Proof of bitrate equations

This repository includes two cores. First is a packet generator that allows the user to generate packets at a specified rate. Second is a bandwidth throttler that allows you to selectively set a maximum bandwidth for an AXI Stream port. This employs backpressure to enforce the bitrate that crosses it is strictly less than the amount specified (and exactly equal if the output applies no backpressure and the input is always ready).

In this document, we prove the equations used for converting between tuning parameters and bitrate achieved. We conclude with:

Packet Generator:

Given:

- Tuning parameters are the Penalty (PE) and Scaling Factor (SF) [default 1000]
- Width of the AXI Stream line is DW bits
- Packets sent are PS bits long (must be a multiple of 8 to be valid in AXI Stream)
- The header added after this core is HS bits long
- The clock rate is CR measured in MHz
- The bandwidth (BW) is given in Mbps

We can calculate the Bandwidth from the Penalty or vice versa as:

BW = $\frac{CR*SF*(PS+HS)}{\left\lceil\frac{PS}{DW}\right\rceil*(PE+SF)}$, PE = $\frac{CR*SF*(PS+HS)}{\left\lceil\frac{PS}{DW}\right\rceil*BW} - SF$

*Note $\left\lceil\frac{A}{B}\right\rceil$ denotes the ceiling integer result of the division function.

Bandwidth Throttler:

Given:

- Tuning parameters are the Penalty (PE) [live tunable] and Scaling Factor (SF) [default 1000]
- Width of the AXI Stream line is DW bits
- The clock rate is CR measured in MHz
- The bandwidth (BW) is given in Mbps

BW=$\frac{CR*DW*SF}{PE+SF}$, PE=$\frac{CR*DW*SF}{BW} - SF$

Algorithm

The algorithm goes as follows, we have two tuning parameters, the Scaling Factor(SF) and the Penalties parameters (PE), both exclusively integers which influence the acceptance function A(t). The behavior of A(t) is as follows:

$$A(t) = \begin{cases} A(t-1) + PE, & if\ data\ is\ sent \\ \min(0, A(t-1) - SF), & if\ data\ is\ not\ sent \end{cases}$$

Where we will send data if the recipient can accept data (tReady is set) and one of the following conditions is true:

(1) A packet is in the middle of being sent or
(2) A(t) $\leq$ SF

The first rule is to ensure that packets are bundled together and instead we use the space between packets to toggle the bitrate.

In the case of the bandwidth throttler we say that if the above conditions is true then we allow a packet to pass if one is available at the input. Otherwise data is not sent if there is nothing to send.

Packet Generator Proof

Note that if there is no backpressure as is typical in networks, A(t-1)-SF will always be greater or equal to 0. The min condition only exists so that A(t) is never negative and will simply reset to 0 if the network is inactive for a while. An alternative way to look at A(t) is that it decays at a rate of SF every cycle to 0 and then grows by SF+PE if data is sent. As A(t) will grow with every packet and then decay to a sub SF value, we can say that over a long time, on average A(t) will not change. If we set PD equal to the proportion of cycles when data is sent, then we can calculate PD as:

$\mathrm{PD} * (PE + SF) - SF = \Delta A(long\ time) = 0,\ \mathrm{PD} = \frac{SF}{PE+SF}$// (eqn 1)

Let us temporarily assume that the packet size is a multiple of the data width of the bus (DW) and there is no header (header size or HS equals 0). On average on PD proportion of cycles we will send DW bits of data. On all other cycles we will send no data. Therefore on average each cycle we will send DW * PD bits of data. If we let CR be our clock rate in Hz, the bandwidth (BW) in bits per second is given by:

BW = CR * DW * PD = $\frac{CR*DW*SF}{PE+SF}$ (eqn 2)

Let us remove the assumption that the packet size is a multiple of the data width of the bus. In this case there may be so called garbage bits which are sent on an active flit but don't actually contain useful data. The number of clock cycles to send each packet equals $\left\lceil \frac{PS}{DW} \right\rceil$ at which time only PS bits of useful data is sent, but a total $\left\lceil \frac{PS}{DW} \right\rceil * DW$ bits have been sent, the rest being garbage. The proportion of useful bits (PU) can therefore be calculated as $PU = \frac{\sum useful}{\sum total} = \frac{PS}{\left\lceil \frac{PS}{DW} \right\rceil * DW}$ (eqn 3). To calculate the useful bitrate we combine eqn2 and eqn3 to give us:

$BW = BW_{eqn2} * PU = \frac{CR*DW*SF}{PE+SF} * \frac{PS}{\left\lceil \frac{PS}{DW} \right\rceil * DW} = \frac{CR*SF*PS}{\left\lceil \frac{PS}{DW} \right\rceil * (PE+SF)}$(eqn4)

* note that in the case that PS is a multiple of DW, $\left\lceil \frac{PS}{DW} \right\rceil = \frac{PS}{DW}$ which makes BW_{eqn2} equivalent to BW_{eqn4}, which makes sense as there are no garbage bits in this scenario.

Lastly let us remove the assumption that there is no header. Lets find PG, proportionally how much larger is the packet if we add a header assuming there is a header that is added after the packet generator. This header is HS bits long. For every PS bits that the generator sends, PS+HS bits are actually being sent after adding the header. This represents a proportional growth in bandwidth of $PG = \frac{PS+HS}{PS}$ (eqn 5) times greater. Factoring this into eqn4 we get the following final relationship:

$BW = BW_{eqn4} * PG = \frac{CR*SF*PS}{\left\lceil\frac{PS}{DW}\right\rceil*(PE+SF)} * \frac{PS+HS}{PS} = \frac{CR*SF*(PS+HS)}{\left\lceil\frac{PS}{DW}\right\rceil*(PE+SF)}$ //(eqn6).

Note if there is no header we can set HS equal to 0 at which point PG equals 1. In this configuration eqn4 and eqn 6 are equivalent.

We will use this equation to calculate the bit rate for a given penalty and SF. Inversely, we can set the SF and calculate the Penalty for a particular desired bandwidth as:

PE = $\frac{CR*SF*(PS+HS)}{\left\lceil\frac{PS}{DW}\right\rceil*BW} - SF$.(eqn 7)

Note PE must be an integer so it is possible we need to round eqn7 giving us a slightly different bandwidth.

Note the reason the SF parameter is variable is that our hardware needs to hold A(t) in an N bit long data unit. Let PSM be the largest PS that will be tested and PEM the largest Penalty value. As A(t) resets to a value smaller or equal to SF after each packet, the highest possible value for A(t) is:

$A_{max} = PSM * PEM + SF < 2^N$(eqn 8). Therefore selecting a high SF gives a better resolution as PE must be an integer but PEM will grow proportionally with SF (one can show multiplying both PE and SF by any positive integer achieves the same bandwidth thus a big SF will require big penalties). The user can select SF by examining the lowest bandwidth they need to test (which has the highest penalty) and finding the the highest SF where A_{max} is still bounded by 2^N. This maximizes the resolution of the test (minimizes the distance between subsequent BW values the test allows).

Bandwidth Limiter

Because we are now also dependent on there being data available to send, the minimum condition on A(t) may now apply but that would only set to limit the output bandwidth to be exclusively smaller than the input bandwidth at every point in time. In all other cases the minimum condition on A(t) still does not apply.

In this mode, it is not possible to know the proportion of garbage bits and therefore will need to assume all bits sent are useful for throughput calculations. The actual bandwidth may be smaller as a result. If one has knowledge of the makeup of the packets being sent one could calculate the proportion that is garbage and improve the equation. Likewise as we are just limiting the bandwidth of the output channel, we do not need to factor in a header added later on. Therefore we can use eqn2 in this scenario where the size of the packets no longer affect the bandwidth namely:

BW=$\frac{CR*DW*SF}{PE+SF}$ (eqn 2)

We can also calculate the Penalty given a desired bandwidth using:

PE=$\frac{CR*DW*SF}{BW} - SF$ (eqn 9)

Note when choosing SF in this mode the eqn8 constraint still applies.

An interesting result is that if we set PE = 0, regardless of SF, then BW = DW * CR which is the maximum bandwidth of the original AXI stream bus. In this configuration this core will never block a packet since A(t) never changes from 0.