CASPER Workshop

Tutorial 7: Implementation of a Green Block

Dev. By : Jason Manley, Mekhala Muley

Doc. By : Mekhala V. Muley gmrt/ncra/tifr

Expected completion time: 3hrs

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**	: **version 1.0 Rev 3 2009 ,** **uboot : uboot-2010-07-15-r3231-dram ,** **Linux Kernel Image : uImage-jiffy-20091110**

2 Introduction

In this tutorial, you will design a “green block” with a dynamic internal structure. This is done using system generator blocks, blocks specific to ROACH board and a matlab script.

The first step will be to design a block using system generator block and do the basic functionality check by simulating the design. Once the successful simulation is done, the design can be tested on actual hardware for which you will need to compile the design. The compilation will generate a .bof file which will be used for programming the board. The output from the actual hardware can be read and plotted by a python script. This ensures that the block is working as expected on the hardware.

The final step will be to make it a parameterised block and write a matlab script for dynamically changing internal structure of the block. At the end of this tutorial you will have a parameterised and scripted green block.

3 Background

Various blocks in the CASPER library are parameterised blocks with dynamic internal structure. For eg. consider the FFT block were the number of stages changes with the number of FFT points to be performed. For such blocks replicating the internal structure every time is not the efficient method. Hence a matlab script is written for redrawing such blocks where the internal structure is replicated depending upon the parameter value.

In this tutorial Coarse delay block will be designed. This delay block will be designed for simultaneous data inputs. The functionality of the coarse delay block is to delay the simultaneous input data stream by specified number of clock cycles. This is acheived by writing the input data samples in memory and reading them with an offset equal to the amount of delay required. This block has dynamic internal structure which can be constructed using a matlab script. The internal structure is replicated with increase in number of simulataneous input ports.

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 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 these cable connections and entries in the /etc/* files of the workshop PCs 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. This tutorial requires just the FPGA board and not the iADC board. The onboard clock of 100MHz will be used for this tutorial.

4. The controlling PC should have following softwares installed in it :

- MATLAB (2008a)
- ISE 11.5
- System Generator 11.5
- Latest CASPER library

5. 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 "[TUT7_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]/[TUT7_BOF_FILE]" to directly program (using the python script explained in "Data acquisition") 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]/[TUT7_MDL_FILE]".

6. **Start the matlab :**

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

5 Creating Your Design

In this tutorial you will design a coarse delay block. The function of the block is to delay the input data stream by specified number of clock cycles. This is acheived by writing the input data samples in memory and reading them with an offset equal to the amount of delay required.

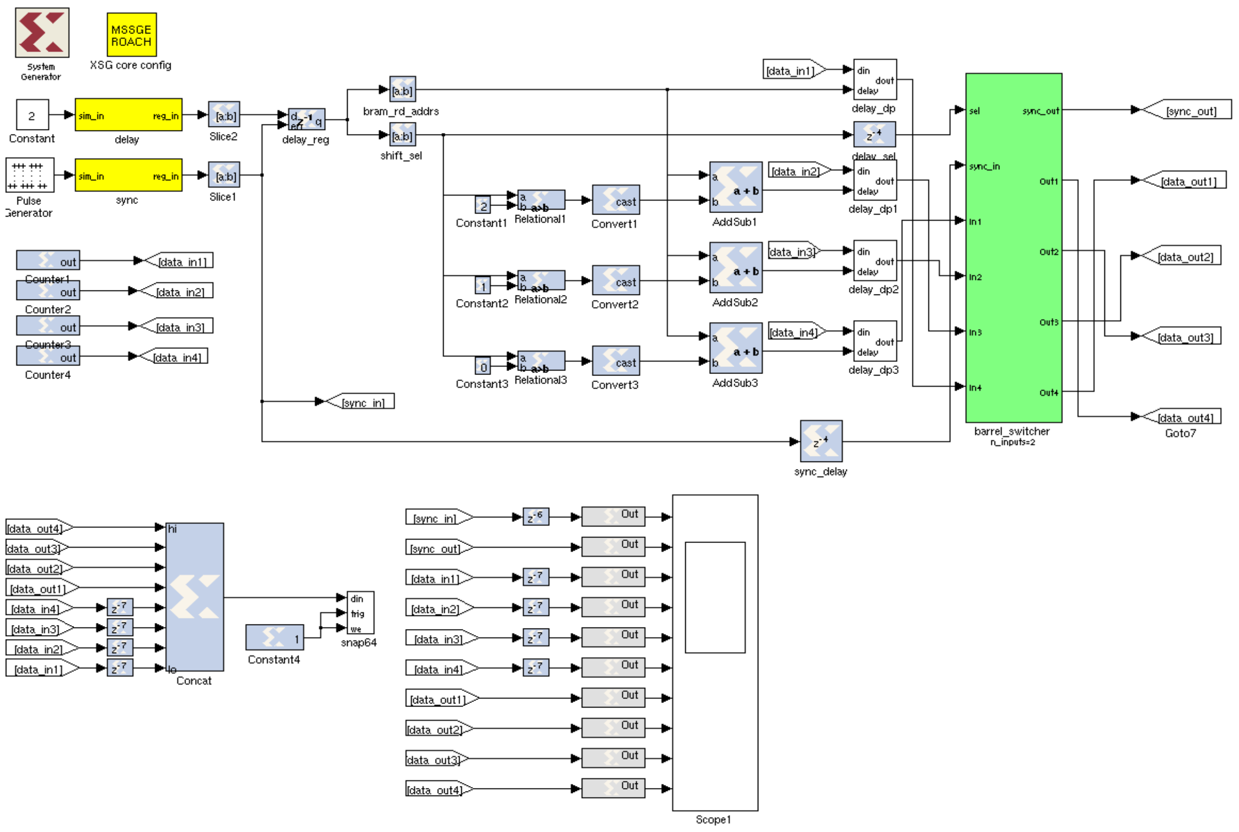
The block will have n-number of simultaneous input as well as output ports. The amount of delay is provided dynamically by the user through software register. Counter data is given as the input to the block (This is used for checking the functionality of the block). The basic functionality can be checked by simulating the design and checking the input as well as output data on the scope.The input data and the delayed output data can be stored in a BRAM using a snap block. This BRAM is read via python script to check the functionality of the block on actual hardware.

In the present case you will design a delay block with 4 simultaneous input port. The delay block will be designed to compensate maximum of 2048 clock cycles and this value is externally provided by you at the time of testing. As the simulataneous inputs are four, hence 4-BRAMs with 512 (i.e. 2048/4) locations are required to store the data. The internal logic is designed in such a way that the delays greater than 4 clock cycles are addressed by certain logic and delays less than 4 clock cycles are acheived via barrel shifter. A sync is provided to synchronize the delay block with rest of the logic in the design.

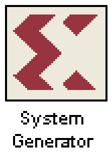
5.1 Create a new model:

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 file.

Add all the blocks from Simulink library browser and arrange them as shown in the figure below:



5.2 Add Xilinx System Generator and XSG Core Config blocks :



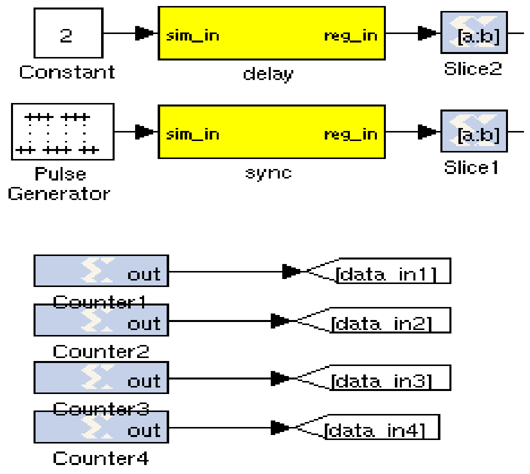
By now you should have used these blocks a number of times. Add *System Generator* block from *Xilinx Blockset -> Basic Elements -> System Generator* libray path. The settings can be left on default.

Also add *XSG core config* from *BEE_XPS System Blockset -> XSG core config*. Set its “Hardware Platform” to ROACH sx95t with “User IP Clock Source” as sys_clk which is an onboard 100MHz crystal. This block will configure the System Generator block which you have added previously. All hardware related blocks are yellow and can be found in the BEE_XPS library.

You need to add these two blocks for all the CASPER designs.

5.3 Inputs to the coarse delay block :

The inputs required to the coarse delay block are
a) simulataneous n input *data_in*.
b) a *sync* : for synchornisation of the coarse delay block with rest of the design
c) a *delay* : for the amount of delay to be compensated
d) an *enable* : for latching the delay value provided dynamically by the user at the sync boundary only.



5.3.1 Add the software registers

Add two *software registers* from *BEE_XPS System Blockset -> Software Register*. Set the I/O direction to “From Processor”. Rename those software registers as *sync* and *delay.*

Sync pulse will ensure the proper syncronization of the coarse delay block with rest of the logic in the design.

Delay is used to provide the delay values from the processor dynamically to the block.

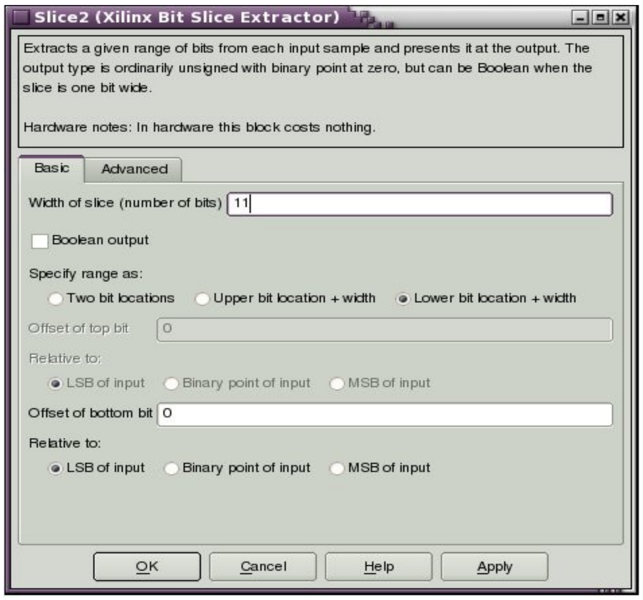
5.3.2 Add blocks for simulation

Add *Pulse Generator* from *Simulink->Sources->Pulse Generator*. Set Pulse type to “Sample based” , Period as “100” , Pulse Width as “1”. This is used for simulation. It simulates a sync pulse with a period of 100 clock cycle and pulse width of 1 clock cycle.

Add *Constant* from *Simulink->Sources->Constant*. Set Constant value to “2”. This is used for simulation. It will delay the input data stream by 2 clock cycles.

5.3.3 Add slice blocks

Add two *slice* blocks from *Xilinx Blockset-> Basic Elements->Slice*. Rename them as *Slice1* and *Slice2.* Set properties of *Slice1* to “Boolean Output”, as the data type of *Sync* input is boolean. Set the properties of *Slice2* as shown in the figure below :


Slice2 (Xilinx Bit Slice Extractor)

Extracts a given range of bits from each input sample and presents it at the output. The output type is ordinarily unsigned with binary point at zero, but can be Boolean when the slice is one bit wide.

Hardware notes: In hardware this block costs nothing.

Basic | Advanced

Width of slice (number of bits): 11

☐ Boolean output

Specify range as: Two bit locations | Upper bit location + width | [✓ Lower bit location + width]

Offset of top bit: 0

Relative to: [✓ LSB of input] | Binary point of input | MSB of input

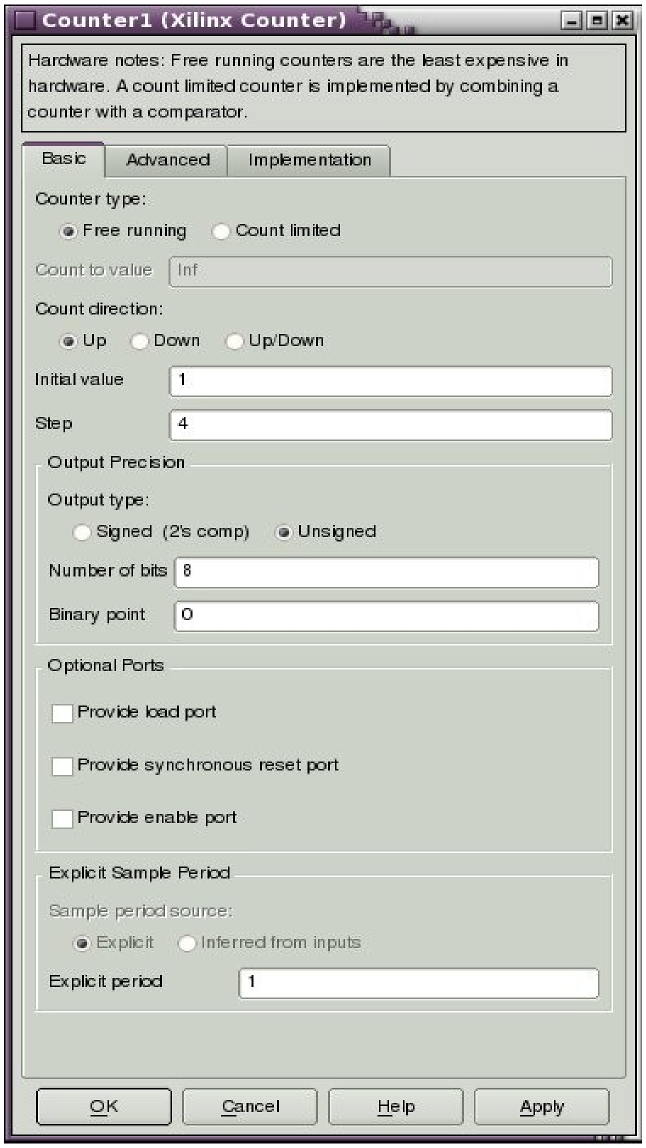
Offset of bottom bit: 0

Relative to: [✓ LSB of input] | Binary point of input | MSB of input

The maximum delay that can be compensated is 2048 i.e. 2^11 hence the bitwidth of the slice is set to 11.

5.3.4 Add Counters

Add four *Counter* blocks from *Xilinx Blockset-> Basic Elements->Counter*. Rename them as *Counter1, Counter2, Counter3, Counter4*. Set the properties of *Counter1* as shown in the figure below:



Counter1 (Xilinx Counter)

Hardware notes: Free running counters are the least expensive in hardware. A count limited counter is implemented by combining a counter with a comparator.

Counter type: [✓ Free running] Count limited

Count to value: Inf

Count direction: [✓ Up] Down Up/Down

Initial value: 1

Step: 4

Output Precision

Output type: Signed (2's comp) [✓ Unsigned]

Number of bits: 8

Binary point: 0

Optional Ports

- [ ] Provide load port
- [ ] Provide synchronous reset port
- [ ] Provide enable port

Explicit Sample Period

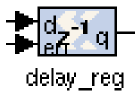
Sample period source: [✓ Explicit] Inferred from inputs

Explicit period: 1

Set all the properties for rest of the counters similar to that of *Counter1* except for “Initial value” . Set 2, 3 and 4 as the “Initial Value” for *Counter2, Counter3* and *Counter4* respectively.

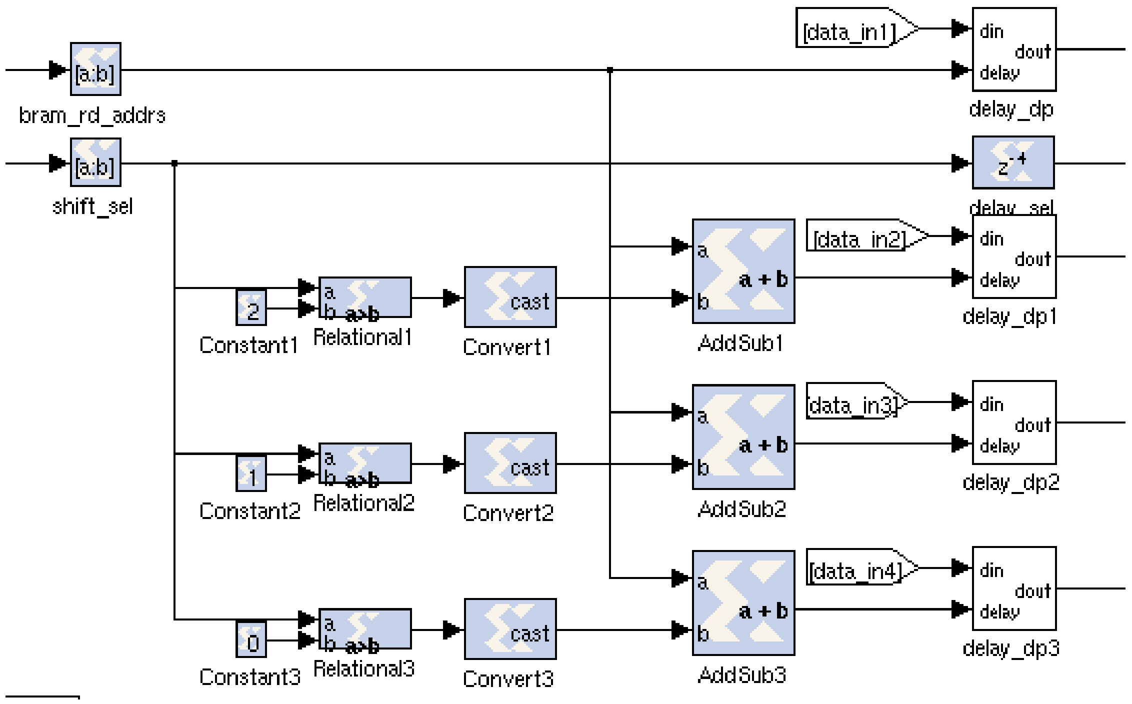
5.4 Latch the delay value :

Add a *register* from *Xilinx Blockset-> Basic Elements->Register* and rename it to *delay_reg*. Select Provide enable port.



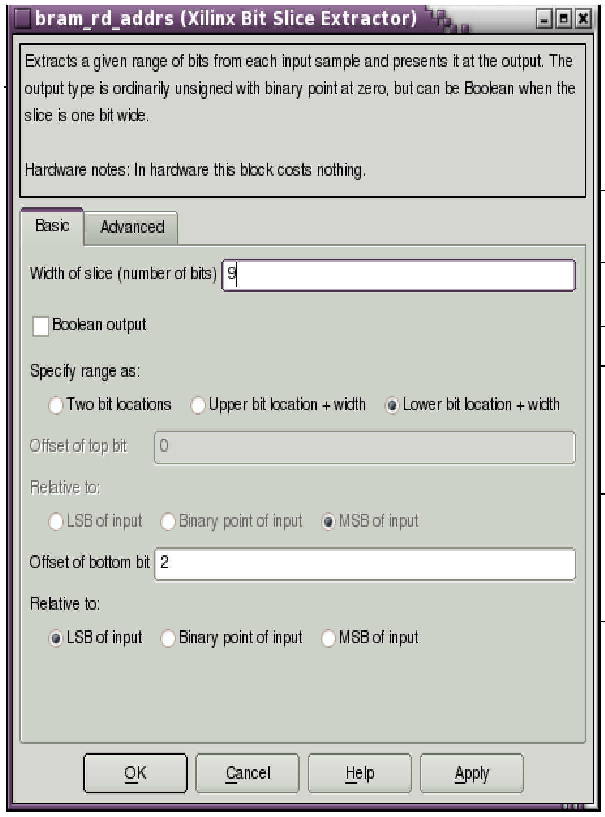
This register is used to latch the delay only when enable (which is connected to sync pulse) goes from low to high.

5.5 Logic for delay compensation greater than 4 clock cycles :



5.5.1 Add slice blocks

Add two *slice* blocks from *Xilinx Blockset-> Basic Elements->Slice*. Rename them to *bram_rd_addrs* and *bs_sel* and set the properties as shown in the figure below:



bram_rd_addrs (Xilinx Bit Slice Extractor)

Extracts a given range of bits from each input sample and presents it at the output. The output type is ordinarily unsigned with binary point at zero, but can be Boolean when the slice is one bit wide.

Hardware notes: In hardware this block costs nothing.

Basic | Advanced

Width of slice (number of bits): 9

Boolean output: ☐

Specify range as: Two bit locations / Upper bit location + width / [✓ Lower bit location + width]

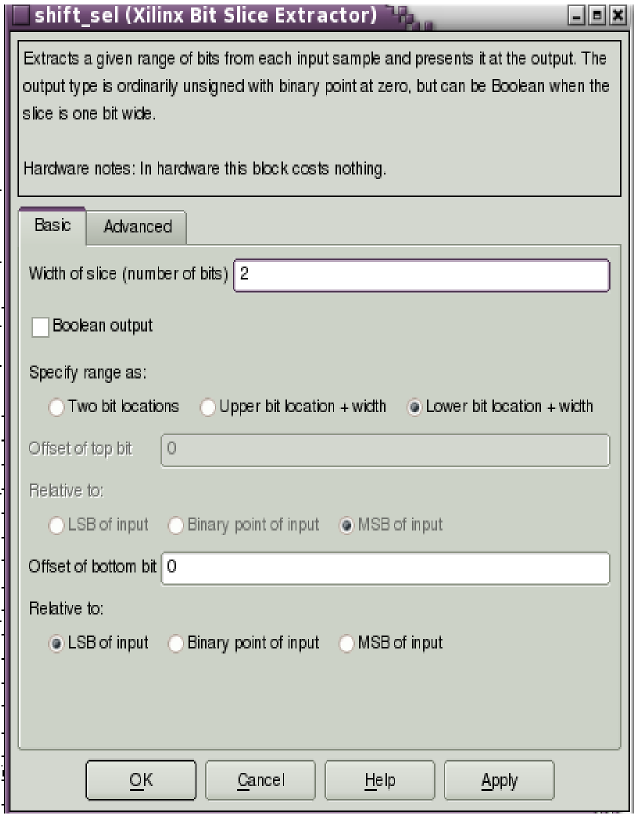
Offset of top bit: 0

Relative to: LSB of input / Binary point of input / [✓ MSB of input]

Offset of bottom bit: 2

Relative to: [✓ LSB of input] / Binary point of input / MSB of input

OK | Cancel | Help | Apply



shift_sel (Xilinx Bit Slice Extractor)

Extracts a given range of bits from each input sample and presents it at the output. The output type is ordinarily unsigned with binary point at zero, but can be Boolean when the slice is one bit wide.

Hardware notes: In hardware this block costs nothing.

Basic | Advanced

Width of slice (number of bits): 2

Boolean output: ☐

Specify range as: Two bit locations / Upper bit location + width / [✓ Lower bit location + width]

Offset of top bit: 0

Relative to: LSB of input / Binary point of input / [✓ MSB of input]

Offset of bottom bit: 0

Relative to: [✓ LSB of input] / Binary point of input / MSB of input

OK | Cancel | Help | Apply

As the data is coming in four parallel stream, 4 BRAMs are required to store the data. The upper MSBs of *delay*(software register) sliced by *bram_rd_addrs* is used as the address of the BRAM for data read out. The lower bits of *delay* (software register) sliced by *shift_sel* is used as the select input for the barrel shifter.

5.5.2 Add constant block

Add three *constant* blocks from *Xilinx Blockset-> Basic Elements->Constant*. Rename them to *Constant1, Constant2 and Constant3*. Set Type to “Unsigned” and Number of bits to “2” for all of them. Set Constant value to “2” for *Constant1*, Constant value to “1” for *Constant2* and Constant value to “0” for *Constant3*.

5.5.3 Add relational operator

Add three *relational* block from *Xilinx Blockset-> Basic Elements->Relational*. Set Comparison to “a>b” for all the blocks.

5.5.4 Add convert block

Add three *convert* blocks from *Xilinx Blockset-> Basic Elements->Converter*. Set Type to “Unsigned”, Number of bits to “1” for all the blocks.

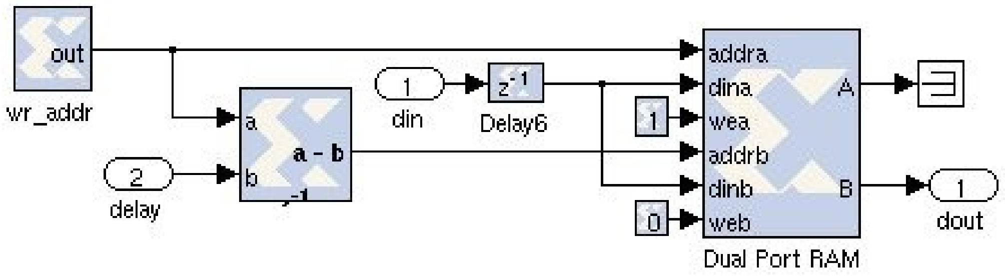
5.5.5 Add Adder block

Add three *adder* block from *Xilinx Blockset->Math->AddSub*. Go to Output Type tab of the block. Select Precision to “Full”.

5.6 Create a subsytem for storing input data stream :

Storing of the data in memory can be implemented using single port RAM or dual port RAM. Depending upon the need the user can select the type of RAM available.

5.6.1 DUAL PORT RAM



5.6.1.1 Add Counter

Add *counter* block from *Xilinx Blockset-> Basic Elements->Counter*. Rename it to *wr_addr*. Set Counter type to “Free running”, Count Direction to “Up”, Initial Vaule to “0”, Step to “1”, Output Type to “Unsigned”, Number of bits to “9”, Binary point to “0”. This counter is used to generate the address for writing the data into dual port RAM of 512 i.e. 2^9 locations.

5.6.1.2 Add substractor block

Add a *substractor* block from *Xilinx Blockset->Math->AddSub*. Go to Basic Tab , set Operation to “Subtraction”. Goto Output Type tab set Precision to “User Defined”, Output Type to “Unsigned”, Number of bits to “9”, binary point to “0”. It basically generates the read address from the write address offseting it by delay value.

5.6.1.3 Add delay and constant blocks

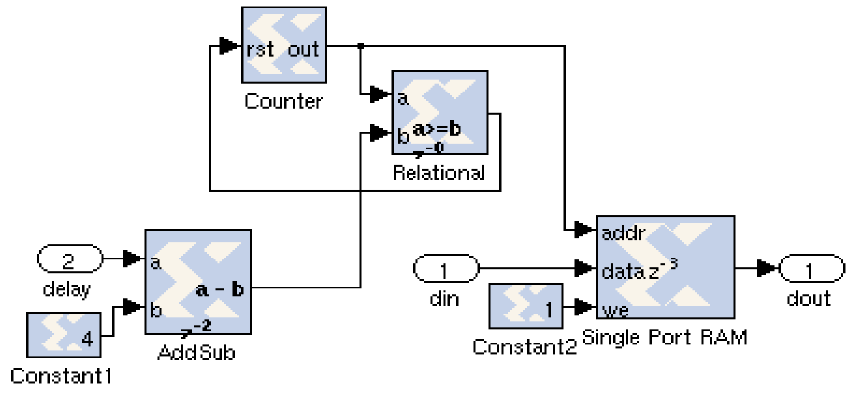
Add a *delay* block from *Xilinx Blockset-> Basic Elements->Delay*. Set the Latency to “1”.

Add two *constant* blocks from *Xilinx Blockset-> Basic Elements->Constant*. Rename it to *Constant1* and set Type to “Boolean”, Constant value to “1”. This block is used to keep the write enable of port A always high for the dual port RAM. Rename the another block to *Constant2* and set Type to “Boolean”, Constant value to “0”. This block is used to keep port B always high i.e. port B will be configured as the reading port of dual port RAM.

5.6.1.4 Add Dual port RAM

Add a *Dual Port RAM* from *Xilinx Blockset->Memory->Dual Port RAM*. Goto Basic tab and set Depth to “512”, Memory Type to “Block RAM”, Latency to “3”. This is used for storing the input data stream.

5.6.2 SINGLE PORT RAM



5.6.2.2 Add a Constant block

Add a *constant* block from *Xilinx Blockset-> Basic Elements-> Constant.* Set Type to "Unsigned", Constant value to "4" and Number of bits to "11".

5.6.2.3 Add substractor block

Add a *substractor* block from *Xilinx Blockset->Math->AddSub*. Go to Basic Tab , set Operation to "Subtraction". Goto Output Type tab set Precision to "User Defined", Output Type to "Unsigned", Number of bits to "9", binary point to "0".

5.6.2.4 Add Relational block

Add *relational* block from *Xilinx Blockset-> Basic Elements-> Relational.*Set Comparison to "a>=b".

All the above logic is used to generate the read and write address for the RAM.

5.6.2.5 Add a constant block

Add a *constant* block from *Xilinx Blockset-> Basic Elements-> Constant.* Set Type to "Boolean" and constant value to "1". This indicates that the data is written into RAM at every clock cycle.

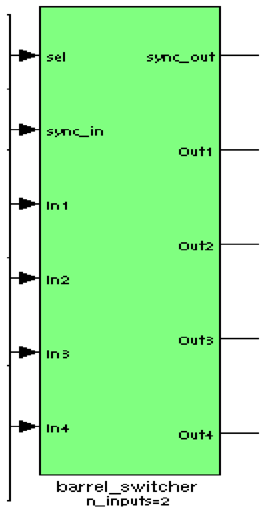
5.6.2.6 Add Single Port RAM

Add a *single port RAM* from *Xilinx Blockset->Memory->Single Port RAM.* Set Depth to 512, Memory Type to "Block RAM", Write Mode to "Read before write" and latency to "3".

5.7 Logic for delay compensation less than 4 clock cycles :

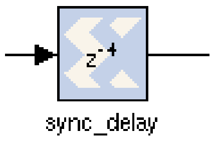
Add Barrel Shifter block

Add a *barrel shifter* block from *CASPER DSP Blockset -> Reorders-> barrel_switcher*. Set the Number of inputs to "2". Barrel shifter basically shifts the data by a specified number in one clock cycle. This is used to acheive the delay compensation less than 4 clock cycles.



5.8 Add a delay block :

Add a *delay* block from *Xilinx Blockset-> Basic Elements->Delay* and rename it to *sync_delay*. Set its latency to "4". This is used to delay the sync pulse in order to match with the pipeline latency encounter by the data within the block.



5.9 Outputs from the coarse delay block :

The outputs from the coarse delay block are *sync_out* and simultaneous n number of *data_out*.

5.10 Add signal routing blocks :

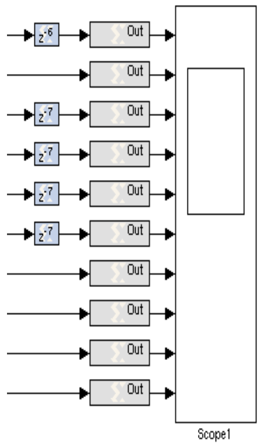
5.10.1 Add goto blocks

Add 10 signal routing *goto* block from *Simulink->Signal Routing-> GoTo*. Rename them to *sync_in,data_in1,data_in2,data_in3,data_in4,sync_out,data_out1,data_out2,data_out3,data_out4*. These are used for signal routing.

5.10.2 Add from blocks

Add 10 signal routing *From* blocks from *Simulink->Signal Routing-> From*. Rename them to *sync_in,data_in1,data_in2,data_in3,data_in4,sync_out,data_out1,data_out2,data_out3,data_out4*. These are used for signal routing.

5.11 Add blocks for simulation:



5.11.1 Add scope

Once all the blocks are dded you need to check the functionality of design. This done by simulating the design. The simulated output from the design can be viewd using a scope. Add *Scope1* from *Simulink->Sinks->Scope*.

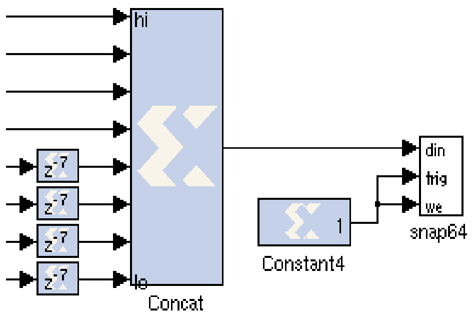
5.11.2 Add delay blocks

Add 5 *delay* blocks from *Xilinx Blockset-> Basic Elements->Delay*. Set the latency of the four blocks to 7 and connect them to data_in signal routing blocks. Set the latency of the fifth delay block to 6 and connect it to sync_in signal routing block.

5.11.3 Add gatewayout blocks

Add 10 *gatewayout* blocks from *Xilinx Blockset -> Basic Elements -> Gateway Out*. Uncheck the “Translate into output port”.

5.12 Add blocks to check the behaviour of the coarse delay block on the hardware :



5.12.1 Add snap block

Add *snap* block from *CASPER DSP Blockset -> Scopes ->snap64*. Snap block is used to capture the data from the FPGA and make it accessible to the user. The block stores the 64 bit data when the ctrl signal goes from low to high. Hence this block is used to check the behaviour of the delay block on actual hardware.

5.12.2 Add delay blocks

Add 4 *delay* blocks from *Xilinx Blockset-> Basic Elements->Delay*. Set the latency to 7 and connect them to data_in signal routing blocks. These are used to match with the pipeline latency encounter by the data stream in the block.

5.12.3 Add a concat block

Add a *concat* block from *Xilinx Blockset-> Basic Elements-> Concat*. Set number of inputs to 8. This is used to give four 8-bit data_in and four 8-bit data_out concatenated together and give it to the snap blocks datain input port.

5.12.4 Add a constant block

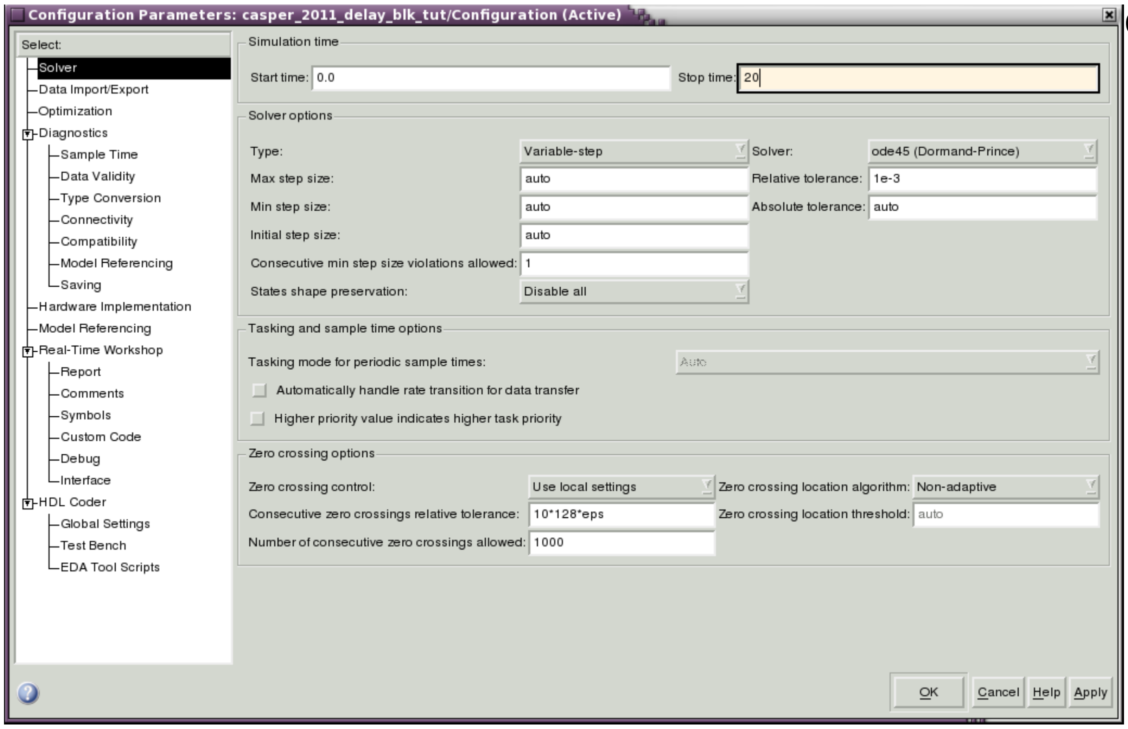
Add a *constant* block from *Xilinx Blockset-> Basic Elements-> Constant*. Set Type to “Boolean” and Constant value to “1”.

6 Simulation

Once the basic design has been created the next step is to simulate it for checking the functionality of the coarse delay block. Simulation results shows whether the output is as expected or not. If not suitable corrections can be made before moving for the next step.

6.1 Add blocks to check the behaviour of the coarse delay block on the hardware :

Goto *Simulation->Configuration paramters*. Set the stop time to 20. This will give simulated output for 20 time stamps.



Simulation time

Start time: 0.0 Stop time: 20

Solver options

Type: Variable-step Solver: ode45 (Dormand-Prince)
Max step size: auto Relative tolerance: 1e-3
Min step size: auto Absolute tolerance: auto
Initial step size: auto
Consecutive min step size violations allowed: 1
States shape preservation: Disable all

Tasking and sample time options

Tasking mode for periodic sample times: Auto
Automatically handle rate transition for data transfer
Higher priority value indicates higher task priority

Zero crossing options

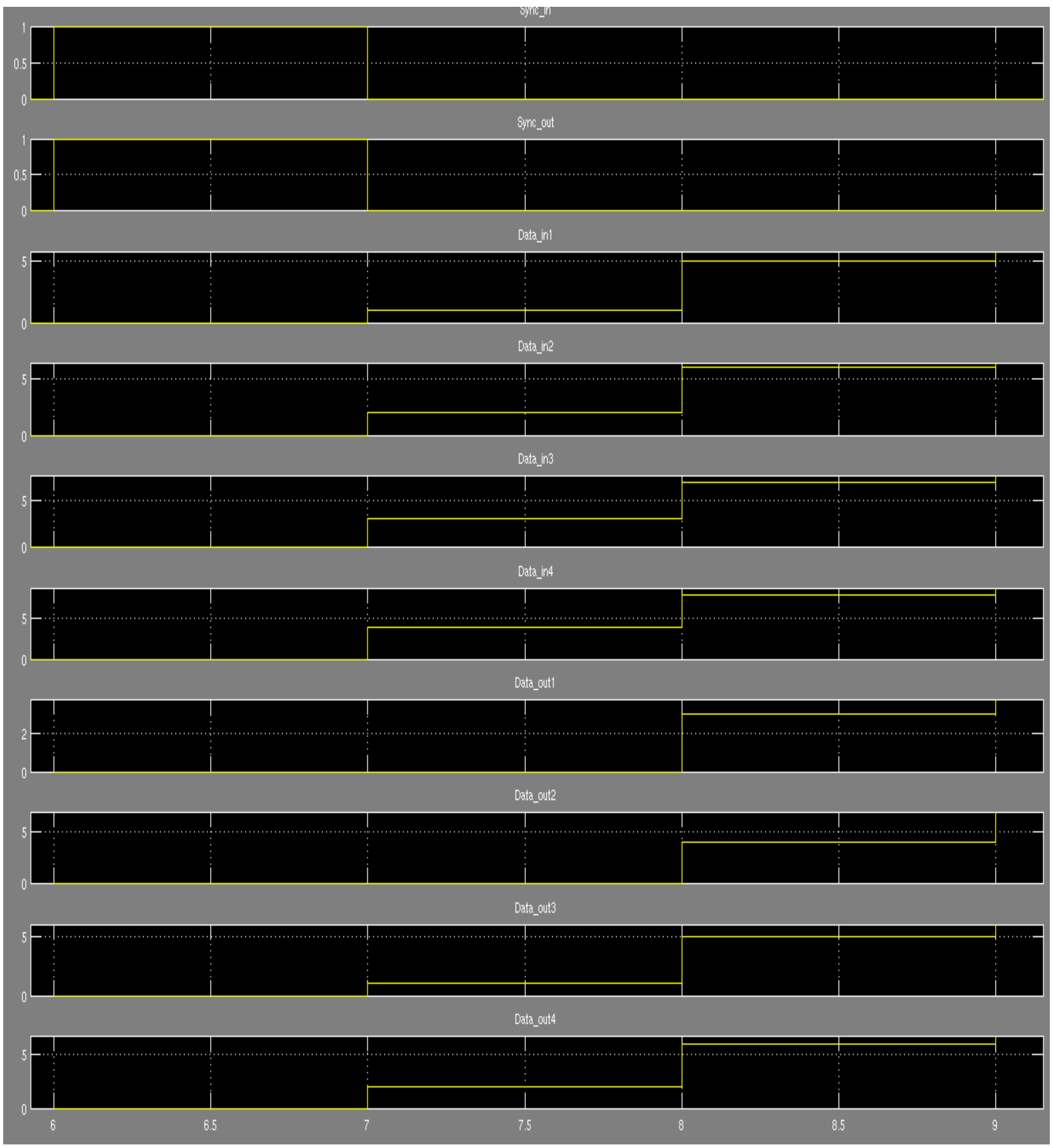
Zero crossing control: Use local settings Zero crossing location algorithm: Non-adaptive
Consecutive zero crossings relative tolerance: 10*128*eps Zero crossing location threshold: auto
Number of consecutive zero crossings allowed: 1000

6.2 Run Simulation

Goto *Simulation->Start*. This will simulate the design for 20 timestamps. And the results can be viewd in the Scope 1.

6.3 Simulation result

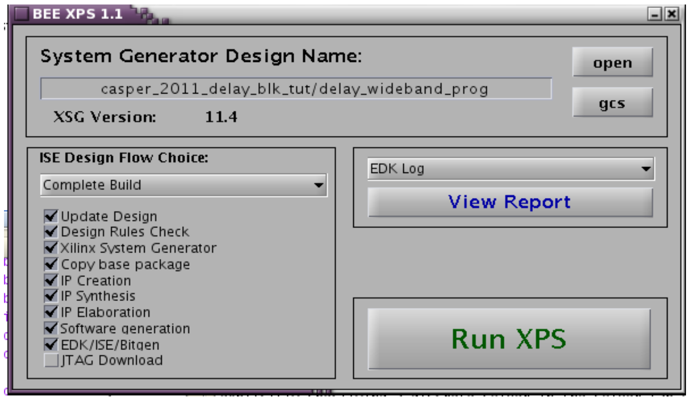
The simulation result shows that the data_in is delayed by 2 clock cycles. Hence data_in1 i.e port1 (connected to 3rd Axes labled as Data_in1) is available at the output port3 (which is connected to the 7th Axes labled as Data_out3) hence showing that the input is delayed by 2 clock cycles.


Sync_in
Sync_out
Data_in1
Data_in2
Data_in3
Data_in4
Data_out1
Data_out2
Data_out3
Data_out4
6
6.5
7
7.5
8
8.5
9

7 Testing of the block on ROACH

7.1 Compilation of the design

Compile the design to test it on the hardware. Goto matlab command window and run the command *bee_xps*. It will open the following window



BEE XPS 1.1

System Generator Design Name: casper_2011_delay_blk_tut/delay_wideband_prog

XSG Version: 11.4

ISE Design Flow Choice: Complete Build

- [✓] Update Design
- [✓] Design Rules Check
- [✓] Xilinx System Generator
- [✓] Copy base package
- [✓] IP Creation
- [✓] IP Synthesis
- [✓] IP Elaboration
- [✓] Software generation
- [✓] EDK/ISE/Bitgen
- [ ] JTAG Download

Make sure the file displayed in the pop-up is correct and then click the <i>Run XPS</i> button. 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 copy this .bof file at location [FPGA_PROG_BOF_DIR] after changing the permissions of the file.

eg. for the bof file [TUT7_BOF_FILE] in the area [STD_BOF_DIR]

```
$ chmod a+x [STD_BOF_DIR]/[TUT7_BOF_FILE]
```

```
$ cp [STD_BOF_DIR]/[TUT7_BOF_FILE] [FPGA_PROG_BOF_DIR]
```

7.2 Data aquisition

A script is written in python to communicate witht the ROACH board.This script programs the board, sets the delay from command line and plots the input and delayed output data stream from the BRAM. Run the python script file **"[TUT7_PYSCRIPT_FILE]"** saved in the following location : **"[STD_PYSCRIPT_DIR]"**

Usage : [STD_PYSCRIPT_DIR]/[TUT7_PYSCRIPT_FILE] *roach_name* -b [bof_file] -d <delay in number of clock cycles>

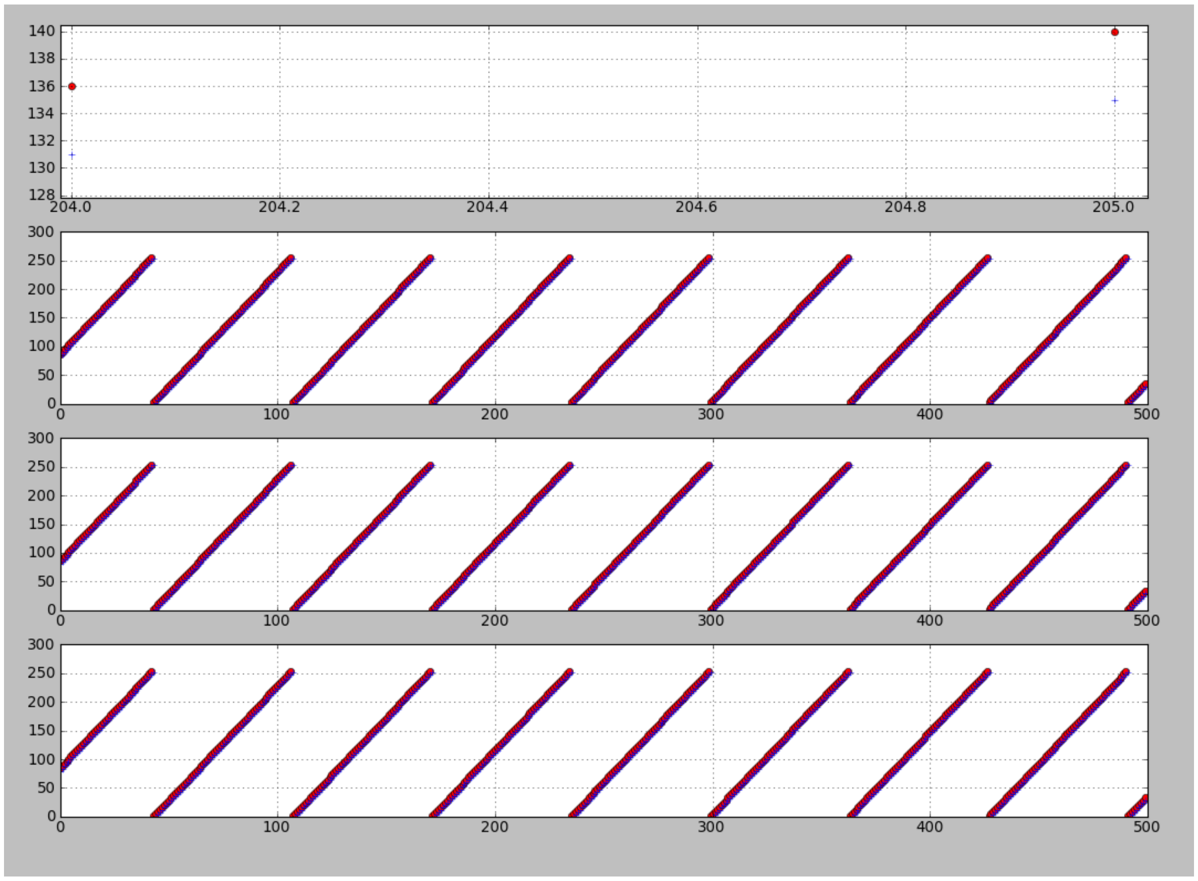
Enter the corresponding Location/File names and roach name/IP. This will program the Roach board with the above BORPH file as well as set the delay of 5 clock cycles.

For eg.

$ [STD_PYSCRIPT_DIR]/[TUT7_PYSCRIPT_FILE] roach030172 -b [tut7_coarse_delay_blk_2011_Sep_26_1250.bof] -d 5

7.3 Plots

The plot shows the data_in and data_out. Hence there are four Axes representing 4 simultaneous data streams. The first axis shows the zoomed data points of data_in1 and data_out1. The Red Dot is for data_in1 and the Blue Plus sign is for data_out1. The difference between the data in and data out is exactly the same as provided i.e of 5 clock cycle.



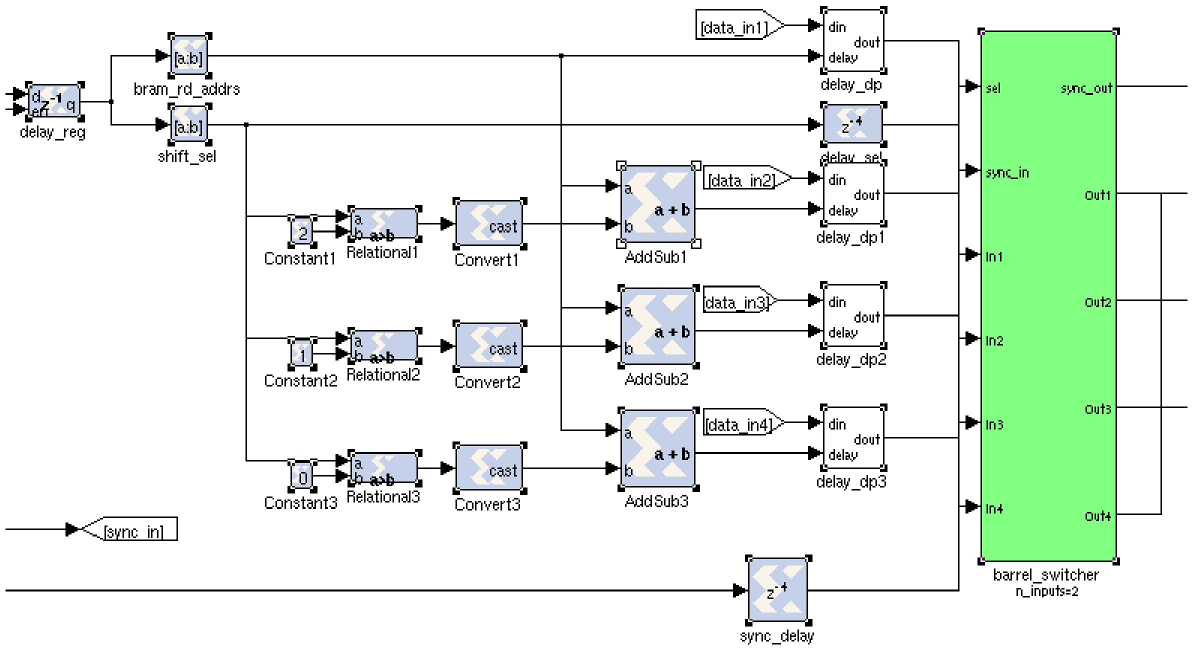
8 Writing a MATLAB script

After the functionality has been checked and verified now you can write a matlab script for the block.

8.1 Masking the block

8.1.1 Create Subsystem

Select all the blocks connected between input and output ports of the coarse delay block as shown in the figure below. Right clcik and select "Create Subsystem". Rename the block as <your_initials>*_coarse_delay_wideband_prog.*

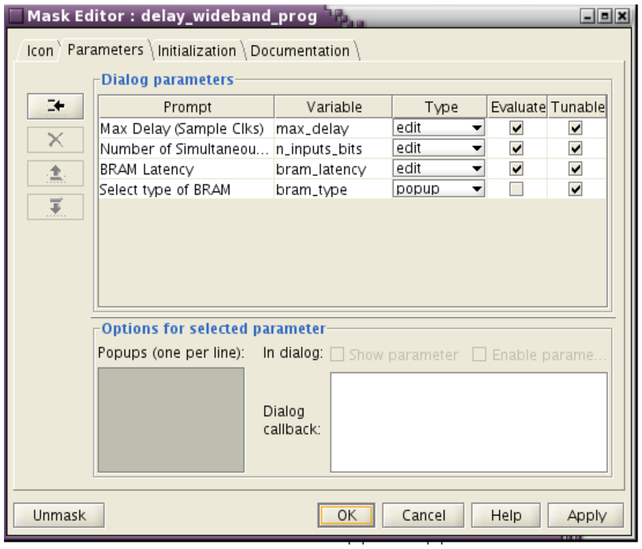


8.1.2 Rename the I/O ports

Rename the input ports as *sync, delay, en, data_in1, data_in2, data_in3 and data_in4*. Similarly rename the output ports as *sync_out, data_out1,data_out2, data_out3 and data_out4*.

8.1.3 Add Paramters

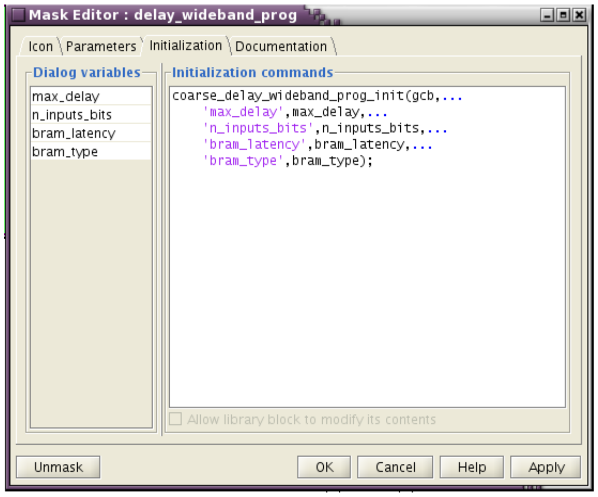
Right click and select “Edit mask”. Following window will open. Go to Parameters tab and add various parameters and corresponding paramter value to it as shown in the figure below:



Prompt	Variable	Type	Evaluate	Tunable
Max Delay (Sample Clks)	max_delay	edit	✓	✓
Number of Simultaneou...	n_inputs_bits	edit	✓	✓
BRAM Latency	bram_latency	edit	✓	✓
Select type of BRAM	bram_type	popup	☐	✓

8.1.4 Function Call

Go to initialization tab and from the initialization commands call the matlab function. The function name should be *<block_name>_init* in our case it will be <your_initials>*_coarse_delay_wideband_prog_init*.
Pass the mask parameters and parameter value to the initialisation function in (*'parameter_name'* , *parameter_value*) pairs where '*parameter_name*' is a text string containing the name of the paramter and *parameter_value* is the associated value as shown in the figure below:



Dialog variables:

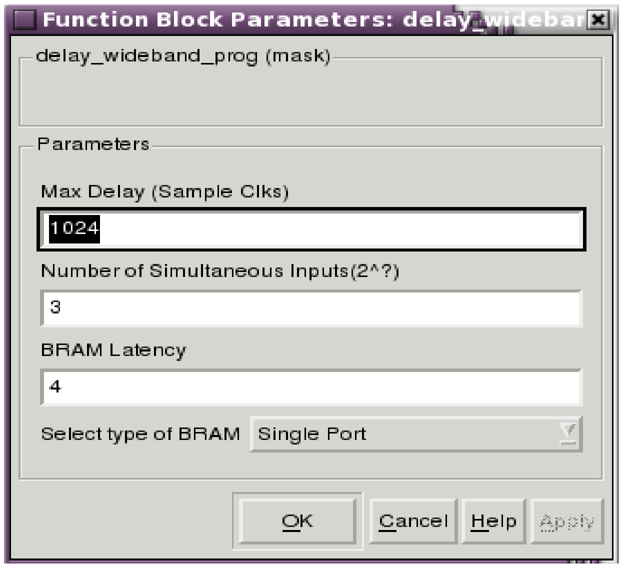
- max_delay
- n_inputs_bits
- bram_latency
- bram_type

```
coarse_delay_wideband_prog_init(gcb,...
    'max_delay',max_delay,...
    'n_inputs_bits',n_inputs_bits,...
    'bram_latency',bram_latency,...
    'bram_type',bram_type);
```

8.1.5 Mask Parameters

Go to documentation tab and write the block name i.e. <your_intials>*_coarse_delay_wideband_prog* in the Mask Type section.

The above steps will pop-up a window when the user double-clicks the block as shown in the figure below:


Function Block Parameters: delay_wideban

delay_wideband_prog (mask)

Parameters

Max Delay (Sample Clks)

1024

Number of Simultaneous Inputs(2^?)

3

BRAM Latency

4

Select type of BRAM Single Port

8.2 Adding the block to the library

Add this coarse delay block to the *CASPER DSP Blockset -> Delays* library. First Open the library then gotoSimulink library browser *Edit-> Unlock Library*. Now paste the block in that library.

8.3 Write the script

Refer to the delay_wideband_prog.m file at the location "[CASPER_LIB_DIR]".

Step 1 : The file name should be *<block_name>_init.m* i.e. <your_initials>*_coarse_delay_wideband_prog_init.m* and save it to the **"[CASPER_LIB_DIR]"**.The initialization function should be <your_name>_*<block_name>_init*.

Step 2 : Write the Comments describing the purpose of the function and how to use it.

Step 3 : Follwoing are the various functions used to write the script for the block
"same_state" : It checks whether the parameters are changed or not. If they are changed then only it will redraw otherwise it will exit.

"check_mask_type" : This is a sanity check . This is used to check that the initialisation script is being called to operate on the appropriate block. This name is mentioed under the Documentation tab in Mask editor for the block.

"munge_block" : The call to this function causes link to the original library block to be disabled if the block is not to be instantiated in a library. This causes changes to local and not to propagate to the library.

"get_var" : This is used to extract the various parameter values passed to the function.

“delete_lines” : It removes all the connections in the existing system.

“reuse_block” : This searches for an existing block of the same type and sets the necessary paramters. Use get_param(gcb,'ObjectParameters') and get_param(gcb,ParamterName) function to get all the necessary paramters of the block. get_param(gcb,'ObjectParameters') function will give name of all the parameter where “gcb” gets the current active block. get_param(gcb,ParamterName) function will return the value corresponding to the ParameterName.

“add_line” : This connects the output ports of one block to the input port of another block.

“clean_blocks” : A call to clean_blocks after all the blocks are instantiated and connected, ensures unconnected blocks are removed.

“save_state” : Each Simulink block has a parameter called “UserData” for storage of data by the useful. The call to save_state causes a data structure to be saved to this parameter. This structure consists of two fields viz state and parameter. State contains a hash of the input arguments (used by the same_state to determine if these have changed). Parameters contains a data structure with all the mask parameters passed to the initialisation function. save_state also update the mask paramters and overwrites variable references with actual values. This forces mask values to change to values passed into the script. If mask values do not change, Simulink does not execute the initialisation script so this also forces a change in mask values as variables are expanded so that future change in these variables cause execution of the initialisation script.

Note : For more detail description on script writing visit following link https://casper.berkeley.edu/wiki/DSP_Block_Standardisation

8.4 Checking for the syntactical/logical error if any in the script

Call the function from the Matlab Command Window. It will list the error if any in the command window. This helps in debugging the script before actually using it with the block.

For eg. Execute the function from matlab command window
delay_wideband_prog_init(gcb,'max_delay',1024,'n_inputs_bits',3,'bram_latency',4,'bram_type','Dual Port')

8.5 Use the block from the library

Pick and place the block from the library to the new model file. Now change few paramters of it. It will give a call to the initialisation script and redraw the block according to the input parameters. You can see that the internal structure of the block changes with the change in input parameter.

9 Conclusion

You have completed the tutorial on designing of the green block. You can now design any block whose internal structure varies dynamically depending upon the masking paramters.