Tutorial for Hardware Interrupts with the Xilinx Zynq Platform Using Linux

Alexander Zoellner

August 21, 2019

1 Introduction

Interrupts are an essential part for communication between hardware and software. Using interrupts prevents wasting precious CPU clock cycles on polling as well as improving the systems' responsiveness.

The Xilinx Zynq 7000 system-on-chip (SoC) series offers a number of interrupt ports between programmable logic (PL) - the FPGA side - and the processing system (PS) - the CPU. This requires a logic implementation on the FPGA side, which can be either one of the interrupt capable IP cores of Xilinx or a custom logic. On the CPU(s), software for configuring and receiving interrupts has to be executed. This software can be run bare-metal (without an operating system) or with an operating system such as Linux. The former is well covered by online tutorials and is pretty straight forward to get a working system. Accomplishing the same using Linux, however, is not that well covered despite being an integral part even when using trivial logic such as timers.

Since it required quite some effort collecting information from user manuals, forum posts and online documentation in order to get interrupts to work with Linux, I would like to spare others (and myself in the future) the trouble of having to repeat the same all over.

Section 2 is for the impatient and those who just want to get the job done without caring too much about the details. Therefore, presented information are kept to a minimum and as generally applicable to other systems as possible. A basic understanding of the work flow with Xilinx's Vivado, compiling a Linux image for a particular board and basic understanding of Linux device drivers are required.

2 Quick Solution

Getting PL interrupts to work in Linux requires a number of steps which can be summarized as following.

- Enable PL-PS interrupt ports in the Processing System in Vivado
- Connecting a port of an IP core to the enabled port
- Adding a devicetree entry
- Registering the interrupt in the device driver

Code 1 shows the entry for the devicetree for the device using the interrupt. The first line is the name of the node which can be chosen arbitrarily as long as it does not conflict one of the existing nodes. The second line (*interrupt-parent*) references the *phandle* of the interrupt handler of the system. The value between the braces has to match the one of the interrupt handlers phandle. The example shows the value *0x4* since the devicetree binary (.dtb) has been decompiled back to the devicetree source (.dts) where all strings have been replaced by actual numbers.

The last line (*interrupts*) defines the actual interrupt to be used. In case of the general interrupt control of arm (which is used by the Zynq platform) interrupts are defined using three cells. The first cell is always set to zero (0) for PL interrupts (see Section **??**).

The second cell is the interrupt number, which is the interrupt id of the PS which the PL connects to minus 32. The Zynq platform offers a total of 16 interrupt ports with interrupt ids 91 to 84 and from 68 to 61. The ports are connected lowest (0) to highest index (15) to the interrupt ids in the same manner, i.e. 61 to 91. Thus, port 0 connects to interrupt id 61, which in turn results in the interrupt number 29 (61 - 32).

The last cell is used for defining the type and level of the interrupt, i.e. the condition which have to be met for an interrupt to occur. For PL interrupts the cell can either have the value 1 or 4. The former is *low-to-high edge triggered* and the latter *active high level-sensitive*. In most cases 1 is what you want for interrupts.

SPI (shared processor interrupts), interrupt number, type and level

```
    hw_timer_0 {
        interrupt-parent = <0x4>;
        interrupts = <0 29 1;
    }
```

Code 1: Devicetree entry

After compiling the devicetree source back to the devicetree binary the interrupt has to be registered by the device driver.

Code 2 shows a minimal example for requesting the interrupt number and registering an interrupt handler. The first two lines are the required header files which have to be included for devicetree related function calls. The lower part after (...) can be placed in either the *init* or *open* (prefered) function. The second argument of *of_find_node_by_name* has to match the name used in the devicetree entry. Subsequently, the node is passed to *irq_of_parse_and_map* for registering the interrupt number. The return value is the index of the interrupt which gets displayed in the first column when calling *cat /proc/interrupts*. Lastly, *request_irg* installs the interrupt handler which gets called when an interrupt occurs.

```
    #include <linux/of_irq.h>
    #include <linux/of.h>

        (...)

    struct device_node *np;

    np = of_find_node_by_name(NULL, "hw_timer_0");
    timer_irq = irq_of_parse_and_map(np, 0);
    request_irq(timer_irq, &timer_intr_handler, 0, "hw_timer_intr", NULL);
```

Code 2: Device driver entry

Please make sure that the hardware (PL) does not generate any interrupt events prior to successfully registering the interrupt.