AGV Documentation - Sohan & Mohak

Started of the entire project with learning about FSM and UART.

Faced a small problem when just made a FSM sample module to get an idea regarding FSM and the hardware implementation of FSM.

Receiver:

- Didn't really face any problem in the receiver part, as the FSM part was already made and had to do a bit of changes.
- Got idea about the receiver part from some Github repos and was able to do it without really facing an issue.
- But later on realized that we will be needing a data validation bit to store the data faced a really small issue of timing here but had a a simple fix by adding a new cleanup state which fixed the issue of timing of the data validation bit by keeping it HIGH for one complete clock cycle.
- While looking into some UART Verilog implementation saw that there was a mid buad_rate check of the start bit and the stop bit, which was pretty clear why it was being used because UART is asynchronous and to get the individuals clock in sync there was this mid baud_rate check one to verify if the data is actually right and to get the clocks in sync.

Distance Processing:

- Well after completing the receiver module which basically converted the 8 series bits to 8 parallel bits, made another module which was named as the RxD module as it got these 8 parallel bits checked the conditions of header and then stored the data into different bytes sequentially, like the CT byte, 2 bytes for FSA, 2 Bytes for LSA and 2N bytes for sample data. Here had the idea that we will store the data of sample data in a 2d array which worked.

- But now we faced the issue of accessing the data from the 2d array in the RxD module in the distanceProcessing module to calculate the max_dist_angle and the min_dist_angle. So there were three ways now either to make a transmitter itself which sends these data in sequence to the distance processing module and then use a receiver again to read the data which will make the entire thing tedious, had another idea of incorporating the distance processing part in the RxD module itself which was the easiest thing to do and the last thing we could do is the creating a memory file.
- So we went with the memory file and made one just to store the sample data which worked really well, apart from this did not face any major issue, one potential issue was the division module which was sorted once it was mentioned that the CT will be of the form of 2^n making the entire process easier.
- Now for the transmitter module as well we needed a data validation bit from the distance processing part which ate one entire day due to some wrong initialization of the counter and updating it and there was the same issue of timing which we faced because of wrong resetting of the data validation bit in the idle block.
- Here while traversing across the whole memory file for finding the outputs we used a for loop then realized that for loops are not feasible in hardware, after which we initialized another counter and made a separate FSM for the for block which worked.
- We thought that the memory block was working because we were sending only one sample data and the outputs were correct but when we sent a few more samples it did not work because the sample data was not even being written inside the memory file, which we couldn’t fix till the last day and we had to push the distance processing prat into the receiver part and we named the entire RxD and distance processing part as the distance Processing part and the module which converted 8 series bits to 8 parallel bits as the RxD module.

Transmitter:

- The Transmitter converts parallel data received from the data processing unit to serial data.
- Initially the transmitter was implemented in a way that it transmits the 6 bytes all at once which meant we had only 4 states in the FSM that were Idle, Start, Transmit, Stop, but then we realized that the UART transmits only 8 bits(1 byte) at once, so we had to implement a few more states. Now we have the Idle, Start and Stop states as it is, additionally we have 6 Transmission states each representing 1 byte, and a start and stop state for each byte transmitted.
- There is a reset pin as well for resetting the transmitter to the idle state and a start_signal_bit to tell the transmitter to start transmitting.
- The r_clock is an internal register that acts as the timing reference for the shifting of
  data bits during transmission. It essentially synchronizes the parallel-to-serial conversion
  process with the baud_tick.