Blink a LED using the ZynqBerry

Bo Joel Svensson
bo.joel.svensson@gmail.com

Guide version	0.1
Last edited	December 19, 2017
Board compatibility	ZynqBerry
Tested Vivado versions	2016.3

1 Introduction

In this document I try to provide a detailed description of how to get started with IO (in its simplest form) towards the outside world using a ZYNQ. The example presented here is the blinking of a led connected to a GPIO pin on the ZYNQ chip. I am a beginner when it comes to electronics (FPGAs as well) and write this much for my own benefit. Often it is when you try to explain something to someone else that you really learn it yourself. If someone else finds this guide helpful, it is an extra bonus. Please help me improve my understanding of the concepts (and improve this guide) by providing feedback, questions or tips, it is all greatly appreciated.

This guide is not going as in-depth in the details of every step compared to the earlier Getting Started With OpenCL on the Zynq Guide[1]. Since many of the concepts are exactly the same you can refer to the Getting Started With OpenCL guide if more details are needed.

To follow the steps presented here, you need some materials:

- A ZYNQ 7000 based development board. I am using the ZynqBerry from Trenz. If you are not using the ZynqBerry, some extra research will be needed from your end.
- A Breadboard or some other means of connecting up components.
- An LED (Light Emitting Diode).
- A resistor (exactly which resistor will depend on your choice of LED).
- Some wire.

[1] https://www.researchgate.net/publication/302300881_Getting_Started_with_OpenCL_on_the_ZYNQ

The resistor I use is 220Ω and the output voltage on the GPIO pin at logical one is $3.3V$. If we assume a voltage drop across the diode of $\approx 1V$. These parameters give a current of $10.5mA$ through the diode. For more information about LEDs see the following links: `http://www.theledlight.com/led101.html` `https://learn.sparkfun.com/tutorials/light-emitting-diodes-l`

There is also some useful documentation to have at hand:

- ZynqBerry Technical Reference (TE0726 TRM): `https://www.trenz-electronic.de/fileadmin/docs/Trenz_Electronic/TE0726/REV03/Documents/TRM-TE0726-03.pdf`
- AXI GPIO v2.0 LogiCORE IP Product Guide: `https://www.xilinx.com/support/documentation/ip_documentation/axi_gpio/v2_0/pg144-axi-gpio.pdf`

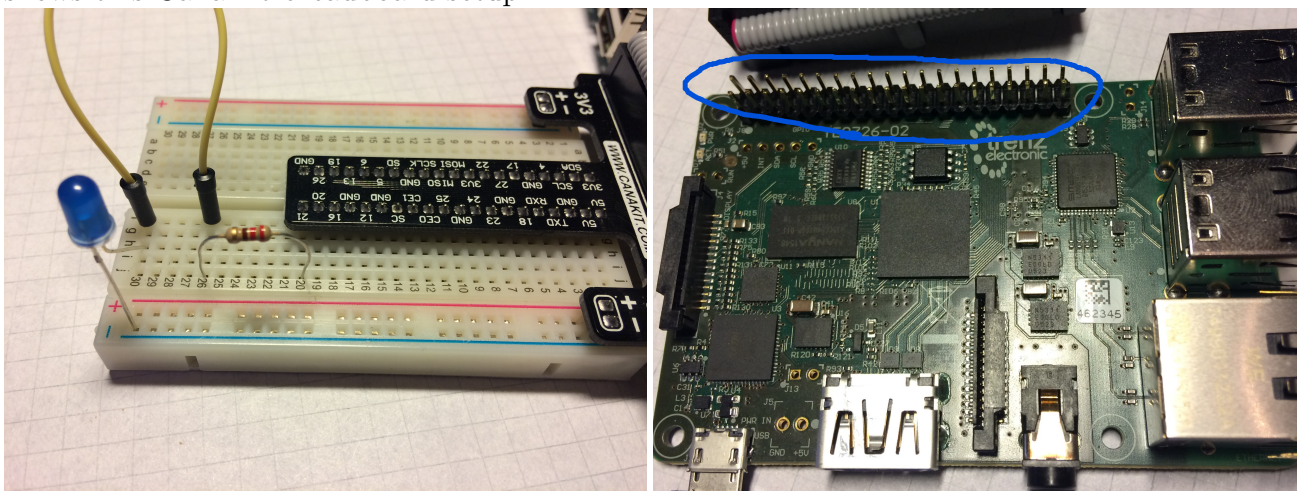
1.1 Initial setup for the zynqberry

The ZynqBerry is a ZYNQ development/experiment/hobbyist board with the same form factor as the Raspberry PI. The ZynqBerry board has gone through a number of revisions with differences in for example the amount of DRAM. I assume that for the procedure outlined in this document all revisions will behave the same.

It helps if you locate the *board files* for your particular board at the Trenz web page (browse for downloads and you can find board files as part of reference designs associated with your board) [2].

1.2 Setting up the Breadboard

This guide assumes the LED is connected to "J8 pin" 40. The J8 interface is the set of connectors on along the "top" of the zynqberry if you have the HDMI port towards you. There are 40 of these pins in total and pin 40 is the top-most right-most of the pins. I use parts of an Raspberry PI CanaKit[3] to make all the pins available on a breadboard. The picture below shows this CanaKit breadboard setup.



If we look in the TE0726 TRM we can find out that pin 40 of the J8 interface is the same thing as something called the Zynq Pin P15. This will be important as this is how we refer to the pin in Vivado later on.

Connect the resistor between pin 40 and the anode (+) of the LED, then the cathode to (-) on the breadboard. With this we are done with the wiring, now let us get to the design on the FPGA side in Vivado.

[2] `http://www.trenz-electronic.de/products/fpga-boards/trenz-electronic/te0726-zynq.html`
[3] `https://www.canakit.com/raspberry-pi/raspberry-pi-3-kits`

2 Creating the Design in Vivado

The work we need to do in Vivado is a three step process:

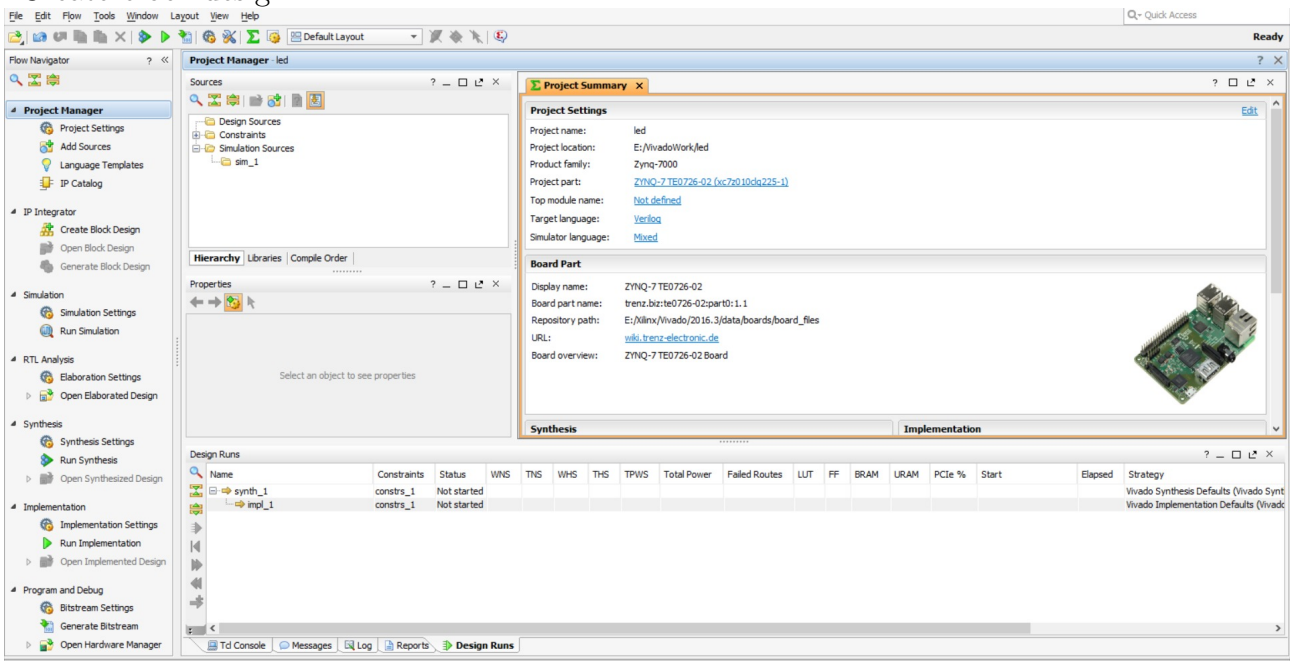
- Create the "Block Design". Instantiate a GPIO block (AXI GPIO).
- Add a constraints file that describes how the GPIO output connects to a package pin on the Zynq.
- And last, Synthesise the design.

2.1 Create a new Project

Creating a project is described very shortly here. If you need more information look at for example the Getting Started With OpenCL Guide[4]. Start up Vivado and perform the steps that are listed below.

- Name it: "led", or whatever you see fit.
- Project type (RTL Project - do not specify sources).
- Select board (If board files are in the right place, select your version of the ZynqBerry).
- Now you will be presented with a Project summary. Just click "finish".

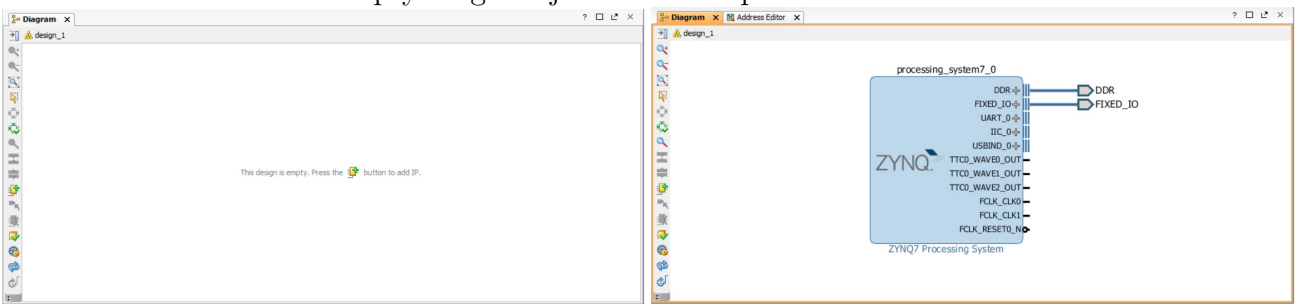
Now we should be at the "Project View" and ready to start with the design. Click on "Create block design".



[4] https://www.researchgate.net/publication/302300881_Getting_Started_with_OpenCL_on_the_ZYNQ

2.2 Create Block Design

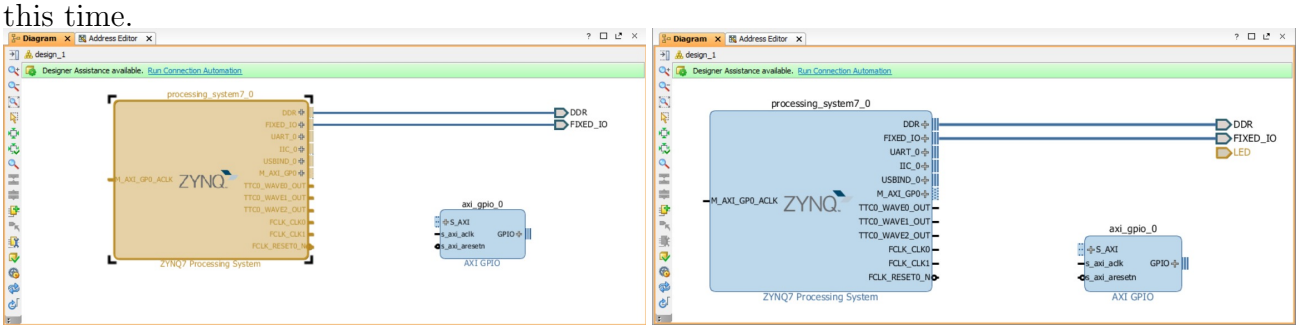
Now we should have an empty diagram just as in the picture on the left below.



Add a Zynq Processing IP to the design and run “Connection automation” with “Apply Board Presets” checkbox ticked.

Now we also need to enable one of the AXI GP master interfaces on the Zynq Processing System. Double click on the Zynq Processing System block to enter into the configuration view and activate the AXI GP0 Interface.

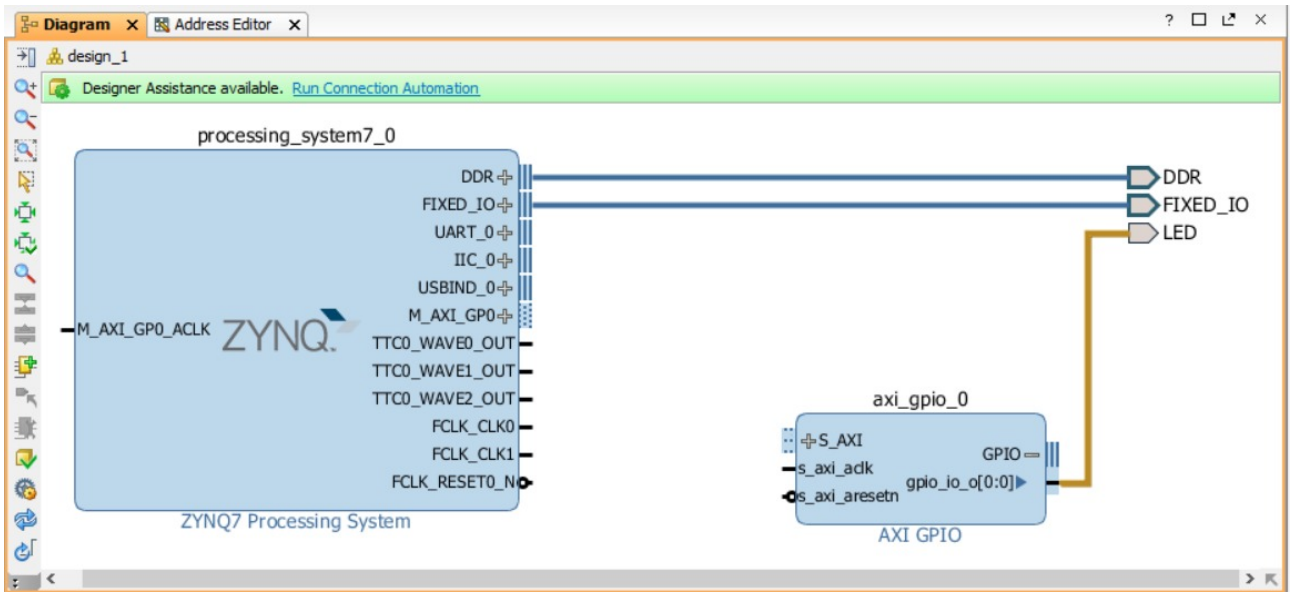
Then it is time to add the AXI GPIO IP Block and a Port that we will later connect to one of the pins to the external world. Click the Add IP button (or right click and select Add IP). Type “GPIO” in the search field and you should find it. Do not run any block automation at this time.



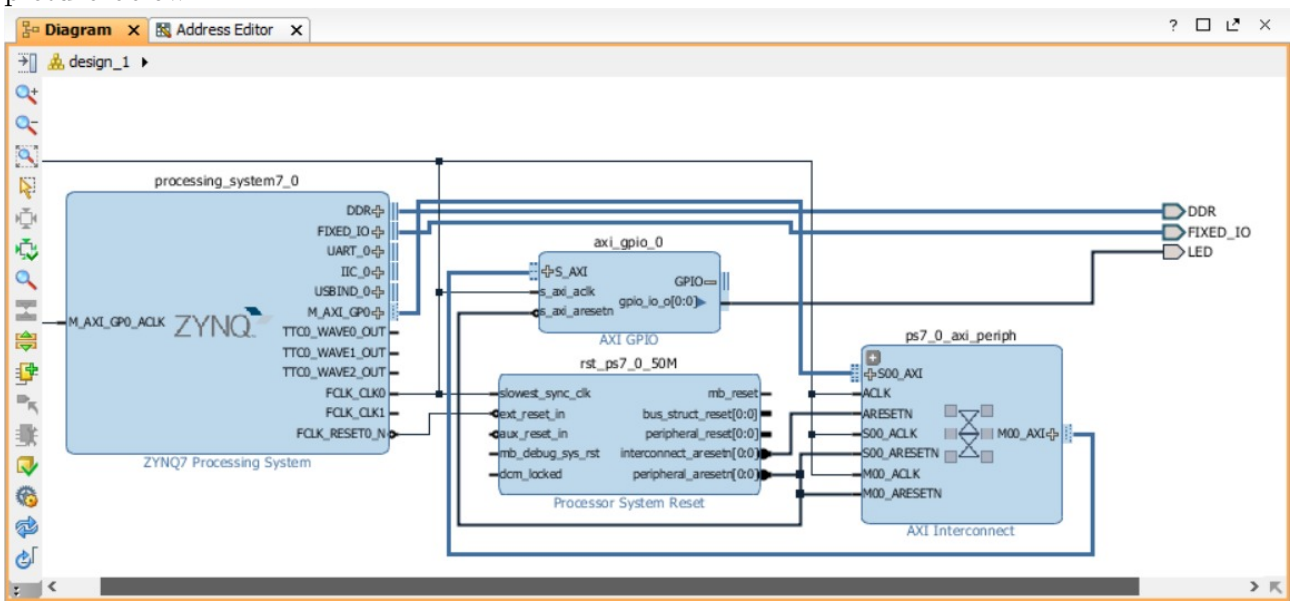
Double click on the AXI GPIO block to enter its configuration view. Here we want to configure it to be “All Outputs” and set the “Width” to 1. This is because we are only interested in one bit of output only data (blink one led).

Right click in the Diagram area (Block design area) and select “Create port”. Name the port “LED” and make it an Output.

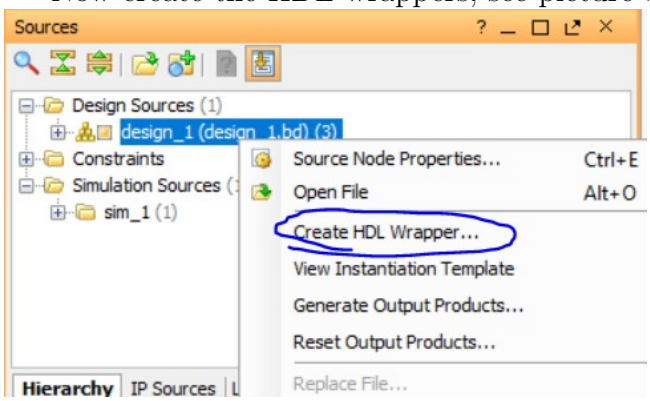
Now connect the GPIO port on the AXI GPIO block to the led. You may need to expand the signals in the GPIO port by clicking on the “+” directly after “GPIO”. Draw a line from gpio_io_o to the LED port.



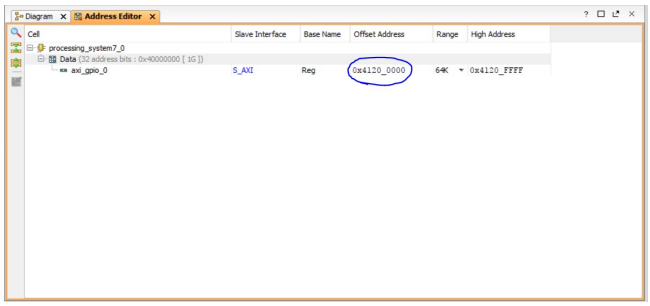
Now run block automation! Everything should then be connected up and look like the picture below:



Now create the HDL-wrappers, see picture below:



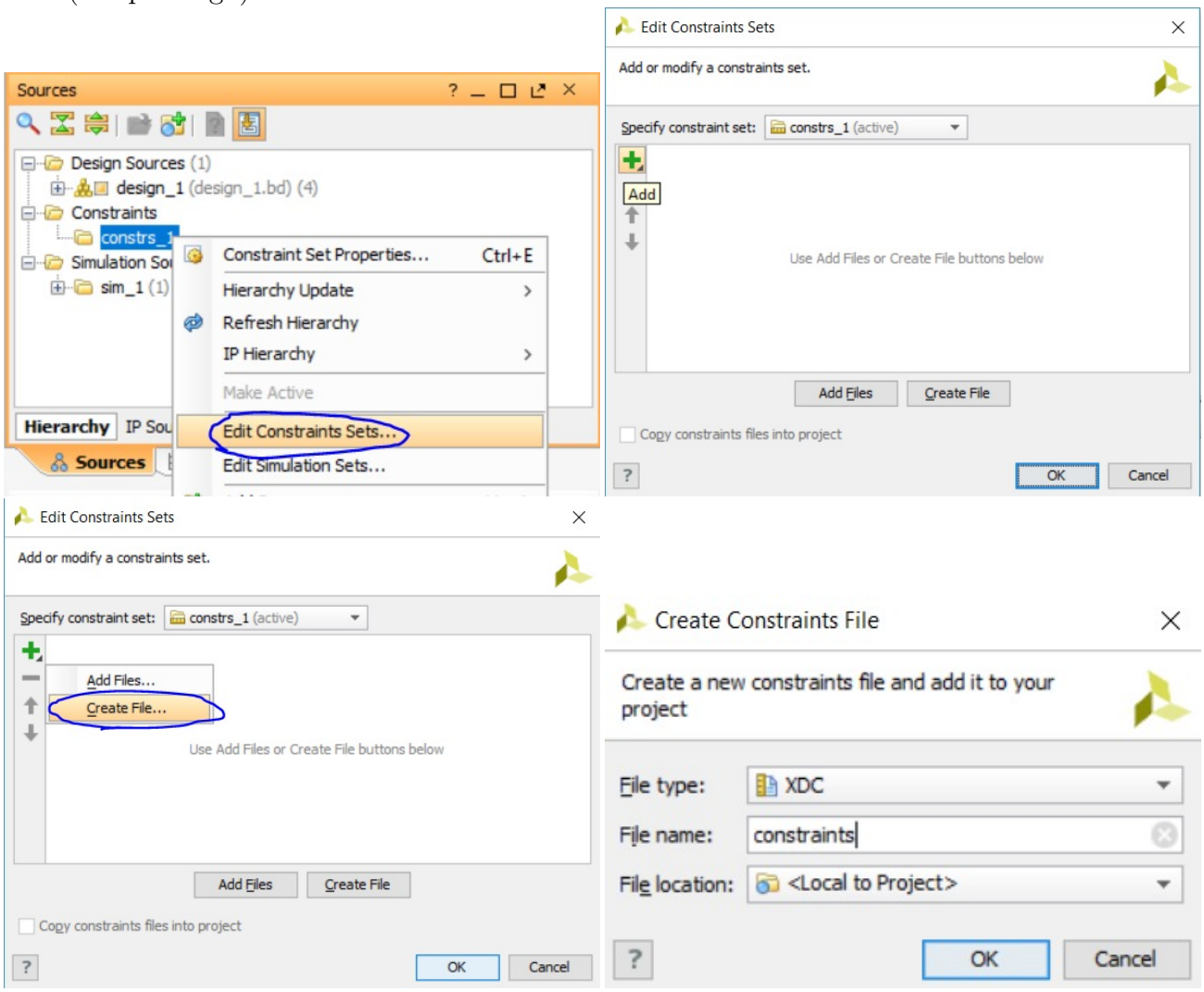
Also take a look at the address editor and memorise the address that the AXI GPIO IP registers are mapped to. we will use this information when writing C code to interact with the LED later on.



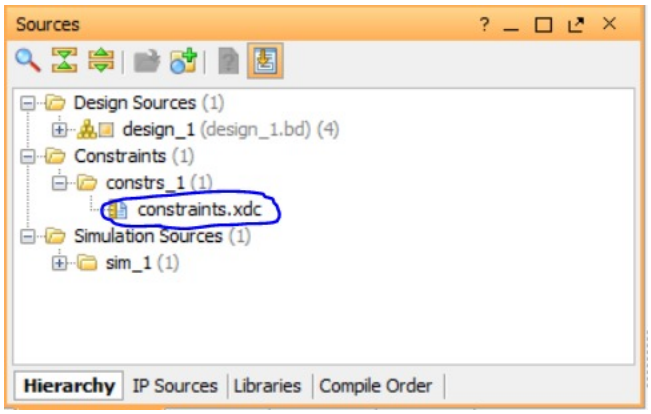
2.3 Create a Constraints File

In order to tell the hardware generation process to connect the LED port that we created to an external pin on the Zynq package, we need to create a constraints file.

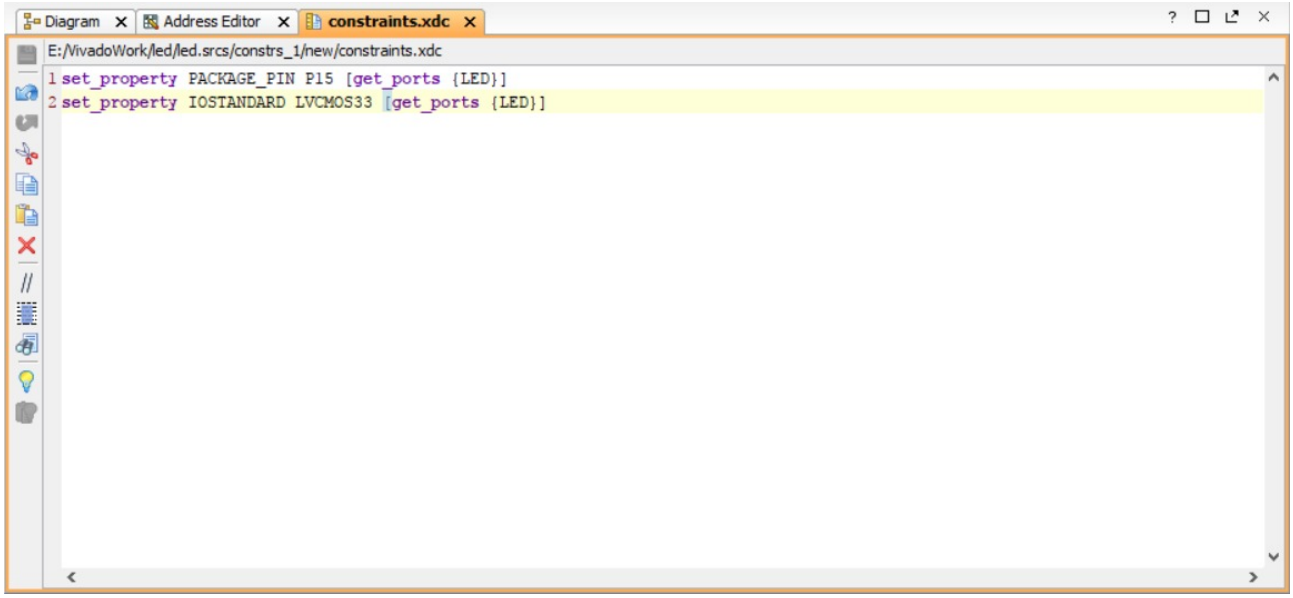
In the Sources window, right clock “constrs_1” and select “Edit Constraints”. Them click Add (the plus sign) and “Create File”.



After creating and naming the constraint file it should now be visible in the sources window and have an “.xdc” file ending.



Edit the constraints file (just double click it) and add the following information:



2.4 Synthesis

Now do in order:

- run synthesis
- run implementation
- run generate bitstream

3 C Code: Fade a LED In and Out

After generating the bitstream, export it (Export Hardware → Include bitstream) and launch the SDK. Create a Xilinx project and start write C Code.

The code I present below cycles the led through different intensities by an ad-hoc attempt to implement Pulse Width Modulation (PWM). Below is the code listing in full and after that its broken apart and explained piece by piece.

```
#include <stdio.h>
#include "platform.h"

volatile unsigned int *led = (volatile unsigned int*)0x41200000;

int main()
{
    init_platform();

    print("Hello World\n\r");
    int duty = 0;
    int direction = 1;
    const int cycle = 1000;
    const int cycles = 50;
    const int one_percent = cycle / 100;


    int d;
    while(1) {
            for (int j = 0; j < cycles; j ++){
            d = duty * one_percent;
                    for (int i = 0; i < cycle; i ++) {
                            if (d > 0) {
                                    *led = 0x1;
                            } else {
                                    *led = 0x0;
                            }
                            d--;
                    }
            }
            duty += direction;
            if (duty >= 100 || duty <= 0) direction = -direction;
    }

    cleanup_platform();
    return 0;
}
```

In the C code we start off with declaring a name for (a pointer to) the memory address that corresponds to the memory mapped register of the AXI GPIO module. Let us call this pointer "led".

```
#include <stdio.h>
#include "platform.h"

volatile unsigned int *led = (volatile unsigned int*)0x41200000;
```

The main function starts out by calling `init_platform`, this function is auto-generated for us and performs some initial configurations. "Hello world" is printed for some feedback over

the serial link. The constants cycle, cycles and one_percent specify the number of iterations of the innermost loop that constitute one cycle, the number of cycles to remain at one level of intensity and how many iterations correspond to one percent of a cycle. the duty variable specify how many percent of a cycle the led should be on. Last direction specify whether we are in the up-in-intensity going phase or downwards.

```
int main()
{
    init_platform();

    print("Hello World\n\r");
    int duty = 0;
    int direction = 1;
    const int cycle = 1000;
    const int cycles = 50;
    const int one_percent = cycle / 100;
```

The d variable specify how many of the iterations of the innermost loop that the led should be on. This value is set every iteration of the outer for loop is modified using the direction value. Whenever the duty value reaches 101 or -1 the direction is changed (from upwards to downwards for example).

```
    int d;
    while(1) {
            for (int j = 0; j < cycles; j ++){
            d = duty * one_percent;
                    for (int i = 0; i < cycle; i ++) {
                            if (d > 0) {
                                    *led = 0x1;
                            } else {
                                    *led = 0x0;
                            }
                            d--;
                    }
            }
            duty += direction;
            if (duty >= 100 || duty <= 0) direction = -direction;
    }

    cleanup_platform();
    return 0;
}
```

4 Conclusion

I hope that following this guide has allowed you to get a led connected to your Zynq development board to flash. I am also a beginner when it comes to hardware, electronics and FPGAs, so please send me feedback on how to improve, add to or clarify the contents of this guide.