Address Translation
&
IRAM Cache

Members:
ori.yeffet@mail.huji.ac.il – Ori Yefet
omrids@gmail.com – Omri Dassa
Group 327

Advisor:
nadav.rotter@intel.com – Nadav Rotter

Introduction

NIC – Hardware component used by computers/servers for communication on a network system.

This project address 2 main components of an Ethernet NIC (Network Interface controller):

- A typical NIC has several CPU controllers which are processing packets of data and rout them according to standard protocols. Those controllers follow a set of commands which reside in the IRAM (Instruction RAM). In order to change or add functionality to the controller's by demand, one can load a different/upgraded set of commands to the IRAM allowing more/other features.
- On top of that, a basic requirement of today NICs is to support several software entities using shared resource. The NIC contains a single public memory space that every software entity[1] need to access (by memory address) while considering itself the sole owner of the memory with no dependency of other software entities.

The 2 problem that we will face in our project are:

- The IRAM is fixed and confined, thus can only support limited number of features. We would like to enable expansion of the IRAM address space without increasing its size, and also do it dynamically so costumers could do so whenever they like.
- By using a system that support several software entities with a public memory there is a need for efficient and fast address translation and protection[2] unit which will translate virtual addresses into unique physical addresses.

Our solutions to both of the problems must hold the followings:

- Must not exceed a bounded area on the NIC.
- Must meet performance specifications
- Must be Versatile and parametric in order to be reused in future projects.

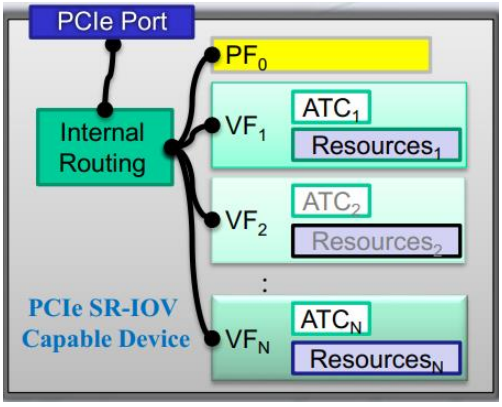
Both of the problems are known issues in the NIC industry. Intel is the leader of NIC development industry worldwide and as such, it is the required to unlock new designs for NIC components that must meet higher performances each passing year. As a result, some components must be redesigned to meet with the new requirements and also some new unexpected problems surface. Address Translation is an example to a problem that must be redesigned and IRAM memory space is to a new problem that must be addressed.

[1] Software entity – is a software process that controls a CPU controller which is hardware device. Every Software entity that tries to communicate with a CPU controller doesn't know that there are another Software entity's that also communicate with a CPU controller.

[2] Address translation unit translate a request from software entity into a hardware shared resource and prevent access of a software entity to another software entity memory space for security reasons.

Methods

Before we approach our current MVP we would like to discuss the strategy we took for designing our solutions. In order to understand the problems better, we investigated older solutions and other methodologies. We learned that a worthy solution for expanding the IRAM address space is a caching mechanism and that both requests from VFs[1] and PFs[2] need to get translation. We also learned that protection for those entities memory space was not covered by previous projects.



The second stage of our investigation process was to look at the adjacent blocks. The IRAM cache interface with the SPI[3] controller which is responsible for uploading the IRAM's data from the NVM[4] and the CPU controller. The Address translation block on the other hand connects with several Hosts[5] that send translation requests and an Interconnect block which receives the translated requests.

The next step was to document our design thoroughly according to the architectural specifications of the group's micro-architects. This document (MAS-micro-architectural spec) includes detailed overview of the block such as block key features, block diagram, data and control flows, clock and reset domains, performance calculation and wave diagrams.

After revising the adjacent blocks to match our planned design we could approach the final step towards the MVP which was to code the solutions according to the documented designs (MAS).

[1] VF – Virtual Function is a software entity that support single root virtualization as defined in PCIe Spec. Share recourses with the PF that is associated with.

[2] PF – Physical Function is a software entity that support single root virtualization as defined in PCIe Spec. It is used to manage number of VFs.

[3] SPI – Serial Peripheral Interface is an interface specification used for short distance communication.

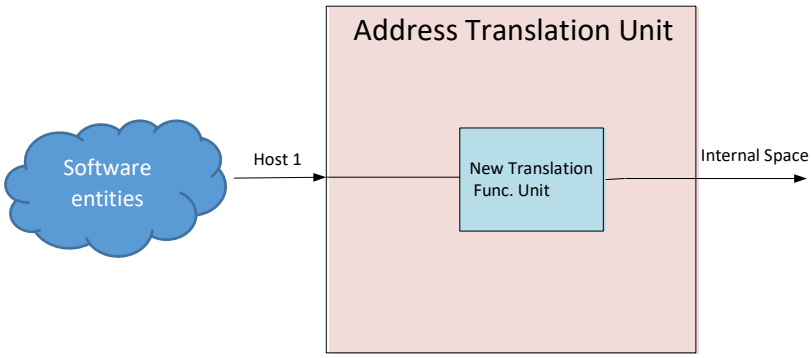
[4] NVM – Non Volatile Memory is a type of computer memory that can retrieve stored information even after having been power cycled.

[5] Host – software entity that control number of PFs and therefor a greater number of VFs.

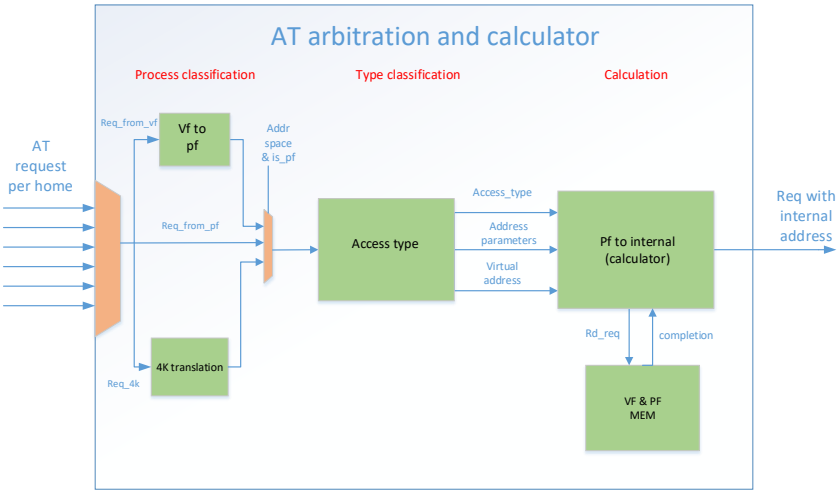
Address Translation - MVP

When a single Host uses the shared resources it can consider himself the sole owner of those resources, as it is indeed the only user of the resources, thus an injective (according to the number of software entities) function can be used in order to translate virtual addresses (from the software entity) to physical addresses (resource).

As the solution for the Address Translation consist of both translation function and arbitration between several Hosts, we first solved the new translation function which is able to translate from several Hosts to a single shared resource but we are testing it only with a single Host.



This is our MVP for the Address Translation unit. The New Translation function features are detailed in the following scheme:



The translation function works as follows:

1. Request arrive to the unit interface from a Host (left side of the scheme) and is being arbitrated- which means that only one request will be processed in each cycle. *the arbiter feature in our MVP is idle and Host#0 always win arbitration.
2. Process Classification is the first stage of translation in which a request is pre-translated according to its origin software entity (VF/4K[1]/PF). The purpose is to unify all software entities to PF characteristics.

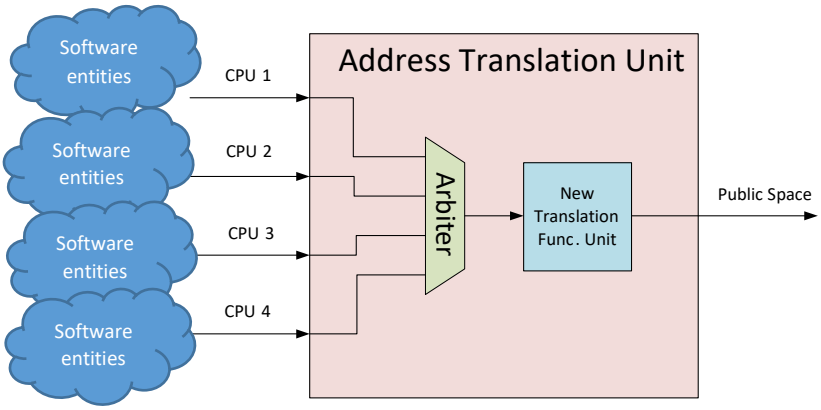
[1] 4K – Special type of VF which has a certain address classification.

3. Type Classification is the second stage in which the type of a request is being determined (access type) according to its address. Some parameters regarding the request are also being calculated.
4. Calculation is the last stage of the translation function. In this stage all of the parameters that were collected are being used for calculating the internal address. Among the parameters are the access type and the virtual address.
5. The request is sent out to the Interconnect block with its internal address.

Regarding our progress, we received a new architectural demand which states that the Address Translation unit should also cover Address protection as mentioned. We decided that the schedule does not allow us to push it for our MVP and therefore this feature will be a target for P1. All other features are implemented and are being verified. Debug deadline is this end of WW09.

Address Translation – P1

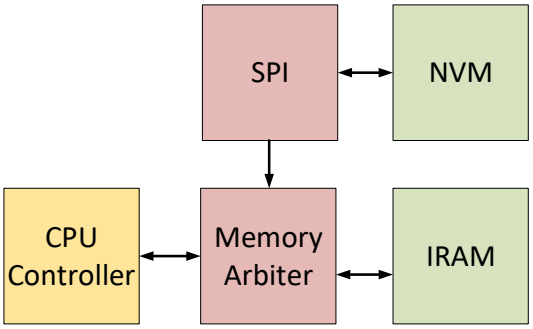
In our last stage of the solution we will add a fair arbitration between the different CPU controllers in the Address Translation unit and check the translation function for requests from several CPUs. This complete solution is offering a way for number of compute units to use one shared memory.



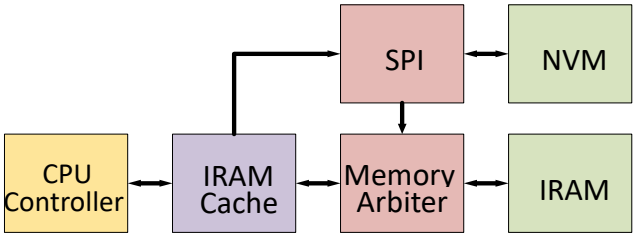
In addition as mentioned, we shall design and implement the address protection feature.

IRAM Cache - MVP

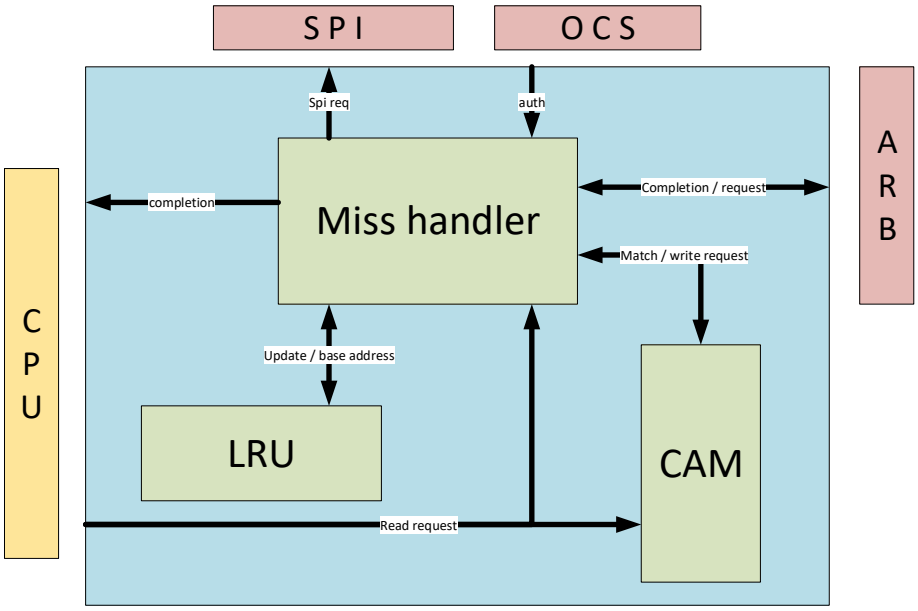
As we know a RAM type memory does not keep its values when the power is off, so in order to load instruction to the IRAM a fetching mechanism must be applied on boot sequence from a NVM which is slower but larger memory. After the boot sequence CPU controller's requests are answered by a Memory Arbiter that fetches data from the IRAM.



In order to expand the IRAM Memory address space we used a caching mechanism and made the IRAM entirely cached. Meaning that each request for instruction from the CPU controller has first been compared in order to identify whether the data reside in the IRAM or not. This solution will let us use both the IRAM and NVM memory space so we can store more instruction for more features, as well as benefiting from the IRAM speed for the most used features.



This is our MVP for the IRAM Cache unit. The caching mechanism features are detailed in the following scheme:



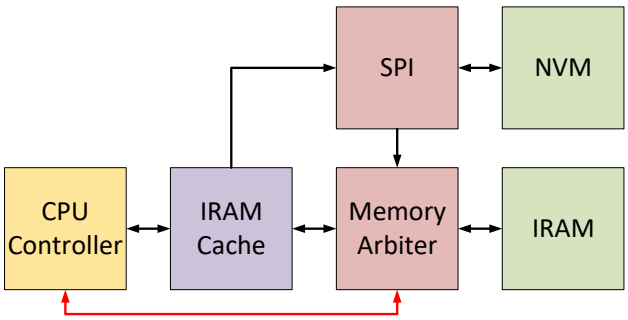
The cache mechanism works as follows:

1. The address of a read request from the CPU is being compared in the CAM table.
2. In case the data reside in the IRAM (i.e HIT), a complete request with the matching address is sent to the IRAM by the MISS HANDLER and the instruction data is rolled back to the CPU.
3. In case the data doesn't reside in the IRAM (i.e MISS), a fetching request is sent to the NVM through the SPI interface by the MISS HANDLER to write the requested instruction data into the IRAM.
4. An authentication indication must be granted from OCS interface to the cache before the fetching a request is completed so only authenticated data is written from the NVM to the IRAM.
5. The LRU sub-block is responsible for managing the address eviction policy. It is implemented as a 2-way hardware link-list and is being updated every cycle.

Regarding our status, we had a very good progress in the last month and we managed to complete the full mapping implementation as well as our P1 target for partial mapping. As mentioned in previous reports, one of the Cache architectural demand was to enable users to lock a part of the IRAM memory space and by doing so ensuring the instruction data of this locked memory will not be evacuated from the IRAM and be replaced. The way we did that is by pinning the desired addresses in the LRU unit so it will not participate in the link-list. Nevertheless, a new architectural demand surfaced to enable a partial mapping of the IRAM in the way of a threshold address that should separate the IRAM space to a cached space and an uncached space. We scheduled the implementation of this feature to P1.

IRAM Cache – P1

As mentioned the partial mapping will also be allowed by a threshold address. A threshold address that is given once the device leaves reset stage and each data segment that reside below the threshold address in the IRAM will be cached and accessed only by the caching mechanism. If the data segment reside above the threshold address should be accessed directly.



Address Translation & IRAM Cache – P2

Both units should be verified by Formal verification. A Formal is a verification aspect that enables a designer to mathematically prove desired properties about his design. By this way one can check for unwanted flows and corner cases. The way to do so is by writing assumptions and assertions which are rules that must hold for the design. The process requires much thinking about the design corner cases and the rules that the design must hold. Debugging errors of a Formal verification is a much more difficult task than of a standard verification.

Schedule

Omri Dassa		Ori Yefet		Date
Priority	**Task**	**Priority**	**Task**	
P0	Debug	P0	Debug	February
P1	Coding Arbitration of several CPUs + Address Protection	P1	Coding Partial-Mapping by threshold address	March
P1	Address Protection + Debug + Synthesis	P1	Debug + Synthesis	April
P2	Formal Verification- writing assumptions and assertions	P2	Formal Verification- writing assumptions and assertions	May
P2	Formal Verification + Debug	P2	Formal Verification + Debug	June

Evaluation and Verification

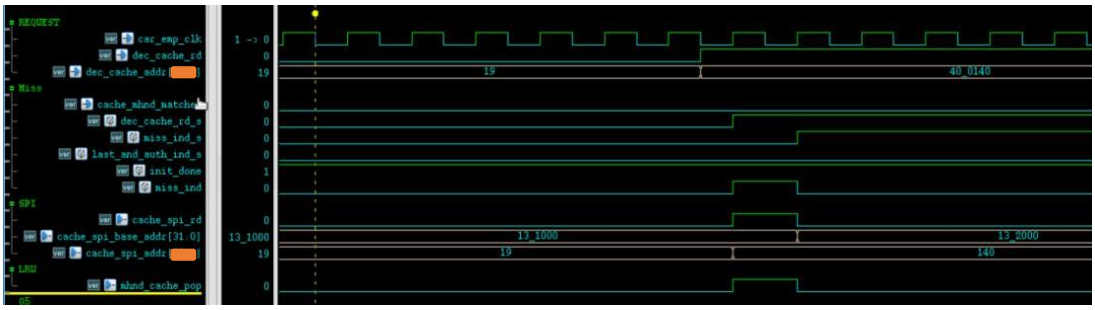
The three main issues that must be addressed in order to evaluate our project are:

- Timing – both of the components described above must meet timing constraints driven from the clock frequency. Meaning the delay time of the logic in use must not exceed the clock cycle. Also, because both of the components intervene in an already existing flows our solution will be compared to the older solutions.
- Size – the components will be measured in terms of Flip-Flops, latches and combinational cells and must not exceed the size defined by the NIC architects.
- Correctness – the components obviously must do what they were designed to do.

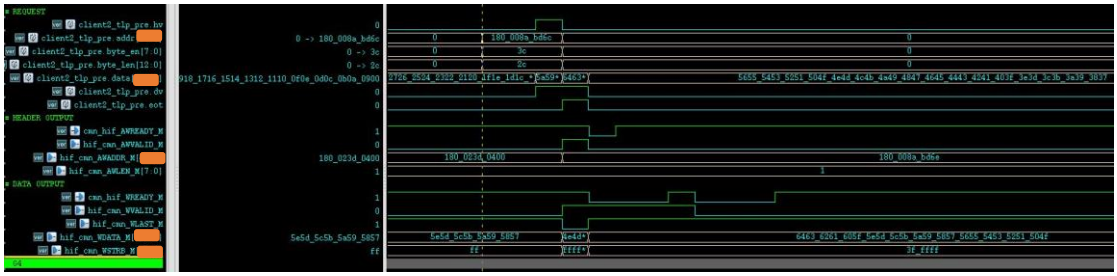
In order to evaluate those parameters we are using 2 tools. The first is to build a test-bench for the components which will check their correctness. The second is executing a synthetizing tool which will count and measure both size and timing. At the present stage of our project we are checking the correctness of our design with various test-benches that are

meant to verify several permutations of inputs. The inputs are uploaded to a scoreboard and are matched with the expected outputs.

This is an example to a wave diagram of the IRAM Cache unit. In the diagram we can look at a successful “MISS”. Upon that miss, the LRU unit is being updated and also a fetching request is sent to the SPI controller.



In the next diagram there is an example to a wave diagram of Address Translation Unit. A translation of a request is being processed. Note that the input address of the request ended with ‘6c’ and was translated into ‘6e’. One can also notice that the communication protocols has changed from the input interface to the output interface as the interconnect unit communicates with a different protocol.



The size and timing measurements are only being calculated when the design is fully verified so we postpone these evaluations to P2.