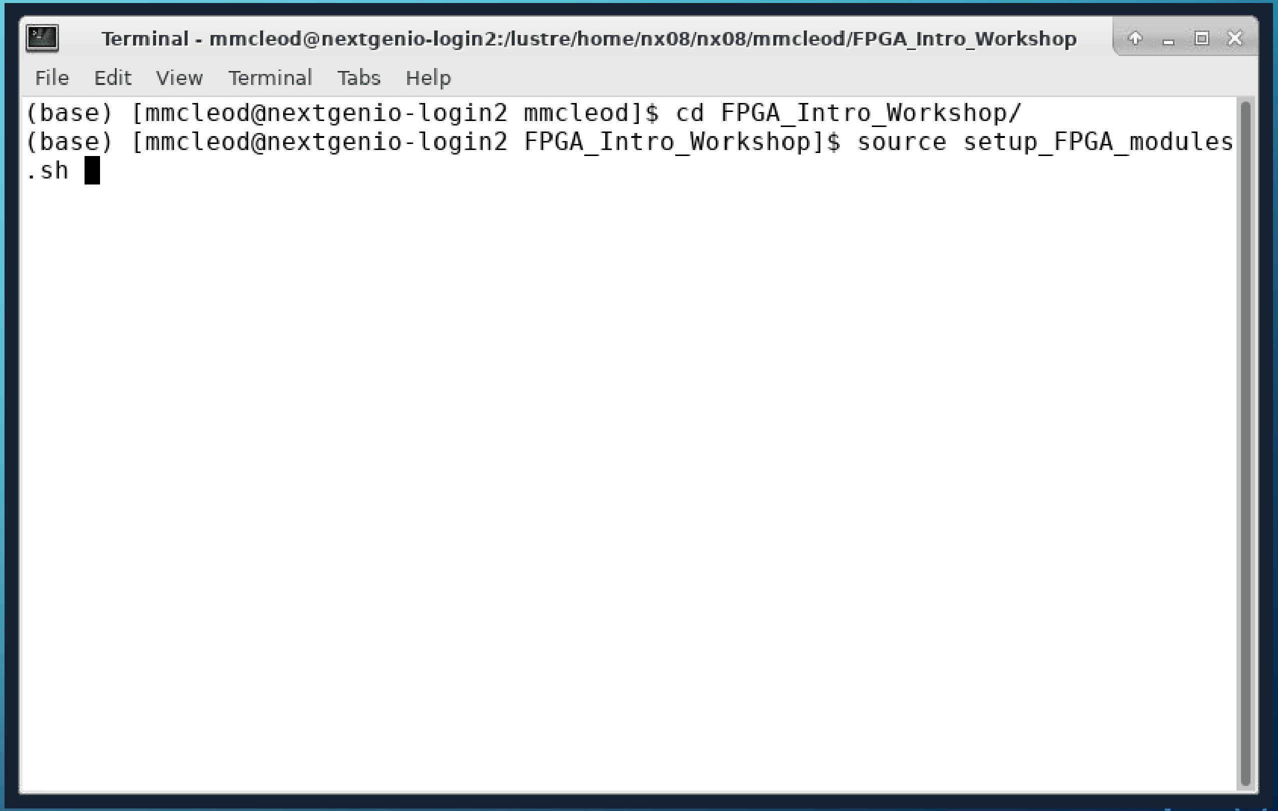
VITIS HLS QUICKSTART

GETTING STARTED

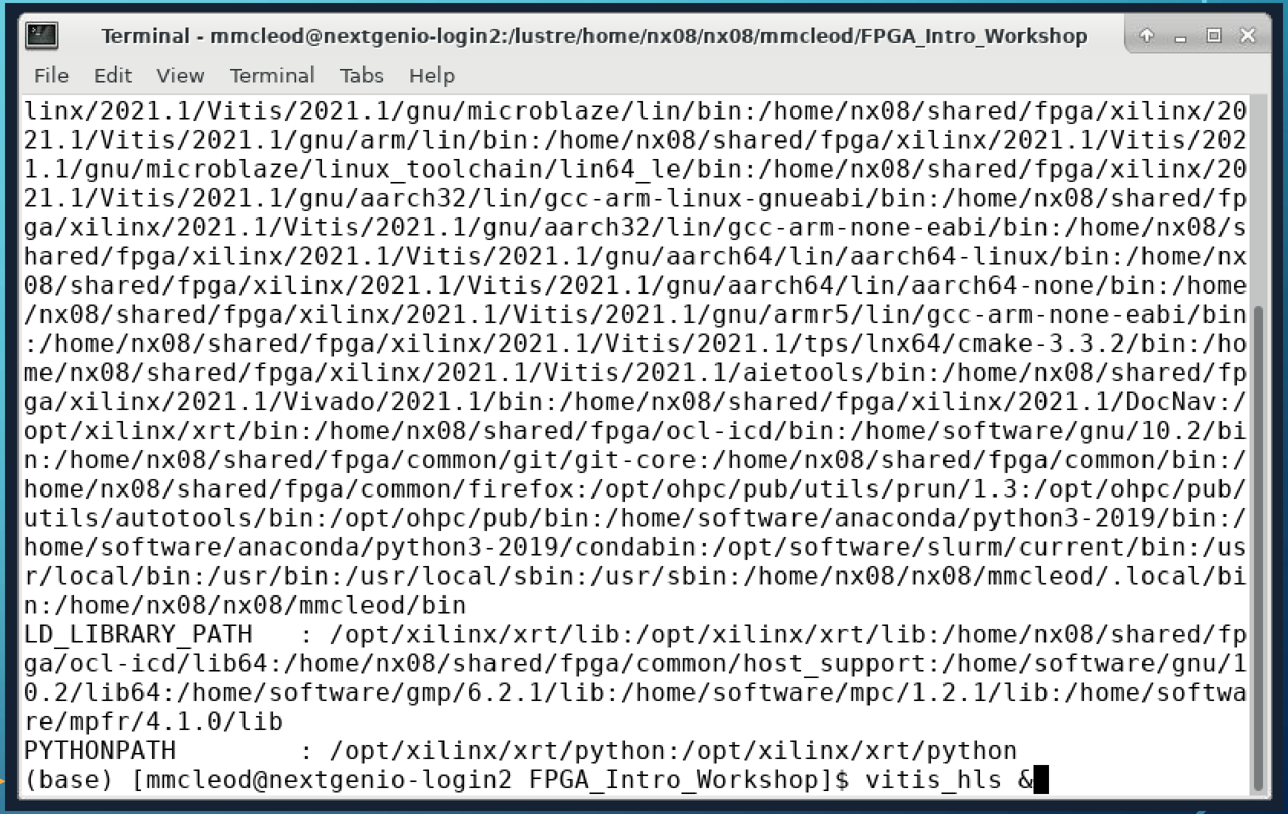
- Open your terminal and go to your FPGA_Intro_Workshop folder
- Source the setup_FPGA_modules.sh script
- This loads the FPGA related modules we need to access the software / do compilation etc.


Terminal - mmcleod@nextgenio-login2:/lustre/home/nx08/nx08/mmcleod/FPGA_Intro_Workshop
File Edit View Terminal Tabs Help

```
(base) [mmcleod@nextgenio-login2 mmcleod]$ cd FPGA_Intro_Workshop/
(base) [mmcleod@nextgenio-login2 FPGA_Intro_Workshop]$ source setup_FPGA_modules
.sh
```

GETTING STARTED

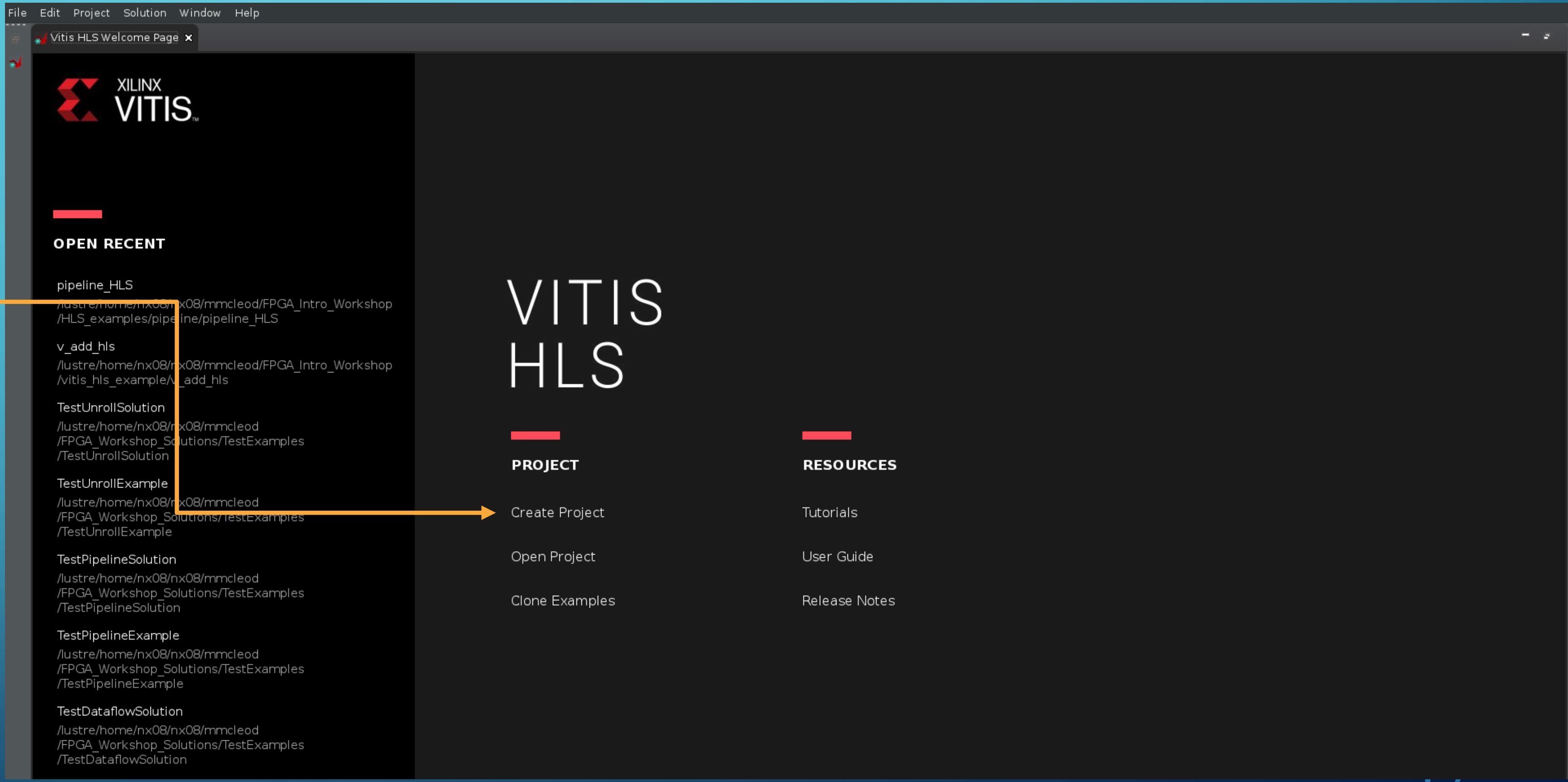
- Once it's done, just type "vitis_hls &" into your terminal


Terminal - mmcleod@nextgenio-login2:/lustre/home/nx08/nx08/mmcleod/FPGA_Intro_Workshop
File Edit View Terminal Tabs Help

```
linx/2021.1/Vitis/2021.1/gnu/microblaze/lin/bin:/home/nx08/shared/fpga/xilinx/20
21.1/Vitis/2021.1/gnu/arm/lin/bin:/home/nx08/shared/fpga/xilinx/2021.1/Vitis/202
1.1/gnu/microblaze/linux_toolchain/lin64_le/bin:/home/nx08/shared/fpga/xilinx/20
21.1/Vitis/2021.1/gnu/aarch32/lin/gcc-arm-linux-gnueabi/bin:/home/nx08/shared/fp
ga/xilinx/2021.1/Vitis/2021.1/gnu/aarch32/lin/gcc-arm-none-eabi/bin:/home/nx08/s
hared/fpga/xilinx/2021.1/Vitis/2021.1/gnu/aarch64/lin/aarch64-linux/bin:/home/nx
08/shared/fpga/xilinx/2021.1/Vitis/2021.1/gnu/aarch64/lin/aarch64-none/bin:/home
/nx08/shared/fpga/xilinx/2021.1/Vitis/2021.1/gnu/armr5/lin/gcc-arm-none-eabi/bin
:/home/nx08/shared/fpga/xilinx/2021.1/Vitis/2021.1/tps/lnx64/cmake-3.3.2/bin:/ho
me/nx08/shared/fpga/xilinx/2021.1/Vitis/2021.1/aietools/bin:/home/nx08/shared/fp
ga/xilinx/2021.1/Vivado/2021.1/bin:/home/nx08/shared/fpga/xilinx/2021.1/DocNav:/
opt/xilinx/xrt/bin:/home/nx08/shared/fpga/ocl-icd/bin:/home/software/gnu/10.2/bi
n:/home/nx08/shared/fpga/common/git/git-core:/home/nx08/shared/fpga/common/bin:/
home/nx08/shared/fpga/common/firefox:/opt/ohpc/pub/utils/prun/1.3:/opt/ohpc/pub/
utils/autotools/bin:/opt/ohpc/pub/bin:/home/software/anaconda/python3-2019/bin:/
home/software/anaconda/python3-2019/condabin:/opt/software/slurm/current/bin:/us
r/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/nx08/nx08/mmcleod/.local/bi
n:/home/nx08/nx08/mmcleod/bin
LD_LIBRARY_PATH    : /opt/xilinx/xrt/lib:/opt/xilinx/xrt/lib:/home/nx08/shared/fp
ga/ocl-icd/lib64:/home/nx08/shared/fpga/common/host_support:/home/software/gnu/1
0.2/lib64:/home/software/gmp/6.2.1/lib:/home/software/mpc/1.2.1/lib:/home/softwa
re/mpfr/4.1.0/lib
PYTHONPATH         : /opt/xilinx/xrt/python:/opt/xilinx/xrt/python
(base) [mmcleod@nextgenio-login2 FPGA_Intro_Workshop]$ vitis_hls &
```

GETTING STARTED

- A new window will open that looks like this
- Click "Create Project"



OPEN RECENT

pipeline_HLS
/lustre/home/nx08/nx08/mmcleod/FPGA_Intro_Workshop
/HLS_examples/pipeline/pipeline_HLS

v_add_hls
/lustre/home/nx08/nx08/mmcleod/FPGA_Intro_Workshop
/vitis_hls_example/v_add_hls

TestUnrollSolution
/lustre/home/nx08/nx08/mmcleod
/FPGA_Workshop_Solutions/TestExamples
/TestUnrollSolution

TestUnrollExample
/lustre/home/nx08/nx08/mmcleod
/FPGA_Workshop_Solutions/TestExamples
/TestUnrollExample

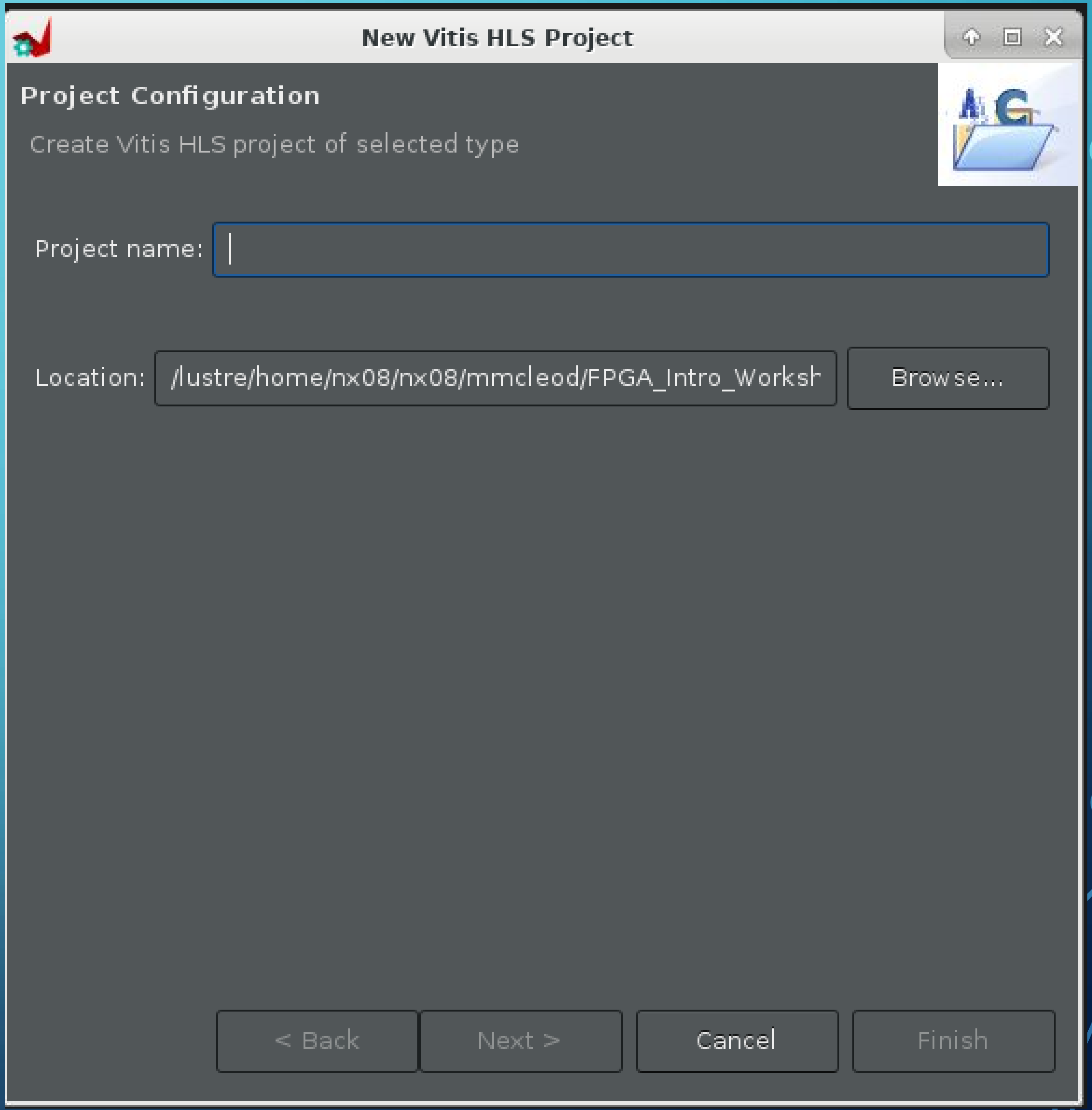
TestPipelineSolution
/lustre/home/nx08/nx08/mmcleod
/FPGA_Workshop_Solutions/TestExamples
/TestPipelineSolution

TestPipelineExample
/lustre/home/nx08/nx08/mmcleod
/FPGA_Workshop_Solutions/TestExamples
/TestPipelineExample

TestDataflowSolution
/lustre/home/nx08/nx08/mmcleod
/FPGA_Workshop_Solutions/TestExamples
/TestDataflowSolution

PROJECT CREATION

- Add a project name. We'll be working on a v_add (vector addition) kernel, so something like v_add_hls
- Select a location for your project to be created: for your first example, choose the vitis_hls_example folder!



Project Configuration

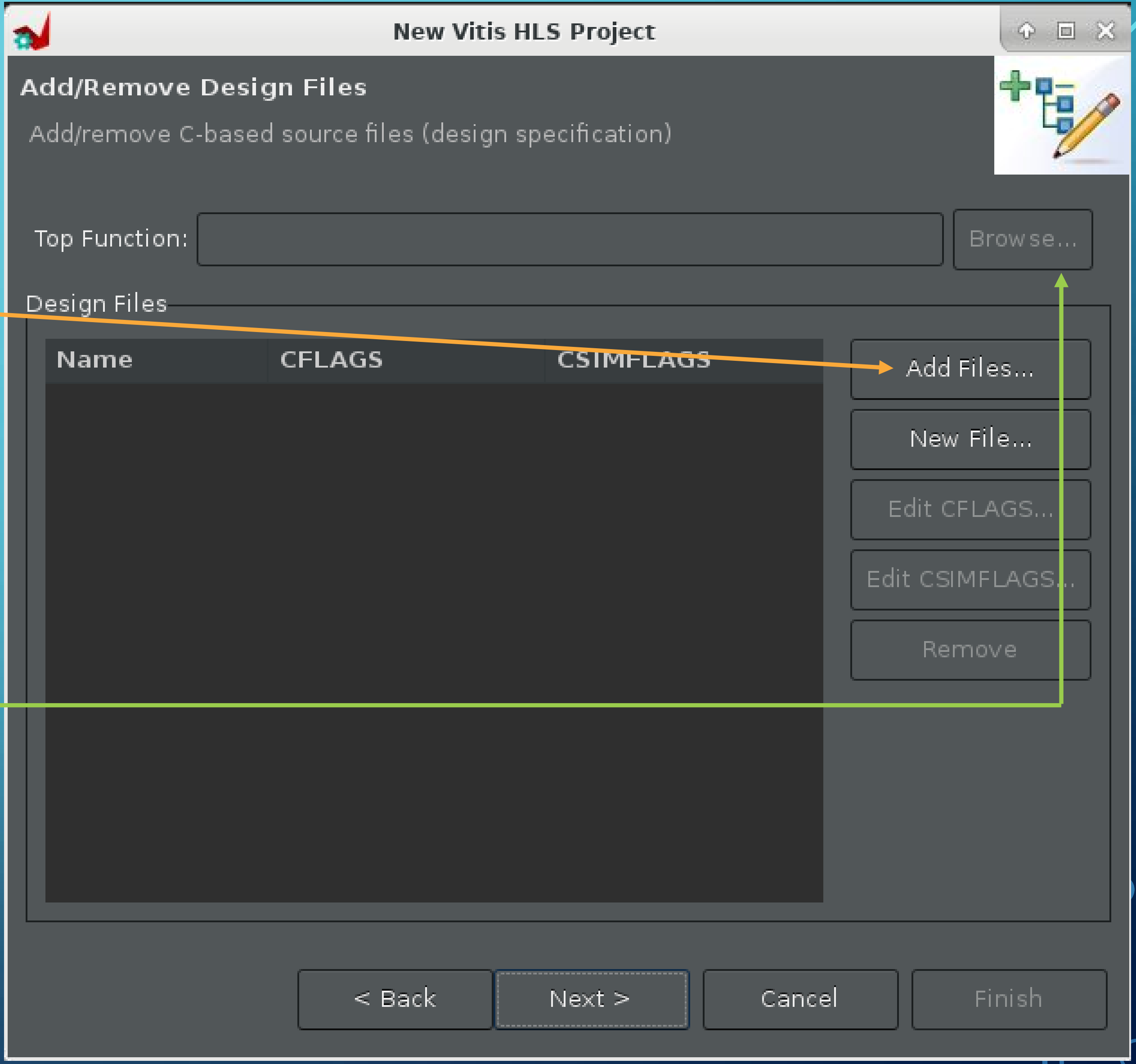
Create Vitis HLS project of selected type

Project name:

Location: /lustre/home/nx08/nx08/mmcleod/FPGA_Intro_Worksh

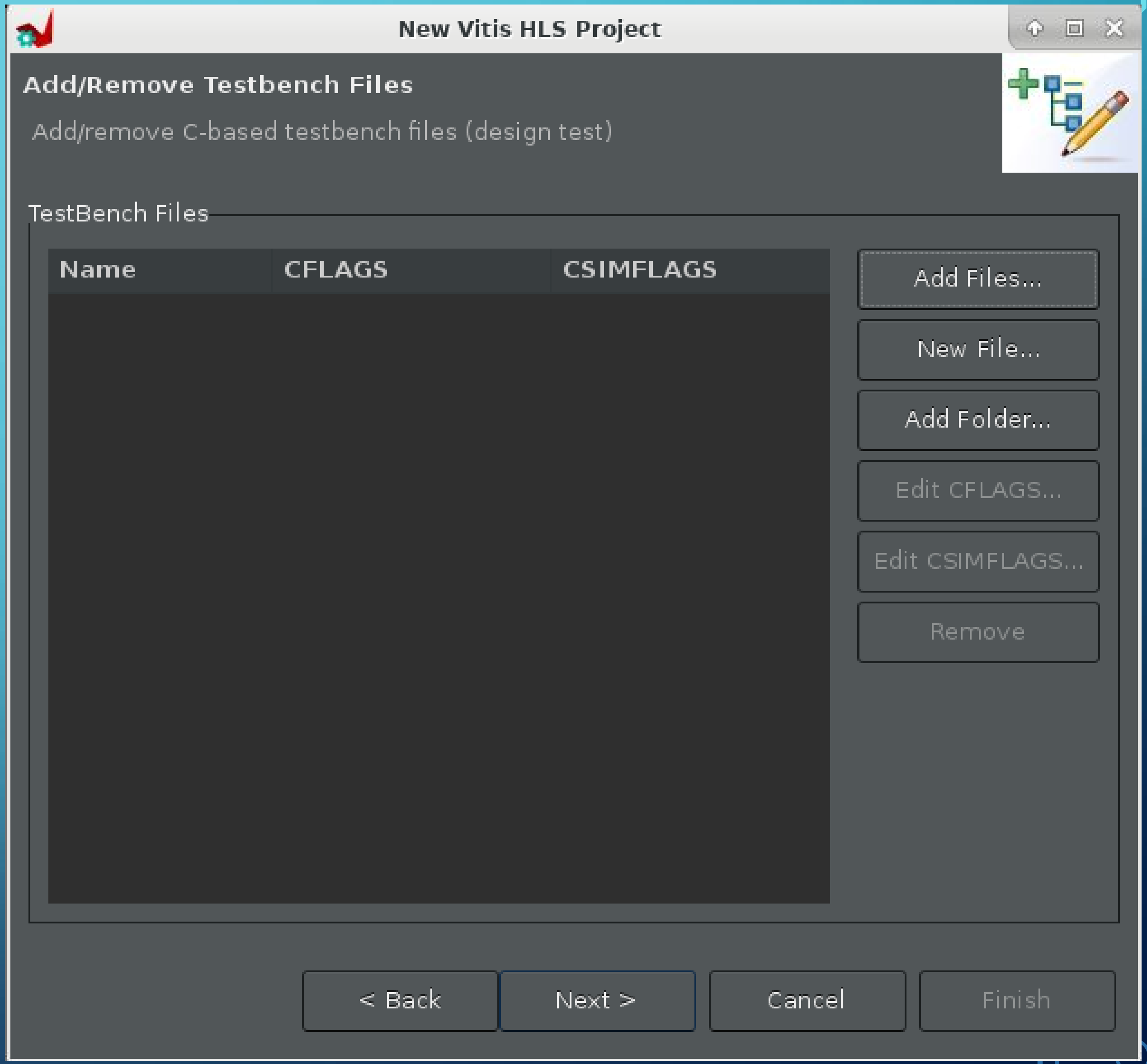
ADDING DESIGN FILES

- First click on Add Files…
- Go into the v_add subfolder of vitis_hls_example and add v_add_kernel.cpp
- Then go to the Top Function section and click Browse…
- Select v_add_kernel (our entry point function for the kernel)



ADDING TEST BENCH

- The test bench is a simple program with a main() which calls the kernel like a normal C function to test its functionality
- Returns 0 for success, other for fail
- Click Add Files… and add v_add_test.cpp


New Vitis HLS Project

Add/Remove Testbench Files

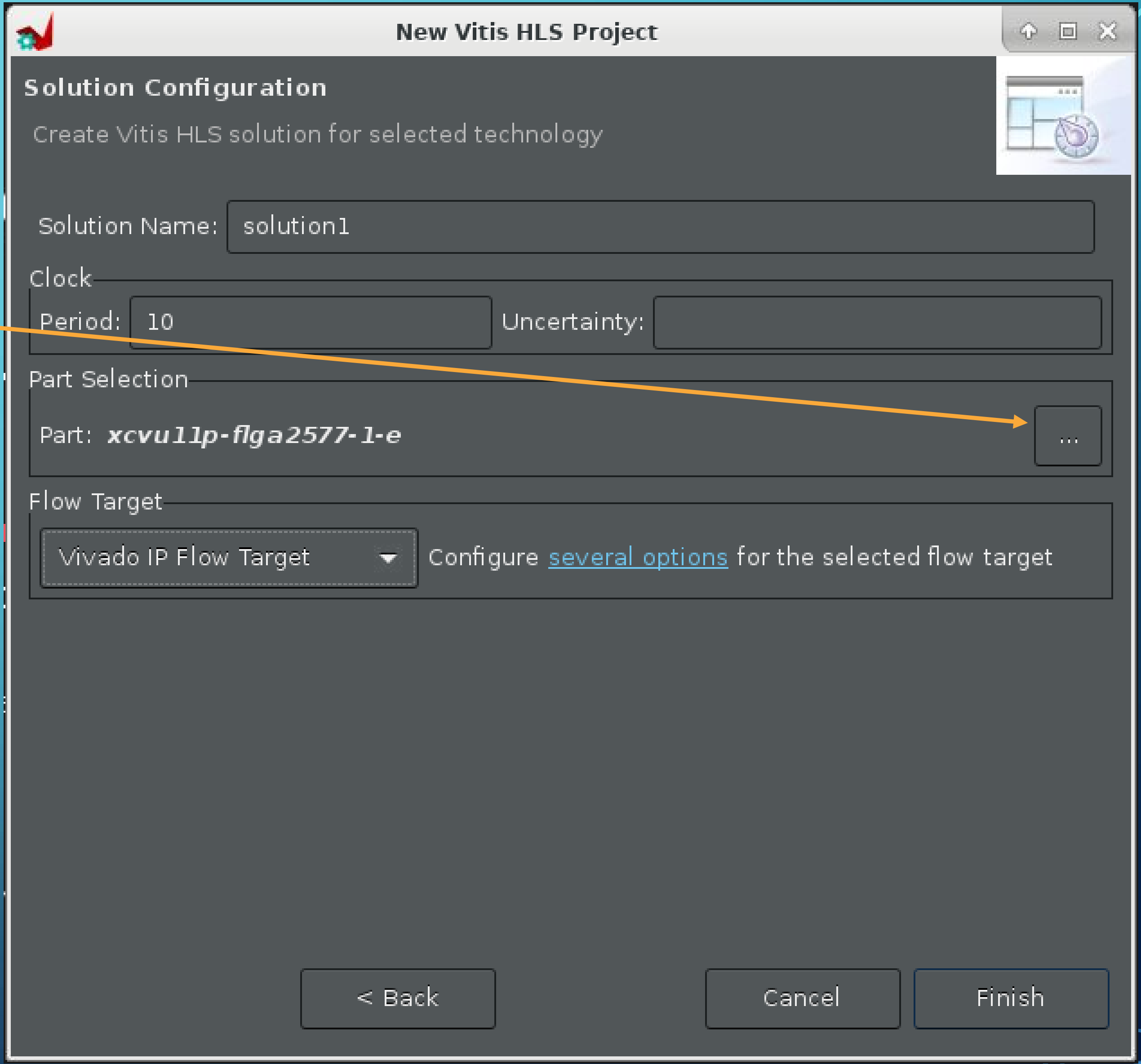
Add/remove C-based testbench files (design test)

TestBench Files

Name	CFLAGS	CSIMFLAGS

SETTING TARGETS

- First got to Part Selection


New Vitis HLS Project

Solution Configuration

Create Vitis HLS solution for selected technology

Solution Name: solution1

Clock

Period: 10

Uncertainty:

Part Selection

Part: *xcvu11p-flga2577-1-e*

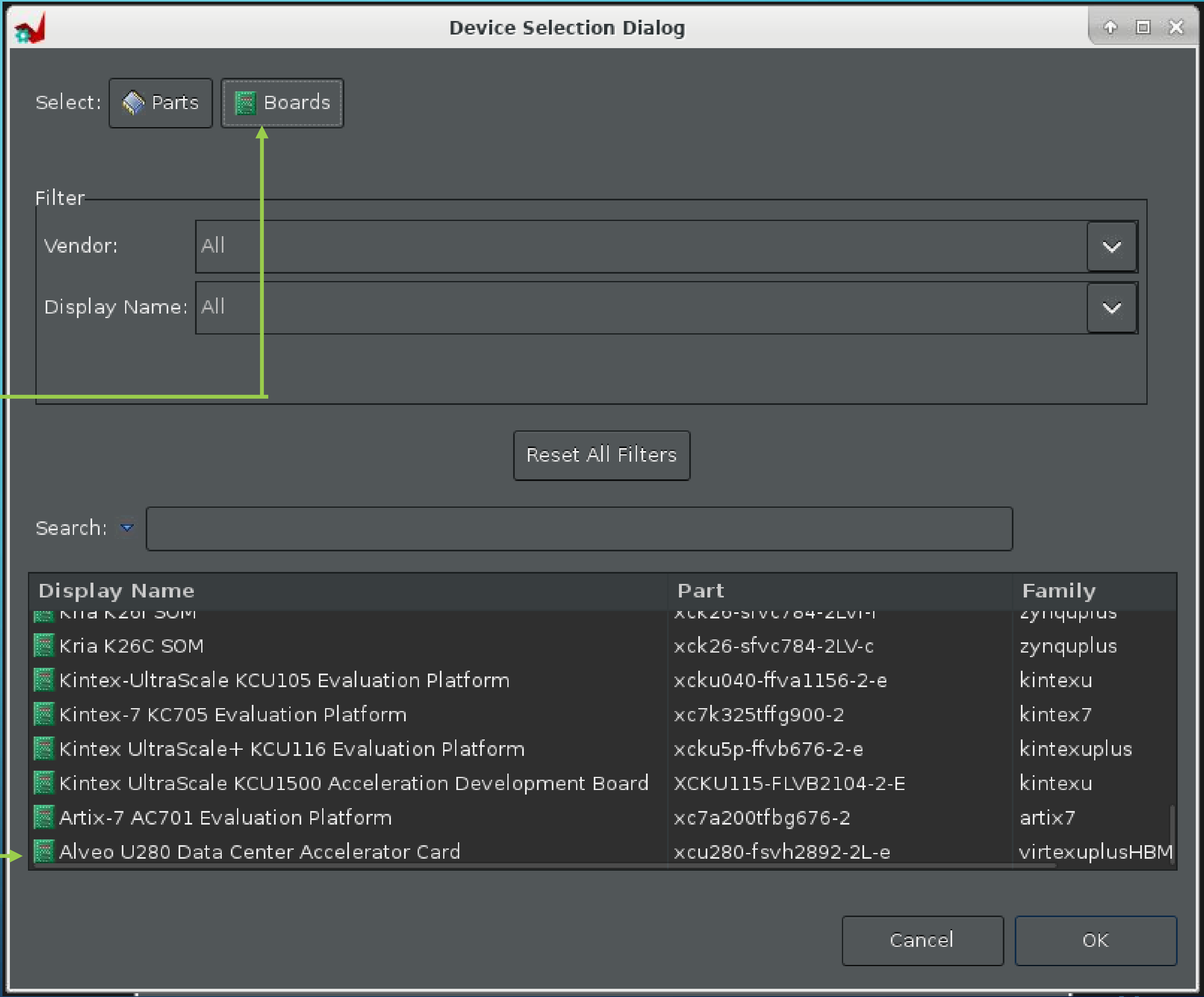
Flow Target

Vivado IP Flow Target

Configure several options for the selected flow target

SETTING TARGETS

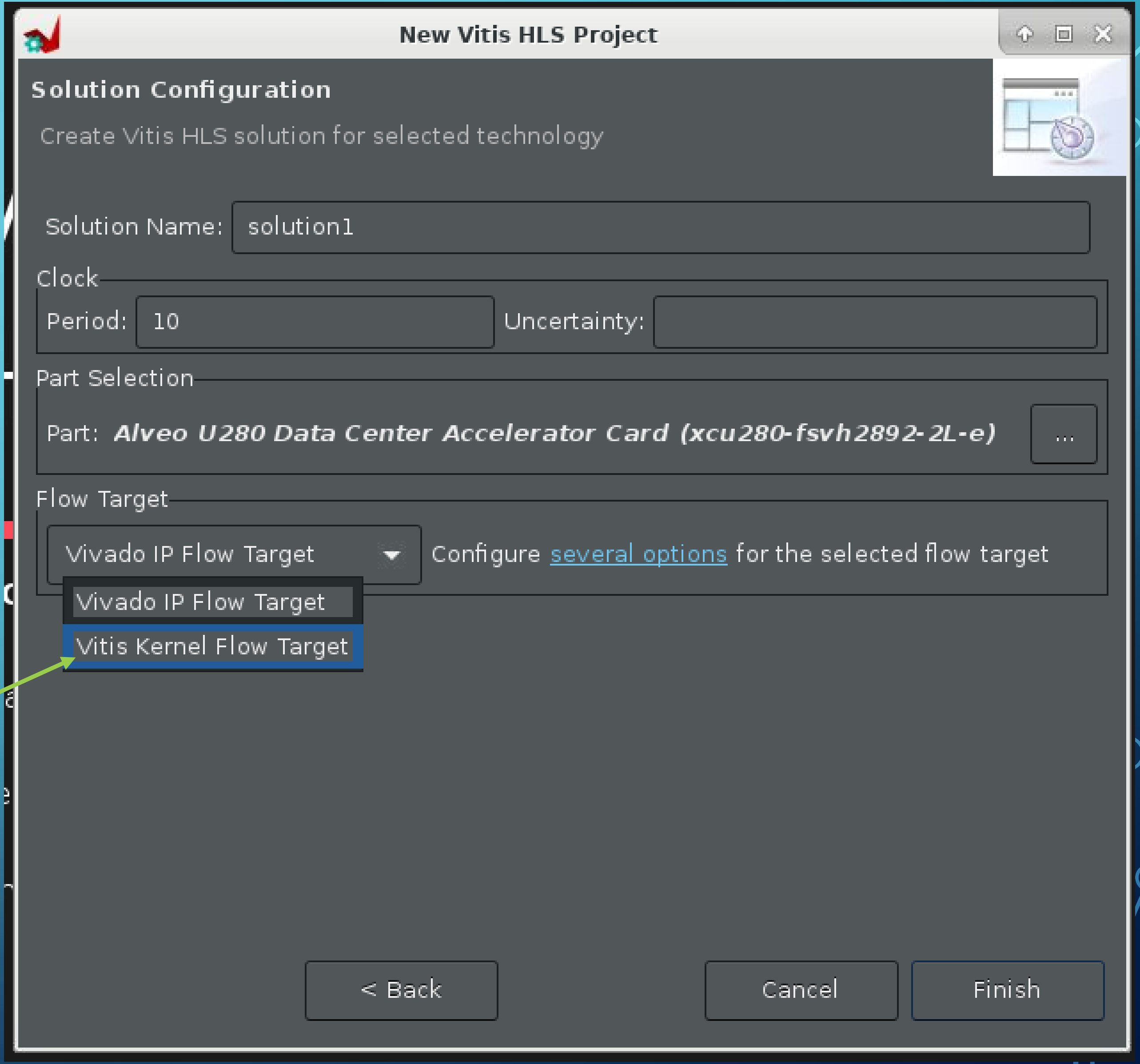
- First got to Part Selection
- Click on Boards
- Then scroll down to find the Alveo U280



Display Name	Part	Family
Kria K26I SOM	xck26-sfvc784-2LVI-i	zynquplus
Kria K26C SOM	xck26-sfvc784-2LV-c	zynquplus
Kintex-UltraScale KCU105 Evaluation Platform	xcku040-ffva1156-2-e	kintexu
Kintex-7 KC705 Evaluation Platform	xc7k325tffg900-2	kintex7
Kintex UltraScale+ KCU116 Evaluation Platform	xcku5p-ffvb676-2-e	kintexuplus
Kintex UltraScale KCU1500 Acceleration Development Board	XCKU115-FLVB2104-2-E	kintexu
Artix-7 AC701 Evaluation Platform	xc7a200tfbg676-2	artix7
Alveo U280 Data Center Accelerator Card	xcu280-fsvh2892-2L-e	virtexuplusHBM

SETTING TARGETS

- First got to Part Selection
- Click on Boards
- Then scroll down to find the Alveo U280
- Then go to Flow Target and select Vitis Kernel Flow Target
- Then click Finish!



New Vitis HLS Project

Solution Configuration

Create Vitis HLS solution for selected technology

Solution Name: solution1

Clock

Period: 10 Uncertainty:

Part Selection

Part: *Alveo U280 Data Center Accelerator Card (xcu280-fsvh2892-2L-e)*

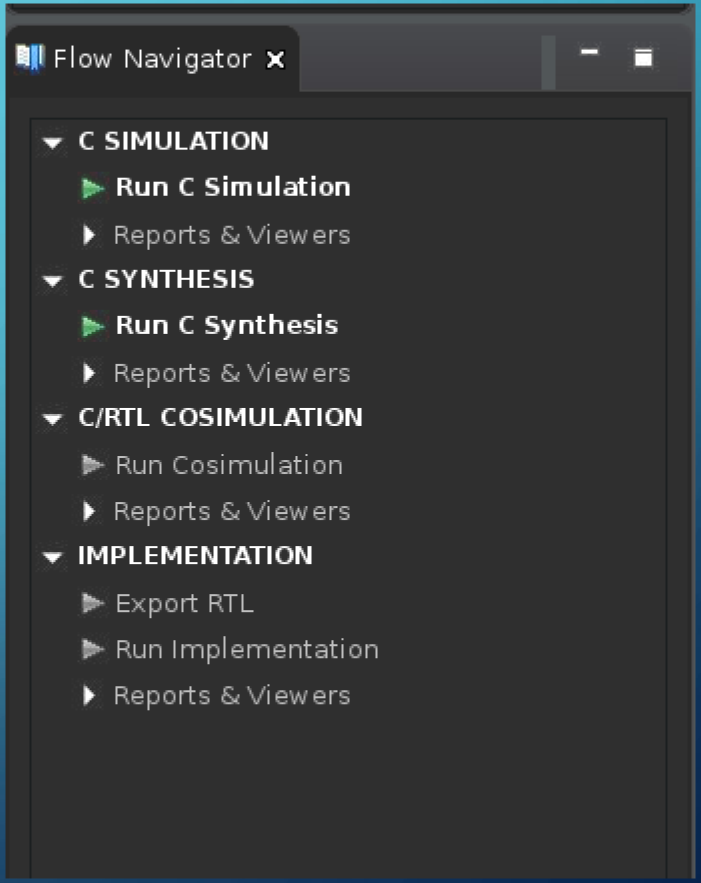
Flow Target

Vivado IP Flow Target

Configure several options for the selected flow target

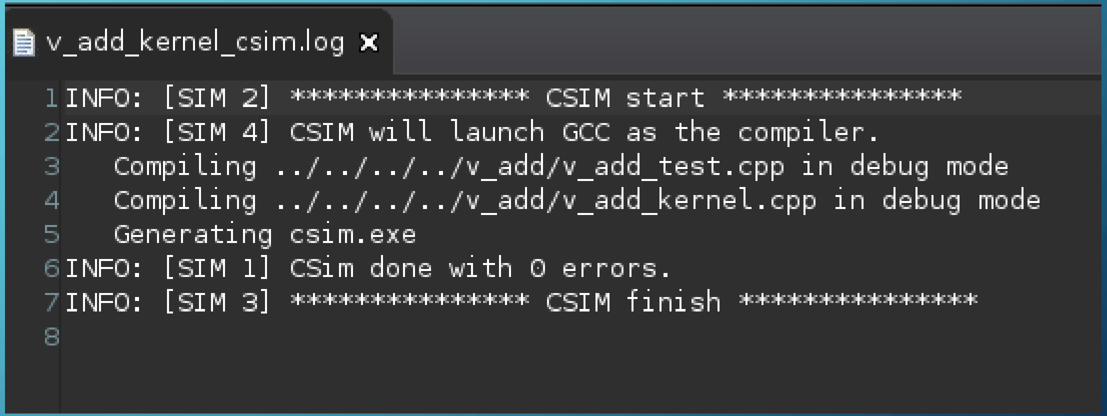
FLOW NAVIGATOR

- In the bottom left you'll see the Flow Navigator
- Lets you do:
  - C Simulation
  - C Synthesis
  - C/RTL Cosimulation
  - (We won't look at Implementation)
- Run each of these in turn by clicking on the arrows
- See HLS notes for more on what these each do



C SIMULATION

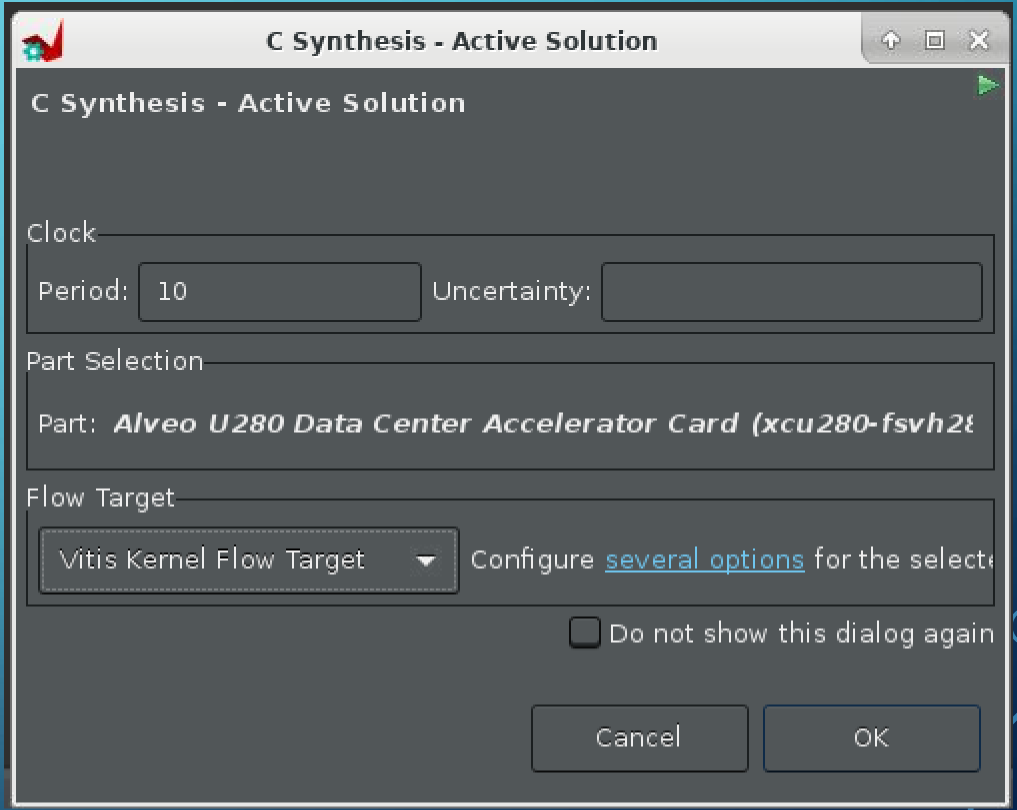
- For now you can just click ahead with the defaults (everything unchecked)
- You'll see a summary declaring your testbench passed with no errors
- C simulation runs your C code and checks that the return is 0 and that the program doesn't crash!
- Basic level of testing



```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../v_add/v_add_test.cpp in debug mode
   Compiling ../../../../v_add/v_add_kernel.cpp in debug mode
   Generating csim.exe
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

C SYNTHESIS

- It should already say Alveo U280 and Vitis Kernel Flow from your project set up, but if it does not then select it here
- Details of the synthesis will appear in the console, but don't worry too much about that until you're familiar with the kinds of things that HLS does


C Synthesis - Active Solution

C Synthesis - Active Solution

Clock

Period: 10 Uncertainty:

Part Selection

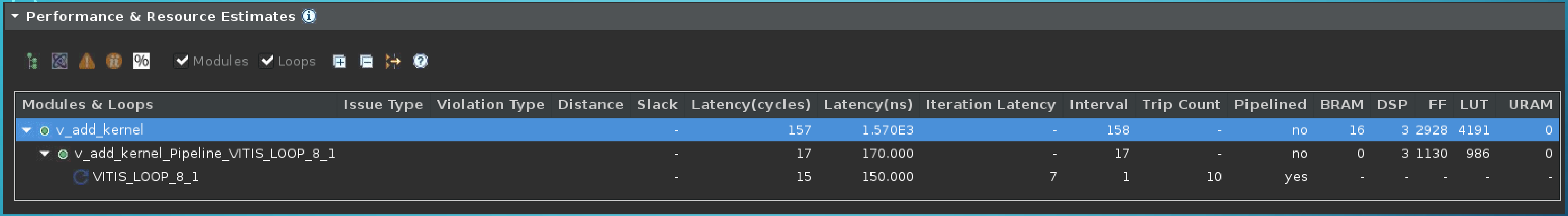
Part: *Alveo U280 Data Center Accelerator Card (xcu280-fsvh2ε*

Flow Target

Vitis Kernel Flow Target Configure several options for the selecte

☐ Do not show this dialog again

C SYNTHESIS



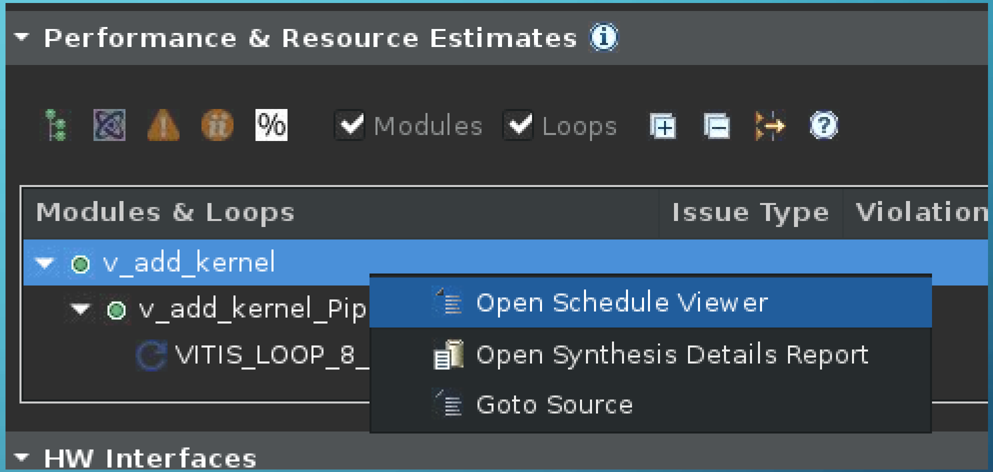
Performance & Resource Estimates

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
v_add_kernel				-	157	1.570E3	-	158	-	no	16	3	2928	4191	0
v_add_kernel_Pipeline_VITIS_LOOP_8_1				-	17	170.000	-	17	-	no	0	3	1130	986	0
VITIS_LOOP_8_1				-	15	150.000	7	1	10	yes	-	-	-	-	-

- Performance & Resource Estimates is the most important part for our purposes!
- Latency (cycles) gives number of clock cycles to execute your kernel
- For the loop at the bottom, note the Iteration Latency (time for a single loop iteration) and the Interval (time between one iteration starting and the next starting).
- Interval of 1 means we start a new iteration every clock cycle, so our loop iterations are overlapping in parallel!

C SYNTHESIS

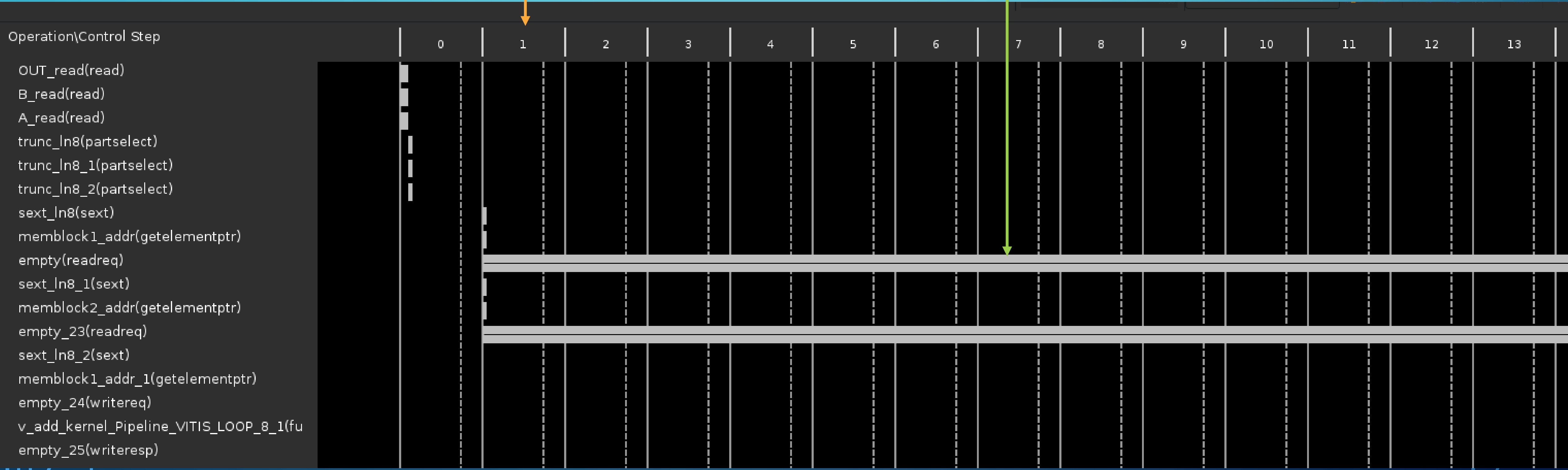
- Open the schedule viewer by right clicking on your top level function, or by selecting it in the "Solution" menu at the top



C SYNTHESIS

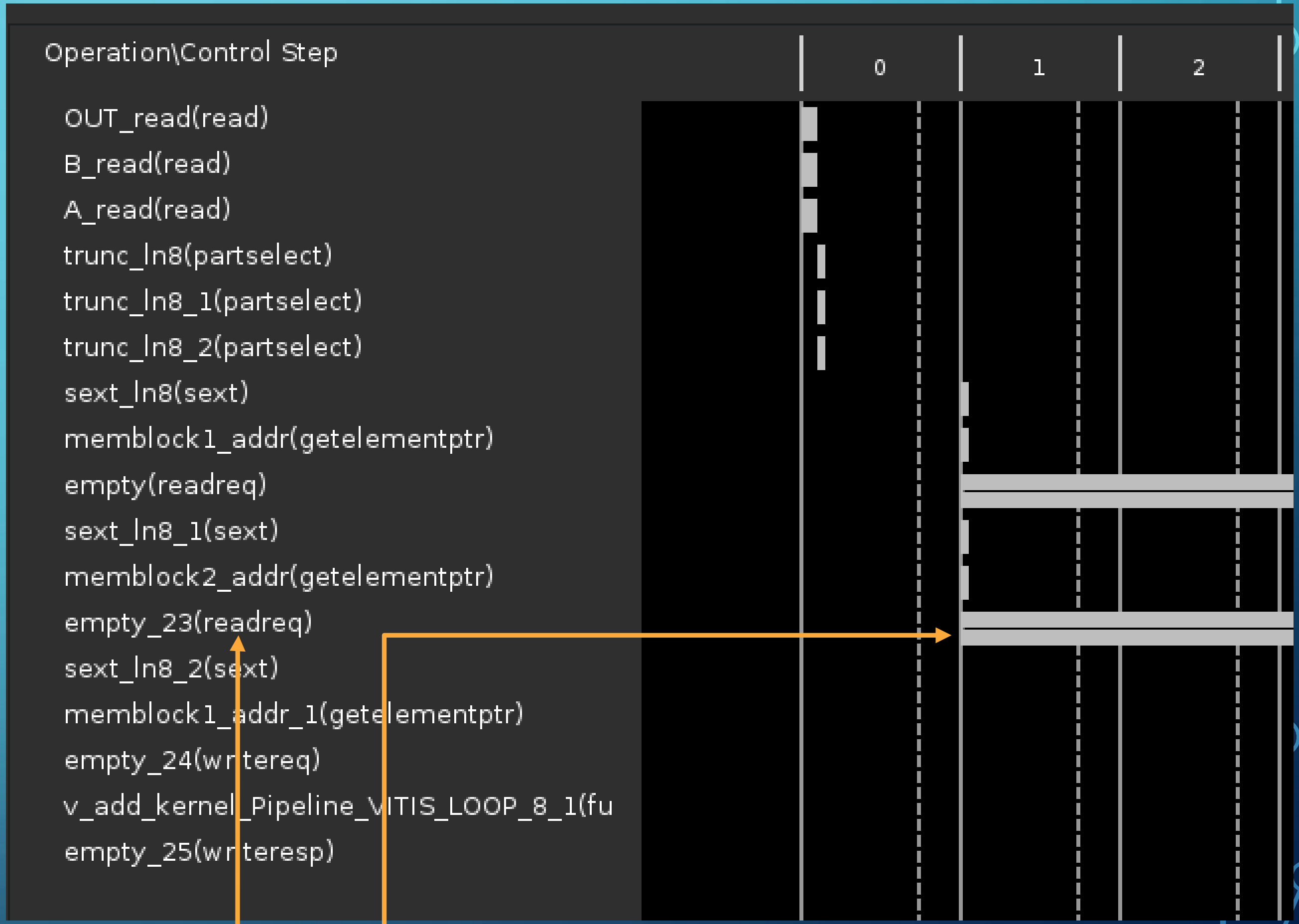
Time in clock cycles

Length of operations



C SYNTHESIS

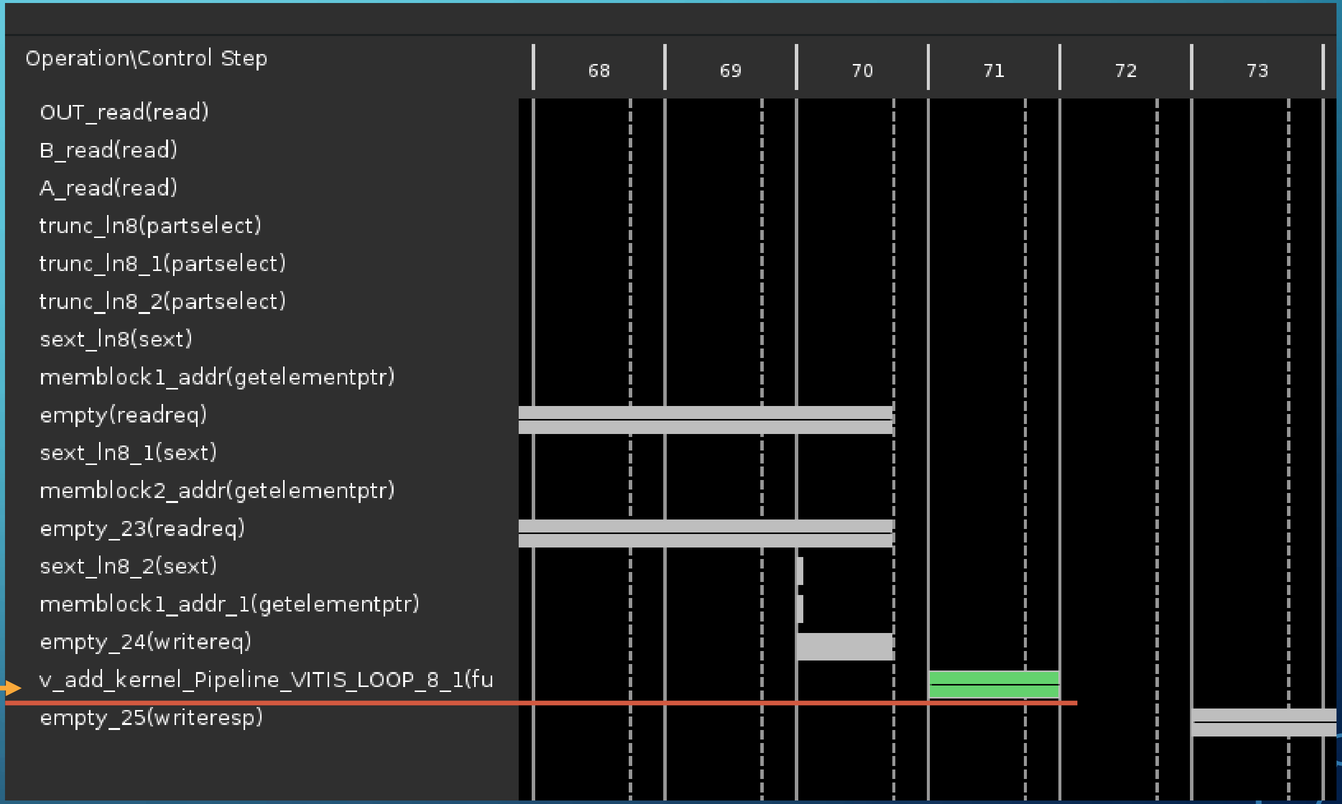
- Don't worry about all these operations: many are low level hardware things that are out of our control at such a high level approach
- Focus on operations that take a long time or are recognisable
- These big operations are read requests: be careful with read/write to global memory!



```
Operation\Control Step                         0    1    2
OUT_read(read)
B_read(read)
A_read(read)
trunc_ln8(partselect)
trunc_ln8_1(partselect)
trunc_ln8_2(partselect)
sext_ln8(sext)
memblock1_addr(getelementptr)
empty(readreq)
sext_ln8_1(sext)
memblock2_addr(getelementptr)
empty_23(readreq)
sext_ln8_2(sext)
memblock1_addr_1(getelementptr)
empty_24(writereq)
v_add_kernel_Pipeline_VITIS_LOOP_8_1(fu
empty_25(writeresp)
```

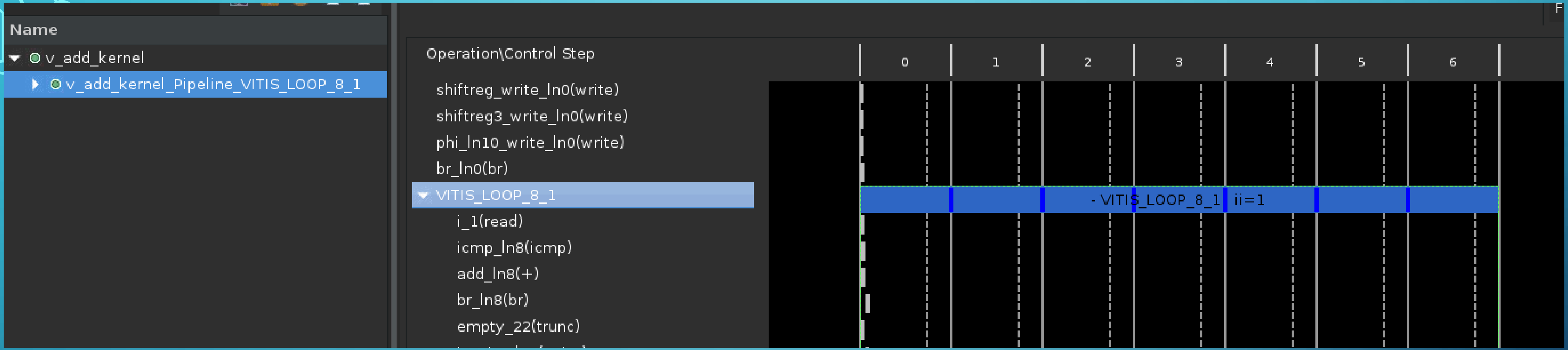
C SYNTHESIS

- Pipelined loops are collapsed in top level view
- Click on the loop in the left hand box, “Module Hierarchy” to bring up the schedule for what happens inside the loop



```
Operation\Control Step      68   69   70   71   72   73
OUT_read(read)
B_read(read)
A_read(read)
trunc_ln8(partselect)
trunc_ln8_1(partselect)
trunc_ln8_2(partselect)
sext_ln8(sext)
memblock1_addr(getelementptr)
empty(readreq)
sext_ln8_1(sext)
memblock2_addr(getelementptr)
empty_23(readreq)
sext_ln8_2(sext)
memblock1_addr_1(getelementptr)
empty_24(writereq)
v_add_kernel_Pipeline_VITIS_LOOP_8_1(fu
empty_25(writeresp)
```

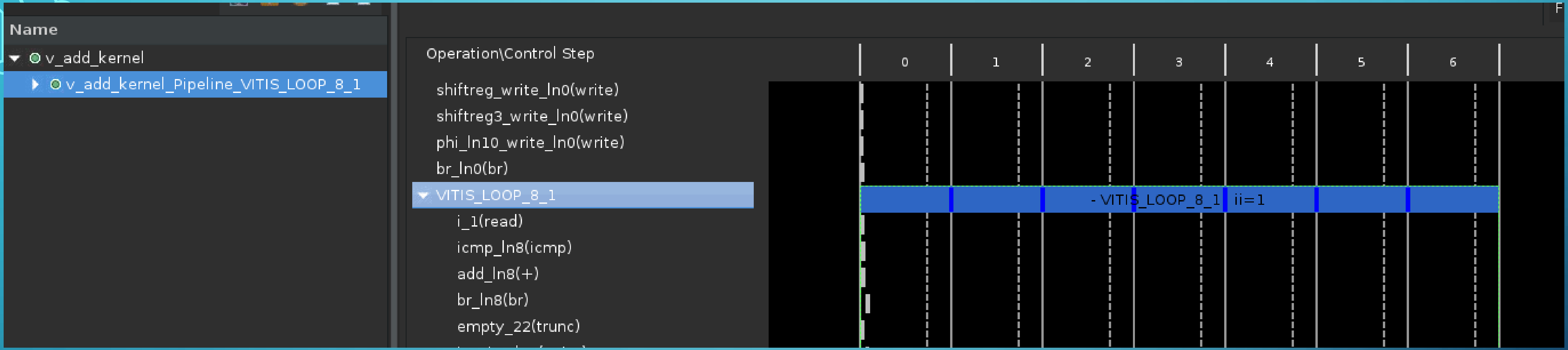
C SYNTHESIS



shiftreg_write_ln0(write)
shiftreg3_write_ln0(write)
phi_ln10_write_ln0(write)
br_ln0(br)
VITIS_LOOP_8_1
i_1(read)
icmp_ln8(icmp)
add_ln8(+)
br_ln8(br)
empty_22(trunc)

- Expand the loop by clicking on VITIS_LOOP_8_1
- The blue bar shows the length of the loop
- The darker divisions in the bar show the start of each new iteration of a loop
- Because the Initiation Interval is 1, there is a new iteration every clock cycle

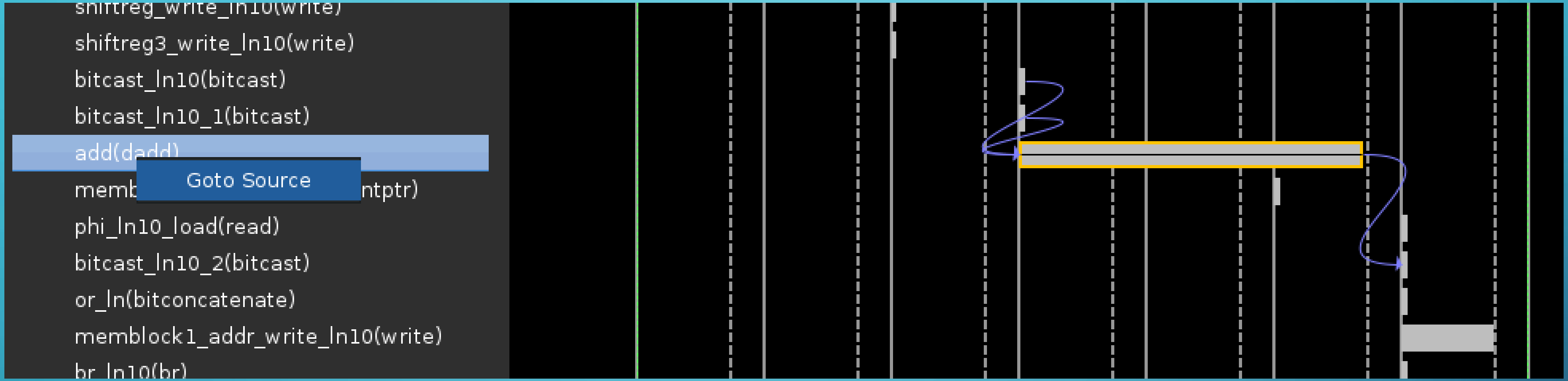
C SYNTHESIS



```
shiftreg_write_ln0(write)
shiftreg3_write_ln0(write)
phi_ln10_write_ln0(write)
br_ln0(br)
VITIS_LOOP_8_1
    i_1(read)
    icmp_ln8(icmp)
    add_ln8(+)
    br_ln8(br)
    empty_22(trunc)
```

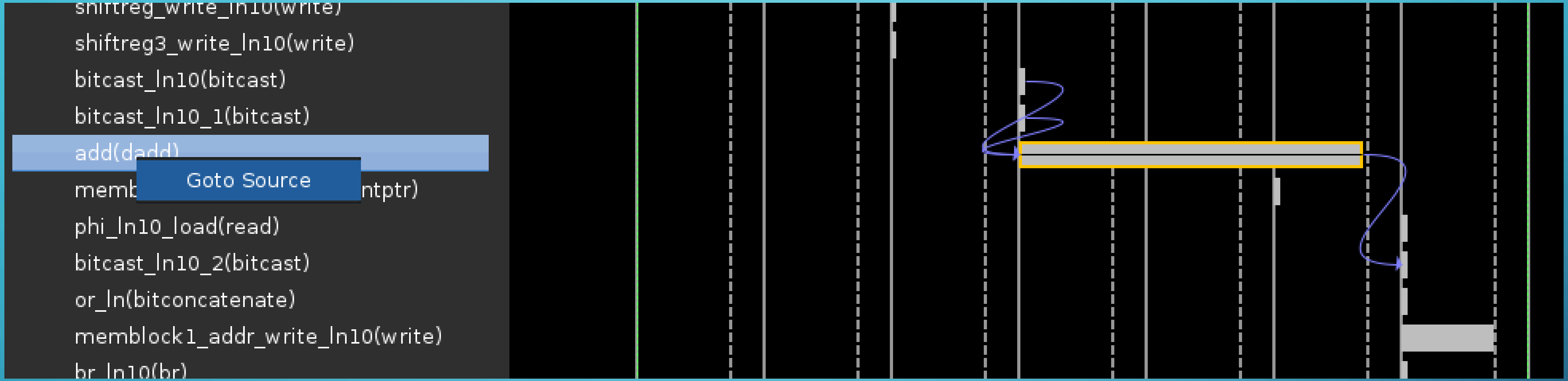
- Expand the loop by clicking on VITIS_LOOP_8_1
- The blue bar shows the length of the loop
- The darker divisions in the bar show the start of each new iteration of a loop
- Because the Initiation Interval is 1, there is a new iteration every clock cycle

C SYNTHESIS


shiftreg_write_ln10(write)
shiftreg3_write_ln10(write)
bitcast_ln10(bitcast)
bitcast_ln10_1(bitcast)
add(dadd)
memb ntptr)
phi_ln10_load(read)
bitcast_ln10_2(bitcast)
or_ln(bitconcatenate)
memblock1_addr_write_ln10(write)
br_ln10(br)

- Scroll down to find the read, write, and add operations.
- Double addition here takes three cycles
- Right click and select Goto Source to bring up the C source code for that operation#
- Arrows in and out of operation show dependencies

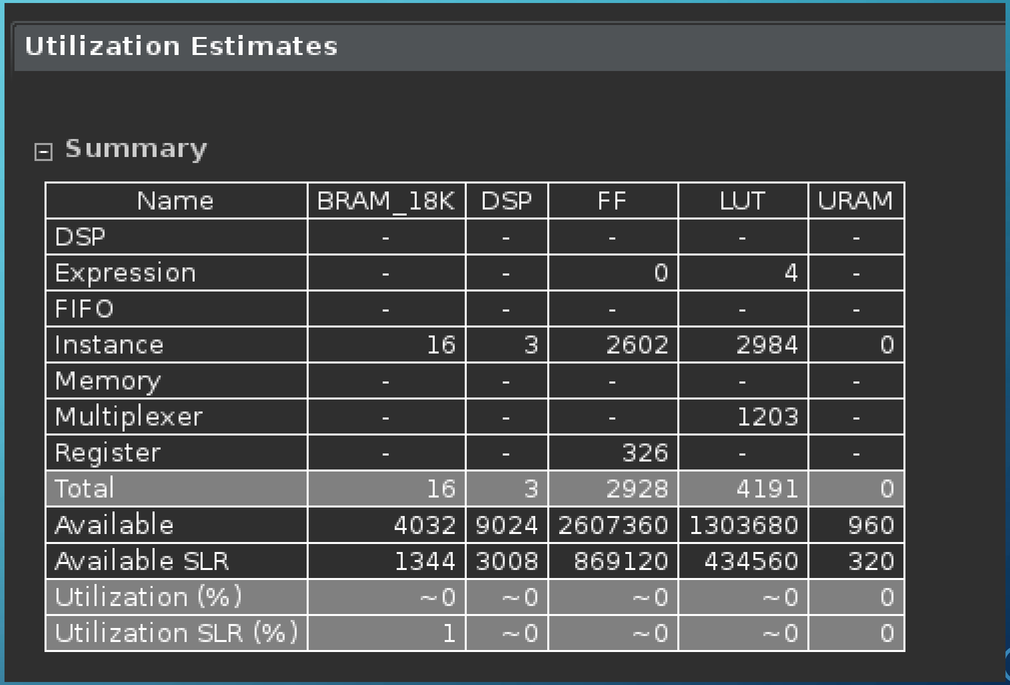
C SYNTHESIS


shiftreg_write_ln10(write)
shiftreg3_write_ln10(write)
bitcast_ln10(bitcast)
bitcast_ln10_1(bitcast)
add(dadd)
memb ntptr)
phi_ln10_load(read)
bitcast_ln10_2(bitcast)
or_ln(bitconcatenate)
memblock1_addr_write_ln10(write)
br_ln10(br)

- Scroll down to find the read, write, and add operations.
- Double addition here takes three cycles
- Right click and select Goto Source to bring up the C source code for that operation#
- Arrows in and out of operation show dependencies

C SYNTHESIS

- You can also right click on an item in the Performance and Resource Estimates and click on "Open Synthesis Details Report"
- This gives you similar information about usage, but also tells you the total you have available to you, and the amount available in this SLR (super logic region)
- Can see your resource utilisation as a percentage of your available resources

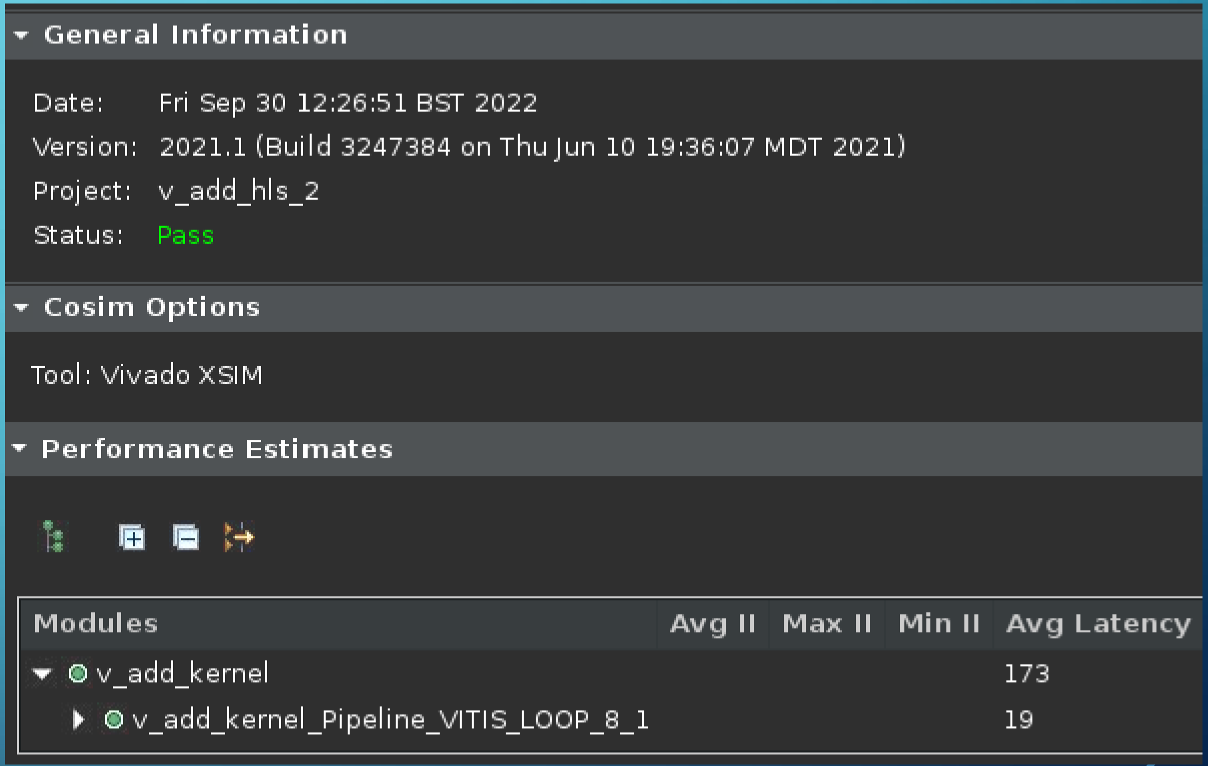

Utilization Estimates

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	4	-
FIFO	-	-	-	-	-
Instance	16	3	2602	2984	0
Memory	-	-	-	-	-
Multiplexer	-	-	-	1203	-
Register	-	-	326	-	-
Total	16	3	2928	4191	0
Available	4032	9024	2607360	1303680	960
Available SLR	1344	3008	869120	434560	320
Utilization (%)	~0	~0	~0	~0	0
Utilization SLR (%)	1	~0	~0	~0	0

C / RTL COSIMULATION

- You can just take the defaults here again
- Runs your test-bench again
- Hardware emulation takes a while – don't use large problem sizes
- Should tell you everything has passed!
- If not, you can use return codes or hls::print statements to explore problems
- Gives updated latency estimates (see Vitis Tutorials for more info on writing good test benches)



General Information

```
Date:     Fri Sep 30 12:26:51 BST 2022
Version:  2021.1 (Build 3247384 on Thu Jun 10 19:36:07 MDT 2021)
Project:  v_add_hls_2
Status:   Pass
```

Cosim Options

```
Tool: Vivado XSIM
```

Performance Estimates

Modules	Avg II	Max II	Min II	Avg Latency
v_add_kernel				173
v_add_kernel_Pipeline_VITIS_LOOP_8_1				19