Control AXI GPIO IP - I/Os with software on MicroBlaze

In this tutorial we are going to add a AXI GPIO IP to the Blockdesign and connect two external LEDs to the AXI GPIO IP Core. After generating the Bitstream we export the design to Vitis and control the LEDs with software(C/C++) running on the MicroBlaze. In the same way you can connect any other I /Os external or internal the Chip.

This tutorial is based on the the reference design of the TE0717 Board and Vitis 2021.2 was used.

- 1. First open the prebuilt reference design with the "_create_win_setup.cmd" script and open the block design.
- 2. Add a second "AXI GPIO IP" from the IP catalog to the block design.



Figure 1: Adding AXI GPIO IP Core

3. Configure the IP by double-clicking on the IP. Make the changes and click "ok".

🝌 Re-customize IP		
AXI GPIO (2.0)		
🛿 Documentation 🛛 🖨 IP Location		
Show disabled ports	Company Name and and a	
	Board IP Configuration	
	GPIO	
	All Inputs	
	All Outputs	
	GPIO Width 2	🛞 [1 - 32]
	Default Output Value 0x00000000	[0x0000000,0xFFFFFFF]
	Default Tri State Value 0xFFFFFFF	[0x0000000,0xFFFFFFF]
	Enable Dual Channel	
• s_axi_aresetn	GPIO 2	

Figure 2: AXI GPIO IP Core Configuration



4. Right click on the port and choose "Make external" in the context menu

Figure 3: Make the AXI GPIO Ports external for the Carrier LEDs

5. Run the connection automation. The AXI GPIO IP Core gets connected to the MicroBlaze via the AXI Interface.

1							
Diagram × Address Editor × Address Map	×						
$ \mathbf{Q}_{1} \mathbf{Q}_{1} $	🕞 🖌 🖻 😽 C 🗟 🗄	Default View 🗸					
* Designer Assistance available. Run Connection Au	tomation						
A Run Connection Automation							
Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configurat right.							
Q ≭ ≑	Description						
 ✓ All Automation (1 out of 1 selected) ✓ ✓ ati_gpio_1 Connect Slave interface (/axi_gpio_1/S_AXI) to a selected Master address space. 							
✓ ⊕ S_AXI Options							
	Master interface	/microblaze_0 (Periph) 🗸					
	Bridge IP	/microblaze_0_axi_periph 🛛 🗸					
	Clock source for driving Bridge IP	/clk_wiz_0/clk_out1 (100 MHz)	~				
	Clock source for Slave interface	Auto	~				
	Clock source for Master interface	/clk_wiz_0/clk_out1 (100 MHz)	~				

Figure 4: Run Connection Automation6. Now the address mapping in the address editor should be set like:

		olare beginerit	master sase i la aress	. ange	indoter ingrit id al coo
~ 岩 Network 0					
✓ ♥ /microblaze_0					
Image: Microblaze_0/Data (32 address bits : 4G)					
1 /axi_gpio_0/S_AXI	S_AXI	Reg	0x4000_0000 🖉	64K -	0x4000_FFFF
/axi_gpio_1/S_AXI	S_AXI	Reg	0x4001_0000	64K •	0x4001_FFFF
/axi_intc_0/S_AXI	s_axi	Reg	0x4120_0000 🖉	64K -	0x4120_FFFF
/axi_quad_spi_0/AXI_LITE	AXI_LITE	Reg	0x44A0_0000 0	64K •	0x44A0_FFFF

Figure 5: Address Editor - AXI GPIO 1

7. Next, the constraints for the two leds have to be set in the constraint file.



Figure 6: LEDs constraints

Now you are all set in Vivado and you can build the bitstream + export the project to Vitis with the following command in the TCL-console(only possible in the script-based reference design from trenz).

Manual steps would be to Generate the Bitstream and export the hardware platform.



Figure 7: Build bitstream and export project

9. After that was successful you can test the functionality of the VIO Core by programming the FPGA and controlling the onBoard LED2. Otherwise continue with Step 10.



Figure 8: Program FPGA and control onboard LED through the VIO IP Core.

- 10. Now you can build the Vitis project with the provided C/C++-Applications with the following command.
 - Manual steps would be to Open Vitis(Tools Launch Vitis IDE) and import the project from the project directory.



Figure 9: Build Vitis project with built vivado design

11. In Vitis open the hello_te0717 application:



12. Add the following lines of code to also make both of the LEDs blink:



Figure 11: hello_te0717 adding code for controlling carrier LEDs

13. Save and Build the project

14. Right-Click on the application and choose "Run Configuration"

🔒 Explorer 🛛					E 🔽 🕴 🕴 🗆	🖻 he	ello_te0717.c ⊠
∽ 📰 hello_te	0717	7_system [TE0717-01-P1C-5-	A]			23	* WHETHER IN AN ACTION
× ፼ hellc > ╬ B > ⊚ Ir	-	New Move To System Project	>			24 25 26	* OR IN CONNECTION WITH * SOFTWARE. *
> 🗁 R	Ď	Paste	Ctrl+V			27 28	* Except as contained i * in advertising or oth
Ƴ 🎥 si > 🔓	×	Refresh	Delete F5			29 30	<pre>* this Software without *</pre>
> [> [24 24	Import Sources Export as Archive				31 32 33®	***************************************
> [1 > ⊯_i % h 15 > ⊯_ide		Build Project Clean Project				34 35	* helloworld.c: simple
	8	Generate Linker Script C/C++ Build Settings				36 37 38 39	<pre>* This application con * PS7 UART (Zyng) is n * bootrom/bsp configur *</pre>
≚ hellc		Team	>			40	* *
Final Spi_bot Final Spi_bot Final Spi_bot Final Spi_bot		Run As	>	Ę	1 Launch Hardware (Sin	gle Ap	oplication Debug)
		Debug As	>	- <mark>1</mark> 2 10 10	2 Launch SW Emulator	(Single	e Application Debug)
		Properties Alt+	Alt+Enter	GDB	3 Launch Hardware (Sin	gle Ap	oplication Debug (GDB)) 3
					Run Configurations		
						47 48 49	<pre>#include <stdio.h> #include "platform.h"</stdio.h></pre>

Figure 12: Run Configuration

15. Since we already programmed the FPGA, uncheck the following:

Name: Debugger_hello_te0717-Default				
😢 Main 🗖 Application 📀 Target Setup 🛛 🕬 Arguments 📼 Environ				
Hardware Platform:	\${sdxTcfLaunchFile:project=hello_te0717;fil			
Bitstream File:	_ide/bitstream/Tutorial0717_01_100_1C_8			
PL Device:	Auto Detect			
		Summary:		
Reset entire system		 Following operations will be perform 1. The following processors will be reform 1) microblaze_0 2. All processors in the system will be 1) microblaze_0 (S:\Tutorial0717\v 		

Figure 13: Run Configuration - Uncheck
 16. Click on Apply and Run to execute the program on the MicroBlaze. You Should now see the LEDs of the module and carrier blinking in sequence. Also if you open up a serial connection to the board with a program like PuTTY, you should see output