FPGA Tool-flows: CASPER and Beyond

Wesley New
SKA-SA
wesley@ska.ac.za

Who am I?

- SKA-SA
- DBE Team - Digital Back End
- Real-time data processing
- Use FPGA based hardware
- ROACH Board
- CASPER Collaboration

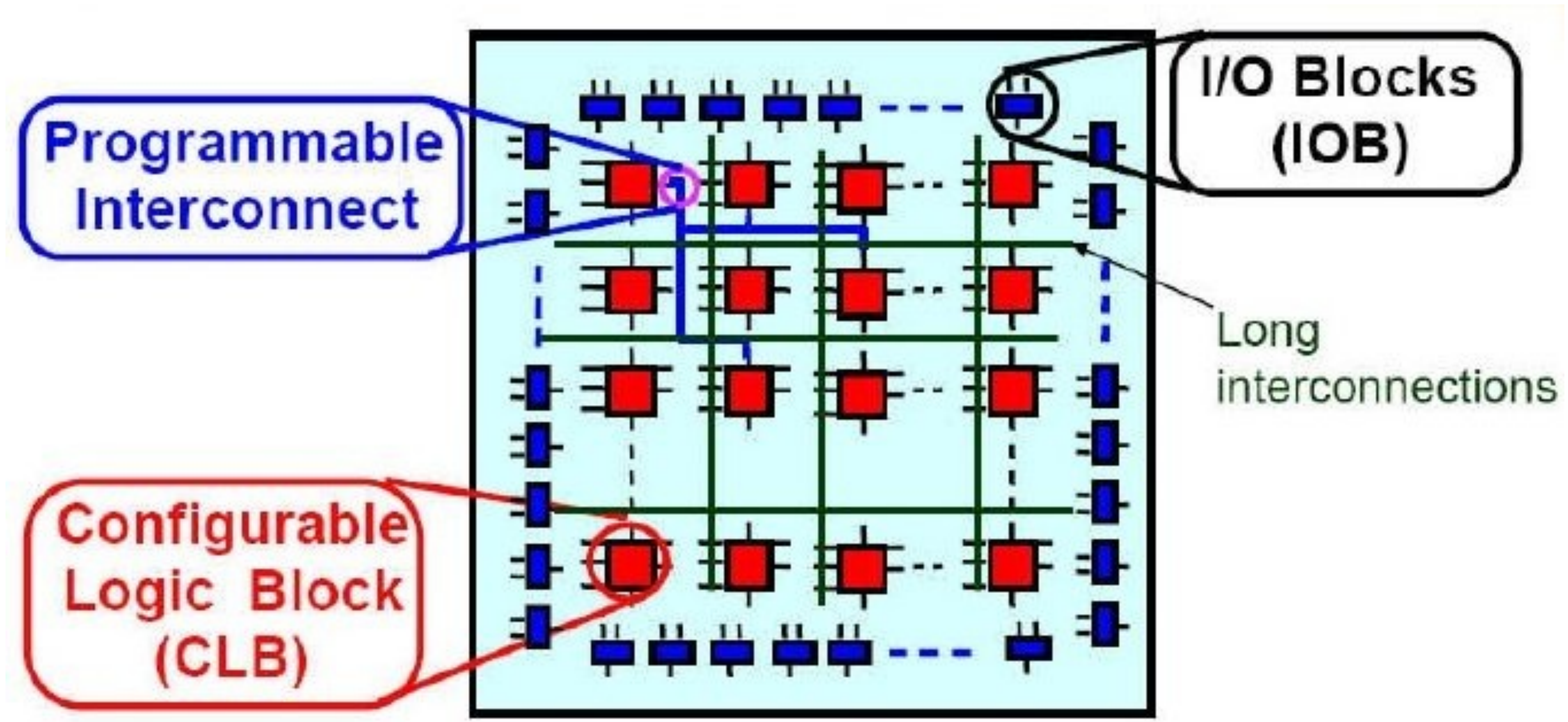
Aims

- Provide and overview of FPGA technologies
- Introduce the CASPER Collaboration
- Examine the CASPER FPGA Design-flow
- Introductions to the tutorials
- Discuss the future of FPGA design-flows and how CASPER and SKA can take advantage of these

Background: FPGAs

- FPGA - Field Programmable Gate Array
- Effectively a reconfigurable semiconductor
- Consists of Logic Elements and Interconnects
- Well suited to parallel DSP computing
- Contain Hard Cores
- Getting progressively more complex
- Design for FPGAs using Hardware Description Languages (HDL) Verilog and VHDL
- There is a move towards higher-level design
- CPU, GPU, FPGA, ASIC

FPGA: Logic and Interconnects



SRAMS cells throughout the FPGA determine the functionality of the device

FPGA Vendors

- 2 Major players, Xilinx and Altera plus a few smaller ones
- Each provide there own software for designing for their FPGAs
- Xilinx - ISE/Vivado
- Altera - Quartus
- Both offer plug-ins for Simulink to take advantage of block diagram style design and simulation
- Complexities of porting designs between vendors
- IP specific to a vendor

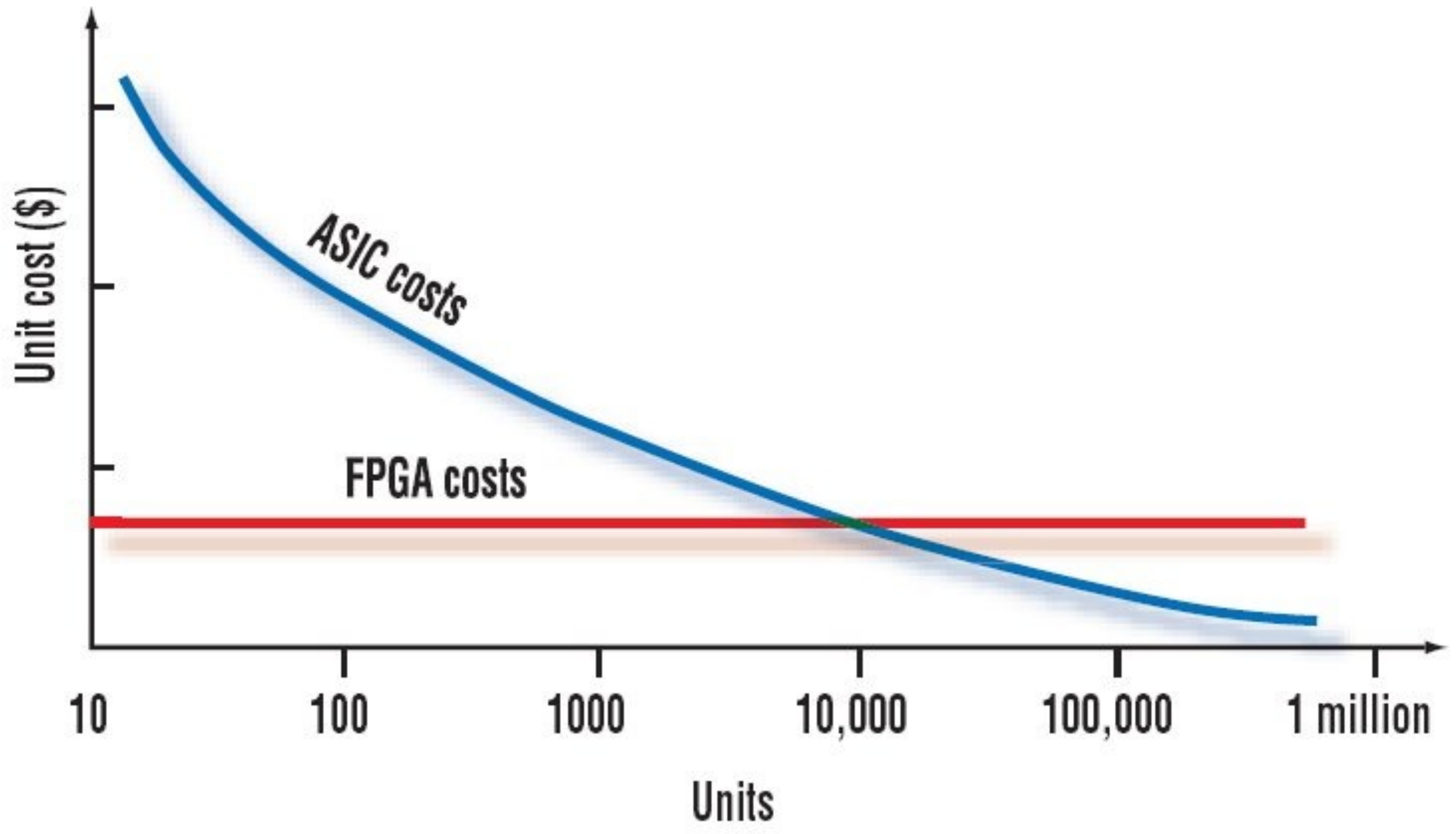
HDL (HW Description Lang)

- 2 Major languages Verilog and VHDL
- Verilog more of a C syntax
- HDL use the event-driven methodology
- Generally values of registers change on the edges of clocks

CPU v GPU v FPGA v ASIC

- Tradeoffs, tradeoffs, tradeoffs
- ASICs, Long time to develop, hard to make changes, high NRE costs, run at a higher speed
- FPGAs, Short development time, expensive per unit, bad at floating-point, highly reconfigurable
- GPU, Easier to design for, good at floating-point, high power consumption
- CPU, very general purpose, easy to develop for, lower performance

FPGA vs ASIC Cost Per Unit



HDL vs Traditions SW

- HDL is very susceptible to bad coding
- This can make designs use more power and resources
- The way of thinking when writing in an HDL is very different to software.
- HDLs statements are concurrent
- Sequential statements are executed simultaneously as apposed to sequentially
- For loops are also executed in parallel
- This is a huge change to the traditional programming mindset

HDL to Bit

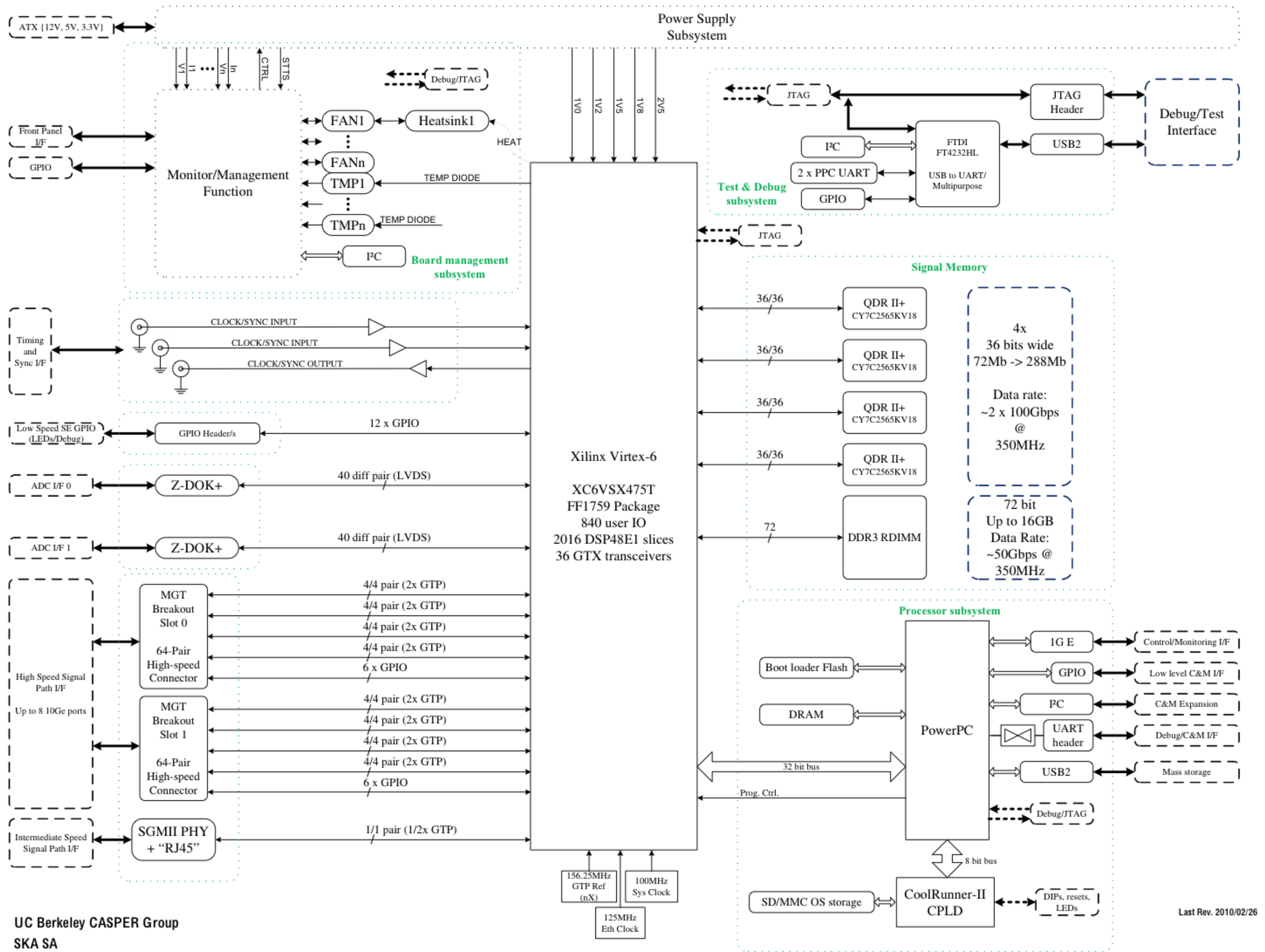
- Synthesis - translate HDL to gates and optimise
- MAP - to the resources of the FPGA
- Place and Route - Place the output of the MAP stage on the FPGA
- Timing analysis - Run thought the design and check that the timing constraints are met
- Bitfile generation - create the file to upload to the FPGA

CASPER

- CASPER, Collaboration for Astronomy Signal Processing and Electronics Research
- Open Source Philosophy
- Provides a series of FPGA based hardware, IBOB, BEE2, ROACH1, ROACH2 and many ADCs
- Provides a design-flow for developing applications for this hardware
- Provides a space in which to share and collaborate in astronomy instrumentation design
- Members from all around the world

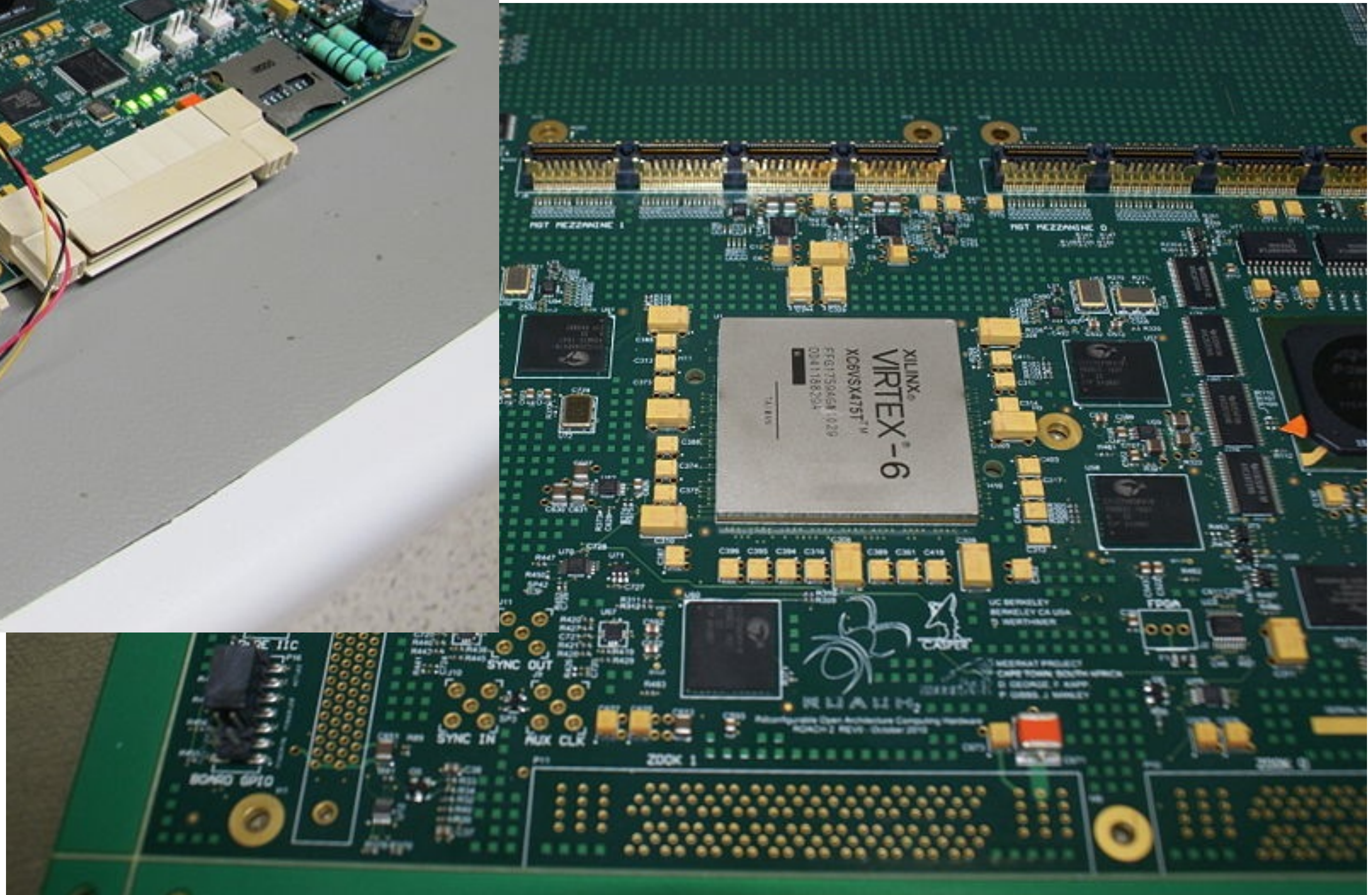
CASPER

ROACH2 Architecture



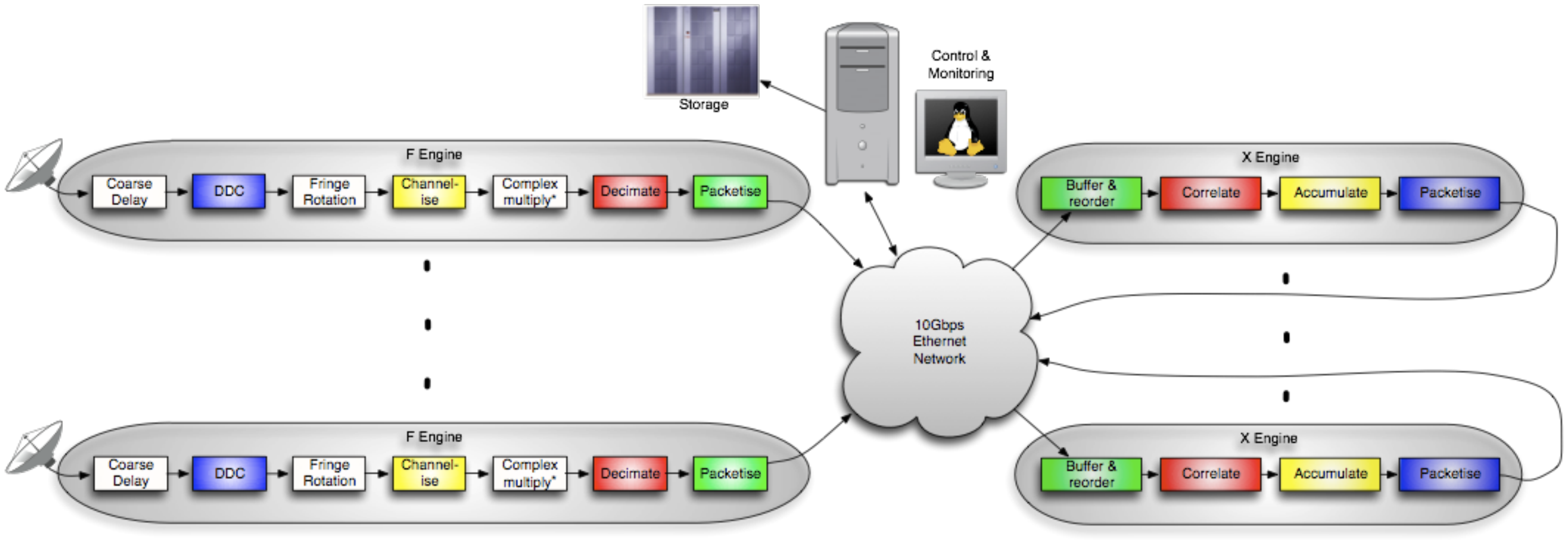
UC Berkeley CASPER Group
SKA SA

ROACH2



ROACH
Reconfigurable Open Architecture Computing Hardware

FPGA: Logic and Interconnects



*Complex multiply allows for fine delay control and per-channel digital gain control.
White coloured blocks not yet implemented.

CASPER/MSSGE Toolflow

- Matlab
- Simulink
- Xilinx System Generator
- Xilinx EDK
- CASPER Libraries, framework and base projects

Matlab/Simulink

- Simulink provides an environment for block diagram design
- Provides blocks to aid in simulating designs
- Such as Signal Generators and Scopes
- It is also possible to pull use the Matlab language to aid in simulation
- Such as generation of inputs, comparing outputs and verify the simulation

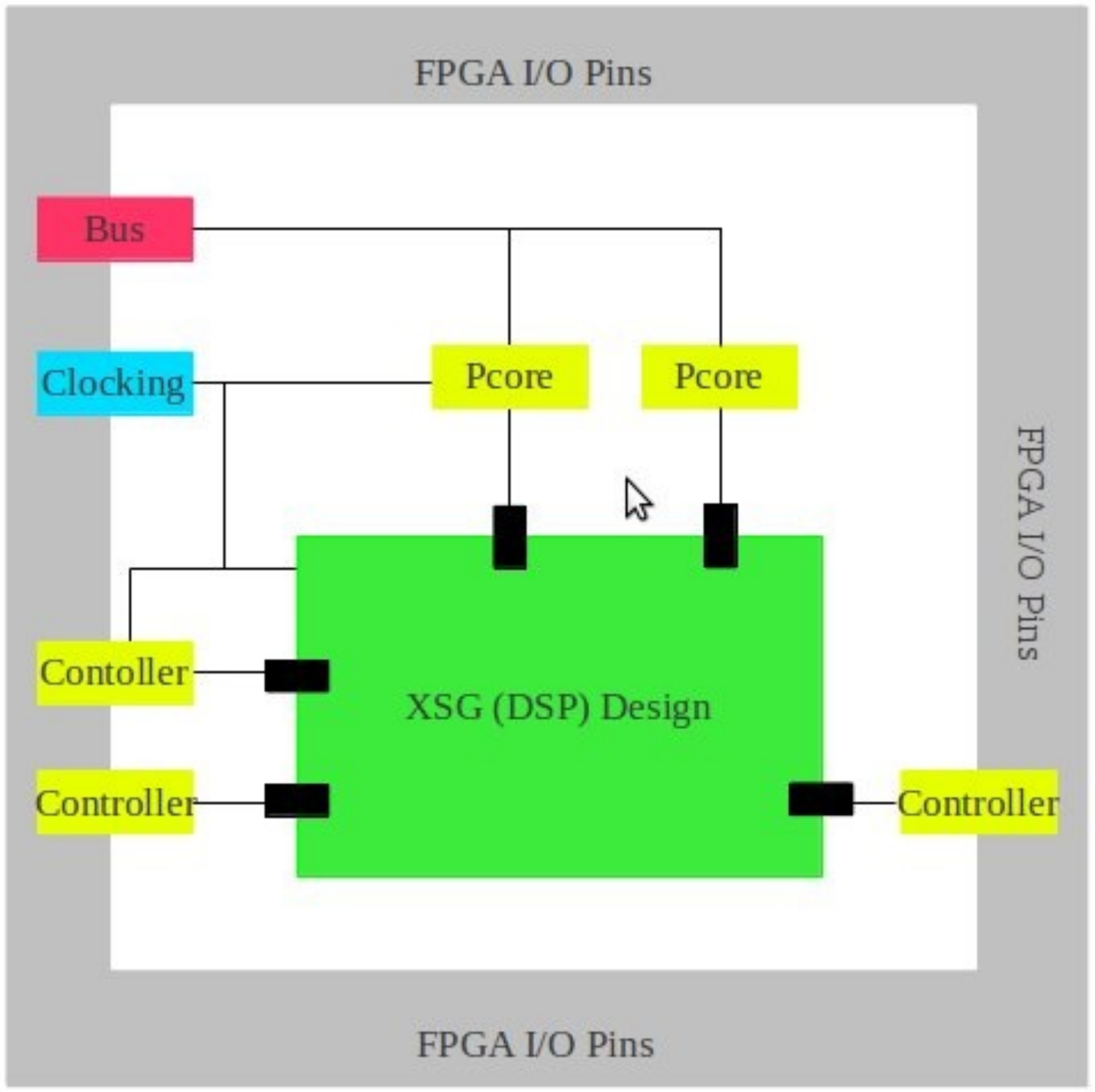
XSG (Xilinx System Generator)

- Plugs into Simulink
- Provides the Xilinx blockset to use in the Simulink environment
- Simulation models are also provided
- Lets the designer target a particular FPGA chip to taylor the blocks for best performance
- Generates a netlist of the whole DSP design
- This is then used pulled into a base project and connected up appropriately

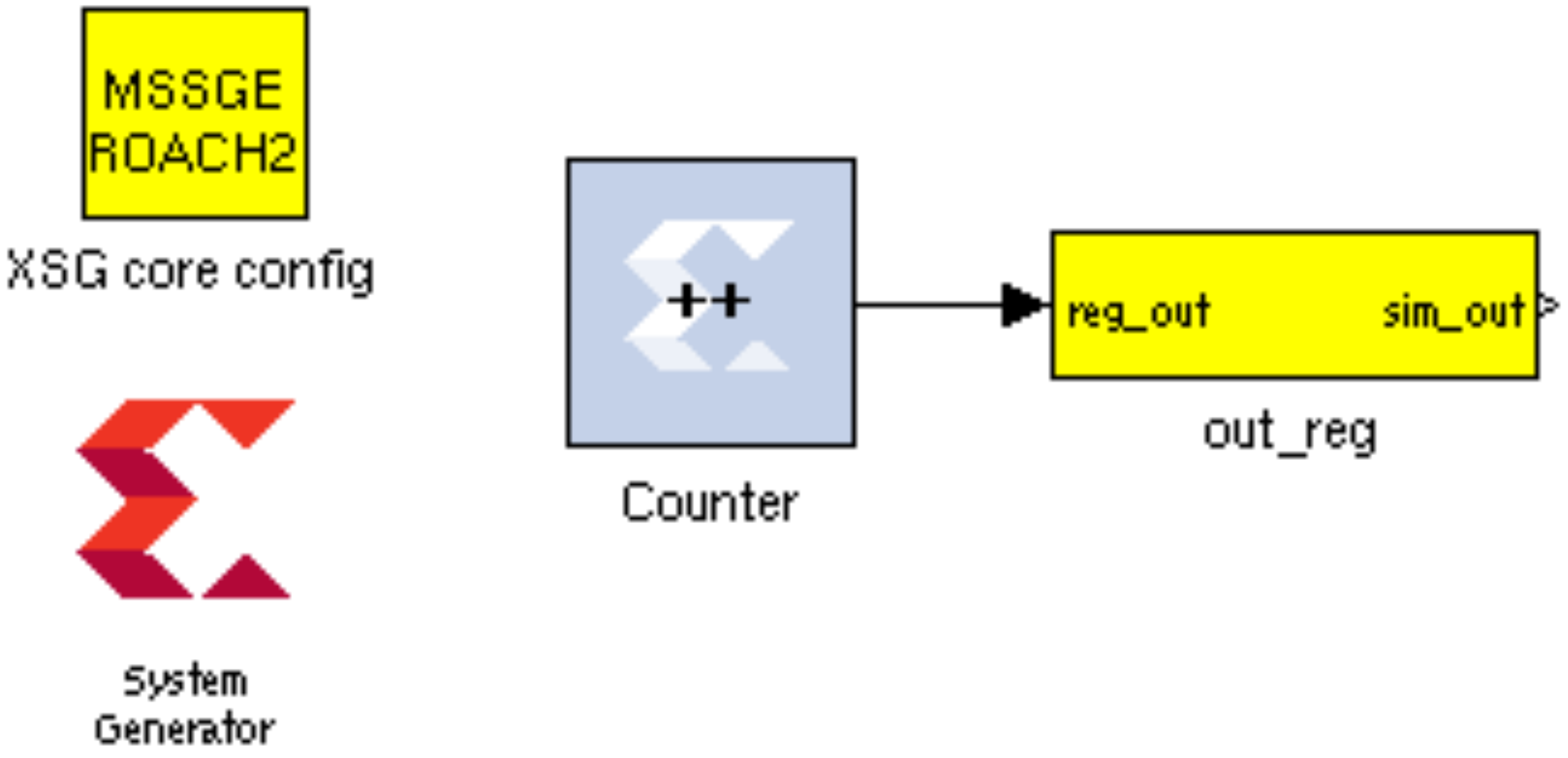
Xilinx EDK

- Pcores controllers
- Used as part of the glue logic to pull aspects of the design together
- Manages the bus infrastructure

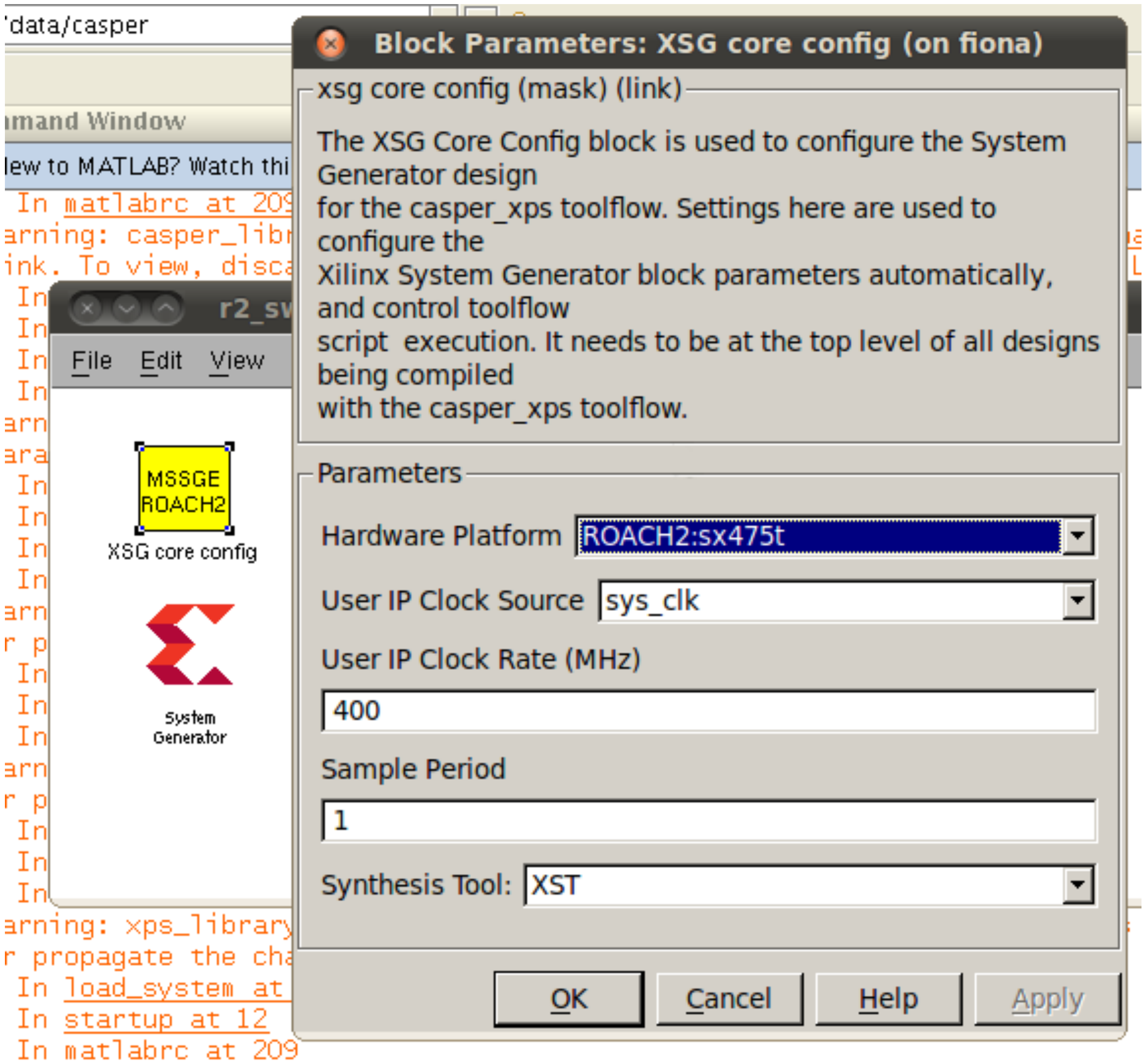
Typical FPGA Design



Simple CASPER Design



Design Configuration



Block Parameters: XSG core config (on fiona)

xsg core config (mask) (link)

The XSG Core Config block is used to configure the System Generator design
for the casper_xps toolflow. Settings here are used to configure the
Xilinx System Generator block parameters automatically, and control toolflow
script execution. It needs to be at the top level of all designs being compiled
with the casper_xps toolflow.

Parameters

Hardware Platform: ROACH2:sx475t

User IP Clock Source: sys_clk

User IP Clock Rate (MHz): 400

Sample Period: 1

Synthesis Tool: XST

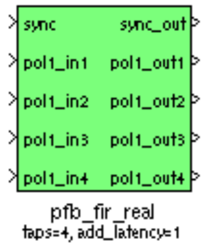
CASPER: Base Designs

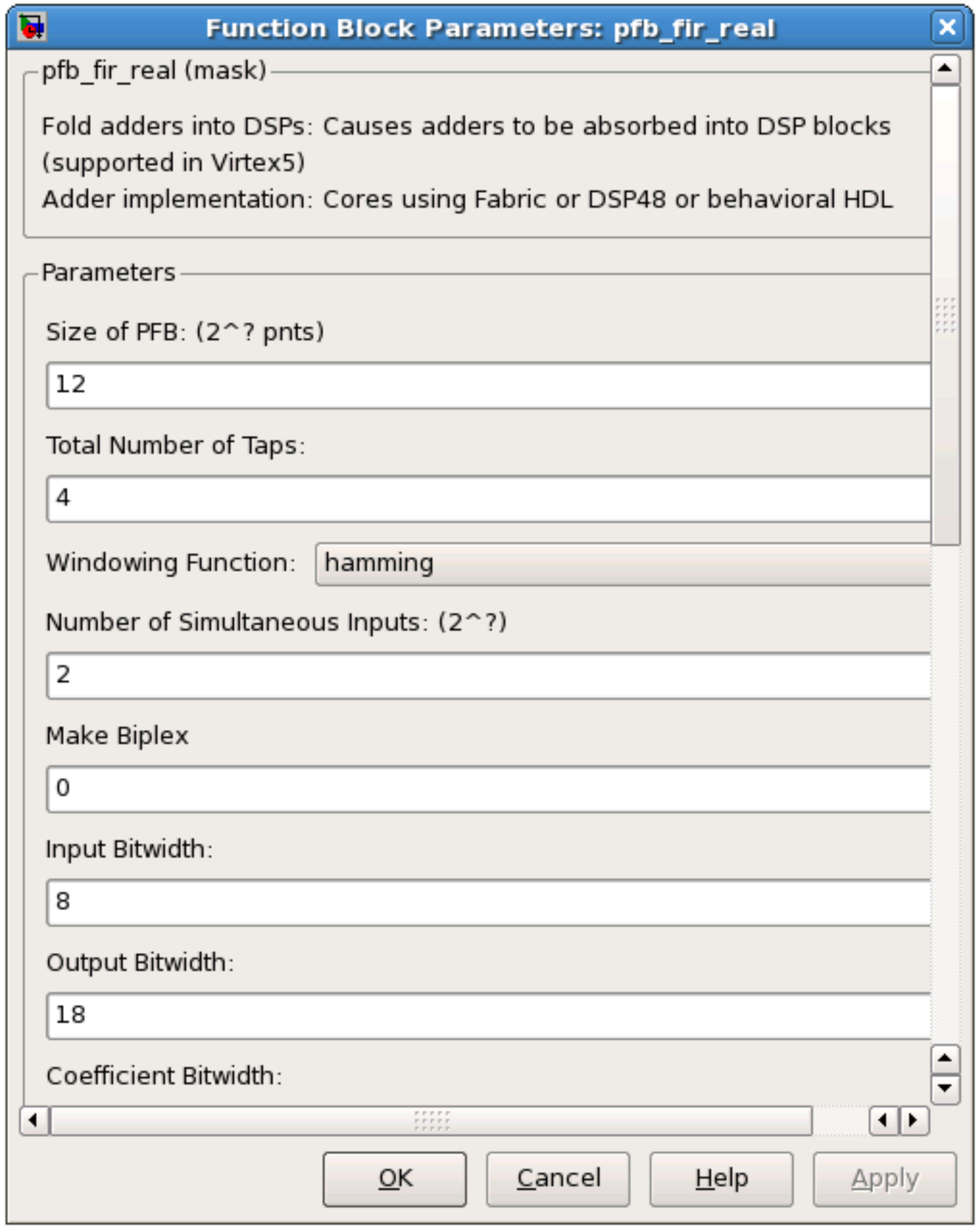
- Each hardware platform supported by the CASPER tools has a base design
- Applications (DSP designs) get pulled into the project
- This manages clocking infrastructure
- Control bus infrastructure
- And configures constraints

CASPER: DSP Libraries

- Green Blocks
- Each block has a mask scrips
- This redraws the underlying block when a parameter is changed
- This allows a huge amount of flexibility when designing a block

PFB FIR





pfb_fir_real (mask)

Fold adders into DSPs: Causes adders to be absorbed into DSP blocks (supported in Virtex5)

Adder implementation: Cores using Fabric or DSP48 or behavioral HDL

Parameters

Size of PFB: (2^? pnts): 12

Total Number of Taps: 4

Windowing Function: hamming

Number of Simultaneous Inputs: (2^?): 2

Make Biplex: 0

Input Bitwidth: 8

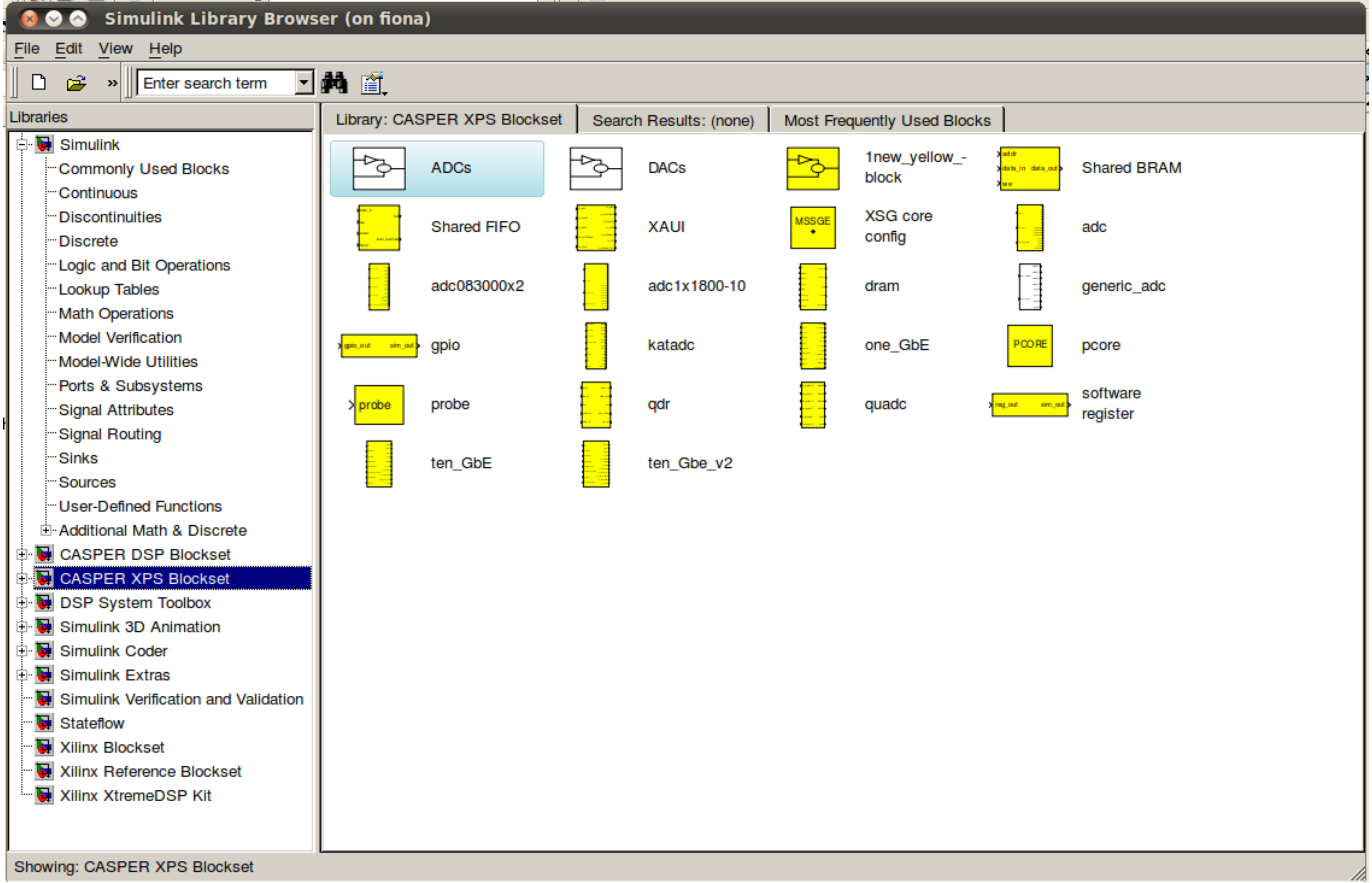
Output Bitwidth: 18

Coefficient Bitwidth:

CASPER: Controller Libraries

- Yellow Blocks
- Any block that interacts with peripherals
- These are scripted to pull in the correct core for the hardware
- Registers accessible from the CPU, DRAM controllers, ADC controllers

Controller Libraries



Simulink Library Browser (on fiona)

File Edit View Help

Enter search term

Libraries

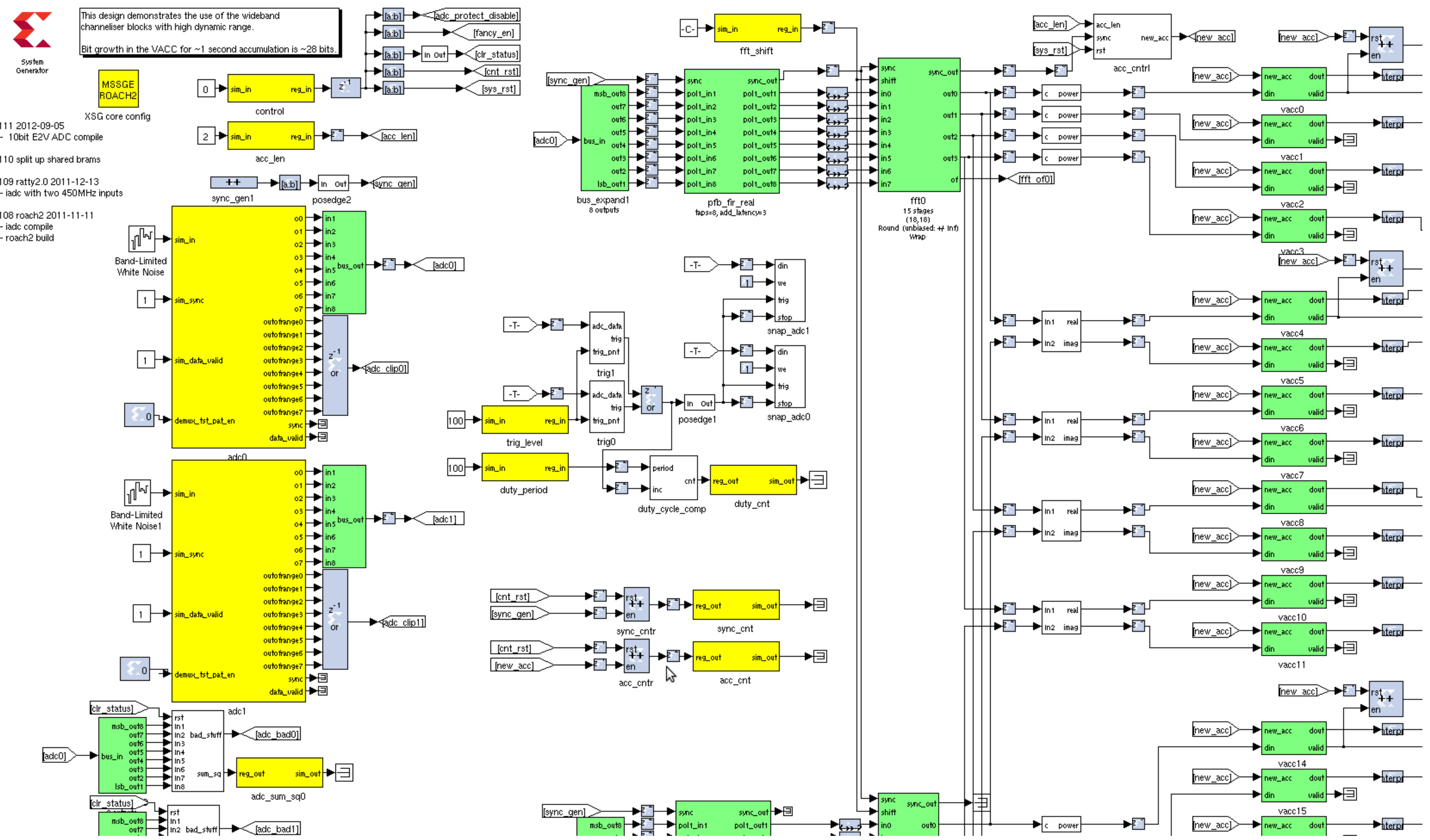
- Simulink
  - Commonly Used Blocks
  - Continuous
  - Discontinuities
  - Discrete
  - Logic and Bit Operations
  - Lookup Tables
  - Math Operations
  - Model Verification
  - Model-Wide Utilities
  - Ports & Subsystems
  - Signal Attributes
  - Signal Routing
  - Sinks
  - Sources
  - User-Defined Functions
  - Additional Math & Discrete
- CASPER DSP Blockset
- CASPER XPS Blockset
- DSP System Toolbox
- Simulink 3D Animation
- Simulink Coder
- Simulink Extras
- Simulink Verification and Validation
- Stateflow
- Xilinx Blockset
- Xilinx Reference Blockset
- Xilinx XtremeDSP Kit

Library: CASPER XPS Blockset | Search Results: (none) | Most Frequently Used Blocks

ADCs	DACs	1new_yellow_-block	Shared BRAM
Shared FIFO	XAUI	XSG core config	adc
adc083000x2	adc1x1800-10	dram	generic_adc
gpio	katadc	one_GbE	pcore
probe	qdr	quadc	software register
ten_GbE	ten_Gbe_v2		

Showing: CASPER XPS Blockset

Complex Design



This design demonstrates the use of the wideband channeliser blocks with high dynamic range.

Bit growth in the VACC for ~1 second accumulation is ~28 bits.

'111 2012-09-05
- 10bit E2V ADC compile

'110 split up shared brams

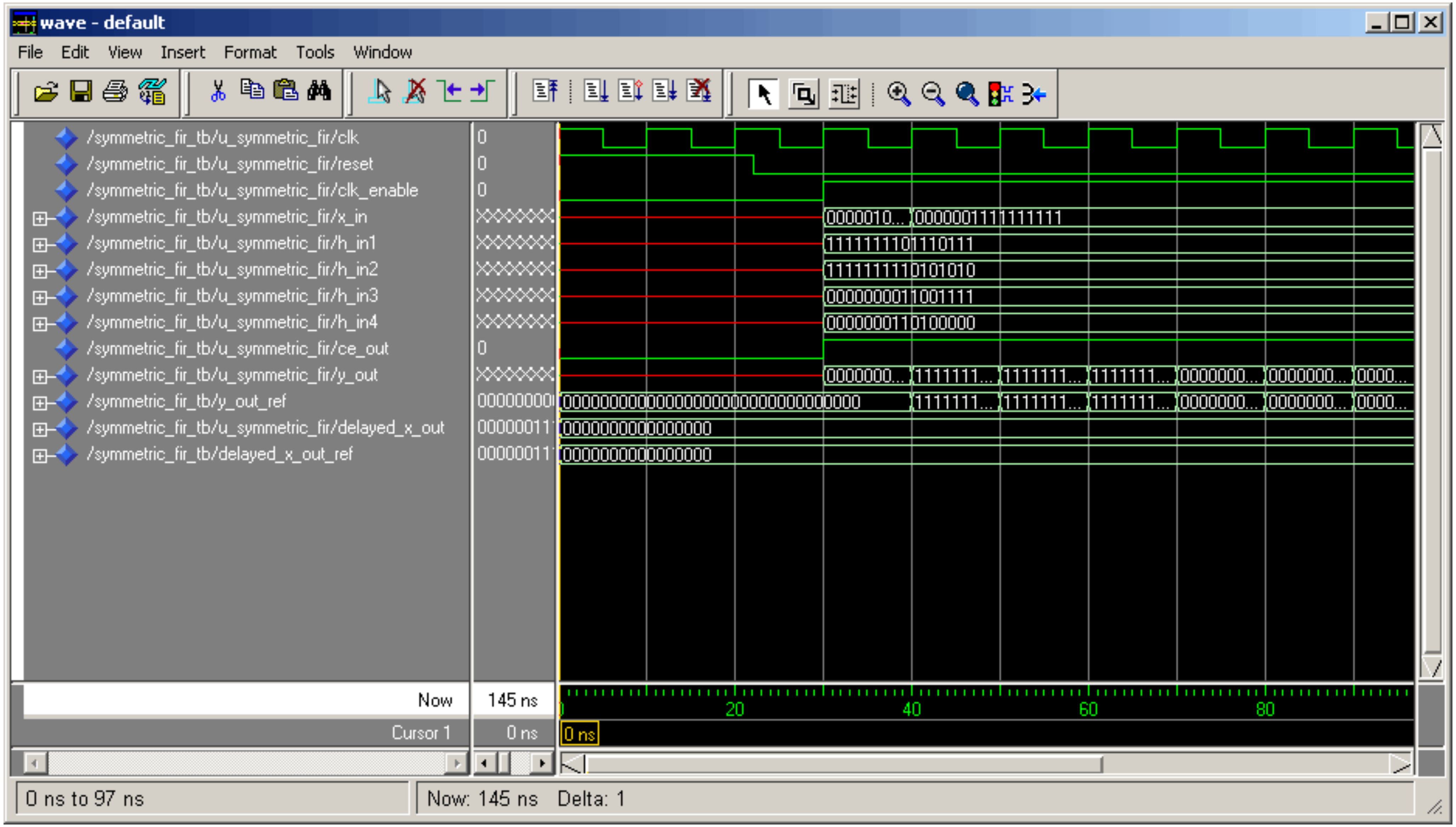
'109 ratty2.0 2011-12-13
- iadc with two 450MHz inputs

'108 roach2 2011-11-11
- iadc compile
- roach2 build

Simulation is Key

- Simulink provides the ability to simulate designs
- This is one of the most important features of the tools
- Unfortunately bit-wise simulation can take days to complete
- Forced to simulate smaller sections of the design and test their integration on the FPGA

Tut1 Bit Simulation



wave - default

```
/symmetric_fir_tb/u_symmetric_fir/clk              0
/symmetric_fir_tb/u_symmetric_fir/reset            0
/symmetric_fir_tb/u_symmetric_fir/clk_enable       0
/symmetric_fir_tb/u_symmetric_fir/x_in             XXXXXX   0000010...  0000000111111111
/symmetric_fir_tb/u_symmetric_fir/h_in1            XXXXXX   1111111101110111
/symmetric_fir_tb/u_symmetric_fir/h_in2            XXXXXX   1111111110101010
/symmetric_fir_tb/u_symmetric_fir/h_in3            XXXXXX   0000000011001111
/symmetric_fir_tb/u_symmetric_fir/h_in4            XXXXXX   0000000110100000
/symmetric_fir_tb/u_symmetric_fir/ce_out           0
/symmetric_fir_tb/u_symmetric_fir/y_out            XXXXXX   0000000...  1111111...  1111111...  1111111...  0000000...  0000000...  0000...
/symmetric_fir_tb/y_out_ref                        00000000  00000000000000000000000000000000  1111111...  1111111...  1111111...  0000000...  0000000...  0000...
/symmetric_fir_tb/u_symmetric_fir/delayed_x_out    00000011  0000000000000000
/symmetric_fir_tb/delayed_x_out_ref                00000011  0000000000000000

Now       145 ns      0   20   40   60   80
Cursor 1  0 ns        0 ns

0 ns to 97 ns         Now: 145 ns   Delta: 1
```

CASPER Success

- Model driven development approach
- Easy to use
- Abstracts the application designer away from the low-level technical aspects of FPGAs, so that he can focus on the application
- Collaboration, open-source

Future of FPGA Design-flows

- Model driven development approach is key
- One click compile solutions
- Easy migration of designs from one hardware platform to another
- High-level Languages used for FPGA design
- Mathworks HDL Coder, MyHDL (Python), C to Gates, Migen

Ideal Simulation

- Bitwise simulation takes a long time
- Need the ability to simulate parts of the design and then use a higher level simulation to verify the design as a whole
- Different levels of simulation
- Bit-wise
- Functional Verification
- Co-simulation
- simulation models for each module in the design

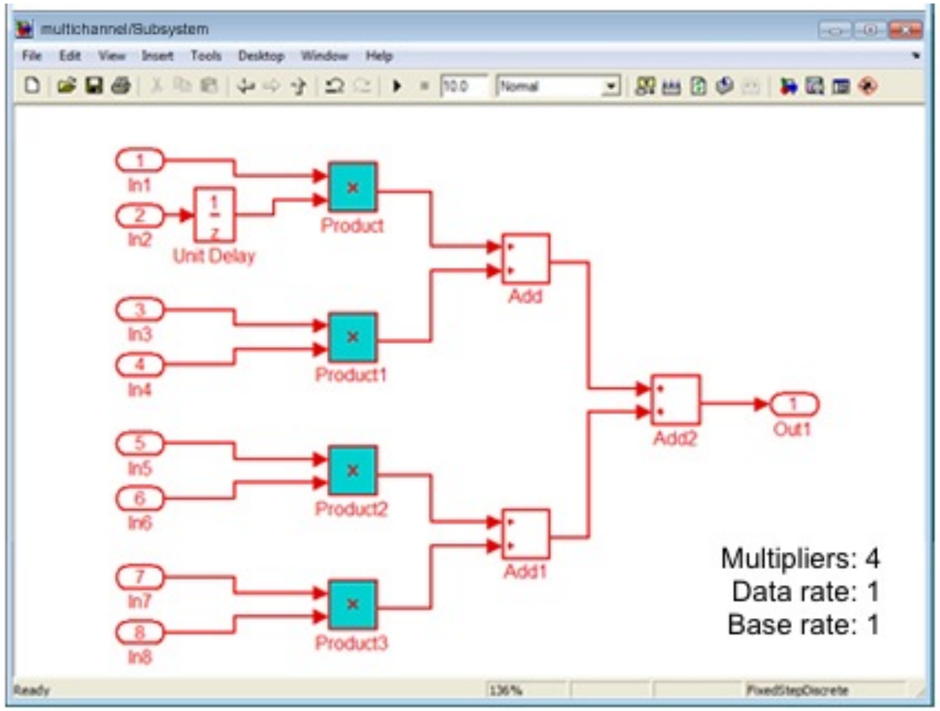
HDL Coder

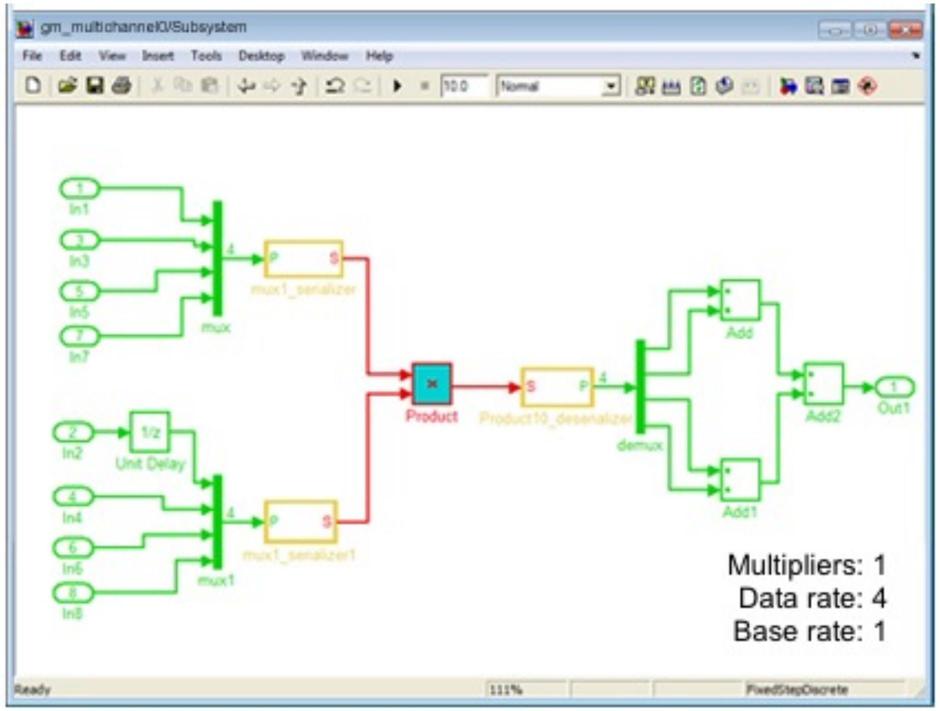
- Model driven development approach is key
- One click compile solutions
- Easy migration of applications from one hardware platform to another

HDL Coder

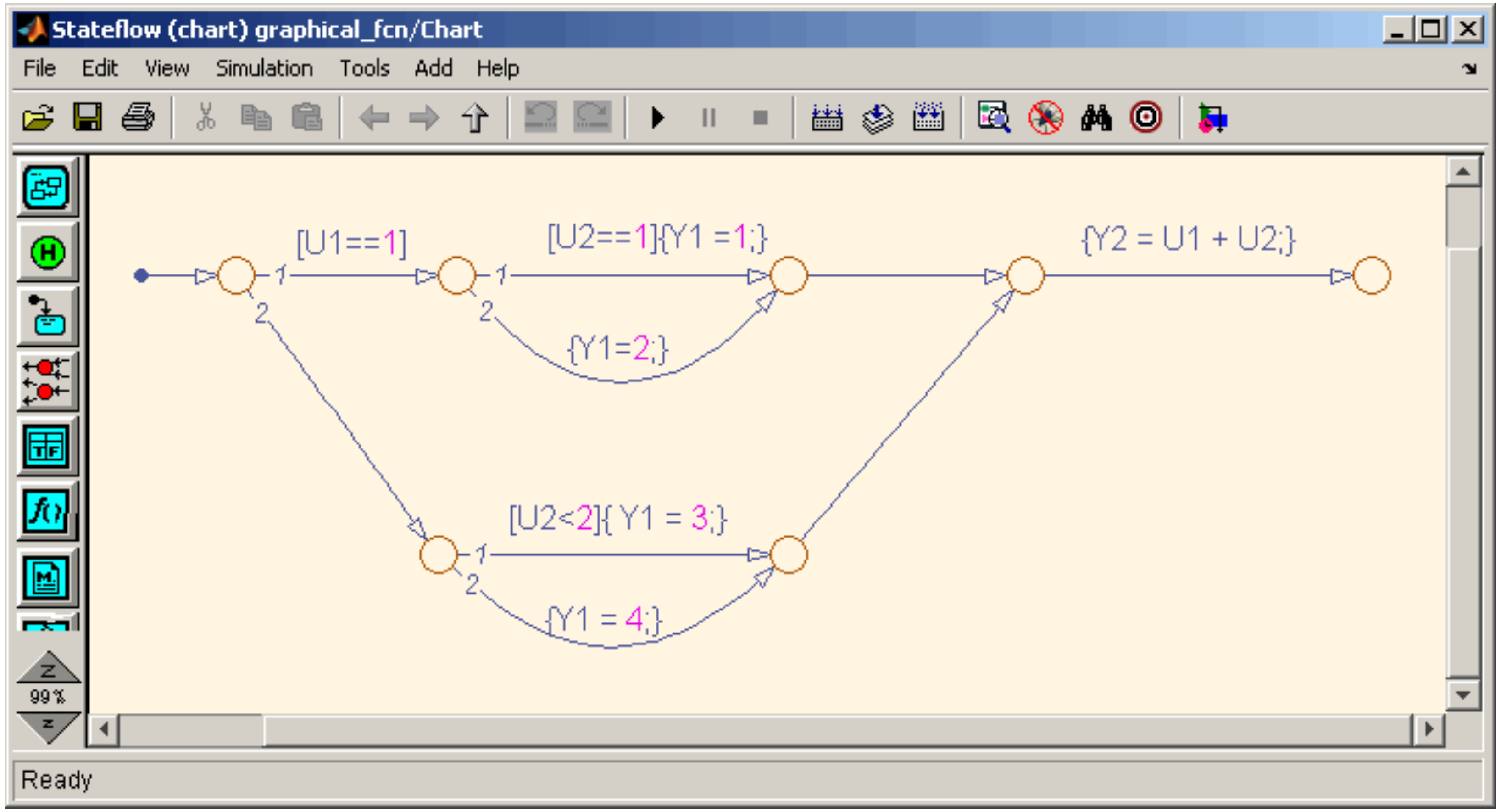
- Generic HDL generator
- Target independent synthesizable Verilog and VHDL, but…
- Convert Mealy and Moore state charts to HDL
- Convert Matlab code to HDL
- Provides automatic pipelining
- Provides resource estimations
- Integration between design documents and design
- Can target custom boards

HDL Coder


multichannel/Subsystem
In1
In2
Unit Delay
Product
Add
In3
In4
Product1
In5
In6
Product2
Add1
Add2
Out1
In7
In8
Product3
Multipliers: 4
Data rate: 1
Base rate: 1
Ready
136%
FixedStepDiscrete


gm_multichannel0/Subsystem
In1
In3
In5
In7
mux
mux1_serializer
In2
Unit Delay
In4
In6
In8
mux1
mux1_serializer1
Product
Product10_deserializer
demux
Add
Add1
Add2
Out1
Multipliers: 1
Data rate: 4
Base rate: 1
Ready
111%
FixedStepDiscrete

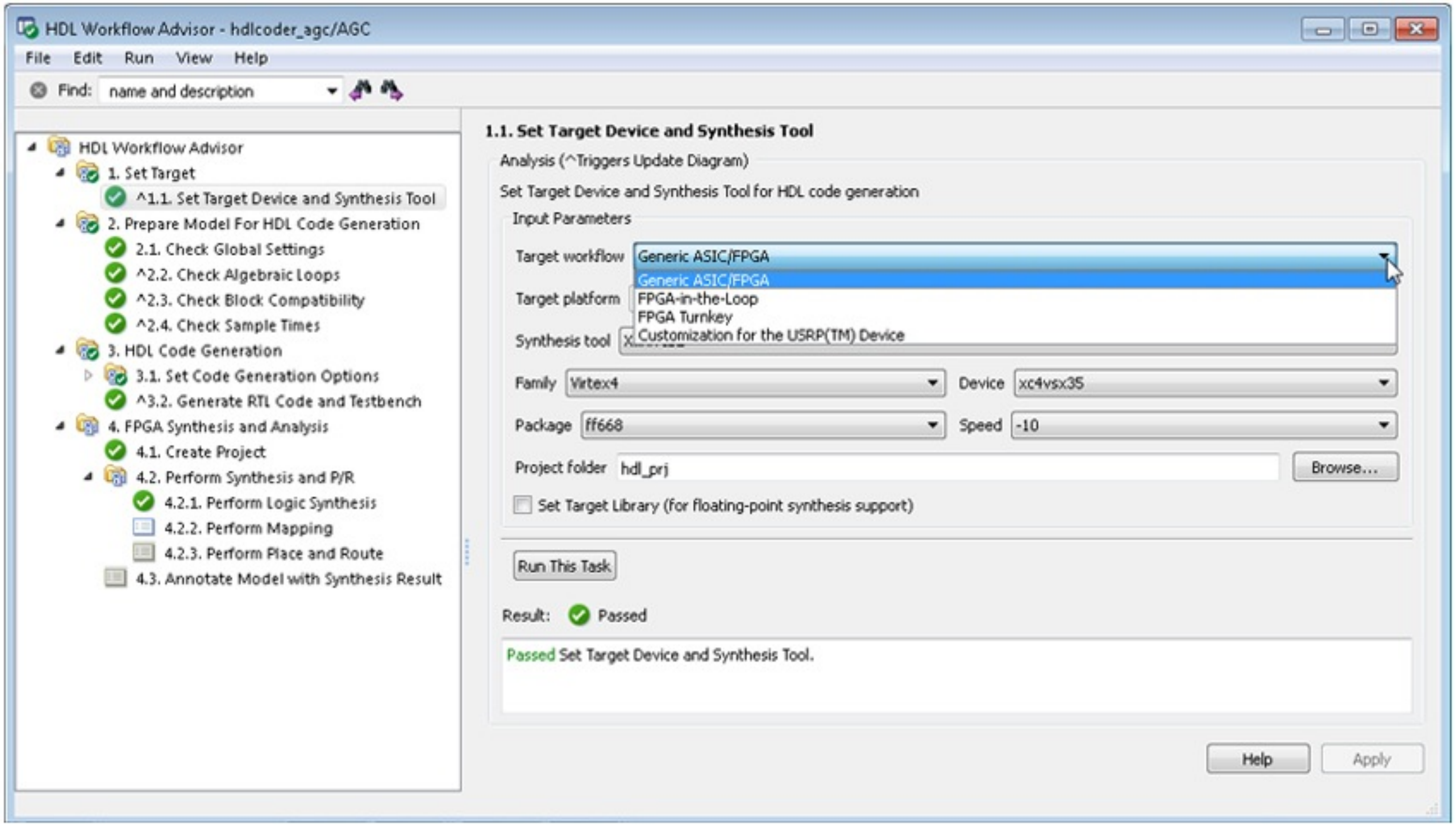
HDL Coder


Stateflow (chart) graphical_fcn/Chart
File Edit View Simulation Tools Add Help
[U1==1]
[U2==1]{Y1 =1;}
{Y2 = U1 + U2;}
{Y1=2;}
[U2<2]{ Y1 = 3;}
{Y1 = 4;}
99 %
Ready

HDL Coder and CASPER

- Can integrate the current mask scripts that are used to redraw the current CASPER blocks, by using HDL Coder the blockset
- Ability to simulate or verify
- Support for custom boards

HDL Coder



HDL Workflow Advisor - hdlcoder_agc/AGC

File Edit Run View Help

Find: name and description

- HDL Workflow Advisor
  - 1. Set Target
    - ^1.1. Set Target Device and Synthesis Tool
  - 2. Prepare Model For HDL Code Generation
    - 2.1. Check Global Settings
    - ^2.2. Check Algebraic Loops
    - ^2.3. Check Block Compatibility
    - ^2.4. Check Sample Times
  - 3. HDL Code Generation
    - 3.1. Set Code Generation Options
    - ^3.2. Generate RTL Code and Testbench
  - 4. FPGA Synthesis and Analysis
    - 4.1. Create Project
    - 4.2. Perform Synthesis and P/R
      - 4.2.1. Perform Logic Synthesis
      - 4.2.2. Perform Mapping
      - 4.2.3. Perform Place and Route
    - 4.3. Annotate Model with Synthesis Result

1.1. Set Target Device and Synthesis Tool

Analysis (^Triggers Update Diagram)

Set Target Device and Synthesis Tool for HDL code generation

Input Parameters

Target workflow: Generic ASIC/FPGA

Generic ASIC/FPGA
FPGA-in-the-Loop
FPGA Turnkey
Customization for the USRP(TM) Device

Target platform:

Synthesis tool: X

Family: Virtex4 Device: xc4vsx35

Package: ff668 Speed: -10

Project folder: hdl_prj Browse...

☐ Set Target Library (for floating-point synthesis support)

Run This Task

Result: ✓ Passed

Passed Set Target Device and Synthesis Tool.

Help Apply

MyHDL Overview

What is MyHDL?

MyHDL is an open-source Python package that enables Python to be used as a hardware description language. It does this by means of the Python Generator and Decorator functionality. MyHDL code can be converted to either Verilog or VHDL and then implemented onto silicon.

The power of Python is that it provides a high level design language and the ability to simulate the design using other Python packages such as NumPy and SciPy.

How does MyHDL Model Hardware in a Functional/O-O Language

- Concurrency

- A = B
- B = A

- Generators provide an elegant solution for modelling concurrency
- A generator is a resumable function
- Instead of return we use yield

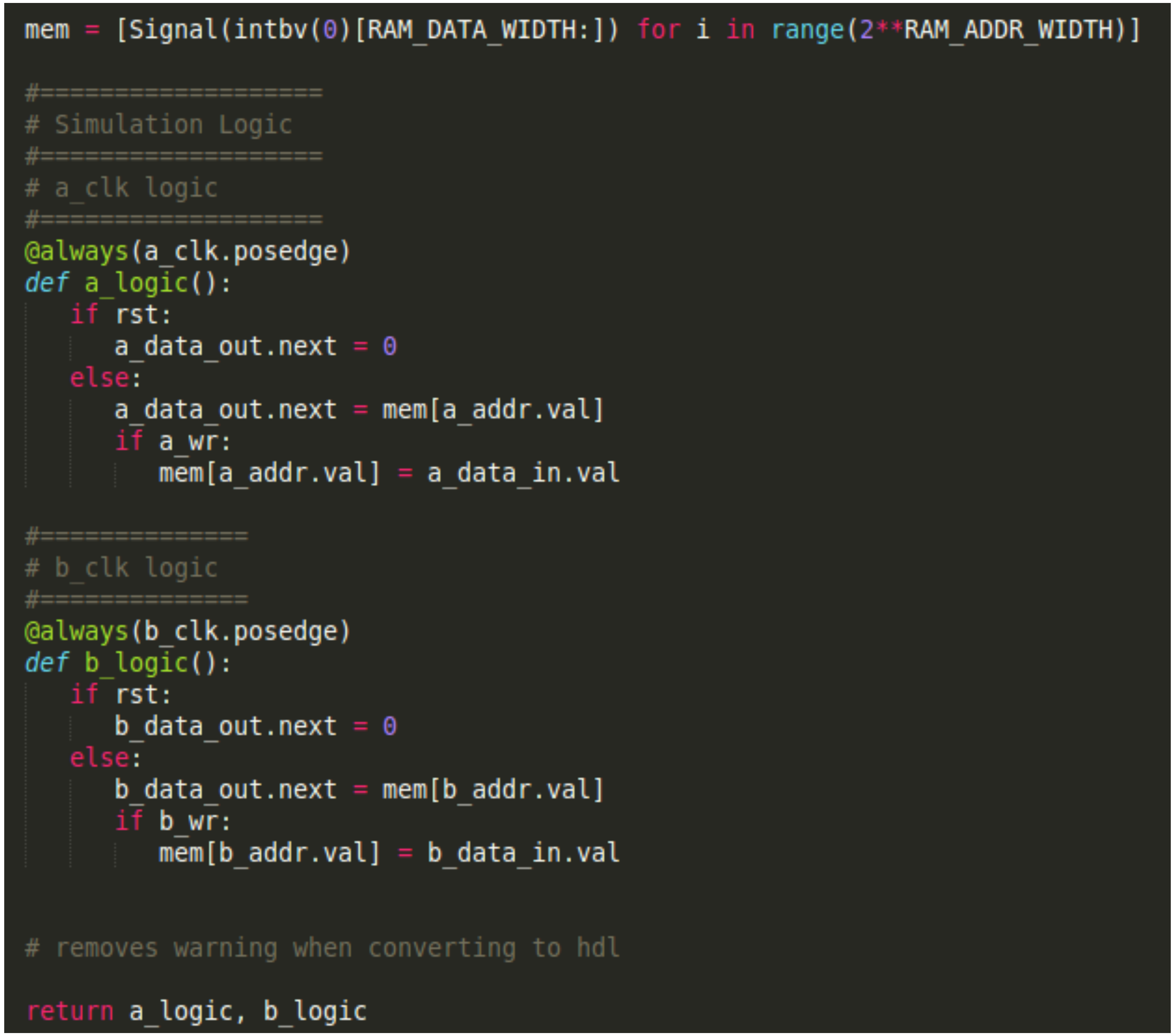
MyHDL

- Enables Python to be used as a high-level modelling language.
- Converts Python to HDL
- Can be used to wrap existing HDL code
- Provides the ability to simulate and model designs
- Allows OO concepts to be used in hardware design ie bus objects

MyHDL Architecture

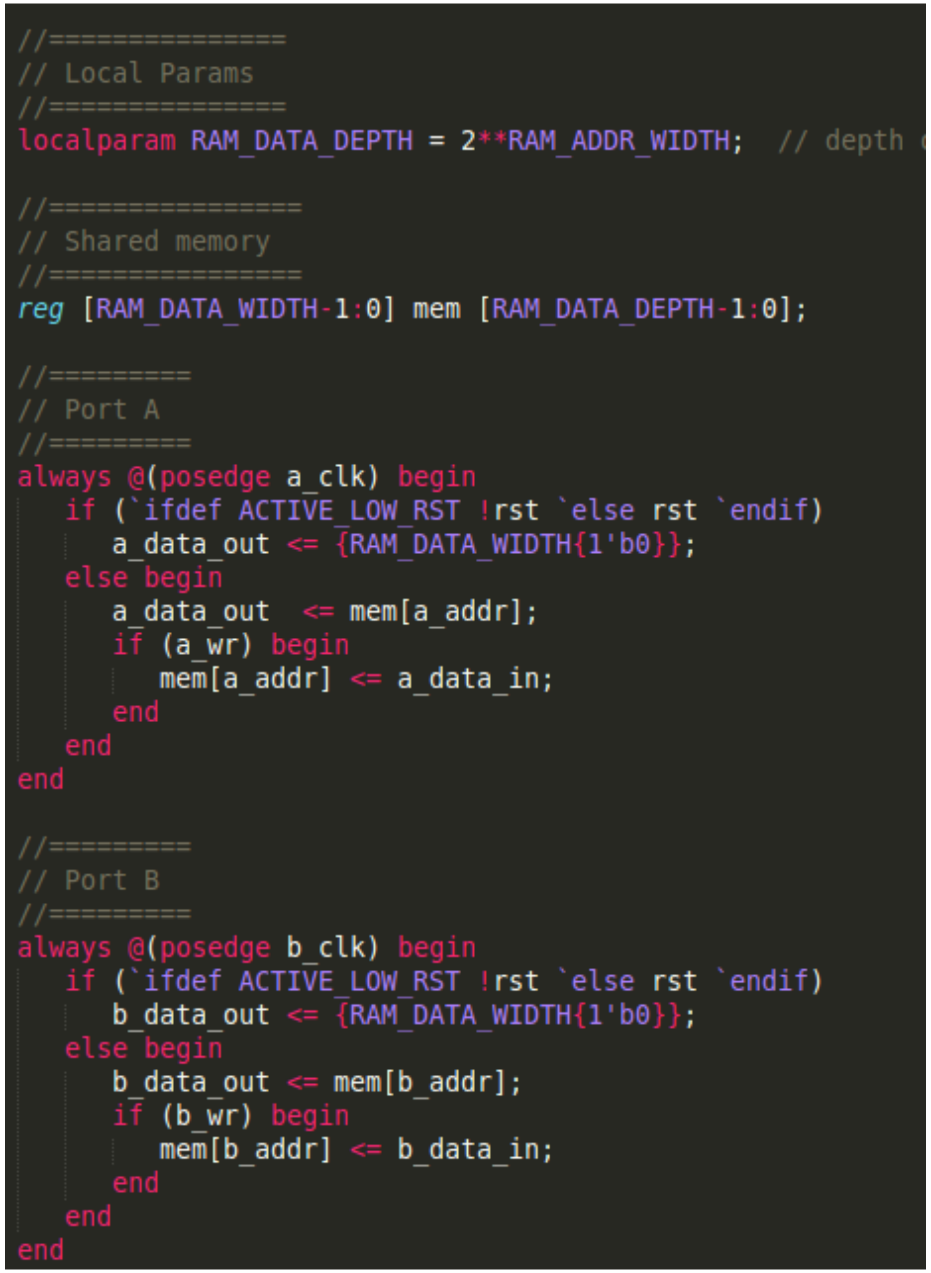
- Python conversion to HDL
- Using MyHDL to wrap HDL modules
- Modelling the HDL modules in Python
- Able to use ngc files, HDL and Python in one design

MyHDL Example



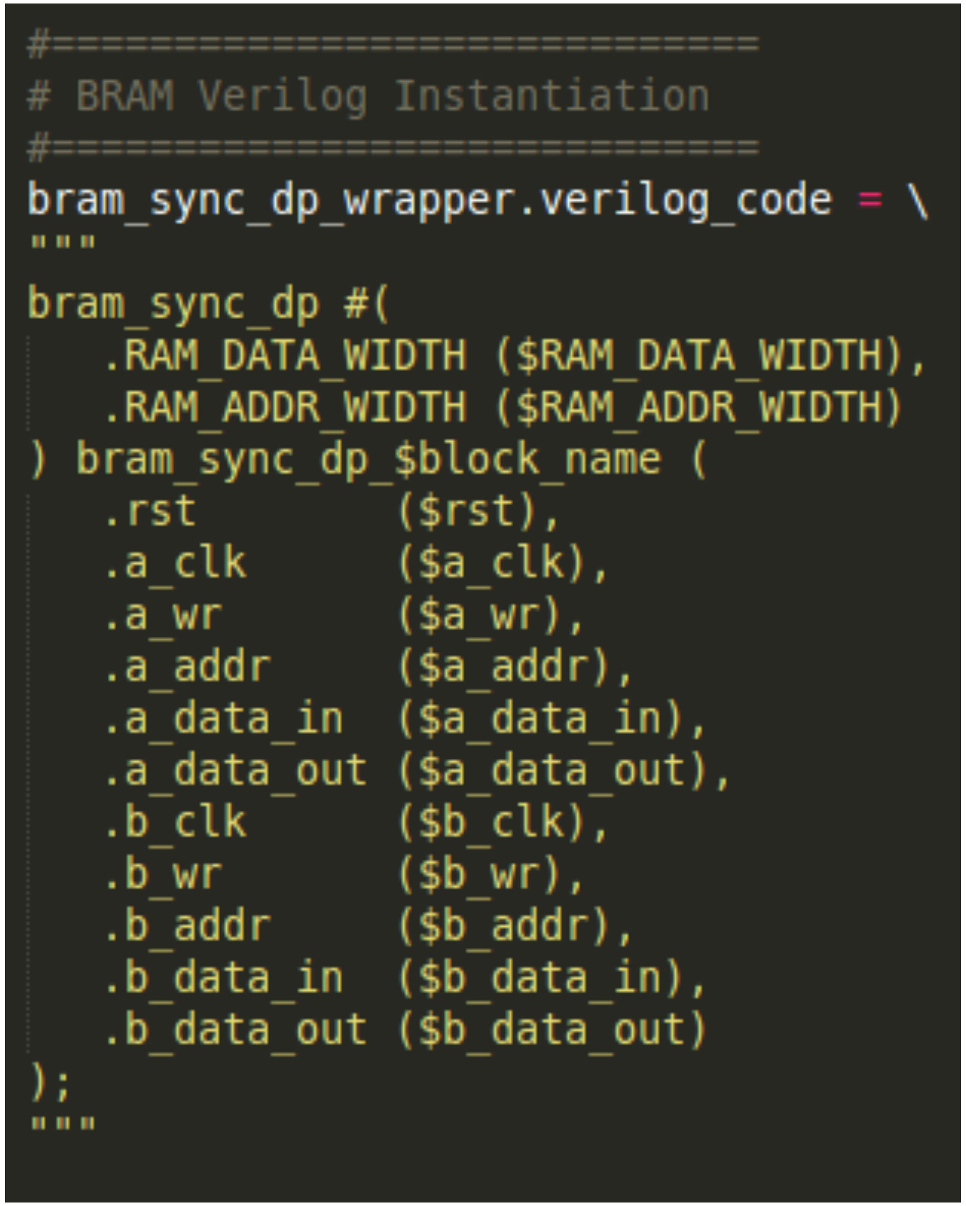
```python
mem = [Signal(intbv(0)[RAM_DATA_WIDTH:]) for i in range(2**RAM_ADDR_WIDTH)]

#===================
# Simulation Logic
#===================
# a_clk logic
#===================
@always(a_clk.posedge)
def a_logic():
    if rst:
        a_data_out.next = 0
    else:
        a_data_out.next = mem[a_addr.val]
        if a_wr:
            mem[a_addr.val] = a_data_in.val

#==============
# b_clk logic
#==============
@always(b_clk.posedge)
def b_logic():
    if rst:
        b_data_out.next = 0
    else:
        b_data_out.next = mem[b_addr.val]
        if b_wr:
            mem[b_addr.val] = b_data_in.val


# removes warning when converting to hdl

return a_logic, b_logic
```

MyHDL Example



```
//===============
// Local Params
//===============
localparam RAM_DATA_DEPTH = 2**RAM_ADDR_WIDTH;  // depth

//================
// Shared memory
//================
reg [RAM_DATA_WIDTH-1:0] mem [RAM_DATA_DEPTH-1:0];

//=========
// Port A
//=========
always @(posedge a_clk) begin
   if (`ifdef ACTIVE_LOW_RST !rst `else rst `endif)
      a_data_out <= {RAM_DATA_WIDTH{1'b0}};
   else begin
      a_data_out  <= mem[a_addr];
      if (a_wr) begin
         mem[a_addr] <= a_data_in;
      end
   end
end

//=========
// Port B
//=========
always @(posedge b_clk) begin
   if (`ifdef ACTIVE_LOW_RST !rst `else rst `endif)
      b_data_out <= {RAM_DATA_WIDTH{1'b0}};
   else begin
      b_data_out <= mem[b_addr];
      if (b_wr) begin
         mem[b_addr] <= b_data_in;
      end
   end
end
```

MyHDL Example



```
#==============================
# BRAM Verilog Instantiation
#==============================
bram_sync_dp_wrapper.verilog_code = \
"""
bram_sync_dp #(
    .RAM_DATA_WIDTH ($RAM_DATA_WIDTH),
    .RAM_ADDR_WIDTH ($RAM_ADDR_WIDTH)
) bram_sync_dp_$block_name (
    .rst         ($rst),
    .a_clk       ($a_clk),
    .a_wr        ($a_wr),
    .a_addr      ($a_addr),
    .a_data_in   ($a_data_in),
    .a_data_out  ($a_data_out),
    .b_clk       ($b_clk),
    .b_wr        ($b_wr),
    .b_addr      ($b_addr),
    .b_data_in   ($b_data_in),
    .b_data_out  ($b_data_out)
);
"""
```

MyHDL Architecture

Levels of Flexibility

- Parameterized modules
- Generate Statements
- Precompiler Directives
- Python Scripting (Redrawing)

Levels of Simulation

- Functional Verification
- Co-Simulation
- Bit-accurate Simulation (Via 3rd party software)

Conclusion

- CASPER is a successful and design flow
- But we need to keep up with the latest technologies and software
- MyHDL has some great methodologies but lacks a block diagram design environment
- HDL Coder very good product and would provides most of the features we need, although it isn't cheap.