Distributed Memory Architecture with Hardware Graph Array for Configuring the Network Interconnections

<TRITA-ICT-EX-2012:208>

QIANG GE
qianng@kth.se

Master thesis
System-On-Chip Design
Royal Institute of Technology (KTH)
Abstract

The Network-on-chip is considered to be a promising architecture with the advent of increase in the integration of the distributed processing elements. The conflict of data transfer through the network became an urgent issue that need to be solved. DiMArch is an existing distributed memory architecture that can solve the data transfer conflict at compile time. This thesis extends DiMArch with a centralized Network-on-Chip manager called HDiMArch, this architecture monitor the memory accessibility and create data interconnection path with hardware. The conflict which occurred at runtime is resolved by establish the Hardware graph array. The HDiMArch is synthesized using TSMC 90nm with different parameters. Area, power and maximum power results are analyzed.
First of all, I would like to acknowledge the precious chance of this thesis from Professor Ahmed Hemani.

I must thank Mohammad Adeel Tajammul, my supervisor who has guided me all the time during my thesis working. His is always enthusiastic and wisdom. I thank for his patience, his advice and especially his encouragement. From him, I have learned to make plans and solve problems step by step with patience. When I am in trouble, he is always there to help even at his dinner time. Thank him so much!!

I would like to thank my parents who have supported my almost 3 years” life in Sweden. Without them, I would never thought about to study aboard. I appreciate my fiancee, Liu Xuan for her support and encouragement.

I would like to thank my friends and classmates, who stayed with me and gave advices during the thesis time. I acknowledge Ruoxi Zhang and Yunfeng Yang for their help during the project time. May all go well with you

Qiang Ge

Stockholm, May 2012.
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoC</td>
<td>Network-on-chip</td>
</tr>
<tr>
<td>DRRRA</td>
<td>Dynamically Reconfigurable Resource Array</td>
</tr>
<tr>
<td>DiMArch</td>
<td>Distributed Memory Architecture</td>
</tr>
<tr>
<td>HAGAR</td>
<td>Hardware Graph Array</td>
</tr>
<tr>
<td>HDiMArch</td>
<td>Distributed Memory Architecture with HAGAR</td>
</tr>
<tr>
<td>CGRA</td>
<td>Coarse Grain Reconfigurable Architecture</td>
</tr>
<tr>
<td>dUnit</td>
<td>Data Unit</td>
</tr>
<tr>
<td>dSwitch</td>
<td>Data Switch</td>
</tr>
<tr>
<td>DMesh</td>
<td>Data Network</td>
</tr>
<tr>
<td>VLSI</td>
<td>Very-Large-Scale-Integration</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

ABSTRACT ............................................................................................................. 2

FOREWORD ............................................................................................................. 3

NOMENCLATURE .................................................................................................... 4

TABLE OF CONTENTS ........................................................................................... 5

INDEX OF FIGURES AND TABLES .......................................................................... 7

1 INTRODUCTION .................................................................................................... 10

1.1 Background ....................................................................................................... 10

1.2 Purpose ............................................................................................................. 10

1.3 Delimitations .................................................................................................... 11

1.4 Method ............................................................................................................. 12

2 FRAME OF REFERENCE ....................................................................................... 13

2.1 DiMArch .......................................................................................................... 13

2.2 DRRA Architecture .......................................................................................... 15

2.1 Different between DiMArch and HDiMArch ...................................................... 15

3 IMPLEMENTATION ............................................................................................. 17

3.1 HDiMArch system overview ........................................................................... 17
3.2 Data Network-on-chip (DMesh) ...........................................................................18
3.2.1 The data Unit ..................................................................................................18
3.2.2 The data Switch .............................................................................................19
3.2.3 Instruction decoder and interconnection creating rule .................................21
3.2.4 Path conflict flag setting ................................................................................28
3.2.5 Memory manager ............................................................................................28
3.3 NoC manager ........................................................................................................29
3.3.1 Instruction Register .......................................................................................29
3.3.2 Arbitrator .........................................................................................................30
3.4 Memory Bank ........................................................................................................33
4 RESULTS AND DISCUSSION .................................................................................34
4.1 Test instructions ....................................................................................................34
4.1.1 Traffic pattern ..................................................................................................35
4.1.2 Synthesis environment ....................................................................................35
4.2 Test the HDiMArch with different data width VS memory depth ......36
4.2.1 Area ..................................................................................................................36
4.2.2 Power ...............................................................................................................37
4.2.3 Clock frequency ...............................................................................................38
4.3 Test the HDiMArch with different buffers .......................................................39
4.3.1 Area ..................................................................................................................40
INDEX OF FIGURES AND TABLES

Figure 1 Work flow of the thesis ........................................................................................................ 123
Figure 2 DRRA Architecture connect with DiMArch ........................................................................ 15
Figure 3 Overview of HDiMArch ...................................................................................................... 189
Figure 4 The structure of dUnit ......................................................................................................... 20
Figure 5 Framework of dSwitch ....................................................................................................... 21
Figure 6 Data transfer in 3 by 4 mesh ............................................................................................. 21
Figure 7 Data transfer flow .............................................................................................................. 23
Figure 8 ID for each dUnit in DMesh ............................................................................................. 24
Figure 9 The eight cases of transfer data in NoC ............................................................................ 26
Figure 10 A example of data transfer between nodes .................................................................... 29
Figure 11 NoC manager ................................................................................................................... 301
Figure 12 The Architecture of instruction register ......................................................................... 32
Figure 13 The monitor function unit of HAGRA .......................................................................... 33
Figure 14 Test flow .......................................................................................................................... 36
Figure 15 The area chart 1 ............................................................................................................. 379
Figure 16 The power chart 1 .......................................................................................................... 40
Figure 17 The clock chart 1 ............................................................................................................ 41
Figure 18 The area chart 2 ............................................................................................................. 42
Figure 19 The power chart 2 .......................................................................................................... 441
Figure 20 The clock freq chart 2 ..................................................................................................... 44
Figure 21 Freq compare between HDiMArch and DiMArch ....................................................... 45
Figure 22 Result of the scalable comparison .................................................................................. 46
Figure 23 The critical path of DiMArch and HDiMArch in different Mesh size ....................... 47
Figure 24 The area percentage of NoC manager in HDiMArch .................................................... 48

Table 1 Unicast coding rule ............................................................................................................. 22
Table 2 Eight cases of data transfer in DMesh ............................................................................... 25
Table 3 Synthesis setting data ....................................................................................................... 357
Table 4 Area result_1 .................................................................................................................... 368
Table 5 The power result_1 ........................................................................................................... 379
Table 6 Clock freq resul_1 .......................................................................................................... 40
1 INTRODUCTION

This chapter describes the background, the motivation, the limitations and the method(s) used in the presented thesis.

1.1 Background

Network-on-Chip (NoC) is an emerging paradigm for communications within Very- Large-Scale-Integration VLSI\cite{1} systems implemented on a single silicon chip. With the advent increase in the integration of the distributed processing elements, NoC is considered to be a promising scheme.\cite{2} NoC is an approach to designing the communication subsystem between IP cores in a System-on-Chip (SoC). In a NoC system, modules such as processor cores, memories and specialized IP blocks exchange data using a network as a "public transportation" sub-system for the information and data.\cite{3} A NoC is constructed from multiple point-to-point data links interconnected by switches, such that messages can be relayed from any source module to any destination module over several links, by making routing decisions at the switches. As a result, the data transfer confliction at the network becomes a critical issue that impacts the performance of NoC architectures.

1.2 Motivation

Distributed Memory Architecture (DiMArch)\cite{4} can be integrated with a Coarse Grain Reconfigurable Architecture (CGRA). DiMArch enables better interaction between computation fabric and memory fabric. This on-chip interconnect network is designed to provide high bandwidth, Low-latency, scalable and reliable amount of memories which supports the parallel transactions.\cite{5} Since the routing conflict are resolved at the compile time, that may be hard to resolve conflicts that occurred at runtime.
The use of HArdware Graph ARray (HAGAR)\textsuperscript{[6]} is to resolve such problem. Solving graph problem in hardware is an area of research on its own and realizes it in acceptable time. HAGAR configures the interconnection between nodes of NoC at run time. Hence this thesis extends DiMArch with centralized HAGAR\textsuperscript{[7]} enable configuring interconnections at runtime\textsuperscript{[8]}. NoC manager is included in HDiMArch which monitors the memory access, creates data interconnection path, resolves conflicts using HAGAR. Besides, the system also has the ability to change the memory access to computation element ratio at runtime.

In order to gain insight research of HDiMArch, we synthesis the fabric with different parameter such as data width and memory depth, by analyze the results like the required power and maximum limitation some conclusion are made. Also the comparison between HDiMArch and DiMArch is documented.

1.3 Organization of the report

The reminder of this thesis is arranged into four chapters:

- Dynamically Reconfigurable Resource Array (DRRA) and DiMArch are described in chapter-2.
- The implementation of HDiMArch and the path creating rule are described in chapter-3.
- Synthesis results of the HDiMArch along with the comparison between DiMArch and HDiMArch is declared at chapter-4.
- Chapter-5 concludes the thesis.
1.4 Method

The work flow of this thesis can be found at Figure 1, each step in the above mention section corresponding to each report sections. The method is divided into document study, Architecture specs & RTL coding, RTL Simulation & Logic Synthesis and tests on architectures.

- Concept and document studies
- Architecture specs & RTL coding
- RTL Simulation and logic Synthesis
- Tests of architectures
  - Analyze the difference of Power, Area and frequency between HDiMArch with different data width and buffers.
  - Find the difference of Maximum frequency between DiMArch and HDiMArch

*Figure 1 Work flow of the thesis*
The HDiMArch is a DiMArch extension which includes the NoC manager. HDiMArch is developed with the suitable interface for DRRA. As the result, this chapter presents the theoretical reference frame such as the features of DiMArch and architecture of DRRA.

2.1 DiMArch

DiMArch can provide high-bandwidth, low-latency, scalable and reliable communication. this system is designed for DRRA,[9] a kind of Coarse Grain Reconfigurable Architecture (CGRA). the following five features should be mentioned.

- **Distributed**: DRRA being a fabric, the computation is distributed across the chip which runs several applications in parallel, which distributed memory, the proposed design enables multiple private and parallel execution environment.

- **Partitioning**: Due to the feature of distributed, the DiMarch enables the compile time re-partitioning.

- **Streaming**: each partitions includes the memory banks (mBanks) can be considered as node and stream data to computation units. That enable elasticity in streaming by modify the delay value.

- **Energy Optimization and Performance**: the DiMArch provide a optimized path between memory and computation. That lower the latency and save the power supply by manage the unused nodes.

- **Scalability**: DiMArch is scalable with clock frequency and size of the network.

The current DiMArch fabric is composed of memory banks (mBanks), a circuit-switched data Network-on-chip (dNoC) as a connection of the mBanks and RFile of DRRA, a packet-switched instruction Network-on-chip (iNoC) to create partitions streaming data and transport instructions from sequencer of the DRRA to instruction Switch. The fabric structure is shown at Figure 2.
Figure 2  DRRA Architecture connect with DiMArch
2.2 DRRA Architecture

The distributed memory architecture is connected with DRRA architecture. DRRA is a Coarse Grain Reconfigurable Architecture (CGRA) capable of hosting multiple, complete Radio and Multimedia application. The basic single DRRA cell is consisted by mDPU (the morphable data-path Units), the RFile (Register Files), a sequencer and an interconnect schematic gluing these elements together. The architecture is shown at the Figure 2:

- **mDPU**: a simple mDPU is a 16-bit integer units with four 16-bit inputs refer to two complex numbers and two 16-bit outputs refer to one complex number. What is more, the mDPU also has two comparators for the corresponding output and counter, mDPU can process the saturation, truncation/rounding, overflow/underflow check. The result bit-width can be configured.

- **Sequencer**: sequencer controls a serial of other components including one mDPU and a RFile, the corresponding switch box is also included.

- **RFile**: 16 words register file with dual read and write ports. RFile has a DSP style AGU (Address Generation Unit) with circular buffer and bit reverse addressing.

2.3 Difference between DiMArch and HDiMArch

The core difference between DiMArch and HDiMArch is the DiMArch has instruction Network-on-chip (iNoC) is replaced by NoC manager. iNoC is considered a packet-switched network used in DiMArch to program the finite state machine of dNoC and mBanks, the iNoC is primarily used for short programming message and life of a certain path is very short. The agility of programming or reconfiguring DiMArchs partitions and behaviors make the DRRA dynamically reconfigurable at compile time of the DRRA. But partition reconfiguring is not available at runtime.
Instead of iNoC, a NoC manager and corresponding instruction bus is introduced for dynamically configure the memory system only by hardware. The concept of HAGAR is established in NoC manager to enable a runtime conflict solution. This part will be described in detail in the following chapter.
3 IMPLEMENTATION

3.1 HDiMArch system overview

The HDiMArch system is the abbreviation of distributed memory system with HAGAR, which is introduced to solve the runtime data routing conflict in hardware.

Since this thesis investigated the feasibility and performance of distributed memory system with HAGAR, we simplify the former DiMArch without the unnecessary units such as AGU[10] and some related control unit as cfsm, mfsm and zfsm. Furthermore, iNoC from DiMArch is removed since we use NoC manager to establish the private partitioning and interconnection creating.[11]

HDiMArch is composed of

A. A set of distributed memory banks,

B. A circuit-switched[12] data Network-on-chip in the form of two-dimension mesh (DMesh) as a connection between memory banks. By using the DMesh, data is exchanged among memory banks. The node of DMesh is named as dUnit.

C. A NoC manager and signal bus used to create partitions, program DMesh to streaming data and transport decoded control signals.

The overview structures of the HDiMArch is shown at Figure 3
As described before, the DMesh is composed by many pairs of dUnit & memory manager which connected as a mesh. The DMesh provide further needed interfaces connecting to the memory bank and to the corresponding interface at the DRRA.

The DMesh is a duplex circuit-switched mesh Network-on-chip. We use circuit-switched network due to its improved latency for streaming nature of applications. Same as the distributed memory architecture, the Number of interfaces between computational section and memory is in accordance with the number of RFile involved. The number can be modified as it is a pre-defined parameter in the code.
3.2.1 The data Unit (dUnit)

The structure of a general dUnit is described in Figure 4.

The dUnit is composed of (a) data Switch (dSwitch), which is consisted by 5 switch cells connected together. (b) an instruction decoder, that not only can translate the received instruction from NoC-manager and decide the temporal behavior for each dUnit, but also connect to NoC-manager and memory-manager to show the accessibility from input data and to the memory through Flag and memory enable port.

3.2.2 The data Switch (dSwitch)

The dSwitch is made up of five switch-cells called North (N), South(S) West(W), East(E) and Memory(M) that represent they connect with 4 directions and memory-manager (Figure 5). Each switch-cell has 4 inputs from other cells and one output. The multiplexor for 4 inputs is driven by signal SEL_CNB given by instruction-decoder.
For instance in Figure 6 below, Instruction_1 programmed a path from dUnit S1 to dUnit D1 called path_1 as shown, then instruction_2 programmed another path from dUnit S2 to dUnit D2 named path_2, at the conflict dUnit Jam of path_1 and path_2, data at path_2 will not transferred until this dUnit is free of occupation from path_1. It will be buffered in switch-cell W of dUnit Jam, and resend to D2 after conflict released. The multiplexor for buffer is also driven by NoC-manager.

The data collected from IMUX is optionally buffered if pipelined mode on and traffic confliction occurred. The description of pipelined mode and traffic confliction can be found in the NoC manager section (3.4).
Since there are five cells for a dSwitch, basically the dSwitch is configured such as read from one direction or memory and write to one or multiple direction or to the memory. This configuration is designed as shown in table 1. By now only Uni-cast (one read cell and one write cell) is implemented for experiments.

<table>
<thead>
<tr>
<th>From</th>
<th>Uni-cast To</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>North</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>East</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>West</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>South</td>
<td>4</td>
</tr>
<tr>
<td>North</td>
<td>Memory</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>East</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>West</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>South</td>
<td>8</td>
</tr>
<tr>
<td>East</td>
<td>Memory</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>North</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>West</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>South</td>
<td>12</td>
</tr>
<tr>
<td>West</td>
<td>Memory</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>North</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>East</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>South</td>
<td>16</td>
</tr>
<tr>
<td>South</td>
<td>Memory</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>North</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>East</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>West</td>
<td>20</td>
</tr>
</tbody>
</table>

### 3.2.3 instruction decoder and interconnection creating rule

Each dUnit in the dMesh received the same instructions which specified the source dUnit and destination dUnit for a path. However, dUnits of the mesh should be programmed individually and arranged for a certain path. As a result, signal decoder is introduced to each dUnit.

The instruction & signal flow is shown in below figure
In the mesh, every dUnit is named according to which column and row it located. Generic statement of VHDL is used to describe the column and row that make dUnit unique from other ones.
Example one:

As shown at Figure 8, ID 10, 01, the first 2 bits represent it is on row 10 while next 2 bits represent that it is on column 01.

According to the ID and source/destination ID given by instruction, instruction decoder will generate particular command only for its corresponding dSwitch by following the universal path creating rule.

In the design, we assume the source dUnit and destination dUnit on the mesh have different IDs, in principle data routing horizontally has higher priority than routing vertically. As a result, eight cases are concluded based on the relative position between Source / destination dUnit basically (table. 2).
### Table 2 Eight cases of data transfer in DMesh

<table>
<thead>
<tr>
<th>Case</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S_R &lt; D_R and S_C &lt; D_C</td>
</tr>
<tr>
<td>2</td>
<td>S_R &gt; D_R and S_C &lt; D_C</td>
</tr>
<tr>
<td>3</td>
<td>S_R &lt; D_R and S_C &gt; D_C</td>
</tr>
<tr>
<td>4</td>
<td>S_R &gt; D_R and S_C &gt; D_C</td>
</tr>
<tr>
<td>5</td>
<td>S_R = D_R and S_C &lt; D_C</td>
</tr>
<tr>
<td>6</td>
<td>S_R = D_R and S_C &gt; D_C</td>
</tr>
<tr>
<td>7</td>
<td>S_R &lt; D_R and S_C = D_C</td>
</tr>
<tr>
<td>8</td>
<td>S_R &gt; D_R and S_C = D_C</td>
</tr>
</tbody>
</table>

Here we set source dUnit is the dUnit receive data from memory while destination dUnit is the dUnit sending data to memory. 
S_C is the abbreviation of the current column number of source dUnit; 
D_C is the abbreviation of the current column number of destination dUnit; 
S_R is the abbreviation of the current row number of source dUnit, 
S_R is the abbreviation of the current row number of source dUnit’s ID, 
M_C is the abbreviation of the current column number of intermediate dUnit; 
M_R is the abbreviation of the current row number of intermediate dUnit;

Also we classify all the dUnits as the source dUnit (S), destination dUnit (D), intermediate dUnit (I) and unoccupied dUnit (U) according to the usage situation.
For an intermediate dUnit, if data at this dUnit will be send to the right side/ left side direction, for instance sending from north to west or sending from north to east. We define this kind of intermediate dUnit as “turn around”.

Since data routing horizontally has higher priority than routing vertically, and a path with conflict dUnit will be considered invalid. As a result, the number of “turn around” dUnit in the (I) intermediate dUnits is no bigger than one: for case 1 to case 4, one “turn around” dUnit; for case 5 to case 8, no “turn around” dUnit.
The following section described the detail path creating rule as shown in Figure 9.
**Case 1: send to southeast (S_C<D_C & S_R<D_R)**

If the current dUnit is source node (S), configure the dSwitch as “read from memory manager and write to east”,

If the current dUnit is destination node (D), configure the dSwitch as “read from north and write to memory manager”,

*[Figure 9 The eight cases of transfer data in NoC]*
If the current dUnit belong to one of the three situations below, it will be regard as intermediate node (I):

- \( M_R = S_R \& S_C < M_C < D_C \): now the dSwitch will be set as W TO E (read from west and write to east);
- \( M_R = S_R \& M_C = D_C \): now the dSwitch will be set as W TO S;
- \( M_C = S_C \& S_R < M_R < D_R \): now the dSwitch will be set as N TO S

If the current dUnit belongs to none of the above situations (U), the dSwitch will be set insensitive to current instructions.

**Case 2:** send to northeast (\( S_C < D_C \& S_R > D_R \))

If the current dUnit is source node (S), configure the dSwitch as M to E

If the current dUnit is destination node (D), configure the dSwitch S to E

If the current dUnit belong to one of the three situations below, it will be regard as intermediate node (I):

- \( M_R = S_R \& S_C < M_C < D_C \): now the dSwitch will be set as W TO E;
- \( M_R = S_R \& M_C = D_C \): now the dSwitch will be set as W TO N;
- \( M_C = S_C \& S_R < M_R < D_R \): now the dSwitch will be set as S TO N.

If the current dUnit belongs to none of the above situations (U), the dSwitch will be set insensitive to current instructions.

**Case 3:** send to southwest (\( S_C > D_C \& S_R < D_R \))

If the current dUnit is source node (S), configure the dSwitch as M to W

If the current dUnit is destination node (D), configure the dSwitch N to M

If the current dUnit belong to one of the three situations below, it will be regard as intermediate node (I):

- \( M_R = S_R \& S_C < M_C < D_C \): now the dSwitch will be set as E TO W;
- \( M_R = S_R \& M_C = D_C \): now the dSwitch will be set as E TO S;
- \( M_C = S_C \& S_R < M_R < D_R \): now the dSwitch will be set as N TO S.

If the current dUnit belongs to none of the above situations (U), the dSwitch will be set insensitive to current instructions.
**Case 4:** send to northwest ($S_C>D_C \& S_R>D_R$)
If the current dUnit is source node (S), configure the dSwitch as M to W
If the current dUnit is destination node (D), configure the dSwitch S to M
If the current dUnit belong to one of the three situations below, it will be regard as intermediate node (I):
- $M_R=S_R \& S_C<M_C<D_C$: now the dSwitch will be set as E TO W;
- $M_R=S_R \& M_C=D_C$: now the dSwitch will be set as E TO N;
- $M_C=S_C \& S_R<M_R<D_R$: now the dSwitch will be set as S TO N.
If the current dUnit belongs to none of the above situations (U), the dSwitch will be set insensitive to current instructions.

**Case 5:** send east ($S_C<D_C \& S_R=D_R$)
If the current dUnit is source node (S), configure the dSwitch as M to E
If the current dUnit is destination node (D), configure the dSwitch E to M
If the current dUnit is $M_R=S_R \& S_C<M_C<D_C$, it will be regard as intermediate node (I), THE dSwitch is set as W to E,
If the current dUnit belongs to none of the above situations (U), the dSwitch will be set insensitive to current instructions.

**Case 6:** send west ($S_C>D_C \& S_R=D_R$)
If the current dUnit is source node (S), configure the dSwitch as M to W
If the current dUnit is destination node (D), configure the dSwitch W to M
If the current dUnit is $M_R=S_R \& D_C<M_C<S_C$, it will be regard as intermediate node (I), THE dSwitch is set as E to W,
If the current dUnit belongs to none of the above situations (U), the dSwitch will be set insensitive to current instructions.

**Case 7:** send to south ($S_R<D_R \& S_C=D_C$)
If the current dUnit is source node (S), configure the dSwitch as M to S
If the current dUnit is destination node (D), configure the dSwitch N to M
If the current dUnit is $M_C = S_C \& S_R < M_R < D_R$, it will be regard as intermediate node (I), THE dSwitch is set as N to S,
If the current dUnit belongs to none of the above situations (U), the dSwitch will be set insensitive to current instructions.

**Case 8**: send to north ($S_R > D_R \& S_C = D_C$)
- If the current dUnit is source node (S), configure the dSwitch as M to N
- If the current dUnit is destination node (D), configure the dSwitch S to M
- If the current dUnit is $M_C = S_C \& D_R < M_R < S_R$, it will be regard as intermediate node (I), THE dSwitch is set as S to N,
- If the current dUnit belongs to none of the above situations (U), the dSwitch will be set insensitive to current instructions.

**Example two:**
An instruction indicates dUnit 0000 as SOURCE and dUnit 1011 as DESTINATION, according to case 1, one and only one path can be created from S to D. dUnits on the path( S, D or I) will be sensitive to instructions, while other dUnit will be set as unoccupied and insensitive for this instruction.

![Figure 10 A example of data transfer between nodes](image)

For source dUnit, instruction will set the dSwitch works on “M to E” mode (read from memory manager and write to east).
For all the intermediate dUnits, they did not transfer date in the same way. Normally three
different situations exist.

- If \( M_R = S_R \) and \( S_C < M_C < D_C \), as dUnit 0001, its instruction decoder will set
dSwitch works on “W to E” mode (read from west and write to east);

- If \( M_R = S_R \) and \( M_C = D_C \), as dUnit 0010, the “corner” position, instruction decoder
there will set “W to S” mode (read from west and write to south);

- If \( M_C = S_C \) and \( S_R < M_R < D_R \), as dUnit 0110, here instruction decoder will set “N to
S” mode (read from north and write to south);

For destination dUnit, instruction will set the dSwitch works on “N to M” mode (read from
north and write to memory_manager).

For other dUnits such as 0100, 1000 and 1111 etc. instruction decoder will set the dSwitch
insensitive to current instruction.

The example above shows how a certain path created when instruction came. The local
instruction decoder works variety based on eight different Source/ destination position case
and the different dUnit types (S, D, I or U) on the created path. This path creating rule covers
all the possible data transfer path and it work correctly on this architecture.

### 3.3.5 Path conflict flag setting

Instruction decoder also provide the current occupation status of local dUnit, if the dUnit is in
a path that generated by previous instructions and the path is still valid, instruction decoder
will set an occupation flag which helps NoC-manager make routing decision.

### 3.3.6 Memory manager

Memory manager is introduced as intermediation of the communication between DMesh and
memory. One more function for the memory manager is sending a release flag when data is
output from DMesh to Memory. As a result, the NoC manager will arrange optimized path
due to the current usage of memory banks.
3.4 NoC manager

NoC manager is a function unit that not only can create, control and hold the traffic of dNoC by processing and sending control signals, but also manage the memory accessibility. Control signals are transferred through Bus, which connected to every instruction decoder contained in dUnit.

The NoC manager for HDiMArch is composed of a set of instruction register and a routing arbitrator which keep the DMesh free from conflict. The architecture is shown at the following Figure 11.

![Figure 11 NoC manager](image)

3.4.1 Instruction Register

Instruction register (Figure.12) is a circular-buffer\cite{13} which stores incoming instructions and send the raw instruction to arbitrator. Two counters (one read and one write) of the circular-buffer are included. Both read and write counters will move to next register if a read or write action happened, besides, a hold signal received from arbitrator will force write counter stay until a release signal received.
3.4.2 Arbitrator

Arbitrator is the core function unit of this HDiMArch. Arbitrator not only creates a certain data transfer interconnections by decoding instructions, but also monitoring the current status of dUnit usage can keep the running instruction generate no conflict.

The DMesh is mapped onto a graph representation consisting of nodes and edges. The dUnits are represented by nodes. Links are the edges between the nodes. If a (bidirectional) link exists between two nodes, the corresponding nodes will be connected in the method of path creating rule. In that way a DMesh can be mapped onto a graph representation. The transformation of the NoC graph representation to the hardware realization as Hardware Graph ARay (HAGAR) or graph matrix\textsuperscript{[14]}.

As shown in Figure 13, the horizontal wires (the number of wires equal to the dUnit the structure has) will represent the signals of active dUnits, the two inputs is the dUnit status which the current instruction needed and the dUnit usage status information collected from memory manager.
For example, if a path from dUnit 2 to dUnit 4 is created at the initiation. According to the rules we mentioned before dUnit 2, 3 and 4 will be occupied. Since every dUnit is released at the initiation. All signals to release each dUnit is set free. As a result, the detection of activated dUnit is 2, 3, 4 and the instruction judge is “OK”. If another path from dUnit 1 to dUnit 4 is required for the next instruction. Since the release signal from memory manager shows dUnit 2, 3, 4 is been used. So the detection of activated dUnit is 1 to 4 but the instruction judge is “NOT OK”.

At the arbitrator, a released instruction is received from instruction register, this instruction would be translated and the data transfer partition is determined under the path create rule which is already described. The arbitrator follows the same path create rule as instruction decoder complied. Nonetheless, the arbitrator does not need to configure the dUnit behavior (for instance: read from south and write to north), the arbitrator only needs to Figure out which usage status (Source, destination, intermediate or unoccupied) is for each dUnit.

Figure 13. The monitor function unit of HAGRA
Meanwhile, the arbitrator also receives flags which came from DMesh, by doing this as shown in figure 13, arbitrator keep watch on the current status of dUnit usage. If one of the dUnit the under processed instruction used is already occupied in DMesh, a conflict occurred and the current instruction will not be sent to control signal bus but sent back to its original register, what’s more, a hold signal, which can keep the write counter, will be sent as a feedback. The next instruction will not be processed until the current instruction has not confliction. This instruction will be broadcast through instruction bus, Then the release signal for signal register generated and make the write counter move down.

For example, the current DMesh usage is as shown by blue dUnit. The instruction under processed will organize a route from S2 to D2 (green dUnit), however, a conflict occurred at block “Jam” (red dUnit), as a result, the instruction will not be decoded by control signals to every dUnit until dUnit “Jam” is free from occupation.

![Diagram of traffic jam in DMesh](image)

**Figure 14 Example of traffic jam in DMesh**

### 3.5 Memory Bank

The memory bank is SRAM macros, typically 2 KB, a design time decision, as the goal is to align memory bank with the memory manager. Memory is controlled by NoC manager. The memory bank received address select, r/w and enable signals. The dNoC arbitrator provide a general purpose timing model using three delays, An initial delay before a loop, an intermittent delay before every r/w within a loop and an end delay at the end of loop before repeating the next iterations. These delays are used to synchronize the memory to register file streams with the computation. Individual delays can be changed computation which makes streaming elastic.
4 RESULTS AND DISSCUSION

4.1 test instructions

In the results chapter the HDiMArch and DiMArch is simulated, synthesized, analyzed and compared with the existing knowledge and theory presented in the frame of reference chapter.

Three kinds of test are established in the following passage (Figure. 14):
First, logic synthesis the fabric of HDiMArch with different parameters (different data width and memory depth) will be done. We analyze the different synthesis results from area, consumed power and maximum frequency;
Second, we add more buffers in the HDiMArch and repeat first step.
Third, we compare this fabric with the DiMArch in the maximum frequency and the occupied Area for each data network at same test environment.

Tests about the fabric

- Analyze the difference of Power, Area and frequency between structures with several data width
- Include more buffers to the HDiMArch and synthesis the fabric.
- Find the difference of Maximum frequency between DiMArch and HDiMArch

*Figure 14 Test flow*
The fabric mentioned in this report is developed by following the first four steps of the traditional ASIC design flow, that is: concept and market research, Architecture specification and RTL coding, RTL Simulation and logic Synthesis, Optimization and Scan Insertion. Since the HDiMArch architecture is parameterized, this fabric can be characterized at later stage for difference parameters such as data width, memory depth, levels of pipeline.

4.1.1 traffic pattern

In order to help observing the power consumption for the HDiMArch without AGU. We have the traffic pattern established in the test bench. Traffic pattern is very important factor for the performance of a network. In the following test, we use the Permutation bit-reverse\(^{16}\) traffic patterns. The power result is concluded after synthesis with timing characters.

4.1.2 synthesis environment

Normally the synthesis environment setting should follow below steps:

- Specify libraries
- Defined design environment
- Set design constraints
- Select compile strategy
- Read and optimize the design

The synthesis setting follow the below table:

<table>
<thead>
<tr>
<th>Set library</th>
<th>Tcbn90g_110g</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set operating conditions</td>
<td>NCCOM</td>
</tr>
<tr>
<td>Set wire load mode</td>
<td>Segmented</td>
</tr>
<tr>
<td>Define clock</td>
<td>Adjusted according to the test target</td>
</tr>
<tr>
<td>Synthesize effort</td>
<td>Medium</td>
</tr>
</tbody>
</table>
Our goal is to find out the frequency limitation of the memory architecture with different parameters by configuring the “clock period” parameter in the script file, and the timing, area and power features of each configured architecture. Moreover, the timing character file (.vcd) should of cause be considered for the power test.

4.2 Test the HDiMArch with different data width

In the memory architecture, we defined the data width equal to the bit width of data. For the following test we defined the block size as 32736 byte. Based on the above setting, we keep these parameters as constant and vary the data width (data width × memory depth = memory block size) 5 cases: 16 bit, 32 bit, 64 bit, 128 bit and 256 bit are selected. The target is to figure out the performance under different combination of data width and memory depth.

- The size of DMesh is a 3 by 4 mesh.
- All the area, frequency and power features are export to the files specified in script.

4.2.1 Area

The following table shows the number of cells and area for the circuit with different structures.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area(nm)</td>
<td>147787</td>
<td>259180</td>
<td>465253</td>
<td>929657</td>
<td>1806403</td>
</tr>
</tbody>
</table>

According to the chart below, it is obviously that the needed area increase dramatically together with the data width.
4.2.2 Power

The following table shows the consumed power by with different data width VS memory depth.

<table>
<thead>
<tr>
<th>Parameters (bit)</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power (W)</td>
<td>0.16</td>
<td>0.22</td>
<td>0.42</td>
<td>0.62</td>
<td>1.23</td>
</tr>
</tbody>
</table>

According to the chart below, it is concluded that the consumed power increased exponentially alone with the increase of data path. That is because the architecture with large data path has more cells. As a result, more power is consumed.
4.2.3 Clock frequency

The following table shows the maximum limitation of clock frequency of each architecture with different data width VS memory depth.

<table>
<thead>
<tr>
<th>Parameters (bit)</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock frequency (GHz)</td>
<td>1.14</td>
<td>1.10</td>
<td>1.08</td>
<td>0.99</td>
<td>0.97</td>
</tr>
</tbody>
</table>

As shown from the chart below, the Maximum clock frequency decreased with the increase of size of data path. On the point of 64 bit the curve did not decreased that sharp as from other point.

The author assumed that with the increase of data path have positive efforts on rise the maximum clock frequency while the increase number of cells has negative efforts on maximum clock frequency. As a result, the trade off of cells and data path affect the clock frequency decreased but not linear.
4.3 test the HDiMArch with different buffers

In this paragraph we compare the HDiMArch with different buffers. The default setting of this system is one buffer. The buffer is introduced to realize the pipelined mode. A pipelined mode is omitted for single cycle multi-hop transfer mode. If the currently occupied dSwitch is not source or destination node as introduced before, by using buffer, the concerned dSwitch route the data to neighbor dSwitch. At destination, data will be directly loaded to DRRA. The number of buffers indicates the number of times one dSwitch can be reused at same time.

4.3.1 Area

The following table shows the needed area for each HDiMArch with different buffers in 5 kind of data path VS memory depth.

<table>
<thead>
<tr>
<th>Area (mm)</th>
<th>16×2048</th>
<th>32×1024</th>
<th>64×512</th>
<th>128×256</th>
<th>256×128</th>
</tr>
</thead>
<tbody>
<tr>
<td>with no buffer</td>
<td>0.44</td>
<td>0.78</td>
<td>1.50</td>
<td>2.85</td>
<td>5.74</td>
</tr>
<tr>
<td>with 1 buffer</td>
<td>1.47</td>
<td>2.59</td>
<td>4.56</td>
<td>9.30</td>
<td>18.20</td>
</tr>
<tr>
<td>with 2 buffers</td>
<td>1.89</td>
<td>2.95</td>
<td>6.09</td>
<td>11.34</td>
<td>26.00</td>
</tr>
<tr>
<td>with 3 buffers</td>
<td>2.24</td>
<td>4.03</td>
<td>8.10</td>
<td>16.44</td>
<td>30.23</td>
</tr>
</tbody>
</table>
As shown below, the more buffers it has, the more area the architecture needed. The number of increased area for a newly added buffer is roughly the same in all 5 situations.

![Areas from buffer 0 to 3](image)

**Figure 18 The area chart 2**

### 4.3.2 Power

The following table shows the consumed power for each case with different buffers in 5 kinds of situations about data path VS memory depth

**Table 8 The power result 2**

<table>
<thead>
<tr>
<th>Power (W)</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 buff</td>
<td>0.08</td>
<td>0.14</td>
<td>0.27</td>
<td>0.48</td>
<td>0.91</td>
</tr>
<tr>
<td>1 buff</td>
<td>0.16</td>
<td>0.22</td>
<td>0.42</td>
<td>0.62</td>
<td>1.23</td>
</tr>
<tr>
<td>2 buff</td>
<td>0.20</td>
<td>0.41</td>
<td>0.75</td>
<td>1.69</td>
<td>3.09</td>
</tr>
<tr>
<td>3 buff</td>
<td>0.43</td>
<td>0.83</td>
<td>1.38</td>
<td>3.12</td>
<td>5.90</td>
</tr>
</tbody>
</table>

According to the chart below, it is concluded that the power increased with the trend of increased buffer.
4.3.3 Clock frequency

The following table shows the maximum clock frequency of each case with different buffers in 5 kinds of situations about data path VS memory depth.

<table>
<thead>
<tr>
<th>Frequency (GHz)</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>for 0 buff</td>
<td>1.7</td>
<td>1.57</td>
<td>1.48</td>
<td>1.32</td>
<td>1.27</td>
</tr>
<tr>
<td>for 1 buff</td>
<td>1.14</td>
<td>1.10</td>
<td>1.04</td>
<td>0.99</td>
<td>0.97</td>
</tr>
<tr>
<td>for 2 buffs</td>
<td>1.09</td>
<td>1.07</td>
<td>1.02</td>
<td>0.94</td>
<td>0.91</td>
</tr>
<tr>
<td>for 3 buffs</td>
<td>1.07</td>
<td>1.05</td>
<td>1.01</td>
<td>0.91</td>
<td>0.88</td>
</tr>
</tbody>
</table>

From the below chart it is shown that the architecture with no buffer has much higher Maximum frequency than any other ones. But no pipelined mode means the efficiency of the memory system will be highly decreased. What is more, the author assumed in larger mesh architecture (10 by 10), the pipelined mode will has less efficiency than smaller mesh architecture (4 by 4) since more nodes are introduced in larger mesh architecture.

On the other hand, if buffers are introduced, there is no obviously difference between the other three situations. The Maximum clock frequency remains stable in the range from 1.1 Ghz to 0.9 Ghz. The frequency decreased slightly alone with the increase of data path.
4.5 Comparison between DiMArch and HDiMArch

4.5.1 Maximum frequency in mesh 3 by 4
Here we compared the Maximum clock frequency show the difference of performance about DiMArch and HDiMArch. Since the two systems have different structures. In this section the report only compare the frequency in the mesh of 3 by 4. Again the data width VS memory depth varies in 5 situations.

Table 10 Freq compare between two systems

<table>
<thead>
<tr>
<th>clock frequency(GHz)</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDiMArch</td>
<td>1.34</td>
<td>1.26</td>
<td>1.18</td>
<td>1.13</td>
<td>1.09</td>
</tr>
<tr>
<td>DiMArch</td>
<td>1.14</td>
<td>1.10</td>
<td>1.04</td>
<td>0.99</td>
<td>0.97</td>
</tr>
</tbody>
</table>
Figure 21 Freq compare between HDiMArch and DiMArch

The table below shows the frequency difference of the two architectures with the mesh 3 by 4. According to the chart, since the HDiMArch has a simple structure than DiMArch, we can see the HDiMArch has roughly 18% higher maximum frequency than memory DiMArch at the same mesh size.

### 4.5.2 Maximum frequency with increasing mesh scale

In this section we enlarge the fabric scale to see the scalable feature for each memory system. The fabric scale varies from 3 by 4, 4 by 5, 5 by 8, 8 by 12 and 12 by 16. The size of the fabric will be extremely large when keeping increasing the scale. The data width and memory depth here we use for all of the fabric is 16b VS 2048.

#### Table 11 Result of scalable feature

<table>
<thead>
<tr>
<th>clock frequency(GHz)</th>
<th>3 × 4</th>
<th>4 × 5</th>
<th>5 × 8</th>
<th>8 × 12</th>
<th>12 × 17</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDiMArch</td>
<td>1.34</td>
<td>1.01</td>
<td>0.88</td>
<td>0.75</td>
<td>0.68</td>
</tr>
<tr>
<td>DiMArch</td>
<td>1.14</td>
<td>1.08</td>
<td>0.8</td>
<td>0.71</td>
<td>0.61</td>
</tr>
</tbody>
</table>
According to the above mentioned chart we noticed during all the situations HDiMArch has higher max frequency than DiMArch.

With the increase of the mesh size, the maximum frequency, the maximum frequency decrease linearly. As we introduced before, differ from the situation shown at Figure 22, the DiMArch is scalable, the reason why the frequency drops is the AGU and some control unit in the DiMArch is not synthesized during the whole test. These missing units play a significant role on the scalability of DiMArch.

Same frequency drop happened on the HDiMArch but the decrease became more and more slightly with the scale of DMesh. The scalable problem on HDiMArch is mainly because of increase size and more interconnection of NoC-Manager. The size of NoC-Manger increased with the trend of large scale. What is more, the number of interconnection from NoC-Manger to the dUnits is also raised. The next section will show the how the NoC manager effect the synthesis.
The critical path of each case is

<table>
<thead>
<tr>
<th>Mesh size</th>
<th>HDiMArch</th>
<th>DiMArch</th>
</tr>
</thead>
</table>

Figure 23 The critical path of DiMArch and HDiMArch in different Mesh size
4.5.3 Area and Power increase for NoC manager in HDiMArch

The above Figure shows the area percentage for NoC manager in HDiMArch with the increase of Scale. The DMesh is synthesized with one buffer.

![Power for NoC manager and DMesh](image1)

![Area for NoC manager and DMesh](image2)

*Figure 24. The area and power percentage of NoC manager in HDiMArch*
According to Figure 23 we can conclude the area and power occupation percentage by NoC manager become much smaller with the increase of scale. The increase interconnection and extra area generated by NoC manager less significant when compare it to the whole system. As a result, we assume the more optimized and simplified the NoC manager is. The better scalability the HDiMArch has.
5 CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

Through the above mentioned content, we can get the following conclusion:

- The data transfer confliction at the network becomes critical problem that impact the performance of NoC architectures.

- The HDiMArch is an extended DiMArch with NoC manager that provide HAGAR. The use of NoC manager is to solve graph problem in hardware is an area of research on its own and realize it in acceptable time. NoC manager configures the interconnection between nodes of NoC at runtime.

- The dSwitch is programmable by the controlling signal once a new instruction is qualified to form a partition.

- The increase width of data width provides HDiMArch a better performance with the cost of very large power consumption and area.

- The buffer used in pipelined mode to work out the confliction happened in data traffic jam. More added buffers will cost extra power consumption and needs more area with a decrease of maximum frequency.

- Although the HDiMArch provides the hardware interconnection creating resolution due to the introduction of NoC manager, however, the NoC manager for the current version is facing a scalable problem. The scale problem becomes not so serious with the increase of the scale of DMesh.
5.2 Future work

The next version of HDiMArch will be developed to connect with DRRA and adds the AGU and corresponding control unit. Applications will be mapped into the DRRA with HDiMArch to test the performance.

The comparison between DRRA with DiMArch and DRRA with HDiMArch in different parameters and scales will give a principle view of the merit and demerit of these distributed memory architectures.

What is more, the optimization of the NoC manager would be a good solution to resolve the scalable problem for HDiMArch.


