Report Number: WUCS-01-11

2001-01-01

OBIWAN - An Internet Protocol Router in Reconfigurable Hardware

Florian Braun, Marcel Waldvogel, and John Lockwood

The ongoing exponential increase of line speed in the Internet and combined with the uncountable requests for increased functionality of network devices presents a major challenge. These demands call for the use of reprogrammable hardware to provide the required flexible high-speed functionaltiy. The Field Programmable Port Extender (FPX) provides such an environment for development of networking components in reprogrammable hardware. We present the high-speed IP routing components in reprogrammable hardware. We present the high-speed IP routing module "OBIWAN" (Optimal Binary search IP lookup for Wide Area Networks) built on top of an IP processing framework.

... Read complete abstract on page 2.

OBIWAN - An Internet Protocol Router in Reconfigurable Hardware

Florian Braun, Marcel Waldvogel, and John Lockwood

Complete Abstract:

The ongoing exponential increase of line speed in the Internet and combined with the uncountable requests for increased functionality of network devices presents a major challenge. These demands call for the use of reprogrammable hardware to provide the required flexible high-speed functionaltiy. The Field Programmable Port Extender (FPX) provides such an environment for development of networking components in reprogrammable hardware. We present the high-speed IP routing components in reprogrammable hardware. We present the high-speed IP routing module "OBIWAN" (Optimal Binary search IP lookup for Wide Area Networks) built on top of an IP processing framework.

OBIWAN – An Internet Protocol Router in Reconfigurable Hardware

Florian Braun, Marcel Waldvogel and
John Lockwood

WUCS-01-11

July 2001

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130

OBIWAN - an Internet Protocol Router in Reconfigurable Hardware

Florian Braun Marcel Waldvogel John Lockwood
Technical Report WU-CS-01-11
Applied Research Laboratory*
Washington University in St. Louis

July 29, 2001

Abstract

The ongoing exponential increase of line speed in the Internet and combined with the uncountable requests for increased functionality of network devices presents a major challenge. These demands call for the use of reprogrammable hardware to provide the required flexible high-speed functionality. The Field Programmable Port Extender (FPX) provides such an environment for development of networking components in reprogrammable hardware. We present the high-speed IP routing module "OBIWAN" (Optimal Binary search IP lookup for Wide Area Networks) built on top of an IP processing framework.

1 Introduction

In recent years, field programmable logic has become sufficiently capable to implement complex networking applications directly in hardware. The Field Programmable Port Extender has been implemented as a flexible platform for the processing of network data in hardware at multiple layers of the protocol stack. An important application for the network layer is routing and forwarding of Internet protocol (IP) packets to other network nodes.

2 Background

In the Applied Research Lab at Washington University in St. Louis, a rich set of hardware components and software for research in the field of ATM and active networking has been developed. The modules described in this document are primarily targeted to this kit, though the design is written in portable VHDL and could be used in any FPGA-based system.

*This research was supported in part by NSF ANI-0096052 and Xilinx Corp.

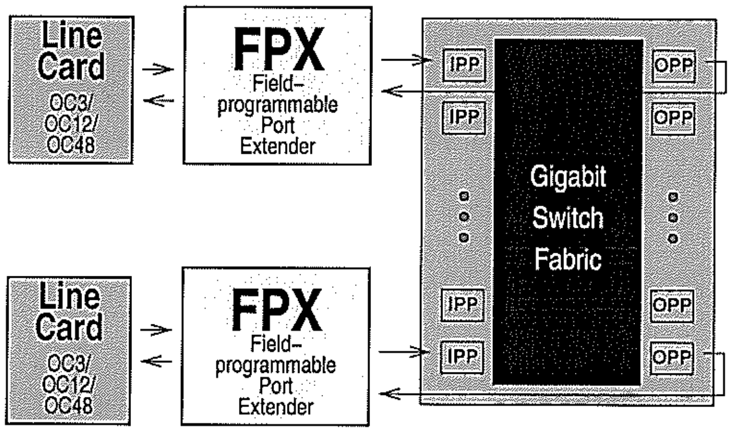


Figure 1: WUGS configuration using the Field Programmable Port Extender

2.1 Switch Fabric

The central component of this research environment is the Washington University Gigabit Switch (WUGS, [1]). It is a fully featured 8-port ATM switch, which is capable of handling up to 20 Gbps of network traffic. Each port is connected through a line card to the switch. The WUGS provides space to insert extension cards between the line cards and the switch itself.

2.2 Field Programmable Port Extender

The Field Programmable Port Extender (FPX, [2, 3]) provides reprogrammable logic for user applications. A configuration of the switch and the FPX is illustrated in Figure 1.

The FPX contains two FPGAs: the Network Interface Device (NID) and the Reprogrammable Application Device (RAD). The NID interconnects the WUGS, the line card and the RAD via a small switch. It also provides the logic to dynamically reprogram the RAD. The RAD can be programmed to hold user-defined modules. Hardware based processing of networking data is made possible that way. The RAD is also connected to two SRAM and two SDRAM components. The memory modules can be used to cache cell data or hold large tables. Figure 2 illustrates the major components on an FPX board.

2.3 FPX Modules

User applications are implemented on the RAD as modules. Modules are hardware components with a well-defined interface which communicate with the RAD and other infrastructure components. The basic data interface is a 32-bit wide, Utopia-like interface. The data bus carries ATM header information, as well as the payload of the

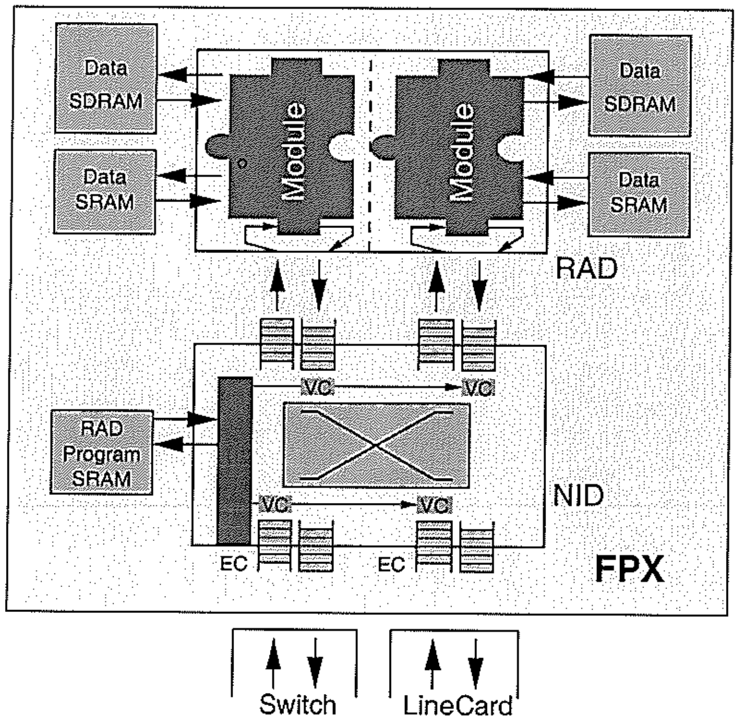


Figure 2: Components on an FPX board

cells. The other signals in the module interface are used for congestion control and to connect to memory controllers to access the off-chip memory. The complete module interface is documented in [4].

Usually, two application modules are present on the RAD. Typically, one handles data from the line card to the switch (ingress) and the other handles data from the switch to the line card (egress). Modules can be replaced by reprogramming the FPGA in the system at any time. In the case of the FPX, this functionality occurs via partial reprogramming of the RAD FPGA. A reconfiguration component performs a handshaking protocol with the modules to prevent loss of data.

2.4 Network Wrappers

Network protocols are organized in layers. Components have been developed for the FPX that allow applications to handle data on different levels of abstraction. A similar implementation exists for IP over Ethernet and the corresponding network layers [5]. On the cell level, a Start of Cell (SOC) signal is given to an application module. For AAL5 frame based applications, Start-of-Frame (SOF) and End-of-Frame (EOF) signals indicate the beginning or the end of an AAL5 frame, respectively. An additional data-enable signal indicates whether valid payload data is being sent. On the IP levels, a signal indicates the start of user payload data.

For each layer one component exists, which wraps up the application itself. Therefore we will refer to the surrounding components as *wrappers*. All wrappers are discussed in detail in [6].

2.4.1 Cell based processing

At the lowest level of abstraction, data is sent in fixed length cells. Applications or wrappers working on that protocol level typically process the ATM header and filter cells by their virtual channel. FPX Modules communicate with software via control cells, ATM cells with a well-defined structure used to perform remote configuration.

The wrapper on the lowest level is the cell processor. It performs every necessary step on the cell level that is common to all FPX modules. First of all, incoming ATM cells are checked against their Header Error Control (HEC) field, which is part of the 5 octet header. An 8 bit CRC is used to prevent errored cells from being misrouted. If the check fails, the cell is dropped.

Accepted cells are queried about their virtual channel information in the next step. The cell processor distinguishes between three different flows:

1. The cell is on the data VC for this module. In this case, the cell will be forwarded to the inner interface of the wrapper and thus to the application.

2. The cell is on the control cell VC and is tagged with the correct module ID. Control cells are processed by the cell processor itself.

3. None of the above, i.e. this cell is not destined for this module. These cells are bypassed and take a shortcut to the output of the cell processor.

2.4.2 Frame based processing

To handle data with arbitrary length over ATM networks, data is organized in frames, which are sent as multiple cells. The most commonly used adaption layer is the ATM Adaption Layer 5 (AAL5, [7, 8]).

The frame processor is a wrapper module for the FPX to handle AAL5 frame data. Its interface is designed to give application modules a more abstract view of the data. The frame processor replaces the Start-of-Cell signal with three signals, namely Start-of-Frame (SOF), End-of-Frame (EOF) and Data-Enable (DataEn) and handles the frame's CRC-32.

2.4.3 IP Packet Processing

The IP processor was developed to support IP based applications. It inherits the signalling interface from the frame processor and adds a Start-of-Payload (SOP) signal, to indicate the payload after the IP header. This wrapper serves two primary functions:

1. Checking the IP header integrity, i.e., the correctness of the header checksum. Forwarding of corrupted packets is suppressed by not propagating appropriate signals to the application.

2. Decrementing the Time To Live (TTL) field. As of RFC 1812 [9] all IP processing entities are required to decrement this field. Once this field reaches zero, the packet should not be forwarded any more. This is to prevent packets from looping around in networks due to mis-configured routers.

2.4.4 The FPX UDP Processor

The UDP processor is a wrapper to support user datagrams over IP. This wrapper takes care of the UDP checksum and the length field in the header for outgoing datagrams. Incoming datagrams are also checked for the checksum. The UDP processor uses almost the same signals as the IP processor, only replacing the SOP signal with the Start-of-Datagram (SOD) signal. Applications can simply process datagrams or even generate new ones without being concerned about correct header values.

3 IP router OBIWAN

We will now present a fully functional Internet Protocol router, which we call OBIWAN (Optimal Binary search IP lookup for Wide Area Networks). The router uses one external SRAM module and an internal bitmap. Routing entries can be configured with control cells as described in section 3.3. The router extracts the destination IP address of incoming IP packets, which are encapsulated in AAL5 frames, buffers the data while the IP lookup is performed and forwards the packets with the VCI being replaced according to the next hop information. The WUGS switches the packets to one of the eight ports according to the new VCI. OBIWAN can operate at full line speed, i.e., 2.4 Gbps. It is designed to even work at the worst case, which is one IP packet in every ATM cell, or one lookup every 16 clock cycles. This provides a number of up to 6.25 million IP packets per second.

3.1 Lookup Algorithm

The lookup algorithm that we used for our implementation is a binary search over prefix lengths using hash tables. This algorithm is documented in detail in [10, 11]. The basic algorithm uses a hash table for each prefix length. A hash key and a value can be determined from the prefix and will be stored in the table. When a lookup is requested, the basic algorithm performs a binary search for the best matching prefix. It starts with the prefix length 16, and depending on a match it continues the search with a longer, i.e., 24 bits or a shorter, i.e., 8 bits length, until the longest matching prefix length has been determined. The number of iterations, i.e., the depth of the binary search tree of the basic scheme is five.

Though the algorithm given above performs very well already, in terms of memory accesses, it still requires 3 simultaneous lookups to guarantee line speed operation. It takes 4 clock cycles to read data from SRAM memory, mainly because of buffers on the chip boundary. Thus reducing the number of memory accesses is our main goal in improving the algorithm above. This can be done by reducing the depth of the binary search tree.

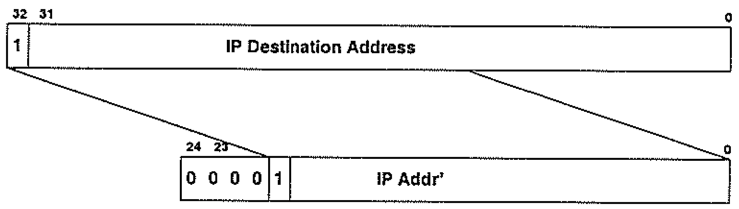


Figure 3: Right align IP addresses

First, the algorithm combines two adjacent prefix length by expanding the shorter prefix to two more specific routing entries. This has been described in [11], which also shows that this expansion does not increase the forwarding database. This step does reduce the maximum depth of our search tree by one.

Many routing entries have a prefix length shorter or equal to 16 and our analysis of network traffic has shown, that around 50% of all routed IP packets will take one of these routes. These results led us to implement a bitmap for the first 16 prefix bits, which indicate, whether a longer prefix exists in the hash tables. This does not only reduce the binary search tree by another level, but also improves the lookup speed for every second IP packet significantly. In fact, we assume that there is a matching 16 bit prefix for every address. Thus if the binary search fails finding a longer prefix, we retrieve the next hop information of the corresponding 16 bit prefix, while a special bit indicates, whether it is valid. For invalid next hops, we use the default route for this address.

To reduce the overall memory consumption of the hash tables and to reduce the probability of collisions, we only use two relatively large hash tables. One for all prefix lengths from 17 to 24, and another one for the range 25 to 32. To distinguish the prefix length of the entries in the tables, the prefixes are right aligned to either 24 or 32 bit, while a leading 1 bit guarantees uniqueness (Figure 3).

The key and value functions use a simple bit extraction algorithm, which is very efficient in hardware. For 17 to 24-bit prefix lengths, the key size is 16 bit, which leaves 9 bits for every value (Figure 4). The function for long prefixes uses 15 and 18 bits (Figure 5). Our final configuration (Figure 6) thus has two hash tables with four buckets per key each. The table for the shorter prefixes contains 64k entries with four buckets per word (*val24*), the other 32k entries with two buckets per word (*val32*). Tests have shown, that this configuration is useful with real routing tables and does not give many collisions. Despite the fixed number of buckets per hash table entry, collisions can still be resolved by expanding a prefix to more entries, thus moving them to other locations in the table.

For each match in a hash table, the lookup algorithm should determine a next hop information. Since this information is only necessary at the very end of the lookup, we separated it from the hash tables and put them in a shadow-table to avoid unnecessary memory bandwidth usage. The shadow tables have almost the same layout as the hash tables, with the difference, that only 8 bit next hop information is available for every entry (*nh24* and *nh32*). Additionally, every entry in the bitmap needs a corresponding

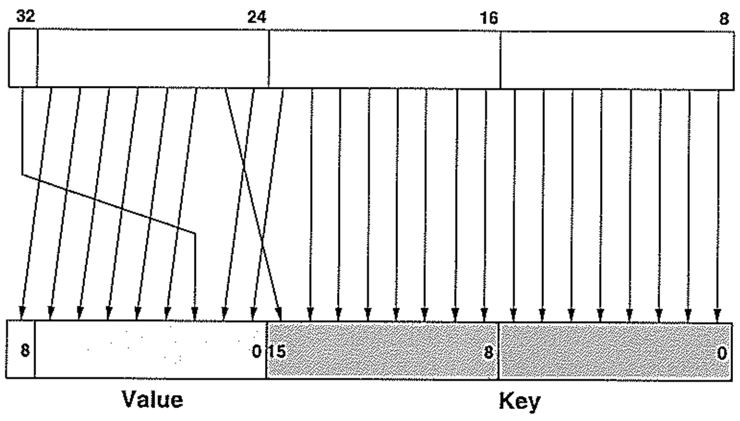


Figure 4: Right align IP addresses

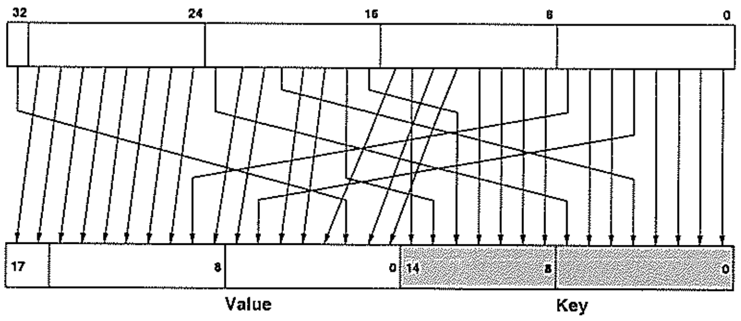


Figure 5: Right align IP addresses

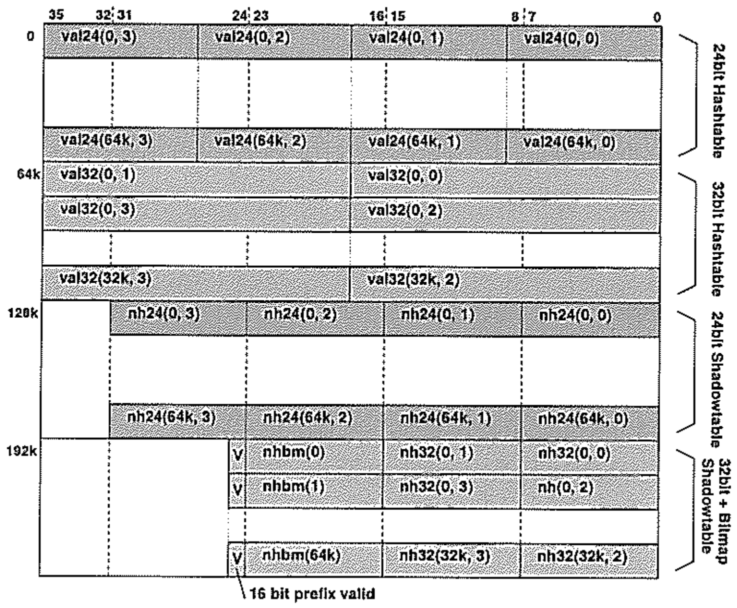


Figure 6: Memory layout for routing table

next hop information as well (*nhbm*). Since the hash table for up to 24 bit prefixes only uses 2 entries per word, we use the remaining space for corresponding bitmap entries. Every bitmap entry has a valid bit (*v*), since the algorithm assumes a 16-bit match before it starts the binary search.

3.2 Implementation

When IP packets first come into the router, the IP destination address is extracted and forwarded to the actual IP lookup engine. At the same time, the whole packet is stored in a FIFO. The packet can only be forwarded when the next hop information, which is a new VCI, is available from the lookup engine. While the packet is written to the FIFO, the payload words are counted. The number is then put in a separate queue. On the output side of the router, the IP packets are forwarded, with the VCI being replaced, as soon as the next hop information is available from the IP lookup. There is also a queue for next hop information, in case the output port is congested and IP packets cannot be forwarded fast enough. There are still two cases, which have to be distinguished:

- At least one packet length is available in the counter queue. Then the complete packet can be sent out on a new VCI, since the length is known.
- The counter queue is empty. Then there is only one packet buffered, which is still incomplete. Therefore all data in the FIFO can be sent out, but switch to the state above, as soon as the actual length is available.

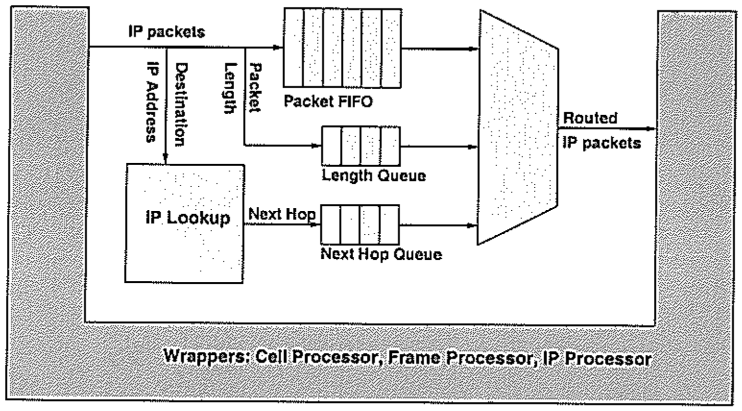


Figure 7: IP router module OBIWAN

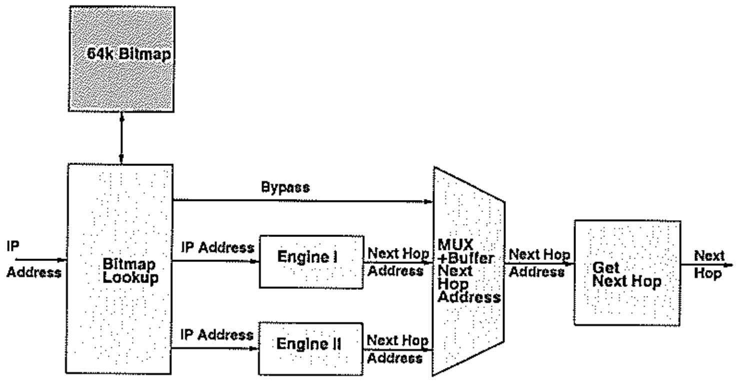


Figure 8: IP routing lookup (block diagram)

The actual IP lookup engine (Figure 8) takes a destination IP address on its input and delivers a next hop information (VCI) on its output after some time. The IP lookup engine works strictly in order. The IP address is first checked against the internal bitmap, which is located in on-chip memory. As mentioned earlier, the bitmap contains the information, if the longest prefix for this address has at most 16 bits. Therefore the bitmap is $2^{16} = 64kb$ in size. If there is no longer prefix than 16 bits, the next step, the binary search over hash table lookups, is skipped and a next hop address is forwarded. A next hop address encodes the location of the next hop information in SRAM. Otherwise the IP address is forwarded to one of two engines, which perform the binary search.

The binary search units (Figure 9) start with a prefix length of 24 bits, compute a key/value pair for the hash table and set the address for memory access. Because of

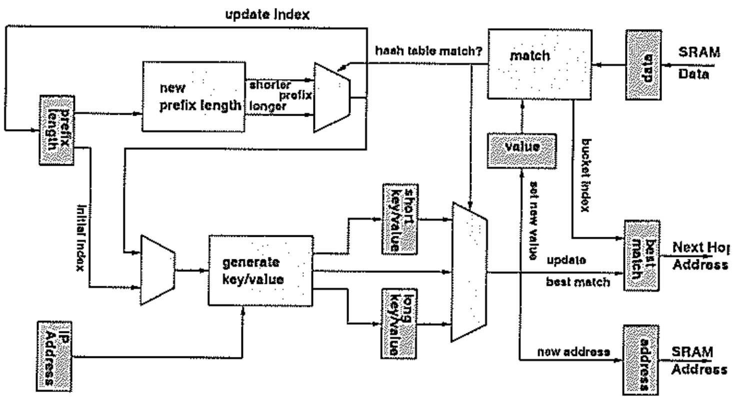


Figure 9: Binary search on hash tables

buffers at the chip boundaries and inside the lookup engine, it takes 6 clock cycles, until the data from memory can be evaluated. During that time, some pre-computations are done: the next possible longer and shorter prefix lengths are determined, the corresponding key/value pairs and memory addresses are computed and stored. When the data words are finally available, the values of all buckets are compared to the value corresponding to the current IP address. On a match, the address for the longer prefix is selected for the next iteration, otherwise the address for the shorter prefix is chosen. A match also updates the next hop address for the best matching prefix, which initially points to the 16 bit prefix of the IP address. After a total of 3 iterations, the next hop address is forwarded to retrieve the next hop information for the current IP address.

We also considered an alternative scheduling scheme. In addition to a hash table entry, the next two possible entries are also retrieved to eliminate some wait cycles. This scheme has not been implemented due to its higher complexity, while it does not obsolete one of the lookup engines.

Before we can read the next hop information from memory, we have to reorder the next hop addresses, coming from three different locations. Since some next hop addresses bypass the binary search, they may arrive out of order. A next hop multiplexer keeps track of orders to the binary search engines, their results and bypassed next hop addresses, buffers them, if necessary, reorders and then forwards them. Getting the next hop information is done by reading a word from the shadow tables. Both the memory address and the position within one word are encoded in a next hop address. If a next hop is read from a bitmap entry, the valid flag is also checked. If this check fails, the next hop is set to the default route, which can be set by sending a control cell to the module.

The IP router OBIWAN is designed for the FPX. The system clock on the FPX is 100 MHz and the FPGA used is a Xilinx Virtex E 1000-7. OBIWAN synthesizes with a speed of 111 MHz while utilizing 1196 lookup tables. The entire router, including the networking wrapper framework and infrastructure components, fits within 18% of the FPX chip.

3.3 Configuration

The OBIWAN router uses FPX control cells to configure virtual channels, default routes and routing tables. Control cells are structured ATM cells with an opcode to specify the command, a module ID to address the module on an FPX, and several parameter fields. OBIWAN uses the module ID 1. Control cells are secured with a 16-bit CRC. Three different commands are used to configure OBIWAN.

1. The virtual channel on which IP packets are processed and the default route next hop information are configured with the VPI/VCI command, which is described in [6]. The default VCI for IP traffic is 0x35(50) and can be changed by setting the application VCI register 0x10(16). IP packets going out on the default route are sent on the VCI that is configured in register 0x20(32).

2. Routing information for OBIWAN is stored in off-chip SRAM as described in section 3.1. To update this memory the Control Cell Processor (CCP) is used by OBIWAN. The CCP is a standard FPX module with the module ID 0 that processes memory update control cells.

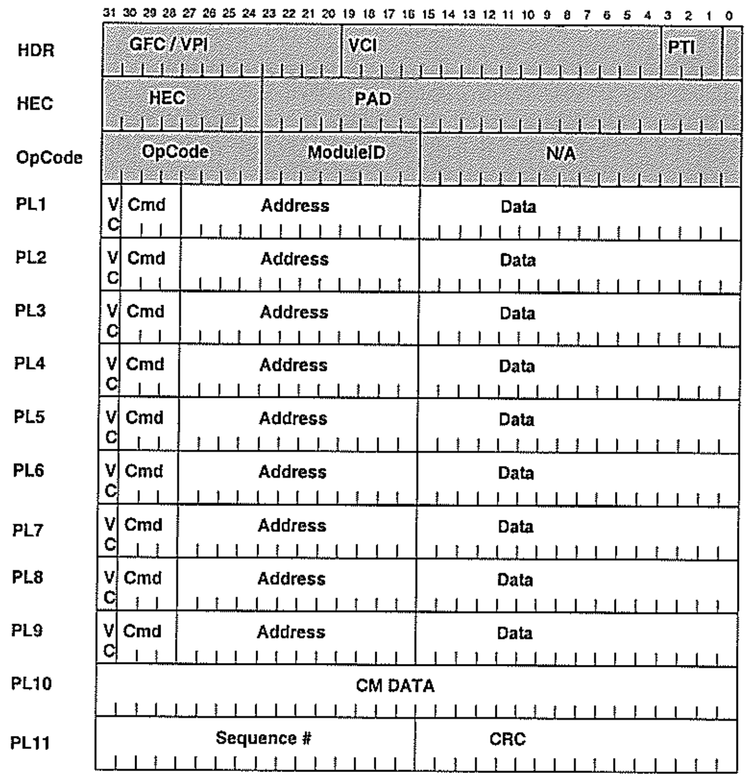
3. Before the routing information in SRAM is checked, our implementation compares the IP address with an internal bitmap. This bitmap can be configured with the control cell opcodes 0x12 (write) and 0x14 (read). This control cell format is only understood by OBIWAN. The layout can be seen in Figure 10. Several bits can be set or read at the same time. Every word of the control cell can address 16 bits out of 65536 if the valid bit is set to one. The write operation can simply overwrite the existing settings, but it can also be combined with an AND or an OR operation to make updates of single bits easier.

3.4 Improvements

Our goal was to have the IP router run at full line speed. Though we have achieved this goal with OBIWAN, there is still room for further improvement in terms of throughput and delay. These are mostly theoretical considerations, as they would only apply to systems faster than several gigabits per second.

First of all, the SRAM bandwidth is not fully utilized. Only 50% of the access cycles are assigned to the 2 lookup engines. The remaining cycles are free to retrieve next hop information, which takes only one clock cycle. Since for 24 bit hash table lookups, only one memory access is necessary, the next hop information could also be retrieved during a lookup engines' second cycle. A more advanced SRAM scheduler would be necessary, but up to four lookup engines are realistic and would double the worst case throughput to up to 12.5 million IP packets per second. This bandwidth cannot be provided by the current FPX implementation.

As we mentioned earlier, 50% of the packages match the bitmap lookup, so the lookup engines are idle every second package, on average. Introducing a FIFO for IP



VC – Valid Command1 = Valid, 0 = Invalid

OpCode	Command		
0x12	000	Clear Bits	BM[Addr] := BM[Addr] AND NOT Dat
	001	Set Bits	BM[Addr] := BM[Addr] OR Data
	010	reserved (set/clear chunk)	
	011	Set Word	BM[Addr] := Data
	1--	reserved	
0x14	000	Read Word	Data := BM[Addr]

Figure 10: Control cell format for bitmap updates

destination addresses, one could make advantage of its distribution. Two packets per 8 clock cycles seems to be a good average throughput. But now the retrieving of the next hop information can be a bottleneck. Recall that only one extra memory access per lookup (3 iterations) can be guaranteed, i.e., $\frac{4}{3} \approx 1.3$.

So far we only considered the worst case where every IP packet fits in only one ATM cell. This is not a realistic assumption, though. An IP header and a UDP header use 7 words of the 12 available already. Since the packet is encapsulated in AAL5 frames as well, which use another 2 trailer words, only 3 words or 12 octets are left for payload data. A TCP header is already 5 words in size, which gives no extra space to payload. Given that most of the IP packets consume more than 2 ATM cells, the potential bandwidth will at least double, since the number of lookups is independent of the packet length. On the other hand, this shows that our IP lookup engine will be idle most of the time.

Heading for the ultimate optimization, one could implement the IP router entirely on the cell-level, thus loosing the advantages of the framework, as easy development and code maintainability. A reasonable way to maintain data integrity is using incremental updates for the IP header checksum [12] and for the AAL5 CRC [13].

4 Conclusions

We presented a fully functional IP router, which runs at 2.4 Gbps (OC-48) and is based on our framework. The entire router consisting of all components (control cell processor, memory interface, frame processor, and OBIWAN router) has been synthesized for a Xilinx Virtex and fit within 18% of an XCV1000E FPGA. This indicates that the layering employed does not adversely affect size or performance. Besides the basic router, there is still plenty of space for future, application-defined functions, such as hardware-accelerated active networking or security processing.

References

[1] Tom Chaney, J. Andrew Fingerhut, Margaret Flucke, and Jonathan S. Turner. Design of a gigabit ATM switch. Technical Report WU-CS-96-07, Washington University in Saint Louis, 1996.

[2] John W. Lockwood, Jon S. Turner, and David E. Taylor. Field programmable port extender (FPX) for distributed routing and queuing. In *ACM International Symposium on Field Programmable Gate Arrays (FPGA'2000)*, pages 137–144, Monterey, CA, USA, February 2000.

[3] John W. Lockwood, Naji Naufel, Jon S. Turner, and David E. Taylor. Reprogrammable Network Packet Processing on the Field Programmable Port Extender (FPX). In *ACM International Symposium on Field Programmable Gate Arrays (FPGA'2001)*, pages 87–93, Monterey, CA, USA, February 2001.

[4] David E. Taylor, John W. Lockwood, and Naji Naufel. Generalized RAD Module Interface Specification of the Field-programmable Port eXtender (FPX). Technical report, WUCS-01-15, Washington University, Department of Computer Science, July 2001.

[5] Hamish Fallside and Michael J. S. Smith. Internet connected FPL. In *Proceedings of Field-Programmable Logic and Applications*, pages 48–57, Villach, Austria, August 2000.

[6] Florian Braun, John W. Lockwood, and Marcel Waldvogel. Layered protocol wrappers for internet packet processing in reconfigurable hardware. Technical Report WU-CS-01-10, Washington University in Saint Louis, Department of Computer Science, June 2001.

[7] B-ISDN ATM adaptional layer AAL Functional Description. CCITT: Recommendation I.362, 1991.

[8] B-ISDN ATM adaptional layer AAL Specification. CCITT: Recommendation I.363, 1991.

[9] Fred Baker. Requirements for IP version 4 routers. Internet RFC 1812, June 1995.

[10] Marcel Waldvogel, George Varghese, Jon Turner, and Bernhard Plattner. Scalable high speed IP routing table lookups. In *Proceedings of ACM SIGCOMM '97*, pages 25–36, September 1997.

[11] Marcel Waldvogel. *Fast Longest Prefix Matching: Algorithms, Analysis, and Applications*. Shaker Verlag, Aachen, Germany, April 2000.

[12] Computation of the Internet checksum via incremental update. Internet RFC 1624, May 1994.

[13] Florian Braun and Marcel Waldvogel. Fast incremental CRC updates for IP over ATM networks. In *Proceeding of 2001 IEEE Workshop on High Performance Switching and Routing*, May 2001.