

**Dr. James P. Davis**

**From:** Dr. James P. Davis [jimdavis@cse.sc.edu]  
**Sent:** Tuesday, May 06, 2003 8:57 PM  
**To:** 'BADAGOWNI, VAHINI'  
**Subject:** RE: questions-612 Project

Hi Vahini,

Let me answer:

(1) Have you downloaded the updated top-level block diagram put together by G. Bhadri? See the course web page for Lectures, Lecture #41. He drew this up for me.

(2) TXD and RXD aren't really related signals, other than the fact that they are connected to the serial communications line (on different pins). TXD is the line on which the 8-bit word, with parity, start and stop bits are shifted out one bit at a time onto the serial line by the Transmitter. RXD is the one-bit serial line on which the Receiver block gets the bits, one at a time, shifting them in, according to the counter block in the Receiver.

(3) the Transmitter doesn't add these other bits until after it reads one 8-bit word out of the Transmit FIFO. Once the word is read, the parity bit is set appropriately by the coordinated logic of the transmit counter and the parity counter. There are several ways this bit of logic could work. One is to calculate the parity, overwrite the bit while still in an 8-bit register, then start shifting by first writing the start bit, then the 8-bits with parity, LSB first, then the stop bit. Or, you can start shifting, calculate the parity in real-time, adding it at the appropriate bit in the sequence (the MSB of the 8-bit word and thus the last one shifted out onto the serial medium. Some teams are doing it both ways.

Regarding the FIFOs, remember that a FIFO is basically a Queue; it has a "head" and a "tail". The consumer block read the head of the queue for the "first" word, so that it can be processed. The producer writes onto the tail of the FIFO queue, as a location becomes available. Each time a word is read, the words get shifted in the queue towards the FIFO head. This operation can be done a couple of ways:

(1) do it using a set of registers coupled in serial, where on each clock edge that a word is read off the queue (gating the clock with a control signal indicating that the word at the head of the queue has been read off the queue), and each word between the head and the tail is shifted through the register chain towards the FIFO's head. As locations in the queue get freed up, the FIFO queue's "full" flag gets set false, so that a data producer can write onto the end of the queue (or onto whatever location represents the last element in the queue).

In a serial configuration (which is also a structural style of model), writing to the last register in the FIFO stage will take some number of clock cycles to pipe the most recently written word to the head of the queue, but it doesn't matter what words precede it--although I think having an extra bit in the queue word to indicate whether the FIFO register contains a valid word or not is a good idea).

(2) do it using a set of parallel registers for FIFO words, keeping track of which register is the head, which is the tail, and which sequence the registers are in. As I indicated in class one day, this could be controlled using DEMUX on the writer side, and MUX on the reader side. The select lines would be controlled, basically, as pointers into the queue. This is probably easier to envision using a behavioral VHDL coding style rather than a structural one, as you could use arrays and indexing to manage the FIFO resources.

Note that this description applied to both queues (Transmitter and Receiver), just that the FIFO queue depth is different between the two. Under normal operating conditions, you would expect the speed of the processor and associated data, address and control buses to be much faster than the serial line speed--coupled with the interrupt mechanism gating the throughput of the CPU (i.e., preventing it from simply pushing the UART to overrun its

data queues). So, we'd expect the queues to not fill up. However, as a precaution, we have queues of some depth for the exceptional cases where the CPU could get hosed up responding to non-maskable interrupts (such as Bus Errors), and not be in a position to clear the Receive FIFO out in a timely manner, for instance.

Note also, that as you consider the behavior of the FIFOs, you have to consider what to do with the boundary conditions, namely (1) what do we do if we want to write to a FIFO that simply stays full (for Receiver is the most crucial case, because the other end of the serial line is not necessarily controlled with interrupt handshaking), and (2) what do we do if we want to read from a queue and the queue is always empty. So, what do we do under these conditions for both Transmitter and Receiver FIFOs?

At any rate, these are some thoughts and outcomes of various discussions I've had with different students over the past few weeks regarding your questions.

(I'll post this to the web page, as I think it will be of general interest to all the project teams.)

rgds,  
j. davis

-----Original Message-----

From: BADAGOWNI, VAHINI [mailto:BADAGOWN@engr.sc.edu]  
Sent: Tuesday, May 06, 2003 7:18 PM  
To: 'jimdavis@cse.sc.edu'  
Subject: questions-612 Project

Dear Dr.Davis:

Thank You for sending the Template for 798. I have some questions in 612 project.

1. stb\_fifo data signal is not given in the main block diagram. Also some other signals connections are missing.
2. It's not clear to me how the data is going from TXD to RXD.
3. Also in trasmit block, stb\_fifo\_data is an 8-bit signal from which data is received. But according to the description, the start, stop & parity bits should also come from the same signal. So I was not clear on that part.

I may have some other questions by tomorrow as I work on the project. So, do you want me to stop by your office or do you want to answer through e-mail?

Thank You for your attention.

-Vahini.