TE0720 User Manual

Authors: Antti Lukats, Thorsten Trenz, Sven-Ole Voigt
Revision: 0.2
Date: 26-Sep-2014 10:35

Table of Contents

Copyright © 2014 Trenz Electronic GmbH

Copyright © 2014 Trenz Electronic GmbH

Overview: TE0720 GigaZee Zynq SoM

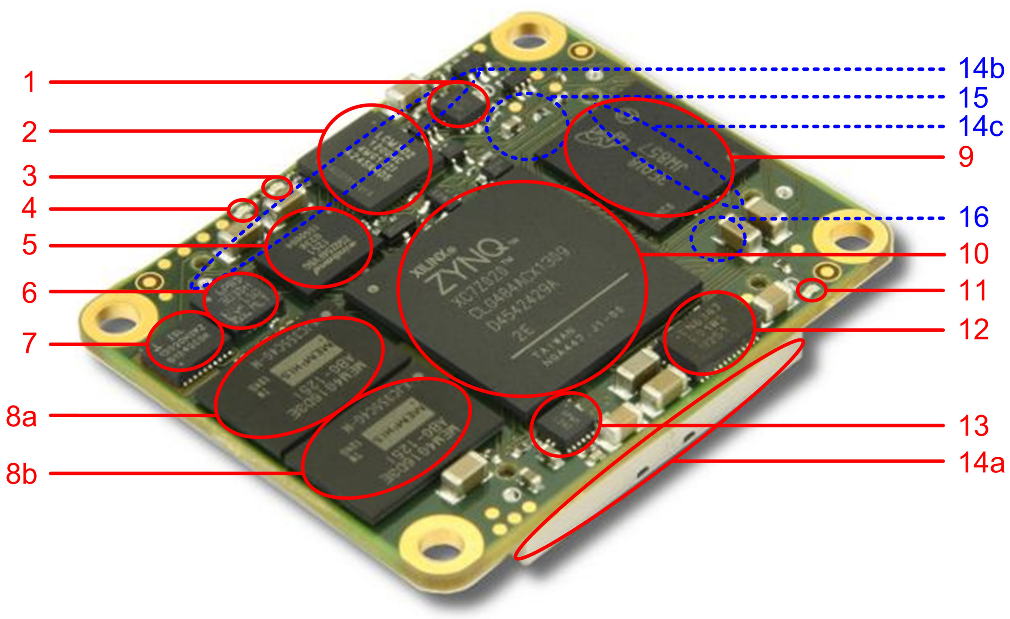


Figure 1: TE0720 GigaZee Zynq SoM (REV 01).

Features

1. 1.5A, 4MHz PowerSoC DC-DC Step-Down Converter with Integrated Inductor (Enpirion EP53F8QI) for 1.8V Power Supply
2. System Controller CPLD (Lattice LCMXO2-1200HC): 1,200 Macrocell CPLD with Block RAM, Flash and PLL
3. User LED 1 (Green)
4. User LED 2 (Red)
5. 32 Mbyte Quad SPI Flash memory (Winbond W25Q256FV)
6. MEMS sensor (ST Microelectronics LSM303DLMTR): 3-axis accelerometer and 3-axis magnetometer
7. Real Time Clock with Embedded Crystal (Intersil ISL12020M): ±5ppm Accuracy
8. 1Gbyte = 2x 256Mbitx16 (32-bit wide) DDR3 Memory (Memphis MEM4G16D3EABG-125I)
9. e·MMC NAND Flash (Micron MTFC4GMVEA-4M, may also be different manufacturer due to availability) usually 4 GByte, depends on assembly option
10. Xilinx Zynq-7000 System-on-Module (Xilinx XC7Z020) - Processing System: Dual ARM Cortex-A9, unified 512Kbyte L2 Cache, 256Kbyte on-chip Memory, 54 Multiplexed I/O Pins (MIOs); Programmable Logic: Artix-7 FPGA, 85K Logic Cells, 560 Kbyte extensible Block RAM (140x 36 Kbit BRAM Blocks), 220 programmable DSP Slices, Dual 12bit 1Msps Analog-to-Digital Converter, 200 I/O Pins (SelectIO Interfaces)
11. User LED 3 / FPGA Done (Green)

Copyright © 2014 Trenz Electronic GmbH

12. 4A High-Efficiency Power SoC DC-DC Step-Down Converter with Integrated Inductor (Enpirion EN6347) for 1.0V Power Supply
13. 1.5A, 4MHz PowerSoC DC-DC Step-Down Converter with Integrated Inductor (Enpirion EP53F8QI) for 1.5V Power Supply
14. Trenz 4x5 Module Socket Connector (3x Samtec LSHM Series Connectors)
15. Gigabit Ethernet Transceiver PHY (Marvell 88E1512)
16. Highly Integrated Full Featured Hi-Speed USB 2.0 ULPI Transceiver (Microchip USB3320C-EZK)

Document Change History

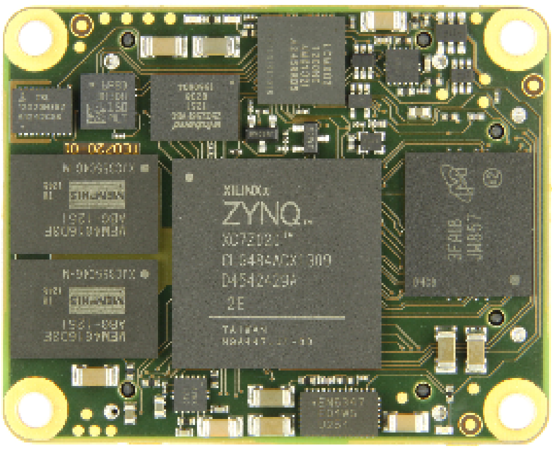
date	revision	authors	description
2014-02-10	0.2	Sven-Ole Voigt	Work in progress
2013-04-17	0.1	Antti Lukats, Thorsten Trenz	Initial release
	All	Antti Lukats, Thorsten Trenz, Sven-Ole Voigt	

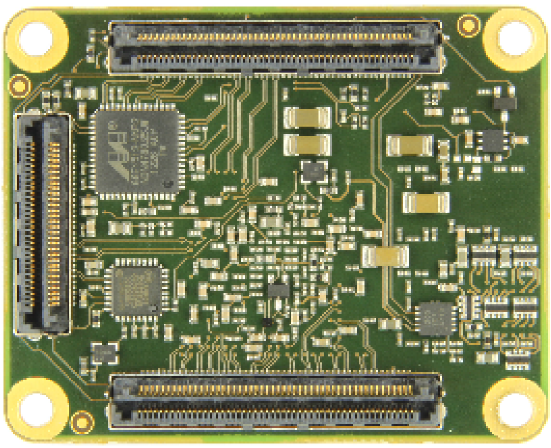
Overview

Trenz Electronic GigaZee XC7Z series are industrial-grade SoMs (system on modules) integrating a leading-edge Xilinx Zynq-7000 SoC, gigabit Ethernet transceiver, 32-bit-wide 1 gigabyte DDR3 SDRAM, 32 megabyte SPI Flash memory for configuration and operation, eMMC, and powerful switch-mode power supplies for all on-board voltages. A large number of configurable I/Os is provided via robust board-to-board (B2B) connectors. All this on a tiny footprint smaller than half a credit card, at the most competitive price. Development boards are available in our online shop; reference designs are available in our open design repositories.

Sample Applications

- Cryptographic hardware module
- Digital signal processing
- Embedded industrial OEM platform
- Embedded system design
- Emulation platforms
- FPGA graphics
- FPGA video processing
- Image processing
- IP (intellectual property) cores
- Parallel processing
- Rapid prototyping
- Reconfigurable computing
- System-on-Chip (SoC) development





Key Features

- Industrial-grade Xilinx Zynq-7000 SoM (system on module)
- Rugged for shock and high vibration

- ARM dual-core Cortex-A9 MPCore @ up to 800 MHz
- 10/100/1000 tri-speed gigabit Ethernet transceiver (PHY), SGMII accessible on a board-to-board connector
- USB 2.0 high-speed ULPI transceiver
- 32-bit-wide 1 Gbyte DDR3 SDRAM
- 32 Mbyte SPI Flash memory supporting XiP
- 4 Gbyte e·MMC (up to 64 Gbyte)
- Plug-on module with 2 × 100-pin and 1 × 60-pin high-speed hermaphroditic strips
- 152 FPGA I/Os (75 LVDS pairs possible) and 14 MIOs available on board-to-board connectors
- 4.0 A x 1.0 V power rail
- 1.5 A x 1.5 V power rail
- 1.5 A x 1.8 V power rail
- System management and power sequencing
- eFUSE bit-stream encryption
- AES bit-stream encryption
- Valid MAC Address and 2 kilobit serial EEPROM
- SHA-256 authentication chip with unique serial number
- temperature compensated RTC (real-time clock)
- MEMS sensor (3-axis accelerometer and 3-axis magnetometer)
- 3 user LEDs
- Evenly-spread supply pins for good signal integrity
- Other assembly options for cost or performance optimization available upon request

Copyright © 2014 Trenz Electronic GmbH

Getting Started

Preloaded (Factory default) SPI Flash Image

The TE0720 module comes with the SPI Flash preloaded with a default bootloader, U-Boot and Linux are setup to run automatically if SPI flash boot mode is selected (Red LED fast blinking after power up). U-Boot is configured with a standard 3 second delay to allow the U-Boot interactive console to be used. The console is connected to PS UART0, mapped to MIO14 (UART0_RX) and MIO15 (UART0_TX), and has a baud rate of 115200. The MIO14 and MIO15 signals are routed to the CPLD on the TE0701 Carrier Board which, in turn, connects these PS UART0 RX and TX signals to the onboard FTDI USB to Multipurpose UART/FIFO IC (FTDI FT2232H). Hence, the console output view can be easily accomplished (e.g., in SDK "Terminal" window or in HTerm) by attaching it to the corresponding virtual COM port (the COM port number may differ):

> Please consult TE0701 Carrier Board User Manual | Connecting FTDI USB-to-UART/FIFO Interface for more details on the required VCOM configuration!

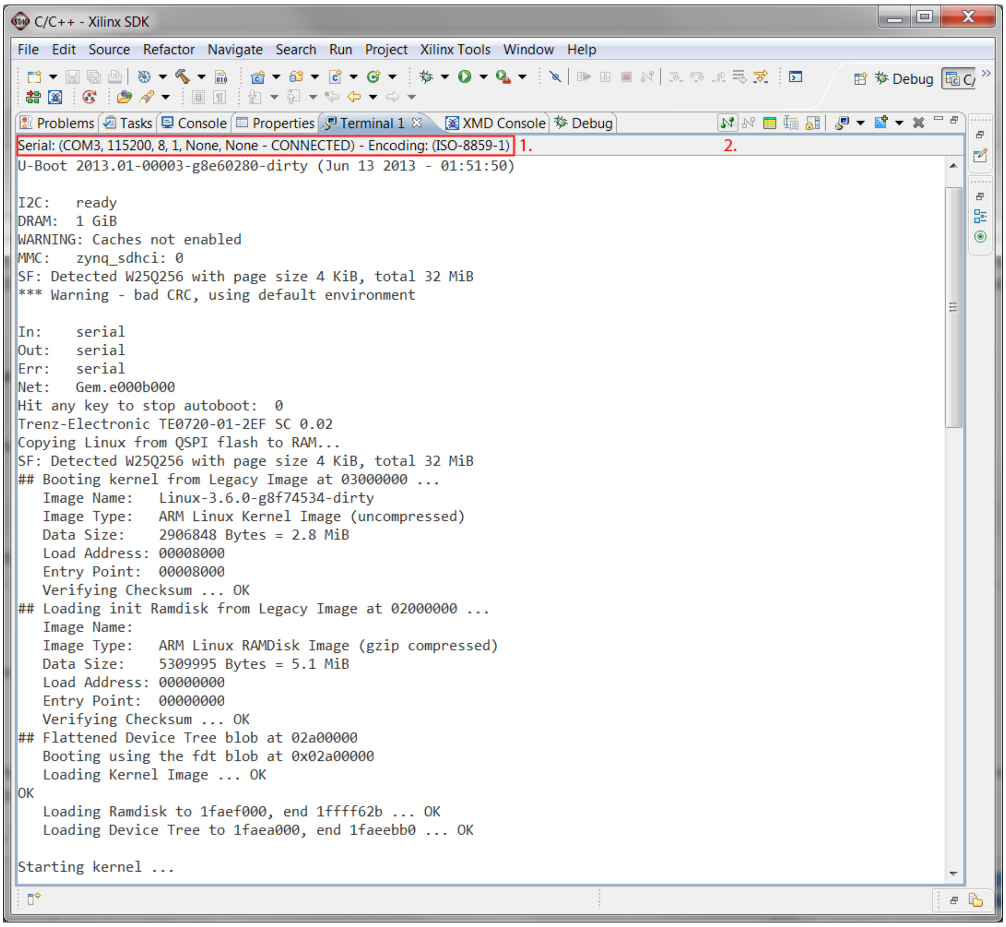


Figure 1: Booting Linux Kernel (by default from SPI Flash) - SDK "Terminal" window

Copyright © 2014 Trenz Electronic GmbH

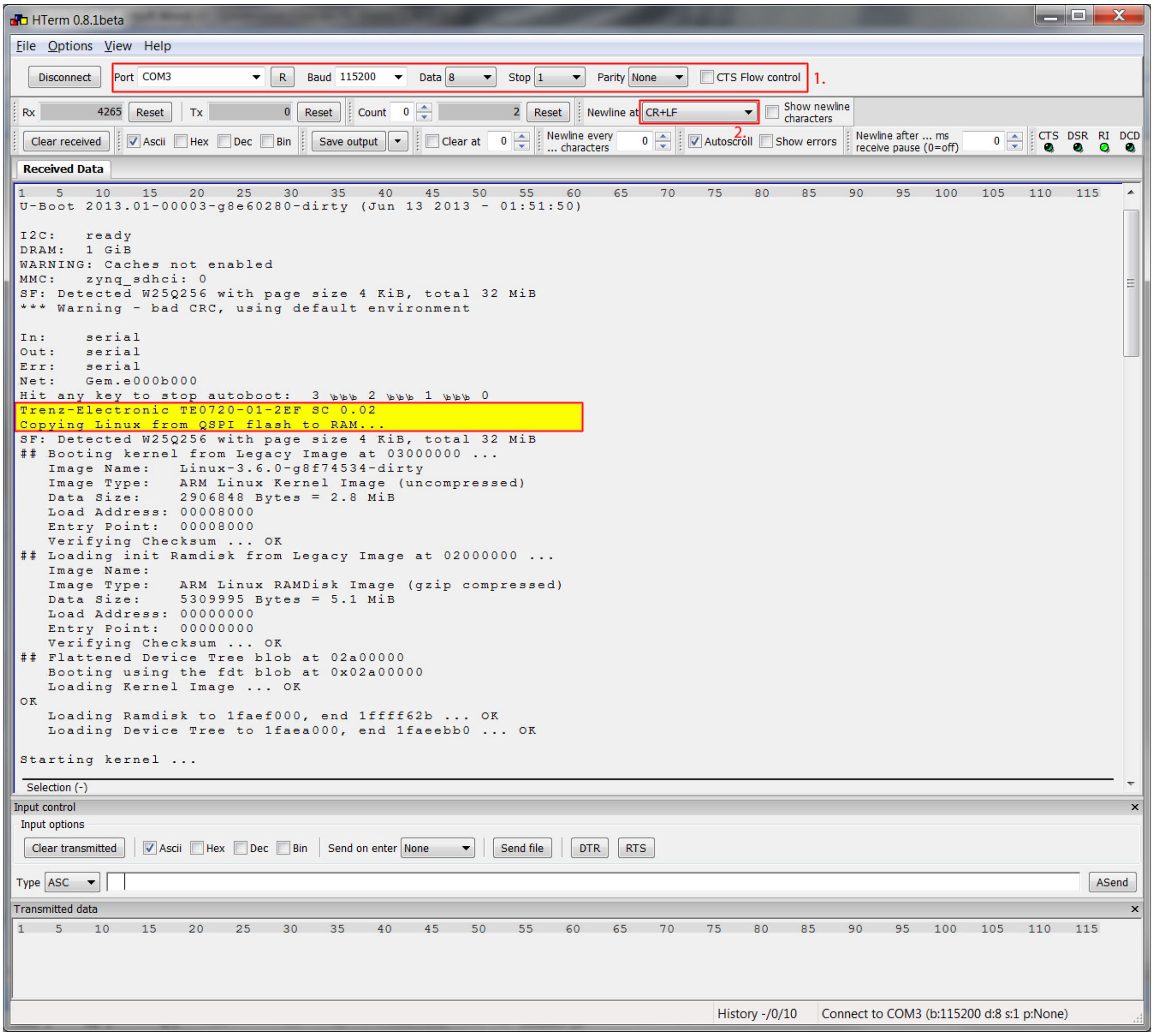
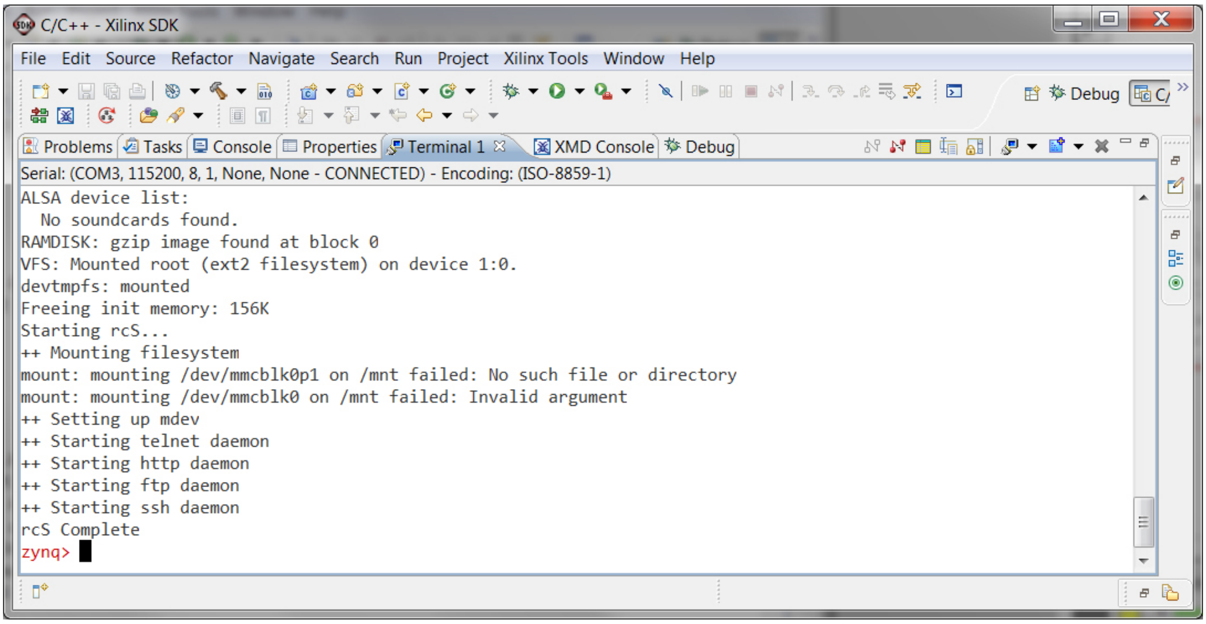


Figure 2: Booting Linux Kernel (by default from SPI Flash) - Another (optional) console output, e.g., in HTerm

After sucessfully starting the kernel on Xilinx Zynq, the following Linux prompt appears:



Congratulations! You have been successfully booting up the preloaded SPI Flash image on the TE07020 GigaZee Zynq SoM.

> ⚠ The preloaded images provided are to demonstrate the board's capability. It is expected that the customer will replace them with their own images.

Boot Procedure

FSBL (First Stage Bootloader)

The primary boot source for the TE0720 is the on-board SPI Flash. After power on the Zynq PS boot ROM fetches the FSBL from the SPI Flash and executes it. Then the FSBL code takes over and initializes the Zynq PS peripherals as well as the DDR3 memory controller. Finally the FSBL proceeds to load the PS object code or FPGA configuration data. Primary factory FSBL for the TE0720 only loads SSBL (Second Stage Bootloader) from SPI Flash and executes it. This way it is possible to boot the O/S (Linux) with or without configuring the FPGA Fabric. Boot configuration can select the order of the images to be loaded. It is possible that the SSBL loads the FPGA only, loads the FPGA and then loads and executes application (O/S) code, or loads the user application without configuring the FPGA fabric.

FPGA images and application code can reside in on-board SPI Flash, in on-board eMMC or on an external SD card.

> ⚠ Note: an FSBL image can not be loaded from eMMC. FSBL is fetched from SPI Flash or from an external SD card.

Note: Xilinx wizard generated FSBL does not properly initialize the SD card detect multiplexer values. By default MIO0 remains selected. If writing your own FSBL it is necessary to initialize it properly. TE0720 standard FSBL selects the EMIO63 pin for SD card detect and WP inputs (this forces the card detect to succeed).

SD0_WP_CD_SEL = 0x003F003F;

SD1_WP_CD_SEL = 0x003F003F;

Pseudo code to initialize the CD and WP selection bit to point to EMIO63 (tied low if not connected in user logic).

SSBL (Second Stage Bootloader)

Second stage bootloader is usually fetched from SPI flash. By default a customized U-Boot is used which is responsible for loading the O/S. If U-Boot functionality is not needed then the user application could be implemented as SSBL image.

Boot Modes

The Zynq-7000 generally supports several boot modes that can be selected by five boot mode pins BOOT_MODE[4:0], where BOOT_MODE[4] / MIO[6] enables the PLLs. The other boot mode pins select the boot source in the following way (see Xilinx Zynq-7000 AP SoC Technical Reference Manual, UG585 v1.6.1, Table 6-2 on page 147):

	BOOT_MODE[0] MIO[5]	BOOT_MODE[2] MIO[4]	BOOT_MODE[1] MIO[3]	BOOT_MODE[3] MIO[2]
Cascaded JTAG				0
Independent JTAG				1
JTAG	0	0	0	0
Quad-SPI	**1**	**0**	**0**	
SD Card	**1**	**1**	**0**	

Table 1: Zynq-7000 AP SoC Boot_Mode MIO Pins (Extract)

As it can be seen in Table 1 the only difference between *Quad-SPI* and *SD Card* boot mode is BOOT_MODE[2] / MIO[4]. This is exactly the only signal which is controlled by the MODE output of the TE0701 on-board CPLD. The BOOT_MODE[0] / MIO[5] and BOOT_MODE[1] / MIO[3] input pins are tightend correspondingly, e.g., on the TE0720 Zynq SoC Module to pull-up (i.e., SPI-DQ3/M3 = PS_MIO5 = 1) and pull-down (i.e., SPI_DQ1/M1 = PS_MIO3 = 0) resistors.

Copyright © 2014 Trenz Electronic GmbH

> The TE0720 Zynq SoC Module on the TE0701 Carrier Boards supports two different boot modes: QSPI and SD Card booting. For more information on configuring the boot mode please refer to "TE0701 Carrier Board User Manual | Configuring Boot Mode".

QSPI Boot Mode

Xilinx Answer record AR47023

> Currently writing to the SPI Flash is only possible from Vivado SDK starting from **2013.4** version. Writing from Impact (any version) is not supported.

U-Boot functions can be used to update SPI Flash.

Xilinx Answer record AR57744 need to be followed, currently the system controller firmware does not provide this safety function.

Update the SPI Flash from an SD Card

To update SPI Flash contents from files stored on an SD card use the U-Boot commands "fatload" and "sf".

Example:

Read boot.bin from SD Card to memory address 0x1000000

> *fatload mmc 0 0x1000000 boot.bin*

Initialize SPI Interface

> *sf probe 0 0 0*

Write 0x450000 bytes from address 0x1000000 to offset 0 in SPI Flash

> *sf update 0x1000000 0 0x450000*

The default U-Boot environment has predefined command sequences to update SD Flash from an SD card. This update contain files used for Linux boot (boot.bin, uImage, uramdisk.image.gz and tevicetree.dtb).

> *run sdfetch*
>
> *run reflash_all*

Update the SPI Flash from a network

To update SPI Flash via Ethernet the U-Boot commands "tftp" and "sf" can be used.

Copyright © 2014 Trenz Electronic GmbH

Host PC network interface should have IP address 192.168.42.2. Direct connection from a host PC to the board is recommended. It is possible to change the IP address of the board from U-Boot, but there could be other issues when the connection goes via a corporate network.

TFTP server software should be installed on the host PC to provide file access via the TFTP protocol. A TFTP server is no longer provided by Microsoft as part of the Windows O/S, so third-party software must be used.

Example of TFTP server configuration

- Download OpenTFTPServer from http://sourceforge.net/projects/tftp-server/
- Install OpenTFTPServer
- Run Open TFTP Server -> Configure from Start menu
- Edit configuration file.

> *[LISTEN-ON]*
>
> *192.168.42.2*
>
> *[HOME]*
>
> *c:\firmware*
>
> *[LOGGING]*
>
> *All*
>
> *[ALLOWED-CLIENTS]*
>
> *[TFTP-OPTIONS]*

- Run Open TFTP Server -> Run Stand Alone

After configuration files in C:\firmware can be downloded via TFTP.

Example:

Download boot.bin from host PC to memory address 0x1000000

> *tftp 0x1000000 boot.bin*

Initialize SPI Interface

> *sf probe 0 0 0*

Write 0x450000 bytes from address 0x1000000 to offset 0 in SPI Flash

> *sf update 0x1000000 0 0x450000*

SD Card Boot Mode

TE0720 can also boot directly from an SD card. In this mode SPI Flash is not used (all code starting from the FSBL is loaded from the SD card).

> ⚠ SDIO0 Bootable slot MIO pins have a 1.8V fixed I/O voltage so the SD card must be connected via a level shifter on the carrier board.

Programming the FPGA with new Configurations

Configuring the FPGA via JTAG

Either Xilinx Impact or ChipScope can be used to load bitstreams over JTAG.

Configuring the FPGA by the Processing System (PS)

Files that can be used for FPGA configuration using the PS DEVCFG interface have to be binary bitstreams with the preamble stripped and the bytes swapped within 32-bit words. The Xilinx bitgen tool can produce files with a BIN extension and a stripped header, but the generated files do not have the correct byte swap.

Example for "promgen" parameters to convert bitstream "top.bit" to byte-swapped binary file "top.bin":

```
promgen.exe -w -b -p bin -o top.bin -u 0 top.bit -data_width 32
```

> ⚠ Bitstreams that do include PS instantiation have to be generated with CCLK as the startup clock.

Configuring the FPGA with U-Boot

The FPGA can be configured or reconfigured from the U-Boot prompt:

```
zynq-uboot> nm 0xf8007080
f8007080: 00000010 ? 0
f8007080: 00000000 ? x
zynq-uboot> nm 0xf8007080
f8007080: 00000000 ? x
zynq-uboot> fatload mmc 0 0x100000 top.bin
reading top.bin
4045564 bytes read in 628 ms (6.1 MiB/s)
zynq-uboot> fpga load 0 0x100000 0x3dbafc
```

Copyright © 2014 Trenz Electronic GmbH

> Writing 0 to address 0xF8007080 is a temporary fix for the first revision of the FSBL shipped. It is required to make the FPGA configuration correctly both in U-Boot and in Linux. An explanation has been given by a Xilinx employee on the Xilinx User Community Forums.

Configuring the FPGA in Linux

The FPGA can also be configured within Linux:

```
zynq> mkdir /tmp/sd
zynq> mount /dev/mmcblk1p1 /tmp/sd
zynq> cd /tmp/sd
zynq> cat top.bin > /dev/xdevcfg
```

> You must apply the write to address 0xF8008070 in U-Boot before booting Linux, otherwise the FPGA load will fail.

Detailed Description

Overview

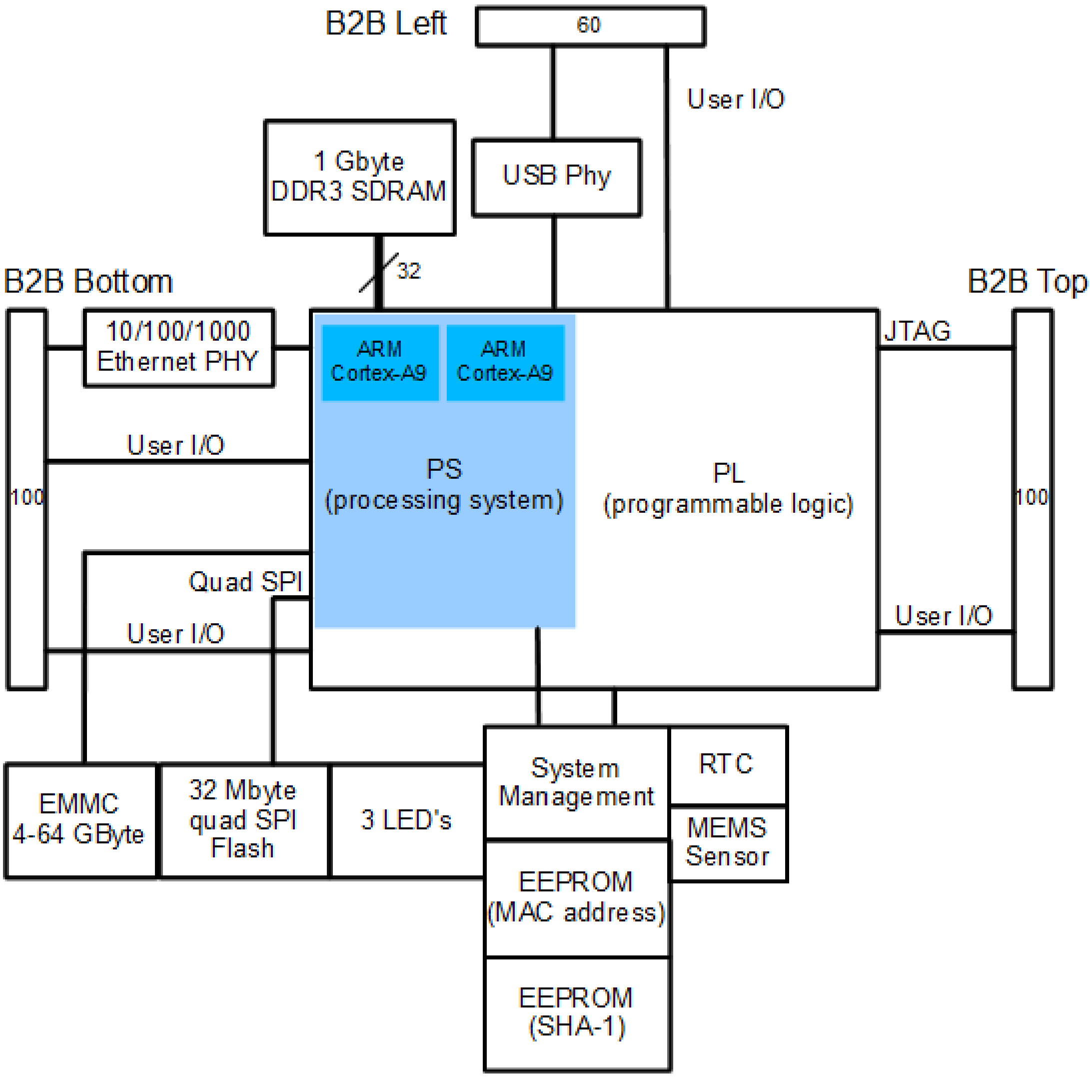


Figure 2: TE0720 GigaZee Zynq SoM Block Diagram

Copyright © 2014 Trenz Electronic GmbH

System Management, Power Supply & Resets

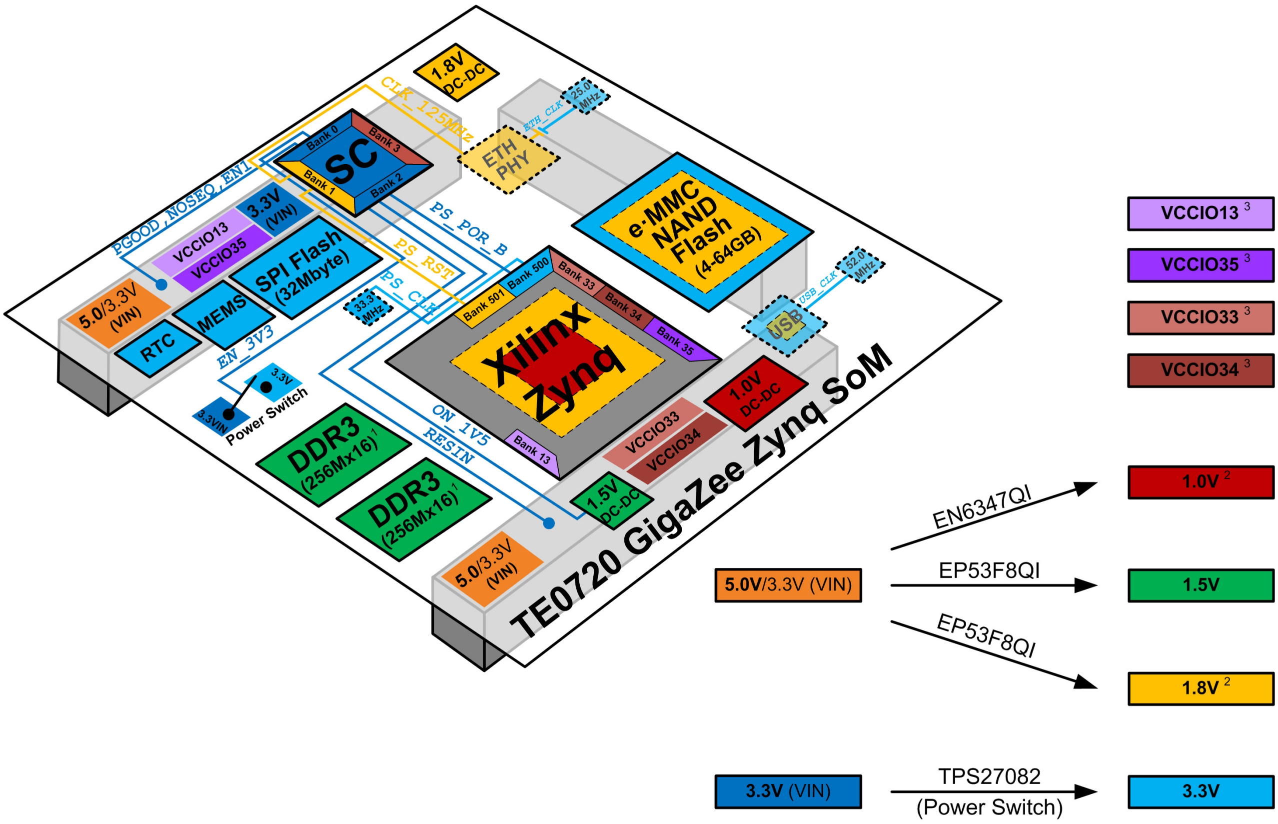


Figure 3: Overview: TE0720 System Management, Power Supply & Resets

[1] Note: The DDR3 SDRAM size depends on assembly option.

[2] 1.0V and 1.8V TE0720 power supply circuits are not shown to create a better overview (see TE0720 User Manual | Power Supply for more details).

[3] Four independent VCCIO voltages (VCCIO13, VCCIO33, VCCIO34, VCCIO35) are fully customizable (1.5V, 1.8V, 2.5V, 3.3V) and can be chosen application-specifically (on a user defined "Carrier Board"). For example, on our "TE0701-03 Carrier Board" the VCCIO voltages have been chosen on the one hand for VCCIO33 and VCCIO34 to be 2.5V by default (or alternatively 3.3V) and on the other hand for VCCIO13 and VCCIO35 to be the user-programmable FMC_VADJ voltage via I2C (see TE0720 User Manual | Carrier Boards for TE0720 - Configuring FMC Power Supply Voltage) by default (or alternatively 2.5V and 3.3V, respectively).

System Management Controller (SC)

Overview: System Management Controller (SC)

A Lattice XO2-1200 CPLD is used as a System Management Controller (referred to as SC in the manual). The SC is responsible for power sequencing, reset generation and Zynq initial configuration (mode pin strapping). Moreover, some on-board ICs are connected to the SC that provides level shifting.

It is possible for the default SC functions and pin functions to be changed. This can be done as a request to Trenz Electronic or it is possible for the user to generate their own designs. Please contact us for details.

The SC wakes up when the 3.3V input power rises above 2.1V (VIN voltage is not needed). The SC can turn on or off all of the other supplies on the module (except in no power sequencing mode when the 1.0V and 1.8 V supplies are forced to start immediately when power is applied to the module).

Custom SC Programming

SC customization is available either by requesting new features or with special agreement by using the users own code. SC code can be updated in the system using the I2C interface. Please contact us for details.

SC B2B Pins

SC is connected directly to the following B2B Pins.

Name	Mode	Default function	Alternative	Description
EN1	input, weak pull-up	Power Enable	IO	Enables the DC-DC converters and on-board supplies. Not used if NOSEQ=1
PGOOD	output, open drain	Power good	SCL or IO	Forced low until all on-board power supplies are working properly.
MODE	input, weak pull-up	Boot mode	SDA or IO	Force low for boot from the SD Card. BOOTMODE is latched at power on only.
RESIN	input, weak pull-up	Reset input	IO	
NOSEQ	input, weak pull-down	Power sequencing Control	Output	Forces the 1.0V and 1.8V DC-DC converters ON when high. Can be used as an I/O after boot.

NOSEQ Pin

This is a dedicated input that forces the module's 1.0V and 1.8V supplies to be enabled if high. This pin has a weak pull-down on the module. If left open the module will power up in normal power sequencing enabled mode. This pin is 3.3V tolerant. This pin is also connected to the System Management Controller. The SC can read the status of this pin (that is it can detect if the module is in power sequencing enabled mode). The SC can also use this pin as output after normal power on sequence. Please check the SC description for the function. SC rev 0.02 maps Ethernet PHY LED0 to NOSEQ by default (the mapping can be changed by software after boot).

No Sequencing mode

If the module is powered from a single 3.3V supply and power sequencing is disabled, then NOSEQ pin should be powered from the main 3.3V input. That is VIN, 3.3Vin and NOSEQ should all be tied together to the input 3.3V power rail. Sequencing mode should not be used if VIN is not 3.3V.

Normal mode

For normal operation leave NOSEQ open or pull down with a resistor.

Normal mode with user function on NOSEQ

NOSEQ can be used as an output after boot. NOSEQ must be low when 3.3V power is applied to the module. Common usage is an LED connected between NOSEQ and GND.

SC pins to the FPGA

	Default function	Direction	Description
XCLK	Clock to FPGA	to FPGA	
XIO1	Clock from FPGA	from FPGA	SCL from EMIO I2C1
XIO2	Data from FPGA	from FPGA	SDA from EMIO I2C1
XIO3	Data to FPGA	to FPGA	SDA to EMIO I2C1
XIO4	ETH PHY LED1	to FPGA	
XIO5	ETH PHY LED2	to FPGA	
XIO6	Interrupt	to FPGA	RTC, MEMS

Default Mode

At power up the System Management Controller starts with default settings.

Pin/Function	Used as/Mapped to	Notes
ETH PHY LED0	XIO to FPGA	
ETH PHY LED1	XIO to FPGA	
ETH PHY LED2	Not used	
ETH PHY CONFIG	Tied logic low	PHY Address set to 0
ETH CLK125MHz	Pass through FPGA B34 SRCC pin	
ETH Clock Enable	Tied logic high	
ETH PHY Reset	Internal RESET	
MIO7	LED1	
MEMS/RTC I2C	XIO to FPGA	
RTC Interrupt	-	
MEMS Interrupt 1	-	
MEMS Interrupt 2	-	

Pin/Function	Used as/Mapped to	Notes
eMMC Reset	Internal RESET	
USB PHY Reset	Internal RESET	
FPGA PUDC	Tied logic low	
FPGA PROG_B	Tied logic high	
Zynq Cascaded JTAG	Enabled (pulled low)	
Zynq boot mode	SPI or SD, depending on bootmode pin	
Zynq SRST	Tied logic high	
Zynq POR	Internal POR/Reset	
PLL	Not used	
LED2	System Status LED	
LED1	MIO7	
NOSEQ Input	NOSEQ at power, LED out after boot	
Power Good 1.5V		
Power Good VTT		
MODE Input		

I2C Address	Function	
0x20	Status reg 1	
0x21	Status reg 2	

LED Control Status

The TE0720 on-board LED devices can be remapped to different functions.

Input port bit	Mapped to
0	Ethernet PHY LED0 output
1	Ethernet PHY LED1 output
2	Ethernet PHY LED2 output
3	PS MIO7
4	Returns RESIN pin level
5	Returns EN1 pin level
6	Returns NOSEQ pin level
7	Returns MODE pin level

LED1 and LED2 function can be changed from the default behaviour using output port bits (3..0)

Copyright © 2014 Trenz Electronic GmbH

D3	D2	D1	D0	LED1 function as
0	0	0	0	Default (MIO7)
0	0	0	1	ETH PHY LED0
0	0	1	0	ETH PHY LED1
0	0	1	1	ETH PHY LED2
0	1	0	0	MIO7
0	1	0	1	Undefined
0	1	1	0	OFF
0	1	1	1	ON
1	x	x	x	Undefined

SC Demystified

System Controller (SC for short) was designed to allow ZYNQ PS system to access module special functions as early as possible without reducing the number of MIO pins that are fully user configurable.

This early communication channel is done using MIO52 and MIO53 pins that are used also as Ethernet PHY management interface for the on-board Gigabit PHY.

In order to simplify the boot process and reduce the number of time the PS peripherals need to be configured or re-initialized SC uses the same protocol on MIO52/MIO53 as the Gigabit PHY itself. This means that FSBL Configures all peripherals to their final function, allocating MIO52 as MIO52 as Ethernet MDIO Interface.

SC Controller appears as "Virtual Ethernet PHY" on the MDIO bus of PS Ethernet 0 Interface. This interface is already available when Zynq PL Fabric is not configured.

It would have been possible to use I2C Protocol on MIO52/MIO53 but in such case some multiplexing would be be needed to choose between two protocols, also it would be needed to change the Peripheral mapping after first init by the FSBL.

For use cases where Ethernet PHY on TE0720 is not used at all, it is still possible to configure SC with design that implements I2C Protocol on MIO52/MIO53 pins.

For most use cases the only need to use this interface is access to MAC Address info, this is normally done by u-boot loader that fetches the MAC Address bytes and sets its environment variables accordingly. Linux image will then also be started so that the MAC Address from EEPROM is used for Ethernet 0 Physical interface.

> Important: u-boot still has trouble properly initializing Ethernet PHY registers. So that with standard u-boot distribution MII Commands are not available from withing u-boot, those commands will just fail, and all 0's will be read from all PHY addresses. This is not a design issue with SC, it is u-boot problem that is not fixed in mainstream u-boot repositories. PHY Init has changed with u-boot versions but the problem is still there, and more severe in the latest revisions of u-boot. PHY initialization fix in u-boot board support file that worked in 2013.1 uboot is no longer working in 2014.1 uboot version.

COM8 - PuTTY

```
U-Boot 2014.01-dirty (Jun 13 2014 - 07:08:53)

I2C:   ready
Memory: ECC disabled
DRAM:  128 MiB
MMC:   zynq_sdhci: 0
SF: Detected S25FL256S_64K with page size 256 Bytes, erase size 64 KiB, total 32 MiB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   Gem.e000b000
Hit any key to stop autoboot:  0
zynq-uboot> mii info
PHY 0x00: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x01: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x02: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x03: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x04: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x05: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x06: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x07: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x08: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x09: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x0A: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x0B: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x0C: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x0D: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x0E: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x0F: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x10: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x11: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x12: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x13: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x14: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x15: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x16: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x17: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x18: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x19: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x1A: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x1B: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x1C: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x1D: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x1E: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
PHY 0x1F: OUI = 0x0000, Model = 0x00, Rev = 0x00,  10baseT, HDX
zynq-uboot> mm e000b000
e000b000: 00000000 ? 10
e000b004: 00080000 ? x
zynq-uboot> mii info
PHY 0x00: OUI = 0x5043, Model = 0x1D, Rev = 0x01,  10baseT, HDX
PHY 0x1A: OUI = 0x7202, Model = 0x01, Rev = 0x00,  10baseT, HDX
zynq-uboot>
```

Initial MII info in uboot 2014.1 finds "fantom" PHYs that all read 0's from all registers. After enabling MII Interface (that u-boot itself should do during init) MII Info command shows correctly PHY at address 0 what is the Marvell Gigabit PHY and virtual PHY at address 0x1A what is the SC. The above is what uboot will show if the MDIO enable bit was not set in the FSBL.

SC Firmware ver 0.02

This is the initial version of the System Controller with only a very limited function set implemented.

System Controller can be accessed as PHY with address 0x1A on the ETH0 Management bus (MIO pins 52, 53). Communication can be established anytime when ETH0 and management interface are enabled also before FPGA PL Fabric is configured. Xilinx default FSBL and standard u-boot do not enable the management interface. It can be done manually in u-boot

```
zynq-uboot> mm e000b000
e000b000: 00000010 ? 10
e000b004: 00080000 ? x
zynq-uboot>
```

The above command enables the management interface on ETH0. TE0720 FSBL does this already so there is no need to do it manually. This command is needed when standard Xilinx wizard generated FSBL is used with plain standard u-boot, TE0720 u-boot does this initialization also. Note: It seems the problem is in the current u-boot for Zynq, where mii_init function is not defined and not invoked.

Version check

System Controller Firmware version and some other version info can be read with u-boot command **mii info**:

```
zynq-uboot> mii info
PHY 0x00: OUI = 0x5043, Model = 0x1D, Rev = 0x01, 100baseT, FDX
PHY 0x1A: OUI = 0x7201, Model = 0x01, Rev = 0x00,  10baseT, HDX
zynq-uboot>
```

PHY at address 0x00 is the ETH0 onboard ethernet PHY Marvell 88E1512.

PHY at address 0x1A is the System Controller. OUI 0x7201 should be decoded as Model TE0720-01. Model 0x01 is Assembly option. Rev 0x00 is the firmware major revision for the System Controller (Rev 0 is the initial version).

Bit Decoding

Reg Addr	Bits	u-boot ENV Variable	Description
2	15:0	board	upper bits of SoM Model
3	15:10	board	lower bits of SoM Model
3	9:4	-	Assembly Variant
3	3:0	scver	SC Firmware Revision Major number
4	15:14	board	FPGA Speed Grade
4	13:12	board	FPGA Temperature Range (0=Commercial, 1=Extended, 2=Industrial)
4	7:0	scver	SC Firmware Revision Minor number

Customized u-boot reads and decodes the model and assembly variant information and stores in readable format in environment variables.

```
zynq-uboot> printenv board
board=TE0720-01-2IF
zynq-uboot>
```

Reading MAC Address

With u-boot command **mii read**:

```
zynq-uboot> mii read 1a 9-b
addr=1a reg=09 data=0004
addr=1a reg=0a data=A3AC
addr=1a reg=0b data=3911
zynq-uboot>
```

This command will read MAC Address from the System Controller. Note: This only works if the ETH0 interface is enabled and if FSBL has enabled MII Management console on ETH0 Interface. 0004A3 is OUI part, AC3911 is the serialized part (lower bits of MAC address).

Customized u-boot does read MAC Address and stores it in environment variables as required, as a result, proper MAC address is used both in u-boot as also in Linux. Setting up MAC Address for Linux involves dynamic rewrite of FDT, this is done with u-boot script that starts Linux.

SC Registers

Most registers and functions are available via ETH PHY Management interface (MIO pins 52 and 53).

Addr	R/W		Descripion
0	RO		
1	RO		
2	RO	ID1	Identifier Register 1
3	RO	ID2	Identifier Register 2
4	RO	ID3	Identifier Register 3
5	RW	CR1	Control Register 1: LED's
6	RW	CR2	Control Register 2; XIO Control
7	RW	CR3	Control Register 3; Reset, Interrupt
8	RO	SR1	Status Register
9	RO	MAChi	Highest bytes of primary MAC Address
0xA	RO	MACmi	Middle bytes of primary MAC Address
0xB	RO	MAClo	Lowest bytes of primary MAC Address
0xC	-		reserved do not use
0xD	RW	MMD_CR	MMD Control Register
0xE	RW	MMD_AD	MMD Address/Data
0xF	-		reserved do no use

Copyright © 2014 Trenz Electronic GmbH

Addr	R/W		Descripion
other	-		reserved do not use

System Controller version 0.02 does not support extended address space - registers 0xD and 0xE are read-write accessible but do not have any function. In feature revision extended address will be used to control SC PLL and other features.

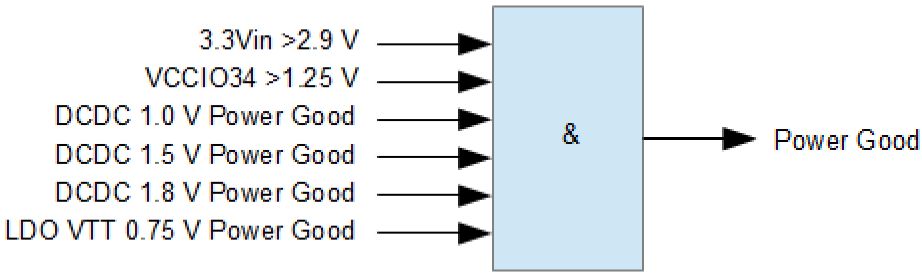
Power Supply

Power Supply Specifications

High-performance DC-DC converters from Enpirion are used for most of the power rails.

Power Sequencer and System Management

Xilinx ZYNQ devices require special power sequencing. The voltage difference between 1.8V VCCAUX and any I/O Voltage must not be higher than 2.65V for a combined duration of 800 ms for each power on-off cycle (or 500 ms at higher temperatures). If TE0720 is operating in "no power sequencing" mode this period time of 800 ms (and 500 ms, respectively) will never be violated in normal use. TE0720 can also be used in power sequencing mode where the 3.3V voltage plane is supplied from 3.3Vin (CPLD power supply) by a dedicated power FET switch (that is, in turn, controlled via an enable signal by the CPLD SC) after the 1.0V and 1.8V supplies have stabilized. However, in power sequencing mode user circuitry must also comply to power-up sequencing rules; VCCIO voltages that are over 2.5V must be turned off when TE0720 has not turned on the core supplies.



Simplified Power Good signal generation.

Power Rails

Rail	Input/Output	Powered from	V	I Rating	Monitored	UVLO	XADC
Vin	Input	External	3.3 - 5.5	8A Connector	>2.2V (indirect by DCDC)	2.2V	
Vin 3.3V	Input	External	3.3	2A Connector	>2.5V (indirect by SC)		
3.3V	Output *1	Vin 3.3V	3.3	2A Connector	>= 3.05V		
1.0V	Internal	Vin	1.0	4A DCDC	+-10%	2.3V	Yes (internal)
1.8V	Internal / Output	Vin	1.8	1.5A DCDC	+-10%	2.2V	Yes (internal)
1.5V	Internal / Output	Vin	1.5	1.5A DCDC	+-10%	2.2V	Yes (internal)
VTT	Internal	1.5V	0.75	+-2A LDO	+-20%	2.3V	indirect via VTTREF
VTTREF	Internal	1.5V	0.75	+-10mA LDO	+-20 (indirect tracks VTT)	2.3V	Yes (Channel 0)
VCCIO_34	Input	External *2	1.5 - 3.3	2A Connector	>= 1.25V		
VCCIO_13	Input	External *2	1.2 - 3.3	2A Connector	not monitored		
VCCIO_33	Input	External *2	1.2 - 3.3	1A Connector	not monitored		

Rail	Input/Output	Powered from	V	I Rating	Monitored	UVLO	XADC
VCCIO_35	Input	External *2	1.2 - 3.3	2A Connector	not monitored		

*1 When used in 3.3V single power supply mode with no sequencing this pin can be connected to Vin 3.3V

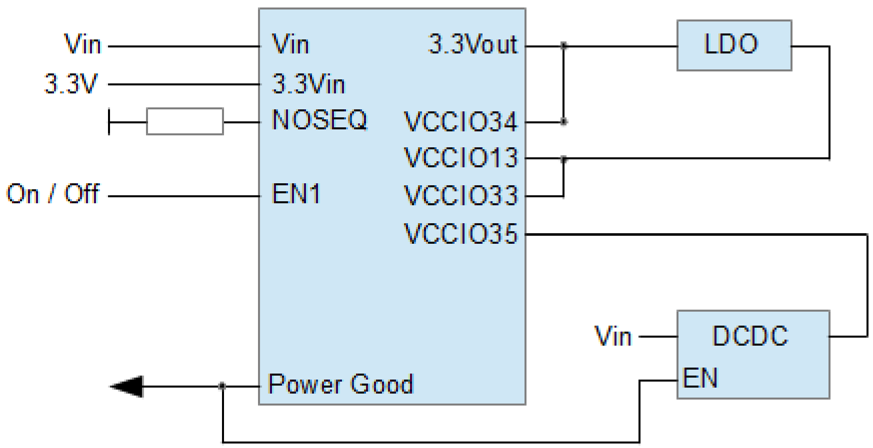
*2 PL I/O Bank VCCIO Inputs can be connected to the module's 3.3V output (for 3.3V I/O Voltage Banks).

I/O Voltages

I/O Bank	Voltage	Notes
PS MIO Bank 0	3.3	SPI Flash, 8 MIO pins on B2B connector
PS MIO Bank 1	1.8	USB PHY, Ethernet PHY, eMMC, 6 MIO pins on B2B connector
PL Bank 0	3.3	
PL Bank 34	1.5 - 3.3	When Bank 34 VCCIO is below 1.25, ZYNQ is held in POR reset state
PL Bank 13	1.2 - 3.3	
PL Bank 33	1.2 - 3.3	
PL Bank 35	1.2 - 3.3	
Ethernet PHY	1.8	
USB PHY	1.8	
eMMC	1.8	

Example Application Diagrams

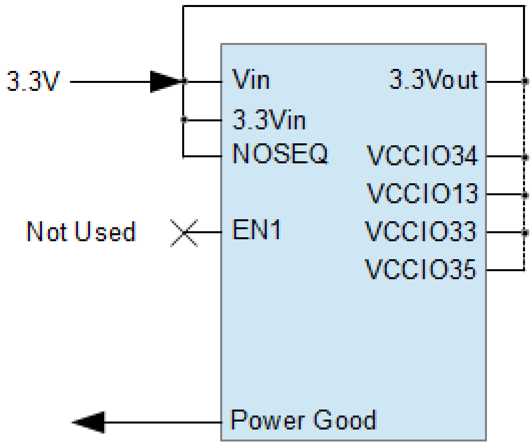
Dual Supply Application



Example power connections. On-board DC-DC converters are supplied separately from 3.3V circuits. All VCCIO Bank Voltages should be derived either from the 3.3V output or controlled by the Power Good signal.

For single supply mode with power sequencing enabled, use the same configuration with Vin powered from 3.3V.

3.3V Single Supply with no Power Sequencing



This is simplest application where all power is derived from single a single 3.3V supply with no power sequencing. All FPGA I/O Banks should be powered from the same 3.3V supply either directly or using a LDO/DC-DC supply. This configuration is used for backwards compatibly with motherboards designed for the TE0600 module. EN1 pin in this configuration does not turn off any of the on-board power supplies. Holding EN1 pin low will assert the POR (power on reset) to the ZYNQ. EN1 can be left floating, pulled up or connected to 3.3V.

XADC Power

For best noise performance XADC is powered from a separate on-board, low power, 1.8V LDO. The LDO provides an Analog power supply for the ZYNQ XADC circuits and also for an on-board 1.25V reference IC. Those components are placed within 3 mm of the BGA package balls on the PCB bottom side.

Backup Battery

The RTC IC is powered from a backup battery which should have a nominal voltage of 3.2V to 3.3V. The ZYNQ internal AES security RAM is also powered from the same VBATT pin as the RTC chip. The backup supply pin on the ZYNQ is connected to VBATT via an ultra-low power LDO.

Operating with a power supply of less than 3.1V

The TE0720 System Managment Controller normally prevents the ZYNQ device from booting if the power is less than 3.05V. In certain cases it is possible to allow the TE0720 module to be operated from a single 2.5V to 2.7V supply. At these voltages the SPI flash boot option is not available and the on-board eMMC is not usable. Boot is only possible via JTAG, or optionally from external SD Card supporting low-voltage operating modes. This option is only considered on special request.

Copyright © 2014 Trenz Electronic GmbH

AC & DC Characteristics

AC Characteristic

	-3	-2	-1
ARM CPU Max	800	733	667
PS DDR3	1066	1066	1066
Logic F/F Toggle	1412	1286	1098
Block RAM	509.68	460.83	388.20
DSP48E1	628.93	550.66	464.25
Global Clock Tree and BUFH	628.0	628.0	464.4
I/O Clock Tree	680	680	600
Regional Clock	420	375	315
MMCM/PLL input or output	800	800	800

Max AC performance in MHz per ZYNQ speed grade. The above table shows some maximum clock frequencies for the PS and PL subsystems of the ZYNQ SoC. Please consult Xilinx datasheets for detailed performance information.

DC Characteristic

Power input pin	Max current
VIN	
3.3Vin	300mA
VBATT	

Lowest on-board consumption is achieved when powering the module from single 3.3V supply. When using split 3.3V/5V supplies the power consumption (and heat dissipation) will rise, this is due to the DC/DC converter efficiency (it decreases when VIN/VOUT ratio rises).

Typical power consumption is between 2-3W.

Ethernet PHY Power-down

If on-board ethernet PHY is not used it can be forced into full powerdown, it requires 3 different setting in PHY registers that force the Copper Interface power-down, RGMII power-down and disable the CLK125 output as well.

Write 0x0800 to register 0 page 0, and 0x0003 to register 16 page 2.

Power saving about 60mW from the PHY power-down, this is only from the PHY IC, Zynq ETH Interface is still in used and clocked, so additional power saving could be achieved by disabling PS ETH as well.

Ethernet PHY clock oscillator can be forced into standby, additional 10mW savings.

USB PHY Power-down

USB PHY can be forced into power down by setting the USB PHY Reset active.

Copyright © 2014 Trenz Electronic GmbH

Resets

The system pin called RESIN is mapped to the Zynq POR Reset pin as default. The function of this pin can be changed during the boot process.

Note: If TE0720 is used with a full O/S like linaro/ubuntu, and the internal eMMC or external SD card is mounted as a Linux live file system, then it is not recommended to reset the system by asserting RESIN. RESIN pin function can be changed from POR reset to an interrupt, enabling the O/S to do a proper shut down. Please check System Management Controller section.

Software forced Resets

The SC starts the Zynq with a cascaded JTAG chain bootstrap option after initial power on. If an independent JTAG option is desired then the Zynq should be restarted (force POR), which latches the new bootstrap options. The SC can do this under software control.

Function not available with SC version 0.2

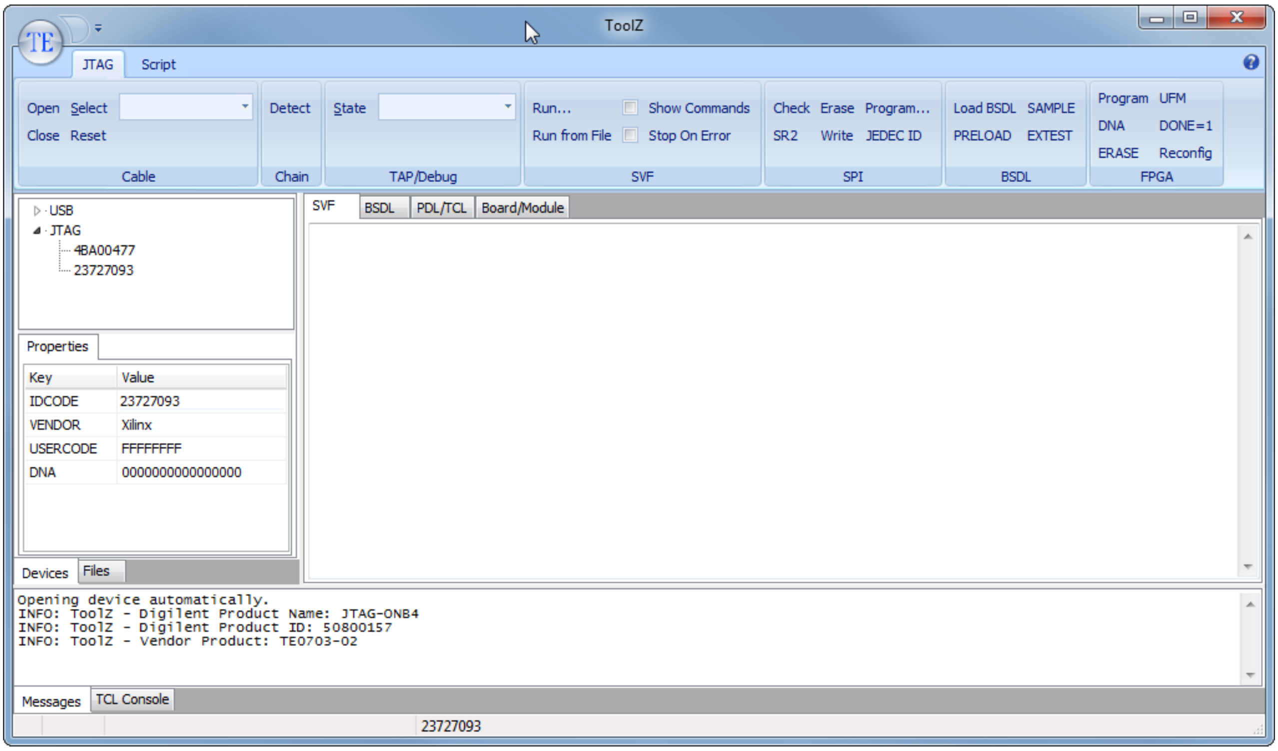
Peripheral Resets

Some on-board peripherals have separate reset inputs. Those are normally asserted by the SC during initial power-up sequence. After the boot process those reset pins can be controlled via the SC.

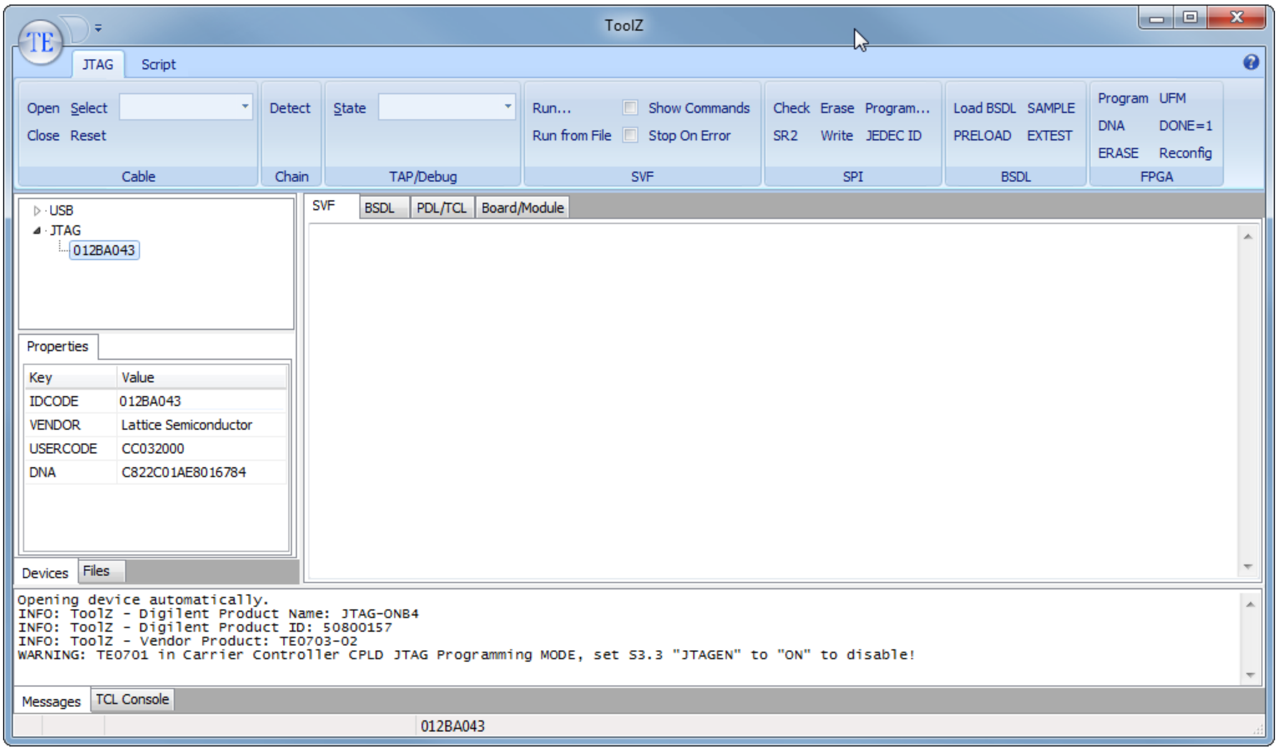
SC update for TE0720-02

TE0720 SC can be updated on TE0703 base board (or on custom baseboards).

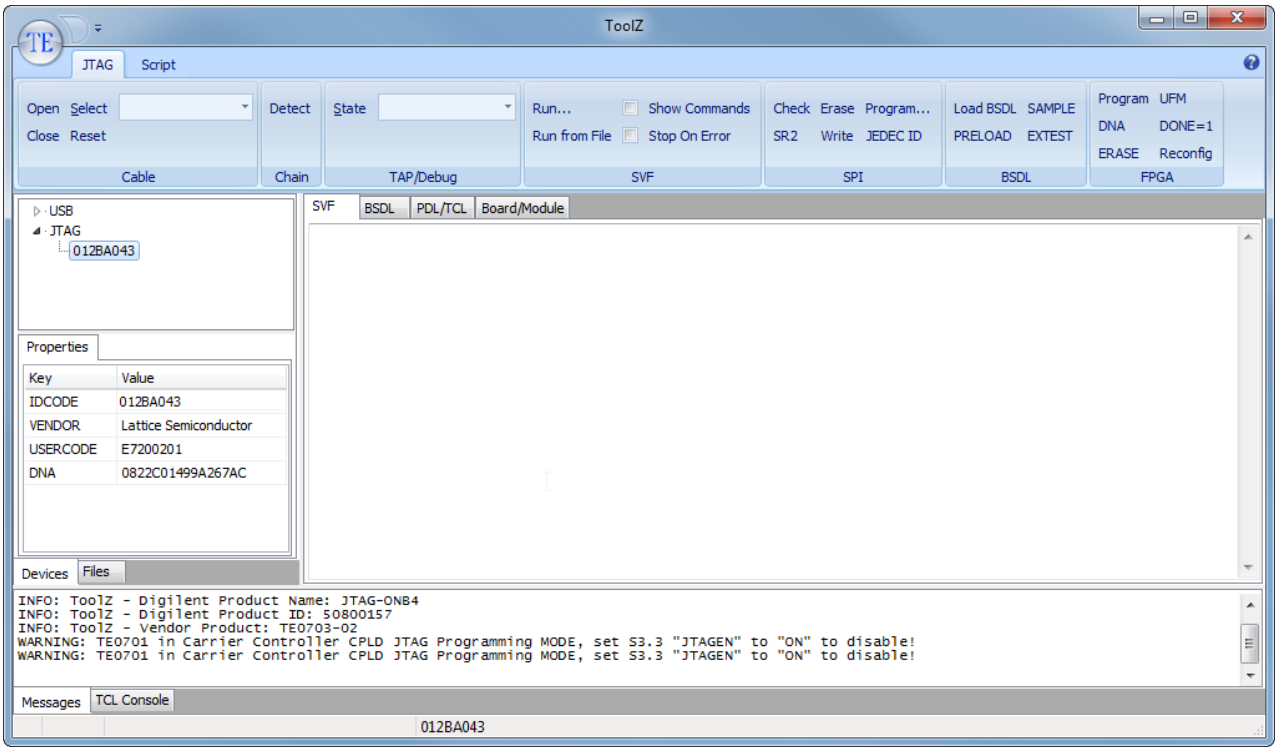
It is best to check the JTAG configuration with ToolZ to be sure what IC is selected in Chain.



Zynq is selected in JTAG, this is normal operational mode. All switches left (ON).



Only S3.3 is moved to right (OFF) position. TE0703 onboard baseboard controller CPLD is in JTAG chain with USERCODE CC03xxxx



Only S3.2 is moved to right (OFF) position. TE0720 onboard system controller CPLD is in JTAG chain with USERCODE E720xxxx,

TE0703 LED's D1, D2 should be both LIT. If not then it is needed to update the CPLD on TE0703 first, to enable module controller programming.

Board-level Components

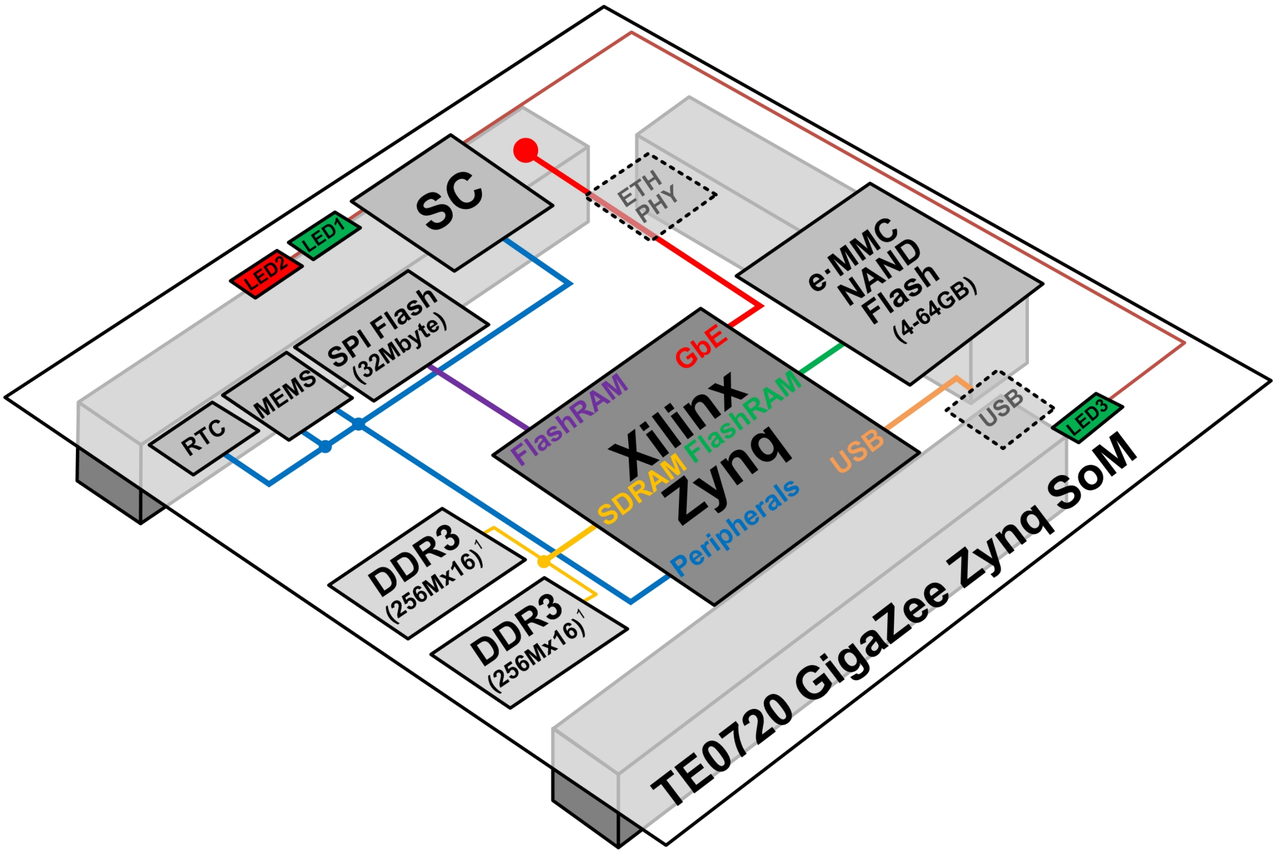


Figure 4: Overview: TE0720 Board-level Components

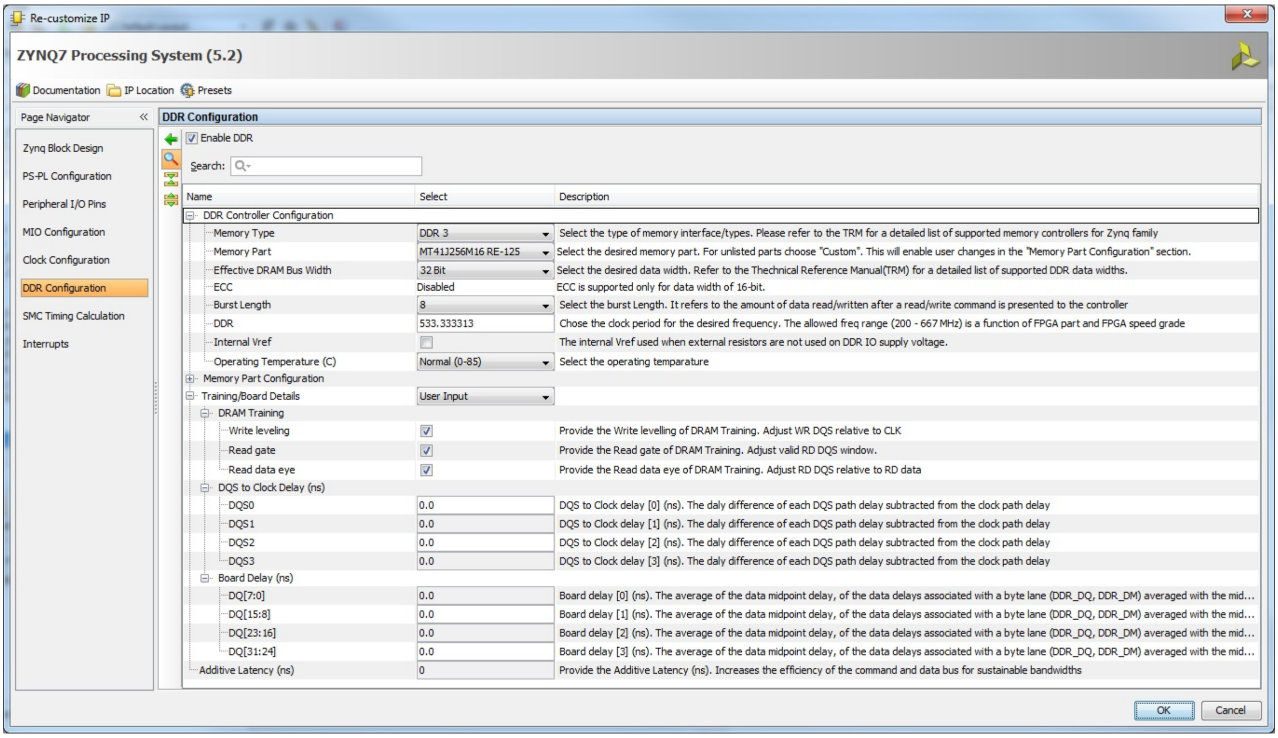
[1] Note: The DDR3 SDRAM size depends on assembly option.

DDR3 SDRAM

Memphis MEM4G16D3EABG-125I DDR3 SDRAM is used which is fully compatible to Micron MT41J256M16. Two RAM devices are used in a fly-by topology configuration with a 32-bit data width.

> ⚠ Different DDR3 devices may be used on different module derivatives.

Configuration



Setting the DDR3 configuration for the TE0720 is straightforward.

Select "Memory Part" as shown in the diagram.

Select "Effective DRAM Bus Width" as 32-bit.

Ensure that "Internal Vref" is disabled.

Set all delays to 0.

Optimal delays are not zero, so it is recommended to load the board initialization file were correct delays are pre-defined.

Manufacturer Documentation

Name	Version	Date
DDR3-settings.jpg	1	2013-06-28 10:21
MEM4G16D3EABG_10.pdf	1	2013-04-06 15:51

e·MMC

Managed NAND - e·MMC is supported by newer version of Xilinx FSBL directly as secondary boot media when FSBL itself is fetched from QSPI Flash.

> ⚠ A special compile flag must be set to enable e·MMC support in FSBL. This flag is not needed for e·MMC support in Linux or U-Boot.

	Version	Notes
FSBL	ISE >= 14.6?	eMMC supported as secondary media
U-Boot		supported but can not format blank media
Linux		fully supported

The e·MMC reset input is connected to the SC and is set normally high (not in reset).

> ⚠ The eMMC reset input is NOT activated in e·MMC after power is applied. So forcing e·MMC reset low has no effect unless e·MMC command to enable reset is sent.

Format internal eMMC Card (Linux)

Full equipped TE0720-01-*F modules have onboard 4G eMMC card. By default this card not have partitions and not formatted. Below you can see sample eMMC configuration (One primary partition, Linux EXT2 filesystem)

Connect to the board using ssh connection or serial console via J5 mini-USB connector

Execute linux commands to create partition

> *zynq> fdisk /dev/mmcblk0*
>
> *The number of cylinders for this disk is set to 117248.*
> *There is nothing wrong with that, but this is larger than 1024,*
> *and could in certain setups cause problems with:*
> *1) software that runs at boot time (e.g., old versions of LILO)*
> *2) booting and partitioning software from other OSs*
> *(e.g., DOS FDISK, OS/2 FDISK)*
>
> *Command (m for help): n*

In command menu press 'n' to create new partition and then Enter.

> *Command action*
> *e extended*
> *p primary partition (1-4)*

Press 'p' to create primary partition and then Enter.

> *Partition number (1-4): 1*

Press '1' for first partition and then Enter.

> *First cylinder (1-117248, default 1):*

Press Enter to select default value.

> *Last cylinder or +size or +sizeM or +sizeK (1-117248, default 117248):*

Press Enter to select default value.

> *Command (m for help):*

Press 'w' to save changes and exit.

Execute linux command to format partition. To EXT2

> *zynq> mke2fs /dev/mmcblk0p1*

To FAT

> *zynq> mkdosfs /dev/mmcblk0p1*

After formatting eMMC drive is ready to use and will be mounted after next reboot.

To mount it immediately

> *zynq> mount /mnt/mmc*

Ethernet

Overview: Ethernet

TE0720 uses a Marvell Alaska 88E1512 Gigabit Ethernet PHY. PHY Datasheets and documentation if needed are available from Marvell (an NDA is required). The Zedboard uses the same PHY with the only difference being the package. The 88E1512 device is the only Marvell PHY from the 88E15xx series supporting an industrial temperature range and a 1.8V I/O Voltage.

The Ethernet PHY RGMII Interface is connected to the Zynq Ethernet0 PS GEM0 (MIO Pins MIO16..MIO27), I/O Voltage is fixed at 1.8V for HSTL signalling.

The internal regulators of the PHY are not used and all power for the device is supplied from the TE0720 DC-DC supplies.

Cable Diagnostic and VCT

Marvell Alaska PHY devices include a built-in Cable Diagnostic Feature and VCT (Virtual Cable Tester). These functions are available over the MDI interface. More information can be found in the Marvell documentation.

Media Autodetect

Power up defaults enable fiber-copper auto-detection, with fiber being the preferred media. If this setting is not changed the PHY will always transmit on SGMII TX pins, also if link is in copper mode. To disable fiber/SGMII

```
zynq-uboot> mii write 0 0x16 0x12
zynq-uboot> mii write 0 0x14 0x8210
zynq-uboot> mii write 0 0x16 0
```

The above writes select page 20, then disable the auto media detect with PHY reset and then select Page 0 again.

Advanced PHY Features

Marvell Alaska PHY devices include support for PTP and SyncE. In these modes LED and CONFIG will be mapped to different hardware functions. TE0720 System Controller can map those pins to the Zynq PL I/O pins so that all operational modes of the PHY can be utilized.

Temperature sensor

Example how to read temperature with u-boot

```
zynq-uboot> mii write 0 0x16 6
zynq-uboot> mii read 0 0x1b
0C54
zynq-uboot>
```

Write to register 22 selects page 6, read from register 26_6 returns temperature value in lower 8 bits. Temperature in Celsius = R26_6[7:0] - 25

PHY Connections

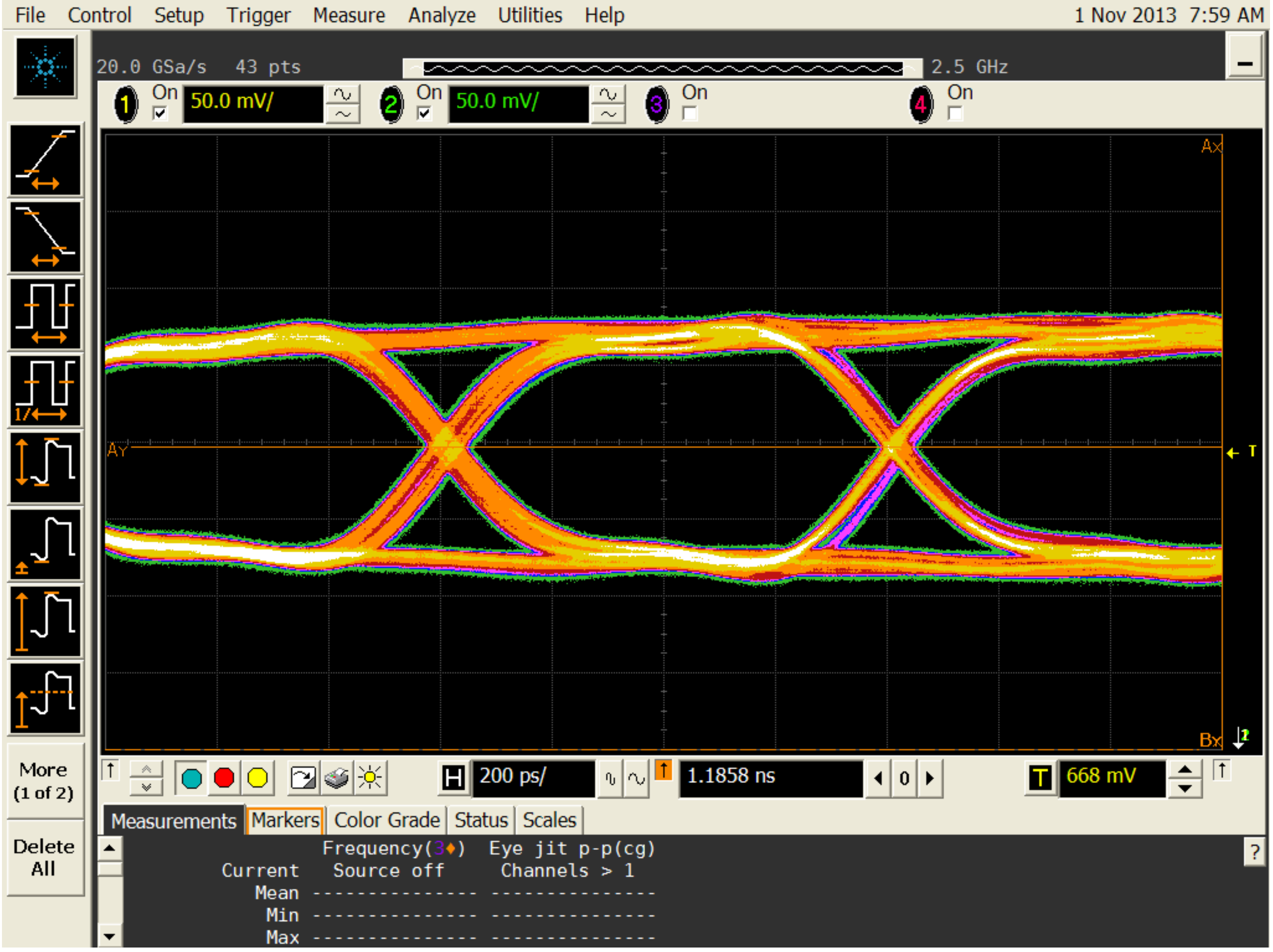
Because the Zynq PS has limited pins (MIO Pins) only RGMII and MDI pins from the PHY are connected directly to the Zynq device. The remaining pins are connected to the SC that provides logic level conversion and interface translation. When the PL is configured those LED pins can optionally forwarded to the PL Fabric. It is also possible to assign the PHY LEDs to the TE0720 on board LED*s.

PHY PIN	Alternate Function	ZYNQ PS	ZYNQ PL	SC	B2B Connector
MDC	-	MIO52	-	yes	-
MDIO	-	MIO53	-	yes	-
LED1		-	XIO	yes	*
LED2		-	XIO	yes	*
LED3	Interrupt	-	-	yes	*
CONFIG		-	-	yes	-
RST	-	-	-	yes	-
RGMII	-	MIO16..MIO27	-	no	-
SGMII		-	-	-	yes
Copper		-	-	-	yes

By default the PHY Address is strapped to 0x00. SC is capable of changing it to 0x01 if needed.

Copyright © 2014 Trenz Electronic GmbH

SGMII/Fiber



SGMII signal EYE captured using SFP2SMA and 1.5m long SMA cables.

> ⚠ Marvell PHY is most likely be left in auto-detect mode after system boot, in such case if copper media is not detected SGMII output is enabled and transmitting. If this is not desired the PHY should be programmed to disable SGMII autodetection.

.

PHY LED Control

The Ethernet PHY LEDs are not directly available on the B2B Connectors.

The SC can however remap the PHY LED signals.

By default the NOSEQ pin is converted to an output pin after the boot process and PHY LED0 is mapped to this B2B pin. During the boot process it is also possible to change this behaviour.

PHY LED0, LED1 and LED2 are also made available to be used in the FPGA fabric where they can be routed to any free FPGA pins.

Signal Name	B2B/FPGA	LED #	PHY default function	
NOSEQ	JM1.	LED0		After boot process PHY LED0 is mapped to NOSEQ
XIO4	M15	LED0	1G/100M: ON=Link, Blink=Activity, OFF=No Link	
XIO5	N15	LED1	100M/10M: ON=Link, Blink=Activity, OFF=No Link	
XIO6	P16	LED2		Note: must be enabled in software

SC rev 0.2 default mapping of Ethernet LED's to B2B and or FPGA Pins

⚠ Marvell PHY LED pins are multipurpose pins with shared and configurable functions.

Default behavior

If Marvell PHY LED control register is not changed during boot process then PHY power up default LED settings apply. Please consult Marvell datasheet for exact features.

PHY LED Demo Design

This demo is for TE0701, for other carrier or custom base board please change user LED mappings as needed.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity PHY_LED_TEST is Port (
 PHY_LED0_IN : in STD_LOGIC; -- forwarded signal from PHY LED[0] output
 PHY_LED1_IN : in STD_LOGIC; -- forwarded signal from PHY LED[1] output
 PHY_LED2_IN : in STD_LOGIC; -- forwarded signal from PHY LED[1] output

 PHY_LED0_OUT : out STD_LOGIC; -- USER I/O Signal in PMOD J5
 PHY_LED1_OUT : out STD_LOGIC; -- USER I/O Signal in PMOD J5
 PHY_LED2_OUT : out STD_LOGIC); -- USER I/O Signal in PMOD J5
end PHY_LED_TEST;
```

```
architecture Behavioral of PHY_LED_TEST is

begin
 PHY_LED0_OUT <= PHY_LED0_IN; -- just route LED signal from phy led out to I/O ir

 PHY_LED1_OUT <= PHY_LED1_IN; -- just route LED signal from phy led out to I/O ir

 PHY_LED2_OUT <= PHY_LED2_IN; -- just route LED signal from phy led out to I/O ir

end Behavioral;
```

Complete code for PHY LED Demo

```
NET "PHY_LED0_IN" IOSTANDARD = LVCMOS33;
NET "PHY_LED0_OUT" IOSTANDARD = LVCMOS33;
NET "PHY_LED2_IN" IOSTANDARD = LVCMOS33;
NET "PHY_LED2_OUT" IOSTANDARD = LVCMOS33;
NET "PHY_LED1_IN" IOSTANDARD = LVCMOS33;
NET "PHY_LED1_OUT" IOSTANDARD = LVCMOS33;

NET "PHY_LED0_IN" LOC = M15;
NET "PHY_LED1_IN" LOC = N15;
NET "PHY_LED2_IN" LOC = P16;

NET "PHY_LED0_OUT" LOC = AA19;
NET "PHY_LED1_OUT" LOC = Y18;
NET "PHY_LED2_OUT" LOC = AA18;
```

Constraint file for TE0720 when used with TE0701 with 3 LED's inserted into into J5 PMOD

Testing of the LED's

Boot normally and break into u-boot, then use impact or other tool to configure FPGA with the LED Demo .bit file. Now you can configure the PHY LED functions and monitor the effects.

```
zynq-uboot>mii write 0x1A 5 0x0010
```

enable LED2 forwarding to FPGA fabric

```
zynq-uboot>mii write 0 0x16 3
```

select Page 3 for Marvell PHY

```
zynq-uboot>mii write 0 0x10 0x1bbb
```

This will make all 3 LED's to blink

```
zynq-uboot>mii write 0 0x10 0x1888
```

This forces all LED's to OFF state

```
zynq-uboot>mii write 0 0x10 0x1999
```

This forces all LED's to ON state

```
zynq-uboot>mii write 0 0x10 0x1730
```

This configures LED0 as LINK, LED1 as Activity and LED2 as 10Mbit

> LED polarity is programmable, default is low(0)=> LED ON

```
zynq-uboot>mii write 0 0x11 0x4415
```

to change polarity of all LEDs

Design files and ready to use bit file (for TE0701-J5) are available from the download area.

On-board LEDs

Overview: On-board LEDs

There are 3 on-board LEDs, with two of them connected to the System Management Controller and one to the Zynq PL (Done pin).

Name	Color	Connected to:	Default mapping:
LED1	Green	SC	PL MIO[7]
LED2	Red	SC	System Controller Status LED
LED3	Green	Zynq PL	FPGA Done - active low

LED1 GREEN

Is mapped to MIO7 after power up. After the Zynq PS has booted it can change the mapping of this LED. If SC can not enable power to the Zynq then this LED will remain under SC control. It is available to the user only after the power supplies have stabilized and the POR reset to the Zynq is released.

LED2 RED

Is used by the SC as global status LED. The SC can show status information on this LED. Vin power is not required.

LED3 GREEN (FPGA Done)

This green LED is connected to the FPGA Done pin which has an active low state. As soon as the Zynq is powered and the 3.3V I/O voltage is enabled, this LED will illuminate. This indicates that the Zynq PL is not configured. Once the Zynq PL has been configured the LED will go off.

During normal operation when the Zynq PL has been configured, the LED can be controlled from the FPGA fabric. Control of the LED in a user design requires the use of Xilinx startup primitive rather than a normal I/O primitive. If the startup primitive is not used then the LED will go off after configuration and remain off irrespectively of the user design. This LED can not be controlled by the SC.

⚠ This LED will not operate if the SC can not power on the 3.3V output rail that also powers the 3.3V circuitry on the module.

LED Status Codes

LED1	LED2	LED3		Description
OFF	OFF	ON	Fatal error on carrier board	This combination after power up is only possible in no sequencing compatibility mode were 3.3Vout is supplied externally. The 1.0V and 1.8V DC-DC supplies are forced on (NOSEQ=1), and the SC is not able to start (3.3Vin below 2.1V). This should never happen if the external power supplies are OK.
OFF	ON	OFF		

Copyright © 2014 Trenz Electronic GmbH

LED1	LED2	LED3		Description
			VIN missing	3.3Vin is present, but the DC-DC supplies are not powered or 3.3Vin is below 3.05V. If the LEDs stay on in this state then 3.3Vout is not turned on, and the Zynq is kept in the POR state.
OFF	1/2 Blink Fast 4 Hz	ON	OK	Zynq boot from SPI Flash has started.
OFF	1/2 Blink Slow 1 Hz	ON	OK	Zynq boot from SD Card has started.
MIO7 or user controlled function	Blink or user controlled function	OFF	OK	LED3 goes off when the FPGA is configured. NOTE: The FPGA design can control this LED too, so it may remain ON or be flashing when the FPGA is configured.
ON	Slow blink 0.5Hz, 1/8 on, 7/8 off	ON or OFF	Powerdown	EN1 input to the module is low. If sequencing is enabled in this mode, then all power supplies on the module are OFF.
ON	ON	ON	Reset	Powered, RESIN input is active low.

TODO: MEMS

In linux MEMS data can be read using device files

cat /sys/bus/i2c/devices/2-001e/iio\:device1/in_magn_x_raw
current magnetometer X value
cat /sys/bus/i2c/devices/3-0018/iio\:device0/in_accel_x_raw
current accelerometer X value

Copyright © 2014 Trenz Electronic GmbH

RTC

An Intersil temperature compensated real time clock IC ISL1202M is used for timekeeping. Battery voltage must be supplied to the module from the main board.

Battery backed registers are accessed at I2C slave address 1101111x

General purpose RAM is accessed at I2C slave address 1010111x

This RTC IC is supported in Linux so it can be used as hwclock device.

RTC data also can be read using device file

cat /sys/bus/i2c/devices/2-006f/rtc/rtc0/time

Copyright © 2014 Trenz Electronic GmbH

USB

USB HS PHY, USB3320 from Microchip (previously SMSC) is used on the TE0720 module. This is the recommended USB PHY for Zynq devices. SMSC offers a design review service for customer schematic and PCB layouts after registration on their website. Note: This design check service does not list USB PHY devices (only LAN Devices), but the SMSC team will consider and reply to design review requests for USB PHY devices too.

ZYNQ Pins	PHY Pins	B2B Name	Notes
MIO30..39	ULPI	-	Zynq USB0 MIO pins are connected to the PHY
	REFCLK	-	52MHz from MEMS oscillator
	REFSEL[0..2]	-	000 GND, select 52MHz reference Clock
	RESETB	-	Connected to SC
	CLKOUT	-	Connected to 1.8V selects reference clock operation mode
	DP,DM	OTG-D_N, OTG-D_N	USB Data lines
	CPEN	VBUS_V_EN	External USB power switch active high enable signal
	VBUS	USB-VBUS	Connect to USB VBUS via a series resistor. Check reference schematic
	SPK_L	-	Connected to SC
	SPK_R	-	Connected to SC
USB0_OC	-	-	Over current detection from external power switch. If desired it can be muxed to an MIO or FPGA I/O pin via EMIO

The schematic for the USB connector and required components is different depending on the USB usage. USB standard A or B connectors can be used for Host or Device modes. A Mini USB connector can be used for USB Device mode. A USB Micro connector can be used for Device mode, OTG Mode or Host Mode.

> ⚠ The same schematic can not support all possible modes and be fully USB Specification Compliant.

SPI Flash

Programming

TE0720-02 uses Spansion S25FL256S0 Flash that is fully supported by all Xilinx Tools.

TE0720-01 uses Winbond W25Q256 Flash that is supported by Xilinx SDK 2013.4 and 2014.1 (it is not supported by Impact 14.7 or Vivado Programmer 2014).

Copyright © 2014 Trenz Electronic GmbH

Board-level Interfaces

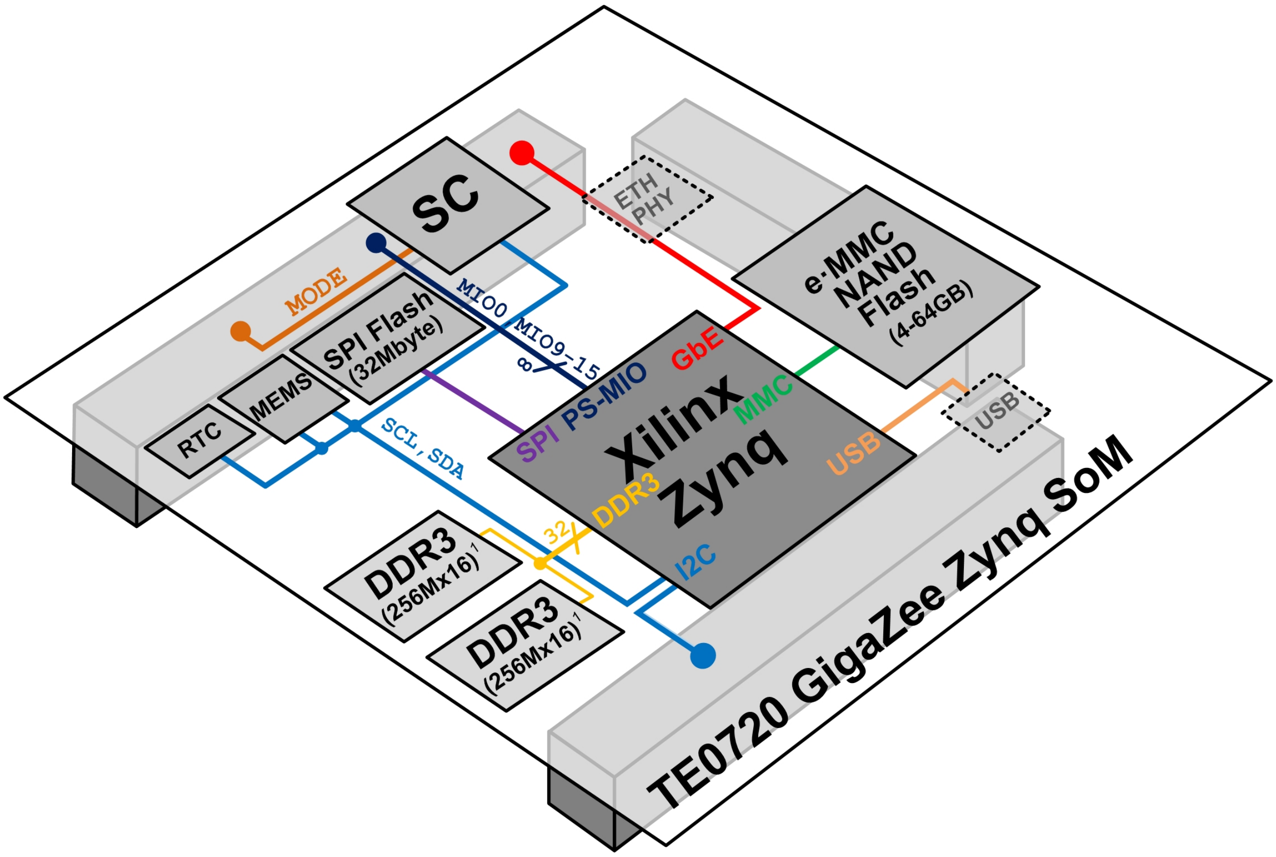


Figure 5: Overview: TE0720 Board-level Interfaces

[1] Note: The DDR3 SDRAM size depends on assembly option.

Copyright © 2014 Trenz Electronic GmbH

Zynq SoM: Multiplexed I/O (MIO) Assignments

Bank	B2B I/O (LVDS Pairs)	VCC I/O Bank	I/O Bank VREF
MIO 0	8	3.3V Fixed	n/a
MIO 1	6	1.8V Fixed	0.9V Fixed
B13	50 (24)	1.2 - 3.3V User Adj.	If a user I/O is used as a Vref pin then the pins available will be reduced.
B33	18 (9)	1.2 - 3.3V User Adj.	User supplied, extra pin
B34	36 (18)	1.5 - 3.3V User Adj.	User supplied, extra pin
B35	48 (24)	1.2 - 3.3V User Adj.	If used need supply to regular I/O pins, reduces I/O count
Total	166 (75)	-	-

MIO Bank 1 pins are used by the SDIO0 core which is required for the SD card boot. A bootable SD card must be connected to these pins.

MIO Bank 1 pins are powered from 1.8V. So if using a normal SD card then an external level shifter is required.

MIO Bank 0 Usage

There are 8 PS MIO pins from Bank 0 available. They can be configured to be connected to different Zynq PS Peripherals.

	MIO0	MIO9	MIO10	MIO11	MIO12	MIO13	MIO14	MIO15	Comment
	GPIO	GPIO	I2C0 SCL	I2C0 SDA	UART1	UART1	UART0	UART0	2x UART + I2C (Recommended configuration)
	GPIO	GPIO	TDI	TDO	TCK	TMS	UART0	UART0	ARM Debug with UART
	GPIO	GPIO	I2C0 SCL	I2C0 SDA	CAN1	CAN1	UART0	UART0	UART + CAN + I2C
	GPIO	GPIO	CAN0	CAN0	UART1	UART1	UART0	UART0	2x UART + CAN
	GPIO	GPIO	CAN0	CAN0	CAN1	CAN1	UART0	UART0	UART + 2x CAN
	GPIO	GPIO	I2C0	I2C0	I2C1	I2C1	UART0	UART0	UART + 2x I2C
	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	8 bit GPIO
	GPIO	GPIO	SDIO1	SDIO1	SDIO1	SDIO1	SDIO1	SDIO1	SD/SDIO (not bootable)
	GPIO	GPIO	SD	SD	SD	GPIO	UART0	UART0	SD/MMC in 1 bit mode with UART
	GPIO	GPIO	SPI1	SPI1	SPI1	SPI	UART0	UART0	SPI and UART
	QSPI1	QSPI	QSPI1	QSPI1	QSPI1	QSPI1	UART0	UART0	Quad SPI and UART

Example MIO configuration for Bank 0 User I/O. When a PS peripheral is not mapped to a MIO pin it can be used via the Zynq FPGA PL Fabric by using the EMIO interface (Quad SPI is not available via EMIO).

Copyright © 2014 Trenz Electronic GmbH

Compatibility with TE07xx series

To be compatible with TE0770 and other TE07xx series modules that have gigabit transceivers, the B34 pins that are dedicated pins on those modules should not be used.

I2C Peripherals

All on-board I2C devices are on a shared bus connected to the System Management Controller. This bus is not available directly to the Zynq PS when the PL is not configured.

It is recommended to map PS I2C1 via EMIO to this I2C bus.

Binary	Hex	HEX >> 1	Device	Notes
1101111x	DE	6F	RTC Registers	
1010111x	A0	57	Battery backup RAM	in RTC IC
0011110x	3C	1E	Magnetometer	MEMS IC
0011000x	30	18	Accelerometer	MEMS IC
0100001x	42	20	System Controller	Emulated I2C GPIO Extender
		21	System Controller	Emulated I2C GPIO Extender
		22	Carrier Controller	on TE0701
		23	Carrier Controller	on TE0701
0111001x	72	39	ADV7511 Registers	on TE0701
0111000x	70	38	ADV7511 Packet Memory	on TE0701, can be changed
0111111x	7E	3F	ADV7511 EDID Memory	on TE0701, can be changed
0111100x	78	3C	ADV7511 CEC Memory	on TE0701, can be changed

I2C Testing with U-Boot

U-Boot can be used as simple I2C test tool.

> *zynq-uboot> i2c probe 0 7f*
>
> *Valid chip addresses: 1E 20 21 22 38 39 3C 3E 3F 57 6F*

Example I2C address scan, RTC, MEMS, ADV7513, System Controller and Carried Controller addresses are detected.

High-Speed I/O

TE0720 module is based on the Zynq 7020 which doesn't have any Gigabit transceivers. However in many cases it is possible to use serial links up to 1.25 GBit/s using FPGA I/O resources. Those serial interfaces can be implemented in any FPGA I/O bank when paying attention to the clocking requirements (clock regions).

SGMII (1.25 Gbit/s) can be implemented with FPGA I/O pins as described in Xilinx XAPP523.

High-speed serial interface ADC converters can be used as described in Xilinx XAPP524.

SGMII (SFP copper or fiber) can be used directly with the Ethernet PHY, as the SGMII pins are available in JM3.

Board-to-Board Connectors

These connectors are hermaphroditic. Odd pin numbers on the module are connected to even pin numbers on the baseboard and vice versa.

GigaZee uses three Samtec Razor Beam LSHM connectors (TOP, BOTTOM and LEFT) on the bottom side.

- Top and Bottom:LSHM-150-04.0-L-DV-A-S-K-TR (100 pins, "50" per row)
- Left: LSHM-130-04.0-L-DV-A-S-K-TR (60 pins, "30" per row)

When using the same type on baseboard the mating height is 8mm. Other mating heights are possible by using connectors with a different height:

Connector on baseboard	Mating height
LSHM-1x0-02.5-L-DV-A-S-K-TR	6.5 mm
LSHM-1x0-03.0-L-DV-A-S-K-TR	7.0 mm
LSHM-1x0-04.0-L-DV-A-S-K-TR	8.0 mm
LSHM-1x0-06.0-L-DV-A-S-K-TR	10.0mm

Other connectors can be assembled on the module on request.

The LSHM connector speed rating depends on the stacking height; please see the following table:

Stacking height	Speed rating
12 mm, Single-Ended	7.5 GHz / 15 Gbps
12 mm, Differential	6.5 GHz / 13 Gbps
5 mm, Single-Ended	11.5 GHz / 23 Gbps
5 mm, Differential	7.9 GHz / 14 Gbps

Mechanical Ratings:

- Shock: 100G, 6 ms Sine
- Vibration: 7.5G random, 3 hours 3 axis

Manufacturer Documentation:

Name	Version	Date
LSHM-1XX-XX.X-X-DV-A-X-X-TR-FOOTPRINT(1).pdf	1	2013-11-28 16:54
LSHM-1XX-XX.X-XX-DV-A-X-X-TR-MKT.pdf	1	2013-11-28 16:56
TC0923--2523_report_Rev_2_qua.pdf	1	2013-11-28 16:55

Name	Version	Date
hsc-report_lshm-lshm-05mm_web.pdf	1	2013-11-28 16:56
lshm_dv.pdf	1	2013-11-28 16:56
tc0929--2611_qua(1).pdf	1	2013-11-28 16:55

Pinout

Connector left

Edit Document

Pin	Net	Type	Bank	FPGA
1	SOUT_N	SGMII_TX		
3	SOUT_P	SGMII_TX		
5	GND			
7	B34_L7_P	DIFFIO	34	J18
9	B34_L7_N	DIFFIO	34	K18
11	GND			
13	B34_L2_P	DIFFIO	34	J16
15	B34_L2_N	DIFFIO	34	J17
17	GND			
19	B34_L4_P	DIFFIO	34	L17
21	B34_L4_N	DIFFIO	34	M17
23	GND			
25	B34_L5_P	DIFFIO	34	N17
27	B34_L5_N	DIFFIO	34	N18
29	GND			
31	B34_L12_P	DIFFIO_CC	34	L18
33	B34_L12_N	DIFFIO_CC	34	L19
35	GND			
37	B34_L8_P	DIFFIO	34	J21
39	B34_L8_N	DIFFIO	34	J22
41	B34_L9_P	DIFFIO	34	J20
43	B34_L9_N	DIFFIO	34	K21
45	GND			
47	OTG D+	DIFFIO		
49	OTG D-	DIFFIO		
51	OTG ID	I		
53	VBUS_V_EN	O		
55	USB_VBUS	I		
57	B34_L22_P	DIFFIO	34	R19

Copyright © 2014 Trenz Electronic GmbH

59	B34_L22_N	DIFFIO	34	T19
2	SIN_N	SGMII_RX		
4	SIN_P	SGMII_RX		
6	GND			
8	B34_L1_P	DIFFIO	34	J15
10	B34_L1_N	DIFFIO	34	K15
12	GND			
14	B34_L18_P	DIFFIO	34	P20
16	B34_L18_N	DIFFIO	34	P21
18	GND			
20	B34_L20_P	DIFFIO_CC	34	P17
22	B34_L20_N	DIFFIO_CC	34	P18
24	GND			
26	B34_L10_P	DIFFIO	34	L21
28	B34_L10_N	DIFFIO	34	L22
30	GND			
32	B34_L13_P	DIFFIO_CC	34	M19
34	B34_L13_N	DIFFIO_CC	34	M20
36	GND			
38	B34_L21_P	DIFFIO	34	T16
40	B34_L21_N	DIFFIO	34	T17
42	B34_L15_P	DIFFIO	34	M21
44	B34_L15_N	DIFFIO	34	M22
46	GND			
48	B34_L17_P	DIFFIO	34	R20
50	B34_L17_N	DIFFIO	34	R21
52	B34_L23_P	DIFFIO	34	R18
54	B34_L23_N	DIFFIO	34	T18
56	B34_VREF	IO_VREF	34	M16/P15
58	B34_L14_P	DIFFIO_CC	34	N19
60	B34_L14_N	DIFFIO_CC	34	N20

Connector top

Edit Document

Pin	Net	Type	Bank	FPGA	
1	VCCIO34	I/O Supply	34		
3	VCCIO34	I/O Supply	34		
5	VCCIO33	I/O Supply	33		
7	VCCIO13	I/O Supply	13		
9	VCCIO13	I/O Supply	13		
11	B33_L7_P	DIFFIO	33	AA22	
13	B33_L7_N	DIFFIO	33	AB22	
15	B33_L8_P	DIFFIO	33	AA21	
17	B33_L8_N	DIFFIO	33	AB21	
19	1.5V	O			
21	B33_L11_P	DIFFIO_CC	33	Y19	
23	B33_L11_N	DIFFIO_CC	33	AA19	
25	B33_L12_P	DIFFIO_CC	33	Y18	
27	B33_L12_N	DIFFIO_CC	33	AA18	
29	B33_VREF	IO_VREF	33	V19/V15	
31	B33_L17_P	DIFFIO	33	AA17	
33	B33_L17_N	DIFFIO	33	AB17	
35	B33_L18_P	DIFFIO	33	AA16	
37	B33_L18_N	DIFFIO	33	AB16	
39	GND				
41	B13_L7_P	DIFFIO	13	AA12	
43	B13_L7_N	DIFFIO	13	AB12	
45	B13_L8_P	DIFFIO	13	AA11	
47	B13_L8_N	DIFFIO	13	AB11	
49	GND				
51	B13_L11_P	DIFFIO_CC	13	AA9	
53	B13_L11_N	DIFFIO_CC	13	AA8	
55	B13_L9_P	DIFFIO	13	AB10	
57	B13_L9_N	DIFFIO	13	AB9	
59	GND				
61	B13_L20_P	DIFFIO	13	T4	
63	B13_L20_N	DIFFIO	13	U4	
65	B13_L17_P	DIFFIO	13	AB7	

67	B13_L17_N	DIFFIO	13	AB6	
69	GND				
71	B13_L16_P	DIFFIO	13	AB5	
73	B13_L16_N	DIFFIO	13	AB4	
75	B13_L18_P	DIFFIO	13	Y4	
77	B13_L18_N	DIFFIO	13	AA4	
79	GND				
81	B13_L15_P	DIFFIO	13	AB2	
83	B13_L15_N	DIFFIO	13	AB1	
85	B13_L21_P	DIFFIO	13	V5	
87	B13_L21_N	DIFFIO	13	V4	
89	B13_IO25	IO	13	U7	
91	VREF_JTAG	O 3.3V			
93	TMS	JTAG	0		
95	TDI	JTAG	0		
97	TDO	JTAG	0		
99	TCK	JTAG	0		
2	VIN	Power input			
4	VIN	Power input			
6	VIN	Power input			
8	VIN	Power input			
10	3.3V	O			
12	3.3V	O			
14	B33_L4_P	DIFFIO	33	W20	
16	B33_L4_N	DIFFIO	33	W21	
18	RESIN	Reset input			
20	GND				
22	B33_L13_P	DIFFIO_CC	33	W17	
24	B33_L13_N	DIFFIO_CC	33	W18	
26	B33_L14_P	DIFFIO_CC	33	W16	
28	B33_L14_N	DIFFIO_CC	33	Y16	
30	GND				
32	B13_L5_P	DIFFIO	13	U12	
34	B13_L5_N	DIFFIO	13	U11	

36	B13_L6_P	DIFFIO	13	U10	
38	B13_L6_N	DIFFIO	13	U9	
40	GND				
42	B13_L1_P	DIFFIO	13	V10	
44	B13_L1_N	DIFFIO	13	V9	
46	B13_L12_P	DIFFIO_CC	13	Y9	
48	B13_L12_N	DIFFIO_CC	13	Y8	
50	GND				
52	B13_L14_P	DIFFIO_CC	13	AA7	
54	B13_L14_N	DIFFIO_CC	13	AA6	
56	B13_L13_P	DIFFIO_CC	13	Y6	
58	B13_L13_N	DIFFIO_CC	13	Y5	
60	GND				
62	B13_L4_P	DIFFIO	13	V12	
64	B13_L4_N	DIFFIO	13	W12	
66	B13_L3_P	DIFFIO	13	W11	
68	B13_L3_N	DIFFIO	13	W10	
70	GND				
72	B13_L10_P	DIFFIO	13	Y11	
74	B13_L10_N	DIFFIO	13	Y10	
76	B13_L2_P	DIFFIO	13	V8	
78	B13_L2_N	DIFFIO	13	W8	
80	GND				
82	B13_L23_P	DIFFIO	13	V7	
84	B13_L23_N	DIFFIO	13	W7	
86	B13_L24_P	DIFFIO	13	W6	
88	B13_L24_N	DIFFIO	13	W5	
90	GND				
92	B13_L19_P	DIFFIO	13	R6	
94	B13_L19_N	DIFFIO	13	T6	
96	B13_L22_P	DIFFIO	13	U6	
98	B13_L22_N	DIFFIO	13	U5	
100	B13_IO0	IO	13	R7	

Copyright © 2014 Trenz Electronic GmbH

Connector bottom

Edit Document

Pin	Net	Type	Bank	FPGA
1	VIN	Power input		
3	VIN	Power input		
5	VIN	Power input		
7	NOSEQ	input		
9	VCCIO35	I/O Supply		
11	VCCIO35	I/O Supply		
13	3.3VIN	Power input		
15	3.3VIN	Power input		
17	MIO45	MIO	501	B9
19	MIO44	MIO	501	E13
21	MIO43	MIO	501	B11
23	MIO42	MIO	501	D8
25	MIO41	MIO	501	C8
27	MIO40	MIO	501	E14
29	GND			
31	B35_L16_N	DIFFIO	35	C22
33	B35_L16_P	DIFFIO	35	D22
35	B35_L24_N	DIFFIO_ADC	35	G22
37	B35_L24_P	DIFFIO_ADC	35	H22
39	1.8V	O		
41	B35_L18_N	DIFFIO_ADC	35	B22
43	B35_L18_P	DIFFIO_ADC	35	B21
45	B35_L15_N	DIFFIO_ADC	35	A22
47	B35_L15_P	DIFFIO_ADC	35	A21
49	B35_L22_N	DIFFIO_ADC	35	G21
51	B35_L22_P	DIFFIO_ADC	35	G20
53	GND			
55	B35_L17_N	DIFFIO_ADC	35	D21
57	B35_L17_P	DIFFIO_ADC	35	E21
59	B35_L13_N	DIFFIO_CC	35	B20
61	B35_L13_P	DIFFIO_CC	35	B19

Copyright © 2014 Trenz Electronic GmbH

63	GND			
65	B35_L14_N	DIFFIO_CC	35	C20
67	B35_L14_P	DIFFIO_CC	35	D20
69	B35_L4_N	DIFFIO	35	G16
71	B35_L4_P	DIFFIO	35	G15
73	GND			
75	B35_L12_N	DIFFIO_CC	35	C19
77	B35_L12_P	DIFFIO_CC	35	D18
79	VBAT	VBAT input		
81	B35_L20_N	DIFFIO_ADC	35	F19
83	B35_L20_P	DIFFIO_ADC	35	G19
85	MIO15	MIO	500	E6
87	MIO0	MIO	500	G6
89	GND			
91	MIO9	MIO	500	C4
93	MIO11	MIO	500	B4
95	MIO10	MIO	500	G7
97	MIO13	MIO	500	A6
99	MIO12	MIO	500	C5
2	GND			
4	PHY_MDI0_P	DIFFIO		
6	PHY_MDI0_N	DIFFIO		
8	GND			
10	PHY_MDI1_P	DIFFIO		
12	PHY_MDI1_N	DIFFIO		
14	NC			
16	PHY_MDI2_P	DIFFIO		
18	PHY_MDI2_N	DIFFIO		
20	GND			
22	PHY_MDI3_P	DIFFIO		
24	PHY_MDI3_N	DIFFIO		
26	GND			
28	EN1	I		
30	PGOOD	O		

Copyright © 2014 Trenz Electronic GmbH

32	MODE	IO		
34	GND			
36	B35_L10_N	DIFFIO_ADC	35	A19
38	B35_L10_P	DIFFIO_ADC	35	A18
40	B35_L9_N	DIFFIO_ADC	35	A17
42	B35_L9_P	DIFFIO_ADC	35	A16
44	GND			
46	B35_L7_N	DIFFIO_ADC	35	B15
48	B35_L7_P	DIFFIO_ADC	35	C15
50	B35_L2_N	DIFFIO_ADC	35	D17
52	B35_L2_P	DIFFIO_ADC	35	D16
54	GND			
56	B35_L8_N	DIFFIO_ADC	35	B17
58	B35_L8_P	DIFFIO_ADC	35	B16
60	B35_L21_N	DIFFIO_ADC	35	E20
62	B35_L21_P	DIFFIO_ADC	35	E19
64	GND			
66	B35_L11_N	DIFFIO_CC	35	C18
68	B35_L11_P	DIFFIO_CC	35	C17
70	B35_L23_N	DIFFIO	35	F22
72	B35_L23_P	DIFFIO	35	F21
74	GND			
76	B35_L5_N	DIFFIO_ADC	35	E18
78	B35_L5_P	DIFFIO_ADC	35	F18
80	B35_L3_N	DIFFIO_ADC	35	D15
82	B35_L3_P	DIFFIO_ADC	35	E15
84	GND			
86	B35_L6_N	DIFFIO	35	F17
88	B35_L6_P	DIFFIO	35	G17
90	GND			
92	MIO14	MIO	500	B6
94	B35_L1_N	DIFFIO_ADC	35	E16
96	B35_L1_P	DIFFIO_ADC	35	F16
98	B35_L19_N	DIFFIO	35	H20

Copyright © 2014 Trenz Electronic GmbH

100	B35_L19_P	DIFFIO	35	H19

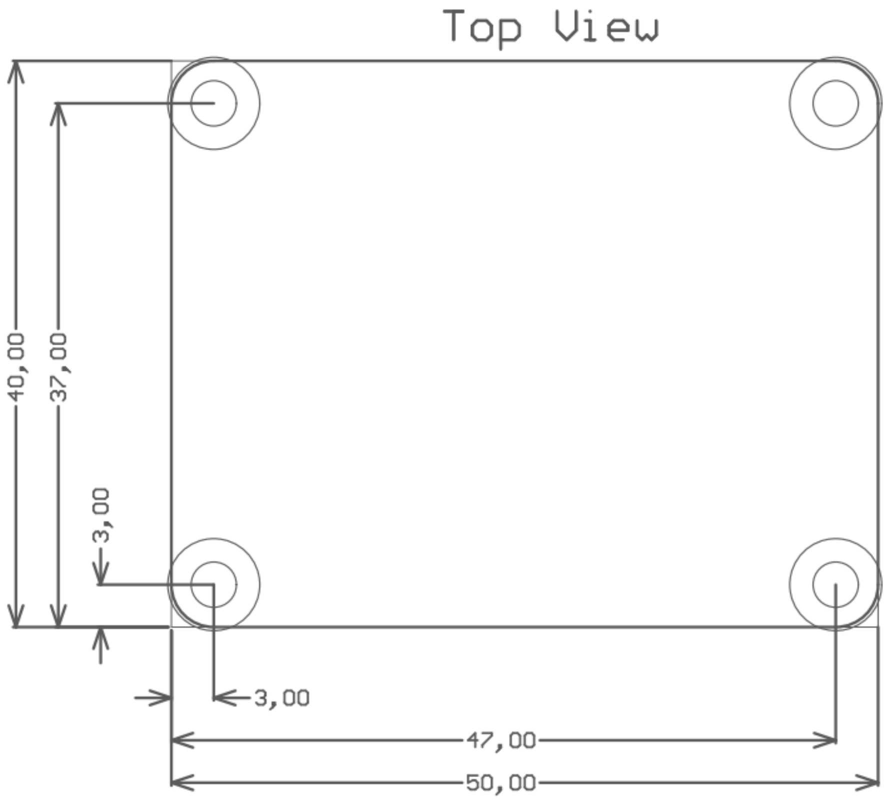
Technical Specifications

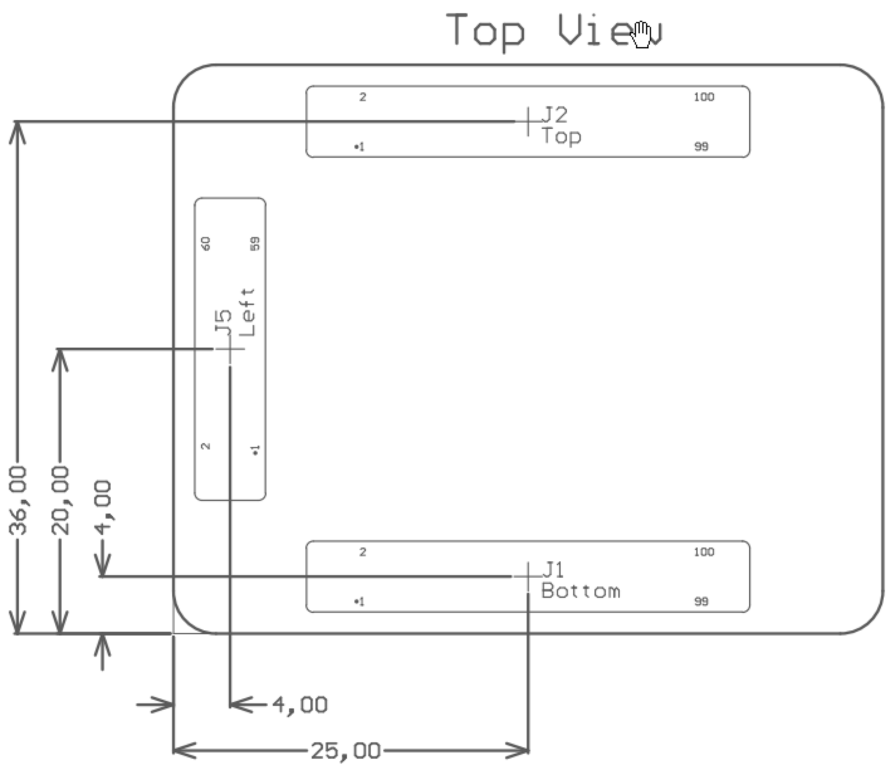
TE0720 Board Dimensions & Attributes

Dimensions

- Module size: 50 mm × 40 mm
- Mating height with standard connectors: 8mm
- PCB thinkness: 1.6mm
- highest part on PCB: approx. 2.5mm. Please download the step model for more exact numbers.

All dimensions are shown in mm.





B2B numbers when looking from top onto carrier board will have odd and even numbers swapped.

Power Supplies

Input	Voltage range +/-10%	Connector current rating
Vin	3.3 V ÷ 5.5 V	max. 8 A
Vin 3.3V	3.3 V	max. 2 A

Vin and Vin 3.3V can be connected to the same source (3.3 V).

Temperature Ranges

Commercial grade modules	0 °C ÷ +70 °C
Industrial grade modules	-40 °C ÷ +85 °C

Depending on the customer design, additional cooling might be required.

Weight

16.2 g	without bolts
22.4	with bolts screwed to the module

TE0720 Schematic

The schematic is available for download here: TE0720 (GigaZee) Schematic

Carrier Boards for TE0720

TE0701 Carrier Board

The documentation on the TE0701 Carrier Board can be found in the TE0701 Carrier Board User Manual. Moreover, in the following sections is described how the TE0701 Carrier Board can be customized by the Zynq FPGA via the onboard I2C bus and how the interfacing of the TE0701 peripherals is accomplished from the TE0720's point of view.

Configuring FMC Power Supply Voltage on TE0701 via I2C (CPLD Firmware Rev 0.1)

The FMC power supply on the TE0701 Carrier Board (i.e., FMC_VADJ) is user programmable via I2C. More precisely, the three output voltage select lines VS0 to VS2 of the Enpirion EN5335QI DC-DC converter with 3-pin programmable voltage output are mapped to the CPLD's "I2C-to-GPIO Port Expander" (VID0=GPIO_output[4] => VS0, VID1=GPIO_output[5] => VS1, VID2=GPIO_output[6] => VS2) 8-bit control register, which can be programmed via the dedicated board-to-board I2C bus (HDMI_SCL, HDMI_SDA) on the I2C slave address 0x22:

Bit	Mapping	Description for Output/Write	Default
0	PHY_LED2	Enable(=1) / disable(=0) yellow LED of PHY on TE0701	0b0
1		reserved	0b0
2		reserved	0b0
3	PG_C2M	signal to FMC connector	0b0
4	VID0	= VS0	0b0
5	VID1	= VS1	0b0
6	VID2	= VS2	0b0
7	EN_FMC	Enable(=1) / disable(=0) FMC_VADJ voltage	0b0

Table 2: Pin assignments of the 8-bit "I2C-to-GPIO Port Expander" Control Register

VID [2:0]	FMC_VADJ Value
0 (000)	3.3V
1 (001)	2.5V
2 (010)	1.8V
3 (011)	1.5V
4 (100)	1.25V
5 (101)	1.2V
6 (110)	0.8V

Copyright © 2014 Trenz Electronic GmbH

VID [2:0]	FMC_VADJ Value
7 (111)	reserved

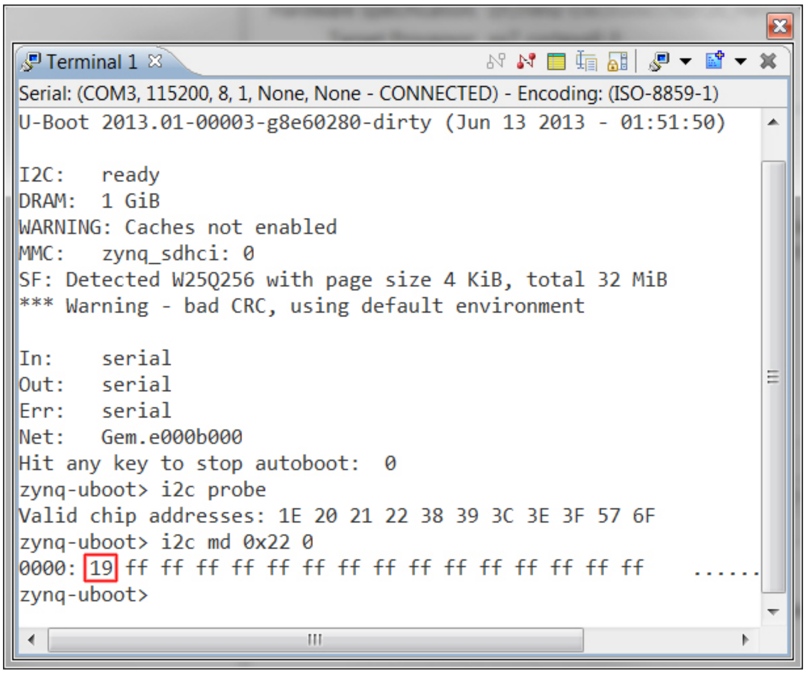
Table 3: VID is the digitally programmable value for the FMC_VADJ

Example: To enable (EN_FMC=1) and set FMC_VADJ to 3.3V (VID[2:0]=000) write 0x80 to I2C address 0x22.

The most significant bit of the 8-bit GPIO register (see Table 2) is directly routed to the enable input of the Enpirion EN5335QI DC-DC converter, which is disabled by default.

Reading I2C-to-GPIO Status Register on TE0701 CPLD (CPLD Firmware Rev 0.1)

The CPLD's 8-bit "I2C-to-GPIO Port Expander" status register can be read via I2C bus (HDMI_SCL, HDMI_SDA) on the I2C slave address 0x22:



In U-Boot a simple I2C test tool can be used (see TE0720 User Manual | I2C Addresses for more details):

- The I2C command "i2c command" can be used to show all devices on the I2C bus.
- To read, e.g., register 0 from I2C device on address 0x22, the command "i2c md 0x22 0" is used. **Note**: Only one 8-bit status register is available on the TE0720 CPLD

Example: Corresponding to the following mapping of the TE0701 CPLD's status register, the returned register content (0x19=0b00011001) can be interpreted as follows (from LSb upwards): no FMC module inserted (FMC_RSNT#=1), power ok signal due to disabled FMC power supply off (POK_FMC=0), SD card is inserted (SD_DETECT#=0), SD card write protection is enabled (SD_WP=1) and the output load on the USB port has not been exceeded (USB_OC#=1).

Bit	Mapping	Description for Output/Write
0	FMC_RSNT#	FMC module inserted? yes=0, no=1
1	POK_FMC	"Power ok" signal from the Enpirion EN5335QI DC-DC converter (see TE0720 User Manual | Carrier Boards for TE0720 for more details)
2	SD_DETECT#	SD Card inserted? yes=0, no=1
3	SD_WP	Write protection on SD Card enabled? yes=1, no=0
4	USB_OC#	Over current (OC) output of the TPS2051 that limits the output current of the Micro USB port and is pulled low, when the output load exceeds the current-limit threshold (see TE0701 Carrier Board User Manual | Configuring Power Supply of the Micro USB Connector for more details).
5		reserved
6		reserved
7		reserved

HDMI Interface of TE0720 on TE0701 Carrier Board

	Zynq FPGA I/O Pins	Notes
HDMI_CLK	N20	
HDMI_DE	N19	
HDMI_VS	T19	
HDMI_HS	R19	
HDMI_D0	T18	
HDMI_D1	R18	
HDMI_D2	R21	
HDMI_D3	R20	
HDMI_D4	M22	
HDMI_D5	K21	
HDMI_D6	M21	
HDMI_D7	J20	
HDMI_D8	T17	
HDMI_D9	J22	
HDMI_D10	T16	
HDMI_D11	J21	
SDA	W21	
SCL	W20	
Interrupt	AA17	
CEC_CLK	AB16	FPGA should emit some suitable clock on this pin if CEC feature is needed
CT_HPD	AB17	Drive high for normal operation
LS_OE	AA16	Drive high for normal operation

TE0720 with TE0603 Carrier

TE0603 was not designed for the TE07xx series, so many new functions are not available. TE0720 will be in "no power sequencing mode" when inserted into a TE0603 baseboard. For proper operation VCCIO must be 3.3V and supplied by the TE0603. To enable this place a jumper to short pins 1 and 2 in pin header J2.

TE0603 before Revision -03 please remove R13 or TE0720 will not boot at all. TE0603-03 do not need a fix.

Normal boot procedure; all LEDs are on as long as reset is active. At reset deactivation; green LED2 goes off and very quickly after that green LED3 goes off indicating that the FSBL has loaded the FPGA bitstream (DONE=1). Red LED1 is blinking fast; this is status indication that QSPI boot mode is selected.

Functions available with TE0603

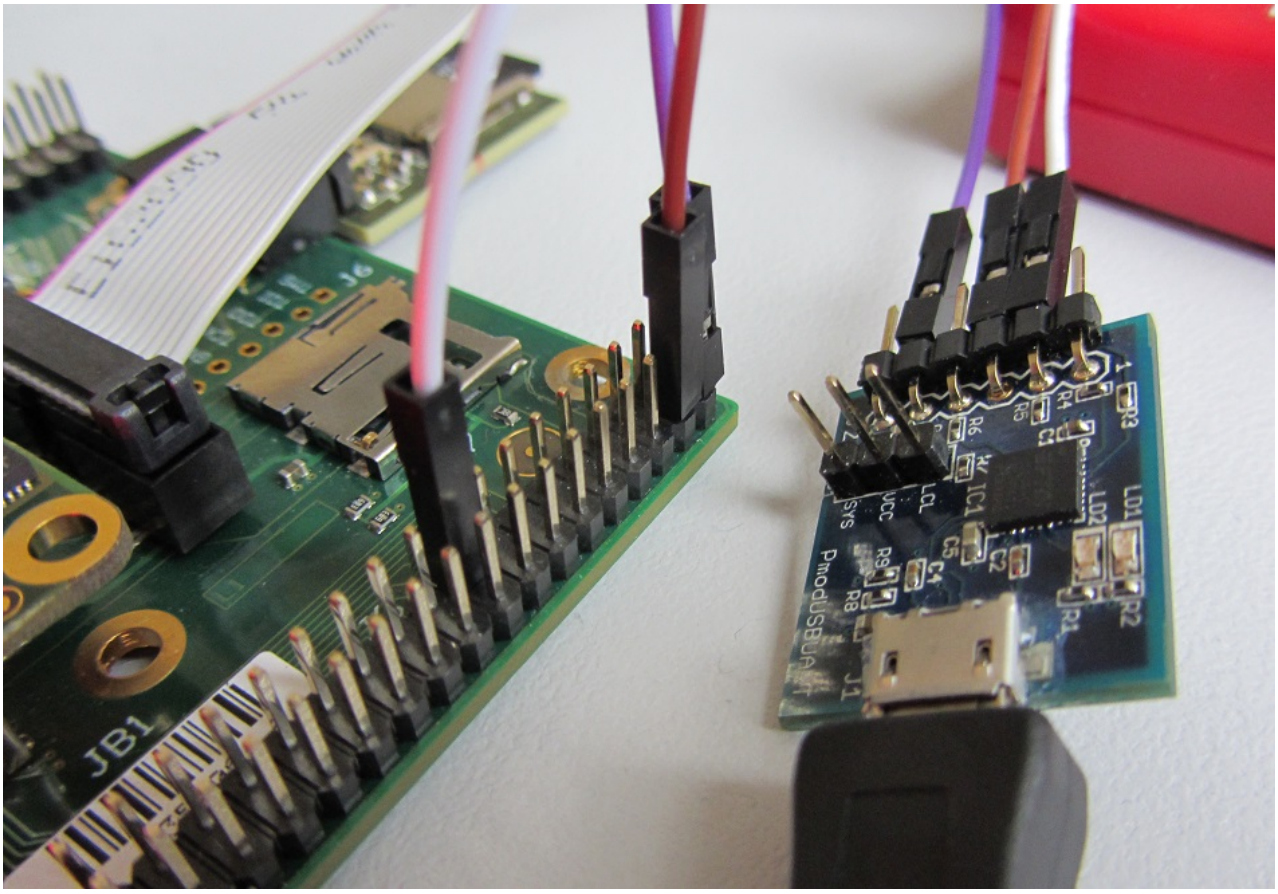
TE0720 Function	TE0603	Description and notes
USB	no	Missing 3rd connector on TE0603
ETH	yes, RJ45	There are extra 50 ohm resistors not needed for TE0720
JTAG	Yes, J5 or J6	With external JTAG Adapter
SD Boot	no	Missing pins and SD card level shifter
UART0 MIO14,15	yes	With external logic level USB UART adapter, pins J3 Pin 37 and 24 - default bootloader and Linux console
-	microSD	Not connected to usable pins on the TE0720
SDIO	J11	Possible via EMIO only with PMOD-SD adapter not connected to MIO pins
MIO0 pins	pin header	yes, J3
FPGA PL I/O	LEDs	EMIO or FPGA controlled
MIO1 pins	yes, J4	4 pins available in J14 (1.8V VCCIO), 2 pins connected to RJ45 LEDs

Ethernet will not work in 1000M mode with long brand X cable. Use either a short good quality cable or remove 8 termination resistors for the ETH PHY on the TE0603. Please be aware that TE0600 modules require those terminations, so this modification is only for TE0720 usage.

This video shows how LED1 and LED2 can be connected (using the System Controller) to the Ethernet PHY, and how the indicated status changes when swapping from a "bad" cable to a "good" cable.

UART Console

TE0720 standard flash image uses UART0 for console, baudrate 115200, mapped to MIO pins 14, 15. Those pins are available at J3 pins 24, 37.



Digilent PmodUSBUART connected to TE0603 for MIO14,MIO15.

Carrier Board Checklist

Schematic Checklist

1	Are B2B pin numbers on the connectors mirrored compared to the module pin numbers?	As B2B connectors are "unisex" type the do mirror pin numbers when connecting. That is pin1 connects to pin2, and pin2 to pin1, etc.
2	Are B2B connectors named JB1, JB2, JB3?	This is not a hard requirement, but it helps to use the same identifiers.
3	Are all GND pins connected to a common ground net?	
4	Are all VIN pins connected together?	
5	Is JB2 pin 92 pin used as VREF for the JTAG interface?	for future compatibility only, currently all modules have 3.3V JTAG
6	Are external circuits/buffers connecting to MIO bank 1 pins powered from JB1 pin 40?	JB1 pins 18, 20, 22, 24, 26, 28 use voltage at pin 40 as VCCIO. Currently it is 1.8V for all released modules.

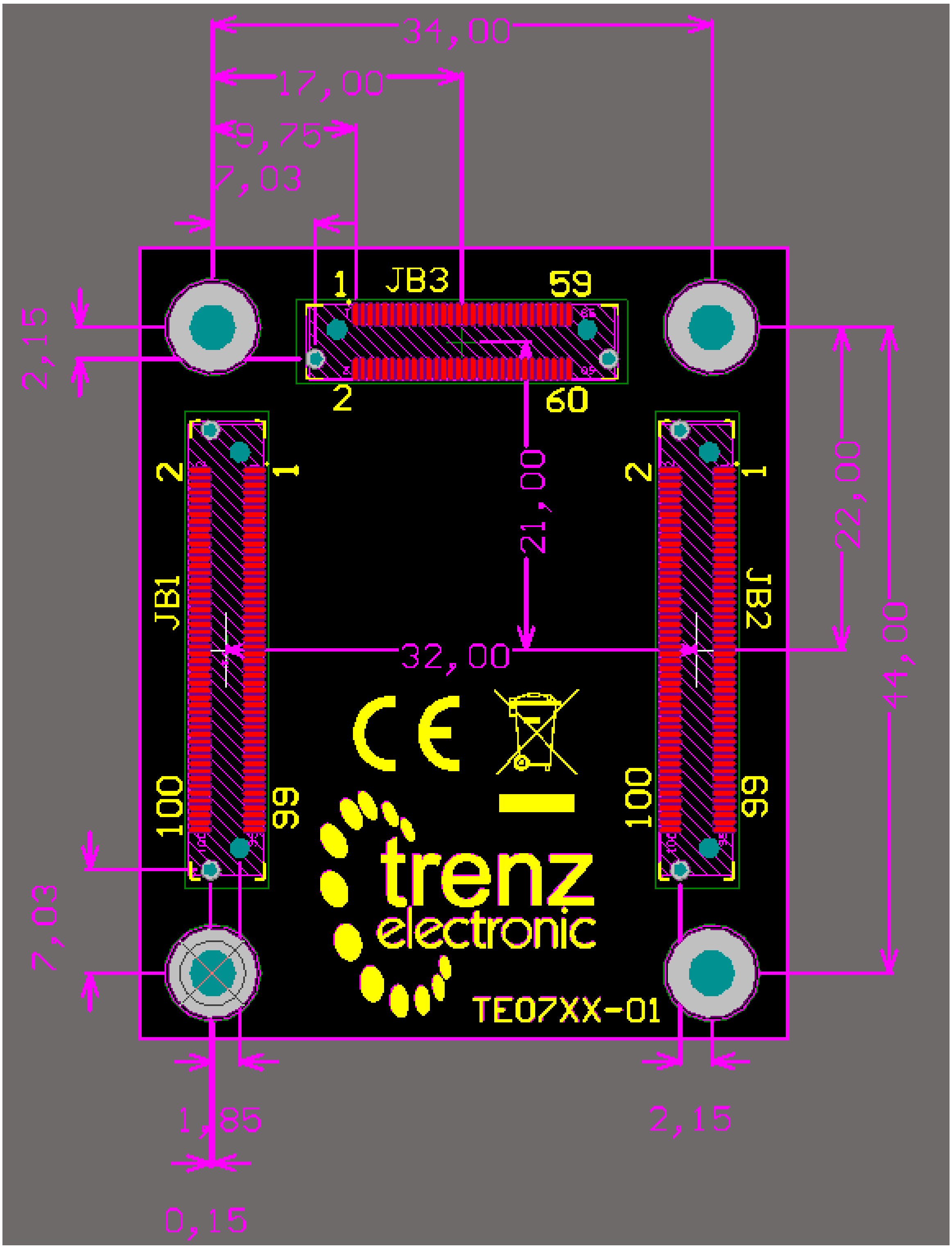
PCB Checklist

1	Are mounting holes placed properly?	Four Mounting holes should always be used. They are required for mounting screws and for module extraction. The mounting holes will also help in dissipating some heat from the module to the carried board PCB. Four holes with a 3.2mm diameter should be placed exactly at the corners of a 37mm by 47mm rectangle.
2	Are B2B headers properly placed?	B2B headers must be placed and aligned very precisely or the module will not align correctly (in the worst case module insertion could destroy the connectors or the PCB). The B2B headers should be locked on the PCB, and it is recommended that the position and placement be checked against placement dimensions before submitting the PCB files.
3	Are B2B headers rotated properly?	As B2B header pin numbers differ from module to the carrier (swap of odd and even numbers), it is recommended that that the rotation is checked in the PCB design.
4	Height clearance below module	Components can be placed below the module but height clearance rules must be obeyed.
5	Power dissipation of components below module	It is not recommended to place any components with high power dissipation below the module, as there will be almost no airflow below the module.

Visual Check of Module placement

It is highly recommended to use the Base board Template designs as a starting point for new PCB designs. If that is not possible, then adding linear dimensions in the design helps to check that all connectors and mounting holes are properly placed.

⚠ This placement is same for all 4x5 Modules!



Top view of the Carrier Board.

Connector numbers as on base! (pin JB1.1 on base would mate to pin JM1.2 on module).

Reference Projects

Binary files of the reference projects can be downloaded from the Trenz-Electronic download area.

> ⚠ Full Board Support Packages for Vivado CAN NOT BE CREATED AT THIS TIME. We have filed a WebCase (990721) and Xilinx has Answer Record AR58180 regarding this. The feature to create board support packages will hopefully be available with some later Vivado releases, but currently it is not known in which version. At least it is not confirmed for 2014.1 for sure.

Working with Reference Projects

DEPRECATED - use Vivado/petalinux 2014.2 or newer

Boot Sequence

Reference projects follow the standard Zynq boot sequence:

Step	File	Name	Description
0	-	BootROM	Zynq boot ROM start up. No user access.
Base system project files			
1	system.bit	Bitstream	PL bitstream
2	FSBL.elf	FSBL	Standard Zynq First Stage Bootloader
U-Boot project files			
3	u-boot.elf	u-boot	U-Boot Second Stage Bootloader
Base system project files			
4	boot.bin, boot.mcs	boot file	Zynq boot image
Linux project files			
5	uImage	Linux kernel	Linux kernel 3.9 wrapped to be used with U-Boot
	devicetree.dtb	Device Tree	Linux device tree blob (binary large object)
	uramdisk.image.gz	Ramdisk	Linux ramdisk image wrapped to be used with U-Boot

The Zynq boot ROM firmware

- reads boot mode settings,
- performs basic configuration,
- downloads and boots FSBL from the selected boot source.

The TE0720 module supports booting from the on-board QSPI Flash memory and from an external SD memory card.

Files from steps 1, 2 and 3 are used to create boot.bin or boot.elf images, which are used to initialize the QSPI Flash memory or an SD memory card.

Linux ramdisk image contain base linux system files with network services (ssh, http). Default IP: 192.168.42.50 SSH Password: "1234"

Copyright © 2014 Trenz Electronic GmbH

Projects Build

To build a high level project (like a Linux kernel), several steps should be performed:

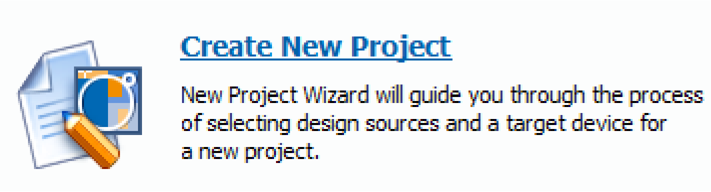
1. Build a base PlanAhead project
2. Build an FSBL - First Stage Boot Loader
3. Create an environment on Ubuntu or CentOS Linux to build U-Boot and Linux kernel
4. Build U-Boot
5. Build a Linux kernel, a device tree blob and a ramdisk

Base PlanAhead Project

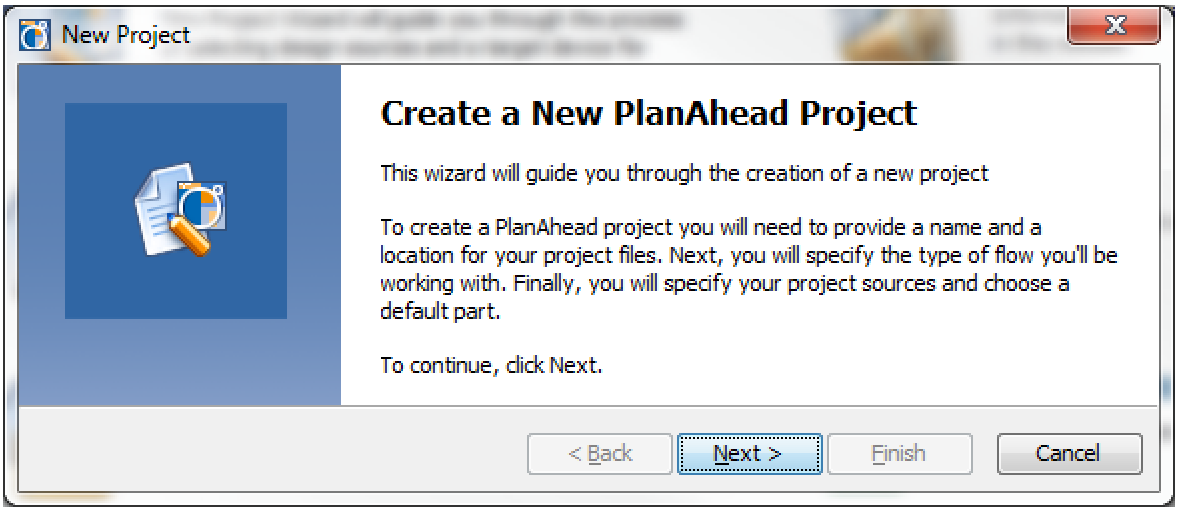
⚠ DEPRECATED - use Vivado 2014.2 or newer and start with Board Part Interface flow

Run Xilinx PlanAhead 14.5.

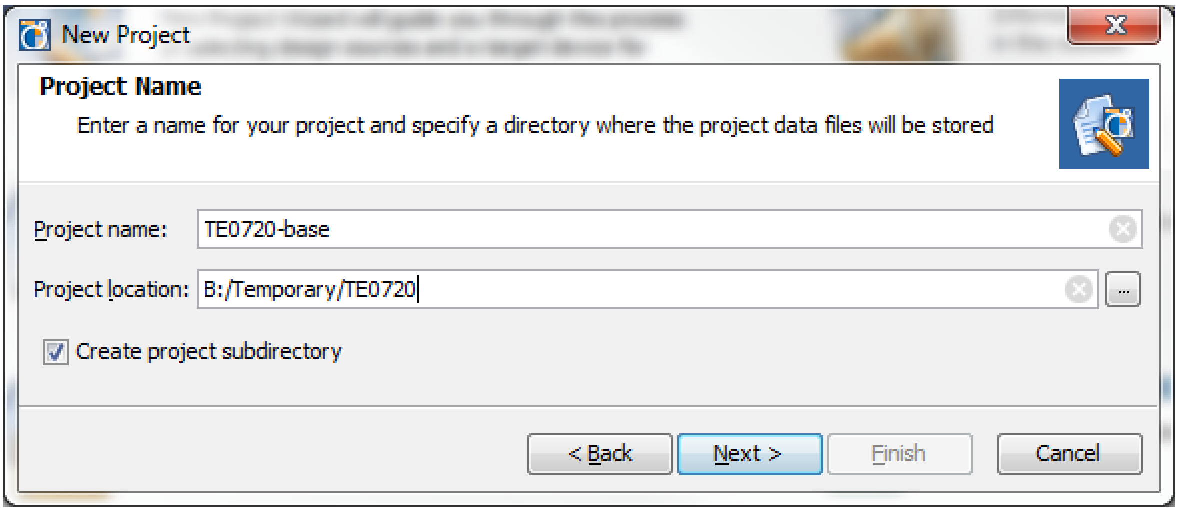
Click "Create New Project".



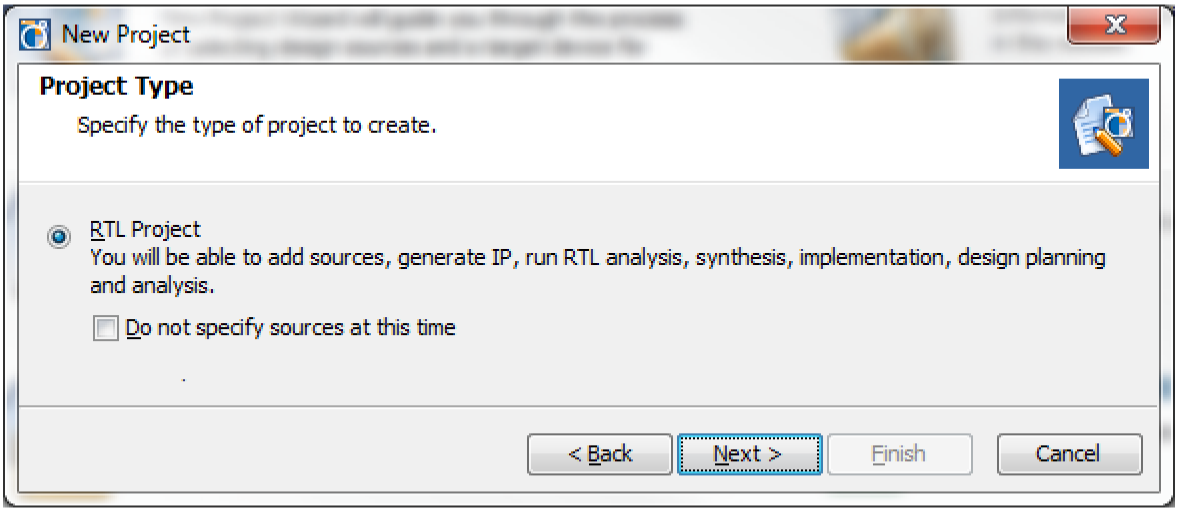
Click "Next" to continue.



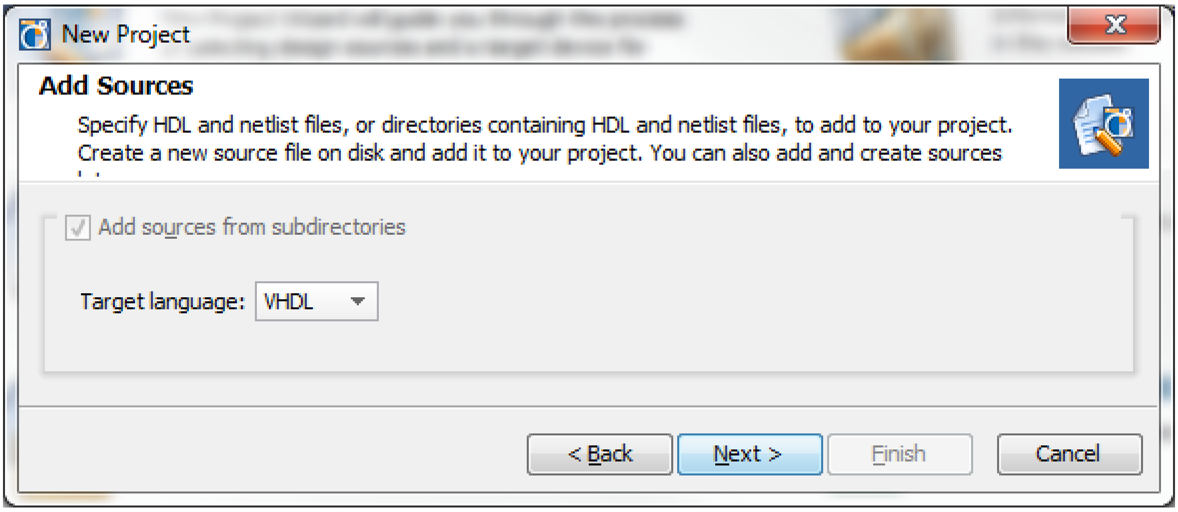
Select the project location and click "Next".



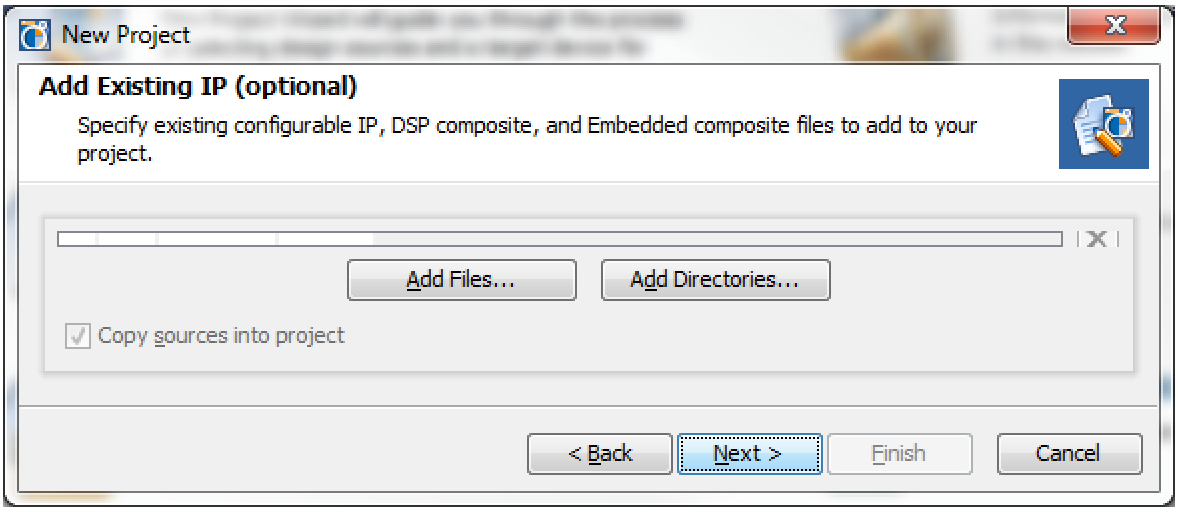
Click "Next".



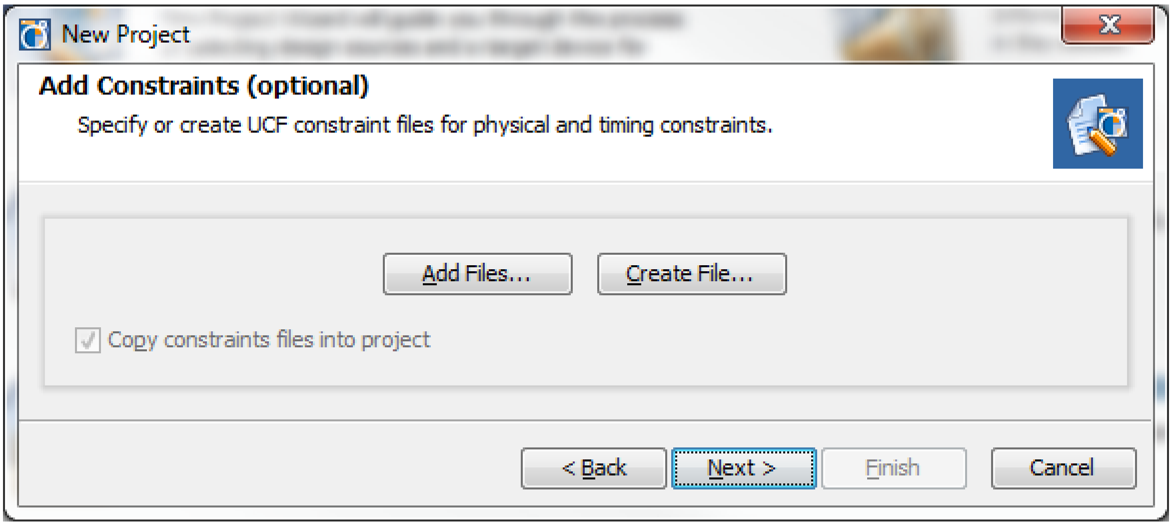
Select VHDL as a target language and click "Next".



Click "Next".

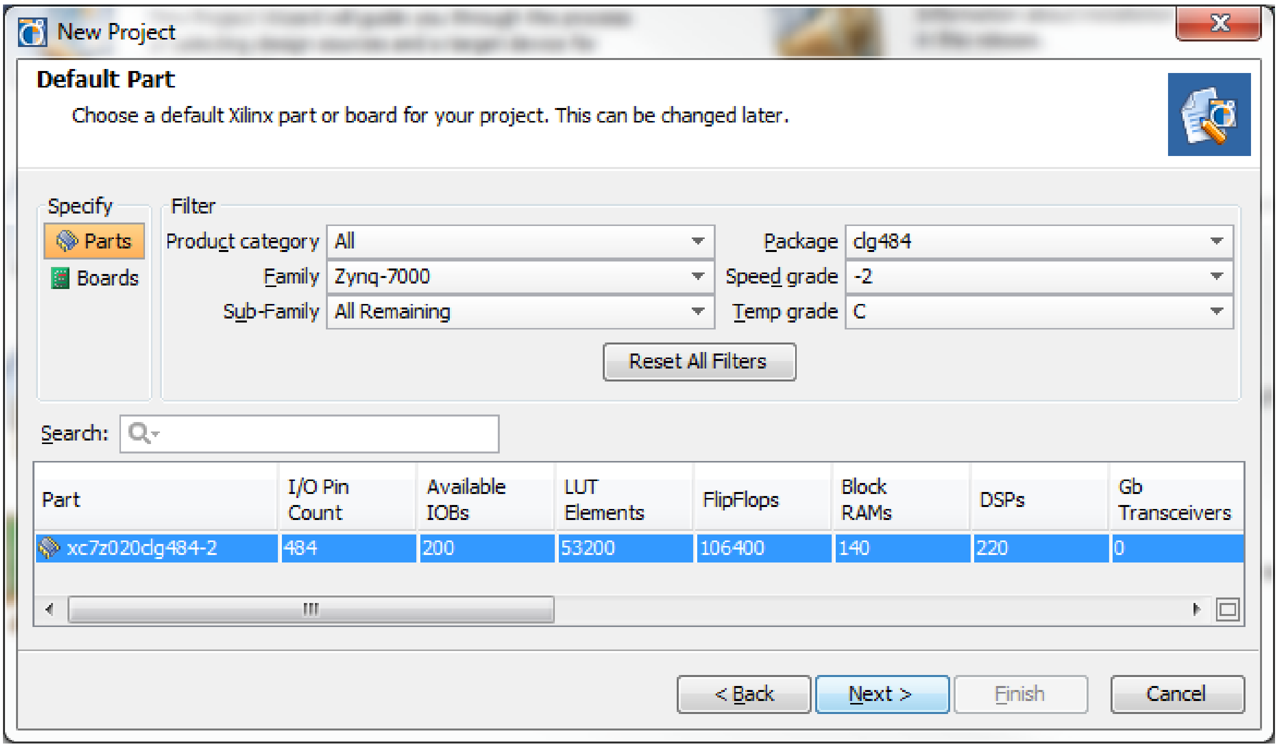


Click "Next".

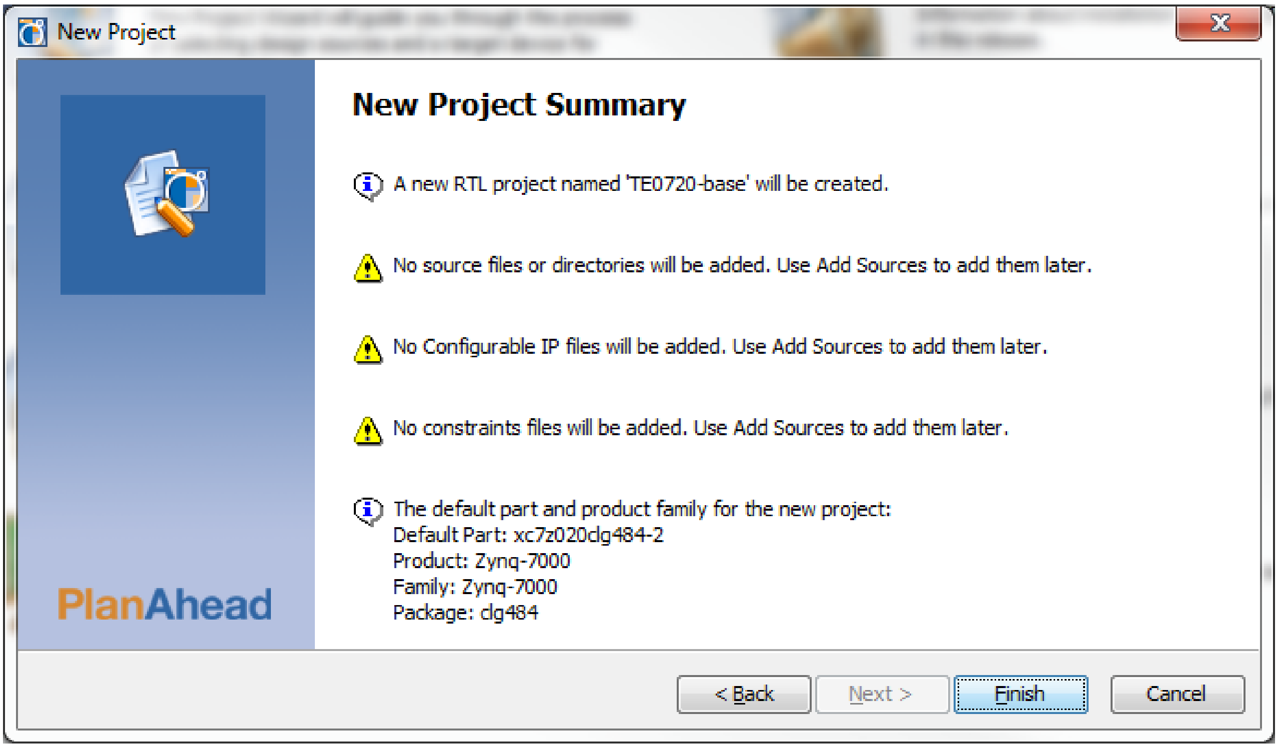


Select your chip using the following filter settings:

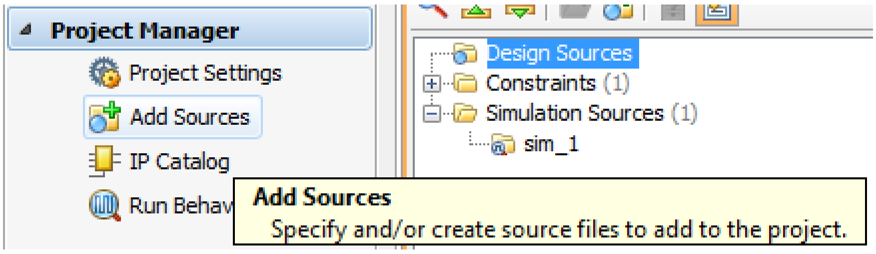
- Family: Zynq-7000
- Package: clg484



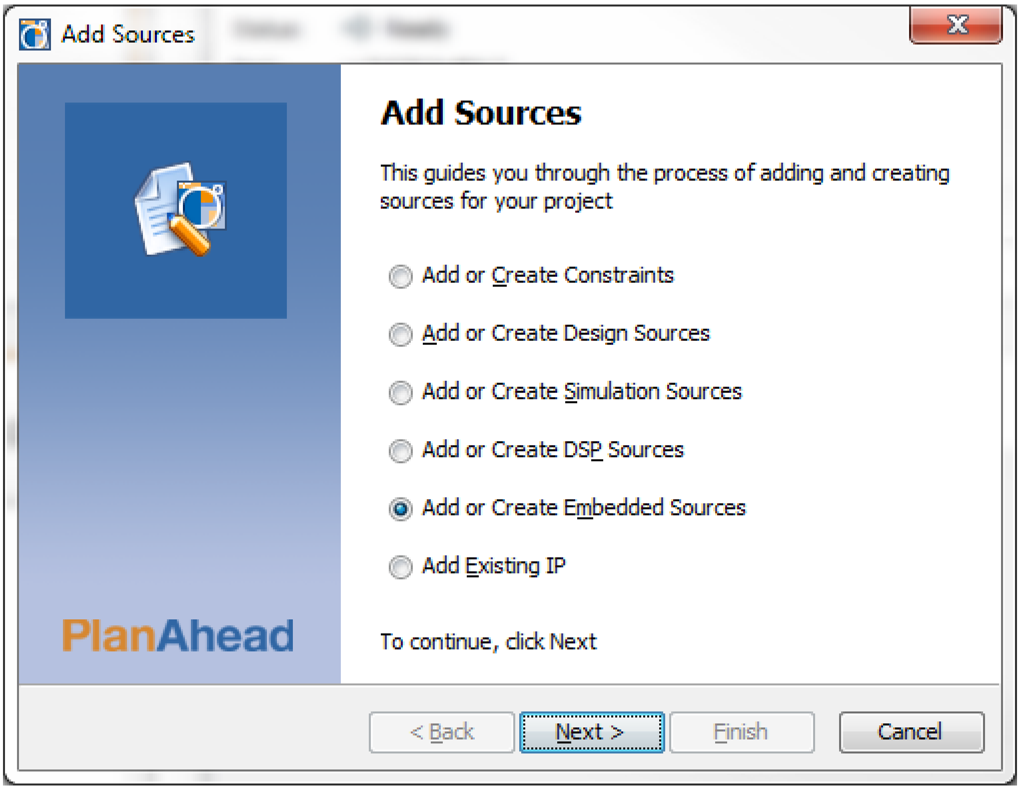
Click "Finish" to create the project.



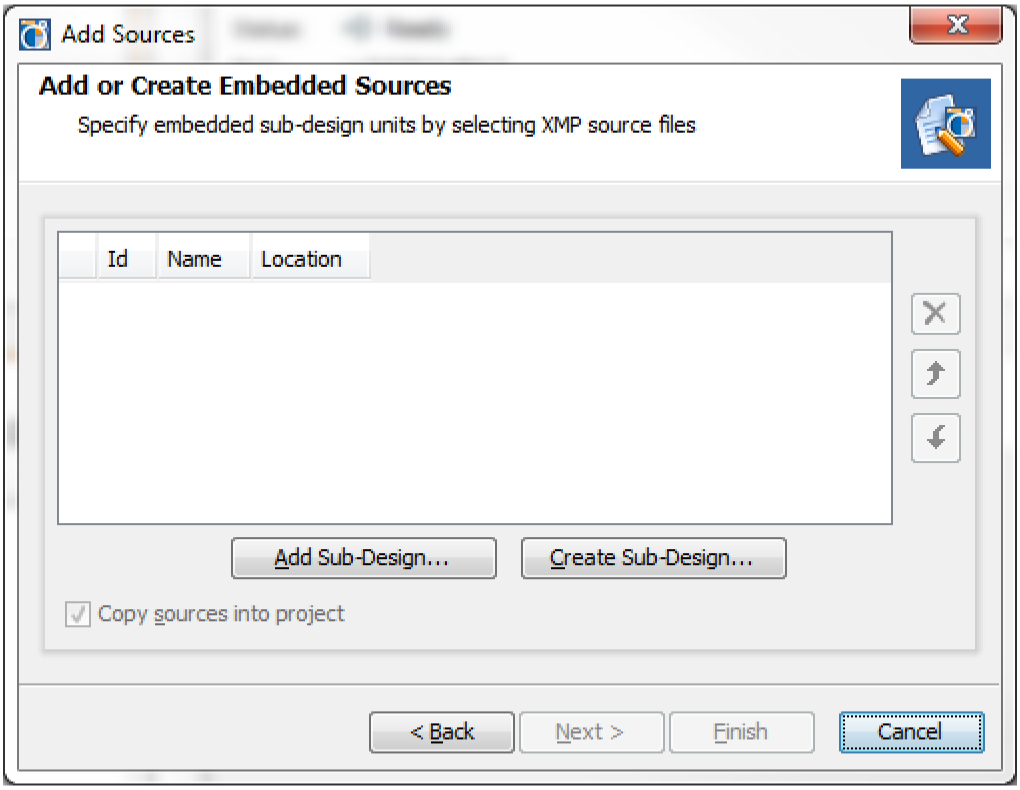
Click "Add Sources" in the "Project Manager" tab.



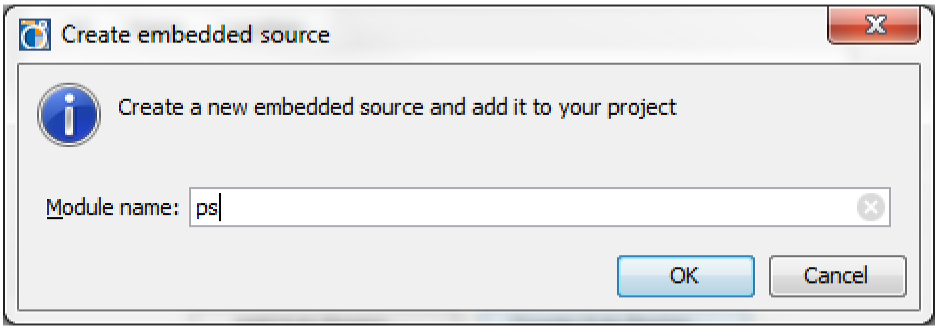
Select "Add or Create Embedded Sources" and then click "Next".



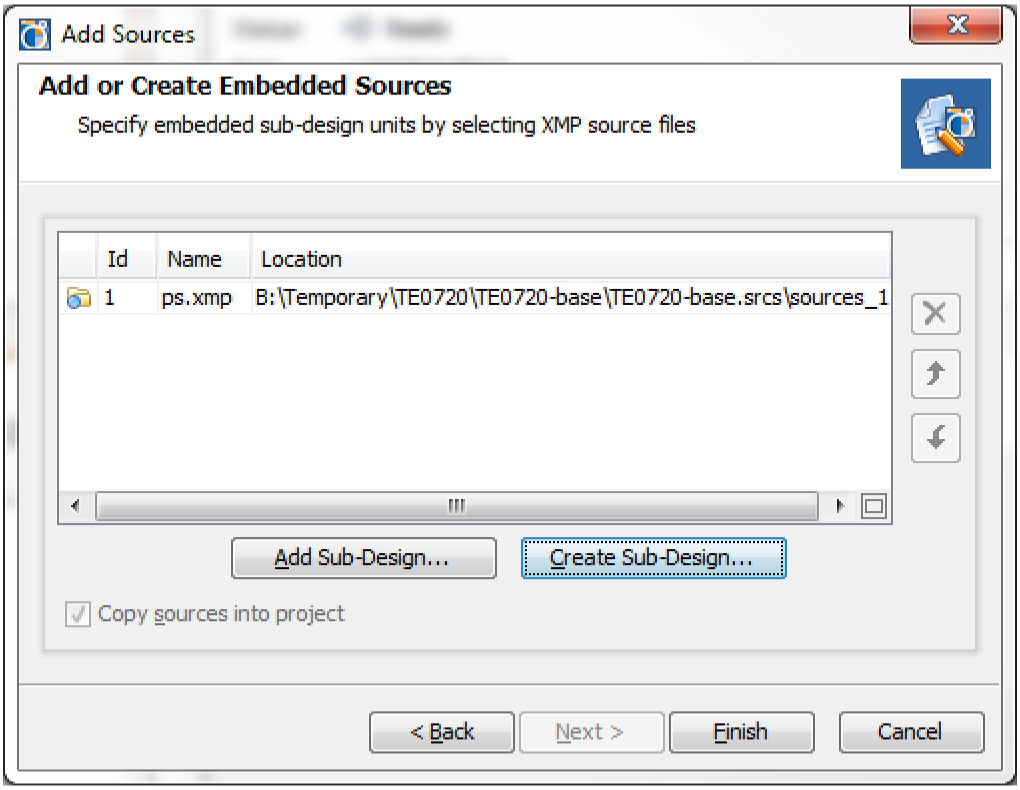
Click "Create Sub-Design..."



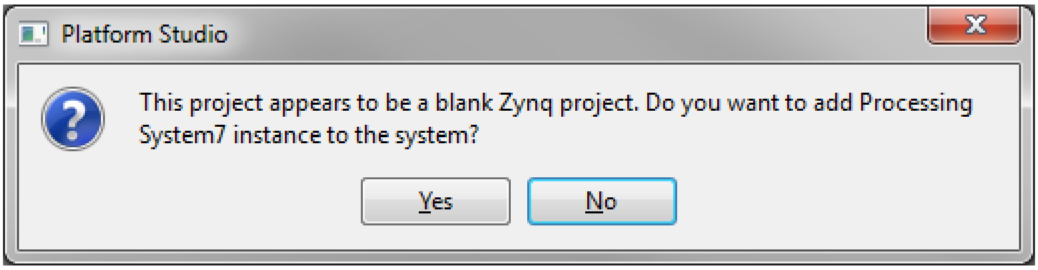
Enter a "Module name" for the Processing System and click "OK".



Click "Finish".



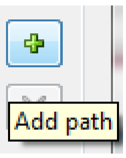
Click "Yes" to add Processing System7 instance to the system.



Download TE0720-01_a.xml from the Trenz Electronic Download Area.

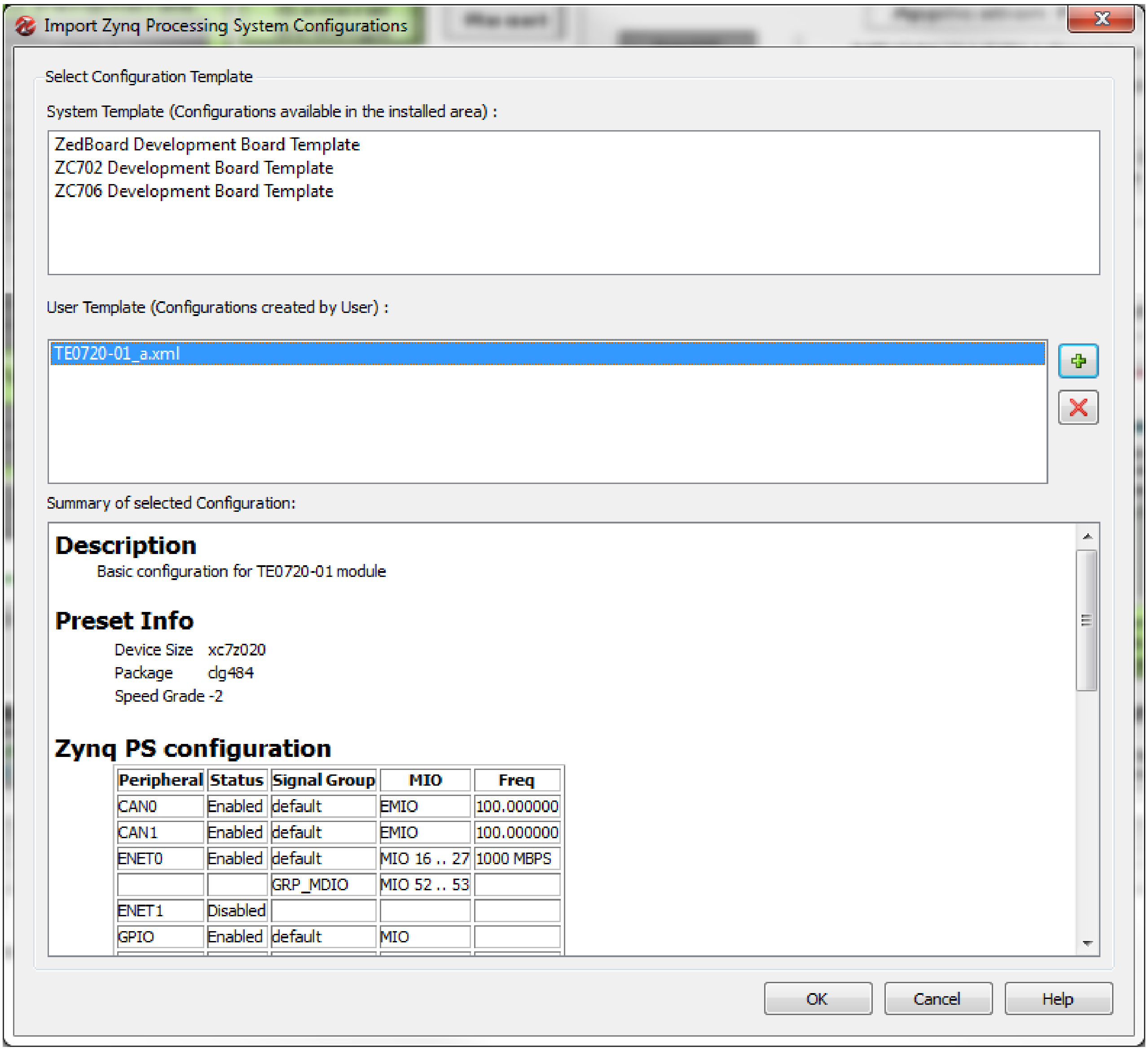
Click "Import" in the Xilinx Platform Studio window.

Click "+" to add a path to the downloaded xml file.

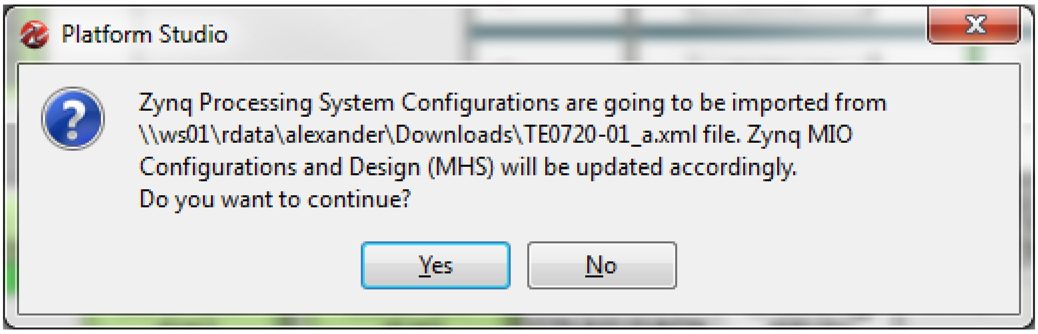


Select TE0720-01_a.xml from the list and click "OK".

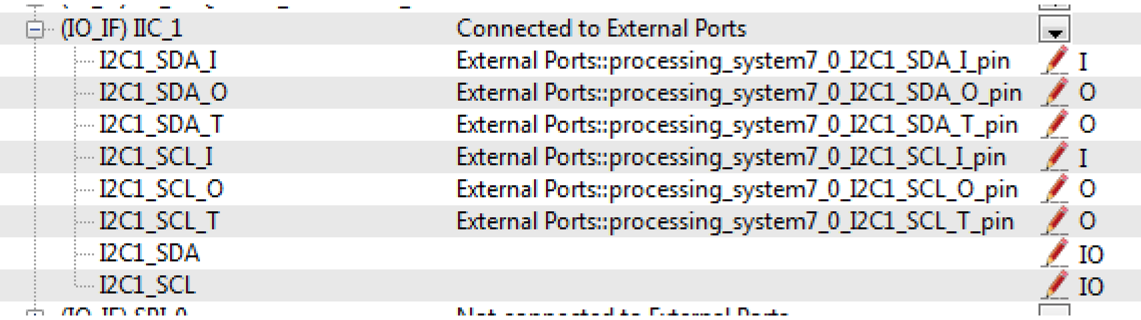
Copyright © 2014 Trenz Electronic GmbH



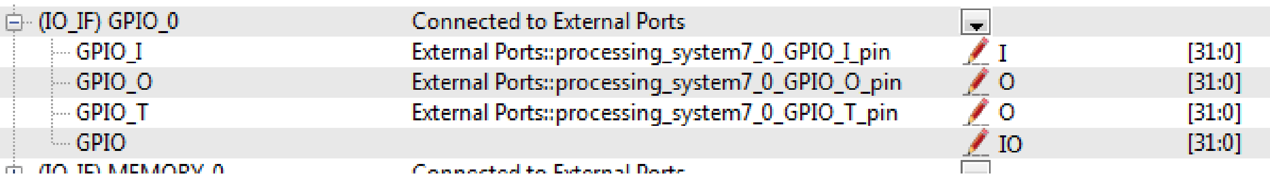
Click "Yes".



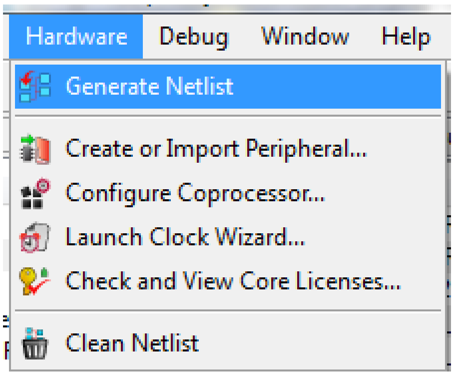
Select the "Ports" tab in "System Assembly View". Open *processing_system7_0->IIC_1* and configure I2C1_SDA_I, I2C1_SDA_O, I2C1_SDA_T, I2C1_SCL_I, I2C1_SCL_O and I2C1_SCL_T as "External Ports".



Select the "Ports" tab in "System Assembly View". Open *processing_system7_0->GPIO_0* and configure GPIO_I, GPIO_O and GPIO_T as "External Ports".



Run **Hardware -> Generate Netlist** from the main menu.

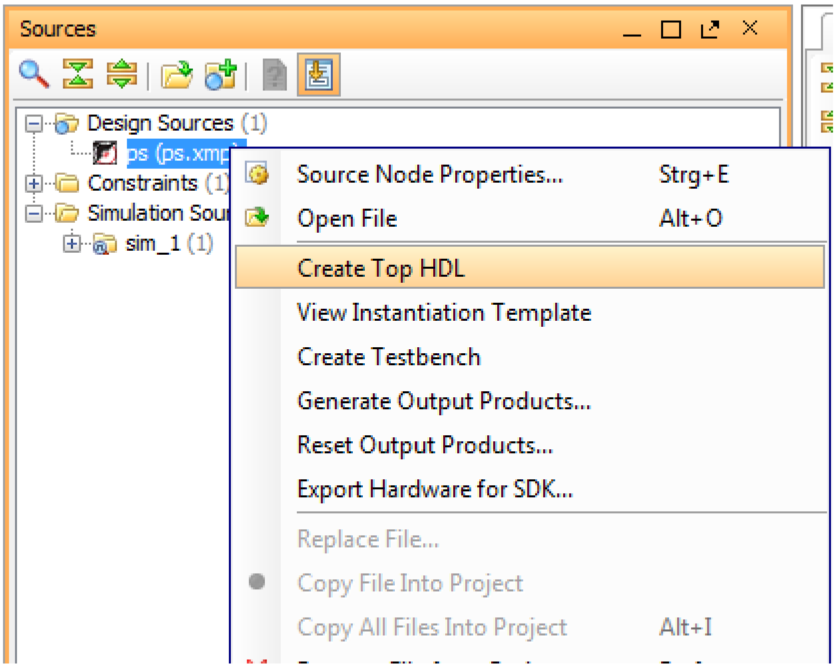


Wait for the synthesis to complete and close "Xilinx Platform Studio".

Starting from this point, we are going to create a custom PS wrapper which communicates with the PS via the EMIO interface and pass I²C signals and one GPIO pin to the on-board CPLD chip.

Inside the CPLD,an I²C switch controlled by a GPIO pin is implemented.

Right-click on the created PS (ps.xmp) file and select "Create Top HDL".



Open the created "ps_stub.vhd" and make the following changes; alternatively, you can replace the created *ps_stub.vhd* with that contained in the TE0720-01-Base.zip archive from the Trenz Electronic Download Area.

Comment external I2C1 and GPIO signals and add CPLD signals to the ps_stub entity.

> *-- processing_system7_0_I2C1_SDA_I_pin : in std_logic;*
>
> *-- processing_system7_0_I2C1_SDA_O_pin : out std_logic;*
>
> *-- processing_system7_0_I2C1_SDA_T_pin : out std_logic;*
>
> *-- processing_system7_0_I2C1_SCL_I_pin : in std_logic;*
>
> *-- processing_system7_0_I2C1_SCL_O_pin : out std_logic;*
>
> *-- processing_system7_0_I2C1_SCL_T_pin : out std_logic;*
>
> *-- processing_system7_0_GPIO_I_pin : in std_logic_vector(31 downto 0);*
>
> *-- processing_system7_0_GPIO_O_pin : out std_logic_vector(31 downto 0);*
>
> *-- processing_system7_0_GPIO_T_pin : out std_logic_vector(31 downto 0)*
>
> *-- I2C - CPLD connection*
>
> *X5 : in STD_LOGIC; -- i2c_sda_in*
>
> *X7 : out STD_LOGIC; -- i2c_sda_out*
>
> *X0 : out STD_LOGIC; -- i2c_sw*
>
> *X1 : out STD_LOGIC; -- i2c_scl_out*

```
-- I2C TE0701
B33_L4_P : inout STD_LOGIC; -- TE0701 SCL
B33_L4_N : inout STD_LOGIC -- TE0701 SDA
);
end ps_stub;
```

Add signals to the architecture section.

```
attribute BOX_TYPE : STRING;
attribute BOX_TYPE of ps : component is "user_black_box";
signal ps_i2c_sda_i : STD_LOGIC;
signal ps_i2c_sda_o : STD_LOGIC;
signal ps_i2c_sda_t : STD_LOGIC;
signal ps_i2c_scl_i : STD_LOGIC;
signal ps_i2c_scl_o : STD_LOGIC;
signal ps_i2c_scl_t : STD_LOGIC;
signal gpio_i : STD_LOGIC_VECTOR(31 downto 0);
signal gpio_o : STD_LOGIC_VECTOR(31 downto 0);
signal gpio_t : STD_LOGIC_VECTOR(31 downto 0);
begin
```

Edit the PS mapping.

```
processing_system7_0_I2C1_SDA_I_pin => ps_i2c_sda_i,
processing_system7_0_I2C1_SDA_O_pin => ps_i2c_sda_o,
processing_system7_0_I2C1_SDA_T_pin => ps_i2c_sda_t,
processing_system7_0_I2C1_SCL_I_pin => ps_i2c_scl_i,
processing_system7_0_I2C1_SCL_O_pin => ps_i2c_scl_o,
processing_system7_0_I2C1_SCL_T_pin => ps_i2c_sda_t,
processing_system7_0_GPIO_I_pin => gpio_i,
processing_system7_0_GPIO_O_pin => gpio_o,
```

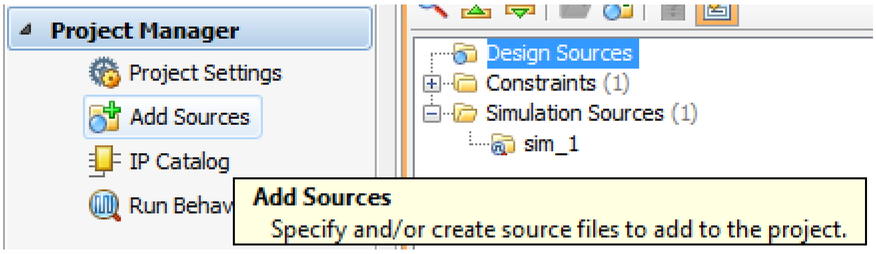
> *processing_system7_0_GPIO_T_pin => gpio_t*
>
> *);*

Add "glue" logic to the architecture section.

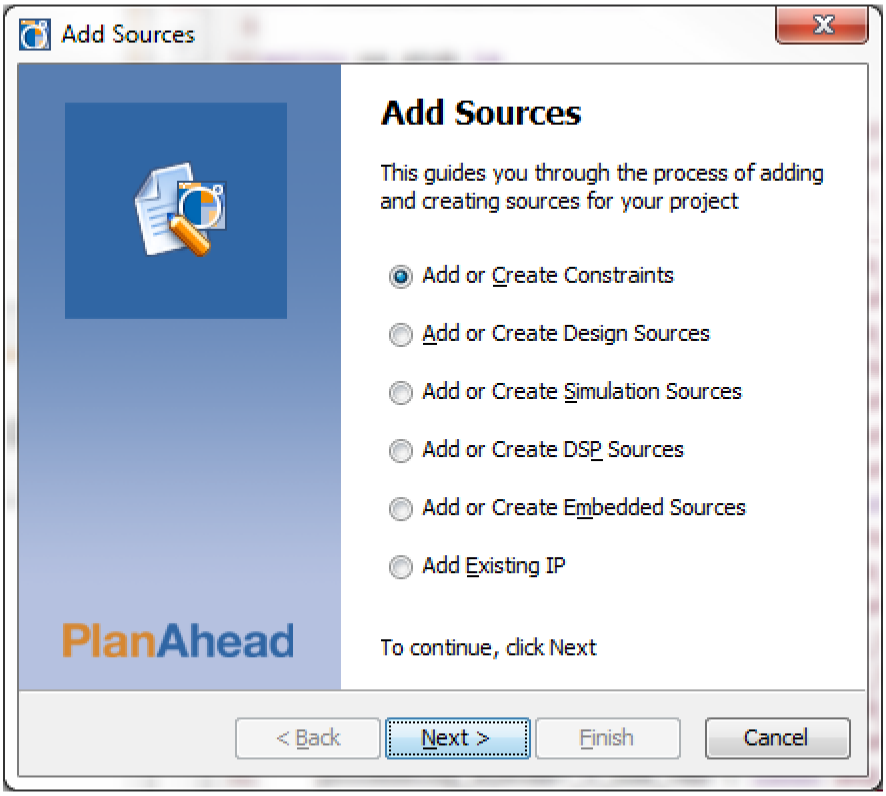
> *gpio_i <= x"12345678"; -- Stub for GPIO input signals*
>
> *X0 <= not gpio_o(0); -- I2C bus switch control signal*
>
> *ps_i2c_sda_i <= X5 and B33_L4_N; -- SDA in*
>
> *ps_i2c_scl_i <= ps_i2c_scl_o or ps_i2c_scl_t; -- SCL feedback*
>
> *X7 <= ps_i2c_sda_o or ps_i2c_sda_t; -- CPLD SDA*
>
> *B33_L4_N <= ps_i2c_sda_o when (ps_i2c_sda_t = '0') else 'Z'; -- TE0701 SDA*
>
> *X1 <= ps_i2c_scl_o or ps_i2c_scl_t; -- CPLD SCL*
>
> *B33_L4_P <= ps_i2c_scl_o or ps_i2c_scl_t; -- TE0701 SCL*

Save the file.

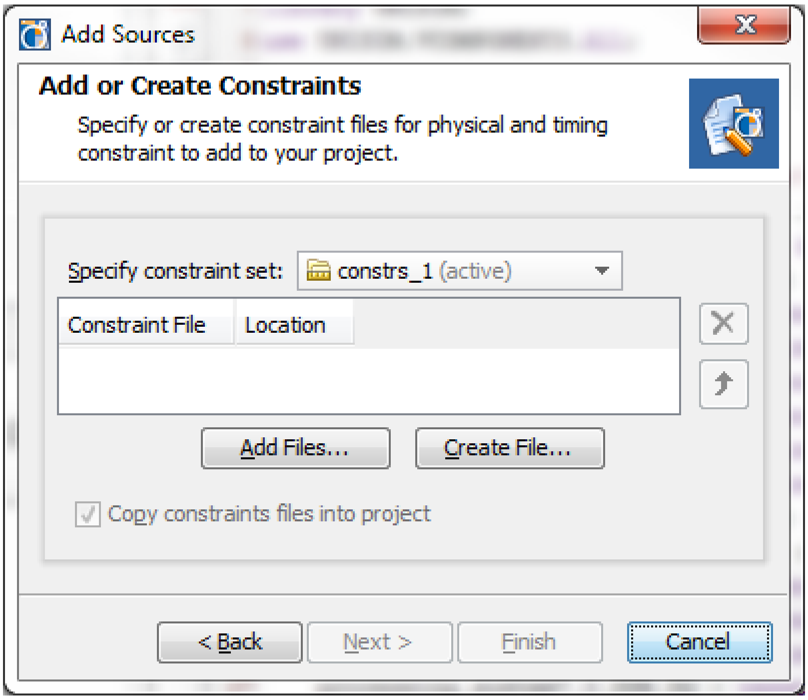
Click "Add Sources" in the "Project Manager" tab.



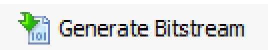
Select "Add or Create Constraints" and click "Next".



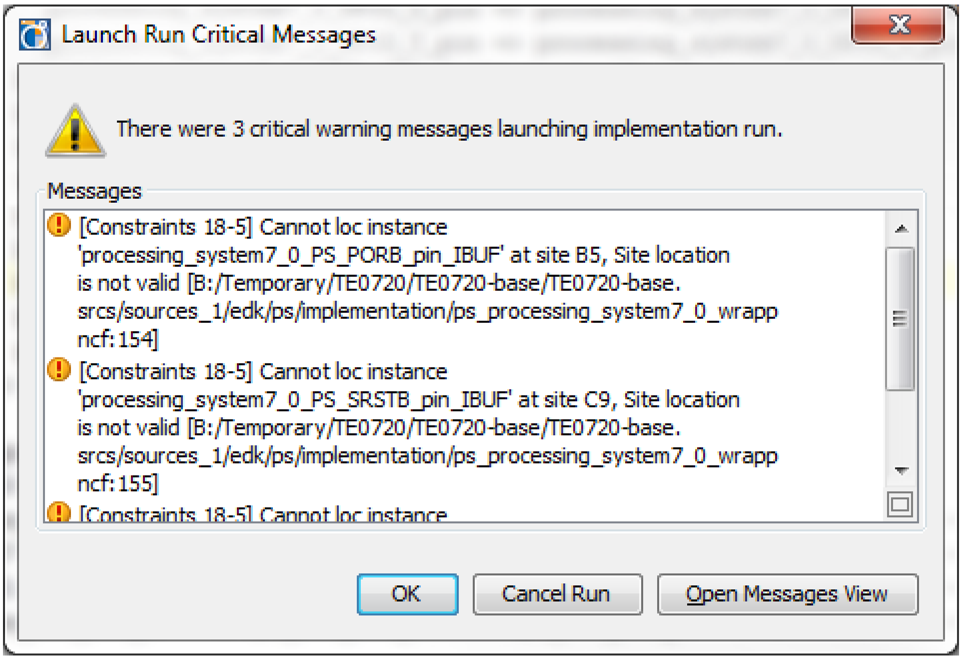
Click "Add Files..." and select *TE0720-01_base.ucf* contained in the TE0720-01-Base.zip archive from the Trenz Electronic Download Area.



Press "Generate Bitstream".

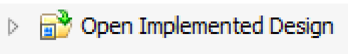


Press "OK" to close the warning window.

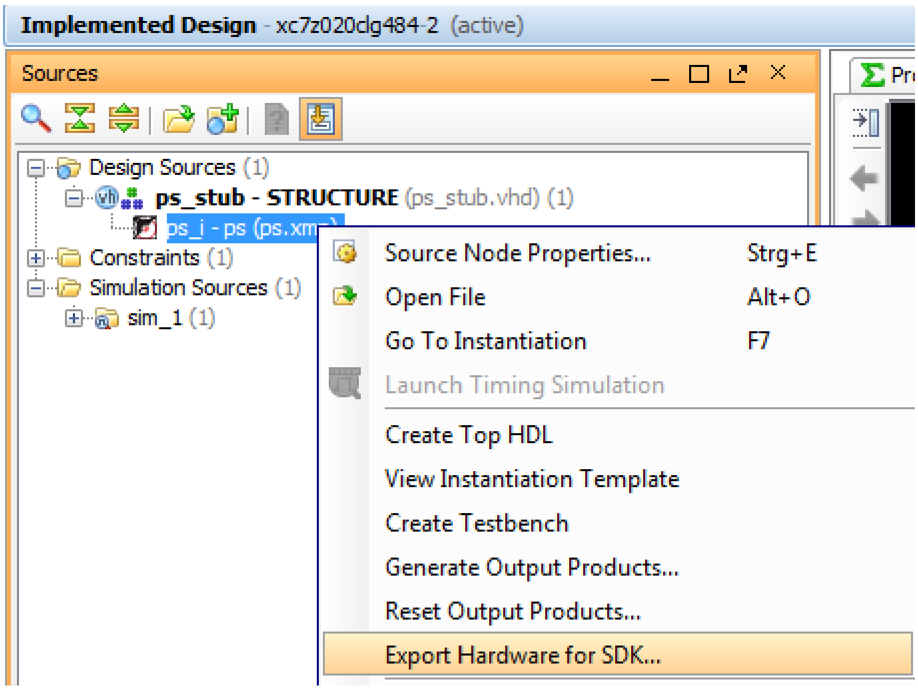


Wait for the operation to complete.

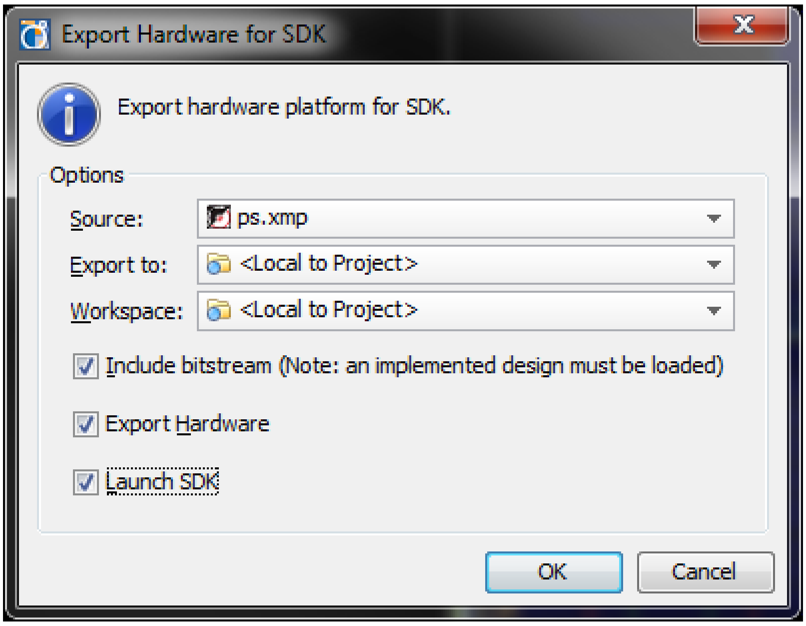
Click "Open Implemented Design".



Right click on the PS block and select "Export Hardware for SDK...".



Select "Include bitstream", "Export Hardware" and "Launch SDK" and press "OK".



This step completes the base system build process. The next step describes how to build an FSBL.

Base XPS Project

⚠ DEPRECATED - use Vivado 2014.2 or newer and start with Board Part Interface flow

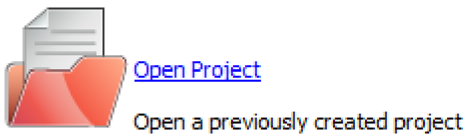
Clone the base project from the following Trenz Electronic GitHub repository.

> *git clone git://github.com/Trenz-Electronic/TE0720-GigaZee-Reference-Designs.git*

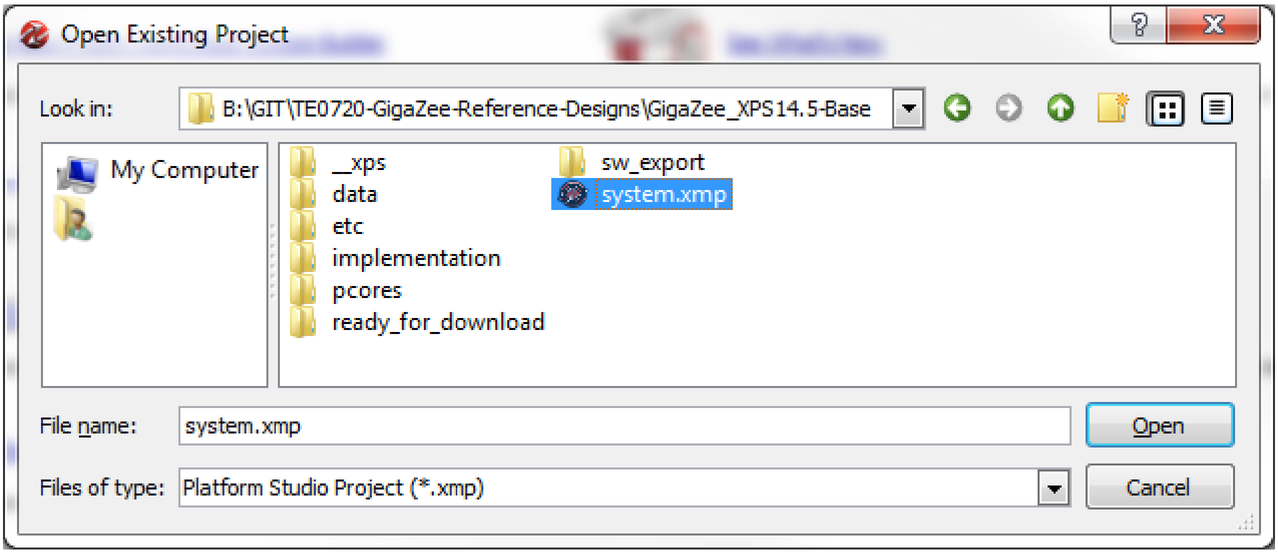
Or just click on "Download ZIP" to download the full project archive without the need to use git.

Run Xilinx XPS 14.5.

Click "Open Project".

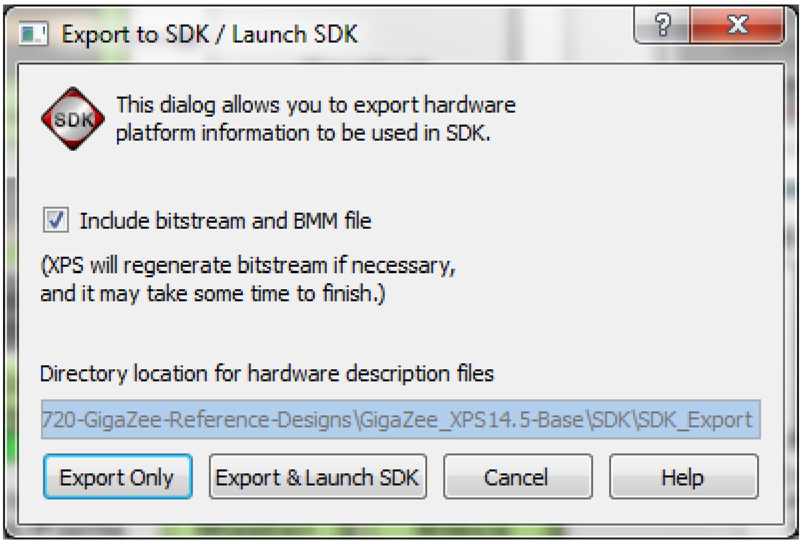


Select "system.xmp" from the project folder and clock "Open".



Click "Export Design" to build the project and export the result to Xilinx SDK.

Click "Export & Launch SDK".



After build, the project will be opened in SDK. The next step is building the First Stage BootLoader.

FSBL - First Stage Boot Loader

DEPRECATED - use Vivado 2014.2 or newer

Vivado 2013.4 FSBL

In 2013.4 much more error detection is added to the FSBL and the handling of those conditions has been changed by Xilinx. This makes it possible that incorrectly generated 2013.4 FSBL does lock-up and prevents any access to JTAG and SPI Flash (if SPI bootmode is selected).

COM15 - PuTTY

```
BAS:IC>TE0701-03 boot...
Xilinx First Stage Boot Loader
Release 2013.4  Jan 14 2014-09:56:54
Devcfg driver initialized
Silicon Version 3.1
Boot mode is QSPI
Single Flash Information
FlashID=0xEF 0x40 0x19
WINBOND 256M Bits
QSPI is in single flash connection
QSPI Init Done
Flash Base Address: 0xFC000000
Reboot status register: 0x60400000
Multiboot Register: 0x0000C000
Image Start Address: 0x00000000
Partition Header Offset:0x00000C80
Partition Count: 2
Partition Number: 1
Header Dump
Image Word Len: 0x000F6EC0
Data Word Len: 0x000F6EBF
Partition Word Len:0x000F6EC0
Load Addr: 0x00000000
Exec Addr: 0x00000000
Partition Start: 0x000065D0
Partition Attr: 0x00000020
Partition Checksum Offset: 0x00000000
Section Count: 0x00000001
Checksum: 0xFFD14B7F
Bitstream
Encrypted
In FsblHookBeforeBitstreamDload function
PCAP:StatusReg = 0x40000A30
PCAP:device ready
PCAP:Clear done
Level Shifter Value = 0xA
Devcfg Status register = 0x40000A30
PCAP:Fabric is Initialized done
PCAP register dump:
PCAP CTRL 0xF8007000: 0x4E00E07F
PCAP LOCK 0xF8007004: 0x0000001A
PCAP CONFIG 0xF8007008: 0x00000508
PCAP ISR 0xF800700C: 0x0802000B
PCAP IMR 0xF8007010: 0xFFFFFFFF
PCAP STATUS 0xF8007014: 0x0006DA30
PCAP DMA SRC ADDR 0xF8007018: 0x00100001
PCAP DMA DEST ADDR 0xF800701C: 0xFFFFFFFF
PCAP DMA SRC LEN 0xF8007020: 0x000F6EC0
PCAP DMA DEST LEN 0xF8007024: 0x000F6EBF
PCAP ROM SHADOW CTRL 0xF8007028: 0xFFFFFFFF
PCAP MBOOT 0xF800702C: 0x0000C000
PCAP SW ID 0xF8007030: 0x00000000
PCAP UNLOCK 0xF8007034: 0x757BDF0D
PCAP MCTRL 0xF8007080: 0x30800100

DMA Done !

FPGA Done !
In FsblHookAfterBitstreamDload function
```

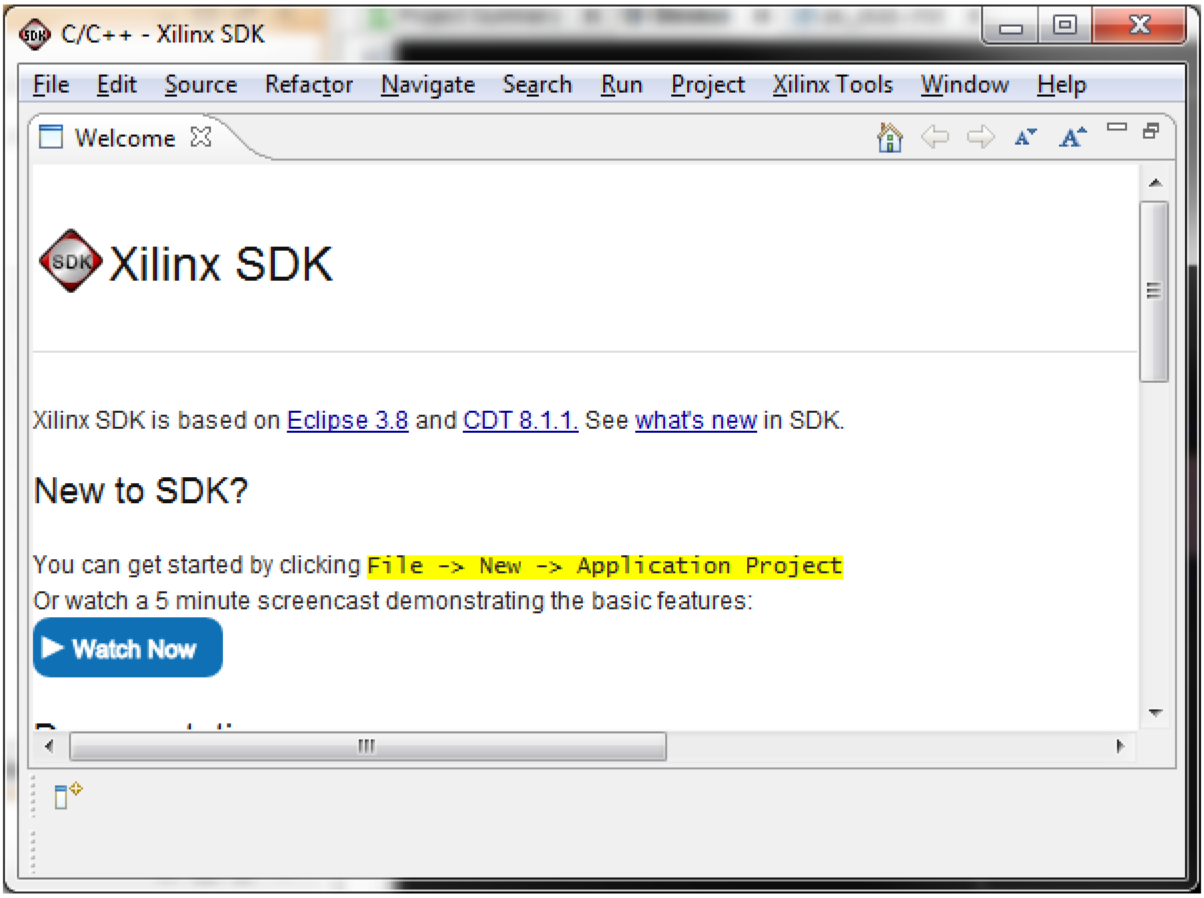
This is FSBL bootlog on TE0701-03 with TE0720, no manual changes made to the FSBL. SPI flash programmed using Xilinx SDK Flash Programmer. Image includes only FSBL and empty BIT file for FPGA, so all it does is FPGA configuration (done LED goes OFF), and JTAG handoff, enabling all JTAG access.

Creating FSBL

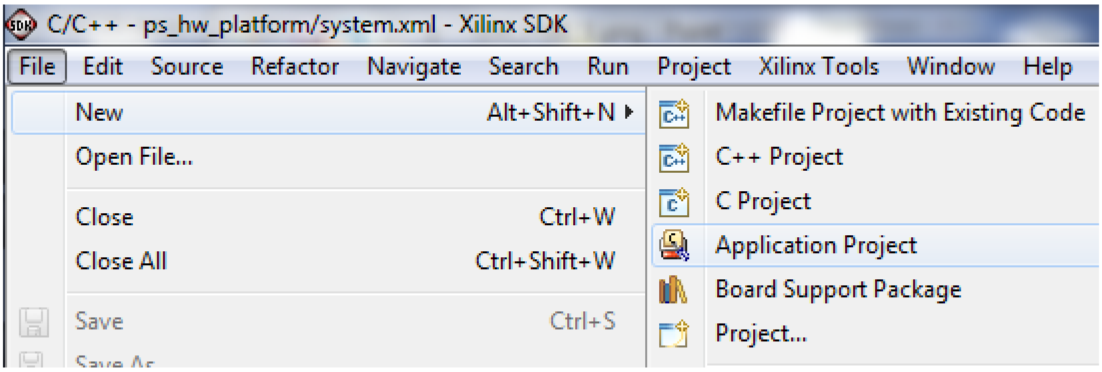
The standard Zynq-7000 FSBL can be used to boot the TE0720 SoM. Unfortunately, SD Card Detect and Write Protect settings in the system configuration is not correctly processed by Xilinx XPS and the generated ps7_init.c file does not contain Card Detect and Write Protect pin configuration code. To make the SD memory card work, apply the patch below or connect the MIO0 pin to ground (default configuration for the Card Detect pin).

The creation of an FSBL requires a base PlanAhead project to be completed and a hardware structure to be exported to SDK (see last steps of the previous Base PlanAhead Project page).

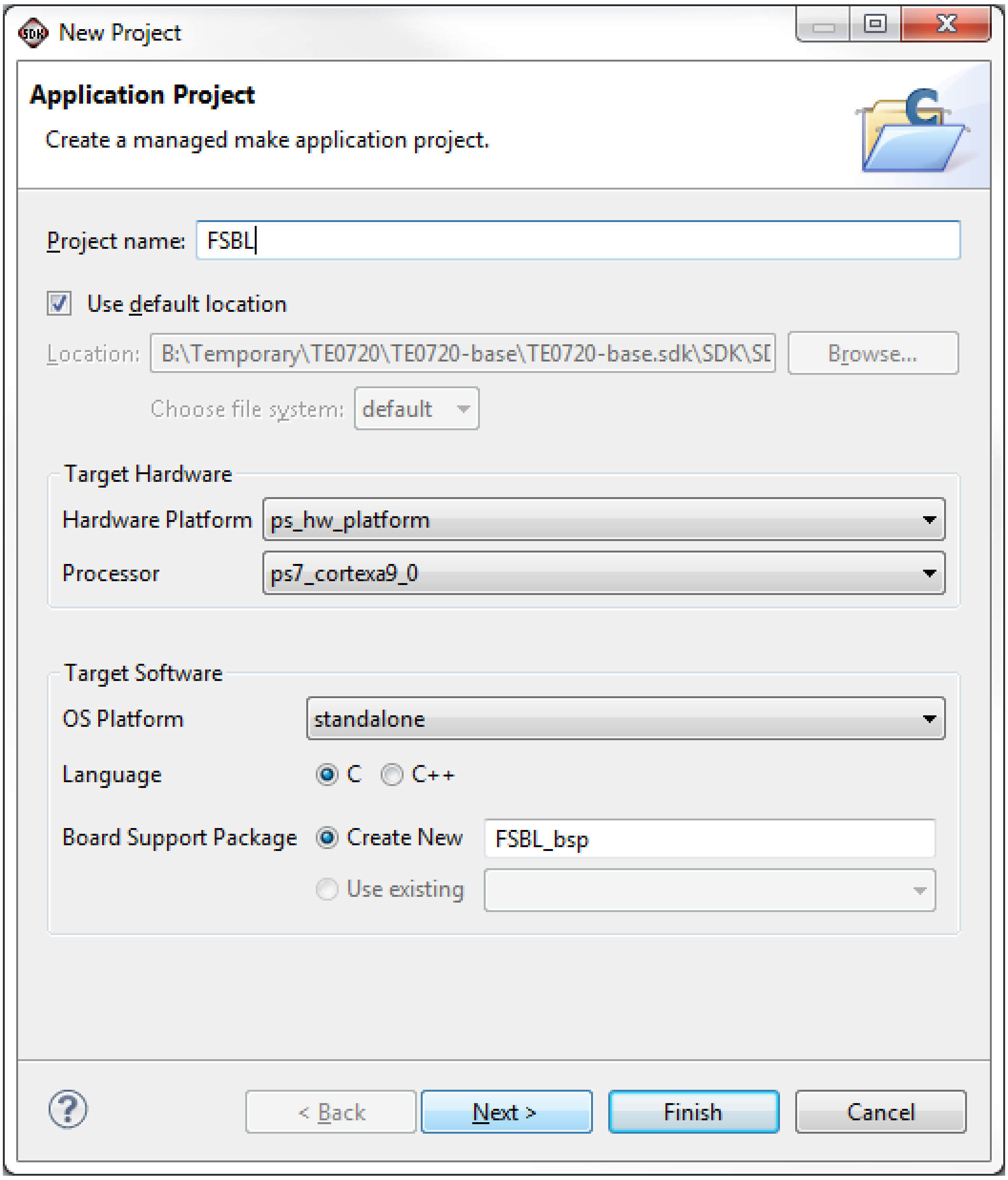
Open the Xilinx SDK and close the "Welcome" tab.



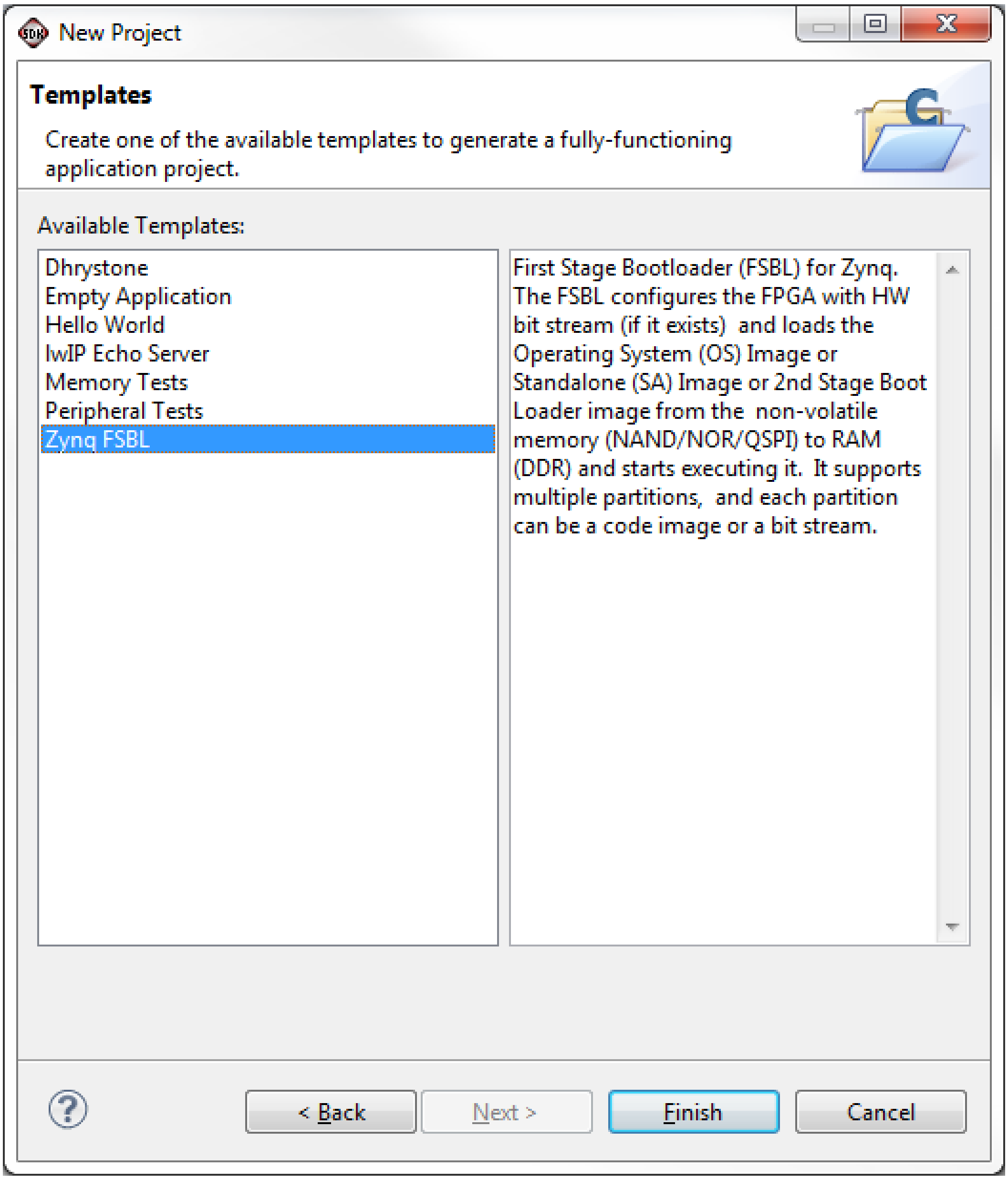
Select **File -> New -> Application Project**.



Specify the "Project name" as "FSBL" and the "Board Support Package" as "FSBL_bsp". Click "Next".



Select the "Zynq FSBL" template and click "Finish".



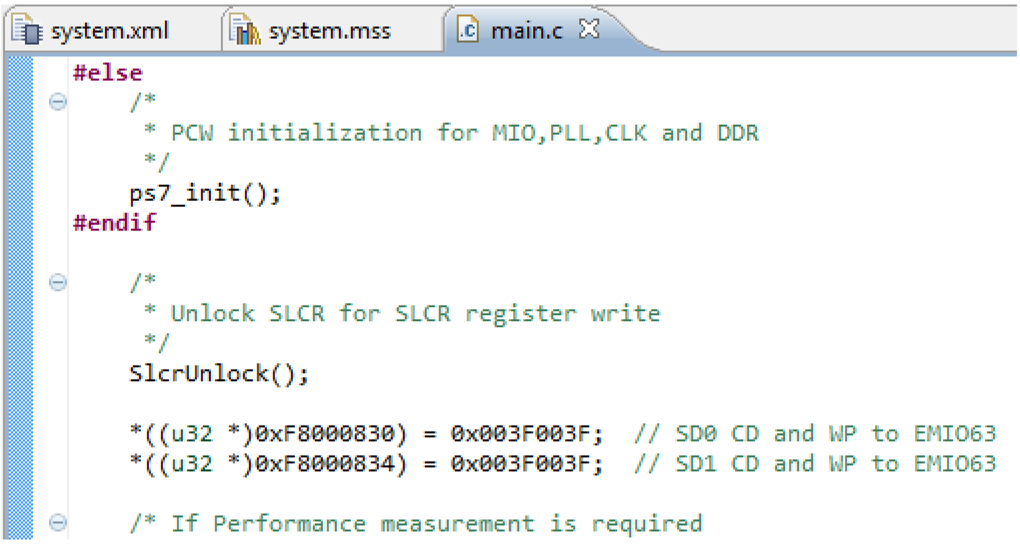
Wait for the build to complete.

If your carrier board doesn't have SD "Card Detect" and "Write Protect" signals, then you have to disable them to make booting from the SD memory card possible.

Insert the following instructions after function "SlcrUnlock();" in "main.c".

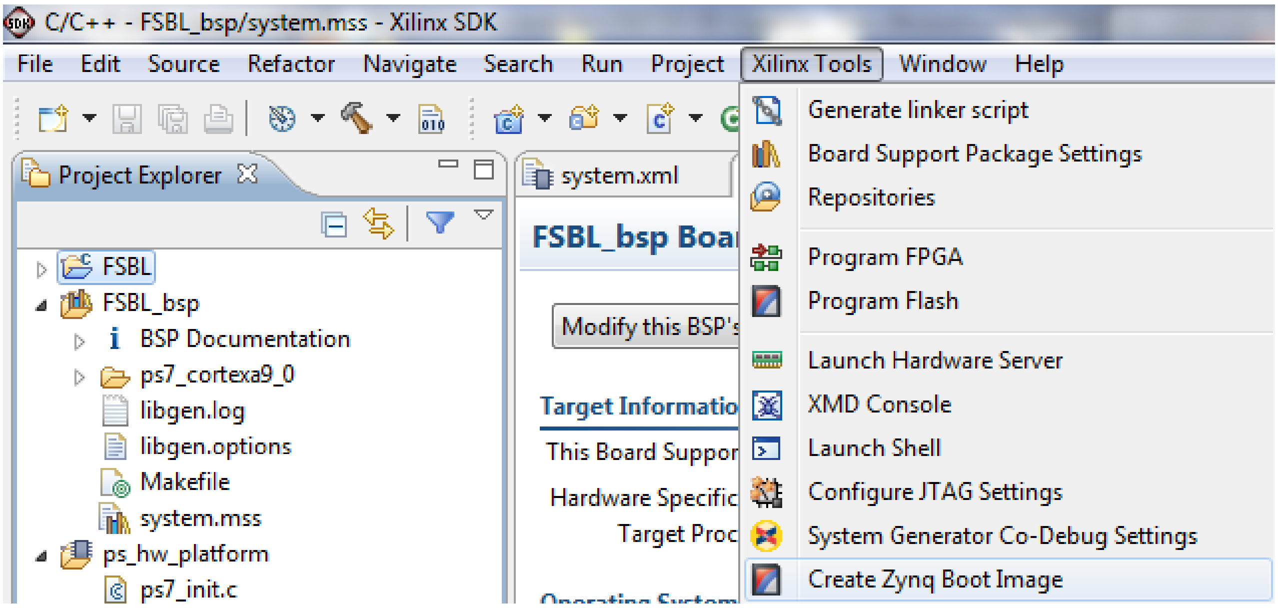
> **((u32 *)0xF8000830) = 0x003F003F; // SD0 CD and WP to EMIO63*
>
> **((u32 *)0xF8000834) = 0x003F003F; // SD1 CD and WP to EMIO63*

This way, the SD "Card Detect" and "Write Protect" signals are connected to EMIO63, which should always be bound to logical 0 or left unconnected (floating).



Save the modified file "main.c": **File -> Save**; the project will be automatically rebuilt.

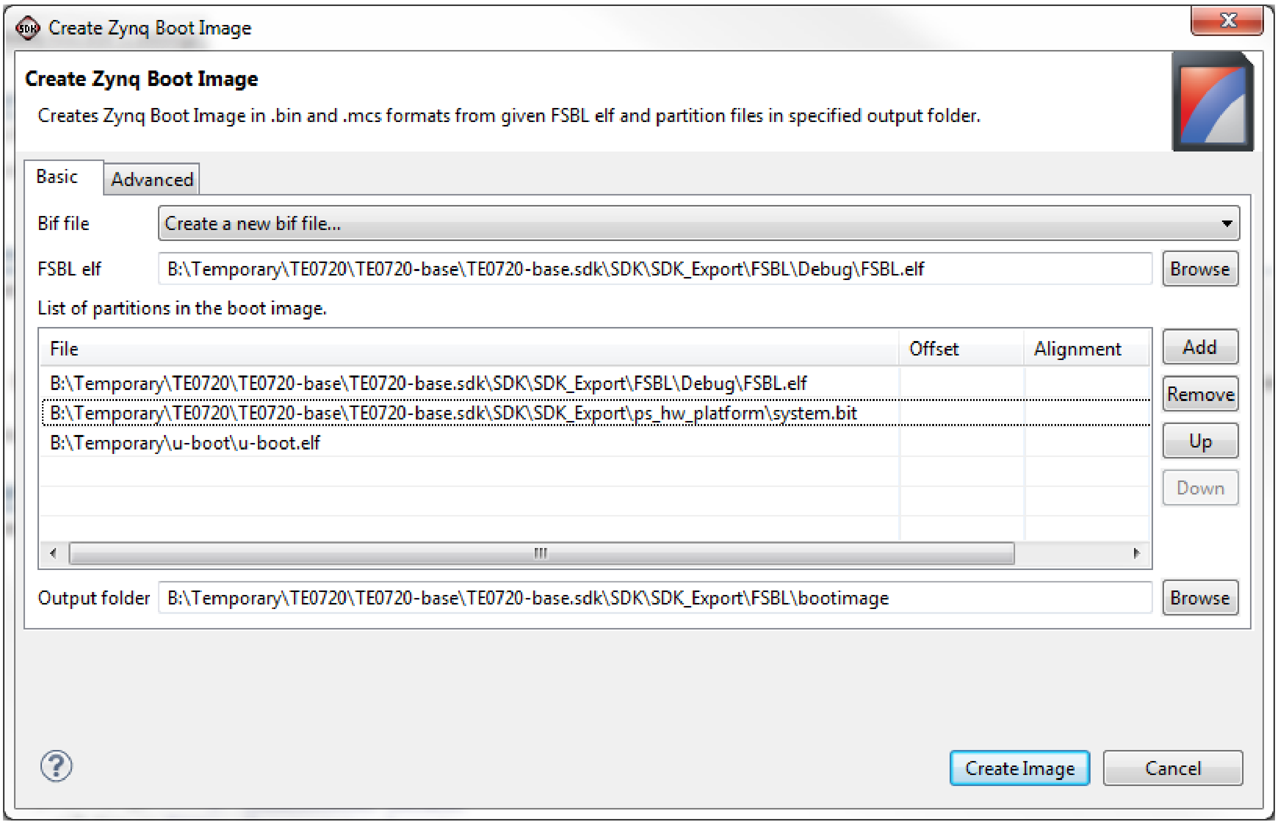
Select the "FSBL" project from the "Project Explorer" panel and run **Xilinx Tools -> Create Zynq Boot Image**.



Click "Add" and select the u-boot.elf file

- contained in the TE0720-01-U-Boot.zip archive from the Trenz Electronic Download Area,
- or built from the source code.

Click "Create Image".



The newly created .\TE0720-base.sdk\SDK\SDK_Export\FSBL\bootimage\u-boot.bin file should be renamed to boot.bin and placed onto the SD memory card.

Build Environment

It is possible to build Linux and u-boot only in a Linux environment; the easiest way to proceed is to install a suitable Linux distribution within a Virtual Machine.

Toolchain

The toolchain is available in the Xilinx download area, but it requires a Xilinx account (username and password). It may be that the user is temporarily not granted download permission. Some time later (could be a few hours to several days) an email from Xilinx will confirm that the user has download permission. The user can now download the installer binaries. When trying to install it on 64-bit Ubuntu, the installer says that it needs 32-bit libs, and provides an URL to a Mentor Graphics web page. This web page requires a Mentor Graphics account; after login, the page provides outdated information about installing ia32-libs.

> Mentor Graphics Sourcery (formerly: CodeSourcery) toolchain has lots of troubles on 64-bit operating systems; some extra libraries need to be installed, both on CentOS and Ubuntu.

Copyright © 2014 Trenz Electronic GmbH

CentOS Linux kernel and the U-Boot build environment

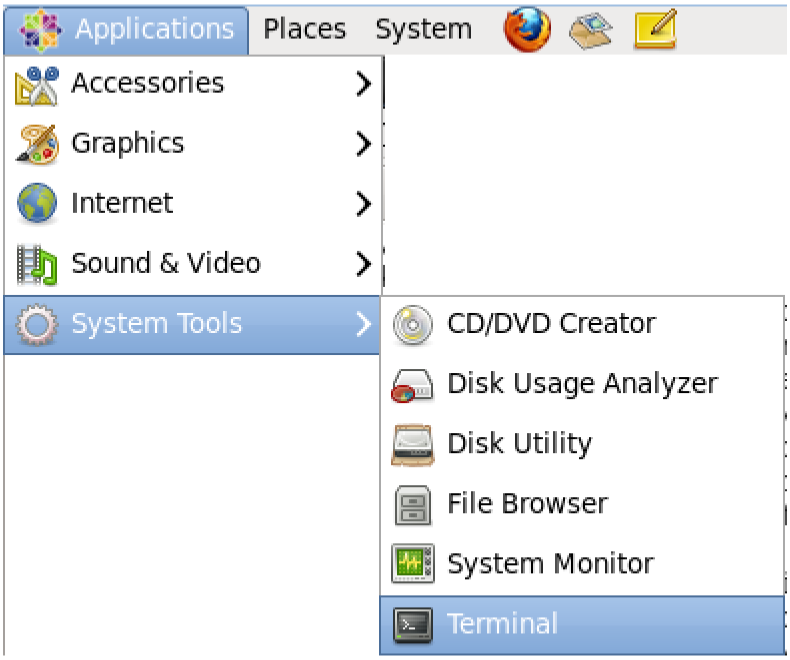
To build U-Boot and the Linix kernel from their source files, you need to create and configure a build environment. VMware Player virtual machine with CentOS 6.4 Linux is used as a build environment.

Required software components:

- VMware Player V5.0.2
- CentOS V6.4 64-bit installer image
- Mentor Sourcery CodeBench GNU Toolchain for Xilinx Zynq-7000 (xilinx-2011.09-50-arm-xilinx-linux-gnueabi.bin, a Xilinx account is required)

Configure CentOS Guest Operating System

Open a "Terminal" window: **Applications -> System Tools -> Terminal**.



Switch to the root user.

> *su*

Install the packages required for the VMware Player tools and the kernel build.

> *yum install make gcc kernel-devel perl*

From the VMware Player menu, run the installation of VMware tools: **Player -> Manage -> Install VMware Tools**.

In the Terminal window, mount the virtual CD-ROM image and install the VMware tools.

> *mkdir /mnt/cdrom*

Copyright © 2014 Trenz Electronic GmbH

```
mount /dev/cdrom /mnt/cdrom

cd /tmp

tar zxpf /mnt/cdrom/VMwareTools*.tar.gz

umount /dev/cdrom

cd vmware-tools*

./vmware-install.pl
```

Add your main user account to the "sudoers" list.

```
cp /etc/sudoers /etc/sudoers.save

echo "user ALL=(ALL) ALL" >> /etc/sudoers
```

where "user" should be replaced by your account name.

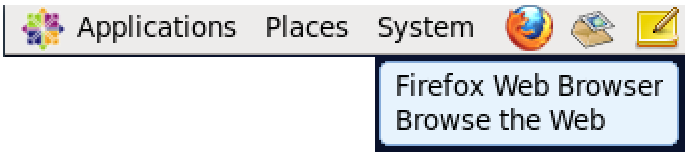
Reboot the virtual machine to apply changes: **System -> Shut Down**.

After reboot, open a "Terminal" window again: **Applications -> System Tools -> Terminal**.

Now complete the configuration.

```
sudo yum update

sudo yum install ncurses-devel git glibc-devel.i686 gtk2-devel.i686 gtk-nodoka-engine.i686 |

libcanberra.i686 libcanberra-gtk2.i686 PackageKit-gtk-module.i686 |

GConf2.i686 ncurses-libs.i686 xulrunner.i686
```

Open the Firefox web browser.



Download the Mentor Sourcery CodeBench GNU Toolchain installer at http://www.xilinx.com/member/mentor_codebench/xilinx-2011.09-50-arm-xilinx-linux-gnueabi.bin (a Xilinx account is required).

Copy the xilinx-2011.09-50-arm-xilinx-linux-gnueabi.bin file to your home directory and run the installation.

```
cp Downloads/xilinx-2011.09-50-arm-xilinx-linux-gnueabi.bin ./

chmod ugo+x xilinx-2011.09-50-arm-xilinx-linux-gnueabi.bin

./xilinx-2011.09-50-arm-xilinx-linux-gnueabi.bin
```

Go through the installation steps choosing the "Typical" installation profile.

Add the path to `CodeSourcery/Sourcery_CodeBench_Lite_for_Xilinx_GNU_Linux/bin/` folder to the `PATH` environment variable (preferably by adding it in the `.bash_profile` script).

Now, your virtual machine environment is ready to build U-Boot and the Linux kernel from their source files.

Ubuntu Linux kernel and U-Boot build environment

Required software components:

- VMware Player V5.0.2
- Ubuntu Desktop 12.04 LTS 32-bit installer image
- Mentor Sourcery CodeBench GNU Toolchain for Xilinx Zynq-7000 (xilinx-2011.09-50-arm-xilinx-linux-gnueabi.bin, a Xilinx account is required)

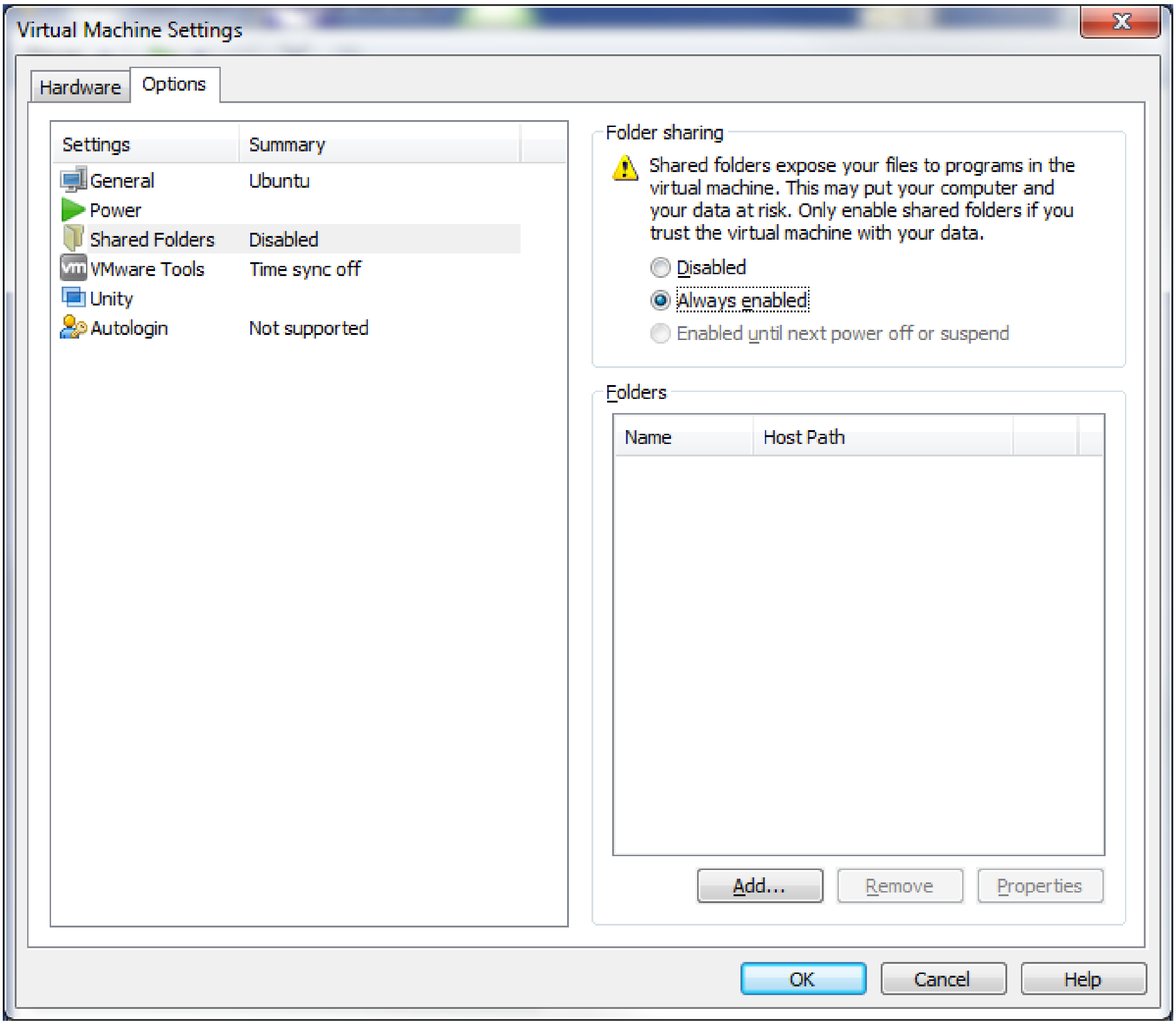
Install Ubuntu Desktop as VMware virtual machine.

After installation enable shared folder function. Shared folder can be used to pass files between guest and host systems.

Open "Virtual machine settings"

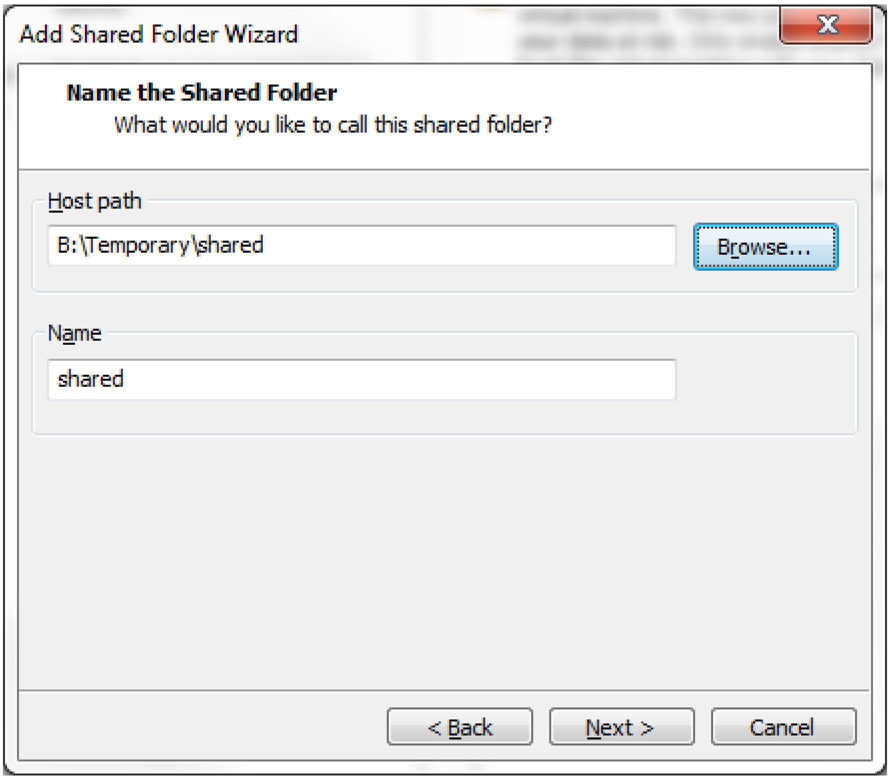
Go "Options" tab and switch "Folder sharing" to "Always enabled" and press "Add...".

Copyright © 2014 Trenz Electronic GmbH

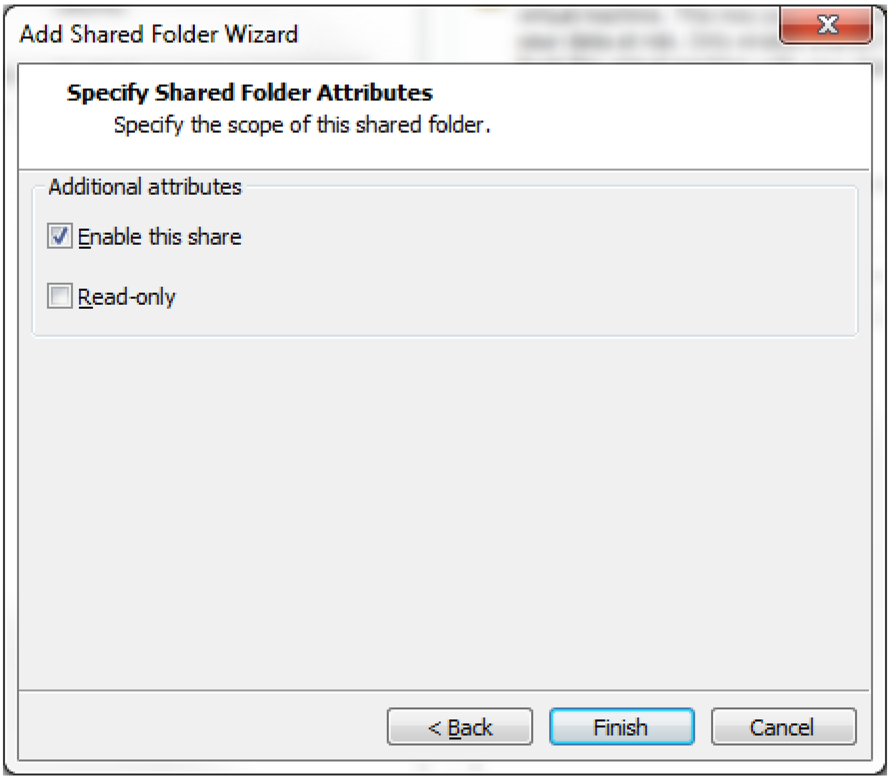

Virtual Machine Settings
Hardware
Options
Settings
Summary
General
Ubuntu
Power
Shared Folders
Disabled
VMware Tools
Time sync off
Unity
Autologin
Not supported
Folder sharing
Shared folders expose your files to programs in the virtual machine. This may put your computer and your data at risk. Only enable shared folders if you trust the virtual machine with your data.
Disabled
Always enabled
Enabled until next power off or suspend
Folders
Name
Host Path
Add...
Remove
Properties
OK
Cancel
Help

Click "Next".

Click "Browse" and select folder for sharing. Set name as "shared". Click "Next".



Click "Finish".



Click "OK" on "Virtual machine settings" window.

In virtual machine window.

Press "Ctrl+Alt+t" to run terminal.

In terminal window

> *sudo apt-get update*
>
> *sudo apt-get install build-essential git ncurses-devel*

```
On Ubuntu 14.04 to install ncurses use

sudo apt-get install libncurses5-dev libncursesw5-dev
```

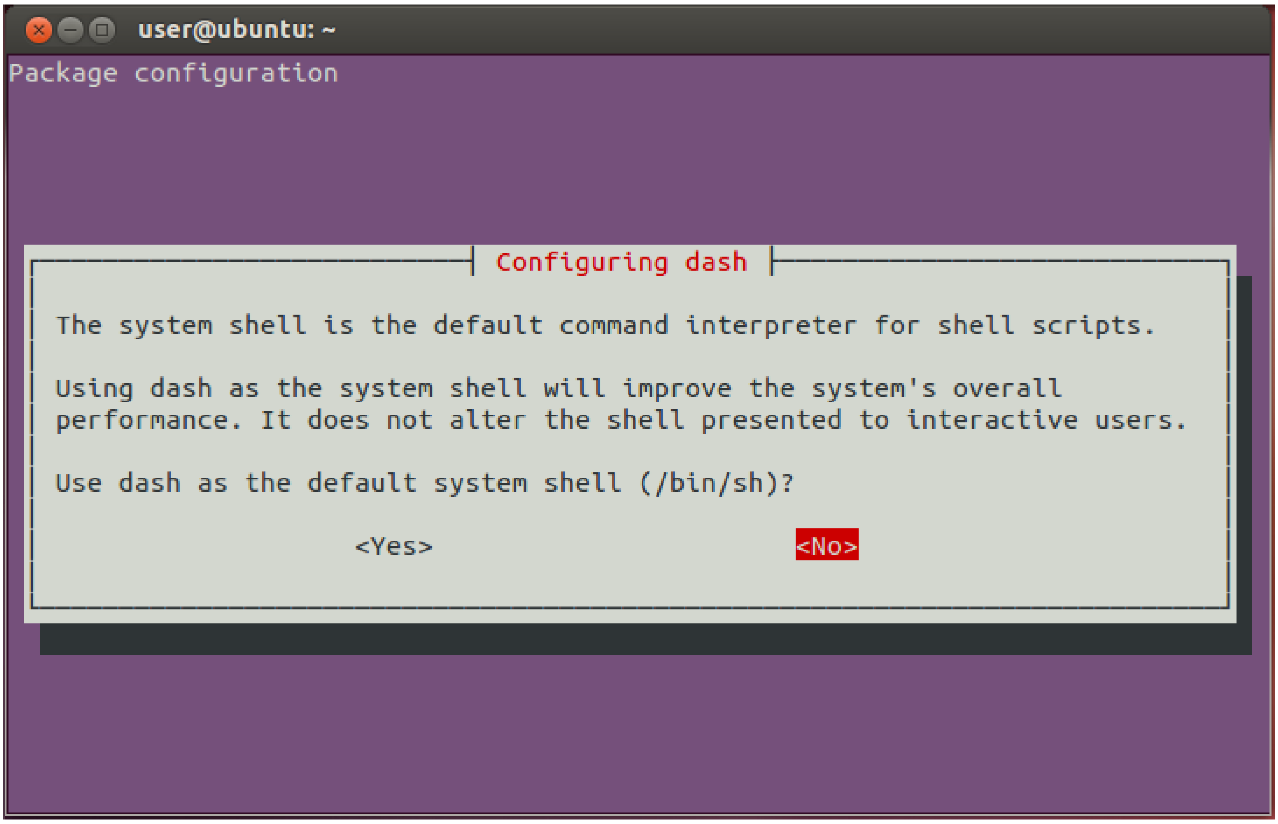
On host system.

Download xilinx-2011.09-50-arm-xilinx-linux-gnueabi.bin from http://www.xilinx.com/member/mentor_codebench/xilinx-2011.09-50-arm-xilinx-linux-gnueabi.bin (You will need Xilinx account to download). Put downloaded file to shared folder.

In virtual machine terminal window.

> *cp /mnt/hgfs/shared/xilinx-2011.09-50-arm-xilinx-linux-gnueabi.bin ./*
>
> *sudo dpkg-reconfigure -plow dash*

Select "<No>"



Run "Sourcery CodeBench GNU Toolchain" installation

> *./xilinx-2011.09-50-arm-xilinx-linux-gnueabi.bin*

Go through the installation steps choosing the "Typical" installation profile.

Now, your virtual machine environment is ready to build U-Boot and the Linux kernel from their source files.

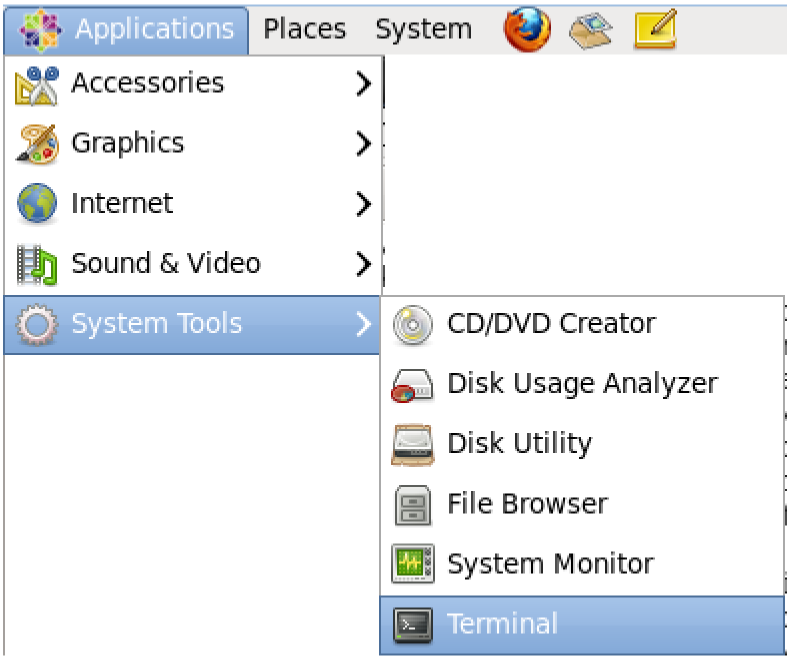
U-Boot

DEPRECATED - use petalinux 2014.2 or newer and TE0720 BSP

Get the U-Boot Repository

Start the build environment (CentOS within the VMware Player virtual machine) prepared in the previous CentOS Linux kernel and the U-Boot build environment page.

Open a Terminal window: **Applications -> System Tools -> Terminal**.



In the terminal window, enter the following command to download the U-Boot source files from the Trenz Electronic Git repository.

> *git clone git://github.com/Trenz-Electronic/u-boot-xlnx.git u-boot-te*

Build U-Boot

In the terminal window, enter the following commands to build U-Boot.

> *cd u-boot-te*
>
> *export CROSS_COMPILE=arm-xilinx-linux-gnueabi-*

The configuration command has to be chosen according to the relevant module version.

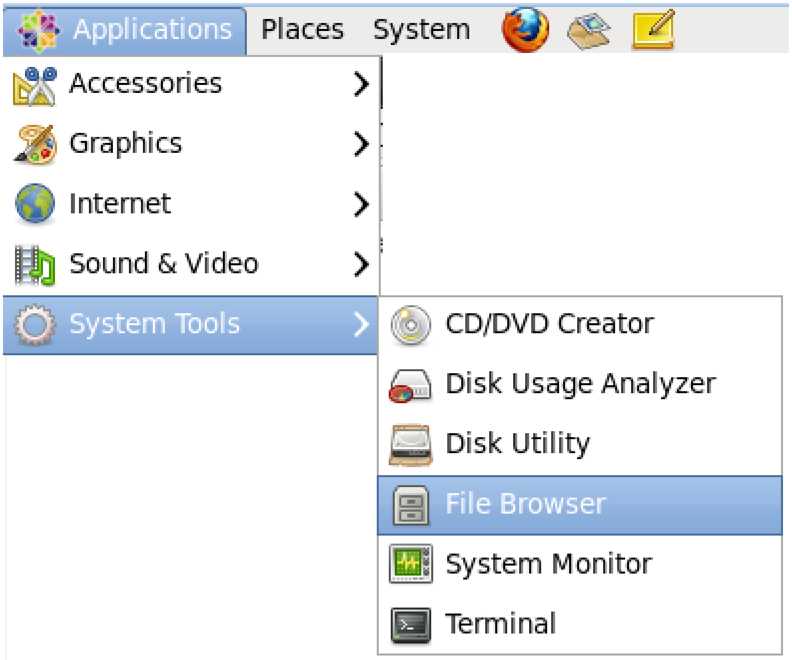
Model	Command
TE0720-01-2IF	make TE0720-01-2IF_config

Model	Command
TE0720-01-2EF	make TE0720-01-2EF_config
TE0720-01-1CF	make TE0720-01-1CF_config
TE0720-01-1CR	make TE0720-01-1CR_config

Build U-boot binary

> *make*

Now, you should have the U-Boot file in the working "u-boot-te" folder. To copy this file to the Microsoft Windows host system, open a file browser: **Applications -> System Tools -> File Browser**.



Locate the "u-boot" file and copy it to the guest system clipboard by pressing "Ctrl+C".

Switch to the Microsoft Windows host system, open a Windows Explorer window, browse to the base PlanAhead project folder and paste the "u-boot" file from the host system clipboard by pressing "Ctrl+V" . Rename "u-boot" to "u-boot.elf".

Now, the "u-boot.elf" file can be used to build the Zynq-7000 boot image (last step of the FSBL build process).

U-Boot user scripting

The default TE0720 u-boot configuration supports user script execution when booting from an SD memory card. The U-Boot script image file **u-boot.cmd**, if it exists, will be executed before the normal boot.

To extend or replace the default SD-boot sequence:

- Create a script file with u-boot commands. Example:
  File name: u-boot.cmd.src
  File content:

> *echo ======= U-Boot user script ==========*

- Pack the script into the u-boot script format:

> *mkimage -T script -C none -n 'User script' -d u-boot.cmd.src u-boot.cmd*

- Copy the u-boot.cmd file to the SD memory card.

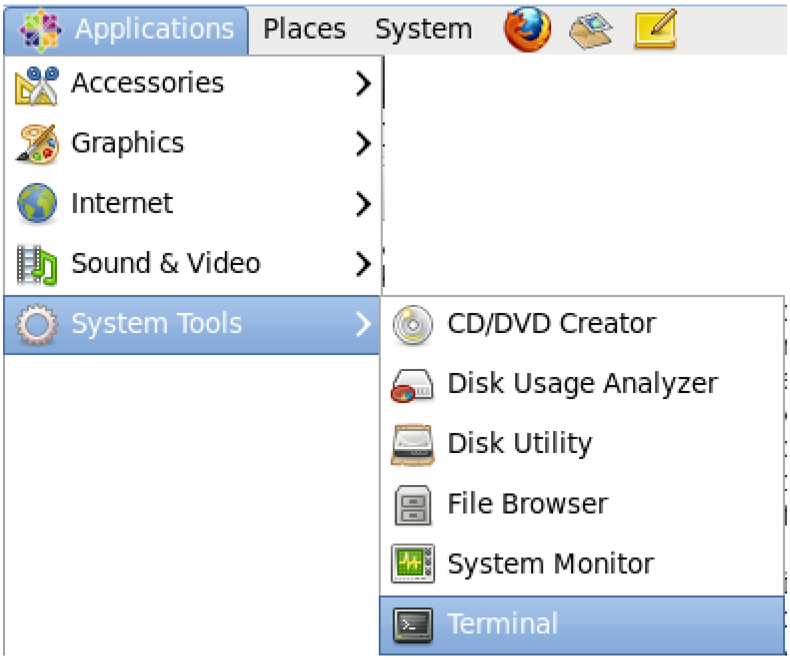
Linux kernel 3.9

⚠ DEPRECATED - use petalinux 2014.2 or newer with TE0720 BSP

Get Trenz Electronic Linux Kernel Repository

Start the build environment (CentOS within the VMware Player virtual machine) prepared in the previous CentOS Linux kernel and the U-Boot build environment page.

Open a Terminal window: **Applications -> System Tools -> Terminal**.



In the terminal window, enter the following command to download the Linux Kernel source files from the Trenz Electronic Git repository.

> *git clone git://github.com/Trenz-Electronic/linux-te-3.9*

Build the Linux Kernel

To build a U-Boot wrapped image, the `PATH` environment variable should contain the path to the `mkimage` executable. It can be done by adding the path to the `~/u-boot-te/tools/` folder (its path depends on your U-Boot repository location) to the `PATH` environment variable, or installing the `uboot-tools` package.

In the terminal window, enter the following commands to build the Linux kernel.

> *cd linux-te*
>
> *export CROSS_COMPILE=arm-xilinx-linux-gnueabi-*
>
> *make ARCH=arm te_zynq_defconfig*

> *make ARCH=arm LOADADDR=0x8000 uImage*

Compile the device tree binary using one of the following commands according to the relevant module version.

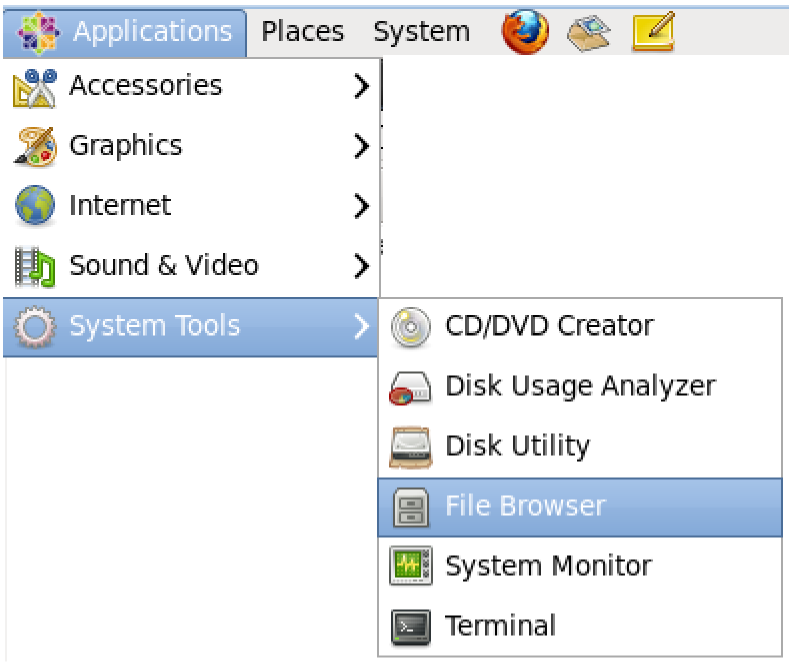
Model	Command
TE0720-01-2IF	make ARCH=arm TE0720-01-2IF.dtb
TE0720-01-2EF	make ARCH=arm TE0720-01-2EF.dtb
TE0720-01-1CF	make ARCH=arm TE0720-01-1CF.dtb
TE0720-01-1CR	make ARCH=arm TE0720-01-1CR.dtb

The resulting device tree blob (TE0720-01-2IF.dtb, TE0720-01-2EF.dtb, TE0720-01-1CF.dtb or TE0720-01-1CR.dtb) will be created into the ./arch/arm/boot/dts folder.

Rename the resulting device tree blob as `devicetree.dtb` by using one of the following commands according to the relevant module version.

Model	Command
TE0720-01-2IF	mv ./arch/arm/boot/dts/TE0720-01-2IF.dtb ./arch/arm/boot/dts/devicetree.dtb
TE0720-01-2EF	mv ./arch/arm/boot/dts/TE0720-01-2EF.dtb ./arch/arm/boot/dts/devicetree.dtb
TE0720-01-1CF	mv ./arch/arm/boot/dts/TE0720-01-1CF.dtb ./arch/arm/boot/dts/devicetree.dtb
TE0720-01-1CR	mv ./arch/arm/boot/dts/TE0720-01-1CR.dtb ./arch/arm/boot/dts/devicetree.dtb

Now, you should have the Linux kernel image (wrapped for u-boot) **./arch/arm/boot/uImage** and the device tree blob **./arch/arm/boot/dts/devicetree.dtb**. To copy these files to the Microsoft Windows host system, open a file browser: **Applications -> System Tools -> File Browser**.



Copyright © 2014 Trenz Electronic GmbH

Locate the files and copy them to the guest system clipboard by pressiing "Ctrl+C".

Switch to the Microsoft Windows host system, open a Windows Explorer window, browse to a suitable folder and paste the files from the host system clipboard by pressing "Ctrl+V".

Preparing boot media

TE0720 module supports 2 boot modes:

- SD memory card boot
- QSPI Flash memory boot

SD memory card boot

The SD memory card Zynq should boot from shall contain a FAT partition with the following files:

File	Build	Description
boot.bin	FSBL - First Stage Boot Loader	Zynq boot image file. Contains fsbl.elf, system.bit and u-boot.elf.
devicetree.dtb	Linux	Linux device tree blob file
uImage	Linux	Linux kernel image (with u-boot wrapper)
uramdisk.image.gz	Build and Modify a Rootfs	Linux ramdisk (with u-boot wrapper)

Care should be taken not to corrupt the file system, please use "halt" or "umount /mnt/sd0" before power off board.

QSPI Flash memory boot

Unfortunately, the Winbond W25Q256 QSPI Flash memory assembled on the TE0720 is not yet suported by Xilinx iMPACT software tool. The easiest way to initialize the on-board QSPI Flash memory is to copy the data from the SD memory card using u-boot.

- Prepare the SD memory card as specified in SD memory card boot
- Connect the UART to the MIO pins (MIO14/MIO15 used in default FSBL), set the baudrate to 115200, no parity
- Set the boot mode to *SD memory card*
- Insert the SD memory card and power on the module
- Stop u-boot boot sequence by pressing the `enter` key during boot countdown
- Load data from the SD memory card to the DDR3 memory using the "**run sdfetch**" u-boot command
- Write data to the QSPI Flash memory from the DDR3 memory using "**run reflash_all**" u-boot command

After that, the QSPI Flash memory boot mode can be enabled and used.

QSPI Flash memory map

Address	Size	File	Description
0x000000	0x450000	boot.bin	Zynq boot image

Address	Size	File	Description
0x450000	0x500000	uImage	Linux kernel image
0x950000	0x020000	devicetree.dtb	Linux device tree blob
0x970000	0x5E0000	uramdisk.image.gz	Linux ramdisk image

To change these addresses, the u-boot default configuration shall be changed and u-boot shall be rebuilt.

Base Vivado Project

Clone the base project from the following Trenz Electronic Github repository
https://github.com/Trenz-Electronic/TE0720-GigaZee-Reference-Designs/tree/master/TE0720-01_Base_Viva

or download it from "Download area"
http://www.trenz-electronic.de/fileadmin/docs/Trenz_Electronic/TE0720-GigaZee/reference_designs/TE0720-

Run Vivado 2013.4

In "Tcl Console"

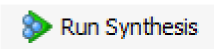
> *cd c:/temp/TE0720-01_Base_Vivado-2013.4/*

where "c:/temp" should be replaced to real path to project directory.

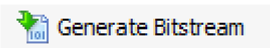
In "Tcl Console"

> *source create.tcl*

This command create new project in "base" directory and import used files.

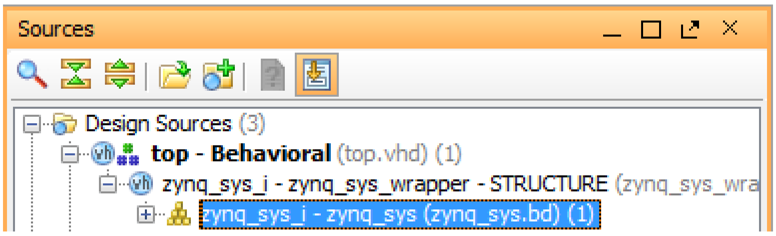
Click 

and wait synthesis to complete.

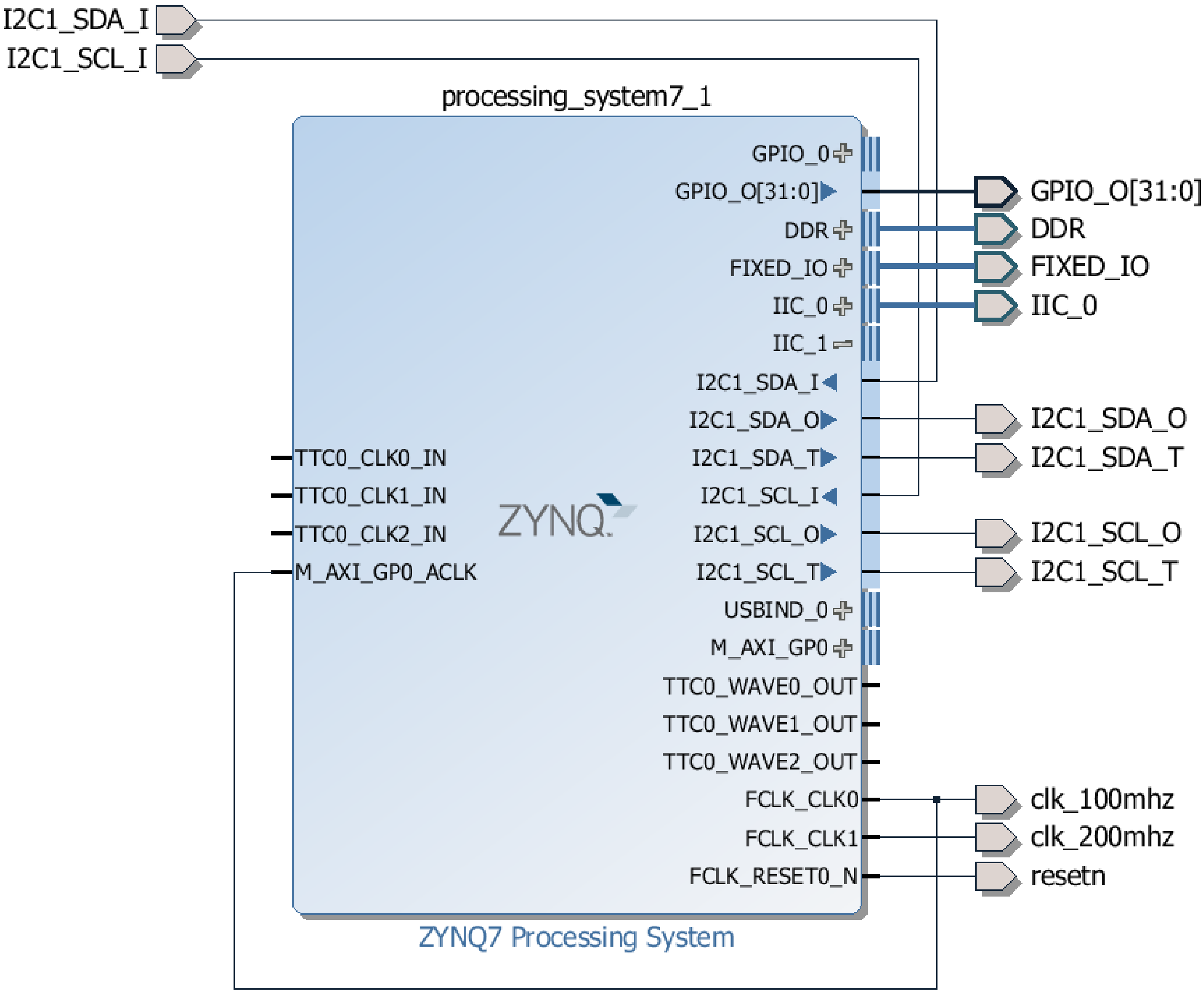
Click 

and wait operation to be completed.

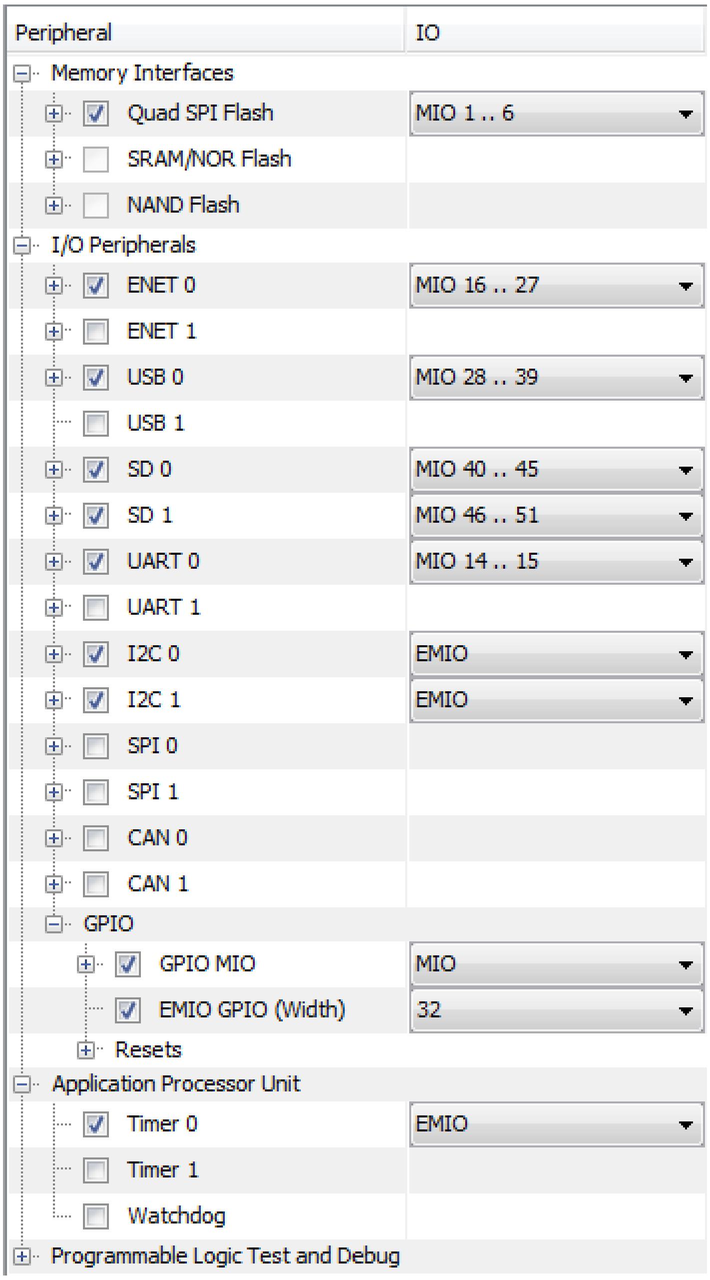
Double click on "zynq_sys_i" to open block design.



"zynq_sys" should looks like

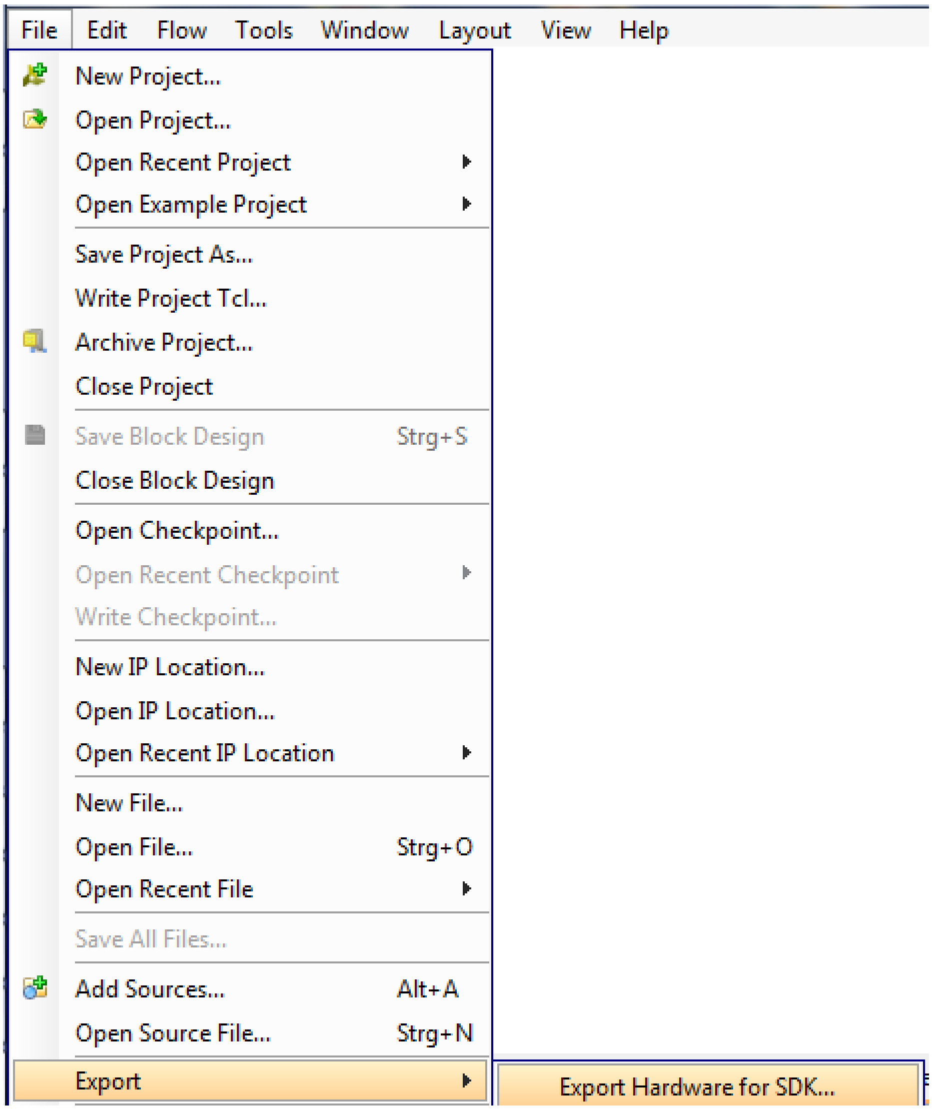


MIO configuration is

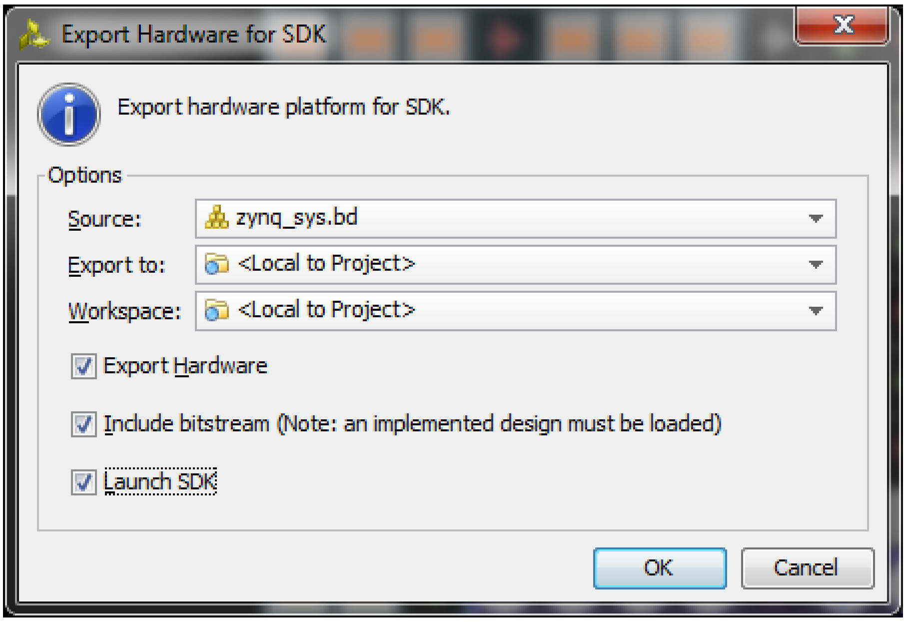


Click 

Select "File - Export - Export Hardware for SDK..."

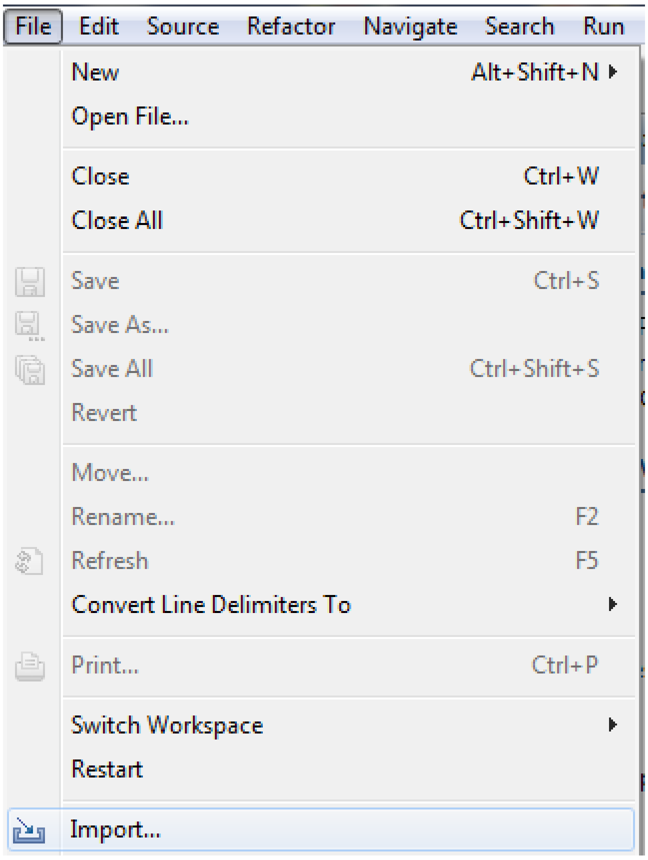


Select all checkboxes and press "OK"

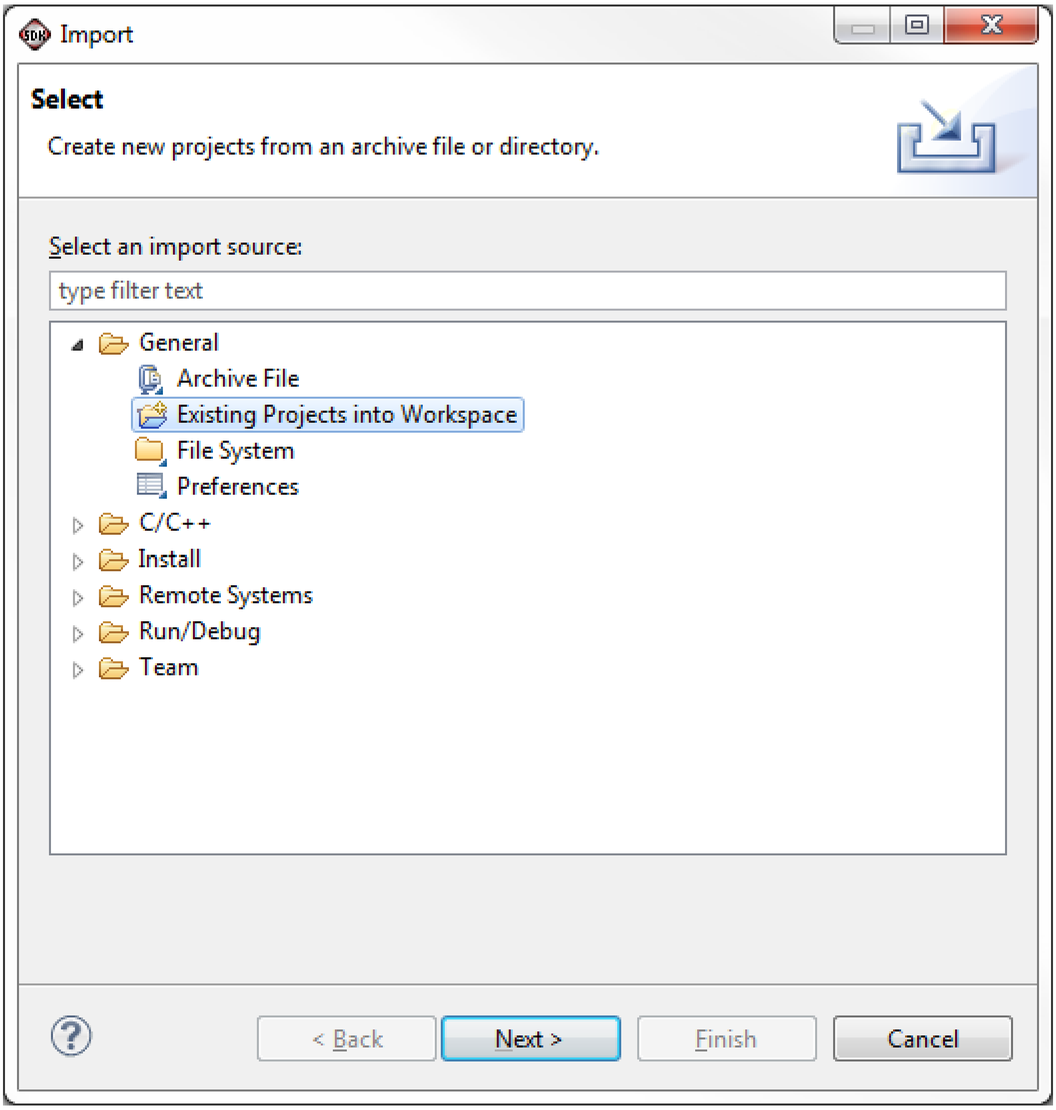


In "Xilinx SDK" window

Select "File - Import..."

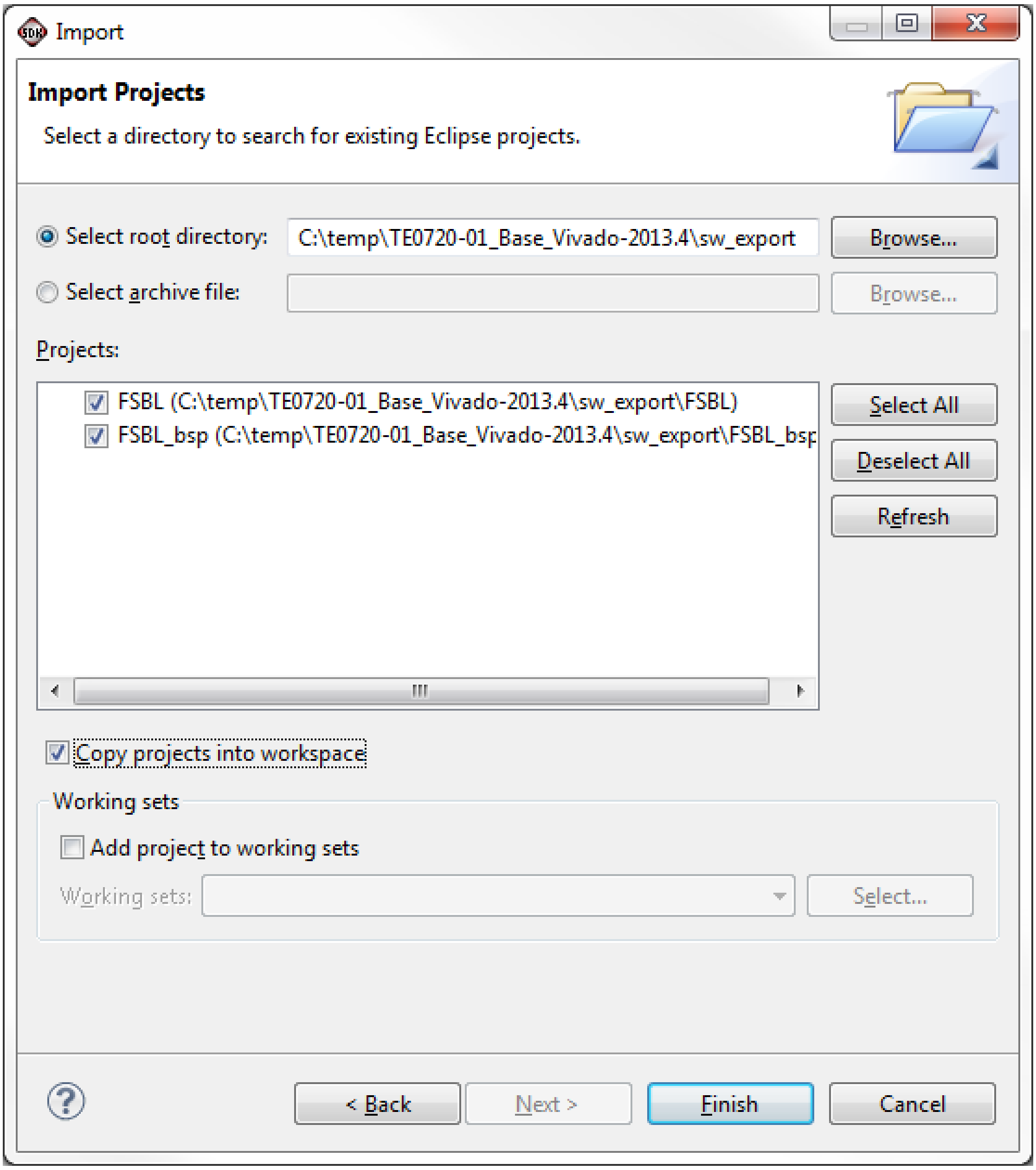


Select "General - Existing Projects into Workspace" and press "Next >"

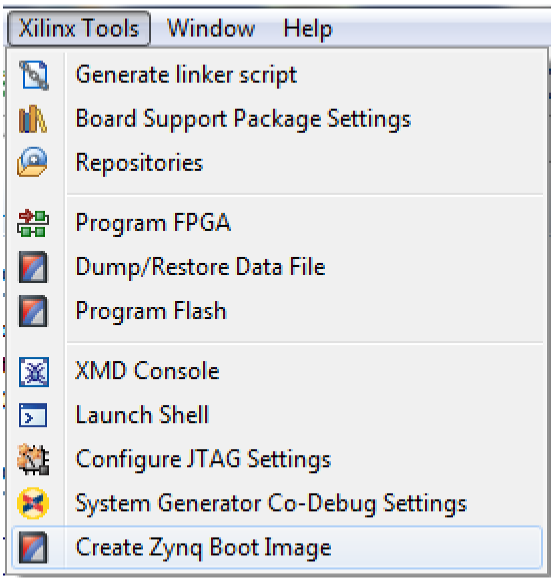


Check "Select root directory:" and press "Browse". Navigate to "sw_export" directory in project.

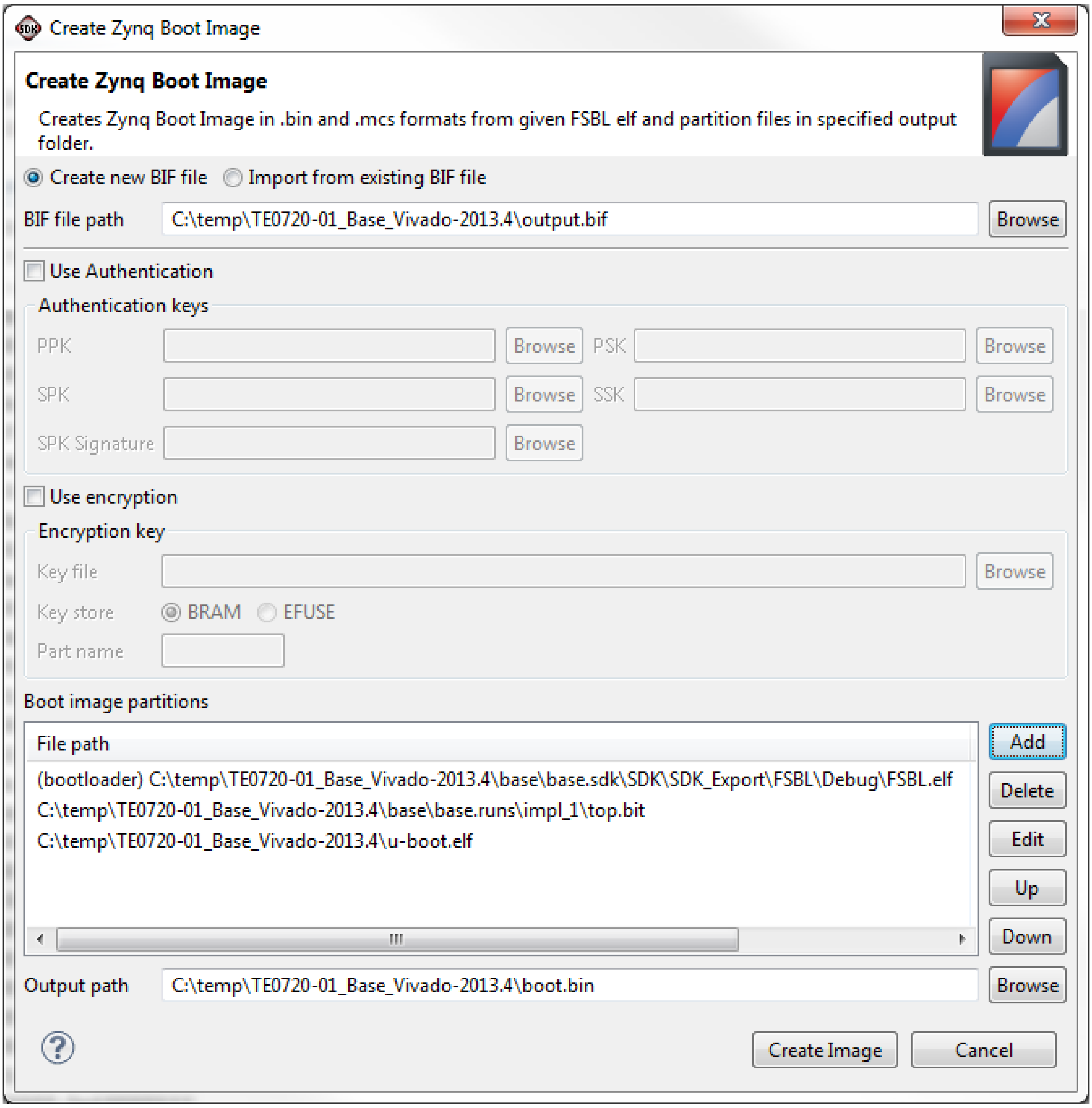
Check "Copy projects into workspace". Press "Finish"



Project will build automatically. After complete build, run "Xilinx Tools - Create Zynq Boot Image".



Select path for output.bif and Add FSBL, bit and u-boot files to Image.



Click "Create Image".

Copy "boot.bin", "devicetree.dtb", "uImage" and "uramdisk.image.gz" files from project directory to SD card.

Vivado Flow (Video and Step-by-Step Tutorial)

Press "<Strg> + <Pos1>" to return back to this overview!

Creating a Vivado Example Project for TE0720 Zynq SoC Module

Video Tutorial (Vivado 2013.2)

The following screen-recording video shows how to create a new project in Vivado, how to set up a Zynq system, how to configure it properly, and how to export it to SDK. In SDK a standard *First Stage Boot Loader* (FSBL) is created and a "Hello world" project is generated as well as compiled. As a final step both software parts (FSBL and "Hello world") are combined in a boot image that is ready to be flashed or copied to an SD card:

The FSBL from this project could also be started in a debugger but it would not make any terminal output. Just proceed to load from SPI flash or SD Card depending on the boot mode when started in a debugger. The "Hello world" project can be run from the debugger but ps7_init (see TE0720 GigaZee Zynq SoM / TE0720 User Manual / FSBL - First Stage Boot Loader) TCL script must be executed first.

To start the "Hello World" project from an SD Card just copy the file "BOOT.BIN" to the SD card.

Note: This project does not include any FPGA bitstream, so the FPGA will not be configured and LED3 on the TE0720 module will remain lit!

Step-by-Step Tutorial (Vivado 2013.3)

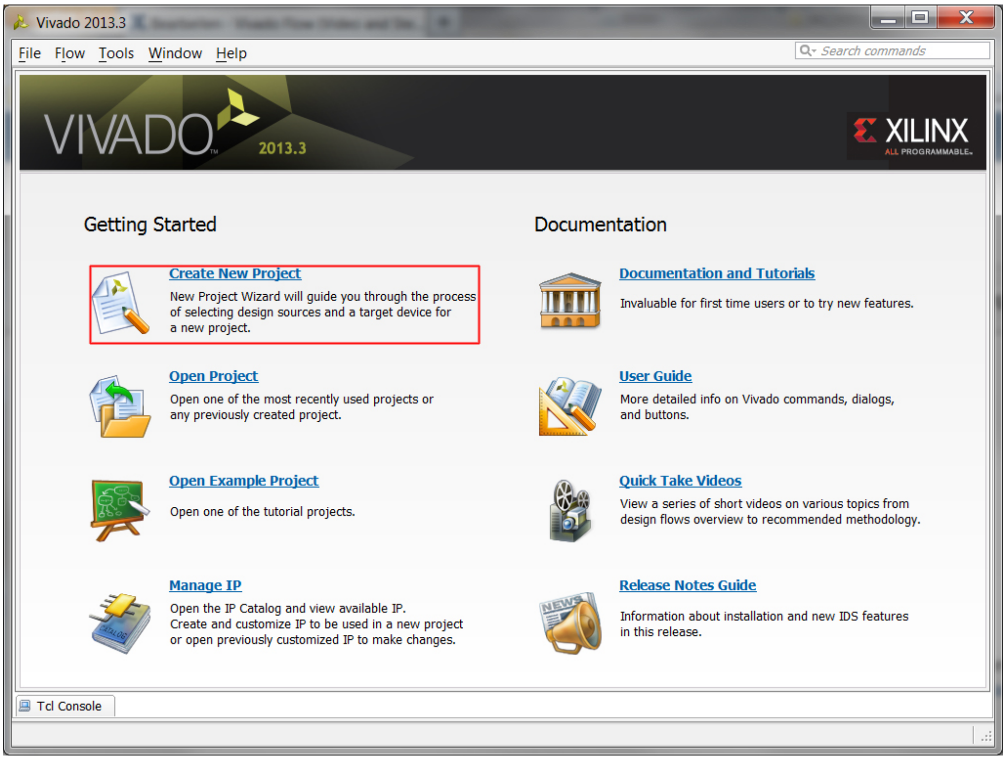
Full Board Support Packages for Vivado CAN NOT BE CREATED AT THIS TIME. We have filed a WebCase (990721) and Xilinx has Answer Record AR58180 regarding this. The feature to create board support packages will hopefully be available with some later Vivado releases, but currently it is not known in which version. At least it is not confirmed for 2014.1 for sure.

Currently, we are working on a patch so that a custom board support package can be made available manually by just copying our TE0720 BSP files into the corresponding Vivado subdirectory (e.g., C: \Xilinx\Vivado\2013.3\data\boards\zynq). Meanwhile, corresponding to the Answer Record AR58180 you have to choose as part the on-board Xilinx Zynq FPGA device (e.g., xc7z020clg484-2) and specify the remaining settings (e.g., pin locking, IO voltages etc.) in a user defined contraint file as described in the following tutorial.

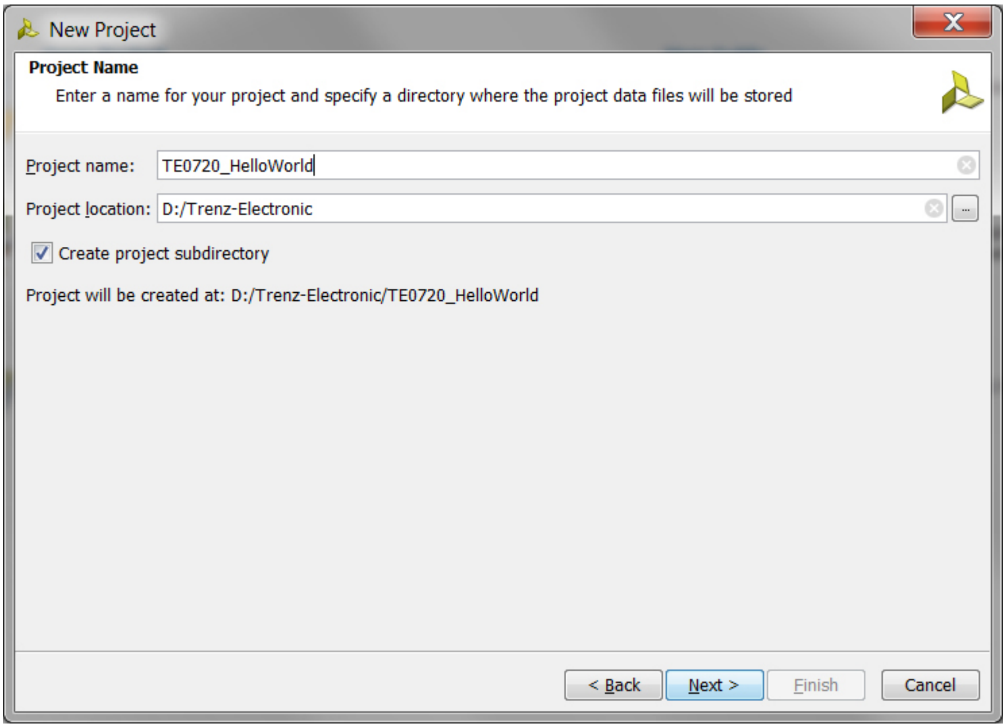
In the following, our video tutorial above is shown in a step-by-step manner (to whom it may prefer):

Getting Started: Create New Vivado Project

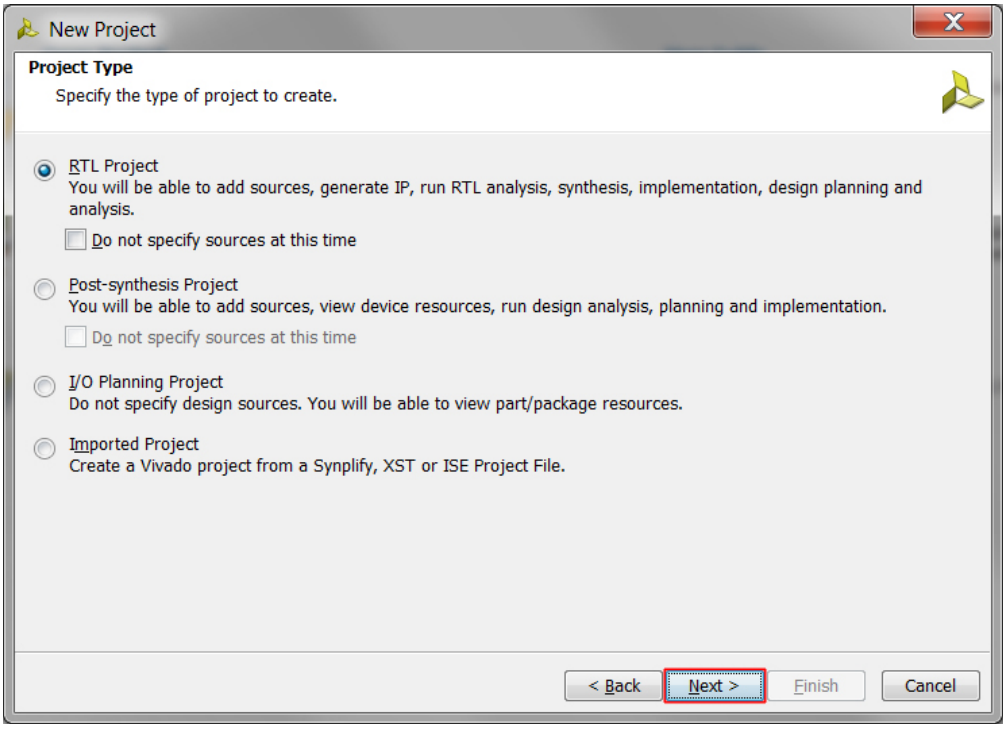
1.) Create a new project in Vivado 2013.3:



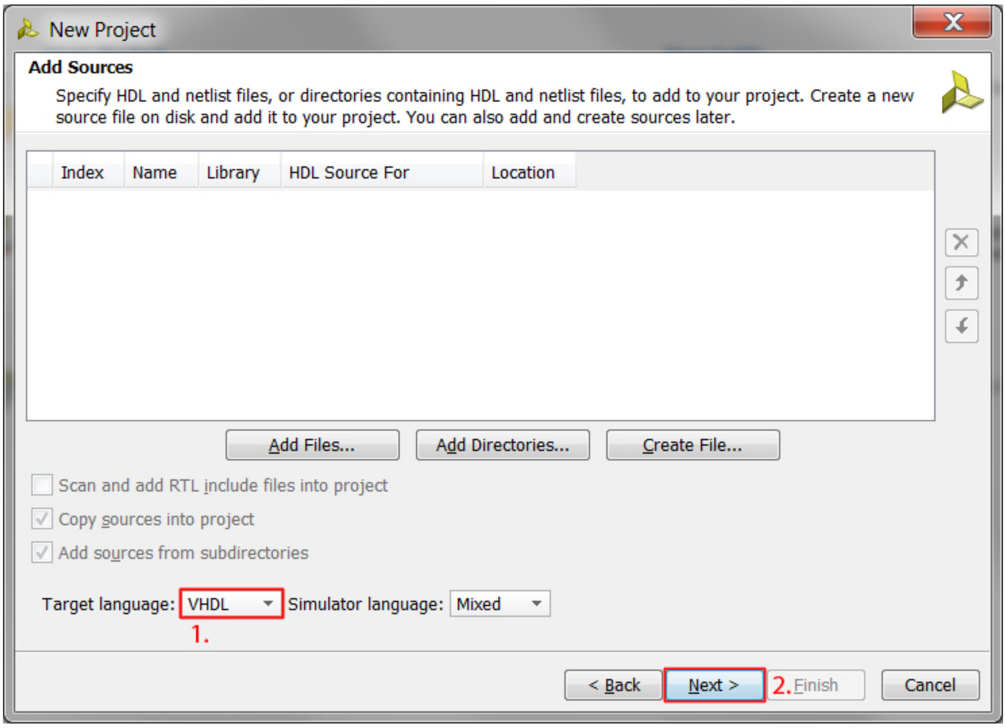
2.) Choose an appropriate project name, e.g., "TE0720_HelloWorld", and project directory location:



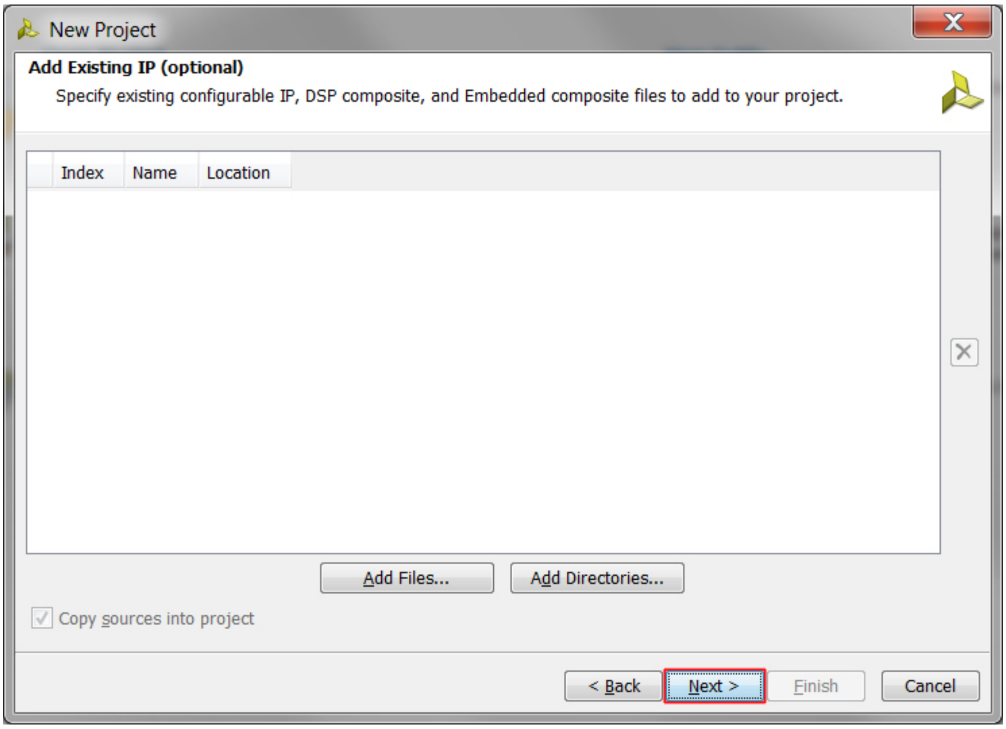
3.) The type of project should be RTL Project with the option "Do not specify soruces at this time" disabled:



4.) The target language is selected to be VHDL:

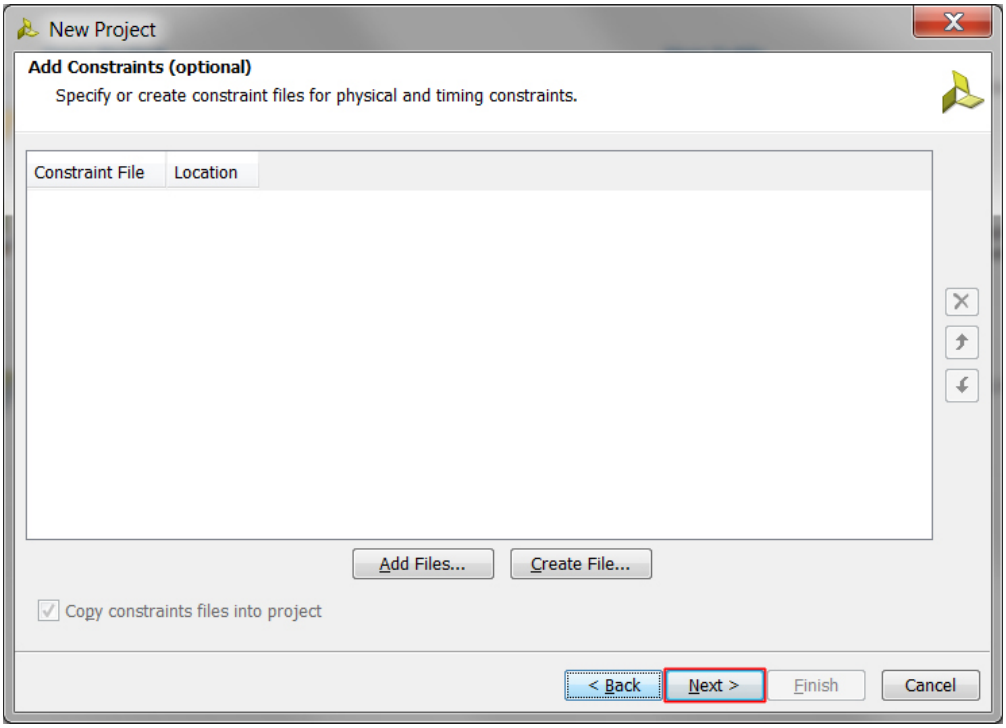


5.) ... and in the dialog no existing IP is added...

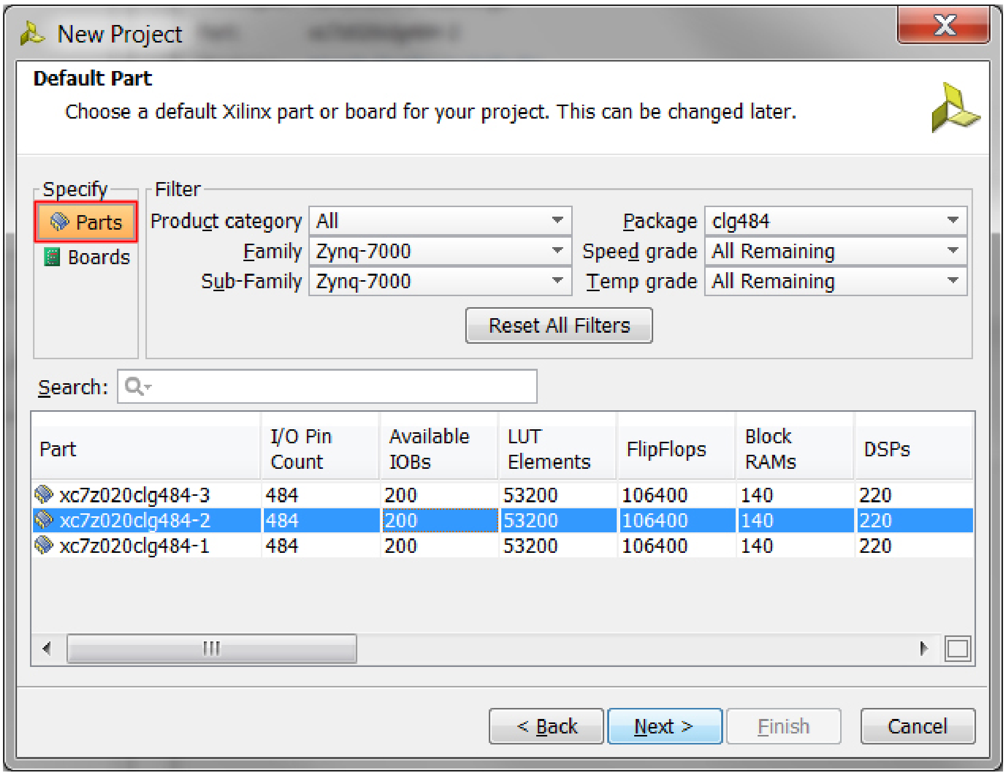


6.) ... as well as no constraints are specified:

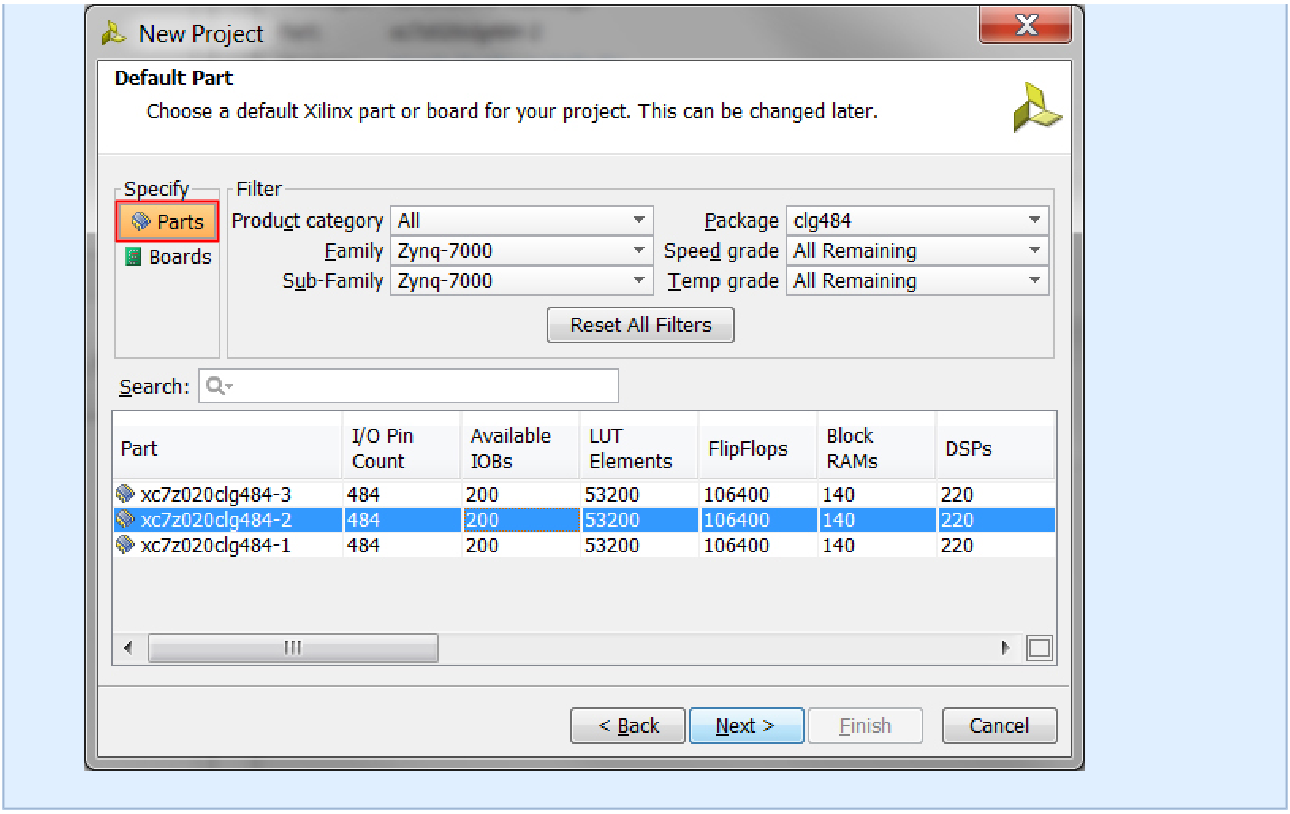
Copyright © 2014 Trenz Electronic GmbH



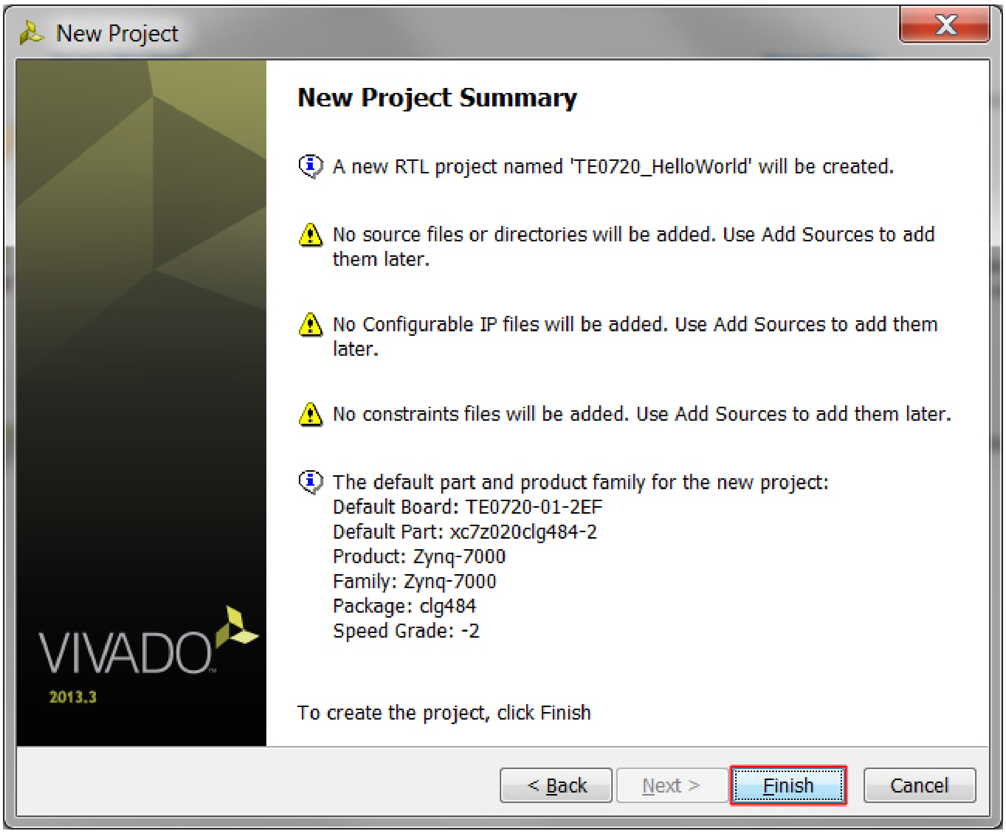
7.) As long as the Vivado patch has not been released yet (see our notice above), please select the on-board TE0720 Zynq FPGA device (e.g., xc7z020clg484-2):



As soon as our custom board "TE0720-01-2EF" will be supported by Xilinx Vivado, it can be chosen in the following way:

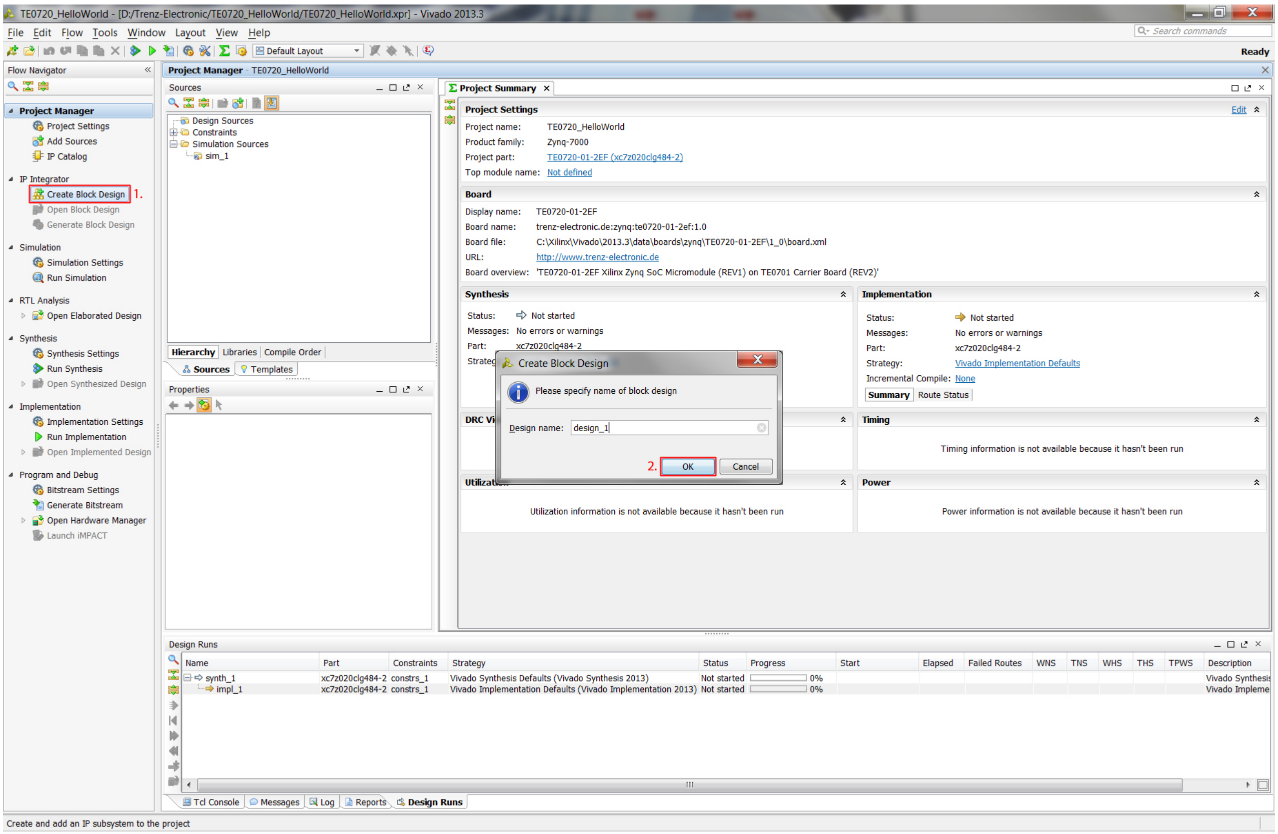


8.) Finally, create the previously parametrized RTL project:

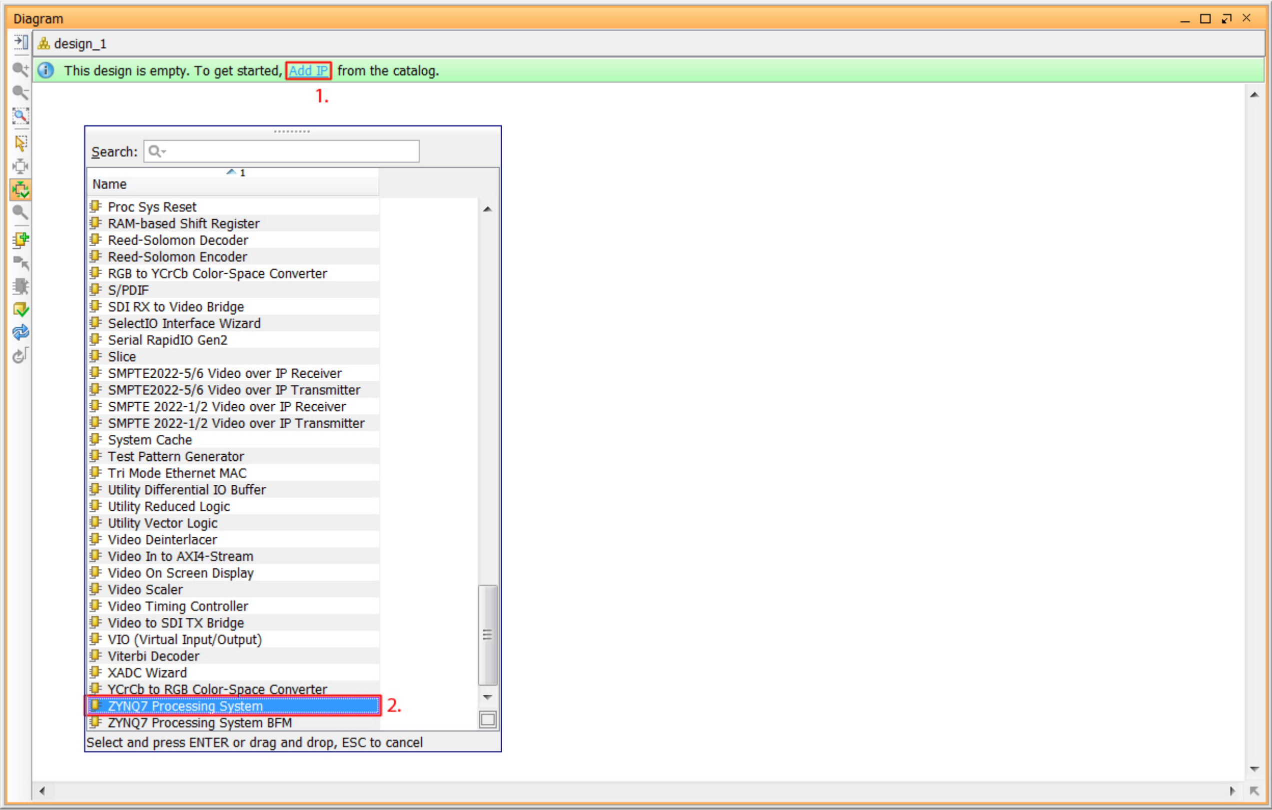


Creating Vivado Block Design (IP Integrator)

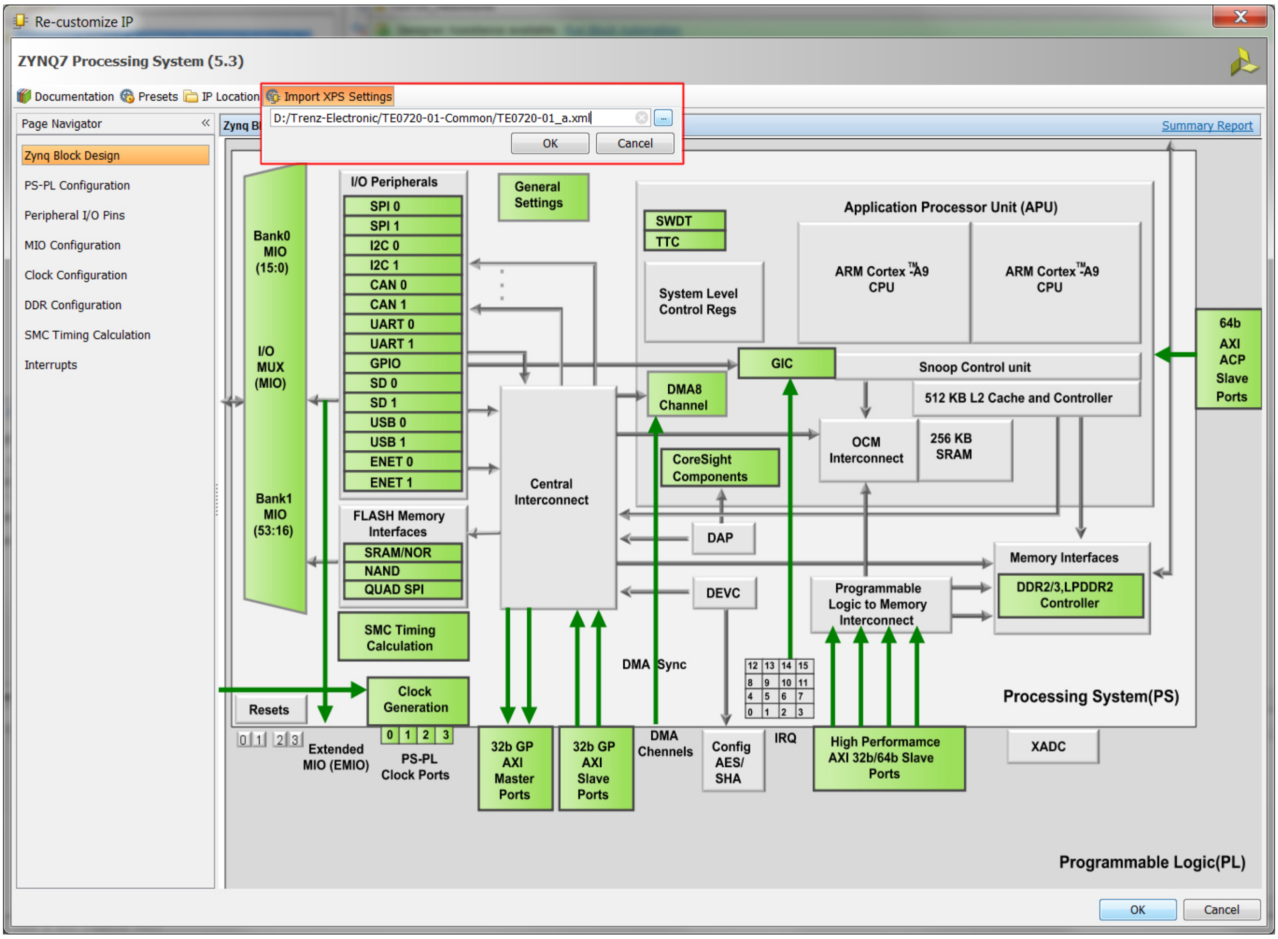
1.) Create a new block design and name it. Here, we choose the default name "design_1":



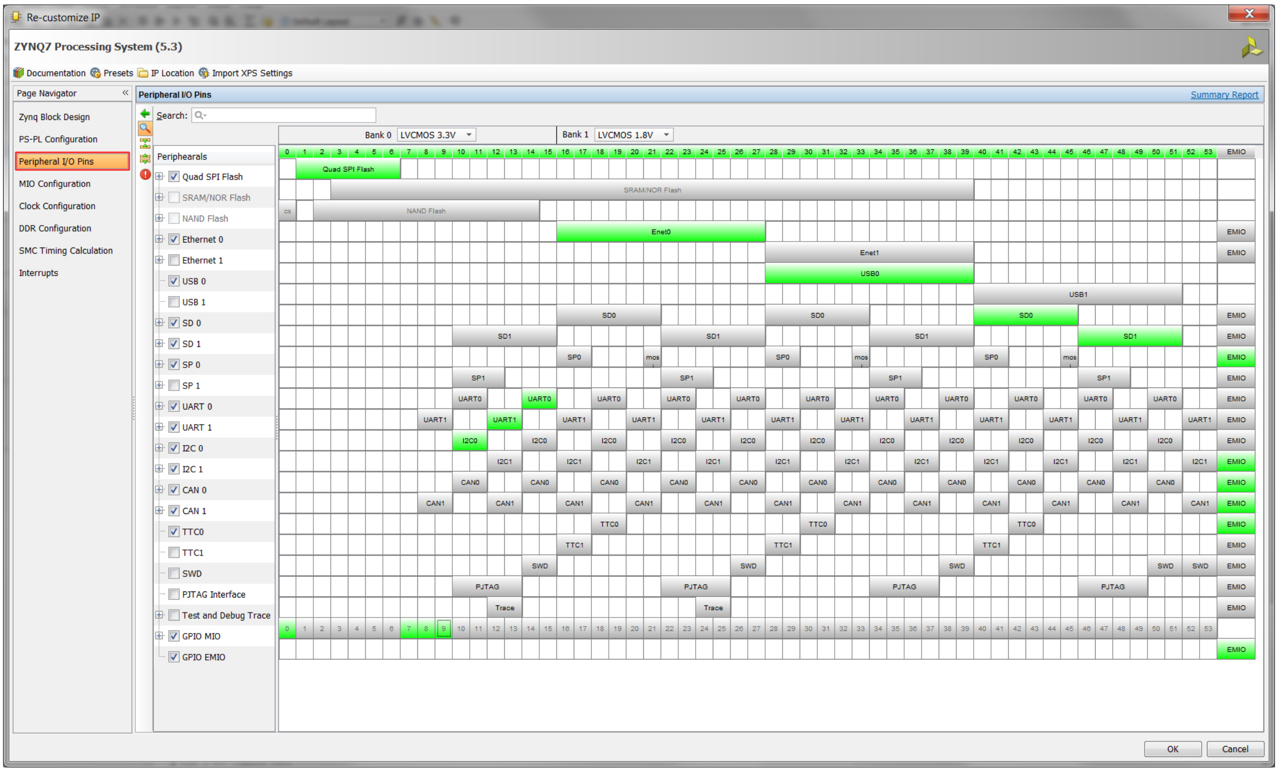
2.) Continue to add new IP (1) from the catalog and select ZYNQ7 Processing System (2):



3.) In the same "Diagram" window a "ZYNQ7 Processing System" block symbol is now visible. After a double click on it, the following "Re-customize IP" dialog will appear, where you click on "Import XPS Settings" and choose the file "TE0720-01_a.xml", which can be found in the archive TE0720-01-Common.zip on the Trenz-Electronic Homepage (see *reference designs* in the download area of the TE0720-GigaZee):

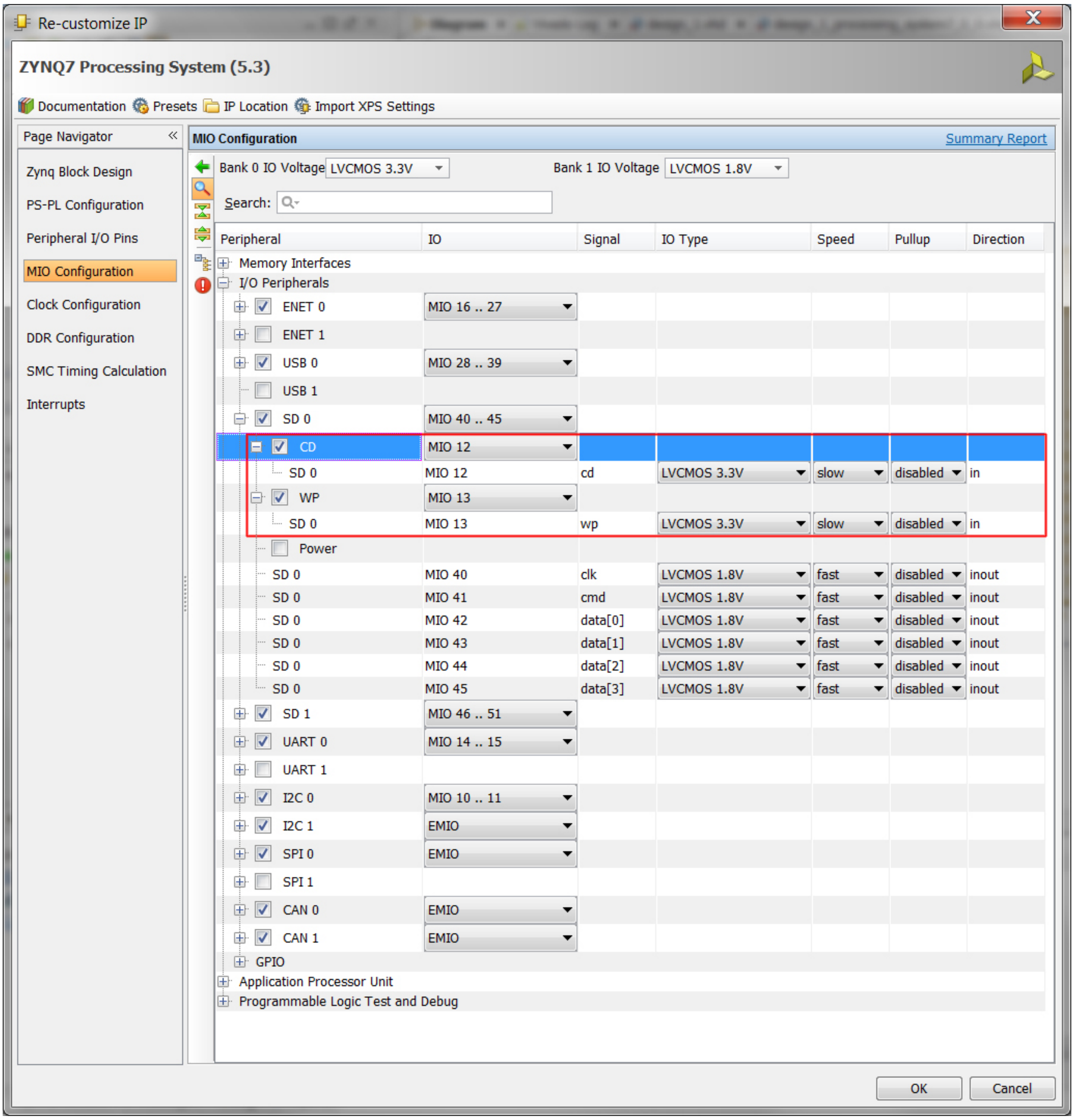


4.) After successful import, the dedicated I/O pin assignments of the enabled peripherals can be checked...



5.) ... (optional) when you would like check by an application on the Zynq FPGA if a SD card is inserted (CD=0) or if its write protection (WP=1) is enabled, then you must assign the corresponding two signals to

MIO pins. Because almost all MIO pins have been already assigned (and MIO7 to MIO9 are not routed to the TE0701's CPLD), a possible solution is to deactivate the second UART1 (just by clicking on green colored "UART1" box, which can be found in the "Peripheral I/O Pins" page; see above) and re-assigned the now available MIO12 and MIO13 pins on the "MIO Configuration" page to the CD and WP pins of the SD 0 port:



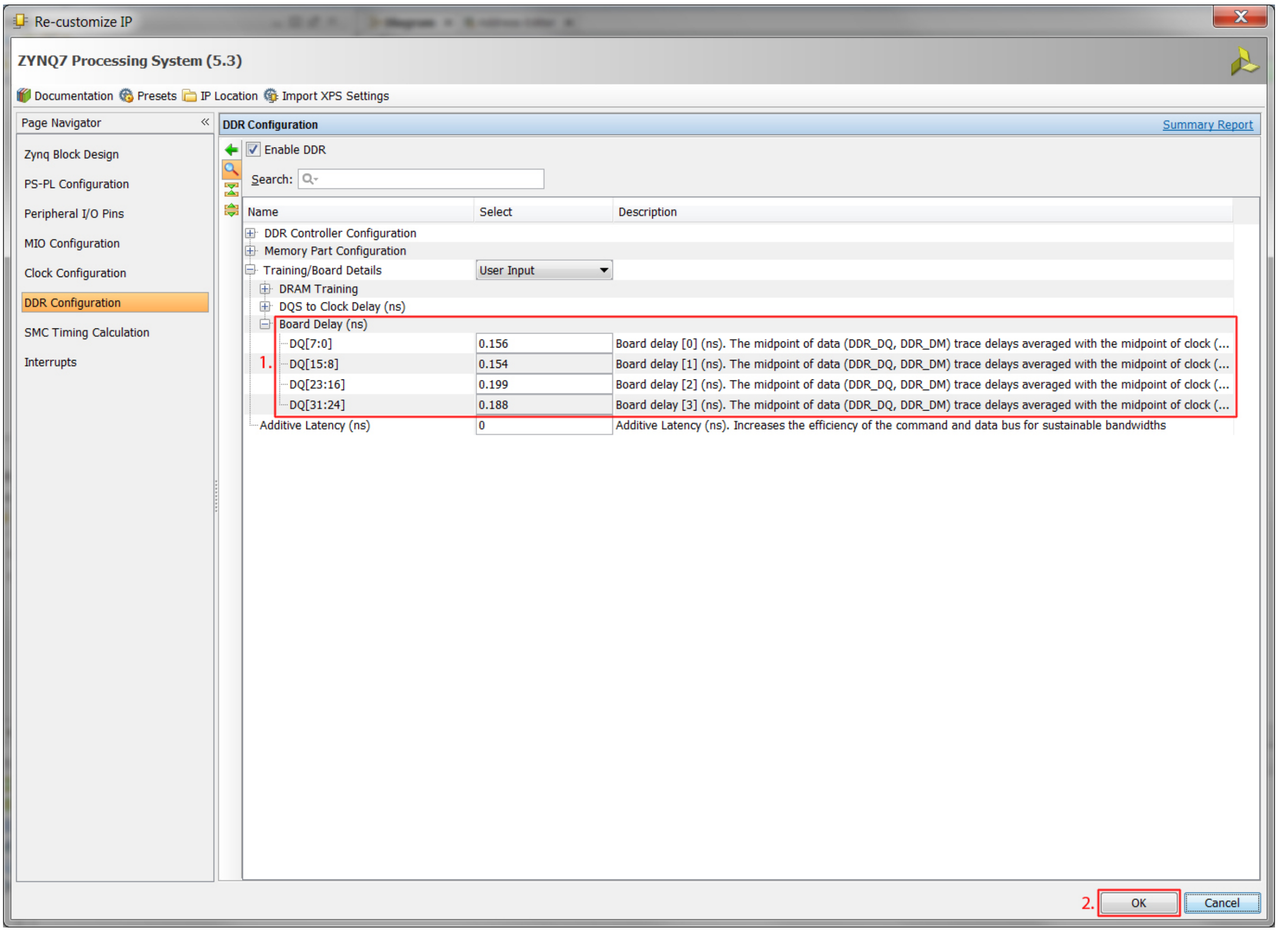
The MIO pins may be preferred towards the EMIO interface because for the latter a *programmable logic* (PL) bitfile must always be provided. Moreover, the EMIO interface is not available for the *First Stage Boot Loader* (FSBL) because at this stage, the PL has not been configured yet. Of course, in this case the TE0701 CPLD must internally connect the SD_DETECT and SD_WP signals to the corresponding signals, e.g., MIO12 and MIO13, respectively. Alternatively, the values

Copyright © 2014 Trenz Electronic GmbH

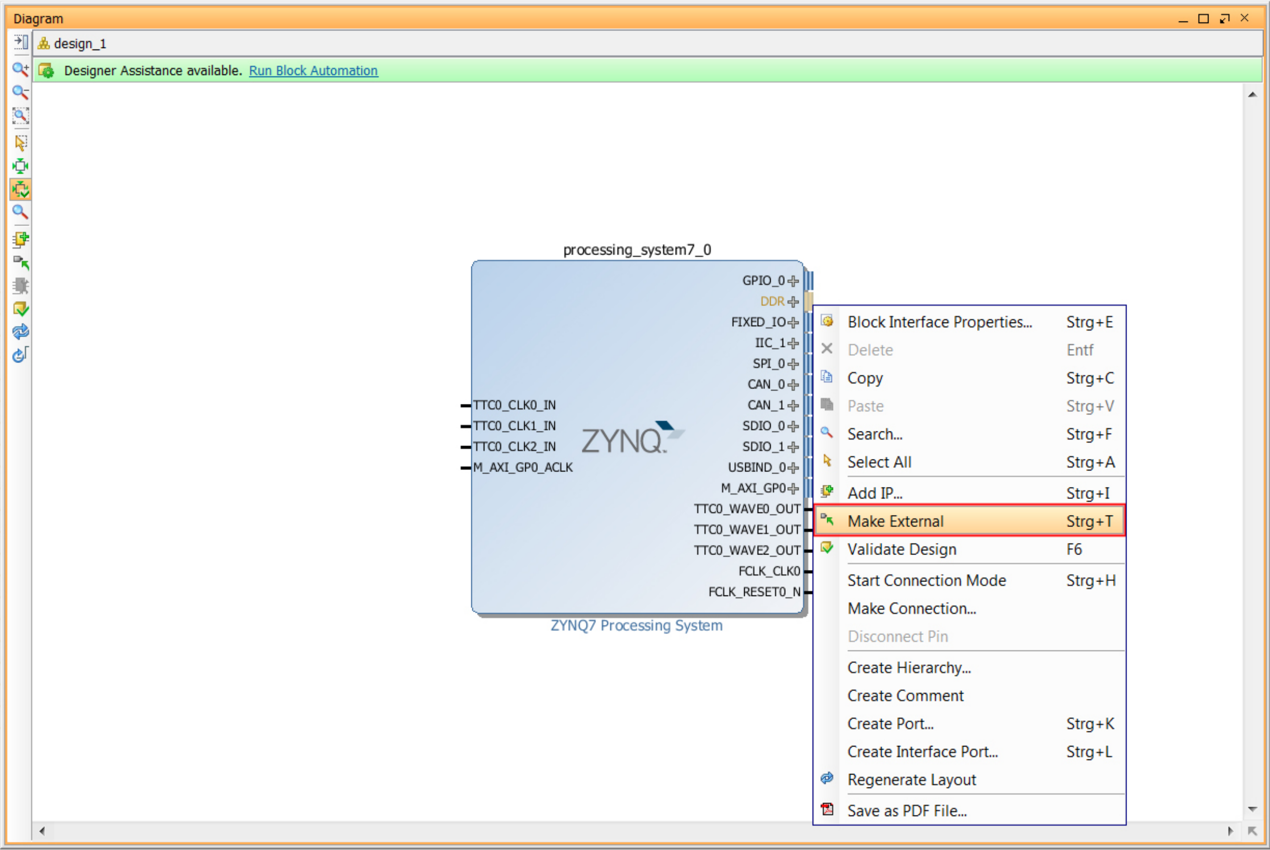
of SD_DETECT and SD_WP can also be read by the Xilinx Zynq after booting up via the on-board I2C bus (see Carrier Boards for TE0720 | Reading I2C-to-GPIO Status Register on TE0701 CPLD for more details).

However, the latest solution will not work for the device manager in Linux. Hence, be aware when tying up the corresponding EMIO pins statically to ground that the SD card WILL NOT be write-protected in dependence of the mechanical switch on the SD card anymore as the user might expect!

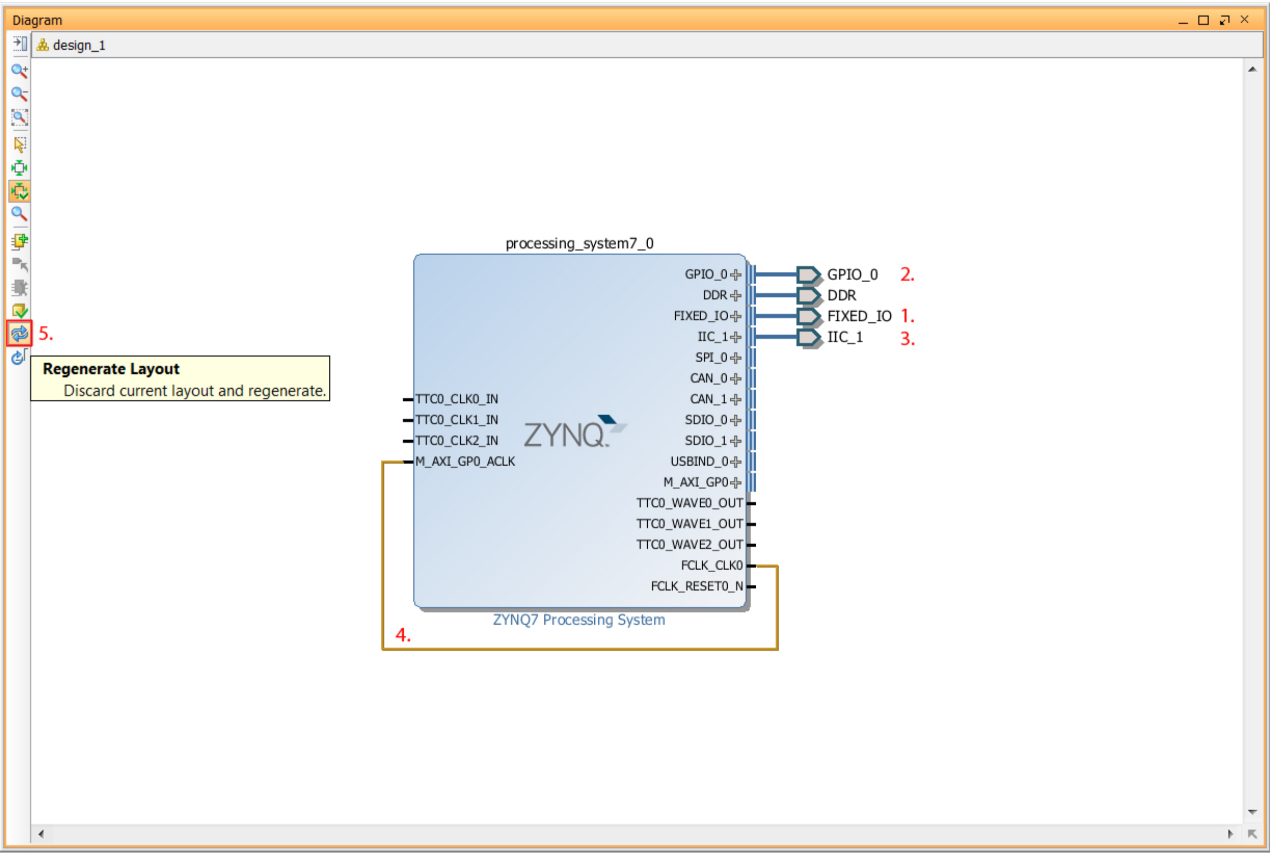
6.) ... also the DDR Configuration should have the following new timing settings (instead of 0.0 ns each):



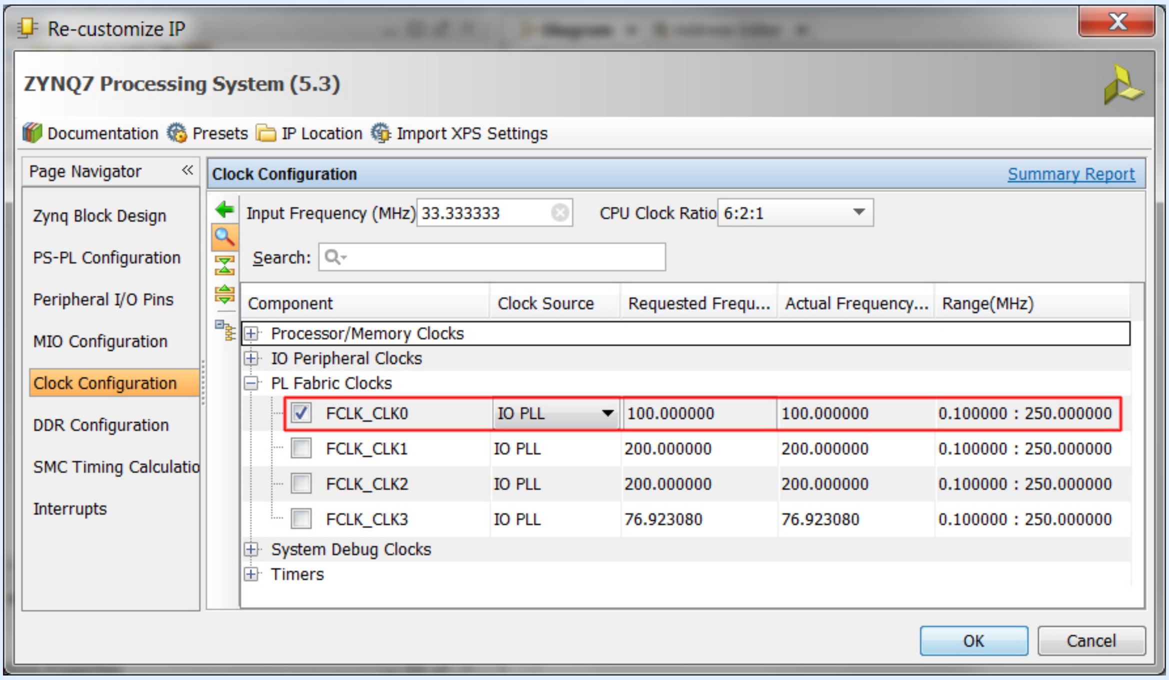
7.) Back in the "Diagram" window right click on the IO port of the DDR interface and select in the context menu "Make External"...



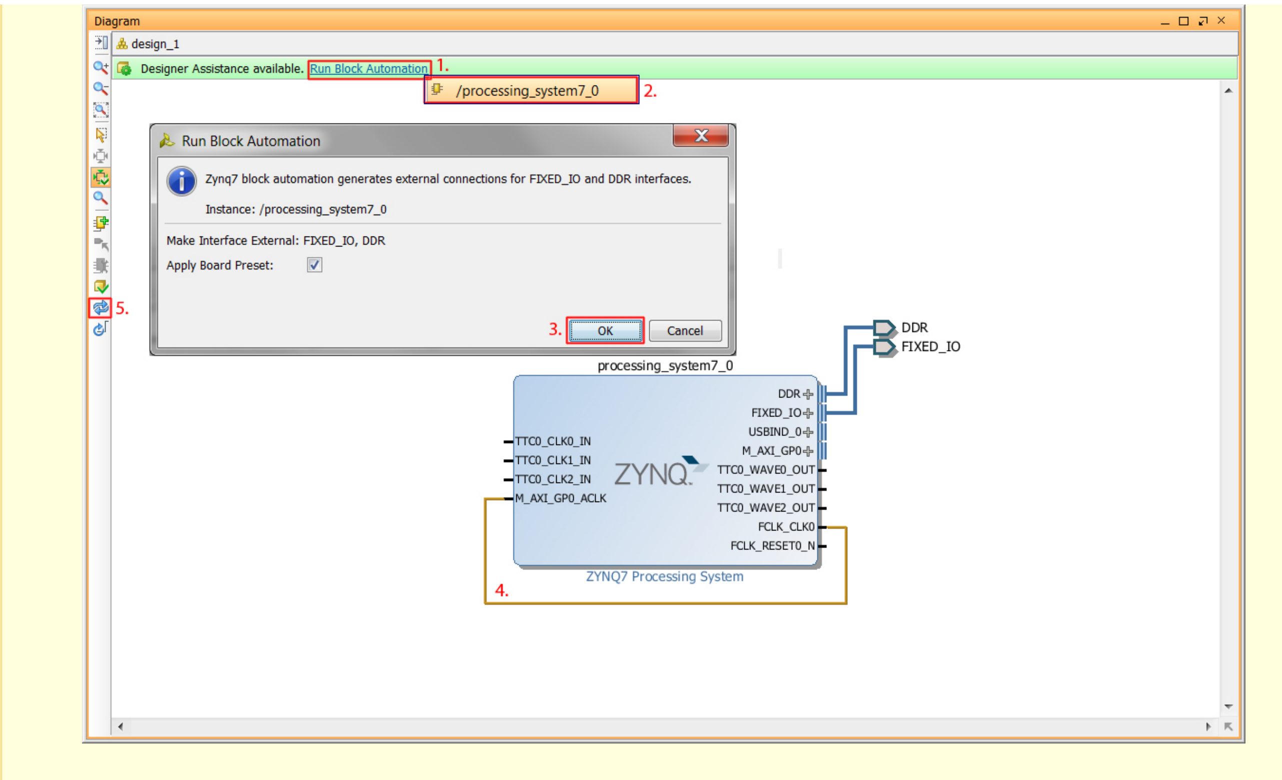
8.) ... repeat step 6 for the ports FIXED_IO (1), GPIO_0 (2), and IIC_1 (3). Then, manually connect the ports "FCLK_CLK0" and "M_AXI_GP0_ACLK" to each other (4) and regenerate layout (5):



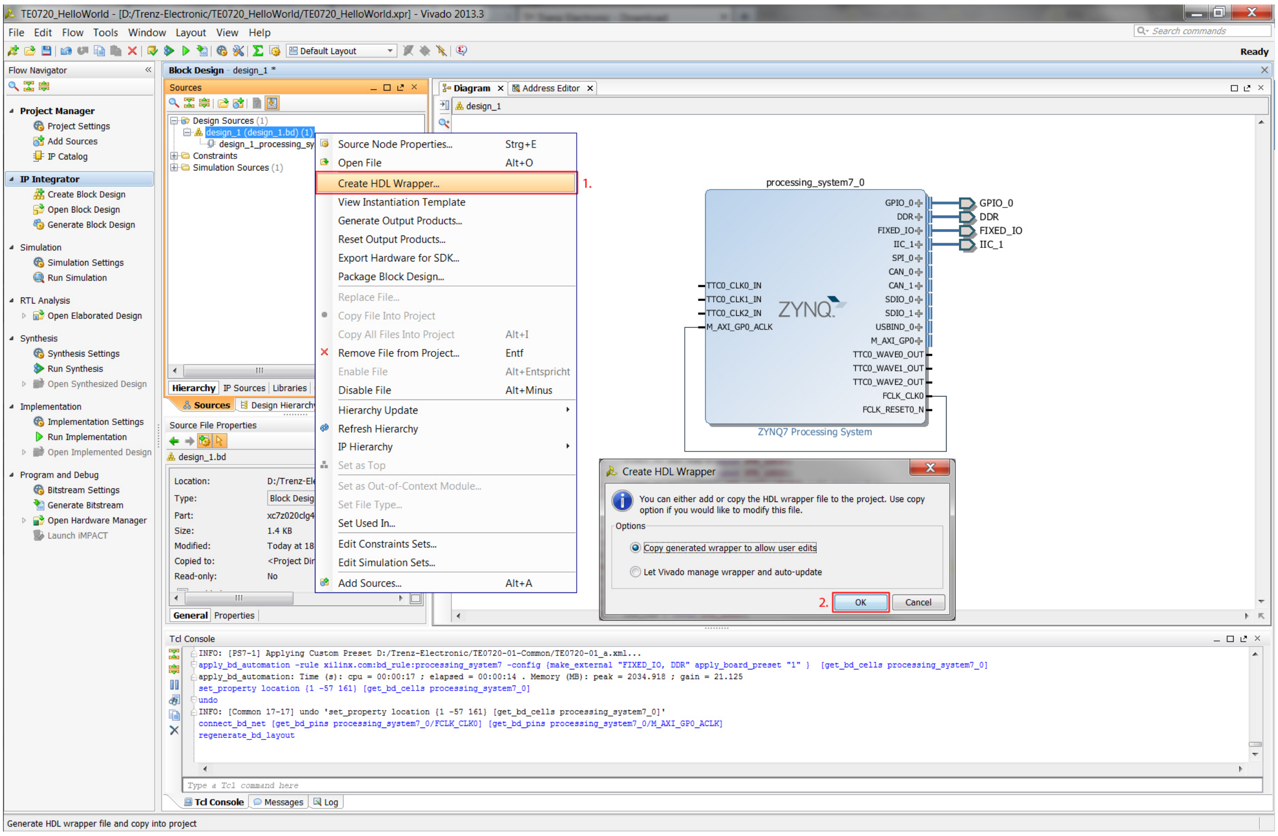
The signal path from "FCLK_CLK0" to "M_AXI_GP0_ACLK" is set because the PS provides four fully programmable clocks (FCLK_CLK*n*; as shown in the following figure) to the PL and it is a common practice to exploit one of these clocks (namely, "FCLK_CLK0") to source the AXI bus clock (in this case with 100 MHz):



In contrast to Vivado 2003.2, the "Run Block Automation" function will remove all peripherals that are not connected internally or externally in the "design_1" diagram as shown in the following figure. Hence, by generating the corresponding HDL wrapper (see next step), all signals will also be removed automatically in the "processing_system7_0" component of the "design_1.vhd" HDL wrapper. For this reason, we recommend to use manually the "Make external" function in the context menu as it has been described previously (particularly, when you are planning to edit the corresponding HDL wrapper later as it will be also shown later in this step-by-step tutorial)!

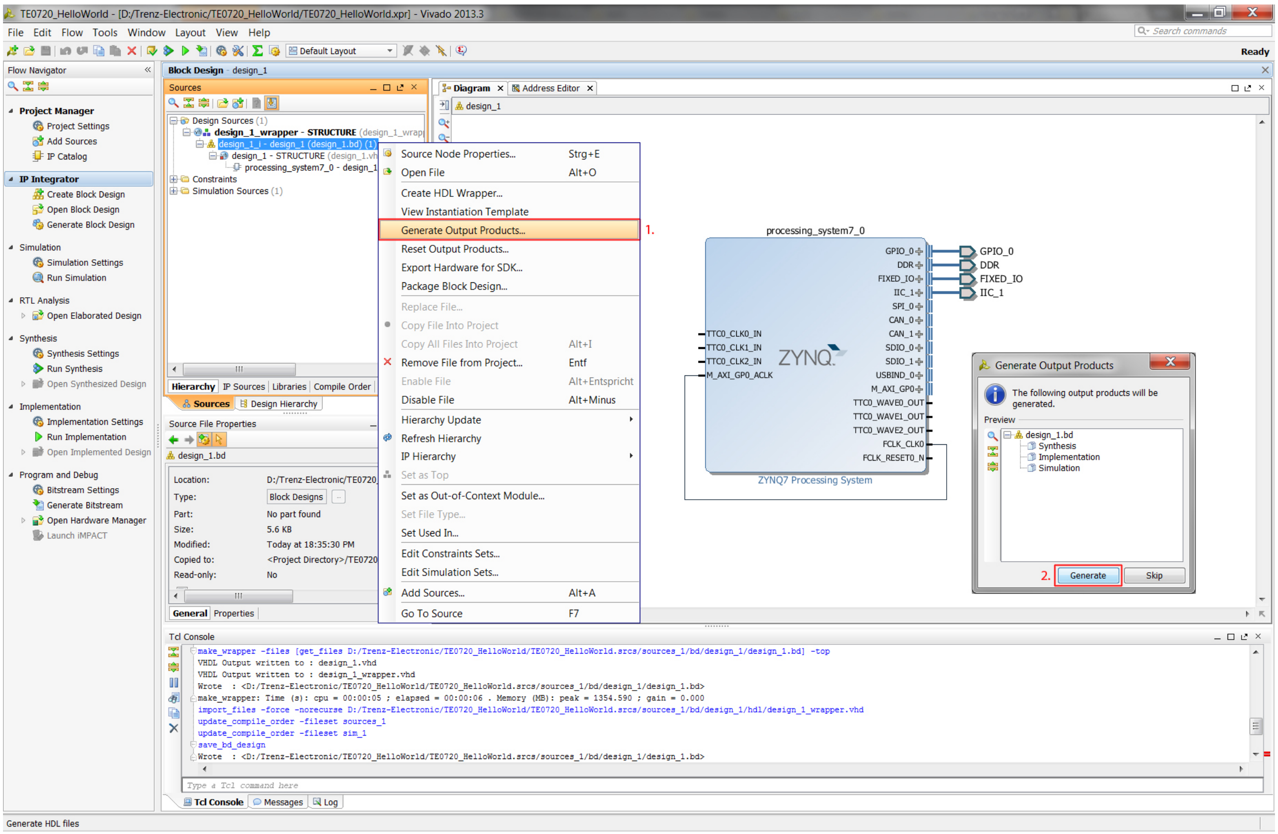


9.) Then, the corresponding HDL wrapper is created (1) by chosing the non-default option "Copy generated wrapper to allow user edit" (2):

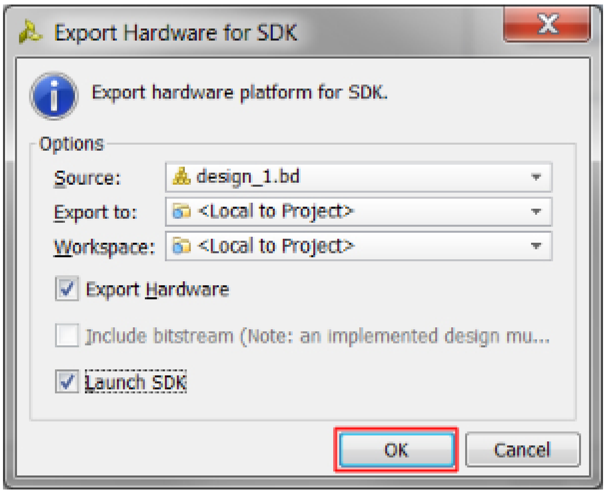


10.) Before the hardware design can be exported, the "IP Integrator" design has to be generated:

Copyright © 2014 Trenz Electronic GmbH



11.) Finally, export hardware for SDK (menu "File -> Export -> Export Hardware for SDK...") and launch SDK:



The exported source "design_1.bd" is a hardware description only and the following files are generated by the Vivado IP Integrator [Xilinx UG898: Embedded Processor Hardware Design (v2013.3), p.41]:

File	Description
system.xml	This file opens by default when you launch SDK and displays the address map of your system.
ps7_init.c	

File	Description
ps7_init.h	The ps7_init.c and ps7_init.h files contain the initialization code for the Zynq Processing System and initialization settings for DDR, clocks, PLLs, and MIOs. SDK uses these settings when initializing the processing system so applications can run on top of the processing system.
ps7_init.tcl	This is the Tcl version of the INIT file.
ps7_init.html	The INIT file describes the initialization data.

Note: See the "Xilinx UG821: Zynq-7000 All Programmable SoC Software Developers Guide" for more information about generated files.

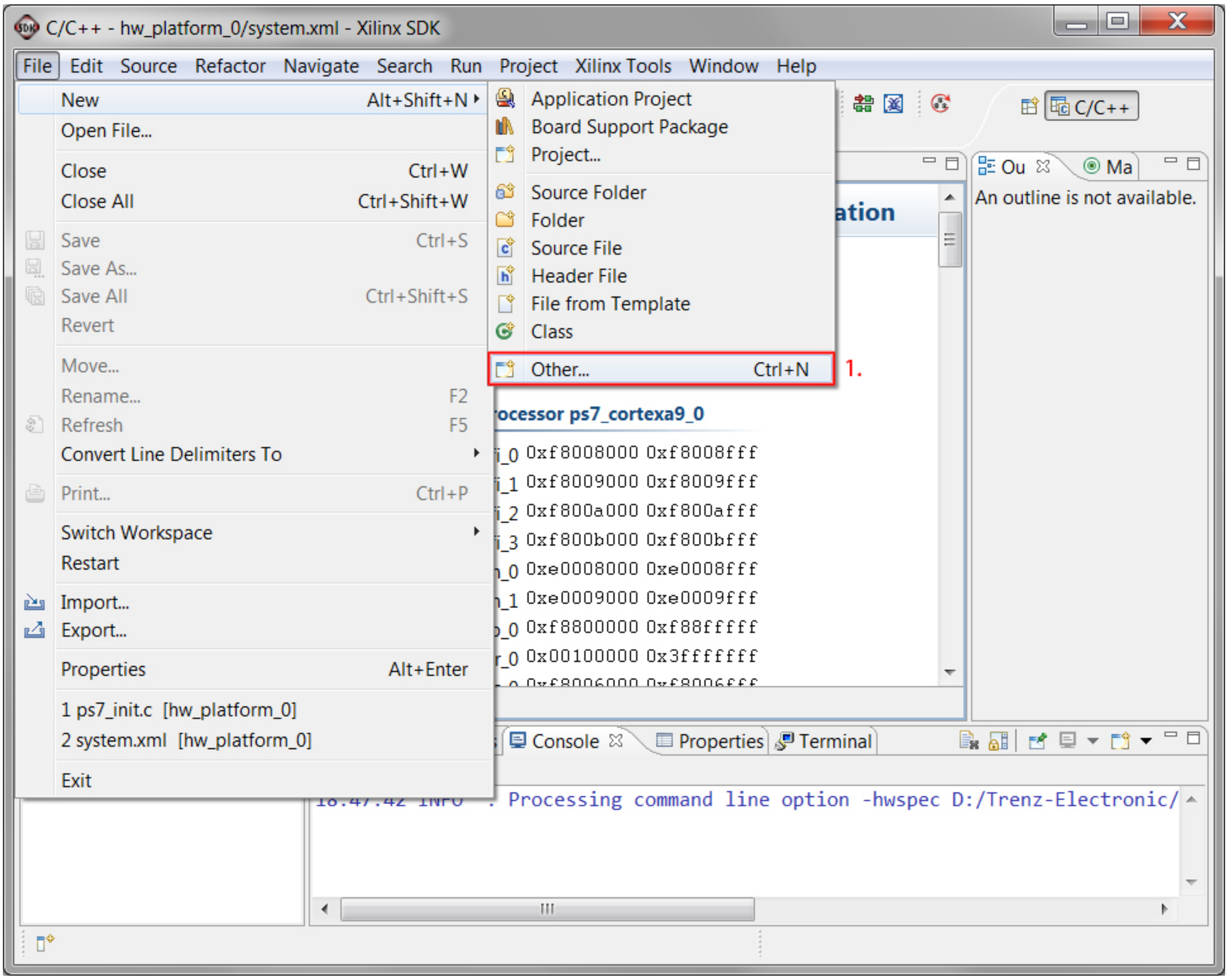
There is no bitstream required for the follwing simple software "Hello World" application to be executed on the Xilinx Zynq *Processing System* (PS), since the ARM Cortex A9 dual core is already present in the FPGA. Basic initialization of this system to run a simple application is done by the device initialization TCL script.

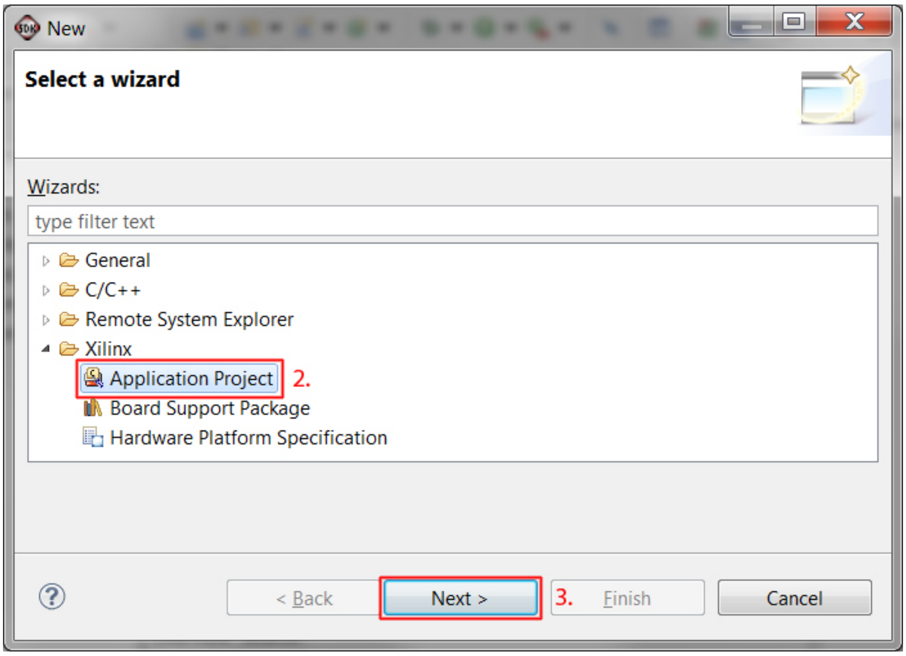
Only if additionally a Xilinx Zynq *Programmble Logic* (PL) design is required to be configured (e.g., to access the TE0701 on-board HDMI interface), the corresponding bitfile must also be exported to SDK. **Note**: The corresponding option can be selected only when the implemented design has been opened in Vivado! For more details on the synthesis and implementation process, please refer to the later section "Vivado Flow | Hardware Synthesis & Implementation".

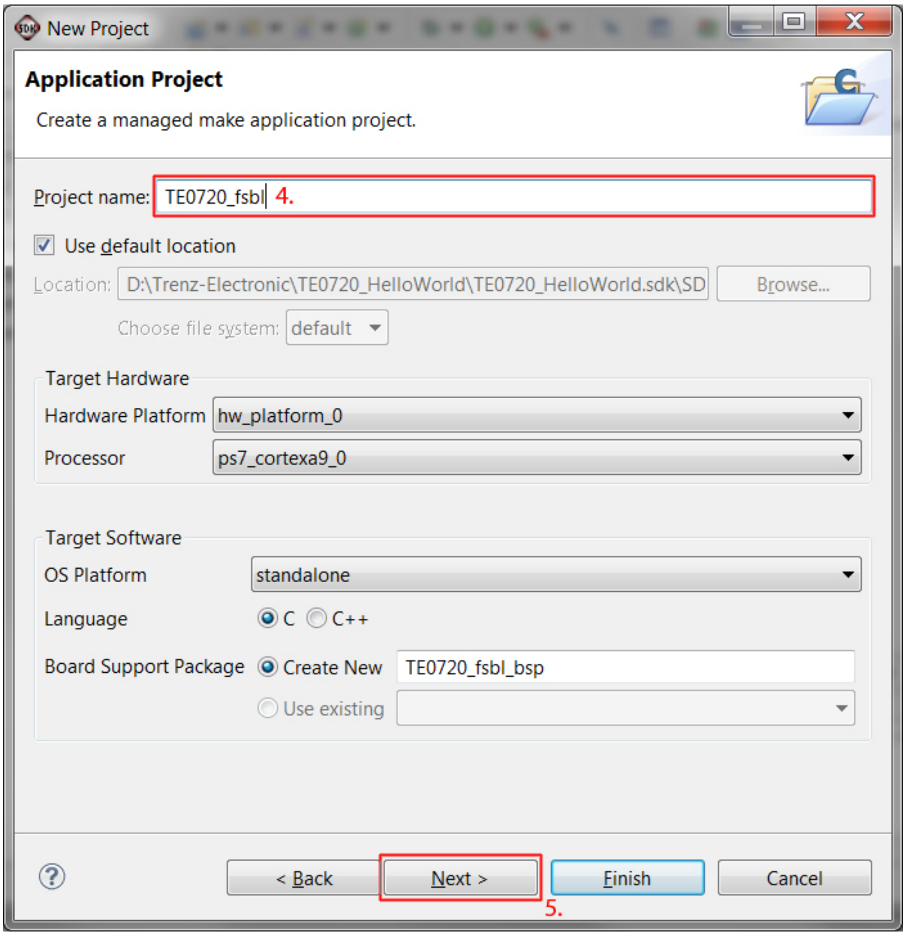
Software Implementation: Create First Stage Boot Loader (FSBL) and "Hello World" application project in SDK

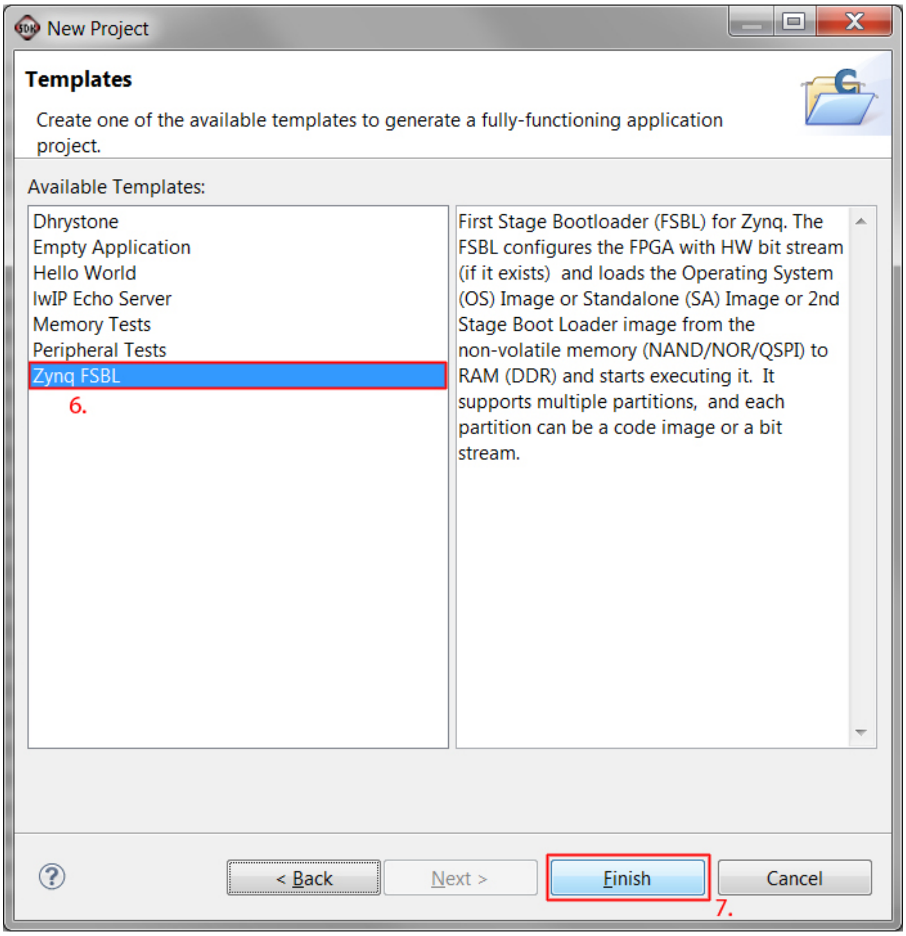
1.) After the SDK GUI has been launched, first of all, a new Xilinx Application Project is created. More precisely, a Zynq *First Stage Boot Loader* (FSBL) project:

Copyright © 2014 Trenz Electronic GmbH

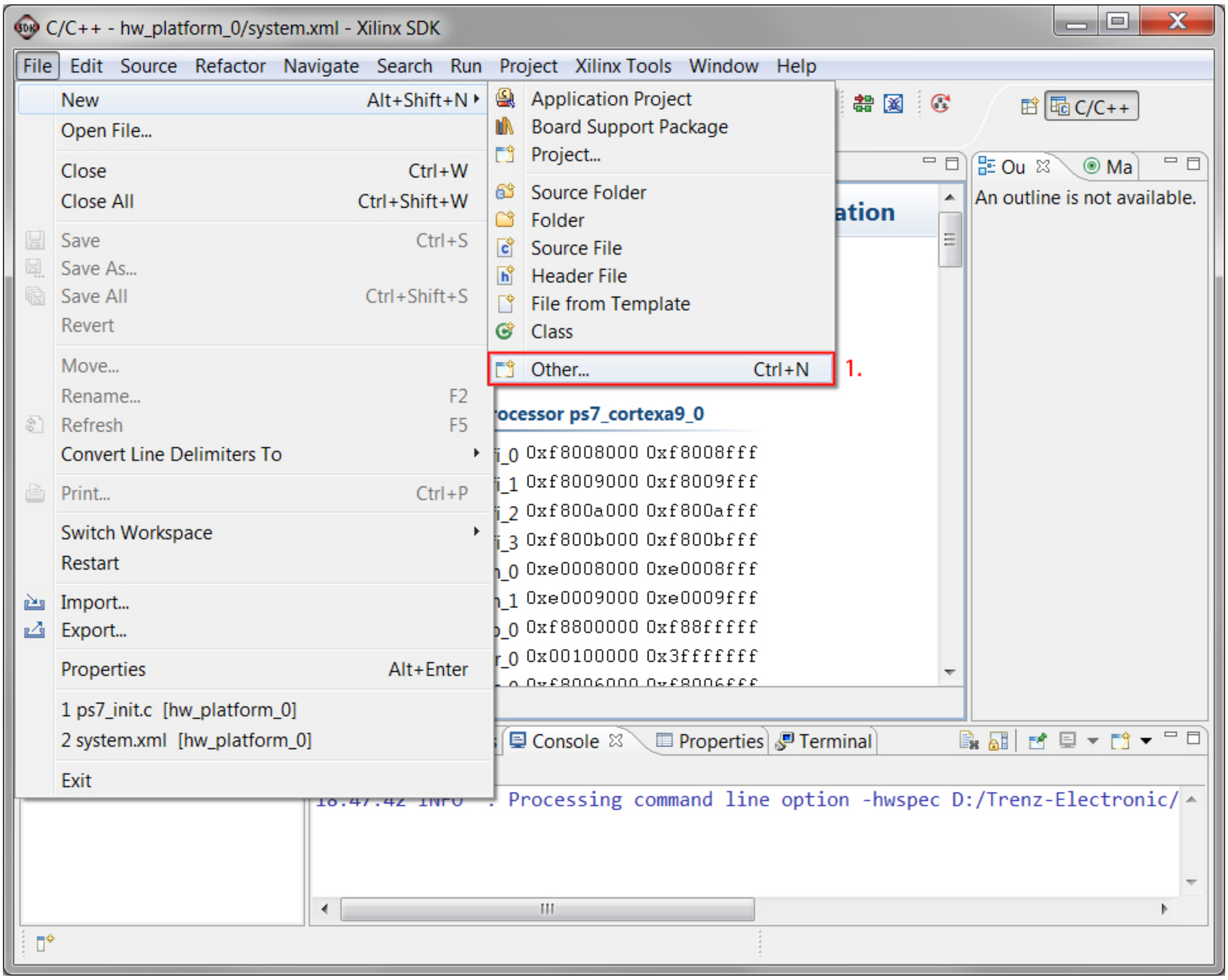

C/C++ - hw_platform_0/system.xml - Xilinx SDK
File Edit Source Refactor Navigate Search Run Project Xilinx Tools Window Help
New Alt+Shift+N
Open File...
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Switch Workspace
Restart
Import...
Export...
Properties Alt+Enter
1 ps7_init.c [hw_platform_0]
2 system.xml [hw_platform_0]
Exit
Application Project
Board Support Package
Project...
Source Folder
Folder
Source File
Header File
File from Template
Class
Other... Ctrl+N
1.
C/C++
An outline is not available.
Console
Properties
Terminal
Processing command line option -hwspec D:/Trenz-Electronic/

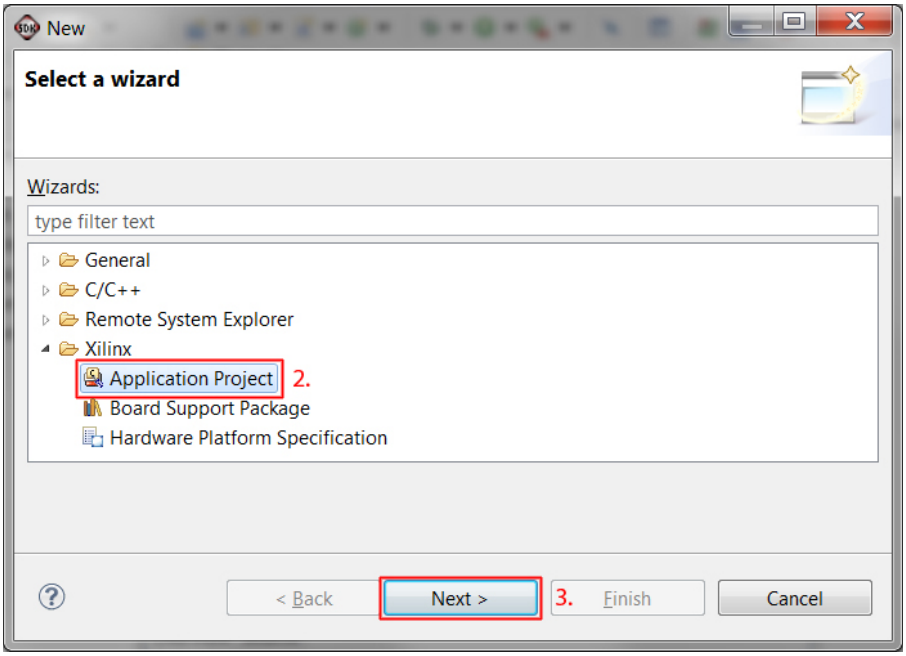

New
Select a wizard
Wizards:
type filter text
General
C/C++
Remote System Explorer
Xilinx
Application Project
2.
Board Support Package
Hardware Platform Specification
< Back
Next >
3.
Finish
Cancel

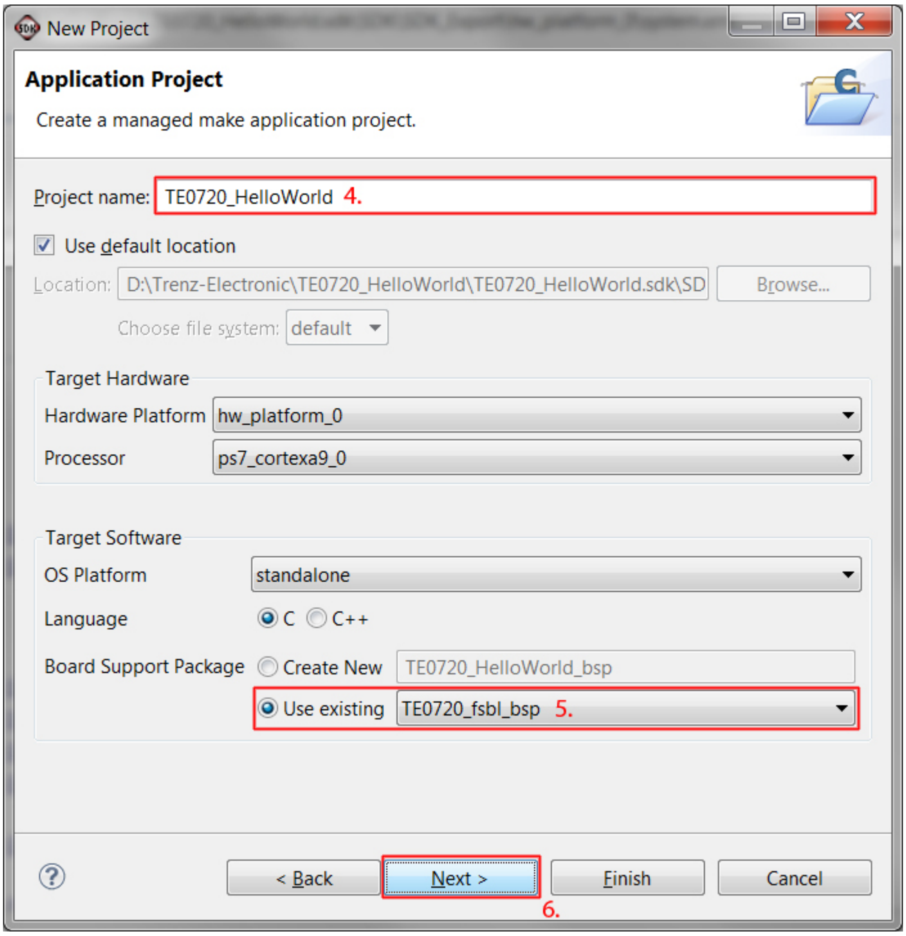

New Project
Application Project
Create a managed make application project.
Project name: TE0720_fsbl
4.
Use default location
Location: D:\Trenz-Electronic\TE0720_HelloWorld\TE0720_HelloWorld.sdk\SD
Browse...
Choose file system: default
Target Hardware
Hardware Platform hw_platform_0
Processor ps7_cortexa9_0
Target Software
OS Platform standalone
Language C C++
Board Support Package Create New TE0720_fsbl_bsp
Use existing
< Back
Next >
5.
Finish
Cancel

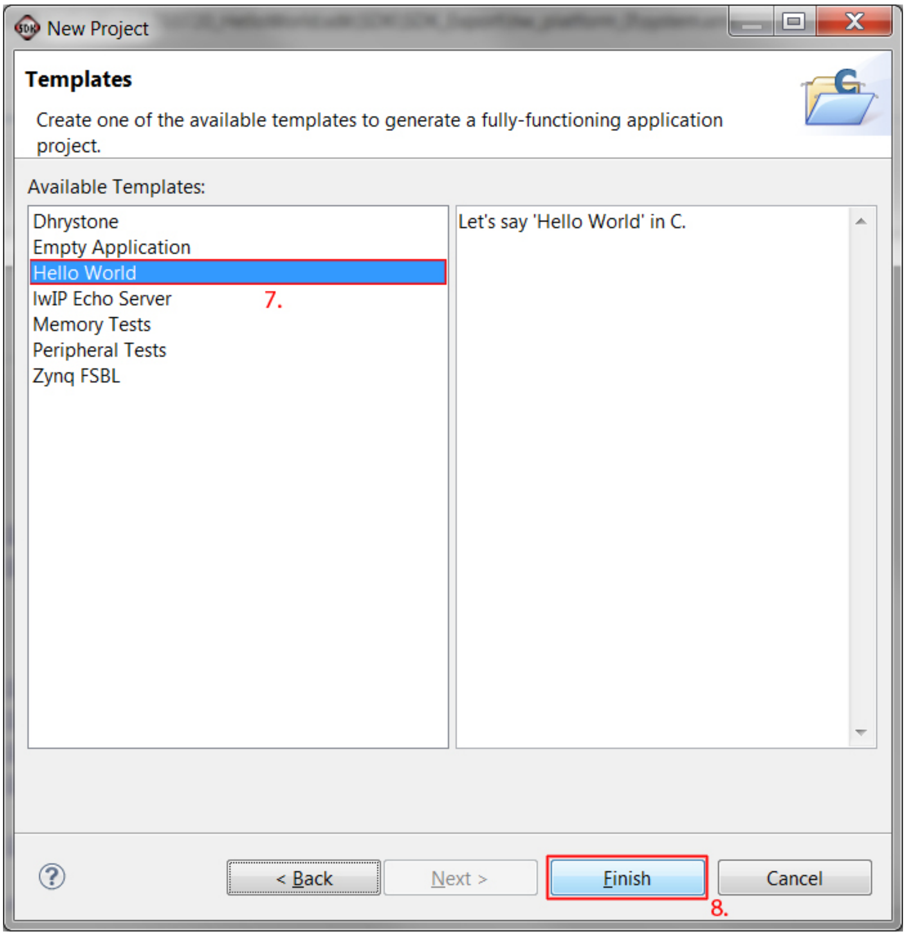


2.) Secondly, a application project is created, which will be associated with the previously generated Zynq FSBL:

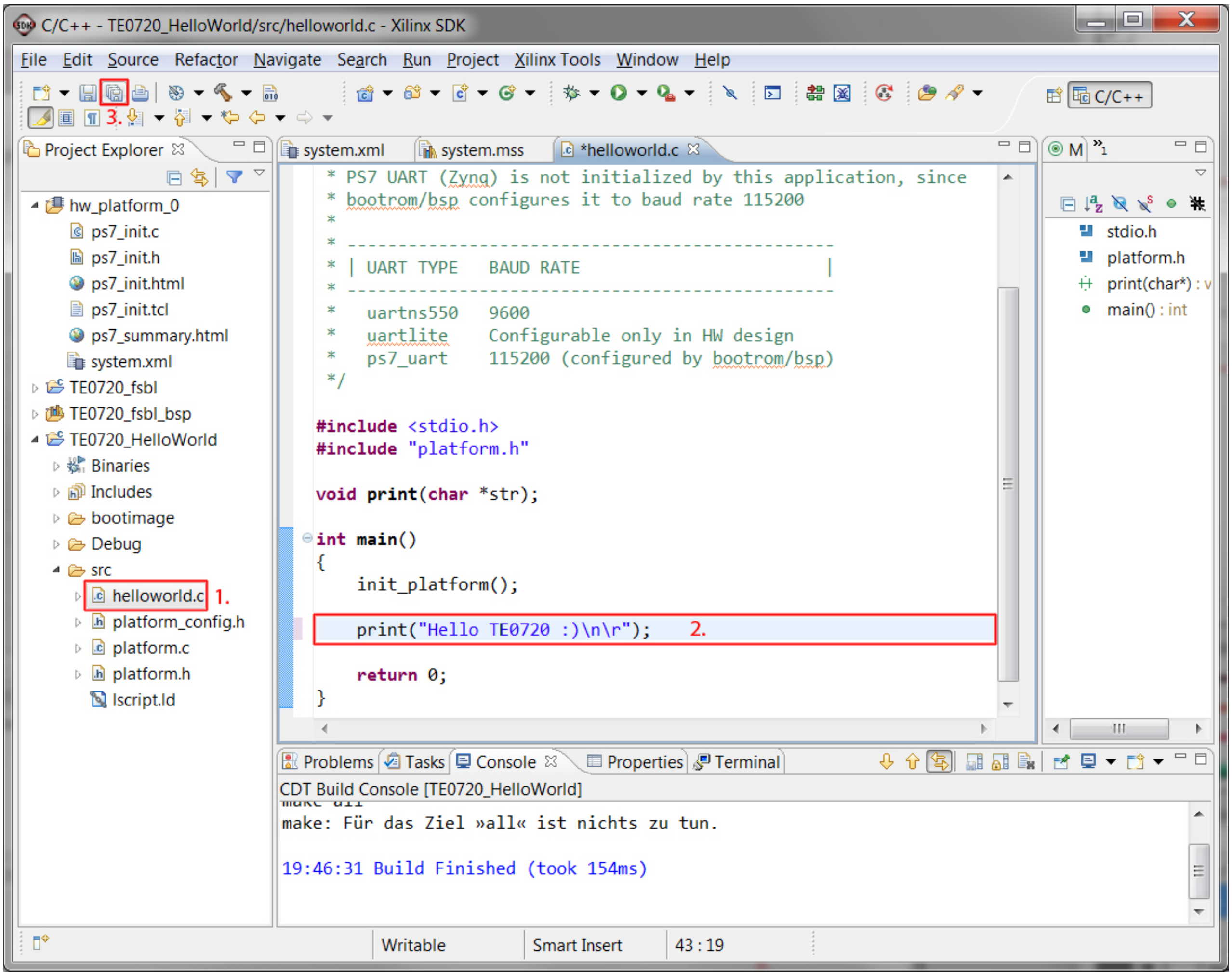

C/C++ - hw_platform_0/system.xml - Xilinx SDK
File Edit Source Refactor Navigate Search Run Project Xilinx Tools Window Help
New Alt+Shift+N
Open File...
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Switch Workspace
Restart
Import...
Export...
Properties Alt+Enter
1 ps7_init.c [hw_platform_0]
2 system.xml [hw_platform_0]
Exit
Application Project
Board Support Package
Project...
Source Folder
Folder
Source File
Header File
File from Template
Class
Other... Ctrl+N
1.
C/C++
Ou
Ma
An outline is not available.
ocessor ps7_cortexa9_0
0xf8008000 0xf8008fff
0xf8009000 0xf8009fff
0xf800a000 0xf800afff
0xf800b000 0xf800bfff
0xe0008000 0xe0008fff
0xe0009000 0xe0009fff
0xf8800000 0xf88fffff
0x00100000 0x3fffffff
Console
Properties
Terminal
Processing command line option -hwspec D:/Trenz-Electronic/


New
Select a wizard
Wizards:
type filter text
General
C/C++
Remote System Explorer
Xilinx
Application Project
2.
Board Support Package
Hardware Platform Specification
< Back
Next >
3.
Finish
Cancel

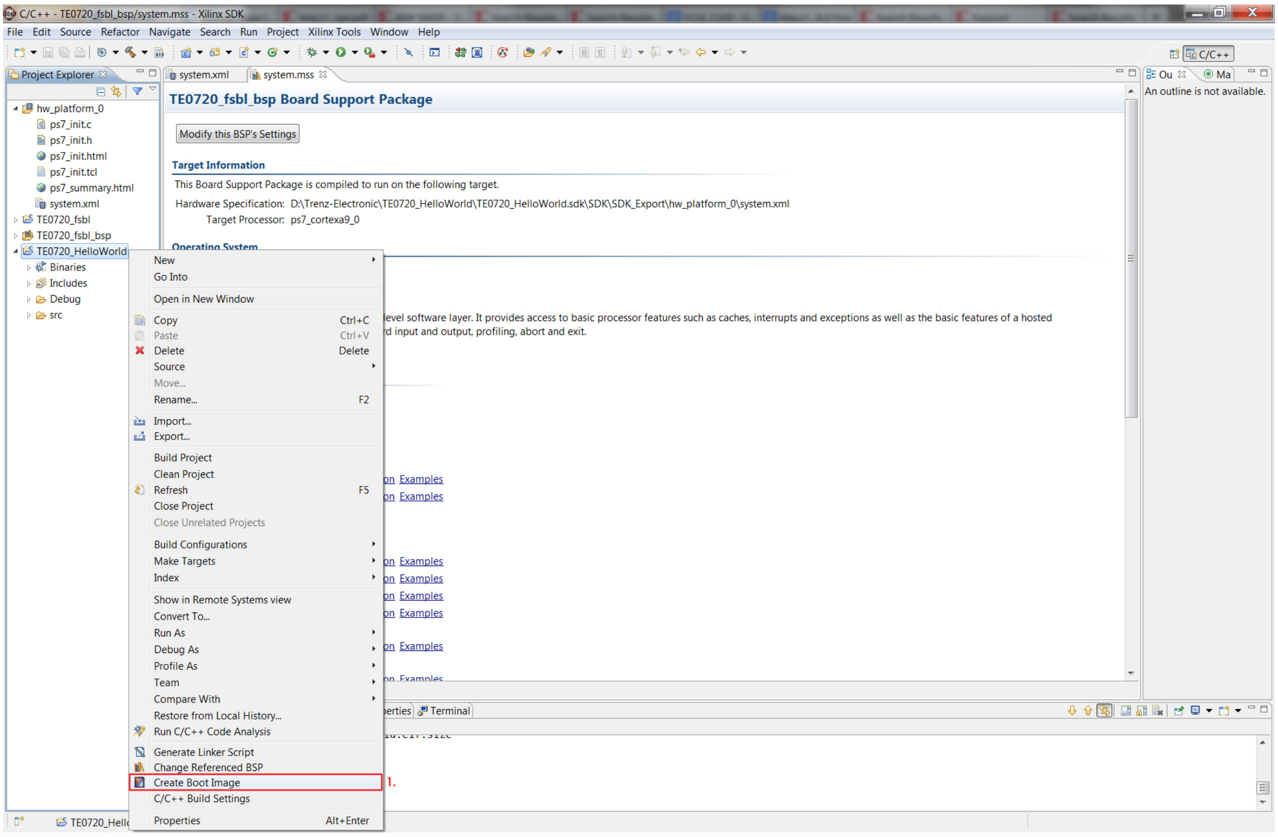

New Project
Application Project
Create a managed make application project.
Project name: TE0720_HelloWorld 4.
Use default location
Location: D:\Trenz-Electronic\TE0720_HelloWorld\TE0720_HelloWorld.sdk\SD
Browse...
Choose file system: default
Target Hardware
Hardware Platform hw_platform_0
Processor ps7_cortexa9_0
Target Software
OS Platform standalone
Language C C++
Board Support Package Create New TE0720_HelloWorld_bsp
Use existing TE0720_fsbl_bsp 5.
< Back
Next >
6.
Finish
Cancel

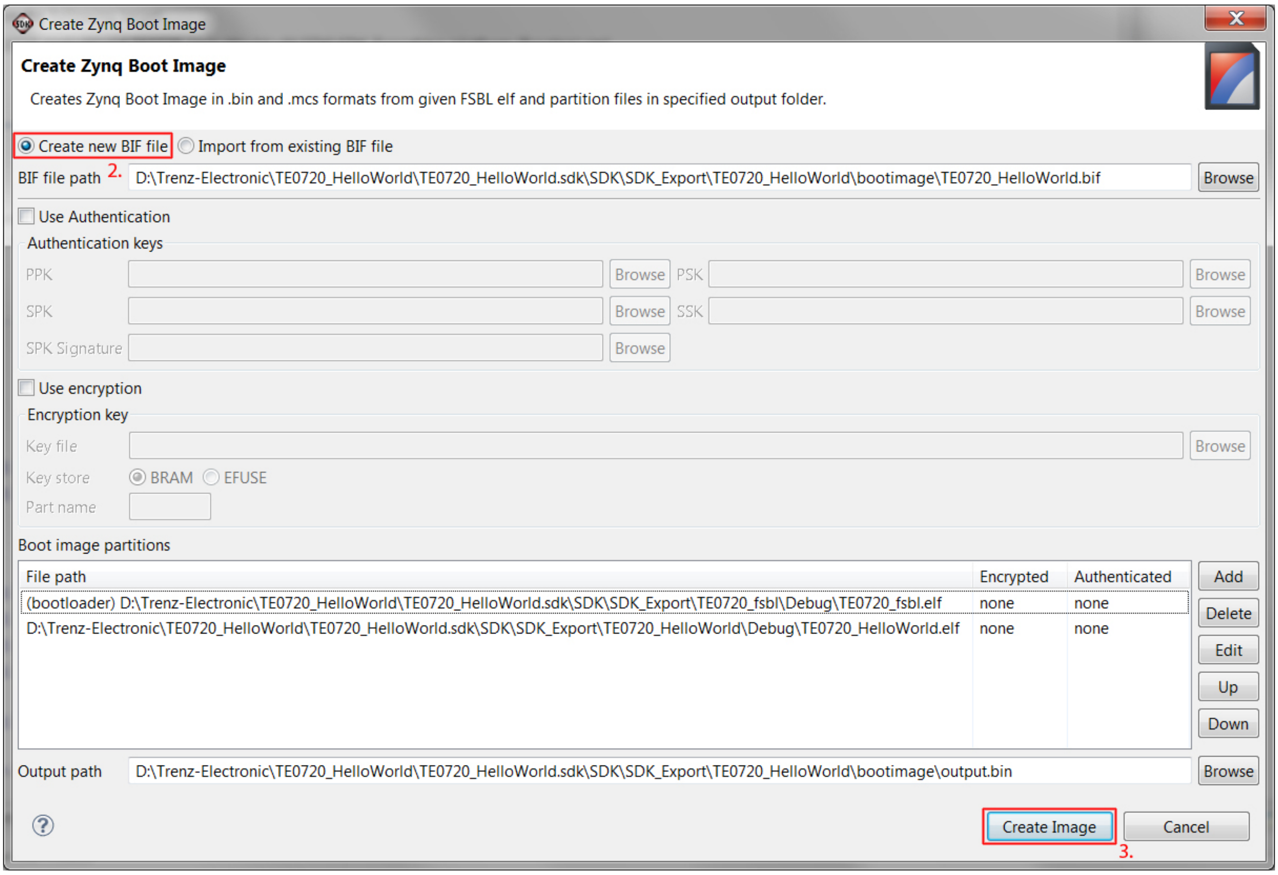


3.) Next, the "Hello World" application project can be customized (**Note**: Saving all files (3) will automatically rebuild the project):



4.) Finally, the boot image must be created:





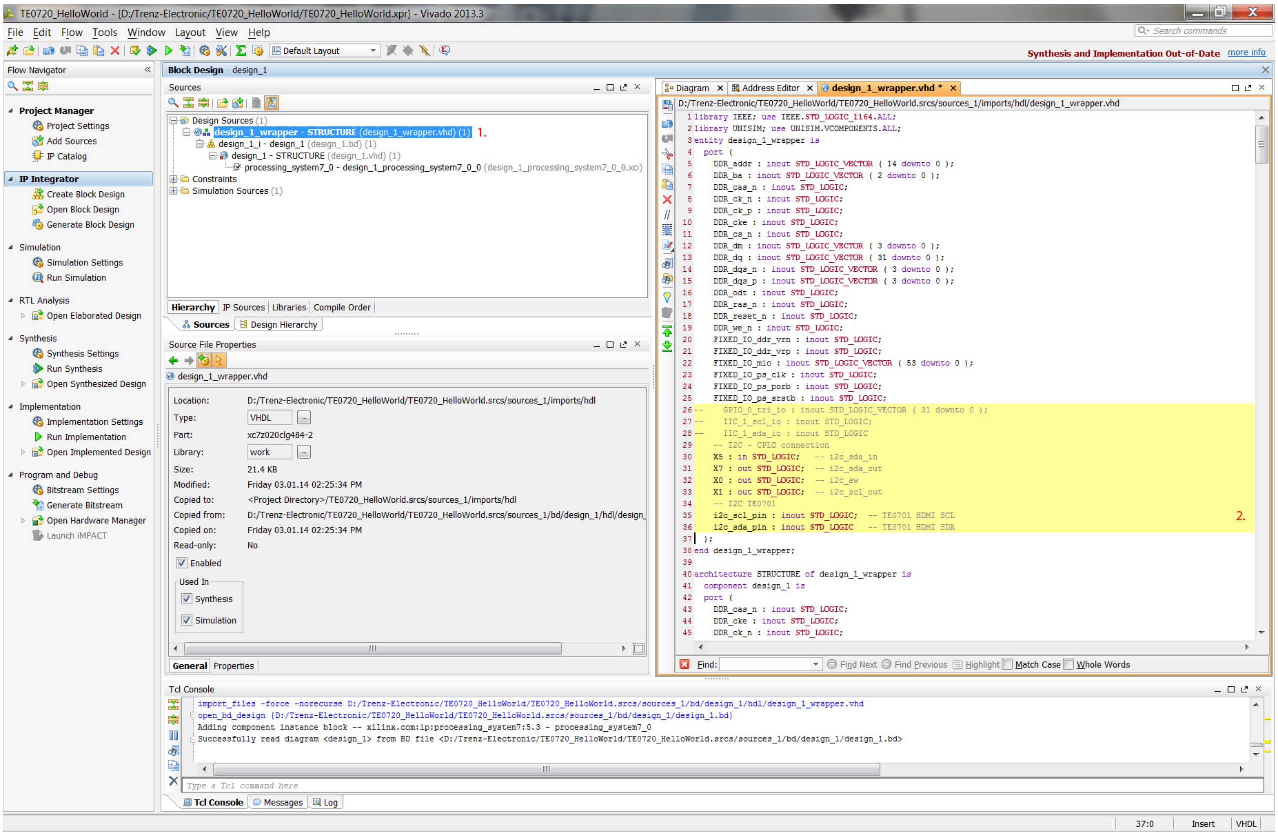
5.) ... and the newly generated boot image "output.bin" (may be renamed to "boot.bin") and copied onto a SD card, which must be formatted with a FAT32 file system.

To learn about how to set up a UART communication channel between the Xilinx Zynq FPGA (UART0) and the host PC, please consult Vivado Flow (Video and Step-by-Step Tutorial) | Debugging the "Hello World" Project.

Hardware Synthesis & Implementation

Starting from this point, we are going to create (similarly to the TE0720 GigZee Zynq SoM | Reference Projects | Base PlanAhead Project, however, the implementation is achieved a little bit differently in Vivado) a custom PS wrapper which communicates with the PS via the EMIO interface and pass I2C signals and one GPIO pin to the on-board CPLD (TE0720 User Manual | System Management Controller). Inside the CPLD, an I2C switch controlled by the GPIO pin X0 is implemented.

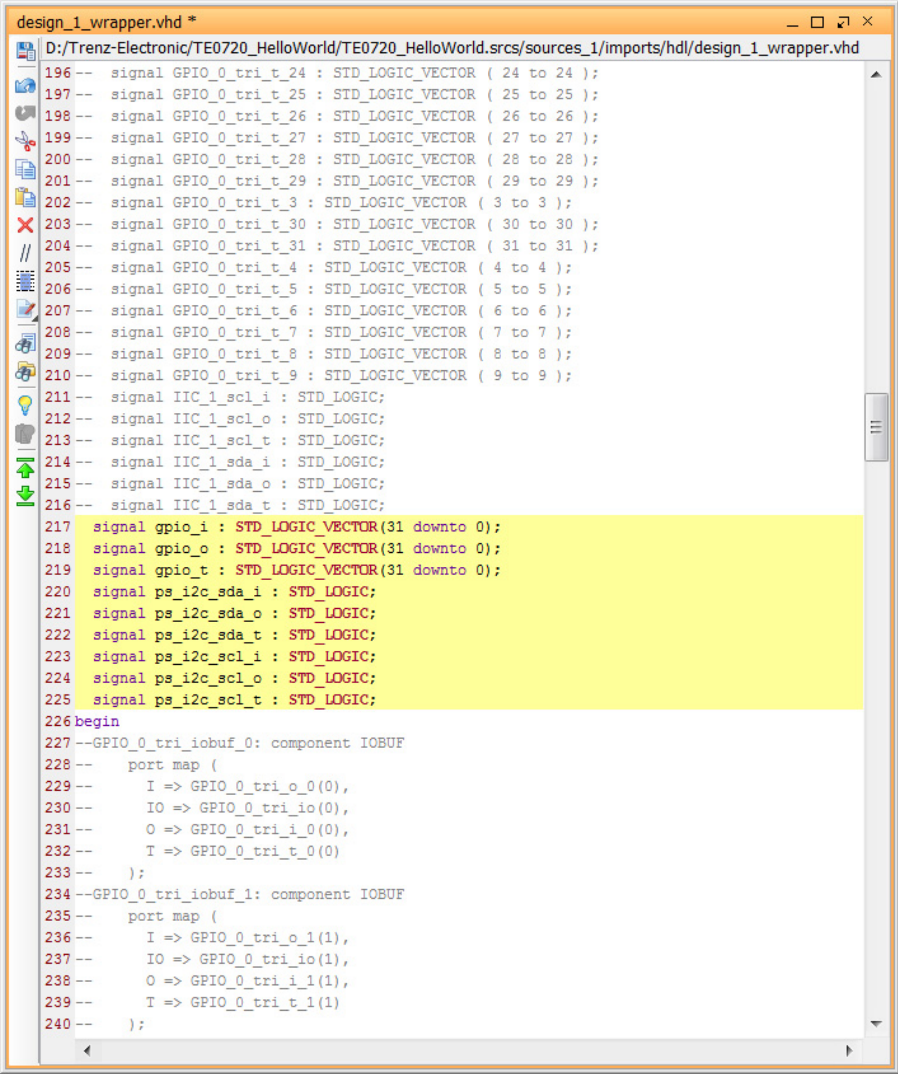
1.) Open the previously generated HDL Wrapper "design_1_wrapper.vhd" and comment external I2C1 and GPIO signals and add CPLD signals to the "design_1_wrapper" entity:



VHDL-Code (to copy & paste):

```
  -- I2C - CPLD connection
  X5 : in STD_LOGIC;    -- i2c_sda_in
  X7 : out STD_LOGIC;   -- i2c_sda_out
  X0 : out STD_LOGIC;   -- i2c_sw
  X1 : out STD_LOGIC;   -- i2c_scl_out

  -- I2C TE0701
i2c_scl_pin : inout STD_LOGIC;  -- TE0701 HDMI SCL
i2c_sda_pin : inout STD_LOGIC   -- TE0701 HDMI SDA
```

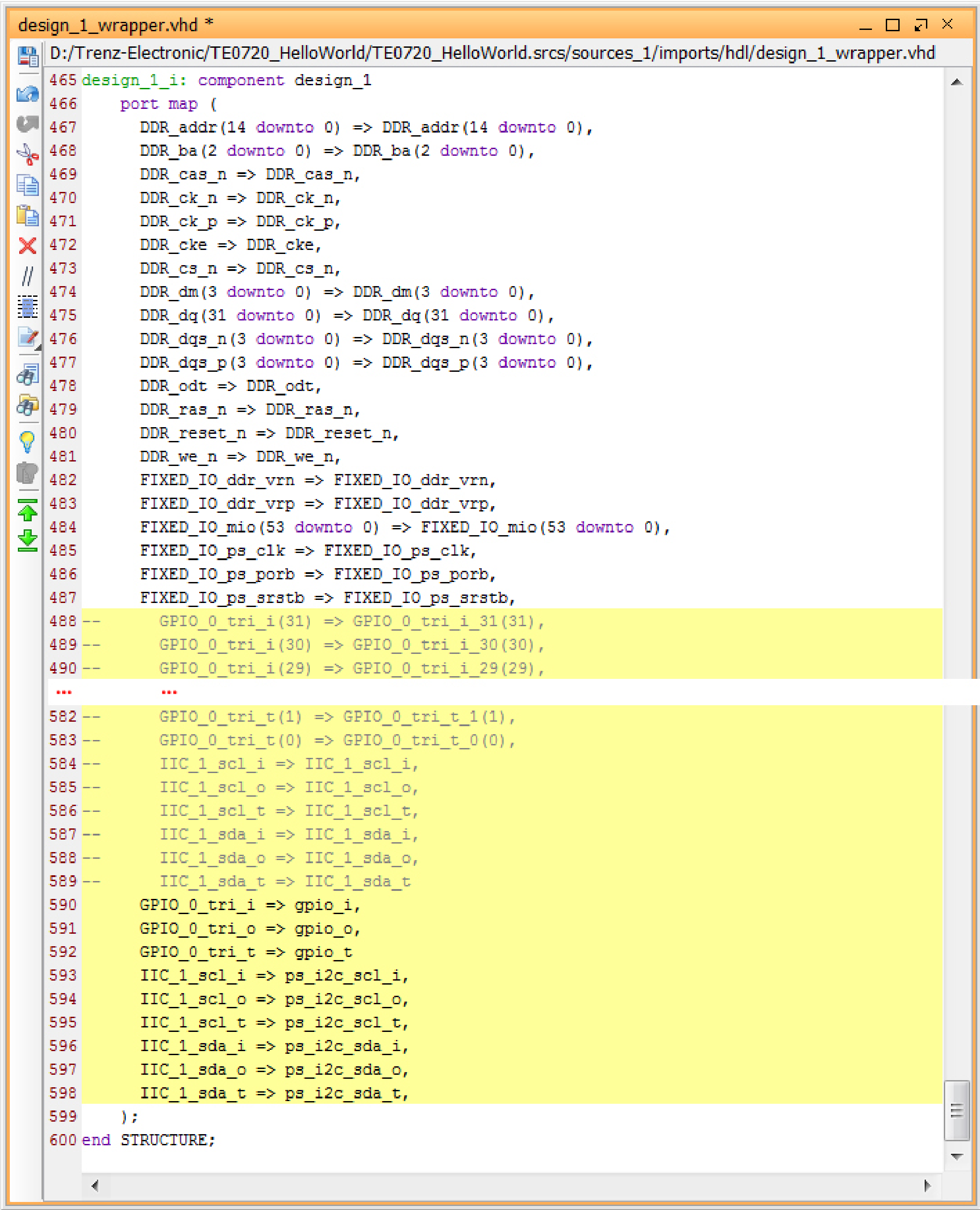
2.) ... add signals to the architecture section (Note: You may also comment the automatically generated I2C1 and GPIO signals as well as the 32 instantiations of the GPIO tristate signal buffers):



VHDL-Code (to copy & paste):

```
signal ps_i2c_sda_i : STD_LOGIC;
signal ps_i2c_sda_o : STD_LOGIC;
signal ps_i2c_sda_t : STD_LOGIC;
signal ps_i2c_scl_i : STD_LOGIC;
signal ps_i2c_scl_o : STD_LOGIC;
signal ps_i2c_scl_t : STD_LOGIC;
signal gpio_i : STD_LOGIC_VECTOR(31 downto 0);
signal gpio_o : STD_LOGIC_VECTOR(31 downto 0);
signal gpio_t : STD_LOGIC_VECTOR(31 downto 0);
```

3.) ... replace the port mapping of GPIO (i.e., by commenting the 32 GPIO_0_tri_i, 32 GPIO_0_tri_o, and 32 GPIO_0_tri_t port mappings and replacing them by the corresponding GPIO vectors) as well as I2C1:

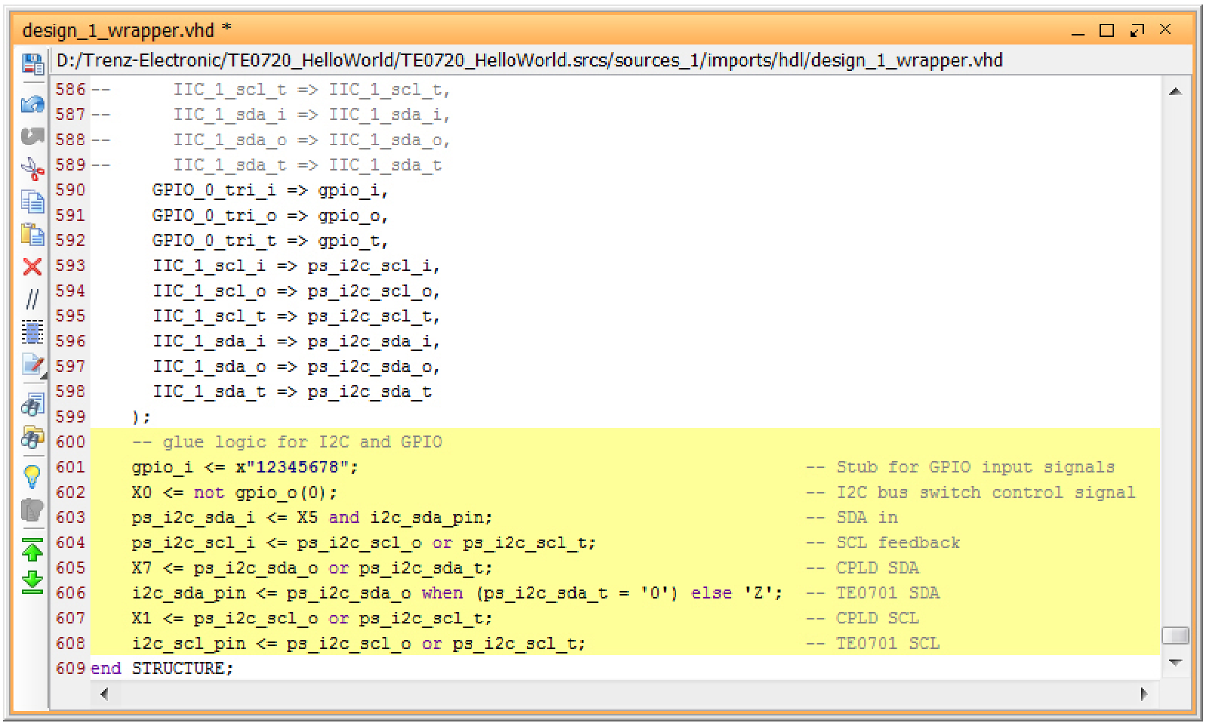


```
design_1_wrapper.vhd *
D:/Trenz-Electronic/TE0720_HelloWorld/TE0720_HelloWorld.srcs/sources_1/imports/hdl/design_1_wrapper.vhd
design_1_i: component design_1
    port map (
      DDR_addr(14 downto 0) => DDR_addr(14 downto 0),
      DDR_ba(2 downto 0) => DDR_ba(2 downto 0),
      DDR_cas_n => DDR_cas_n,
      DDR_ck_n => DDR_ck_n,
      DDR_ck_p => DDR_ck_p,
      DDR_cke => DDR_cke,
      DDR_cs_n => DDR_cs_n,
      DDR_dm(3 downto 0) => DDR_dm(3 downto 0),
      DDR_dq(31 downto 0) => DDR_dq(31 downto 0),
      DDR_dqs_n(3 downto 0) => DDR_dqs_n(3 downto 0),
      DDR_dqs_p(3 downto 0) => DDR_dqs_p(3 downto 0),
      DDR_odt => DDR_odt,
      DDR_ras_n => DDR_ras_n,
      DDR_reset_n => DDR_reset_n,
      DDR_we_n => DDR_we_n,
      FIXED_IO_ddr_vrn => FIXED_IO_ddr_vrn,
      FIXED_IO_ddr_vrp => FIXED_IO_ddr_vrp,
      FIXED_IO_mio(53 downto 0) => FIXED_IO_mio(53 downto 0),
      FIXED_IO_ps_clk => FIXED_IO_ps_clk,
      FIXED_IO_ps_porb => FIXED_IO_ps_porb,
      FIXED_IO_ps_srstb => FIXED_IO_ps_srstb,
--      GPIO_0_tri_i(31) => GPIO_0_tri_i_31(31),
--      GPIO_0_tri_i(30) => GPIO_0_tri_i_30(30),
--      GPIO_0_tri_i(29) => GPIO_0_tri_i_29(29),
...        ...
--      GPIO_0_tri_t(1) => GPIO_0_tri_t_1(1),
--      GPIO_0_tri_t(0) => GPIO_0_tri_t_0(0),
--      IIC_1_scl_i => IIC_1_scl_i,
--      IIC_1_scl_o => IIC_1_scl_o,
--      IIC_1_scl_t => IIC_1_scl_t,
--      IIC_1_sda_i => IIC_1_sda_i,
--      IIC_1_sda_o => IIC_1_sda_o,
--      IIC_1_sda_t => IIC_1_sda_t
      GPIO_0_tri_i => gpio_i,
      GPIO_0_tri_o => gpio_o,
      GPIO_0_tri_t => gpio_t
      IIC_1_scl_i => ps_i2c_scl_i,
      IIC_1_scl_o => ps_i2c_scl_o,
      IIC_1_scl_t => ps_i2c_scl_t,
      IIC_1_sda_i => ps_i2c_sda_i,
      IIC_1_sda_o => ps_i2c_sda_o,
      IIC_1_sda_t => ps_i2c_sda_t,
    );
end STRUCTURE;
```

VHDL-Code (to copy & paste):

```
GPIO_0_tri_i => gpio_i,
GPIO_0_tri_o => gpio_o,
GPIO_0_tri_t => gpio_t,
IIC_1_scl_i => ps_i2c_scl_i,
IIC_1_scl_o => ps_i2c_scl_o,
IIC_1_scl_t => ps_i2c_scl_t,
IIC_1_sda_i => ps_i2c_sda_i,
IIC_1_sda_o => ps_i2c_sda_o,
IIC_1_sda_t => ps_i2c_sda_t
```

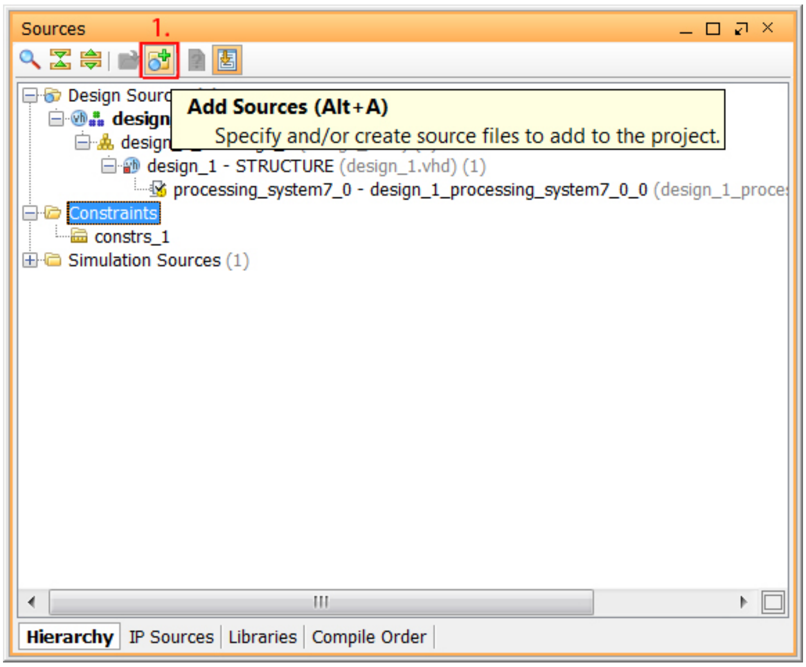
4.) ... and finally, some "glue" logic has to be added to the architecture section:

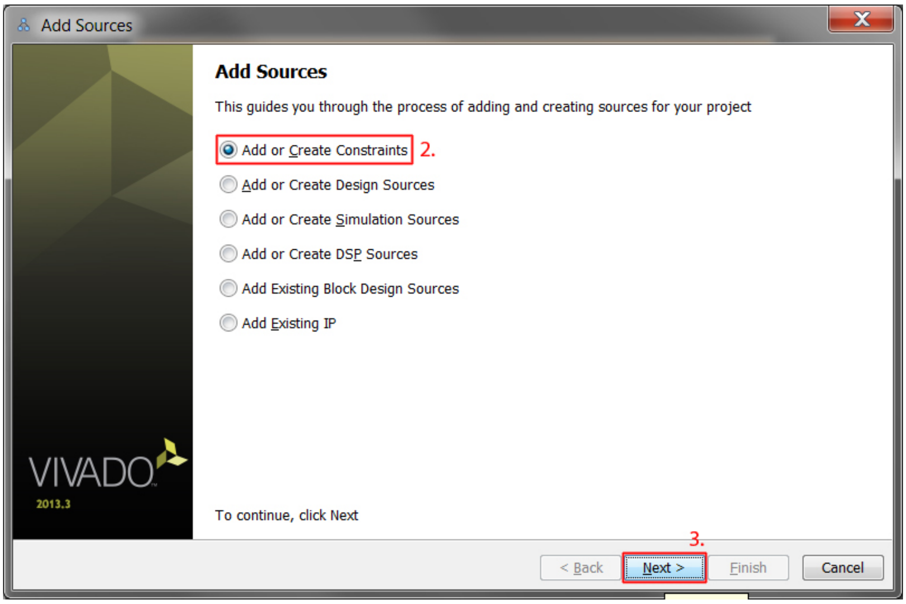


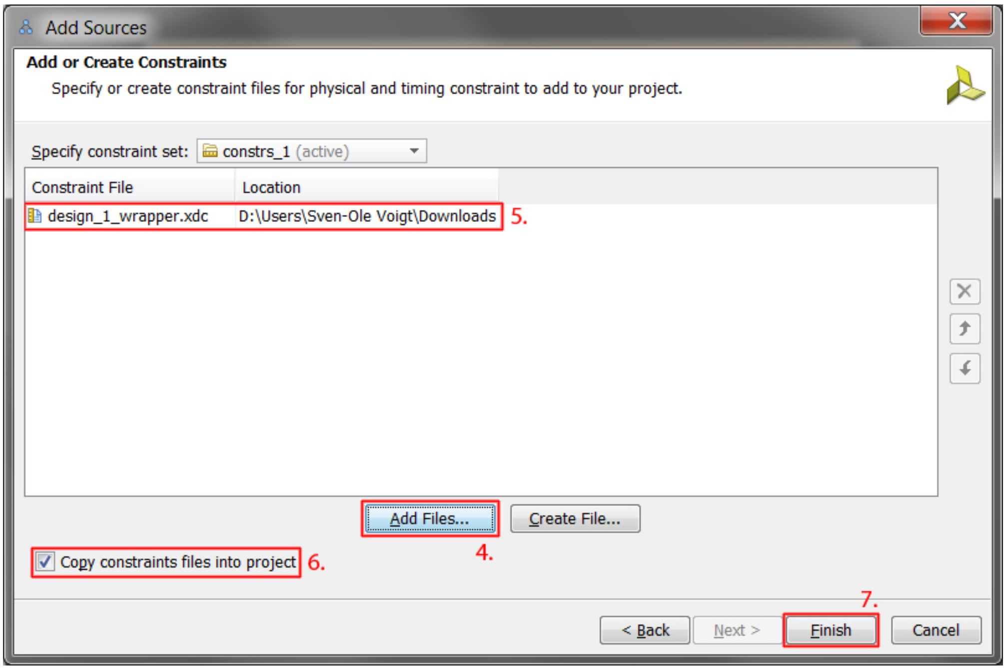
VHDL-Code (to copy & paste):

```
-- glue logic for I2C and GPIO
gpio_i <= x"12345678";                          -- Stub for GPIO input signals
X0 <= not gpio_o(0);                            -- I2C bus switch control signa

ps_i2c_sda_i <= X5 and i2c_sda_pin;             -- SDA in
ps_i2c_scl_i <= ps_i2c_scl_o or ps_i2c_scl_t;   -- SCL feedback
X7 <= ps_i2c_sda_o or ps_i2c_sda_t;             -- CPLD SDA
i2c_sda_pin <= ps_i2c_sda_o
when (ps_i2c_sda_t = '0') else 'Z';             -- TE0701 SDA
X1 <= ps_i2c_scl_o or ps_i2c_scl_t;             -- CPLD SCL
i2c_scl_pin <= ps_i2c_scl_o or ps_i2c_scl_t;    -- TE0701 SCL
```

5.) Create a new XDC constraint file and add the following XDC constraint file "design_1_wrapper.xdc":


Sources
1.
Add Sources (Alt+A)
Specify and/or create source files to add to the project.
Design Sources
design_1 - STRUCTURE (design_1.vhd) (1)
processing_system7_0 - design_1_processing_system7_0_0
Constraints
constrs_1
Simulation Sources (1)
Hierarchy
IP Sources
Libraries
Compile Order


Add Sources
Add Sources
This guides you through the process of adding and creating sources for your project
Add or Create Constraints
2.
Add or Create Design Sources
Add or Create Simulation Sources
Add or Create DSP Sources
Add Existing Block Design Sources
Add Existing IP
VIVADO
2013.3
To continue, click Next
3.
< Back
Next >
Finish
Cancel

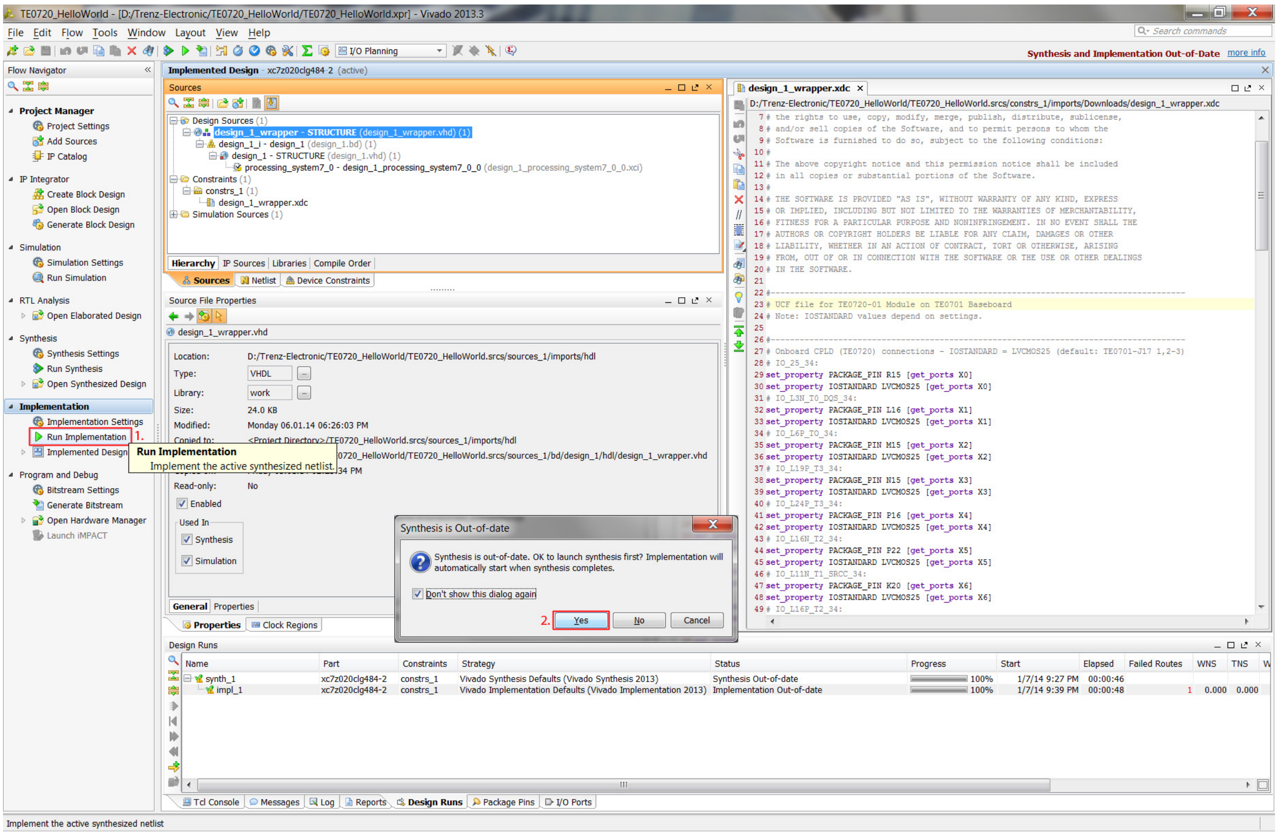


Generally, if you would like to use an "old" UCF file, you may use our following perl script (which follows the instructions of the Xilinx UG911 (v2013.3) "ISE-Vivado Design Suite Migration Guide"; see section "UCF to XDC Mapping", p.23) to easily convert the physical constraints from an existing UCF input file:

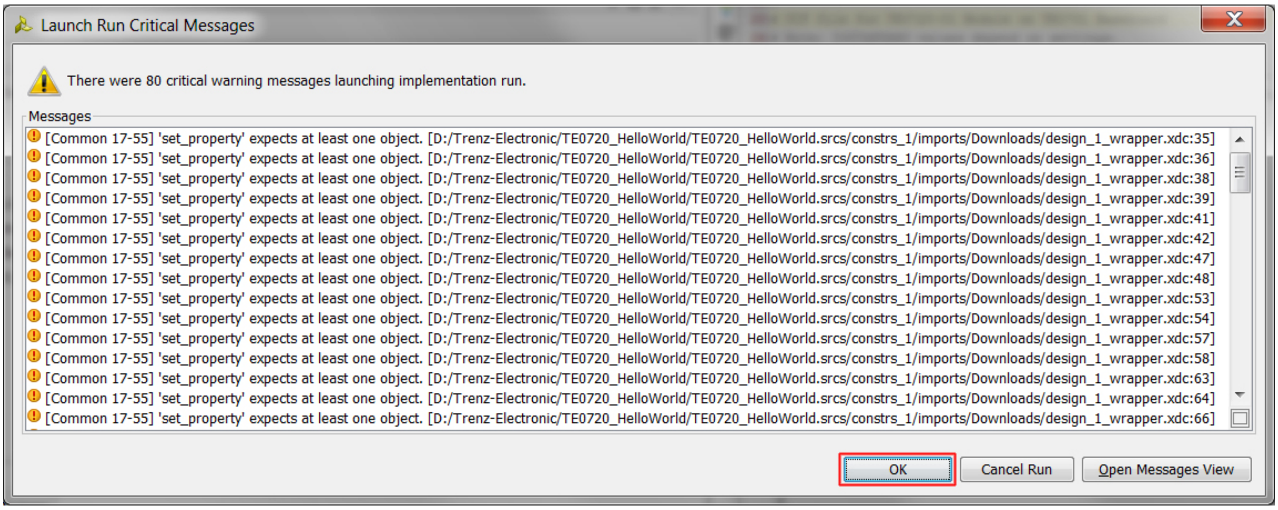
```
#--------------------------------------------------------------------------

#-- Copyright (c) 2014 by Trenz Electronic.
#-- Holzweg 19a, 32257 Buende, Germany, www.trenz-electronic.de
#--------------------------------------------------------------------------

#-- Project:     TE07xx Series
#-- File:        ucf2xdc.pl
#-- Description: Perl script to convert Xilinx ucf to xdc contraint files.

#-- History:     2014-01-07, SOV, Created.
#--------------------------------------------------------------------------



open(IN,'input.ucf');
open(OUT,'> output.xdc');

while ($line = <IN>) {
    # remove leading, trailing and multiple spaces
    $line =~ s/^ //g;
    $line =~ s/ $//g;
    $line =~ s/  / /g;
```

```
        # comment or empty lines are simple copied without any modification
        if ((substr($line,0,1) =~ "#") || (substr($line,0,1) =~ "\n")){
            print OUT $line;
        }
        else {
            if ($line =~ /(?<=NET )\w*/){
                $NET = $&;
                if ($line =~ /(?<=LOC = )\w*/){
                    # UCF: NET "<net_name>" LOC = <pin_name>
                    #      -> XDC: set_property PACKAGE_PIN <pin_name>
                    #      [<get_ports net_name>]
                    $LOC = $&;
                    print OUT "set_property " . "PACKAGE_PIN " . $LOC . "
                    [get_ports " . $NET . "]" . "\n";
                }
                if ($line =~ /(?<=IOSTANDARD = )\w*/){
                    # UCF: NET "<net_name>" IOSTANDARD = <iostandard_type>
                    #      -> XDC: set_property IOSTANDARD <iostandard_type>
                    #      [<get_ports net_name>]
                    $IOSTANDARD = $&;
                    print OUT "set_property " . "IOSTANDARD " . $IOSTANDARD
                    . " [get_ports " . $NET . "]" . "\n";
                }
            }
        }
    }
    close(OUT);
    close(IN);
```

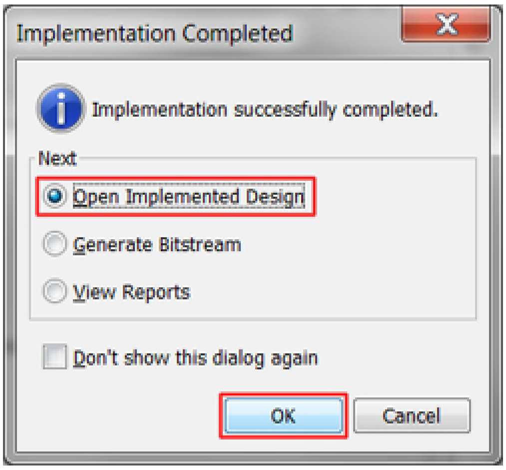
6.) To synthesize and implement the "Hello World" project, you just have to click on the "Run Implementation" button. **Note**: Whenever the synthesis is out-of-date, it will be asked if it should be done first:



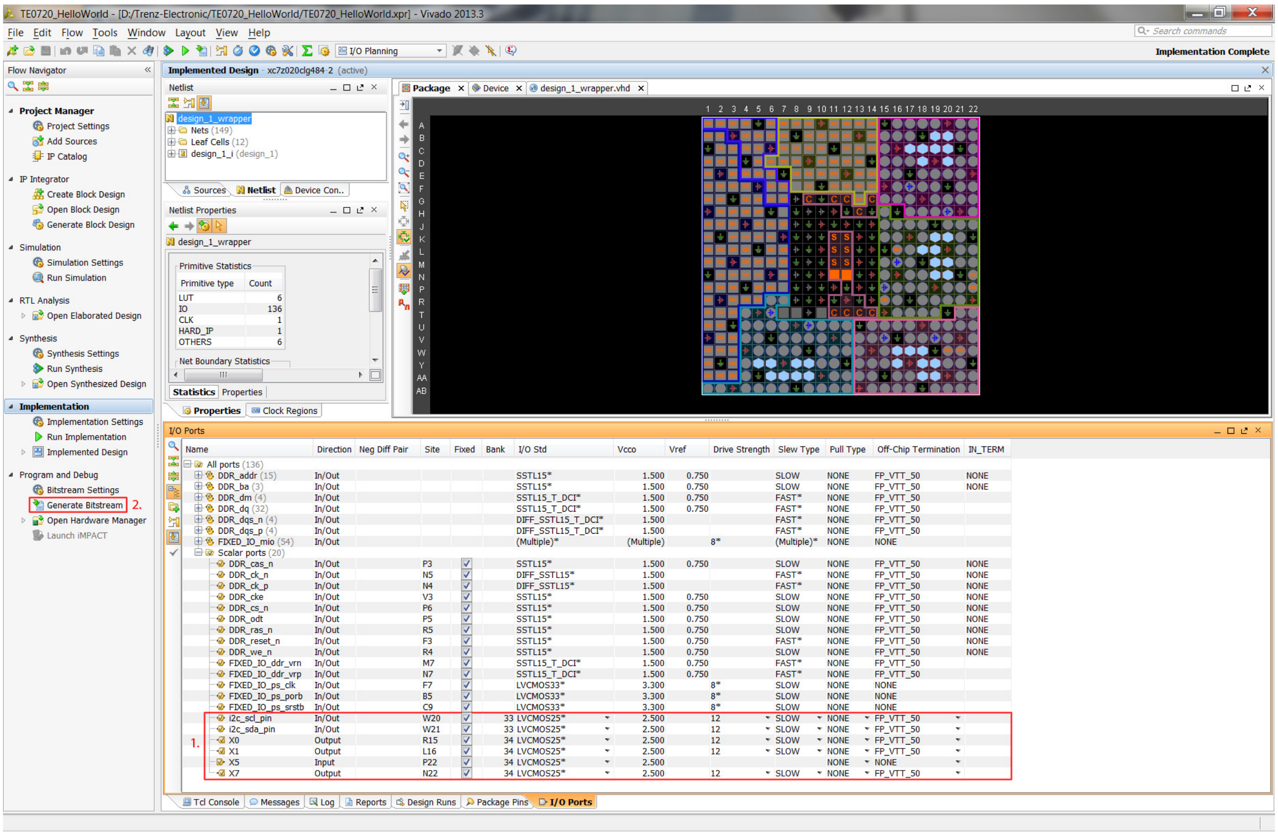
7.) After some time the synthesis and implementation processes complete and the following dialog windows with about 80 critical warnings appear. That is fine, because the XDC contraint file given consists of many more ports than we used in this simple "Hello World" project (e.g., X2, X3, X4, X6 etc.), so you can relax and press "ok":



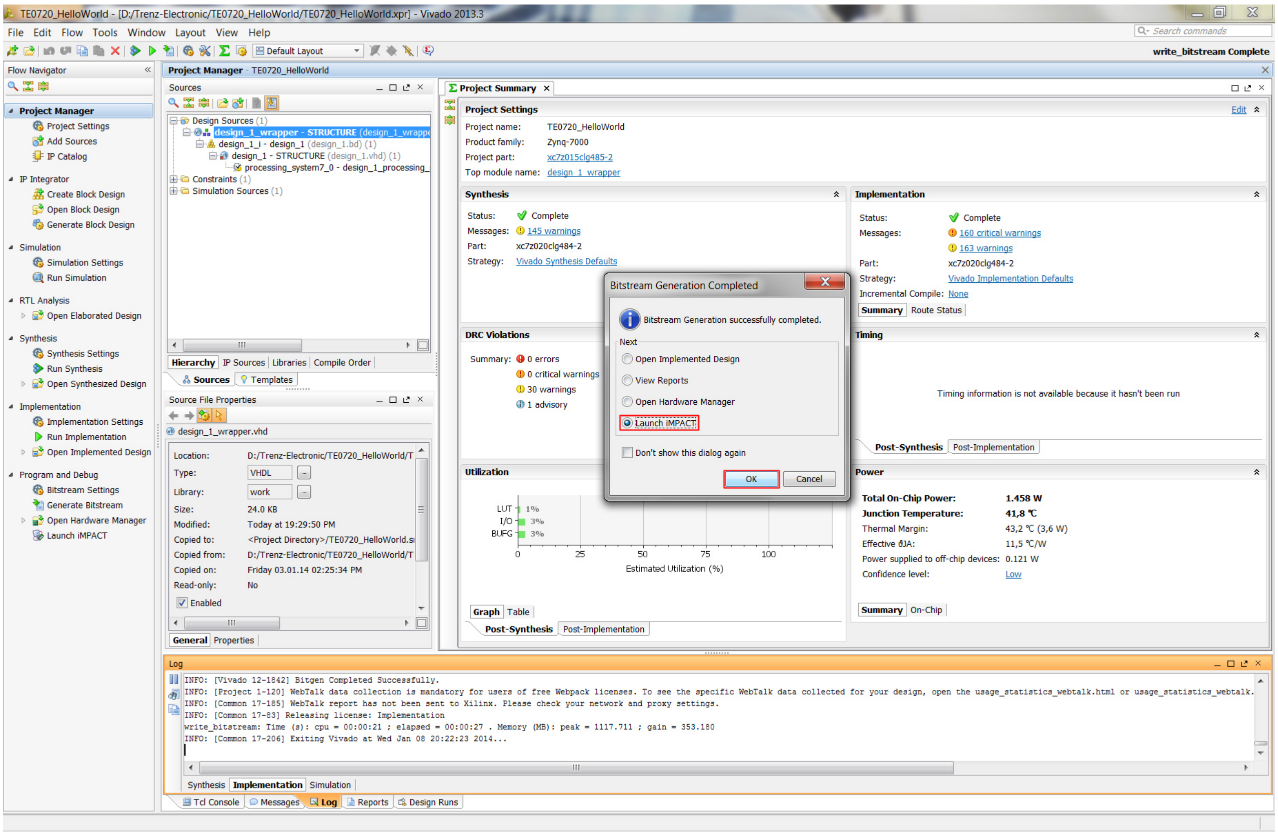
8.) Then, another dialog window appears. Keep the default selection that opens the implemented design:



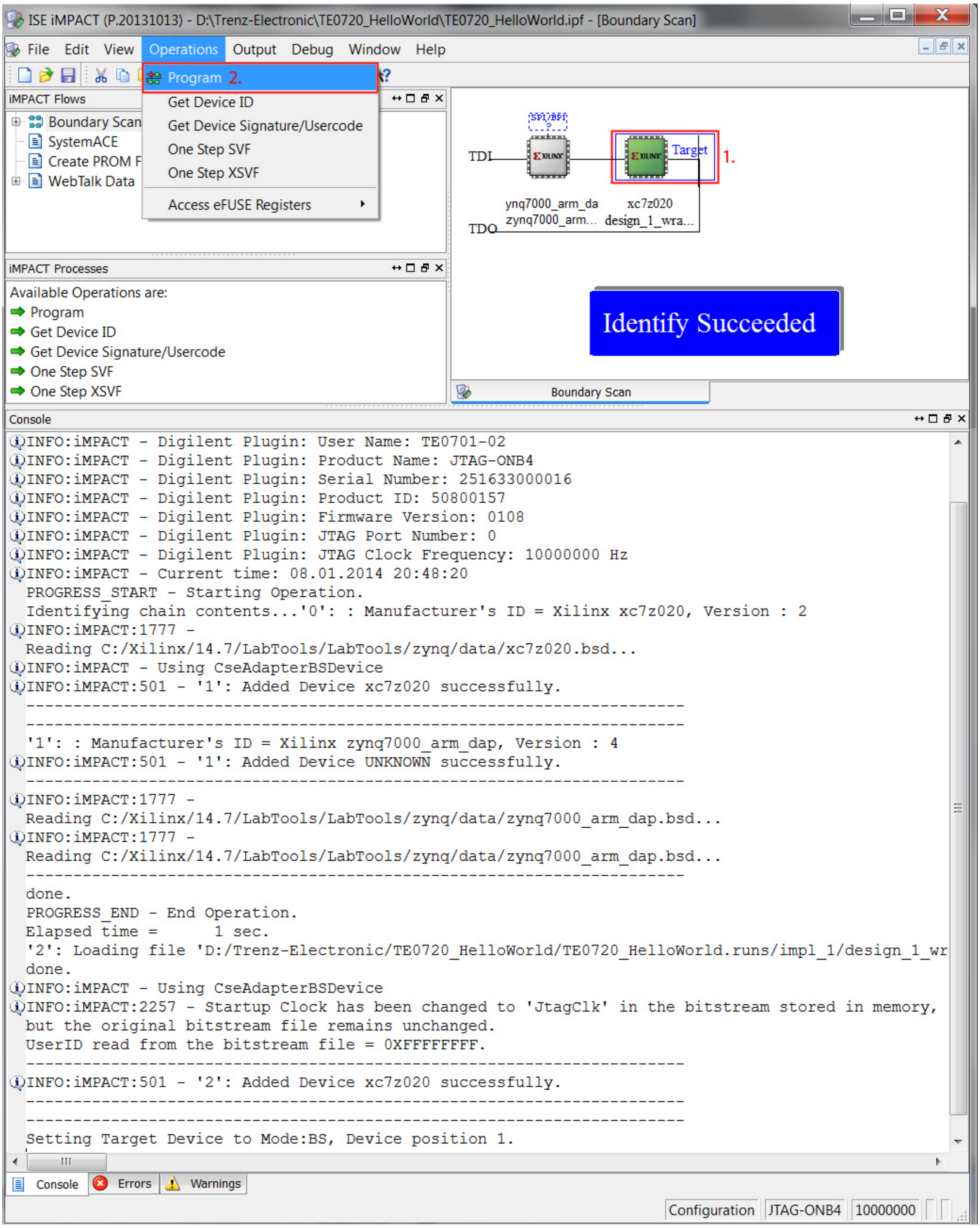
9.) After the implemented design has been opened you can check on I/O ports if the constraints have been configured correctly (1) and when everything is accurate, the bitfile can be finally generated (2):



10.) As soon as the bitfile has been successfully generated, in another dialog window you can choose to launch Xilinx iMPACT to program your FPGA device:



11.) In Xilinx iMPACT the project's bitfile "design_1_wrapper.bit" has already been automatically assigned to the xc7z020 target (i.e., the PL of the Xilinx Zynq FPGA). After selecting it (1), you can choose in the main menu "Operations" to finally program the FPGA (2):



A successful FPGA configuration can only be seen (for now) by checking if the green LED3 (see TE0720 GigaZee Zynq SoM User Manual | On-board LEDs) has switched off. Because this is admittedly quite a bit unsatisfying, in the following section we will extend our "Hello World" application by a I2C software driver on a standalone platform and afterwards our tutorial will continue on "How to run an embedded Linux system on TE0720 GigaZee Zynq SoM (Vivado Flow)".

Software Implementation: "Hello World 2.0" (implementing access to I2C peripherals via Xilinx Zynq PL custom logic)

TODO!!!

Debugging the "Hello World" Project

Video Tutorial

The following screen-recording video shows how to execute "Hello World" from the debugger (Xilinx XMD launched from SDK):

Step-by-Step Tutorial

The simple "Hello World" application above shall now be executed step-by-step in the SDK environment:

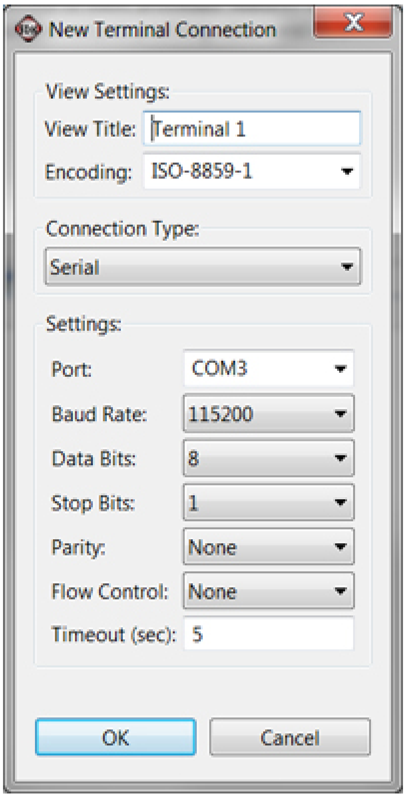
1.) Plug-in 12V power supply.

2.) Plug a standard USB A-Male to Mini-B cable into the USB port of the Host PC and into the mini USB connector (see (19) in Figure 1 of TE0701 Carrier for TE07xx series) on the TE0701 carrier board.

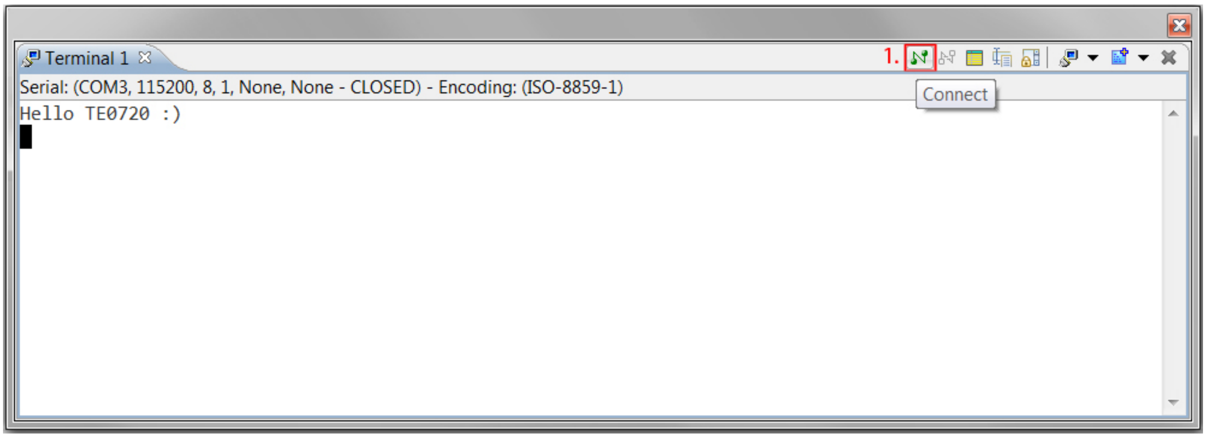
3.) Optional: Plug-in the SD card (see section "Software Implementation: Create *First Stage Boot Loader* (FSBL) and *"Hello World"* application project in SDK") into the slot on the TE0701 Carrier board.

4.) Create a new terminal connection to the virtual COM of the FTDI USB Converter (channel B) and choose the following settings:

Copyright © 2014 Trenz Electronic GmbH

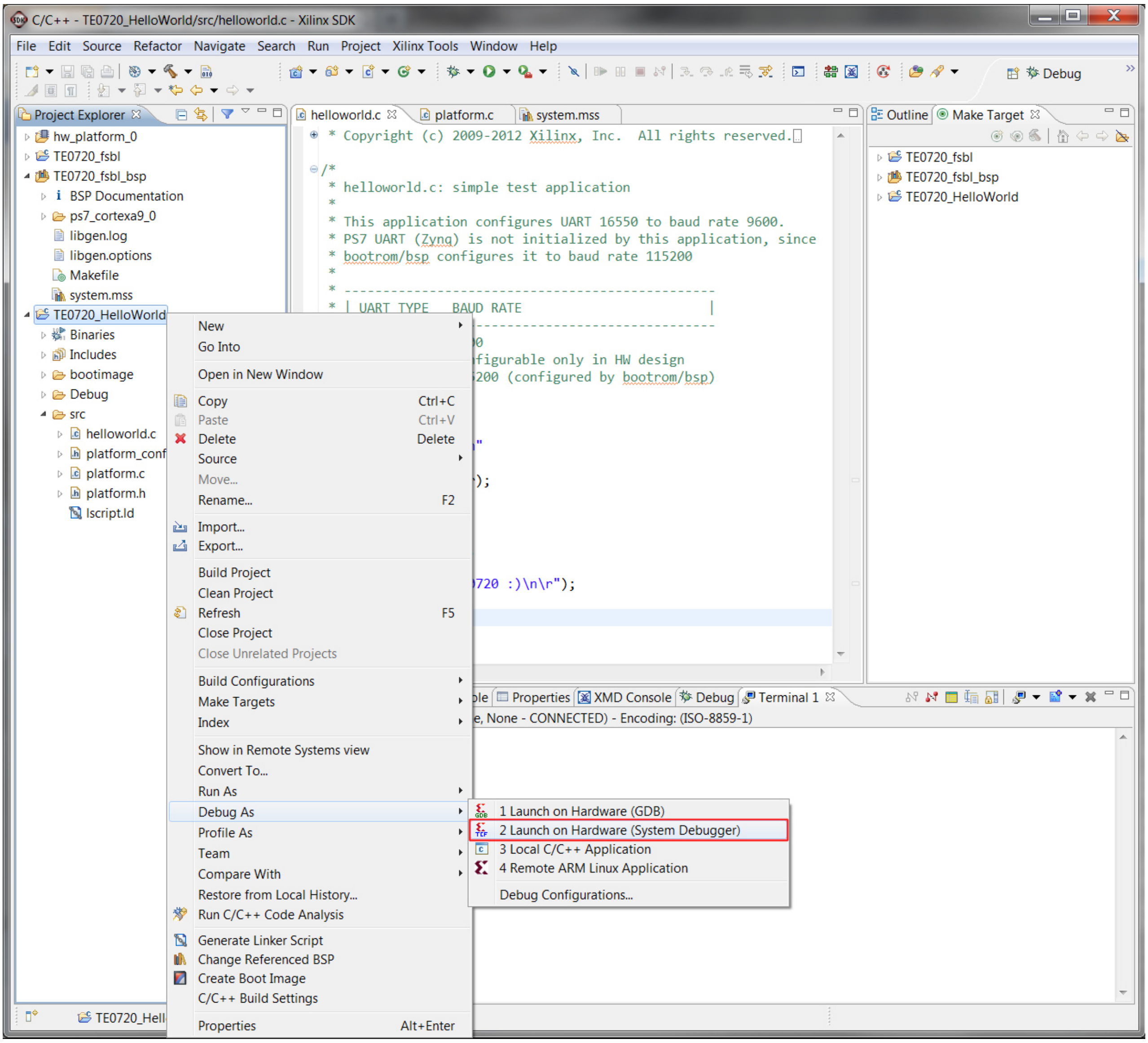


5.) Press the "Connect" button (1) and the "Hello world" output will shortly appear:

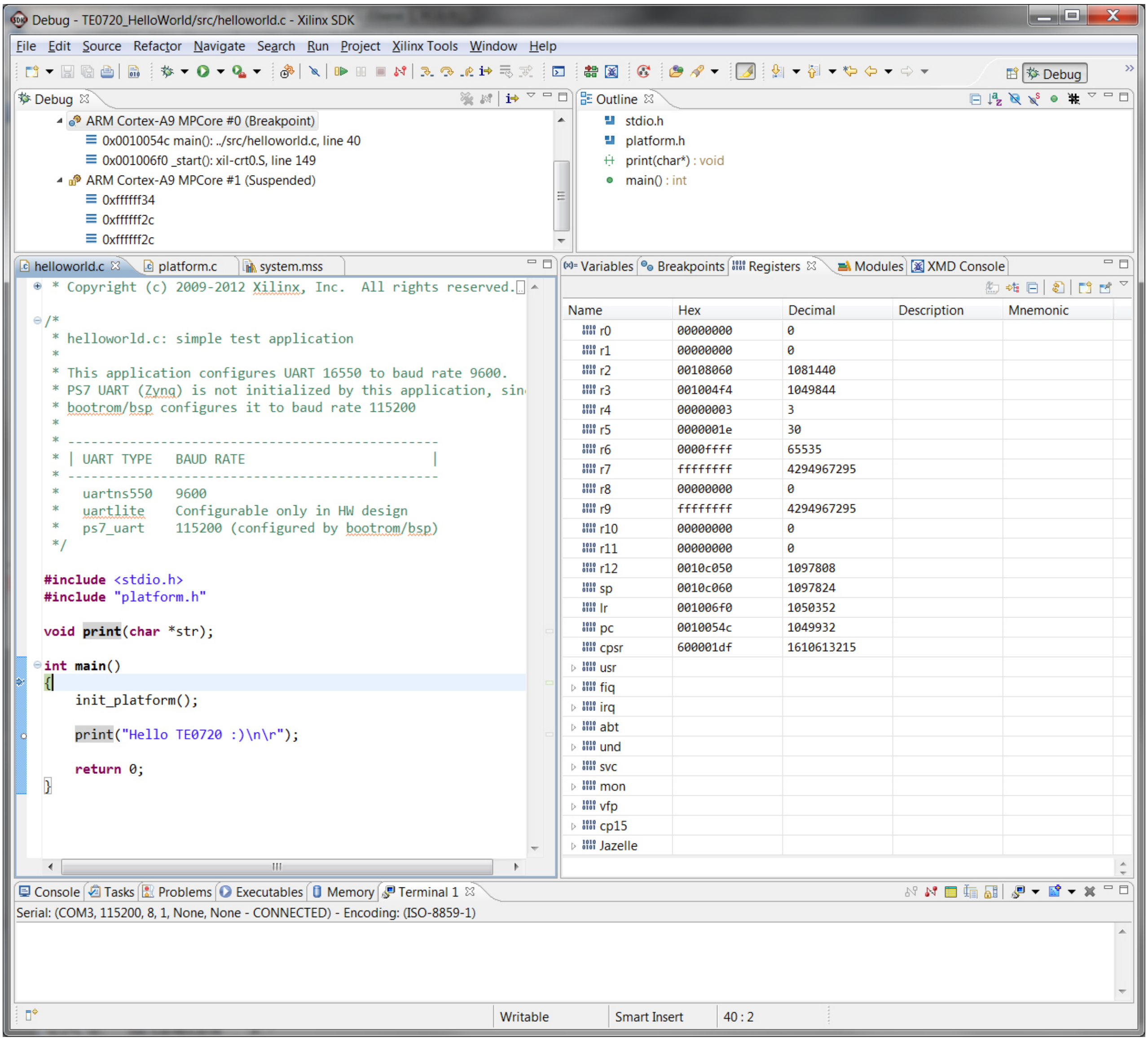


The software application (when running from the SD card) might have already finished execution, so you may push the S2 switch button on the TE0701 Carrier Board (see (17) in Figure 1 of TE0701 Carrier Board User Manual) to "restart" the Zynq SoC Module (see TE0701 Carrier Board User Manual | User Push Buttons (PBs) and LEDs for more details).

6.) Of course, you can also download the same or a different application onto the running *processing unit* (PS) of the Zynq FPGA:



7.) ... and the program execution is (by default) halted in the beginning of "main". By pressing F6 on the keyboard you can debug your program step-by-step:


Debug - TE0720_HelloWorld/src/helloworld.c - Xilinx SDK
File Edit Source Refactor Navigate Search Run Project Xilinx Tools Window Help
Debug
ARM Cortex-A9 MPCore #0 (Breakpoint)
0x0010054c main(): ../src/helloworld.c, line 40
0x001006f0 _start(): xil-crt0.S, line 149
ARM Cortex-A9 MPCore #1 (Suspended)
0xffffff34
0xfffff2c
0xfffff2c
Outline
stdio.h
platform.h
print(char*) : void
main() : int
helloworld.c
platform.c
system.mss
* Copyright (c) 2009-2012 Xilinx, Inc. All rights reserved.
/*
* helloworld.c: simple test application
*
* This application configures UART 16550 to baud rate 9600.
* PS7 UART (Zynq) is not initialized by this application, sin
* bootrom/bsp configures it to baud rate 115200
* | UART TYPE BAUD RATE |
* uartns550 9600
* uartlite Configurable only in HW design
* ps7_uart 115200 (configured by bootrom/bsp)
*/
#include <stdio.h>
#include "platform.h"
void print(char *str);
int main()
{
init_platform();
print("Hello TE0720 :)\n\r");
return 0;
}
Variables
Breakpoints
Registers
Modules
XMD Console
Name
Hex
Decimal
Description
Mnemonic
r0 00000000 0
r1 00000000 0
r2 00108060 1081440
r3 001004f4 1049844
r4 00000003 3
r5 0000001e 30
r6 0000ffff 65535
r7 ffffffff 4294967295
r8 00000000 0
r9 ffffffff 4294967295
r10 00000000 0
r11 00000000 0
r12 0010c050 1097808
sp 0010c060 1097824
lr 001006f0 1050352
pc 0010054c 1049932
cpsr 600001df 1610613215
usr
fiq
irq
abt
und
svc
mon
vfp
cp15
Jazelle
Console
Tasks
Problems
Executables
Memory
Terminal 1
Serial: (COM3, 115200, 8, 1, None, None - CONNECTED) - Encoding: (ISO-8859-1)
Writable
Smart Insert
40 : 2

FPGA design without PS

TE0720 is a Zynq-7000 SoC based module, and it is normally used with the PS subsystem enabled and included in the design. Of course, it is also possilble to use the TE0720 without instantiating the PS in the design. Such a design can be downloaded into the module through the JTAG port (in a volatile way). It can also be stored to (in a non-volatile way) and loaded at configuration time from QSPI, eMMC or SD Card.

> The Zynq-7000 SoC design file must be generated with `FPGA Start-Up Clock` option set to `CCLK,` otherwise the SoC bitstream will not be enabled when configured by FSBL.

The regular FSBL is able to configure the Zynq PL fabric with a design without any PS instance, but a special compile-time flag **NON_PS_INSTANTIATED_BITSTREAM** must be defined.

> If executing from a SD memory card, FSBL must have the DDR memory initialized properly, otherwise the PL configuration fails, as FSBL copies the SoC bitstream from the SD memory card to a temporary DDR location!

Copyright © 2014 Trenz Electronic GmbH

Xilinx repository

DEPRECATED - use petalinux 2014.2 or newer and TE0720 BSP

Using official Xilinx linux kernel repository with TE0720

This patch was tested only with kernel 3.10 (commit efc27505715e64526653f35274717c0fc56491e3)

Known issues:

- QSPI Flash not working in u-boot

Clone linux-xlnx and u-boot-xlnx repositories to prepared environment

> *git clone git://github.com/Xilinx/u-boot-xlnx.git*
>
> *git clone git://github.com/Xilinx/linux-xlnx.git*

Download patches
http://www.trenz-electronic.de/fileadmin/docs/Trenz_Electronic/TE0720-GigaZee/reference_designs/TE0720-

Unzip and copy files

> *unzip TE0720-01_linux-xlnx.zip*
>
> *cp linux-xlnx_TE0720.patch linux-xlnx*
>
> *cp u-boot-xlnx_TE0720.patch u-boot.xlnx*

Build linux kernel.

> *cd linux-xlnx*
>
> *git apply linux-xlnx_TE0720.patch*
>
> *make ARCH=arm TE0720_defconfig*
>
> *make ARCH=arm LOADADDR=0x8000 uImage*
>
> *make ARCH=arm TE0720-01-xxF.dtb*

Copyright © 2014 Trenz Electronic GmbH

> *make ARCH=arm TE0720-01-xxR.dtb*

Build U-Boot

> *git apply u-boot-xlnx_TE0720.patch*
>
> *make zynq_TE0720_config*
>
> *make*

High speed ADC Interfacing

TE0720 has no length matching between differential pairs from the FPGA fabric to B2B Connectors. Trace lengths are available in download area, and could be used together with Xilinx provided FPGA Package delay timings to adjust system trace delays.

Zynq 7020-CLG484 largest package delay difference within one bank is 37 mm (B13 L4 to L22). Such large trace differences can not be matched on the module.

For most high speed input designs this delay matching is not needed, as long as the total delay differences are not too large. The delays can be matched using IDELAYE2 primitive on each signal line that needs to be matched. The delays can be set fixed or can be adjustable. Possible use could have the data line delays fixed, and clock/strobe delay variable.

With 300MHz calibrate clock the PCB trace lengths can be matched to be +-4 mm from the mid delay of the group.

References

UG471

XAPP524

XAPP524 does not have IDELAY on data lanes!

Copyright © 2014 Trenz Electronic GmbH

Petalinux

Based on Petalinux 2014.2 release, Kernel 3.14.2

FSBL

Xilinx standard FSBL is used, only fsbl_hooks.c is customized to perform all needed pre-initialization code.

> FSBL bsp files are not auto updated with petalinux-config --get-hw-description so please if you change the system in Vivado and import HDF and if there are changes that are relevant to FSBL please copy over the ps7_init.c and ps7_init.h files into the FSBL build folder manually.

MAC Address handling

MAC Address is read out from EEPROM and set correctly to the PS7 ETH IP Core in order to use and keep this MAC address the following boot process should not attempt to set it again. For this the MAC address is set to empty string in the petalinux configuration. This causes a warning "can not set MAC Address during boot process - this is not an error, MAC address is set correctly and can be seen in linux using ifconfig. Notice that in u-boot it looks like MAC address is not set, as the u-boot variable is not set.

Debug

Booting U-Boot via JTAG

Booting U-Boot via JTAG

Step 1: download files ps7_init.tcl and u-boot.elf from download area

Step 2: connect JTAG Cable

Step 3: connect logic level UART adapter RX to MIO15 (PS UART0 TX) and TX to MIO14 (PS UART0 RX) connection example for TE0603

Step 4: open serial port terminal, setting 115200, 8N1

Step 5: start Xilinx XMD - use Xilinx SDK menu, Xilinx Tools> XMD Console or start XMD from commandline

Step 6: make sure files **ps7_init.tcl** and **u-boot.elf** are "in current folder" as seen by XMD (use pwd to check where current dir points and cd to change directory if needed)

Step 7: type ***connect arm hw***<CR>

Step 7: type ***source ps7_init.tcl***<CR>

Step 8: type ***ps7_init***<CR>

Step 9: type ***dow u-boot.elf***<CR>

Step 10: type ***con***<CR>

At this point u-boot (loaded over JTAG) should be running and prompt should appear in serial console - please press some key in the console window, or u-boot may continue to load linux!

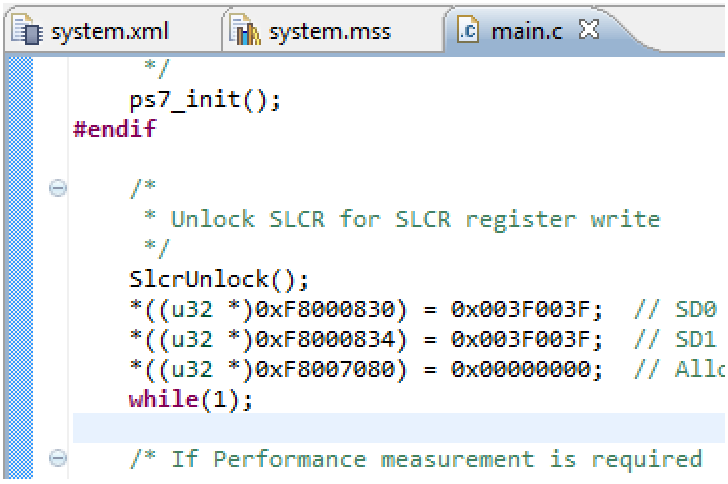
> If TE0720 has valid boot images either in SPI Flash or on SD card, it is necessary to prevent linux from loading, either remove SD card, or in case SPI boot, press any key when u-boot loaded from SPI flash is waiting for user input. Should linux start, then it may not be possible to connect to ARM core from XMD.

Old instructions

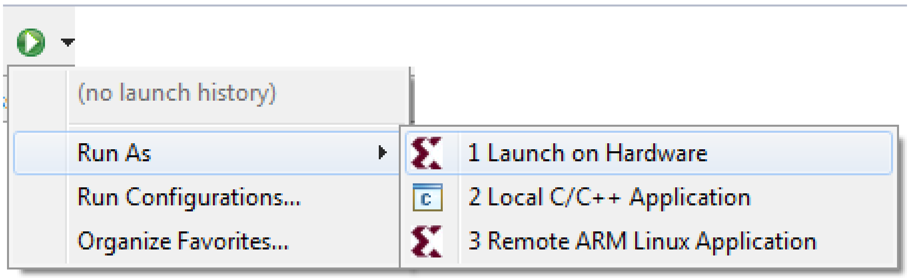
Build base hardware project using instructions from Base PlanAhead Project or Base XPS Project

Build FSBL by instructions from FSBL - First Stage Boot Loader

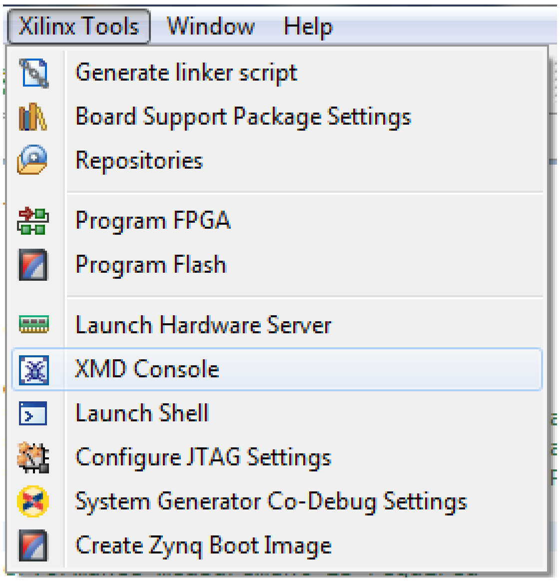
In SDK open main.c file and add infinite loop instruction after system initialization. This loop required to stop normal boot sequence.



Select Run->Run As -> Launch on Hardware



Open XMD Console

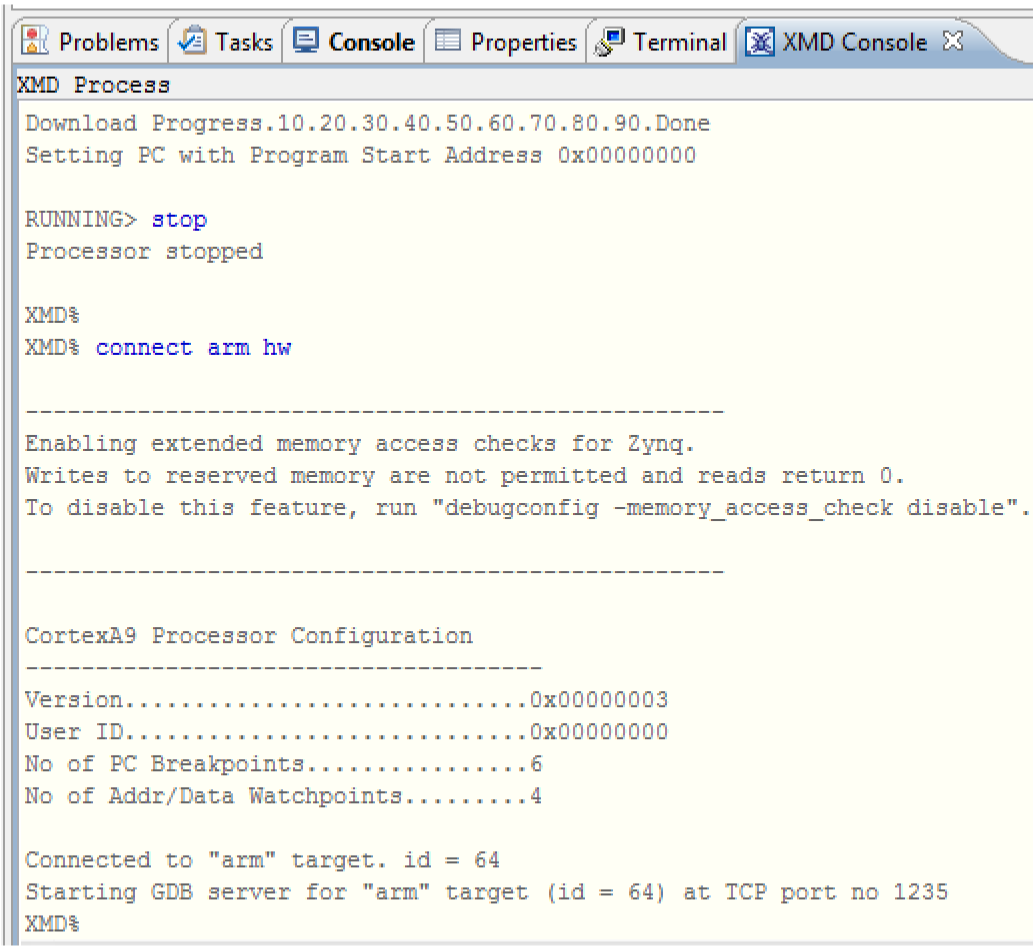


In XMD console type

> *stop*
>
> *connect arm hw*

After this commands system will be initialized and ready to load programms and data to DDR memory.

Copyright © 2014 Trenz Electronic GmbH

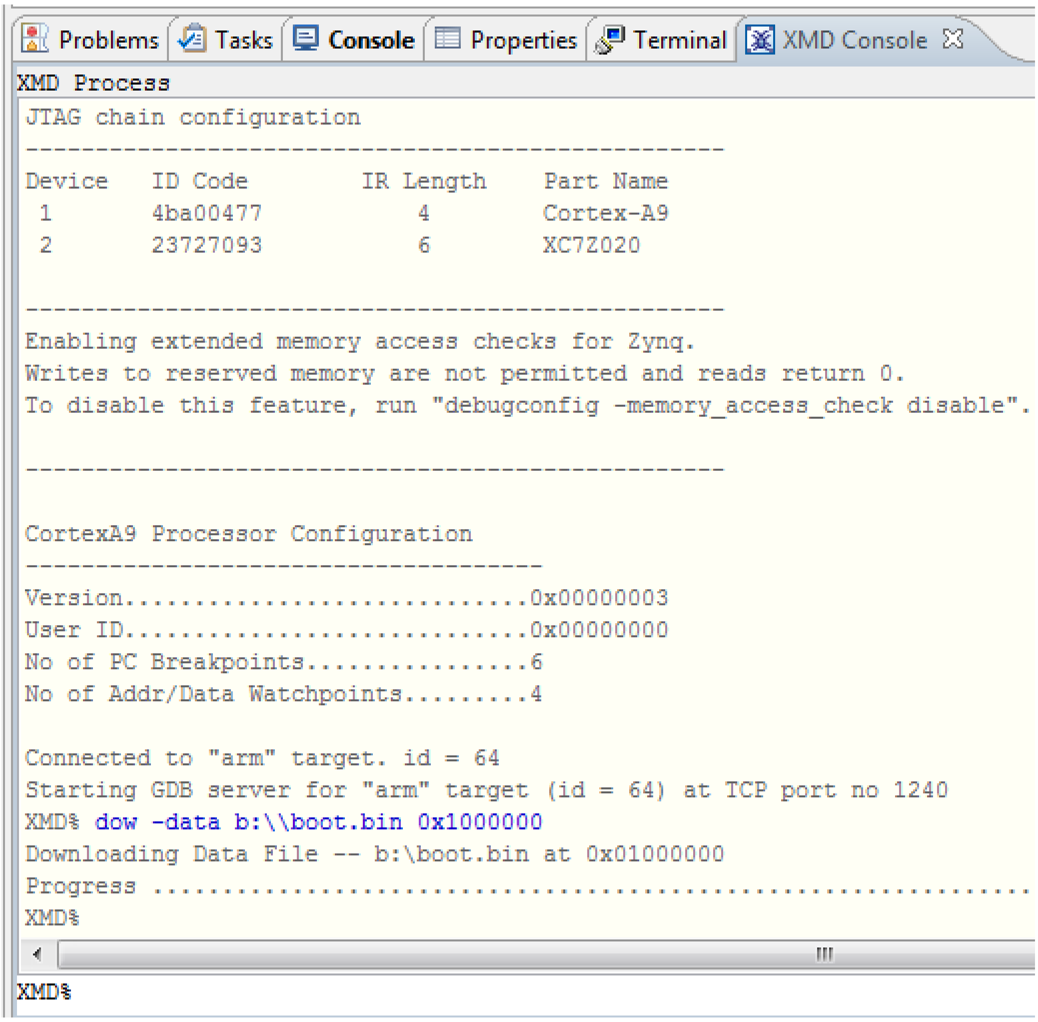



```
Download Progress.10.20.30.40.50.60.70.80.90.Done
Setting PC with Program Start Address 0x00000000

RUNNING> stop
Processor stopped

XMD%
XMD% connect arm hw

--------------------------------------------------
Enabling extended memory access checks for Zynq.
Writes to reserved memory are not permitted and reads return 0.
To disable this feature, run "debugconfig -memory_access_check disable".

--------------------------------------------------

CortexA9 Processor Configuration
-------------------------------------
Version.............................0x00000003
User ID.............................0x00000000
No of PC Breakpoints................6
No of Addr/Data Watchpoints.........4

Connected to "arm" target. id = 64
Starting GDB server for "arm" target (id = 64) at TCP port no 1235
XMD%
```

Optionally, binary data can be downloaded to memory. This data can be used in u-boot to initialize SPI Flash.

> *dow -data FILENAME.BIN 0x100000*

where 0x100000 is location in DDR memory where binary file FILENAME.BIN would be loaded



Download u-boot.elf file

> *dow b:\u-boot.elf*

where b:\ should be replaced with actual path to u-boot.elf file

```
Problems  Tasks  Console  Properties  Terminal  XMD Console
XMD Process
No of Addr/Data Watchpoints.........4

Connected to "arm" target. id = 64
Starting GDB server for "arm" target (id = 64) at TCP port no 1235
XMD% dow b:\u-boot.elf
warning: Data caches are enabled in the system control register.
This could reduce the download speed and if the processor is in
unknown state, it could lead to interrupt failures, etc. Please
run "rst [-processor]" command before dow, to overcome this issue
Processor started. Type "stop" to stop processor
User Interrupt, Processor Stopped at 0x000036e0
Downloading Program -- b:u-boot.elf
    section, .text: 0x04000000-0x0402d73f
    section, .rodata: 0x0402d740-0x04039395
    section, .hash: 0x04039398-0x040393ef
    section, .ARM.extab: 0x040393f0-0x04039407
    section, .ARM.exidx: 0x04039408-0x0403942f
    section, .data: 0x04039430-0x0403b4d3
    section, .got.plt: 0x0403b4d4-0x0403b4df
    section, .u_boot_list: 0x0403b4e0-0x0403bb83
    section, .rel.dyn: 0x0403bb84-0x04041e93
    section, .dynsym: 0x04041e94-0x04041fa3
    section, .bss: 0x0403bb84-0x0407b3bb
Download Progress..10.20.30.40.50.60.70.80.90.Done
Setting PC with Program Start Address 0x04000000
XMD%

XMD%
```

> *run*

```
Problems  Tasks  Console  Properties  Terminal  XMD Console
XMD Process
Version............................0x00000003
User ID............................0x00000000
No of PC Breakpoints................6
No of Addr/Data Watchpoints.........4

Connected to "arm" target. id = 64
Starting GDB server for "arm" target (id = 64) at TCP port no 1237
XMD% dow b:\u-boot.elf
Downloading Program -- b:u-boot.elf
    section, .text: 0x04000000-0x0402d73f
    section, .rodata: 0x0402d740-0x04039395
    section, .hash: 0x04039398-0x040393ef
    section, .ARM.extab: 0x040393f0-0x04039407
    section, .ARM.exidx: 0x04039408-0x0403942f
    section, .data: 0x04039430-0x0403b4d3
    section, .got.plt: 0x0403b4d4-0x0403b4df
    section, .u_boot_list: 0x0403b4e0-0x0403bb83
    section, .rel.dyn: 0x0403bb84-0x04041e93
    section, .dynsym: 0x04041e94-0x04041fa3
    section, .bss: 0x0403bb84-0x0407b3bb
Download Progress..10.20.30.40.50.60.70.80.90.Done
Setting PC with Program Start Address 0x04000000
XMD% run

RUNNING> 0
XMD%

XMD%
```

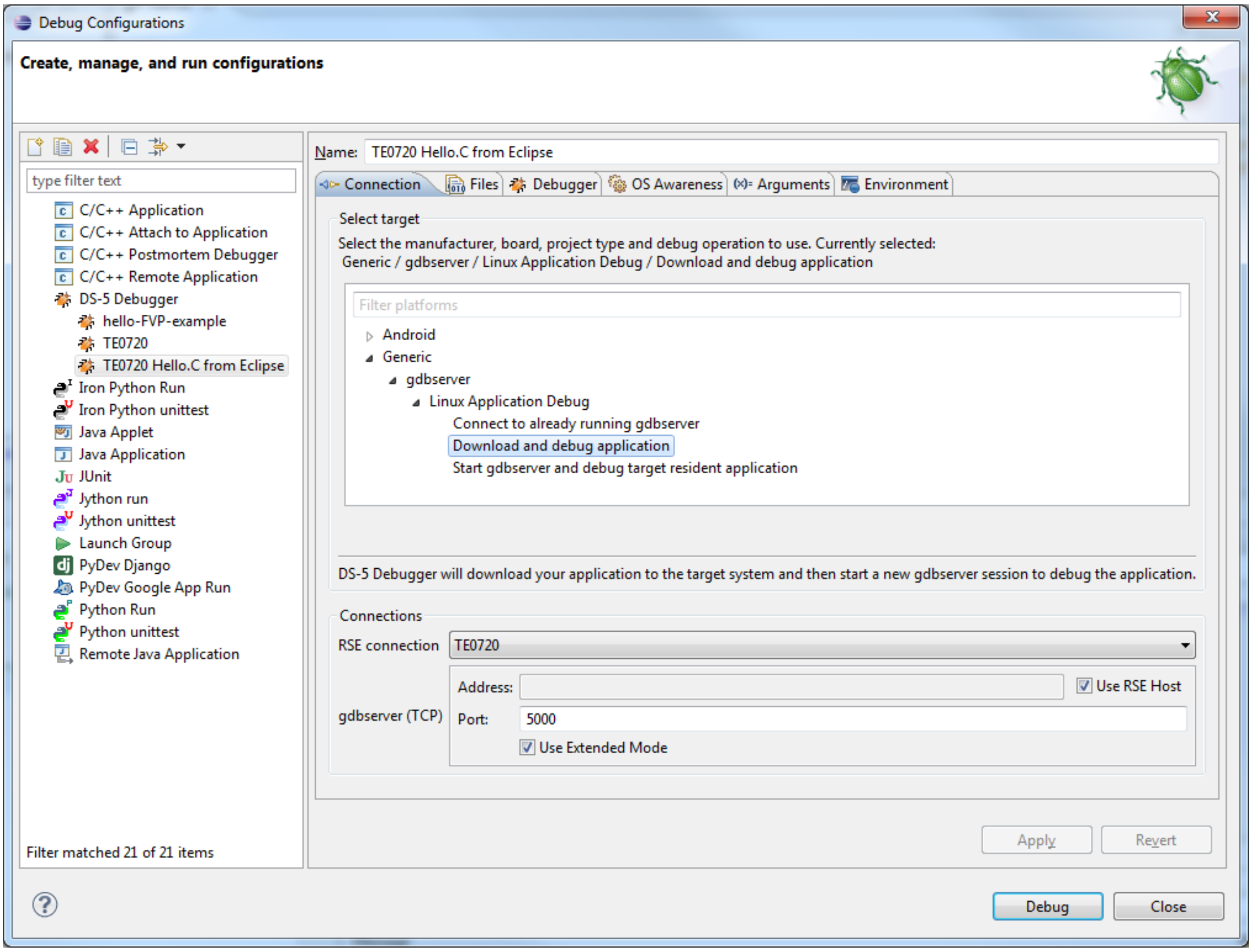
This command runs U-Boot from DDR memory.

ARM DS-5

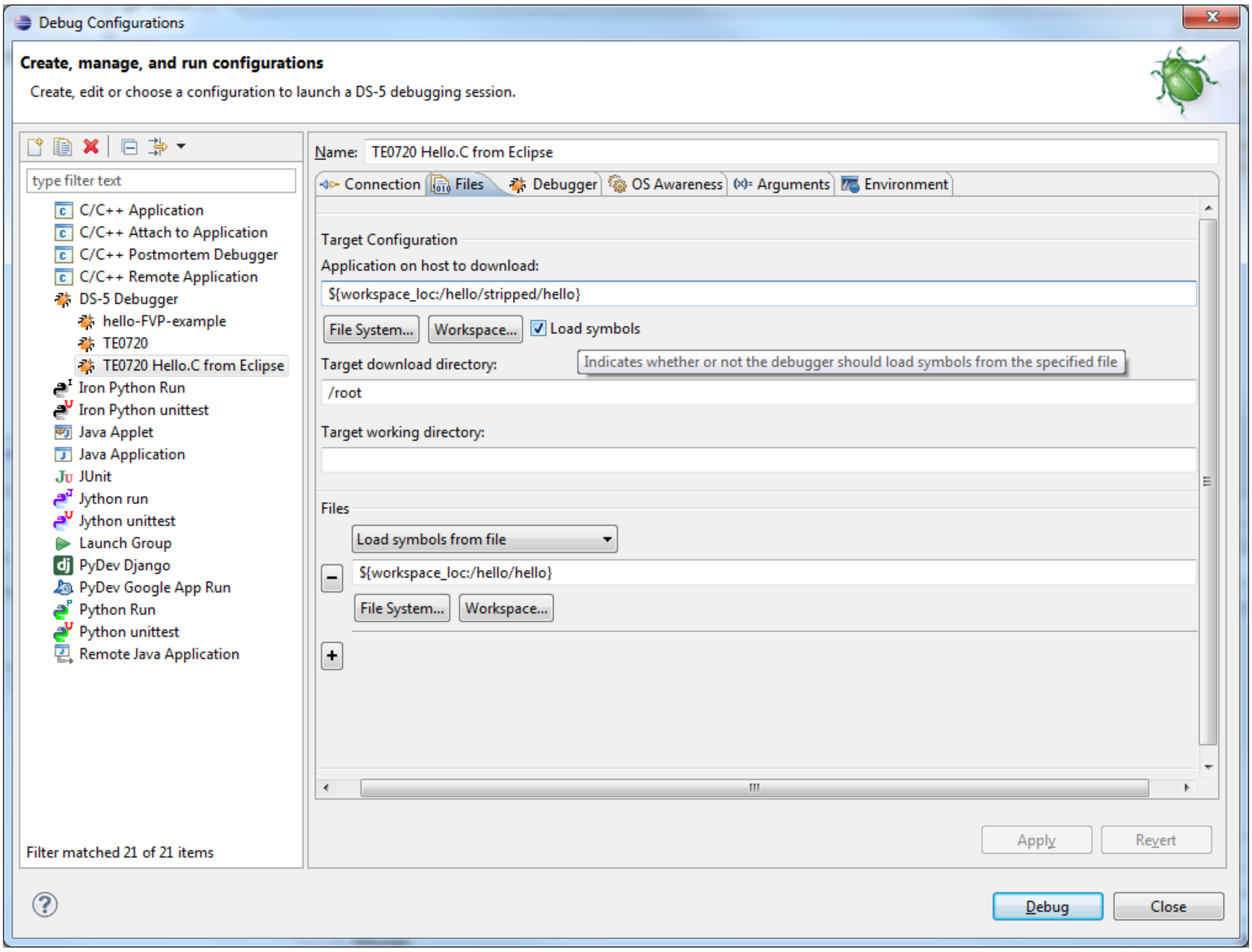
ARM offers a free and paid version of their development studio DS-5. The free version can be used for Linux application debug using gdbserver over Ethernet.

> ⚠ All PDF guides and other information at ARM and Xilinx websites about the use of DS-5 with Zynq appear to be out-dated. The screen-shots looks different, the instructions do not match, etc.

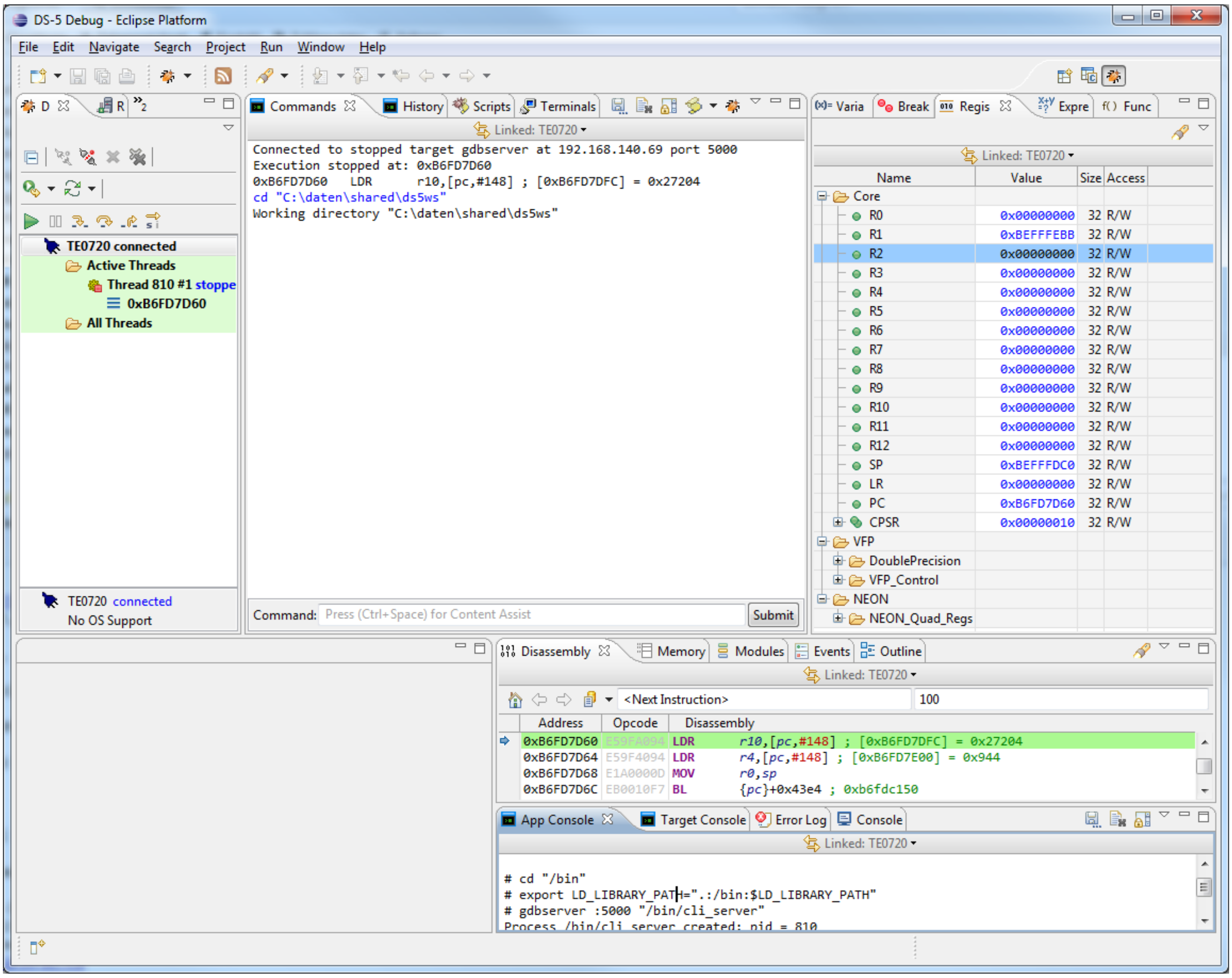
It is is possible to setup everything manually, but it takes time, and some steps need to be done manually each time the TE0720 is powered on. Basically you need to fix the IP addresses so that the TE0720 is visible and accessible from your PC host. Next you need to make sure gdbserver pre built executable is on the TE0720 file system and accessible (in newer builds it is included).



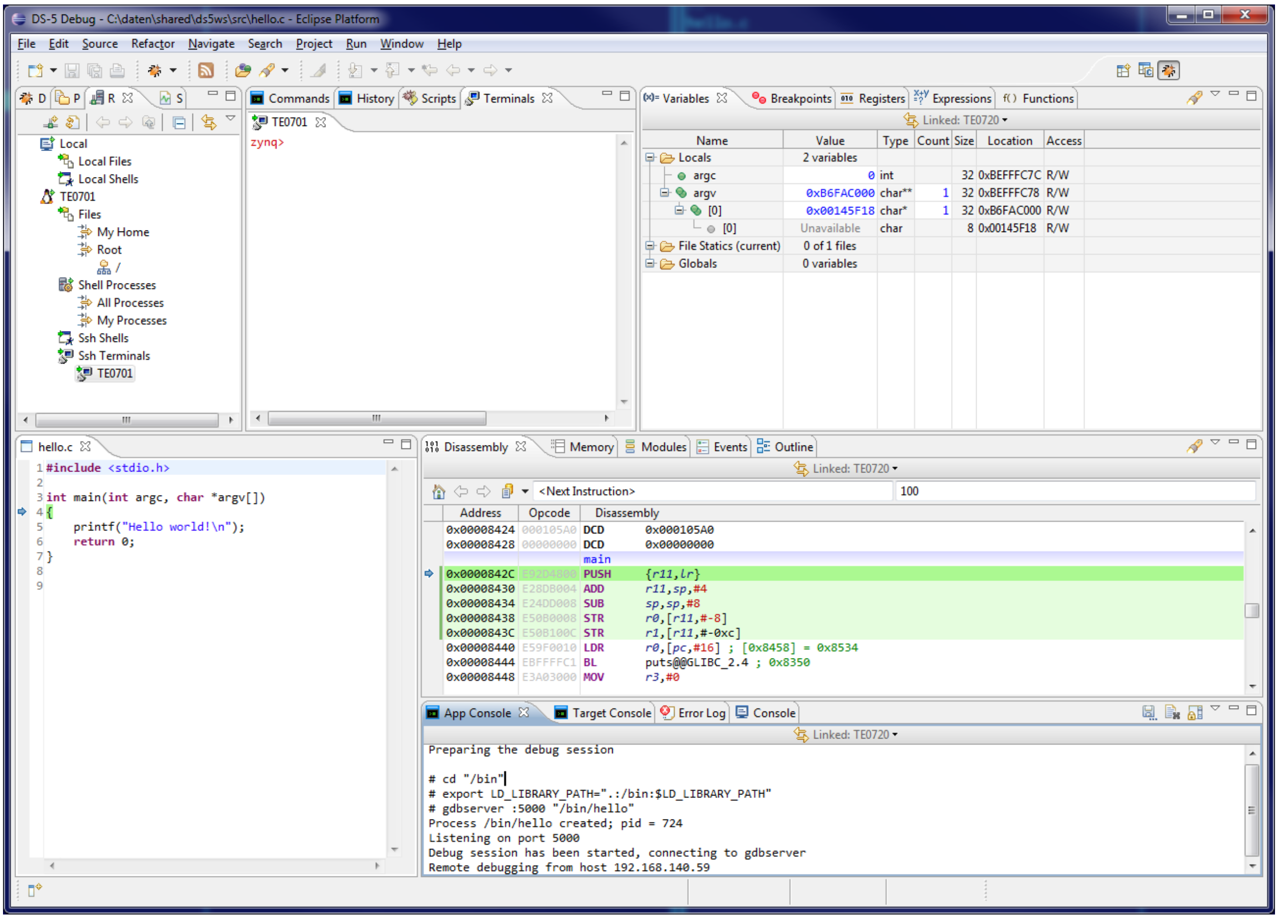
Select "Download and debug application", and RSE connection previously configured.



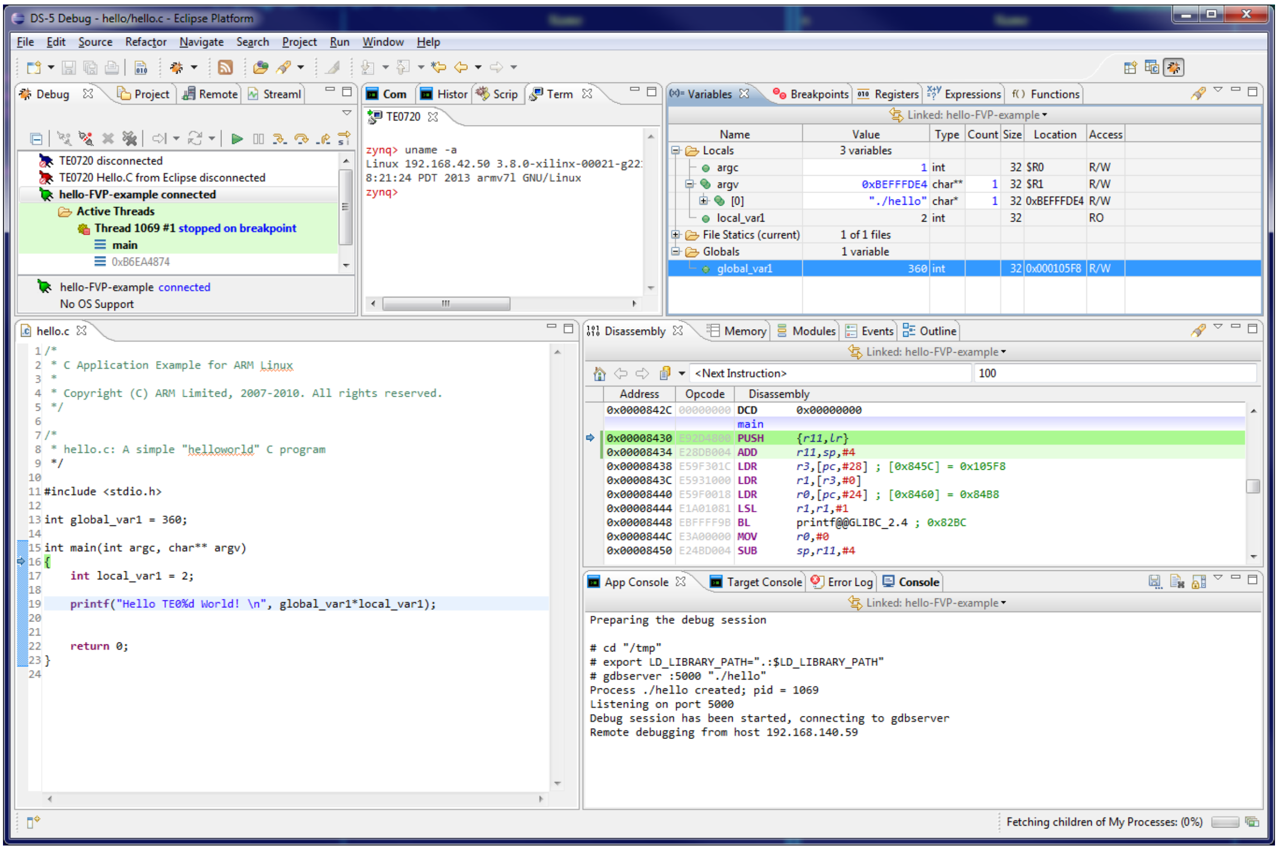
Very important, you must select /hello/stripped/hello as application to download and /hello/hello to load symbols from. If you forget the "stripped" then you get error launching gdbserver!



ARM DS-5 debugger view with active gdbserver connection to TE0720 on TE0701 carrier. TE0720 default firmware, gdbserver executable from ARM DS-5 distribution, no custom software needed to make the debugger connection.

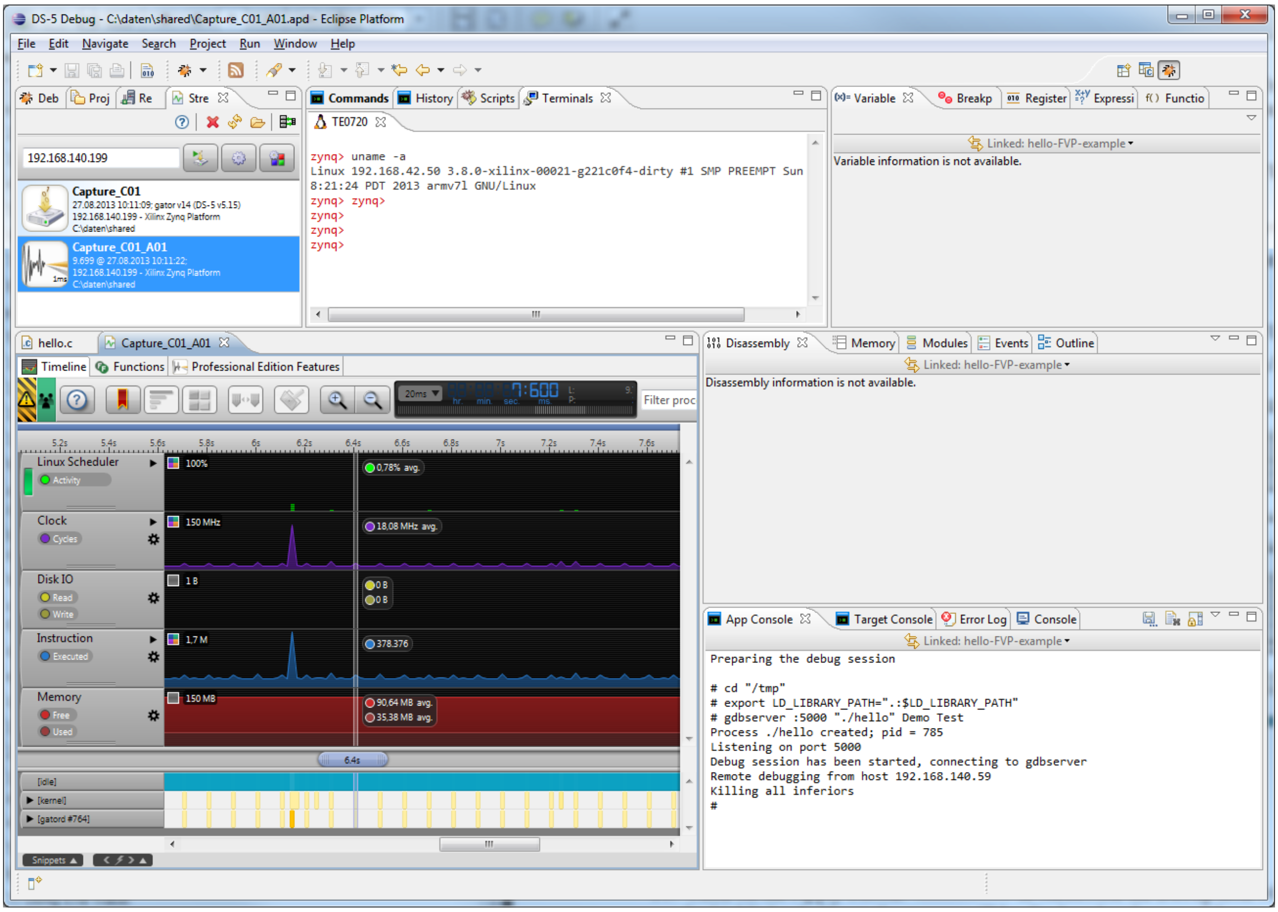


Debug session with symbols loaded and ssh console opened to the target.



Simple modified Helloworld example. This time compiled within DS-5 on a Windows machine and downloaded and executed on a TE0720.

Streamline



Streamline profiling on TE0720 - a simple screen of activity.

when using proper build all that is needed is to add init.sh to SD card

```
#!/bin/sh
ifconfig eth0 192.168.140.199
insmod /root/gator.ko
/bin/gatord &
```

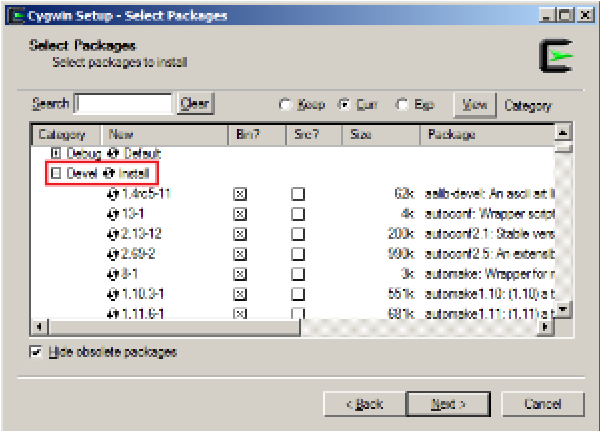
Example init.sh to set fixed IP address and start gator

Copyright © 2014 Trenz Electronic GmbH

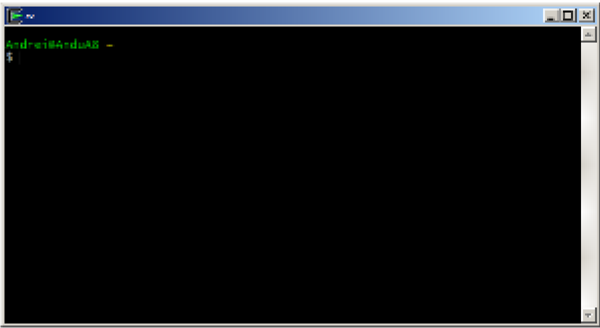
Compile/Install

Instructions for 64-bit Windows (i.e. always download 64-bit packages):

1. Install Cygwin from: http://cygwin.com/install.html
    a. In the step "Cygwin Setup - Select packages", select collection "Devel" and click on the word "Default" to change the installation type to "Install":

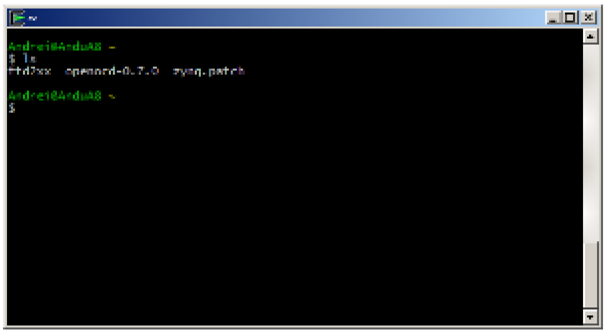


    b. Cygwin requests installation of additional packages to resolve dependencies - in this step, click "Next".
    c. Get a cup of tea of coffee as it takes a while to download all required packages.
2. On the start menu, click "Cygwin64 Terminal".



    a. The cygwin terminal opens:
    b. A home directory in the Cygwin installation is created automatically. The home directory can be found in the directory "C:\cygwin64\home\USERNAME" where "USERNAME" is your Windows username.
    c. Minimize the cygwin terminal to the taskbar.
3. Get the latest OpenOCD ZIP archive from: http://sourceforge.net/projects/openocd/files/openocd/ .
    a. Copy the contents to your home directory in the Cygwin installation.
    b. Thus, the OpenOCD sources should be in the directory "C:\cygwin64\home\USERNAME\openocd-0.7.0", where USERNAME is your Windows username and OpenOCD version is "0.7.0".
4. Get the latest FTDI D2XX drivers as a ZIP archive from http://www.ftdichip.com/Drivers/D2XX.htm .
    a. Copy the contents to your home directory in the Cygwin installation.
    b. Rename the long directory name "CDM v2.08.28 Certified" to "ftd2xx".
    c. Plug the FTDI JTAG dongle (TODO: is dongle name correct?) and install the drivers from this directory unless you have the drivers installed already.

5. Click here to download the zynq patch (origin: http://sourceforge.net/mailarchive/forum.php?thread_name=20130403122006.003E12432C%40openc) and save it in your home directory in the Cygwin installation.
6. Activate the cygwin terminal and verify that your home directory contains both directories, "ftd2xx" and "openocd-0.7.0" and the patch file "zynq.patch" by running command "ls":



7. Change the working directory to "openocd-0.7.0" by running the command "cd openocd-0.7.0".
8. Apply the zynq patch by running the command "patch -p1 < ../zynq.patch".
9. Run the configuration script with a following command:
   ./configure --enable-maintainer-mode --disable-werror --disable-shared --enable-ft2232_ftd2xx --build=i686-pc-linux-gnu --host=i686-w64-mingw32 --with-ftd2xx-win32-zipdir=$HOME/ftd2xx
   This will generate Makefile-s needed for the compilation of OpenOCD.
10. Compile OpenOCD with the command "make". This takes a few minutes.
11. Minimize the cygwin terminal
12. Executable "openocd.exe" can be found in the directory "C:\Cygwin64\home\USERNAME\openocd-0.7.0\src", where USERNAME is your Windows username.

Shorter alternative to steps 3... 11:

1. Click here to get the mkopenocd.sh skript and save it in your home directory in the Cygwin installation.
2. Activate the cygwin terminal and run the following commands:
   a. chmod +x mkopenocd.sh
   b. ./mkopenocd.sh

> *> halt*
>
> *number of cache level 1*
>
> *ZYNQ_PS.cpu cluster 0 core 0 multi core*
>
> *target state: halted*
>
> *target halted in ARM state due to debug-request, current mode: System*
>
> *cpsr: 0x6000015f pc: 0x0001df18*
>
> *MMU: disabled, D-Cache: disabled, I-Cache: disabled*
>
> *> mdw 0 4*

Copyright © 2014 Trenz Electronic GmbH

> *0x00000000: ea000023 ea00000d ea000010 ea00001c*
>
> *>*

Example session with openocd, target platform 7020 (non ES silicon) on TE0720 using TE0701 on board JTAG circuitry.

```
interface ft2232
ft2232_layout "digilent-hs1"
ft2232_latency 2
ft2232_device_desc "Digilent USB Device"
ft2232_vid_pid 0x0403 0x6010

adapter_khz 10000
set _CHIPNAME ZYNQ_PS
set _TARGETNAME $_CHIPNAME.cpu

jtag newtap ZYNQ_PL bs -irlen 6 -ircapture 0x1 -irmask 0x03 -expected-id 0x23727093
jtag newtap $_CHIPNAME dap -irlen 4 -ircapture 0x1 -irmask 0xf -expected-id 0x4ba00477
target create $_TARGETNAME cortex_a8 -chain-position $_CHIPNAME.dap -coreid 0
$_TARGETNAME configure -work-area-phys 0x0 -work-area-size 0x1000
```

Example config for openocd, the above does not support SRST ARM reset signal, openocd patch is available to support reset assertion.

Name	Version	Date
image2013-7-26 21:54:20.png	1	2013-07-26 19:54
image2013-7-26 21:54:57.png	1	2013-07-26 19:54
image2013-7-26 21:55:30.png	1	2013-07-26 19:55
image2013-7-26 21:55:58.png	1	2013-07-26 19:55
image2013-7-26 21:56:23.png	1	2013-07-26 19:56
image2013-7-26 21:57:25.png	1	2013-07-26 19:57
image2013-7-26 22:0:7.png	1	2013-07-26 19:59
image2013-7-26 22:12:14.png	1	2013-07-26 20:12
image2013-7-26 22:12:46.png	1	2013-07-26 20:12
image2013-7-26 22:13:21.png	1	2013-07-26 20:13
image2013-7-26 22:13:36.png	1	2013-07-26 20:13
image2013-7-26 22:14:52.png	1	2013-07-26 20:14
image2013-7-26 22:28:12.png	1	2013-07-26 20:28
image2013-7-26 22:3:46.png	1	2013-07-26 20:03
image2013-7-28 10:39:37.png	1	2013-07-28 08:39

Name	Version	Date
image2013-7-28 10:5:6.png	1	2013-07-28 08:05
image2013-7-28 10:6:22.png	1	2013-07-28 08:06
image2013-7-28 9:38:36.png	1	2013-07-28 07:38
mkopenocd.sh	1	2013-07-28 22:27
zynq.patch	1	2013-07-28 21:40

Copyright © 2014 Trenz Electronic GmbH

DCC Console

DCC Console allows uart console style debugging over ARM JTAG.

There are no bare-metal examples but Xilinx u-boot repository can be checked for the DCC console driver implementation. For experiments there are pre-compiled u-boot-dcc images within Xilinx SDK installation (location depends on the version of installed Xilinx tools).

Xilinx precompiled OCM RAM u-boot images

C:\Xilinx\Vivado\2014.2\data\xicom\cfgmem\uboot

06/10/2014 08:35 PM 582,145 nand_16.bin
06/10/2014 08:34 PM 581,977 nand_8.bin
06/10/2014 08:34 PM 419,444 nor.bin
06/10/2014 08:34 PM 472,007 qspi_dual_parallel.bin
06/10/2014 08:34 PM 472,007 qspi_dual_stacked.bin
06/10/2014 08:35 PM 472,271 qspi_single.bin

The above images are compiled version of u-boot from Xilinx git repository. They are just renamed from BIN to ELF. It is possible to execute them on TE0720 as well from SDK debugger. The one of interest is sqpi_single it allows u-boot to access QSPI flash on TE0720.

The only requirement is that OCM RAM blocks must be remapped to high memory, this is where the u-boot object is loaded.

Write 0x00000000F to OCM_CFG register (at 0xF8000910) before loading the uboot image into OCM RAM. When executed the u-boot will send console over ARM DCC channel to Xilinx debugger terminal.

Handling and usage precautions

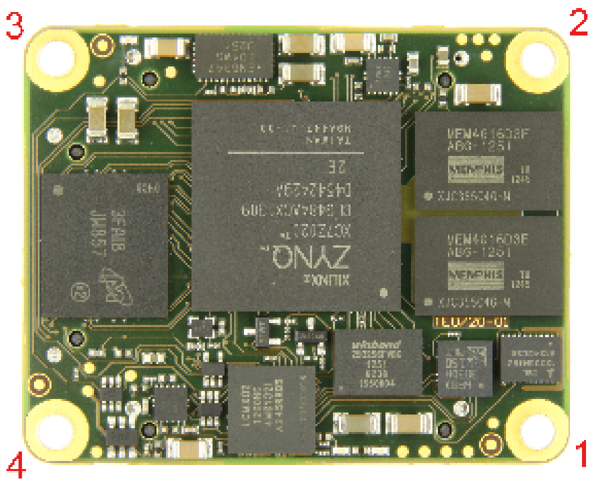
General

- Unpack and handle the module in an ESD safe workplace only.
- Keep the module away from moisture and dust.
- Remove power completely before plugging the module onto or unplugging it from a carrier board.
- Do not apply any voltage to any pin, when the module is not powered.
- Do not apply any voltage to any I/O pin, when the Power Good signal of the module is not active.

Removal Instructions

Samtec LSHM connectors are shock-proof and vibrations resistant, and have therefore high mating forces. It is recommended to remove the module via the mounting holes. When mounted with distance holders, unscrew the screws on the baseboard bottom about 2 mm, then press the baseboard equally (e.g. on a desktop) to lift the module. Repeat this until the module is unplugged [see Video below]. If this is not possible, you have to use a soft lever.

> Important: start by pulling corners 3 and 4, then pull corners 1 and 2!



Take a soft lever, like a plastic pen, and start lifting the module at position 1 about 1 mm, then proceed with position 2.

> You must lift positions 1 and 2 at least 1-1.5 mm before proceeding to lift positions 3 and 4.

Lift positions 3 and 4 a few millimeters. Repeat with positions 1 and 2 until the module is unplugged.

Failure to follow this procedure will very likely cause the left connector to break on the base board, as it is very hard to apply a controlled pull force by hand.

Winbond 32MByte SPI Flash in 2013.4

TE0720 on-board flash can be programmed directly only with SDK Flash Programmer starting from Vivado 2013.4 version. Previous versions can not access the flash properly. Please notice that ISE/Impact 14.7 do not support Winbond 32MB, only SDK 2013.4 does!

FSBL generated with SDK 2013.4 may not boot if some error is detected, in such cases it forces bootrom fall-back in a way that may make the Zynq device to appear as "broken"!

This is not related to any hardware issues, TE0720 is alive, ZYNQ is alive, and correct operation can be recovered.

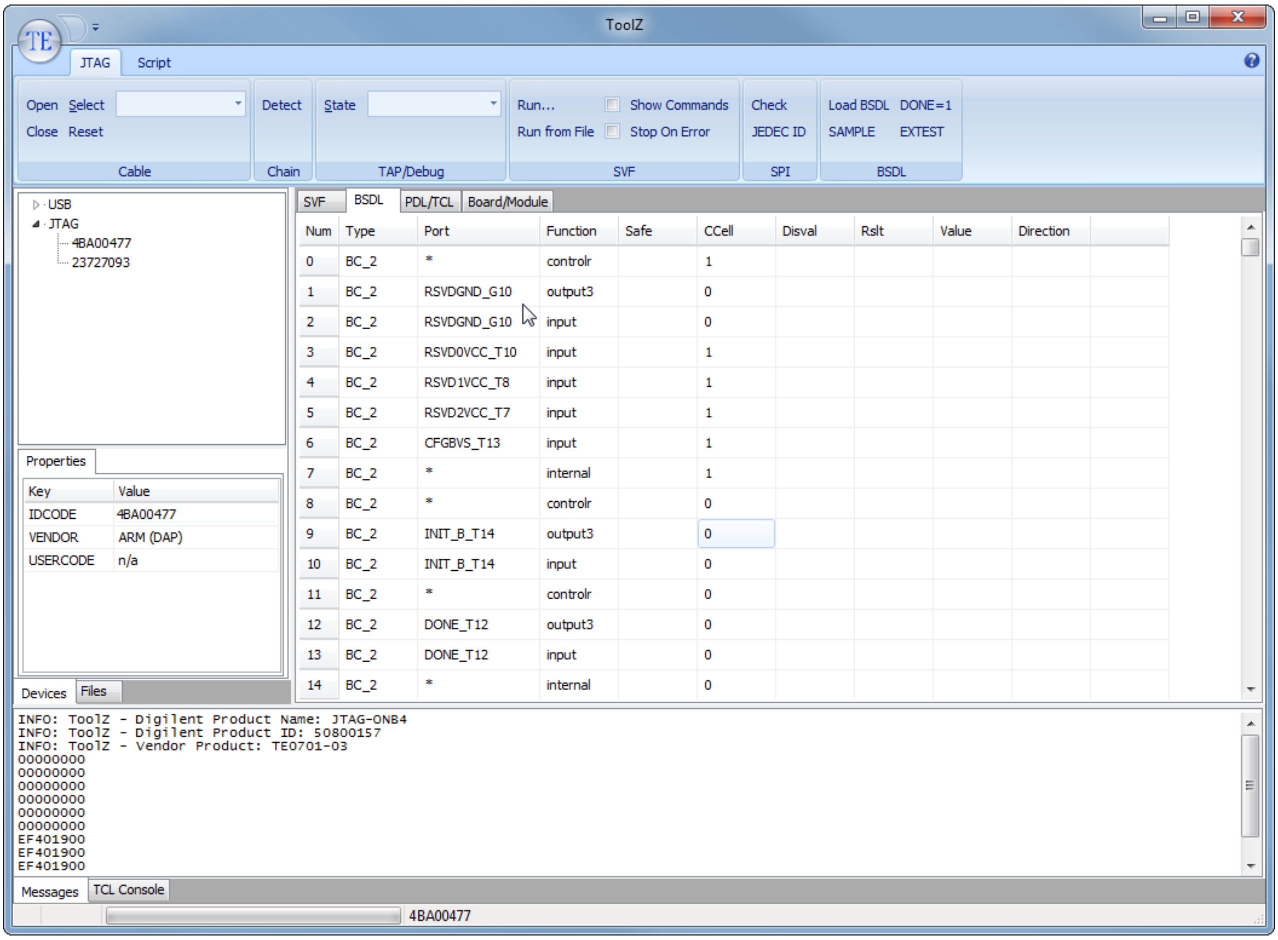
Problem description:

If SPI flash is programmed with 2013.4 version of FSBL, and if this FSBL detects some error conditions, then following happens: After POR Reset, ARM DAP TAP "disappears" from the chain. JTAG chain will look like it has only one device, that returns ZYNQ FPGA IDCODE on JTAG discovery.

JTAG Command IDCODE will return garbage. JTAG boundary scan functions are not available either. As long as this condition persists ZYNQ is completely not accessible from JTAG.

After some time (more than 10 seconds, this may however depend on the SPI flash contents) - ZYNQ bootrom gives up, and both JTAG TAPS re-appear in the JTAG Chain again. JTAG IDCODE command works now again, JTAG boundary scan is accessible, and ARM DAP is accessible as well. FPGA Configuration interface is however DISABLED, internal INIT is not released, and Impact or any other download tool can not Configure the FPGA Fabric over JTAG.

Copyright © 2014 Trenz Electronic GmbH



This is how BOOTROM locked device looks (after the bootrom timeout), INIT_B is set as output driving 0 from the ZYNQ device (both CONTROLR and OUTPUT3 bits are 0).

Boundary scan still works, in the screen-shot above "EF401900" is the JEDEC ID from the SPI Flash on TE0720 read using boundary scan (while ZYNQ is in bootrom locked error state).

Recovery Instructions:

On TE0701 insert SD Card, power-up, remove SD-Card, press Reset button, then reprogram Flash with known good image using SDK 2013.4 Flash Programmer.

If only available boot mode is SPI Flash boot then after the bootrom timeout (when 2 devices are back in JTAG chain) SDK Flash Programmer can reprogram the Flash.

Legal Notices

Document Warranty

The material contained in this document is provided “as is” and is subject to being changed at any time without notice. Trenz Electronic does not warrant the accuracy and completeness of the materials in this document. Further, to the maximum extent permitted by applicable law, Trenz Electronic disclaims all warranties, either express or implied, with regard to this document and any information contained herein, including but not limited to the implied warranties of merchantability, fitness for a particular purpose or non infringement of intellectual property. Trenz Electronic shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein.

Limitation of Liability

In no event will Trenz Electronic, its suppliers, or other third parties mentioned in this document be liable for any damages whatsoever (including, without limitation, those resulting from lost profits, lost data or business interruption) arising out of the use, inability to use, or the results of use of this document, any documents linked to this document, or the materials or information contained at any or all such documents. If your use of the materials or information from this document results in the need for servicing, repair or correction of equipment or data, you assume all costs thereof.

Copyright Notice

No part of this manual may be reproduced in any form or by any means (including electronic storage and retrieval or translation into a foreign language) without prior agreement and written consent from Trenz Electronic.

Technology Licenses

The hardware / firmware / software described in this document are furnished under a license and may be used /modified / copied only in accordance with the terms of such license.

Environmental protection

To confront directly with the responsibility toward the environment, the global community and eventually also oneself. Such a resolution should be integral part not only of everybody's life. Also enterprises shall be conscious of their social responsibility and contribute to the preservation of our common living space. That is why Trenz Electronic invests in the protection of our Environment.

REACH (Registration, Evaluation, Authorisation and Restriction of Chemicals) compliance statement

Trenz Electronic is a manufacturer and a distributor of electronic products. It is therefore a so called downstream user in the sense of REACH. The products we supply to you are solely non-chemical products (goods). Moreover and under normal and reasonably foreseeable circumstances of application, the goods supplied to you shall not release any substance. For that, Trenz Electronic is obliged to neither register nor to provide safety data sheet.

According to present knowledge and to best of our knowledge, no SVHC (Substances of Very High Concern) on the Candidate List are contained in our products.

Furthermore, we will immediately and unsolicited inform our customers in compliance with REACH - Article 33 if any substance present in our goods (above a concentration of 0,1 % weight by weight) will be classified as SVHC by the European Chemicals Agency (ECHA).

RoHS (Restriction of Hazardous Substances) compliance statement

Trenz Electronic GmbH herewith declares that all its products are developed, manufactured and distributed RoHS compliant.

WEEE (Waste Electrical and Electronic Equipment)

Information for users within the European Union in accordance with Directive 2002/96/EC of the European Parliament and of the Council of 27 January 2003 on waste electrical and electronic equipment (WEEE).

Users of electrical and electronic equipment in private households are required not to dispose of waste electrical and electronic equipment as unsorted municipal waste and to collect such waste electrical and electronic equipment separately. By the 13 August 2005, Member States shall have ensured that systems are set up allowing final holders and distributors to return waste electrical and electronic equipment at least free of charge. Member States shall ensure the availability and accessibility of the necessary collection facilities. Separate collection is the precondition to ensure specific treatment and recycling of waste electrical and electronic equipment and is necessary to achieve the chosen level of protection of human health and the environment in the European Union. Consumers have to actively contribute to the success of such collection and the return of waste electrical and electronic equipment.

Presence of hazardous substances in electrical and electronic equipment results in potential effects on the environment and human health. The symbol consisting of the crossed-out wheeled bin indicates separate collection for waste electrical and electronic equipment.

Trenz Electronic is registered under WEEE-Reg.-Nr. DE97922676.

Copyright © 2014 Trenz Electronic GmbH