AXI Stream Routing

***CONSISTENCY NOTE: due to new university policy some of the terms here have changed. As a result what used to be a Slave port is now called a Receiver port and the Master port is now called the Sender port. In online forums and most textbooks the old terms are still used.

In this mini tutorial I will try to show you how to do routing with AXI Streams so that you have multiple senders and receivers. In particular let us try to build a configuration where 3 Senders, call them S, try to send a packet to three receivers call them R.

Axi Stream Basics

As you know there are 2 mandatory signals in AXI Stream, Valid and Ready. A transaction occurs every time Valid = 1 and Ready = 1 on the positive edge of the clock (posedge). The sender is in control of the Valid line and the receiver of the ready line. On the senders perspective if they have data to send, they set Valid = 1, then after the posedge they check if Ready was also 1 to know if the message was sent or not. From the readers perspective if they can take data next cycle they set Ready to 1 and then after the posedge they look back to see if Valid was 1 and if it is they take and start processing it. At no point is there any confusion, both the sender and receiver always know if a message was sent or not, and they can send 1 message flit per clock cycle. They both also have the power to stop the messages from coming.

Longer Messages

Now suppose we want to send messages where the length is not known. We can make for example TDATA be 64 bits wide but maybe messages can be 64, 128, 256, or some weird length like 24 bits wide. Note AXI does require all messages to be multiples of 8 bits/1 byte long. The sender needs a way to tell the receiver how long each message is.

First to know the number of flits or pieces of messages we use an extra 1-bit signal called TLAST. This is very simple, TLAST=1 if this is the last flit, 0 otherwise. If the receiver sees TLAST=1 they know that the next flit corresponds with a different message. For instance with a 64bit TDATA, if we want to send 256b then we would send 64 bits at a time over 4 flits and on the fourth flit TLAST is 1.

In particular this is all needed if we are routing, since we don't want the router to mix up messages so for instance say all 3 senders are sending messages to the same target. Let us use MX.Y.Z to denote Message Number Y from sender X and this is flit Z. In this example the messages are of different sizes and they are all sending at the same time. You will learn of an AXI Stream switch that does the routing but this switch needs to organize the messages since the receiver can only get one message at a time and preserve that messages travel in order together. I have shown an example below to show one possible reordering that is valid. Flits in green have TLAST=0, red have TLAST=1 to mark end of packets

Sender 1	M1.0.0	M1.0.1	M1.0.2	M1.2.0	M1.2.1										
Sender 2	M2.0.0	M2.0.1	M2.1.0	M2.1.1	M2.1.2										
Sender 3	M3.0.0	M3.1.0	M3.1.1	M3.1.2	M3.1.3										
Receiver	M1.0.0	M1.0.1	M1.0.2	M2.0.0	M2.0.1	M3.0.0	M3.1.0	M3.1.1	M3.1.2	M3.1.3	M1.2.0	M1.2.1	M2.1.0	M2.1.1	M2.1.2

The other thing that you may or may not need is support for weird sized packets. AXI Stream requires packets to be multiples of 8 in size but doesn't require multiples of your TDATA size. For instance say you have a 64 bit bus but want to sent a 96 bit message. With what I said above you can send 64 or 128 bits but how does the sender tell the receiver it is only 96 bits. Clearly we have to send 128 bits but we need to be able to say that the last 32 bits are garbage to ignore. For this we have TKEEP.

With TKEEP we have 1 bit for each byte to mark if it is good data (1) or garbage data (0). In 99% of cases cores assume that you group all the garbage data together. So for example I want to send 96 bits. With the first flit I send:

TDATA = First 64 bits of data

TKEEP = 'b11111111

TLAST = 0

Clearly the TLAST = 0 means there is more data to send. The next flit contains the following:

TDATA<Upper 32 bits> = Last 32 bits of data

TDATA<Lower 32 bits> = ANY Garbage Number (0 works)

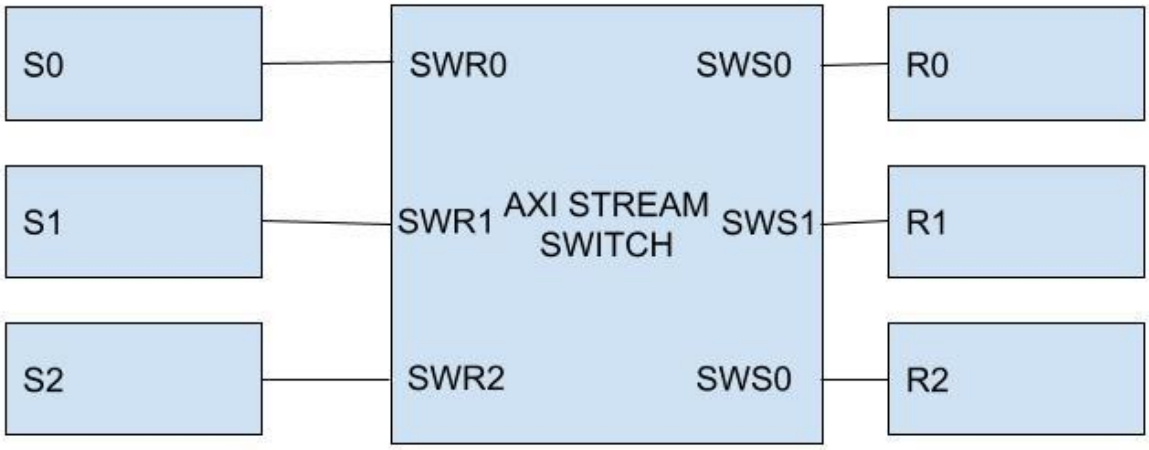
TKEEP = ‘b11110000 (First 4 bytes are valid, rest is garbage)

TLAST = 1 (end of message)

So now the receiver knows this is the end of the message and only the upper 32 bits of this second flit have data that I care about, the rest is garbage. He knows it is 32 bits because there are four 1s in the TKEEP, each corresponding to 1 byte worth of data.

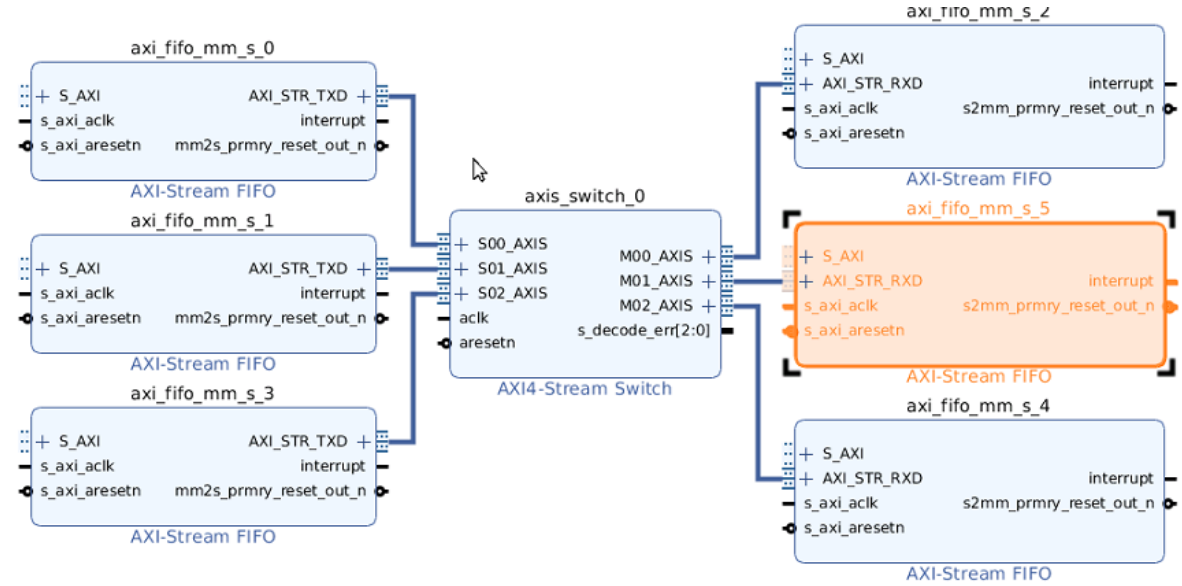
Routing

So far we have explored how AXI Stream works, how to send messages of any length that I want, now we will talk about how to have 1 channel with multiple senders and receivers. Lets go back to the example with 3 Senders and 3 Receivers. In this and any case we still just have point to point connections but now we have something called the AXI Stream Switch that handles the reordering of messages as they pass through so for example it would look like:

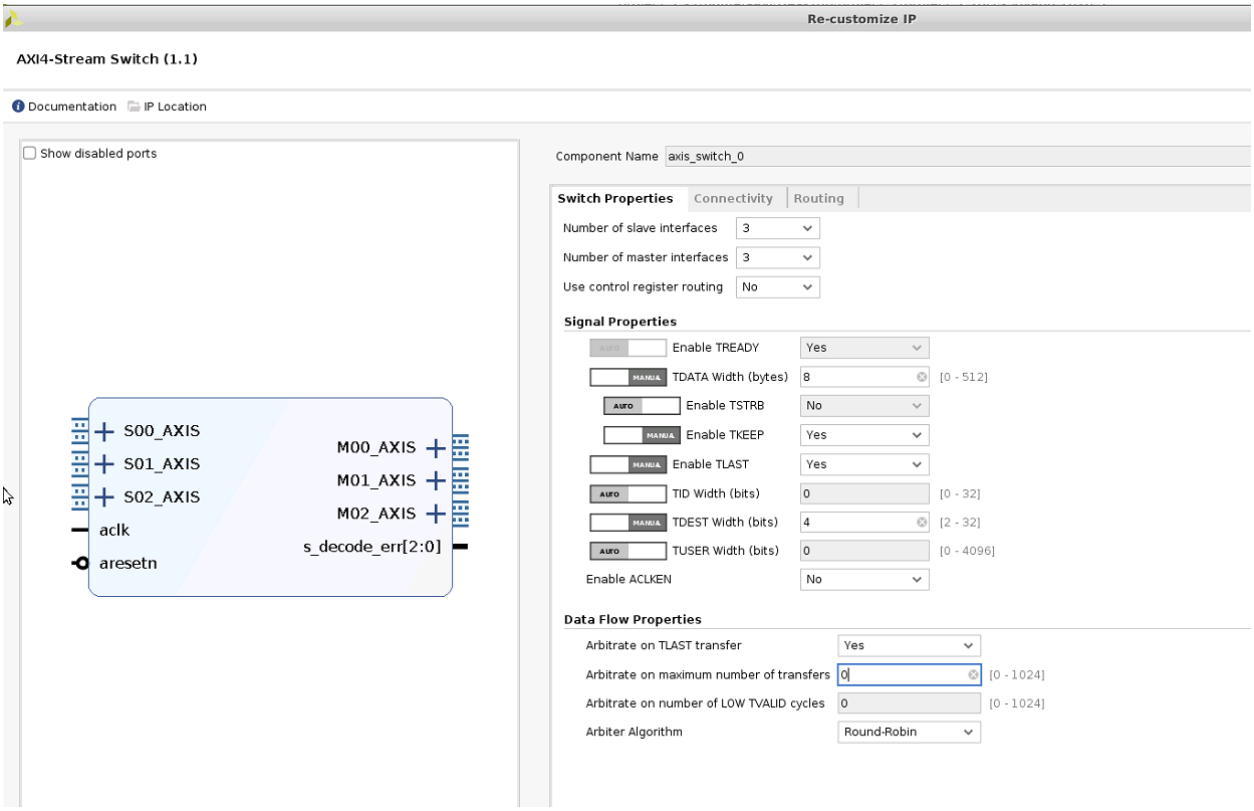


Notice here that we still just have point to point connections but now we have a connection either from Sender to Switch or from Switch to receiver. The switch does the routing and reordering. For the re-ordering and serializing so long as you follow the conventions with the Longer messages part you are good (e.g. use TLAST to indicate end of messages …).The only thing left is to tell the switch where to send each messages, for this we introduce a new signal called the TDEST signal. This signal can be of any width up to 32 bits. We need to program the switch to know for each of its sender connections (connected to the 3 receivers) which message is for which sender. We do this by configuring the switch to know which range of TDEST correspond with each of those ports. Then the senders can each set the TDEST of each message so it arrives where you want it to. For instance above we can program the switch so that TDEST=0 corresponds with SWS0, TDEST=1-5 corresponds with SWS1, and TDEST 6-13 corresponds with SWS2. They let you use ranges since you can also have trees of switches if the network is too big to fit on a switch, so you need to tell one switch the range that is on another switch.

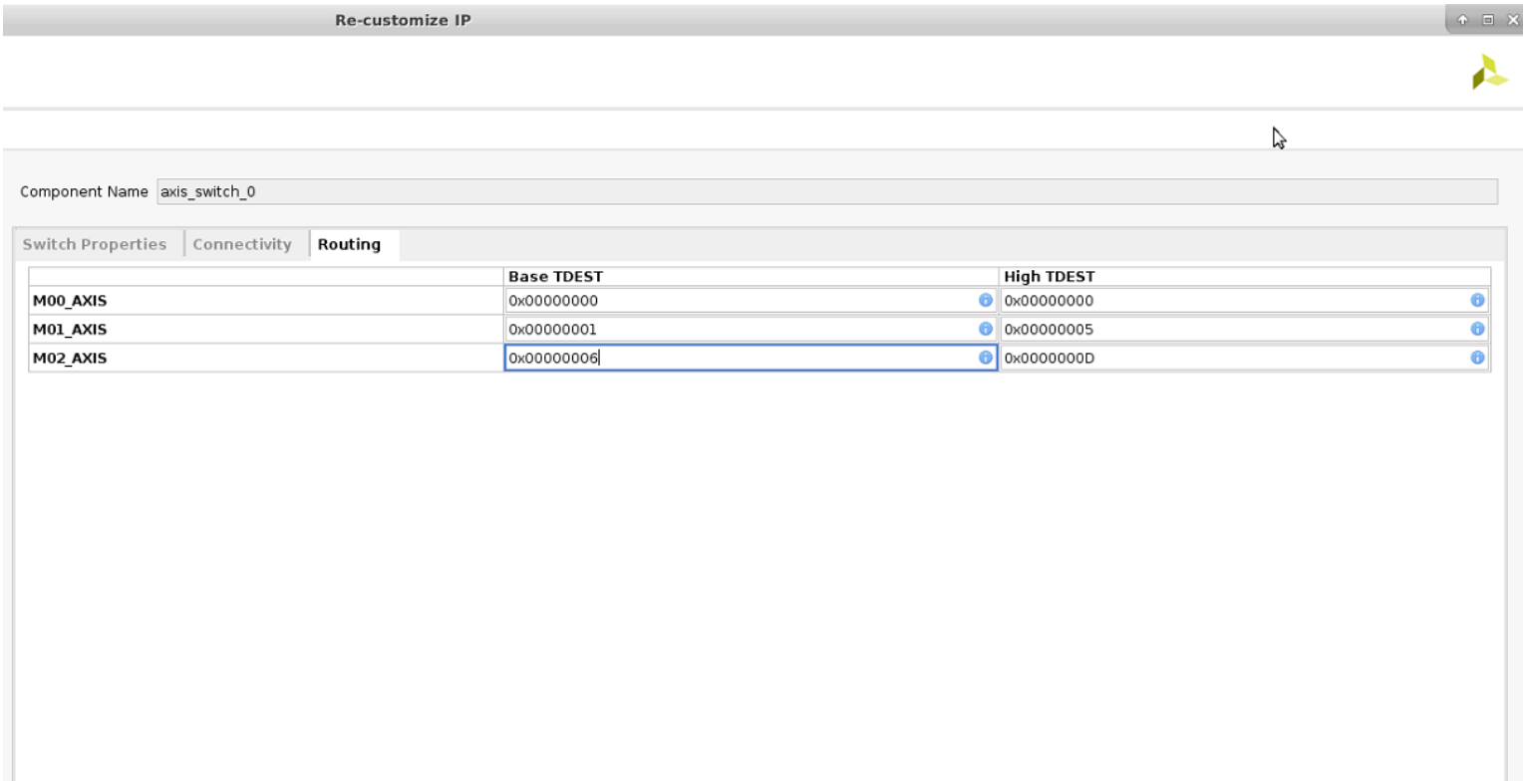
I built a simple example below with three arbitrary senders and 3 receivers. Note it uses S and M assuming the old names of Slave and Master so where it says S it means receiver and M means sender.



Inside the AXI4-Stream switch core I configure it as follows:



Here I configured it to have 3 slaves (receivers) and 3 masters (senders). The TDATA width of 8 corresponds with 64 bits. I have enabled KEEP and Last and DEST is 4 bits wide. In data flow properties I have told it to arbitrate on TLAST and on 0 maximum transfer. This exact configuration is needed to avoid mixing packets. Note these options are only available after enabling TLAST. As for the Arbiter algorithm, Round Robin tries to be more fair at the cost of a bit more logic use while fixed priority allows for a less fair but less logic if fairness doesn't matter.



In the Routing window I can set the low and high TDEST corresponding with each Sender port on the switch which goes to the receiver cores. I used the example configuration. Note the numbers here are in Hexadecimal so port 3 that was 6-13 is now 0x6 to 0xD.

In the connectivity window just check everything.

In HLS

With HLS we simply add the last keep and dest to the structure corresponding with the AXI Stream port. We would define something like

Struct dataword {

ap_uint<64> data;

ap_uint<8> keep;

ap_uint<1> last;

ap_uint<8> dest;

};

We then use it as usual. Just remember to write the correct dest value so it gets where you want it to get.

***NOTE: Galapagos also uses TDEST for routing so all of this also applies to Galapagos.