John Rodriguez, Daniel Dunham, Brodie Gerloff

UART with Read-back Memory Project Report

4/17/20

**Summary**

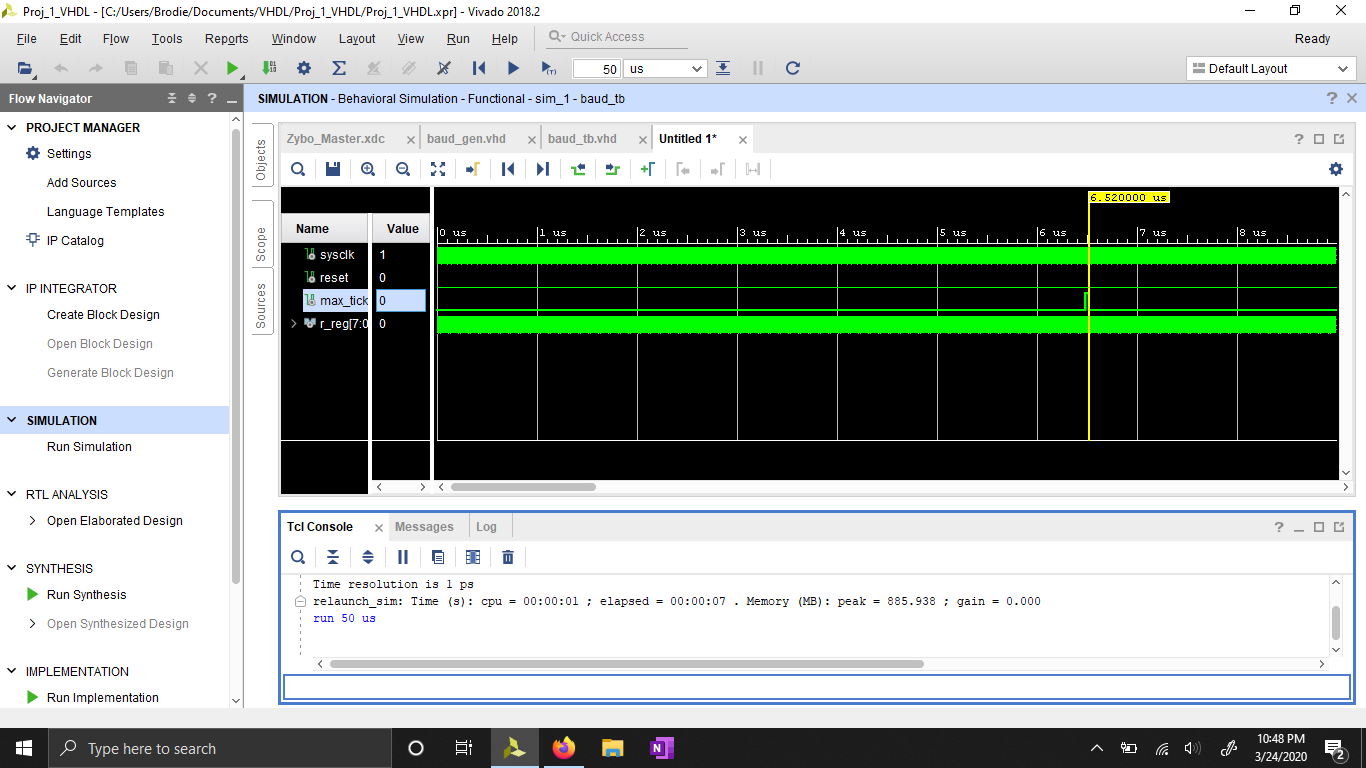
This report covers the creation of a VHDL based UART device. The device is capable of receiving data from an external source and saving it into an internal RAM before sending the data back to the original source. The UART device is designed for use with the Zybo Z7 board, but can easily be redesigned for use with any FPGA board.

**Module 1 : UART**

The UART is implemented in VHDL and contains three sub-modules: Baudrate Tick Generator, TX Module and RX Module. The baud generator creates a tick at 16 times the desired baud rate to properly synchronize the receiver and transmitter module. The code for the generator and TX/RX modules can be found in the following files:

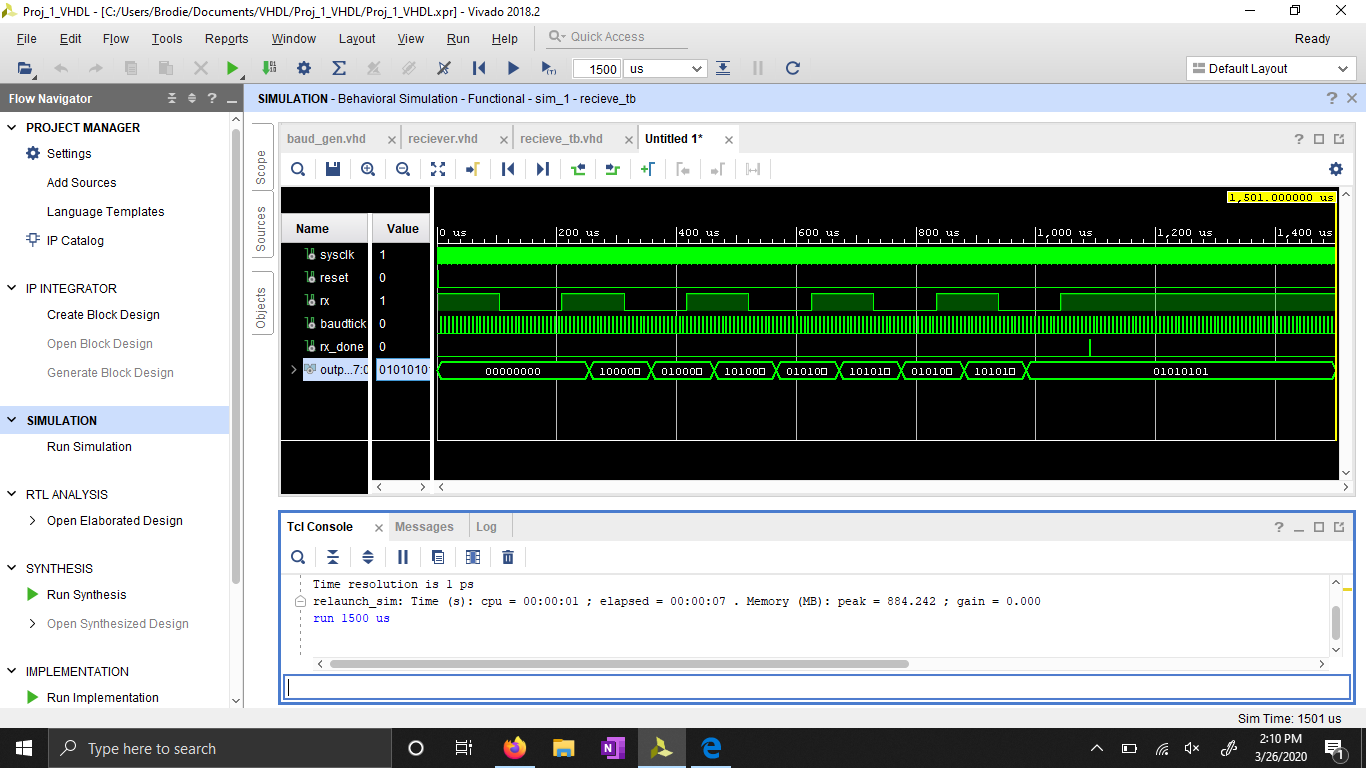
* Baud Generator: baud\_gen.vhd
* TX Module: transmitter.vhd
* RX Module: receiver.vhd

Testbench

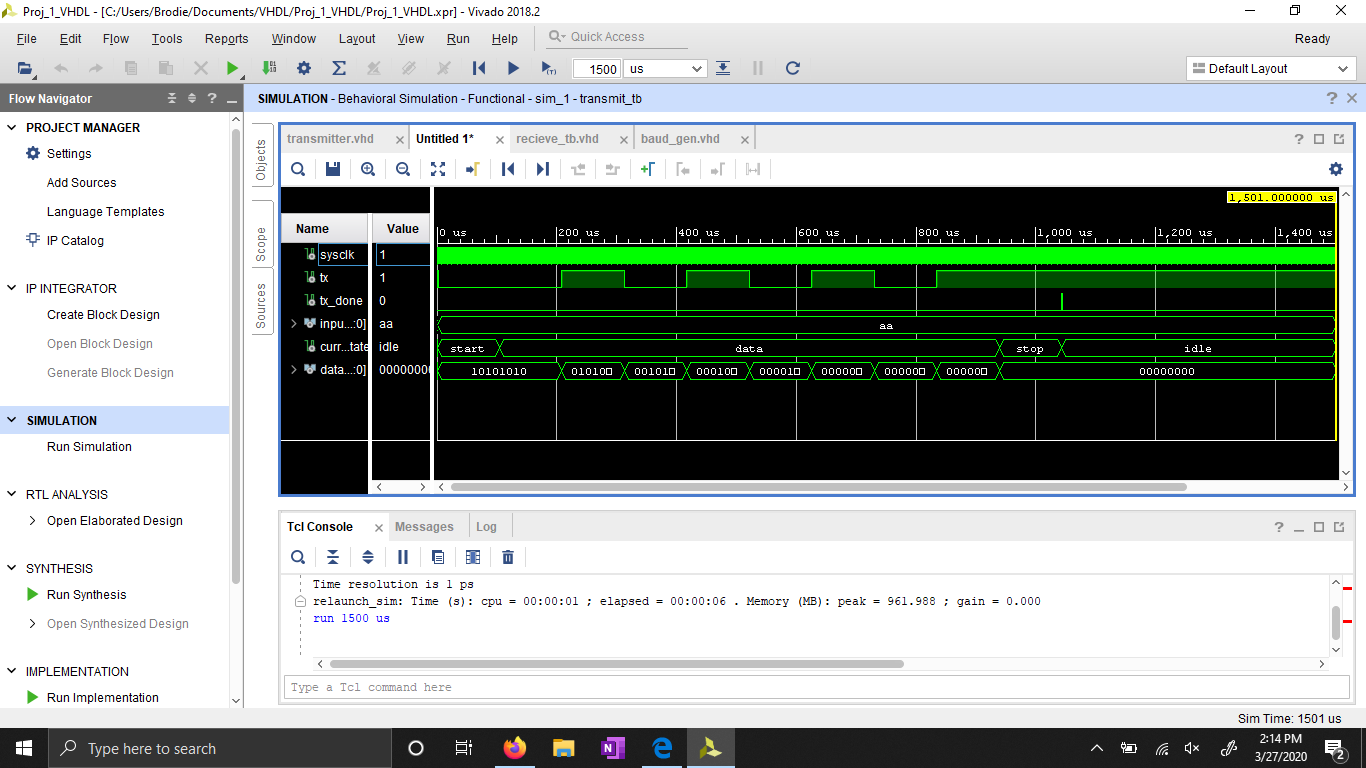


**Figure 1- Output from Baud Testbench**

The output shows that a tick occurs every 6.52 us, making a baud rate of 9585, an error of about 0.15%



**Figure 2- Output from Receiver Testbench (receiver\_tb.txt)**



**Figure 3- Output from Transmitter Testbench (transmit\_tb.txt)**

Figure 2 shows the receiver accepting data from the RX line, and shifting the incoming bits into a buffer. Since the data is received LSB first, the bits shift from left to right as they are received. Figure 3 shows the transmitter taking data from a buffer and outputting the LSB for the correct period of time before shifting the buffer to the right. The data is sent and received at the proper time for both modules.

**Module 2 : Synchronous FIFO, Register File Based**

The FIFO buffer is used on both the receiving and transmitting side to allow 8 bytes of data to be held for either module. This data is held until it is entered into the ram (for the receiving side) or entered into the transmitter (for the transmitter side). The FIFO buffer includes an asynchronous reset, but otherwise is a synchronous design which only allows reading and writing operations to occur on a rising clock edge.

FIFO Buffer is found in the file “FIFObuffer.vhd”

Testbench

The testbench for the FIFO generates an input stream to be written to the buffer. For ease of testing, the input stream is just an 8-bit unsigned integer which starts at zero and increments by 1 after every write operation. Figure 4 shows the output waveform of the FIFO testbench. Numbered red boxes have been placed around certain parts of the waveform in order to draw attention to important aspects of the FIFO buffer’s behavior.

Box 1 shows that after 8 write operations, the “full” flag goes high. Box 2 shows that with a full buffer, after a read operation occurs, the full flag goes back low, and once 8 read operations have been performed, the “empty” flag goes high. Boxes 3a and 3b show that starting with an empty buffer, when 4 write operations occur, followed by 4 read operations, the buffer is empty again and the “empty” flag is appropriately set high. Lastly, box 4 shows that with a full FIFO buffer, when a read operation occurs then the “full” flag goes low, and if it is followed by another write operation, the “full” flag goes back high. It also shows that If a write operation occurs whenever the buffer is not full, and a read operation occurs whenever the buffer is full, a cycle of read and write operations will occur one after the other, with the “full” flag toggling on and off appropriately. The behavior highlighted by the red boxes demonstrates that the FIFO buffer is working as expected. Also, the “d\_in” and “d\_out” signals in the waveform show that the correct data was written into, and read out of, the buffer in the correct order.



**Figure 4- Output from FIFO Buffer testbench (FIFObufferTB.txt)**

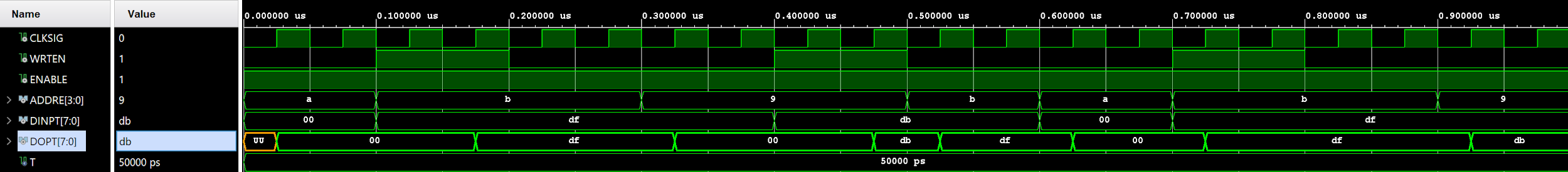
**Module 3 – UART with Readback BRAM Module**

The ram for the system was designed to function similar to the FIFO buffers, but to allow the data to be maintained if it is needed by the device. The RAM controller will keep track of the last byte entered and the last byte sent, but the data inside is not deleted until it has been sent. This allows it to function as a type of buffer for the data, but could be easily reworked into a memory storage for another internal device, such as an ALU or a microcontroller. The RAM will not send data until the transmit buffer is available, and will stop receiving data when it is full of unsent data. If not sending or receiving, waits in an idle state and outputs all zeroes (0x00).

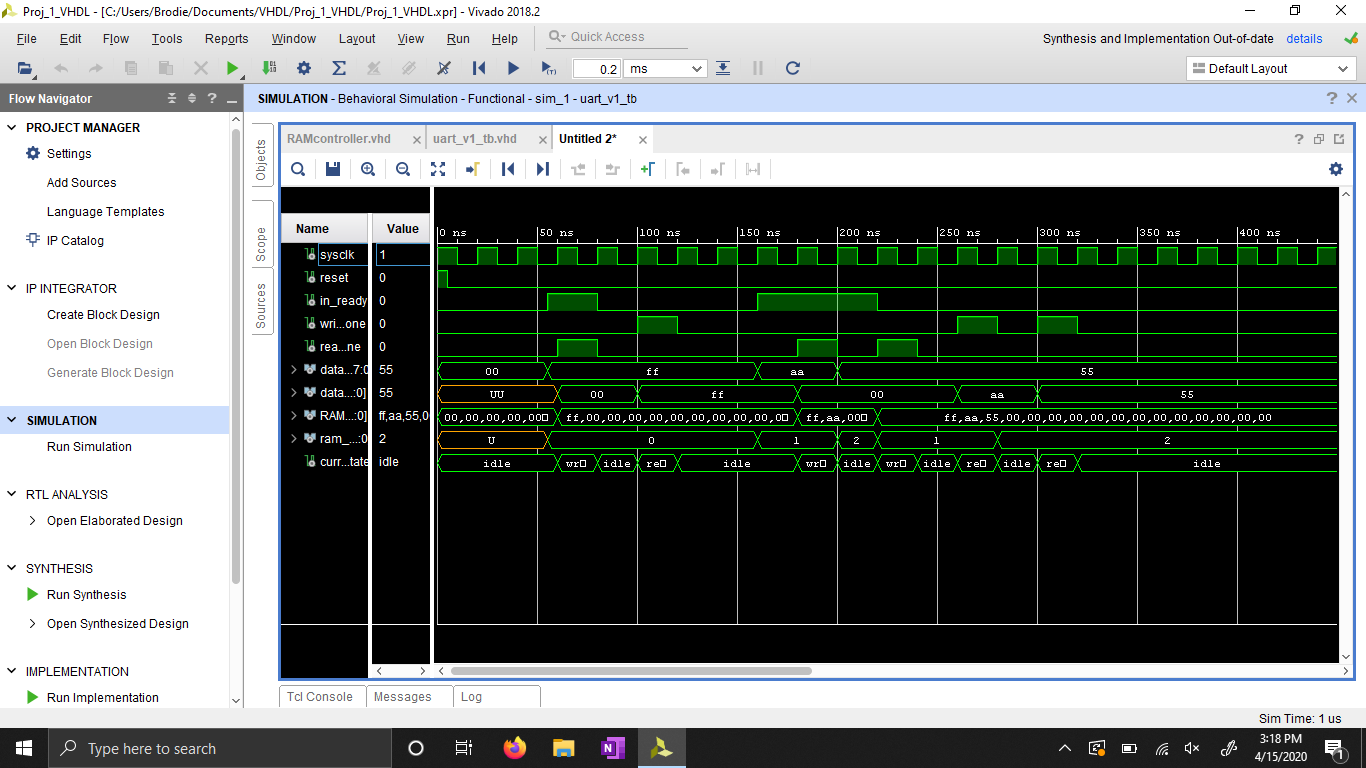
RAM File: “RAM\_BUILD.vhd”

RAM Controller File: “RAMcontroller.vhd”

Testbench



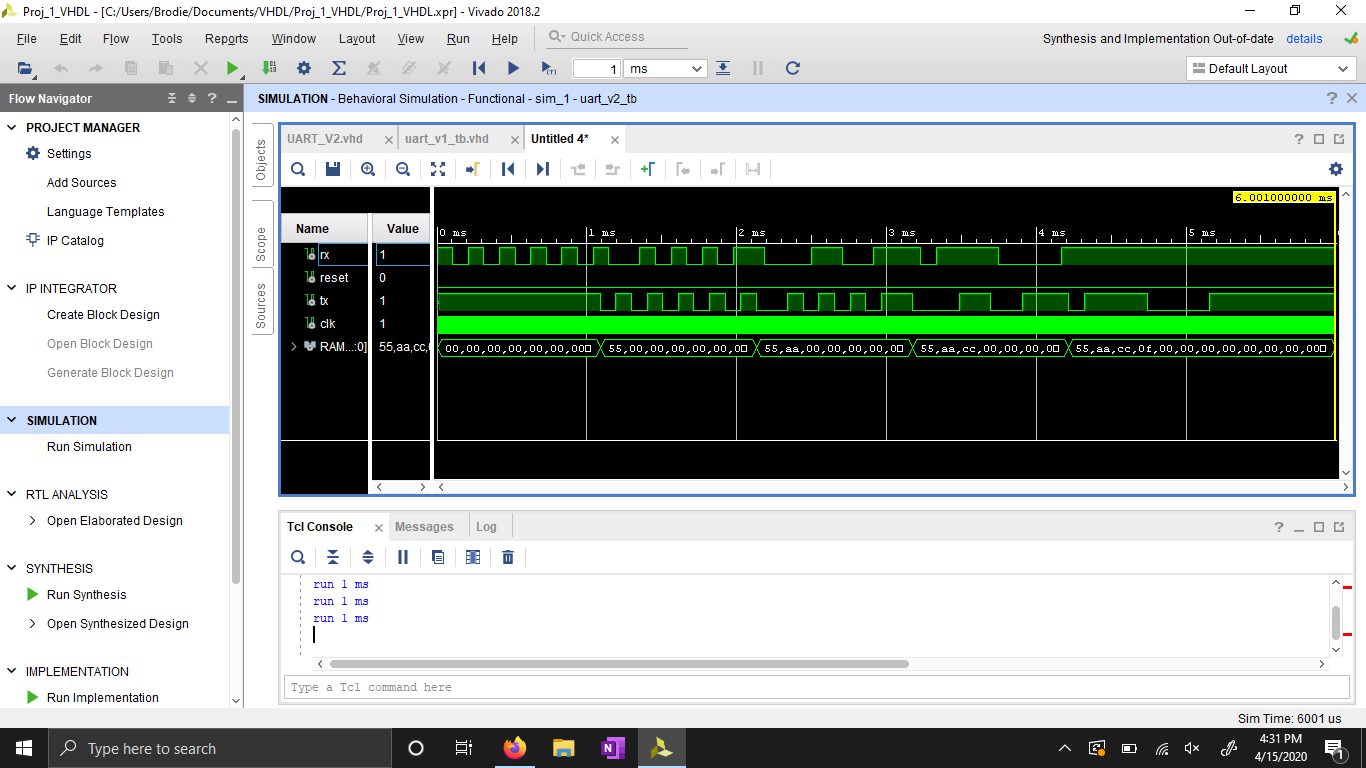
**Figure 5- Output from RAM Testbench**



**Figure 6- RAM Controller Testbench (ram\_control\_tb.txt)**

In the testbench for the ram controller, the functionality of the RAM is also shown. The testbench assumes it can always send data. When the “in\_ready” bit is high, the controller assigns the next empty (or already used) memory location to the current input. The device outputs a “write\_done” bit to notify the FIFO buffer that the memory was received. The RAM will continue to fill while it is not full.

Optional Demo



**Figure 7- Full System Testbench (UART\_V2\_TB.vhd)**

The testbench receives and transmits back four bytes, with the correct data being sent. The output sends as soon as the byte is available, which causes the byte to be transmitted back almost immediately. The bottom waveform shows the current data in ram, which is updated during the stop bit of the receiver.