This document is to write down some considerations I made, so that either you can check if I’m right, and we could use some of this in a final report (if we have to do one). Probably you know most of the things written here but I write everything down so there aren’t any logically ambiguous steps.

## Processes in VHDL

Instructions in a process are executed serially, but instantaneously. So we have to take into account the order, but we can discard any problems about the latency between the beginning and the end of the process. This is fundamental for the following considerations.

## UART receiver

As the receiver unit is designed up to now, it receives the 8 bits in a series from the USB-port, and stores them in a *std\_logic\_vector*. Each component is set during a different state of the state machine, which has also the *idle* and *start – stop* states. Important, the process is triggered by the clock signal, which has a frequency extremely high compared to the incoming signal, so we can consider that the triggering of the process is continuous, without any problematic latency. The output (both *data\_valid* and *received\_data*) are null in all states except from the state named *bit\_7*, when the state machine is storing the signal into the last bit.

**N.B.:** at the moment, *stop* state is defined but not used. I propose to change the behaviour inserting the *stop* state and activating the output in this state, not in *bit\_7*.

When the state machine returns to idle, then all the outputs go back to 0. This means that, even if the data incoming from the USB port are in a row just one after the other, still we would have the stop and the start bits, so the state machine will always loop over all the states (idle state included, which is where it resets *data\_valid* and *received\_data* to 0).

This way we are sure that the output of the UART receiver will be 1 (output valid) only for very little time, and these phases will always be interrupter by a long time of 0 output (when the receiver is collecting all the data from the USB port).

## FIR filter

The coefficients of the FIR filter must be fixed, so their value must be set when defining the implementation of the architecture (as we already did, I checked and it’s correct).

Both for the coefficients and for the input/output data, we must understand if we want to use positive and negative numbers, or positives only. In the latter case, we should change all the types from *signed* to *unsigned*, so that we have one more bit to store the information on the values.

The fir filter must process the data **only** when the *data\_valid* is 1. This means that **every** process (which are executed simultaneously and instantaneously) must set all their signals to 0 anytime *data\_valid* is 0. When *data\_valid* becomes 1, all processes start processing all their data at the rising edge of the clock. The process which collects the processed signals and produces the output, should set the *out\_valid* to 1 and the processed data to the *std\_logic\_vector* to its correct value.

Since everything is instantaneous, it should work nice. We just have to be sure that the data incoming from the UART receiver remain constant for more than 1 clock cycle (which I’m now sure).

As soon as the *data\_valid* signal incoming from the UART receiver goes back to 0, every signal inside the FIR filter is reset to 0.

**To conclude, I strongly believe that *data\_valid* behaves exactly as the reset signal in the code we copied.**

## UART transmitter

It should work nice because it is triggered by the rising edge of the *clock* signal (continuously). The process is a state machine in which every state (excluding the *idle*, *start* and *stop* states) sends to the USB port the corresponding bit received as input.

**There is a problem:** the UART transmitter is designed to have a constant input (which is the output of the FIR filter). In fact the state of the state machine in the UART transmitter changes every clock cycle from the *idle* state to the *start* state, but all the other states change only every baudrate cycle. This is a problem, because the FIR filter probably resets its output to 0 much more rapidly than the time needed by the UART transmitter to flatten the 8 bits into one channel with the frequency of the baudrate. We should probably store the signal from the fir filter into a buffer which allows us to decouple the output time of the fir filter with the time needed by the transmitter to do its job.

Everything else seems working nice to me.