Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By James Michael Schmidt

Entitled TOWARDS MANY-CORE PROCESSOR SIMULATION ON CLOUD COMPUTING PLATFORMS

For the degree of Master of Science in Electrical and Computer Engineering

Is approved by the final examining committee:

Jaehwan Lee
Chair

Brian King

Mihran Tuceryan

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Jaehwan Lee

Approved by: Yaobin Chen 7/26/2011
Head of the Graduate Program Date

Graduate School Form 20
(Revised 9/10)

PURDUE UNIVERSITY
GRADUATE SCHOOL

Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation:

TOWARDS MANY-CORE PROCESSOR SIMULATION ON CLOUD COMPUTING PLATFORMS

For the degree of Master of Science in Electrical and Computer Engineering

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Executive Memorandum No. C-22,* September 6, 1991, *Policy on Integrity in Research.**

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

James Michael Schmidt

Printed Name and Signature of Candidate

7/14/2011

Date (month/day/year)

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/c_22.html

TOWARDS MANY-CORE PROCESSOR SIMULATION ON CLOUD COMPUTING PLATFORMS

A Thesis

Submitted to the Faculty

of

Purdue University

by

James M. Schmidt

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

August 2011

Purdue University

Indianapolis, Indiana

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Jaehwan John Lee for all of his support and help during my pursuit of a Masters degree. Thank you to Dr. Brian King for all his advice and Dr. Mihran Tuceryan for helping out when I needed him. I would also like to thank my wife for being a great first line editor and motivator.

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

Figure Page

GLOSSARY

SaaS	A Cloud Computing idea meaning Software as a Service.
NoC	Network-on-Chip, an approach to the communication among resources on a chip that utilizes the same ideas of computer networks. Network-on-Chips are also referred to as on-chip interconnect.
Resource	A simulated representation of any high level component in a simulated system such as a processor core, I/O, or memory.
Simulation Configuration	The symbolic design of a simulator field such as a processor core type.
Simulation Framework	Software used to gather simulation configuration information.
Simulation Designer	A student, researcher, or other indivual using the simulation framework.
Azure	Mirosoft Cloud Computing Platform.
Role	Types of resources available on Azure.
Worker Role	A Role that is used for general tasks such as long running programs or computationally intensive work.

Web Role	A Role that is commonly used as a front end performing light weight intermittent work. It uses ASP.Net and IIS 7. This allows for general web site creation.

ABSTRACT

Schmidt, James M. M.S.E.C.E., Purdue University, August 2011. Towards Many-Core Processor Simulation On Cloud Computing Platforms. Major Professor: Jaehwan J. Lee.

Growth of interest and need for many-core systems have steadily increased over the recent years. Industry trends lead many-core systems to become increasingly larger and more complex. Because of these realities it is important to researchers, academia, and industry that the design of these many-core systems be straightforward and comprehensive. There is a need for a many-core simulator that can be simple to use and learn from for students, dynamic and capable of emulating large systems for researchers, and flexible with fast turnover for industry designers. At the same time, as many-core systems have been becoming popular and complex, and hence their design, the long standing field of Cloud Computing has become more prevalent and feasible to use. Such cloud computing platforms as Windows Azure allow for the easy access and use of resources that in the past were simply not available to ordinary users. Large tasks can be performed in SaaS Cloud Computing models and be accessible from a small, lightweight device using nothing more than a web browser. As a solution to the needs for designing future many-core systems, we present a Many-Core Simulator on Azure Cloud Computing Platform called M3C Simulator. This is targeted at teaching, research, and industry and as such needs to be easy to use, flexible, and powerful. The Could Computing service model meets all these needs. This thesis discusses overall design of the M3C Simulator and how it leverages Cloud Computing resources, the simple-to-use and understand Interface layout, and the software design including program flow and dynamic compilation.

1. INTRODUCTION

1.1 Description

Many-core heterogeneous processors improve both the performance and power efficiency when considering a number of applications [1, 2]. These gains have led to the growing appreciation and adoption of heterogeneous processor designs and the movement of industry to place highly specialized processor resources physically close to or even on the same chip [3–5].

With the reemergence in recent years of Cloud Computing and the growing availability and practicality of using Cloud Computing platforms, the way teachers, programmers, and designers interact and view their fields has rapidly been changing. The way individuals such as students interact with materials such as lessons are changing. Groups do not have to buy their own hardware, maintain it, and worry as much if they are over or under utilizing it. Cloud Computing platforms such as Microsoft's Azure allow for great diversity in the types of devices and applications that can be accessed and manageable scalability of resources, allowing companies or even individuals to only use and pay for what they need.

With the rising need for growingly complex many-core heterogeneous processor design, simulator systems that can manage and correctly model these systems are important. A number of simulators have been designed that try to model many-core heterogeneous processors with mixed results. Some simulator systems' rigid design focused on a specific group or type of architecture limiting their flexibility [6–11]. Other simulators are very cumbersome to learn and operate such as ones based on specialized descriptive languages that take long periods of time to learn before any practical usage can be obtained [12–15]. Another problem with simulators of many-core heterogeneous processors is that such systems often have to model numerous

and complex stages putting a large burden on any computing system they run upon. For students trying to model a sixteen-core system, it may overtake their computing resources while larger core systems are far out of their reach. Industry or researching designers, while having access to sufficient but fixed resources, may not have easy access to resources with limited time to study or test their designs. Any simulator that means to be a practical and truly useful resource to both academic and industrial fields would have to be esay-to-use, flexible, and powerful.

To overcome the adversities presented thus far, we propose the combination of a simulation system modeled after a familiar easy-to-use and flexible many-core heterogeneous simulator and the powerful and flexible resources available through modern Cloud Computing platforms. Our combination of these ideas into a simulator system is referred to as the M3C simulator. This simulator differs in a number of respects from other many-core heterogeneous simulators. By leveraging Cloud Computing resources in a Software as a Service environment, the M3C simulator is available to a wide range of lightweight devices as well as not being limited to individuals with advanced knowledge of a narrow description language or those with access to industry resources. This thesis will familiarize readers with the design and implementation of the M3C simulator.

1.2 Objectives

The original objective of this research was loosely defined to be the porting of an old many-core heterogeneous simulator named Mhetero to a Cloud Computing platform. Mhetero was in turn developed from an XML architecture description language simulator [16].

As research progressed, the objectives of the project were expanded and over time the Mhetero simulator was the model of a new simulator which could better leverage the resources available on a Cloud Computing platform. By the end of the

research, the objective has expanded to include several tasks to be completed and characteristics to be added in the simulators.

A set of objectives for the M3C simulator includes:

1 A new many-core heterogeneous simulator that leverages the Azure Cloud Computing platform's resources was to be designed and implemented.

2 An esay-to-use interface system had to be created that would be familiar to users of the Mhetero simulator.

3 The new simulator had to be able to use simulation files that users of the old Mhetero simulator had created so that it would be easy for them to recreate their simulations. Framework files and other files such as DLLs had to be integrable as well.

4 Users needed to be able to work with the simulator using a range of computationally powerful devices.

5 The research had to be created in such a way that it would be easy to modify and understandable for future researchers who would be carrying on further research.

The research encompassed a number of focuses including networking, computer architecture, software design, and cloud computing. The purpose of this research was to allow students, researchers, and other types of users to quickly design and test a many-core heterogeneous idea. The simulator was to be powerful and flexible with the use of a Cloud Computing platform. The simulator had to not only provide a platform for teaching computer architecture concepts but also show potential for further development and eventual use in industry and research. Additionally, we wanted to show that simulators of varying types could and should be implemented in a Cloud Computing environment because of the great potential therein.

1.3 Related Work

1.3.1 An XML-Based ADL Framework for Automatic Generation of Multithreaded Computer Architecture Simulators

An XML-Based ADL Framework for Automatic Generation of Multithreaded Computer Architecture Simulators describes research that is along the lines of an Architecture Description Language or ADL [16]. In the research an XML-based ADL, its compiler, and a generation method was constructed and described. Their system allowed for automatic generation of multithreaded simulations for computer architecture ideas and concepts. By using a well-known XML language, they allowed users to easily change other structure of their simulations. The research was used in both the Mhetero and M3C simulators as a background ADL. It plays an important role in the fast turnaround when using the configurable settings.

1.3.2 A Dynamically Configurable Discrete Event Simulation Framework for Many-core Chip Multiprocessors

A Dynamically Configurable Discrete Event Simulation Framework for Many-core Chip Multiprocessors is a chapter of a book and describes the grandfather program for this simulator called the Mhetero simulator [17]. The simulator was designed to be flexible, esay-to-use, and easy to learn from and understand. It was mainly created as a teaching tool and had a great number of advantages for designing and implementing many-core heterogeneous processing systems.

1.3.3 A Distributed Parallel Simulator for Multicores

Graphite: A Distributed Parallel Simulator for Multicores uses a number of Linux machines to distribute simulations across multiple machines. This greatly speeds up the running of simulations in most cases [18]. They distributed the tasks among multiple machines in such a way that a seamless simulation is created, including

synchronization across the system. The M3C attempts a similar idea but substitutes the Cloud Computing idea in place of a group of Linux machines.

1.4 Organization of Thesis

In this thesis, we will discuss the design and characteristics of the M3C simulator. We will start with the overall design of the M3C simulator (Chapter 2). This will include the design and recreation of the many-core heterogeneous simulator and the design and parallelization of the simulator when considering the utilization of the Azure Cloud Computing platform. We split this up into five parts: the Resource structure, Modular design, Resource design connectivity, Cloud Computing structure, and the Cloud Computing connectivity. These five components describe the major aspects of both the Resource Simulation design and the Cloud Computing Simulation design. Next we will discuss the layout and interface configuration for the M3C simulator (Chapter 3). We will then move on to discuss the software design and implementation (Chapter 4). This will include discussion of simulator flow or in short how the simulator goes from the interface inputs, to modeling the system, to separation of the work, and finally to the interface output. Finally we will wrap up the thesis with a summary and discussion of future work that can further the utility of the M3C simulator as well as utilizing it (Chapter 5).

2. SIMULATOR DESIGN

2.1 Making a Many-Core Simulator

2.1.1 Overview

Today's evolution of computer architecture has led to a number of designs and products related to the many-core processor field including both homogeneous and heterogeneous processors. Industry leaders such as Intel have systems that include many-core homogeneous processors in their designs. For example, Intel's new second generation Core i7 contains a four core processor [19]. AMD's Phenom II processors are a competing brand which also contain a three to six core processor [20]. IBM's Power7 processor design includes four, six, or eight core processors [21]. While these processor designs have number of cores in the single digit, it is not uncommon to see sixteen core systems or even like the Azul System's Vega 3, which has fifty-four cores [22]. Some many-core heterogeneous processor examples include IBM's Cell and Xilinx V4 [23, 24]. The industry indeed is trending towards larger and larger many-core homogeneous and heterogeneous processors, and instead of the regular two dimensional layout, three dimensional structures are being considered [25].

Many-core heterogeneous processor simulator designs in the past have been limited by learning barriers because they were based on simulator specific language or limited in scope and flexibility dealing with a specific architecture. One of the benefits of the M3C simulator is its flexibility and the wide range of designs and connections that can be achieved. A visual interface makes understanding the design and tools simple and intuitive for users with basic to advanced knowledge. Thus, various designs can be created and modified easily.

2.1.2 M3C Simulator Resource Structure

The M3C simulator resources are split up into a number of types and objects that are contained within each other. These major types and objects include: The Simulator itself, individual simulation, Cores, Modules, Instructions, IO Components, Storage Resources, Networks, Routers, and Connections (shown in Figure 2.1).

The simulator resources for the M3C simulator includes:

Simulator: The Simulator itself is the governing control and structure of the M3C simulator system. The interfaces and data configurations are created and managed in this section. This is the parent of all other objects and types, and there is only one of them per user or user group.

Simulation: A single simulation containing a simulation layout or configuration. The simulation files are loaded and saved in XML code that is interchangeable with the Mhetero simulator [16]. This is the parent of all configuration types, and any number of these can be made for one simulator.

Core: A high level resource type and object. Core types are the main resource in the M3C simulator and are used to describe processing cores. A Core type description is configured with all of its components, mainly Modules, Instructions, IO Components, and Storage Resources. Once a simulation is built and executed, a specified number of Core objects are instantiated from any Core types.

Module: A section within a Core type and later within Core objects which describes how the core instance acts given a set of corresponding Instructions. There can be a number of Modules within a Core type or object, and the way they are interconnected and executed can be set in the simulation configuration.

Instruction: An Instruction type and single Instruction can be described. Instructions such as Load or Add can be designed like ARM or MIPS instruction set or even a new set.

IO Component: A section of a Core type that is created for every instance of a Core object. The IO Component describes the communication of a Core object and the outside. Any number of input and output buffers can be described in this part.

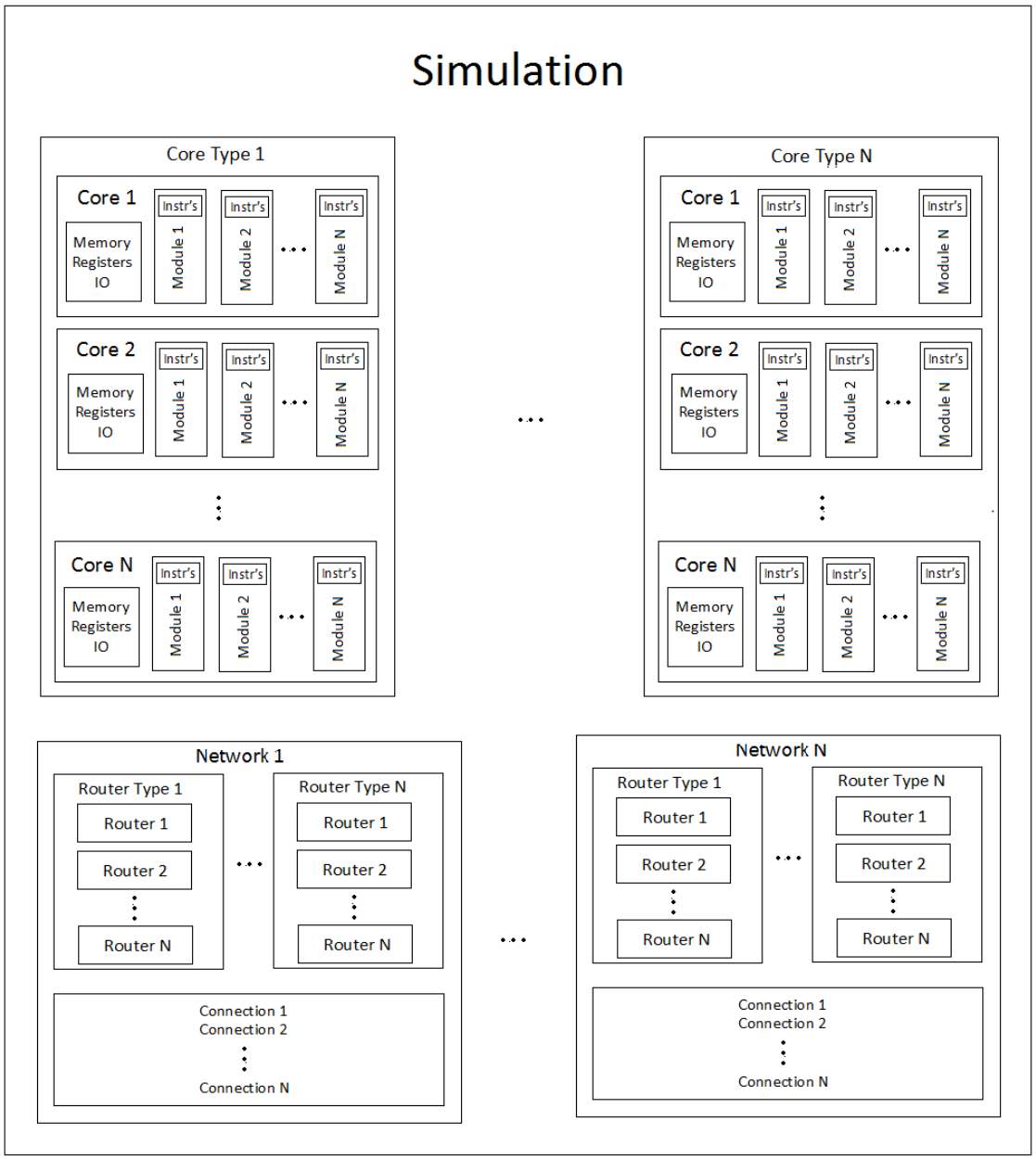


Fig. 2.1.: Simulation layout and design for the M3C simulator

Storage Resources: Memory and register descriptions and interactions can be described from the simulation configuration section. These are manifested in each Core object. Both memory and registers that are assigned to each Core object can be updated or examined using the IO Component.

Routers: A component that acts as an interface between Core objects in a system. Input and output buffers of varying number and size can be created, and characteristics or behaviors can be assigned to each Router type.

Network: A high level component on the same level as a Core type. A Network describes interrelationships between instances of Router types and Core types. Router types are described within Networks. Connections are assigned within Networks as well.

Connection: A one way description of the interaction between a Router object and a Core object, a Core object and another Core object, or a Router object and another Router object. Connections can be assigned in any number of ways for any number of resources up to and including fully connecting all resources of a simulation.

Figure 2.1 shows a Simulations structure. A Simulation contains any number of Core types and Networks.

Within each Core type, Modules, Instruction, IO, and Storage are described and interconnected. Just how Core types are configured and how these configurations are managed and executed will be described in greater detail later. Each Core type is instantiated into a number of Core instances or objects each with the same Module arrangement and initial IO and Storage setup. Each Core instance is given a different program that is created separately and that contains, for instance, compiled machine code for an ARM program.

Within each Network, Routers and Connections are described. Just how Networks are configured and how these configurations are managed and executed will also be described in greater detail later. Router types are configured, and upon the simulation being run, instances are created and they follow their described behavior. Connections

are assigned from source to destination giving a specific Router instance or Core instance.

2.1.3 M3C Simulator Resource Modular Design Concepts

Modules are one of the concepts in the M3C simulator that make it flexible, extendable, and configurable. Modules can play several roles in the simulator. They can represent stages, components, or a number of other experimental units. In Figure 2.2a one can see the basic five stage pipeline [26]. Modules are executed in the order of priority that can be set in the Simulator Configuration. In the case of Figure 2.2a, there are five stages: Fetch, Decode, Execute, Memory, and Write Back. Within the Modules, a user can configure them to do any number of tasks using a familiar language family of the C, C++, or C#.

In addition, an external modules or precompiled Dynamic-Link-Library (DLL) can be used in place of a code description to detail the behavior of a Module. Using external modules, a user can gain more control over the simulation, adding more detail where they require. Additional functions, classes, behaviors, variables, and other implementations can be introduced in this manner.

Communication and flow between Modules can be configured and managed in a wide range of topologies. Data channels like stage buffers can be established between modules that act as communication lines. These data channels are automatically managed by the simulator framework.

Figure 2.2a shows a unidirectional communication path for the five stage pipeline. This is the most basic channel setup. Figure 2.2b shows the same five stage pipeline with Forwarding added in. When the Memory stage is reached, data is set through the channels so that the Fetch, Decode, and Execute stages have updated memory information. When the Write Back stage is reached, data is set through the channels so that Execute has updated register information. Finally in Figure 2.2c, a new design

is shown with nine Modules and a number of interconnection data channels. Figure 2.2c shows that there can be a wide variety of Module designs and layouts.

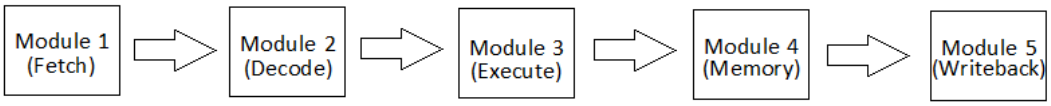
2.1.4 M3C Simulator Resource Design Connectivity

Communication between resources in the simulation is another important idea. Networks, as discussed earlier, act as communication managers. Just as in a large scale Network, these Networks on Chips or NoCs interconnect resources through connection. There can be many Networks defined and interconnected linking instances of Core types with each other. There are two main parts to a Network, as it pertains to the M3C simulator, Routers and Connections.

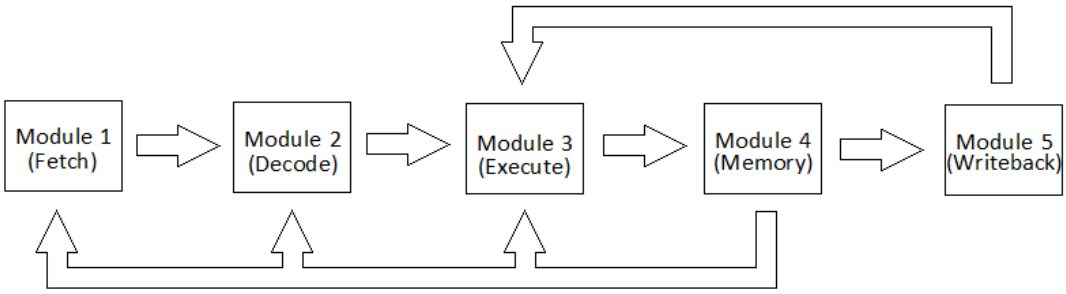
Router types are simplified Core types with the IO Component strengthened and the other components removed. A number of Router types can be configured in a number of ways. Data input and output buffers are created and assigned to each of these types. The behavior of the Router objects is set by the user and how data flows from each of the input and output buffers.

Connections are the most numerous in most simulations. Connections define half duplex links between Router and Core objects. Figure 2.3 shows a representation of how Core objects, Router objects, and Connections interact with each other. One can see from Figure 2.3 that numerous connections can be defined for each Router object and/or Core object. Output and input queues or buffers of varying length can be assigned to each resource, and the Connection uses these queues or buffers.

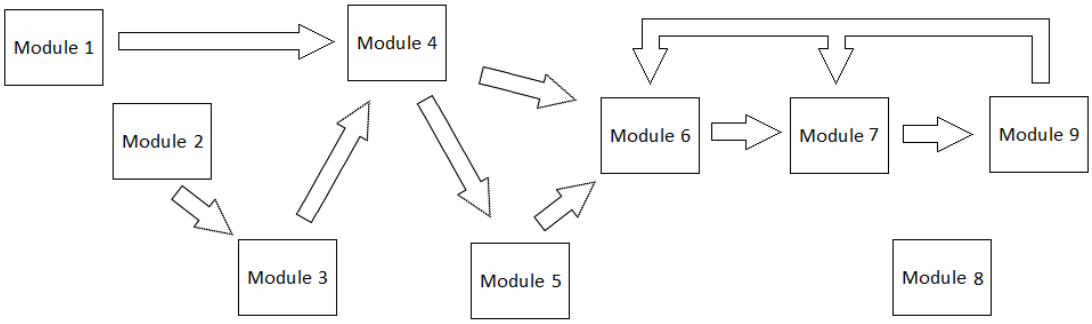
One can begin to see the potential for variation and specialized designs that can be developed from this connection schema. Figure 2.4a shows an example of a traditional 2D-mesh Core and Router NoC topology. Each Router object is directly connected to a Core object. The Routers are interconnected in a two dimensional layout with Routers having four input and output buffers corresponding to four neighboring routers. Figure 2.4b shows an example of a new Core and Router NoC. Routers play dynamic roles with some connecting to four Core objects and to a central Router and



(a) Basic 5 Stage Pipline



(b) 5 Stage Pipline with Forwarding



(c) A New Design

Fig. 2.2.: Module topology examples

one Router connecting to two Core Objects and then to the same centralized Router. This gives a three dimensional layout to the system. This new NoC topology is just to show that many different topologies can be constructed depending on the user's needs and creativity.

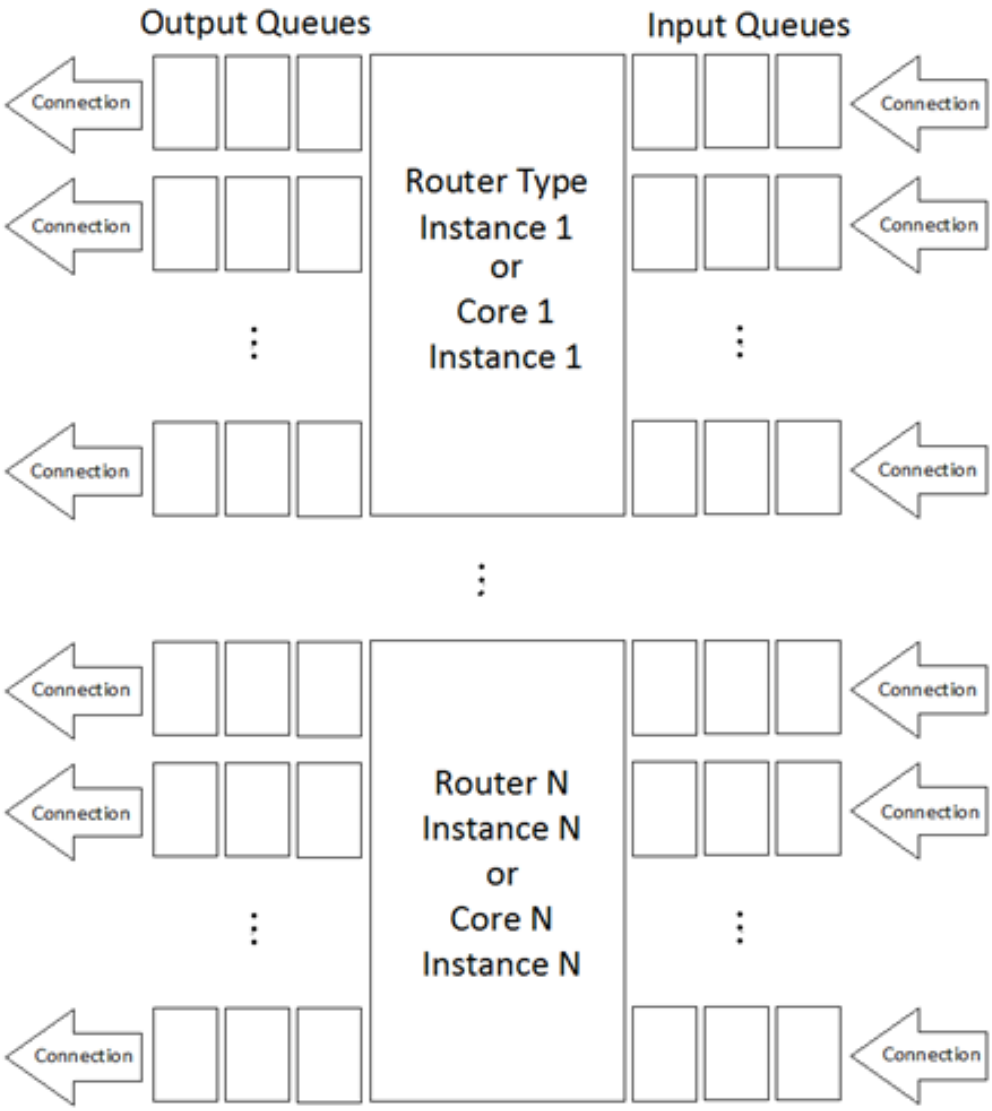
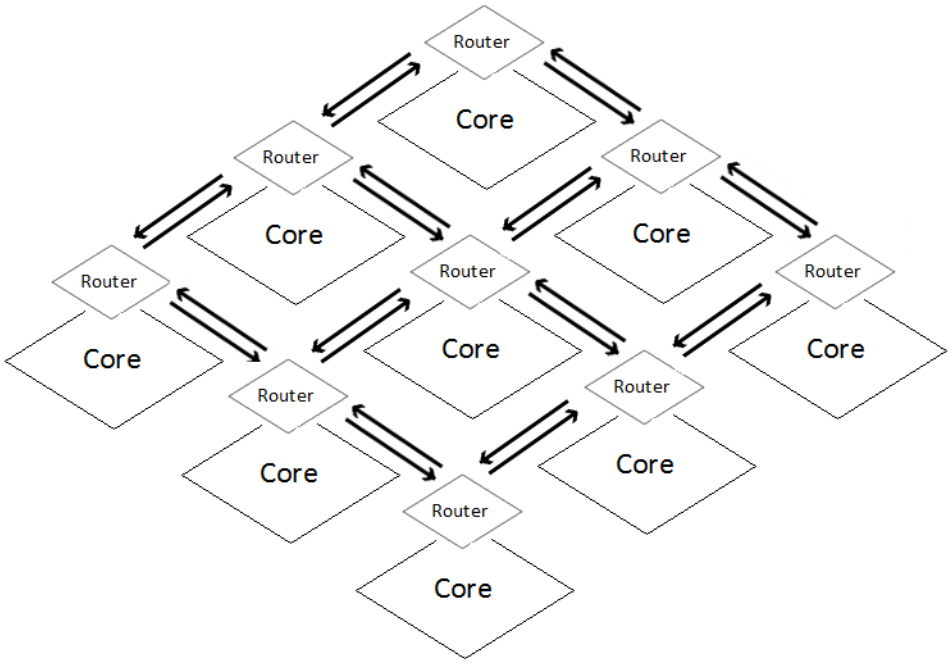
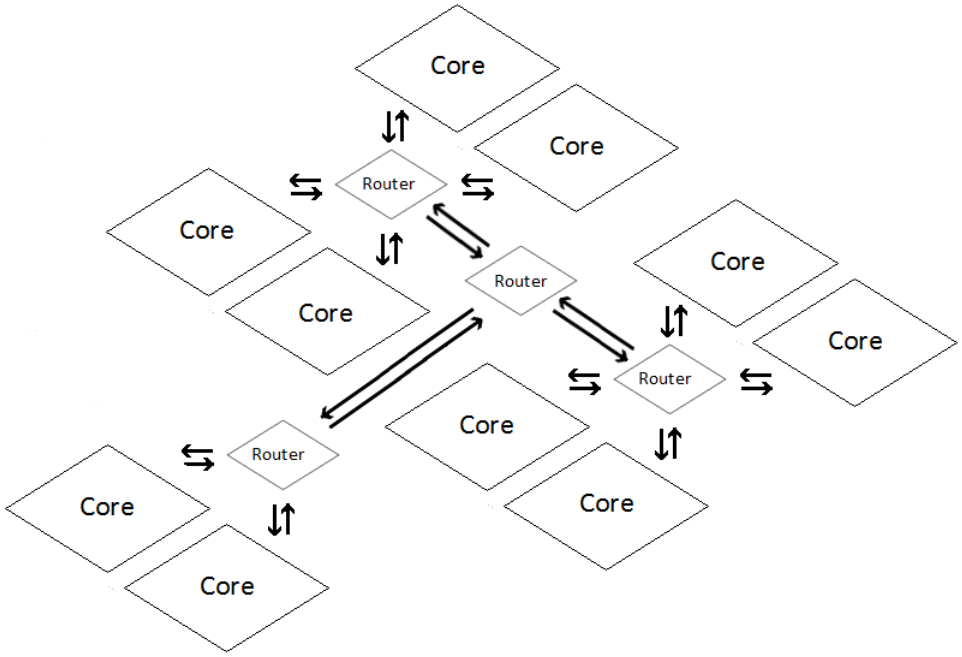


Fig. 2.3.: Resource connection diagram



(a) Tradional



(b) A New Design

Fig. 2.4.: Resource connection topology examples

2.2 Simulation Dynamics on the Cloud

2.2.1 Overview

Cloud Computing is an old idea which in recent years has been publicized, made more and more practical, and has now become prevalent. Cloud Computing itself is a very complex subject with many varying definitions and viewpoints ranging from the basic concept to the types of services that should be referred to as Cloud Computing.

NIST defines Cloud Computing as follows: Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics (On-demand self-service, Broad network access, Resource pooling, Rapid elasticity, Measured Service); three service models (Cloud Software as a Service (SaaS), Cloud Platform as a Service (PaaS), Cloud Infrastructure as a Service (IaaS)); and, four deployment models (Private cloud, Community cloud, Public cloud, Hybrid cloud). Key enabling technologies include: (1) fast wide-area networks, (2) powerful, inexpensive server computers, and (3) high-performance virtualization for commodity hardware [27].

The M3C simulator uses the Azure Cloud Computing Platform. The Azure Cloud Computing Platform is a Cloud Software as a Service (SaaS) model. Azure allows users to create programs and then host them on its services with storage capacity. A virtual instance of something like Windows XP can be run when a user wants to use it, and when the user is finished, it can be stored for future use. The user only has to pay for the resources they consume and does not have to overestimate or buy resources to match their highest usage expectation.

Azure Cloud Computing Platform has a couple of fundamental resource types available to developers. Azure Roles are one of these resources. Roles play a special part in Azure having a number of distinct and useful characteristics. One useful

characteristic of Roles is their ability to be quickly, easily, and cheaply scaled up and down depending on the needs of developers and their designs. Another characteristic of Azure Roles is the communication abilities built right into them called External and Internal Endpoints which we will discuss more of later.

There are three primary Azure Roles: Virtual, Web, and Worker. Virtual Role is not used in the M3C simulator, although it was originally, before Azure Cloud Computing Platform was examined closely, to be the resource the simulator would use. This was when the original objectives of the research were to just port the Mhetero simulator to a Cloud Computing Platform.

Web Roles are commonly used as a front end service being able to host websites to which users can connect over a web browser. Web Roles use ASP.Net and IIS 7 which are familiar to several developers allowing for a smooth transition for general website creation. Our experience with ASP.Net was limited at the start of this project and had to be developed early on because, as will be shown in the next chapter, websites play a large role in making the M3C simulator available to users from all manner of devices. Web Roles are not meant to do computationally heavy work, but are rather meant to be used as interfacing and control.

Worker Roles unlike Web Roles do not have an easy-to-use connection to the outside world. They are meant to operate long running programs with computationally intensive work loads. Worker Roles like Web Roles can be interconnected using Internal Endpoints; however using External Endpoints is not as easy for developers. These Roles are used as background work horses and in the M3C simulator perform many of the heaviest work, leaving Web Roles to do the interfacing and control.

Figure 2.5 shows a basic abstract view of how Azure Cloud Computing Platform interacts with users and tasks. A user can connect using a number of devices such as smartphones or personal computers. As long as a device can run a web browser and connect to the Internet, Azure resources can be obtained. The Azure Cloud can be imagined as a set of Server Farms that Microsoft manages with differing or similar networking topologies spread around the world. Each Server Farm has several

machines that can run a number of our Roles. For the simulation developer, there are numerous levels of abstraction here that do not need to be closely examined. The developer does not care if their Roles all work on the same machine or even in the building, the Azure Cloud handles all of this. Roles from different machines that need to work together can be accessed from each other to perform their tasks.

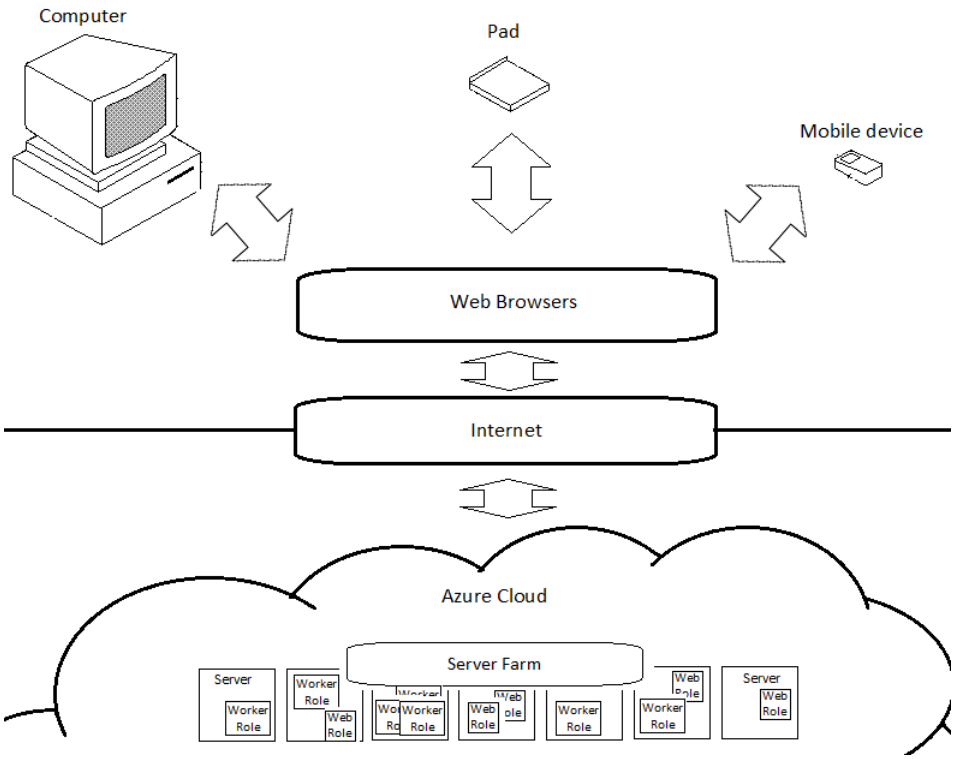


Fig. 2.5.: Simple diagram showing how users can connect to Azure Cloud Computer Platform

2.2.2 M3C Simulator Cloud Computing Structure

A large advantage that using Azure's Cloud Computing Platform lends to a developer is the ability to easily parallelize tasks using Azure's built-in structure. If properly created and managed, programs and tasks can be scaled up or down with a push of a button. Because of this scalability, one simple light or medium weight resource can be duplicated and the work can be shared among its duplicates. This allows for division of labor, making large cumbersome programs unnecessary and even cost inefficient. In this research, the M3C simulator is designed in such a way to take advantage or leverage this characteristic.

The M3C simulator's Cloud Computing structure uses two kinds of Azure Roles: Web Role and Worker Role. A Web Role acts as a front end. The Web Role has a URL associated with it at which users such as students can connect allowing them access to the Azure Cloud Computing Platform. The URL leads a user to a website that acts as an interface to configure and control the M3C simulator. ASP.Net is used to design and create user interfaces on Web Roles similar to Windows Forms, however with a great difference in execution, design, and management. As will be shown later, the interface design is similar to the Mhetero simulator's interface design so that users of the grandfather simulator feel comfortable and do not have to learn a new system. In addition to the user interface, the Web Role controls the Worker Role's set up and intercommunication.

For every Web Role, there is a number of Worker Roles that can be set as a ratio in the code. The Worker Roles are given Core type and Router type information. When the user gives the number of each Core type and Router type that each Worker Role is to instantiate, the Worker Roles create a set of Core objects and Router objects.

There can be one Worker Role only, in which case there is no division of Core objects and Router objects. If there are two or more Worker Roles for one Web Role, the current model for the M3C simulator puts all the Router objects on one of the Worker Roles and then evenly divides the number of Core objects among the

remaining Worker Roles. Router types are simpler versions of Cores types, and so far it has been noticed they can easily be handled by a single Worker Role.

Each of the Worker Roles runs a set of objects keeping track of their information such as memory. The information for each object is continuously updated before each tick of a Core object's operation. The corresponding Web Role checks in on each Worker Role in turn and displays the information to the screen. There are a couple of current designs on how NoCs are handled and memory is updated across an M3C simulation. The Web Role can act as an intermediated transporter as well as a controller. Data can be kept in a list within the Web Role and when a Network requires an update between two resource objects, the Web Role can send the data from its bank and then a message to the destination resource object that it needs to switch new data. Another model is for the Web Role to alert the source resource object that it needs to send information to a destination resource object. The Worker Role that contains the source resource object will then search for the corresponding destination resource object to update its information. Once the Worker Role containing the source resource object finds the destination resource object, it alerts the Worker Role containing the destination resource object. The information is then updated for the destination resource object.

Figure 2.6 shows a Web Role with nine Worker Roles below it. This is referring to a ratio of one to nine. There are eight Worker Roles that will potentially divide a set of Core objects between them with one Worker Role devoted to Router objects. The Worker Roles shown in Figure 2.6 are duplicates including code and design. They are created from a template and nine instances are instantiated.

In Figure 2.7 one can see when a Web Role is instantiated more than once. If a ratio is set like before to be one to nine and a corresponding number of Worker Roles are instantiated then the first instance of the Web Role connects with the first nine Worker Roles. The next Web Role takes the next nine Worker Roles and so on till there are no more Worker Roles or there are no more Web Roles.

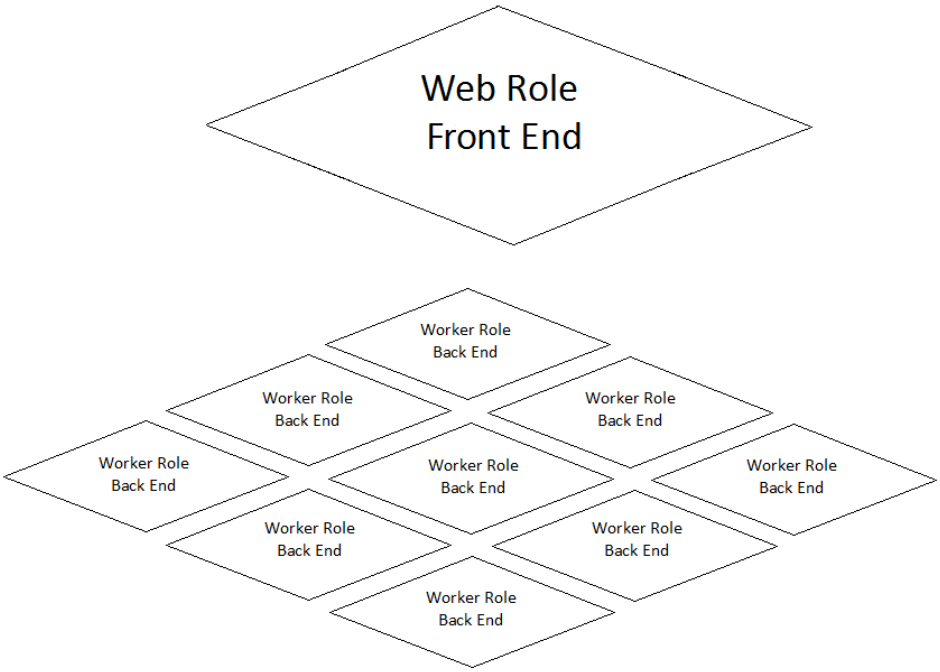


Fig. 2.6.: Role layout for the Cloud Computing design of the M3C simulator

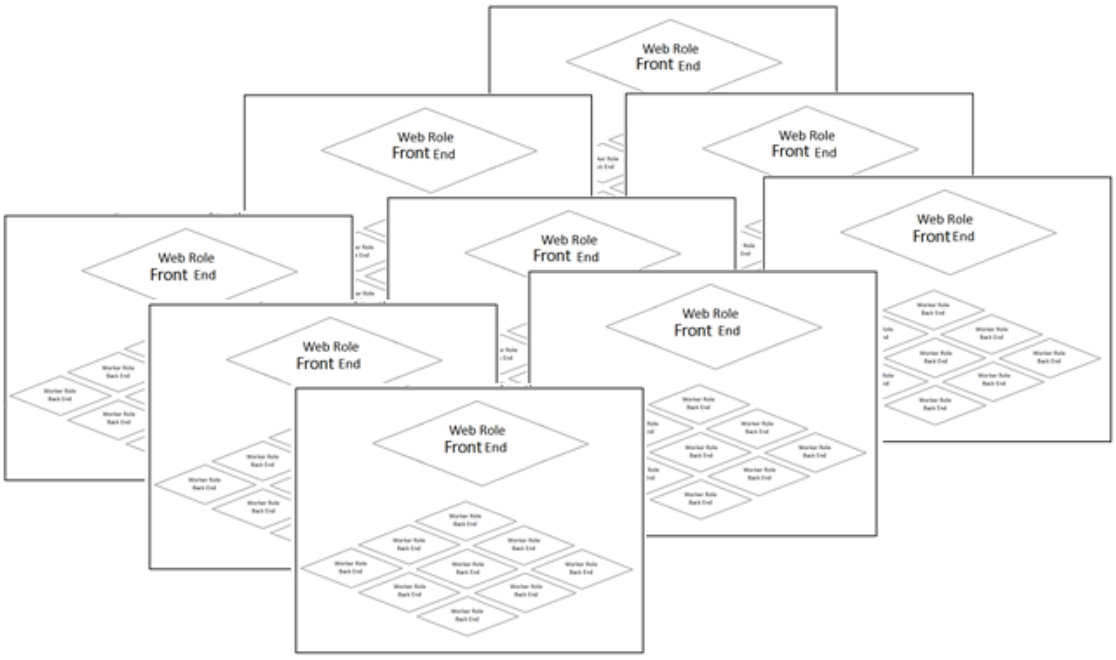


Fig. 2.7.: Multiple instances of the role layout for the Cloud Computing design of the M3C simulator

2.2.3 M3C Simulator Cloud Computing Connectivity

Like the Core objects and Router objects, the Worker Roles and Web Roles are interconnected in their own network. This network plays a role in the lower level NoC. Each instance of a Role has an individual Azure Role ID corresponding to its creation.

The Roles can have External or Internal Endpoints that work as communication ports to the world outside and inside the Azure environment. As the Web Roles host websites, they communicate with the outside world using one External Endpoint each. Internally, Roles can use TCP Bindings through an Internal Endpoint.

Each Worker Role has two Internal Endpoints they use to host TCP Bindings. One of these Internal Endpoints is used by the controlling Web Role to control, monitor, and send messages to each pair of the Worker Roles. Another Internal Endpoint is used to communicate between each pair of the Worker Role instances. Communication is defined by the type, either a Worker Role or a Web Role, by the Role's name, and finally by the Role's instance ID.

These connections, as they will be described later, are temporary while the host is maintained. When a Worker Role or a Web Role wants to talk to a Worker Role, they search to find the correct type, name, and ID and then a Binding is created. These Bindings are used to create Channels to complete any communication, and then both Bindings and Channels are terminated.

If there are N Worker Roles that correspond to a Web Role, then there are N possible temporary connections. Every Worker Role can possibly be temporarily connected to every other Worker Role. This means there can be $N\text{-}1$ connections per Worker Role and $N(N\text{-}1)$ total connections among the Worker Roles. Overall that means there are $N+N(N\text{-}1)$ possible temporary connections making a fully connected system. This is the maximum connection, and depending on the design or configuration, there can be much less connections used.

3. THE MANY-CORE PROCESSOR SIMULATOR

3.1 Setup Interfaces

3.1.1 Overview

The user interface for the M3C simulator had to be easy to use for beginners such as students in a computer architecture class, versatile and useful enough for experts such as researchers, and familiar to users coming from the Mhetero simulator. This section and the following section will outline the user interface for the M3C simulator and discuss how the interface interacts with the internal tasks and configurations.

Because the M3C simulator uses a Web Role hosting a website, the user interface is accessed through a web browser. This adds interesting challenges to the creation, interaction, and maintenance of the simulator. In this section, one will see and learn more about the main menu choices, and how the simulator works over the Internet to handle files and keep track of configurations. We will also take a look at the main simulator configuration interface and how it has a major influence on the rest of a simulation.

3.1.2 Menu and File Configurations

Figure 3.1 shows the menu options for the M3C simulator. There are three options for Simulations: New, Open, and Save. When the new option is selected, the current simulation is wiped and a reset starts a new empty simulation. The Save option saves the current simulation into an XML format [16]. A download box opens so that a user may select where and how they want to save the file, as shown in Figure 3.2. The Open option triggers a browse button as shown in Figure 3.2. The browse button allows users to connect to their computer and select a file whose path will

be displayed in the navigation text box. If the user presses the upload button, the file will be transferred from the user's computer to the Web Role. This XML file that was uploaded is parsed, and the nodes and sections are propagated into Core types, Networks, and Router types. This is done with the XML tools available from Microsofts System.Xml library [28–30]. After the simulation has been filled out, a build command is instigated and the simulation attempts to build.

Because the M3C simulator was built to interact with the grandfather simulator Mhetero, an Add/Update Template/Associate Files option is available. Upon selecting this option, a user can use an upload file system similar to the one generated by the Open option. Framework files can be uploaded that act as a template for building Core types and Router Types. Users of the Mhetero simulator will have to be careful because these files are not all the same as the grandfather simulator, having been modified extensively for the M3C simulator. A number of files are important to the simulator and different versions of these files can be entered at any time using this option. A text box next to the upload file system shows a record of what files have been uploaded. DLLs, program, and data files may also be uploaded using this option. This option is very important upon starting a new M3C simulator. After the simulator is refreshed, files should not need to be uploaded in any case except for edition changes.

The Remove Template/Associate Files option activates a remove file system shown in Figure 3.2. A user may enter a file name they wish to remove in case local storage on the Web Role is being filled up. Though this should not be a problem due to the large local storage available and small size of the template.

The Edit Simulator Configuration option opens the Simulation Configuration screen. We will look at the Simulation Configuration screen shortly.

The Run Simulator option activates the run command and if the simulation is currently activated in the Worker Roles, the Simulation Run screen is loaded.

When the Rebuild option is selected, the build command is activated and the current simulation is built. This is used primarily when a totally new simulation has

been made or when new Template, program, or data files have been uploaded. We will go into this in more depth later.

At the bottom of the web page visible from all screens is a Simulation Information text box. This text box gives informative comments about the status of a simulation, including errors.

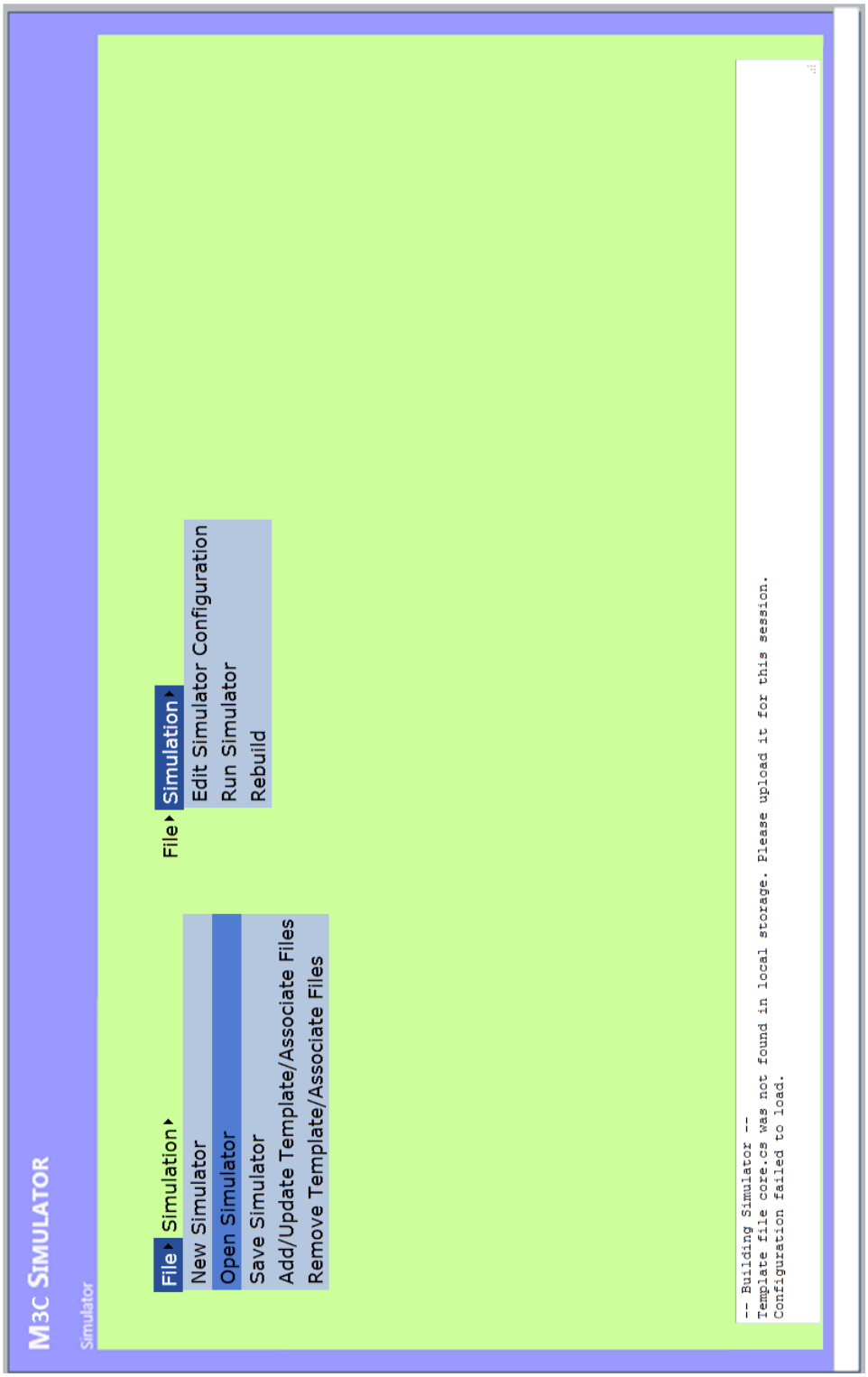


Fig. 3.1.: Main home page menu

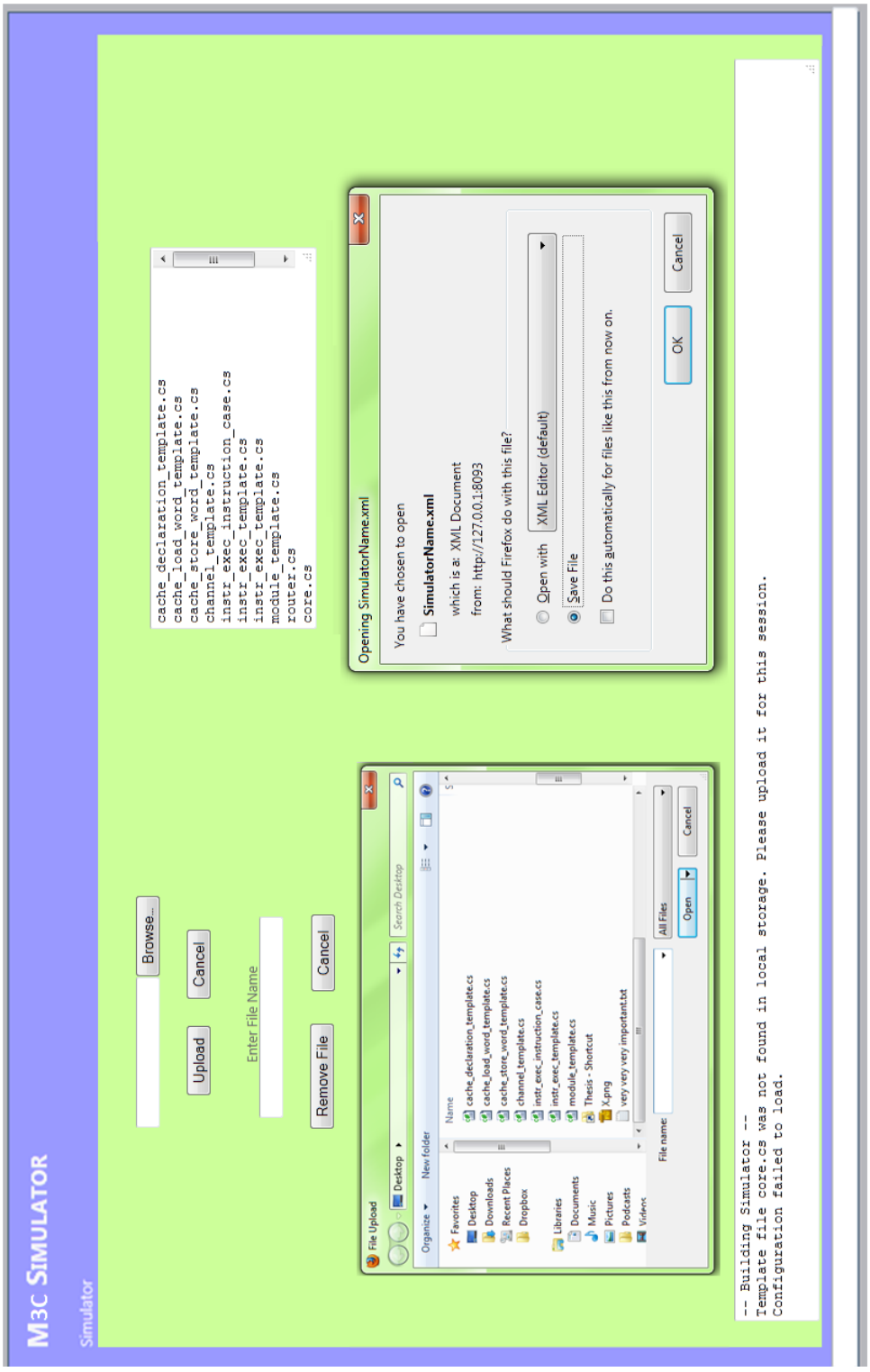


Fig. 3.2.: Upload and download interaction screens

3.1.3 Simulation Configuration

Figure 3.3 shows the Simulation Editor also called the Simulation Configuration screen. This screen is the main information and organization page for a simulation. There are a number of important configuration options on this screen that allow accesses to the Core types and Networks of a simulation. At the top of the screen, the name of the simulation is shown. An Execution Mode drop-down menu allows users to select from several execution modes; however for the M3C simulation all simulations run with one thread per resource to fully harness the Azure Cloud Computing Platform's resources. The Sync. Cycles option is also not used in the current version of the simulation. Future versions will implement this option. Because all simulations run as one thread per resource, the Threads In Pool option is superfluous. This option was kept from the Mhetero simulator for the purposes of being able to transfer files back to the grandfather simulator.

Under the Core Types section in Figure 3.3, there are a number of names in a list box. These are names for Core types that were made or loaded. If the Add button next to the Core types list box is pressed, a new empty Core type with a new name is created. When a Core type is selected the Delete and Edit buttons are enabled. The Delete button removes a Core type from the simulation and all its associated configurations. When the Edit button is pressed, the Resource Configuration Screen is loaded, giving accesses to the Resource Configuration Menu. Each sub-screen of the Resource Configuration screen will have the previously selected Core Types information available.

Under the Network Types section in Figure 3.3 there are a number of names in a list box. These names represents Networks. Like the Core Types list box, the Add button will add a new empty Network to the list box and Delete will remove a selected one. The Edit button opens the Network Configuration Screen with the Network Configuration Menu. The previously selected Network's information is available to each of the sub-screens of the Network Configuration Screen.

The Esc and OK buttons keep the current simulation status updating any fields that were changed and not already updated.

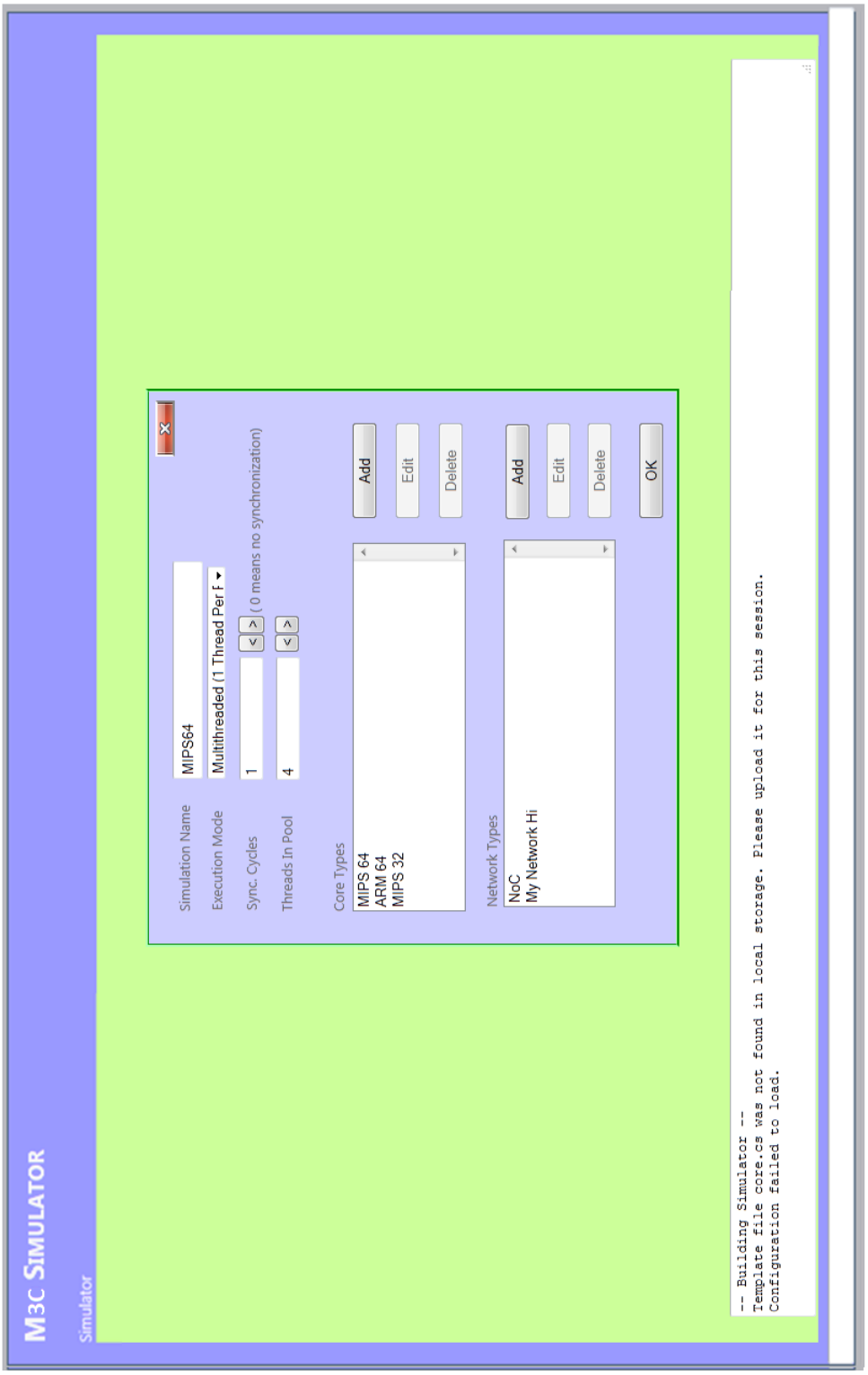


Fig. 3.3.: Simulation Configuration screen

3.2 Core Interface

3.2.1 Overview

After a user selects a Core Type from the Simulation Editor screen and then presses the Edit button, the Resource Configuration main screen opens. This section allows users to both examine and edit current configuration settings for a given Core Type. Settings such as number of cores, program files, registers, modules, memory, cache levels, and NoC interface can be modified. The Resource Configuration Menu allows the user to navigate among several sub-screens that will display and interact with the user.

Upon pressing the Esc or OK button the current settings are updated if they have not already been so and the screen returns to the Simulation Editor screen.

3.2.2 Basic and Program Configurations

The first section on the Resource Configuration Menu is the Basic Configuration sub-screen. This sub-screen allows users to rename the Core type. It is also the place to indicate how many instantiations of the Core type will be created when the simulation is run.

There are several buttons to the right of the Basic Configuration sub-screen that allow users to set programs to instances of the current Core type. The Add Program & Data button if selected adds a new empty row to the Program and Data Files to Execute menu list box as seen in Figure 3.4. The Delete Program & Data button as it can be assumed will delete a selected row from the Program and Data Files to Execute menu list box.

When a row is selected on the Program and Data Files to Execute menu list box, the user can use the Select Program button. This option opens an upload system like the one used in Open from the main screen's menu. A program file can be uploaded

at this point and will be saved in the Web Role's Local Storage. The row on the Program and Data Files to Execute menu list box is updated as well.

When a row is selected on the Program and Data files to Execute menu list box, the user can use the Select Data button. This option opens an upload system like the one used in Open from the main screen's menu. A Data file can be uploaded at this point and will be saved in the Web Role's Local Storage. The row on the Program and Data Files to Execute menu list box is updated as well.

The program and data files will be used later when the Core objects are instantiated. On this screen, we can indicate which instance of the current Core type will be using what program and data files. Unlike the grandfather simulator, the M3C simulator will give all the unspecified instances of a Core type the first program and data files to be entered so no Core object is action-less.

The program files and data files are created from the compilation of code. For instance, if a user designs a MIPS Core type that can recognize MIPS machine code, the program and data files will be composed of the machine code of a MIPS program. This is prepared separately from the simulator, but it is hoped in the future to add this ability into the simulator for some instruction sets.

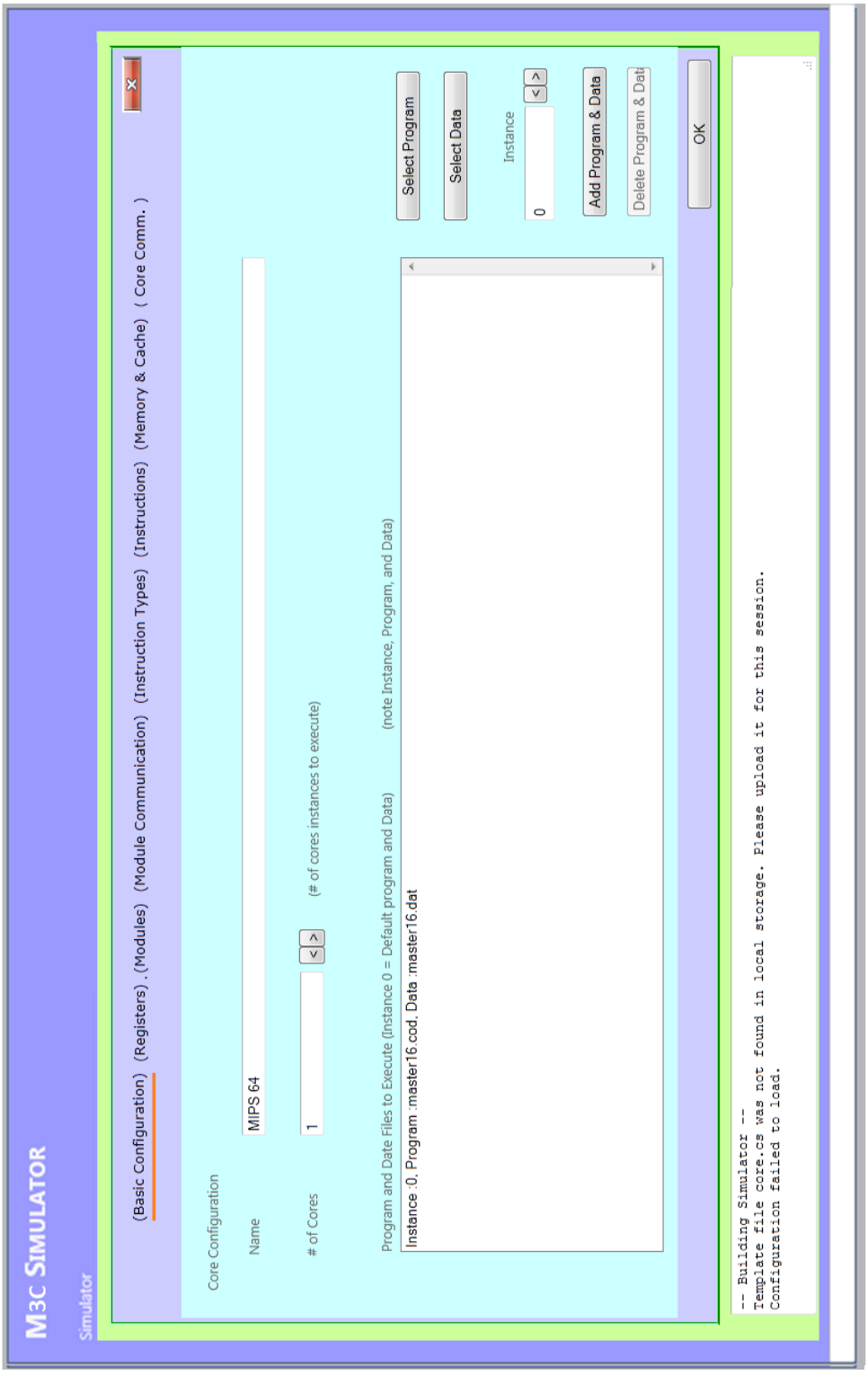


Fig. 3.4.: Resource Configuration, Basic Configuration sub-screen

3.2.3 Register Configuration

Figure 3.5 shows the Register sub-screen. In this sub-screen a user can create registers that a Core type would use including a program counter.

Registers can be set up in a number of ways, and mainly the name, number of registers, and data type can be defined. This information will be used in the program later as if they were registers in a physical core.

A user can press the Add Bank button to the right of the sub-screen to add a new register bank to the Register Bank list box, but only after a name is placed in the Name text box. If a register bank is selected from the Register Bank list box, the Delete Bank button may be used to remove it from the list and from the simulation.

After a Row of the Register Bank list box is selected, a user can modify the register bank it represents. The Data Type section allows the user to change the data type of a register bank among several types. A user may specify a number of registers that are present in any register bank. These registers will be represented by arrays in the code later and can be referenced in other sections of the Resource Configuration screen. A register can be referenced like any array in C programming language; for example if a register name is *reg* and four registers are present, then register three can be accessed by writing *reg[2]*. Later, whenever source code is required, a user can refer to one of the register banks and one of the registers they contain, and use them corresponding to the data type it represents. Modules and instructions will refer to this register to do work and store data.

If any modifications are made to the register banks, then the Submit Edit button may be pressed to update the simulation.

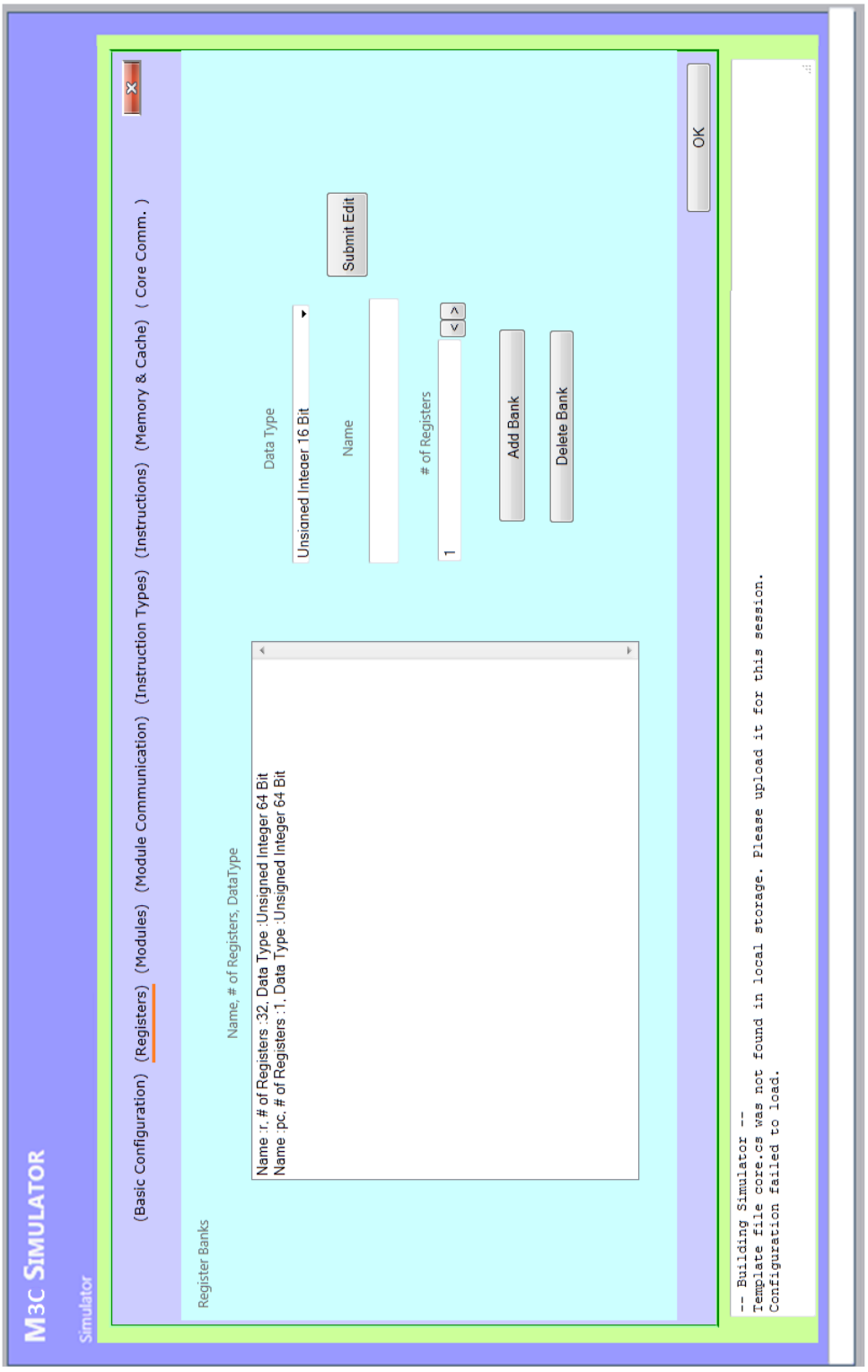


Fig. 3.5.: Resource Configuration, Registers Configuration sub-screen

3.2.4 Module Configuration

Modules, as described earlier, are a stage-like resource that can be interconnected to take on behaviors, like for instance pipeline stages. For more information please refer back to Chapter 2. Modules are configured and channels between them set in two sub-screens shown in Figures 3.6 and 3.7, respectively.

Considering Figure 3.6, there are a large number of important control features. To the left of the sub-screen, the Module Name list box contains all the modules that are present in a Core type. A user may add a new Module by entering a name into the Module Name Input text box and then selecting the Add Module button. A user may also delete a Module by selecting it from the Module Name list box and pressing the Delete Module button.

When a Module is selected from the Module Name list box, several data points on the sub-screen are filled in. The Module's name is placed in the Module Name text box. If the user has given the selected Module a priority, this is loaded at this point. A user may assign or change a Module's execution priority by changing the Execution Priority text box. The priority of a Module indicates at which point in a Core object's execution it will be encountered. If the priority is one, then the Module will be first every tick of the Core object's execution.

The Source Code text box allows users to write or describe a behavior for a module. The Source Code text box may refer to registers or Module channels. When the dynamic code file is created for the simulator, the Source Code section will be placed into a section of the code corresponding to the Modules's priority. A function is reserved for the selected Module.

In this sub-screen a user may add a DLL per Module that represents the Module. The DLL can act as another Module and as described earlier add extra functionality for users that wish to do so. These Modules still have to have names and priorities. Just as other files, DLLs need to be uploaded and, if the simulator is reset, will need to be re-added for any simulation to be run properly.

Upon leaving this sub-screen or changing the selected Module, all data for a previously selected Module is stored.

Figure 3.7 shows the sub-screen Module Communication. This sub-screen details the Module Channels for communication between Modules.

At the left of Figure 3.7 one can see the Channel Name list box. A channel may be added by entering a name in the Channel Name Input text box and then pressing Add Channel. If a channel is selected from the Channel Name list box, it may be deleted with all its corresponding information by pressing the Delete Channel button.

When a channel is selected, its corresponding information is loaded. The channel name can be changed in the Channel Name text box. The Source Module and Destination Module drop-down boxes are propagated with the names of the Modules in the Module sub-screen. If a Module is deleted, it will be removed from the drop-down box and a channel will be set to another source or destination.

Each data channel has a set of variables that are the real method by which Modules get information from other Modules. A variable can be added by adding a name to the Variable Name text box and selecting Add Channel Variable. If variable is selected, then it may be deleted using the Delete Channel Variable button or edited using the Submit Edit button.

Upon leaving this sub-screen or changing the selected data channel, all data for a previously selected data channel is stored.

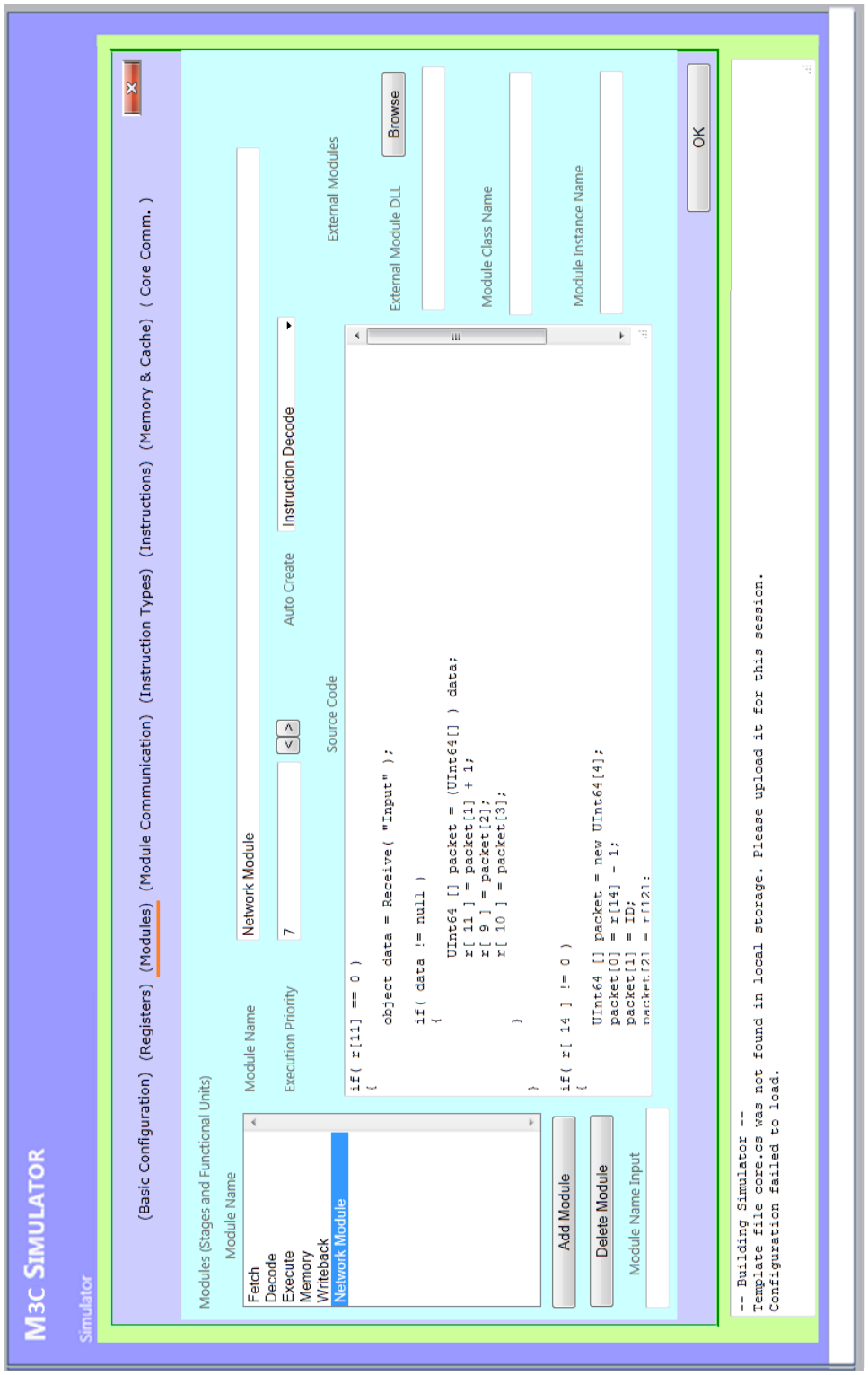


Fig. 3.6.: Resource Configuration, Modules Configuration sub-screen

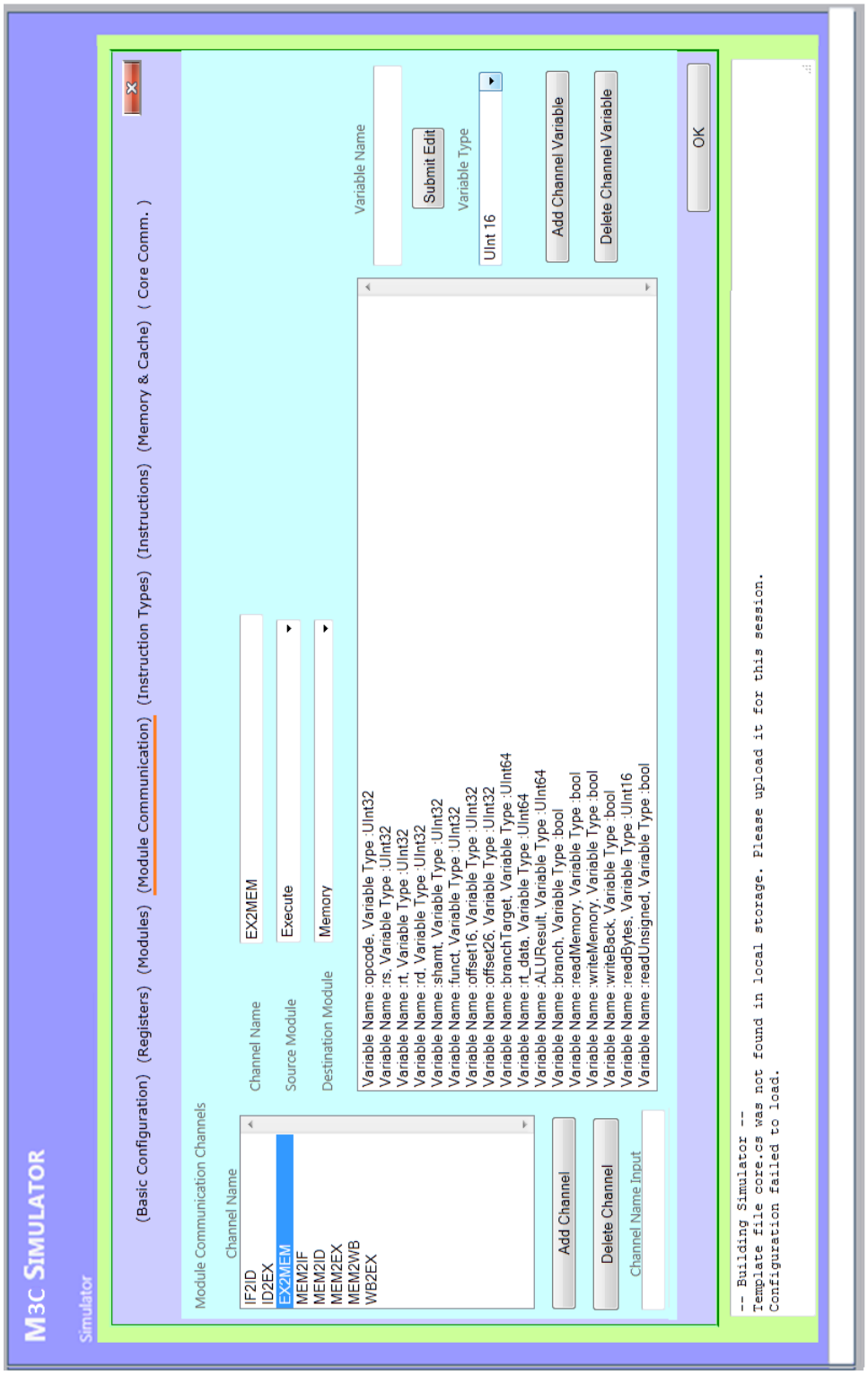


Fig. 3.7.: Resource Configuration, Modules Communication Configuration sub-screen

3.2.5 Instruction Configuration

Instructions are managed using two sub-screens shown in Figures 3.8 and 3.9. The first sub-screen allows users to specify instruction types. The next sub-screen allows the user to create specific instructions for each type such as a load or add.

The Instruction Types sub-screen shown in Figure 3.8 allows a user to add data types using the Add Instruction Type button as long as a name has been entered in the Inst. Type Name Input text box. If an Instruction Type is selected, the user may remove an instruction type from the Inst. Type Name list box. This sub-screen allows users to specify the high and low bits of instructions for opcodes in the Opcode Position section.

For every type, there is a set of Variables that describe a selected instruction type. A new variable can be added by the Add Variable button as long as a name is placed in the Variable Name Input text box first. Each variable has a range of bits indicating its high and low bits. This can all be edited by selecting a variable, changing the information that is placed in the text boxes, and then pressing the Edit button.

Instruction types are updated when a selected type is changed or when the sub-screen is changed.

In the Instructions sub-screen shown in Figure 3.9, instructions are individually added and edited by the user. An instruction may be added by pressing the Add Instruction button if a name is first placed in the Instruction Name Input text box. An instruction may be deleted if it is selected and a user presses the Delete Instruction button.

When an instruction is selected in the Instruction Name list box, information is loaded into the other parts of the sub-screen including the instruction's name. The Opcode field may be set and the instruction type may be selected from the Instruction Type drop-down menu, which in turn is propagated by the instruction types from the Instruction Types sub-screen. If an instruction type is deleted, a new instruction type is selected for all instructions that previously used that instruction type. Each

instruction may have its own source code that is added into the code depending on the user's configuration options in the Module sections of the Resource Configuration screen. The source code governs the behavior of each instruction and is written in a familiar language such as C, C++, or C#.

Instructions are updated when a selected instruction is changed or when the sub-screen is changed.

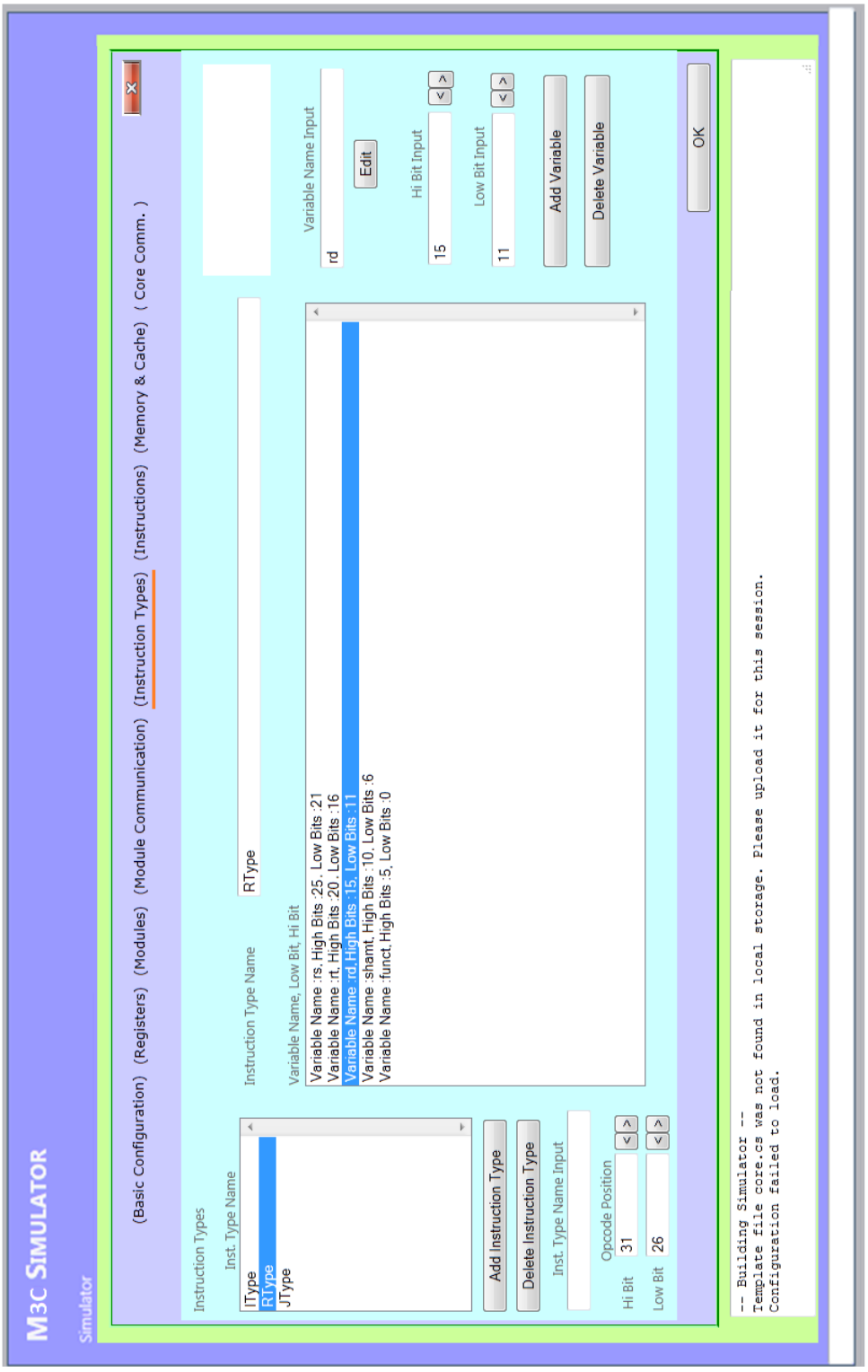


Fig. 3.8.: Resource Configuration, Instruction Types Configuration sub-screen

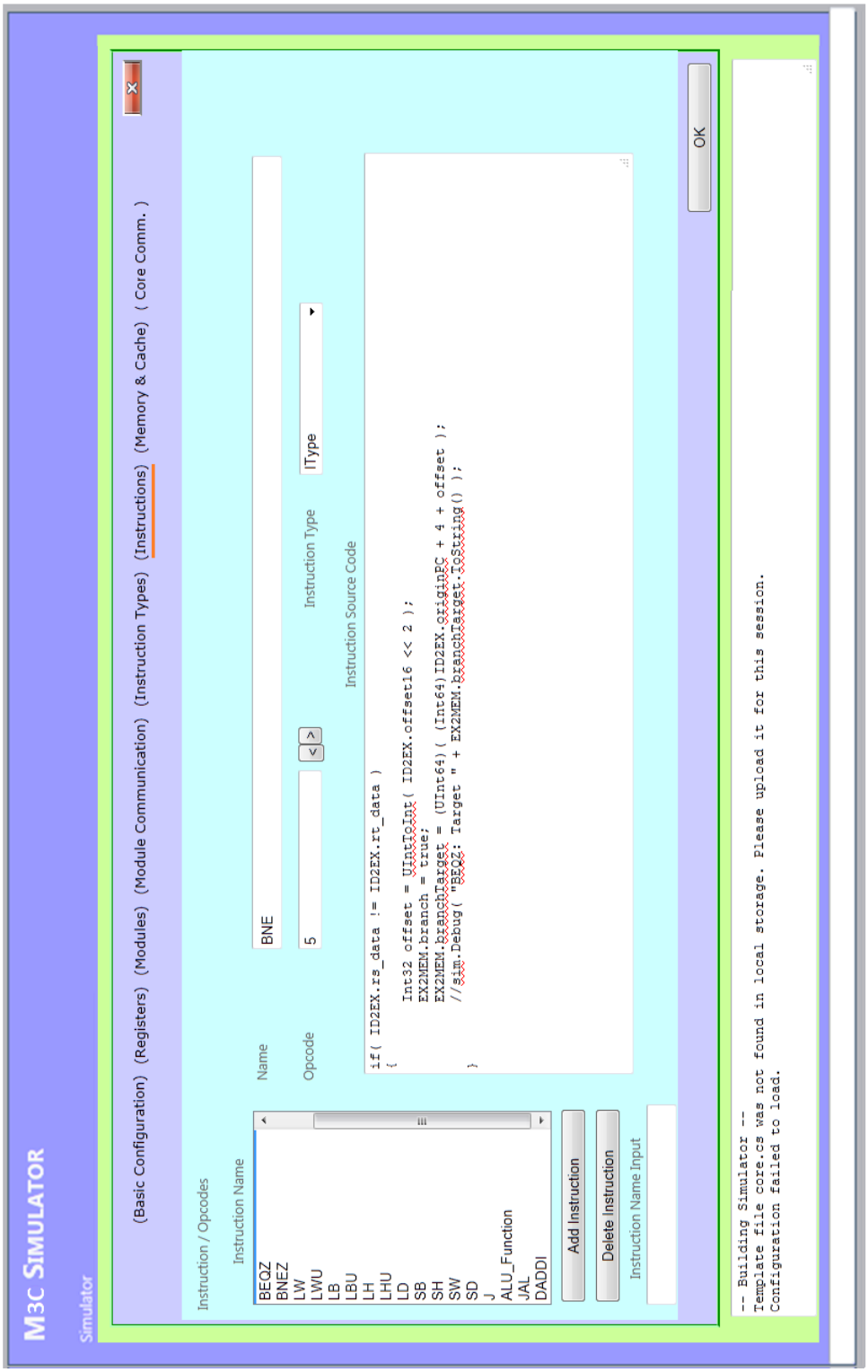


Fig. 3.9.: Resource Configuration, Instruction Configuration sub-screen

3.2.6 Memory and Cache Configurations

Memory and Cache is set in the Memory & Cache sub-screen shown in Figure 3.10. Memory can be configured in the top section of this sub-screen.

Data Memory Type drop-down menu allows a user to specify what type of variable data is stored in memory. # of Words allows a user to indicate the size of memory allocated for data storage given in words. Latency (Cycles) is a section that allows the user to specify an access latency or penalty to the data memory section. Instruction Position indicates the instructions offset in memory, and Data Position indicates where data start in memory. Instr. Memory Type, as one may assume, allows a user to set the type of variables where the instructions are stored. Instructions have their own # of Words Section and Latency (Cycles) section.

Cache levels can be configured in the bottom of this sub-screen. Cache levels may be added using the Add Cache Level button and deleted if selected using the Delete Cache Level button. Each cache level has a number of pieces of information associated with, including the level of the cache. Types of cache are selected including Direct Mapping, Set Associative, and Fully Associative caching. The replacement mechanism is selected to be either LRU or Random. The user may also select the number of sets, Blocks per Set and Words per Block. Latency of a cache access needs to be set as well.

After selecting a cache level, a user may edit it by changing any of the drop-down menus below the Cache Level list box and then pressing the Edit Cache Level button.

These options are important for proper configuration of a Core type, and each Core type should be examined and set depending on the design's specifications. For students, this is a good way of learning how Cache works and a way of introducing them to the concept of latency.

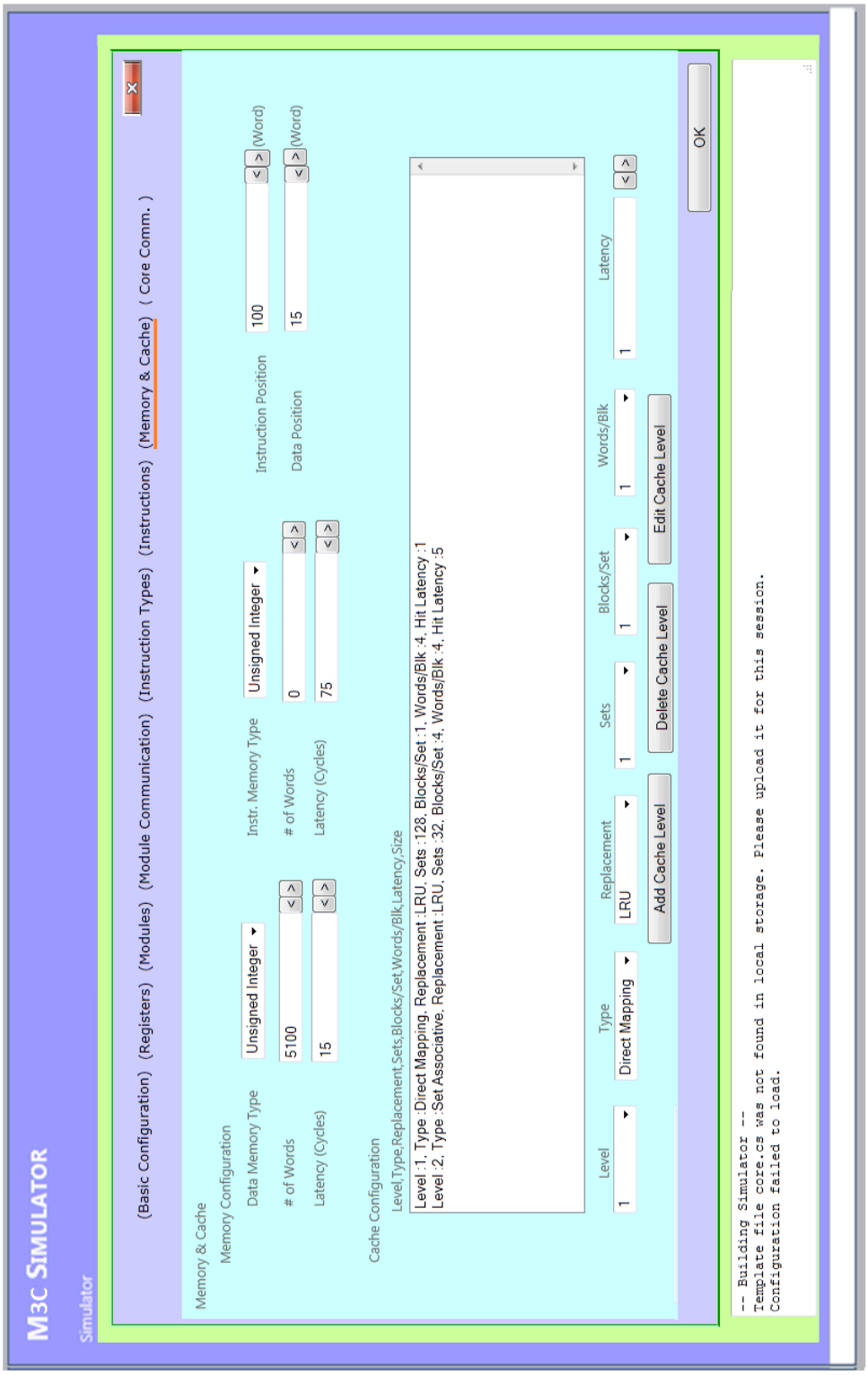


Fig. 3.10.: Resource Configuration, Memory and Cache Configuration sub-screen

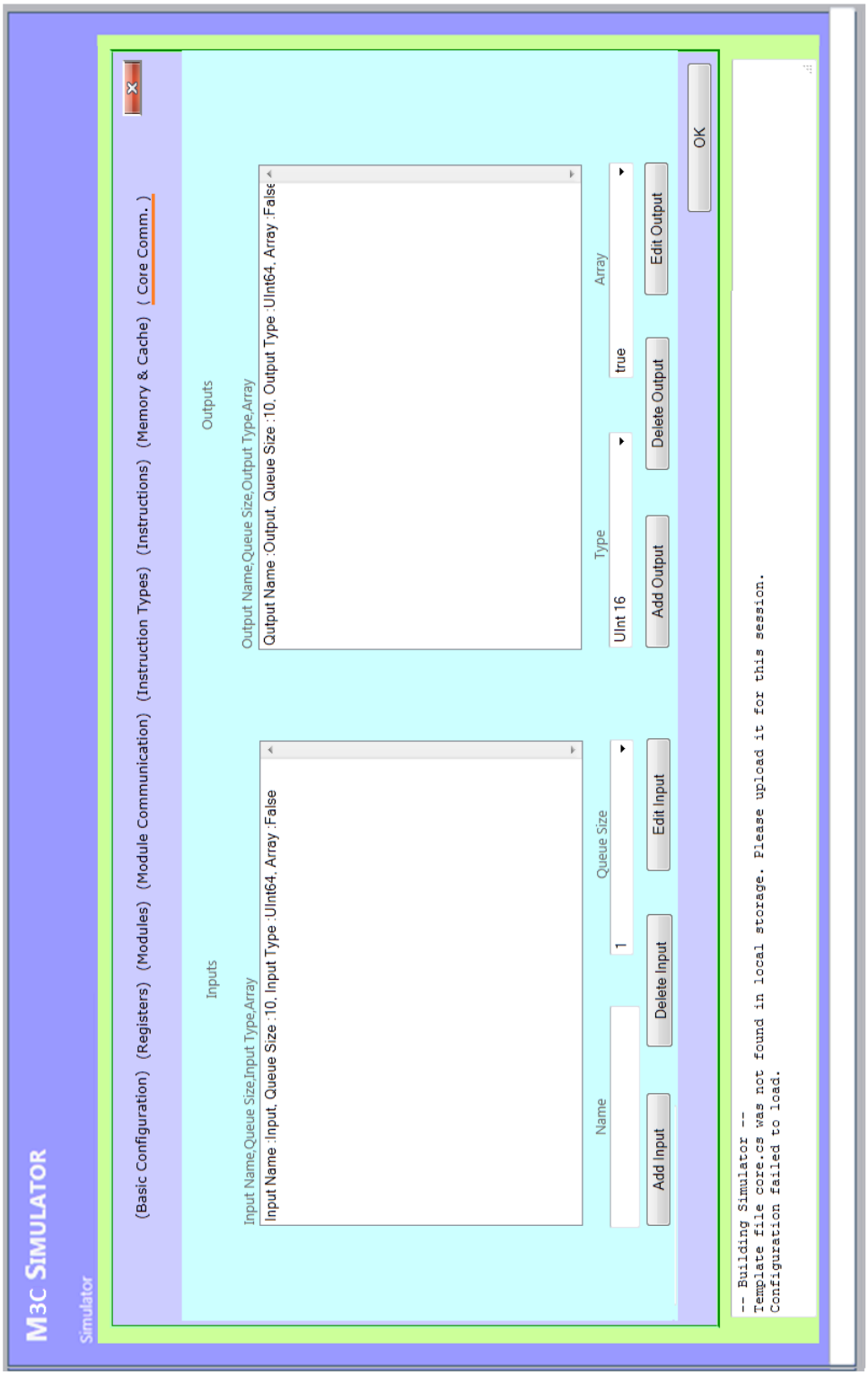


Fig. 3.11.: Resource Configuration, NoC Interface Configuration sub-screen

3.2.7 Core Comm. Configurations

The Core Comm. interface sub-screen is a straightforward part of the Resource Configuration screen. A user may select a number of input or output buffers used to communicate with any other Core objects or Router objects. The size of each buffer or queue may be set, as well as its data type, and whether it is an array or not. Data may be pushed onto these buffers or queues or pulled off of them. Figure 3.11 shows the sub-screen.

3.3 Network Interface

3.3.1 Overview

The Network Configuration screen allows users to make a range of configuration changes to a network. Routers may be created and behaviors set to them. Connections are also specified in a sub-screen of this section. This screen allows users to interconnect the Core objects, using the Core Comm. buffers, and the Router objects as was shown in Chapter 2. Figure 3.12 shows the Basic Configuration sub-screen where users can change the name of a Network.

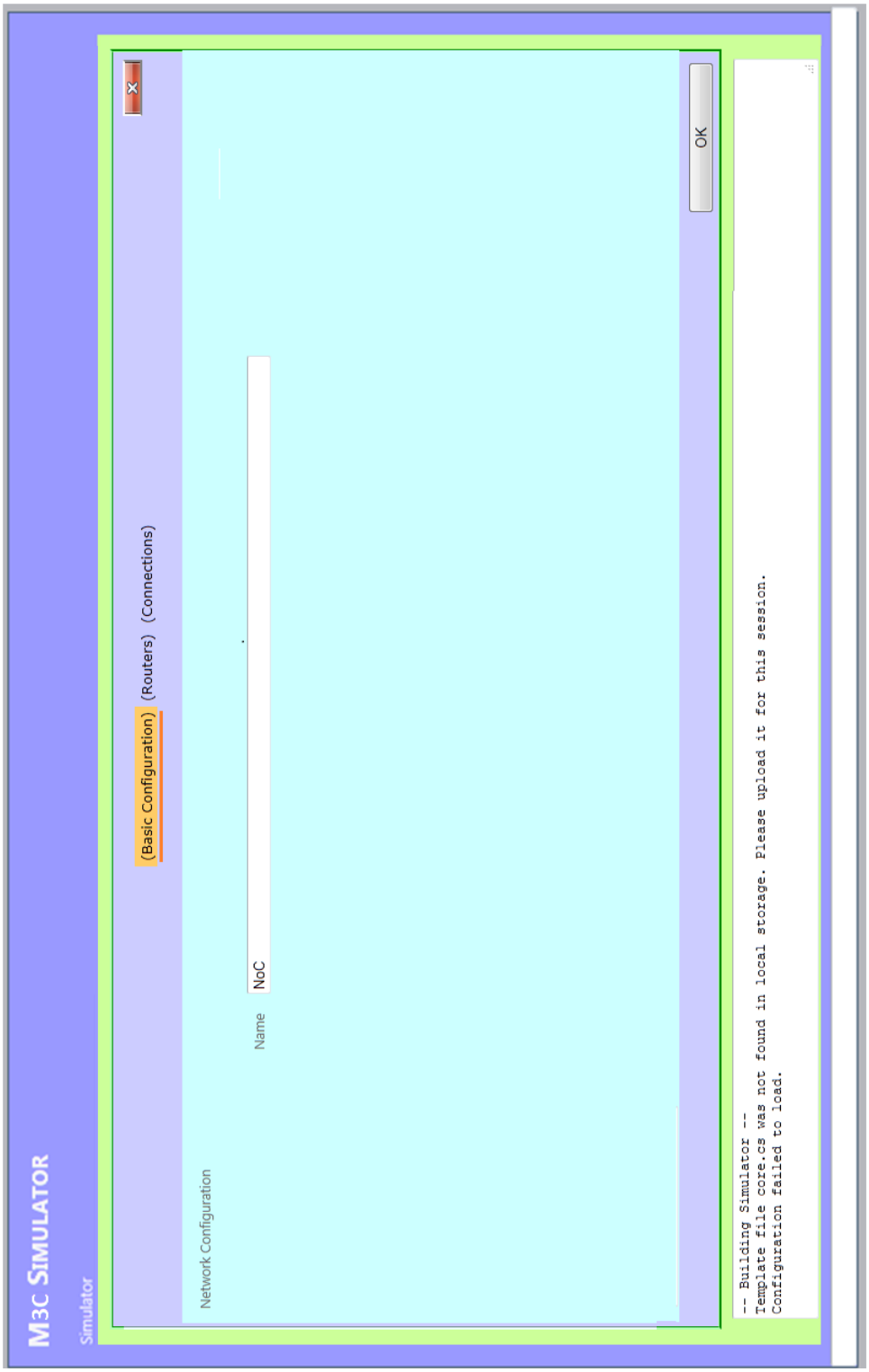


Fig. 3.12.: Network Configuration, Basic Configuration sub-screen

3.3.2 Router Configuration

The Routers sub-screen allows a user to create and configure Router types. Routers types, as described before, are basically stripped-down Core types.

A Router type may be added by putting a name in the Router Types Input text box and pressing the New Router Type button. The Router is added to the Router Types list box with the specified name. By selecting a Router type and then pressing the Delete Router Type button, a user may eliminate the selected Router type from the Router Types list box and all its associated information and configuration data along with any references to it in other parts of the simulation.

When a Router type is selected, the information or configuration related to that Router type is loaded. Router names can be changed in the Name text box. Like the Core types, Router types have to be instantiated so an Instance text box needs to be set to indicate how many Routers of a given type are to be created and run.

Each Router type as shown in Figure 3.13 has an input and output list box. These list boxes are both like and unlike the NoC Interfaces list boxes in the Resource Configurations screen. Names of input and output buffers or queues can be set here. For each buffer or queue, a name is required and then a new buffer or queue may be added using the Add Input or Add Output buttons. If an input or output buffer or queue is selected, it may be deleted by pressing the Delete Input or Delete Output buttons. Users may also specify the size of the buffer or queue and if it is an array. The type of variable each buffer or queue contains is also set.

The Source Code text box allows a user to describe the behavior of a Router type. This section is usually meant to code in how the input and output buffers or queues work and interact with other Router types or Core types. Description of Router type's behavior is used when Router instances are connected. The input and output buffers are created for each of the Router instances. The next section will describe how a Router instance connection is described.

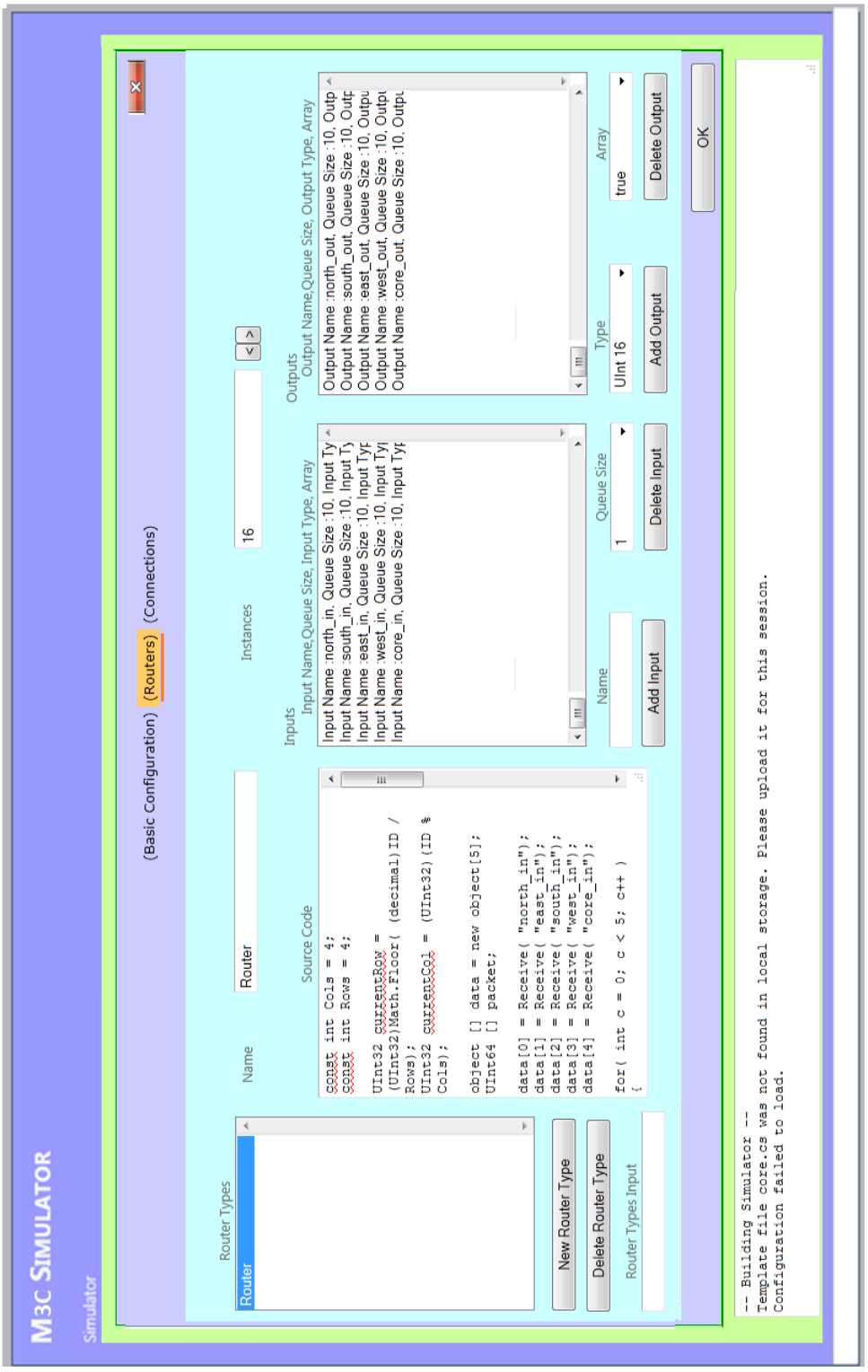


Fig. 3.13.: Network Configuration, Router Configuration sub-screen

3.3.3 Connection Configuration

Connections play a large part in the networking and updating of NoCs in both the grandfather simulator and M3C simulator as discussed in Chapter two. As described earlier in detail, connections contain information about one-way channels between either Core objects and another Core object, Core objects and Router objects, or Router objects and another Router object.

Figure 3.14 shows the Connections sub-screen. One can see from the example that the typical connections in a simulation can be numerous and complex. As stated earlier, a user can create a fully connected system and when the system consists of 64 Core objects with corresponding Router objects, the number of connections can become very large. At the student level, these connections can be kept small while a researcher or industry designer can create as much inter-complexity as they desire. For extremely larger systems, it may also be easier to go into the saved XML files and modify them instead.

A user may add a new Connection by pressing the Add Connection button. If a connection is selected, a user may delete a connection by selecting the Delete Connection button, resetting the Connection list box.

Each Connection has a number of configured parts to it. First, one can see by looking at the Connection list box that a type is placed before an arrow that points to another type. These types are any Core types in the current simulation and any Router types in the current Network.

When a user selects a Connection, they can select from a drop-down menu called Source Resource that is propagated with all the current simulation's Core types and all the current Network's Router types. When this is done, the user can indicate which instance of that type is being considered. The Source Output drop-down box is then propagated from the output buffers or queues that the Router and Core types have defined. The Destination Resource, Destination instance, and Destination Input

work very similarly to the Source version of them, with the inputs from the indicated resource being propagated.

After selecting a Source and Destination Resource for a connection, the Connection list box will display the name of the Source Resource with an arrow pointing to the Destination Resource. The arrow indicates the link or connection direction. By choosing an instance for the Resource, a particular link between two Resource instances can be defined. If a Router type has an output buffer named north_link and a Core type has an input buffer named in_link, then a user may specify that the north_link is to send data to the in_link. By choosing an instance of the Router type to be connected to an instance of the Core type, the user finishes a single connection and defines the buffers and objects involved. These steps are repeated for all the connections that are desired. Again, when an instance of a described Router or Core type is created, it has its own individual input and output buffers not tied to any other instance until the Connection configuration is made.

Additionally, a transmission delay can be set for a given Connection.

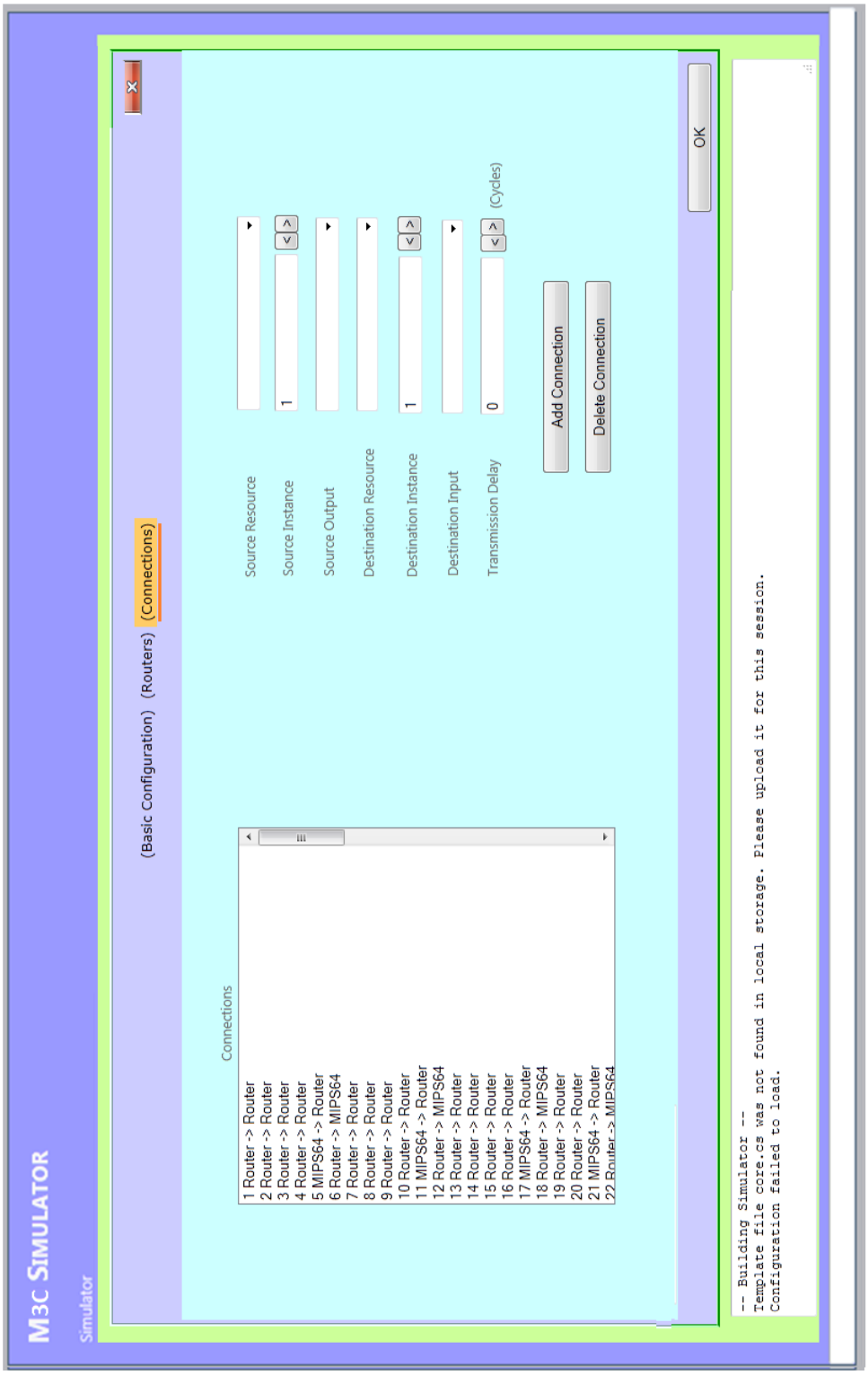


Fig. 3.14.: Network Configuration, Connection Configuration sub-screen

3.4 Simulation Execution

3.4.1 Overview

Once the simulation is configured and built correctly, the user may select the Run Simulation menu option. When this occurs, the Simulation Run screen comes up and is propagated with information about all the Core objects that will be run. This section describes the controls for this step and how they work as the simulation progresses.

3.4.2 Simulator Run Page

When a simulation is properly configured and built and the Simulation Run screen is open, two list boxes (see Figure 3.15) appear that show propagated information about all the Core objects that will be run. We used two boxes to show the way the code refreshes information in a limited space.

The first list box, called the Core Status list box, gives information such as a Core object's individual instance ID, the current status of the core, and the cycle or tick that a Core instance is currently at. This data is updated as the Core instances execute.

The second list box, called the Simulation Status, corresponds to the Core Status list box. When one is selected, the corresponding index of the other is selected as well. This second list box shows more information including the core instance, update status, the number of instructions fetched, the number of cache hits, and the number of cache misses. This will later be expanded to include timing information when synchronization is reintroduced.

This screen also has a Debug text box through which users can monitor debugging information they put in the source code throughout the configuration's setup. This option may be disabled for faster simulation execution.

There are four buttons on the screen other than the Esc and OK buttons. The first button, called Start Sim, when pressed, initiates each of the Core instances to start executing. Each Core instance's data is updated to the screen at every preprogrammed time period. When the Stop Sim button is pressed, the simulation stops at the current point, updating the Core instances on the screen once last time.

When the Show Data Mem. button is pressed, a new sub-screen is opened that shows a list box containing the selected Core instance's memory information. This can be seen on the left side of Figure 3.16. The word number is shown with the corresponding Value shown in hexadecimal.

When the Show Registers button is pressed, a new sub-screen is opened that shows a list box containing the selected Core instance's register information. This can be seen on the right side of Figure 3.16. The register is indicated with the corresponding Value shown in hexadecimal.

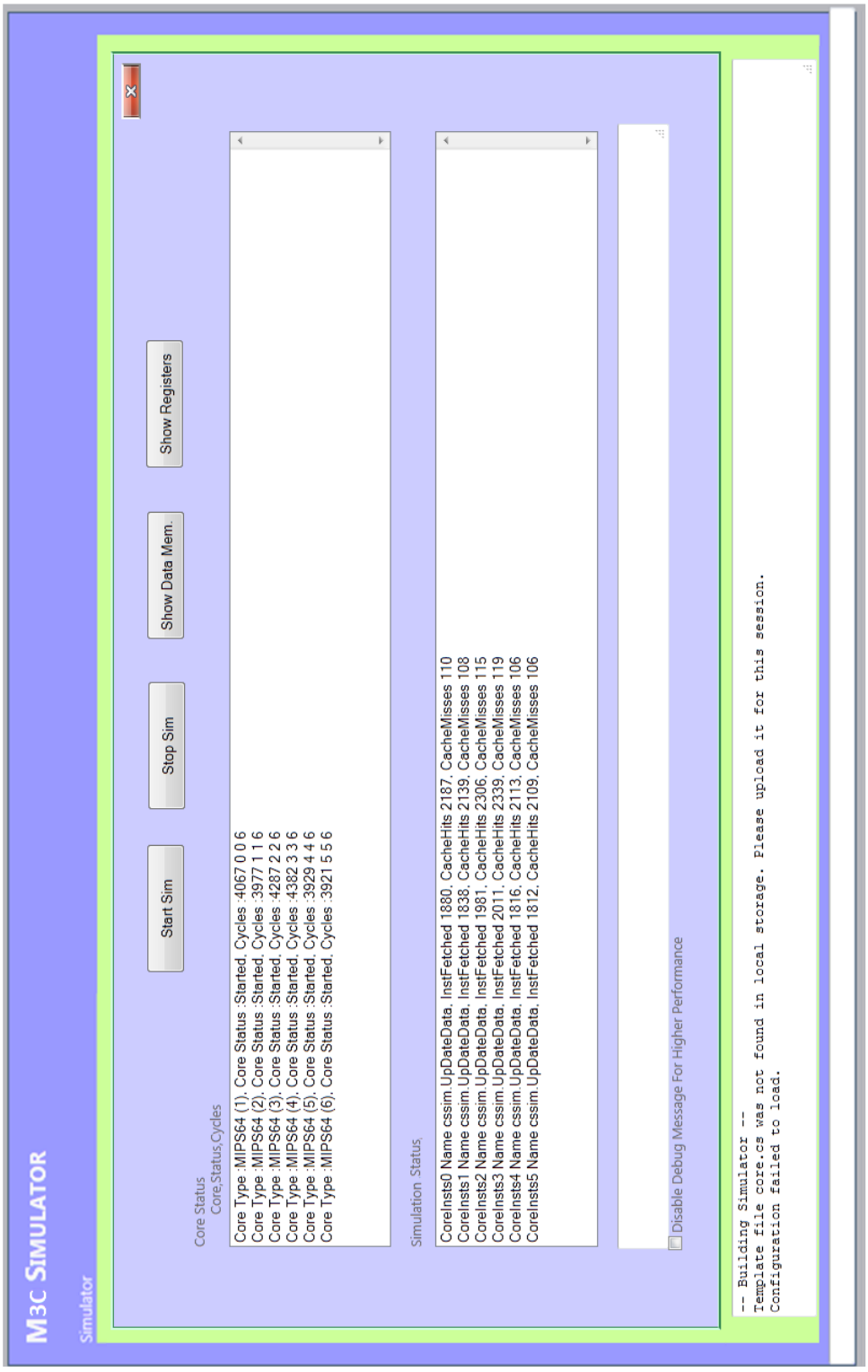


Fig. 3.15.: Simulator Run screen

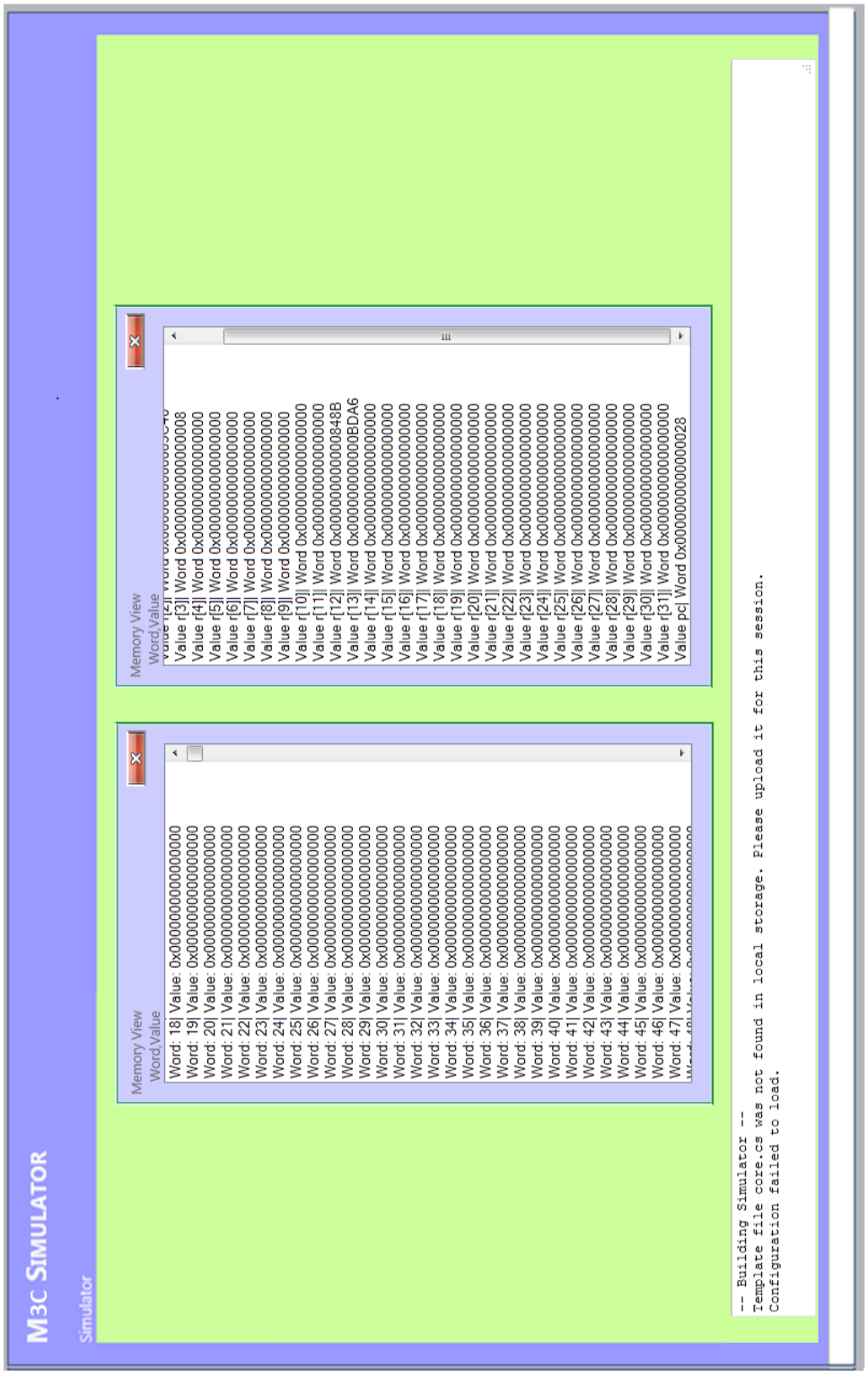


Fig. 3.16.: Memory and Register layout and display

3.5 Cloud Interfacing Tools

Azure has a number of controls that can be useful for teachers, researchers, and industry designers. You can indicate the number of Roles that are created. You can also use the SDK provided by Microsoft to check modifications to the simulator, shown in Figure 3.17.

The Azure SDK was used for testing and verification for this research and it is hoped that future researchers, when funding is provided, will publish the M3C simulator to the wider Azure Cloud Computing Platform.

In Figure 3.17, one can see a running deployment with one Web Role with a red dot and three Worker Roles with purple dots. Most testing and verifications were done with three Worker roles. The Azure SDK makes partitions in a computer and then runs a Windows Azure Compute Emulator and a Windows Azure Storage Emulator. The partitions keep past and current running deployment information, including local storage and Worker or Web Role instances.

Figure 3.18 shows how simple it is to create new instances of a Role. A user like a researcher only has to push a button to get more or less instances of a Role. If the ratio of Web Roles to Worker Roles needs to be modified, they will need to do it inside the code.

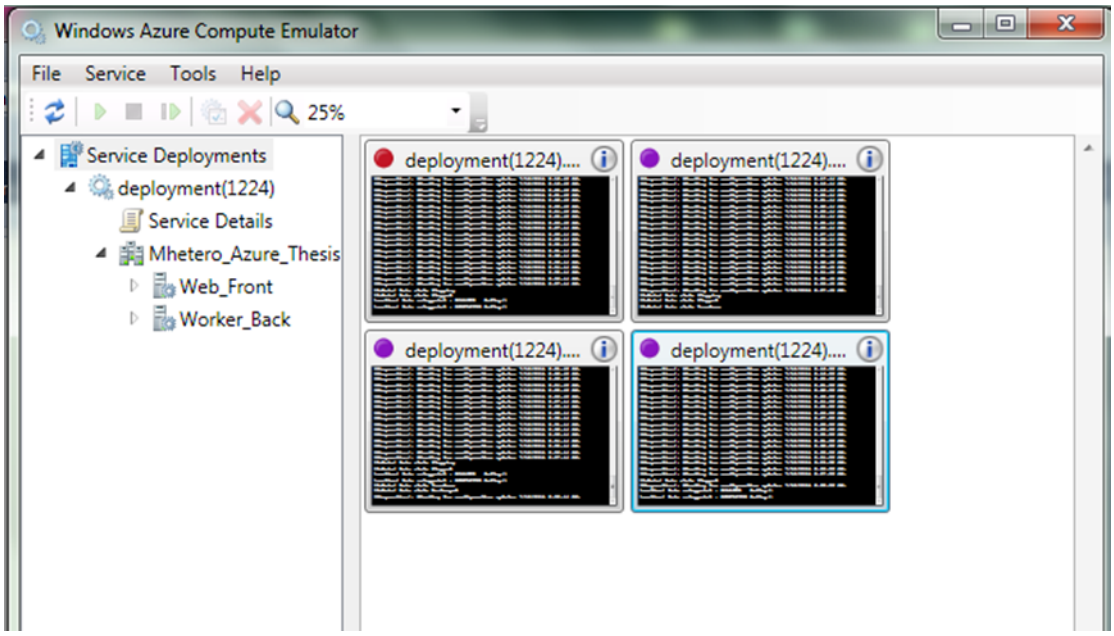


Fig. 3.17.: Azure Cloud Computing SDK tools

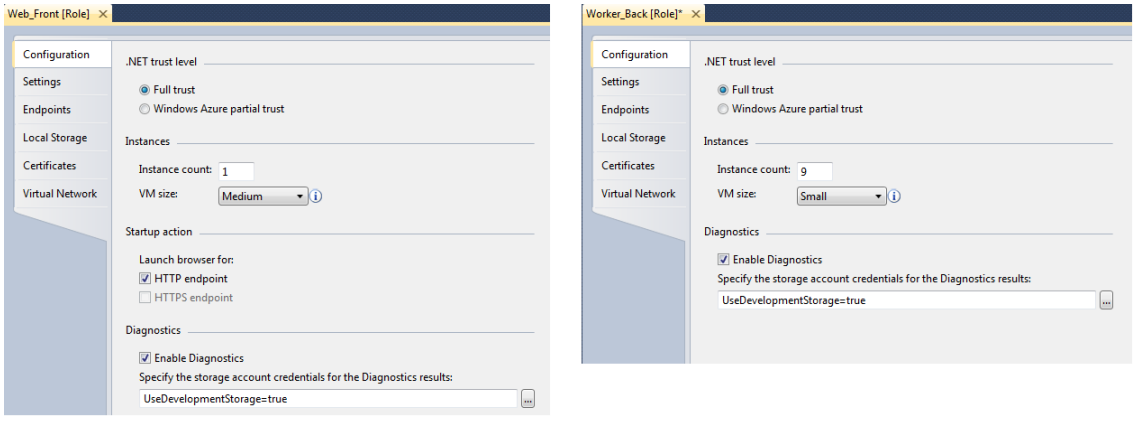


Fig. 3.18.: Azure Cloud Computing Role tools

4. SOFTWARE DESIGN

4.1 Simulator Layer

M3C can be best viewed as a set of layers. In each layer, there are different players and channels. In this section we will be describing the Simulator Layer diagram shown in Figure 4.1 and how each layer fits in and works with the whole simulator.

There are three section layers that make up the M3C simulator. These are the User Interface section layer, the Web Role as Website section layer, and finally the Core Part Worker Roles section layer. The first section layer is made up of the users, their device, and a web browser. A user connects to the Internet using a device that can gain access to it and can run simple web browsing applications. They then navigate to a given URL with a specific port number, gaining access to the Azure Cloud Computing Platform and their own website interfaces.

The next section layer is the Web Role as Website. This section layer communicates with the User Interface section layer using XML simulation files to load a simulation. The user may also load template files, DLLs, program, and data files. If the user requests a save to occur, the Web Role allows a download of an XML simulation file to occur. The user also using the website GUI Interface layer can gain abstract control over all other layers.

The Web Role as Website section layer has itself three layers. The first layer is the GUI Interface layer and is the major interface between a user and the rest of the simulator's layers. When the user wishes to configure a given simulation or create a new one and then configure it, an intermediate layer with four sub-layers is used. This layer is referred to as the Configurations/Build layer.

Within the Configuration/Build layer, a Simulation, Resource, Network, and Control sub-layers can be accessed. The Simulation sub-layer allows configurations as well

as eventual building of a set of Core types and Networks that are loaded from the current simulation. This information is kept in a temporary class object that is set and then destroyed and reloaded for each simulation.

The Resource and Network sub-layers work with the Simulator sub-layer. They hold the information to configure class objects of Networks, Router types, and Core types. These are stored within the Simulation sub-layer's class object.

The Control sub-layer acts as an intermediate between the next Distribution-Collection-Control layer and the Configuration/Build layer. For instance, it relates to the next layer what Core types should be created, how many of each, what Worker Roles they are to preside in, and how they are to be handled. They also lay out maps so that later layers and feedback layers can decode or translate what and where information is coming from.

The next layer is the Distribution-Collection-Control layer. As its name implies, it handles three major tasks and acts as a go-between middle layers and the last section layer. Its first job is to distribute information it receives from the Configuration/Build layer including any control messages to each of the Worker Roles. Its next task is to collect and pass back information to the Simulation Display layer. The last task it performs is to pass back additional up-date control information from the Control sub-layer to the Worker Roles. The layer takes the maps and other control information that were set up in the Control sub-layer and uses them to perform the other two tasks.

The Simulation Display layer is the feedback layer of this section layer and as it is indicated in Figure 4.1, it deals with Running, Started, and Stopped conditions. This means at all times when a simulation is either running, started, or stopped (note that these statuses will be examined in later sections), this layer will be considering and feeding the GUI Interface layer information.

The last section layer, Core Part Worker Roles, is really composed of a number of Worker Roles all having the same layered structure. The Destruction-Collection-Control layer interacts with each individual Worker Role's RX and TX layers. This

communication is done using Windows Communication Foundation TCP bindings over internal endpoints, which will be described in detail in Section 4.2.1 [31, 32].

Each Worker Role has an Rx and Tx layer which take in information and command messages and give out information and response messages. The next layer is a Code Graph creation layer in which by the use of dynamic programming and the configuration's setup in the Web Role, a code graph is created [32, 33]. For the case of the thesis, a code graph comes from a CodeDom graph [32, 33]. It is an object graph made up of CodeDom elements in a structure similar to a tree. CodeDom will be described more in Secion 4.2.2. The code graph is then moved to the next Graph Compiler layer where it is compiled, assembled, and put into a set of threads [32–40]. Upon the control message to start the threads, the Run and Network Communications layers start each thread, then update the other Worker roles, and prepare updated information for the Web Role to pick up.

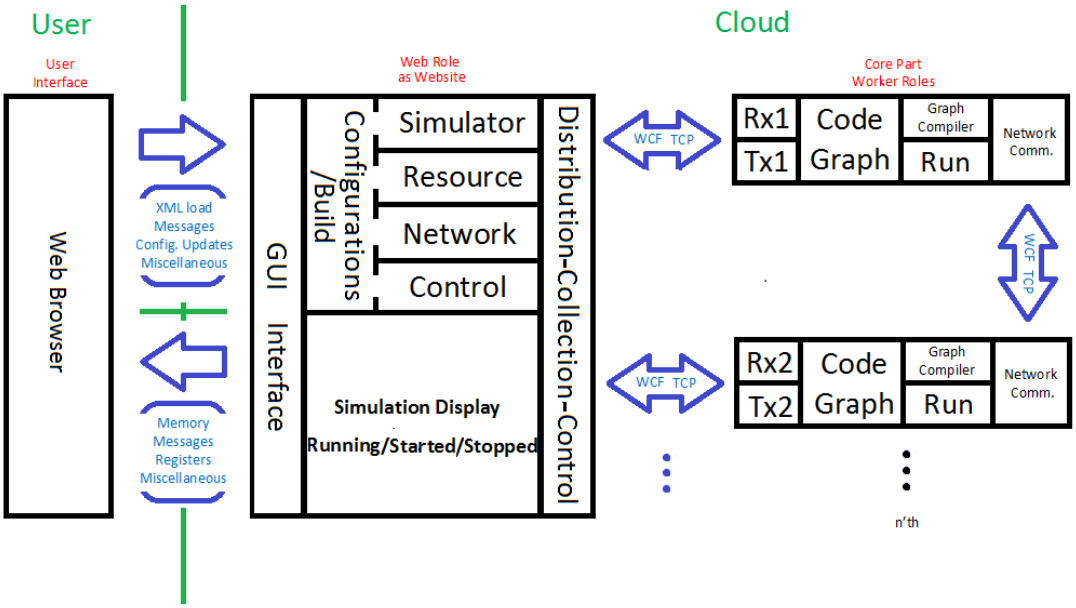


Fig. 4.1.: Layered Design for the M3C simulator

4.2 Discussion of Programming Specifics

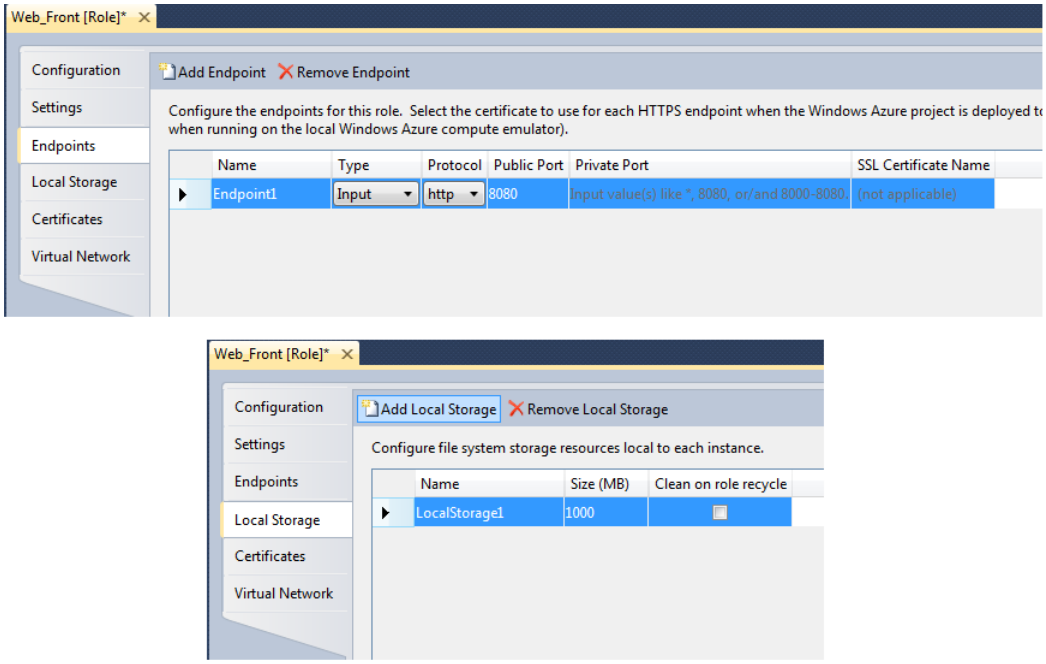
4.2.1 Windows Communication Foundation

Windows Communication Foundation or WCF is a way of looking at communications that Microsoft developers created [31]. In the case of the M3C simulator, WCF is a good way to set up communications between Web Roles and Worker Roles. Each Role, as mentioned earlier, can use External or Internal Endpoints and set them up as shown in Figure 4.2. Figure 4.2 also shows that each Web Role and Worker Role have local storage of 1000 MBs.

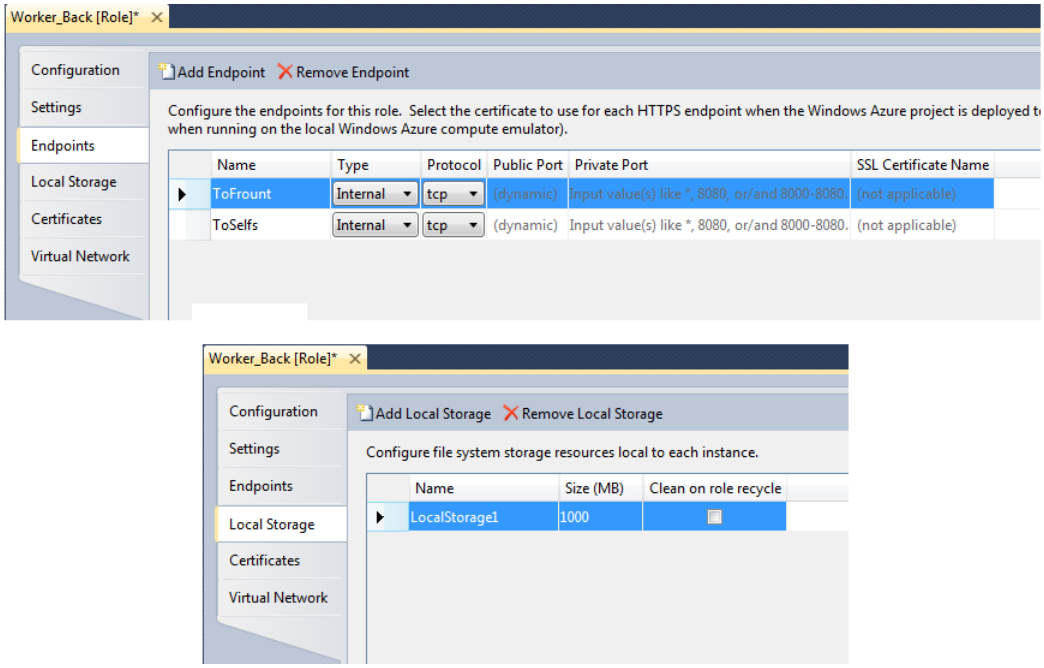
To create and control Endpoints in the M3C simulator, the following steps have to be taken. First, an Internal Endpoint in the Worker Roles is used to host with a TCP Binding to create a channel as shown in Figure 4.3. This code snippet from the M3C simulator shows a variable named host being created as a ServiceHost object using the ToFront Endpoint [41, 42]. Next, a NetTcpBinding is created with no security due to the internal nature of the communication [43]. Finally, we use our host to set up a contract using an Interface that both the Worker Role and the Web Role share our Binding [32, 41, 42]. We set up this host using the current Worker Role ID and Endpoint even though each Worker Role has the same Endpoint name [44].

On the Web Role side, a temporary connection is made when communication is desired. This process is shown in Figure 4.4. This is the same as the hosting of the service; however in this case we are setting up multiple channels using an interface shared between the Web Role and Worker Roles. Using the RoleEnvironment to cycle through each Worker Role, each channel is set and the interface contracts are used [44, 45]. NetTcpBindings are still being used, as one can see, with no security set due to the internal nature.

For more information, it is strongly advised that one visits the MSDN web page at http://msdn.microsoft.com/en-us/library/aa480210.aspx [45].



(a) Web Role Endpoint and storage



(b) Worker Role Endpoint and storage

Fig. 4.2.: Azure Cloud Computing Configuration tools

```
var baseaddress = string.Format("net.tcp://{0}",
    RoleEnvironment.CurrentRoleInstance.InstanceEndpoints["ToFrount"].IPEndpoint);

var host = new ServiceHost(typeof(Cores_Routers_To_Control), new Uri(baseaddress));

var binding = new NetTcpBinding(SecurityMode.None);
binding.MaxReceivedMessageSize = 1000000;
binding.ReaderQuotas.MaxStringContentLength = 1000000;

host.AddServiceEndpoint(typeof(ICores_Routers_To_Control), binding, "TalkToFrount");
```

Fig. 4.3.: WCF Endpoint hosting

```
public void OnReset()
{
  ICores_Routers_To_Control[] channel = new ICores_Routers_To_Control[3];

  string roleID = RoleEnvironment.CurrentRoleInstance.Id;
  string[] par = roleID.Split('.');
  instancenum = Convert.ToInt32(par[par.Length - 1]);

  for (int c = 0; c < RoleEnvironment.Roles["Worker_Back"].Instances.Count; c++)
  {
    if (c >= instancenum * 3 && c < instancenum * 3 + 3)
    {
      var factory = new ChannelFactory<cssim.ICores_Routers_To_Control>(new NetTcpBinding(SecurityMode.None));

      EndpointAddress holder = new EndpointAddress(string.Format("net.tcp://{0}/TalkToFrount",
          RoleEnvironment.Roles["Worker_Back"].Instances[c].InstanceEndpoints["ToFrount"].IPEndpoint));

      channel[c - instancenum] = factory.CreateChannel(holder);

      channel[c - instancenum].Templats(null, null, 0, 0, 0);
      channel[c - instancenum].SetControlState(ControlStatus.Reset);

    }
  }
}
```

Fig. 4.4.: WCF Endpoint Connection

4.2.2 Dynamic Programming

Dynamic Programming plays a large role in the creation and running of Core objects and Router objects. Dynamic Programming means that code can be prepared, compiled, and added to a program after that program is already running.

Figure 4.5 shows a simple class and interface that can be used to run dynamically generated code. A string of code is created that, if run, will work in the current assembly environment. Next, a CompilerParameters object is created, and references to System.dll and the current assembly are added [33, 35]. A CSharpCodeProvider object is created and uses the CompilerParameters object with the string of code to be run [36].

The result of the compilation is given to a CompilerResults object [33,38]. If there are no errors from this compilation, the compiled assembly code can be put into an Assembly object [34]. The assembly is then used to create an instance of the class defined in the code string using the method CreateInstance, and cast as an interface that the new class created from the code string and the current program assembly use to interact [32, 34].

Using an interface type, a user can now run functions inside the newly created class that came from the code designed using the configuration.

```
public interface IHelloworld
{

    string Init(bool hi);
    string runit(bool hi);

}



public class Sim_Backup
{
 public bool start(bool hi)
 {
   CSharpCodeProvider codeProvider = new CSharpCodeProvider();

   IHelloworld[] same;
   Assembly[] coreAssemblies;

   coreAssemblies = new Assembly[0];
   coreAssemblies = new Assembly[1];
   same = new IHelloworld[0];
   same = new IHelloworld[1];

   CompilerParameters compilerParamters = new CompilerParameters();
   compilerParamters.IncludeDebugInformation = true;

   compilerParamters.GenerateInMemory = true;

   compilerParamters.ReferencedAssemblies.Add("System.dll");
   compilerParamters.ReferencedAssemblies.Add(Assembly.GetExecutingAssembly().Location);

   string hold = "using System; using System.Collections.Generic; using System.Text; usi

   //Compile Processor Code
   CompilerResults results
               = codeProvider.CompileAssemblyFromSource(compilerParamters, hold);
   if (results.Errors.Count == 0)
   {
       coreAssemblies[0] = results.CompiledAssembly;
       same[0] = (IHelloworld)coreAssemblies[0].CreateInstance("MainWorker.Class1",
               false);
       same[0].Init(true);
       return true;
   }
 }
}
```

Fig. 4.5.: Dynamic Programming basic example

4.3 State Diagram and Simulation Flow

To keep Web Roles communicating with Worker Roles in such a way that the Web Role could control each Worker role and know which execution point they are ready for, status messages and status states were created. The two primary statuses are Simulation Status and Control Status.

A Web Role uses control messages to set a Worker Role's Control Status so that the Worker Role knows what it should do at the earliest time available.

Both Web Roles and Worker Roles keep a Simulation Status; however Web Roles only use their Simulation Status to check against the Worker Role's Simulation Status. Simulation Status indicates the current stage or layer at which a Simulation is held.

Figure 4.6 shows the State Diagram as a combination of both the Web Role and the corresponding Worker Roles.

A list of the Simulation Statuses with a description of the events and related settings associated with them can be described as follows:

Uninitialized: No simulation has been loaded, an error has occurred, or the user has reset the simulator

Ready : Occurs after the simulator is built

- The Web Role uses the user configuration on the web page to make a template of Core types and Router types

- The Web Role sends a Build status to the Worker Roles with the Template files and any program files

- The Workers each uses the template files to build assemblies and sets simulation status to Ready

Running : Occurs after build and the user opens the Simulator Run interface

- Web Role opens the Simulator Run interface

- Web Role sends a Run message to the Worker Roles with information about what cores and routers they will instantiate

- Worker Role creates instances of the cores and routers it was assigned and sets them into threads

- Worker Role sets simulator state to Running

Started : When core threads are running and routers and cores are updating

- Web Role sends Start control message

- Worker Roles start their threads

- The cores update their data such as current memory and number of ticks

- Web Role gets data and posts it to screen in intervals

Stopped : when the user asked for a stop

Error : when an error occurs, mainly in building

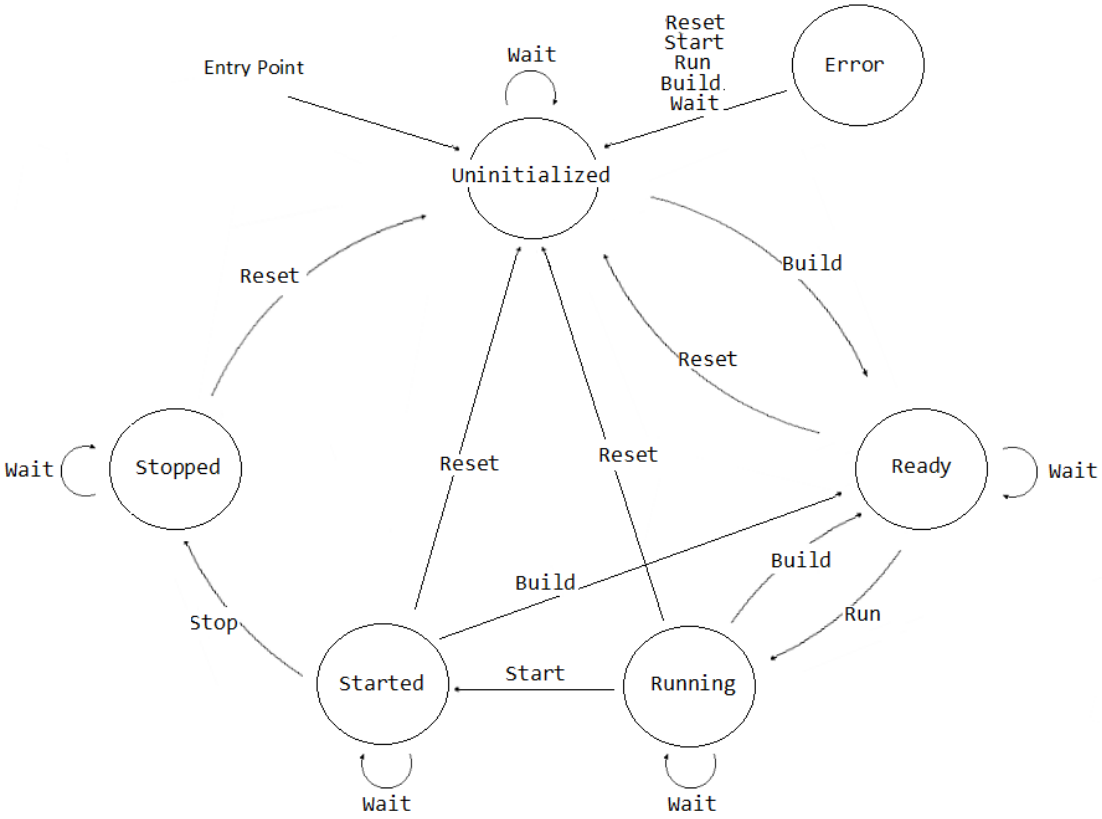


Fig. 4.6.: Control and Simulation State Diagram

5. SUMMARY AND CONCLUSION

5.1 Summary

In this thesis, several topics were discussed regarding the M3C simulator. The M3C simulator was defined as a many-core heterogeneous processor system simulator. The past grandfather simulator was analyzed and compared to the new M3C simulator, and the upgrades were discussed. We started the description of the M3C simulator by looking at the many-core simulator aspect, including the how resources are structured, designed, and interconnected. Then we went on to talk about how the M3C simulator is designed to leverage Azure Cloud Computing platform and both the design's structure and interconnectivity.

We examined and described each of the user's interfaces and how they work. The Setup interface and overall site layout were detailed. Then how the Resource Configuration section is used and its effects on Core types were touched upon. Furthermore, we discussed the Network Configuration section, and the effect and usages of both Router types and Connections in a Simulation. Next we talked about how a user interacts and receives data from the Simulation Run interface. We wrapped up the section about interfaces with a consideration of the cloud interface tools and how they can be used in the future.

We then examined some of the Software Design behind the M3C simulator. We discussed the simulator's layered architecture and how layers work together. Specific programming paradigms were examined that were used heavily in the simulator. Finally, we took a look at the M3C simulation from the perspective of the State Diagram.

5.2 Suggestions for Future Work

The following is a list of recommendations and thoughts for future work.

1 Final preparation for deployment to the Cloud

2 Redesigning Networking implementation for the simulator that does not rely so heavily on worker roles' threading speed

3 Try other simulator system on the cloud

4 Create more user friendly and informative output GUI

5 Create a set of scenarios for testing and teaching

6 Create a heterogeneous core type and later expand into a many resources system

5.3 Conclusion

The M3C simulator discussed in this thesis provides several contributions in the task of improving many-core heterogeneous simulators. The simulator's design leveraged Azure Cloud Computing Platform resources in such a way to make simulations partial and worthwhile to emulate. The M3C simulator's interface system was made familiar to users of the Mhetero simulator. Simulation files, template files, DLLs, program files, and data files can be used in both the M3C simulator and its grandfather simulator, the Mhetero simulator. The M3C simulator allows a user to access the simulator from a range of devices of varying computing power. While the simulator is far from being ready for true researcher or industry usage, this thesis is an important set of steps in the right direction, and it lays the ground works for future researchers that will pick up where it left off to easily understand and be able to change the current simulator. The designs are sound and, if perfected, will be useful for future work. In short, all of the objectives of the thesis have been completed.

In general, this thesis shows that simulator systems using such Cloud Computing Platforms as Windows Azure are not only practical but also both time and cost

effective. Researchers and teachers should in the future try and use such resources not only in many-core and heterogeneous system simulation, but in many more areas.

LIST OF REFERENCES

LIST OF REFERENCES

[1] R. Kumar, D. Tullsen, and N. Jouppi, "Heterogeneous chip multiprocessors," *Computer*, vol. 18, Nov. 2005.

[2] H. Meyr, "Heterogeneous mp-soc–the solution to energy-efficient signal processing," in *Multiprocessor SoC MPSoC Solutions/Nightmare*, Design Automation Conference, 2004.

[3] Intel, "Futuristic intel chip could reshape how computers are built." http://www.intel.com/pressroom/archive/releases/2009/20091202comp_sm.htm. Visited Jan. 2010.

[4] AMD, "The future is fusion — amd." http://sites.amd.com/us/fusion/Pages/index.aspx. Visited Jan. 2010.

[5] W. Wolf, "The future of multiprocessor systems-on-chips," in *Multiprocessor SoC MPSoC Solutions/Nightmare*, Design Automation Conference, 2004.

[6] M. Yourst, "Introducing ptlsim." http://www.ptlsim.org/. Visited July 2011.

[7] "M5." http://www.m5sim.org. Visited Jan. 2010.

[8] "bochs: The open source IA-32 emulation project." http://bochs.sourceforge.net/. Visited Jan. 2010.

[9] J. Emer, P. Ahuja, and E. Borch, "Asim: A performance model framework," *Computer*, pp. 68–76, 2002.

[10] "Gxemul." http://gxemul.sourceforge.net/. Visited Jan. 2010.

[11] P. Magnusson and et al., "Simics: A full system simulation platform," *Computer*, vol. 35, pp. 50–58, 2002.

[12] N. Honarmand, H. Sohofi, M. Abbaspour, and Z. Navabi, "Processor description in apdl for design space exploration of embedded processors," *Proc. EWDTS*, 2007.

[13] A. Halamb, "Expression: A language for architecture exploration through compiler/simulator retargetability." http://www.cs.ucr.edu/ vahid/courses/269_w00/date99_dutt.pdf, 1999.

[14] M. Reshadi, P. Mishra, N. Bansal, and N. Dutt, "Rexsim: A retargetable framework for instruction-set architecture simulation." http://www.cecs.uci.edu/technical_report/TR03-05.pdf. Visited July 2011.

[15] G. Hadjiyiannis and et al., "ISDL: An instruction set description language for retargetability," *In Proc. Design Automation Conference*, pp. 299–302, 1997.

[16] C. Barnes, P. Vaidya, and J. Lee, "An XML-based ADL framework for automatic generation of multithreaded computer architecture simulators," *Computer Architecture Letters*, vol. 8, Apr. 2009.

[17] C. Barnes and J. J. Lee, "A dynamically configurable discrete event simulation framework for many-core chip multiprocessors." http://www.intechopen.com/source/pdfs/11537/InTech-A_dynamically_configurable_discrete_event_simulation_framework_for_many_core_chip_multiprocessors.pdf.

[18] J. E. Miller, H. Kasture, G. Kurian, C. GruenwaldIII, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal, "Graphite: A distributed parallel simulator for multicores," *HPCA-16*, 2010.

[19] Intel, "Intel products processors 2nd generation intel core i7 processor." http://www.intel.com/products/processor/corei7/index.htm. Visited June 2011.

[20] AMD, "AMD phenom II processors." http:///www.amd.com/us/products/desktop /processors/phenom-ii/Pages/phenom-ii.aspx. Visited June 2011.

[21] IBM, "IBM power7 processors." http://www.03.ibm.com/systems/power/news/announcement/20100209_annc.html. Visited June 2011.

[22] Vega, "Products, vega appliance, overview.." http://www.azulsystems.com/products/vega/overview. Visited June 2011.

[23] IBM, "The cell architecture." http://domino.watson.ibm.com/comm/research.nsf /pages/r.arch.innovation.html. Visited June 2011.

[24] XILINX, "Configuration for virtex-4 fpga." http://www.xilinx.com/products/design_resources/config_sol/v4/config_v4.htm. Visited June 2011.

[25] L. Xue, Y. Gao, and J. Fu, "A high performance 3d interconnection network for many-core processors," *Computer Engineering and Technology (ICCET), 2010 2nd International Conference*, April 2010.

[26] D. Patterson and J. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann, 2004.

[27] NIST, "NIST cloud computing program." http://www.nist.gov/itl/cloud/index.cfm. Visited June 2011.

[28] MSDN, "Xmltextwriter class (system.xml)." http://msdn.microsoft.com/en-us/library/system.xml.xmltextwriter.aspx. Visited Jan. 2010.

[29] MSDN, "Xmldocument class (system.xml)." http://msdn.microsoft.com/en-us/library/system.xml.xmldocument.aspx. Visited Jan. 2010.

[30] MSDN, "Xmlnode class (system.xml)." http://msdn.microsoft.com/en-us/library/system.xml.xmlnode.aspx. Visited Jan. 2010.

[31] MSDN, "Windows communication foundation." http://msdn.microsoft.com/en-us/library/ms735119Visited June 2011.

[32] MSDN, "interface (c# reference)." http://msdn.microsoft.com/en-us/library/87d83y5b.aspx. Visited Jan. 2010.

[33] MSDN, "Codedomprovider compileassemblyfromsource method." http://msdn.microsoft.com/en-us/library/system.codedom. compiler.codedomprovider.compileassemblyfromsource.aspx. Visited Jan. 2010.

[34] MSDN, "Assembly class (system.reflection)." http://msdn.microsoft.com/en-us/library/system.reflection.assembly.aspx. Visited Jan. 2010.

[35] MSDN, "Compilerparameters class (system.codedom.compiler)." http://msdn.microsoft.com/en-us/library/system.codedom.compiler. compilerparameters.aspx. Visited Jan. 2010.

[36] MSDN, "Csharpcodeprovider class." http://msdn.microsoft.com/en-us/library/microsoft.csharp.csharpcodeprovider(VS.85).aspx. Visited Jan. 2010.

[37] MSDN, "Compiling to msil." http://msdn.microsoft.com/en-us/library/c5tkafs1

[38] MSDN, "Compilerparameters referencedassemblies property (system.codedom.compiler)." http://msdn.microsoft.com/en-us/library/system.codedom.compiler.compilerparameters.referencedassemblies.aspx. Visited Jan. 2010.

[39] MSDN, "Thread.sleep method (int32) (system.threading)." http://msdn.microsoft.com/en-us/library/d00bd51t.aspx. Visited Jan. 2010.

[40] MSDN, "Thread class (system.threading)." http://msdn.microsoft.com/en-us/library/system.threading.thread.aspx. Visited Jan. 2010.

[41] MSDN, "Servicehost." http://msdn.microsoft.com/en-us/library/system. servicemodel.servicehost.aspx. Visited June 2011.

[42] MSDN, "Endpoint." http://msdn.microsoft.com/en-us/library/system.net. endpoint.aspx. Visited June 2011.

[43] MSDN, "Binding." http://msdn.microsoft.com/en-us/library/system. windows.data.binding.aspx. Visited June 2011.

[44] MSDN, "Roleenvironment." http://msdn.microsoft.com/en-us/library/microsoft.windowsazure.serviceruntime.roleenvironment.aspx. Visited June 2011.

[45] MSDN, "WCF overview." http://msdn.microsoft.com/en-us/library/aa480210.aspx. Visited June 2011.