**ОП: «Информатика и вычислительная техника»**

**Уровень образования: бакалавриат**

**О Т Ч Е Т**

| **по** | **производственной** | **практике** |
| --- | --- | --- |
| *(производственной / научно-исследовательской / преддипломной)* | | |

| Выполнил студент гр. | БИВ185 |
| --- | --- |
| Михайлов Ярослав Алексеевич | |
| *(ФИО)* | |
|  | |
| *(подпись)* | |

| **Проверили:** |  |
| --- | --- |
| Ассистент Американов А. А. | |
| *(должность, ФИО руководителя от организации)* | |
|  |  |
| *(оценка)* | *(подпись)* |
|  |  |
| МП | *(дата)* |
| Доцент ДКИ Варнавский А.Н. | |
| *(должность, ФИО руководителя от факультета)* | |
|  |  |
| *(оценка)* | *(подпись)* |
|  |  |
|  | *(дата)* |

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 3](#_heading=h.30j0zll)

[1](#_heading=h.1fob9te) Модели сетей на кристалле 4

[2](#_heading=h.3znysh7) Модель gpNoCSim++ 5

[3](#_heading=h.2et92p0) Работа с симулятором gpNoCSim++ 8

[4](#_heading=h.tyjcwt) Сравнение симуляций различных топологий с помощью gpNocSim++ 11

[5](#_heading=h.3dy6vkm) Интеграция модели в единую среду моделирования 12

[ЗАКЛЮЧЕНИЕ 14](#_heading=h.1t3h5sf)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 15](#_heading=h.4d34og8)

[ПРИЛОЖЕНИЕ А 16](#_heading=h.2s8eyo1)

# **ВВЕДЕНИЕ**

Целью прохождения практики является изучение и модификация высокоуровнего симулятора gpNocSim++ для добавления в единую среду высокоуровневого моделирования СтнК.

Задачи практики – изучить принципы топологического подхода при проектировании сетей на кристалле, провести обзор различных моделей для проектирования СтнК, изучить симулятор gpNocSim++, провести моделирование различных топологий в симуляторе gpNocSim++, модифицировать симулятор gpNocSim++ для добавления в единую среду высокоуровневого моделирования СтнК.

Содержание практики:

1. Применение топологического подхода при проектировании СтнК
2. Обзор различных моделей для проектирования СтнК
3. Запуск и анализ работы симулятора gpNocSim++
4. Моделирование различных топологий в симуляторе
5. Интеграция gpNocSim++ в единую среду симуляции сетей на кристалле

Результаты практики:

1. Предоставление схемы симулятора gpNocSim++
2. Предоставление результатов моделирования различных топологий СтнК с помощью gpNocSim++
3. Предоставление кода модифицированного gpNocSim++
4. Предоставление отчета по практике

# **Модели сетей на кристалле**

Вследствие развития полупроводников многоядерные вычислительные системы стали нормой – едва ли сейчас можно кого-то удивить наличием четырех- или восьмиядерного процессора, для серверных же решений стандартом стали процессоры с минимум 32 или 64 ядрами. При этом наличие столько большого количества вычислительных блоков на одном кристалле требует хорошо организованной системы обмена сообщений между ними. Долгое время основой коммутации между блоками служила шинная архитектура, то есть все блоки подключены к общей шине. Но подобный подход исчерпывает себя при большом количестве IP-блоков: возрастают задержки при обмене сообщениями и растет энергопотребление всей системы в целом. На данный момент самым перспективным решением этой проблемы, позволяющим размещать на одном кристалле десятки и сотни ядер, являются сети на кристалле.

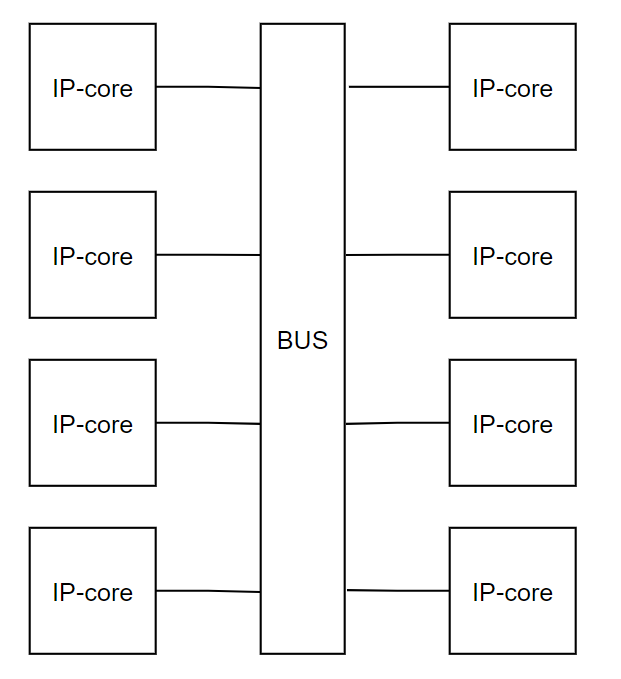


Рисунок 1. Шинная архитектура.

Сеть на кристалле (СтнК) – способ обмена сообщениями между вычислительными блоками на одном кристалле на основе коммутации пакетов. В СтнК каждый IP-блок процессора соединен с маршрутизатором, который и отвечает за обмен сообщениями между этим IP-блоком и остальными, которые также в свою очередь подключены к соответствующим маршрутизаторам. Сами маршрутизаторы же между собой соединены в сеть и обмениваются между собой информацией при помощи пакетов. Подобные сети позволяют с легкостью решать проблему масштабируемости многоядерных вычислительных систем.

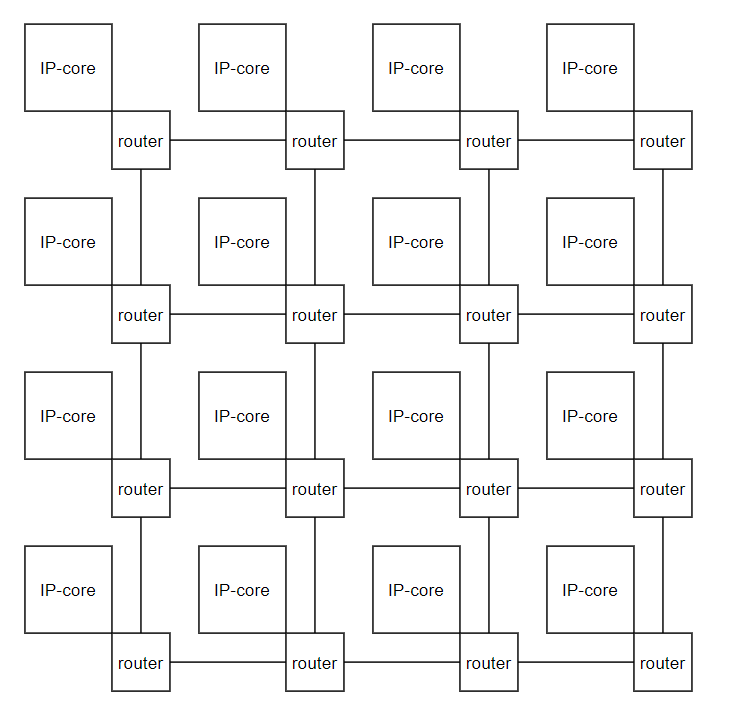


Рисунок 2. СтнК с топологией Mesh2D(4,4)

Однако разработка сетей на кристалле требует решения множества вопросов: необходимо обеспечить заданную пропускную способность сети, минимизировать задержки между отправкой и получением пакетов, снизить общее энергопотребление сети. Так как СтнК представляют собой достаточно комплексную структуру, на каждом из этапов разработки необходимо оценивать те или иные решения с помощью различных моделей. В связи с этим можно выделить несколько видов моделей СтнК: аналитические, модели высокого уровня, низкоуровневые модели и модели смешанного типа. В данной работе была рассмотрена высокоуровневая модель gpNoCSim++.

# **Модель gpNoCSim++**

Более подробно рассмотрим структуру модели gpNoCSim++. Данная модель является симулятором общего назначения для сетей на кристалле, написанном на языке высокого уровня (Java 1.4), и имеет собственный пользовательский интерфейс (рис. 3).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3. Пользовательский интерфейс gpNoCSim++.

В рассматриваемой модели имеется встроенная поддержка нескольких самых популярных топологий сетей на кристалле: fat tree, mesh, torus, extended fat tree, octal, WK-recursive. Таким образом gpNoCSim++ предоставляет возможность разработчикам выбрать наиболее подходящую топологию из целого перечня поддерживаемых. Однако теоретически в модель можно добавить поддержку и пользовательских топологий, реализовав несколько интерфейсов и переопределив несколько классов.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 4. Список поддерживаемых топологий.

Симулятор gpNoCSim++ для моделирования работы СтнК оперирует понятиями коммутатор (свитч) и маршрутизатор (роутер). Так, каждый коммутатор имеет ряд портов для подключения с соседними коммутаторами и IP-блоками. Каждый порт в свою очередь включает в себя два набора буферов – входной и выходной, маршрутизатор и контроллер. Маршрутизатор расшифровывает заголовок и определяет выходной порт. Контроллер же разделен на два отдельных контроллера: по одному для входного и выходного каналов.

Принцип работы каждого wormhole-коммутатора в модели следующий: на вход сначала поступает заголовок пакета, содержащий информацию о назначении данного пакета; после дешифровки заголовка, коммутатор просто устанавливает соединение со следующим в маршруте коммутатором; каждый же участок полезной нагрузки пакета просто напрямую перенаправляется по только что установленному соединению, что позволяет сохранять порядок данных внутри пакета. Если же в процессе дешифровки заголовка выясняется, что требуемое соединение крайне загружено в данный момент времени, все части пакета остаются простаивать на своих текущих местах.

Рассмотрим теперь порядок работы модели (рис. 5):

1. Модель считывает входные параметры из файла *nocSimParameter.txt*
2. Если в файле конфигурации присутствуют еще не рассмотренные сети, то модель приступает к их моделированию, в противном случае выполнение прекращается.
3. Строится рассматриваемая сеть и задаются ее ключевые параметры.
4. Если симуляция сети завершена, то ее результаты записываются в выходной файл (*nocSimOutput.txt*), иначе начинается итеративная симуляция работы сети.

Более подробно взглянем на порядок действий модели внутри каждой итерации цикла симуляции:

1. Для каждого IP-блока сообщения из блока подаются в соответствующий коммутатор.
2. Для каждого коммутатора создаются требуемые входящим трафиком соединения, входящий траффик из входного буфера перемещается в выходной буфер, выходной трафик перемещается из выходного буфера этого коммутатора во входной буфер соседнего IP-блока или коммутатора.
3. Для каждого IP-блока данные из входного буфера поступают в сам блок.
4. Для каждого блока и коммутатора обновляются флаги текущих состояний.

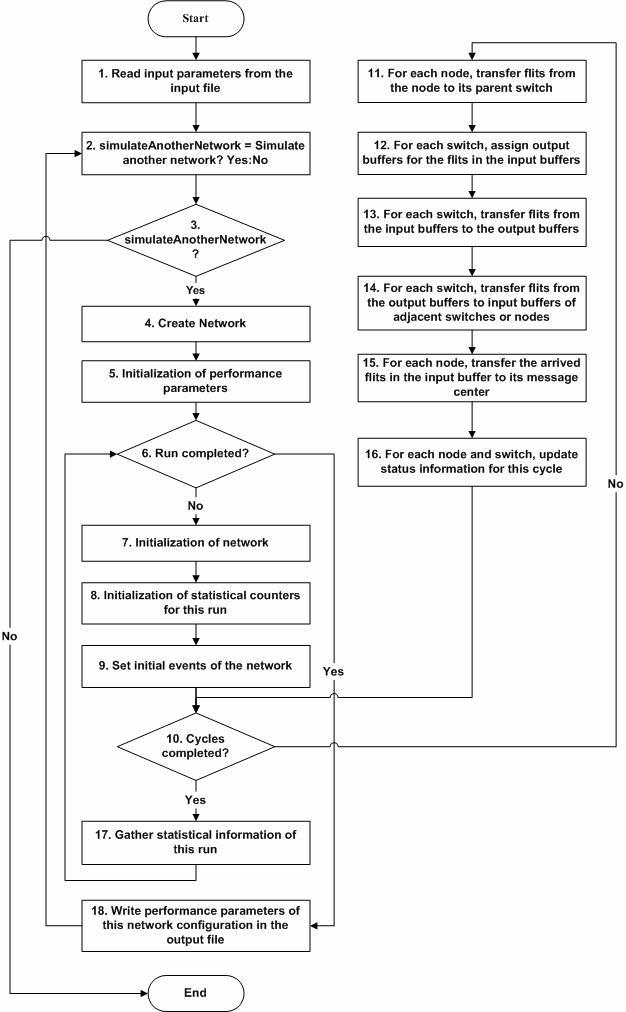


Рисунок 5. Алгоритм работы симулятора.

Опишем теперь собственно программную реализацию модели, то есть из какого набора компонентов (классов) состоит симулятор gpNoCSim++. А именно:

1. Controller – данный класс целиком описывает (и реализует) процесс выполнения симуляции.
2. Network Manager – отвечает за создание и инициализацию новый сетей, также создает объекты классов HelpingUtility и StatisticalData.
3. Network – класс собственно самой сети, который соединяет между собой блоки и коммутаторы.
4. Switch – класс коммутатора.
5. Router – интерфейс, позволяющий переопределить способ маршрутизации сообщений.
6. Node – данный класс проверяет виртуальные каналы на наличие свободных, прикрепляет пакеты к каналам, а также перенаправляет полученные данные закрепленному за ним IP-блоку.
7. NodeTraffic – этот абстрактный класс определяет методы, необходимые для описания конфигурации системы маршрутизации пакетов.
8. LinkControllers – каждый подобный контроллер имеет независимую FIFO-память, которая отвечает за буферизацию входящих пакетов до их отправки на выходные порты устройства. Так, при поступлении на вход нового заголовка пакета, именно эти контроллеры и определяют наиболее подходящий выходной порт.
9. Virtual Channel Buffers – классы InputVCBuffer и OutputVCBuffer определяют структуры входного и выходного буферов, а также способы хранения данных в портах.
10. Flit – класс, описывающий данные, связанные с частью соответствующего пакета: тип, ip-адреса источника и назначения, данные и время создания.
11. StatisticalData – класс, собирающий и вычисляющий различные показатели работы модели.
12. HelpingUtility – утилитарный класс, ответственный за генерацию случайных значений, а также за чтение входного файла с параметрами.

# **Работа с симулятором gpNoCSim++**

Как уже было сказано ранее, симулятор gpNoCSim++ позволяет разработчику выбрать топологию сети на кристалле из следующего списка: fat tree, mesh, torus, extended fat tree, octal, WK-recursive. Рассмотрим подробно процесс симуляции работы СтнК с топологией mesh2D(4,4).

Для начала пользователю предлагает определить конкретные параметры сети (рис. 6): топологию, количество IP-блоков, время между получением одного и генерацией другого сообщения, размер этого сообщения и flit length (на какие части будет делиться пакет). Также он может указать необходимое количество итераций цикла симуляции и собственно число этих циклов, а также возможность запуска симуляции в многопоточном режиме.

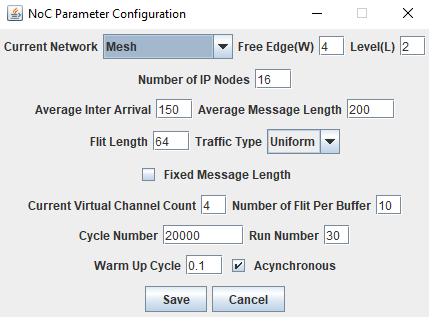


Рисунок 6. Конфигурация симулятора.

После того как все параметры были сохранены (в этот момент симулятор записывает их в уже упомянутый файл *nocSimParameter.txt*), на главном экране появится сообщение «Configuration changes successfully saved» и можно будет начинать моделирование. В процессе моделирования на главном экране появится лог симуляции (рис. 7), описывающий процесс отправки сообщений между IP-блоками.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 7. Лог работы симулятора.

По завершении работы симулятора появится окно Simulation Results (рис. 8), в котором будут приведены различные статистические данные последнего моделирования: пропускная способность сети, среднее время задержки в сети, процент использования входных и выходных буферов, среднее число хопов, а также среднее число отправленных и полученных пакетов в каждом цикле симуляции.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 8. Результаты моделирования.

Однако даже несмотря на большой выбор доступных для моделирования топологий, алгоритмов маршрутизации и прочих предустановленных конфигураций симулятора и сети, иногда возникает необходимость провести симуляцию СтнК с какими специфичными, требующими тонкой настройки параметрами. В этом случае разработчику в зависимости от поставленной задачи необходимо выполнить следующее:

1. Чтобы изменить принцип работы или архитектуру коммутатора, необходимо создать новый класс с именем NewArchSwitch (например, для топологии Mesh это будет класс MeshSwitch), который будет реализовывать интерфейс Switch с переопределением всех соответствующих методов.
2. Для добавления нового алгоритма маршрутизации необходимо создать новый класс, реализующий метод getDestination интерфейса Router.
3. При необходимости добавить новый способ генерации трафика, нужно в новом классе реализовать интерфейс NodeTraffic и отдавать его в качестве параметра в Node.
4. Изменение способа кодирования пакетов заключается в переопределении методов createHeaderFlit и createDataFlit интерфейса NodeTraffic.
5. Переопределяя метод setNextMsgGenTime в том же интерфейсе NodeTraffic, можно изменить способ генерации сообщений.
6. Изменить способ генерации пакетов различной длины можно переопределив метод getMessageSize в интерфейсе NodeTraffic.
7. Для добавления новых параметров быстродействия симулятора требуется изменить класс StatisticalData.

# **Сравнение симуляций различных топологий с помощью gpNocSim++**

Проведем моделирование различных топологий СтнК с одинаковыми остальными параметрами, а именно: average inter arrival – 150, average message length – 200, flit length – 64, количество IP-блоков в сети – 16, количество итераций цикла – 20000, количество симуляций – 30. В качестве характеристики для сравнения выберем отношение полученных пакетов к отправленным. Моделировать будем следующие доступные топологии: fat tree, mesh, torus, extended fat tree, octal, WK-recursive. Результаты моделирования приведены на рисунке 9.



Рисунок 9. Сравнение результатов моделирования различных топологий.

# **Интеграция модели в единую среду моделирования**

Добавим рассмотренную до этого модель gpNocSim++ в единую среду автоматизированного моделирования сетей на кристалле UHLNoCS. Для этого, однако, пришлось изменить исходный код симулятора – как уже было сказано, gpNocSim++ обладает собственным пользовательским интерфейсом, который бы только мешал при работе с ней из-под единой среды. Более того, для корректной работы симулятора в него также была добавлена поддержка аргументов командной строки для указания расположения входного файла конфигурации и выходного файла с результатами моделирования.



Рисунок 10. Структура единой среды моделирования СтнК UHLNoCS.

Таким образом, для работы из-под среды UHLNoCS необходима специальная (измененная) версия симулятора gpNocSim++: лишенная пользовательского интерфейса и поддерживающая аргументы командной строки. Код модифицированных классов представлен в приложении А.

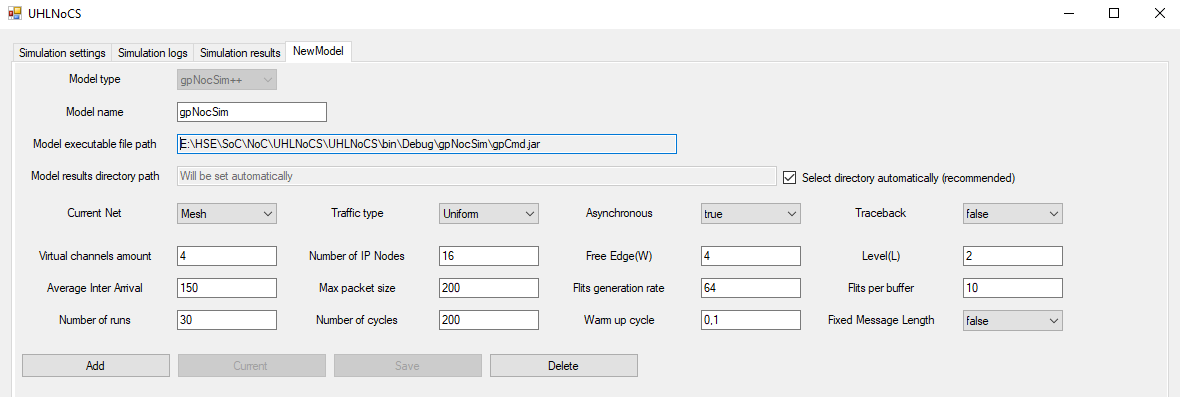
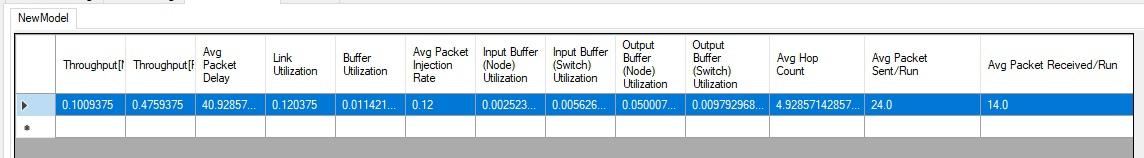


Рисунок 11. Настройка параметров модели в среде UHLNoCS.

Рисунок 12. Результаты моделирования в среде.



# **ЗАКЛЮЧЕНИЕ**

В процессе выполнения производственной практики были изучены принципы топологического подхода при проектировании сетей на кристалле, проведен обзор различных моделей для проектирования СтнК, изучен симулятор gpNocSim++ - проанализирована его структура и принцип работы, а также рассмотрены возможности по его дальнейшей модификации, проведено моделирование различных топологий в данном симуляторе. Итогом работы является успешная интеграция симулятора gpNocSim++ в единую среду высокоуровневого моделирования СтнК UHLNoCS.

Во время выполнения были приобретены знания о сетях на кристалле, различных моделях – высоко- и низкоуровневых – данных сетей, топологическом подходе их моделирования. Также были получены навыки работы с высокоуровневыми симуляторами СтнК (в частности с симулятором gpNocSim++) и единой средой UHLNoCS.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Недбайло Ю. А. Разработка сети на кристалле для перспективных многоядерных микропроцессоров //Труды Московского физико-технического института. – 2017. – Т. 9. – №. 2 (34).
2. H. Hossain, M. Ahmed, A. Al-Nayeem, T. Z. Islam, M. M. Akbar Gpnocsim - A General Purpose Simulator for Network-On-Chip// 2007 International Conference on Information and Communication Technology. – 2007. – P. 254-257.
3. Jamali M. A. J. et al. A study on WK-recursive topology using gpNoCsim++ simulator and comparison to Other topologies //2009 17th IFIP International Conference on Very Large Scale Integration (VLSI-SoC). – IEEE, 2009. – С. 181-184.
4. Romanov O.Y. Optimization of topologies of networks on chip// Herald of the National Technical University "KhPI". Subject issue: Information Science and Modelling. – Kharkov: NTU "KhPI". – 2011. – № 36. – P. 149 – 155.
5. Романов А. Ю. Анализ подходов к синтезу сетей на кристалле с использованием регулярных топологий //Вестник Южно-Уральского государственного университета. Серия: Компьютерные технологии, управление, радиоэлектроника. – 2015. – Т. 15. – №. 1.

# **ПРИЛОЖЕНИЕ А**

**Листинги**

Листинг А1 – GpNocSimApplication.java

import gpApp.gui.GpNoCSimRunable;  
  
public class GpNocSimApplication {  
 public static void main(String[] args) {  
 Thread runnable = new Thread(new GpNoCSimRunable(args[0], args[1]));  
 runnable.start();  
 }  
}

Листинг А2 – HelpingUtility.java

package gpApp.network.common;  
  
import gpApp.gui.GpNoCSimRunable;  
  
import java.io.\*;  
import java.nio.charset.StandardCharsets;  
import java.util.Random;  
import java.util.StringTokenizer;  
import java.util.Vector;  
  
/\*\*  
 \* network.common.HelpingUtility class defines the methods for generating random values with  
 \* uniform distribution using {@link Random}.  
 \* <p/>  
 \* network.common.HelpingUtility class also reads the input parameters and their associated  
 \* values from the input configuration file.  
 \* <p/>  
 \* Conversion of the clock speed to appropriate speed by a factor is also  
 \* performed in this class.  
 \*/  
public class HelpingUtility {  
 /\*\*  
 \* a member of the java.Util.Random class  
 \*/  
 private Random rand = null;  
  
 /\*\*  
 \* list of input parameters  
 \*/  
 private Vector allParamSet = null;  
  
 /\*\*  
 \* Constructor method. Performs initialization of the class variables.  
 \*/  
 public HelpingUtility() {  
 rand = new Random(12345);  
 allParamSet = new Vector();  
 }  
  
 /\*\*  
 \* Sets the seed for the randomizer  
 \*  
 \* @param seed seed value  
 \*/  
 public void setRandSeed(int seed) {  
 rand = new Random(seed);  
 }  
  
 /\*\*  
 \* Instantiates the rand variable.  
 \*/  
 public void setRandomSeed() {  
 rand = new Random();  
 }  
  
 /\*\*  
 \* Returns a random number from 0 to 1 using Uniform Distribution.  
 \*  
 \* @return random value from 0 to 1  
 \*/  
 public double getNextRandomNumber() {  
 return rand.nextDouble();  
 }  
  
 /\*\*  
 \* Returns a set of input parameters for a particular network speicied by  
 \* the arguement of the method.  
 \*  
 \* @param index Index of the network.Network to be tested  
 \* @return a set of input parameters  
 \*/  
 public Vector getParamSet(int index) {  
 if (index < allParamSet.size())  
 return (Vector) allParamSet.get(index);  
  
 return null;  
 }  
  
 /\*\*  
 \* A simple method to get file path from class path.  
 \* @param fileName  
 \* @return file path from class path  
 \*/  
 public String getFilePath(String fileName){  
 return getClass().getClassLoader().getResource("").getPath() + "/text/" + fileName;  
 }  
  
 /\*\*  
 \* Reads the input configuration file. Each empty line defines a new set of  
 \* input parameters. Typically a set of input parameters correspond to the  
 \* simulation for a particular topology or configuration.  
 \* <p/>  
 \* A typical line of the file is of the format <parameter name>=<parameter  
 \* value>. The parameter name and its value is stored in the  
 \* {@link ParamDTO} object.  
 \*  
 \* @param parameterFile the name of the input configuration file  
 \* @see BufferedReader  
 \* @see StringTokenizer  
 \*/  
  
 public void readParameterFromFile(String parameterFile) {  
  
 StringTokenizer theTokenizer;  
 String parameter, value;  
 Vector paramSet = new Vector();  
  
 try(InputStream inputStream = new FileInputStream( parameterFile);  
 InputStreamReader streamReader = new InputStreamReader(inputStream, StandardCharsets.UTF\_8);  
 BufferedReader paramReader = new BufferedReader(streamReader)) {  
  
 String paramLine = paramReader.readLine();  
  
 while (paramLine != null) {  
 if (paramLine.equalsIgnoreCase("")) {  
 allParamSet.add(paramSet);  
 paramSet = new Vector();  
 System.out.println("Added: ");  
  
 } else {  
  
 theTokenizer = new StringTokenizer(paramLine, " =;,:",  
 false);  
 if (theTokenizer.countTokens() >= 2) {  
 parameter = theTokenizer.nextToken();  
 value = theTokenizer.nextToken();  
 paramSet.add(new ParamDTO(parameter, value));  
 }  
 }  
 paramLine = paramReader.readLine();  
 System.out.println("ReadLine: " + paramLine);  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 /\*\*  
 \* Returns a converted cycle for the factor specified in the arguement.  
 \*  
 \* @param cycle cycle value of an entity, for example a resource  
 \* @param factor ratio of the speed between an entity, for example a resource  
 \* and the switch  
 \* @return value the cycle with respect to the switch  
 \*/  
  
 public int getConvertedCycle(int cycle, double factor) {  
 return (int) Math.floor((double) (cycle) \* factor);  
 }  
  
}

Листинг А3 – FileExtractor.java

package gpApp.tool;  
  
import gpApp.network.common.HelpingUtility;  
  
import java.io.\*;  
import java.util.HashMap;  
  
/\*\*  
 \* User: Amir Azimi  
 \* Date: Mar 8, 2009  
 \* Time: 10:39:41 PM  
 \*/  
public class FileExtractor {  
 public void resultFileReader() {  
 try {  
 File resultFile = new File(new HelpingUtility().getFilePath("nocSimOutput.txt"));  
 DataInput input = new DataInputStream(new FileInputStream(resultFile));  
 String line = input.readLine();  
 HashMap resultArray[][] = new HashMap[8][6];  
 while (line != null) {  
 if (line.contains("Input Configuration: ......")) {  
 HashMap trafficInfo = new HashMap();  
 line = input.readLine();  
 while (!line.contains("Total")) {  
 if (line.contains(" network.Network")) {  
 line = line.replace(" network.Network", " network.Network\t");  
 } else if (line.contains(" Throughput[Net]")) {  
 line = line.replace(" Throughput[Net]", " Throughput[Net]\t");  
 } else if (line.contains(" Throughput[Flits leaving network.unit.switches.Switch]")) {  
 line = line.replace(" Throughput[Flits leaving network.unit.switches.Switch]",  
 " Throughput[Flits leaving network.unit.switches.Switch]\t");  
 } else  
 if (line.contains(" Buffer(network.unit.node.Node & network.unit.switches.Switch) Utilization")) {  
 line = line.replace(" Buffer(network.unit.node.Node & network.unit.switches.Switch) Utilization",  
 " Buffer(network.unit.node.Node & network.unit.switches.Switch) Utilization\t");  
 } else if (line.contains(" Avg Packet Delay")) {  
 line = line.replace(" Avg Packet Delay", " Avg Packet Delay\t");  
 } else if (line.contains(" Avg Hop Count")) {  
 line = line.replace(" Avg Hop Count", " Avg Hop Count\t");  
 } else if (line.contains(" Avg Packet Sent/Run")) {  
 line = line.replace(" Avg Packet Sent/Run", " Avg Packet Sent/Run\t");  
 }  
 String str[] = line.split(" \*\t+ \*");  
 int networkType = 0;  
 if (str.length == 2) {  
 if (str[0].equalsIgnoreCase(" network.Network")) {  
 trafficInfo.put("Network", str[1]);  
 } else if (str[0].equalsIgnoreCase(" Avg Msg Production Rate(cycle/msg)")) {  
 trafficInfo.put("Avg Msg Production Rate", str[1]);  
 } else if (str[0].equalsIgnoreCase(" Throughput[Net]")) {  
 trafficInfo.put("Throughput[Net]", str[1]);  
 } else  
 if (str[0].equalsIgnoreCase(" Throughput[Flits leaving network.unit.switches.Switch]")) {  
 trafficInfo.put("Throughput[Flits leaving network.unit.switches.Switch]", str[1]);  
 } else if (str[0].equalsIgnoreCase(" Avg Packet Delay")) {  
 trafficInfo.put("Avg Packet Delay", str[1]);  
 } else if (str[0].equalsIgnoreCase(" Link Utilization")) {  
 trafficInfo.put("Link Utilization", str[1]);  
 } else  
 if (str[0].equalsIgnoreCase(" Buffer(network.unit.node.Node & network.unit.switches.Switch) Utilization")) {  
 trafficInfo.put("Buffer(network.unit.node.Node & network.unit.switches.Switch) Utilization", str[1]);  
 } else if (str[0].equalsIgnoreCase(" Avg Packet Injection Rate")) {  
 trafficInfo.put("Avg Packet Injection Rate", str[1]);  
 } else if (str[0].equalsIgnoreCase(" Avg Packet Not Produced")) {  
 trafficInfo.put("Avg Packet Not Produced", str[1]);  
 } else if (str[0].equalsIgnoreCase(" Input Buffer(network.unit.node.Node) Utilization")) {  
 trafficInfo.put("Input Buffer(network.unit.node.Node) Utilization", str[1]);  
 } else  
 if (str[0].equalsIgnoreCase(" Input Buffer(network.unit.switches.Switch) Utilization")) {  
 trafficInfo.put("Input Buffer(network.unit.switches.Switch) Utilization", str[1]);  
 } else if (str[0].equalsIgnoreCase(" Output Buffer(network.unit.node.Node) Utilization")) {  
 trafficInfo.put("Output Buffer(network.unit.node.Node) Utilization", str[1]);  
 } else  
 if (str[0].equalsIgnoreCase(" Output Buffer(network.unit.switches.Switch) Utilization")) {  
 trafficInfo.put("Output Buffer(network.unit.switches.Switch) Utilization", str[1]);  
 } else if (str[0].equalsIgnoreCase(" Avg Hop Count")) {  
 trafficInfo.put("Avg Hop Count", str[1]);  
 } else if (str[0].equalsIgnoreCase(" Avg Packet Sent/Run")) {  
 trafficInfo.put("Avg Packet Sent/Run", str[1]);  
 } else if (str[0].equalsIgnoreCase(" Avg Packet Received/Run")) {  
 trafficInfo.put("Avg Packet Received/Run", str[1]);  
 }  
 }  
 line = input.readLine();  
 }  
 resultArray[Integer.valueOf(trafficInfo.get("Avg Msg Production Rate").toString()).intValue() / 25 - 1]  
 [Integer.valueOf(trafficInfo.get("Network").toString()).intValue()] = trafficInfo;  
 } else {  
 line = input.readLine();  
 }  
 }  
 createResultHTMLBasedOnTrafficRate(resultArray);  
 createResultHTMLBasedOnTopology(resultArray);  
 createResultHTMLBasedOnNetParameters(resultArray);  
  
 } catch (FileNotFoundException e) {  
 e.printStackTrace(); //To change body of catch statement use File | Settings | File Templates.  
 } catch (IOException e) {  
 e.printStackTrace(); //To change body of catch statement use File | Settings | File Templates.  
 }  
 }  
  
 private void createResultHTMLBasedOnNetParameters(HashMap[][] resultArray) {  
 try {  
 HelpingUtility helpingUtility = new HelpingUtility();  
 String dirPath = helpingUtility.getFilePath("results");  
 File dirFile = new File(dirPath);  
 if(!dirFile.isDirectory()){  
 dirFile.mkdir();  
 }  
  
 File htmlFile = new File(helpingUtility.getFilePath("results/extracted-results-parameters.html"));  
 if (!htmlFile.exists()) {  
 htmlFile.createNewFile();  
 }  
 DataOutput output = new DataOutputStream(new FileOutputStream(htmlFile));  
 output.write(("<html><head><style>\n" +  
 "body{\n" +  
 "font-size:14;\n" +  
 "font-family:tahoma;\n" +  
 "}\n" +  
 "table{\n" +  
 "border:1 solid black;\n" +  
 "\n" +  
 "}\n" +  
 "th{\n" +  
 "background-color:#0099CC;\n" +  
 "font-size:16px;\n" +  
 "}\n" +  
 "</style>" +  
 "<Title>Extracted Results</title>" +  
 "</head><body>\n").getBytes());  
  
 /\*\*  
 \* Throughput Table For All Topologies  
 \*/  
 output.write(("<br><h1>Net Throughput</h1><br>").getBytes());  
 output.write(("<table border=\"1\"><tr >" +  
 "<th>Avg Msg Production Rate</th>" +  
 "<th>Fat Tree</th>" +  
 "<th>Mesh</th>" +  
 "<th>Torus</th>" +  
 "<th>Extended Fat Tree</th>" +  
 "<th>Octal</th>" +  
 "<th>WK-Recursive</th>" +  
 "</tr>\n<tr>").getBytes());  
 for (int i = 0; i < 8; i++) {  
  
 output.write(("<td>" + resultArray[i][0].get("Avg Msg Production Rate") + "</td>").getBytes());  
 for (int j = 0; j < 6; j++) {  
 output.write((  
 "<td>" + resultArray[i][j].get("Throughput[Net]") + "</td>").getBytes());  
 }  
 output.write("</tr>".getBytes());  
 }  
 output.write("</table><br>".getBytes());  
  
 /\*\*  
 \* Switch Throughput Table For All Topologies  
 \*/  
 output.write(("<br><h1>Switch Throughput</h1><br>").getBytes());  
 output.write(("<table border=\"1\"><tr >" +  
 "<th>Avg Msg Production Rate</th>" +  
 "<th>Fat Tree</th>" +  
 "<th>Mesh</th>" +  
 "<th>Torus</th>" +  
 "<th>Extended Fat Tree</th>" +  
 "<th>Octal</th>" +  
 "<th>WK-Recursive</th>" +  
 "</tr>\n<tr>").getBytes());  
 for (int i = 0; i < 8; i++) {  
  
 output.write(("<td>" + resultArray[i][0].get("Avg Msg Production Rate") + "</td>").getBytes());  
 for (int j = 0; j < 6; j++) {  
 output.write((  
 "<td>" + resultArray[i][j].get("Throughput[Flits leaving network.unit.switches.Switch]") + "</td>").getBytes());  
 }  
 output.write("</tr>".getBytes());  
 }  
 output.write("</table><br>".getBytes());  
  
 /\*\*  
 \* Average Packet Delay Table For All Topologies  
 \*/  
 output.write(("<br><h1>Average Packet Delay</h1><br>").getBytes());  
 output.write(("<table border=\"1\"><tr >" +  
 "<th>Avg Msg Production Rate</th>" +  
 "<th>Fat Tree</th>" +  
 "<th>Mesh</th>" +  
 "<th>Torus</th>" +  
 "<th>Extended Fat Tree</th>" +  
 "<th>Octal</th>" +  
 "<th>WK-Recursive</th>" +  
 "</tr>\n<tr>").getBytes());  
 for (int i = 0; i < 8; i++) {  
  
 output.write(("<td>" + resultArray[i][0].get("Avg Msg Production Rate") + "</td>").getBytes());  
 for (int j = 0; j < 6; j++) {  
 output.write((  
 "<td>" + resultArray[i][j].get("Avg Packet Delay") + "</td>").getBytes());  
 }  
 output.write("</tr>".getBytes());  
 }  
 output.write("</table><br>".getBytes());  
  
 /\*\*  
 \* Link Utilization Table For All Topologies  
 \*/  
 output.write(("<br><h1>Link Utilization</h1><br>").getBytes());  
 output.write(("<table border=\"1\"><tr >" +  
 "<th>Avg Msg Production Rate</th>" +  
 "<th>Fat Tree</th>" +  
 "<th>Mesh</th>" +  
 "<th>Torus</th>" +  
 "<th>Extended Fat Tree</th>" +  
 "<th>Octal</th>" +  
 "<th>WK-Recursive</th>" +  
 "</tr>\n<tr>").getBytes());  
 for (int i = 0; i < 8; i++) {  
  
 output.write(("<td>" + resultArray[i][0].get("Avg Msg Production Rate") + "</td>").getBytes());  
 for (int j = 0; j < 6; j++) {  
 output.write((  
 "<td>" + resultArray[i][j].get("Link Utilization") + "</td>").getBytes());  
 }  
 output.write("</tr>".getBytes());  
 }  
 output.write("</table><br>".getBytes());  
  
 /\*\*  
 \* Utilization Utilization Table For All Topologies  
 \*/  
 output.write(("<br><h1>Buffer Utilization</h1><br>").getBytes());  
 output.write(("<table border=\"1\"><tr >" +  
 "<th>Avg Msg Production Rate</th>" +  
 "<th>Fat Tree</th>" +  
 "<th>Mesh</th>" +  
 "<th>Torus</th>" +  
 "<th>Extended Fat Tree</th>" +  
 "<th>Octal</th>" +  
 "<th>WK-Recursive</th>" +  
 "</tr>\n<tr>").getBytes());  
 for (int i = 0; i < 8; i++) {  
  
 output.write(("<td>" + resultArray[i][0].get("Avg Msg Production Rate") + "</td>").getBytes());  
 for (int j = 0; j < 6; j++) {  
 output.write((  
 "<td>" + resultArray[i][j].get("Buffer(network.unit.node.Node & network.unit.switches.Switch) Utilization") + "</td>").getBytes());  
 }  
 output.write("</tr>".getBytes());  
 }  
 output.write("</table><br>".getBytes());  
  
 /\*\*  
 \* Avg Packet Injection Rate Table For All Topologies  
 \*/  
 output.write(("<br><h1>Avg Packet Injection Rate</h1><br>").getBytes());  
 output.write(("<table border=\"1\"><tr >" +  
 "<th>Avg Msg Production Rate</th>" +  
 "<th>Fat Tree</th>" +  
 "<th>Mesh</th>" +  
 "<th>Torus</th>" +  
 "<th>Extended Fat Tree</th>" +  
 "<th>Octal</th>" +  
 "<th>WK-Recursive</th>" +  
 "</tr>\n<tr>").getBytes());  
 for (int i = 0; i < 8; i++) {  
  
 output.write(("<td>" + resultArray[i][0].get("Avg Msg Production Rate") + "</td>").getBytes());  
 for (int j = 0; j < 6; j++) {  
 output.write((  
 "<td>" + resultArray[i][j].get("Avg Packet Injection Rate") + "</td>").getBytes());  
 }  
 output.write("</tr>".getBytes());  
 }  
 output.write("</table><br>".getBytes());  
  
 /\*\*  
 \* Avg Packet Not Produced Table For All Topologies  
 \*/  
 output.write(("<br><h1>Avg Packet Not Produced</h1><br>").getBytes());  
 output.write(("<table border=\"1\"><tr >" +  
 "<th>Avg Msg Production Rate</th>" +  
 "<th>Fat Tree</th>" +  
 "<th>Mesh</th>" +  
 "<th>Torus</th>" +  
 "<th>Extended Fat Tree</th>" +  
 "<th>Octal</th>" +  
 "<th>WK-Recursive</th>" +  
 "</tr>\n<tr>").getBytes());  
 for (int i = 0; i < 8; i++) {  
  
 output.write(("<td>" + resultArray[i][0].get("Avg Msg Production Rate") + "</td>").getBytes());  
 for (int j = 0; j < 6; j++) {  
 output.write((  
 "<td>" + resultArray[i][j].get("Avg Packet Not Produced") + "</td>").getBytes());  
 }  
 output.write("</tr>".getBytes());  
 }  
 output.write("</table><br>".getBytes());  
  
 /\*\*  
 \* Avg Hop Count Table For All Topologies  
 \*/  
 output.write(("<br><h1>Avg Hop Count</h1><br>").getBytes());  
 output.write(("<table border=\"1\"><tr >" +  
 "<th>Avg Msg Production Rate</th>" +  
 "<th>Fat Tree</th>" +  
 "<th>Mesh</th>" +  
 "<th>Torus</th>" +  
 "<th>Extended Fat Tree</th>" +  
 "<th>Octal</th>" +  
 "<th>WK-Recursive</th>" +  
 "</tr>\n<tr>").getBytes());  
 for (int i = 0; i < 8; i++) {  
  
 output.write(("<td>" + resultArray[i][0].get("Avg Msg Production Rate") + "</td>").getBytes());  
 for (int j = 0; j < 6; j++) {  
 output.write((  
 "<td>" + resultArray[i][j].get("Avg Hop Count") + "</td>").getBytes());  
 }  
 output.write("</tr>".getBytes());  
 }  
 output.write("</table><br>".getBytes());  
  
  
 output.write("</body>".getBytes());  
 //System.out.print("Success to write html file");  
 } catch (FileNotFoundException e) {  
 e.printStackTrace(); //To change body of catch statement use File | Settings | File Templates.  
 } catch (IOException e) {  
 e.printStackTrace(); //To change body of catch statement use File | Settings | File Templates.  
 }  
  
 }  
  
 private void createResultHTMLBasedOnTopology(HashMap[][] resultArray) {  
 try {  
 HelpingUtility helpingUtility = new HelpingUtility();  
 String dirPath = helpingUtility.getFilePath("results");  
 File dirFile = new File(dirPath);  
 if(!dirFile.isDirectory()){  
 dirFile.mkdir();  
 }  
  
 File htmlFile = new File(helpingUtility.getFilePath("results/extracted-results-topology.html"));  
  
 if (!htmlFile.exists()) {  
 htmlFile.createNewFile();  
 }  
 DataOutput output = new DataOutputStream(new FileOutputStream(htmlFile));  
 output.write(("<html><head><style>\n" +  
 "body{\n" +  
 "font-size:14;\n" +  
 "font-family:tahoma;\n" +  
 "}\n" +  
 "table{\n" +  
 "border:1 solid black;\n" +  
 "\n" +  
 "}\n" +  
 "th{\n" +  
 "background-color:#0099CC;\n" +  
 "font-size:16px;\n" +  
 "}\n" +  
 "</style>" +  
 "<Title>Extracted Results</title>" +  
 "</head><body>\n").getBytes());  
 for (int i = 0; i < 6; i++) {  
 String netName = "";  
 if (i == 0) {  
 netName = "Fat Tree";  
 } else if (i == 1) {  
 netName = "Mesh";  
 } else if (i == 2) {  
 netName = "Torus";  
 } else if (i == 3) {  
 netName = "Extended Fat Tree";  
 } else if (i == 4) {  
 netName = "Octal";  
 } else if (i == 5) {  
 netName = "WK-Recursive";  
 }  
 output.write(("<br><h1> " + netName + "</h1><br>").getBytes());  
 output.write(("<table border=\"1\"><tr >" +  
 "<th>Avg Msg Production Rate</th>" +  
 "<th>Net Throughput</th>" +  
 "<th>Switch Throughput</th>" +  
 "<th>Avg Packet Delay</th>" +  
 "<th>Link Utilization</th>" +  
 "<th>Buffer Utilization</th>" +  
 "<th>Avg Packet Injection Rate</th>" +  
 "<th>Avg Packet Not Produced</th>" +  
 "<th>Avg Hop Count" + "</th></tr>\n").getBytes());  
 for (int j = 0; j < 8; j++) {  
  
 output.write(("<tr>" +  
 "<td>" + resultArray[j][i].get("Avg Msg Production Rate") + "</td>" +  
 "<td class=\"trafficType\">" + resultArray[j][i].get("Throughput[Net]") + "</td>" +  
 "<td>" + resultArray[j][i].get("Throughput[Flits leaving network.unit.switches.Switch]") + "</td>" +  
 "<td>" + resultArray[j][i].get("Avg Packet Delay") + "</td>" +  
 "<td>" + resultArray[j][i].get("Link Utilization") + "</td>" +  
 "<td>" + resultArray[j][i].get("Buffer(network.unit.node.Node & network.unit.switches.Switch) Utilization") + "</td>" +  
 "<td>" + resultArray[j][i].get("Avg Packet Injection Rate") + "</td>" +  
 "<td>" + resultArray[j][i].get("Avg Packet Not Produced") + "</td>" +  
 "<td>" + resultArray[j][i].get("Avg Hop Count") + "</td>" +  
 "</tr>").getBytes());  
 }  
 output.write("</table><br>".getBytes());  
 }  
 output.write("</body>".getBytes());  
 //System.out.print("Success to write html file");  
 } catch (FileNotFoundException e) {  
 e.printStackTrace(); //To change body of catch statement use File | Settings | File Templates.  
 } catch (IOException e) {  
 e.printStackTrace(); //To change body of catch statement use File | Settings | File Templates.  
 }  
  
 }  
  
 private void createResultHTMLBasedOnTrafficRate(HashMap[][] resultArray) {  
 try {  
 HelpingUtility helpingUtility = new HelpingUtility();  
 String dirPath = helpingUtility.getFilePath("results");  
 File dirFile = new File(dirPath);  
 if(!dirFile.isDirectory()){  
 dirFile.mkdir();  
 }  
 File htmlFile = new File(helpingUtility.getFilePath("results/extracted-results-traffic-rate.html"));  
 if (!htmlFile.exists()) {  
 htmlFile.createNewFile();  
 }  
 DataOutput output = new DataOutputStream(new FileOutputStream(htmlFile));  
 output.write(("<html><head><style>\n" +  
 "body{\n" +  
 "font-size:14;\n" +  
 "font-family:tahoma;\n" +  
 "}\n" +  
 "table{\n" +  
 "border:1 solid black;\n" +  
 "\n" +  
 "}\n" +  
 "th{\n" +  
 "background-color:#0099CC;\n" +  
 "font-size:16px;\n" +  
 "}\n" +  
 "</style>" +  
 "<Title>Extracted Results</title>" +  
 "</head><body>\n").getBytes());  
 for (int i = 0; i < 8; i++) {  
 output.write(("<br><h1>traffic rate " + resultArray[i][0].get("Avg Msg Production Rate") + "</h1><br>").getBytes());  
 output.write(("<table border=\"1\"><tr >" +  
 "<th>Topology</th>" +  
 "<th>Net Throughput</th>" +  
 "<th>Switch Throughput</th>" +  
 "<th>Avg Packet Delay</th>" +  
 "<th>Link Utilization</th>" +  
 "<th>Buffer Utilization</th>" +  
 "<th>Avg Packet Injection Rate</th>" +  
 "<th>Avg Packet Not Produced</th>" +  
 "<th>Avg Hop Count" + "</th></tr>\n").getBytes());  
 for (int j = 0; j < 6; j++) {  
 String netName = "";  
 if (j == 0) {  
 netName = "Fat Tree";  
 } else if (j == 1) {  
 netName = "Mesh";  
 } else if (j == 2) {  
 netName = "Torus";  
 } else if (j == 3) {  
 netName = "Extended Fat Tree";  
 } else if (j == 4) {  
 netName = "Octal";  
 } else if (j == 5) {  
 netName = "WK-Recursive";  
 }  
  
 output.write(("<tr>" +  
 "<td>" + netName + "</td>" +  
 "<td class=\"trafficType\">" + resultArray[i][j].get("Throughput[Net]") + "</td>" +  
 "<td>" + resultArray[i][j].get("Throughput[Flits leaving network.unit.switches.Switch]") + "</td>" +  
 "<td>" + resultArray[i][j].get("Avg Packet Delay") + "</td>" +  
 "<td>" + resultArray[i][j].get("Link Utilization") + "</td>" +  
 "<td>" + resultArray[i][j].get("Buffer(network.unit.node.Node & network.unit.switches.Switch) Utilization") + "</td>" +  
 "<td>" + resultArray[i][j].get("Avg Packet Injection Rate") + "</td>" +  
 "<td>" + resultArray[i][j].get("Avg Packet Not Produced") + "</td>" +  
 "<td>" + resultArray[i][j].get("Avg Hop Count") + "</td>" +  
 "</tr>").getBytes());  
 }  
 output.write("</table><br>".getBytes());  
 }  
 output.write("</body>".getBytes());  
 //System.out.print("Success to write html file");  
 } catch (FileNotFoundException e) {  
 e.printStackTrace(); //To change body of catch statement use File | Settings | File Templates.  
 } catch (IOException e) {  
 e.printStackTrace(); //To change body of catch statement use File | Settings | File Templates.  
 }  
 }  
  
 public static void main(String[] args) {  
 FileExtractor f = new FileExtractor();  
 f.resultFileReader();  
 }  
}

Листинг А4 – gpNocSim.cs

using System;

using System.Globalization;

using System.IO;

namespace UHLNoCS.Models

{

public class GpNocSim : Model

{

private const string CONFIG\_FILE\_NAME = "nocSimParameter.txt";

private const string RESULT\_FILE\_NAME = "nocSimOutput.txt";

private string configFilePath;

public static string[] nets = new string[] { "Fat Free", "Mesh", "Torus", "Extended Fat Free", "Octal", "W Recursive" };

public static string[] trafficTypes = new string[] { "Uniform", "Local" };

public static int defaultNet = 1;

public static int defaultTrafficType = 0;

public static int defaultVirtualChannelCount = 4;

public static int defaultNumberOfIpNodes = 16;

public static int defaultWKW = 4;

public static int defaultWKL = 2;

public static int defaultAvgInterArrival = 150;

public static int defaultAvgMessageLength = 200;

public static bool defaultFixedMessageLength = false;

public static int defaultFlitLength = 64;

public static int defaultFlitPerBuffer = 10;

public static int defaultNumOfRuns = 30;

public static int defaultNumOfCycles = 200;

public static float defaultWarmUpCycle = 0.1f;

public static bool defaultTrace = false;

public static bool defaultAsync = true;

public static bool defaultDebug = false;

private int net;

private int trafficType;

private int numberOfIpNodes;

private int wkw;

private int wkl;

private int avgInterArrival;

private int avgMessageLength;

private bool fixedMessageLength;

private int flitLength;

private int flitPerBuffer;

private int virtualChannelCount;

private int numOfRuns;

private int numOfCycles;

private float warmUpCycle;

private bool trace;

private bool async;

private bool debug;

public GpNocSim(string NewType, string NewName, string NewExecutableFilePath, string NewResultsDirectoryPath, string NewConfigFilePath,int net, int trafficType, int numberOfIpNodes, int wkw, int wkl, int avgInterArrival, int avgMessageLength, bool fixedMessageLength, int flitLength, int flitPerBuffer, virtualChannelCount, int numOfRuns, int numOfCycles, float warmUpCycle, bool trace, bool async, bool debug) : base(NewType, NewName, NewExecutableFilePath, NewResultsDirectoryPath + RESULT\_FILE\_NAME)

{

Type = NewType;

Name = NewName;

ExecutableFilePath = NewExecutableFilePath;

ResultsDirectoryPath = NewResultsDirectoryPath + RESULT\_FILE\_NAME;

this.ConfigFilePath = NewConfigFilePath;

this.Net = net;

this.TrafficType = trafficType;

this.NumberOfIpNodes = numberOfIpNodes;

this.Wkw = wkw;

this.Wkl = wkl;

this.AvgInterArrival = avgInterArrival;

this.AvgMessageLength = avgMessageLength;

this.FixedMessageLength = fixedMessageLength;

this.FlitLength = flitLength;

this.FlitPerBuffer = flitPerBuffer;

this.VirtualChannelCount = virtualChannelCount;

this.NumOfRuns = numOfRuns;

this.NumOfCycles = numOfCycles;

this.WarmUpCycle = warmUpCycle;

this.Trace = trace;

this.Async = async;

this.Debug = debug;

}

public int Net { get => net; set => net = value; }

public int TrafficType { get => trafficType; set => trafficType = value; }

public int NumberOfIpNodes { get => numberOfIpNodes; set => numberOfIpNodes = value; }

public int Wkw { get => wkw; set => wkw = value; }

public int Wkl { get => wkl; set => wkl = value; }

public int AvgInterArrival { get => avgInterArrival; set => avgInterArrival = value; }

public int AvgMessageLength { get => avgMessageLength; set => avgMessageLength = value; }

public bool FixedMessageLength { get => fixedMessageLength; set => fixedMessageLength = value; }

public int FlitLength { get => flitLength; set => flitLength = value; }

public int FlitPerBuffer { get => flitPerBuffer; set => flitPerBuffer = value; }

public int VirtualChannelCount { get => virtualChannelCount; set => virtualChannelCount = value; }

public int NumOfRuns { get => numOfRuns; set => numOfRuns = value; }

public int NumOfCycles { get => numOfCycles; set => numOfCycles = value; }

public float WarmUpCycle { get => warmUpCycle; set => warmUpCycle = value; }

public bool Trace { get => trace; set => trace = value; }

public bool Async { get => async; set => async = value; }

public bool Debug { get => debug; set => debug = value; }

public string ConfigFilePath { get => configFilePath; set => configFilePath = value; }

public void PrepareForSimulation()

{

GenerateConfigFile();

}

public void GenerateConfigFile()

{

string ConfigInfo = "";

ConfigInfo += "CURRENT\_NET = " + net + "\r\n";

ConfigInfo += "WK\_W = " + wkw + "\r\n";

ConfigInfo += "WK\_L = " + wkl + "\r\n";

ConfigInfo += "AVG\_INTER\_ARRIVAL = " + avgInterArrival + "\r\n";

ConfigInfo += "AVG\_MESSAGE\_LENGTH = " + avgMessageLength + "\r\n";

ConfigInfo += "FLIT\_LENGTH = " + flitLength + "\r\n";

ConfigInfo += "NUMBER\_OF\_IP\_NODE = " + numberOfIpNodes + "\r\n";

ConfigInfo += "CURRENT\_VC\_COUNT = " + virtualChannelCount + "\r\n";

ConfigInfo += "NUM\_FLIT\_PER\_BUFFER = " + flitPerBuffer + "\r\n";

ConfigInfo += "NUM\_CYCLE = " + numOfCycles + "\r\n";

ConfigInfo += "NUM\_RUN = " + numOfRuns + "\r\n";

ConfigInfo += "TRAFFIC\_TYPE = " + trafficType + "\r\n";

ConfigInfo += "WARM\_UP\_CYCLE = " + warmUpCycle.ToString(new CultureInfo("en-US")) + "\r\n";

ConfigInfo += "FIXED\_MESSAGE\_LENGTH = " + fixedMessageLength.ToString().ToLower() + "\r\n";

ConfigInfo += "TRACE = " + trace.ToString().ToLower() + "\r\n";

ConfigInfo += "ASYNCHRONOUS = " + async.ToString().ToLower() + "\r\n";

ConfigInfo += "DEBUG = " + debug.ToString().ToLower() + "\r\n" + "\r\n";

File.WriteAllText(configFilePath, ConfigInfo);

}

public string[,] CollectSimulationResults()

{

string ResultsSectionHeader = "Performance Measurements.......";

int ThroughputNetLine =1;

int ThroughputNetIndex = 0;

int ThroughputFlitsLine = 2;

int ThroughputFlitsIndex = 1;

int AvgPacketDelayLine = 3;

int AvgPacketDelayIndex = 2;

int LinkUtilizationLine = 4;

int LinkUtilizationIndex = 3;

int BufferUtilizationLine = 5;

int BufferUtilizationIndex = 4;

int AvgPacketInjrateLine = 6;

int AvgPacketInjrateIndex = 5;

int InputBufferUtilizationNodeLine = 8;

int InputBufferUtilizationNodeIndex = 6;

int InputBufferUtilizationSwitchLine = 9;

int InputBufferUtilizationSwitchIndex = 7;

int OutputBufferUtilizationNodeLine = 10;

int OutputBufferUtilizationNodeIndex = 8;

int OutputBufferUtilizationSwitchLine = 11;

int OutputBufferUtilizationSwitchIndex = 9;

int AvgHopCountLine = 12;

int AvgHopCountIndex = 10;

int AvgPacketSentRunLine = 13;

int AvgPacketSentRunIndex = 11;

int AvgPacketRecievedRunLine = 14;

int AvgPacketRecievedRunIndex = 12;

int CharacteristicsAmount = 13;

int SectionsAmount = 0;

string[] ResultSections = File.ReadAllLines(GetResultFilePath());

foreach (string ResultString in ResultSections)

{

if (ResultString.Contains(ResultsSectionHeader))

{

SectionsAmount++;

}

}

string[,] Result = new string[SectionsAmount, CharacteristicsAmount];

int SectionIndex = -1;

int SectionLineIndex = 0;

foreach (string ResultString in ResultSections)

{

if (ResultString.Contains(ResultsSectionHeader))

{

SectionIndex++;

SectionLineIndex = 0;

}

if (SectionIndex == 0) {

if (SectionLineIndex == ThroughputNetLine)

{

Result[SectionIndex, ThroughputNetIndex] = Common.ExtractValueFrom(ResultString);

}

else if (SectionLineIndex == ThroughputFlitsLine)

{

Result[SectionIndex, ThroughputFlitsIndex] = Common.ExtractValueFrom(ResultString);

}

else if (SectionLineIndex == AvgPacketDelayLine)

{

Result[SectionIndex, AvgPacketDelayIndex] = Common.ExtractValueFrom(ResultString);

}

else if (SectionLineIndex == LinkUtilizationLine)

{

Result[SectionIndex, LinkUtilizationIndex] = Common.ExtractValueFrom(ResultString);

}

else if (SectionLineIndex == BufferUtilizationLine)

{

Result[SectionIndex, BufferUtilizationIndex] = Common.ExtractValueFrom(ResultString);

}

else if (SectionLineIndex == AvgPacketInjrateLine)

{

Result[SectionIndex, AvgPacketInjrateIndex] = Common.ExtractValueFrom(ResultString);

}

else if (SectionLineIndex == InputBufferUtilizationNodeLine)

{

Result[SectionIndex, InputBufferUtilizationNodeIndex] = Common.ExtractValueFrom(ResultString);

}

else if (SectionLineIndex == InputBufferUtilizationSwitchLine)

{

Result[SectionIndex, InputBufferUtilizationSwitchIndex] = Common.ExtractValueFrom(ResultString);

}

else if (SectionLineIndex == OutputBufferUtilizationNodeLine)

{

Result[SectionIndex, OutputBufferUtilizationNodeIndex] = Common.ExtractValueFrom(ResultString);

}

else if (SectionLineIndex == OutputBufferUtilizationSwitchLine)

{

Result[SectionIndex, OutputBufferUtilizationSwitchIndex] = Common.ExtractValueFrom(ResultString);

}

else if (SectionLineIndex == AvgHopCountLine)

{

Result[SectionIndex, AvgHopCountIndex] = Common.ExtractValueFrom(ResultString);

}

else if (SectionLineIndex == AvgPacketSentRunLine)

{

Result[SectionIndex, AvgPacketSentRunIndex] = Common.ExtractValueFrom(ResultString);

}

else if (SectionLineIndex == AvgPacketRecievedRunLine)

{

Result[SectionIndex, AvgPacketRecievedRunIndex] = Common.ExtractValueFrom(ResultString);

}

}

SectionLineIndex++;

}

return Result;

}

public string GetResultFilePath() { return this.ResultsDirectoryPath + "\\" + RESULT\_FILE\_NAME; }

}

}