**Brief: Remarks. Bugs fixing.**
**Data: 22/07/2013 (last update)**
**Author: Vladimir Rybalkin**
**Supervisor: Christian de Schryver**

**Malfunctions observed by Thomas:**
The queues are connected via AXI4 stream protocol, meaning that I should only be able to read a value, if a valid one is available. However, I can continuously read zeroes from the queue (meaning the valid flag of the out-going port is always or at least initially set). There also has been some issue when writing values (which I currently can't fully recall. However, it is probably related to this zero issue anyway and I can't run any other meaningful tests with the queue as it currently is).

**Experiment 1:**
in_TValid='0' – there is no valid data from master. out_TReady='1' – slave is ready to receive data from queue. in_TReady='1' – slave (queue) is empty and ready to receive data.
**Observation:**
rd_ptr starts changing values, even though the queue is empty and in_TValid='0'. num_data – wraps around.
**Reason:**
In order to start reading from the queue out_TReady = '1' condition is considered to be sufficient. Then we increment rd_ptr and decrement num_data. We do not check if there is something in the queue.
**Solution:**
We have to start reading from the queue only when out_TReady = '1' and there is something in the queue; at least one of the elements is not empty. We have to check both conditions.

**Experiment 2:**
In_TValid='1' – master has valid data for the slave (queue). out_TReady='0' – slave is not ready to receive data from the queue.
**Observation:**
num_data (the number of items in the queue) becomes DEPTH-1 as max; means that the last memory cell is always empty.
**Reason:**
Writing condition is wrong. Writing happens when num_data is less than DEPTH-1.
**Solution:**
We have to compare num_data with the DEPTH, as num_data is not a pointer but number of items.

**Experiment 3:**
The same as before Experiment 2;
**Observation:**
When the queue has the last empty place the in_TReady signal is already '0'.
**Reason:**
Condition of the signal deassertion is wrong.
**Solution:**
We have to compare num_data with the DEPTH, as num_data is not a pointer but number of items.

**Experiment 4:**
out_TReady ='1' – slave is ready to receive data from the queue. in_TValid='1' - there is valid data from master.
**Observation/Reason:**
When there is a valid data from the master and the queue is empty, the data from master forwarded to the slave without being written to the queue. On the coming clock cycle the same data is written to the queue and then this data goes from the queue to the output. As the result, any data written to the empty queue appears twice at the output of the queue. First time because the data is forwarded; the second time as it goes from the queue.

**Solution:**
We have to set out_TValid only when there is at least one element in the queue. As a result the slave won’t read forwarded data; it will read the data only from the queue.

**Experiment 5/Observations:**
out_TValid initially is always zero. It can be set only if there is data in the queue, which requires in_TValid signal to be set before; or when queue is empty but in_TValid is set. So, out_TValid is strongly depends on in_TValid. Maybe, there is problem with in_TValid signal but from the MicroBlaze processor side. So, without setting in_TValid from the master side of the Microblaze processor, slave side of the MicroBlaze does not have to observe out_TValid from the queue. From my point of view, there is problem from MB side. The behavior of the out_TValid signal from the side of the queue is adequate.

**Experiment 6/Observation:**
In the current design out_TLast depends on num_data. So, every time when num_data is equal to 1, out_TLast becomes 1 indicating the boundary of a packet. From my point of view, this behavior of the out_TLast signal corrupts the idea of this signal in AXI4 – Stream protocol. According to AMBA® 4 AXI4-Stream Protocol: Specification, TLAST signal indicates the boundary of a packet; So, only master knows where the boundary of the packet is. From my point of view, out_Tlast has to mimic in_Tlast and be synchronous to data.

**Experiment 7:**
**Problem:** the queue overwrites values. It is possible to write more than DEPTH values to the queue without taking values out. For example, if the DEPTH is 128. It is possible to write 130 values (0 to 129) to the queue without reading. Afterwards, it is possible to read only last two values (those that were above the actual queue capacity) 128 and 129. Writing more values results in also reading more values (e.g. 130, 131, 132, …). This malfunction is observed by Thomas.
**Reason/Solution:** the problem is related to incorrect declaration of the num_data signal. The num_data signal is a capacity of the queue which can vary from 0 to DEPTH. In the current implementation, the range is from 0 to DEPTH-1, which leads to overflow. The range is changed to DEPTH. The problem is solved.