Lab 1: polling and interrupts

Download the reference workspace

If you have the Z710 board download the workspace
from here lab1_polling_interrupts.zip or peripherals_lab_sw_blank-main.7z
If you have a Nexys, download from project_nexys_lab1_2023.zip

Open XSDK and, when asked for a workspace, point to the .sdk folder that you have extracted from the archive. If needed, use the menu File->Import->General->Existing project...
and select the same .sdk directory as root folder, to import the design_1_wrapper_hw_platform_0.

Analyzing your system

Check the system composition, connectivity, address ranges. You can use the .hdf file in the workspace for the addresses.
The view of the block design is available here design_lab1_blockdesign_2023.pdf

Create a new application template

To create a new application on the SDK, click on **File**, then select **New -> Application Project**.
A guided procedure for creating a new project will be displayed. Specify the application name (without blank spaces), eg: hello_world.
Make sure the field Target Hardware points to the correct hardware design (in our case, design_1_wrapper_hw_platform_0).
Then select **Create New** on the **Board Support Package**.
The tool will automatically populate the name of the board support package according to the name of the application project.
Once the previous steps have been completed, in the panel Project Explorer there will be two new folders:

- a folder hello_world containing binary files and sources files (.c and .h),
- a folder hello_world_bsp, i.e. the board support package.

Writing the software - accessing the GPIOs with polling

Use a while loop that periodically checks the input peripherals and updates the LEDs with the latest changed value in the inputs. Please recall the previous lab for how to interact with data registers.

Execute the application on the MB system implemented on the board

The FPGA chip on the board must be configured with the bitstream created using Vivado. This can be done as follows:

- make sure the Zybo is turned on and connected to the host PC through the micro USB cable
- from the menu **Xilinx Tools** select **Program FPGA**.
-
- bcMake sure the field **Hardware Platform** refers to the design **design_1_wrapper_hw_platform_0**.
- Select **bootloop** on **ELF File to Initialize in Block RAM**, then click on **Program**.

Once the board has been programmed with the file bit the green LED with label DONE should be on. Then right-click on the folder hello_world, then choose **Run As** and finally **Run Configurations**.
Double-click the Xilinx C/C++ application (GDB) on the left panel, a new configuration is created.
Verify the run configuration, it should point to the .elf file that you want to execute.
Select **Run** to execute.

Analyze your binary file

Check the binary .elf file in your application project derived from compilation. Find the instructions executing the main() and try to understand them. Analyze the placement of the stack.

Writing the software - accessing GPIOs with interrupts

When multiple interrupts have to be handled, interrupt lines must be multiplexed in a single interrupt; it is also necessary to keep track of the information relating to who, among the interrupt lines, has generated the event. This task is usually performed by a specific block, called interrupt controller. In the provided system the xps_intc block performs the task of interrupt controller; multiple interrupt sources are connected to the interrupt controller, which is in turn connected to the MicroBlaze.
Please check the datasheets for the GPIOs and for the interrupt controller. While managing the interrupts you'll have to access the following registers of GPIO blocks and the interrupt controller.
Interrupt controller:
Datasheet at the following link (look at Pag 15 for the Register Space and at page 28 for the Programming sequence)
https://www.xilinx.com/support/documentation/ip_documentation/axi_intc/v4_1/pg099-axi-intc.pdf

- ISR - Interrupt Status Register - Offset 0x0 - Read / write register used to read which interrupts are active and pending
- IER - Interrupt Enable Register - Offset 0x8 - Read / write register used to enable selected interrupts
- IAR - Interrupt Acknowledge Register - Offset 0xC - Write-only register u*sed for clear interrupt requests
- MER - Master Enable Register - Offset 0x1C - Read / write register used to enable the IRQ request to the processor

GPIO controller:
Datasheet at the following link
https://www.xilinx.com/support/documentation/ip_documentation/axi_gpio/v2_0/pg144-axi-gpio.pdf

- IP IER - IP Interrupt Enable Register - Offset 0x0128 - Read / write register used to enable / read interrupts in each channel
- GIER - Global Interrupt Enable Register - Offset 0x011C - Read / write register used to enable interrupt request to interrupt controller
- IP ISR - IP Interrupt Status Register - Offset 0x120 - Read / write register containing the interrupt status for each channel - The register has TOW (Toggle-On-Write) operation, ie the value of a specific register bit is "inverted" whenever the value 1 is written on it.

- Define the interrupt handler to be considered as such by the compiler.

  Working in C language, the use of interrupts requires two steps
  a) a way to make the linker know how to specify the address of our ISR routine in the interrupt vector
  b) a way to specify to the compiler that our function will be an ISR, and must therefore be filled in appropriately (with the correct management of stacks and registers, and the call to RTID at the end of the execution).
  These two steps can be done simply by using the following function prototype (https://en.wikipedia.org/wiki/Function_prototype) that exploits the keyword __attribute__ (https://gcc.gnu.org/onlinedocs/gcc-3.2/gcc/Function-Attributes.html):

```
void myISR (void) __attribute__ ((interrupt_handler));
```

  myISR is the name of the interrupt handler function, you can choose any name.

- Implement your interrupt handler. You have to:

- check which interrupt sources are involved (check the list of registers above to see which registers have to be read)
- serve the involved one(s): the LEDs must be updated with the value corresponding to the latest changed input peripheral
- acknowledge the interrupt (to the interrupt controller and to the peripheral, check list of registers above)
- Setup the interrupts in the main() You must:

- Enable interrupt registers in each GPIO
  - globally (inside the GPIO controller) enable the interrupts management (GIER register)
  - individually enable the interrupts of each channel (IPIER register)
- Enable interrupt generation in the interrupt controllers

- globally (inside the INTC) enable the interrupts management (MER register)
- individually enable each interrupt (IER register)

- Enable interrupt receiving in the Microblaze:
  The MicroBlaze processor supports an external interrupt source (connected to the input port Interrupt), other systems may offer different support to interrupts. See the MicroBlaze reference manual (page 62):
  https://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf
  The processor reacts to interrupts if the *Interrupt Enable (IE)* bit in the *Machine Status Register (MSR)* is set to 1 (MSR is *described at page 22*). This operation is performed by the built-in function:
  microblaze_enable_interrupts ();

  When an interrupt occurs, the instruction in the execution stage is completed, while the instruction in the *decode* stage is replaced by a branch to an interrupt vector. The return address from the interrupt routine (the value of the PC associated with the instruction in the decode stage at the time of the interrupt) is automatically loaded into the general-purpose register R14. In addition, the processor also disables future interrupts by setting the IE bit to zero in the MSR. The IE bit is automatically set back to one when the RTID instruction is executed.
- Check the interrupt functionality.
- Create long interrupt service routines, what happens when interrupt overlap?

Analyze the interrupt implementation

Check the assembly. How is the interrupt handler implemented? How do you get to execute it in the program code? Where is the interrupt vector in the Microblaze instruction memory?

Change the optimization level for the polling implementation

Change the C/C++ build settings for your application. Right-click on the application project, select C/C++ settings and look for the Optimization tab. You should change the optimization level to -O2.
Check again the functionality and look for any problem appearing.