Asynchronous Design and You

Douglas Zorn

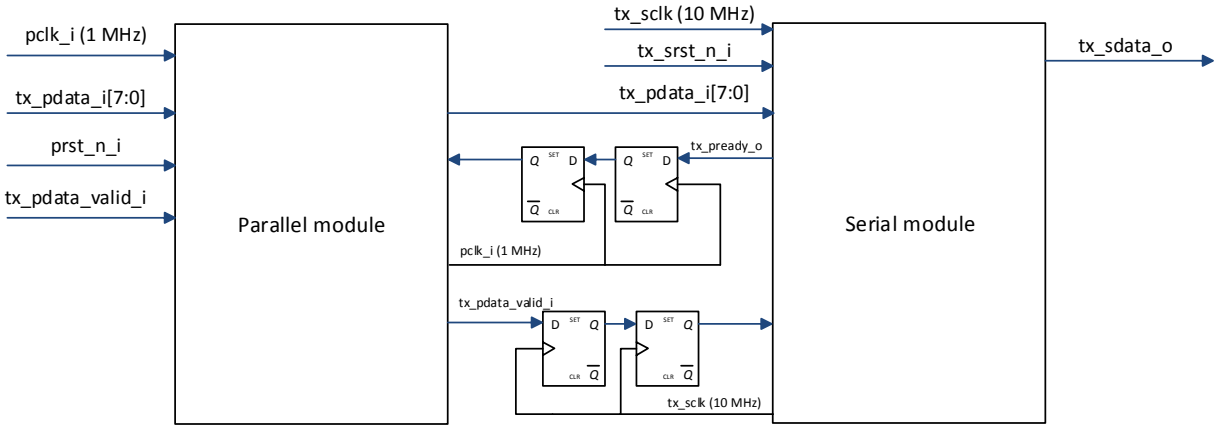
CE 125 winter 2014

UART

Designing a UART introduces many concepts, one of which is working with multiple clock domains and the problems that arise as a result. One of the difficulties is passing data safely from one clock domain to another. The issue is that the two clocks operate independent of each other and may or may not be in phase of with each other. This can create an issue when transferring to the other clock domain. There is the potential of violating hold times for flops which results in meta-stability. Clifford E. Cummings addresses these issues in a paper he wrote called, “Simulation and Synthesis Techniques for Asynchronous FIFO Design.” In his paper he discusses the use of synchronizers, which are cascading flops. The idea is that the first flop’s hold time may be violated but by the next clock cycle its output will have stabilized and the next flop will clock a stable value.

For my UART synchronizes are used to pass signals from different clock domains. This is first seen with the transmit module. The first part of the module takes 8-bit parallel data and converts it to serial data. The serial module has to send a ready signal to the parallel module. The parallel module also has to signal to the serial module that it has valid data. Both of these data signals are passed through synchronizers. Figure 1 below shows the block diagram for passing these signals.

Figure 1: Parallel to Serial Synchronizer

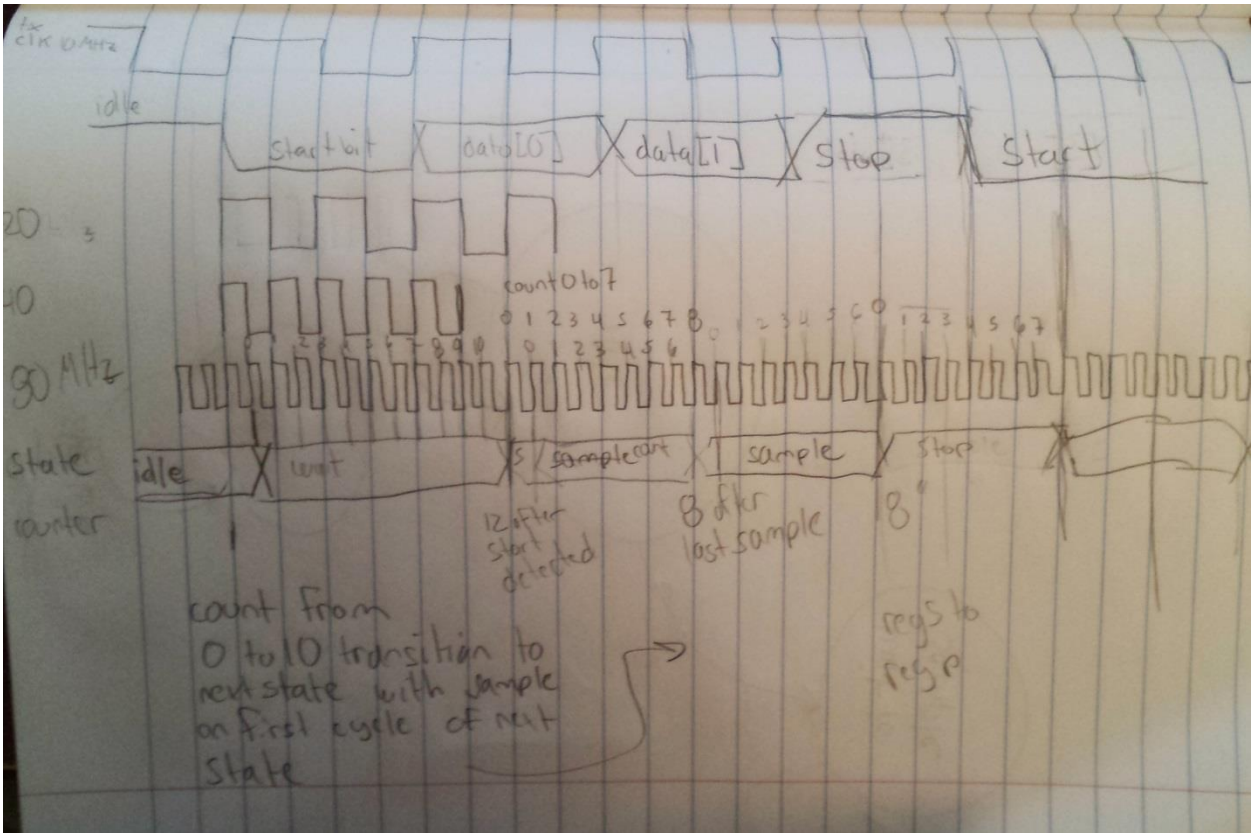


Once the Serial module is both ready and received the synchronized valid bit from the parallel module that data is safely sampled and transferred to the serial module where a 10 MHz clock transfers the data to tx_sdata_o.

The receiver

The receiver portion of the UART was driven by an 80 MHz clock. This was done so that tx_sdata_o could be sampled in the middle of the bit to ensure accurate sampling. Special care was made to design a state machine that could accomplish this sampling. Figure 2 shows a picture of the timing diagram from my lab notebook which the state machine was based off of.

Figure 2: Timing Diagram



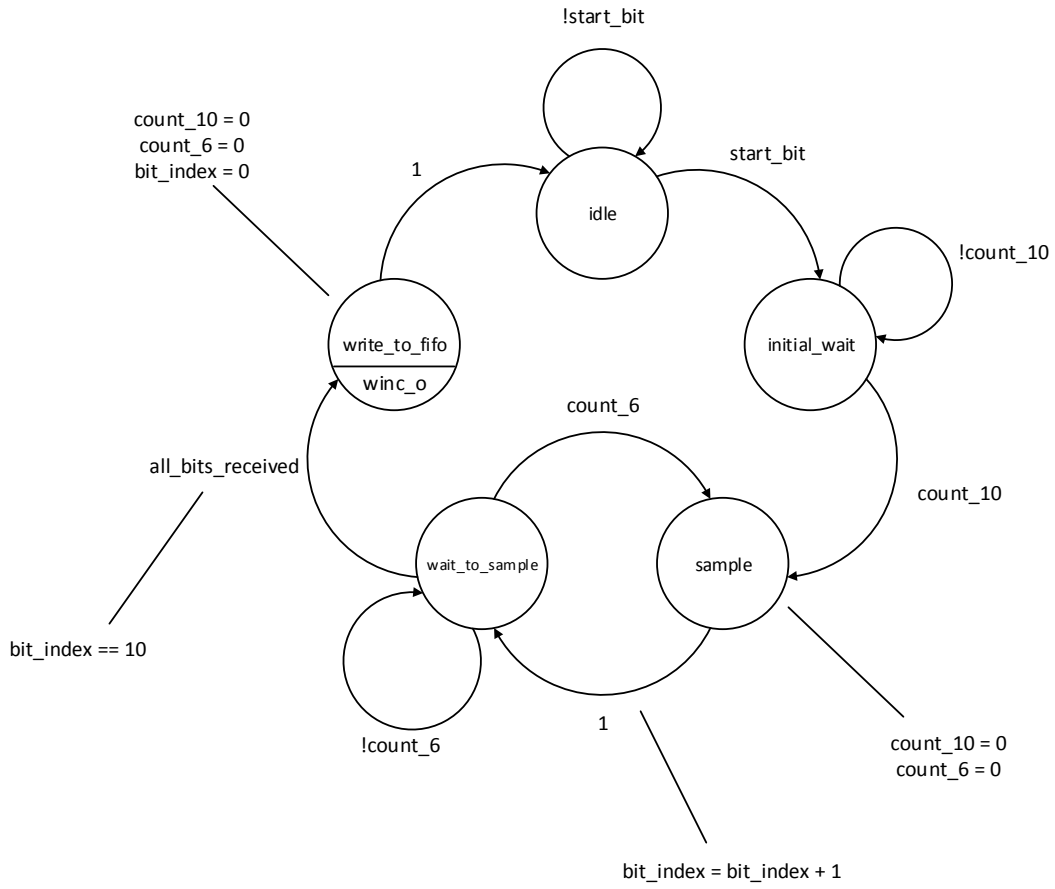
The top of the page shows the 1 MHz serial clock. Just below that is the serial data from the transmit module, tx_sdata_o. Three lines below that is the 80 MHz clock that the serial receiver operates at. Lastly, the fourth line shows an early version of the state transitions from the state machine.

The receiver state machine

The finite state machine was driven by an 80 MHz in order to sample tx_sdata_o as best as possible. With the clock running at 8 times the frequency of tx_sdata_o, this allowed the bit to be sampled in the middle. This was accomplished with timers. Except for the first data bit, all other bits were sampled when a timer reached a count of 4'd6.

There were numerous versions in my notebook of the state diagram. It took a while to get this one right. It wasn't until I realized that a FIFO had to be implemented at the receiver end in order to get the serial data back to parallel. Once I figured this out the diagram was pretty straight forward. Figure 3 on the next page illustrates all the states and transitions.

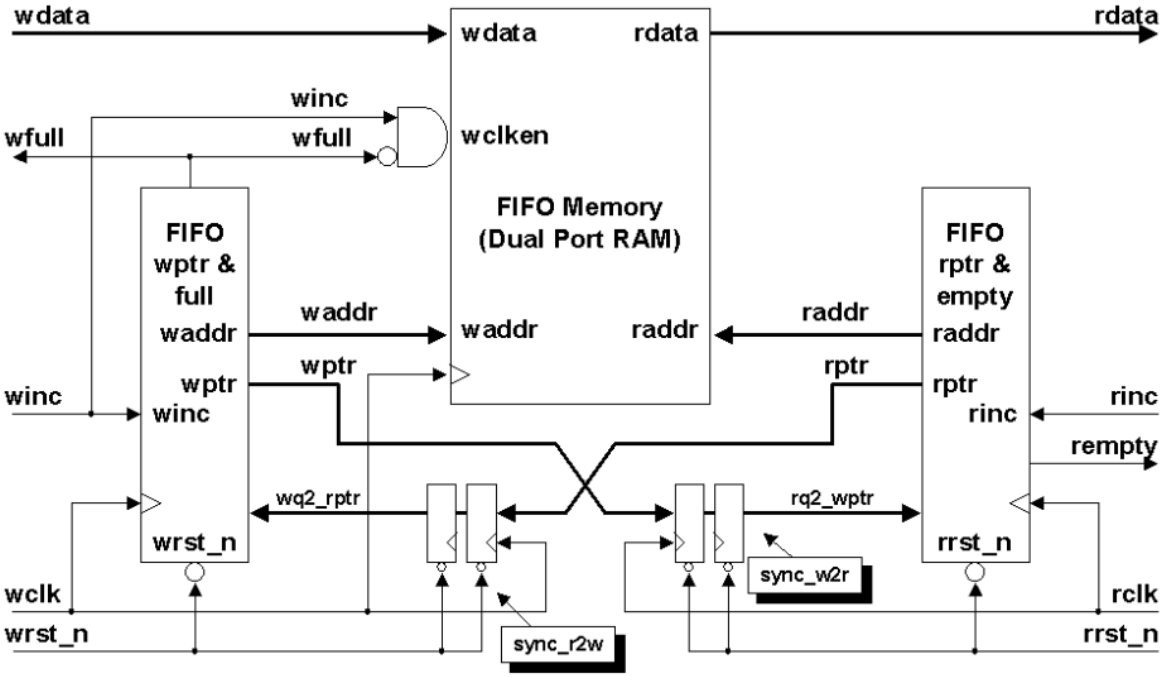
Figure 3: State Diagram



The state worth mentioning the most is write_to_fifo. It's at this state that winc_o is asserted for the FIFO. This is done in order to allow the state machine to return to its idle state as quickly as possible. It was important that it was done quickly as it would have been possible for a new UART packet to come in and it had to be in idle state to catch the condition.

It is at this point that I ran out of time to implement Cummings asynchronous FIFO. At the write_to_fifo state the serial data had already been transferred to a parallel register, but it still had to cross to the parallel clock domain which operated at 1 MHz. Cummings block diagram of the FIFO is reproduced below in figure 4.

Figure 4: Cummings FIFO block Diagram from "Simulation and Synthesis Techniques for Asynchronous FIFO Design"



My state machine asserted winc_o along with wdata_o [9:0] (8 data bits, parity, plus stop bit). In my case wclk would have been the 80 MHz clock and rclk would have been the 1 MHz clock.

Thoughts and Conclusion

Changing clock domains requires special considerations. When changing from a fast to slower clock domain asynchronous FIFOs with the use of synchronizers ensure that you can accurately recovery the data safely. Although I didn't get a chance to implement the FIFO, Cummings paper goes into great detail about the implementation that would have been used in my design.