2 - Reacting to interrupts with Linux on the terasic DE10 development board

Table of Contents

Required Software & Hardware

!! We are using a different Linux image than in the previous tutorial !!

Linux System image: [DE10 Standard Linux SD Card Image]
DE10 Board
Micro SD Card with 8GB+
Windisk Imager 32 (or alternative tool to flash image to sd card)
Intel SoC EDS Version 16.0 (others might work, 16.0 was verified working)
Quartus 17.1 (others might work, 17.1 was verified working)

Installing Linux

- Flash image to SDCard with Windisk Imager 32 or alternative Tool.
- Insert the SD Card in the board.
- Set the MSEL Pins of the DE10 board to 01010 to boot from the SD Card.

Testing interrupts

The Linux installation will flash the FPGA with a .rbf (raw binary file) after it has booted. The provided IP already includes an interrupt sender for the Push Buttons. We will use this to test the basic system functionality, before we create our own IP. To react to interrupts, we must compile the kernel module and include it into the linux system.

Connect to the linux system with a serial terminal and navigate to the `/tutorial_files` directory. In this directory there are two header files: one for the interrupt IDs of the peripherals and one for the physical addresses of the hardware that can be accessed.

Switch to the `pushbutton_irq_handler` folder. Here you can find the c code for the kernel module as well as a makefile to create the kernel module. To build the kernel module execute the `make` command in the command line. Now we have to include the created kernel module `pushbutton_irq_handler.ko`. This is done with the following command:

```
insmod pushbutton_irq_handler.ko
```

Now you can press any of the buttons which will then create an interrupt. Every time an interrupt occurs, the LED counter will increase by one.

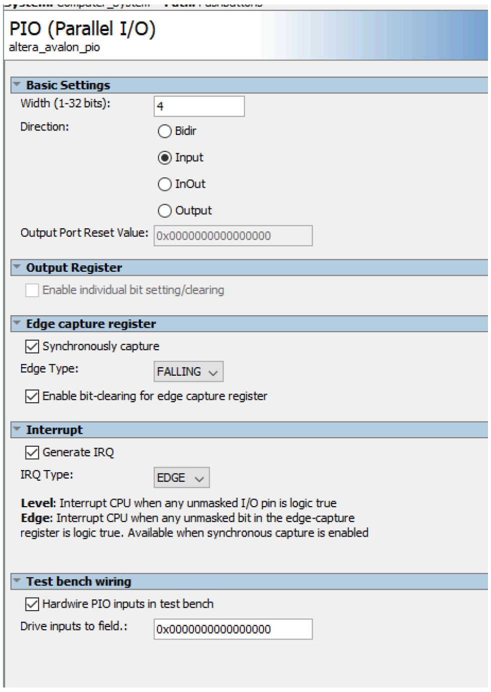
Edit Quartus Project to support interrupts

Now we want to use our own IP with this linux distribution, to create a custom IP that can trigger interrupts. While we could change the h files of the linux distribution to match the addresses and Interrupt line numbers that we choose, it is easier to use the existing settings in the platform designer when creating the system.

We will extend the project from the previous tutorial, so use this as a starting point.

Open the Platform Designer.

Add a Pio block for the Push Buttons



Connect reset, clock, slave to lightweight master, irq to interrupt receiver:

You can configure the HPS component to provide 64 general-purpose FPGA-to-HPS interrupts, allowing soft IP in the FPGA fabric to trigger interrupts to the MPU's generic interrupt controller (GIC). The interrupts are implemented through the following 32-bit interfaces:

• f2h_irq0—FPGA-to-HPS interrupts 0 through 31
• f2h_irq1—FPGA-to-HPS interrupts 32 through 63

The FPGA-to-HPS interrupts are asynchronous on the FPGA interface. Inside the HPS, the interrupts are synchronized to the MPU's internal peripheral clock (periphclk).

We connect to the `f2h_irq0` interface, as our interrupt line number is smaller than 31. Connecting to both interfaces will cause an error at a later point, when trying to create the required header files!

Export the connection under the name `button_pio_external_connection`.

If we have a look at the header file on the `linux sytem interrupt_ID.h` we can see that the interrupt line for the pushbuttons is 73.

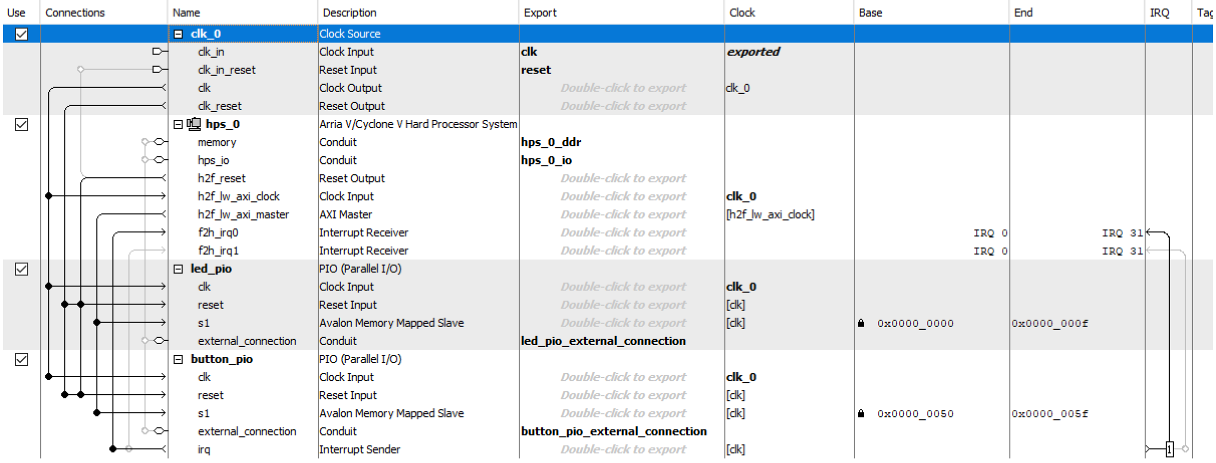
```
  GNU nano 2.2.6              File: interrupt_ID.h

/* FPGA interrupts (there are 64 in total; only a few are defined below) */
#define INTERVAL_TIMER_IRQi                                               72
#define KEYS_IRQ                                                          73
#define FPGA_IRQ2                                                         74
#define FPGA_IRQ3                                                         75
#define FPGA_IRQ4                                                         76
#define FPGA_IRQ5                                                         77
#define AUDIO_IRQ                                                         78
#define PS2_IRQ                                                           79
```

This means we have to set the interrupt line number to 1. The interrupt lines 72-135 of the ARM controller are reserved for interrupts that originate from the FGPA. This means that the interrupt number 1 is equal to 73 on the HPS side.

Next, we will have a look at the assigned base addresses of the LEDs and the Push Buttons in the `address_map_arm.h` file:

```
#define LEDR_BASE              0x00000000
#define HEX3_HEX0_BASE         0x00000020
#define HEX5_HEX4_BASE         0x00000030
#define SW_BASE                0x00000040
#define KEY_BASE               0x00000050
#define JP1_BASE               0x00000060
#define JP2_BASE               0x00000070
#define PS2_BASE               0x00000100
```

Assign the base address of `0x0000_0050` to the pushbuttons and `0x0000_0000` to the LEDs. Keep both fixed by clicking on the lock symbol next to it. The final result should look like this:



Select Generate-> Generate HDL to generate the system.

Ensure to include the generated IP if you have created a new project.

We need to add the keys to the system. First, we add the input to the top-level module:

```
        ///////// KEY /////////
        Input [3:0]   KEY,
```

Next, we add the internal connection:

```
Wire [3:0] fpga_button_pio;
assign fpga_button_pio = KEY;
```

And finally assign it:

```
.button_pio_external_connection_export (fpga_button_pio)
```

Before compilation ensure, that the Keys have the right pin assignments (Assignments->Pin Planner)

KEY[0]	Unknown	PIN_AJ4	3B
KEY[1]	Unknown	PIN_AK4	3B
KEY[2]	Unknown	PIN_AA14	3B
KEY[3]	Unknown	PIN_AA15	3B

Now we can compile the design.
After the compilation is completed, we program the FPGA with the resulting .sof file. The order of the programming is of importance! First, linux needs to be booted, then the FGPA needs to be programmed and then the kernel module should be included. If everything went well we can now see the exact same thing as before: every time we press a button an interrupt is triggered and the soc reacts to it by increasing the LED counter. Only this time our custom IP is creating the interrupts. If we adjust the Verilog code to only support 3 buttons, we can definitely ensure that our custom IP is creating the interrupts.

```
    ///////// KEY /////////
    input       [2:0]   KEY,
  ...

   wire [2:0] fpga_button_pio;
```