;
634
635 //        $display("SP = ", rx);
636     }
637
638     sfg exec {
639         od_z = rx;
640         rx = ic_ld ? id_x : rx;
641
642 //        $display("SP = ", rx);
643     }
644     }
645 fsm ctrl(SP) {
646     initial s0;
647     state s1;
648
649     @s0 (init) -> s1;
650     @s1 (exec) -> s1;
651 }
652 //-----------------------------------------------
653 // End of SP register
654 //-----------------------------------------------
```

```
655
656 //-----------------------------------------------
657 // LV register
658 //-----------------------------------------------
659 dp LV(
660     in id_x : tc(32);
661     out od_z : tc(32);
662     in ic_ld : ns(1)
663     ) {
664     reg rx : tc(32);
665
666     sfg init {
667         rx = 0xC000;
668         od_z = rx;
669
670 //        $display("LV = ", rx);
671     }
672     sfg exec {
673         od_z = rx;
674         rx = ic_ld ? id_x : rx;
675
676 //        $display("LV = ", rx);
677     }
678     }
679 fsm ctrl(LV) {
680     initial s0;
681     state s1;
682
683     @s0 (init) -> s1;
684     @s1 (exec) -> s1;
685 }
686 //-----------------------------------------------
687 // End of LV register
688 //-----------------------------------------------
689
690 //-----------------------------------------------
691 // CPP register
692 //-----------------------------------------------
693 dp CPP(
694     in id_x : tc(32);
695     out od_z : tc(32);
696     in ic_ld : ns(1)
697     ) {
698     reg rx : tc(32);
699
700     sfg init {
701         rx = 0x4000;
702         od_z = rx;
703
704 //        $display("CPP = ", rx);
705     }
706     sfg exec {
707         od_z = rx;
708         rx = ic_ld ? id_x : rx;
709
710 //        $display("CPP = ", rx);
711     }
712     }
713 fsm ctrl(CPP) {
714     initial s0;
715     state s1;
716
717     @s0 (init) -> s1;
718     @s1 (exec) -> s1;
719 }
720 //-----------------------------------------------
721 // End of CPP register
722 //-----------------------------------------------
723
724 //-----------------------------------------------
725 // TOS register
726 //-----------------------------------------------
727 dp TOS(
728     in id_x : tc(32);
729     out od_z : tc(32);
730     in ic_ld : ns(1)
731     ){
732     reg rx : tc(32);
733
734     sfg exec {
735         od_z = rx;
736         rx = ic_ld ? id_x : rx;
737
738 //        $display("TOS = ", rx);
```

```
739         }
740         }
741     hardwired ctrl(TOS) {exec;}
742     //-----------------------------------------------
743     // End of TOS register
744     //-----------------------------------------------
745
746     //-----------------------------------------------
747     // OPC register
748     //-----------------------------------------------
749     dp OPC(
750         in id_x : tc(32);
751         out od_z : tc(32);
752         in ic_ld : ns(1)
753         ){
754         reg rx : tc(32);
755
756         sfg exec {
757             od_z = rx;
758             rx = ic_ld ? id_x : rx;
759
760     //        $display("OPC = ", rx);
761         }
762         }
763     hardwired ctrl(OPC) {exec;}
764     //-----------------------------------------------
765     // End of OPC register
766     //-----------------------------------------------
767
768     //-----------------------------------------------
769     // H register
770     //-----------------------------------------------
771     dp H(
772         in id_x : tc(32);
773         out od_z : tc(32);
774         in ic_ld : ns(1)
775         ){
776         reg rx : tc(32);
777
778         sfg exec {
779             od_z = rx;
780             rx = ic_ld ? id_x : rx;
781
782     //        $display("H = ", rx);
783         }
784         }
785     hardwired ctrl(H) {exec;}
786     //-----------------------------------------------
787     // End of H register
788     //-----------------------------------------------
789
790     //-----------------------------------------------
791     // IOD register
792     //-----------------------------------------------
793     dp IOD(
794         in id_x, id_y : tc(32);
795         in ic_x, ic_mem : ns(1);
796         out od_z : tc(32)
797         ) {
798         reg rx : tc(32);
799
800         sfg exec {
801
802             rx = ic_x ? id_y : ic_mem ? id_x : rx;
803             od_z = rx;
804
805     //        $display("IOD = ",$dec, rx);
806         }
807         }
808     hardwired ctrl(IOD) {exec;}
809     //-----------------------------------------------
810     // End of IOD register
811     //-----------------------------------------------
812
813     //-----------------------------------------------
814     // IOA register
815     //-----------------------------------------------
816     dp IOA(
817         in id_x, id_y : tc(32);
818         in ic_x, ic_mem : ns(1);
819         out od_z : tc(32)
820         ) {
821         reg rx : tc(32);
822
```

```
823         sfg exec {
824
825             rx = ic_x ? id_y :
826                  ic_mem ? id_x : rx;
827             od_z = rx;
828
829     //        $display("IOA = ",$dec, rx);
830     //        $display("IOA: od_z: ", od_z);
831         }
832         }
833     hardwired ctrl(IOA) {exec;}
834     //-----------------------------------------------
835     // End of IOA register
836     //-----------------------------------------------
837
838     dp ALU : alu
839     dp Shifter : shifter
840     dp BusB : busB
841     dp Ctrl : mic1IO_ctr
842     //-------------------------------------------------------
843     //               DATAPATH
844     //-------------------------------------------------------
845     dp datapath(
846         in  mdr_in : tc(32);     // data from memory
847         out mdr_out : tc(32);     // data to memory
848         in  mdr_ld : ns(1);      // load MDR from memory
849
850         out pc_out : tc(32);     // address to memory
851         in  mbr_in : ns(8);      // byte from memory
852         in  mbr_ld : ns(1);      // load MBR from memory
853
854         out mar_out : ns(32);     // address to memory
855         out mbr_out : ns(8);      // byte to mic1IO_ctr
856
857         out status_N, status_Z : ns(1);
858
859         in B_sel : ns(4);         // select data for the B-bus
860         in alu_fct : ns(8);       // select alu function AND shifter function
861         in C_sel : ns(11);          // select target registers
862
863         out iod_out : tc(32);    // I/O data register out
864         out ioa_out : tc(32);    // I/O adress register out
865
866         in iod_in : tc(32);        // I/O data to IOD register
867
868         in ioa_in : ns(32);        // I/O address to IOA register
869
870
871
872         in iod_ld : ns(1);        // IOD load signal
873         in ioa_ld : ns(1)        // IOA load signal
874         ) {
875
876         sig s_od_mar, s_od_mdr, s_od_pc, s_od_mbr_un, s_od_mbr_sg,
877             s_od_sp, s_od_lv, s_od_cpp, s_od_tos, s_od_opc, s_od_h,
878             s_od_iod, s_od_ioa : tc(32);
879         sig s_C_sel_mar, s_C_sel_mdr, s_C_sel_pc, s_C_sel_sp,
880             s_C_sel_lv, s_C_sel_cpp, s_C_sel_tos, s_C_sel_opc,
881             s_C_sel_h, s_C_sel_iod, s_C_sel_ioa : ns(1);
882         sig s_od_mux91 : tc(32);
883         sig s_od_alu : tc(32);
884         sig s_ic_alu : ns(6);
885         sig s_od_shifter : tc(32);
886         sig s_ic_sl, s_ic_sr : ns(1);
887         sig s_ic_mux91 : ns(4);
888         sig s_ctr_rd, s_ctr_wr, s_ctr_fetch :ns(1);
889         sig s_ctr_mbr : ns(8);
890         sig s_alu_status_N, s_alu_status_Z : ns(1);
891         sig s_alu_fct : ns(8);
892             sig s_alu_status : ns(2);
893         sig s_rom_adr : ns(9);
894         sig s_rom_rd : ns(1);
895         sig s_rom_in : ns(36);
896
897         use MAR(s_od_shifter, s_od_mar, s_C_sel_mar);
898         use MDR(mdr_in, s_od_shifter, s_C_sel_mdr, mdr_ld, s_od_mdr, mdr_out);
899         use PC(s_od_shifter, s_od_pc, pc_out, s_C_sel_pc);
900         use MBR(mbr_in, mbr_ld, s_od_mbr_sg, s_od_mbr_un, s_ctr_mbr);
901         use SP(s_od_shifter, s_od_sp, s_C_sel_sp);
902         use LV(s_od_shifter, s_od_lv, s_C_sel_lv);
903         use CPP(s_od_shifter, s_od_cpp, s_C_sel_cpp);
904         use TOS(s_od_shifter, s_od_tos, s_C_sel_tos);
905         use OPC(s_od_shifter, s_od_opc, s_C_sel_opc);
906         use H(s_od_shifter, s_od_h, s_C_sel_h);
```

```
907         use IOD(iod_in, s_od_shifter, s_C_sel_iod, iod_ld, s_od_iod);
908         use IOA(ioa_in, s_od_shifter, s_C_sel_ioa, ioa_ld, s_od_ioa);
909         use ALU(s_od_h, s_od_mux91, s_od_alu, s_ic_alu, s_alu_status);
910         use Shifter(s_od_alu, s_od_shifter, s_ic_sl, s_ic_sr);
911         use BusB(s_od_mdr, s_od_pc, s_od_mbr_sg, s_od_mbr_un, s_od_sp,
912             s_od_lv, s_od_cpp, s_od_tos, s_od_opc, s_od_iod, s_od_ioa,
913             s_od_mux91, s_ic_mux91);
914
915         sfg exec{
916             mar_out = s_od_mar;
917             mbr_out = s_ctr_mbr;
918             iod_out = s_od_iod;
919             ioa_out = s_od_ioa;
920             status_N = s_alu_status[1];
921             status_Z = s_alu_status[0];
922
923             s_C_sel_mar = C_sel[0];
924             s_C_sel_mdr = C_sel[1];
925             s_C_sel_pc = C_sel[2];
926             s_C_sel_sp = C_sel[3];
927             s_C_sel_lv = C_sel[4];
928             s_C_sel_cpp = C_sel[5];
929             s_C_sel_tos = C_sel[6];
930             s_C_sel_opc = C_sel[7];
931             s_C_sel_h = C_sel[8];
932             s_C_sel_iod = C_sel[9];
933             s_C_sel_ioa = C_sel[10];
934
935             s_ic_mux91 = B_sel;
936             s_ic_alu = alu_fct[5:0];
937             s_ic_sr = alu_fct[6];
938             s_ic_sl = alu_fct[7];
939
940     //        $display("Datapath: iod_out: ", iod_out);
941     //        $display("Datapath: ioa_out: ", ioa_out);
942     //        $display("Datapath: iod_in: ", iod_in);
943     //        $display("Datapath: ioa_in: ", ioa_in);
944     //        $display("Datapath: mdr_out: ", mdr_out);
945         }
946         }
947     hardwired ctrl(datapath) {exec;}
948     //-------------------------------------------------------------
949     // End of datapath
950     //-------------------------------------------------------------
951
952     //-----------------------------------------------------
953     // Begining of GCD
954     //-----------------------------------------------------
955
956     dp gcd_controller(
957         in ic_req : ns(1);
958         out oc_ack : ns(1);
959         in  ic_status  : ns(2);
960         out oc_command : ns(10)) {
961
962         reg stat :ns(2);
963
964
965         sfg setInit{//req = 0;
966                     oc_ack = 0;
967                     oc_command = 0b0101000000;
968                    }
969
970         // idle state
971         sfg idle {
972
973             oc_command = 0b0101000000; oc_ack = 0;
974     //        $display("gcd_ctrl sfg:",$sfg);
975     //        $display("gcd ctrl: req sig = ",ic_req);
976          //   $display("gcd ctrl, req reg = ",req);
977
978             }
979
980         // Input X in regB
981         sfg i1 {oc_command = 0b0000000110; oc_ack = 0;
982
983     //        $display("gcd_ctrl sfg:",$sfg);
984     //        $display("gcd ctrl: req sig = ",ic_req);
985           //  $display("gcd ctrl, req reg = ",req);
986
987             }
988
989         // Input Y in regB while moving X to regA
990         sfg i2 {oc_command = 0b0101001110; oc_ack = 0;
```

```
991   //           $display("gcd_ctrl sfg:",$sfg);
992   //           $display("gcd ctrl: req sig = ",ic_req);
993
994
995              }
996
997   //regB - regA, don't save to register
998   //only check sign of result.
999       sfg a0 {oc_command = 0b1111110000; oc_ack = 0;
1000             stat = ic_status;
1001  //           $display("gcd_ctrl sfg:",$sfg);
1002  //           $display("gcd ctrl: req sig = ",ic_req);
1003
1004             }
1005
1006      // when result is positive
1007      // Calculate regB - regA and write to regB
1008      sfg a1 {oc_command = 0b1111110101; oc_ack = 0;
1009  //           $display("gcd_ctrl sfg:",$sfg);
1010  //           $display("gcd ctrl: req sig = ",ic_req);
1011    //         $display("gcd ctrl, req reg = ",req);
1012             }
1013
1014      // Display result
1015      sfg a2 {//oc_command = 0b0101001110;
1016             oc_ack = 0;
1017  //           $display("gcd_ctrl sfg:",$sfg);
1018  //           $display("gcd ctrl: req sig = ",ic_req);
1019
1020             }
1021
1022      // Negate regA
1023      sfg a3 {oc_command = 0b1110111000; oc_ack = 0;
1024  //           $display("gcd_ctrl sfg:",$sfg);
1025  //           $display("gcd ctrl: req sig = ",ic_req);
1026
1027             }
1028
1029      // when result is negative
1030      // Calculate regB - regA and write to regA
1031      sfg a4 {oc_command = 0b1111111000; oc_ack =0;
1032  //           $display("gcd_ctrl sfg:",$sfg);
1033  //           $display("gcd ctrl: req sig = ",ic_req);
1034
1035             }
1036
1037      sfg finish {oc_command = 0b0101000000;
1038                  // req = ic_req;
1039                  oc_ack = 0; //$finish;
1040  //              $display("gcd_ctrl sfg:",$sfg);
1041  //              $display("gcd ctrl: req sig = ",ic_req);
1042        //        $display("gcd ctrl, req reg = ",req);
1043                }
1044
1045      sfg x0 {oc_command = 0b0101000000; oc_ack = 1;}// $display("sfg: ", $sfg);}// req = ic_req;}
1046
1047      sfg x1 {oc_command = 0b0101000000; oc_ack = 0;}// req = ic_req;}
1048
1049
1050  } fsm test(gcd_controller) {
1051      initial init;
1052      state idle, s0,s1, s2, s3, ack0, ack1, s_end;
1053
1054      @init (setInit) -> idle;
1055      @idle if (ic_req == 1) then (i1) -> s0;
1056            else (idle) -> idle;
1057      @s0 (i2) -> s1;
1058      @s1 (a0) -> s2;
1059      @s2 if (stat[0] == 1) then (finish) -> s_end;
1060          else if (stat[1] == 1) then (a4) -> s3;
1061          else (a1) -> s1;
1062      @s3 (a3) -> s1;
1063      @s_end (x0) -> ack0;
1064      @ack0 if (ic_req == 1) then (x0) -> ack0;
1065            else (x1) -> ack1;
1066      @ack1 (x1) -> idle;
1067    }
1068
1069
1070  // GCD ALU
1071  // Implements the ALU from fig. 4.2, page 206 in the book of Tannenbaum
1072  dp gcd_alu(
1073      in  id_x, id_y : tc(32);
1074      out od_z       : tc(32);
```

```
1075        in  ic_sel     : ns(6);
1076        out oc_stat    : ns(2)
1077      ) {
1078      sig s_N, s_Z : ns(1);
1079
1080      sfg exec {
1081
1082    //        $display("----------------------------");
1083    //        $display("CYCLE: ", $cycle);
1084
1085        od_z = (ic_sel==0b011000)? id_x :
1086               (ic_sel==0b010100)? id_y :
1087               (ic_sel==0b011010)? ~id_x :
1088               (ic_sel==0b101100)? ~id_y :
1089               (ic_sel==0b111100)? id_x + id_y :
1090               (ic_sel==0b111101)? id_x + id_y + 1 :
1091               (ic_sel==0b111001)? id_x + 1 :
1092               (ic_sel==0b110101)? id_y + 1 :
1093               (ic_sel==0b111111)? id_y - id_x :
1094               (ic_sel==0b110110)? id_y - 1 :
1095               (ic_sel==0b111011)? -id_x :
1096               (ic_sel==0b001100)? id_x & id_y :
1097               (ic_sel==0b011100)? id_x | id_y :
1098               (ic_sel==0b010000)? 0 :
1099               (ic_sel==0b010001)? 1 :
1100               (ic_sel==0b010010)? -1 :
1101               0 ;
1102        s_N = (od_z < 0);  // flag N
1103        s_Z = (od_z == 0); // falg Z
1104        oc_stat = s_N#s_Z;
1105      }
1106    }
1107    hardwired ctrl(gcd_alu) {exec;}
1108
1109    // Register
1110    // If the load (ld) signal is high, the input is sampled
1111    dp gcd_register(
1112        in  id_x  : tc(32);
1113        out od_q  : tc(32);
1114        in  ic_ld : ns(1)
1115      ) {
1116
1117      // Local state
1118      reg rx : tc(32);
1119
1120      // Actions
1121      sfg exec {
1122        rx = ic_ld ? id_x : rx;
1123        od_q = rx;
1124      }
1125    }
1126    hardwired ctrl(gcd_register) {exec;}
1127
1128    // MUX 2:1
1129    // Implements a multiplexer which selects one of two input signals
1130    // to be passed through to the output
1131    dp gcd_mux21(
1132        in id_x1, id_x2 : tc(32);
1133        out od_z : tc(32);
1134        in ic_sel : ns(1)
1135      ) {
1136
1137      sfg exec {
1138        od_z = (ic_sel==0) ? id_x1 : id_x2;
1139      }
1140    }
1141    hardwired ctrl(gcd_mux21) {exec;}
1142
1143    // DEMUX 1:2
1144    // Implements a demultiplexer which passes the input to one
1145    // of two output signals. The output not selected gets the value
1146    // zero
1147    dp gcd_demux12(
1148        in  id_x         : tc(32);
1149        out od_z1, od_z2 : tc(32);
1150        in  ic_sel       : ns(1)
1151      ) {
1152
1153      sfg exec {
1154        od_z1 = (ic_sel==0) ? id_x : 0;
1155        od_z2 = (ic_sel==0) ? 0 : id_x;
1156      }
1157    }
1158    hardwired ctrl(gcd_demux12) {exec;}
```

```
1159
1160
1161  // DATAPATH
1162  // Implements the datapath of the simple computer
1163  // The datapath is controlled by a command of 10-bits,
1164  //
1165  //   bit | description
1166  // ------+--------------------------------
1167  //   9-4 | controls the ALU (fig.4-2, 206)
1168  //     3 | load on regA
1169  //     2 | load on regB
1170  //     1 | mux select 0=left, 1=right
1171  //     0 | demux slect 0=left, 1=right
1172  // ------+--------------------------------
1173  //
1174  dp gcd_regA : gcd_register
1175  dp gcd_regB : gcd_register
1176  dp gcd_mux1 : gcd_mux21
1177  dp gcd_demux1 : gcd_demux12
1178  dp gcd_alu1 : gcd_alu
1179
1180  dp gcd_datapath(
1181      in  id_x : tc(32);
1182      out od_z : tc(32);
1183      in  ic_command : ns(10);
1184      out oc_status : ns(2)
1185    ) {
1186
1187    sig s_x, s_y : tc(32);
1188
1189    // Internal wiring
1190    sig s_1, s_2, s_3, s_4, s_regBout, s_regAout, s_alu_out : tc(32);
1191    sig s_9 : ns(2);
1192    sig s_alu : ns(6);
1193    sig s_rega, s_regb, s_mux, s_demux : ns(1);
1194
1195    use gcd_regA(s_3, s_regAout, s_rega);
1196    use gcd_regB(s_4, s_regBout, s_regb);
1197    use gcd_alu1(s_regAout, s_regBout, s_alu_out, s_alu, s_9);
1198    use gcd_mux1(s_2, s_1, s_4, s_mux);
1199    use gcd_demux1(s_alu_out, s_3, s_2, s_demux);
1200
1201    sfg exec {
1202      s_1 = id_x;
1203      s_alu = ic_command[9:4];
1204      s_rega = ic_command[3];
1205      s_regb = ic_command[2];
1206      s_mux = ic_command[1];
1207      s_demux = ic_command[0];
1208      od_z = s_alu_out;
1209      oc_status = s_9;
1210  //      $display("gcd_dp: alu out = ",$dec,s_3);
1211  //      $display("gcd_dp: regA ind = ",$dec,s_3);
1212  //      $display("gcd_dp: regA ud = ",$dec,s_regAout);
1213  //      $display("gcd_dp: regB ind = ",$dec,s_4);
1214  //      $display("gcd_dp: regB ud = ",$dec,s_regBout);
1215    }
1216  }
1217  hardwired ctrl(gcd_datapath) {exec;}
1218
1219
1220  dp bufReg : gcd_register
1221
1222
1223  //The wrapper datapath to connect and control the gcd system.
1224  dp gcd_system(
1225      in addr_to_device : tc(32);
1226      in data_to_device : tc(32);
1227
1228      out addr_from_device : tc(32);
1229      out data_from_device : tc(32);
1230
1231      out iod_ld : ns(1);
1232      out ioa_ld : ns(1);
1233
1234      in iord : ns(1)
1235  )
1236
1237  {
1238
1239
1240  sig inToGCDdp, outFromGCDdp : tc(32);
1241  sig s_req, s_ack, s_buf_ld: ns(1);
1242  sig s_dp_command : ns(10); //control signal from gcd_controller to gcd_datapath
```

```
1243  sig s_dp_status : ns(2); //status signal from gcd_datapath to gcd_controller
1244  sig s_out_data : tc(32); // data signal from gcd_datapath to data output port
1245  sig s_in_data : tc(32); //data signal from register to gcd_datapath
1246
1247  use gcd_controller(s_req, s_ack, s_dp_status, s_dp_command);
1248  use gcd_datapath(s_in_data, data_from_device, s_dp_command, s_dp_status);
1249  use bufReg(data_to_device, s_in_data, s_buf_ld);
1250
1251  reg gcdRun : ns(1);
1252
1253  sfg setInit {gcdRun = 0;
1254               iod_ld = 0;
1255               ioa_ld = 0;
1256               s_buf_ld = 0;
1257               addr_from_device = 0;
1258               s_req = gcdRun;
1259              }
1260
1261  sfg setIdle{addr_from_device = 0;
1262  //          data_from_device = 0;
1263           iod_ld = 0;
1264           ioa_ld = 0;
1265           s_req = gcdRun; s_buf_ld=0;
1266           gcdRun = s_ack;
1267  //          $display("gcdSystem, sfg: ",$sfg);
1268  //          $display("gcdSystem, addr_to_device = ",$dec, addr_to_device);
1269  //          $display("gcdSystem, register X in = ",$dec, data_to_device);
1270  //          $display("gcdSystem, register X out = ",$dec, s_in_data);
1271  //          $display("gcdSystem, req signal = ", s_req);
1272                      }
1273
1274  sfg read{addr_from_device = 0;
1275           iod_ld = 1;
1276           ioa_ld = 1;
1277           s_req = gcdRun; s_buf_ld=0;
1278  //          $display("gcdSystem, sfg: ",$sfg);
1279  //          $display("gcdSystem, addr_to_device = ",$dec, addr_to_device);
1280                    }
1281
1282  sfg writeX{s_buf_ld = 1;
1283             addr_from_device = 0;
1284    //         data_from_device = 0;
1285             iod_ld = 1;
1286             ioa_ld = 1;
1287             s_req = gcdRun;
1288  //           $display("gcdSystem, sfg: ",$sfg);
1289  //           $display("gcdSystem, addr_to_device = ",$dec, addr_to_device);
1290  //           $display("gcdSystem, register X in = ",$dec, data_to_device);
1291  //           $display("gcdSystem, register X out = ",$dec, s_in_data);
1292  //           $display("gcdSystem, req signal = ", s_req);
1293
1294                      }
1295
1296  sfg writeY{s_buf_ld = 1;
1297             addr_from_device = 0;
1298      //       data_from_device = 0;
1299             iod_ld = 0;
1300             ioa_ld = 0;
1301  //           $display("gcdSystem, sfg: ",$sfg);
1302  //           $display("gcdSystem, addr_to_device = ",$dec, addr_to_device);
1303  //           $display("gcdSystem, register X in = ",$dec, data_to_device);
1304  //           $display("gcdSystem, register X out = ",$dec, s_in_data);
1305  //           $display("gcdSystem, req signal = ", s_req);
1306            }
1307
1308  sfg setupGCDrun {gcdRun = 1;
1309                   s_buf_ld = 0;
1310                   s_req = gcdRun;
1311                   addr_from_device = 0;
1312      //       data_from_device = 0;
1313                   iod_ld = 0;
1314                   ioa_ld = 0;
1315                   }
1316
1317
1318  sfg inputX{s_buf_ld = 1; //load Y value into buffer
1319             s_req = gcdRun;
1320             addr_from_device = 0;
1321      //       data_from_device = 0;
1322             iod_ld = 0;
1323             ioa_ld = 0;
1324  //           $display("gcdSystem, sfg: ",$sfg);
1325  //           $display("gcdSystem, addr_to_device = ",$dec, addr_to_device);
1326  //           $display("gcdSystem, register X in = ",$dec, data_to_device);
```

```
1327  //           $display("gcdSystem, register X out = ",$dec, s_in_data);
1328  //           $display("gcdSystem, req signal = ", s_req);
1329            }
1330
1331  sfg inputY{s_req = 1; s_buf_ld = 0;
1332             addr_from_device = 0;
1333  //         data_from_device = 0;
1334             iod_ld = 0;
1335             ioa_ld = 0;
1336  //           $display("gcdSystem, sfg: ",$sfg);
1337  //           $display("gcdSystem, addr_to_device = ",$dec, addr_to_device);
1338  //           $display("gcdSystem, register X in = ",$dec, data_to_device);
1339  //           $display("gcdSystem, register X out = ",$dec, s_in_data);
1340             }
1341
1342  } fsm ioCtrl(gcd_system){
1343        initial init;
1344        state idle,writtenX, startGCD, setupGCD, runningGCD, stoppingGCD;
1345
1346        @init (setInit) -> idle;
1347        @idle if (addr_to_device != 0 & iord ==1) then (read) -> idle;
1348              else if (addr_to_device != 0 & iord == 0) then (writeX) -> writtenX;
1349              else (setIdle) -> idle;
1350        @writtenX if (addr_to_device != 0 & iord ==1) then (read) -> writtenX;
1351              else if (addr_to_device != 0 & iord == 0) then (setupGCDrun) -> setupGCD;
1352              else (setIdle) -> writtenX;
1353        @setupGCD (inputX) -> startGCD;
1354        @startGCD (inputY) -> runningGCD;
1355        @runningGCD if (s_ack == 0) then (setIdle) -> runningGCD;
1356              else (setIdle) -> stoppingGCD;
1357        @stoppingGCD (read) -> idle;
1358  }
1359
1360
1361  //-----------------------------------
1362  // End of GCD
1363  //-----------------------------------
1364
1365  //-------------------------------------------------------------
1366  // System
1367  //-------------------------------------------------------------
1368  system micIO_system {
1369
1370      controller(ctr_rd, ctr_wr, ctr_fetch, mbr_out, status_N, status_Z,
1371                 B_sel, alu_fct, C_sel, iord);
1372
1373      memory_1(ctr_rd, ctr_wr, ctr_fetch, mdr_in, mdr_out, mdr_ld, mbr_in,
1374               mbr_ld, pc_out, mar_out);
1375
1376      datapath(mdr_in, mdr_out, mdr_ld, pc_out, mbr_in, mbr_ld, mar_out,
1377               mbr_out, status_N, status_Z, B_sel, alu_fct, C_sel,
1378           iod_out, ioa_out, iod_in, ioa_in, iod_ld, ioa_ld);
1379
1380      daisy_chain_block(iord, ioa_out, iod_out, ioa_in, iod_in, iod_ld,
1381               ioa_ld, ioa_out2, iod_out2, ioa_in2, iod_in2, iod_ld2,
1382               ioa_ld2, a2d, d2d, afd, dfd, iodfd, ioafd, diord);
1383
1384      daisy_chain_terminator(ioa_out2, iod_out2, ioa_in2, iod_in2, iod_ld2,
1385                             ioa_ld2);
1386      gcd_system(a2d, d2d, afd, dfd, iodfd, ioafd, diord);
1387  //    lcd(a2d, d2d, diord, afd, dfd, ioafd, iodfd);
1388
1389  }
1390  //-------------------------------------------------------------
1391  // End of System
1392  //-------------------------------------------------------------
```

APPENDIX E

Test data paths and programs

E.1 Terminator test

```
1  //-----------------------------------------------
2  // Daisy chain terminator
3  //-----------------------------------------------
4  dp daisy_chain_terminator(
5      in address_from_micIO : tc(32);
6      in data_from_micIO : tc(32);
7
8      out address_to_micIO : tc(32);
9      out data_to_micIO : tc(32);
10
11     out iod_ld : ns(1);
12     out ioa_ld : ns(1)
13     ) {
14     sfg exec {
15     $display("==============================");
16     $display("Terminator: Cycle: ",$cycle);
17     $display("Terminator: Address from micIO: ", $hex, address_from_micIO);
18         $display("Terminator: Data from micIO: ", $hex, data_from_micIO);
19         $display("Terminator: Address to micIO: ", $hex, address_to_micIO);
20         $display("Terminator: Data to micIO: ", $hex, data_to_micIO);
21
22
23         $display("Terminator: Data to micIO: ", $dec, data_to_micIO);
24
25         address_to_micIO = 0;
26         data_to_micIO = (address_from_micIO != 0) ? 0xfeedbeef : 0;
27         iod_ld = (address_from_micIO != 0) ? 1 : 0;
28         ioa_ld = (address_from_micIO != 0) ? 1 : 0;
29     }
30     }
31 hardwired ctrl(daisy_chain_terminator) {exec;}
32 //-----------------------------------------------
33 // End of Daisy chain terminator
34 //-----------------------------------------------
35
36 dp termiTB(
37     out address_to_termi : tc(32);
38     out data_to_termi : tc(32);
39
40     in address_from_termi : tc(32);
41     in data_from_termi : tc(32);
42
43     in iod_ld : ns(1);
44     in ioa_ld : ns(1)
45  ){
46
```

```
47  reg dataIn, addressIn : tc(32);
48  reg ioaIn, iodIn : ns(1);
49
50  sfg init{
51  address_to_termi = 0;
52  data_to_termi = 0;
53
54  //stores answer from terminator
55  dataIn = data_from_termi;
56  addressIn = address_from_termi;
57  iodIn = iod_ld;
58  ioaIn = ioa_ld;
59
60  $display("=====TB:   sfg: ",$sfg,"   ======");
61  $display("TB: addressIn: ",addressIn);
62  $display("TB: dataIn: ",dataIn);
63  $display("TB: address_from_termi: ",address_from_termi);
64  $display("TB: data_from_termi: ",data_from_termi);
65  $display("TB: iod_ld: ",iod_ld);
66  $display("TB: ioa_ld: ",ioa_ld);
67  }
68
69  sfg sendIN {
70
71  //Send test vector to terminator
72  address_to_termi = 0x25;
73  data_to_termi = 0x00;
74
75  //stores answer from terminator
76  dataIn = data_from_termi;
77  addressIn = address_from_termi;
78  iodIn = iod_ld;
79  ioaIn = ioa_ld;
80
81
82  $display("=====TB:   sfg: ",$sfg," ========");
83  $display("TB: addressIn: ",addressIn);
84  $display("TB: dataIn: ",dataIn);
85  $display("TB: address_from_termi: ",address_from_termi);
86  $display("TB: data_from_termi: ",data_from_termi);
87  $display("TB: iod_ld: ",iod_ld);
88  $display("TB: ioa_ld: ",ioa_ld);
89  }
90
91  sfg sendOUT {
92
93  //Send test vector to terminator
94  address_to_termi = 0x25;
95  data_to_termi = 0x25;
96
97  //stores answer from terminator
98  dataIn = data_from_termi;
99  addressIn = address_from_termi;
100 iodIn = iod_ld;
101 ioaIn = ioa_ld;
102
103
104 $display("=====TB:   sfg: ",$sfg," ========");
105 $display("TB: addressIn: ",addressIn);
106 $display("TB: dataIn: ",dataIn);
107 $display("TB: address_from_termi: ",address_from_termi);
108 $display("TB: data_from_termi: ",data_from_termi);
109 $display("TB: iod_ld: ",iod_ld);
110 $display("TB: ioa_ld: ",ioa_ld);
111 }
112
113 sfg receive{
114 address_to_termi = 0;
115 data_to_termi = 0;
116
117 //stores answer from terminator
118 dataIn = data_from_termi;
119 addressIn = address_from_termi;
120 iodIn = iod_ld;
121 ioaIn = ioa_ld;
122
123 $display("====TB:   sfg: ",$sfg," ======");
124 $display("TB: addressIn: ",addressIn);
125 $display("TB: dataIn: ",dataIn);
126 $display("TB: address_from_termi: ",address_from_termi);
127 $display("TB: data_from_termi: ",data_from_termi);
128 $display("TB: iod_ld: ",iod_ld);
129 $display("TB: ioa_ld: ",ioa_ld);
130 }
```

```
131
132 sfg success{
133
134 address_to_termi = 0;
135 data_to_termi = 0;
136
137     $display("*****************************");
138     $display("Terminator test succeeded!");
139     $display("*****************************");
140     $finish;
141 }
142
143 sfg fail{
144
145 address_to_termi = 0;
146 data_to_termi = 0;
147
148     $display("*****************************");
149     $display("Terminator test failed!");
150     $display("*****************************");
151     $finish;
152 }
153
154 }
155 fsm testTerm(termiTB) {
156    initial s0;
157    state s1,s2,s3,s4,s5,s6,s7,s8,s9, error, finish;
158
159    @s0  (init) -> s1;
160    @s1 if(addressIn == 0 & dataIn == 0 & ioaIn == 0 & iodIn == 0 )
161        then (sendIN) -> s2;
162        else (init) -> error;
163    @s2 if(dataIn == (tc(32)) 0xfeedbeef & addressIn == 0 & ioaIn == 1 & iodIn == 1)
164        then (init) -> s3;
165        else (init) -> error;
166    @s3 if (addressIn == 0 & dataIn == 0 & ioaIn == 0 & iodIn == 0)
167        then (sendIN) -> s4;
168        else (fail) -> error;
169    @s4 if(dataIn == (tc(32)) 0xfeedbeef & addressIn == 0 & ioaIn == 1 & iodIn == 1)
170        then (init) -> s5;
171        else (init) -> error;
172    @s5 if(addressIn == 0 & dataIn == 0 & ioaIn == 0 & iodIn == 0)
173        then (sendOUT) -> s6;
174        else (init) -> error;
175    @s6 if(dataIn == (tc(32)) 0xfeedbeef & addressIn == 0 & ioaIn == 1 & iodIn == 1)
176        then (init) -> s7;
177        else (init) -> error;
178    @s7 if(addressIn == 0 & dataIn == 0 & ioaIn == 0 & iodIn == 0)
179        then (sendOUT) -> s8;
180        else (init) -> error;
181    @s8 if(dataIn == (tc(32)) 0xfeedbeef & addressIn == 0 & ioaIn == 1 & iodIn == 1)
182        then (init) -> s9;
183        else (init) -> error;
184    @s9 if (addressIn == 0 & dataIn == 0 & ioaIn == 0 & iodIn == 0)
185        then (success) -> finish;
186        else (fail) -> error;
187
188    @error (fail) -> error;
189    @finish (success) -> finish;
190 }
191
192 system S{
193
194 daisy_chain_terminator(tb_address_term, tb_data_term, term_address_tb, term_data_tb,
195     term_iod_tb, term_ioa_tb );
196
197 termiTB(tb_address_term, tb_data_term, term_address_tb, term_data_tb, term_iod_tb,
198     term_ioa_tb ) ;
199
200 }
```

E.2 Daisy_chain_block test

```
1 //-----------------------------------------------
2 // Daisy chain block
3 //-----------------------------------------------
4 dp daisy_chain_block(
5
6     in iord : ns(1);
7
```

```
8       in adress_from_mic2 : tc(32);
9       in data_from_mic2 : tc(32);
10
11      out adress_to_mic2 : tc(32);
12      out data_to_mic2 : tc(32);
13
14      out iod_ld_to_mic2 : ns(1);
15      out ioa_ld_to_mic2 : ns(1);
16
17      out adress_to_terminator : tc(32);
18      out data_to_terminator : tc(32);
19
20      in adress_from_terminator : tc(32);
21      in data_from_terminator : tc(32);
22
23      in iod_ld_from_terminator : ns(1);
24      in ioa_ld_from_terminator : ns(1);
25
26      out adress_to_device : tc(32);
27      out data_to_device : tc(32);
28
29      in adress_from_device : tc(32);
30      in data_from_device : tc(32);
31
32      in iod_ld_from_device : ns(1);
33      in ioa_ld_from_device : ns(1);
34
35      out device_iord : ns(1)
36      ) {
37      reg lower_bound, upper_bound : tc(32);
38
39      sfg init {
40                  device_iord = iord;
41                  adress_to_mic2 = 0;
42                  data_to_mic2 = 0;
43                  iod_ld_to_mic2 = 0;
44                  ioa_ld_to_mic2 = 0;
45                  adress_to_terminator = 0;
46                  data_to_terminator = 0;
47                  adress_to_device = 0;
48                  data_to_device = 0;
49                  // gives this DCB address range 1 to 4.
50                  lower_bound = 2;
51                  upper_bound = 5;
52              }
53
54      sfg exec {
55
56          $display("----------------------------");
57          $display("CYCLE: ", $cycle);
58  //        $display("Daisy chain: Address from mic2: ", $hex, adress_from_mic2);
59          $display("Daisy chain: Data from mic2: ", $hex, data_from_mic2);
60  //        $display("Daisy chain: Address to mic2: ", $hex, adress_to_mic2);
61  //        $display("Daisy chain: Data to mic2: ", $hex, data_to_mic2);
62  //          $display("Daisy chain: Address to device: ", $hex, adress_to_device);
63  //          $display("Daisy chain: Address to terminator: ", $hex, adress_to_terminator);
64  //        $display("Daisy chain: IOD load to cpu: ", iod_ld_to_mic2);
65  //        $display("Daisy chain: IOA load to cpu: ", ioa_ld_to_mic2);
66
67
68          adress_to_terminator = (adress_from_mic2 > upper_bound) ?
69                                  adress_from_mic2 :
70                      (adress_from_mic2 < lower_bound) ?
71                      adress_from_mic2 : 0;
72          data_to_terminator = (adress_from_mic2 > upper_bound) ?
73                                 data_from_mic2 :
74                   (adress_from_mic2 < lower_bound) ?
75                   data_from_mic2 : 0;
76
77  //The address sent to the device is normalized which means that the address will
78  //lay in the range between 1 and the number of addresses used by the device.
79  //The reason for this is that device then doesn't have to know which address space
80  //it is assigned to.
81
82          adress_to_device = (adress_from_mic2 > upper_bound) ?
83                          0 : (adress_from_mic2 < lower_bound) ?
84                        0 : adress_from_mic2 - (lower_bound - 1);
85          data_to_device = (adress_from_mic2 > upper_bound) ?
86                   0 :
87                   (adress_from_mic2 < lower_bound) ?
88                   0 : data_from_mic2;
89          adress_to_mic2 = (adress_from_mic2 > upper_bound) ?
90                           adress_from_terminator :
91                   (adress_from_mic2 < lower_bound) ?
```

```
92                  adress_from_terminator : adress_from_device;
93          data_to_mic2 = (adress_from_mic2 > upper_bound) ?
94                 data_from_terminator :
95                 (adress_from_mic2 < lower_bound) ?
96                 data_from_terminator : data_from_device;
97          iod_ld_to_mic2 = iod_ld_from_terminator | iod_ld_from_device;
98          ioa_ld_to_mic2 = ioa_ld_from_terminator | ioa_ld_from_device;
99          device_iord = iord;
100     }
101     }
102     fsm ctrl(daisy_chain_block) {
103     initial s0;
104     state s1;
105
106     @s0 (init) -> s1;
107     @s1 (exec) -> s1;
108 }
109
110 //-----------------------------------------------
111 // End of Daisy chain block
112 //-----------------------------------------------
113
114 //-----------------------------------------------
115 // Daisy chain block test bench
116 //-----------------------------------------------
117
118 dp DCBtestbench(
119
120     out iord : ns(1);
121
122     out address_from_mic2 : tc(32);
123     out data_from_mic2 : tc(32);
124
125     in address_to_mic2 : tc(32);
126     in data_to_mic2 : tc(32);
127
128     in iod_ld_to_mic2 : ns(1);
129     in ioa_ld_to_mic2 : ns(1);
130
131     in address_to_terminator : tc(32);
132     in data_to_terminator : tc(32);
133
134     out address_from_terminator : tc(32);
135     out data_from_terminator : tc(32);
136
137     out iod_ld_from_terminator : ns(1);
138     out ioa_ld_from_terminator : ns(1);
139
140     in address_to_device : tc(32);
141     in data_to_device : tc(32);
142
143     out address_from_device : tc(32);
144     out data_from_device : tc(32);
145
146     out iod_ld_from_device : ns(1);
147     out ioa_ld_from_device : ns(1);
148
149     in device_iord : ns(1)
150     ) {
151
152 reg addr_to_cpu, data_to_cpu, addr_to_term, data_to_term,
153 addr_to_dev, data_to_dev : ns(32);
154 reg iod_ld_to_cpu, ioa_ld_to_cpu, dev_iord : ns(1);
155
156
157 //Initial set up of signals
158 // several times because it prints out the values of the register.
159 sfg init {
160     //output ports
161     iord = 0;
162     address_from_mic2 = 0;
163     data_from_mic2 = 0;
164     address_from_terminator = 0;
165     data_from_terminator = 0;
166     iod_ld_from_terminator = 0;
167     ioa_ld_from_terminator = 0;
168     address_from_device = 0;
169     data_from_device = 0;
170     iod_ld_from_device = 0;
171     ioa_ld_from_device = 0;
172     //registers
173     addr_to_cpu = address_to_mic2;
174     $display("addr_to_cpu: ",addr_to_cpu);
175     data_to_cpu = data_to_mic2;
```

```
176         $display("data_to_cpu: ",data_to_cpu);
177         addr_to_term = address_to_terminator;
178         $display("addr_to_term: ",addr_to_term);
179         data_to_term = data_to_terminator;
180         $display("data_to_term: ",data_to_term);
181         addr_to_dev = address_to_device;
182         $display("addr_to_dev: ",addr_to_dev);
183         data_to_dev = data_to_device;
184         $display("data_to_dev: ",data_to_dev);
185         iod_ld_to_cpu = iod_ld_to_mic2;
186         $display("iod_ld_to_cpu: ",iod_ld_to_cpu);
187         ioa_ld_to_cpu = ioa_ld_to_mic2;
188         $display("ioa_ld_to_cpu: ",ioa_ld_to_cpu);
189         dev_iord = device_iord;
190         $display("dev_iord: ",dev_iord);
191
192     }
193
194     //test to see if the DCB sends only zeros while not receiving
195     //anything.
196     sfg silent{
197     $display("i silent");
198         iord = 0;
199         address_from_mic2 = 0;
200         data_from_mic2 = 0;
201         address_from_terminator = 0;
202         data_from_terminator = 0;
203         iod_ld_from_terminator = 0;
204         ioa_ld_from_terminator = 0;
205         address_from_device = 0;
206         data_from_device = 0;
207         iod_ld_from_device = 0;
208         ioa_ld_from_device = 0;
209         //registers
210         addr_to_cpu = address_to_mic2;
211         data_to_cpu = data_to_mic2;
212         addr_to_term = address_to_terminator;
213         data_to_term = data_to_terminator;
214         addr_to_dev = address_to_device;
215         data_to_dev = data_to_device;
216         iod_ld_to_cpu = iod_ld_to_mic2;
217         ioa_ld_to_cpu = ioa_ld_to_mic2;
218         dev_iord = device_iord;
219     }
220
221     //test to see if the DCB forwards the data and addresses
222     //received from the cpu correctly with the read flag set.
223     sfg forwardRD{
224     $display("i forwardRD");
225         iord = 1;
226         address_from_mic2 = 10;
227         data_from_mic2 = 110;
228         address_from_terminator = 0;
229         data_from_terminator = 0;
230         iod_ld_from_terminator = 0;
231         ioa_ld_from_terminator = 0;
232         address_from_device = 0;
233         data_from_device = 0;
234         iod_ld_from_device = 0;
235         ioa_ld_from_device = 0;
236         //registers
237         addr_to_cpu = address_to_mic2;
238         data_to_cpu = data_to_mic2;
239         addr_to_term = address_to_terminator;
240         data_to_term = data_to_terminator;
241         addr_to_dev = address_to_device;
242         data_to_dev = data_to_device;
243         iod_ld_to_cpu = iod_ld_to_mic2;
244         ioa_ld_to_cpu = ioa_ld_to_mic2;
245         dev_iord = device_iord;
246     }
247
248     // test to see if the DCB forwards the data and addresses
249     // received from the cpu correctly with the write flag set.
250     sfg forwardWR{
251     $display("i forwardWR");
252         iord = 0;
253         address_from_mic2 = 33;
254         data_from_mic2 = 52;
255         address_from_terminator = 0;
256         data_from_terminator = 0;
257         iod_ld_from_terminator = 0;
258         ioa_ld_from_terminator = 0;
259         address_from_device = 0;
```

```
260         data_from_device = 0;
261         iod_ld_from_device = 0;
262         ioa_ld_from_device = 0;
263         //registers
264         addr_to_cpu = address_to_mic2;
265         data_to_cpu = data_to_mic2;
266         addr_to_term = address_to_terminator;
267         data_to_term = data_to_terminator;
268         addr_to_dev = address_to_device;
269         data_to_dev = data_to_device;
270         iod_ld_to_cpu = iod_ld_to_mic2;
271         ioa_ld_to_cpu = ioa_ld_to_mic2;
272         dev_iord = device_iord;
273     }
274
275     //test to see if the DCB returns signals from the
276     //terminator correctly and relays the control signals.
277     sfg return{
278     $display("i return");
279         iord = 0;
280         address_from_mic2 = 17;
281         data_from_mic2 = 10;
282         address_from_terminator = 0;
283         data_from_terminator = 25;
284         iod_ld_from_terminator = 1;
285         ioa_ld_from_terminator = 1;
286         address_from_device = 0;
287         data_from_device = 0;
288         iod_ld_from_device = 0;
289         ioa_ld_from_device = 0;
290         //registers
291         addr_to_cpu = address_to_mic2;
292         data_to_cpu = data_to_mic2;
293         addr_to_term = address_to_terminator;
294         data_to_term = data_to_terminator;
295         addr_to_dev = address_to_device;
296         data_to_dev = data_to_device;
297         iod_ld_to_cpu = iod_ld_to_mic2;
298         ioa_ld_to_cpu = ioa_ld_to_mic2;
299         dev_iord = device_iord;
300     }
301
302     // test to see if the DCB forwards data and address
303     // to the connected device if the address is correct.
304     sfg catchDataWR{
305     $display("i catchDataWR");
306         iord = 0;
307         address_from_mic2 = 3;
308         data_from_mic2 = 15;
309         address_from_terminator = 0;
310         data_from_terminator = 0;
311         iod_ld_from_terminator = 0;
312         ioa_ld_from_terminator = 0;
313         address_from_device = 0;
314         data_from_device = 0;
315         iod_ld_from_device = 0;
316         ioa_ld_from_device = 0;
317         //registers
318         addr_to_cpu = address_to_mic2;
319         data_to_cpu = data_to_mic2;
320         addr_to_term = address_to_terminator;
321         data_to_term = data_to_terminator;
322         addr_to_dev = address_to_device;
323         data_to_dev = data_to_device;
324         iod_ld_to_cpu = iod_ld_to_mic2;
325         ioa_ld_to_cpu = ioa_ld_to_mic2;
326         dev_iord = device_iord;
327     }
328
329     // test to see if the DCB forwards data and address
330     // to the connected device if the address is correct
331     // (with iord high).
332     sfg catchDataRD{
333     $display("i catchDataRD");
334         iord = 1;
335         address_from_mic2 = 3;
336         data_from_mic2 = 2717;
337         address_from_terminator = 0;
338         data_from_terminator = 0;
339         iod_ld_from_terminator = 0;
340         ioa_ld_from_terminator = 0;
341         address_from_device = 0;
342         data_from_device = 0;
343         iod_ld_from_device = 0;
```

```
344         ioa_ld_from_device = 0;
345         //registers
346         addr_to_cpu = address_to_mic2;
347         data_to_cpu = data_to_mic2;
348         addr_to_term = address_to_terminator;
349         data_to_term = data_to_terminator;
350         addr_to_dev = address_to_device;
351         data_to_dev = data_to_device;
352         iod_ld_to_cpu = iod_ld_to_mic2;
353         ioa_ld_to_cpu = ioa_ld_to_mic2;
354         dev_iord = device_iord;
355     }
356
357     // test to see if the DCB forwards data and address
358     // from the connected device if anything comes from it.
359     sfg dataFromDev{
360     $display("i dataFromDev");
361         iord = 0;
362         address_from_mic2 = 3;
363         data_from_mic2 = 0;
364         address_from_terminator = 0;
365         data_from_terminator = 0;
366         iod_ld_from_terminator = 0;
367         ioa_ld_from_terminator = 0;
368         address_from_device = 0;
369         data_from_device = 3537;
370         iod_ld_from_device = 1;
371         ioa_ld_from_device = 1;
372         //registers
373         addr_to_cpu = address_to_mic2;
374         data_to_cpu = data_to_mic2;
375         addr_to_term = address_to_terminator;
376         data_to_term = data_to_terminator;
377         addr_to_dev = address_to_device;
378         data_to_dev = data_to_device;
379         iod_ld_to_cpu = iod_ld_to_mic2;
380         ioa_ld_to_cpu = ioa_ld_to_mic2;
381         dev_iord = device_iord;
382     }
383
384     // test to see if the DCB forwards data and address
385     // to the connected device if lowr bound address is
386     // used
387     sfg lowerbound{
388     $display("i lowerbound");
389         iord = 0;
390         address_from_mic2 = 2;
391         data_from_mic2 = 32;
392         address_from_terminator = 0;
393         data_from_terminator = 0;
394         iod_ld_from_terminator = 0;
395         ioa_ld_from_terminator = 0;
396         address_from_device = 0;
397         data_from_device = 0;
398         iod_ld_from_device = 0;
399         ioa_ld_from_device = 0;
400         //registers
401         addr_to_cpu = address_to_mic2;
402         data_to_cpu = data_to_mic2;
403         addr_to_term = address_to_terminator;
404         data_to_term = data_to_terminator;
405         addr_to_dev = address_to_device;
406         data_to_dev = data_to_device;
407         iod_ld_to_cpu = iod_ld_to_mic2;
408         ioa_ld_to_cpu = ioa_ld_to_mic2;
409         dev_iord = device_iord;
410     }
411
412     // test to see if the DCB forwards data and address
413     // to the connected device if the upper bound address
414     // is used
415     sfg upperbound{
416     $display("i upperbound");
417         iord = 0;
418         address_from_mic2 = 5;
419         data_from_mic2 = 64;
420         address_from_terminator = 0;
421         data_from_terminator = 0;
422         iod_ld_from_terminator = 0;
423         ioa_ld_from_terminator = 0;
424         address_from_device = 0;
425         data_from_device = 0;
426         iod_ld_from_device = 0;
427         ioa_ld_from_device = 0;
```

```
428         //registers
429         addr_to_cpu = address_to_mic2;
430         data_to_cpu = data_to_mic2;
431         addr_to_term = address_to_terminator;
432         data_to_term = data_to_terminator;
433         addr_to_dev = address_to_device;
434         data_to_dev = data_to_device;
435         iod_ld_to_cpu = iod_ld_to_mic2;
436         ioa_ld_to_cpu = ioa_ld_to_mic2;
437         dev_iord = device_iord;
438     }
439 
440     // test to see if the DCB forwards data and address
441     // to the next link in the chain if the address is under the
442     // address space assigned to it.
443     sfg underRange{
444     $display("i underRange");
445         iord = 0;
446         address_from_mic2 = 1;
447         data_from_mic2 = 9;
448         address_from_terminator = 0;
449         data_from_terminator = 0;
450         iod_ld_from_terminator = 0;
451         ioa_ld_from_terminator = 0;
452         address_from_device = 0;
453         data_from_device = 0;
454         iod_ld_from_device = 0;
455         ioa_ld_from_device = 0;
456         //registers
457         addr_to_cpu = address_to_mic2;
458         data_to_cpu = data_to_mic2;
459         addr_to_term = address_to_terminator;
460         data_to_term = data_to_terminator;
461         addr_to_dev = address_to_device;
462         data_to_dev = data_to_device;
463         iod_ld_to_cpu = iod_ld_to_mic2;
464         ioa_ld_to_cpu = ioa_ld_to_mic2;
465         dev_iord = device_iord;
466     }
467 
468     // test to see if the DCB forwards data and address
469     // to the next link in the chain if the address is over the
470     // address space assigned to it.
471     sfg overRange{
472     $display("i overRange");
473         iord = 0;
474         address_from_mic2 = 6;
475         data_from_mic2 = 8;
476         address_from_terminator = 0;
477         data_from_terminator = 0;
478         iod_ld_from_terminator = 0;
479         ioa_ld_from_terminator = 0;
480         address_from_device = 0;
481         data_from_device = 0;
482         iod_ld_from_device = 0;
483         ioa_ld_from_device = 0;
484         //registers
485         addr_to_cpu = address_to_mic2;
486         data_to_cpu = data_to_mic2;
487         addr_to_term = address_to_terminator;
488         data_to_term = data_to_terminator;
489         addr_to_dev = address_to_device;
490         data_to_dev = data_to_device;
491         iod_ld_to_cpu = iod_ld_to_mic2;
492         ioa_ld_to_cpu = ioa_ld_to_mic2;
493         dev_iord = device_iord;
494     }
495 
496     sfg success{
497 
498         $display("********************************");
499         $display("Daisy_Chain_Block test succeeded!");
500         $display("********************************");
501         $finish;
502     }
503 
504     sfg fail{
505 
506         $display("*****************************");
507         $display("Daisy_Chain_Block test failed!");
508         $display("*****************************");
509         $finish;
510     }
511 
```

```
512
513
514  } fsm testDCB(DCBtestbench){
515
516   initial s0;
517   state s1, s2, s3, s4, s5, s6, s7, s8, s9, s10,
518  s11, s12, error, finish;
519
520   @s0 (init) -> s1;
521
522   @s1 if(addr_to_cpu == 0 & data_to_cpu == 0 &
523  addr_to_term == 0 & data_to_term == 0 &
524  addr_to_dev == 0 & data_to_dev == 0 & iod_ld_to_cpu == 0 &
525  ioa_ld_to_cpu == 0 & dev_iord == 0)
526    then (silent) -> s2;
527    else (fail) -> error ;
528
529   @s2 if(addr_to_cpu == 0 & data_to_cpu == 0 &
530  addr_to_term == 0 & data_to_term == 0 & addr_to_dev == 0 &
531  data_to_dev == 0 & iod_ld_to_cpu == 0 & ioa_ld_to_cpu == 0 &
532  dev_iord == 0)
533    then (forwardRD) -> s3;
534    else (fail) -> error ;
535
536   @s3 if(addr_to_cpu == 0 & data_to_cpu == 0 &
537  addr_to_term == 10 & data_to_term == 110 & addr_to_dev == 0 &
538  data_to_dev == 0 & iod_ld_to_cpu == 0 & ioa_ld_to_cpu == 0 &
539  dev_iord == 1)
540    then (forwardWR) -> s4;
541    else (init,fail) -> error ;
542
543   @s4 if(addr_to_cpu == 0 & data_to_cpu == 0 &
544  addr_to_term == 33 & data_to_term == 52 & addr_to_dev == 0 &
545  data_to_dev == 0 & iod_ld_to_cpu == 0 & ioa_ld_to_cpu == 0 &
546  dev_iord == 0)
547    then (return) -> s5;
548    else (fail,init) -> error ;
549
550   @s5 if(addr_to_cpu == 0 & data_to_cpu == 25 &
551  addr_to_term == 17 & data_to_term == 10 & addr_to_dev == 0 &
552  data_to_dev == 0 & iod_ld_to_cpu == 1 & ioa_ld_to_cpu == 1 &
553  dev_iord == 0)
554    then (catchDataWR) -> s6;
555    else (fail,init) -> error;
556
557   @s6 if(addr_to_cpu == 0 & data_to_cpu == 0 &
558  addr_to_term == 0 & data_to_term == 0 & addr_to_dev == 2 &
559  data_to_dev == 15 & iod_ld_to_cpu == 0 & ioa_ld_to_cpu == 0 &
560  dev_iord == 0)
561    then (catchDataRD) -> s7;
562    else (fail,init) -> error;
563
564   @s7 if(addr_to_cpu == 0 & data_to_cpu == 0 &
565  addr_to_term == 0 & data_to_term == 0 & addr_to_dev == 2 &
566  data_to_dev == 2717 & iod_ld_to_cpu == 0 &
567  ioa_ld_to_cpu == 0 & dev_iord == 1)
568    then (dataFromDev) -> s8;
569    else (fail,init) -> error;
570
571   @s8 if(addr_to_cpu == 0 & data_to_cpu == 3537 &
572  addr_to_term == 0 & data_to_term == 0 & addr_to_dev == 2 &
573  data_to_dev == 0 & iod_ld_to_cpu == 1 & ioa_ld_to_cpu == 1 &
574  dev_iord == 0)
575    then (lowerbound) -> s9;
576    else (fail,init) -> error;
577
578   @s9 if(addr_to_cpu == 0 & data_to_cpu == 0 &
579  addr_to_term == 0 & data_to_term == 0 & addr_to_dev == 1 &
580  data_to_dev == 32 & iod_ld_to_cpu == 0 & ioa_ld_to_cpu == 0 &
581  dev_iord == 0)
582    then (upperbound) -> s10;
583    else (fail,init) -> error;
584
585   @s10 if(addr_to_cpu == 0 & data_to_cpu == 0 &
586  addr_to_term == 0 & data_to_term == 0 & addr_to_dev == 4 &
587  data_to_dev == 64 & iod_ld_to_cpu == 0 & ioa_ld_to_cpu == 0 &
588  dev_iord == 0)
589    then (underRange) -> s11;
590    else (fail,init) -> error;
591
592   @s11 if(addr_to_cpu == 0 & data_to_cpu == 0 &
593  addr_to_term == 1 & data_to_term == 9 & addr_to_dev == 0 &
594  data_to_dev == 0 & iod_ld_to_cpu == 0 & ioa_ld_to_cpu == 0 &
595  dev_iord == 0)
```

```
  then (overRange) -> s12;
  else (fail,init) -> error;

 @s12 if(addr_to_cpu == 0 & data_to_cpu == 0 &
addr_to_term == 6 & data_to_term == 8 & addr_to_dev == 0 &
data_to_dev == 0 & iod_ld_to_cpu == 0 & ioa_ld_to_cpu == 0 &
dev_iord == 0)
  then (success,init) -> finish;
  else (fail,init) -> error;

 @error (init) -> error;
 @finish (init,success) -> finish;
}

//-----------------------------------------------
system S{

    daisy_chain_block(iord, ioa_out, iod_out, ioa_in, iod_in, iod_ld,
              ioa_ld, ioa_out2, iod_out2, ioa_in2, iod_in2, iod_ld2,
              ioa_ld2, a2d, d2d, afd, dfd, iodfd, ioafd, diord);

    DCBtestbench(iord, ioa_out, iod_out, ioa_in, iod_in, iod_ld,
              ioa_ld, ioa_out2, iod_out2, ioa_in2, iod_in2, iod_ld2,
              ioa_ld2, a2d, d2d, afd, dfd, iodfd, ioafd, diord);

}
```

E.3 IJVM test program using GCD

```
//
// Name
//
//   ijvmtest.jas
//
// Description
//
//   This program tests the IJVM instruction set.
//
// Author
//
//   Andrew S. Tanenbaum  February 16, 1999
//
// Modification History
//
//   Ray Ontko  April 11, 1999
//   Added tests for LDC_W, ILOAD, ISTORE, INVOKEVIRTUAL and IRETURN
//   Also added OUT instructions to help identify which ERROR occurred.
//
// Notes
//
//   The INP instruction is not tested by this program.
//   The ERR instruction is not tested by this program.
//

.constant
objref  0xCAFE  // may be any value.  Needed by invokevirtual.
my_max  100
.end-constant

.main
.var
my_var
.end-var

    BIPUSH 19    // mark the bottom of the stack
    BIPUSH 20    // # iterations

L1:    BIPUSH 3
    BIPUSH 3
    IF_ICMPEQ L2
    BIPUSH 48
    BIPUSH 1
    IADD
    OUT
    BIPUSH 32
    OUT
    GOTO ERR

L2:    BIPUSH -1
    BIPUSH -1
```

```
52      IF_ICMPEQ L3
53      BIPUSH 48
54      BIPUSH 2
55      IADD
56      OUT
57      BIPUSH 32
58      OUT
59      GOTO ERR
60
61  L3:    BIPUSH 4    // start testing IADD
62      BIPUSH 7
63      IADD
64      BIPUSH 11
65      IF_ICMPEQ L4
66      BIPUSH 48
67      BIPUSH 3
68      IADD
69      OUT
70      BIPUSH 32
71      OUT
72      GOTO ERR
73
74  L4:    BIPUSH -5
75      BIPUSH -9
76      IADD
77      BIPUSH -14
78      IF_ICMPEQ L5
79      BIPUSH 48
80      BIPUSH 4
81      IADD
82      OUT
83      BIPUSH 32
84      OUT
85      GOTO ERR
86
87  L5:    BIPUSH 10    // start testing ISUB
88      BIPUSH 7
89      ISUB
90      BIPUSH 3
91      IF_ICMPEQ L6
92      BIPUSH 48
93      BIPUSH 5
94      IADD
95      OUT
96      BIPUSH 32
97      OUT
98      GOTO ERR
99
100 L6:    BIPUSH -3
101     BIPUSH -7
102     ISUB
103     BIPUSH 4
104     IF_ICMPEQ L7
105     BIPUSH 48
106     BIPUSH 6
107     IADD
108     OUT
109     BIPUSH 32
110     OUT
111     GOTO ERR
112
113 L7:    BIPUSH 0xF2    // start testing IAND
114     BIPUSH 0x31
115     IAND
116     BIPUSH 0x30
117     IF_ICMPEQ L8
118     BIPUSH 48
119     BIPUSH 7
120     IADD
121     OUT
122     BIPUSH 32
123     OUT
124     GOTO ERR
125
126 L8:    BIPUSH 0xF2    // start testing IOR
127     BIPUSH 0x31
128     IOR
129     BIPUSH 0xF3
130     IF_ICMPEQ L9
131     BIPUSH 48
132     BIPUSH 8
133     IADD
134     OUT
135     BIPUSH 32
```

```
136         OUT
137         GOTO ERR
138
139     L9:    BIPUSH 20    // start testing DUP
140         DUP
141         IADD
142         BIPUSH 40
143         IF_ICMPEQ L10
144         BIPUSH 48
145         BIPUSH 9
146         IADD
147         OUT
148         BIPUSH 32
149         OUT
150         GOTO ERR
151
152     L10:    BIPUSH 32    // start testing POP
153         BIPUSH 17
154         POP
155         BIPUSH 32
156         IF_ICMPEQ L11
157         BIPUSH 48
158         BIPUSH 1
159         IADD
160         OUT
161         BIPUSH 48
162         BIPUSH 0
163         IADD
164         OUT
165         BIPUSH 32
166         OUT
167         GOTO ERR
168
169     L11:    BIPUSH 9    // start test of IFLT
170         IFLT ERR
171         BIPUSH 0
172         IFLT ERR
173         BIPUSH -1
174         IFLT L12
175         BIPUSH 48
176         BIPUSH 1
177         IADD
178         OUT
179         BIPUSH 48
180         BIPUSH 1
181         IADD
182         OUT
183         BIPUSH 32
184         OUT
185         GOTO ERR
186
187     L12:    BIPUSH 1    // start testing IFEQ
188         IFEQ ERR
189         BIPUSH 0
190         IFEQ L13
191         BIPUSH 48
192         BIPUSH 1
193         IADD
194         OUT
195         BIPUSH 48
196         BIPUSH 2
197         IADD
198         OUT
199         BIPUSH 32
200         OUT
201         GOTO ERR
202
203     L13:    BIPUSH 16    // start testing SWAP
204         BIPUSH -5
205         SWAP
206         BIPUSH 16
207         IF_ICMPEQ L14
208         BIPUSH 48
209         BIPUSH 1
210         IADD
211         OUT
212         BIPUSH 48
213         BIPUSH 3
214         IADD
215         OUT
216         BIPUSH 32
217         OUT
218         GOTO ERR
219
```

```
220 L14:    BIPUSH -5
221         IF_ICMPEQ L15
222         BIPUSH 48
223         BIPUSH 1
224         IADD
225         OUT
226         BIPUSH 48
227         BIPUSH 4
228         IADD
229         OUT
230         BIPUSH 32
231         OUT
232         GOTO ERR
233
234 L15:    LDC_W my_max    // start testing LDC_W
235         BIPUSH 100
236         IF_ICMPEQ L16
237         BIPUSH 48
238         BIPUSH 1
239         IADD
240         OUT
241         BIPUSH 48
242         BIPUSH 5
243         IADD
244         OUT
245         BIPUSH 32
246         OUT
247         GOTO ERR
248
249 L16:    BIPUSH 83    // start testing ISTORE, IINC and ILOAD
250         ISTORE my_var
251         IINC my_var 4
252         BIPUSH 99
253         POP
254         ILOAD my_var
255         BIPUSH 87
256         IF_ICMPEQ L17
257         BIPUSH 48
258         BIPUSH 1
259         IADD
260         OUT
261         BIPUSH 48
262         BIPUSH 6
263         IADD
264         OUT
265         BIPUSH 32
266         OUT
267         GOTO ERR
268
269 L17:    LDC_W objref   // start testing INVOKEVIRTUAL and IRETURN
270         BIPUSH -1
271         BIPUSH -10
272         INVOKEVIRTUAL cmp
273         BIPUSH 1
274         IF_ICMPEQ L18
275         BIPUSH 48
276         BIPUSH 1
277         IADD
278         OUT
279         BIPUSH 48
280         BIPUSH 7
281         IADD
282         OUT
283         BIPUSH 32
284         OUT
285         GOTO ERR
286
287 L18:    LDC_W objref
288         BIPUSH -10
289         BIPUSH -1
290         INVOKEVIRTUAL cmp
291         BIPUSH -1
292         IF_ICMPEQ L19
293         BIPUSH 48
294         BIPUSH 1
295         IADD
296         OUT
297         BIPUSH 48
298         BIPUSH 8
299         IADD
300         OUT
301         BIPUSH 32
302         OUT
303         GOTO ERR
```

```
304
305 L19:    LDC_W objref
306       BIPUSH -10
307       BIPUSH -10
308       INVOKEVIRTUAL cmp
309       BIPUSH 0
310       IF_ICMPEQ L20
311       BIPUSH 48
312       BIPUSH 1
313       IADD
314       OUT
315       BIPUSH 48
316       BIPUSH 9
317       IADD
318       OUT
319       BIPUSH 32
320       OUT
321       GOTO ERR
322
323 L20:    LDC_W objref
324       BIPUSH -10
325       BIPUSH 10
326       INVOKEVIRTUAL cmp
327       BIPUSH -1
328       IF_ICMPEQ L21
329       BIPUSH 48
330       BIPUSH 2
331       IADD
332       OUT
333       BIPUSH 48
334       BIPUSH 0
335       IADD
336       OUT
337       BIPUSH 32
338       OUT
339       GOTO ERR
340
341 L21:    LDC_W objref
342       BIPUSH 10
343       BIPUSH -10
344       INVOKEVIRTUAL cmp
345       BIPUSH 1
346       IF_ICMPEQ L22
347       BIPUSH 48
348       BIPUSH 2
349       IADD
350       OUT
351       BIPUSH 48
352       BIPUSH 1
353       IADD
354       OUT
355       BIPUSH 32
356       OUT
357       GOTO ERR
358
359 L22:    LDC_W objref
360       BIPUSH 0
361       BIPUSH 0
362       INVOKEVIRTUAL cmp
363       BIPUSH 0
364       IF_ICMPEQ L23
365       BIPUSH 48
366       BIPUSH 2
367       IADD
368       OUT
369       BIPUSH 48
370       BIPUSH 2
371       IADD
372       OUT
373       BIPUSH 32
374       OUT
375       GOTO ERR
376
377 L23:    LDC_W objref
378       BIPUSH 1
379       BIPUSH 10
380       INVOKEVIRTUAL cmp
381       BIPUSH -1
382       IF_ICMPEQ L24
383       BIPUSH 48
384       BIPUSH 2
385       IADD
386       OUT
387       BIPUSH 48
```

```
388         BIPUSH 3
389         IADD
390         OUT
391         BIPUSH 32
392         OUT
393         GOTO ERR
394
395     L24:    LDC_W objref
396         BIPUSH 10
397         BIPUSH 1
398         INVOKEVIRTUAL cmp
399         BIPUSH 1
400         IF_ICMPEQ L25
401         BIPUSH 48
402         BIPUSH 2
403         IADD
404         OUT
405         BIPUSH 48
406         BIPUSH 4
407         IADD
408         OUT
409         BIPUSH 32
410         OUT
411         GOTO ERR
412
413     L25:    LDC_W objref
414         BIPUSH 10
415         BIPUSH 10
416         INVOKEVIRTUAL cmp
417         BIPUSH 0
418         IF_ICMPEQ L26
419         BIPUSH 48
420         BIPUSH 1
421         IADD
422         OUT
423         BIPUSH 48
424         BIPUSH 8
425         IADD
426         OUT
427         BIPUSH 32
428         OUT
429         GOTO ERR
430
431     L26:    NOP         // test NOP
432
433                 // iterate
434         BIPUSH 1
435         ISUB
436         DUP
437         IFEQ FINAL
438         GOTO L1
439
440     FINAL:    POP         // see if the marker is still there
441         BIPUSH 19
442         IF_ICMPEQ GCD
443         BIPUSH 48
444         BIPUSH 0
445         IADD
446         OUT
447         BIPUSH 32
448         OUT
449         GOTO ERR
450
451     GCD: //added by Esben Rugbjerg
452         BIPUSH 1
453         BIPUSH 33
454         OUT
455         POP
456         BIPUSH 1
457         BIPUSH 9
458         OUT
459         POP
460         BIPUSH 1
461         IN
462         BIPUSH 3
463         IF_ICMPEQ OK
464         GOTO ERR
465
466
467     OK:    //changed by Esben Rugbjerg
468         BIPUSH 35
469         IN
470         HALT
471     ERR://changed by Esben Rugbjerg
```

```
472         BIPUSH 40
473         IN
474         HALT
475     .end-main
476
477     // cmp returns -1 if p1 < p2, 0 if p1 = p2, and 1 if p1 > p2
478     .method cmp(p1,p2)
479     .var
480     temp
481     .end-var
482         ILOAD p1
483         ILOAD p2
484         ISUB
485         ISTORE temp
486         ILOAD temp
487         IFLT lt
488         ILOAD temp
489             IFEQ eq
490     gt:     BIPUSH 1
491              GOTO done
492     lt:     BIPUSH -1
493         GOTO done
494     eq:     BIPUSH 0
495     done:   IRETURN
496     .end-method
```

APPENDIX F

VHDL ROM generator for the Mic-1IO

F.1 Perl program to generate VHDL ROM file for Mic-1IO

```
#!/usr/bin/perl -w

my $input_file = shift @ARGV;    # Read $input_file from first
                 # program argument
my $output_file = shift @ARGV;  # Read $output_file from from second program argument

my $mem_data_string;
my $final_data_string;

my @input_data_array = `hexdump -C $input_file`; # Hexdump $input_file to
                        # @input_data_array

foreach(@input_data_array) { # Apply the following to all lines of hexdump output
    s/(^.{8})|(\s)|(\|.*\|$)//g; # Remove leading address, whitespace characters
                                 # and ASCII values from line
    $mem_data_string .=  $_; # append all processed lines to $mem_data_string
}

$mem_data_string =~ s/.{8}//; # Remove the four leading "magic" numbers from
                  # $mem_data_string

while ($mem_data_string) { # Do the following until $mem_data_string are empty
    $mem_data_string =~ s/(.{10})//; # Remove matched block of 10 characters from
                     # $mem_data_string
    $final_data_string .= "X\"$1\", "; # Append matched block of 10 characters to
                        # $mem_data_string in VHDL format
}

open OUT, ">$output_file";

printf OUT "LIBRARY IEEE;\n";
printf OUT "USE IEEE.STD_LOGIC_1164.ALL;\n";
printf OUT "USE IEEE.STD_LOGIC_ARITH.ALL;\n";
printf OUT "USE IEEE.STD_LOGIC_MISC.ALL;\n";
printf OUT "USE IEEE.STD_LOGIC_UNSIGNED.ALL;\n";
printf OUT "\n";
printf OUT "LIBRARY UNISIM;\n";
printf OUT "USE UNISIM.All;\n";
printf OUT "--use UNISIM.VPKG.all;\n";
printf OUT "\n";
printf OUT "entity mic1IO_rom is\n";
printf OUT "  port(\n";
printf OUT "    address:     in std_logic_vector(8 downto 0);\n";
```

```
44  printf OUT "    rd:             in STD_LOGIC;\n";
45  printf OUT "    odata:          out std_logic_vector (38 DOWNTO 0);\n";
46  printf OUT "    RST : in std_logic;\n";
47  printf OUT "    CLK : in std_logic\n";
48  printf OUT "  );\n";
49  printf OUT "end mic1IO_rom;\n";
50  printf OUT "\n";
51  printf OUT "architecture rtl of mic1IO_rom is\n";
52  printf OUT "  signal read_address : std_logic_vector(8 downto 0);\n";
53  printf OUT "  signal outputdata : std_logic_vector(39 downto 0);\n";
54  printf OUT "  type rom_type is array (0 to 511) of std_logic_vector(39 downto 0);\n";
55  printf OUT "  constant ROM : rom_type := (\n";
56
57
58  while ($final_data_string) { # Do the following until $final_data_string are empty
59
60      $final_data_string =~ s/((X".{10}", ){8})//; # Remove 8 blocks of VHDL data
61                           # from $final_data_string
62      my $match = $1; # Save mached blocks in match to avoid overwriting
63
64      if (!$final_data_string) { # If last VHDL data has been processed
65      $match =~ s/, $//; # Remove ", " from last line
66      }
67
68      printf OUT "    " . $match . "\n"; #Output line to OUT file descriptor
69  }
70
71
72  printf OUT " );\n";
73  printf OUT "begin\n";
74  printf OUT "  process(clk)\n";
75  printf OUT "  begin\n";
76  printf OUT "    if(clk'event and clk = '1') then\n";
77  printf OUT "      if(rd = '1') then\n";
78  printf OUT "        read_address <= address;\n";
79  printf OUT "      end if;\n";
80  printf OUT "    end if;\n";
81  printf OUT "  end process;\n";
82  printf OUT "  outputdata <= ROM(conv_integer(read_address));\n";
83  printf OUT "  odata <= outputdata(39 downto 1);\n";
84  printf OUT "\n";
85  printf OUT "end rtl;\n";
86
87  close OUT;
```

APPENDIX G

Implementation guidelines

The following pages contain the implementation guidlines for the Mic-1IO.