Pharos: Bandwidth Monitoring System User Manual

Arzhang Rafii

June 2020

1 What It Is

Pharos is a performance monitoring instrumentation for multi-FPGA systems. The bandwidth monitoring core is a set of instrumentation that snoops on AXI-streams and provides certain information about the passing traffic. The instrumentation is designed so that if does not affect the passing traffic, i.e. will not back pressure. Bandwidth monitor outputs counts the number of flits that are transmitted during a certain measurement period. It also provides the average size of the AXI-stream packets that are transmitted from or to the port that is under inspection. There are three cores within every bandwidth monitor core: a) monitorizer b) packet snooper c) packet size EMA. The next sections of this document will discuss these cores in more detail.

2 How It Works

Figure 1 shows a top-level view of the bandwidth monitor IP. The AXI-Stream that is under inspection is to be connected to the input port snooper_in_stream. This port is of the *Monitor* type. This means that it will simply monitor the AXI stream traffic and will not affect it (the same way Xilinx ILAs monitor signals). The duration of measurement is controlled by the *measure* signal. Measurement is started when measure goes high and is stopped when it is set to low. The outputs *number_of_flits* and *cycle_count* will only be available after the measurement is stopped (at the negative edge of the measure signal). number_of_flits is a measure of how many flits are transmitted on the line we are snooping on. *cycle_count* indicates the duration of measurement in units of clock cycles. The inputs *read_frequency* and *Beta* are used by the packet average calculator (See section 2.3 for more details on this). The I/O ports are described in Table 1 in more detail.

Figure 2 shows the expanded IP as it is instantiated in Vivado. Bandwidth monitor is made out of three main cores. *monitorizer* provides an AXI-Stream monitor port, *packet_snooper* measures the number of flits and the packet size of each individual packet, and *packet_size_EMA* calculates an exponential moving average of the size of the transmitted packets. The next 3 sub-sections describe these cores in more detail.

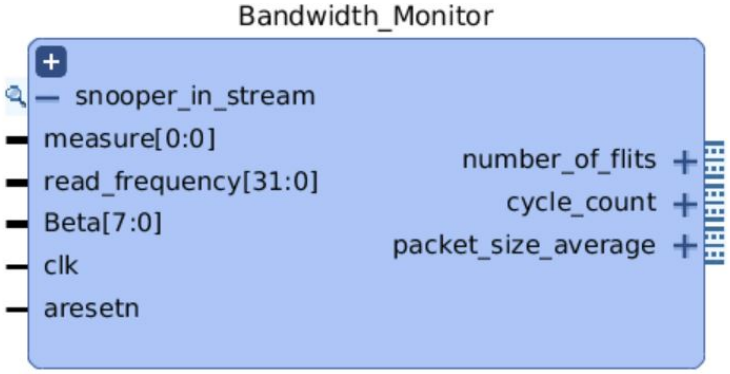

Figure 1: Bandwidth monitor top level core

2.1 Monitorizer

This core has two purposes. The first purpose is to provide a the bandwidth monitor a *Monitor* type AXI-Stream input. This is done to prevent the bandwidth monitor from affecting the passing traffic. Using an AXI-Stream

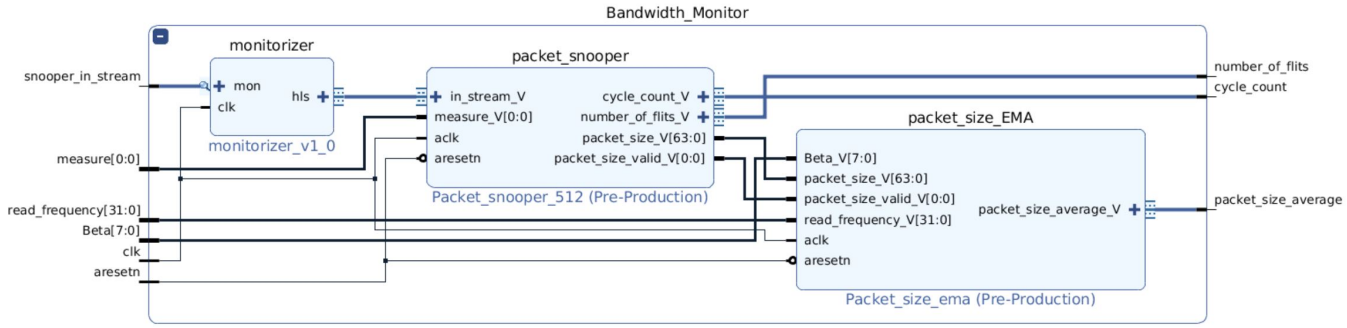

Figure 2: Bandwidth monitor top level core

Table 1: Bandwidth monitor I/O ports

Port Name	Direction	Port Type	Description
snooper_in_stream	input	AXI-Stream	input AXI-stream (the AXI-stream traffic that is under inspection)
measure	input	1-bit raw wire	enable signal used to start and stop the measurement
read_frequency	input	32-bit raw wire	output enable for the packet_size_average output (see section 2.3)
Beta	input	10-bit raw wire	smoothing factor for the exponential moving average (see section 2.3)
aclk	input	1-bit raw wire	clock signal
aresetn	input	1-bit raw wire	asynchronous active-low reset signal
number_of_flits	output	AxI-Stream	number of flits that are transmitted in the duration of measurement
cycle_count	output	AXI-Stream	duration of measurement (number of cycles that the measure signal was high
packet_size_average	output	AXI-Stream	average packet size of all the packets (outputted at read_frequency)

monitor will ensure there is no back pressure on the AXI line by the performance monitoring instrumentation. The second purpose is to detect an AXI-Stream transaction. An AXI-Stream transaction is done when the valid (of the sender) and ready (of the receiver) signals are both high. Monitorizer core ANDs the valid and ready signals of the input AXI-Stream and feeds it to the valid signal of the *packet_snooper*.

2.2 Packet Snooper

packet snooper receives its AXI-Stream input from the *monitorizer* core. As described in section 2.1, the valid signal of this AXI-Stream indicates a transaction on the AXI-Stream we are inspecting. When the input measure signal goes high, packet snooper counts the cycles in which the valid signal of the input AXI-Stream is high to determine the number of transmitted flits. The number of flits and the number of cycles the measure signal was high are outputted on the negative edge of *measure*.

Packet snooper also inspects the keep signal of the input AXI-Stream to determine the size of each transmitted packet. This is also done only when measure singla is high. Whenever the last flit of the packet is detected (last channel of AXI-Stream), the packet size is sent out to the *packet_size_EMA* core. This signal is accompanied by a valid, *packet_size_valid*.

2.3 Packet Size EMA

The purpose of this core is to calculate an exponential moving average (EMA) of the packet sizes. The average calculation is done as fixed-point calculations. The exponential moving average is calculated by the following formula:

$$EMA = (\beta)(current\_packet\_size) + (1 - \beta)(EMA_{previous}) \quad (1)$$

The variable Beta is the smoothing factor of the EMA. It can be chosen to be between 0 and 1. The closer it is to 1, the more the EMA follows new inputs, and the smaller Beta is, the more EMA follows its history. Beta is an 8-bit fixed-point variable, with one integer bit. It is defined in HLS as:

```
typedef ap_ufixed<8,1> ap_fs;
ap_fs Beta;
```

Beta is meant to be inputted as a fixed-point value. It must be between 0x00 and 0x80. Table 2 provides some examples of input values and their equivalent Real values.

Table 2: Examples of the Beta variable input

input value in binary	**input value in base hex**	**equivalent real value**
00000000	0x00	0
00010000	0x10	0.125
01000000	0x40	0.5
01110000	0x70	0.875
10000000	0x80	1

A new packet size average is calculated in every cycle that *pacet_size_valid* input is high. $EMA_{previous}$ holds the previous calculated average. Packet size average is outputted at certain intervals determined by the input *read_frequency*. For example, if *read_frequency* is set to 1000, then a new average is outputted every 1000 cycles.