HLS Hero Workshop

Course Workbook

Table of Contents

About this Workbook

The contents of this workbook are created by Adiuvo Engineering & Training, Ltd.

If you have any questions about the contents, or need assistance, please contact Adam Taylor at adam@adiuvoengineering.com.

Pre-Lab

HLS Hero

Required Hardware

None Required

Downloads and Installations

Step 1 – Download and install the following at least 1 day prior to the workshop. This may take a significant amount of time and drive space.

Watch the video available here to show how to configure the installation

Vitis 2021.2	Download

Lab 1

Vitis HLS Flow

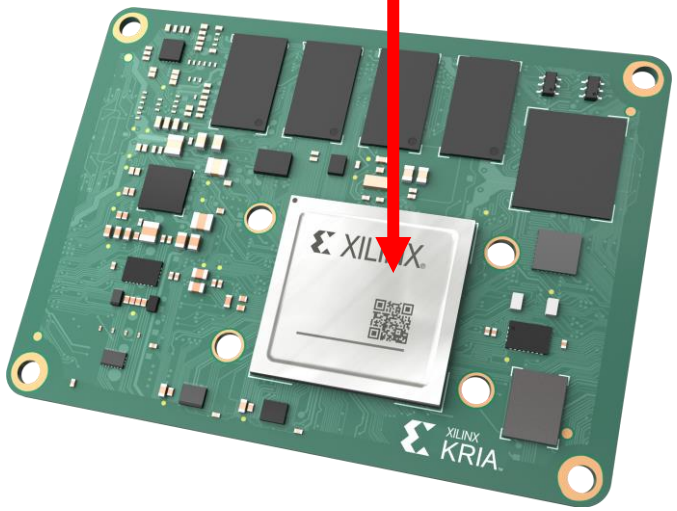
What is HLS

High Level Synthesis (HLS) enables generation of RTL modules from higher level language such as C, C++, OpenCL

Of course, SW engineers still consider these low-level languages.

HLS offers several benefits for the development of Signal / Data / Image processing algorithms

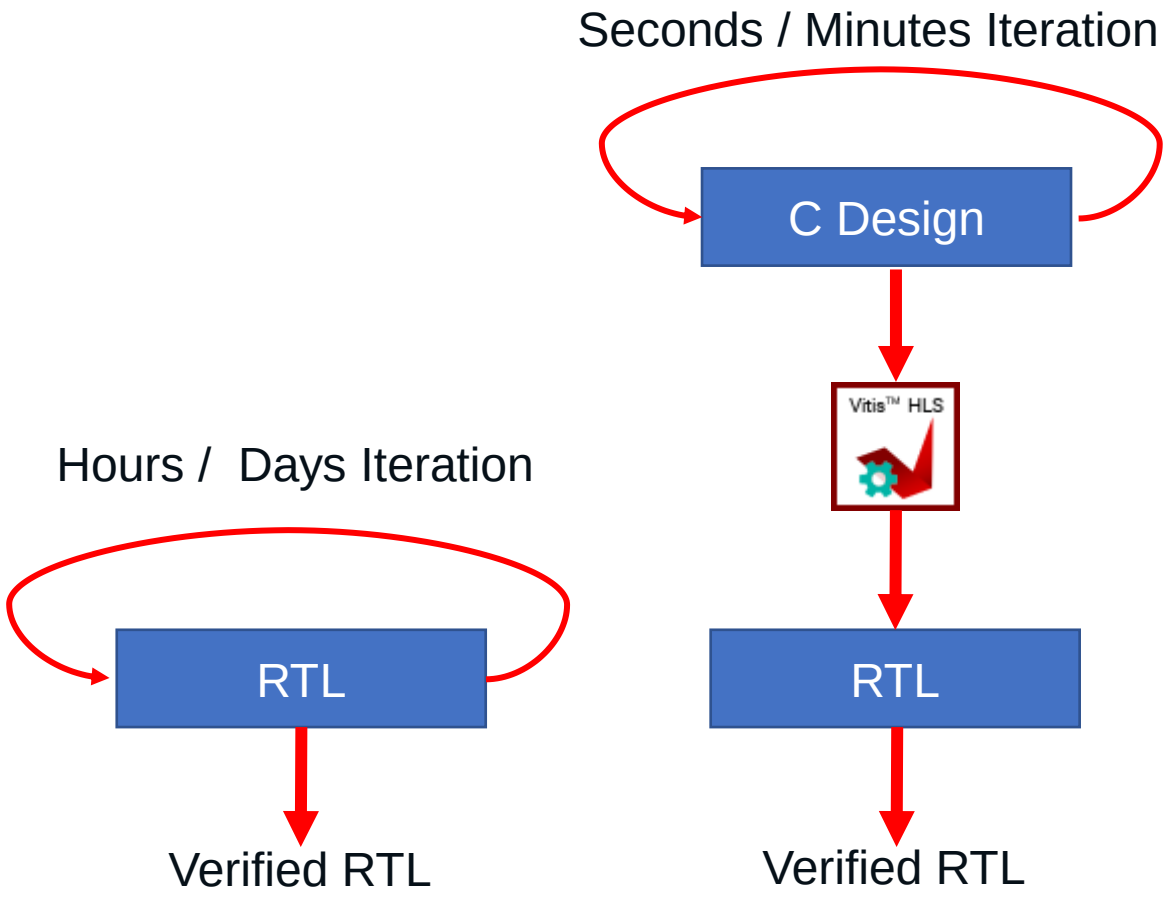
```
    error = set_point - sample;

    p = error * KP;
    i = i_prev + (error  * ts * KI);
    d = KD * ((error - error_prev) / ts);

    op = p+i+d;
    error_prev = error;
    if (op > pmax)  {
        i_prev = i_prev;
        op = pmax;
    }else{
        i_prev = i;
    }
    return op;
}
```



HLS Benefits

Developing in Higher Level language enables a faster iteration time.

- Development Time decreased as untimed language – No RTL / Behavioral level
- Increased level of abstraction – accelerates development
- Verification time is reduced as untimed Simulation



Creating HLS Solutions

Software written for CPUs and software written for FPGAs is fundamentally different

1. Not all C constructs can be synthesised
2. Learn about synthesizable C/C++ coding styles.
3. Need to focus on correct micro architecture
    1. Understand the producers and consumers
    2. Decompose the algorithm into small section which interconnect
    3. Understand throughput required for each element to achieve overall performance goals
4. Learn how to interpret the design reports

HLS Flow

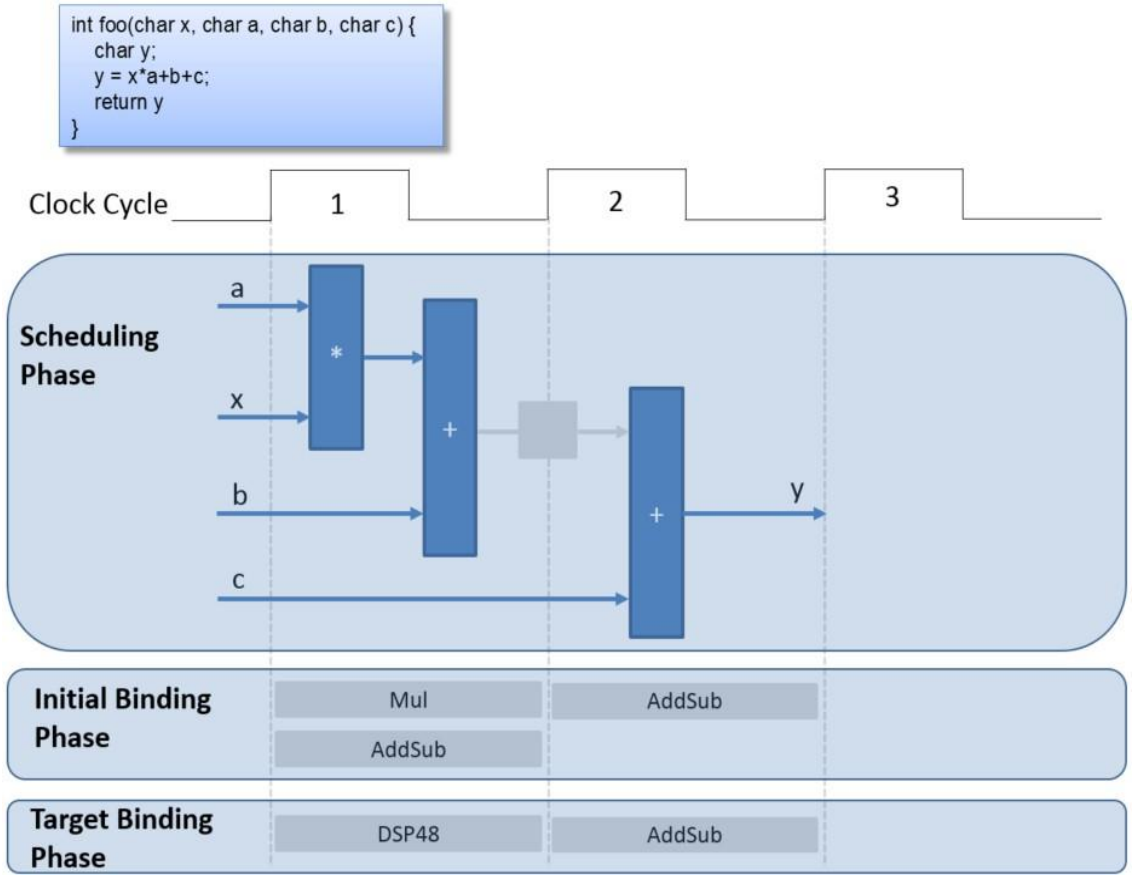
Create Solution, Simulate & Debug	Create the C Module and Test Bench – Leverage Libraries where possible Verify algorithm performance in SW
Synthesize	Convert C Module to RTL
Analyse & Optimize	Optimize Interfaces and Performance of synthesized implementation
Co Simulation	Simulation of the RTL module against the C Test Bench
Package the IP	Package for use in Vitis (XO) or Vivado (Xact IP)

Untimed to Timed

Scheduling
- Determines which operations occur during each clock cycle

Binding
- Determines which hardware resource implements each scheduled operation

Control logic extraction
- Extracts the control logic to create a finite state machine (FSM) that sequences the operations in the RTL design

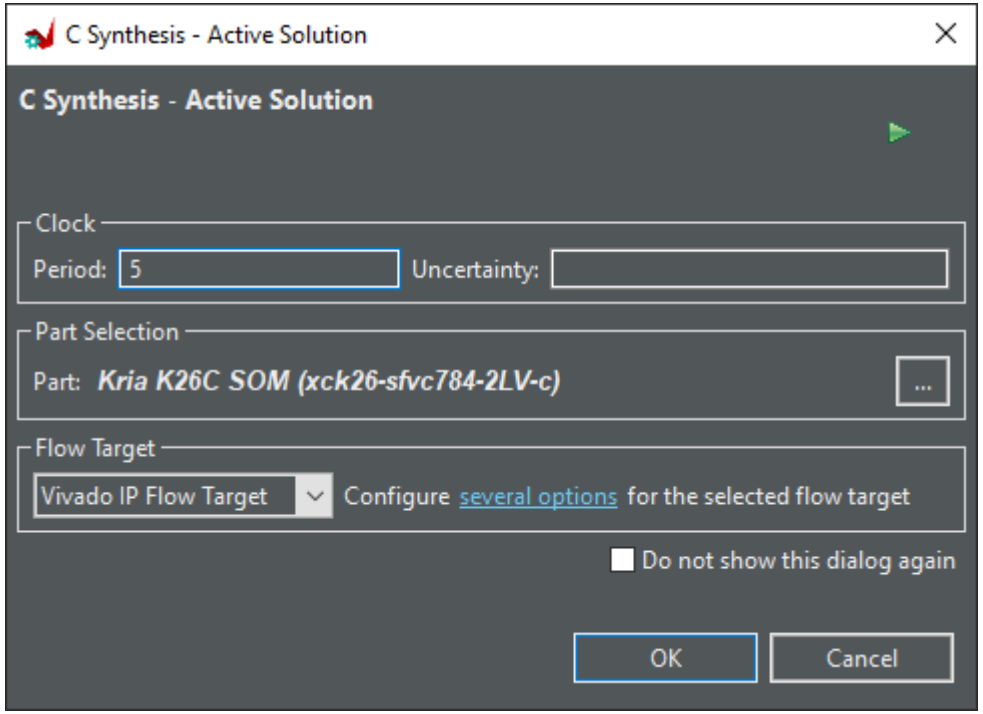


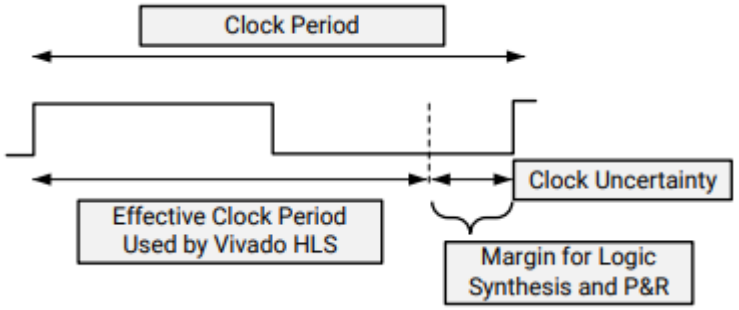
Clock Definition

Clock frequency & selected device used to determine timing of operations – Scheduling Phase

Uncertainty leaves time for final component placement and net routing

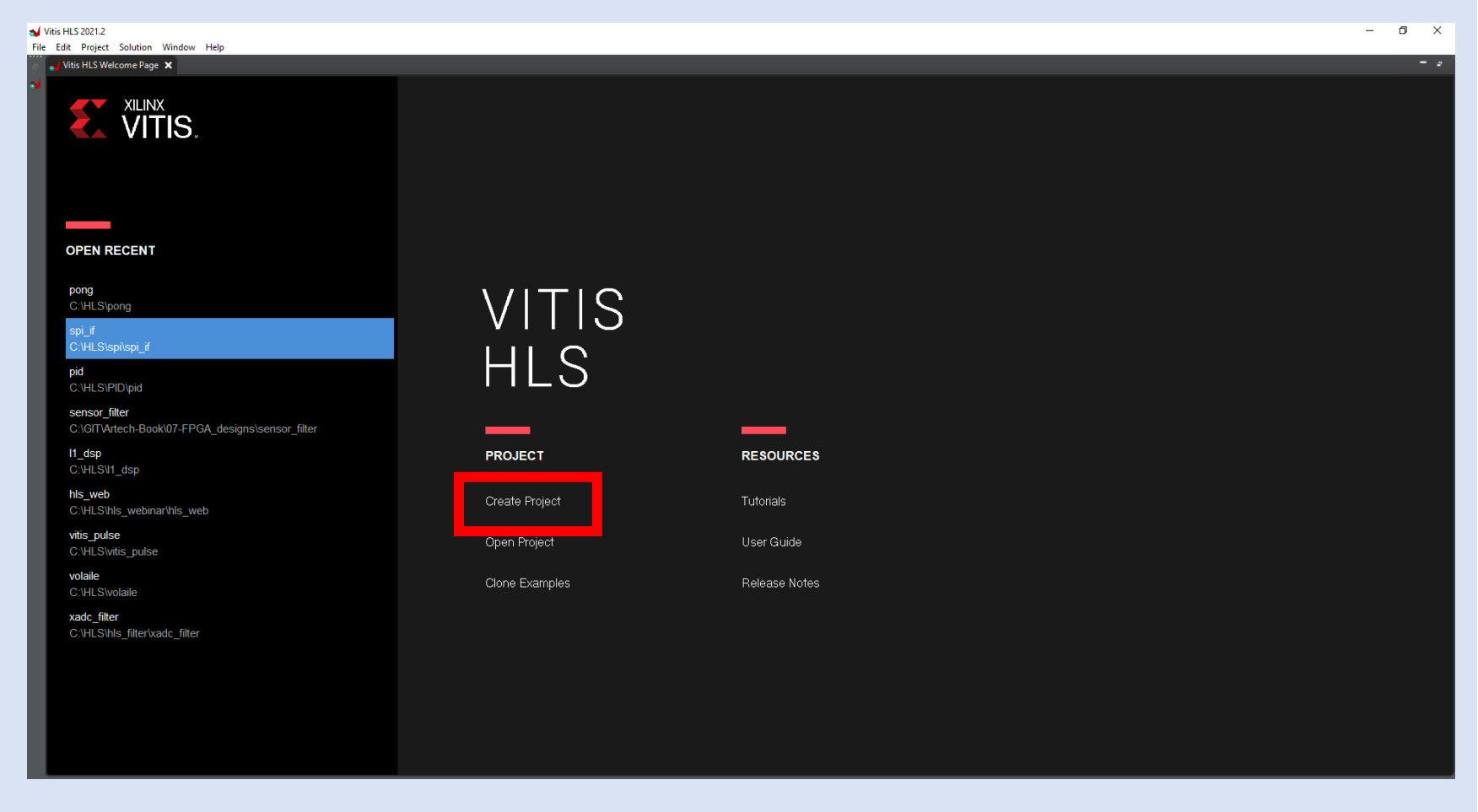
If left blank 27% is used





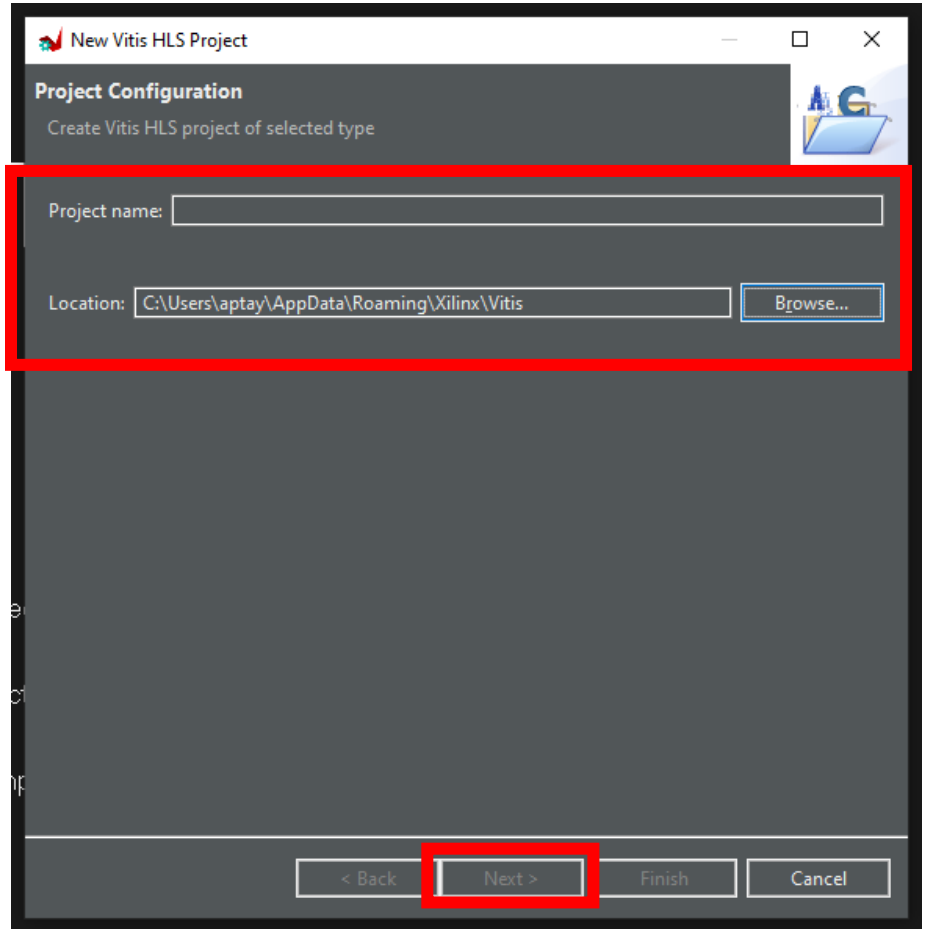
Lab 1: Vitis HLS Flow

Step 1 – Open Vitis HLS 2021.2 and create a new project



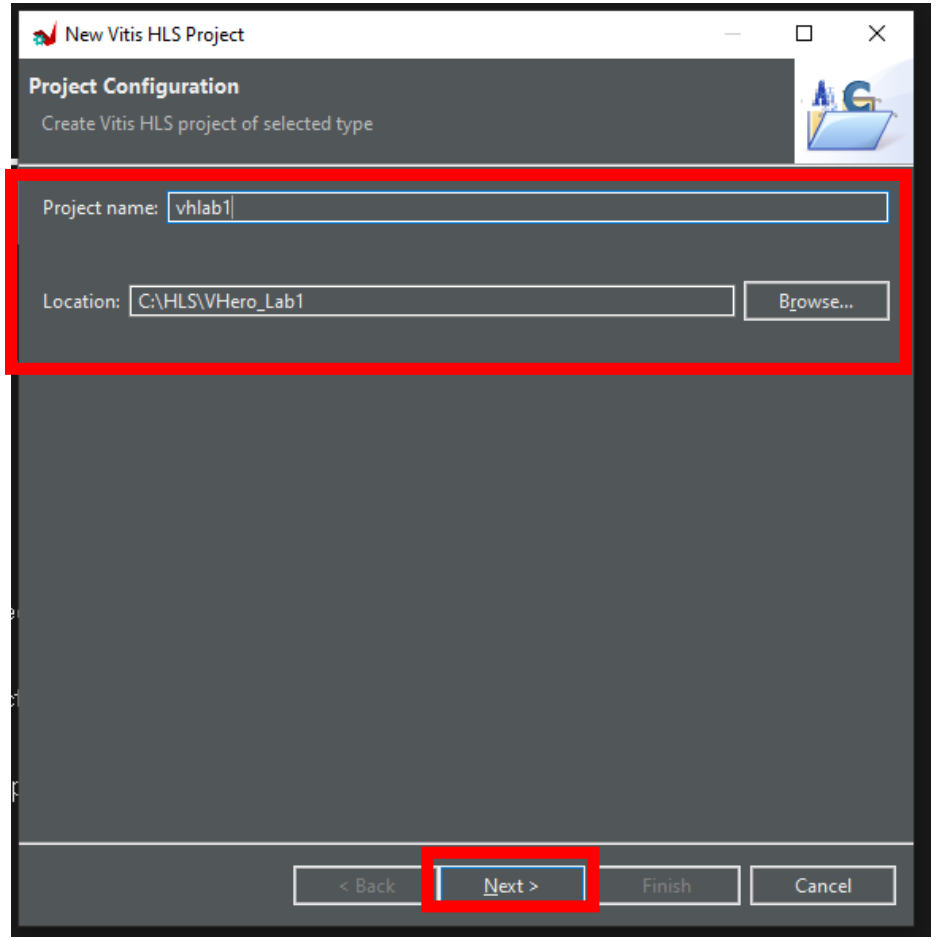
Lab 1: Vitis HLS Flow

Step 1 – Enter a project name and select a location and click next



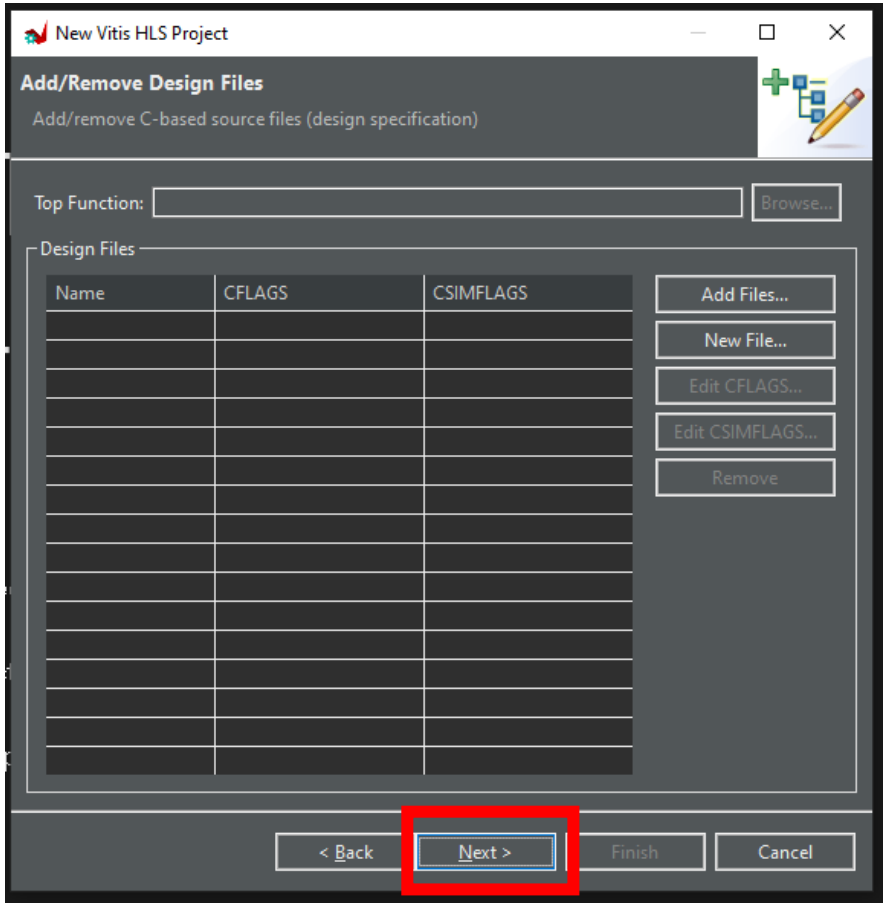
Lab 1: Vitis HLS Flow

Step 1 – I used the name VHlab1



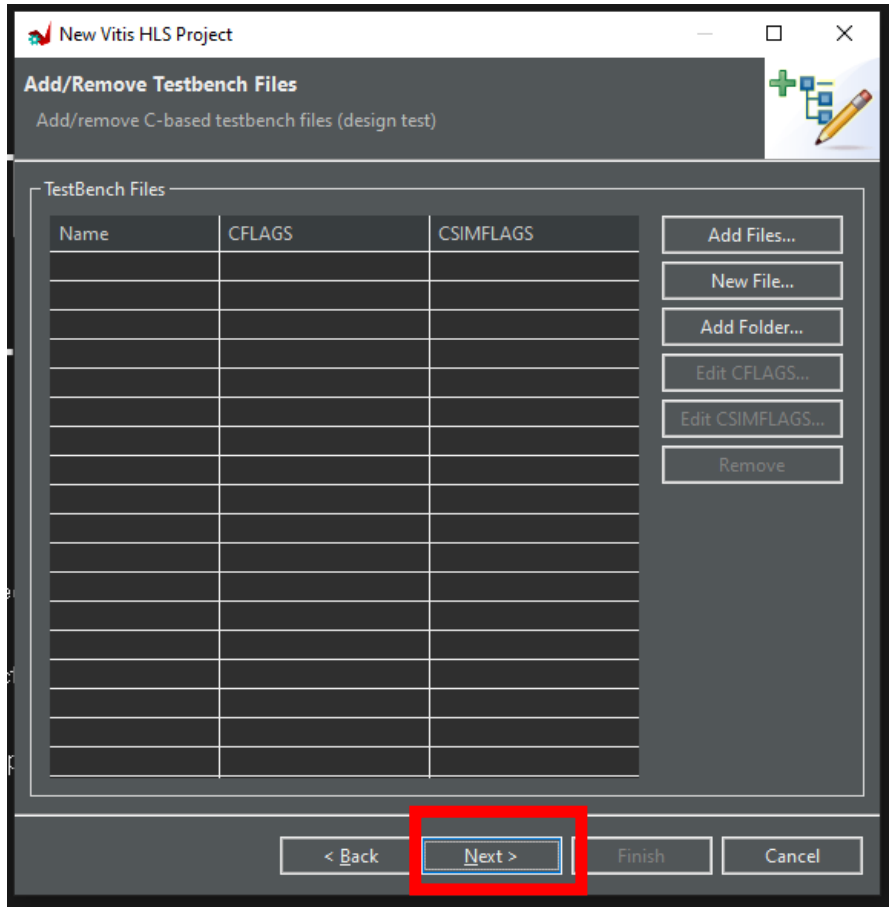
Lab 1: Vitis HLS Flow

Step 1 – Click Next



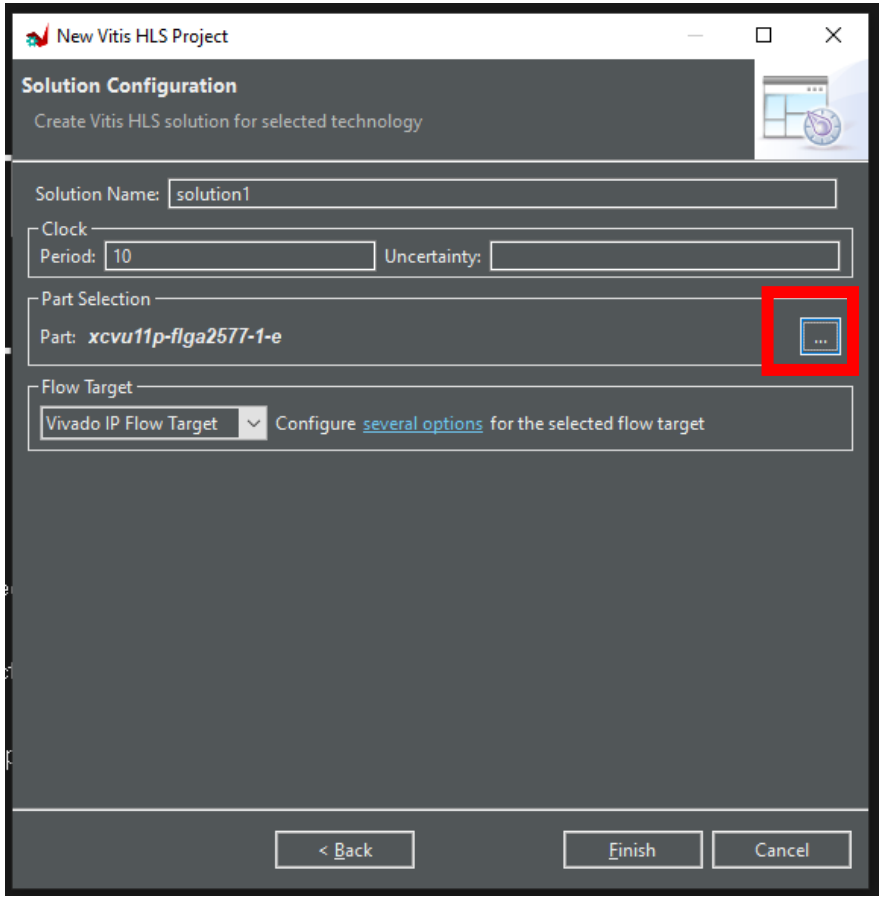
Lab 1: Vitis HLS Flow

Step 1 – Click Next



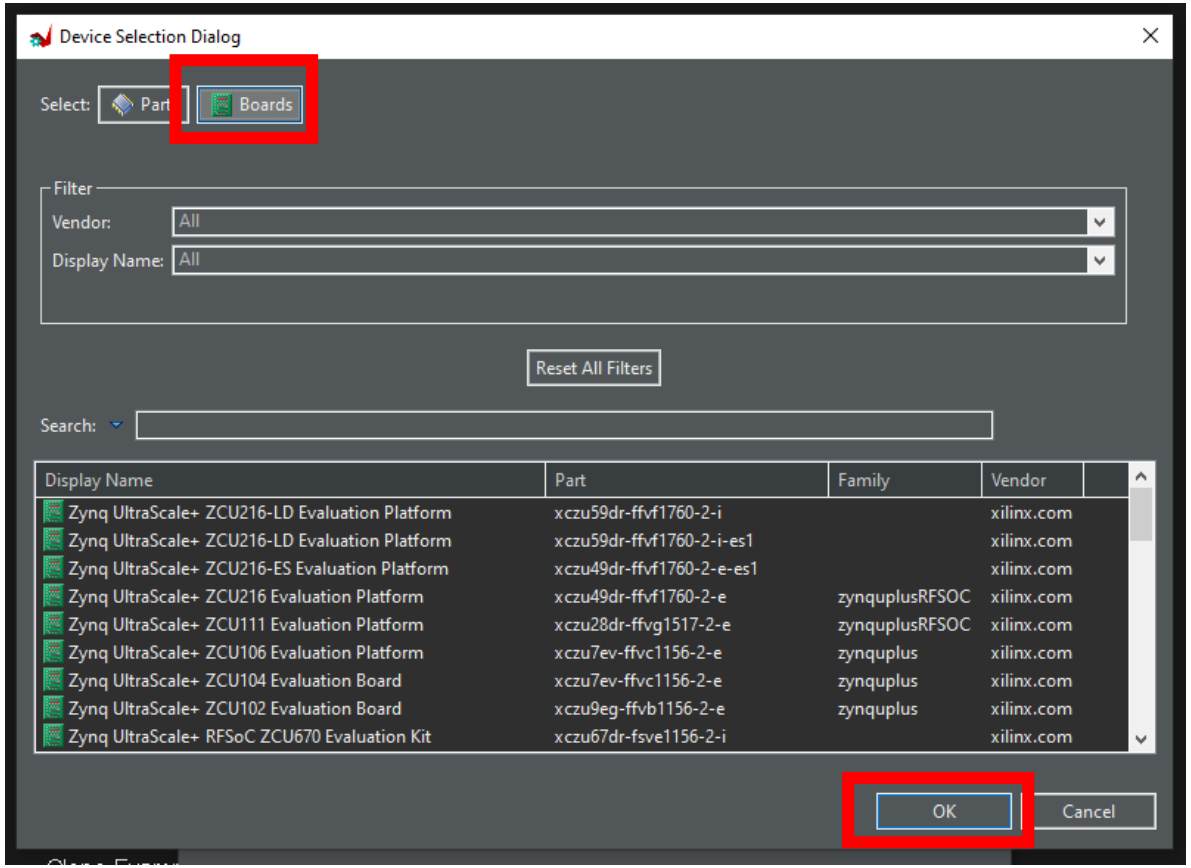
Lab 1: Vitis HLS Flow

Step 1 – Leave all unchanged except for part selection, click to select



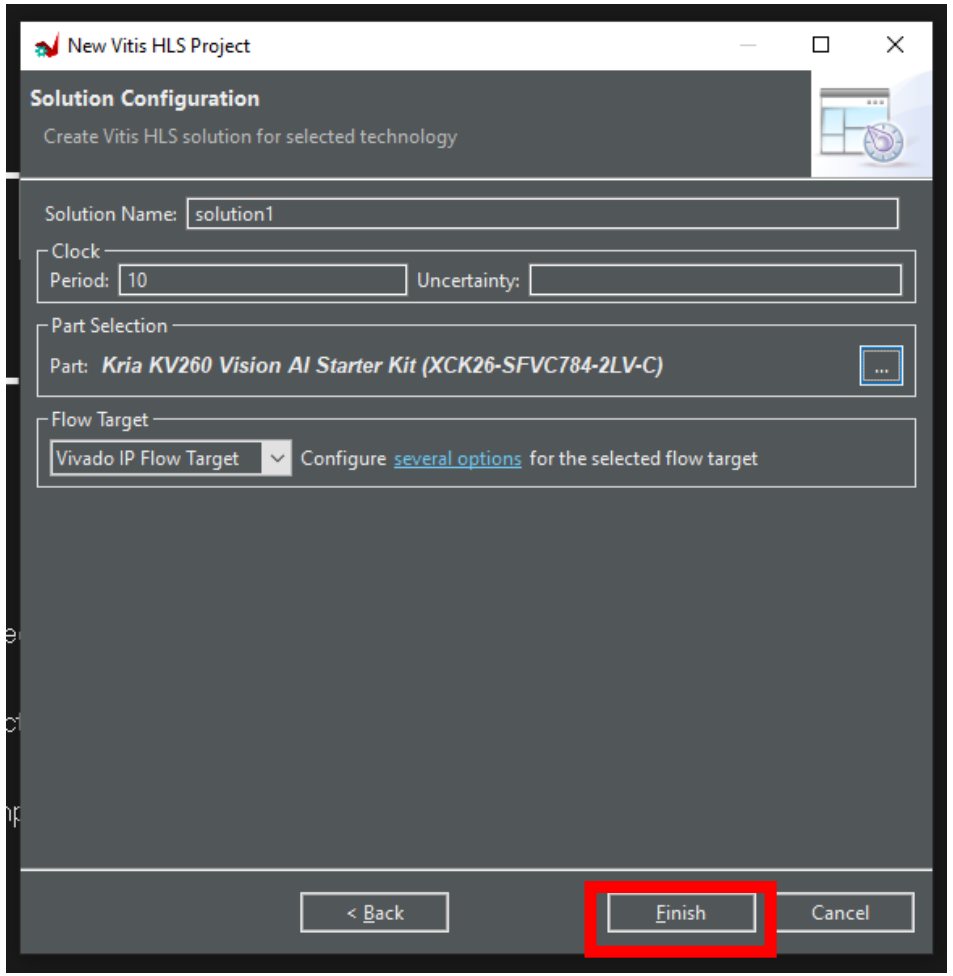
Lab 1: Vitis HLS Flow

Step 1 – Select board and select the SP701 Spartan 7 Board, Click OK



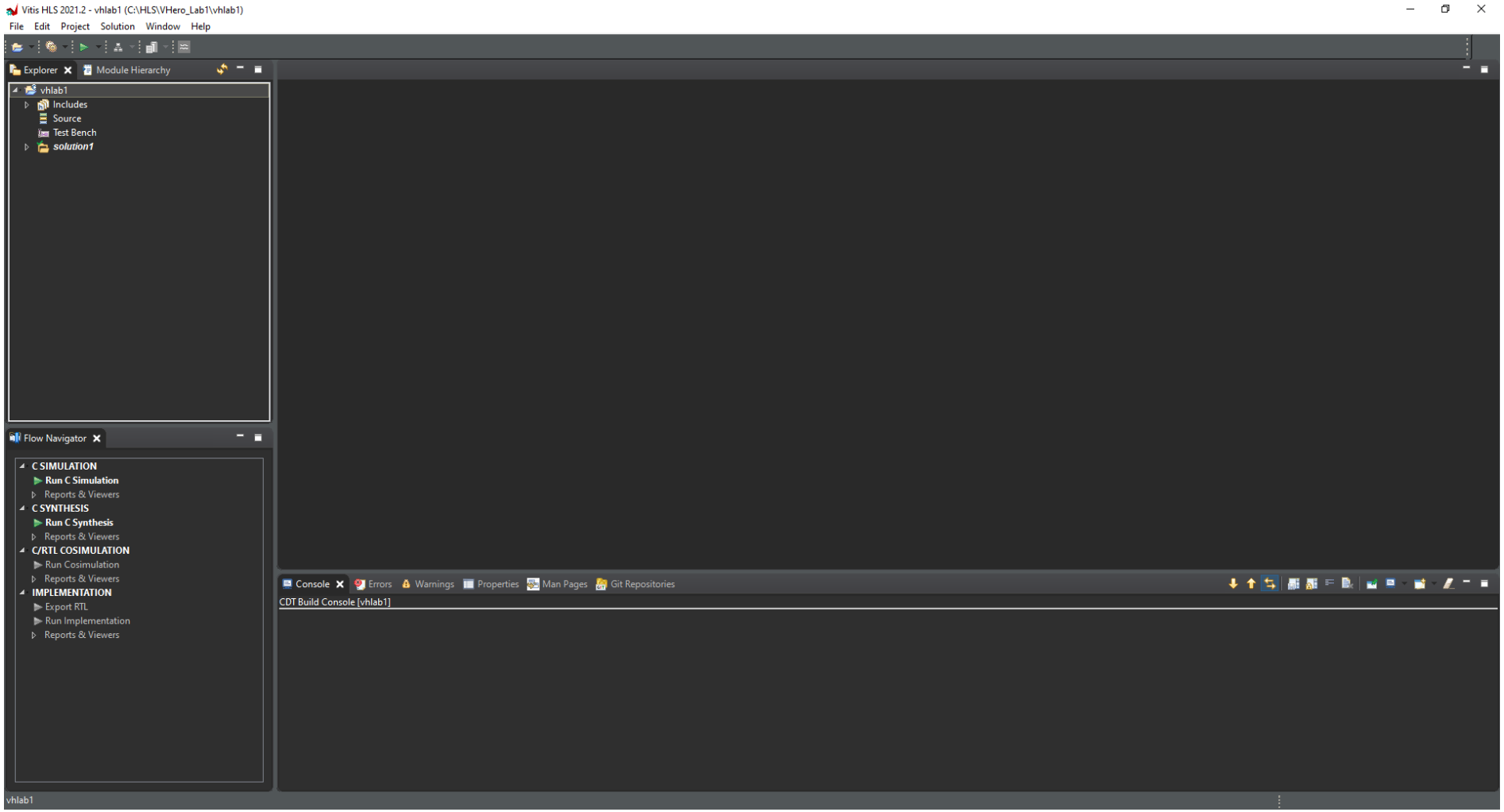
Lab 1: Vitis HLS Flow

Step 1 – Click Finish to create the project



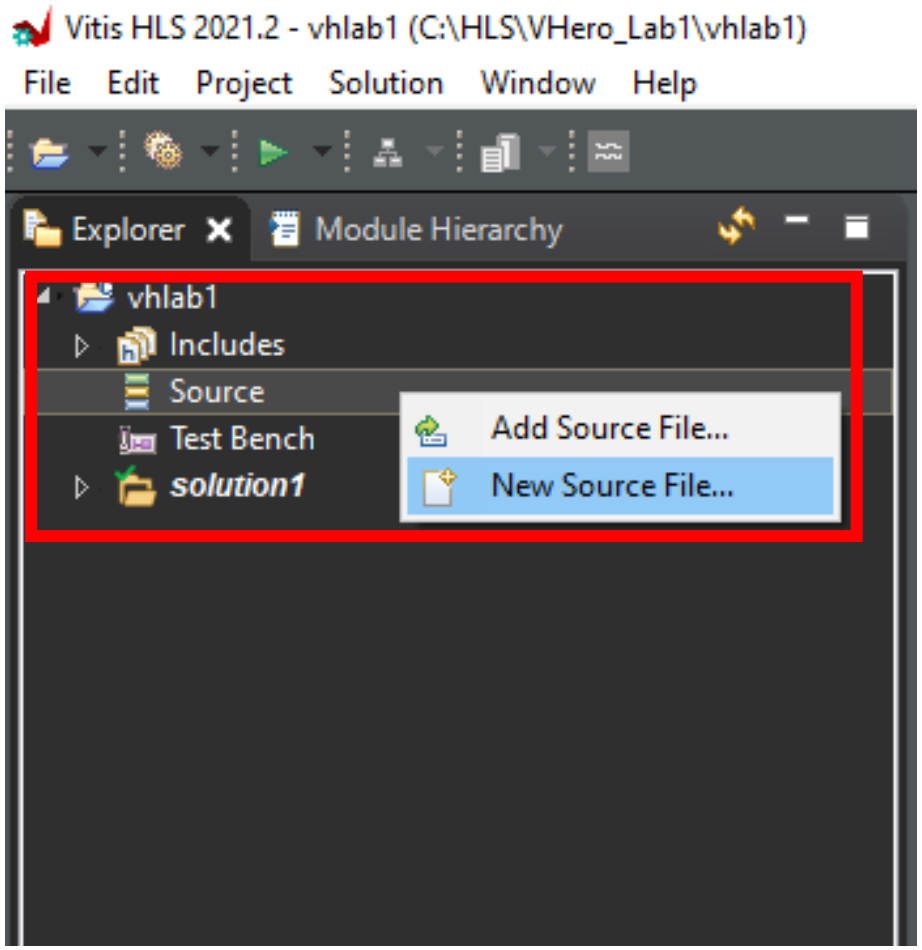
Lab 1: Vitis HLS Flow

Step 1 – This will open the project ready for us to create the application



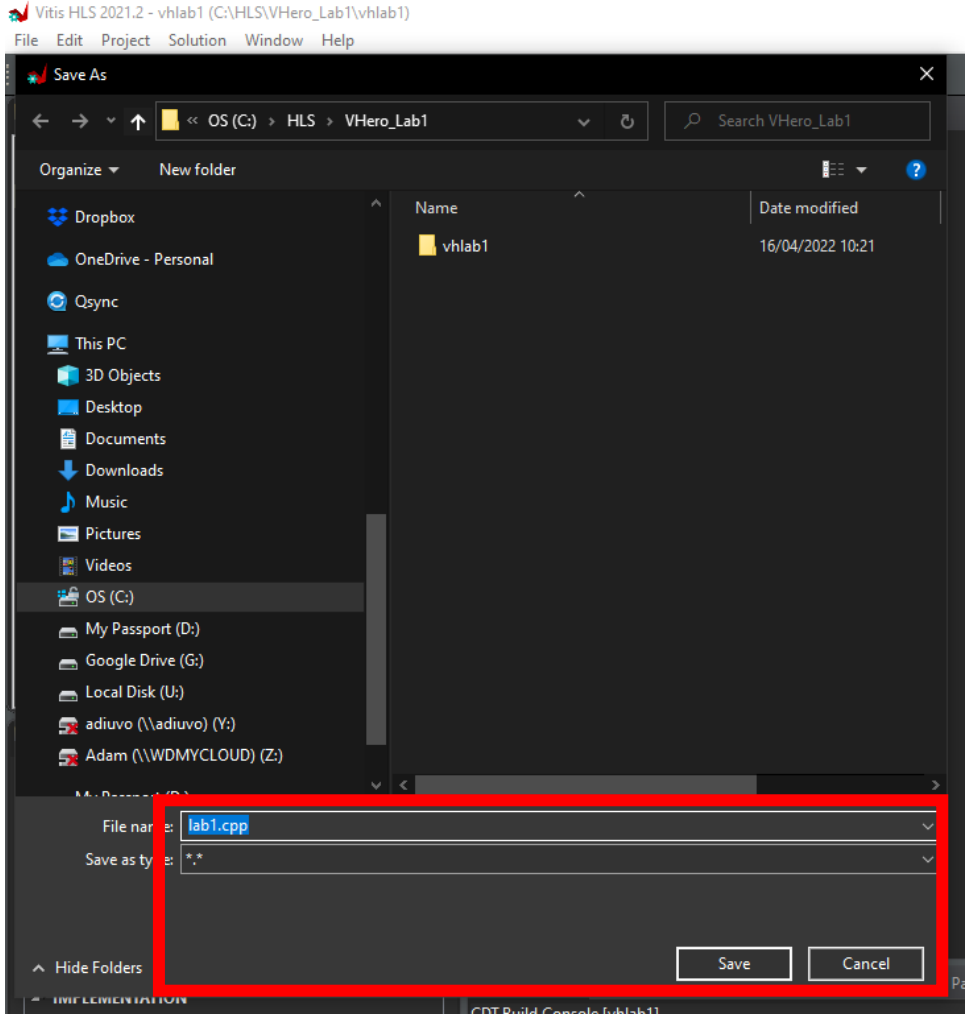
Lab 1: Vitis HLS Flow

Step 1 – Right Click on Source and Select new Source File



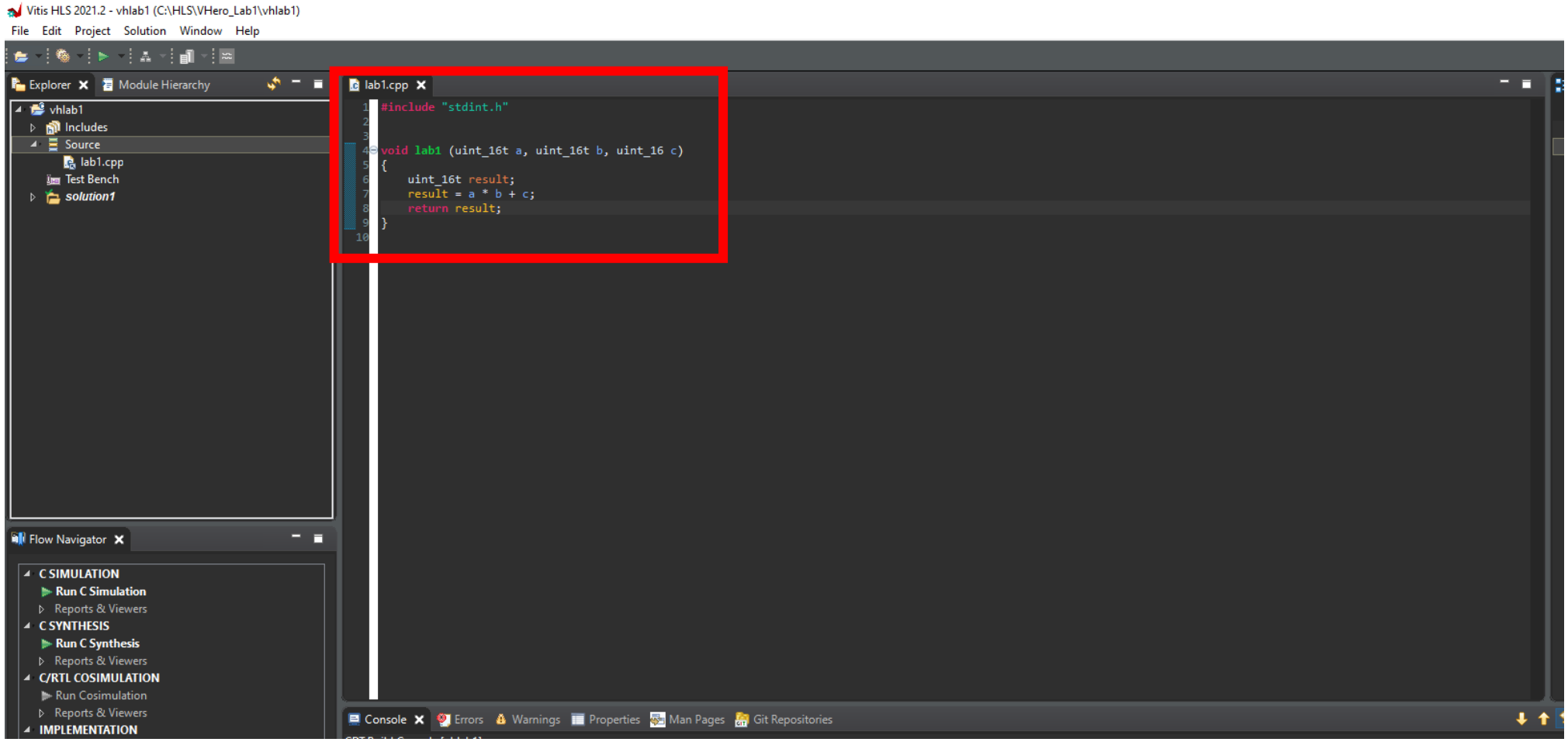
Lab 1: Vitis HLS Flow

Step 1 – Enter the name of the file as lab1.cpp, click save



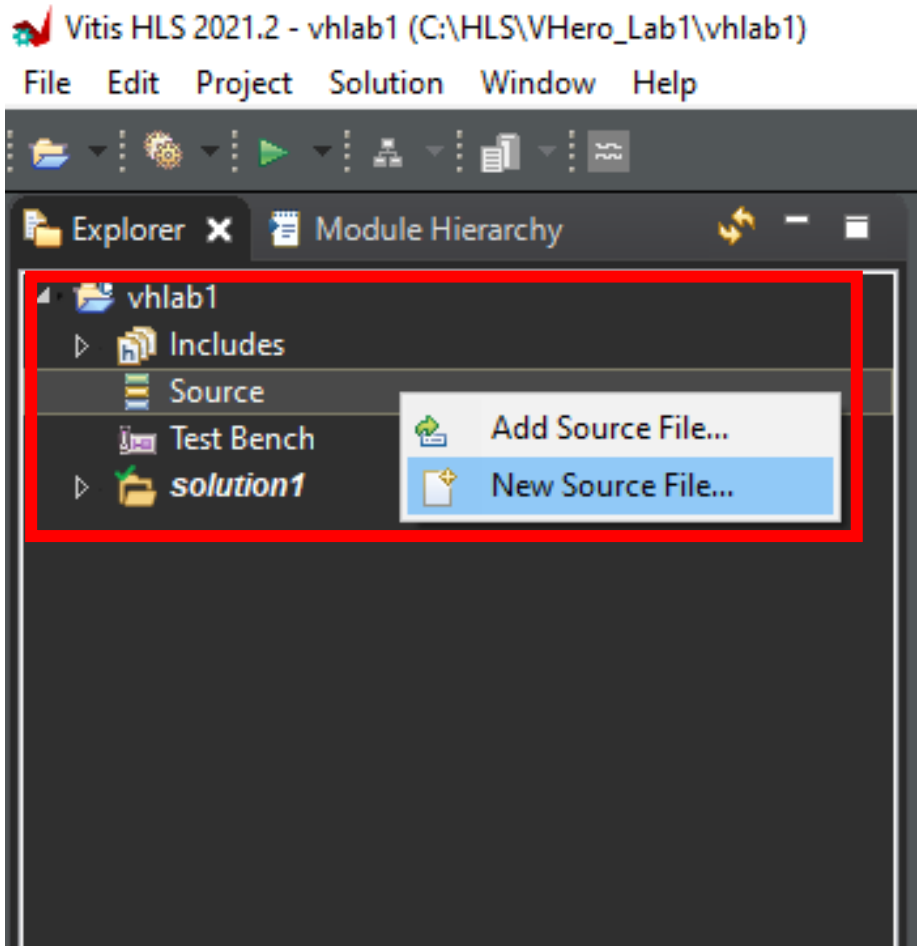
Lab 1: Vitis HLS Flow

Step 1 – Enter the code as below – or Copy from the files on the GitHub. Save the file.



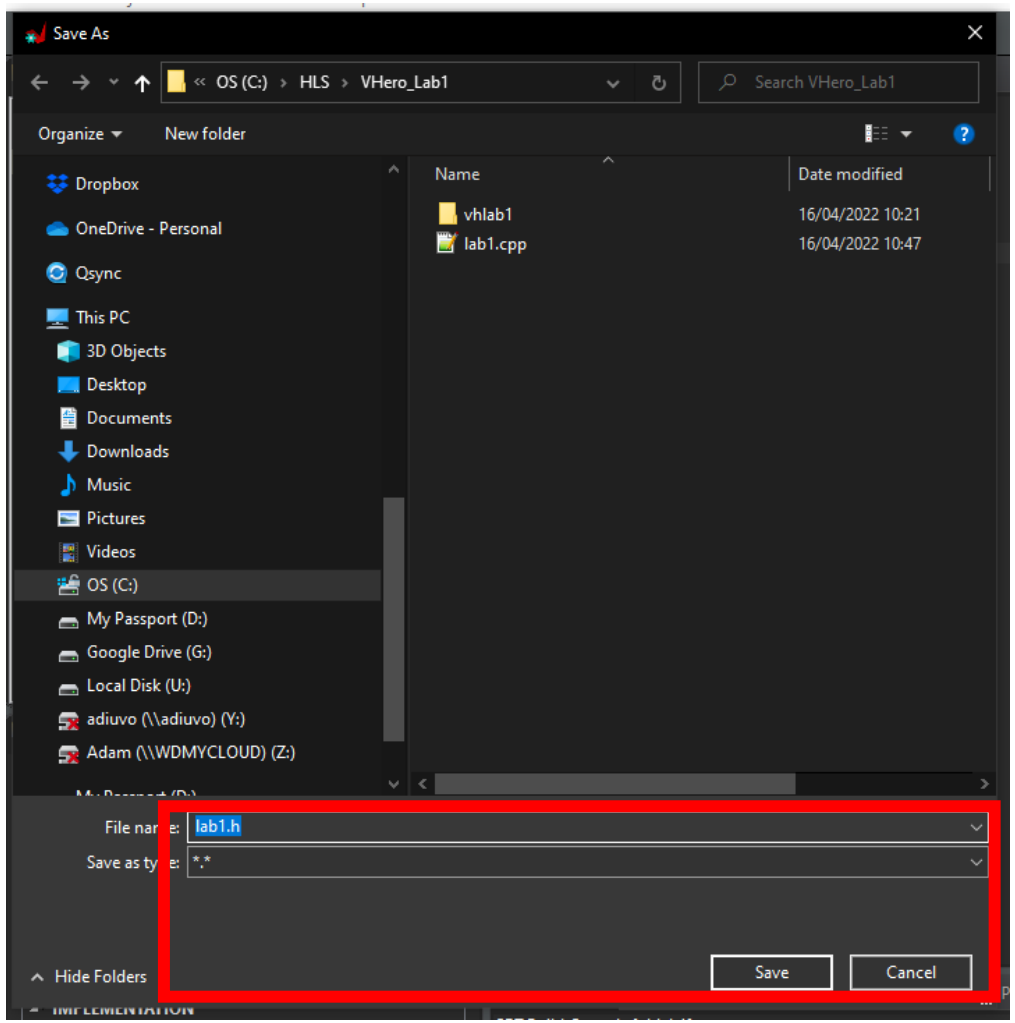
Lab 1: Vitis HLS Flow

Step 1 – Right Click on Source and Select new Source File



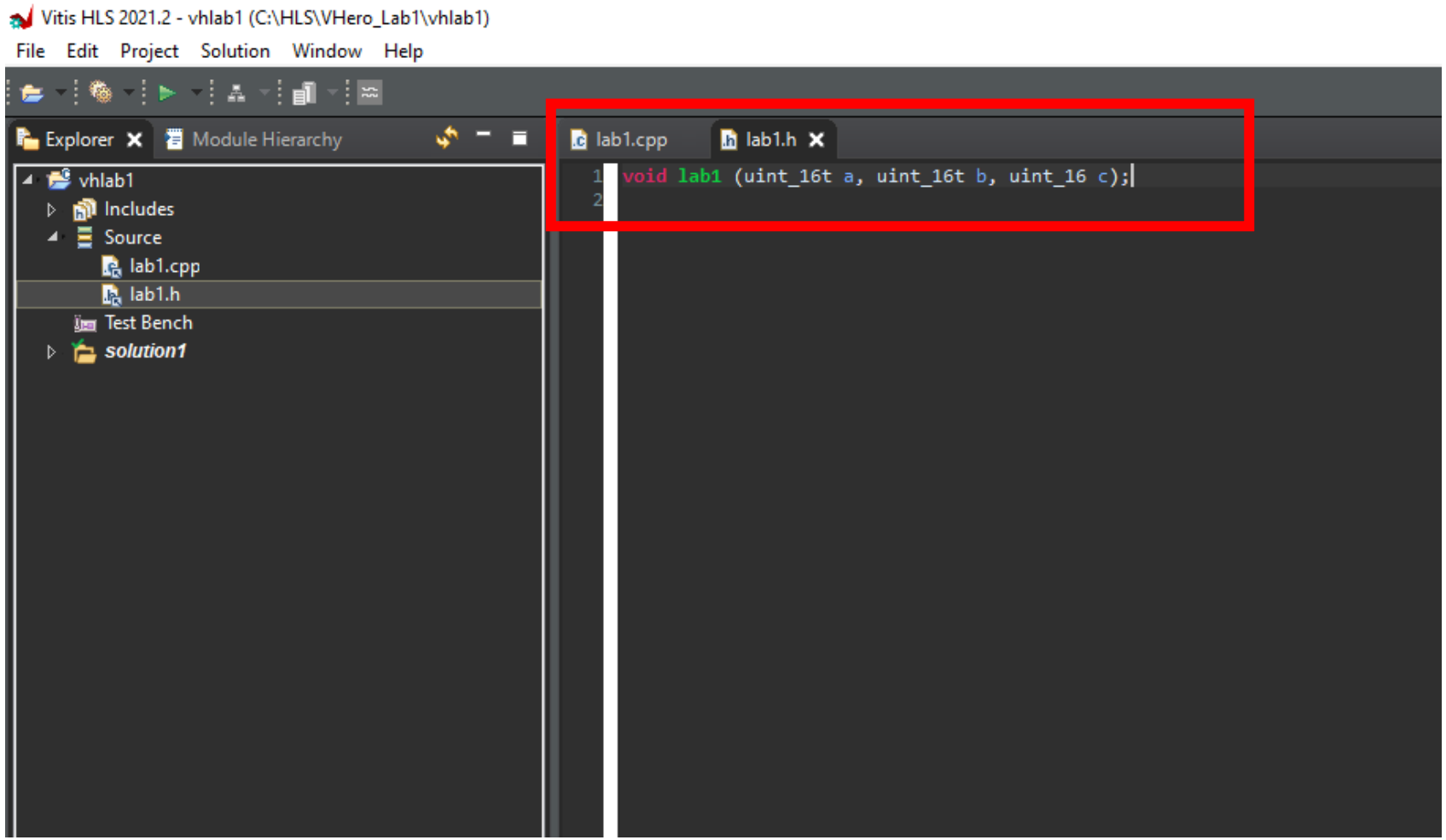
Lab 1: Vitis HLS Flow

Step 1 – Enter the name of the file as lab1.h click save



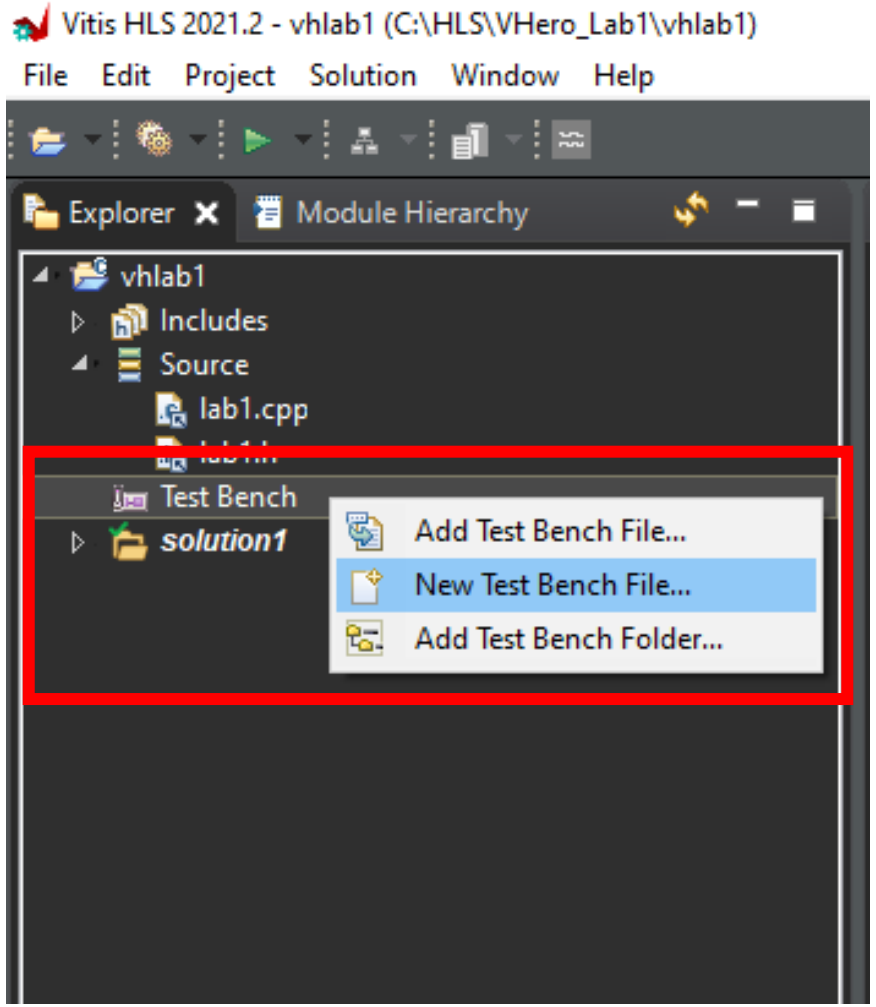
Lab 1: Vitis HLS Flow

Step 1 – Enter the code as below – or Copy from the files on the GitHub. Save the file.



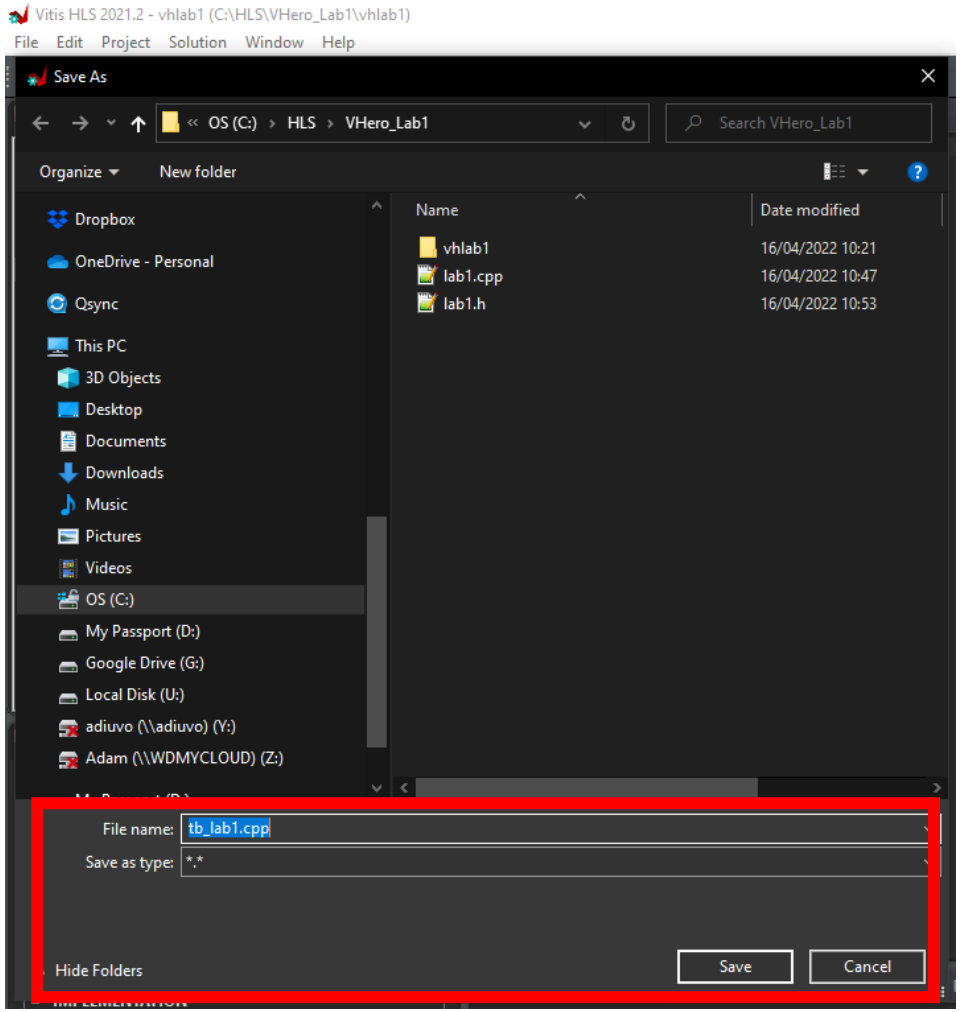
Lab 1: Vitis HLS Flow

Step 1 – Right Click on Test Bench and Select new Test Bench File.



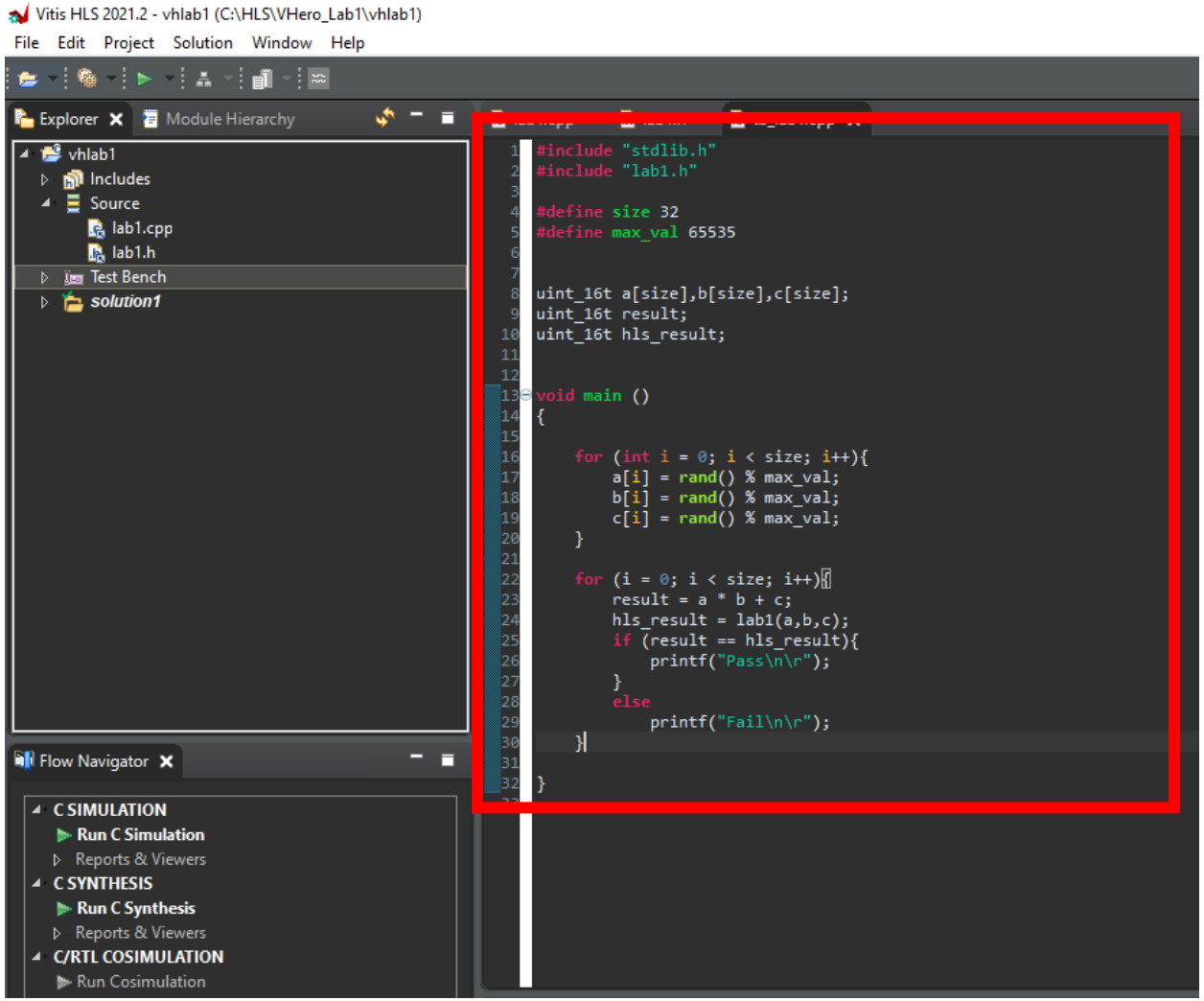
Lab 1: Vitis HLS Flow

Step 1 – Enter the name of the file as tb_lab1.cpp click save



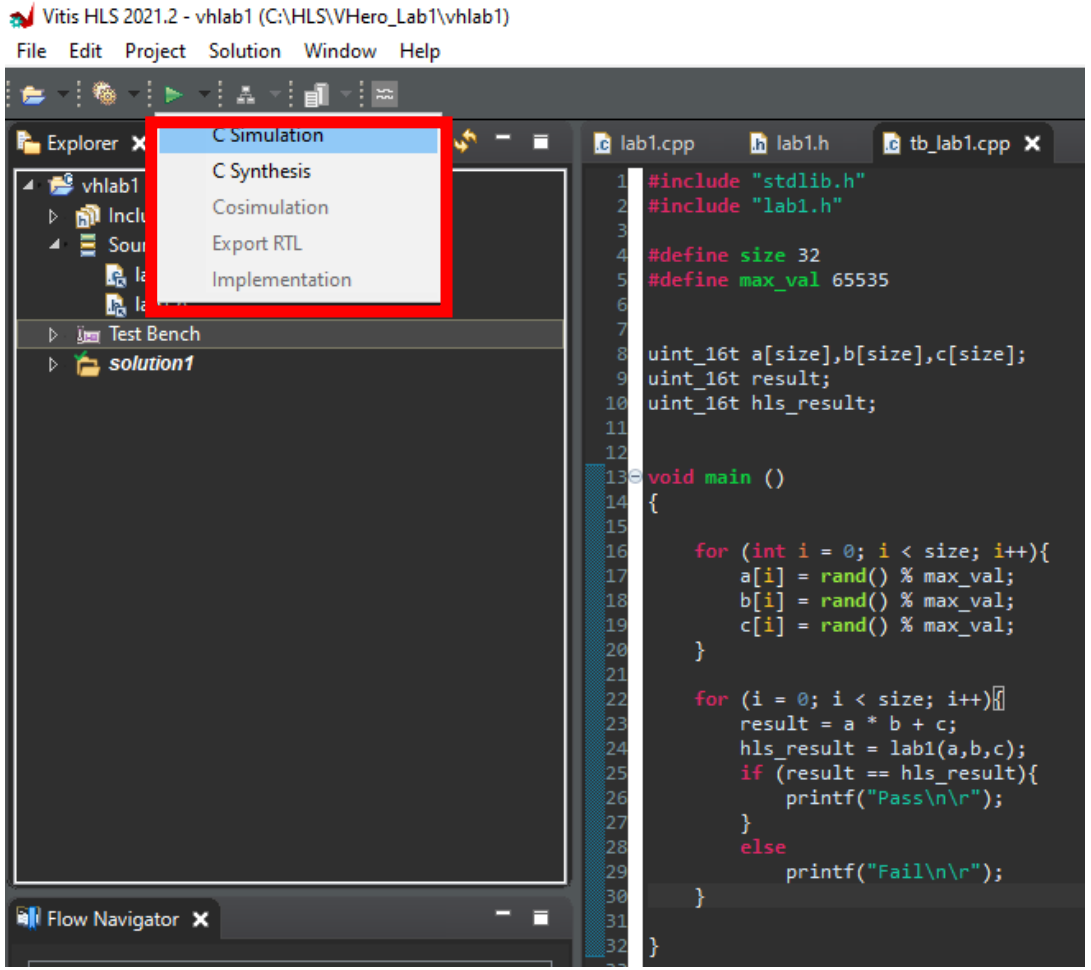
Lab 1: Vitis HLS Flow

Step 1 – Enter the code as below – or Copy from the files on the GitHub. Save the file.



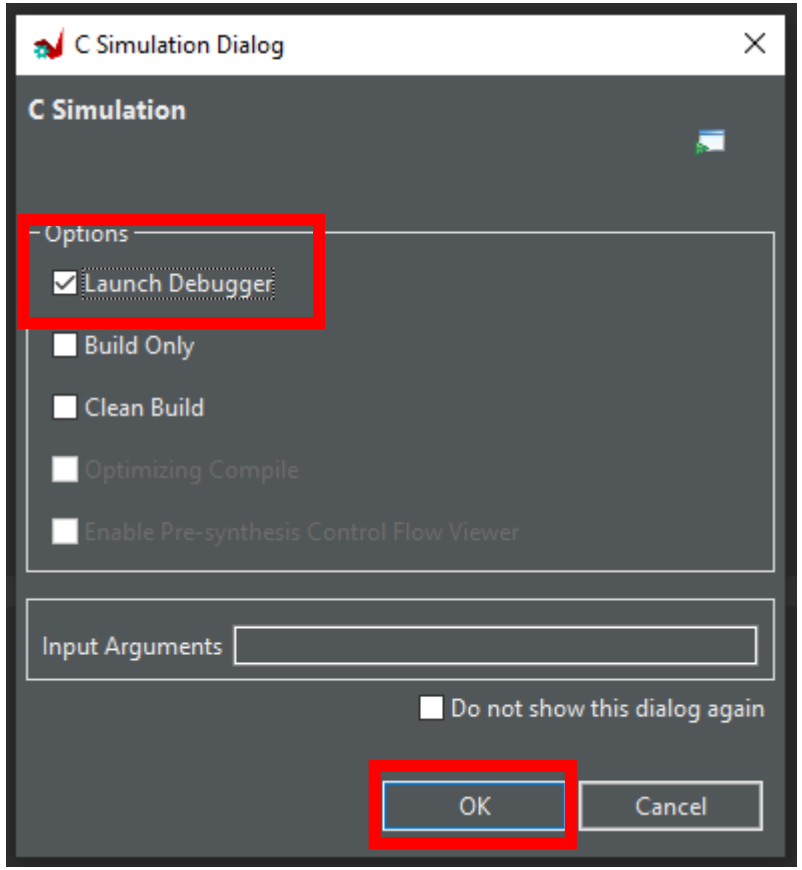
Lab 1: Vitis HLS Flow

Step 1 – From the Run menus select C Simulation



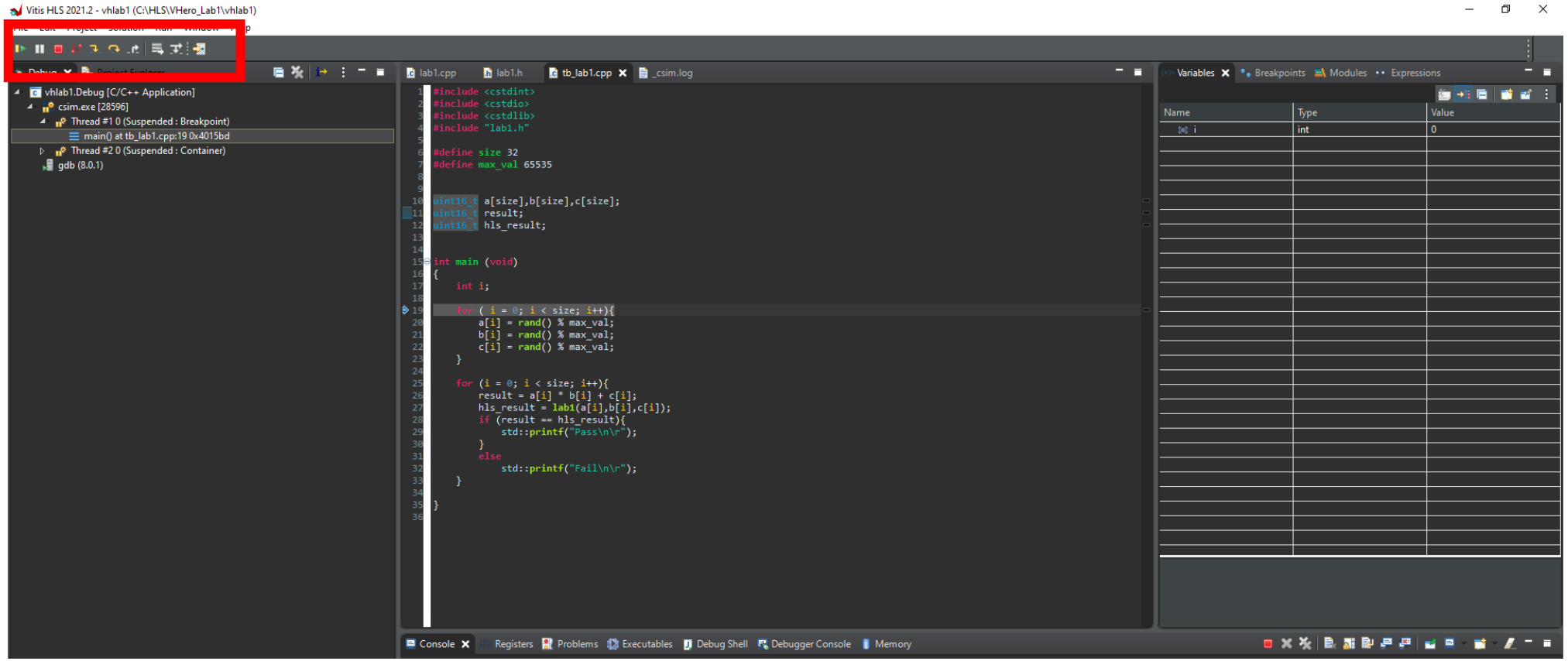
Lab 1: Vitis HLS Flow

Step 1 – On the dialog box select the launch debugger option, click OK



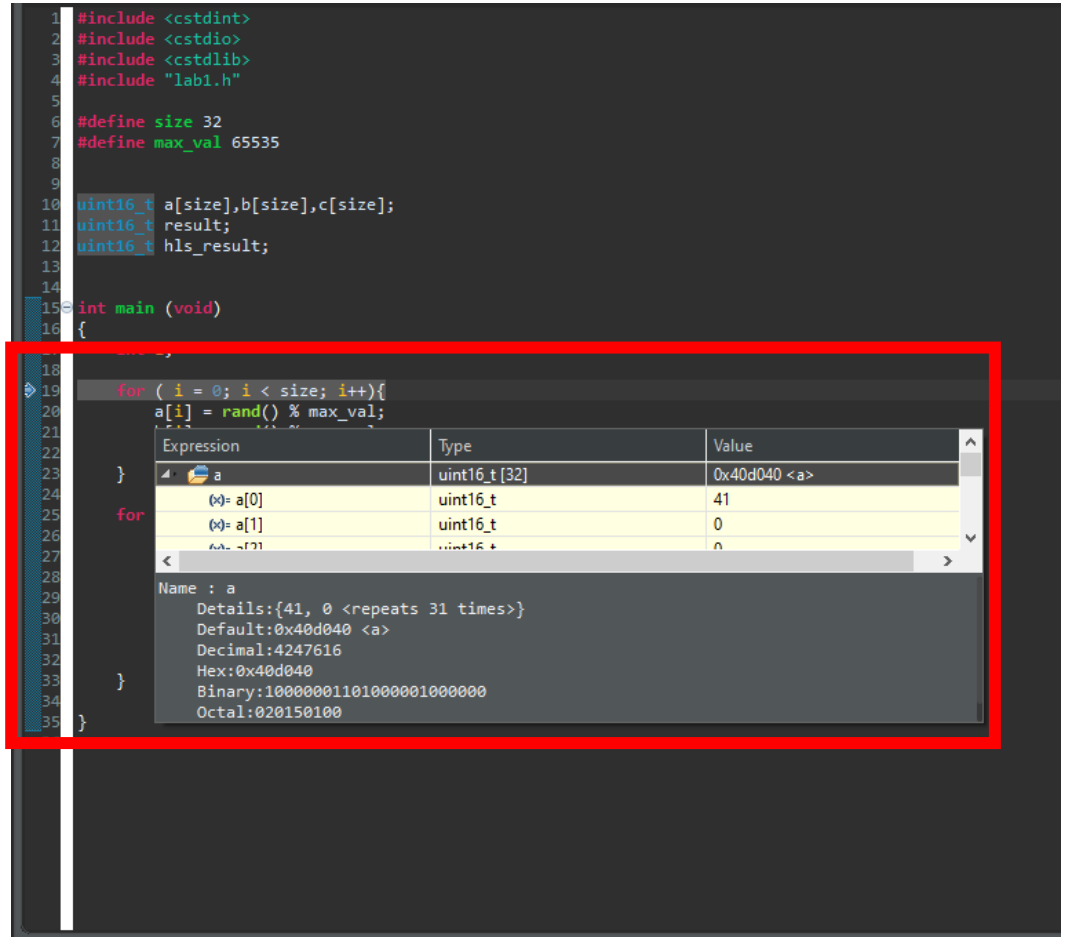
Lab 1: Vitis HLS Flow

Step 1 – This will open the debugger view, with the program paused for execution – we can run, pause, single step using the menu, along with insert breakpoints



Lab 1: Vitis HLS Flow

Step 1 – Single Step a few of the instructions, hold the pointer over a variable and notice the contents pop up for inspection



Lab 1: Vitis HLS Flow

Step 1 – Double click in the margin of line 29 to insert a breakpoint – this will be shown by a blue circle.

```
14
15 int main (void)
16 {
17     int i;
18
19     for ( i = 0; i < size; i++){
20         a[i] = rand() % max_val;
21         b[i] = rand() % max_val;
22         c[i] = rand() % max_val;
23     }
24
25     for (i = 0; i < size; i++){
26         result = a[i] * b[i] + c[i];
27         hls_result = lab1(a[i],b[i],c[i]);
28         if (result == hls_result){
29             std::printf("Pass\n\r");
30         }
31         else
32             std::printf("Fail\n\r");
33     }
34
35 }
36
```

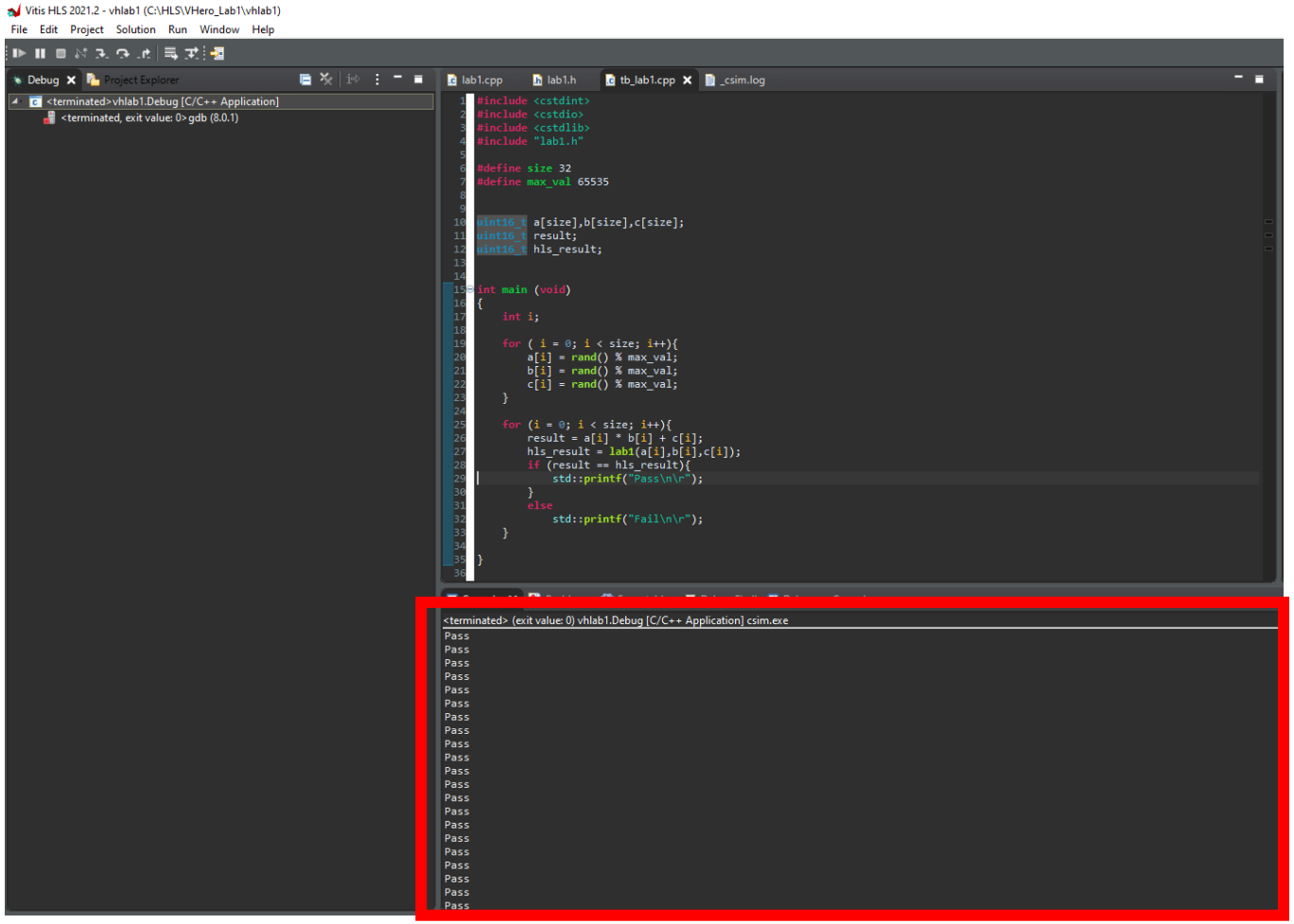
Lab 1: Vitis HLS Flow

Step 1 – Click Run and you will see the breakpoint be hit, double click again in the margin to remove the breakpoint and hit run.

```
14
15  int main (void)
16  {
17      int i;
18
19      for ( i = 0; i < size; i++){
20          a[i] = rand() % max_val;
21          b[i] = rand() % max_val;
22          c[i] = rand() % max_val;
23      }
24
25      for (i = 0; i < size; i++){
26          result = a[i] * b[i] + c[i];
27          hls_result = lab1(a[i],b[i],c[i]);
28          if (result == hls_result){
29              std::printf("Pass\n\r");
30          }
31          else
32              std::printf("Fail\n\r");
33      }
34
35  }
36
```

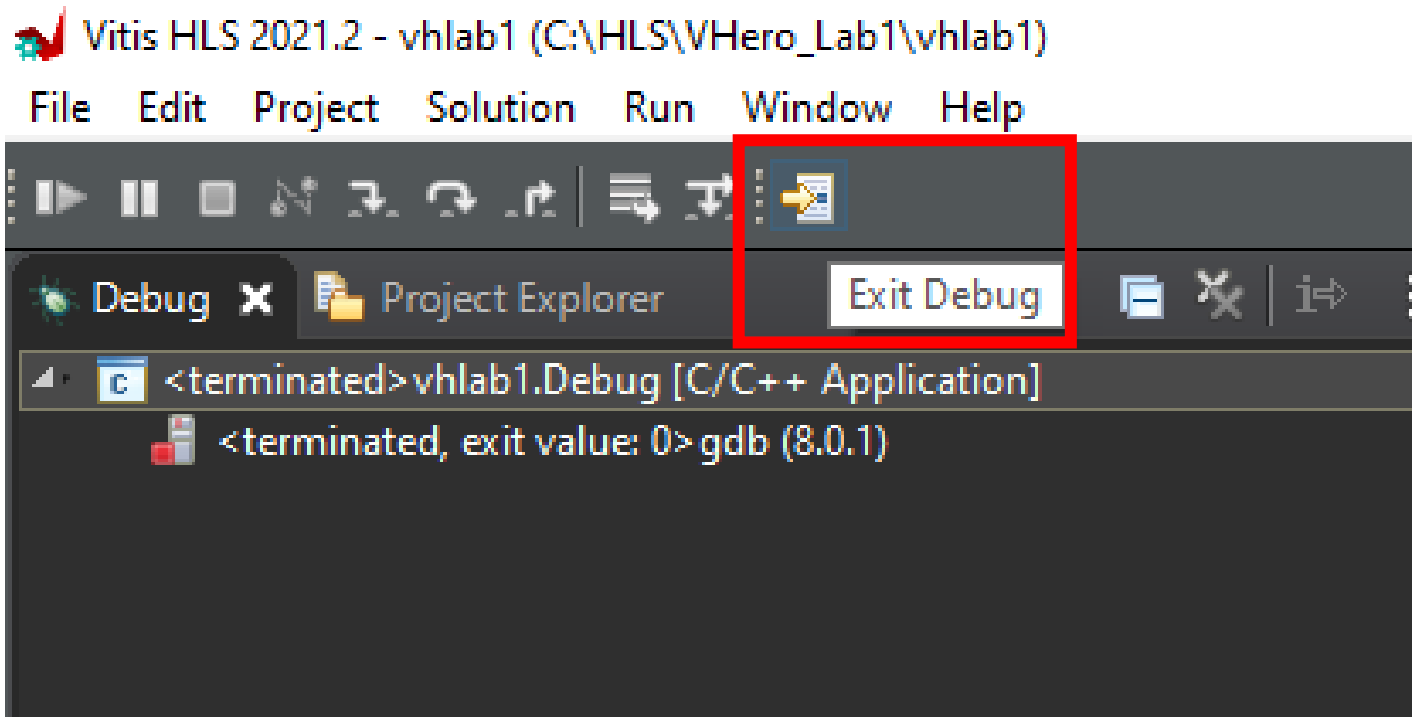
Lab 1: Vitis HLS Flow

Step 1 – The application should run through, outputting pass or fail in the console and then terminate its run.



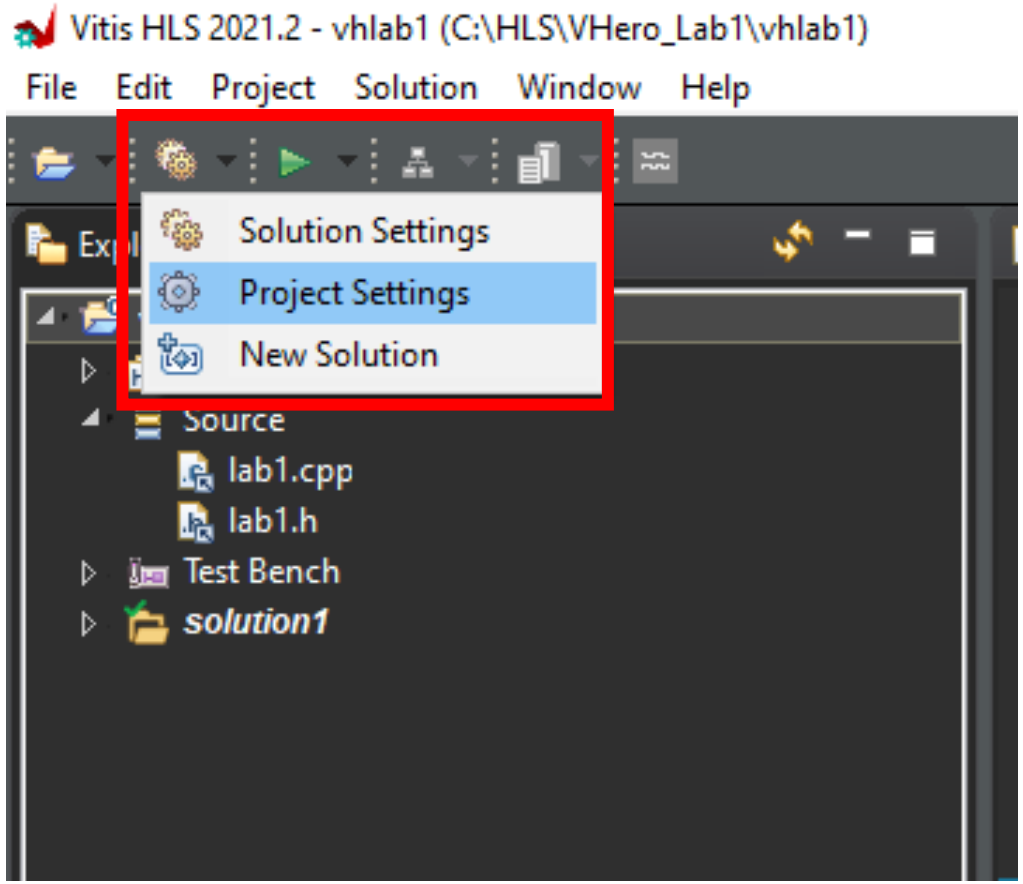
Lab 1: Vitis HLS Flow

Step 1 – Exit the debug display by clicking exit debug



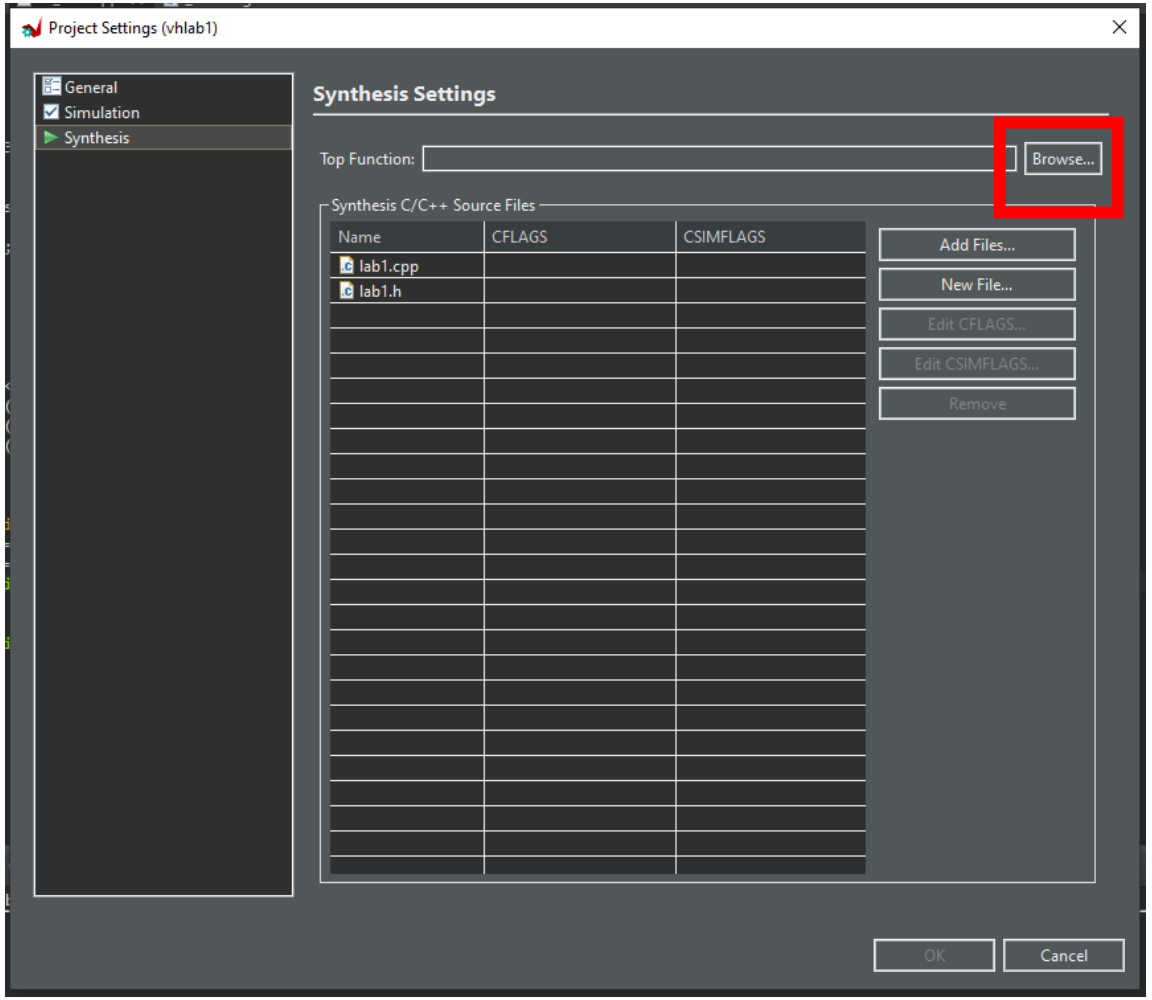
Lab 1: Vitis HLS Flow

Step 1 – From the setting menu select Project Settings



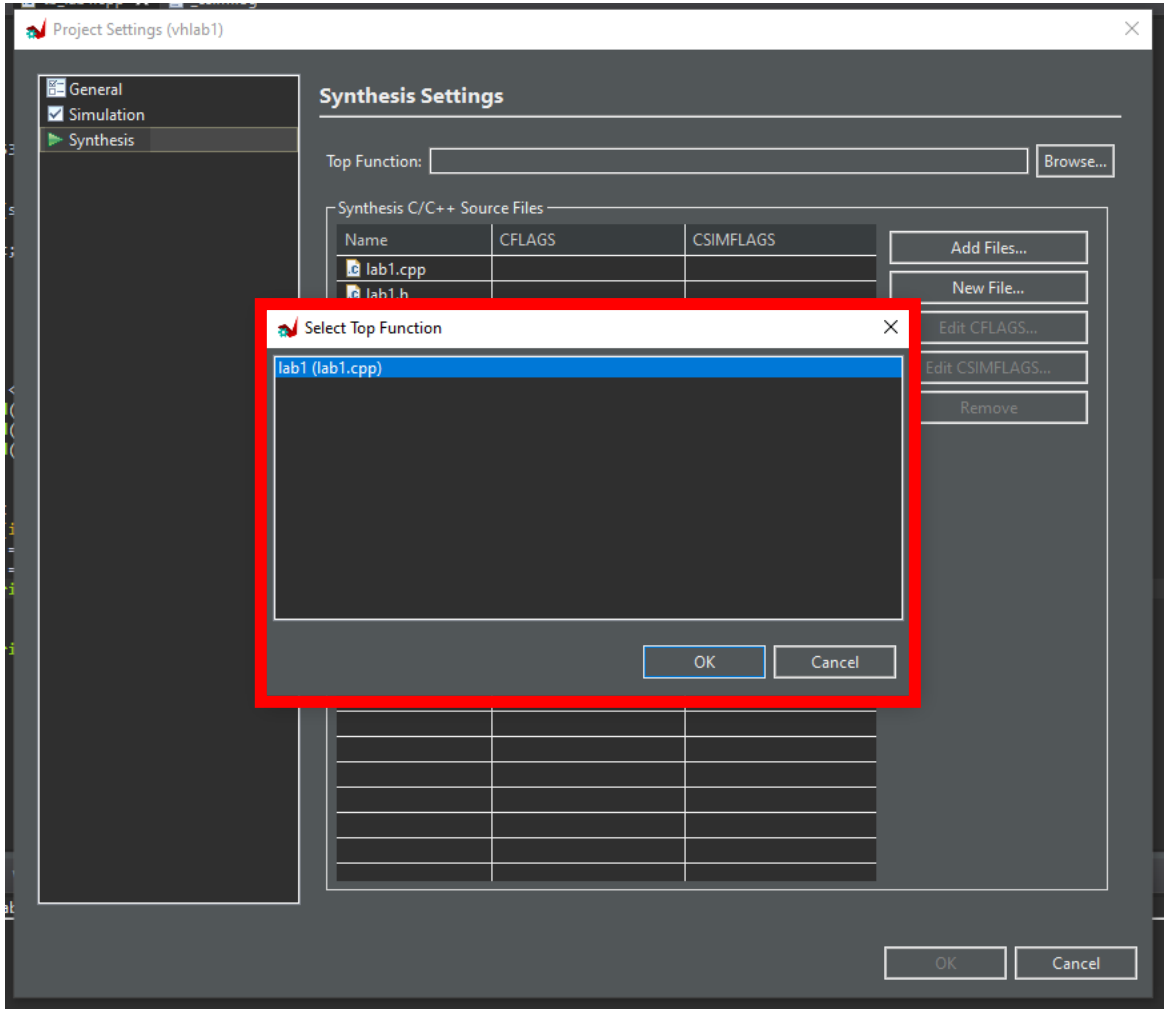
Lab 1: Vitis HLS Flow

Step 1 – Select Browse to select the top function



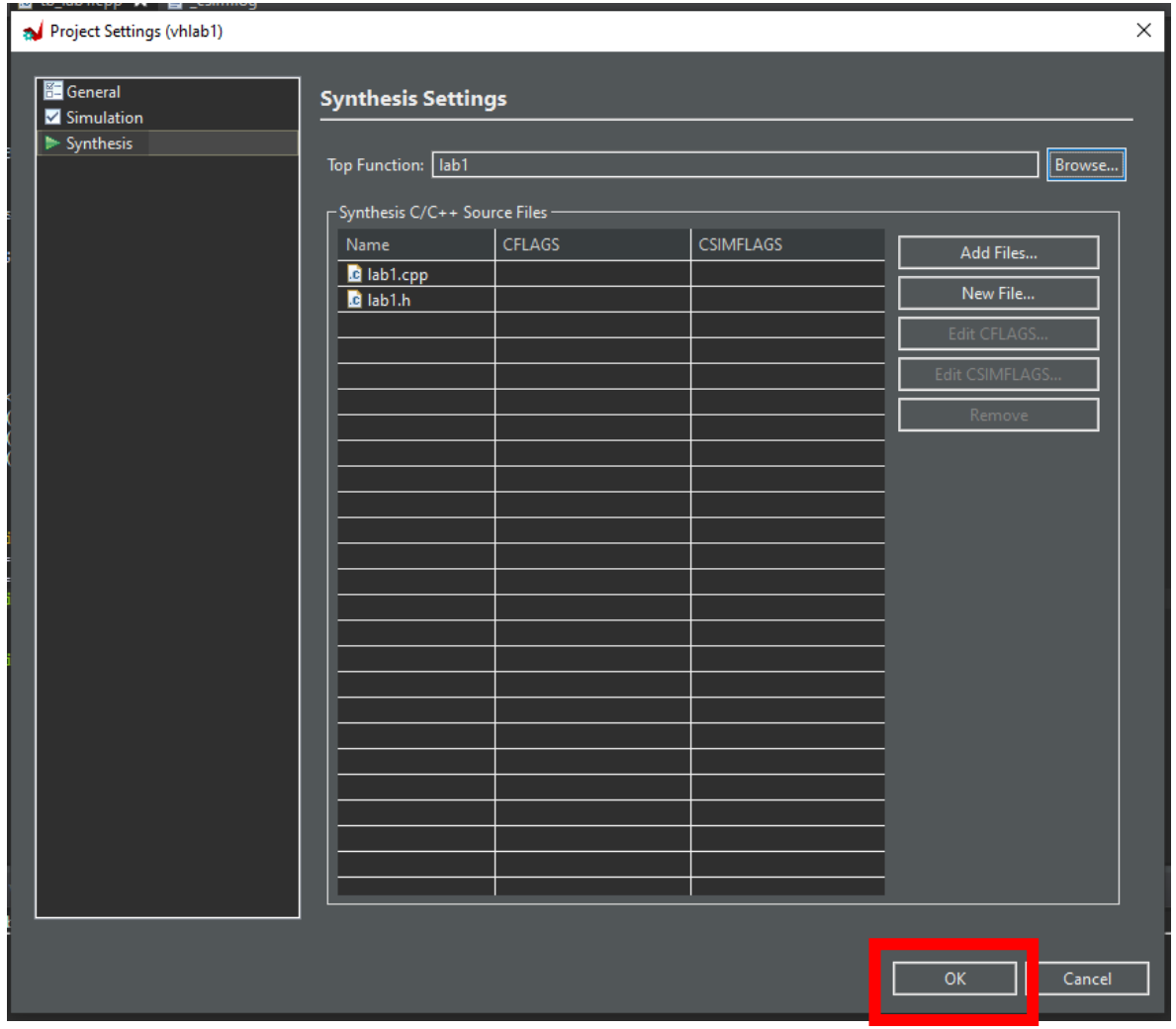
Lab 1: Vitis HLS Flow

Step 1 – Select lab1 (or the name of your function if you named it differently)



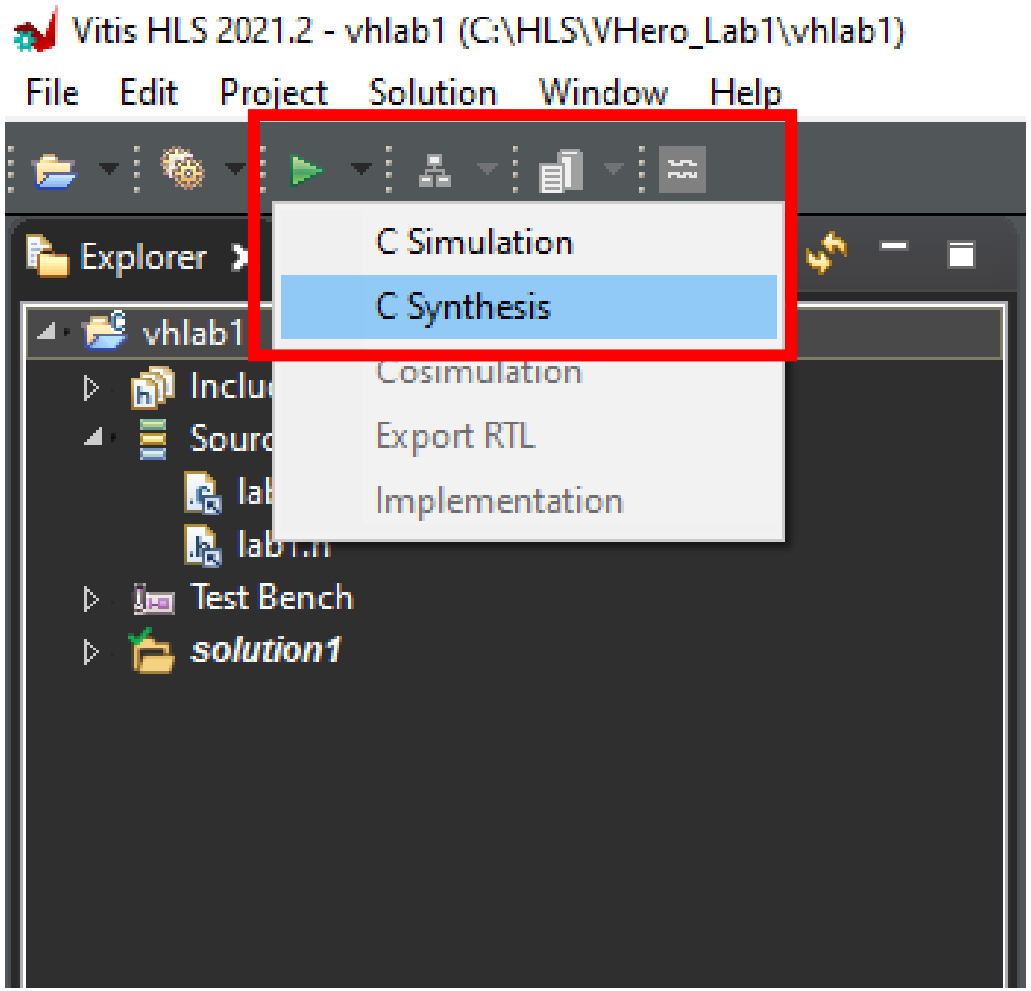
Lab 1: Vitis HLS Flow

Step 1 – Click OK



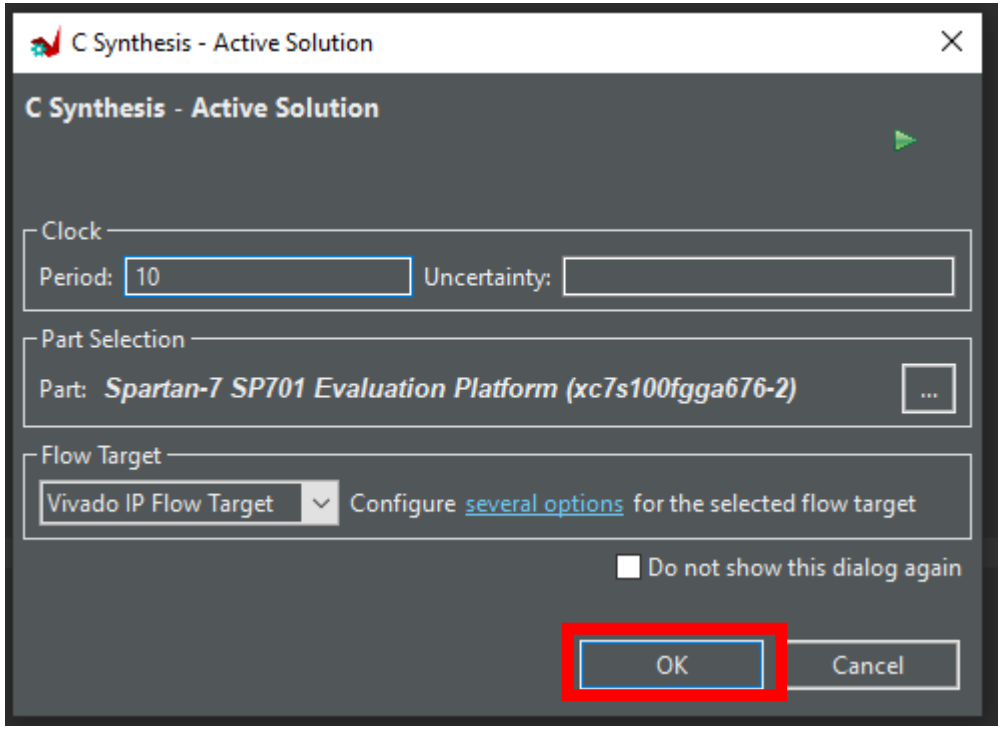
Lab 1: Vitis HLS Flow

Step 1 – Click on run and select C Synthesis



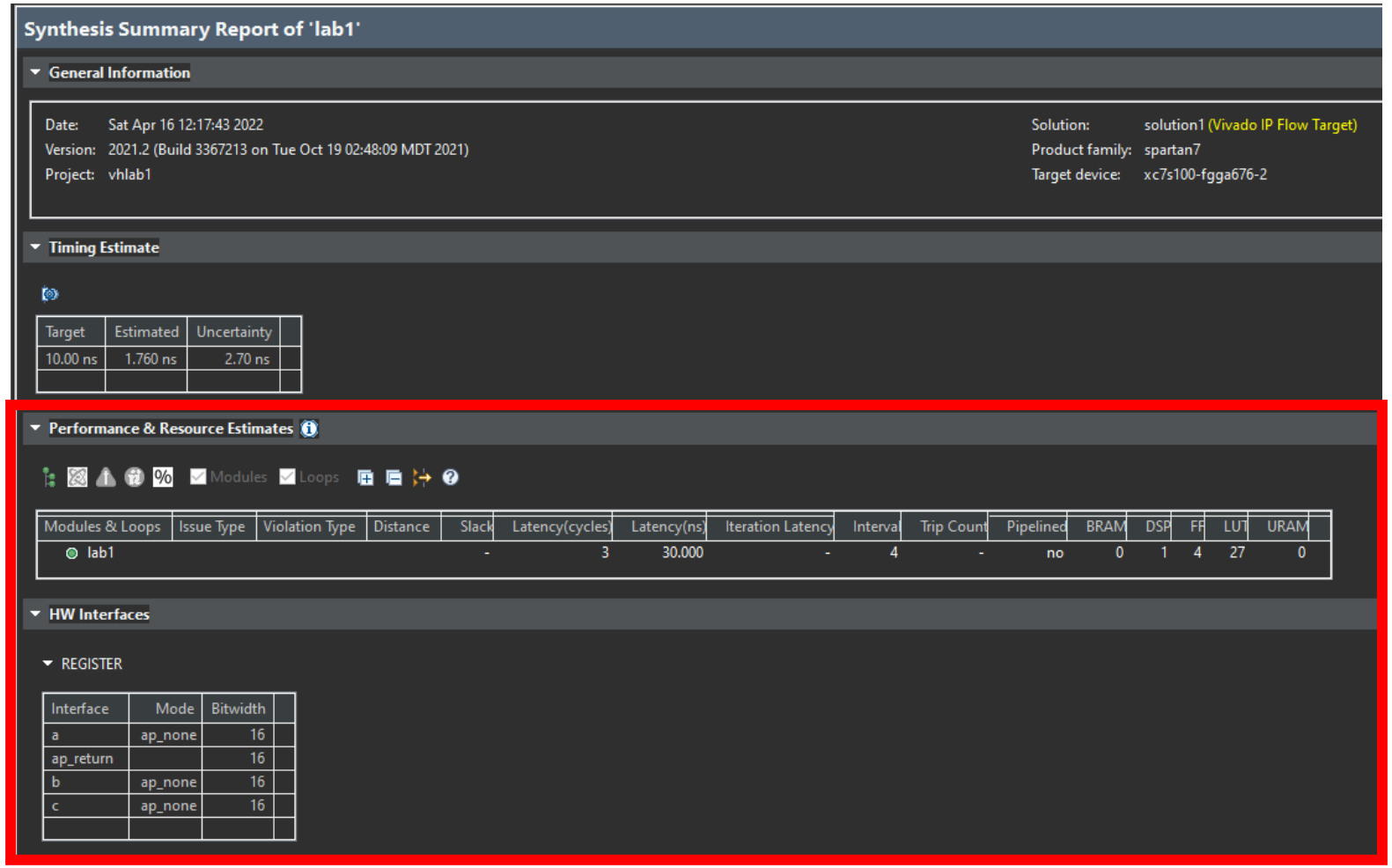
Lab 1: Vitis HLS Flow

Step 1 – Leave all options unchanged and select OK



Lab 1: Vitis HLS Flow

Step 1 – Once synthesis is complete you will see the report open notice the resources and interfaces



Synthesis Summary Report of 'lab1'

General Information

Date: Sat Apr 16 12:17:43 2022
Version: 2021.2 (Build 3367213 on Tue Oct 19 02:48:09 MDT 2021)
Project: vhlab1

Solution: solution1 (Vivado IP Flow Target)
Product family: spartan7
Target device: xc7s100-fgga676-2

Timing Estimate

Target	Estimated	Uncertainty
10.00 ns	1.760 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
lab1				-	3	30.000	-	4	-	no	0	1	4	27	0

HW Interfaces

REGISTER

Interface	Mode	Bitwidth
a	ap_none	16
ap_return		16
b	ap_none	16
c	ap_none	16

Lab 1: Vitis HLS Flow

Step 1 – Scroll down the report you will see the Top level Controls and Interface Info

TOP LEVEL CONTROL

Interface	Type	Ports
ap_clk	clock	ap_clk
ap_rst	reset	ap_rst
ap_ctrl	ap_ctrl_hs	ap_done ap_idle ap_ready ap_start

SW I/O Information

Top Function Arguments

Argument	Direction	Datatype
a	in	unsigned short
b	in	unsigned short
c	in	unsigned short
return	out	unsigned short

SW-to-HW Mapping

Argument	HW Interface	HW Type
a	a	port
b	b	port
c	c	port
return	ap_return	port

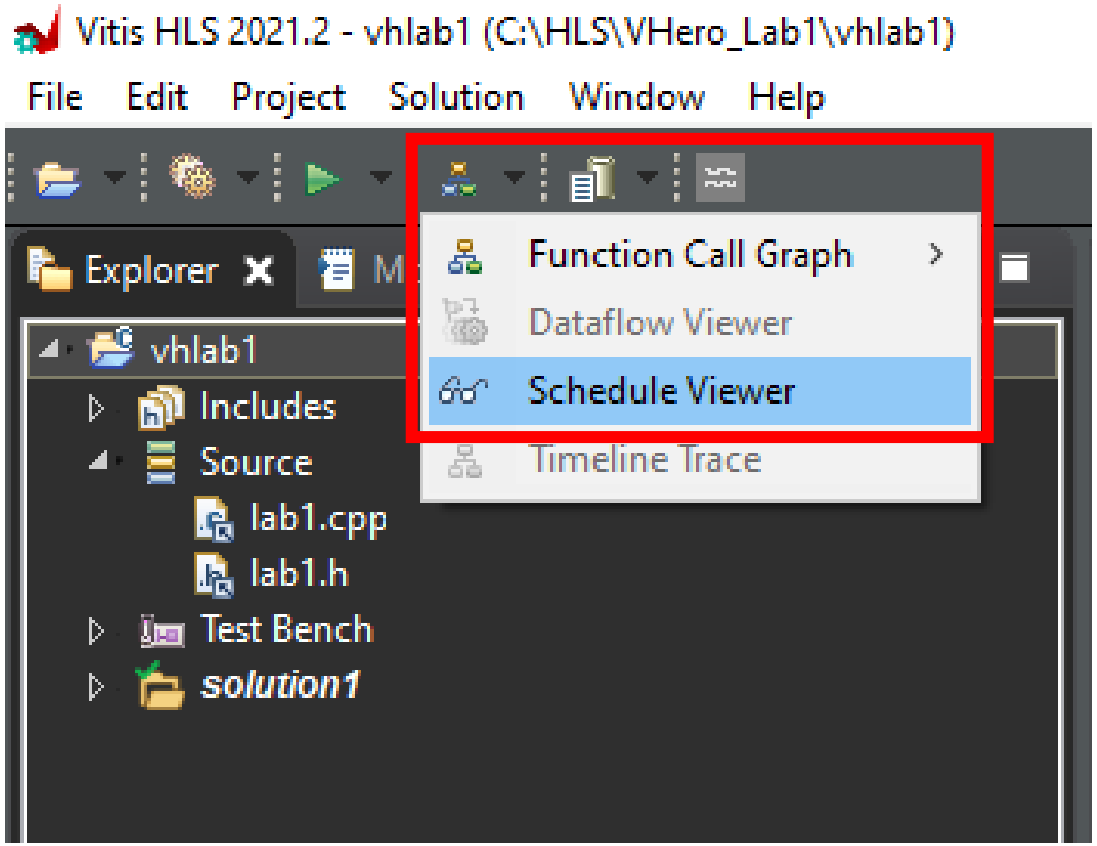
Bind Op Report

No filter settings

Name	DSP	Pragma	Variable	Op	Impl	Latency
lab1	1					
mac_muladd_16s_16s_16ns_16_4_1_U1	1			muladd	dsp	6
mul			mul_ln7	mul		3
add			result	add		3

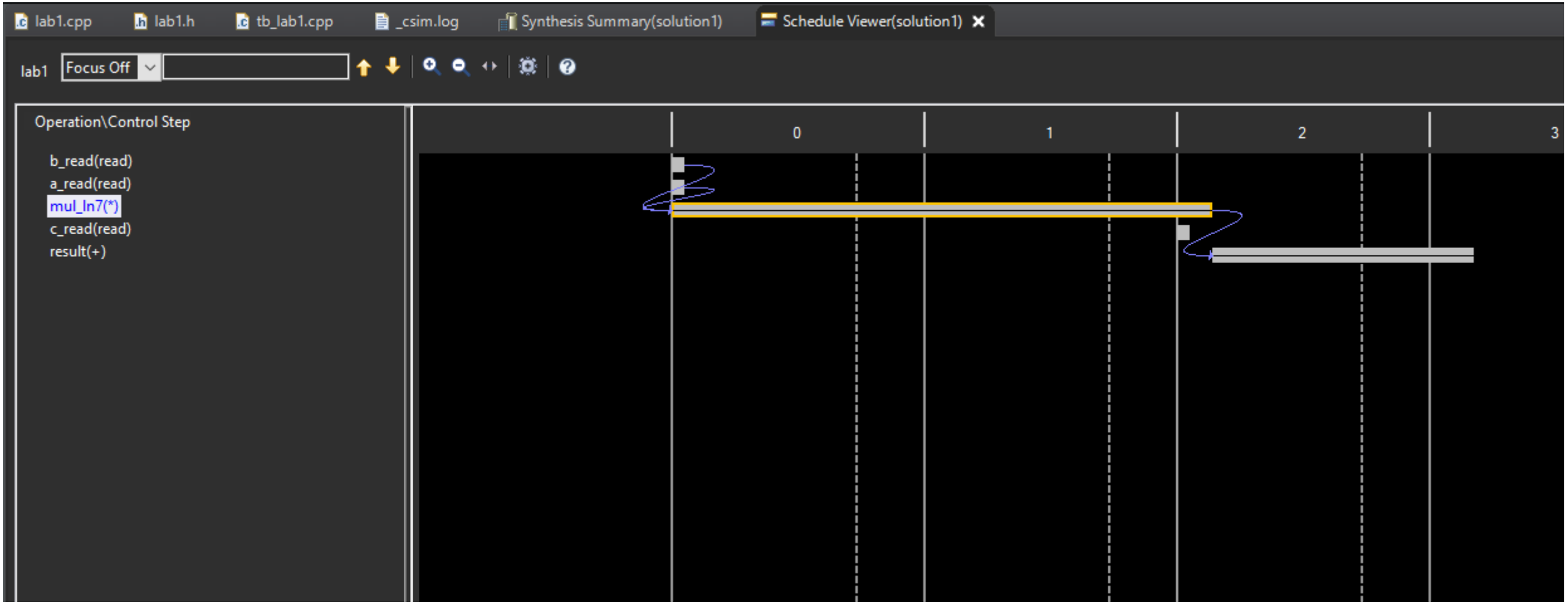
Lab 1: Vitis HLS Flow

Step 1 – From the Schedule menu select Schedule View



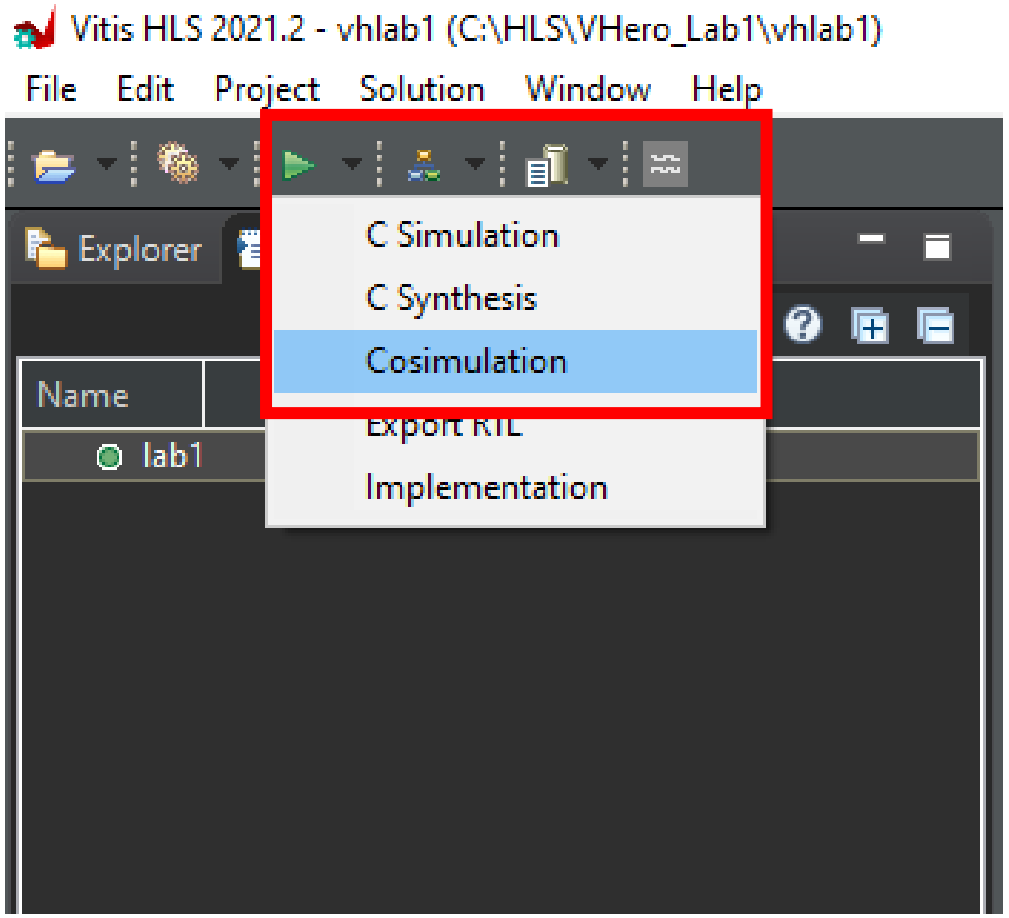
Lab 1: Vitis HLS Flow

Step 1 – Examine the schedule view



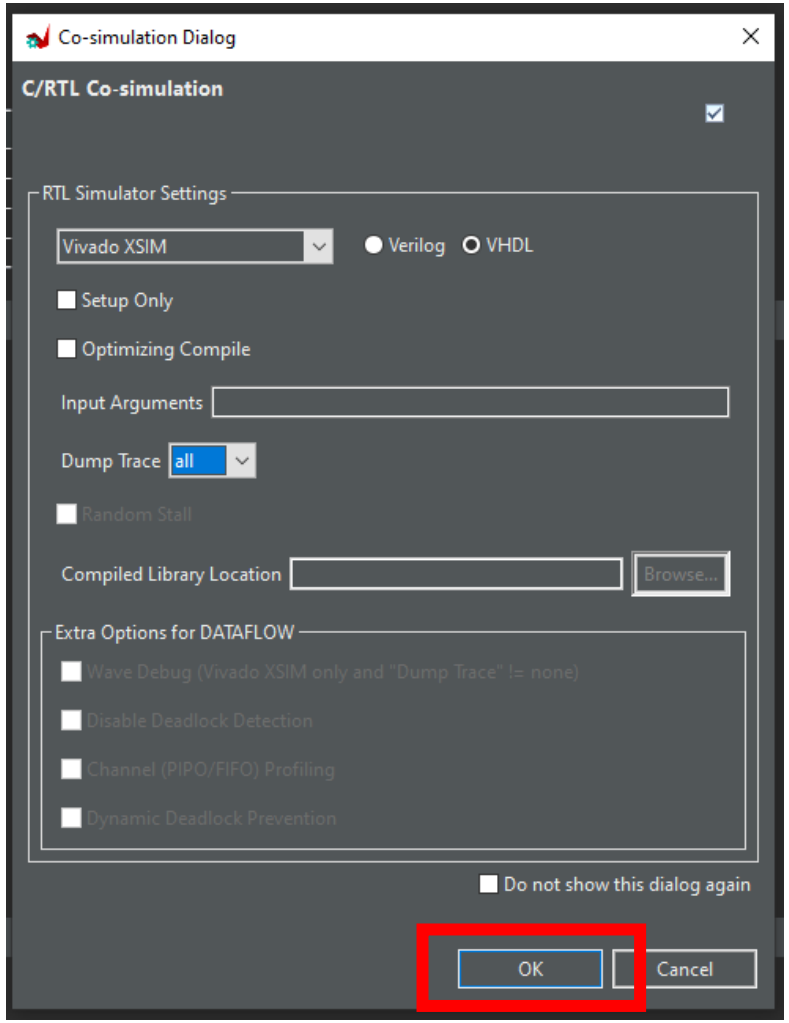
Lab 1: Vitis HLS Flow

Step 1 – From the run menu select Co-Simulation



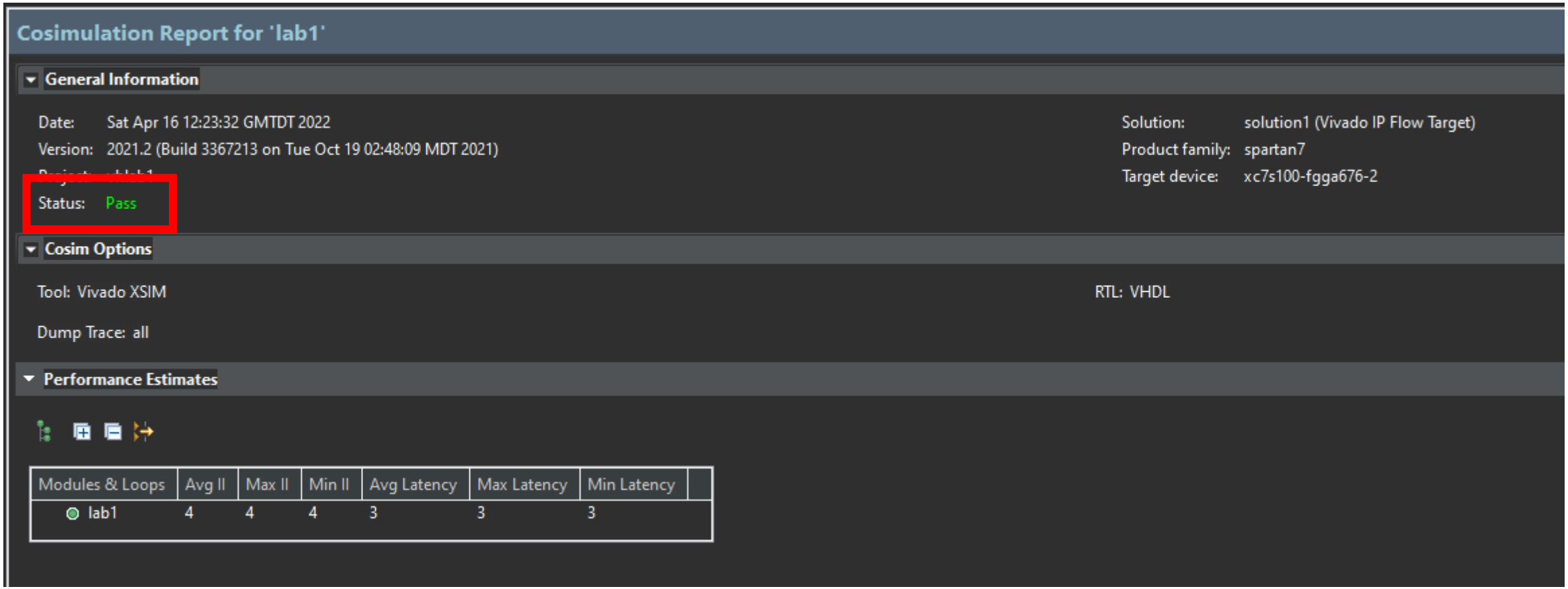
Lab 1: Vitis HLS Flow

Step 1 – Leave the options unchanged and click OK



Lab 1: Vitis HLS Flow

Step 1 – Observe the report which shows the completion of the Co Simulation and pass / fail.



Lab 1 Summary

The concludes lab 1, throughout this lab we have demonstrated

1. How to create a project in Vitis HLS
2. How to create source and test benches
3. How to run C simulation
4. How to set up the project for synthesis
5. How to synthesize the design
6. How to run Co-Simulaiton

Lab 2

Vitis HLS Interfacing

Interfacing

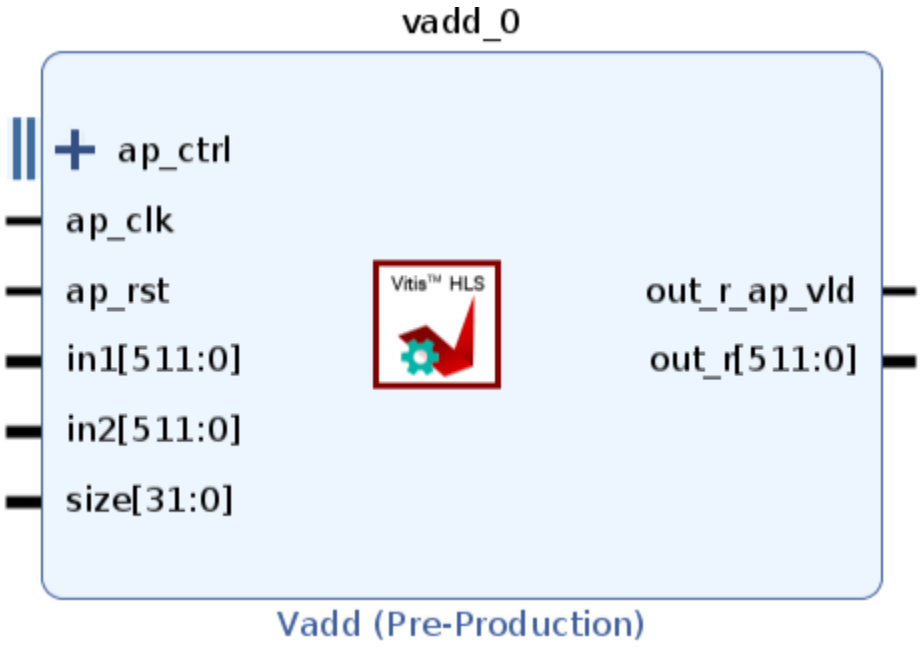
Interfacing depends on flow

Vitis - XO

- AXI – Data Accessed via Memory Map
- AXIS – Data Accessed via Stream
- AXI Lite – Register Access

Vivado – IP XACT

- Block Level – Controls Flow / Status
- Protocol Level – Controls



Vivado Interfacing

Instantiate appropriate interface to integrate with design

Do we want to start and stop block, synchronize or be data driven.

Are we controlling the block from SW

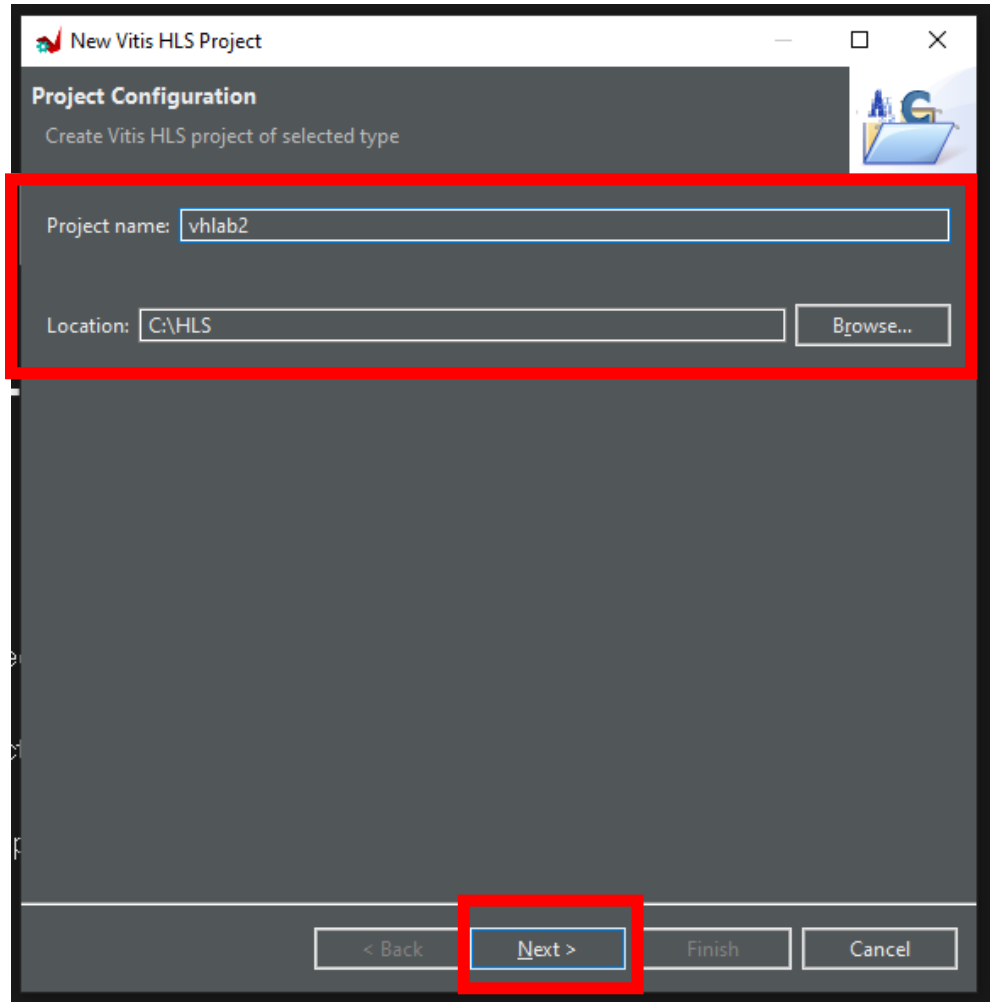
What about data interfaces.

Paradigm	Description	s
Memory	Data is accessed by the kernel through memory such as DDR, HBM, PLRAM/BRAM/ URAMSupported Interface Protocol	`ap_memory`, BRAM, AXI4 Memory Mapped (`m_axi`)
Stream	Supported InterfaceData is streamed into the kernel from another streaming source, such as video processor or another kernel, and can also be streamed out of the kernel.	`ap_fifo`, AXI4-Stream (`axis`)
Register	Data is accessed by the kernel through register interfaces performed by register reads and writes.	`ap_none`, `ap_hs`, `ap_ack`, `ap_ovld`, `ap_vld`, and AXI4-Lite adapter (`s_axilite`).

C-Argument Type	Supported Paradigms	Default Paradigm	Default Interface Protocol		
			Input	Output	Inout
Scalar variable (pass by value)	Register	Register	ap_none	N/A	N/A
Array	Memory, Stream	Memory	ap_memory	ap_memory	ap_memory
Pointer	Memory, Stream, Register	Register	ap_none	ap_vld	ap_ovld
Reference	Register	Register	ap_none	ap_vld	ap_vld
`hls::stream`	Stream	Stream	ap_fifo	ap_fifo	N/A

Lab 2: Vitis HLS Interfacing

Step 1 – Create a new project called VHLab2, click OK.



Lab 2: Vitis HLS Interfacing

Step 1 – Create a new source file called lab2.cpp and enter the code below or copy from the GitHub.

lab2.cpp | lab2.h | tb_lab2.cpp | _csim.log | Synthesis Summary(solution1)

```cpp
#include "stdint.h"
#include "lab2.h"

void lab2 (uint16_t loop, uint16_t a[32], uint16_t b[32], uint16_t c[32], uint16_t result[32])
{
    uint16_t i;
    lab2_label0:for (i = 0; i < loop; i++){
        result[i] = a[i] * b[i] + c[i];
    }
}
```

Lab 2: Vitis HLS Interfacing

Step 1 – Create a new source file called lab2.h and enter the code below or copy from the GitHub.

lab2.cpp | lab2.h | tb_lab2.cpp | _csim.log | Synthesis Summary(solution1)

```
#define size 32
void lab2 (uint16_t loop, uint16_t a[32], uint16_t b[32], uint16_t c[32], uint16_t result[32]);
```

Lab 2: Vitis HLS Interfacing

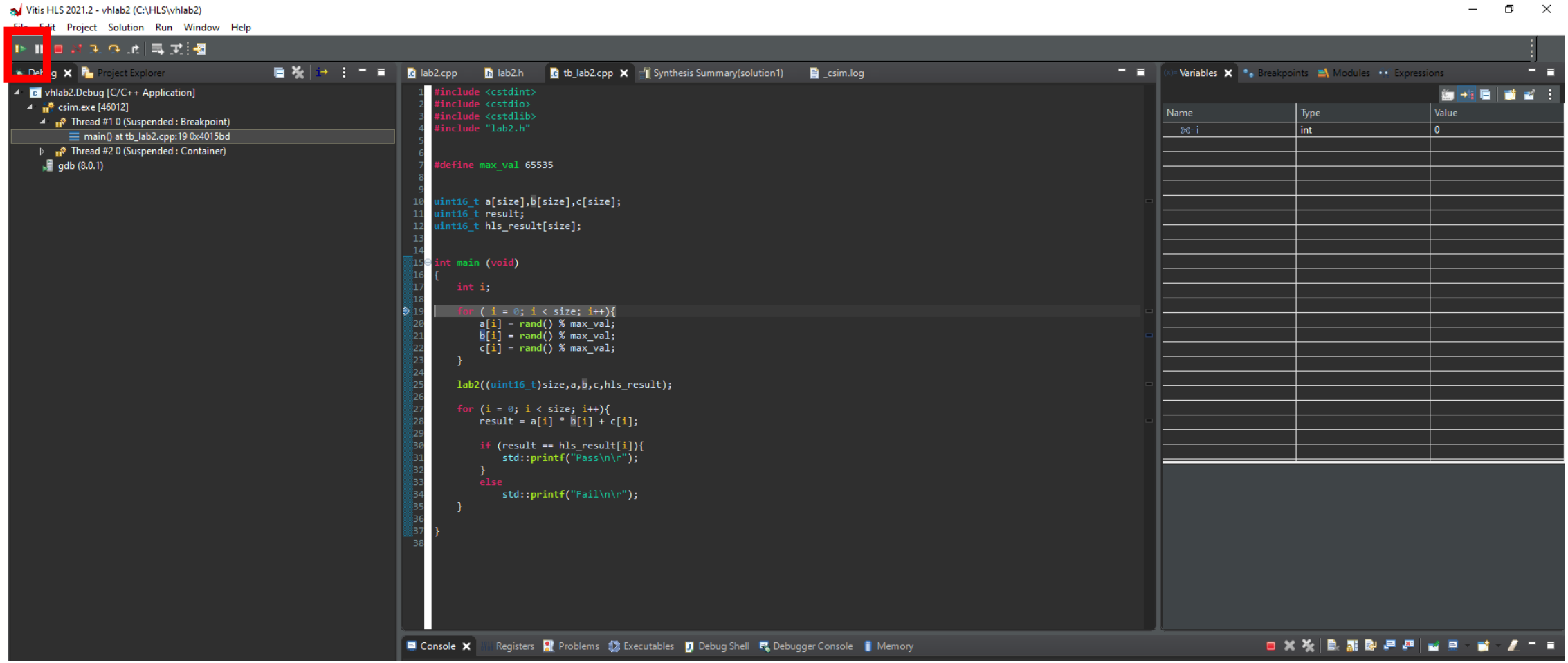
Step 1 – Create a new source file called tb_lab2.cpp and enter the code below or copy from the GitHub.

lab2.cpp | lab2.h | tb_lab2.cpp | _csim.log | Synthesis Summary(solution1)

```cpp
#include <cstdint>
#include <cstdio>
#include <cstdlib>
#include "lab2.h"


#define max_val 65535


uint16_t a[size],b[size],c[size];
uint16_t result;
uint16_t hls_result[size];


int main (void)
{
    int i;

    for ( i = 0; i < size; i++){
        a[i] = rand() % max_val;
        b[i] = rand() % max_val;
        c[i] = rand() % max_val;
    }

    lab2((uint16_t)size,a,b,c,hls_result);

    for (i = 0; i < size; i++){
        result = a[i] * b[i] + c[i];

        if (result == hls_result[i]){
            std::printf("Pass\n\r");
        }
        else
            std::printf("Fail\n\r");
    }

}
```

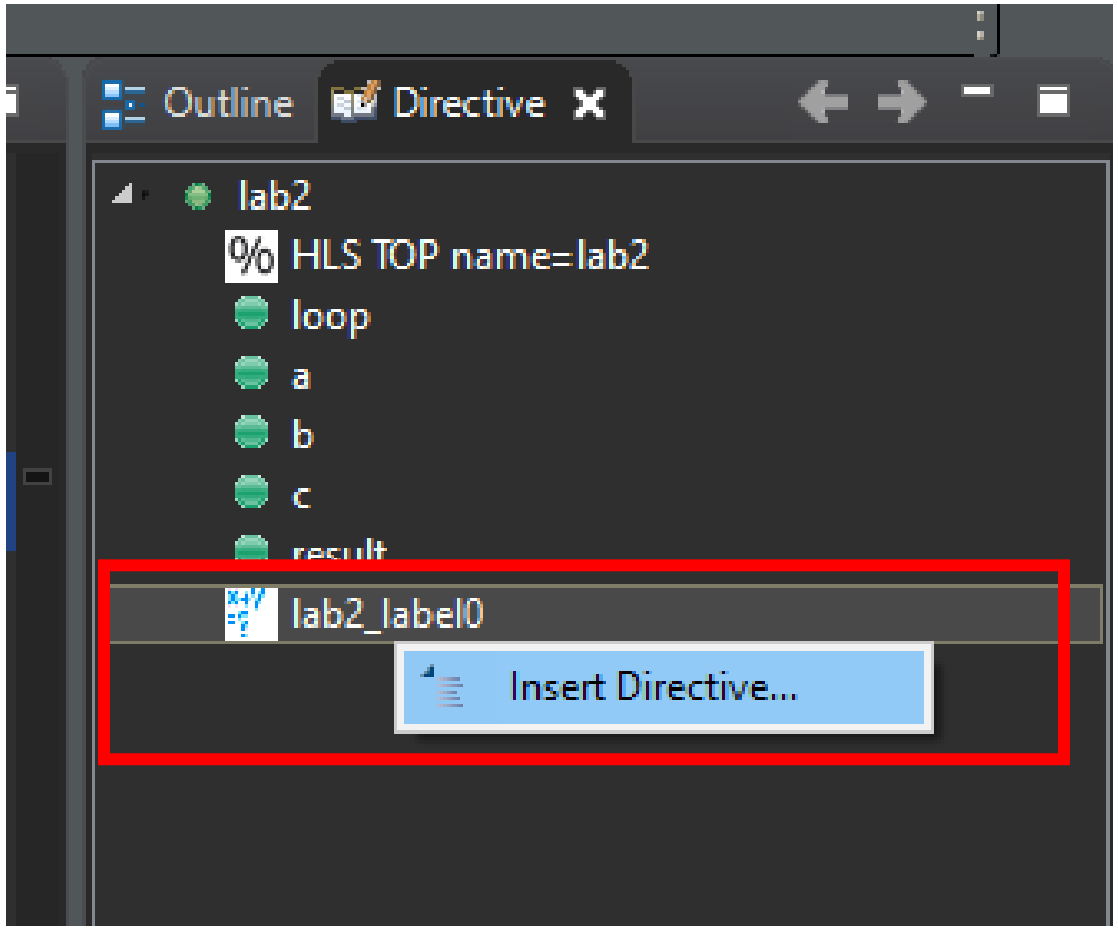
Lab 2: Vitis HLS Interfacing

Step 1 – Run a C Simulation in the debugger, notice how this time the application is batch based



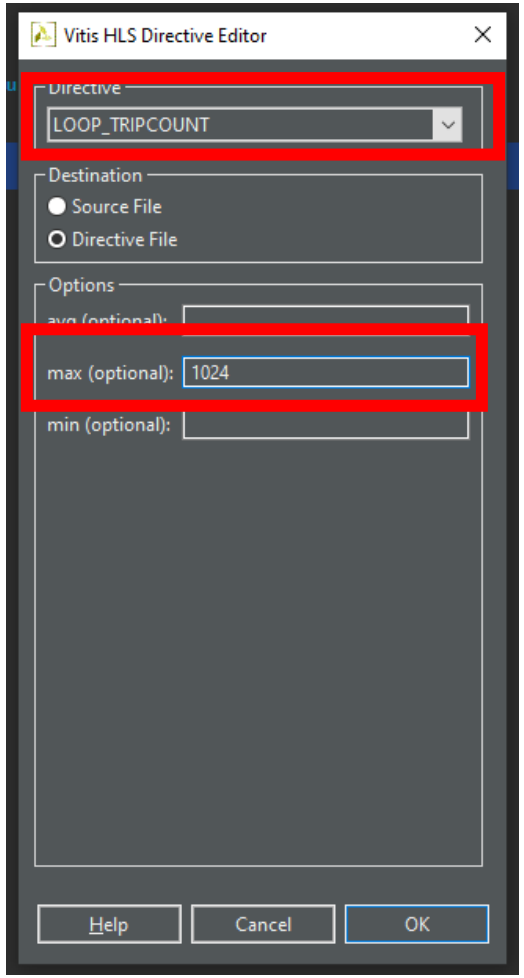
Lab 2: Vitis HLS Interfacing

Step 1 – Close the C Simulation and open the Lab2.cpp file, select the directives' view. Right click on the lb2_label0 and select Insert Directive.



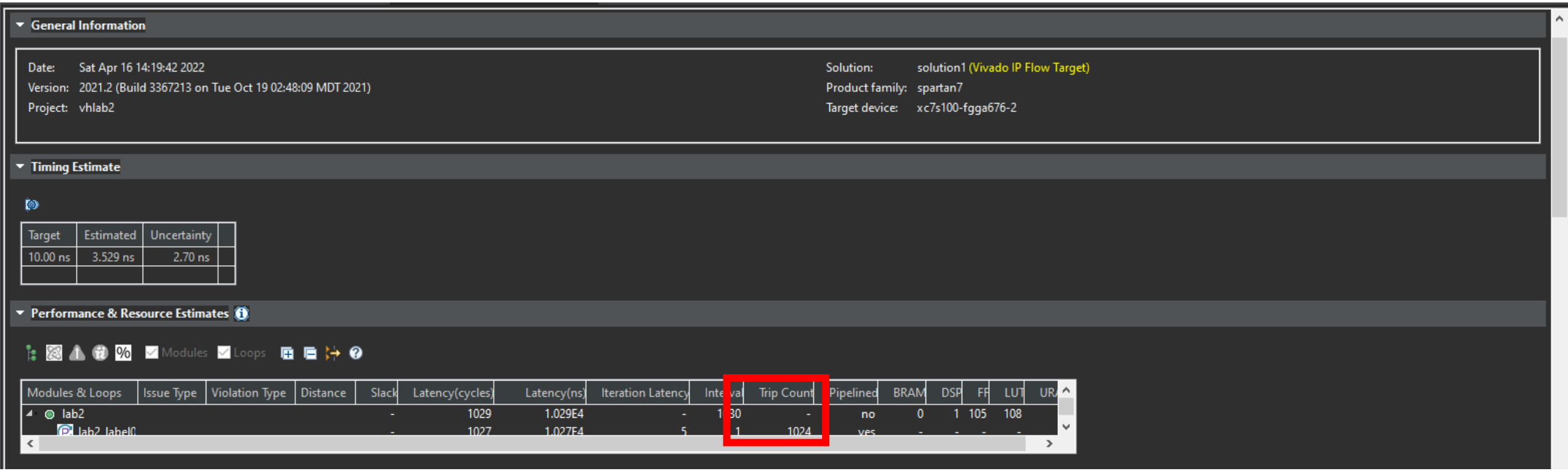
Lab 2: Vitis HLS Interfacing

Step 1 – Insert the directive LOOP_TRIPCOUNT this informs the tool how many times the loop will execute in the maximum case.



Lab 2: Vitis HLS Interfacing

Step 1 – Run the Synthesis you will notice the Trip Count is now recorded under the lab2 label0



Lab 2: Vitis HLS Interfacing

Step 1 – Examine the HW Interfaces and notice the RAM Interfaces are AP_MEMORY and Register Interfaces.

HW Interfaces

AP_MEMORY

Interface	Bitwidth
a_address0	5
a_q0	16
b_address0	5
b_q0	16
c_address0	5
c_q0	16
result_address0	5
result_d0	16

REGISTER

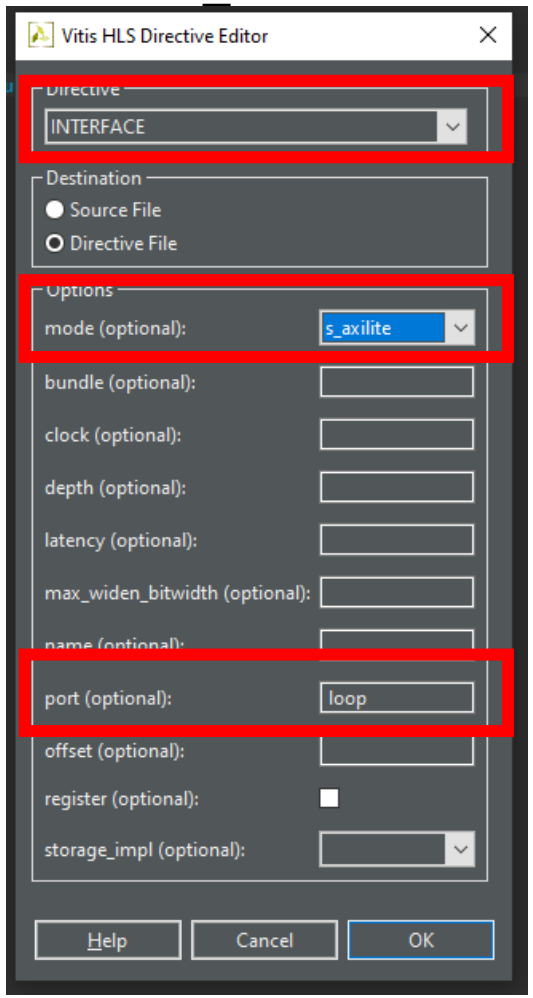
Interface	Mode	Bitwidth
loop_r	ap_none	16

TOP LEVEL CONTROL

Interface	Type	Ports
ap_clk	clock	ap_clk
ap_rst	reset	ap_rst
ap_ctrl	ap_ctrl_hs	ap_done ap_idle ap_ready ap_start

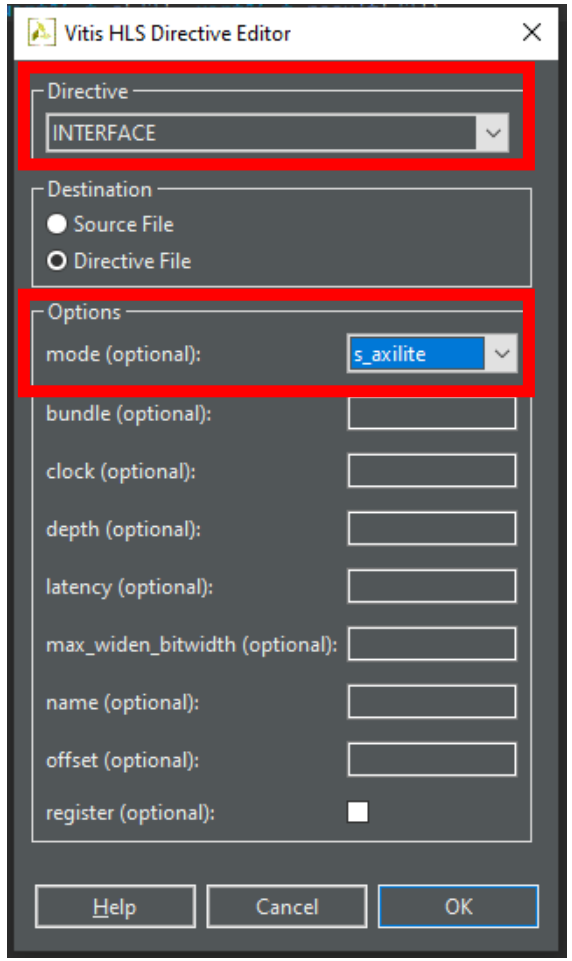
Lab 2: Vitis HLS Interfacing

Step 1 – In the synthesis view add in a new directive for the source code for the port loop. Select INTERFACE directive and for the mode select S_AXILITE



Lab 2: Vitis HLS Interfacing

Step 1 – In the synthesis view add in a new directive for the source code for the function. Select INTERFACE directive and for the mode select S_AXILITE



Lab 2: Vitis HLS Interfacing

Step 1 – The Directives should appear as shown in the directives tab – Rerun synthesis

lab2.cpp | lab2.h | tb_lab2.cpp | _csim.log | Synthesis Summary(solution1)

```
#include "stdint.h"
#include "lab2.h"

void lab2 (uint16_t loop, uint16_t a[32], uint16_t b[32], uint16_t c[32], uint16_t result[32])
{
    uint16_t i;
    lab2_label0:for (i = 0; i < loop; i++){
        result[i] = a[i] * b[i] + c[i];
    }
}
```

Outline | Directive

- lab2
  - HLS INTERFACE mode=s_axilite port=return
  - HLS TOP name=lab2
  - loop
  - HLS INTERFACE mode=s_axilite port=loop
  - a
  - b
  - c
  - result
  - lab2_label0
    - HLS LOOP_TRIPCOUNT max=1024

Lab 2: Vitis HLS Interfacing

Step 1 – Examine the HW Interfaces in the report and notice control of the block and loop register are now AXI registers

HW Interfaces

S_AXILITE Interfaces

Interface	Data Width	Address Width	Offset	Register
s_axi_control	32	5	16	0

S_AXILITE Registers

Interface	Register	Offset	Width	Access	Description
s_axi_control	loop_r	0x10	32	W	Data signal of loop_r

AP_MEMORY

Interface	Bitwidth
a_address0	5
a_q0	16
b_address0	5
b_q0	16
c_address0	5
c_q0	16
result_address0	5
result_d0	16

Lab 2 Summary

The concludes lab 2, throughout this lab we have demonstrated

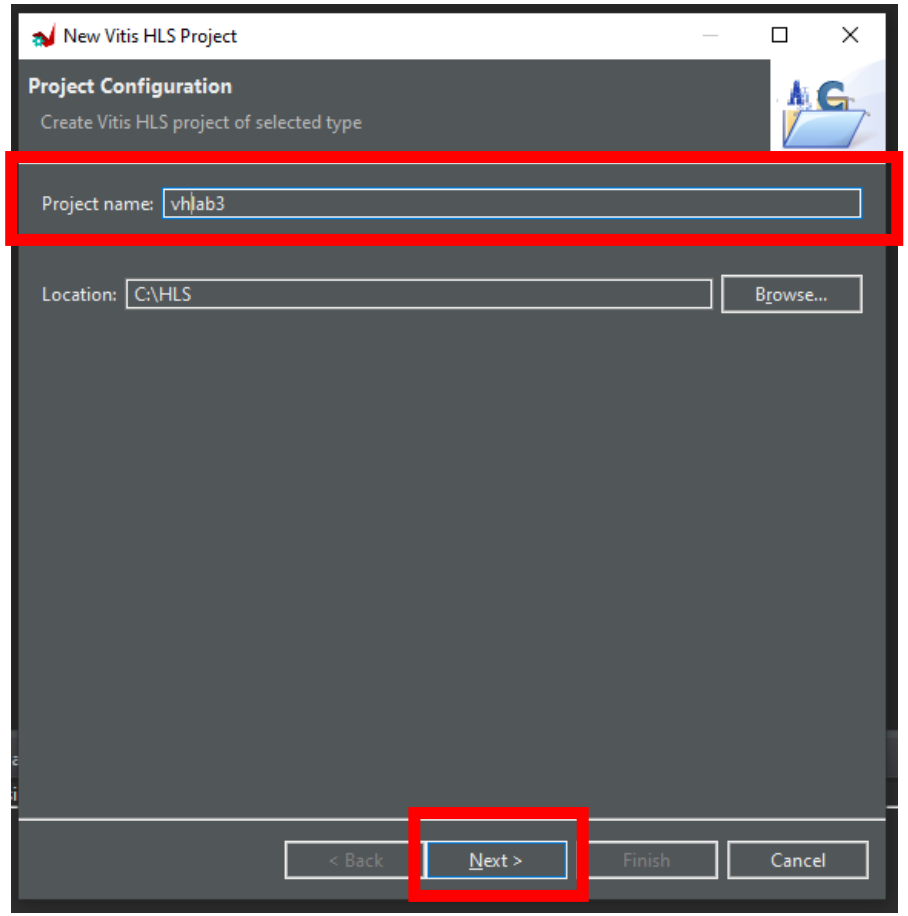
1. How to inform the synthesis tool about number of loop iterations
2. How to control register interfaces to use AXI Interfaces
3. How to control the IP Block control signals be Implemented as AXI

Lab 3

Vitis HLS Optimization

Lab 3: Vitis HLS Optimization

Step 1 – Create a new project called VHlab3.cpp



Lab 3: Vitis HLS Optimization

Step 1 – Create a new source file called lab3.cpp and enter the code below or copy from the GitHub.

Synthesis Summary(solution1) | lab3.cpp | lab3.h

```
#include "lab3.h"

void sensor_filter( ip_type input[16], op_type *output)
{
    int32_t accumulator=0;
    accum: for(int i = 0; i<16; i++)
    {
        accumulator += (int32_t ) input[i];
    }
    *output = accumulator / 16;
}

```

Lab 3: Vitis HLS Optimization

Step 1 – Create a new source file called lab3.h and enter the code below or copy from the GitHub.

```
#include "stdint.h"
#include "stdio.h"
#include "stdlib.h"
#include <string.h>

typedef int16_t ip_type;
typedef int16_t op_type;

void sensor_filter( ip_type input[16], op_type *output);
```

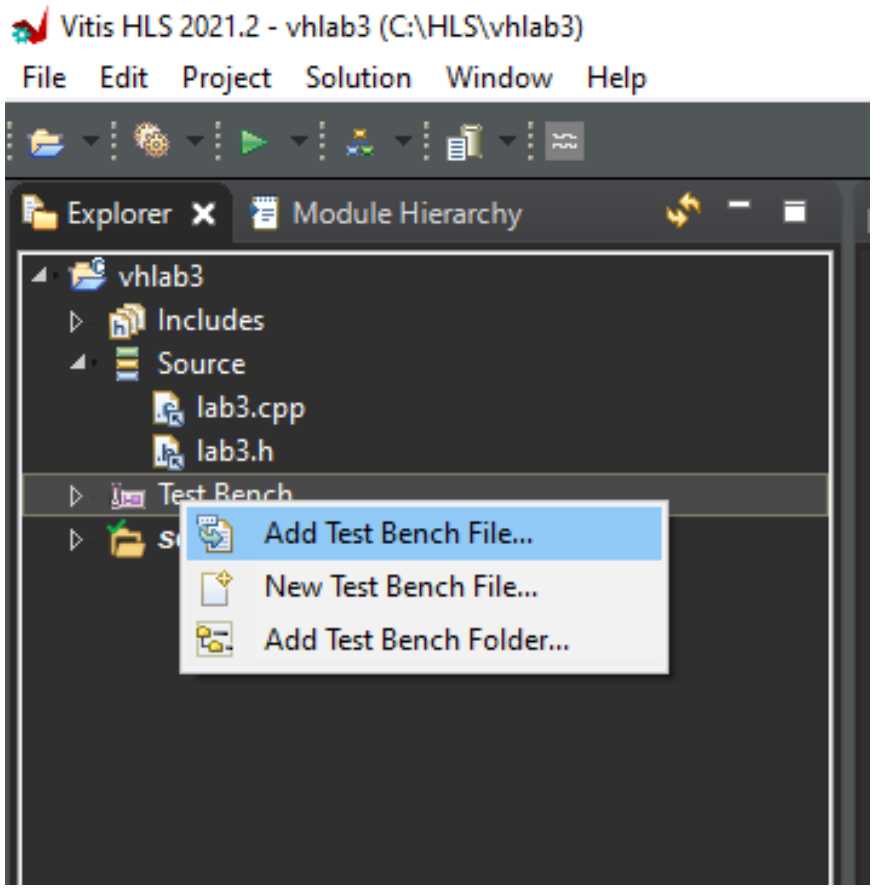
Lab 3: Vitis HLS Optimization

Step 1 – Create a new source file called tb_lab3.cpp and enter the code below or copy from the GitHub.

```
#include "lab3.h"

#define MAX_LINE_LENGTH 80

int main()
{
    FILE *fp;
    char line[80];
    ip_type input[16];
    op_type output;

    printf("HLS Example\n\r");
    // Open a file for the output results
    fp=fopen("input.dat","r");
    if (fp == NULL)
        printf("error opening file\n\r");
    int i = 0;
    while (fgets(line, MAX_LINE_LENGTH, fp) != NULL) {
           input[i] = atoi(line);                        //convert string to integer format
           i++;
    }
    fclose(fp);
    //run the sensor
    sensor_filter(input, &output);
    printf("Result from HLS module = %d \n\r",output);
}
```

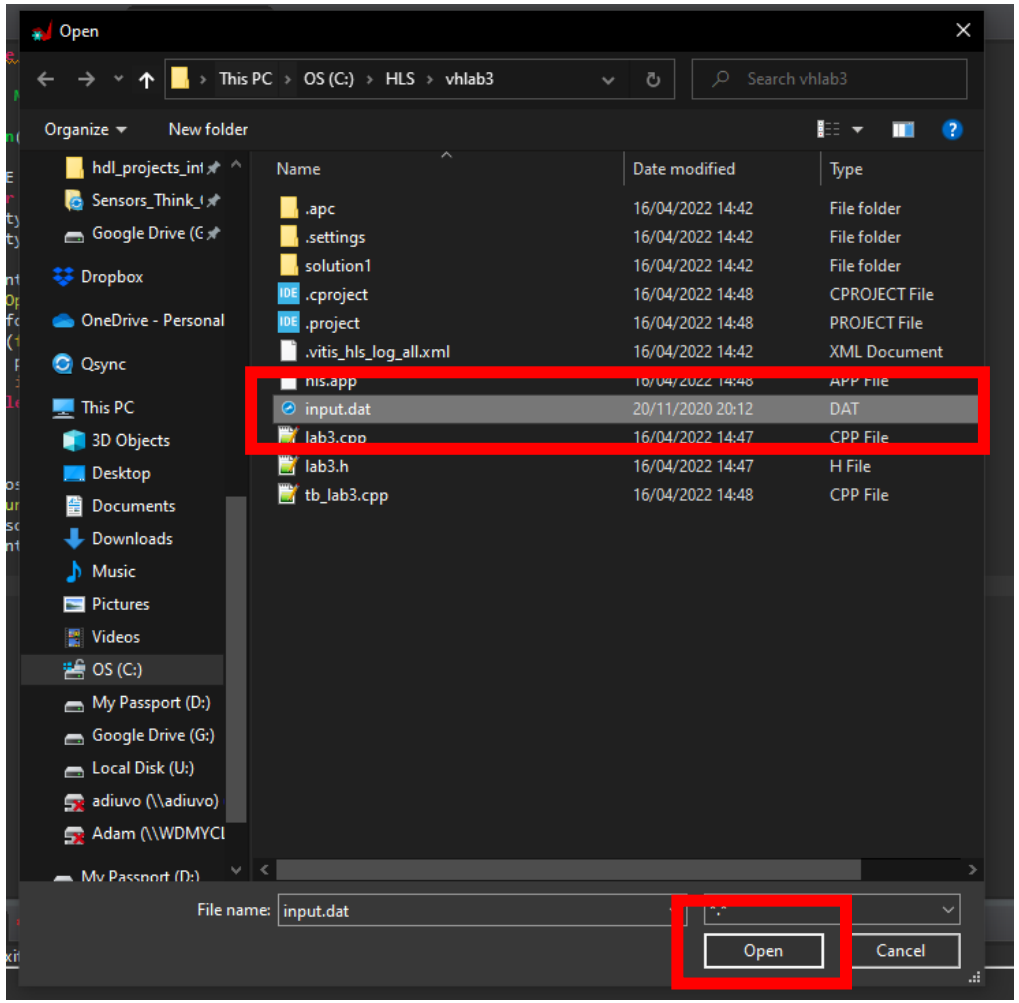
Lab 3: Vitis HLS Optimization

Step 1 – Right Click on Test Bench File and select add Test Bench File



Lab 3: Vitis HLS Optimization

Step 1 – Add in the input.dat file provided on the GitHub



Lab 3: Vitis HLS Optimization

Step 1 – Run the C Simulation and ensure the result is as shown below.

lab3.cpp | lab3.h | tb_lab3.cpp | _csim.log

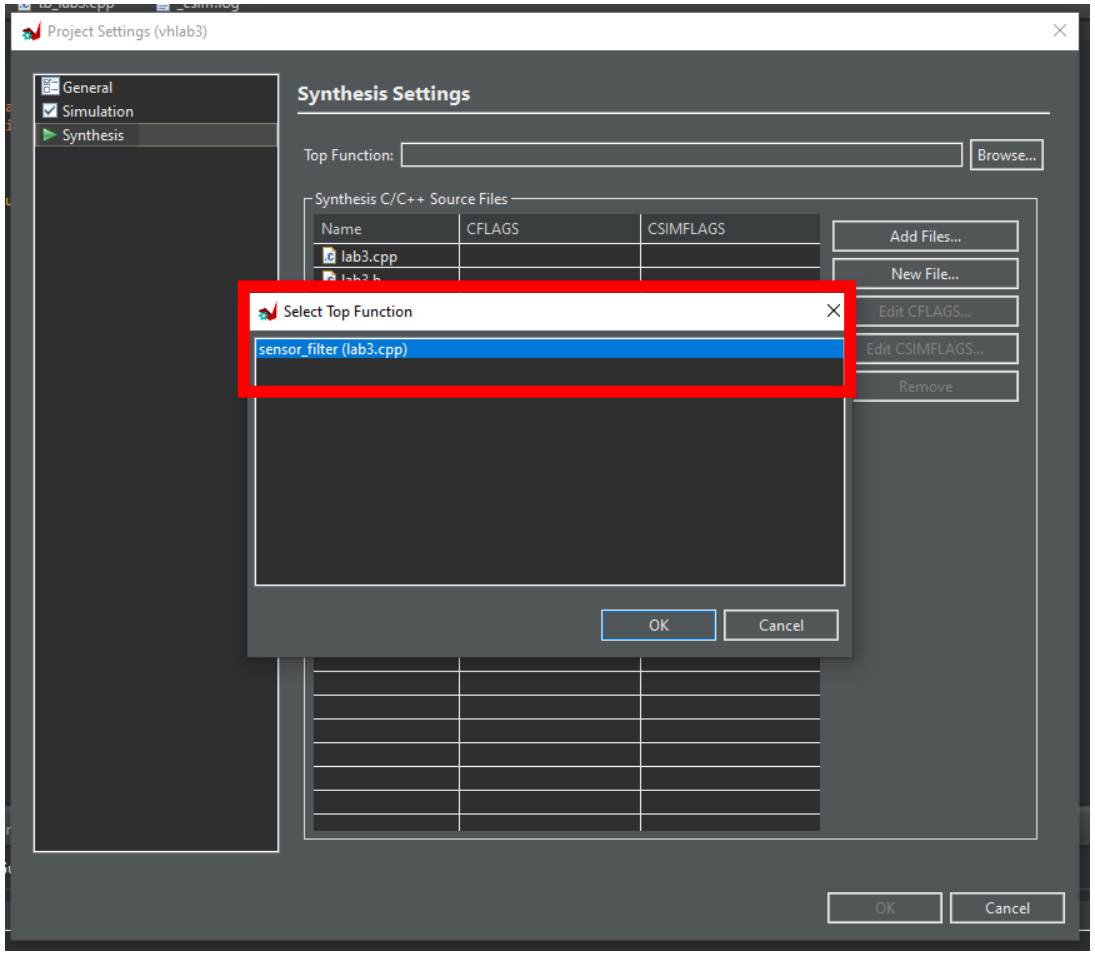
```c
#include "lab3.h"

#define MAX_LINE_LENGTH 80

int main()
{
    FILE *fp;
    char line[80];
    ip_type input[16];
    op_type output;

    printf("HLS Example\n\r");
    // Open a file for the output results
    fp=fopen("input.dat","r");
    if (fp == NULL)
        printf("error opening file\n\r");
    int i = 0;
    while (fgets(line, MAX_LINE_LENGTH, fp) != NULL) {
            input[i] = atoi(line);                    //convert string to integer format
            i++;
    }
    fclose(fp);
    //run the sensor
    sensor_filter(input, &output);
    printf("Result from HLS module = %d \n\r",output);
}
```

Console | Problems | Executables | Debug Shell | Debugger Console

```
<terminated> (exit value: 0) vhlab3.Debug [C/C++ Application] csim.exe
HLS Example
Result from HLS module = 2233
```

Lab 3: Vitis HLS Optimization

Step 1 – Select the top function of Sensor_Filter to allow synthesis – perform synthesis.



Lab 3: Vitis HLS Optimization

Step 1 – Observe the synthesis interval and resources reported in the synthesis report.

Synthesis Summary Report of 'sensor_filter'

General Information

Date: Sat Apr 16 14:54:04 2022
Version: 2021.2 (Build 3367213 on Tue Oct 19 02:48:09 MDT 2021)
Project: vhlab3

Solution: solution1 (Vivado IP Flow Target)
Product family: spartan7
Target device: xc7s100-fgga676-2

Timing Estimate

Target	Estimated	Uncertainty
10.00 ns	4.912 ns	2.70 ns

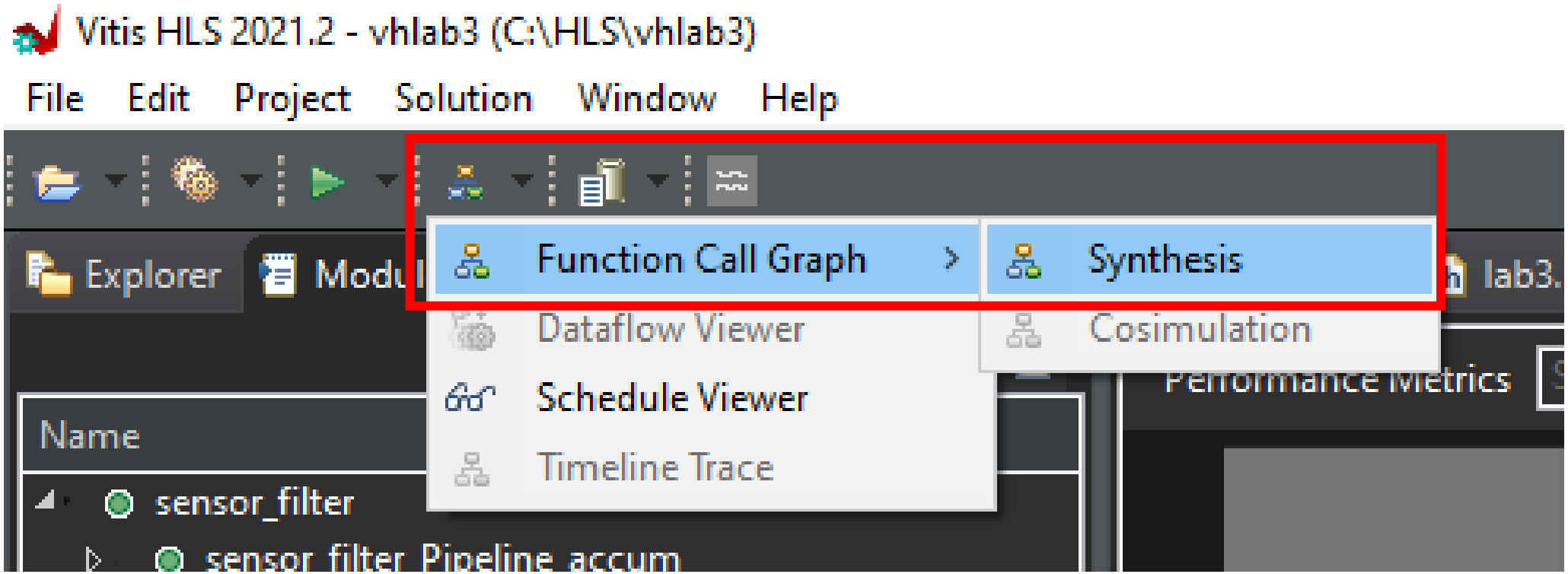
Performance & Resource Estimates

Modules Loops

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
sensor_filter				-	20	200.000	-	21	-	no	0	0	32	184	0
sensor_filter_Pipeline_accum				-	18	180.000	-	18	-	no	0	0	28	97	0
accum				-	16	160.000	2	1	16	yes	-	-	-	-	-

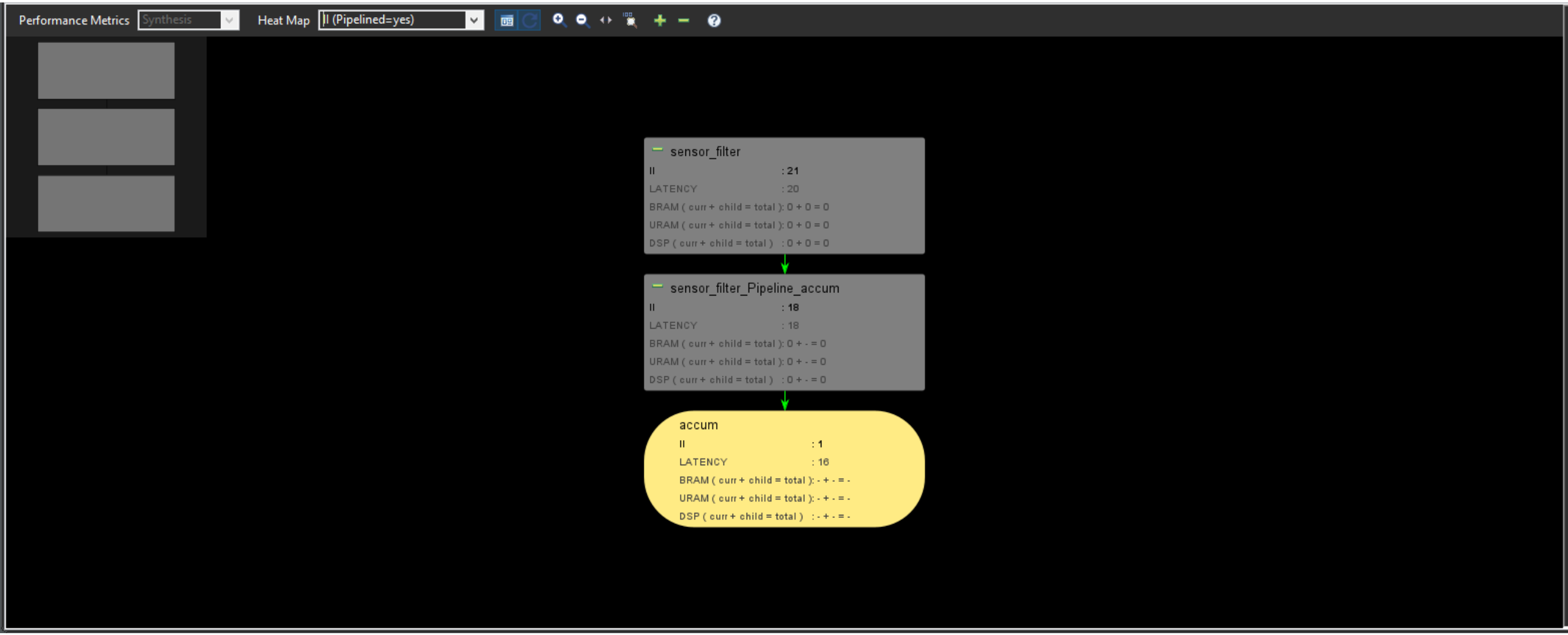
Lab 3: Vitis HLS Optimization

Step 1 – Open the Function Call Graph for the Synthesis.



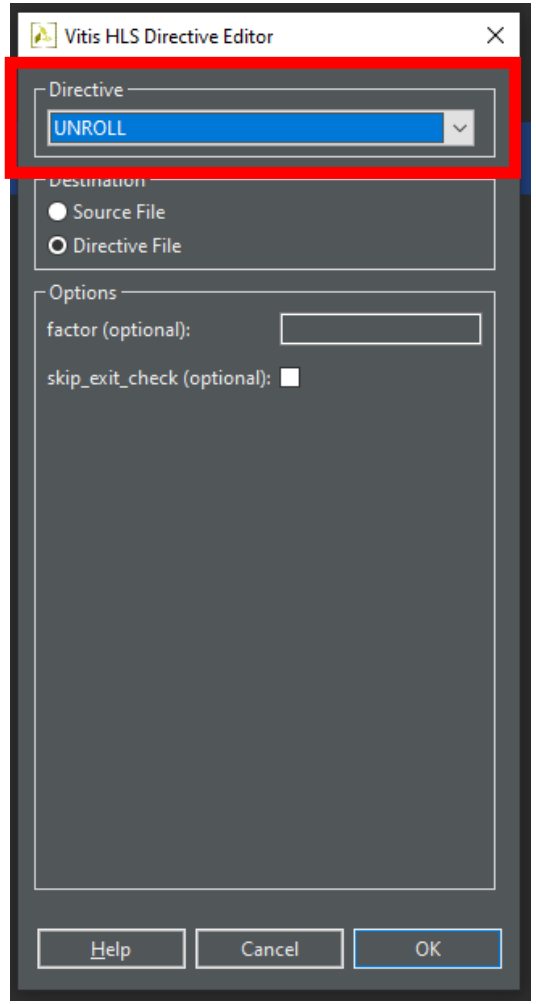
Lab 3: Vitis HLS Optimization

Step 1 – Observe the information presented, change the heat map.



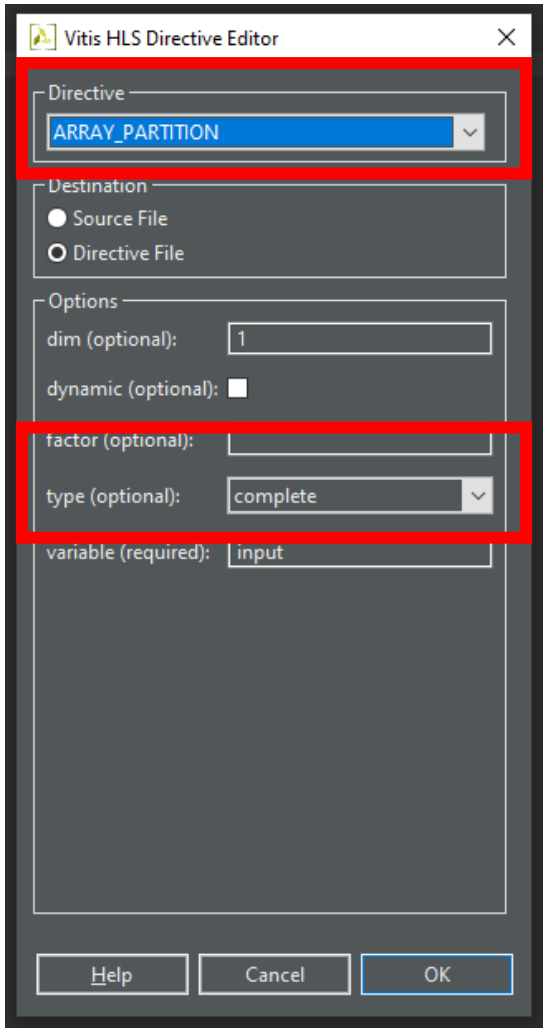
Lab 3: Vitis HLS Optimization

Step 1 – In the directives view for the source code add in a pragma to unroll the loop labelled accum.



Lab 3: Vitis HLS Optimization

Step 1 – In the directives view for the source code add in a pragma to partition the array Input .



Lab 3: Vitis HLS Optimization

Step 1 – Rerun synthesis and observe the interval and resources again.

Date: Sat Apr 16 16:32:27 2022
Version: 2021.2 (Build 3367213 on Tue Oct 19 02:48:09 MDT 2021)
Project: vhlab3

Solution: solution1 (Vivado IP Flow Target)
Product family: spartan7
Target device: xc7s100-fgga676-2

Timing Estimate

Target	Estimated	Uncertainty
10.00 ns	7.152 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latenc	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
sensor_filter				-	1	10.000		2	-	no	0	0	39	452	0

Lab 3 Summary

The concludes lab 3, throughout this lab we have demonstrated

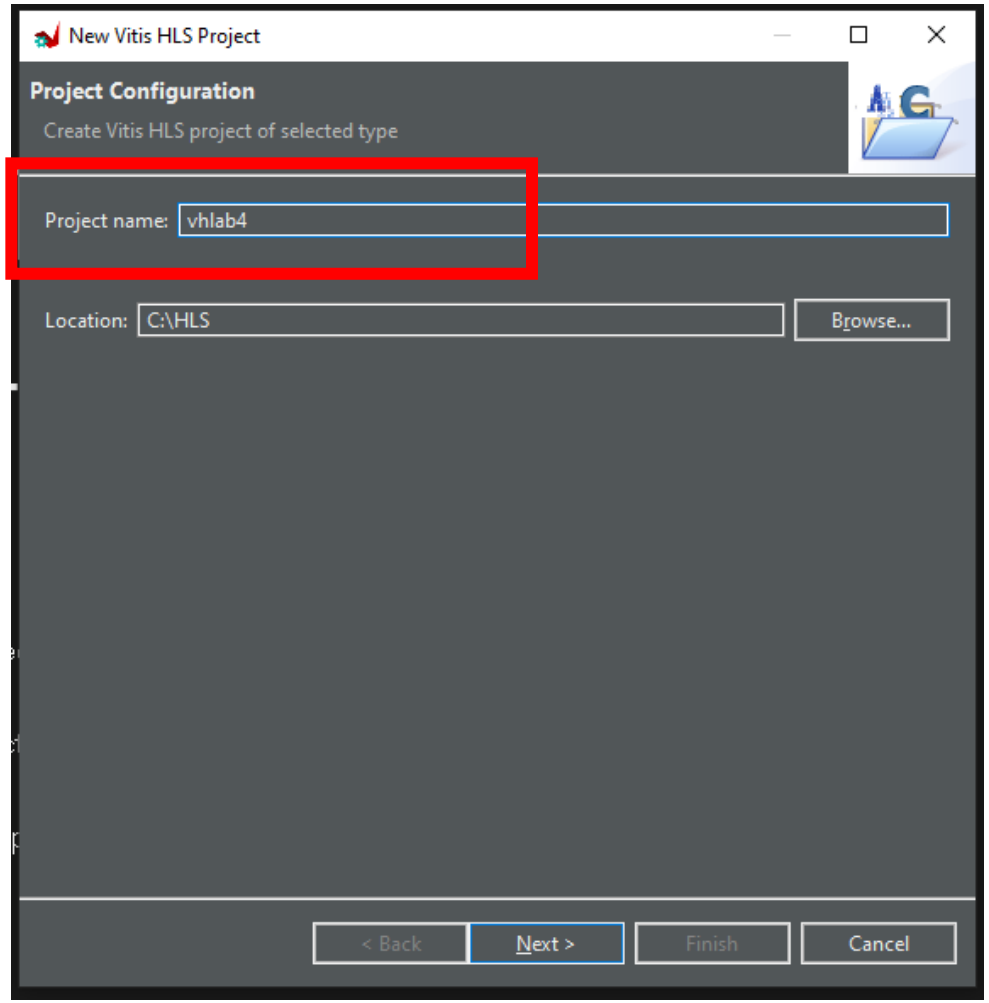
1. How to add optimization pragmas
2. How optimization pragma will change the resource and latency performance
3. How to visualize the optimization impacts

Lab 4

Vitis HLS Arbitrary Precision Math

Lab 4: Vitis HLS Arbitrary Precision Math

Step 1 – Create a new project called vhlab4.



Lab 4: Vitis HLS Arbitrary Precision Math

Step 1 – Create a new source file called lab4.cpp and enter the code below or copy from the GitHub.

```
#include "lab4.h"

static data_type error_prev =0;
static long_long_data_type i_prev=0;

data_type PID (data_type set_point, data_type KP, data_type KI, data_type KD, data_type sample, data_type ts, data_type pmax)
{
#pragma HLS INTERFACE s_axilite port=sample
#pragma HLS INTERFACE s_axilite port=ts
#pragma HLS INTERFACE s_axilite port=KI
#pragma HLS INTERFACE s_axilite port=KP
#pragma HLS INTERFACE s_axilite port=set_point
#pragma HLS INTERFACE s_axilite port=pmax
#pragma HLS INTERFACE s_axilite port=KD
#pragma HLS INTERFACE s_axilite port=return

    data_type error;
    long_data_type p;
    long_long_data_type i, d;
    long_long_data_type op;

    error = set_point - sample;

    p = error * KP;
    i = i_prev + (error  * ts * KI);
    d = KD * ((error - error_prev) / ts);

    op = p+i;//+d;
    error_prev = error;
    if ((op) > pmax) {
        i_prev = i_prev;
        op = pmax;
    }else{
        i_prev = i;
    }
    return op;
}
```

Lab 4: Vitis HLS Arbitrary Precision Math

Step 1 – Create a new source file called lab4.h and enter the code below or copy from the GitHub.

```
#include "ap_fixed.h"

#define float_type

#ifdef float_type
    typedef float data_type;
    typedef float long_data_type;
    typedef float long_long_data_type;
#else
    typedef ap_fixed<16,8> data_type;
    typedef ap_fixed<32,16> long_data_type;
    typedef ap_fixed<48,24> long_long_data_type;
#endif
data_type PID (data_type set_point, data_type KP, data_type KI, data_type KD, data_type sample, data_type ts, data_type pmax);
```

Lab 4: Vitis HLS Arbitrary Precision Math

Step 1 – Create a new source file called tb_lab4.cpp and enter the code below or copy from the GitHub.

```
#include "lab4.h"

#include <stdio.h>
#define iterations 40

int main(void)
{

data_type set_point = -80.0;
data_type sample[iterations] = {-90.000,-88.988,-87.977,-86.966,-85.955,-84.946,-83.936,-82.928,-81.920,-80.912,-80.283,-79.926,
        -79.784,-79.774,-79.829,-79.898,-79.955,-79.993,-80.011,-80.017,-80.016,-80.010,-80.005,-80.002,-80.000,-79.999,
        -79.999,-79.999,-79.999,-80.000,-80.000,-80.000,-80.000,-80.000,-80.000,-80.000,-79.999,-80.000,-80.001,-80.000};



data_type kp = 19.6827; // w/k
data_type ki = 0.7420; // w/k/s
data_type kd = 0.0;
data_type op;

printf("testing cpp\r\n");


for (int i =0; i<iterations; i++){
    op = PID (set_point, kp, ki, kd, sample[i], 12.5, 40);
    printf("result %s\r\n", op.to_string(10,true).c_str());
}
return 0;
}
```

Lab 4: Vitis HLS Arbitrary Precision Math

Step 1 – Run the Synthesis observe the interval and the resources required.

Synthesis Summary Report of 'PID'

General Information

Date: Sat Apr 16 17:22:25 2022
Version: 2021.2 (Build 3367213 on Tue Oct 19 02:48:09 MDT 2021)
Project: vhlab4

Solution: solution1 (Vivado IP Flow Target)
Product family: spartan7
Target device: xc7s100-fgga676-2

Timing Estimate

Target	Estimated	Uncertainty
10.00 ns	6.514 ns	2.70 ns

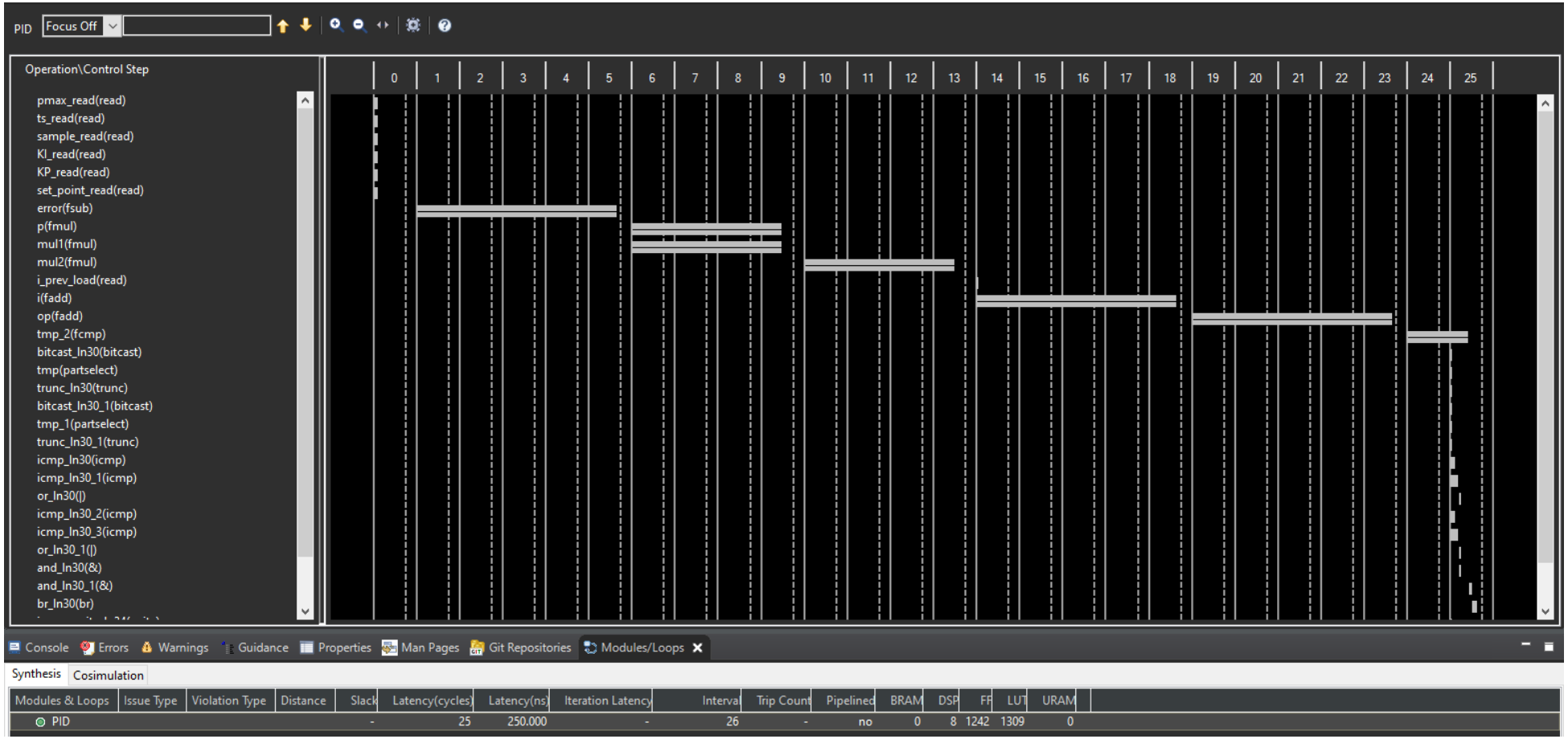
Performance & Resource Estimates

Modules Loops

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latenc	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
PID				-	25	250.000		26	-	no	0	8	1242	1309	0

Lab 4: Vitis HLS Arbitrary Precision Math

Step 1 – Examine the analysis view to see the operations involved.



Lab 4: Vitis HLS Arbitrary Precision Math

Step 1 – In the file lab4.h comment out the line define float_type.

```
#include "ap_fixed.h"

//#define float_type

#ifdef float_type
    typedef float data_type;
    typedef float long_data_type;
    typedef float long_long_data_type;
#else
    typedef ap_fixed<16,8> data_type;
    typedef ap_fixed<32,16> long_data_type;
    typedef ap_fixed<48,24> long_long_data_type;
#endif
data_type PID (data_type set_point, data_type KP, data_type KI, data_type KD, data_type sample, data_type ts, data_type pmax);
```

Lab 4: Vitis HLS Arbitrary Precision Math

Step 1 – Rerun synthesis and observe the interval and resources.

Synthesis Summary Report of 'PID'

General Information

Date: Sat Apr 16 17:25:05 2022
Version: 2021.2 (Build 3367213 on Tue Oct 19 02:48:09 MDT 2021)
Project: vhlab4

Solution: solution1 (Vivado IP Flow Target)
Product family: spartan7
Target device: xc7s100-fgga676-2

Timing Estimate

Target	Estimated	Uncertainty
10.00 ns	7.040 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Laten	Interva	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
PID				-	6	60.000		7	-	no	0	4	492	568	0

Lab 4 Summary

The concludes lab 4, throughout this lab we have demonstrated

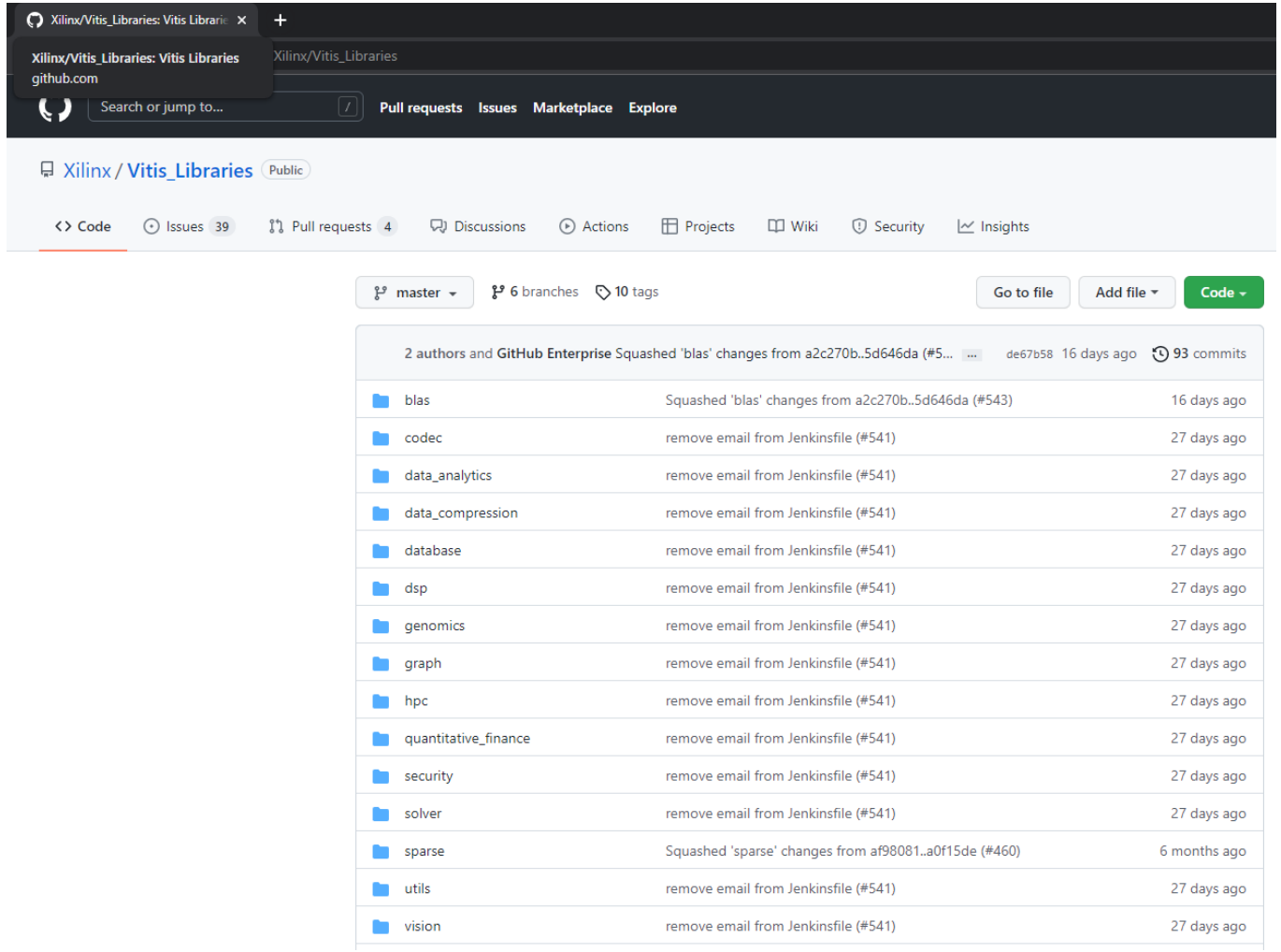
1. How to switch between floating point and arbitrary precision in your application
2. The benefit of using arbitrary precision numbers
3. The difference in performance between the use of floating-point vs arbitrary precision

Lab 5

Vitis HLS Vitis Libraries

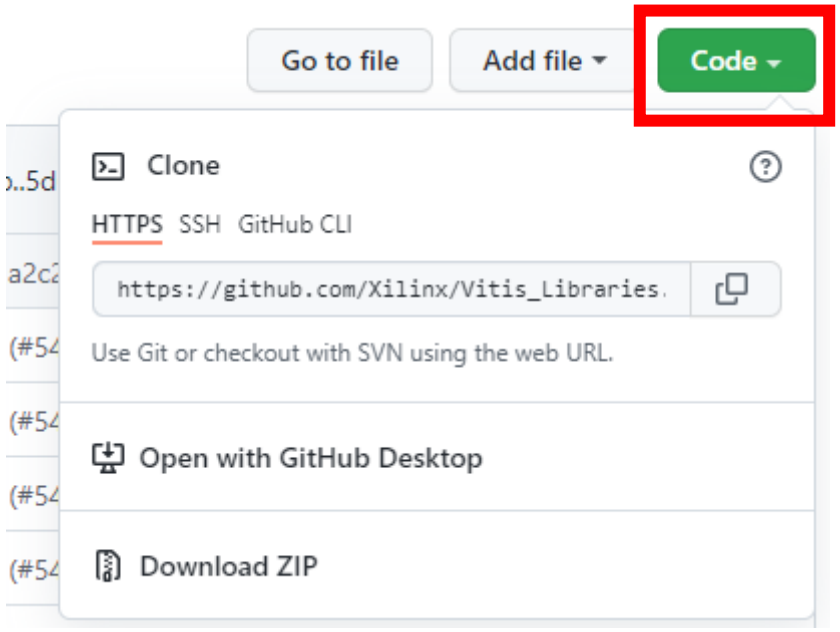
Lab 5: Vitis HLS Vitis Libraries

Step 1 – Open the Vitis libraries repository in GitHub.



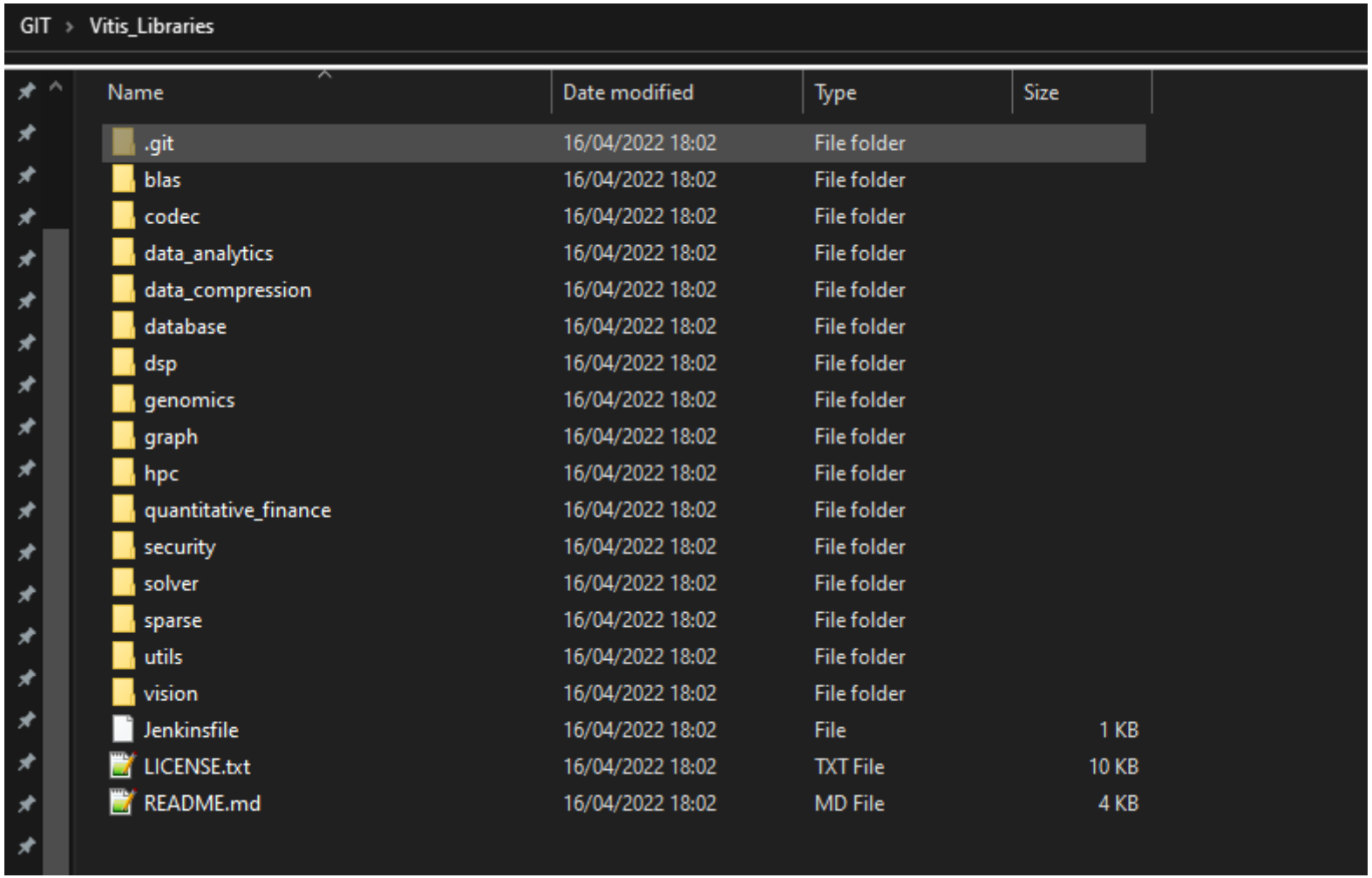
Lab 5: Vitis HLS Vitis Libraries

Step 1 – Download or clone the repository.



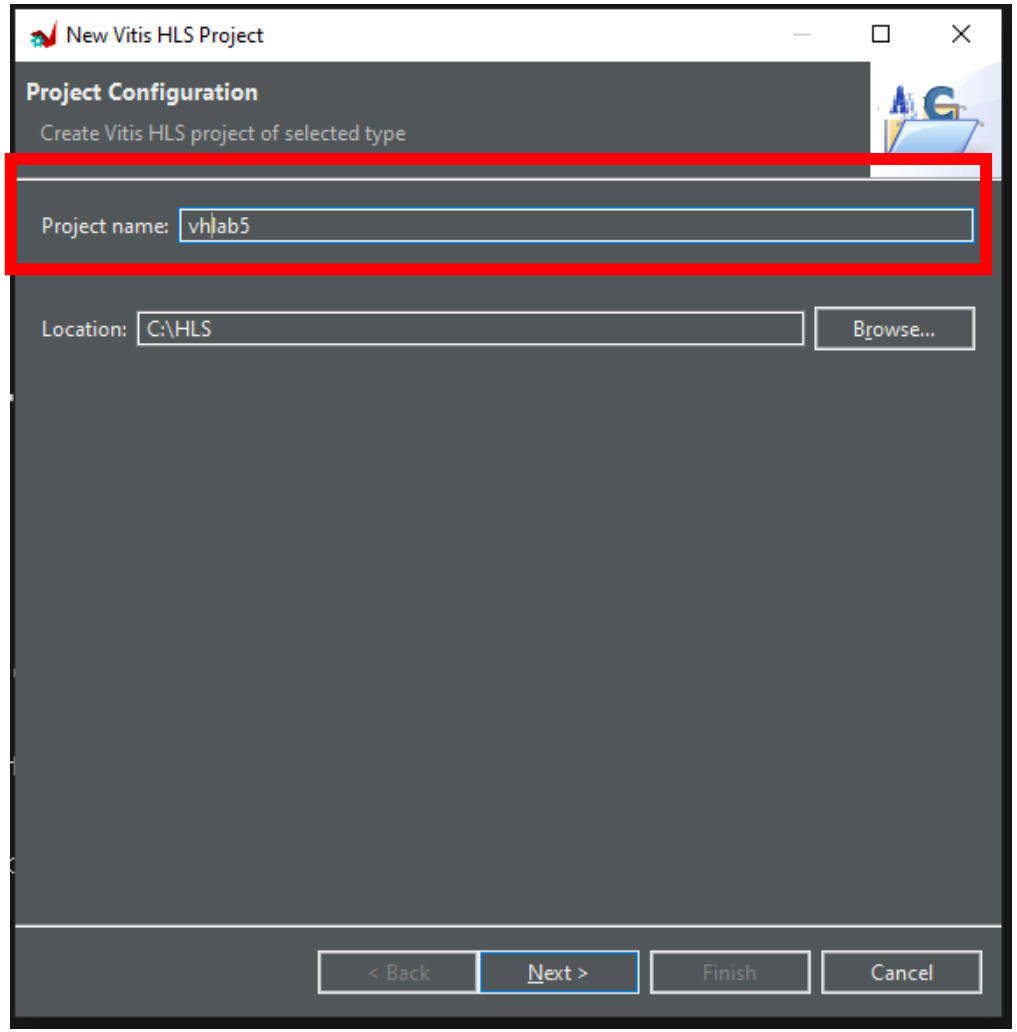
Lab 5: Vitis HLS Vitis Libraries

Step 1 – If you downloaded the Zip Extract the files .



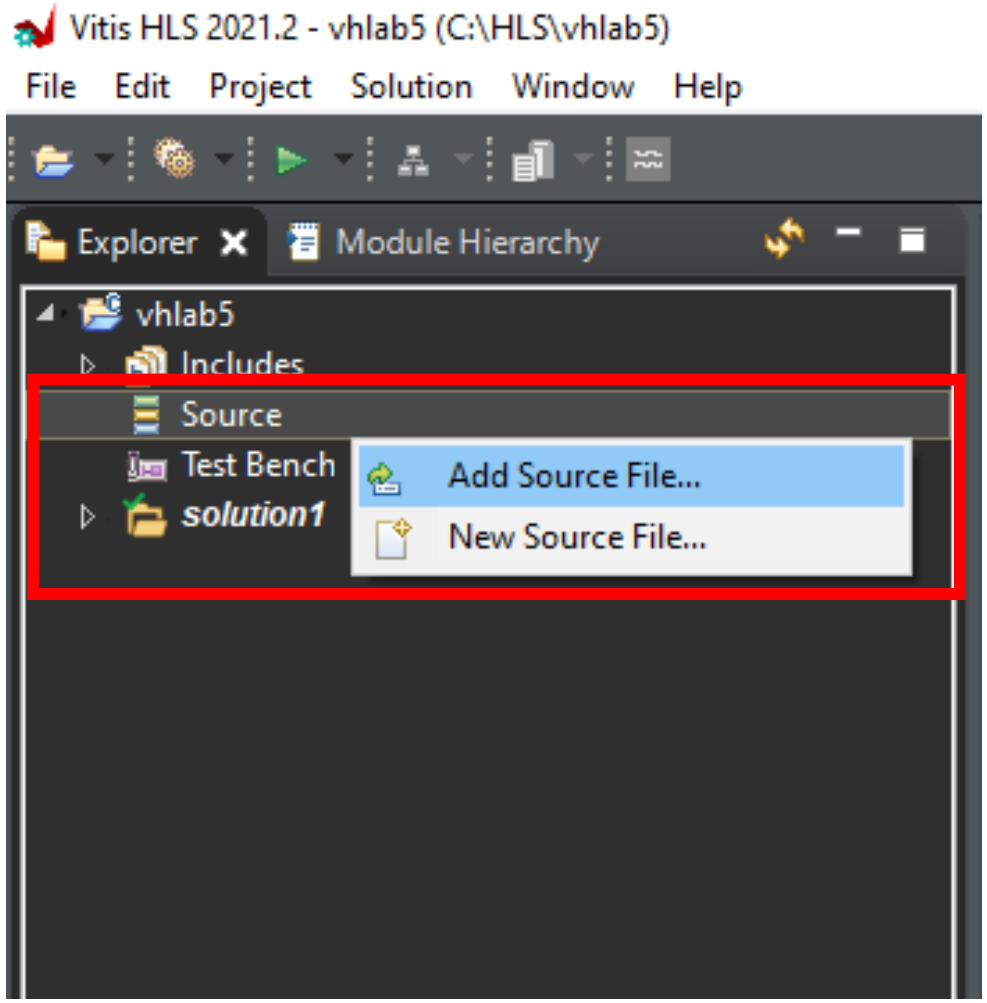
Lab 5: Vitis HLS Vitis Libraries

Step 1 – Create a project called vhlab5.



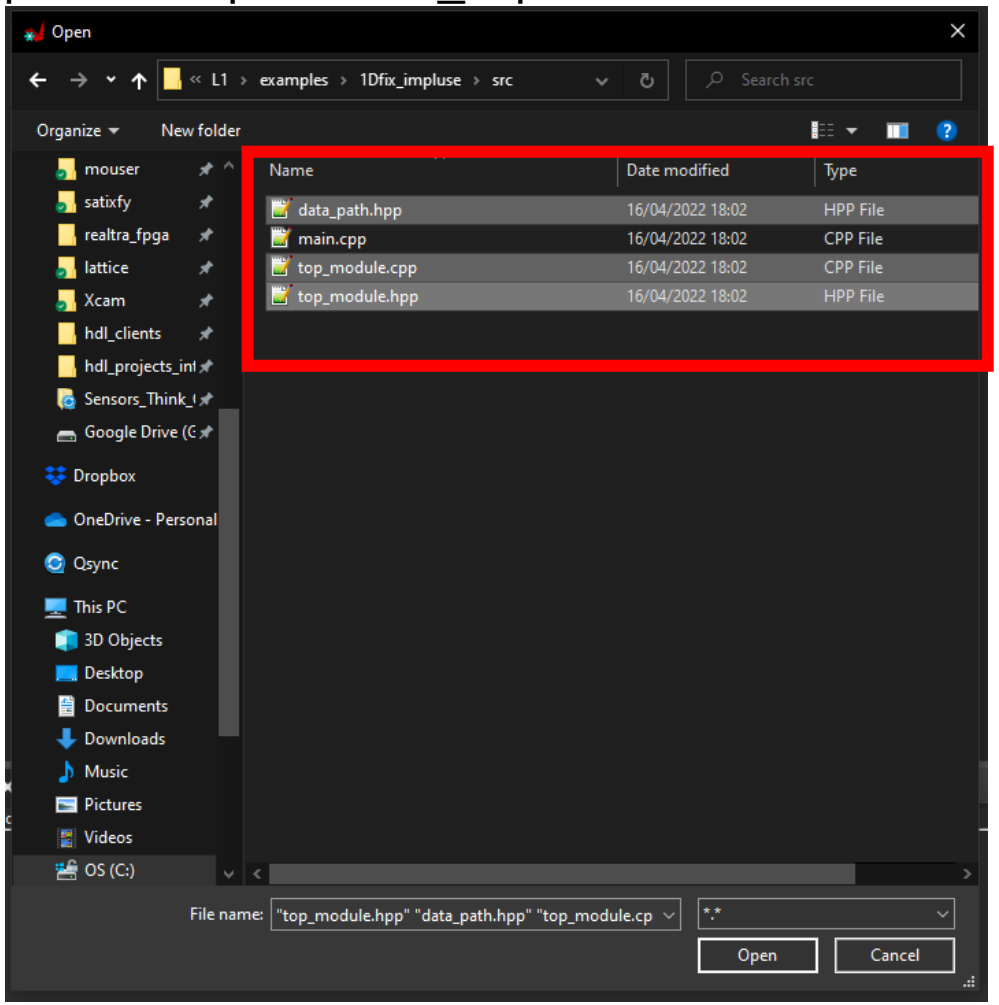
Lab 5: Vitis HLS Vitis Libraries

Step 1 – Right click on source and select Add Source File .



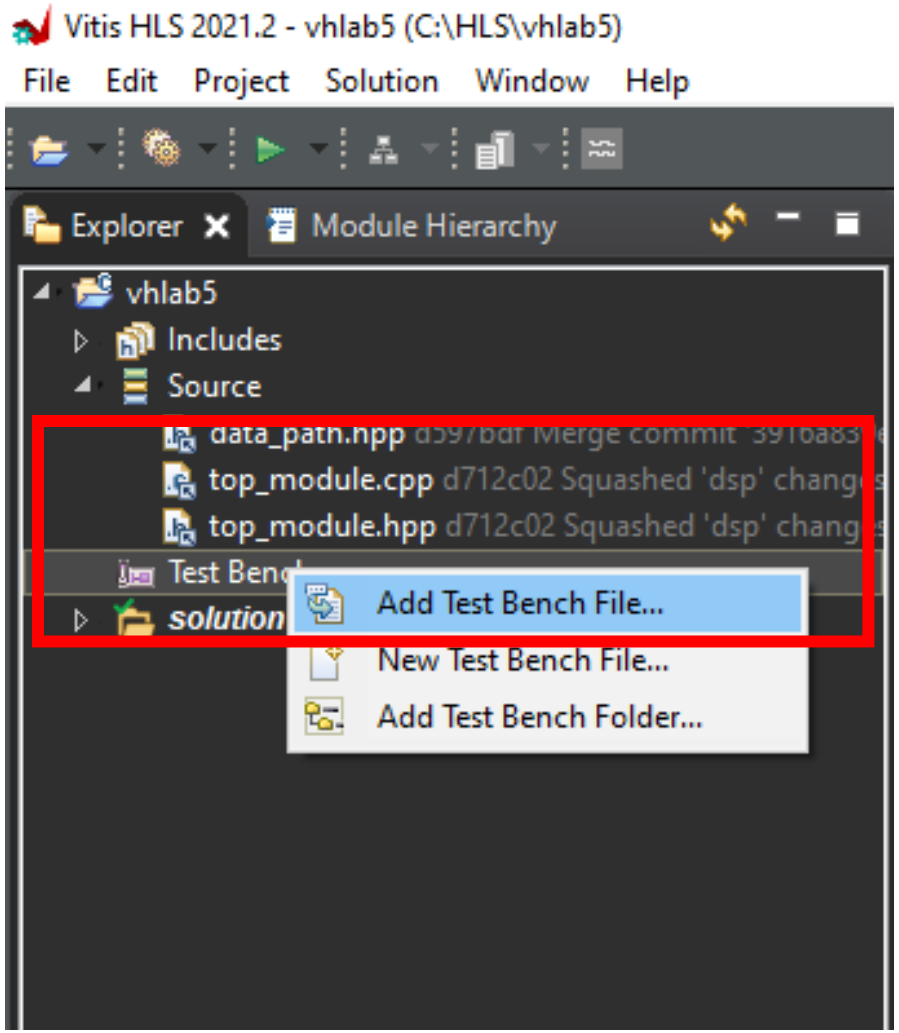
Lab 5: Vitis HLS Vitis Libraries

Step 1 – Select the files, data_path.hpp, top_module.cpp and top_module.hpp from the location <install>\Vitis_Libraries\dsp\L1\examples\1Dfix_impluse\src



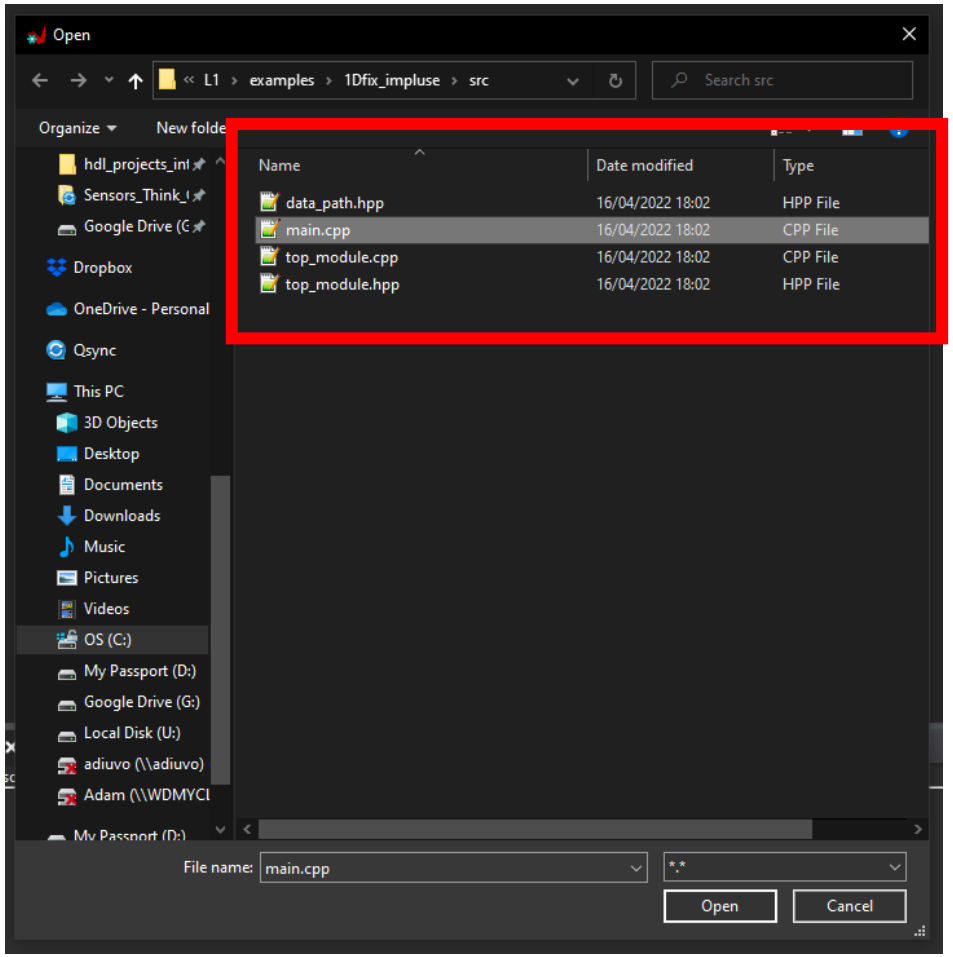
Lab 5: Vitis HLS Vitis Libraries

Step 1 – Right click on Test Bench and select Add Source File .



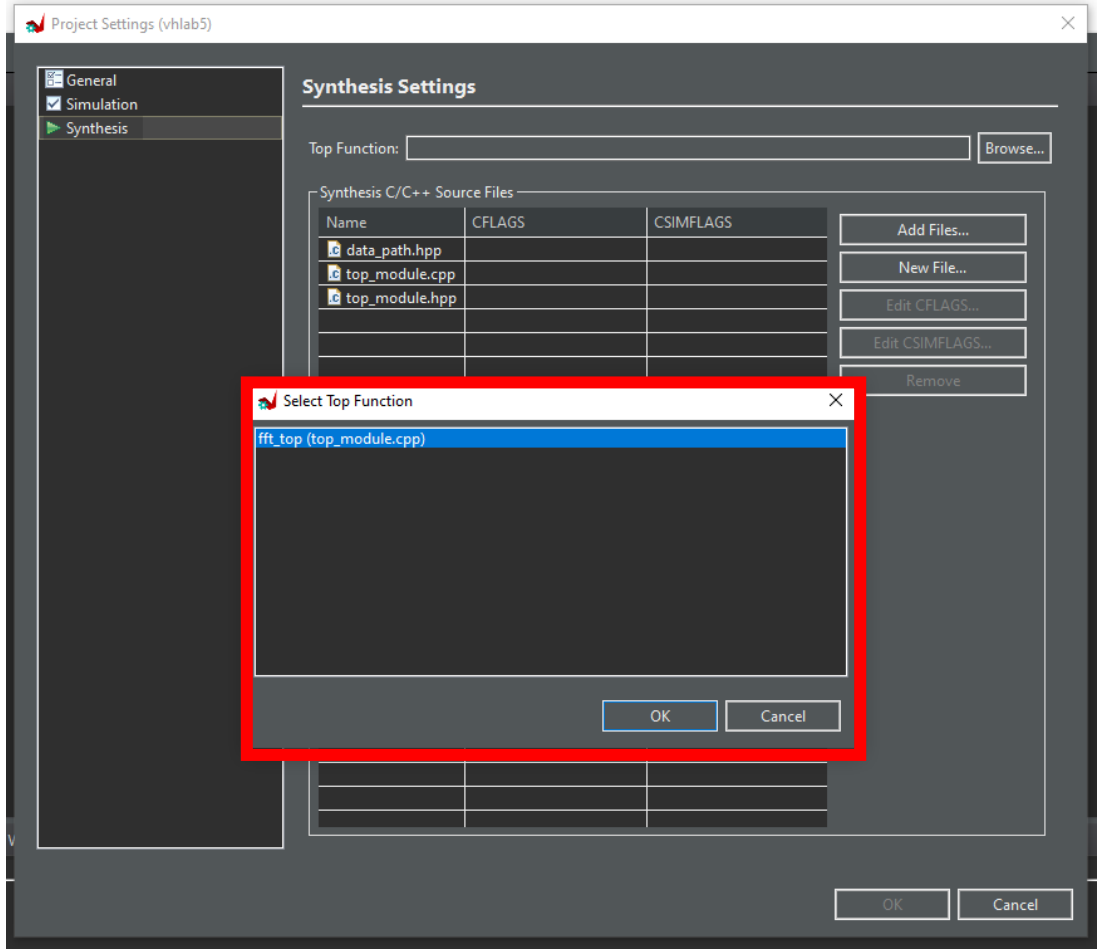
Lab 5: Vitis HLS Vitis Libraries

Step 1 – Select the file main.cpp from the location <install>\Vitis_Libraries\dsp\L1\examples\1Dfix_impluse\src



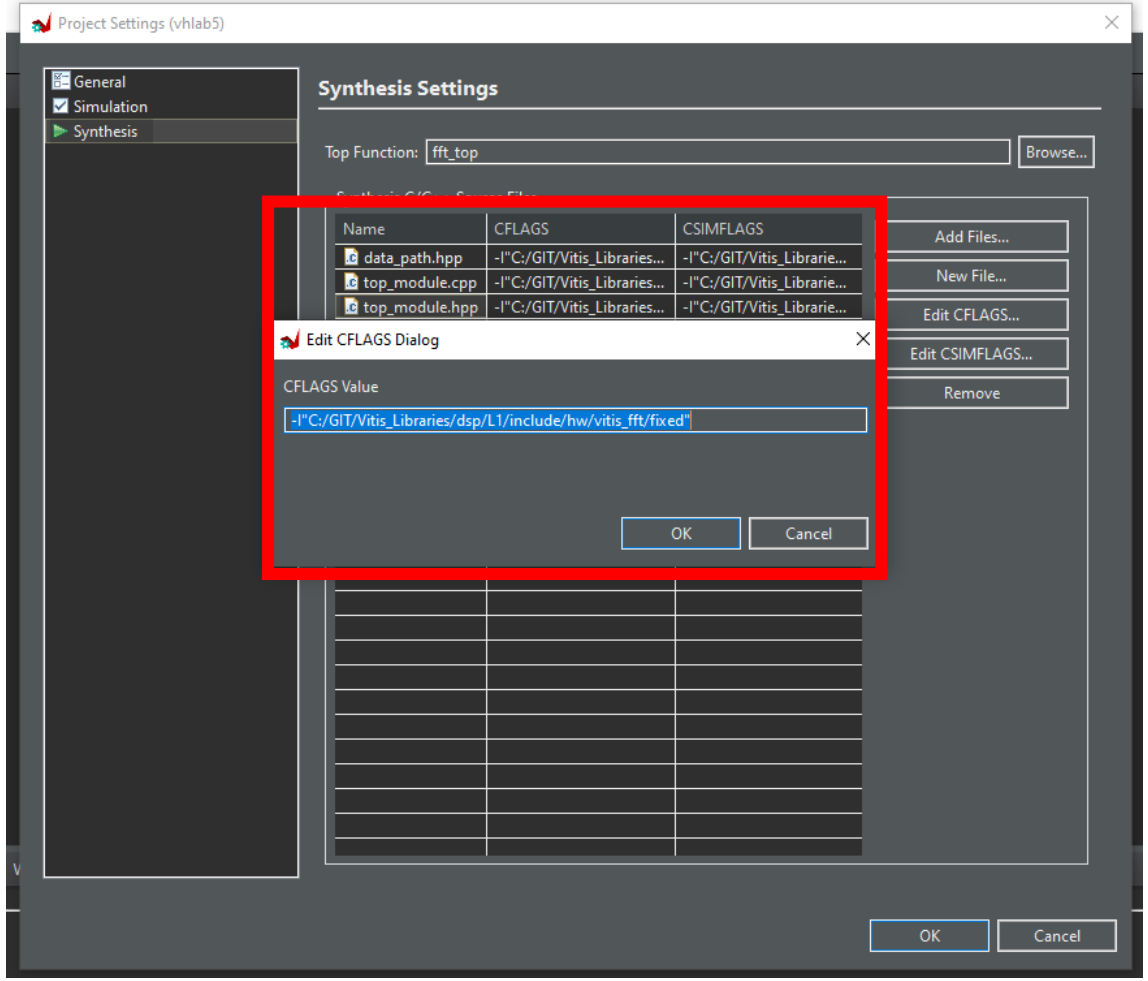
Lab 5: Vitis HLS Vitis Libraries

Step 1 – Select the fft_top as the top function for the project



Lab 5: Vitis HLS Vitis Libraries

Step 1 – Set the Cflags and Csimflags to
-I../../../../../GIT/Vitis_Libraries/dsp/L1/include/hw/vitis_fft/fixed



Lab 5: Vitis HLS Vitis Libraries

Step 1 – Run synthesis this might take some time

Synthesis Summary(solution1)

Synthesis Summary Report of 'fft_top'

General Information

Date: Sat Apr 16 18:51:07 2022
Version: 2021.2 (Build 3367213 on Tue Oct 19 02:48:09 MDT 2021)
Project: vhlab5

Solution: solution1 (Vivado IP Flow Target)
Product family: spartan7
Target device: xc7s100-fgga676-2

Timing Estimate

Target	Estimated	Uncertainty
10.00 ns	7.669 ns	2.70 ns

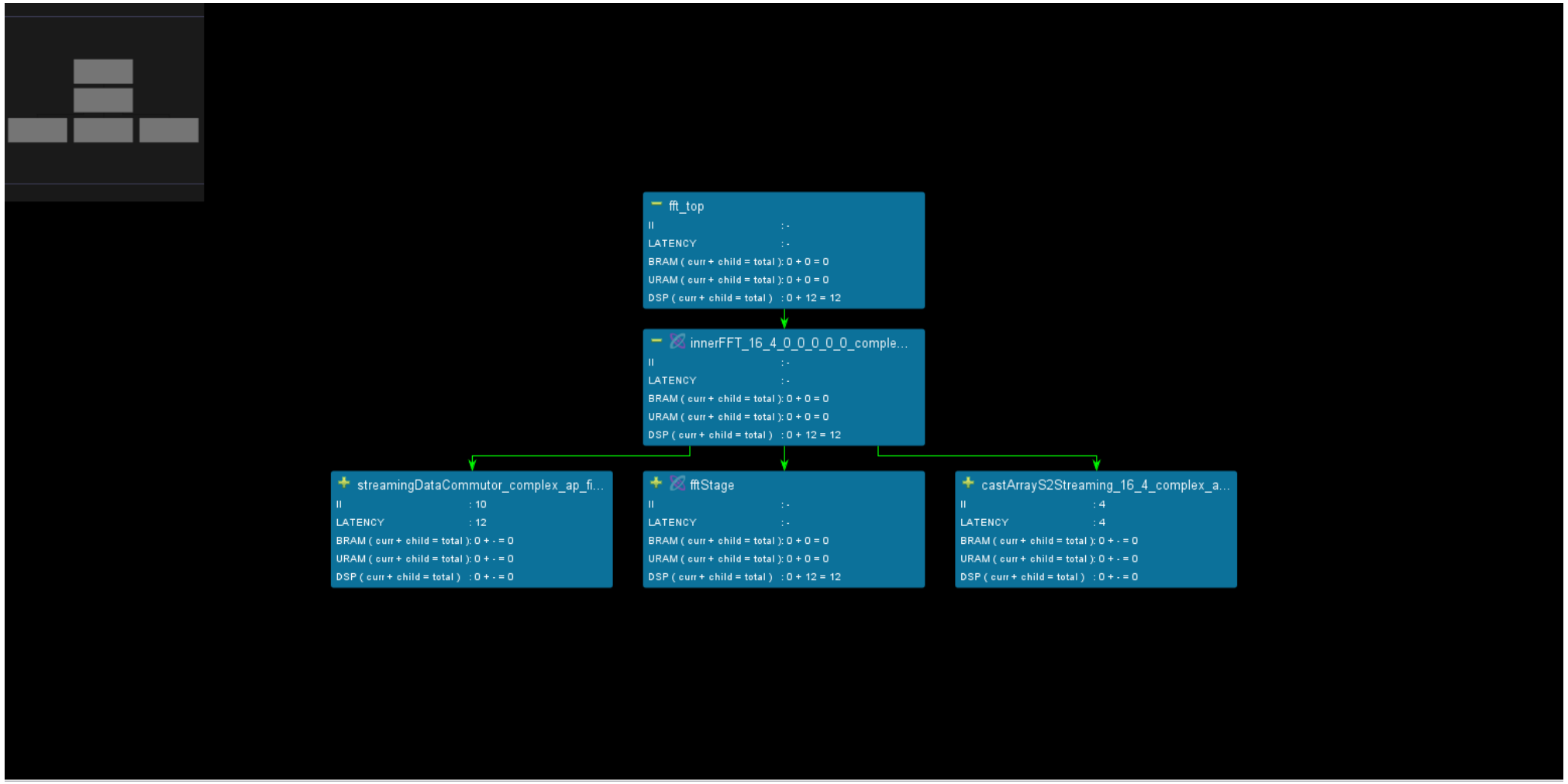
Performance & Resource Estimates

Modules Loops

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval
innerFFT_16_4_0_0_0_0_0_complex_ap_fixed_16_2_5_3_0_complex_ap_fixed_16_2_5_3_0_complex_ap_fixed_16_2_5_3_0_complex_ap_fixed_21_7_5_3_0_s				-0.37	-	-	-	-
castArrayS2Streaming_16_4_complex_ap_fixed_16_2_5_3_0_complex_ap_fixed_16_2_5_3_0_s				-	4	40.000	-	4
streamingDataCommutor_complex_ap_fixed_16_2_5_3_0_s				-	12	120.000	-	10
fftStage	Timing Violation			-0.37	-	-	-	-

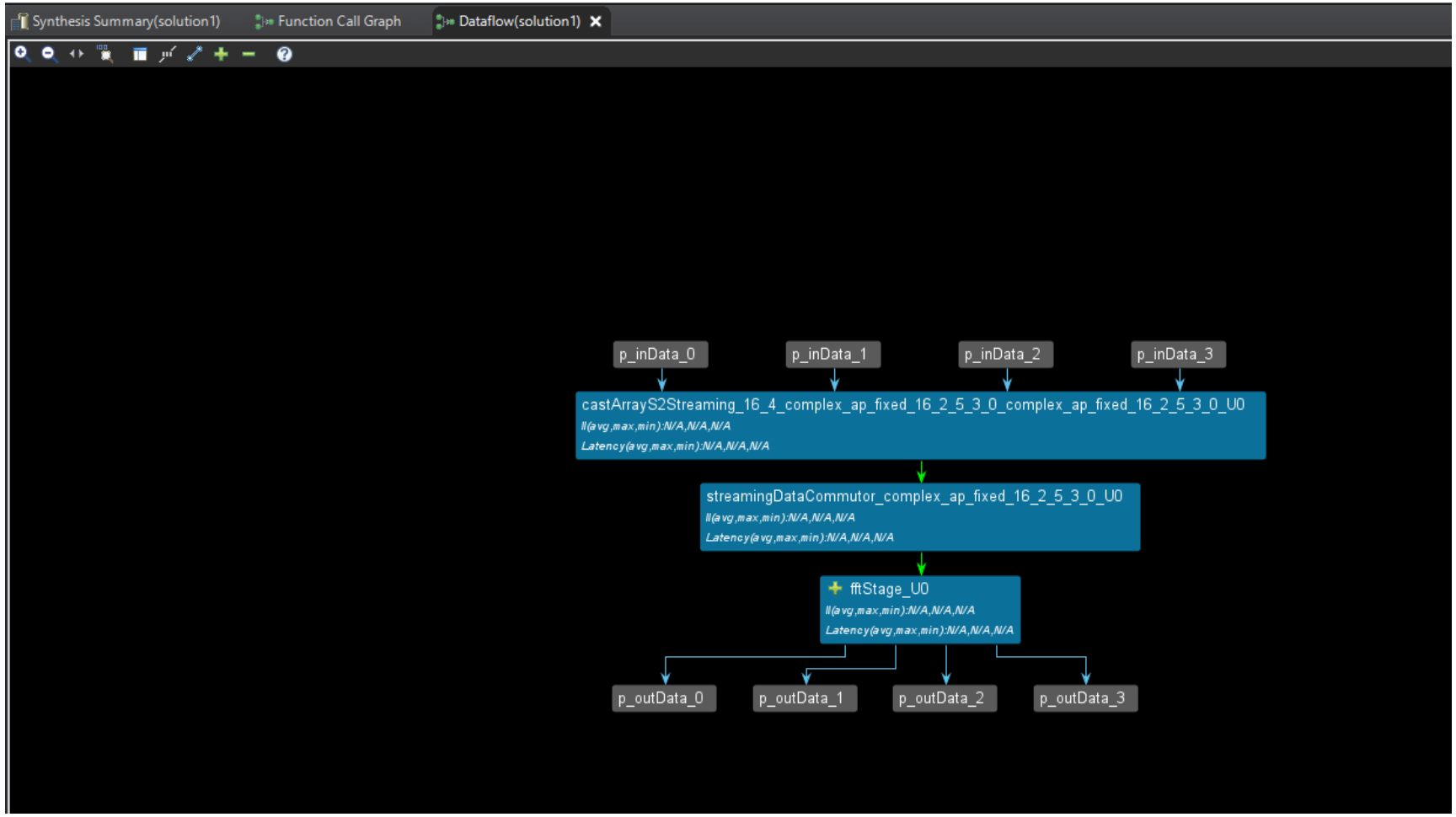
Lab 5: Vitis HLS Vitis Libraries

Step 1 – Examine the function call graph



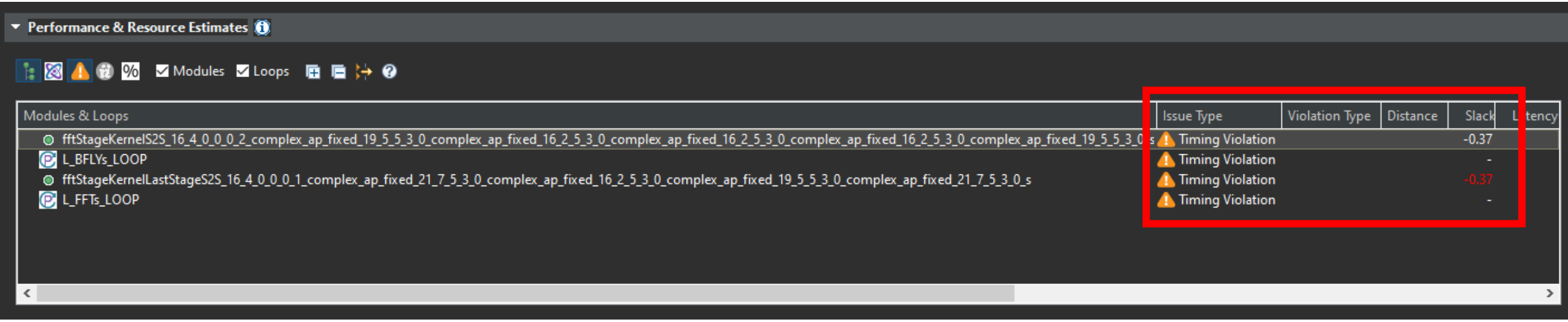
Lab 5: Vitis HLS Vitis Libraries

Step 1 – Examine the Dataflow graph



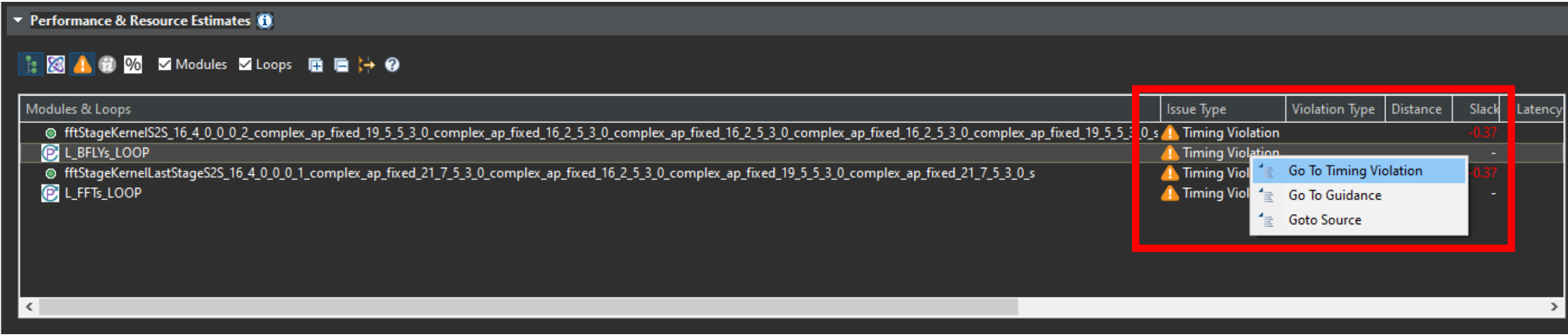
Lab 5: Vitis HLS Vitis Libraries

Step 1 – Notice there is a timing error,



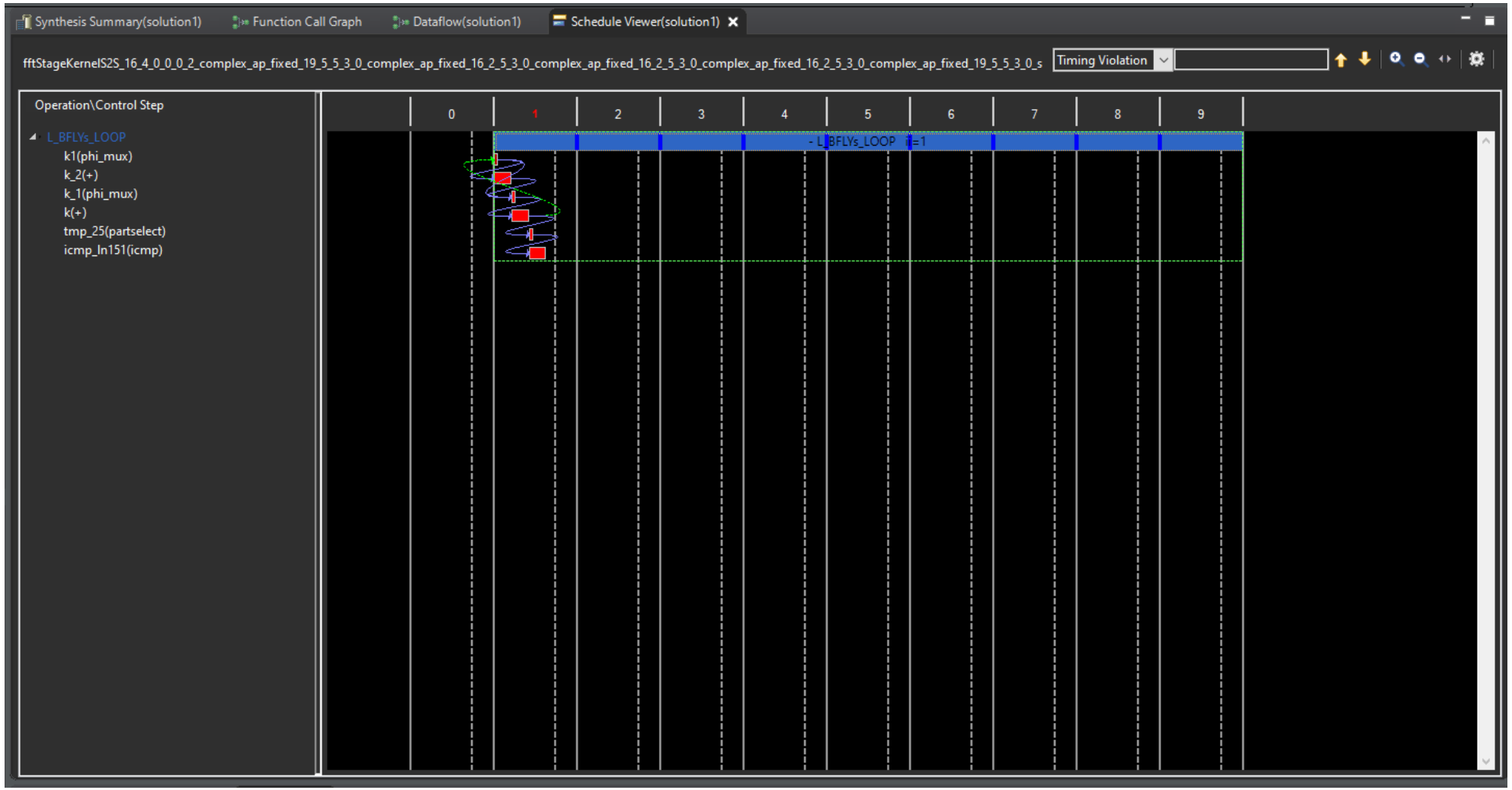
Lab 5: Vitis HLS Vitis Libraries

Step 1 – Right click on the error and select go to timing violation



Lab 5: Vitis HLS Vitis Libraries

Step 1 – Observe the timing violation in the schedule viewer



Lab 5: Vitis HLS Vitis Libraries

Step 1 – Back in the timing violation, go to the source- notice the line of code causing the delay is highlighted.

Synthesis Summary(solution1) | Schedule Viewer(solution1)

Project: vhlab5 Target device: xc7s100-fgga676-2

Timing Estimate

Target	Estimated	Uncertainty
10.00 ns	7.669 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

Modules & Loops

fftStageKernelS2S_16_4_0_0_0_2_complex_ap_fixed_19_5_5_3_0_complex_ap_fixed_16_2_5_3_0_complex_ap_fixed_16_2_5_3_0_complex_ap_fixed_16_2_5_3_0_complex

HW Interfaces

AP_FIFO

Interface	Data Width
p_inData_0	32
p_inData_1	32
p_inData_2	32
p_inData_3	32
p_outData_0	42
p_outData_1	42
p_outData_2	42
p_outData_3	42

TOP LEVEL CONTROL

Interface	Type	Ports

hls_ssr_fft.hpp

```
138                         hls::stream<SuperSampleContainer<t_R, T_out> >& p_fftOutDa
139     //#pragma HLS INLINE recursive
140
141     const int NO_OF_FFT_STAGES = ssrFFTLog2<t_L>::val / ssrFFTLog2<t_R>::val;
142     const unsigned int s = NO_OF_FFT_STAGES - stage;
143     static const bool isFirstStage = (NO_OF_FFT_STAGES == stage);
144     const int no_of_ffts_in_stage = ssrFFTPow<t_R, s>::val;
145     const int current_fft_length = t_L / ssrFFTPow<t_R, s>::val;
146     const int no_bflys_per_fft = current_fft_length / t_R;
147
148 L_FFTs_LOOP:
149     for (int f = 0; f < no_of_ffts_in_stage; f++) {
150     L_BFLYs_LOOP:
151         for (int k = 0; k < no_bflys_per_fft; k++) {
152 #pragma HLS PIPELINE II = 1 rewind
153             if (p_fftReOrderedInput.empty()) {
154                 k--;
155             } else {
156                 T_in X_of_ns[t_R];
157                 T_complexMulOutType complexExpMulOut[t_R];
158
159                 //#pragma HLS data_pack variable = complexExpMulOut
160
161                 T_out bflyOutData[t_R];
162
163                 //#pragma HLS data_pack variable = bflyOutData
164
165                 SuperSampleContainer<t_R, T_in> temp_super_sample_in = p_fftReOrd
166             // The following loop should be unrolled for implementation
167             L_READ_R_IN_SAMPLES:
168                 for (int n = 0; n < t_R; n++) {
169 #pragma HLS UNROLL
170                     X_of_ns[n] = temp_super_sample_in.superSample[n];
171                 }
172                 Butterfly<t_R> Butterfly_obj;
173                 Butterfly_obj
174                     .template calcButterFly<t_L, isFirstStage, t_scalingMode, tra
175                         X_of_ns, bflyOutData, p_complexExpTable);
176                 twiddleFactorMulS2S<t_L, t_R, transform_direction, butterfly_rnd_
177                     bflyOutData, complexExpMulOut, p_twiddleTable, (k << (ssrFFTL
178
179                 SuperSampleContainer<t_R, T_out> temp_super_sample_out;
180             L_WRITE_R_BUTTERFLY_OUT_SAMPLES:
```

Lab 5 Summary

The concludes lab 5, throughout this lab we have demonstrated

1. How to access Vitis Libraries
2. How to include Level One elements from the library within Vitis HLS
3. How to include the Vitis libraries to build to application
4. How to observe the results in the function call and dataflow graph
5. How to identify a timing violation
6. How to view the violation in the schedule viewer
7. How to identify the violation in the source code.