SpartanMC

Quick Guide

Table of Contents

List of Figures

List of Tables

Quick Guide

1. Overview

This quick start guide is meant to walk you through the basic steps to configure SpartanMC SoC Kit and to build two sample system-on-chips (SoC), using SpartanMC toolchain.

Note: The SpartanMC toolchain supports synthesis tools for both Xilinx (*ISE*) and Lattice (*Synplify*) FPGAs, but only under Linux. Because of this, it could be a little hard to read and understand this manual for those who do not use Linux, or have neither of the two synthesis tools installed on their machines.

2. Getting Started with SpartanMC

2.1. Requirements

Before you enter the world of SpartanMC, you should first review the following requirements. This may save you some trouble by knowing ahead of time what software you will need.

As mentioned above, the SpartanMC SoC Kit currently only supports Linux, and is known to work on the following distributions:

Distribution	Architecture	Version
CentOS	x86, x64	CentOS 6 or newer
Fedora	x86, x64	Fedora 15 or newer
Ubuntu	x86, x64	Ubuntu 10 or newer
Debian	x86, x64	Debian 6 or newer

Table 1: Supported Linux Distributions

Configuring the SpartanMC toolchain requires that you have several software packages installed, which are listed in the table below:

Package	Version	Notes
GNU Make	3.8 or newer	Makefile/build process
GCC	4.6 or newer	C/C++ Comiler
JDK	1.8 or newer	Java Development Kit

Package	Version	Notes
Apache Ant	1.8 or newer	Java build process
GNU Autoconf	2.5 or newer	Configuration script builder

Table 2: Software for Configuring SpartanMC Toolchain

2.2. Downloading SpartanMC SoC Kit

The SpartanMC SoC Kit is distributed as a set of three pieces. The first piece is the SpartanMC suite that contains all of Verilog source files, libraries and tools needed to construct a SpartanMC system. The other two pieces are the GNU C Compiler and GNU binutils that have been ported to the specific 18-bit SpartanMC architecture.

Each of the three pieces is a TAR archive that is compressed with the gzip program, as shown below.

- `spartanmc.tar.gz`
- `spartanmc-gcc.tar.gz`
- `spartanmc-binutils.tar.gz`

You can obtain the whole SoC Kit from the official website of SpartanMC project at `http://www.spartanmc.de`

2.3. Unpacking the Archives

You should create a directory for the SpartanMC SoC Kit to live (e.g. `~/spartanmc-soc-kit`), move the three downloaded archives to this directory and unpack them there using the following command:

```
find ./ -name '*.tar.gz' -exec tar -xvzf {} \;
```

2.4. Setting Up Your Environment

After successfully unpacking the archives, two new subdirectories should have already been created:

- `bin`
- `spartanmc`

Before configuring the SpartanMC suite, you may need to set some environment variables in `~/.bashrc` as follows.

- `export SPMC_SOC_KIT=/path/to/current/direcory`
- `export SPMC_BIN=$SPMC_SOC_KIT/bin`
- `export PATH=$SPMC_BIN:$PATH`

- `export JAVA_HOME=/path/to/jdk`
- `export SPARTANMC_ROOT=$SPMC_SOC_KIT/spartanmc`

After setting up your environment, you should log out and log in again, or you may use the following command as well:

```
source ~/.bashrc
```

2.5. Configuring the SpartanMC SoC Kit

Now, you need to change directory to where the SpartanMC suite lives:

```
cd $SPARTANMC_ROOT
```

and run the `autogen.sh` script that generates the `configure` script automatically.

```
./autogen.sh
```

The `configure` script accepts many command line options that enable or disable optional features. Before you run the `configure` script, we highly recommend you to take a look at the list of the acceptable options, using the `-h` option:

```
./configure -h
```

You should choose suitable options from the list, depending on which synthesis tool you use. In case of *ISE* , there is one obligatory option: `--with-ise-dir` , which specifies the full pathname of where *ISE* binaries has been installed. Additionally, you need to run the Xilinx settings-script at first before you start configuring the SpartanMC SoC Kit, as shown below:

```
source /path/to/ise/ISE_DS/settings64.sh
./configure --with-ise-dir=/path/to/ise/ISE_DS/ISE/bin/lin64
```

Note: By default, we assume that you have installed *ModelSim* and set up its environment variables already. If this is not the case, you need to disable support for it, using `--disable-modelsim` .

For *Synplify* , there are two obligatory options instead: `--with-diamond-dir` and `--enable-diamond` . The former one has the same usage as `--with-ise-dir` , and the latter one turns support for *Synplify* tools on.This implies that the SpartanMC SoC Kit was intended to be developed for Xilinx FPGAs and therefore uses *ISE* as default synthesis tool. In addition to *ISE* and *Synplify* , we are currently trying to add support for FPGAs from other vendors (e.g. Altera) into the SpartanMC SoC Kit as well.

3. Hello World

This section gives a traditional *hello world* example in which a simple SoC is to be designed using the SpartanMC toolchain. This SoC sends a "hello world" message to your host computer via serial port. For this purpose, it is designed to consist of a SpartanMC core, a UART, a clock generator and a firmware. From this example, you will learn the following:

- How to create a new SpartanMC project.
- How to customize a SoC using *JConfig* .
- How to create a new firmware for the SoC.
- How to build your project and download it into a FPGA device.

Note: The target device used in this example is a *Spartan-601 Evaluation Board* (SP601) from Xilinx, which means *ISE* will be used as the synthesis tool.

3.1. Creating a SpartanMC Project

In order to create a new SpartanMC project, you need to run the following command under `$SPARTANMC_ROOT` :

```
make newproject +path=/path/to/new/project
```

After running the command above, the project directory including a makefile should be created. The next step is to specify the hardware configuration of the SoC. Therefore, the created makefile provides a target to run *JConfig* .

3.2. JConfig

JConfig is a software tool that aims to provide a user-friendly GUI for configuring individual SpartanMC systems. Before starting *JConfig* you should create a new firmware that will be used for the SoC. To do this, first go to the the newly created project directory and execute:

```
cd /path/to/new/project
```

```
make newfirmware +path=firmware
```

You can now start *JConfig* in the project directory by executing the following command:

```
make jconfig
```

As Figure 1 illustrates, the GUI of *JConfig* consists of four major parts:

- *Toolbar* includes *New* , *Open* , *Save* , *Build* and *Schematic* buttons located from left to right.
- *System Manager* shows all hardware components of a SoC in a tree structure.
- *Component Editor* is used to configure each component respectively.

- *Message Window* displays current operational status while configuring a SoC, such as warnings or errors.

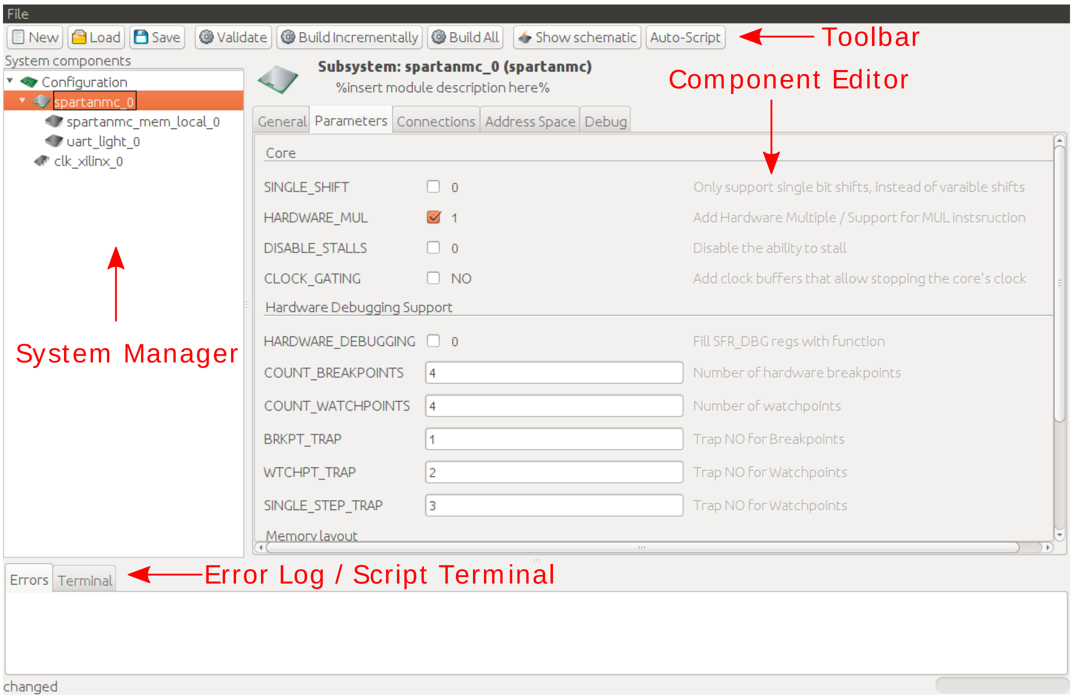

Figure 1: JConfig

3.3. Customizing the SpartanMC System

The first step is to select the target device. You should do this as follows:

- Click the *New* button in the *toolbar* .
- Choose *SP601* as Target in pop-up windows drop down list.
- Click OK

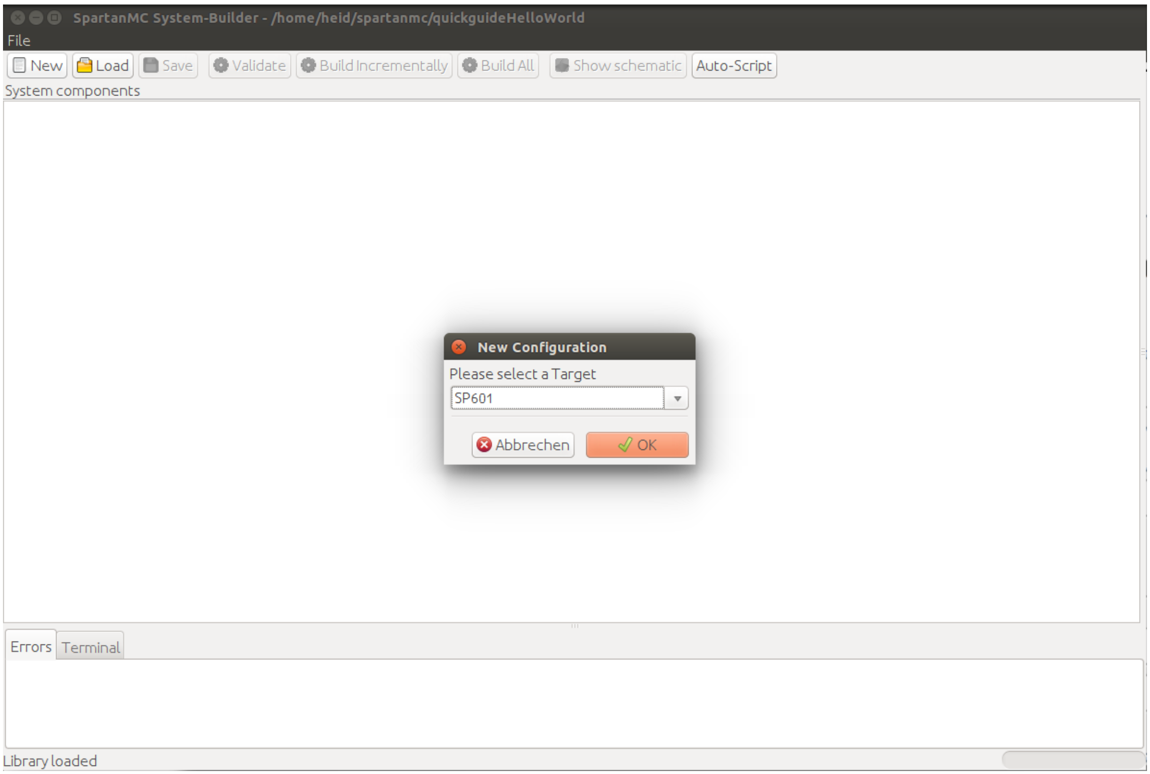

Figure 2: Selecting Target Device

In the next step, you need to select hardware components used in the SoC.

- Right click the `Configuration` node in the *System Manager* .
- Verify that the auto-script button is activated. If the Auto-Script button in the toolbar is activated, for example a local memory is automatically added to each new spartanmc core and many connections will automatically be set. Otherwise this has to be done manually.
- Choose `Subsystem module -> Processor -> SpartanMC core` from the pop-up menu.

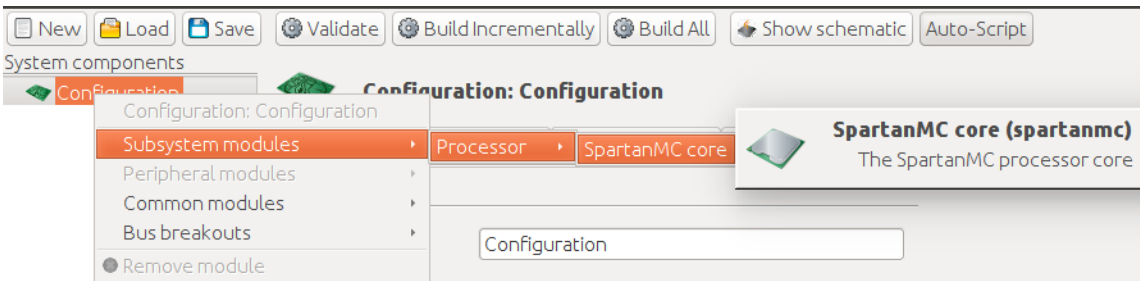

Figure 3: Selecting SpartanMC Core

- Right click `Configuration` again.
- Choose `Common modules -> Clocks -> Xilinx DCM Clock`

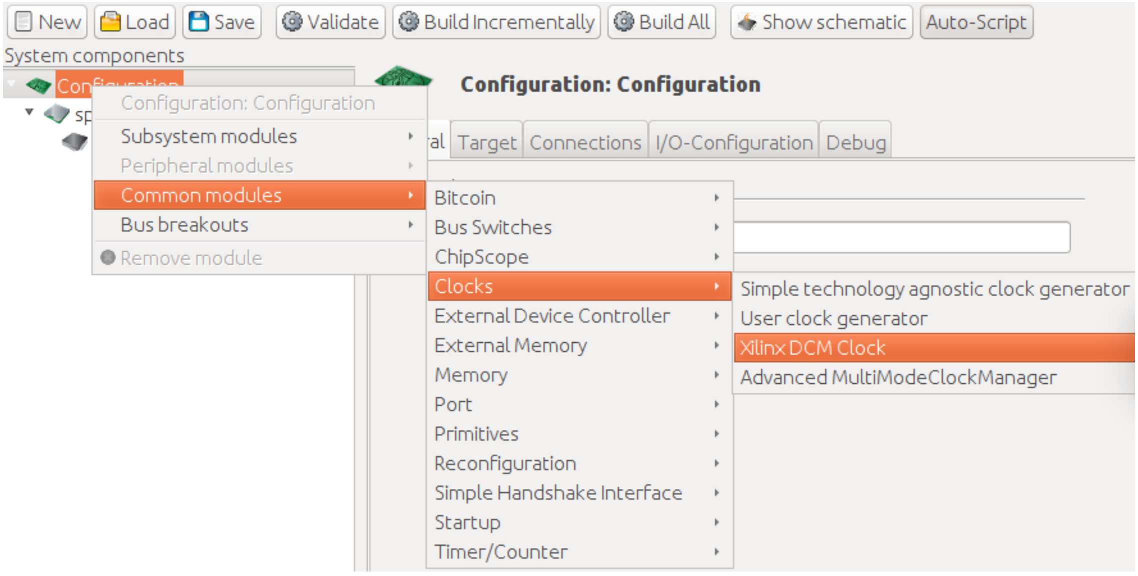

Figure 4: Selecting System Clock Generator

- Right click `spartanmc_0` .
- Choose `Peripheral -> Bus -> UART Light`

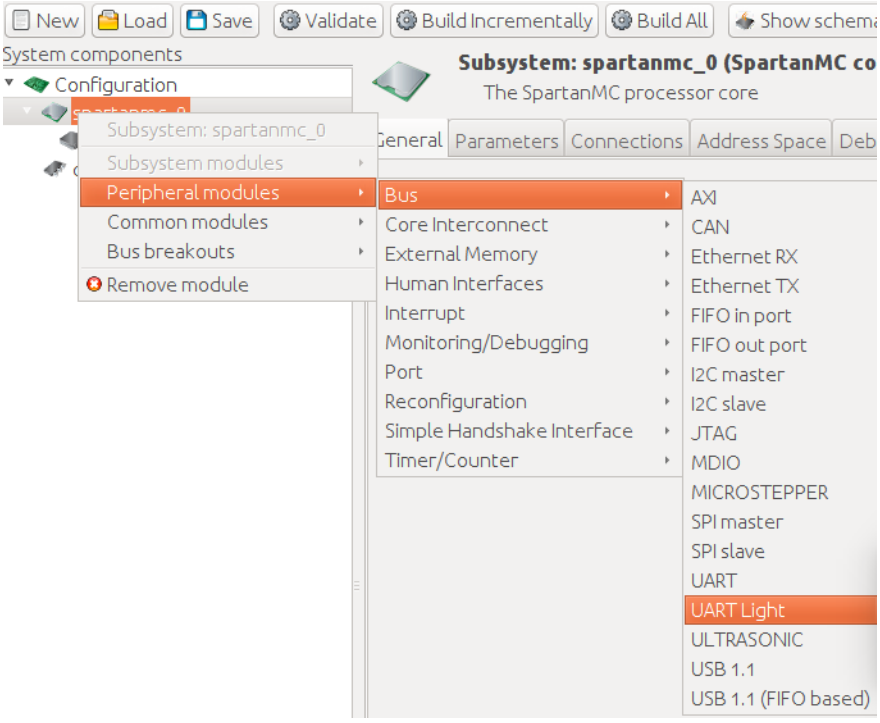

Figure 5: Selecting UART Light

The diagram below shows how the *System Manager* should look so far:

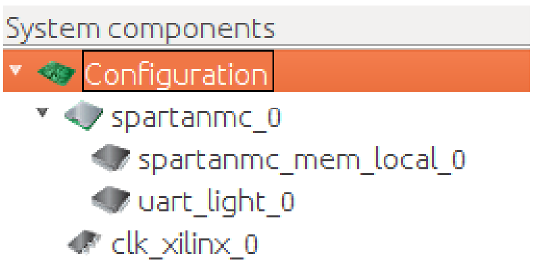

Figure 6: System Manager after Adding Hardware Components

In the third step, you should configure each of the three hardware components, using the *Component Editor* . If you click a hardware component in the *System Manager* , the *Component Editor* will be adapted to the component accordingly.

Note: Only the `Parameters` and `Connections` tab of a hardware component need to be edited in examples from this manual.

First, you need to configure the system clock generator. The current version of the SpantanMC core can run stable at 75 MHz on the SP601 board. However, the SP601 board provides only a 27 MHz oscillator for the purpose of generating a user clock. In order to create a higher clock frequency, the *Digital Clock Manager* (DCM) of Xilinx must be used. The clock generator of SpartanMC is actually a simple wrapper module of the DCM and therefore has the same input and output signals as the DCM. The signal used to drive the SpartanMC core, namely `clkfx` , is generated based on the ratio of two user-defined integers, a multiplier (`CLKFX_MULTIPLY`) and a divisor (`CLKFX_DIVIDE`). Its frequency is derived from the input clock (`clk_gen`) as follows.

$F_{clkfx} = (F_{clk\_gen} * \mathrm{CLKFX\_MULTIPLY})/\mathrm{CLKFX\_DIVIDE}$

The diagrams below illustrate how to configure the system clock generator.

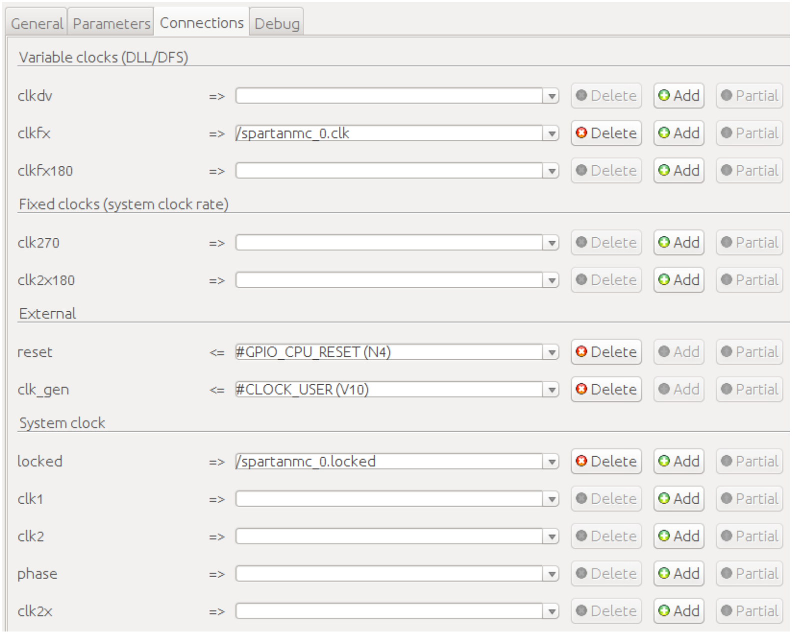

Figure 7: Configuring Connections of System Clock Generator

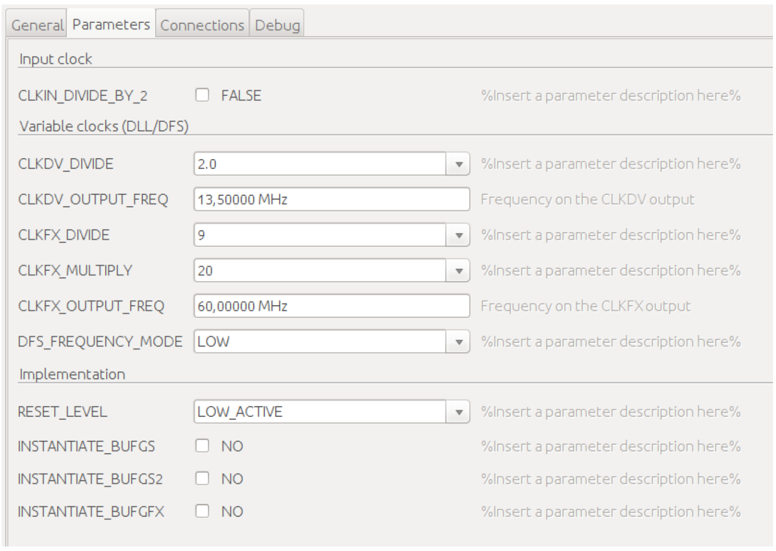

Figure 8: Configuring Parameters of System Clock Generator

To configure the UART, do exactly the same as shown in the following Figures. Most of the parameters and connections are already set. The UART clock frequency parameter is automatically derived from the clock connected to the spartanmc core.

Figure 9: Configuring Connections of UART Light

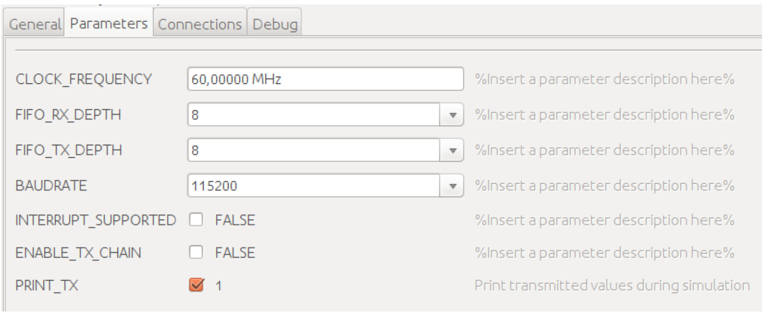

Figure 10: Configuring Parameters of UART Light

The added memory needs no configuration for now and can remain with the default configuration.

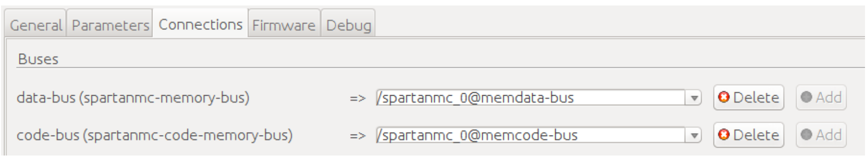

Figure 11: Configuring Connections of Local Memory

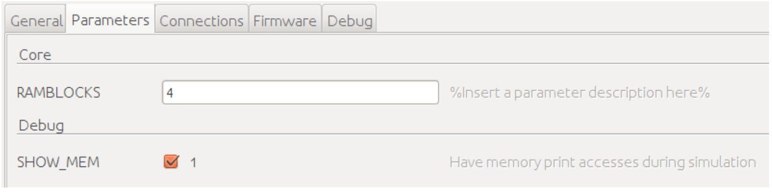

Figure 12: Configuring Parameters of Local Memory

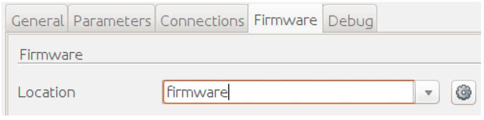

Figure 13: Configuring Firmware of Local Memory

According to the settings of the two components above, the SpartanMC core will be configured automatically. This means that customizing your first SpartanMC system has almost been accomplished. The last thing you should do is define the IO pins.

- Click the `Configuration` node in the *System Manager* .
- Choose the tab `I/O-Configuration` in the *Component Editor* .
- Invert the reset button as shown in the following Figure

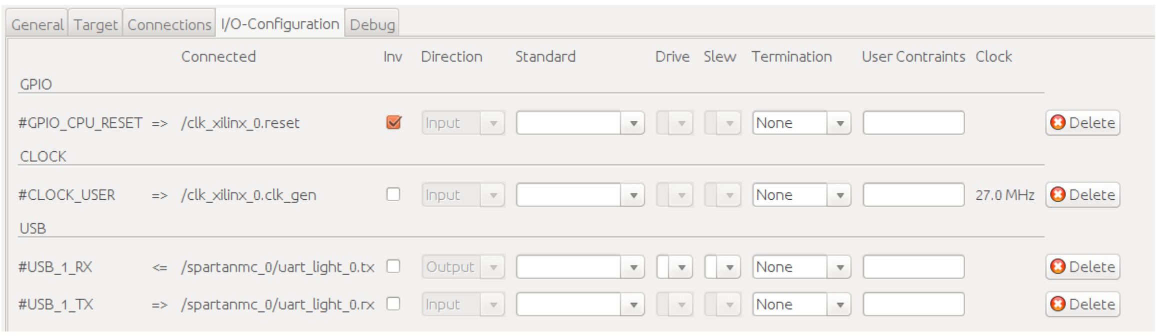

Figure 14: Defining IO Pins

After all these steps above have been completed successfully, you can save the customizations by clicking the *Save* button on the *Toolbar* and build the system by simply clicking the *Build All* button. Also, you can display the top-level design of the system by clicking the *Show Schematic* button, as shown below.

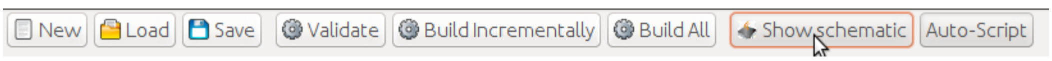

Figure 15: Show Schematic

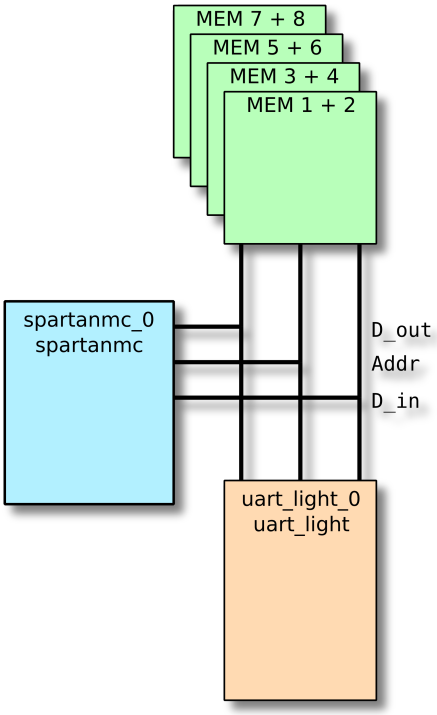

Figure 16: Schematic

Now, you may close *JConfig* or just let it run in the background.

3.4. Creating the Firmware

If not already done, you can create a firmware by running the following command under the project directory (i.e. where you have started *JConfig*).

```
make newfirmware +path=firmware
```

Once the new firmware has been created, the firmware directory should contain:

- `config-build.mk` is used to specify GCC options.
- `include` folder is where all local header files are to be placed.
- `src` folder is where all C source files are to be placed.

Next, you need to create a C file as shown in the following, name it `main.c` and save it under `src` .

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

FILE * stdout = &UART_LIGHT_0_FILE;

void main() {
   printf("hello world\n");
```

```
}
```

`UART_LIGHT_0` is a defined alias for the structure of the type uart_light_regs_t with the name spartanmc_0_uart_light_0. The code for that is located in `/path/to/your/project/system/subsystems/subsystem_0/peripherals.h` . Each peripheral is automatically assigned to such a constant. The name will be the upper case peripheral name used in *JConfig* .

Up to now, your first SpartanMC system has been completely finished.

3.5. Downloading the SoC into FPGA

First, you need to connect the `USB JTAG` port and the `USB UART` port on the *Spartan-601 Evaluation Board* to your computer, using mini-B USB cable. After you power on the board, the `USB UART` port will be recognized as one of *TTY* devices such as `/dev/ttyUSB0` .

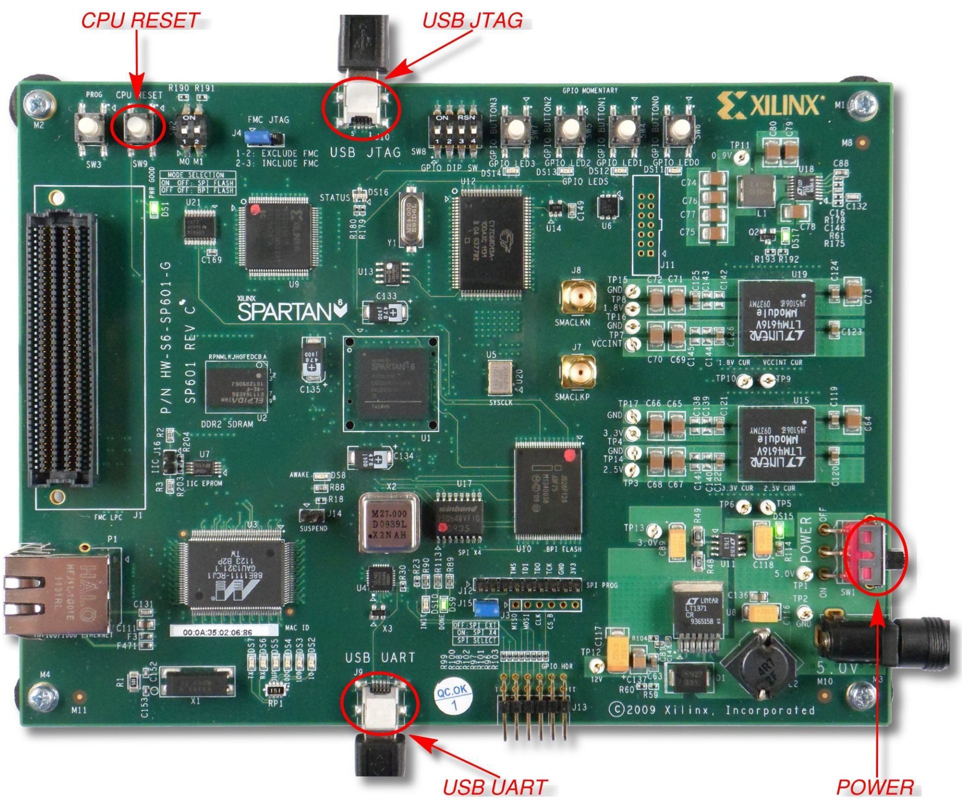

Figure 17: Connections of Spartan-601

Next, open a new console and run the following two commands:

```
stty -F /dev/ttyUSB0 115200 cs8 -echo raw
cat /dev/ttyUSB0
```

Note: If the `USB UART` port is not recognized as `/dev/ttyUSB0` , you need to replace `/dev/ttyUSB0` in the commands above with its actual device name.

Finally, open another console and run the following command under the project directory:

```
make all program
```

After waiting around two minutes for synthesizing and downloading the SoC, you should see "hello world" in the first console. Every time you push the `CPU RESET` button on the board, "hello world" will be printed once again.

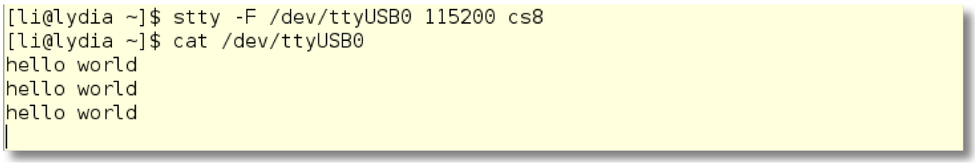

Figure 18: Default Output

Note: If you use *CentOS 6.5* , you have to assert another option of `stty` manually, namely `clocal` , in the following manner:

```
stty -F /dev/ttyUSB0 115200 cs8 -echo raw clocal
```

Sometimes, maybe you just want to change your firmware a little bit and use the same hardware system further, for example, let the SoC built above greet the world in german (i.e. print "Hallo Welt" instead of "hello world"). In this case, you can avoid resynthesizing the whole system and save a lot of time by typing the command shown below, which will replace the old firmware with the new one directly.

```
make updateRam program
```

4. Pseudo Random Number Generator

This section describes a more complex SpartanMC system that generates pseudo random numbers, namely a pseudo random number generator (PRNG). The random seed of the PRNG can be set at runtime via USB, and the generated random numbers are sent to the host computer via USB as well. This example is intended as an exercise for readers who want to dig a little deeper into the SpartanMC SoC Kit, and does not explain every detail. If you are not ready for this yet, just skip this section.

4.1. Customizing the SpartanMC System

Assuming that you have created a new SpartanMC project for this example already, you can now begin customizing the SoC, using *JConfig* . This system is composed of a SpartanMC core, a USB 1.1 controller, an interrupt controller, a clock generator and a firmware. After selecting the hardware components needed, the *System Manager* should look like the following.

Note: Before starting jConfig please remember to create an empty firmware to be used in the configuration.

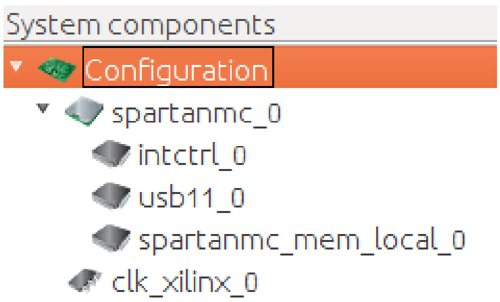

Figure 19: System Manager of PRNG

At first the created firmware has to be registered in the memory module of the spartanmc.

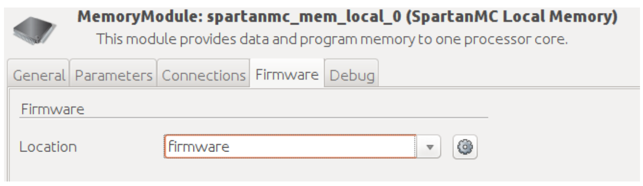

Figure 20: Setting the firmware in the memory

The system clock generator needs to be configured in the almost same way as in the *hello world* example, except that `clkfx` is also employed to drive the USB 1.1 controller.

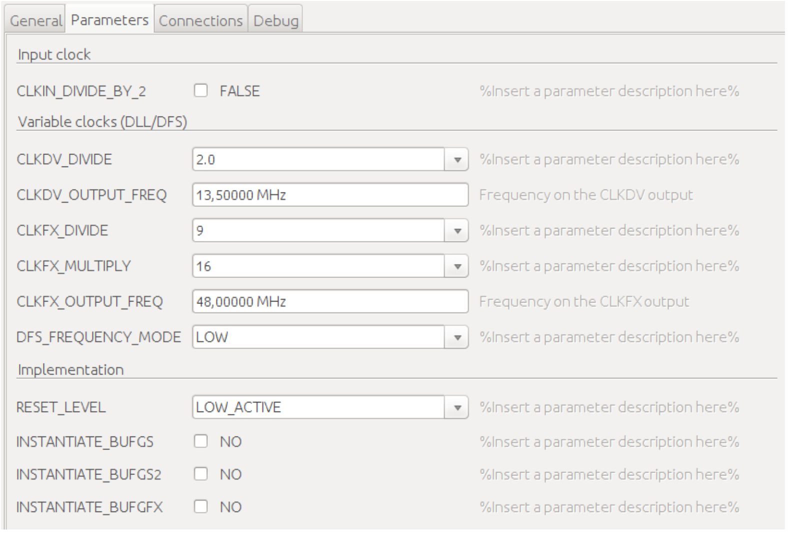

Figure 21: Configuring Parameters of System Clock Generator

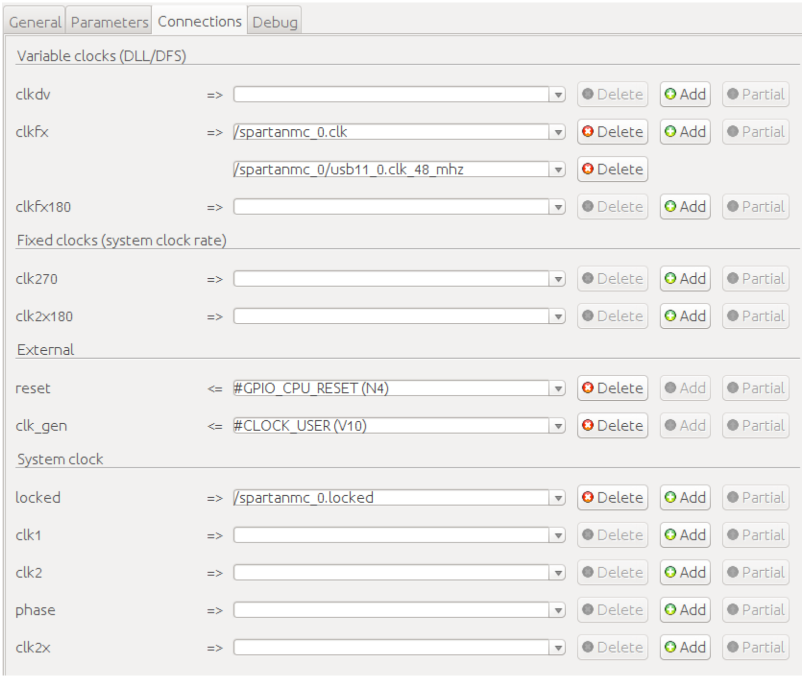

Figure 22: Configuring Connections of System Clock Generator

Note: By clicking the green add button on the right side of `clkfx` , a second textbox can be inserted to connect to the USB controller.

The following figures illustrate how to configure the interrupt controller. For connecting the interrupt sources you need partial connections. Those can be configures by clicking the partial button.

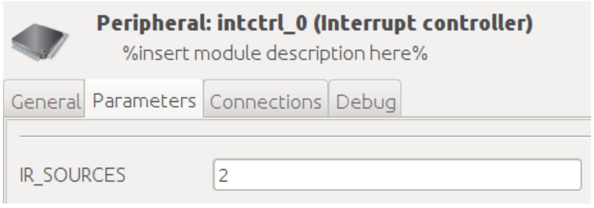

Figure 23: Configuring Parameters of Interrupt Controller

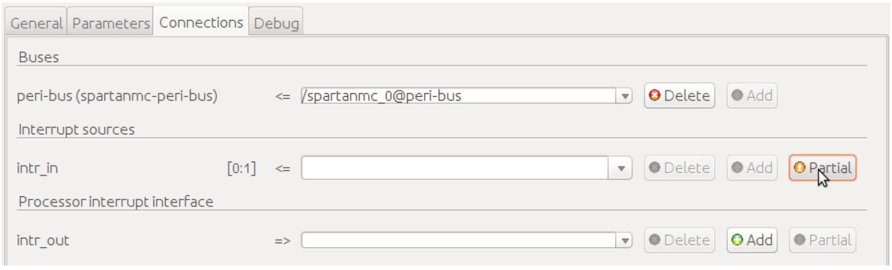

Figure 24: Configuring Connections of Interrupt Controller

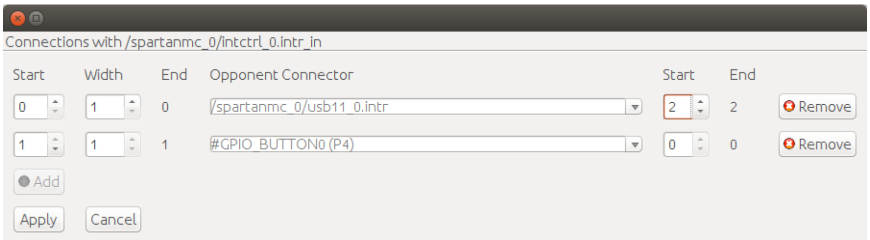

Figure 25: Configuring partial connection of the interrupt sources

The USB 1.1 controller is designed to operate at 48 MHz so that the full bandwidth (12 Mbit/s) can be reached. It supports *Direct Memory Access* (DMA) to reduce processor usage while transmitting data via USB. In contrast to a normal peripheral like UART, it adopts an extra DMA interface to the SpartanMC core. To configure the USB 1.1 controller, you need to edit its `Parameters` and `Connections` tab as follows.

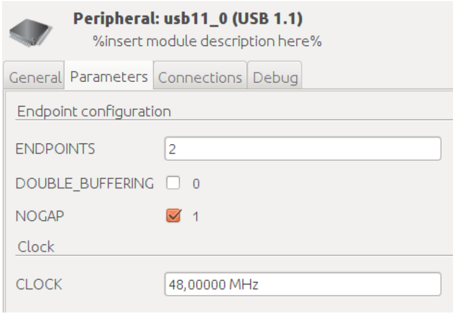

Figure 26: Configuring Parameters of USB 1.1 Controller

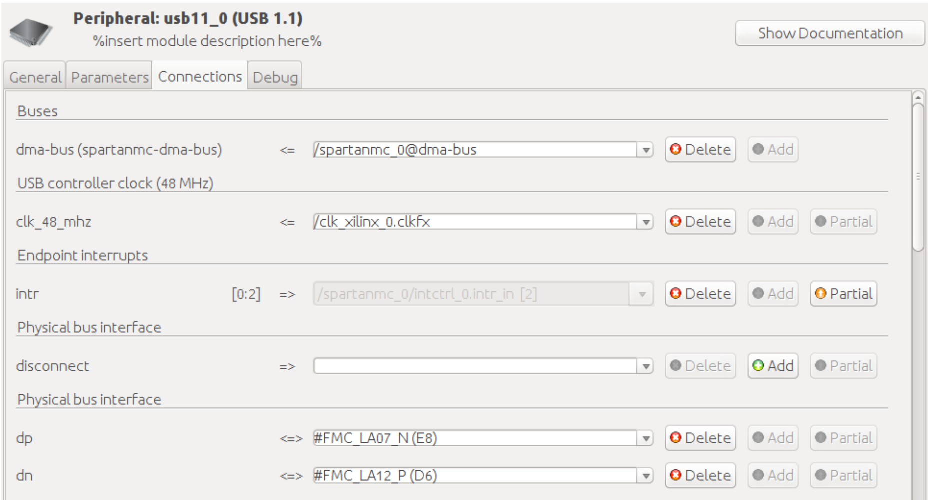

Figure 27: Configuring Connections of USB 1.1 Controller

Note: You do not need to configure any signal of the USB 1.1 controller, which is tagged with `Controller debug interface` or `Controller status` .

The following figure shows how to define the IO pins used by this system.

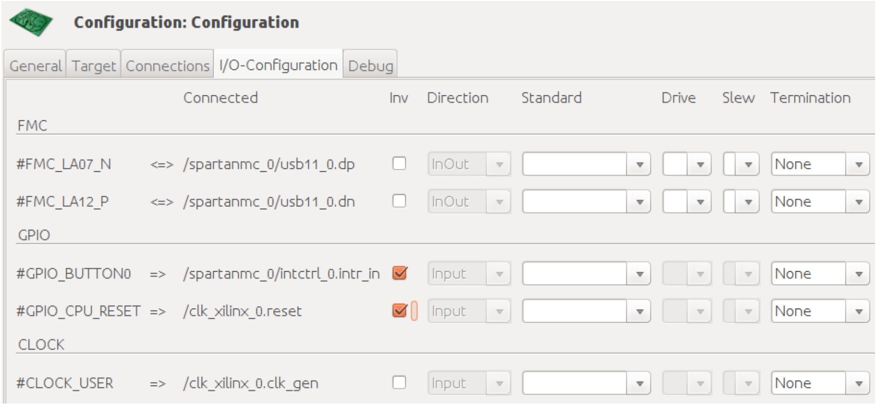

Figure 28: Defining IO Pins

After you have saved and built the hardware part, you may close *JConfig* or just let it run in the background.

4.2. Creating the Firmware

Before compiling the firmware, you should set the Flag `LIB_OBJ_FILES` in `config-build.mk` to the following, since the interrupt library shall be used:

```
LIB_OBJ_FILES:=peri interrupt
```

Now, copy the following C source code into `main.c` .

```
#include #include "peripherals.h"
#include <interrupt.h>
#include "usb_init.h"

#define PACKET_SIZE 32

struct usb_ep tx = USB_ENDPOINT(&USB11_0_DMA,1);
struct usb_ep intr = USB_ENDPOINT(&USB11_0_DMA,2);

unsigned int lfsr;

unsigned int get_random_number() {
   unsigned int i, bit;

   for (i=0; i<16; i++) {
      bit = (lfsr^(lfsr>>2)^(lfsr>>3)^(lfsr>>5))&1;
      lfsr = (lfsr>>1)|(bit<<15);
   }

   return lfsr;
}
```

```
void main() {
   unsigned int i;

   lfsr = 0xACE1;

   usb_init(&USB11_0_DMA, 1);
   usb_ep_intr_en(&intr);
   usb_ep_packet_receive(&intr);
   interrupt_enable();
   while(1) {
      usb_ep_wait_txready(&tx);
      for(i=0; i<PACKET_SIZE; i++) {
         tx.data[i] = get_random_number();
      }
      usb_ep_packet_send(&tx, PACKET_SIZE);
   }
}
void isr00() {
   usb_ep_intr_clear(&intr);
   lfsr = intr.data[0];
   usb_ep_packet_receive(&intr);
}

/* The last unknown 'strong' corresponds always unknown
'strong' */
void isr01() {
   lfsr = 0;
}
```

Furthermore, the descriptors of the USB 1.1 controller are defined in a header file called `usb_init.h` , which needs to be placed in the `firmware/include` folder. This file can be found under `$SPARTANMC_ROOT/examples/prng/firmware/include` .

4.3. Downloading the SoC into the FPGA

Since the *Spartan-601 Evaluation Board* does not have the USB 1.1 physical layer controller (PHY) integrated, you need a custom PHY as shown in the schematic. One such PHY is integrated in our custom board.

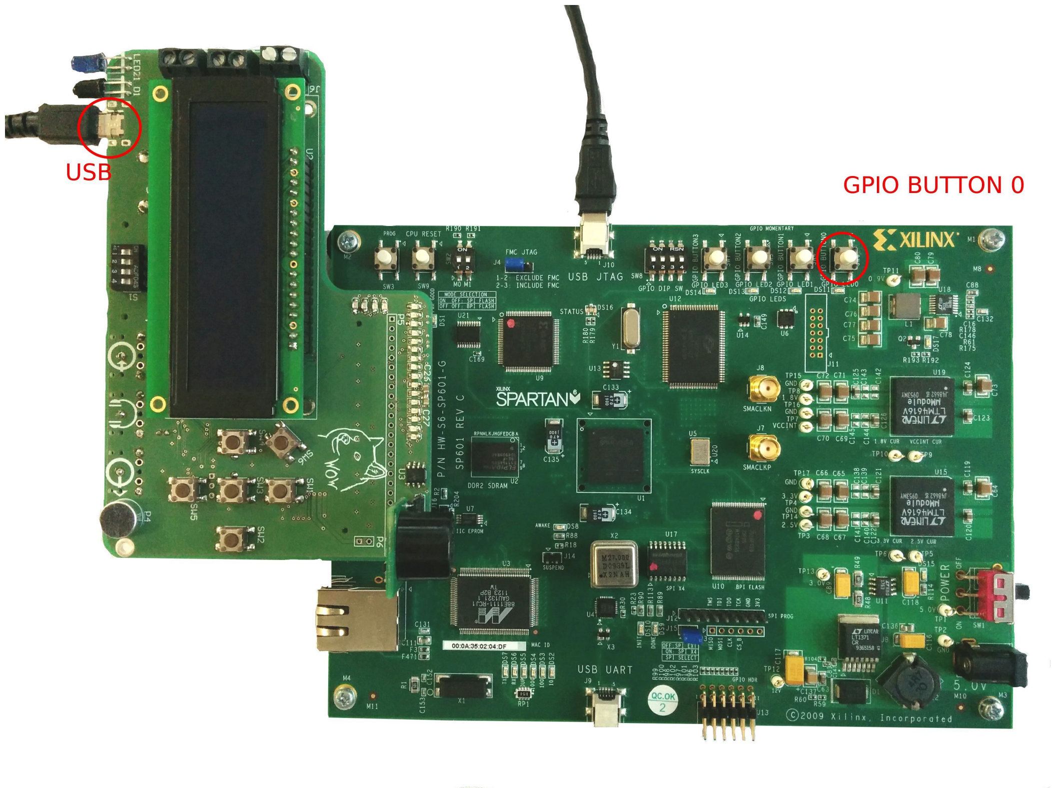

Figure 29: Connections of Spartan-601

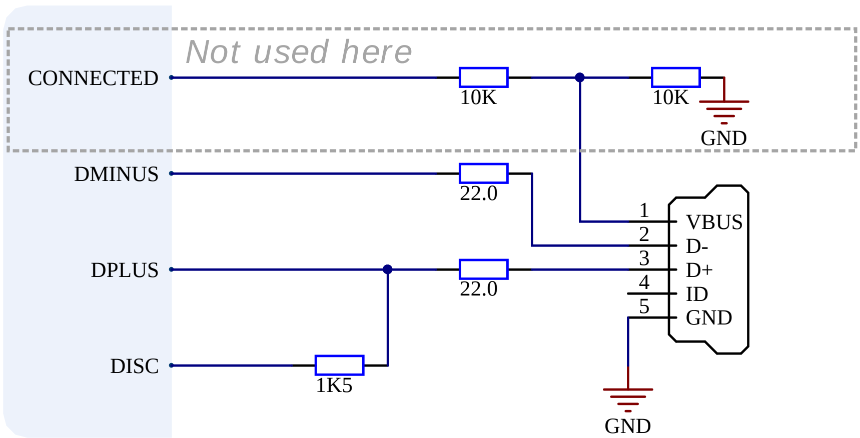

Figure 30: USB PHY

In addition, a computer-side software has been developed, which is intended to test the USB 1.1 controller and USB 1.1 PHY. This software hides low level details of how to communicate with a USB port. Due to this, after the SoC has been synthesized and downloaded into the FPGA, the only thing you need to do is type the following commands in one console.

```
make -C $SPARTANMC_ROOT/examples/prng/firmware/util
```

```
export PATH=$SPARTANMC_ROOT/examples/prng/firmware/util:$PATH
usbcat -v 0x6666 -p 0xaffe | hexdump -Cv
```

Consequently, an infinite sequence of random numbers will be printed in the console.

```
[li@lydia usbcat]$ usbcat -v 0x6666 -p 0xaffe | hexdump -Cv
00000000  12 9e 2c 66 43 d8 05 83  28 4f 6a d7 26 de 1a 7c  |..,fC...(Oj.&..||
00000010  13 27 09 33 0a 00 29 48  13 01 61 71 36 b0 10 6f  |.'.3..)H..aq6..o|
00000020  16 0f 75 85 65 20 77 bd  07 d8 5a c3 01 cd 75 31  |..u.e w...Z...u1|
00000030  2d aa 22 80 1f 7c 77 df  23 ad 16 e6 61 4c 26 84  |-."..|w.#...aL&.|
00000040  0f 66 2d a8 6a ea 36 ea  54 42 29 18 63 4d 10 25  |.f-.j.6.TB).cM.%|
```

Figure 31: Default Output

If you push `BUTTON0` , the interrupt routine `isr01` will be invoked, which sets the random seed to `0` . As a result, the output will change to an infinite sequence of `0` .

```
00026880  56 01 7f e1 64 1a 1e 5f  0e 49 4c 30 68 fb 58 35  |V...d.._.IL0h.X5|
00026890  0b d7 10 0e 46 64 75 92  25 30 55 ab 5c 69 66 d5  |....Fdu.%0U.\if.|
000268a0  76 d7 1e 76 22 06 00 00  00 00 00 00 00 00 00 00  |v..v"...........|
000268b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000268c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000268d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Figure 32: Output after Pushing BUTTON0

If you want to set a new random seed, you need to abort the currently running command (i.e. press `Ctrl-C`), and to type the following command that invokes the interrupt service routine `isr00` , which resets the random seed to `0x3031` .

```
echo "01" | usbcat -v 0x6666 -p 0xaffe | hexdump -Cv
```

Note: In this example, the ASCII codes of two arbitrary characters are used as 16-bit random seed.