Cyclone IV EP4CE115F29 device

Ludovic Provost 300208450

Dexter Day 300192263

Group #16

Project: UART Design

CEG3155: Digital Systems II

Prof. Rami Abielmona

Due December 6th 2023

Table of Content

Objectives

The objective of this project is to design and build a complete UART in VHDL.

Upon completion, the student must be able to:

- Design, realize and test transmitter and receiver modules;
- Design, realize and test a baud rate generator;
- Demonstrate a complete understanding for the design of a UART and its interface in a real-time system.

Problem and Solution Discussion

Transmitter FSM:

The transmitter controller consists of 4 states and implements a counter to be able to count how many bits have been sent out. Its state diagram can be found in figure 1.1.1 and the design equations associated with it are found in figure 1.1.2.

Figure 1.1.1: Transmitter controller state diagram

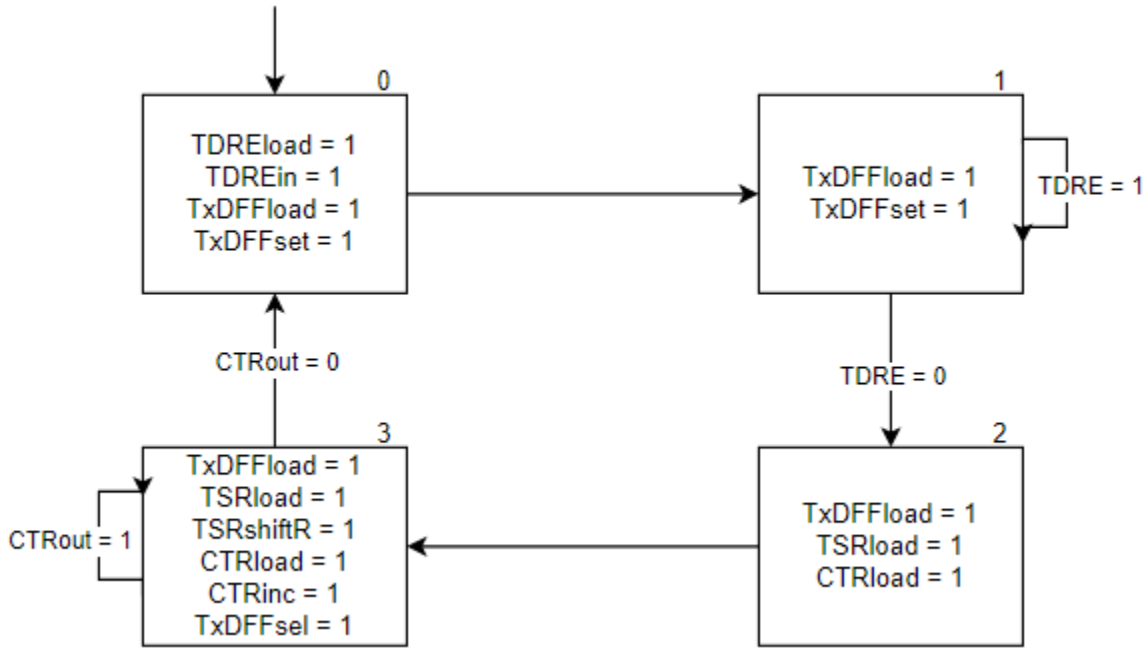


Figure 1.1.2: Transmitter controller design equations

Output	Equation
TDREload	y0’
TDREin	y0’
TxDFFload	y1 or y0
TxDFFset	y0
TSRload	y1
TSRshiftR	y1 and y0
CTRload	y1
CTRinc	y1 and y0
TxDFFSEL	y1 and y0

Receiver FSM:

The receiver needed two main counters: a counter to keep track of the clock cycles and a counter to keep track of how many bits have been sampled. The former should be able to track for 8 or 12 clock cycles, which was accommodated with a select: “CLKCTRSEL”. Since the ways in which our FSM should react to overrun errors and framing errors were not specified, we designed our receiver controller to go into a trap when an error is detected.

Figure 1.2.1: Receiver controller state diagram

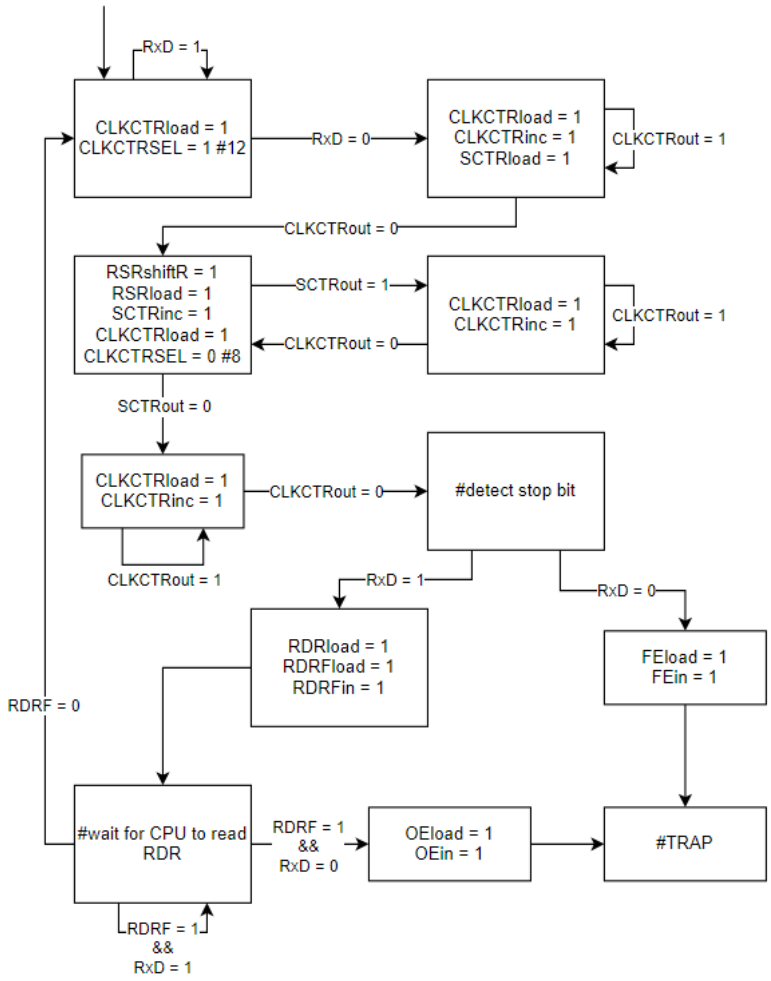


Figure 1.2.2: Receiver controller design equations

Output	Equation	Output	Equation
CLKCTRload =	y0 or y1 or y2 or y3 or y4	RDRload =	y6
CLKCTRinc =	y1 or y3 or y4	RDRFload =	y6
CLKCTRSEL =	y0	RDRFin =	y6
SCTRload =	y1 or y2	FEload =	y9
SCTRinc =	y2	FEin =	y9
RSRload =	y2	OEload =	y8
RSRshiftR =	y2	OEin =	y8

Complete UART component:

We decided to implement a control block for the status register of the UART. As can be seen in figure 1.3.1 below, the signals to change the TDRE and RDRF bits are muxed with the select being UARTselect. This gives full control of the SCSR to the UART FSM (through that AD) when UARTselect is asserted, otherwise the transmitter controller (TC) and receiver controller (RC) have control. We also decided against the UART FSM directly changing the SCSR register, since passing the load and write signals of each bit from the UART FSM to the UART would prove redundant. Therefore, we used the address decoder (AD) to change the RDRF and TDRE bits depending on the signals from the UART FSM. For example, If the UART FSM wanted to read from RDR, it would mean that RDRF was '1' but now needed to be changed to '0'. The AD could do that without the need for the UART FSM to take control of the UART to set the SCSR. The same logic was applied to the case where the UART FSM loads TDR. This would mean that TDRE would need to be deasserted. The AD's input/output table can be found in figure 1.3.2 below.

Figure 1.3.1: SCSR control

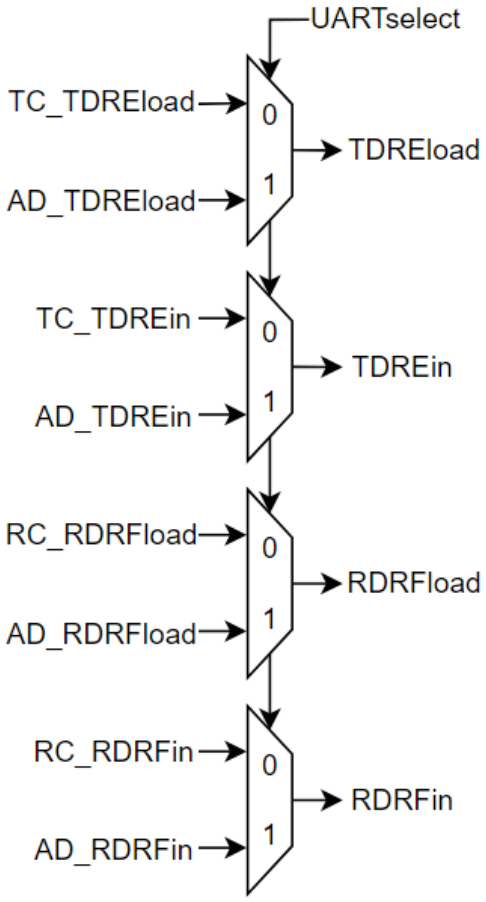


Figure 1.3.2: Address decoder logic

ADDR[1..0]	R/W*	DATA_BUS_IN	DATA_BUS_OUT	AD_TDREload	AD_TDREin	AD_RDRFload	AD_RDRFin
00	1	-	RDRout	0	-	1	0
00	0	TDRin	-	1	0	0	-
01	1	-	SCSRout	0	-	0	-
01	0	-	-	0	-	0	-
1d	1	-	SCCRout	0	-	0	-
1d	0	SCCRin	-	0	-	0	-

UART FSM:

The UART FSM ended up having a lot of states, all of which can be seen in figure 1.4.1 below. Standing out are the initialization state (state #0) that is used to initialize the SCCR. For the rest, the FSM mostly goes between 3 consecutive states: the first state is to send the appropriate character, and it is followed by a state that waits for the UART to be done sending as well as a state that also retrieves the sent data via the TxD and RxD of the UART being linked together. The appropriate character selection was done with an 8-to-1 mux which can be found in figure 1.4.3 below. The appropriate characters that represent the colors (Mcolor and Scolor variables in the figures below) are also the output of a 4-to-1 mux whose select signals are the State_Information bits. This circuit realization can be seen in figure 1.4.4 below. The binary values used for the characters were taken online and can be found in the figure 1.4.5. It is important to notice that we selected the character “M” = $(4d)_{16}$ to correspond to CHARsel = “000”. In retrospect, this was not ideal for debugging, because it means that when the current state doesn’t need to output a character, the CHARsel bits are not set which means that the DATA_OUT will appear as “4d” even if it is not affecting the result.

Figure 1.4.1: UART FSM state diagram

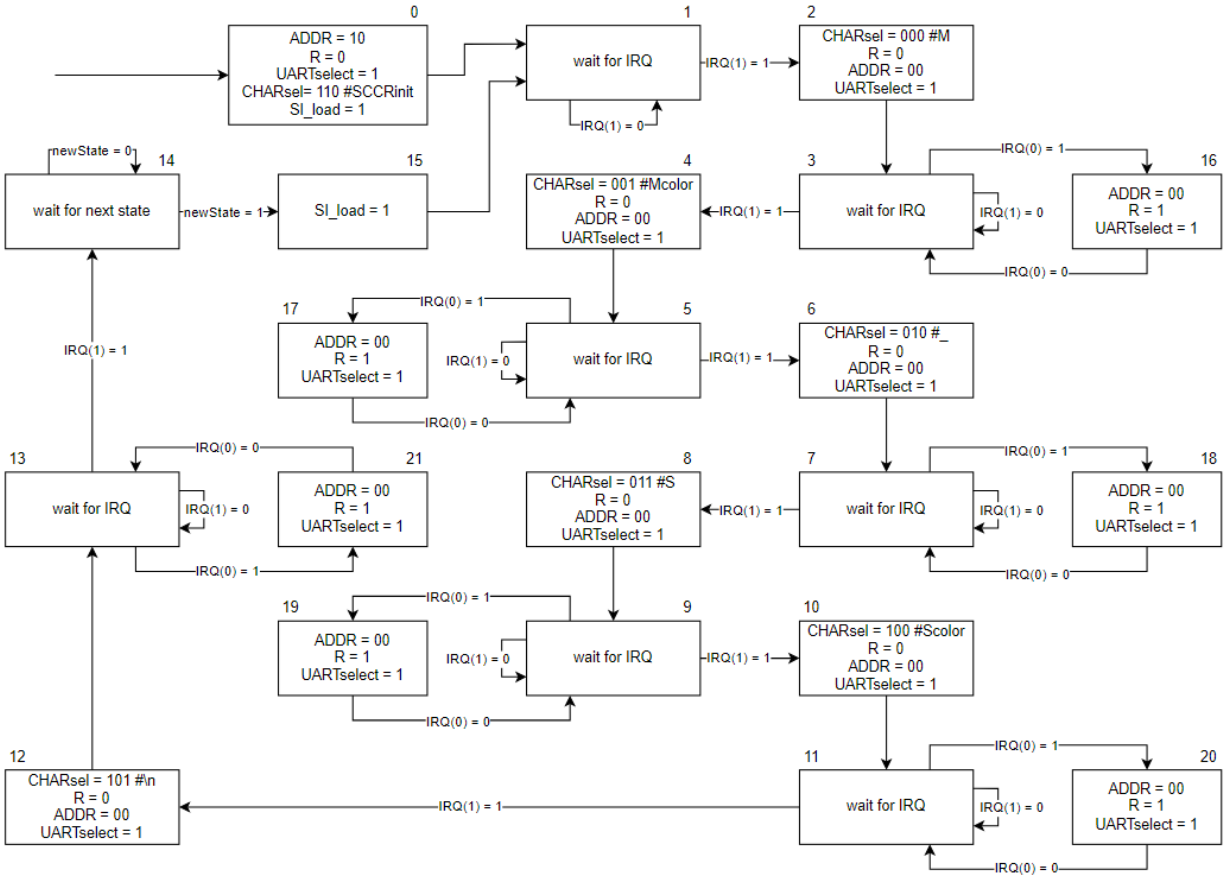


Figure 1.4.2: UART FSM design equations

Output	Equation	Output	Equation
ADDR(0) =	0	CHARSEL(0) =	y4 or y8 or y12
ADDR(1) =	y0	CHARSEL(1) =	y0 or y6 or y8
R =	y16 or y17 or y18 or y19 or y20 or y21	CHARSEL(2) =	y0 or y10 or y12
UARTselect =	y0 or y2 or y4 or y6 or y8 or y10 or y12 or y16 or y17 or y18 or y19 or y20 or y21	SI_load =	y0 or y15

Figure 1.4.3: Character selection logic

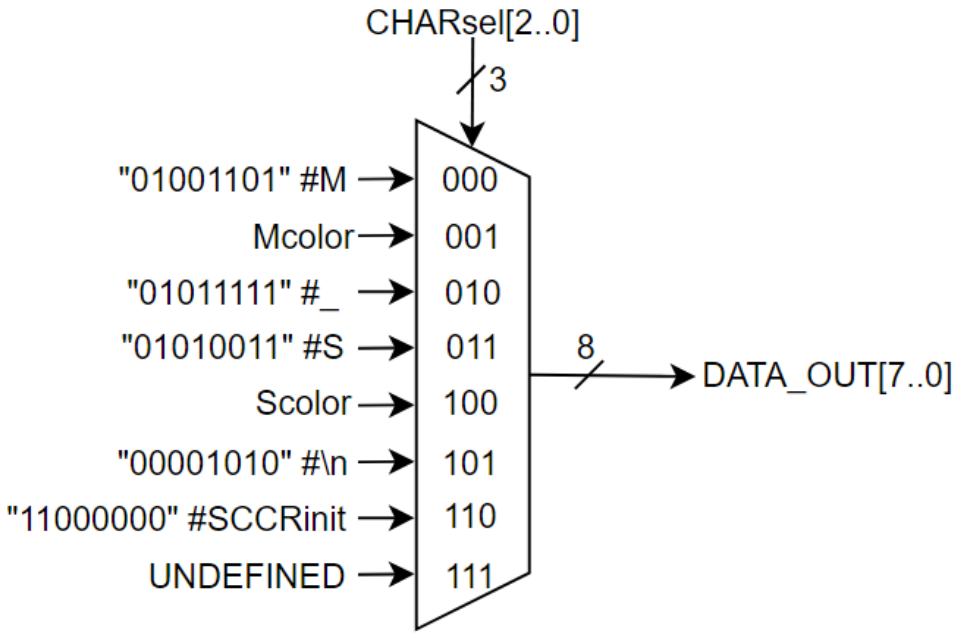


Figure 1.4.4: Color character selection logic

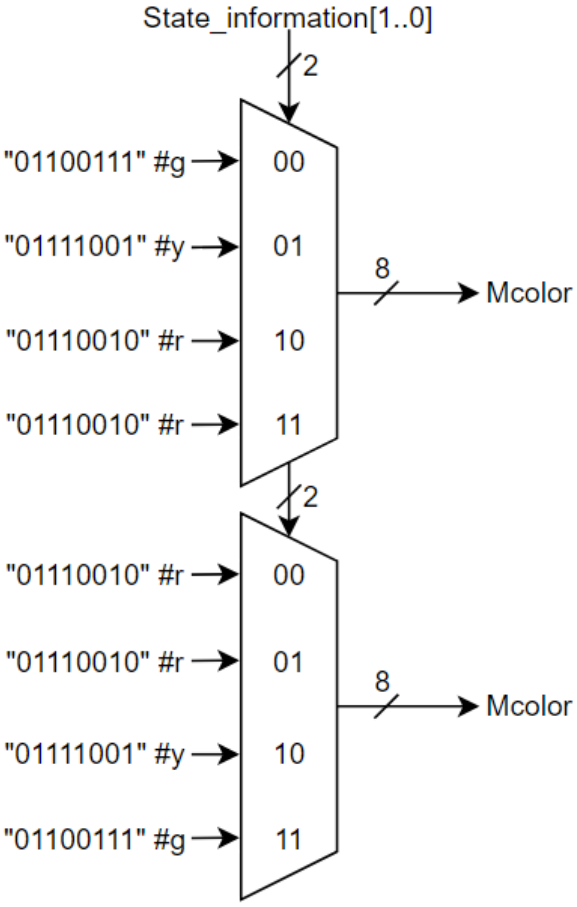


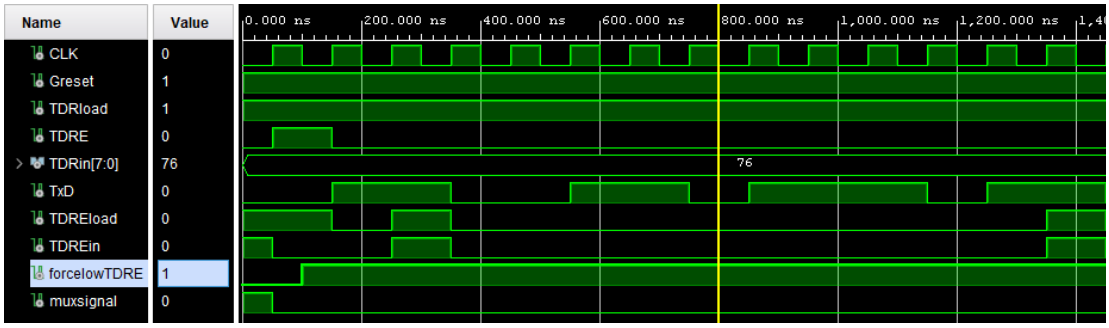
Figure 1.4.5: Characters used

symbol	Binary	HEX
M	01001101	4D
_	01011111	5F
S	01010011	53
g	01100111	67
y	01111001	79
r	01110010	72
Newline (\n)	00001010	0A

Verification

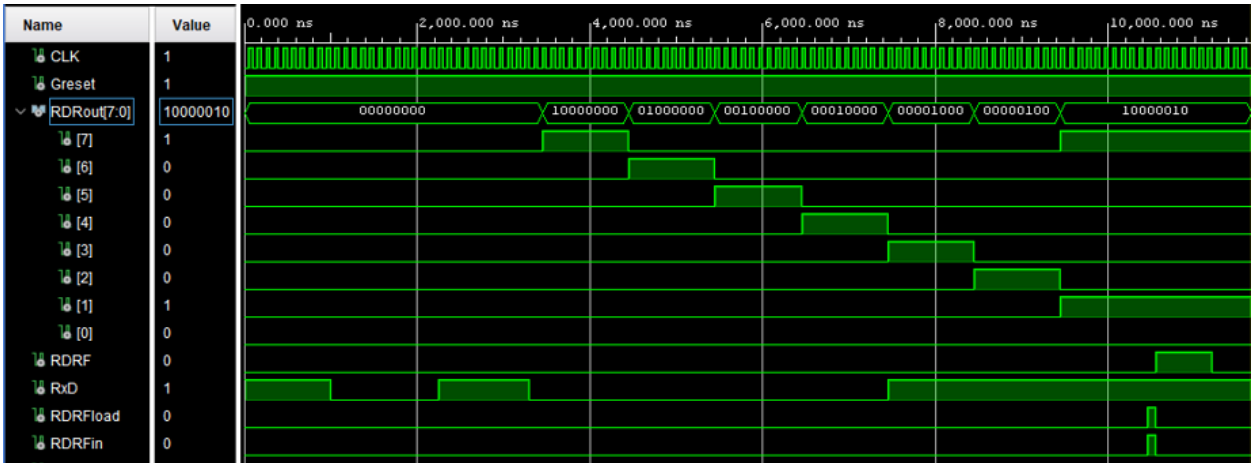
Functional Simulations of major components:

Figure 2.1.1: Simulation of Transmitter



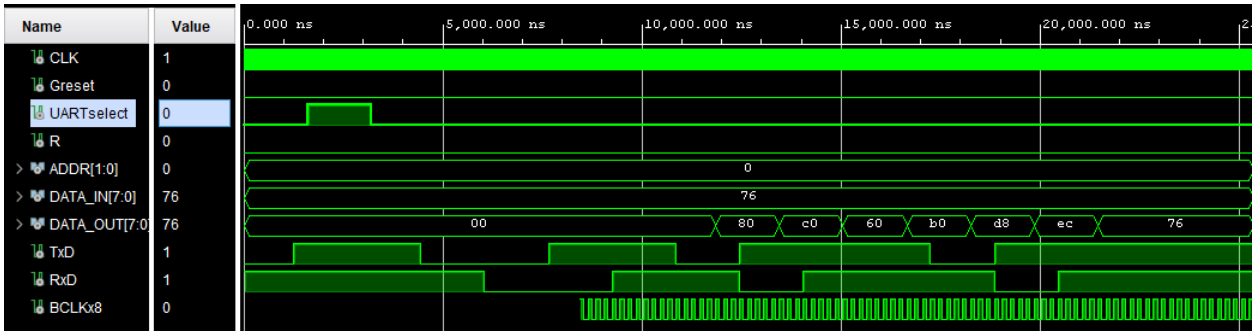
In this simulation screenshot, the byte to send was $(76)_{16}$ = $(01110110)_2$. Since transmission is done with LSB first and taking into account the start bit '0', we would expect TxD to take the following values: "001101110". The TxD = '1' at 160 ns and at ~1240 ns are considered the stop bits. Looking at the value of TxD in between, we can see that it matches the expected value of "001101110".

Figure 2.1.2: Simulation of Receiver



In this simulation, we were trying to receive “10000010”. Following the RxD signal, we can see that the waveform captured our received byte well and we can also observe that the RDR register in the UART ended with the correct value. We can also notice the time at which the RDRF value is set by the receiver controller.

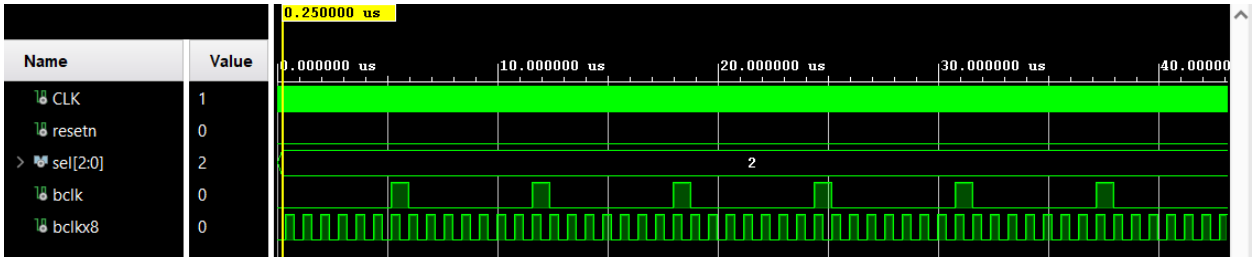
Figure 2.1.3: Simulation of UART with TxD = RxD



Since this simulation was done before the completion of the UART FSM, the following signals had to be set manually: UARTselect, R and ADDR. As expected, RxD matches TxD. We can see that the DATA_IN, which is the bus responsible for taking data from the UART FSM to the UART holds $(76)_{16} = (01110110)_2$: the value to be transmitted. DATA_OUT is the bus taking data from the UART to the UART FSM (RDR in this case). As expected, as the bits are sent through TxD and received through RxD, the value of the data received (DATA_OUT) slowly appears and shifts right and ends with the correct byte of $(76)_{16} = (01110110)_2$.

Testbenches Utilized in the Verification Process:

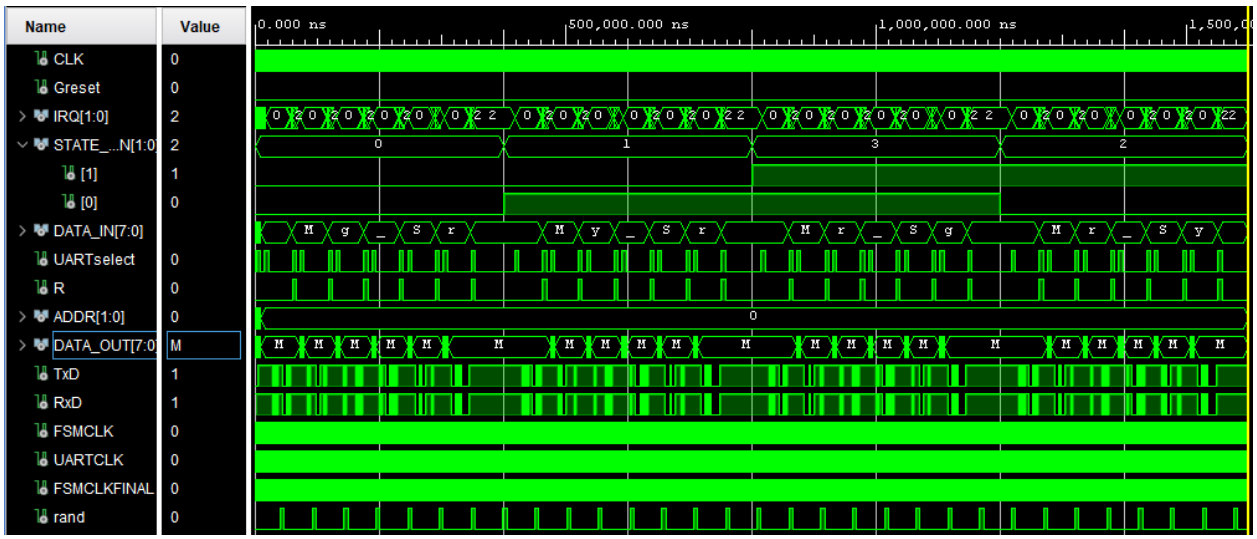
Figure 2.2.1: Baud Rate Generator simulation



The baud-rate generator is used to provide two separate divided signals from the clock. The first in this implementation is an output of the 256 counter which uses a mux to select a divided clock. Each bit of this counter's flip flops is equal to a division of 2, starting from the LSB to MSB. This clock is shown above in BCLKx8, the second clock is the expiration of a 8-bit counter that uses this previous clock as its input. This means that for every 8 cycles of BCLKx8, there should be one cycle of BCLK. This is successful as shown above. The select bits determine how much the original signal is divided, in this case it is by 4 as it is the second bit of the 256 counter.

Top-Level Simulation:

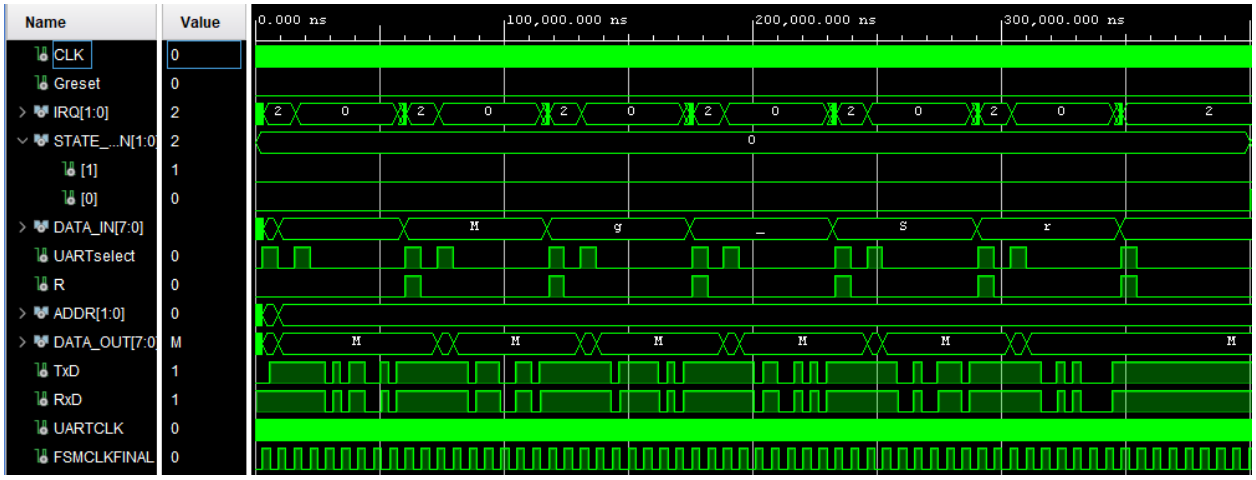
Figure 2.3.1: Full cycle of states



Following the 4th signal from the top: STATE_INFORMATION, we can see that each state is covered (“00”, “01”, “11”, “10”). DATA_OUT are the bytes that the UART FSM sent to the UART for the latter to send out through TxD, while DATA_IN are the bytes that the UART has received through RxD and sent to the UART FSM. Since TxD and RxD are connected, both DATA_IN and DATA_OUT are equal, but delayed. Since there is a lot to see, the next 4 figures (figure 2.32 to 2.3.5) show in more detail each one of the states. Since DATA_OUT can not be fully shown even when looking at 1 specific state, figure 2.3.6 offers a better view of the signal

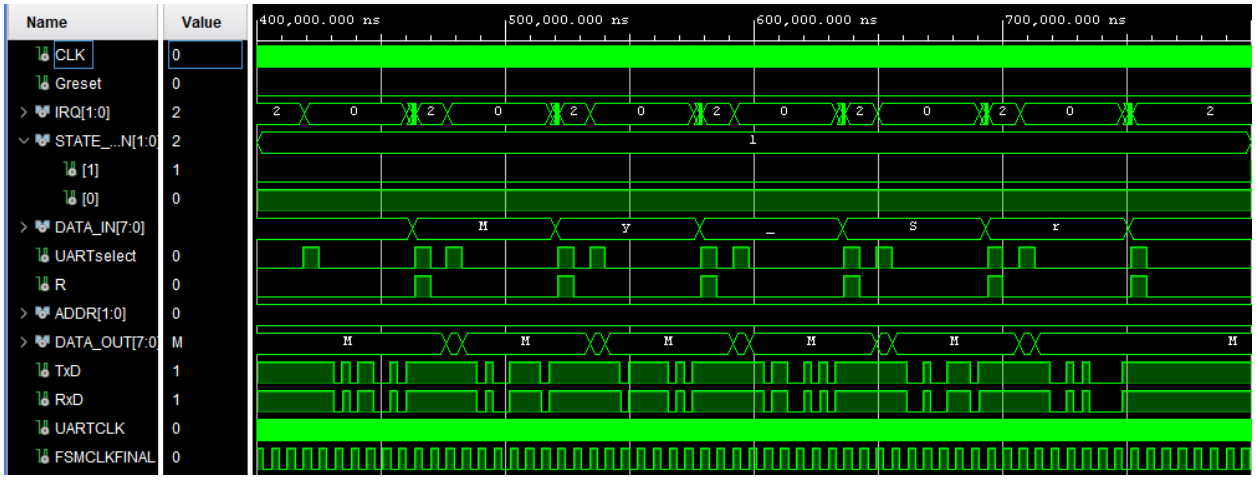
and shows a zoomed-in version of the beginning of the waveform shown in figure 2.3.3. Also note that the reason for which “M” is seen so often in the waveform was explained in the *UART FSM* sub-section of the *problem and solution discussion* section. This does not affect the UART because the AD was blocking that bus from accessing any registers in the UART.

Figure 2.3.2: State Mg_Sr



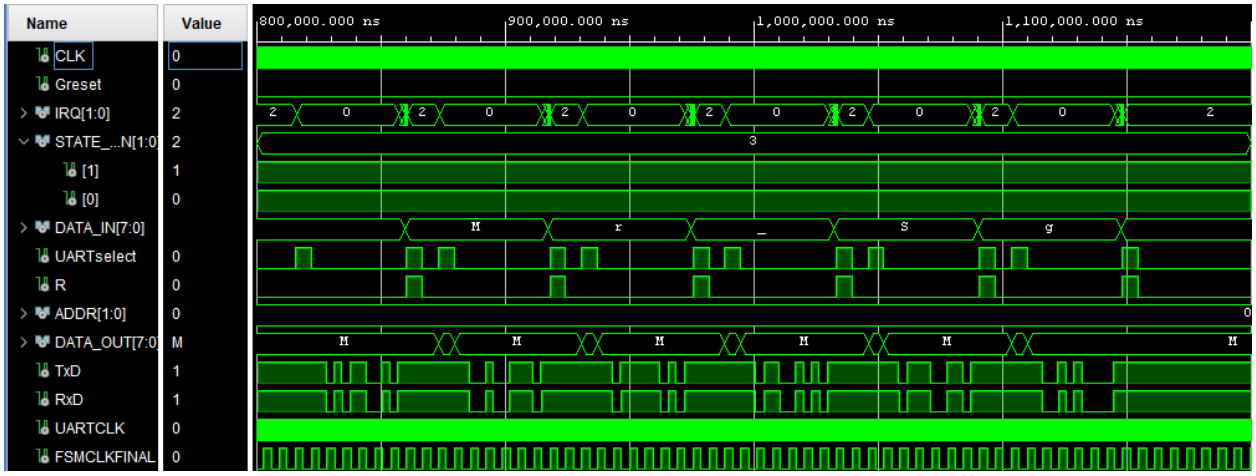
The state_information = “00” in the above waveform indicates the current state as being “Mg_Sr”. DATA_IN (the information sent through TxD and received through RxD) takes the expected values of “M”, “g”, “_”, “S”, “r” and “\n”.

Figure 2.3.3: State My_Sr



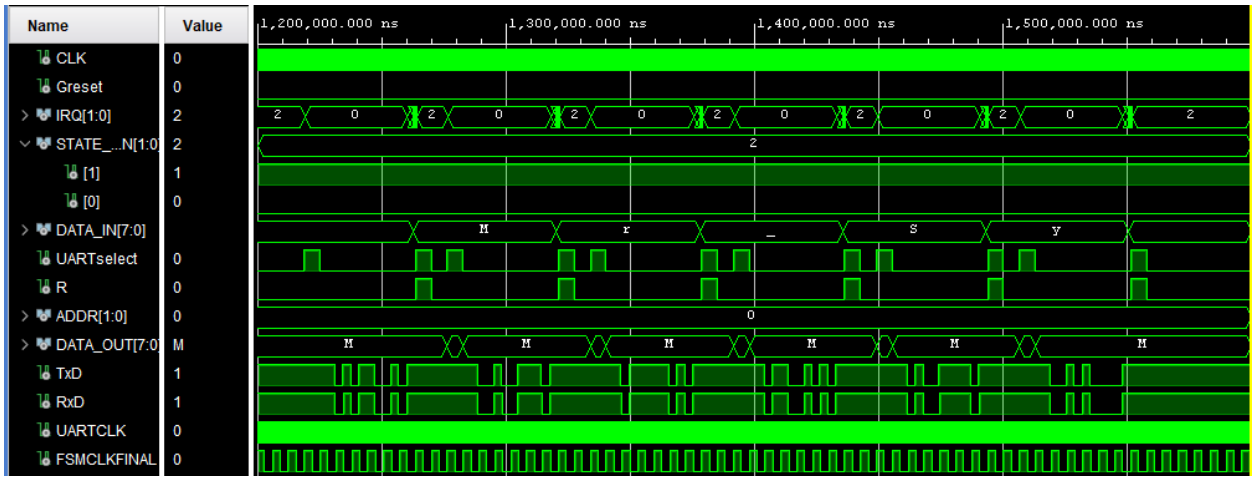
The state_information = “01” in the above waveform indicates the current state as being “My_Sr”. DATA_IN (the information sent through TxD and received through RxD) takes the expected values of “M”, “y”, “_”, “S”, “r” and “\n”.

Figure 2.3.4: State Mr_Sg



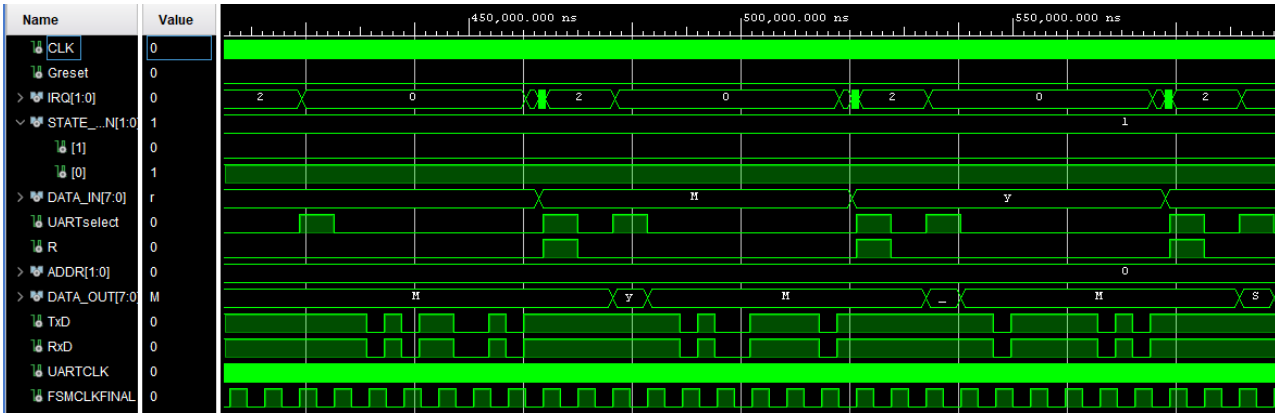
The state_information = “11” in the above waveform indicates the current state as being “Mr_Sg”. DATA_IN (the information sent through TxD and received through RxD) takes the expected values of “M”, “r”, “_”, “S”, “g” and “\n”.

Figure 2.3.5: State Mr_Sy



The state_information = “10” in the above waveform indicates the current state as being “Mr_Sy”. DATA_IN (the information sent through TxD and received through RxD) takes the expected values of “M”, “r”, “_”, “S”, “y” and “\n”.

Figure 2.3.6: Zoomed-in waveform of state My_Sr



On this zoomed in version of figure 2.3.3, we can see that DATA_OUT takes on the correct values of “M”, ”y”, “_” and “S” (the rest is not visible in this screenshot). Since a byte from DATA_OUT is only registered by the UART to be sent if UARTselect = ‘1’ and R = ‘0’, we can now see that the extra “M” values don't interfere with the UART’s function.

Lab Discussion

We encountered our first major problem when designing the receiver controller. With a total of 11 states, 4 inputs and 14 outputs, the state diagram looked like a mess to derive a transition table from. Getting design equations from that transition table through K-maps would have required us to use 16x16 K-map for each output which is completely unrealistic. To combat this issue, we decided to opt for a one-hot state encoding. We would be trading in complexity for quantity, as we would need 11 D flip-flops; one to represent each state. This also allowed us to get design equations directly from the state diagram. We ended up implementing our receiver FSM, transmitter FSM and UART FSM in the same way.

There were several problems when developing the baud-rate generator, mostly to do with the implementation of a 41 clock divider, which in the end was not used. This system worked in practice as it provided the two different clocks with a divide of 8, which was all the receiver and transmitter required to work together effectively. Besides that, only small hiccups were found in the logic of the generator and once solved it functioned properly as shown in Figure 2.2.1.

As can be seen in figure 1.2.1, we designed our receiver controller to react to overrun errors and framing errors by going into a trap. In practice, this should never be implemented since it disables the receiver until it is resetted, but we believe that it was outside the scope of this project to implement a dynamic system that responds well to lost frames.

We designed our interrupt generator slightly differently than the proposed solution. Instead of sending out 1 bit of information, 2 bits are sent: IRQ[1..0]. Previously, the UART needed to check the TDRE and RDRF bits in SCSR after an IRQ was detected to be able to tell which of the two between the receiver and transmitter needed attention. Now, IRQ(1) is asserted when the transmitter needs attention and IRQ(0) is asserted when the receiver needs attention.

In subsection *UART FSM* of section *problem and solution discussion*, we explained that the byte “4d” should be seen in the simulation waveform more often than it is actually used because the data bus is set to “4d” when the character select signal is “000”. In the text below figure 2.3.6, we revisited this particularity and concluded that it was insignificant because although the data bus takes on the value of “4d”, the registers in the UART will only be connected to the data bus if UARTselect = ‘1’ and R = ‘0’.

Conclusion and Appendices

After all elements of the UART and UART FSM were completed, the project could be tested and connected to lab 3 and verified through simulation, these simulations were correct and demonstrated that the lab was an overall success. The receiver, transmitter, and baud-rate generator were all successfully synthesized and tested helping to give us a much better understanding of the real-time function of the UART and allowing us to complete the project using structural VHDL. All final VHDL source files are commented and available here along with the simulation waveforms: Drive folder of VHDL files for the project