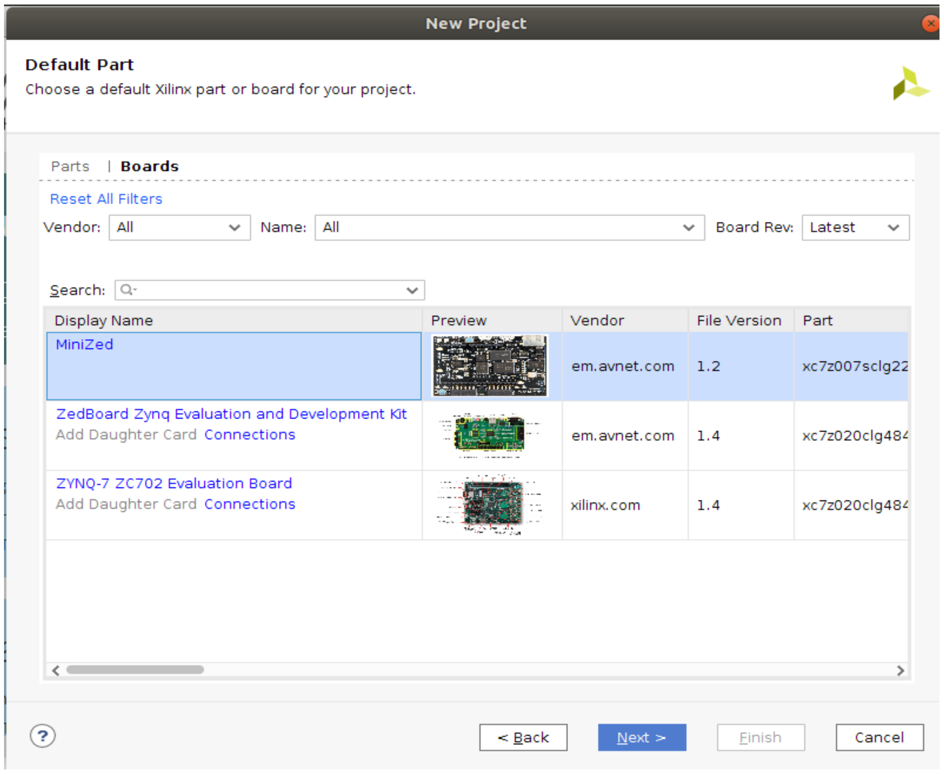
This tutorial describes the steps to create a *hardware platform* for the MiniZed board which only uses the processing system (PS), and simple application examples which blink an LED and communicate with a connected system (such as the lab computers or your PC) via the UART port (and in turn the USB port).
You are recommended to keep the following documents handy while working with the MiniZed platform and Zynq system:
[1] https://www.avnet.com/wps/wcm/connect/onesite/1945b4c1-4e40-46dd-92c1-46329304e185/MiniZed-HW-UG-v1-0-V1_0.pdf?MOD=AJPERES&attachment=false&id=1573009082950
Be careful of the auto inserted blanks in the above URL when you copy then into a web browser.

[2] https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf
[2] is a big document (1843 pages), but it is a reference document, and should be the first document to check before starting to use a peripheral. For example, a big source of confusion when first starting to work with the PL is, that it does not work by default unlike the PS. Section "2.4 PS–PL Voltage Level Shifter Enables" describes why this is so, and how to make it work.

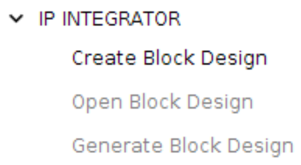
A Quick Tutorial Application for PS

1. Start Vivado and choose Create Project. Enter project name, location etc. and click Next.
2. Choose RTL project, and make sure the box "Do not specify sources at this time" is selected.
3. In the next page, select the Boards tab, choose MiniZed, and finish the creation.

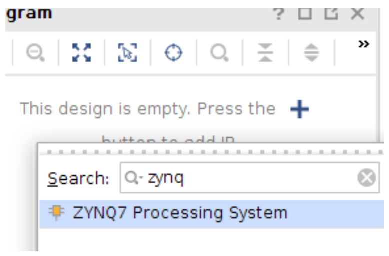


Display Name	Preview	Vendor	File Version	Part
MiniZed		em.avnet.com	1.2	xc7z007sclg22
ZedBoard Zynq Evaluation and Development Kit Add Daughter Card Connections		em.avnet.com	1.4	xc7z020clg484
ZYNQ-7 ZC702 Evaluation Board Add Daughter Card Connections		xilinx.com	1.4	xc7z020clg484

4. Click “Create Block Design” on the left-hand side, or under the “Flow” menu, and choose default options.

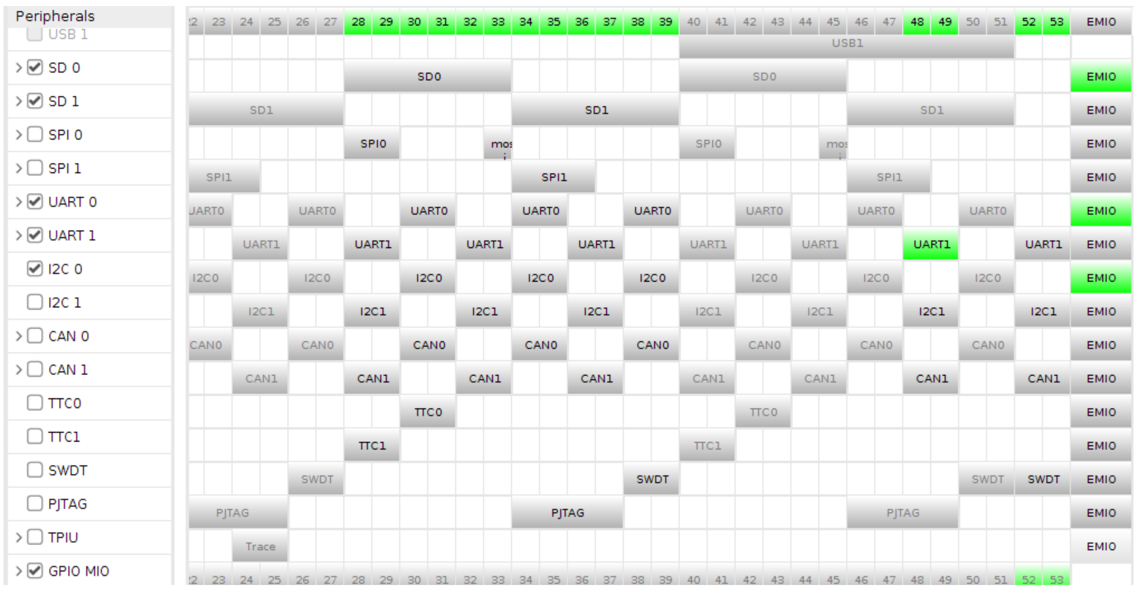


5. Click “Open Block Design”. There should be an empty diagram on the GUI.
6. Click the Add IP button (+ button) on the diagram, and choose ZYNQ7 Processing System.



7. Click on “Run Block Automation” which should have just popped-up. You can also right-click the diagram and choose it. Use the defaults (“apply board preset” should be selected) and click OK. This configures the processing system (PS) according to the specifications of the board we have previously selected (MiniZed).

   For example, if you double click the Zynq7 Processing System, under Peripheral I/O pins, you should see that UART1 is connected to pins 48-49, and UART0 to EMIO (which connects to the PL side). A search for “48” at the user guide of MiniZed [1] reveals that UART#1 is indeed connected to MIO pins 48-49.


Peripherals
- [ ] USB 1
- [✓] SD 0
- [✓] SD 1
- [ ] SPI 0
- [ ] SPI 1
- [✓] UART 0
- [✓] UART 1
- [✓] I2C 0
- [ ] I2C 1
- [ ] CAN 0
- [ ] CAN 1
- [ ] TTC0
- [ ] TTC1
- [ ] SWDT
- [ ] PJTAG
- [ ] TPIU
- [✓] GPIO MIO

22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 EMIO
USB1
SD0 SD0 EMIO
SD1 SD1 SD1 EMIO
SPI0 SPI0 EMIO
SPI1 SPI1 SPI1 EMIO
UART0 UART0 UART0 UART0 UART0 UART0 UART0 UART0 EMIO
UART1 UART1 UART1 UART1 UART1 UART1 UART1 UART1 EMIO
I2C0 I2C0 I2C0 I2C0 I2C0 I2C0 I2C0 I2C0 EMIO
I2C1 I2C1 I2C1 I2C1 I2C1 I2C1 I2C1 I2C1 EMIO
CAN0 CAN0 CAN0 CAN0 CAN0 CAN0 CAN0 CAN0 EMIO
CAN1 CAN1 CAN1 CAN1 CAN1 CAN1 CAN1 CAN1 EMIO
TTC0 TTC0 EMIO
TTC1 TTC1 EMIO
SWDT SWDT SWDT SWDT EMIO
PJTAG PJTAG PJTAG EMIO
Trace EMIO

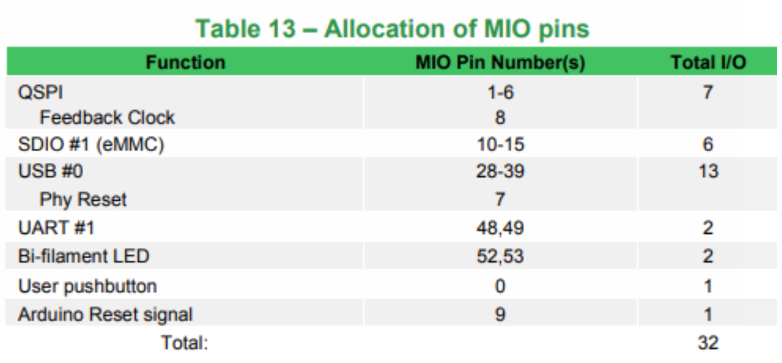
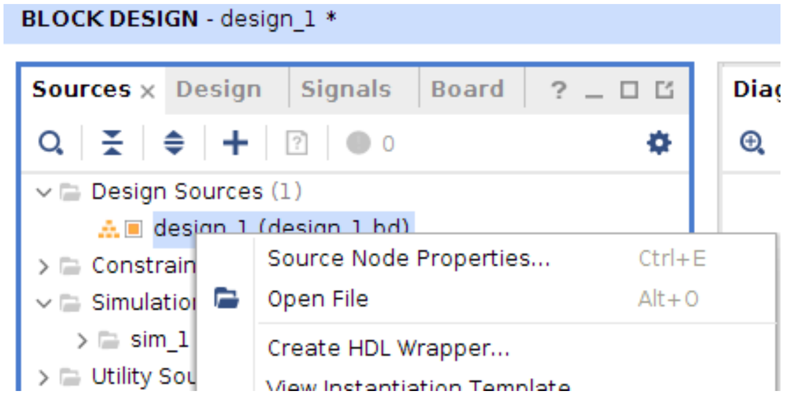


Table 13 – Allocation of MIO pins

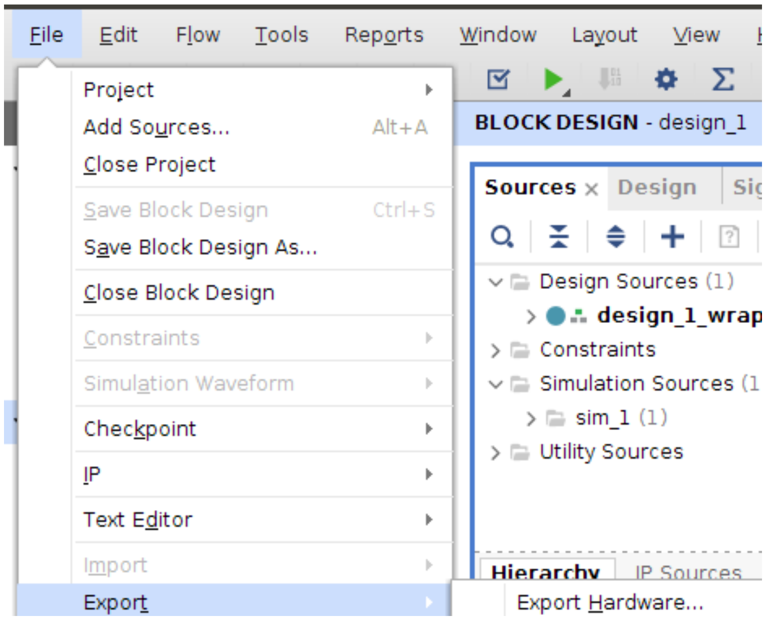
Function	MIO Pin Number(s)	Total I/O
QSPI	1-6	7
Feedback Clock	8	
SDIO #1 (eMMC)	10-15	6
USB #0	28-39	13
Phy Reset	7	
UART #1	48,49	2
Bi-filament LED	52,53	2
User pushbutton	0	1
Arduino Reset signal	9	1
Total:		32

Also note that the Bi-filament LED is connected to pins 52-53, which are automatically configured in the block design (GPIO MIO pins 52-53) thanks to the block automation.

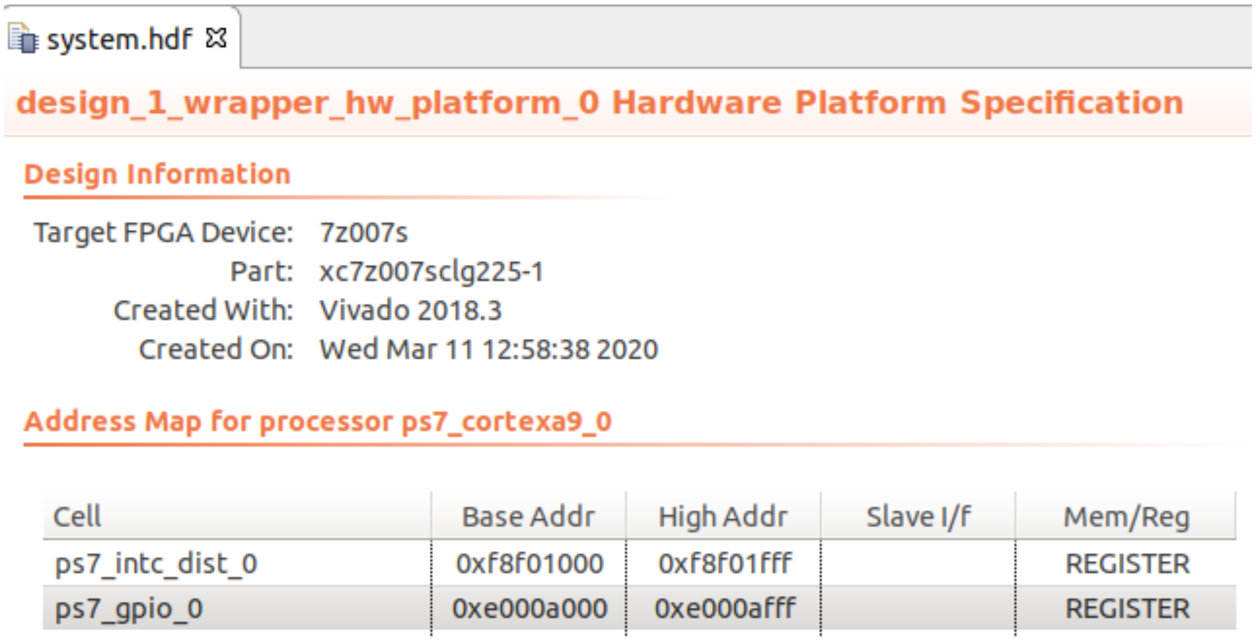
8. Right-click your design under Sources, and click "Create HDL Wrapper...", and let Vivado manage the wrapper.



9. Since we only want to test basic functionality, we will not be using the FPGA (PL) side of the SoC. So click "Generate Bitstream" to directly generate the BitStream for the hardware platform (which only contains the Zynq PS block). This step might take a few minutes. You do not need to open the implemented design, so you can press "Cancel" when Vivado offers you to open it when the bitstream is generated.
10. Click File -> Export -> Export Hardware **after** the bitstream is generated. Make sure that "Include bitstream" checkbox is selected.



11. Click File -> Launch SDK. If you have used the default locations at the previous step and included the bitstream, just press OK to launch the SDK where we will program the ZYNQ PS.
12. In the SDK, notice the address map for the processor. We are interested in this example to interact with the LEDs, which is connected to the PS GPIO, whose registers have a base address of 0xe000a000 in this case.


system.hdf

design_1_wrapper_hw_platform_0 Hardware Platform Specification

Design Information

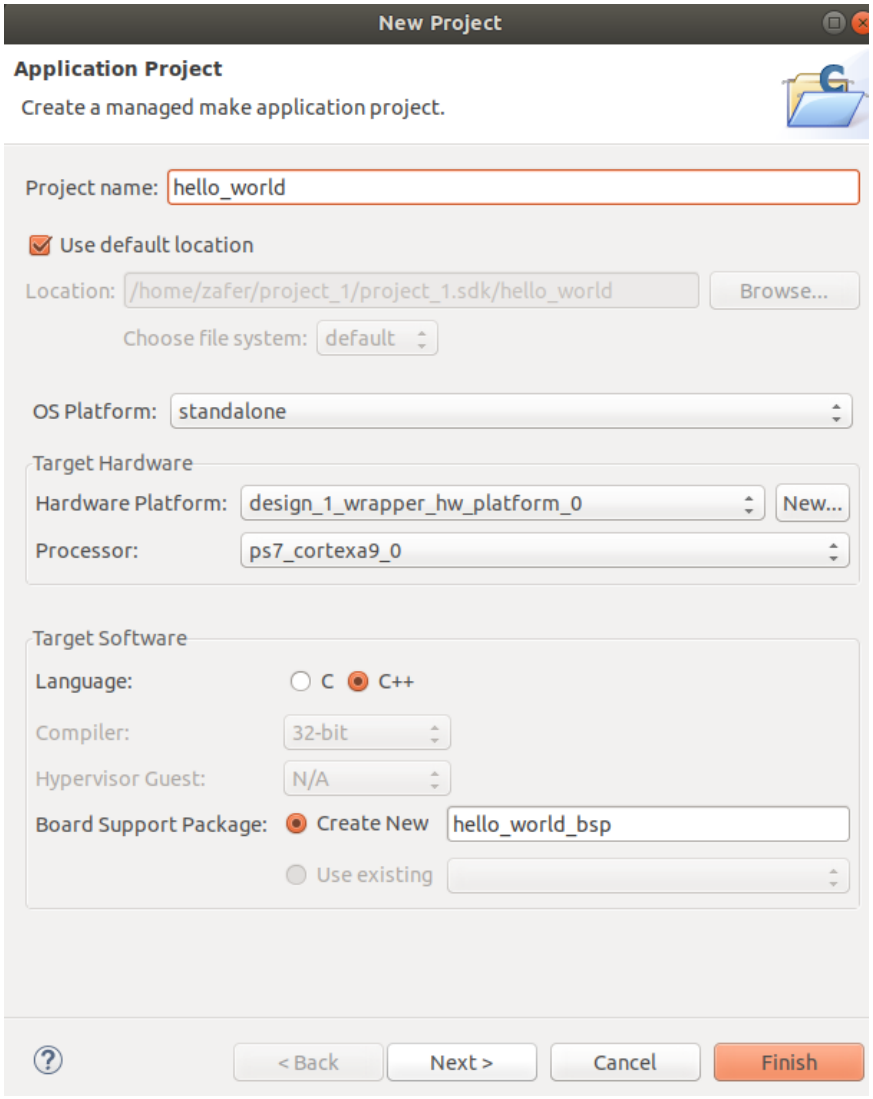
Target FPGA Device:	7z007s
Part:	xc7z007sclg225-1
Created With:	Vivado 2018.3
Created On:	Wed Mar 11 12:58:38 2020

Address Map for processor ps7_cortexa9_0

Cell	Base Addr	High Addr	Slave I/f	Mem/Reg
ps7_intc_dist_0	0xf8f01000	0xf8f01fff		REGISTER
ps7_gpio_0	0xe000a000	0xe000afff		REGISTER

13. In the SDK, click File -> New -> Application Project. Choose a project name, a project language (in this case I will choose C++), and make sure that the design (for which we exported the bitstream) is selected as the "Hardware Platform". We want to run the system bare-metal, so choose standalone as the OS platform. We also need a new

board support package, which will be created automatically at this step. Board support package provides peripheral drivers and any other libraries which you can use in your code. Choose Empty Application in the next page and then Finish.



New Project

Application Project

Create a managed make application project.

Project name: hello_world

[✓] Use default location

Location: /home/zafer/project_1/project_1.sdk/hello_world

Choose file system: default

OS Platform: standalone

Target Hardware

Hardware Platform: design_1_wrapper_hw_platform_0

Processor: ps7_cortexa9_0

Target Software

Language: C [✓ C++]

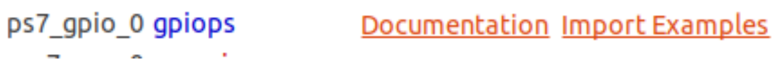
Compiler: 32-bit

Hypervisor Guest: N/A

Board Support Package: [✓ Create New] hello_world_bsp

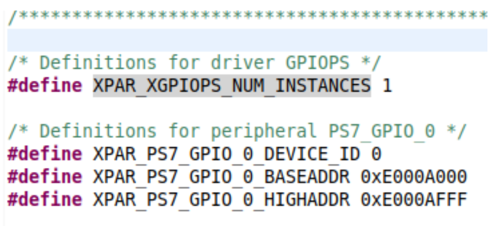
Use existing

14. In the board support package, open “system.mss” if not already opened, and notice the documentation and examples for ps7_gpio_0 driver, which we will use to toggle the LED.



ps7_gpio_0 gpiops Documentation Import Examples

If you open for example xgpiops_hw.h, you will see that the register offsets are automatically defined for us, which you would need to look-up from the datasheet of the device otherwise. Similarly “../ps7_cortexa9_0/include/xparameters.h” under BSP defines many things such as the address mapping, so you do not do these manually. Remember the address map from system.hdf? These are defined for us automatically here!

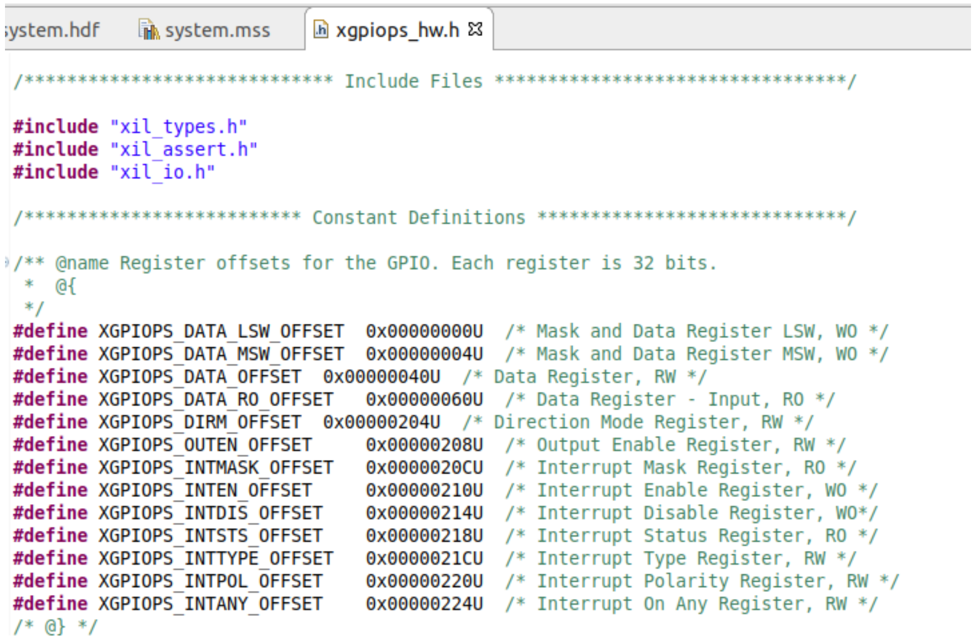


```
/**********************************************

/* Definitions for driver GPIOPS */
#define XPAR_XGPIOPS_NUM_INSTANCES 1

/* Definitions for peripheral PS7_GPIO_0 */
#define XPAR_PS7_GPIO_0_DEVICE_ID 0
#define XPAR_PS7_GPIO_0_BASEADDR 0xE000A000
#define XPAR_PS7_GPIO_0_HIGHADDR 0xE000AFFF
```

Since no (sane) person has time to do everything from scratch, we will use one of the provided examples to blink the LEDs, but you should know that you can find all the drivers inside the BSP if needed.



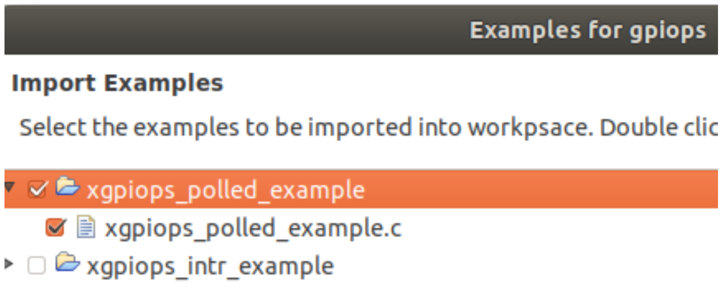
```
/***************************** Include Files *********************************/

#include "xil_types.h"
#include "xil_assert.h"
#include "xil_io.h"

/************************** Constant Definitions *****************************/

/** @name Register offsets for the GPIO. Each register is 32 bits.
 *  @{
 */
#define XGPIOPS_DATA_LSW_OFFSET  0x00000000U  /* Mask and Data Register LSW, WO */
#define XGPIOPS_DATA_MSW_OFFSET  0x00000004U  /* Mask and Data Register MSW, WO */
#define XGPIOPS_DATA_OFFSET  0x00000040U  /* Data Register, RW */
#define XGPIOPS_DATA_RO_OFFSET   0x00000060U  /* Data Register - Input, RO */
#define XGPIOPS_DIRM_OFFSET  0x00000204U  /* Direction Mode Register, RW */
#define XGPIOPS_OUTEN_OFFSET     0x00000208U  /* Output Enable Register, RW */
#define XGPIOPS_INTMASK_OFFSET   0x0000020CU  /* Interrupt Mask Register, RO */
#define XGPIOPS_INTEN_OFFSET     0x00000210U  /* Interrupt Enable Register, WO */
#define XGPIOPS_INTDIS_OFFSET    0x00000214U  /* Interrupt Disable Register, WO*/
#define XGPIOPS_INTSTS_OFFSET    0x00000218U  /* Interrupt Status Register, RO */
#define XGPIOPS_INTTYPE_OFFSET   0x0000021CU  /* Interrupt Type Register, RW */
#define XGPIOPS_INTPOL_OFFSET    0x00000220U  /* Interrupt Polarity Register, RW */
#define XGPIOPS_INTANY_OFFSET    0x00000224U  /* Interrupt On Any Register, RW */
/* @} */
```

15. Choose the simpler xgpiops_polled example after clicking “Import Examples” for ps7_gpio_0 inside system.mss.



16. Now copy everything inside the xgpiops_polled_example.c file into your main.cc file (or simply use this application project). We need to set the pins for the LEDs though, since the SDK does not automatically know about this.
Modify the code as shown below:

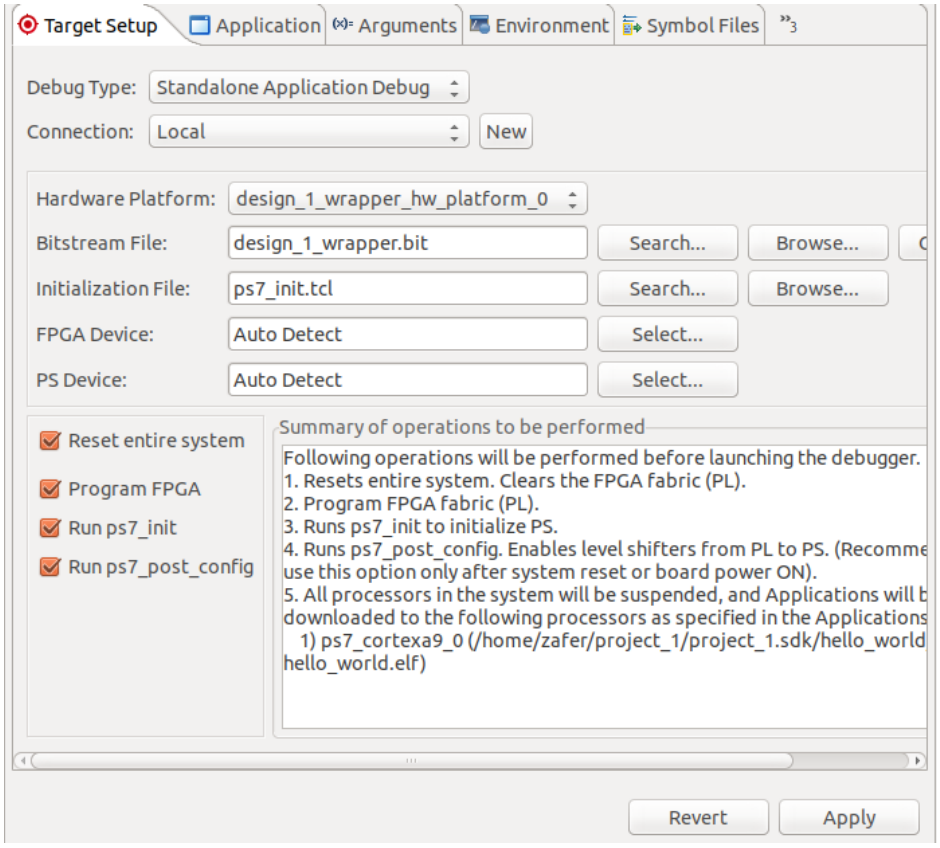
```
Input_Pin = 0; // user pushbutton
Output_Pin = 52; // bi-filament LED pin 1 (we'll only blink one of them)

/* You can remove below commented block
 * switch (Type_of_board) {
    case XPLAT_ZYNQ_ULTRA_MP:
        Input_Pin = 22;
        Output_Pin = 23;
        break;

    case XPLAT_ZYNQ:
        Input_Pin = 14;
        Output_Pin = 10;
        break;
    }*/
```

Notice that we took this information from the MiniZed user manual (check step 7 of this document).

17. To run the code on MiniZed, press the little downward arrow next to the play button, and choose Run Configurations. Target Setup window should look something like as shown below:



Debug Type: Standalone Application Debug
Connection: Local
Hardware Platform: design_1_wrapper_hw_platform_0
Bitstream File: design_1_wrapper.bit
Initialization File: ps7_init.tcl
FPGA Device: Auto Detect
PS Device: Auto Detect

- [✓] Reset entire system
- [✓] Program FPGA
- [✓] Run ps7_init
- [✓] Run ps7_post_config

Summary of operations to be performed

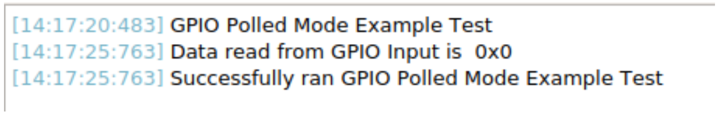
```
Following operations will be performed before launching the debugger.
1. Resets entire system. Clears the FPGA fabric (PL).
2. Program FPGA fabric (PL).
3. Runs ps7_init to initialize PS.
4. Runs ps7_post_config. Enables level shifters from PL to PS. (Recomme
use this option only after system reset or board power ON).
5. All processors in the system will be suspended, and Applications will b
downloaded to the following processors as specified in the Applications
  1) ps7_cortexa9_0 (/home/zafer/project_1/project_1.sdk/hello_world
hello_world.elf)
```

Make sure that you can see the FPGA Device and the PS Device by clicking Select…, and select the Application project that we have created in the Application tab. If everything went well, you should see the LED blinking for the specified number of times.

18. If you want to communicate with the MiniZed, the easiest way to do so is via UART. For this, you can load the UART example from ps7_uart_1 in system.mss, and play around with it. Make sure that UART1 is used in the example, since this is the one that is

actually connected on Minized, by using the definition
"XPAR_XUARTPS_1_DEVICE_ID", and not "XPAR_XUARTPS_0_DEVICE_ID":
#define UART_DEVICE_ID   XPAR_XUARTPS_1_DEVICE_ID

Also, you can select this UART port as the default stdout, so that you can directly print to the output. If this is not working, click "Modify this BSP's Settings" in system.mss, select "standalone" under Overview, and choose ps7_uart_1 for both stdin and stdout, since we are using these on the MiniZed. Now you can see textual output using a serial communication program (a SDK terminal is also provided with Xilinx SDK).



```
[14:17:20:483] GPIO Polled Mode Example Test
[14:17:25:763] Data read from GPIO Input is  0x0
[14:17:25:763] Successfully ran GPIO Polled Mode Example Test
```

19. If you want to use the other peripherals, Xilinx provides drivers (and examples for most) of the peripherals, so they are a good starting point.