Innerspec Embedded Systems Engineer Assignment

In this work I will present the tasks performed in relation to the practice for the job position of embedded systems engineer for Innerspec.

Although in this week, I have been trying to make contact with Qt, I have not managed to elaborate a UI similar to the mentioned one, mainly due to the question of the embedded Chart. So my work is mainly focused on the VHDL firmware of the FPGA.

A Basys3 development board was provided by Innerspec on Wednesday 20/05 at morning. So even though I had read the assignment on Friday, I was unable to start downloading designs to the chip until that day.

Regarding my prototyping tasks, I was working at home without the availability of a waveform generator. So I simulated the analog input with a potentiometer acting as a resistive divider.

In the JXADC PMOD connector, I used the 3.3V signal and its ground (pins 6-12 and 5-11 respectively). I connected a fixed resistor R1 at the top of the 1.1Mohm value divider and a 500kohm value potentiometer at the bottom. In this way, using the full linear range of the potentiometer, the one-volt limitation of the analog input channels is not exceeded.

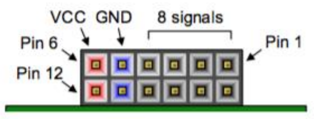
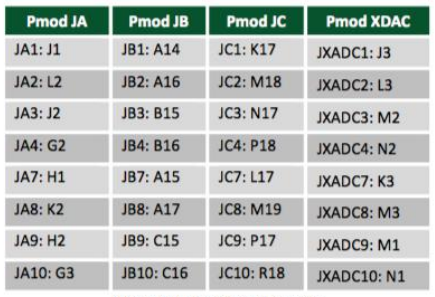
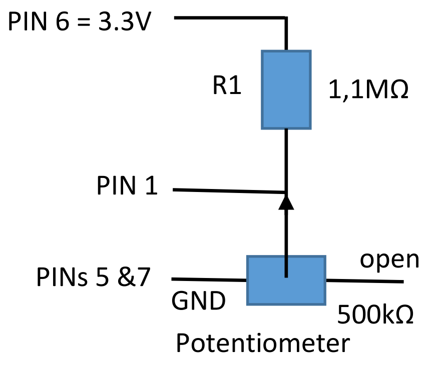


Figure 20. Pmod ports; front view as loaded on PCB.



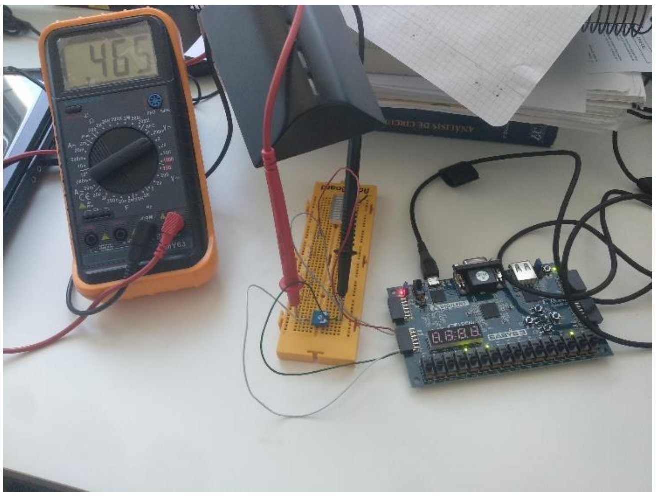
Pmod JA	Pmod JB	Pmod JC	Pmod XDAC
JA1: J1	JB1: A14	JC1: K17	JXADC1: J3
JA2: L2	JB2: A16	JC2: M18	JXADC2: L3
JA3: J2	JB3: B15	JC3: N17	JXADC3: M2
JA4: G2	JB4: B16	JC4: P18	JXADC4: N2
JA7: H1	JB7: A15	JC7: L17	JXADC7: K3
JA8: K2	JB8: A17	JC8: M19	JXADC8: M3
JA9: H2	JB9: C15	JC9: P17	JXADC9: M1
JA10: G3	JB10: C16	JC10: R18	JXADC10: N1

Table 6. Basys 3 Pmod pin assignment.



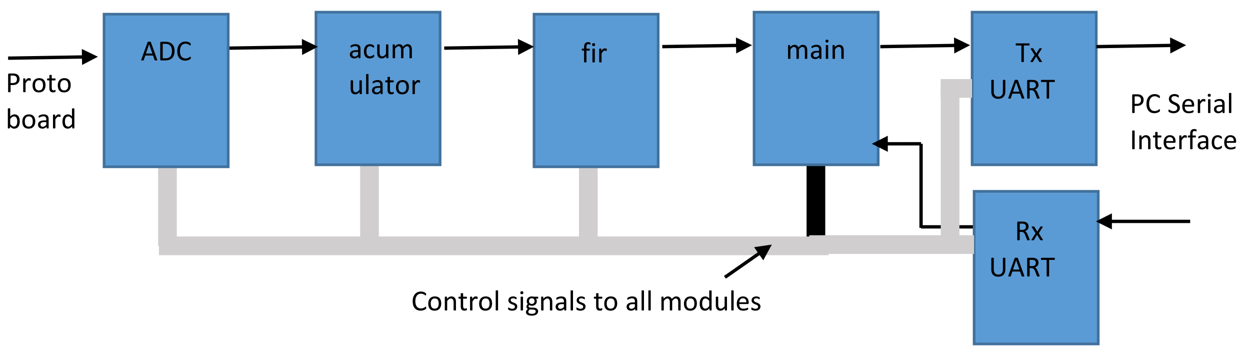
PMOD connections

Below is a photograph of the working situation. I use the multimeter to check if the values we are measuring though the serial match with the real input.


Basys3 with potentiometer

Next, I will describe how the different stages of the FPGA hardware design are structured. The layout consists of 6 .vhd files and this is how they are interconnected.

- main.vhd
- uart_tx.vhd
- uart_rx.vhd
- xadc_wiz_o.vhd
- acumulator.vhd
- fir_filter.vhd

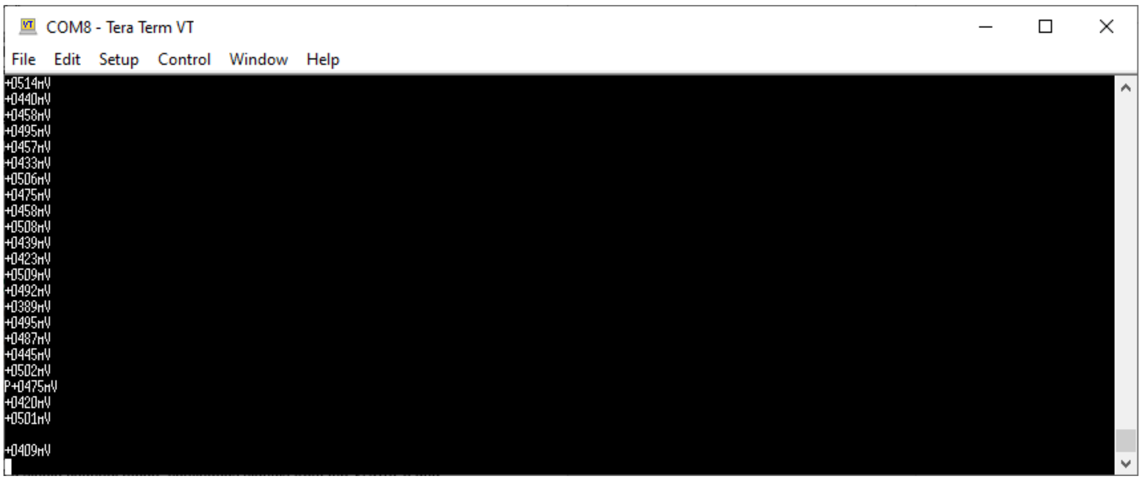


Block Diagram

For the interpretation of the commands from the UI, I have created a state machine in the serial reception. The PC will deliver the following commands:

- B↵: Starts
- Q↵: Stops
- A↵: Averaging Samples
- S↵: Sampling Rate
- D↵: Digitalize
- C↵: Coefficients

In the “Averaging samples”, “sampling rate”, “digitizer” and “coeffs” states, the machine states waits for the user to enter the corresponding data following by an enter.
Since all communication is via serial, I have had to develop a function package to work with ASCII characters. Functions to convert to ASCII and to convert from ASCII.



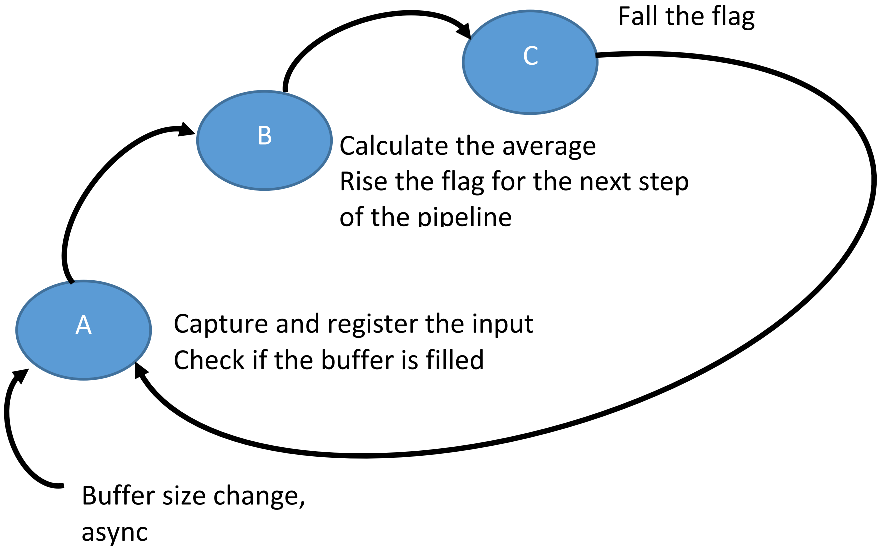
```
+0514mV
+0440mV
+0458mV
+0495mV
+0457mV
+0433mV
+0506mV
+0475mV
+0458mV
+0508mV
+0439mV
+0423mV
+0509mV
+0492mV
+0389mV
+0495mV
+0487mV
+0445mV
+0502mV
P+0475mV
+0420mV
+0501mV

+0409mV
```

Serial Interface with Local echo “P” and “↵”

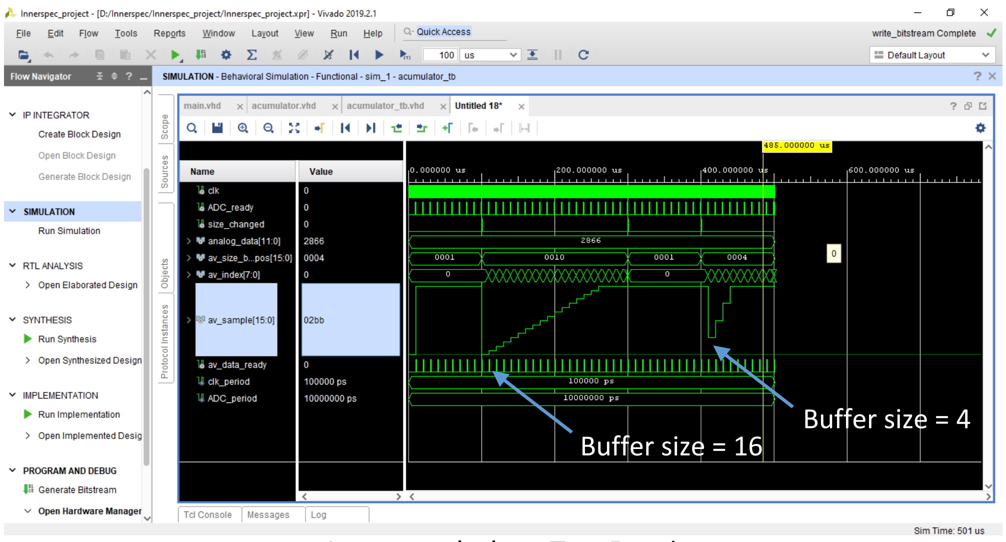
acumulator.vhd

ACUMULATOR_FSM

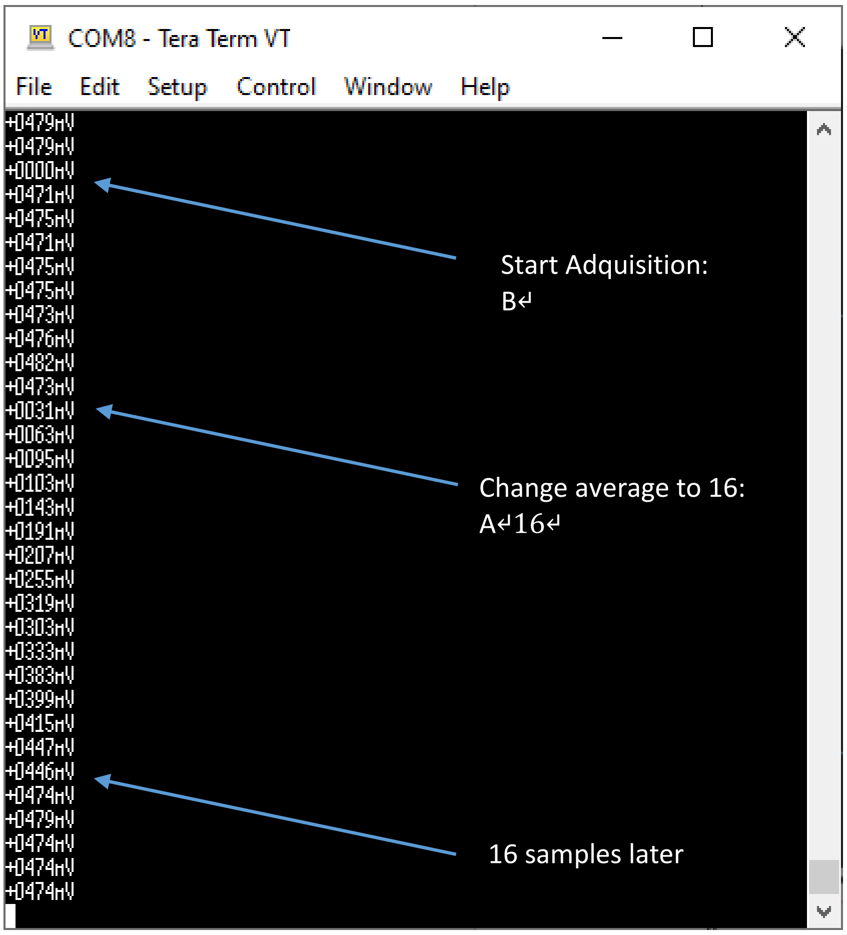

Fall the flag
C
B
Calculate the average
Rise the flag for the next step of the pipeline
A
Capture and register the input
Check if the buffer is filled
Buffer size change, async

Average calculator FSM

To simulate this component, a test_bench has been generated. The input *analog_data* is a constant, which is what we acquired from the ADC in my situation. In the bench test it can be seen that when changing the buffer size to run the average, the accumulator response time changes. In 100us, *av_buffer_size* = 16 and in 400us *av_buffer_size* = 4;



Average calculator Test Bench



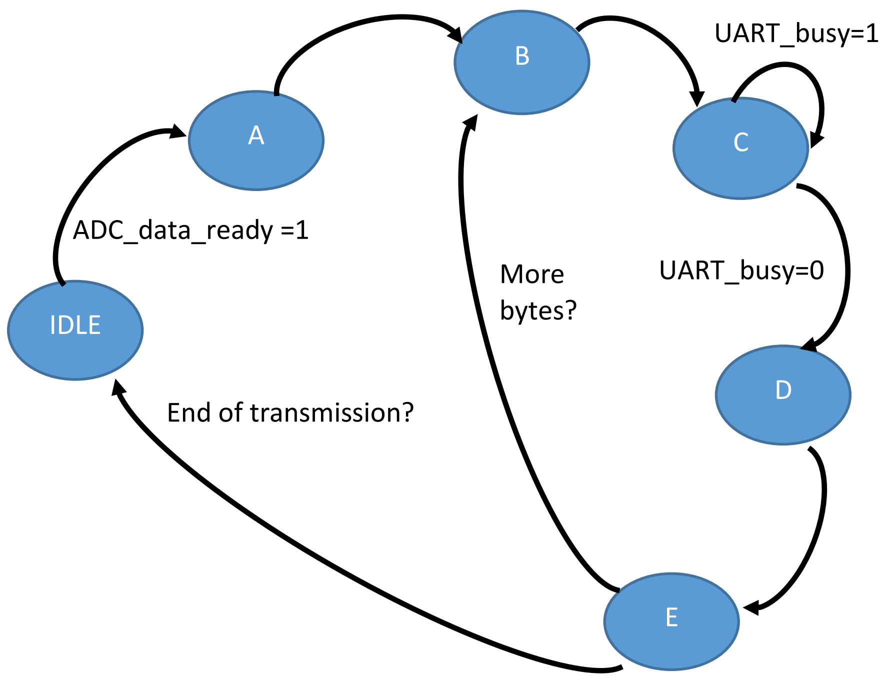
```
+0479mV
+0479mV
+0000mV
+0471mV
+0475mV
+0471mV
+0475mV
+0475mV
+0473mV
+0476mV
+0482mV
+0473mV
+0031mV
+0063mV
+0095mV
+0103mV
+0143mV
+0191mV
+0207mV
+0255mV
+0319mV
+0303mV
+0333mV
+0383mV
+0399mV
+0415mV
+0447mV
+0446mV
+0474mV
+0479mV
+0474mV
+0474mV
+0474mV
```

Average Calculator Working

main.vhd

In the main, all other components are instantiated. It also contains three state machines to transmit through the serial port and to receive and decode the orders.

TX_FSM

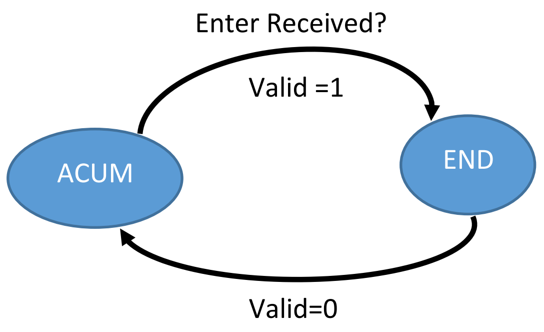


UART Transmission FSM

A: read data and convert it to ASCII
B; send byte to UART
C: data_valid = 1;
D: data_valid = 0
E: wait until end of transmissión and check end of chars to send

RX_FSM

This little state machine is only in charge of accumulating characters received by the serial port and waiting for the intro to arrive. When so, it activates a flag to call the main state machine of the reception.

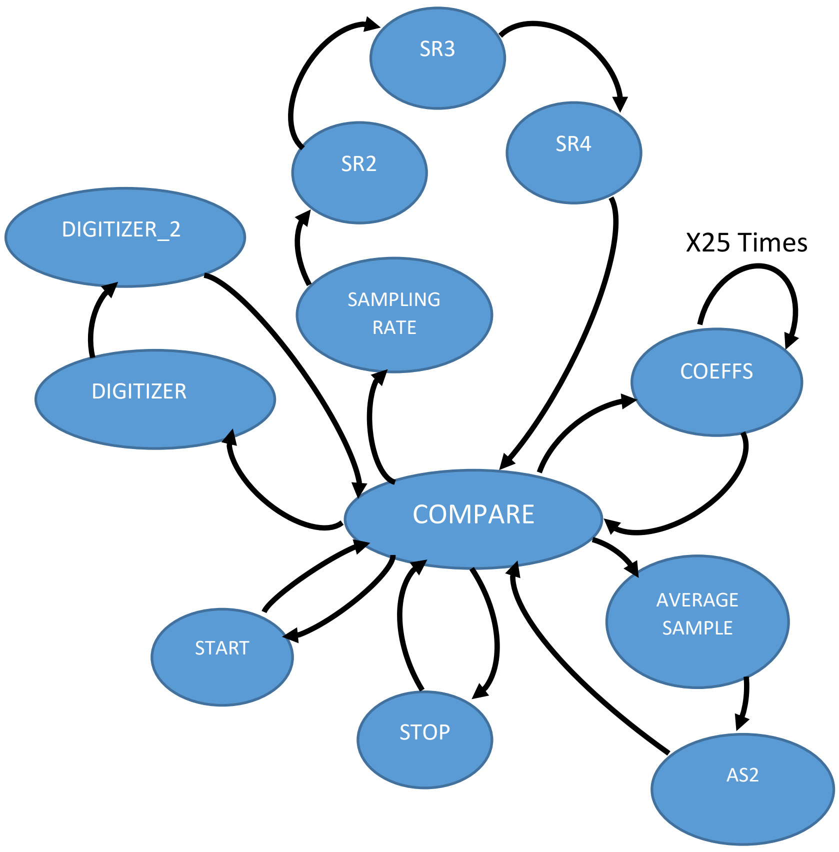


UART Reception FSM

ACUM: Store data (max 32) until a CR+NL is received.
END: State to send the Data Valid pulse.

MAIN_RX_FSM

Transition diagram between states when a correct order from the uart is received



UART Decoding FSM

SAMPLING RATE: read new adquisition frequency, and configure ADC. The three substates are to disable readings when changing the frequency.
DIGITIZER; read digitizer factor. The substate is to check if we have read a correct positive value.
START: rise ADC_Start
STOP: ADC_Start =0;
AVERAGING SAMPLES: read the number of samples to do the average.The next substate is to check if exceed the maximum size of the buffer or if its negative
COEFFS: Stay in the same state during the transmission of the 25 FIR coefficients.

UART_RX.vhd
115200 bauds, 8 bits plus start and stop and new line CR+NL.
Module taken from Nandland.

Waits for the CR+NL to interpretate a correct order.

Max length buffer, 32 chars

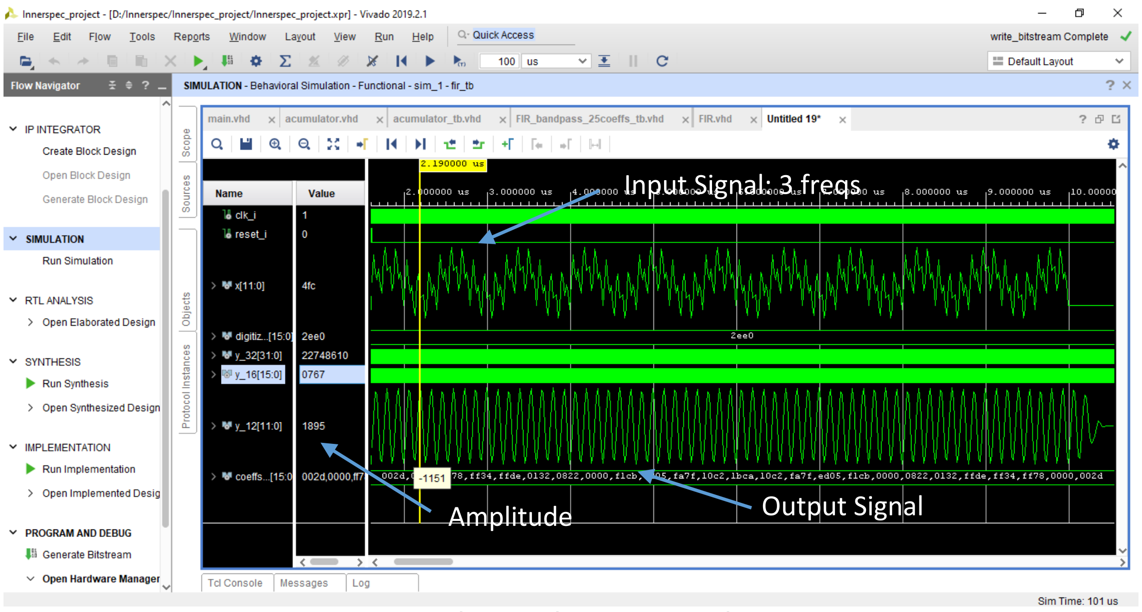
UART_TX.vhd
115200 bauds, 8 bits plus start and stop and new line CR+NL
Module taken from Nandland

Sends the outputs data from the filter.

fir_filter.vhd

The FIR filter model has been taken from the vhdl.es website. It is a model for a low pass filter but with the python script of the repository another set of band pass coefficients can be generated. That script is already working and my idea was to do it in the high level UI and then send the coefficients through the serial port. But as I don't have UI, I included a state in the RX_FSM to send the 25 coefficients through serial.

First calculating the coefficients with the python script and then generating an input file, that same filter was run in a functional test_bench to see its behavior. The attached screenshot shows a wave with three frequencies at the input and as the output only contain the center frequency.

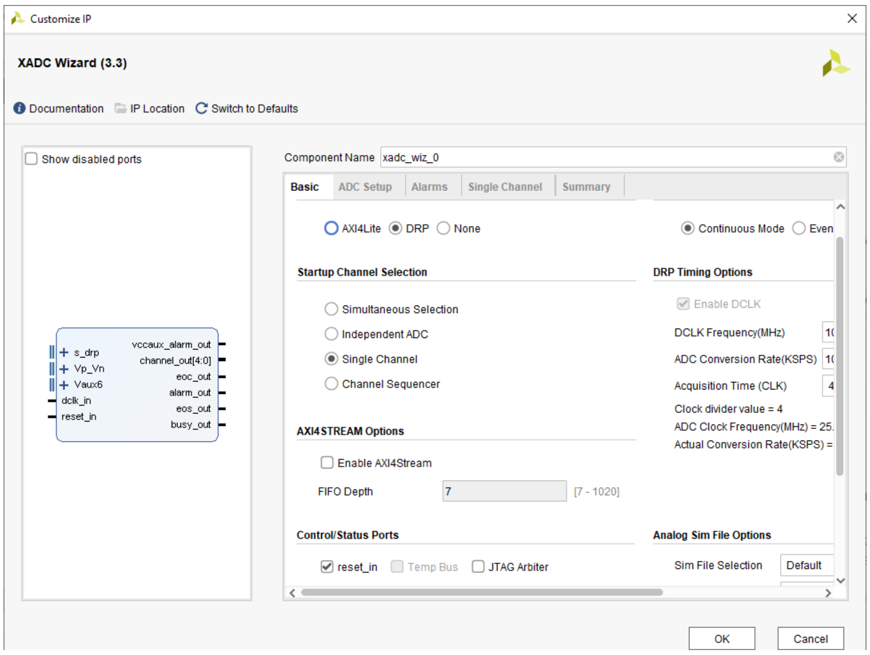

Innerspec_project - [D:/Innerspec/Innerspec_project/Innerspec_project.xpr] - Vivado 2019.2.1
SIMULATION - Behavioral Simulation - Functional - sim_1 - fir_tb
Input Signal: 3 freqs
Amplitude
Output Signal
Sim Time: 101 us

Name	Value
clk_i	1
reset_i	0
x[11:0]	4fc
digitiz..[15:0]	2ee0
y_32[31:0]	22748610
y_16[15:0]	0767
y_12[11:0]	1895
coeffs..[15:0]	002d,0000,ff7

Band Pass Filter Test Bench

With the help of this test bench, the value of the filter digitizer has been calculated to adjust the output for that input to its maximum dynamic range. The start value of that scale factor is 12000. You can see how for this value of digitizer factor, the amplitude of the signal is around 2000 what is the maximum (2048) for a 12 bits signed signal.

ADC:



Customize IP

XADC Wizard (3.3)

Documentation   IP Location   Switch to Defaults

Show disabled ports

Component Name xadc_wiz_0

Basic | ADC Setup | Alarms | Single Channel | Summary

AXI4Lite [✓ DRP] None   [✓ Continuous Mode] Even

Startup Channel Selection

Simultaneous Selection
Independent ADC
[✓ Single Channel]
Channel Sequencer

DRP Timing Options

[✓ Enable DCLK]
DCLK Frequency(MHz) 1
ADC Conversion Rate(KSPS) 1
Acquisition Time (CLK) 4
Clock divider value = 4
ADC Clock Frequency(MHz) = 25.
Actual Conversion Rate(KSPS) =

AXI4STREAM Options

Enable AXI4Stream
FIFO Depth 7 [7 - 1020]

Control/Status Ports

[✓ reset_in]   Temp Bus   JTAG Arbiter

Analog Sim File Options

Sim File Selection Default

OK   Cancel

XADC Wizard

As an analog input I have configured the XADC thanks to its IP wizard to acquire, in a single channel, unipolar mode, and continuous signals from the VAUXP6 and VAUXN6 channels (pins J3 and K3 of the FPGA respectively). The XADC has a DRP interface and all alarms have been disabled.

Future tasks

This is the work that I have been able to do during this week. As tasks that have been incomplete or need more time I would highlight the following:

- Restructure code so that it can be interpreted easier.
- More debugging in some aspects, for example the reception with the uart in hardware.
- Try to find other programming methodologies so that the implementation in the FPGA will be more efficient. Regarding to this, I must say that in the current design the running phase synth_design has lasted up to an hour and a half.
- Work in the interface to have visual information and make the use more friendly than in the command line of the COM.
- Try the whole system with a waveform generator.