Grant Agreement number: 288574

Project acronym: **vIrtical**

Project title: SW/HW extensions for virtualized heterogeneous multicore platforms

Seventh Framework Programme

Funding Scheme: Collaborative project

FP7 -ICT -2011-7

Objective ICT-2011.3.4 Computing Systems

Start date of project: 15/07/2011 Duration: 36 months

D 4.2 *Hardware support for compression mechanisms at the NoC level*

Due date of deliverable: Month 22

Actual submission date: Month 23

Organization name of lead beneficiary and contributors for this deliverable: UPV
Work package contributing to the Deliverable: WP4

Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

APPROVED BY:

Partners	**Date**
All partners	**31st May 2013**

Table of Contents

Abstract ..... 3#

Glossary ..... 4#

1.# Introduction ..... 5#

2.# Related Work ..... 6#

3.# Compression Opportunities ..... 7#

4.# Baseline Network Interface (NI) ..... 9#

4.1.# Baseline Packet Format ..... 10#

5.# Parallel Compression Strategy ..... 12#

5.1.# Implementation ..... 13#

6.# Evaluation Results ..... 17#

6.1.# Compression Rate Achieved ..... 17#

6.2.# Implementation Overheads ..... 19#

6.3.# Discussion ..... 21#

7.# Conclusions ..... 24#

References ..... 25#

Abstract

In this deliverable we provide the research performed towards compression mechanisms researched in the context of the vIrtical project, in particular, in Task 4.1. The goal is to design compression strategies at the NoC level, allowing messages to be sent in a compressed manner, thus saving communication costs, mainly by reducing the number of transmitted flits and consequently the energy consumed. The provided mechanism relies on the abundance of memory data blocks filled with zeros, thus easily compressible by using a zero-detection strategy. These findings were obtained from D1.1.

In this deliverable we provide a hardware implementation for both compression and decompression at a generic network interface (NI). The mechanisms have been designed in isolated mode in order to be easily adapted to customized NIs used in the project.

Results show the effectiveness of the compression and decompression mechanisms and the low overhead they introduce. The percentage of traffic reduced by the compression strategy justifies the added resources for compression and decompression. The area overhead for the compression and decompression mechanisms required for a system with coherence support is 17.94% and 0% respectively whereas the added power consumption is 75.74% and 4%. However, traffic between memory and the L2 cache for the applications analyzed is reduced by a factor of 3, which justifies the overhead.

Glossary

ACRP: Aligned Consecutively Repeated Pattern

ADDR: Address

AMBA: Advanced Microcontroller Bus Architecture

AXI: Advanced eXtensible Interface

DST: Destination

FIFO: First In First Out

Flit: Flow control digit

FT: Flit Type

GPPA: General Purpose Processor Accelerator

ID: Identifier

MC: Memory Controller

nACRP: Non-Aligned Consecutively Repeated Pattern

nAnCRP: Non-Aligned Non-Consecutively Repeated Pattern

NI: Network Interface

NoC: Network-on-Chip

NZ: Non-zero

OCP: Open Core Protocol

SRC: Source

1. Introduction

This deliverable provides implementation solutions to compression strategies at NoC level. Traffic compression is an important aspect of a NoC setting as potentially we can save energy very easily. Data streams sent through the NoC can be compressed at injection and decompressed at ejection, thus travelling through the NoC in a compressed mode, saving power and even latency (messages will be shorter).

This work has been done taking a basic network interface (NI) as a baseline, thus not adopting any current NI technology. The aim for this choice is to show the overheads of the compression strategies in an isolated mode and, most important, being implemented as a single component that is ready to be incorporated to existing NI solutions. Indeed, the solutions take into account only the existence of a decoupling buffer at the NI for the injection of messages, and a reception buffer at the NI for the ejection of messages from the NoC. Communication between the core and the NI is not affected by our solutions and is completely orthogonal to the solutions we provide.

Indeed, there is not much related work about NIs and the way compression strategies can be plugged in to NoC systems. We provide a section in this deliverable about related work. In the next Sections, besides the related work section, we provide implementation aspects and evaluation analysis.

We have chosen to build a solution where the NI has a message-size interface with the end-node. In other words the end node transfers a whole message to the NI. We refer to this solution as ***parallel compression***. The NI needs to allocate a message-size buffer, with the corresponding buffering overhead. The benefit in this case comes with the fact that the NI has complete access to the message and has more opportunities for an effective compression. At ejection, however, the NI needs to reconstruct the message and thus, an extra latency is incurred before delivering the complete message to the end node. Because compression is targeted to large memory blocks, we opted for a parallel compression strategy.

This deliverable is closely linked to Deliverable 1.1, where compression opportunities were identified. Therefore, this current deliverable builds on top of the results of D1.1. However, the compression strategies analyzed in this deliverable have been designed with some assumptions of the target system. We assume the use of a coherence protocol running on top of the NoC, thus most of the traffic will be made of messages generated by the coherence protocol, including coherence commands embedded in short messages and memory blocks embedded in large messages. We will focus on compression opportunities of memory blocks, thus focusing on large messages injected into the network. Memory blocks are seized to 64-byte blocks (512-bit wide). Also, we assume the possibility of building a NoC with different virtual channels by physically replicating one VC-less network.

The compression solution provided in this deliverable, assumes certain packet formats in the network. In particular, the baseline flit format will be specified and customized to the compression solution provided. This is of need since the compressed messages need to be uncompressed at the destination; thus, compression information needs to be injected together with the messages. Also information for the coherence protocol must be included (source node address and memory address). Our flit size is set to 32 bits.

2. Related Work

When considering related work, we must distinguish two different subjects, Network Interfaces (NIs) as a whole and proposals of compression strategies for NoCs. Since designing a NI is not within the scope of this project, we have focused on NoC compression strategies.

Compression of the transmitted information does not seem to be a largely explored field. There are many papers on test data compression for NoCs, but on these papers the data is compressed offline, out of the network, and only decompression is applied in the NIs. Some examples can be found at [1] [2] [3] [4] [5] [6] . Since compression is not performed in the NoC, the strategies explained in these papers are not applicable to our needs. Other papers [7] also deal with compression of the routing table, which again is not what we were looking for.

Some papers however study NoC compression strategies, which may be applicable to our project. Some examples of these are [8] [9] [10] [11] [12] [13]

In [8] (2006), a real-time compression technique that reduces the amount of bits sent is presented. The USBR technique removes the bits which do not change, and sends some extra information (which bits change and how long the block of data is). This technique aimed at data streams where the most significant bits are less likely to change than least significant bits. In a memory coherence protocol this is not the case, so it is not applicable to the type of data involved in our project.

The authors in [9] (2007), integrate compression and decompression with a coherence protocol. Basically, a protocol processing core is included in multi-core nodes to perform both the coherence protocol, compression and decompression. This is an interesting approach but we prefer to add a small module with simple logic to the necessary NIs, rather than including an extra core. It is our intention to modify the existing components the least possible and keep our design as independent as it can be from other components or protocols.

In [10] (2008) two compression strategies are presented. The first one, named Cache Compression is performed between the L1 and L2 (inside the cache) and is meant for storage (squeezing more data in the same sized cache structure). The second one is called Compression in the NIs and it consists in compressing the information right before injecting it, so it is aimed for transmission. It is this second strategy that is more interesting for our purpose: a simple compressor/decompressor can be integrated in the NIs. The pitfall of this strategy is that it needs to integrate such modules in every NI, when it is not necessary for us, and that it uses a table-based frequent-pattern compression.

Adaptive data compression for on chip network performance optimization is presented in [11] . It uses a table-based strategy, improving it by using shared tables and pipelining compression and injection of the flits. It is interesting, but the nature of the data characteristic to our applications of interest makes a table redundant (since our compression target are compression strings, there is no need to use tables or calculate frequent patterns).

A similar approach is shown in [12] as well, except that instead of frequent patterns, it is based on frequent values. The authors claim that with a small code-book for end to end communications, that does not need to be the same among different cores, they reduce power consumption in the router up to a 16.7%. This proposal is again not applicable to our needs.

Finally, in [13] an interesting proposal is presented: Although it uses tables (in fact it needs not only compression tables but also candidate ones, increasing redundancy), it allows to completely eliminate some of the transmissions (in the best case), sending only a short message and using matching status bits in the directory, which simplifies and fastens miss handling.

Our particular solution is not compliant with any of these found in previous literature. We have developed our own new proposal based on the massive appearance of zeros in the applications analyzed. If further study should show that this is an anomaly, and other common applications lack this characteristic, a strategy such as the one presented in [13] would be a good option. However, with our current data, zero elimination is the best option.

3. Compression Opportunities

In this section we highlight the compression opportunities identified in Deliverable 1.1. These findings influence on the compression solutions we provide. Detailed information can be found in the mentioned deliverable.

Basically, in D1.1 we evaluated representative workloads for the targetted architecture and realized that long strings of only zeros were very common in the traffic between the L2 and the memory controller. We distinguished three cases:

- **Aligned Consecutively Repeating Patterns (ACRP)**. Patterns in this category are consecutive and aligned to byte size. One example is "000000010000000100000001…" where the pattern "00000001" is repeated three times (consecutive repetitions) and is byte aligned.
- **non-Aligned Consecutively Repeating Patterns (nACRP)**. This category groups those patterns that are still consecutive but not necessarily aligned to any data size. One example is "111100000001000000010000000 1…" where the pattern "00000001" is repeated three times (consecutive repetitions) and is not byte aligned.
- **non-Aligned non-Consecutively Repeating Patterns (nAnCRP).** This category groups those patterns not necessarily aligned to any data size and not consecutive, this is: a pattern that is repeated along the trace but we do not consider if it is consecutively repeated or not. One example is "11110111100100000011111000001111111 1…". where the pattern "1111" is repeated 5 times (not necessarily consecutive) and alignment is not considered.

In D1.1 we concluded that block aligned zero-strings, representing a large hit rate for all-zero streams, and furthermore easing the implementation at the NoC level of such compression mechanisms were the more interesting patterns. Nevertheless, taking into account that our transmission unit is one flit (4 bytes) and in this flit a header must be included, we realized that the most interesting type of pattern studied in D1.1 is in fact nACRP, since it allows us to totally avoid sending some of the flits, thus saving power and time, and does not impose any further restriction. In order to be compressed, zero strings must be of at least 25 bits and these 25-bit chunks must be aligned with the beginning of the flit, but this alignment is different from that studied in D1.1 and needs not be explicitly calculated or studied.

In Table 1 and Figure 1 we can see that 93.45% of the bits transmitted between L2 and the memory controller fall under the nACRP pattern type. Due to the aforementioned restriction, not all these bits will be eliminated from our transmission, but the distribution visible in the graph shows that more that 80% of the transmitted information can in fact be compressed with high resulting benefits.

(nACRP)	SHA1	SHA256	SHA512	AES-128-ECB	AES-256-ECB
00000000	93,45	93,26	93,42	92,83	92,74

Table 1. Comparison of the most interesting patterns (nACRP).

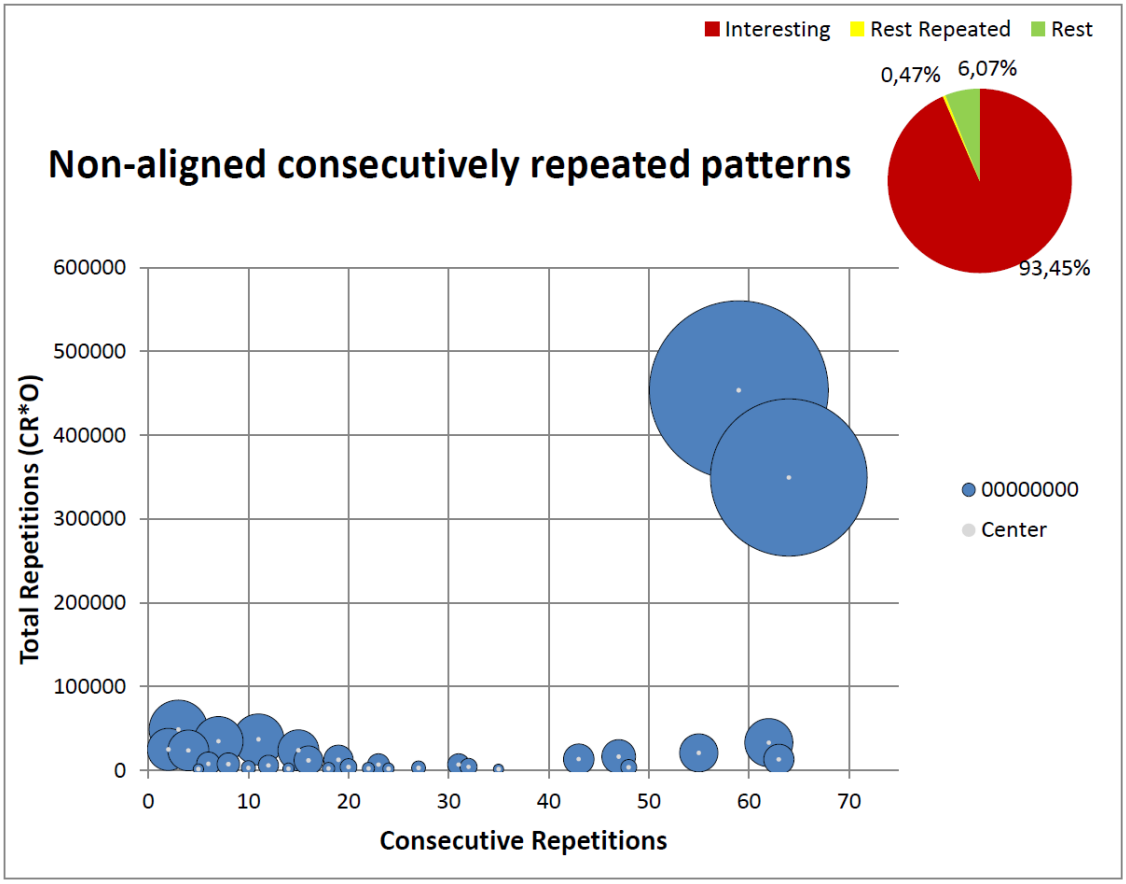


Figure 1. nACRP distribution.

Therefore, as a conclusive deduction, we target non-aligned long zero streams of memory data blocks as the main source of compression opportunities. In the following sections we describe the solution implemented. However, we first describe the NI used as baseline.

4. Baseline Network Interface (NI)

Figure 2 shows the schematic of the NI used as baseline. It implements two ports connected to the network, one in each direction. Therefore, injection and ejection of messages are supported by the NI. The NI has one buffer where messages to be injected are allocated by the injection logic. This logic is driven by the node connected to the network (through the NI) and is specific of the protocol used (AMBA, AXI, OCP, …). Likewise, the NI has an ejection buffer where messages received from the network are temporarily stored and delivered to the node. The logic to read from this buffer is also dependant of the protocol.

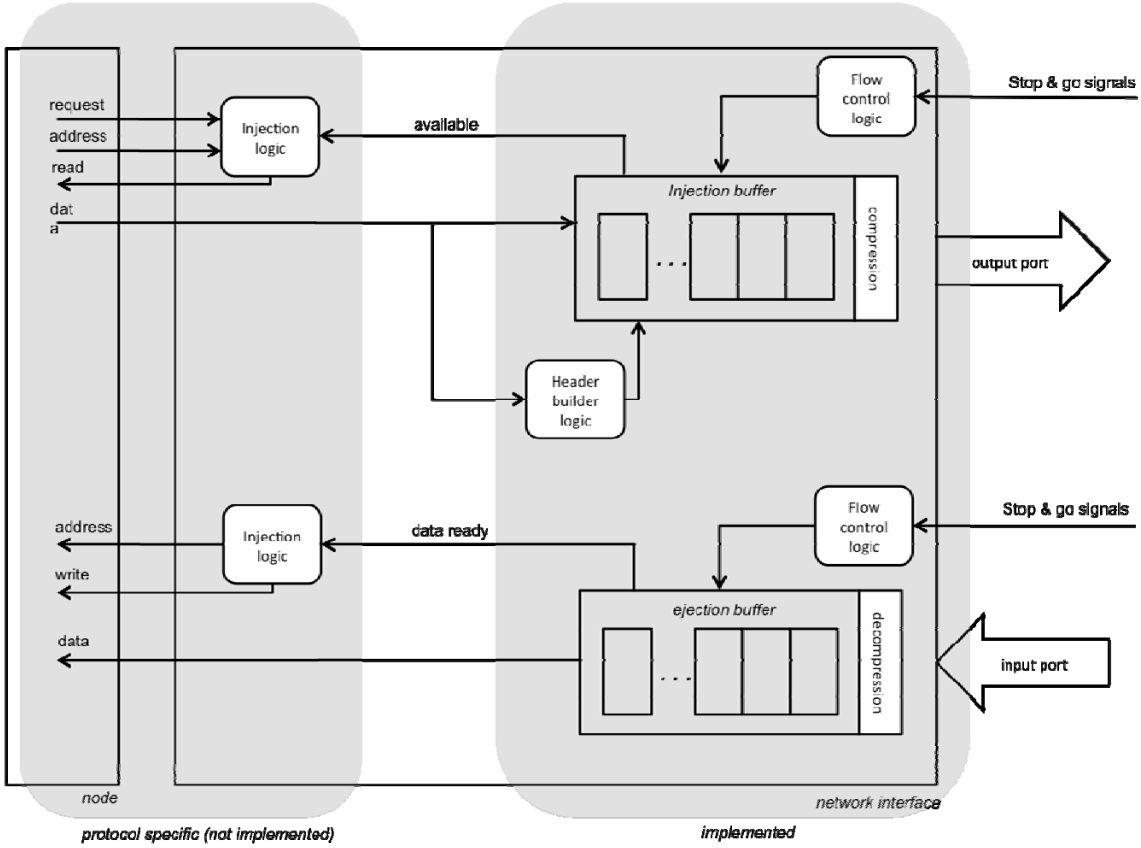


Figure 2. Baseline network interface (NI).

The size of both buffers will be customized to different configurations both in number of slots and slot width. We will evaluate different configurations ranging from one single slot (minimum area constraints) to eight slots (maximum buffer).

The NI has three extra logic blocks, all of them implemented in our solutions. The header builder logic is in charge of preparing the header of the flits of the message to be injected. At both sides of injection/reception we also implement a flow control logic. In particular, we implement the Stop&Go flow control protocol.

Finally, as shown in the figure, we add two logic blocks for the compression/decompression solution. These blocks are intimately linked to the buffers. We analyze the overhead of such solutions in terms of area overheads, operating frequencies, and power consumption.

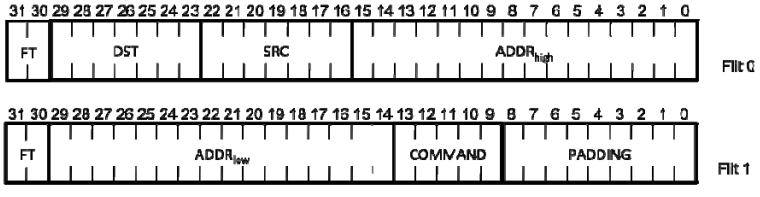
The NI evaluated in this work does not support virtual channels. Indeed, injection and ejection buffers consist of a single FIFO queue each. Virtual channels are an interesting addition to a network on chip, mostly for performance issues (boosting network throughput) but also for correctness reasons. For instance, in a request-reply protocol, both types of messages need to be decoupled by traveling through different virtual channels. Otherwise, protocol-level deadlocks can be induced. This is the case assumed in the GPPA device (more details in Deliverable 4.1). To fully support virtual channels and to decouple request messages from reply messages, we opt for a replication of the NI. Indeed, only the standard (not implemented) part

shown in the above figure will be replicated. The network can also be replicated providing separate resources for each traffic class. In addition, the results obtained in this deliverable in terms of area must be multiplied by the number of virtual channels needed in order to get an approximate estimation. Indeed, most of the area needs of the NI are caused by the buffers, which need to be replicated for the support of the request-reply protocol.

The compression solutions focus on memory blocks. Therefore, short messages triggered by the coherence protocol are not targetted. This means that compression logic needs to be used only in one particular virtual channel, the one used by long messages carrying memory blocks. This motivates also for a design of the NI in isolation of the virtual channel requirements, since the final solution will require a NI for the request layer with no compression logic built-in and a NI for the reply layer with built-in compression logic. Since we provide results for different configurations, it will be easy to obtain the final requirements for a system implementing a request-reply protocol with and without the compression mechanism.

4.1. Baseline Packet Format

The NI assumes the packet format shown in Figure 3 and Figure 4. The first one corresponds to short messages generated by the coherence protocol. The protocol will trigger a request (coded in the COMMAND field) to a specific destination end node (coded in the DST field) and with a 32-bit address (coded in the ADDR field). The format also codes the source end node sending the message (coded in the SRC field). Notice that a short message will be compounded in two flits injected through the NoC.

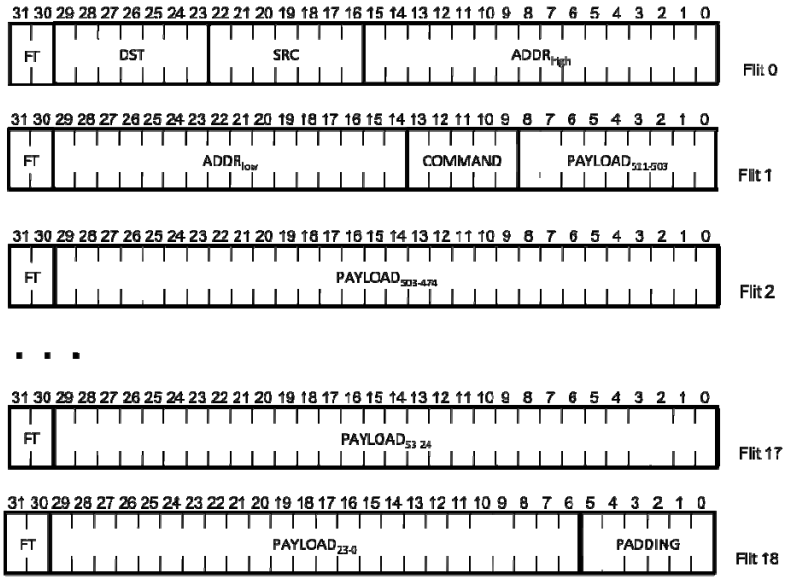


FT	Description
00	Non-valid
01	Tail
10	Payload
11	Header

Figure 3. Short packet format (baseline configuration).

Each flit includes two bits to code the flit type. Only two types are possible for short messages: Header (11) and Tail (01). Although we could use only one bit to code the type, we prefer to use two bits in order to make it compatible with long message formats.

For long messages more data needs to be injected, thus packet format is made of up to 19 flits. The first flit is identical to the short packet format one, including destination, source and part of the address. The second flit also includes the remainder bits of the address, and instead of the padding, a 5 bit coherence command and 9 bits of the payload (the most significant bits). The payload is the memory block that is being transmitted, (512 bits). Flit 1 to Flit 18 transport part of the payload (memory block). The final flit includes a pading of 6 bits. In long messages, three flit types are used: header (11), tail (01) and payload (10).



FT	Description
00	Compressed payload (or non valid)
01	Tail
10	Uncompressed payload
11	Header

Figure 4. Long packet format (baseline configuration).

5. Parallel Compression Strategy

In this section we develop the strategy for compressing messages with a parallel approach, where the whole message is simultaneously accessed from the NI.

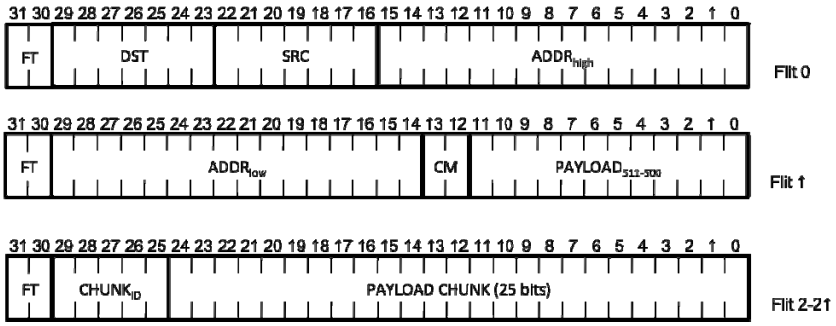
The memory blocks are the target of the compression solutions. The parallel approach takes the memory blocks and divides it in chunks. In our case, chunk size will be set to 25 bits. If all the bits of a chunk are set to zero, then the chunk is not transmitted over the network.

In order to support chunks and align them to flits, we need to redefine the packet format. The new packet format for long messages is shown in Figure 5. Short messages are not compressed and, therefore, the format shown in Figure 3 is used.

Long messages carry a memory block, which is the target of our compression strategy. 64-byte memory blocks are divided in 20 chunks of 25 bits each and a remainder set of 12 bits (the highest order bits of the memory block). 25-bit chunks have been selected as it allows a perfect match with the flit and packet width.

The first flit is not modified as it does not include payload bits. However, the second flit is modified to carry the remainder bits of the payload (as the block size is not multiple of chunk size). The remainder bits of the payload will not be compressed.

However, for every 25-bit chunck a flit can potentialy be injected into the network. The flit for a chunck includes the flit type field, the chunk id field (from 0 to 19, thus 5 bits) and the payload chunk (25 bits).



FT	Description
00	Non valid
01	Tail
10	Payload
11	Header

Figure 5. Long packet format (memory blocks) for parallel compression.

Figure 6 shows an example where a memory block is processed and 4 flits are injected. Flits #2 and #3 correspond to two chunks with some of their bits different from zero.

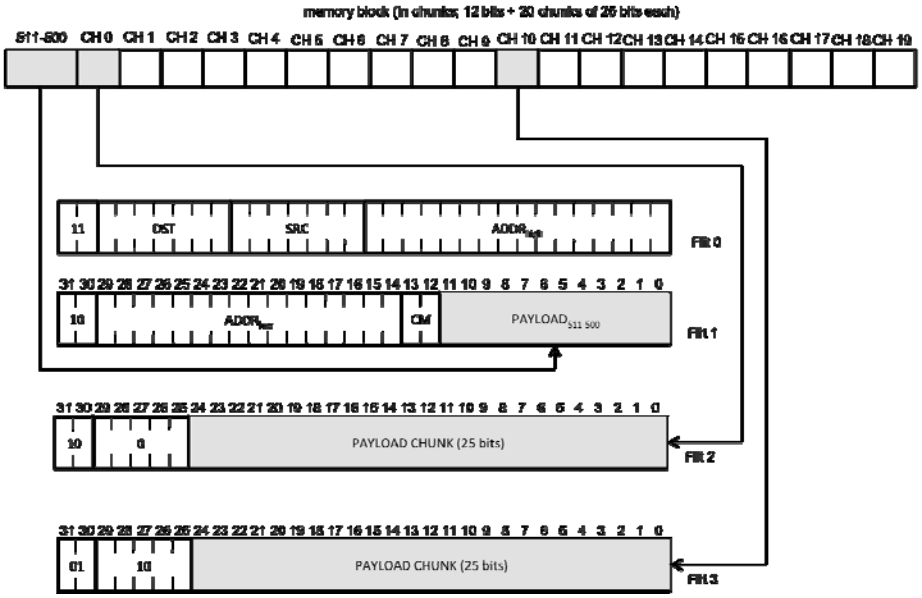


Figure 6. Example of parallel compression.

Thus, for a memory block with all its bits set to zero, the strategy will only inject into the network flits #0 and #1. This is the maximum compression rate that can be achieved with this strategy. Notice that the two flits carry other information fields (address, source, destination), which are typically non-zero fields.

5.1. Implementation

Figure 7 shows the logic details for the parallel compression strategy. In this approach we make the slot width of both injection and ejection buffers equal to the size of the message. Thus, whenever a new node requests to inject a new message, the injection logic copies the entire message into the injection buffer (if at least one slot is free).

When a message arrives to the NI (512 bits), the control information for the header (destination node, source node, and memory address) is added, thus arriving 558 bits in total. These bits are all treated as payload, except that the first 60 bits (the first two flits) are never compressed. Those bits correspond to the source, destination, address, and the 12 most significant bits of the payload.

The buffer slot is logically divided in chunks of 25 bits, including the 60 bits aforementioned. Chunks #2 to #21 are sent to the OR stage where all-zero chunks will be detected. Then, the resulting output of the OR stage will be used to compute the flit type and tail bit as well as the selection of the flit to inject into the network. The selection drives the multiplexer shown in the figure. Upon an injection of a flit, the associated output of the OR stage is reset in order to allow the injection of the next flit for the packet. Let us describe each stage details.

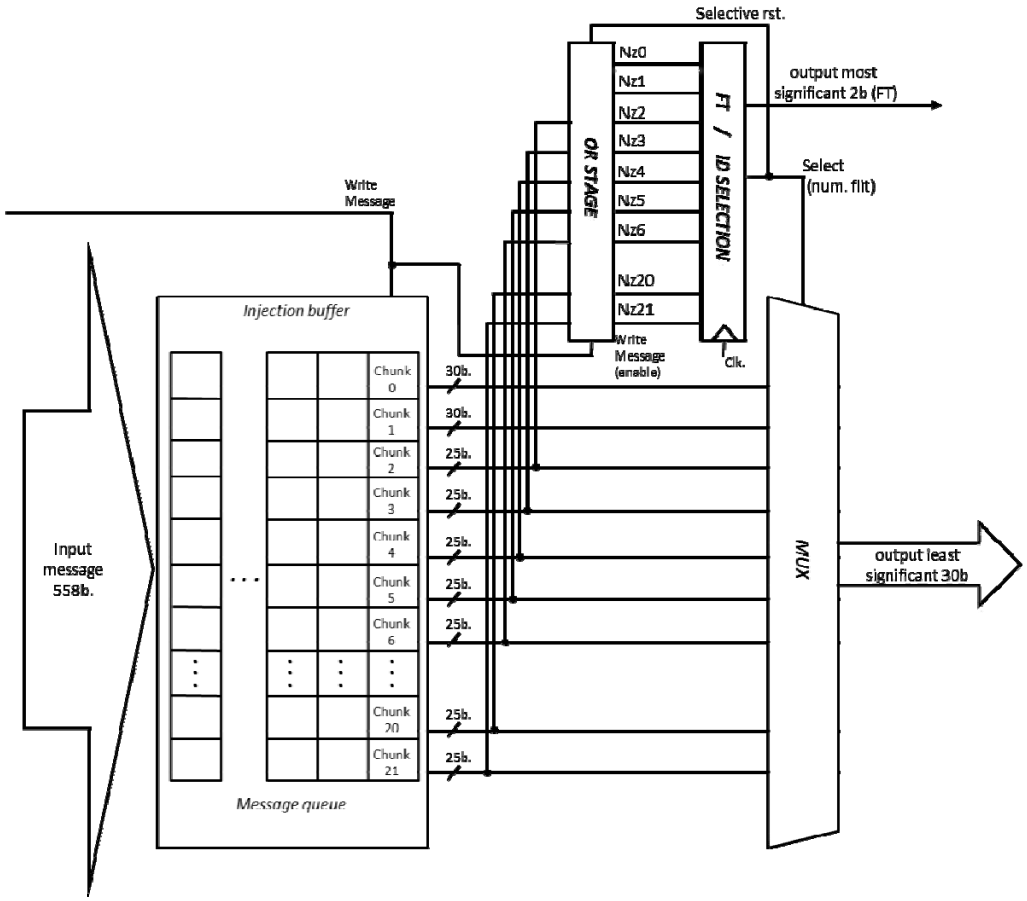


Figure 7. Injection NI with compression (general view).

All the bits of the message are exposed to the OR stage, shown in Figure 8. A 22-bit register stores the output of the OR gate for every chunk (notice that the first two chunks are not computed with an OR gate and are always set to one. The register (Nz) will keep account of the chunks that need to be injected (i.e. are non-zero). This register is written once per message injected and this occurs when the message is exposed at the head of the queue (in the first slot), thus can be driven by the write signal of the buffer.

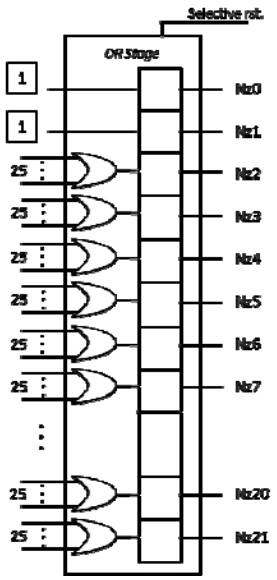


Figure 8. OR stage of the compression mechanism.

The Nz register output is now fed to the FT/ID selection logic, shown in Figure 9. This logic selects the next flit to inject. To do this, a priority encoder is used with all the Nx bits as input. The encoder selects the most significant bit set to one. This will be the first flit to inject. The Select signal is then used to drive the multiplexer at the injection port.

Once one flit is selected, the logic must be configured to select the next flit at the next injection cycle. To do this, the select signal is used in the OR stage in order to reset the appropriate Nz bit. Thus, at the next cycle a different chunk will be selected (if any).

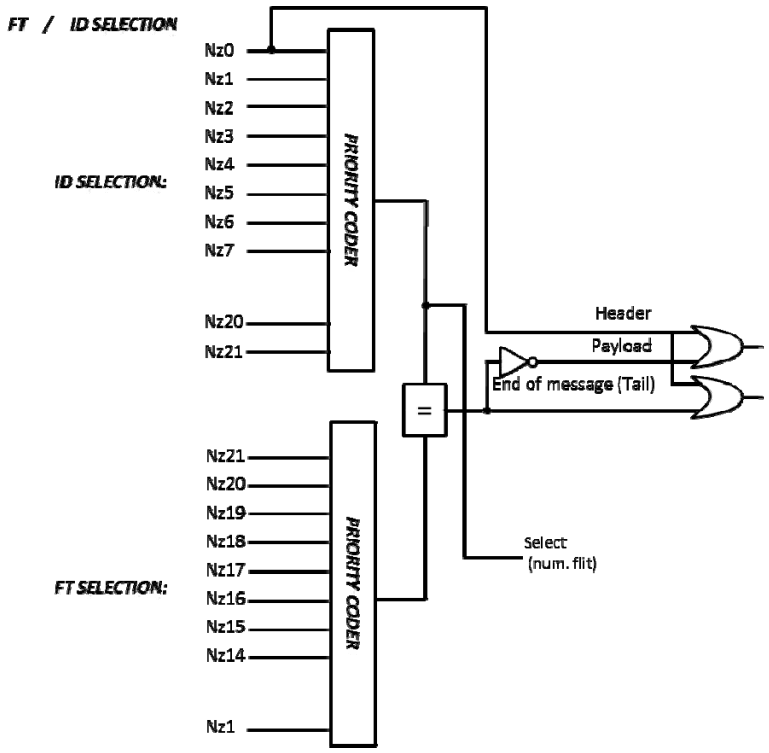


Figure 9. FT/ID selection stage of the compression mechanism.

In addition, the FT/ID selection logic computes the flit type and the flit number for every injected flit. The most significant bit of the flit must be set to one for header and payload flits, and set to zero only for a tail flit (notice that a flit can not be header and tail at the same time since compressed packets consist of two flits minimum). Similarly, the second bit of the FT field (bit 30) has to be set to 1 for a tail or header flit and to zero only for payload flits. See the table in Figure 5. Therefore, the complexity lies only in computing which of the chunks is the last one, since it needs to be set as the tail FT. This is achieved by an additional priority encoder but this time the Nz bits are inputed in reverse order. Hence the last chunk with the Nz signal set (meaning the chunk is non-zero chunk) is identified. With a comparator of the two outputs of the priority enconders, upon a match, the last flit being injected is identified as the tail flit.

The decompression strategy, shown in Figure 10, is much simpler since we do not need to calculate anything. When the first flit arrives, (with the FT field set to Head), the 30 remaining bits are placed at the beginning of the reception buffer. The second flit is treated the same way. If the second flit is not the tail of the message, all subsequent flits are placed according to the flit address (bits 29 to 25 of the flit). The Tail flit (FT equals Tail, i.e. 01) activates the *end of message* signal to indicate that the message is complete and the buffer slot is full.

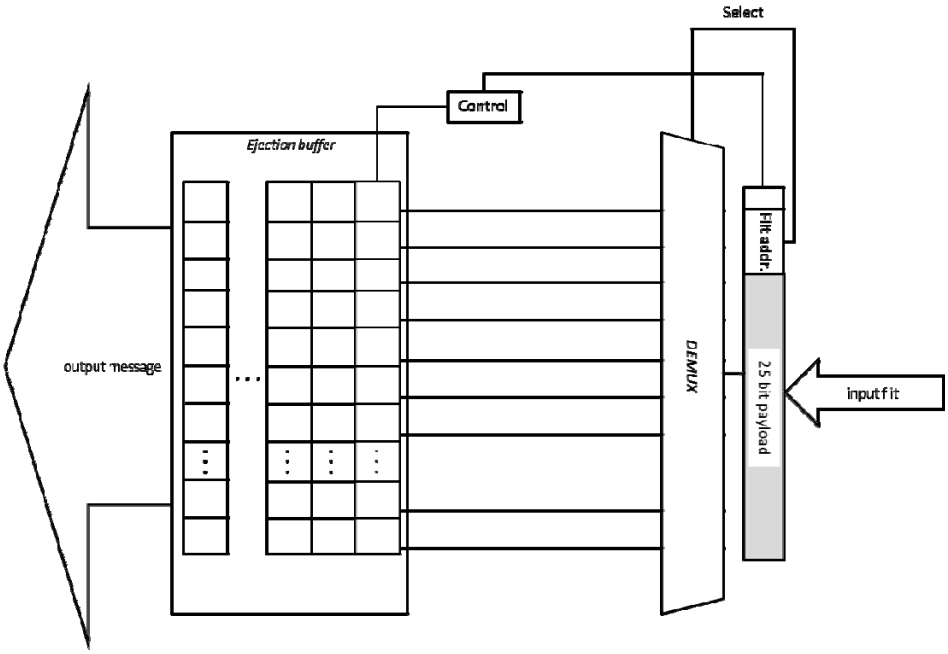


Figure 10. NI ejection with compression mechanism.

Notice that untransmitted chunks need to be coded at destination as zeros. Thus, the receiving slot is initialized every time to all zeros.

6. Evaluation Results

In this section we provide the results when analizing both the performance of the compression strategies, and the overheads of the implemented solutions. Therefore, we present the results into two separate parts. In the first part, we provide compression effectiveness by analyzing the percentage of traffic that is really compressed by the solution. In this part we use our simulation platform and inject the application memory access traces used in Deliverable 1.1. In the second part, we focus our attention on the implementation overheads. We show the typical results of area overheads, operating frequencies, and power consumption estimations of the implemented modules.

The combinations of both parts allows to decide the effectiveness of the compression solution. After the evaluation we provide a discussion section where we focus on the combination of both parts.

6.1. Compression Rate Achieved

One important aspect of the compression mechanism is its effectiveness in compressing traffic. For this, we have analyzed the compression rate achieved. We use the gMemNoCsim simulation platform (previously used in D1.1). With this simulator we model the scenario shown in Figure 11 (similar to the one used in D1.1). Four cores are attached to the same router. The cores are modeled by the private L1 cache. Also, an L2 cache is implemented and attached to the same router. The memory controller (MC) is attached to a neighbor router.

An invalidation directory-based coherence protocol with MOESI is implemented. Flit size is set to 32 bits, short messages sized to 64 bits (2 flits) and long messages sized to 512 bits (block size).

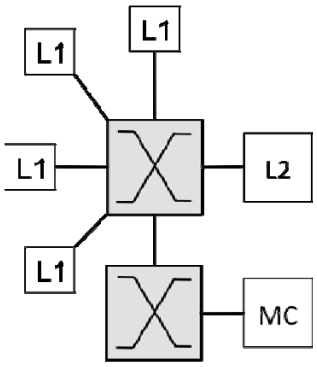


Figure 11. Simulated scenario.

Figure 12 shows the number of flits injected into the NoC when, both, no compression, and parallel compression is enabled. The figure shows the flits corresponding to the traffic between the L2 cache and the memory controller (MC). This traffic is the one that carries most of the memory blocks back and forth. As can be seen, the parallel compression strategy achieves significant traffic reduction in all the application cases analyzed. On average, traffic is reduced by a factor of 3.5 (250% reduction).

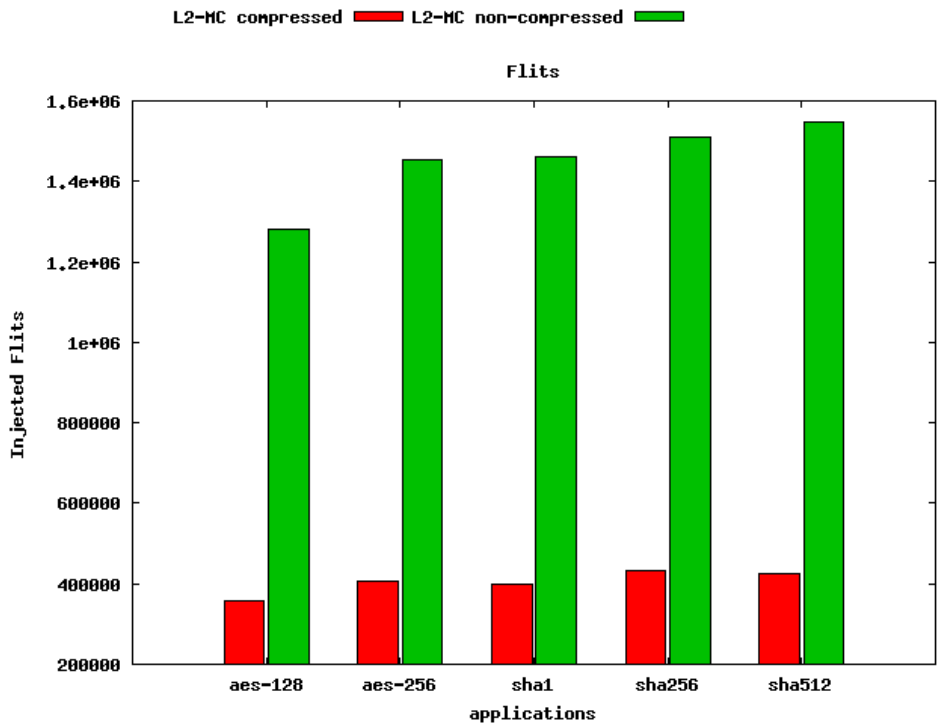


Figure 12. Number of flits injected into the NoC.

This reduction has a significant impact on the energy consumed by the NoC, since much less traffic is injected. In particular, dynamic power will be reduced. However, for the execution time of the applications, the impact is negligible as shown in Figure 13. Execution time is barely affected. The reasson for this low impact is that memory blocks accesses between the MC and the L2 bank are not in the critical path of the application. The L2 bank needs to send later the block to the L1 cache and the processor will be blocked only for the first word to arrive, not the whole block.

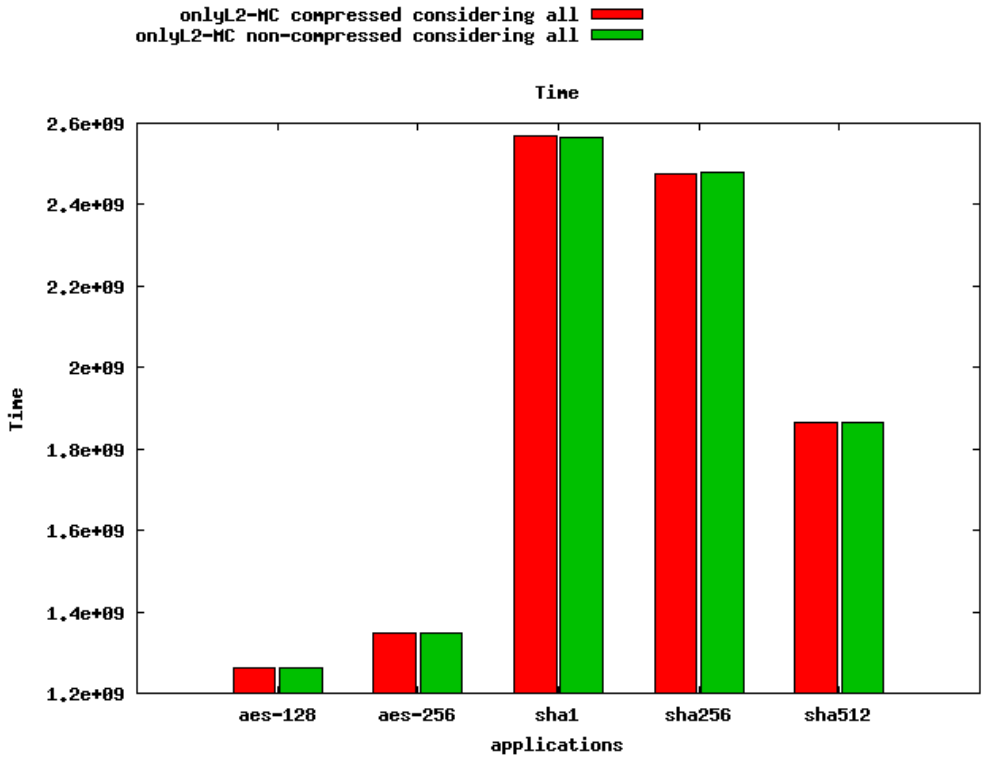


Figure 13. Execution time of application cases.

6.2. Implementation Overheads

The compression design has been synthesized using the 45nm technology open source Nangate [45nm. FreePDK] with Synopsys Design Compiler [15] . In Table 2 we show the area overheads of the injection part of the NI. The table shows the area for two models, one with no compression logic added (baseline) and one with the compression logic. We show results for different number of slots (ranging from one to eight). Notice that width of slot is set to 558 bits (a whole large message and its header per slot).

	With no compression		With compression		Compression Overhead (%)
Slots	Area	Normalized to 1 slot	Area	Normalized to 1 slot	
1	6587.09	1	7768.6	1	17.94
2	12156.7	1.84	13313.9	1.72	9.52
4	23444.7	3.56	24789.3	3.20	5.74
8	45839.2	6.96	47911.5	6.16	4.52

Table 2. Area overheads for the injection part of the NI. Different number of 558-bit slots.

As we increase the number of slots we achieve larger area overheads. Indeed, the buffer is the most demaning area resource. This trend is seen in both injection logic implementations. The most interesting result from the table is, however, the compression overhead in terms of area. As we can see, the logic for the compression is 18% for one slot. As we increase the buffer, the area overhead decreases down to 4.52% for 8 slots. Also notice that when considering the whole NI this overhead is even lower, (probably negligible).

For the ejection part of the NI, see Table 3, we see similar trends. As the number of receiving slots increases (each slot is 558 bits wide), the area overhead increases. When looking at the compression overheads, it ranges from no overhead for one slot to 14% for the 8 slot solution.

	With no compression		With compression		Compression Overhead (%)
Slots	Area	Normalized to 1 slot	Area	Normalized to 1 slot	
1	5710.4	1.00	5741.8	1.00	0
2	11168.3	1.96	11788.5	2.05	6
4	22334.2	3.91	25120.6	4.37	12
8	45823.3	8.02	52134.6	9.08	14

Table 3. Area overheads for the ejection part of the NI. Different number of 558-bit slots.

Now, let us turn our attention to power consumption. Power has been estimated by using Cadence Encounter [14] (both for Place&Route and measurements). Table 4 shows the power consumed by the injection part of the NI with no compression capabilities (baseline). It shows the internal, switching and leakage power consumption. As the number of slots increases, percentages of power consumption keep the same. However, total power consumption increases, reaching a factor of 1.39 for an 8-slot solution.

	Internal		Switching		Leakage		Total power	Normalized to 1 slot
Slots	Value	%	value	%	value	%		
1	2.324	74.87	0.7245	23.34	0.05549	1.788	3.104	1
2	5.736	74.18	1.887	24.4	0.1103	1.426	7.733	2.49
4	11.97	74.16	3.948	24.46	0.2233	1.383	16.14	5.20
8	25.35	73.7	8.605	25.02	0.4388	1.276	34.39	11.08

Table 4. Power consumption for the injection part of the NI (no compression). Different number of 558-bitb slots.

Table 5 shows the injection part of the NI but when adding the compression engine. The last column shows also the compression overhead. As we can see also, power consumption keeps similar to the one achieved when no compression is included. Even, power consumption is reduced for the 1 and 2 slot solutions. It has to be noted that the power consumption achieved by injecting less flits onto the network is not accounted in these tables.

	Internal		Switching		Leakage		Total power	Norm'd to 1 slot	Compression overhead (%)
Slots	Value	%	value	%	value	%			
1	3.869	70.93	1.508	27.65	0.07769	1.424	5.455	1	75.74
2	9.053	71.76	3.428	27.17	0.1352	1.072	12.62	2.31	63.19
4	18.52	72.01	6.951	27.02	0.2502	0.9725	25.72	4.72	59.36
8	36.78	72.31	13.61	26.76	0.4749	0.9337	50.86	9.32	47.89

Table 5. Power consumption for the injection part of the NI (compression). Different number of 558-bit slots.

Finally, for power consumption, Table 6 and Table 7 show power consumption for the ejection part of the NI, when no compression or compression is used, respectively. In this case, due to the simpler logic necessary, power overhead is well below 10%.

	Internal		Switching		Leakage		Total power	Normalized to 1 slot
Slots	Value	%	value	%	value	%		
1	4.77	76.55	1.413	22.67	0.0484	0.7767	6.231	1.00
2	9.974	74.89	3.238	24.31	0.1063	0.7983	13.32	2.14
4	19.03	76.34	5.673	22.76	0.2247	0.9013	24.93	4.00
8	39.69	70.59	16	28.47	0.531	0.9444	56.22	9.02

Table 6. Power consumption for the ejection part of the NI (no compression).

	Internal		Switching		Leakage		Total power	Norm'd to 1 slot	Compression overhead (%)
Slots	Value	%	value	%	value	%			
1	4.939	76.28	1.488	22.99	0.04768	0.7365	6.475	1.00	4
2	9.546	75.8	2.936	23.31	0.1122	0.8906	12.59	1.94	-6
4	20.01	71.39	7.779	27.75	0.2395	0.8546	28.03	4.33	12
8	39.14	68.58	17.45	30.57	0.4845	0.849	57.07	8.81	1

Table 7. Power consumption for the ejection part of the NI (compression).

In Table 8 and Table 9 we show the minimum clock period necessary for each implementation to work properly. These results were obtained by using PrimeTime [16] . Here we can see there is a significant impact on operating frequency. It is advisable to pipeline the solution with two stages, thus obtaining an adequate injection rate of 1 flit every 0.63ns.

	Without compression	With compression	Compression overhead (%)
Slots	Minimum Period	Minimum Period	
1	0.63	1.44	128.57
2	0.66	1.74	163.64
4	0.77	2.04	164.94
8	0.92	2.46	167.39

Table 8. Timing overhead for the injection part of the NI. Different number of 558-bit slots.

	Without compression	With compression	Compression overhead (%)
Slots	Minimum Period	Minimum Period	
1	0.50	0.55	128.57
2	0.65	0.67	163.64
4	0.68	0.78	164.94
8	0.77	0.83	167.39

Table 9. Timing overhead fot he ejection part of the NI. Different number of 558-bit slots.

6.3. Discussion

The previous results show the area, power, and latency overheads of different parts of the NIs, also for different configurations of number of slots. Now, we need to compound those results for a possible NI being used by the coherence protocol. This means, the NI will be made of different components, servicing different message clases of the coherence protocol. In detail, the protocol triggers short and long messages and only long messages, carrying memory blocks are subject to be compressed. This means, the NI needs to be built with different injector and with different ejector configurations.

Table 10 shows the area overheads and power consumption of the NI with no compression mechanism included but for slot sizes of 32 bits. The table shows the results for both components, injection and ejection. These components will be used to handle short control messages of the protocol.

Slots	Area	Normalized to 1 slot	Internal		Switching		Leakage		Total power	Normalized to 1 slot
1	314.32	1	0.249	80.79%	0.057	18.42%	0.002	0.79%	0.309	1
2	655.20	2.08	0.628	76.1%	0.192	23.19%	0.006	0.72%	0.826	2.67
4	1321.25	4.20	1.366	76.27%	0.412	23.02%	0.013	0.71%	1.791	5.80
8	2643.48	8.41	2.778	77.4%	0.785	21.87%	0.026	0.72%	3.589	11.61

Table 10. Area and power overheads for both injection and ejection parts of the NI. Different number of 32-bit slots.

With all these results, now we can build the overheads of a final NI with the following characteristics:

- Injection of long messages with compression facility enabled
- Ejection of long messages with compression facility enabled
- Injection of short messages with no compression facility
- Ejection of short messages with no compression facility

For the number of slots at each component we need to consider the flit size. Indeed, wide slots will need different network cycles to inject the whole message, while thin slots will need fewer cycles. Therefore, is reasonable to use few slots for long messages while more slots for short messages. We select one slot for large messages and two slots for short messages.

With all these considerations, we derive our final results by adding the different overheads previously presented. The final NI will consist of two injection buffers, one for long compressible messages and one for short non-compressible ones; and the two corresponding ejection buffers. Table 11 shows the results achieved for each of these buffers as well as the global results.

NI component	Characteristics	Area	Overhead (%)	Power	Overhead (%)
Injection long messages	1 slot; with compression	7768.6	17.94	5.455	75.74

Ejection long messages	1 slot; with decompression	5741.8	0	6.475	3.91
Injection short messages	2 slots; no compression	655.20	0	0.826	0
Ejection short messages	2 slots; no decompression	655.20	0	0.826	0
Total		14820,8	8.91%	13.582	24%

Table 11. Final area and power consumption overhead for a NI with two injectors and two ejectors (one for long messages with compression capabilities and one for short messages with no compression capabilities).

As we can see, the overhead is significant but the potential reduction of the number of flits injected in the network (Figure 12. Number of flits injected into the NoC.) suggests that the global power consumption will be as well reduced. To corroborate this, we have derived the approximate power consumption of the network per flit for both the presented baseline NI implementation and our compression-enhanced implementation. In the graph in Figure 14 we can see the difference in power consumption for different injection rates. In every graph, the X-axis corresponds to the compression rate [0:1] and the Y-axis corresponds to the difference in power consumption between our solution and the baseline (we have taken as a baseline a NI with one injector and one ejector of 2 32-bit slots each). We have only taken into account the long message network, since the other network is exactly the same in both cases and would be nullified.

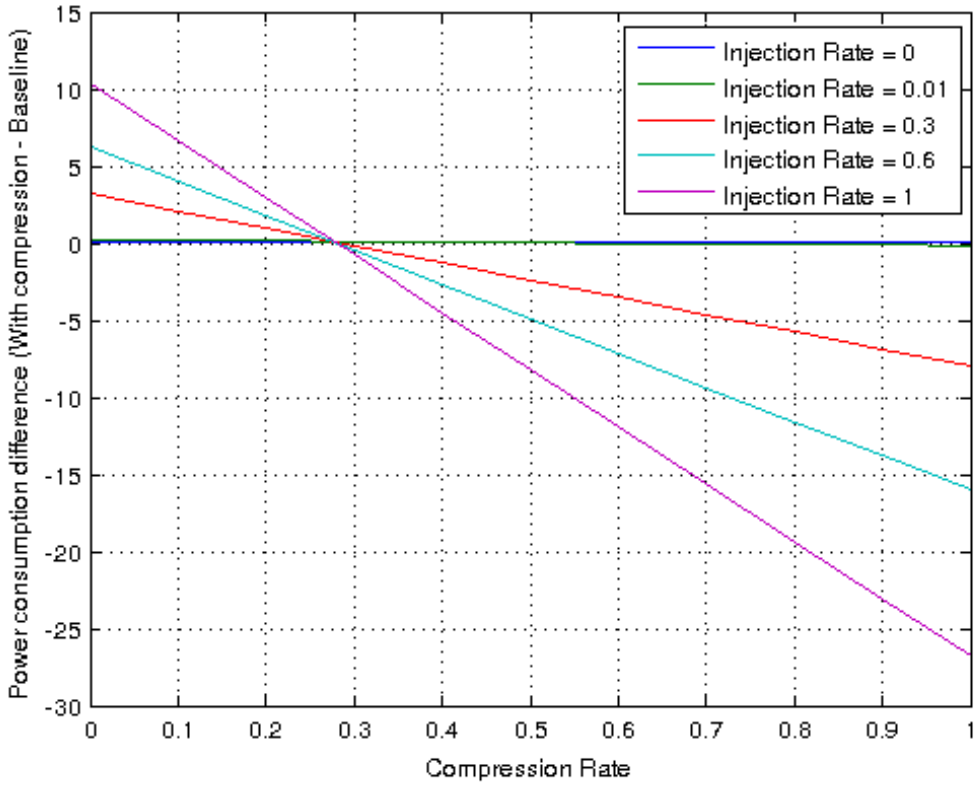


Figure 14. Study of the difference in power consumption of our solution and the baseline depending on compression-rate for different injection rates.

In the previous graph we can see that depending on long message injection rate, we can save different amounts of power, but in any case the compression rate needed is 0.3 (we need to

send at least 30% less flits in order to get some power benefit). Since our compression rate is much beyond the needed 30%, it is assured that with data that follows the pattern of the applications studied in D1.1 some power would be saved.

Since the power consumed by the NI with compression facilities is higher than that consumed by the baseline NI, it seems interesting to provide some heuristic in order to turn the mechanism on and off, depending on its relevance. Since messages between L2 and MC are only a small fraction of the total messages on the net, and in every case a previous control message (from the coherency protocol) must have been sent, we can turn on and off the mechanism according to these short control messages (that travel across the other subnetwork). This possibility needs to be further studied, but it would be bounded between a maximum and a minimum value. The maximum value would be the already presented in Figure 14. The minimum value is presented in Figure 15. Study of the difference in power consumption of our solution and the baseline depending on compression-rate for different injection rates with no static power., where we show the result with only de dynamic power component.

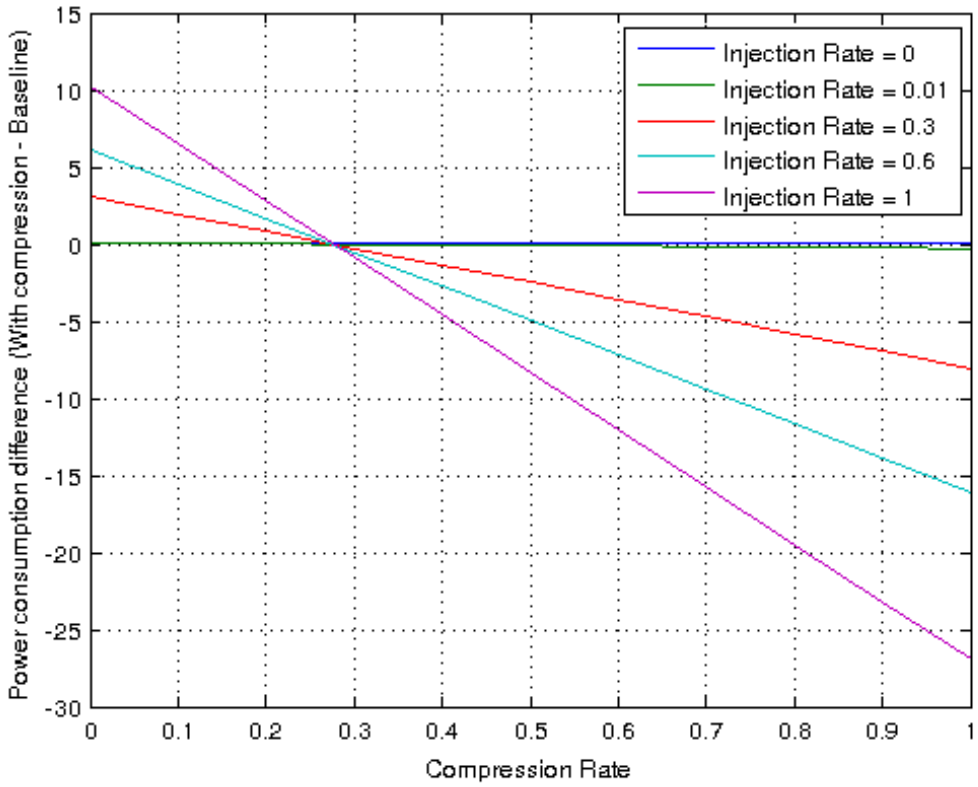


Figure 15. Study of the difference in power consumption of our solution and the baseline depending on compression-rate for different injection rates with no static power.

As we can see from the graphs, the difference is not very high because the static power consumed by the subnetwork is negligible, (it is mostly represented by the leakage power of its components).

7. Conclusions

In this deliverable we have provided the results for Task 4.1 in the context of the vIrtical project. Our goal was to design and analyze compression strategies at the NoC level, saving communication costs, by reducing the number of transmitted flits. The provided mechanism relies on the abundance of memory data blocks filled with zeros (as seen in D.1.1), thus easily compressible by using a detection strategy

Our compression mechanism avoids sending flits without information (i.e. not sending flits that only consist of zeros). This technique is both easily applied and highly efficient. This compression scheme is only applied to long messages (a full memory block, 512bits), not to short messages (of only 2 flits).

The results displayed in the previous section show the effectiveness of the compression and decompression mechanisms and the low overhead they introduce. The percentage of traffic reduced by the compression strategy (a factor of 3.5) justifies the overheads in resources for compression and decompression. The area overhead for the compression and decompression mechanisms required for a system with coherence support is 8.91% whereas the added power consumption is 24%.

References

[1] A. Chandra and K. Chakrabarty, ``Frequency-Directed Run-Length (FDR) Codes with Application to System-on-a-Chip Test Data Compression," VLSI Test Symposium, 19th IEEE Proceedings on. VTS 2001

[2] M. Tehranipoor, M. Nourani and K. Chakrabarty, ``Nine-Coded Compression Technique for Testing Embedded Cores in SoCs," Very Large Scale Integration (VLSI) Systems, IEEE Transactions 2005

[3] L. Ciganda, F. Abate, P. Bernardi, M. Bruno and M. Sonza Reorda, An enhanced FPGA-based Low-Cost Tester Platform exploiting effective Test Data Compression for SoCs, Design and Diagnostics of Electronic Circuits & Systems, 2009.

[4] J. Dalmasso, E. Cota, M. Flottes and B. Rouzeyre, Improving the Test of NoC-based SoCs with Help of Compression Schemes, in IEEE Computer Society Annual Symposium on VLSI 2009

[5] V. Froese, R. Ibers, and S. Hellebrand, ``Reusing NoC-Infrastructure for Test Data Compression," in 28th IEEE VLSI Test Symposium 2010

[6] S. Chaki, C. Giri and H. Rahaman, ``Binary Difference Based Test Data Compression for NoC Based SoCs," in IEEE Computer Society Annual Symposium on VLSI 2012

[7] M. Palesi, S. Kumar, and R. Holsmark, ``A Method for Router Table Compression for Application Specific Routing in Mesh Topology NoC Architectures," in S. Vassiliadis et al. (Eds.): SAMOS 2006, LNCS 4017, pp. 373–384, 2006

[8] Simon Ogg, Bashir Al-Hashimi, ``Improved Data Compression for Serial Interconnected Network on Chip through Unused Significant Bit Removal," in Proceedings of the 19th International Conference on VLSI Design 2006

[9] L. Vittanala and M. Chaudhuri, ``Integrating Memory Compression and Decompression with Coherence Protocols," in Distributed Shared Memory Multiprocessors in International Conference of Parallel Processing 2007

[10] R. Das, A. Mishra, C. Nicopoulos, D. Park, V. Narayanan, R. Iyer, M. Yousif and C. Das, ``Performance and Power Optimization through Data Compression in Network-on-Chip Architectures," in High Performance Computer Architecture, IEEE 14th International Symposium 2008

[11] Y. Jin, K. Yum and E. Kim, ``Adaptive Data Compression for High-Performance Low-Power On-Chip Networks," in Microarchitecture, 41st IEEE/ACM International Symposium 2008

[12] P. Zhou, B. Zhao, YuDu, YiXu, Y. Zhang, J. Yang and L. Zhao ``Frequent Value Compression in Packet-based NoC Architectures," in Design Automation Conference. Asia and South Pacific 2009

[13] B. An, M. Lee, K. Yum, and E. Kim, ``Efficient Data Packet Compression for Cache Coherent Multiprocessor Systems," in Data Compression Conference 2012

[14] Encounter User Guide, Cadence Design Systems, Inc. (http://www.cadence.com/), 2011

[15] Design Compiler User Guide, Synopsys Inc. (http://www.synopsys.com/) , 2011

[16] PrimeTime User Guide, Synopsys Inc.(http://www.synopsys.com/), 2011