VHDL | FPGA synthesis

Environment

Nous utiliserons la suite logicielle **Vivado** de **Xilinx** (now AMD) qui regroupe sous une même interface graphique un ensemble intégrant les outils de compilation, élaboration, synthèse et simulation de code HDL. A noter également la présence de l'outil **Vitis_HLS** qui permet l'exploration architecturale à partir de code C/C++ En complément, nous disposons également du simulateur **QuestaSim** (formerly known as **modelsim**).

Enfin, accessibles à tous sans licence, les outils **ghdl** et **gtkwave** vous permettent vos premiers pas dans l'univers du VHDL.

Quelques fonctions de conversions :

•	conv_std_logic_vector (integer, size,	$\Rightarrow std_logic_vector$	use ieee.std_logic_arith.all
•	conv_integer (std_logic_vector)	<i>⇒ integer</i>	use ieee.std_logic_unsigned.all
•	to_stdlogicvector (bit_vector)	⇒ std_logic_vector	use ieee.std_logic_1164.all
•	F <= std_logic(signal A or signal B)	⇒ std_logic	conversion en std_logic

Design Flow

- **Compilation** des entités, architectures, [configurations] et d'une architecture de test.
- Élaboration de l'architecture de test précédemment compilée générant un snapshot.
- Simulation du snapshot.

GHDL & GTKwave for behavioural simulation

* Compilation ghdl -a --ieee=synopsys -fexplicit <packages.vhd> <components.vhd> <testbench.vhd>
* Elaborate ghdl -e --ieee=synopsys -fexplicit testbench
* Run simulation ghdl -r --ieee=synopsys -fexplicit testbench --wave=testbench.ghw
* View results gtkwave testbench.ghw



Xilinx series 7 (Zynq, Spartan7 ...) *e.g boards like Zybo, Pynq* ...

Xilinx series 6 (Spartan6) e.g boards like Basys2 or Basys3 (for older designs)

Links

Master Secil/Siame VHDL <u>https://moodle.univ-tlse3.fr/mod/page/view.php?id=456296</u> (you'll find some useful links intended to speed up your learning curve)

VHDL course https://secil.univ-tlse3.fr/teaching/francois/UE-VHDL/VHDL_course.pdf

VHDL 4 FPGA synthesis practical exercises (this file) https://secil.univ-tlse3.fr/teaching/francois/UE-VHDL/VHDL_practical_exercises.pdf

Table of contents

Environment	1
GHDL & GTKwave for behavioural simulation	1
Links	2
Xilinx Vivado setup	5
additional IP blocks	5
Practical exercises base files	7
TP1 - Pulse generator	8
project setup	8
pulse_gen design	12
behavioural simulation	
pulse_gen generic parameter(s)	14
design synthesis	15
post-synthesis simulation	15
TP2 - Pulse generator synthesis	
top-level synthesis and generic parameters	16
comment-out generic map	17
post-synthesis timing simulation	18
pulse_gen with synchronous RST	18
TP3 - Pulse generator constraints	19
Timing constraints	
pluse_gen unconstrained paths	
Synthesis Constraints Wizard	21
Implementation Constraints Wizard	
Full timings simulation	23
[master2] Bitstream generation and hardware download	23
TP4 - Synthesizable Log2 function	
log2 @ cpu_package	24
log2 hardware component	
[master2] optimized log2 component	
TP5 - Registers bank for Risc processor	27
generic parameters	27
Behavioural simulation	27
Registers Bypass design and simulation	
Synthesis with constraints	28
[master2] BRAM inference and maximum reachable frequency	
TP6 - Memory bank for RISC processor	
Simulation	
[master2] bitstream & JTAG debug	
TP7 - dual ports FIFO	
Behavioural simulation	
IP8 - BLUG CONTROLLER.	
Use case	

Links	
[M2] Zybo Z7-20 board	
Links	40
[M2] PWM	41
Behavioural simulation	41
Synthesis and Implementation	42
Post-synthesis simulation	
IO planning (implementation)	
Bitstream generation	
Hardware Manager	
[optional] Bitstream readback	
Download bitstream	
Tests 🙂	
[M2] my AXI-enabled PWM IP	
Links	49
Editing an IP from main project	
IP customization	51
[optional] AXI simulation with the AXI VIP	55
Block design	
Bitstream generation and export hardware	
Launch Vitis IDE	60
Compilation	65
	65
Zybo board JTAG mode	
[M2] AXI IP encoder	
[M2] AXI IP encoder [M2] PicoRV32	
[M2] AXI IP encoder [M2] PicoRV32	67 68
[M2] AXI IP encoder [M2] PicoRV32 Tp1-1 : Banc Registres double port lecture pour µP RISC	67 67 68 69 70
[M2] AXI IP encoder [M2] PicoRV32 Tp1-1 : Banc Registres double port lecture pour µP RISC Tp1-2 : Simulation Banc Registres	67
[M2] AXI IP encoder [M2] PicoRV32 Tp1-1 : Banc Registres double port lecture pour μP RISC Tp1-2 : Simulation Banc Registres Tp1-3 : Banc Registres générique	67 68 69 70 70 73
[M2] AXI IP encoder [M2] PicoRV32 Tp1-1 : Banc Registres double port lecture pour μP RISC Tp1-2 : Simulation Banc Registres Tp1-3 : Banc Registres générique Tp1-4 : Bypass D→Q	67 68 69 70 70 73 73
[M2] AXI IP encoder [M2] PicoRV32 Tp1-1 : Banc Registres double port lecture pour μP RISC Tp1-2 : Simulation Banc Registres. Tp1-3 : Banc Registres générique Tp1-4 : Bypass D→Q Tp2-1 : Banc Mémoire pour μP RISC	67 68 69 70 70 73 73 73 78
[M2] AXI IP encoder [M2] PicoRV32. Tp1-1 : Banc Registres double port lecture pour μP RISC. Tp1-2 : Simulation Banc Registres. Tp1-3 : Banc Registres générique. Tp1-4 : Bypass D→Q. Tp2-1 : Banc Mémoire pour μP RISC. Tp2-2 : Simulation Banc Mémoire.	67 68 69 70 70 70 73 73 73 73 73
[M2] AXI IP encoder	67 68 69 70 70 73 73 73 73 78 79 82
[M2] AXI IP encoder [M2] PicoRV32. Tp1-1 : Banc Registres double port lecture pour μ P RISC. Tp1-2 : Simulation Banc Registres. Tp1-3 : Banc Registres générique. Tp1-4 : Bypass D→Q. Tp2-1 : Banc Mémoire pour μ P RISC. Tp2-2 : Simulation Banc Mémoire. Tp3-1 : Additionneur Carry Look-Ahead. Tp3-2 : Simulation Adder_CLA.	67 68 69 70 70 70 73 73 73 73 73 73 73 73 73 73 73 73 78 79 82 82
[M2] AXI IP encoder [M2] PicoRV32. Tp1-1 : Banc Registres double port lecture pour μ P RISC. Tp1-2 : Simulation Banc Registres. Tp1-3 : Banc Registres générique. Tp1-4 : Bypass D→Q. Tp2-1 : Banc Mémoire pour μ P RISC. Tp2-2 : Simulation Banc Mémoire. Tp3-1 : Additionneur Carry Look-Ahead. Tp3-2 : Simulation Adder_CLA. Tp4-1 : Unité Arithmétique Logique (ALU).	67 68 69 70 70 73 73 73 73 73 78 79 82 82 82 82 82
[M2] AXI IP encoder [M2] PicoRV32 Tp1-1 : Banc Registres double port lecture pour μ P RISC Tp1-2 : Simulation Banc Registres Tp1-3 : Banc Registres générique Tp1-4 : Bypass D→Q Tp2-1 : Banc Mémoire pour μ P RISC Tp2-2 : Simulation Banc Mémoire Tp3-1 : Additionneur Carry Look-Ahead Tp3-2 : Simulation Adder_CLA Tp4-1 : Unité Arithmétique Logique (ALU) Tp4-2 : Simulation ALU	67 68 69 70 70 73 73 73 73 73 73 73 73 73 73 73 73 73
[M2] AXI IP encoder [M2] PicoRV32. Tp1-1 : Banc Registres double port lecture pour μ P RISC. Tp1-2 : Simulation Banc Registres. Tp1-3 : Banc Registres générique. Tp1-4 : Bypass D→Q. Tp2-1 : Banc Mémoire pour μ P RISC. Tp2-2 : Simulation Banc Mémoire. Tp3-1 : Additionneur Carry Look-Ahead. Tp3-2 : Simulation Adder_CLA. Tp4-1 : Unité Arithmétique Logique (ALU). Tp4-2 : Simulation ALU. Tp5-1 : Banc Mémoire pour μ P RISC.	67 68 69 70 70 73 73 73 73 73 73 82 82 82 82 82 82 82 82 82 82
[M2] AXI IP encoder. [M2] PicoRV32. Tp1-1 : Banc Registres double port lecture pour μ P RISC. Tp1-2 : Simulation Banc Registres. Tp1-3 : Banc Registres générique. Tp1-4 : Bypass D→Q. Tp2-1 : Banc Mémoire pour μ P RISC. Tp2-2 : Simulation Banc Mémoire. Tp3-1 : Additionneur Carry Look-Ahead. Tp3-2 : Simulation Adder_CLA. Tp4-1 : Unité Arithmétique Logique (ALU). Tp4-2 : Simulation ALU. Tp5-1 : Banc Mémoire pour μ P RISC. Tp5-2 : Simulation Banc Mémoire.	67 68 69 70 70 73 73 73 73 73 73 73 73 73 73
[M2] AXI IP encoder [M2] PicoRV32. Tp1-1 : Banc Registres double port lecture pour μ P RISC. Tp1-2 : Simulation Banc Registres. Tp1-3 : Banc Registres générique. Tp1-4 : Bypass D→Q. Tp2-1 : Banc Mémoire pour μ P RISC. Tp2-2 : Simulation Banc Mémoire. Tp3-1 : Additionneur Carry Look-Ahead. Tp3-2 : Simulation Adder_CLA. Tp4-1 : Unité Arithmétique Logique (ALU). Tp4-2 : Simulation ALU. Tp5-1 : Banc Mémoire pour μ P RISC. Tp5-2 : Simulation Banc Mémoire. Tp5-1 : Banc Mémoire pour μ P RISC. Tp5-2 : Simulation Banc Mémoire. Tp6-1 : DATAPATH.	67 68 69 70 70 73 73 73 73 73 78 79 82 82 82 82 82 82 82 90 91 94
[M2] AXI IP encoder [M2] PicoRV32. Tp1-1 : Banc Registres double port lecture pour μ P RISC. Tp1-2 : Simulation Banc Registres. Tp1-3 : Banc Registres générique. Tp1-4 : Bypass D \rightarrow Q. Tp2-1 : Banc Mémoire pour μ P RISC. Tp2-2 : Simulation Banc Mémoire. Tp3-1 : Additionneur Carry Look-Ahead. Tp3-2 : Simulation Adder_CLA. Tp4-1 : Unité Arithmétique Logique (ALU). Tp4-2 : Simulation ALU. Tp5-1 : Banc Mémoire pour μ P RISC. Tp5-2 : Simulation Banc Mémoire. Tp5-1 : Banc Mémoire pour μ P RISC. Tp5-2 : Simulation Processeur RISC.	67 68 69 70 70 73 73 73 73 73 73 73 73 73 73
[M2] AXI IP encoder. [M2] PicoRV32. Tp1-1 : Banc Registres double port lecture pour μ P RISC. Tp1-2 : Simulation Banc Registres. Tp1-3 : Banc Registres générique. Tp1-4 : Bypass D→Q. Tp2-1 : Banc Mémoire pour μ P RISC. Tp2-2 : Simulation Banc Mémoire. Tp3-1 : Additionneur Carry Look-Ahead. Tp3-2 : Simulation Adder_CLA. Tp4-1 : Unité Arithmétique Logique (ALU). Tp4-2 : Simulation ALU. Tp5-1 : Banc Mémoire pour μ P RISC. Tp5-2 : Simulation Banc Mémoire. Tp5-1 : Banc Mémoire pour μ P RISC. Tp5-2 : Simulation ALU. Tp5-1 : Banc Mémoire pour μ P RISC. Tp5-2 : Simulation Banc Mémoire. Tp6-1 : DATAPATH. Tp6-2 : Simulation Processeur RISC. Tp7-1 : Unité de Contrôle.	67 68 69 70 70 73 73 73 73 73 73 73 73 73 73
[M2] AXI IP encoder [M2] PicoRV32 Tp1-1 : Banc Registres double port lecture pour μ P RISC Tp1-2 : Simulation Banc Registres Tp1-3 : Banc Registres générique Tp1-4 : Bypass D \rightarrow Q Tp2-1 : Banc Mémoire pour μ P RISC Tp2-2 : Simulation Banc Mémoire Tp3-1 : Additionneur Carry Look-Ahead Tp3-2 : Simulation Adder_CLA Tp4-1 : Unité Arithmétique Logique (ALU) Tp4-2 : Simulation ALU Tp5-1 : Banc Mémoire pour μ P RISC Tp5-2 : Simulation Banc Mémoire Tp6-1 : DATAPATH Tp6-2 : Simulation Processeur RISC Tp7-1 : Unité de Contrôle Tp8-1 : FIFO double port	67 68 69 70 70 73 73 73 73 73 73 73 73 73 73
[M2] AXI IP encoder [M2] PicoRV32 Tp1-1 : Banc Registres double port lecture pour μP RISC Tp1-2 : Simulation Banc Registres. Tp1-3 : Banc Registres générique. Tp1-4 : Bypass D→Q. Tp2-1 : Banc Mémoire pour μP RISC Tp2-2 : Simulation Banc Mémoire. Tp3-1 : Additionneur Carry Look-Ahead. Tp3-2 : Simulation Adder_CLA. Tp4-1 : Unité Arithmétique Logique (ALU). Tp4-2 : Simulation ALU. Tp5-1 : Banc Mémoire pour μP RISC. Tp5-2 : Simulation Banc Mémoire. Tp5-1 : Banc Mémoire pour μP RISC. Tp5-2 : Simulation ALU. Tp5-1 : Banc Mémoire pour μP RISC. Tp6-2 : Simulation Banc Mémoire. Tp6-1 : DATAPATH. Tp6-2 : Simulation Processeur RISC. Tp7-1 : Unité de Contrôle. Tp8-1 : FIFO double port. Tp8-2 : Simulation FIFO double port.	67 68 69 70 70 73 73 73 73 73 73 73 73 73 73
Zybo board JTAG mode.[M2] AXI IP encoder.[M2] PicoRV32.Tp1-1 : Banc Registres double port lecture pour μ P RISC.Tp1-2 : Simulation Banc Registres.Tp1-3 : Banc Registres générique.Tp1-4 : Bypass D→Q.Tp2-1 : Banc Mémoire pour μ P RISC.Tp2-2 : Simulation Banc Mémoire.Tp3-1 : Additionneur Carry Look-Ahead.Tp3-2 : Simulation Adder_CLA.Tp4-1 : Unité Arithmétique Logique (ALU).Tp5-1 : Banc Mémoire pour μ P RISC.Tp5-1 : Banc Mémoire pour μ P RISC.Tp5-2 : Simulation ALU.Tp5-1 : Banc Mémoire pour μ P RISC.Tp5-2 : Simulation Processeur RISC.Tp6-1 : DATAPATH.Tp6-2 : Simulation Processeur RISC.Tp7-1 : Unité de Contrôle.Tp8-1 : FIFO double port.Tp8-2 : Simulation FIFO double port.Tp8-1 : Mémoire Statique Synchrone.	67 68 69 70 70 73 73 73 73 73 73 73 73 73 73

Xilinx Vivado setup

Vivado is part of the '**Hardware targets**' tools from Xilinx ecosystem. It enables you to design FPGA's content through its UI, to synthesise and to simulate them.

Note: it's also possible to simulate your design with external tools like **modelsim** ... some additional setup and extra care are required.

According to the envisioned targets, there exists two flavours:

- Xilinx ISE ⇒ for series 6 and older FPGA
- Xilinx Vivado \Rightarrow for series 7 and newer boards

We'll make use of the **Xilinx Vivado** tools in combination with our **Zybo**, **Zybo-z7** and **Pynq** boards. For more advanced designs, for example those that make use of the embedded ARM processor (Hard IP¹), you'll need to have access to Intellectual Property blocks coming both from Xilinx and third-party designers.

Regarding the boards support files, extra care has already been undertaken leading to a seamless access to the boards definitions themselves. Henceforth, you'll just need to select the proper board from the Vivado's UI.

additional IP blocks

Zybo boards (and derivatives) from **Digilent** manufacturer makes use of specific IPs blocks. Those IPs have already been copied within the Xilinx tools repository, but you need to <u>register</u> them on a per user basis.

Launch the tools suite





Project → *Target Language* → *VHDL*

¹ IP stands for Intellectual **P**roperty. ARM processors in Zynq devices are **Hard IPs** while others will be **Soft IPs**.

Université de Toulouse / Faculté des Sciences et de l'Ingénieur / UE synthèse matérielle

. *	Settings	^	
Q- Tool Settings	Project Specify various settings related to project.	4	
Project IP Defaults	Default Project Directory		-
 Vivado Store Board Repository Example Project Reposito 	 <u>S</u>tart in directory (/home/devel) <u>L</u>ast project's directory 		
Source File Display	<u>H</u> ome directory (/home/devel) Desktop directory (/home/devel/Desktop)		
Help > Text Editor	Specify project directory:		
3rd Party Simulators > Colors Selection Rules	Target Language		

IP Defaults → *IP catalog* → *add vivado-libray path*

<u>k</u> *	Settings <@frontal.siame.univ-tlse3.fr> ^	×
Q- Tool Settings Project	IP Defaults Specify default IP example directory and IP repository search paths.	
IP Defaults > Vivado Store Source File Display	Default IP Example Project Directory Last project's directory Specify project directory	
WebTalk Help > Text Editor 3rd Party Simulators > Colors	IP Catalog Default IP Repository Search Paths	
Selection Rules Shortcuts > Strategies > Remote Hosts > Window Behavior	+ = 1 1 /nfs/xilinx/vivado-library	
?	Note: Default IP repository search paths will be applied only at new project creation. OK Cancel Apply Restore	

Practical exercises base files

In order to ease things a bit, you can start from files available at

- [M1] https://secil.univ-tlse3.fr/teaching/francois/UE-VHDL/M1/
- [M2] https://secil.univ-tlse3.fr/teaching/francois/UE-VHDL/M2/

TP1 - Pulse generator

This first exercise will enable you to undertake a **behavioural** simulation of a simple component: a **pulse** generator.



As a first step, retrieve the following files

https://secil.univ-tlse3.fr/teaching/francois/UE-VHDL/M1/pulse_gen.vhd https://secil.univ-tlse3.fr/teaching/francois/UE-VHDL/M1/test_pulse_gen.vhd

project setup

Launch the Vivado tools suite

vivado

You create a first project named 'zybo-tp1'





This is a RTL project but we'll add files later

À *	New Project	~ ^ X
Project Type Specify the type of project to create.		A
 <u>B</u>TL Project You will be able to add sources, create block and analysis. 	k designs in IP Integrator, generate IP, run RTL analysis, syn	thesis, implementation, design planning
Do not specify sources at this time Project is an extensible Vitis platform		

... click on boards and select Zybo-Z7-20

<u>}.</u> *			New Project					\sim	^ ×
Default Part Choose a default Xilinx part or board for you	r project.							I	4
Parts Boards To fetch the latest available boards fr	rom git repository, click (on 'Refre	sh' button. Dismi	55					
Vendor: All	Name: All				~	Board Rev: 1	.atest	~	•
Q ≍ ≑ •t	~								
Display Name	Preview S	Status	Vendor	File Version	Part	I/O Pin Count	Board Rev	Availa	
Zybo Z7-10	Ir	nstalled	digilentinc.com	1.1	xc7z010clg400-1	400	B.2	100	^
Zybo Z7-20	Ir	nstalled	digilentinc.com	1.1	xc7z020clg400-1	400	B.2	125	

The project is now initialised, the main UI will appear and we'll start to add files

À *				
<u>File E</u> dit F <u>l</u> ow <u>T</u> ools Rep <u>o</u> rts	<u>W</u> indow Layout <u>View H</u> elp <u>Q- Quick Access</u>			
🕞 🔶 🤌 🖩 🖿 🗙 🕨				
Flow Navigator 🛛 😤 🌻 🔔	PROJECT MANAGER - zybo-tp1			
✓ PROJECT MANAGER	Sources			
🔅 Settings	Sources			
Add Sources				
Longuago Templatas	🚍 Design Sources			
Language remplates	> 🚍 Constraints			
👎 IP Catalog	🗸 🚍 Simulation Sources			
	🖨 sim_1			
✓ IP INTEGRATOR	> 🚍 Utility Sources			

We'll start to add 'design sources' files

🍌 🖈	Add Sources	~ ^ X
	Add Sources This guides you through the process of adding and creating sources for your project	
	 Add or <u>c</u>reate constraints <u>A</u>dd or create design sources Add or create <u>s</u>imulation sources 	

select 'pulse_gen.vhd' with 'copy sources into project' option disabled

À *		Ac	d Sources		~ ^ X
Add or Crea Specify HDL, n to add to your	ate Desig etlist, Block project. Cre	n Sources Design, and IP fil eate a new source	es, or directorie e file on disk ar	es containing those file typ Id add it to your project.	es 🍌
$ +_{2} = $	↑ ↓				
	Index	Name	Library	Location	
•	1	pulse_gen.vhd	xil_defaultlib	/home/devel/teaching-vho	JI/VHDL
Scan and	Add	f Files Ad	d Directories Dject	<u>C</u> reate File	
☐ Copy sou Add sour	rces into pr ces from su	oject bdirectories			
?		< <u>B</u> a	ck <u>N</u> ex	t > <u>F</u> inish	Cancel

there are some errors ... quite normal since it's not finished

Now we'll add a simulation file in order to test our future pulse generator. Click on 'add file' in the project navigation bar and select 'simulation sources':

À *	Add Sources	~ ^ 😵			
ML Editions	Add Sources This guides you through the process of adding and creating sources for your project Add or create constraints Add or create design sources Add or create simulation sources				
elect 'test_pulse_gen.vhd' along with 'copy sources into project' option disabled					

🍌 🖈	Add	Sources		~ ^ X			
Add or Create Simulation Sources Specify simulation specific HDL files, or directories containing HDL files, to add to your project. Create a new source file on disk and add it to your project.							
Specify simulation set:	🕞 sim_1	~					
Index	Name	Library	Location				
• 1	test_pulse_gen.vhd	xil_defaultlib	/home/devel/teaching-v	hdl/VHDL			
<u>A</u> dd Files <u>Ad</u> d Directories <u>C</u> reate File Scan and add RTL include files into project Copy sources into project							
Add sources from su	bdirectories						
✓ Include all design s <u>c</u>	urces for simulation			Recurse a			
?	< <u>B</u> a	ck <u>N</u> ext	t > <u>F</u> inish	Cancel			

pulse_gen design

Our pulse generator will feature a **1MHz input clock**, an **asynchronous active low RST** line and will deliver a **1s output pulse**. To achieve this, you need to complete the 'pulse gen.vhd' file.

The test bench file 'test_pulse_gen.vhd' is a test architecture (i.e without I/O) that will define both a 1MHz internal clock, a timeout process along with another process that will feed signals on your pulse_gen component.

Note: in order to clarify the simulation results, you'll set a pulse output every **ten** MCLK's rising edge.

behavioural simulation

After having completed the 'pulse_gen.vhd' file, it's now time to achieve your first simulation. It it worth mentioning that there exists 3 levels of simulation:

- **behavioural** \rightarrow to test functionalities of your design,
- **post-synthesis** → inner components of your design are mapped on hardware resources, now you'll have propagation delays,
- **post-implementation** (i.e place and route) \rightarrow post-synthesis + routing delays like in real life.

Hence we'll start with the first level of simulation: **behavioural** On the vertical side panel, click 'Run Simulation' and select 'Run Behavioural Simulation'



Université de Toulouse / Faculté des Sciences et de l'Ingénieur / UE synthèse matérielle

Flow Navigator		PROJECT MANAGER - Z	ybo-tpl	
> PROJECT MAN	IAGER	Sources	? _ 🗆 🛙	×
	ι	Q ¥ ≑ +	2 0	٥
Create Bloc	ck Design	🗸 🚍 Design Sources ()	1)	
Open Block	Design	• pulse_gen	(behaviour) (pulse_gen.vhd)	
Generate E	Block Design	✓ ☐ Simulation Source	es(1)	
		∨ 🚍 sim_1 (1)		
✓ SIMULATION		∨ ●∴ test_pu	<pre>ilse_gen(behaviour) (test_pulse_gen.vhd) (1)</pre>	
Run Simula	tion	pgen0	: pulse_gen(behaviour) (pulse_gen.vhd) ~	
	Run Behavioral Simulati	on		
 RTL ANALY 	Run Post-Synthesis Fun	ctional Simulation	1	
> Open E	Run Post-Synthesis Tim	ing Simulation		
	Run Post-Implementatio	on Functional Simulation		
🕨 🕨 Run Syneme		an nining sinulation	S Compile Order	

You'll need to specify a 10s simulation time, restart simulation button then run for specified time button and finally press **zoom fit** button.

Mindow Layout ★ ∑ % SIMULATION - Be	View	Help Q. II ► Im mulation - Functiona	Quick Access 10 s v ar sim 1 test_pul	■ II C se_gen				
Untitled 1	\sim	2 3	1					_ D @ X
Q, 💾 🔍	Q 🔀	× + + + +	n≛ ≛r +F	Ter ⇒F −F E				\$
	4							<mark>14.000000</mark> 🗸
Name	Value	0.000000 us	2.000000 us	4.000000 us	6.000000 us	18.000000 us	10.000000 us	12.000000 us
🖁 E_CLK	0							
14 E_P	0							
16 E_RST	1							
ТІМЕОЦТ	25000000			2	50000000000 ps			
堤 cikpulse	500000 p				500000 ps			

Now ... Why did you get such an 'ASSERTION ERROR' in your test bench ??



>>> WELL DONE, you successfully ran your simulation test bench <<<

pulse_gen generic parameter(s)

You'll now add the **MAX_CPT** generic parameter to your pulse_gen component. It's default value will be set to **1E06**.

Update pulse_gen architecture.

Update test_pulse_gen.vhd to apply MAX_CPT=10 at the ppgen0 instantiation, then restart the behavioural simulation.

```
entity pulse_gen is
   generic (
        MAX_CPT : natural := 1E06
);
   port (
        RST, MCLK: in std_logic;
        P : out std_logic
   );
end pulse_gen;
```

design synthesis

Well now synthesise our design \Rightarrow Vivado will instantiate hardware elements like flip-flops, logic cells etc to implement your design.

In the 'Design Runs' tab (lowest part of UI), click the green arrow and click OK not modifying default synthesis parameters.

Tcl Console Messages		Log Report	ts	Design F	Runs	×	
Q ¥ ♦	$ \ll >$	> +	%				
Name	Constraints	Status	WNS	5 TNS	WHS	THS	
✓ ▷ synth_1	constrs_1	Not started					
⊳ impl_1	constrs_1	Not started					

🍌 🖈 🛛 Synthesis Completed	× After your design has	heen s	uccessf	ully synthes	ised select	the 'View
i Synthesis successfully completed.	Reports ' option.	+	+		+	+
Next	Site Type	Used	Fixed	Prohibited	Available	Util%
O <u>R</u> un Implementation	+ Slice LUTs*	+	++	0	+	0.16
O Open Synthesized Design	LUT as Logic	28	0	0	17600	0.16
• View Reports	LUT as Memory	0	0	0	6000	0.00
	Slice Registers	21	0	0	35200	0.06
Don't show this dialog again	Register as Flip Flop	21	0	0	35200	0.06
	Register as Latch	0	0	0	35200	0.00
OK Cancel	F7 Muxes	0	0	0	8800	0.00
	F8 Muxes	0	0	0	4400	0.00

post-synthesis simulation

Now	ha	ving	your	design	synthesised,
click	on	the	'Run	Simula	cion' on the
side		ра	anel	and	choose
post	-sy	ynth	lesis		Timing
Simu	ılat	cion	L		

~	SIMULATION		> • test_pulse
	Run Simulatio	n Run Behavioral Simul	ation
~	RTL ANALYSI:	Run Post-Synthesis F	unctional Simulation
	> Open Ela	Run Post-Synthesis T	iming Simulation
	SVNTHESIS	Run Post-Implementa	ation Functional Simulation
•		ran ost-implemente	

>>> What's happening ? What do you conclude ?? <<<

TP2 - Pulse generator synthesis

In the previous practical exercises <u>TP1 - Pulse generator</u>, we've not been able to simulate the synthesised design ... because synthesis applied **default generic values** and thus removed the generic parameter !

Hence, we need to manage two issues:

- having our pulse_gen component synthesised with the requested generic value,
- remove generic map from test_pulse_gen.vhd since our synthesised component does not exhibit generic parameters

Note: it is worth mentioning that this issue only occurs with top-level designs having generic parameters.

top-level synthesis and generic parameters

In order to give generic parameters values to top-level components, you'll need to create a TCL file at the root of your project:

• pulse_gen_synth.pre.tcl

```
# set generic parameter for synthesis
set_property generic {MAX_CPT=8} [current_fileset]
```

$\mathsf{Settings} \to \mathsf{Project} \ \mathsf{Settings} \to \mathsf{Synthesis} \to \mathsf{tcl.pre} \ \mathsf{field}$

		settings	~ X
oject Settings	Synthesis Specify various settings as	sociated to Synthesis	4
Beneral Bimulation Elaboration	Constraints		
Synthesis	Default <u>c</u> onstraint set:	😂 constrs_1 (active)	~
mplementation Bitstream	Report Settings		_
P	Strategy: 👍 Vivado 🤅	Synthesis Default Reports (Vivado Synthesis 2021)	~
ol Settings	Settings		
Project			^
P Defaults	🗌 Write Incremental Sy	nthesis	
/ivado Store Source File	Incremental synthesis:	Not set	
)isplay ∦ebTalk	S <u>t</u> rat eg y:	🔓 Vivado Synthesis Defaults* (Vivado Synthesis 2021) 🛛 🗸 💾	
Help	Description:	Viva do Synthesis Defaults	
Text Editor	~Synth Design (vivado)		
3rd Party Simulators	tcl.pre*	/nfs/home/francois/zybo-counter/zybo-co.	-
Colors	tcl.post		
Selection Rules	-flatten_hierarchy	rebuilt 🗸	
Strategies	-gated_clock_conver	sion off 🗸	
Remote Hosts	-bufg	12	~
Remote Hosts Window Behavior	-bufg Select an option above to	12 see a description of it	~
>		OK Cancel Apply Rest	

file will appear as an utility source file



New synthesis run will reports differently this time:

+	+ +	+			+
Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs* LUT as Logic	3	0 0	0	17600 17600	0.02
Slice Registers Register as Flip Flop	0 4 4	0 0	0	35200 35200	0.00
Register as Latch F7 Muxes F8 Muxes	0 0 0	0 0 0	0 0 0	35200 8800 4400	0.00 0.00 0.00
+	+ +	+			+

++	+	+
Ref Name	Used	Functional Category
+	++	+
FDCE	4	Flop & Latch
LUT3	3	LUT
LUT1	2	LUT
IBUF	2	IO
OBUF	1	IO
BUFG	1	Clock
+	++	+

FDCE: D flip-flop with Clock Enable and Asynchronous Clear FDPE: D flip-flop with Clock Enable and Asynchronous Preset FDSE: D flip-flop with Clock Enable and Synchronous Set FDRE: D flip-flop with Clock Enable and Synchronous Reset [ug901] Xilinx Vivado design suite user guide: synthesis

comment-out generic map

• test_pulse_gen.vhd

post-synthesis timing simulation

Restart the post-synthesis timing simulation, your output ought to change according to MAX CPT value.



>>> Do you notice something ?? <<<

Propagation delays now appear between E CLK rising edge and E P output update:

				2	, <mark>00</mark> 3.781 ns	
Name	Value	 2,000.000 ns 2,000.000 ns	2,002.000 ns		2,004.000 ns	2,006.0
LE_CLK	1					
14 E_P	1			Γ		
14 E_RST	1					
				з	<mark>.781 ns</mark>	
		 0.000 ns	2.000 ns		4.000 ns	6. <mark>000</mark> n

⇒ What's the MCLK's maximum frequency of your pulse_gen component on this chip ?

pulse gen with synchronous RST

This time, we'd like you to modify your pulse gen architecture to implement a synchronous reset.

+	++ Used +	Functional Category
FDRE		Flop & Latch
LUI4 IBUF	2	IO
OBUF LUT3	1 1	IO LUT
LUT2		LUT
BUFG		Clock
+	+ +	+

FDCE: D flip-flop with Clock Enable and Asynchronous Clear FDPE: D flip-flop with Clock Enable and Asynchronous Preset FDSE: D flip-flop with Clock Enable and Synchronous Set FDRE: D flip-flop with Clock Enable and Synchronous Reset see [ug901] Xilinx Vivado design suite user guide: synthesis

What do you notice on chronogrammes? regarding the instantiated resources ??

TP3 - Pulse generator constraints

In the previous exercises, we simulated our pulse_gen component without any timing constraints to validate against ... hence the unconstrained paths reported by the Vivado's Report Timing summary (left toolbar Synthesis \rightarrow Report Timing Summary) see <u>pluse_gen unconstrained paths</u> on next page

Timing constraints

A *constraint file* enables you to express timing requirements against a clock frequency your design is supposed to perform with. In subsequent practical exercises, you'll discover that this constraint file is also used for I/O Planning (i.e which IO ports your design makes use of). These files exhibit a xdc extension.

You'll now create a constraint file. To achieve this, click on '⁺' in Sources area and select add or Create constraints → create a file named 'pulse gen.xdc'



Your design now features a new constraint file named 'pulse gen.xdc'.

Instead of trying to fill it on yourself, we'll make use of the Synthesis \rightarrow Constraints Wizard



pluse_gen unconstrained paths



Synthesis Constraints Wizard

The first time you click on this feature, it will ask you to select a target (i.e which XDC file) \Rightarrow pulse_gen.xdc The second time, the wizard will start :)

In this application case, we'll only focus on the MCLK constraints. Set MCLK as being a 125MHz clock then *skip to finish*.

<u>}</u> ×			Timing	Constraints Wi	zard		~	^
Primary Primary name a	r y Clocks clocks usu: nd waveforr	ally enter m (rising	the design though and falling edge tim	input ports. S nes) to describ	pecify the peri the duty cyc	iod and optior le if not 50%.	nally a More	4
Q		· 파,						1
	Object	Name	Frequency (MHz)	Period (ns)	Rise At (ns)	Fall At (ns)	Jitter (ns)	
	🔟 MCLK	MCLK	125.000	8.000	0.000	4.000		

This will lead to a constraint rule added to your pulse gen.xdc :

```
# Master Clock timing constraint
create_clock -period 8.000 -name MCLK -waveform {0.000 4.000} [get_ports MCLK]
```

Note: this rule DOES NOT DEFINE a clock, instead, it specifies a clk our system is supposed to comply with.

Now you restart Synthesis \rightarrow Report Timing summary



>>> Slack OUGHT to be positive <<<

Setup slack: Required time - Arrival time \rightarrow delay before next CLK front that will memorize the data Hold slack: Arrival time - Required time \rightarrow delay the data change after the CLK front.

Introduction to slack and others timing considerations in FPGAs <u>http://www.ece.utep.edu/courses/web5375/Notes_files/ee5375_timing_fpga.pdf</u>

Implementation Constraints Wizard

Why did we choose 125MHz for MCLK in the previous example ?

That's because the Zybo board features an independent PL_CLK at 125MHz coming from the onboard Ethernet IC.



As shown in the upper figure, you can see that the **K17** pin is used as the PL_CLK input. Additionally, **E17** pin will be able to reset the Ethernet IC hence suspending the 125MHz clock.

Up To now, we've been through the synthesis process that translates your VHDL code to instantiated physical elements of our FPGA. Next step is to map IO pads to our design \Rightarrow IO planning

Implementation \rightarrow Run Implementation , then select layout \rightarrow IO planning in the Vivado upper tool bar.

Layo	but	⊻iew	<u>H</u> elp	Q- Quick Access				
	Det	fault Lay	out					
	I/O Planning							
	Floorplanning							
=	Tim	ning Anal	ysis					

Below is a detailed overview of buttons and leds available on the Zybo board (switches not shown).



Now you'll affect Zybo GPIO to our pulse_gen ports according to the following setup:

Tel Concola	Massages Log	Daparta Dasi	an Dune Tir	ning	Dower b	lathodologu	DBC	Dackar		I/O Borto	
TCI CUNSUle	Messages Lug	Reports Desi	gri kuns m	iiirig	Power M	letunnnndå	DRC	Раскад	e pins	10 Purts	×
Q ₹ ♦	R + H										
Name	Direction	Board Part Pin	Board Part Ini	terface	Neg Diff Pa	air 🛛 Package P	in	Fixed	Bank	I/O Std	
🗸 🕞 All ports (3	3)										
🗸 🕞 Scalar p	oorts (3)										
MCLK	IN					K17	~	 Image: A start of the start of	35	LVCM0S33*	
P 🖉	OUT					M14	~	\checkmark	35	LVCM0S33*	•
🕑 RST	IN					R18	~		34	LVCMOS33*	Ŧ

When finished, save as **CTRL** + **S** \rightarrow Vivado will ask you whether it ought to save these constraints in the default constraints file (i.e pulse gen.xdc) \rightarrow yes !

In the end, your pulse gen.xdc file ought to looks like this:

```
# Master Clock timing constraint
create_clock -period 8.000 -name MCLK -waveform {0.000 4.000} [get_ports MCLK]
set_property PACKAGE_PIN K17 [get_ports MCLK]
set_property IOSTANDARD LVCMOS33 [get_ports MCLK]
set_property PACKAGE_PIN M14 [get_ports P]
set_property IOSTANDARD LVCMOS33 [get_ports RST]
set_property PACKAGE_PIN R18 [get_ports RST]
```

... a more elegant way is to make use of the 'dict' feature: pulse gen.xdc

```
# Master Clock timing constraint
set_property -dict { PACKAGE_PIN K17 IOSTANDARD LVCMOS33 } [get_ports MCLK];
create_clock -period 8.000 -name MCLK -waveform {0.000 4.000} [get_ports MCLK];
# Reset (warning pull down resistors on board)
set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCMOS33 } [get_ports RST];
# P output
set_property -dict { PACKAGE_PIN M14 IOSTANDARD LVCMOS33 } [get_ports P];
```

It's now time to restart Synthesis + Implementation :)

Full timings simulation

i un uningo sinialation					2,008	3.363 n
This time, you'll select: <i>Simulation</i>	Name	Value	 1,999.989 ns 2,000.000 ns	2,005.000	ns	2,010.0
\rightarrow Run Simulation	l⊌ e_clk	1				
\rightarrow Post-Implementation timing	16 E_P	1				
simulation	谒 E_RST	1				
What do you observe ??			1		8.37	4 ns
			 0.000 ns	5.000 ns		10.000

[master2] Bitstream generation and hardware download

Ultimate step that will generate the .bit file. This file will get downloaded into the FPGA's config RAM that will organise all resources as specified in synthesis and implementation steps. This file will get downloaded through JTAG (usb prog port) (*hint: change the pulse duration to be able to see it!*)

TP4 - Synthesizable Log2 function

This time, we'd like you to write a **synthesizable** Log2 component that will feature the following generic parameter(s):

• BUS_WIDTH : size of input and output buses with 8 as default value



log2 @ cpu_package

Once you retrieved the 'cpu_package.0.vhd' file, you'll find definition for several log2 (overloaded) functions:

fonction log2
calcule le logarithme base2 d'un entier naturel, ou plus exactement
renvoie le nombre de bits nécessaires pour coder une valeur
function log2 (I: in natural) return natural;
<pre>function log2 (vI: in std_logic_vector) return std_logic_vector;</pre>

Your first task will be to implement a synthesizable version of the log2 function in the package body part of the cpu package file.

Run a <u>behavioral simulation</u> by means of the 'test log2.vhd' file.

log2 hardware component

As you noticed, your log2 function exhibits **unconstrained** std_logic_vector both as input and output. Synthesis can manage this situation by dynamically selecting the proper bus size through component instantiation.

 \Rightarrow complete the 'log2_hw.vhd' file with the generic parameter **BUS_WIDTH**; your architecture will make use of your previously written log2 function.

Undertake a **post-synthesis timing** simulation.

Next step: have a closer look at the schematic of the resulting synthesis: what do you notice on output ??



[master2] optimized log2 component

In order to overcome the issue related to the wrong size of the log2 component's output, you'll need to:

- reshape output and subsequents to std_logic_vector(log2(BUS_WIDTH)-1 downto 0);
- import your log2 function from cpu_package as a new myLog2 function to settle in the declarative part of your log2 component's architecture.

Undertake a **post-synthesis timing** simulation and have a look at the generated schematic that ought to looks like this later:



TP5 - Registers bank for Risc processor

We'd like you to create a bank of 8 registers 32 bits featuring dual read access. One notable point is that reading register 0 always sends back value 0.



Signals featuring **'*'** means that they are active low. ADR_A and ADR_B addresses buses lead to asynchronous reading on ports QA and QB respectively.

RST* is synchronous and leads to all registers filled with 0. Write operations through ADR_W and active W* signal will be undertaken on the rising edge of CLK.

generic parameters

- DBUS_WIDTH internal words and data buses size, default to 32
 - **ABUS_WIDTH** size of all addresses buses, default to **5**

Notes

To declare a 32bits wide 8 registers bank in the declarative part of your architecture: type FILE_REGS is array (integer range 0 to (2**5)-1) of std_logic_vector (31 downto 0); signal REGS : FILE_REGS;

Behavioural simulation

You'll complete the design of our registers bank from the 'registres.0.vhd' file and then, add the simulation source 'test_registres.0.vhd'.

Note	e: to g	ain	ac	cess	s to
ALL	signa	als	in	obje	ects
and	below,	yo	u'll	nee	d to
tell	the	sin	nula	ator	to
unde	ertake	а	re	ecurs	sive
inclu	ision				

✓ ⇒ ■ test_registr	es test_regist VHDL Entity	> 🗑 ADR_A[2:0]			
🔳 regf0	1dd to Wovo Window	👕 🗑 ADR_B[2:0]			
	Log to Wave Database	Objects in Scope			
	Go to Source Code	Objects in Scope and below			
	Go to Instantiation Source Code	W QB[31:0]			
	Set Current Scope To Active	W REGS[0:7][31:0]			

Around 80ns in the waveform window, you notice an **issue** that occurs whenever the processor is trying to **read and write** the same register at the same time !

To overcome this feature bug, we want you to implement the **bypass mechanism**.

Registers Bypass design and simulation



We need to detect the condition which lead to the previously mentioned issue:

if write and ADR_A or ADR_B equal ADR_W then QA or QB <= D;

This time, you'll need to **remove** the 'registres.0.vhd' file with the new one 'registres.1.vhd'. In this file, we want you to rewrite in the concurrent area of your architecture the two existing processes named P_READQA and P_READQB.

Check the operationality of your **bypass** by means of a **behavioural simulation**.

Synthesis with constraints

This step will try to tackle the following objectives:

- determining the maximum frequency a processor can access our bank of registers,
- use BlockRAMs instead of traditional FF

Note: remember to set generic parameters for synthesis as what has been done in <u>top-level synthesis and</u> <u>generic parameters</u>

Note: in a similar way to what was done in <u>Synthesis Constraints Wizard</u>, define the same 125MHz clock.

Ref Name	Used	Functional Category	Site Type	Used	Fixed	Prohibited	 Available +	Util% ++
+ FDRE LUT5 LUT6 OBUF MUXF7 IBUF LUT4 LUT3 LUT1 BUFG	224 128 66 64 44 7 2 1 1	Flop & Latch LUT IO MuxFx IO LUT LUT LUT Clock	Block RAM Tile RAMB36/FIFO* RAMB18 +	0 0		0 0	60 60 120	0.00 0.00 0.00

We're not using any **BRAM** primitives :(

To found out the maximum affordable frequency of our design, we need to discover the longest path in our design \Rightarrow *Synthesis* \rightarrow *Report Timing summary*

Tcl Console Messages Log	I	Reports D	esign Runs	Timing	x Debug		
Q X ≑ C Ш ●		Q - 2	1 🗞 II	O U	nconstrained	l Paths - NOM	NE - NONE - S
User Ignored Paths	\sim	Name	From	То	Total Delay	Logic Delay	Net Delay
✓		👍 Path 41	ADR_W[1]	QA[0]	6.448	3.876	2.572
🗸 🚍 CLK to NONE		👍 Path 42	ADR_W[1]	QA[10]	6.448	3.876	2.572
Setup (10)		👍 Path 43	ADR_W[1]	QA[11]	6.448	3.876	2.572
Hold (10)		👍 Path 44	ADR_W[1]	QA[12]	6.448	3.876	2.572
> 🚞 NONE to CLK	l	👍 Path 45	ADR_W[1]	QA[13]	6.448	3.876	2.572
🗸 🚍 NONE to NONE		👍 Path 46	ADR_W[1]	QA[14]	6.448	3.876	2.572
Setup (10)		👍 Path 47	ADR_W[1]	QA[15]	6.448	3.876	2.572
Hold (10)	~	👍 Path 48	ADR_W[1]	QA[16]	6.448	3.876	2.572
Timing Summary - timing_1							

Then, as you may already have guessed ... the longest path is when you're undertaking both read and write operations on the same register involving all of the bypass logic

 \Rightarrow 150MHz will be our most achievable operational frequency.

[master2] BRAM inference and maximum reachable frequency

For comparison purposes, we want you to design our bank of registers by means of Block RAM primitives. You'll drive synthesis and post-synthesis timing simulation to find out the max sustainable frequency.

Below is **ug473** excerpt showing **WRITE_FIRST** (i.e transparent mode) policy. Note that pipeline register output s not used



To achieve this, best is to start with TP6 - Memory bank for RISC processor

TP6 - Memory bank for RISC processor

We propose you to carry out our future data & instructions L1 caches as a generic memory bank featuring the following parameters:

- DBUS_WIDTH internal words and data buses size, default to 32
 - nb words within our memory bank, default to 8
- FILENAME

MEM SIZE

file whose content will be our memory bank defaults values, default to *empty string*.



Functional principle

RST^{*} will be **synchronous**. It either enables loading a file's content whose name has been given as the generic parameter **FILENAME**, or all memory elements will get assigned value **0**.

EN signal validates memory access whose address is set on the ADR bus while WEN signal will tell this operation is a write (i.e read otherwise). On the next rising edge of CLK, data will be set on the DO bus (read) or written within the corresponding word.



Simulation

Undertake a behavioural simulation of your memory component by means of the 'memory.3.vhd' and 'test_memory.3.vhd' test bench. You'll also need your 'cpu_package.0.vhd' as source file (log2 function) along with the 'rom_file.0.vhd' that will get used to pre-fill your memory on reset. Note: 'rom file.0.txt' ought to be added as source file (it will be tagged as TEXT file).

Dr THIEBOLT François - IRIT

WARNING: in 'test_memory.3.vhd', you ought to specify the absolute filename to the 'rom file.0.txt' file:

```
constant FILENAME : string := "/home/<user>/tpvhdl/rom_file.0.txt";
```

Once functional behaviour is achieved, start **design synthesis** and then carry out a **post-synthesis timing simulation** to determine the maximum sustainable frequency.

Hint: don't forget to add the following pre-synthesis constraints like what was done in <u>top-level synthesis and</u> <u>generic parameters</u>

• memory_synth.pre.tcl

```
# set generic parameter for synthesis
set_property generic {DBUS_WIDTH=32 MEM_SIZE=32 FILENAME="rom_file.0.txt"} [current_fileset]
```

Note: below some size threshold, Vivado may decide to infer RAM as distributed RAM (i.e LUT)

>>> How many BRAM did you infer ??? <<<

Block RAM inference

As you may have noticed, you'll need your previously written log2 function to express the size of the ADR bus. Starting from the memory.3.vhd file, you infer Block RAMs from the Xilinx architecture.

Hint: have a look at ug901 'RAM HDL Coding Techniques' to know how to infer BRAM at synthesis without any architecture specific component instantiation.

 \Rightarrow you discover that you need to remove the RST line and disable the 'Z' feature when EN='0' Restart design synthesis and observe the output results:

	Site Type		Used	 Fixed	Prohil	bited	Avai:	lable	Util	.8
Slid	ce LUTs*	+	0	0	+	0	:	17600	0.0	0
LU	JT as Logic		0	0		0	:	17600	0.0	0
LU	JT as Memory		0	0		0		6000	0.0	0
Slic	ce Registers		0	0		0	:	35200	0.0	0
Re	egister as Flip	Flop	0	0		0	:	35200	0.0	0
Re	egister as Latcl	h	0	0		0	:	35200	0.0	0
F7 N	luxes		0	0		0		8800	0.0	0
F8 N	luxes		0	0		0		4400	0.0	0
+		+		+	+		+	4		+
+	Site Type	+ Used	Fixed	+ d Proh	ibited	+ Avai:	lable	+ Util%	- + 5	
Bloc	ck RAM Tile	0.5	(+ D	0		60	0.83	- + -	
R <i>I</i>	AMB36/FIFO*	0	() (0		60	0.00)	
R <i>I</i>	AMB18	1	() (0		120	0.83	5	
	RAMB18E1 only	1								
+		+	+	+		+		+	-+	

Tcl Console	Messages	Log Reports Design		ın Run	s :	×			
Q ≭ ♦	I∢ ≪ ►	\gg	+ %						
Name	Constraints	Status	5		LUT	FF	BRAM	URAM	DSP
✓ ✓ synth_1	constrs_1	synth_design Complet		plete!	0	0	0.5	0	0



Ok, good you really got so far :)

Unfortunately, whatever simulator you may use, VHDL timing simulation does not exist for Xilinx primitives (only verilog and system verilog) $\dots \Rightarrow$ it's time to test in real life :)

[master2] bitstream & JTAG debug

Same steps as [master2] Bitstream generation and hardware download then open the Hardware Debugger

to be continued ...

TP7 - dual ports FIFO

On se propose de réaliser une FIFO* synchrone double port selon les caractéristiques génériques suivantes :

• **DBUS_WIDTH** Taille d'un mot du banc avec 32 bits par défaut.

• ABUS_WIDTH Profondeur d'utilisation avec 3 bits d'adresse par défaut - soit 2**3 mots -.

*First In First Out



Principe de fonctionnement

Le signal RST*, synchrone, remet à **0** les pointeurs lecture et écriture mais n'efface pas le contenu des registres. Il positionne également la sortie DO <= 'Z'.

Tant que WEN* est actif, la FIFO continue de se remplir de façon circulaire à chaque **front montant** d'horloge écrasant ainsi les données les plus anciennes.

Tant que REN^{*} est actif, la FIFO continue de présenter une donnée sur le bus à chaque **front montant** d'horloge jusqu'à être vide, auquel cas elle présentera la valeur **0**.

Le bus de sortie Q devant être raccordé à d'autres unités, la sortie passe en haute impédance quand REN* inactif. Les signaux EMPTY, MID et FULL représentent respectivement l'état vide, <u>au moins</u> à moitié plein* et plein. Ces derniers seront mis à jour sur front montant d'horloge.

*Le signal MID='1' quand remplissage FIFO >= 50%.

Les signaux WEN* et REN* peuvent être actifs simultanément.



Chronogramme



Implémentation

Vous compléterez le fichier fifo.0.vhd comprenant entité et architecture.

La lecture / écriture étant circulaire, vous ferez attention aux conditions limites pour l'établissement des indicateurs de remplissage. Ensuite, afin d'accélérer les transferts, les pointeurs devront déjà être positionnés sur le futur emplacement de lecture ou d'écriture, l'incrément se faisant après l'opération proprement dite. Enfin, le pointeur de lecture référence **toujours** la donnée la plus ancienne.

Cas particuliers

Lecture & Ecriture & FIFO vide \Rightarrow Q=0 & mémorisation de la donnée en entrée.

FIFO pleine & Écriture \Rightarrow mémorisation de la donnée et incrément du pointeur de lecture.

FIFO pleine & Lecture & Ecriture \Rightarrow Q=donnée la plus ancienne et mémorisation de la nouvelle.

Notes

• Le type **buffer** associé à un signal permet de relire un port configuré en sortie.

Behavioural simulation

Vous compléterez le fichier test_fifo.0.vhd avec une instance du composant fifo possédant 4 mots mémoire de 4 bits.

Synthesis

You will, as usual, undertake a synthesis of your design.

However, having already been using **BRAM** primitives, we'd like you to implement our FIFO based on these primitives. Indeed, according to <u>Xilinx ug473 documentation</u>, BRAMs embed all logic bases to implement a FIFO. You are welcomed to tamper on the initial features of our FIFO, the main goal being to reduce the overall usage of LUT and FF vs BRAM elements.

>>> the less LUTs and FFs you use, the better will be your design (and rating) !! <<<

TP8 - BLDC controller

BLDC stands for "brushless direct current" and is related to a type of motor making use of permanent magnets on their rotor. These high reliability motors are intended for heavy duty purposes and can be found in drones, servers ... and hard drives !



Unlike DC motors, a BLDC motor requires 3 phases sequenced in a precise timing.

As shown above, your "*BLDC controller*" circuit will be in charge of driving the six transistors of the "*driver*" part of the system. It is worth mentioning that "*electrical speed*" may differ from "*mechanical speed*" (e.g startup phase with mechanical load), hence we'll make use of a feedback signal. This signal will either come from a hall effect sensor located within the stator ... or (and probably easier to use) a simple optical signal that detects the position of the disk in the hard drive BLDC use case (see photos below).



Mechanical Two Pole Pair Rotation

Excerpt from Elektor magazine: "BLDC Guide du débuttant"



Excerpt from Elektor magazine: "BLDC Guide du débuttant"

Speed control

You'll make use of PWM control to adjust the speed of the brushless motor. Additionally, there will be a ramp UP and ramp DOWN effect to avoid high pulses of energy, exemple:

• motor is at stop and the speed control is set at its maximum ... then you'll smoothly increase the speed up to the specified value.

BLDC controller

By means of a FPGA, we'd like you to implement the logical part of the controller as a component. Unfortunately, this kind of controller requires a LOT of parameters ... but to ensure your success as a first step undertaking such a device, we'll restrain ourselves to the following generic parameters:

• MAX_CPT \rightarrow defines a whole phase cycle expressed as number of ticks of the main clock

As an example, let's say our main clock has a 1MHz frequency. On the other end, our motor requires a 50Hz phase cycle \Rightarrow MAX_CPT will get set at (1MHz/50) at instantiation (or synthesis) time.

to be continued

Use case

In order to avoid destroying one of our electrical vehicles (!), we'll make use of specifically prepared hard drives for such an experiment.



to be continued

Links

[Elektor] BLDC newbies guide https://www.elektormagazine.fr/articles/contr%C3%B4le-des-moteurs-bldc-guide-du-d%C3%A9butant

Master 2



Links

VHDL base files <u>https://secil.univ-tlse3.fr/teaching/francois/UE-VHDL/M2/</u>

Zybo-Z7 reference manual

https://secil.univ-tlse3.fr/teaching/francois/UE-M2-VHDL/ZYBO-Z7-reference-manual-B.pdf

Zybo-Z7 schematic <u>https://secil.univ-tlse3.fr/teaching/francois/UE-M2-VHDL/ZYBO-Z7_schematic-D1.pdf</u>

[M2] PWM

As a first step, we'll implement a PWM module that will get connected to a 125MHz clock generator located on our Zybo-Z7. The outcome will be a 1s variable pulse width driving a led.

To help you starting, you'll need the following files:

https://secil.univ-tlse3.fr/teaching/francois/UE-VHDL/M2/pwm.vhd https://secil.univ-tlse3.fr/teaching/francois/UE-VHDL/M2/test_pwm.vhd



The duty=0 corner case: since our PWM features an EN pin, we'll consider duty=0 as an active slot.

Behavioural simulation

You'll need to follow the TP1 - Pulse generator as a guideline:

- select Zybo-Z7 20 board
- add pwm.vhd component file + test_pwm.vhd simulation file
- complete these files
- launch behavioural simulation

Note: to launch the simulation, don't forget to re-enable generic map along with the behaviour architecture.

Synthesis and Implementation

Now you'll follow the TP2 - Pulse generator synthesis as a guideline:

Add the 'pwm_synth.pre.tcl' script to the synthesis of your design:

• pwm_synth.pre.tcl

```
# set generic parameter for synthesis
set_property generic {SYS_CLK=125000000 PWM_FREQ=1 DUTY_RES=4} [current_fileset]
# set generic parameters intended to post-synthesis simulation pwm_clk = sys_clk / 20)
#set property generic {SYS_CLK=125000000 PWM_FREQ=6250000 DUTY_RES=4} [current_fileset]
```

Synthesis output will exhibits:

```
Starting RTL Elaboration : Time (s): cpu = 00:00:03 ; elapsed = 00:00:04 . Memory (MB): peak =
2810.660 ; gain = 0.000 ; free physical = 16041 ; free virtual = 37790
INFO: [Synth 8-638] synthesizing module 'pwm' [/home/devel/teaching-vhdl/VHDL/pwm.vhd:40]
Parameter sys_clk bound to: 125000000 - type: integer
Parameter pwm_freq bound to: 1 - type: integer
Parameter duty_res bound to: 4 - type: integer
INFO: [Synth 8-256] done synthesizing module 'pwm' (0#1)
[/home/devel/teaching-vhdl/VHDL/pwm.vhd:40]
```

Note: SYS CLK=125E06 will be considered a string ... without issue in the upcoming steps 🙂

Post-synthesis simulation

Please pay attention to the fact that being a top-level design, our 'pwm' module won't be able to make use of VHDL generic parameters.

Hence, we need to remove the *generic map* part of the 'test_pwm.vhd' file. Please pay attention to the 'structure' architecture now in use.

• test_pwm.vhd

```
.....
-- instantiation et mapping du composant registres
--pwm0 : entity work.pwm(behaviour) --behavioural simulation
            generic map (sys clk => sys freq,
___
_ _
                               pwm_freq => pwm_freq,
___
                               duty_res => duty res)
pwm0 : entity work.pwm(structure) --post-synthesis functional simulation
            port map (CLK => E CLK,
                         RST => E RST,
                         EN => E EN,
                         DUTY => E DUTY,
                         P => E P);
.....
```

In addition, you may consider using **alternate generic values** for post-synthesis simulation, see pwm_synth.pre.tcl.

IO planning (implementation)

It's now time to start the Implementation (i.e IO planning):

• connect the onboard 125MHz clock to our CLK input



- connect the 4 slides switches SW[3..0] as the DUTY input bus,
- connect the BTN0 as the EN input (active low ---look at the pull-down resistors!)
- connect the 4 leds as the P output
- RST signal will get tied to '1' RST signal will get tied to BTN1 (pull-down resistor!)



See TP3 - Pulse generator constraints as a guideline:

Launch Synthesis \rightarrow Open Synthesized design \rightarrow Constraint Wizard to create the pwm.xdc constraint file:

```
• pwm.xdc
```

```
# Master Clock timing constraint
create_clock -period 8.000 -name CLK -waveform {0.000 4.000} [get_ports CLK]
set_property PACKAGE_PIN K17 [get_ports CLK]
set_property IOSTANDARD LVCMOS33 [get_ports CLK]
set_property IOSTANDARD LVCMOS33 [get_ports EN]
set_property PACKAGE_PIN P16 [get_ports RST]
set_property IOSTANDARD LVCMOS33 [get_ports RST]
set_property PACKAGE_PIN G15 [get_ports {DUTY[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DUTY[0]}]
```

set_property	PACKAGE_PIN P15 [get_ports {DUTY[1]}]
set_property	<pre>IOSTANDARD LVCMOS33 [get_ports {DUTY[1]}]</pre>
set_property	PACKAGE_PIN W13 [get_ports {DUTY[2]}]
<pre>set_property</pre>	<pre>IOSTANDARD LVCMOS33 [get_ports {DUTY[2]}]</pre>
set_property	PACKAGE_PIN T16 [get_ports {DUTY[3]}]
<pre>set_property</pre>	<pre>IOSTANDARD LVCMOS33 [get_ports {DUTY[3]}]</pre>
<pre>set_property</pre>	PACKAGE_PIN D18 [get_ports P]
<pre>set_property</pre>	IOSTANDARD LVCMOS33 [get_ports P]

Note: best is to first create this XDC file through the Wizard ... then to modify it on your own

TODO: set RST as '1'

TODO: add debug core to our design

Bitstream generation

Select 'impl_1' (i.e implementation), click the green RUN icon, then continue with Bitstream generation

Tcl Console	Messages L	og Reports	Design Runs	×					
Q ¥ ♦ I ≪ ► >> + %									
Name	Constraints	Status		LUT	FF	BRAM	URAM	DSP	
∨ 🗸 synth_1	constrs_1	synth_design C	synth_design Complete!		54	0	0	0	
✓ impl_1	constrs_1	write_bitstrean	n Complete!	48	54	0	0	0	

Hardware Manager

This is the main application responsible for establishing a dialog with the hardware ... that will enable you to upload or readback a bitstream.

 $\textit{Open Hardware Manager} \rightarrow \textit{Open Target} \rightarrow \textit{auto connect}$

✓ PROGRAM AND DEBUG	Hardware ?	_ 0 Ľ ×
Generate Bitstream	$Q_{i} \mid \underbrace{\clubsuit}_{i} \mid \underbrace{\r}_{i} \mid $	٥
 Open Hardware Manager 	Name	Status
Open Target	V I localhost (1)	Connected
Program Device	✓ ■	Open
Add Configuration Memory D	arm_dap_0 (0)	N/A
Add Configuration Memory D	✓ ⊕ xc7z020_1 (1)	Programmed
	T XADC (System Monitor)	

[optional] Bitstream readback

In order to backup a readable bitstream, use the following command in the TCL console:

readback_hw_device [current_hw_device] -bin_file zyboZ7-20_original.bin

https://docs.xilinx.com/r/en-US/ug908-vivado-programming-debugging/Bitstream-Verify-and-Readback-for-FP GAs-and-MPSoCs

Download bitstream

Now you just need to click on '*Program device*' and it will select the latest generated bit stream that will get downloaded to the config RAM of your FPGA.



Tests 🙂



[M2] my AXI-enabled PWM IP²

We'll now turn our PWM to an AXI-enabled PWM IP. This leverages the needs for large systems integration involving the PS part (i.e on board ARM9 processors).

To reach this goal, we'll start creating a new VHDL project without specifying any HDL source code.

Then, we'll start the creation of an AXI component that exhibits the following features:



Select

— Create a new AXI4 peripheral

À 🗶	Create and Package New IP	~ ^ ×
Create Please se	Peripheral, Package IP or Package a Block Design elect one of the following tasks.	4
Pac	kaging Options	
	 Package your current project Use the project as the source for creating a new IP Definition. 	
	 Package a block design from the current project Choose a block design as the source for creating a new IP Definition. 	
	 Package a specified directory Choose a directory as the source for creating a new IP Definition. 	
Cre	ate AXI4 Peripheral	
	Create a new AXI4 peripheral Create an AXI4 IP, driver, software test application, IP Integrator AXI4 VIP simulation and debug demonstration design.	
?	< <u>B</u> ack <u>N</u> ext > <u>F</u> inish	Cancel

² To help yourself creating an IP, search over the net for "Xilinx IP integrator" then select "Vivado Design Suite Tutorial: Creating and Packaging Custom IP"

À 🗶 🔰	Create and Packa	ge New IP	ĺ~	^ X				
Peripheral Detai	ils							
Specify name, version peripheral	on and description	for the new	I					
Name: Version: Display name: Description: IP location: Overwrite e	myAXIpwm 1.0 myAXIpwm_v1.0 AXIpwm /home/devel/ip_re	po						
< <u>B</u> ack ▲ Add Interfaces Add AXI4 interfaces su	<u>N</u> ext >	<u>Finish</u> Create	Cance e and Pack	age New	IP			< ^ >
Enable Interrupt	Support	H ■ Interfaces ① SO0_AXI		Name Interface Data Wie Memory Number	e Type e Mode dth (Bits) / Size (Bytes)	S00_AXI Lite Slave 32 64 4		 ✓ ✓
•			[< <u>B</u> ack	Ne	ext >	<u>F</u> inish	Cancel

Université de Toulouse / Faculté des Sciences et de l'Ingénieur / UE synthèse matérielle

<u>}</u> *	Create and Package New IP	~ ^ X
	Create Peripheral	
MLEditions	 Peripheral Generation Summary 1. IP (user.org:user:myAXIpwm:1.0) with 1 interface(s) 2. Driver(v1_00_a) and testapp more info 3. AXI4 VIP Simulation demonstration design more info 4. AXI4 Debug Hardware Simulation demonstration design more info 	Î
	 /home/devel/ip_repo Next Steps: Add IP to the repository Edit IP 	
E XILINX.	Click Finish to continue	v
?	< <u>B</u> ack <u>N</u> ext > <u>F</u> inish	Cancel

Now you can going on with IP customization

This will dynamically create a new project <home>/ip_repo/edit_myAXI_pwm_v1_0.xpr

Links

Digilent Zybo's AXI IP + Vitis baremetal driver <u>https://digilent.com/reference/programmable-logic/guides/getting-started-with-ipi</u>

AXI IP creation

https://indico.ictp.it/event/a14283/session/62/contribution/252/material/slides/2.pdf

[Xilinx] Vivado Design Suite Tutorial: Creating and Packaging Custom IP (UG1119)

HOW TO CREATE an AXI4-FULL CUSTOM IP with AXI4-LITE and UART INTERFACES in VIVADO <u>https://www.mehmetburakaykenar.com/how-to-create-an-axi4-full-custom-ip-with-axi4-lite-and-uart-interfaces-i</u>n-vivado/192/

Fixing Xilinx's Broken AXI-lite Design in VHDL https://zipcpu.com/blog/2021/05/22/vhdlaxil.html [sample]https://github.com/ZipCPU/wb2axip/blob/master/bench/formal/xInxdemo 2020 2.vhd

Create Custom AXI Cores Part 1: Straight to the Finish Line https://www.hackster.io/cospan/create-custom-axi-cores-part-1-straight-to-the-finish-line-a70e5e

Designing a Custom AXI IP on Vitis

https://www.hackster.io/pablotrujillojuan/designing-a-custom-axi-ip-on-vitis-a0ad06

Editing an IP from main project

ì

Editing an IP means it will **launch a new project** embedded within your current project. To start editing your IP from your main project, click on *IP catalog* then select your IP + right click then select *Edit in IP Packager*

Flow Navigator 🗧 🚔 ? 🔔	PROJECT MANAGER - zyboZ7-AXI-pwn	n
V PROJECT MANAGER	Project Summary × IP Catalog	$_{3}$ \times IP Catalog (2) \times IP Catalog
Settings	Cores Interfaces	
Add Sources	Q	
Language Templates	Search: Q-	
+ Ir Catalog	Name	AXI4
✓ IP INTEGRATOR	User Repository (/home/devel/	/ip_repo/myAXIpwm_1.0)
Create Block Design	Attrenpretation with the second se	
Open Block Design	V 🖻 User Repository (/nfs/xilin	Properties Ctrl+E
Generate Block Design	> 🚍 AXI Peripheral	IP <u>S</u> ettings
	Embedded Processing	Add Repository
✓ SIMULATION	> AXI Peripheral	
Run Simulation	> UserIP	Edit in IP Packager
	> 📄 Video & Image Process	Disable_IP
✓ RTL ANALYSIS	> 📄 VIVADO HLS IP	Delete IP Delete

IP customization

Roughly speaking, IP customization is a packaging of our previously designed PWM component.



In the main window of the newly created project, you can see two files:

- the IP component myAXIpwm_v1_0
- myAXIpwm_v1_0_S00_AXI the AXI slave interface component, part of the IP component itself



Now we'll add the source code of your previously designed PWM component:

PROJECT MANAGER - edit_myAXIpwm_v1_0				
Sources	? _ 🗆 🖾 ×	Project Summary × Package IP - n	nyAXIpwm ×	
$\mathbf{Q} \mid \mathbf{\Xi} \mid \mathbf{E} \mid $	٥	Packaging Steps	Ports and Interfaces	
> • # myAXIpwm_v1_0(arch_imp) (myAXIpwm_v1_0.vhd) (1)		✓ Identification	Q ¥ ♦ + ⊕	C
> Constraints		 Compatibility 	Name	Interface Mode
> Simulation Sources (1)		✓ File Groups	> SOO_AXI	slave
		 Customization Parameters 	> I SOO_AXI_RST	slave
		 Ports and Interfaces 	> Soo_AXI_CLK	slave



PROJECT MANAGER - edit_myAXIpwm_v1_0

It's now time to integrate an instance of our original PWM within the AXI S00 instance and to map ports + generic parameters ... and extend AXI_S00_inst accordingly.

Finally, extend both generic parameters and ports of the top-level IP and map them to the AXI_S00 instance.

Add the various generic parameters as editable with a proper range of values.

In the end, you'll see your pwm inst integrated

within the AXI interface instance





In order to help yourself: have a look to VHDL 'AXI' labelled files already available for download (see Links

Here are the details of the mapping between our AXI registers and our PWM component. It's now time to complete the modification of our AXI interface.



[optional] AXI simulation with the AXI VIP

In order to enable a simple testing of your AXI-enable PWM IP, we'll make use of the AXI verification IP https://support.xilinx.com/s/article/1058302?language=en_US

to be continued

Block design

It's now time to create a *Block Design* featuring the PS part (i.e Dual ARM9 CPU) and your AXI-enabled IP. In the end, you'll need to *Create Port* (right-click in design) that will be named 'PWM_OUT': connect it with your PWM output.



Address Path Properties			? _ 🗆 ग 🗙	
1 /myAXIpwm_0/S00_AXI			← ⇒ ≎	Your PWM registers are set to 0x43C0_0000 (linux mmap later)
¥ ≑				Using the Address Editor, you can change some parameters like the range value.
Name	Base Address	Range	High Address	In the Block Design view, click to configure your myAXIpwm 0:
Source				• svs clk (50MHz)
/processing_system7_0				• Sys_{i} ($(1Hz)$)
/processing_system7_0/M_AXI_GP0	0x4000_0000	1G	0x7FFF_FFFF	• pwin_cik (THZ),
✓ Apertures				• duty_res (4)
//processing_system7_0/M_AXI_GP0	0x4000_0000	1G	0x7FFF_FFFF	
Connections				It's now time to Validate your design then Generate Block Design
/ps/_0_axi_periph/s00_couplers/auto_pc		10		
Image: A control of the second sec	0X0	4G	UXFFFF_FFFF	Add a constraint file to map the PWM OUT to an onboard LED.
Image: With the second seco	UXU	4G	UXFFFF_FFFF	At the Project Managerl, Design Source right click on mvAXlpwm \rightarrow add HDL wrapper
v + /ps/_0_axi_periph/xbar	0.40	10		
/// /ps/_0_axi_periph/xbar/S00_AXI	0x0	4G	OXFFFF_FFFF	then launch Cenerate hitfile (i.e. it will undertake both IP synthesis, whole desir
Post_o_axi_periprixbar/woo_Axi Section	UXU	40	UXFFFF_FFFF	then radicer denerate bitme (i.e. it will diridertake both in synthesis, whole desig
Pestilation				synthesis + implementation too)
/myAXIpwm_0/S00_AXI	0x0	16	0xF	
a		10	074	
General Path Apertures				
agram × Address Editor × Addr	ress Map	× d	esign_1.vhd	×
	Lan	a un	assigned (0)	Excluded (0)
2. 🖹 🚔 🕂 🗍 🗹 Assign	ied (1)	O 01	assigned (0)	
2 🛣 🤤 ↓ 🗂 🕑 Assign ame	ied (1)	V 01	^1 Inter	face Slave Segment Master Base Address Range Master High Address
A 🛣 🤤 ↓ Ĵ 🕑 Assigr ame 😫 Network 0	ied (1)	e un	^1 Inter	face Slave Segment Master Base Address Range Master High Address
A ★ ♥ ↓ ♪ ♥ Assign ame Processing_system7_0	ied (1)	V 01	^1 Inter	face Slave Segment Master Base Address Range Master High Address
Assign Assign	ed (1) address bit	s : 0x4	1 Inter	face Slave Segment Master Base Address Range Master High Address
Assign Assign	ed (1) address bit	s : 0x4	1 Inter 00000000 [1G] S00_A	face Slave Segment Master Base Address Range Master High Address XI S00_AXI_reg 0x43C0_0000 ⊘ 128 ▼ 0x43C0_007F



Bitstream generation and export hardware

As previously undertaken, see <u>Bitstream generation</u> to generate a '.bit' file.

Next step is to Export a hardware description of our global system in order to develop a simple, bare metal, application that will interact with our PWM IP.

<u> </u>	•			🚴 🖈 Export Hardware Platform 🗸 🗸	^ X
<u>F</u> ile	<u>E</u> dit F <u>l</u> ow <u>T</u> ools Project	Rep <u>o</u> rts	<u>W</u> indow La <u>v</u> out <u>V</u> iew	Output Set the platform properties to inform downstream tools of the intended use of the target platform's hardware design.	4
	Add So <u>ur</u> ces <u>C</u> lose Project	Alt+A	BLOCK DESIGN - design_1 Sources Design × Sign	 Pre-synthesis This platform includes a hardware specification for downstream software tools. 	
	<u>S</u> ave Block Design S <u>a</u> ve Block Design As	Ctrl+S	이 꽃 뇌	 Include bitstream This platform includes the complete hardware implementation and bitstream, in addition to the hardware specification for software tools. 	
	<u>C</u> lose Block Design Chec <u>k</u> point	Þ	 design_1 External Interfaces Interface Connections 	< <u>B</u> ack <u>N</u> ext > <u>F</u> inish Cance	5
	<u>C</u> onstraints	Þ	> 🚍 Ports > 🚞 Nets	👗 🖈 Export Hardware Platform 🗸 🔨	×
1		•	> <pre>> <pre> myAXIpwm_0 (myAXIpwr > <pre> processing_system7_0 (Z</pre></pre></pre>	Files	٦
	Text E <u>d</u> itor I <u>m</u> port	⊢ ⊢	>	Enter the name of your hardware platform file, and the directory where the XSA file will be stored.	•
	Expor <u>t</u>	•	Export <u>H</u> ardware	XSA file name: mvAXIpwm system	
	<u>P</u> rint	Ctrl+P	Expor <u>t</u> Block Design		
	E <u>x</u> it		Export Bitstream File	Export to: //nts/nome/trancois/zyboz/-Axi-pwm	
✓ RT	L ANALYSIS		Export Si <u>m</u> ulation	The XSA will be written to: /nfs/home/francois/zyboz7-AXI-pwm/myAXIpwm_system.xsa	

Launch Vitis IDE

At this point, you have a '.xsa' file that describes our whole system featuring the *myPWM* IP block.

By means of the **Vitis IDE**, we'll import the hardware design to develop a bare metal app. able to interact with our PWM IP on its AXI bus.



A 🔨						
<u>F</u> ile <u>E</u> dit F <u>l</u> ow	<u>T</u> ools	Rep <u>o</u> rts	<u>W</u> indow	La <u>y</u> out	<u>V</u> iew	<u>H</u> elp
		<u>V</u> alidate Desi	gn			F6
Flow Navigator	(Create and P	ac <u>k</u> age New	IP		
✓ PROJECT MANAGER		Create Interf	ace Definitio	n		
Settings		Enable Dynai	mic Function	eXchange.		
Add Sources	<u> </u>	<u>R</u> un Tcl Scrip	t			
Language Templ	. F	Property Edit	or			Ctrl+J
₽ IP Catalog	, ,	Associate EL	<u>F</u> Files			
. 2	(Generate Me	mory Config	uration File	e	
✓ IP INTEGRATOR	(Compile Sim	ulation Libr <u>a</u>	ries		
Create Block Des	1	Vivado S <u>t</u> ore.				
Open Block Desi	(C <u>u</u> stom Com	mands			•
Generate Block [l	Launch Vitis I	DE			
Concrute Dioen	ΩI	Language <u>T</u> e	mplates			
✓ SIMULATION	\$	Settings				

🚽 🗶 👘	Vitis IDE Launcher	\sim	×
Select a direct	tory as workspace the workspace directory to store its preferences and development artifacts.		
<u>W</u> orkspace:	/nfs/home/francois/workspace Browse		
 Use this a Restore ot 	is the default and do not ask again Ther Workspace Cancel Laur	nch	

Create an 'application project'



You'll then get prompted to create a new platform from your previously generated XSA file.

 ✓ ✓ New Application Project ✓ ∧ >
Platform
Note: A platform project will be generated automatically in workspace for the selected XSA. It can be customized later.
Select a platform from repository
Hardware Specification
XSA File: /nfs/home/francois/zyboz7-AXI-pwm/myAXIpwm_system.xsa Browse
Boot Components Generate boot components
Platform name: myAXIpwm_system
Cancel Finish
 ✓ ✓ New Application Project ✓ ∧ >
Application Project Details Specify the application project name and its system project properties
Application project name: myPWM_app
System Project Create a new system project for the application or select an existing one from the workspace
Select a system project details
+Create new System project name: myPWM_app_system
Target processor
Pack Next > Cancel Finish

*

Once your application project name has been defined, you'll select 'standalone' OS (i.e bare metal --- no OS) allong with 'Empty C++ app'

√ ★	New Application Pro	oject	~ ^ X	😼 *	New Applic	ation Project		~ /
Domain Select a domain for your proje	ct or create a new domain			Templates Select a template to creat	e your project.			••
Select the domain that the ap Note: New domain created by selected in the next step Select a domain	plication would link to or cre r this wizard will have all the r Domain details	ate a new domain requirements of the applicat	ion template	Available Templates: Find: Find: Embedded software of Dhrystone	E levelopment templates	Empty Applicati An empty C++ Ap	on (C++) pplication	
Create new	Name: Display Name: Operating System: Processor: Architecture:	standalone_ps7_cortexa9_ standalone_ps7_cortexa9_ standalone ps7_cortexa9_0 32-bit		Empty Application Empty Application(Hello World IwIP Echo Server IwIP TCP Perf Clien IwIP TCP Perf Serv IwIP UDP Perf Clien	(C++) C) t er nt ver			
?	< Back	Next > Cancel	Finish	Memory Tests	-+			
				?	< Back	Next >	Cancel	Finish



... then right click on the src folder to add a new file ...

... click on *Advanced* to link to the existing <code>myAXIpwm.c</code> file you'll find from the <u>Practical exercises base files</u>

🖌 🗶 Create New File 🗸 🗸	×
File Create a new file resource.	-
Enter or select the parent folder:	
myAXIpwm/src	
▼ 🖉 myAXIpwm [standalone_ps7_cortexa9_0]	
▶ Ø_ide	
src Since Si	
🗁 myAXIpwm_system [zyboz7-AXI-pwm]	
RemoteSystemsTempFiles	
r ≥ zyboz7-AAI-pwm	
File name: myAXIpwm.d	
<< Advanced	
/muAXInum c Prouso Variables	
	· _
Choose file system: default 🔻	
Resolved location: file:////www.c	
Cancel Finish	

Compilation

Select 'myAXIpwn_system' (it's important to select the system level), then click on build to launch the compilation ... you may need to undertake some corrections within your C++ source code that will interact with your IP block.



Zybo board | JTAG mode

Before testing your whole project on your hardware, set **JP5** to **JTAG** mode before powering on the Zybo board.



... then upload both FPGA configuration along with your standalone app. by clicking on the RUN icon within the assistant frame



You can follow the whole upload within the XSCT console

XSCT console
attempting to launch hw_server
***** Xilinx hw_server v2022.1.0 **** Build date : Apr 10 2022 at 06:24:21 ** Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
INFO: hw_server application started INFO: Use Ctrl-C to exit hw_server application
INFO: To connect to this hw_server instance use url: TCP:127.0.0.1:3121
<pre>initializing 0% OMB 0.0MB/s ??:?? ETA 28% 1MB 2.2MB/s ??:?? ETA 50% 1MB 1.9MB/s ??:?? ETA 72% 2MB 1.8MB/s ??:?? ETA 93% 3MB 1.8MB/s 00:02 Downloading Program /home/devel/workspace/myAXIpwm/Debug/myAXIpwm.elf section, .text: 0x00100000 - 0x001035db section, .init: 0x001035dc - 0x001035d5 section, .fini: 0x001035d4 - 0x00103798 section, .fini: 0x001035d4 - 0x00103798 section, .data: 0x0010370 - 0x0010401f section, .data: 0x00104020 - 0x00104023 section, .mu_tbl: 0x00108000 - 0x00104023 section, .init_array: 0x0010c00 - 0x0010c0c7 section, .fini_array: 0x0010c0c8 - 0x0010c0cb section, .fini_array: 0x0010c0c7 section, .fini_array: 0x0010c0c7 section, .fini_array: 0x0010c0c6 - 0x0010c0cb section, .heap: 0x0010c133 section, .heap: 0x0010c134 - 0x0010e13f section, .heap: 0x0010c134 - 0x0010e13f</pre>
0% OMB 0.0MB/s ??:?? ETA 100% OMB 0.5MB/s 00:00 Setting PC to Program Start Address 0x00100000 Successfully downloaded /home/devel/workspace/myAXIpwm/Debug/myAXIpwm.elf Info: ARM Cortex-A9 MPCore #0 (target 2) Stopped at 0xffffff28 (Suspended) xsct% Info: ARM Cortex-A9 MPCore #0 (target 2) Running

... if everything goes right, you now have a 1s blinking led 🙂

[M2] AXI IP encoder

TO BE CONTINUED

[M2] PicoRV32

PicoRV32 https://github.com/YosysHQ/picorv32

END