Santa Clara University
ELEN266
Assignment: Inner Product

Instructor: Hossein Omidian
Copyright: Chris Dick

Winter 2023

In this assignment you will design an FPGA implementation of a circuit to compute inner-products using the Vivado/Vitis High Level Synthesis (HLS) design flow.

This assignment counts towards you final grade and is worth 20 marks, instructions for submitting your work are provided at the end of this document.

Some design files will be supplied as a starting point for the project, this will be discussed in class.

Note that some of the GUI screenshots in this document might not look exactly like the corresponding GUIs in the version of the tools you are using, but that's ok, if there are any differences, they will be relatively minor1

1 Inner-Product

The inner product $<a; b>$, also referred to as dot product, of two length N column vectors a and a is defined as

$$< a; b >= a^T b = \sum_{i=0}^{N} a_i b_i \tag{1}$$

Evaluating Equation 1, requires N multiplications and $N - 1$ additions, so it is a calculation that is of order N, or in big O notation, $O(N)$. Note that N memory reads are required to read all of the values in the a and similarly N memory reads are required to access all of the data in the b array. One write operation is required to return the final result to memory.

The inner product is a very common linear algebra operation that forms the foundation of many machine learning and signal processing functions such as filtering and matrix multiplication.

In this assignment you will learn how to implement inner product calculations on an FPGA using C as the programming language. You will learn how to

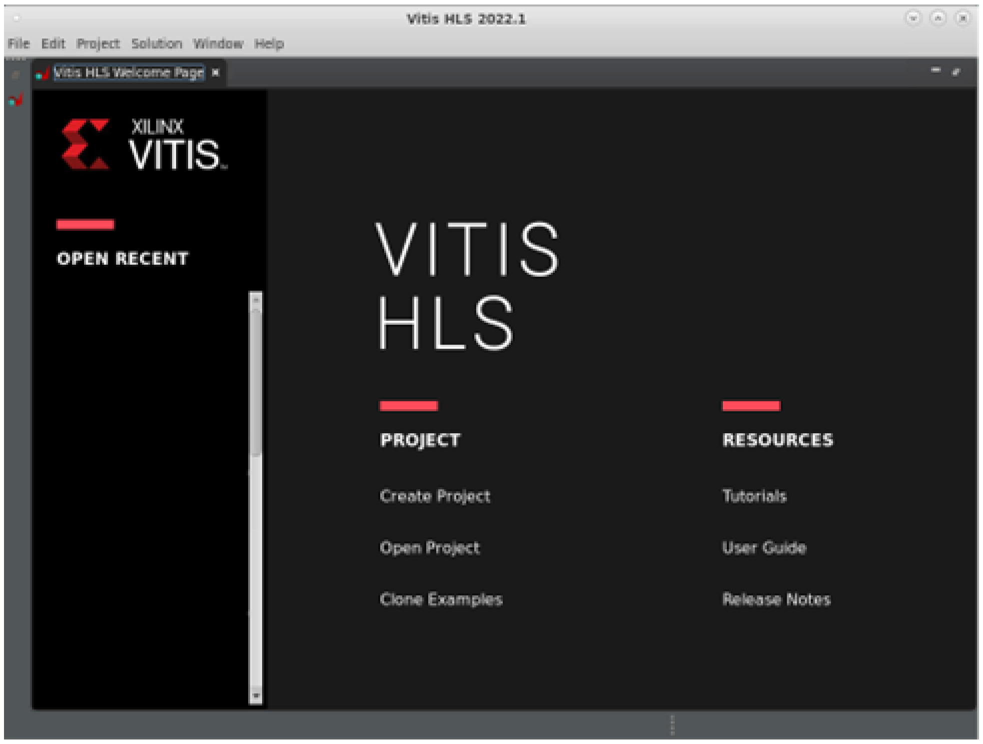


Figure 1: Vitis HLS welcome screen

use the AMD Vivado/Vitis HLS tool chain and how to generate different versions of $<a; b>$ that provide design tradeoffs between hardware cost (area) and performance as measured by the number of inner product calculations computed per second.

1.1 Baseline Implementation: Solution 1

This section describes the process for generating an HLS implementation of the inner product calculation.

The following sections of this document contain screenshots of various Vitis HLS windows. They may not exactly match the screens that you will see when running the tools as there are some differences depending on the version of the tools that you are running.

Start Vitis HLS. The Welcome screen of Figure 1 will open. Create a new project by selecting Create New Project located on the left of the window. In the next few steps the C/C++ source code for the design and associated testbench and other files required by the testbench are added to the project.

The top-level design function, the function that is to be implemented in the FPGA is called *inner_product*. The corresponding source file is *inner_product.cpp*.

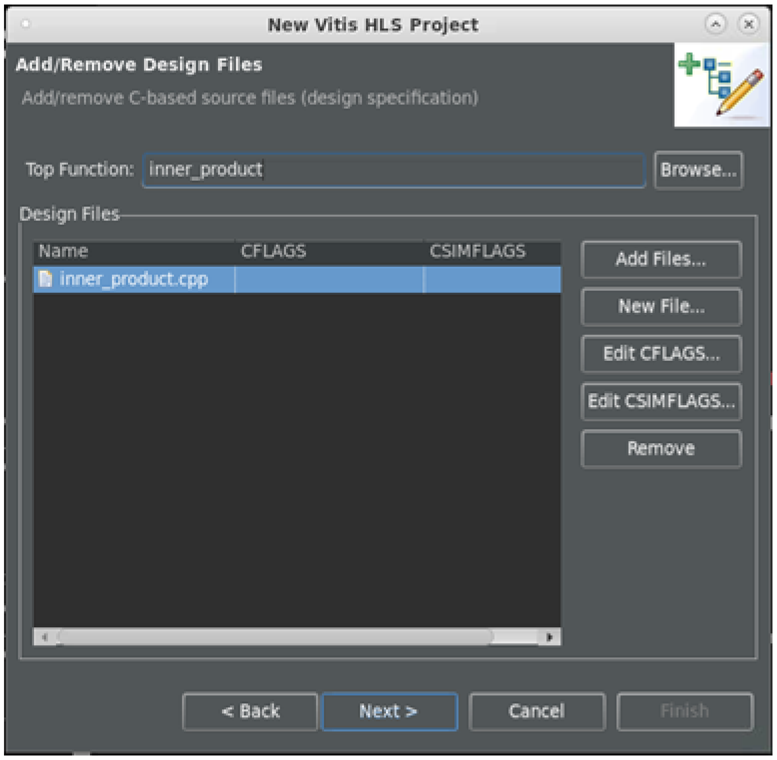
In the text box at the top of Figure 2 enter the top-level function name, that is, *inner_product*. Click Add Files, navigate to the location of the file *inner_product.cpp*, select it, and add it to the project. If the design employs multiple C/C++ source files, then all of the files would be added to the project. This project only has one source code file. Click through to the next configuration screen by selecting Next located at the bottom of the window. Click Add Files on the page shown in Figure 3 to add the testbench *tb_inner_product.cpp* to the project. The testbench has a dependency on the header file *inner_product.h*. The header file is automatically added to the project as long as it resides in the main project directory, that is the directory containing *inner_product.cpp* and *tb_inner_product.cpp* in this case. Click through to the Solution Configuration page shown in Figure 4. Leave the default name, Solution1 in the Solution Name textbox. The Clock Period textbox captures the target clock period for the design, the target operating clock frequency for the FPGA. There is no guarantee that the design will, out-of-the-box, achieve this target. It might be necessary to apply various implementation directives and re-factor the C/C++ source code in order to close timing and accomplish other design objectives, particularly for high clock frequency designs. The next step is to select the FPGA device by clicking through the button on the right side of the Part Selection section of the page. From the menu shown in Figure 4 select the *Kintex xc7k160tfbg676-2* device.

Click Finish and the Vitis HLS Integrated Development Environment (IDE) screen shown in Figure 6 is displayed. Some annotations have been overlaid on this page to identify the functions for several of the key icons in the menu ribbon.

Go to *Solution* in the menu ribbon and select *Solution Setting*. In the *General* tab, select *config_compile* and make sure *pipeline_loops* Value is set to zero as it's shown in figure 7.

The IDE provides an environment for editing code, executing C/C++ code, RTL simulation and running the entire FPGA tool chain starting with high-level synthesis of the C/C++ source code through to place-and-route in the backend physical implementation tools. The project files can be accessed from the Explorer panel on the left of the page. Open inner_product.cpp from the Explorer panel by clicking on source and selecting inner_product.cpp from the file list - in this case there is only a single file in the list.

Listing 1 show the testbench for the project. Two length N vectors, a and b are declared (line 13), these will be the inner product input vectors. Variable s will hold the result of the inner product calculation produced from the golden reference inner product function and $s1$ will store the result of the inner product computed by the design function described in *inner_product.cpp*. The input arrays are initialized with some random integers with the for loop starting on line 15. Line 21 is the call to the inner product reference function *inner_product_reference()* and line 21 calls the design function *inner_product()*. Line 26 validates the design function by comparing the result of the inner product, *s1*, generated by the design function with the golden reference value *s*. The comparison sets the simulation status flag, *sim_status*, to 1 if a failure is


New Vitis HLS Project

Add/Remove Design Files

Add/remove C-based source files (design specification)

Top Function: inner_product

Design Files

Name	CFLAGS	CSIMFLAGS
inner_product.cpp		

Figure 2: Adding the C/C++ source code to the project

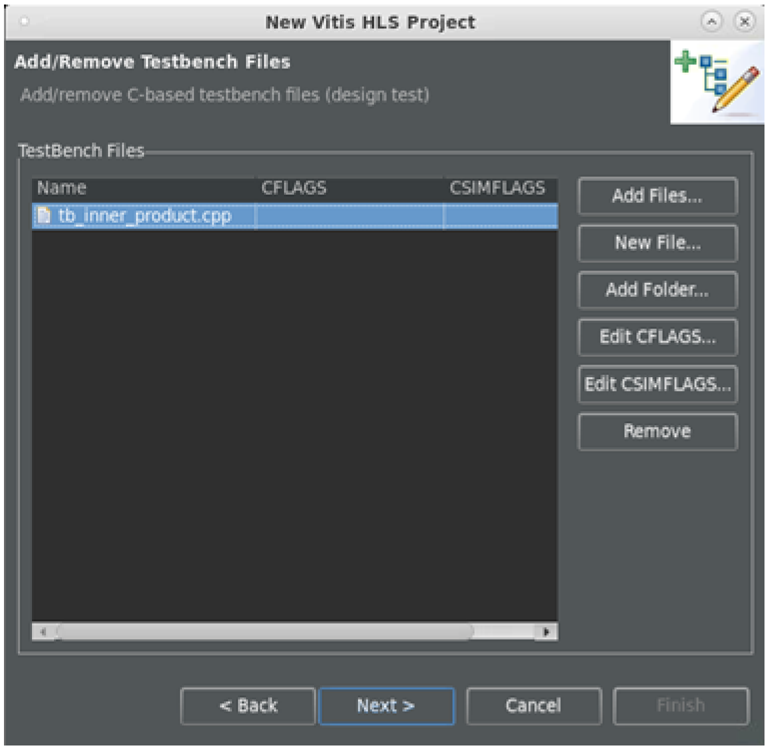
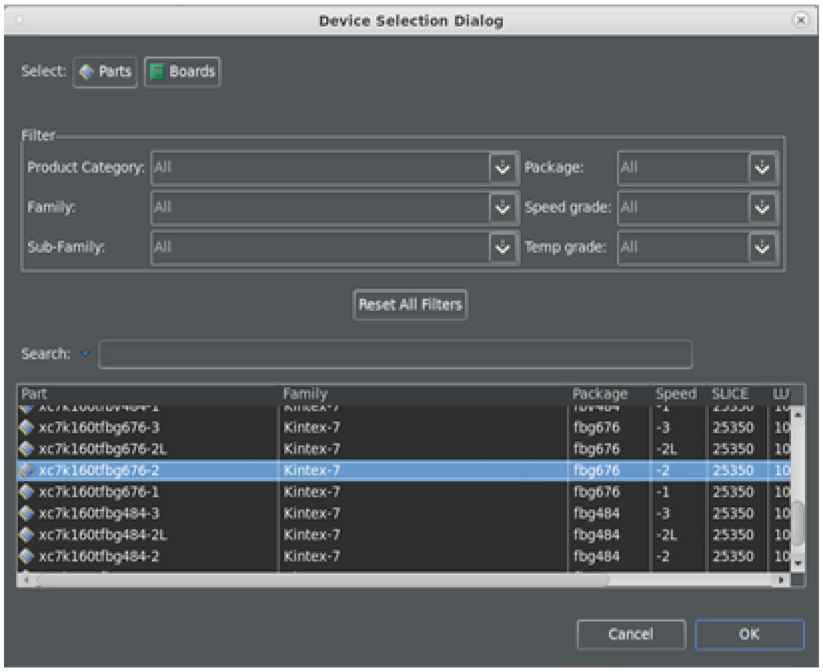


Figure 3: Adding the project testbench.



Filter

Product Category: All
Package: All
Family: All
Speed grade: All
Sub-Family: All
Temp grade: All

Search:

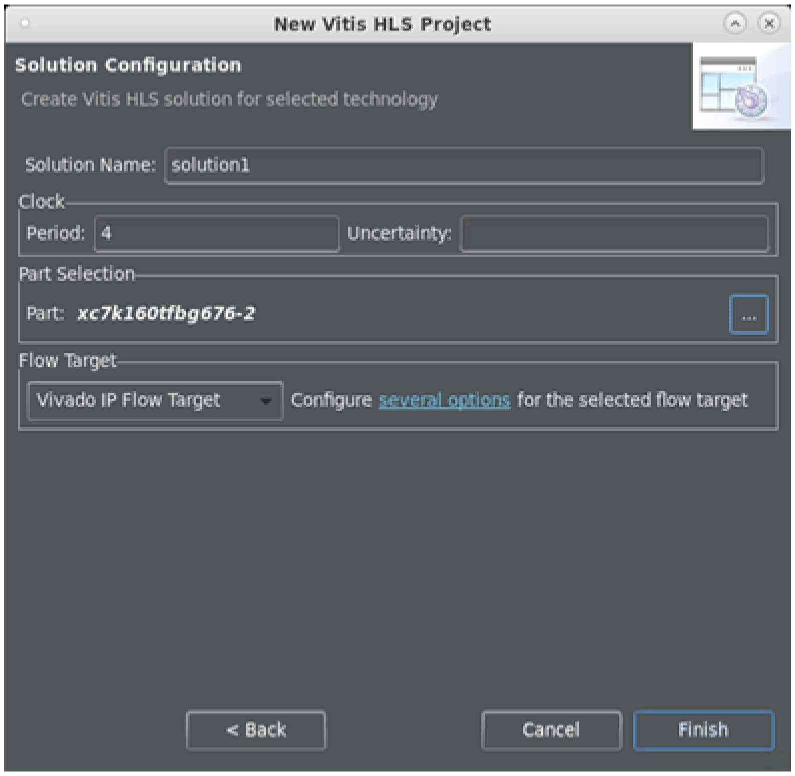
Part	Family	Package	Speed	SLICE	LU
xc7k160tfbg676-3	Kintex-7	fbg676	-3	25350	10
xc7k160tfbg676-2L	Kintex-7	fbg676	-2L	25350	10
xc7k160tfbg676-2	Kintex-7	fbg676	-2	25350	10
xc7k160tfbg676-1	Kintex-7	fbg676	-1	25350	10
xc7k160tfbg484-3	Kintex-7	fbg484	-3	25350	10
xc7k160tfbg484-2L	Kintex-7	fbg484	-2L	25350	10
xc7k160tfbg484-2	Kintex-7	fbg484	-2	25350	10

Figure 4: Selecting the target FPGA

detected. Line 36 returns the result of the simulation to the caller. It is best practice to always return a value to the caller of main(). A return value of 0, or false, indicates a successful execution, while a return value of 1, or true designates a failure to the caller.

Listing 1: Testbench for the inner product design.

```
 1
 2  #include <math.h>
 3  #include <iostream>
 4  #include <fstream>
 5  #include "inner_product.h"
 6
 7  int main ()
 8  {
 9      int sim_status = 0;
10
11      int i;
12
13      data_t a[N], b[N], s, s1;
14
15      for (i=0; i<N; i++)
16      {
17          a[i] = rand() % MAX_NUM;
18          b[i] = rand() % MAX_NUM;
19      }
```


New Vitis HLS Project

Solution Configuration

Create Vitis HLS solution for selected technology

Solution Name: solution1

Clock

Period: 4 Uncertainty:

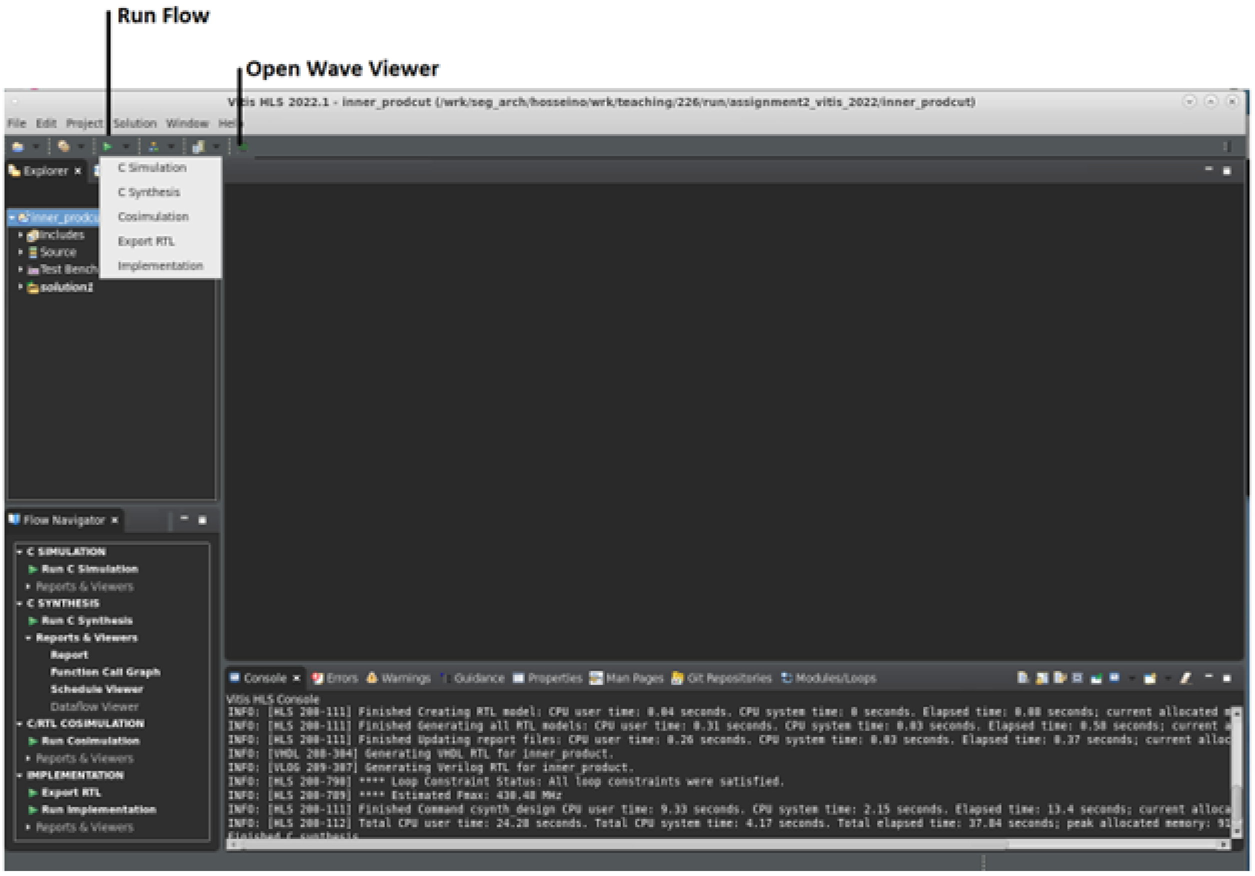
Part Selection

Part: *xc7k160tfbg676-2*

Flow Target

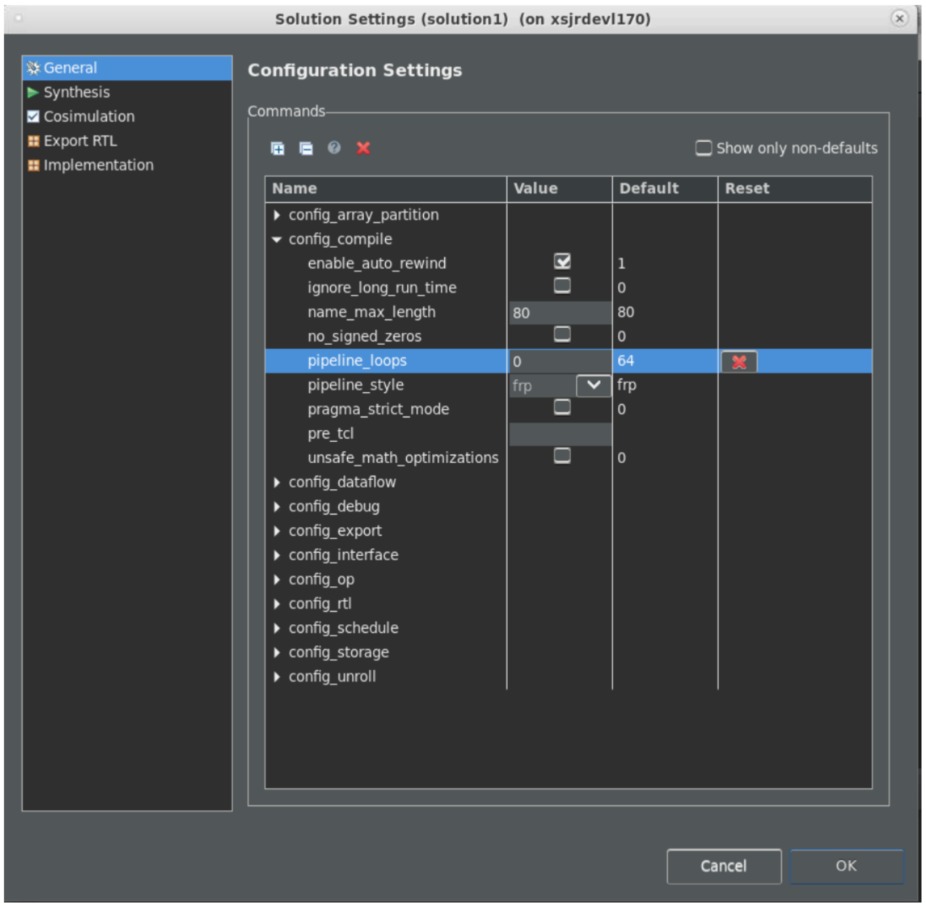
Vivado IP Flow Target Configure several options for the selected flow target

Figure 5: The final page in the project configuration setup



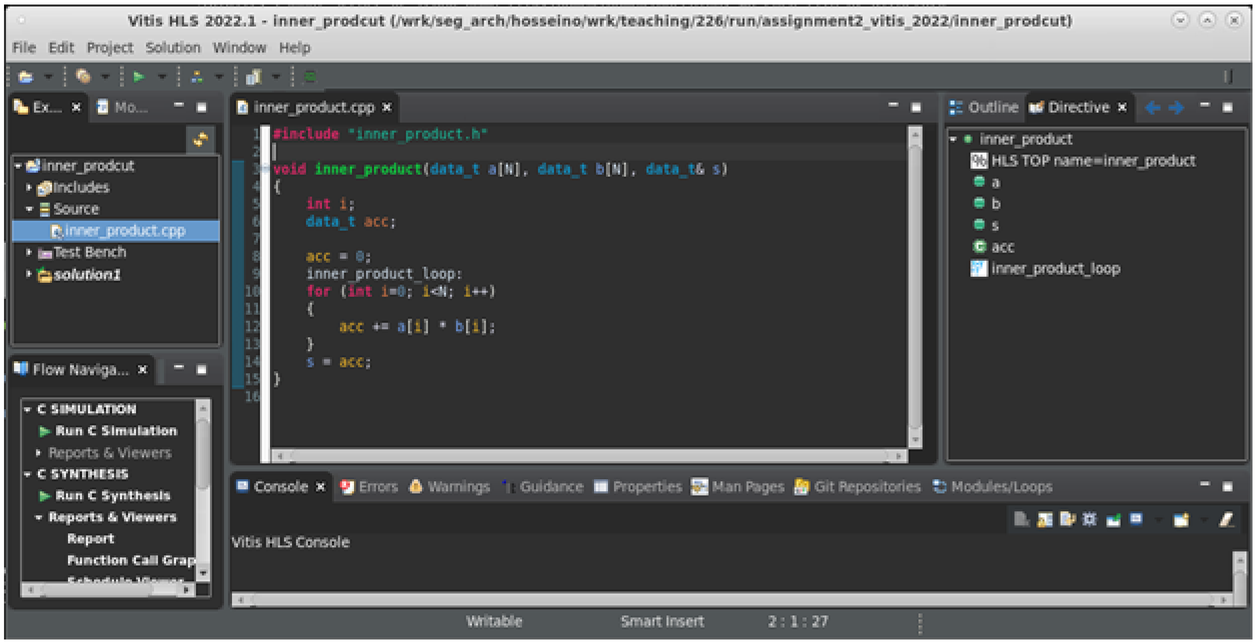
```
Vitis HLS Console
INFO: [HLS 200-111] Finished Creating RTL model: CPU user time: 0.04 seconds. CPU system time: 0 seconds. Elapsed time: 0.08 seconds; current allocated m
INFO: [HLS 200-111] Finished Generating all RTL models: CPU user time: 0.31 seconds. CPU system time: 0.03 seconds. Elapsed time: 0.58 seconds; current a
INFO: [HLS 200-111] Finished Updating report files: CPU user time: 0.26 seconds. CPU system time: 0.03 seconds. Elapsed time: 0.37 seconds; current alloc
INFO: [VHDL 208-304] Generating VHDL RTL for inner_product.
INFO: [VLOG 209-307] Generating Verilog RTL for inner_product.
INFO: [HLS 200-790] **** Loop Constraint Status: All loop constraints were satisfied.
INFO: [HLS 200-789] **** Estimated Fmax: 430.48 MHz
INFO: [HLS 200-111] Finished Command csynth_design CPU user time: 9.33 seconds. CPU system time: 2.15 seconds. Elapsed time: 13.4 seconds; current alloca
INFO: [HLS 200-112] Total CPU user time: 24.28 seconds. Total CPU system time: 4.17 seconds. Total elapsed time: 37.04 seconds; peak allocated memory: 91
Finished C synthesis
```

Figure 6: Vitis HLS IDE



Name	Value	Default	Reset
config_array_partition			
config_compile			
enable_auto_rewind	☑	1	
ignore_long_run_time	☐	0	
name_max_length	80	80	
no_signed_zeros	☐	0	
pipeline_loops	0	64	✗
pipeline_style	frp	frp	
pragma_strict_mode	☐	0	
pre_tcl			
unsafe_math_optimizations	☐	0	
config_dataflow			
config_debug			
config_export			
config_interface			
config_op			
config_rtl			
config_schedule			
config_storage			
config_unroll			

Figure 7: Solution Settings - Pipeline setting



```
#include "inner_product.h"

void inner_product(data_t a[N], data_t b[N], data_t& s)
{
    int i;
    data_t acc;

    acc = 0;
    inner_product_loop:
    for (int i=0; i<N; i++)
    {
        acc += a[i] * b[i];
    }
    s = acc;
}
```

Figure 8: Opening the C++ source file *inner_product.cpp*.

```

    inner_product_reference(a, b, s);
    inner_product(a, b, s1);

    std::cout << s << '\t' << s1 << std::endl;

    if (s != s1)
        sim_status = 1;

        // clean-up

    if (sim_status==0)
        std::cout << "simulation passed" << std::endl;
    else
        std::cout << "simulation failed" << std::endl;

    return(sim_status);

}

void inner_product_reference(data_t a[N], data_t b[N], data_t &s)
{
    s = 0;
    for (int i=0; i<N; i++)
        s += a[i] * b[i];
}
```

Listing 2: Header file for inner product design project.

```
#ifndef __INNER_PRODUCT_H__
#define __INNER_PRODUCT_H__
#include "ap_fixed.h"
#define N 8
#define MAX_NUM 100

typedef int data_t;

// function prototypes
void inner_product(data_t a[N], data_t b[N], data_t &s);
void inner_product_reference(data_t a[N], data_t b[N], data_t &s);

#endif
```

Listing 3: C source code expressing the required inner product calculation.

```
#include "inner_product.h"

void inner_product(data_t a[N], data_t b[N], data_t& s)
{
 int i;
 data_t acc;

 acc = 0;
 inner_product_loop:
 for (int i=0; i<N; i++)
 {
  acc += a[i] * b[i];
 }
 s = acc;
}
```

Listing 4: HLS Console window pane reporting the steps invoked when running a C simulation.

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../inner_product_source_files_v1/
       tb_inner_product.cpp in debug mode
   Compiling ../../../../inner_product_source_files_v1/
       inner_product.cpp in debug mode
   Generating csim.exe
29111 29111
simulation passed
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

The header file *inner_product.h*, Listing 2 contains the definitions for the data types used in the design and also the length of the vectors a and b.

Listing 3 is the C code for the design function *inner_product()*, again this is the compute kernel that is to ultimately be realized in the FPGA. The testbench, Listing 1, is purely for design verification and is not compiled to hardware.

Simulate the design by clicking on the Run C Simulation button in the IDE Run Flow drop-down, refer to Figure 6. The dialog window shown in Figure 9

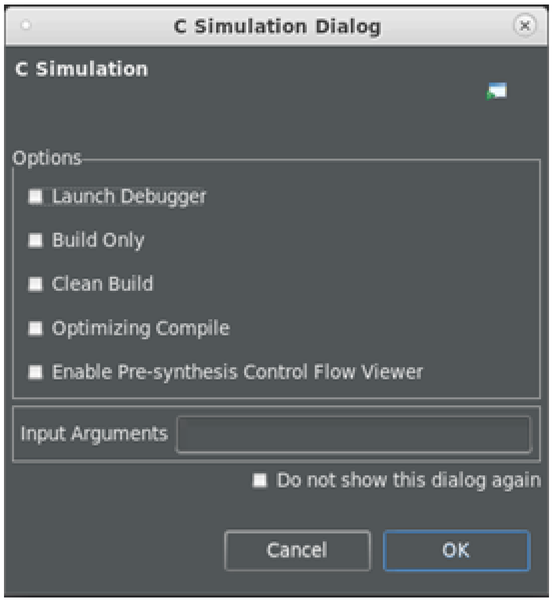


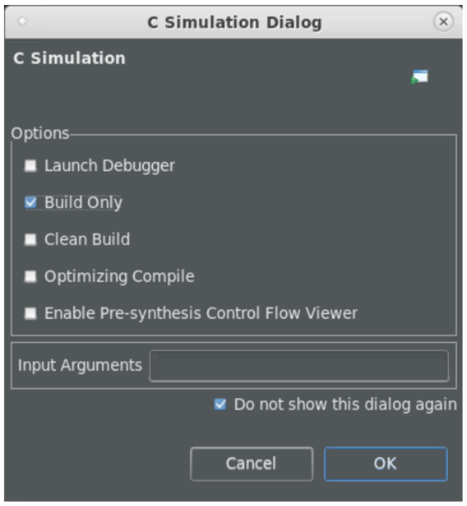
Figure 9: This dialog window is opened when first running C simulation from withing the HLS IDE.

will open. Referring to Figure 10, check the Clean Build and Do not show this dialog box again checkboxes and then click OK to start the simulation. Figure 11 shows the simulation dialog in the Console window. For clarity, the contents of the Console pane are shown in Listing 4. Read through the summary and note the points in the flow where the various project files are opened, the target clock period and FPGA device for the design reported, and in the final part of the dialog confirmation of the final result (pass or fail) is reported.

After a successful C simulation C synthesis can be invoked (refer to Figure 6 for the icon in the IDE Run Flow drop-down to select) to generate the RTL (Register Level Transfer Logic) for the design. Figures 12 to ?? show the synthesis report for the design.

Figure 12 shows the timing and latency summary for the design. The target clock frequency was defined as 4 ns, high-level synthesis is reporting that the estimated clock frequency of the final design is 2.323 ns, which meets the timing goal with a margin of 4 2.323 = 1.08 ns. Note that this is an estimate, the final operating frequency of the design can only be conclusively known once the design is run through place-and-route and timing analysis performed in this final design. Figure 12 also summarizes the design resource utilization. Figure 13 shows the block interface definition. This is the design hardware interface for this component.

Selecting the Open Viewer tab from the menu ribbon reveals the design perspective shown in Figure 14. As we can see Iteration Interval (II) for the


C Simulation Dialog

C Simulation

Options
- [ ] Launch Debugger
- [✓] Build Only
- [ ] Clean Build
- [ ] Optimizing Compile
- [ ] Enable Pre-synthesis Control Flow Viewer

Input Arguments

[✓] Do not show this dialog again

Figure 10: C simulation dialog window with several simulation options selected.

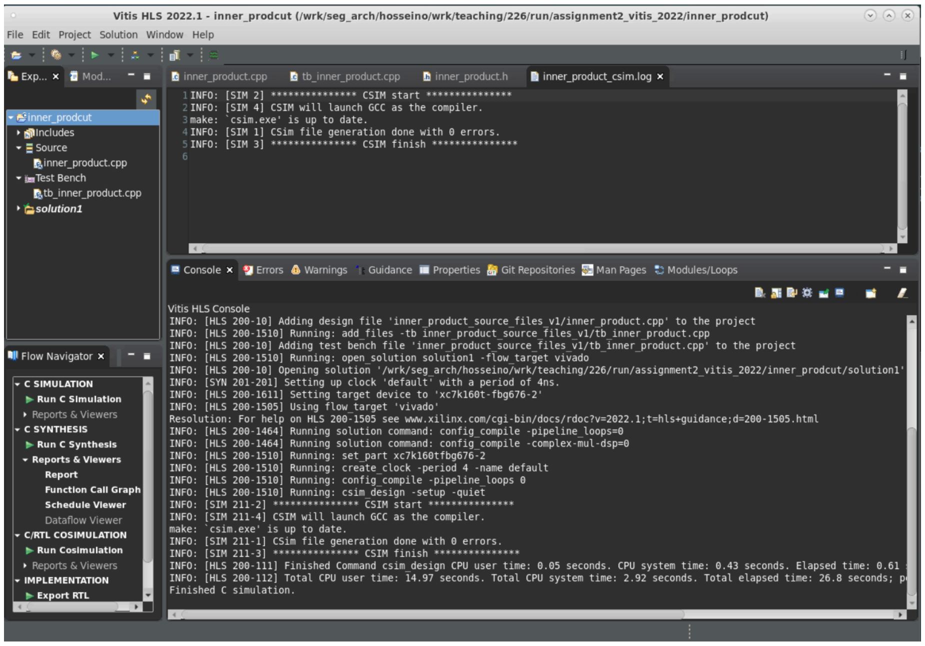
design is II = 5 clock cycles and that the trip count is N-trip = 8. The iteration latency is the latency, specified in clock cycles, for completing one iteration of the main compute loop, *inner_product_loop* in Listing 3. The trip count is the number of times the loop is executed, which is equal to the length of the the vectors **a** and **b**. Recall that this example was compiled with $N = 8$. Since this loop is not pipelined, iteration i + 1 does note begin until iteration i has completed. Therefore, the latency of the compute loop is

$$L_{compute} = II \times N_{trip} = 5 \times 8 = 40 \tag{2}$$

Referring again to the Performance Profile, observe that the function level latency is Nfunc = 41 clock cycles, which means that it takes 41 clock cycles to deliver the result of the calculation to memory. 40 clock cycles correspond to compute and then an additional cycle to write the result to memory. The Initiation Interval for the function is 42 clock cycles. This defines the minimum time interval between successive calls to the function, or the time between consuming successive function input operands. In this case one N = 8 length inner product can be performed at the rate of one vector operation per 42 clock cycles. The majority of this time is composed of the compute latency $L_{compute} = 40$.

For a design with a single Multiply-Accumulator (MAC), the theoretical lower bound for computing the length N inner product is N clock cycles. Or expressed another way, one vector operation can be computed every N clock cycles. This of course is supported with a new MAC operation every clock cycle, corresponding to II=1. In the next section of the assignment, we will see how to provide architectural design directives to realize a time-optimal design with II=1.

C synthesis generates an RTL representation of the design. The next step



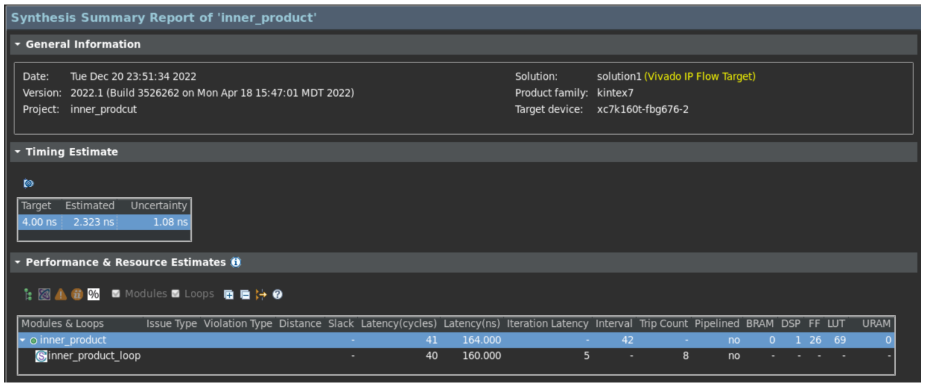
```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
make: `csim.exe' is up to date.
INFO: [SIM 1] CSim file generation done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

```
Vitis HLS Console
INFO: [HLS 200-10] Adding design file 'inner_product_source_files_v1/inner_product.cpp' to the project
INFO: [HLS 200-1510] Running: add_files -tb inner_product_source_files_v1/tb_inner_product.cpp
INFO: [HLS 200-10] Adding test bench file 'inner_product_source_files_v1/tb_inner_product.cpp' to the project
INFO: [HLS 200-1510] Running: open_solution solution1 -flow_target vivado
INFO: [HLS 200-10] Opening solution '/wrk/seg_arch/hosseino/wrk/teaching/226/run/assignment2_vitis_2022/inner_prodcut/solution1'
INFO: [SYN 201-201] Setting up clock 'default' with a period of 4ns.
INFO: [HLS 200-1611] Setting target device to 'xc7k160t-fbg676-2'
INFO: [HLS 200-1505] Using flow_target 'vivado'
Resolution: For help on HLS 200-1505 see www.xilinx.com/cgi-bin/docs/rdoc?v=2022.1;t=hls+guidance;d=200-1505.html
INFO: [HLS 200-1464] Running solution command: config_compile -pipeline_loops=0
INFO: [HLS 200-1464] Running solution command: config_compile -complex-mul-dsp=0
INFO: [HLS 200-1510] Running: set_part xc7k160tfbg676-2
INFO: [HLS 200-1510] Running: create_clock -period 4 -name default
INFO: [HLS 200-1510] Running: config_compile -pipeline_loops 0
INFO: [HLS 200-1510] Running: csim_design -setup -quiet
INFO: [SIM 211-2] *************** CSIM start ***************
INFO: [SIM 211-4] CSIM will launch GCC as the compiler.
make: `csim.exe' is up to date.
INFO: [SIM 211-1] CSim file generation done with 0 errors.
INFO: [SIM 211-3] *************** CSIM finish ***************
INFO: [HLS 200-111] Finished Command csim_design CPU user time: 0.05 seconds. CPU system time: 0.43 seconds. Elapsed time: 0.61
INFO: [HLS 200-112] Total CPU user time: 14.97 seconds. Total CPU system time: 2.92 seconds. Total elapsed time: 26.8 seconds; p
Finished C simulation.
```

Figure 11: The HLS Console panel (the bottom pane in the Figure) provides status information on the various steps involved in the simulation and reports the simulation completion status, in this case the pane is indicating that the simulation Csim was completed with 0 errors

in the design flow is to validate the functional correctness of the RTL with the original generative C code. Theoretically, since the RTL is produced from the C code it should be correct by construction, and most of the time that is the case. However, it is possible, through the specification of certain pragmas or implementation directives, to produce a design that is not functionally correct. This can occur if the programmer makes incorrect assumptions about, for example, memory access behavior of the design, and then, these design directives, provide incorrect guidance to the C synthesis flow so causing synthesis to generate incorrect RTL. There is also always the possibility, and of course this is true of any software, that the C synthesizer and related tools has bugs, and that this could be the cause of an error in the C synthesis process. So in general, it is considered good practice to cross verify the RTL with the C source code. This step is referred to as Run C/RTL Cosimulation in the Vitis HLS flow, refer to Figure 6.

Starting the cosimulation process will open the window shown in Figure 15. Select the Vivado XSIM from the pull-down sub-menu in the Verilog/VHDL Simulator section. Select the **all** option from the Dump Trace pull-down. This will make all of the signals in the design available to the waveform viewer that we will look at next. All other options can be left at their default values. Now click


Synthesis Summary Report of 'inner_product'

General Information

Date:	Tue Dec 20 23:51:34 2022	Solution:	solution1 (Vivado IP Flow Target)
Version:	2022.1 (Build 3526262 on Mon Apr 18 15:47:01 MDT 2022)	Product family:	kintex7
Project:	inner_prodcut	Target device:	xc7k160t-fbg676-2

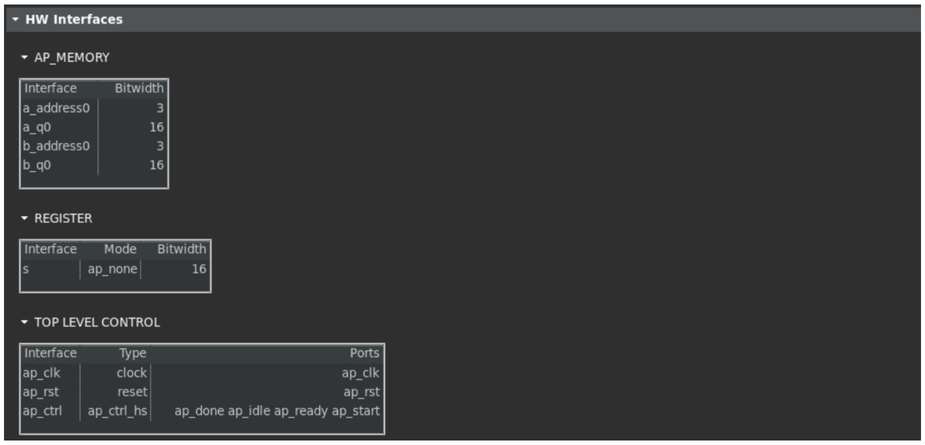
Timing Estimate

Target	Estimated	Uncertainty
4.00 ns	2.323 ns	1.08 ns

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
inner_product				-	41	164.000	-	42	-	no	0	1	26	69	0
inner_product_loop				-	40	160.000	5	-	8	no	-	-	-	-	-

Figure 12: C synthesis report - top part of IDE Synthesis Report panel.


HW Interfaces

AP_MEMORY

Interface	Bitwidth
a_address0	3
a_q0	16
b_address0	3
b_q0	16

REGISTER

Interface	Mode	Bitwidth
s	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

Figure 13: Hardware Interfaces report

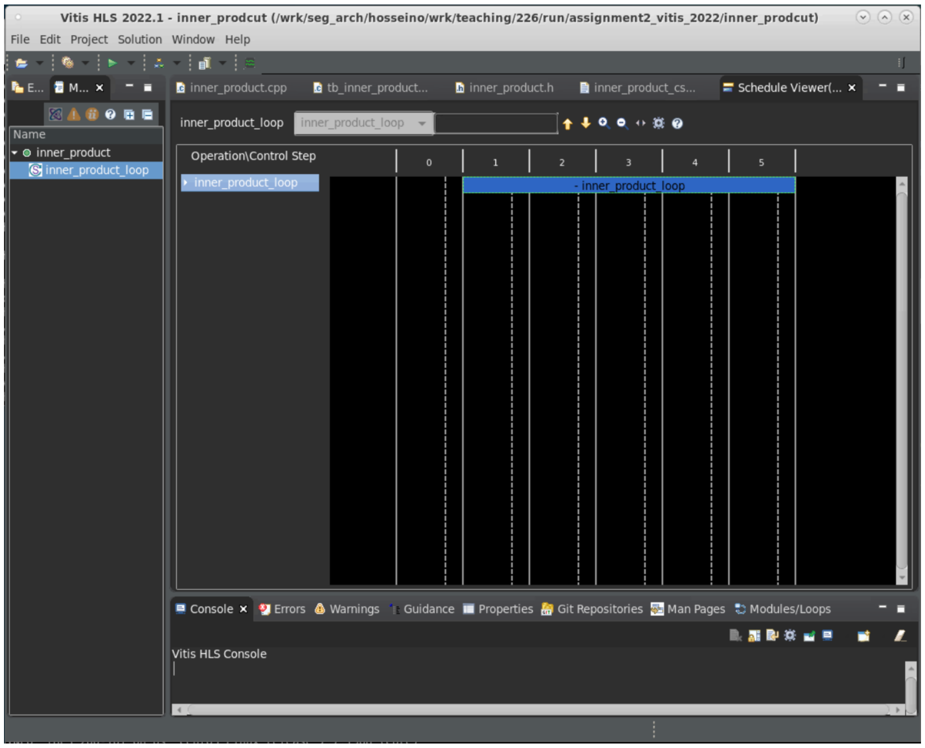
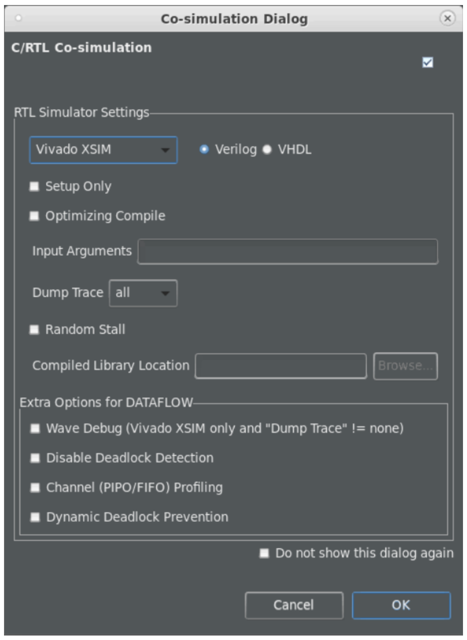


Figure 14: Caption



Co-simulation Dialog

C/RTL Co-simulation

RTL Simulator Settings

Vivado XSIM [✓ Verilog] VHDL

Setup Only

Optimizing Compile

Input Arguments

Dump Trace: all

Random Stall

Compiled Library Location

Extra Options for DATAFLOW

Wave Debug (Vivado XSIM only and "Dump Trace" != none)

Disable Deadlock Detection

Channel (PIPO/FIFO) Profiling

Dynamic Deadlock Prevention

Do not show this dialog again

Figure 15: Running and configuring C/RTL cosimulation.

OK to start cosimulation. Figure 16 shows the console status for a successful cosimulation.

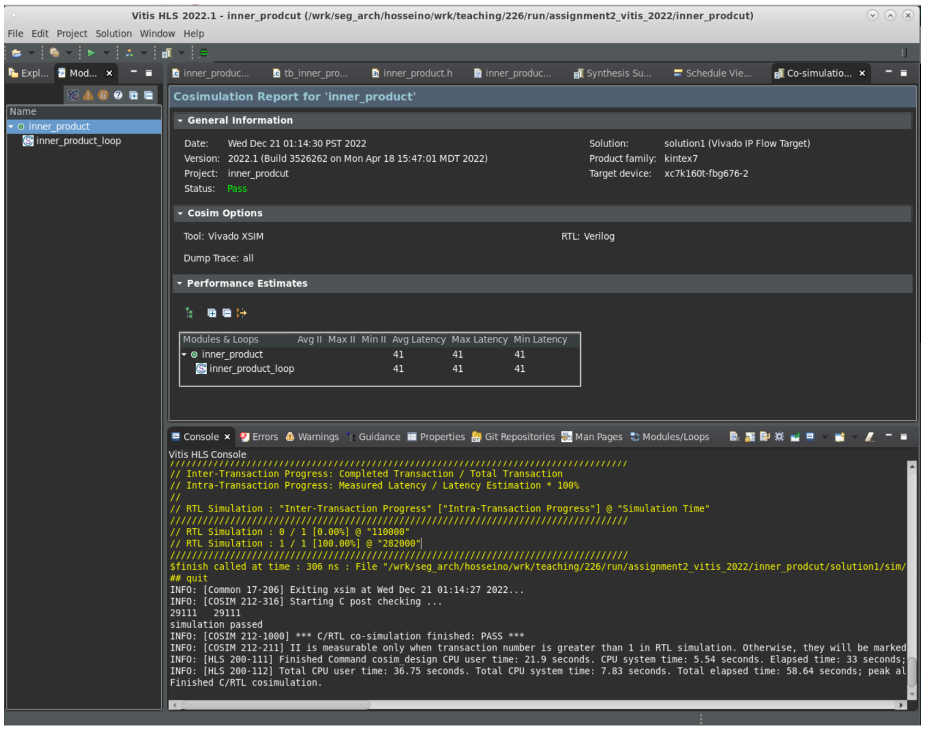
After RTL cosimulation has been completed the RTL simulation database can be viewed in the Vitis Waveform Viewer that is invoked using the Open Wave Viewer in the HLS IDE menu ribbon. The resulting waveform is shown in Figure 17.

In this simulation the input vectors are initialized as a = [83, 77, 93, 86, 49, 62, 90, 63] and b = [86, 15, 35, 92, 21, 27, 59, 26]. The running sum of the point-by-point products that lead to the final inner product is as follows

$$7138, 8293, 11548, 19460, 20489, 22162, 27473, 29111 \tag{3}$$

These sequences can be observed in Figure 17. Also shown is the start signal being asserted and the ap_ready and ap_done signals. The transition of start to true (logic 1) initiates the calculation. We see the memory addresses for the memories storing a and b being generated and associated data values being read from the memories. When ap_done is asserted, as shown in the lower part of the figure, the inner product calculation is available. This happens 41 clock cycles after start is asserted. On the 42nd clock cycle the next inner product calculation begin.

The next step in the flow is to place-and-route (PAR) the design, this step is initiated via Implementation on the menu ribbon, refer to Figure 6. After selecting Implementation the window in Figure 18 is presented. Select the Vi-


Vitis HLS 2022.1 - inner_prodcut (/wrk/seg_arch/hosseino/wrk/teaching/226/run/assignment2_vitis_2022/inner_prodcut)

Cosimulation Report for 'inner_product'

General Information

Date:	Wed Dec 21 01:14:30 PST 2022	Solution:	solution1 (Vivado IP Flow Target)
Version:	2022.1 (Build 3526262 on Mon Apr 18 15:47:01 MDT 2022)	Product family:	kintex7
Project:	inner_prodcut	Target device:	xc7k160t-fbg676-2
Status:	Pass		

Cosim Options

Tool: Vivado XSIM   RTL: Verilog

Dump Trace: all

Performance Estimates

Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
inner_product				41	41	41
inner_product_loop				41	41	41

Vitis HLS Console

```
////////////////////////////////////////////////////////////////////////////////////
// Inter-Transaction Progress: Completed Transaction / Total Transaction
// Intra-Transaction Progress: Measured Latency / Latency Estimation * 100%
//
// RTL Simulation : "Inter-Transaction Progress" ["Intra-Transaction Progress"] @ "Simulation Time"
////////////////////////////////////////////////////////////////////////////////////
// RTL Simulation : 0 / 1 [0.00%] @ "110000"
// RTL Simulation : 1 / 1 [100.00%] @ "282000"
////////////////////////////////////////////////////////////////////////////////////
$finish called at time : 306 ns : File "/wrk/seg_arch/hosseino/wrk/teaching/226/run/assignment2_vitis_2022/inner_prodcut/solution1/sim/
## quit
INFO: [Common 17-206] Exiting xsim at Wed Dec 21 01:14:27 2022...
INFO: [COSIM 212-316] Starting C post checking ...
29111   29111
simulation passed
INFO: [COSIM 212-1000] *** C/RTL co-simulation finished: PASS ***
INFO: [COSIM 212-211] II is measurable only when transaction number is greater than 1 in RTL simulation. Otherwise, they will be marked
INFO: [HLS 200-111] Finished Command cosim_design CPU user time: 21.9 seconds. CPU system time: 5.54 seconds. Elapsed time: 33 seconds;
INFO: [HLS 200-112] Total CPU user time: 36.75 seconds. Total CPU system time: 7.83 seconds. Total elapsed time: 58.64 seconds; peak al
Finished C/RTL cosimulation.
```

Figure 16: C/RTL cosimulation console dialog after succesful completion of the process.

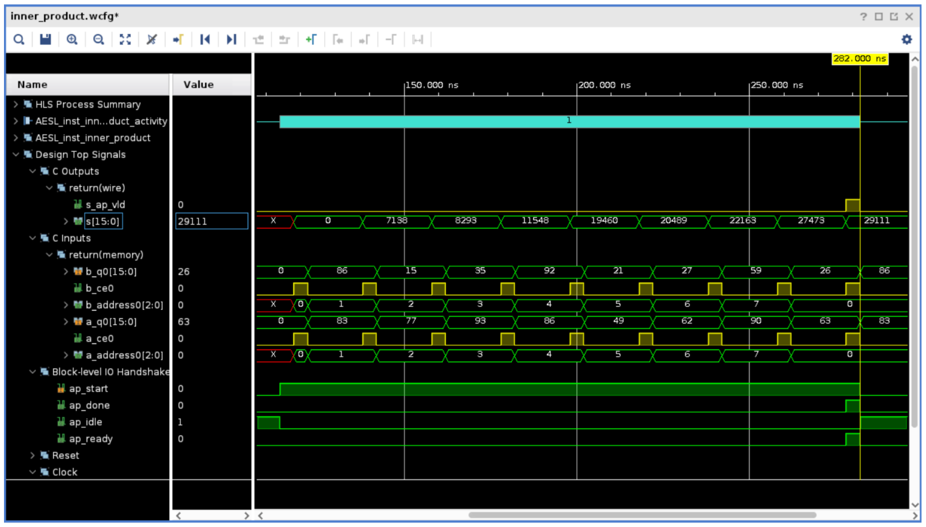


Figure 17: Timing diagram for the N = 8 inner product design. The view presented is generated using the Vivado simulator, automatically invoked by selecting Open Wave Viewer from the HLS IDE.

vado synthesis, synthesis, place route check box and click OK. This will invoke the FPGA physical implementation tool flow, and execute the steps of logic synthesis and place-and-route.

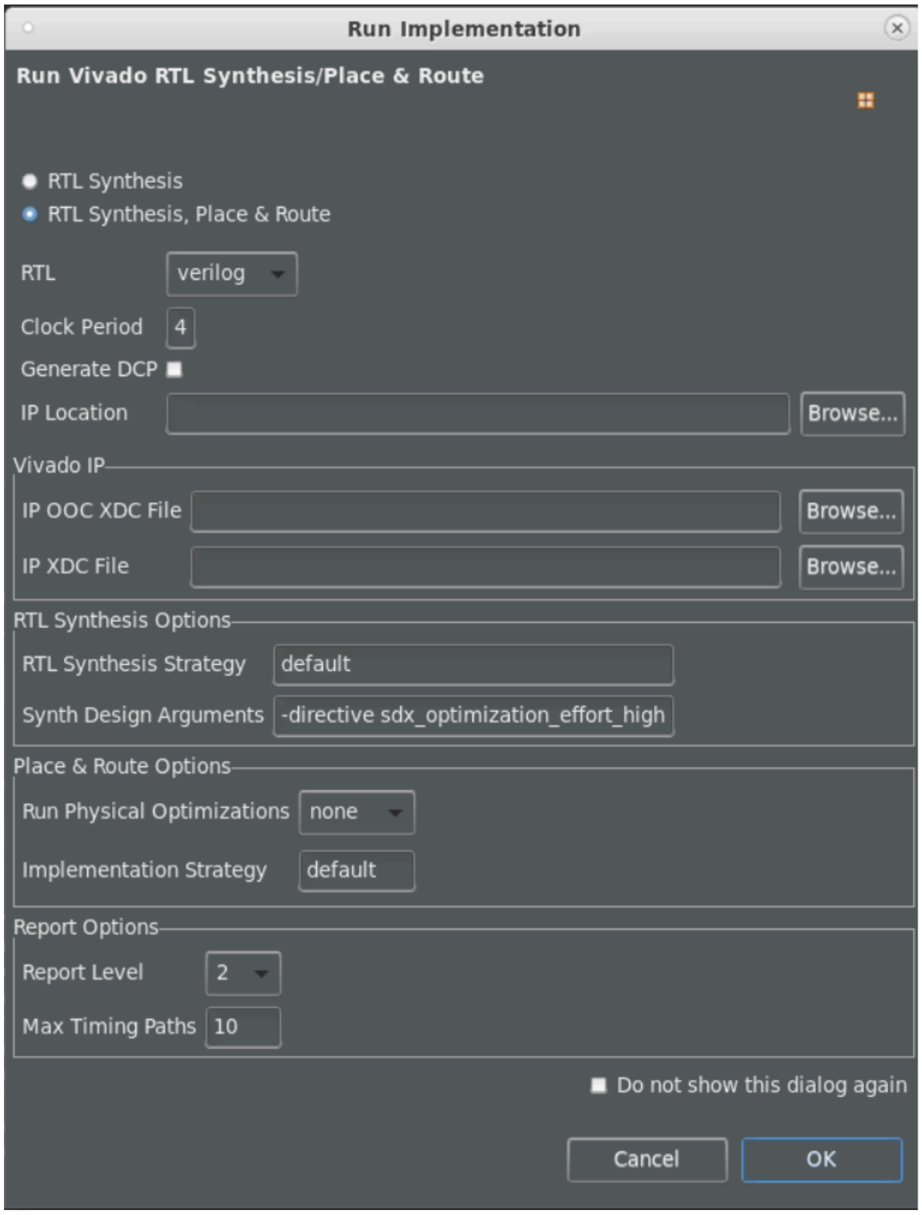
Figure 19 shows the Report and console dialog after completion of the physical implementation flow.

1.2 Loop Pipelining: Solution 2

In this section of the assignment a 2nd design implementation, or *Solution*, will be produced using the same C source code of Section 1.1, but with guidance directives supplied to high-level synthesis to produce a throughput optimized implementation. The goal is to generate a design that employs a single multiplier, just like in Solution 1, but that has a much greater efficiency by engaging the multiplier for useful work on every clock cycle. That is, the loop is to have II=1. This is accomplished by *loop pipeling* in which iteration *i+1* starts on the clock cycle following the start of iteration *i*.

Create a new solution by selecting Project/New Solution from the menu. Leave the default values in the Solution Configuration window, click Finish.

Bring the *Directives* sub-window in to scope as shown in Figure 20. To pipeline the *inner_product_loop* right click on the *inner_product_loop* label in the Directives window to open the directives menu shown in Figure 21. Selection Pipeline from the Directive pull-down . Click ok and the Directives window will look like Figure 22. Note the HLS PIPELINE rewind directive that is attached to the loop *inner_product_loop*. Run C synthesis and review the performance


Run Implementation

Run Vivado RTL Synthesis/Place & Route

○ RTL Synthesis
● RTL Synthesis, Place & Route

RTL	verilog
Clock Period	4
Generate DCP	☐
IP Location	

Vivado IP

IP OOC XDC File	
IP XDC File	

RTL Synthesis Options

RTL Synthesis Strategy	default
Synth Design Arguments	-directive sdx_optimization_effort_high

Place & Route Options

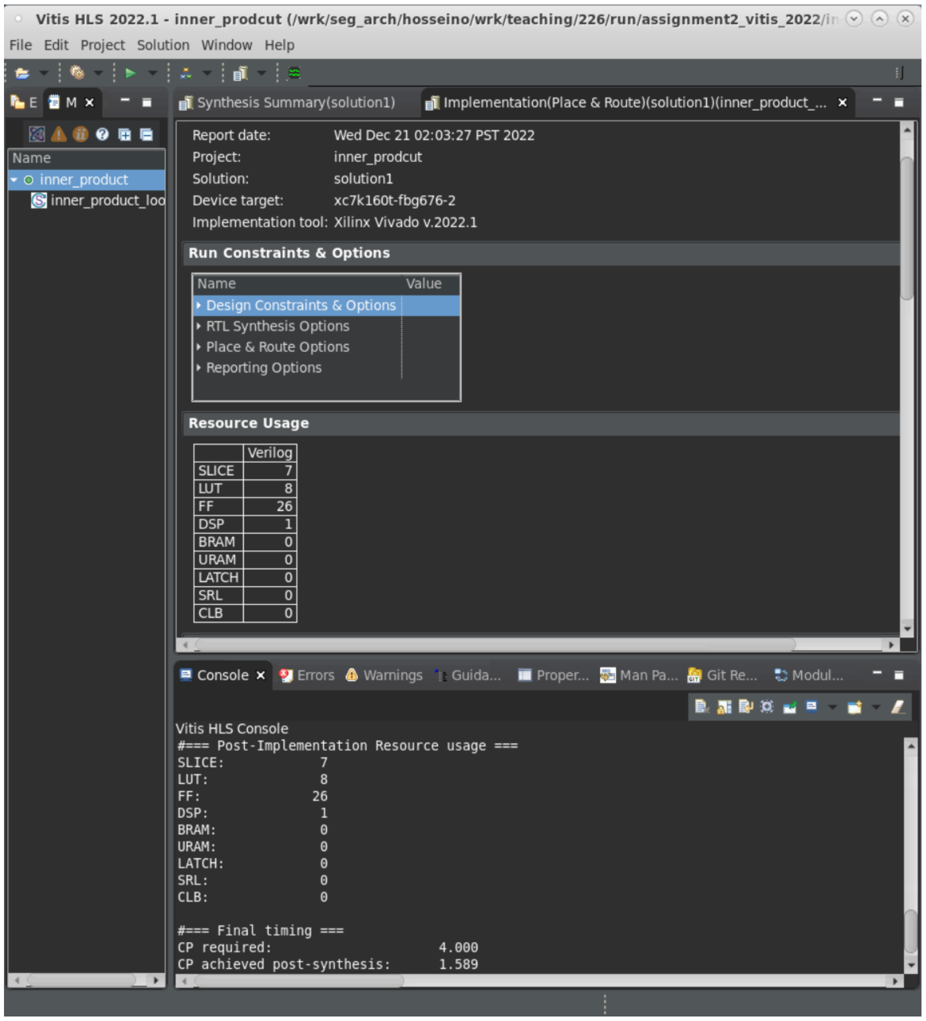
Run Physical Optimizations	none
Implementation Strategy	default

Report Options

Report Level	2
Max Timing Paths	10

☐ Do not show this dialog again

Figure 18: Running implementation (Place-and-Route)


Vitis HLS 2022.1 - inner_prodcut (/wrk/seg_arch/hosseino/wrk/teaching/226/run/assignment2_vitis_2022/i
File Edit Project Solution Window Help
Synthesis Summary(solution1)
Implementation(Place & Route)(solution1)(inner_product_...
Name
inner_product
inner_product_loo
Console Errors Warnings Guida... Proper... Man Pa... Git Re... Modul...

Report date:	Wed Dec 21 02:03:27 PST 2022
Project:	inner_prodcut
Solution:	solution1
Device target:	xc7k160t-fbg676-2
Implementation tool:	Xilinx Vivado v.2022.1

Run Constraints & Options

Name	Value
Design Constraints & Options	
RTL Synthesis Options	
Place & Route Options	
Reporting Options	

Resource Usage

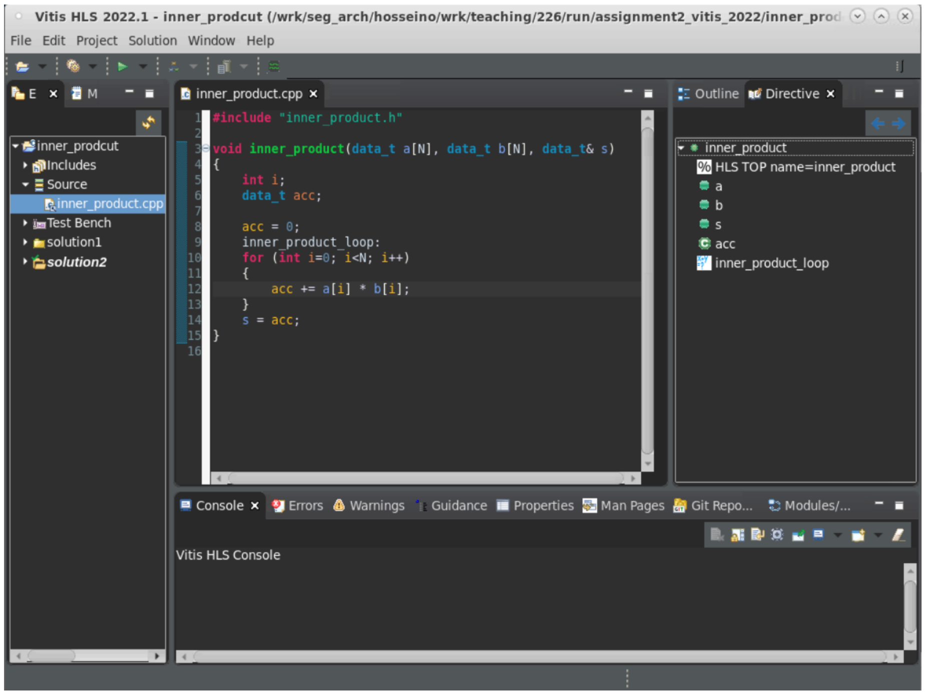
	Verilog
SLICE	7
LUT	8
FF	26
DSP	1
BRAM	0
URAM	0
LATCH	0
SRL	0
CLB	0

Vitis HLS Console

```
#=== Post-Implementation Resource usage ===
SLICE:              7
LUT:                8
FF:                26
DSP:                1
BRAM:               0
URAM:               0
LATCH:              0
SRL:                0
CLB:                0

#=== Final timing ===
CP required:                    4.000
CP achieved post-synthesis:     1.589
```

Figure 19: The Report and Console dialog after completing design Implementation

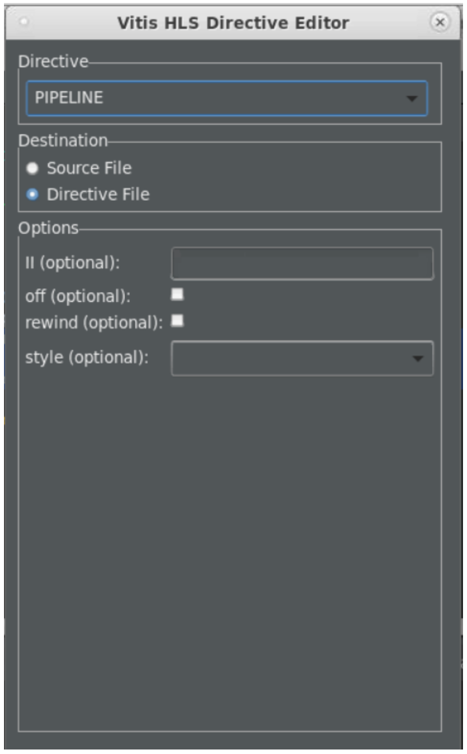

Vitis HLS 2022.1 - inner_prodcut (/wrk/seg_arch/hosseino/wrk/teaching/226/run/assignment2_vitis_2022/inner_prod
File Edit Project Solution Window Help
E M
inner_prodcut
Includes
Source
inner_product.cpp
Test Bench
solution1
solution2
inner_product.cpp
Outline Directive
inner_product
HLS TOP name=inner_product
a
b
s
acc
inner_product_loop
Console Errors Warnings Guidance Properties Man Pages Git Repo... Modules/...
Vitis HLS Console

```
#include "inner_product.h"

void inner_product(data_t a[N], data_t b[N], data_t& s)
{
    int i;
    data_t acc;

    acc = 0;
    inner_product_loop:
    for (int i=0; i<N; i++)
    {
        acc += a[i] * b[i];
    }
    s = acc;
}
```

Figure 20: With the *Directives* panel in scope, implementation directives can be added to the design to generate different performance-cost points in design space.



Vitis HLS Directive Editor

Directive

PIPELINE

Destination

( ) Source File
[✓ Directive File]

Options

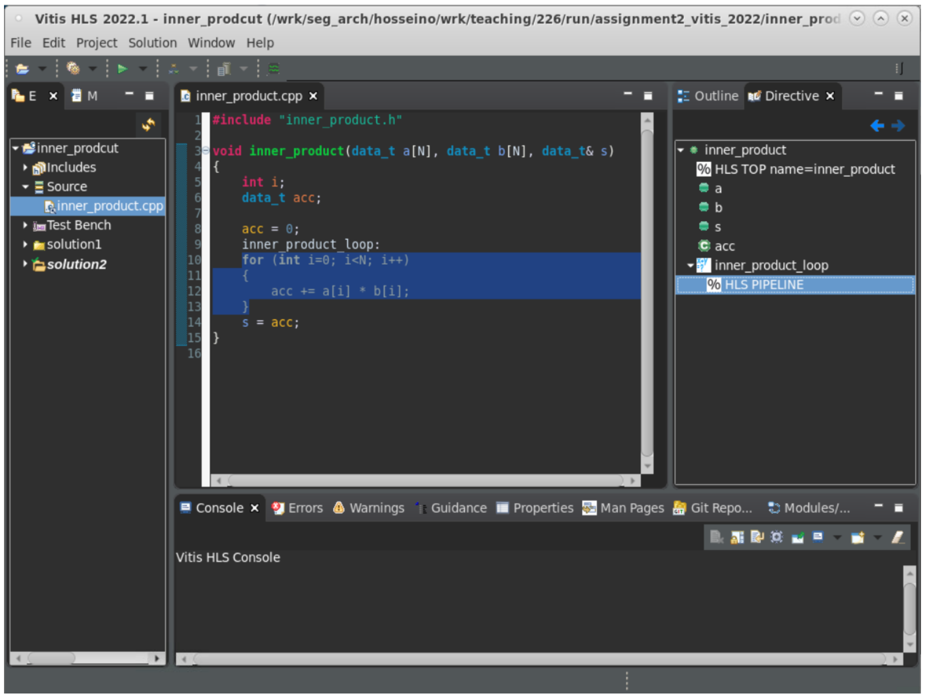
II (optional):

off (optional): ☐

rewind (optional): ☐

style (optional):

Figure 21: The *Directives* menu.



```
#include "inner_product.h"

void inner_product(data_t a[N], data_t b[N], data_t& s)
{
    int i;
    data_t acc;

    acc = 0;
    inner_product_loop:
    for (int i=0; i<N; i++)
    {
        acc += a[i] * b[i];
    }
    s = acc;
}
```

Figure 22: The *Directives* window showing that the *inner_product_loop* and associated pipeline directive.

profile in the Analysis context. Note that the loop has an II=1 and the function has II=14. With a loop II of 1 a new multiplication is initiated on every clock cycle. The function initiation interval of 8 indicates that the function can be invoked every 8 clock cycles, which is a cadence equal to the length of the input vectors in this example.

Contrast the loop II of 1 in this Solution compared to the II of 4 in the first design. In the first design te multiplier was engaged 25% of the time. In Solution 2 the multiplier is operating at 100% efficiency. For an FPGA clock frequency of f_{FPGA} MHz, the 2nd design is realizing f_{FPGA} Multiply-Accumulates per second (MACs) compared to the $f_{FPGA}/4$ MACs of Solution 1.

1.3 Function Pipeline: Solution 3

The goal of this section of the assignment is to perform further optimizations to implement a design that supports function level pipeline with an II of 1. This means that one complete inner product calculation can be started on consecutive clock cycles.

First create a new solution, *Solution3* in the project workspace. Uncheck the Copy directives and constraints from solution checkbox so that any directives previously defined for a previous solution will not be propagated to this solution. New directives will be specified for this implementation. With the source code inner_product.cpp window open, access the Directives window. Right click on the top-level function name *inner_product*, select Insert Directive and following steps similar to those in Section 1.2, add a PIPELINE directive to the function. Run C synthesis and then look at the Performance Profile in the Analysis view. The result should show a function initiation interval of 4. Meaning the function can be invoked every 4 clock cycles. This is a factor of two improvement over the function initiation interval of 8 in Solution2, but not the initiation interval of 1 that is the design objective for this solution. What is going on?

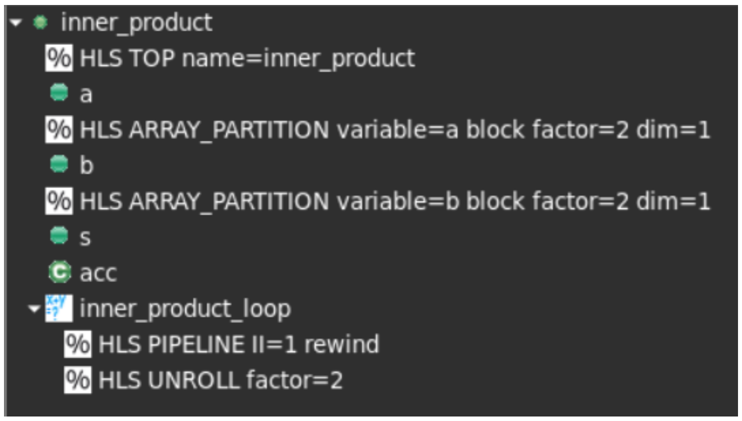
Scroll through the dialog in the console window and you should see the warning.

Try to figure out what caused this warning and timing violation?

Explore achieving funtion II of 1 by adding memory partitioning directives to each of the a and b and function pipleining.

1.4 Loop Pipelining with Partial Unrolling : Solution 4

So far two useful designs have been produced from a common C code base . The fully pipelined single multiplier sequential inner product circuit in Solution2 and the 8-multiplier fully parallel and task-level pipelined design of Solution3. These two designs represent two extremes in design space. There are intermediate design points that offer alternative tradeoffs between implementation cost and performance as measured in terms of vector operations per second. In this section of the assignment an implementation that is positioned, performance-wise, in between the extremes of the sequential and parallel designs will be generated. This design will use 2-multipliers. Create a new solution, Solution4. The



```
inner_product
  HLS TOP name=inner_product
  a
  HLS ARRAY_PARTITION variable=a block factor=2 dim=1
  b
  HLS ARRAY_PARTITION variable=b block factor=2 dim=1
  s
  acc
  inner_product_loop
    HLS PIPELINE II=1 rewind
    HLS UNROLL factor=2
```

Figure 23: Inner product compute loop pipelined and unrolled by a factor of 2 to generate a design resourced with 2 multipliers and performing 2 multiplications on each clock cycle.

main compute loop *inner_product_loop* is to be pipelined and partially unrolled. Add the directives to the design as shown in Figure 23. This unrolls the loop by a factor of 2 and generates an implementation that is resourced with two multipliers. Since the loop is also pipelined, this means that 2 multiplications are computed on every clock cycle. One inner product is computer every N/2 clock cycles. The memories must also be partitioned in order to supply the appropriate number of read ports to supply the required number of operands to the multipliers. In this case each multiplier requires an operand from the a and b memories, so each memory must have 2 ports. Look at the interface description to confirm this is the case.

Run C/RTL Cosimulation to confirm that the RTL generated with this set of directives is funtionally correct.

1.5 Loop Pipelining with Partial Unrolling : Solution 5

Make a new solution and create a 4-multiplier design and that executes 4 multiplications on every clock cycle.

Make sure to run C/RTL Cosimulation to confirm that the RTL generated with this set of directives is functionally correct.

2 Assignment Submission

Your assignment submission is to consist of the following components, the submission of the archived project and a small project report which is to include the items listed below.

Use word, or other software of your choice, to generate your report. Don't simply print this page and send me a image of the tables completed by hand,

Table 1: Resource footprint for the 5 implementations of the inner product function.

	solution1	solution2	solution3	solution4	solution5
BRAM					
DSP48					
FF					
LUT					

Table 2: The compute rates of the 5 inner product architectures.

	solution1	solution2	solution3	solution4	solution5
inner_product_loop() pipelined	no	yes	-	yes	yes
inner_product() function pipelined	no	no	yes	no	no
inner_product_loop(); Partially Unrolled	no	no	no	yes, by 2	yes, by 4
MPYs/Sec.					
Vector Ops/Sec.					

take the time to prepare a small report, no hand written work please.

1. Use the Vitis HLS project archive capability to archive and then email your archive to me. A project is archived by selecting, from the HLS IDE ribbon menu, File / Archive Project. Uncheck the Active Solution Only checkbox to ensure that all of the project solutions are added to the archive and check the Include Run Results checkbox so that all of the synthesis and implementation results are included in the archive. Save your archive file using the naming convention first_name_lastname_elen226_a1. The archive .zip file is saved in the project directory.

2. For the 5 solutions you generated in this assignment complete the following table and include it in your project report. Take the time to generate a report and build your own version of the tables below.

3. For the 5 solutions you generated complete the following table and include it in your project report. Use the LUT, FF and DSP48 resource count from the place-and-route runs rather than the estimates generated from C Synthesis. To complete the entries for the final two lines in the table, $MPYs/Sec.$ and Vector $Ops/Sec.$, assume that the FPGA clock frequency is 250 MHz.