# Tutorial: Creating Peripherals with Interrupts in Altera's SoPC Builder

Shangru Li and Stephen A. Edwards, April 2012

1. Open SoPC Builder
2. Create a new component
3. Select the component VHDL file to add: *irComp.vhd* for this tutorial.
4. Move to the *Signals* tab and change the following things:

| Signal | Interface | Signal Type |
|---|---|---|
| clk | clock | clk |
| reset_n | clock | reset_n |
| irq | new interrupt sender | irq |

5. Move to *Interfaces* and look at the "interrupt_sender" section. Change the associated addressable interface to be *avalon_slave_0*.

Next, add the CPU, SRAM, JTAG, and the component you just created. Name it *irqsource*. Auto assign the address. You also need to assign the IRQ priorities for the components because both JTAG and your component can generate interrupts. You can change the number in the IRQ column (0 has the highest priority), or you can also use "auto assign IRQ."

## The irComp.vhd file

This component is just a counter that gives an interrupt (set IRQ to be 1) every 0.7 seconds. The CPU writing to this component resets IRQ to 0. You can ignore the other processes in this component since only these two are related to interrupts.

```
process (clk)
begin
  if rising_edge(clk) then
    if reset_n = '0' then
      counter <= (others => '0');
    else
      counter <= counter +1 ;
  end if; end if;
end process;

process (clk)
begin
  if rising_edge(clk) then
    if reset_n = '0' then
      irq <= '0';
    else
      if counter = "00000000000000000000000001" then
        irq <= '1';
      elsif write = '1' and chipselect = '1' then
        irq <= '0';   -- important: reset the irq
  end if; end if; end if;
end process;
```

## The irq-main.c file

After enabling the interrupt, the *main* function sits in an infinte loop. When the interrupt comes, the *irqhandler* function will be called and print a message on the console.

```
#include <stdio.h>
#include "system.h"
#include <sys/alt_irq.h> // the irq functions
#include <io.h>
#include <alt_types.h>

static void irqhandler (void * context, alt_u32 id)
{
  printf ("interrupt occurred\n");
  IOWR_16DIRECT(IRQSOURCE_BASE, 0, 0); // reset request
}

int main()
{
  printf("main() started\n");
  alt_irq_register( IRQSOURCE_IRQ, NULL,
                    (void*)irqhandler ); // register the irq

  for (;;) {} // Do nothing forever

  return 0; // Never executed
}
```

## Notes

1. If you didn't name the component *irqsource*, you can refer to the *system.h* file to discover the name of the IRQ and use it as a parameter to the *alt_irq_register* function. The *system.h* file is in syslib/debug/system_description. It define the addresses and IRQ, if any, for all your components.
2. It is important to use the IRQ for VGA control. In *vga_raster*, when *vcount* reaches the bottom of the screen, there will be hundreds of cycles (the back- and front-porch periods) that can be used to change the image on the screen (for instance, the position of the sprites). This cures the glitches many of you were experiencing with the ball in lab 3. A good strategy is to have *vga_raster* interrupt the CPU and the end of the frame so the CPU can update the contents of the screen.