DESIGN AND IMPLEMENTATION OF MOBILITYFIRST ROUTER ON THE NETFPGA PLATFORM

BY NISWARTH MUDALIAR

A thesis submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Yanyong Zhang

and approved by

_____

_____

_____

New Brunswick, New Jersey

May, 2013

ABSTRACT OF THE THESIS

Design and Implementation of MobilityFirst Router on the NetFPGA Platform

by Niswarth Mudaliar

Thesis Director: Yanyong Zhang

We present the hardware prototype design and evaluation of routers in *MobilityFirst*, a Future Internet Architecture. We chose NetFPGA 1G platform to implement the router. The main task of the router is to perform lookup on MobilityFirst packets which has a two-level addressing scheme (GUID and NA), each level with flat address space. We have designed the MF router to get maximum performance for flow based routing. The forwarding information base of the router has been spread out in Binary CAMs, Block RAMs and external SRAMs which can be updated online. We have achieved a throughput as high as 982Mbps per port (in Gigabit Ethernet). We have provided a way for the router to send updates for missed addresses which allows the host controller to keep a close watch on the network traffic and update the forwarding information base, online. We allow host controller to cache GNRS lookups in the MF router to allow automatic flow based NA binding to be done at line-speeds. Our router buffers the packets if they fail lookups until the forwarding information base is updated and the lookup succeeds. This prototype enables us to analyze the performance bottleneck of MobilityFirst Architecture so that we could continue to improve the architecture and evaluate the performance simultaneously.

Acknowledgements

I would like to express my gratitude towards Prof. Zhang for giving me this opportunity to be a part of this research. I sincerely thank her for all the guidance and encouragement she has provided me. I would like to convey special thanks to Ivan Seskar, Prof. Richard Martin, Prof. Raychaudhuri and Dr. Kiran Nagaraja for their thoughtful insights and feedbacks. I am thankful to WINLAB Rutgers for allowing me to use state of the art equipment and facilities which enabled me to complete my work successfully. I would also like to thank Dr. Richard Howard for his encouragement and support which helped me in managing my Thesis work.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The explosive growth in the mobile and handheld devices and ever increasing need for higher data rates puts tremendous burden on network designers to meet the ends. A lot of research has been done globally to improve upon the existing infrastructure. One of the directions of research is to improve the existing internet architecture, which originally had been built to support and connect immobile computers, and make it better suited to mobiles devices. Mobility on IPv4 and IPv6 had been proposed more than a decade ago using Mobile IP to support mobile devices through tunneling and/or translation [1] but the issues with tunneling overhead and security exist due to the underlying fact that IP was designed for immobile end devices.

MobilityFirst has been proposed by WinLab Rutgers as a Future Internet Architecture and the research activity is supported by the National Science Foundation [2]. As stated by Dipankar Raychaudhuri *et al.* [3], the requirements for a robust Future Internet Architecture ranges from support for seamless user and device mobility with tolerance to bandwidth variation and disconnection, to supporting high throughput and having spectral efficiency (in wireless edge networks) while providing strong security.

MobilityFirst addresses the above requirements by having clean separation between device/user ID and network location, unlike the current internet architecture which works on IP addresses which is used to resolve both. Moreover, MobilityFirst uses a flat address space for both addresses as explained in section 2.1 , which poses a challenge on designing a fast-lookup hardware router which can work at line speeds while routing MobilityFirst packets. Fortunately, VLSI technologies have advanced so far that we

can have complex lookup mechanisms which can work at reasonably high speeds at low cost and power. This thesis presents a prototype design for the MobilityFirst Router on the NetFPGA platform.

1.1 Original Contribution

Our original contribution is the design of MobilityFirst Harware Router on NetFPGA platform. We have developed some basic lookup schemes and the RTL for the router. Our base design works on two Identifiers symultaneously to resolve the output port for a MobilityFirst Packet. It is capable of auto-binding of NA to a GUID from within the router which increases the speeds of initial or late binding. We evaluated the performance of exact-match lookup and predictive lookup(bloom filters) schemes. Our design has pipelined lookups which help us reach near line-speed performance. Our desing exploits the different on-board memories such as SRAM and DDR2 DRAM, and different on-chip memory such as CAMs, BlockRAMs and registers to hold routing table information and buffer packets. Our design allows software interaction through IOCTL calls from host controller over PCI bus to read and write into status and command registers implemented on FPGA. Our design also opens up a new direction for research for MobilityFirst Flow based custom harware router designing.

1.2 Results

We were able to achieve near line-speed throughput of 982Mbps per port giving us a total througput of 3.927Gbps on 4 ports in single direction (and $3.927 * 2 = 7.854$Gbps bidirectionally), for a packet length of 1500Bytes when the packets do not belong to the same flow. We are able to achieve 982Mbps per port for any packet length when the packets belong to the same flow.

We were able to achieve near line-speed throught of 974Mbps for bloom filter implementation for packet lengths as small as 174Bytes, and may or may not belong to the flow. Our Bloom Filter Implementation generates false positive ratios close to theoretically estimated values.

1.3 Outline of the Thesis

Chapter 2 describes the privious work done in this field. Chapter 3 discusses the design of the MobilityFirst Router and the different ideas being put to test. Chapter 4 does deeper into the design details and explains implementation of the Router on NetFPGA 1G platform. Chapter 5 shows the output of various experiments done to validate the hypothesis. Chapter 6 concludes this thesis and presents scope for future work in this direction.

Chapter 2

Prior Work

2.1 MobilityFirst Architecture

MobilityFirst Architecture [4], is design to support secured Mobility of devices while maintaining reasonably high through-puts. The design is centered around the concept of decoupling Device Identifiers (GUID) with Network Address (NA). It presents ideas for a central Global Name Resolution Service (GNRS) to combine the two level address by a process of Binding which can happen once at the source and on-the-fly while the data packets are being routed. In addition, it proposes to use strong Public Keys to generate GUID and NA to maximize the security.

2.2 Content Addressable Memories (CAM)

Content Addressable Memory (CAM) [5] is a type of Static Random Access Memory which can search for content in the memory parallely in all locations and return the address (or addresses) of the content. Ternary CAMs or TCAMs [6] are extension to CAMs where Ternary mode allows to mask certain bits (don't care bits) of the data while comparison. This Ternary mode has been heavily exploited for Longest Prefix Matching in IP prefix lookups [7]. Research in designing highly specialized CAM structures for IP [8] shows that it might be beneficial to do custom memory design to increase the lookup speeds. However to prototype a router for a new Internet Architecture, designing custom memory may not be an economical and necessary option. Therefore, research [9] is being done to use off-the-self CAMs in conjunction with TCAMs to allow fast lookups of flat lables(IDs). In addition to this, the router architecture for modern Internet Architecture, is evolving, and some older ideas are being *rehashed*. Route

Caching [10] which was considered to be not very effective, is now being revisited for its potential to support flow based routing in a very large address space.

2.3 Bloom Filters

Bloom filters have been used in many networking and database applications. It is used for set association. Michael Mitzenmacher *et al.* [11] have derived the optimal value of hash functions to get a reasonable low false positive ratio, assuming that the hash functions are well crafted for the data-set being used to build the Bloom Filter. The bloom filter data structures can be easily stored in memory in a way which allows very fast query resolution [12]. Although the drawback of false positives is an issue, the advantages of bloom filters certainly place them amongst the best predictive lookup schemes.

2.4 NetFPGA 1G Platform

NetFPGA 1G [13] is an open platform designed by Stanford University for educational research in networking and router design. The 1G stands for Gigabit Ethernet. It gives us the freedom to design custom hardware for dedicated routers such as MobilityFirst router. NetFPGA Group also provides reference designs for both RTL and Host application for a router. NetFPGA 1G platform uses Virtex2Pro XC2VP50 [14] FPGA from Xilinx, which has a good amount of logic cells and Block RAMs, and is fast enough to prototype a Gigabit Ethernet router.

Chapter 3

Design of MobilityFirst Router

The design phase started off on a top down approach, where the requirements are gathered first and then design of lower layers is successively done. However, as the time progressed and as the practical limitations and constraints manifested themselves, the design was modified to accomplish the task of building the MobilityFirst router prototype.

Some assumptions were made to simplify the requirements. They were:

1. The MobilityFirst packets would have a GUID and a NA which together would let the packet reach the destination.

2. The hardware limitations and timing budget allowed us to use a 32 bit GUID and a 32 bit NA. Although, with a more advanced platform, the design could support 160 bit GUIDs as well.

3. The NA is a globally unique network address whereas the GUID is a globally unique destination identifier.

4. Both NA and GUID will be generated using cryptographic techniques and will be spread out in a uniformly random order within the space they may span.

5. The NA field should be present and can have any non-zero arbitary value for a valid NA. A value '0' is considered as NULL address and would indicate absence of a valid NA.

6. The MobilityFirst router would handle packets for both inter-domain and intra-domain routing, where NA will be used to distinguish between the two.

The basic components of a router are the input ports, output ports, a switching fabric and a routing processor [4], and the performance bottleneck of a router lies in the forwarding table lookup. Traditional IP based routers use longest prefix matching (LPM) for which TCAM (Ternary Content Addressable Memory) is best utilized. Longest prefix matching works best of hierarchical network address. With the large number of networks, a router may have to accommodate a large set IP prefixes. CAMs can allow for storing 100,000 entries but are expensive and have high power consumption.

The design of MobilityFirst Architecture assumes a flat, non-hierarchical NA and GUID which makes longest prefix matching method useless for our router design. If we propose to use a CAM to hold all the possible NAs and GUIDs, we would require a very large CAM which could hold millions of entries. Such a router would be very expensive and would consume huge amounts of power. Therefore we propose a way of using other memory resources to do the job.

A packet header structure of MobilityFirst packet is as shown in figure 3.1. It can be seen that the packet has 160 bit Desination and Source GUID fields and 160 bit Destination and Source NA fields. For the router prototype, we have reduced the GUID and NA to 32 bit (MSB of 160 bit fields, big endian) to have enough hardware on the NetFPGA 1G available to explore fast lookup approaches. It should be noted that the existing design can be scaled to support 160 bit NAs and GUIDs as well if we go for a more advanced FPGA platforms.

0		31
Service ID	*Header Length*	*Next Header*
Protocol ID	*Hop Count*	*Payload Offset*
Payload Length		
Destination GUID (short) - 5w		
Destination Network Address – 5w		
Source GUID (short) – 5w		
Source Network Address – 5w		
Service Header Extension(s) ...		

Figure 3.1: MobilityFirst Packet Header Structure

The MobilityFirst Architecture places an emphasis on the transferring data PDU in chunks or blocks which can be as large as a few hundred megabytes. Since it will be using the underlying Layer-2 (IEEE 802.3 MAC layer) protocol [15] to send packets on the network, using a jumbo frame (9000 bytes) format is more advantages over the standard 1500 Byte Ethernet frame. However for this prototype, we have restricted ourselves to the standard Ethernet frames. Large chunks would generate flows of streaming data packets. Hence we designed the MobilityFirst router to be a flow-based router.

When a router receives a packet, it performs lookups for the GUID and NA simultaneously. If the NA matches router's NA (local NA), it is assumed that the packet belongs to a local network of the router. In the case of local NA, GUID looked up yields a destination MAC address and output port and the packet is forwarded to the local end point. This is intra-domain routing. For NA other than local NA, NA look-up searches for the output port of the router (the next hop towards the destination) and the packet is forwarded respectively. MAC addresses for next hop are programed into the router during initialization. This type of packet forwarding is inter-domain routing.

From the above, firstly it can be seen that a router should be able to resolve the output port from a large set of NAs within a short time to handle the traffic going across the networks. It needs a large table of NA entries (possibly all the NAs globally deployed) and a fast lookup approach. Secondly, a router may have a smaller GUID lookup table for a local working-set of GUIDs.

The third scenario is when an incoming packet needs a GNRS lookup which happens on the host controller and might take several milliseconds to get resolved. This packet is identified when it does not have a NA (NULL NA value), or an NA which has expired when the packet hop count reduces to zero. GNRS lookup implementation would be done on the host controller since it requires more complex algorithms. However, once the GNRS lookup is complete, GUID-NA pair can be cached in the router and the successive packets can have NA binding done from within the router.

3.1 High-level Design of Router

The MF router has four Gigabit Ethernet ports, the forwarding plane and Forwarding Information Base on NetFPGA 1G card which is connected to the Host controller over PCI bus. The incoming packets are queued to be processed by an output port lookup block, and then released to their respective output ports, where they are queued again to be transmitted over the physical link. Figure 3.2 depicts the high level router design.

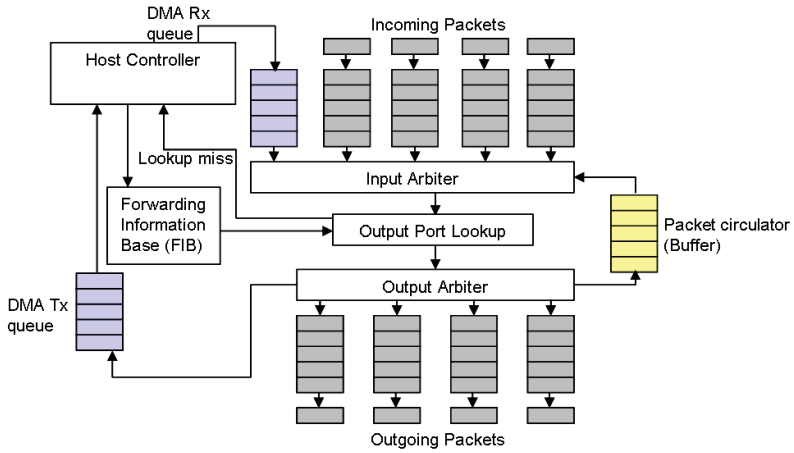

Figure 3.2: High-level Router Design

This design is based on the Stanford Universities NetFPGA 1G DRAM Reference Router Project. The Packet Circulator was added to allow buffering of packets which require slower host controller lookup operations. One set of DMA channel was included to forward and receive control packets to and from Host controller, but the host-side DMA support for MobilityFirst router is not implemented and hence it is left for future work. Packets come in through the GMII (Gigabit MII Double Date rate) interface at 1 Byte per 8 nsec (125 MHz). The packet is first checked for its integrity and then packed into 64bit words and moved at 64 bit per 16 nsec (62.5 MHz) inside the router.

Number of incoming flows , N = 4

Packet data rate per flow Rin = 8bits / 8nsec = 125 MB/sec

Total incoming data rate = N * Rin = 500 MB/sec

Packet flow with in the router = Rrt = 64bits/16nsec = 500MB/sec

Hence if the Output Port Lookup latencies are ignored and it is assumed that there are no packets which were previously buffered in Packet Circulator, the throughput of the router matches exactly that of the incoming flows. If there are packets in the packet circulator, or the DMA channel, our bandwidth would be shared amongst 6 flows reducing the throughput below 100%. However, allowing the lookups to take place in parallel to transferring the packet from input queues to output queues (pipelining) eliminates stall cycles for packets which are long enough to absorb their lookup times. As for the Packet Circulator, we propose to allow circulation only when Host Controller performs an update in Forwarding Information Base, to reduce bandwidth sharing during circulation. We are using NetFPGA core clock at 62.5MHz instead of 125MHz which is the default in NetFPGA, due to hardware timing closure issues which will be discussed in the Chapter 4.

Therefore, the bottleneck of the router is the Output Lookup Block and the focus of this research is to have a lookup strategy which will allow us to forward a packet based on both GUID and NA identifiers at the highest speed possible with the existing infrastructure.

3.2 GUID and NA Identifiers

By definition, GUID and NA will be unique and flat identifiers. GUID can be the address of a device or a content and will remain associated to the device or content unless it is reassigned (which is expected to happen rarely, since we would have a very large GUID space to accommodate everything). NA is the address of the Network in which the device or the content belongs. The space of NA is also sufficient to encompass all possible networks around the world. Both GUID and NA (NA is derived from the router's GUID) are expected to be cryptographically generated. To design ahead, they are assumed to be spread-out in the space they span, in a uniformly random order as

shown in figure 3.3.

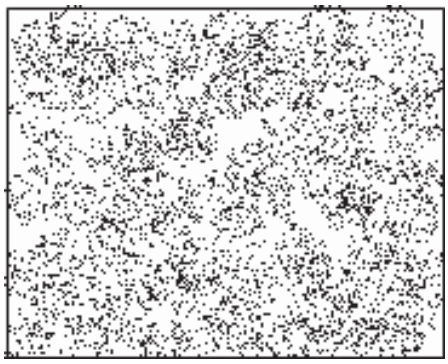

Figure 3.3: GUID and NA Spatial Distribution

If n is the number of bits in the identifier, then 2^n is the space of the identifiers. Having uniform distribution gives us a probability distribution function:

Let Random variable x, have values Rx = [0 *to* 2^n]
fx(x) = $1/(2^n)$ if x belongs to Rx
else fx(x) = 0

The number of NAs would be large (in the order of millions) and the number of GUIDs will be substantially larger. It would not be possible for a router to have all the possible GUID and NA entries in its FIB in a cost effective way. However, it was can be noted that a router would need to service only a small set of GUIDs, the ones which are bound within its local network. Therefore, we proposed to use a small table for the GUID FIB table in the forwarding plane. The NA lookup still requires a large table and hence more resources have been dedicated towards NA FIB. The packets containing a GUID which is yet to be bound to a NA needs a GNRS lookup and a query is sent to the host controller to perform a GNRS lookup which might take a longer time (in the order of a few milliseconds). Hence, results of GNRS lookups is allowed to be cached in MobilityFirst Router to allow flow based direct binding.

3.2.1 Inter-Domain Routing

If the NA of the packet doesn't match the NA of the router, it should forward the packet to another network through point to point links to other backbone routers. This is known as Inter-Domain routing as shown in figure 3.4.

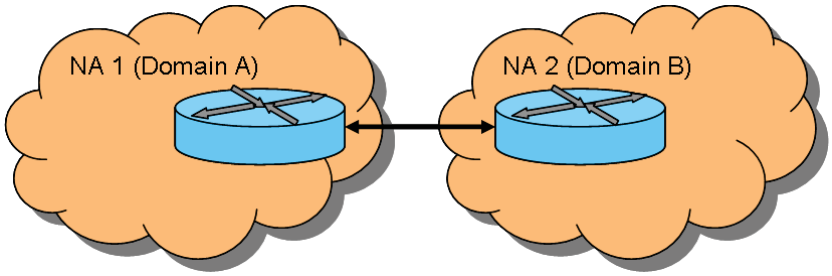

Figure 3.4: Inter-Domain Routing

The Output lookup block looks-up for the NA in the NA FIB and resolves the output port. Due to not being able to take the advantage of LPM, we had to find another solution to high speed lookups. We got inspired from general processor architecture and designed a 2-level cache to store the lookup data. The level-1 (L1) cache is very fast and support line rates on all the ports but is smaller in size and hence can accommodate only a few entries. The level-2 (L2) cache is slower but much larger than the L1 cache. Caching helps us when there is temporal locality and for MobilityFirst packets, the chuck based packet flows could take the advantage of such architecture. We chose to use a Binary Content Addressable Memory (BCAM) to act as the L1 cache. We planned to use external high speed SRAM to be used as L2 cache. The implementation details will be discussed in the Chapter 4. While the space issue is resolved, there was another issue of looking up the entries. BCAM is faster in access time and also lookup since a CAM does a parallel search in all locations and responds in one clock cycle. SRAM, on the other hand, has a higher latency to access data and cannot lookup the data in parallel. Therefore, we deployed some basic hardware solutions such as hashed seperate chaining (with linear and binary search) and bloom filters to address this problem. Details of these will be discussed in section 3.3.

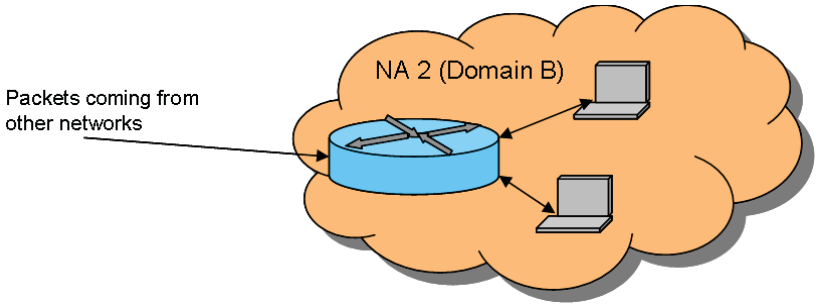

Figure 3.5: Intra-Domain Routing

3.2.2 Intra-Domain Routing

If the NA of the packet matches that of the router itself, it should be forwarded to a specific node within the same network. This is Intra-Domain routing (figure 3.5). In this case the GUID FIB is looked up and destination MAC address is updated in the packet before forwarding it to the respective port. As discussed in the previous section, the number of GUID entries which could be present in the router's GUID FIB does not need to be very large, because the working set of GUIDs present in the router's network will should be small (as compared to total number of GUIDs). Hence, a straight forward approach of using a Binary CAM (BCAM) was chosen. The GUID FIB BCAM stores GUIDs, and indexes to memory (FPGA Block RAM) to get the MAC address and output port. The standard IP based router would have an ARP table for this, but since the ARP hasn't been designed so far for MobilityFirst, we have taken an assumption that host controller would have the knowledge of MAC addresses and would write them directly into the GUID FIB.

3.2.3 GNRS lookup support

When a packet arrives with an expired NA or NULL NA, it needs a process of GNRS lookup to obtain NA and binding to attach the NA to the packet. GNRS lookups require long processing time (in the order of a few milliseconds) and takes place at the control plane of MobilityFirst. Hence such lookups are cached in the GUID FIB (BCAM + FPGA Block RAM) to allow rest of the packets of the flow to be bounded and routed directly from the forwarding plane. The host controller is expected to remove the entry

from GUID BCAM, once the GUID-NA pair is expected to have expired.

3.3 Output Port Lookup Block Design

The Output Port Lookup is the bottleneck of MobilityFirst Router. It has to support all valid lookup permutations at high-speeds. Figure 3.6 shows the design of this block. It has four components, Packet Parser, NA Lookup, GUID Lookup and the Packet Header Update.

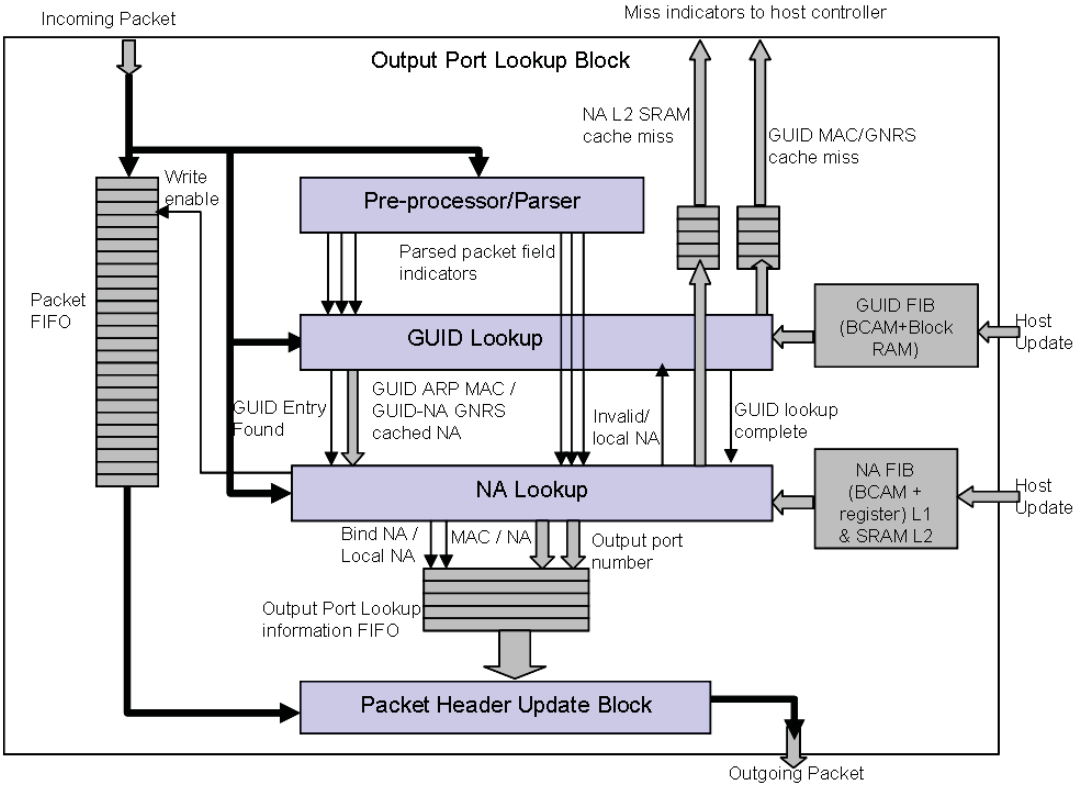

Figure 3.6: Output Port Lookup (Pipelined) Block

We started off with a non-pipelined design which would stall the input queues for lookups. Hence the packet would have to wait in the input queue untill the result of the output port lookup is resolved. Obviously, these stall cycles directly impacted the throughput of the packets. We did a comparison of the throughputs for Linear and Binary search speeds for different hash chain lenghts using the non-pipelined design.

Later we improved the design by pipelining the lookups. In this version, we allow the packet to be transfered into a Output Port Lookup (OPL) FIFO while simultaneously

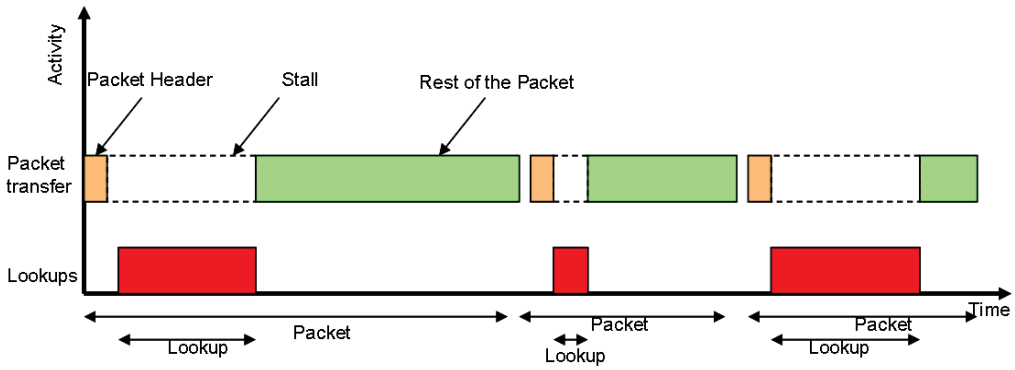

Figure 3.7: Non-Pipelined Lookups

performing lookups. If the packet is long enough for lookup to complete, the Header Update block starts reading from the OPL-FIFO, updates header and forwards packet to Output Queue arbiter.

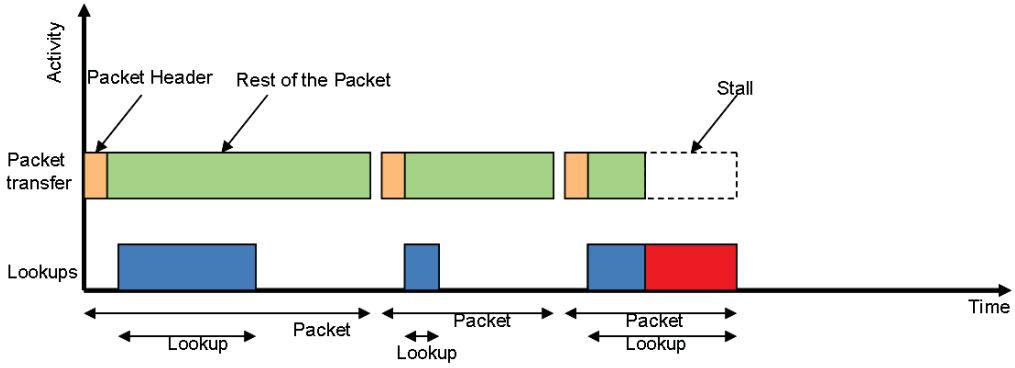

Figure 3.8: Pipelined Lookups

Both NA and GUID lookups work in parallel to resolve the fate of the packet. Since the GUID field precedes NA field in MobilityFirst Packets, GUID lookup starts ahead of NA. If the packet type is of intra-domain routing, NA stops lookup as soon as it realizes that NA is the local NA. NA lookup Block then waits for the GUID lookup to complete and obtain the destination MAC address from GUID FIB. If the packet type is of inter-domain routing, NA continues to lookup and obtains the output port information from NA FIB. If the packet type is of GNRS GUID-NA binding, the NA waits for GUID lookup to complete and get the cached NA. Thereafter, NA lookup follows the regular NA search method to obtain the output port from NA FIB. Once the lookup is complete, NA lookup block inserts the lookup information: 12bit output

port, 32bit NA or 48 bit MAC address, 1bit Bind NA flag and 1bit update MAC, into a small Info FIFO. Header Update Block reads from Info FIFO is read and reads the packet Packet FIFO, updates packet and transfers it to Output Queue Arbiter which sends it out through the respective output port.

3.3.1 GUID FIB Lookup Design

Figure 3.9 shows the GUID FIB lookup Block. It uses a set of Binary Content Addressable Memory with FPGA Block RAM to hold the GUID lookup table. The current design uses the same table for GUID ARP (Intra-domain) cache and GUID NA (GNRS lookup cache) and one bit flag in the table distinguishes between the two entries. GUID lookup also gets information from NA lookup block about the NA received in the packet. If NA was a local NA or invalid NA, a GUID lookup miss is important to host controller and is inserted in GUID miss queue.

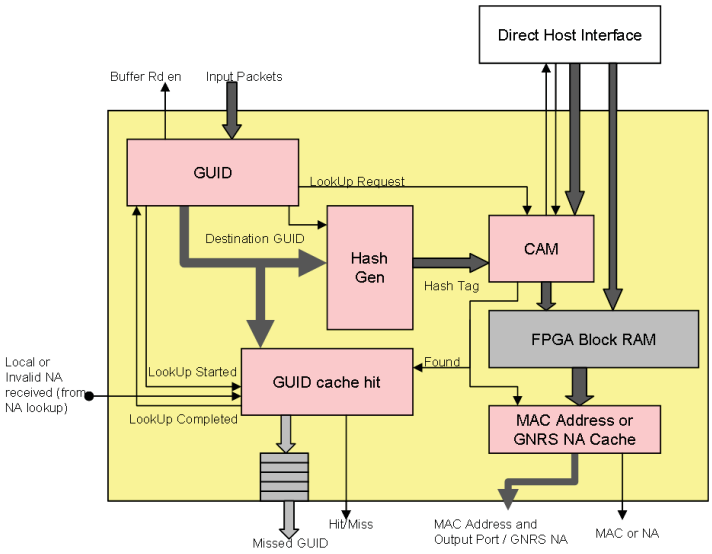

Figure 3.9: GUID FIB Lookup

3.3.2 2-Level NA Lookup Design

Figure 3.10 shows the 2-Level NA lookup Block.

When NA is parsed from an incoming packet or obtained from GNRS GUID-NA cache, a search request is sent to L1 BCAM and the L2 SRAM cache simultaneously.

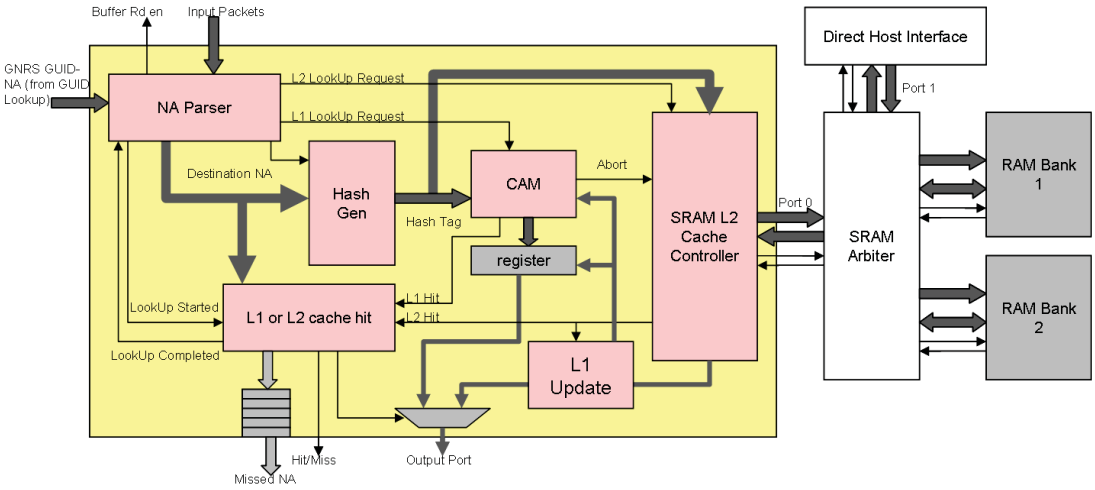

Figure 3.10: NA FIB Lookup

The BCAM responds in 64ns (which takes 4 clock cycles of 16 ns). If there is a hit at L1, output port is obtained L2 cache lookup is aborted. If there is a miss at L1, the state machine waits for L2 to respond (which might take several clock cycles), the lookup time for L2 depends on the search scheme. If there is a hit in L2 then output port is obtained. If there is a miss at L2, then a L2 miss indication is inserted into the L2 miss indication queue and the packet is directed towards the Packet Circulator. The L2 miss indication is not fully reliable in Bloom Filter scheme due to the false positives, but that is the inherent drawback of Bloom Filters.

L2 lookup, being done on SRAM takes multiple cycles and hence 3 approaches were chosen to perform quick hardware search:

1. Seperate Chaining Hash Table with Linear Search
2. Seperate Chaining Hash Table with Binary Search
3. Counting Bloom Filters

3.3.3 Seperate Chaining Hash Table with Linear Search

As mentioned previously, the probability density function of random variable representing NA, has a uniformly random distribution over the space of NA and hence direct bit-extraction was chosen as the hashing function (this has some space saving advantage

as will be discussed later). Hence the 12-bit MSB of NA (32 bit) is used as the hash to index the table. However, due to the limitation of memory (4 MB SRAM), not all NAs can be inserted into hash table. The SRAM used has 32-bit wide address bus and hence fits nicely for storing NAs (or GUIDs) which we have taken to be 32-bit for this prototype. It can be seen that even 160 bits can fit in since it is 5 32-bit words without any internal fragmentation.

If only $1 * 1024 * 1024$ NA can be entered and are chosen based on the frequency of being used by the host controller, there may be variations in the distribution of the final set that is entered in hash table. For this reason, the 12-bit Hash generated does not index SRAM directly but instead indexes a Translation Look-aside Buffer (TLB) which is implpememted in SRAM and has the physical address of the hash chain in SRAM. This TLB is stored in first 2^{12} i.e. 4 K Word locations. This implies that rest of the 1020 K Word locations can be used for storing NAs in 2^{12} i.e. 4K sections (hash chain).

Each section is nothing but a chain in the Hash table and can store up to $1020K/4K$, 255 NA entries. Now, with the help of TLB, we could have some variations in the size of individual sections. Since the top 12-bits were already matched while hashing, we need only LSB 20 bits to be stored at a SRAM 32-bit word location which gives us 12-bits free for use. We are using 4 MS Bits to store the output port information (currently un-encoded 4 bits for 4 output ports).

Figure 3.11 shows the SRAM Memory Map and the L2 Lookup Block.

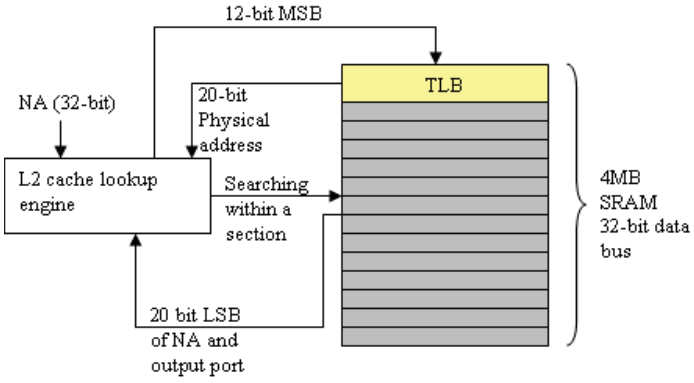

Figure 3.11: SRAM Memory Map and Lookup

In linear search, we start from the lowest address of the chain and progress until a match is found or the section ends.

3.3.4 Seperate Chaining Hash Table with Binary Search

The section (chain) address translation is the same as in linear search seperate chaining Hash table. In binary search, the NAs are expected to be ascending order and a binary search is performed within the section. It might sound at first that binary search is always better than linear search since it has a time complexity of $log_2 N$ while linear search has complexity of N in the worst case (here N is the size of the section). However, the SRAM access latency presents advantage to the linear search. SRAM is access fully pipelined but the pipeline length is 8 (including Arbiter). Hence it takes 8 clock cycles to get data after the address has already been sent. This causes a lot of wastage of bandwidth for binary search which has to wait for the data before sending the next address. On the other hand, linear search has to wait the first 8 cycles to start but later can compare data on every clock due to the burst mode of SRAM. This makes linear search $(8 + 1)$ times faster than binary search (Binary search needs 1 extra clock cycle to compare). There is a certain threshold of section size where binary search starts to outperform linear search and this can be calculated.

We have the condition $8 + (9 * Log_2 N) = (N + 8)$ for the threshold which gives us $N = 51$ as the theoretical limit. Above this limit Binary search is better than Linear and below this limit, Linear search outperforms binary.

Comparison of Linear search vs Binary search times on Pipelined SRAM

Hence if linear search should be used, size of the section should be $\cong 64$ which means, the number of sections can be increased to $16 * 2^{14}$ and the hash bits can be increased to 14bits. Above the threshold size, binary search is better option, and if size of section and the number of sections and hence the hash bits can be adjusted accordingly. Binary search also imposes a constraint on the order of data, i.e. they should be in ascending order. This makes the host controller L2 cache update longer as it now has an average update time complexity of $N/2$ as opposed to 1 in case of linear search.

3.3.5 Counting Bloom Filters

Bloom filters can be effectively used for set association and have been used in networking for high speed predictive lookups. Bloom Filters tell wether an element belongs to a set. The concept for Bloom Filter is simple, generate K different Hash Keys of an element using K Hash functions, and set the bits in a hash table of M bits, corresponding to the Hash keys, to 1 . After all N elements have been inserted into the Hash table, the initialization of Bloom Filter is complete. Such a Hash table which has only bits to represent presence or absence of elements is called a Bloom Filter. To check if an element is present in the Bloom Filter, we need to hash it using K hash functions and check wether all bits corresponding to the hash keys are 1. If any of the K bits is 0, then element is not present in the set. Hash tables inherently have collisions and this leds to presence of false positives. This is the main drawback of Bloom filters. However, with well designed hash functions and resonable N/M ratio (load factor), we can reduce the false positives to a minimum. Mitzenmacher*et. al* [11] have derived the expression for optimal value of K and minimum false positives that present the theoretical bounds. The Counting bloom filter is an extension to standard bloom filters, where we keep a count instead of a bit for every hash key. This allows us to delete an entry from bloom filter which can be done by decrementing the corresponding counts. This does not affect the false positive rate in any way.

$K = ln(2 * (M/N))$. For our implementation we have chosen $K = 3$, and to get low false positives, we need a (N/M) ratio to be less than 0.099574.

We are using one bloom filter per output port (hence four bloom filters), and perform search in all 4 bloom filters simultaneously. We used the 32 bit word access of SRAM to our benefit. We have SRAM which is 32-bit word aligned, and we propose to use 4 bits for a count. Therefore in one 32bit word we can accomodate 8 counts i.e. twice the number of output ports. There for we can have $M = (2^{20}) * 2$ for each bloom filter. If $M = 2^{21}$ then $N = 0.099574 * M = 208822$ NA entries per port which would give us a low false positive ratio. Figure 3.12 shows the SRAM map for our Bloom Filter

implementation.

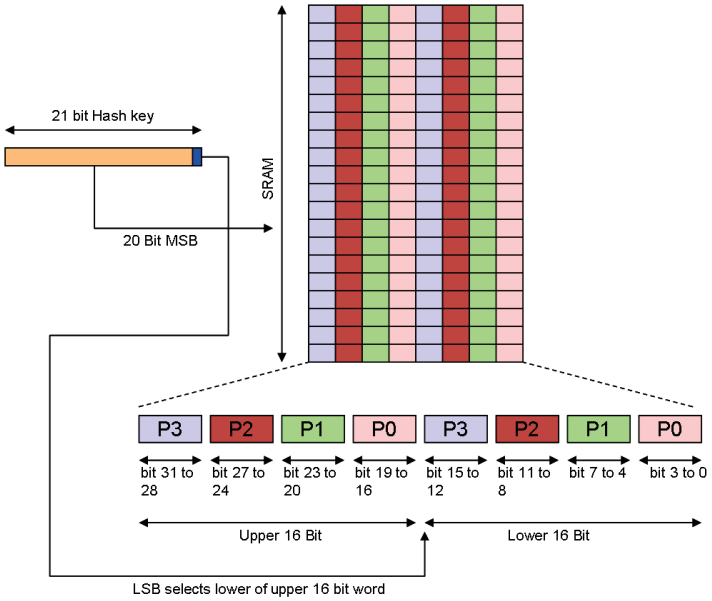

Figure 3.12: SRAM Memory Utilization for Bloom Filter Implementation

Mitzenmacher*et. al*, have shown that for above optimal value of K, we can calculate the false positive rate $f = (0.6185)^{m/n}$ which for our case (for K = 3) would result in $f = 0.00802$. In practice, we are able to achive false positive close to $f = 0.017$.

As mentioned by Qiao *et. al* [12], we can use $K = 3$ i.e. 3 hash functions for a reasonable evaluation of Bloom filter implementation. We chose the following 3 Hash Functions for out implementation:

- LSB 21 Bits of IEEE 32-bit CRC with generator polynomial $G(x) = x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1$

- Bit Xor plus bit extraction

  $h20(x) = x31 \oplus x30$; $h19(x) = x29 \oplus x28$; $h18(x) = x27 \oplus x26$; $h17(x) = x25 \oplus x24$;

  $h16(x) = x23 \oplus x22$; $h15(x) = x21 \oplus x20$; $h14(x) = x19 \oplus x18$; $h13(x) = x17 \oplus x16$;

  $h12(x) = x15 \oplus x14$; $h11(x) = x13 \oplus x12$; $h10(x) = x11 \oplus x10$; $h9(x) = x9 \oplus x8$;

  $h8(x) = x7 \oplus x6$; $h7(x) = x5 \oplus x4$; $h6(x) = x3 \oplus x2$; $h5(x) = x1 \oplus x0$;

  $h4(x) = x31$;

  $h3(x) = x24$;

  $h2(x) = x16$;

$h1(x) = x8;$

$h0(x) = x0;$

- H3 class Hash function [16] [17]

Since one 32bit Word of SRAM, stores two entries of all four bloom filters, we just need 3 read cycles from SRAM to get the output of all 4 bloom filters. When the router is initialized, host controller downloads all 4 bloom filters into SRAM. L2 cache controller state machine calculates 3 addresses by 3 hash functions and perform 3 pipelined reads from SRAM. If the count at all three hashed addresses is non-zero, the particular output port is valid for the packet. Due to the false positive nature of bloom filters, the packet may be sent out from a wrong port. But since there are no false negatives, the packet definitely is sent out from the intended port. The network has to bare the extra overhead due to extra packets generated form false positives. Hence, we would recommend to use such routers adjacent to exact matching scheme routers which would drop these packets. Alex Fabrikant *et. al* [18] have mentioned some methods to mitigate the false positves. Sarang Dharmapurikar *et. al* [19] have come up with a design which combines the Bloom Filter to improve exact-match scheme average performance, where a false positive cannot occur.

3.4 Comparison of Exact Match and Bloom Filter Lookup schemes

3.4.1 Speed of Lookup

Exact match scheme of seperate chaining hash table with binary search for 4096 bins gives us an average length of 255 elements per chain. The worst case lookup times is $8 + (9 * log_2 N)$ clock cycles of 16nsec which is equals to 1279.18ns or 1.28us. Exact match scheme of seperate chaining hash table with linear search can be done best for 51 elements or less in a chain. Assuming the chain length is 51 and in worst case scenario, lookup time should be $8 + N$ clock cycles of 16nsec which equals to 944ns. For comparison with binary search, if chain length is 255, time required for linear search in worst case would be 4208ns. The Bloom filter has fixed lookup times and is always

$t_{hash} + 8 + 3$ clock cycles of 16nsec. Our implementation of hash function takes only 3 extra clock cycles, therefore $t_{hash} = 3$, giving us a fixed lookup time of 224ns. We have tested for minimum packet lengths of 174Bytes which requires $174 * 8 = 1390ns$ and since we have to service four ports (assuming no time shared for packet circulator) Bloom Filter lookup would need $224*4 = 896ns$. If we add 5 clock cycles for the lookup state machine latencies, we would get a total of $896 + 320 = 1216ns$ which is still less than 1390ns. Hence we can support line-speed lookups for Bloom Filter lookup scheme even for the smallest packet size we have considered for our implementation.

3.4.2 Memory requirement

Exact Match lookup scheme requires us to store the NA entries in SRAM. For our implementation, we need 32 bit to store one NA entry. However since our hashing involves direct bit extraction, we get some free bits in the 32bit word of SRAM to store output port information. We have used 4096 bins which required 12 bit MSB to be used as Hash key, leaving 12 bits to store output port information. As discussed in section 3.3.3, we can store $1020*1024 = 1044480$ entries in SRAM. Bloom filter implementation allows us to use 208822 entries per port giving us a total capacity of $208822*4 = 835288$ which is less than match capacity. However for the low false positve ratio and constant high search speed we could allow for some extra memory usage. However it should be noted that the total capacity only holds good if NAs are *equally distributed amongst all four ports which may not happen in practice.* For a load factor of 0.1 we should be have a false positve ratio of 0.0082. In practice we are able to achieve a false positive rate of 0.0174.

3.4.3 Energy Efficiency

Accessing 32-bit data with 20 bit Address on external SRAM at 16nsec per clock will cause good amount of dynamic power dissipation. The average number of memory access for exact match is always higher than the bloom filter because bloom filter requires only 3 read cycles to do find the output port.

3.4.4 Router NA FIB Update time from the Host Controller

The Binary search exact match is clearly the worst of all three since it requires a binary search to find the place to replace an entry. If the entry needs to be inserted then the performance is even worse due to movement of all entries below the insert address within the chain. We would therefore prefer to replace the entry rather than inserting it. The Bloom Filter perform the second best with calculation of 3 Hash key addresses and 3 Memory Writes for inserting each entry in the bloom filter. Linear search exact match is the best scheme for updates since host controller can simply append the NA entry at the end or replace any entry as per the update policy.

Chapter 4

Implementation of MobilityFirst Router

4.1 NetFPGA Card Specifications

NetFPGA 1G platform [13] was selected to prototype the MobilityFirst router. It has been designed by Stanford University for academic research in router design and networking. The following are the specifications of 1G platform:

- Field Programmable Gate Array (FPGA) Logic
  Xilinx Virtex-II Pro 50
  53,136 logic cells
  4,176 Kbit block RAM
  up to 738 Kbit distributed RAM
  2 x PowerPC cores
  Fully programmable by the user

- Gigabit Ethernet networking ports
  Connector block on left of PCB interfaces to 4 external RJ45 plugs
  Interfaces with standard Cat5E or Cat6 copper network cables using Broadcom PHY BCM5464SR
  Wire-speed processing on all ports at all time using FPGA logic,
  1 Gbits * 2 (bi-directional) * 4 (ports) = 8 Gbps throughput

- Static Random Access Memory (SRAM)
  Suitable for storing forwarding table data
  Zero-bus turnaround (ZBT), synchronous with the logic
  Two parallel banks of 18 MBit (2.25 MByte) ZBT memories

Total capacity: 4.5 MBytes

Cypress: CY7C1370D-167AXC

- Double-Date Rate Random Access Memory (DDR2 DRAM)

  400 MHz Asynchronous clock

  Suitable for packet buffering

  25.6 Gbps peak memory throughput

  Total capacity: 64 MBytes

  Micron: MT47H16M16BG-5E

- Multi-gigabit I/O

  Two SATA-style connectors to Multi-Gigabit I/O (MGIO) on right-side of PCB

  Allows multiple NetFPGAs within a PC to be chained together

- Standard PCI Form Factor

  Standard PCI card

  Can be used in a PCI-X slot

  Enables fast reconfiguration of the FPGA over PCI bus without using JTAG cable

  Provides CPU access to memory-mapped registers and memory on the NetFPGA hardware

- Hardware Debugging ports

  JTAG cable connector can be used to run Xilinx ChipScope Pro

- Flexibile, Open-source code

  BSD-style open-source reference router available from the NetFPGA.org website.

4.2 NetFPGA System Architecture

NetFPGA consists of two FPGAs, the Xilinx Spartan II (CPCI) FPGA which acts as a PCI bridge and FPGA programmer for the main VirtexII-Pro 50 (CNET) FPGA. Both FPGAs are RAM based volatile FPGAs and need to be reprogrammed after every power on. There is an on-board PROM from which bit file is loaded into CPCI

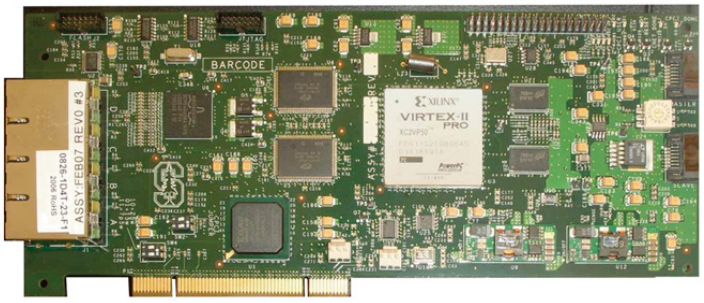

Figure 4.1: NetFPGA PCI Card

FPGA automatically on power on. When the host controller has booted, it programs the CNET FPGA over PCI bus through CPCI FPGA. Figure 4.2 shows the System Architecture.

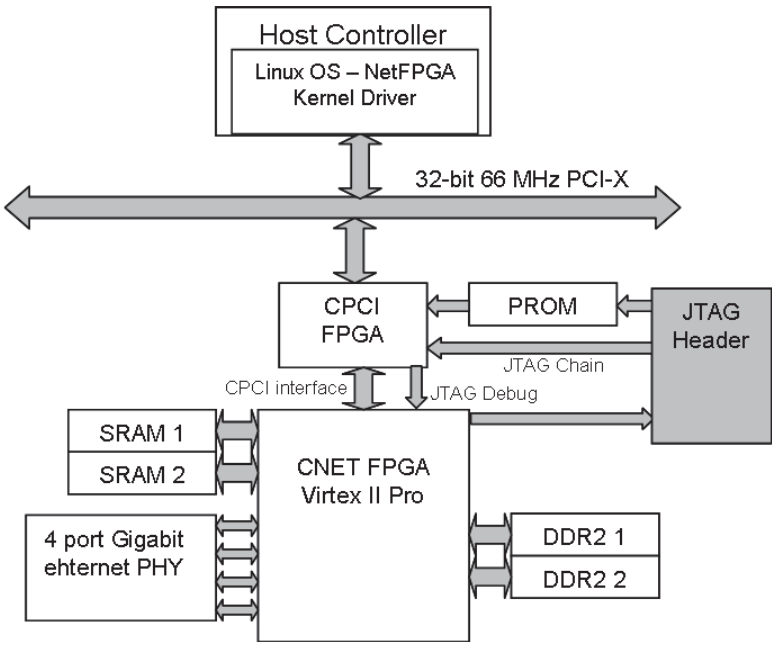

Figure 4.2: NetFPGA System Architecture

The system constists of a host computer (Host) which supports the NetFPGA PCI card over a 32 bit/66MHz PCIx slot. The CPCI FPGA works as a PCI bridge and a VertexII-Pro CNET FPGA is used to implement MobilityFirst Router. Each of the SRAMs have 19bit Address and 36bit Data bus and are clock synchronous to the CNET FPGA working at 62.5MHz for our design. The DDR2 memories are 32 MBytes each and have 24 bit Address and 16bit Data bus and work at 200MHz. The Gigabit

Ethernet PHY has four gigabit Ethernet ports and is connected using RGMII interface which has 4bit data bus and data is transferred at both edges of the clock (DDR). The clock frequency is 125 MHz for gigabit Ethernet.

4.3 MobilityFirst Router RTL Design

4.3.1 Top level Module

This design is based on the Stanford University's NetFPGA Reference Router Design [20]. We have modified the design to suit out requirements. The RTL top level module connects the internal blocks to the external world outside the VertexII-Pro. We are mainly interested in using the SRAM and DRAM DDR2 memories on NetFPGA board, the CPCI interface, which is a simplified PCI interface to the host controller and the RGMII interface to the gigabit Ethernet PHY. Hence we preserved these ports in top level. Figure 4.3 depicts the top level module of MobilityFirst Router.

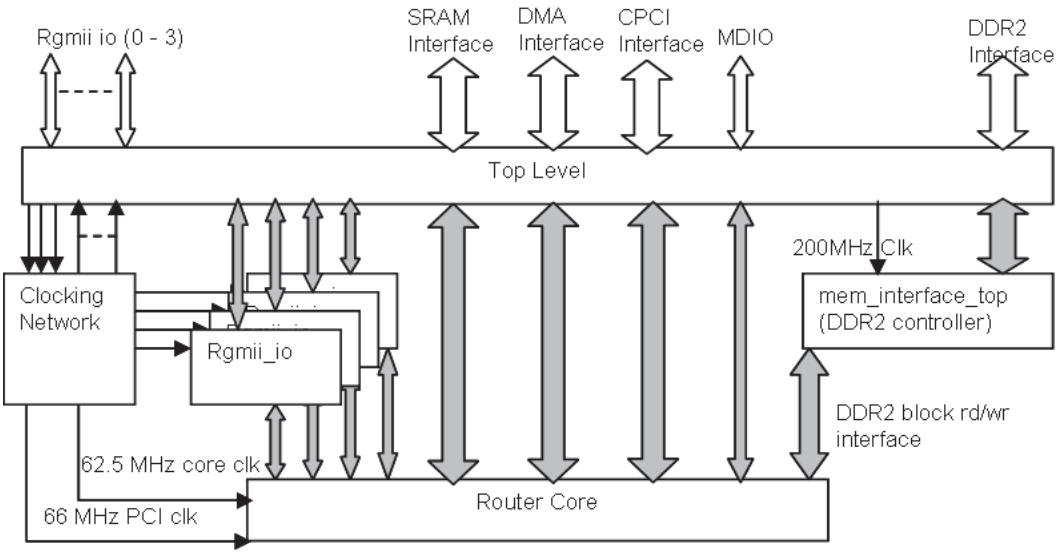

Figure 4.3: Top level Module

Clocking Network

The current design requires 8 Clock domains:

- 62.5 MHz Core Clock: This clock is fed through a DCM into the global clock buffer to clock the entire RTL of router core.
- 200 MHz DDR2 Clock: This clock is fed to a DDR2 controller which internally uses two DCMs for its operation. The DDR2 controller generates two clocks which are 90 degree out of phase (to maintain setup and hold requirements) and enable double data rate data transfers to DDR2 chip.
- 125 MHz RGMII Tx Clock: This is stabilized through a DCM and used to transfer TX data over RGMII interface for all 4 ports.
- Independent 125MHz RGMII Rx Clocks: These are 4 individual clocks soming from the PHY and are stabilized using 4 DCMs. They are used to read data over RGMII interface into respective RGMII blocks.

Hence, it can be seen that the net utilization of DCMs is 100% i.e. 8 DCMs in Virtex II pro XC2VP50.

The NetFPGA has a default core and SRAM clock of 125MHz but does provide a method to reduce it to 62.5 MHz [21], page 12. We chose the reduced clock to comply with the setup times. To reduce the core and SRAM clock, following functions were calls are required:

```
NF2_WR32(CPCI_CNET_CLK_SEL_REG, (unsigned int)0x0000);
  ResetDevice();
```

The NF2_WR32() function is an IOCTL call to write in NetFPGA device. The CPCI_CNET_CLK_SEL_REG is the address of clock select register in CPCI FPGA. Calling ResetDevice() resets the CNET FPGA.

RGMII IO Module

The Reduced Gigabit Medium Independent Interface (RGMII) is used transfer data between the PHY and CNET FPGA. RGMII IO Module converts the RGMII signals to

8bit GMII interface which is transferred to the MAC module for reception and converts 8bit GMII from MAC back to RGMII signals for transmission. This translation is required because the MAC module understands GMII interface only. We have made no modifications to this module.

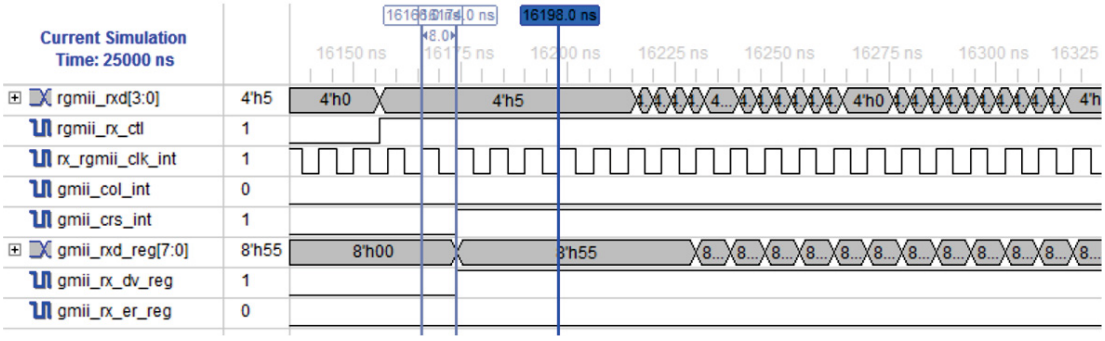

Figure 4.4: RGMII Rx to 8-bit GMII Data

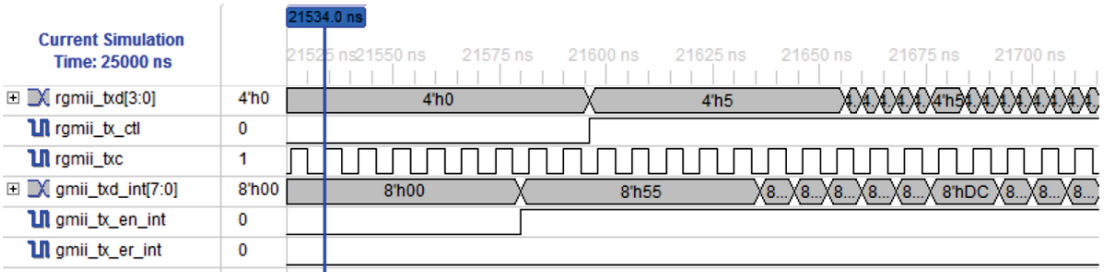

Figure 4.5: 8-bit GMII data to RGMII Tx

DDR2 Controller

Since Virtex II Pro does not have a hard IP for DRAM controller, this soft IP had been generated by Xilinx MIG tool. The tool also generates the constraints file for the controller. These constraints help in proper place and route of the controller. The original constraint file (.ucf) was modified by NetFPGA group at Stanford to match their board schematics. We have made no major modifications to this module.

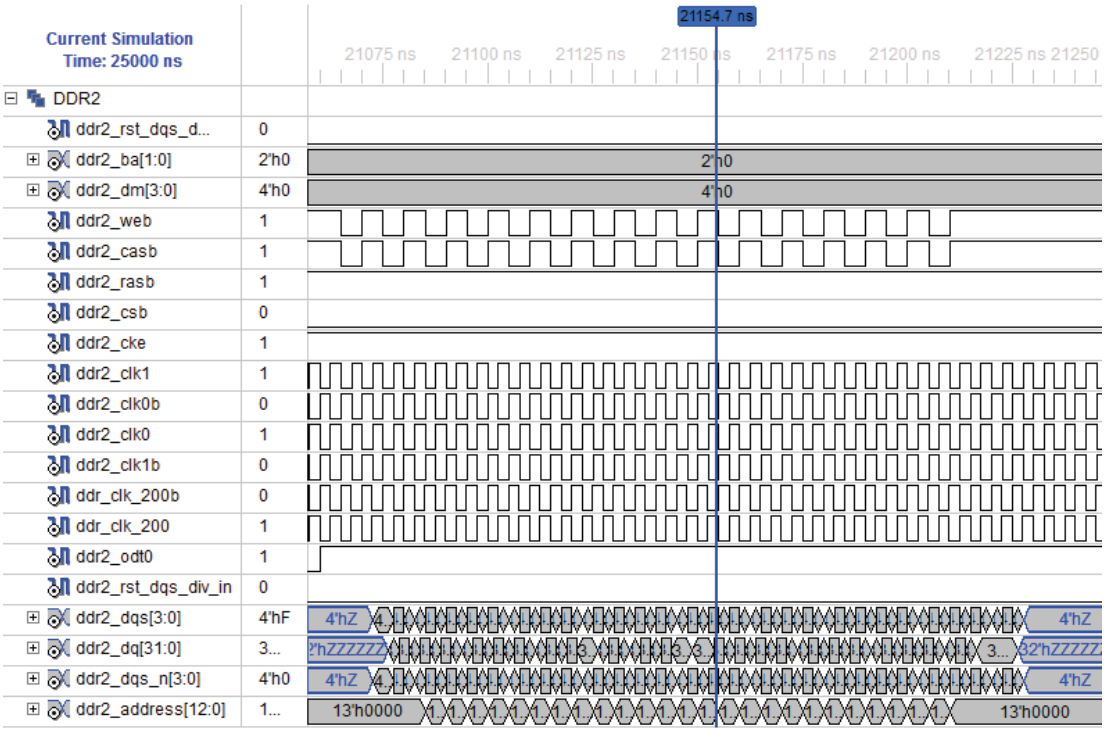

Figure 4.6: DDR2 Write Cycle

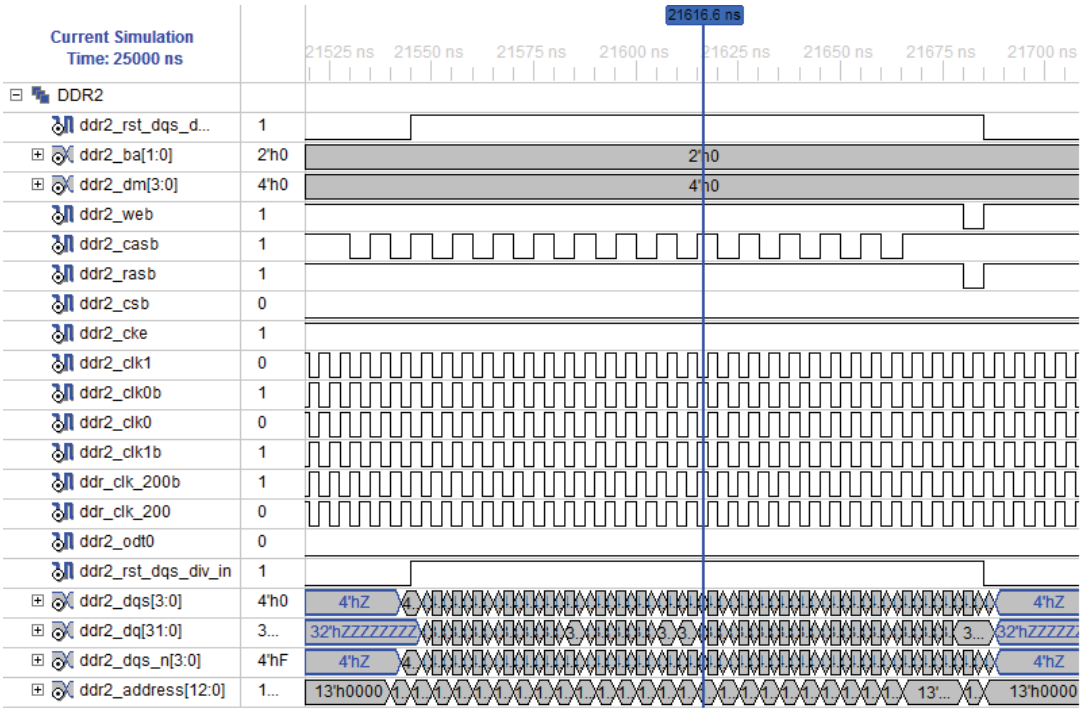

Figure 4.7: DDR2 Read Cycle

4.3.2 The Router Core

This module encapsulates the majority of the router components. Figure 4.8 shows the compenents of the Router Core Module.

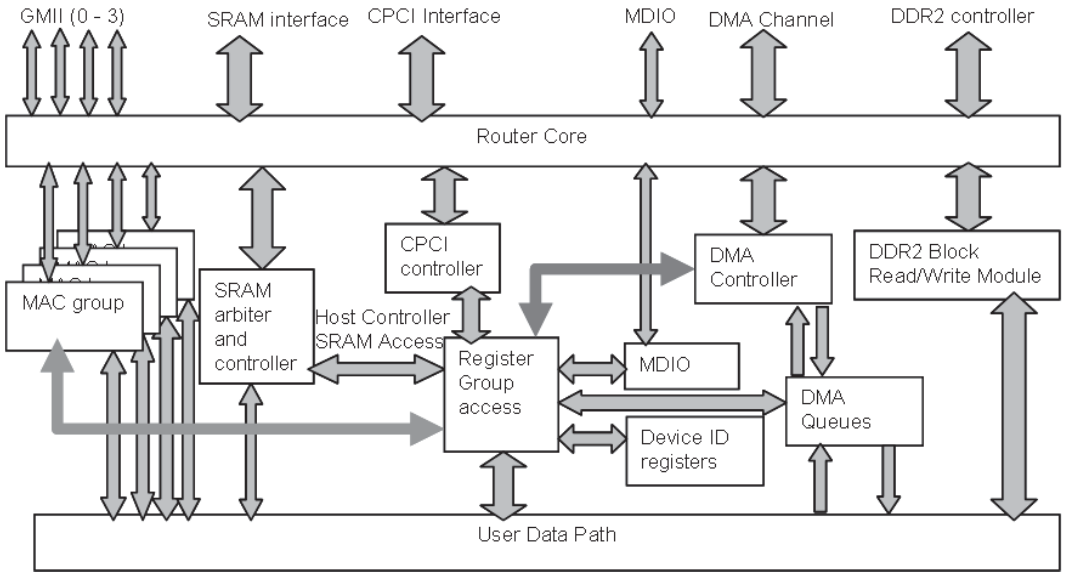

Figure 4.8: Router Core Module

As shown in the figure, the core level instantiates the SRAM arbiter (and Controller), DDR2 Block Read write module, CPCI controller, the core register group, the MAC modules, MDIO, Device ID registers and the User Data Path. We would discuss about each of these block in the coming sections. The DMA controller and one pair of DMA queues (Tx and Rx) have been added but the host controller support for DMA has not been implemented for MobilityFirst Router. Hence DMA block will be excluded from the scope of the Thesis.

The Medium Access Control (MAC) Group Module

Core instantiates four of these modules for the 4 Ethernet ports.

The major components of this module are the GMAC layer module which instantiates the transmitter and the receiver modules. The GMAC Receiver checks for packet integrity through CRC. The original Stanford University design had used a Xilinx IP core for this block, TEMAC. It needed its own license and was a heavy duty GMAC

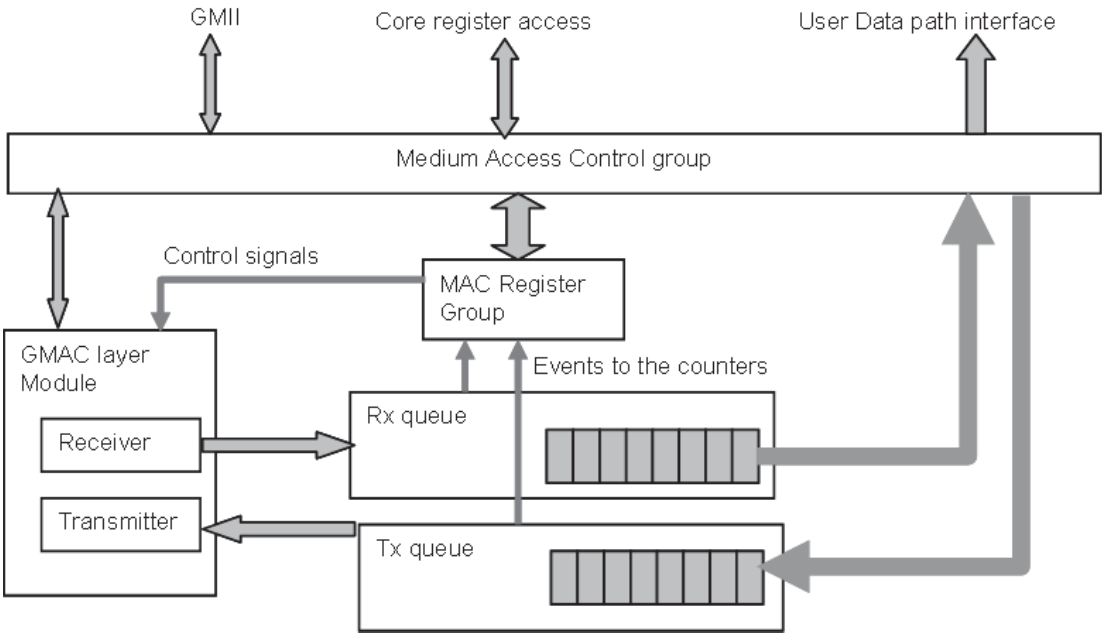

Figure 4.9: MAC Group Module

layer IP which had all features of *IEEE 802.3* specifications. We did not require such a comprehensive GMAC to prototype hence we went ahead by using a code base contributed to the NetFPGA community by Erik Rubow from University of California San Diego. The modification done to this module was to add a MAC address filter. For the prototype, the following MAC addresses have been used for MobilityFirst Router:

- `48'h04_DEAD_BEEF_04: Port 0`
- `48'h08_DEAD_BEEF_08: Port 1`
- `48'h0C_DEAD_BEEF_0C: Port 2`
- `48'h10_DEAD_BEEF_10: Port 3`

The MAC address filter would not allow any incoming packet with other MAC addresses, not even Broadcast Address.

The Transmitter Module of GMAC was modified so that it could update the source MAC address on a packet before transmitting it. GMAC receiver is synchronous to GMII Rx clk (125MHz) and transmitter is synchronous to GMII Tx clk (125 MHz).

The Rx Queue Module is the original design from NetFPGA DRAM Router Project. No modifications have been done to this module. It packs 8bit data from GMAC to 64 bit Data and adds a 64-bit header to the packet. Along with the 8 bits, a ninth bit is added to indicate End Of Packet (EOP).

The assembling of 9bit to 72bit is done using an Asynchronous FIFO (size 8KB) which has a 9bit Din and 72bit Dout. Data is written synchronous to GMAC Rx clock and read synchronous to Router Core clock (62.5 MHz).

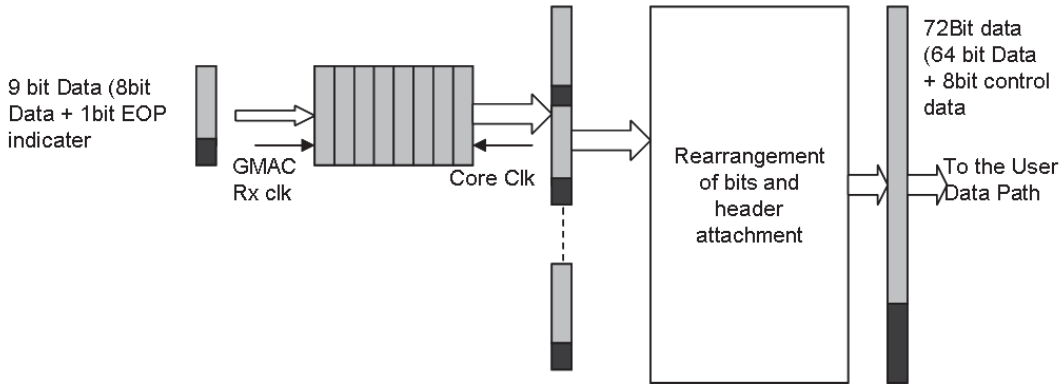

Figure 4.10: Assembly of Data Bytes at MAC Rx Queue

The 8-bit control value is set to 0xFF for the header word and zero of the Packet Data words except for the *last word* of the packet where one of the bits should be set indicating that it is the *last word* of the packet [20]. Bytes are padded at the end of a packet to make packet 64bit word aligned, but the control bits still hold the information about the *last byte* of the packet in the *last word* of the packet.

Format of the 64 Bit header is shown in figure 4.11.

16 bits	4 bits	7 bits	9 bits	4 bits	4 bits	12 bits
Output Port (Un-encoded)	**Mobility First Info**	Unused (future length increase)	Packet Word Length	Input Port (Encoded)	Unused (future length increase)	Packet Byte Length

Figure 4.11: Router Internal packet header

We decided to use the existing header format from NetFPGA Reference Router Design and add some information in the unused section. Tx queue does the opposite of Rx Queue, i.e. converts the 72 bits of Data and Control information into 8bit data which

can be fed to GMAC Transmitter module. No change has been made to Rx Queue and Tx Queue modules and they have been taken from NetFPGA Reference Router Design.

MAC group Registers module originally had a control register and a set of event counter registers for MAC group module. To have enough slack in timing during place and route, the counter registers of this module were removed, however their address space hasn't been reused and they can be added back if required. The Register addressing is mentioned in the last section of this chapter.

CPCI Interface Controller Module

This original module has been used from the NetFPGA Reference Router Design. Its main function is to collect write and read requests from the Host Controller over the PCI bus (through CPCI Bridge) and respond. PCI bus would have wait states to allow a Read or Write Requests to complete which is infact variable because certain registers of CNET Router take more clock cycles to finish Read or Write operation. This is primarily because many resources such as BCAM, Block RAM and SRAM are Memory mapped and need more clock cycles for their read/write cycles. This module works on PCI clock domain at 66MHz and on Core clock domain of 62.5 MHz and therefore requires Asynchronous FIFOs to cross the clock domains. Both FIFOs have been generated by Xilinx Coregen.

Register Group Access Module

This module de-multiplexes PCI address into the Core Register Address, User Data Path register Address and SRAM Address and forwards Read/Write requests to the respective modules. The interface to each of the above modules is as follows:

1. Module specific width wide Address Bus (Output Port)
2. 32-bit Write Data bus, (Output Port)
3. 32-bit Read Data bus, (Input Port)

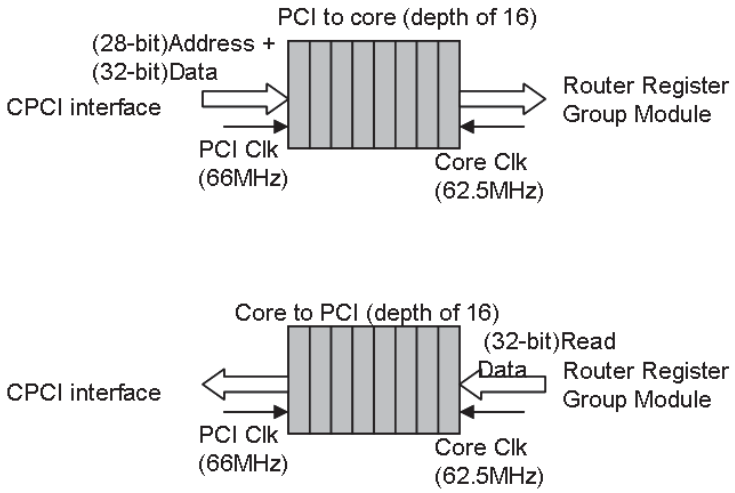

Figure 4.12: CPCI Asynchronous FIFO

4. 1-bit Host Request (Output Port)

5. 1-bit Read/Write (1/0) Signal (Output Port)

6. 1-bit Ack from module (Input Port)

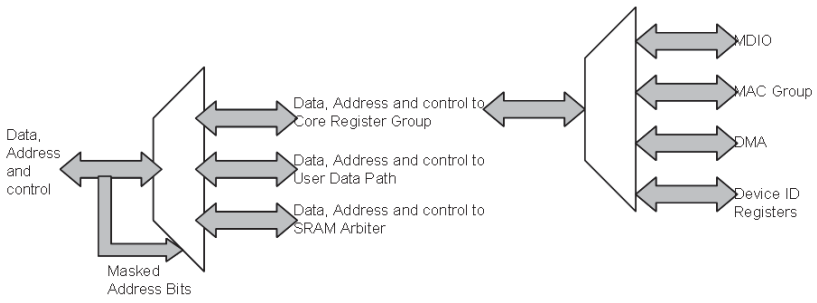

Figure 4.13: Router Register Group

The Core Register Group acts as a central arbiter to the MDIO, MAC group, DMA controller, DMA Queues and Device ID Module registers in a star structure. This module has been used from the original NetFPGA Reference Router Design.

MDIO Module

This original module has been used from NetFPGA Reference Router Design. This modules converts PCI register reads and writes to MDIO reads and writes to read and

write into the Gigabit Ethernet PHY registers.

Device ID register Group Module

This original module has been used from NetFPGA Reference Router Design. This modules contains the information on Device ID and version number through which host controller can identify MobilityFirst Router.

SRAM Arbiter

The base for this module has been taken from NetFPGA Reference Router Design and has been modified for MobilitFirst Router Design. This Module has two ports. Port0 is being used by NA L2 Cache Lookup Module. Port1 is used by Host Controller to directly Read or Write from or into SRAM through CPCI interface. SRAM Arbiter grants 20 access cycles to Port0 before allowing Port1 to pre-empt and get the access for 15 cycles. However our design doesn't allow preemption to occur since we ensure that when Host Controller is updating the NA FIB lookup tables, NA L2 cache lookups is temporarily disabled. Hence both Ports get dedicated access to SRAM by system level design. SRAM Arbiter Port1 register access converts CPCI read/ write requests to SRAM read/write requests. Ack' signal is asserted only after the read or write cycle is complete. There are 2 SRAMs on board, 2MB each, and are 32-bit word aligned. Therefore the address bus for each is 19 bits wide. However the Host controller and NA L2 Cache lookup modules see the SRAM as one 4MB address space with 32-bit Data bus. Hence, they have 20-bit addressing. SRAM arbiter uses the MSB (20th bit) to select between the 2 SRAM Blocks, by asserting write enable on the respective SRAM and selecting between Data buses while reading data. Figure 4.14 shows the data paths of SRAM arbiter.

DDR2 Block Read/Write Module

The base design of this module was taken from NetFPGA Reference DRAM Router Design. There were many changes made to this block to meet place and route timing and functionality. This module performs read and write in blocks into the DDR2 DRAM

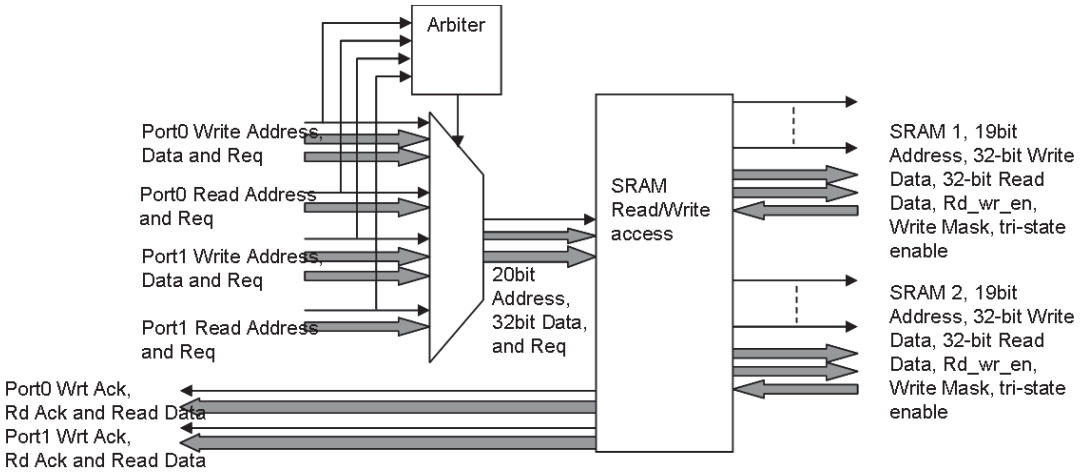

Figure 4.14: SRAM Arbiter

through the MIG tool generated DDR2 controller. DDR2 works efficiently when used for block data transfers because of the huge overhead of starting a read or a write cycle in DDR2 memory. In addition it has refresh cycles which preempt any read or write cycles to maintain integrity of data in DRAM. This module works on two clock domains, the $62.5MHz$ core clk domain and 200MHz DDR2 clock. The design uses two Asynchronous FIFOs to transfer data across the two clock domains.

There are two DDR2 chips of 32MB on NetFPGA board. Each has 16 bit wide data bus and since data is transferred at both edges of 200MHz, we transfer $(16+16)*2 = 64$ bits every 5ns. While writing data in DDR2, there should not be a case of unavailability of data in the FIFO (FIFO empty) to have a continuous flow of data. Similarly, while reading data out of DDR2, FIFO full should never occur, because the read cycle cannot be stalled.

Hence the design assumption for DRAM Write is:
FIFO Data in rate should always be more the FIFO data out.
To meet this requirement, original design used

```
For core clk =  125MHz
(72*2) = 144 bits wide Data in  and 64bits wide Data out
The FIFO writing rate = 125M * 144 = 18000Mbits per second
```

```
and FIFO reading rate = 200M * 64 = 12800Mbits per second

For core clk =  62.5MHz
(72*4) = 288 bits wide Data in  and 64bits wide Data out
The FIFO writing rate = 62.5M * 288 = 18000Mbits per second
and FIFO reading rate = 200M * 64 = 12800Mbits per second
```

Hence if the above width combination of FIFO is chosen then design assumption is met for both core clocks. We had to reduce the clk to 62.5 MHz to have a clean timing closure and therefore we would need 288Bit Din FIFO. Widening data path to 288bit would increase hardware and place more burden on Place and Route. In order proceed with the router design for the moment, we did not double the data path, instead we decided to lose on availability of DDR2 and went ahead with 144bit data bus. Since this would fail the design assumption, the size of the asynchronous FIFO was increased to 2KB (largest block size), which would first buffer the complete packet and then start the DDR2 write cycle, hence no pipelining in DDR2 Write.

This implies that, if block (packet) size is 1500 bytes, there would be stall cycles of $(1500 * 8/64) * 5nsec = 938ns$ for a packet of 1500 bytes. This drawback will limit the throughput of packet buffering, we decided to leave this to be optimized and pipelined in future.

A similar approach is used on the read side where a 2KB 144-bit Dout Asynchronous FIFO is used to buffer a packet. Althought reading is still pipelined and we can allow User Data Path to start reading from FIFO symultaneously to DDR2 read, since 2KB is larger than the longest packet length.

NetFPGA reference router design transfers fixed size (2034 bytes) blocks into 2048Byte Memory Segments. The Write pointer and Read pointer point to such 2048Byte segments. The trouble with this design is since the transfer is block-wise, we always require a continuous flow of packets to push the older packets through DRAM queues. In the even of bursty traffic, packets may have arbitarily large Inter-frame gaps, leading to incomplete packets sent out on the MAC ports. While prototyping, we wanted to test many different scenarios which will not always a continuous flow of packets, therefore we

modified the design to handle variable Block length read or writes so that the DRAM queue could be used in a packet-wise manner. With variable block size, maximum size of a block can be 2KB which is sufficient for the largest packet we can handle i.e. of $1500 + 18$ bytes. Figure 4.15 depicts the internals of this module.

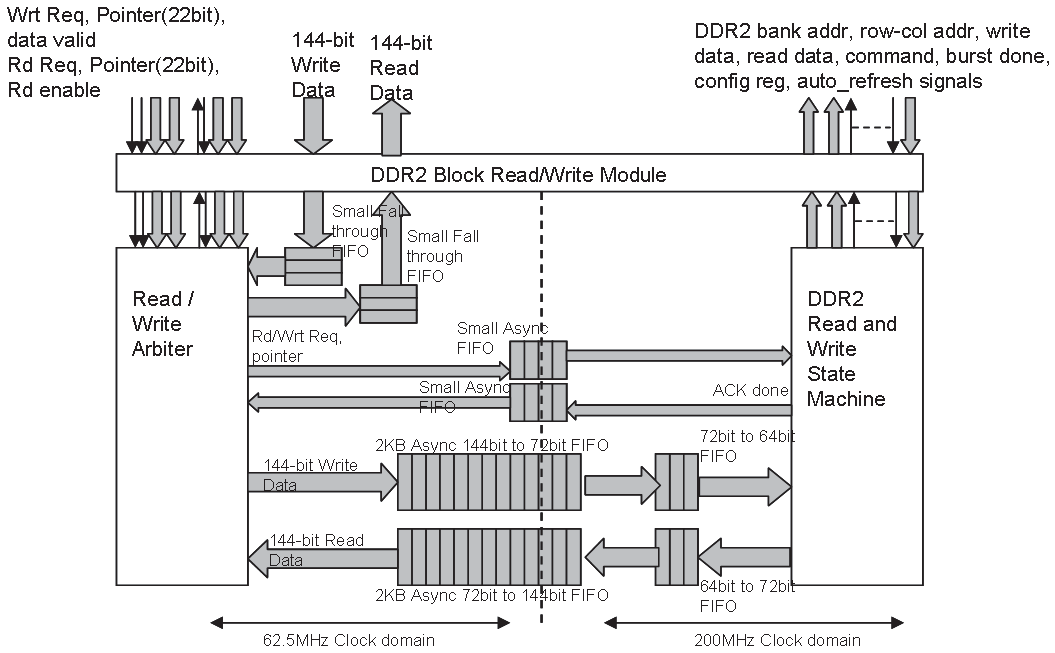

Figure 4.15: DDR2 Block Read Write Module

There are two disadvantages of using a scheme, first is the internal fragmentation which would happen because the DDR2 queue pointers are fixed at chunk size of 2KB. Infact, the 2KB chunk would be under-utilized even if a packet length is 1518 bytes. This problem can be solved reducing the chunk size which would reduce the wastage of memory, but would require writing into multiple consecutive chunks, increasing the complexity of DDR2 Block Write and Read Module. The second issue is with reading a block. Since the size of packet length is not know prior to reading the header of the packet, we had to start a read cycle with unknown length. Moreover the length in the NetFPGA header has number of data bytes in the packet, but since we are transferring even the control bytes in DDR2, we need to recalculate the total byte count that has to written into DDR2. Another subtlety lies in the padding, as we are converting 72bit words to 144bit words, we may have to pad to make the packet multiple 144bit words.

Beyond this, 64bit data to be written in DDR2 needs padding to keep it 64bit word alligned. All these calculations are difficult to prcocess in 5ns DDR2 Clock domain. Hence, we pre-calculate these values at 16ns Core Clock domain, attach Length Info Header to the packet, and write the packet in DDR2 Memory.

While DDR2 Read operation, we start off with a fixed read size (32 Bytes) from the starting address as per the read pointer. The Data valid has a latency of more than 16 Address reads, but then with the first data valid, DDR2 read module reads the length from the packet and modifies the read length while continuing to read the rest of the packet. This Length Info header is passed along to DRAM Queue Read Block in User Data Path so that similar scheme can be used to read the packets. Figure 4.16.

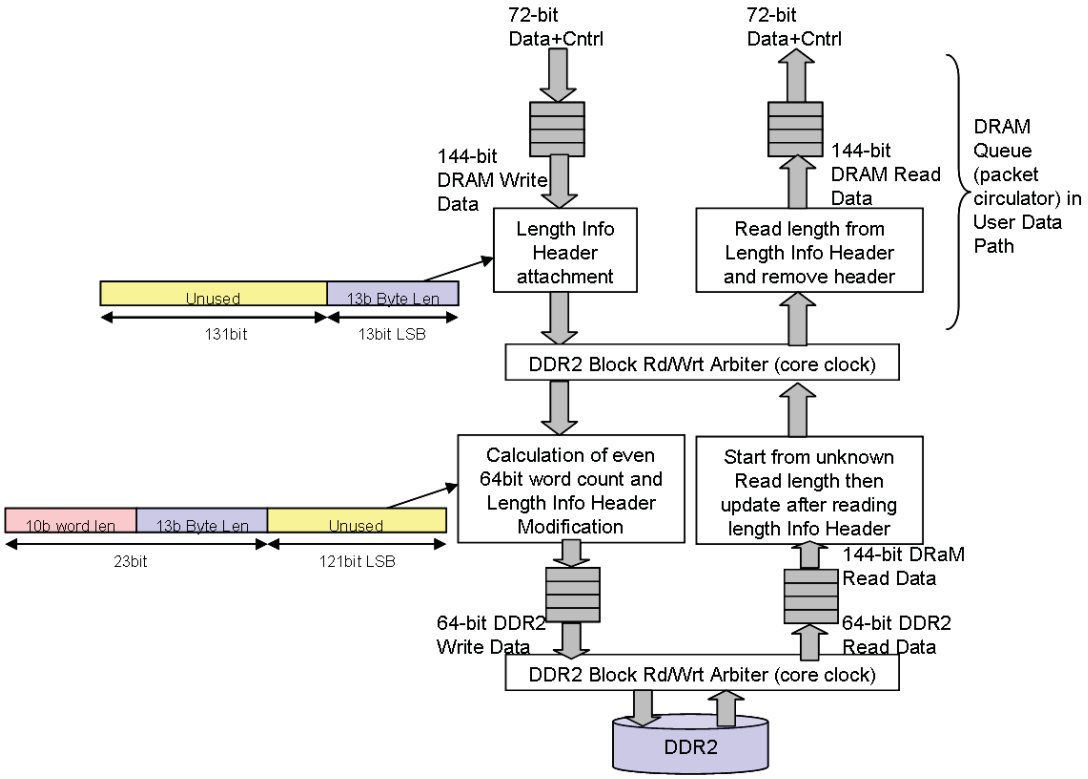

Figure 4.16: DDR2 Length Information Header

Closing Timing on Place and Route (PAR)

We have referred to the timing constraint problem many times and it would be discussed in this section. Timing constrains in digital VLSI design important factors which allow the PAR tool to place and route a synthesized net-list on the silicon(ASIC) or an FPGA. The goal is to place and route the logic circuit while not violating setup or hold times. The design did not face any hold time violations but was failing on setup time requirements. The setup time is the time before which an input logic level should be stabilized so that it can be registered by a flip-flop. The timing constraints tell place and route tool about a clock period. While the design itself should not have logic which would result in an impossible to meet timing netlist, a lot also depends on how the PAR works.

$T_{clk} - T_{setup} >= T_{pd}$ where T_{pd} is the propogation delay.

$T_{pd} = T_{logic} + T_{path}$

As it can be seen above that even if T_{logic} is small, if T_{path} i.e. delay caused due to path between two flip-flops is too high, it could result in a setup time violation. We attempted to reduce the T_{logic} as much as possible from the Verilog description and optimized RTL. We were still getting a large T_{pd} due to a large Tpath. To understand why this has happened, we would discuss briefly about the floor-plan of router design on Virtex II pro. The figure 4.17 shows the placement of the MobilityFirst Router RTL. The IO pin assignment for DDR2 controller forces DRAM Block read/wrt to be placed close to left corner. This modules has data paths to router core which in turn has data paths with PCI Bus, SRAM and RGMII interfaces. Hence, due to this large distance and large number of Data lines, place and route tool is not able to close timing at 125MHz. We increased the slack to compensate for this stringent placement and long Tpath delays, by reducing the clock to 62.5MHz. In addition we also had to add shift registers (3 stages) on the data paths to DRAM Block Read/write module to increase the slack. We then had to use programmed empty and programmed full signals of FIFOs to transfer the data correctly.

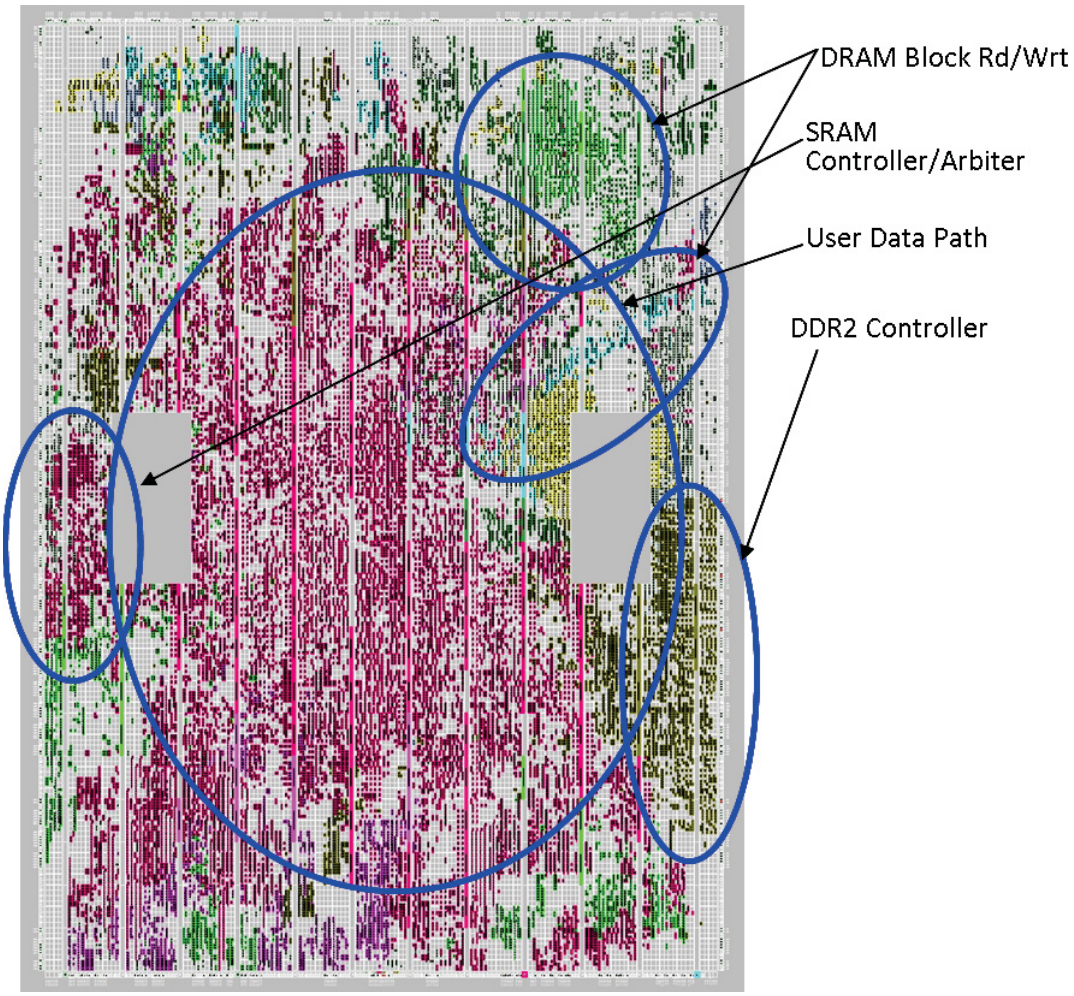

Figure 4.17: MF Router Virtex2Pro50 FloorPlan

4.3.3 The User Data Path Module

This design is based on the original NetFPGA DRAM Reference Router design [Reference Pipeline]. User Data Path (UDP) module is functional core of the router. Our design has five Rx FIFOs coming from the MAC Rx queues of 4 ports and 1 from DMA Rx. The first block they encounter is the Input Arbiter. As the name suggests, this modules selects from of the 4 Rx Queues and a DRAM Queue and connects the path to Output Port Lookup block. The DRAM queue is the Packet Circulator which we had discussed earlier in Chaper 3. Input Arbiter has a round-robin method for selecting from one of the inputs. Output Port Lookup module performs GUID and NA lookups and decide on the final fate of the packet. After looking up the output port, it amends

the Packet Header shown in figure 4.11, and sends packet to Output Queue Arbiter. Output Queue Arbiter is a module which sends simultaneously to all the output ports which have been selected for that packet. It's capability of sending into more that one output ports (duplicating packets) can be used for multicasting. For our bloom filter implementation, it causes the packet to go through correct output port as well as may cause it to be forwarded to an incorrect one due to false positives. Output Queue Arbiter gets the write data count from the Output queues and send/drops a packet on a particular queue based on its capacity to accept the packet. It does this by comparing write data count of the queue with the incoming packet length. Hence, either the packets are dropped or sent out, but output queue is never stalled. The DMA interface might be used to send control packets to host controller. Use of DMA channel is out of scope of Thesis.

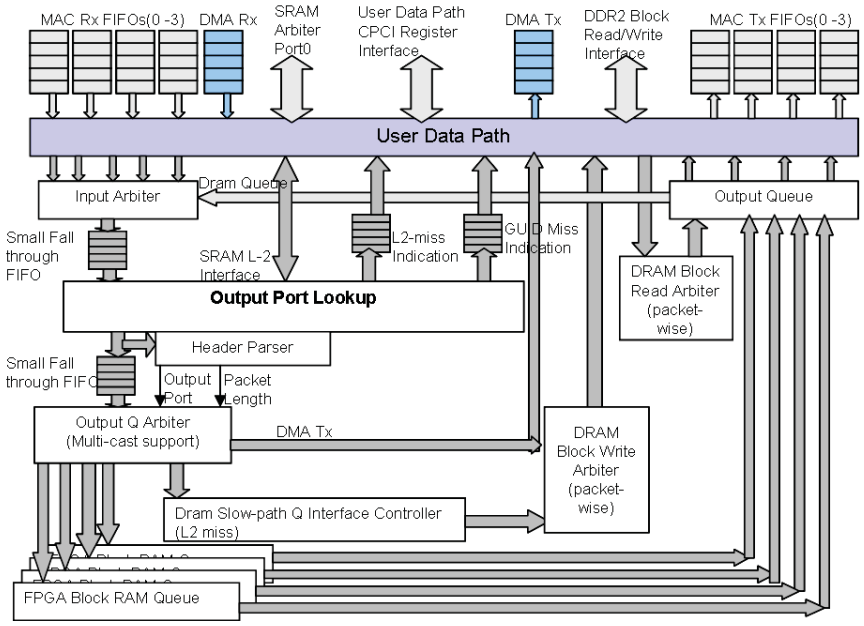

Figure 4.18: User Data Path Module

User Data Path Register Pipeline

This is NetFPGA Reference Router Register Pipeline. This has big advantage over the start topology of register connected to a single arbiter as in the Core Module Register Group. Register Pipeline allows easy insertion or deletion of modules into the pipeline.

The modules connected on the pipeline for Mobility First is as shown in figure 4.19.

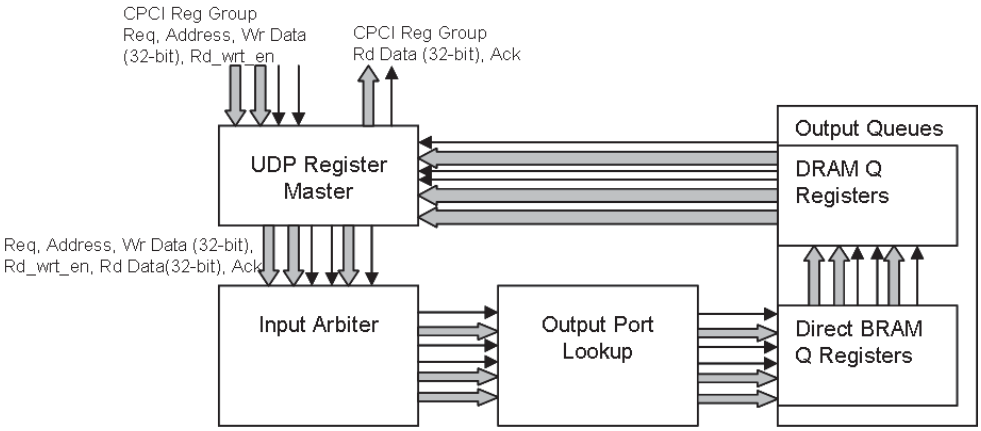

Figure 4.19: UDP Register Pipeline

4.3.4 Output Port Lookup Module

This module was re-designed specifically for MobilityFirst Router Implementation. Figure 4.20 shows the internal blocks of this module. It instantiates the NA and GUID lookup blocks. When a packet comes in from Input Arbiter, it first passes through a Packet FIFO. As the packet is being written in FIFO, Packet Pre-processor parses MobilityFirst packet header and indicates GUID and NA blocks about the 64-bit word they are supposed to extract their respective fields from. Both lookup blocks work in synchronism to decide final fate of packet. Once the output ports, MAC addresses or binding NA have been found, the information is released to Header Update block. This block updates output ports, MAC address and or NA, online while the packet is being sent out of the module. If the lookup activity does not complete untill the end of packet (EOP), input queue is stalled untill the output is resolved. This is done to keep NA and GUID lookup state-machines synchronized to the Start of Packet and End of Packet.

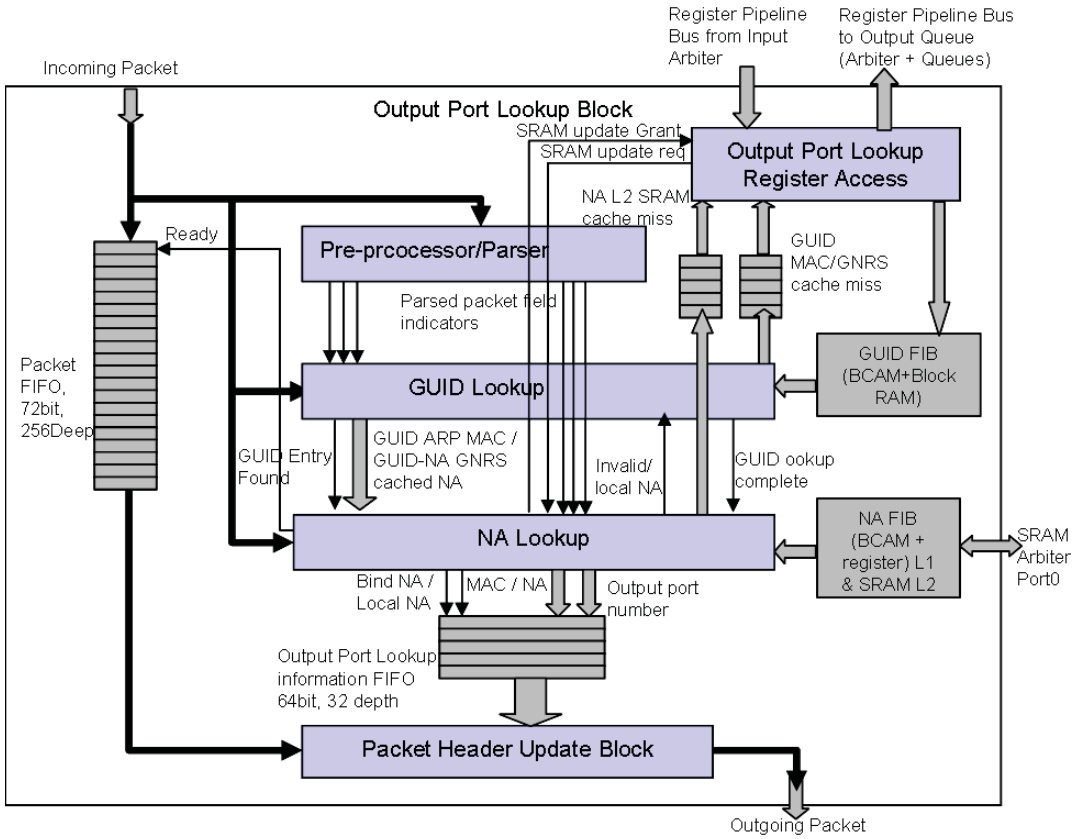

Figure 4.20: Output Port Lookup Module

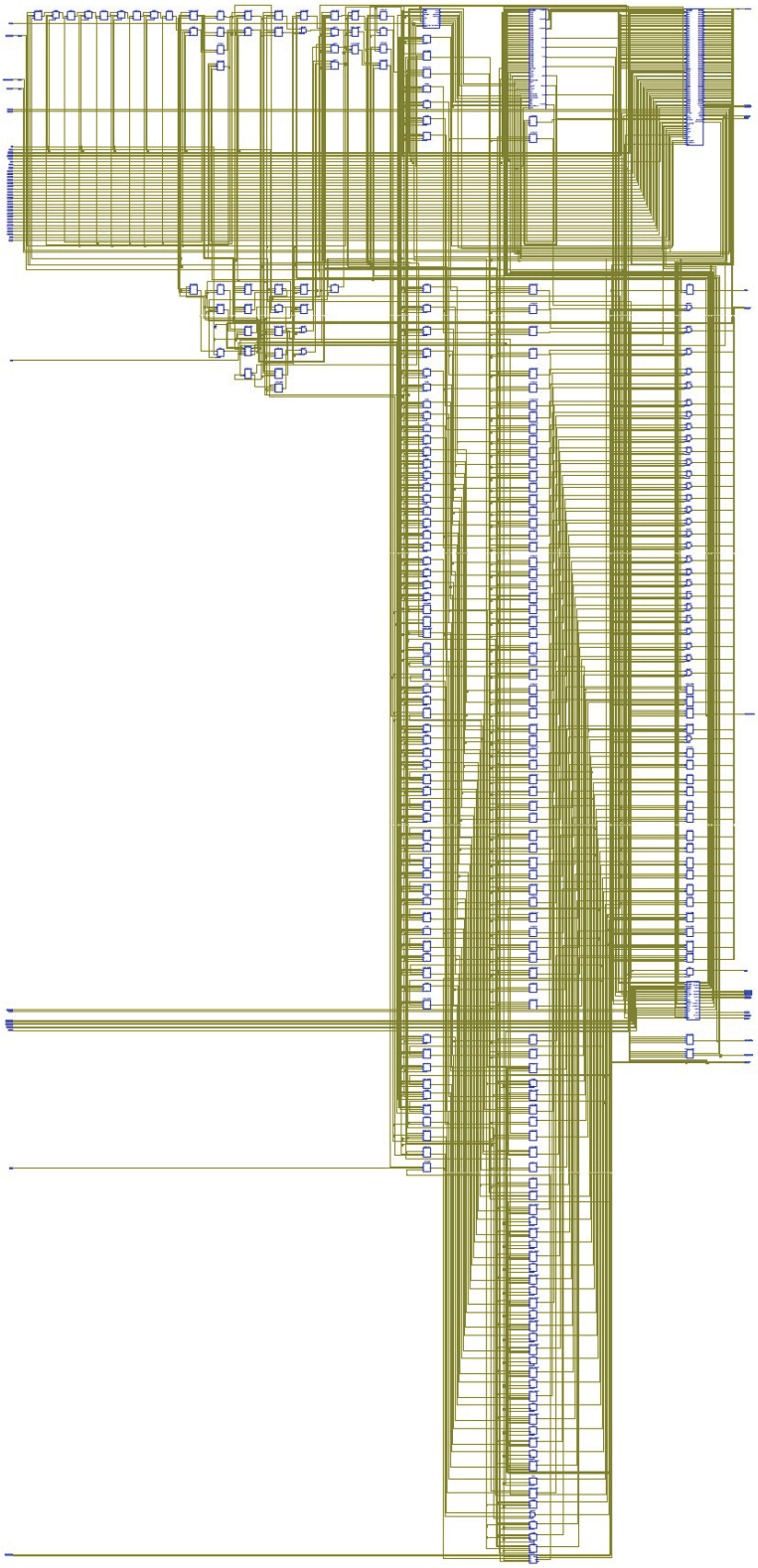

Figure 4.21: Output Port Lookup RTL Schematic

4.3.5 NA 2-Level Cache Output Port Lookup Module

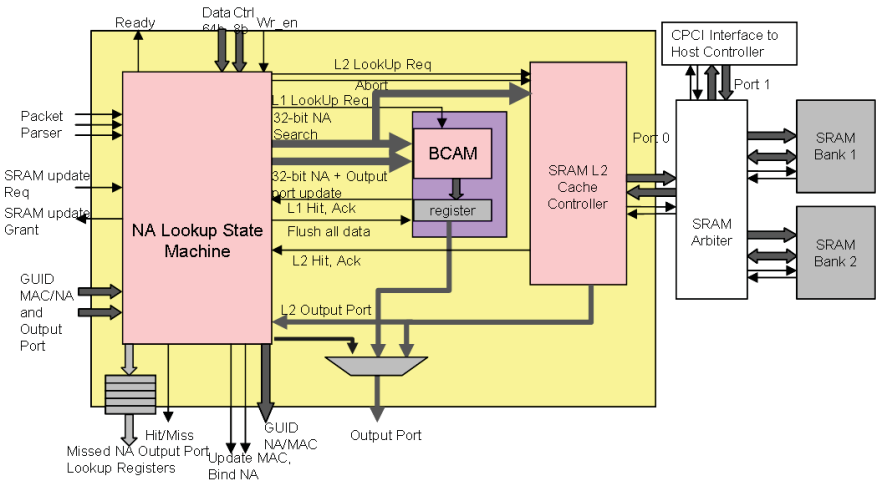

Figure 4.22: NA Lookup Module

NA lookup module starts a lookup cycle simultaneously on the Binary Content Addressable Memory(BCAM) (depth 32) and SRAM L2 Cache lookup. BCAM responds in 4 clock cycles (64ns) and is always faster than the L2 cache lookup. BCAM has un-encoded output which is encoded to address a register bank which holds output port information. We had proposed the 3 searching schemes for L2 Cache lookup:

Binary Search in Seperate Chaining Hash Table

In this scheme, 32-bit NA is hashed using straightforward 12-bit MSB extraction. This 12 bit address is looked-up in TLB section of SRAM. TLB contains physical address of the section of Hash table. The immediate next entry in TLB indicates start of Next section of Hash Table. The second address is read, and lowered by one value to get the High Address of the section (chain). Then a binary search is done within the section until either the NA is found or the section search is complete. Ack is returned when search is complete. Hit and Output Port are returned if NA is found. As it has been said earlier, SRAM data width is 32-bit and we need to store NA as well as output port in each location. Since the hashing method was direct bit extraction, we need only lower 20-bits of NA leaving 12 spare bits which is used for storing output port.

Linear Search Seperate Chaining Hash Table

This scheme is similar to the previous scheme with the only difference that instead of Binary search, a linear search is done to find NA within a section. Linear search in a small section proves to be much faster than Binary search because it can exploit the burst read mode of SRAM.

Counting Bloom Filters

In this scheme, 32-bit NA is hashed using 3 hash functions 3.3.5:

1. 21bit LSB of CCITT 32 bit hash
2. Combination of XOR and Bit extraction for 21bit Key
3. 21bit H3 Class hash function

The three hash addresses are read for their counts for the respective ports. A 4bit count is used to indicate presence of absence of NA in the bloom filter. All 4 Ports together require 16 bits which means we could use the 32 bit SRAM data to store two sets of 4 Bloom filters. This makes the capacity of SRAM to have a hash table of 2^{21}.

Coming back to NA lookup state-machine, if BCAM fails to find NA but the L2 cache finds it in SRAM, NA-Output Port combination is written in BCAM. The replacement policy for BCAM is First In First Out (FIFO). If NA is not found in L2 also, then the NA is inserted in NA Miss indication queue, and packet is sent to Packet Circulator. If the packet had its NA inserted in the Miss Indication queue previously, it is not inserted again to avoid duplicate requests to the host controller. This is checked by a bit in the 4-bit reserved for MobilityFirst Router information in NetFPGA Packet header as shown in figure 4.11.

Exclusive permission to Host Controller to Access SRAM

The Host controller wants to update SRAM lookup table, it has to first request to NA lookup block to do so. NA lookup block grants permission to host controller when it is not using L2 cache SRAM. NA lookup does not use L2 cache as long as Host does not

de-assert the request. During this time it only uses L1 cache to search NA and packets out of the 32 flows, will be sent to packet circulator. When Host de-asserts request, BCAM is flushed and for the time it is being flushed, only L2 cache is used to lookup NA. Once BCAM flush is complete, normal 2-level lookups are resumed.

Incase of local NA, NA lookup stops the lookup process as the output port is expected to be found by GUID lookup block. Incase of invalid NA (NULL value is 32bit 0), this module waits for GUID lookup to complete and get GNRS cached NA. After receiving NA from GUID GNRS cache, it starts a regular NA Lookup. If GUID lookup cannot find an entry, lookup is halted and packet is dropped.

Figure 4.23: NA Lookup RTL Schematic

4.3.6 GUID Lookup Module

4.3.7 GUID Lookup

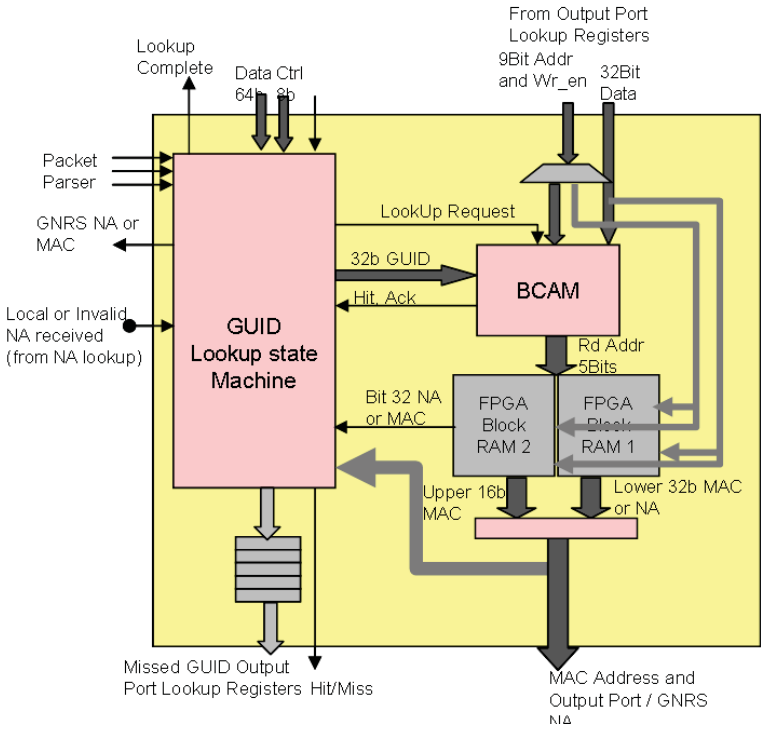

Figure 4.24: GUID Lookup Module

GUID Lookup starts two clock cycles ahead of NA lookup due to its place in MobilityFirst Packet packed in 64 Bit Data. It sends a search request to BCAM and gets a response in 5 clks, the extra last clock is to access FPGA block RAMs. There are two 32bit (depth 32) FPGA block RAMs which together hold either $1 + 48$ bits Flag and MAC address or $1 + 32$ bits Flag and GNRS cached NA. The flag bit distinguishes between the two types of entries. If there is a miss in the BCAM, GUID lookup waits for NA lookup to indicate whether the packet had a local NA or an Invalid NA, so that it can insert the missed GUID in missed Queue. Other GUIDs misses do not need to be added to the missed queues since they will not be serviced by this router. The Host controller has the CAM and BRAMs memory mapped into Output Port Lookup Register Space and can update the entries any time. To invalidate an entry, Host can write 32'h0 in GUID BCAM or replace it with other GUID.

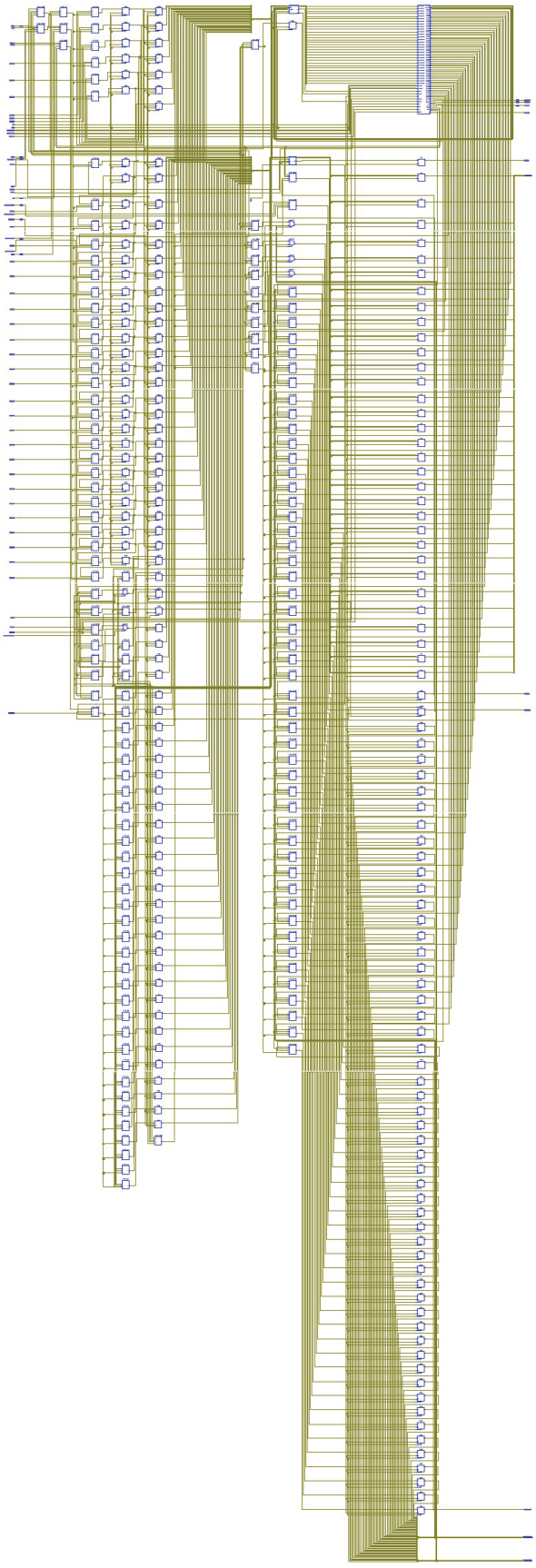

Figure 4.25: GUID Lookup RTL Schematic

4.3.8 Output Queues

This module instantiates the Header parser, Output Queue arbiter, Direct Port Queues (FPGA Block RAM) and DRAM queue (Packet Circulator implemented on DDR2 DRAM). As the name suggests, Header parser parses the header and sends length and output information to the Output Queue Arbiter. A small fifo is used to transfer header information to allow pipelining. Output Queue Arbiter, directs packets to all the output queues to which the packet should be send (this is possible since the output port information in packet header is one-hot encoded). Before transferring the data to respective output queues, OQ Arbiter checks for the free write space available in the Queue FIFOs. If the length of the packet is less than or equal to free space of the intended Output Queue FIFO, then write enable for that queue is enabled. If no Output Queue FIFOs have enough space, packet is dropped and the flow is never stalled.

Direct Queues have FIFO of 8KB implemented in the FPGA Block RAMs. They also have the MAC address of destination next hops. For inter-domain packets, the destination MAC address is updated into the packets at their respective Direct Queues. If however, the packets are intra-domain, there is a flag bit in the MobilityFirst Information bits of NetFPGA packet header, which indicates that the packet is intra-domain. When this intra-domain flag bit is set, destination MAC address is not updated by Direct Queues. Intra-domain flag bit is set by the Header Update Block in Output Port Lookup module, which updates the destination MAC address found in GUID-MAC lookup, for intra-domain packets.

DRAM Queues have a Write Block which has a 4KB FIFO (DRAM Write FIFO). This FIFO converts 72bit words to 144bit words before transferring the data to Block DRAM Read/Write Module. It also calculates and appends the length info header to the packets. The sencond Module is the DRAM Queues is a Read Block which has a 4KB FIFO which converts 144bit words to 72bit words before passing the packets to MAC Group Module. It removes the length info header from the packet while doing so.

Chapter 5

Results

5.1 Emulation Results

The figure 5.1 shows Emulation setup for MobilityFirst Prototypoe router. Using this setup we were able to perform various micro-benchmarking tests. The setup consists of two NetFPGA connected back to back. One acts as packet generator and checker, the other acts as the MobilityFirst Router. Packet generator is able to generate packets simultaneously on all 4 ports and perform the check on the received packets. There is an internal timer which starts on start of test and stops when no packet has been received for 3000000us. Hence the final time is calculated by subtracting 3000000 from the timer value. All configuration and status registers are accessible through PCI bus. Additonally, the packet generator has an option to read test scenarios from SRAM. For using SRAM, Test host controller should download the test in SRAM before starting the test. To test Bloom Filters, we need a random data set to evaluate false positives. For this reason, we generated random data sets using Matlab and transffered the data in SRAM. Most of the test aim at testing the NA lookup, which requires longer lookup times. Therefore, tester sends packets with 4 different GUIDs for 4 ports of the tester, and different NAs which are sent in a pattern to generate different scenarios.

Table 5.1 compares the linear and binary search schemes. The cross-over point is a little higher than the theoritical value of N = 51. This is because of the queuing delays we did not consider during the calculations. More over, since the NetFPGA does not have a large input queue, for testing purposes, we needed to send packets at a rate slightly lower than the average lookup rate to avoid packet dropping at input queues. We chose non-pipelined version of design get actual thoughput with respect to lookup delays.

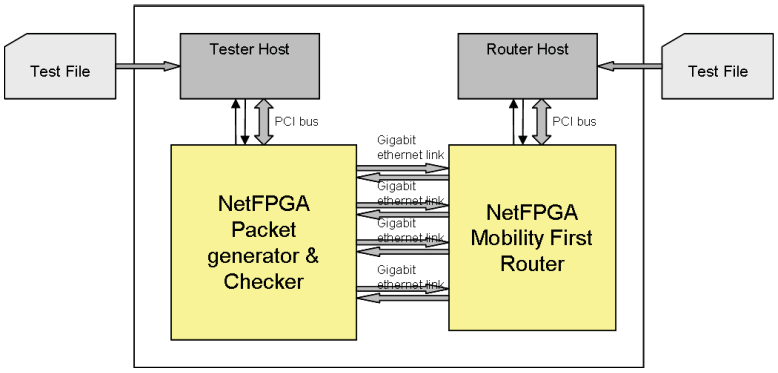

Figure 5.1: Emulation Setup

Packets Sent,Received	Packet Length	Size of Hash Chain	Binary Search Time(usec)	Linear Search Time(usec)	Binary Search Throughput(Mbps)	Linear Search Throughput(Mbps)
1000000	174	31	5008010	4144016	277.9547165	335.906039
1000000	174	63	5560010	5168030	250.3592619	269.3482816
1000000	174	127	6120011	7216067	227.4505716	192.9028652
1000000	174	255	6688011	11312187	208.133629	123.0531285

Table 5.1: Non-pipelined NA Lookup Performance Comparison of Linear and Binary Search for different Chain Lengths

Table 5.2 shows performace when we forced all packets to be looked up from L1 cache i.e. BCAM, only. We can observe that, due to pipelining, the throughput remains high irrespective of the packet length. The curve which we can see is due to the fixed delays which seperate the packets, and become insignificant as the packet length is increased.

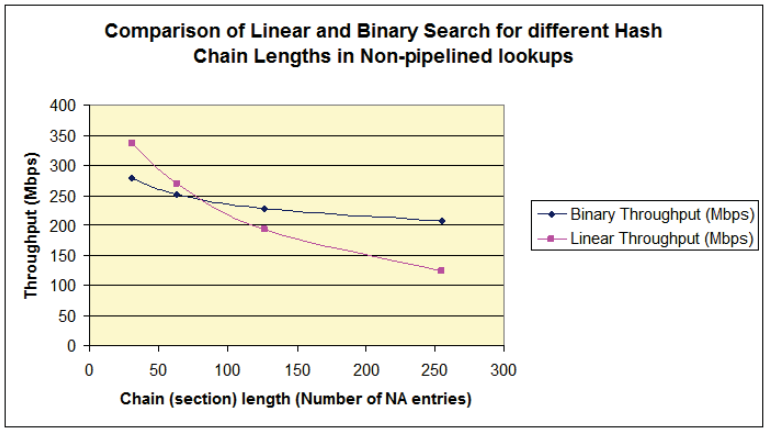

Figure 5.2: Non-Pipelined NA L2 cache Linear and Binary Search Scheme comparison

Packets Sent,Received	Packet Length	Test Time(usec)	Throughput(Mbps)	Net Throughput(Mbps)
1000000	174	1600019	869.9896689	3479.958675
1000000	500	4224044	946.9598328	3787.839331
1000000	1000	8208044	974.6536446	3898.614579
1000000	1500	12232064	981.0282222	3924.112889

Table 5.2: Pipelined NA L1 cache Lookup Performance

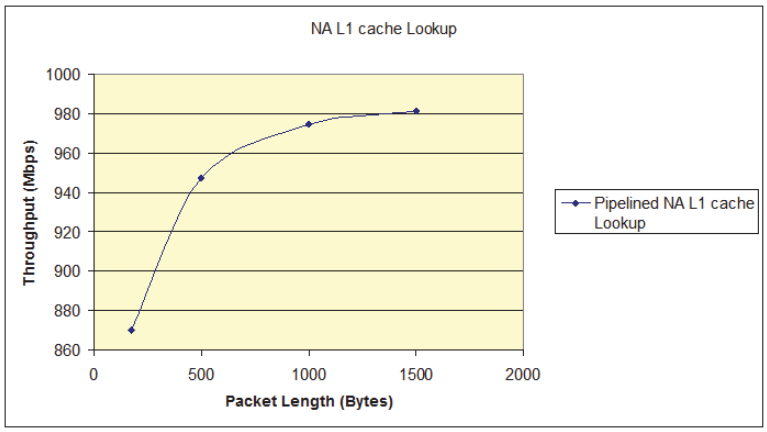

Figure 5.3: Pipelined NA L1 cache lookup Performance

Packets Sent,Received	Packet Length	Test Time(usec)	Throughput(Mbps)	Net Throughput(Mbps)
1000000	174	6064011	229.5510348	918.2041391
1000000	500	6088024	657.0276333	2628.110533
1000000	1000	8200085	975.5996432	3902.398573
1000000	1500	12232064	981.0282222	3924.112889

Table 5.3: Pipelined NA L2 Cache Binary Search Performance in Hash Chain length of 255

Table 5.3 shows performace when we forced all packets to be looked up from L2 cache only by generating packet stream which has zero locality. We sent address which spanned over the entire address space inserted in SRAM. Due to pipelining, longer packets do not get delayed even for the worst case in binary search in hash chain of 255.

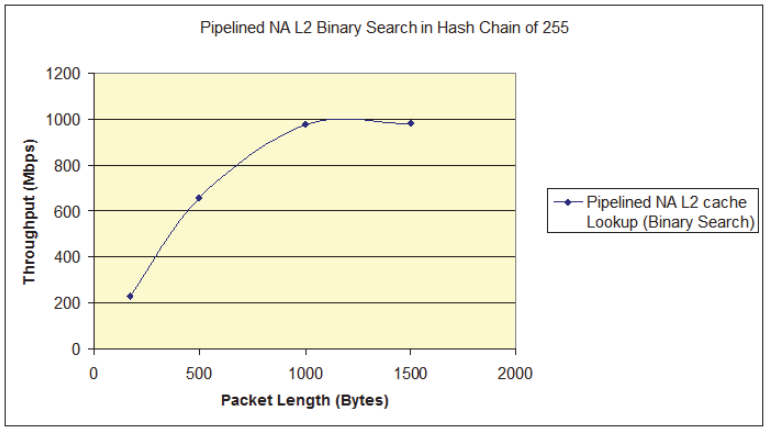

Figure 5.4: Pipelined NA L2 cache Binary Search in Hash Chain length of 255

Packets Sent,Received	Packet Length	Test Time(usec)	Throughput(Mbps)	Net Throughput(Mbps)
1000000	174	167781	869.9408872	3479.763549
1000000	500	444610	943.3526012	3773.410405
1000000	1000	866602	967.9737642	3871.895057
1000000	1500	1291041	974.618157	3898.472628

Table 5.4: Pipelined NA L2 Cache Bloom Filter Search Performance

Table 5.4 shows performace Bloom Filter L2 cache lookup scheme. Our bloom filter implementation requires fixed amount of delay which is smaller than the length (time) of the smallest packet. For the test, we generated multiple sets of random numbers using Matlab which were then written in tester and router's SRAMs respectively.

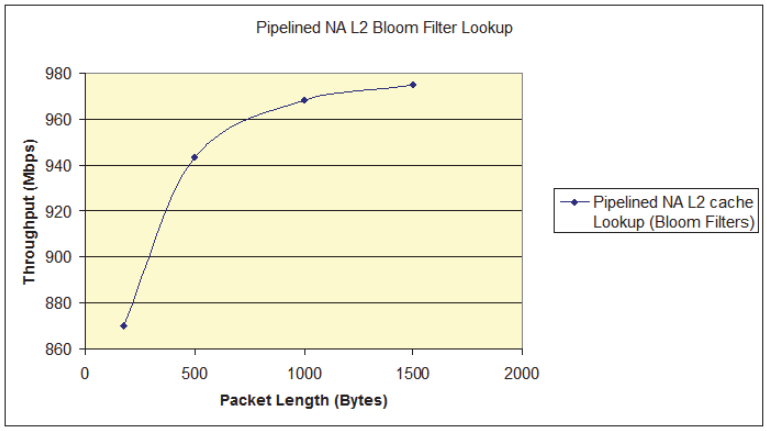

Figure 5.5: Pipelined NA L2 cache Bloom Filter Search Performance

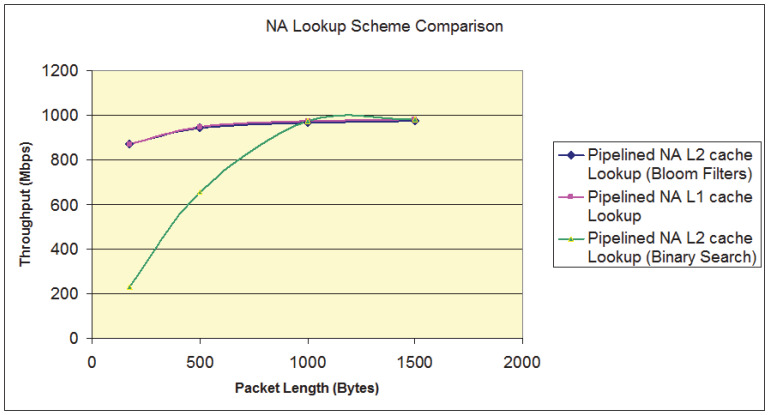

Figure 5.6: Performance Comparison between NA lookup schemes

Port 0 Test, Number of packets sent 262143

Port Number	Num packets expected	Num of packets received	Num False Positive packets	False Positive Ratio
0	262143	262143	0	0
1	0	32637	32637	0.12450075
2	0	32793	32793	0.125095845
3	0	32714	32714	0.124794482

Port 1 Test, Number of packets sent 262143

Port Number	Num packets expected	Num of packets received	Num False Positive packets	False Positive Ratio
0	0	32770	32770	0.125008106
1	262143	262143	0	0
2	0	32905	32905	0.125523092
3	0	32861	32861	0.125355245

Port 2 Test, Number of packets sent 262143

Port Number	Num packets expected	Num of packets received	Num False Positive packets	False Positive Ratio
0	0	32735	32735	0.124874591
1	0	32661	32661	0.124592303
2	262143	262143	0	0
3	0	32794	32794	0.125099659

Port 3 Test, Number of packets sent 262143

Port Number	Num packets expected	Num of packets received	Num False Positive packets	False Positive Ratio
0	0	32700	32700	0.124741076
1	0	32791	32791	0.125088215
2	0	32764	32764	0.124985218
3	262143	262143	0	0

Table 5.5: Bloom Filter False Positives for Load Factor (n/m) = 0.231

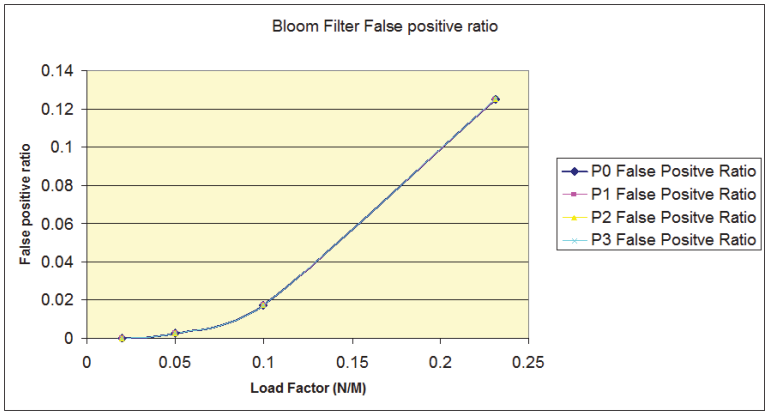

Figure 5.7: Bloom Filter False Postive Ratios

Port 0 Test, Number of packets sent 209714

Port Number	Num packets expected	Num of packets received	Num False Positive packets	False Positive Ratio
0	209714	209714	0	0
1	0	3660	3660	0.01745234
2	0	3667	3667	0.017485719
3	0	3664	3664	0.017471413

Port 1 Test, Number of packets sent 209714

Port Number	Num packets expected	Num of packets received	Num False Positive packets	False Positive Ratio
0	0	3688	3688	0.017585855
1	209714	209714	0	0
2	0	3650	3650	0.017404656
3	0	3653	3653	0.017418961

Port 2 Test, Number of packets sent 209714

Port Number	Num packets expected	Num of packets received	Num False Positive packets	False Positive Ratio
0	0	3644	3644	0.017376045
1	0	3660	3660	0.01745234
2	209714	209714	0	0
3	0	3680	3680	0.017547708

Port 3 Test, Number of packets sent 209714

Port Number	Num packets expected	Num of packets received	Num False Positive packets	False Positive Ratio
0	0	3614	3614	0.017232994
1	0	3658	3658	0.017442803
2	0	3661	3661	0.017457108
3	209714	209714	0	0

Table 5.6: Bloom Filter False Positives for Load Factor (n/m) = 0.1

Port 0 Test, Number of packets sent 104856

Port Number	Num packets expected	Num of packets received	Num False Positive packets	False Positive Ratio
0	104856	104856	0	0
1	0	293	293	0.002794308
2	0	292	292	0.002784771
3	0	290	290	0.002765698

Port 1 Test, Number of packets sent 104856

Port Number	Num packets expected	Num of packets received	Num False Positive packets	False Positive Ratio
0	0	286	286	0.00272755
1	104856	104856	0	0
2	0	303	303	0.002889677
3	0	278	278	0.002651255

Port 2 Test, Number of packets sent 104856

Port Number	Num packets expected	Num of packets received	Num False Positive packets	False Positive Ratio
0	0	288	288	0.002746624
1	0	305	305	0.002908751
2	104856	104856	0	0
3	0	291	291	0.002775235

Port 3 Test, Number of packets sent 104856

Port Number	Num packets expected	Num of packets received	Num False Positive packets	False Positive Ratio
0	0	288	288	0.002746624
1	0	270	270	0.00257496
2	0	286	286	0.00272755
3	104856	104856	0	0

Table 5.7: Bloom Filter False Positives for Load Factor (n/m) = 0.05

Port 0 Test, Number of packets sent 41942

Port Number	Num packets expected	Num of packets received	Num False Positive packets	False Positive Ratio
0	41942	41942	0	0
1	0	8	8	0.00019074
2	0	8	8	0.00019074
3	0	9	9	0.000214582

Port 1 Test, Number of packets sent 41942

Port Number	Num packets expected	Num of packets received	Num False Positive packets	False Positive Ratio
0	0	8	8	0.00019074
1	41942	41942	0	0
2	0	10	10	0.000238424
3	0	7	7	0.000166897

Port 2 Test, Number of packets sent 41942

Port Number	Num packets expected	Num of packets received	Num False Positive packets	False Positive Ratio
0	0	10	10	0.000238424
1	0	10	10	0.000238424
2	41942	41942	0	0
3	0	9	9	0.000214582

Port 3 Test, Number of packets sent 41942

Port Number	Num packets expected	Num of packets received	Num False Positive packets	False Positive Ratio
0	0	10	10	0.000238424
1	0	9	9	0.000214582
2	0	8	8	0.00019074
3	41942	41942	0	0

Table 5.8: Bloom Filter False Positives for Load Factor (n/m) = 0.02

Load factor (n/m)	Port 0	Port 1	Port 2	Port 3
0.231	0.124874591	0.124727089	0.125201385	0.125083129
0.1	0.017398298	0.017449161	0.017449161	0.017479361
0.05	0.002740266	0.00275934	0.002800666	0.002730729
0.02	0.00022253	0.000214582	0.000206635	0.000198687

Table 5.9: Avergae False positive ratio per port for different Load Factors

Packets Sent,Received	Packet Length	Test Time(usec)	Throughput(Mbps)	Net Throughput(Mbps)
1000000	174	1632018	852.9317691	3411.727077
1000000	500	4224043	946.960057	3787.840228
1000000	1000	8200083	975.5998811	3902.399524
1000000	1500	12232064	981.0282222	3924.112889

Table 5.10: GUID-NA binding and Forwarding

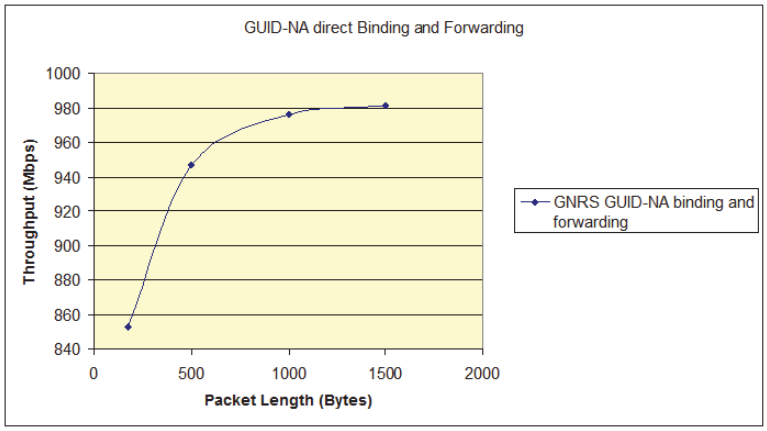

Figure 5.8: GUID-NA binding and Forwarding

Packets Sent,Received	Packet Length	Test Time(usec)	Throughput(Mbps)	Net Throughput(Mbps)
1000000	174	1600018	869.9902126	3479.96085
1000000	500	4224044	946.9598328	3787.839331
1000000	1000	8208044	974.6536446	3898.614579
1000000	1500	12232063	981.0283024	3924.11321

Table 5.11: GUID-MAC Lookup, intra-domain forwarding

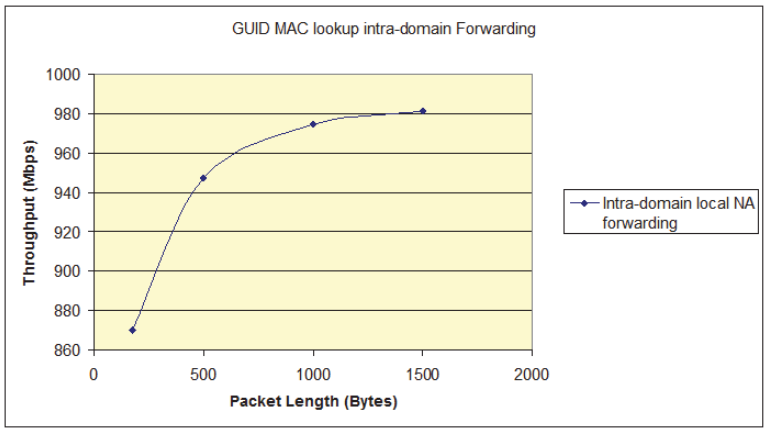

Figure 5.9: GUID-MAC Lookup, intra-domain forwarding

5.2 Scalibility and Hit ratios

The Host controller memory which has the L3 cache, does not have any limit and hence is considered to be infinite. We have made the provision for the host controller to receive L2 miss indications from router. In addition, the packets which need L3 lookups are queued in the packet circulator DDR2 buffer. We have 64MB of DRAM, and the block size is of 2K, enabling us to buffer upto 32K packets. We conducted a small experiment where 4 packets containing NA which are not present in L2 cache are sent. For the purposes of evaluating router's performance alone, host controller has a set of these 4 NAs readily available for L2 update. In the experiment, host waits for 4 L2 miss indications and then updates the SRAM with 4 NAs, and packets are then sent out back to the tester. The time required to do so was 96us for 4 packets of length 500Bytes. Then we generated the same scenario with NA already present in L2 cache giving us a round-trip time of 32us. Hence, we could roughly estimate that L3 cache access delay is 64us i.e. 16us per NA. We could expect a similar result for a GUID miss updation as well.

Hence, the L1 cache lookup time is 64ns, L2 cache(depending on the scheme) lookup time ranges from 304nsec to 1350ns and L3 cache lookup time is more than or equal to 16000ns.

Depending upon the scale and hit ratio we would have to increase the size of L1 and L2 caches to maintain the performance. The pipelined design allows us to have 100 percent L2 hit ratio and still achive line rate above a certain threshold.

Let us assume that a certain traffic generates a L1HitRatio, L2HitRatio and L3Hitratio, then total time required to forward these packets (in ns) would be

Maximum of

$T_{AvgRoute} = ((L1HitRatio * 64) + (L2HitRatio * 1350) + (L3HitRatio * 16000))$ and

$T_{AvgRoute} = AvgPacketLength * 8/4$ for 4 ports simultaneous.

We have seen that L2 Cache based lookup schemes can perform line-speed forwarding, for simplicity let's assume, packet length's are always 1500Bytes. The $T_{AvgRoute} = 1500 * 2 = 3000ns$. In addition, $L1HitRatio = 0$. Then to maintain line speed lookups $L3HitRatio = (3000 - (L2HitRatio * 1350))/16000$. For Binary (exact match) search, max number of NA in L2 cache $= 1020 * 1024 = 1044480$, and let total number of NA in MF be Num_{NA}, then $L2HitRatio = 1044480/Num_{NA}$ and $L3HitRatio = 1 - L2HitRatio$.

Therefore, $(1 - L2HitRatio) = (3000 - (L2HitRatio * 1350))/16000$

$L2HitRatio = 0.8873$

$Num_{NA} = 1177049$ values.

Hence to support an address space of $Num_{NA} = 100000000$, and maintain same $L2HitRatio = 0.8873$, L2cache should contain 88737201 NA entries which would require SRAM size of $(1024/1020) * 88737201 = 89.085MWords$ (32-bit word). This is how the memory would scale.

Similarly, for Bloom filters,

$(1 - L2HitRatio) = (3000 - (L2HitRatio * 304))/16000$

$L2HitRatio = 0.8282$

$Num_{NA} = (208822 * 4)/L2HitRatio = 1008514$ values.

Hence to support an address space of $Num_{NA} = 100000000$, number of NAs in Bloom Filters should be 82823649 and size of SRAM required would be $((82823649/4) * 10.0427)/2 = 103971633MWords$ for 4 port router.

5.3 Simulation Results

Figure shows the simulation environment for MobilityFirst Router. We used the ISIM which is built-in simulator in Xilinx ISE, to simulate the design. We could easily use ModelSim to test the design as well, we just need to generate libraries for different Xilinx IP cores which have been used in the design.

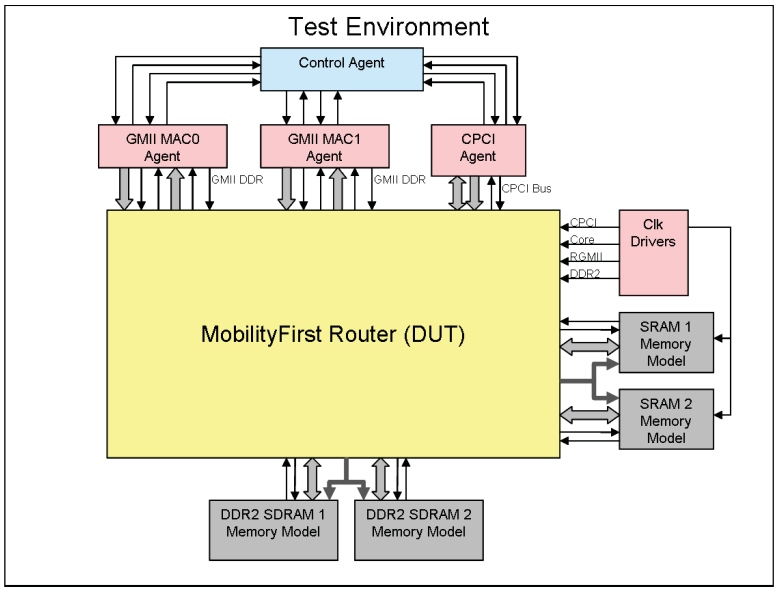

Figure 5.10: Simulation Environment

The number of tests performed to get the RTL design to do the right thing are very large. Hence, we have included only a few important and relevent block level simulation results here.

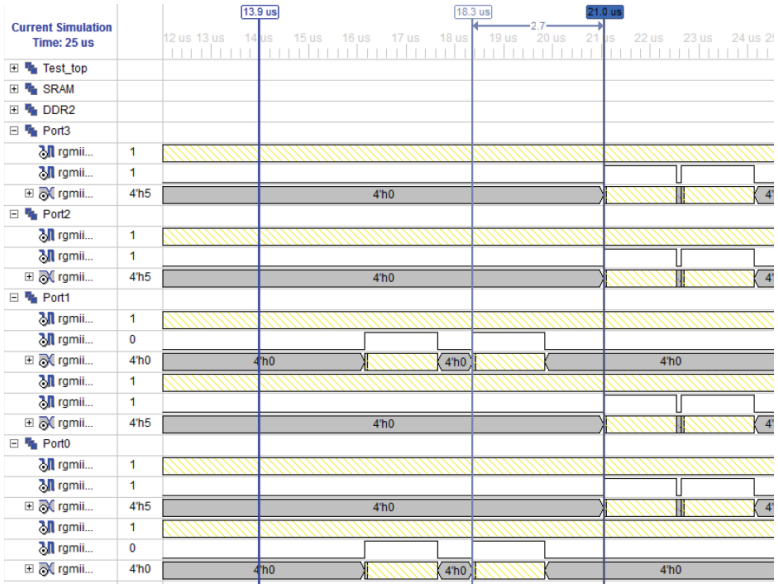

Figure 5.11: Packet 1 on Port0 and Port1 dropped and Packet 2 sent out on all ports

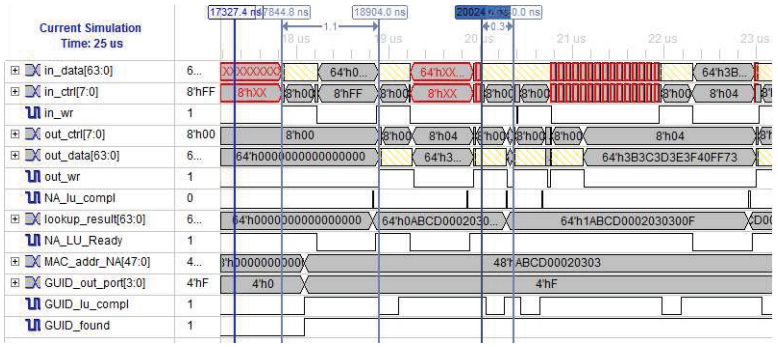

Figure 5.12: First 2 Packects dropped after lookup miss, followed by GUID MAC translation of 2 Packets

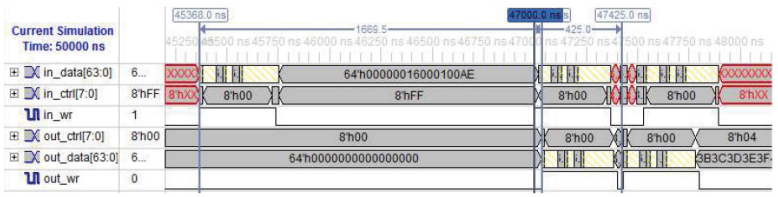

Figure 5.13: NA L2 Cache lookup followed by L1 Cache hit

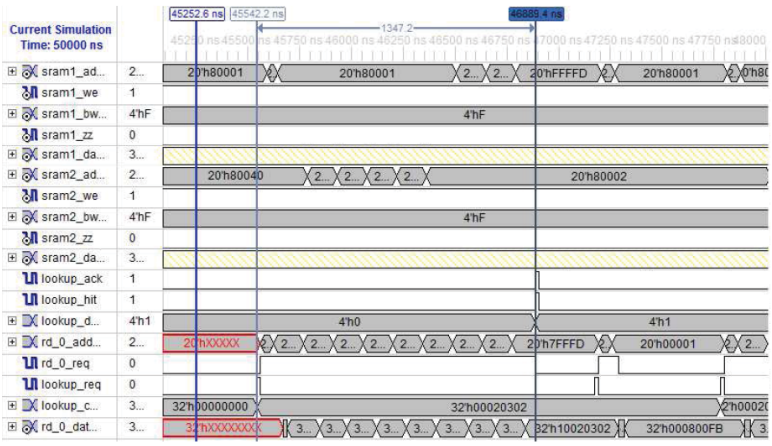

Figure 5.14: L2 Cache SRAM lookup

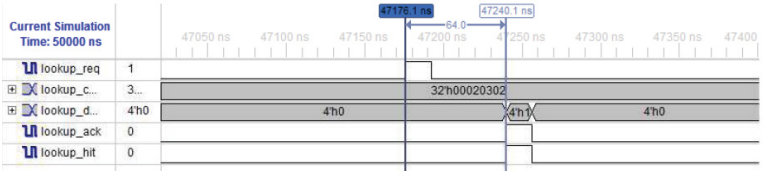

Figure 5.15: L1 Cache BCAM lookup

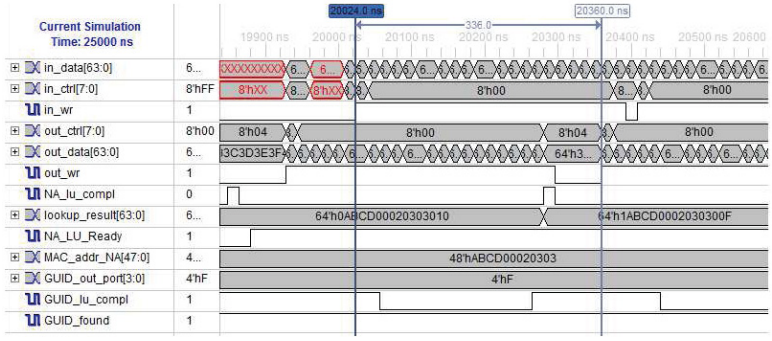

Figure 5.16: GUID MAC translation

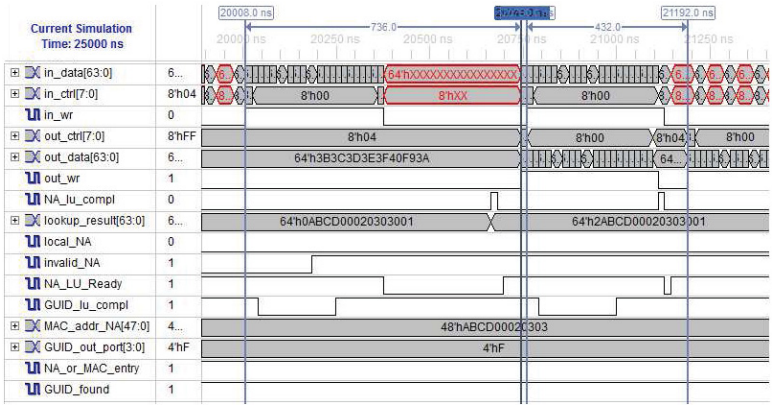

Figure 5.17: GUID GNRS NA cache lookup

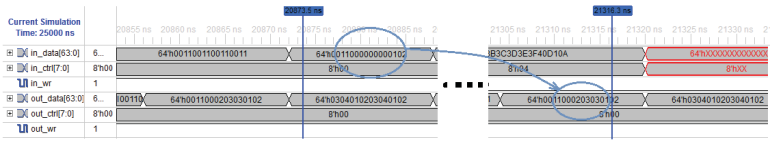

Figure 5.18: Linespeed NA Binding

Chapter 6

Conclusion and Future Work

Based on the experimental results, the MobilityFirst Router performs best for the longest packets lengths for all scenarios. Our Bloom Filter based design has highest lookup speed and for a resonable tradeoff for memory, it can yield a very low false positive ratio (0.0002 for load factor of 0.02). MobilityFirst Router would work best for large data Flows between the routers which matches with the design of chunk based data transfers at higher layers of MobilityFirst Protocol Stack. Our router can work as both an edge-router and a back-bone router, however it is better suited for backbone applications due to the large NA FIB it can hold.

6.1 Future Work

The MobilityFirst Router Prototype brings many opportunities to expand the design and explore new possibilities. We want to evaluate the MF router performance on a real-world traces. One of the features we are interested in adding is the Multi-cast packet forwarding which would save the network from huge amounts of duplicate data packets. The Bloomfilters with more data specific hash functions may be evaluated. Both NA and GUID lookups can be fully pipelined with careful designing which would increase the throughputs even for the smaller packets. Support for jumbo frames is also in the list since the overhead of MobilityFirst packet processing would then be smaller due to less frequent lookup operations. We have included a single DMA channel and would like to evaluate its performance to handle Control Packets. NetFPGA 10G is the next generation platform from Stanford University, which has an advanced FPGA Virtex5, larger on-board memory, advanced peripherals and better tool chain support from Xilinx.

References

[1] C. Perkins, "Mobile networking through mobile ip," *Internet Computing, IEEE*, vol. 2, no. 1, pp. 58–69, 1998.

[2] http://www.nets-fia.net/.

[3] I. Seskar, K. Nagaraja, S. Nelson, and D. Raychaudhuri, "Mobilityfirst future internet architecture project," in *Proceedings of the 7th Asian Internet Engineering Conference*, ser. AINTEC '11. New York, NY, USA: ACM, 2011, pp. 1–3. [Online]. Available: http://doi.acm.org/10.1145/2089016.2089017

[4] D. Raychaudhuri, K. Nagaraja, and A. Venkataramani, "Mobilityfirst: a robust and trustworthy mobility-centric architecture for the future internet," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 16, no. 3, pp. 2–13, Dec. 2012. [Online]. Available: http://doi.acm.org.proxy.libraries.rutgers.edu/10.1145/2412096.2412098

[5] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (cam) circuits and architectures: A tutorial and survey," *Solid-State Circuits, IEEE Journal of*, vol. 41, no. 3, pp. 712–727, 2006.

[6] R. A. Kempke and A. J. McAuley, "Ternary cam memory architecture and methodology," Nov. 24 1998, uS Patent 5,841,874.

[7] G. Varghese, *Network algorithmics*. Chapman & Hall/CRC, 2010.

[8] S. K. Maurya and L. T. Clark, "A dynamic longest prefix matching content addressable memory for ip routing," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, no. 6, pp. 963–972, 2011.

[9] Y. Sun, N. Egi, G. Shi, and J. Wu, "Content-based route lookup using cams."

[10] C. Kim, M. Caesar, A. Gerber, and J. Rexford, "Revisiting route caching: The world should be flat," in *Passive and Active Network Measurement*. Springer, 2009, pp. 3–12.

[11] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2004.

[12] Y. Qiao, T. Li, and S. Chen, "One memory access bloom filters and their generalization," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 1745–1753.

[13] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo, "Netfpga–an open platform for gigabit-rate network switching and routing," in *Microelectronic Systems Education, 2007. MSE'07. IEEE International Conference on*. IEEE, 2007, pp. 160–161.

[14] S. J. Xilinx, "Ca,virtexe, virtex2, virtex2pro, and spartan3 datasheets, 2006."

[15] "Ieee standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks - specific requirements. supplement to carrier sense multiple access with collision detection (csma/cd) access method and physical layer specifications. frame extensions for virtual bridged local area network (vlan) tagging on 802.3 networks," *IEEE Std 802.3ac-1998*, pp. i–, 1998.

[16] M. Ramakrishna, E. Fu, and E. Bahcekapili, "Efficient hardware hashing functions for high performance computers," *Computers, IEEE Transactions on*, vol. 46, no. 12, pp. 1378–1381, 1997.

[17] J. Xu and M. Singhal, "Cost-effective flow table designs for high-speed routers: architecture and performance evaluation," *Computers, IEEE Transactions on*, vol. 51, no. 9, pp. 1089–1099, 2002.

[18] M. Yu, A. Fabrikant, and J. Rexford, "Buffalo: bloom filter forwarding architecture for large organizations," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 2009, pp. 313–324.

[19] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood, "Fast hash table lookup using extended bloom filter: an aid to network processing," in *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4. ACM, 2005, pp. 181–192.

[20] J. Naous, G. Gibb, S. Bolouki, and N. McKeown, "Netfpga: reusable router architecture for experimental research," in *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*, ser. PRESTO '08. New York, NY, USA: ACM, 2008, pp. 1–7. [Online]. Available: http://doi.acm.org.proxy.libraries.rutgers.edu/10.1145/1397718.1397720

[21] http://wiki.netfpga.org/NetFPGA_PCB_r3.pdf.