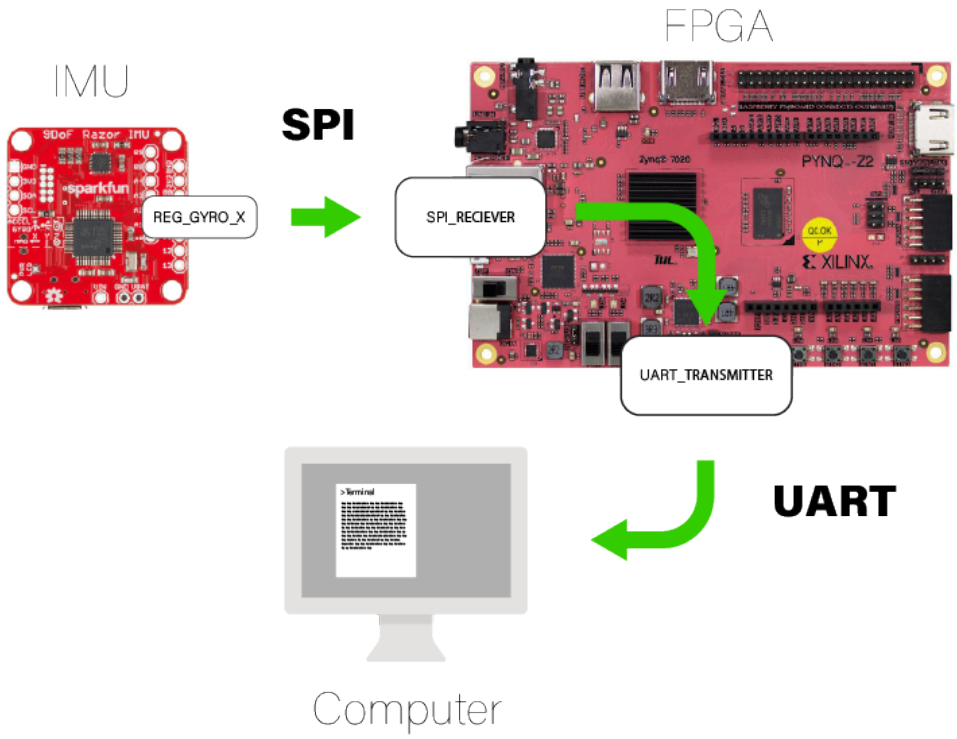
Assignment 1 -IMU interface through SPI and UART

Embedded Systems

Karina Bay Bloch (kablo17) & Nicoline L. Thomsen (nthom18)

October 2021



Contents

1 Introduction

The purpose of this assignment is to build a digital circuit to interact with a given IMU, with 9 degrees of freedom, and receive data from it to show on a PC terminal with UART. Starting with the IMU, information can be gathered from it by means of SPI or I^2C, where SPI is the chosen communication method because of the assignment aim. When the information has been acquired, it will be transmitted from the FPGA to the connected computer by the means of UART. this process should be continuous as long as it is enabled to do so. See figure 1 to see the plan of action for the layout of the project.

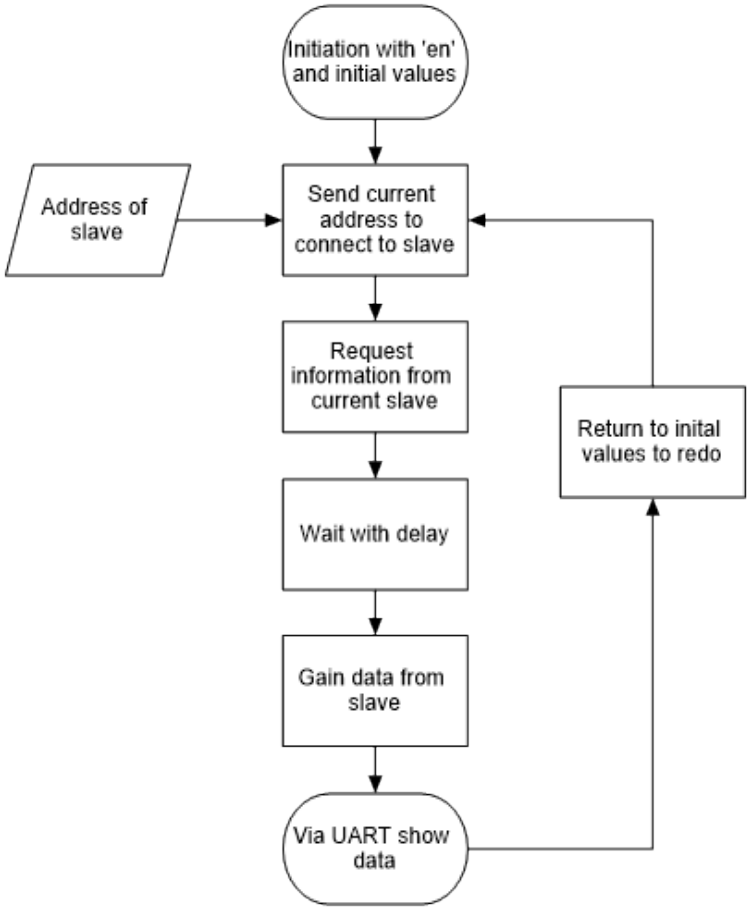


Figure 1: Flowchart of the process that is sought after in this assignment

Figure 2 shows a block diagram of the overall system, which consists of two main modules that handles SPI and UART communication. Both of which are explained in detail in the following chapters. The `clk_divider` module uses the system clock of $125MHz$ and divides it to match to computer baud rate of $115200bps$. This module is from the lecture exercise solutions. The inputs 'en' and 'rst' are button 3 and 0 on the FPGA board respectively.

(ノ °□°) ノ ⌒ ┻━┻

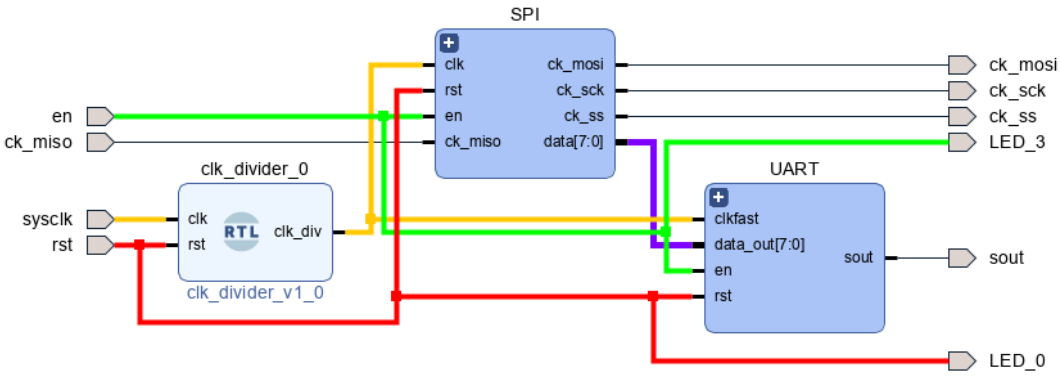


Figure 2: Block Diagram of system

Before testing the system on hardware, a behavioural simulation was run to make sure the system had the expected behaviour, see figure 3. This was a more time efficient method, as generating a bit-stream file for hardware employment takes a substantial amount of time. However the system was also frequently tested on the hardware to identify various issues with the programming, because it is better for observing the behaviour in real time.

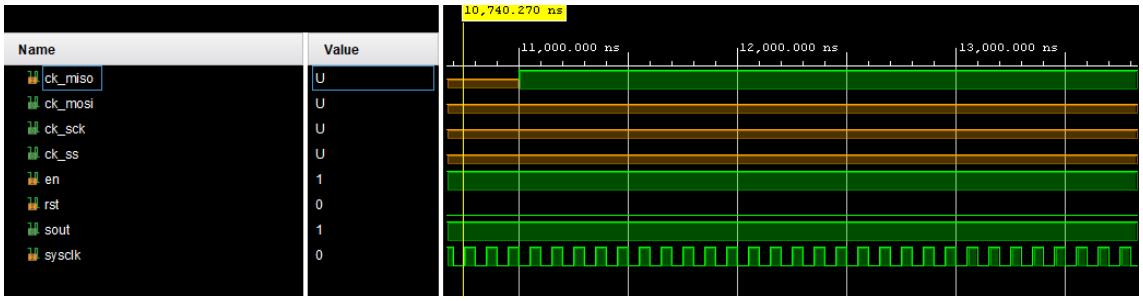


Figure 3: Screenshot of simulation setup

2 SPI

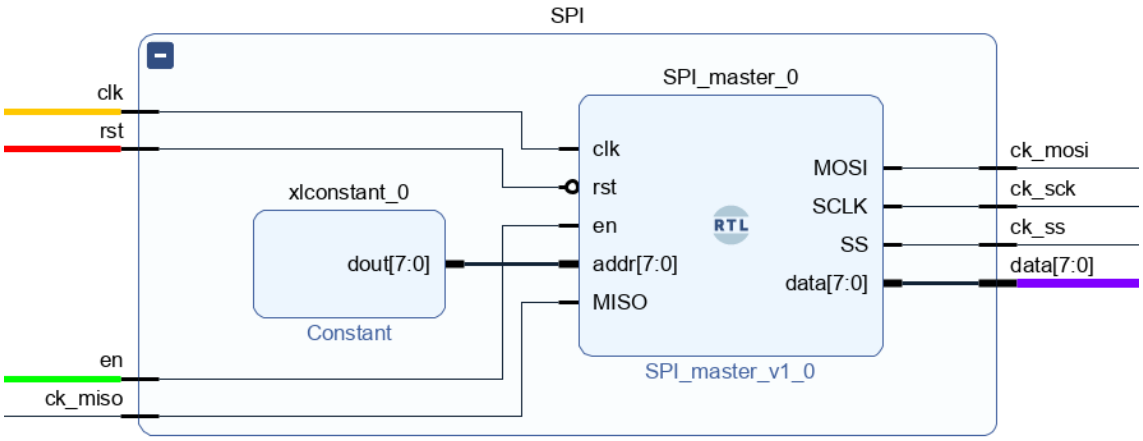


Figure 4: Block Diagram of SPI

(ノ °□°) ノ ⌒ ┻━┻

The protocol for communication between the FPGA and the IMU is SPI (Serial Peripheral Interface-bus). The module used to handle this protocol is `SPI_master`, which is shown on the block diagram on figure 4. The `xlconstant` is a constant value that provides the address of interest to access in the IMU.

On figure 5 is a state machine for the `SPI_master` module. As input there is 'clk' which has to be lower than $7MHz$, the maximum supported by the IMU. For this reason the `clk_divider` output is used. Other inputs are 'rst' to reset, 'en' to enable the communication, 'addr' which is the address to access in the IMU, and the last input is 'MISO' (Master Input Slave Output). The outputs are 'MOSI' (Master Output Slave Input), 'SCLK' to provide the slave with a clock signal, 'SS' which is Slave Select and 'data' which the 8-bit data output from the IMU. This output is fed into the UART system to forward the data to the computer serial terminal.

When enable is set it will change form state 0 to state 1, where 'SS' is set low (slave select is active low) and the address is read from the input pin. In state 2 the adress is transmitted to the IMU one bit at a time, MSB first. When all 8 bits have been sent there is a delay of 8 clock cycles, and from there it enters state 4 and begins receiving data. For 8 clock cycles it receives one bit at a time, and in state 5 the data is put on the 'data' output pin, and slave select is deactivated.

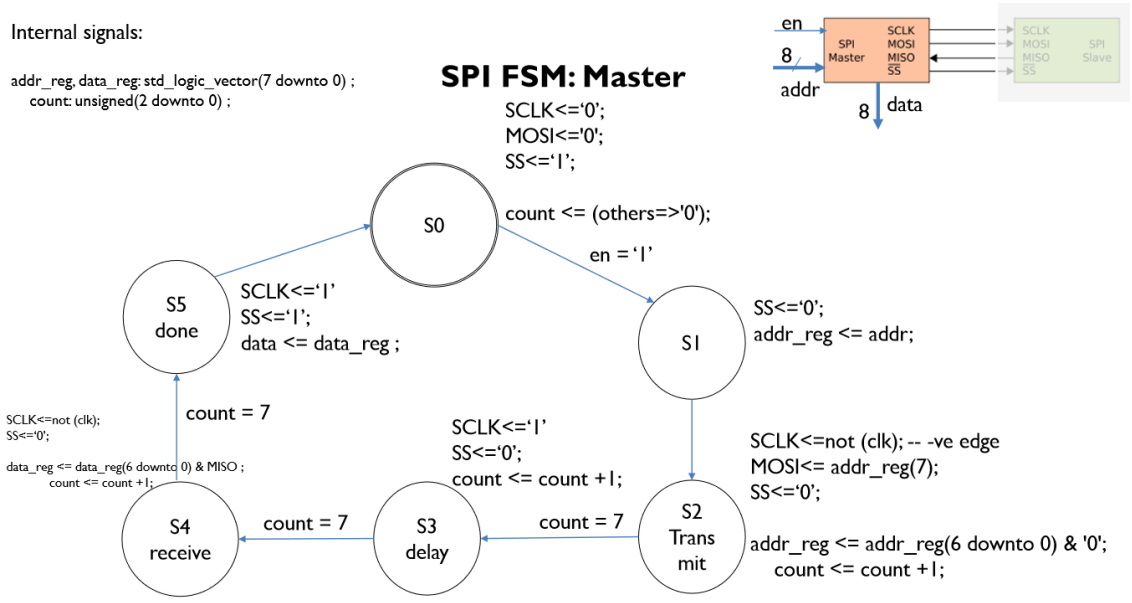


Figure 5: State machine for the `SPI_master` module. Image from slides Lecture 3.

(ノ °□°) ノ ⌒ ┻━┻

3 UART

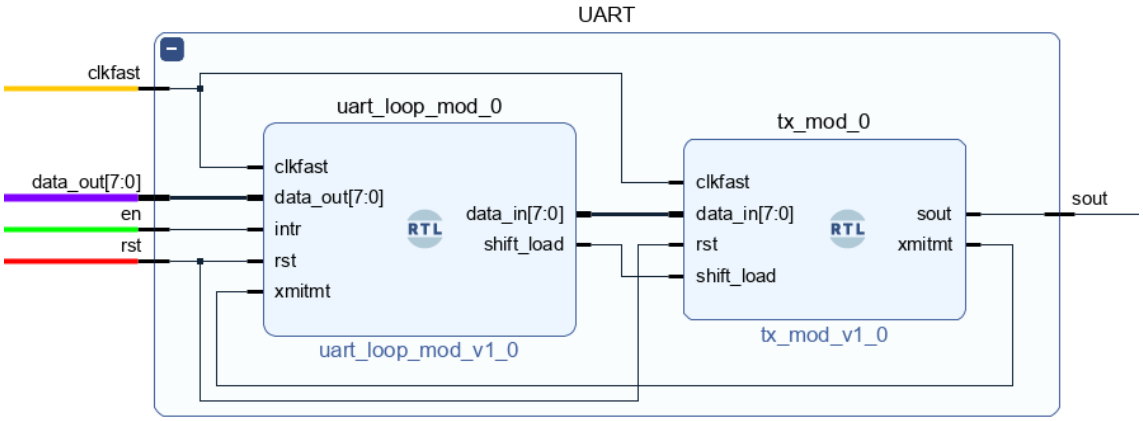


Figure 6: Block Diagram of UART

The protocol for transmitting the data gathered form the IMU to the computer is UART (Universal Asynchronous Receiver/Transmitter). The modules used for this has been repurposed from the lecture exercise solutions. These modules are besides `clk_divider`, the `uart_loop_mod` and `tx_mod`, shown in figure 6.

The `uart_loop_mod` module handles when to start and stop forwarding data to the writing module, `tx_mod`, depending on whether the `tx_mod` module is ready to send new data. Figure 7 shows the state machine of the `uart_loop_mod` module. It has input 'clkfast', which is a clock that matches the computer baud rate (115200 bps), 'en' to enable to module and 'rst' to reset. 'data_out' is the 8-bit logic vector to be sent to the computer. This value is directly written to the output 'data_in'. The other output, 'shift_load', will be set to 1 and only be disabled when the input 'xmitmt' is set, indicating that the write operation was complete.

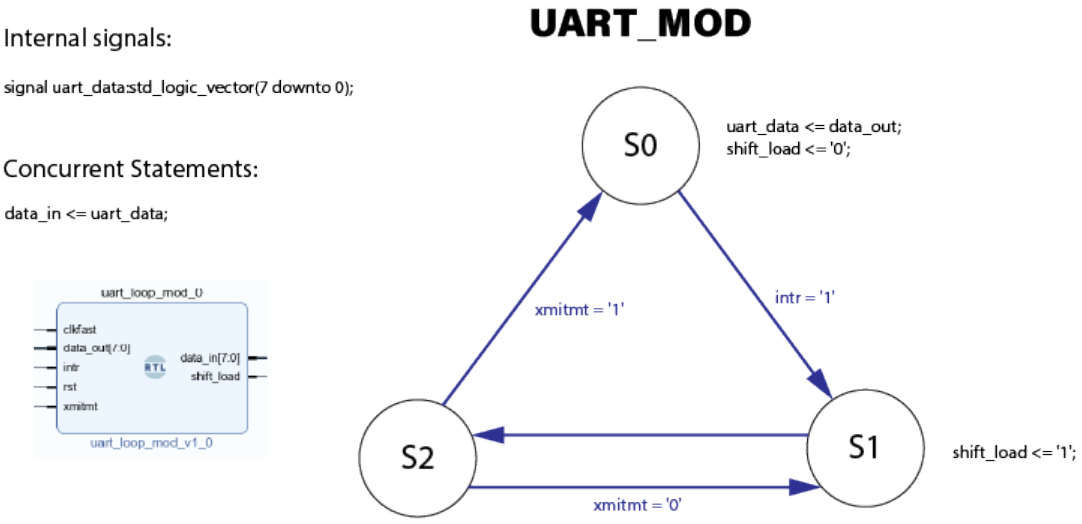


Figure 7: State machine for the `uart_loop_mod` module.

`tx_mod` is the module that handles the UART protocol. It has four inputs, 'clkfast' to match baud rate, 'data_in' for the data to be transmitted, 'rst' to reset the operation and 'shift_load'

(ノ °□°) ノ ⌒ ┻━┻

to indicate a new data-string. It has two outputs, 'sout' for the bits sent to the computer and 'xmitmt' to indicate to `uart_loop_mod` when new data is ready to be transmitted. On figure 8 is the state machine of the `tx_mod` module. It initiates when the 'shift_load' is set. Concurrently along all the states 'sout' is being set to the value of the LSB of the internal signal, 'reg_data'. In state 1 the 'reg_data' is shifted right and filled with 1's. It waits 16 clock cycles as per the UART protocol before sending the next bit. After sending 10 bits (including start and stop bit) it continues to state 4 where it sets 'xmitmt' and returns to state 0, ready for new data.

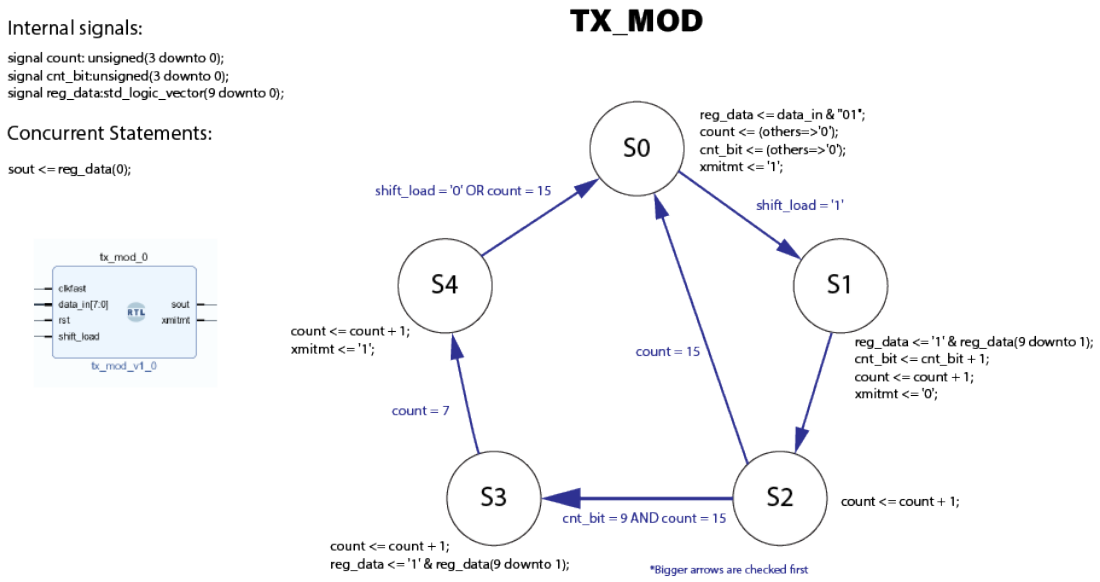


Figure 8: State machine for the `tx_mod` module.

4 Demo

There were unidentified problems with the SPI protocol, which led to an incomplete program. It is therefore not possible to provide a demonstration of the working code. On figure 9 is a sample output of the code. What this output suggests is that no data is received from the IMU. It is the same output as if there were no wired connection at all.

The IMU is correctly powered, as indicated by the red LED. A possible issue is that the IMU is not initiated correctly, and will not provide any data from any register per default without configuration. However, a method for writing to the registers where not produced in time.

(ノ °□°) ノ ⌒ ┻━┻

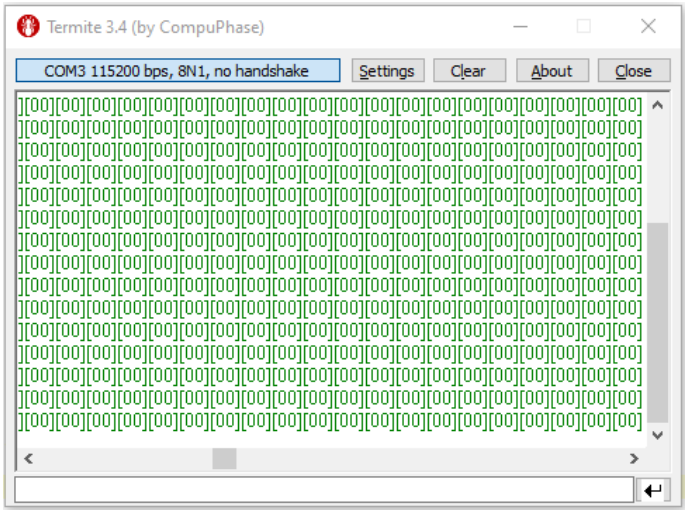


Figure 9: Sample of terminal output

5 Conclusion

Although the results were not comprehensible or fulfilling, further work should be done to achieve the targeted results. However, experience was gained from working with UART and the program Vivado, and while working with the SPI were not as successful as imagined, it has become more understandable.

(ノ °□°) ノ ⌒ ┻━┻