International Journal on Recent Researches in Science, Engineering & Technology (IJRRSET)
A Journal Established in early 2000 as National journal and upgraded to International journal in 2013 and is in existence for the last 10 years. It is run by Retired Professors from NIT, Trichy. Journal Indexed in JIR, DIIF and SJIF.
Available online at: www.jrrset.com

ISSN (Print) : 2347-6729
ISSN (Online) : 2348-3105
JIR IF : 2.54
SJIF IF : 4.334
Cosmos: 5.395

Volume 9, Issue 2 - February 2021 - Pages 40-46

OPEN-SOURCE NETWORKING TECHNOLOGIES THAT FACILITATE THE INTEGRATION OF HARDWARE AND SOFTWARE

Dr. A. ROSI[1], Dr M MUTHULAKSHMI[2], JINU MATHEW[3], MUBEENA ASHRAF[4]
Professor[1], Associate Professor[2], Assistant Professor[3,4]
Department of Electronics and Communication Engineering,
INDIRA GANDHI INSTITUTE OF ENGINEERING AND TECHNOLOGY
NELLIKUZHI P.O, KOTHAMANGALAM, ERNAKULAM (DIST) PINCODE 686691

Abstract

With the advancement of network speeds reaching the range of tens of Gigabits per second, the task of developing packet processing software that can effectively manage such enormous data quantities will become more challenging. Therefore, it is evident that there is a need for a suitable open-source system that can function as a prototype platform for evaluating new network capabilities while ensuring efficient processing, accurate timestamping, and reduced power consumption. Hardware-based solutions such as NetFPGA may fulfil all of these requirements, as opposed to relying only on software. The main obstacle to implementing an open-source FPGA-based solution is the significant amount of time and work required for its development. Thanks to the widespread availability of circuit synthesis tools that are based on high-level languages (HLLs), it is now feasible to develop networking applications that are hardware-based. This may be achieved with a relatively easy learning process, as opposed to using hardware description languages (HDLs) in the past. This article explores the use of state-of-the-art High-Level Synthesis tools to enable the use of FPGAs in programming, which in turn allows for the integration of existing open-source hardware-based platforms for networking applications. We compared the time and effort needed to build a network flow monitor using traditional hardware development approaches with the time and effort needed to produce the same thing using High-Level Languages. The first results are quite promising, particularly considering the significant reduction in the time required for development from months to weeks.

Keywords: Field-Programmable Gate Arrays (FPGAs), High-Level Language, Hardware Description Language (HDL), Packet Processing, High-Speed Networks, High-Level Synthesis, Network Flow Monitor.

Transition to Data Link

Technological improvements are causing a significant expansion in the capacity of communication networks. Presently, installations use connections with a capacity of 10 Gbps, however 40 and even 100 Gbps are becoming prevalent. Packet processing software must be implemented in high-speed networks to carry out various network tasks. Security measures such as firewalls, intrusion detection and prevention systems (IDS/IPS), and lawful interception, as well as network performance, including the analysis of latency, jitter, loss, and throughput, are two examples of areas that may be examined. The processing infrastructure must possess sufficient flexibility to promptly accept application changes. Currently, it is most pragmatic to use software that operates on standard x86 processors because of the abundant availability of software engineers, the straightforwardness of the approach, the quick development cycles, and the inherent flexibility of software. Software-only solutions are currently unable to meet the demanding performance requirements of newer network bit rates. High-performance

International Journal on Recent Researches in Science, Engineering & Technology (IJRRSET)
A Journal Established in early 2000 as National journal and upgraded to International journal in 2013 and is in existence for the last 10 years. It is run by Retired Professors from NIT, Trichy. Journal Indexed in JIR, DIIF and SJIF.
Available online at: www.jrrset.com
ISSN (Print) : 2347-6729
ISSN (Online) : 2348-3105
JIR IF : 2.54
SJIF IF : 4.334
Cosmos: 5.395
Volume 9, Issue 2 - February 2021 - Pages 40-46

network drivers serve as the fundamental building blocks of open-source software designed for ultra-fast networks, such as Packets Hader, PFRing, or Intel DPDK. They exhibit excellent performance at a data transfer rate of 10 gigabits per second on the most advanced and widely available equipment. Attaining throughputs more than 10 Gbps successfully without packet losses is currently problematic due to the limited access speed between applications and network devices. The occurrence of high and unpredictable latency may be attributed to the several hops that each packet has to traverse. This makes it unsuitable for applications such as high-frequency trading. Lastly, software driver-based timestamping results in imprecision and irregularity due to its batch processing approach instead of individually timestamping each packet.

Consequently, these drivers are unable to provide consistent and fast processing at higher rates or accurately date packets when necessary [1]. When software-based solutions do not meet expectations, one alternative is to transfer part or all of the packet-processing programme to the network device. Reflecting on the development of networking technology, it is evident that advanced hardware has consistently been used in cutting-edge packet processing devices. Currently, several Network Interface Controllers (NICs) already delegate protocol activities, such as TCP and IPSec, to enhance system efficiency by assuming responsibilities typically carried out by software. Regrettably, the limited capacity of these systems severely restricts the possibility for customisation. NetFPGA [2] enables the creation of high-performance, open-source hardware, while delegating less significant tasks to software running on an x86 CPU.

Work on the Netfpga-10g

The NetFPGA project, which is open-source, showcases the increasing popularity of FPGA-based devices in the field of networking packet processing applications.
Originally conceived as a research and educational instrument, NetFPGA has gained significant popularity in the academic sphere for its ability to rapidly prototype innovative methods for future network design. NetFPGA was developed by Stanford University and Xilinx Research Labs, with input from the community. The main elements of NetFPGA-10G [5], the updated version, consist of a Xilinx Virtex-5 FPGA, four bi-directional 10 Gbps Ethernet connections, and a PCI Express card. The platform has two distinct memory banks, in addition to the internal memory of the FPGA. These memory banks, known as Block RAMs, have a capacity of 18 Kbits per block. They are specifically designed to accommodate a diverse variety of network applications. Unlike the second kind, which has 288 MB of low-latency dynamic RAM specifically for packet buffering, the first type has 27 MB of high-speed static RAM that is specifically tailored for quick lookup tables. The board has the capability to establish communication with a host computer by using PCI Express Gen 1 channels. Nevertheless, the NetFPGA 10G platform may function autonomously with just a 12 V power source, eliminating the need for a PCI Express connector. This makes it an ideal choice for running line-rate applications with little power use. Figure 1 provides a comprehensive depiction of the arrangement of the hardware. The NetFPGA-10G, being an open-source platform, is valuable for academics aiming to develop network applications that use FPGAs. The developer community has the ability to use a centralised database for the purpose of storing and exchanging code, binaries, and other resources that are utilised in the process of software development. The Xilinx Embedded Development Kit (EDK) is used for developing NetFPGA-10G applications. Projects using the EDK are divided into two distinct categories: a) projects that concentrate on the hardware platform executed on the FPGA, and b) projects that concentrate on the software executed by the embedded processor on the FPGA.
The hardware base is built upon many components including Ethernet MAC, PCI Express interface (PCIe), integrated soft processor (MicroBlaze), direct memory access (DMA), and user-created modules.
The designer determines the selection of hardware cores to be included into the FPGA platform. The

International Journal on Recent Researches in Science, Engineering & Technology (IJRRSET)
A Journal Established in early 2000 as National journal and upgraded to International journal in 2013 and is in existence for the last 10 years. It is run by Retired Professors from NIT, Trichy. Journal Indexed in JIR, DIIF and SJIF.
Available online at: www.jrrset.com
ISSN (Print) : 2347-6729
ISSN (Online) : 2348-3105
JIR IF : 2.54
SJIF IF : 4.334
Cosmos: 5.395
Volume 9, Issue 2 - February 2021 - Pages 40-46

embedded processor is tasked with executing the software of the EDK project and plays a vital role on the NetFPGA-10G by setting the Ethernet ports. An EDK project is used for constructing both hardware and software components for the FPGA. On the other hand, a NetFPGA-10G project includes software code, including drivers and user applications, that will be executed on the FPGA.

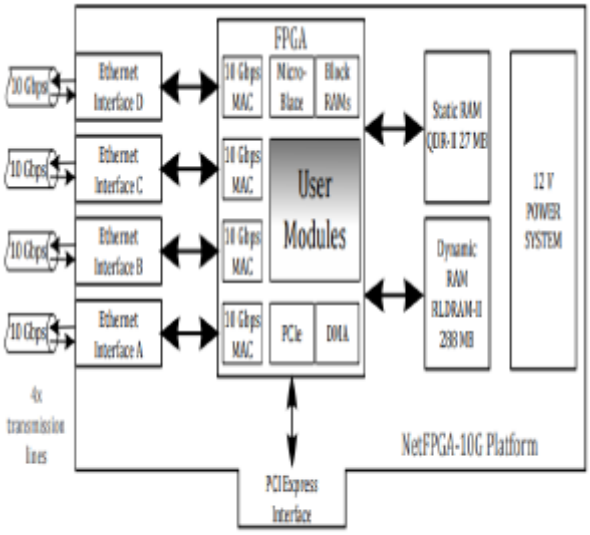

Fig. 1: NetFPGA-10G structure.

PCI Express is used for establishing communication between the FPGA and the x86 host machine. Collectively, these components provide the necessary framework for developing open-source network applications that fully use the capabilities of contemporary technology. The first stage in a typical design cycle for network applications on the NetFPGA-10G is downloading the latest versions of the available projects from the public repository, as shown in Figure 2. Task 1. To initiate the new design, the developers must choose the one that is most suitable for their needs. The objective is to identify methods to repurpose current functionalities, hence allowing for increased allocation of time towards constructing the new entity. Prioritise the selection of the host computer, if applicable, and the FPGA software for first execution. To reach such a conclusion involves finding a compromise between the time it takes to design something and the level of assurance in its performance (measured by the number of clock cycles it takes to execute each task).

After completing the HDL design flow, which involves coding and validating Verilog or VHDL, developers will proceed to create their own hardware modules (task 2.a) if they are not already available in the repository. It is important to note that this step is the most time-consuming in the design flow. In order to control each Ethernet interface, a sufficient number of these hardware modules may be created. After the development or modification of all the modules, the next stage is integration (task 2.b), which entails connecting them together using on-chip communication protocols. If needed, the last step in creating an FPGA embedded system involves modifying the executable programme of the embedded processor (task 2.c). Once the hardware and embedded software have been prepared to run on the FPGA, the next phase in the design process is to create the host computer tasks that are not time-critical (referred to as task 3). A Linux PCI Express driver may be obtained from the NetFPGA-10G repository. It can be used as it is or customised according to your requirements. Furthermore, user-level programmes developed using the conventional C/C++ software development process may be used in conjunction with the described driver to handle jobs that need relatively low performance. Developers

International Journal on Recent Researches in Science, Engineering & Technology (IJRRSET)
A Journal Established in early 2000 as National journal and upgraded to International journal in 2013 and is in existence for the last 10 years. It is run by Retired Professors from NIT, Trichy. Journal Indexed in JIR, DIIF and SJIF.
Available online at: www.jrrset.com
ISSN (Print) : 2347-6729
ISSN (Online) : 2348-3105
JIR IF : 2.54
SJIF IF : 4.334
Cosmos: 5.395
Volume 9, Issue 2 - February 2021 - Pages 40-46

may choose to share their work to the community whenever they are happy with the functionality of the design and have conducted thorough testing (task 4).

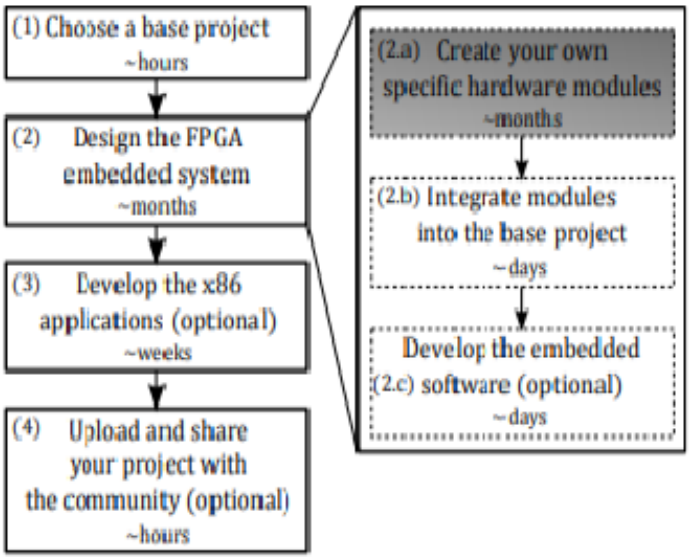

Fig. 2: Typical design flow in the NetFPGA platform.

The following is a detailed analysis of the time needed for development: Task 2.a is the most time-consuming since it takes up a significant portion (70%-90%) of the whole development period, which spans over many months. Additionally, there is Task 2.c, which may need several days to do, depending on the specific application. Furthermore, there is Task 3, which, if executed, would necessitate many weeks to complete. An skilled engineer can likely perform the majority of the remaining tasks within a few hours. Ultimately, the cost of software-based improvements is far higher in terms of the amount of time required by individuals. The FPGA programming approach is characterised by the fact that the creation of bespoke hardware modules (task 2.a) is the most time-consuming aspect of the procedure. The Register Transfer Level (RTL) is the most advanced level of Hardware Description Language (HDL), which is used for constructing circuits by including details about data flow and timing. The designer benefits from the greater degree of abstraction provided by HDLs compared to the circuit that finally executes on the FPGA.

HDL synthesis tools transform transfer functions between registers in the RTL model into logic gates. However, the hardware registers preserve a direct correspondence with their counterparts in the HDL RTL model. Consequently, the amount of time needed to create an HDL design is much more than that needed for software solutions, since HDL coding involves pre-determining the structure of the hardware being developed. Consequently, FPGAs have not been widely used in the networking industry. In order to bridge the gap between software and hardware network improvements and fully benefit from both, it is essential that we reduce the amount of time allocated to task 2.a.

The Importance of High-Level Languages in Resolving Critical Situations

Accelerate hardware development by using contemporary High-Level Synthesis (HLS) techniques. High-Level Synthesis (HLS) tools alter the programming approach of FPGAs, enabling the integration of High-Level Languages (HLL) during the design capture phase. Consequently, they dilute the difference between a Central Processing Unit (CPU) and a Field-Programmable Gate Array (FPGA)'s

International Journal on Recent Researches in Science, Engineering & Technology (IJRRSET)
A Journal Established in early 2000 as National journal and upgraded to International journal in 2013 and is in existence for the last 10 years. It is run by Retired Professors from NIT, Trichy. Journal Indexed in JIR, DIIF and SJIF.
Available online at: www.jrrset.com
ISSN (Print) : 2347-6729
ISSN (Online) : 2348-3105
JIR IF : 2.54
SJIF IF : 4.334
Cosmos: 5.395
Volume 9, Issue 2 - February 2021 - Pages 40-46

programming approach [6]. Various types of high-level languages are available, ranging from graphical representations to improvised languages created by extending traditional ones. While the notion of HLS has been developed over a long period of time [7], it is only in recent years that new tools with great potential and effectiveness have been accessible. Rapid advancements are being made in the electrical industry towards the broad use of these technologies based on high-level synthesis (HLS). Several tools have the capability to process an ANSI-C, C++, or SystemC source file and produce HDL code. There are many factors contributing to the success of C/C++ as a design entry. Firstly, there is a substantial amount of pre-existing code that is well known and familiar to most computer and electrical professionals. C/C++ is widely used for prototyping and development in several application fields, such as networking. Furthermore, it is a rational approach for Hardware/Software co-design, beginning with a software programme and then transferring to the hardware those components that need enhanced performance, all the while using the same language. The following are examples of C-to-Silicon compilers: Ca dance's C-to-Silicon, Synopsys's Symphony C Compiler, Calypte's Catapult C, Impulse's Codeveloper, Xilinx's Vivado-HLS, Blue spec's BSC (Blue spec Compiler), Jacquard computing's ROCCC 2.0 (Riverside Optimising Compiler for Configurable Computing), and blue spec's.

High-level language (HLL)

High-level language (HLL) may assist in the hardware design process by providing a more abstract and intuitive way to describe and manipulate hardware components. HLL allows designers to write code that closely resembles the intended behaviour of the hardware, making it easier to understand and verify the design. Additionally, HLL often includes built-in libraries and tools that can automate some design tasks, increasing productivity and reducing the likelihood of errors.

When using HLL, the first phases of execution are accomplished with greater speed, allowing for a more extensive exploration of the potential design options within a reduced timeframe. Figure 3 indicates that software simulation is sufficiently fast to proceed to the subsequent iteration of the design process. Thus, the time-consuming and intricate HDL simulation step becomes unnecessary.

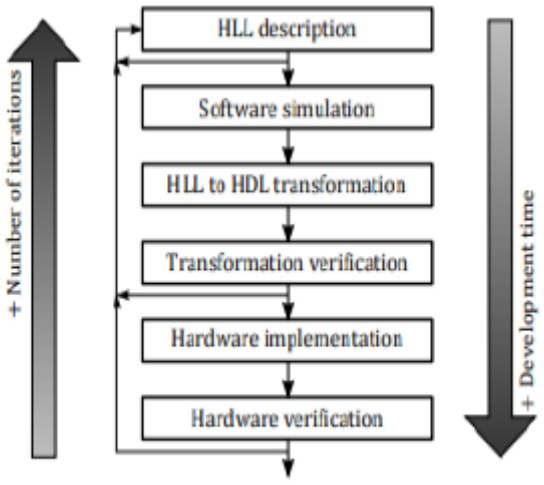

Fig. 3: Hardware design flow using High-Level Langauges.

Hardware description language (HDL) translation from HLL to C-to-hardware provides valuable insights into hardware performance, including execution cycles, frequency, and area utilisation. Using HDLs allows designers to experiment with design options and add features, which are more costly, and get immediate feedback.

One typical issue about HLS tools is that they save design time but hinder performance by limiting architects from fine-tuning the design at the lowest levels. The expressiveness of HLL and lower

International Journal on Recent Researches in Science, Engineering & Technology (IJRRSET)
A Journal Established in early 2000 as National journal and upgraded to International journal in 2013 and is in existence for the last 10 years. It is run by Retired Professors from NIT, Trichy. Journal Indexed in JIR, DIIF and SJIF.
Available online at: www.jrrset.com
ISSN (Print) : 2347-6729
ISSN (Online) : 2348-3105
JIR IF : 2.54
SJIF IF : 4.334
Cosmos: 5.395
Volume 9, Issue 2 - February 2021 - Pages 40-46

development time make these arguments dubious [6], [8]. This allows designers to access a larger design space than using the HDL technique. Although hardware description languages (HDLs) provide comparable benefits, the RTL-based programming paradigm necessitates a static design, preventing future optimisations without code rewrites. HLLs simplify implementation details that aren't crucial to performance, allowing designers to focus on system-level issues like communication and data storage.

How to construct hardware for networking software using HLL technique.

Unfortunately, C/C++ programming language lacks hardware features and parallelism needed for NetFPGA application development. Complexity increases with additional parts. Unfortunately, HLS tools' answers to these issues lack standardisation. Currently, a successful hardware design relies on architecture-specific C/C++ code.

Xilinx's Viv ado-HLS software was used for this project [9]. This application may convert a C/C++ algorithm model into an HDL description for use in Xilinx FPGAs like the NetFPGA-10G. The Viv ado HLS tool generates hardware cores without HDL code for use in EDS projects. Use this tool to construct time-accurate circuits by considering both the clock frequency and the target device. Job delays are monitored in clock cycles and reported. Similar to HDL-designed hardware, tasks function without jitter, such as when timestamping packets. If the original code does not meet processing requirements, directives (#pragma statements) may be used to use parallelism, pipelines, latency regulation, interfaces, and other hardware features. When several clock domains are available, dual-clock FIFOs may be utilised at the EDK level to connect the produced cores regardless of the tool's module creation.

HLL streamlines the complex and time-consuming process of generating processing logic for each packet. A dual-clock FIFO may connect the 10G-MAC clock domain to the DMA domain, allowing applications to evaluate and aggregate Ethernet packets for x86 machine software. All user-added intelligence (processing and communication) will be developed and verified using HLL model capture.

Conclusion

FPGA-based packet processing systems outperform x86-based alternatives in performance and predictability. Most network engineers find it unattractive owing to the time and cost of development. However, new High-Level Synthesis tools provide promise for resolving these obstacles. Current FPGA systems, such the open-source NetFPGA-10G, may benefit from modern HLS tools, as shown here. Additionally, we identified the main barriers to the widespread adoption of High-Level Languages. New FPGA programming paradigms enable HLL capture, reducing application development time from months to weeks compared to HDL-based hardware development.

This article shows how to construct hardware-based network applications without HDL knowledge using flow records at 10 Gbps line-rate. Additionally, high-level design solutions demonstrated good performance and hardware resource utilisation. This lays the basis for an HLL-based (C/C++) application framework for packet processing. By abstracting hardware details, the framework may bridge the gap between software and hardware development in networking applications.

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.

International Journal on Recent Researches in Science, Engineering & Technology (IJRRSET)
A Journal Established in early 2000 as National journal and upgraded to International journal in 2013 and is in existence for the last 10 years. It is run by Retired Professors from NIT, Trichy. Journal Indexed in JIR, DIIF and SJIF.
Available online at: www.jrrset.com
ISSN (Print) : 2347-6729
ISSN (Online) : 2348-3105
JIR IF : 2.54
SJIF IF : 4.334
Cosmos: 5.395
Volume 9, Issue 2 - February 2021 - Pages 40-46

[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]. Gong R, Liu X, Jiang S, Li T, Hu P, Lin J, Yu F, Yan J. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. InProceedings of the IEEE/CVF international conference on computer vision 2019 (pp. 4852-4861).

[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]. Sommer C, Eckhoff D, Brummer A, Buse DS, Hagenauer F, Joerer S, Segata M. Veins: The open source vehicular network simulation framework. Recent advances in network simulation: the OMNeT++ environment and its ecosystem. 2019:215-52.

[11]. Ferrari A, Filer M, Balasubramanian K, Yin Y, Le Rouzic E, Kundrát J, Grammel G, Galimberti G, Curri V. GNPy: an open source application for physical layer aware open optical networks. Journal of Optical Communications and Networking. 2020 Jun 1;12(6):C31-40.

[12]. Basar E, Yildirim I. Reconfigurable intelligent surfaces for future wireless networks: A channel modeling perspective. IEEE Wireless Communications. 2021 Apr 6;28(3):108-14.