Broadcast and DMA idea

This document contains a proposal for a design of the Beta’s Broadcast and DMA engines. The main reason for proposing the changes listed here is to simplify the readout of the FIFOs to the SBC. We have the following challenges to contend with in reading out the FIFOs.

1) The Readout must be efficient
2) DMA must halt and restart for each address change
3) Ideally it should be possible to halt the DMA from the PCI side, if we want to initiate PIO
4) Ideally it should be possible to halt the DMA from the LBUS side, if we receive PIO

Since we will use the PLX to control DMA from the FIFOs, we should try to maximize the convenience this chip offers. However, if we do not know the size of each source, it is not possible to choose optimal DMA transfer sizes. Furthermore, we may have to implement additional logic to restart DMA, if a source is not fully transferred. If a source is exhausted before the DMA finishes, then we need logic to abort the DMA. Furthermore, we must count all words transferred as they exit the FIFOs (at high speed) so that the Mapper registers are properly updated.

The following approach may help to simplify our readout, by removing the need to monitor address information as the PLX controls DMA and by determining the exact size of each source as it is transmitted into the FIFOs. Thus allowing us to choose the DMA size to match each source size.

The scheme requires a reconfiguration of the Block Ram as shown on page 2. This scheme removes the Address FIFO in favor of an “address change” FIFO similar to one we discussed earlier. A third FIFO is added to keep track of how many MBUS words are transmitted from each source.

The following specifics of the D0 trigger are used here:

1) A single source uses only one address for the duration of its broadcast
2) A source holds Boss until it is complete and the releases Boss, forcing re-arbitration for the bus before another source may begin broadcasting.

Minimal assumptions are made about source sizes.

No changes are made to our draft Mapper plans.
# Block Ram requirements for DMA functions

Total Block Ram used = 137 of 140

## FIFOs

<table>
<thead>
<tr>
<th>Data (DA)</th>
<th>Address (AD)</th>
<th>Source Size (sSize)</th>
</tr>
</thead>
<tbody>
<tr>
<td>128b x 4k</td>
<td>9b x 1k</td>
<td>12b x 1k</td>
</tr>
</tbody>
</table>

- 128 Block Rams (1b x 4k x 128)
- 3 Block Rams (4b x 1k x 3)
- only lowest bit used in 3rd Block Ram here

## Translation Buffer (Mapper)

- 24b x 512
- 3 Block Rams (8b x 512 x 3)

Note: This scheme assumes the following
- Average source size $\geq$ 4 MB Words
- Max source size $\leq$ 64 KB
Broadcast machine (with new AD and sSize FIFOs)

A 12-bit counter (wCount) is set to 0 at power up. The Broadcast engine is Idle (0) until an address in the Broadcast range is decoded. When a valid DMA address is decoded we transition to state (1), 128 bits of MBUS data is pushed onto the DA FIFO, 9 bits of address are pushed onto the AD FIFO and we transition to state (2). In state (2) we push MBUS data onto the DA FIFO at each DSTROBE and increment wCount. When BOSS goes FALSE, the source has finished broadcasting and we transition to state (3). Here we push wCount onto the sSIZE FIFO and transition to IDLE (0) to await another broadcast to begin.

Notes:
• Minimum time expected between DSTROBEs from an MBT broadcast is ~ 80 ns w/ reasonable firmware improvements (Comment from Drew). The current time between DSTROBEs is ~ 100-125ns.
The DMA state machine relies on the excellent performance of the DMA readout that we have already observed. The FIFO readout via DMA controlled by the PLX works very well with our 1st draft firmware and it is quite easy to setup. If we can tell the PLX the exact details of each DMA burst, we need to do practically nothing in the firmware to manage the DMA of our sources.

This scheme would work as follows: The DMA engine is Idle(0) when there is less than one complete source in the DA fifo. When the EF goes FALSE in the sSize FIFO, we move to state (1) and may setup DMA. The head of the sSIZE FIFO gives us the number of MBUS words to transmit (this is stored in a 12-bit counter, nXfer), the head of the AD FIFO gives us the MBUS address for this source. Using the MAPPER, we determine the PCI base address for the transfer. Next we setup DMA registers in the PLX via local bus accesses:
1) DMAPADR0 = Mapper(Address)+DMAbase // target address on SBC
2) DMASIZ0 = nXfer << 16 // number of bytes to transmit
3) DMALADR0 = 0x2000 // local address of FIFO for beginning DMA
4) Finally the Mapper channel can be incremented by the number of 32-bit words in this transaction.

Then we move to state (2) and start the DMA by writing an appropriate word to DMACSR0. The Go command we send is 0x3. After this command is sent the PLX will take control of both busses and transmit nXfer MBUS words. After every 4th 32-bit word is accessed, nXfer is decremented by one. We move back to Idle when nXfer is reduced to zero.
Notes:

In this scheme we never have to actively halt DMA because an address changes during a transfer, or because the data FIFO goes empty. All DMA transactions are essentially on auto pilot by the PLX.

Why do we need to keep track of the DA FIFO readout with nXfer? Because we need some way to tell if the DMA has finished before setting up the DMA for the next source. The PLX may temporarily halt the DMA before it is finished with the source for a number of reasons. And we don’t want to start the next DMA until we have finished with the previous one. If we count the words transmitted from the FIFO, then we do not need to read the DMACSR0 register in the PLX to determine if the DMA has completed.

In this scheme we will always set the DMA size to the optimal value. The PCI latency register can be set in the PLX so latencies in PIO from the SBC can be minimized. In this case if the SBC wants to initiate a PIO transaction, the DMA would halt after a defined latency time and the PLX would resume DMA after the PIO is completed. (We may have to study PIO further, because if the PIO takes a long time we must be careful that PCI retries are handled properly… This is independent of the example above.) If a PIO request comes into the Xilinx from the MBUS while the PLX is controlling DMA we may have to rely on the LBUS latency timer to expire before the PIO transaction can complete. (There may be another option here, but the PLX documentation is confusing)