# MUTABLE CODESIGN FOR EMBEDDED PROTOCOL PROCESSING

*Todd Sproull*

Applied Research Laboratory
Washington University
St Louis, USA

*Gordon Brebner, Christopher Neely*

Xilinx Research Labs
Xilinx, Inc.
San Jose, USA

## ABSTRACT

This paper addresses exploitation of the capabilities of platform FPGAs to implement embedded networking for systems on chip. In particular, a methodology for exploring trade-offs between the placement of protocol handling functions in programmable logic and on an embedded processor is demonstrated. This is facilitated by two new design tool capabilities: first, being able to describe programmable logic based functions in a more software-like manner; and second, being able automatically to generate efficient interfaces between a programmable logic fabric and an embedded processor. The methodology is illustrated by an example of a simple web server, targeted at Xilinx Virtex-II Pro or Virtex-4 FX platform FPGAs. Trade-offs both of complete protocol placement and of within-protocol placement are systematically investigated in terms of resources used and packet handling latency. This provides an excellent range of service times, corresponding to differing logic fabric and memory resource requirements. The work points the way to highly fluid allocation of functions to implementations, beyond conventional static codesign.

## 1. INTRODUCTION

Embedded networking is a technology of critical importance to systems to be deployed in the future, where devices and appliances have some level of networking capability to communicate with their environment and other systems within it. The scale and complexity of such networking need not necessarily be as great as seen in traditional telecommunications and computer networking, in particular because the resources available and cost models for the networked systems do not allow it. Therefore, a technical challenge is to provide design processes that support the cheap and simple implementation of the communication protocols to be used.

In this paper, the capabilities of contemporary platform FPGAs are exploited through new design tools, in order to illustrate how highly mutable implementations of embedded networking for systems on chip can be produced, and how trade-offs between implementations can be rapidly investigated. A particular focus for investigation is the use of an embedded processor within the logic fabric of the FPGA. Although hard processor blocks are studied here, the general discussion is equally applicable to the use of soft processors configured in the logic fabric.

In order to explore thoroughly trade-offs between implementation of functions in programmable logic and on an embedded processor, it is necessary to adopt a very fluid notion of hardware/software codesign. That is, the conventional situation, where top-level system partitioning decisions are made on function placement, with a processor-centric flavor to the system architecture, represents just one codesign point. Here, the flavor of system architectures is very much *interface-centric,* that is, driven by the required behavior of the system's interface to its environment. Mutable assignments of functions to implementations can both be investigated at design time, and also be considered as dynamic alternatives at run time.

Such an exploration requires appropriate design tool support, and the work here relies on two main tool advances resulting from research within Xilinx Research Labs. The first is a compiler for a description language for networking functions expressed in a software-like form that is more abstract and domain-specific than hardware description languages. This has been described in detail in an earlier publication [4]. The second is a mapper that automatically generates extremely efficient interfaces between the programmable logic fabric and an embedded processor, something of particular use for implementing trade-off points rapidly.

The general methodology is demonstrated through an example of implementing a simple web server on a platform FPGA. This server has a gigabit Ethernet connection to its environment, and uses a tailored embedded networking subset of the IP, TCP and HTTP protocols to communicate. Eight different trade-offs in terms of allocating protocol functions to programmable logic or to the embedded processor are shown, exhibiting a range of acceptable server response times, at different latency, and logic and memory, resource points. Although only semi-automated at this stage, the whole exercise was carried out by a non hardware expert within a total of six weeks.

## 2. PLATFORM FPGAS AND MUTABLE CODESIGN

Platform FPGAs are exemplars of reconfigurable system-on-chip platforms, with their programmable interconnect being an exemplar of a reconfigurable network on chip. They present both unique design challenges and unique design opportunities. Unlike general system-on-chip design, of course, the mix of on-chip resources is pre-determined by the FPGA vendor rather than the designer, which places constraints on the explorable design space. In this paper, the focus is on the effective use of embedded processors, as a programmable companion to programmable logic. Note that, although this is the focus, it is not the intention to accord the processor special status within the system on chip, as is often the case in codesigned systems. Rather, it is regarded here as just one particular function block, alongside the various other types of block available in a platform FPGA.

The exact nature of the hard processor itself is not a major concern here, but its interfacing to the logic fabric is particularly important, when investigating implementation trade-offs for functions. For example, in the case of the Xilinx Virtex-II Pro [1], the on-chip memory (OCM) interface is available for use, alongside a soft instantiation of the IBM CoreConnect bus architecture. In the case of the Virtex-4 [1], the auxiliary processor unit (APU) interface is also available. The best way of harnessing these options is influenced by the use model for the embedded processor within the overall system on chip.

Codesign methodologies seek to avoid inefficiencies associated with traditional system design that develop separate hardware and software components. The inefficiencies arise through a specification being developed and sent to hardware and software engineers, with the hardware-software partition decided *a priori* and adhered to as much as is possible, because any changes in this partition may necessitate extensive redesign. Incompatibilities might arise across the hardware-software boundary, and sub-optimal systems may arise as a result of the pre-defined partitioning of functions. The work reported in this paper is concerned with enabling very fluid codesign, specifically in the embedded networking domain. The intent is to allow experimentation with placement of functions at design time, and indeed to open up the possibility of changing the placement of functions at run time.

Two central requirements to achieving mutable codesign are, first, that functions are described in representations that are capable of efficient mapping to logic or processor or both and, second, that efficient interfacing between logic and processor can be easily constructed. It need not be the case that there is a single representation for functions, for example, an ‘everything written in C’ approach. Rather, and as seen in this work, the key is to have representations at a high enough level of abstraction that producing different representations of functions can be speedy.

In the embedded networking methodology described in this paper, the overall system architecture is interface-centric, with an embedded processor available to assist in handling activities arising from the system’s environment. That is, there is no main program running on the processor that acts as a driving force for the system’s operation. Thus, the architecture is empathetic with the software decelerator (or, to use more neutral terminology, *software assistant*) model [2, 3], and indeed the practical experiments have benefited from tool development arising from research into software decelerators. However, in order to allow for maximum fluidity in codesign, there is no explicit use of either a software decelerator model or a hardware accelerator model. The hardware and software components are viewed as co-workers.

To investigate trade-offs in a mutable codesign framework, it is necessary to have apt measures that reflect the operational requirements of the embedded networking systems. Here, two dimensions are considered. The first one is area, measuring the programmable logic resource and embedded processor resource required. The second one is time, measuring the packet handling time for the embedded networking functions. For area measurement, the number of lookup tables (LUTs) is used here as the measure of programmable logic resource. No charge is made for the silicon area occupied by the embedded processor, since it is assumed as a fixture present whether used or not. Instead, the amount of instruction and data memory is used here as the measure of embedded processor resource. To combine these two area measures into a single figure, each 12 bytes of embedded processor memory are counted as one LUT here. This is motivated by an estimate that one Xilinx Virtex-II LUT occupies approximately the same silicon area as 12 bytes of BlockRAM memory. For time measurement, the latency between a request arriving from the network environment and the completion of all resulting responses sent back is used as the measure of time. This is an apt domain-specific measure for embedded networking, rather than raw clock frequencies, because the requirement is that clock frequencies are sufficient to match external interface rates and need not necessarily be higher.

## 3. CASE STUDY: A WEB SERVER ON CHIP

As a case study of the mutable codesign methodology, together with use of recent research tools, the implementation of a simple web server on a platform FPGA was investigated. The initial target FPGA was the Xilinx Virtex-II Pro, with a migration path to the newer Virtex-4 FX. The HAEC language [4], developed in Xilinx Research Labs as part of research into network processing using platform FPGAs, was used for representing functions targeted at programmable logic. C was used for representing functions targeted at the embedded PowerPC processor.

### 3.1. Web server functions

The web server responds to requests for web pages that are sent to the platform FPGA over a gigabit Ethernet link. This involves a stack of four communication protocols:

1. HTTP, for web requests and responses; over
2. TCP, for end-to-end data streaming; over
3. IP, for end-to-end packet transfer; over
4. Ethernet MAC, for Ethernet packet transfer.

There is an immediate possible processor-centric implementation point for this server, which would be to run Linux (or some other OS) on the PowerPC and use its standard TCP/IP networking facilities with standard web server software. However, the case study here is concerned with embedded networking, where an aim is to use minimal resources to support necessary functionality and no more.

The nature of the web server implemented was that it implements only the minimum protocol subsets required to communicate — importantly though, always keeping within the protocols. This aspect is seen particularly in the case of the TCP protocol. First, only a single TCP connection (and hence web request) is handled at a time, and it is always initiated by a client, by definition. The level of functionality of the TCP responder is similar to that of [5], relying on the client to handle retransmissions and timeouts if error recovery is necessary. The responder thus handles only basic TCP packet header processing, checksum calculations, as well as the connection setup (SYN/ACK) and connection teardown (FIN/ACK) subprotocols [6].

Once a connection is set up, the expectation is that the TCP responder receives an HTTP request message within a single TCP packet. It then immediately issues a TCP ACK packet, followed by the requested web page data in another TCP packet, and finally a teardown TCP FIN packet. In principle, the three TCP packets sent back could be reduced to just two, or even one, while keeping within protocol. However, for the case study, this approach gave more realism in terms of extrapolating the behavior to situations where web page data spans multiple TCP packets.

Traditional approaches for performing TCP protocol processing in hardware, whether by ASIC or FPGA, typically involve the use of offload engines [7], following a hardware acceleration model. These engines can perform a variety of tasks, such as calculating checksums or handling segmentation and reassembly [8, 9]. Earlier work has demonstrated full implementation of TCP/IP on platform FPGA based systems, including the use of offload engines combined with use of the embedded processor.

The Gigabit System Reference Design (GSRD) [10] is an example of providing TCP checksum offload capabilities in programmable logic. Using the GSRD, [11] demonstrates a full high performance TCP/IP implementation on a Xilinx Virtex-II Pro FPGA. In [12], an implementation of the lightweight Internet Protocol (lwIP) stack running a TCP echo server is demonstrated. In [13], there is an example of a web server running on the Virtex-II Pro. In all these cases though, the TCP processing either exists entirely in the PowerPC or with just the TCP checksum calculation placed in programmable logic. Here, more general possibilities are investigated.

| Version name | Description |
|---|---|
| HTTP | Programmable logic only, PowerPC not in use |
| TCP | Logic processes up to TCP and PowerPC processes HTTP |
| TCP/DATA | PowerPC processes TCP SYN/FIN only and logic processes remaining TCP functionality |
| TCP/SYN/FIN | Logic processes TCP SYN/FIN and checksums, and PowerPC processes remaining functionality |
| TCP/CHKS | Logic processes TCP checksums and PowerPC processes remaining TCP functionality |
| IP | Logic processes up to IP and PowerPC processes TCP and HTTP |
| ETH | Logic processes only Ethernet MAC |
| PPC | PowerPC only: web server protocol stack exists entirely in software |

**Table 1**. Web server codesign points

### 3.2. Implementation trade-offs explored

To explore trade-offs between programmable logic and embedded processor, eight different codesign points were selected for the web server implementation. These involve partitioning of the protocol stack along a protocol boundary in some cases, and partitioning within the TCP protocol in other cases. Table 1 provides a summary of each of the eight codesign points explored.

Aside from the handling of the varous protocols, the only other functionality present in each design was the physical gigabit Ethernet interface. This was always implemented by the standard Xilinx Gigabit MAC (GMAC) core implemented in programmable logic.

At one extreme point, named **HTTP**, all of the protocol handling is implemented in programmable logic alone. At the other extreme point, named **PPC**, it is implemented on the embedded processor alone. The latter corresponds to the earlier example shown in [13]. These extreme cases serve as benchmarks in terms of demonstrating the maximum and minimum latency and resource utilization.

In between the two extreme codesign points, there are six intermediate points. The first of these, named **TCP**, involves placing the simple HTTP processing into the PowerPC — this involves accepting an HTTP request message, and returning a fixed web page as an HTML-formatted response. All of the other protocol handling — Ethernet MAC, IP and TCP — remains in programmable logic.

The second to fourth intermediate points involve different partitions of the implementation of TCP. In the second, named **TCP/DATA**, processing of the SYN and FIN packets used for TCP connection setup and teardown is handled by the PowerPC, with all other TCP functions remaining in programmable logic. A rationale for this partition is to place less frequently used, or potentially more algorithmically complex, functionality on the embedded processor. In the third intermediate point, named **TCP/SYN/FIN**, processing of TCP data transfers is handled by the PowerPC with other TCP functions (checksum calculation and processing of SYN/FIN packets) remaining in programmable logic. A rationale for this partition is to place TCP data packet handling alongside HTTP handling. In the fourth intermediate point, named **TCP/CHKS**, only calculation of TCP checksums remains in programmable logic, with all other functions handled by the PowerPC. This corresponds to the conventional checksum offload case.

The fifth intermediate point, named **IP**, involves all TCP (and HTTP) processing being handled by the PowerPC, with the remaining (Ethernet MAC and IP) processing being handled in programmable logic. The final intermediate point, named **ETH**, involves the IP processing also being handled by the PowerPC. This processing includes swapping source and destination IP addresses, computing IP header checksums and checking the IP protocol field to determine the higher level protocol processing.

### 3.3. Mutable codesign methodology

The first step in the methodology was to produce a HAEC representation of the whole web server, leading to an implementation entirely in programmable logic. In this representation, separate HAEC threads were used for each of the four protocols (Ethernet MAC, IP, TCP and HTTP). The HAEC external interface mechanism was used to integrate the gigabit Ethernet core, and the memory block mechanism was used to create buffers for storing arriving and departing packets. This resulted in a working implementation of the web server for the Virtex-II Pro, without the embedded PowerPC being used. A benefit of using HAEC was that the system was implemented by a non hardware expert, and indeed a HAEC novice, within four weeks.

Two factors were then critical to enabling the speedy development of the other codesign points. The first was that the high-level nature of the HAEC language simplified the removal of particular functions that were to be reassigned to the processor, without disturbing the remainder of the implementation in programmable logic. In the case of complete protocol handling, complete threads could be removed. In the case of partial TCP handling, separable parts of the TCP threads could be cleanly removed. The second factor was that a further Xilinx Research Labs prototype tool was available to completely automate the integration of the embedded processor component with the programmable logic component. It takes an arbitrary block of C code to act as a packet handler, and wraps its PowerPC implementation into a black box with a highly efficient gigabit-rate interface via the OCM bus from and to the programmable logic fabric. This interfacing arrangement uses the techniques described in [2]. Thus, the only new work to create a new codesign partition point was to write a piece of C code to match the function of an existing piece of HAEC code. Ultimately, a C event handler was created that contained the entire protocol handling functionality of the web server. As a sanity check, this could be compared with an existing similar C implementation [13].

The codesign methodology described is semi-automated, but one that proved realistic and productive in the setting of embedded networking. There are two main steps towards a fully automated approach based on a single system description. The first would involve automation of the movement of pieces of high-level code from one side of the logic-processor partition to the other. The second would involve mapping a single description efficiently to both hardware and software. The current compromise is attractive because the remaining non-automated features are those that engage human creativity, expressed at a high level of abstraction, with the more unpleasant aspects taken care of by tools such as the HAEC compiler and the PowerPC black-boxer. This exercise proved very worthwhile purely in terms of using two previously independent tools in tandem, and so pointing the way to future integration.

## 4. EXPERIMENTAL RESULTS

The web server system was targeted at a Xilinx XC2VP7 Virtex-II Pro platform FPGA, which includes eight multi-gigabit transceivers. The standard Xilinx ISE 6.3 tools were used to produce the bitstream for the FPGA from the VHDL description of the programmed soft platform generated by the HAEC compiler. The system was tested using a Xilinx ML300 [1] board, which includes four gigabit Ethernet interfaces, one of which was connected by optical fiber to a Linux workstation that acted as a client for the server. The GMAC core, which has an eight-bit packet interface, was clocked at 125 MHz for full gigabit Ethernet rate. The threads in logic, which operate on 32-bit data, were clocked at 31.25 MHz. The PowerPC was clocked at 300 MHz, with the OCM bus running at 100MHz.

| Version name | Logic | PowerPC | Total |
|---|---|---|---|
| HTTP | 1312 | 0 | 1312 |
| TCP | 1312 | 4391 | 5703 |
| TCP/DATA | 1312 | 6551 | 7863 |
| TCP/SYN/FIN | 1312 | 7951 | 9263 |
| TCP/CHKS | 1312 | 10111 | 11423 |
| IP | 928 | 15651 | 16579 |
| ETH | 576 | 22911 | 23487 |
| PPC | 0 | 27850 | 27850 |

**Table 2**. Protocol handling latency

| Version name | Logic (LUTs) | PowerPC (bytes) | Total (pseudo-LUTs) |
|---|---|---|---|
| HTTP | 2806 | 0 | 2806 |
| TCP | 2683 | 640 | 2736 |
| TCP/DATA | 1955 | 832 | 2024 |
| TCP/SYN/FIN | 2119 | 976 | 2200 |
| TCP/CHKS | 1391 | 1168 | 1488 |
| IP | 1185 | 1424 | 1304 |
| ETH | 488 | 2804 | 722 |
| PPC | 0 | 3092 | 258 |

**Table 3**. Resources for protocol handling

For the codesign trade-off experiments, ModelSim simulation was used, to allow fast evaluation of the different codesign points. In all cases, timings were measured for handling a single HTTP request. Through a simulation harness, TCP packets were transmitted to the FPGA to set up a connection (TCP three-way handshake), then to request a web page, and then to close the connection. Separate latency measurements were made for the amount of time spent on protocol handling in programmable logic and on the embedded processor. The latter measurements include the time for transferring packets between the logic fabric and the processor. In carrying out the PowerPC measurements, an initial HTTP request was made in order to fully populate the PowerPC instruction and data caches, and the measurements were made for a subsequent request.

### 4.1. Latencies

Table 2 reports the protocol handling latency (in nanoseconds) associated with the programmable logic and embedded processor for each codesign point. This latency does not include time taken for the physical reception and transmission of packets, and also eliminates any unavoidable latency from arbitrary gaps between the receipt of packets.

The PowerPC-only codesign point has the highest latency, which is as might be expected. When Ethernet MAC handling is moved to logic, the total latency decreases by 17%, and when the IP handling is moved to logic, latency is reduced by an additional 23%. With the TCP checksum handling only moved, an additional 19% latency reduction is experienced. The most dramatic decrease in latency occurs when the PowerPC is not used at all, where the protocol processing is 21 times faster.

These various total latency decreases would have been fairly predictable in advance, certainly the decreasing trend, if not the exact numbers. However, the use of the codesign methodology has pointed out the constant programmable logic latency for all the different partitions across the TCP protocol. This is because of the concurrent execution of threads, and indicates that calculation of the TCP checksum is the dominant factor in the concurrent mix. Thus, with the threaded implementation, a full implementation of TCP costs as little as just checksum offload, from a latency perspective, although clearly has resource costs. Note too that there is no latency overhead when the HTTP protocol is also implemented in logic, since the HTTP thread executes concurrently with the TCP thread.

In terms of the actual web server application, the latencies show a range from 1.31 $\mu$s to 27.85 $\mu$s for handling a request for a web page. Given that a simple fixed web page is served, these latencies can be interpreted as the embedded networking overhead on top of the time required for more sophisticated web page serving. The whole latency range is very positive in terms of fast response.

### 4.2. Logic and memory resources

Table 3 reports the amount of programmable logic (in terms of LUTs) and the size (in bytes) of the PowerPC code+data responsible for protocol processing, for each version of the web server. The number of LUTs does not include the fixed number for the gigabit Ethernet MAC core: around 1400 LUTs on Virtex-II Pro. However, for Virtex-4 FX, there is no LUT overhead since the hard tri-mode Ethernet block can be used. The number of bytes of code+data does not include the fixed number used for low-level support functions, which was around 4300 bytes in the configuration used. In fact, this overhead could be significantly reduced if unused support functions were eliminated. The third column show an overall ‘pseudo-LUT’ total, counting each 12 bytes of PowerPC memory as one LUT.

It can be seen that, in general, the resource utilization for TCP handling is fairly low, as is desirable in this context, but it is important to remember that this is a stripped-back embedded networking version of TCP, which is much less complex than a full implementation. Performing the TCP checksums in logic is a fairly inexpensive codesign point. However, once TCP connection state processing is introduced to logic, the logic utilization grows fairly quickly.

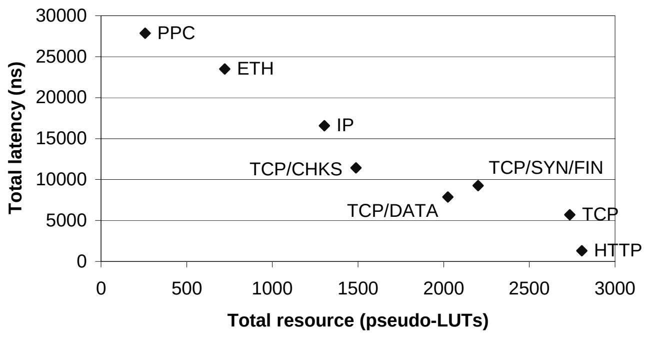

**Fig. 1**. Summary of resource latency trade-offs

When the entire TCP handling is placed in logic, the resource utilization has more than doubled from the case where TCP is entirely handled by the PowerPC. It is interesting to note that the TCP/DATA codesign point illustrates better logic utilization then the TCP/SYN/FIN codesign point. Having the connection set up and tear down state processing implemented on the PowerPC saves a fair amount of resource. The HTTP implementation is extremely simple, thus its logic implementation requires a relatively small number of LUTs. Upon receiving an HTTP request, it just responds with a web page containing the word "Hello", and so here there is little algorithmic complexity that might be better placed on a processor.

### 4.3. Area-time results

Figure 1 summarizes the area-time trade-offs for the different codesign points. This scatter graph illustrates a clear 're-source step' occurring for the TCP protocol, and emphasizes a non-trivial pattern of latency gain resulting from moving different TCP functions between logic and processor. As might be expected, performing partitioning between protocols has a greater relative effect on latency than partitioning within the TCP protocol.

### 4.4. Results for Virtex-4 FX implementation

The designs have also been investigated for Virtex-4 FX platform FPGA implementation. The newer technology provides two main benefits for the web server system. First, the presence of two hard tri-mode Ethernet MAC blocks means that it is no longer necessary to consume LUTs for the soft GMAC core, saving around 1400 LUTs. This overhead was not included on the LUT counts reported here. It was found that there was no significant change in the numbers of LUTs used for protocol handling. Second, the Virtex-4 FX has the same PowerPC 405 embedded processor as Virtex-II Pro, but it is possible to clock it faster. Using a 400 MHz clock instead of the 300 MHz clock used in the Virtex-II Pro experiments results in the expected 25% reduction in latency, and a slightly larger reduction if the OCM clock rate was raised to 200 MHz.

## 5. CONCLUSIONS

This paper has illustrated a practical methodology for rapidly investigating trade-offs between different codesign points, when implementing embedded networking protocols on a platform FPGA. Two new research tools were critical to automating the exploration process: first, a domain-specific compiler for mapping software-style function descriptions to programmable logic; and second, an assembler for generating efficient interfaces between programmable logic and code running on the embedded processor. Given these, it was possible for a non hardware expert to carry out the necessary programming for all of the codesign points within a total of six weeks. The resulting implementations have attractive attributes in terms of resources used and/or packet handling latency, and a variety of measures have been presented in order to assess their relative merits.

## 6. REFERENCES

[1] Xilinx, "Homepage," `http://www.xilinx.com/`.

[2] P. James-Roxby, G. Brebner, and D. Bemmann, "Time-critical software deceleration in an FCCM," in *Proc. 12th IEEE FCCM*, 2004, pp. 3–12.

[3] E. Keller, G. Brebner, and P. James-Roxby, "Software decelerators," in *Proc. 13th FPL*, 2003, pp. 385–395.

[4] E. Keller and G. Brebner, "Programming a hyper-programmable architectures for networked systems," in *Proc. 3rd IEEE FPT*, 2004, pp. 1–8.

[5] Miniweb, "Homepage," `http://www.sics.se/-adam/miniweb/`.

[6] W. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. New York: Addison-Wesley, 1994.

[7] Broadcom, "BCM5706 TCP offload engine," `http://www.broadcom.com/`.

[8] Qlogic, "Homepage," `http://www.qlogic.com/`.

[9] D. Schuehler and J. Lockwood, "A modular system for fpga-based tcp flow processing in high-speed netwo rks," in *Proc. 14th FPL*, 2004, pp. 301–310.

[10] Xilinx, "Gigabit system reference design," Xilinx XAPP536, Aug. 2004.

[11] S. Narayanaswarmy, "High performance TCP/IP on Xilinx FPGA devices using the Treck embedded TCP/IP stack," Xilinx XAPP546, Dec. 2004.

[12] S. Thammanur and C. Borrelli, "TCP/IP on Virtex-II Pro devices using lwIP," Xilinx XAPP663, Aug. 2004.

[13] M. Muggli, M. Ouellette, and S. Thammanur, "Web server reference design using a PowerPC-based embedded system," Xilinx XAPP434, Nov. 2004.