

525.742

Mini Topic

Version Control for FPGA Designs

Version Control of FPGA Designs

- Having a system to control changes/revisions is basically non-negotiable in any design team with multiple designers and a changing design
 - Track who changed what and when
 - Be able to return to various points in the design and branch off from there.
 - Endless other benefits that are clear to software developers
 - Essentially copies of every design you've ever made, and *notes about what changed between them.*
- Version control of the FPGA design is not as straightforward as that of a pure software design
 - Lots and lots of generated files
 - Vendor specific non-text (or not comprehensible text) “source”
 - You “can” just take an entire Xilinx design and put it into version control software, but it will likely not be very useful other than for a backup system

IP Versions

The screenshot shows the 'Review and Package' step of the IP Packaging Wizard. On the left, a 'Packaging Steps' sidebar lists: Identification (checked), Compatibility (checked), File Groups (icon), Customization Parameters (icon), Ports and Interfaces (icon), Addressing and Memory (checked), Customization GUI (icon), and Review and Package (highlighted). The main area has a yellow banner stating 'IP has been modified.' Below this is a 'Summary' section with the following details: Display name: npr_chan_v1_0, Description: npr_chan_v1_0, and Root directory: /home/dougwen/testing_repo/tp2fpga/ip/tp2_fpga_engines/npr_chan_4096. At the bottom is an 'Edit IP Project Changes' section with four entries, each with a 'Merge changes' link: 'Merge project changes to File Groups Wizard', 'Merge project changes to Customization Parameters Wizard', 'Merge project changes to Ports and Interfaces Wizard', and 'Merge project changes to Customization GUI Wizard'.

- Xilinx does have a good way to manage multiple different versions of packaged IP blocks.
- IP can be packaged with an appropriate version many times, and multiple different copies of the IP can be kept in the repo
 - Users of the IP will be informed of the availability of the new version in their top level designs, but not required to update
- This isn't revision control by any means, but it does provide a nice infrastructure for updating IP without worrying about breaking existing designs
- Just copy the whole packaged IP directory and start editing whatever you want, repackage with a different version and the IP repository management software will understand
 - Designs can continue to use the old version of the IP, but will be prompted by vivado telling that a newer version is available

Traditional RCS (git, SVN...etc.)

- Fundamental problem is that VHDL source code in and of itself doesn't solely define the project.
 - Block designs?
 - Project Settings?
 - There are TONS of these right?
 - The “source” for block designs and project settings is basically some crazy XML file which isn't useful for you to read. So, while version controlling these files could be a solution, you'd lose that ability to glance at the diff between two versions and see what is actually different.
- A very practical method is to create scripts for generating the project, (including building the block designs), and version control those scripts. (Like what we did in the radio-periph-lab)
- Note : many very well established projects / design teams will forgo the GUI altogether and the entire design flow will be script based.

https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_2/ug1198-vivado-revision-control-tutorial.pdf

Example Block-Design Script Pieces

```
# Create instance: axi_gpio_0, and set properties
set axi_gpio_0 [ create_bd_cell -type ip -vlnv xilinx.com:ip:axi_gpio:2.0 axi_gpio_0 ]
set_property -dict [ list \
    CONFIG.C_ALL_OUTPUTS {0} \
    CONFIG.C_DOUT_DEFAULT {0x00000000} \
    CONFIG.C_GPIO_WIDTH {32} \
] $axi_gpio_0
```

```
connect_bd_net -net rx_clk_in_n_1 [get_bd_pins rx_clk_in_n] [get_bd_pins axi_ad9361_0/rx_clk_in_n]
connect_bd_net -net rx_clk_in_p_1 [get_bd_pins rx_clk_in_p] [get_bd_pins axi_ad9361_0/rx_clk_in_p]
connect_bd_net -net rx_data_in_n_1 [get_bd_pins rx_data_in_n] [get_bd_pins axi_ad9361_0/rx_data_in_n]
connect_bd_net -net rx_data_in_p_1 [get_bd_pins rx_data_in_p] [get_bd_pins axi_ad9361_0/rx_data_in_p]
connect_bd_net -net rx_frame_in_n_1 [get_bd_pins rx_frame_in_n] [get_bd_pins axi_ad9361_0/rx_frame_in_n]
```

```
create_bd_addr_seg -range 0x00010000 -offset 0x44110000 [get_bd_addr_spaces axi_pcie3_1/M_AXI]
create_bd_addr_seg -range 0x00010000 -offset 0x44120000 [get_bd_addr_spaces axi_pcie3_1/M_AXI]
```

Instantiating blocks, setting parameters for those blocks, connecting them together, setting addresses.... There are text commands for all. Create this script right in IP integrator by saying File -> Export Block Design

Example Project Creation Script

```
#setup constraints, right now just default to 4ch fmc. we can setup the other later
create_fileset -constrset constrs_4chfmc
add_files -fileset constrs_4chfmc -norecurse $origin_dir/src/constrs/vc709_customfmc_4ch.xdc
create_fileset -constrset constrs_fmcomms
add_files -fileset constrs_fmcomms -norecurse $origin_dir/src/constrs/vc709_4chfmcomms.xdc
set_property constrset constrs_4chfmc [get_runs synth_1]
set_property constrset constrs_4chfmc [get_runs impl_1]

#add the HDL file
add_files -norecurse $origin_dir/src/hdl/toplevel_vc709.vhd

# generate the block design
cd src/bd
source bd_vc709_make.tcl

# Set 'sources_1' fileset file properties for remote files

update_compile_order -fileset sources_1
set_property strategy Performance_Explore [get_runs impl_1]
```

Same deal – File -> Write Project TCL will get you this file, which you can hand-edit, or export as needed from the GUI

Suggested Design Flow

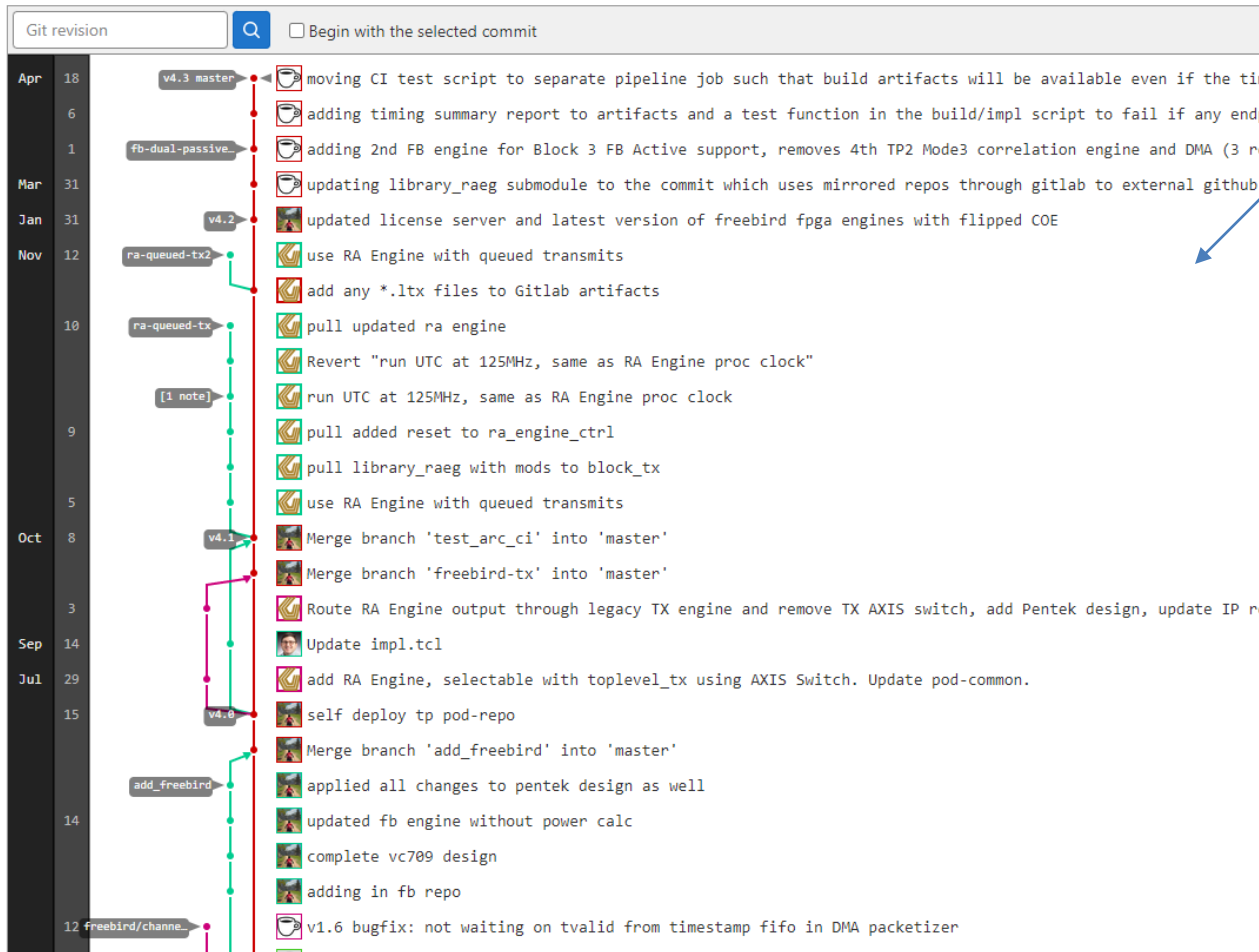
- Get yourself a working script to make a simple project with block design and sources included (the lab-6 design is a really good starting point)
 - Set up project
 - Add sources
 - Create the block design
- Commit these files to your revision control system (git, Subversion..)
- As the design evolves
 - Any changes which are obvious (adding a new file, changing a parameter or HDL...etc) can be done in text without even opening Vivado.
 - Edit your block design in the GUI as much as you like, and when done, File->Export Block Design and overwrite the existing TCL script
 - Commit changes frequently to a server, so you have a nice record of what changed, who changed it, and good backups. Any previous version can easily be re-created at a moment's notice by checking out that revision. When weird behavior is discovered, this may be necessary so that you can go back and test with old versions to find out when or how behavior was introduced.
- Once you have this set up, copying projects and creating a new project is even easier!
- Making designs be creatable with scripts lends itself to more automation as well....

Summary : this is a little more difficult to set up, and sometimes takes a little longer to make changes. It will 1000% pay for itself

Further Automation

- Hosted repositories and “Continuous Integration” frameworks (ex. Gitlab) can be used for even tighter collaboration in teams
- Since building the design is as easy as running a script, it isn’t hard to configure a server to build the design, make bitfiles, (and if you are really advanced, test those files)

Further Automation



History of changes
Click on any to see detail


```

4091 WARNING: [Synth 8-3331] design rd_dc_fwft_ext_as has unconnected port SRST
4092 WARNING: [Synth 8-3331] design rd_dc_fwft_ext_as has unconnected port EMPTY_FWFT
4093 WARNING: [Synth 8-3331] design rd_dc_fwft_ext_as has unconnected port ALMOST_EMPTY_FWFT
4094 WARNING: [Synth 8-3331] design rd_fwft has unconnected port SRST
4095 WARNING: [Synth 8-3331] design rd_fwft has unconnected port SAFETY_CKT_RD_RST
4096 WARNING: [Synth 8-3331] design rd_fwft has unconnected port RAM_ALMOST_EMPTY
4097 WARNING: [Synth 8-3331] design rd_status_flags_as has unconnected port SRST
4098 WARNING: [Synth 8-3331] design rd_status_flags_as has unconnected port SAFETY_CKT_RD_RST
4099 WARNING: [Synth 8-3331] design rd_status_flags_as has unconnected port RD_PNTR_PLUS2[3]
4100 WARNING: [Synth 8-3331] design rd_status_flags_as has unconnected port RD_PNTR_PLUS2[2]
4101 WARNING: [Synth 8-3331] design rd_status_flags_as has unconnected port RD_PNTR_PLUS2[1]
4102 WARNING: [Synth 8-3331] design rd_status_flags_as has unconnected port RD_PNTR_PLUS2[0]
4103 WARNING: [Synth 8-3331] design rd_bin_cntr has unconnected port SRST
4104 WARNING: [Synth 8-3331] design rd_logic has unconnected port RD_EN_INT0_LOGIC
4105 WARNING: [Synth 8-3331] design rd_logic has unconnected port RD_RST_INT0_LOGIC
4106 WARNING: [Synth 8-3331] design rd_logic has unconnected port RD_RST_BUSY
4107 WARNING: [Synth 8-3331] design rd_logic has unconnected port RAM_WR_EN
4108 WARNING: [Synth 8-3331] design rd_logic has unconnected port RST_FULL_FF
4109 WARNING: [Synth 8-3331] design rd_logic has unconnected port ALMOST_FULL_FB
4110 WARNING: [Synth 8-3331] design rd_logic has unconnected port FULL
4111 WARNING: [Synth 8-3331] design rd_logic has unconnected port WR_PNTR_PLUS1_RD[3]
4112 WARNING: [Synth 8-3331] design rd_logic has unconnected port WR_PNTR_PLUS1_RD[2]
4113 WARNING: [Synth 8-3331] design rd_logic has unconnected port WR_PNTR_PLUS1_RD[1]
4114 WARNING: [Synth 8-3331] design rd_logic has unconnected port WR_PNTR_PLUS1_RD[0]
4115 WARNING: [Synth 8-3331] design rd_logic has unconnected port PROG_EMPTY_THRESH[3]
4116 WARNING: [Synth 8-3331] design rd_logic has unconnected port PROG_EMPTY_THRESH[2]
4117 WARNING: [Synth 8-3331] design rd_logic has unconnected port PROG_EMPTY_THRESH[1]
4118 WARNING: [Synth 8-3331] design rd_logic has unconnected port PROG_EMPTY_THRESH[0]

```

Duration: 175 minutes 22 seconds

Finished: 4 months ago

Timeout: 12h (from project) 

Runner: #2176 (7GbpheKJ) arc-pod-fpga-0

Tags: fpga

Job artifacts



These artifacts are the latest. They will not be deleted (even if expired) until newer artifacts are available.

Download

Browse

Commit [e0bca361](#) 

adding 2nd FB engine for Block 3 FB
Active support, removes 4th TP2 Mode3 correlation engine and DMA (3 remaining)

 Pipeline #408911 for fb-dual-passive-active 

test

→  build

Can easily retrieve the log from the build, the *artifacts* (bitfile, ltx...etc), without even having to run vivado on your own machine – since it is built in the cloud

This all may seem like overkill, but as FPGA designs become large systems with multiple developers and many users, it is invaluable

High Level Synthesis

- Holy Grail for many designers is the idea that one could program in C, Model a system in Matlab, or Simulink, press a button and have it running in an FPGA
- Many tools proceeding towards this **goal**
 - Xilinx System Generator
 - Altera DSP Builder
 - Matlab HDL Coder
 - Vivado HLS
- FPGA knowledge is still needed in order to make effective use of these tools
- DSP Tools are very good when you are just hooking up preexisting blocks.
 - Provide a means to model in a more user friendly modelling environment like Matlab
 - Moving beyond this to more custom things can sometimes end up being lots of trouble.
- Some brief examples and resources will follow. You can play with these on your own (no board is required)

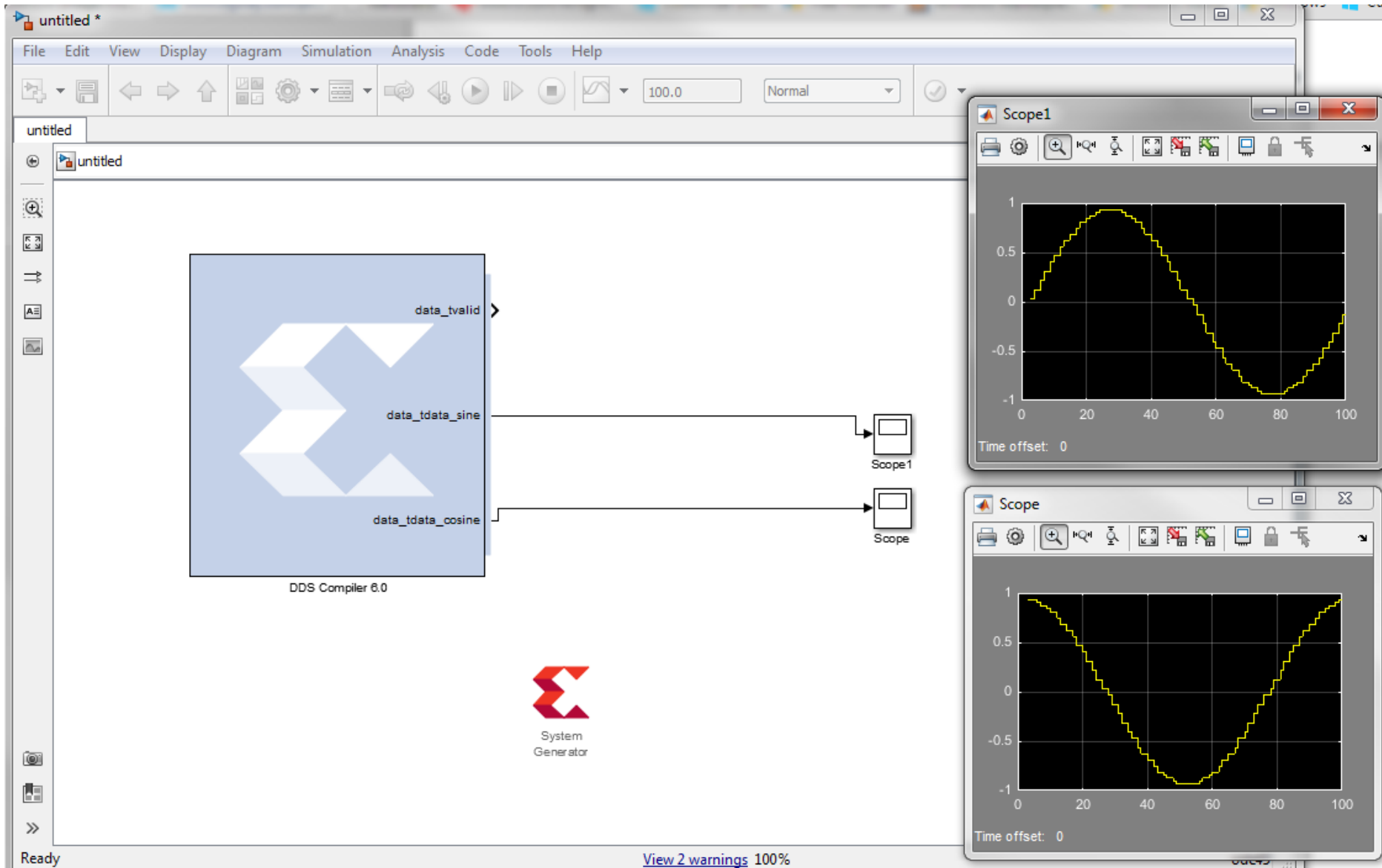
System Generator

- Manual with step-step tutorials
 - http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_1/ug948-vivado-sysgen-tutorial.pdf
- Simulink users will find it quite intuitive
- FPGA designers may find the abstraction hard to tolerate (where is my clock? What is the sample rate? Logic utilization?)

On your own experimentation

- Start System Generator from Windows Start bar
- Type “simulink” at the command prompt
- Select new model to create a new blank model
- Right click in white space -> Xilinx Block Add
 - All of the cores you are used to seeing here
 - Example with DDS compiler, System Generator Token, and scope on next page to illustrate

Example – Instantiated a DDS, changed the frequency to a fixed 1MHz



Vivado HLS

- C, C++, System C → FPGA automation
- http://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_2/ug871-vivado-high-level-synthesis-tutorial.pdf
- Why?
 - Coding in C can be quite efficient
 - Simulation is **way** faster
 - Benefit : a C model of what you are doing in the FPGA which is guaranteed to match the actual implementation (there isn't some human behind the scene who can confuse the logic when porting).
 - Possible Benefit : software acceleration, picking sections of C code that are a bottleneck in your process, and building accelerators for them without having to fully understand the code. {"Possible" is the key word}
 - End result is likely not as optimized as a Core from the IP library, but has the benefit of being flexible.
- An understanding of FPGA design is still required here, optimization is also quite dependent on specialized directives. Example to follow.

Making an HLS Peripheral

- Lets assume that we have some code that needs to multiply a number by the next 30 smaller numbers from it (a “quasi factorial”)

```
void quasi_factorial (  
    int *outdata,  
    int indata  
) {  
  
    int i;  
    int acc=indata;  
    MyLoop: for (i=1;i<30;i++) {  
        acc = acc * (indata-i);  
    }  
    *outdata=acc;  
}
```



I want to create a
peripheral to do this in
HW!

Simulation (Verifying C Code)

- Simulation in C can save a lot of time as well –
- Example “Testbench”

```
#include <stdio.h>
void quasi_factorial ( int *outdata, int indata);

int main()
{
    int result;

    printf("hello, welcome to the quasi factorial tester...\n");
    quasi_factorial(&result,5);
    printf("quasi factorial of 5 = %d\n",result);
    quasi_factorial(&result,10);
    printf("quasi factorial of 10 = %d\n",result);
    return 0;
}
```

Solutions -> Run C Synthesis

- HLS will synthesize a design to match this C code.
- Results show 4 DSP48s (necessary to do a 32 bit multiply) and lots of time. How could we change this?

Latency (clock cycles)

Summary

Latency		Interval		Type
min	max	min	max	
175	175	176	176	none

Detail

+ Instance

+ Loop

Utilization Estimates

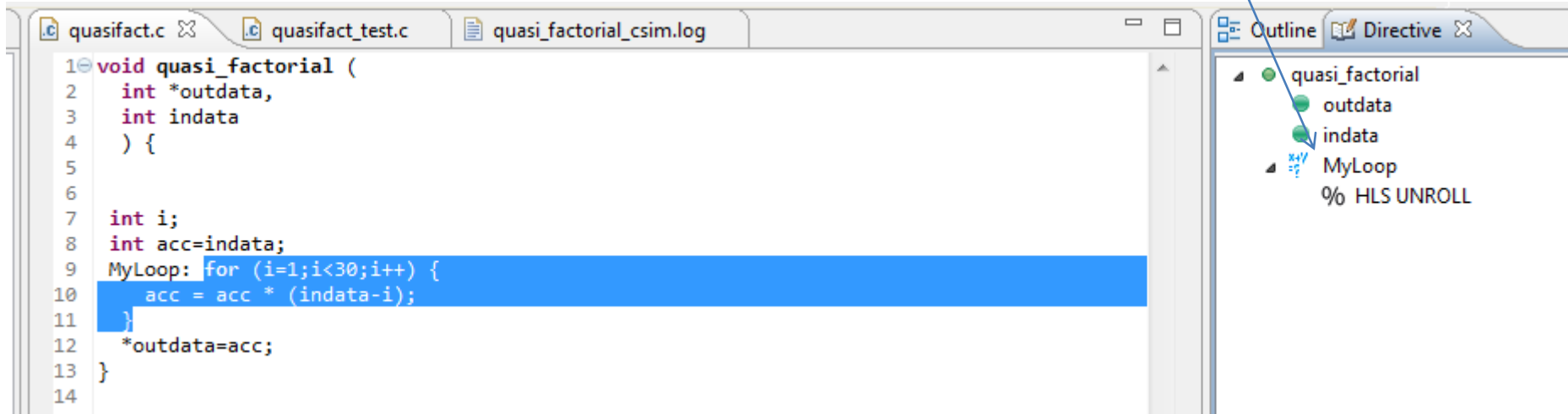
Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	39
FIFO	-	-	-	-
Instance	-	4	0	0
Memory	-	-	-	-
Multiplexer	-	-	-	39
Register	-	-	49	-
Total	0	4	49	78
Available	280	220	106400	53200
Utilization (%)	0	1	~0	~0

Optimizing

Millions of options exist here,
this is only meant to graze the
surface

Directive tab allows you to add
directives to particular sections of the
code. We've added an Unroll directive



Revised Utilization / Speed

Shows the use of 30 times as many multipliers, and an increase in speed.

Directives, (and changing your C code to be more inline with what it can optimize) are the way that this process will proceed.

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.52	1.25

Latency (clock cycles)

Summary

Latency		Interval		Type
min	max	min	max	
29	29	30	30	none

Detail

+ Instance

+ Loop

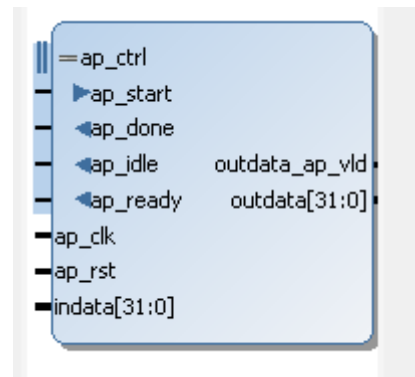
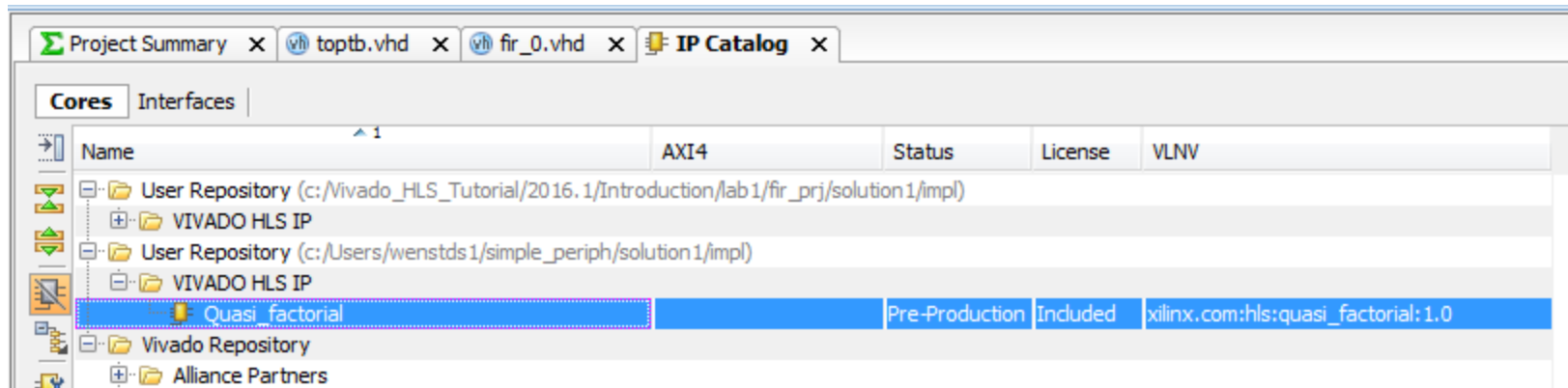
Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	928
FIFO	-	-	-	-
Instance	-	116	0	0
Memory	-	-	-	-
Multiplexer	-	-	-	24
Register	-	-	926	-
Total	0	116	926	952
Available	280	220	106400	53200
Utilization (%)	0	52	~0	1

Export to IP Catalog

- Solution -> Export RTL
 - Creates a ZIP file that is a piece of IP that you can use in a standard Vivado project



Ap_clk : (obviously) clk

Ap_rst : synchronous reset

indata : input data

Ap_ready : core is ready for new data

Ap_idle : core is doing nothing

Ap_done : core is finished

Outdata_ap_vld : core has new data on outdata

Outdata : the answer

All signals synchronous to ap_clk

Lab 10

- The first part of the aforementioned tutorial, with a couple of slight modifications and answering some simple questions.
- An 11-tap FIR filter coded in C. Tutorial leads you through the mapping of that function into hardware, and simulation.
- There are endless different ways that a design could be implemented in hardware, so the tutorial will also investigate some of the options that can be selected to tradeoff size, throughput, latency...etc.

This project is only for you – it is NOT a graded lab

Board Return

- Tracy Gauthier from the EP administration office will be contacting you via email to coordinate the return of the kits.
 - Please include everything : (Zybo, Power Supply, USB Cables, PMOD, Sound Card, Audio Cable, SD Card) and anything I forgot here
 - Local students will likely be asked to drop off
 - Remote students will get a pre-paid label
- Please contact me via Teams if you have any questions
- Don't forget to fill out the course evaluations, we really do read them!