# 

The task was divided into two main parts. The first handled the output ports of the Rx module and the input ports of data processor while the other division handled the output ports of the data processor and the input ports of the Tx module. Specifically, one of us worked on interfacing between the Rx module and validating the input commands alongside wrapping the entire module together using sub entities and port mapping. The other person worked on selecting and propagating the data from the data processor to the Tx module in accordance to the input commands. We both established handshaking protocols between our entities and ensured synchronous communication with external modules.

# Command Parser

This module processes the P, L and ANNN commands from the Rx module. It will pass the ASCII characters of these commands to the Tx module for printing and verify the validity of these commands for processing. While the data is being processed, this module will ignore any commands and wait for the processing to finish.

# Command Processor

This module acts as the main interface (black box) for the command processor and encapsulates all the sub entities together through internal signals. This allows for the project to be easily split into sub components so that we can easily delegate tasks and work on different files individually. It also improves the testability of our code as it allows us to test the sequential logic of each architecture without having to sift through the signals of the entire command processor.

This modules currently acts as the main interface between the sub components and the Tx module to ensure that no more than one entity can transmit data into the printing buffer at a time.

# Tx Interface

Although this module has not been integrated for our current implementation, it demonstrates how we could improve modularity by introducing an interface between the command processor and the Tx module. It can be improved to allow for characters to be shifted into a buffer so that the command processor does not have to wait for the Tx module to finish before proceeding. Not only this, but space characters can be sent to the transmitter automatically after an end character has been printed.

# Data Commands

This data commands module is designed so that after an Annn command has been received, all NNN bytes from the data processor will be printed to the output. First it does a handshake with data parse when the command ANNN or aNNN has been received. Once one has been received it’ll set start high for one clock cycle and then wait for the data ready to go high. When it does the new byte will be passed from the data processor to the Tx module, afterward a space will be sent to the Tx module. Bytes and spaces will be sent to the Tx module in succession (with a start being sent to the data processor each time), until NNN bytes have been processed and seqdone goes high.

# L command

This module is designed to pass the needed data for an L command to the Tx module: the peak byte and the 3 bytes preceding and following it, in the order in which they were processed (with spaces in between each byte). First it does a handshake with command parse to recognise that an L command has been sent to the command processor. The 1st byte on DataResults from the data processor is then passed to the Tx module, followed by a space. The code then looks for all bytes on the DataResults line, until all bytes have been sent to the Tx module. The program then waits for the next L command to be received.

# P command

This module is designed to pass the data needed for a P command to the Tx module: the peak byte followed by a space, and then the index of the peak byte. First it does a duel handshake with the command parse module to be notified that a P command has been received by the command processor. The 4th byte in dataResults is then sent to the Tx module, a space is then sent to the Tx, followed by all 3 numbers in the maxIndex being sent. The program then waits for another P command to be received.