CASPER Workshop

Tutorial 4: Wideband Pocket Correlator

Dev. By : W. New (Version 1)

Doc. By : Irappa M. Halagali , Mekhala V. Muley & Shelton Gnanaraj J. (version 2)

Expected completion time: 2hrs

Contents:

1 The Hardware and software required for this tutorial.

1. PC : Dell Intel(R) Core(TM) i3 CPU 530 @ 2.93GHz width 64 bit & 4GB RAM
2. OS : Linux 2.6.35-30-generic #54-Ubuntu 10.10 SMP x86_64 GNU/Linux
3. Matlab : 2008a
4. Xilinx : version 11.5
5. Casper : gits_100511
6. corr pack : corr-0.6.5
7. Python : version 2.6
8. minicom : version 2.4 (compiled on Jun 3 2010)
9. ROACH unit with iADC card : version 1.0 Rev 3 2009,
   uboot : uboot-2010-07-15-r3231-dram ,
   Linux Kernel Image : uImage-jiffy-20091110,
   iADC : BEE2 DUAL 1 GHz ADC BOARD version 1.1
10. Signal generator to feed clock of 800MHz , 0dbm to ROACH unit through iADC's clk_i input.
11. Input signals from waveform generator should be of -13dbm@400 MHz BW (Total power over BW) to the I+ & Q+ inputs of iADC.

2 Introduction

In this tutorial, you will create a simple Simulink design which uses the iADC board on ROACH and the CASPER DSP blockset to process a wideband signal, channelize it and output the visibilities through ROACH's PowerPC.

'Introduction to Simulink' Introduction to Simulink, 'Implementation of green block' Coarse Delay Block - A Green Block, '10 Gbe Interface' Tutorial_10GbE and are reasonably comfortable with Simulink and basic Python. We will focus here on higher-level design concepts, and will provide you with low-level detail preimplemented.

3 Background

Some of this design is similar to that of the 'The Wideband Spectrometer' tutorial. So completion of that is recommended.

3.1 Antennae and Baselines

When doing correlation on a set of antennae we introduce the term baseline. A baseline is the product of the signal from two antennas. We calculate all baselines. For example, if we have 3 antennae, A, B and C, we need to perform correlation across each baseline, AB, AC and BC. We also need to do auto-correlations, which will give us the power in each signal. ie AA, BB, CC. We will see this implemented later.

3.2 Polarization

Dish type receivers are typically dual polarized (horizontal and vertical feeds). Each polarization is fed into separate ADC inputs. When correlating these antennae, we differentiate between full Stokes correlation or a half Stokes method. A full Stokes correlator does cross correlation between the different polarizations (i.e. for a given two antennas, A and B, it multiplies the horizontal feed from A with the vertical feed from B and vice-versa). A half stokes correlator only correlates like polarizations with each other, thereby halving the compute requirements.

Our correlator here is a half Stokes correlator. We will be feeding the similar polarizations of two antennae to the two inputs of the single ADC.

3.3 The Correlator

The correlator we will be designing is a 2 antenna single channel correlator i.e it takes the similar polarization from two antenna and correlates the signal, hence we will be populating only one adc board on the ROACH.

4 Setup

The lab at the workshop is preconfigured with the CASPER libraries, Matlab and Xilinx tools.
Please refer the file "LOCATIONSandFILES.pdf" in the home/Desktop area or LOCATIONSandFILES slides displayed , for the locations/directories and files information required in the tutorial. Note : The Date and Time portion of the BOF file name will be different! It depends upon when (Date & Time) you complile your model file !

Note : All the following cable connections and entries in the /etc/* files of the workshop PCs are already done.You are not required to do any of the following setup and they are informatory in nature.You can verify points 1 to 4 on the setup you are working on and if you have any doubts regarding them kindly contact the lab instructor. Kindly go through point 5 to decide the way you will implement the tutorial.

1. Connect the Serial port cable between the ROACH board's P2 connector and serial port of the PC (on which minicom program exists).

2. Connect the Ethernet cable to J25 port of the ROACH board from the PCs eth1 port. /etc/ethers file should have mac address and corresponding ip address. In the /etc/network/interfaces file , eth1 should be configured. And in the file /etc/hosts , ip address and corresponding roach board(host) name entry to be done.

3. Feed the clock of **600MHz , 0 dbm (~630mvPkPk without any splitter)** to the clk_i input of the iADC card (which is plugged in the ZDOK 0 connector near to mmc card/power supply) from the signal generator. The python script **"[TUT4_CONFIG_PYSCRIPT_FILE]"** generates the soft sync and hence there is no need to give a external sync pulse.

4. Connect the input signals to I+ & Q+ of the iADC 0 (in the ZDOK 0 connector) from the noise generator. The signals will be referred as a(adc0I+), b(adc0Q+). The input signals should be of -13dbm(~282mvPkPk with 2 way power splitter)@300 MHz BW (total power over BW) at the iADC card input. A low pass filter of 200 Mhz BW is introduced in the signal path to show a band shape. The output will generate two self signals "aa", "bb" and one cross "ab" for this tutorial. Thus we get 2 auto-correlations and 1 cross-correlation.

5. Either Create your own directory at "[USER_DIR]" where you can save and compile your model file or save any work that you may do. There are three ways to implement this tutorial.
A)You can either copy the mdl file "[TUT4_MDL_FILE]" from the the area "[STD_MDL_DIR]" to the directory that you have created at "[USER_DIR]" and compile it in the MSSGE (Matlab-Simulink-System Generator) environment

OR

B)You can use the bof file kept in the area "[FPGA_PROG_BOF_DIR]/[TUT4_BOF_FILE]" to directly program (using the python script explained in "Software") the FPGA and look at the results

OR

C)Follow the steps given below to create the mdl file similar to the file "[STD_MDL_DIR]/[TUT4_MDL_FILE]".

6. **Start the matlab :**

$ cd [MATLAB_START_DIR]
[MATLAB_START_DIR]$ ./[MATLAB_START_FILE] &

5 Creating Your Design

Create a new model:

Start Matlab and open Simulink (either by typing simulink on the Matlab command line, or by clicking the Simulink icon in the taskbar). Create a new model and add the *Xilinx System Generator* and *XSG core config* blocks as before in Introduction to simulink.

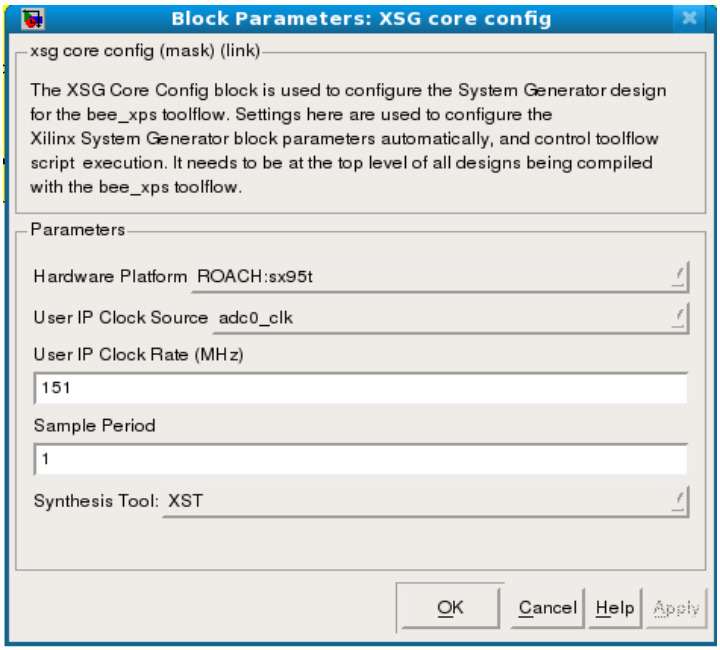
5.1 System Generator and XSG Blocks

By now you should have used these blocks a number of times. Pull the **System Generator** block into your design from the Xilinx Blockset menu under Basic Elements. The settings can be left on default.

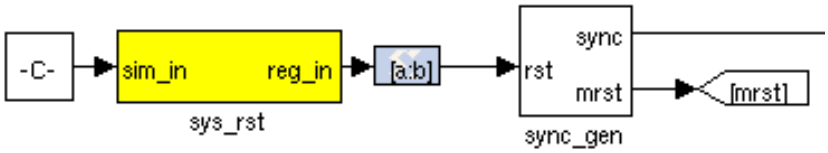
The **XSG** block can be found under the BEE_XPS System Blockset. Set the Hardware platform to ROACH:sx95t, the Clock Source to adc0_clk and the rest of the configuration as the default.

Make sure you have an ADC plugged into ZDOK0 to supply the FPGA's clock!

Set the XSG core config block as shown below



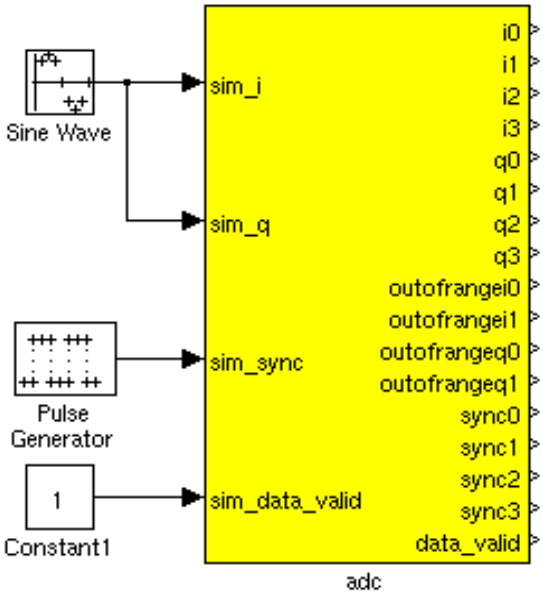
5.2 Sync Generator



The Sync Generator puts out a sync pulse which is used to synchronize the blocks in the design. See the CASPER memo on sync pulse generation for a detailed explanation and the iBOB iADC tutorial for an example on its basic use.

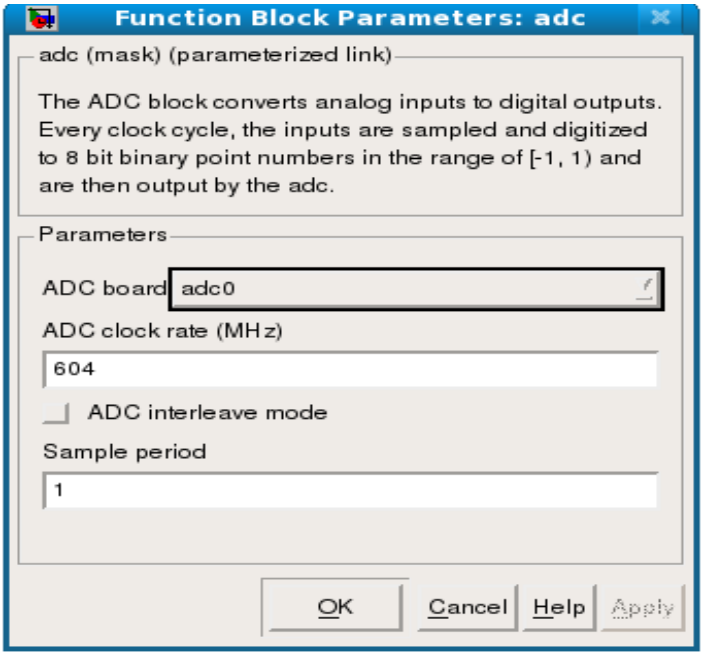
Whenever a sys_rst is given through the software register a mrst signal is generated which resets the entire MAC block. Sync pulse is generated with a period of 2^27 clock cycles and resets all the blocks.

5.3 ADC



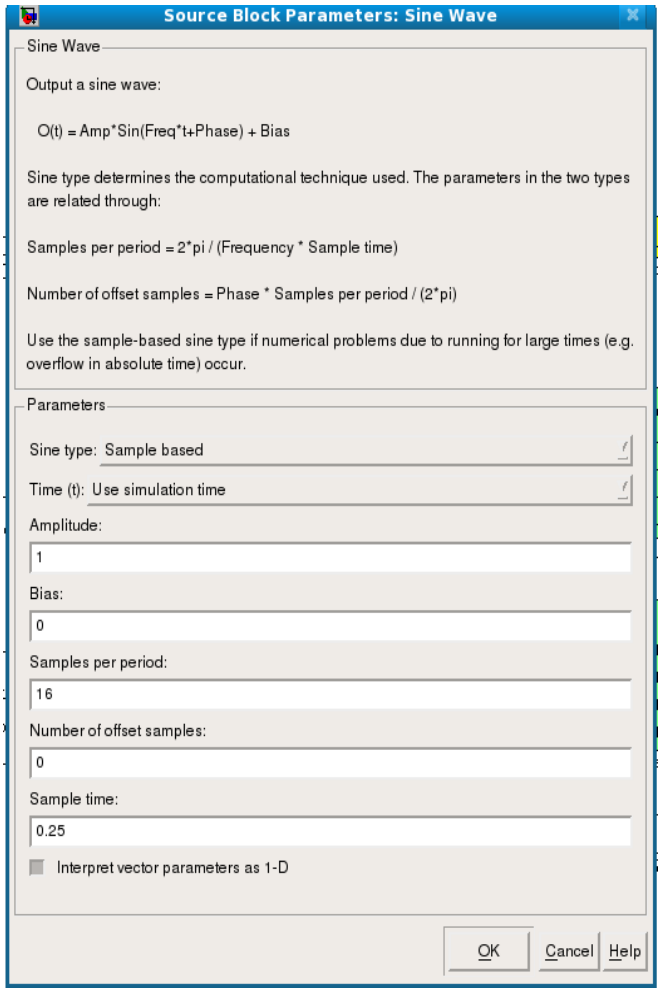
Connection of the ADC is as in Wideban spectrometer tutorial except for the sync outputs.

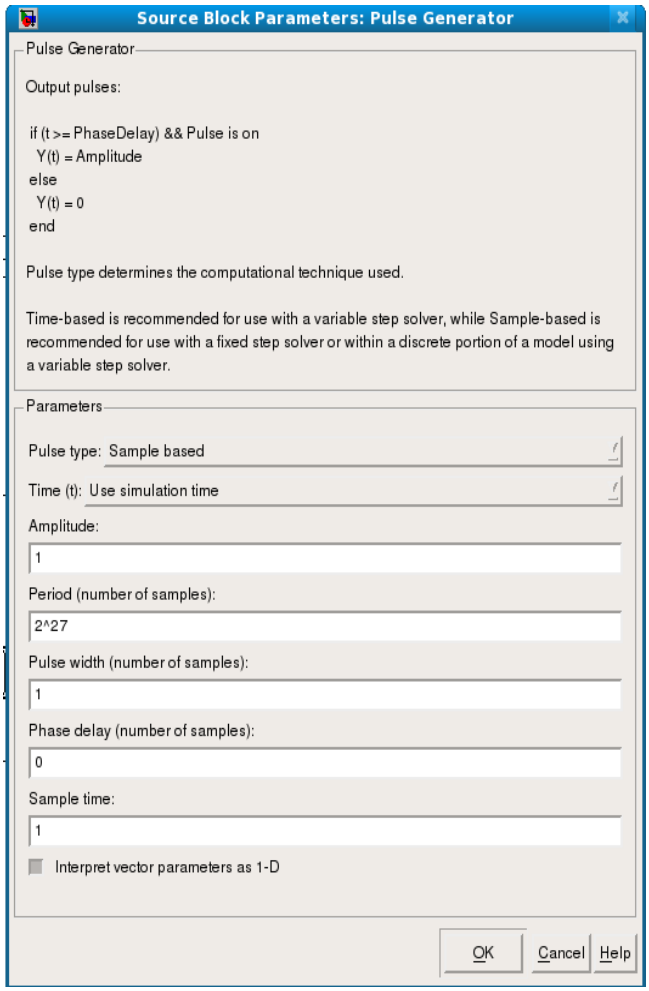
Set up the ADC as shown below



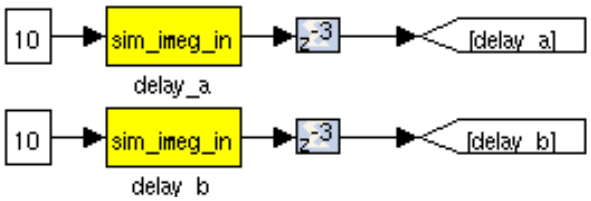
For the purposes of simulation (and to satisfy Simulink's requirements that all inputs be connected), we need to put input signals into the ADCs. These blocks are pulse generators in the case of sync inputs and any analogue source for the RF inputs (noise, CW tones etc).

The setup for these blocks is as shown below





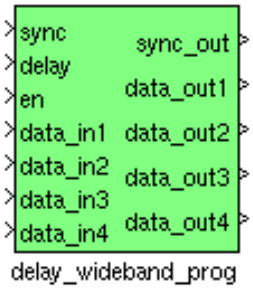
5.4 Software Registers



This part of the Simulink design sets up a software register which can be configured through the controlling software from the PC to control the correlator. Set the yellow **software register's** IO direction as **from processor.** You can find it in the BEE_XPS System blockset. The constant block input to this register is used only for simulation.

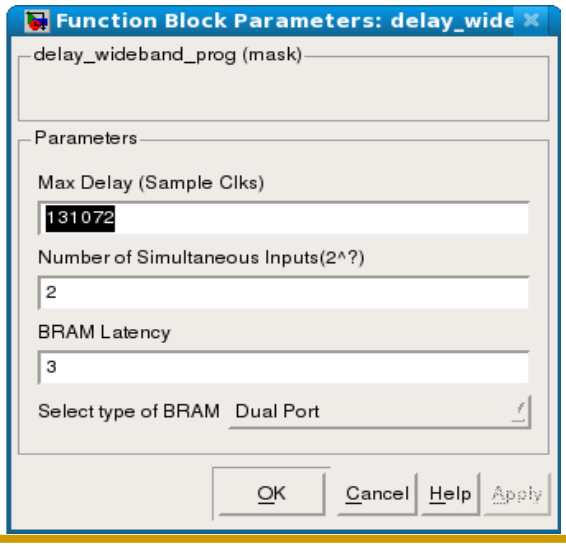
There are several such software registers in the design, a couple of those are shown in the above figure.

5.5 Coarse Delay Block



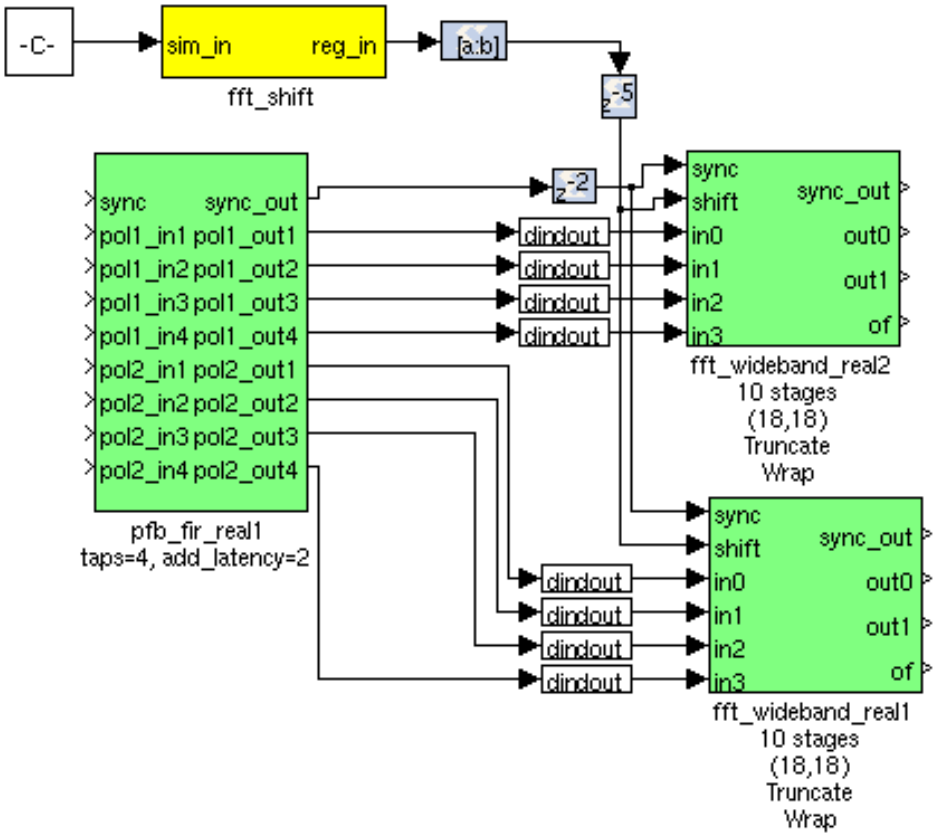
The coarse delay block can be found under Casper DSP Blockset → Delays. The delay block is used to delay the simultaneous data stream by specified number of clock cycles.

Configure the delay blocks as shown below:

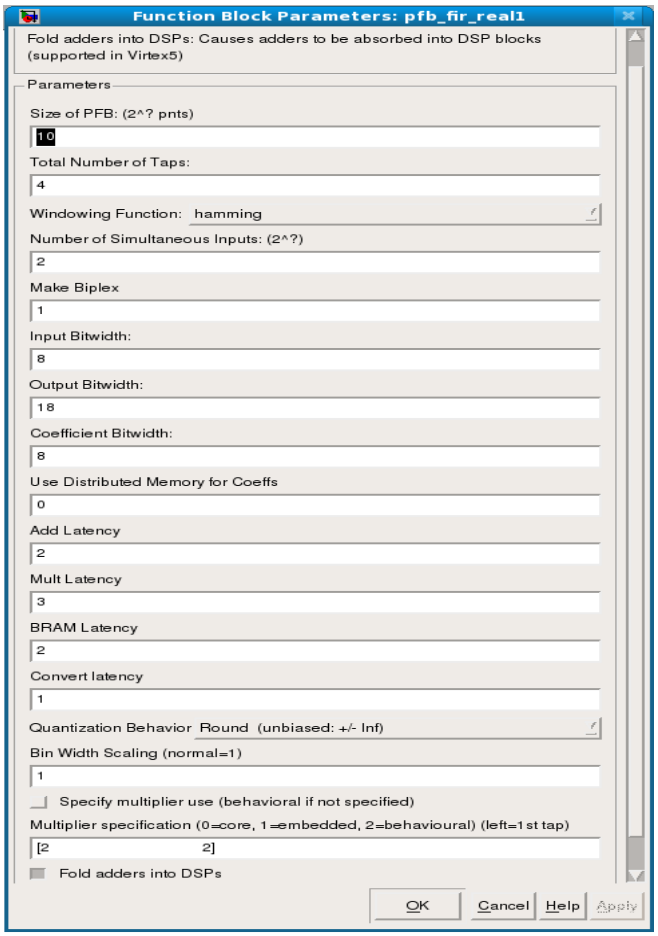


5.6 PFB and FFTs

The PFB FIR and FFT are the heart of this design. The PFB and the FFT are connected as shown below

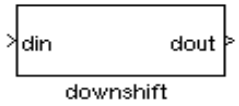


Configure the PFB_FIR_real blocks as shown below:



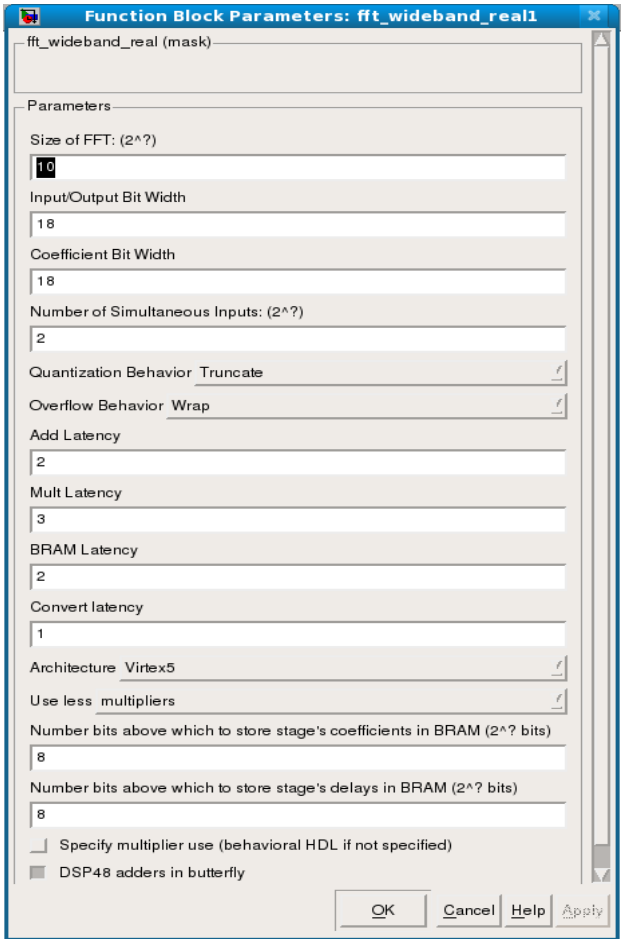
Downshifting block

There is potential to overflow the first FFT stage if the input is periodic or signal levels are high as shifting inside the FFT is only performed after each butterfly stage calculation. For this reason, we recommend casting any inputs up to 18 bits with the binary point at position 17 (thus keeping the range of values -1 to 1), and then downshifting by 1 bit to place the signal in one less than the most significant bits.

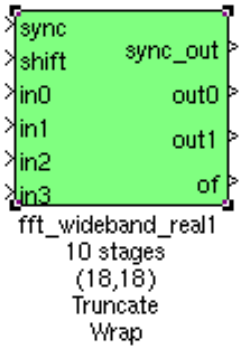


The downshifing block is a static block and hence is not present in the casper library. The block has to be copied from the **"[TUT4_MDL_DOWNSHIFT_FILE]"** mdl file present in the location **"[STD_MDL_DIR]"**.

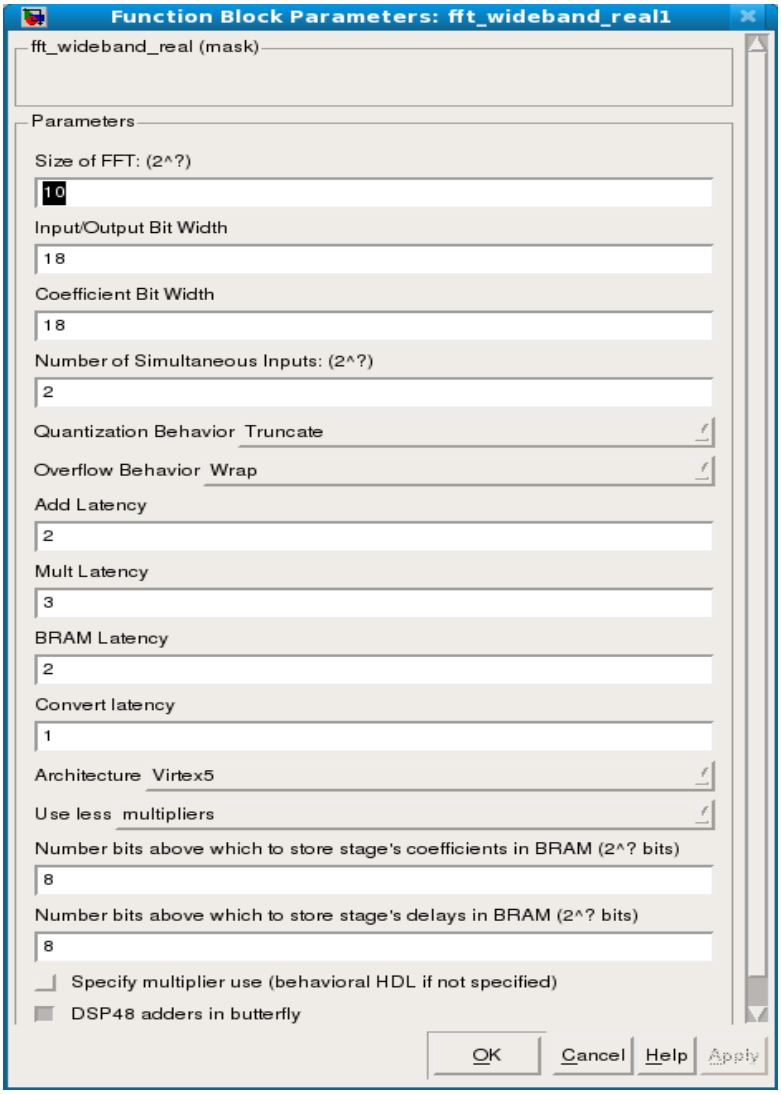
The fft_wide_band_real block should be configured as follows:



5.7 Fine Delay Block

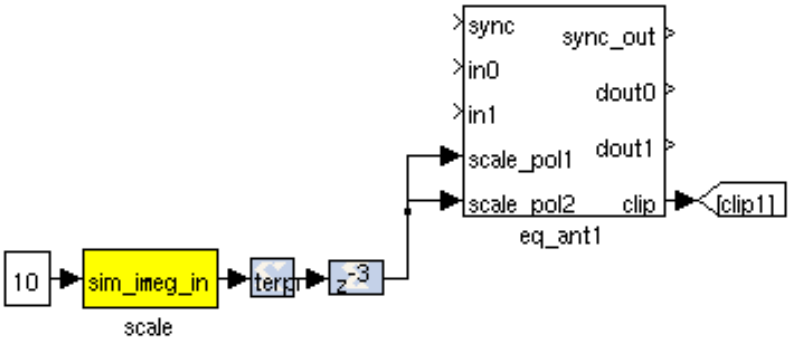


The fine delay block is parameterised as shown below



5.8 Equiliser

The equaliser block is set to 512 FFT channels and the Subsystem is designed as seen below. The equiliser reduces the bit growth that was introduced in the PFB and FFT. We can do this because we do not need the full dynamic range.



The block is a static block and hence is not present in the casper library. The block has to be copied from the **"[TUT4_MDL_EQ_FILE]"** mdl file present in the location **"[STD_MDL_DIR]"**.

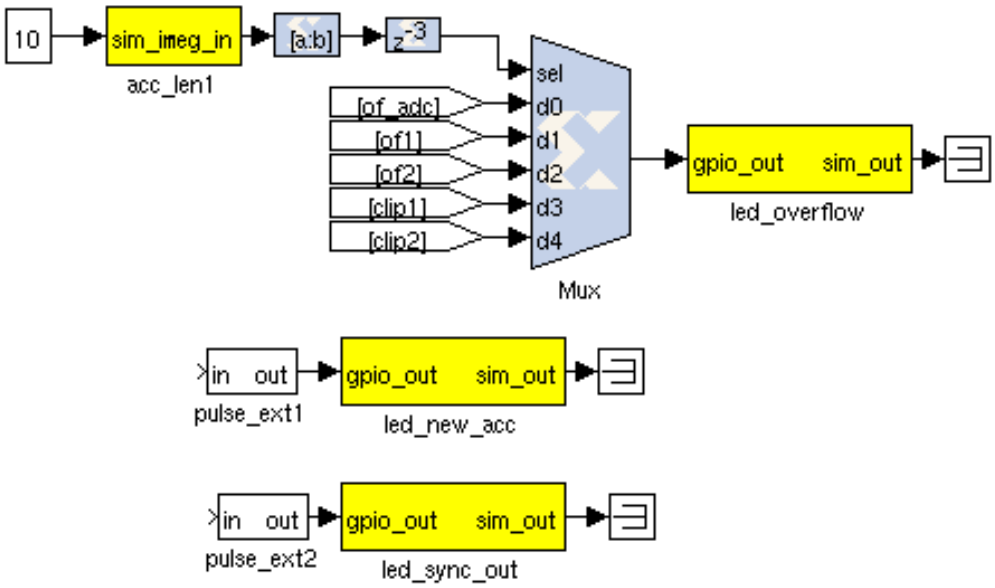
5.9 LEDs

The following sections are more periphery to the design and will only be touched on. By now you should be comfortable putting the blocks together and be able to figure out many of the values and parameters.

As a kind of debug output we can wire up the LEDs to certain signals. We light an LED with every sync pulse. This is a sort of heartbeat showing that the design is clocking and the FPGA is running.

The led "led_new_acc" gives a visual indication of when an accumulation is complete while the "led_overflow" led indicates any clipping encountered in ADC, FFT or Qunatiser stages.

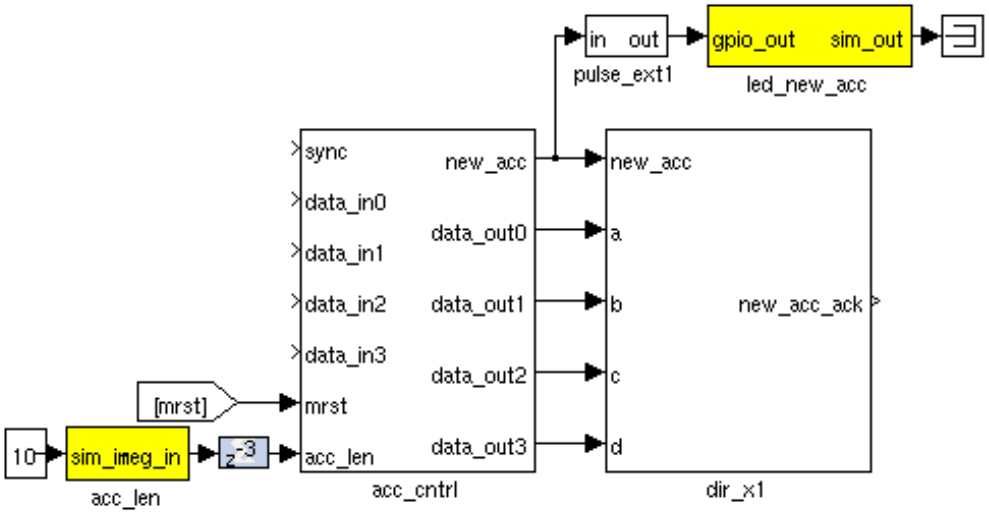
Since the signals might be too short to light up an LED and for us to actually see it (consider the case where a single ADC sample overflows; 1/800MHz is 1.25nS – much too short for the human eye to see) we add a negedge delay block which delays the negative edge of a block, thereby extending the positive pulse. A length of 2^23 gives about a 10ms pulse.



5.10 The MAC operation

The multiply and accumulate is performed in the dir_x (direct-x) blocks, so named because baselines are calculated directly, in parallel (as opposed to the packetised correlators' X engines which process serially).

Accumulation for each baseline takes place in BRAM using the same simple vector accumulator used in the 'wideband spectometer' tutorial.



The MAC block is a static block and hence is not present in the casper library. The block has to be copied from the **"[TUT4_MDL_MAC_FILE]"** mdl file present in the location **"[STD_MDL_DIR]"**.

CONTROL:

Sys_rst software register is for resetting the complete design. Sync LED provides a "heartbeat" signal to instantly see if your design is clocked sensibly. Similarly the coarse delay, fractional delay and fringe stop values can be provided runtime via script. New accumulation LED gives a visual indication of data rates and dump times. Also the Overflow LED is the indication of data overflow at any stage in the design flow.

6 Compilation

By giving bee_xps command in the matlab window , we will get a pop-up. Make sure the file displayed in the pop-up is correct and then press RUN to start the compilation. After compilation , it creates a directory named after the model file name without the .mdl extension. There is a sub directory named bit_files. In this bit_files directory there are .bit and .bof file. We need the .bof file to program the FPGA.
You need to save this .bof file at location [FPGA_PROG_BOF_DIR] .

7 Software

The python scripts are located in the **"[STD_PYSCRIPT_DIR]"** directory. We first need to run **"[TUT4_CONFIG_PYSCRIPT_FILE]"** to program the FPGA and configure the design. Then we can run the script **"[TUT4_PLOT_PYSCRIPT_FILE]"** to plot the self, cross and the phase.

Copy the bof file to be programed which is compiled by you , to the directory **"[FPGA_PROG_BOF_DIR]"** after changing the permissions of the file.

eg. for the bof file [TUT4_BOF_FILE] in the area [STD_BOF_DIR]
$ **chmod a+x** [STD_BOF_DIR]/[TUT4_BOF_FILE]
$ **cp** [STD_BOF_DIR]/[TUT4_BOF_FILE] [FPGA_PROG_BOF_DIR]

Usage:[STD_PYSCRIPT_DIR]/[TUT4_CONFIG_PYSCRIPT_FILE] **<ROACH name/IP> -b <bof file>**

eg.
$[STD_PYSCRIPT_DIR]/[TUT4_CONFIG_PYSCRIPT_FILE] **roach030172 -b** [TUT4_BOF_FILE]

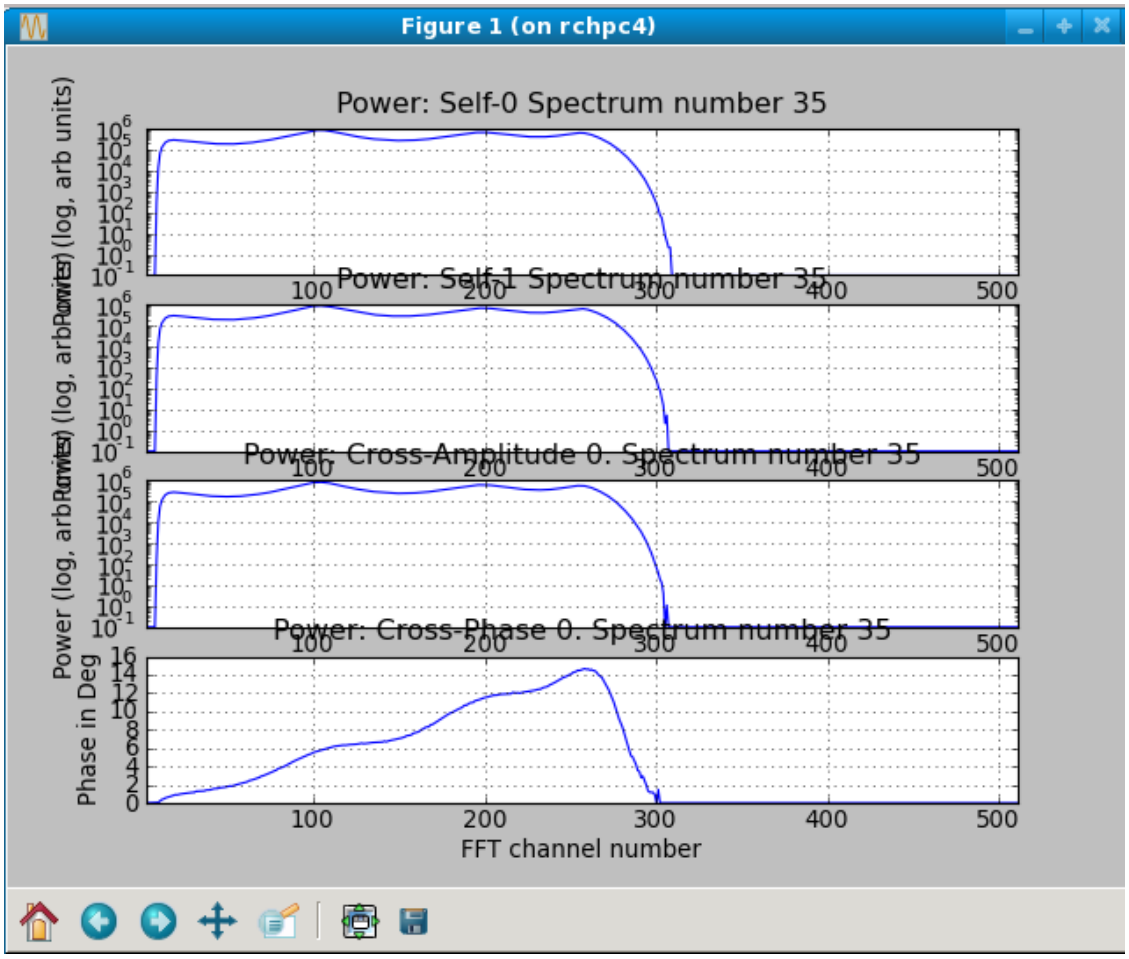
#Enter the corresponding Location/File names and roach name/IP.

Note : Enter your roach number! You will see on display;

$[STD_PYSCRIPT_DIR]/[TUT4_PLOT_PYSCRIPT_FILE] <ROACH name/IP> -l

#Enter the corresponding Location/File names and roach name/IP.

This script grabs auto-correlations, cross-correlation and the phase from the brams and plots them. The following plot gives the self spectrum, cross correlation and the phase between the two input signals. Option -l for logirithm scale.



8 Conclusion

Pocket correlator tutorial completed. We have observed the auto-correlations AA & BB for the two inputs we have fed to iADC and we have also observed the cross power and phase spectrum.