From 1G to 10G: Code Reuse in Action

Gianni Antichi
Dept. of Information Engineering
University of Pisa, ITALY
gianni.antichi@iet.unipi.it

Muhammad Shahbaz
Computer Laboratory
University of Cambridge, UK
muhammad.shahbaz@cl.cam.ac.uk

Stefano Giordano
Dept. of Information Engineering
University of Pisa, ITALY
s.giordano@iet.unipi.it

Andrew W. Moore
Computer Laboratory
University of Cambridge, UK
andrew.moore@cl.cam.ac.uk

ABSTRACT

Ever increasing traffic quantities and link-bandwidths force network devices to meet ever-increasing demands; the march to 100G is well under way. The high-speed networking of today is no longer that of five years ago: Unfortunately, such growth contrasts with current financial forces and this leads organisations to find ways to save money. As a result many developers face the common problem: how to make existing, systems reusable in this new, higher-speed scenario? To attack this problem, we propose new, flexible, legacy support mechanics for designs built using System on a Chip (SoC) and System on FPGA (SoFPGA). We illustrate our approach using the widely used, open-source, NetFPGA platform presenting a migration path for existing 1G designs to plugin into the new NetFPGA 10G board without alteration to code structure.

Categories and Subject Descriptors

B.5.0 [**Register-Transfer-Level Implementation**]: General; C.2.m [**Computer-Communication Networks**]: Miscellaneous

General Terms

Design

Keywords

Protocol Bridges, Reusable Hardware, NetFPGA

1. INTRODUCTION

Network speeds increase, demand-for and supply-of bandwidth has meant a move to 100G is now in progress. Even now, 10G is increasingly commodity with high-end motherboards shipping with 10G Ethernet interfaces on them. Despite increasing performance the return on investment is not linear; Moore's law has provided faster (and smaller silicon) but communications has increased at a higher-still rate[11].

Furthermore, FPGA clock rates have remained stable for sometime, with 250-300Mhz being a maximum clock rate on high-end commodity FPGA products for greater than 5 years. Thus, when wanting to manipulate high speed network streams such as those at and beyond 10Gbps, FPGA-based designs must capitalise on the increased gate-count: the designs must become larger and inevitably more complex. This in-turn, lengthens design and development time. None of this is news to the high speed networking community. In contrast organisations are encouraged to do "more with less"[1] and financial pressures have meant that code-reuse is encouraged — if only because of a lack of implementer resource.

Code reuse is not a new concept [12], but the incentive continues to push all developers from those working in the highest level language to the lowest level hardware implementations. From hardware-designers perspective, such an approach is also not new.

The Xilinx and Altera IP stack along with open-source code repositories such as `opencores.org` have utilized or capitalised upon this approach for many years. Code reuse often permits a minimization of the total coding effort: development, debugging, eventual extensions, and support.

Following a keen motivation for code-reuse and a significant existing codebase we propose a flexible legacy support mechanics for designs built using System on a Chip (SoC) and System on FPGA (SoFPGA). We illustrate our approach using the widely used, open-source, NetFPGA platform presenting a migration path for existing 1G designs to plugin into the new NetFPGA 10G board without alteration to code structure.

2. NETFPGA

The NetFPGA [19] is a low-cost reconfigurable hardware-software ecosystem optimized for high-speed networking originally developed by the HPNG (High Performance Networking Group) at Stanford University. It was conceived as a tool for both teaching networking design and a research platform for developing new high performance networking systems. The first generation of the NetFPGA that used Gigabit Ethernet was developed in 2005 [16][23] and since its release this platform has seen wide-spread use by researchers and edu-

[1]"Moore with less" if you prefer.

cators alike. In 2009, efforts started on the development of a next generation of the NetFPGA board [4], one capable of 4×10 Gbps. The platform, now well established in its beta programme is being tested and used by over a hundred users worldwide.

2.1 *1st* Generation

The first generation of NetFPGA boards is a standard PCI card that plugs into a standard PC. The card consists of a Xilinx Virtex-II Pro Field-Programmable Gate Array (FPGA) which is programmed with user-defined logic and has a clock of 125 MHz. The PCI interface connecting the host PC to the NetFPGA is managed by a small Xilinx Spartan II FPGA. Four Gigabit Ethernet ports, 4.5 MiB of static RAM (in 2 banks) and 64MiB of DDR2 dynamic RAM are also on-board.

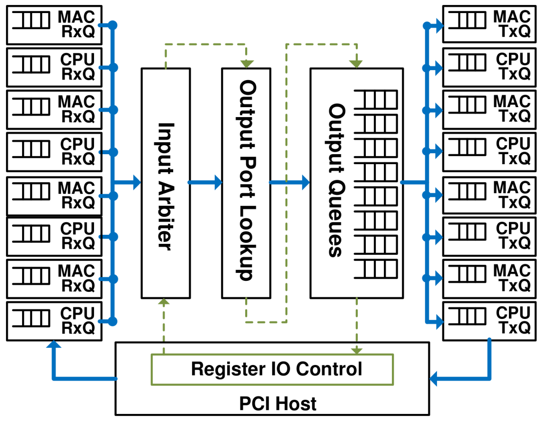

Figure 1: The NetFPGA Reference Pipeline

The reference pipeline, as shown in Figure 1 [8], is partitioned into two planes: (1) Control plane and (2) Data plane. The buses shown in blue (*i.e.*, continuous lines) constitutes the data plane and the ones in green (*i.e.*, dashed lines) form the control plane. Overall, reference pipeline presents eight receive queues, eight transmit queues, Input Arbiter, Output Port Lookup, Output Queues and Register IO Control. The Output Port Lookup is where custom modules like NIC, Switch etc., are inserted. Each "MAC" (Media Access Controller) is a physical network port with an associated queue, and each MAC queue has an associated CPU queue used for communication between the NetFPGA and the host PC. The queues are multiplexed by Input Arbiter into a single packet stream and then de-multiplexed to respective ports according to the Output Port Lookup module.

The data plane is 64 bits wide running at 125 MHz giving a peak bandwidth of 8Gbps. Packets are passed between modules using a non-standard FIFO-like simple pull interface with four signals: `WR`, `RDY`, `DATA`, and `CTRL`. These signals in conjunction are known as *Packet Bus*. The data signal occurs when the destination module asserts its `RDY` signal and in response the source module places a valid data on the `DATA` bus with associated control information *i.e.* header, body, or end of packet on the `CTRL` bus, finally asserting the `WR` signal. The control plane, on the other hand, is a 32-bits wide daisy-chain bus running at 125MHz. It follows a similar protocol like structure to that of the IBM Device Control Register (DCR) bus which was originally used for accessing control and status information in parallel with the primary bus without halting the overall system [10]. The control plane operates over six signals: `REQ`, `ACK`, `RD_WR_L`, `ADDR`, `DATA`, and `SRC` forming a *Register Bus*. The source module places the address and data information on to the `ADDR` and `DATA` bus respectively. It then populates the `SRC` bus with the source ID, used to uniquely identify masters and sets the `RD_WR_L` signal to high if a read operation is requested or low if a write one is requested. Finally, the source module asserts the `REQ` signal till it receives acknowledgement back through the daisy-chain over the `ACK` signal.

2.2 *2nd* Generation

The NetFPGA-10G design embodies much of the original NetFPGA design philosophy. The hardware itself is based upon a Xilinx Virtex-5 FPGA on a PCI Express board providing 4×10 SFP+ Gigabit Ethernet ports, 27 MBs QDRII SRAM, 288 MBs RLDRAM-II, and Two Platform XL Flash (128 MB). Additional I/O capacity is provided through two high-speed QTH Samtec connectors. The combination of wide high-speed memory interfaces as 3×36 bit QDRII SRAM interfaces and 2×64 bit RLDRAM-II interfaces provide an ideal memory solution for most common networking applications. From a platform design perspective, the 10G differs in several significant ways from the 1G platform. For example, the interface standards for hardware components have been completely redesigned, relying instead upon the industry standard AMBA AXI architecture. Additionally, the platform now utilises industry-standard tools for dealing with design composition, automated register mapping, and IP library management[2]. These replace the custom tools and design environment of the earlier platform. By using standard modules and tools there exists a greater opportunity for IP reuse.

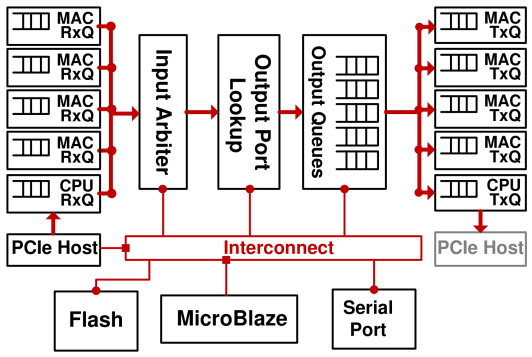

Figure 2: The NetFPGA 10G Reference Pipeline

The modules use Xilinx AXI Streaming Protocol Specification [3] as the standard data-plane interface while the control-plane uses the AXI Lite Protocol Specification. These standards were originally released by ARM as AXI4. Xilinx implements a cut down version that places some application-specific restrictions on the original AMBA AXI4 Specifications. The data plane for the reference pipeline, shown in Figure 2, is similar in most parts to its 1G counterpart, although the control plane has changed significantly. The data plane as illustrated is 256 bits wide and operates at 160MHz

[2]IP, *Intellectual Property,* cores provide the standard atom of construction for the hardware design & synthesis process.

to provide a peak bandwidth of 40Gbps. The control plane is no longer the daisy-chained bus-like model instead it's a complete embedded system with support for interconnects and CPUs. A significant advantage of this system is that it assists in the building of systems where there is a tight coupling between the CPU and data path, e.g., Openflow with integrated on-chip controller. Additionally, in contrast with the 1st generation NetFPGA where only one master - the host system (PCI) - could be used for accessing registers, in the NetFPGA 10G platform both the embedded Microblaze processor and host system (PCIe) can act as master for register accesses.

3. SPANNING 1G TO 10G

Since the introduction of NetFPGA-1G, back in 2006, the research community has published numerous solutions in the field of networks, on-chip architectures and high-speed packet processing and monitoring algorithms capable of handling data at line rate. The most successful and broadly used applications developed using this platform are network interface card (NIC), IPv4 reference router and learning CAM switch, to name the few. Other than being used as a prototype system, it has also extensively been used as a testing framework in various scenarios and environments.

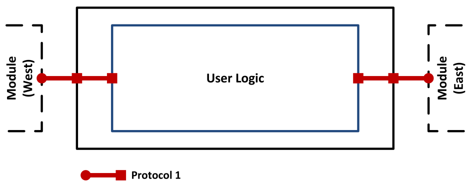

Figure 3: Integration using entire design re-modification

For example, in one arrangement it provides the services of a configurable packet generator tool to test a network by spawning packets of diverse nature varying both in their structure and length. For over six year, NetFPGA-1G has accrued a collection of versatile set of applications contributed by academics and researchers all over the world and has matured significantly. Though it has been a desirable platform for research and development for many years but with recent advancements in contemporary networks, where 10G Ethernet is becoming a commodity, it can no longer keep up with the demand of ever increasing bandwidth. In response to this issue the members of the NetFPGA team have come up with their next generation of networking platform. Capable of handling multiple 10G of data rates and having high density of on-board FPGA fabric and memories, it can answer many of the problems found in the present day network infrastructure.

NetFPGA-10G is a clean-slate program. The architecture resembles in some parts with its previous version but communication protocols are no longer compatible. This hampers the pathway for integrating existing 1G designs to the 10G without having to make significant modifications. Two solutions exists for this problem; either re-modify the whole design using the new communication protocol or implement a bridging logic to overlap the two varying protocols to interoperate, as shown in Figure 3, 4 respectively.

Realizing the latter approach, we present a migration path

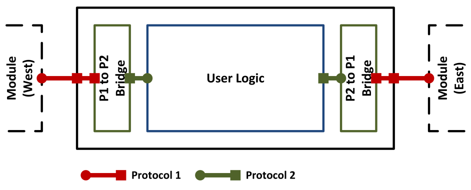

Figure 4: Integration using bridges

for existing 1G designs to plugin into the 10G platform without alterations in the code structure i.e. processing logic and bus protocol etc.

4. ARCHITECTURE

The proposed architecture consists of two bridges, (1) Data-plane bridge and (2) Control-plane bridge, that when plugged into the NetFPGA-10G platform provides a seamless integration path enabling migration of NetFPGA-1G modules. Figure 5 illustrates that the Data plane bridge resides between Input Arbiter and Output Queues module. At the outer boundaries it connects to the AXI Streaming interfaces of the neighboring modules and internally exposes Packet Bus interfaces for the NetFPGA-1G Output Port Lookup (OPL) modules. The Control plane bridge acts as a conduit for moving transactions from an interconnect, (based upon the Xilinx AXI Interconnect [25]), to the NetFPGA-1G daisy-chain. At one end, the bridge attaches itself as a slave to the interconnect and, at the other end, it presents itself as a master on the Register Bus daisy-chain, flowing through the NetFPGA-1G OPL module(s) and terminating at the Control plane bridge, as shown in Figure 5.

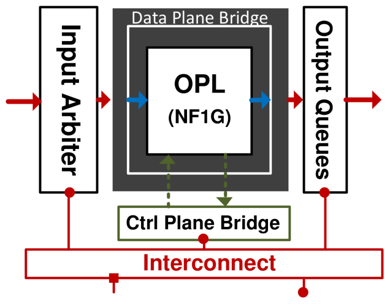

Figure 5: Top Level Architecture

4.1 Data Plane Bridge

The data plane for NetFPGA-10G is a collection of modules connected through AXI4 Stream (AXIS) bus whereas in the case of 1G it's a Packet Bus (PB). Though both of them are streaming interfaces, they differ significantly in their specifications and are not interoperable without the use of coupling logic. Thus, to support 1G module(s) to run on 10G platform we have implemented this coupling logic in the form of bridge for seamless data plane transformation. The bridge operates on packets received from the Input Arbiter (Figure 5,) on AXI Stream slave bus and pushes them out to the NetfPGA-1G OPL module via the master Packet Bus.

For the reverse data-flow the complimentary operations occur: packets are received on the slave Packet bus from

OPL and packets are sent out to the Output Queues on the AXI Stream master bus. Figure 6 shows the bridge logic is composed of two stages: specification conversion and width conversion.

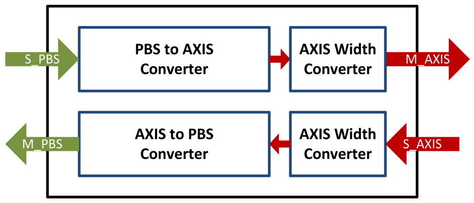

Figure 6: Packet Bus Stream to AXI4 Stream Bridge

The specification conversion stage performs the translation in each direction between the Packet Bus (PB) interface [21] and the AXI Stream interface [20]. The structure and signal mapping of the two interfaces is shown in Figure 7. For the case of PB Slave to AXIS Master, the interfaces conform using a mapping function δ^n where n corresponds to six signals of the AXIS interface. The mapping function describes the semantics for connecting these individual signals. As the Packet Bus and AXIS interface are big endian and little endian respectively, the δ^1 enforces the constraint of having the data bus between each of the interfaces connected in byte reverse order. Similarly, δ^2 and δ^4 specify the correct order of byte and metadata extraction as mentioned in the interfaces specification [21, 20]. The remaining mapping functions are straight forward. The mapping of AXIS Slave to Packet Bus master obeys a similar set of semantic rules represented by a mapping function ψ^n; this is shown in Figure 7.

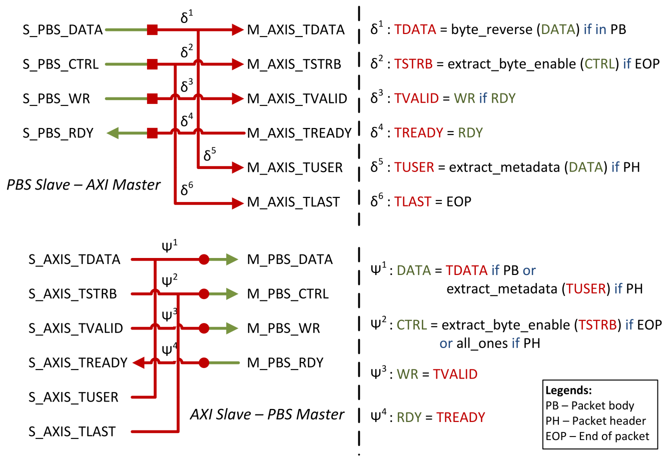

Figure 7: Packet Bus Stream to AXI4 Stream Signaling

The width conversion stage allows configuring the AXIS to PB data width ratio. A ratio of 1:1 means that both the buses have same operational data width, a ratio of 1:2 means that AXIS is twice the size of PB, and a ratio of 1:n where $n \leq 4$ means that AXIS is n times the size of PB. NetFPGA-10G platform natively operates with 256 bits wide data path, (§ 2.2), and NetFPGA-1G by default supports a data width of 64 bits, § 2.1). This property of variable data width configuration in the bridge helps in two different ways when importing NetFPGA-1G modules:

1. NetFPGA-1G pipeline modules with 64 bit data path can be added into the NetFPGA-10G pipeline without modification, while
2. NetFPGA-1G pipeline modules that are fully parameterize over data width can be added to operate at line rate in the NetFPGA-10G pipeline by extending the data width to 256 bits.

4.2 Control Plane Bridge

In contrast with the NetFPGA-1G, the control plane in NetFPGA-10G is no longer a daisy-chain bus-like model instead it's a complete embedded system with support for interconnects, memories and CPUs. This extends the control plane capabilities from being a means for only accessing control and status information to a more complex and flexible system of building designs having contingent constraints on CPU and data plane proximity. Openflow switch implementation with an integrated controller within a single fabric also provides desirable performance or security side-effects[3].

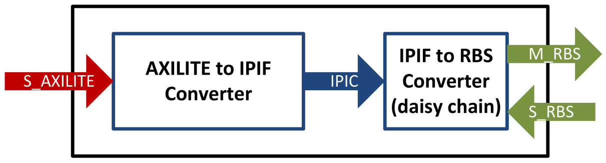
S_AXILITE
AXILITE to IPIF Converter
IPIC
IPIF to RBS Converter (daisy chain)
M_RBS
S_RBS

Figure 8: Register Bus Stream to AXILITE Bridge

Upholding our principle of providing a seamless integration path to permit existing NetFPGA-1G modules be used on NetFPGA-10G, we have implemented a bridging logic for control plane. This is shown in Figure 8. The bridge is composed to two *specification conversion* stages.

The first stage receives transaction requests from AXI Interconnect [3] on its AXI4LITE slave interface, (Figure 5), and converts them to IPIC (IP Interconnect), a slightly simple protocol based upon the Xilinx AXI4LTIE IPIF (IP Interface) core [26]. The semantics and structural transformation for AXI4LITE signals to IPIC using the mapping function α^n where $n \in [1:8]$ are shown in Figure 9.

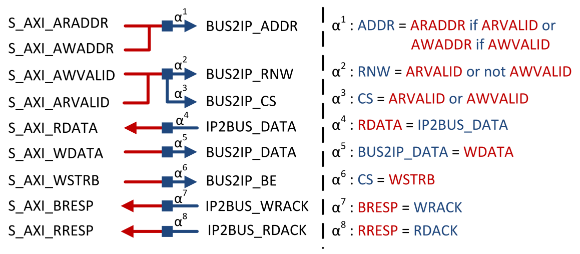

Figure 9: AXI4LITE to IPIC Signaling

The second stage, alongside protocol conversion from IPIC to Register Bus Stream (RBS) [21], also acts as an arbitrator

[3]As might be done under the DARPA MRC programme[4]
[4]http://www.darpa.mil/Our_Work/I2O/Programs/Mission-oriented_Resilient_Clouds_(MRC).aspx

on Master RBS (M_RBS) for IPIC and Slave RBS (S_RBS). It is shown in Figure 8. It operates as a priority arbiter with S_RBS always being served prior to IPIC in case of simultaneous requests. The signaling and structural mapping for IPIC to RBS are shown in Figure 10. Like the first stage, the second stage also implements a mapping function β^n where $n \in [1:9]$ that annotates the semantics for transformation and arbitration among these two protocols. As an example, for the mapping function β^n where $n=1$, the stage assigns `BUS2IP_ADDR` to `M_RBS_ADDR` if the request is from IPIC, indicated by asserting `BUS2IP_CS` signal, and `S_RBS_REQ` signal is low. If the `S_RBS_REQ` signal is high then `M_RBS_ADDR` is latched with `S_RBS_ADDR`, thereby dropping the IPIC request altogether. The remaining functions operate in a similar manner.

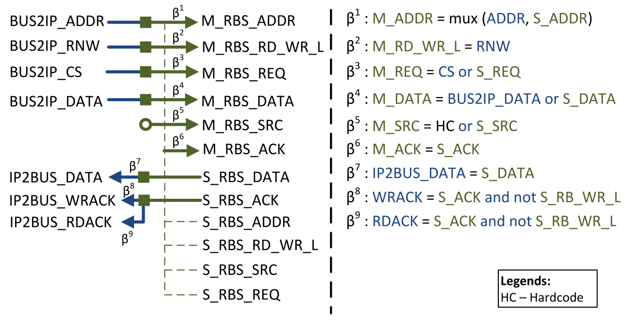

Figure 10: IPIC to RBS Signaling

4.2.1 *Features*

Memory Map: the AXI Interconnect shown in Figure 5, provides a contiguous 4GBs of address space accessible by both the host system via PCIe interface and the on-chip micro-processor, e.g., the Microblaze. The modules such as Memory controllers, Ethernet MACs, Bridges, and CPUs etc., attached to the interconnect are mapped within this address space. In the same way, the control plane bridge connected as slave to the AXI interconnect maps itself in this address space using the parameters C_BASE_ADDR and C_HIGH_ADDR. In the RBS domain, the bridge exposes a contiguous address window (always starting with the base address of zero) for assigning modules in the register pipeline each with a distinct memory range for IO accesses.

Multiple Masters: the NetFPGA-1G control plane, (also referred to as the register pipeline,) is capable of hosting multiple masters each, identified with their unique source identification number (SRC_ID). To conform with this property of the RBS protocol, we have provided the control plane bridge with a unique source identifier which is configured using the C_RBS_SRC_ID parameter inside the core. Whenever a request is received on the S_RBS with SRC_ID $\neq$ C_RBS_SRC_ID, the bridge forwards the request over the M_RBS to the next module in the pipeline, otherwise, it latches the request and sends the response out on the AXI4 LITE interface.

Finite Daisy-chain Depth: the NetFPGA-1G control plane is a set of modules connected together as a daisy-chain. The number of these modules is limited by the control plane bridge using the parameter C_RING_SIZE. The bridge uses this parameter as a timeout value for the wait-timer to expire. This is to avoid the system from going into a live-lock when no acknowledgment is received for the bridge at its S_RBS interface.

5. PERFORMANCE EVALUATION

We evaluated the actual performance of the 1G learning Ethernet switch (ported on the 10G board using the proposed port mechanics) through a variety of experiments. Some tests are taken by means of a flexible high performance traffic generator developed on a commodity multi–core platform [5], which is able to generate up to 10Gb/s traffic, even in the worst case scenario (*i.e.*, 64 byte packet length). We measured the device utilization of our mechanics for the ported switch OPL and tested the maximum number of packets it is able to process without losing data.

5.1 Test Framework

Evaluating the performance of our bridge (*i.e*, using the ported switch) requires the development of a native AXI system in order to perform comparative tests. Therefore, we developed a fully functional layer 2 10*G* learning CAM switch using the Xilinx AXI Streaming Protocol Specification as interface standard. We started from the NetFPGA 10G reference pipeline, (Figure 2), and modified the OPL module accordingly. We set the width of the User Data Path to 256 bits in order to achieve full 40Gbps transport. We implemented the learning CAM feature using the Xilinx Application Note [24]. The 16-entry CAM IP was configured as SRL16-based CAM with a 16 clock-cycle write operation and a single-cycle search operation.

5.2 Device Utilization

Table 1 and 2 show the device utilization of both Data Plane Bridge and Control Plane Bridge in terms of Slices, LUT Flip Flop pairs and Block RAMs of our bridges.

Resources	Virtex 5 TXT Utilization	Utilization Percentage
Slices	3944 *out of* 299520	1.3%
LUT Flip Flop pairs	2630	–
Block RAMs	9 *out of* 324	2.7%

Table 1: Resource utilization of the Data Plane Bridge.

The Data Plane Bridge architecture uses 1.3% of the available slices on the Xilinx Virtex V-TXT FPGA, while the Control Plane Bridge requires less than 0.01%. 2.7% of the available block RAMs are used by the Data Plane Bridge. Most of it, is due to the FIFOs that enable the data width conversion. While NetFPGA-1G support a data plane width of 64-bit, the NetFPGA-10G supports a variable width as described in § 2.2.

Then, we compared (table 3) the device utilization for the OPL of the native AXI Switch with the ported switch. This module is responsible for demultiplexing the single packet stream (*i.e.*, from Input Arbiter) into the right output queue. It carries all the logic that implements the forwarding decisions (*i.e.*, the learning CAM feature). The resource utilization of the ported switch is greater with respect to the native one, since more resources are needed for both OPL module and the Bridge (*i.e*, the module in charge of 1G-to-10G conversion).

Resources	Virtex 5 TXT Utilization	Utilization Percentage
Slices	299 *out of* 299520	0.01%
LUT Flip Flop pairs	176	–
Block RAMs	0 *out of* 324	0%

Table 2: Resource utilization of the Control Plane Bridge.

Project	Slices (%)	LUT Flip-Flop pairs	Block RAMs (%)
Native AXI Switch	0.66%	1302	2%
Ported Switch	1.79%	3496	3.39%

Table 3: Comparison between resource utilization of the native AXI switch OPL and the ported switch.

5.3 Maximum Packet Rate

Figure 11 shows the actual testbed. We equipped a PC (*i.e.*, 2.4GHz Xeon processors, 12-core, 24GB RAM) with an Intel^{TM}Ethernet Server Adapter X520-DA2 and we connected both ports of NIC to the NetFPGA using HP ProCurve 10GbE SFP+ Direct Attach cables.

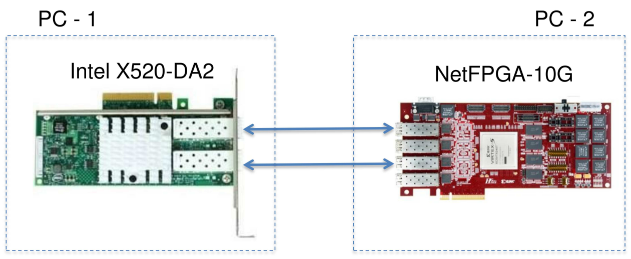

Figure 11: The testbed

In order to assess the performance of our bridge, we carried out some speed tests by using the software traffic generator mentioned above. Such a device is able to completely saturate a 10 Gigabit link with minimum sized packets, thus recreating the worst case scenario for a network device performing packet-by-packet processing; actually we always performed our tests with minimum sized packets. Firstly, we programmed the NetFPGA with our ported switch, then with a native AXI solution to compare results. We used both 10G ports of the Intel NIC to generate an increasing traffic rate up to 20Gbps.

Figure 12 shows the comparative results in terms of throughput. The native AXI switch is able to process all of the packets without losses. This is not surprising, since it was conceived to sustain up to 40Gbps of traffic. On the other hand, the ported switch is able to process all the received traffic without losses when it receives up to 10Gbps of traffic; whereupon it starts dropping some packets. In particular, when the offered rate is 15Gbps it drops the 27.80% of the packets, while at 20Gbps it drops the 43.64%. Since the old-1G OPL module of the switch was not fully parameterizable over data width (*i.e.*, it is 64 bits wide) it could not be extended to operate at line rate (*i.e.*, 40Gbps, considering the four 10G ports of NetFPGA) by increasing the data

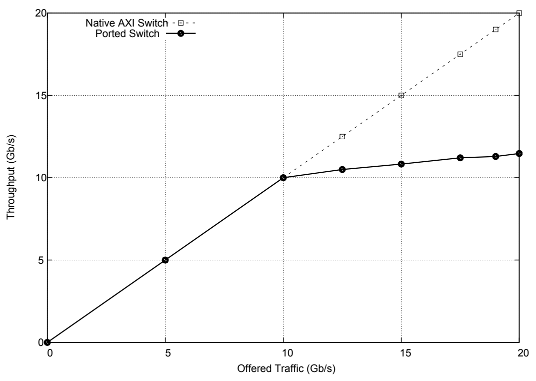

Figure 12: Comparison of the obtained throughput with growing rates of offered traffic.

width to 256 bits. This causes such losses in performances. However, we are able to port it to operate at 10G without losses.

6. RELATED WORK

Reuse and the sharing of hardware IP remains a core practice in system design. In a need to meet aggressive time-to-market objectives, organisations have focused upon two prior areas relevant to this research: the reuse of old IP on new architectures; and the compilation of high level programming languages to FPGA.

Reuse in code is not new, it has been an area of both ongoing interest and active research research [12, 17, 7]. Specific examples include Meng *et al.* [18] who propose a soft IP interface modification methodology (SIPIMM) for systems on Field Programmable Gate Array (FPGA); this purports to minimize the need for interface modification and thus enable interface reuse. In contrast, Rowson *et al.* [22] suggest a new system design methodology that separates communication from behaviour. Code reuse may even be generalised as Design Patterns and through this are also asserted to have a strong heritage and utility in computer network architecture design [9].

A number of approaches may be taken to enable code reuse, including multi-target compilation, and domain-specific languages. The related field of multi-target compilation has given us approaches such as Kiwi of Greaves *et al.* [13] used to transform C# (parallelizable) programs into circuits. Certainly, such an approach helps to make reconfigurable computing technology accessible to software engineers who are willing to write parallel programs however the technology of tools is still rather limited. The need to enhance the development tools available and achieve more abstraction in languages to make hardware development easier for software programmed is also analyzed in [14]. Attig *et al.* [2] introduces PP: a simple high-level language for describing packet parsing algorithms in an implementation-independent manner. It can be compiled to give high-speed FPGA-based packet parsers that can be integrated alongside other packet processing components to build network nodes. In direct contrast, Kobiersky *et al.* [15] utilize an XML description to auto-generate finite state machines for protocol handling at up to 20 Gb/s rates. This is then strongly optimised for resource consumption and speed by an automatic HDL code generator. A predecessor to *PP*, Brebner [6] introduced G, a high-level packet-centric language for describing packet processing specifications in an implementation-independent manner, and demonstrates that G can be compiled to give high-performance FPGA-based packet processing components. A final example, Anwer *et al.* [1] present Switchblade: a platform for rapid deployment of network protocols on programmable hardware.

Given the context of this past work, we present our work as an example that attempts to employ some of the principles of code reuse while meeting our desired goals of work-minimization.

7. CONCLUSIONS

This paper presents a new, flexible, legacy support mechanics for designs built using System on a Chip (SoC) and System on FPGA (SoFPGA). We showed our approach using the widely used, open-source, NetFPGA platform presenting a migration path for existing 1G designs to plugin into the new NetFPGA 10G board without alteration to code structure. A case study is discussed to illustrate how this port mechanics can be efficiently applied on a real-word design. We performed also some comparative tests in order to verify the actual performance of a ported design against a native 10G one. Nowadays, improved code reusability is fundamental in system design and our bridges could help developers to migrate old 1G designs to the new shiny 10G scenario saving a remarkable amount of time.

8. REFERENCES

[1] M. Anwer, M. Motiwala, M. Tariq, and N. Feamster. Switchblade: A platform for rapid deployment of network protocols on programmable hardware. In *ACM SIGCOMM, 2010.*, 2010.

[2] M. Attig and G. Brebner. 400 gb/s programmable packet parsing on a single fpga. In *ACM/IEEE Symposium on Architectures for Networking and Communications Systems, 2011. ANCS'11.*, 2011.

[3] AXI Reference Guide. *http://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf.*

[4] M. Blott, J. Ellithorpe, N. McKeown, K. Vissers, and H. Zeng. Fpga research design platform fuels network advances. *Xilinx Xcell Journal*, (73), 2010.

[5] N. Bonelli, A. Di Pietro, S. Giordano, and G. Procissi. Flexible high performance traffic generation on commodity multi-core platforms. In *Traffic Monitoring and Analysis, 2012. TMA'12*, 2012.

[6] G. Brebner. Packets everywhere: The great opportunity for field programmable technology. In *International Conference on Field-Programmable Technology, 2009. FPT'09.*, 2009.

[7] M. Champman and A. Van der Merwe. Contemplating systematic software reuse in a small project-centric company. In *Saicsit, 2008.*, 2008.

[8] G. Covington, G. Gibb, J. Lockwood, and N. Mckeown. A packet generator on the netfpga platform. In *Field Programmable Custom Computing Machines, 2009. FCCM'09. 17th IEEE Symposium on*, pages 235–238. Ieee, 2009.

[9] J. Day. *Patterns in Network Architecture: A Return to Fundamentals.* Prentice Hall, 2008.

[10] Device Control Register Bus 3.5 Architecture Specifications. *https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/2F9323ECB-C8CFEE0872570F4005C5739/$file/DcrBus.pdf.*

[11] C. A. Eldering, M. L. Sylla, and J. A. Eisenach. Is there a mooreÃIJs law for bandwidth? *IEEE Communications Magazine*, 37(10):117–121, 1999.

[12] W. Frakes and K. Kang. Software reuse research: Status and future. *IEEE Transactions on Education*, 31(7):529–536, 2005.

[13] D. Graves and S. Singh. Kiwi: Synthesis of fpga circuits from parallel programs. In *International Symposium on Field-Programmable Custom Computing Machines, 2008. FCCM'08.*, 2008.

[14] J. Hopf, G. Itzstein, and D. Kearney. Hardware join java: A high level language for reconfigurable hardware development. In *IEEE International Conference on Field-Programmable Technology, 2002. FPT'02.*, 2002.

[15] P. Kobiersky, J. Korenek, and L. Polcak. Packet header analysis and field extraction formultigigabit networks. In *IEEE Design and Diagnostics of Electronic Circuits and Systems Symposium, 2009. CS'09.*, 2009.

[16] J. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo. Netfpga–an open platform for gigabit-rate network switching and routing. In *Microelectronic Systems Education, 2007. MSE'07. IEEE International Conference on*, pages 160–161. IEEE, 2007.

[17] S. McConnell. *Rapid Development: Taming Wild Software Schedules.* Microsoft Press, 1996.

[18] X. Meng, B. ThÃűrnberg, and N. Lawal. Soft-ip interface modification methodology. In *International Conference on Information and Electronics Engineering*, 2011.

[19] NetFPGA. *http://www.netfpga.org.*

[20] NetFPGA-10G Standard IP Interfaces. *https://github.com/NetFPGA/NetFPGA-10G-live/wiki/Standard-IP-Interfaces.*

[21] NetFPGA-1G Reference Pipeline. *http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/ReferenceRouterWalkthrough#Reference_Pipeline_Details.*

[22] J. Rowson and A. Sangiovanni-Vincentelli. Interface-based design. In *Design Automation Conference*, 1997.

[23] G. Watson, N. McKeown, and M. Casado. Netfpga: A tool for network research and education. In *Workshop on Architecture Research using FPGA Platforms*, 2006.

[24] Xilinx CAM Application Note. *http://www.xilinx.com/support/documentation/anmeminterfacestorelement_cam.htm.*

[25] Xilinx LogiCORE AXI Interconnect IP. *http://www.xilinx.com/support/documentation/ip_documentation/ds768_axi_interconnect.pdf.*

[26] Xilinx LogiCORE AXI4-Lite IPIF Data Sheet. *http://www.xilinx.com/support/documentation/ip_documentation/axi_lite_ipif_ds765.pdf.*