### Preface

Thanks so much for Mr. Joseph Attard for writing this amazing tutorial book for the <u>Z-turn Board</u>. Joseph is a Senior lecturer II working at Malta College for Arts Science and Technology on the island of Malta. He became an electronics enthusiast since a very early stage which culminated in a BEng Hons degree in Electronic Systems from Portsmouth University UK (2010) and a Master degree in MicroElectronic and MicroSystems from the University of Malta (2016). He has gained great experience from building various embedded systems using both PIC microcontrollers and Xilinx FPGAs. He makes sure to keep up with technology and always finds time to learn new microprocessor/FPGA systems. His passion led him to Xilinx Zynq 7 System-on-Chip (SoC), and after an extensive research on available Computer-on-Module Boards, he settled for MYIR's Z-turn Board which is one of the best cost per available-peripheral Computer-on-Module boards, available on the market. Joseph's email address is *pic18f4455@yahoo.com* 

In this book, Joseph shared a lot of content on how to work with the Z-turn board, starting from simply creating a project in Vivado to flash an LED, continuing to Detecting Switch inputs, all the way to interfacing the Zynq 7 System on Chip to multiple analogue sensors through multiple XADC channels. All the above-mentioned interfacing is done from both the ARM Cortex A9, commonly known as the Processing System and the Artix 7 FPGA, commonly known as Programmable Logic, both residing within the Zynq 7 SoC.

Being a lecturer, Joseph did not simply show the steps of how to achieve the goals one is set to achieve when using the Z-turn Board, but also pointed out hidden procedures, one has to undergo, while implementing these steps. He makes sure to explain the reasons why, one has to go through the required hidden procedures, and this added material, makes it really helpful for beginners and established engineers alike, to quickly get used to the Z-turn board.

This book includes a lot of C code and VHDL code written by the author himself. This is accompanied by lots of comments and explanations from where the C functions are derived in SDK which for the novice engineer would be quite difficult to understand. All VHDL code is originally written by the author and one has to have a good understanding of how VHDL works to obtain the full benefit of this book.

The Z-turn board is one of the best off-the-shelf Computer on Module Boards available in the market today. It has a vast array of peripherals ranging from very high-speed interface connectors, HDMI, USB etc. It is advisable by the author to invest a little bit more in MYIR's <u>Z-turn Cape IO</u> <u>board</u> which could be connected directly underneath the Z-turn board. This Cape IO board offers better IO capabilities for those users who would like to interface the Z-turn Board to external peripherals such as LEDs, switches, motors, sensors, etc.

Given the vast amount of peripherals present on the Z-turn board and the amount of computer

power present on the Zynq 7, one cannot ignore the potential one can achieve, in areas such as Machine Learning, Machine Vision and AI. By writing this book, Mr. Joseph Attard and MYIR are hoping to make the Z-turn board, the preferred choice for both novice engineers and experienced engineers alike, in their endeavor to learn how to work with Xilinx Zynq 7 SoC.

This work will not stop here and MYIR encourage more and more players to join in sharing knowledge with the general public for a better future.

# Catalogue

- Chapter 1...... Creating a Project for the Z\_turn ONLY for the FPGA part of the Zynq 7
- Chapter 2...... Steps to create an A9 Hard Core project using both Vivado and SDK
- Chapter 3...... Flashing LEDs from both Processing System and Programmable Logic
- Chapter 4...... Detecting Switch Inputs from Programmable Logic
- Chapter 5...... Using the DIP switches with the Processing System
- Chapter 6...... Interfacing with the Button Switch on the Z-turn board
- Chapter 7...... Processing System Dual AXI block control
- Chapter 8...... Information on XADC
- chapter 9...... Sampling External ADC from the Processing System
- Chapter 10...... Multiple Analogue Sensing using XADC Data is common to both PS and PL V2
- Chapter 11...... Event driven sampling of multiple XADC channels from the Programmable Logic

# **Getting Started**

The first point of reference to start working with the Z-turn board is a Youtube video that shows the link from where to download the board-support-files and how to install them correctly. This is given below:

https://www.youtube.com/watch?v=VDYoweTZtfU

This video will introduce the Github link below, from where to download the board-support-files.

https://github.com/q3k/zturn-stuff

Then the Github site will take you to the following wiki page:

https://reference.digilentinc.com/reference/software/vivado/board-files?redirect=1

The above is the wiki page that shows how to install the board files for Vivado. The following sections show how to install the board support files for Vivado.

### Installing the Board-Support-Files

The board files are used by Vivado. These consist of XML files used by Vivado to recognize various development boards. The board files were downloaded from Github link stated above.

The board file folder must be copied to the location shown by figure 1 below:

lonie Share View		
↑ 📜 > This PC > Local Disk (C:) > Xilinx > Viva	do > 2017.4 > data > boards > board_files	
25005	Name	Date modified
access	📜 ac701	02/02/2018 16:42

Figure 1: Location where to copy the Board support Files

Figure 2 below shows the copied folder in the **board\_files** directory within the **Xilinx** Directory. The folder must be copied **as is**! That is, **<u>do not remove or add any files</u>** to the copied folder!

Vcu1525	02/02/2018 16:42	File folder
📕 xm105	02/02/2018 16:49	File folder
📕 zc702	02/02/2018 16:43	File folder
📕 zc706	02/02/2018 16:48	File folder
zcu102	02/02/2018 16:50	File folder
zed	02/02/2018 16:43	File folder
🣕 zturn-7z020	05/02/2018 19:53	File folder

Figure 2: Z-turn folder seen with the other board support files within Xilinx Directory

Sh	are View	there are two folders	_
41	GitHub_Z_turnBoard_stuff > zturn-stuff-master > zturn-stuff-master > boards >	> board_files > zturn-7z020 > 2.1	ບ v ບ Sear
	Name	Date modified you should save them as	Туре
	board	they are 05/02/2018 19:21	XML Document
	part0_pins	05/02/2018 19:21	XML Document
	D preset	05/02/2018 19:21	XML Document
	□ · ·		

Figure 3: Github folder contents

Figure 3 above shows the contents of one of the folders downloaded from Github. It is advisable to copy both folders into the Xilinx directory as they are in the folder indicated by figure 2.

The following section shows how to create a project for the Z-turn board in Vivado. The project will only include a simple VHDL module, therefore only the **Programmable Logic** part of the **Zynq 7 System-on-Chip** will be used. The following sections show the full procedure, right up to programming the **Zynq 7**.

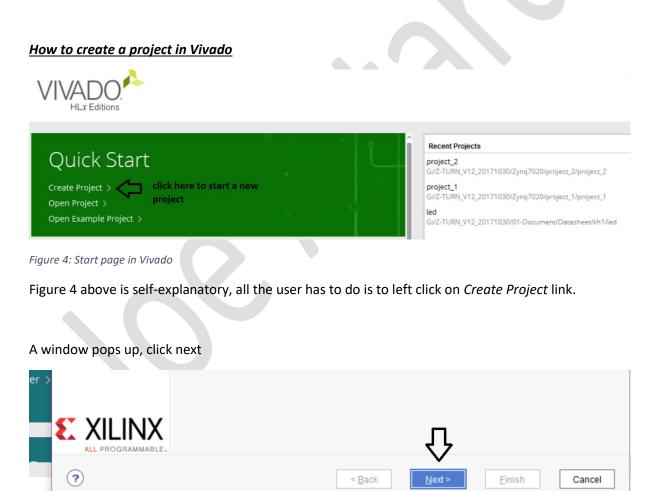


Figure 5: Pop Up Window 1

For the first pop-up window, click on Next.

Project Name Enter a name for your project and specify a directory where the project data files will be stored.	
Project name: LightingLED1 name the project here	
Project location: G:/Z-TURN_V12_20171030/Zynq7020/LightingLED1 Create project subdirectory Project will be created at: G:/Z-TURN_V12_20171030/Zynq7020/LightingLED1/LightingLED1/LightingLED1	2
name of the folder	
leave this ticked	



Figure 6 shows where to write the name of the new project, and what one needs to do, to store in the desired location within the PC.

Clic	< 0	on <b>NEXT</b> again.	
	-	r <b>ct Type</b> y the type of project to create.	
(		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         Image: Do not specify sources at this time	
(	D	Post-synthesis Project: You will be able to add sources, view device resources, run design analysis, planning and implementation.	

### Figure 7: Project Type Window

Since in this project, a VHDL module is going to be created, then one should leave the window shown in Figure 7 as is. Click on **NEXT** again.

À New Project	t		×
Add Sourc	es		
	· · ·	P files, or directories containing those files, to add to your project. Create a new iect. You can also add and create sources later.	
ۍ ۍ	lick the ADD butto	n here	
+, -			
	Add Files		
	Add Directories		
	Create File	select "create file"	
_			

Figure 8: Adding the VHDL source file

In the next window, shown by Figure 8 in the previous page, the user will be asked whether any source files shall be created and/or included in the project. So, one must first click on the **plus (+)** sign in the left corner and then select **create file** from the drop-down list.

This will lead to another pop-pup window which asks for the name of the module and whether VHDL or Verilog will be used as the preferred language for the module created. Since the author only knows how to program in VHDL, then the file type is going to be VHDL. The file should be given a name that is related to the function of the module. In this case, since the VHDL module is going to light an LED, the name implies the module's function! All this is shown in figure 9 below:

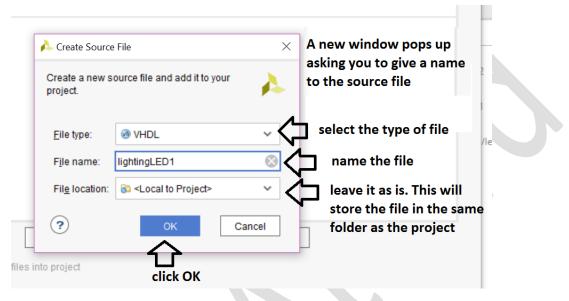


Figure 9: Give a name to the VHDL module

Figure 10 below shows the new VHDL module is part of the project. One could add as many modules as needed. This could become handy if a top-down approach is used to build the system.

Add So	ources							
				ectories containing those fi also add and create sourc			Create a new	
J	, con	tinue adding fi	les <mark>from h</mark> e	re				
+	-	+   +						
	Index	Name	Library	HDL Source For		Location		
Ø	1	lightingLED1.vhd	xil_defaultlib	Synthesis & Simulation	•	<local project="" to=""></local>	$\langle \Box$	
		or continue	adding file	s from here other	wise	e click NEXT		
		]	Add Files	Add Directories		Create File		
		l						

*Figure 10: Add source files to the project* 

the next pop up window asks you if you are going
e used in the source file
Back Next > Einish Cancel

No constraints files added at this time!!

+ |-| + | + |

### Figure 12: Constraints File

below

Vivado will then ask whether a constraints file should be added. Since the board support files have been included, one does not need to include a constraints file at this stage, so click **NEXT** for this window, without doing any changes.

Select:  Parts Filter	Boar	ds 🤇	bet		r vendor	has ad			lect the par L file for the	
Product category:	All		~		Spe	e <u>d</u> grade	: All	~		
<u>F</u> amily:	All				✓ <u>T</u> en	np grade:	All	~		
Package:	All	~		Rese	et All Filters	the	n the o		elected aints file d	
Search: Q-				~		aut	omati	cally		
Part		I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gb Transceivers	GTPE2 Transc
xc7z015clg485-2		485	150	46200	92400	95	0	160	4	4
xc7z015clg485-1		485	150	46200	92400	95	0	160	4	4
@ xc7z015iclg485-1L		485	150	46200	92400	95	0	160	4	4
xc7z020clg400-3		400	125	53200	106400	140	0	220	0	0
@		400	125	53200	106400	140	0	220	0 l selecte	d the p
xc7z020clg400-2										
xc7z020clg400-2 xc7z020clg400-1		400	125	53200	106400	140	0	220	0	0 <

*Figure 13: Choosing the Zynq 7 SoC according to part number* 

In the next pop-up window, one can either choose the Zynq 7 according to the part number resident on the development board, or even better one can choose the board itself by first clicking on the

**boards** tab and then choose the Z-turn board from the list. This is only possible **if** the Board Support files of the relative board are included in the Xilinx folder as described earlier in this chapter.

Select: 🧔 Parts	Boards 1	) click on boa	rds			
Filter/ Preview						
Ve <u>n</u> dor:	All	~				
Display <u>N</u> ame:	All	~				
Board Re <u>v</u> :	Latest	~				
	Reset All Filters					
Search.	ou need to install th	*	-			
fir	st for this to be ava	ailable. I have	covered this			
Display Name in	another document	t	Vendor	Board Rev	Part	
Kintex-Ultrascale	e Alphadata board		alpha-data.com	1.0	xcku060-ffva1156-2-e	
ZedBoard Zyng Evaluation and Development Kit			em.avnet.com	d	xc7z020clg484-1	
Z-turn Board (MY	(S-7Z020-C)	elect your bo	a royir.com	4	xc7z020clg400-1	

### Figure 14: Selecting the Z-turn Board from the list

Click on <b>NEXT</b>
----------------------

	New Project Summary
HLx Editions	A new RTL project named 'UART_LED' will be created.
	👴 No source files or directories will be added. Use Add Sources to add them later.
	O No constraints files will be added. Use Add Sources to add them later.
	The default part and product family for the new project: Default Board: Z-turn Board (MYS-7Z020-C) Default Part: xc72020clg400-1 Product: Zynq-7000 Family: Zynq-7000 Package: clg400 Speed Grade: -1
EXILINX	To create the project, click Finish
•	<back next=""> Einish Cancel</back>

Figure 15: Project Summary

Figure 15 is the last window that pops up while setting up the project. It contains a summary of all the previous settings done. All that needs to be done is to click on *FINISH*.

Once *FINISH* is pressed, a new window pops up. In the new window, one can enter the inputs and outputs of the VHDL module that was created before.

🝌 D	efine Module											$\times$
Foi N F	fine a module a each port spe ISB and LSB va Ports with blank en this wine	cified: alues w cnames	rill be s will r	ignore( not be v	d unless written.	its Bus	s column is	checke		tity jus	st like X	ise
Me	odule Definition	ı										
	Entity name:	[	lightir	gLED1	1						6	3
	Architecture name: Behavioral								6	3		
	I/O Port Defini	itions										
	+  -		₽									
	Port Name	Dired	tion	Bus	MSB	LSB						
	LED	out	~		0	0			click	on NE	хт	
	分	$\mathbf{\hat{1}}$	4		4				Ĺ	ļ		_
?	give the whethe	-			-				OK or		Cance	эI
	input											

Figure 16: Define the inputs and outputs of the VHDL module

Now, at this point, one might not know exactly how many inputs and outputs, the module might end up with, however if one has any idea of any common inputs and outputs that the module might have(such as the clock and the reset inputs), one could include them immediately in the table shown in figure 16 above. However, if currently, the user does not have any idea what the names of the inputs/outputs are going to be, it is perfectly safe to just click on the **OK** button and continue with the next pop-up window without submitting any names.

Vivado opens and one can find the source files as shown in *Figure 17* below.

	< 🕨	ta \$ Σ ≈ % ×			
Flow Navigator 🛛 😤 🌲	? _	PROJECT MANAGER - LightingLED1			
PROJECT MANAGER	Î	Sources all the source	files reside ? _ D 🛚 X	Project Summary	
Settings Add Sources			*	Settings Edit	
Language Templates		ම lightingLED1(Behavioral double click හා the name of	f the source file to open it	Project name: Project location:	LightingLED1 G:/Z-TURN_V12
IP INTEGRATOR		Hierarchy Libraries Compile	Order ? _ □ Ľ ×	Product family: Project part:	Zynq-7000 xc7z020clg400-
Create Block Design			A   A   A	Top module name:	lightingLED1



Double clicking on the VHDL source file so that one can write the VHDL code that eventually will be translated into hardware later by Vivado.

₩13 ₩ <b>7 Z</b> ZZ X/ XS				E Delault Layout
ROJECT MANAGER - LightingLED1				
Sources	? _ 🗆 🖒	$\times$	Project Summary × lightingLED1.vhd ×	
Q   ¥   ♦   +   ?   ● 0		¢	_20171030/Zynq7020/LightingLED1/LightingLED1/LightingLED1.srcs/so	urces_1/new/lightingLED
<ul> <li>Design Sources (1)</li> <li>IightingLED1(Behavioral) (lighting</li> </ul>	gLED1.vhd)	^		the source file
Hierarchy Libraries Compile Order			<ol> <li> Uncomment the following library declaration if i</li> <li> any Xilinx leaf cells in this code.</li> </ol>	appears here. Scroll down an
Source File Properties	? _ 🗆 🖒	×	31 ;library UNISIM; 32 use UNISIM.VComponents.all; 33 ;	write the architecture
IghtingLED1.vhd	$\leftarrow$ $\rightarrow$ $\mid$	•	34 🖯 entity lightingLED1 is	
General Properties			35 Port ( LED : out STD_LOGIC);	

Figure 18: Typical VHDL source file

Figure 18 and Figure 19 show the same VHDL file and code. In this project, an LED will be lit. It is a simple instruction, however at this point, the objective of this chapter is to show all the steps needed to develop a Zynq 7 project that will operate only the Programmable Logic part.

Sources	? _ 🗆 🖾 X	Project Summary × LEDonOff.vhd * ×
Q     ₹     €     +     ⊡     0       ~ □     Design Sources (1)       @▲     LEDonOff(Behavioral) (LEDonOff.vho       Hierarchy     Libraries     Compile Order	• 	-TURN_V12_20171030/Zynq7020/LEDonOff/LEDonOFF/LEDonOFF.EDonOFF.sr Q A X B V Q X Q Q X Q Q Q Q Q Q Q Q Q Q Q Q Q Q
Source File Properties	? _ 🗆 🖾 X	40 begin 41 LEDout(0) <= '1';
LEDonOff.vhd     General Properties	♦   ♦   ♦	42 LEDout(1) <= '0'; 43 LEDout(2) <= '1'; 44

Figure 19: Simple VHDL instruction

Once the VHDL code is written, the code must be saved from the icon shown in Figure 19. By saving the file, Vivado is also checking the syntax and if there is any syntax error, Vivado will pop up a window.

# Creating a Block Design

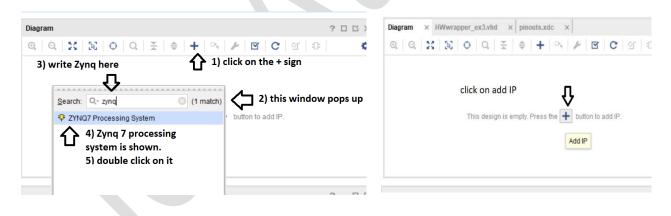
The next step is to create a **Block Design**. This approach is the easiest approach one should take when developing a project for the Zynq 7 System-on-Chip especially because the board support files of the Z-turn board are already included in Vivado. That way, Vivado would know the features and parameters of the Z-turn board and would give warnings or error messages if one would try to use hardware that is outside the hardware settings of the Z-turn board. Figure 20 on the next page, shows the steps one has to go through to create a block design. Once the block design has been created, a new file-type will be generated by Vivado where a **schematic representation** of the hardware could be drawn. This replaced the *canvas* in Xilinx ISE. The next step is to include the processing system as shown in figure 21 on the next page.

1	Settings	Q ≍ ≑ + ? ●	0	0		
	Add Sources		0	Settings	Edit	
	Language Templates	Design Sources	🝌 Create Block Design	1	×	UART_LED
ւ	<sup>₽</sup> IP Catalog	Hierarchy Libraries Compile	Please specify name	of block design		G:/Z-TURN_\
				ock design a name	se 🔑 🕹	Zynq-7000
✓ IF	PINTEGRATOR	Properties	L/ Bive your bit	sek design a name		Z-turn Board
	Create Block Design	k on "create block n"	Design name:	blockdesign 🧲	× 1	Not defined
	Open Block Design	Select an object to	Directory:	🗟 <local project="" to=""></local>	~	
	Generate Block Design		Specify source set:	Design Sources	~	
× s	IMULATION	Tcl Console Messages Log				
	Run Simulation	Q	? 3) click OK	ОК	Cancel	
		Name Constraints S	tatus wivo i		wo rotarrov	ver Failed F

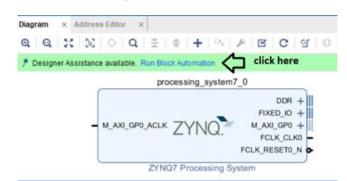
Figure 20: Creating a Block Design

### Why do we have to include the Zynq Processing System in our Design?

The reason for including a **Zynq Processing System** in our design is because *it is the only way* how one can download the VHDL code to create hardware in the **Programmable Logic** part! This means that when the **boot-image file** is created later in SDK, this will be fetched by the Processing System of the SoC and after reading it, the hardware part will be configured in the Programmable Logic. So even though in this example, only the Programmable Logic part is going to be active, the Processing System part must also be included in the hardware design! Another reason for including the Zynq Processing System in the hardware design is because the 100 MHz clock required by the sequential circuits within the Programmable Logic part can only be provided by the Zynq Processing Part! Figure 21 shows the three steps needed to create a Zynq Processing System. There are **two ways** how to add the Zynq Processing System both shown in Figure 21.









Now that the Zynq processing system is included in the schematic, it would be a good idea to click on **Run Block Automation** as shown in Figure 22. This will enable the board settings as provided by Github and therefore establish the peripherals that are already connected on the Z-turn board. Figure 23 below shows a pop-up window that asks the user to confirm the pre-set board settings.

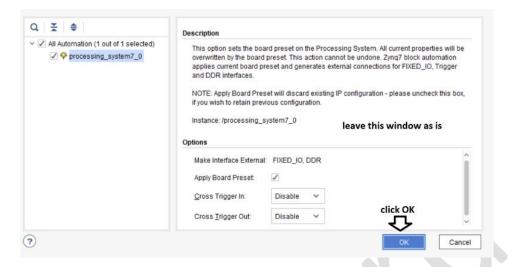


Figure 23: Preset Z-turn Board Settings

Just click **OK** on this window and leave everything as is.

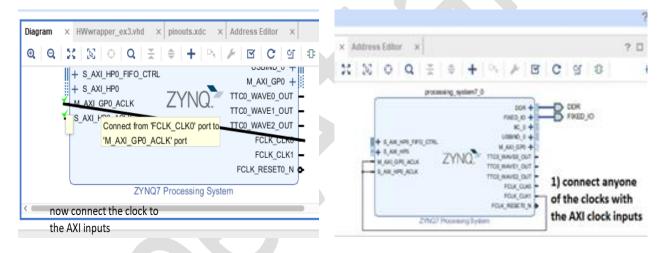


Figure 24: Expanded Zynq Processing System Block Diagram

Figure 24 shows the expanded version of the *Zynq Processing block* after the *Run Block Automation* has been enabled. Not to generate any errors, the next step is to connect the *FCLK\_CLKO* which is the main *100 MHz* clock output of the Zynq Processing System, to the AXI inputs at the left of the Processing System block.

Next include the VHDL module written previously. This is done by right clicking on the canvas or Block Design window, select Add Module from the list. A new window pops up with the suggested VHDL

Diagram (0, 0,		lashingLED_attempt1.vhd		ngLED5.vnd ×  ?
			1:	write click on the canvas
		Source File Properties	Ctrl+E	].
	X	Delete		e 🕂 button to add IP.
		Сору	Ctrl+C	
		Paste	Ctrl+V	
	O,	Search	Ctrl+F	
	We:	Select All	Ctrl+A	
	+	Add IP	Ctrl+I	? _
		Add Module 🗲 2	: Select '	'Add Module"
LEDattemp		IP Settings		<pre>ittemptl.srcs/sources_l/new/Flashin</pre>
attempt1/	g	Validate Design	F6	<pre>:mptl.srcs/sources_l/new/FlashingLE</pre>
		Create Hierarchy		

modules highlighted. Double click on the module and it will be added in block form in the schematic diagram.

### Figure 25: Adding a VHDL module to the schematic

🝌 Add Module	× Figure
Select a module to add to the block design.	Figur
Module type: RTL V	conv sche here writt
FlashingLED_attempt1 (FlashingLED_attempt1.vhd)	mod
this window pops up	wind could
click on the vhdl module you have created	
✓ Hide incompatible modules	
OK   Cancel	
Diagram × HWwrapper_ex3.vhd × p	inouts.xdo
$\mathbf{Q} \mid \mathbf{Q} \mid \mathbf{X} \mid \mathbf{\Sigma} \mid \mathbf{\Theta} \mid \mathbf{Q} \mid \mathbf{X} \mid \mathbf{\Theta}$	⊧   +
HWwnapper_ox3_0	

#### Figure 26: Selecting the VHDL module

Figure 26 shows the VHDL module that can be converted into block diagram to be added to the schematic diagram of the project. It must be noted here that if more than one VHDL modules have been written and are part of the project, all of these modules will be listed in the lower part of this window. One can select which one of the modules could be included in the schematic.

Diagram × HWwrapper	_ex3.vhd × pinouts.xdc	× Address Editor	×
<b>Q</b>   <b>Q</b>   💥   🕅   🗇	Q   ≝   ⊕   <b>+</b>   □	- 🔌 🗹 C	9 :0:
HWwrapper_ex3_0 HWwrapper_ex3_v1_0	Z		+ DOR FIXED_IO

Now you can actually DRAG the .VHD file from the source window into the canvas and the VHDL hardware block is created

Figure 27: VHDL module in schematic

Figure 27 shows the VHDL module is part of the schematic. Since in this particular project, an LED is only to be lit first and then code could be changed for the same LED to flash instead of just having the LED lit steadily, there is no need for a clock input to the VHDL block - it is going to be free-running! That is, it has nothing to do with the PS part of the SoC!

### Creating a Hardware Wrapper

After deciding on which VHDL modules are going to form part of the system, the next step is to create a *hardware-wrapper*. This is another technical name for the *top-level module* of a system where it will include all the components of the system. Failing to do the steps in Figure 28 will result in an incomplete system and therefore will not work in practice!

PROJECT MANAGER 2) go to sour	ces Sources × Design	1	Create HDL Wrapper_	4) choose this	
<ul> <li>Settings</li> <li>Add Sources</li> <li>Language Templates</li> </ul>	Q ± 0 + · Design Sources (1)		View Instantiation Template Generate Output Products Reset Output Products	-	
P Catalog 3) right click on t block design PINTEGRATOR			Replace File Copy File Into Project Copy All Files Into Project	201	
Create Block Design Open Block Design Generate Block Design	Source File Properties	×	Remove File from Project Enable File Disable File	Delete At-Casan At-Manut	

Figure 28: Creating a Hardware Wrapper

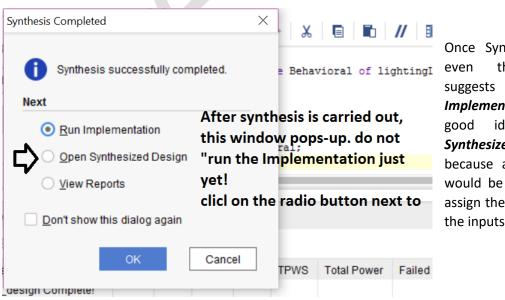
Once the hardware wrapper is created, it is time to synthesize the design. Figure 29 shows how to synthesize the design by clicking once on run Synthesis.

	Source File Properties
Run Simulation	IightingLED1.vhd
✓ RTL ANALYSIS	General Properties
> Open Elaborated Design after writing your code <ul> <li>sclick ons</li> <li>Run Synthesis"</li> </ul>	Tcl Console     Message       Q     ≚     ♦
Synthesis     Northesis     Synthesized Design	Name     Const       ∽ ▷ synth_1     constr       ▷ impl_1     constr

Figure 29: Synthesize the code

Launch the selected synthesis or implementation runs. Launch directory: I and Directory: Options Launch runs on local host: Number of jobs: 4 Generate scripts only Dgnt show this dialog again Cick on OK Cancel Source File Properties I igure 30: Launching Synthesis Source File Properties I igure 30: Launching Synthesis Source File Properties I igure 30: Launching Synthesis Ceneral Properties Console Messages Log Report Pesign Runs X Number of Signer Status Number of Jobs: 4 Cick on OK Cancel Cick on the "Design Runs" tab and notice the synthesis being done Cick on the "Design Runs" tab and notice the synthesis being done	Launch Runs			$\times$					
Launch directory: S Options Launch runs on local host: Number of jobs: 4 Leave the synthesis to complete. Figure 31 show the design runs. Leave the synthesis to complete. Figure 31 show the design runs. Leave the synthesis to complete. Figure 31 show the design runs. Leave the synthesis to complete. Figure 31 show the design runs. Leave the synthesis to complete. Figure 31 show the design runs. Leave the synthesis to complete. Figure 31 show the design runs. Leave the synthesis to complete. Figure 31 show the design runs. Leave the synthesis Leave the synthesis to complete. Figure 31 show the design runs. Leave the synthesis Leave the synthesis to complete. Figure 31 show the design runs. Leave the synthesis Leave the synthesis to complete. Figure 31 show the design runs. Leave the synthesis Leave the synthesis to complete. Figure 31 show the design runs. Leave the synthesis Leave the synthesis to complete. Figure 31 show the design runs. Leave the synthesis Leave the synthesis to complete. Figure 31 show the design runs. Leave the synthesis Leave the synthesis to complete. Figure 31 show the design runs. Leave the synthesis	aunch the selected synthesis or im	plementation runs.		4	on <b>OK</b>	becaus	e it is bet	ter to let th	
Options       31 show the design runs.         Image: stripts only       Image: stripts only         Image: string       Image: string <t< th=""><th>Launch <u>d</u>irectory: 🔂 <default la<="" th=""><th>unch Directory&gt;</th><th></th><th>~</th><th></th><th></th><th></th><th></th><th>igure</th></default></th></t<>	Launch <u>d</u> irectory: 🔂 <default la<="" th=""><th>unch Directory&gt;</th><th></th><th>~</th><th></th><th></th><th></th><th></th><th>igure</th></default>	unch Directory>		~					igure
Generate scripts only         Dgnt show this dialog again         Cick on OK         Cancel         igure 30: Launching Synthesis         Source File Properties         igure 30: Launching Synthesis         Ceneral         Properties         It ightingLED1.vhd         General         Properties         Console         Mame         Constraints         Status         WNS         Name         Constraints         Status         WNS         Ns         WHS         THS	Options					•		•	igure
Dgnt show this dialog again       this window pops-up, click on OK         Igure 30: Launching Synthesis         Source File Properties         ? - I X         41         42         12<	<u>Launch runs on local host</u>	Number of jobs: 4	~						
Dgnt show this dialog again       click on OK         OK       Cancel         igure 30: Launching Synthesis         Source File Properties         @ lightingLED1.vhd         General       Properties         Tcl Console       Messages         Log       Reports         Design Runs       Click on the "Design Runs" tab and notice the synthesis being done         Name       Constraints         Status       WNS         Name       Constraints         Status       WNS         Name       Constraints         Status       WNS         Name       Constraints         Status       WNS         NS       WHS         TNS       WHS         TNS       WHS         TNS       WHS         TNS       WHS         TNS       WHS         Yes       Total Power         Failed Routes       LUT	O Generate scripts only								
igure 30: Launching Synthesis Source File Properties ? - • • • • • • • • • • • • • • • • • •	Don't show this dialog again	•	ps-up,						
Source File Properties       ? _ □ □ ×       41         IghtingLED1.vhd       IED <= '1';		С (к	Canc	el					
Source rile Properties       42       LED <= '1';	ure 30: Launching Synthesis								
Image: Status	Source File Properties	? _ 🗆	Ľ ×						
General       Properties         Tcl Console       Messages       Log       Reports       Design Runs       ×       Click on the "Design Runs" tab and notice the synthesis being done         Q       X       I       X       Y       X       TNS       WHS       THS       TPWS       Total Power       Failed Routes       LUT         V       O synth_1       constrs_1       Running synth_design       Image: Status	IightingLED1.vhd	$\leftarrow$ $\rightarrow$	•	43 E					
Q       X       I       X       I       X       Synthesis being done         Name       Constraints       Status       WNS       TNS       WHS       THS       TPWS       Total Power       Failed Routes       LUT         V       O synth_1       constrs_1       Running synth_design       V       <	General Properties			44	<				
Q       A       A       A       A       A       A       A         Name       Constraints       Status       WNS       TNS       WHS       THS       TPWS       Total Power       Failed Routes       LUT         V O synth_1       constrs_1       Running synth_design       MNS       TNS       WHS       THS       TPWS       Total Power       Failed Routes       LUT	Tcl Console Messages	Log Reports Design Ru	ns ×			-		b and notice	the
✓ U synth_1 constrs_1 Running synth_design	Q ₹ \$  4 ≪	▶ ≫ <del>  %</del>		_		enig de			
			WNS	TNS	WHS THS	TPWS	Total Power	Failed Routes	LUT

Figure 31: Design Runs



Once Synthesis is done, even though Vivado suggests to **Run Implementation**, it is a good idea to **Open Synthesized Design** because at this point, it would be a good idea to assign the external pins to the inputs and outputs.

Figure 32: Synthesis Ready window

### Assigning Pin numbers to the System

Q   ₹   ≑   =		0	$\mathbf{E} \models \mathbf{e}$	ቒ   0,   ;;	$\mathbb{N} \oplus$			
V Internal VREF							P	
🗎 0.6V			$\diamond$					
Drop I/O banks on volta VREF.	ges or the "NONE" folde		~					
I/O Port Properties	× Clock Regions	? _ 🗆 🛙	3					
LEDout[2]		←   →   ∅	F					
			0					
General Properties	Configure Powe	er en			A20	Column 1996	-	
					B19			
Tcl Console Messa	ges Log Reports	Design Runs Pa	ackage Pins I/O	Ports ×	B20			? _
Q, ₹ ♦ ₦	+ 13				C20			select a voltage
Name	Direction	Board Part Pin B	Board Part Interface	Neg Diff Pair	D18 D19 Pin	Fixed	Bank	I/O Std
<ul> <li>All ports (3)</li> </ul>					D20	_		$\cdot$
LEDout (3)	001	O planner is not			E17 ~	₩ □		default (LVCMOS18)
	OUT the c	onstraints file so	you need to u	se				default (LVCMOS18)
LEDout[2]								1-
		nenu to start it			ign the pinout	×		default (LVCMOS18)

Figure 33: Assigning Pin numbers

Figure 33 illustrates one way to assign pin numbers to the inputs/outputs of the system. Since in this system, there are only three LEDs used, only these LEDs must be connected. After opening the Synthesized model, click on the *I/O Ports* tab and adjust the pin numbers according to the Z-turn board circuit diagram.

Name	Direction	Board Part Pin	Board Part Interface	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std
~ 🔞 LEDout (3)	OUT					$\checkmark$	34	LVCMOS33*
- LEDout[2]	OUT				Y16 ~	$\checkmark$	34	LVCMOS33*
- LEDout[1]	OUT				Y17 ~	$\checkmark$	34	LVCMOS33*
-	OUT				R14 ~	$\checkmark$	34	LVCMOS33*
Scalar ports (0	)							
<								

Figure 34: Assiging Voltage Levels to pins

Figure 34 shows how to assign voltage levels to pins. These must be *LVCMOS33* not to generate any errors meaning that the outputs are capable of outputting *3V3*. Also, the *square boxes* under *Fixed* column must be *ticked* not to generate any misleading errors!

Now save the new constraints file as shown in Figure 35.

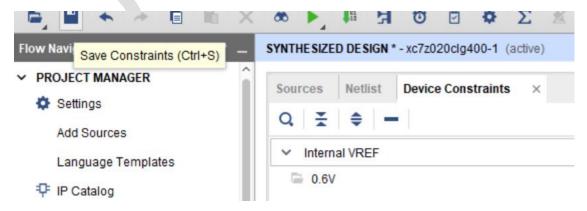


Figure 35: Saving the new constraints file

A pop-up window just like the one shown in Figure 36 shows up. Click on OK.

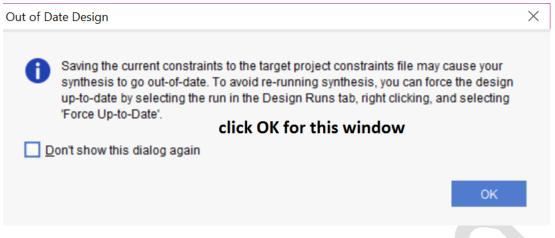


Figure 36: New Constraints File Warning Window

Select a target file to write r Choosing an existing file w			2
constraints.	this window	pops-u	ıp.
• <u>C</u> reate a new file	give the cons in the same d		
Eile type:	XDC	~	
File name:			
Fil <u>e</u> location:	🛜 <local project="" to=""></local>	~	
O Select an existing fil	e		
<select a="" targ<="" td=""><td>et file&gt;</td><td>~</td><td></td></select>	et file>	~	
	af را	fter cli	ck OK
3	OK		Cancel

This means that since the constraints have been changed, a new constraints file will be generated - all there is to do is to give it a name - Vivado will take care of the rest.

Figure 38 shows the new constraints file name is pinouts.XDC and it is now part of the source file list.

Figure 37: Naming the new constraints file

SYNTHE SIZED DE SIGN - xc7z020clg400-1 (active)						
Sources × Kist In the sources window	Ľ					
🔍 😤 🜲 🕂 🕜 you can see the						
Constraints file created	^					
pinouts.xdc (target)						
Hierarchy Librarics Compile Order						

Figure 38: The New constraints file forming part of the source files

SYNTHESIZED DESIGN - xc7z020clg400-1 (active)	
Sources × Netlist ? _ 🗆 🖸	y × Device × lightingLED1.vhd × Schematic × pinouts.xdc
$Q \neq A$	/12_20171030/Zynq7020/LightingLED1/LightingLED1/LightingLED1.srcs/constr
Constraints (1) of the constraints file to show the syntex used	Q     →   →   X   □   □   //   □   ♀   1 set property PACKAGE_PIN Y16 [get ports LED]
pinouts.xdc (target)	2 set_property IOSTANDARD LVCMOS33 [get_ports LED]
Hierarchy Libraries Compile Order	
Source File Properties $? = \Box \boxtimes X$	
🗈 pinouts.xdc 🔶 🔶	

Figure 39: Generated Syntax for new constraints file

Figure 39 above shows the new way how to initialize the pinouts in Vivado. This is very different from Xilinx XISE code, so it is advisable to let Vivado do all the work!

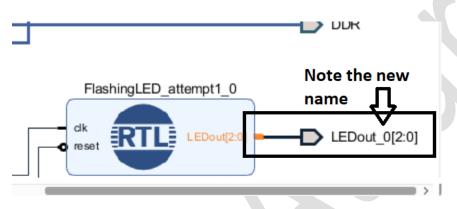


Figure 40: Naming the pins

Note in Figure 40 above the \_0 naming convention. This is due to creating a *hardware wrapper* which is necessary for the bitstream to be generated correctly. So this indicates that the hardware wrapper has been created correctly and now one can proceed to generating the bitstream file.

However, to generate the bitstream file one has to re-synthesize again. However, this time, instead of running synthesis, one can *run implementation* as shown in Figure 40.

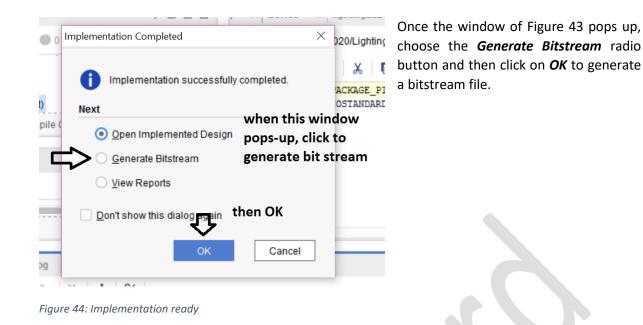
Figure 40 also shows that the old synthesis files are all out-dated and that new ones have to be regenerated due to the changes done in the constraints file.

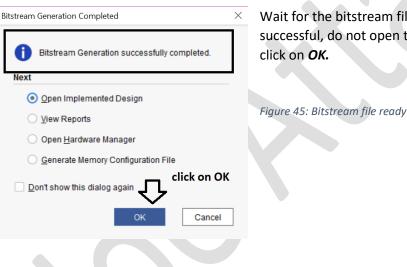
Nepoli Glock Networka						ec_prope	ICI FACIN	05_FIN 110 [	gec_
Report Clock Interaction	🗈 pi	Synthesis is Out-of	-date					$\times$	33 [
🖄 Report Methodology	Hierarchy L	Synthese	ic ic out of a	late. OK to Journe	h synthesis first? In	plomonto	ion will out	matically	1
Report DRC	Source File Prop			completes.	ar synulesis instein	ipiementa	uon win auto	Jinducally	
Scroll down to "Run the Implementation" and Click on it	pinouts.xdc	Don't show	this dialog	again	<b>↓</b> Yes	No		Cancel	
🛸 Report Power	General Pro				Tes	140		Calicer	
Schematic	Tcl Console	Messages Log	Reports	Design Run	s Find Results	×			1
	ର   <del>ମ</del>	0 C							
Run Implementation	Name Direc	tion	Interface	Neg Diff Pair	Package Pin	Fix	ed Bank	I/O Std	
> Onen Implemented Design	🛛 LED OUT				Y16	× .	34	LVCMOS33*	

Figure 41: Launching synthesis again

À Launch Runs			×	click	on <b>OK</b>	'again.			
Launch the selected	d synthesis or impleme	ntation runs.	A						
Launch <u>d</u> irectory:	🛜 <default d<="" launch="" th=""><th>irectory&gt;</th><th>~</th><th></th><th></th><th></th><th></th><th></th><th></th></default>	irectory>	~						
Options									
Launch run	is on local host. Num	ber of jobs: 4	~						
◯ <u>G</u> enerate s	cripts only								
Don't show this	s dialog again	ОК	Cancel						
Figure 42: re-laund	ching synthesis aga	in							
	sining synthesis aga								
Tcl Console	Messages L	og Reports	Design Run	s ×	Find Re	sults			
Q   ¥   4	€   ≪	▶   ≫   <b>+</b>	%						
Name	Constraints	Status		WNS	TNS	WHS	THS	TPWS	Total Po
✓ O synth_1	constrs_1	Running synth	_design		No	te			
⊠ impl_	1 constrs_1	Queued		7					
						In the	ese sna	apshots	one can
Q	)   €   ≪	▶   ≫   <b>+</b>	%				-	-	ough the
Name	Constraints	Status			WNS	-		-	nd also
✓ ✓ synth_1	constrs_1	synth_design	Complete!				-	-	entation ire done
O impl_1	1 constrs_1	Running Des	ign Initializat	ion		-			ors, the
now imp	olementatio	n is running	Ş					-	pops up.
<									

Figure 43: running through the synthesis and implementation stages





Wait for the bitstream file to be generated and once successful, do not open the Implemented Design but click on *OK*.

## Export Hardware

Once the bitstream file is generated, the next step is to export the hardware. This is done by making sure that Vivado is in the dashboard, that is, there are no files opened. Click on File, then scroll down to Export, select Export Hardware from the list. This is shown below.

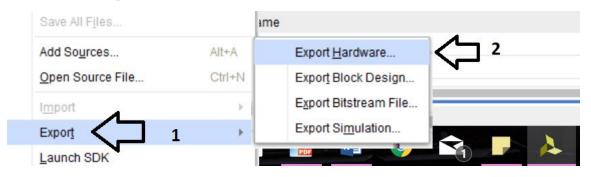


Figure 46: Export Hardware

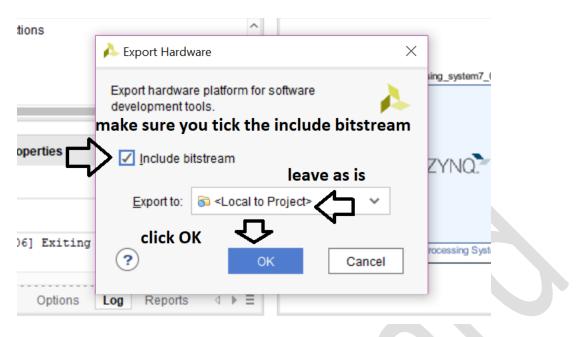


Figure 47: Include the Bitstream file

Make sure to include the bitstream file by *ticking the square box* before hitting *OK* below!

### <u>Launch SDK</u>

The next step is to *Launch SDK* from within the project itself. This is done by once again, clicking on *File*, scrolling down to *Launch SDK* and click on it. It must be mentioned here that if one is using a *13*" *laptop* or a small screen, it can happen that *Launch SDK* will *not* be immediately *visible*. If this happens, all that must be done is scroll down to the last option on the *File-List* and then use the *arrow-down key* to reveal the *Launch SDK* option.

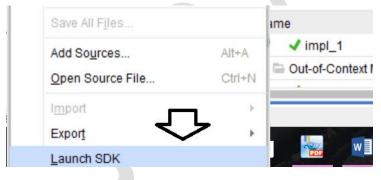


Figure 48: Launching SDK

The next window pops up, click on **OK**. This means that the folders and files created by SDK will be stored in the same directory as the Vivado project itself.

🝌 Launch SDK	×	·	
Launch software development to Leave as is	ol. 🗼		
Exported location: 🕞 <loc< td=""><td>al to Project&gt; 🗸 🗸</td><td></td><td></td></loc<>	al to Project> 🗸 🗸		
Workspace: 🔂 <local p<="" td="" to=""><td>'roject&gt; ❤</td><td></td><td></td></local>	'roject> ❤		
click on OK	Cancel		
Figure 49: Store the SDK project	within the Vivado Proje	ct	
ALL PROGRAMMABLE		to Xilinx So	oftwa
Create App Create a new Xilinx® SDI	Vication Project	$\langle \neg$	Go throu
Figure 50: SDK opens		$\mathbf{\lambda}^{-}$	~
🙀 flashingLED_attempt1.sdk - C/	'C++ - blockdesign_atter	mpt1_wrapper_hw_pla	tform_0/system.hdf - Xilinx S
File Edit Navigate Search Pro			-
<mark>☆</mark>	) 🔻 🔌 🖂 🏭 🔯 🎯	/ · · · · · · · · · · · · · · · · · · ·	
Project Explorer ⊠	▣ 🔄 🔻 ▽ ▫ נ	□ system.hdf ¤	
<ul> <li></li></ul>			_attempt1_wrapper_

blockdesign_attempt1_wiappei_n		ockdesign_att	empt1 wrar	oper
blockdesign_attempt1_wrapper	.bit 🗸 🚽	·····	hh	
Init_gpl.c		sign Information		
B ps7_init_gpl.h A project	t is automatically	get FPGA Device:	77020	
Isps7_init.c Isps7_init.c	in SDK	5	xc7z020clq400-1	
ⓑ ps7_init.h		Created With:	5	
Init.html		Created On:	Wed Feb 28 13:4	0:39 2
ps7_init.tcl				
🗈 system.hdf	Add	dress Map for pro	cessor ps7_corte	exa9_[(
	Ce	ell	Base Addr	High .
	ps	s7_intc_dist_0	0xf8f010	0xf8f
	ps	s7_gpio_0	0xe000a	0xe00

Figure 51: An SDK project is created automatically

ps7\_scutimer\_0

ne7 clor A

0xf8f006... 0xf8f0

0vf80000 0vf80

Now that SDK is opened, one must create a *First Stage Boot Loader File* in short *FSBL* file. This file is a bootloader file and once copied to the SD card, and the Zynq 7 is powered up, it will look for this bootloader file to start operating.

SDK	lashingLED_attempt1.sdk - C/C++ - blockdesi	ign_attempt1_wrap	per	_hw_platform_0/system.hdf - Xilinx	SDK
File	Edit Navigate Search Project Run Xilinx	Window Help		A	
	New 2 2	Alt+Shift+N >	٩	Application Project	
	Open File			SPM Project	
	Open Projects from File System			Board Support Package	
	Close	Ctrl+W		Project	hw_platform
	Close All	Ctrl+Shift+W	63 C3	Source Folder Folder	
	Caulo	Ctrl i S	г¢	6 F1	

Figure 52: Creating a new application

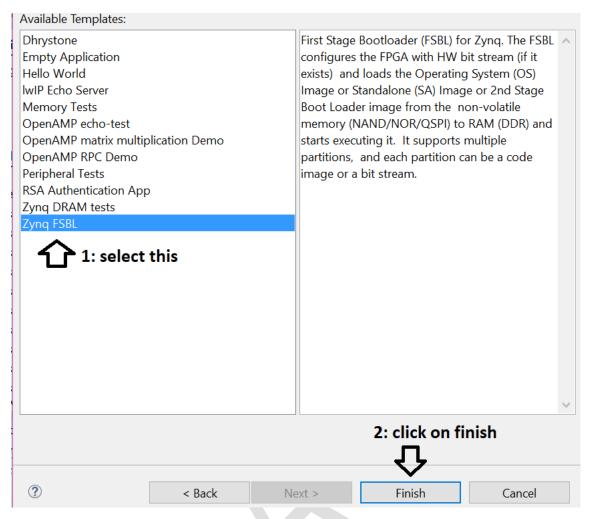
Click on *File*, then select *New*, a sub-menu appears on the right, click on *Application Project*.

Project name: Block	DesignHWwrapper_FS		e FSBL file
Use default locati	on	V name th	
Location: G:\Z-TURN	_V12_20171030\Zync	7020\flashingLEDattemp	bt1\flashingLED_i Browse
Choose file	system: default \vee		
OS Platform: standa	alone		~
Target Hardware			
Hardware Platform:	blockdesign_attempt	t1_wrapper_hw_platform	Note Vew
Processor:	ps7_cortexa9_0		That the hardware
			platform is taken 1
Target Software			the board files we included before
Language:	● C ○ C++		included before
Compiler:	32-bit	$\sim$	
Hypervisor Guest:	N/A	$\sim$	
Board Support Pack	age:	BlockDesignHWwrapp	er_FSBL_bsp
	O Use existing		$\sim$
	c	lick on next	
	-		
		~>	
?	< Back	Next >	Finish Cancel

Figure 53: Naming the FSBL project

Figure 53 shows the new window that pops up. All that must be done is just give it a name. It is recommended to include the letters FSBL in your project name so that one can distinguish it from the C project that could be generated later!

As Figure 53 suggests, click on **NEXT**.



*Figure 54: Selecting the type of the project* 

Figure 54 shows how the FSBL project type is selected. So, highlight **Zynq FSBL** first and then click on **FINISH**.

	Peripheral Drivers		
	Drivers present in the Board Support Package.		
	ps7 afi 0 generic view console		
	Overview Source		
°	Problems @ Tasks		🗏 SDK Log 🛛
	↓ ↓       ↓ ↓	•	11:23:44 INF 11:23:46 INF
	<pre>make -c ps/_cortexay_0/llosrc/xiirsa_vi_4/src -s include SHU "Running Make libs in ps7_cortexa9_0/libsrc/canps_v3_2/src"</pre>	^	11:23:47 INF
	<pre>make -C ps7_cortexa9_0/libsrc/canps_v3_2/src -s libs "SHELL=0 "Compiling canps"</pre>		11:23:47 INF 11:23:47 INF
		$\mathbf{\vee}$	
	< >>		<

Figure 55: Viewing the Console in SDK

To create the FSBL project, SDK takes a while dependent on the type of processor one has, it may take several minutes.

LEDonOff - C/C++ - LEDonOFF\_FSBL\_bsp/system.mss - Xilinx SDK File Edit Navigate Search Project Run Xilinx Window Help

3 - 🔚 🐘 🗞 - 🍕 - 🛤 🎋 - 💽 - 🔍 🖂 🏭 🗖 🖬 🦉				0.000110,000		output, profiling
				Close Unrelated Projects		entation: <u>standalone_v6</u>
Project Explorer 🛛 🕞 🕏 🔽 🖓 🖓 🖓 🖓	system.hdf			Build Configurations	>	
LEDonOFF_FSBL  right click on the	LEDonOF			build configurations		eral Drivers
BLEDonOFF_FSBL_bsp				Run As	>	present in the Board Supr
> i BSP Documentation	Modify this			Debug As	>	datamover 0 axidma
> 🍃 ps/_cortexa9_0				Compare With	>	v Source
🚡 Makefile	Target Infor			Restore from Local History		Jource
system.mss	This Board S	onnecti		Restore from Local History		ms 🛱 🧟 Tasks 📮 Console
/ Jacobian J	Hardware Sp	vare Se		C/C++ Build Settings		2 warnings, 0 others
is ps7_init_gpl.c	Targe	TCF Ag	5	Generate Linker Script		tenBoot Image
ⓑ ps7_init_gpl.h		J TcfGd		Change Referenced BSP	select crea	_
Image: Image	Operating S	) IciGai		Create Boot Image		rnings (2 items)
ps7_init.h	Board Supp			Team		
<ul> <li>w ps7_init.html</li> <li>m ps7_init.tcl</li> </ul>	Na			learn	- /	
system.hdf	Vers			Configura	``````````````````````````````````````	
System.nu	Descript					

Figure 56: FSBL project created

Figure 56 shows a newly created FSBL project, that is part of the main Vivado project. The next step is to create a boot image file. This will be copied to the SD card, then the Zynq 7 will look for it, as soon as it is powered up. To create a boot image file, one must *right-click* on the newly created FSBL project and from the list, choose *create boot image file*. This is shown in figure 56.

Create Boot Imag	e	633
Creates Zynq Boot I	mage in .bin format from given FSBL elf and partition files in specified output folder.	
Architecture: Zynq	~	
Create new BIF file	O Import from existing BIF file	
Basic Security		
Output BIF file path:	$\label{eq:constraint} G: \c true \c $	Browse
UDF data:		Browse
Split	Output format: BIN v	
Output path:	$\label{eq:constraint} G: \climeter Link \climeter$	Browse
Boot image partition:	s	
	Encrypt.	
Xynq7020\flashingL	EDattempt1\flashingLED_attempt1\flashingLED_attempt1.sdk\BlockDesignHWwrapper_FSBL\Debug\BlockDesignHWwrapper_FSBLelf none	Add
	flashingLED_attempt1\flashingLED_attempt1.sdk\blockdesign_attempt1_wrapper_hw_platform_0\blockdesign_attempt1_wrapper.bit none	Delete
	.bit file and the FSBL files are automatically included in the bootloader	
file being	created	Edit
-		

Figure 57: Create Boot Image Window



A new window pops up. In the lower part of this window, one can find the *FSBL.elf* file and the *.bit* file created earlier in Vivado. So, the Zynq 7 first reads the bootloader file and then reads the *.bit* file to create the hardware in the programmable logic part. Click on Create Image button and SDK will generate an

image file that has to be copied to the SD card for booting the Zynq 7.

Peripheral Drivers		
Drivers present in the Board	Support Package.	
ps7 afi 0 <mark>gene</mark> r	ric	
<	The console shows the	
Overview Source	boot image file is created	
🕄 Problems 🧔 Tasks 토 Consc	ble 🛱 🗆 Properties 🖳 SDK Terminal	
	-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0	] 🛨 📑 🛨 💡
CDT Build Console [BlockDesig		
make: Notning to be ad	one tor main-build .	<b>^</b> :
12:05:54 Build Finishe	ad (took 317ms)	1
12.05.54 build fillishe		:
		$\checkmark$
Figure 58: Console shows the ima	ge file ready	

# Copying the image file to SD card

SOK File nath

After the image file is created, the next step is to copy it on SD card. This is done just using the normal copy/paste combination of commands in Windows. However, one has to find the right image file and the following snap shots shows the steps.

	Z-TU	RN_V12_20171030 > Zynq7020 > LEDonOff	5 ~	Search LEDonC
Drganize 🔹 New f	older	this is my project		
Ъ 3D Objects	^	Name	Date modified	Туре
늘 Desktop		I.metadata	20/02/2018 22:42	File folder
Documents		LEDonOFF	20/02/2018 22:23	File folder
湨 Downloads		EDonOFF_FSBL	20/02/2018 22:47	File folder
🌗 Music		📙 LEDonOFF_FSBL_bsp	20/02/2018 22:46	File folder
📄 Pictures		📜 RemoteSystemsTempFiles	20/02/2018 22:42	File folder
P Videos		📙 Z-turn7020_hw_platform	20/02/2018 22:46	File folder
		SDK	20/02/2018 22:42	Text Document



Figure 59 shows the project location in the hard disk.



Figure 60: Location of the image file

Figure 60 shows the location of the boot image file that must be copied to SD card using the copy/paste commands in Windows. Figure 60 also shows that only the **BOOT** file has to be copied only!

Insert an SD card in an SD card to USB adapter. Insert the adapter in the USB port in the laptop or PC. Windows should detect the SD card and opens a new window showing the contents of the SD card. Use the copy/paste commands in Windows to copy the BOOT image file.

After copying the file, make sure to disable or disengage the SD card from Windows not to corrupt it. Insert it in the slot of the Z-turn board, power up the board and the LEDs should light up. Please note that as the board schematics dictate, the LEDs need a logic 0 on the pins for them to light up. Enjoy!

# Steps to create an A9 Soft core project using both Vivado and SDK

## Introduction

In this chapter, the ARM Cortex A9 will be used to control the flashing of LEDs. This time, instead of writing VHDL code to create hardware, *C* instructions will be written in a typical *C* project. So, in this project, the author will show how to create a *C* project from SDK and which functions should be used to light the LEDs located at MIO 0 and MIO 9. Once again, the whole process how to create a project will be shown so that one will become more familiar with the steps needed to create a project in Vivado.

### Starting a new project in Vivado

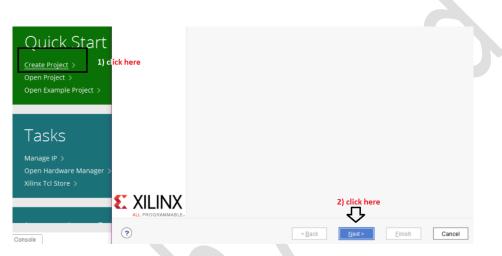


Figure 2. 1 : Starting a Project in Vivado

Figure 2.1 speaks for itself, click on *Create Project* 

	Project Name Enter a name for		there the project data files will be stored.	
HLx Editions			1) give the project a name	
	<u>P</u> roject name:	ZynqPS_UART_flashingLED	⊗	
	Project location	n: G:/Z-TURN_V12_20171030/Zynq	7 <sub>020</sub> 2) here you can create a subfolder 🛛 🔊 🗔	
Quick Start	🗸 Create proj	ect subdirectory	however this is unnecessary since Vivado will	
<u>Create Project</u> >	Project will be o	created at: G:/Z-TURN_V12_201710	30/Zynq7020/ZynqPS_UART_flashingLED create a folder for you automatically	
Open Project >				
Open Example Project >			3) click on NEXT	

Figure 2. 2: Name the Project

Give a name to the project and make sure that it will be stored in the desired folder. Click on **NEXT** below.

🝌 New Project

	ect Type	
۲	<u>R</u> TL Project You will be able to add sources, create block designs in IP Integrator, generate IF implementation, design planning and analysis.	P, run RTL analysis, synthesis,
0	Post-synthesis Project: You will be able to add sources, view device resources, re implementation.	un design analysis, planning and leave this page as it is and click on NEXT below
0	I/O Planning Project Do not specify design sources. You will be able to view part/package resources Figure 2. 3: Type of Project Page	JV

Figure 2.3 shows the type of project one can use, since this project will be based on the hard core A9 processor or the *Processing System* part of the Zynq 7, this page will not affect the project, so it will be left untouched. Click *NEXT*.

ew Project	
	ist, Block Design, and IP files, or directories containing those files, to add to your project. Create a new k and add it to your project. You can also add and create sources later.
+, -	±   ∓
	at this point you do not need to add any source files for now
	Use Add Files, Add Directories or Create File buttons below
	Add Files Add Directories Create File

Figure 2. 4: Adding Source Files

Figure 2.4 asks to add source files. Again, since this project will focus on the Processing System part of the Zynq 7, no source files need to be created at this point. Click on **NEXT**.

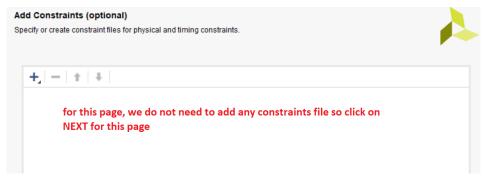


Figure 2. 5: Adding Constraints File

The next page asks for the *constraints file* or how the pins will be connected. This is *unnecessary* because since the *Board Support Files* have been included in Vivado folder as described in chapter 1. So, click on *NEXT*.

Filter/ Preview				
Ve <u>n</u> dor:	All	~		
Display <u>N</u> ame:	All	~		
Board Re <u>v</u> :	Latest	~		
	Reset All Filters			
earch: Q- Y	ou need to install the	XML files for your b	oard	
carcii.	rst for this to be avail	able. I have covered	l this	
ti	st for this to be available			
	another document	Vendor	Board Rev	Part
)isplay Name in		Vendor alpha-dal		Part       xcku060-ffva1156-2-e
)isplay Name in Kintex-Ultrascal	another document	alpha-dai	ta.com 1.0	
Display Name in	another document e Alphadata board Evaluation and Development k	alpha-dai	ta.com 1.0	xcku060-ffva1156-2-e

Figure 2. 6: Selecting the Z-turn Board

Figure 6 shows how to select the Z-turn board from the list. This will make sure that Vivado and SDK will configure the environments to comply with the Z-turn board peripherals and characteristics.

Just to refresh one's memory, figure 2.7 and 2.8 below, show the XML files downloaded from Github and where these files should be stored within Vivado directory.

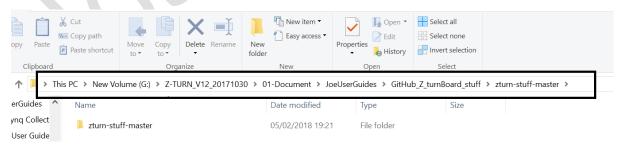


Figure 2. 7: Github folder

Then after extracting the files, copy them to:

e sna	ire view					
	🔏 Cut			New item 🕶	📑 🛛 📭 Open 👻	Select a
y Paste	No. Copy path	Move Copy	Delete Rename	New Easy access -	Properties Edit	Select n
y Paste	Paste shortcut	Move Copy to • to •	- Delete Rename	folder	+ History	💾 Invert se
Clipboard	d	Org	anize	New	Open	Selec
▶   >	This PC > Local Dis	sk (C:) 👂 Xilinx	> Vivado > 20	17.4 » data » boards 🔅	→ board_files →	
Guides	^ Name			Date modified	Туре	5
q Collect	ac701			02/02/2018 16:4	I2 File folder	
ser Guide	📜 adm-pcie	3-ku3		02/02/2018 16:4	File folder	
- Malta (	📜 adm-pcie	-7v3		02/02/2018 16:4	I2 File folder	
	📙 kc705			02/02/2018 16:4	File folder	
- Person	📜 kcu105			02/02/2018 16:4	18 File folder	
tion Elect	📜 zc702			02/02/2018 16:4	13 File folder	
	📜 zc706			02/02/2018 16:4	18 File folder	
	📜 zcu102			02/02/2018 16:5		
≥cts	zed			02/02/2018 164	13 File folder	-
	zturn-7z0	20		05/02/2018 19:5	53 File folder	
o ents						
		Figu	re 2. 8: Z-turn fo	older within Vivado	22	
	VIVADO.	New Projec	t Summary			
	HLx Editions	A new RTL	project named 'UART_	LED' will be created.		
		A No cource	files or directories will I	be added. Use Add Sources to add t	hom later	
		- NO SOULCE	mes of directories will t	le added. Ose Add Sources to add i	nem later.	
		No constra	aints files will be added.	Use Add Sources to add them later	:	
		Default Bo	ng-7000 clg400			
			roject, click Finish		₽	
	•			< Back	Einish Cancel	
		Fi	nure 2 9. Projec	t Summary Page		

Figure 2. 9: Project Summary Page

Figure 2.9 shows the project summary page. Click on FINISH to open the project in Vivado.

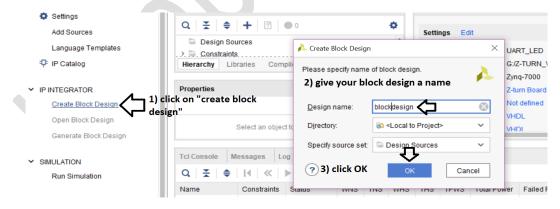


Figure 2. 10: Creating a new Block Design

Once the project environment is opened in Vivado, one can immediately create a new Block Design. This will open a new schematic window where one can connect all the components in the design. Follow the steps in Figure 2.10 above to create a new schematic.

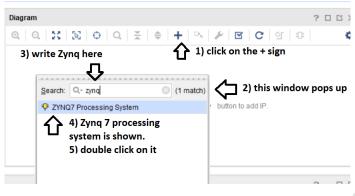


Figure 2. 11: Schematics Window

Figure 2.11 shows an opened schematics window. It also shows the steps to create a new *A9 hardcore processor* always referred to as the *Processing System*.

Diagram × Addre	ess Editor X	
Q Q X X	$  \odot   Q   \stackrel{\times}{=}   \Leftrightarrow   H   \stackrel{\circ}{=}   A  $	🖌 🛛 🖬 🗶 🖉 🖉 🖉 🖉
🗯 Designer Assistan	ce available. Run Block Automation	click here
ne A9 soft core	processing_system7	_0
ppears on the anvas	M_AXI_GP0_ACLK ZYNQ	DDR +    FIXED_IO +    M_AXI_GPO +    FCLK_CLKO - FCLK_RESETO_N •
	ZYNQ7 Processing Sys	stem

Figure 2. 12: Block Automation

After the *Processing System* part is created, one must click on *Run Block Automation* so that all the hardware peripherals of the A9 core will be enabled. This is shown in Figure 2.13. Make sure that *Apply Board Preset* will be ticked!

	Q ₹ ♦	Description		
	✓ ✓ All Automation (1 out of 1 selected) ✓ ♥ processing_system7_0	This option sets the board preset on the Processing System. All current properties will be overwritten by the board preset. This action cannot be undone. 2mg7 block automation applies current board preset and generates external connections for FRED_IO, Trigger and DDR interfaces. NOTE: Apply Board Preset will discard existing IP configuration - please uncheck this box, if you wish to relain previous configuration. Instance: (processing_system7_0 leave this window as is Options		
		Make Interface External: FIXED_IO, DDR Apply Board Preset @ross Triggerin: Disable Cross Trigger Out Disable Cross Trigger Out Disable Di		

Figure 2. 13: Apply Board Presets

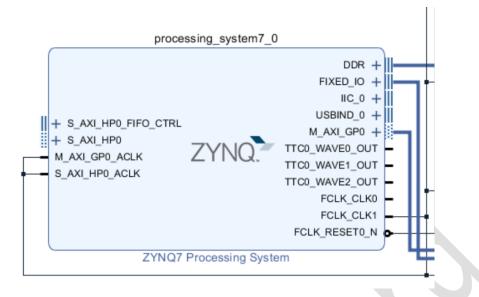


Figure 2. 14: Connecting the AXI clocks

Make sure to connect the AXI clocks as shown in Figure 2.14 to avoid errors.

Now it is time to create a *Hardware Wrapper*. This serves as a *top-level* block to the sub-systems in the system.

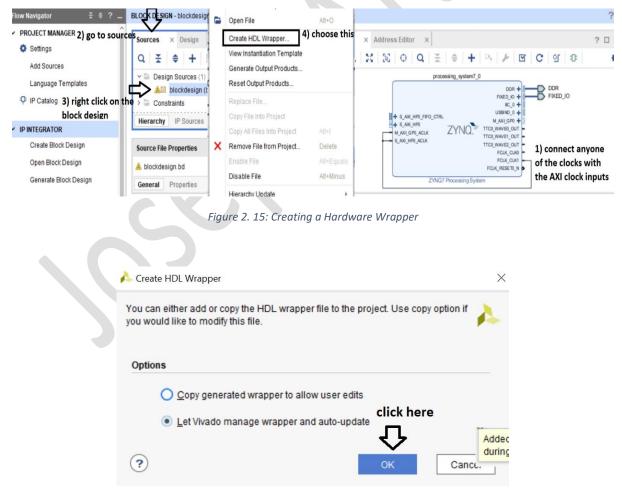


Figure 2. 16: Let Vivado create all the settings

Referring to Figure 2.13 - after double clicking the block diagram of the A9 core, one must be aware of how Vivado shows which peripherals can be used in software. Apart from others, there are two UARTs. At least UART 1 must be enabled, so that one will be able to use the basic C project template called "Hello World" within SDK. Using this C project template, one will solve all the linker problems between the software part and the hardware part. It is recommended to use this template and not the blank C project template! If the UART peripherals are removed from the Zynq 7 Processing System, then this C project could not be used in SDK.

If peripherals that are not used in the processing system part are disabled, the firmware might not work, so it is recommended that when using the Board Support Files, one will NOT disable any of the peripherals already enabled in the preset version of the Zynq Processing System!

Since in this project, the focus is on how to enable the Processing System of the Zynq 7 and also how to use the built-in functions in SDK to enable certain peripherals, at this point, no VHDL module will be created, so after the HDL wrapper is created, one can generate the bit-stream file straight away!

### After creating the HDL wrapper, one can generate the bit stream.

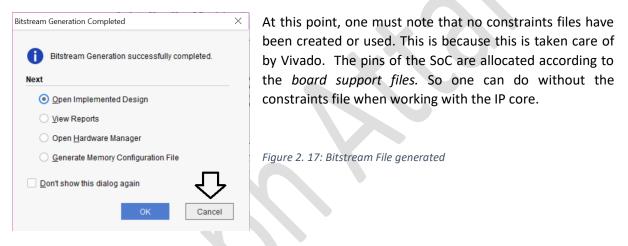


Figure 2.18 shows the next step after the bitstream file has been created. The hardware file must be exported and this is done from the *File* menu. *File*  $\rightarrow$  *Export*  $\rightarrow$  *Export* Hardware

<b>`</b>	<u>S</u> ave File	Ctrl+S	✓ synth_1 (active)
	Sa <u>v</u> e File As		✓ impl_1
	Save All Files		Out-of-Context Module Runs
J	Add So <u>u</u> rces	Alt+A	> 🗸 blockdesign
·	Open Source File	Ctrl+N	Export <u>H</u> ardware
	I <u>m</u> port	Þ	Export Bitstream File
	Export	×.	Export Simulation
Ex.	Launch SDK		

Figure 2. 18: Exporting the Bitstream File

À Export Hardv	ware	×
development to	re platform for software ools. I <b>need to tick this</b>	box 🗼
🗹 Include b	itstream	
<u>E</u> xport to:	🗟 <local project="" to=""></local>	~
?	ОК	Cancel

Figure 2. 19: Include the Bitstream File

Make sure to tick the square box as shown in Figure 2.19.

Then one can open SDK from Vivado. This will create a subfolder within the hardware-project-folder where all the SDK files will be saved.



WARI\_LED.sdk - C/C++ - blockdesign\_wrapper\_hw\_plattorm\_0/system.hdt - Xilinx SDK

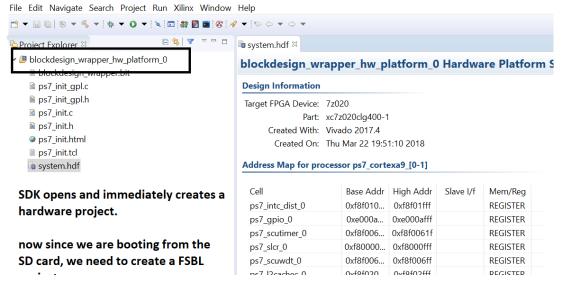


Figure 2. 21: Opening SDK

Figure 2.21 shows the SDK linked to the Vivado project. The next step is to create a *First Stage Boot Loader (FSBL)* project that will link all the C and VHDL project together. Figure 2.22 shows the steps to create an FSBL project which will be used by the Zynq 7 Processing System to load both the hardware and software of the system.

	ar c/c·· biocracsign_ma	pperpider.eo/	Sterman Anna Ser
ile Edit Nav	igate Search Project Run X 1)	Alt+Shift+N >	Application Project 2)
Open File.	-/		<ul> <li>SPM Project</li> <li>Board Support Package</li> <li>Project</li> </ul>
Close Close All		Ctrl+W Ctrl+Shift+W	<ul><li>is Source Folder</li><li>is Folder</li></ul>
Save		Ctrl+S	🖻 Source File
Choose file syst OS Platform: standalon Target Hardware Hardware Platform: blc	a) give a name to the FSB   2_20171030\Zynq7020\UART_LED\UART_LED.sc   2_20171030\Zynq7020\UART_LED\UART_LED.sc   term:   default ~   bckdesign_wrapper_hw_platform_0 7_cortexa9_0 it's a C project it's a C project C \C++ 32-bit the name is here	dk\UART_LE Browse V New V	oject Figure 2. 23: Naming the FSBL project Figure 2.23 show what one must fill to create a new FSBL project.
?	< Back Next > Fi	nish Cancel	

Empty Application Hello World IwiP Echo Server Memory Tests OpenAMP echo-test OpenAMP RPC Demo Peripheral Tests RSA Authentication Ap Zyng DRAM tests Zyng PSBL 1) choose this		configures the FPGA with HW I exists) and loads the Operating Image or Standalone (SA) Imag Boot Loader image from the in memory (NAND/NOR/QSPI) to starts executing it. It supports partitions, and each partition of image or a bit stream.	g System (OS) ge or 2nd Stage on-volatile PRAM (DDR) and multiple			
		click FINISH				
		心				
?	< Back N	ext > Finish	Cancel			
	•	iaione lalone is a simple, low-lev exceptions as well as the b exit.		•	•	
-	Peripheral Drivers Drivers present in the I	Roard Support Package		ws if there are a hat the software	-	
C	overview Source	V thro	ugh			
	Problems 🧟 Tasks 📮	Console 🛛 🗖 Properties			SDK Log ¤	
	ake -C ps7_corte: Running Make lib:	[LED_FSBL_bsp] Luge in ps/_cortexa ka9_0/libsrc/xilrsa s in ps7_cortexa9_0 ka9_0/libsrc/canps_	a_0/1105rc/x a_v1_4/src -s 0/libsrc/canp	include "SHE s v3 2/src"	22:00:34 INFO 22:00:37 INFO 22:00:39 INFO 22:00:39 INFO 22:00:39 INFO	: Launching : XSCT serv : Successfu : Processfu : Successfu

Figure 2. 25: SDK console

Figure 2.25 show the SDK console running. This is a useful part of SDK and therefore one must make sure to listen to it!

Now after the FSBL project has been successfully created, one must create a C project. This project consists of many C source files and header files, together with the functions needed by the application in quation!

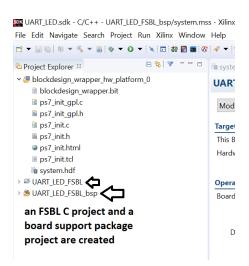


Figure 2. 26: FSBL project created

	ake application project.	create a new application project again	n
Project name: UART_I	LED_C_program		
✓ Use default location	n 🏠 give it	a name	
Location: G:\Z-TURN_	V12_20171030\Zynq702	20\UART_LED\UART_LED.sdk\UART_LE	Browse
Choose file s	system: default 🖂		
OS Platform: standa	lone		~
Target Hardware			
Hardware Platform:	blockdesign_wrapper_hw	v_platform_0	~ New
Processor:	ps7_cortexa9_0		$\times$
Target Software			
Language:	● C ○ C++		
	32-bit	this is mirrored fro	m
Hypervisor Guest:	N/A	above	
Board Support Packa	ge:      Create New U	ART_LED_C_program_bsp	
	◯ Use existing UA	ART_LED_FSBL_bsp	

#### Templates

Create one of the available templates to generate a fully-functioning application project.

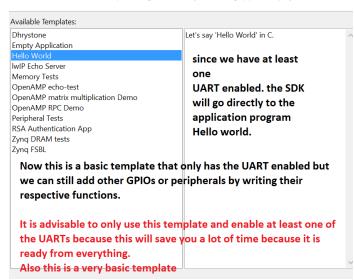


Figure 2. 27: Creating a C project

-

To create a new C application, one must file click on FILE -> new -> new application -> the window of Figure 2.27 pops up. Give the C project a name and leave the rest is it is.

As already stated previously, it is advisable to use this template as the base of the C project because Vivado will not generate any unnecessary errors due to incorrectly created C project.

Figure 2. 28: Creating a C project

Only the **Hello World** template is good to build a C application. *It is advisable to make sure that one of the UARTs is enabled as otherwise this application <u>will not be available to the user</u>.* 

The above happened to the author when he tried to open the *hello world* template when in Vivado the author disabled all the peripherals and only the *blank-application-project* was available!!

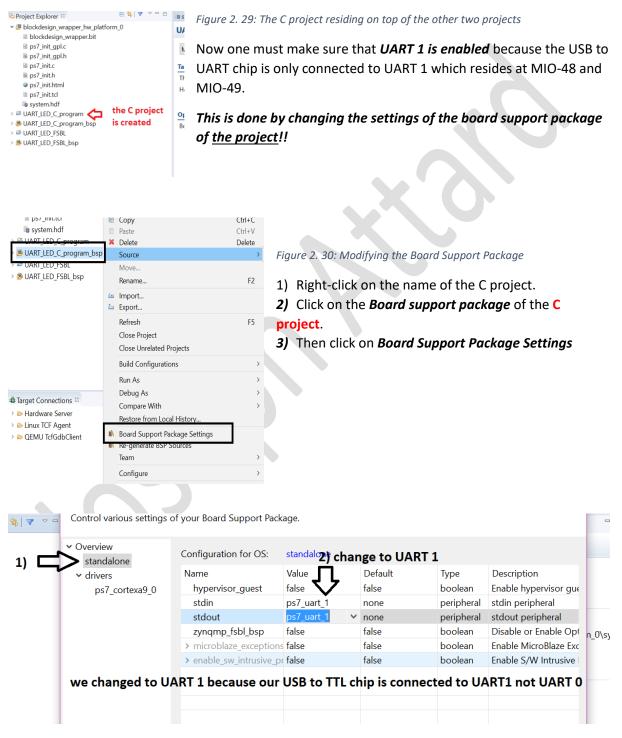


Figure 2. 31: Changing the Board Support Packages to change UART 1 settings

Documentation: <u>standalone_v6_5</u>				
Peripheral Drivers	once any of the bsp files i	s changed, the SD	)K will re-	
Drivers present in the Board Support Package.	generate all the software	•	~	
	generate an the software	5	>	
Overview Source				
😨 Problems 🔊 Tasks 🖻 Console 🕸 🗆 Properties	🗉 SDK Terminal 📃 🗖	I SDK Log ¤		
CDT Build Console [UART_LED_C_program_bsp] kunning make include in ps/_cortexa: make -C ps7_cortexa9_0/libsrc/usbps_ "Running Make include in ps7_cortexa make -C ps7_cortexa9_0/libsrc/xadcps	v2_4/src -s include "SHEL ^ 9_0/libsrc/xadcps_v2_2/src	22:00:39 INFO 22:00:39 INFO 22:33:40 INFO	: Processing co : Successfully : Refreshed bui	done setting XSCT servormmand line option -hws done setting SDK works ld settings on project MSS file content so r

Figure 2. 32: Console in SDK shows all the transformation being done

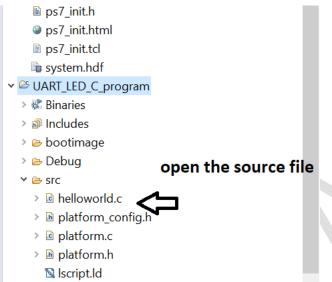


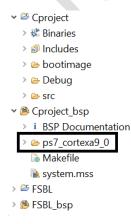
Figure 2.33 show the location of the *Hello World*, C program. Now it is time to open it and write some code to flash an LED and send data on the serial port.

Vivado also has an extra feature that once you save the project it will start re-building on its own.

Figure 2. 33: Location of the C project in SDK

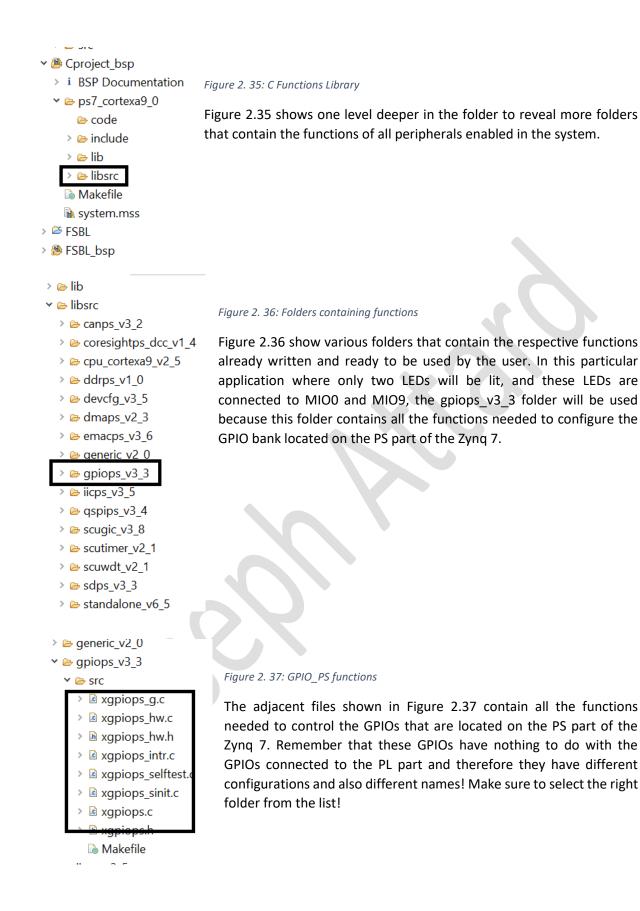
## The Software Part in SDK

So far, the Vivado project was created and linked to the SDK from where C instruction could be written to control in this application the two LEDs connected with pins MIO0 and MIO9 of the Processing System part of the Zynq 7. Double-clicking on helloworld.c file will open it on SDK to be edited as follows:



#### Figure 2. 34: Location of all C functions

Figure 2.34 show the folder where all the C functions available to the user reside. Click on the arrow pointing to this folder to reveal more folders.



## Configuring the Processing System (PS) GPIOs

The following routine should be used to configure nearly all the peripherals from the Processing System side of the Zynq 7. One must include the following include directive for the functions to be eligible:

## #include "xgpiops.h"

First look for the lookupConfig():

```
XGpioPs_Config *XGpioPs_LookupConfig(u16 DeviceId)
```

The above function is located in *xgpiops\_sinit.c* file.

The above function *returns* a pointer of type *XGpioPS\_Config* while is *passed a parameter* of type *u16*.

The DeviceID parameter can be replaced with *XPAR\_PS7\_GPI0\_0\_DEVICE\_ID*. This is located in *xgpiops\_g.c* file

Now **cut** the XGpioPs\_Config from the name of the function above and **paste** it as one of the initial data types at the beginning of the main (). Assign a name to this pointer as shown below:

## XGpioPs\_Config \*ConfigPtr;

Then equate the above function to the name given to the pointer. The final result is shown below:

## ConfigPtr = XGpioPs\_LookupConfig(XPAR\_PS7\_GPIO\_0\_DEVICE\_ID);

Now its time to initialize the GPIO\_PS peripheral. Use the following function:

## s32 XGpioPs\_CfgInitialize(XGpioPs \*InstancePtr, XGpioPs\_Config \*ConfigPtr,u32 EffectiveAddr)

which is located in *xgpiops.c* file. The above function returns a type of *signed 32 (s32)* and is passed various parameters.

So **cut** the **s32** and **paste** it as part of the data type list at the top of the main function. Assign a name to a variable of type **s32**.

## s32 Status;

Cut and paste as one of the data types *XGpioPs* and give it a name as well at the beginning of the main ().

## XGpioPS Gpio;

For \*ConfigPtr use ConfigPtr as before so the above function will now look like this:

## Status = XGpioPs\_CfgInitialize(&Gpio, ConfigPtr, ConfigPtr->BaseAddr);

The *BaseAddr* was taken from *XGpioPs\_hw.h* file.

Now if one is to equate a function's return variable to a variable just like what has been done with *XGpioPs\_CfgInitialize()*, it is good practice to *check its validity* by writing the following *if-statement*. If this if-statement is not included in the code, a warning is generated in the output file.

#### Setting the Direction of the Pin and Enabling the Output

Now for each pin in the GPIO, one has to *set* its direction -whether the pin is going to act as an **input** or an **output**. Also, if the pin is going to act as an output, one has to **enable** that output! The following functions are used:

void XGpioPs\_SetDirectionPin(XGpioPs \*InstancePtr, u32 Pin, u32 Direction)

void XGpioPs\_SetOutputEnablePin(XGpioPs \*InstancePtr, u32 Pin, u32 OpEnable)

The above two functions are located in *XGpioPs.c* file. <u>Before each function, there is a description of</u> what the function should do and sometimes there are also hints on the parameters. So, make sure to read the comments before every function to have a better understanding of its effects and also what type of parameters should be passed to it!

The first function above:

void XGpioPs\_SetDirectionPin(XGpioPs \*InstancePtr, u32 Pin, u32 Direction)

returns a void and therefore expect nothing from it,

the first parameter that should be passed is the name of the instance – in this case it is &Gpio

**u32 Pin** is the pin number the function will be affecting, since in this particular example, two LEDs connected to MIO0 and MIO9 are going to be used, then this function should be written for two pins – pin 0 and pin 9.

**u32 Direction:** for this parameter, the function accepts either 0 if the pin is going to act as an input, or 1, if the pin is going to act as an output

now for the second function:

**void XGpioPs\_SetOutputEnablePin**(XGpioPs \*InstancePtr, u32 Pin, u32 OpEnable)

again, it returns a void so nothing should be expected from it

XGpioPs \*InstancePtr should be replaced once again with &Gpio

*u32 Pin* should be replaced with the *pin number* – for this example Pin should be replaced with either **0 or 9**, while *u32 OpEnable* should be replaced with **1** if the outputs are **enabled** and **0** if the outputs are **disabled**!

#### Writing to the individual Pin

The last function that needs explanation is the following:

void XGpioPs\_WritePin(XGpioPs \*InstancePtr, u32 Pin, u32 Data)

The above function writes to the individual pins.

Once again it returns a void and therefore expect nothing once the function is ready.

#### Replace XGpioPs \*InstancePtr with &Gpio

Replace u32 Pin with the pin number - in this case it has to be 0 or 9

Replace **u32 Data** with either **1** for logic high or **0** for logic 0.

The LEDs connected to MIO 0 and MIO 9 on the Z-turn board are connected such that a logic 0 will switch them on while a logic 1 will switch them off!

So the main() will look like this:

```
int main()
{
      int Status;
             XGpioPs_Config *ConfigPtr;
             XGpioPs Gpio; /* The driver instance for GPIO Device. */
    init platform();
    /* * Initialize the GPIO driver. */
             ConfigPtr = XGpioPs_LookupConfig(XPAR_PS7_GPI0_0_DEVICE_ID);
             Status = XGpioPs_CfgInitialize(&Gpio, ConfigPtr,
                                        ConfigPtr->BaseAddr);
             if (Status != XST_SUCCESS)
                    return XST_FAILURE;
             // LED1 gpio setting
             XGpioPs SetDirectionPin(&Gpio, 0, 1);
             XGpioPs_SetDirectionPin(&Gpio, 9, 1);
             XGpioPs_SetOutputEnablePin(&Gpio, 0, 1);
             XGpioPs_SetOutputEnablePin(&Gpio, 9, 1);
        while (1) {
             XGpioPs_WritePin(&Gpio, 0, 0);
             XGpioPs_WritePin(&Gpio, 9, 0);
             delay();
             XGpioPs_WritePin(&Gpio, 0, 1);
             XGpioPs_WritePin(&Gpio, 9, 1);
             delay();
      cleanup platform();
      return 0;
}
```

Now since the A9 core will execute the above code very rapidly, a delay function was introduced so that one can see the LEDs blinking. A typical delay function can be written as shown below:

Another way to flash the LEDs on MIO 0 and MIO 9 simultaneously is by using the following functions instead of the ones used in the main code previously.

#### void XGpioPs\_Write(XGpioPs \*InstancePtr, u8 Bank, u32 Data)

the only parameter that is different this time is the **u8 Bank** which has to be replaced with **0** as a number, **u32 Data** can be replaced with either a *decimal number* which is <u>not recommended</u> for this instance or better with a hexadecimal number. Thus, to write a logic 1 in both MIO 9 and MIO 0 simultaneously using the above function, one must convert it to:

#### void XGpioPs\_Write(&Gpio, 0, 0x00000201);

The same procedure could be written for the direction function

void XGpioPs\_SetDirection(XGpioPs \*InstancePtr, u8 Bank, u32 Direction)

could be replaced with:

#### XGpioPs\_SetDirection(Gpio, 0, 0x00000201);

And

void XGpioPs\_SetOutputEnable(XGpioPs \*InstancePtr, u8 Bank, u32 OpEnable)

could be replaced with:

## XGpioPs\_SetOutputEnable(&Gpio, 0, 0x00000201);

Again Gpio is according to how the XGpioPs \*InstancePtr was equated at the beginning of the main()

XGpioPs Gpio;

#### Creating the Boot Image File

This time to create a boot-image file, one has to right-click on the C project and not on the FSBL project! This is shown in figure 2.38 below:

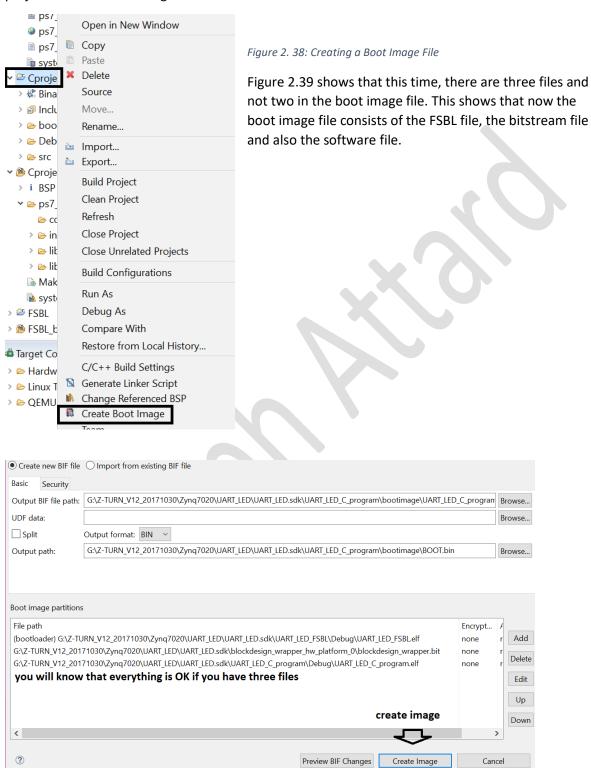


Figure 2. 39: Check files in Boot Image File

2	rasic shortcut to ▼ to ▼ ▼	folder	- Kenter History		
liploard	Organize	New	Open	Select	
📜 🕨 This	PC > New Volume (G:) > Z-TURN_V12_2017103	30 > Zynq7020 > UART_LED	> UART_LED.sdk 3	<ul> <li>UART_LED_C_program</li> </ul>	m 🕨 bootimage
ides 🐴	Name	Date modified	Туре	Size	
ollect	o воот	22/03/2018 22:40	PowerISO File	4,085 KE	3
Guide	O UART_LED_C_program	22/03/2018 22:40	PowerISO File	1 KE	3
/alta (	notice the root				
erson	copy the boot image and paste i	n the SD card			
Elect					

Figure 2. 40: Location of the Boot Image File

After the boot image file is created, it can be located in the C project and not the in the FSBL project, so make sure to select the right boot image file!

Copy the boot file to the SD card and insert it in the SD slot on the Z-turn board. Enjoy!

# Flashing LEDs simultaneously but independently from the Processing System and Programmable Logic Fabric

#### Introduction

In this project, the versatility, flexibility and parallelism that could be achieved using the Zynq 7 System-on-Chip will be discussed. A VHDL entity that flash an LED will be created. After that, a software program will be written in *C* that also flashes an LED connected to the one of the MIO pins will be created. The VHDL entity will work *independently* from the program that will be flashed in the Zynq Processing System. So, in this project, there will be two <u>independent hardware</u> working in parallel!

The process how to create a project in Vivado and how to create a VHDL file have already been explained in chapters 1 and 2 so those will not be covered anymore. This chapter will go through the steps to achieve the goal of using both the Processing System and Programmable Logic independently for the first time.

## Defining the Port terminals of the VHDL entity

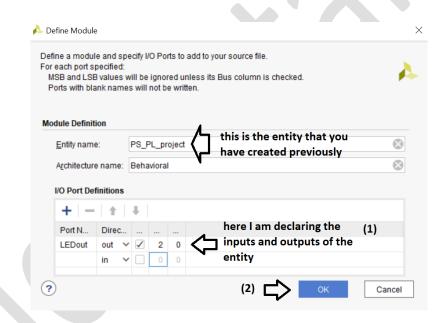
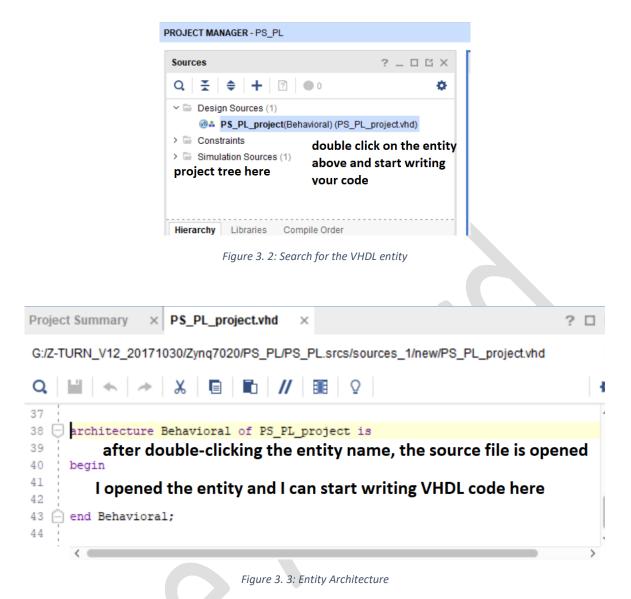


Figure 3. 1: Creating terminals for the VHDL entity

Figure 1 show the procedure to write the port terminals of the VHDL entity that has been created while setting up the project environment. Notice how a bus could be created!

Once Vivado has opened, look in the sources tab and double click on the name of the VHDL entity to open it up to start writing code. This is shown in Figure 2 on the next page.



The way of thinking when writing code in VHDL is a bit different when compared to writing code for a microprocessor. This is because even though VHDL contains sequential statements, one has to keep in mind that when writing code in VHDL for an FPGA, the code will be translated into hardware as opposed to writing code that will be flashed in memory of a microcontroller and then executed one instruction at a time. This will be explained in the following paragraphs.

First of all the primary system clock on the Z-turn board is 100 MHz. This is very convenient because its period is 10 ns and therefore it can track even relatively high frequency signals. So, let's say one would like to flash an LED with 100 milli-second frequency (100 ms). Let's also assume that the duty cycle is 50% so that the time the LED will be switched on will be the equal to the time the LED will be switched off. Do the following:

$$\frac{100 \ ms}{10 \ ns} = 10,000,000$$

10,000,000 is the number the 100 MHz clock has to count so that 100 ms pass.

So, assuming 50% duty cycle, create a variable within an process that counts up with every rising edge of the primary clock. From 0 up to 5,000,000, the LED will be swtiched off while from 5,000,000 up to

10,000,000, the LED will be switched on. When 10,000,000 is reached the counter is reset to zero and the process will start all over.

The VHDL code for the above explanation is shown in code snippet 3.1 Note that the <u>reset switch</u> is <u>acitve low</u>, this has to be according to how the reset switch is connected <u>in hardware</u>, otherwise the counter will remain reset all the time and the LED will never light up, so be careful!

```
43 - process(clk, reset)
44
    variable count: integer;
45 ¦ begin
46 - if reset = '0' then count := 0;LEDs(2) <= '0';
47 ; elsif clk'event and clk = '1' then count := count + 1;
48 E
         if count < 5000000 then LEDs(2) <= '0';
49
         elsif count >= 5000000 and count < 10000000 then LEDs(2) <= '1';
50 -
         end if;
51
         if count >= 10000000 then count := 0; end if;
52 - end if;
53 📄 end process;
54
55 : LEDs(1 downto 0) <= "11"; --for now these two LEDs are switched off
E.C
```

```
Code Snippet 3. 1: Flashing LED in VHDL
```

Save the source code and Synthesize the code.

À Launch Runs			×
Launch the selected synthesis or in	nplementation runs		4
Launch <u>d</u> irectory: 💦 <default la<="" td=""><td>unch Directory&gt;</td><td></td><td>~</td></default>	unch Directory>		~
Options			
• Launch runs on local host	Number of jobs:	4	~
O Generate scripts only			
	click	OK	
Don't show this dialog again	र	ֈ	
	0	к	Cancel



Now, create a block design to include the VHDL entity together with the Zynq processing system.

Language remplates	1			
₽ IP Catalog	Q ≍ ≑ + 2 ●	0	♦ ← →	• • •
IP INTEGRATOR	✓ Design Sources (1) → PS_PL_project(Behavi	🍌 Create Block Design		×
Create Block Design	) > Constraints	Please specify name	of block design.	
Open Block Design	Hierarchy Libraries Compile			
Conorata Block Design		r	name the block des	ign
Generate Block Design	Source File Properties × Cloc	<u>D</u> esign name:	blockdesign (2)	$\otimes$
<ul> <li>SIMULATION</li> </ul>	PS_PL_project.vhd	Directory:	🛜 <local project="" to=""></local>	~
Run Simulation	General Properties	Specify source set:	Design Sources	~
<ul> <li>RTL ANALYSIS</li> <li>Open Elaborated Design</li> </ul>	Tcl Console Messages Log	?	ОК	Cancel
	Q 풒 ≑ 백 + 뇌			



After the block design file is created, in the empty canvas, right click on the canvas and choose Add Module from the list. This will access the VHDL entity that has been created previously and will be made available in block-form to be added in the schematic. Figure 3.6 show the procedure:

Sources Design × Signals Board ? _ I   Q I I I I   A blockdesign   (1) block design created     Source File Properties   ? _ I II   Source File Properties   ? _ III   Y PS_PL_project.vhd   IIII Properties   ? _ IIII X   Search   Ctrl+P   Search   Ctrl+P   Search   Ctrl+P   Select All   Ctrl+A   Hessages   Log   Reports   Design Runs	BLOCK DE SIGN - blockdesign				
<ul> <li>k blockdesign</li> <li>(1) block design created</li> <li>(2) Now right click on the canvas and select "Add Module"</li> <li>(2) Now right click on the canvas and select "Add Module"</li> <li>Source Node Properties Ctrl+E</li> <li>Delete</li> <li></li></ul>	Sources Design × Signals Board ? _ [	Diagram × PS_	PL_project.vhd ×		
(1) block design created Source File Properties ? _ □ □ × @ PS_PL_project.vhd ← → ☆ General Properties Log Reports Design Runs	Q 王 뇌 🔅	0, 0, 5; ;		+   🔤	🎤 🛛 🖾 🖉 🖉
Source File Properties       ? _ □ Ľ ×		(2) Now righ	nt click on the canvas	and selec	t "Add Module"
Source File Properties       ? _ □ Ľ ×			Source Node Properties	Ctrl+E	
Source File Properties       ? _ □ Ľ ×		×	Delete	Delete	button to add IP
Image: Paste Ctrl+V         Image: Paste Ctrl		8	Сору	Ctrl+C	ballon to add in .
General     Properties     Properties     Select All     Ctrl+P       Tcl Console     X     Messages     Log     Reports     Design Runs		10	Paste	Ctrl+V	
Tcl Console × Messages Log Reports Design Runs	Image: Second	Q.	Search	Ctrl+F	
Tcl Console × Messages Log Reports Design Runs	General Properties	We	Select All	Ctrl+A	
TCLCONSOLE × Messages Log Reports Design Runs Add Module		+	Add IP	Ctrl+I	
			Add Module		
			IP Settings		



À Add Module	×	
Select a module to add to the I	block design.	Figure 3. 7: The VHDL entity is available in the list
Module type: RTL Search: Q- @ PS_PL_project (PS_PL_)	project.vhd)	The adjacent pop up window shows the VHDL entities that could be added in the schematic. Double Click on it.
this window pops up, double clicking on it a underneath	. Select your entity by and then click OK	The entity shows up in block form in the canvas as shown in Figure 3.8 below.
✓ Hide incompatible modu	(2) Iles	
?	OK Cancel	
	Diagram × PS PL	_project.vhd ×
	-	
	ଷ୍   ପ୍   💥   🕅	○   Q   꽃   ≑   <b>+</b>   ▷,   №   ⊠   C   의   4
	🏄 Designer Assistand	ce available. Run Connection Automation
	a block diagram created that rep the entity I have just created	PS PL project 0

Figure 3. 8: VHDL entity in block form

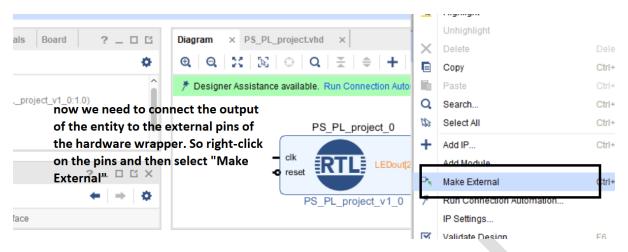


Figure 3. 9: Connecting the terminals to external pins

Figure 3.9 shows the procedure to define the pins as external pins so that later on they could be connected to physical pins of the SoC.

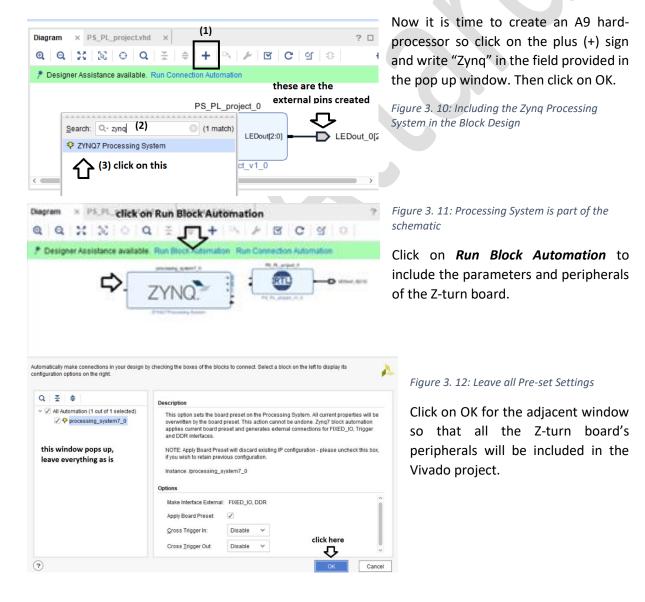


Diagram × PS_PL_project.vhd × Address Edit	or ×
$\mathbf{Q} \mid \mathbf{Q} \mid \mathbf{X} \mid \mathbf{\Sigma} \mid \mathbf{O} \mid \mathbf{Q} \mid \mathbf{X} \mid \mathbf{\varphi} \mid \mathbf{H} \mid$	୍ୟ 🖌 🗹 🖂 ଓ 🗠 🕄 ଅନ୍
₱ Designer Assistance available. Run Connection Auto	mation
DDR + FORD, 0 + FORD	PS_PL_project_0 PS_PL_project_0 PS_PL_project_v1_0 The FIXED_IO pins are connected to the right peripherals automatically. this is becuase we used the board files that we have included in the Vivado
ZYNQ7 Processing System the A9 processor is extended.	project. They are also assigned external pins of the SoC automatically again according to the board files. However now we need to assign the



appropriate pins to the LEDs on the board for my hardware design block

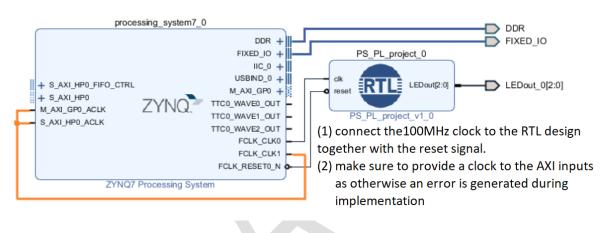
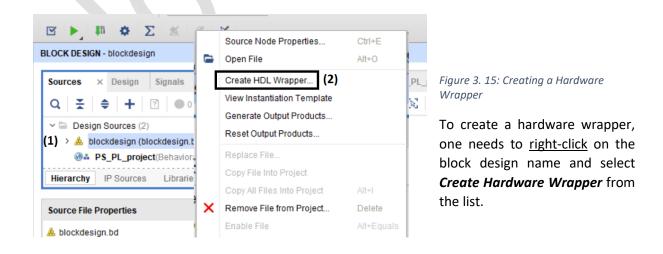


Figure 3. 14: connect the 100 MHz clock

Connect the **100 MHz** clock which is denoted as **FCLK\_CLKO** on the Zynq Processing System diagram to the VHDL entity's clock input. Even though Figure 3.14 shows that the **AXI\_GPn** inputs are connected to **FCLK\_CLK1**, it is <u>advisable not to do so</u> but to connect them also to the 100 MHz clock **FCLK\_CLK0**!



You can either add or copy the HDL wrapper file to the project. Use you would like to modify this file.	Leave Vivado to do all the work
This window pops up. Leave everythi	ing as is
Options	
O Copy generated wrapper to allow user edits	(OK
Let Vivado manage wrapper and auto-update	•
? ок	Cancel
🗹   🕨   👫   🍄   ∑   🖄   🖉   🎗	6
BLOCK DE SIGN - blockdesign	6
BLOCK DE SIGN - blockdesign	
	?_□C Diagram × PS_PL_project.vhd ×
BLOCK DE SIGN - blockdesign	? _ □ ⊡     Diagram     ×     PS_PL_project.vhd     ×       Updating      ♦     ●     Q     ¥     S     ●     Q     ¥
BLOCK DE SIGN - blockdesign Sources × Design Signals Board	?_□C Diagram × PS_PL_project.vhd ×
BLOCK DE SIGN - blockdesign Sources × Design Signals Board Q	? _ I I       Diagram × PS_PL_project.vhd ×         Updating O I       Q         Q       X         Make Sure that the updating is ready before
BLOCK DE SIGN - blockdesign Sources × Design Signals Board Q ∓ ≑ + ? ● 0 In Sources (2)	? _ I I       Diagram × PS_PL_project.vhd ×         Updating O O       Q         Q       Q         Q       Q         Make Sure that the updating is ready before
BLOCK DE SIGN - blockdesign Sources × Design Signals Board Q	? _ I I       Diagram × PS_PL_project.vhd ×         Updating O O       Q       C       Q       Z         Make Sure that the updating is ready before       Diagram × PS_PL_project.vhd ×       Q       Z

Figure 3. 16: Leave Vivado to Update

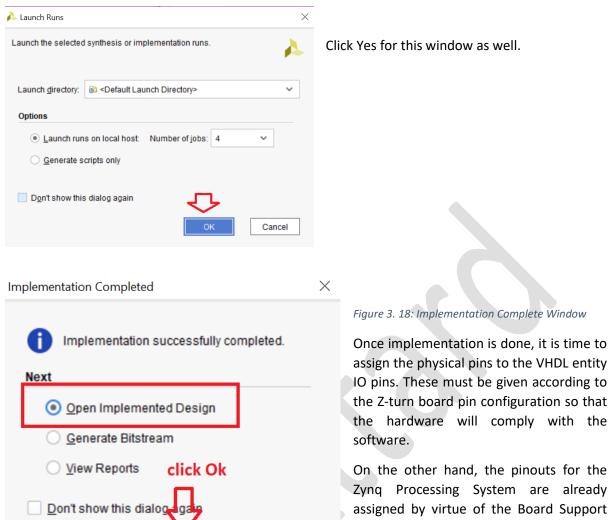
# Leave Vivado to update before continuing as otherwise Vivado will get mixed up!

Now click on Run Implementation.

Generate Block Design	Design Sources (1)
✓ SIMULATION	Blockdesign_wrapper(STRUCTURE) (blockdesign_wrapper:
Run Simulation	Synthesis is Out-of-date X
1	Hierarchy IF FO C
<ul> <li>RTL ANALYSIS</li> <li>Open Elaborated Design</li> </ul>	Synthesis is out-of-date. OK to launch synthesis first? Implementation will automatically start when synthesis completes.
Y SYNTHESIS	
Run Synthesis	General Pro
> Open Synthesized Design	Tcl Console     ×     Messages     Log     Reports     Design Runs
MPLEMENTATION Click here	
<ul> <li><u>Run Implementation</u> (1)</li> <li>Open Implemented Design</li> </ul>	<pre>export_simulation -of_objects [get_files G:/Z-TURN_V12_20171030/Zynq7020/PS_PL/PS_PL.srcs/source make_wrapper -files [get_files G:/Z-TURN_V12_20171030/Zynq7020/PS_PL/PS_PL.srcs/sources_1/bd/bl add_files -norecurse G:/Z-TURN_V12_20171030/Zynq7020/PS_PL/PS_PL.srcs/sources_1/bd/blockdesign/</pre>

Figure 3. 17: Synthesis Out-of-date

For Figure 3.17, click **Yes.** This is because of the new changes that have been done since the last synthesis.



Zynq Processing System are already assigned by virtue of the Board Support Files that has been installed as described in chapter 1.

Now open the implemented design and click on the IO Port tab as shown in Figure 3.19.

Cancel

OK

oard Part Interface No se pins to my ent	-	Package Pin		Fixed	Bank (Multiple)	I/O Std (Multiple)*
	-	Package Pin				
se pins to my ent	ntity's			$\checkmark$	(Multiple)	(Multiple)*
	F				13	default (LVC
m to reflect whe	ere the	V10	~		13	default (LVC
ected on the boa	oard V	V6	~		13	default (LVC
	1	W6	~		13	default (LVC
	ected on the bo	lected on the board	ected on the board V6 W6			

Figure 3. 19: Vivado assigns random pin numbers to terminals

Vivado assigns random pin numbers to the terminals of the hardware. These have to be changed to comply with the design of the Z-turn board.

ages Log Reports	Design Runs	Timing Power	Methodology	y DRC Packa	age Pins	I/O	Ports ×		?
+ H									٦
Direction	Board Part Pin	Board Part Interface	Neg Diff Pair	Package Pin		Fixed	Bank	I/O Std	
INOUT Changed t	ne pins acco	rding the Z-turn l	ooard.			$\checkmark$	(Multiple)	(Multiple)*	
OUT Also we ne	ed to chang	e the voltage fro	m 1V8			$\checkmark$	34	LVCMOS33*	
OUT to 3V3 as s	hown			R14	~	$\checkmark$	34	LVCMOS33*	
OUT				Y16	~	$\checkmark$	34	LVCMOS33*	
OUT				Y17	~	$\checkmark$	34	LVCMOS33*	

Figure 3. 20: Pins Comply with Z-turn Board

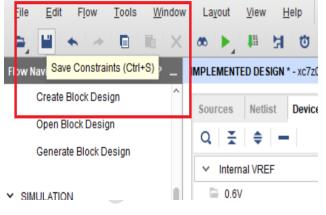
Now the pins are assigned the correct pin number so that the entity will be physically connected to the LEDs on the Z-turn board.

ଦ ≍ ≑ ୟ + ਖ਼								
Name	Direction	Board Part Pin B	oard Part Interface	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Sto
FIXED_IO_mio[5]	INOUT				A6	$\checkmark$	500	LVCM
FIXED_IO_mio[4]	INOUT				B7	$\checkmark$	500	LVCM
FIXED_IO_mio[3]	INOUT	this confirms that the	•		D6	$\checkmark$	500	LVCM
FIXED_IO_mio[2]	INOUT	connected correctly be			B8	$\checkmark$	500	LVCM
FIXED_IO_mio[1]	INOUT	one of the LEDs is conn	nected to pin E	6	A7	$\checkmark$	500	LVCM
FIXED_IO_mio[0]	INOUT				E6	$\checkmark$	500	LVCM
<								

Figure 3. 21: Confirming that MIO pins are correctly assigned

Figure 3.21 confirms that the Zynq Processing System pins are correctly connected to the peripherals. This could be confirmed from the schematic diagrams provided by **MYIR**.

## 🔥 PS\_PL - [G:/Z-TURN\_V12\_20171030/Zynq7020/PS\_PL/PS\_PL.xpr] - Vivado 201



#### Figure 3. 22: Saving the new constraints file

Since pinouts have been changed from the ones assigned by Vivado, a new constraints file will be created and saved. This new constraints-file will be part of the project and takes precedence over the one created automatically by Vivado.



À Save Constraints		×
Select a target file to write n Choosing an existing file w constraints. thi		e new 🔥
• <u>C</u> reate a new file		
<u>F</u> ile type:	XDC	*
File name:	name the const	raints file
Fil <u>e</u> location:	🛜 <local project="" to=""></local>	*
O Select an existing fil	e	
<select a="" targ<="" td=""><td>get file&gt; then click</td><td>OK</td></select>	get file> then click	OK
?	ок	Cancel

Figure 3. 23: Naming the new constraints file

Click on *Generate BitStream*. If Any windows pop up because *Synthesis* or *implementation* are <u>out</u> <u>of date</u>, just click on **OK** to re-do these stages again.

Bitstream Generation Completed X
Bitstream Generation successfully completed.
Next
O View Reports
Open Hardware Manager
O Generate Memory Configuration File
Don't show this dialog again
OK Cancel
Figure 3. 24: Bitstream Generated

Save Constraints As. (1) Fi Close Implemented Design	e viockdesign_wrapper(STRUCTU)
Open Checkpoint	archy IP Sources Libraries
Open Re <u>c</u> ent Checkpoint <u>W</u> rite Checkpoint	ce File Properties × Clock R
New IP Location	ockdesign_wrapper.vhd
Open IP Location Open Recent IP Location	eral Properties
Ne <u>w</u> File	onsole Messages Log
Open File Cl Open Recent <u>F</u> ile	K   +   D   €   X
Open IP-XACT File	(3) Export <u>Hardware</u>
Open Interactive Report	Export Constraints
Save All Files	Export Pblocks
Add Sources Al	Export IBIS Model
Open Source File C	rl+N Export I/O Ports
Import (2)	Export Bitstream File
Export (2)	Export Simulation

Figure 3. 25: Export the Hardware

After the bitstream file is generated, it is time to export hardware, so click on File  $\rightarrow$  Export  $\rightarrow$ 

#### Export Hardware.

À Export Hardware	×	
Export hardware platform for software development tools. (1) Include bitstream		Tick the <i>include bitstream</i> box and then click <i>OK</i>
Export to: Solution Project> (2) (2) (3) (3) (3) (3) (4) (4) (4) (4) (4) (4) (4) (4) (4) (4	]	

From the *file menu*, select *Launch SDK*. Now in the author's 13-inch laptop, sometimes this option is *not seen* on the screen so one might think that it is not included. Well *it is at the very end of the file menu* so one needs to scroll with the *arrow-down button* just one place down and then *hit enter* on the keyboard and the following window pops up as shown in Figure 3.26.

À Launch SDK		×
Launch software devel	opment tool.	
Exported location:	🗟 <local project="" to=""></local>	~
Workspace: 🛜 <	Local to Project>	~
?	ок	Cancel

Figure 3. 26: Start SDK from within Vivado

SDK will launch automatically and by default it will be pointing to the workspace where the Vivado project is.

PS\_PL.sdk - C/C++ - blockdesign\_wrapper\_hw\_platform\_0/system.hdf - Xilinx SDK File Edit Navigate Search Project Run Xilinx Window Help

Project Explorer ≅	\$ 7	🛍 system.hdf 🛛		
blockdesign_wrapper_hw_platform_0 blockdesign_wrapper.bit		blockdesign_wr	apper_hw_p	latfo
I source of the source of		Design Information		
b ps7_init_gpl.h Init.c b ps7_init.c b ps7_init.h Init.html		Created With:	xc7z020clg400-1 Vivado 2017.4	
<ul> <li>■ ps7_init.tcl</li> <li>■ system.hdf</li> </ul>		Created On: Address Map for pro	Fri Mar 23 20:39	
<ul> <li>ps7_init.td</li> <li>psystem.hdf</li> </ul>				
<ul> <li>ps7_init.tcl</li> <li>system.hdf</li> <li>his is the platform</li> </ul>		Address Map for pro	b <b>cessor ps7_cort</b> o Base Addr	exa9_[(
<pre>&gt;&gt; ps7_init.tcl &gt;&gt;&gt; system.hdf &gt;</pre>		Address Map for pro	Base Addr 0xf8f010	exa9_[( High / 0xf8f(
<pre>ps_init.td is psstem.hdf this is the platform project. It has all the necessary files for the</pre>		Address Map for pro Cell ps7_intc_dist_0 ps7_gpio_0	Base Addr 0xf8f010 0xe000a	High A 0xf8f0 0xe00
■ ps7_init.tcl		Address Map for pro Cell ps7_intc_dist_0 ps7_gpio_0 ps7_scutimer_0	Base Addr 0xf8f010 0xe000a 0xf8f006	High A Oxf8f0 Oxe00 Oxf8f0

#### Figure 3. 27: SDK IDE

The hardware part of the project is complete, and the files needed by SDK are already loaded in SDK. At this point one must create a new application project that will generate the necessary files to boot from the SD card. This application is called the **FSBL** application and this is what follows next.

B PLsdk - C/C++ - blockdesign\_wrapper\_hw\_platform\_0/system.hdf - Xilinx SDK File cdf1\_Navigate Search Project Run Xilinx Window Help

New 🤁 (2)	Alt+Shift+N >		Application Project (3)	
Open File			SPM Project	
Open Projects from File System			Board Support Package	
Close	Ctrl+W		Project	_
Close All	Ctrl+Shift+W	63	Source Folder	
Close All	Ctri+Shirt+W		Folder	
Save	Ctrl+S	C	Source File	
Save As		h	Header File	
Save All	Ctrl+Shift+S		File from Template	
Revert		ଙ	Class	В
Move			Other Ctrl+N	
Rename	F2	51	wap tor processor psr_cortexa:	<u>_[o-</u>
Refresh	F5		Base Addr Hi	

Figure 3. 28: Creating a new FSBL project

<b>Αρριιcaτιon Project</b> Create a managed make	application project	ct. <b>this v</b>	vindow pops up	
Project name: FSBL_proje		lame the pr	oject	
Use default location Location: G:\Z-TURN_V12 Choose file syste		17020\PS_PL\PS_P	L.sdk\FSBL_project	Browse
OS Platform: standalone Target Hardware				~
Hardware Platform: blo Processor: ps7	ckdesign_wrappei 'cortexa9 0	r_hw_platform_0		~ New
Target Software Language:	● C ○ C++		the name of the created automa	tically
Compiler: Hypervisor Guest:	32-bit N/A		while typing the the the typing the	
Board Support Package:	Create New Use existing	FSBL_project_bs	p	~
		Hit NFXT	underneath	

Figure 3. 29: Name the FSBL project

#### Templates



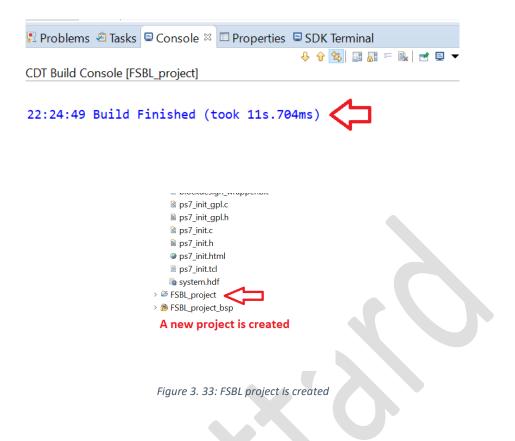
Create one of the available templates to generate a fully-functioning application project.

Dhrystone	First Stage Bootloader (FSBL) for Zynq. The FSBL	^
Empty Application	configures the FPGA with HW bit stream (if it	
Hello World	exists) and loads the Operating System (OS)	
IwIP Echo Server	Image or Standalone (SA) Image or 2nd Stage	
Memory Tests	Boot Loader image from the non-volatile	
OpenAMP echo-test	memory (NAND/NOR/QSPI) to RAM (DDR) and	
OpenAMP matrix multiplication Demo	starts executing it. It supports multiple	
OpenAMP RPC Demo	partitions, and each partition can be a code	
Peripheral Tests	image or a bit stream.	
RSA Authentication App		
Zynq DRAM tests		
	_	
Zynq DRAM tests Zynq FSBL A new window pops up. this t shown above	ime select Zynq FSBL as	
Zynq DRAM tests Zynq FSBL <b>A new window pops up. this t</b>	ime select Zynq FSBL as	
Zynq DRAM tests Zynq FSBL A new window pops up. this t shown above	ime select Zynq FSBL as	
Zyng DRAM tests Zyng FSBL A new window pops up. this t shown above Hit FINISH underneath	ime select Zynq FSBL as Select the FSBL template	

Peripheral Drivers	While the FCDL puste at it.	haing granted aligh on the						
Drivers present in the Board	Supp <mark>While the FSBL project is</mark>							
<	Console tab to see the pro	ogress						
Overview Source	ל ל							
🗈 Problems 🧔 Tasks 🗳 Conso	ole 🛿 🗖 Properties 📮 SDK Terminal	🗖 🗖 📓 SDK Log 🛱						
"Running Make libs in	<pre></pre>							
	SUK LOG ~							
-0 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 -	22:10:37 INFO : Launching	g XSCT server: xsct.bat -interactiv	v					
<pre>'sa_v1_4/src -s include SHE }_0/libsrc/canps_v3_2/src" )s_v3_2/src -s libs "SHELL=C</pre>	22:10:40 INFO : Successfu 22:10:42 INFO : Successfu	fully done setting XSCT server conne fully done setting SDK workspace ng command line option -hwspec G:/2						
Also take note of the	workspace		ì					
progress as shown h		Building workspace: (14%)						

Figure 3. 32: Notice the progress bar

Make sure to check the progress bar on the lower right of the screen. This is necessary to understand when one should continue to open a new C project.



At this point it is very important to check whether the hardware part of the system is working before trying the software part or the PS part. So create a boot image file from the FSBL project and check that the programmable logic is working.

Now create a new <u>*C project*</u> where C code will control the Zynq Processing System. This is done by clicking on **File**  $\rightarrow$  **New**  $\rightarrow$  **Application Project then** 

		When this v	vindow pops	
Project name: My_C_pro	ject	up you need	to give the C	
✓ Use default location		project a na	me	
ocation: G:\Z-TURN_V1	2_20171030\Zynd	7020\PS_PL\PS_PI	L.sdk\My_C_project	Browse
Choose file syst	tem: default 🖂			
	ie ockdesign_wrappe 7_cortexa9_0	er_hw_platform_0	Make sure that are the settings	
Target Software Language:	● C ○ C++ 32-bit	t	his is created wh yping the name c	
Compiler:		— <b>—</b> p	oroject above	
Compiler: Hypervisor Guest:	N/A	્રપ્		

Figure 3. 34: Naming the C project

Dhrystone Empty Application Helio World MP Echo Server Memory Tests OpenAMP echo-test OpenAMP matrix multiplication Demo OpenAMP RPC Demo Peripheral Tests RSA Authentication App Zynq DRAM tests Zynq FSBL	Let's say 'Hello World' in C. This project template is the only one that will compile correctly so I have to choose it by force. Also to be able to use this application, I have to make sure that I have anyone of the UARTs of my A9 core enabled, otherwise this application project could not be selected.	
? < Back No	ext > Finish Cancel	

Figure 3. 35: Choosing the Hello World Template

Again, wait for SDK to finish compiling and building the workspace.

As stated before, at least one of the UARTs must be enabled for this C project to be available for the user. It must be said that the UART that could be used for debugging is **UART 1** and therefore one needs to make sure that it is properly enabled from the C-project's Board Support Package (BSP). The following figures will illustrate the steps.

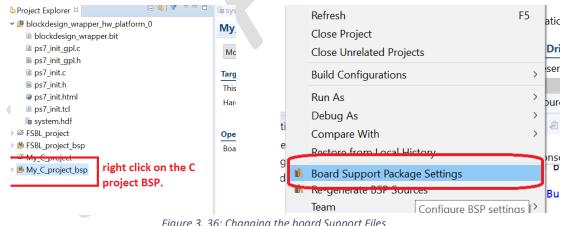


Figure 3. 36: Changing the board Support Files

When the changes are done, hit **OK**, then the whole project is re-built again automatically.

#### Board Support Package Settings

#### **Board Support Package Settings**

Control various settings of your Board Support Package.

✓ Overview standalone (1)	Configuration for OS:	standalone			
✓ drivers	Name	Value	Default	Туре	Description
ps7_cortexa9_0	hypervisor_guest	false	false	boolean	Enable hypervisor gue
	stdin	ps7_uart_1	none	peripheral	stdin peripheral
	stdout	ps7_uart_1	none	peripheral	stdout peripheral
	zynqmp_fsbl_bsp	false	false	boolean	Disable or Enable Opt
	> microblaze_exception	s false	false	boolean	Enable MicroBlaze Exc
	> enable_sw_intrusive_p	r false	false	boolean	Enable S/W Intrusive
Locate the <i>HelloWorl</i>	<i>d.c</i> file: ● psr_munum		This Board S	aunne	
	ps7_init.tcl		Hardware St		
	system.hdf		Target		
	<ul> <li>FSBL_project</li> <li>FSBL_project hsp</li> <li>FSBL_project hsp</li> <li>FSBL_project</li> </ul>	C project and hello-world ( and double o open it and o	C source file Vers Click on it toescript edit it Documental Peripheral D Drivers pres < v Overview Sou	ion: ion: ion: ion: <b>river</b> ent in	
	Target Connections 🛛	_ <b>₽</b>	😫 🗖 🖪 Problems 🗸	a Tasl	
	Figure 3.	37: Locating Hello	World File		

#### Software for the Processing System

Now it is time to write the software to flash the LED connected to MIO 0 and MIO 9 on the Z-turn board. The user does not have to assign any pins in the constraints file because these form part of the Board Support Files.

The first thing is to include the xgpiops.h file

<pre>#include</pre>	<stdio.h></stdio.h>
<pre>#include</pre>	"platform.h"
<pre>#include</pre>	"xil_printf.h"
<pre>#include</pre>	"xgpiops.h"

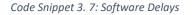
Code Snippet 3. 3: Include the xgpiops.h

 $\times$ 

```
int main()
    int status;
{
    XGpioPs_Config *ConfigPtr;
    XGpioPs PS_GPIO;
    init_platform();
    /*Configuring Bank 0 which is actually bank 500 on Zyng SoC*/
     ConfigPtr = XGpioPs_LookupConfig(XPAR_PS7_GPI0_0_DEVICE_ID);
     status = XGpioPs_CfgInitialize(&PS_GPIO, ConfigPtr,ConfigPtr -> BaseAddr);
     if(status != XST SUCCESS)
     {
         printf("initialisation failure");
         return XST_FAILURE;
     }
                      Code Snippet 3. 4: Initializing the MIO port
/*set the port direction 0 = input 1=output*/
XGpioPs_SetDirection(&PS_GPI0, 0, 0xFFFFFFF);
/*Enable the port 0 = pin disabled, 1=pin enabled*/
XGpioPs_SetOutputEnable(&PS_GPIO, 0, 0xFFFFFFF);
/*On the Z-turn board the two LEDs assigned to the PS part of the Zynq 7
 *are connected to MIO0 and MIO9. In this program i used the Port-write
 *function instead of the pin-write function*/
          Code Snippet 3. 5: Setting the Port Direction and Enabling the Output Port
         while(1)
         {
            debounce();
            print("Hello World\n\r");
            XGpioPs_Write(&PS_GPIO, 0, 0x00000201);//MIO0 and MIO9
            delay();
            XGpioPs_Write(&PS_GPIO, 0,0x00000000);
            delay();
         3
        cleanup_platform();
        return 0;
          Code Snippet 3. 6: Switching on and off the LEDs + printing on Serial Port
```

All the above instructions are explained in chapter 2 and therefore it will not be repeated here.

```
•void delay (void)
{
    for(unsigned long i = 0; i < 10000000; i++)
    {
        //do nothing
    }
}
•void debounce (void)
{
    for(unsigned long i = 0; i < 100000000; i++)
    {
        //do nothing
    }
}</pre>
```



The delays are used so that the LEDs could be seen blinking.

Saving the C file will immediately start the build operation again. When this is finished one can create the boot image file that will be saved in the SD card. This is show in figure 3.38.

🐞 system.hd		Rename	F2	it_pla
> SBL_project	1.21	Import		int("H
> M ESBL project	<u> </u>	Export		
✓ <sup>™</sup> My_C_projec > <sup>™</sup> Binaries > <sup>™</sup> Includes		Build Project Clean Project		
> 🗁 Debug		Refresh	F5	
> 🗁 src		Close Project		
> 🌁 My_C_projec		Close Unrelated Projects		
		Build Configurations	>	
		Run As Debug As Compare With 1) right click on the C 2) select "Create Boot"		
👛 Target Connecti		Restore from Local History		- 🖉 Tasks
> 🖻 Hardware Se > 🖻 Linux TCF Ag > 🖻 QEMU TcfGd	2 1	C/C++ Build Settings Generate Linker Script Change Referenced BSP		nsole [N
	<b></b>			Build
		Team	>	
Figure 3. 38: ci	rea	ting a boot image file		

Creates Zynq Boot Image in .bin format from given FSBL elf and partition files in specified output folder.

Architecture: Zynq	~	
Create new BIF file	$\bigcirc$ Import from existing BIF file	
Basic Security		
Output BIF file path:	$\label{eq:c_rurn} G:\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \$	Browse
UDF data:		Browse
Split	Output format: BIN ~	
Output path:	G:\Z-TURN_V12_20171030\Zynq7020\PS_PL\PS_PL.sdk\My_C_project\bootimage\BOOT.bin	Browse

This window pops up. It shows that SDK will create a .bif file and a boot.bin file.

(bootloader) G:\Z-TURN_V12_20171030\Zynq7020\PS_PL\PS_PLsdk\FSBL_project\Debug\FSBL_project.elf       none       none         G:\Z-TURN_V12_20171030\Zynq7020\PS_PL\PS_PLsdk\blockdesign_wrapper_hw_platform_0\blockdesign_wrapper.bit       none       none         G:\Z-TURN_V12_20171030\Zynq7020\PS_PL\PS_PLsdk\blockdesign_wrapper_hw_platform_0\blockdesign_wrapper.bit       none       none         G:\Z-TURN_V12_20171030\Zynq7020\PS_PL\PS_PLsdk\blockdesign_wrapper_hw_platform_0\blockdesign_wrapper.bit       none       none         Take note of the project files and their sequence. first the First Sequence Boot Loader       first	File path	Encrypt	Auther
G:\Z-TURN_V12_20171030\Zynq7020\PS_PL\PS_PLsdk\My_C_project\Debug\My_C_project.elf none none <b>Take note of the project files and their sequence. first the First Sequence Boot Loader first</b>	(bootloader) G:\Z-TURN_V12_20171030\Zynq7020\PS_PL\PS_PL.sdk\FSBL_project\Debug\FSBL_project.elf	none	none
Take note of the project files and their sequence. first the First Sequence Boot Loader first	G:\Z-TURN_V12_20171030\Zynq7020\PS_PL\PS_PL.sdk\blockdesign_wrapper_hw_platform_0\blockdesign_wrapper.bit	none	none
	G:\Z-TURN_V12_20171030\Zynq7020\PS_PL\PS_PL.sdk\My_C_project\Debug\My_C_project.elf	none	none
	Take note of the project files and their sequence. first the First Sequence Boot Loader	r first	
then the .bit file from Vivado and the last file is the .elf file created by the C project	then the .bit file from Vivado and the last file is the .elf file created by the C project		
	Hit Create Image underneath		

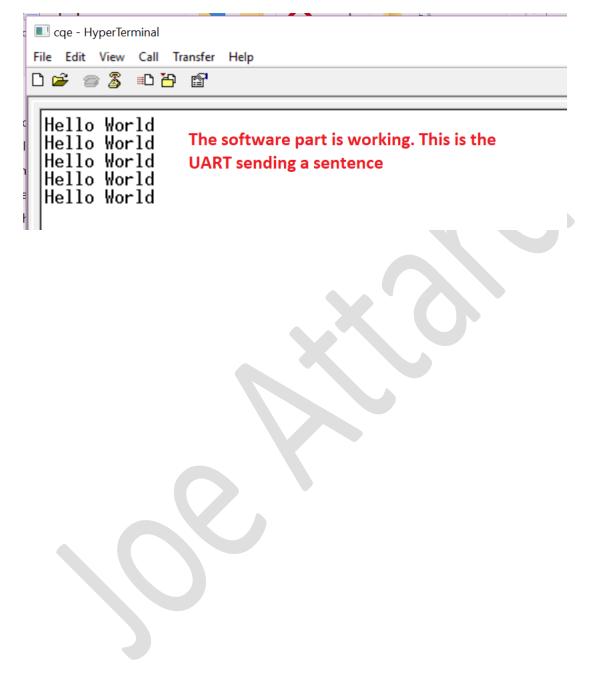
Figure 3. 39: Three files make up the boot image file

py Paste	Paste shortcut	Move Copy to • to •	Delete Rename	New folder	Properties	Invert selection		
Clipboard			rganize -TURN_V12_201710	New 30 > Zynq7020 > PS_P	Open PL > PS_PL.sdk > My_	Select C_project > bootima	ige	the location where the boot.bin file resides $_{\rm Se}$
e - Malta (	^ Name			Date modified	Туре	Size		-
e - Person	() BOOT			23/03/2018 22:5	2 PowerISO File	4,08	5 KB	This is the boot.bin file
ation Elect	Ø My_C_pre			23/03/2018 22:5			1 KB	
s	Copy thi	s boot.bin	file and paste	e it on the SD card	l			

Figure 3. 40: Location of the Boot image file

ြန္မ

Eject the SD card from the computer and insert it in the Z turn board. The program should start, there will be the two green LEDs on MIO 0 and MIO 9 blinking together with the LEDs that are connected to the Programmable Logic part. On the serial monitor, the Processing System will transmit the *Hello World* message.



## Detecting the slide switches on the Z-turn Board from Programmable Logic

In this chapter, three out of the four DIP switches located on the Z-turn board will be used as select inputs to a multiplexer (MUX). The MUX will light a combination of LEDs according to the combination of the switches' inputs.

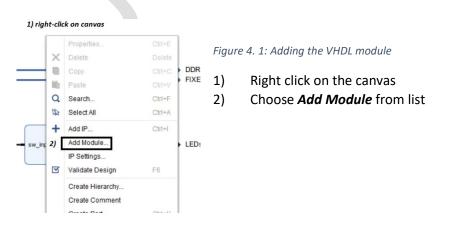
This chapter will focus on the interface part of the switches with Programmable logic and therefore the VHDL code is simple.

The process how to create a project together with how to include a VHDL source file has already been illustrated in previous chapters, so this will not be covered here.

Code snippet 4.1 shows the VHDL code to implement a MUX in hardware. The LEDs and the switches are both data busses.

```
entity MUX is
     Port ( sw input : in STD LOGIC VECTOR (2 downto 0);
            LEDs : out STD LOGIC VECTOR (2 downto 0));
end MUX;
architecture Behavioral of MUX is
 begin
 LEDs <= "001" when sw input = "001" else
         "010" when sw_input = "010" else
         "011" when sw input = "011" else
         "100" when sw_input = "100" else
         "101" when sw input = "101" else
         "110" when sw_input = "110" else
         "111" when sw input = "111" else
         "ZZZ";
end Behavioral;
             Code Snipper 4. 1: VHDL Code of a simple MUX
```

Create a block Design: This has already been shown in previous chapters, so it will not be repeated here. Make sure to include both the Zynq Processing System and the VHDL module (the MUX).



Add Module Select a module to add to the block design.	<ul> <li>The adjacent window pops up. Select the VHDL</li> <li>module of the MUX, then click on <i>OK</i>.</li> </ul>
Module type: RTL ~	Click on <i>Run Automation</i> at the top of the canvas and click on OK.
1) MUX (MUX.vhd) here you have all the VHDL modules that you created You can add as many instances as you like to your design	
Hide incompatible modules  Hide incompatible modules  Cancel	
BLOCK DESIGN - block design	Source Node Properties Ctrl+E Open File Alt+O
Sources     ×     Design     Signals     E       Q     ₹     €     +     ?     ●	Create HDL Wrapper     2)       /iew Instantiation Template     hd       Generate Output Products     Reset Output Products
> @ block_design(STRUC	Replace File Copy File Into Project Copy All Files Into Project Alt+I
	Remove File from Project Delete
Save the design.	
processing_sys	stem7_0
	DDR + DDR FIXED_IO + FIXED_IO USBIND_0 + H

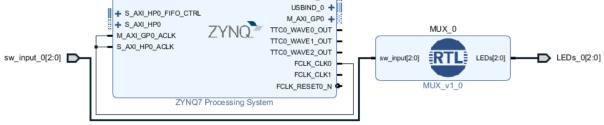


Figure 4. 3: The Full circuit diagram

Figure 4.2 shows the circuit diagram for this project. Note that *FCL\_CLKO* is connected to *M\_AXI\_GPO\_ACLK* and *S\_AXI\_HPO\_ACLK*. Note also that the MUX will be implemented in combinational logic, therefore no clock signal is required.

## Click on *Run Implementation*

> Open Elaborated Design	
	Å block_design.bd 🗧 🖷
✓ SYNTHESIS	<
Run Synthesis	General Properties
> Open Synthesized Design	Tcl Console × Messages Log Reports Des
	Q ≚ ≑ II ₪ III 🖬
<ul> <li>Run Implementation</li> <li>Open Implemented Design</li> </ul>	VHDL Output written to : G:/Z-TURN_V12_201 make_wrapper: Time (s): cpu = 00:00:06 ; e add_files -norecurse G:/Z-TURN_V12_2017103 undate compile order -fileset sources 1
If there are no errors then when prom	a completed to open the implemented design, click on OK.
	ementation successfully completed.
Next	n Implemented Design
	erate Bitstream
	/ Reports

Don't show this dialog

Figure 4. 4: Implementation Ready

Click on *OK* for the above window. Choose the *I/O Ports* tab and click on the arrow (>) of both *LEDs\_0* and *sw\_inputs\_0*.

Tcl Console Messages Log	Reports	Design Runs	s Timing	Power	DRC Pa	ackage Pins	I/O Ports ×	1) select IO por	ts
Q X ≑ <b>4 +</b> 5	I								
Name	Direction	1	Board Part Pin	Board Pa	rt Interface	Neg Diff Pa	ir Package Pin	Fixed	Bank
> 🗟 DDR_6075 (71)	INOUT							$\checkmark$	502
> 🗟 FIXED_IO_6075 (59)	INOUT							$\checkmark$	(Multiple)
> 🐔 LEDs_0 (3)	OUT	(2) change th	e pinouts to	the ones a	lready co	nnected on th	ne Z-turn board		13
> 📴 sw_input_0 (3)	IN								13
Scalar ports (0)									

Figure 4. 5: I/O Tab

> 🗟 DDR_6075 (71)	INOUT		$\checkmark$	502	(Multiple)*
> 🗟 FIXED_IO_6075 (59)	INOUT		$\checkmark$	(Multiple)	(Multiple)*
V Contraction (3)	OUT		$\checkmark$	34	LVCMOS33*
LEDs_0[2]	OUT	R14	~ 🗸	34	LVCMOS33*
LEDs_0[1]	OUT	Y16	~ 🗸	34	LVCMOS33*
LEDs_0[0]	OUT	Y17	~ ✓	34	LVCMOS33*
✓ <sup>™</sup> sw_input_0 (3)	IN		$\checkmark$	(Multiple)	LVCMOS33*
Image: Sw_input_0[2]	IN	J15	~ 🗸	35	LVCMOS33*
Image: Sw_input_0[1]	IN	G14	~ 🗸	35	LVCMOS33*
Image: Sw_input_0[0]	IN	T19	× 🗸	34	LVCMOS33*

Figure 4. 6: Pinouts of the system

#### To arrive at Figure 4.6, one has to do the changes illustrated step by step in Figures 4.7.

Power										HSTL_II_18 HSTL_I_18 HSUL_12		
Reports	Design Runs	Timing	Power	DRC	Package Pins		/O Ports	×		LVCMOS12 LVCMOS15 LVCMOS18		?
3oard Part Pin	Board Part I	nterface	Neg Diff Pai	r Pac	kage Pin		Fixed	Bank 2)	5	LVCMOS25 LVCMOS33 V		Ve
	1)pi	nouts ch	anged	Y16		~	$\checkmark$		34	default (LVCMOS18)		
				Y17		~	$\checkmark$		34	default (LVCMOS18	)	•
									13	default (LVCMOS18	)	v

Figure 4. 7: Changing the operating voltage of the IOs and their pinouts

Figure 4.7 shows that the pinouts for the LEDs are changed to match the location of the LEDs on the Z-turn board. Apart from that one must change the operating voltage for both LEDs and switches to LVCMOS33. This will avoid errors later.

Since the constraints file was changed, Vivado asks to save the changes in a different constraints file. Give a name to the new constraints file.

Select a target file to write new unsaved constrain Choosing an existing file will update that file with constraints.	
● <u>C</u> reate a new file	
Eile type: 1 XDC	~
File name: name the constrain	nts file
Fil <u>e</u> location: 🔂 <local project<="" th="" to=""><td>~</td></local>	~

Figure 4. 8: Name the new constraints file

Double click on *generate bitstream* so that the .bit file is created. When the bitstream is created, one has to *export the hardware including the bitstream file*, then launch *SDK*.

Open File	Guito	ne				
Open Recent <u>F</u> ile			th 1 (activa)			
Open IP-XACT File		synth_1 (active)				
		V √ impl_1				
Save All F <u>i</u> les		Out	-of-Context Module Runs			
Add Sources from file	тепи: (	(2)	Export <u>H</u> ardware			
Open Source File	Ctrl+N		Export Block Design			
I <u>m</u> port			Export Bitstream File			
Export (1)	•		Export Simulation			
Launch SDK (3)						

Figure 4. 9: Exporting the hardware and Launching SDK

Since the Zynq Processing System is not used in this project, there is no need to create a *C* code project. All one must do after the *.bit* file is generated, and the project is *exported*, is create a *FSBL* project in *SDK* and create a boot image <u>from</u> the FSBL project. This is shown in Figure 4.10.

Project Explorer 🛛	🖹 🚖 🛛 🔻	~ - 8	system.hdf
🛩 进 block_design_wrap	pper_hw_platform_0		FSBL_app_bs
block_design_w	rapper.bit		abb_as
gps7_init_gpl.c			Modify this BSF
📓 ps7_init_gpl.h			modify this bot
gs7_init.c			Target Informat
🖹 ps7_init.h			This Board Supp
ps7_init.html			Hardware Specif
ps7_init.tcl			
🗎 system.hdf			Target Pro
> 🖉 FSBL_app 🛛 🖊	ight click on the FSBL ap	plicatio	n Operating Syste
> 🎊 FSBL_app_bsp 🛛 🕇	hen select create boot i	mage	Board Support F

Figure 4. 10: FSBL project

The steps to create a First Stage Boot Loader project has been described in chapter 1 so there is no need to show how it is done here.

		Debug As	>
		Compare With	>
👛 Target Co		Restore from Local History	
> 🗁 Hardw		C/C++ Build Settings	
> 🗁 Linux 🛛	5	Generate Linker Script	
> 🗁 QEMU	Au -	Change Referenced BSP	
- 42.000		Create Boot Image	
		leam	>
		Configure	>

After the Boot image file is created, it can be found in the project  $\rightarrow$  SDK  $\rightarrow$  FSBL project  $\rightarrow$  Bootimage folder.

Figure 4. 11: Create a Boot Image File

#### Using the DIP switches with the Processing System

In this chapter, the Processing System will monitor the state of the DIP switches which are connected to the Programmable Logic part of the Zynq 7. According to the combination of the state of the switches, the tri-colour RGB LEDs, which are also connected to the Programmable Logic will light to indicate which of the switches is active. The RGB LEDs are connected to pins R14, Y16 and Y17 while the switches are connected to J15, G14, T19 and R19. These were derived from the schematic diagram of the Z-turn board.

The following stages have been discussed in previous chapters so they will not be included again here.

- 1) Create a Vivado Project
- 2) **<u>DO NOT</u>** create a VHDL file
- 3) Click to create a block design
- 4) On the canvas click on the plus-sign (+) in the middle
- 5) Write *Zynq* in the field of the pop-up window then select the *Zynq processing system* from the list available
- 6) Right-click somewhere on the canvas and left-click on add IP
- 7) Write AXI *GPIO* in the field provided in the pop-up window
- 8) Select the AXI GPIO from the selection list
- 9) **Double-Left-click** in the **middle** of one of the AXI GPIO blocks and the pop-up window in Figure 5.1 pops up.

Show disabled ports	Component Name axi_gpio_0		
	Board IP Configuration		
	Associate IP interface with board interfa	ice	
	IP Interface	Board Interface	
	GPIO	rgb led	
	GPI02	sws 4bits	
+ S_AXI GPIO +	Clear Board Parameters		

Figure 5.1 shows the AXI block named axi\_gpio\_0. This AXI block has <u>two channels</u> named **GPIO** and **GPIO2**. Both channels are <u>32 bits</u> wide.

The drop down menu shows that the AXI block can be connected to the external peripherals according to the Z-turn board's support files which the user must download and install in the Vivado path so that Vivado would know which type of dev-board, one is using and therefore there are presets that could be utilized. This was discussed in chapter 1.

Since the preset configurations are going to be kept, then GPIO and GPIO2 will consist of three outputs and four inputs respectively.

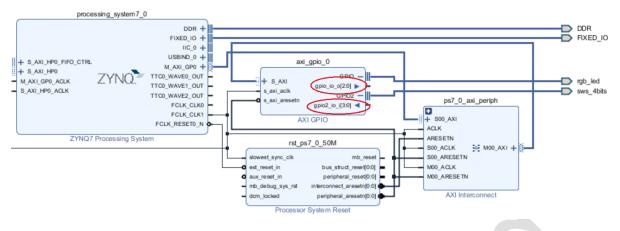


Figure 5. 2: AXI block channels configured as input and output

As shown in Figure 5.2, the channels within the AXI GPIO block assume their directions whether they are inputs or outputs automatically if the presets are used! note **rgb\_led** and **sws\_4bits** assigned to the pins according to the Board Support files!

Whenever there is an <u>AXI block</u>, one must include the <u>AXI interconnect block</u> together with the <u>Processor System Reset</u> block. This will make life easier at a later stage when compiling the project. So by right-clicking on the canvas and selecting the <u>ADD IP</u> for both cases, one will be able to include these two blocks in the design. Make sure to reduce the number of Master interfaces of the ACI interconnect block from 2 to 1 as shown in Figure 5.3.

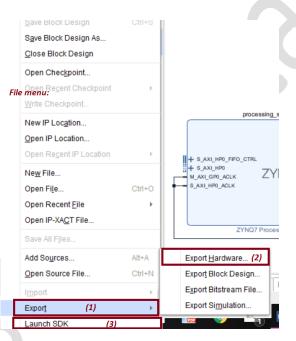
AXI Interconnect (2.1)	4
1 Documentation 🕞 IP Location	
Component Name ps7_0_axi_periph	
Top Level Settings Slave Interfaces	
Number of Slave Interfaces 1	Î
Number of Master Interfaces 1 ~	
Interconnect Optimization Strategy Custom ~	

Figure 5. 3: Reduce the number of Master Interfaces in the AXI Interconnect Block

It is very important to note that, the user is not restricted to what is available on the dev-board bought! Because if **"custom"** is selected from the drop-down list, then it will become a free 32-bit GPIO which can be connected to any external peripheral.

10) After setting up the AXI blocks, one needs to click on *Run Auto-Connection* button on the top of the canvas. This will route all the blocks to interface the Zynq Processing System with the AXI GPIO blocks.

- 11) Now *point the mouse to the pins* of the AXI GPIO blocks and one by one, right-click and click on *"make external"*.
- 12) Do not forget to create a *hardware wrapper* by right-clicking on the block design and select *"create hardware wrapper"* from the list
- 13) After all the above is done, double-click on "Run implementation".
- 14) When prompted to open the implementation design, click on *OK* because from the implementation screen, *one can designate the appropriate pin assignments* according to the schematics of the dev-board. However, this time, since only the *presets* of the board are used, the pin assignments *should be already correct*. It is not a bad idea if one would double-check that the pin assignments are correct!
- 15) At this point all that needs to be done is to generate the bitstream file.
- 16) After the bit stream is successfully generated, the project should be <u>exported</u> and the <u>bitstream included</u> as shown in Figure 5.3.
- 17) Then *SDK* could be launched from within the project. The *SDK* should be resident to the project itself so click on *OK* when prompted.



*Figure 5. 4: Export Hardware include the bitstream file* 

Once SDK is opened it will immediately open a project for the hardware created in Vivado. At this point one needs to create a *First Stage BootLoader Application* from the file menu.

- 18) In SDK, create a FSBL application
- 19) Then create a hello-world application
- 20) Enable UART 1 from the Board Support Package of the Hello World code application by right clicking on the hello-world application project then select *change the referenced BSP*

Points 18 to 20 have already been shown in previous chapters.

#### Now the software part

The focus of this chapter is on this part because the previous parts should by now be familiar with the learner. So first include the libraries for the AXI GPIO block. This is called **gpio\_v4\_3** and could be found in the path shown in Figure 5.5

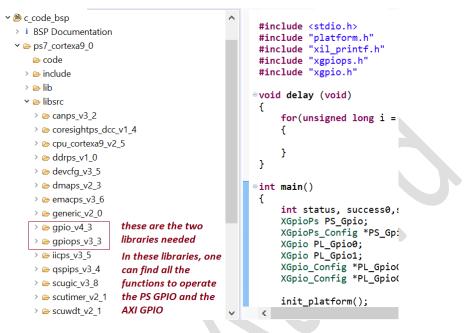
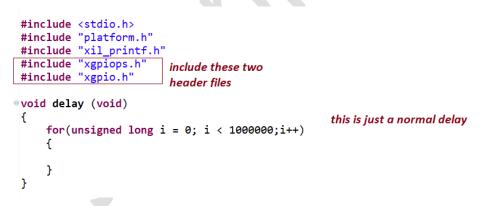


Figure 5. 5: Path to find the AXI GPIO library

All the functions that control the AXI GPIO block are listed in the library shown in Figure 5.5 above. For this project, the two LEDs connected to MIO 0 and MIO 9 will also be used. Incidentally these two LEDs are connected to pins **E6** and **B5**, however nothing should be done cause its already part of the constraints file due to the board support files.



The *xgpiops.h* header files are included in the C source file to have access to gpiops\_v3\_3 library or the MIO port pins. The *xgpio.h* is included to have access to the Programmable Logic pins via the AXI GPIO block.

When the processor enters the delay function, it till loop in the delay for 1 million times and then returns to the main program. That way, the main program is slowed so the LEDs can be seen blinking. Next initialize the MIO port and the AXI block as shown in the code on the next page.

Code Snippet 5. 1: Initializing the MIO port

The above function is found in *xgpiops\_sinit.c* file. The argument *u16 DeviceId* is found in *xgpiops\_g.c* file. The original function is the following:

XGpioPs\_Config \*XGpioPs\_LookupConfig(u16 DeviceId)

*XGpioPs\_Config* is the return type of this function so this function must be equated to a pointer that has the same attributes as *XGpioPs\_Config*. This is done by declaring a pointer of the same type at the beginning of the main () and equate that pointer to the above function. So, declare the variable:

XGpioPs\_Config \*ConfigPtr;

Then equate it to the function:

ConfigPtr= XGpioPs\_LookupConfig(XPAR\_PS7\_GPIO\_0\_DEVICE\_ID);

After the LookupConfig(), one has to initialize the port. This is done by

```
s32 XGpioPs_CfgInitialize(XGpioPs *InstancePtr, XGpioPs_Config *ConfigPtr,u32 EffectiveAddr)
```

The above function returns a *s*32 value so one must declare a variable of type *s*32 at the beginning of the main () and equate it to this function. This is done below:

s32 status;

XGpioPs \*InstancePtr must also be declared at the beginning of the main (), like so:

XGpioPs myPSGpio;

XGpioPs\_Config \*ConfigPtr has to be replaced with ConfigPtr like before and for u32 EffectiveAddr, one must write ConfigPtr -> BaseAddr. This was defined in xgpiops\_hw.h file.

#define XGpioPs\_WriteReg(BaseAddr, RegOffset, Data) \
 Xil\_Out32((BaseAddr) + (u32)(RegOffset), (u32)(Data))

The *IF* statement that follows will check whether the previous function has been successful. If not, the program will stop there and nothing else happens.

Now to initialize the AXI GPIO block, the following function found in *xgpio\_sinit.c* must be used.

int XGpio\_Initialize(XGpio \* InstancePtr, u16 DeviceId)

again, the above returns a variable or type *int* and therefore such a variable has to be declared at the beginning of the main (). The *XGpio* \* *InstancePtr* must be replaced with an instance pointer that must be declared as well. This is shown in the following declarations:

#### XGpio myGpio;

And this should be written with an *ampersand* (&) sign in front of it.

#### int success;

The *device ID* should be copied from *xgpio\_g.c* file. The complete statement is shown below:

# /\* Initialise the PL GPIO driver\*/ /\*int XGpio\_Initialize(XGpio \*InstancePtr, u16 DeviceId);\*/ success=XGpio\_Initialize(&myGpio,XPAR\_AXI\_GPIO\_0\_DEVICE\_ID);

Just like for the initialization of the MIO port, one can use an IF statement to verify the success of the initialization. Here it is not included; however it is a good idea that it will be done.

From *xgpiops.c* file, use

#### void XGpioPs\_SetDirectionPin(XGpioPs \*InstancePtr, u32 Pin, u32 Direction)

The above function does not expect a return variable, so it does not have to be equated. For the instance pointer argument should be replaced with **&myPSGpio**, the **u32 Pin** should be replaced with the pin number – in this case **0** since one of the LEDs is connected to bit 0 and the direction argument should be filled with **1** because it defines an <u>output</u> while **0** defines an <u>input</u>.

```
/*Now we need to set the port direction of each pin in the PS GPIO*/
XGpioPs_SetDirectionPin(&myPSGpio,0,1); //MIO[0] set as output
XGpioPs_SetDirectionPin(&myPSGpio,9,1); // MIO[9] set as output
/*both MIO outputs above have an LED connected with them on the board*/
```

Now as an observation, even though the project works, however since the last argument is defined as a <u>32-bit</u> argument, one must write it in 32-bit form so it would be advisable to write the second function for MIO 9 as follows:

XGpioPs\_SetDirectionPin(&myPSGpio,9,0x00000200); // MIO[9] set as output

Or to be even safer one should use the bank function and not the individual pin function as follows:

void XGpioPs\_SetDirection(XGpioPs \*InstancePtr, u8 Bank, u32 Direction)

where *XGpioPs* \**InstancePtr* is replaced with *&myPSGpio* as before, the *u8 Bank* will be replaced with *0* because it is bank 0 and *u32 Direction* will be replaced by 0x00000201

XGpioPs\_SetDirection(&myPSGpio, 0, 0x00000201);

It is time to enable the outputs so from xgpiops.c file copy:

void XGpioPs\_SetOutputEnablePin(XGpioPs \*InstancePtr, u32 Pin, u32 OpEnable)

XGpioPs \*InstancePtr is replaced with &myPSGpio, u32 Pin is replaced with 0 and 9 respectively in different function calls, and u32 OpEnable is replaced with 1 declaring that the output is now enabled.

/\*for the MIO outputs to work, i need to enable them by \*/
XGpioPs\_SetOutputEnablePin(&myPSGpio,0,1); //enable MIO[0] pin
XGpioPs\_SetOutputEnablePin(&myPSGpio,9,1); //enable MIO[9] pin

Now for the AXI GPIO block, one only has to set the direction of the individual channels as a whole and not individual pins. This is illustrated below:

```
/* Now i am going to set the direction of the PL GPIO*/
XGpio_SetDataDirection(&myGpio,2,0x0000000F); //bits 3:0 are connected to switches
XGpio_SetDataDirection(&myGpio,1,0x00000000);//RGB LEDs are connected to the
//lower 3 bits of this port
```

Code Snippet 5. 2: Setting the AXI GPIO direction

The function can be found in *xgpio.c* file.

void XGpio\_SetDataDirection(XGpio \*InstancePtr, unsigned Channel,u32 DirectionMask)

*XGpio* \**InstancePtr* is replaced with *&myGPIO*, channel argument is replaced with either 1 or 2. 1 represents *GPIO channel* while 2 represents *GPIO2 channel*. *U32 DirectionMask* determines the pin direction – in the AXI GPIO case, <u>logic 0</u> represents that pin is an <u>output</u> while <u>logic 1</u> means that the pin is an <u>input</u>. It is the reverse for the MIO port!

The code snippet 5.2 shows that channel 1 is declared as output while channel 2 is declared as input.

Now for the while (1) code. A while (1) statement defines an infinite loop. This means that the microcontroller will continue looping inside this loop forever. The code must check the state of the switch-bank and according to the state of each switch, it will light a combination of the RGB LEDs. To check the state of the switches one must read the whole channel and filter out the un-needed bits. This is called Bit-Masking where a bitwise-AND-function is done with individual bits of the channel. By ANDing with 0 all the bits that are not of interest will be discarded while when ANDing with 1 – the bits will be considered.

An if statement was used that selects the pattern of the LEDs according to the state of the switches.

The bank-read function was used to read the state of the bank as a whole shown below:

u32 XGpioPs\_Read(XGpioPs \*InstancePtr, u8 Bank)

it returns an **unsigned 32-bit** variable containing the state of the whole channel. Again **XGpioPs \*InstancePtr** is replaced by &myGPIO while u8 Bank is replaced by the channel number, in this case 2 because that is where the switches are assigned. In the same statement the returned value from the read function is immediately ANDed with another variable to extract the state of the switches. It must be noted that, the switches are effectively connected to the least significant nibble of the channel.

The code lights the LEDs connected to the MIO port if the combination of the switches does not match any from the if statements, otherwise the MIO LEDs will be switched off while the LEDs on the Programmable Logic part will reflect the combinations of the switches' inputs. Code is show in the next page.

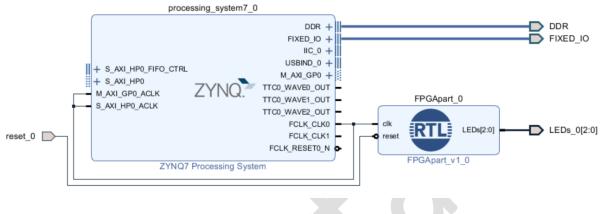
```
while(1)
ſ
print("press a switch\n\r");
delay();
sw detection = (XGpio DiscreteRead(&myGpio,2) & 0x0000000F);//read channel 2 cause switch
//are connected to channel 2 bits 3:0
/* Switches are connected as active low switches */
if(sw_detection == 0x0000001)
{
    XGpio_DiscreteWrite(&myGpio,1,0x0000000E); //LED will light with logic 0
// XGpioPs_Write(&myPSGpio, 0,0x00000000); //switch off LEDs on PS bank 0
}
else if(sw detection == 0x0000002)
{
   XGpio_DiscreteWrite(&myGpio,1,0x0000000D); //LED will light with logic 0
   XGpioPs_Write(&myPSGpio, 0,0x00000000); //switch off LEDs on PS bank 0
//
}
else if (sw_detection == 0x00000004)
{
   XGpio_DiscreteWrite(&myGpio,1,0x0000000B); //LED will light with logic 0
    //XGpioPs_Write(&myPSGpio, 0,0x0000000); //switch off LEDs on PS bank 0
}
else
{
     XGpio_DiscreteWrite(&myGpio,1,0x0000000F); //switch off LEDs on PL side
   XGpioPs_Write(&myPSGpio, 0,0x00000009); //write in bank 0 to affect MIO[9]
11
                                               // and MIO[0]
```

```
}
```

# Interfacing with the Button Switch on the Z-turn board

The Z-turn board has 2 button switches. Both switches are active low. One of these button switches is connected to the *Reset* pin of the Zynq 7 while the second button switch is designated as *USER* button and therefore one could use it in his projects.

The procedure to create a Vivado project, how to create a block design and how to include the Zynq Processing System is already covered in previous chapters. Do not forget the create a hardware wrapper before generating the bitstream file. Figure 6.1 shows a typical system.





The block diagram shows a separate VHDL module created in the Programmable Logic. This module is used to test whether the boot image file has been loaded in the Zynq 7 properly, so for this project, the VHDL module could be removed. The reset switch of the VHDL module is not the same reset switch mentioned in the introduction of this chapter.

Notice also that the button switch connected to MIO 50 is not seen in the diagram just like the LEDs connected to MIO 0 and MIO 9. These are default in the constraints file generated by the Board Support Files.

% FIXED_IO_mio (54)	INOUT			$\checkmark$	(Multiple)	(Multiple)*
FIXED_IO_mio[53]	INOUT		C11	$\checkmark$	501	LVCMOS18
FIXED_IO_mio[52]	INOUT		C10	$\checkmark$	501	LVCMOS18
FIXED_IO_mio[51]	INOUT		B9	$\checkmark$	501	LVCMOS18
FIXED_IO_mio[50]	INOUT		B13	$\checkmark$	501	LVCMOS18
FIXED_IO_mio[49]	INOUT		C12	$\checkmark$	501	LVCMOS18
FIXED_IO_mio[48]	INOUT		B12	$\checkmark$	501	LVCMOS18
FIXED_IO_mio[47]	INOUT		B14	$\checkmark$	501	LVCMOS18
FIXED_IO_mio[46]	INOUT		D16	$\checkmark$	501	LVCMOS18
FIXED_IO_mio[45]	INOUT		B15	$\checkmark$	501	LVCMOS18
FIXED_IO_mio[44]	INOUT		F13	$\checkmark$	501	LVCMOS18

Figure 6. 2: Part of the constraints file

Figure 6.2 shows part of the constraints file. If the VHDL module was not included in project, all that had to be done is just generate a bitstream file straight away, however since in this project, a VHDL module was also included, a new constraints file was created to accommodate the changes in the Programmable Logic part.

The **USER** button switch is connected to MIO 50. The pin is connected via a pull up resistor to 1V8. For this particular project, leave the default voltage of 1V8 in the IO settings in Vivado as shown in Figure

6.2. Generate a bitstream file, export the hardware including the bitstream file and Launch SDK from within the Vivado project.

In the following section, the software part and its intricacies will be discussed.

#### The Software

In SDK, the usual FSBL application has to be created as part of the project. After that create a new C project following the usual steps as described in previous chapters.

The *gpiops\_v3\_3* library will be used in this project. Therefore, include the *xgpiops.h* file as usual.

Initialize the MIO bank as shown in the following code. Detailed explanation of this code has already been covered in previous chapters.

```
PS_GpioConfigPtr = XGpioPs_LookupConfig(XPAR_PS7_GPI0_0_DEVICE_ID);
PSGpioStatus =XGpioPs_CfgInitialize(&PSGpio, PS_GpioConfigPtr,PS_GpioConfigPtr ->BaseAddr)
if(PSGpioStatus != XST_SUCCESS)
{
    return XST_FAILURE;
}
XGpioPs_SetDirection(&PSGpio, 0,0xFFFFFFF); // 1= output 0 = input
XGpioPs_SetOutputEnable(&PSGpio, 0,0xFFFFFFF); // 1 = enable
```

```
Code Snippet 6. 1: Initializing the MIO bank
```

Even though MIO 50 is on **bank 1**, this does not mean that any changes to the *device-ID* has to be made. It should remain the same *XPAR\_PS7\_GPIO\_0\_DEVICE\_ID*.

Code snippet 6.1 also shows that all MIO pins have been enabled as outputs. This is convenient to make sure that all the pins are enabled. However, it is advised to configure the pins needed as inputs separately, by using the following function:

void XGpioPs\_SetDirectionPin(XGpioPs \*InstancePtr, u32 Pin, u32 Direction)

where **XGpioPs \*InstancePtr** is replaced by the name of the instance in this case **&PSGpio**, **u32 Pin** must be replaced with the pin number, in this case **50** (for MIO 50) and the direction should be set to **0** as it must be configured as <u>input</u>.

```
XGpioPs_SetDirectionPin(&PSGpio, 50, 0); //this sets MIO 50 to input
```

The author tried using the function where all the bank is configured with one function using the

XGpioPs\_SetDirection(&PSGpio, 1,0xFFFBFFFF); // only MIO 50 is set as input in bank 1

But for some reason, it did not work!

Now, since the focus of this chapter is to learn how to use the button switch on MIO 50, the C code does a simple task to demonstrate its use. It waits for a button press then blinks both LEDs on MIO 0 and MIO 9 at the same time, once, then waits for another switch press.

To read from the port, the same concept must be adopted. Use the

u32 XGpioPs\_ReadPin(XGpioPs \*InstancePtr, u32 Pin)

function. This function returns a u32 value. *XGpioPs \*InstancePtr* should be declared as shown in previous chapters while *u32 Pin* should be replaced with the pin number - in this case *50* because it is connected to MIO 50.

```
sw_MI050 = XGpioPs_ReadPin(&PSgpio, 50);
if(sw_MI050 != 1)
{
    XGpioPs_WritePin(&PSgpio, 0, 1);
}
else
{
    XGpioPs_WritePin(&PSgpio, 0, 0);
}
```

Code Snippet 6. 2: detecting the Button Switch

The return value of the read-pin function is stored in a variable of type u32. Since the switch is active low, when the switch is idle (not pressed), the function returns a 1 because of the pull-up feature of the circuit. Once the switch is pressed, it connects MIO 50 to ground and therefore the function returns a 0. This is monitored by the *IF* statement. Note that the LEDs light when a logic 0 is at the output of the pin.

# Processing System Dual AXI block control

In this chapter, two AXI GPIO blocks will be controlled from the Processing System. This is the maximum number of AXI GPIO blocks one can use with the Processing System because the **xgpio\_v4\_4** library functions within SDK, only cater for two AXI GPIO blocks. However, it must be said that each AXI GPIO block has a total of 64 IOs and therefore there is more than enough IOs with two AXI GPIO blocks!

In the meantime, one can use as many AXI GPIO blocks as the application needs, with the Programmable Logic part of the Zynq 7. However, again this number is limited with the number of physical IOs the Zynq 7 has, but in case of internal connections, one can add as many AXI GPIO blocks as needed. The final circuit is shown in Figure 7.1 below.

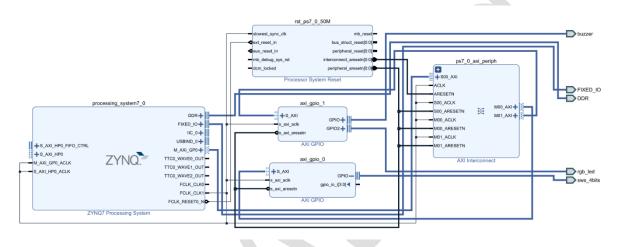


Figure 7. 1: final Block Design

The process to create a project and a block design has already been discussed in previous chapters so these will not be mentioned here anymore. Also, when using AXI GPIO blocks, one needs to include the *AXI interconnect block* and the Processor Reset Block, both shown in Figure 7.1. Again, these have been explained in previous chapters and therefore they will not be mentioned here.

One way to use two AXI GPIO blocks is to configure one of the AXI block to accept inputs while the second AXI GPIO block can be configured to be all made up of outputs. That way, one can have up to 64 inputs and up to 64 outputs! Do we need more?!

In the project described in this chapter AXI\_GPIO\_0 block's channel 1, is connected to the DIP switches and therefore from the 64 inputs, only 4 are used. On the other hand, AXI\_GPIO\_1 is configured as output block. The two available channels within the AXI block are used, channel 1 is connected to the onboard piezo buzzer while channel 2 is connected to the LEDs. Both peripherals reside on the Programmable Logic side. To implement these settings, one must go through the following steps:

- 1) Double Click on the AXI Block shown in Figure 7.2
- 2) For channel 1 within AXI\_GPIO\_0, click on the drop-down menu and select sws\_4bits as shown in Figure 7.3
- 3) Now double click again on AXI\_GPIO\_1 and select the LEDs and buzzer as shown in Figure 7.4.

AXI GPIO (2.0)		4			
🚺 Documentation 🛛 🗁 IP Location					
Show disabled ports	Component Name axi_gpio_0				
	Board IP Configuration				
	Associate IP interface with board interface IP Interface	Board Interface			
	GPIO	Custom			
	GPIO2	Custom 👻			
+ S_AXI s_axi_aclk GPIO +	Clear Board Parameters				
	Figure 7. 2: Double Click on the AXI bl	lock			
AXI GPIO (2.0)		4			
1 Documentation 🛛 🗁 IP Location					
Show disabled ports	Component Name axi_gpio_0				
	Board IP Configuration				
	Associate IP interface with board interface				
	IP Interface	Board Interface			
	GPIO GPIO2	sws 4bits  Custom			
	01102	buzzer			
IF S AXI	Clear Board Parameters	rgb led			
s_axi_acik GPIO +     • s_axi_aresetn		sws 4bits			
	Figure 7. 3: Selecting the DIP switch	es			
AXI GPIO (2.0)		À			
1 Documentation 📄 IP Location					
Show disabled ports	Component Name axi_gpio_1				
	Board IP Configuration				
	Associate IP interface with board interface				
	IP Interface	Board Interface			
	GPIO GPIO2	buzzer  rgb led  v			
+ S_AXI s_axi_aclk GPIO + s_axi_aresetn GPIO2 +	Clear Board Parameters				
	1				



The next thing is to <u>assign an address</u> to both the AXI GPIO blocks to be memory mapped, shown in Figure 7.5 below:

Cell						
	Slave Interface	Base Name	Offset Address	Range	High Address	
✓ ₱ processing_system7_0						
👻 🖽 Data (32 address bits : 0)	(40000000 [ 1G ])					
🚥 axi_gpio_0	S_AXI	Reg	0x4120_0000	64K 🔹	0x4120_FFFF	
🚥 axi_gpio_1	S_AXI	Reg	0x4121_0000	64K 🔹	0x4121_FFFF	assigned to
🗸 🖨 Unconnected Slaves						the AXI blocks
🚥 processing_system7_0	S_AXI_HP0	HP0_DDR_LOWOCM				
Double Click on the individual AXI block	-	7. 5: Assigning Ad				

The above is achieved if one *right-clicks* on the individual AXI GPIO block, and from the menu select *Assign Address*. The addresses will be <u>automatically</u> given to the respective AXI GPIO block. This step is also shown in one of the previous chapters.

Save the block design and create a Hardware Wrapper. Then synthesize the model.

Open the Synthesized design and check the assigned pinouts of the DIP switches, the piezo buzzer and the LEDs. Due to the board support files, these should have the right pin assignments.

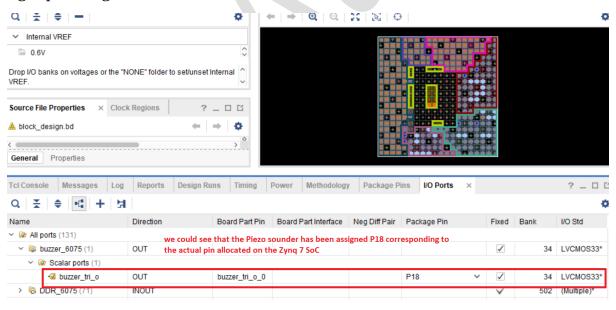


Figure 7. 6: Pin assignment of the Buzzer

Note that the working voltage is changed to 3V3 and the box under the column called *Fixed* is ticked. Also note the name of the pinout, together with the assigned pin number P18 which is taken from the Board Support Files and therefore nothing should be changed from this tab. The above also <u>holds true for the switches and the LEDs</u>. These are explained better in Figure 7.7 and Figure 7.8 on the next page.

# Authored by Joseph Attard

sws_4bits_tri_i (4)	IN					$\checkmark$	wumpte	LVCMOS33*
Image: Sws_4bits_tri_i[3]	IN	sws_4bits_tr	these are	J15	~	$\checkmark$	35	LVCMOS33*
Image: Sws_4bits_tri_i[2]	IN	sws_4bits_tr	the pins connected to	G14	~	note t worki	ne ng voltage	LVCMOS33*
Image: ws_4bits_tri_i[1]	IN	sws_4bits_tr	the DIP	T19	~	is 3V3		LVCMOS33*
Image: Sws_4bits_tri_i[0]	IN	sws_4bits_tr	switches	R19	~	$\checkmark$	34	LVCMOS33*

Eiguro 7	7. Dinc	accianad	to the	חוח י	cuvitchoc
riyure 7.	7. PIIIS	ussiyneu	10 1116	: DIP	switches

✓ <sup>1</sup> gb_led_tri_o (3)	OUT					$\checkmark$	34	LVCMOS33*
·	OUT	rgb_led_tri_o	here we can see the	Y17	~	$\checkmark$	34	LVCMOS33*
·	OUT	rgb_led_tri_o	RGB LEDs pinouts	Y16	~	$\checkmark$	34	LVCMOS33*
·	OUT	rgb_led_tri_o		R14	~	$\checkmark$	34	LVCMOS33*
Scalar ports (0)					_			

*Figure 7. 8: Pinouts where the LEDs are connected* 

Note that the LEDs and the Buzzer cannot form part of the same channel within the same AXI GPIO block because Vivado does not allow this to happen. So, to accommodate both the RGB LEDs and the Buzzer, two channels within the same AXI GPIO block must be used.

After saving to a new constraints file, it is time to generate a bitstream file, export the hardware (including the bitstream) and launch SDK from within the project.

À Export Hardv		×
	This window pops up	
Export hardward development to	re platform for software pols.	A
	(1) Tick here	
✓ Include b	itstream	
Export to:	👼 <local project="" to=""></local>	~
(2) cl	ick OK	
?	СК	Cancel

*Figure 7. 9: Include the Bitstream File when exporting the hardware* 

Image: State of the state	Help	< SDK			
Image: Constraint of the second s	■ system.hdf ¤ block_design_wrapper_hw_platform_0 Har				
<ul> <li>ps7_init_gpl.c</li> <li>ps7_init_gpl.h</li> <li>ps7_init.c</li> <li>ps7_init.h</li> <li>ps7_init.html</li> <li>ps7_init.tcl</li> </ul>	Design InformationTarget FPGA Device:7z020Part:xc7z020clg400-1Created With:Vivado 2017.4Created On:Sun May 13 21:29:22 2018				
<ul> <li>system.hdf</li> <li>SDK pops up and goes immediately to the Vivado project workspace.</li> <li>Above one can see the hardware project included in SDK</li> </ul>	Address Map for proce Cell ps7_intc_dist_0 ps7_gpio_0 ps7_scutimer_0	Base Addr 0xf8f010 0xe000a 0xf8f006		Slave	

*Figure 7. 10: SDK project linked to the Vivado project* 

Create a First Stage Boot Loader (FSBL) project, then create a C project from where control of the AXI GPIO blocks will be done. This is what will be learnt new in this chapter!

Again, choose the Hello World project from the list as shown in Figure 7.11.

Available Templates:		
Dhrystone	Let's say 'Hello World' in C.	^
Empty Application Hello World		
wir Echo Server		
s Memory Tests		
<ul> <li>OpenAMP echo-test</li> <li>OpenAMP matrix multiplication Demo</li> </ul>		
OpenAMP RPC Demo		
Peripheral Tests		
RSA Authentication App		
Zynq DRAM tests Zynq FSBL		
Zynd i Sbe		
c		
in la		
J .		
1		
	Click here	$\sim$
c		
4		
· ② < Back	Next > Finish Car	ncel
Figure 7. 11: Choosin	g the Hello World project from	list

	SDK Log 🛛			₽ ₽
Path source	18:58:51 I 18:58:53 I 18:58:54 I 18:58:54 I 18:58:54 I	IFO : X IFO : S IFO : S	aunching XSCT server: xsc (SCT server has started su Successfully done setting Successfully done setting Processing command line op	ccessfully. XSCT server conne SDK workspace
>	<			>
			Building workspace: (9%)	
		я <sup>х</sup> -	へ 🖸 🍖 🌰 ปุ๊ม 🌾 🛄 🖸 U	<u>IC 19:12</u> IS 17/05/2018 3

Figure 7. 12: Wait for SDK to finish compiling

Figure 7.12 is very important! One must wait for the SDK workspace to finish building! This could be checked from the bottom-right-corner of the screen.

If the UART is needed by the C application, one must modify the Board Support Package as shown in Figure 7.13.

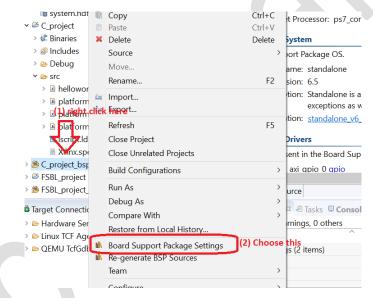


Figure 7. 13: Opening the Board Support Package

Configuration for OS: (2) Select this	standalone			
Name	Value	Default	Туре	Description
hypervisor quest	false	false	boolean	Enable hyper
stdin	ps7_uart_1	none	peripheral	stdin periphe
stdout	ps7_uart_0 ∨	none	peripheral	stdout perip
zynqmp_fsbl_bsp	none		boolean	Disable or Er
> microblaze_exception	sps7_coresight_cor	np_0	boolean	Enable Micrc
> enable_sw_intrusive_p	r <mark>ps7_uart_0</mark> ps7_uart_1		boolean	Enable S/W I
up				
change from UART 0 to	UART 1			
	Name hypervisor quest stdin stdout zynqmp_fsbl_bsp microblaze_exceptions enable_sw_intrusive_p	Name Value hypervisor quest false stdin ps7_uart_1 stdout ps7_uart_0 v zynqmp_fsbl_bsp none microblaze_exceptions ps7_uart_0 enable_sw_intrusive_pr ps7_uart_1	Name     Value     Default       hypervisor quest     false       stdin     ps7_uart_1       none       stdout     ps7_uart_0        zynqmp_fsbl_bsp     none       microblaze_exceptions     ps7_coresight_comp_0       enable_sw_intrusive_pr     ps7_uart_0       ps7_uart_1     ps7_uart_1	Name     Value     Default     Type       hypervisor quest     false     false     boolean       stdin     ps7_uart_1     none     peripheral       stdout     ps7_uart_0      none     peripheral       zynqmp_fsbl_bsp     none     boolean       microblaze_exceptions     ps7_coresight_comp_0     boolean       enable_sw_intrusive_pr     ps7_uart_0     boolean

Control various settings of your Board Support Package.

Figure 7. 14: Change to UART 1

Wait for SDK to compile the project.

# The Software for this project

Open the *Hello\_World.c* file located in the src folder in the C project. Include the *xgpiops.h* and *xgpio.h* header files using the #include "xxxxxx.h" directive at the beginning of the C file.

#include <stdio.h>
#include "platform.h"
#include "xil\_printf.h"
#include "xgpiops.h"
#include "xgpio.h"

As shown in previous chapters, now open the <u>gpio\_v4\_3</u> folder. This contains the library of functions that could be used with the AXI GPIO blocks. To use the MIO port, one has to open the <u>gpiops\_v3\_3</u> folder. Now from these two folders one can extract or copy the functions to configure both the MIO bank and the AXI GPIO blocks.

```
//initialise the PS GPIO
PS_GpioConfigPtr= XGpioPs_LookupConfig(XPAR_PS7_GPI0_0_DEVICE_ID);
status = XGpioPs_CfgInitialize(&PS_Gpio, PS_GpioConfigPtr,PS_GpioConfigPtr -> BaseAddr);
if(status != XST_SUCCESS)
{
    return XST_FAILURE;
}
//set whether the PS port pins are to act as inputs or outputs 1 = output 0 = input
XGpioPs_SetDirection(&PS_Gpio,0,0x00000201); //MIO_0 & MIO_9 are set as outputs
//enable the individual pins of the same bank
XGpioPs_SetOutputEnable(&PS_Gpio, 0, 0x00000201);
                             Code Snippet 7. 1: Initializing the MIO Bank
 //initialise the AXI GPIO 0 block
 PL_GpioConfigPtr0 = XGpio_LookupConfig(XPAR_AXI_GPIO_0_DEVICE_ID);
 success0 = XGpio_Initialize(&PL_Gpio0, XPAR_AXI_GPIO_0_DEVICE_ID);
 if(success0 != XST_SUCCESS)
 {
     return XST_FAILURE;
 }
 //initialize the AXI GPIO 1 block
PL GpioConfigPtr1 = XGpio LookupConfig XPAR AXI GPIO 1 DEVICE ID;
success1 = XGpio_Initialize(&PL_Gpio1, XPAR_AXI_GPIO_1_DEVICE_ID);
if(success1 != XST_SUCCESS)
{
    return XST_FAILURE;
}
//In this application we need both channels of AXI GPI01 to be set as outputs
XGpio_SetDataDirection(&PL_Gpio1,1,0x00000000); //piezo connected with channel 1 of AXI 1
XGpio_SetDataDirection(&PL_Gpio1, 2 0x00000000);//LEDs are connected to channel 2 of AXI 1
```

//In this application we have a separate AXI block for the inputs. Only channel 1 is used XGpio\_SetDataDirection(&PL\_Gpio0, 1,0x0000000F);//switches are connected to channel 1 of AXI 0 Code Snippet 7. 2: Initializing the two AXI blocks

Looking at *Code Snippet 7.2*, one should notice that there are <u>two</u> instances of AXI GPIO. These are named as <u>*PL Gpio0*</u> and <u>*PL Gpio1*</u>. These are the names of the AXI GPIO blocks!

Within AXI\_GPIO1 block, two channels are used. These are named as channel 1 and channel 2. <u>Two different channels</u> were needed, one to drive the piezo buzzer while the second channel was used to drive the RGB LEDs. This is because the *Board* Support information was used in this project. A workaround to be more efficient with the IO pins, is to change the channel to <u>custom</u> when <u>configuring the AXI GPIO block</u> <u>within Vivado</u>, opt for <u>4 outputs</u> and these should be enough to control each LED and the buzzer. However, in this project, the main focus is to show how to control two AXI GPIO blocks from the Processing System.

All the variables and pointers used in above functions must be declared at the beginning of the main () as shown in Code Snippet 7.3.

```
int status, success0,success1,sw_positions;
XGpioPs PS_Gpio; // PS GPIO pointer pointing to the bank where the IOs are
XGpioPs_Config *PS_GpioConfigPtr; //this is the configuration pointer for the PS GPIO
XGpio PL_Gpio0;//this is the first AXI GPIO 0 pointer
XGpio PL_Gpio1;//this the second pointer for AXI GPIO 1
XGpio Config *PL GpioConfigPtr0; //these are the configuration pointers for
XGpio Config *PL GpioConfigPtr1; //both FPGA AXI blocks
                      Code Snippet 7. 3: Declaring the pointers and variables
  while(1)
   print("check switches\n\r");
 //read the position of the switches and mask
   sw_positions = (XGpio_DiscreteRead(&PL_Gpio0, 1) & 0x0000000F);
   switch(sw_positions)
   {
   case 0x00000000:
           XGpio_DiscreteWrite(&PL_Gpio1, 1, 0x00000000); //switch off piezo
           XGpio_DiscreteWrite(&PL_Gpio1, 2, 0x0000000F); //switch off PL LEDs
           XGpioPs_Write(&PS_Gpio, 0, 0x00000201);//MIO_0 & MIO_9 are off
           break:
   case 0x0000001:
       XGpio DiscreteWrite(&PL Gpio1, 1, 0x00000001); //switch on piezo
       XGpio_DiscreteWrite(&PL_Gpio1, 2, 0x0000006); //switch off PL LEDs
       XGpioPs_Write(&PS_Gpio, 0, 0x00000201);//MIO_0 & MIO_9 are off
       break;
   case 0x00000002:
       XGpio_DiscreteWrite(&PL_Gpio1, 1, 0x00000000); //switch off piezo
       XGpio_DiscreteWrite(&PL_Gpio1, 2, 0x00000001); //switch on PL LEDs
       XGpioPs_Write(&PS_Gpio, 0, 0x00000201);//MIO_0 & MIO_9 are off
       break;
```

```
case 0x0000003:
         XGpio_DiscreteWrite(&PL_Gpio1, 1, 0x00000000); //switch off piezo
         XGpio DiscreteWrite(&PL Gpio1, 2, 0x00000002); //switch on PL LEDs
         XGpioPs_Write(&PS_Gpio, 0, 0x00000201);//MIO_0 & MIO_9 are off
         break;
 case 0x0000004:
         XGpio_DiscreteWrite(&PL_Gpio1, 1, 0x00000000); //switch off piezo
         XGpio_DiscreteWrite(&PL_Gpio1, 2, 0x00000003); //switch on PL LEDs
         XGpioPs_Write(&PS_Gpio, 0, 0x00000201);//MIO_0 & MIO_9 are off
         break;
 case 0x00000005:
         XGpio_DiscreteWrite(&PL_Gpio1, 1, 0x00000000); //switch off piezo
         XGpio DiscreteWrite(&PL Gpio1, 2, 0x00000004); //switch on PL LEDs
         XGpioPs Write(&PS Gpio, 0, 0x00000201);//MIO 0 & MIO 9 are off
         break:
 case 0x0000006:
         XGpio_DiscreteWrite(&PL_Gpio1, 1, 0x00000000); //switch off piezo
         XGpio_DiscreteWrite(&PL_Gpio1, 2, 0x00000005); //switch on PL LEDs
         XGpioPs_Write(&PS_Gpio, 0, 0x00000201);//MIO_0 & MIO_9 are off
         break;
case 0x0000007:
       XGpio_DiscreteWrite(&PL_Gpio1, 1, 0x00000000); //switch off piezo
        XGpio_DiscreteWrite(&PL_Gpio1, 2, 0x0000006); //switch on PL LEDs
        XGpioPs_Write(&PS_Gpio, 0, 0x00000201);//MIO_0 & MIO_9 are off
        break;
case 0x0000008:
        XGpio_DiscreteWrite(&PL_Gpio1, 1, 0x00000000); //switch off piezo
        XGpio_DiscreteWrite(&PL_Gpio1, 2, 0x00000007); //switch on PL LEDs
        XGpioPs_Write(&PS_Gpio, 0, 0x00000200);//MIO_0 on only
        break:
case 0x0000009:
       XGpio_DiscreteWrite(&PL_Gpio1, 1, 0x00000000); //switch off piezo
        XGpio_DiscreteWrite(&PL_Gpio1, 2, 0x00000007); //switch on PL LEDs
       XGpioPs_Write(&PS_Gpio, 0, 0x00000001);//MIO_9 on only
        break;
case 0x000000A:
            XGpio_DiscreteWrite(&PL_Gpio1, 1, 0x00000001); //switch on piezo
            XGpio_DiscreteWrite(&PL_Gpio1, 2, 0x00000000); //switch on PL LEDs
            XGpioPs Write(&PS Gpio, 0, 0x00000200);//MIO 0 on only
            break;
case 0x000000B:
           XGpio_DiscreteWrite(&PL_Gpio1, 1, 0x00000000); //switch off piezo
            XGpio_DiscreteWrite(&PL_Gpio1, 2, 0x0000000E); //switch on PL LEDs
            XGpioPs_Write(&PS_Gpio, 0, 0x00000001);//MIO_0 and MIO_9 on only
            break;
```

case	0x0000000C:		
		XGpio_DiscreteWrite(&PL_Gpio1, 1, 0x00000001); XGpio_DiscreteWrite(&PL_Gpio1, 2, 0x0000000E); XGpioPs_Write(&PS_Gpio, 0, 0x00000200);//MIO_0 break:	//switch on PL LEDs
case	0x0000000D:	-	
cusc		<pre>XGpio_DiscreteWrite(&amp;PL_Gpio1, 1, 0x00000000);</pre>	
		XGpio_DiscreteWrite(&PL_Gpio1, 2, 0x0000000C); XGpioPs_Write(&PS_Gpio, 0, 0x00000001);//MIO_0 break;	
case	0×0000000E:		
		<pre>XGpio_DiscreteWrite(&amp;PL_Gpio1, 1, 0x00000000); XGpio_DiscreteWrite(&amp;PL_Gpio1, 2, 0x00000003); XGpioPs_Write(&amp;PS_Gpio, 0, 0x00000200);//MIO_0 break;</pre>	//switch on PL LEDs
case	0x000000F:		
		XGpio_DiscreteWrite(&PL_Gpio1, 1, 0x00000000); XGpio_DiscreteWrite(&PL_Gpio1, 2, 0x00000000); XGpioPs_Write(&PS_Gpio, 0, 0x00000000);//MIO_0 break;	//switch on PL LEDs

Code Snippet 7. 4: Main Code

Code Snippet 7.4 detects the positions of the 4 DIP switches and according to their relative positions, a combination of RGB LEDs and the on-board buzzer are used to determine how the DIP switches are placed. A switch-case statement is used this time because it is more efficient than the if-else statement for this application.

All that is needed now is to save the .C file and this will start compilation. If no errors are found, then create a Boot image file, copy it to SD card, transfer the SD card to the Z-turn board. Power the Z-turn board, and change the DIP switches' positions repeatedly and notice the combination of the RGB LEDs together with the Buzzer. Enjoy!

Page **10** of **10** 

# <u>XADC – The Analogue to Digital Converter Block within the 7 Series FPGAs</u> and the Zynq 7

ADC stands for Analogue to Digital Conversion. This means that an analogue voltage is sampled, and this analogue voltage is represented by a decimal number. The decimal number is not infinite because it is restricted by the number of bits. Most common microcontrollers have 10-bit ADC peripherals however the Zynq 7 has a 12 -bit ADC and therefore the input voltage can be represented by a decimal number from 0 to 4095. The recommended reference voltage by Xilinx for the analogue inputs is 1.25V, however on the Z-turn board, the external reference voltage XADC\_VCC is 1.8V obtained via an inductor which suppresses further any noise on XADC\_VCC supply rail. Strictly speaking, this means that the analogue signal applied to any one of the analogue inputs should not exceed 1.8V! However, it is recommended by the author, not to exceed the analogue input voltage by more than 1V.

The Zynq 7 SoC has internal parameters that could be sampled via the internal XADC. These might need to be monitored if the Zynq 7 is part of a critical system to make sure that the Zynq 7 is operating within its parameters. These internal parameters include but is not limited to die-temperature, Auxiliary Vcc which is the reference voltage for the auxiliary XADC channels, etc, etc. One can check Xilinx UG480 for more information on this. Therefore, as a first XADC project, it would be ideal to check these internal parameters, and in subsequent projects, one will endeavour to explain more complex projects using XADC block.

Apart from checking the internal parameters, the Zynq 7 has one 12-bit channel which is able to sample at an impressive rate of 1 Mega Sample Per Second (1MSPS). This is called Vp\_0/Vn\_0 in the Xilinx literature such as the UG480, however in the Z-turn board's cape IO schematic, these are marked as XADC\_INPO and XADC\_INN respectively. Using the board support files, one does not have to worry about the pinouts because they will be automatically assigned by Vivado, however one has to be careful when reading the schematics by MYIR because they are a bit confusing! In fact as a reference point, one should check the orientation of the 1.8V from the header pins and also the 3V3 header pin. This check will help to determine the orientation of Vp\_0 / Vn\_0 relative to the schematics offered by MYIR.

One other thing that might confuse novices is that Vp and Vn are referred to as two separate channels. This depends on how one sees it, because if the a differential voltage is applied between Vp and Vn, then one might still regard it as a single channel. On the other hand, most of the analogue voltages such as those from analogue sensors are referenced to ground (0V) and therefore Vn must be connected directly to analogue 0V of the signal which might also be the same ground of the digital system....and again the channel would be regarded as single as well!

Apart from this dedicated external 1MSPS channel, the Zynq 7 has another 16 analogue to digital channels referred to as auxiliary channels. In UG480, these are named as VAUXP[0] and VAUXN[0] for channel 0, VAUXP[1] / VAUXN[1] for channel 1, etc. The analogue pins are shared with digital pins and therefore these particular pins are multi-functional. They are also differential type and therefore one can apply two separate voltages on  $AuxVp_n$  /  $AuxVn_n$ , and the internal op-amp will do the subtraction between the input voltages. The second **n** (highlighted in **italic-bold**) present the **number** of the auxiliary analogue input pair. These channels sample at a maximum rate of 250 kilo-Samples per Second, and they are four times slower

than the dedicated ADC input. Having said that, it still has a relatively high sampling rate. From Chapter 9 onwards, this book will discuss the external analogue channels in more detail, but for now let's move on to see how to sample the internal parameters of the Zynq 7.

# Sampling Zynq 7 internal parameters

In this project, the internal parameters of the Zynq 7 are sampled and read from the Processing System. These are then output on LEDs to validate the system's operation by making sure that the digital result is changing with every change in the input analogue voltage. For this test, the dev-board designed by the author was used on top of the Cape IO board by MYIR. This dev-baord was created to compliment the Z-turn board. It has two  $10K\Omega$  preset-potentiometers and two  $5k\Omega$  preset-pots, together with four more DIP switches, four push-to-make switches, a single seven-segment display and a set of 18 SMT LEDs. All of these components are completely isolated from the Zynq 7.

As always, start by creating a new Vivado project. For this project there is no need to create a VHDL file and therefore skip those steps. When Vivado IDE is opened on the project, one can create a block design and add the Zynq Processing System. Since the system is going to output the decimal equivalent of the analogue input signal on LEDs, one can include the AXI interconnect block and AXI GPIO block as part of the schematics.

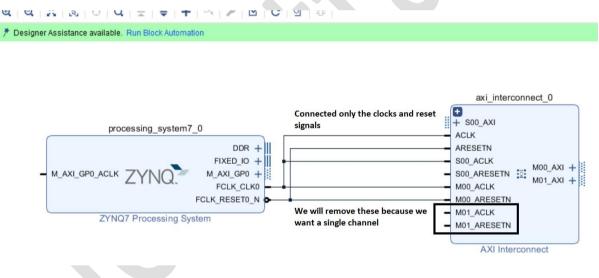


Figure 8. 1: configure the AXI interconnect block

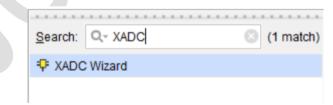
The AXI interconnect block is configured to accept two AXI slave blocks (even though they are called M01...see Figure 8.1), in this project only one is needed so double click on the AXI interconnect block and the configuration window pops up shown in Figure 8.2.

omponent Name	xi_interconnect_0				
op Level Settings	Slave Interfaces	Master Interfaces			(1) Double click on the block
lumber of Slave Int	erfaces	1	~	(2)	ۍ ا
lumber of Master Ir	iterfaces	2	~ <	change this to 1	ki_interconnect_0
nterconnect Optimi:	zation Strategy	Custom	~		AXI
					TN
(I Interconnect incl	-	tomatic converter insertior	-		LK MOD AXL
		the interfaces of the AXI In			RESETN 2 MOLAXI
	or protocol, a conven	rator's parameter propag			DLK MOT_PORT
in width, clock	P is inserted, IP integ				
in width, clock If a converter I	IP is inserted, IP integ e converter to match th		aron aatomaa	-	RESETN
in width, clock If a converter I configures the To see which	e converter to match th conversion IPs have I	he design. been inserted, use the IP	integrator	-	CLK
in width, clock If a converter I configures the To see which	e converter to match th conversion IPs have I	he design.	integrator		
in width, clock If a converter configures the To see which 'expand hiera	e converter to match th conversion IPs have I rchy' buttons to explore	he design. been inserted, use the IP	integrator ect hierarhcy.	address editor.	CLK

Change the number of Master interfaces to 1 and leave the page as it is. Click on OK. This is how the diagram looks like now:

₱ Designer Assistance available. Run Block Automation	
processing_system7_0	axi_interconnect_0
- M_AXI_GP0_ACLK ZYNQ - M_AXI_GP0_ACLK ZYNQ - M_AXI_GP0 + H FICLK_CLK0 FCLK_RESET0_N - ZYNQ7 Processing System	ACLK ARESETN S00_ACLK S00_ARESETN M00_ACLK M00_ARESETN AXI Interconnect
Figure 8. 3 Zyng 7 connected to AXI interconnect	

then include the XADC block:



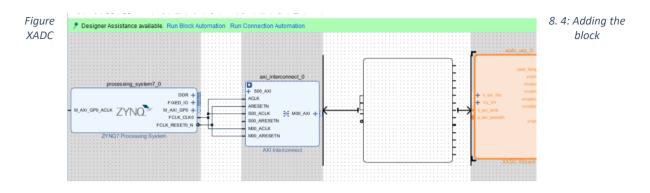


Figure 8. 5: blocks in the block design can be moved

Figure 8.5 shows that blocks can be moved even though with some restrictions. One can left click on the object block and hover the mouse over the canvas for the object block to move along with the mouse.

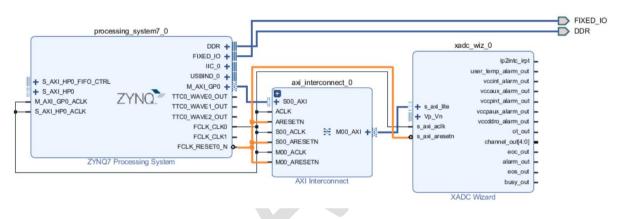


Figure 8. 6: connect the resets

Connect the reset pins of all the blocks together. A better solution is to use the Processing System Reset block as shown in previous chapters.

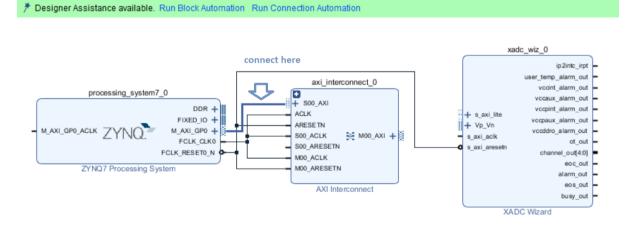




Figure 8.7 show how the data from the Processing System is connected to the AXI interconnect block.

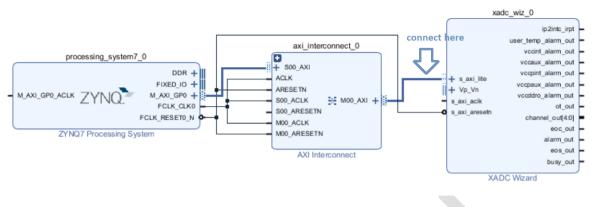
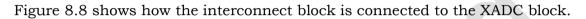


Figure 8. 8: Connect data bus 2



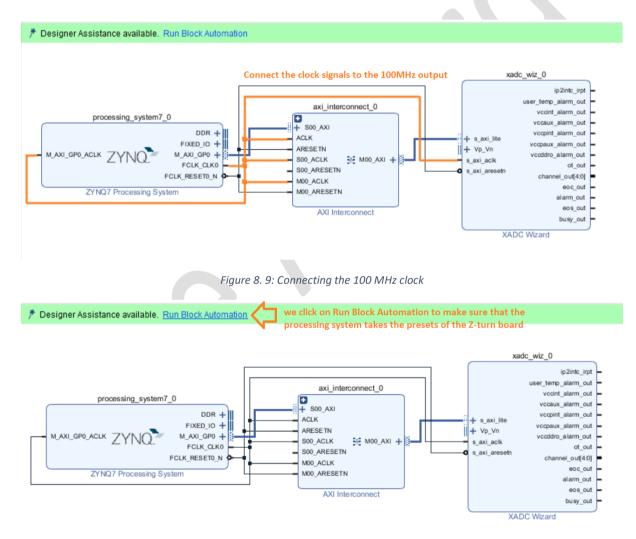


Figure 8. 10: Run Block Automation

Click on *Run Connection Automation* so that the Zynq Processing System takes the pre-set configuration assigned in the Board Support Files.

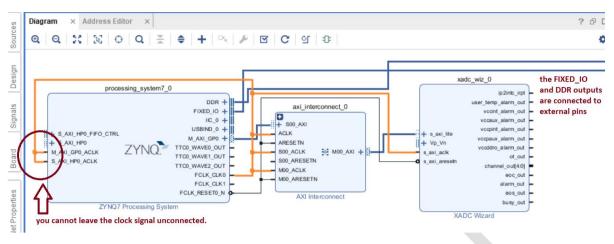


Figure 8. 11: Full Block diagram of the Zynq Processing System

Note that in Figure 8.11 the M\_AXI\_GP0\_ACLK and S\_AXI\_HP0\_ACLK are connected. These two clocks must be connected to the same clock signal as the whole system otherwise Vivado will generate and error.

# Configuring the XADC to sample internal parameters

Show disabled	l ports	Compo	nent Name xado	c_wiz_0			
		Basic	ADC Setup	Alarms	Channel Sequenc	er Summary	
		Interfa	ace Options				Timing Mode
	ip2intc_irpt - user_temp_alarm_out - vccint_alarm_out -		● AXI4Lite 🔾		None		<ul> <li>Continuous</li> </ul>
+ s_axi_lite	vccaux_alarm_out = vccpint_alarm_out = vccpaux_alarm_out =	Startu	p Channel Selec	tion			DRP Timing Options
II + Vp_Vh - s_axi_aclk -o s_axi_aresetn	vccddro_alarm_out – ot_out –		Simultaneou	s Selection	ı		Enable DCL
	channel_out[4:0] = eoc_out =		Independent	ADC			DCLK Frequency
	alarm_out = eos_out =		Single Chanr	nel	[	Selects the opera	ating mode of XADC sion
	busy_out -		Channel Seq	uencer			Acquisition Time
	_						Clock divider valu

Figure 8. 12: XADC basic tab

Double-click on XADC wizard and **change** the setting from **single channel** to **channel sequencer**. The channel sequencer is selected so that the XADC block will hop from one parameter to the next. The ADC result will be stored in respective status registers and therefore when one would like an ADC result of a particular parameter, the latest result will be retrieved.

Note at this point that the result resides between bit 4 and bit 16 of the status register and therefore this must be shifted to the right by four places to obtain the right binary weights of the bits.

		Basic         ADC Setup         Alarms           Sequencer Mode         Continuous	Channel Sequencer	vou can choos
	ip2intc_irpt -	ADC Calibration	Supply Sens	None
user_temp_alarm_out vocint_alarm_out vccaux_alarm_out + s_axi_lite vccpint_alarm_out - vccaux_alarm_out + vp_vh vccpaux_alarm_out - s_axi_aclk vccddro_alarm_out o s_axi_aresetn channel_out[4:0] e eoo_out alarm_out	ADC Offset Calibration ADC Offset and Gain Cali C Enable CALIBRATION Averagi	bration Ser	16     the ADC will taken and the approximation       nsor Offs     64       256     an average restriction	
	alam_out -	External Multiplexer Setup		
	_	External Multiplexer Setup if we are going to use more ADC External Multiplexer	inputs we tick here	This shows that i am using the
	alarm_out - eos_out -	if we are going to use more ADC	inputs we tick here VP VN	This shows that i am using the 1MSPS channel and not the AUX channels
	alarm_out - eos_out -	if we are going to use more ADC	VP VN	1MSPS channel and not the AUX channels to

Figure 8. 13: ADC setup tab

For the ADC setup page, it would be a good idea to configure the XADC block to sample at least 16 times before the result is available to the user. Leave the rest of the settings as they are.

Show disabled po	orts	Component I	Name xadc_wiz_	0				
		ľ		fr	om here you set	the alarms		
		Basic AD	C Setup Alarn	ns Cha	annel Sequencer	Summary		
		✓ Over T	emperature Alarm	n (°C)		🗸 User T	emperature A	larm (°C)
	ip2intc_irpt - user_temp_alarm_out - vccint_alarm_out -	Trigger	125.0	$\otimes$	[-40.0 - 125.0]	Trigger	85.0	(
	vccaux_alarm_out = vccpint_alarm_out =	Reset	70.0	$\otimes$	[-40.0 - 125.0]	Reset	60.0	(
<pre>+ s_axi_lite + vp_vh s_axi_aclk s_axi_aresetn</pre>	vccpaux_alarm_out vccddro_alarm_out ot_out = channel_out[4:0] =		T Alarm (Volts)			VCCAL	JX Alarm (Volt	ts)
	eoc_out = alarm_out = eos_out =	Lower	0.97	$\otimes$	[0.0 - 1.05]	Lower	1.75	6
	busy_out -	Upper	1.03	$\otimes$	[0.0 - 1.05]	Upper	1.89	e

Figure 8. 14: ADC Alarms Tab

The Alarms could be switched off my removing the tick from the boxes.

Show disabled ports		Component Name	by ticking the which parame xadc wiz 0 confirmed yet	ator i w				
		Basic ADC Set	up Alarms (	Channe	I Sequencer	Sumr	nary	
			Channel En	able	Average Ena	able	Bipolar	Acquisition Time
		CALIBRATION				1		
ip2into_impt - user_temp_alarm_out - vccint_alarm_out - vccaux_alarm_out -	TEMPERATURE							
	VCCINT							
+ s_axi_lite	vccpint_alarm_out -	VCCAUX						
II + ∨p_∨h - s_axi_aolk	vccddro_alarm_out - ot_out -	VCCBRAM						
-o s_axi_aresetn	channel_out[4:0] = eoc_out =	VCCPINT						
	alarm_out = eos_out =	VCCPAUX						
	busy_out -	VCCDDRO						
		VP/VN	$\checkmark$					
		VREFP						
		VOEDN						

Figure 8. 15: ADC Channel Sequencer Tab

Figure 8.15 shows a list of ADC channels that one can include in the sequencer settings. At that time the author was just testing so to correct the statements done in Figure 8.15, the internal parameters that must be sampled, could be ticked in the list and these will be enabled. However, it must also be said that even though in Figure 8.15 the internal parameters were not selected, they could still be sampled from the processing system.

Create a hardware wrapper.

After the hardware wrapper, the hardware is exported by File $\rightarrow$  Export  $\rightarrow$  Export Hardware. Then launch SDK from the **File menu**.

# The Software



Figure 8. 16: List of libraries in SDK

After launching SDK, do not forget the create a FSBL application and a Hello World application. The XADC library is located at the very bottom of the list of folders as shown Figure 8.16. Double click on it to find all the support files for this XADC block.

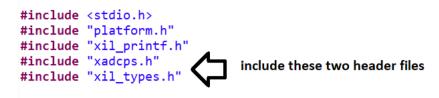


Figure 8. 17: Include Directives

Initialize the XADC peripheral using the following instructions:

From: *xadcps\_sinit.c* get the lookup function:

#### XAdcPs\_Config \*XAdcPs\_LookupConfig(u16 DeviceId)

This function returns an **XAdcPS\_Config** type and therefore one must declare it at the beginning of the main function.

# XAdcPs\_Config \*XADC\_ConfigPtr;

The **u16** DeviceId is listed in xadcps\_g.c and in the parameter-list it should be replaced by: XPAR\_PS7\_XADC\_0\_DEVICE\_ID. So, the complete statement should look like:

# XADC\_ConfigPtr=XAdcPs\_LookupConfig(XPAR\_PS7\_XADC\_0\_DEVICE\_ID);

Another part of the initialization process is the configuration function that is found in *xadcps.c* file. This looks like this:

int XAdcPs\_CfgInitialize(XAdcPs \*InstancePtr, XAdcPs\_Config \*ConfigPtr,

u32 EffectiveAddr)

therefore, it returns an integer type and one must replace the instance pointer with **&XAdcPs**-instance-pointer-name. The above statement is written as:

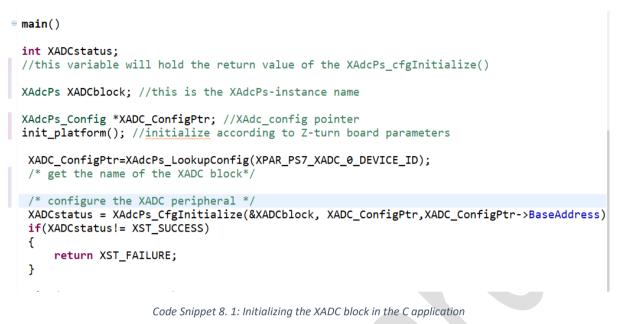
# XADCstatus=XAdcPs\_CfgInitialize(&XADCblock,XADC\_ConfigPtr,XADC\_Config Ptr->BaseAddress);

To avoid unnecessary warnings, check whether the configuration initialization was successful or not by:

#### *If (XADCstatus!= XST\_SUCCESS)*

```
{
    return XST_FAILURE;
}
```

The statement-listing up till now is:



There are some statements that are unique to the XADC initialization. This is one of them:

The **self-test function** will check whether there are any problems with the XADC by resetting the device, then writes a value in the Alarm Threshold Register, then resets the device again. This function is found in *xadcps\_selftest.c* and returns a value of type *int*.

# int XAdcPs\_SelfTest(XAdcPs \*InstancePtr);

# selfteststatus = XAdcPs\_SelfTest(&XADCblock);

*selfteststatus* is a variable of type *int* that stores the returned value of the self-test function.

```
if (selfteststatus != XST_SUCCESS)
```

return XST\_FAILURE;

}

{

# this will suppress the warning that we are not using the *int* variable

Next, stop the channel sequencer by selecting the mode to be as single channel mode. The function resides in *xadcps.c* and is:

# void XAdcPs\_SetSequencerMode(XAdcPs \*InstancePtr, u8 SequencerMode)

To select a <u>mode</u> for the parameter **u8 SequencerMode**, there is a list in **xadcps.h** file.

# #define XADCPS\_SEQ\_MODE\_SINGCHAN 3 /\*\*< Single channel -No Sequencing \*/</pre>

```
* #define XADCPS_SEQ_MODE_SAFE 0 < Default Safe Mode >
#define XADCPS_SEQ_MODE_ONEPASS 1 < Onepass through Sequencer >
#define XADCPS_SEQ_MODE_CONTINPASS 2 < Continuous Cycling Sequencer >
#define XADCPS_SEQ_MODE_SINGCHAN 3 < Single channel -No Sequencing >
#define XADCPS_SEQ_MODE_SIMUL_SAMPLING 4 < Simultaneous sampling>
#define XADCPS_SEQ_MODE_INDEPENDENT 8 < Independent mode > */
```

Code Snippet 8. 2: Definintion statements found in xadcps.h file

The next thing is to disable the alarms by using the function

#### void XAdcPs\_SetAlarmEnables(XAdcPs \*InstancePtr, u16 AlmEnableMask)

found in *xadcps.c* file.

XAdcPs\_SetAlarmEnables(&XADCblock, 0x0000); //0 = disables alarm ; 1 = enables alarm

Restart the sequencer and make it sample internal parameters such as Zynq 7 temperature, VCCINT etc.

#### XAdcPs\_SetSequencerMode(&XADCblock, XADCPS\_SEQ\_MODE\_SAFE);

Select the channels to sample:

The function is found in *xadcps.c* file.

#### int XAdcPs\_SetSeqChEnables(XAdcPs \*InstancePtr, u32 ChEnableMask)

and the parameters are found in *xadcps\_hw.h* file.

Important Note:

Each function carries with it a description and in the description, there will be a note indicating from where the parameters can be selected giving the name of the header file and also how the parameters are named. An example follows:

```
∍/**
 * This function enables the specified channels in the ADC Channel Selection
 * Sequencer Registers. The sequencer must be disabled before writing to these
 * regsiters.
 * @param
         InstancePtr is a pointer to the XAdcPs instance.
 * @param
         ChEnableMask is the bit mask of all the channels to be enabled.
 *
       Use XADCPS SEO CH * defined in xadcps hw.h to specify the Channel
 *
       numbers. Bit masks of 1 will be enabled and bit mask of 0 will
 *
       be disabled.
 *
       The ChEnableMask is a 32 bit mask that is written to the two
 *
        16 bit ADC Channel Selection Sequencer Registers.
 *
 * @return
        - XST_SUCCESS if the given values were written successfully to
        the ADC Channel Selection Sequencer Registers.
 *
        - XST_FAILURE if the channel sequencer is enabled.
 * @note
           None
```

There are some ready-made **macros** that one can use that convert raw ADC data into the quantity one would like to measure such as the internal temperature of the processor etc. These macros are found in **xadcps.c** file.

# u16 XAdcPs\_GetAdcData(XAdcPs \*InstancePtr, u8 Channel);

```
⊜ /**
* Get the ADC converted data for the specified channel.
* @param
        InstancePtr is a pointer to the XAdcPs instance.
         Channel is the channel number. Use the XADCPS_CH_* defined in
* @param
      the file xadcps.h.
       The valid channels are
       - 0 to 6
       - 13 to 31
  @return A 16-bit value representing the ADC converted data for the
      specified channel. The XADC Monitor/ADC device guarantees
       a 10 bit resolution for the ADC converted data and data is the
       10 MSB bits of the 16 data read from the device.
* @note
         The channels 7,8,9 are used for calibration of the device and
     hence there is no associated data with this channel.
ou16 XAdcPs_GetAdcData(XAdcPs *InstancePtr, u8 Channel)
```

Code Snippet 8. 3: ADC result function

Declare a variable of type **16** bit unsigned (u16) at the beginning of the main function. Then use it to store the return variable of the **XAdcPs\_GetAdcData()**.

The returned data from this function is just a decimal number from 0 to 4095 since it is 12 bits wide. This number must be processed again to reflect the analogue quantity it is monitoring. There are two macros that use 32-bit unsigned numbers, and therefore the returned number from the above function will be stored in a 32-bit number not in a u16 data type!!

The instance pointer is always the one declared somewhere above:

**&XADCblock** and the **u8 Channel** is taken from **xadcps.h** Underneath there is a list of u8 channels:

Code Snippet 8. 4: ADC Channel List

The macro that converts the 12-bit ADC data into temperature resides in *xadcps.h*. From the comments that accompany it, it was determined that it returns a *float* type.

Code Snippet 8. 5: Raw to Temperature Macro

Note that AdcData is of type **u32** while the returned data from the previous function XAdcGetAdcData() is of type u16. However, this might not impose a problem since the data will be stored in the correct word position within the 32-bit word.

Because the above macro accepts **u32** data, the **print()** already written in the helloworld program had to be changed to **printf()** because the 32-bit raw data is not supported in the print() function. On the other-hand, the **printf()** supports **%lu** which means **unsigned long data** type.

The whole software program is listed below:

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xadcps.h"
#include "xil_types.h"
void delay (void);
int main()
{
     int XADCstatus;
     //this variable will hold the return value of the XAdcPs cfgInitialize()
     int selfteststatus;
     // this varialbe holds the return value of the self test function
     int SeqEnable;
     /*this variable holds the return value for the seq_enables ()*/
    XAdcPs XADCblock; //this is the XAdcPs-instance name
    XAdcPs_Config *XADC_ConfigPtr; //XAdc_config pointer
     init_platform(); //initialize according to Z-turn board parameters
     u32 rawCPU temp, rawintVCC, rawBRAMvoltage, rawAUXVCC;
    float CPUtemp,intVCC,BRAMvoltage,AUX_VCC;
XADC_ConfigPtr=XAdcPs_LookupConfig(XPAR_PS7_XADC_0_DEVICE_ID);
/* get the name of the XADC block*/
/* configure the XADC peripheral */
XADCstatus = XAdcPs_CfgInitialize(&XADCblock, XADC_ConfigPtr,XADC_ConfigPtr->BaseAddress);
if(XADCstatus!= XST_SUCCESS)
{
    return XST_FAILURE;
}
/*Run a self test by resetting the device, write a name in the Alarm Threshold
 * Register and resetting the device again*/
selfteststatus = XAdcPs_SelfTest(&XADCblock);
if(selfteststatus != XST SUCCESS)
{
    return XST_FAILURE;
}
```

```
while(1)
      rawCPU temp = XAdcPs GetAdcData(&XADCblock, XADCPS CH TEMP);
      CPUtemp = XAdcPs_RawToTemperature(rawCPU_temp);
      printf("raw temp data: %lu while temp in degree Celcius: %f\n\r",rawCPU_temp,CPUtemp);
      rawintVCC = XAdcPs_GetAdcData(&XADCblock, XADCPS_CH_VCCINT);
      intVCC = XAdcPs_RawToVoltage(rawintVCC);
      printf("raw intVCC data: %lu while internal VCC: %f\n\r",rawintVCC,intVCC);
      rawBRAMvoltage = XAdcPs_GetAdcData(&XADCblock, XADCPS_CH_VBRAM);
      BRAMvoltage = XAdcPs_RawToVoltage(rawBRAMvoltage);
          printf("raw BRAM voltage: %lu while BRAM voltage: %f\n\r",rawBRAMvoltage,BRAMvoltage);
          rawAUXVCC = XAdcPs_GetAdcData(&XADCblock, XADCPS_CH_VCCAUX);
          AUX_VCC = XAdcPs_RawToTemperature(rawAUXVCC);
          printf("raw Aux VCC: %lu while Aux VCC: %f\n\r",rawAUXVCC,AUX_VCC);
      printf("\r\n");
printf("\r\n");
          delay();
     }
    cleanup platform();
    return 0;
}
void delay (void)
{
        for(unsigned i = 0; i < 100000000; i++)</pre>
        {
                //do nothing
        }
}
```

It must be mentioned here that the raw data from the XADC should be shifted to the right by 4 places. This is not shown in this program however, it will be pointed out in the future programs.

#### Chapter 9 Sampling External ADCs from the Processing System

The previous chapter dealt with sampling the internal parameters of the System on Chip. As promised, this chapter will show how to monitor external analogue inputs from the Processing System. Later, the same will be done, but this time from the Programmable Logic.

So, create a project as shown in previous chapters. Do not include any VHDL modules. Once the Vivado project is opened, click on *Create a Block Design*. In the block design include the Zynq Processing System, the AXI interconnect and the XADC wizard.

The AXI interconnect should be configured to have one master output as shown in Figure 9.1 below.

Re-customize IP					×
(I Interconnect (2	2.1)				4
Documentation 📑 I	P Location				
Component Name a	xi_interconnect_0				
Top Level Settings	Slave Interfaces	Master Interfaces			
Number of Slave Interfaces		1	~	(1) double Click on the interconnect block (2) change the number of Masters to 1	¢ î
Number of Master Interfaces		2	~	(3) hit OK	
Interconnect Optimization Strategy		1	^		- 11
		2			

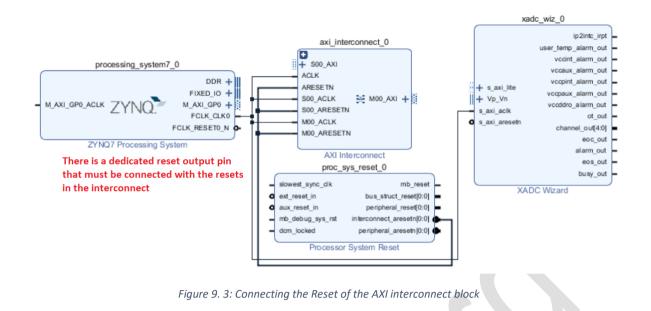
Figure 9. 1: Configure the AXI interconnect block

This time, the Processing System Reset Block will be added in the design. This will help reduce the warning and errors.

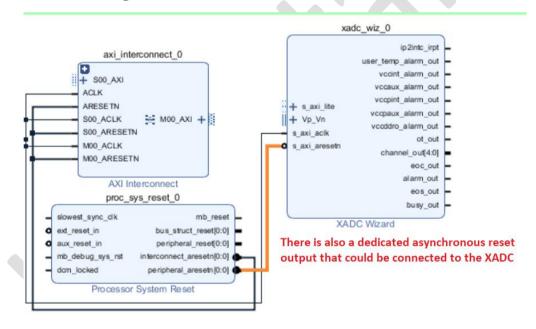
<ul><li>(1) right click agai</li><li>(2) write "reset" i</li><li>(3) double click or</li></ul>	n field	anvas sor System Reset"		XADC Wizard	1
	<u>S</u> earch:	Q- reset		(2 matches)	
	Proce	essor System Reset			
	👎 Simu	lation Reset Generato	r		

Figure 9. 2: Including the Processing System Reset Block

Now connect the resets of the AXI interconnect to the dedicated reset on the Processing System Reset block as shown in Figure 9.3 on the next page.

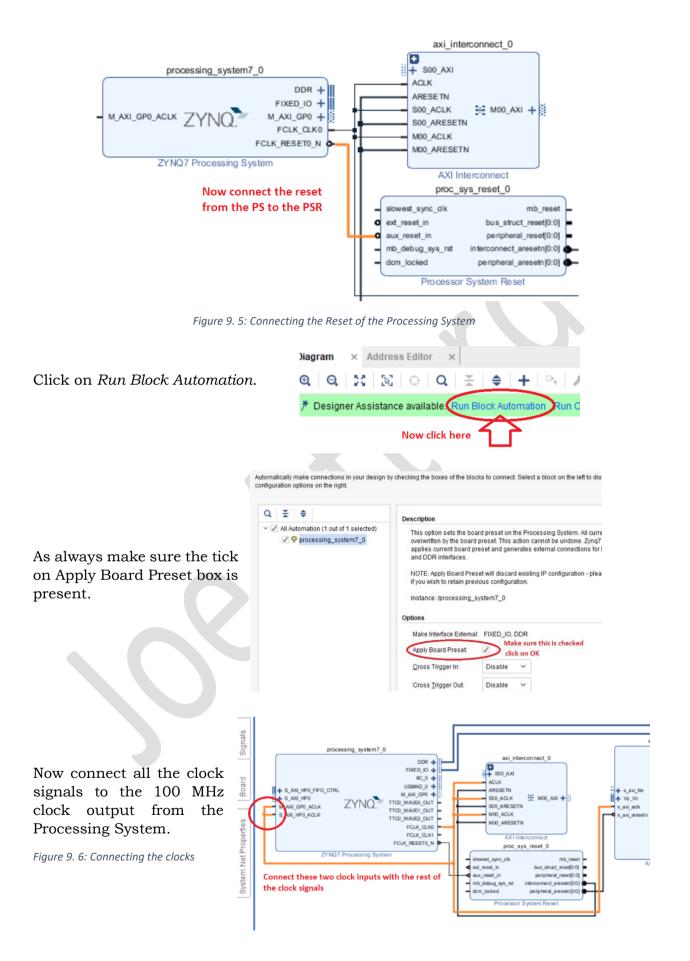


Now, connect the reset of the XADC wizard block to the Processing System Reset block as shown in Figure 9.4 below.





Now connect the Reset of the Processing System to the Reset block as shown in Figure 9.5 on the next page.



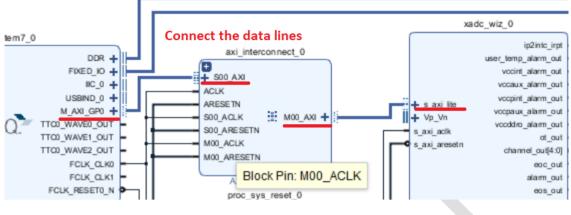


Figure 9. 7: Connecting the Data lines

Figure 9.7 shows how to connect the data lines between the Zynq Processing System, the AXI interconnect block and the XADC wizard.

The next thing to do is to configure the XADC wizard, so double click on the XADC block and follow the instructions in the Figures underneath.

Show disabled ports	Compo	onent Name xad	lc_wiz_0		
	Basic	ADC Setup	Alarms	Channel Sequencer	Summa
	Interf	face Options			
ip2into_ipt - user_temp_alarm_out - vocint_alarm_out -		O AXI4Lite ○	DRP 🔘	None	
vccaux_alarm_out - + s_axi_lite vccpint_alarm_out - vccpaux_alarm_out -	Startu	up Channel Sele	ction		
p_vn vccddro_alarm_out — xi_aolk ot_out —		O Simultaneou	us Selection	n	
channel_out[4:0] = eoc_out =					
alarm_out = eos_out = busy_out =		Single Chan			
busy_out		Channel See		.) select the channel s iode	equencer
	AXI4S	STREAM Options			
Figure 9.	8: Selec	ct Channe	el Sequ	lencer	
	8: Selec	ct Channe	el Sequ	lencer	
Figure 9.	8: Selec	ct Channe	el Sequ	encer	
	8: Selec		el Sequ		this to 16
Component Name xadc_wiz_0	Channel Sec		nmary		this to 16
Component Name xadc_wtz_0 Basic ADC Setup Alarms	Channel Sec	quencer Sun	nmary ing 16		this to 16
Component Name xadc_wiz_0 Basic ADC Setup Alarms Sequencer Mode Continuous	Channel Sec	quencer Sun Channel Averag Supply Sensor	nmary ing 16	change	this to 16
Component Name xadc_wtz_0 Basic ADC Setup Alarms Sequencer Mode Continuous ADC Calibration	Channel Sec	quencer Sun Channel Averag Supply Sensor ( Sensor	ing 16 Calibration	change	this to 16
Component Name xadc_wiz_0 Basic ADC Setup Alarms Sequencer Mode Continuous ADC Calibration	Channel Sec Channel Sec bration	quencer Sun Channel Averag Supply Sensor ( Sensor	ing 16 Calibration		this to 16
Component Name xadc_wiz_0 Basic ADC Setup Alarms Sequencer Mode Continuous ADC Calibration ADC Offset Calibration ADC Offset and Gain Cali	Channel Sec Channel Sec bration	quencer Sun Channel Averag Supply Sensor ( Sensor	ing 16 Calibration		this to 16
Component Name xadc_wiz_0 Basic ADC Setup Alarms Sequencer Mode Continuous ADC Calibration ADC Offset Calibration ADC Offset and Gain Cali C Enable CALIBRATION Average External Multiplexer Setup	Channel Sec Channel Sec bration	quencer Sun Channel Averag Supply Sensor Sensor Ø Sensor	ing 16 Calibration		this to 16
Component Name xadc_wiz_0 Basic ADC Setup Alarms Sequencer Mode Continuous ADC Calibration ADC Offset Calibration ADC Offset and Gain Cali C Enable CALIBRATION Averagi External Multiplexer Setup	Channel See	quencer Sun Channel Averag Supply Sensor Sensor Ø Sensor	ing 16 Calibration		this to 16

Figure 9. 9: Enable an average of 16

asic ADC Se	tup	Alarms	Chann	el Sequencer	Summa	гу	
		Channel E	Inable	Average En:	able	Bipolar	Acquisition
CALIBRATION				in this page i			
TEMPERATURE				we will read the internal tem the BRAM voltage		nal tempei	perature
VCCINT VCCAUX VCCBRAM				external analogue input Vp Vaux1 and Vaux 8 because they are			
					near each other		
VCCPINT							
VCCPAUX							
VCCDDRO VP/VN							
		$\checkmark$	]				
VREFP			]				
VREFN			]				

Figure 9. 10: Selecting the Channels

Figure 9.10 shows that the channels of interest must be selected from the list. The Auxiliary pins are not shown in Figure 9.10 but they are selected. Now close the XADC wizard configuration. Click on *Run Block Automation*.

	@   Q   X   ∑   ⊕   Q   X   €   +   ▷   ⊁   ⊠   C   ଏ
Ĭ	Designer Assistance available. Run Connection Automation Click here

For the window of Figure 9.11, click on OK.

Q ≚ ≑	Description
<ul> <li>✓ All Automation (1 out of 1 selected)</li> <li>✓ ✓ ♥ proc_sys_reset_0</li> <li>✓ ≫ slowest_sync_clk</li> </ul>	Connect clock-pin ({/proc_sys_reset_0/slowest_sync_clk}) to selected clock source. Also configure and connect clock-pins of connected bridge-IPs(AXI Interconnect, Smartconnect as needed. Also infer Processor System Reset block and connect synchronous reset source to associated reset pin(s) as needed.
	Clock Source: /processing_system7_0/FCLK_CLK0 (100 MHz) V
	just hit OK underneath

Figure 9. 11: Warning Box

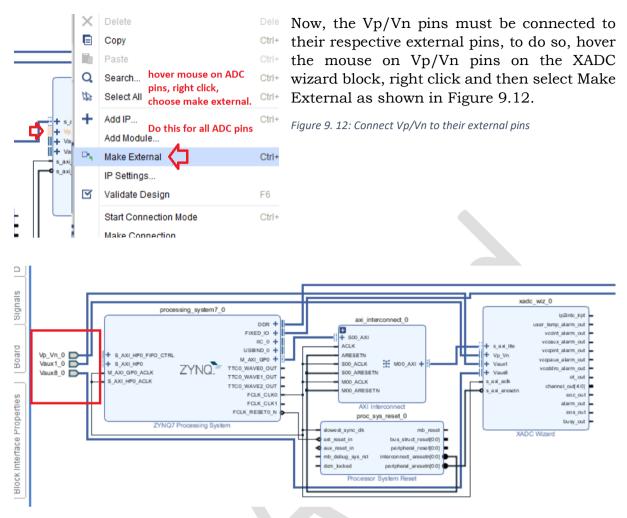


Figure 9. 13: Showing the External Pins of the ADCs

Note in Figure 9.13 that the auxiliary channels 1 and 8 have been enabled together with the dedicated Vp/Vn ADC. It would be a good idea if the design is validated.

	Validate Design (F6)	
Critical Messages	×	Fig
There were two critical warning messages while validating this d lessages	esign.	If di
<ul> <li>[BD 41-1356] Address block  is not ma . Please use Address Editor to eithe it.</li> <li>[BD 41-1356] Address block  is not ma</li> </ul>	r map or exclude	in Eo lo
. Please use Address Editor to eithe it.	r map or exclude	T1 01
<b>~</b>		

#### Figure 9. 14: Critical Warnings

If the warning in Figure 9.14 are displayed, then follow their instructions and use the address Editor to assign memory locations to the XADC block. This is illustrated in Figure 9.15 on the next page.

	4					
BLC	OCK DESIGN - block_design *					
ses	Diagram × Address Editor	×				
Sources	Q ≚ ≑ 🖻	_				
0	Cell	Slave Interface	Base Name	Offset Address Ran		
E	✓ ₱ prpcessing_system7_0		2000110.00			
Design	<ul> <li>Data (32 address bits : 0)</li> </ul>	40000000[1G])	Use the drop dov			
	<ul> <li>Unmapped Slaves (1)</li> </ul>	40000000[10])	arrows on the lef select the xadc w			
2	m xadc_wiz_0	s_axi_lite		12410		
Signals	<ul> <li>xauc_wiz_o</li> <li>Unconnected Slaves</li> </ul>	S_axi_iite	Reg			
<u> </u>						
73	processing_system7_0	S_AXI_HPU	HP0_DDR_LOWOCM			
ard	1					
	Figure 9. 15	: Assigning an addre	ss to the XADC block			
	<ul> <li>Data (32 address bits : 0x4000)</li> </ul>	0000 [ 1G ])	at all a set of the			
	<ul> <li>Unmapped Slaves (1)</li> </ul>		right click choose Assig	n Address		
	- xadc_wiz_0 s_ax	i_lite Reg				
~	Unconnected Slaves		Assign	Address		
		I_HP0 HP0	DDR_L	lies		
			Unmaj	Segment		
	Figure 9.	16: Choose the Assign	n Address from list			
Cell	Slave Interf	ace Base Name	Offset Address	Range High Address		
~ ₽	processing_system7_0					
~	🖽 Data (32 address bits : 0x40000000 [ 1	G ])				
	🚥 xadc_wiz_0 s_axi_lite	Reg	0x43C0_0000	6 • 0x43C0_FFFF		
× 🗎	Unconnected Slaves					
	m processing_system7_0 S_AXI_HP0	HP0_DDR_LC	WOCM			

Figure 9. 17: The XADC block is assigned an address

Now it is a good idea to validate the design from the Block Design Menu.



Figure 9. 18: Validation Successful message

Now create a Hardware Wrapper.

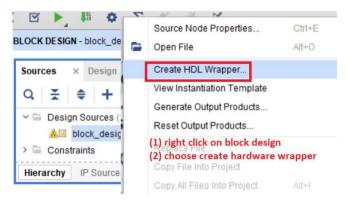


Figure 9. 19: Steps to Create a Hardware Wrapper

Sources ×	Design	Signals B	oard ? _ U
Q	+	? 0	Updating 🥥
	Sources (1) ock_desigr ints	) (block_design	.bd)

Figure 9. 20: Making sure that updates are done

Figure 9.20 shows an important step. If the updates are not done and the synthesis begins, Vivado will generate an error. So, make sure that the *Updating* message disappears before clicking to synthesize and implementation.

lame	Direction	Board Part Pin	Board Part Interface	Neg Diff Pair	Package Pin		Fixed	Bank	I/O Std
🐼 All ports (136)									
> 🗟 DDR_6075 (71)	INOUT						$\checkmark$	502	(Multiple)*
> 🗟 FIXED_IO_6075 (59)	INOUT						$\checkmark$	(Multiple)	(Multiple)*
<ul> <li>Vaux1_0_6075 (2)</li> </ul>	IN			The pi	nouts tally		<ul> <li>Image: A start of the start of</li></ul>	35	default (LVCMOS
Y 🐼 Scalar ports (2)	(1) open th	e implemented des	lan	with t	he schematics				
▷ Vaux1_0_v_n		the boxes as shown			D18	~	$\checkmark$	35	default (LVCMOS
▷ Vaux1_0_v_p	IN (3) Leave t	he default voltage t	o 1V8		E17	~	$\checkmark$	35	default (LVCMOS
<ul> <li>Vaux8_0_6075 (2)</li> </ul>	IN						$\checkmark$	35	default (LVCMOS
<ul> <li>Scalar ports (2)</li> </ul>									
Image: Vaux8_0_v_n	IN				A20	~	$\checkmark$	35	default (LVCMOS
	IN				B19	~	$\checkmark$	35	default (LVCMOS
Vp_Vn_0_6075 (2)	IN						$\checkmark$	0	
<ul> <li>Scalar ports (2)</li> </ul>									
▷ Vp_Vn_0_v_n	IN				L10	~	$\checkmark$	0	
☑ Vp_Vn_0_v_p	IN				K9	~	$\checkmark$	0	

#### Figure 9. 21: Pinouts

Figure 9.21 is just a check to see that the pinouts assigned automatically by Vivado tally with the schematics. Make sure that the Fixed boxes are ticked. Now click on Generate Bitstream File and wait. If there are no errors and the bitstream file is generated successfully, export the hardware by File  $\rightarrow$  Export  $\rightarrow$  Export Hardware.

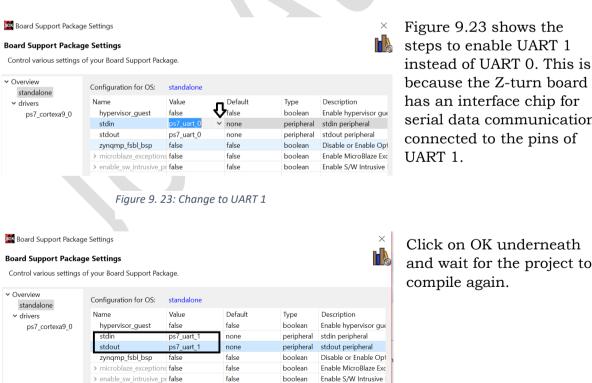
When it is done exporting the hardware, Launch SDK from within the Vivado project.

#### The Software

In SDK, create a new FSBL project and a new Hello World project. These two steps have been shown in previous chapters. For this project, the ADC readings will be shown on a serial terminal so the following BSP adjustments must be made. This has been shown before in previous chapters.

i ps7_init.c ₪ ps7_init.h		Go Into	d Su	
⊛ ps7_init.ŀ		Open in New Window		Spe
🗎 ps7_init.t		Сору	Ctrl+C	get l
🛍 system.h	B	Paste	Ctrl+V	geen
> 🐸 C code	×	Delete	Delete	s Sy:
> Description (Description)		Source (1) right click on the C	>	орон
> SBL_app		Move code project bsp folder		Nan
> 1 FSBL_app_b		Rename	F2	ersic
	24 24	Import (2) Choose Board Support Export Package Settings		iptic
		Refresh	F5	tatic
		Close Project		l Dri
		Close Unrelated Projects		eser
		Build Configurations	>	
		Run As	>	Sour
# <b>-</b>		Debug As	>	
Target Connect		Compare With	>	5 🖾
> 🗁 Hardware S		Restore from Local History		varn
Einux TCF A EMU TcfG	۱ <b>۱</b>	Board Support Package Settings		ngs
	IV.	Re-generate BSP Sources		.95

Figure 9. 22: Open the BSP Setting



has an interface chip for serial data communication connected to the pins of UART 1.

Now open the HelloWorld.c file.

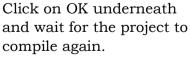




Figure 9. 24: Include the XADC library

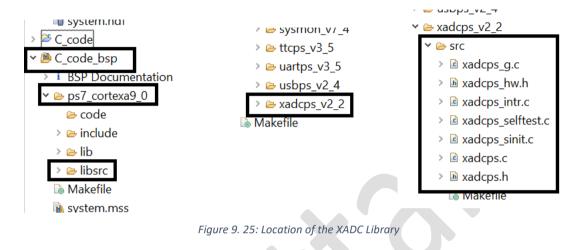


Figure 9.25 shows the location of the XADC library within the C project environment. It also shows the functions associated with the XADC block.

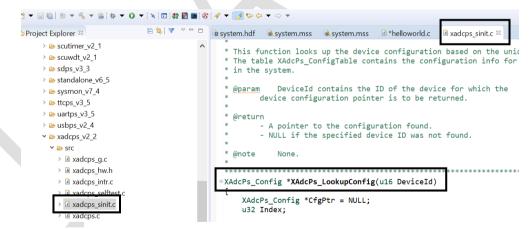


Figure 9. 26: Location of the Lookup function

From the project tree double click on *xadcps\_sinit.c* file. Copy the lookup function name in the helloworld.c file.

#### XAdcPs\_Config \*XAdcPs\_LookupConfig(u16 DeviceId)

Now this function returns an **XAdcPs\_Config** type. Therefore, one must declare a variable at the beginning of the main function and equate this statement to the variable. Also, a parameter of type u16 DeviceId should be passed to this function.

#### This parameter is obtained from:

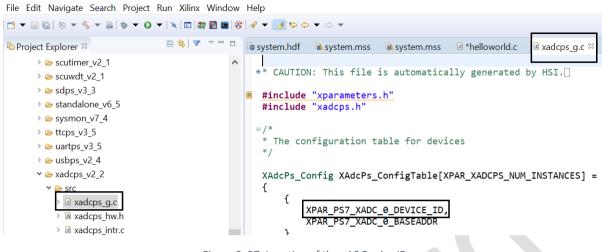


Figure 9. 27: Location of the u16 Device ID

So, the proper statement should be:

#### XADC\_configPtr = XAdcPs\_LookupConfig(XPAR\_PS7\_XADC\_0\_DEVICE\_ID);

File Edit Navigate Search Project	Run Xilinx Window	Help
📑 🕶 🗑 🚱 🖛 🍕 🕶 🙆 🎋 🕶 🖸	- 🗙 🗈 🏶 🗾 🚱	A + 🗾 👳 🗢 + 🔿 +
Project Explorer ≅ > escutimer_v2_1 > escuwdt_v2_1 > escuwdt_v2_1 > estandalone_v6_5 > estandalone_v6_5 > estandalone_v7_4 > ettcps_v3_5 > euartps_v3_5 > eusps_v2_4 ~ exadcps_v2_2	□ \$\$, ▼ ○ □	<pre>system.hdf in system.mss in system.mss in the loworld.c in xadcps.c ≥ Unexpected errors may occur if the address mapping is changed after this function is invoked. "ereturn - XST_SUCCESS if successful. "enote The user needs to first call the XAdcPs_LookupConfig() API which returns the Configuration structure pointer which is passed as a parameter to the XAdcPs_CfgInitialize() API. "***</pre>
<pre>&gt; src &gt; in xadcps_g.c &gt; in xadcps_hw.h &gt; in xadcps_intr.c &gt; in xadcps_selftest.c &gt; in xadcps_sint.c &gt; in xadcps.c &gt; in xadcps.c</pre>	Eigure Q	<pre>int XAdcPs_CfgInitialize(XAdcPs *InstancePtr, XAdcPs_Config *ConfigPtr,</pre>
	rigure 9.	28: Location of the Initialization Function

Also, part of the initialization is the function in Figure 9.28.

It returns an **int** type, therefore this must be declared as a variable at the beginning of the main function. Apart from that, there is an **instance-pointer** of type **XAdcPs**. This must also be declared on top of the main function. The complete statement should look like Code Snippet 9.1:

```
main()
int init_success;
XAdcPs XADCperipheral; Variable declarations
XAdcPs_Config *XADC_configPtr; here
init_platform();

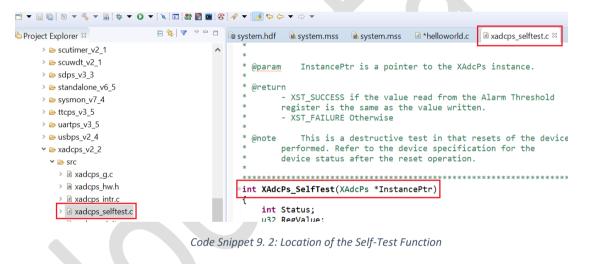
/* Initialise the XADC*/
XADC_configPtr = XAdcPs_LookupConfig(XPAR_PS7_XADC_0_DEVICE_ID);
init_success=XAdcPs_CfgInitialize(&XADCperipheral,XADC_configPtr,XADC_configPtr->BaseAddree
if(init_success != XST_SUCCESS)
{
    return XST_FAILURE; Complete initialization
}
```

Code Snippet 9. 1: Complete Initialization statement

Notice the instance-pointer: &XADCperipheral

## <u>Self-Test</u>

The self-test function is used to reset the XADC and to check whether the XADC is healthy. This will return a variable of type *int* and therefore this must be equated to another variable that should be declared at the beginning of the main function. This function resides in:



```
Code Snippet 9. 3: Writing the Self-test function in
                                                  int init_success,STstatus;
code
                                                  XAdcPs XADCperipheral;
                                                  XAdcPs_Config *XADC_configPtr;
                                                  init_platform();
                                                  /* Initialise the XADC*/
                                                  XADC_configPtr = XAdcPs_LookupConfig(XPAR_PS7_XAI
                                                  init_success=XAdcPs_CfgInitialize(&XADCperiphera.
                                                  if(init_success != XST_SUCCESS)
                                                  {
                                                      return XST_FAILURE;
                                                  }
                                                  /*Self test. This should also reset the XADC*
                                                  STstatus=XAdcPs_SelfTest(&XADCperipheral);
                                                  if(STstatus != XST_SUCCESS)
                                                  ł
                                                      return XST_SUCCESS;
```

ect Explorer 🛛	🖻 🔄 🔻 🖵 🗖	ធ system.hdf 🛯 🗟 system.mss 🖉 *helloworld.c 🛛 🖻 xadcps.c 🛛
<pre>&gt; e scutimer_v2_1 &gt; e scuwdt_v2_1 &gt; e sdps_v3_3 &gt; e standalone_v6_5 &gt; e sysmon_v7_4 &gt; e ttcps_v3_5 &gt; e uartps_v3_5 &gt; e uartps_v2_4 &gt; e xadcps_v2_2 &gt; e xadcps_v2_2 &gt; e xadcps_hw.h &gt; e xadcps_hw.h &gt; e xadcps_intr.c &gt; e xadcps_selftest.c &gt; e xadcps_sinit.c &gt; e xadcps_c. &gt; m xadcps.h</pre>		<pre>** This function sets the specified Channel Sequencer Mode in the Configu * Register 1 : * Default safe mode (XADCPS_SEQ_MODE_SAFE) - One pass through sequence (XADCPS_SEQ_MODE_ONEPASS) - Continuous channel sequencing (XADCPS_SEQ_MODE_CONTINPASS) - Single Channel/Sequencer off (XADCPS_SEQ_MODE_SINGCHAN) - Simulataneous sampling mode (XADCPS_SEQ_MODE_SINUL_SAMPLING) - Independent mode (XADCPS_SEQ_MODE_INDEPENDENT) * @param InstancePtr is a pointer to the XAdcPs instance. * @param SequencerMode is the sequencer mode to be set. * Use XADCPS_SEQ_MODE_* bits defined in xadcps.h. * @return None. * * @note Only one of the modes can be enabled at a time. Please * read the Spec of the XADC for further information about the sequencer modes. *</pre>
🚡 Makefile Makefile		<pre>void XAdcPs_SetSequencerMode(XAdcPs *InstancePtr, u8 SequencerMode)</pre>

Figure 9. 29: Location of the Sequence Function

Figure 9.29 shows the location of the sequencer function. The XADC has to be stopped so that the configuration registers could be written to, to configure the XADC block. The function resides in *xadcps.c.* The parameters are listed in the comments list as illustrated in Figure 9.29.

/\* now we stop the sequencer\*/
XAdcPs\_SetSequencerMode(&XADCperipheral, XADCPS\_SEQ\_MODE\_SINGCHAN);

As the comments in Figure 9.29 show, the XADC sequencer is stopped if the parameter passed to the SetSequence() is *SINGCHAN*.

The XADC should be put into safe mode so that the configuration registers could be changed:

/\* we put the XADC in safe mode so that we can change the configuration registers\*/
XAdcPs\_SetSequencerMode(&XADCperipheral, XADCPS\_SEQ\_MODE\_SAFE);

Code Snippet	9.5:	XADC in	Safe Mode
--------------	------	---------	-----------

: © ▼ ≪ ▼	- 🔍 🖂 🏶 🗾 🐼	
ect Explorer ⊠	🖻 😫 🔷 🗠 🗖	🖻 system.hdf 🛯 🗟 system.mss 🔹 *helloworld.c 🔹 xadcps.c 🛛
<pre>&gt; © scutimer_v2_1 &gt; © scuwdt_v2_1 &gt; © sdps_v3_3 &gt; © standalone_v6_5 &gt; © sysmon_v7_4 &gt; © ttcps_v3_5 &gt; © uatps_v3_5 &gt; © uatps_v2_4 ~ © xadcps_v2_2 ~ © src &gt; © xadcps_g.c &gt; © xadcps_selftest.c &gt; © xadcps_selftest.c &gt; © xadcps_s.h</pre>	^	<pre>/*** * This function enables the alarm outputs for the specified alarms in * Configuration Register 1. * @param InstancePtr is a pointer to the XAdcPs instance. * @param AlmEnableMask is the bit-mask of the alarm outputs to be {     in the Configuration Register 1.     Bit positions of 1 will be enabled. Bit positions of 0 will be     disabled. This mask is formed by OR'ing XADCPS_CFR1_ALM_*_MASF     XADCPS_CFR1_OT_MASK masks defined in xadcps_hw.h. * @return None. * @note The implementation of the alarm enables in the Configurati     register 1 is such that the alarms for bit positions of 1 will     be disabled and alarms for bit positions of 0 will be enabled.     The alarm outputs specified by the AlmEnableMask are negated     before writing to the Configuration Register 1. * </pre>
🚡 Makefile		**************************************

Figure 9. 30: Location of the Alarms Function

Code Snippet 9. 4: Stopping the XADC

The first configuration is to disable the alarms. 0 will disable the alarms while 1 will enable the alarms.

```
/* Disable the alarms*/
XAdcPs_SetAlarmEnables(&XADCperipheral,0x0000);
```

Code Snippet 9. 6: Alarms Function

}  ® ▼ % ▼ 🖬 i 🎋 ▼ O	) 🔻 🔌 🔽 😫 🖉 🖬 🚱 🖋 🔻 🍠 🌣 🗇 🔻 🔿 💌
Explorer 🛛	🖻 😫 🔻 🍷 🗖 🗈 system.hdf 🛚 🗟 system.mss 🔹 system.mss 🗈 *helloworld.c 🛛 🗟 xadcps.c 🕸
<pre>scutimer_v2_1 scutimer_v2_1 scutimer_v2_1 sdps_v3_3 standalone_v6_5 sysmon_v7_4 tcps_v3_5 uartps_v3_5 uartps_v3_5 uartps_v2_4 scutimer_v2_2  </pre>	<pre>/** * This function enables the specified channels in the ADC Channel Se: * Sequencer Registers. The sequencer must be disabled before writing * regsiters. * @param InstancePtr is a pointer to the XAdcPs instance. * @param ChEnableMask is the bit mask of all the channels to be er Use XADCPS_SEQ_CH_* defined in xadcps_hw.h to specify the Cl * numbers. Bit masks of 1 will be enabled and bit mask of 0 will * be disabled. * The ChEnableMask is a 32 bit mask that is written to the two 16 bit ADC Channel Selection Sequencer Registers. * @return *     - XST_SUCCESS if the given values were written successfully 1 the ADC Channel Selection Sequencer Registers. * @note None * **********************************</pre>
akefile	<pre>int XAdcPs_SetSeqChEnables(XAdcPs *InstancePtr, u32 ChEnableMask) </pre>
stem.mss	

Figure 9. 31: Location of Channel Enable Function

Figure 9.31 shows the location where the function to enable the individual channels is located. The comments give a hint on how to identify the parameters that could be passed to this function and their location. This function returns a value of type *int* and therefore this must be declared again at the beginning of the main(). The parameters are listed in the Figure 9.32 and can be found in *xadcps\_hw.h* file.

🖻 🔍 🔶 🔺 📑 👘 🖌 🗿 🦄	- 🔍 🖸	#P 🗹 🛙	- 🛷 🔻 🎿 🍫 🤇	ኈ ▾ ⇔ ▾				
Explorer 🛱	😑 🔩   🤜	~ -	system.hdf	🗟 system.mss	🗟 system.mss	*helloworld.c	🖻 xadcps.c	xadcps_hw.h      □
👄 scutimer_v2_1			 * @{				L	
≥ scuwdt_v2_1			*/					
			#define	XADCPS_SEQ_CH	CALIB 0×0000	0001 /**< ADC	Calibration	Channel */
≥ sdps_v3_3			#define	XADCPS_SEQ_CH		00000020 /**<		
👄 standalone_v6_5			#define	XADCPS_SEQ_CH		00000040 /**<		
👄 sysmon_v7_4				XADCPS_SEQ_CH		:00000080 /**<		
➡ ttcps_v3_5								ure Channel */
≥ uartps_v3_5				XADCPS_SEQ_CH		00000200 /**<		
≥ usbps v2 4				XADCPS_SEQ_CH		(00000400 /**<		
								uts Channel */
👄 xadcps_v2_2				XADCPS_SEQ_CH				
Y 🗁 SFC				XADCPS_SEQ_CH				
> id xadcps_g.c				XADCPS_SEQ_CH XADCPS_SEQ_CH				
> in xadcps_hw.h				XADCPS_SEQ_CH				
> 🖻 xadcps intr.c				XADCPS_SEQ_CH				
> 🗟 xadcps_selftest.c				XADCPS_SEQ_CH				
> 🖻 xadcps_sinit.c				XADCPS SEQ CH				
				XADCPS SEQ CH				
> 🖻 xadcps.c				XADCPS SEQ CH				
> la xadcps.h				XADCPS SEQ CH				
🗅 Makefile				XADCPS SEQ CH				
lakefile			#define	XADCPS_SEQ_CH	_AUX09 0×0200	0000 /**< 10t	h Aux Channel	*/
/stem.mss			#define	XADCPS_SEQ_CH	_AUX10 0×0400	0000 /**< 11t	h Aux Channel	*/
			#define	XADCPS_SEQ_CH	_AUX11 0×0800	00000 /**< 12t	h Aux Channel	*/
app			#define	XADCPS SEO CH	AUX12 0×1000	00000 /**< 13t	h Aux Channel	*/
ann hen			 -					

Figure 9. 32: Channel Parameter List

The **channels' enable function** should tally with the hardware we enabled in the XADC wizard which is part of the hardware-block-design. This is shown in Figure 9.33 again, on the next page, so that the student will not get confused.

Show disabled ports	Component Name xa	dc_wiz_0		Figure 9. 33: ADC channels
	Basic ADC Setup	Alarms Channe	el Sequenc	Again, the AUXVp01/AUXVn01 o
		Channel Enable	Averag	Vn8AUXVp08/AUXVn08 are not sho
	CALIBRATION	$\checkmark$		in Figure 9.33, however they
ip2into_irpt - user_temp_alarm_out -	TEMPERATURE	$\checkmark$		included so make sure that in
vccint_alarm_out = vccaux_alarm_out =	VCCINT	VOONT	function parameters, this will	
+ vp_vh vccpint_alarm_out -	VCCAUX			-
+ Vaux1 vccddro_alarm_out - + Vaux8 ot_out -	VCCBRAM			included!
s_axi_aclk channel_out[4:0] = s_axi_aresetn eoc_out =	VCCPINT			
alarm_out = eos_out =	VCCPAUX			
busy_out	VCCDDRO			
	VP/VN	<ul><li>✓</li></ul>		
	VREFP			
/*now we will on:		ld tally w	with w	like to monitor what we have enabled in the
* The enabled ch * XADC wizard fo	orming part o _SetSeqChEnab XADCPS_SEQ_O	oles(&XADO CH_TEMP	Cperi XADC	≥.*/ oheral, XADCPS_SEQ_CH_CALIB   PS_SEQ_CH_VPVN   XADCPS_SEQ_CH_VBRAM   CPS_SEQ_CH_AUX08);

Code Snippet 9. 7: Syntax to Enable the ADC channels

This time, the Processing System will sample both internal parameters and also external ADC channels in the same program! Now, the sequence by which the channels will be sampled has to be configured as wel. The function that takes care of this resides in *xadcps.c* file. The parameters can be found in *xadcps\_hw.h* file.

ivavigate search Project		window	пер
© ▼ ≪ ▼ <b>D</b> + ▼ O	▼ 🔌 🖂 🏭	2 🖬 🐼	
t Explorer 🛛	🖻 🔹   🔺	~	🖻 system.hdf 🛚 🗟 system.mss 🗈 system.mss 🗈 *helloworld.c 🛛 🖻 xadcps.c 🖾 🕨 xadcps_hw.l
<pre>scutimer_v2_1 scutimer_v2_1 scutimer_v2_1 sdps_v3_3 standalone_v6_5 sysmon_v7_4 ttcps_v3_5 uartps_v3_5 uartps_v2_4 scatcps_v2_2 scatcps_v2_2 scatcps_v2_2 scatcps_int.c a xadcps_selftest.c a xadcps_sc a xadcps_sint.c a xadcps_sc a xadcps_h </pre>		^	<pre>** ** ** ** ** ** ** ** ** ** ** ** **</pre>
lo Makefile ∕lakefile			<pre>int XAdcPs_SetSeqInputMode(XAdcPs *InstancePtr, u32 InputModeChMask) </pre>
vstem.mss			c .

Figure 9. 34: Location of the Sampling Sequence Function

It is a good idea to copy the parameters of the channel enables function in the function shown in Figure 9.34.

Code Snippet 9. 8: Sampling Sequence

The sampling will start from the first parameter. Once its finished and it stores its digital equivalent in the respective status register, the XADC samples the next channel according to the list of parameters shown in Code Snippet 9.8. It continues to sample the channels one after the other until all the channels are sampled. The XADC will start all over again if the next function is included.

```
/* Before starting to sample data we will set the XADC to continuously
 * sample the channels*/
XAdcPs_SetSequencerMode(&XADCperipheral, XADCPS_SEQ_MODE_CONTINPASS);
```

Code Snippet 9. 9: Function to sample continuously

The *get data()* is used to get the 12-bit decimal equivalent of the quantity you are monitoring. Its location is shown in Figure 9.35.

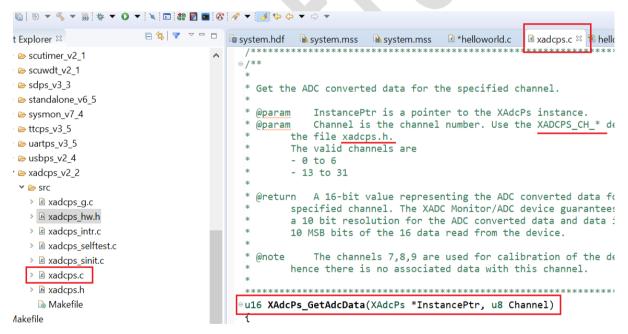


Figure 9. 35: Location of the GetADCData function

The parameters for the above function are stored in *xadcps.h* and are shown Figure 9.36.

8 <b>- 4 - 8 + 0</b>	🕶 🔌 🖂 🏥 🖾 🖉	A 🛨 🔰 🌣 🗇	▼ ⇔ ▼					
xplorer 🛙	▣ 🔄 🔻 ▽ 🗆 🗆	system.hdf	🗟 system.mss	🗟 system.mss	le helloworld.c	🖻 xadcps.c	🗈 helloworld.c	li 🛙 🗎 🛙 🗎 🗎 🗎 🗎 🗎 🗎 🗎
<ul> <li>scutimer_v2_1</li> <li>scuwdt_v2_1</li> <li>sdps_v3_3</li> </ul>	^	/*******	*****	***** Consta	nt Definition	IS *******	******	*****/
standalone_v6_5sysmon_v7_4			Indexes for	the different	channels.			
▶ ttcps_v3_5 ▶ uartps v3 5		* @{ */						
▶ usbps_v2_4		#define X	ADCPS_CH_TEM ADCPS_CH_VCC	INT 0x1 /	**< On Chip T **< VCCINT */		*/	
▶ xadcps_v2_2 '			ADCPS_CH_VCC ADCPS_CH_VPV	N 0x3 /	**< VCCAUX */ **< VP/VN Ded		log inputs */	
<ul> <li>&gt; kadcps_g.c</li> <li>&gt; kadcps_hw.h</li> </ul>		#define X	ADCPS_CH_VRE ADCPS_CH_VRE	FN 0x5 /	**< VREFP */ **< VREFN */			
> 🖻 xadcps_intr.c		#define X		PLY_CALIB 0x	. <b>07</b> /**< Suppl	y Calib Dat		*/
<ul> <li>&gt; la xadcps_selftest.c</li> <li>&gt; la xadcps_sinit.c</li> </ul>				NERR_CALIB 0×	**< ADC Offse 09 /**< Gain	Error Chan	nel Reg */	
<ul> <li>≥ is xadcps.c</li> <li>&gt; is xadcps.h</li> <li>is Makefile</li> </ul>		#define X #define X	ADCPS_CH_VCC ADCPS_CH_VCC ADCPS_CH_VCC ADCPS_CH_AUX	PAUX 0x0E / PDRO 0x0F /	**< On-chip P **< On-chip P	S VCCPAUX O S VCCPDRO O	Channel , Zyng Channel , Zyng Channel , Zyng st Aux Channel	*/ */
kefile tem.mss			ADCPS_CH_AUX				ast Aux channe	

Figure 9. 36: Location of the parameters for the GetADCData Function

One other thing that needs to be clarified is the fact that get\_data() is returning a 16 bit variable, however it has to be stored in a 32 bit variable. This is because the builtin macro converts the 12-bit data from the ADC into either voltage or temperature and the macro itself takes care to do the conversion, which is hidden from the user.

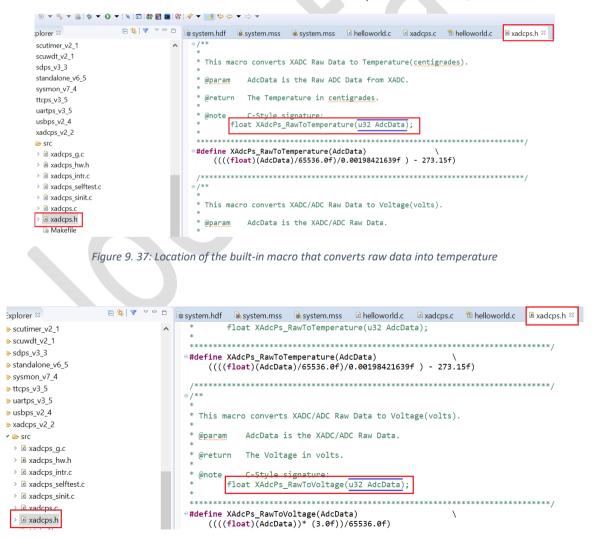


Figure 9. 38: Location of the built-in macro that converts raw data into voltage

The parameter list provided in the header file did not include all the auxiliary channel definitions and therefore the author had to include them manually. UG480, states that auxiliary channel 0 has an address of 16. The addresses continue to increment such that channel 15 has an address of 31. Figure 9.39 shows the definition statements that where included by the author in the header file. The list is shown in the red box.

#define XADCPS_CH_ADC_CALIB	<pre>0x0 /&lt; On Chip Temperature / 0x1 /**&lt; VCCINT */ 0x2 /**&lt; VCCAUX */ 0x3 /**&lt; VP/VN Dedicated analog inputs */ 0x4 /**&lt; VREFP */ 0x5 /**&lt; VREFN */ 0x6 /**&lt; On-chip VBRAM Data Reg, 7 series */ LIB 0x07 /**&lt; Supply Calib Data Reg */ 0x08 /**&lt; ADC Offset Channel Reg */ ALIB 0x09 /**&lt; Gain Error Channel Reg */ 0x0D /**&lt; On-chip PS VCCPINT Channel , Zyng */ 0x0E /**&lt; On-chip PS VCCPAUX Channel , Zyng */ 0x0F /**&lt; On-chip PS VCCPDRO Channel , Zyng */ 16 /**&lt; Channel number for Aux00 Channel */</pre>
<pre>#define XADCPS_CH_AUX01 #define XADCPS_CH_AUX02 #define XADCPS_CH_AUX03 #define XADCPS_CH_AUX04 #define XADCPS_CH_AUX05 #define XADCPS_CH_AUX06 #define XADCPS_CH_AUX07 #define XADCPS_CH_AUX08</pre>	<pre>17 /**&lt; Channel number for Aux01 Channel */ 18 /**&lt; Channel number for Aux02 Channel */ 19 /**&lt; Channel number for Aux03 Channel */ 20 /**&lt; Channel number for Aux04 Channel */ 21 /**&lt; Channel number for Aux05 Channel */ 22 /**&lt; Channel number for Aux06 Channel */ 23 /**&lt; Channel number for Aux07 Channel */ 24 /**&lt; Channel number for Aux08 Channel */</pre>
<pre>#define XADCPS_CH_AUX_MAX</pre>	31 /**< Channel number for Last Aux channel */

Figure 9. 39: Adding the channel numbers manually

Since the definition statements where included manually by the author, it is imperative to either click on <u>Save all</u> or make sure that the header file is saved before saving the actual C file.

The following is the get\_data() together with the macros in the while (1) loop:

```
while(1)
ſ
    ADC_valueTemp=XAdcPs_GetAdcData(&XADCperipheral, XADCPS_CH_TEMP);
    temp=XAdcPs_RawToTemperature(ADC_valueTemp);
    ADC_valueVp=XAdcPs_GetAdcData(&XADCperipheral, XADCPS_CH_VPVN);
    Vp=XAdcPs_RawToVoltage(ADC_valueVp);
    ADC valueVBRAM=XAdcPs GetAdcData(&XADCperipheral, XADCPS CH VBRAM);
    VBRAM=XAdcPs_RawToVoltage(ADC_valueVBRAM);
    ADC_valueAux01=XAdcPs_GetAdcData(&XADCperipheral, XADCPS_CH_AUX01);
    Aux01=XAdcPs_RawToVoltage(ADC_valueAux01);
    ADC_valueAux08=XAdcPs_GetAdcData(&XADCperipheral, XADCPS_CH_AUX08);
    Aux08=XAdcPs_RawToVoltage(ADC_valueAux08);
    printf("Internal temperature: %f\n\r",temp);
    printf("Vp voltage: %f\n\r",Vp);
    printf("VBRAM voltage: %f\n\r",VBRAM);
printf("Aux01 voltage: %f\n\r",Aux01);
printf("Aux08 voltage: %f\n\r",Aux08);
    delay();
```

Print() must be changed to printf() statement because this will generate an error of too many parameters. Create a boot image file and copy it to SD card. What follows is the whole code:

```
* helloworld.c: simple test application
  * This application configures UART 16550 to baud rate 9600.
  * PS7 UART (Zynq) is not initialized by this application, since
  * bootrom/bsp configures it to baud rate 115200
  *this project will attempt to control multiple ADC inputs
  *such as some of the internal parameters, Vp and also Aux1
  * UART TYPE BAUD RATE
  * _____
  *
     uartns550 9600
     uartlite
                 Configurable only in HW design
     ps7_uart 115200 (configured by bootrom/bsp)
  */
 #include <stdio.h>
 #include "platform.h"
#include "xil_printf.h"
 #include "xadcps.h"
 void delay (void);

int main()

 {
     int init_success,STstatus,ch_status,SeqModeStatus;
     u32 ADC_valueTemp,ADC_valueVp,ADC_valueVBRAM,ADC_valueAux01,ADC_valueAux08;
     float temp,Vp,VBRAM,Aux01,Aux08;
     XAdcPs XADCperipheral;
     XAdcPs_Config *XADC_configPtr;
     init_platform();
     /* Initialise the XADC*/
     XADC_configPtr = XAdcPs_LookupConfig(XPAR_PS7_XADC_0_DEVICE_ID);
     init_success=XAdcPs_CfgInitialize(&XADCperipheral,XADC_configPtr,XADC_configPtr->BaseAc
     if(init_success != XST_SUCCESS)
     {
         return XST_FAILURE;
     }
     /*Self test. This should also reset the XADC*/
     STstatus=XAdcPs_SelfTest(&XADCperipheral);
     if(STstatus != XST_SUCCESS)
     {
         return XST_SUCCESS;
     }
 /*Self test. This should also reset the XADC*/
 STstatus=XAdcPs_SelfTest(&XADCperipheral);
 if(STstatus != XST_SUCCESS)
 {
     return XST_SUCCESS;
 }
 /* now we stop the sequencer*/
 XAdcPs_SetSequencerMode(&XADCperipheral, XADCPS_SEQ_MODE_SINGCHAN);
 /* we put the XADC in safe mode so that we can change the configuration registers*/
 XAdcPs_SetSequencerMode(&XADCperipheral, XADCPS_SEQ_MODE_SAFE);
                                                                                   Line:
 /* Disable the alarms*/
 XAdcPs_SetAlarmEnables(&XADCperipheral,0x0000);
 /*now we will enable the channels we would like to monitor
  * The enabled channels should tally with what we have enabled in the
  * XADC wizard forming part of our hardware.*/
 ch_status=XAdcPs_SetSeqChEnables(&XADCperipheral, XADCPS_SEQ_CH_CALIB |
                  XADCPS_SEQ_CH_TÈMP | XADCPS_SEQ_CH_VPVN | XADCPS_SEQ_CH_VBRAM |
XADCPS_SEQ_CH_AUX01 | XADCPS_SEQ_CH_AUX08);
```

```
if(ch_status != XST_SUCCESS)
{
    return XST_FAILURE;
3
'*Now we need to set which channels will be sampled in sequence*/
SeqModeStatus=XAdcPs_SetSeqInputMode(&XADCperipheral, XADCPS_SEQ_CH_CALIB |
         XADCPS_SEQ_CH_TEMP | XADCPS_SEQ_CH_VPVN | XADCPS_SEQ_CH_VBRAM |
         XADCPS_SEQ_CH_AUX01 | XADCPS_SEQ_CH_AUX08);
if(SeqModeStatus != XST_SUCCESS)
{
    return XST_FAILURE;
}
/* Before starting to sample data we will set the XADC to continuously
 * sample the channels*/
XAdcPs_SetSequencerMode(&XADCperipheral, XADCPS_SEQ_MODE_CONTINPASS);
while(1)
ſ
     ADC_valueTemp=XAdcPs_GetAdcData(&XADCperipheral, XADCPS_CH_TEMP);
     temp=XAdcPs_RawToTemperature(ADC_valueTemp);
     ADC_valueVp=XAdcPs_GetAdcData(&XADCperipheral, XADCPS_CH_VPVN);
     Vp=XAdcPs RawToVoltage(ADC valueVp);
     ADC_valueVBRAM=XAdcPs_GetAdcData(&XADCperipheral, XADCPS_CH_VBRAM);
     VBRAM=XAdcPs_RawToVoltage(ADC_valueVBRAM);
     ADC_valueAux01=XAdcPs_GetAdcData(&XADCperipheral, XADCPS_CH_AUX01);
     Aux01=XAdcPs_RawToVoltage(ADC_valueAux01);
     ADC valueAux08=XAdcPs GetAdcData(&XADCperipheral, XADCPS CH AUX08);
     Aux08=XAdcPs_RawToVoltage(ADC_valueAux08);
     printf("Internal temperature: %f\n\r",temp);
    printf("Vp voltage: %f\n\r",Vp);
printf("VBRAM voltage: %f\n\r",VBRAM);
printf("Aux01 voltage: %f\n\r",Aux01);
     printf("Aux08 voltage: %f\n\r",Aux08);
     delay();
}_
          - . -
                  ...
     cleanup platform();
     return 0:
}
void delay (void)
ł
     for(unsigned i=0;i<10000000;i++)</pre>
     ſ
     }
}
```

So, the focus of this chapter was to show how to sample internal parameters, the dedicated ADC channel and two auxiliary ADC channels from the Processing System. During the discussion, the short comings encountered by the author were highlighted and their workaround explained. It must be said that the voltage and temperature macros offered by Xilinx are not so reliable and one should write his/her own functions to convert to temperature and voltage. In the next chapters, the XADC will be sampled from the Programmable Logic part.

## <u>Monitoring two ADC channels with data simultaneously shared between the</u> <u>PS and PL parts of the Zynq 7</u>

## <u>Introduction</u>

In this chapter, two external analogue inputs will be monitored by both the *Processing System* part and the *Programmable Logic* part of the Zynq 7. It will be shown how easy it is to use the XADC block simply because the advanced hardware included in the XADC makes life so much easier for the design engineer!

## Creating a Vivado Project

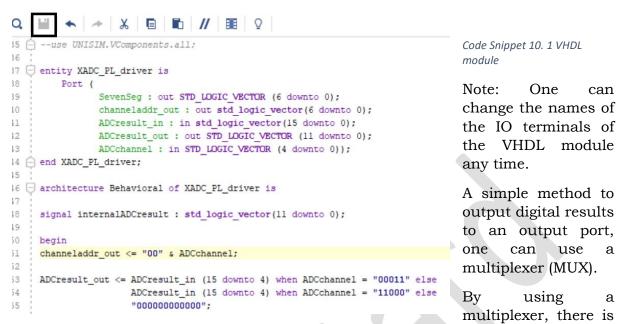
There are sections in chapters 1 and 2 that explain in detail how to create a Vivado project both for the PL part and also for the PS part, so this part of the document will be skipped.

After creating the project and also creating a VHDL source file as part of the project, one should wait for Vivado to update as shown in Figure 10.1 below:

Q         ★         +         Image: Constraint of the second s	Updating 🔱	-	
Hierarchy Libraries Compile Order	DC. PLdriver.vbd)	×	Settings E Project nam Project locat Product fami
Figure 10. 1: Wait	for Vivado to	Update	2
PROJECT MANAGER - Multiple_XAD	DC_input_data_s	hared_P	'S_PL
PROJECT MANAGER - Multiple_XAU Sources	DC_input_data_s	_	PS_PL
Sources Q X = XAL		_	-

Figure 10.2 shows the location of the VHDL module within the project. Double click on it to edit it.

The XADC module will be configured to run in <u>continuous mode</u>, therefore it will output the ADC result in 12-bit digital form, together with the corresponding <u>channel-address</u>. Since the main objective of this exercise is to learn how to configure the XADC block and interface it with both PS and PL parts of the Zynq 7, the VHDL module will output the raw 16-bit result directly to the LEDs. It is known that the XADC result resides between bit 4 and bit 15 and therefore some form of processing is needed to obtain the actual value as a decimal number! The Zynq Processing system will do its own processing (shifting to the right by 4 bits) on the XADC data while the VHDL module will do its own separate processing. Code snippet 10.1 shows the VHDL code to implement a simple multiplexer because the objective of this project is to make sure that the XADC data is available for both PS and PL at the same time! Also note how simple concept used to implement shifting of data in VHDL!



no issue of timing constraints or worrying that the XADC block is not in sync with the VHDL module because the multiplexer used will use the channel address of the analogue input as the select bits to select which digital data it will be output.

For this experiment a custom-made development board designed by the author was connected to the cape IO board by MYIR. It was fully isolated from both the inputs and the outputs, to make sure that the pins of the Zynq 7 will never get damaged. This dev-board extension had 18 LEDs driven by opto-transistors, 4 preset pots, 4 push-to-make switches, 4 slide switches and a single seven segment display.

Save the VHDL module and create a block design.

Note: One could start from the block diagram and then write the VHDL code after. There is no priority!

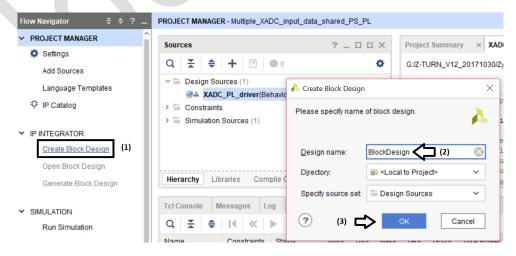


Figure 10. 3: Creating a Block Design

Click on "create a Block Design". In the following pop up window give a name to the block design and then click on OK.

## Adding the Zynq PS system

The Zynq PS system must be added to your block design because it will download the .bit file of the VHDL part of the project to the FPGA part of the Zynq SoC. Apart from that, in this project, the Zynq 7 Processing System will be used to monitor <u>in</u> <u>parallel</u> the ADC data from XADC.

Figure 10. 4: Adding an IP to the Block Design
To add an IP on the canvas, one can either hover the mouse on the + sign in the middle or on the + sign that is part of the menu. Click on either one of them and a new pop up window pops up.
igure 10. 5: Calling the IPs
Vrite the name of the IP in the field
provided and double click on it to add it
o the block design.
Image: Second system       Figure 10. 6: Zynq PS is part of the Block         Design       Design
Click on <u>Run Block</u>
Automation PR +

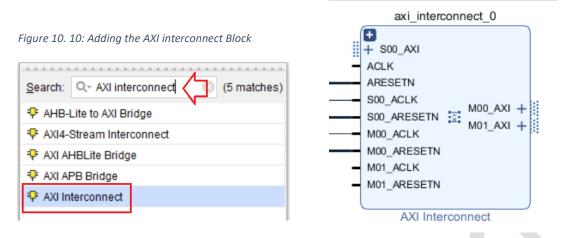
Automatically make connections in your design configuration options on the right.	by checking the boxes of the blocks to connect. Select a block on the left to display its	
Q X € ✓ Z All Automation (1 out of 1 selected) Ø processing_system7_0	Description         This option sets the board preset on the Processing System. All current properties will be overwritten by the board preset and generates external connections for FDKED_IO, Trigger and DDR interfaces.         NDTE: Apply Board Preset will discard existing IP configuration - please uncheck this box, if you wish to retain previous configuration.         Instance: /processing_system7_0         Options         Make Interface External: FDED_IO, DDR         Apply Board Preset         Origons Trigger In:         Disable	Figure 10. 7: Leave all the Presets
(?)	OK Cancel	
+ S_AXI_HP0_FIFC + S_AXI_HP0 M_AXI_GP0_ACLK S_XXI_HP0_ACLK	DDR + FIXED_IO + IIC_0 + USBIND_0 + M_AXI_GP0 + TTCO_WAVE0_OUT TTCO_WAVE2_OUT FCLK_CLK0 FCLK_RESET0_N	Figure 10. 8: Connect the AXI clocks to the Zynq PS

For this project, two AXI blocks will be used, one to interface the XADC block with the Zynq Processing System and one to send the processed XADC result to the VHDL module. The XADC will output the XADC result in 16 bit format together with the 6 bit channel address. The other AXI block is used to interface the Processing System with the Programmable Logic fabric. Another way to add an IP block is shown in Figure 10.9

	Properties	Ctrl+E	
×	Delete	Delete	
<b>.</b>	Сору	Ctrl+C	Figure 10. 9: Adding another IP block
16	Paste	Ctrl+V	To add another <i>IP</i> , right-click on the canvas and select
Q.	Search	Ctrl+F	Add IP from the menu.
50	Select All	Ctrl+A	<u>Add II</u> from the mend.
+	Add IP	Ctrl+I	
	Add Module		
1	IP Settings		

Where there is an AXI block, there should also be an *AXI interconnect block*. This ensures maximum data rate transfer between the Zynq Processing System and the Programmable Logic in the block-design.

#### Adding the AXI interconnect Block



*In the field, write AXI interconnect and then double-click on it from the list.* 

AXI Interconnect (2.1)	AXI Interconnect (2.1)		
Documentation IP Location     Component Name axi_interconnect_0		AXI interconnect must have two master and one slave interface for <b>this application</b> .	
Top Level Settings Slave Interfaces M	aster Interfaces		
Number of Slave Interfaces	1 ~		
Number of Master Interfaces	2 🗸		
Interconnect Optimization Strategy			

With the adjacent settings, two AXI GPIO blocks could be connected to the AXI interconnect. Each AXI GPIO block has two channels. In the first AXI GPIO block, both channels will be used as input channels while the second AXI GPIO will have its channels configured as outputs.

XI Interconnect (2	2.1)		
Documentation	IP Location		
Component Name	xi_interconnect_0		
Top Level Settings	Slave Interfaces	Master Interfaces	
Number of Slave Inte	erfaces	1	~
Number of Master In	terfaces	2	~

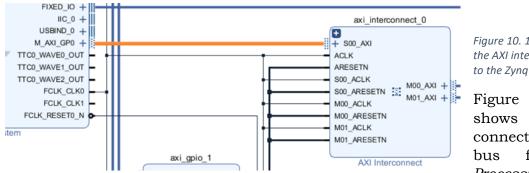


Figure 10. 12: Connecting the AXI interconnect block to the Zynq PS

Figure 10.12 shows how to connect the data bus from the *Processing System* 

to the *AXI interconnect*. It is advisable to include the *Processor System Reset block* to reduce the amount of warnings while synthesizing the design.

## Include the Processing System Reset

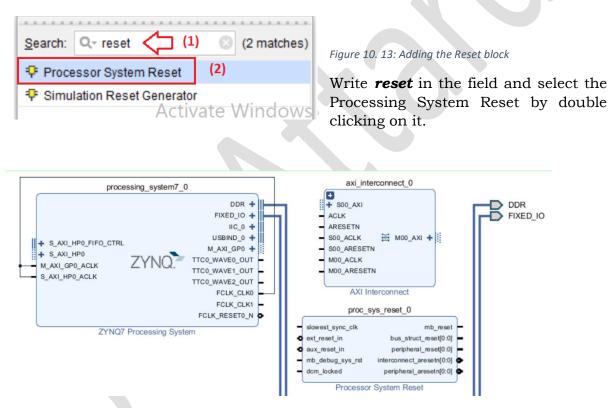


Figure 10. 14: The PS Reset block is part of the Block Design

Now it is time to wire the three blocks together. Start from the reset pins. Connect it to the reset output of the Processing System. This is shown in Figure 10.15.

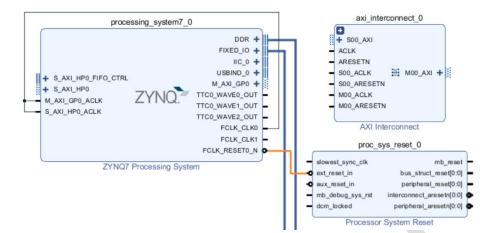


Figure 10. 15: Connecting the Zynq PS to the PS Reset Block

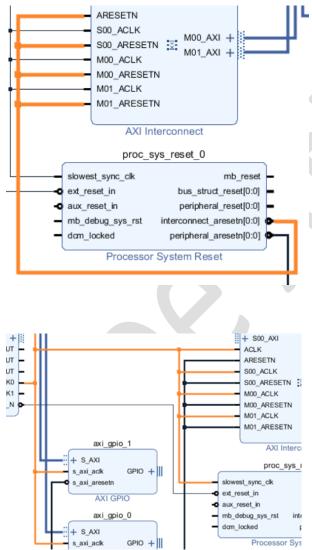


Figure 10. 16: Connecting the PS Reset with the AXI interconnect Block

Figure 10.16 shows all the reset inputs of the AXI interconnect block are connected to the *interconnect\_aresetn* [] of the Processing System block. This makes sure that there will be minimal delay when resetting the system.



All clock signals should be connected to the same 100 MHz clock signal, emerging from the Processing System part. This ensures full synchronisation.

## Include the AXI GPIO

The AXI GPIO will be used to interface the PS system with XADC. Another AXI GPIO will be used to interface the output pins located on the FPGA part of the SoC to the PS so that the PS part will drive the LEDs connected to the PL part of the Zynq Processing System. The LEDs should give a clear visual indication whether the ADC result is actually reflecting a change in the analogue voltage input.

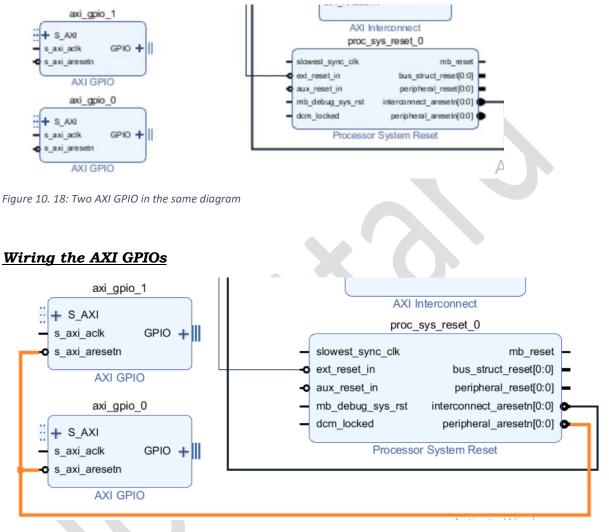
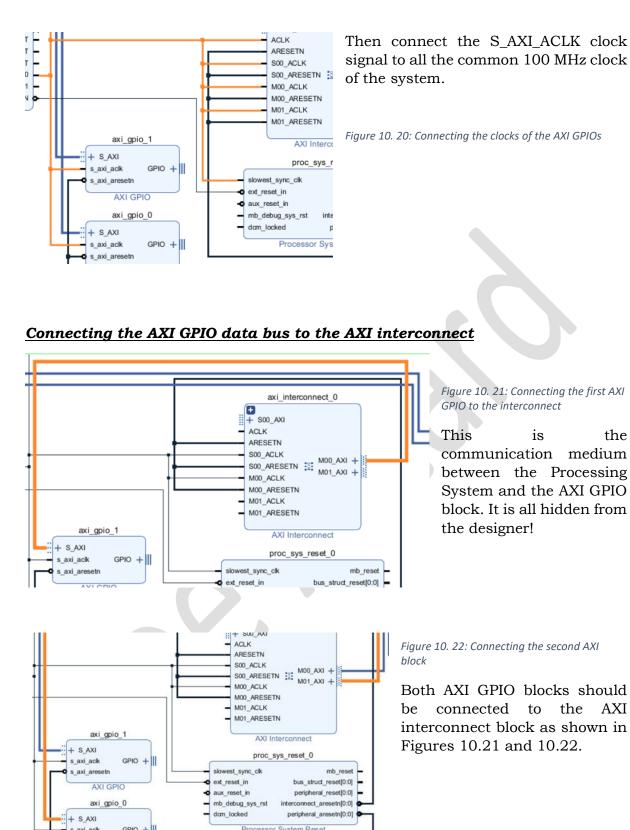


Figure 10. 19: Connecting the AXI GPIO Reset

First, make sure that the AXI GPIO block reset is connected to the *peripheral\_aresetn[]* input.

the

AXI



Changing the width of the data busses of the AXI GPIOs

As discussed before, one of the AXI GPIO blocks will be used as an interface between the Processing System and the XADC, while the other AXI GPIO block be used to extend the external pinouts of the Processing System by using some of the external

pins allocated to the Programmable Logic side of the System-on-Chip. The following Figures show how to configure the AXI GPIO blocks for custom applications.

Compo	nent Name a	ki_gpio_1									
						Figure 10	. 23: Co	onfiguring	the AXI GP	10 1	
Board	IP Configur	ation				Leave	the	board	interf	ace	as
Associ	iate IP interfac	e V IP Confi	ouration			custo					
IP Inte		-[		Board Interfa	ice	eusce.					
GPIO				Custom							
GPIO2	2			Custom							
CI	ear Board Par	ameters									
Component	Name axi_gpio_1										
Board IP (	Configuration			G	PIO 2						
GPIO					🖌 All	Inputs					
✓ AII	Inputs					Outputs					
All (	Outputs					Outputs				_	
GPIO W	Vidth 16		🔇 [1 - 32]		GPIO V	Vidth	5			8 [1	- 32]
Default	Output Value 0x0000	0000	(0x0000000,0xFFFFF	FF]	Default	t Output Valu	e 0x0	0000000		()	x0000
Default	Tri State Value 0xFFFF	FFFF	0 [0x0000000,0xFFFFFF	FF]							
🗸 Enable I	Dual Channel										
GPIO 2				Figure	10. 24: 0	Configuring	the AX	I GPIO 2			
✓ AII	Inputs										
				Chat	nnel	1 w1ll a	accot	nmodat	te the	16-	b1f

ADC result from the XADC, while channel 2 accommodates the channel address from XADC.

So, Figure 26 above shows a single AXI GPIO block consisting of two channels. One of the channels is made up of 16 bits while the second channel is made up of 5 bits. Note that both channels are configured as inputs.

The second AXI GPIO will also be left as **custom**.

AXI GPIO (2.0)						
Documentation 🛛 🕞 IP Location			Figure 10. 25: Conf	iguring A	XI GPIO 3	
Show disabled ports	Component Name axi_gpio_0 Board IP Configuration		Configuring GPIO.	the	second	AXI
	Associate IP interface with board interfa	ce				
	IP Interface	Board Interface				
	GPIO	Custom				
	GPI02	Custom				
+ s_AXI	Clear Board Parameters					

	D	efault Output Value ( efault Tri State Value (		The cor spe sec goi	re 10. 26: Configuring AXI GPIO 2_2 e second AXI GPIO will drive 18 LEDs nnected on the devBoard that designed ecifically for the z-turn board. The cond channel of this AXI GPIO is not ng to be used.
VCLK	Q #2	g the XADC in th Search Select All Add IP Add Module	Ctr	'l+F 'l+A	Figure 10. 27: Adding the XADC Right-click anywhere on the canvas then choose Add IP.
	ADC V	<b>X- XADC</b> Vizard ck on <i>XADC</i> wiza	(1 match) rd.		gure 10. 28: Call XADC from List Vrite XADC in the field provided and
+ ) - s_a	s_axi_lite /p_Vn Ixi_aclk Ixi_areset	xadc_wiz_0 ip2intc_ipt user_temp_alarm_out vccint_alarm_out vccpirt_alarm_out vccpaux_alarm_out vccddro_alarm_out ot_out alarm_out eoc_out alarm_out eos_out busy_out	to configu application Also, the changed fro	orig ire XAI om A	ginal XADC block. Double-clicking on it it according to the needs of this DC mode of communication will be <i>AXI4 lite</i> to <i>Dynamic Reconfiguration Port</i> the XADC block, the PS and PL.

# <u>Configuring the XADC block to be compatible with the software and hardware of this project</u>

## The BASIC page:

Basic ADC Setup Alarms Channel Sequencer Summary		Figure 10. 30: XADC Basic Page
Interface Options	Timing Mode	gale 201 00174 12 0 2 40.0 1 4 ge
AXI4Lite   DRP  None	Continuous Mode	
Startup Channel Selection	DRP Timing Options	
Simultaneous Selection	Enable DCLK	
O Independent ADC	DCLK Frequency(MHz)	
Single Channel	ADC Conversion Rate(KS	
Channel Sequencer	Acquisition Time (CLK)	

In the basic page:

- change the interface options to <u>DRP</u>
- leave the timing mode in <u>continuous mode</u>.
- Change the startup channel selection to *channel sequencer*

AXI4STREAM Options	Figure 10. 31: XADC Configuration 2
Enable AXI4Stream	• Leave the AXI4STREAM as is
Control/Status Ports	• Remove the tick from <i>reset_in</i> box so that the XADC will be <i>free running</i>
reset_in     Temp Bus     JTAG Arbiter  Event Mode Trigger	• Note the Event Mode Trigger is not an option. This is because the continuous sampling
onvst in O convstclk in	mode was selected.

## Analogue Sim File Option

Leave everything as is in the Analogue Sim File option section.

#### The ADC Setup Page

Channel Se	equencer	Summa	гу		
~	Channel Av	eraging	None	~	
	Supply Sen	isor Calil			
	Se	nsor Offs			
bration	✓ Se	nsor Offs	256	n Ganufá	atior
ng					
VP VN		~			
	<ul><li>✓</li><li>bration</li><li>ng</li></ul>	Supply Ser	Channel Averaging Supply Sensor Call Sensor Offs bration Sensor Offs ng	Channel Averaging None Supply Sensor Call B Sensor Offs 64 256 bration Sensor Offserance can ng	Channel Averaging None Channel Averaging None Channel Averaging None Channel Supply Sensor Can Channel

#### it.

Leave the ADC calibration as is.

Leave channel for MUX as is.

## The Alarms Page

over 1	emperature Alarm (°C)		User T	emperature Alar
Trigger	125.0	40.0 - 125.0]	Trigger	85.0
			Reset	60.0
Reset /CCIN	70.0 [	40.0 - 125.0]	$\sim$	JX Alarm (Volts)
	F Alarm (Volts)	0 - 1.05]	$\sim$	JX Alarm (Volts)

#### Figure 10. 32: The ADC Setup Page

In the ADC setup page, leave <u>sequencer mode</u> in <u>continuous</u> mode so that the XADC will operate in <u>free running</u> mode.

Opt for <u>averaging 16</u> and therefore XADC will sample and add 16 ADC results and output their average. This is very convenient because a low pass filter is created in hardware and the designer does not have to worry about

#### Figure 10. 33: XADC Alarms Page

Remove all the ticks in the alarms page. For this project the alarms are not needed.

Show disabled ports	Component Name xa	dc_wiz_0	
	Basic ADC Setup	Alarms Channel Se	equencer
	VP/VN		
	VREFP		
	VREFN		
channel_out[4:0]	vauxp0/vauxn0		
+ s_drp eoc_out - + Vp_Vn alarm out -	vauxp1/vauxn1		
+ Vaux8 eos_out -	vauxp2/vauxn2		
busy_out	vauxp3/vauxn3		
	vauxp4/vauxn4		
	vauxp5/vauxn5		
	vauxp6/vauxn6		
	vauxp7/vauxn7		
	vauxp8/vauxn8		

## The Channel Sequencer Page

Figure 10. 34: XADC Channel Sequencer Page

In the channel sequencer page, the channels that are to be sampled must be selected.

Note in the left pane that the XADC block has reduced in size due to the changes we have done



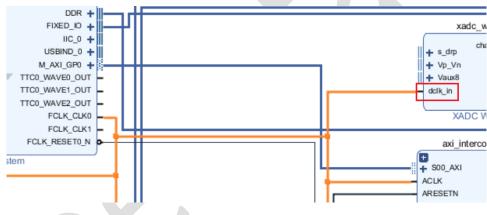
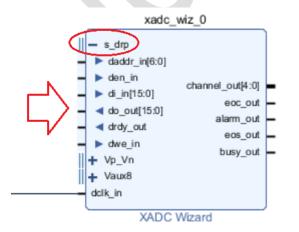


Figure 10. 35: Connecting the XADC Clock

**Dclk\_in** should be connected to the 100 MHz clock because this is the default clock input for the 1MSPS can be achieved on *Vp/Vn analogue inputs*.



#### Figure 10. 36: Extending the DRP Bus

Hover the mouse over **s\_drp**. Notice two arrows pointing downwards. At that point left-click the mouse to reveal the DRP busses as shown in Figure 10.36



*Figure 10. 37: Connecting the Vp/Vn to external pins* 

**Vp/Vn** will be connected to their dedicated external pin. To do this, hover the mouse on **Vp/Vn**, right-click and choose *make external* as shown in Figure 10.37. Do the

same for **Vaux8**.

Now according to *page 73 of UG480*, for XADC to operate in continuous mode, one must do the following connections:

- Connect channel[4:0] to daddr\_in[4:0] daddr\_in[6:5] must be connected to logic 0.
- Connect **d\_en\_in** with **eoc\_out**
- Connect **drdy\_out** with **dwe\_in**

For point 1 above, the successful way to do it is to connect them to a VHDL module and concatenate "00" to bits 6:5 of daddr\_in.

#### channeladdr\_out <= "00" & ADCchannel;</pre>

where **ADCchannel** is connected to **channel\_out** of the XADC block.

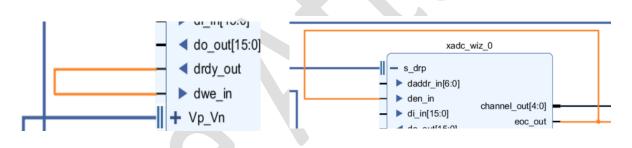
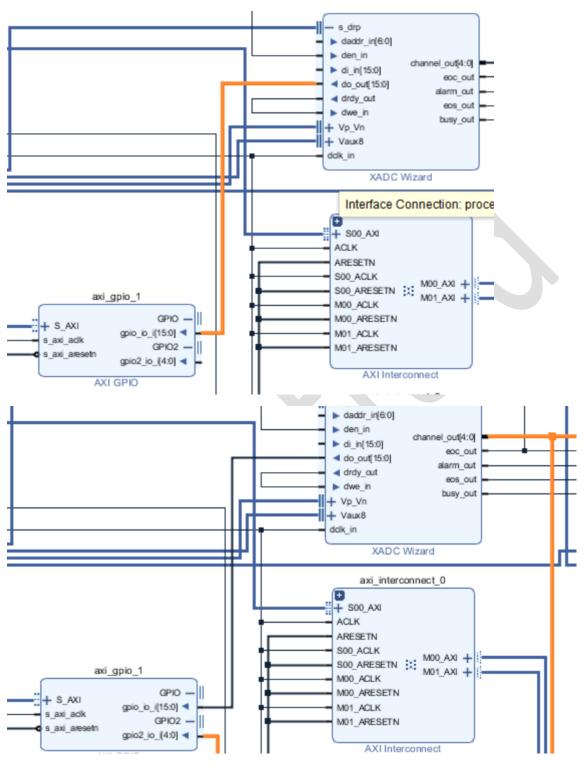


Figure 10. 38: Wiring the XADC

Figure 10.38 show the rest of the connections of the XADC block.



#### Connecting the ADC result bus to the AXI GPIO block



## Including the VHDL module in the block design

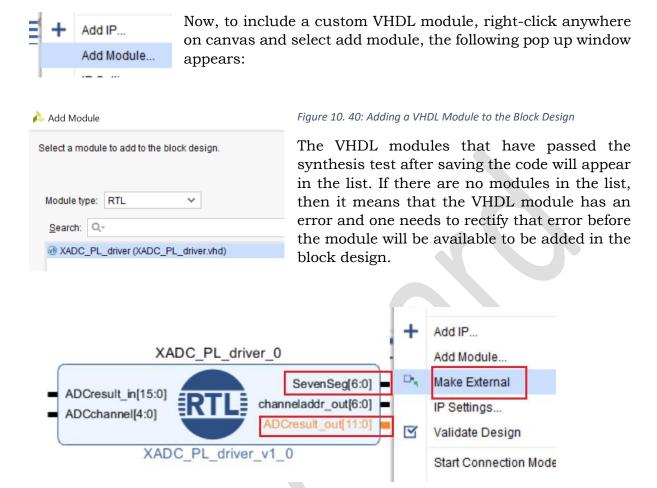
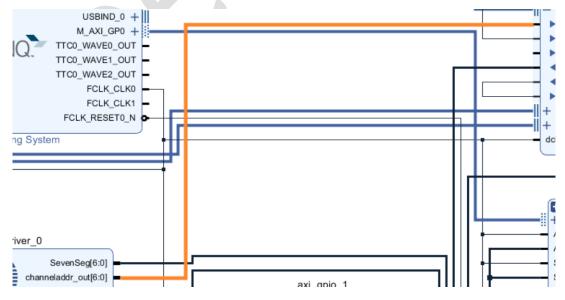


Figure 10. 41: Connecting the HDL module to outside peripherals

Hover mouse on the respective pins, right click and then choose "make external".



*Figure 10. 42: Connecting the Channel Address Bus via the VHDL module* 

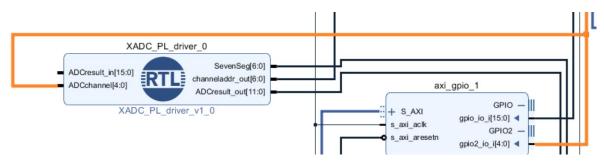


Figure 46: The channel address is input to the VHDL module

The **ADCchannel** bus is connected to **channel\_out** of the XADC block. This will make sure that **daddr\_in[6:5]** will be connected to <u>logic 0</u> and **daddr\_in[4:0]** will be connected to **channel\_out** and therefore the channel address is still 7 bits wide but only the first 5 bits are really selecting which channel is being sampled! This is specified on page 73 of UG480.

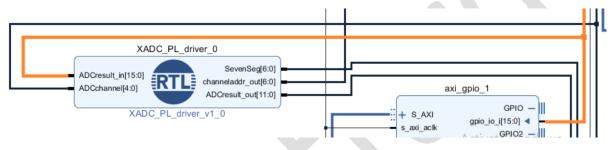


Figure 10. 43: ADC result bits shared between AXI GPIO and VHDL module

**ADCresult** is connected to **do\_out** of *XADC*. This data will be shared with the PS via the AXI GPIO.

The architecture of the VHDL module is shown in the code snippet 10.1 below:

C

begin
if rising edge(clk) then
if ADCchannel = "00011" then
if internalADCresult >= 0 and internalADCresult < 400 then sevenseg <= "0111111";0
elsif internalADCresult >= 400 and internalADCresult < 800 then sevenseg <= "0000110";1
elsif internalADCresult >= 800 and internalADCresult < 1200 then sevenseg <= "1011011";2
elsif internalADCresult >= 1200 and internalADCresult < 1600 then sevenseg <= "1001111";3
elsif internalADCresult >= 1600 and internalADCresult < 2000 then sevenseg <= "1100110";4
elsif internal integer lt >= 2000 and internalADCresult < 2400 then sevenseg <= "1101101";5
elsif internalADCresult >= 2400 and internalADCresult < 2800 then sevenseg <= "1111101";6
elsif internalADCresult >= 2800 and internalADCresult < 3200 then sevenseg <= "0000111";7
elsif internalADCresult >= 3200 and internalADCresult < 3600 then sevenseg <= "1111111";8
elsif internalADCresult >= 3600 and internalADCresult < 4096 then sevenseg <= "1100111";9
else sevenseg <= "0000000";
end if;
end if;
end if;
end process;
- end Behavioral;

Code Snippet 10. 2: VHDL code

# Validating the schematic

Click on the validate schema icon on the tool bar of th	ne canvas
🕒 🖉 🖸 Figure 10. 44: Validate the Block Design	
processing_system7_(	e
	A
There were four critical warning messages while validating this design.	
Messages	
[BD 41-1356] Address block  is not mapped into	
. Please use Address Editor to either map or exclude it.	
[BD 41-1356] Address block  is not mapped into . Please use Address Editor to either map or exclude it.	
[BD 41-1356] Address block  is not mapped into . Please use Address Editor to either map or exclude it.	
[BD 41-1356] Address block  is not mapped into	
. Please use Address Editor to either map or exclude it.	
OK Open Messages View	

Figure 10. 45: Critical Warning that can be solved

To solve the above warnings, one has to follow the steps in Figure 10.46.

Diagram × Address E	ditor × XADC_PL_dr	iver.vhd	Figure 10. 46: The Address Editor
Q,   ¥   €   111	Slave Interface	Base N	• Click on the <i>address edit</i> reveal the problematic blocks.
✓ ♀ processing_system7	′_0		• <i>Highlight</i> one of the blo
<ul> <li>Data (32 address</li> <li>Unmapped Sla</li> </ul>	bits : 0x40000000 [ 1G ])		then right click on it and ch Assign address
axi_gpi	Assign Address	Pog	• Do the same for the see block
Unconnected SI	Properties		• Figure 10.47 shows assigned address of the AXI block
processing_	Unmap Segment Exclude Segment		

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
✓ ₱ processing_system7_0					
🛩 🖽 Data (32 address bits : 0	x40000000 [ 1G ])	1			
🚥 axi_gpio_0	S_AXI	Reg	0x4120_0000	6 🔻	0x4120_FFFF
🚥 axi_gpio_1	S_AXI	Reg	0x4121_0000	6 🔻	0x4121_FFFF
Unconnected Slaves					
processing_system7_0	S_AXI_HP0	HP0_DDR_LOWOCM			

Figure 10. 47: The AXI GPIO are assigned an address

#### Create a Hardware Wrapper

Create a hardware wrapper for the block design. This will act like a top-level module.

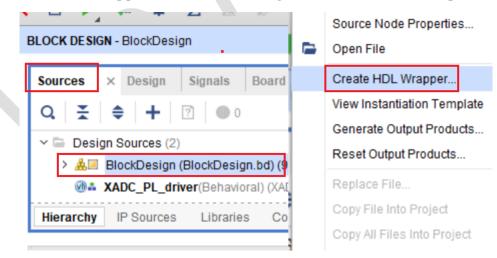


Figure 10. 48: Creating a Hardware Wrapper

#### Start synthesis

After the hardware wrapper is created, it is time to run *synthesis*. Click on *Run Synthesis* on the left-hand-side of the IDE and click on *OK* for the following window.

OUT gpio\_io\_o\_0[17] v LVCMOS33\* v LVCMOS33\* gpio\_io\_o\_0[16] OUT gpio\_io\_o\_0[15] OUT ~ LVCMOS33\* v LVCMOS33\* OUT gpio\_io\_o\_0[13] OUT v LVCMOS33\* gpio\_io\_o\_0[12] OUT v LVCMOS33\* gpio\_io\_o\_0[11] OUT v LVCMOS33\* gpio\_io\_o\_0[10] OUT LVCMOS33\* gpio\_io\_o\_0[9] OUT LVCMOS33\* √ gpio\_io\_o\_0[8] OUT L16 ~  $\checkmark$ 35 LVCMOS33\* ✓ gpio\_io\_o\_0[7] OUT K19 ~  $\checkmark$ 35 LVCMOS33\* ✓ gpio\_io\_o\_0[6] OUT L19  $\checkmark$ 35 LVCMOS33\* ✓ gpio\_io\_o\_0[5] OUT M17  $\checkmark$ 35 LVCMOS33\*  $\checkmark$ √ gpio\_io\_o\_0[4] OUT M19 35 LVCMOS33\*  $\checkmark$ ✓ gpio\_io\_o\_0[3] OUT F16 35 LVCMOS33\*  $\checkmark$ OUT 35 LVCMOS33\* gpio\_io\_o\_0[2] E18  $\checkmark$ OUT 35 LVCMOS33\* gpio\_io\_o\_0[1] D19 Activate Windoss LVCMOS33\* OUT gpio\_io\_o\_0[0] E17 Go to Settings to activate Windows.

Before running implementation, it would be wise to assign the appropriate pin numbers to all external pins as shown in Figure 10.49.

Figure 10. 49Part of the pin assignments

Since there is not enough IOs to cover both PS and PL, half of them will be allocated to the PS part through the AXI GPIO and half of them to the PL part.

The single seven segment display is connected to the Programmable Logic part while

HSTL_II_18	the RS232 port will be used to send data to PC to confirm that PS is reading XADC data.
HSUL_12	Note that all the voltages where changed to 3V3 instead of
LVCMOS15 LVCMOS18 LVCMOS25	their default 1V8. Notice the tick on the pins that were assigned a pinout!
LVCMOS33 v default (LVCMOS18)	Figure 10. 50: Changing the Operating Voltage to 3V3
default (LVCMOS18)	
default (LVCMOS18)	

SevenSeg_0 (7)	OUT				$\checkmark$	35	LVCMOS33*
·☑ SevenSeg_0[6]	OUT		H20	~	$\checkmark$	35	LVCMOS33*
- SevenSeg_0[5]	OUT		G20	~	$\checkmark$	35	LVCMOS33*
- SevenSeg_0[4]	OUT		H18	~	$\checkmark$	35	LVCMOS33*
- SevenSeg_0[3]	OUT		K18	~	$\checkmark$	35	LVCMOS33*
SevenSeg_0[2]	OUT		L17	~	$\checkmark$	35	LVCMOS33*
- SevenSeg_0[1]	OUT		F20	~	$\checkmark$	35	LVCMOS33*
√ SevenSeg_0[0]	OUT		G18	A ctin		Vindows	LVCMOS33*
Context (4)				ACUV	ate v	VIIIGOVIS	

#### Figure 10. 51: Assignment of Seven Segment Pins

Due to the changes made to the external configuration of the design, a new constraints file has to be created because now it differs from the standard constraints file included with the board support files. So, Figure 10.51 shows that a new constraints file will be created that will include the changes made.

<u>F</u> ile <u>E</u> dit F <u>l</u> ow <u>T</u> ools <u>W</u> indo	w La <u>v</u> out <u>V</u> iew <u>H</u> elp	Q- Quick Access
	🗴 🕨 👬 🧿	🖻 🌣 <u>Σ</u> 继 🖉 😹
Flow Navigator 😤 🗢 ? _ Language Lemplates	SYNTHE SIZED DE SIGN * - synth_1	xc7z020clg400-1 (active)
₽ IP Catalog	Sources Netlist Device C	A Save Constraints
<ul> <li>IP INTEGRATOR</li> <li>Create Block Design</li> </ul>	Q X ♦ - ✓ Internal VREF 0.6V	Select a target file to write new unsaved constraints to. Choosing an existing file will update that file with the new constraints.
Open Block Design Generate Block Design	Drop I/O banks on voltages or the VREF.	● <u>C</u> reate a new file
✓ SIMULATION Run Simulation	VO Port Properties × Clock	Eile type:  B XDC  ✓  File name: pinouts  S <local project="" to=""> ✓</local>
<ul> <li>RTL ANALYSIS</li> <li>&gt; Open Elaborated Design</li> </ul>	General Properties Conf	Select an existing file ✓ select a target file> ✓
✓ SYNTHESIS	Q   X   ♦   •   +	$\overline{\mathbf{v}}$
Run Synthesis	Name > yssarou al(16)	Сапсеl

Figure 10. 52: Creating a new constraints file

After saving the new changes in the pinouts, the IDE will ask you to save to the new constraints file . Just give it a name and then click on *OK*.

#### <u>Bitstream failure</u>



This may cause I/O contention or incompatibility with the board power or connectivity affecting performance, signal integrity or in extreme cases cause damage to the device or the components to which it is connected. To correct this violation, specify all I/O standards. This design will fail to generate a bitstream unless all logical ports have a user specified I/O standard value defined. To allow bitstream creation with unspecified I/O standard values (not recommended), use this command: set\_property SEVERITY {Warning} [get\_drc\_checks NSTD-1]. NOTE: When using the Vivado Runs infrastructure (e.g. launch\_runs Tcl command), add this command to a .tcl file and add that file as a pre-hook for write\_bitstream step for the implementation run. Problem ports: <a href="mailto:eos\_out\_0">eos\_out\_0</a>, <a href="mailto:eos\_out\_0">busy\_out\_0</a>, and <a href="mailto:eos\_out\_0">add this command</a>.

alarm_out_0	OUT		T19	~	34	LVCMOS33*
lousy_out_0	OUT		P16	~	34	default (LVCMOS18)
✓ eoc_out_0	OUT		P15	~	34	default (LVCMOS18)
I eos_out_0	OUT		P18	~	34	default (LVCMOS18)

These must be changed to 3V3.

- alarm_out_0	OUT	T19	¥ 🗌	34	LVCMOS33*
← busy_out_0	OUT	P16	× 🗌	34	LVCMOS33*
C eoc_out_0	OUT	P15	× 🗌	34	LVCMOS33*
eos_out_0	OUT	P18	¥ 🗌	34	LVCMOS33*

#### Figure 10. 54: Error Messages

#### Now for the second error:

0	[DRC UCIO-1] Unconstrained Logical Port: 38 out of 189 logical ports have no user assigned specific location constraint (LOC). This may cause I/O contention
	or incompatibility with the board power or connectivity affecting performance, signal integrity or in extreme cases cause damage to the device or the components
	to which it is connected. To correct this violation, specify all pin locations. This design will fail to generate a bitstream unless all logical ports have a user
	specified site LOC constraint defined. To allow bitstream creation with unspecified pin locations (not recommended), use this command: set_property
	SEVERITY {Warning} [get_drc_checks UCIO-1]. NOTE: When using the Vivado Runs infrastructure (e.g. launch_runs Tcl command), add this command to a .tcl
	file and add that file as a pre-hook for write_bitstream step for the implementation run. Problem ports: ADCresult_out_0[11], ADCresult_out_0[10],
	ADCresult_out_0[9], ADCresult_out_0[8], gpio_io_o_0[17], gpio_io_o_0[16], gpio_io_o_0[15], gpio_io_o_0[14], gpio_io_o_0[13], gpio_io_o_0[12],
	gpio_io_o_0[11], gpio_io_o_0[10], gpio_io_o_0[9], eos_out_0, eoc_out_0 and (the first 15 of 19 listed).

Figure 10. 55: The Second Error

This error was generated because not all external pins were assigned a physical IO pin. As the message suggests, create a .tcl file and pre-hook it to reduce this error into a warning.

So, click on File  $\rightarrow$  new file

Open IP Location Open Recent IP Location	XO
Ne <u>w</u> File	Figure 10. 56: Adding a .tcl File
Diagram × Address Editor ×	XADC_PL_driver.vhd × warningsTCL.tcl ×
G:/Spartan3/SineFunction/ConvertDeg	
Q 🔛 🛧 🥕 🐰 🗉	
<pre>1 set_property SEVERITY {warr</pre>	ning} [get_drc_checks UCI0-1];

Figure 10. 57: Creating a .tcl File

The TCL file is an option under the **new file** selection in the **File** menu. The above statement in the TCL file was written by the author to reduce the errors into warnings for those pins who were not assigned any external IO pins.

Make sure that there is space between the braces and the square brackets and a semicolon at the end!

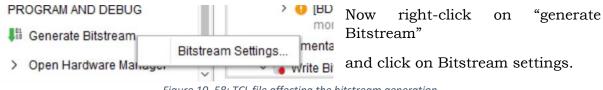


Figure 10. 58: TCL file affecting the bitstream generation

2-	Bitstream	
Project Settings	Specify various settings related to writing bitstream	P-
General Simulation Elaboration Synthesis Implementation	<ul> <li>Configure additional bitstream settings.</li> <li>Write Bitstream (write_bitstream)</li> <li>tcl.pre</li> </ul>	
Bitstream	tcl.post	
a vinx auto 0 xdb	Figure 10. 59: Pre-Hooking the TCL File	
		<pre>Het_property SEVERITY (warning) [get_DRC_checks</pre>
<pre>winx_auto_0_xdb </pre>		

Figure 10. 60: Selecting the new TCL File

Implementation	tcl.pre	G:/Spartan3/SineFunction/ConvertDeg2Ra	
Bitstream	tcl.post		
> IP	-raw_bitfile		
	-mask_file		
ool Settings	-no_binary_bitfile		
Project IP Defaults	-bin_file		
Source File	-readback_file		
Display	-logic_location_file		
WebTalk	-verbose		
Help	More Options		
<ul> <li>Text Editor</li> <li>3rd Party Simulators</li> <li>Colors</li> <li>Selection Rules</li> </ul>			
Shortcuts Strategies Window Behavior	tcl.pre pre-step tcl hook		

Figure 10. 61: confirming the TCL File

Do not forget to click on Apply! Then OK

Click on generate bitstream again and see what happens.

litstream Generation Completed	×	
Bitstream Generation successfully completed.		
Open Implemented Design		
◯ <u>V</u> iew Reports		
Open <u>H</u> ardware Manager		
O Generate Memory Configuration File		
Don't show this dialog again		
OK Cancel		

Figure 10. 62: Successful Generation of the Bitstream File

Now export the project to hardware including the bitstream file.

## Exporting the hardware design

					L.			
	Add So <u>u</u> r	ces		Alt+A		Export	<u>H</u> ardwa	are
	Open Sou	urce File		Ctrl+N		Export	Block D	esign
	I <u>m</u> port					E <u>x</u> port	Bitstrea	m File
	Export			•		Export	Si <u>m</u> ula	tion
S		Figure 10 Export Hardwa development to Export to:	ire platform f cools.	ior softwar	re >	rdware Cancel	×	

Figure 10. 64: Including the Bitstream File

Now launch SDK from within the Vivado IDE environment. *File -> Launch SDK* 

Export

Click on OK for the following window.

Now create an FSBL project.

File  $\rightarrow$  new  $\rightarrow$  application project  $\rightarrow$ 

🔤 New Project

**Application Project** 

Create a managed make application project.

FSBLproject	$\sim$	give a name to the FSBL project	
t location			
	FSBLproject t location		

Figure 10. 65: Naming the FSBL project

#### Click on **NEXT** underneath.

#### Templates

Create one of the available templates to ge

Available Templates:						
Dhrystone						
Empty Application						
Hello World						
IwIP Echo Server						
Memory Tests						
OpenAMP echo-test						
OpenAMP matrix multiplication Demo						
OpenAMP RPC Demo						
Peripheral Tests						
RSA Authentication App						
Zyng DRAM tests						
Zvna FSBL						

Choose Zynq FSBL from list and then click on FINISH underneath.

Make sure that you allow SDK to create the work environment.

Figure 10. 66: Choosing the FSBL Project

Now let's create a C project

File  $\rightarrow$  new  $\rightarrow$  application project  $\rightarrow$  give a name to the project

🔤 New Project

#### **Application Project**

Create a managed make application proje

Project name: C\_project

✓ Use default location

Click on **NEXT** 

#### Templates

Create one of the available templat	This time select <b>hello world</b> from the list and click on <b>FINISH</b> .							
Available Templates:	Figure 10. 67: Selecting the C project							
Dhrystone								
Empty Application								
Hello World								
IWIP Echo Server								
Memory Tests								
OpenAMP echo-test								

The SDK will add the C project to the Vivado project.

∽ 🐸 C_project	Figure 10. 68: Locating t	he Hello World C program
> 🐝 Binaries > 🔊 Includes	• Include the AX	XI GPIO library #include " <b>xgpio.h</b> "
<ul> <li>&gt; Debug</li> <li>&gt; src</li> <li>&gt; helloworld.c</li> <li>&gt; helloworld.</li></ul>	<ul> <li>▶ ps7_cortexa9_0</li> <li>▶ code</li> <li>&gt; ≥ include</li> <li>&gt; ≥ lib</li> <li>&gt; ≥ libsrc</li> </ul>	The xgpio library is in libsrc folder under ps7_cortexa9_0
Xilinx.spec	rary	<ul> <li>&gt; addrps_v1_0</li> <li>&gt; advcfg_v3_5</li> <li>&gt; admaps_v2_3</li> <li>&gt; admaps_v3_6</li> <li>&gt; ageneric_v2_0</li> <li>&gt; agpio_v4_3</li> <li>&gt; agpiops_v3_3</li> <li>&gt; aicps_v3_5</li> <li>&gt; appips_v3_4</li> </ul>

AXI GPIO block library for the PS part is shown in Figure 10.69. This is different from the **gpiops** Library so watch out!

- > generic\_v2\_0
   > gpio\_v4\_3
   > src
   > kgpio\_extra.c
   > xgpio\_g.c
   > xgpio\_i.h
   > xgpio\_intr.c
  - > 🗈 xgpio\_l.h
  - > is xgpio\_selftest.c
  - > le xgpio\_sinit.c
  - > kgpio.c
  - > h xgpio.h
  - 🗟 Makefile

## Initializing the AXI GPIO

In *xgpio\_sinit.c* file copy the lookup().

## XGpio\_Config \*XGpio\_LookupConfig(u16 DeviceId)

**U16 DeviceId** can be found in **xgpio\_g.c** file. There can be <u>only two</u> instances of AXI GPIO. If the application needs three then one has to see whether there is a workaround. The above function call is changed to:

#### AXIgpio1Ptr = XGpio\_LookupConfig(XPAR\_AXI\_GPIO\_0\_DEVICE\_ID);

Figure 10. 70: The AXI GPIO Library

Then write the function call:

# int XGpio\_CfgInitialize(XGpio \* InstancePtr, XGpio\_Config \* Config,UINTPTR EffectiveAddr)

converted to:

#### AXIgpio1success=XGpio\_CfgInitialize(&AXIgpio1,AXIgpio1ConfigPtr,AXIgpio1ConfigP tr->BaseAddress);

AXIgpio1success is of type *int*.

Now use the returned variable value to check whether the initialization has been successful or not.

if(AXIgpio1success != XST\_SUCCESS)

```
{
    return XST_FAILURE;
}
```

Usually if there is a failure here, the program will stop running here.

Repeat the same instructions to AXI GPIO 2.

```
int main()
ſ
    XGpio_Config *AXIgpio0ConfigPtr;
    XGpio_Config *AXIgpio1ConfigPtr;
    int AXIgpio0success,AXIgpio1success;
    XGpio AXIgpio0, AXIgpio1;
    init_platform();
    /* Initialise AXI GPIO 0*/
     AXIgpio@ConfigPtr = XGpio LookupConfig(XPAR AXI GPIO 0 DEVICE ID);
     AXIgpio0success = XGpio_CfgInitialize( &AXIgpio0,AXIgpio0ConfigPtr,AXIgpio0ConfigPtr
             ->BaseAddress);
     if(AXIgpio0success != XST_SUCCESS)
     {
         return XST_FAILURE;
     }
     /* Initialise AXI GPIO 1*/
     AXIgpio1ConfigPtr = XGpio_LookupConfig(XPAR_AXI_GPIO_1_DEVICE_ID);
          AXIgpio1success = XGpio_CfgInitialize( &AXIgpio1,AXIgpio1ConfigPtr,AXIgpio1Confi
                 ->BaseAddress);
          if(AXIgpio1success != XST_SUCCESS)
          {
             return XST_FAILURE;
          }
```

Code Snippet 10. 3: Initializing the AXI GPIOs

Now set the direction of **each channel** in each AXI GPIO block.

## void XGpio\_SetDataDirection(XGpio \*InstancePtr, unsigned Channel, u32 DirectionMask)

The above function is found in *xgpio.c* file. *0* means that particular bit is an <u>output</u> while *1* means that particular bit as <u>input</u>.

```
/*All of AXI GPIO 0 is set as outputs*/
XGpio_SetDataDirection(&AXIgpio0,1,0x00000000);
```

```
/* All of AXI GPIO 1 are set as inputs*/
XGpio_SetDataDirection(&AXIgpio1,1,0xFFFFFFF);
XGpio_SetDataDirection(&AXIgpio1,2,0xFFFFFFFF);
```

Code Snippet 10. 4: Port Direction of each AXI GPIO

The *channel* number can be either channel 1 or channel 2 within each AXI block.

Now read from <u>channel 2 of AXI GPIO 1</u> to know <u>which ADC channel</u> is giving the ADC result from <u>channel 1 of AXI GPIO 1</u> and then output the value in **channel 1** of AXI GPIO 2.

#### u32 XGpio\_DiscreteRead(XGpio \* InstancePtr, unsigned Channel)

the above function returns a value of type *u***32**.

#### ADCchannel = (XGpio\_DiscreteRead(&AXIgpio1,2) & 0x0000001F);

now to write to a channel in one of the AXI GPIOs use:

#### void XGpio\_DiscreteWrite(XGpio \* InstancePtr, unsigned Channel, u32 Data)

```
while(1)
{
   ADCchannel = (XGpio_DiscreteRead(&AXIgpio1,2) & 0x0000001F);
   /* the ADC channel is 5 bits wide*/
   if (ADCchannel == 0x00000003) //reading Vp/Vn "00011"
   {
     ADCresultVp = ((XGpio_DiscreteRead(&AXIgpio1,1) & 0x0000FFF0) >> 4);
      /* result is stored between 15:4 */
     XGpio_DiscreteWrite(&AXIgpio0, 1, ADCresultVp);
      printf("Vp ADC result is:%d",(int)ADCresultVp);//type cast u32 to int
   }
   else if (ADCchannel == 0x00000018) //reading Aux8 "11000"
   {
       ADCresultVAux8 = ((XGpio_DiscreteRead(&AXIgpio1,1) & 0x0000FFF0) >> 4);
      /* result is stored between 15:4 */
     XGpio_DiscreteWrite(&AXIgpio0, 1, ADCresultVAux8);
      printf("Aux8 ADC result is:%d",(int)ADCresultVAux8); //type cast from u32
   }
                                                            //to int type
}
```

Code Snippet 10. 5: Reading and Writing from AXI GPIO

In the code above, the *read function* together with *bit-masking* was used so that if there are any other 1s which are not of interest will be removed. So, read the channel number first, then use it as a reference to know which ADC channel is transmitting data at the output. Send them on UART and at the same time display the result on LEDs through the AXI GPIOs to access the pins located on the PL side.

<ul> <li>C_project</li> <li>C_project_bsp</li> <li>FSBLproject</li> <li>FSBLproject_b</li> </ul>	New Go Into Open in New Window		Figure 10. 71: Board Support Package To be able to communicate with PC, <b>UART</b>							
	<ul> <li>Copy</li> <li>Paste</li> <li>Delete</li> <li>Source</li> <li>Move</li> <li>Rename</li> </ul>	Ct <b>1</b> must be made as the default UART not UART 0.								
	Import       Export       Refresh       Close Project       Close Unrelated Projects	€. • (	Board Support Package Settings         Control various settings of your Board Support Package.         Overview       Configuration for OS:       standalone         v drivers       Name       Value       Default							
	Build Configurations         Run As         Debug As         Compare With         Restore from Local History         Board Support Package Settings         Re-generate BSP Sources		ps7_cortexa9_0	hypervisor_guest stdin stdout zynqmp_fsbl_bsp > microblaze_exception > enable_sw_intrusive_	false ps7_uart_0 none ps7_coresight_0 ps7_uart_0 ps7_uart_1	false none ✓ none comp_0				

## Reconfiguring the Board Support Package

Board Support Package Settings

#### **Board Support Package Settings**

Control various settings of your Board Support Package.

Make sure to wait for the update to take place because it takes a few minutes.

0

✓ Overview	Configuration for OS	standalone				
standalone	Configuration for OS:	standalone				
✓ drivers	Name	Value	Default			
ps7_cortexa9_0	hypervisor_guest	false	false			
	stdin	ps7_uart_1	none			
	stdout	ps7_uart_1	none			
	zynqmp_fsbl_bsp	false	false			
	> microblaze_exception	s false	false			
	> enable_sw_intrusive_p	r false	false			

## Creating the Boot Image File

<ul> <li>C_project</li> <li>C_project</li> <li>FSBLproje</li> <li>FSBLproje</li> <li>FSBLproje</li> </ul>	×	Open in New Window Copy Paste Delete Source Move Rename Import	The Zynq 7 can only be programmed in two ways, either using a JTAG cable or via the ARM Cortex A9 by loading a bootloader file on SD card. The ARM A9 reads the boot image file from the SD card and loads the PL part of the Zynq SoC. Right click on the C project and select <b>create boot image</b> .
		Export Build Project Clean Project Refresh Close Project Close Unrelated Projects	Figure 10. 72: Creating the Boot image file
ä Target Conn >	2 1	Build Configurations Run As Debug As Compare With Restore from Local History C/C++ Build Settings Generate Linker Script Chanae Referenced BSP Create Boot Image	
Architecture:			
Basic Secu		file O Import from existing BI	F TILE
Output BIF fi	le pa	ath: G:\Z-TURN_V12_20171030	\Zynq7020\Multiple_XADC_input_data_shared_PS_PL\Multiple_XADC_input_data_shared_PS_PLsdk\C_project\bootimage\C_project.bif
UDF data:			
Split		Output format: BIN ~	
Output path:		G:\Z-IURN_V12_20171030	\Zynq7020\Multiple_XADC_input_data_shared_PS_PL\Multiple_XADC_input_data_shared_PS_PLsdk\C_project\bootimage\BOOT.bin 
Boot image p	artit	ions	
\Z-TURN_V12	_20	171030\Zynq7020\Multiple_XA	\Multiple_XADC_input_data_shared_PS_PL\Multiple_XADC_input_data_shared_PS_PLsdk\FS8Lproject\Debug\FS8Lproject.elf DC_input_data_shared_PS_PL\Multiple_XADC_input_data_shared_PS_PLsdk\BlockDesign_wrapper_hw_platform_0\ <u>BlockDesign_wrapper.bit</u> DC_input_data_shared_PS_PL\Multiple_XADC_input_data_shared_PS_PLsdk\C_project\Debug\ <u>C_project.elf</u>

Make sure that there are *three files* in the ISO file. Click on *Create boot image* underneath.

Now look for the generated boot image file in the appropriate folder. Copy and paste on the SD card.

	5	Paste shortcut	to 🕶	to 👻	•	folder	r	· •	listory	Invert	selection			
Clipbo	ard			Orgar	ise		New		Open	Sele	ct			
Jick access		Name		~			Date modified		Туре		Size			
Jesktop	*	() BOOT					24/11/2018 16:20	)	PowerISO File		4,133	в КВ		
Downloads	*	C_project	:				24/11/2018 16:20	)	PowerISO File		1	KB		

Figure 10. 73: Locating the BOOT image file in File Explorer within Windows

## <u>Event driven sampling of multiple XADC channels from the Programmable</u> <u>Logic</u>

The aim of this chapter is to sample XADC channels using event driven technique from a VHDL module. Using this technique, the programmer has full control of the XADC block and therefore could determine which ADC channel to sample and when the ADC result is available.

In previous chapters, the XADC was configured to do continuous sampling and all that was needed from the designer's side is to sample the channel address available at the output of the XADC block and route the ADC data on the **do\_out** bus to the output port. This is convenient, it is challenge to write an XADC driver that is able to operate the XADC block in event-driven mode.

During experimentation, it was discovered that at power-on-reset, **the XADC needed some finite time to settle, before the first sample.** This should be in the form of a small delay of 100ms.

#### End-of-Conversion Signal

In previous code the **DRDY** signal was being monitored while the **EoC** signal was not. This was recommended by **UG480 on page 74 section Dynamic Reconfiguration Port Configuration (DRP)**. However, taking a closer look at the timing diagram of the **Event-Driven Sampling**, one will soon realize that it was **imperative** to check **EoC** signal. By doing so, the XADC was given enough time to finish the previous conversion and start a new sample. UG480, shows that the XADC needs **four clock cycles between conversions** especially if the next conversion is going to be done from a different channel.

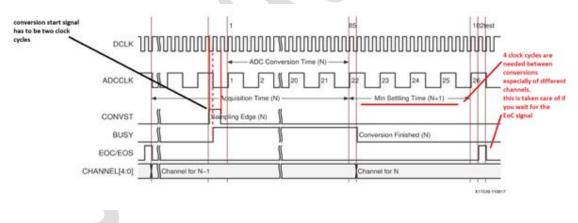


Figure 11. 1: Event Driven Mode Timing Diagram

## <u>CONVST signal</u>

From the timing diagram it is clearly seen that **two clock cycles** are needed for **CONVST** signal to be effective. This gives time for the **BUSY** signal to become **logic 1**.

- So, check the **BUSY** signal to be at logic 1 after driving the **CONVST** signal to logic low.
- Make sure that the **BUSY** signal goes low again.

• Then wait for the EoC signal to go high.

Now read the channel ADC data from the data bus. This is described in the next section.

## <u>Reading ADC data from the data bus</u>

Now to read the ADC result from XADC, one must go through the following steps according to timing diagram of Figure 11.2. This was taken from page 75 of UG480:

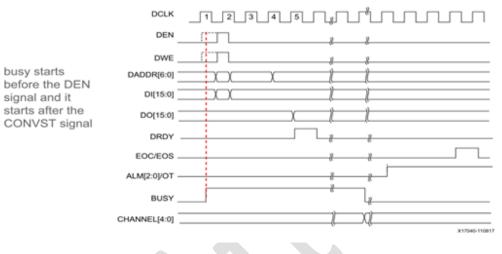


Figure 11. 2: DRP Timing Diagram

So, after the **EoC** is asserted, it is time to read the ADC result by following these steps:

- Assert the **D\_en** signal to logic 1 for one clock cycle
- The *D\_we* input of the XADC should be hardwired to ground
- Wait for **DRDY** signal of XADC to go high
- Get the ADC data

Obviously the above must be implemented in a state machine, and therefore one must make sure to store the ADC result in a register.

Since more than one XADC channel must be read, with my VHDL code, another state-machine was created to controls which channel the controller will sample. The following sections show the block diagram and explain the VHDL code.

#### <u>The Block Diagram</u>

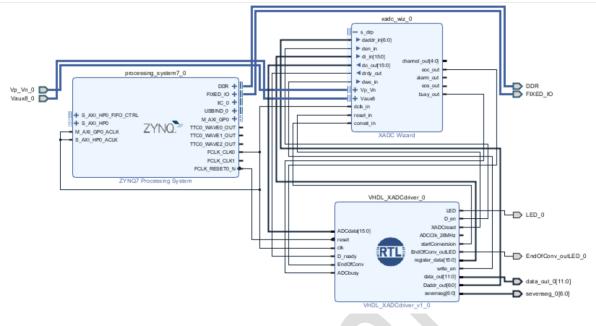


Figure 11. 3: System Block Diagram

One should go through all the steps described in previous chapters to draw the circuit as shown in Figure 11.3. The XADC configuration will be explained next.

	XADC Wizard (3.3)						
	Occumentation IP Location						
	Show disabled ports	Component Name xadc_wiz_0					
		Basic ADC Setup Alarms Channel S					
		Interface Options					
		O AXI4Lite      DRP O None					
	+ s_drp + Vp_Vn channel_out[4:0] =	Startup Channel Selection					
	+ Vaux8 alarm_out - dolk_in eos_out -	Simultaneous Selection					
	ereset_in busy_out - convst_in	O Independent ADC					
		Single Channel					
		Channel Sequencer					

Figure 11. 4: XADC Basic Configuration Page

After double-clicking the XADC wizard block, the *basic* page pops up. Enable the *DRP* radio button and the Channel Sequencer button as shown in Figure 11.4. Scroll the horizontal bar to the right to reveal the *Timing Mode* section as shown in Figure 11.5. Tick the *Event Mode* radio button. Leave the frequencies as they are.

Basic	ADC Setup	Alarms	Channel Sequencer	Summary		
		Timing N	Node			
		0	Continuous Mode 💿 E	vent Mode		
		DRP Tim	ning Options			
		$\checkmark$	Enable DCLK			
		DCL	K Frequency(MHz)	100	$\otimes$	[8.0 - 250.0]
		ADO	Conversion Rate(KSPS)	1000	$\otimes$	[39.0 - 1000.0]
		Acq	uisition Time (CLK)	4	~	
		Clo	ck divider value = 4			
		ADO	Clock Frequency(MHz) =	25.00 Scrol	the bar	to the right



Basic	ADC Setup	Alarms Channel Seque	ncer Summa	y	
	[7 - 1020]				
		Analog Sim File Options	]		
Arbiter		Sim File Selection	Default	~	
		Analog Stimulus File	design	$\otimes$	
		Sim File Location	J		scroll down
		Waveform Type	CONSTANT	~	~~
		Frequency (KHz)	1.0		[0.1 - 240.38]
		Number of Wave	1		[1 - 1000]

Figure 11. 6: Analog Sim File Options

Leave the Analog Sim File Options section as it is.

Component Name xadc_wiz_0									
Basic ADC Setup Alarms Channel Sequencer Summ	nary								
AVIASTREAM Options									
AXI4STREAM Options									
Enable AXI4Stream									
Enable Axi4Stream									
FIFO Depth 7 [7 - 102	01								
	-1								
Control/Status Ports									
🖌 reset_in 📃 Temp Bus 📃 JTAG Arbiter									
Event Mode Trigger									
Lvent mode mgger									
convst in Convstclk in									

Figure 11. 7: Event Mode Settings

Leave the *AXI4STREAM* section as it is but make sure to tick the *reset in* square and the *convst in* radio button.

Leave the ADC setup page as it is.

nsic AD	C Setup	Alarms	Channel Sequencer	Summary		
	IT Alarm (V	/olts)			JX Alarm (Volts)	
Lower	0.97		[0.0 - 1.05]	Lower	1.75	
Upper	1.03		[0.0 - 1.05]	Upper	1.89	
	RAM Alarm	n (Volts)		VCCP	int Alarm (Volts)	
Lower		n (Volts)	[0.0 - 1.05]	Lower		
	0.95	n (Volts)	[0.0 - 1.05]		0.95	

Figure 11. 8: Alarms Setup page

Remove all the ticks in the alarms setup page as shown in Figure 11.8. make sure to scroll down and disable the remaining alarms.

	Compon	ent Name xad	lc_wiz_0				
	Basic	ADC Setup	Alarms	Channel	Sequencer	Summary	
	VP/VN	$\Diamond$	V	]			
	VREFP						
	VREF	1					
	vauxp0	/vauxn0					
	vauxp1	Nauxn1					
	vauxp2	Nauxn2					
	vauxp3	Wauxn3					
	vauxp4	Wauxn4					
	vauxp5/vauxn5						
	vauxp6	Wauxn6					
	vauxp7	/vauxn7					
	vauxp8	Avauxn8 🗘		]			

Figure 11. 9: Selecting the Analogue Channels

In the *Channel Sequencer* page, select the channels for the application. In this project, only the external Vp/Vn and *Auxiliary channel 8* are selected.

That's it! Now click on OK to finish the XADC setup. Now its time to reveal the VHDL code.

#### The VHDL driver

#### Code Snippet 11. 1: Entity Declaration

Code snippet 11.1 shows the entity declaration. This shows all the inputs and the outputs of the VHDL driver.

Code Snippet 11. 2: Declaring the internal signals

Code snippet 11.2 show all the internal signals used. Note the state machine declaration.

```
process (clk,reset)
begin
if reset = '0' then master_current_state <= Ms0;
elsif rising_edge(clk) then master_current_state <= Master_next_state;
end if;
end process;</pre>
```

Code Snippet 11. 3:

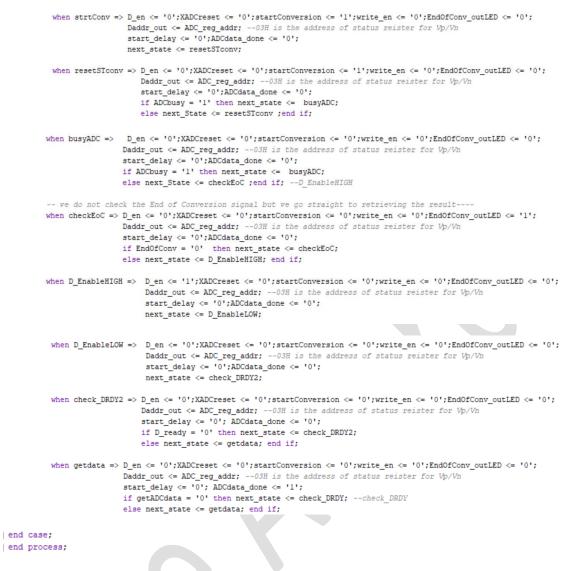
The process shown in Code Snippet 11.3 is used to control the master state machine. As can be seen, it's clock is the 100 MHz clock.

```
process(master_current_state,ADCdata_done)
begin
case master_current_state is
when Ms0 => ADC_reg_addr <= "0000011"; getADCdata <= '0';
    master_next_state <= Ms1;
when Ms1 => ADC_reg_addr <= "00000011"; getADCdata <= '1';
    if ADCdata_done = '1' then master_next_state <= Ms2;
    else master_next_state <= Ms1; end if;
when Ms2 => ADC_reg_addr <= "0011000"; getADCdata <= '0';
    master_next_state <= Ms3;
when Ms3 => ADC_reg_addr <= "0011000"; getADCdata <= '1';
    if ADCdata_done = '1' then master_next_state <= Ms0;
    else master_next_state <= Ms3; end if;
end case;
end process;</pre>
```

Code Snippet 11. 4: Master SM

Code Snippet 11.4 shows the syntax for the master state machine (SM). There are two handshake lines between the master state machine and the slave state machine. The slave state machine is the one that is directly interfaced to the XADC block. The master SM issues a signal (*getADCdata signal*) to trigger the slave SM, then it waits for the slave SM to finish (*ADCdata\_done signal*). There is also the channel address denoted as *ADC\_reg\_addr*, that selects which ADC channel should be sampled.

```
process(clk, reset)
                 begin
                 if reset = '0' then current_state <= resetXADC;</pre>
                 elsif clk'event and clk = 'l' then current state <= next state;
                 end if;
                 end process;
                                                      process(current_state, ADCbusy, EndOfConv, D_ready, end_delay, getADCdata)
begin
case current state is
when resetXADC => D_en <= '0';XADCreset <= '0';startConversion <= '0';write_en <= '0';EndOfConv_outLED <= '0';
                  Daddr_out <= ADC_reg_addr; --03H is the address of status reister for Vp/Vn
                  start_delay <= '0';ADCdata_done <= '0';</pre>
                  next_state <= exitReset;</pre>
when exitReset => D_en <= '0';XADCreset <= '1';startConversion <= '0';write_en <= '0';EndOfConv_outLED <= '0';
                  Daddr_out <= ADC_reg_addr; --03H is the address of status reister for Vp/Vnp
                  start_delay <= '0';ADCdata_done <= '0';</pre>
                  next_state <= check_DRDY;</pre>
                  D_en <= '0';XADCreset <= '0';startConversion <= '0';write_en <= '0';EndOfConv_outLED <= '0';</pre>
when delay =>
                  Daddr_out <= ADC_reg_addr; --03H is the address of status reister for Vp/Vnp
                  start_delay <= '1';ADCdata_done <= '0';</pre>
                  if end_delay = '1' then next_State <= check_DRDY;
                  else next_state <= delay; end if;</pre>
when check_DRDY => D_en <= '0';XADCreset <= '0';startConversion <= '0';write_en <= '0';EndOfConv_outLED <= '0';
                   Daddr_out <= ADC_reg_addr; --03H is the address of status reister for Vp/Vn
                    start_delay <= '0';ADCdata_done <= '0';</pre>
                   if D_ready = '0' then next_state <= strtConv;
                   else next_state <= check_DRDY; end if;
```



Code Snippet 11. 5: Event Driven XADC driver

So in the first three states, the SM will first reset the XADC block, then it will go into a delay of 100 mS. At this point the SM will monitor the DRDY signal, and once this goes low, the SM will trigger the conversion process. The SM will then wait for the busy line to go high at which point the start-conversion signal will be reset. The SM waits for the end-of-conversion signal to go high, at which point, the data enable signal will be set by the SM for one clock cycle. Then the SM will monitor the DRDY signal again to go high. After that, the SM will enable the 16-bit ADC data and copies it from the data bus.

```
process(start_delay,clk)
variable count : integer;
begin
if start_delay = '0' then count := 0; end_delay <= '0';
elsif rising_edge(clk) then count := count + 1;
    if count < 10000000 then end_delay <= '0';
    elsif count >= 10000000 then end_delay <= '1';
    end if;
if count >= 10000000 then count := 0; end if;
end if;
end process;
```

data\_out <= ADCdata(15 downto 4) when ADC\_reg\_addr = "0011000" else "000000000000";

Code Snippet 11. 6: Delay of 100 ms

As already stated, in this project two ADC channels were employed and therefore these had to be displayed on different display-media. Code Snippet 11.6 shows the 100 mS delay process and also a statement that assigns the raw 12-bit data to output LEDs. It must be stressed here the versatility of the FPGA as opposed to a microcontroller. In a microcontroller, the raw data had to be shifted to the left by 4 places, however with FPGAs, all one has to do is to extract the bits of interest and assign them to a new register already placed according to their binary weight!

The ADC data is only shown on the LEDs when the address of the Auxiliary channel 8 is available at the output of the XADC block. This is very convenient.

2
3 4
5
6
7 8
9

Code Snippet 11. 7: Seven Segment Driver

Code Snippet 11.7 shows a convenient way to convert the voltage of a signal from the ADC into decimal levels from 0 to 9. However, to use the relational operators in VHDL, one has to use integer data types. So, a concurrent statement was included to convert the input ADC data from unsigned to integer. Then the new variable or register could be used in the when-else statement.

That should wrap everything up, in the next chapter, the same project will be extended to include the Processing System of the Zynq 7 System-on-Chip.