G7 Infrared Application Note

Kenan Kigunda
February 24, 2014

Introduction

This tutorial explains how to control the infrared emitter (LTE 4208) and receiver (R3208E)—both of which are currently in stock—using the DE0 Nano board. This procedure can also be adapted for the DE2 board. After following this tutorial you will be able to turn on the emitter using a pushbutton on the board, and to check its output on the receivers using an oscilloscope.

Hardware Setup

Emitters (LTE 4208)

- Power diodes using GPIO power pins at 5 V.
- Each emitter diode has maximum current $I_D = 20$ mA.
- Each emitter diode has voltage drop $V_D = 1.2$ V.
- Limit current through diodes with resistor R_D, with value chosen so that the current does not exceed 20 mA.
- Turn diodes on and off using 2N4401 transistor (in stock) connected to GPIO output pin at 3.3 V.

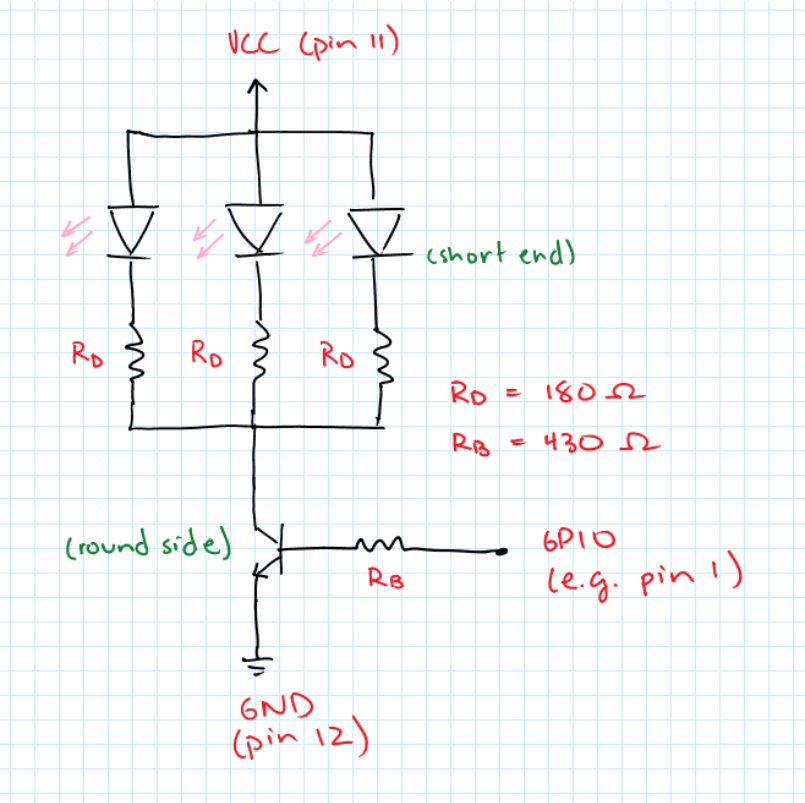

Figure 1 Infrared emitter circuit

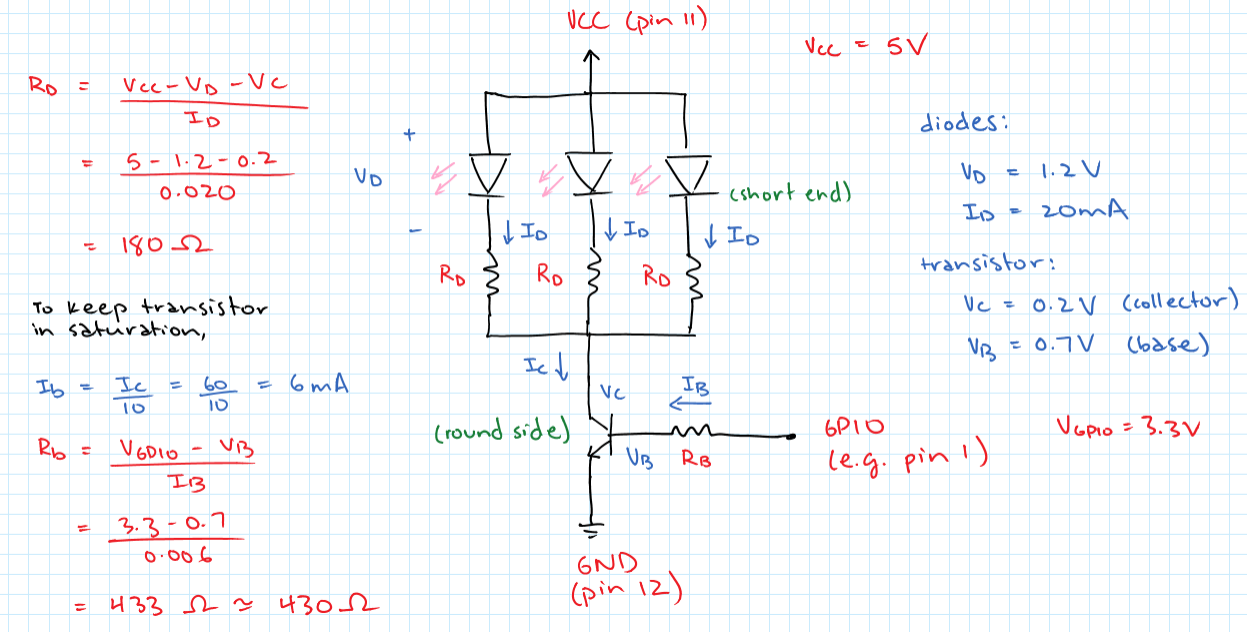

Figure 2 Infrared emitter circuit with calculation details

Receivers (R3208E)

- Each receiver is a phototransistor that turns on as it receives infrared light.
- GPIO input pins pulled up towards 3.3 V when infrared light received, pulled down to ground through 10 kΩ pull-down resistor when no infrared light received.

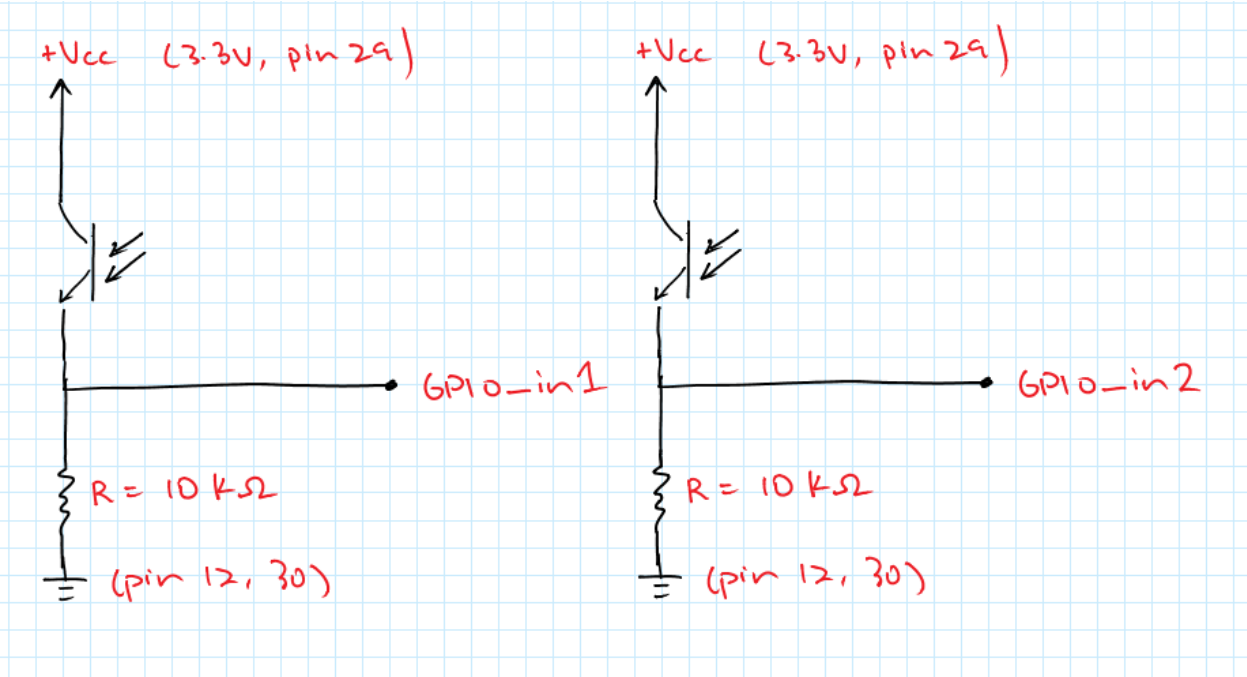

Figure 3 Infrared receiver circuits

Software Setup

Qsys

1. Add the following components.
    a. **Clock Source (clk_sys)**
    b. **NIOS II Processor (cpu)**
    c. **System ID Peripheral (sysid)**
    d. **JTAG UART (jtag_uart_0)**
    e. **SDRAM Controller (sdram)**
    f. **Interval Timer (sys_clk_timer)**
    g. **PIO (pio_led)**
    Width: 7
    Direction: Output
    Enable individual bit setting/clearing
    h. **PIO (pio_key_left)**
    Width: 1
    Direction: Input
    Synchronously capture
    Edge Type: ANY
    Enable bit-clearing for edge capture register
    Generate IRQ
    IRQ Type: EDGE
    i. **PIO (pio_ir_emitter)**
    Width: 1
    Direction: Output
2. Connect all the clocks together
3. Connect all instances of **Avalon Memory Mapped Slave** to the **data_master** under **cpu**.
4. Connect the **Avalon Memory Mapped Slave** of **sdram** to the **instruction_master** under **cpu**.
5. Create a global reset network using **System > Create Global Reset Network**.
6. Assign base addresses using **System > Assign Base Addresses**.
7. Connect all available IRQs under the **IRQ** column.
8. Export **Clock Input**, **Reset Input** as *clk* and *reset*. Export all conduits (**sdram > wire**, **pio_led > external_connection**, etc.) and rename the exports to have the same name as the corresponding component.
    e.g. Export **pio_led > external connection** as *pio_led*.
9. Refresh using **File > Refresh System**. Your system should like Figure 4, below.
10. Under the **Generation** tab, click **Generate.** The system should generate with no warnings or errors.

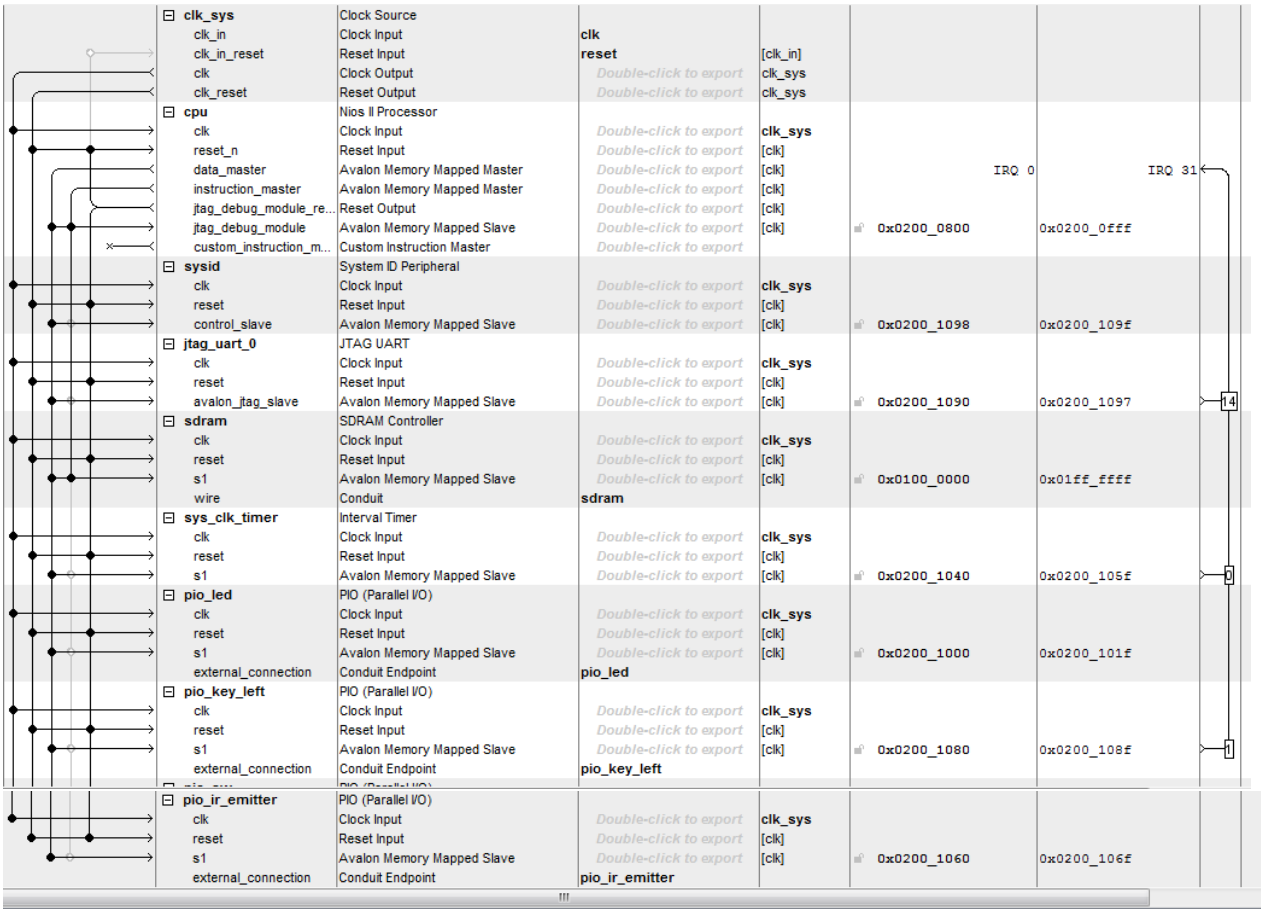

Figure 4 Final Qsys sytem

Toplevel

- Add your Qsys file using **Project > Add/Remove Files in Project**.
- Import the pin assignments for your board using **Assignments > Import Assignments**. Both *DE2.qsf* and *DE0.qsf* are available on eClass.

The full toplevel *de0_nano_system.vhd* is available alongside this tutorial. Since the toplevel is largely similar to the toplevel for any program on the DE0 Nano or DE2 boards, this section covers the important points for infrared control.

In the top level entity:

- *GPIO_0* and *GPIO_1* are of the type *inout std_logic_vector(33 downto 0)*, representing the two expansion headers on the board.
- *KEY* is of the type *in std_logic_vector(1 downto 0)*, representing the first two pushbuttons on the board.

When instantiating the Qsys component:

- *pio_key_left_export* is mapped to *KEY(1)*, since *KEY(0)* may be used for *reset*
- *pio_ir_emitter_export* is mapped to *GPIO_1(0)*

uCOS

The full *main.c* is available alongside this tutorial. This section covers the most important points.

- To control the emitter using the pushbutton, messages are passed between the ISR triggered by the pushbutton and a task that controls the emitter.

```
/* Messages */
// Together these avoid passing 0 to the queue (which is not allowed since
queue messages cannot be null)
// and invert pushbutton input (since the pushbutton returns 0 when it is
pushed down).
#define IR_QUEUE_SEND_BASE     1
        // Use: IR_QUEUE_SEND_BASE + IORD
#define IR_QUEUE_RECEIVE_BASE 2
        // Use: IR_QUEUE_RECEIVE_BASE - OSQPend
```

- The interrupt service routine *isr_on_ir_pushbutton* is triggered whenever the state of the left pushbutton changes and is used to send a message to *ir_task* using a quque. In the ISR, we read the state of the pushbutton, post to the queue, and then mask the edge capture register to end the ISR.

```
static void isr_on_ir_pushbutton(void * context) {
        // Read the state of the pushbutton and post it to the queue.
        //printf("Pressed\n");
        int message = IR_QUEUE_SEND_BASE +
                        IORD_ALTERA_AVALON_PIO_DATA(PIO_KEY_LEFT_BASE);
        OSQPost(queue, (void*)message);
        // Mask to mark the end of the ISR.
        IOWR_ALTERA_AVALON_PIO_EDGE_CAP(PIO_KEY_LEFT_BASE,
                                PIO_KEY_LEFT_BIT_CLEARING_EDGE_REGISTER);
}
```

- The *ir_task* loops and waits for a message on the queue. When a message is received, it prints the status given by the message and writes the status to *PIO_IR_EMITTER_BASE*, which turns the GPIO driving the IR emitter on or off.

```
void ir_task(void* pdata)
{
        INT8U err;
        while (1)
        {
                // Read the value from the queue.
                int status = IR_QUEUE_RECEIVE_BASE -
                                (int)OSQPend(queue, WAIT_FOREVER, &err);
                if (err == OS_NO_ERR) {
                        // Print the result and send it to the emitter.
                        printf("IR: %d\n", status);
                        IOWR_ALTERA_AVALON_PIO_DATA(
                                PIO_IR_EMITTER_BASE, status);
                }
        }
}
```