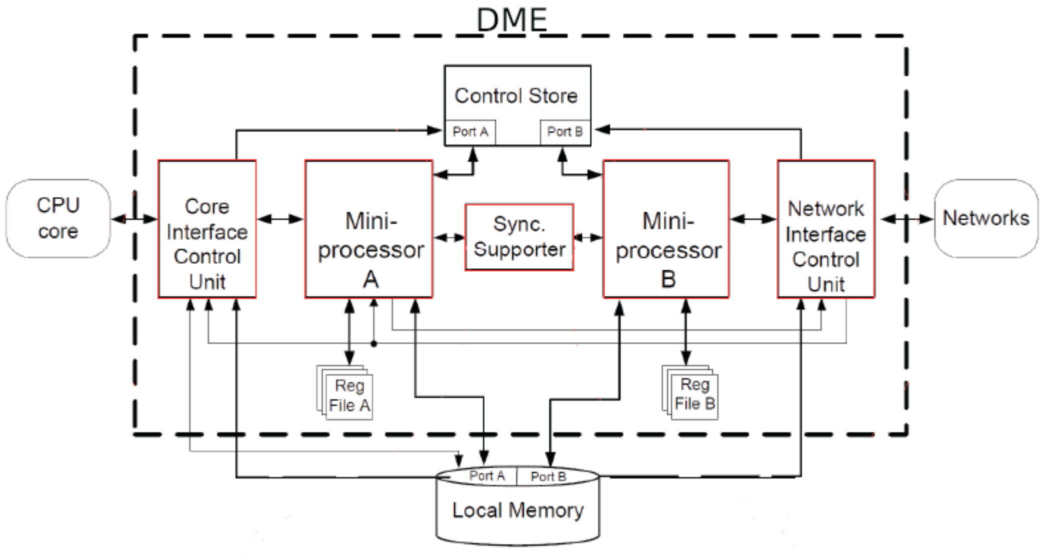
CHALMERS



Energy and Performance Model of DME

Master of Science Thesis in the Programme Integrated Electronic System Design

LIANXU HONG

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, November 2010

Energy and Performance Model of DME

LIANXU HONG

Examiner: SVEN KNUTSSON

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

[Cover:The internal structure of DME refer to Page 5 for more]

Department of Computer Science and Engineering
Göteborg, Sweden November 2010

Abstract

As a part of MOSART, an international EU project, an efficient and high-level framework is needed to help rapid and accurate estimation of energy and performance for McNoc (Multi-core Network on Chip). This thesis is about characterization of the Data Management Engine (DME) which is triggered by the Leon3 processor. In this thesis, a power model was established and used to characterize the micro-functions in the Data Management Engine (DME) to be able to get a quick estimation of energy and performance without having to run slow gate-level simulation and slow power extraction. The result was validated by being tested on a FFT program.

Keywords: MOSART, McNoC, Data Management Engine, Leon3 processor, Power Model, FFT program

Contents

Acknowledgements

I am deeply indebted to my supervisor Sandro Penolazzi and examiner Sven Knutsson. It is pleasant to do this research project with Sandro Penolazzi, because of his tremendous support for preparing this thesis and for his direction towards my research. I also want to thank Xiaowen Chen, Bin Liu, Zhonghai Lu for their help and advice during the research. Finally, I would like to thank my family for supporting me.

1 Introduction

The thesis is proposed within the frame of MOSART, an international EU project involving several partners, both from academy and industry. MOSART aims at designing and implementing a complete and advanced platform, based on the Networks on Chip (NoC) developed at KTH, to be used for wireless applications.

For several years, KTH has been doing research on Networks on Chip (NoC) and has built Nostrum [3], a mesh based NoC which uses bufferless switches implementing a deflective routing algorithm. Among the extensions proposed within the MOSART frame, is the introduction of a Data Management Engine (DME). The DME is a hardware block located in each Processor-Memory (PM) node of the network. Its main purpose is to bridge processor cache, local memory and Network Interface (NI). This is shown in Figure 1.1.

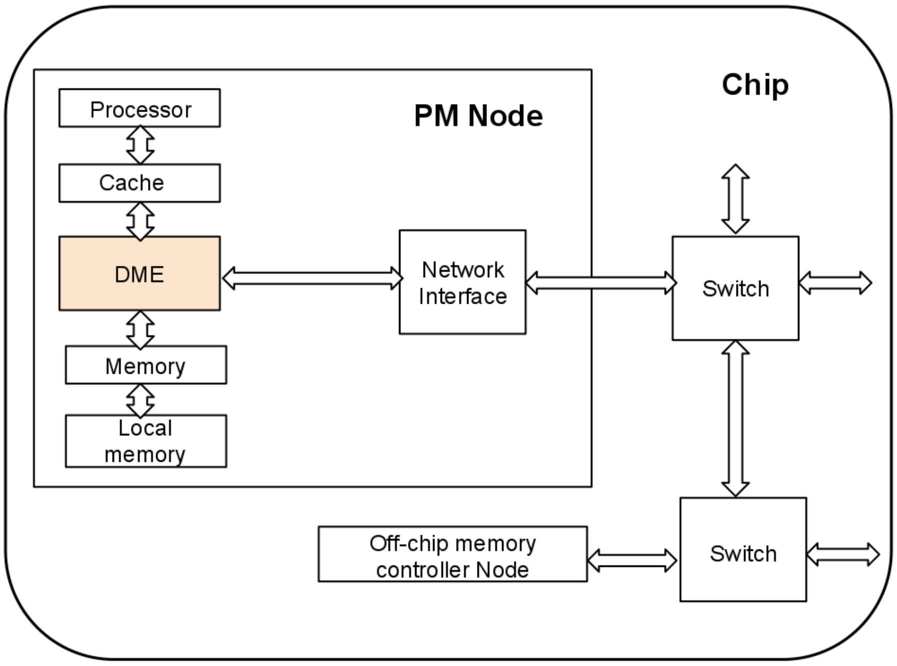


Figure 1.1: DME location [2]

The DME was developed to help the CPU core make operations related to memory more effective. Furthermore, the DME is an interface between the CPU core and the network. The DME makes it easy to connect the CPU core to the network without any modification and supports global memory operations for the CPU core. All in all, the DME supports several function related to memory as follows [1]:

- Distributed Shared Memory Addressing
  Maintain the illusion of a global, shared memory via physical addressing or virtual-to-physical memory translation for logical addressing.
- Memory access and Synchronization
  Local access and remote access. Consistency in access is guaranteed by several synchronization mechanisams when memory access happens on a shared region.
- Memory management
  Effectively allocate and deallocate memory through the whole multi-core chip.
- Support for coherent cache
  Updating cache directory, the copy of cache block.
- Data movement
  Multiple models of data transfer.

This thesis work was carried out at the Electronic Systems Department, School of ICT, KTH, Kista, Sweden, as a part of MOSART, an efficient and high-level framework is needed to help rapid and accurate estimation of energy and performance of the McNoC (Multi-core Network on Chip). The reason for doing this thesis is to be able to get a quick power estimate without having to run slow gate-level simulation and slow power extraction using the tool every time. This thesis is about the characterization of each micro-function running in the Data Management Engine (DME) which is triggered by the Leon3 processor (For more details about the Leon3 processor, refer to Section 2.1.1).

In Chapter 2, the Application platform and the DME is introduced, then the power model and the procedure of characterization are demonstrated, finally the results are reported and analysed. In Chapter 3, the validation of the characterization results are reported. In Chapter 4, conclusions and future work are discussed.

2 Characterization of DME

2.1 Network on Chip

McNoC (Multi-core Network on Chip) is a NoC (Network on Chip) implementation developed at KTH and it is a 2D mesh packet switched network with configurable size [2]. The latest McNoC refers to Figure 2.1 which consists of Processor-Memory (PM) and is connects by switches (also called routers, represented by circles in the figure). Each PM is a SoC (System on Chip) which is composed of a processor (in this case the Leon3), and a Data Management Engine (DME,also called Dual Microcoded Controller) and is connected using the Advanced High-performance Bus (AHB) [4]. Since McNoC is a network, the memory is required to be able to be accessed by other nodes through switches. DME was introduced to manage the memory accessing. Private memory is only allowed to be accessed by the node itself and shared memory is allowed to be accessed by all nodes, both the node itself and other nodes. Instructions can only be stored in private memory in contrast to data that can be stored both in private memory and in shared memory.

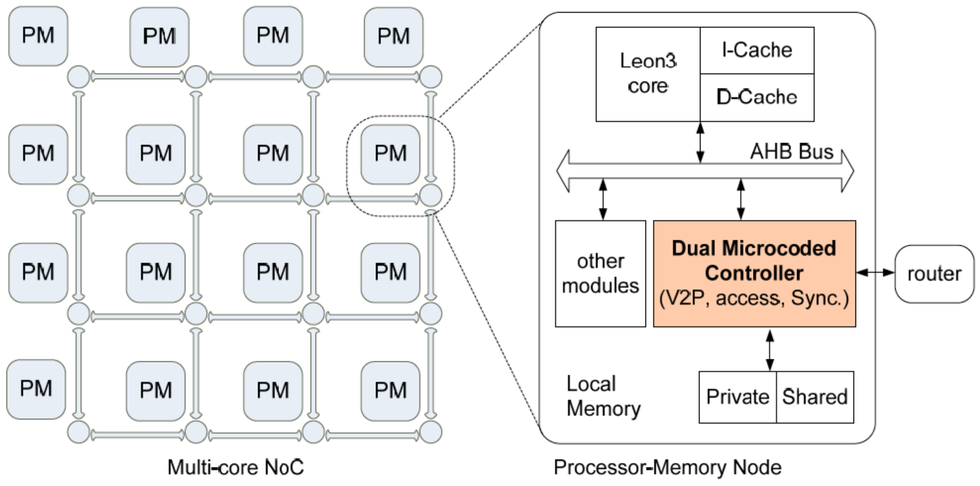


Figure 2.1: Application plaform [2]

2.1.1 Leon3

The Leon3 processor core [8] is a synthesizable VHDL model of a 32-bit processor compliant with the SPARC V8 architecture [5]. Four different cases regarding the instructions/data presence in the Leon3 cache were considered as follows:

1. instruction cache hit and data cache hit
   DME is in idle state and performs no operations.

2. instruction cache hit and data cache miss
   DME does not perform instruction fetch instead it receives a command from the processor, which in turn triggers the corresponding DME micro-function for data handling.

3. instruction cache miss and data cache hit
   Leon3 fetches instructions from Local private memory through DME. No other operations are performed.

4. instruction cache miss and data cache miss
   DME works both as a bridge between Leon3 and memory (for instruction fetch) and as an engine for data handling.

The data cache and instruction cache are configurable. Since the main reason for the DME to exist is for the handling of data in a distributed shared memory architecture, only the second and fourth case are considered. For the characterization of the micro-functions, the Leon3 processor is configured with data cache but without instruction cache. Data cache and instruction cache are not included for validation of the characterization results. The reason for this configuration will be disscussed in Section 2.6.1 and Section 3.2.

2.1.2 Switch

For the PM node communication, data packets are transmitted through switches. One switch has 5 inputs and 5 outputs for duplex communication with 5 different targets, and the switch is thus connected with 4 neighbour switches and 1 PM node. When receiving a Packet Data Unit (PDU), the switch would check the destination and guide the PDU to the next switch or its connected PM node. The hardware view of the switch is shown in Figure 2.2. PDU going through switch will cause 1 cycle delay which will be observed in the time consumption of each micro-function.

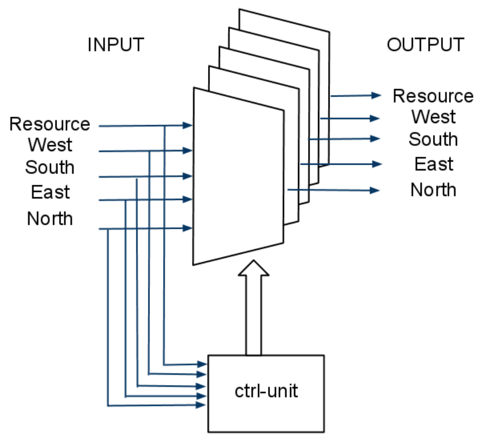


Figure 2.2: Switch model fabric [6]

2.2 Data Management Engine

As shown in Figure 2.3, the DME contains six main parts, namely, Core Interface Control Unit (CICU), Network Interface Control Unit (NICU), Control Store, Mini-processor A and B, and the Synchronization Supporter.

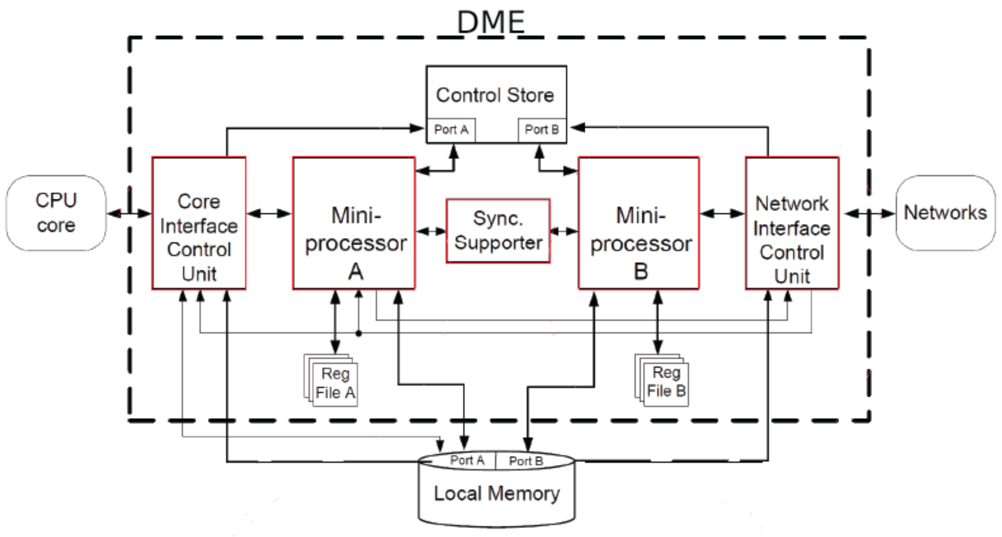


Figure 2.3: DME internal structure [2]

The Mini-processor A executes microcode from the control store through port A and uses register file A for temporary data storage. Its operation is triggered by a command from the local Leon3 processor. Mini-processor B executes microcode from the control store through port B, and uses register file B for temporary data storage. The Mini-processor B is similar to the Mini-processor A. The difference is that, rather than being triggered by a command from the local Leon3 processor, the Mini-processor B is triggered by a command from a remote Leon3 processor via the interconnect. As the names suggest, the CICU is the interface with the CPU core and the NICU is the interface with the network. The CICU receives local memory read/write access requests as local commands from the core, activates the Mini-processor A to process the requests, and receives replies from Mini-processor A or the local memory. The NICU receives remote memory read/write access requests as remote commands from other cores, activates Mini-processor B to process the requests and then sends back the replies to the interconnect. The Control Store works as an instruction cache. For each command, it loads the corresponding piece of microcode from the local memory (if not already in the Control Store) and provides the microcode as instructions to the two Mini-processors. The operation of the Control Store is completely controlled by the two Mini-processors. The Synchronization Supporter coordinates the two Mini-processors A and B for the exclusive access of shared variables, such as locks and semaphores. Register files A and B are temporary data storages, which may be considered as part of Mini-processor A and B, respectively. Due to this structure, the DME can process two streams of commands in parallel, one coming through the CICU and processed by Mini-processor A, and the other coming through the NICU and processed by Mini-processor B. During parallel computation, particular attention must be paid that the two Mini-processors do not request concurrent access to the same memory location. This is handled by the synchronization block.

2.2.1 Distributed Shared Memory

Memories are distributed in each node and tightly integrated with the processors. All local memories can logically form a single global memory address space. However, we do not treat all memories as shared. As illustrated in Figure 2.4, the local memory is partitioned into two parts: private and shared. Two addressing schemes are introduced: physical addressing and logic (virtual) addressing. The private memory can only be accessed by the local processor and the addressing is physical. All of the shared memories are visible to all nodes and are organized as a Distributed Shared Memory (DSM) and this memory uses virtual addressing. The boundary address between the private memory space and the shared memory space is configurable.

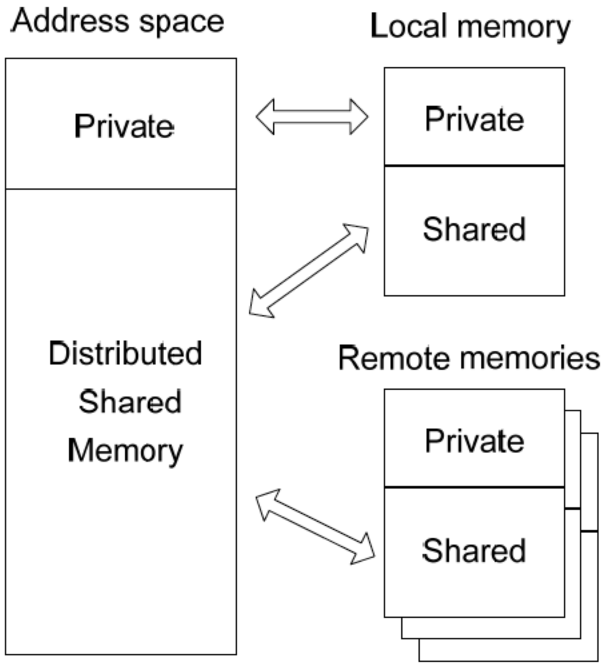


Figure 2.4: DSM organization [2]

2.3 Introduction of Characterization

The steps involved in the thesis procedure are shown in Figure 2.5. At the beginning, the design package of the DME was given together with some programs that were working well at RTL level. The first step is to synthesize the RTL model of the DME, simulate and verify that the gate-level version functionally works correctly. If the netlist is working well, then the characterization can go on, otherwise the block should be returned to the developer. Use the power model which will be introduced in Section 2.4 to charaterize the DME. The power model is composed of extracting the simulation time at RTL level and using the simulation time to characterize the micro-functions running in each block of the DME at gate-level. Finally, validate the results with one or more real applications to check whether this power model can give a quick estimation of energy and performance of the DME.

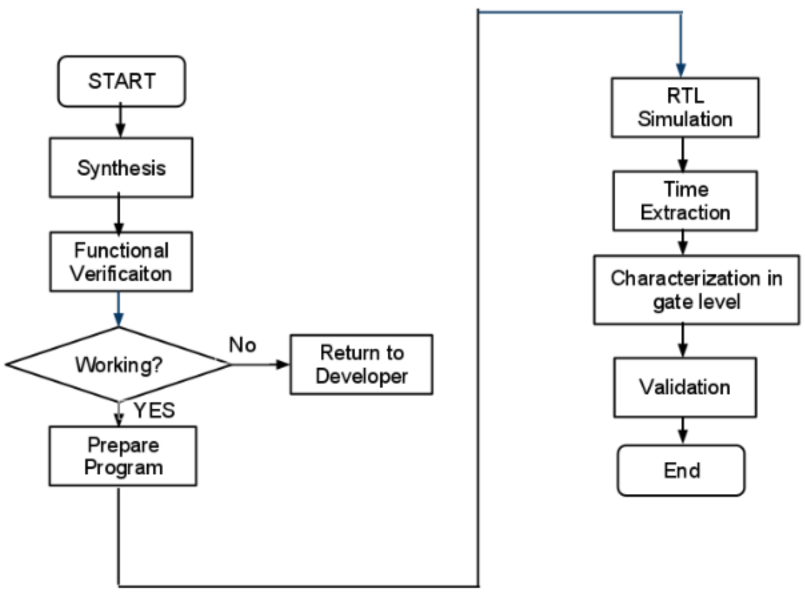


Figure 2.5: Procedure of thesis

2.3.1 Correspondence Between Leon3 and DME

Since the DME is triggered by the Leon3 processor and there is a one to one mapping between Leon3 commands [8] and DME micro-functions, the DME characterization process is basically done by controlling the instructions that are executed by the Leon3 processor. In this respect, the power model, which will be introduced in Section 2.4, will be used to estimated the power. Table 2.1 shows the correspondence between Leon3 instructions and DME micro-functions. Each DME micro-function itself is composed of several micro-instructions, but the characterization was executed at the micro-function level. As showed in Table 2.1, all the load/store related Leon3 instructions trigger the LOAD_WORD and STORE_WORD DME micro-function respectively. The ldd (load double word) and std (store double word) Leon3 instructions trigger the corresponding DME micro-function twice. The DME micro-function BLOCK/UNLOCK is implemented by predefined C-Macros that need to be manually inserted into the application software. All the Leon3 instructions which are not load/store related do not trigger any DME micro-functions. The instruction fetch is also implemented through the LOAD_WORD micro-function. Some micro-functions like LOAD_BYTE that will be developed in future research is now redirected to the LOAD_WORD micro-function. Table 2.2 reports the redirected relationship inside DME. It enables the use of the same model to get the characterization result of the new micro-functions.

Leon3 instruction	DME micro-function
ld (load word)	LOAD_WORD
lduh (load unsigned half word)	LOAD_HWORD
ldub (load unsigned byte)	LOAD_BYTE
ldsh (load signed half word)	LOAD_HWORD
ldsb (load signed byte)	LOAD_BYTE
ldd (load double word)	LOAD_WORD +LOAD_WORD
st(store word)	STORE_WORD
sth(store half word)	STORE_HWORD
stb(store byte)	STORE_BYTE
std(store double word)	STORE_WORD +STORE_WORD
Software-defined macros	BLOCK/UNLOCK
other instructions	—

Table 2.1: Relation between Leon3 instructions and DME micro-functions

LOAD_BYTE LOAD_HWORD	LOAD_WORD
STORE_BYTE STORE_HWORD	STORE_WORD

Table 2.2: Redirected relationship in DME

2.3.2 Synthesis

There are several programs included in the DME package which can be used to test whether each block functionally is working well at RTL level. After synthesis, these programs were also used to check whether the netlist was working correctly. The netlist for the CICU was unfortunately out of control and was returned to the developer. Therefore it is not included in the following descriptions.

2.3.3 Platform

The following platform (Figure 2.6) has been established for the entire characterization which is a 7x1 (7 rows, 1 column) network with the memory addresses and names specified in Table 2.3. In the characterization, the source node is node 1 which means that the program is stored in the local private memory of node 1. The source node initiates the data access request to local private memory, local shared memory and also to shared memory in the destination nodes (node 2-7).

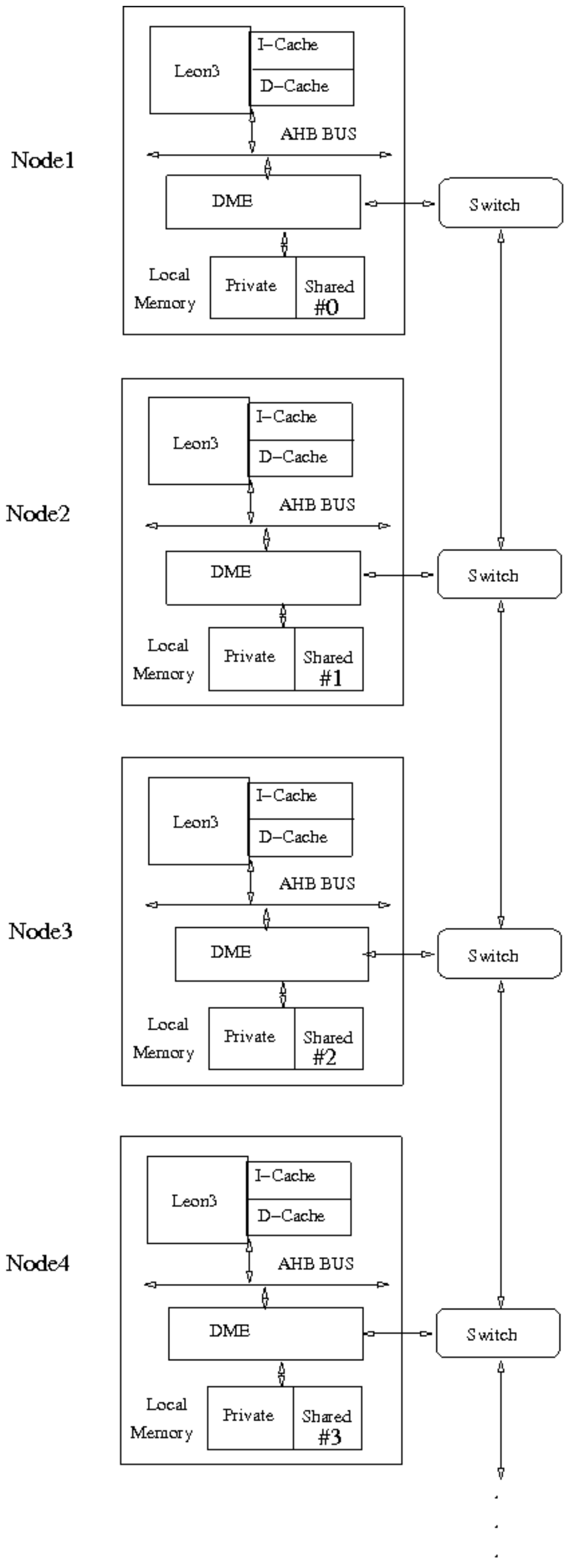


Figure 2.6: The platform used

Memory Space	Memory Name
0x40000000 0x401FFFFF	Private Memory
0x40200000 0x4021FFFF	Shared Memory #0
0x40220000 0x4023FFFF	Shared Memory #1
0x40240000 0x4025FFFF	Shared Memory #2
0x40260000 0x4027FFFF	Shared Memory #3
0x40280000 0x4029FFFF	Shared Memory #4
0x402A0000 0x402BFFFF	Shared Memory #5
0x402C0000 0x402DFFFF	Shared Memory #6
0x402E0000 0x402FFFFF	Shared Memory #7

Table 2.3: Memory space

2.4 Power Model

The power model used to test the power of each instruction is composed of Nops and the instruction under test (IUT).The testing program is shown in Figure 2.7.

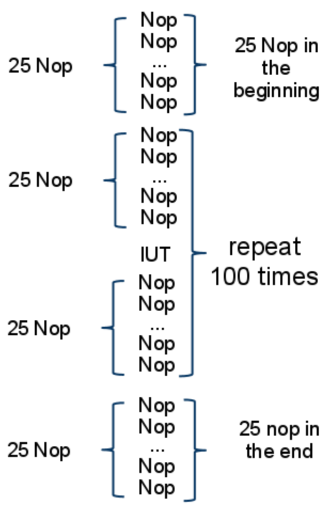


Figure 2.7: Testing pragram

Each time, one instruction is under test. This IUT is repeated 100 times to get the average energy performance. Between two IUTs of these repetitions, 50 nop instructions has been inserted to make sure that each time one IUT has been executed. And another 50 nop instructions has been insterted in the beginning and end of the testing part of program to isolate the testing from initialization instructions and terminating instructions. For example, if the IUT is ld, it will be repeated 100 times. Between each two ld, 50 nop instructions will be insterted. And another 50 nop instructions will be inserted in the beginning and end of the testing part of program. After the characterization of ld is finished, another IUT, e.g. st, will be tested in the same way and be characterized.

The following formula has been used to establish the power estimate. First, the instruction under test would be Nop, which means Leon3 in an idle state. When Leon3 is in an idle state, there would be no instruction fetch and data fetch from DME which means DME is also in an idle state. The following energy and power estimate is related to DME. When the Leon3 processor executes the instruction under test, the corresponding micro-function or idle state of the DME will be triggered. The simulation time would exclude 25 Nop in the beginning and the end since there is some delay between Leon3 and DME. So the testing part of program is composed of 100 IUT plus 5000 Nop.

$$E_{5000Nop} = P_{5000Nop} * t_{simulation} \tag{2.1}$$

where:

- $E_{5000Nop}$ is the energy of DME when5000 Nop executed Leon3.
- $P_{5000Nop}$ is the power of DME when 5000 Nop executed in Leon3.
- $t_{simulation}$ is the simulation time of DME when 5000 Nop executed in Leon3.

The energy of each Nop would be:

$$E_{Nop} = \frac{E_{5000Nop}}{5000} \tag{2.2}$$

$$E_{IUT} = E_{100IUT}/100 \tag{2.3}$$

where:

- E_{IUT} is the Energy of 1 micro-function under test executed in DME.
- E_{100IUT} is the Energy of 100 micro-functions under test executed in DME.

The Energy of 100 micro-functions under test executed in DME.

$$E_{100IUT} = E_{5000Nop+100IUT} - E_{Nop} * 5000 \tag{2.4}$$

where:

- E_{100IUT} is the Energy of 100 micro-functions under test running in DME.
- $E_{5000Nop+100IUT}$ is the Energy of 5000 Nop and 100 micro-functions under test of DME when executed 5000 Nop and 100 corresponding instructions executed by Leon3.

The total simulation time was extraction from the gate-level simulation.

2.5 RTL simulation

Firstly, the beginning and end time of the testing part of the program should be extracted at RTL-level to get the running time of the program. The reason for doing this is that the time can not be oberserved by gate-level simulation. After getting the time, the DME characterization will be carried out on the gate-level netlist. The time consumption in cycles can be observed in the console which is shown in Figure 2.8, Figure 2.9 and Figure 2.10.

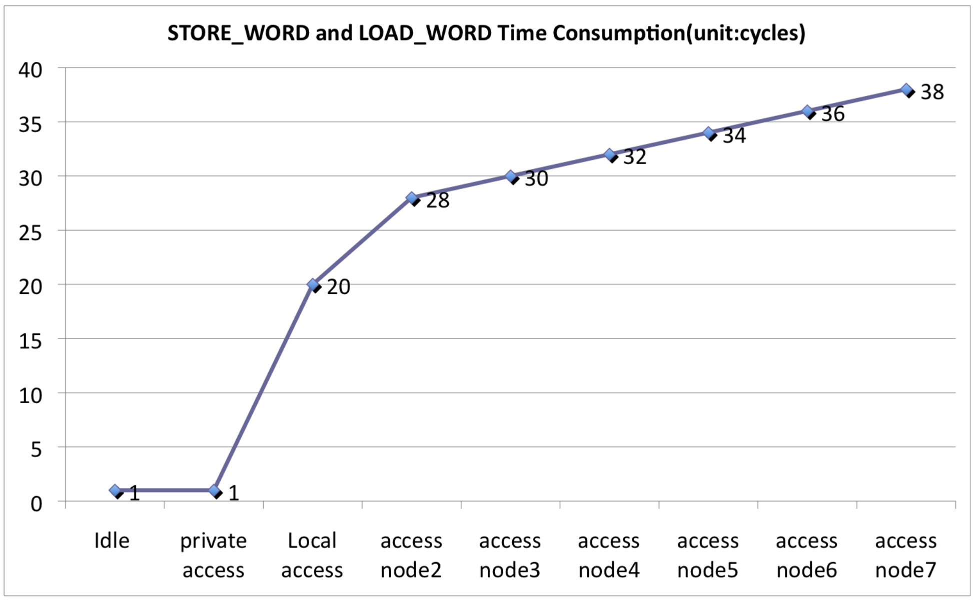


Figure 2.8: STORE_WORD and LOAD_WORD time consumption

Looking at the time result, which is expressed in number of cycles, it can be seen that 19 more cycles are used for local shared memory access compared to local private memory access due to the operation of virtual-to-physical address translation performed by the DME. 8 extra cycles are needed to perform remote access to an adjacent node. The cause of these extra cycles is that the operation requires traversing 2 switches two times each and that each switch takes 1 cycle to move a packet from input to output, 4 cycles are consumed by the switches, while the remaining 4 cycles are consumed in the remote PM node. Finally, each extra switch included in the communication path introduces 2 clock cycles of delay.

From Figure 2.9 and Figure 2.10, it can be found that BLOCK micro-functions always takes 4 cycles more than the UNLOCK micro-function. Since BLOCK and UNLOCK micro-functions are always being used in pairs, the virtual-to-physical address translation will cost time in BLOCK but do not need to be recalculated in UNLOCK.

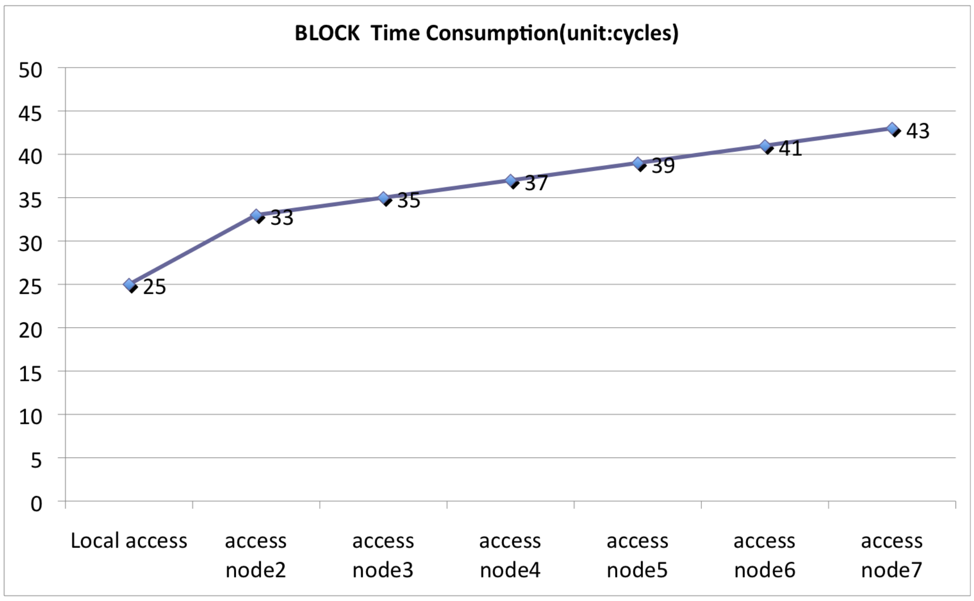


Figure 2.9: BLOCK time consumption

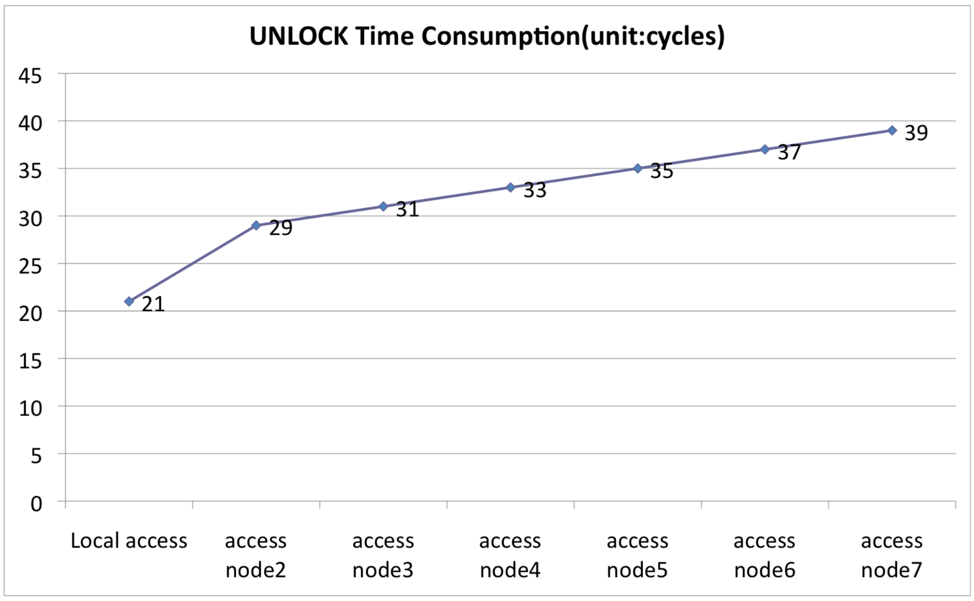


Figure 2.10: UNLOCK time consumption

2.6 Gate Level Simulation

Gate-level simulation with the time achieved at RTL-level simulation generates files with accurate time and power reports. Using the power model, the result of each block can be observed and reported in the following part.

2.6.1 Result for Each Component

From the power model, it is known that the micro-function under test needs to avoid effect from instruction fetch when the characterization is proceeding. In this section, the result of the characterization of each block of DME would be demonstrated under the condition that Leon3 is configured with instruction cache, resulting in a hit condition, and without data cache. Different tables refer to different micro-function.

The idle state energy consumption per cycle of each component of the DME is reported in Table 2.4. Here, the idle state means that Leon3 does not fetch instruction nor data from DME. As mentioned in previous section Mini-processor A and B have similar inner structure, so their idle states consumes more or less the same energy.

Block	Energy per cycle(pJ)
MiniProcessorA	17.33
MiniProcessorB	17.42
NICU	5.76
Synchronizor	0.61

Table 2.4: The idle state of each block of DME

When some operations are performed, some blocks are in an idle state. The energy result for individual DME blocks allow us to estimate the working block of the DME. STORE_WORD and LOAD_WORD micro-functions are complementary to each other and cause a similar behaviour in the DME's internal logic, both for energy and time. The characterization results of them are shown in Table 2.5 and Table 2.6.

	Local private access		Local shared access	
Time[cycles]	1		20	
Energy[pJ]				
MiniA	17.33		375.95	
MiniB	17.42		348.35	
NICU	5.76		115.24	
Synchronizer	0.61		12.15	
	Remote shared access(2)		Remote shared access(3)	
Time[cycles]	28		30	
Energy[pJ]	node1	node2	node1	node3
MiniA	515.58	485.15	549.92	519.81
MiniB	487.67	500.08	522.50	534.91
NICU	161.71	164.09	173.23	175.61
Synchronizer	17.00	17.00	18.22	18.22
	Remote shared access(4)		Remoteshared access(5)	
Time[cycles]	32		34	
Energy[pJ]	node1	node4	node1	node5
MiniA	587.03	554.53	619.44	589.11
MiniB	557.34	569.79	592.17	604.58
NICU	184.77	187.15	196.28	198.66
Synchronizer	19.43	19.43	20.65	20.65
	Remote shared access(6)		Remote shared access(7)	
Time[cycles]	36		38	
Energy[pJ]	node1	node6	node1	node7
MiniA	656.54	623.83	690.79	658.49
MiniB	627.00	639.46	661.84	674.29
NICU	207.81	210.19	219.34	221.72
Synchronizer	21.86	21.86	23.08	23.08

Table 2.5: Energy consumption for STORE_WORD

	Local private access		Local shared access	
Time[cycles]	1		20	
Energy[pJ]				
MiniA	17.33		369.87	
MiniB	17.42		348.35	
NICU	5.76		115.24	
Synchronizer	0.61		12.15	
	Remote shared access(2)		Remote shared access(3)	
Time[cycles]	28		30	
Energy[pJ]	node1	node2	node1	node3
MiniA	514.94	485.18	549.31	519.84
MiniB	487.67	487.68	522.53	534.44
NICU	161.78	163.69	173.30	175.22
Synchronizer	17.00	17.00	18.22	18.22
	Remote shared access(4)		Remote shared access(5)	
Time[cycles]	32		34	
Energy[pJ]	node1	node4	node1	node5
MiniA	586.49	554.56	618.87	589.15
MiniB	557.37	569.32	592.20	604.12
NICU	184.84	186.76	196.35	198.28
Synchronizer	19.43	19.43	20.65	20.65
	Remote shared access(6)		Remote shared access(7)	
Time[cycles]	36		38	
Energy[pJ]	node1	node6	node1	node7
MiniA	656.05	623.90	690.34	658.56
MiniB	627.04	639.01	661.87	673.85
NICU	207.90	209.82	219.42	221.35
Synchronizer	21.86	21.86	23.08	23.08

Table 2.6: Energy consumption for LOAD_WORD

In private access, the only working part is the CICU which can be observed in RTL simulation while all other components get almost the exact same energy consumption as in the idle state since private memory address is physical addressing which means that the DME works as a bridge between Leon3 and private memory. For local shared access, Mini-processor A needs to calculate virtual-to-physical address transformation and consumes more energy than in the idle state while all other parts remains in idle state. Mini-processor B and NICU are supposed to work only when a remote access occurrs. When node 1 initiate a remote access for other nodes, for example node 3, the working component is NICU for node1 and NICU and Mini-processor B for node 3. It can be observed in Table 2.5 and Table 2.6 that the Mini-processor B of destination node consumed more energy than the Mini-processor B of the source node.

Table 2.7 and Table 2.8 report the characterization result for the BLOCK/UNLOCK micro-functions respectively. As mentioned before, there is no Leon3 instruction that is directly triggering the DME BLOCK/UNLOCK micro-functions. Instead, there are some predefined C-macros that need to be manually inserted in the application software by the programmer and that cause the DME BLOCK/UNLOCK micro-functions to execute. When calling the C-macros, the memory address to be blocked/unlocked has to be given by the programmer. It was also in this way that BLOCK and UNLOCK micro-functions for access to different memory locations were measured.

	Local private access	Local shared access
Time[cycles]	-	25
Energy[pJ]		
MiniA	-	449.39
MiniB	-	427.92
NICU	-	141.40
Synchronizer	-	15.60

	Remote shared access(2)		Remote shared access(3)	
Time[cycles]	33		35	
Energy[pJ]	node1	node2	node1	node3
MiniA	592.79	564.24	564.24	627.62
MiniB	567.26	628.12	628.12	602.09
NICU	187.78	189.50	189.50	199.30
Synchronizer	19.78	20.48	20.48	20.99

	Remote shared access(4)		Remote shared access(5)	
Time[cycles]	37		39	
Energy[pJ]	node1	node4	node1	node5
MiniA	664.27	633.58	696.98	668.21
MiniB	636.93	697.78	671.76	732.56
NICU	210.83	633.58	222.35	224.08
Synchronizer	22.21	697.78	23.42	24.13

	Remote shared access(6)		Remote shared access(7)	
Time[cycles]	41		43	
Energy[pJ]	node1	node6	node1	node7
MiniA	733.63	702.90	768.45	737.56
MiniB	706.60	767.41	741.44	802.2
NICU	233.88	235.61	245.41	247.14
Synchronizer	24.64	25.34	25.85	26.56

Table 2.7: Energy consumption for BLOCK

	Local private access		Local shared access	
Time[cycles]	-		21	
Energy[pJ]				
MiniA	-		379.65	
MiniB	-		358.25	
NICU	-		118.36	
Synchronizer	-		13.10	
	Remote shared access(2)		Remote shared access(3)	
Time[cycles]	29		31	
Energy[pJ]	node1	node2	node1	node3
MiniA	524.28	494.93	559.11	529.59
MiniB	497.59	496.96	532.42	531.75
NICU	164.74	166.52	176.27	178.05
Synchronizer	17.35	18.00	18.57	19.22
	Remote shared access(4)		Remote shared access(5)	
Time[cycles]	33		35	
Energy[pJ]	node1	node4	node1	node5
MiniA	595.77	564.27	628.48	598.90
MiniB	567.26	566.55	602.09	601.33
NICU	187.80	189.59	199.32	201.11N
Synchronizer	19.78	20.43	20.99	21.65
	Remote shared access(6)		Remote shared access(7)	
Time[cycles]	37		39	
Energy[pJ]	node1	node6	node1	node7
MiniA	665.14	633.58	699.96	668.24
MiniB	636.93	636.13	671.76	670.92
NICU	210.85	212.64	222.38	224.17
Synchronizer	22.21	22.86	23.42	24.07

Table 2.8: Energy consumption for UNLOCK

BLOCK/UNLOCK micro-functions are only allowed to perform on shared memory, so there are no results for local private memory access. For local shared memory, only Mini-processor A and the synchronization supporter are working while other components are idle. When node 1 initiates a remote BLOCK for another node, for example node 3, the working component is Mini-processor A and NICU for node1 while NICU, Mini-processor B and Synchronization Supporter are working for node 3. This can be observed in Table 2.7. First, the Mini-processor B of the destination node consumes more energy than that of the source node whcich consumes the idle state energy multiplied by the corresponding number of cycles. Second, Synchronization Supporter in the destination node consumes more than that of the source node. When node 1 initiates a remote UNLOCK for other nodes, e.g. node 3, the working component for node1 is NICU while NICU and Synchronization Supporter are working for node 3. It can be seen from Table 2.8 that the Mini-processor B of the destination node consumes almost the same energy as that of the source node. With the same reason of the different time consumption between BLOCK and UNLOCK, BLOCK and UNLOCK are always used in pairs, the virtual-to-physical address translation will be realized in BLOCK but the result can be used directly in UNLOCK.

3 Model Validation

3.1 Validation Method

The previous characterization results have been validated using a real application. In this case, the processor Leon3 is without cache, which means that every instruction and data should be fetched through the DME. The validation has been carried out in the following way. The instructions has been stored in node1. Data has been stored in local private memory, local shared memory of node1 and remote shared memory of the other 6 nodes. The estimated DME energy has been calculated using Equation 3.1. The instruction fetch only relates to the component CICU which was not working, so the validation did not take instruction fetch into account.

$$E_{Estimated} = t_{idle} * p_{idle} + E_{Working} \tag{3.1}$$

where:

- $E_{Estimated}$ is the Energy estimate for the DME when running the real application
- t_{idle} is the idle time when running the real application
- p_{idle} is the idle power of DME
- $E_{working}$ is the estimated working energy of DME

The estimated working Energy was calculated by Equation 3.2.

$$E_{working} = \sum_{i=1}^{N} Num_{MFUT_i} * E_{MFUT_i} \tag{3.2}$$

where:

- Num_{MFUT_i} is the number of micro-function i under test
  A provided scripts can transform C program into Leon3 commands and get the

statistic result of each command. Shown in Table 2.1, is the one-to-one mapping between the Leon3's commands and the DME's micro-functions. Using the statistic result of Leon3 commmands and the one-to-one mapping between the Leon3's command and the DME's micrco-functions, the statistic result of each DME's micro-function can be calculated.

- E_{MFUT_i} is the Energy of micro-function i under test

When running the real application, besides the working time, there will be some idle time which is calculated by Equation 3.3.

$$t_{idle} = t_{total} - \sum_{i=1}^{N} Num_{MFUT_i} * t_{MFUT_i} \tag{3.3}$$

where:

- t_{total} is entire application running time which can be extracted from console

- t_{MFUT_i} is the consumption time of 1 micro-function i under test running in DME

$$E_{measured} = t_{total} * p_{measured} \tag{3.4}$$

where:

- $p_{measured}$ is the measured power for the entire application

- $E_{measured}$ is measured Energy for the entire application

Both the measured energy and the estimated energy has excluded the initialization energy of the DME. The error rate has been calculated by

$$ErrorRate = \frac{E_{Estimated} - E_{measured}}{E_{measured}} \tag{3.5}$$

3.2 Validation with Real Application

The characterization results presented above have been validated for accuracy against gate-level energy extraction. The application used for validation is a FFT program.

Leon3 was configured without both instruction cache (I-Cache) and data cache (D-Cache) to make sure that each instruction and data fetch was from memory. When the

CICU is working, the configuration will be better for validation since the instruction fetch is taken into account.

As mentioned in the validation method, validation has been performed on the same case studies that were used during characterization, i.e. data in local private, local shared and remote shared memory. The validation results are shown in Table 3.1. In each case study we are comparing measured energy versus estimated energy values for each of the DME components. Error in percentage was reported, calculated on the energy summation for Mini-processor A, Mini-processor B, NICU and Synchronizer. As evident from the table, the accuracy resulting from our model is very high, since the error is always below 0.1%. The explanation for such a high accuracy lies in the high regularity and determinism of the DME operation. It should be noted that for both the estimated and measured energy the energy for DME initialization was excluded since this was fixed in time and energy consumption as shown in 3.2.

	Local Private access		Local Shared access		Remote shared access(2)		Remote shared access(3)	
Total instructions[#]	683485		683485		683485		683485	
Total time[ms]	58.128		58.433		58.548		58.579	
Measured energy[μJ]					node1	node2	node1	node3
MiniA	50.359		50.715		50.822	50.725	50.848	50.752
MiniB	50.622		50.888		50.989	51.044	51.015	51.075
NICU	16.747		16.835		16.867	16.894	16.875	16.903
Synchronizer	1.765		1.744		1.778	1.778	1.779	1.779
Estimated energy[μJ]								
MiniA	50.356		50.641		50.744	50.721	50.770	50.747
MiniB	50.623		50.889		50.989	50.994	51.015	51.025
NICU	16.746		16.834		16.868	16.869	16.876	16.878
Synchronizer	1.765		1.774		1.778	1.778	1.779	1.779
Error[%]	-0.0025		-0.0615		-0.0639	-0.0656	-0.0639	-0.0622

	Remote Shared access(4)		Remote Shared access(5)		Remote shared access(6)		Remote shared access(7)	
Total instructions[#]	683485		683485		683485		683485	
Total time[ms]	58.610		58.640		58.671		58.701	
Measured energy[μJ]	node1	node4	node1	node5	node1	node6	node1	node7
MiniA	50.882	50.781	50.901	50.805	50.935	50.834	50.962	50.860
MiniB	51.042	51.098	51.068	51.124	51.095	51.151	51.122	51.178
NICU	16.884	16.911	16.893	16.920	16.902	16.929	16.911	16.921
Synchronizer	1.780	1.780	1.781	1.781	1.782	1.782	1.782	1.782
Estimated energy[μJ]								
MiniA	50.798	50.774	50.823	50.800	50.851	50.827	50.878	50.853
MiniB	51.042	51.051	51.069	51.078	51.095	51.105	51.122	51.131
NICU	16.885	16.887	16.894	16.896	16.903	16.905	16.912	16.913
Synchronizer	1.780	1.780	1.780	1.780	1.781	1.781	1.782	1.782
Error[%]	-0.0688	-0.0647	-0.0638	-0.0630	-0.0696	-0.0646	-0.0687	-0.0513

Table 3.1: Validation

ini time[cycle]=26730				
	MiniA	MiniB	NICU	Synchronizer
ini Energy[nj]	463.029	465.552	154.015	16.232

Table 3.2: Initialized energy of DME

4 Conclusion

4.1 Some Conclusions

The thesis goal to develop an efficient and high-level framework to help rapid and accurate estimation of energy and performance without having to run slow gate-level simulation and slow power extraction using the tool has been achieved. The characterization results can be used to estimate the energy consumption of a real application instead of using the tool to give the exact number. The results can be also used as a reference to improve the product. The established power model was successfully implemented in DME characterization.

4.2 Future Work

When DME develop other micro-instructions, it would be easy to get the time and energy consumption as the power model has been established and the scripts has been finished. For example, develop the LOAD_BYTE, LOAD_HWORD instead of redirecting to LOAD_WORD right now.

References

[1] X. Chen, *"DME-TP0.2 Data Management Engine Technique Report"*,2010

[2] X. Chen, *DME User Guide* Release 4.0, 2010.

[3] M. Millberg, *The Nostrum protocol stack and suggested services provided by the Nostrum backbone*, Technical Report TRITA-IMIT-LECS R 02:01, LECS, Dept. of IMIT, KTH, Stockholm, Sweden, 2003.

[4] D. Calusini, *Implementation of an AMBA-NoC Interface and Performance Evaluation of a NoC-emulated AHB Bus*, master thesis, KTH, 2008.

[5] SPARC International Inc.,*The SPARC Architecture Manual* Version 8, Revision SAV080SI9308.

[6] S. Penolazzi, *An Empirical Power Model of the Links and the Deflective Routing Switch in Nostrum*, master thesis, KTH, 2005.

[7] S. Penolazzi, A. Hemani and L. Bolognino,*A General Approach to High-Level Energy and Performance Estimation in SoCs.*In Journal of Low Power Electronics(JOLPE), Volume 5, Number 3, October 2009, pp. 373384.

[8] Aeroflex Gaisler, *GRIB IP Core Users Manual* Version 1.0.22,2010.