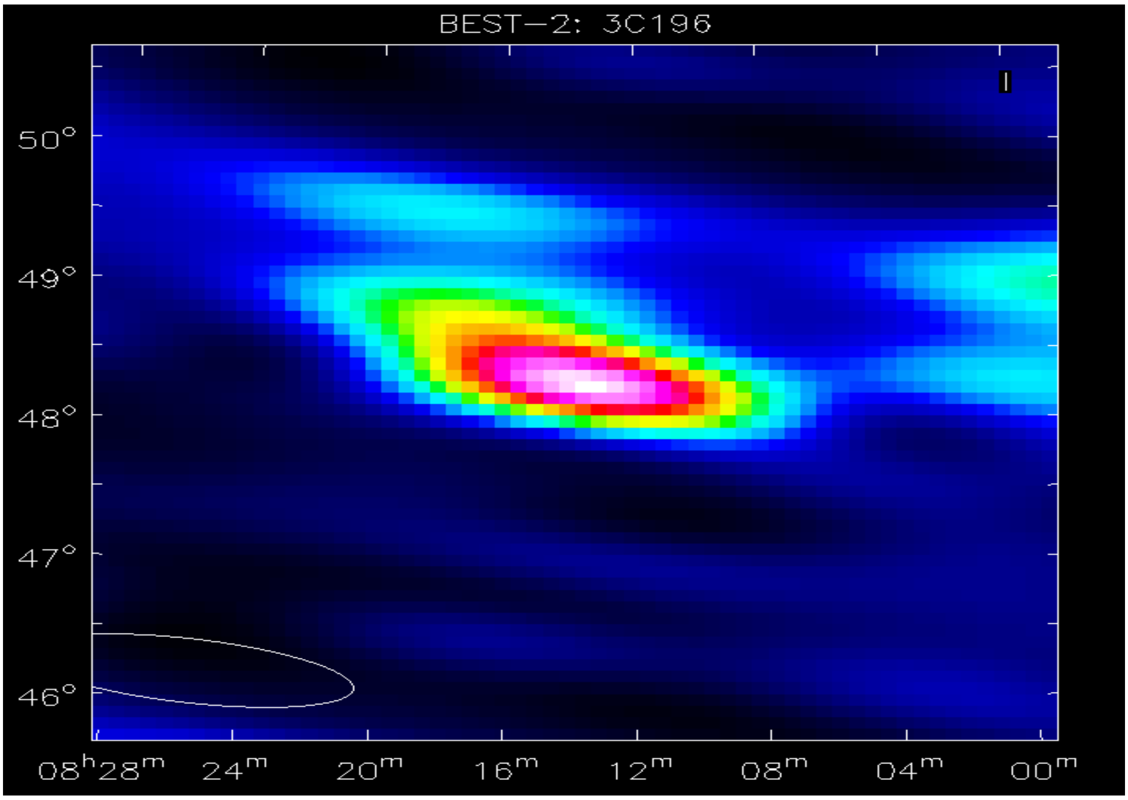
A Digital Backend Architecture for Fourier Imaging

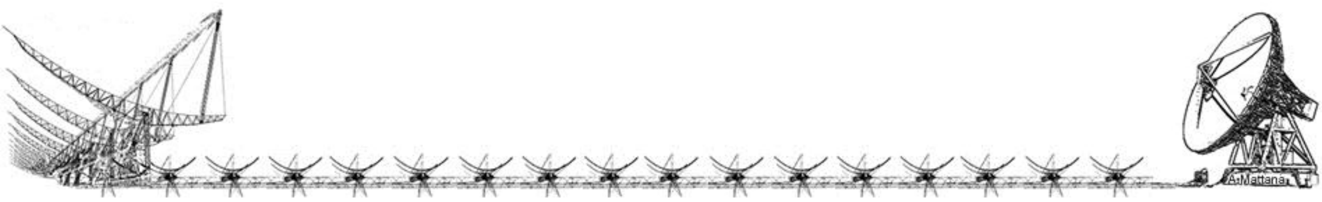
A. Mattana, M. Bartolini, G. Naldi

IRA 449/11



Referee: Andrea Orlati

Contents



Preface

This document wants to describe a possible FPGA architecture to develop a Fourier Imager using the Medicina Northern Cross Radio telescope. The test bed for this project is a portion of the North-South arm also knows as BEST-2 (Basic Element for *SKA* Training), it is a 8 cylinders having 4 RX each in single polarization.

The digital backend used is a CASPER (Parsons, A., et al., "Digital Instrumentation for the Radio Astronomy Community", astro-ph/0904.1181, April 2009) board based on XILINX FPGAs, the ROACH board.

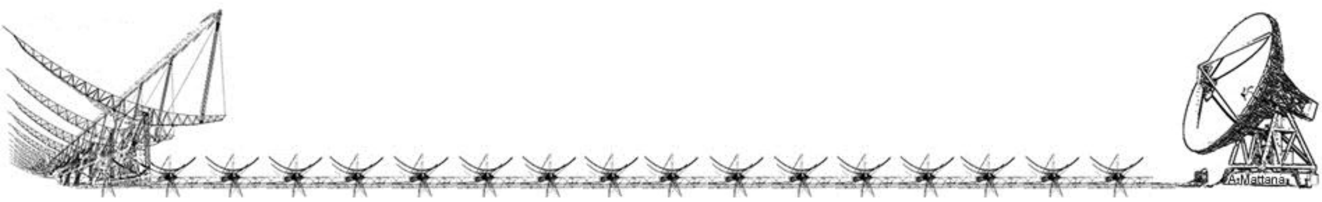
The development has been performed by using the Xilinx System Generator embedded in the Mathworks Matlab which allows to use a Xilinx Blockset plus custom radio astronomy libraries realized by the CASPER CONSORTIUM.

In order to simplify the reading of this book the description of the activity of each block has been rotated in a horizontal layout, the input signals come from left and the result signals leave to the right. Where no explicit, time goes from right to left.

This document describes only the project architecture and not the software running on the workstation or the radio astronomic post processing tools.

This project has been realized in collaboration with the Oxford University, and a special thanks goes to Kris Zarb-Adami, Jack Hickish, Griffin Foster, Danny Price, and the local team at IRA, Stelio Montebugnoli, Germano Bianchi and Marco Schiaffino.

The image in the cover is a result obtain after the post processing done by Griffin Foster.



Digital Backends

CASPER group

The term CASPER means "Collaboration for Astronomy Signal Processing and Electronics Research". The CASPER was born at the Berkeley University of California, with a collaborations of several institute and laboratories. The primary goal of CASPER is to streamline and simplify the design flow of radio astronomy instrumentation by promoting design reuse through the development of platform-independent, open-source hardware and software.

The CASPER group aim is to couple the real-time streaming performance of application-specific hardware with the design simplicity of general-purpose software. By providing parameterized, platform independent gateware libraries that run on reconfigurable, modular hardware building blocks, we abstract away low-level implementation details and allow astronomers to rapidly design and deploy new instruments.

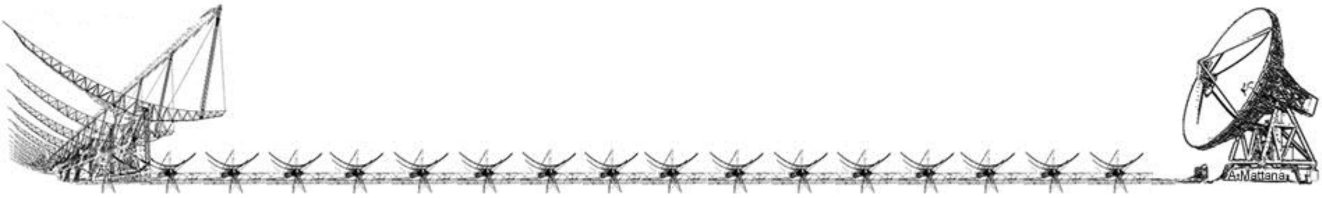
ROACH

Reconfigurable Open Architecture Computing Hardware is the last CASPER released board. The ROACH has in total 4 FPGA:

- a Xilinx VIRTEX 5 (package: XC5VSX95T-1FF1136) dedicated for the user as DSP
- an AMCC 440EPx Embedded Processor is a CPU 400-667MHz as a Linux Power PC
- an Actel AFS600 FPGA as a system supervisor
- a Xilinx XC2C256 CPLD as a JTAG programmer emulator

We have 5 ROACH boards at Medicina (up to now) and we need to use 3 of them for this project. The board comes with many interface peripherals such DDR2 RAM, corner turn memory, gpio pins and leds and high speed data transfer links such 10GbEth link, 10/100 Mb Ethernet link.

Only one ROACH board is dedicated to the data acquisition and the synchronization with the time and the regularity of the samples is guaranteed from a clock and a PPS signal both locked to the hydrogen maser atomic clock. The communication between the boards will be implemented



with a peer-to-peer XAUI protocol, while the data will be passed to the workstation encapsulated in UDP packets.

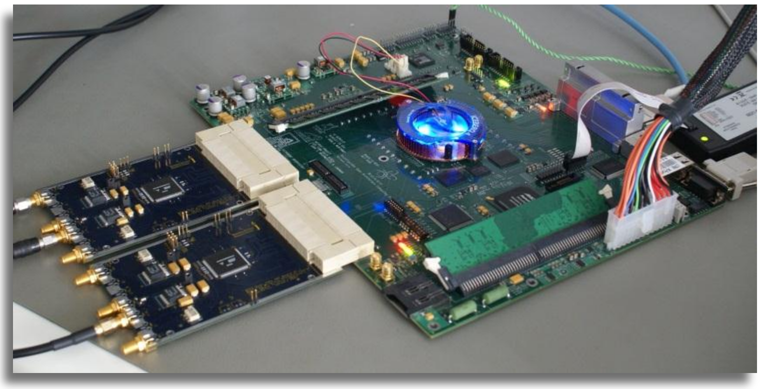


Figure 1: one of the ROACH boards at Medicina without the chassis.

Programming and Libraries

The System Generator is a DSP design tool from Xilinx that enables the use of The Mathworks model-based design environment Simulink for FPGA design. Over 90 DSP building blocks are provided in the Xilinx DSP blockset for Simulink. These blocks include the common DSP building blocks such as adders, multipliers and registers. Also included are a set of complex DSP building blocks such as forward error correction blocks, FFTs, filters and memories.

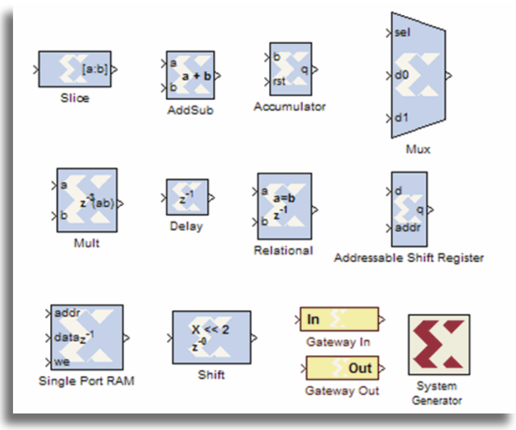


Figure 2: A view of some XILINX blocks used in Simulink

These blocks leverage the Xilinx IP core generators to deliver optimized results for the selected device.

The Embedded Development Kit (EDK) is a suite of tools and Intellectual Property (IP) that enables you to design a complete embedded processor system for implementation in a Xilinx FPGA device. Think of it as an umbrella covering all things related to embedded processor systems and their design. The Xilinx ISE software must also be installed to run EDK.

The CASPER group has realized an open source library set customized for the astronomers and optimized for the FPGAs mounted on their boards, and to easily use the integrated peripherals.

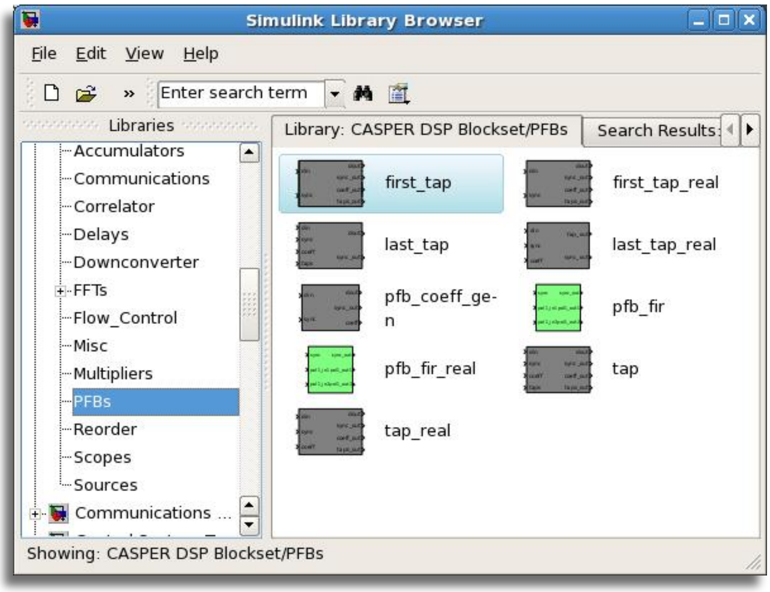


Figure 3: An example view of the CASPER DSP blockset to synthetize a polyphase filters bank

Programming with the system generator becomes easy with respect to use the VHDL language and it is very similar to use any electronic IDE tool, simulation included. This is the reason because this document will be rotated horizontally, in a vertical style you cannot represent in a good shape the layout of circuits, the view in horizontal is much better also to represent a signal in the time in a graph.

Introducing the project

The antenna acquisition system used for this project is the BEST-2 demonstrator. It is a part of the north-south arm of the Medicina Northern Cross Antenna composed of 8 cylinders having 4 receivers (RX from now on) each. Therefore there are 32 analogic signals to be sample and processed. All the Northern Cross does not have a dual polarization. All the RF BEST-2 signals are transmitted to the receiver control room via optical link preserving any electrical properties.

Figure 4: BEST-2 section of the Medicina Northern Cross Radiotelescope, courtesy of Marco Schiaffino

The Analog to Digital converter (AD) installed in the ROACH board computes 64 input x 12 bit and 40Mbps, and we use only 32 input leaving spare the others. This is a custom AD, the library to drive the acquisition in Simulink has been written by the Oxford team that has modified also the polyphase filter bank block to manage the interleaved data flow. The project has been divided in 3 ROACH boards for capability reasons:

- F-engine that acquire the IF signals, filters the input signals in sub-bands and applies equalizations and quantizations
- X-engine that works as a correlator for the array calibration
- S-engine that compute the 2D FFT for the imaging

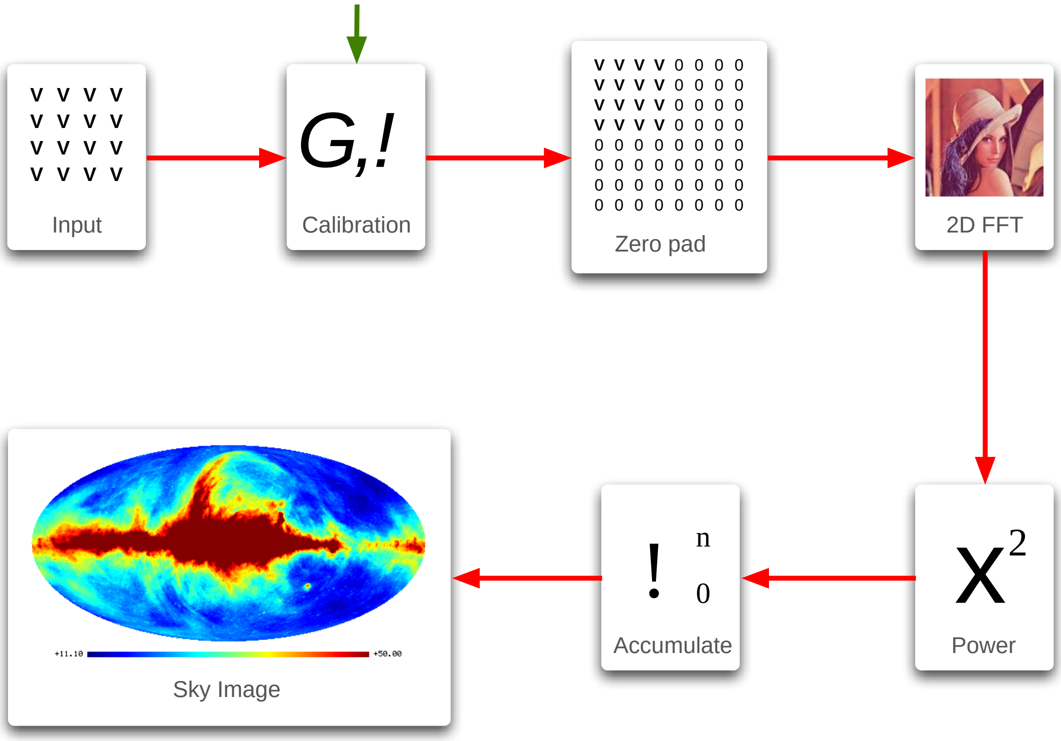
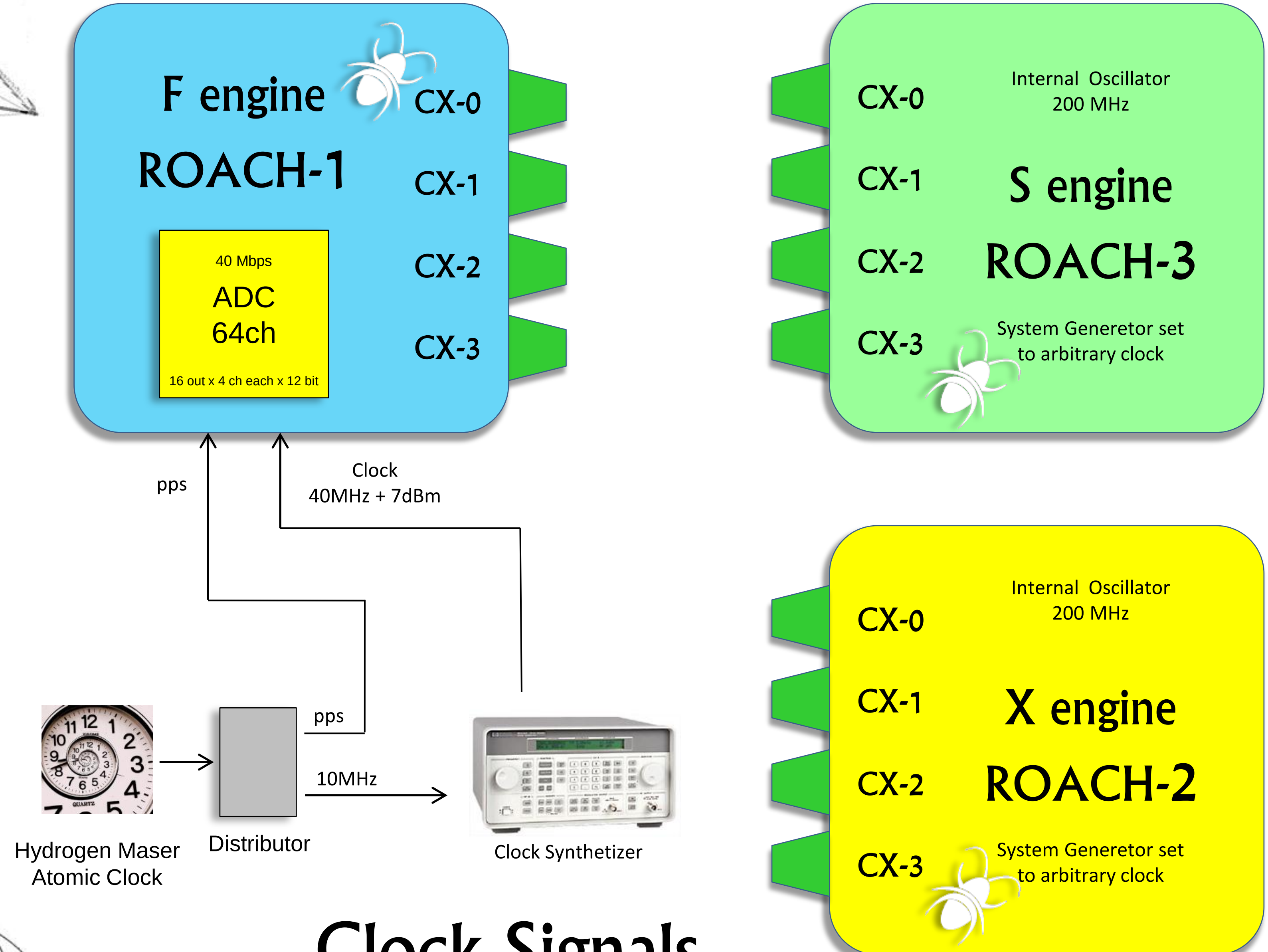
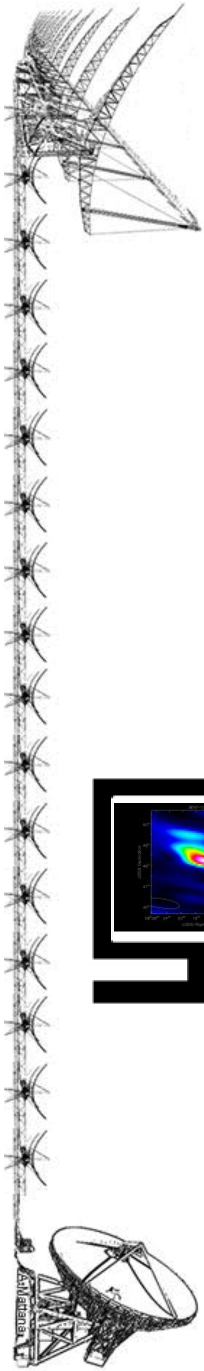


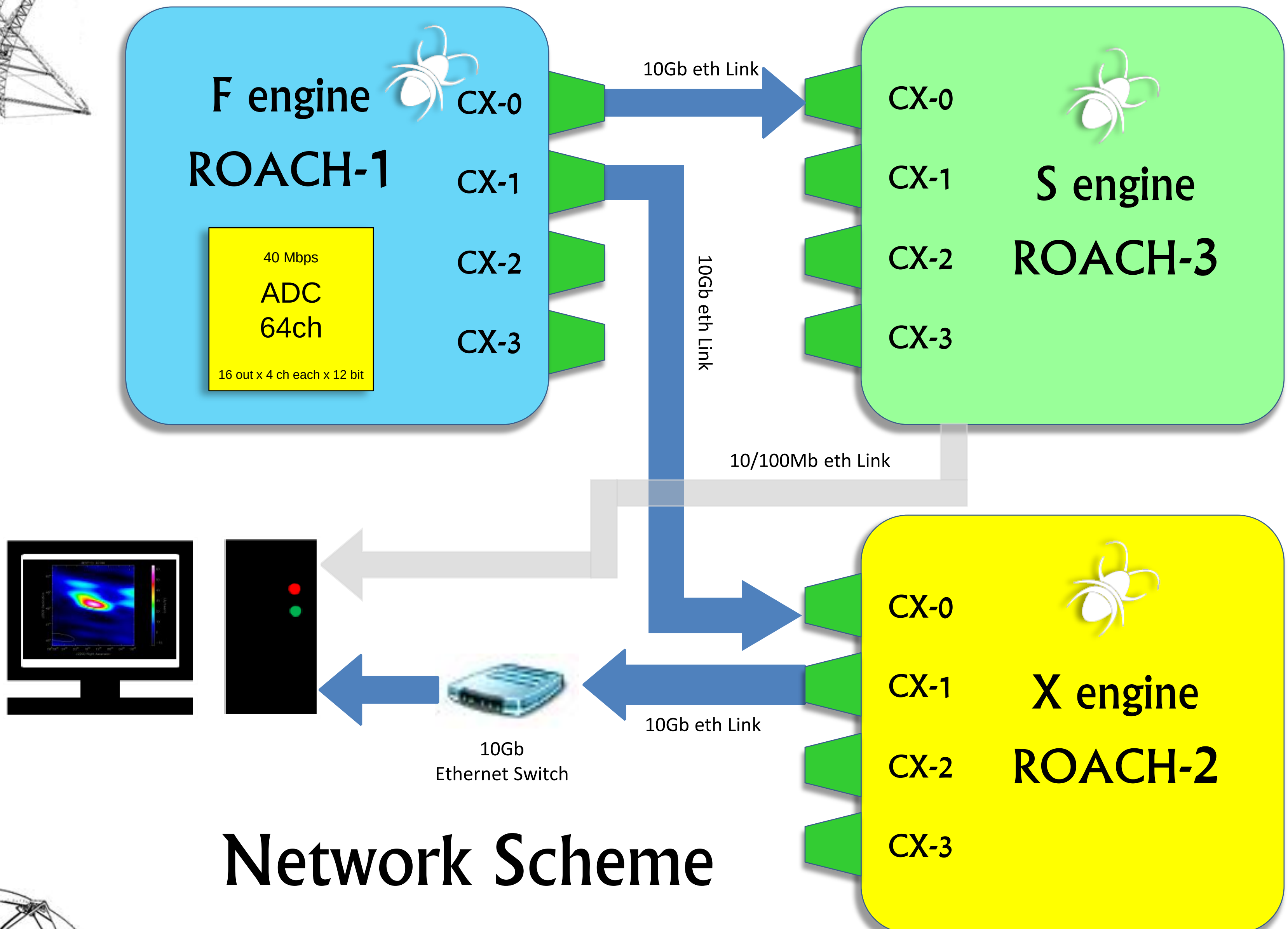
Figure 5: Conceptual project scheme, courtesy of Jack Hickish (Oxford University)

As described in the picture above, the analogic signals coming from the antenna are digitized, then calibrated, a 2D FFT is applied taking into account the BEST-2 single polarization (the second pol is zero padded), and finally there is an accumulation of the power spectra. To obtain the image the analysis needs to be completed with astronomical tools like the NRAO CASA.

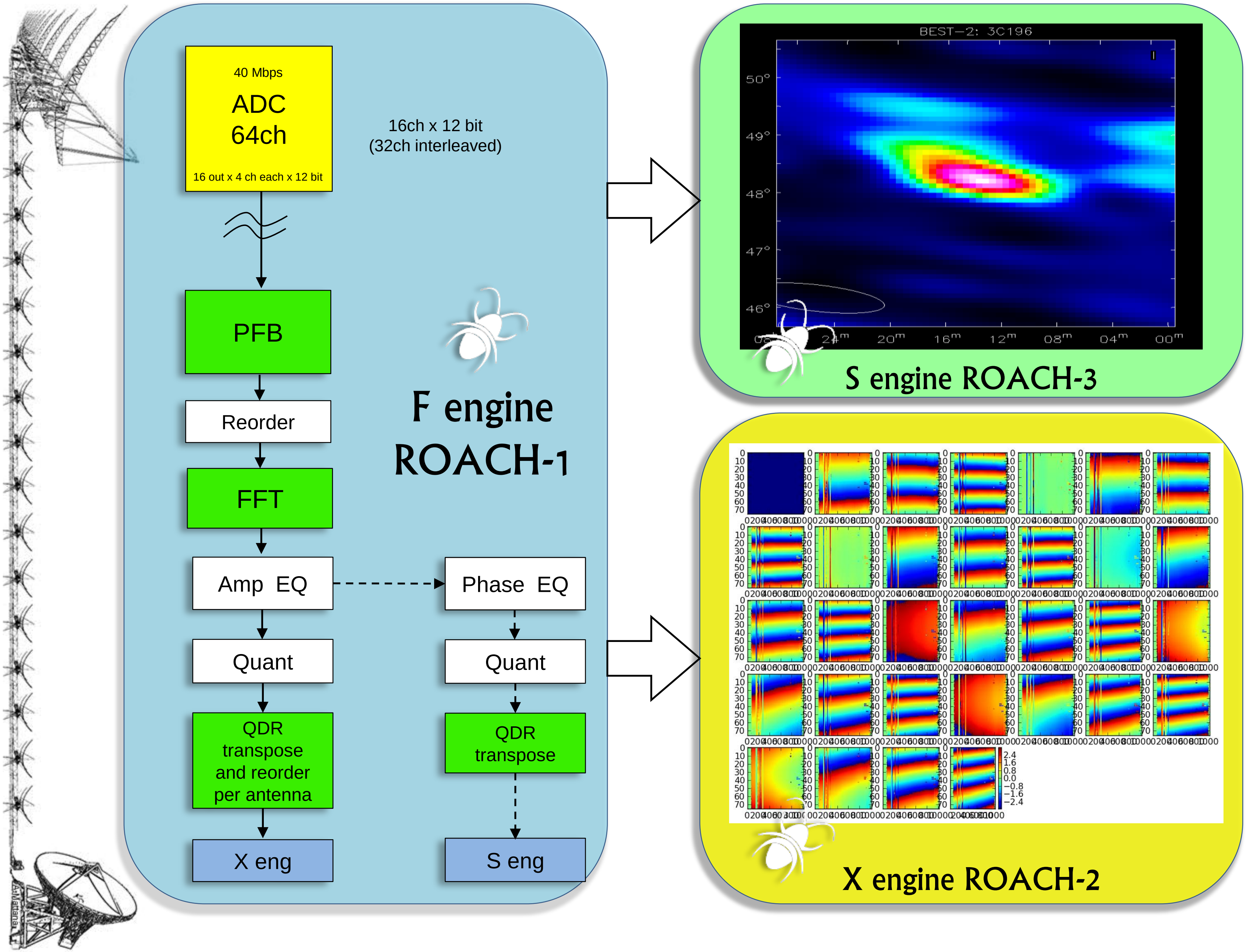


Clock Signals



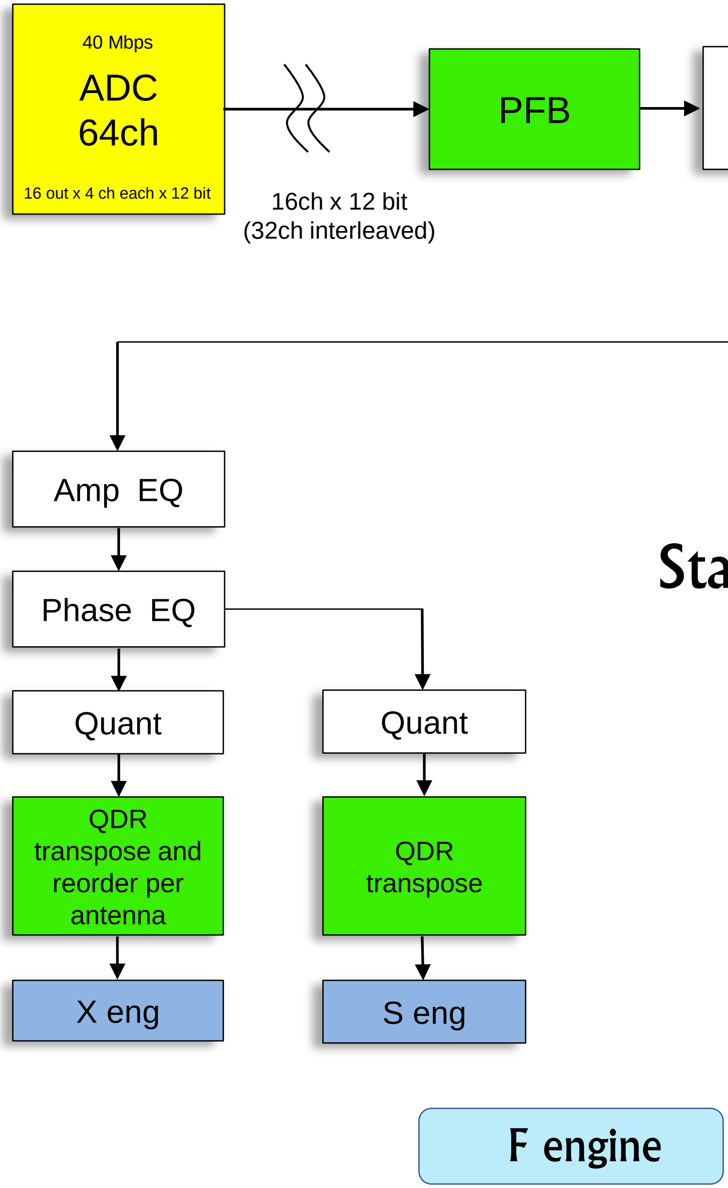


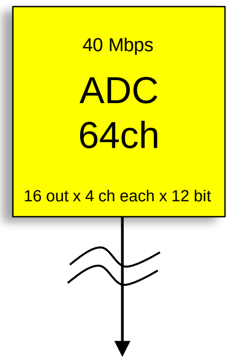
Network Scheme


40 Mbps
ADC
64ch
16 out x 4 ch each x 12 bit
16ch x 12 bit
(32ch interleaved)
PFB
Reorder
FFT
Amp EQ
Phase EQ
Quant
Quant
QDR
transpose
and reorder
per antenna
QDR
transpose
X eng
S eng
F engine
ROACH-1
BEST-2: 3C196
S engine ROACH-3
X engine ROACH-2

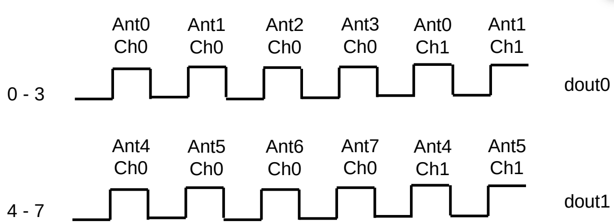
Starting from the F-engine

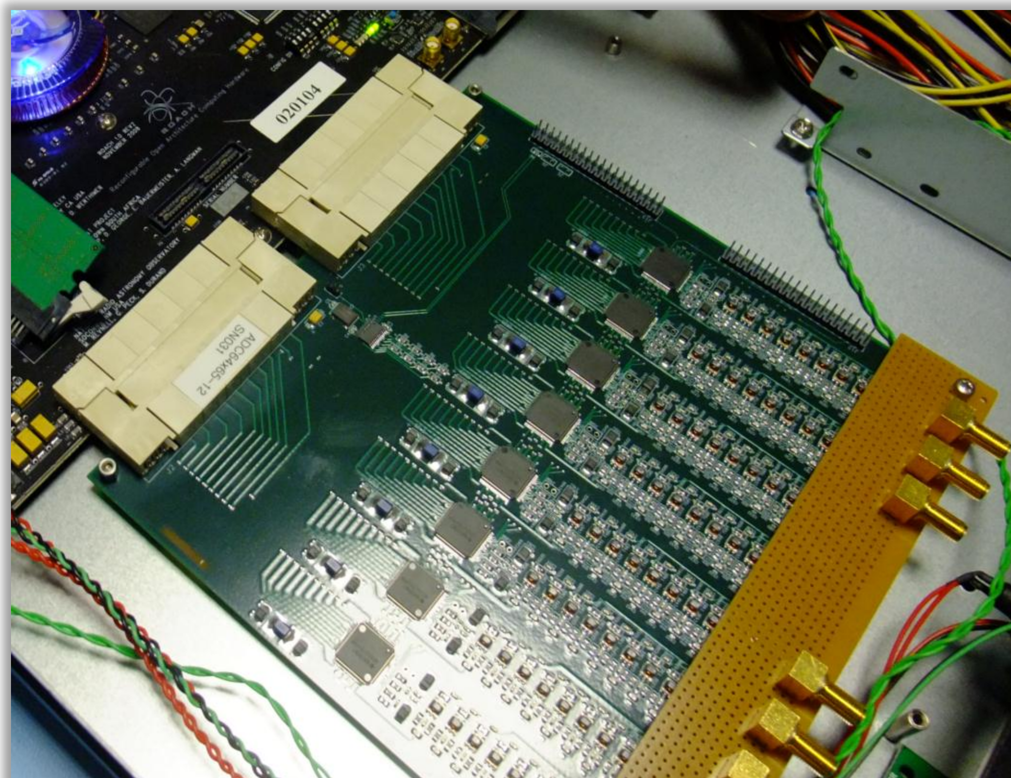
Roach-1





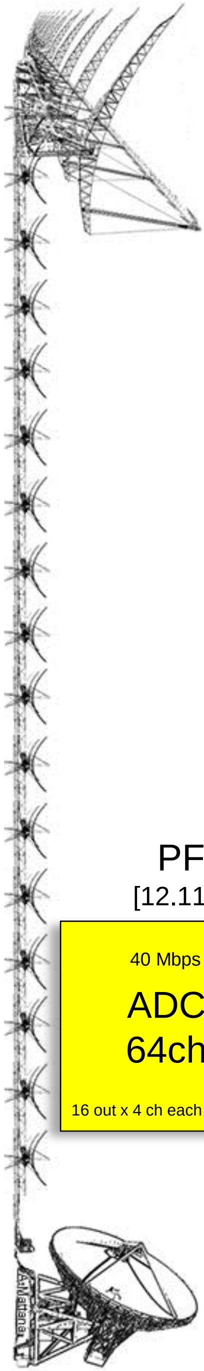
16ch x 12 bit
(32ch interleaved)

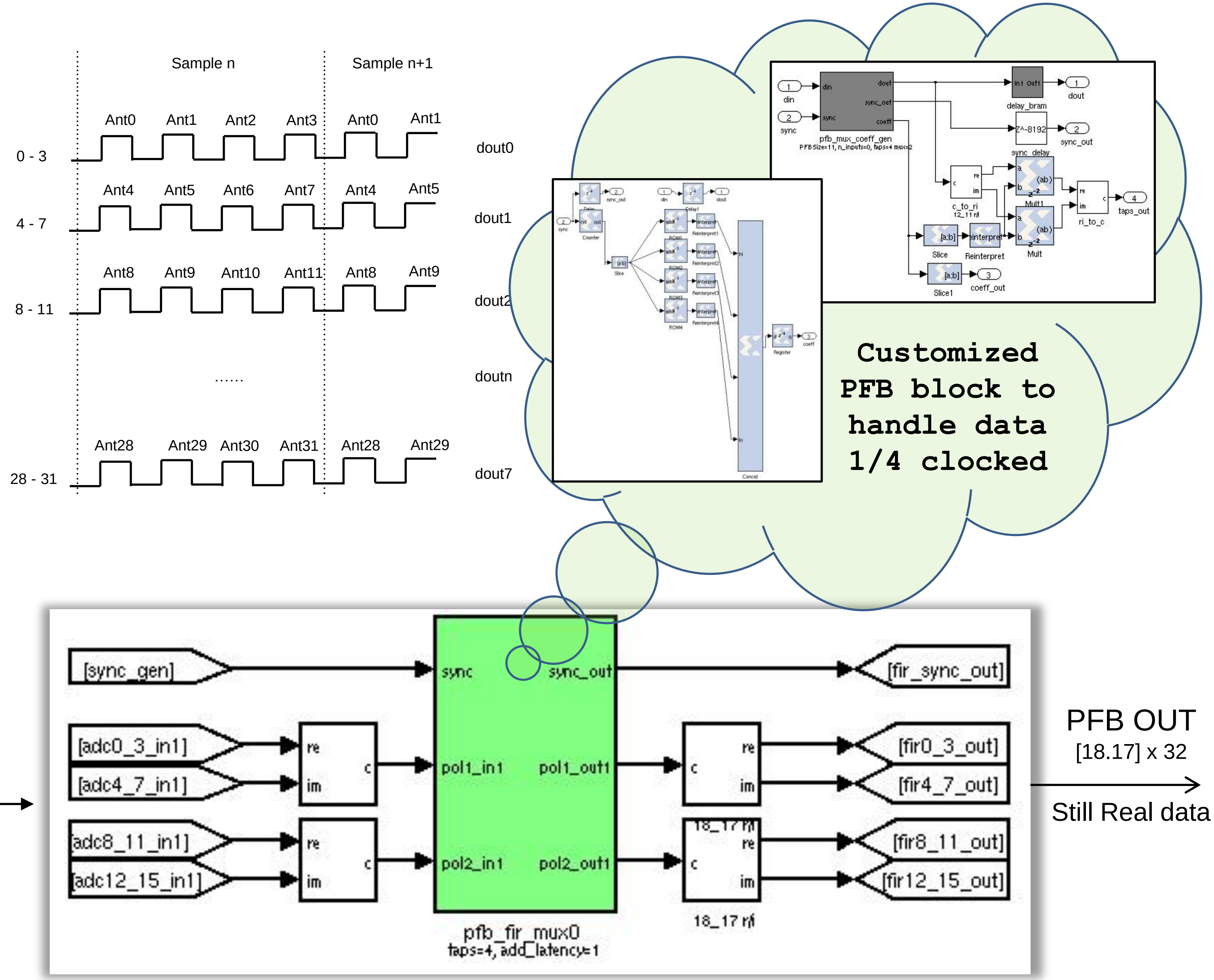




The 64 input AD takes both the ZDOKs connector on the ROACH board, there is a supply connector to allow to plug 64 SMA connectors

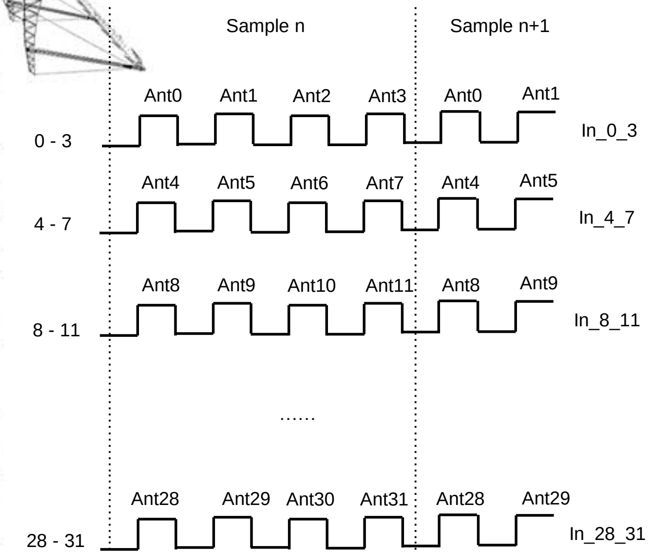
F engine

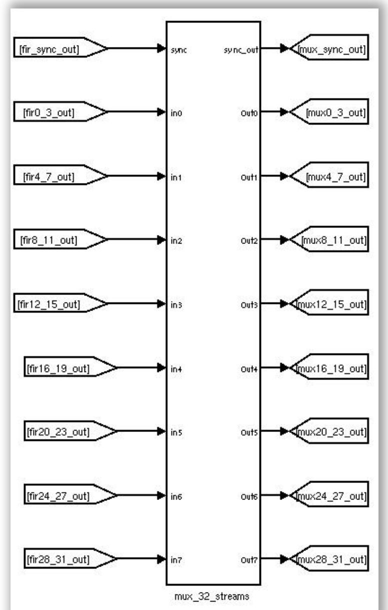


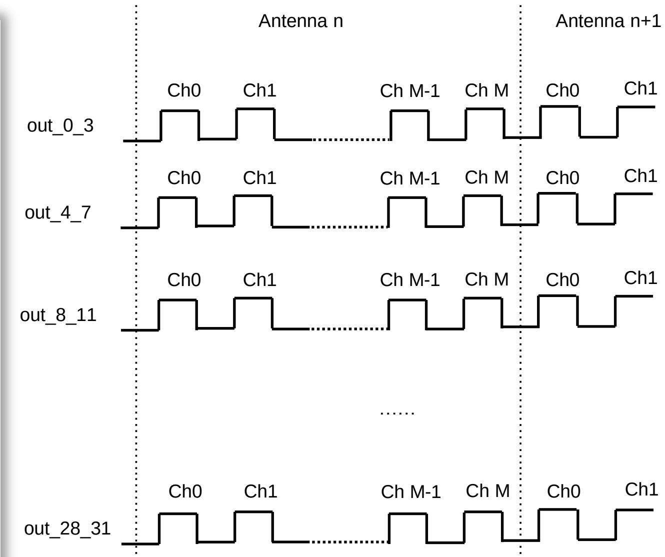


The customized polyphase filter bank

Reorder







Where $M = 2^{n\_fft} - 1$

n_fft = 11

Reorder Init

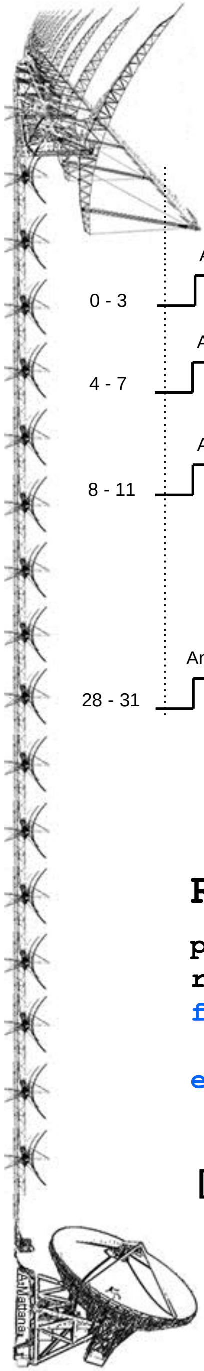
```
part_reorder = [0:4:2^(n_fft+2)-1];
reorder = [];
for multiplex_index = [0:3]
    reorder = [reorder,[part_reorder]+multiplex_index];
end
```

Example using n_fft = 3:

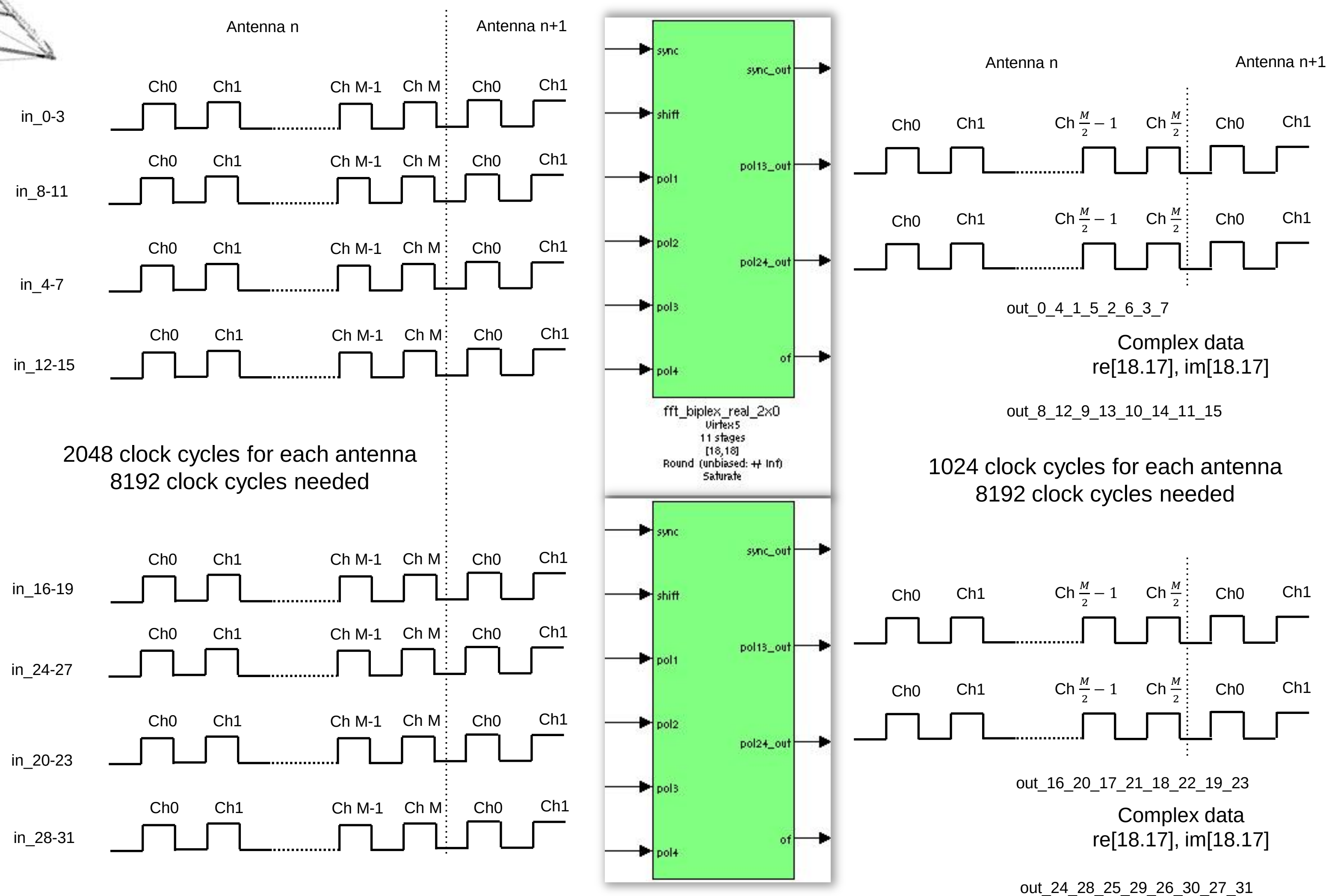
0	4	8	12	16	20	24	28
1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31

Data is reordered so that an entire window can be shifted through the FFT

F engine



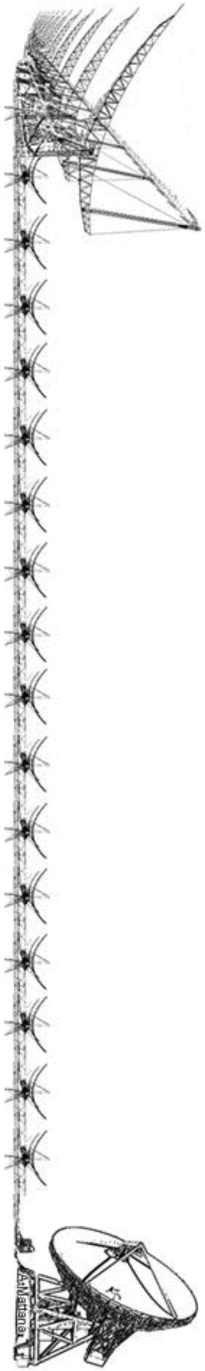
A real-sampled biplex FFT, with output demuxed by 2



F engine

n_fft = 11

$M = 2^{n\_fft} - 1$

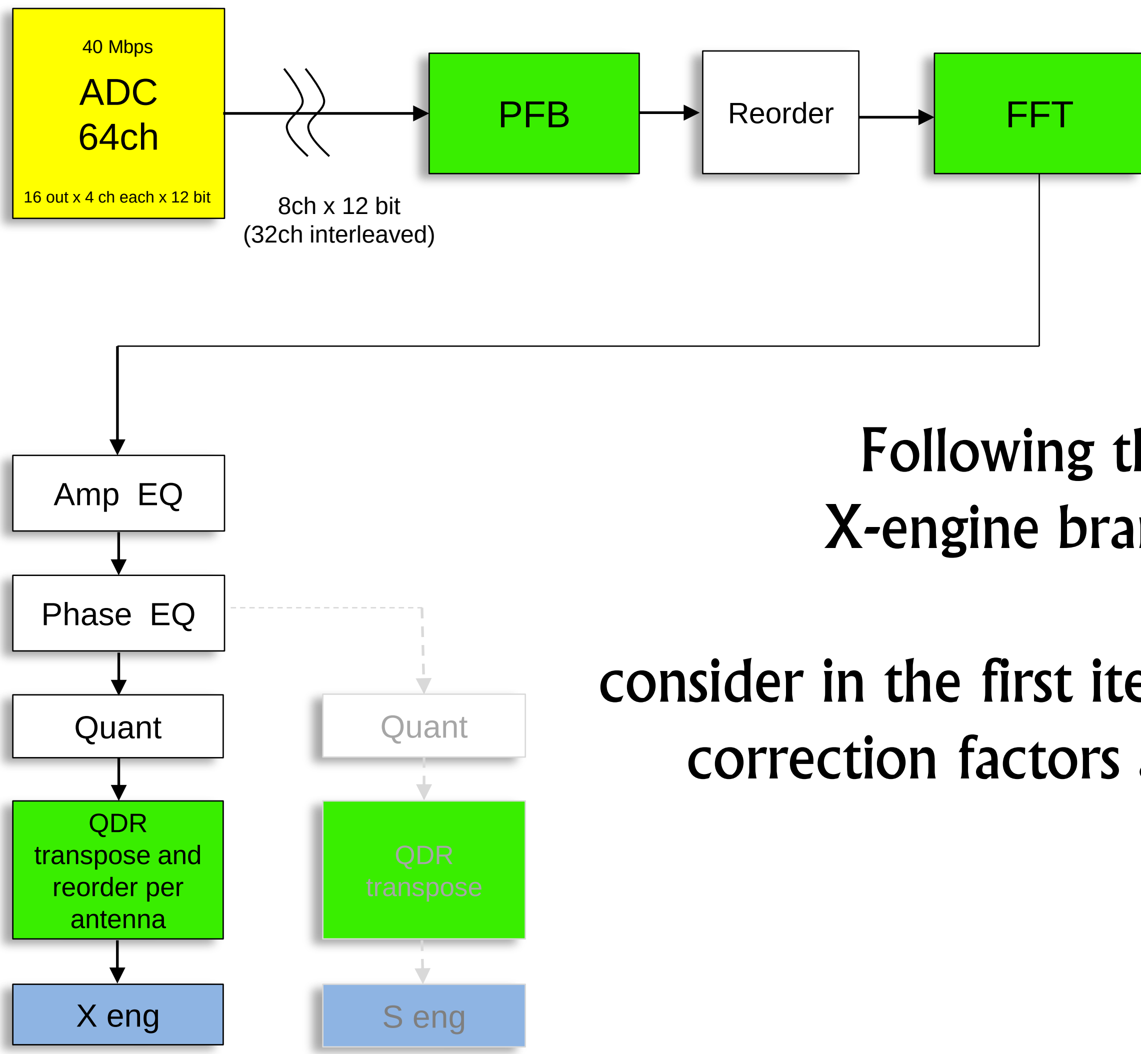


In order to make more readable this document we assume from now on this new antennas numbering which we will taking into account at the and

FFT real output sequence	New numbering	Prefix
0-4-1-5-2-6-3-7	0-1-2-3-5-6-7	0-7
8_12_9_13_10_14_11_15	8-9-10-11-12-13-14-15	8-15
16_20_17_21_18_22_19_23	16-17-18-19-20-21-22-23	16-23
24_28_25_29_26_30_27_31	24-25-26-27-28-29-30-31	24-31

!! IMPORTANT !!

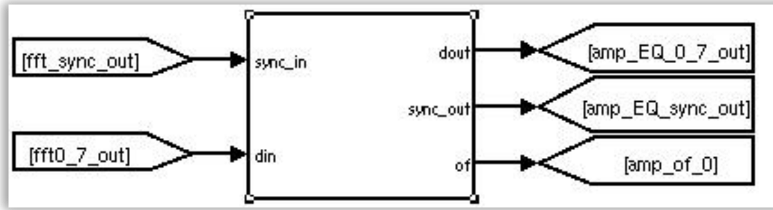
F engine



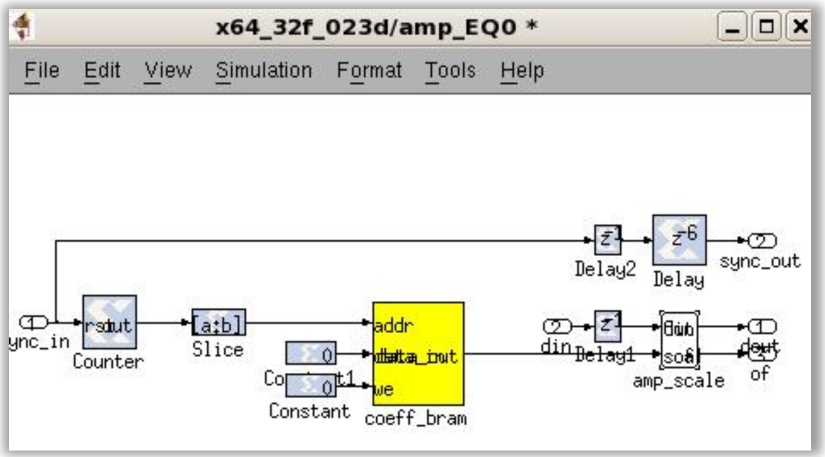
Following the X-engine branch

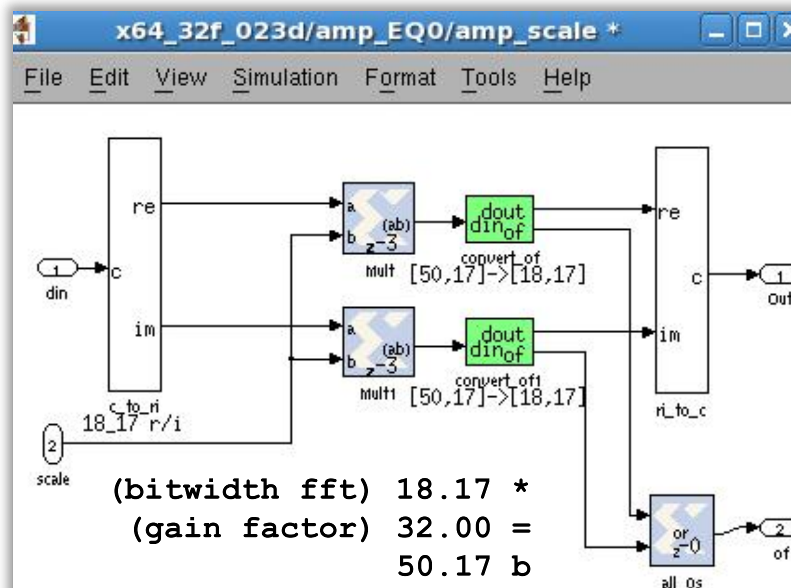
consider in the first iteration phase correction factors all zeroes

Equalise amplitude and pass data to be quantized to 4 bits and sent to the X engine over XAUI 18 bit precision is maintained so that phase corrections can be added to the data stream going to the fft imaging system.



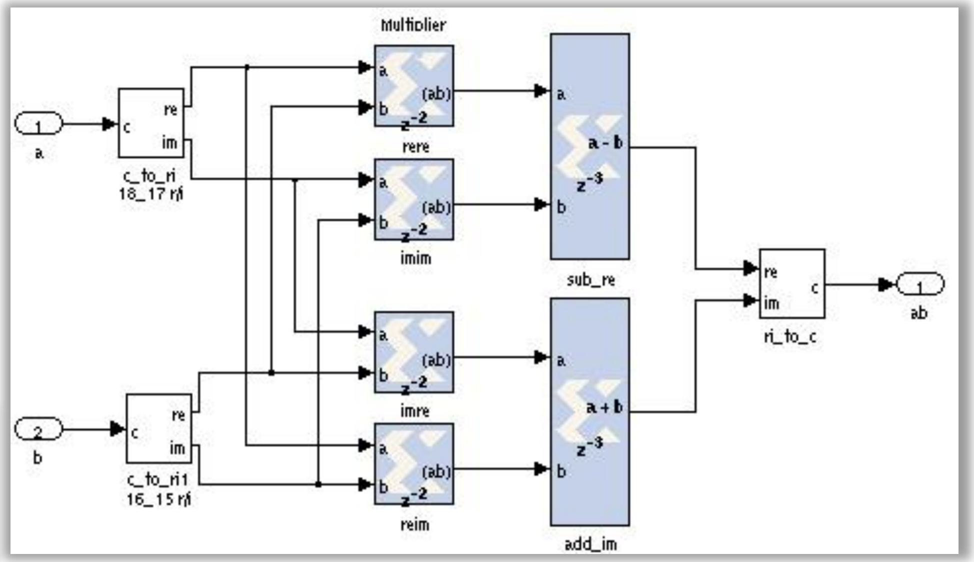
Amp EQ0



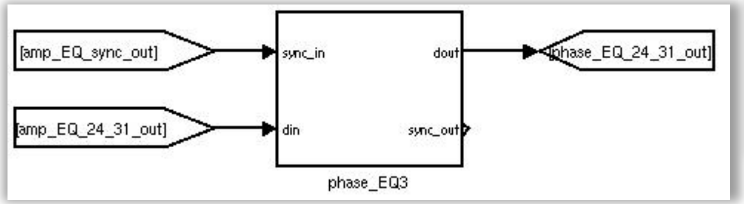


F engine

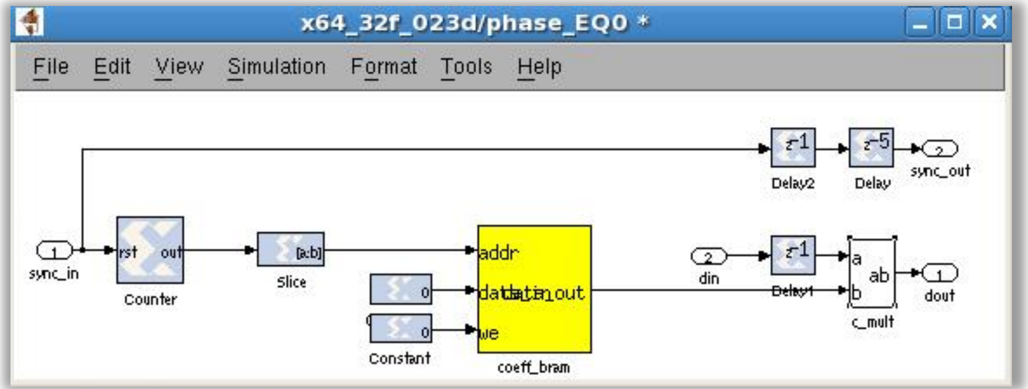
Phase corrections preparations



c_mult details



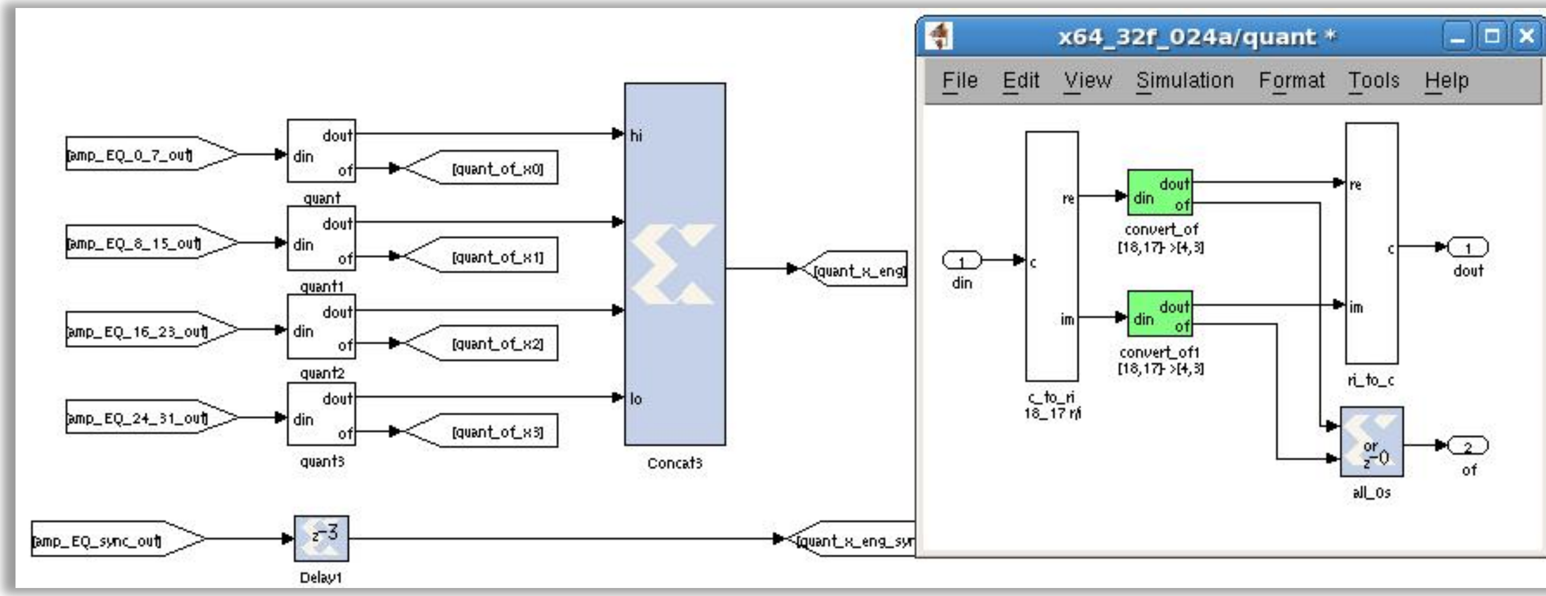
At the beginning all coefficients are set to zero and data has to send to the x-engine




```
(bitwidth fft) 18.17 *
(phase correction) 16.15 =
                   35.32 b
```

F engine

Quantization



From 18.17 (x2) to 4.3 (x2), quant_x_eng 32 bit

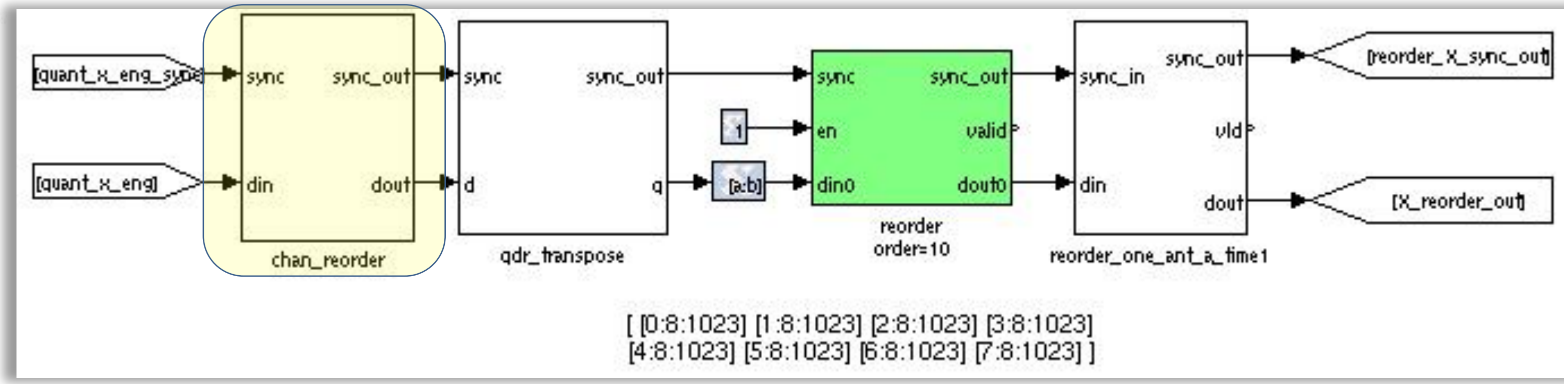
quant_x_eng:

a0c0, a0c1, …, a0c1023, a1c0, a1c1, …, a7c1023
a8c0, a8c1, …, a8c1023, a9c0, a9c1, …, a15c1023
a16c0, a16c1, …, a16c1023, a17c0, a17c1, …, a23c1023
a24c0, a24c1, …, a24c1023, a25c0, a25c1, …, a31c1023

t_0 t_1 $t_{\frac{M}{2}}$ $t_{\frac{M}{2}+1}$ $t_{\frac{M}{2}+2}$ $t_{\left(\frac{M}{2}\right)*8}$

F engine

X engine expects groups of a single channel and a single antenna



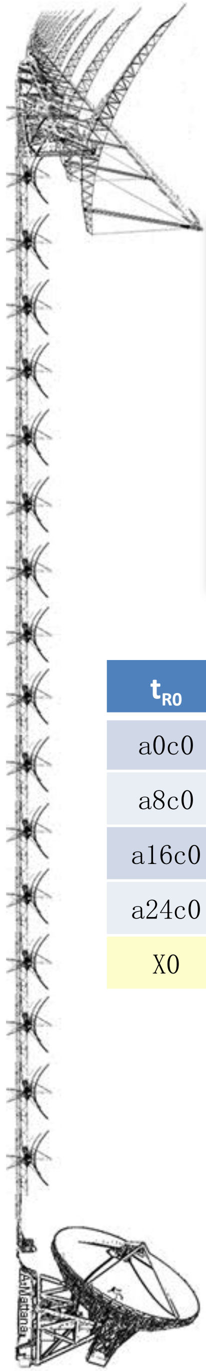

```
Chan_reorder init

pr = [0 512];
n_chan = 1024;

map = []
for i=[0:(n_chan/2)-1]
    map=[map, pr+i];
end

(result map:

0 512 1 513 2 514 3 515...
...510 1022 511 1023)
```

t_0	t_1	t_{end}
a0c0	a0c1	a7c1023
a8c0	a8c1	a15c1023
a16c0	a16c1	a23c1023
a24c0	a24c1	a31c1023

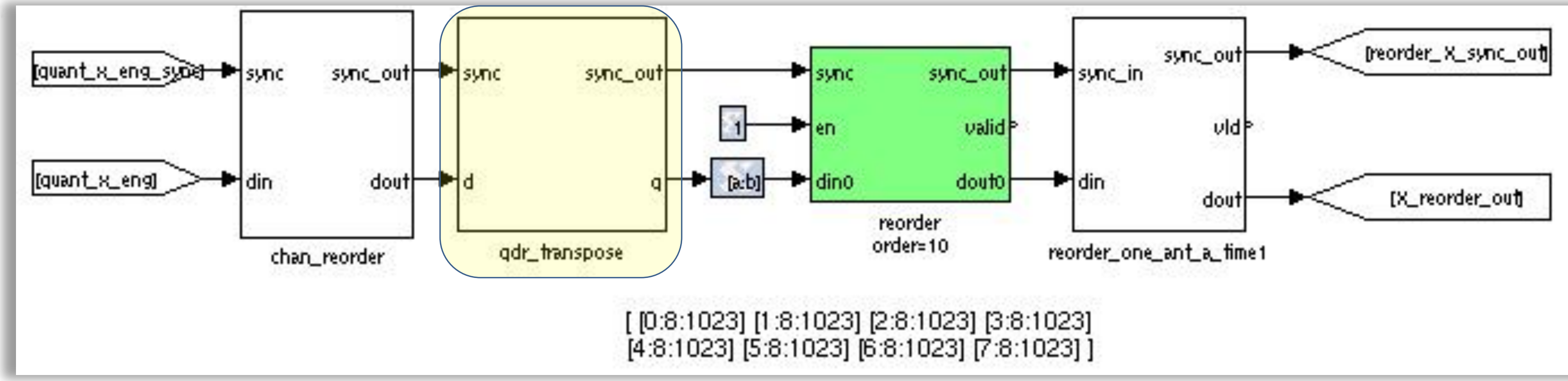
⇨

t_{R0}	t_{R1}	t_{Rend}
a0c0	a0c512	a7c1023
a8c0	a8c512	a15c1023
a16c0	a16c512	a23c1023
a24c0	a24c512	a31c1023
X0	X1	X8191

F engine



X engine expects groups of a single channel and a single antenna



t_{R0}	t_{R1}	t_{R1024}	t_{Rend}
a0c0	a0c512	a1c0	a7c1023
a8c0	a8c512	a9c0	a15c1023
a16c0	a16c512	a17c0	a23c1023
a24c0	a24c512	a25c0	a31c1023
X0	X1	X1024	X8191

Using QDR1 as 2^{10} x 2^{10} matrix

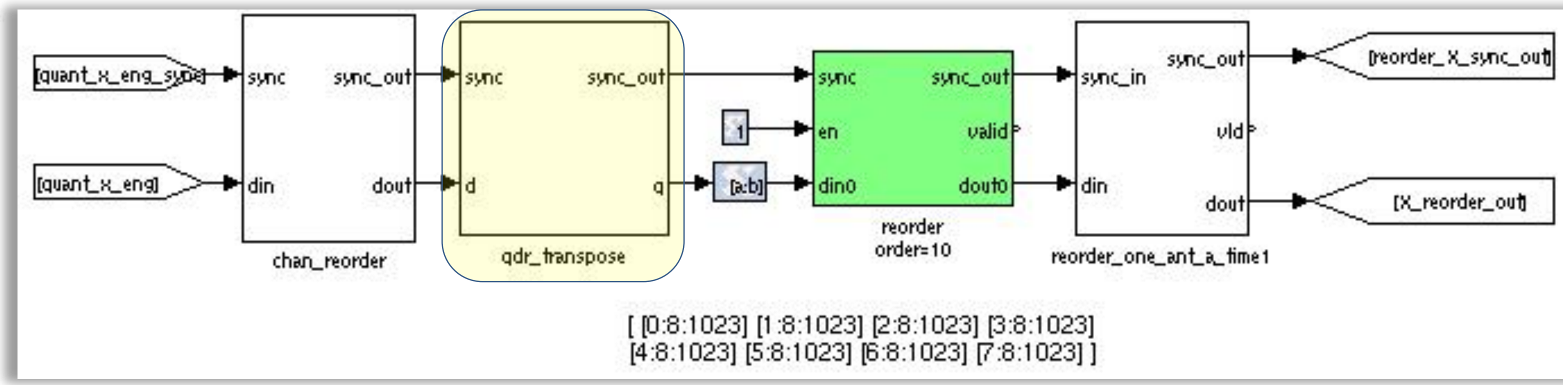
M =

	C_0	C_1	C_{1022}	C_{1023}
R_0	X0	X1	X1022	X1023
R_1	X2^10	X1025	X(2^10)*2-2	X(2^10)*2-1
R_2	X(2^10)*2	X(2^10)*2+1	X(2^10)*3-2	X(2^10)*3-1
R_{1022}	X(2^10)*1022	X(2^10)*1022+1	X(2^10)*1023-2	X(2^10)*1023-1
R_{1023}	X(2^10)*1023	X(2^10)*1023+1	X(2^10)*1024-2	X(2^10)*1024-1

1024*1024 cells means also 128 spectra of each antenna (8192 needed for one spectra)

F engine

X engine expects groups of a single channel and a single antenna

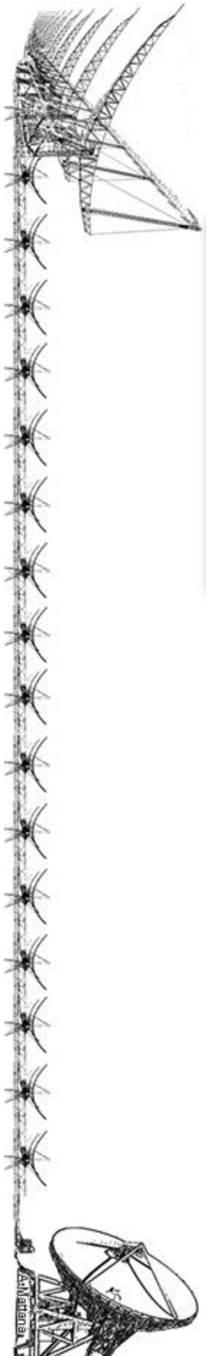


M =

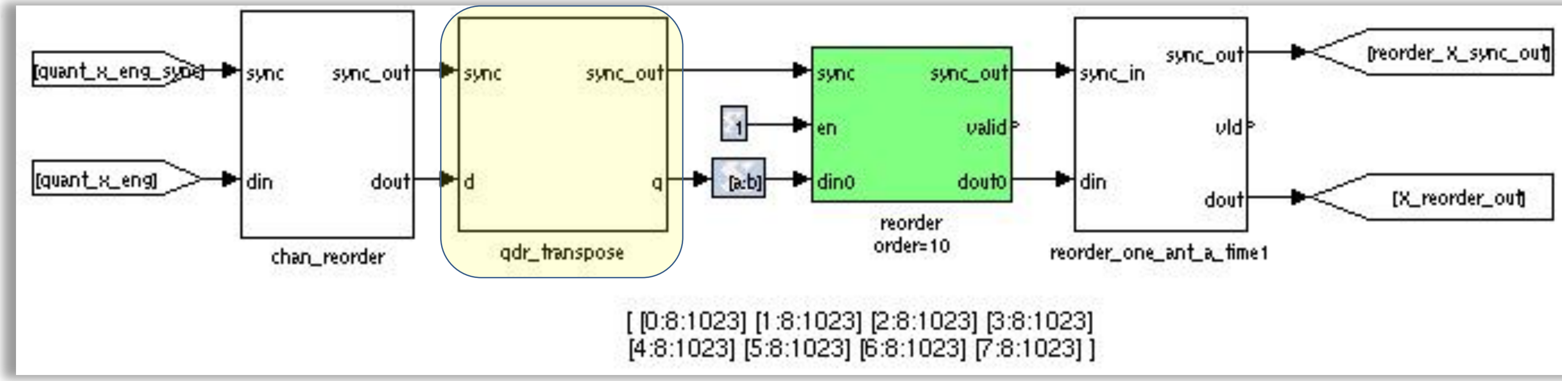
	C_0	C_1	C_{1022}	C_{1023}
R_{0-7}	Spectrum1	of 32 antennas	in 8 row	(8x1024)
R_{8-15}	Spectrum2	of 32 antennas	in 8 row	(8x1024)
R_{16-32}	Spectrum3	of 32 antennas	in 8 row	(8x1024)
...				
$R_{1016-1023}$	Spectrum128	of 32 antennas	in 8 row	(8x1024)

1024*1024 cells means also 128 spectra of each antenna (8192 needed for one spectra)

F engine



X engine expects groups of a single channel and a single antenna



t

x15	x14	x13	x12	x11	x10	x9	x8	x7	x6	x5	x4	x3	x2	x1	x0		x0	x1	x2	x3
																	x4	x5	x6	x7
																	x8	x9	x10	x11
																	x12	x13	x14	x15

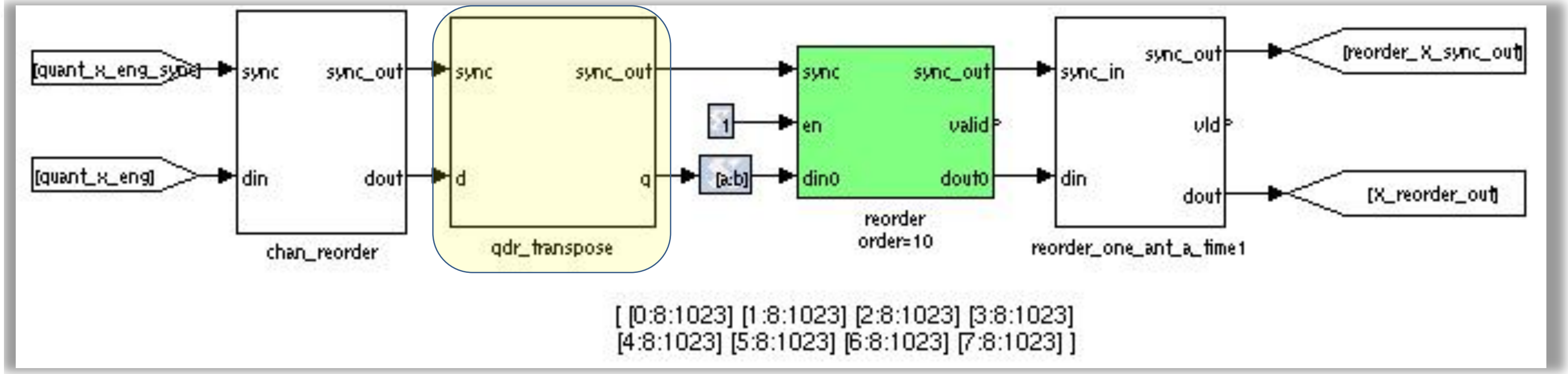
Considering time moving from left to right, the QDR transpose data as shown below

x0	x4	x8	x12																	
x1	x5	x9	x13																	
x2	x6	x10	x14																	
x3	x7	x11	x15		x15	x11	x7	x3	x14	x10	x6	x2	x13	x9	x5	x1	x12	x8	x4	x0

t

F engine

X engine expects groups of a single channel and a single antenna

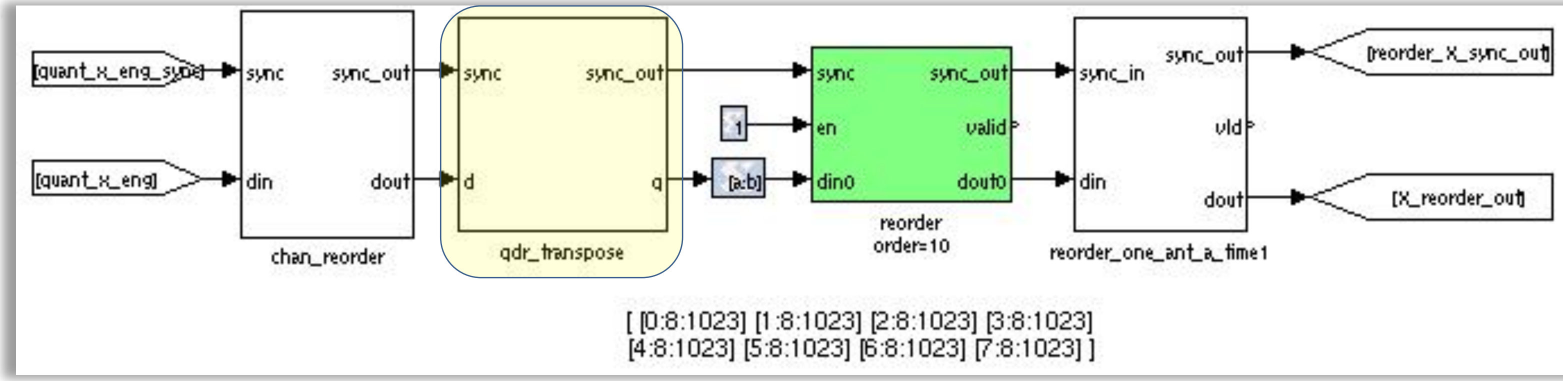


$M^T =$

	C_0	C_1	C_{1022}	C_{1023}
R_0	X0	X2^10	X(2^10)*1022	X(2^10)*1023
R_1	X1	X2^10+1	X(2^10)*1022+1	X(2^10)*1023+1
R_2	X2	X2^10+2	X(2^10)*1022+2	X(2^10)*1023+2
R_{1022}	X1022	X(2^10)*2-2	X(2^10)*1023-2	X(2^10)*1024-2
R_{1023}	X1023	X(2^10)*2-1	X(2^10)*1023-1	X(2^10)*1024-1

F engine

X engine expects groups of a single channel and a single antenna



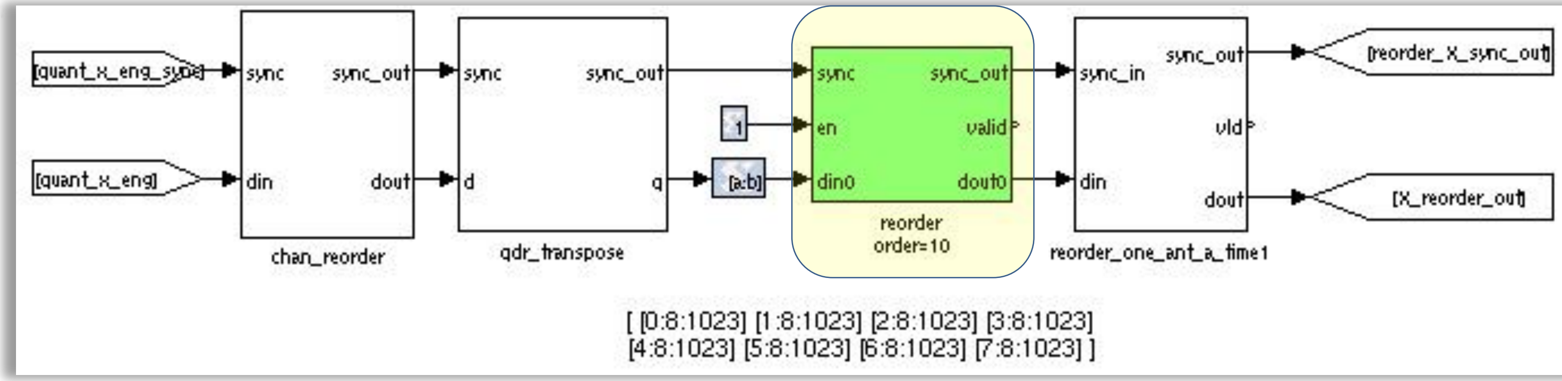
Every 8 column the same frequency channel is referring to the next spectra, next channels are grouped by 8x128 (=1024=2^10)

⇨ $M^T =$

t_0	t_1	t_2	t_8	$t_{2^{10}}$	$t_{2^{20}}$
a0c0	a1c0	a2c0	a0c0	a0c1	a7c1023
a8c0	a9c0	a10c0	a8c0	a8c1	a15c1023
a16c0	a17c0	a18c0	a16c0	a16c1	a23c1023
a24c0	a25c0	a26c0	a24c0	a24c1	a31c1023
X0	X2^10	X(2^10)*2	X(2^10)*8	X1	X(2^20)

F engine

X engine expects groups of a single channel and a single antenna



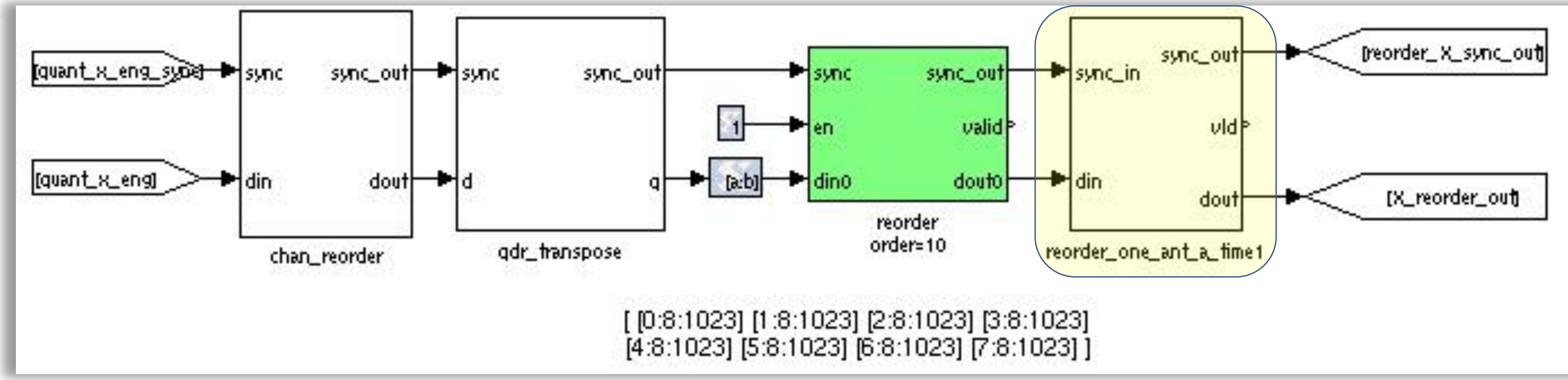
The reorder map at this stage groups data by 8 (let's call M^{T8})
[0 8 16 24 32 … 1008 1016 1 9 17 … 999 1007 1015 1023]
Therefore grouping same channels of different acquisitions

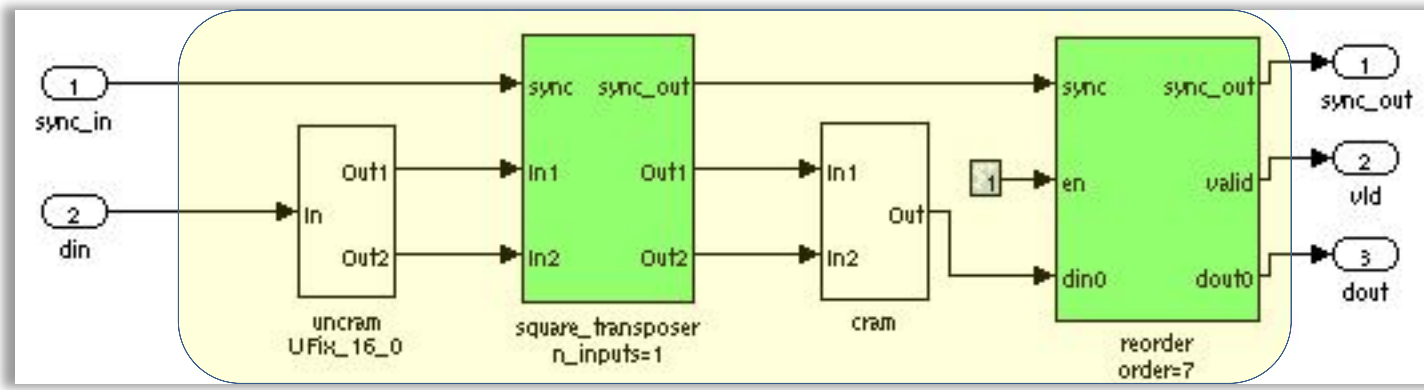
$\Rightarrow M^{T8} =$

t_0	t_1	t_2	t_{127}	t_{128}	$t_{2^\wedge 10}$	$t_{2^\wedge 20}$
a0c0	a0c0	a0c0	a0c0	a1c0	a0c1	a7c1023
a8c0	a8c0	a8c0	a8c0	a9c0	a8c1	a15c1023
a16c0	a16c0	a16c0	a16c0	a17c0	a16c1	a23c1023
a24c0	a24c0	a24c0	a24c0	a25c0	a24c1	a31c1023
X0	X(2^10)*8	X(2^10)*16	X(2^10)*1016	X2^10	X1	X(2^20)

F engine

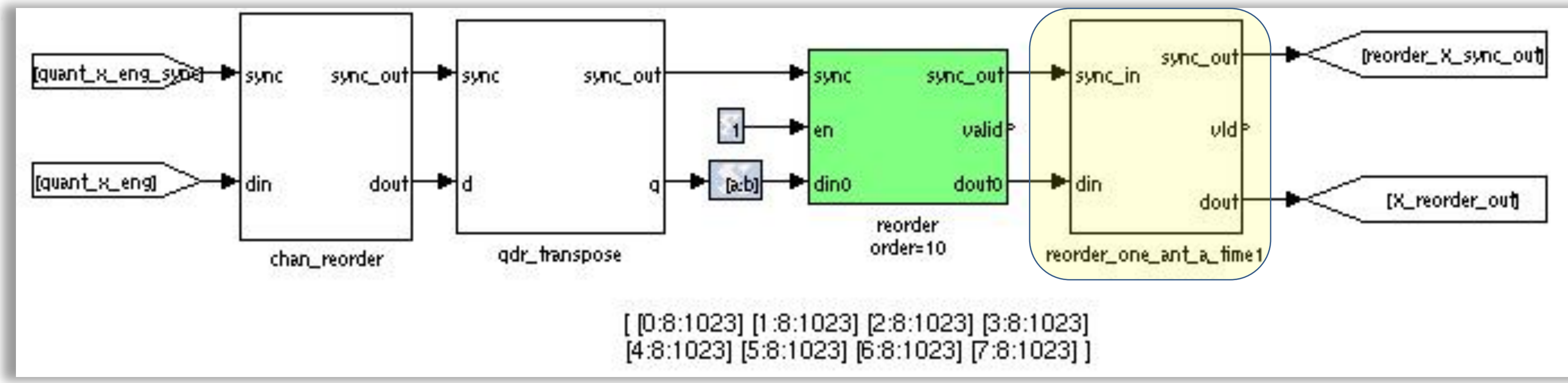
X engine expects groups of a single channel and a single antenna

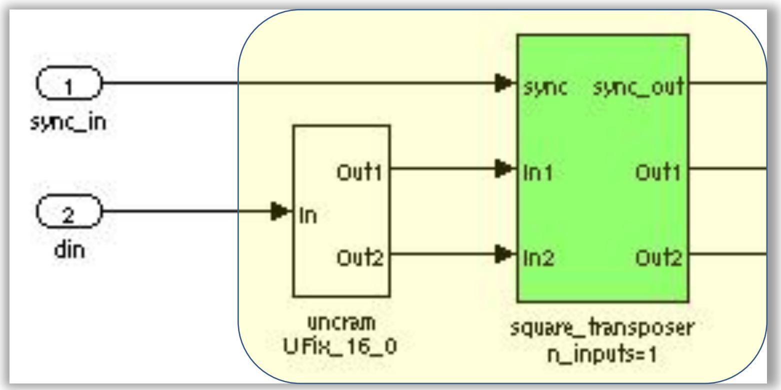




F engine

X engine expects groups of a single channel and a single antenna

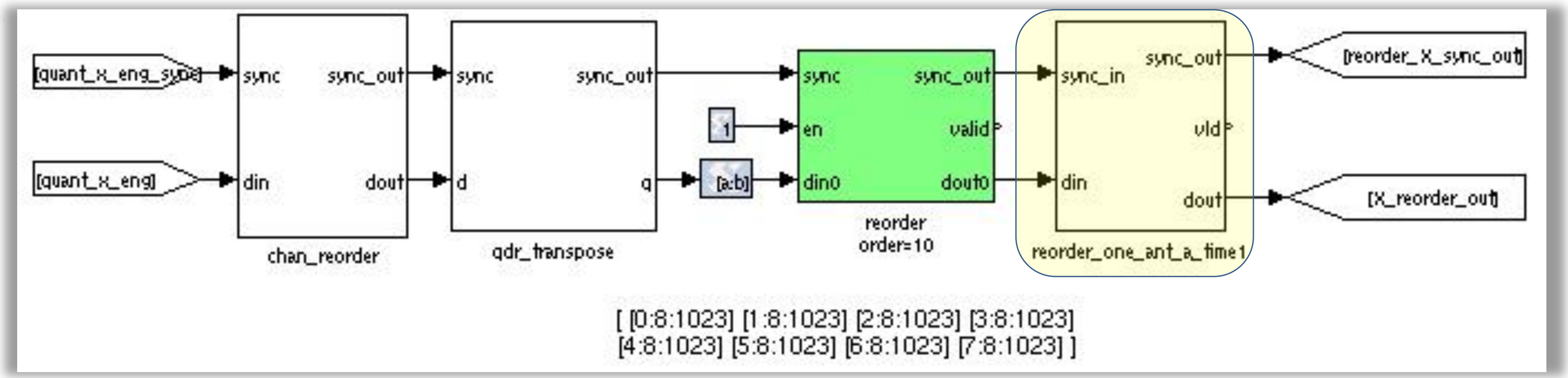


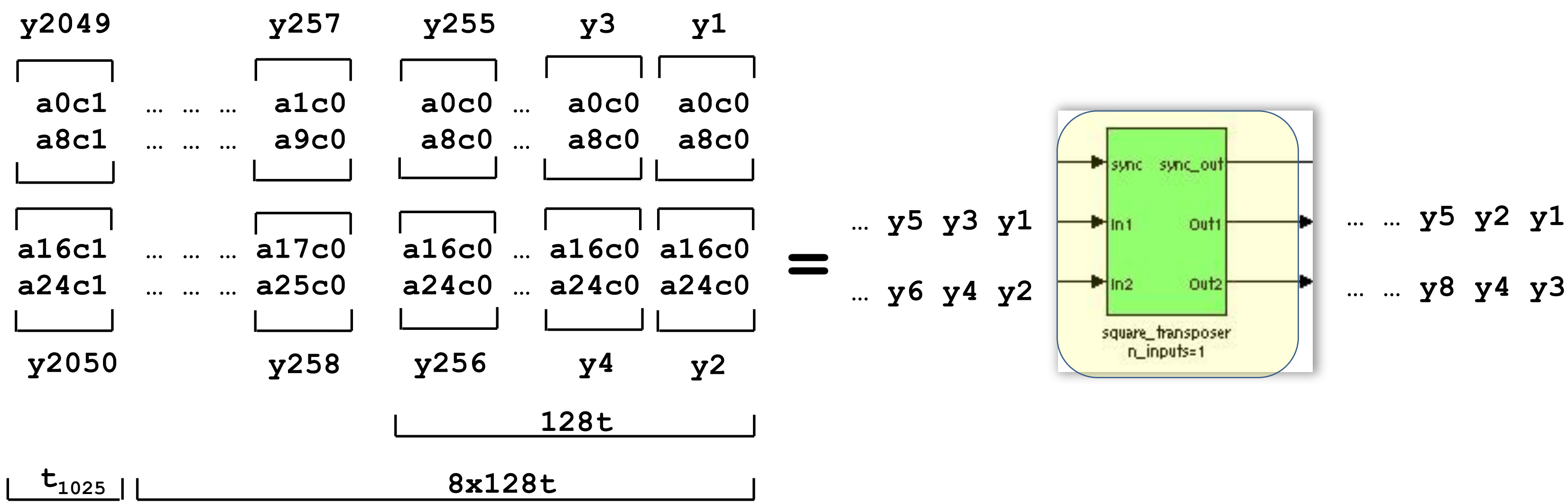


Uncram split a 32 bit word in two separate streams of 16 bit (high and low)
Square transposer presents a number of parallel inputs serially on the same number of output lines.

F engine

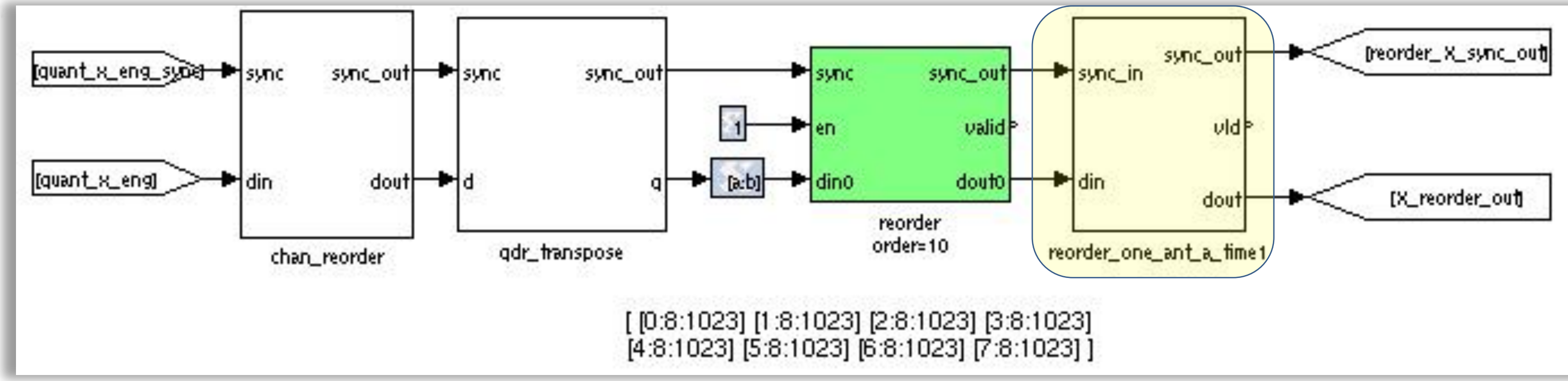
X engine expects groups of a single channel and a single antenna

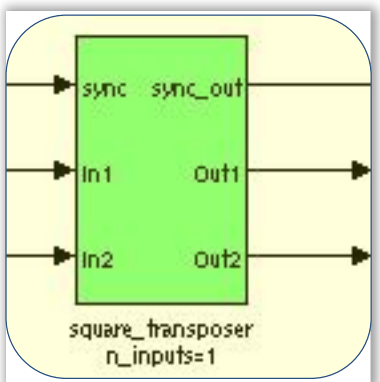


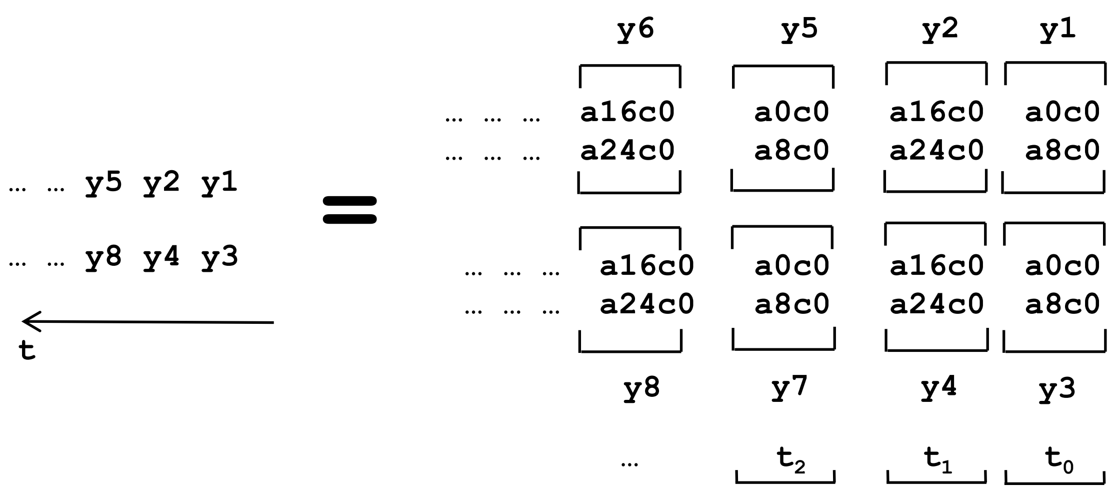


F engine

X engine expects groups of a single channel and a single antenna

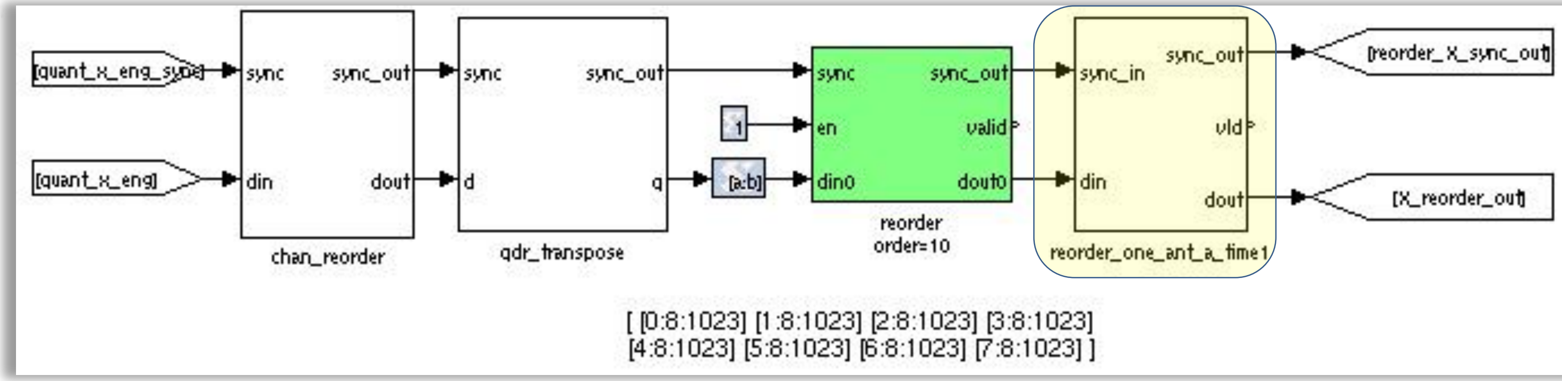






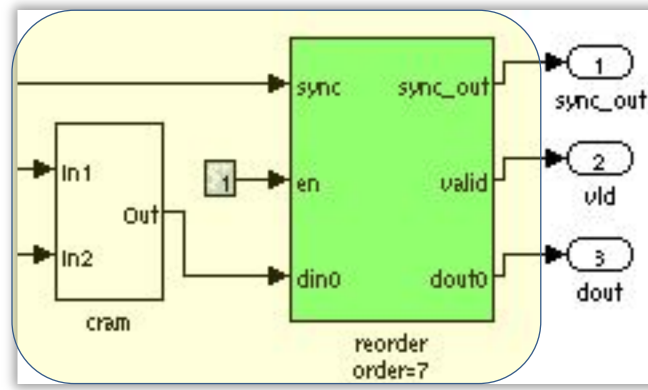
F engine

X engine expects groups of a single channel and a single antenna



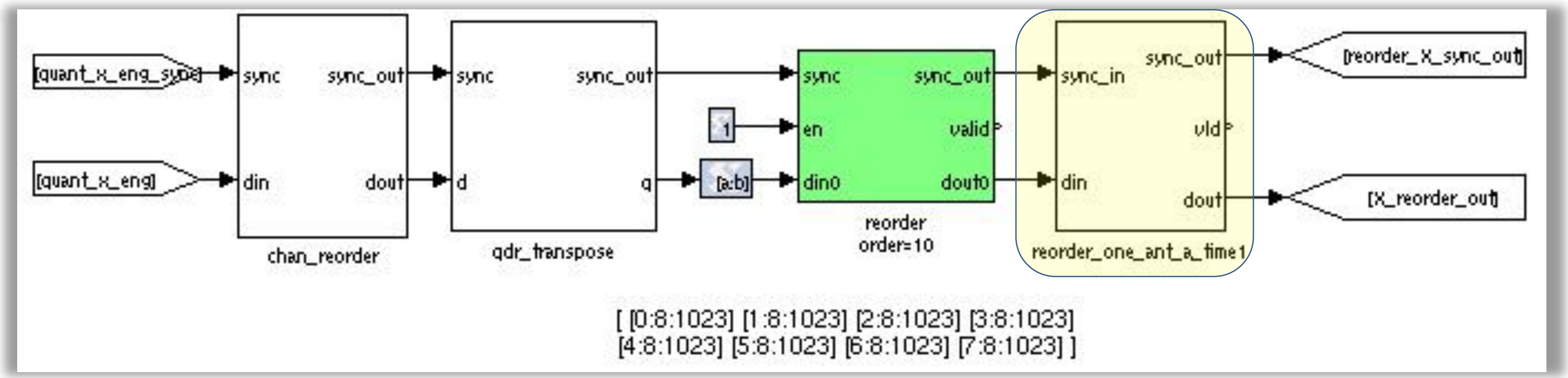
Reorder_one_ant_a_time init

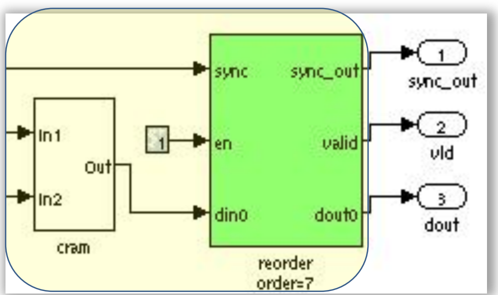
```
spec_chan = 10; % mask parameter
block_size = 7; % mask parameter

partial_reorder = [0:2:2^block_size - 1]

reorder = []
for n = [0:1]
    reorder = [reorder, [partial_reorder]+n];
end

(result: [0 2 4 6 ... 126 1 3 5 7 ... 127])
```

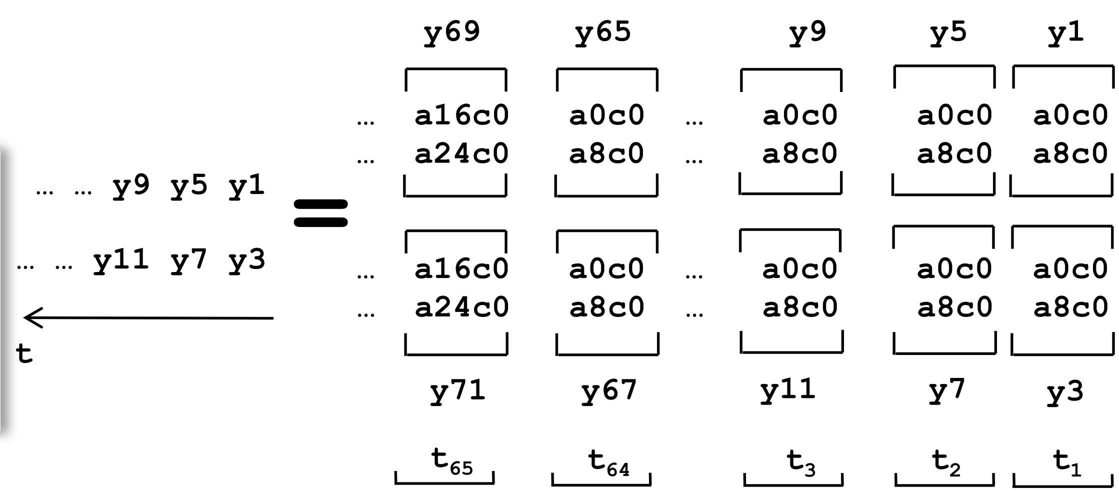


F engine

X engine expects groups of a single channel and a single antenna



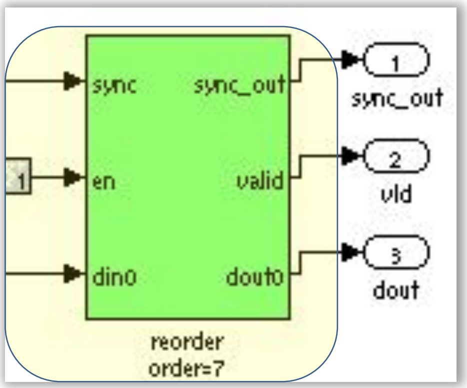




F engine

64 clock cycle for a complete set of 128 samples of the same frequency channel of 2 antennas (complex number, r/i 4.3 bit each)

(read this table from right to left and bottom up!)



t_{Z17}	t_{Z16}	t_{Z15}	t_{Z14}	t_{Z13}	t_{Z12}	t_{Z11}	t_{Z10}	t_{Z9}
a16c1	a0c1	a23c0	a7c0	a22c0	a6c0	a21c0	a5c0	a20c0
a24c1	a8c1	a31c0	a15c0	a30c0	a14c0	a29c0	a13c0	a28c0
a16c1	a0c1	a23c0	a7c0	a22c0	a6c0	a21c0	a5c0	a20c0
a24c1	a8c1	a31c0	a15c0	a30c0	a14c0	a29c0	a13c0	a28c0
64t	64t	64t	64t	64t	64t	64t	64t	64t

t_{Z8}	t_{Z7}	t_{Z6}	t_{Z5}	t_{Z4}	t_{Z3}	t_{Z2}	t_{Z1}	t_{Z0}
a4c0	a19c0	a3c0	a18c0	a2c0	a17c0	a1c0	a16c0	a0c0
a12c0	a27c0	a11c0	a26c0	a10c0	a25c0	a9c0	a24c0	a8c0
a4c0	a19c0	a3c0	a18c0	a2c0	a17c0	a1c0	a16c0	a0c0
a12c0	a27c0	a11c0	a26c0	a10c0	a25c0	a9c0	a24c0	a8c0
64t	64t	64t	64t	64t	64t	64t	64t	64t

F engine

X engine data order

32 bit x 64t each
antennas interlived by 8 bit

t_{Z15}	t_{Z14}	t_{Z13}	t_{Z12}	t_{Z11}	t_{Z10}	t_{Z9}	t_{Z8}	t_{Z7}	t_{Z6}	t_{Z5}	t_{Z4}	t_{Z3}	t_{Z2}	t_{Z1}	t_{Z0}
a23	a7	a22	a6	a21	a5	a20	a4	a19	a3	a18	a2	a17	a1	a16	a0
a31	a15	a30	a14	a29	a13	a28	a12	a27	a11	a26	a10	a25	a9	a24	a8

Considering the original number the X-engine
will receive data in this order

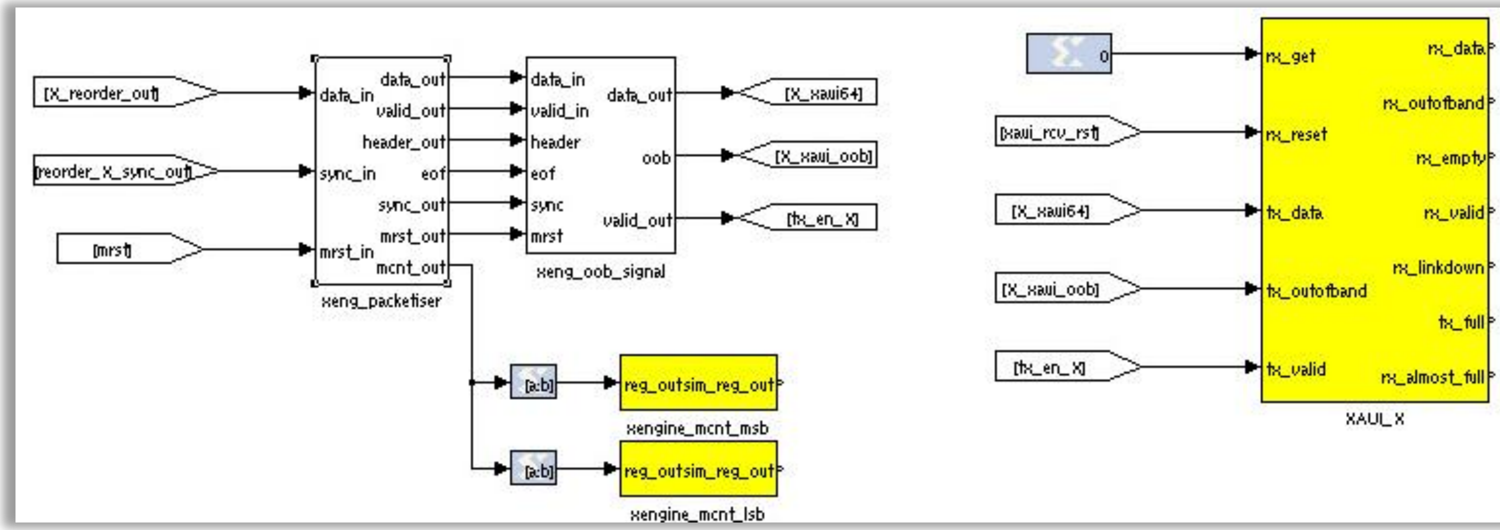
FFT real output sequence	New numbering	Prefix
0-4-1-5-2-6-3-7	0-1-2-3-4-5-6-7	0-7
8_12_9_13_10_14_11_15	8-9-10-11-12-13-14-15	8-15
16_20_17_21_18_22_19_23	16-17-18-19-20-21-22-23	16-23
24_28_25_29_26_30_27_31	24-25-26-27-28-29-30-31	24-31

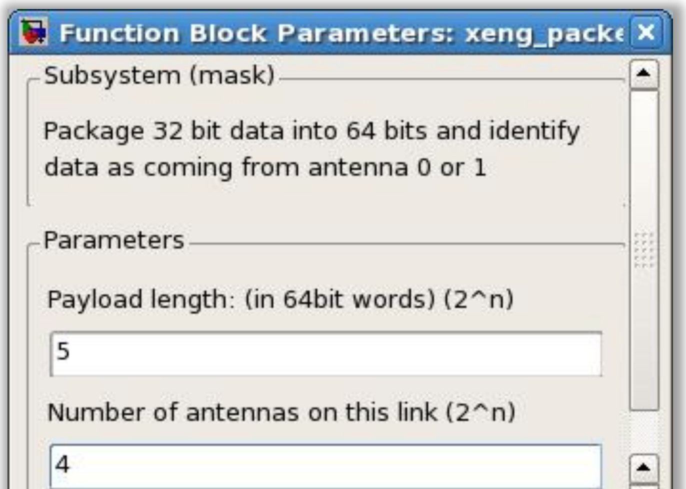
the final sequence is

t_{Z15}	t_{Z14}	t_{Z13}	t_{Z12}	t_{Z11}	t_{Z10}	t_{Z9}	t_{Z8}	t_{Z7}	t_{Z6}	t_{Z5}	t_{Z4}	t_{Z3}	t_{Z2}	t_{Z1}	t_{Z0}
a23	a7	a19	a3	a22	a6	a18	a2	a21	a5	a17	a1	a20	a4	a16	a0
a31	a15	a27	a11	a30	a14	a26	a10	a29	a13	a25	a9	a28	a12	a24	a8

F engine

Packing data to send over XAUI CX4 to X engine ROACH





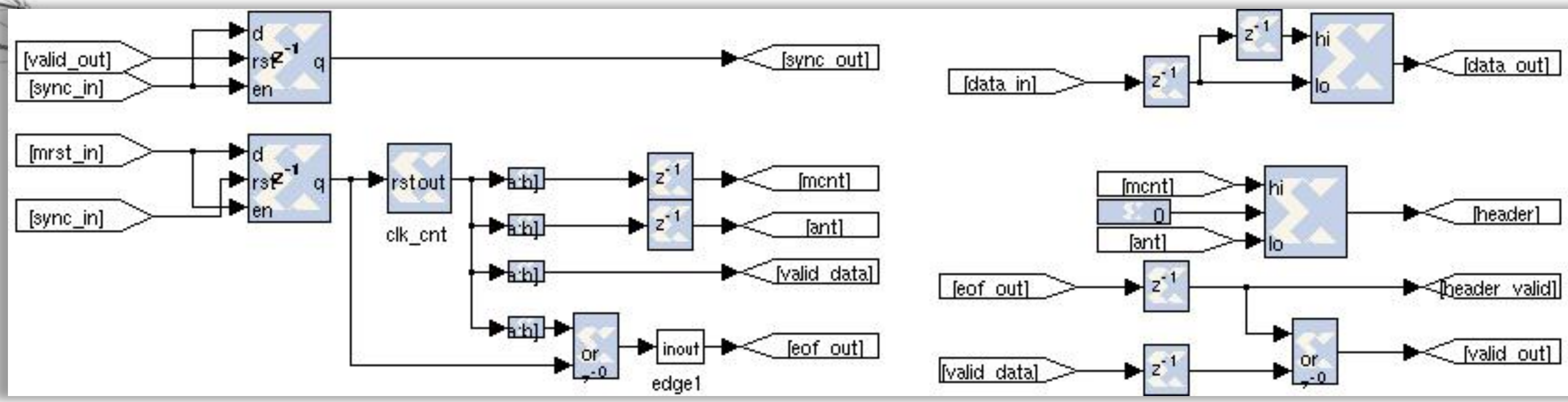
The 'antenna' number is used to index the packets which make up one integration.

Data is tagged with a 'mcnt' number and sync and eof headers.

These are later decoded and used for error checking and data ordering in the X-engines

F engine

…packaging…



data in: 128 samples * 2 ant * 8b each = 32b * 64t = 32 words * 64b

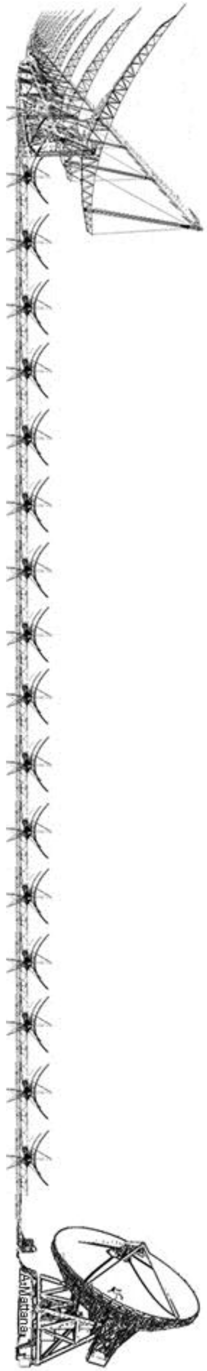
ant_bits = 4 (2^4 = 16 antennas in dual pol, instead of 32 ant single pol)
nwrd_bits = 5 (2^5 = 32 payload length)

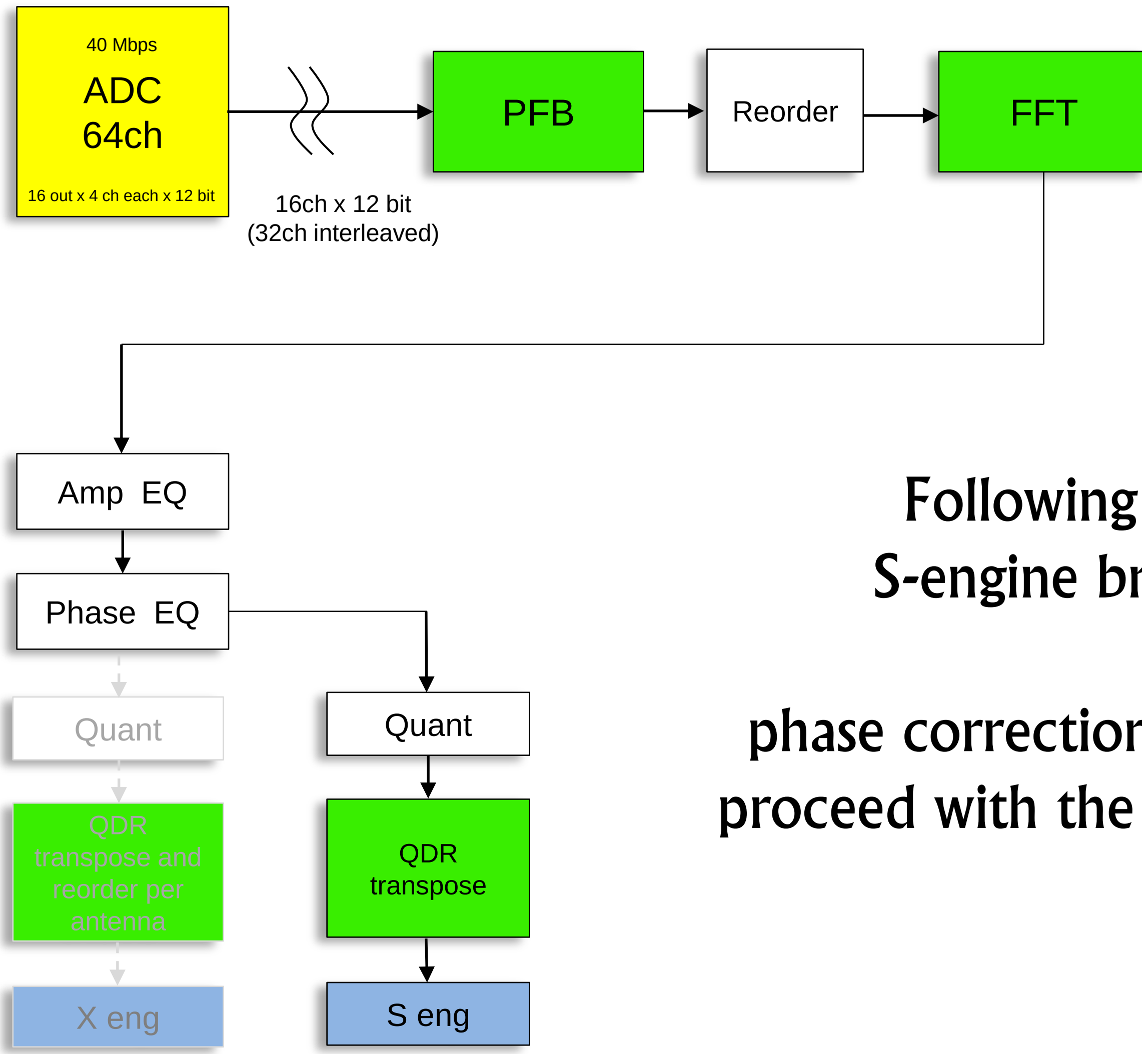
clk_cnt #bits = 48 + ant_bits + nwrd_bits + 1
(the last additional bit is needed to concatenate and validate 64bit of data)

mcnt #bits = 48 (counts the channel frequencies)

header #bits = 64 = mcnt[48] + zeroes pads + ant[4]

F engine

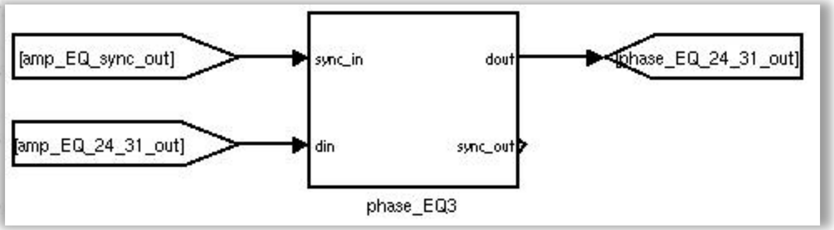




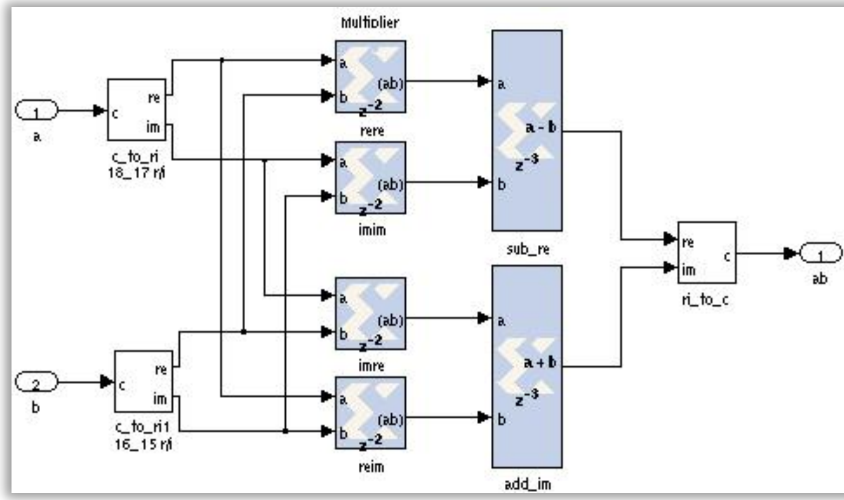
Following the S-engine branch

phase corrections allow to proceed with the spatial FFT

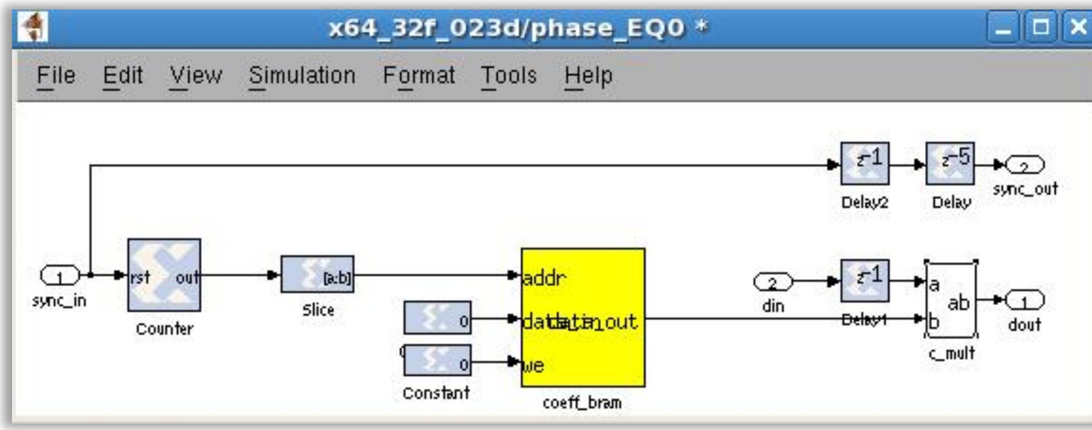
Phase corrections done!



Phase corrections are obtained multiplying the complex conjugate of the new coefficents



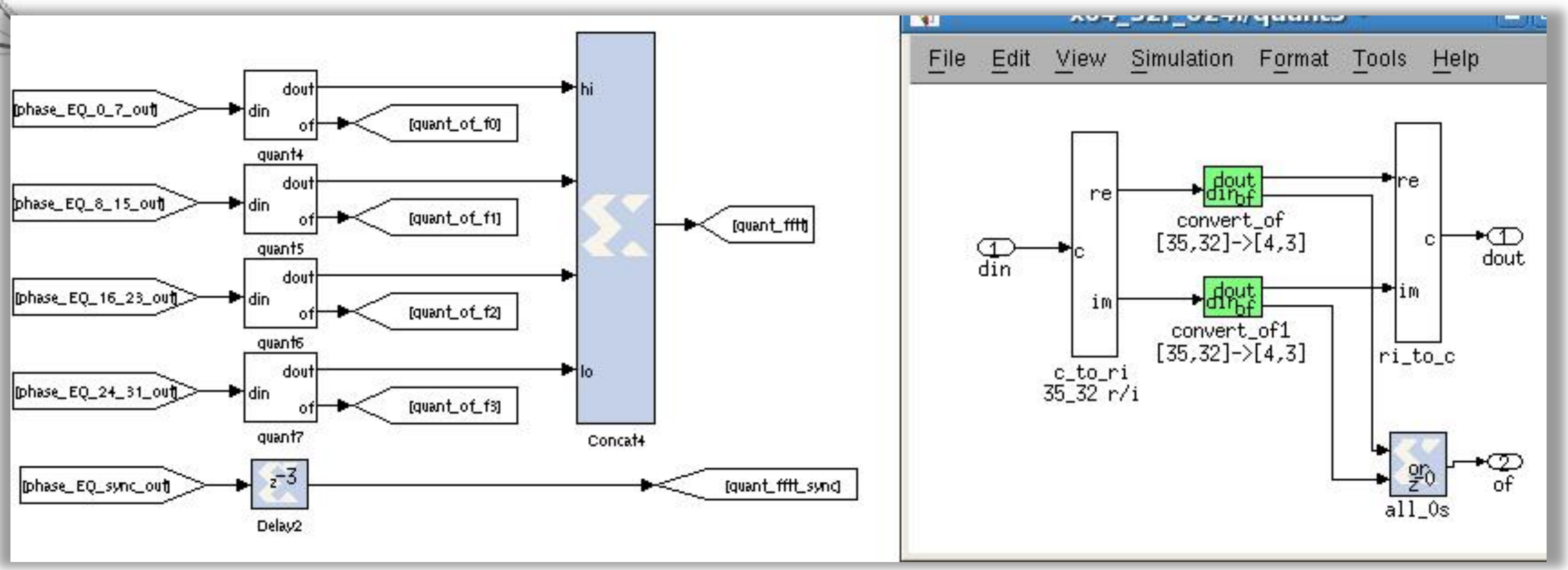
c_mult details



(bitwidth fft) 18.17 *
(phase correction) 16.15 =
35.32 b

F engine

Quantization



From 35.32 (x2) to 4.3 (x2), quant_fft 32 bit

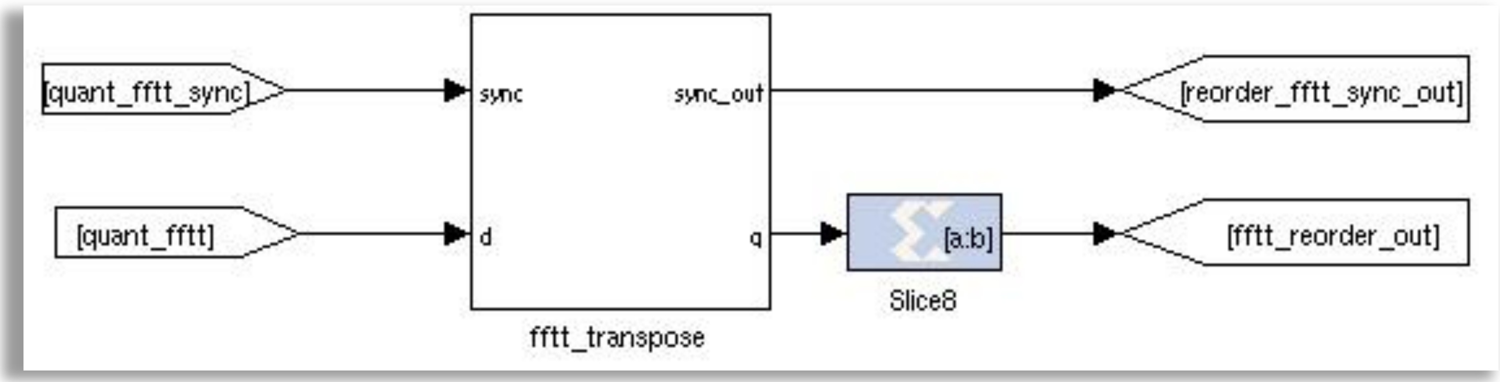
quant_fft

a0c0,	a0c1,	…,	a0c1023,	a1c0,	a1c1,	…,	a7c1023
a8c0,	a8c1,	…,	a8c1023,	a9c0,	a9c1,	…,	a15c1023
a16c0,	a16c1,	…,	a16c1023,	a17c0,	a17c1,	…,	a23c1023
a24c0,	a24c1,	…,	a24c1023,	a25c0,	a25c1,	…,	a31c1023

t_0 t_1 $t_{\frac{M}{2}}$ $t_{\frac{M}{2}+1}$ $t_{\frac{M}{2}+2}$ $t_{\left(\frac{M}{2}\right)*8}$

F engine

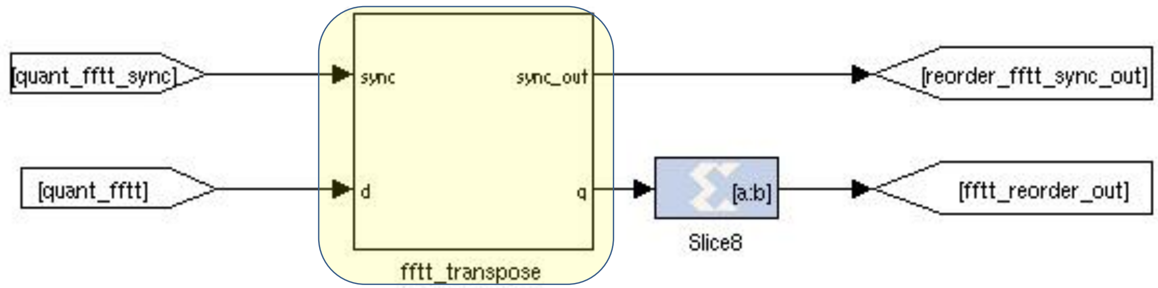
The Fourier Imager (S engine) expects blocks of all antennas per one frequency channel

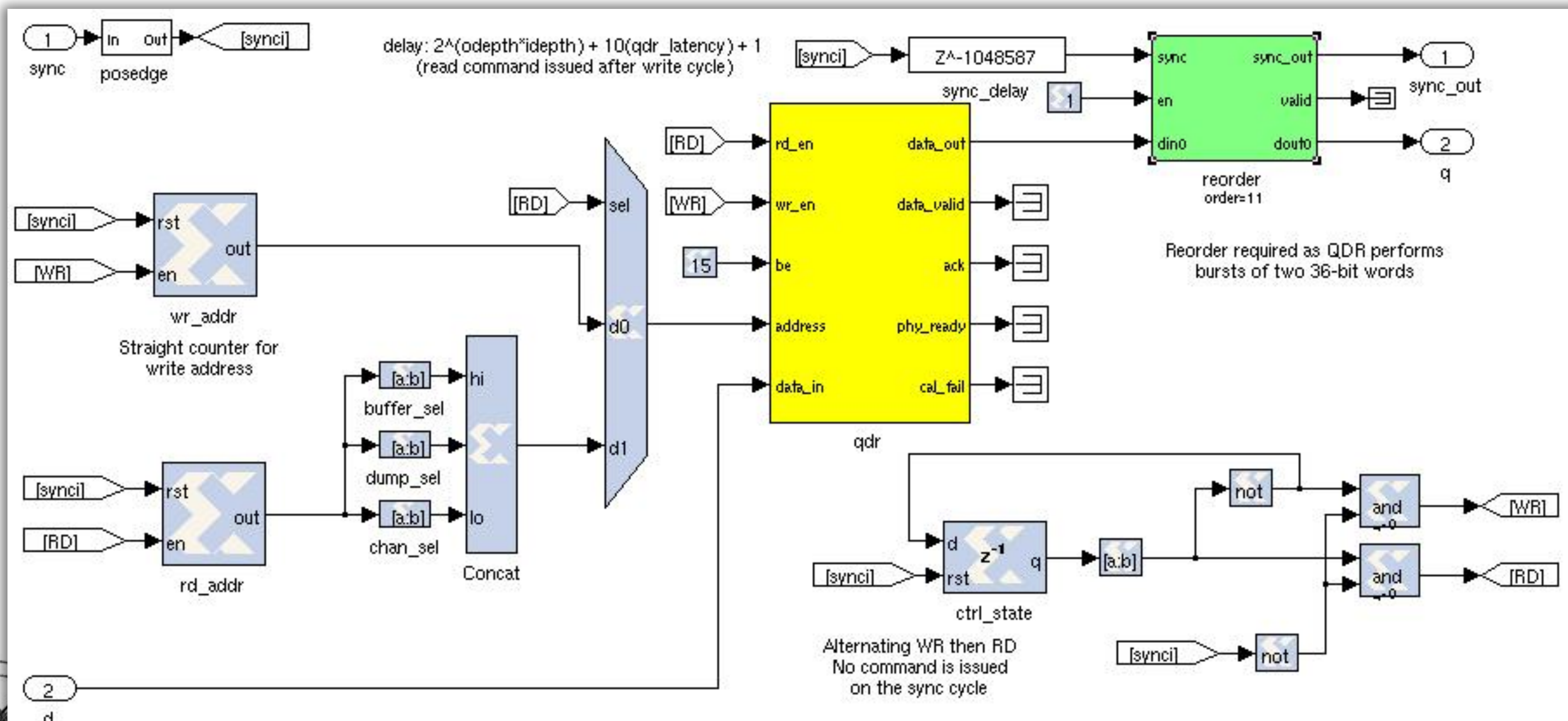


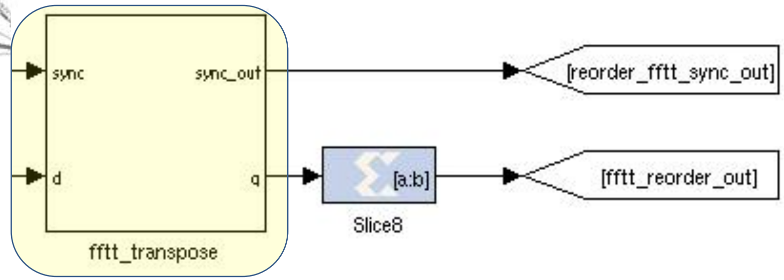
This is done by using the second Corner Turn Memory (QDR) available aboard in the ROACH

F engine

The logic driving the QDR chip is always the same







t_0	t_1	t_{1024}	t_{end}
a0c0	a0c1	a1c0	a7c1023
a8c0	a8c1	a9c0	a15c1023
a16c0	a16c1	a17c0	a23c1023
a24c0	a24c1	a25c0	a31c1023
X0	X1	X1024	X8191

Using QDR1 as 2^{10} x 2^{10} matrix

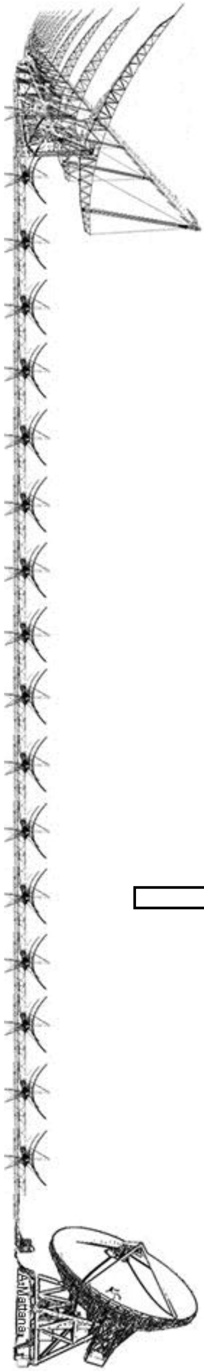
M =

	C_0	C_1	C_{1022}	C_{1023}
R_0	X0	X1	X1022	X1023
R_1	X2^10	X1025	X(2^10)*2-2	X(2^10)*2-1
R_2	X(2^10)*2	X(2^10)*2+1	X(2^10)*3-2	X(2^10)*3-1
R_{1022}	X(2^10)*1022	X(2^10)*1022+1	X(2^10)*1023-2	X(2^10)*1023-1
R_{1023}	X(2^10)*1023	X(2^10)*1023+1	X(2^10)*1024-2	X(2^10)*1024-1

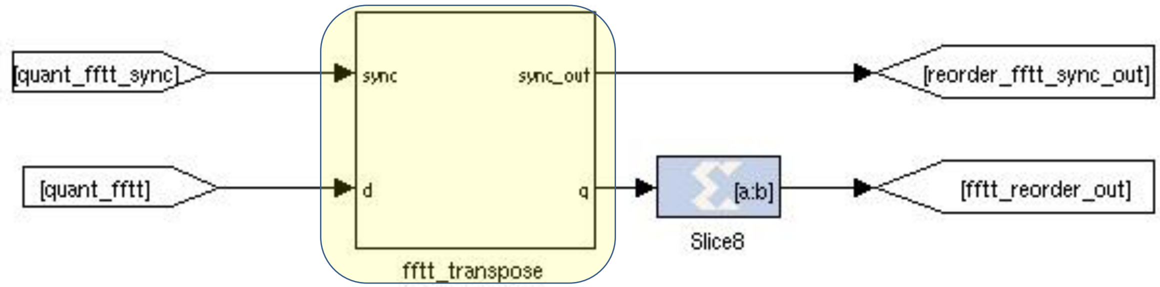
1024*1024 cells means also 128 spectra of each antenna

The Fourier Imager (S engine) expects blocks of all antennas per one frequency channel

F engine



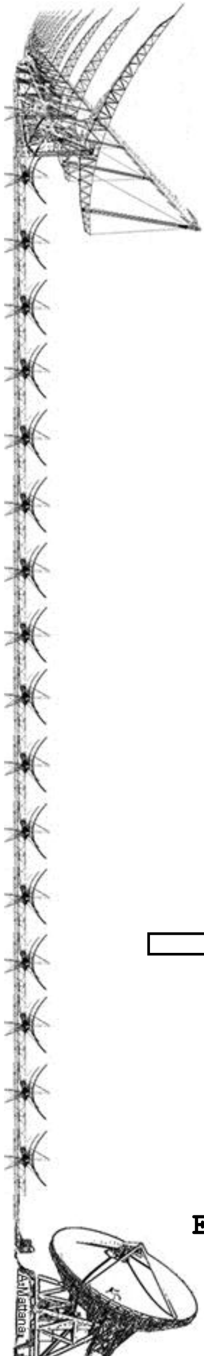
The Fourier Imager (S engine) expects blocks of all antennas per one frequency channel



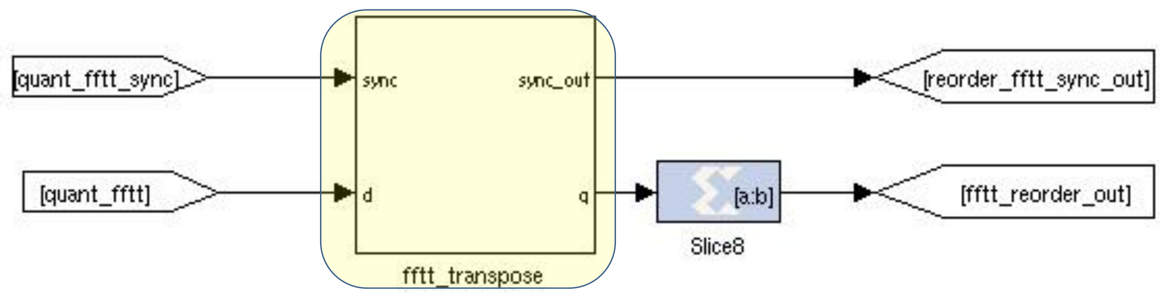
$\Rightarrow M^T =$

	C_0	C_1	C_{1022}	C_{1023}
R_0	X0	X2^10	X(2^10)*1022	X(2^10)*1023
R_1	X1	X2^10+1	X(2^10)*1022+1	X(2^10)*1023+1
R_2	X2	X2^10+2	X(2^10)*1022+2	X(2^10)*1023+2
R_{1022}	X1022	X(2^10)*2-2	X(2^10)*1023-2	X(2^10)*1024-2
R_{1023}	X1023	X(2^10)*2-1	X(2^10)*1023-1	X(2^10)*1024-1

F engine



The Fourier Imager (S engine) expects blocks of all antennas per one frequency channel



$\Rightarrow \quad M^T =$

t_0	t_1	t_2	t_8	$t_{2\wedge 10}$	$t_{2\wedge 20}$
a0c0	a1c0	a2c0	a0c0	a0c1	a7c1023
a8c0	a9c0	a10c0	a8c0	a8c1	a15c1023
a16c0	a17c0	a18c0	a16c0	a16c1	a23c1023
a24c0	a25c0	a26c0	a24c0	a24c1	a31c1023
X0	X2^10	X(2^10)*2	X(2^10)*8	X1	X(2^20)

Every 8 column the same frequency channel is referring to the next spectra, next channels are grouped by 8x128 (=1024=2^10)

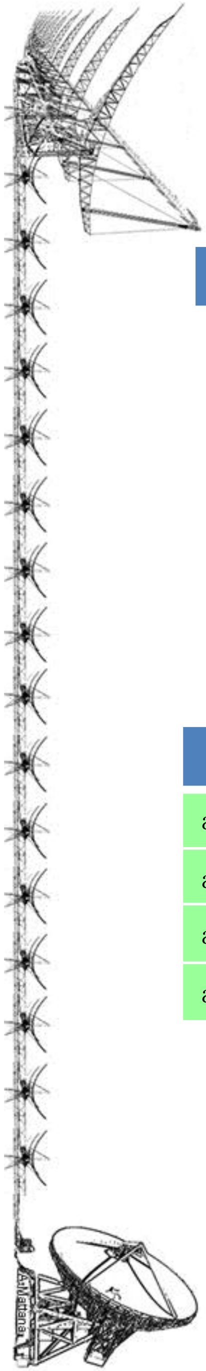
S engine data order

t_{2^20}	t_{3072}	t_{2048}	t_{1024}	t_9	t_8	t_7	t_6	t_5	t_4	t_3	t_2	t_1	t_0
a7c1023	a0c3	a0c2	a0c1	a1	a0	a7	a6	a5	a4	a3	a2	a1	a0
a15c1023	a8c3	a8c2	a8c1	a9	a8	a15	a14	a13	a12	a11	a10	a9	a8
a23c1023	a16c3	a16c2	a16c1	a17	a16	a23	a22	a21	a20	a19	a18	a17	a16
a31c1023	a24c3	a24c2	a24c1	a25	a24	a31	a30	a29	a28	a27	a26	a25	a24

Considering the original number the S-engine will receive data in this order

FFT real output sequence	New numbering	Prefix
0-4-1-5-2-6-3-7	0-1-2-3-4-5-6-7	0-7
8_12_9_13_10_14_11_15	8-9-10-11-12-13-14-15	8-15
16_20_17_21_18_22_19_23	16-17-18-19-20-21-22-23	16-23
24_28_25_29_26_30_27_31	24-25-26-27-28-29-30-31	24-31

F engine



S engine data order

$t_{2^{\wedge}20}$	t_{3072}	t_{2048}	t_{1024}	t_9	t_8	t_7	t_6	t_5	t_4	t_3	t_2	t_1	t_0
a7c1023	a0c3	a0c2	a0c1	a1	a0	a7	a6	a5	a4	a3	a2	a1	a0
a15c1023	a8c3	a8c2	a8c1	a9	a8	a15	a14	a13	a12	a11	a10	a9	a8
a23c1023	a16c3	a16c2	a16c1	a17	a16	a23	a22	a21	a20	a19	a18	a17	a16
a31c1023	a24c3	a24c2	a24c1	a25	a24	a31	a30	a29	a28	a27	a26	a25	a24

the final sequence is

$t_{2^{\wedge}20}$	t_{3072}	t_{2048}	t_{1024}	t_9	t_8	t_7	t_6	t_5	t_4	t_3	t_2	t_1	t_0
a7c1023	a0c3	a0c2	a0c1	a4	a0	a7	a3	a6	a2	a5	a1	a4	a0
a15c1023	a8c3	a8c2	a8c1	a12	a8	a15	a11	a14	a10	a13	a9	a12	a8
a23c1023	a16c3	a16c2	a16c1	a20	a16	a23	a19	a22	a18	a21	a17	a20	a16
a31c1023	a24c3	a24c2	a24c1	a28	a24	a31	a27	a30	a26	a29	a25	a28	a24

....

128s x freq ch 1

128 samples of the frequency channel 0

F engine

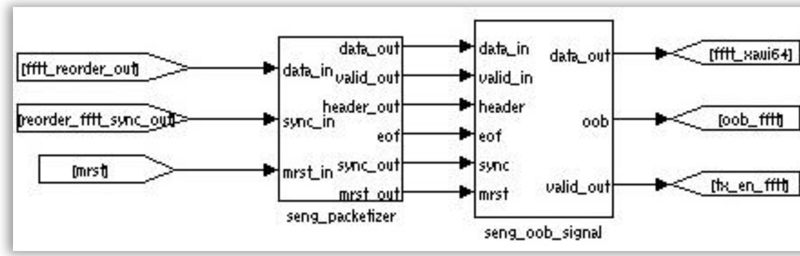
Packing data to send over XAUI CX4 to S engine ROACH

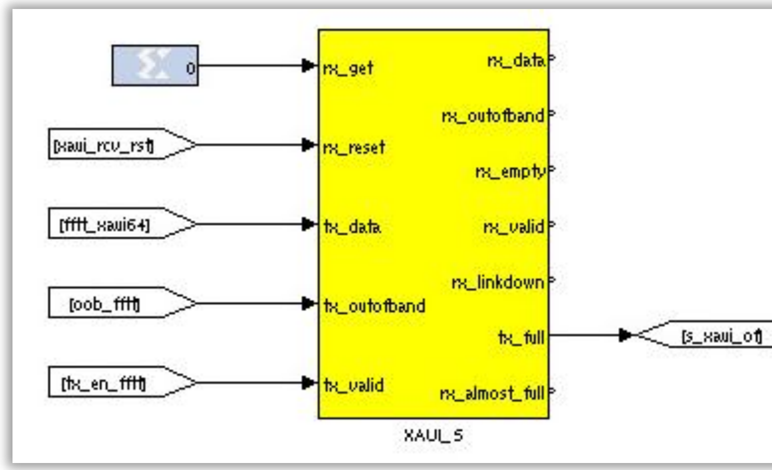
The S-engine integration length is the payload length by the number of windows (aka packets) per frequency channel.

The 'antenna' number is used to index the packets which make up one integration. Using standard X-engine ordering logic should sort things out on the rx side.

Data is tagged with a ‘mcnt’ number and sync and eof headers.

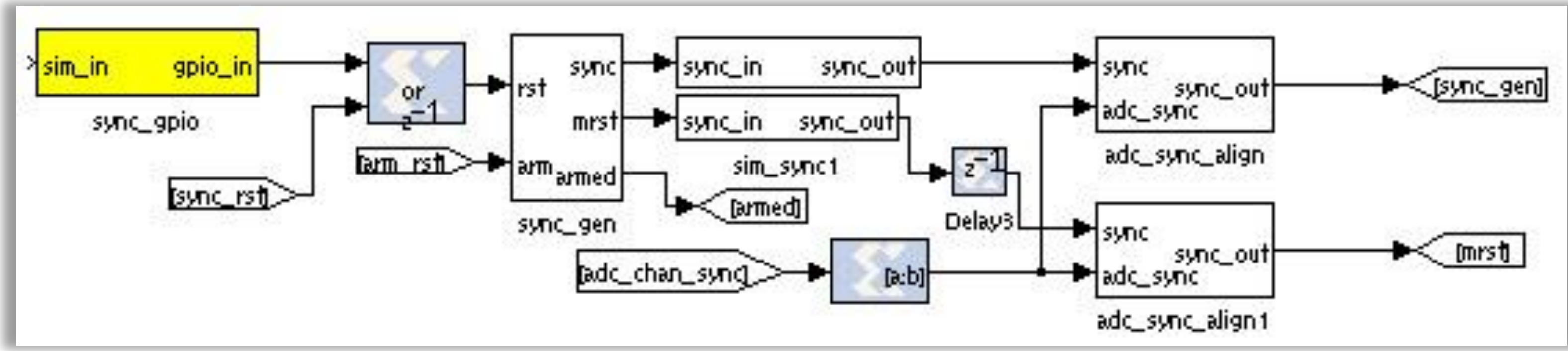
These are later decoded and used for error checking / data ordering in the spatial imager and X-engines





F engine

Control Signals and Registers



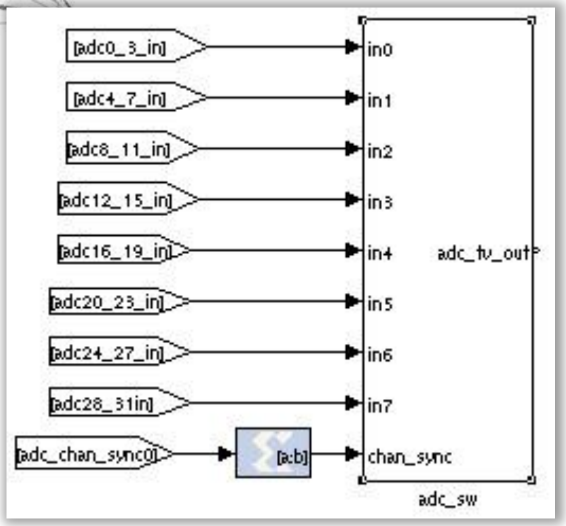
SYNC GEN

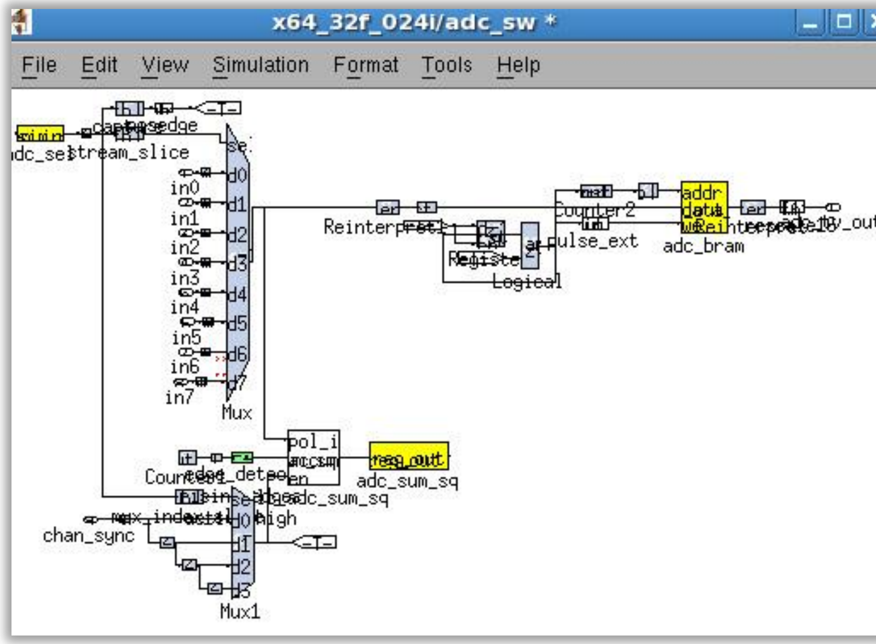
We tag on some logic after the sync gen to ensure that a sync pulse arrives the clock before the adc_channel sync, which signifies the arrival of the first multiplexed channel on the adc lines

2^N periods:
fft mux -- 13
QDR transpose -- 11
post QDR reorder -- 10
reorder 1_ant_a_time -- 7
LCM(13,11,10,7) = 10010

F engine

Control Signals and Registers





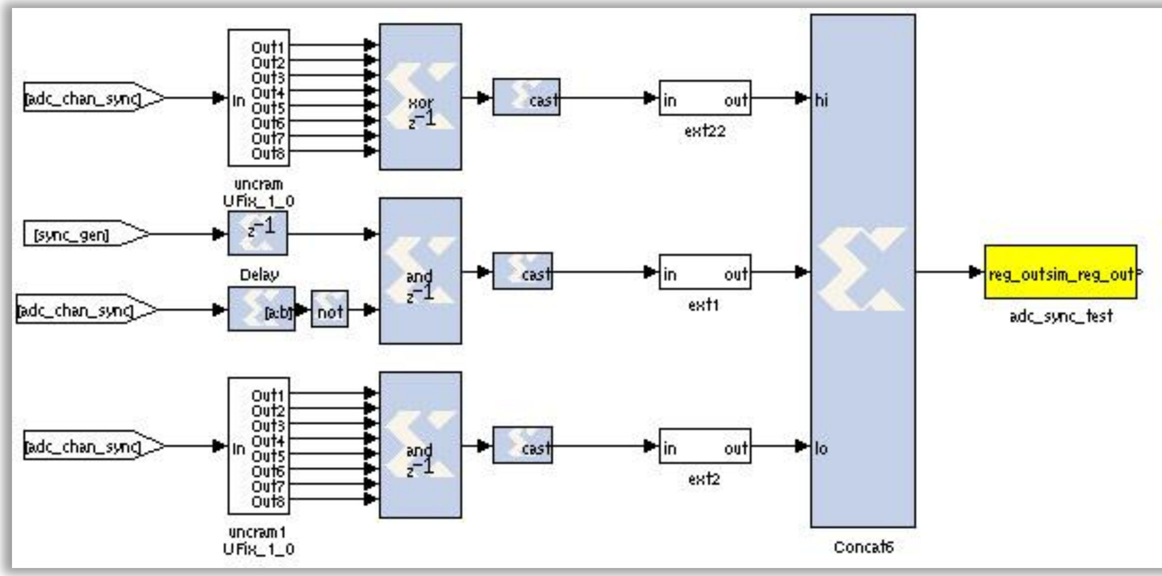
ADC SNAP

Stores the AD samples of the selected channel in a block RAM

adc_sel	in	selects the AD output line (0-3, 4-7, …, 28-31)
adc_bram	out	1024*32b block ram
adc_sum_sq	out	the square sum of the AD chan selected

F engine

Control Signals and Registers



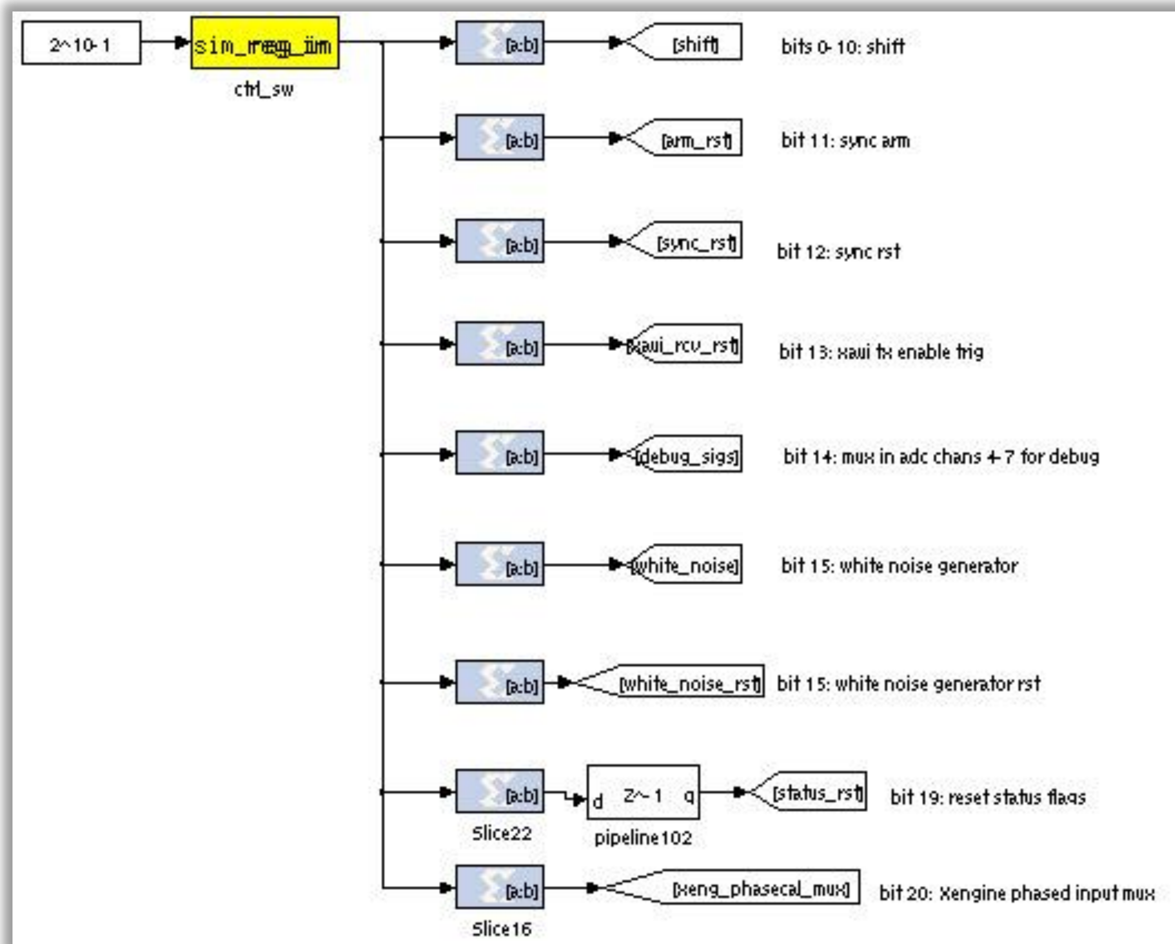
ADC SYNC TEST

The adc_sync_test reg allows the user to confirm that all 8 ADC chips are syncing together, and that these ADC syncs are arriving the clock before the master sync.

If all is going well, the register should show one

F engine

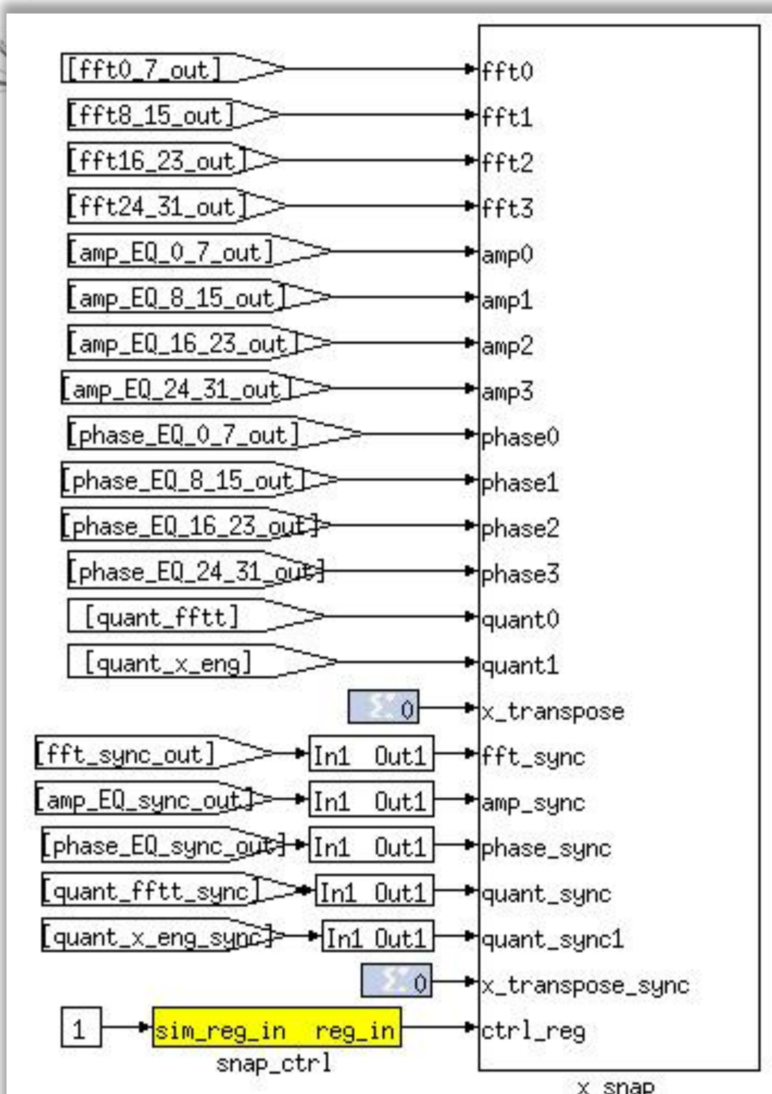
Control Signals and Registers



CTRL_SW: The ctrl_sw reg is used to manage the F engine

F engine

Control Signals and Registers



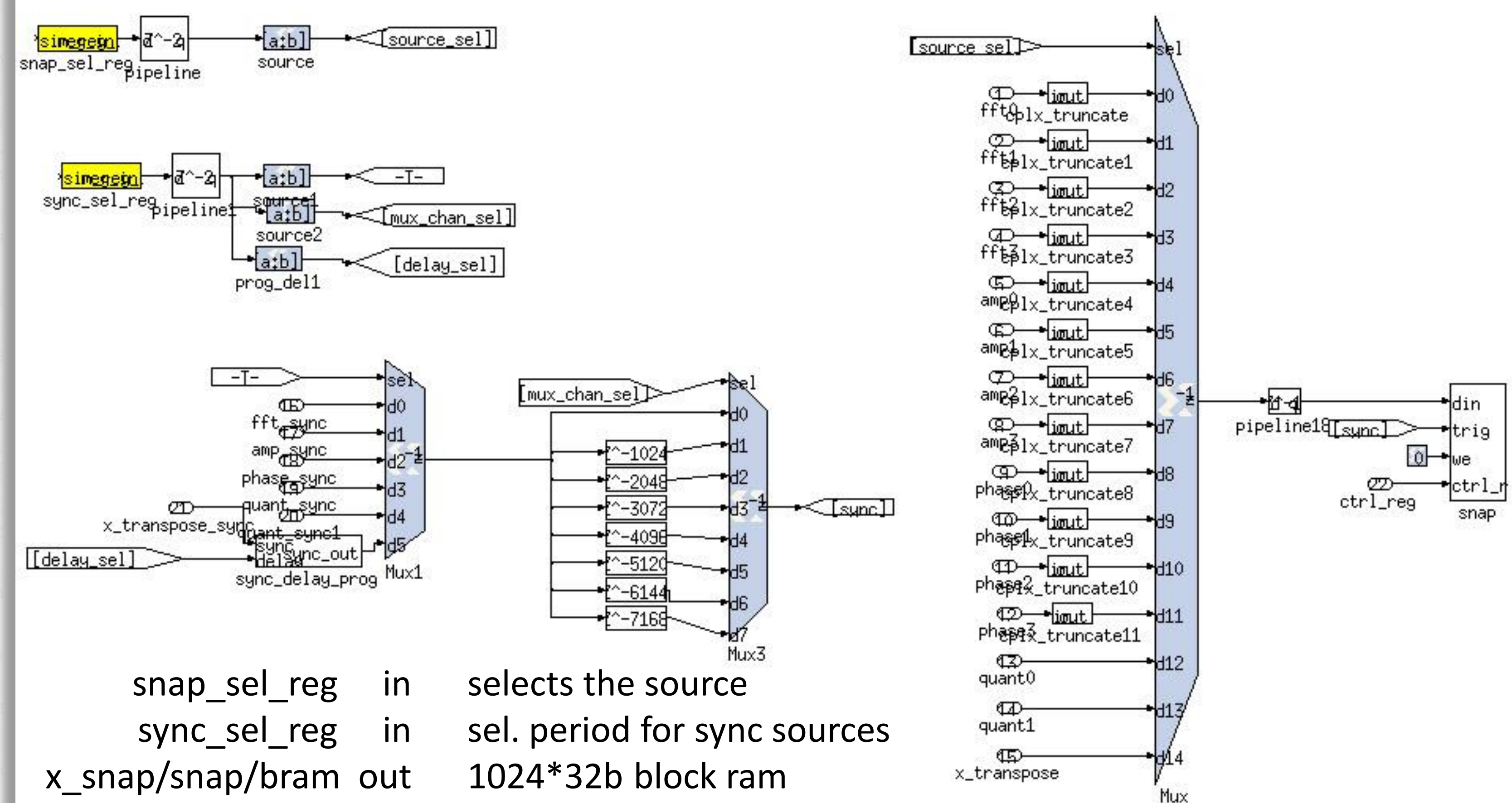
X_SNAP

You can see a snap of each stage of the F-engine by selecting a source, a period (only in case the source is a sync signal), and the output is stored to a 1024x32b block ram

(more details in next page)

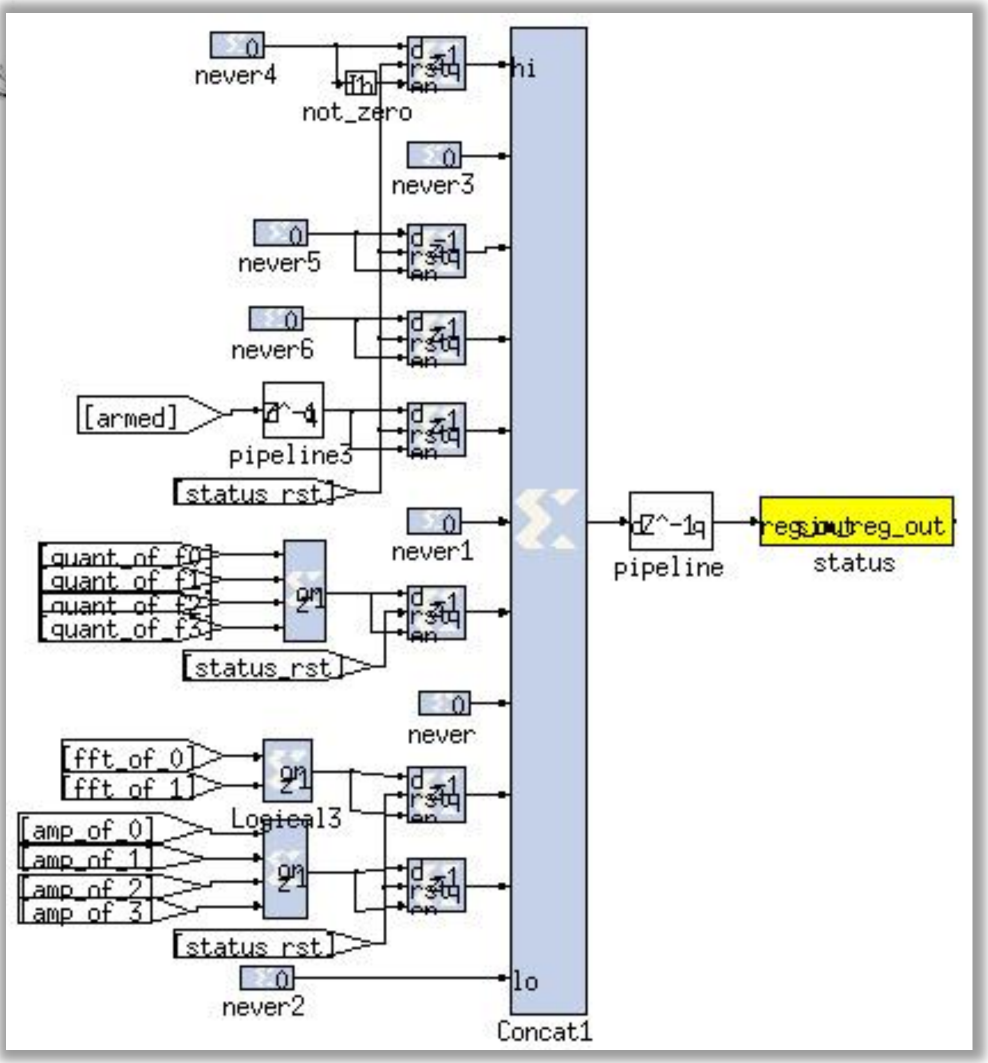
F engine

Control Signals and Registers



snap_sel_reg	in	selects the source
sync_sel_reg	in	sel. period for sync sources
x_snap/snap/bram	out	1024*32b block ram

Control Signals and Registers



STATUS

All the status flags are collected in the status register

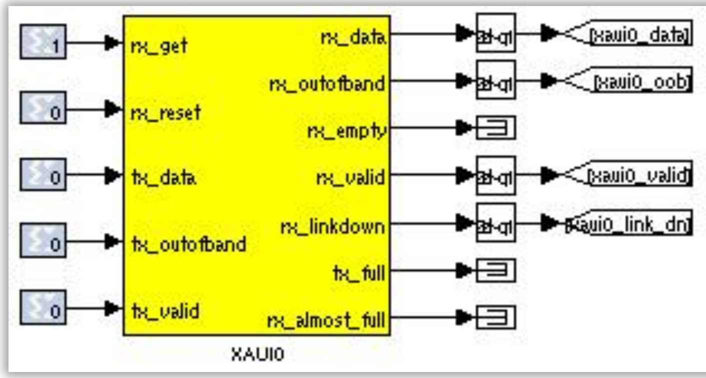
F engine

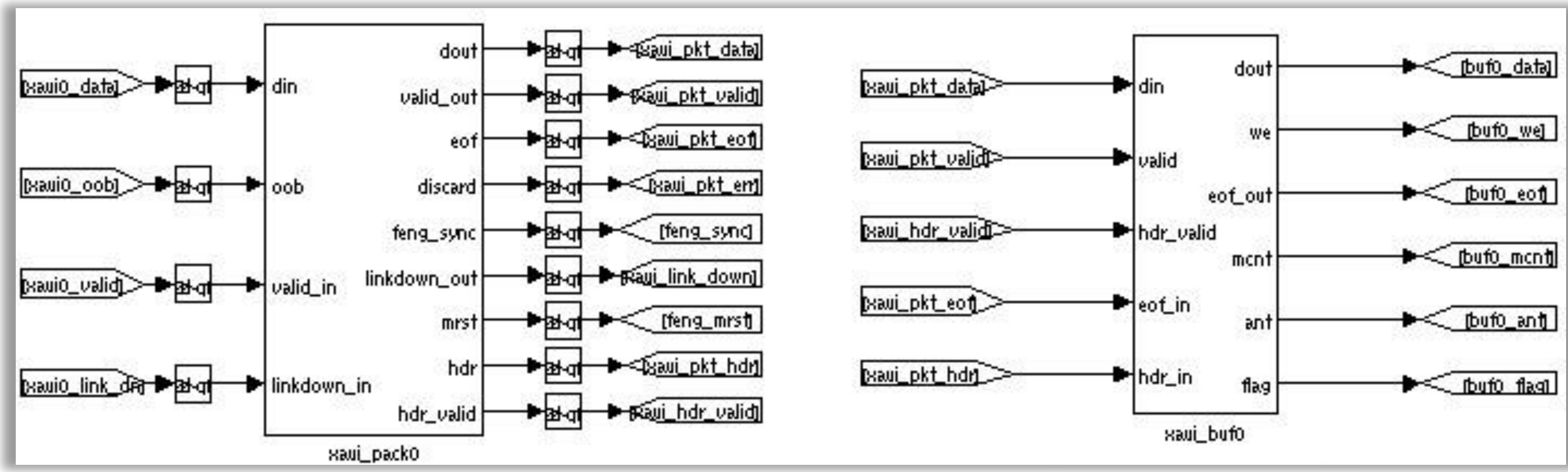
The X-engine Correlator

Roach-2

X engine

Getting data from F engine

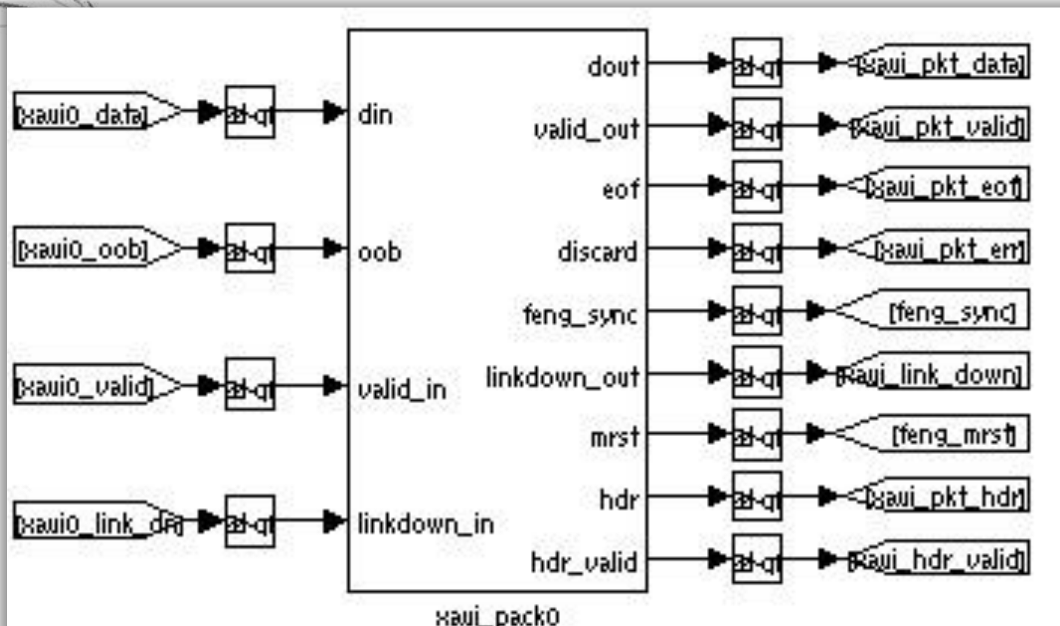


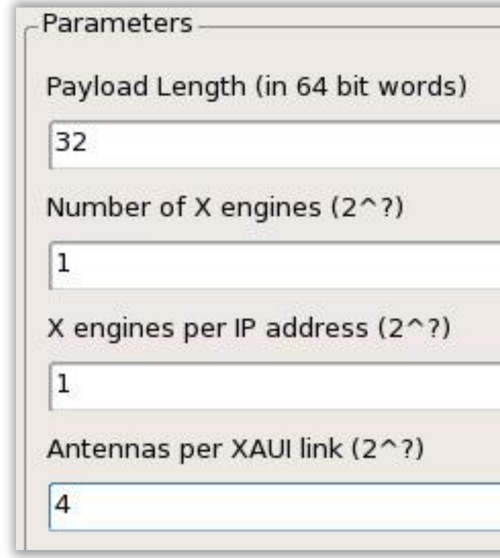


Accept 32 data words (64 bits each) from roach, plus a 1 word header
(16 words contains 128 samples of a single frequency for a single antenna * 2)

X engine

Getting data from F engine



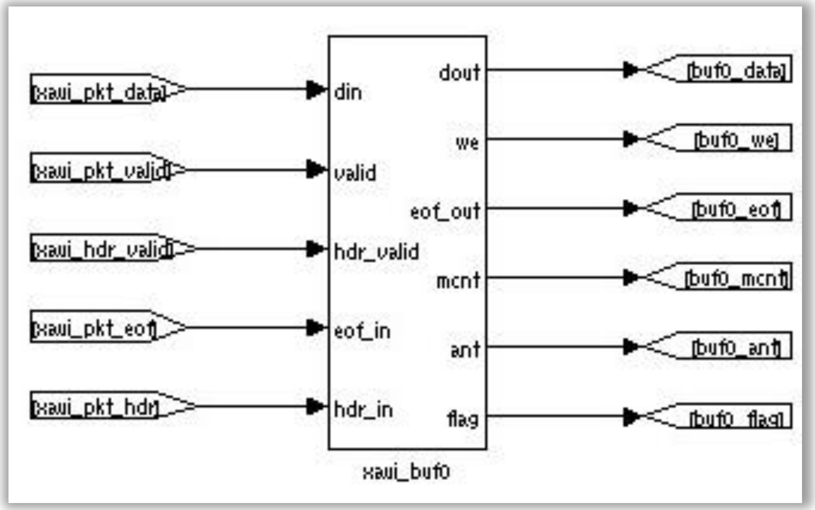


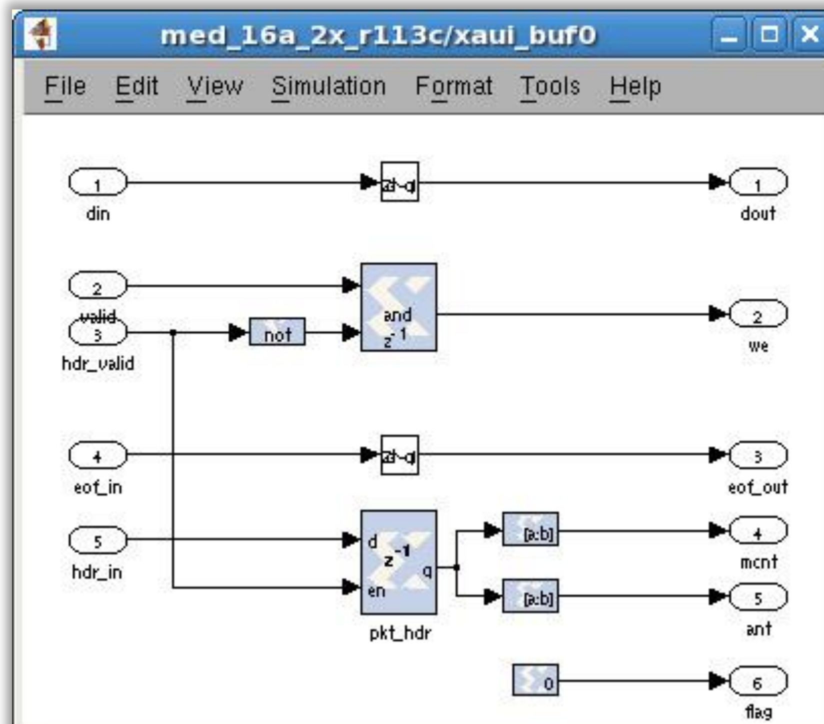
Packetizes data coming in over a XAUI interface. A packet consists of a 64 bit header (48 bits of "mcnt" and 16 bits of antenna), followed by 64 * "payload" bits.

"Mcnt" (master count) is a counter which keeps track of channel frequencies and how many packets have been transmitted since the last "mrst".

X engine

Getting data from F engine

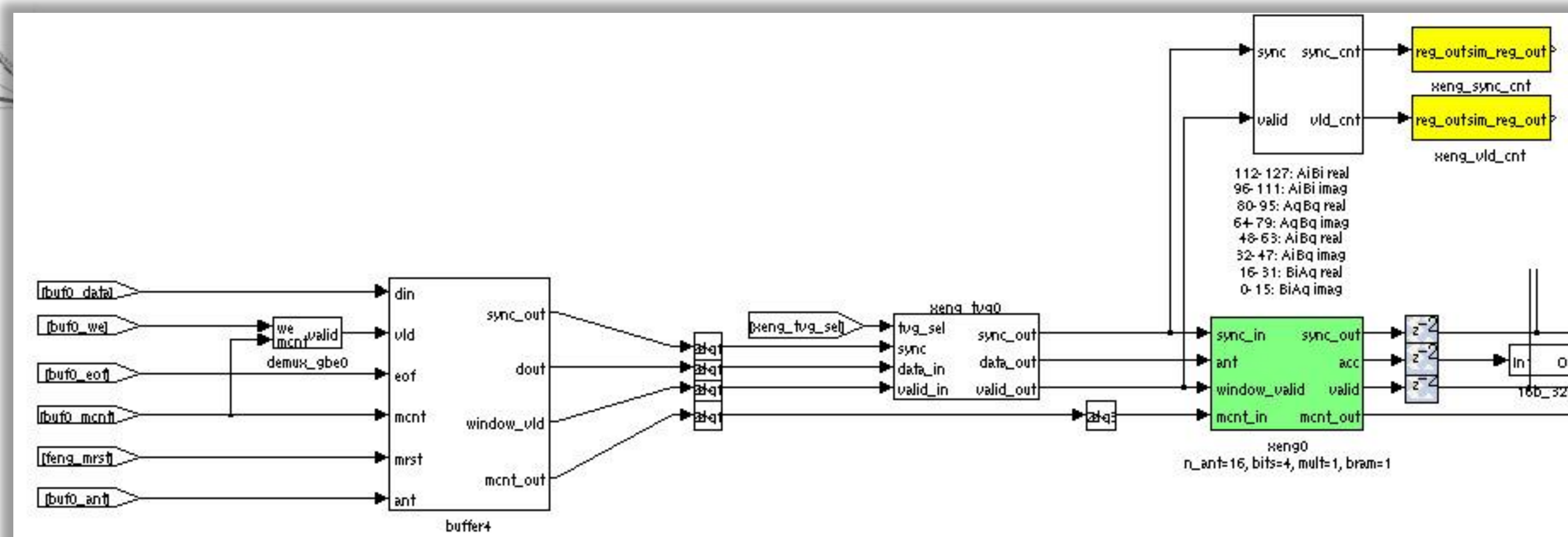




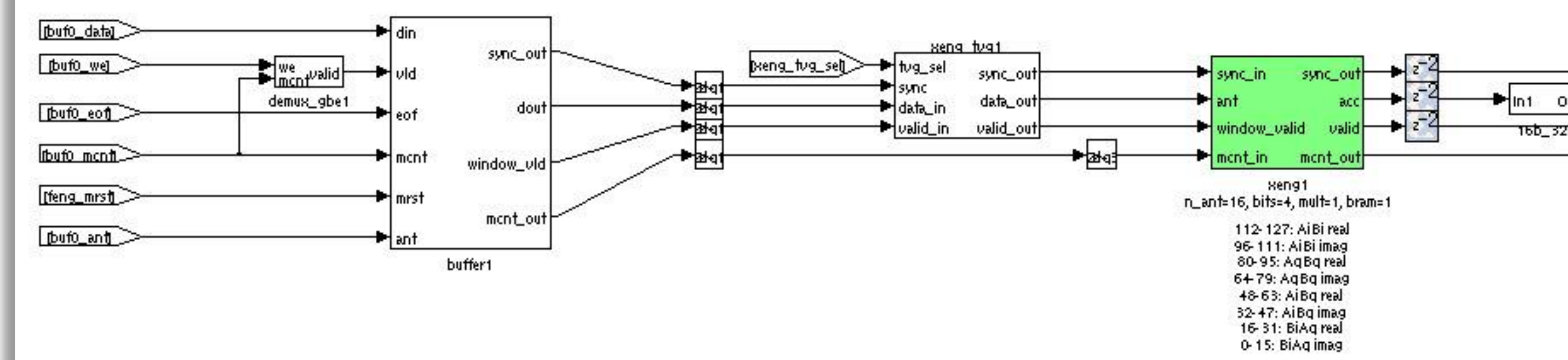
Decodes packet header in mcnt and ant

X engine

X engines

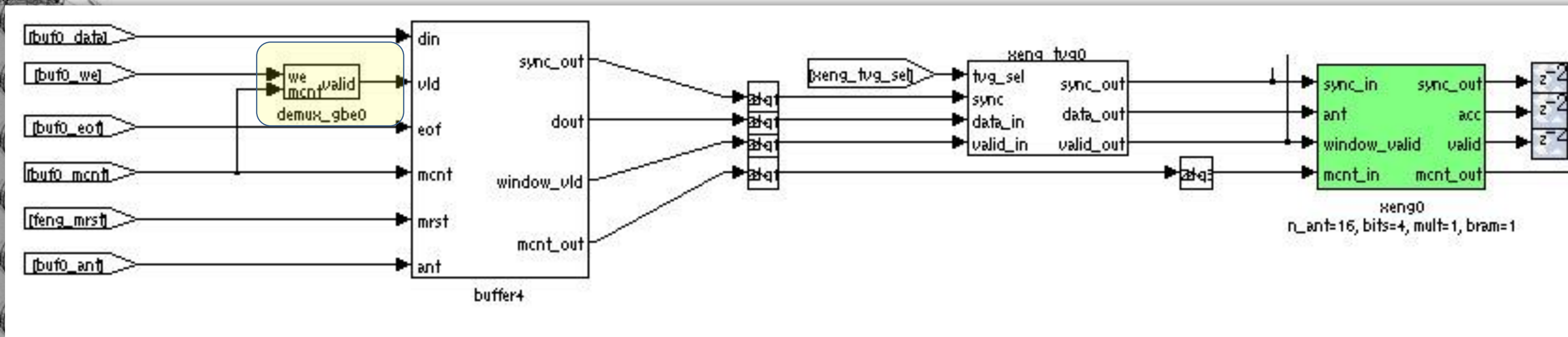


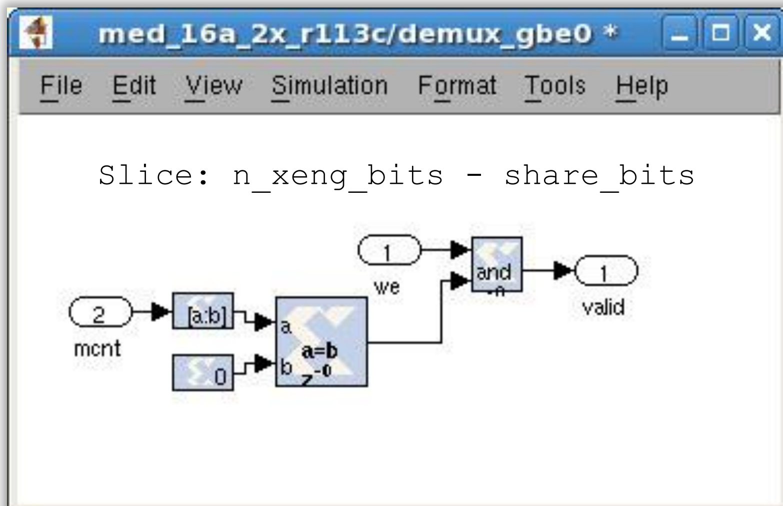
There are 2 X-engine in the design,
the coming in channel frequencies are splitted by even and odd

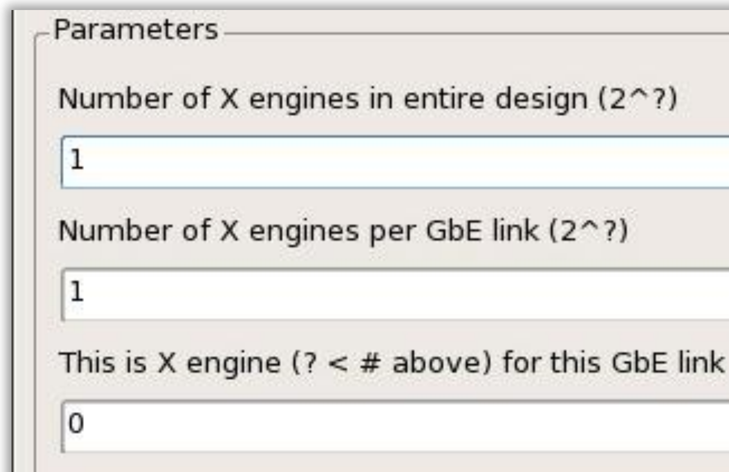


X engine

X engines



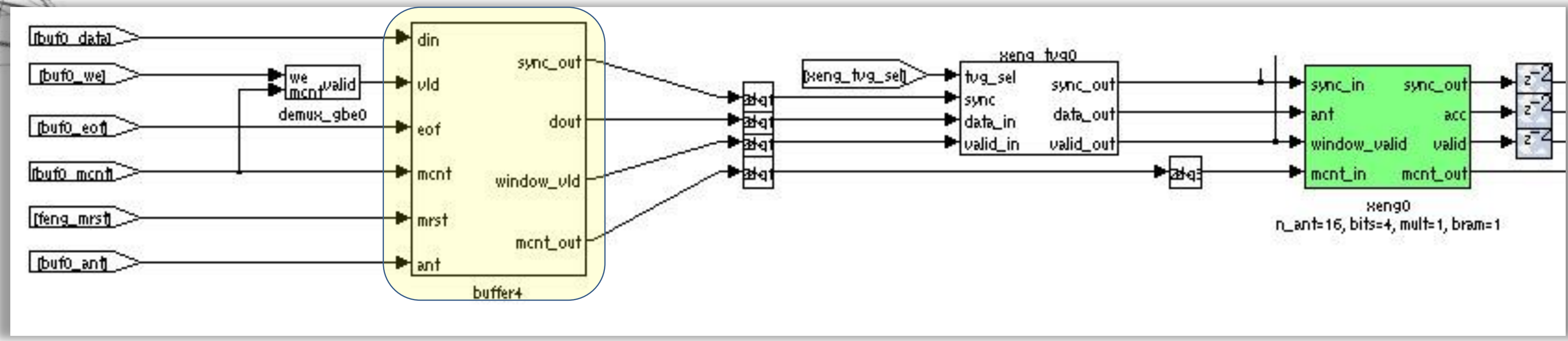


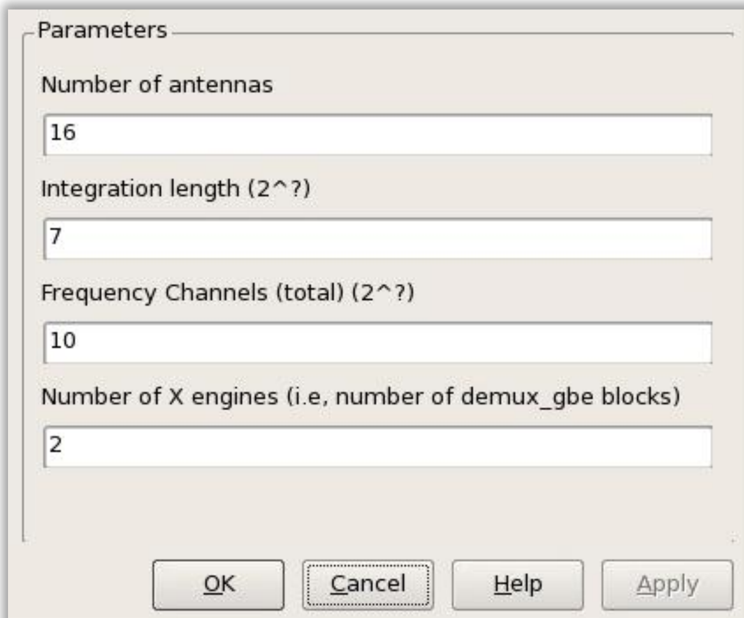


Demux gbe select even or odd frequencies (last bit of mcnt)

X engine

X engines

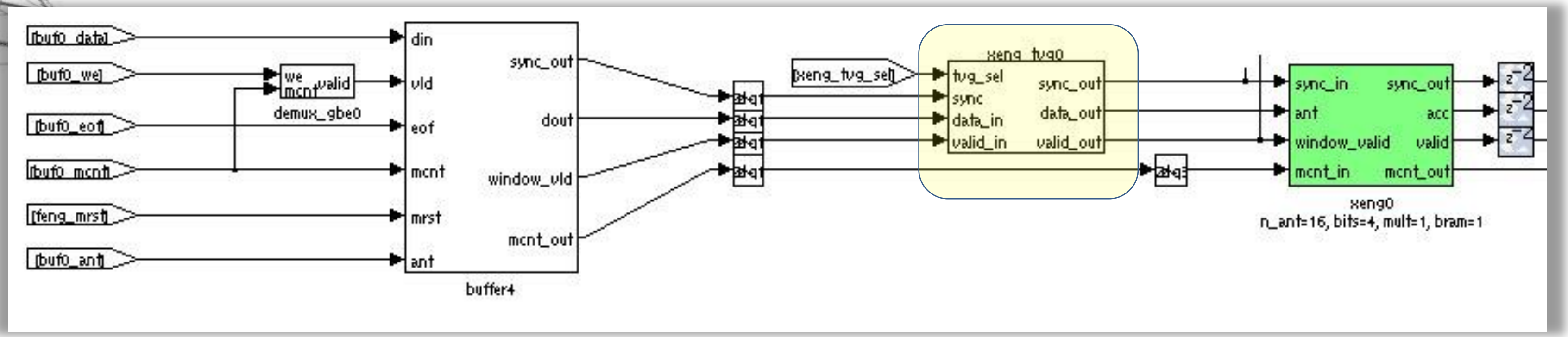


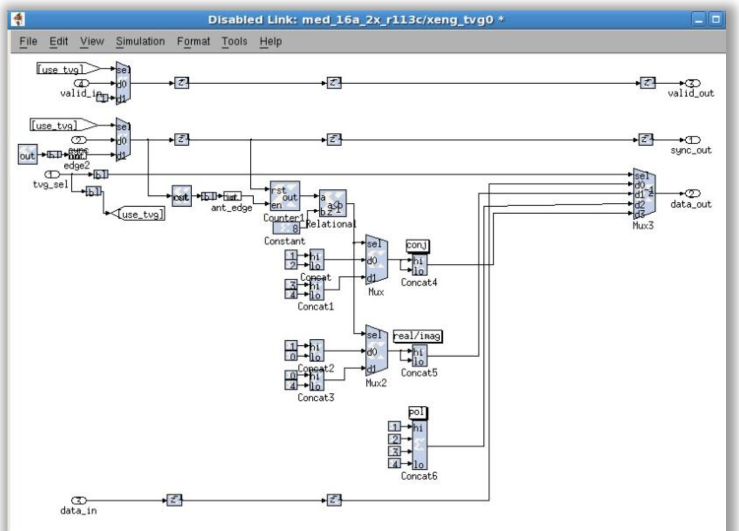


The BUFFER block collects data in a dual port ram and release it in a continuous flow to the x-engine

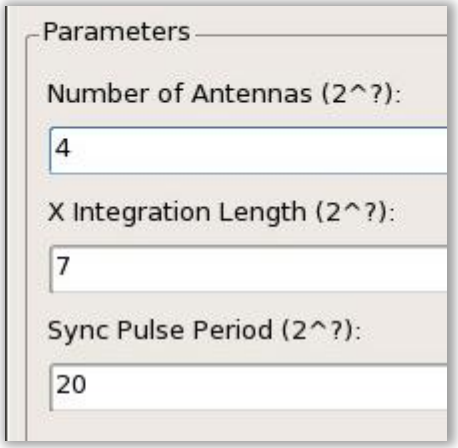
X engine

X engines





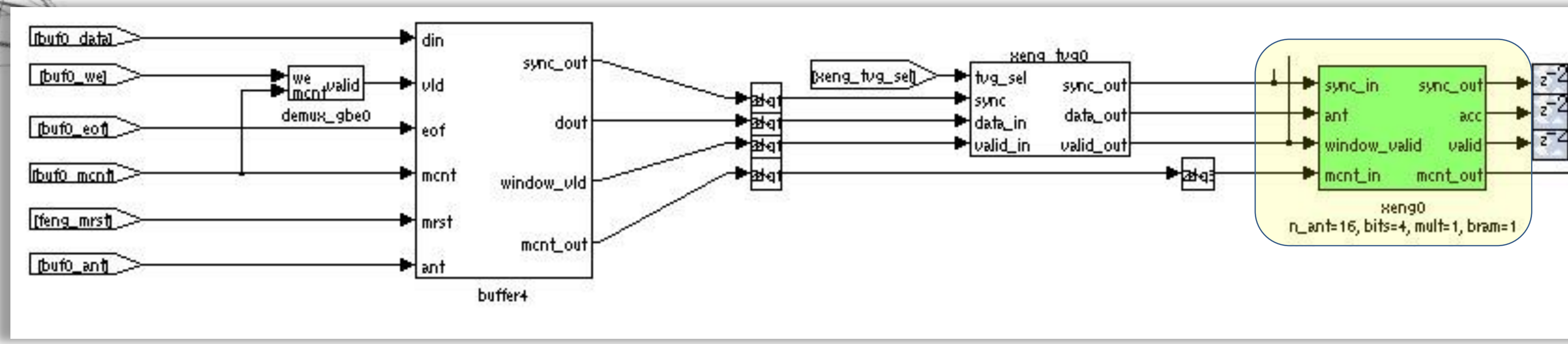
TVG block: Test Vector Generator



There is a way to simulate the data coming in to test the stand alone system

X engine

X engines

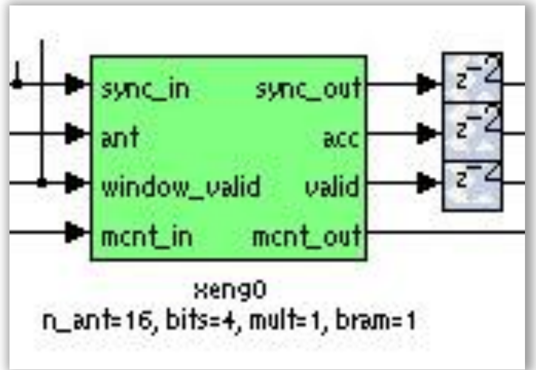


The CASPER X engine is a streaming architecture block where complex antenna data is input and accumulated products (for all cross-multiplications) are output in conjugated form. Because it is streaming with valid data expected on every clock cycle, data is logically divided into windows. These windows can either be valid (in which case the computation yields valid, outputted results) or invalid (in which case computation still occurs, but the results are ignored and not presented to the user).

(*CASPER Library Reference Manual*, last updated November 17, 2008)

X engine

CASPER Windowed X-Engine block



Data is input serially: antenna A, antenna B, antenna C etc. Each antenna's data consists of dual polarization, complex data. The bit width of each component number can be set as a parameter, n bits. The X-engine thus expects these four numbers of n bits to be concatenated into a single, unsigned number.

CASPER convention dictates that complex numbers are represented with higher bits as real and lower bits as imaginary. The top half of the input number is polarization one and the lower half polarization two.

	t_{256}	t_{255}	t_{128}	t_{127}	t_0		→
…	C1real	B1real	B1real	A1real	A1real	most_sig 4b	→
…	C1imag	B1imag	B1imag	A1imag	A1imag	4b	→
…	C2real	B2real	B2real	A2real	A2real	4b	→
…	C2imag	B2imag	B2imag	A2imag	A2imag	least_sig 4b	→

X engine

CASPER Windowed X-Engine block

The x-engine assumes that antennas are dual polarisation, and so between a pair of antennas, i and j, the correlator output gives all 4 polarisation combinations:

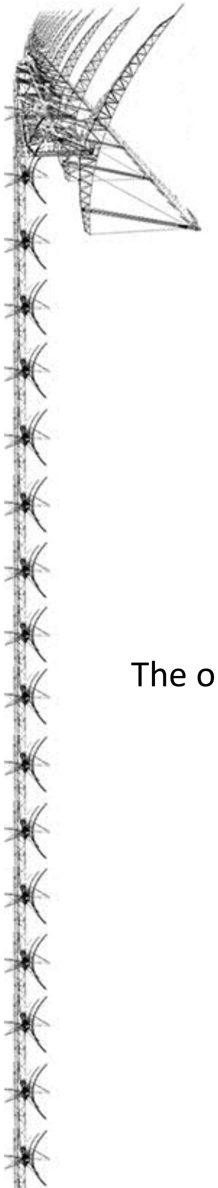
xi,xj
yi,yj
xi,yj
yi,xj

If antennas are single pol, then you can input four antennas -- a,b,c,d -- with the mapping xi -> a, yi -> b, xj -> c, yj -> d. The output is then:

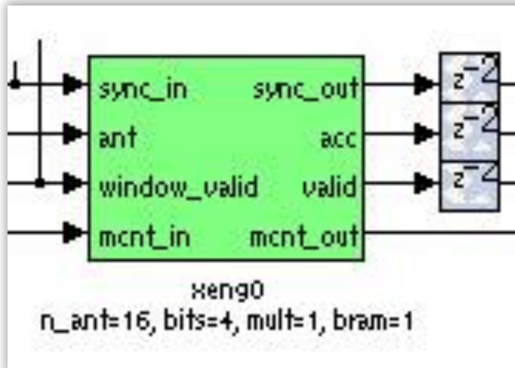
ac
bd
ad
bc

So you recover all the combinations you want. This is how the X-engine gets all the baselines with only 16 antennas. Half of the 32 are designated 'x' pol, and the other half 'y' pol.

X engine



CASPER Windowed X-Engine block

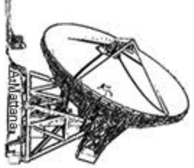


The windowed X-engine will produce num baselines = $n\_ants * \frac{n\_ants+1}{2}$ valid outputs.

The output of the X-engine configured for N antennas can be mapped into a table with $\frac{n\_ants}{2} + 1$ columns and N rows as follows (bracketed values are from previous window):

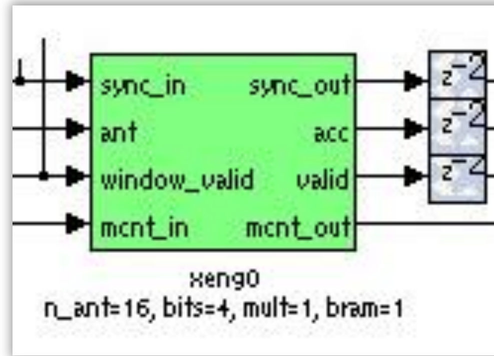
1^{st}	0×0	(0×N)	(0×(N−1))	(0×(N−2))	…	→
2^{nd}	1×1	0×1	(1×N)	(1×(N−1))	…	→
3^{rd}	2×2	1×2	0×2	(2×N)	…	→
4^{th}	3×3	2×3	1×3	0×3	…	→
5^{th}	4×4	3×4	2×4	1×4	…	→
6^{th}	5×5	4×5	3×5	2×5	…	→
…	…	…	…	…	…	→

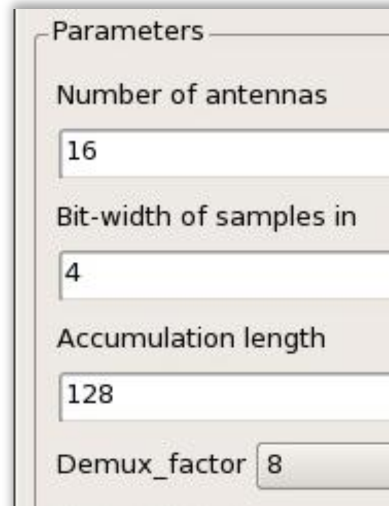
X engine



X engine output order baselines for 16 antennas read from right to left, from top to bottom

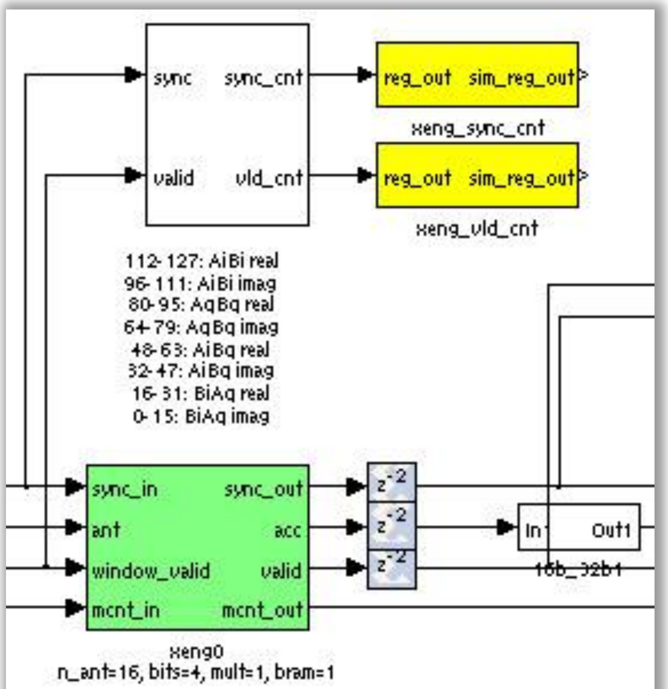
	A	B	C	D	E	F	G	H	I
1	0,0								
2	1,1	0,1							
3	2,2	1,2	0,2						
4	3,3	2,3	1,3	0,3					
5	4,4	3,4	2,4	1,4	0,4				
6	5,5	4,5	3,5	2,5	1,5	0,5			
7	6,6	5,6	4,6	3,6	2,6	1,6	0,6		
8	7,7	6,7	5,7	4,7	3,7	2,7	1,7	0,7	
9	8,8	7,8	6,8	5,8	4,8	3,8	2,8	1,8	0,8
10	9,9	8,9	7,9	6,9	5,9	4,9	3,9	2,9	1,9
11	10,10	9,10	8,10	7,10	6,10	5,10	4,10	3,10	2,10
12	11,11	10,11	9,11	8,11	7,11	6,11	5,11	4,11	3,11
13	12,12	11,12	10,12	9,12	8,12	7,12	6,12	5,12	4,12
14	13,13	12,13	11,13	10,13	9,13	8,13	7,13	6,13	5,13
15	14,14	13,14	12,14	11,14	10,14	9,14	8,14	7,14	6,14
16	15,15	14,15	13,15	12,15	11,15	10,15	9,15	8,15	7,15
17	0,0	15,0	14,0	13,0	12,0	11,0	10,0	9,0	8,0
18	1,1	0,1	15,1	14,1	13,1	12,1	11,1	10,1	9,1
19	2,2	1,2	0,2	15,2	14,2	13,2	12,2	11,2	10,2
20	3,3	2,3	1,3	0,3	15,3	14,3	13,3	12,3	11,3
21	4,4	3,4	2,4	1,4	0,4	15,4	14,4	13,4	12,4
22	5,5	4,5	3,5	2,5	1,5	0,5	15,5	14,5	13,5
23	6,6	5,6	4,6	3,6	2,6	1,6	0,6	15,6	14,6
24	7,7	6,7	5,7	4,7	3,7	2,7	1,7	0,7	15,7
25	8,8	7,8	6,8	5,8	4,8	3,8	2,8	1,8	0,8
26	9,9	8,9	7,9	6,9	5,9	4,9	3,9	2,9	1,9

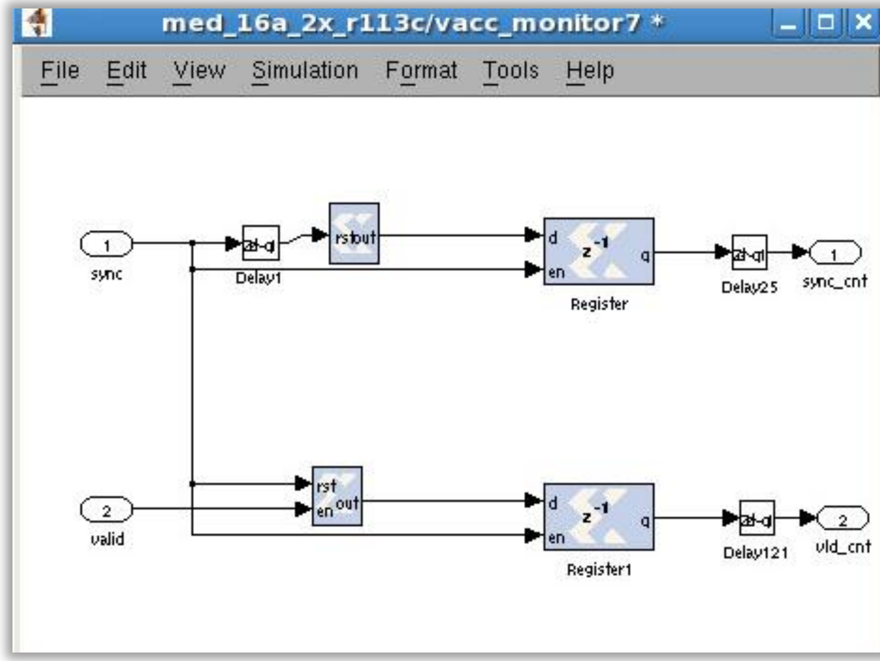




The green cells are the only ones that the xeng block actually outputs because there is a descramble block inside there that removes the duplicated "red" baselines

X engines

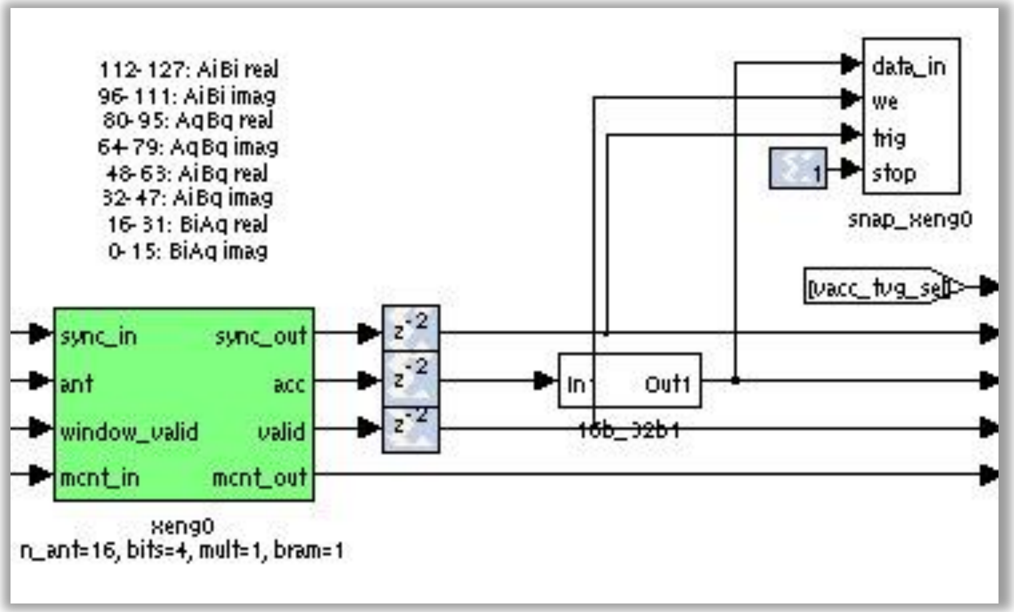




Sync and Valid signals are counted and registered

X engine

X engines

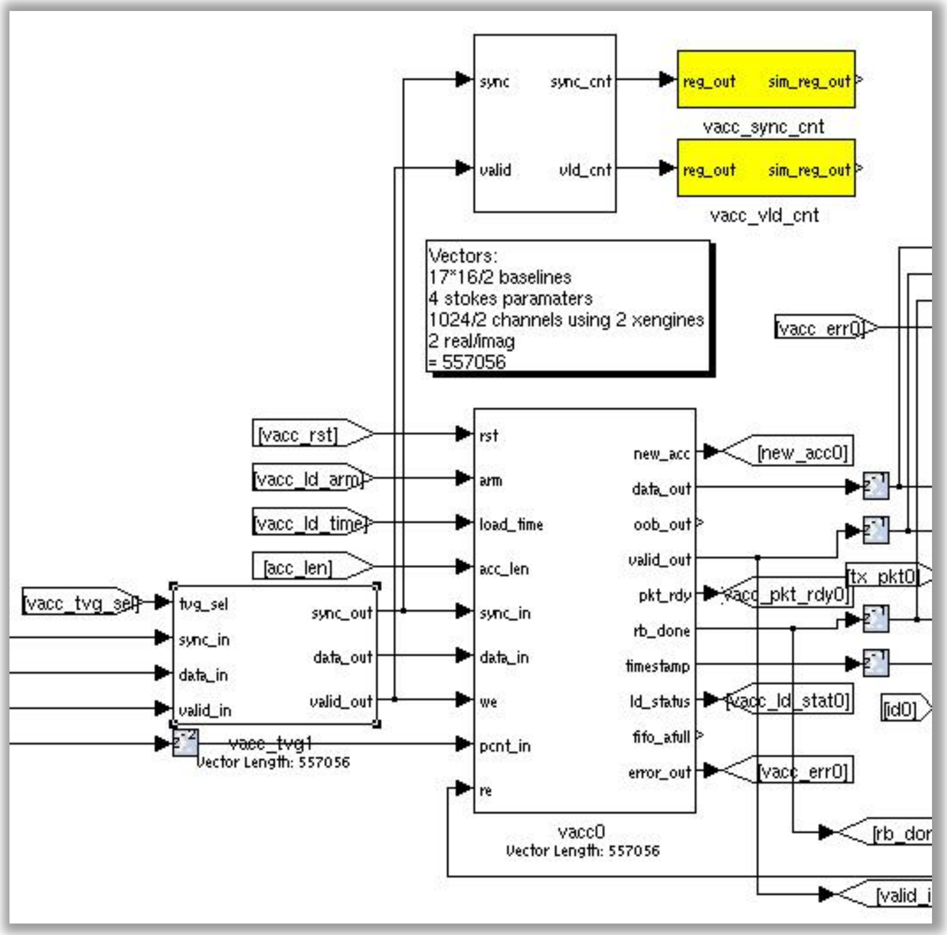


SNAP_XENG0

There is also a snap block which allows to read from bram a snap of the computed baselines

X engine

Vector Accumulator and Packetizer



VECTORS:

16*17/2 (baselines) *
1024/2 (channels) *
4 (stokes parameter) *
2 (width re/im)

= 557056 size

using QDR0 and QDR1

X engine

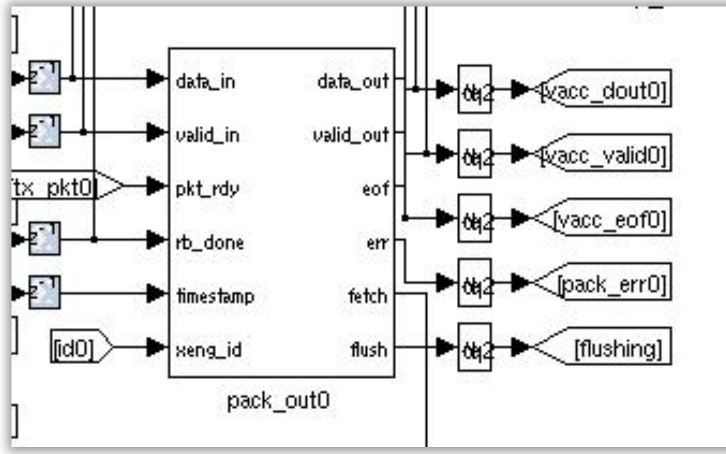
Vector Accumulator and Packetizer

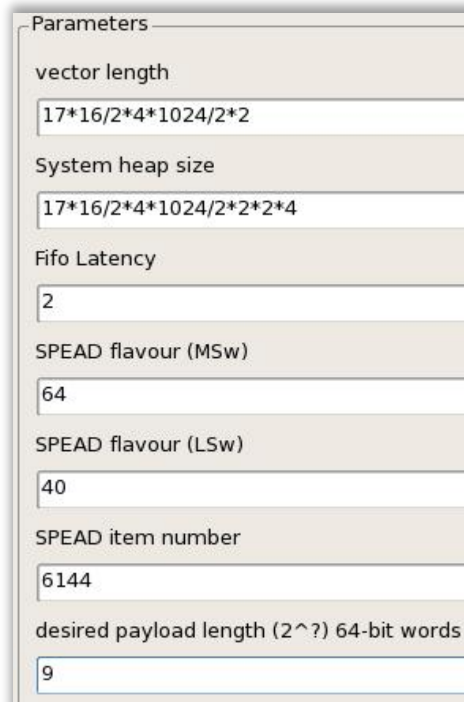
This block generates packets from a datastream.

Packets are created with a fixed header followed by a user-specified number of 64-bit options.

Requires a 64-bit data stream.

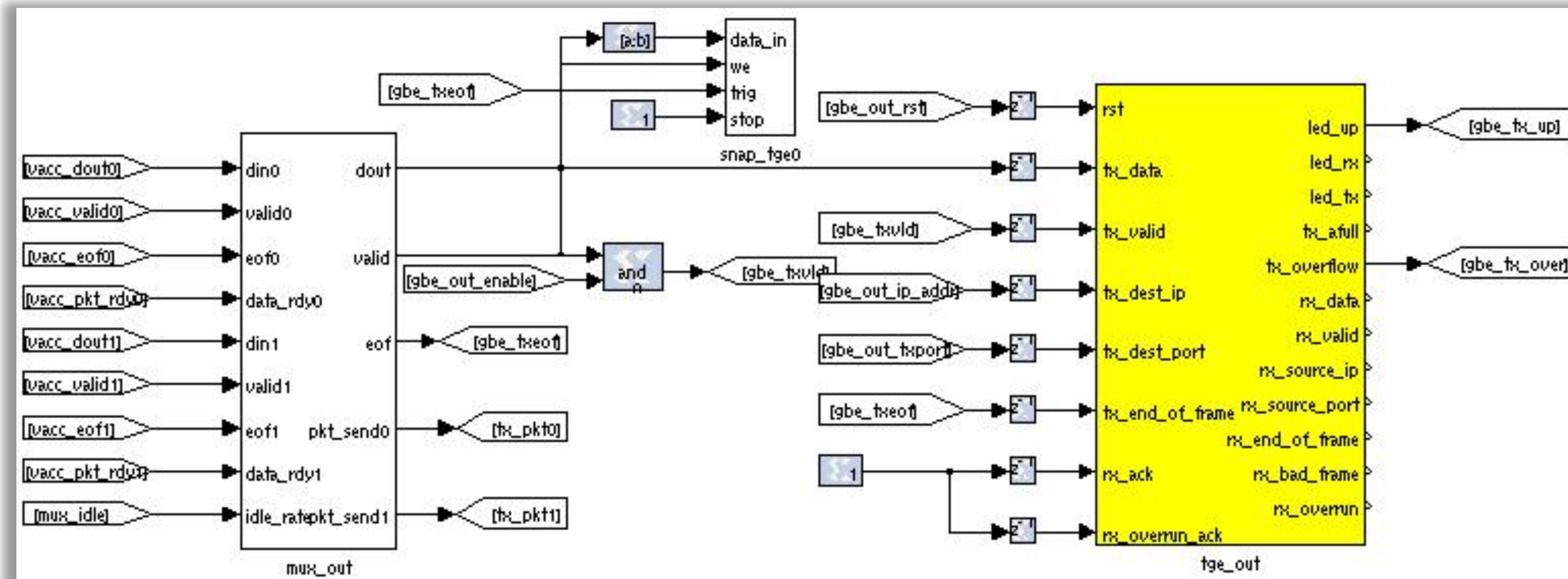
```
Init:

vector_bits = ceil(log2(vector_len));
pkt_len=2^(pkt_bits);
```





X engine

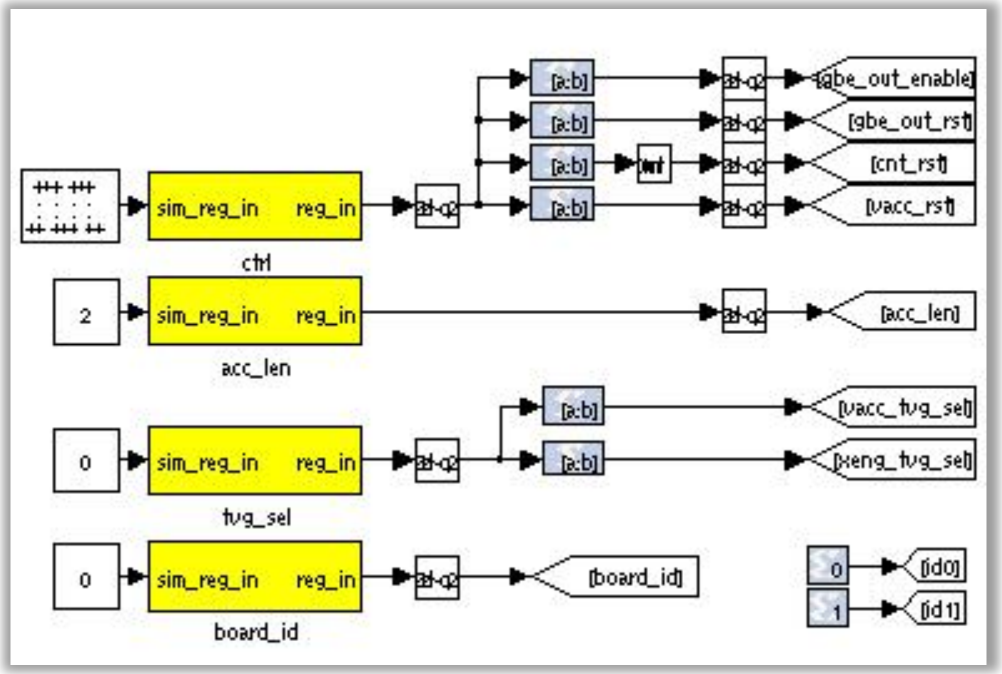
Sending data over 10GbE of the 2 X-Engine accumulations



Mux_Out is simply the semaphore for the packet traffic, the tx_pkt0 and tx_pkt1 signals are the green light for the vector accumulator pack_outs
In case of collision vacc_dout0 has priority

X engine

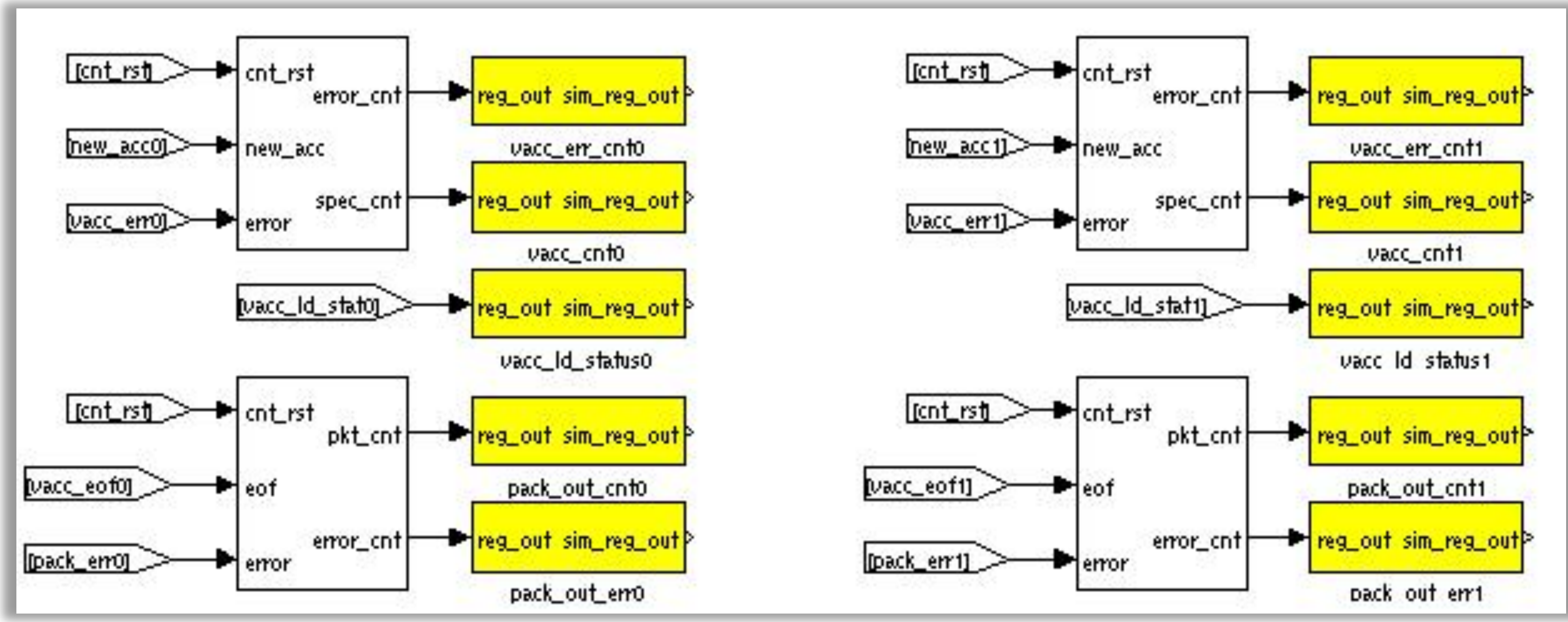
Main Control Signals and Registers



INPUT REGISTERS

X engine

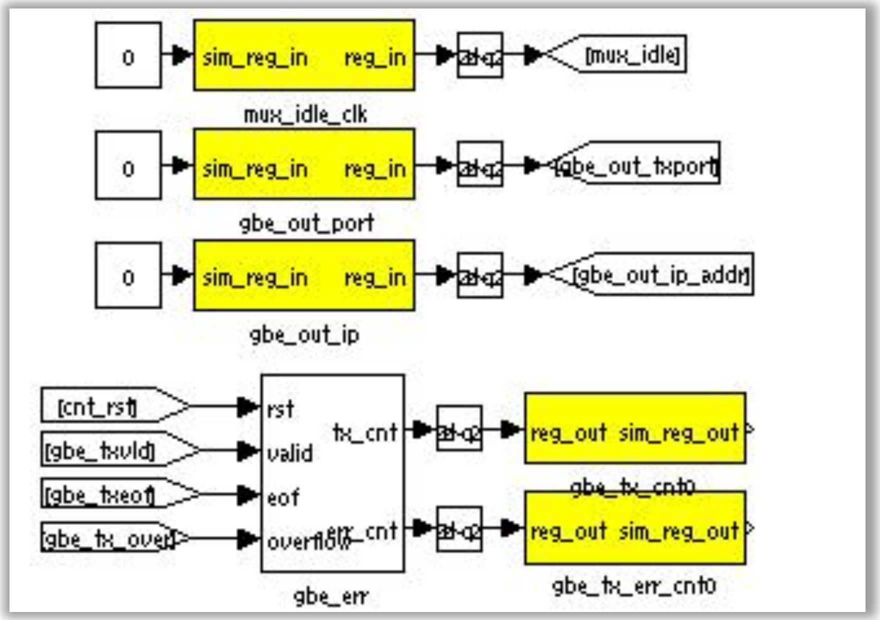
Main Control Signals and Registers

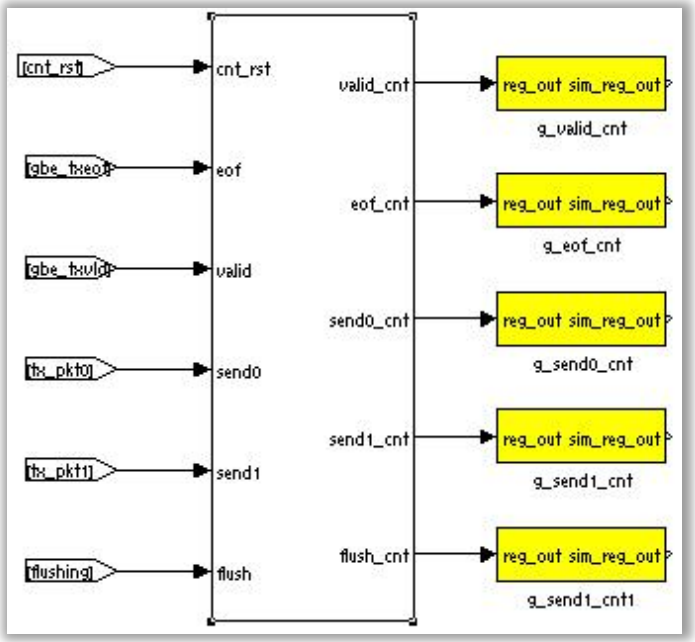


Keeping tracks of packets

X engine

Main Control Signals and Registers

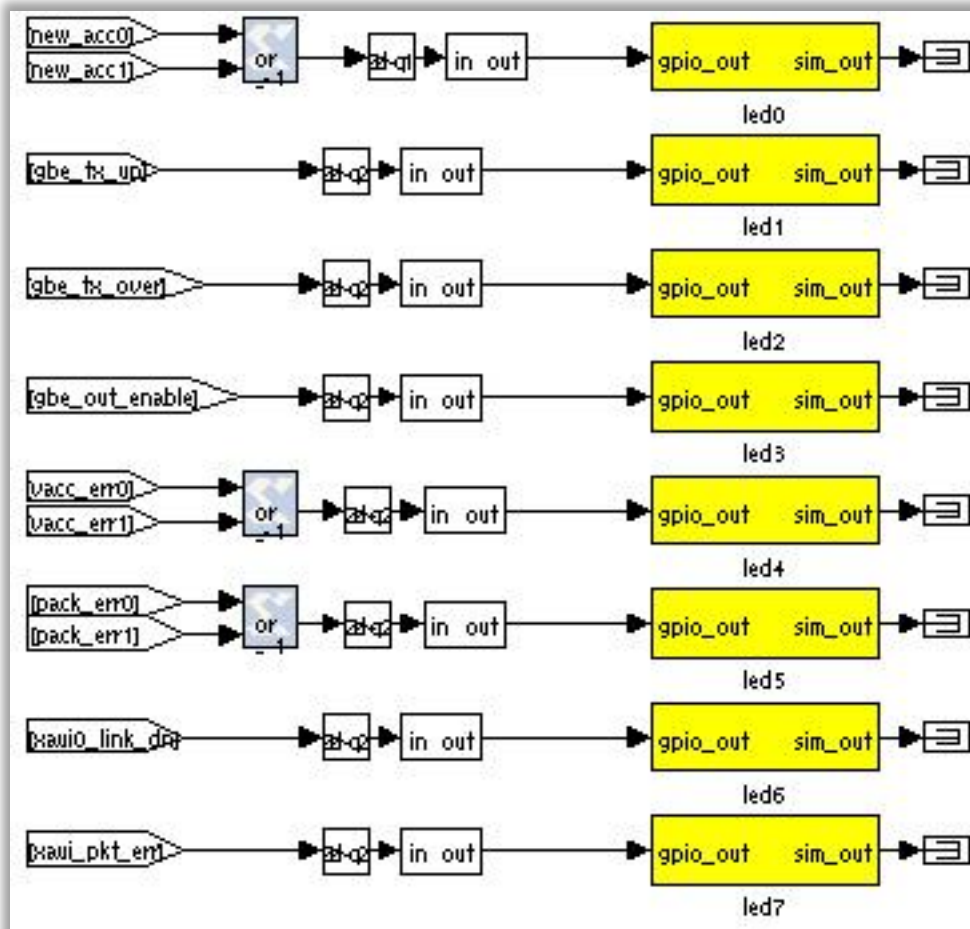




10 GbE configurations and stats

X engine

Main Control Signals and Registers



ROACH'S LEDS

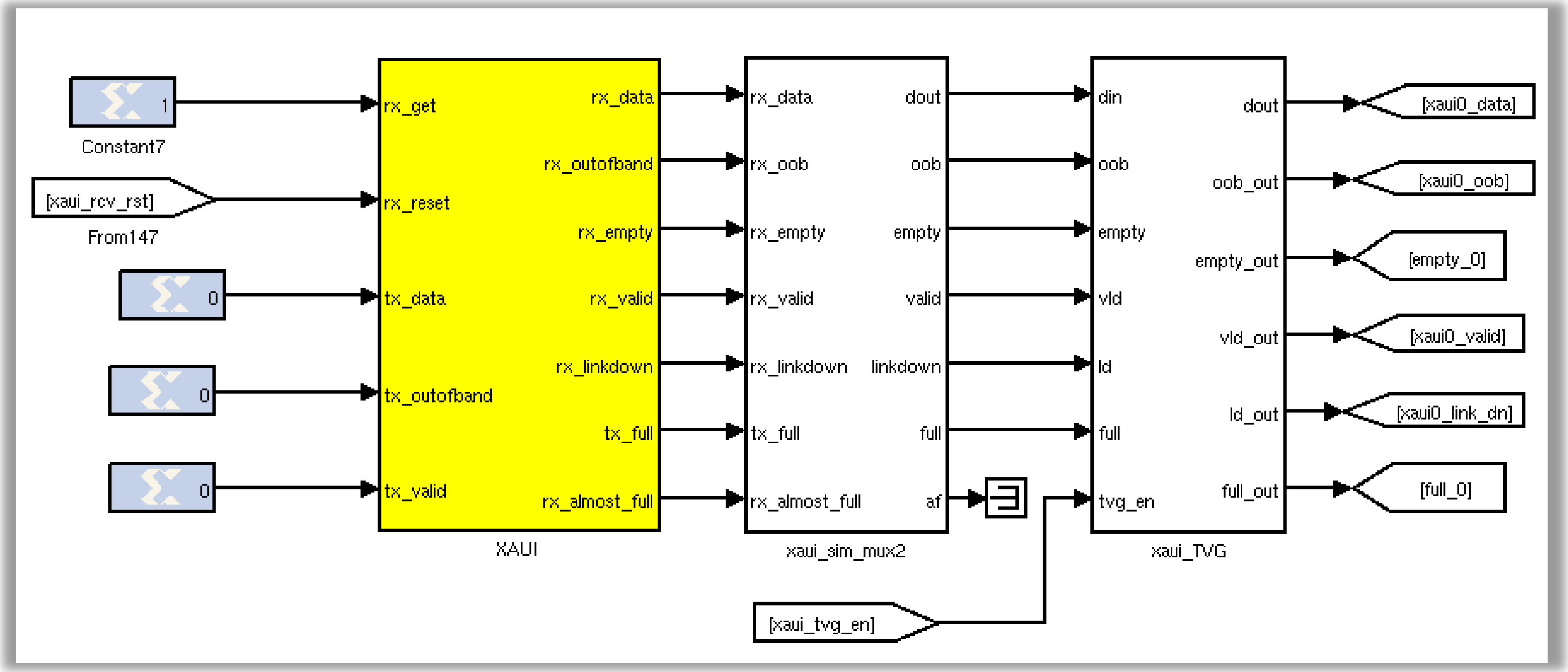
X engine

The S-engine (Spatial FFT)

Roach-3

S engine

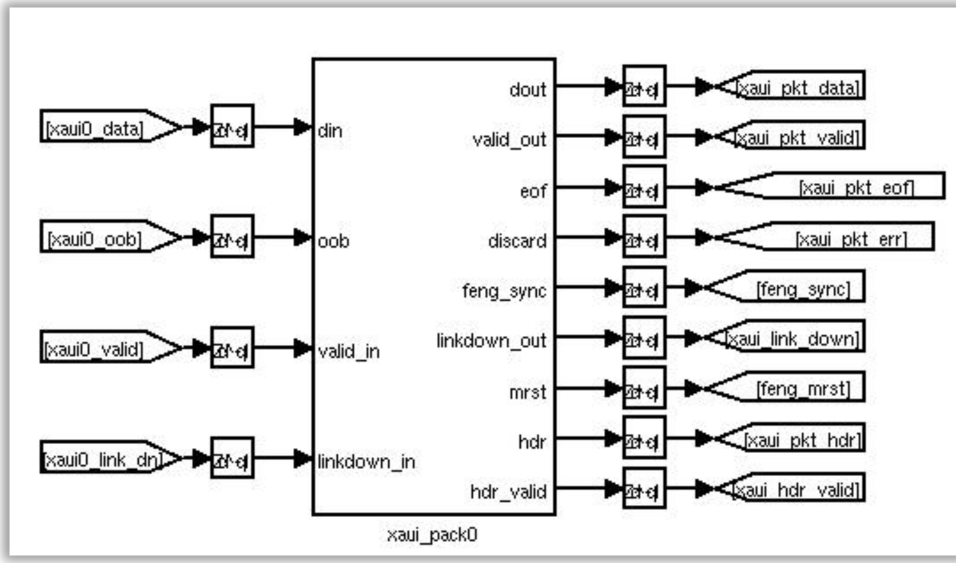
Getting data from F engine

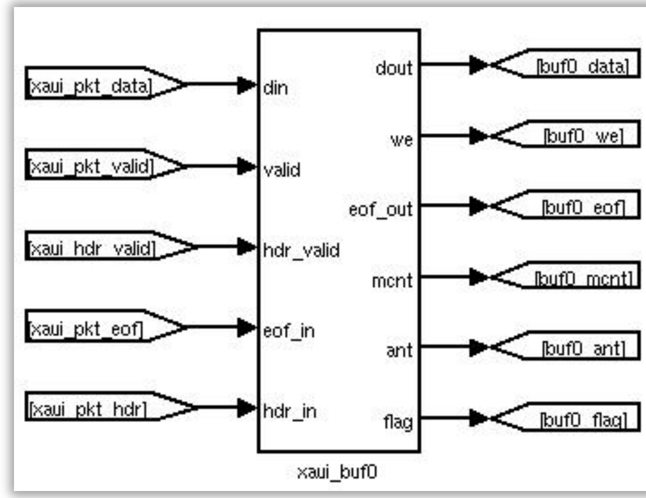


Accept 32 data words (64 bits each) from roach, plus a 1 word header (16 words contains 128 samples of a single subband of each antenna). Input data can be also simulated commanding a register.

S engine

Getting data from F engine



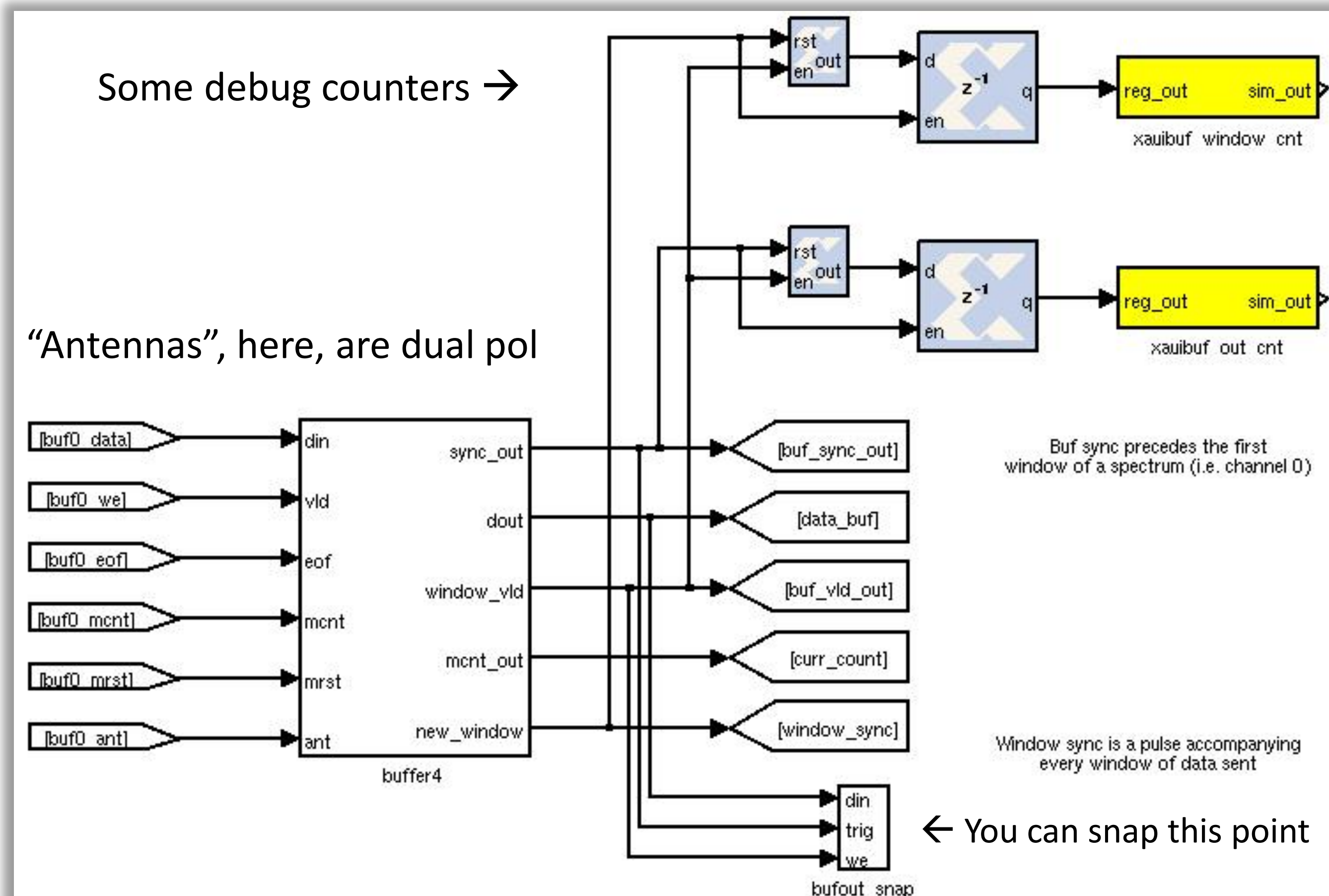


4 words containing 32 antennas of a single frequency in order.
128 x 4 = 512 words contain 128 ordered antenna for a single frequency.

We send in blocks of 2^5, indexing with an “antenna” number

S engine

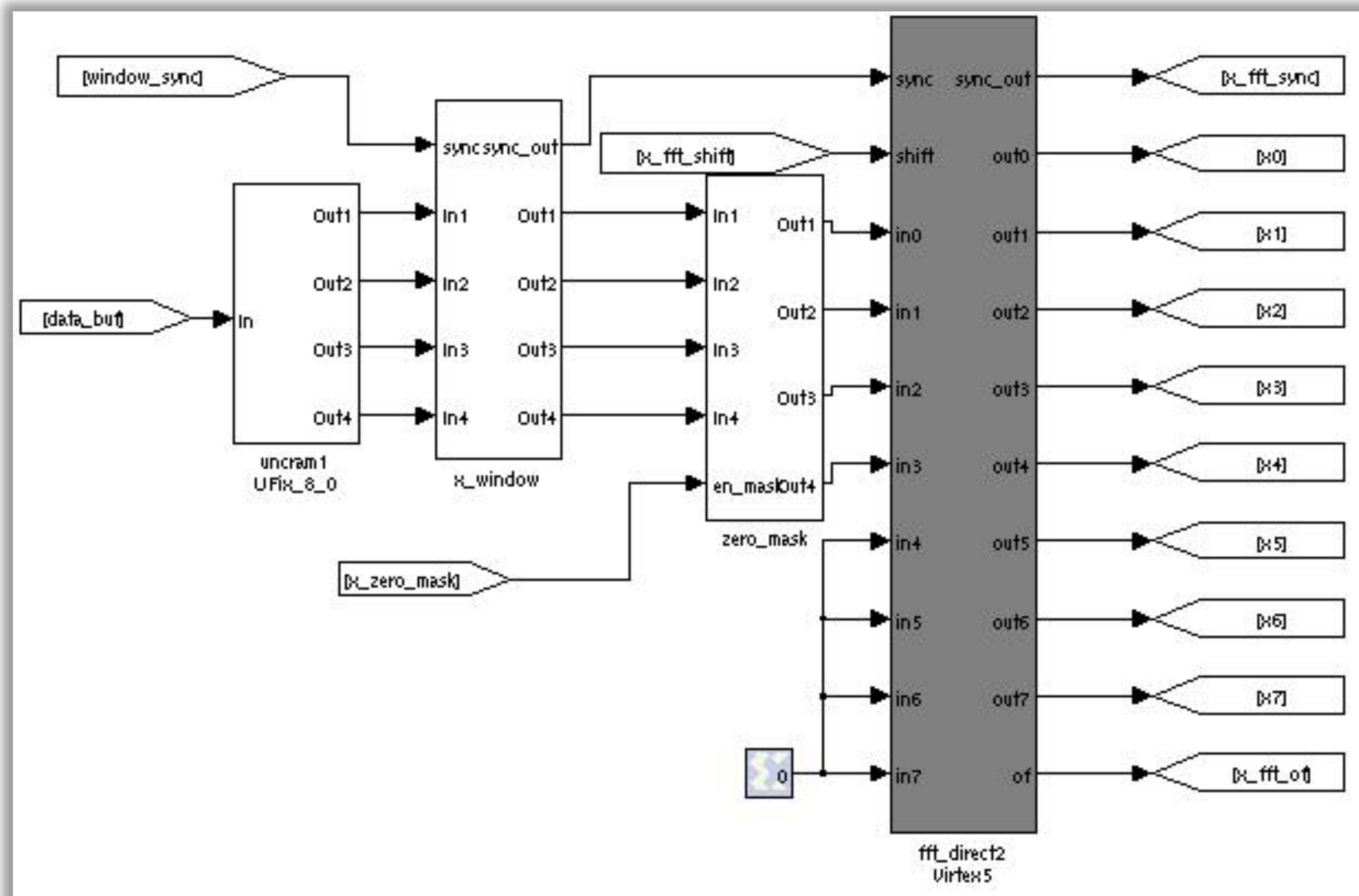
Buffering and preparing data for Spatial FFT



The Spatial FFT works in dual pol, we will see later on how to handle a single pol

S engine

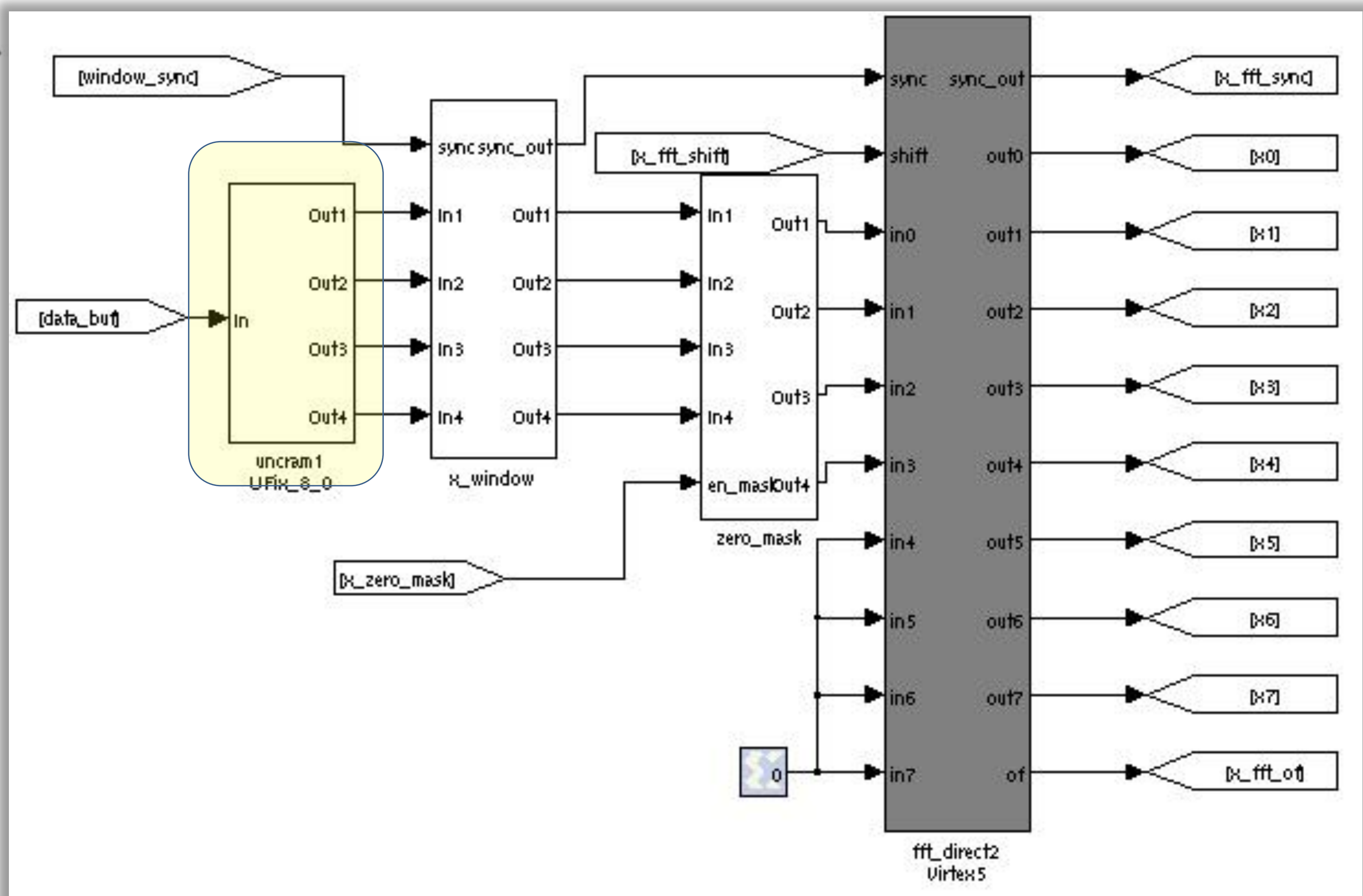
Spatial FFT - Stage 1



4 antennas per clock → 8 clocks for a complete set

S engine

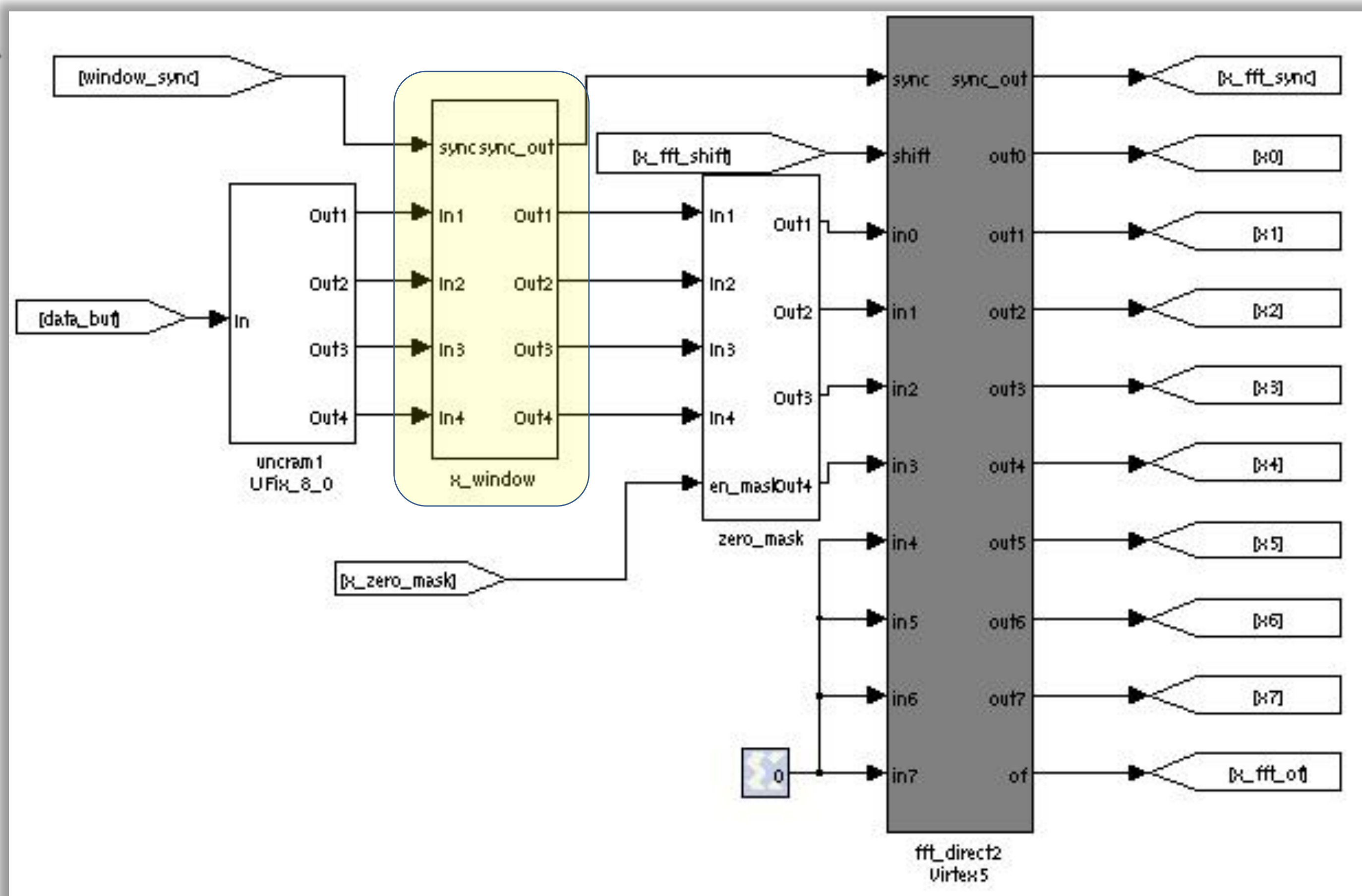
Spatial FFT - Stage 1



Each antenna subband processed separately

S engine

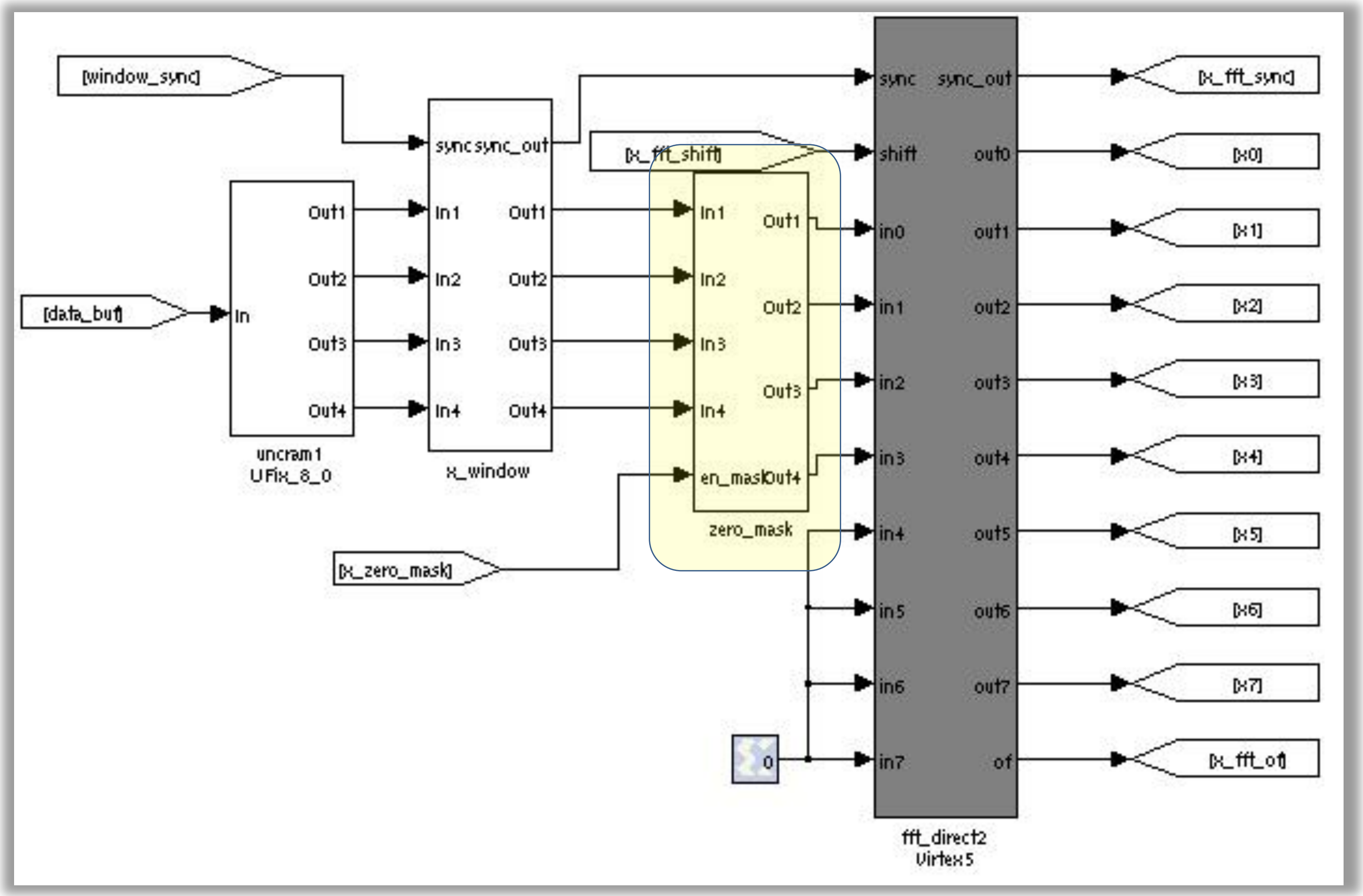
Spatial FFT - Stage 1



Any kind of window (registers preloaded) applied

S engine

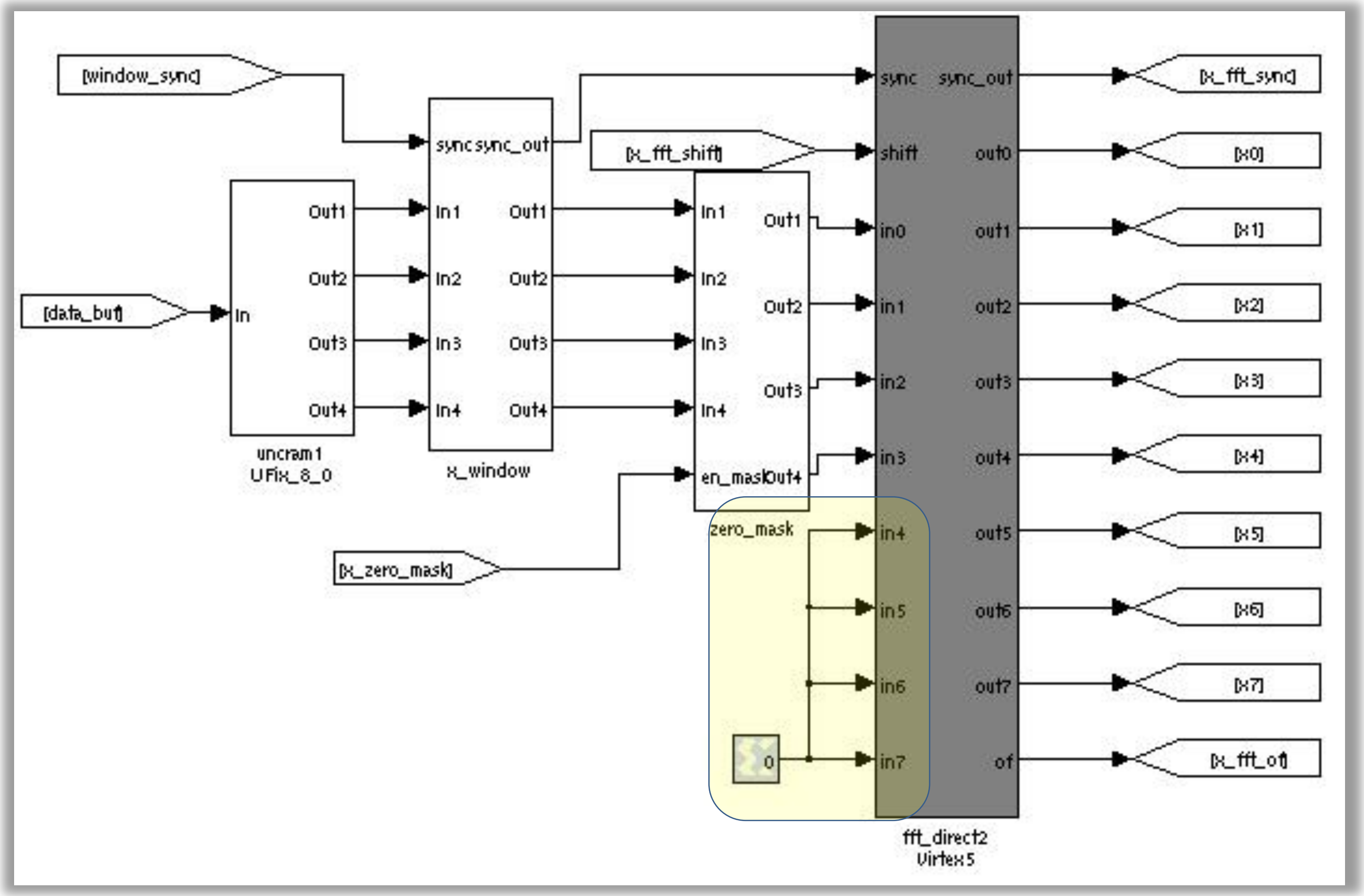
Spatial FFT - Stage 1



Subband masks
(if needed, useful either for debug and to filter interference)

S engine

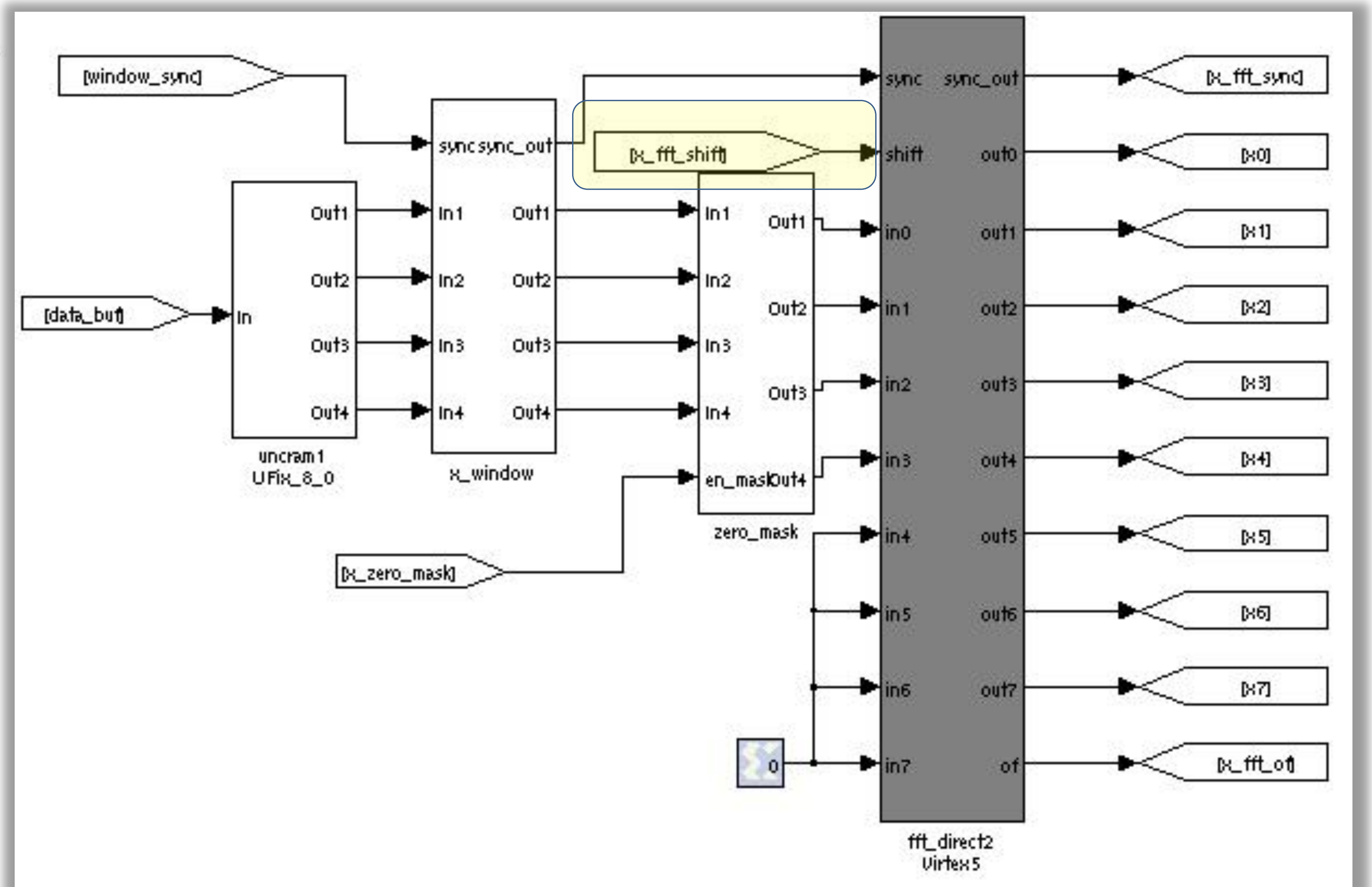
Spatial FFT - Stage 1



Zero padding of the second polarization

S engine

Spatial FFT - Stage 1

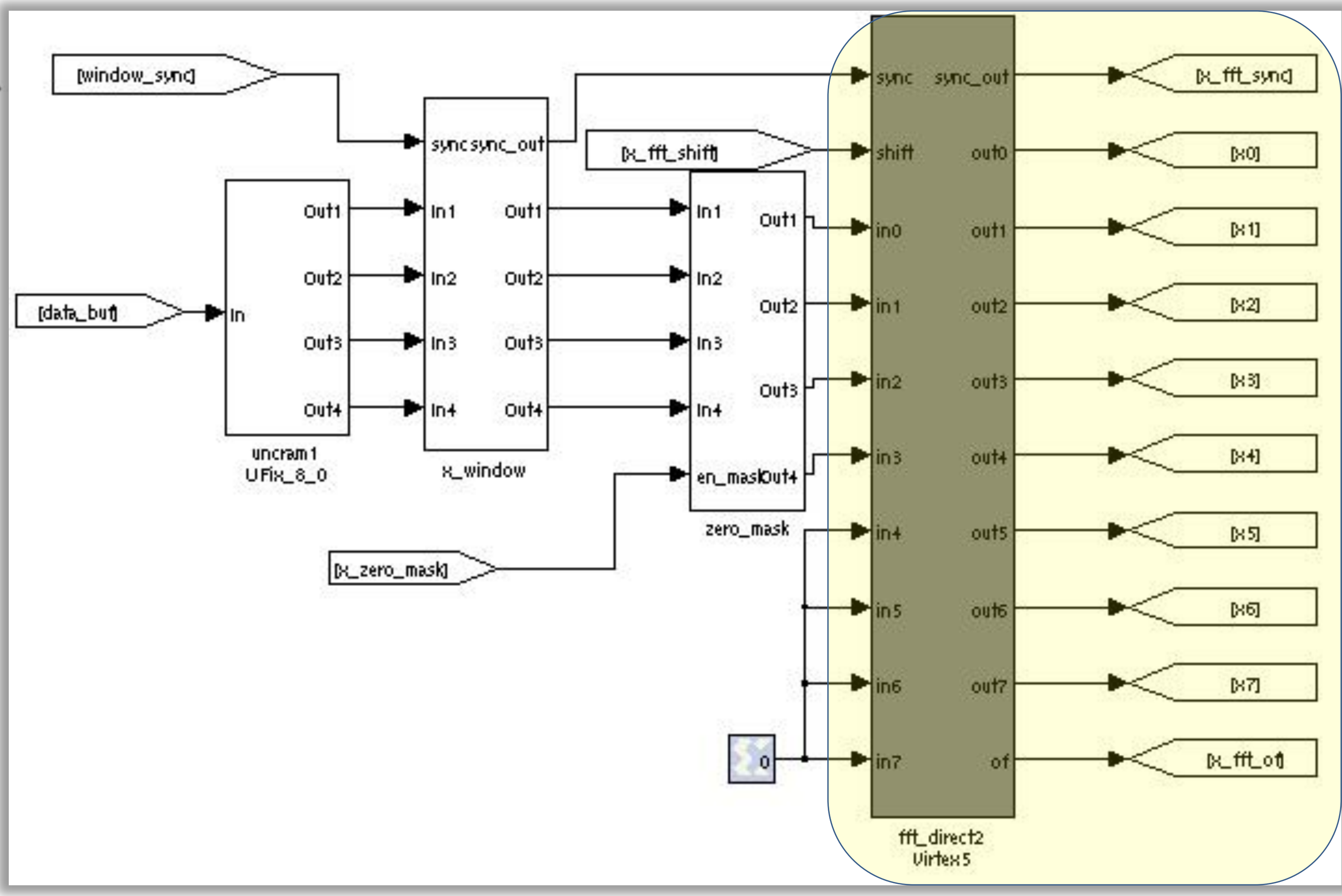


Keep control of overflows

(Sets the shifting schedule through the FFT to prevent overflow. Bit 0 specifies the behavior of stage 0, bit 1 of stage 1, and so on. If a stage is set to shift (with bit = 1), then every sample is divided by 2 at the output of that stage.)

S engine

Spatial FFT - Stage 1

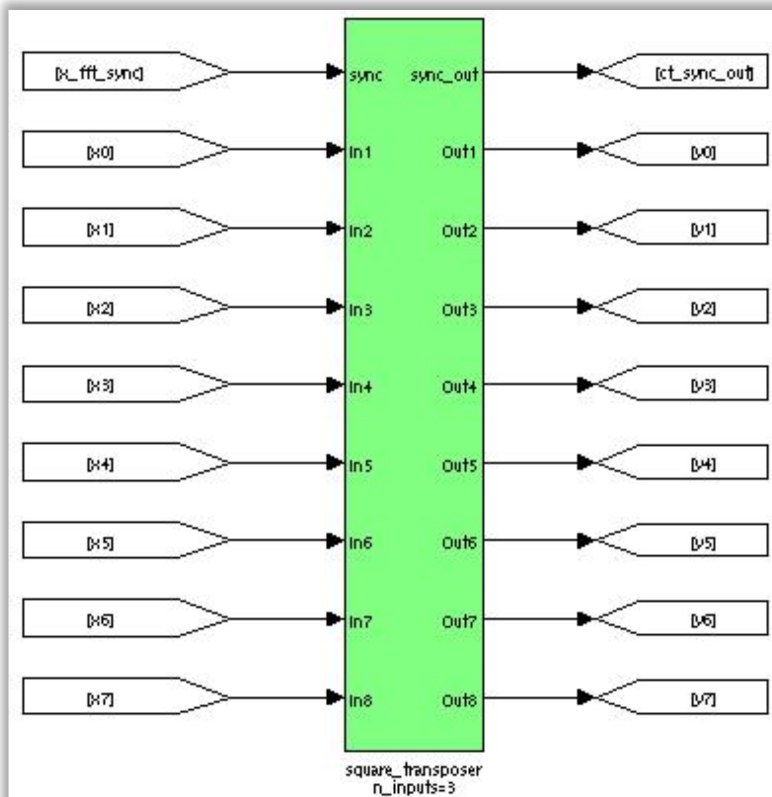


Computes the Fast Fourier Transform with 2^N channels for time samples presented 2^P at a time in parallel.

Outputs 8 beamlets per clock and it takes 8 clocks for all rows.

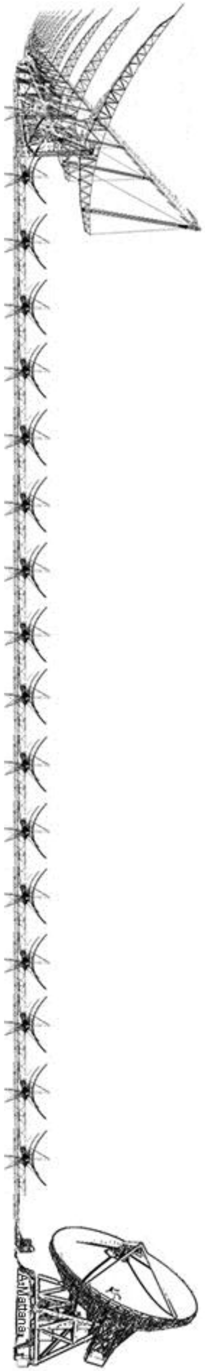
S engine

Spatial FFT - Tranpose



Transpose a $2^3 \times 2^3$ Matrix

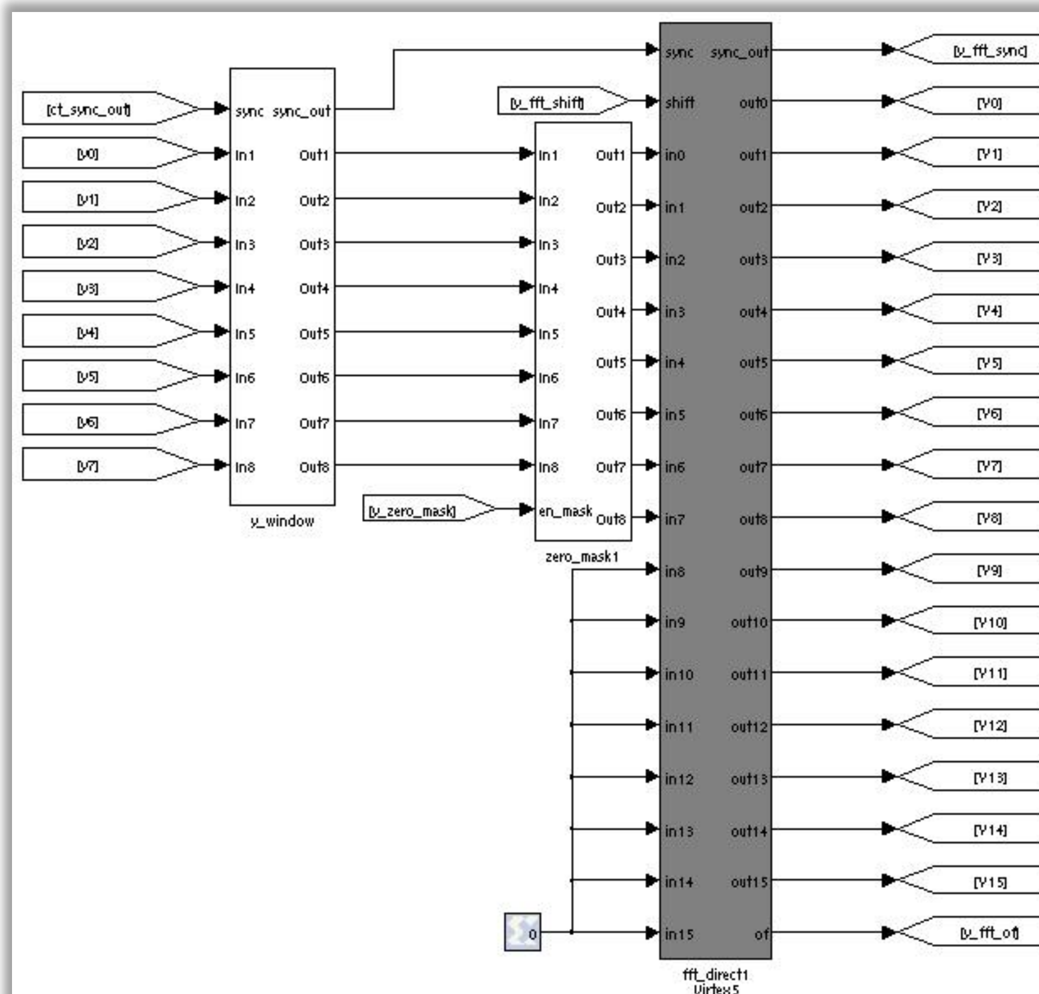
S engine



Spatial FFT – 2D FFT stage

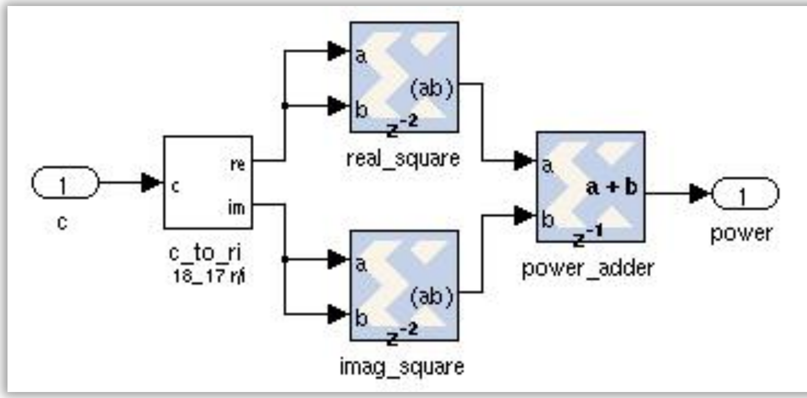
The windowed FFT along the second dimension is similar to the first one

It's a 16 streams x 8 clocks deep



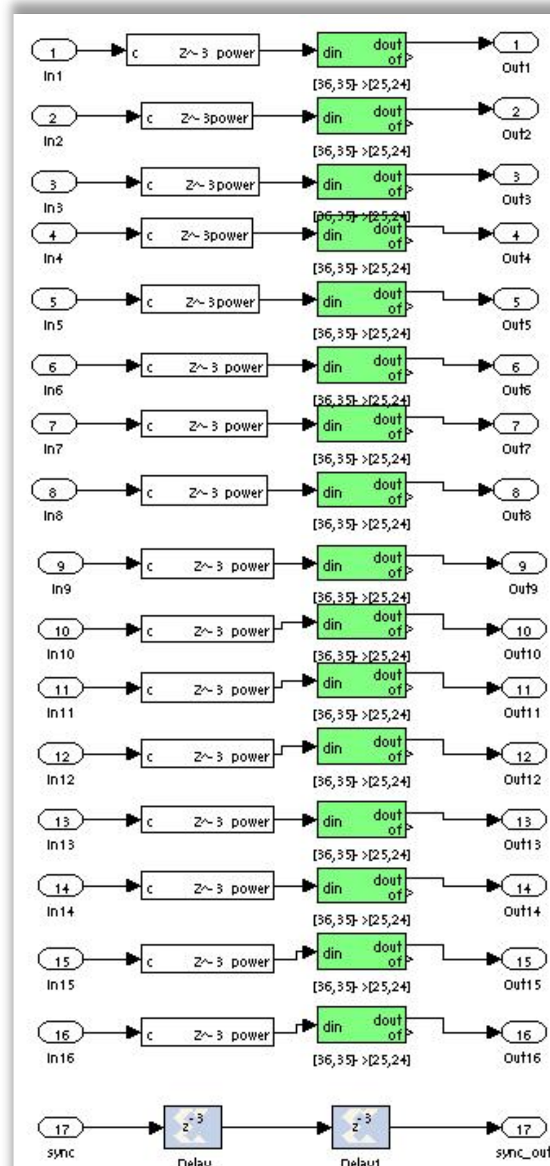
S engine

Spatial FFT – Power spectrum



Squares real and imaginary components of complex input and adds them.

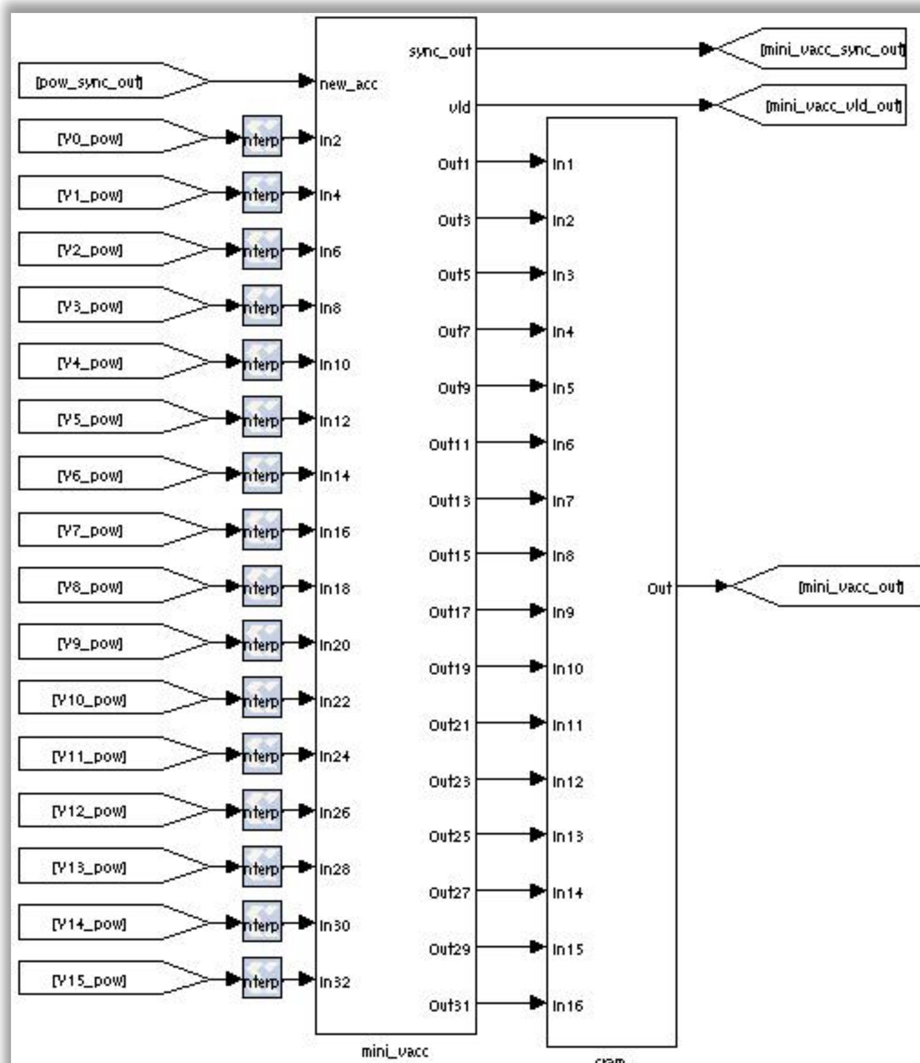
This is done for each beam, and then, there is a cast from 36.35 to 25.24.



S engine

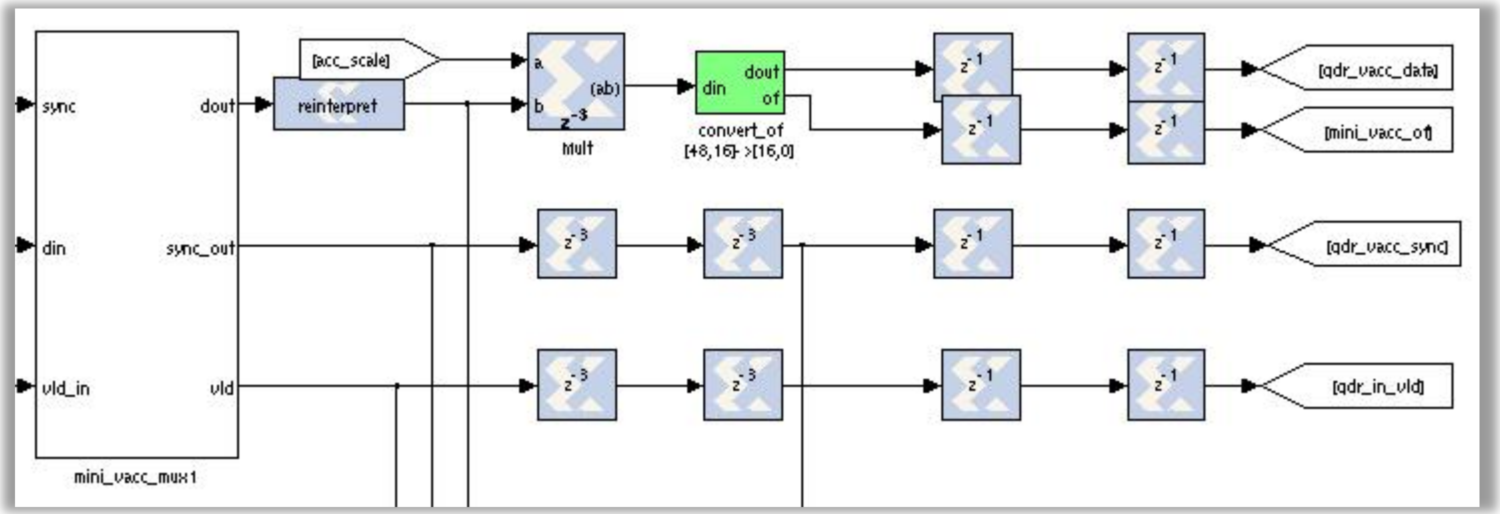
Spatial FFT – Accumulation

Data reinterpreted as unsigned 32bit is accumulated by 128, and then concatenated



S engine

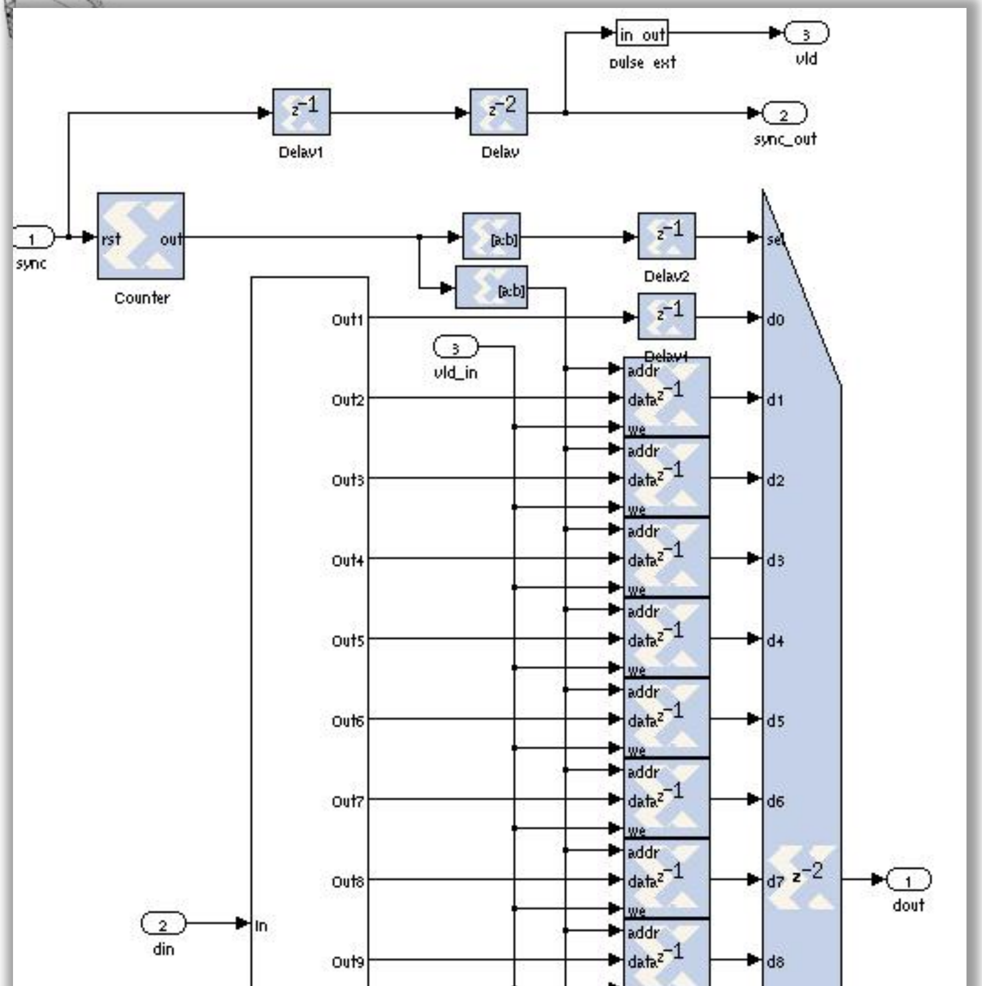
Spatial FFT – Integration



Data is accumulated, serialized, and then quantized

S engine

Spatial FFT – Parallel to serial



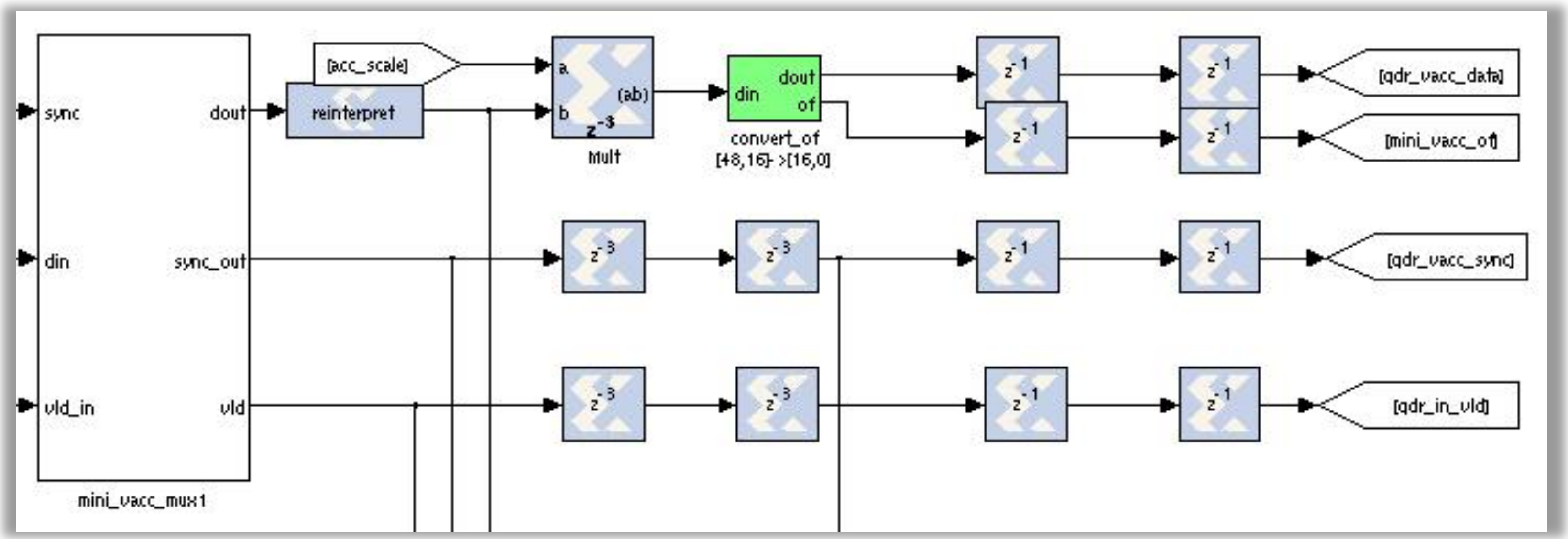
(…going into the vector accumulator…)

Serialization is done registering data into single port RAMs and then a MUX loops around addresses.

Previous accumalations guarantees that there is enough time to serialize before overlap.

S engine

Spatial FFT – Quantization

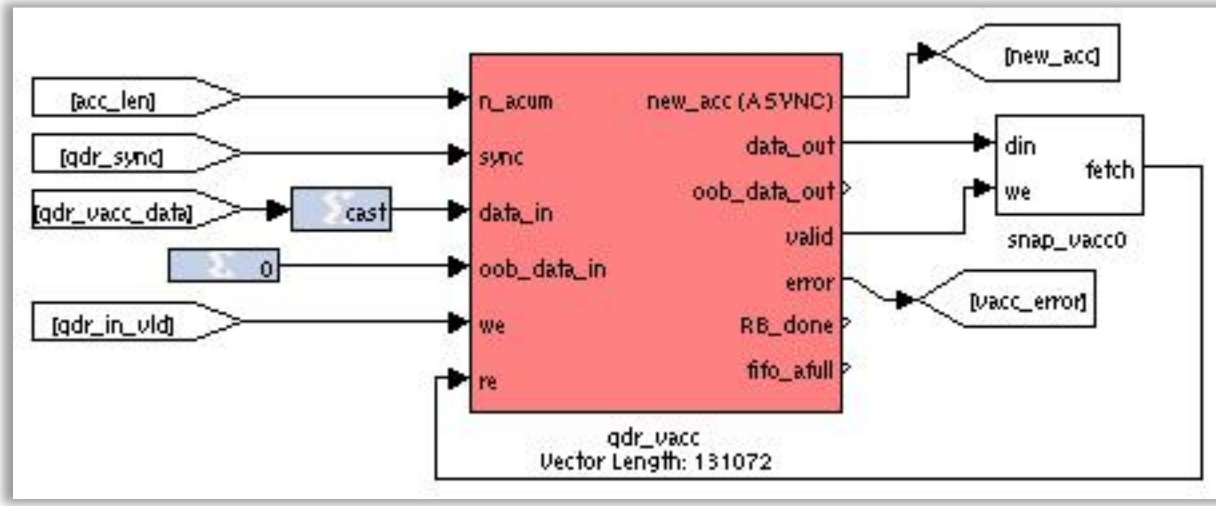


You can apply an accumulation factor scale by programming a register (default 1)

The cast shape is: 48.16 → 16.0

S engine

Spatial FFT – QDR, data ready for processing



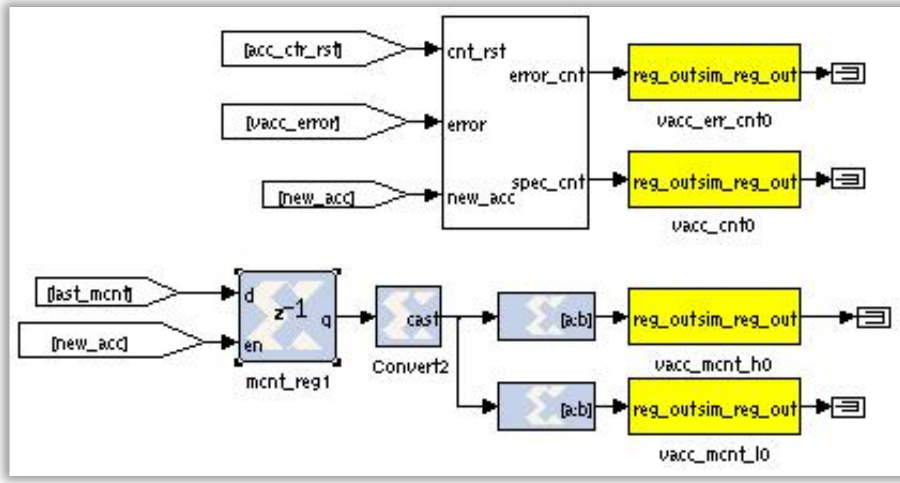
Data is reinterpreted as unsigned 32.0 to fit the requirements of the ROACH corner turn memory (QDR, CASPER stdlib)

The vector length (# cycles) is set to 1024*32*4 = 131072

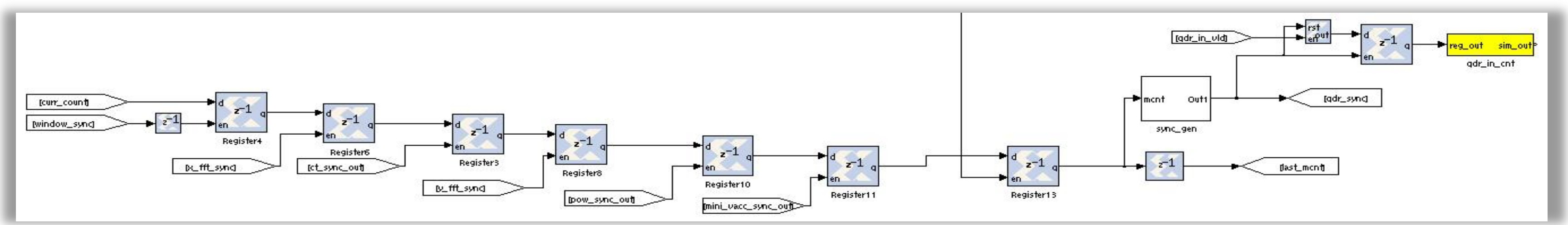
It is possible to get the 2D FFT simply by reading the snap bram block on the 10/100 Mb Ethernet port using the katcp library

S engine

Main Control Signals and Registers

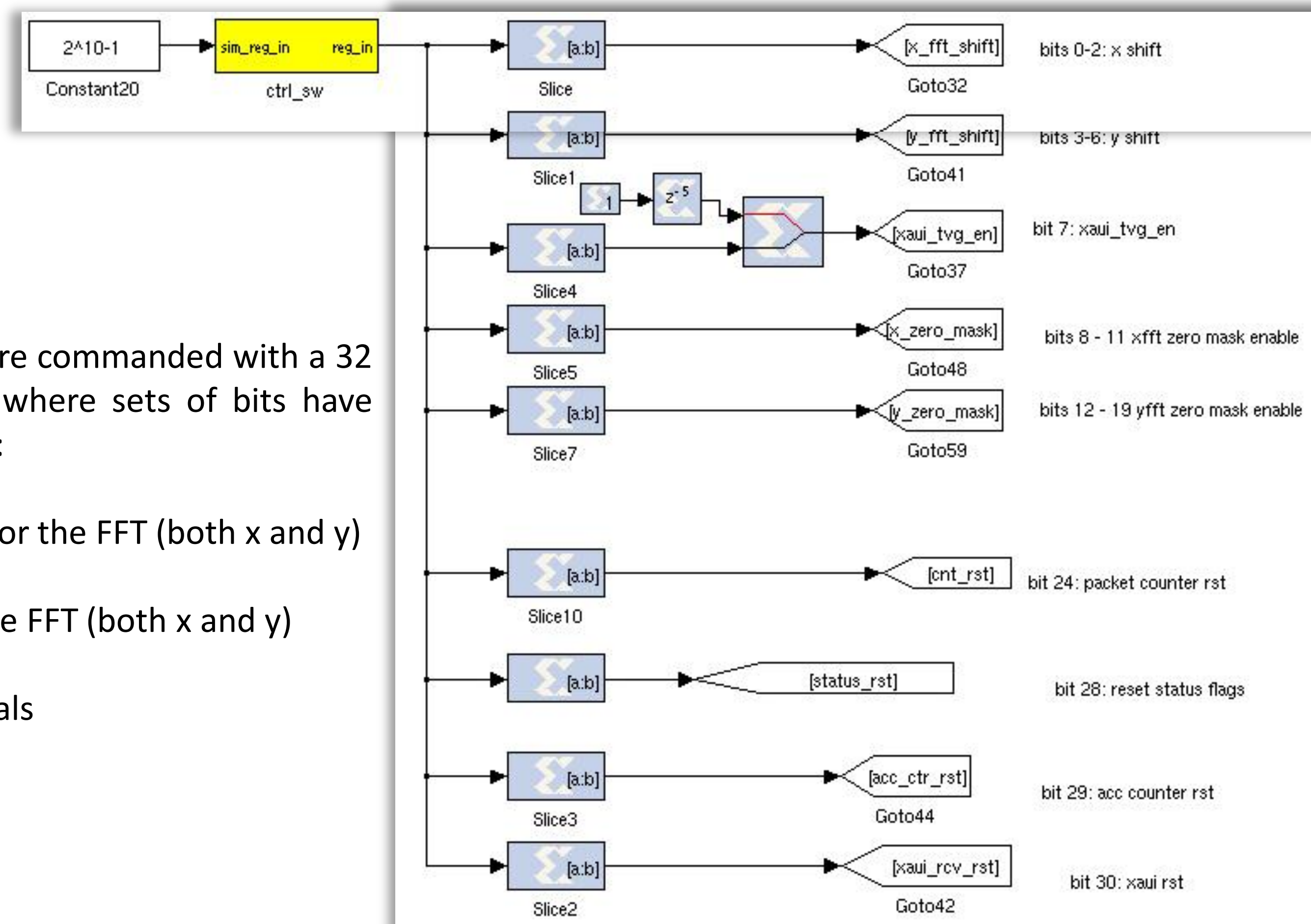


There are few output registers useful to debug the system parameters such master counts or even number of iterations of the QDR



S engine

Main Control Signals and Registers

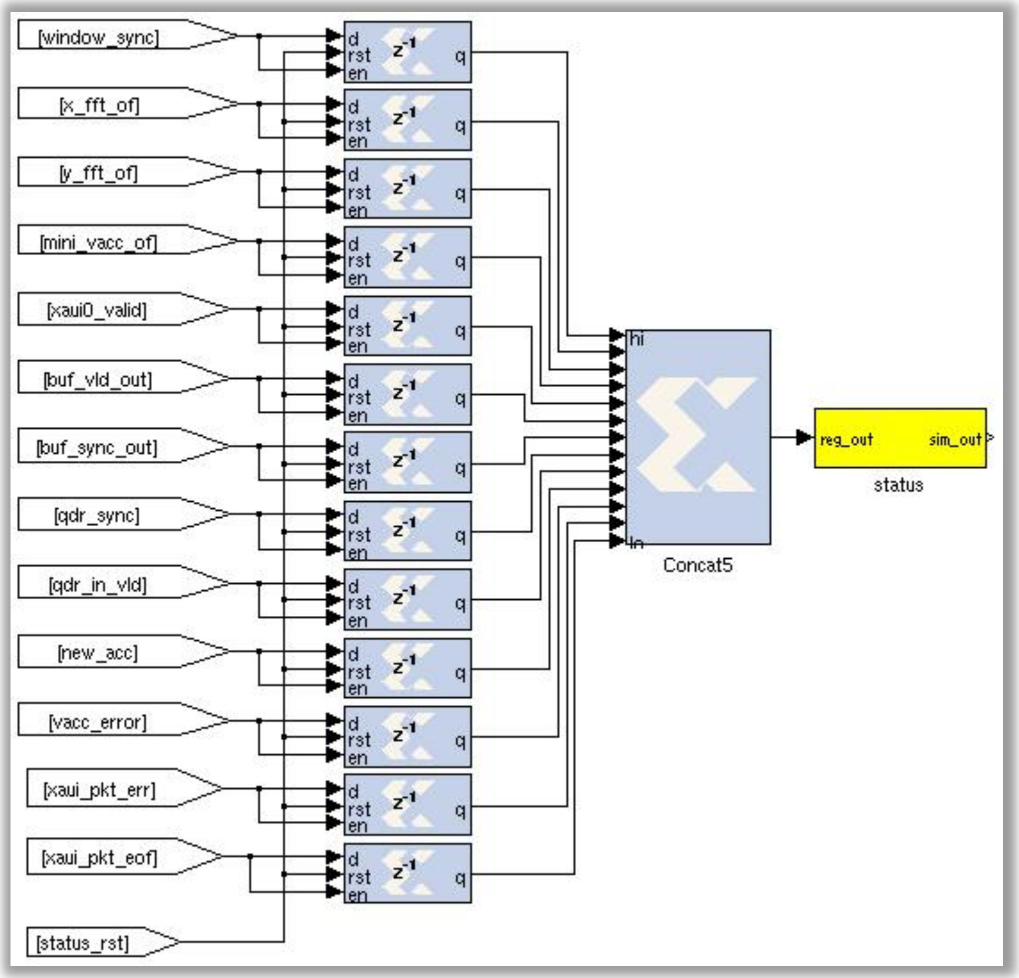


Most of the controls are commanded with a 32 bit register "ctrl_sw" where sets of bits have several meaning, such:

- Shifting schedule for the FFT (both x and y)
- Zeroes mask for the FFT (both x and y)
- Various Reset signals

S engine

Main Control Signals and Registers

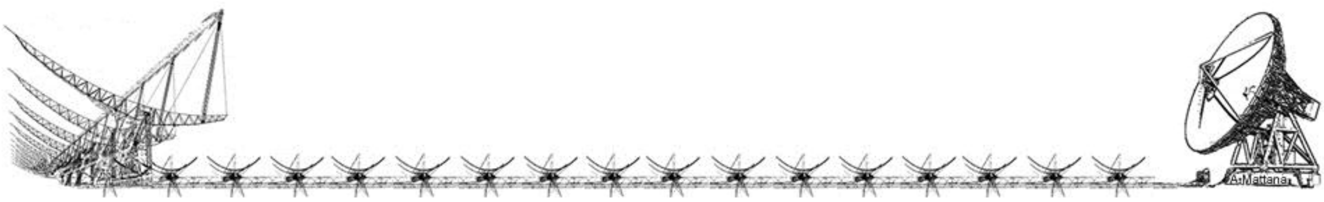


S engine

Even the output status is a sequence of bit concatenated in a 32 bit word

Acronyms

BEST	Basic Elements for SKA Training
CASA	Common Astronomy Software Applications
CASPER	Collaboration for Astronomy Signal Processing and Electronics Research
EDK	Embedded Development Kit
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
IDE	Integrated Development Environment
IF	Intermediate Frequency
IP	Intellectual Property
NRAO	National Radio Astronomy Observatory
PFB	Poliphase Filter Bank
QDR	Quad Data Rate
RF	Radio Frequency
ROACH	Reconfigurable Open Architecture Computing Hardware
SKA	Square Kilometer Array



Bibliography

[1] Parsons, A., et al., "*Digital Instrumentation for the Radio Astronomy Community*", astro-ph/0904.1181, April 2009.

[2] Montebugnoli, S.; Bianchi, G.; Monari, J.; Naldi, G.; Perini, F.; Schiaffino, M. "*BEST: Basic Element for SKA Training*", Wide Field Science and Technology for the Square Kilometre Array, Proceedings of the SKADS Conference held at the Chateau de Limelette, Belgium, 3-6 November 2009, p. 331-336. (2010)