Silicom Denmark A/S
FPGA SOLUTIONS

SOLUTION BRIEF

Utilizing OpenNIC, DPDK and SR-IOV on FB4XXVG@Z21D

Author: Lars Munch <lmu@silicom.dk>, Senior Software Engineer

Abstract

This paper presents a solution which utilizes OpenNIC, DPDK and SR-IOV on FB4XXVG@Z21D cards to create scalable network applications on the host computer. Simultaneously, it offloads critical functions, such as PTP, to the embedded Processor System.

Key words: OpenNIC, DPDK, SR-IOV, FB4XXVG@Z21D cards, Time synchronization

Introduction

The FB4XXVG@Z21D card is equipped with a Zynq™ UltraScale+™ MPSoC device which provides a 64-bit quad-core ARM® Cortex®-A53 processor. While this Processor System (PS) is powerful enough to run network applications such as PTP, its performance can be constrained when dealing with high packet load, reaching its processing limits quickly. To mitigate this, the host can assume the responsibility of packet processing for the majority of network traffic. This solution brief outlines the approach taken to address this problem on the FB4XXVG@Z21D card.

Solution overview

In the process of creating the solution, several key considerations were taken into account. The chosen solution should rely on well-established technologies to simplify the tasks for developers and integrators working with the card. Moreover, these technologies should ideally be open source, enabling users to customize and tailor the solution to their specific requirements.

The following three key components were selected:

- The project OpenNIC [1] was chosen to provide a standard Linux network interface to the user.
- The Data Plane Development Kit (DPDK) [2] was chosen to accelerate packet processing workloads.
- Single-root input/output virtualization (SR-IOV) [3] was chosen to partition the traffic into virtual NICs that can be directly utilized by DPDK or a virtual machine (VM).

OpenNIC

The primary objective of OpenNIC, developed by Xilinx/AMD, is to facilitate rapid prototyping of hardware-accelerated network-attached applications. It is important to note that OpenNIC is not a comprehensive SmartNIC solution. Instead, it offers an FPGA-based NIC platform designed for the open source community with well-defined data and control interfaces and is designed to enable easy integration of user logic. It consists of multiple components: a NIC shell, a Linux kernel driver, and a DPDK poll mode driver.

The OpenNIC shell uses the Xilinx/AMD Queue-based Direct Memory Access (QDMA) [4] subsystem. The QDMA subsystem is an optimized DMA engine for PCIe, designed for high bandwidth and packet count data transfers. It offers versatile configuration options, supports memory-mapped and stream DMA, and can operate in interrupt or polling mode. Users can customize descriptors and DMA for complex traffic management.

QDMA relies on instructions from the host OS to move data in both directions (Host to Card and Card to Host), with interface options for AXI4 memory map (MM) and AXI4-Stream. It stands out with its queue-based concept, inspired by high-performance computing interconnects like RDMA, providing low overhead and flexibility. By allocating queues to PCIe Physical Functions (PFs) and Virtual Functions (VFs), a single QDMA core and PCIe interface can be shared across diverse applications.

Extending OpenNIC

By default, OpenNIC does not natively support the Zynq™ UltraScale+™ MPSoC, and this support was subsequently added by Silicom. Additionally, OpenNIC does not inherently support SR-IOV, but the QDMA IP does. Consequently, the OpenNIC shell and drivers were extended to encompass SR-IOV functionality, leveraging the capabilities provided by the QDMA IP.

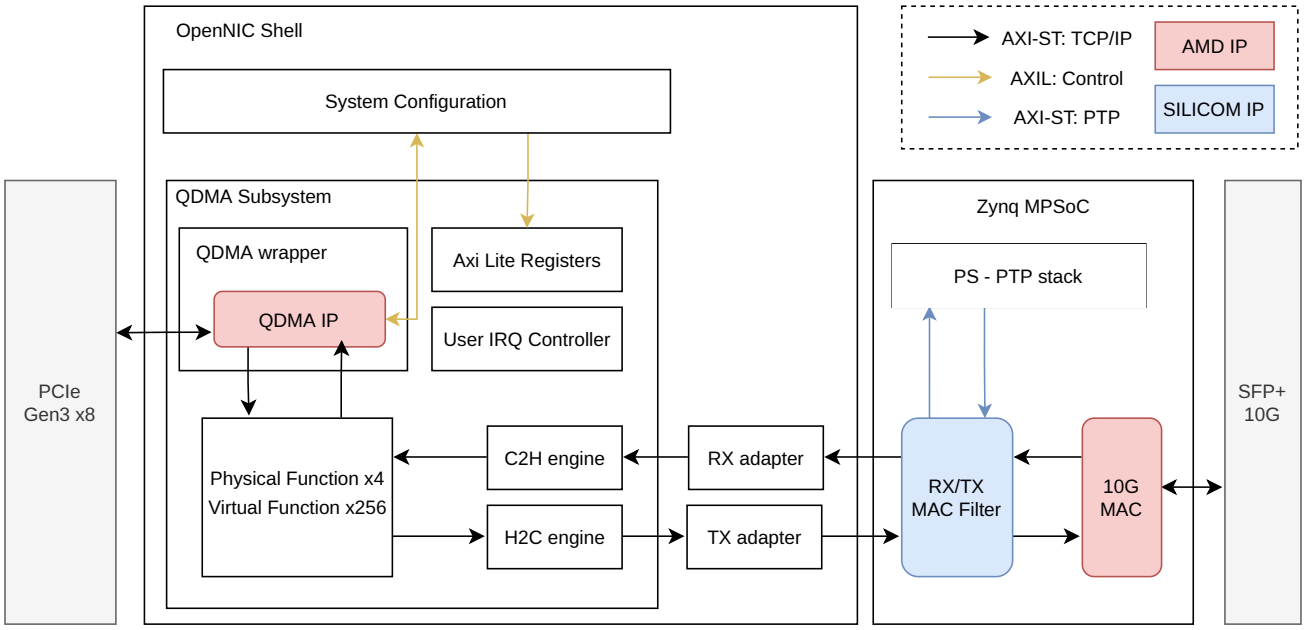

Figure 1. OpenNIC shell integration

A block diagram illustrating the integration of the OpenNIC shell with adaptations from Silicom is depicted above.

The primary adaptation introduced by Silicom is the implementation of the MAC TX/RX filter. The MAC TX/RX filter is a configurable block with several capabilities. The purpose of this block is to act as an interface between the receivers of two streams of data and the Ethernet Subsystem IP. In this case, the two streams of data are the PS and the Host (QDMA subsystem).

The filter can be configured though the Silicom extended OpenNIC Linux driver. In this use case, the MAC destination registers can be programmed to determine the endpoint of a packet. The filter parses the incoming packets and verifies if the MAC destination matches that of the PS. If there is a match, the packet is forwarded to the PS. If that is not the case, it will look up the destination MAC address in a table that associates MAC addresses with queues in the QDMA. The queue can belong to a Physical Function (PF) or, in the case of SR-IOV, a Virtual Function (VF). In the event of multicast or broadcast packets, the packet is forwarded to all endpoints.

Finally, this block partially controls the time stamping of ingress and egress packets. Due to the PTP Timestamp functionality architecture, only the packets from and to the PS are timestamped. By prioritizing traffic to and from the PS, networking applications such as PTP, can be offloaded to the PS while still being able to maintain an accurate PTP clock.

OpenNIC SR-IOV integration

SR-IOV, which stands for Single Root I/O Virtualization, is a technology designed to enhance the efficiency and scalability of virtualized environments, particularly in data center and cloud computing settings. SR-IOV allows for the creation of multiple Virtual Functions (VFs) within a single Physical Function (PF) on a network adapter, enabling more direct and efficient communication between virtual machines (VMs) and physical hardware.

This technology eliminates much of the traditional software overhead that can impede network performance in virtualized environments. Instead of relying on the hypervisor to manage and route data, SR-IOV empowers VMs to communicate directly with the physical network adapter through their allocated VFs, enhancing data throughput and reducing latency.

The introduction of SR-IOV has been pivotal in optimizing network performance, making it an essential component in modern virtualized infrastructures. By enhancing network connectivity and reducing the virtualization layer's impact on data transfer, SR-IOV contributes to the scalability, efficiency, and overall performance of virtualized environments. This introduction to SR-IOV provides a foundational understanding of the technology's significance in virtualization and data center architectures.

A mailbox system, in the context of Single Root Input/Output Virtualization (SR-IOV) and virtualization technologies, serves as a crucial communication mechanism that enables interaction and data exchange between Virtual Functions (VFs) and the Physical Function (PF) of a network or I/O device. This system plays a pivotal role in orchestrating the coordination, control, and data flow within a virtualized environment.

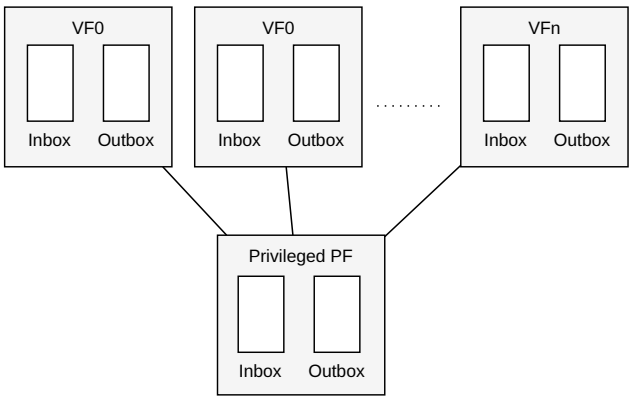

Figure 2. Mailbox communication

Mailboxes are essentially dedicated memory spaces or data structures that VFs and the PF use to transmit messages, commands, and data back and forth. They act as secure and isolated channels for VF-to-PF, and PF-to-VF communication, allowing different virtual instances to communicate efficiently while maintaining data integrity and security.

In practice, VFs and the PF can use mailboxes for various purposes, including configuration updates, request handling, error reporting, and status checks. This bidirectional communication

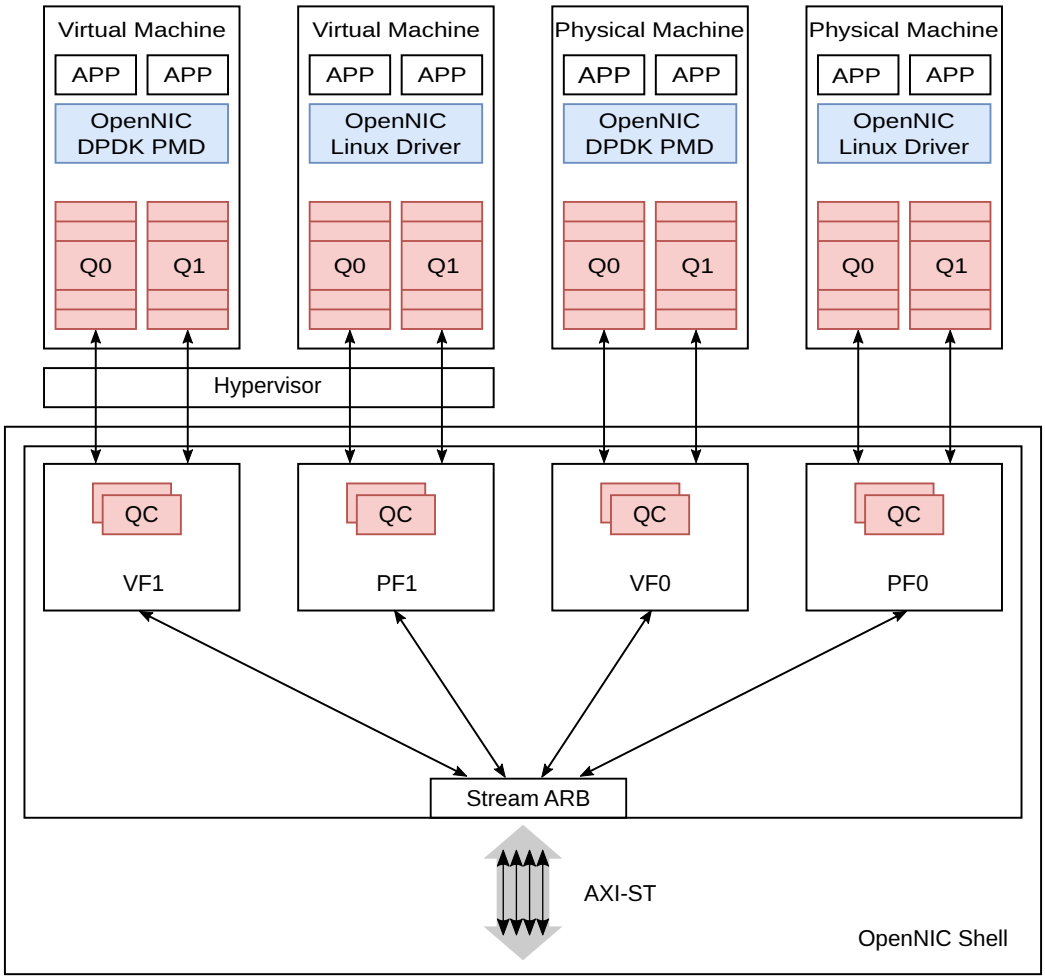

Figure 3. OpenNIC SR-IOV usage combinations

framework ensures that virtualized instances can interact seamlessly with the underlying physical hardware without directly affecting other VFs or the PF.

Overall, the mailbox system is a fundamental component of SR-IOV and virtualization, enabling streamlined communication and data transfer, enhancing system management, and contributing to the overall efficiency and security of virtualized environments.

In a virtualized setup, only the driver linked to a Physical Function possesses sufficient privileges to program and access the QDMA (Quick Data Mover Architecture) registers. Therefore, for functions with lower privileges, specifically all Virtual Functions, must interact with the privileged driver through the mailbox mechanism provided by the QDMA.

Each function (both PF and VF) has an inbox and an outbox that can fit a message. The actual communication API is be defined by the software to facilitate this interaction.

In this solution, the driver linked to a privileged Physical Function is the OpenNIC driver, which, as previously mentioned, does not natively support SR-IOV. Consequently, the OpenNIC driver was extended to support SR-IOV and provide the necessary mailbox communication API. This API is made compatible with the API already provided by the DPDK QDMA/OpenNIC poll mode driver.

The OpenNIC Linux driver is extended to include responsibilities for the following tasks:

- Enabling and disabling Virtual Functions in the QDMA.
- Handling mailbox interrupts.
- Managing queue configuration messages.
- Configuring MAC TX/RX filters based on information from the VF.

In addition to the OpenNIC Linux driver extensions, the DPDK QDMA/OpenNIC poll mode driver is extended to include mailbox messages for MAC TX/RX filter configuration.

It is important to note that while Silicom chose to implement MAC TX/RX filters for directing traffic to the VF queues, this approach can be replaced with a more complex logic, such as a flow table, to enable more advanced traffic steering.

As depicted in Figure 3, adding SR-IOV to OpenNIC offers a wealth of different usage combinations:

- OpenNIC can be used on a Physical Function as a normal Network Interface utilizing the Linux kernel network stack while DPDK can be used to process a high traffic load on the Virtual Functions.
- DPDK can be used to process a high traffic load on the host using a Physical Function while also using DPDK to process traffic from the Virtual Functions directly on the host.
- The Virtual Function can be handled by a virtual machine by utilizing PCI Passthrough using the DPDK QDMA/OpenNIC poll mode driver while still running OpenNIC on the Physical Function on the host.

The flexibility of this solution enables users to tailor the setup to their own specific requirements.

```
ptp4l[501.840]: master offset          0 s2 freq      -5 path delay       178
ptp4l[502.840]: master offset          2 s2 freq      -3 path delay       177
ptp4l[503.840]: master offset          0 s2 freq      -4 path delay       178
ptp4l[504.840]: master offset         -2 s2 freq      -6 path delay       177
ptp4l[505.840]: master offset          2 s2 freq      -3 path delay       177
ptp4l[506.840]: master offset          2 s2 freq      -2 path delay       177
ptp4l[507.840]: master offset          0 s2 freq      -4 path delay       178
```

Listing 1. Output from ptp4l client running on the PS

Performance

The RX/TX performance test focuses on assessing the data reception and transmission capabilities of DPDK on OpenNIC, ensuring that it can handle high volumes of network traffic efficiently. Due to the architecture of splitting the traffic between the PS and the host, it is possible to achieve close to line rate performance while still maintaining an accurate PTP clock on the PS.

All the tests has been performed while running a LinuxPTP [5] client on the PS. From the output of `ptp4l` shown in Listing 1, it can be observed that the master offset remains unaffected by the high traffic load when acting as a PTP client.

The FB4XXVG@Z21D card is able to run at both 10G and 25G. In the following performance tests have been executed on an Intel i5-11400 @ 2.60GHz in a PCIe 3.0 slot using the 25G setup.

RX performance has been tested using the `dpdk-testpmd` tool from DPDK in RX-only mode. Packet sizes are without FCS:

Table 1. RX performance

Packet size	Packages (pps)	Throughput (bps)
64	35,511,450	18,181,862,520
128	20,557,049	21,050,419,152
256	11,159,244	22,854,132,472
512	5,829,947	23,879,463,136
1024	2,981,822	24,427,086,848

The numbers demonstrate line rate performance at all packet sizes. When including FCS, preamble, and IFG for 64-byte packets, we get:

$$35\,551\,450 \times (4+8+12) \times 8 + 18\,181\,862\,520 = 25\,000\,060\,920 bps \quad (1)$$

TX performance has been tested using the ‘dpdk-testpmd‘ tool from DPDK in TX-only mode. Packet sizes are without FCS:

Table 2. TX performance

Packet size	Packages (pps)	Throughput (bps)
64	29,326,786	15,024,511,032
128	20,559,269	21,052,691,496
256	11,160,182	22,856,053,136
512	5,830,017	23,879,752,520
1024	2,981,908	24,427,793,008

As can be seen from the results, line rate is achieved for packet sizes of 128 bytes and above in the TX direction.

Conclusion

In conclusion, this solution leverages OpenNIC, DPDK, and SR-IOV to enhance the capabilities of FB4XXVG@Z21D cards, addressing the challenges posed by high-packet-load network applications. By offloading critical functions such as PTP to the embedded Processor System, this solution empowers the host computer to efficiently process the majority of network traffic.

Key components, including OpenNIC, DPDK, and SR-IOV, were thoughtfully chosen to ensure compatibility with open-source technologies, simplifying integration for developers and integrators. The extension of OpenNIC to accommodate the Zynq™ UltraScale+™ MPSoC and SR-IOV support is instrumental in achieving this solution’s goals.

Performance tests demonstrate the capability of FB4XXVG@Z21D cards to achieve line-rate performance for various packet sizes while maintaining accurate PTP synchronization. The results showcase the ability to handle high volumes of network traffic efficiently.

This solution offers a robust framework for scaling network applications on the host computer while offloading critical tasks to the embedded Processor System, making it a valuable addition to FB4XXVG@Z21D cards in demanding networking environments. It exemplifies how a well-integrated and open-source approach can significantly enhance network processing capabilities.

References

1. OpenNIC Project: https://github.com/Xilinx/open-nic
2. DPDK: https://www.dpdk.org
3. SR-IOV: https://en.wikipedia.org/wiki/Single-root_input/output_virtualization
4. QDMA: https://docs.xilinx.com/r/en-US/pg302-qdma
5. LinuxPTP Project: https://linuxptp.sourceforge.net