Measured Latency Introduced by RF Network-on-Chip (RFNoC™) Architecture

Joshua S. Sunderlin JOSHUA.SUNDERLIN@JHUAPL.EDU
Johns Hopkins University Applied Physics Laboratory, Air and Missile Defense Sector, Laurel, MD 20723 USA

Abstract

Radio Frequency Network-on-Chip (RFNoC™) is a recently developed architecture for Universal Software Radio Peripheral (USRP) Software-Defined Radios (SDR). This new architecture works to leverage the USRP's field-programmable gate array (FPGA) chip to configure digital signal processing (DSP) blocks. A USRP-based SDR application benefits from high adaptability and quicker development time but typically shows high latency. The research in this paper focuses on measuring the latency introduced by the RFNoC architecture in a consistent and repeatable manner to provide a benchmark for SDR applications. By focusing on the *tvalid* interface of the AXI data stream and probing it at different points within the layers of RFNoC architecture, data packets can be tracked and the latency can be quantified and characterized. This study seeks to provide a better understanding of RFNoC's capabilities so that users can determine whether it meets their performance needs and to recommend future improvements by diagnosing the layers of RFNoC where the most latency is incurred.

1. Introduction

Radio Frequency Network-on-Chip (RFNoC) is an open source framework to develop software-defined radio (SDR) applications that can run on FPGA-embedded universal software radio peripheral (USRP) transceivers. RFNoC was developed to speed up and streamline the process of developing signal processing algorithms as well as modularize and seamlessly integrate the majority of non-processing related tasks, such as flow control and data movement, into a common paradigm on the FPGA (Braun et al., 2016). While this higher layer of abstraction of background details is useful for certain applications, including academia and quick algorithm testing and demonstration, it could have deleterious effects on certain SDR applications that involve more intense computational or timing requirements. To determine and benchmark the throughput and timing characteristics caused by the RFNoC framework, a test to calculate the latency propagating through an implementation of working digital signal processing (DSP) blocks in an RFNoC-enabled USRP will be demonstrated and quantified.

One of the primary motivations behind the development of RFNoC was to migrate the heavy processing tasks of an SDR application from the general purpose processor (GPP) of a host computer to the FPGA on the USRP, which is a more capable and appropriate setting for certain radio frequency (RF) signal tasks than the GPP (Braun et al., 2016). Given the complete ability to implement an FPGA in whatever custom configuration is needed for a particular computing task or situation, timing and throughput requirements should be able to be met by the final design in most traditional development settings. Unfortunately, as a tradeoff to this capability, the time needed to properly design and test a given FPGA design is typically longer and more intensive than GPP implementation (Braun et al., 2016; Malsbury & Ettus, 2013). This customization potential of FPGAs also means that the entire backbone of the design must be set by the designer, which can be a daunting task. RFNoC attempts to alleviate this situation by implementing all of these steps as a common framework so that the user can focus on algorithm development inside individual computation engine (CE) blocks, also known as user intellectual property (IP) cores, and then combine them to create the processing signal flow to load onto the USRP (Braun et al., 2016; Malsbury & Ettus, 2013; Braun & Cuervo, 2017; Braun & Pendlum, 2014). This scenario should, in theory, act as a good compromise between time spent creating powerful custom IP cores within the FPGA and time spent bringing a finalized design to fruition.

As RF environments become more congested and networks' data transmission needs become more stringent, the transceivers that are used to propagate and receive signals of interest have to keep up with these ever-evolving scenarios. Typically, designers factor some amount of extra redundancies, such as forward error correction and addi-

Proceedings of the 7^{th} GNU Radio Conference, Copyright 2017 by the author(s).

tional bandwidth, to add robustness to the data link. Even with these allowances, modern data links have increasingly complicated timing and throughput requirements (Truong et al., 2013). It was of interest to determine whether the RFNoC development environment could support these types of applications.

This article examines the magnitude and sources of latency in a USRP SDR platform that utilizes the RFNoC architecture for its signal processing framework. Section 2 provides background information for this experiment, Section 3 explains the approach of this experiment, Section 4 presents the measurement results of this experiment, and Section 5 concludes the paper.

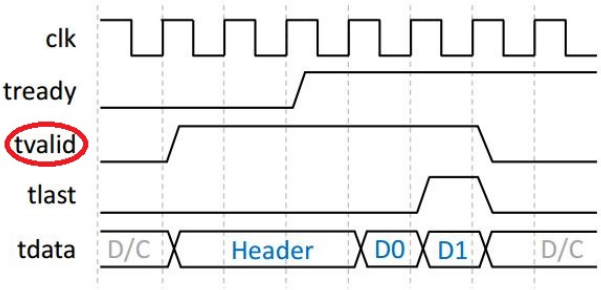

Figure 1. RFNoC Data Stream Structure

2. Background

To measure the latency introduced by RFNoC overhead, the *tvalid* signal of the AXI stream, which indicates when data packets are present, was monitored at different points. By measuring this signal, the latency between blocks, as well as within each block, was quantified in a repeatable manner. When the *tvalid* interface's signal edge transitions to logic high, there is data present at whatever point in the circuit is being scoped (Braun & Cuervo, 2017; Pendlum, 2014). By finding the change in time between different sets of these points, the latency through different parts of the signal processing chain can be measured.

This method also applies as this *tvalid* interface changes variables as the data packet moves into different submodules in the RFNoC block's FPGA code. To illustrate this, these variable names have been overlaid onto a graphic of the RFNoC architecture (Fig. 5). To establish enough data points to follow the data packet through the RFNoC architecture, two CE's were created: Block 1 and Block 2. To start, the time delay between a data packet arriving at Block 1 and the same data packet arriving downstream at Block 2 was measured. After this measurement was made, the data packet was tracked as it moved through Block 1. These results were compared to the same results in Block 2 to observe whether they were consistent. The next sections will cover the different layers of the RFNoC architecture in more detail.

2.1. Crossbar Switch

All of the CE blocks that are incorporated in a signal processing chain are connected via a Crossbar switch. The AXI stream standard is the supported data stream protocol in RFNoC and contains four stream interfaces: *tready*, *tvalid*, *tlast*, and *tdata* (Fig. 1) (Malsbury & Ettus, 2013; Braun & Cuervo, 2017; Pendlum, 2014). It should be noted that *tvalid* stays high for the duration of the *tdata* packet. The interface *tvalid* was chosen because it indicates whether the data packet contains usable information or not. Another advantage of this method is that this single-bit stream interface propagates through the NoC block and will provide a total picture of the latency effect of the RFNoC architecture (Braun & Cuervo, 2017; Pendlum, 2014).

Incoming data packets arrive at CE blocks from previous CE blocks, which are referred to as upstream, in terms of their earlier address from the Crossbar switch. Likewise, outgoing data packets coming from a CE block are sent to downstream CE blocks (Braun et al., 2016; Malsbury & Ettus, 2013). As seen in Fig. 5, the Crossbar feeds the NoC Shell on a data packet's incoming path and the NoC Shell feeds the Crossbar for an outgoing data packet (Braun et al., 2016; Braun & Cuervo, 2017; Pendlum, 2014). More specifically, this incoming data packet is referenced by the variable *i_tvalid* and the outgoing data packet is referenced by *o_tvalid* (Malsbury & Ettus, 2013; Braun & Cuervo, 2017; Pendlum, 2014).

2.2. NoC Shell

As can be seen in Fig. 2, the NoC Shell contains a Clock Crossing FIFO to change the clock frequency between the Crossbar and CE block. It also contains a demultiplexer to parse the incoming data packet into four different parts: data, response, command, and flow control. Likewise, a multiplexer combines these four different parts after they have been processed to repackage the outgoing data packet to be transmitted back out to the Crossbar switch. The variables *str_sink_tvalid* for the incoming data packet and *str_src_tvalid* for the outgoing data packet connect the NoC Shell to the AXI Wrapper (Braun et al., 2016; Braun & Cuervo, 2017; Pendlum, 2014).

2.3. AXI Wrapper

As shown in Fig. 3, the AXI Wrapper strips away the incoming data packet's header information to either be discarded or saved for later use, depending on the AXI stream-

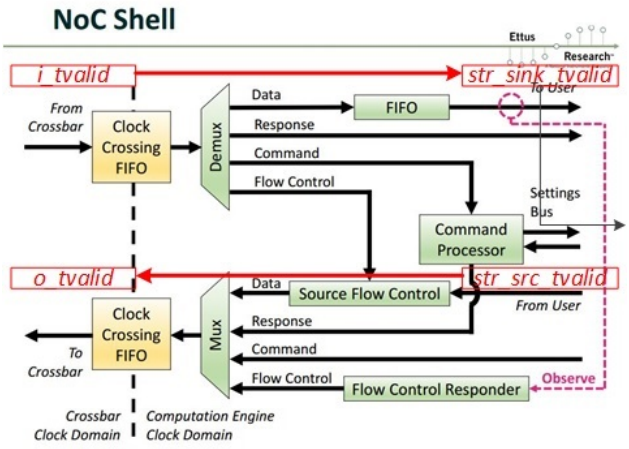

Figure 2. RFNoC NoC Shell Components

ing mode, and then resizes the input data. The outgoing data packet is conversely created by resizing the output data before it is framed and a header is appended back onto the data stream. A settings bus coming from the NoC Shell also goes through a control FIFO to feed control data into the User IP section of the NoC Block. The incoming *tvalid* data interface is expressed by the variable *m_axis_data_tvalid* as raw AXI stream data goes to the User IP and the outgoing data packet is expressed by *s_axis_data_tvalid* once it comes back from the User IP (Braun et al., 2016; Braun & Cuervo, 2017; Pendlum, 2014).

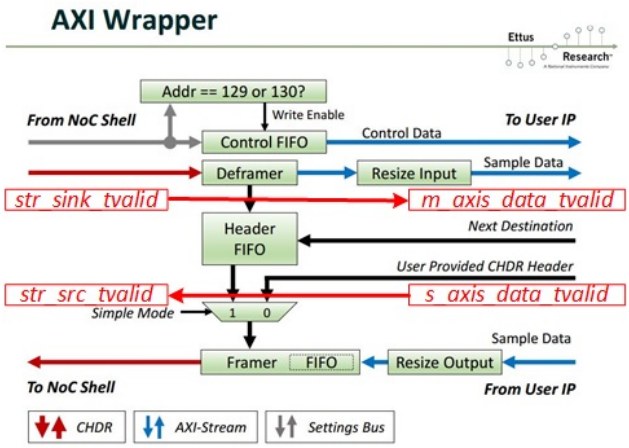

Figure 3. RFNoC AXI Wrapper Components

2.4. User IP

The User IP section of the NoC Block FPGA code is where the user has a chance to implement signal processing algorithms and/or manipulate settings registers to actually make usable signal processing blocks. The incoming data is indicated by the *m_axis_data_tvalid* interface and the outgoing data is represented by *s_axis_data_tvalid*. To take the baseline data reading for latency results, a simple pass-through was implemented where *s_axis_data_tvalid* = *m_axis_data_tvalid*. Because this data was simply passed from input to output without registering it, the latency makes no significant contribution for the purposes of this analysis. Obviously, for more complicated signal processing algorithms that require more clock cycles to complete, the latency at this step should increase. The AXI mode was also set to simple, meaning that each data packet contains a 32-bit data payload, the first 16 bits for in-phase (I) data and the second 16 bits for quadrature-phase (Q) data. This also signifies that the header of the incoming data packet was stored once it had been parsed in the AXI Wrapper and then appended to the outgoing data packet. A larger data payload could be realized by using an extended AXI format to support 64 or 128 bits appended to a new header created and sent from the User IP (Braun et al., 2016; Malsbury & Ettus, 2013; Braun & Cuervo, 2017; Pendlum, 2014).

3. Approach

The GNU Radio Companion (GRC) signal flow that was implemented, illustrated in Fig. 4, includes the Block 1 and Block 2 RFNoC blocks that were described before as well as three other blocks to complete the design that was used for this experiment. The Signal Source block resides on the host computer and is used to start the data stream so that the design is active and the *tvalid* interface can be followed. The Null Sink block also resides on the host computer and just completes the signal path. Along with Block 1 and Block 2, the DmaFIFO block also resides on the FPGA layer of the USRP transceiver. This block is utilized to buffer the Ethernet connection between the host computer and the USRP so that the FPGA layer of the transceiver can ingest data at a high enough rate to support its internal clock frequency settings and prevent underruns which would cause GRC to stop working (Braun & Cuervo, 2017).

4. Results

For the experiment, the host computer used was equipped with an Intel core-i7 clocked at 2.60 GHz, 8GB RAM, USB 2.0, and Gigabit Ethernet, running Ubuntu 16.04 LTS. This was connected over Ethernet to a National Instruments USRP-2953R which was equipped with a Xilinx Kintex-7

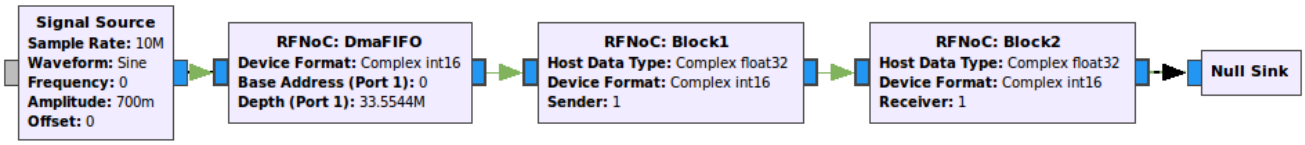

Figure 4. GRC Design and Test Setup

FPGA (XC7K410T). For the software environment, GNU Radio Companion ver. 3.7.11, GNU C++ ver. 5.4.0, and Universal Hardware Driver (UHD) ver. 4.0 were used. An Agilent Technologies DSO-X 3034A oscilloscope, featuring 350 MHz bandwidth and a 4 GSa/s sample rate, was directly connected to the USRP's GPIO connection to take these measurements at a trigger level of 1.5 Volts. Each data collection point in the circuit was measured 100 times to gather the mean and standard deviation and the intra-block and block-to-block latencies were also measured 100 times to confirm the combined latency of all of these data points.

For the purpose of clarifying the results that were found, the latency will be viewed at each stage of the RFNoC architecture in terms of an incoming data path and then in terms of an outgoing data path. The incoming data path will encapsulate a data packet entering a CE block from the Crossbar and end at the point where it enters the User IP. The outgoing data path will begin at the point of exiting the User IP and end with the data packet leaving the current CE block and transmitting back over the Crossbar to the next downstream CE block. This process will be conducted on both Block 1 and Block 2 to determine how consistent the results are and to see if there is any impact because of the upstream/downstream blocks that are used in the GRC signal flow. The latency between a data packet entering Block 1 and entering Block 2 will also be measured which will give insight into the latency over the Crossbar from block-to-block.

4.1. Incoming Data Latency

The respective latency results for the submodules on the incoming data path are:

Table 1. Measured latency values for incoming data path

Measurements		
RFNoC Layer	Mean (μ)	Std. Deviation (σ)
NoC Shell	93.02 ns	2.35 ns
AXI Wrapper	13.01 ns	0.39 ns
User IP	1.85 ns	0.12 ns

4.2. Outgoing Data Latency

The respective latency results for the submodules on the incoming data path are:

Table 2. Measured latency values for outgoing data path

Measurements		
RFNoC Layer	Mean (μ)	Std. Deviation (σ)
AXI Wrapper	1707 ns	0.34 ns
NoC Shell	67.89 ns	2.35 ns
Crossbar	65.39 ns	21.28 ns

4.3. Total Data Latency

There are two larger latency findings of interest from this experiment. Within one CE block, the latency can be determined by the time difference between *i_tvalid* and *o_tvalid*. This result, for both Block 1 and Block 2, was consistently 1.885 μs (σ = 1.67 ns). The next important latency measurement is latency between an incoming data packet arriving at two consecutive CE blocks off the Crossbar which can be determined by measuring the time difference between the first block's *i_tvalid* interface and the second block's *i_tvalid* interface. This time incorporates the first result of 1.885 μs along with the 65.39 ns it takes to send a data packet over the Crossbar from the output of one block to the input of the next block. This was measured to give a complete block-to-block latency result of 1.948 μs (σ = 19.02 ns).

The latency amounts for the intrablock and block-to-block paths were both calculated from the summation of the mean of the individual RFNoC submodules as well as directly measured with an oscilloscope in the manner previously described. This complete breakdown, including both the calculated mean (μ_calc), measured mean (μ_meas), and measured standard deviation (σ_meas), is shown clearly in Table 3.

4.4. Interpretation of Results

From these results, we can determine that the baseline latency for block-to-block data transfer is just shy of 2 μs. That means that each RFNoC block that is instantiated in

Table 3. Calculated and measured total latency values

Total Latency Comparison			
Measurement	μ_calc	μ_meas	σ_meas
Intrablock	1.883 μs	1.885 μs	1.67 ns
Block-to-block	1.948 μs	1.948 μs	19.02 ns

the GRC signal flow chain will, at a minimum, add 1.948 μs to the overall latency of the design. This will have an impact of what types of designs can be supported by RFNoC-enabled USRPs. Another interesting detail of these findings is that the AXI Wrapper layer of the CE block's outgoing data stream dominates the overall latency result. In fact, it represents 90.6% of the latency of one CE block and 87.6% of the latency between two consecutive CE blocks. If this number can be brought down and improved upon, more RF applications could potentially be supported by the RFNoC architecture. All of the other results were on the order of nanoseconds which is a much more favorable result. With the exception of the Crossbar, the variance of each layer is also quite small and negligible to the mean latency results. In context, the FPGA's clock is said to run at an overall rate of 200 MHz which corresponds to a clock cycle of 5 ns. This means that the majority of RFNoC's architecture layers require between 1 and 19 FPGA clock cycles while the outgoing AXI Wrapper takes approximately 342 clock cycles to complete. The complete breakdown of the measured latency values is shown overlaid on the RFNoC block diagram in Fig. 5.

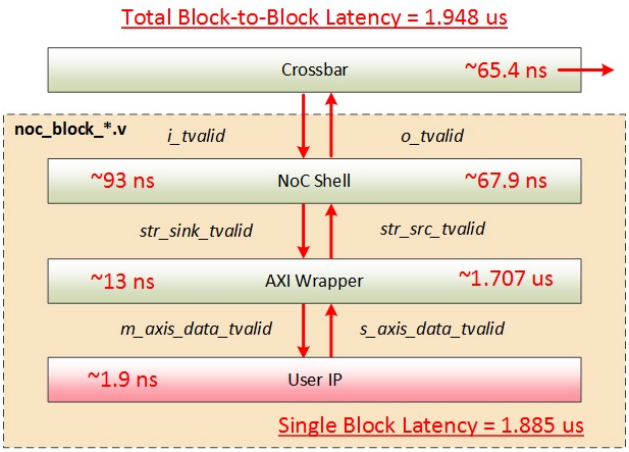

Figure 5. RFNoC Latency Measurements

5. Conclusion

By completing multiple trials to characterize the variance of the measured value, the latency of block-to-block data transmission through the RFNoC architecture overhead has been proven to an accurate figure. By customizing Verilog code, including creating custom IP blocks, a consistent and repeatable manner with which to categorize the AXI data stream bus and share signals between Computation Engines can be derived. Characterizing the timing and throughput of signal flow between these CE blocks is useful to determine whether RFNoC is a compatible environment for a multitude of different RF transmission and reception applications. Given this knowledge, we can get a more complete understanding of the timing constraints that the RFNoC architecture will introduce on an RF design.

From the result seen for the latency measurement, it can be inferred that the RFNoC architecture adds enough delay to block-to-block data transmission that certain real-time RF applications would not be supported. As communication standards become more complicated, their signal processing chains become longer and more computationally intensive. This has the net effect of making each block in the chain more complicated and thus takes more clock cycles to complete the algorithms that they contain. Thus, the latency results that were measured in this scenario must be added to every block that is necessary to implement a design. A recommendation for further study would be to implement a more complicated CE block between Block 1 and Block 2 and measure the latency induced by a more complex algorithm. Another further experiment would be to implement a full signal processing chain for a specific communication receiver and decoder to see if it could support the necessary timing needs.

As RFNoC is open source and constantly being improved, the block-to-block latency may reduce over time as the code base changes. Specifically, improvements to the outgoing AXI Wrapper path could yield very valuable decreases in overall latency within RFNoC. Hopefully, with the aid of this latency analysis, these improvements can be implemented so that the raw processing power of the FPGA on the USRP can be optimized to the fullest extent possible. By reducing the latency number, RFNoC will become more appropriate for a larger variety of SDR applications.

Acknowledgment

This work was sponsored by the Defense Advanced Research Projects Agency under Contract No. HR0011-12-D-0001-0050. The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

The author would also like to acknowledge the feedback and mentoring of multiple colleagues at the Johns Hopkins University Applied Physics Laboratory, including David Rouse, Thomas Drake, Oscar Somerlock, Matt Gordon,

Dan Chew, and Sam Berhanu.

References

Braun, Martin and Cuervo, Nicolas. Getting started with rfnoc development, 2017. URL https://kb.ettus.com/Getting_Started_with_RFNoC_Development. Ettus Research LLC.

Braun, Martin and Pendlum, Jonathon. Rfnoc: Rf network on chip, 2014. URL https://www.ettus.com/content/files/RFNoC_Wireless_at_VT_Intro.pdf. Ettus Research LLC.

Braun, Martin, Pendlum, Jonathon, and Ettus, Matt. Rfnoc - rf network-on-chip. In *Proceedings GNU Radio Conference*, 2016.

Malsbury, John and Ettus, Matt. Simplifying FPGA design with a novel network-on-chip architecture. In *Proceedings of the Second Workshop on Software Radio Implementation Forum*, SRIF '13, pp. 45–52, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2181-5. doi: 10.1145/2491246.2491251. URL http://doi.acm.org/10.1145/2491246.2491251.

Pendlum, Jonathon. Rfnoc deep dive: Fpga side, 2014. URL https://www.ettus.com/content/files/RFNoC_Wireless_at_VT_FPGA.pdf. Ettus Research LLC.

Truong, Nguyen B., Suh, Young-Joo, and Yu, Chansu. Latency analysis in gnu radio/usrp-based software radio platforms. In *IEEE MILCOM*, 2013.