College of Engineering
UNIVERSITY OF WISCONSIN–MADISON

Minilab 3 Report

Professor: George Tzimpragos
Course: ECE 554 - Digital Engineering Laboratory
Authored By: Shantanu Chaudhuri and Harrison Doll
Last Reviewed: February 19th, 2025

Authored By: Shantanu Chaudhuri and Harrison Doll
Last Reviewed: February 19th, 2025

Table of Contents

Assignment Outcomes & Challenges 3-4

Experiment Conducted & Testing 5-6

Github Link 7

Assignment Outcomes & Challenges

In this lab, we successfully implemented a SPART to interface between a processor and a serial I/O port, enabling communication with lab workstations. The process began with understanding the basic architecture of SPART and how it interacts with the processor and serial I/O port. This involved working through the design specifications, particularly how to manage asynchronous data reception and transmission between the FPGA and the workstation via RS-232 voltage levels. Once the baseline design was established, we focused on implementing the bus interface, where we had to ensure proper data transfer between the processor and SPART using the DATABUS, while handling I/O control signals like IOCS and IOR/W for reading and writing data.

A significant part of the implementation was the Baud Rate Generator (BRG), which needed to adjust for varying clock frequencies and maintain consistent baud rates. We had to design the BRG to take the clock frequency and desired baud rate, calculate the appropriate divisor, and use this value to generate an enable signal for both the transmitter and receiver. This required us to understand clock domain crossing and how to use enable signals in the absence of separate transmitter and receiver clocks. Another crucial step involved the hardware testbench, where we simulated the operation of the SPART to ensure that characters could be transmitted and received correctly. We verified the functionality by testing communication at multiple baud rates and ensuring that the data was accurately transferred from the workstation to the processor and back.

One of the biggest challenges we encountered was the management of asynchronous data reception. Since the received data on the RxD pin is asynchronous with respect to the system clock, we had to implement synchronization techniques to ensure reliable data capture. Initially, we faced issues with timing mismatches that caused data loss or incorrect reception. This required us to carefully design and test synchronization mechanisms using flip-flops to capture the RxD signal at the appropriate clock edge.

Another challenge was configuring the Baud Rate Generator to handle varying clock frequencies. The need to adjust the divisor for different clock settings presented difficulties, especially when calculating the exact divisor value to maintain accurate baud rates. After several iterations and testing, we were able to fine-tune the divisor calculations and verify that the generated baud rates met the specifications.

Synchronization between the processor, SPART, and the serial I/O port also proved to be challenging. Ensuring that the data transfer was properly aligned between the processor and SPART, especially during read and write operations, required precise timing control. Debugging these synchronization issues was time-consuming, but by carefully analyzing timing diagrams and adjusting our design, we were able to achieve stable communication.

Overall, this minilab provided valuable experience in working with serial communication protocols, asynchronous data handling, and FPGA-based design. Despite the challenges, we were able to successfully implement and test the SPART, gaining a deeper understanding of how to interface hardware components and manage timing and synchronization in a real-world communication system.

Experiment Conducted & Testing

To ensure the proper functionality of the system, we used the simulation log output to verify each stage of data transmission and reception between the two SPART modules, the driver, and the printf module. The test started by sending individual bits of data, beginning with the first bit, and then sequentially checking each bit to confirm correct transmission. The log output showed the sequence of bits sent: 0, 0, 0, 1, 0, 0, 1, 0, which corresponds to the ASCII value for the letter 'H'.

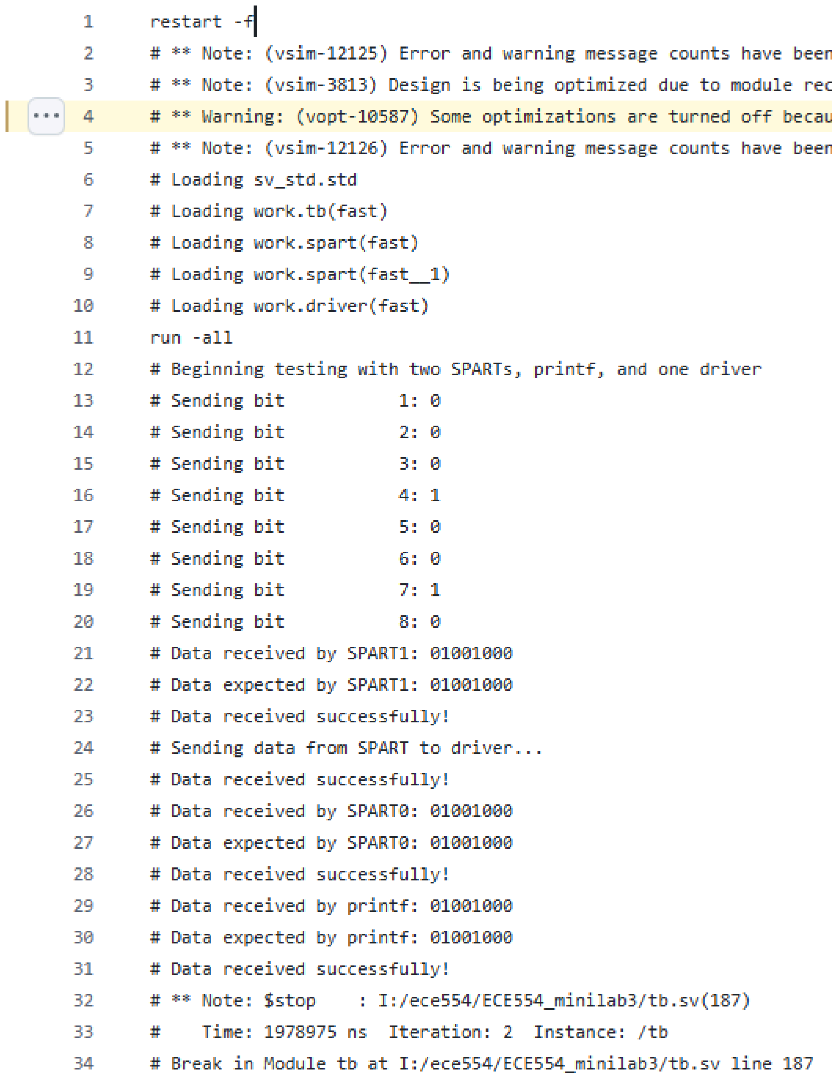
After each bit was sent, we verified the data received by SPART1. The log output indicated that SPART1 received the expected data 01001000, matching the data that was transmitted. This confirmed that the data from SPART0 was correctly received by SPART1, ensuring proper functionality of the serial communication link. The same check was conducted for SPART0, where the data received by SPART0 was also correctly matched to the expected 01001000, further validating the communication integrity.

Next, we checked the interaction between the SPART and the driver. The log confirmed that the data sent from SPART to the driver was received successfully. Similarly, the printf module correctly received the same data 01001000, matching the expected output. This confirmed that the data was accurately passed through each component in the chain, and was properly printed by the printf module.

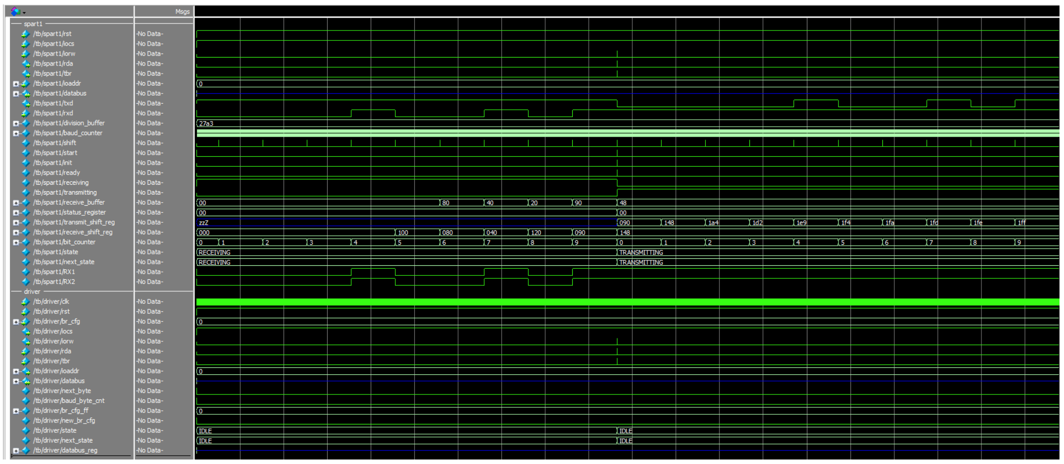
In the final stages of the simulation, after confirming the data integrity at each module, the test was stopped at the specified breakpoint in the testbench code at line 187, and the overall process was verified to be functioning correctly.

Through this testing procedure, the log output demonstrated that the system was successfully transmitting and receiving data without errors at each stage. All expected data was accurately received and processed by the respective modules, confirming that the implementation was working as intended.

Authored By: Shantanu Chaudhuri and Harrison Doll
Last Reviewed: February 19th, 2025



```
restart -f
# ** Note: (vsim-12125) Error and warning message counts have been
# ** Note: (vsim-3813) Design is being optimized due to module rec
# ** Warning: (vopt-10587) Some optimizations are turned off becau
# ** Note: (vsim-12126) Error and warning message counts have been
# Loading sv_std.std
# Loading work.tb(fast)
# Loading work.spart(fast)
# Loading work.spart(fast__1)
# Loading work.driver(fast)
run -all
# Beginning testing with two SPARTs, printf, and one driver
# Sending bit           1: 0
# Sending bit           2: 0
# Sending bit           3: 0
# Sending bit           4: 1
# Sending bit           5: 0
# Sending bit           6: 0
# Sending bit           7: 1
# Sending bit           8: 0
# Data received by SPART1: 01001000
# Data expected by SPART1: 01001000
# Data received successfully!
# Sending data from SPART to driver...
# Data received successfully!
# Data received by SPART0: 01001000
# Data expected by SPART0: 01001000
# Data received successfully!
# Data received by printf: 01001000
# Data expected by printf: 01001000
# Data received successfully!
# ** Note: $stop    : I:/ece554/ECE554_minilab3/tb.sv(187)
#    Time: 1978975 ns  Iteration: 2  Instance: /tb
# Break in Module tb at I:/ece554/ECE554_minilab3/tb.sv line 187
```



Authored By: Shantanu Chaudhuri and Harrison Doll
Last Reviewed: February 19th, 2025

Github Files and Process Explained

We created our repository in github and just to maintain ease and mitigate any risk of losing anything - https://github.com/fuzzy41316/ECE554_minilab3 - It should be public, but please let us know if we need to adjust any settings for visibility.