Hello World – Using Microblaze Softcore

In this tutorial we will create a Vivado project and add a MicroBlaze softcore. For the programming of the softcore we will use Vitis. Later on we will load the program to the flash of the FPGA. If we are successful, we will receive "Hello World" on our terminal program.

Tools we need:

- Alchitry AU
- USB-C Cable
- Vivado 2022.2
- Vitis 2022.2
- Alchitry Loader 1.0
- A terminal program of your choice

Content

Part one: Creating the Block Design in Vivado	2
Create a new Project	2
Creating the Block Design	3
Adding Constraints	7
Generate Bitstream	8
Part two: Writing the Program in VITIS	10
Create a Platform Project	10
Create an Application Project	11
Part three: Loading the Configuration and the Program in the Flash	13

Part one: Creating the Block Design in Vivado

Create a new Project

Open VIVADO and create a new Project by clicking on "Create Project"



Create Project > Open Project > Open Example Project >

On the next page specify project name and location.

Project Name	
Enter a name for yo	ur project and specify a directory where the project data files will be stored.
Project name:	SimpleMicroBlaze
Project location:	F:/Workspace_Vivado/
🗹 Create projec	tsubdirectory
Project will be cr	eated at: F:/Workspace_Vivado/SimpleMicroBlaze

Next we select RTL Project and "Do not specify sources at this time", as we have no source file at this moment.

۲	RTL 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.
	☑ Do not specify sources at this time
	Project is an extensible Vitis platform

Than we have to select our FPGA. The Alchitry Au-Board is using the "Xc7a35tftg256-1".

S 🗸	(30 matches)	
I/O Pin Count	Available IOBs	LUT Elements
256	170	20800
236	106	20800
324	210	20800
	I/O Pin Count 256 236	I/O Pin Count Available IOBs 256 170 236 106

Click on "finish".

After some seconds you should see the Vivado develop environment.

Elle Edit Flow Iools Report	The second	lick Access										
Flow Navigator	PROJECT MANAGER - Simple/MicroBlaze											
PROJECT MANAGER												
O Settings	Sources		? _			ct Summary						
Add Sources	Q = Z = 0 + 12 = 0			٥	Over	view Dashboa	ard					
Language Templates	Design Sources Design Sources				Sett	ings Edit						
IP Catalog	Gimulation Sources Simulation 1				Proj	ject name: iect location:	SimpleMi	croBlaze ace_Vivado/Sin	mialicroBiaza			
✓ IP INTEGRATOR	> 🗇 Utility Sources					duct family:	Artix-7		- premier o briazo			
Create Block Design						ject part	xc7a35tt					
Open Block Design						rmodule name: get language:	Not defin Verilog	bd				
Generate Block Design						ulator language:	Mixed					
✓ SIMULATION					Synt	thesis				Implementat	ion	
Run Simulation	Hierarchy Libraries Compile Order				Stat		Not sta			Status:		
Y RTLANALYSIS	Properties		? _	o c x		ius. Isages:		reo rs or warnings		Messages:		
> Open Elaborated Design			-	÷ o	Part		xc7a35			Part		
						itegy: port Strategy:		Synthesis Defau Synthesis Defau		Strategy: Report Strate		
✓ SYNTHESIS						emental synthesis		tically selected		Incremental		tation
Run Synthesis												
 Open Synthesized Design 					DRC	Violations				Timing		
✓ IMPLEMENTATION	Select an object to see	e properties				Run Im	plementation	to see DRC res	ults		Rur	n Imel
Run Implementation												
> Open Implemented Design					Utilis	zation				Power		
Y PROGRAM AND DEBUG						Run S	unthesis to se	e utilization resu	dte		Run	n Imel
Generate Bitstream	Tcl Console Messages Log Reports	Design Runs	×									
Open Hardware Manager	9, ₹ ● ≪ ▶ ≫ +	%										
/ open hardware manager		WNS TNS	WHS THS	WBSS	TPWS	Total Power F	ailed Routes	Methodology	RQA Score	QoR Suggestions	LUT F	FI
	✓ ▷ synth_1 constrs_1 Not started											
	▷ impl_1 constrs_1 Not started											

Creating the Block Design

In Vivado the block design is used to create the top file by using graphic blocks. In the Project Manager we click on "Create Block Design".

FI	ow Navigator	-
~	PROJECT MANAGER	
	🔅 Settings	
	Add Sources	
	Language Templates	
	👎 IP Catalog	
~	IP INTEGRATOR	
	Create Block Design	
	Open Block Design	
	Generate Block Design	
-		

On the next dialog we just click on "OK".

🝌 Create Block Design	1	×
Please specify name	of block design.	4
<u>D</u> esign name:	design_1	\otimes
Directory:	Second to Project>	~
Specify source set:	🖻 Design Sources	~
?	ОК	Cancel

Now you see the empty diagram.

+ 🗠 🖋 🗹 🖈 C 🕒 Efault View	*
This design is empty. Press the 🕂 button to add IP.	
	+ ▶ ☑ ∴ ☑ □

As Vivado tells us, you can add IP Blocks by clicking on the plus "Add IP". For our design we need:

- MicroBlaze
- MicroBlaze Debug Module (MDM)
- AXI Interrupt Controller
- Clocking Wizard
- Processor System Reset
- AXI Uartlite

Add them all.

Diagram × Address Editor × Address Map ×		
$\mathbf{Q} \mid \mathbf{Q} \mid \boldsymbol{\Sigma} \mid \boldsymbol{\Sigma} \mid \boldsymbol{\Theta} \mid \mathbf{Q} \mid \boldsymbol{\Xi} \mid \boldsymbol{\oplus} \mid \mathbf{H} \mid \mathbf{P}_{\mathbf{V}}$	🖌 🗹 🖈 C 🖄 :	Default View 🗸
₱ Designer Assistance available. Run Block Automation Ru	in Connection Automation	
aux_reset_in peripheral_reset[0:0] mb_debug_sys_rst interconnect_aresetn[0:0] dm_lockedorginbaral_areacth[0:0]	clk_wiz_0 set clk_out1 k_in1 locked locking Wizard + INTEF + DEBU Cik Reset	

Afterwards there are all blocks lying around without any connection.

Some of these blocks need configurations to make. Luckily Vivado helps us here. Click on "Run Block Automation".

You can see, our MicroBlaze needs some configuration. Please give it 64KB of Local Memory. The preselected 8KB are not enough for the "printf" command which we need later. Even the size optimized "xil_printf" will not fit in 8KB of local memory.

Q X ↓ ✓ All Automation (1 out of 1 selected) ✓ ♥ microblaze_0	MicroBlaze Debug Module are added and connected	omation generates local memory of selected size, and caches can be configured. , Peripheral AXI Interconnect, Interrupt Controller, a clock source, Processor System Reset as needed. A preset MicroBlaze configuration can also be selected. ons can be found in the tooltips.
	Preset Local Memory Local Memory ECC Cache Configuration Debug Module Peripheral AXI Port Interrupt Controller Clock Connection	None 64KB None None Debus Colu Debus Colu Select the MicroBlaze cache size in kilobytes, ranging from 4KB to 64KB. Enab Selecting None disables the caches and the AXI cache interfaces. Larger caches give better performance, but use more on-chip memory. You can open the MicroBlaze Configuration Wizard after running the automation //clk_Y to choose other cache sizes.

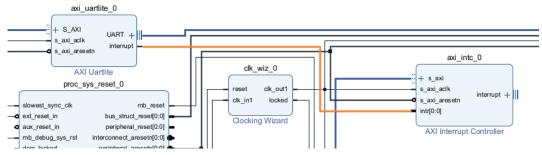
After clicking on "OK" you can see that Vivado created some connections.

Next click on "Run Connection Automation" and select all and quit with "OK".

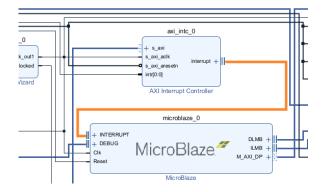
🍌 Run Connection Automation		×
Automatically make connections in your design the right.	by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on	4
Q X X All Automation (6 out of 6 selected) V Y * avi_intc_0 V * sai V	Select an interface pin on the left panel to view its options	
•	OK	ncel

Vivado did a great job! Almost everything is now connected. However, our interrupt line is missing. If the AXI Uartlite receives a signal it will generate an interrupt flag. We are not using the receiving function in this tutorial. However, I am sure you will use it later.

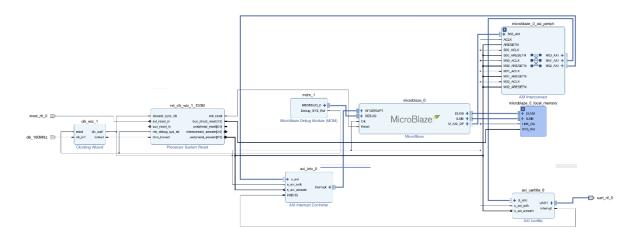
So, connect the interrupt output of the AXI Uartlite with the AXI Interrupt Controller:



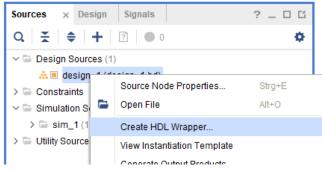




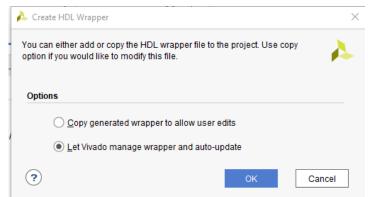
Our diagram is finish and should look like this:



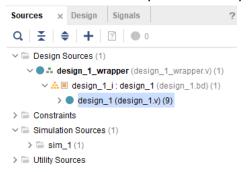
Now, that we have our diagram, we need one more step to generate our top file. Therefore right click on **Design_1->Create HDL Wrapper...**



This wrapper translates the diagram to the top file with its Verilog design and automaticly keeps it up to date.



In the Sources-Window you can see why the call it wrapper:



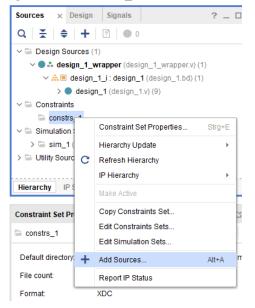
If you open the design_1.v, you can see the typical Verilog design of the top file:

•	
13	module design_1
14	(clk_100MHz,
15	reset_rtl_0,
16	uart_rtl_0_rxd,
17	<pre>uart_rtl_0_txd);</pre>
18	(* X_INTERFACE_INFO = "xilinx.com:signal:clock:1.0 (
19	(* X_INTERFACE_INFO = "xilinx.com:signal:reset:1.0]
20	(* X_INTERFACE_INFO = "xilinx.com:interface:uart:1.)
21	(* X_INTERFACE_INFO = "xilinx.com:interface:uart:1.)
22	
23	<pre>wire axi_intc_0_interrupt_INTERRUPT;</pre>
24	<pre>wire axi_uartlite_0_UART_RxD;</pre>
25	<pre>wire axi_uartlite_0_UART_TxD;</pre>
26	<pre>wire axi uartlite 0 interrupt;</pre>

Adding Constraints

Now that our block diagram and top file are finished, we need a constraint file to specify our inputs and outputs.

Right click on Constr_1->Add Sources...



On the next windows please create a new file.

Specify constraint set: 📮 constrs_1 (Active)
+, - + +	Create a new constraints file and add it to your project
Copy constraints files into project	Add Files Create File
•	< <u>B</u> ack <u>Next></u> <u>Finish</u> Cancel

Open Alchitry.xdc and add these lines:

set_property PACKAGE_PIN N14 [get_ports clk_100MHz]
set_property IOSTANDARD LVCMOS33 [get_ports clk_100MHz]
set_property PACKAGE_PIN P6 [get_ports reset_rtl_0]
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property PACKAGE_PIN P15 [get_ports uart_rtl_0_rxd]
set_property PACKAGE_PIN P16 [get_ports uart_rtl_0_rxd]
set_property IOSTANDARD LVCMOS33 [get_ports uart_rtl_0_rxd]
set_property IOSTANDARD LVCMOS33 [get_ports uart_rtl_0_rxd]

I guess these lines are not unfamiliar to you ...

And done!

Generate Bitstream

The last step in Vivado is to generate a bitstream. In the Flow Navigator click on "Generate Bitstream"

- ✓ PROGRAM AND DEBUG
 - Senerate Bitstream
 - > Open Hardware Manager

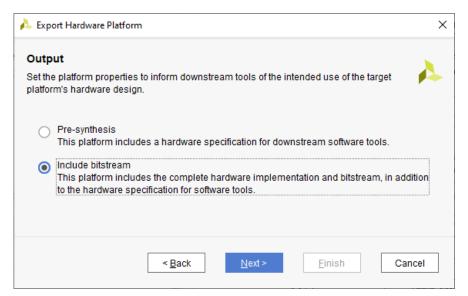
In the top right corner you can see what Vivado is doing at the moment. Synthesis, Implementation, ... and after some minutes it will state "write_bitstream Complete".

write_bitstream Complete

Now, we have to export our hardware design, so that we can use it in VITIS. Click on **File->Export->Export Hardware Platform**.



Select "Include bitstream".



On the next page you can see were the XSA file will be stored.

À Export Hardwa	re Platform	×
Files Enter the name o stored.	f your hardware platform file, and the directory where the XSA file will be	4
<u>X</u> SA file name:	design_1_wrapper	۲
<u>E</u> xport to:	F:/Workspace_Vivado/SimpleMicroBlaze	• • • •
	The XSA will be written to: F:\Workspace_Vivado\SimpleMicroBlaze\design	1_1
	< <u>B</u> ack <u>Next</u> > <u>F</u> inish Ca	incel

Click on Finish

Done!

Part two: Writing the Program in VITIS

Create a Platform Project

First we have to create a Platform Project. Open Vitis and click on File->New->Platform Project...



Click Next

On the next dialog we click on "Browse" and navigate to our XSA-File

→ * ↑	npleMicroBlaze	ر ٽ v	SimpleMicroBlaz	e durch
nisieren 🔻 Neuer Ordner				
^ Name	Änderungsdatum	Тур	Größe	
SimpleMicroBlaze.cache	10.04.2023 22:28	Dateiordner		
SimpleMicroBlaze.gen	10.04.2023 22:17	Dateiordner		
SimpleMicroBlaze.hw	10.04.2023 21:47	Dateiordner		
SimpleMicroBlaze.ip_user_files	10.04.2023 21:47	Dateiordner		
SimpleMicroBlaze.runs	10.04.2023 22:28	Dateiordner		
SimpleMicroBlaze.sim	10.04.2023 21:47	Dateiordner		
SimpleMicroBlaze.srcs	10.04.2023 22:11	Dateiordner		
design_1_wrapper.xsa	10.04.2023 22:45	XSA-Datei	99 KB	
v				
Dateiname: design_1_wrapp	AC V C 3	~ *	.xsa;*.dsa;	

Click on "Finish"

In the Explorer section, we can see our Platform project.

✓
> 🗁 export
> 🗁 hw
> 🔁 logs
> 🗁 microblaze_0
🗁 resources
🚽 platform.spr
platform.tcl

It is out of date. So select it, and built it. Project->Build Project.

Create an Application Project

Second, we need our application project. Click on **File->New->Application Project...** On the second dialog, we select our Hardware Platform:

Find: 🕂 Add 🌣 Manage							
Name	Board	Flow	Vendor	Path			
📑 Alchity [custom]		Embedded SW Dev	xilinx	F:\Workspace_Vitis\Alchity\export\Alchity\Alchity.xpfm			
📑 Simple_Microblaze_HW [custom]		Embedded SW Dev	xilinx	F:\Workspace_Vitis\Simple_Microblaze_HW\export\Simple_Mic			
📑 SimpleMircoBlaze_Platform [custom]		Embedded SW Dev	xilinx	F:\Workspace_Vitis\SimpleMircoBlaze_Platform\export\SimpleM			
📑 Tutorial_MicroBlaze [custom]		Embedded SW Dev	xilinx	F:\Workspace_Vitis\Tutorial_MicroBlaze\export\Tutorial_MicroB			

On the next dialog, we give it a name:

✓ New Application Project

Application Project Details

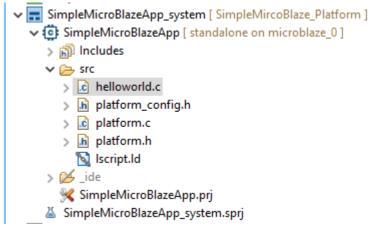
Specify the application project name and its system project properties

Application project name:	SimpleMicroBlazeApp
System Project Create a new system pr	oject for the application or select an existing one from the workspace ()
Select a system project	System project details
+ Create new	System project name: SimpleMicroBlazeApp_system

On the last Dialog, we select the template: "Hello World" and click on Finish.

🚽 Ne	w Application Project				×
Templ	ates			•	••
Select	t a template to create your project.			L	
Availa	ble Templates:				
Find:			Hello World		
V Er	mbedded software development templates	^	Let's say 'Hello World' in C.		
	Dhrystone				
	Empty Application (C++)				
	Empty Application(C)				
	Hello World				
	IwIP Echo Server				
	IwIP TCP Perf Client				
	IwIP TCP Perf Server				
	IwIP UDP Perf Client				
	IwIP UDP Perf Server	\sim			

The helloworld.c file is located under src:



As you can see this is already a working program.

```
47
48 #include <stdio.h>
49 #include "platform.h"
50 #include "xil_printf.h"
51
52
53⊖ int main()
54 {
        init_platform();
55
56
        print("Hello World\n\r");
57
58
        print("Successfully ran Hello World application");
59
        cleanup_platform();
60
        return 0;
61 }
62
```

Built it.

Part three: Loading the design and the program into the flash

Now that we have our hardware design and our application software, we have to create the binary file which we can upload to our FPGA.

Still in Vitis and still our application project selected click on Xilinx->Program Device.

A dialog opens. In the Software Configuration-Section, open the dropdown menu and click on "Browse..." (Sometimes there is a double click necessary, otherwise the browse window will not appear ... a bug?)

🚽 Program Devi	ice				×
Program Device	2				→
Specify the bitst	ream and the ELF file	s that reside in BRAM memory			<u>-</u>
Project:	GCode_GRBL_system	n	~		
Connection:	Local		~	New	
Device:	Auto Detect			Select	
Bitstream/PDI:	\${project_loc:GCod	e_GRBL}/_ide/bitstream/design_1_wrappe	r.bit	Search	Browse
Partial Bitstrea	im				
BMM/MMI File:	\${project_loc:GCod	e_GRBL}/_ide/bitstream/design_1_wrappe	r.mmi	Search	Browse
Software Config	uration				
Processor		ELF/MEM File to Initialize in Block RAM			
microblaze_0		Browse	~		
		bootloop <none></none>			
		Browse			
Skip Revision C	back				
	HELK				
			Generate	Program	Cancel

Navigate to the elf-File and open it:

┥ Select E	LF/MEM file				×
$\leftarrow \rightarrow$	✓ ↑	lazeApp → Debug	~ (・ Debug durchsuch	h
Organisier	ren 🔻 Neuer Ordner				?
^ ۵	Name	Änderungsdatum	Тур	Größe	
	src	11.04.2023 21:28	Dateiordner		
	SimpleMicroBlazeApp.elf	11.04.2023 21:28	ELF-Datei	103 KB	
	Dateiname: SimpleMicroBlazeApp.el	f	× *.e	elf	~
				Ö <u>f</u> fnen Abbrechen	

Click on "Generate"

Software Configuration	
Processor	ELF/MEM File to Initialize in Block RAM
microblaze_0	F:\Workspace_Vitis\SimpleMicroBlazeApp\Debug\Si
Skip Revision Check	
	Generate Program Cancel

Now our bitstream is ready to upload into our FPGA. So connect your Alchitry-Board to your computer. But before we upload it, let us start our terminal program and connect to our Alchitry-Board. As you have seen in helloworld.c it will only send once "Hello World". I do want to miss it.

💤 HTerm 0.8.9 - [hterm.cfg]								
File Options View Help									
Disconnect Port C	OM9	~ R	Baud 9600	✓ Data 8 √	Stop 1	V Parity None	e ~ 🗆	CTS Flov	v control
Rx 0 Re	eset Tx		0 Reset Co	unt 0 🔹	0	Reset New	line at CR	~	Show n
Clear received	Hex []Dec 🗌 Bin	Save output	Clear at	0 🔺 🛛 Nev	vline every 0 haracters	•	Autoscr	oll 🗌 Show
Sequence Overview X	Received D	ata							
	1 5	10 15	20 25	30 35	40 45	50 55	60	65	70 7

Now we are prepared to see the "Hello World". Open Alchitry Loader.

Alchitry Loader V1.0.0	_		×
		Open E	Bin File
Board: Alchitry Au Alchitry Cu		Flash E	EPROM
✓ Program Flash		Era	se
Status:		Prog	ram

Click on "Open Bin File" change to "All Files" and navigate to download.bit.

♥ Öffnen ×								
$\leftarrow \ \rightarrow$	✓ ↑ Workspace_Vitis → SimpleMicroB) $ ho $ bitstream	durchsuchen					
Organisie	eren 🔻 Neuer Ordner				==			
_ ^	Name	Änderungsdatum	Тур	Größe				
1	design_1_wrapper.bit	11.04.2023 21:25	BIT-Datei	470 KB				
	📄 design_1_wrapper.mmi	11.04.2023 21:25	MMI-Datei	6 KB				
	download.bit	11.04.2023 21:45	BIT-Datei	477 KB				
[
<u> </u>								
	Dateiname: download.bit			✓ All Files	~			
				Ö <u>f</u> fnen	Abbrechen			

Click on Program:

Alchitry Loader V1.0.0 —		×
$\fbox{\label{eq:constraint} F:\Workspace_Vitis\SimpleMicroBlazeApp_ide\bitstream\download.bi}}$	Open B	in File
Board: Alchitry Au Alchitry Cu	Flash EE	PROM
Program Flash	Eras	se
Status: Done.	Progr	ram

And

🗗 HTerm 0.8.1beta - [htern	n.cfg]
File Options View Help	0
Disconnect Port CO	M9 V R Baud 9600 V Data 8 V Stop 1 V Parity None V
Rx 54 Re	eset Tx 0 Reset Count 0 1 Reset Newline at CR
🕴 Clear received 🕴 🗹 Asci	i Hex Dec Bin Save output V Clear at O V Kewline every O V Autoscr
Sequence Overview X	Received Data
	1 5 10 15 20 25 30 35 40 45 50 55 60 65 _μ Hello World _{μμ}
	Successfully ran Hello World application

Success!!