International journal of basic and applied research

www.pragatipublication.com

ISSN 2249-3352 (P) 2278-0505 (E)

Cosmos Impact Factor-5.86

Technology for Open Source Networking that Integrates Hardware and Software

Dr Madhusudhana Reddy[1,] Dr S Zahiruddin [2] K Pavan Kumar[3],G Sunil Kumar [4]
[1,2] Asst. Professor, Department of ECE, K. S. R. M College of Engineering(A), Kadapa
[3,4] Asst , Professor, Department of ECE, K. S. R. M College of Engineering(A), Kadapa

Abstract—

It will be challenging to develop packet processing software that can keep up with networks moving at tens of gigabits per second. A suitable open-source system is needed to serve as a prototype platform for testing new network capabilities, ideally one that ensures line-rate processing, exact timestamping, and reduced power consumption. Hardware-based solutions like NetFPGA may provide for these requirements in addition to software. The time and effort required to create such an open-source FPGA-based solution is the biggest obstacle to adoption. Now that HLL-based circuit synthesis tools are widely available, hardware-based networking applications may be developed with a more manageable learning curve than was previously achievable when using HDLs. Current open-source hardware-based platforms for networking applications are discussed, along with how they might be fed by the new programming paradigm of FPGAs made feasible by cutting-edge High-Level Synthesis tools. We compared the time and effort needed to build a network flow monitor using traditional hardware development techniques to that of building the same thing with High-Level Languages. The first results seem promising, particularly given the significant reduction in the development time (from months to weeks).

Key words

Network Flow Monitor, High-Level Synthesis, Packet Processing, High-Speed Networks, High-Level Language, and Network Functions on Field-Programmable Gate Arrays.

TRANSITION TO D ATA LINK

As technology develops, communication network capacities increase dramatically. Interfaces of 40 Gbps and even 100 Gbps are becoming increasingly prevalent, while 10 Gbps is currently used in most deployments. In order to carry out a wide range of network tasks, packet processing programs must be placed inside of such high-speed networks. Both security (through firewalls, IDS/IPS, and lawful interception) and network performance (by analysis of delay, jitter, loss, or throughput) are examples of such uses. To keep up with the rapid pace at which applications are evolving, the underlying processing infrastructure must be flexible. Because of the enormous number of available software engineers, the ease of the process, the short development cycles, and the inherent flexibility of software, using software that runs on traditional x86 processors is now the most viable option. The performance requirements of the modern network at the higher data rates are too great to be satisfied by software alone. Open-source software for lightning-fast networks (such Packets Hader, PFRing, or Intel DPDK) relies on high-performance network drivers. They function well at 10 Gbps on the cutting edge of today's commodity hardware. Since the current access speed between applications and network devices is limited, achieving throughputs greater than 10 Gbps consistently without packet losses is difficult. Due to the several hops that each packet must travel, latency may be significant and unpredictable, making it unsuitable for use in applications like high-frequency trading. Finally, since it is conducted in batches rather than on individual packets, software driver-based timestamping causes inaccuracy and jitter. Due to this, these drivers cannot provide low-latency processing at line rates or offer accurate packet timestamping when needed [1]. When software-based solutions fall short in performance, offloading part or all of the packet-processing application to the network device is a possibility. In retrospect, it's easy to see that cutting-edge packet processing devices have always relied on specialized hardware. Many network interface cards (NICs) currently conduct protocol offloading (TCP/IP and IPSec) to relieve software of some of its workload, hence increasing system efficiency. Unfortunately, the limited nature of these systems leaves little possibility for personalization. You can use NetFPGA [2] to make high-performance, open-source hardware while delegating less crucial tasks to software running on an x86 CPU.

WORK ON THE NETFPGA-10G

An example of the rising acceptance of FPGA-based devices for networking packet processing applications is the open-source NetFPGA project. To rapidly prototype new approaches to future network architecture, academic institutions have adopted NetFPGA, a technology initially designed for research and education.

International journal of basic and applied research

www.pragatipublication.com

ISSN 2249-3352 (P) 2278-0505 (E)

Cosmos Impact Factor-5.86

NetFPGA was developed by researchers at Stanford University and Xilinx Research Labs [2]. The second-generation version of NetFPGA-10G [5] is built on a Xilinx Virtex-5 FPGA, four full-duplex 10 Gbps Ethernet connections, and a PCI Express card. The FPGA's internal memory (Block RAMs, with 18 Kbits per block) is complemented by two additional memory banks on the platform, allowing it to serve a broad variety of network applications. The first kind uses 27 MB of high-speed static RAM, which is optimized for speedy lookup tables, while the second type uses 288 MB of low-latency dynamic RAM, which is built for packet buffering. PCI Express Gen 1 lanes allow the board to communicate with a host computer. But the NetFPGA 10G platform just needs a 12 V power supply, so it may function autonomously (i.e., without being coupled to any PCI Express socket); this may be the ideal choice for running line-rate applications with a very low power consumption. In Fig. 1 we can see the overall design of the hardware. The NetFPGA-10G is a valuable open-source platform for researchers developing network applications using FPGAs. The software development community has access to a shared database where they may store and exchange source code, binary files, and other development assets. The Xilinx Embedded Development Kit (EDK) is used to develop software for use with the NetFPGA-10G. Both hardware- and software-centric projects may make use of the EDK, with the former focusing on the FPGA's hardware platform and the latter on the embedded processor's software-centric tasks. The base hardware is made up of things like an Ethernet MAC, PCI Express interface (PCIe), MicroBlaze integrated soft processor, DMA, and custom-built modules. Which hardware cores make it onto the FPGA platform is a choice made by the designer. The NetFPGA-10G relies heavily on its integrated CPU, which controls the configuration of the Ethernet ports and runs the software for the EDK project. NetFPGA-10G projects include software code (such as drivers and user programs) that will operate on the FPGA, while EDK projects are used to design hardware and software for the FPGA.

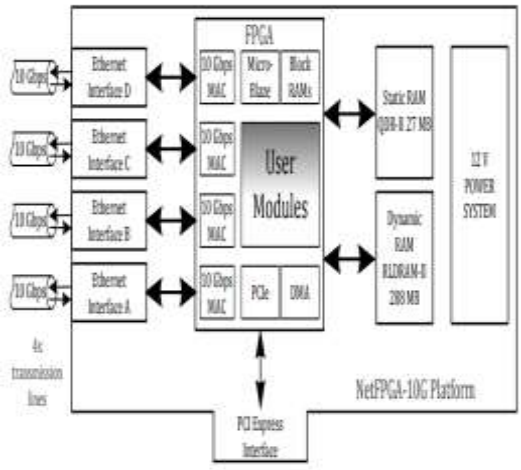

Fig. 1: NetFPGA-10G structure.

manufactured by an x86 processor. The FPGA is connected to the x86 host computer through PCI Express. When put together, they provide the basis for developing open-source, network-based applications that make effective use of today's cutting-edge innovation. As shown in Fig. 2, the first step in a typical design cycle for network applications on the NetFPGA-10G is to download the most up-to-date versions of the available projects from the public repository. (task 1). The designers will need to choose a model that suits their needs well as a basis for the redesign. Finding methods to repurpose components already in existence will save up more time for developing the new functionality. First, you'll need to decide which host computer to use and which FPGA program to run. Making such a call requires finding a happy medium between the time required for development and the reliability of the performance (measured in clock cycles) of each task. If the hardware modules the developer needs are not available in the repository, they will need to build them themselves (task 2.a), which is the most time-consuming part of the HDL design cycle (which also involves Verilog or VHDL encoding and validation). These hardware modules may be produced in whatever quantity necessary to handle the management of all Ethernet interfaces. After all of the modules have been designed or improved, they must be connected using on-chip communication protocols during integration (task 2.b). If required, the last step in building an FPGA embedded system is updating the executable software running on the embedded processor (task 2.c). Designing the host computer actions that are not time critical (task 3) comes after the hardware and

International journal of basic and applied research

www.pragatipublication.com

ISSN 2249-3352 (P) 2278-0505 (E)

Cosmos Impact Factor-5.86

embedded software are ready to run on the FPGA. The NetFPGA-10G repository includes a Linux PCI Express driver that may be used "as is" or changed to meet your specific requirements. It's also possible for user-level programs created in the conventional C/C++ software development cycle to operate in tandem with the aforementioned driver to take on somewhat slow-moving activities. When developers are happy with the design's functionality and have thoroughly tested it (task 4), they may release it to the community.

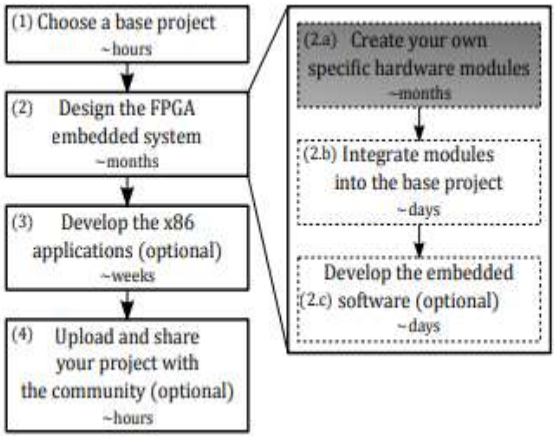

Fig. 2: Typical design flow in the NetFPGA platform.

Here is a breakdown of the development schedule: Task 2.a is the most time-consuming, taking several months (70-90 percent of the total development time). Third, there's Task 3, which, if developed, would take several weeks to accomplish, and Task 2.c, which may take a few days to complete (depending on the application). An skilled engineer may just need a few hours to do the remaining tasks. Last but not least, the cost of software-based innovations is much higher when measured in terms of person months. Due to the nature of the FPGA programming approach, the creation of unique hardware modules (task 2.a) is notoriously time-consuming. A model that incorporates data and time flow (RTL) is utilized at the Register Transfer Level (RTL), the highest level of HDL, to construct circuits. The designer is helped by the fact that HDLs provide a greater level of abstraction than the circuit that actually runs on the FPGA. Hardware registers remain identical to their HDL RTL model counterparts despite HDL synthesis tools translating transfer functions between registers into logic gates. Since HDL codification necessitates resolving a-priori the architecture of the hardware being implemented, the time necessary to build an HDL design is much greater than that required for software solutions. Because of this, FPGAs have been mostly ignored by the networking industry. If we want to take advantage of both software and hardware network improvements, we need to reduce the amount of time spent on task 2.a.

SAVING THE DAY: HIGH-LEVEL LANGUAGES

Use cutting-edge High-Level Synthesis (HLS) to shorten the hardware development cycle. By adapting the FPGAs' programming paradigm, High-Level Synthesis (HLS) technologies enable HLL to be included into the design capture phase. As a consequence, they blur the line between the programming models of a CPU and an FPGA [6]. There are several varieties of High-Level Languages, including graphical descriptions and ad hoc languages that are constructed via the expansion of more standard languages. HLS as a concept has been around for a while [7], but only in the last several years have we seen the introduction of really useful and promising new tools. The electrical industry is making rapid strides toward the general use of these HLS-based solutions. HDL code may be generated from many different source files, including ANSI-C, C++, and SystemC. C/C++ has been widely used because of its many design advantages. For one thing, there is a wealth of already-existing code, which is likely to be familiar to any computer or electrical specialist. C/C++ is widely used as a prototype and development language across various application disciplines. Hardware/software co-design may be rationalized by starting with a software program and porting the parts that need more power to the hardware while keeping the same programming language. Blue Spec's BSC (Blue Spec Compiler), Xilinx's Viv ado-HLS, Synopsys' Symphony C Compiler, Calypte's Catapult C, Impulse's Codeveloper, Jacquard Computing's ROCCC

International journal of basic and applied research

www.pragatipublication.com

ISSN 2249-3352 (P) 2278-0505 (E)

Cosmos Impact Factor-5.86

2.0 (Riverside Optimizing Compiler for Config long-lasting Computing), and Ca dance's C-to-Silicon are also examples.

How HLL may aid in the hardware design process

When using HLL, the first stages of implementation are completed significantly more quickly, and more of the possible design space may be investigated in a shorter amount of time. Figure 3 suggests that software simulation is rapid enough to go to the next iteration of the design process. Therefore, there is no need for the time-consuming and complicated HDL simulation stage.

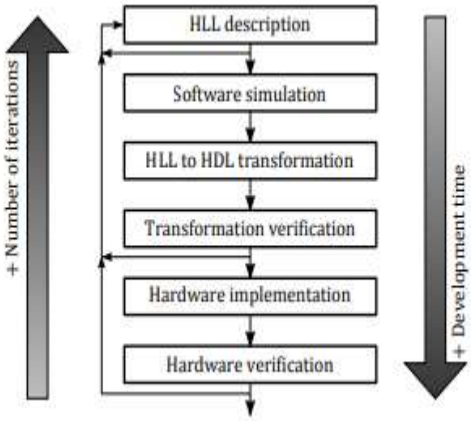

Fig. 3: Hardware design flow using High-Level Langauges.

High-level language (HLL) to hardware description language (HDL) translation, also known as C-to-hardware, is very efficient and provides further information into the hardware's performance (e.g., number of cycles spent in the execution, maximum frequency, area utilization, etc.). Therefore, the designer may try out other design options or add more features (which are more costly to execute using HDLs) and get immediate feedback. Despite its usefulness in shortening the design process, one of the most frequent criticisms leveled with HLS tools is that they limit architects' ability to fine-tune the design at the most fundamental levels. These claims are questionable due to HLL's expressiveness and the shorter development time, as shown by [6], [8]. Therefore, the designers have access to a far broader design space than they would with the HDL approach. It's possible that hardware description languages (HDLs) might provide comparable benefits, but the programming paradigm based on RTL description necessitates the use of a static architecture, which makes it impossible to make changes to the code for the sake of optimization in the future. The designer is then free to focus on system-level performance issues (such as processes communicating with one another or storing data) that have a greater impact on performance as a whole, as HLLs hide all implementation details that aren't crucial to performance (such as optimising state machines, timing closure issues, resource allocation and scheduling, etc.).

The HLL development methodology for building hardware for use in networking software

Since NetFPGA applications need hardware characteristics and parallelism, C/C++ lacks these features. Including these new factors increases the level of complexity. Unfortunately, there is currently not a lot of consistency in how these issues are handled by the various HLS tools. A well-executed hardware design nowadays is not the product of a generic C/C++ code but rather a code that has been tailored for a given architecture.

For this task, we used Xilinx's Viv ado-HLS software [9]. This software can take a model of an algorithm written in C/C++ and output an HDL description that can be used with Xilinx FPGAs (like the one found on the NetFPGA-10G). The Viv ado HLS tool may generate HDL-free hardware cores that can be used in an EDK project without the need for any further coding. Time-accurate circuits may be designed using this tool since it considers both the clock frequency and the target device. Each job's delay is given in terms of elapsed time, or clock cycles. Consequently, there is no jitter in the jobs themselves (for instance, while timestamping packets),

just as there is no jitter in HDL-designed hardware. If the original code does not meet the processing requirements, then directives (#pragma statements) may be used to take use of parallelism, pipelines, control latency, provide interfaces, and other hardware features. When several clock domains are available, the tool still generates a module for each clock domain, but dual-clock FIFOs may be utilized at the EDK level to connect the cores generated for each clock domain. By using HLL, the most difficult and time-consuming part of building the processing logic run on each packet is eliminated. If an application required analysis of all the Ethernet packets it received and delivery of aggregated information to a software layer on the x86 system, a dual-clock FIFO might be used to interface the 10G-MAC clock domain to the DMA domain. To create and ensure the reliability of all user-added intelligence (i.e. processing and communication), HLL model capture will be employed.

CONCLUSION

Compared to solutions based on commodity x86 hardware, the performance and predictability of packet processing systems developed in FPGA are clearly superior. However, most network engineers find it unappealing due to the time and money needed for development. Surprisingly, the advent of new High-Level Synthesis tools offers hope for overcoming these challenges. Current FPGA-based platforms, such as the free and open-source NetFPGA-10G, may benefit greatly from the use of state-of-the-art HLS tools, as we have shown here. In addition, we have described the most significant obstacles that prevent High-Level Languages from gaining general acceptance. FPGAs now have a programming paradigm that makes it possible to capture designs using HLL, cutting down application development time from months to weeks when compared to a conventional hardware development flow based on hardware description languages (HDLs).

This article demonstrates how to create hardware-based network applications without familiarity with HDLs via the production of flow records at 10 Gbps line-rate. In addition, the performance and hardware resource utilisation of solutions developed using a high-level design process were found to be satisfactory. In doing so, the groundwork is laid for an HLL-based (usually C/C++) application framework for packet processing. The framework would further abstract hardware specifics, enabling the traditionally wide gap between software and hardware development in networking applications to be closed.

REFERENCES

[1] V. Moreno, P. Santiago del Rio, J. Ramos, J. Garnica, and J. GarciaDorado, "Batch to the Future: Analysing Timestamp Accuracy of High-Performance Packet I/O Engines," Communications Letters, IEEE, vol. 16, no. 11, pp. 1888 –1891, november 2012.

[2] M. Blott, J. Ellithorpe, N. McKeown, K. Vissers, and H. Zeng, "FPGA Research Design Platform Fuels Network Advances," Xcell Journal, pp. 24–29, 2012.

[3] J.-P. Deschamps, G. Sutter, and E. Cant, Guide to FPGA Implementation of Arithmetic Functions, ser. Lecture Notes in Electrical Engineering. Springer, 2012, vol. 149. [Online]. Available: http://dx.doi.org/10.1007/978-94-007-2987-2

[4] J. Schonw ¨ alder, A. Pras, and J.-P. Martin-Flatin, "On the future ¨ of Internet management technologies," Communications Magazine, IEEE, vol. 41, pp. 90–97, Oct 2003.

[5] "NetFPGA-10G board description," 2012. [Online]. Available: http://netfpga.org/10G specs.html

[6] Xilinx Inc., Introduction to FPGA Design with Vivado HighLevel Synthesis. UG998, July 2013. [Online]. Available: http://www.xilinx.com/support/

[7] G. Martin and G. Smith, "High-level synthesis: Past, present, and future," IEEE Design & Test of Computers, vol. 26, no. 4, pp. 18–25, 2009.

[8] A. Cornu, S. Derrien, and D. Lavenier, "HLS tools for FPGA: Faster development with better performance," in Reconfigurable Computing: Architectures, Tools and Applications. Springer, 2011, pp. 67–78.

[9] Xilinx Inc., Vivado Design Suite User Guide. HighLevel Synthesis. UG902, July 2012. [Online]. Available: http://www.xilinx.com/support/

[10] C. Estan, K. Keys, D. Moore, and G. Varghese, "Building a better NetFlow," in ACM SIGCOMM Computer Communication Review, vol. 34, no. 4. ACM, 2004, pp. 245–256.

ISSN 2249-3352 (P) 2278-0505 (E)

Cosmos Impact Factor-5.86

[11] M. Forconesi, G. Sutter, and S. Lopez-Buedo, "Open source code of nf bram and nf qdr," 2013. [Online]. Available: https://github.com/hpcn-uam/HW-Flow-Based-Monitoring

[12] M. Forces, G. Sutter, S. Lopez-Buedo, and J. Aracil, "Accurate and flexible flow-based monitoring for high-speed networks," Field Programmable Logic and Applications, 2013.