

# **ZYBO Video Workshop**

Paris, FRANCE

23.03.2017



# 1 Theoretical background

Software is everywhere. The flexibility it offers to designers allows it to be used in a multitude of applications. Many consumer, industrial or military products are either running software or began as a software model or prototype executing on a generic circuit, or processor. Decades of advances in software engineering resulted in ever higher abstractions, ever smarter tools, ever increasing number of automatic optimizations that improve code re-use, shorten design time and increase performance. Continuous performance increase quantified by the number of instructions executed per second has been driven at first by the increase in processing frequencies, then by parallelization of algorithms and simultaneous execution of tasks by multiple processing cores.

The ubiquitous nature of software lead to most of the engineering problems to be approached with software solutions at first. Depending on the application a software-only approach might not meet the requirements, be those latency, throughput, power or other. An expensive option would be handing the algorithm over to a hardware engineer for a custom circuit implementation. The entry cost of application-specific integrated circuit (ASIC) design is still high despite advancements in fabrication technologies. Depending on the product forecasts, and ASIC design might not be viable economically.

Bridging the gap between generic processor circuits and ASICs are FPGAs, allowing the use of blank reprogrammable hardware logic elements to implements a custom circuit. It offers a lower barrier of entry to power savings and performance benefits of fabrication technologies without the cost of ASIC. Also, an algorithm optimized for FPGA implementation benefits from the inherently parallel nature of custom circuits.

#### 2 Hardware

The Digilent Zybo development board is well-suited for prototyping an algorithm running in software at first and then off-loading sub-tasks for processing to custom circuits. It is based on a Xilinx Zynq 7010 SoC, a hybrid between a dual-core ARM A9 (processing system, PS) and Artix-7 based FPGA (programmable logic, PL). Low-latency, high-throughput coupling between PS and PL allows for software implementation, where design-time is more important than performance, and hardware, where performance is critical.

The programming model for software usually makes use of programming languages that abstract from hardware particularities. While this offers increased portability and ways to apply automatic compiler optimizations, avoiding knowledge about the underlying hardware is not possible anymore close to the performance limits.

FPGA design can make use of two different programming models. One is RTL description in VHDL/Verilog, the other is high level synthesis in C/C++. High level synthesis represents a somewhat similar programming model to software programming. However, for a worthwhile improvement over software implementation of the same algorithm, one needs to have a good understanding of the underlying hardware architecture. Much more so than for software in general.

#### 2.1 FPGA Architecture

Field programmable gate array (FPGA) is a large array of configurable logic blocks (CLB), interconnect wires and input/output (I/O) pads. The CLB is made up of look-up tables (LUT) and flip-flops (FF), in



varying numbers depending on the exact FPGA architecture. This structure is generic enough to implement any algorithm. During programming the LUTs are programmed to implement a certain logic function, and FFs to pipeline the data flow synchronous to a clock signal. Interconnect is also programmed to wire LUTs, FFs, input pads and output pads together resulting in a custom hardware circuit implementing a certain algorithm.

Current FPGA architecture also include hard primitive blocks that specialize a certain function that would otherwise be too costly in terms of logic utilization or too slow in terms of throughput to implement in generic logic. For example, digital signal processing (DSP) blocks are available to implement a multiply-accumulate circuit with no generic logic utilization. These blocks are optimized enough to offer superior performance for the specific task. Another example is dual-port static RAM (BRAM), that offers higher capacities than RAM implemented in LUTs. These primitive blocks are by default automatically inferred for certain HDL constructs like the multiply operator (\*) for DSP or array access for BlockRAM.

LUT is a memory element that implements a truth-table. Depending on the exact architecture, each LUT has a number of inputs that address a location in the truth-table. The value stored at that address is the output of the function implemented. During programming the truth-table is populated to implement the desired function. It can also be thought of and used as a 2<sup>N</sup>-memories, called distributed RAM. It is a fast memory type because it can be instantiated all over the FPGA fabric, local to the circuit that needs data from it.

FF is a storage element that latches new data on its input when clock and clock enable conditions are true and permanently provides the stored data on its output.

BRAM is a dual-port RAM that stores a larger set of data. It holds 18Kb or 36Kb and can be addressed independently over two ports for both read and write. In essence, two memory locations can be accessed simultaneously in the same clock cycle.

#### 2.2 Parallelism and program execution

A processor core executes software instructions in a sequence. Higher-level programming languages translate language statements into assembly instructions that perform the function. Under this abstraction, the addition of two variables usually involves more than one instruction. Apart from the actual arithmetic operation that accesses internal registers, memory load and store instructions will be needed. Performance improvements result in optimizing those memory accesses using caches. Each memory level trades access latency for storage capacity, so less and less data is available at memories of lower latencies. The job of the programmer and compiler is to ensure that for critical areas of an algorithm the spatial locality of data is high and can be accessed with the lowest latency possible.

It requires considerable effort and performance analysis tools to optimize code for execution time.

The FPGA is massively parallel by nature. Every LUT can execute a different function at the same time, so it is possible to have multiple arithmetic logic units (ALU) executing addition operations, for example is parallel. On a processor, the ALU is shared and these would have to be executed sequentially. Memories can be instantiated close to where they are needed, resulting in high instantaneous memory bandwidth.

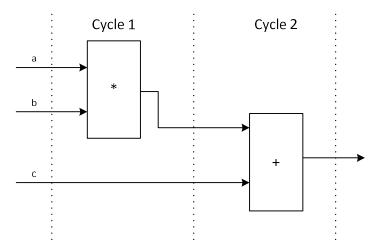


The role of high level synthesis tools is to extract the best possible circuit implementation from a C/C++ code that is functionally correct and meets the requirements. It analyzes data dependencies determining which operations could and should execute in each clock cycle. Depending on the targeted clock frequency and FPGA, some operations might take more cycles to complete. This step is called **scheduling**. Next, the hardware resources are determined that implement the scheduled operation best. This is called **binding**. The last step in the synthesis is the **control logic extraction** which creates a finite state machine that controls when the different operations should execute in the design.

For multi-cycle operations **pipelining** is performed in the scheduling phase. Imagine the following C statement:

x=a\*b+c;

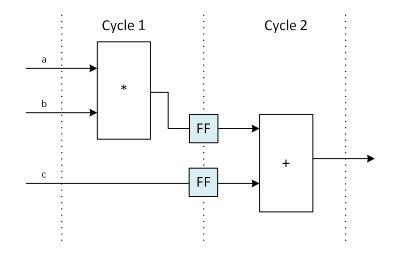
If the clock period is too small for the multiplication and addition to complete in one clock cycle, it will be scheduled for two cycles. For every set of inputs a, b, and c it takes two cycles to obtain the result. It follows that in cycle 2 the multiplier does not perform any operation; it only provides the result calculated in the previous cycle.



This inefficiency becomes more apparent, when this statement is executed in a loop, ie. the circuit processes more than one set of input data.

If there was a storage element between cycles, the result from cycle 1 would be saved, and the multiplier would be free to perform a calculation for the next set of inputs. This concept is called pipelining and it is a major optimization opportunity increasing the throughput tremendously.





#### 2.3 Performance metrics

The previous example is a great opportunity to introduce some performance metrics definitions. The **latency** of the statement above is two, as it takes two cycles to output the result. In the first non-pipelined case the **initiation interval (II)** is also two, since it takes two cycles for the circuit to accept a new set of inputs. However, in the second pipelined case the II is just one, because the circuit is able to accept a new set of inputs in every cycle, and will output a result in every cycle. The latency is still two, as the result for the first set of inputs will appear after two cycles. If the circuit processes 10 sets of input data, the non-pipelined versions will have a total latency of 20 cycles ((#-1) \* II + latency). The pipelined versions will only take 11 cycles ((#-1) \* II + latency) to provide all the 10 results.

These performance metrics are calculated by the tools for both loops and functions, and are considered the most important feedback mechanism for the designer to evaluate the synthesized hardware circuit.

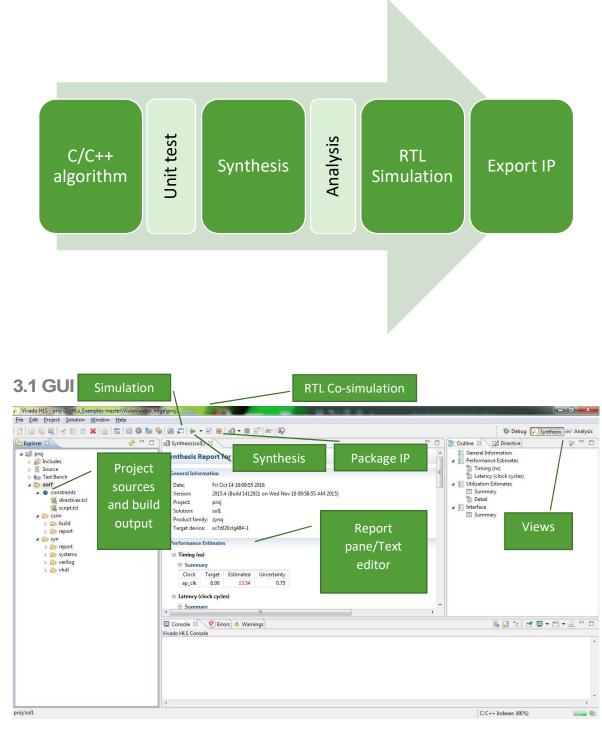
## **3 Vivado HLS**

Xilinx's offering in high-level synthesis is part of the Vivado suite and is called Vivado HLS. The workflow is an iterative approach with simulations as verification steps inserted along the way to make sure the design meets the requirements and is functionally correct right from the initial stages. Vivado HLS can:

- compile, execute and debug the C/C++ algorithm,
- synthesize into RTL implementation,
- provide analysis features,
- generate and execute RTL simulation testbenches,



• export the RTL implementation as an IP module.

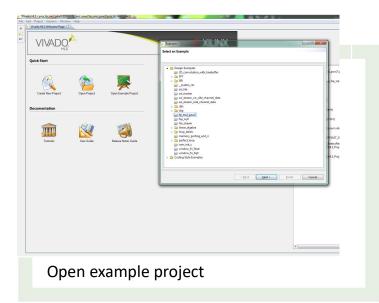


The GUI layout is quite similar to other software IDEs. The project explorer lists the source, include and testbench files. Simulation and synthesis outputs are also visible here grouped into solutions. The workflow action buttons are in the toolbar ordered by their sequence in the workflow. In the upper right corner three layout views are available each fitting the current workflow step.



## 4 Task One – Getting familiar with the interface

Let us open an example project to get more familiar with the interface.



Launch Vivado HLS 2015.4 from the Start Menu.

On Linux run vivado\_hls from the shell.

Click the Open Example Project button on the Welcome Page.

Choose Design Examples/fp\_mul\_pow2 from the list of projects

🔥 Examples			
Specify Example Project Location Specify where to create the example project.		AG	
Project name: [fp_mul_pow2		Browse	Browse to the location of your choice on your local storage drive
			You may choose zybo_workshop/hls_projec for location. Click OK
	Back Next > Finish	Cancel	
Save project			



Explorer      Complete Reproduction	128 double_num_t x_num, prod; 129 130 x num.fp num = x;	Image: Section of the sectio	Open fp_mul_pow2.h below Includes.
C/Xiime/Vivado,HLS/2015.4/mog/     C/Xiime/Vivado,HLS/2015.4/mog/     C/Xiime/Vivado,HLS/2015.4/mog/     C/Xiime/Vivado,HLS/2015.4/mog/     C/Xiime/Vivado,HLS/2015.4/mog/     C/Xiime/Vivado,HLS/2015.4/mig/     D/mig/p_mul_powl/p_mul_pow	<pre>111 stringer 45% Pr_MTULE_DOUDCE_TSTS 16 (r_unabsorp = 0.077   (r_unabsorp = 0.077   (r_unabsorp = 0.077   (r_unabsorp = 0.077   (r_unabsorp = 0.077 - 0.077   (r_unabsorp = 0.077   (r_unabsor</pre>	e over	Open fp_mul_pow2.c below Source.
Girkinfg.mul.pon2.frg.mul.po     Girkinfg.mul.pon2.frg.mul.po     Girkinfg.mul.pon2.frg	10     } cbse       10     prod.sign = x_punction;       10     prod.sign = x_punction;       10     prod.sign = x_punction;       10     prod.sign = x_punction;       101     prod.sign = x_punction;       102     prod.sign = x_punction;       103     cetum prod.fp.run;       104     cetum prod.fp.run;       105     cetum prod.fp.run;       106     cetum prod.fp.run;       107     @ Creact R:		Open fp_mul_pow2_test.c below Test Bench.
> 🍙 report	Vivade HS Centole	* *	Notice that we are in the Synthesis view (upper right corner).

A Vivado HLS project is much like any other C/C++ software project. There is a source file defining two functions, a header file declaring the functions and some data types. There is also a test bench source file, which is a regular application with a main entry point that runs test on the functions, validating them on functional correctness. Test benches are used for C simulation, which is the first validation step in the design process. The successfulness of C simulation is determined by the return value of the test bench. It is expected to return 0 for a success, and any non-zero value for failure.

Discuss the implementation of the double\_mul\_pow2 function and the test bench.

C Simulation Dialog	4
f Options	E E
🕅 Launch Debugger	pi
e 🔄 Build Only	
🔄 🕅 Clean Build	
i 🔲 Optimizing Compi	le
input Arguments	
	🔲 Do not show this dialog box again.
e	OK Cancel
1	UK Cancer

Click the Run C Simulation button on the toolbar.

Leave simulation options at their default values and click OK

Discuss the results of the C simulation and the messages shown in the console.

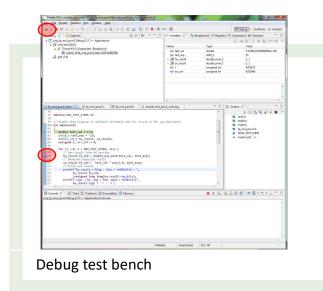


📮 Console 🖾 🔍 👰 Errors 💧 Warnings								<b>=36</b> (6).E	1	z   🛃 🗳 🕶 🙆	2 H
/ivado HLS Console											
Compiling(apcc)///fp mul pow2.c in debug		aba									
<pre>@I [HLS-10] Running 'c:/Xilinx/Vivado HLS/2015.4/bin/</pre>			ed.	wi	64 (	n/a	ncc ev	. '			
for user 'Elod' on host 'elap' (Windows N									v 1	0 14.03.19 +0200 20	16
in directory 'D:/hls/fp mul pow2/fp mul p											10
@I [APCC-3] Tmp directory is apcc db	0	- pro	J'	P-			2, 2014	c romr		in, build	
@I [APCC-1] APCC is done.											
Generating csim.exe											
hw result = 4.57764e-005 : bits = 0x3F08000000000000	:	sign	=	+.	exp	=	-15,	mant	=	0x08000000000000	
hw_result = 5.06868e+250 : bits = 0x73FC515880000000	:	sign	=	+,	exp	=	832,	mant	=	0x0C515880000000	
hw result = -7.0227e-111 : bits = 0xA910E39140000000	:	sign	=	-,	exp	=	-366,	mant	=	0x00E39140000000	
hw_result = 2.682e+205 : bits = 0x6A9562BA20000000		sign	=	+,	exp	=	682,	mant	=	0x0562BA20000000	
hw result = -1.7309e-150 : bits = 0xA0D6A9E380000000	:	sign	=	-,	exp	=	-498,	mant	=	0x06A9E380000000	
hw_result = -1.10749e-225 : bits = 0x913A3C65E0000000	:	sign	=	-,	exp	=	-748,	mant	=	0x0A3C65E0000000	
hw_result = 2.11551e-204 : bits = 0x15A5398680000000	:	sign	=	+,	exp	=	-677,	mant	=	0x05398680000000	
hw_result = 1.82008e-195 : bits = 0x178101B200000000	:	sign	=	+,	exp	=	-647,	mant	=	0x0101B200000000	
hw_result = 3.58909e+261 : bits = 0x763D2DCB40000000	:	sign	=	+,	exp	=	868,	mant	=	0x0D2DCB40000000	
hw_result = 7.93181e-005 : bits = 0x3F14CAF280000000	:	sign	=	+,	exp	=	-14,	mant	=	0x04CAF280000000	
hw_result = -4.16729e-134 : bits = 0xA43E4A1B80000000											
hw_result = 5.09937e+176 : bits = 0x64A01B8680000000	:	sign	=	+,	exp	=	587,	mant	=	0x001B8680000000	
hw_result = 6.96589e-138 : bits = 0x2374BD0560000000		0			10.0						
hw_result = -3.3959e+111 : bits = 0xD71697D580000000											
hw_result = 1.22026e-079 : bits = 0x2F8CEFFDA0000000											
hw_result = -4.15363e-081 : bits = 0xAF3F851E60000000	:	sign	=	-,	exp	=	-268,	mant	=	0x0F851E60000000	
*** Test passed ***											
@I [SIM-1] CSim done with 0 errors.											-
	111										

C Simulation	5	Click the Run C Simulation button of the toolbar.
Options		the toolbal.
Launch Debugger		Check the Launch
Build Only		Debugger option a
🔄 Clean Build		
Optimizing Compile		click OK
Input Arguments		
	🔲 Do not show this dialog box again.	
	OK Cancel	
5 K		

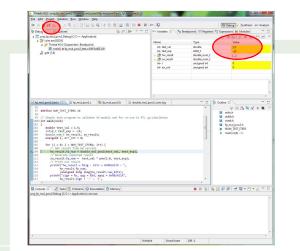
Notice how the Debug view gets activated, the test bench started and stop at the first instruction of the main function. The test bench can be run step-by-step, breakpoints set, variables and expressions evaluated just like any other software project.





Double click on the blue column in line 109 to place a breakpoint at the line that calls the double\_mul\_pow2 function.

Click the Resume button in the toolbar to run the test bench until the breakpoint is hit



Step into the double\_mul\_pow2 function

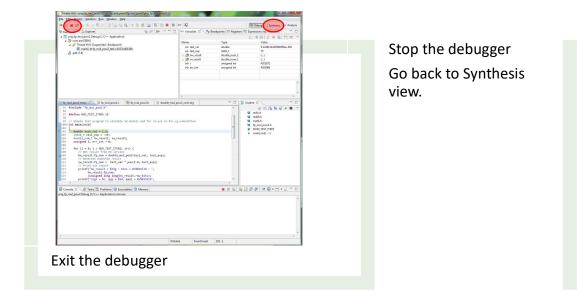
Notice how the variables test\_val and test\_exp changed before the breakpoint was hit.

Click the Step Into button in the toolbar or press F5 on your keyboard.

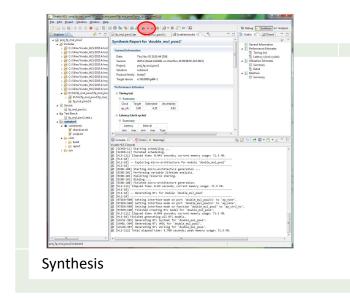
Keep pressing F6 to execute the function statement-by-statement.







Notice how solution1 in the project view has a csim folder now. Synthesis directives and simulation/synthesis results are grouped into solutions. Having multiple solutions allows us to try different settings, devices, clock periods on the same set of source files and analyze the results for each.



Synthesize the design by clicking the C Synthesis button in the toolbar.

Watch the messages in the console until synthesis completes.

Notice the new syn folder in solution1 and the Synthesis Report that opened automatically.

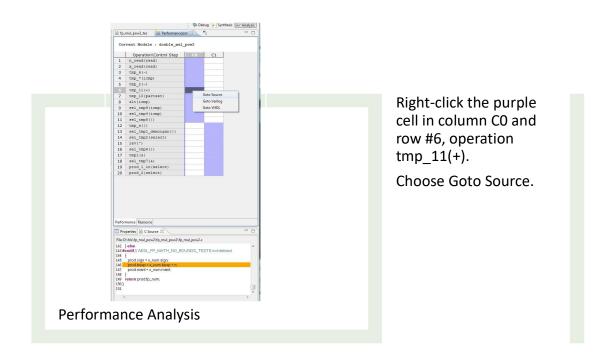


Discuss the report. What did HLS synthesize? What are the latency and interval values? What are the interfaces that got generated?

🗄 Module Hierarchy 🛛 🖻 🔍 🗆	🗊 Synthesis(solut 🕞 Performance(sol 🗵 🐂 😐 🗆	
BRAM         DSP         FF         LUT         Latency         Intervi <ul></ul>	Current Module : double_mul_pow2	Open the Analysis view.
e m Petersana Potte 27 E Resource Potte 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Department (construct term)         CO         C1           1         n_cread (read)            2         n_cread (read)            3         trp_(c)            6         org_(c)            7         trp_(c)            8         org_(c)            9         stit(unp)            10         st_t_upp(cons)            11         st_t_upp(cons)            12         org_(t)            13         st_t_upp_(descoptan)()            14         st_t_upg_(descoptan)()            15         reav(*)            18         st_t_upg7(4)	
	19 prod 1 in (select) - Performance Resource	
	Properties S E Properties S	
	Property Value	
4 <u> </u>	· · · ·	
Analysis		

The Analysis view helps in understanding and evaluating the synthesized design. The synthesized modules and loops can be seen on the left, along with timing and logic utilization information. In this case double\_mul\_pow2 does not have any sub-blocks, it is a flat function. Selecting an item will bring up the Performance view on the right. This shows the control states of the logic (C0, C1) and each operation that is scheduled to execute in that state.





When the synthesized design satisfies all the project requirements, the next step is running an RTL simulation to verify that it is functionally correct. In Vivado HLS terminology this is called C/RTL Cosimulation. Vivado HLS is capable of automatically generating an RTL test bench by running the C test bench and using the inputs from there as stimuli and the outputs as expected values.



Verligs VHOL Simulator Selection	Click on the C/RTL Cosimulation button on the toolbar Choose "all" for the Dump Trace option Click OK. Review the messages in Console
----------------------------------	--

The Dump Trace option will export the RTL simulation waveforms that can be opened in Vivado Simulator, for example.

• ☑ 册 : ப → 🖬 🗗 : 6♂ : 🗐 on(solu 🖾 → ³ Open Wave Viewer) 'double mul pow2'	Click on the Open Wave Viewer button on the toolbar Wait for Vivado to open Open the Window menu and go to Waveform
View simulation waveforms	

Analyze the simulation waveforms. Look for input values, results. Measure latencies and initiation intervals.

Since the hardware is now validated, all that is left is to package it up into a reusable format.





A Export RTL Dialog		
Export RTL	te ti irr	Click on Export RTL
Format Selection n IP Catalog	Configuration	button on the toolbar
d Options Evaluate Verilog		Leave options on thei defaults
d d		Click OK
		Wait for export to complete
1	Do not show this dialog box again.	

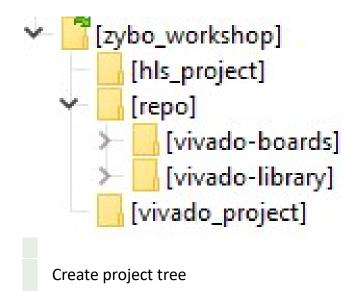
The exported IP files are generated in the active solution folder under impl. Locate the files and explore the sub-folders.

This concludes our first task – Getting familiar with the interface.



## 5 Task Two – Create a pass-through video pipeline

In this step we are going to create an FPGA project that decodes DVI input and forwards it to the VGA output. This pipeline will serve as the base design that will accept the IP exported from HLS. We are going to create it in Vivado block design re-using IP available from Digilent and Xilinx. The Digilent IPs are available online at <a href="https://github.com/Digilent/vivado-library/archive/master.zip">https://github.com/Digilent/vivado-library/archive/master.zip</a> or among the workshop materials.



Copy the folder called "zybo\_workshop" to the root of your local hard drive.

If you choose a location other than root, make sure the path has no spaces in it. Take note of the path as you will need it later.

File Home Share	View							~ 0
Pin to Quick Copy Paste access Clipboard	X Cut Copy path Paste shortcut	Move Copy to to Urganize	New folder	₩ New item • Easy access • New	Properties	Edit History	Select all Select none Invert selection Select	n
🗧 🔶 👻 🛧 🚺 %AP	PDATA%\Xilinx\Viv	vado∖			<u>ت</u> ~	Search Vivad	lo	P
Macromedia Microsoft Mozilla Notepad++ Pulse Secure Skype VMware	C:\Users\r	hs_err_pid19504.log hw_ila_data_1_3612_1481645525.btr init.tcl ogyorge\AppData\Roaming\Xilinx\Vi earch View Encoding Language	vado\init. Seţting	s <u>M</u> acro <u>R</u> un	Text D BTREE TCL F Plugins	le <u>Window ?</u>	Size 36 KB 35 KB 0 KB 1 KB	
Xilinx Common	init.td 🖾	t param board.repoPaths	C   ma		Contract Contract	include a	-boards/new"	•
ICE		-					•	
ISE PlanAhead updatemem								
PlanAhead	length : 74 lin	es:1 Ln:1 Col:7			1.4180	dows (CR LF)	UTF-8	

Add Digilent board definition files to Vivado

On Windows browse to: %APPDATA%\Xilinx\Vivado\

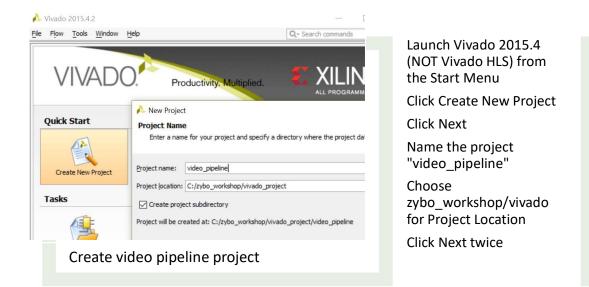
On Linux cd to: \$HOME/.Xilinx/Vivado/

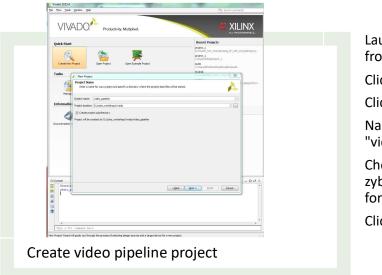
Copy the provided "init.tcl" there.

If you copied "zybo\_workshop" to a location other than c:\, edit this file. Make sure the path is absolute and use forward slash "/" as path separator even on Windows.

Save the file and close the editor.







Launch Vivado 2015.4 from the Start Menu

Click Create New Project

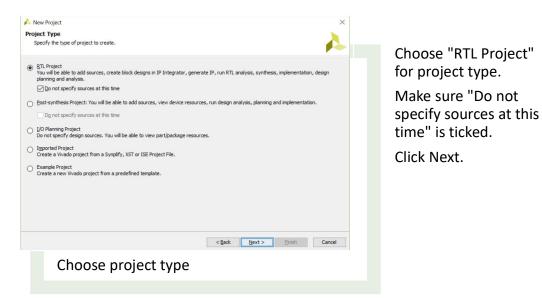
Click Next

Name the project "video\_pipeline"

Choose zybo\_workshop/vivado for Project Location

**Click Next** 





A New Project Default Part Choose a default Xilinx part or board for your project. This can be changed later Select Boards Select: 
Parts Boards
Filter Choose digilentinc.com Vendor: digilentinc.com for Vendor Display Name: All Remaining Board Rey: Latest Reset All Filters Choose Zybo from the Search: Q list below. Bloci RAM Display Name Board Rev Part I/O Pin Count File Ve ndor Basys3 Cmod A7-15t Cmod A7-15t Genesys2 Nexys4 Nexys4 Nexys Video Zedboard xc7a35tcpg236-1
 xc7a15tcpg236-1
 xc7a15tcpg236-1
 xc7a35tcpg236-1
 xc7a325tfg900-2
 xc7a100tcsg324-1
 xc7a100tcsg324-1
 xc7a100tcg484-1
 xc7a200tbg484-1
 xc7a200tbg484-1 ilentinc.com C.0 ilentinc.com B.0 ilentinc.com B.0 236 1.1 50 25 50 445 135 135 365 140 **Click Next** 23 236 nc.com H 900 324 324 484 **Click Finish** com C.1 com A.0 < <<u>Back Next</u> > Einish Cancel Choose the Digilent Zybo as target

If "Zybo" is not showing among the known boards, go back a few steps and make sure init.tcl is installed at the correct location and it has a valid path in it. Restart Vivado and make sure the Tcl Console is showing that init.tcl has been successfully sourced.



🝌 Project Settings

20

Genera

Simulation 80

Elaboratio

8

Synthesis

Bitstrea

IP

+

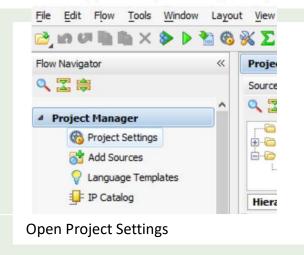
4

General



Open Project Settings from the Flow Navigator on the left

🝌 video\_pipeline - [C:/zybo\_workshop/vivado\_pr



Repository Manager Packager IP Cache Select the IP category (i) Add directories to the list of repositories. You may then add additional IP to a selected repository. If an IP is disabled then a tool-tip will alert you to the reason. Switch to the Repository Manager tab Click the green plus button Browse to the zybo workshop/repo/vivadolibrary folder **Click Select** Vivado will parse the folder and should find IP definitions there Click OK to close Project OK Cancel Apply Settings. Add Digilent IP definitions to Vivado

In this project we are going to use the block design flow to create the FPGA design. This helps us reuse any available IP so that we can focus on the processing IP created in HLS. The following IPs are going to be used from Digilent: DVI-to-RGB (DVI Sink), RGB-to-VGA.

And from Xilinx: Video In to AXI4-Stream, AXI4-Stream to Video Out.

Refresh All



Click Create Block Design on the left toolbar Leave the defaults and

click OK

No. 200     No. 200     No. 200     No. 200     No. 200     No. 200       No. 200     No. 200     No. 200     No. 200     No. 200     No. 200       No. 200     No. 200     No. 200     No. 200     No. 200     No. 200       No. 200     No. 200     No. 200     No. 200     No. 200     No. 200       No. 200     No. 200     No. 200     No. 200     No. 200     No. 200       No. 200     No. 200     No. 200     No. 200     No. 200     No. 200       No. 200     No. 200     No. 200     No. 200     No. 200     No. 200       No. 200     No. 200     No. 200     No. 200     No. 200     No. 200       No. 200     No. 200     No. 200     No. 200     No. 200     No. 200       No. 200     No. 200     No. 200     No. 200     No. 200     No. 200       No. 200     No. 200     No. 200     No. 200     No. 200     No. 200       No. 200     No. 200     No. 200     No. 200     No. 200     No. 200       No. 200     No. 200     No. 200     No. 200     No. 200     No. 200       No. 200     No. 200     No. 200     No. 200     No. 200     No. 200       No. 200     No. 200     No. 200
A Functionary Comparison Com
Survivas sitings     Survivas sitings     Survivas sitings     Survivas sitings     Survivas sites
1 m /
Teste and ed an P subsystem to the project

Block Design - design\_1 - 🗆 🖻 × 🚰 Diagram 🗙 📳 Zybo\_B.xdc 🔅 Board < ☎ ⊜ ፪ ∞ ♪ ♪ 🔆 🛔 📥 design\_1 
 Q
 Image: Control in the second sec 0+ 0-1 ŵ ₩ P & ī -Block Properties \_ 🗆 🖻 × 9 ← → 🛐 🦎 J dvi2rgb\_0 Name: dvi2rgb\_0 Parent name: design\_1 e General Properties IP ; · · · Use Board interfaces

Click on the Board tab to see the interfaces that are available for the Zybo in board design flow.

Double-click on System Clock

Accept the default of instantiating a new Clocking Wizard

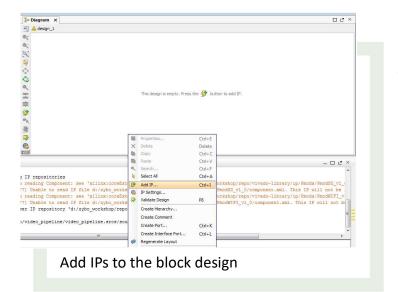
Double-click on HDMI In

Accept the default of instantiating a new DVI to RGB Converter IP





- Diagram X			0 e ×	
*] 🛕 design_1				
2 E S S S S S S S S S S S S S S S S S S	This design is empty. Pres	is the 🐓 but	ton to add IP-	Right-click on an emp space in the diagram and choose Add IP
	Properties	Ctrl+E		Search for VGA and double-click "RGB to
· · · · · · · · · · · · · · · · · · ·	X Delete Copy	Delete Ctrl+C		VGA output"
	Paste	Ctrl+V	- 0 e ×	
	Search	Ctrl+F	- 0 0 ^	
IP repositories	Select Al	Ctrl+A		
reading Component: see 'xilinx:coreExt	Add IP	Ctrl+1	orkshop/repo/vivado-library/ip/Pmods/PmodSD_vl_(	
) Unable to read IP file d:/zybo_works reading Component: see 'xilinx:coreExt ) Unable to read IP file d:/zybo works	<ul> <li>IP Settings</li> <li>Validate Design</li> </ul>		PmodSU_v1_0/component.xml. This IP will not be orkshop/repo/vivado-library/ip/Pmods/PmodWIFI_v/ PmodWIFI_v1_0/component.xml. This IP will not be	
r IP repository 'd:/zybo_workshop/repo /video_pipeline/video_pipeline.srcs/sou =	Create Hierarchy Create Comment Create Port Create Interface Port Pregenerate Layout	Ctrl+K Ctrl+L	,*	



Repeat for "Video In to AXI4-Stream", "AXI4-Stream to Video Out,, "Video Timing Controller", and two instances of "Constant"



v_vid_in_axi4s_0	Block Interface Properties	Ctrl+E		
V_Vid_in_axi+s_0	Highlight     Delete	Delete	F	Right-click the dout
-wd_io_n_ce -ock vtiming_out-} If to RGB )	Сору	Ctrl+C	i	nterface on the
aresets underflow	Paste	Ctrl+V Ctrl+F		clonstant 1 block and
Video In to AXI4-Stream	Select All	Ctrl+A )		—
-	Add IP	Ctrl+I Ctrl+T	C	hoose "Make External
	Run Connection Automation	curr		Papast for
	- 🔞 IP Settings Validate Design	F6		Repeat for
	Start Connection Mode	Ctrl+H		gb2vga_0/vga_pRed,
0 v_vid_in_axi4s_0	Make Connection		r	gb2vga_0/vga_pGreer
.0 v_axi4s_vid_out_0	Disconnect Pin Disconnect Pin Inside Hierarchy		r	gb2vga_0/vga_pBlue,
	Create Hierarchy			gb2vga_0/vga_pHSync
rgb2vga_0	Create Comment	100000		
	Create Interface Port	Ctrl+L	I,	gb2vga_0/vga_pVSync
<u></u>	Save as PDF File			
Detection/Generation	Default/Constant Frame Sync Pos	sition		
Detection/Generation Optional Features	Default/Constant Frame Sync Pos	sition		
Optional Features	terface	sition		
Optional Features	terface	sition		
Optional Features	terface	sition		
Optional Features	terface sce oport tor to Detector or to fsync_in	sition		Double-click the v_tc_0 block
Optional Features  Include AXI4-Lite In Include INTC Interfa Interlaced Video Sug Synchronize Genera  Max Clocks Per Line 4096 • Frame Syncs 1 • Enable Generation	terface sce pport tor to Detector or to fsync_in Max Lines Per Frame 4096 ▼	sition )	b	block
Optional Features  Indude AXI4-Ute In Indude INTC Interfa Interlaced Video Sup Synchronize Generation Max Clocks Per Line 4096 Frame Syncs 1 Frame Syncs 1 CEnable Generation Generation Options	terface ace oport tor to Detector or to fsync_in Max Lines Per Frame 4096 v Detection Options		t C	olock Configure it like shown
Optional Features  Include AXI4-Lite In Include INTC Interfa Interlaced Video Sug Synchronize Genera  Max Clocks Per Line 4096 • Frame Syncs 1 • Enable Generation	terface ace oport tor to Detector or to fsync_in Max Lines Per Frame 4096 v Detection Options		t C	block
Optional Features  Indude AXI4-Ute In Indude INTC Interfa Interlaced Video Sup Synchronize Generation Max Clocks Per Line 4096 Frame Syncs 1 Frame Syncs 1 CEnable Generation Generation Options	terface poprt tor to Detector or to fsync_in Max Lines Per Frame 4096 * Detection Options Field ID Detection	tection	t C	olock Configure it like shown
Optional Features  Include AXI4-Lite In Include INTC Interfa Include INTC Interfa Synchronize Generation Generation Options Field ID Generation	terface ace oport tor to Detector or to fsync_in Max Lines Per Frame 4096 Detection Options Field ID Det ration	tection nk Detection	t C	olock Configure it like shown
Optional Features	terface ace oport tor to Detector or to fsync_in Max Lines Per Frame 4096 * Detection Options Field ID Detection Pield ID Detection Vertical Blan neration Vertical Blan	tection nk Detection	t C	olock Configure it like shown
Optional Features  Include AXI+-Life Im Include INTC Interfa Interlaced Video Sup Synchronize Generation Generation Options  Field ID Generation Vertical Blank Gener Horizontal Blank Generation	terface ace pport tor to Detector or to fsync_in Max Lines Per Frame 4096 * Detection Detection Options Field ID Detection Pield ID Detection Pield ID Detection Pield ID Detection Vertical Blan heration Vertical Syn	tection ak Detection Mank Detection ac Detection	t C	olock Configure it like shown
Optional Features  Include AXI4-Ute In Include INTC Interfa Include INTC Interfa Synchronize Generation Generation Options  Field ID Generation Vertical Blank Gener Vertical Blank Generation Vertical Blank Generation	terface see poort tor to Detector or to fsync_in Max Lines Per Frame 4096 + Detection Options Field ID Detection Detection Options ation Vertical Blan meration Horizontal B ation Vertical Sync erration Horizontal S	tection nk Detection Mark Detection to Detection Sync Detection	t C	olock Configure it like shown
Optional Features	terface ace opport tor to Detector or to fsync_in Max Lines Per Frame 4096 Detection Options Field ID Det ration Vertical Blan meration Vertical Blan terration Vertical Syn meration Vertical Syn terration Vertical Syn meration Vertical Syn terration Vertical Syn meration Vertical Syn	tection nk Detection kank Detection sc Detection Sync Detection o Detection	t C	olock Configure it like shown
Optional Features  Include AXI4-Lite In Include INTC Interfa Synchronize Generation Generation Options Field ID Generation Vertical Blank Gener Vertical Sync Generation Horizontal Blank Gener Vertical Sync Generation Horizontal Sync Generation Vertical Sync Generation Horizontal Sync Generation	terface acc apport tor to Detector or to fsync_in Max Lines Per Frame 4096     Max Lines Per Frame 4096	tection nk Detection kank Detection sc Detection Sync Detection o Detection	t C	olock Configure it like shown



he phase is ca		lative to the active	input clock.		
Dutput Cloc		put Freq (MHz) juested	Actual	Phase (degrees) Requested	Actual
-		luested			
✓ dk_out1	200	0	200.000	0.000 (	0.000
dkout2	100	.000	N/A	0.000	N/A
	100	.000	N/A	0.000	N/A.
clk_out4	100	.000	N/A	0.000	N/A
dk_out5	100	.000	N/A	0.000	N/A
clk_out6	100	.000	N/A	0.000	N/A
dk_out7	CK SEQUE	.000 ICING Sequence Num		0.000 ocking Feedback Source	N/A S
	CK SEQUE		Ck	ocking Feedback Source	
USE CLO	CK SEQUE Clock	ICING	Ck	Source	Ş
USE CLO	CK SEQUE Clock	ICING	Ck	Ocking Feedback Source  O Automatic ( Automatic (	S
USE CLO	CK SEQUE Clock It1 It2 It3	Sequence Num	Ck	Source     Automatic (     Outomatic (	S Control On-Chip Control Off-Chip olled On-Chip
USE CLO	CK SEQUER Clock A1 A2 A3 A4	Sequence Num	Ck	Ocking Feedback Source  O Automatic ( Automatic (	S Control On-Chip Control Off-Chip olled On-Chip
USE CLO Output ck_o ck_o ck_o ck_o ck_o ck_o	CK SEQUEI CC SEQUEI Clock At1 At1 At2 At3 At4 At5 At5 At6	Sequence Num 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Ck	Source     Automatic (     Outomatic (	S Control On-Chip Control Off-Chip olled On-Chip
USE CLO Output ck_o ck_o ck_o ck_o ck_o	CK SEQUEI CC SEQUEI Clock At1 At1 At2 At3 At4 At5 At5 At6	Sequence Num 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Ck	Source     Automatic (     Outomatic (	S Control On-Chip Control Off-Chip olled On-Chip
USE CLO Output ck_o dk_o dk_o dk_o dk_o	Ck sequer Clock at1 at2 at3 at4 at5 at5 at5 at7	Sequence Num 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Ck	Source     Automatic (     Outomatic (	S Control On-Chip Control Off-Chip olled On-Chip
	CK SEQUEI CC SEQUEI At1 At2 At3 At4 At5 At5 At5 At5 At5 At7 At1 At1 At1 At1 At1 At1 At1 At1 At1 At1	Sequence Num           1           1           1           1           1           1           1           1           Duputs	Ck	Odring Feedback Source	S Control On-Chip Control Off-Chip olled On-Chip olled Off-Chip

#### Double-click the clk\_wiz\_0 block

Open the Output Clocks tab.

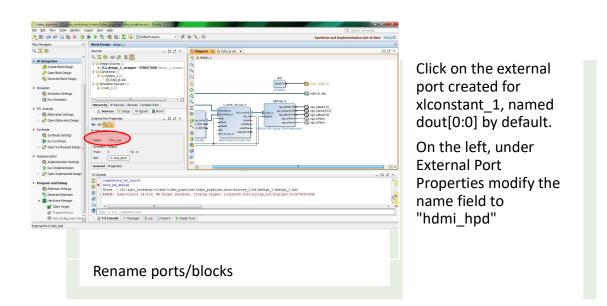
Configure it like shown on the left

The IP should generate a 200MHz clock from the 125MHz on-board clock

Component Name design_1_dvi2rgb_0_0	
Enable DDC ROM	
Enable serial dock output	Double-click the
Add BUFG to PixelClk	dvi2rgb_0 block
Resets active high	Configure it like shown
TMDS clock range	on the left.
○ >=120 MHz (1080p)	on the left.
(e) < 120 MHz	
Preferred resolution	
○ 1280x720	
1600x900     1600x900	
○ 1280x1024	
○ 1920×1080	
DVI to RGB Video Decoder configu	tion
DVI to NOD VIGEO DECOGEI COIIIgu	

DVI Sinks are required to bring the hot plug detect (HPD) pin high to signal their presence to DVI Sources. We will name this port hdmi\_hpd and tie to a constant high value.







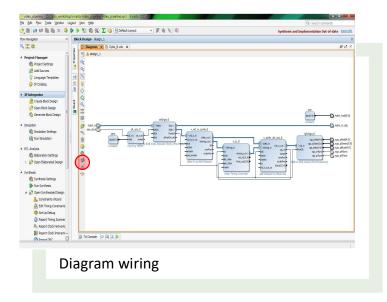
Re-customize IP Constant (1.1) Documentation 🔂 IP Location		
Show disabled ports	Component Name         design_1_skonstant_0_0_0           Const Width         1           Const Vid         0	
	OK Cancel	
Constant	onfiguration	

In the same manner, select the xlconstant\_0 block and rename it to "zero".

Double-click the xlconstant\_0 block

Configure it so the Const Val is "0".

Repeat for xlconstant\_1 block, but rename it to "one" and configure it for Const Val 1.

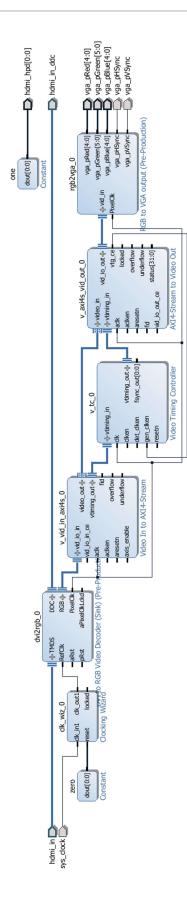


Wire the blocks like shown on the next page.

Click-and-hold on one interface and drag it to another to establish a connection.

The Regenerate Layout button on the toolbar to the left of the diagram will re-arrange the blocks into a more readable layout.





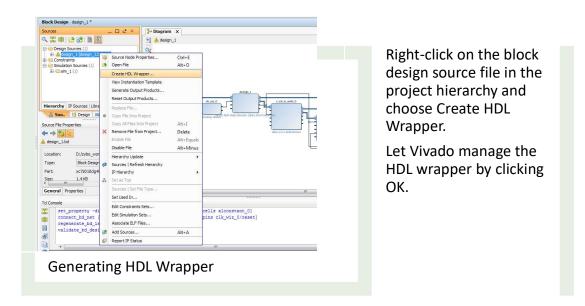


🛤 ei ill ill X 📑 🏷 🕨
igstor « Block I
*
At theory in the second

Validate the design by clicking on the corresponding button on the toolbar on the left

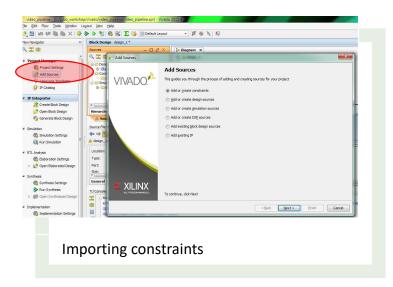
There should be no errors reported

If there are, revisit the wiring between blocks



All that is left is adding the constraint file which tells the synthesis tool about physical constraints for the design like which FPGA pin to use for each interface and timing constraints like the maximum frequency for the DVI pixel clock.





Click the Add Sources button on the left toolbar

Choose Add or create constraints

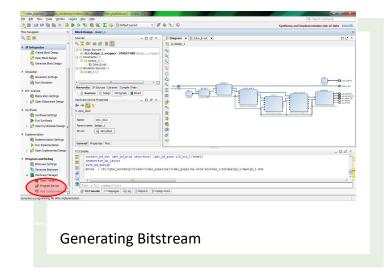
Click Next

Click Add files

Browse to the provided Zybo\_B.xdc

Make sure the "Copy constraints files" option is ticked.

Click Finish

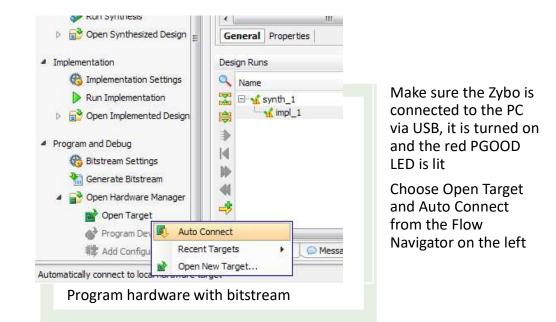


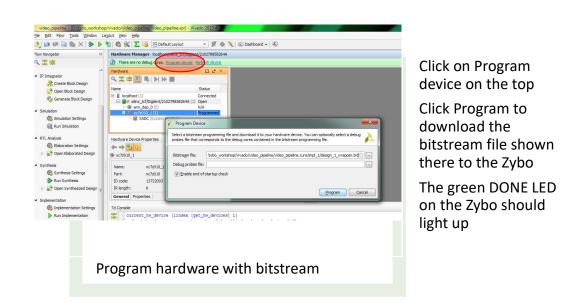
Click Generate Bitstream in the Flow Navigator on the left.

If asked, save the design and confirm that synthesis and implementation should be run.

When bitstream generation is completed, choose "Open Hardware Manager", which is also accessible in Flow Navigator.







The board is now ready to forward video input on its DVI port to VGA. Connect the Zybo to an HDMI source like a laptop and to a VGA monitor. The laptop should recognize it as a display and you should be able to extend your desktop to it. The extended desktop should be forwarded by the Zybo to the VGA monitor.

Copyright Digilent, Inc. All rights reserved. Other product and company names mentioned may be trademarks of their respective owners.

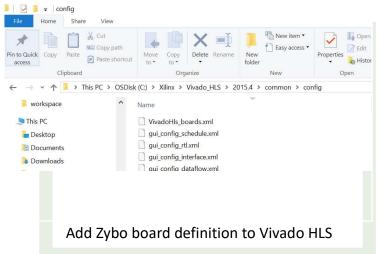


## 6 Task Three – Edge detection in HLS

The video pipeline created in Task 2 provides a good basis for image processing functions defined in HLS. The bus between blocks "Video In to AXI4-Stream" and "AXI4-Stream to Video Out" is a streaming interface sending data pixel-by-pixel in raster format. While it may seem unnecessary to convert the RGB video data to AXI-Stream then back, this step ensures the greatest interoperability between IPs. The RGB video stream is a continuous stream of pixels forming lines interleaved by blanking intervals. It lacks a hand-shake mechanism that could stop the stream for a while when the downstream processing logic requires it. AXI-Stream transmits data more efficiently by packing pixel data and framing signals. Furthermore, thanks to hand-shake signals it allows for buffering and stopping the stream momentarily. All Xilinx Video Processing IP use AXI-Stream interfaces, if needed these can be easily inserted into the stream. Due to the streaming nature of the data, different processing blocks can even be daisy-chained by attaching the output of one to the input of another. This is called video processing pipeline.

The interface of the pipeline is an essential design aspect of an HLS processing core. The input, output and control interfaces all need to be modeled in C/C++. Fortunately, the data type modeling AXI-Stream already exists in HLS template libraries.

So our task is writing a processing block (function), that accepts an AXI-Stream RGB video input (argument), and outputs the similarly formatted processed video data (argument). The project requirements are 1280x720@60Hz resolution and a stable video feed.



Browse to your Vivado\_HLS installation folder.

For example, on Windows: C:\Xilinx\Vivado\_HLS\2015.4\ common\config

Or on Linux:

/opt/Xilinx/Vivado\_HLS/2015. 4/common/config

Overwrite VivadoHls\_boards.xml with the one provided among the workshop materials



A New Vivado HLS Project	-		
Project Configuration		AG	
Create Vivado HLS project of selected type			
Project name:       edge_detect         Location:       C:\zybo_workshop\hls_project	Br	owse	Launch Vivado HLS 2015.4 (NOT Vivado 2015.4) from the Start Menu Click Create New Project Name the project edge_detect Place it under zybo_workshop\hls_project Click Next
< <u>B</u> ack <u>N</u> ext > Einish	C	ancel	
Now Vivado HLS project			
New Vivado HLS project			
New Vivado HLS Project	— [	з х	
Add/Remove Files		*12	
Add/remove C-based source files (design specification)		-	
Top Function:		Browse	Click New File Browse to
Name CFLAGS  edge_detect.cpp edge_detect.h	Add F New F Edit CFL Remo	File	zybo_workshop\hls Name it edge_detect.cpp Repeat for edge_detect.h
< <u>B</u> ack <u>N</u> ext > Einish	C	ancel	Click Next
Create new source and header	files		



🝌 New Vivado HLS Projec	t	– 🗆 X	
Add/Remove Files		+	
Add/remove C-based test	bench files (design test)	Þ	
TestBench Files			Click New File
Name edge_detect_test.cpp	CFLAGS	Add Files New File Add Folder Edit CFLAGS Remove	Browse to zybo_workshop\hls Name it edge_detect_test.cpp Click Next
Create	new source file for test		
New Vivado HLS Project		×	
Create Vivado HLS solutio	n for selected technology		
Solution Name: solution1 Clock Period: 13.5 Part Selection Part: <b>Digilent Zybo (xc</b>	Uncertainty:		Enter 13.5 for clock period Click the browse button for part
1		konnadi	selection
			Click Boards
			Choose Digilent Zybo in the list of boards
< <u>B</u> a	ick <u>E</u> inish	Cancel	Click Finish
Create			

Now that the project is created we can get on with writing actual C++ code. The following files will be written.



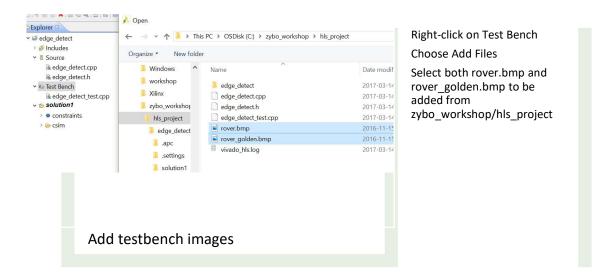
```
#include "hls_video.h"
typedef ap_axiu<24,1,1,1> interface_t;
typedef hls::stream<interface_t> stream_t;
void edge_detect(stream_t& stream_in, stream_t& stream_out);
#define MAX_HEIGHT 720
#define MAX_WIDTH 1280
typedef hls::Mat<MAX_HEIGHT, MAX_WIDTH, HLS_8UC3> rgb_img_t;
#define INPUT_IMAGE "rover.bmp"
#define OUTPUT_IMAGE "rover_output.bmp"
```

```
#include "edge_detect.h"
void edge_detect(stream_t& stream_in, stream_t& stream_out)
{
    int const rows = MAX_HEIGHT;
    int const cols = MAX_WIDTH;
    rgb_img_t img0(rows, cols);
    rgb_img_t img1(rows, cols);
    rgb_img_t img2(rows, cols);
    rgb_img_t img3(rows, cols);
    hls::AXIvideo2Mat(stream_in, img0);
    hls::CvtColor<HLS_RGB2GRAY>(img0, img1);
    hls::CvtColor<HLS_GRAY2RGB>(img2, img3);
    hls::Mat2AXIvideo(img3, stream_out);
}
```

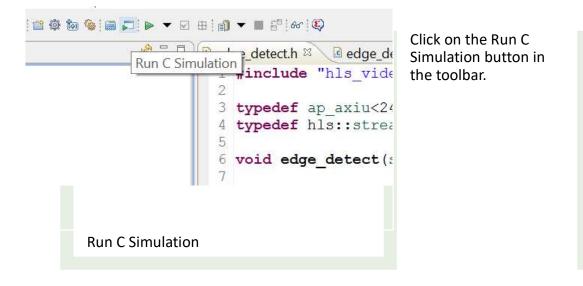


```
#include "edge_detect.h"
#include "hls_opencv.h"
int main()
{
    int const rows = MAX_HEIGHT;
    int const cols = MAX_WIDTH;
    cv::Mat src = cv::imread(INPUT_IMAGE);
    cv::Mat dst = src;
    stream_t stream_in, stream_out;
    cvMat2AXIvideo(src, stream_in);
    edge_detect(stream_in, stream_out);
    AXIvideo2cvMat(stream_out, dst);
    cv::imwrite(OUTPUT_IMAGE, dst);
    return 0;
}
```

As shown in Task 1, the HLS flow is going to be followed: the processing function written, a test bench written for it, synthesis, report analysis, C/RTL co-simulation and IP export. The process is iterated until all the requirements are met.







General Simulation	Synthesis Setti	ngs		
Synthesis	Top Function: edge_e	detect	Browse	
	Synthesis C/C++ Sou	urce Files		Click on the Project Menu
	Name	CFLAGS	Add Files	
	edge_detect.h		New File	Choose Project Settings
			Edit CFLAGS	Choose Synthesis on the
			Remove	left
				Click Browse next to Top
				Function
				Choose edge_detect
				Click OK
				Click on the Run C
			OK Cancel	Synthesis button in the
			Callee	toolbar to start hardware

After hardware synthesis completes, review the report for clues on whether project requirements are met. If analysis determines that the synthesized code does not meet the requirements, HLS can be directed towards a better design. This is achieved using directives. These influence the choice HLS makes during synthesis both relating to generated logic and interfaces. In this task, we are going to

Copyright Digilent, Inc. All rights reserved. Other product and company names mentioned may be trademarks of their respective owners.



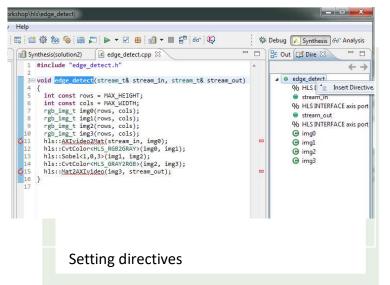
set the DATAFLOW and INTERFACES directives. To be able to compare the results with and without the directives, a new solution can be created.

•!~ D	٩	Project Settings	▶ ▼ 🛛 🖶 🗊 ▼
È Explore ➤ 😂 edg > 🔊 In ➤ 喜 Sc		Index C Source Run C Simulation Add Source New Source	P □ D edge Synth Gene
* ≡ 3( @ @ ₩		Add Test Bench New Test Bench Add Test Bench Folder	Date Versi
		<b>New Solution</b> Set Active Solution Close Inactive Solution Tabs	Proje Solu Prod
> 😕 SI	10	Compare Reports	Targe

Open the Project menu and choose New Solution.

Click on Finish to accept the defaults. Notice that settings from solution1 are going to be copied to the new solution.

Solution2 now becomes active.



Open edge\_detect.cpp, which has the function that needs directives applied

On the right side panel, click on the Directives tab

Select stream\_in **and** stream\_out interfaces

Right-click on the selection and choose Insert Directive



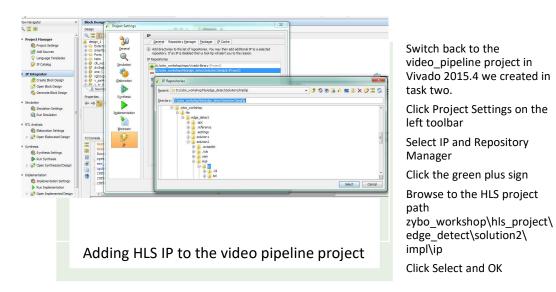
Vivado HLS Directive Editor	
Directive	
INTERFACE	
Destination	
Source File  Directive File	
Options	In the dialog that opens
mode (optional):	<b>c</b> ,
register (optional);	choose the INTERFACE
depth (optional):	directive
achter (cheasan)	
	For mode option choose
port (required): stream_in, stream_out	axis to instruct synthesis
	to generate an AXI-
	J. J
	Stream interface for
clock name (optional):	stream in and
	stream out.
	stream_out.
	Similary, select function
Help Cancel OK	edge detect and
	0 =
Choosing directive options	activate the DATAFLOW
choosing unective options	directive on it.

Run hardware synthesis one more time and compare the results to that of solution1. Once the design meets the requirements, it can be packaged and exported as an IP. Just choose the Export RTL action in the top toolbar.

Export RTL ect.cpp @ edge_detect_te	es Synthes	
🝌 Export RTL Dialog	×	
Export RTL	+	Click on the Export R button in the top toolbar.
	guration	Keep the defaults by clicking on OK.
Evaluate Verilog V		Notice the impl subdirectory in
Do not show this dialo	og box again. ancel	solution2 that will be created.
ОК С	uncer	



The next step is importing the video processing IP in the Vivado project and inserting it into the video pipeline.



Right-click on an empty space in the diagram and choose

Add IP Search for and double-click

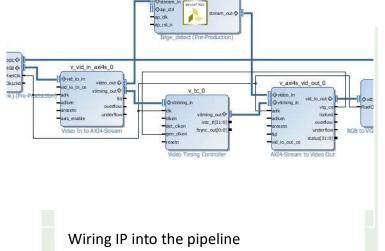
on Edge\_detect, which is our HLS IP

Click on the wire between v\_vid\_in\_axis4s/video\_out and v\_axi4s\_vid\_out/video\_in

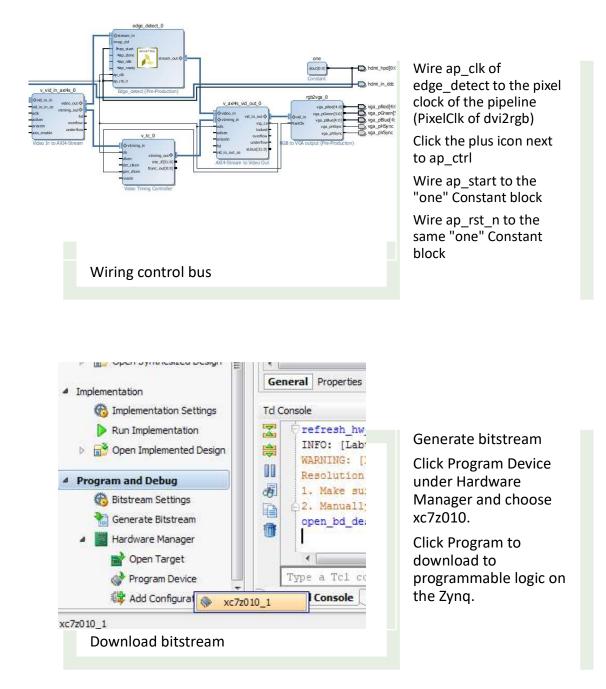
Press the Delete key

Wire video\_out to stream\_in of edge\_detect

Wire stream\_out to video\_in







Zybo should now forward incoming video data to the VGA after applying the edge detection algorithm on it. Display any image, movie or just the Windows desktop on the secondary display to see edge detection in action.

This concludes our workshop. Thank you for attending!