Front End Device for Content Networking

Jeremy Buboltz
School of EECS
University of Central Florida
Orlando, FL 32816, USA
je963524@pegasus.cc.ucf.edu

Taskin Kocak
Dept. of Electrical and Electronic Engineering
University of Bristol
Bristol, BS8 1UB, UK
t.kocak@bristol.ac.uk

Abstract

The bandwidth and speed of network connections are continually increasing. The speed increase in network technology is set to soon outpace the speed increase in CMOS technology. This asymmetrical growth is beginning to causing software applications that once worked with then current levels of network traffic to flounder under the new high data rates. Processes that were once executed in software now have to be executed, partially if not wholly in hardware. One such application that could benefit from hardware implementation is high layer routing. By allowing a network device to peer into higher layers of the OSI model, the device can scan for viruses, provide higher quality-of-service (QoS), and efficiently route packets. This paper proposes an architecture for a device that will utilize hardware-level string matching to distribute incoming requests for a server farm. The proposed architecture is implemented in VHDL, synthesized, and laid out on an Altera FPGA.

1 Introduction

The bandwidth and speed of network connections are continually increasing. Systems that are based on software applications not only have to process the software application but also have the overhead involved in unpacking/packing data through the network protocol stack. Processes that were once executed in software now have to be executed, partially if not wholly in hardware. One such application that could benefit from hardware implementation is high layer routing.

1.1 Content Networking

Content networking (also known as layer-7 routing) has many applications. By accessing the payload of packets, more details about that packet can be gained. Routers, firewalls, spam and virus filters could all benefit from Content Networking. Some of these applications, like firewalls, spam and virus filters, require analyzing the entire contents of a packet. However, for routing, only a particular aspect of the payload may of interest. Perhaps the field of interest is the host, a cookie or a URL [7].

A difficulty in content networking, which causes content switches to be relatively rare, is that most of the high layer protocols are designed to be worked with on general purpose CPUs and also involve complex protocol processing [11]. For those applications that require inspecting the entire payload, the major bottleneck is having to process the entire payload. This can consume a vast amount of resources. This is especially true of software based systems that are limited by their base hardware which was designed to be general purpose. By implementing a single purpose hardware device, the speed of the device will be greatly increased. The ever increasing speed of network traffic is only stressing this point.

1.2 Front End Devices

Clusters of workstations or PCs are becoming a popular platform to deliver websites and content on the Internet. This architecture has proven to provide high performance for a relatively low cost [8, 9]. Another benefit is the ability to scale the resources to meet the demands placed on the system. To deliver requests to the individual machines of a server farm, a device is needed to accept all incoming traffic and to assign a request to a particular machine to handle [10]. These devices are called Front End Devices (FEDs), because they sit at the front of a server farm accepting all requests. A major problem now arises, if a similar workstation or PC is used for a FED, then the number of attached machines in the server farm is typical limited to ten or less [9]. This limits the scalability of

978-3-9810801-3-1/DATE08 © 2008 EDAA

the server farm, detracting from one of the major benefits of this architecture.

In this work, a design and implementation of a hardware FED is presented. One application would be a group of web servers specialized for specific tasks. For example, an administrator might assign a specific server to handle all multi-media requests, whereas another server is used to handle all XML processing. Or perhaps a server has been assigned to hold databases and handle all database requests. Because of the ever increasing size of the Internet and increasing network speeds, a software based FED could be stressed to handle the load of a moderate server farm. However, by implementing the FED in hardware, then the increase in operating speed would result in the ability to handle larger server farms.

1.3 String Matching

The most important component of implementing a hardware FED is being able to find the appropriate server the packet needs to be assigned to. To do this, the hardware FED must match the request with one of a known set of strings. Therefore, a strong hardware level string-matching mechanism is needed.

With a wide variety of applicability, string matching has been continually optimized. String matching can be applied to all sorts of problems. String matching solutions can be categorized into two subcategories: exact string matching and approximate string matching. Each field can then be further divided into hardware and software based solutions. Approximate string matching is commonly used for spell checking or other related functions. Exact string matching is more suited for the requirements of a FED. Two of the most popular algorithms for exact string matching are Aho-Corasick [1] and the Boyer-Moore [12] algorithms. The Boyer-Moore algorithm is sub-linear for the average case. However, for length of text, n, and length of pattern, m, in the worst case complexity is O(n*m). For the Aho-Corasick algorithm, time complexity is O(n+m). Another method to search strings is with a content addressable memory (CAM). A CAM can search a specific length of data in a single memory access. By activating every memory cell with every search, the CAM has proven to be very fast. However, this method also consumes lots of power and can take up a fair amount of silicon area [2, 13].

The Aho-Corasick (AC) algorithm is capable of being implemented as a finite state machine. Characters are fed into the state machine serially, with each successful match leading further down a state path to a final state of a match. Failures will either result in a return to the initial state or a jump to another portion of the state graph [1]. With the AC algorithm, only one state will be active at a time. Thus, the AC implementation will consume less power then a CAM implementation. With the AC algorithm's logical transition to a hardware structure the control circuitry needed will be minimized. The Boyer-Moore algorithm would have needed a large amount of control circuitry to implement in hardware. This results in a larger design with a larger energy footprint than an AC implementation. To minimize the footprint and power consumption of the design, the Aho-Corasik string matching algorithm was chosen for implementation in the proposed FED.

2 Design

2.1 Proposed Architecture

To help illustrate the proposed FED architecture lets first follow the flow of a packet through the system. This flow can also be seen in Figure 1. As packets arrive, the IP header is checked to verify that it is a TCP packet. If a packet is determined to be a GET request, the application layer data is extracted from the packet in the stored buffer. Next that data is sent, a byte at a time, to the Character Decoder. The Character Decoder is used to eliminate the need for individual decoders located at each state. After being decoded, a single output line will be enabled. If that output line is connected to the first state, then that state will become active. The process will continue the same way for the next byte. Once one of the output states has been reached, the state encoder will encode the data and send it to the port look-up table as an address. The port look-up table will hold a value that is used to select a port. Once a port has been selected, the data will move from the Buffer, through the Port Selector and out the specified port.

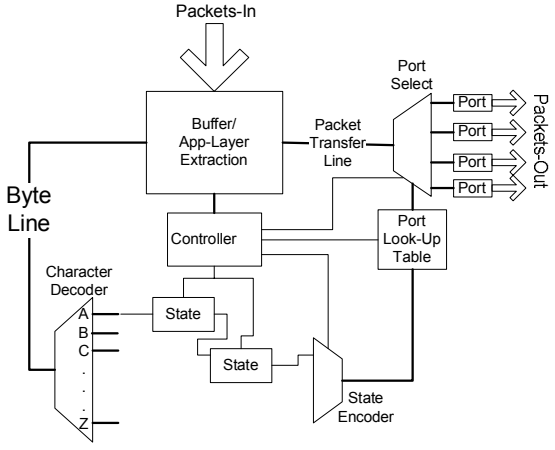

Figure 1 – Proposed FED Architecture

Figure 2 is a state diagram that illustrates the described flow through the proposed FED. This diagram shows some of the communication lines that will need to be present to help the control circuit to navigate from state to state.

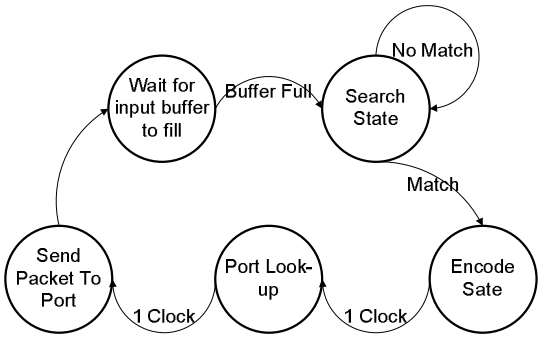

Figure 2 - FED State Diagram

Figure 3**Figure 3** depicts the design hierarchy for implementing the proposed architecture in VHDL. This diagram shows how the individual components are architected and how these components are then joined together to form the completed overall design. From the diagram, there are seven individual components that will be utilized.

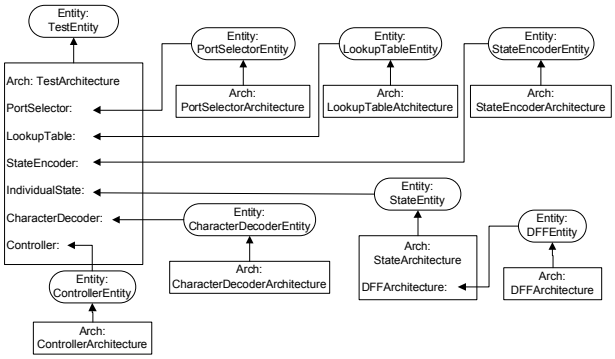

Figure 3 - The VHDL Architecture Of The Proposed FED

2.2 Sample Case

To implement the design and test the overall system, an imaginary server farm was created. The system entailed four separate servers for specific purposes. Therefore, the proposed FED will have a single port entering and four ports exiting to the servers. First, all video requests will be tasked to a single server. Second, there is a server for XML processing. Third, there is a server for basic file transfers. And lastly, there is a default server that all remaining traffic will be sent to. The following file types will be matched by the proposed FED: DOC, GIF, HTM, JPEG, JPG, MPEG, MPG, MOV, PDF, PPT, TIFF, TXT, WMV, XLS, XML. These file types were chosen because of their prevalence on the Internet. If the proposed FED reaches the end of a packet and no match was found, then the packet will be sent to the default server.

In total there are 17 characters represented (not case-sensitive). Therefore, the Character Decoder will be an 8-bit to 17 output lines. Next, there will be 41 D-Flip-Flop gates, each representing another state in the Aho-Corasick Finite State Machine. There are fifteen end states, one for each file type, with a sixteen end state of no match. So, the State Encoder will be a 16 line to 4-bit encoder. Because there are only four output ports, a two bit enumeration can be used to represent any of the four output ports. Therefore, the Port Look-up Table will only have to store 2 bits. So, the Port look-up Table will be 16 rows (one for each of the match states) by 2-bits. The Port Selector will then be a 1-to-4 MUX with a 2 bit select line.

3 Results

3.1 System Simulation

Once all of the components are developed and tested individually, the components are connected for integration and system level testing. The overall circuit has been simulated with the results shown in Figure 4. The packet data is read in and transferred using the ‘s’ signal. The data is then passed to the Character Decoder. The Character Decoder then asserts the appropriate output lines, which are connected to the inputs of the individual states. Next, we can see that the proposed FED matches a file format as states 17, 18 and 19 are traversed. These states represent the three letters of the file format ‘gif’. Because of the delay for the characters to get decoded and for the D-flip-flops to latch onto the output in the Individual States we can see that the Individual States actually output an active signal a full clock cycle after the byte has been passed to the Character Decoder.

Once a final state has been reached and the character for the last state has been matched, then the output to the final state is asserted. In Figure 4, state 19 is the final state in the string of states used to match the file format ‘gif’. When the output of a final state is asserted, then the EndStateReached variable is asserted. This signal then kicks off the second portion of the circuit. First, the state is encoded by the State Encoder, as seen in the EncodedState variable.

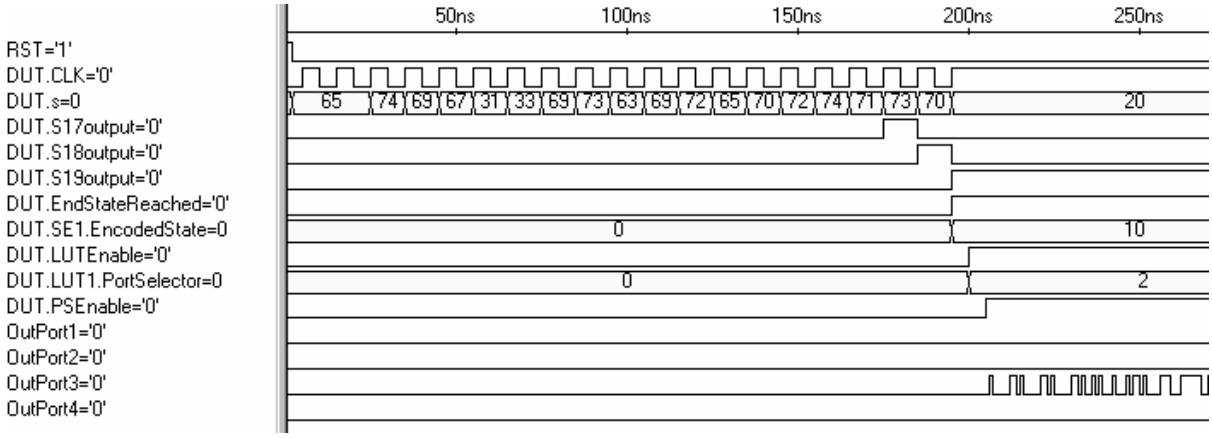

Figure 4 - Completed Simulation

Next, the look-up table is enabled and the encoded state is used to find the appropriate port number in the table. The look-up table has been pre-encoded with the appropriate file format to server matching. In Figure 4, the PortSelector variable is outputting a value of 2, signaling that the packet should be assigned to the third server. Next, the Port Selector is enabled and the packet information is sent to the Port Selector on a transfer line. This can be seen in Figure 4, as the PSEnable variable is asserted, then the packet is transferred out of OutPort3.

3.2 Synthesis

After the functional simulation was completed, the VHDL code was synthesized and laid-out using Altera's Quartus II software. After the VHDL code for the Front End Device is imported into the Quartus II program, VHDL compilation can occur. Figure 5 is an annotated schematic of the RTL for the proposed FED. All of the components of the proposed FED have been highlighted.

As part of the synthesis process, the Quartus software then translates the VHDL into hardware. This hardware can include combinational logic, registers, and Logic Cells (LCs). These hardware components are available resources on the FPGA chosen as the target board. Table 1 lists the total number of FPGA elements used per VHDL entity, and the number of elements that were implemented at the stated level of implementation. For instance, the FEDEntity block implements all of the VHDL entities, so it consumes 115 LCs. However, because the FEDEntity only implements other VHDL entities, it does not require any logic, only registers that are used to store date as it is passed between the other entities. This is denoted by

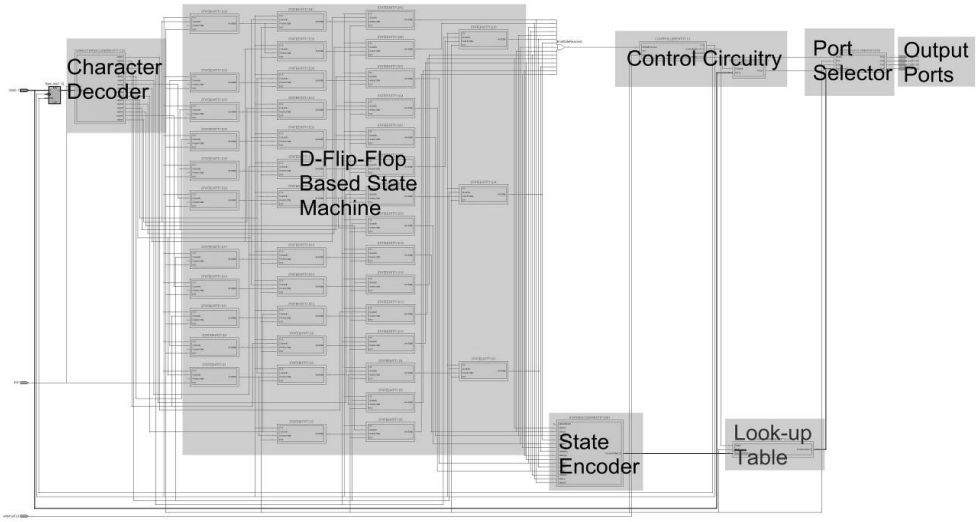

Figure 5 - Annotated RTL View

the seven in parenthesis following the number 115. However, referencing the Register Only LCs column shows that the seven LCs consumed by the FEDEntity were simply for the use of registers.

VHDL Logic Block	Logic Cells	LUT only LCs	Register only LCs
FEDENTITY (Total)	108 (7)	108 (0)	7 (7)
Controller	3 (3)	3 (3)	0 (0)
Character-Decoder	12 (12)	12 (12)	0 (0)
LookUpTable	2 (2)	2 (2)	0 (0)
PortSelector	4 (4)	4 (4)	0 (0)
StateEncoder	9 (9)	9 (9)	0 (0)
StateEntity – State 1	2 (1)	2 (1)	0 (0)
State 1: DFFEntity	1 (1)	1 (1)	0 (0)
StateEntity – State 2	1 (0)	1 (0)	0 (0)
State 2: DFFEntity	1 (1)	1 (1)	0 (0)

Table 1 - Resource Usage Per VHDL Entity

An interesting point to notice is that there are two different implementations for the 42 StateEntities. The first version represented by 'StateEntity – State 1' in the table does not use require the standard AND logic preceding the D-Flip-Flop, as in Figure 2. This is because the state is the first state in a matching string. So, there is no input to denote that the previous state (previous letter) is active. Since the D-Flip-Flop requires an ALUT itself, the total FPGA resource consumption for this version of the StateEntity is a single ALUT. However, the typical StateEntity does contain internal logic as well as a D-Flip-Flop, resulting in the 'StateEntity – State 2' VHDL entity in 2. This entry shows that 2 ALUTs are used, one for the D-Flip-Flop and one for the internal logic.

3.3 Layout

Once, the Quartus software has completed synthesis, all of the hardware required to completely implement the circuit on the FPGA is known. So, the next step is to layout the pieces, or in the case that the circuit is going to be implemented on a FPGA, to assign the required resources to the available resources on the FPGA. A model EMP240F100I5 from the Max II FPGA family contains 240 Logic Elements. Referring to Table 1, the sample circuit implemented requires roughly 45% of the available resources of the FPGA.

Once layout is completed, a more detailed timing analysis can be performed. The proposed FED has a worst-case set-up time (tsu) of 0.106 ns, a clock-to-output delay (tco) of 17.997 ns, an intrinsic [pin-to-pin] delay (tpd) of 8.921 ns, and hold time (th) of 1.433 ns for the FPGA operating at 140 MHz. Therefore, if the average URL length is 100 bytes, then the proposed FED can search 1.4 million queries per second. This shows that the true limiting factor on the device is how quickly an implementation can pass queries to the proposed FED.

Lastly, a power analysis can be performed once layout is complete. With a toggle rate of 50% on the input I/O pins, the proposed FED design will consume 61.48 mW. This is the sum of the Core Dynamic Thermal Power Dissipation of 9.97 mW, Core Static Thermal Power Dissipation of 39.61 mW and the I/O Thermal Power Dissipation of 11.91 mW.

4 Related Work

In [7], a router is implemented that checks for the URL in a request packet. There was a large overhead found related to the retrieval and converting the string. The solution was to format a link on a website into a code. The special links were denoted with a preamble before to coded portion of the link began [7]. The disadvantages are that the router could not be used on an existing website without any work having to be done to the website. Also, a factor of the human interface of the website was degraded by encoding the URL of links. The proposed device in [7] was capable of supporting 0.6 – 0.9 million queries per second.

In [6], a firewall was implemented in hardware. By allowing the firewall to search the content of the packet, it was capable of performing spam and virus filtering. The firewall also limited the rate of per-flow traffic to prevent denial-of-service attacks. The firewall was implemented as a system-on-programmable-chip with an architecture that will help mitigate against future attacks as new exploits are developed. By implementing their design in hardware, the authors were able to achieve multiple gigabit throughput speeds. If an ASIC was used, then when adding extra functionality the entire chip would have to go through the design process again.

Much of the current research has been applied towards creating high wire-speed network intrusion detection systems (NIDS). Because of the large number of threats faced by networks and computers and increasing line speeds, software based NIDS have become unusable [3, 4, 14].

A method is presented in [5] to reduce space consumption by removing the character matchers from the individual states and creating a single 'character decoder.' This decoder allows multiple states to all use the same decoder. This helps to eliminate repetition in the design. This method is also implemented in the FED proposed in this paper.

A difficulty in content-based switching is that most web services use TCP as their high layer protocol. To access any payloads, a TCP connection must first be made. Once the connection has been made with the switch, it is difficult to migrate the connection to the server. So in [11], work was performed to help eliminate the difficulty of migrating form of TCP splicing allows the port controllers on the switch to handle any minor processing needed to continue the connection. This allows the processor to be freed for other actions.

5 Conclusions and Future Work

Modern web services and large websites are moving towards distributing the amount of incoming traffic among multiple servers. However, as the speed and amount of internet traffic increase, a single point of failure is the distribution node. As the rate of growth of network speed increases, software solutions which have to process the software application as well as packing and unpacking through the TCP/IP stack begin to fail under the increased network load. To increase the front-end device's ability to handle ever increasing workloads, this paper proposed migrating the FED to a hardware only architecture. A VHDL implementation of a FED for hardware level string matching has been constructed. This device would make an ideal component or add-in to a router or any other front-end device with other features such as security. The proposed FED was synthesized for a MAX II FPGA from Quartus. A synthesized operational frequency of 140 MHz with a power consumption of 61 mW was achieved.

Future work for this project would involve more fully integrating the FPGA with the design of popular routers. This would involve accessing the data path of the router, allowing the network processors to send data to the FGPA chip. Also, a mechanism for TCP hand-off could be implemented. By handing off the TCP connection, stress would be relieved from the router.

References

[1] A. Aho and M. Corasick, Efficient String Matching: An Aid to Bibliographic Search, *Communications of the ACM*, 18(6):333-340, 1975.

[2] S. Fide and S. Jenks. A Survey of String Matching Approaches in Hardware, Technical Report, UC-Irvine, 2006.

[3] M. Aldwairi, T. Conte, and P. Franzon. 2005. Configurable string matching hardware for speeding up intrusion detection. *SIGARCH Comput. Archit. News* 33, 1 (Mar. 2005), 99-107.

[4] B. L. Hutchings, R. Franklin, and D. Carver. Assisting Network Intrusion Detection with Reconfigurable Hardware. *Proceedings of Field-Programmable Custom Computing Machines*, 2002.

[5] C. R. Clark and D. E. Schimmel. Scalable Parallel Pattern-Matching on High-Speed Networks. *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, CA, April 2004.

[6] J. W. Lockwood, C. Neely, C. Zuver, J. Moscola, S. Dharmapurikar, and D. Lim. An extensible, system on-programmable-chip, content-aware Internet firewall. *Proc. of the Field Programmable Logic and Applications*, Lisbon, Portugal, Sept. 2003.

[7] M. Luo, C. Yang, and C. Tseng. Content management on server farm with layer-7 routing. *Proc. of the 2002 ACM Symposium on Applied Computing*, Madrid, Spain, March 11 - 14, 2002.

[8] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel. Scalable content-aware request distribution in cluster-based network servers. *Proc. of the 2000 USENIX Annual Technical Conference*, San Diego, CA, June 2000.

[9] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-aware request distribution in cluster-based network servers. *SIGPLAN*, no. 33, 11; Nov. 1998.

[10] E. A. Brewer. (2001, Jul/Aug) Lessons from giant-scale services. *IEEE Internet Computing*. 5(4). pp. 46-55

[11] G. Apostolopoulus, D. Aubespin, V. Peris, P. Pradhan, D. Saha. (2000, Mar) Design, Implementation and Performance of a Content-Based Switch. *INFOCOM 2000* . 3(3) pp. 1117-1126

[12] R.S. Boyer and J.S. Moore, A fast string searching algorithm. *Carom. ACM* 20 (10), 262-272, 1977.

[13] T. Kocak and F. Basci. A power-efficient TCAM architecture for network forwarding tables. *Journal of Systems Architecture*, 52 (5), 307-314, 2006.

[14] T. Kocak and I. Kaya. Low-power bloom filter architecture for deep packet inspection. *IEEE Communications Letters*, 10 (3), 210-212, 2006.