Design and implementation of a TCP checksum offload engine based on the multiport memory controller and the processor local bus

Christian Valerio-Regalado
Tecnológico de Monterrey, Monterrey, Nuevo León, México, cvalerio@itesm.mx

Alfonso Avila-Ortega
Tecnológico de Monterrey, Monterrey, Nuevo León, México, aavila@itesm.mx

ABSTRACT

The TCP protocol has a checksum operation to verify the data integrity. This operation is performed over all bytes of a TCP segment, consuming several clock cycles and occupying precious processing time from embedded processors. This paper presents the implementation of the TCP checksum in an independent hardware module to reduce the processing time. This module, called the TCP Checksum Offload Engine (TCOE), integrates a Multiport Memory Controller (MPMC) and a Processor Local Bus (PLB) to improve the response time. The experimental results reported performance improvements in the TCOE-based system compared to the performance of the baseline implementation for segment sizes greater than or equal to 79 bytes. The clock cycles of the TCOE-based system also represented only 1% of the baseline clock cycles after increasing segment size and reaching its maximum value.

Keywords: Hardware-software co-design, TCP protocol, checksum operation

RESUMEN

El protocolo TCP lleva a cabo una operación de suma de comprobación para verificar la integridad de los datos. Esta operación se realiza sobre todos los bytes de un segmento TCP con un consumo de varios ciclos de reloj y con alta demanda de tiempo de procesamiento de los procesadores embebidos. En este artículo presenta la implemetnación de la suma de comprobación del TCP en un módulo de hardware independiente. Este módulo, llamado TCP Checksum Offload Engine (TCOE), integra un controlador de memoria multipuerto (MPMC) y un bus del procesador local (PLB) para reducir el tiempo de procesamiento. Los resultados experimentales reportaron mejoras en el rendimiento del sistema basado en TCOE en comparación con el rendimiento de la implementación de referencia considerando tamaños de segmento mayores que o iguales a 79 bytes. Los ciclos de reloj del sistema basado en TCOE también representarón solo el 1% de los ciclos de reloj de referencia después de incrementar el tamaño de segmento y llegar a su valor máximo.

Palabras claves: Co-diseño de hardware y software, protocolo TCP, operación de suma de comprobación

1. INTRODUCTION

Embedded processors have become a common component for digital networks. These processors execute network specific tasks as well as data-generation and data-consumption tasks. Thus, these processors usually have to share their limited resources, like memory and CPU, between the network and non-network tasks. Network tasks, like the processing of network protocols, have become a bottleneck in computer communication as a result of the growth disparity found between network and computer technologies (Bhattacharya and Varsha, 2006). Improvements at the physical network layer have resulted in the emergence of network technologies like the 10-Gigabit Ethernet.

The nominal clock rate of single core processors limits their capability to meet the performance requirements found in the network technologies (Faulkner and Brampton, 2009). Embedded processors, specific purpose architectures with fewer amounts of hardware resources, face even greater difficulties to meet these performance requirements.

The most resource-demanding code sections of tasks related to network protocol processing are the operations performed over a large number (whole bunch) of bytes that are sent or received. These code sections implement data copy and checksum operations (Bhattacharya and Varsha, 2006). Data coy operations move data back and forth from user to kernel space. Data copy operations also exchange data between kernel space and the Network Interface Card (NIC.) Transport layer protocols like TCP and UDP make extensive use of checksum operations. The TCP/IP protocol stack is the most extensively used set of network protocols for internet connection. Embedded systems with internet as a part of their application environment require a TCP/IP stack to establish an internet connection. Thus, the design of internet-enabled embedded systems must meet the computing demand imposed by the protocol stack without becoming overloaded.

This paper focuses on the hardware/software implementation of the TCP protocol. The TCP checksum, one of the two resource-demanding operations, is implemented in hardware to meet the computing demands of internet-enabled embedded systems. TCP checksum has an important place in terms of the CPU occupation required to process the TCP protocol (Bhattacharya and Varsha, 2006). Research efforts, described in the next section, have attempted to minimize the latency generated by the checksum calculation. This paper presents a new implementation of the hardware/software interface using two components: (1) the Multiport Memory Controller (MPMC) and (2) the Processor Local Bus (PLB). The MPMC enables a faster access to the data required for checksum calculation and the PLB communicates the checksum hardware to the embedded processor. The rest of the paper is organized as follows. Section 2 reviews previous work about improving TCP checksum processing. Section 3 describes the architecture of the TCP Checksum Offload Engine (TCOE) module. Section 4 explains the internals and behavior of the module. Section 5 describes the operation of the hardware/software interface. Section 6 presents the experimental results. The last section presents the conclusions and the future work.

2. RELATED WORK

Research efforts have addressed the TCP checksum calculation to improve its performance. Three important strategies have been identified (Wang and Wang, 2005). The first strategy, known as copy and sum, exchanges data between kernel space and the Network Interface Card (NIC) to perform the checksum calculation. In (Clark, 1982), Clark proposed this strategy and reports no implementation. The second strategy, presented in (Finn and Hotz, 1996) describes a possible implementation of the zero copy mechanism. No copy of the data is required to calculate the checksum. The data sum operations, part of the checksum calculation process, are obtained during the data exchange between the NIC and the network. Thus, no checksum result needs to be part of the TCP segment to be sent; the checksum value becomes part of trailer at the data link layer protocol. This strategy introduces compatibility issues to communicate computers in the same network. A gateway compatible with zero copy mechanism has to place the checksum value in the correct field position of the TCP header before it leaves the network. The third strategy, presented in (Kleinpaste and Steenkiste, 1995) integrates aspects of the first and second strategies. It requires copying data from the host memory to the NIC calculating the checksum during a "send" data transfer. It also requires calculating sums during a "receive" data transfer that moves data from the network to the NIC. A special NIC, named communication acceleration board (CAB), is necessary for communication compatibility.

The three strategies described above introduce specific requirements such as: a specific purpose processor for data copying, special network cards, special data link protocols, or new TCP header conventions. The solution proposed in this paper maintains an acceptable performance without specific requirements; the proposed solution works with existent hardware requiring special changes in neither the TCP conventions nor the data link protocols.

3. THE PROPOSED TCOE-BASED SYSTEM

Figure 1 shows the hardware/software implementation of the system. The software is executed at the PowerPC 405 processor. The specific purpose hardware is the TCP Checksum Offline Engine (TCOE); this hardware performs the resource-demanding checksum calculations. The Processor Local Bus (PLB) communicates the TCOE module with the processor. The Multiport Memory Controller (MPMC) directly connects the TCOE module to the data memory. TCOE to memory interfacing required one out of eight of the MPMC ports. This port is a Native Port Interface (NPI) type. The NPI is a Personality Interface Module (PIM) for low level direct access to the MPMC core (Xilinx, 2011). This low level port eases the customization of the interface between the MPMC and the TCOE module to meet response time needs.

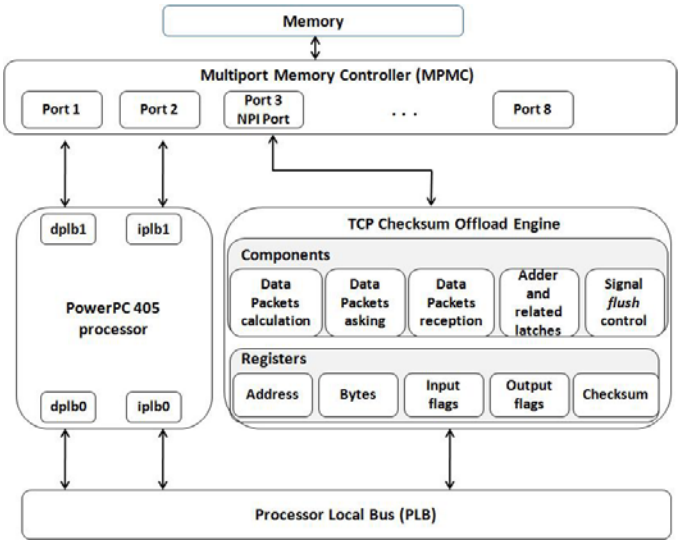

Figure 1: The proposed architecture

The architecture of the proposed system contributes to satisfy the computing demands required by the internet-enabled embedded systems. The TCOE module accesses directly the data memory to bring the large amount of data needed to compute the checksum operations of the TCP protocol. Thus, the PowerPC 405 processor consumes no computation cycles with a reduction of the overall latency to compute the checksum operations. The TCOE module, illustrated in Figure 1, has five internal registers accessible via PLB. The first holds the initial address of the data. The second register keeps the total number of bytes. The third one, a status register, holds input flag bits, the forth register holds output flag bits and the fifth register holds the checksum result. The TCOE module is also composed by five main components: the first component calculates the size and number of byte packets to request, the second component performs the packet requests, the third component receives the packets, and the fourth component adds the incoming bytes. The bytes are added as soon as these bytes are received. Finally, the fifth component controls the signal flush.

4. TCOE INTERNAL STRUCTURE AND BEHAVIOR

Before starting a checksum calculation the processor must enter the initial address and the number of bytes to sum into the first and second registers respectively. Any change in this two registers makes the first component to recalculate the number and size of the packets to ask to the MPMC. The minimum packet size is 8 bytes for a 64-bit port. The packet length can be 8, 16, 32, 64, 128 or 256 bytes. The NPI delivers 8 bytes in a single clock cycle. To deliver a 128-byte packet, the NPI needs 16 clock cycles. Due to performance and complexity reasons (Valerio, 2012) the selected packet lengths were 8, 64, 128 or 256 bytes. This process to determine the number of packets and the packet length is transparent to the processor; the processor only needs the address and the number of bytes to do the rest.

After that, the processor must generate a rising edge at the less-significant-bit (lsb) of the third register to start the sum. This action makes the second component to clean the checksum ready flag and start asking packets to the NPI port. Then, the third component is prepared to receive any data coming from the NPI port. Receiving data means controlling the information that is passed to the fourth component (the adder) after all data has been received and summed, to perform the final carry adding, as it is indicated by the checksum calculation (Forouzan, 2007), and finally to raise the checksum ready flag.

Some of the first bytes of the first packet and some of the last bytes of the last packet received could be discarded due to the need to align the packets with an address divisible by its length (e.g. a 128 bytes length packet must start in an address which in binary notation its first less significant seven bits are zero). The information about the discarded bytes is passed to the adder by the component in charge of receiving bytes. The adder has eight latches that filter the data coming from the NPI port, one for each byte, based on the information received. The latches use this information to let pass the actual data byte, or pass a null byte which contains only zeros. The adder is a nine input component. It receives the 8 bytes from the NPI port and a ninth feedback input. This last input comes from a latch connected to the adder output that holds the last result of the sum, and like the other eight inputs, this one has another latch connected to it to filter the incoming data.

Data is added as it is coming from the NPI port. Carries from any single sum are accumulated, so the adder output is 32 bits wide to keep the cumulative carries. The component in charge of receiving data is the one performing the final carry adding, consisting in adding all the 16 carry bits to the first less significant 16 bits. And, this operation is doing twice because the first carry adding could generate an extra carry. When the double carry sum is complete, the component raises the checksum ready flag, which is the lsb of the fourth register.

The fifth component controls the flush signal. By activating this signal, the component can stop the NPI port sending asked data, until the port is required to send data again. This is useful to eliminate the last clock cycles of the last packet which could contain data that was asked due to the address alignment rule but it is not intended to be summed. The processor has to do the following 6 steps in order to perform a checksum calculation. Step one is to enter the initial address in the first register. Step two is entering the number of bytes to be sum in the second register. Steps three and four are entering a ‘0’ followed by a ‘1’ in the lsb of the third register to generate a rising edge. Step five is to check if the lsb of the fourth register is ‘1’. Finally, once step five is true, step six is to read the result of the checksum from register fifth. If the validation in step five is not true, the processor can do any other relevant task and then return to validate step five again.

EXPERIMENTAL RESULTS

The experimental platform was a Xilinx’s Virtex II Pro development board that contains an FPGA Virtex II Pro XC2VP30. This FPGA also has a PowerPC 405 hard core embedded processor. This board includes an Ethernet port and contains an FPGA with enough resources to carry out the experiments needed to validate the proposed implementation. The Multiport Memory Controller (MPMC) and the PLB were part of the Xilinx Embedded Development Kit (EDK). The NPI ports can be configured either 32 or 64 bit wide. A 64-bit width was selected to maximize the throughput of the port.

The functional testing of the TCOE-based system had four stages. The first stage was to exhaustively check all the components separately. The PowerPC 405 was the first component to test. The software-processor version of a test program was coded and tested at this stage to establish the baseline measurements. The test program, written in C language, coded a test sequence that requested to perform N-byte checksums starting at a fixed address and incrementing the number of bytes from 1 to the maximum value of N. The TCP protocol specifies a maximum value of 65535 for N. At the second state, the TCOE module was individually tested using simulation tools. At the third stage, the interfaces between the processor and the TCOE module, and the interface between the NPI port and the TCOE module were also tested. These tests included simulations of the NPI port. At the fourth stage, the complete TCOE-based system was uploaded into the FPGA. It was possible to obtain timing diagrams of the FPGA implementation of the TCOE-based system using the Chipscope tool from Xilinx. These timing diagrams were compared to the timing diagrams obtained by simulation to ensure functional correctness.

A TCOE-Module version of the test program was developed and used to test the functionality of the system. The results of the software-processor and the TCOE-Module versions were compared to ensure functional correctness again.

Performance tests were conducted to determine the degree of accomplishment of the computing demands required by the internet-enabled embedded systems. Two metrics were selected for performance evaluation. The first performance metric is the time needed to communicate the PowerPC processor with the TCOE module through the PBL bus. The second performance metric is time needed to perform the checksum operation without considering the communication time. Table 1 shows experimental results obtained with the FPGA development system. The table has the cycle count that the processor takes to read or write to the five registers connected to the PLB. The values at the average clock-cycle count column were calculated as the mean value among 100 measurements. The measurements were obtained from the VirtexII-pro development system using the Chipscope tool.

Table 1: Clock cycles for PLB register interfacing access

Register access transaction	Average clock cycle count
Writing on Address register	482.24
Writing on # Bytes register	443.39
Writing on Input Flags register	443.93
Reading Output Flags register	272.49
Reading Checksum register	273.07

This first metric is obtained by adding the five types of transactions (in Table 1) plus an additional input flag register transaction (to generate the rising edge on the lsb.) Thus, the minimum time needed for communication during a checksum operation is 2359 clock cycles. This number can be larger if the ready flag is not asserted on the first try. A reassertion of the ready flag results in an additional access to the output flag register.

There are two instances of the second performance metric: (1) the time needed by the PowerPC processor to calculate checksum operation using the software-processor version of the test program and (2) the time needed by the TCOE module to calculate the checksum operation. Table 2 reports the results for these two instances obtained from the VirtexII-pro development system. The percentage column is the ratio between TCOE value divided by the Software-processor value. The value in the “Without PLB” column is obtained by subtracting 2359 cycles from the TCOE value.

Table 2 shows the different clock cycle counts required to compute the checksum operation for specific group sizes. The total number of clock cycles is almost the same for group sizes between 1 and 500 bytes. The TCOE module is faster to compute the checksum compared to the processor-TCOE communication. Group sizes between 500 and 1000 bytes slightly increment the clock cycle counts. An increment in the group size is translated in an increment in the clock cycle count for group sizes above 1000 bytes. Checksum calculation for group sizes below 80 bytes requires a larger clock cycle count in the TCOE module compared to the software-processor computations. The performance of the TCOE module constantly improves for group sizes above 80. At 65535 bytes, maximum value imposed by TCP, the TCOE module cycles are less than 1% of the processor cycles.

Figure 2 presents graphs of TCOE module and Software-processor versus the group size. A linear curve is useful to fit the data. The slope for the TCOE module and the software-processor are 0.1492 cycles/bytes and 23.986 cycles/bytes respectively. The graph shows the faster software-processor growth compared to the other. The TCOE module cycles counts show a slight change and are smaller than the software-processor cycle counts.

Table 2: Cycle count to checksum N bytes

N Bytes	TCOE Module (cycles)	Software-Processor (cycles)	Percen-tage	Without PLB (cycles)
1	2609.42	711.51	366.74%	250.37
10	2600.75	943.24	275.73%	241.70
20	2601.05	1178.94	220.63%	242.00
30	2600.63	1443.56	180.15.%	241.58
40	2600.93	1719.13	151.29%	241.88
50	2601.41	1899.50	136.95%	242.36
60	2600.64	2134.63	121.83%	241.59
70	2595.91	2404.07	107.98%	236.86
80	2594.70	2669.24	97.20%	235.65
90	2595.15	2850.77	91.03%	236.10
100	2594.62	3087.90	84.03%	235.57
150	2609.56	4323.49	60.36%	250.51
500	2599.94	12674.46	20.51%	240.89
1000	2645.74	24728.64	10.70%	286.69
3000	3011.93	72697.21	4.14%	652.88
10000	4020.67	240683.31	1.67%	1661.62
20000	5470.95	480401.90	1.139%	3111.90
30000	6925.61	720437.56	0.96%	4566.56
40000	8493.96	960171.36	0.89%	6134.91
50000	10014.65	1199693.07	0.84%	7655.60
60000	11596.51	1440187.15	0.81%	9237.46
65535	12400.60	1572534.65	0.79%	10041.55

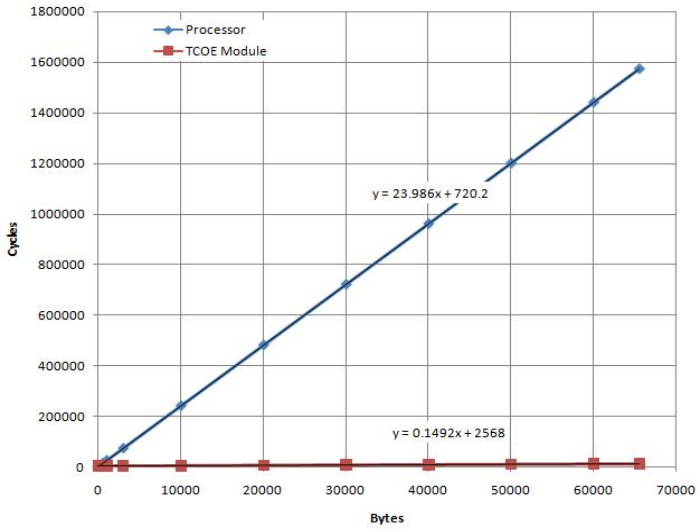

Figure 2: Dispersion graph: bytes vs cycles for module and processor

Figure 3 shows the clock cycle counts of the software-processor and the TCOE module within a group size range going from 1 to 100 bytes. TCOE clock cycle counts show insignificant change and are greater than the software-processor clock cycle counts. The slope of the software-processor clock cycle counts is the same for Figure 2 and Figure 3. The experimental behavior of the two clock cycle counts can be approached using two equations. The intersection point can be obtained by solving the equations. This intersection point is 78.53 and can be rounded to 79. This intersection value is a threshold indicating whether the TCOE module performs better or not. Thus, a group size larger than 79 indicates a better performance of the TCOE module. The greater the group size, the better performance the TCOE module according to Figures 2 and 3.

Finally, an important observation to remark from Table 2 is the fact that the TCOE module could perform better than the software-processor even from a 1 byte checksum, as it can be seen by comparison of columns “Without PLB” and “Software-Processor”. The PLB communication degrades the TCOE module performance.

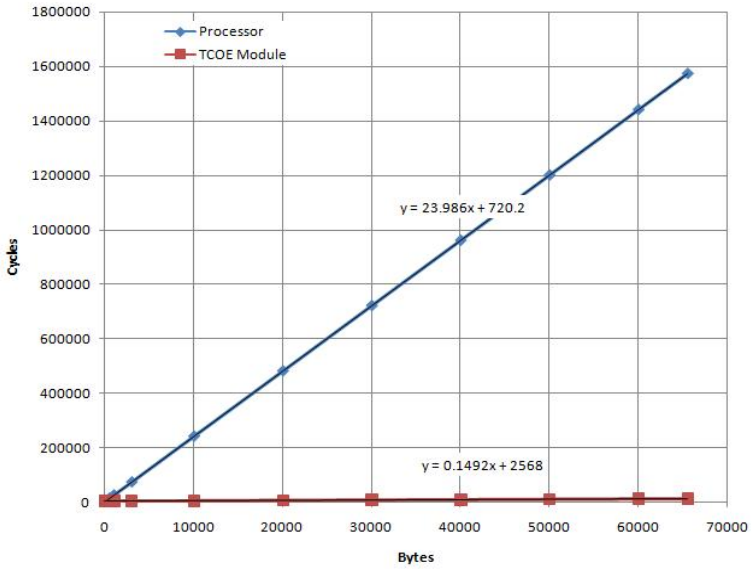

Figure 3: Zoom to the first 100 bytes with same data from Figure1

Table 3 summarizes the FPGA resources used by the TCOE module. The GCLK has a utilization percentage of 18%. This utilization is the highest among the FPGA resources required by TCOE module. The FPGA still has room to synthesize another component 4 times larger. The size can be larger than 4 times if the utilization of GCLKs is lower.

Table 3: FPGA resource utilization

Resource	Used	Available	Usage
Slices	749	13696	5%
Slice Flip-Flops	482	27392	1%
4 input LUTs	1359	27392	4%
IOs	379	-	-
Bonded IOBs	0	556	0%
GCLKs	3	16	18%

5. CONCLUSIONS

This paper presents the design and implementation of a hardware module that performs TCP checksum operations. The TCP checksum is a resource-demanding operation. The hardware implementation has the purpose of satisfying the computing demands of internet-enabled embedded systems. This module is known as “TCP Checksum Offload Engine” (TCOE). The TCOE module is interfaced to an embedded processor using the Processor Local Bus (PLB) and to a data memory using the Xilinx’s Multiport Memory Controller (MPMC).

To calculate a checksum operation, the TCOE module receives the initial address, the number of bytes and the start signal. The TCOE module stores the checksum result in a register accessible by the PLB. The TCOE module uses a Native Interface Port (NPI) of the Multiport Memory Controller (MPMC) to bring the data to be summed directly from the data memory. Performance tests demonstrated that the TCOE-based system calculated the checksum operation faster than the software-processor based system. The TCOE-based system required less number of clock cycles for group sizes larger or equal to 79 bytes. The TCOE-based system has the potential to offer faster checksum calculations for any group size, if the communication latency between the TCOE module and the processor is significantly reduced or eliminated. Thus, the most efficient TCOE-based system can be established either by calculating checksums with TCOE module for N values larger than 79 clock cycles, or by calculating the checksums using the processor for N values smaller than 79 clock cycles.

Important future work is an exhaustive performance evaluation of the TCOE-based system under an experimental computer networking environment. Other important goal is the identification and implementation of highly efficient communication mechanism between the TCOE module and the processor. A communication mechanism that interfaces the TCOE module to the memory bus is a promising option.

REFERENCES

Bhattacharya, S.P., and Varsha, Apte. (2006). "A Measurement Study of the Linux TCP/IP Stack Performance and Scalability on SMP systems". *First International Conference on Communication System Software and Middleware*, pp 1-10.

Clark, D.D. (1982). "Modularity and Efficiency in Protocol Implementation". *RFC 817*, pp 1-10.

Faulkner, M., and Brampton, S. (2009). "Evaluating the Performance of Network Protocol Processing on Multi-core Systems". *Advanced Information Networking and Applications*, pp 16-23, 26-29.

Finn, G., and Hotz, R.V. (1996). "The Impact of Zero-Scan Internet Checksumming Mechanism". *ACM SIGCOMM Computer Communication Review*, Vol. 26, No. 5, pp 27-39.

Forouzan, B.A.(2007). *TCP/IP Protocol Suite*, 3rd edition, Mcgraw-Hill International, New York.

Kleinpaste, K., and Steenkiste, P. (1995). "Software Support for Outboard Buffering and Checksumming". *ACM SIGCOMM Computer Communication Review*, Vol. 25, No. 4, pp 87-98.

Valerio, C. I. (2012). "Diseño, desarrollo e implementación de un motor de descarga de la suma de verificación del protocol de control de transmission (TCP) para sistemas embebidos que hacen uso del controlador de memoria multipuerto (MPC) y el bus local de procesador (PLB)", M.S. Thesis, Tecnológico de Monterrey, Nuevo León, México.

Wang, W.F., and Wang, J.J. (2005). "Study of Enhanced Strategies for TCP/IP Offload Engines". *11th Internacional Conference on Parallel and Distributed Systems*, pp 398-404.

Xilinx Corporation. (2011). Multiport Memory Controller (MPMC), http://www.xilinx.com/support/ , 01/11/11.

Authorization and Disclaimer

Authors authorize LACCEI to publish the paper in the conference proceedings. Neither LACCEI nor the editors are responsible either for the content or for the implications of what is expressed in the paper.