Getting Started with OpenCL on the ZYNQ

Bo Joel Svensson bo.joel.svensson@gmail.com Rakesh Tripathi rakesht@chalmers.se

Guide version	0.5
Last edited	October 12, 2018
Board compatibility	zynqberry
Tested Vivado versions	2015.4, 2018.2

Disclaimer

All content provided in this document is for informational purposes only. The authors makes no guarantees as to the accuracy or completeness of any information within this document.

The authors will not be liable for any errors or omissions in this information nor for the availability of this information. The authors will not be liable for any losses, injuries, or damages from the display or use of this information.

1 Introduction

This document attempts to provide a complete walk through of the entire OpenCL HLS work flow using Xilinx Vivado. The Board we target in this version (0.5) of the document is the Trenz ZynqBerry. In future versions we will try to highlight the parts that differ when using a ZedBoard.

This document is work in progress and new versions will be posted as we refine the procedure and gain a deeper understanding of all the details.

All feedback, hints, tips, corrections and explanations of details we are vague upon, would be greatly appreciated and acknowledged in future revisions.

1.1 Initial setup for the zynqberry

The procedure outlined in this document has been tested against a ZynqBerry board ¹. The version of this board that we use is the 128MB variant "TE0726-02" which has now been superseded by the "TE0726-02M" variant with more memory. While we do not have access to an "TE0726-02M" board and we cannot test, we assume the procedure explained in this document will apply with minimal changes.

In order to make the description of the procedure as complete as possible we are not basing this guide on an existing (vendor supplied) example Vivado project. Only one piece of data is taken from a reference design provided by Trenz, the so called "board files". We copy the board files available in the "test board" reference design ². Under linux these board files are copied

¹http://www.trenz-electronic.de/products/fpga-boards/trenz-electronic/te0726-zynq.html

 $^{^{2}} http://www.trenz-electronic.de/download/d0/Trenz_Electronic/d1/TE0726/d2/Reference\%20 Designs/d3/2015.4/d4/test_board.html$

to directory Xilinx/Vivado/2015.4/data/boards/board_files of your Xilinx vivado install tree. If these board files have been added correctly it will be possible to select the ZynqBerry as a target for Vivado.

1.2 Guide structure

This guide is split into three parts that goes through: first writing a simple OpenCL program and synthesizing it using Vivado HLS, second designing a system (in Vivado) that interfaces the hardware generated by HLS in step 1 with the processing system and the memory system in the Zynq chip, finally we show how to develop software (in the SDK) for the processing system that starts computations in the OpenCL generated hardware.

2 Part 1: Vivado HLS and OpenCL

In this section we develop an OpenCL program for vector addition (vadd). This vadd computation is given pointers to three vectors (arrays), two inputs and one output, and performs element wise addition of the inputs into the output.

2.1 Creating a Vivado HLS project

Start Vivado_hls and create project using the following steps:

- step1: Create a project and name it "vadd_OpenCL", see figure 1.
- step2: Now you are asked to provide a name for the top level function. This is the function that specifies the interface to the generated hardware. Name the top level function (vadd). This step is outlined in figure 2.
- step3: We are not adding a testbench file. See figure 2.
- step4: Now it is time to configure the solution details. We can leave the solution name unchanged ("solution1") and then we select the device:
 - Family: Zynq
 - Package: clg225
 - Speed grade: -1

There should now be just one available selection "xc7z010clg225-1". See figure 3.

Now project configuration is done and we can hit finish and enter into the development environment. This should look like the left part of figure 4.

😣 🔿 🗊 Vivado HLS		🛞 🗊 New Vivado HLS Project
کا <u>Vivado HLS Welcome Page</u> ۲۵		Project Configuration
VIVADO.		Project name: [vadd_OpenCL]
Quick Start Create New Project Documentation	Recent Projects	Location: /home/joels/Vivado Browse
Tutorials User Guide Release Notes Guide	(()))) (()))	<back next=""> Cancel Finish</back>

Figure 1: Project creation and naming

😣 🗊 New Vivado	o HLS Project		🛞 🗊 New Vivad	o HLS Project	
Add/Remove File Add/remove C-bas	s sed source files (design specification)	+	Add/Remove File Add/remove C-bas	es sed testbench files (design test)	+
Top Function: vac	bl	Browse	TestBench Files		
Design Files			Name	CFLAGS	Add Files
Name	CFLAGS	Add Files			New File
		New File			Add Folder
		Edit CFLAGS			Edit CFLAGS
		Remove			Remove
	< Back Next > Cancel	Finish		<back next=""> Can</back>	cel Finish

Figure 2: Identification of the Top function and addition of testbench file. We are not providing any testbench.

🛞 💷 New Vivado HLS Project	😣 🗐 Device :	Selection Dialog						
Solution Configuration Part must be specified.	Select: 🔷 Pa	arts Boards						
Solution Name: solution1	RTL Tool	Filter Product Catego	ry: All			Package:	clg225	•
Period: 10 Uncertainty: Part Selection		Family: Sub-Family:	zynq All			 Speed grade Temp grade: 	All	v
Part: [Please select part]	Search: 🔻	amily Dacka	78 50.89	Reset All	Filters	EE	DSP	BDAM
	varu v vc7z010c z	rynq clg22	-1	4400	17600	35200	80	120
< Back Cancel Finish							Cancel	ОК

Figure 3: Solution configuration. Select the "xc7z010clg225-1" device.

2.2 Writing a simple OpenCL kernel

The example kernel used in this guide is very simple and is outlined in total below. Add a new source file to the project. Right click "source" in the *Explorer* under "vadd_OpenCL" and select "New file". Name the file "vadd.cl". The file extension is important ".cl".

```
#include <clc.h>
```

```
__kernel void __attribute__ ((reqd_work_group_size(128,1,1)))
vadd( __global int *a, __global int *b, __global int *c) {
    int i = get_global_id(0);
    c[i] = a[i] + b[i];
}
```

After creating the new file type in the OpenCL code as above.

		88		
🔕 🗇 💿 Vivado HLS - vadd_OpenCL (/home/joels/Vivado/vadd_OpenCL)		Namn: Va	add.cl	
	to Debug 💽 Synthesis & Analysis	Spara i mappen: 🔄	joels Vivado vadd_OpenCL source	Skapa mapp
Co Explorer 🛛 🕜 🖻 🗖	" 🗖 🔡 Outline 🕱 🖉 Directive 👘 🗖	Platser	Namn	▲ Storlek Ändrad
∀ vadd_OpenCL P ⊚ includes	An outline is not available.	Q Sök Tidigare använda		
E Source de Martes Bench ▶ 🎂 solution1		 joels Skrivbord Filsystem Enhet C 		
		in Dokument Musik Bilder Video Hämtningar		
Console 23 (9) Errors & Warnings	Ration de la constante de la			
Vvado HLS Console				
vadd_OpenCL		+		

Figure 4: IDE view and source file creation.

2.3 Synthesize the OpenCL code

After writing the OpenCL, synthesis and exporting the IP remains in order to conclude the part of the work that takes place in vivado_hls. If the code has been entered correctly this should go through synthesis without problems. Hit the green "synthesis" button in the toolbar.

😸 🗐 🗊 Vivado HLS - vadd_OpenCL (/h	ome/joels/Vivado/vadd_OpenCL)	
😗 🛛 🛯 🖳 🖌 🖻 🖻 🗙 🕹	50 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	🕸 Debug 💽 Synthesis ص
🔁 Explorer 🛛 🤌 🗖 🗖	☑ vadd.cl 🗊 Synthesis(solution1) 🛛	🗖 🗖 📴 Outline 🛛 🖓 Directive 🗖 🗖
 vadd_OpenCL includes is Source image Test Bench is solution1 	Synthesis Report for 'vadd' General Information Date: Sat Apr 23 19:48:32 2016 Version: 2015.4 (Build 1412921 on Wed Nov 18 09:58:55 AM 2 Project: vadd_OpenCL Solution: solution1 Product family: zynq Target device: xc7z010clg225-1 Performance Estimates Summary Clock Target Estimated Uncertainty	2015) General Information Gen
	Console X	usage: 59.4 MB.

As the synthesis finishes a post synthesis report is brought up.

Now export the generated hardware description into the IP catalog. This step makes our vadd hardware unit available for use in Vivado. Click the "Export RTL" button in the toolbar.

The choices we make in the "Export RTL" dialog are shown in figure 5. We choose "IP Catalog" and VHDL as the desired language. One can also provide identification details using "Configuration" button but we leave these settings unchanged.

🛞 🗐 Export RTL Dialog		🔕 💷 IP Iden	tification Dialog	
Export RTL	#	Configuration	n	a
Format Selection				
IP Catalog ‡ Config	guration	Vendor:		
Options		Library:	hls	
Evaluate VHDL		Version:	1.0	
		Description:	An IP generated by	/ Vivado HLS
		Display Name:		
		Taxonomy:		
				01/
Do not show	v this <mark>dialog box again.</mark>		Cancel	OK
Cancel	ОК			

Figure 5: The "Export RTL" dialog.

We are now done with vivado_hls and will start up Vivado.

2.4 Programming Interface Generated by HLS

After synthesizing hardware from the OpenCL code new directories appeared called "impl" and "syn" containing VHDL code. Within one of these files we find information that is important to keep in mind later when writing the software that interfaces with the generated hardware, this information will be used in section 4.1. The information we seek, is located in the "vadd_control_s_axi.vhd" file and shows the layout of the memory mapped interface for communication with the vadd hardware unit:

```
-----Address Info------
-- 0x00 : Control signals
__
         bit 0 - ap_start (Read/Write/COH)
         bit 1 - ap_done (Read/COR)
         bit 2 - ap_idle (Read)
___
___
         bit 3 - ap_ready (Read)
___
         bit 7 - auto_restart (Read/Write)
___
         others - reserved
-- 0x04 : Global Interrupt Enable Register
--
         bit 0 - Global Interrupt Enable (Read/Write)
         others - reserved
-- 0x08 : IP Interrupt Enable Register (Read/Write)
         bit 0 - Channel 0 (ap_done)
___
___
         bit 1 - Channel 1 (ap_ready)
         others - reserved
___
-- OxOc : IP Interrupt Status Register (Read/TOW)
--
         bit 0 - Channel 0 (ap_done)
__
         bit 1 - Channel 1 (ap_ready)
__
         others - reserved
-- 0x10 : Data signal of group_id_x
         bit 31~0 - group_id_x[31:0] (Read/Write)
--
-- 0x14 : reserved
-- 0x18 : Data signal of group_id_y
         bit 31~0 - group_id_y[31:0] (Read/Write)
___
-- 0x1c : reserved
-- 0x20 : Data signal of group_id_z
         bit 31~0 - group_id_z[31:0] (Read/Write)
___
-- 0x24 : reserved
-- 0x28 : Data signal of global_offset_x
         bit 31~0 - global_offset_x[31:0] (Read/Write)
___
-- 0x2c : reserved
-- 0x30 : Data signal of global_offset_y
         bit 31~0 - global_offset_y[31:0] (Read/Write)
__
-- 0x34 : reserved
-- 0x38 : Data signal of global_offset_z
         bit 31~0 - global_offset_z[31:0] (Read/Write)
__
-- 0x3c : reserved
-- 0x40 : Data signal of a
___
         bit 31~0 - a[31:0] (Read/Write)
-- 0x44 : reserved
-- 0x48 : Data signal of b
         bit 31~0 - b[31:0] (Read/Write)
-- 0x4c : reserved
-- 0x50 : Data signal of c
__
         bit 31~0 - c[31:0] (Read/Write)
-- 0x54 : reserved
-- (SC = Self Clear, COR = Clear on Read, TOW = Toggle on Write, COH = Clear on Handshake)
```

This gives us the "offsets" from some base address to where each of the register used by the vadd hardware is located. Later, in vivado, a complementary step will provide us with the base address, see section 3.3.

The directly important pieces of information here is the control register, the group_id registers and the a,b and c data registers.

- **Control**: using this register we can start computations in the vadd hardware unit and also poll for the done signal.
- Group id: group_id_x, group_id_y, group_id_z specifies a three dimensional workgroup id. Since the OpenCL kernel we use is meant for one dimensional "NDRanges" only group_id_x is of importance. This value (group_id_x) is changed between invocations of vadd if the data we operate upon is larger than what can be computed by one workgroup instance (the only valid value for the others is zero).
- Argument pointers: pointer to memory where the vadd hardware can fetch and store data should be written to the a,b,c register.

3 Part 2: Vivado

This section presents step by step instructions on how to integrate the OpenCL kernel IP-block designed earlier into a Zynq base system.

3.1 Creating a Vivado project

Begin by starting Vivado. This presents you with the view shown in figure 6. Select "Create New Project" and click "Next".



Figure 6: Vivado project creation wizard.

Choose a name and location for the project, in this case "ZynqOpenCL" and a directory called "Vivado". Click "Next". In the next window select "RTL Project" and check "Do not specify sources..". This part of the procedure is shown in figure 7.

🛞 🐵 New Project	😣 💿 New Project
Project Name Enter a name for your project and specify a directory where the project data files will be stored.	Project Type Specify the type of project to create.
Eroject name: Zynq0penCL	 BTL Project You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis. @ to not specify sources at this time Dest-synthesis Project: You will be able to add sources, view device resources, run design analysis, planning and implementation. Do not specify sources at this time Uo Planning Project O cont specify design sources. You will be able to view part/package resources. Imported Project Create a New Vivado project from a Synplify. XST or ISE Project File. Evanple Froject Create a new Vivado project from a predefined template.
< Back Next > Einish Cancel	< Back Next > Einish Cancel

Figure 7: Project name and project type.

Now it is time to select the "Default Part" to use as target platform. Click "Boards". If the steps in section 1.1 has been performed there should be an option "ZYNQ-7 TE0726-02" for the zynqberry board. Select the suitable board then click "Next" and then "Finish'

😣 💷 New Project						😣 🕕 New Project		
Default Part Choose a default Xilinx part or board for your p	roject. This can	be changed	l later.		1		New Project Summary	
Select: ● Parts ■ Boards ✓ Filter ✓ ✓ Vegdor: All ✓ Display Name: All ✓ Board Rey: Latest ✓						VIVADO.	A new RTL project named '2ynqOpenCL' will bu The default part and product family for the n Default Board 2YNO-7 F0726-02 Default Part: xr2010elg2251 Product: 2ynq-7000 Parkiage: clg225	a created. ow project:
Search: Q- Display Name	Reset All Fil	lters Board Rev	Part	I/O Pin Cou	nt File Ver		Speed Grade: -1	
ZedBoard Zyng Evaluation and Development Kit	em.avnet.com	d	xc7z020clg484-1	484	1.3			
ZVNO 7 TEO726-02	trenz biz	0.1	@vc7z010clg223-1	225	1.0			
ZVN0-7 TE0726-02M	trenz hiz	0.2M	% xc7z010clg225-1	225	1.2			
Artiv-7 AC701 Evaluation Platform	viliny com	1.1	@ xc7a200tfbr676-	2 676	1.2			
ZYNQ-7 ZC702 Evaluation Board	xilinx.com	1.0	*c7z020clg484-1	484	1.2			
4 100		< <u>B</u> ac	k Next >	inish	Cancel		To create the project, click Finish	ack Next > Finish Cancel

Figure 8: Project configuration wizard board selection.

This concludes the project configuration procedure.

3.2 Designing the system

Now we have entered Vivado and are presented with a "Project Manager" view, a "Project Summary" and the "Flow Navigator". In the Flow Navigator select "Create Block Design". The default name "design_1" is fine and we can keep it and just hit "OK". This should bring up a "Block Diagram View" as is shown in figure 9.



Figure 9: Vivado Block Diagram view.

In the Diagram view "design_1" we click the add IP button. The shape of this button should now be shown in the middle of the diagram view but can later be found in the vertical toolbar next to the diagram view. Click "add IP" and enter "zynq" into the search field. Select the "ZYNQ7 Processing System".



The diagram view should now contain a Zynq processing system as shown in figure 10. Not that there is a "Run Block Automation" link within the block diagram at this point. Hit this link and mark "All Automation" and then click Ok. The block automation dialog is shown in figure 11.



Figure 10: Vivado Block Diagram view with ZYNQ processing system.



Figure 11: Block automation dialog for the Processing System.

After allowing the block automation for the processing system to apply the default settings, the block diagram should look as in figure 12.



Figure 12: Block automation dialog for the Processing System.

Now it is time to add the vadd IP block to the design but before doing that we need to point out to Vivado where that IP can be found. Find the "IP settings" button in the toolbar within the Diagram view and click it. Then click the "Repository Manager" tab and the plus (+) symbol. Find the "impl" directory of the vadd_OpenCL vivado_hls project and click select. This process is outlined in figure 13.

Clicking "Select" should bring up the "Add Repository" dialog. Just click "OK" and then we are back to the list of IP Repositories, but now augmented with our recently added IP. Click "OK".

🛞 💿 Project Settings	🛞 🐵 Project Settings
IP General Image: General General Image: Gene	P General
OK Cancel	pply
🛞 🗊 IP Repositories	
Recent: C/home/joels/Vivado	- 🕹 🕲 🗶 🔌 🗐 🗷 😒
Directory: /home/joels/Vivado/vadd_OpenCL/solut	ion1/impl
<pre></pre>	
	Select Cancel

Figure 13: Outline of how to add our custom vadd IP to the IP repository.

	8 💿 Project Settings
Add Repository I repository was added to the project Repository Impl (/home/joels/Vivado/vadd_OpenCL/solution1) Ps:1 Interfaces:0 OK	Project Settings General Gen
	Refresh All

Figure 14: Outline of IP repository configuration continues.

With the IP repository configured we can add the vadd IP to the design. Click the "Add IP" button in the toolbar in the Diagram view. Type "vadd" in the search field and the "Vadd" IP should appear in the list window. Select it.



After adding the Vadd IP the block Diagram should look as in figure 15. Now we need to connect the Vadd unit to the processing system but in order to that we need to go into the processing system block and reconfigure it. If you look at the Vadd_0 unit it has a "s_axi_control" interface and a "m_axi_gmem" interface. These interfaces needs to be connected (as well as the clock and reset). Luckily much of this connecting can be done for us automatically, if only we configure the processing system block correctly.



Figure 15: Outline of IP repository configuration continues.

Double click on the ZYNQ7 Processing system in the in the diagram view. This should bring up a view of the internals of the processing system as below:



There are two interfaces that needs to be configured inside the processing system. the "32b GP AXI Master Ports" and the "32b GP AXI Slave Ports". Double click on the Master ports and configure according to the left side of figure 16. Then do the same for the Slave ports and the right side of figure 16.

😕 🕕 Re-customize I	P				🙆 🕕 Re-customize IF	P				
ZYNQ7 Processing	System (5.5)			4	ZYNQ7 Processing	Sys	tem (5.5)			1
👹 Documentation 🚳 Pro	esets 🚞 IP Location 🚳 Import XPS Settings				👹 Documentation 🚳 Pre	esets	🗀 IP Location 🚳 Import XPS Settings			
Page Navigator	PS-PL Configuration		g	iummary Report	Page Navigator «	PS	-PL Configuration			Summary Report
Zyng Block Design	Search: Q-				Zyng Block Design	+	Search: Qr			
PS-PL Configuration	Name	Select	Description		PS-PL Configuration	1 🗳	Name	Select	Description	
Peripheral VO Pins	General		7. u	_	Peripheral I/O Pins		© General			_
MiQ Configuration		0	Enable AXI Non Secure Transaction		MID Configuration		AXI Non Secure Enablement GP Slave AXI Interface	0 +	Enable AXI Non Secure Transaction	
al La C	∲+M AXI GP0 interface	1	Enables General purpose AXI master interface 0		at the Constant		-S AXI GP0 interface	R	Enables General purpose 32-bit AXI Slave interface 0	
Clock Configuration	I AXI GP1 interface		Enables General purpose AXI master interface 1		Clock Configuration		S AXI GP1 interface		Enables General purpose 32-bit AXI Slave interface 1	
DDR Configuration	GP Slave AXI Interface HP Slave AXI Interface				DDR Configuration		HP Slave AXI Interface ACP Slave AXI Interface			
SMC Timing Calculation	ACP Slave AXI Interface				SMC Timing Calculation		DMA Controller			
Interrupts	DMA Controller	-	- 11 M		Interrupts		PS-PL Cross Trigger interface		Enables PL cross trigger signals to PS and vice-versa	
	◎•PS-PL Cross Trigger Interface		Enables PL cross trigger signals to PS and vice-versa							
			ОК	Cancel					ОК	Cancel

Figure 16: Configuration of the AXI Master/Slave ports.

When this configuration of the processing system is completed the ZYNQ Processing System in the diagram view should show the newly added interfaces. Compare to figure 17.



Figure 17: Block diagram view after configuration of the processing system.

Most important here is to note that this change makes the "Run Connection Automation" link to appear at the top of the block diagram view. Click this link and make the following selections:



At this point the diagram view should look similar to figure 18. Notice how two AXI interconnects have been automatically added to the design and connects to the processing system and to the vadd unit. One of these interfaces connect to the control port on the vadd unit and is used to program the vadd unit control registers. The other interface is used by the vadd unit for memory accesses.



Figure 18: Block diagram view after configuration of the processing system.

At this stage the design is mostly complete. We do, however, need to go into the "Address editor" and do some small tweaks. Within the Address editor some address ranges are listed as excluded, we need to set these as included.

Plagram × MAddress Editor ×					000		Diagram × N Address Editor ×					
Cell	Slave Inter	face Base Name Offset Addre	ss Ran	ge High Address		-	Cell	Slave Interf	ace Base Name Offset Address	Range High Addr	855	
P-0 processing_system7_0							P-9 processing_system7_0					
Data (32 address bits : 0x4000000							Data (32 address bits : 0x40000000					
😂 9-9 vadd 0							P Vada 0	103				
Data m axi gmem (32 address bits	: 4G)					1 6	P M Data m axi gmem (32 address bits :	4(3)	C00 000 10 0-0000 0000	1004 - 0-0755 5555		
processing_system/_0	5_A00_GP0	GP0 DDR L0 0x0000 0000	128M	* 0x0/HF_HHH			- a processing system/ 0	5 A01 GPO	GP0_05PI_UN0x5000_0000	120H * 0x0/PF_PPP		
(b) Deckyled Address Segments (2)	5,00,000	GP0_QSP1_DN GIPC00_0000	1014	* WPOP_PPP			- se processing system7.0	S AN GPO	GP0 IOP 045800 0808	4M * OxFR3E FEFE		
was processing patent? 0	6 AM 680	GR0 10R 0v5000 0000	414	OVERIDE EELE			- is processing system7.0	S AN GPO	GPD M AXI GPD 0x4000 0000	1G * GYTEF FFFF		
- m processing system7 0	S AN GPO	GP0 M AXI GP0 0x4000 0000	16	OX7FFF FFFF								
												I

Now it is time to go through the process that in the end results in a bitstream that we can use to program the Zynq FPGA. This procedure is outlined below, in text, and following this in pictures:

- Validate the design: In the tools menu, click Validate design. This should finish with a message saying "Validation successful".
- Create HDL wrappers: This is done in the sources window by right clicking on "design_1" and selecting "Create HDL Wrappers".
- Save and run synthesis: In the flow navigator under synthesis, click run synthesis.
- Synthesis completed: In this dialog choose "Run Implementation".
- Implementation completes: Potentially reporting a great number of critical warnings. Fortunately these can be ignored. One should however check where the warnings come from. If there are warnings directly in relation to the IP you developed yourself it may be important.

- Implementation Completed dialog: Choose "Generate Bitstream".
- After generation of bitstream one can take a look at the implemented design.



OK

Cancel



🛞 Synthesis Completed	
Synthesis successfully completed. Next	Launch Run Critical Messages
 <u>Run Implementation</u> <u>Open Synthesized Design</u> <u>V</u>iew Reports 	 [Common 17-55] 'set_property' expects at least one object. [/home/joels/Vivado/ZynqOpenCL/ZynqOpenCL.srcs/sources_1/bd/ design_1/ip/design_1_processing_system7_0_//esign_1_processing_system7_0_(Resolution: if [get_<value>] was used to populate the object, check to make sure this command returns at least one valid object.</value> [Common 17-55] 'set_property' expects at least one object. [/home/joels/Vivado/ZynqOpenCL/ZynqOpenCL.srcs/sources_1/bd/ design_1/ip/design_1_processing_system7_0_//design_1_processing_system7_0_(Resolution: if [get_<value>] was used to populate the object, check to make sure this command returns at least one valid object.</value> [Common 17-55] 'set_property' expects at least one valid object.
Don't show this dialog again	OK Cancel Run Open Messages View
OK Cancel	

Figure 19: Synthesis completed and an example of critical warnings.

Implementation Completed	Bitstream Generation Completed
Implementation successfully completed.	Bitstream Generation successfully completed.
 O Open Implemented Design Generate Bitstream View Reports 	© Open Implemented Design View Reports Open <u>H</u> ardware Manager
Don't show this dialog again	Don't show this dialog again

Figure 20: Implementation complete and Bitstream generation completed



At this point we have completed the part of this guide that takes place in Vivado. The next step is to go into the file menu and select "Export Hardware" (include the bitstream) and then to "Launch SDK".

	🛞 🗊 Launch SDK				
😸 💷 Export Hardware	Launch software development tool.				
Export hardware platform for software development tools.					
Include bitstream	Exported location: 🛜 <local project="" to=""> 🔻</local>				
Export to: 🔂 <local project="" to=""></local>	Workspace: 🔂 <local project="" to=""> 🔹</local>				
OK Cancel	OK Cancel				

Figure 21: Export hardware and launch the SDK

3.3 Important details from the Address Editor

Before going into the part of the guide that takes place in the SDK we want to point out some details from the Address Editor. Make a note of the address mentioned for the s_axi_control. In this case this address is 0x43C00000. It is on this address and onwards that the control registers for the vadd hardware unit is mapped into the address space.

Add	dress Editor					_ 🗆 🖓 ×
a	Cell	Slave Interface	Base Name	Offset Address	Range	High Address
	♀-♀ processing_system7_0					
		[1G])				
鲁	Les vadd_0	s_axi_control	Reg	0x43C0_0000	64K	 0x43C0_FFFF
made	¢-⊈ vadd_0					
龖	∳- Ⅲ Data_m_axi_gmem (32 address bits :	4G)				
	processing_system7_0	S_AXI_GP0	GP0_DDR_LO	0x0000_0000	128M	 Ox07FF_FFFF
	processing_system7_0	S_AXI_GPO	GP0_QSPI_LIN	0xFC00_0000	16M	 OxFCFF_FFFF
	– m processing_system7_0	S_AXI_GP0	GP0_IOP	0xE000_0000	4M	 0xE03F_FFFF
	processing_system7_0	S AXI GPO	GP0_M_AXI_GP0	0x4000_0000	1G	 Ox7FFF_FFFF
		2 - Fax Eta / Cr				

4 Part 3: Xilinx SDK

When the Xilinx SDK has launched (after launching it from the File menu in Vivado) we are presented with a view like the left part of figure 22. Here we just click the "File" menu and "New" "Application Project". In the right part of figure 22 we name our application project "HelloOpenCL" and click next and select "hello world" then "Finish".

Now we need to perform one key piece of configuration to the Board Support Package, the "system.mss" file of the "HelloOpenCL_bsp". The configuration we need to change is the stdin/stdout under "Overview", "Standalone". Both stdin and stdout should be pointed to "ps7 uart 1" and not to uart 0 as per default.

Next we go into the Xilinx tools menu and clicks "Generate linker script". Here we want to make the heap larger. Find the "Heap Size" box and enter for example 33554432 (for 32mb). The default setting of 1KB will not be enough for what we are going to do.

Now it is time to write the C code that talks to the vadd unit. Edit the "helloworld.c" file in the "HelloOpenCL" project as listed in figure 25.

After writing the code we can right click on the "HelloOpenCL" project in the "Project Explorer" and choose "Debug as" and "Debug Configuration". In the debug configuration

😣 🖨 🗊 C/C++ - design_1_wrapper	_hw_platform_0/system.hdf - Xilinx SDK			😣 🗈 New Project				
1 · 2 0 6 0 · 5 · 6	5 # 🛛 🖬 🖬 🖸 🎯 + 🚳 + Ĉ + @ + 🌣	• O • 🧣 • 🔌 🤔	📌 💌 🗉	Application Project		-6		
9 • 6 • * • • • • B		Q Quick A	Access	Create a managed make	e application project.			
🏷 Project Explorer 🛛 🗖 🗖	🔓 system.hdf 🕴	- 0	₩ © ₩ 🗎 🗖		and a second			
E 😫 🔻 🎽	design_1_wrapper_hw_platform_0 Hardwa	re Platform	An outline is not available.	Project name: HelloOp	benCL]		
design_1_wrapper_hw_platform	Specification			🐷 Use default location				
drivers	Design Information		•	Location: //home/ioels	Location: //home/loels/Vivado/ZvngOpenCL/ZvngOpenCL sdk/HelloC			
G design_1_bd.tcl design_1_wrapper.bit c ps7 init opl c	Target FPGA Device: 7z010 Created With: Vivado 2015.4			Choose file s	ystem: default 💲			
<pre>ps_init_gpl.h ps7_init_gpl.h ps7_init.c</pre>	Created On: Sat Apr 23 21:12:56 2016 Address Map for processor ps7_cortexa9_0			OS Platform: standa	lone	:		
ps7_init.h	ps7_intc_dist_0 0xf8f01000 0xf8f01fff			Target Hardware				
ps7_init.html	ps7_gpio_0 0xe000a000 0xe000afff			Hardware Platform:	1: design_1_wrapper_hw_platform_0 🗘			
g ps/_init.tci	ps7_scutimer_0 0x18100800 0x18100811			Processor:	ps7 cortexa9 0	:		
. g systemato	ps7_scuwd(0 0xt8100620 0xt8100611 ps7_lctachec_0 0xt810000 0xt81000t ps7_scuc_0 0xt8100000 0xt81000t ps7_qspi_lnear_0 0xt6100000 0xt61111 ps7_pmu_0 0xt800000 0xt6800str1 ps7_afi_1 0xt8008000 0xt800str1 0xt8008000 0xt800str1			Target Software Language: Compiler: Board Support Packag	C C++ 32-bit : age: Create New HelloOpenCL_bsp			
👛 Target Connections 🛿 🔤	🖹 Pr 🛱 🕢 Ta 📮 Co 🔲 Pr 📮 S 🖓 🗖	🗐 SDK Log 🛱	🔒 📪 🗖		O Use existing			
 ▲ 40° ► ⇔ Hardware Server ► ⇔ Linux TCF Agent ► ⇔ QEMU TcfGdbClient 	0 items Description Re:	09:22:12 INFO : Lat 09:22:13 INFO : XSI 09:22:13 INFO : Pro	unching XSDB server: xsdb DB server has started succ ocessing command line opt:					
		·		•	< Back Next > Cancel	Finish		

Figure 22: Left: SDK just started. Right: new application project.

😣 💷 New Project		😑 😑 ເ C/C++ - HelloOpenCL_bsp	/system.mss - Xilinx SDK	
Templates	G	1 • 8 8 8 • 5 • 6	ii 🕃 🗱 📓 📓 🖬 🖸 🖆 ד 🚳 ד 🖻 ד 🚱 ד 🌾 ד 🛈 ד 🍕 ד 🔌	🧶 🛷 🝷 🔳 🗐
Create one of the available templates to g application project.	enerate a fully-functioning	9 · 6 · • • • • • 1 B	Q Quick A	ccess 🗈 🖻 🗟 C/C++
Available Templates:	5	Project Explorer 🛚 🗖	🖬 system.hdf 🔒 system.mss 🛛 🗖 🗖	An outline is not available.
Dhrystone Empty Application Hello World IwP Echo Server Memory Tests OpenAMP rests OpenAMP RPC Demo Peripheral Tests RSA Authentication App Zyng DRAM tests Zyng FSBL	Let s say Helio Worto In C.	 ✓ design_1_wrapper_hw_platform ► drivers ✓ design_1_bd.tdl I design_1_wrapper.bit I ps7_init_gpl.h I ps7_init_gpl.h I ps7_init.th Ø ps7_init.th Ø ps7_init.th Ø ps7_init.th Ø ps7_init.th Ø ps7_init.th Ø system.hdf ► MelloOpenCL ► MelloOpenCL_bsp 	HeiloOpenCL_bsp Board Support Package Modifythis BSP's Setting: Re-generate BSP Sources Target Information This Board Support Package is compiled to run on the following target. Hardware Specification: /home/joels/Vivado/ZynqOpenCL/ZynqOpenCL.sdk/des Target Processor: psr_cortexa9_0 Operating System Board Support Package OS. Name: standalone Version: S3 Description: Standalone is a simple, low-level software layer. It provides acce input and output, profiling, abort and exit. Documentation: standalone v_3 Peripheral Drivers Overview Source	
		👛 Target Connections 🛿 🔄 🖶 🧟	E Pr № 2 Ta ■ Co ■ Pr ■ S ■ □ ■ SDK Log №	
		 ► Hardware Server ► Linux TCF Agent ► QEMU TcfGdbClient 	012212 INF0 1 Lat 012212 INF0 3 Lat 012213 INF0 3 Lat	Inching XSUB Server: XSdb B server has started suc ocessing command line opt:
Sack	Next> Cancel Finish			

Figure 23: Board support package configuration.

select "Reset entire system" and "Program FPGA" then "Apply".

Now to start the application in Debug mode right click on "HelloOpenCL" select "Debug as" and "Launch on hardware". The ZynqBerry should now be connected and its LEDs will be on while the device is being programmed. The SDK will automatically enter into Debug mode and you can press "Resume" (F8) button to run. In order to see any output from the device you need to have a terminal link to it. On linux using the screen command works well: screen /dev/ttyUSB1. This part of the procedure is shown in figure 27.

😣 🗉 🛛 Board Support	Package Settings					😕 🐵 Generate a linker script				
Board Support Package Settings Control various settings of your Board Support Package.						Generate linker script Control your application's memory map.				
 ♥ Overview standalone ♥ drivers ps7_cortexa9_0 	Configuration for OS: sta Name stdin stdout • microblaze_exceptions • enable_sw_intrusive_profil	ndalone Value ps7_uart_1 ps7_uart_1 false false	Default none none false false	Type peripheral peripheral boolean boolean	Description stdin peripl stdout peri Enable Mici Enable S/W	Output Settings Project: HelioOpenCL Output Script: InCL/ZynqOpenCL.sdk/HelioOpenCL/src/Iscript.id Browse Modify project build settings as follows: Set generated script on all project build configurations Hardware Memory Map Memory Label Available Avail	Basic Advanced Place Code Sections in: ps7_ddr_0_S_AXL_BASEADDR v Place Data Sections in: ps7_ddr_0_S_AXL_BASEADDR v Place Heap and Stack in: ps7_ddr_0_S_AXL_BASEADDR v Heap Size: [33554432] Stack Size: 1 KB			
?				Cancel	ОК	0	Cancel Generate			

Figure 24: BSP Uart settings and the linker script

```
#include <stdlib.h>
#include "platform.h"
                                                                                                              *a_addr = (unsigned int)a;
                                                                                                             *a_addr = (unsigned int)s;
*b_addr = (unsigned int)b;
*c_addr = (unsigned int)c;
#include "xil_mmu.h"
#include "xil_cache.h"
#include "xil_cache_l.h"
                                                                                                             /* set the workgroup identity */
*wg_y = 0;
*wg_z = 0;
*wg_x = 0;
void print(char *str);
volatile char *control = (volatile char*)0x43C00000;
                                                                                                              *o_x = 0;
volatile int *wg_x = (volatile int*)0x43C00010;
                                                                                                             *o_y = 0;
*o_z = 0;
volatile int *wg_y = (volatile int*)0x43C00018;
                           = (volatile int*)0x43C00020;
= (volatile int*)0x43C00028;
volatile int *wg_z
volatile int *o_x
volatile int *o_y
                            = (volatile int*)0x43C00030;
= (volatile int*)0x43C00038;
                                                                                                             print("Status of control register: \n\r");
                                                                                                             volatile int *o_z
volatile int *a_addr = (volatile int*)0x43C00040;
volatile int *b_addr = (volatile int*)0x43C00048;
volatile int *c_addr = (volatile int*)0x43C00050;
                                                                                                                        _____ (i << i)
print("1");
} else {
                                                                                                                                   print("0");
                                                                                                                         }
#define WG_SIZE_X 128
#define WG_SIZE_Y 1
                                                                                                              3
                                                                                                             print("\n\r");
#define WG_SIZE_Z 1
                                                                                                              print("Starting OpenCL kernel execution\n\r");
int main()
                                                                                                              *control = *control | 1; /* start */
{
      init_platform();
                                                                                                             /* waiting for hardware to report "done" */
while (! ((*control) & 2));
print("DONE!\n\r");
      /* more initialization */
     Xil_SetTlbAttributes(0x43c00000,0x10c06); /* non cacheable */
     int *a;
int *b;
                                                                                                             Xil_DCacheInvalidate();
      int *c;
      int i;
                                                                                                              for (i = 0; i < WG_SIZE_X; i ++) {</pre>
      int ok = 1;
                                                                                                                        if (c[i] != 3) ok = 0;
                                                                                                             }
     a = (int*)malloc(WG_SIZE_X *sizeof(int));
b = (int*)malloc(WG_SIZE_X *sizeof(int));
                                                                                                             print("Success!\n\r");
} else {
      c = (int*)malloc(WG_SIZE_X *sizeof(int));
     print("Generating input data: \n\r");
for (i = 0; i < WG_SIZE_X; i ++) {
    a[i] = 1;
    b[i] = 2;
    c[i] = 0;
                                                                                                                        print("Error: Something went wrong!\n\r");
                                                                                                             }
                                                                                                             cleanup_platform();
return 0;
                                                                                                        }
      Xil_DCacheFlush();
```



😣 🗇 Debug Configurations				😕 😑 🕒 C/C++ - HelloOpenCL/sr	/helloworld.c - Xilinx SDK			
Create, manage, and run config	gurations		1	1 • E & B & B • • • •	₩ 6° \$\$ 80 8 8 8 5 6° + 6° + 6° + 6° +	\$ • 0 • Q • *	k 🙋 🛷 🔹 🗵 🗉	
Start Performance Analysis using s	system Debugger		26					
🕐 🐘 🗶 📄 🔅 🔻	Name: HelloOpenCL Debug				* Copyright (C) 2009 - 2014 Xilinx, Inc. A	<pre>% % % % % % % % % % % % % % % % % % %</pre>	B 12 R X + #	
type filter text	Target Setup Applica	ation 🗮 ATG Configuration 😔 Arg	guments 👼 Environment "4	design_1_wrapper_hw_platform	n ©/*			
HelioperCL belioperCL believe CL believ	Debug Type: Standalone Ap Connection: Local	plication Debug 1 New		 V ≅ HelloOpenCL V ∰ Binaries N includes M Debug V ≧ src 	This application configures UART 16550 t F57 UART (Zyng) is not initialized by th bootrom/bsp configures it to baud rate 1	■ stdio.h ■ platform.h ⊕ print(char*):void ● main():int		
	Bitstream file: Const Bitstream file: desig Initialization file: ps7_i PS Device: Auto PS Device: Auto Reset entire system Program FPGA	n_1_wrapper.bit nn_1_wrapper.bit nit.td Detect Summary of operations to be pp following operations will be pe 1. Resets entire system. Clears 2. Program PEGA fabric (PL).	Search Browse Generate. Search Browse Select erformed before launching the debugger. the FPGA fabric (PL).	Bindowerd C. Bartom_configh Battom.configh Battom.c Battom.b Script.id Beript.id Beript.id	<pre>i user Time Davo Aste surritise configurable only in NW de ps?uurr lis200 (configurable only in NW de ps?uurr lis200 (configurable bool r/, veid print(char *str); einc tamin() (init_platform();); }</pre>	isign (rom/bsp)		
	Run ps7_init	3. Runs ps7_init. 4. Runs ps7_post_config. Enable (Recommended to use this opti	es level shifters from PL to PS.	🖨 Target Connections 😫 📄 🖻	🖺 Pr 🗱 🖉 Ta 🖳 Co 🗔 Pr 🖳 S 🚍 🗖	SDK Log 😫	📭 🖷 🚥	
Filter matched 6 of 16 items				Be Hardware Server Be Linux TCF Agent Be QEMU TcFGdbClient	0 items 9:22:12 INF0 : Launching XSD 0 items 9:22:13 INF0 : SB server h 0 escription Re: 9:22:13 INF0 : Processing co			
3			Close Debu	/HelloOpenCL/src/helloworld.c				

Figure 26: Debug configuration and screen interaction with the ZynqBerry.



Figure 27: Debug the software and screen interaction.

4.1 C Code Walkthrough

Note that the code shown in figure 25 and gone through in this section, contains the line:

```
Xil_SetTlbAttributes(0x43c00000,0x10c06); /* non cacheable */
```

in the hope that this would make all read and writes to control registers bypass the cache. Newer versions of the Xilinx libraries have defined a set of names (in xil_mmu.h) for these attribute bit patterns. There the following definition can be found:

```
#define NORM_NONCACHE 0x11DE2
```

So let's for now assume that the correct way to set the registers as non-cacheable is the following: and that the values used in the c code listings are incorrect:

```
Xil_SetTlbAttributes(CONTROL, NORM_NONCACHE);
```

If anyone know the details of this in depth and want to share the knowledge, please write an email to bo.joel.svensson@gmail.com.

The code for interfacing with the generated hardware is given in full in figure 25 but is here given a step by step explanation.

The code starts out by including some headers. This is just shown here for completeness.

```
#include <stdlib.h>
#include "platform.h"
#include "xil_mmu.h"
#include "xil_cache.h"
#include "xil_cache_l.h"
```

void print(char *str);

The code below, declares names for the programming registers. The base address was for this was found in section 3.3 and the offsets to each specific register is found in section 2.4.

```
volatile char *control = (volatile char*)0x43C00000;
                     = (volatile int*)0x43C00010;
volatile int *wg_x
                     = (volatile int*)0x43C00018;
volatile int *wg_y
                     = (volatile int*)0x43C00020;
volatile int *wg_z
                     = (volatile int*)0x43C00028;
volatile int *o_x
volatile int *o_y
                     = (volatile int*)0x43C00030;
volatile int *o_z
                     = (volatile int*)0x43C00038;
volatile int *a_addr = (volatile int*)0x43C00040;
volatile int *b_addr = (volatile int*)0x43C00048;
volatile int *c_addr = (volatile int*)0x43C00050;
```

The workgroup size is 128 (in the x direction). This means that each "run" of the generated hardware will perform 128 element wise additions.

```
#define WG_SIZE_X 128
#define WG_SIZE_Y 1
#define WG_SIZE_Z 1
```

This also means that the smallest amount of additions we can perform using the vadd hardware is 128 and that we can only perform multiples of 128 additions by repeatedly launching work on the vadd hardware with different workgroup identities. This restriction comes the use of the "reqd_work_group_size(128,1,1)" attribute used in the implementation of vadd in vivado_hls. This attribute can be left out resulting in a more flexible (but less efficient) hardware implementation with a more complicated interface.

The main function starts out by performing some standard initialization but we also add a step that marks the range of memory containing the programming registers as "non cacheable".

```
int main()
{
    init_platform();
    /* more initialization */
    Xil_SetTlbAttributes(0x43c00000,0x10c06); /* non cacheable */
```

The following piece of code declares pointers and allocates memory for the input and output to the vadd computation. It also declares a counter variable i (used in some loops later on) and an ok status variable.

```
int *a;
int *b;
int *c;
int i;
int ok = 1;
a = (int*)malloc(WG_SIZE_X *sizeof(int));
b = (int*)malloc(WG_SIZE_X *sizeof(int));
c = (int*)malloc(WG_SIZE_X *sizeof(int));
```

Generate some input data and flush the cache to ensure that all the data we generated has been stored all the way to DRAM before launching the vadd computation.

The next step is to program the registers of the vadd unit and prepare for launching a workgroup. The workgroup id is set to (0,0,0).

```
*a_addr = (unsigned int)a;
*b_addr = (unsigned int)b;
*c_addr = (unsigned int)c;
/* set the workgroup identity */
*wg_y = 0;
*wg_z = 0;
*wg_x = 0;
*wg_x = 0;
*o_y = 0;
*o_y = 0;
*o_z = 0;
```

The next piece of code prints the contents of the control register. This serves no important purpose for the application but only provides a way to visually inspect that the control status (which should be "idle").

We instruct the vadd hardware to start computing by putting a one at bit position zero in the control register.

```
print("Starting OpenCL kernel execution\n\r");
*control = *control | 1; /* start */
```

And then we wait for the hardware to report done in bit position two.

```
/* waiting for hardware to report "done" */
while (! ((*control) & 2));
print("DONE!\n\r");
```

Xil_DCacheInvalidate();

After the hardware reports to be done, we invalidate the cache of the processing system in order to ensure that we will see the fresh data that the programmable logic has computed (without any involvement of the cache hierarchy, so the changes in memory are not yet visible to the ARM cores).

After that we can check the result for correctness.

And we are done.

cleanup_platform();
return 0;

}

5 Conclusion

We hope that following this guide has allowed you to run OpenCL on a Zynq device. Please send us feedback or questions.