HOCHSCHULE ULM

UNIVERSITY OF APPLIED SCIENCES

PROJECT BUMBLEBEE

HOWTO COLLECTION

Yvette Groner

Hannes Heinisch

Daniel Klitzke

Simon Krais

Christoph Löser

Christian Müller

30/01/2015

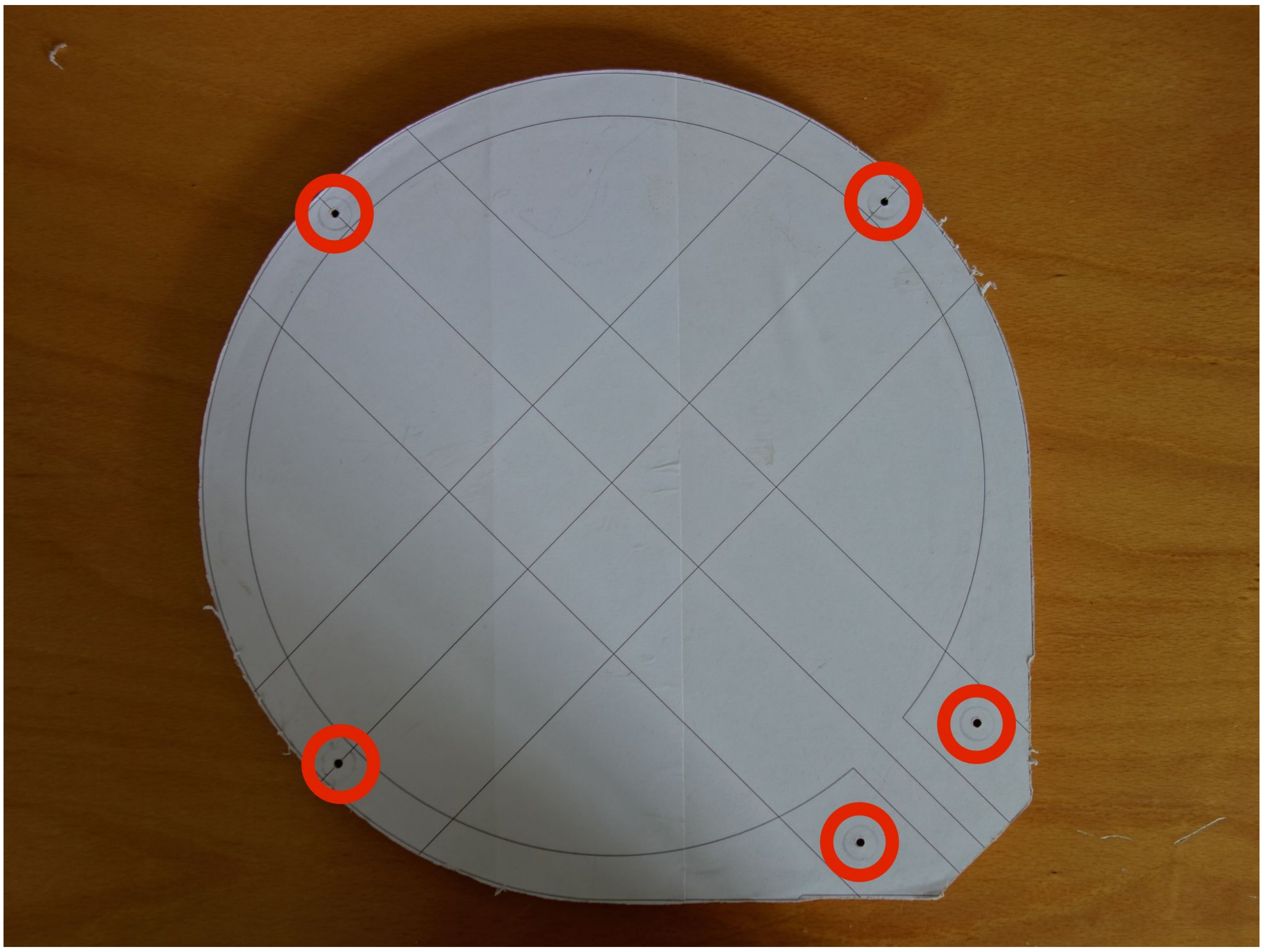
TABLE OF CONTENTS

1 Bumper Production

The bumpers consist of hard foam and can be easily replaced. To produce more of them, you first need a board of hard foam. In theory you can also choose any other material that's light and can be milled. We used these boards:

http://www.bauhaus.info/daemmplatten/ursa-xps-d-n-iii-l-hartschaumplatte-/p/13893056?continueUrl=/waermedaemmung/c/10000846?q%3D%253Arelevance%26show%3DPage%26page%3D4%26pageSize%3D12%26view%3DGallery&activeCategory=10000846

One of these boards is big enough that you get three bumpers out of it.

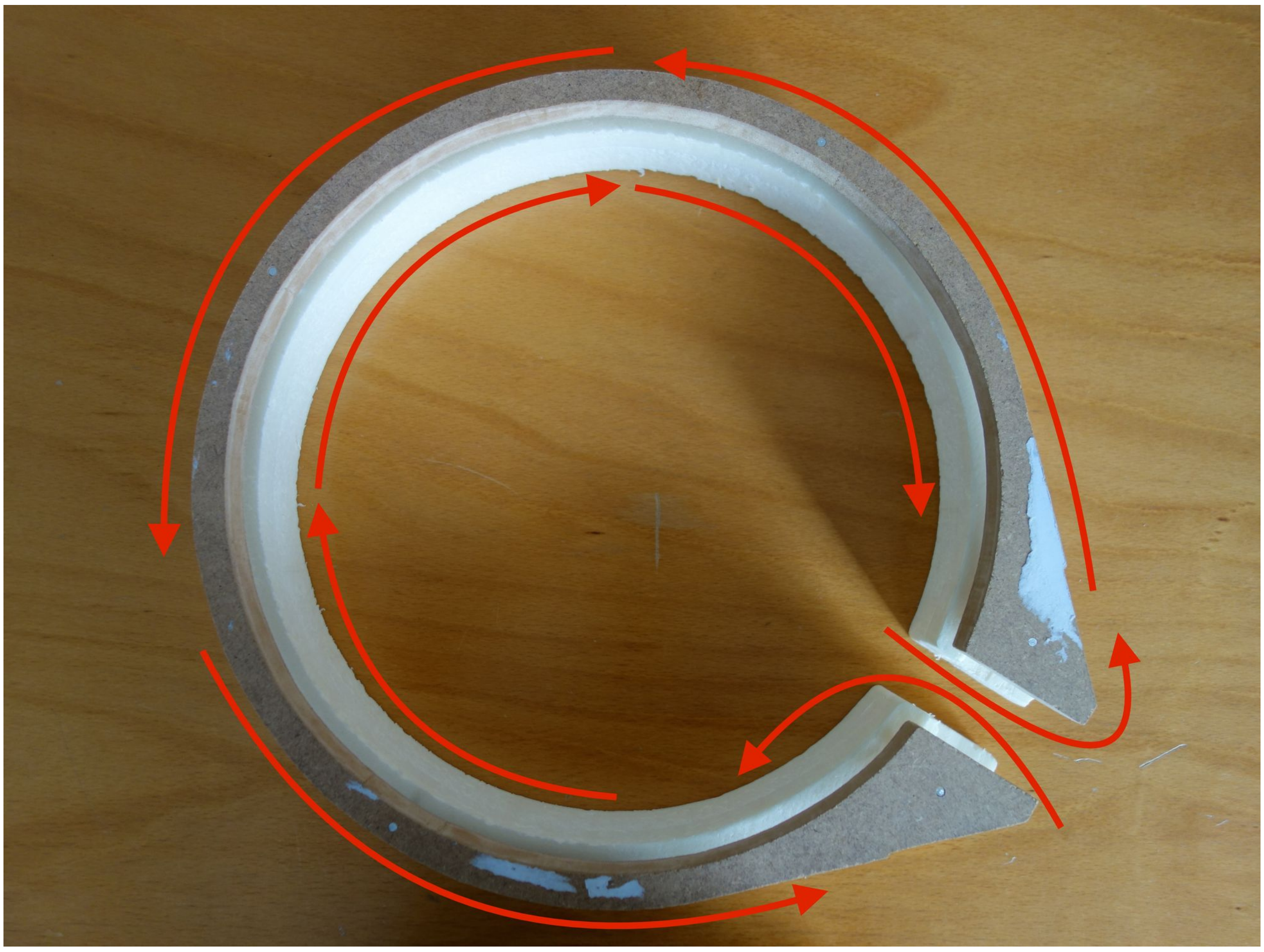


Graphic 1: Drilling pattern

1.) Place the first pattern on the board. The finished Part will be exactly as big as the pattern. Make sure that it is slightly fixed to prevent movement while drilling the marked holes (4mm).

2.) Remove the first pattern and attach the second one with the pins facing towards the board. The pins should fit smoothly into the drilled holes.

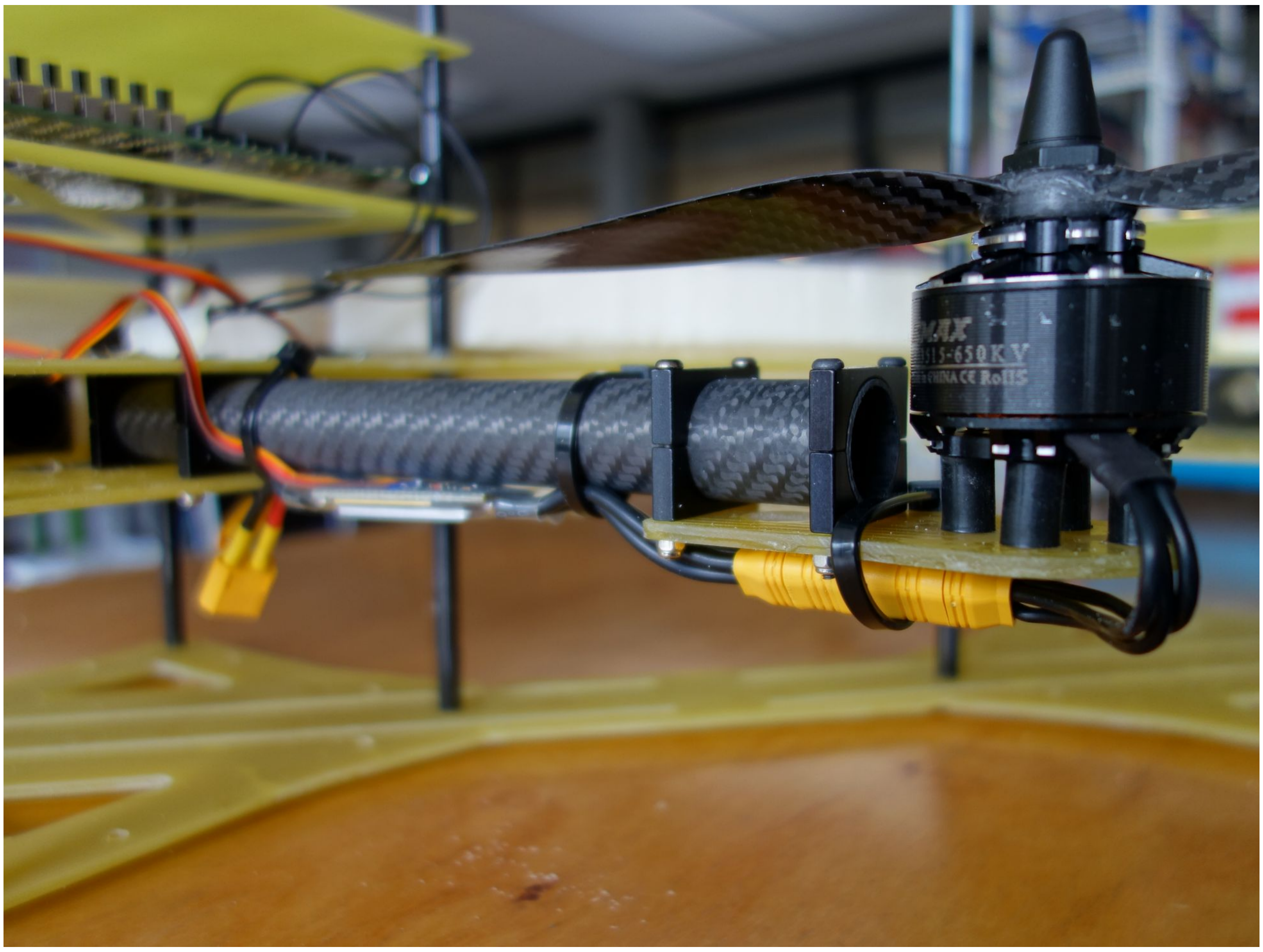
Note: The second pattern is 3mm smaller on each side. That was needed to fit our milling machine. If your machine is different, you have to produce your own pattern!


Graphic 2: Milling pattern

3.) Take your milling machine and work your way along the pattern in the shown direction and stabilize the machine by using the wooden cutouts. When moving in the wrong direction the material will melt which leads to unwanted marks on the surface. They are hard to remove afterwards and should be avoided. After finishing one side, remove the pattern and turn around the board. Insert the pattern again and repeat.

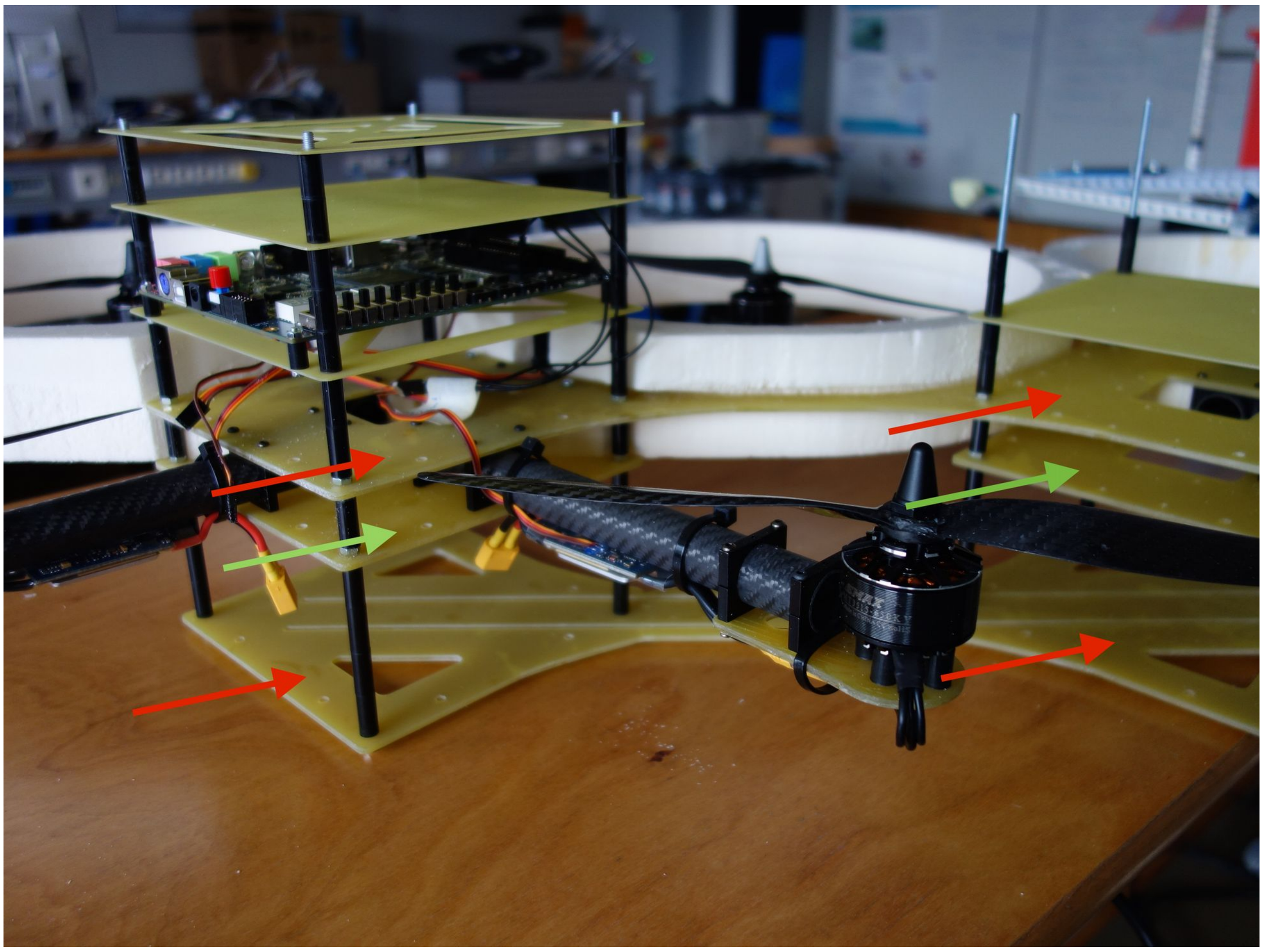
2 Motor and Controller Assembly

When mounting the motors and the controllers make sure to place the controllers somewhere in the air-flow to ensure proper cooling. Use zip-ties or something similar to fix them.


Graphic 3: Motor and controller assembly

3 From Six to Four Rotors

If you need a smaller model or have a low payload you can rebuild the frame to get a four rotor design. To do that you just have to replace the parts shown in the picture below (red) with the ones marked green.

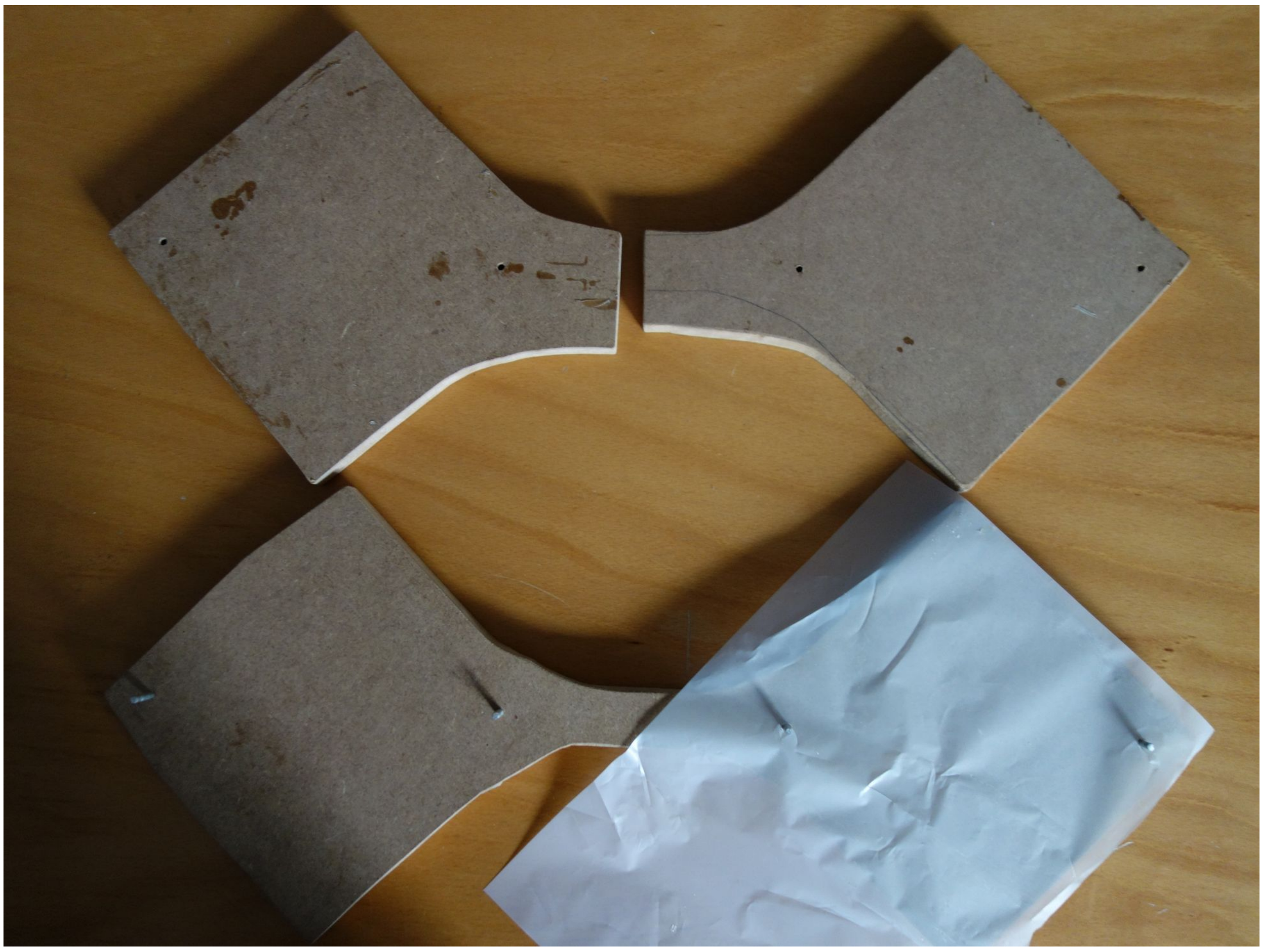


Graphic 4: Marked frame parts

4 Frame Parts Manufacturing

If you need additional parts, please contact Mr. Kroner (HS-Böfingen) regarding the milling machine and Mr. Kubat (HS-Ulm) regarding the material. The required files can be found in the folder "Frame_Parts".

To glue the different parts together we use a simple construction to ensure proper alignment. Without the help of small pins both sides would slightly change position while applying force.

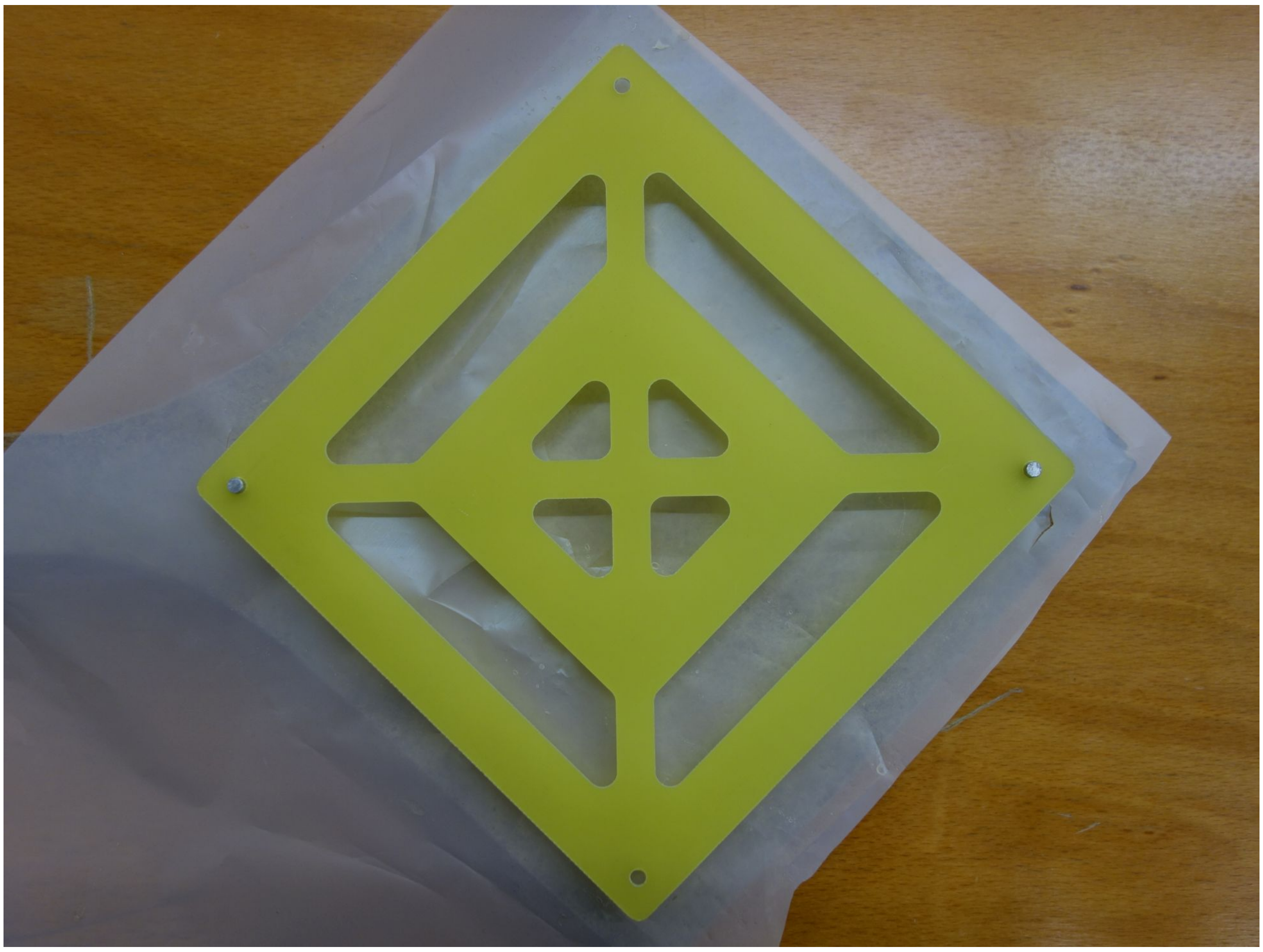


Graphic 5: Glue-tool with Teflon foil on one side.

This tool can be used for most parts. When gluing smaller parts together they have to be handled separately.

When gluing two parts together, make sure that all surfaces are clean. Apply the Teflon foil first, before putting the first parts in. Make sure that all parts fit together as intended first! When ready, prepare the glue and apply a thin layer equally on the whole surface. When working with the bigger frame parts, it is a good idea to ask a second person for help.

Then quickly place the other half on top and add another layer of Teflon foil. Finally add the two wooden counterparts on top and fix everything up with screw-clamps.


Graphic 6: Glue-tool with one part.

5 DigiView

DigiView tool is very useful tool to analyze some signals of a bus or wire in general. This how-to should explain some simple steps to use DigiView. Therefor a I2C connection and a PWM signal should be analyzed.

5.1 Setup the Device

Mainly the DV3400 device is used in the laboratory. To set it up you just need to connect it via an USB cable with a computer. Also the device needs to be connected to a power supply. After that you can turn it on.

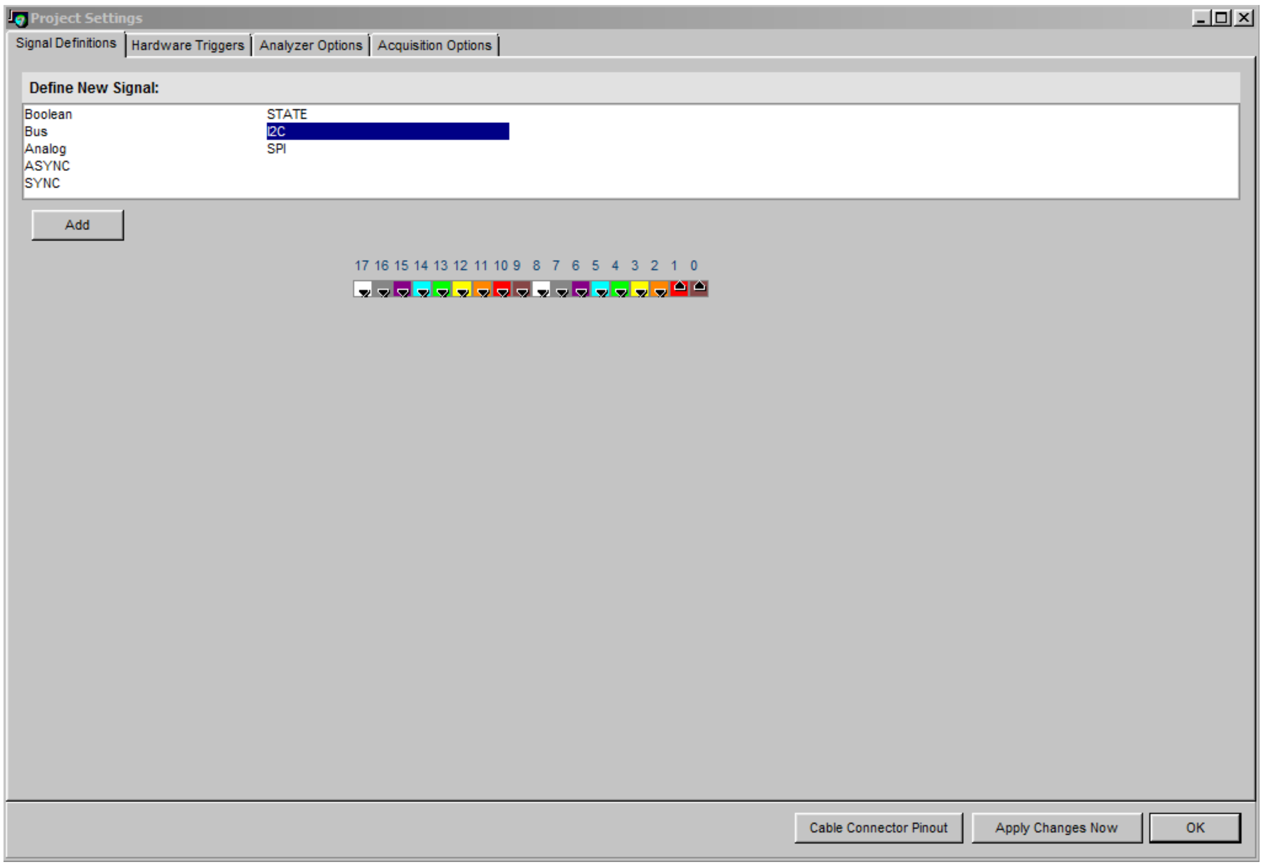
5.2 Setup the Software

The Software "DigiView“ is already installed on the computers in the laboratory. After starting the program, you can decide if you want to create a new configuration or use a stored configuration. We always created a new configuration. Set it up with the device DV3400 and a mode of 400Mhz and 18 Channels. Normally 18 channels, that means one of the connection at the device, are enough. Finally click "Yes" and the program is ready.

5.3 Configure the Signals

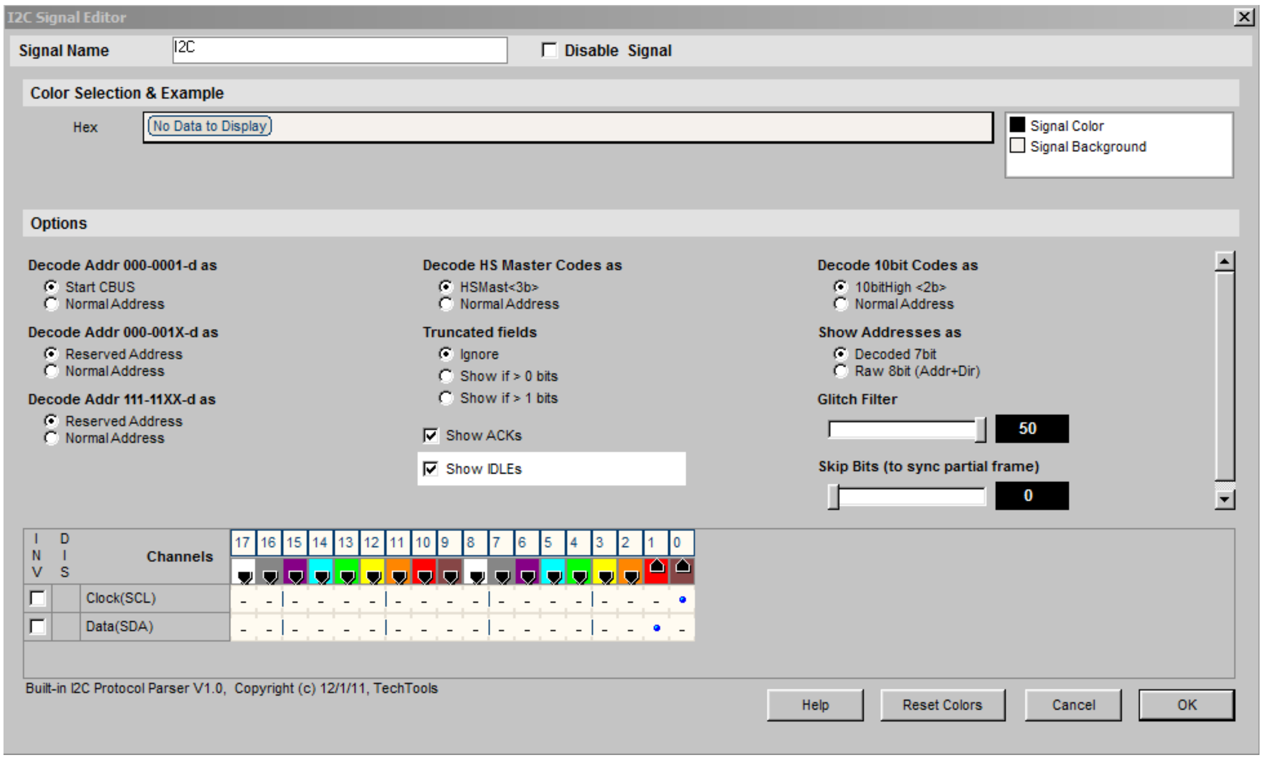
With a new configuration the default bus has all 18 Channels. That is most of the time not really helpful. That is why we create new specifications for our individual usage. The configuration to manage the signals can be opened in the tab "Config“ > "Signals“. There delete the existing bus, to avoid confusion during measurements.

To add a new signal, just select at the top a specification like I2C and click add. A window with all special setups for this specification opens.



Graphic 7: Signal management

For the I2C standard, it is helpful to enable the “Show ACKs” and “Show IDLEs”. This lets the program print an “A” for the acknowledge flag in the measurement to find some mistakes quicker. Then you have to define where the SCL and the SDA signals are connected to. In this example the SCL is connected with the brown wire and the SDA with the red one. After that you can create it with the “OK” button.



I2C Signal Editor

Signal Name: I2C   ☐ Disable Signal

Color Selection & Example

Hex: No Data to Display   ■ Signal Color ☐ Signal Background

Options

Decode Addr 000-0001-d as: [✓ Start CBUS] Normal Address

Decode Addr 000-001X-d as: [✓ Reserved Address] Normal Address

Decode Addr 111-11XX-d as: [✓ Reserved Address] Normal Address

Decode HS Master Codes as: [✓ HSMast<3b>] Normal Address

Truncated fields: [✓ Ignore] Show if > 0 bits, Show if > 1 bits

☑ Show ACKs

☑ Show IDLEs

Decode 10bit Codes as: [✓ 10bitHigh <2b>] Normal Address

Show Addresses as: [✓ Decoded 7bit] Raw 8bit (Addr+Dir)

Glitch Filter: 50

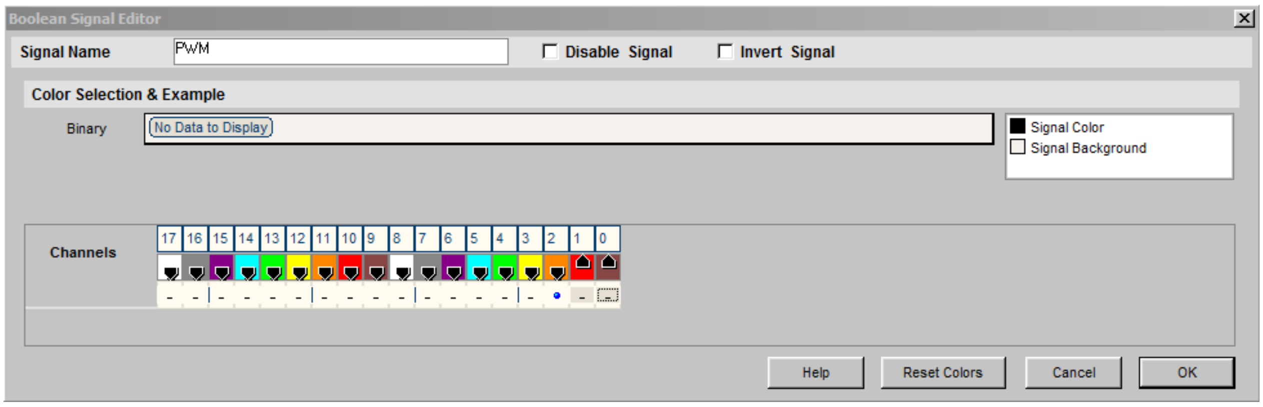
Skip Bits (to sync partial frame): 0

I N V	D I S	Channels	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
☐		Clock(SCL)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	●
☐		Data(SDA)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	●	-

Built-in I2C Protocol Parser V1.0, Copyright (c) 12/1/11, TechTools

Graphic 8: I2C configuration

For the PWM signal you should use a Boolean value. Therefore you can only define on which input pin it is connected to the signal. In the example it is the orange one.



Boolean Signal Editor

Signal Name: PWM   ☐ Disable Signal   ☐ Invert Signal

Color Selection & Example

Binary: No Data to Display   ■ Signal Color ☐ Signal Background

Channels: 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 (selected: 2)

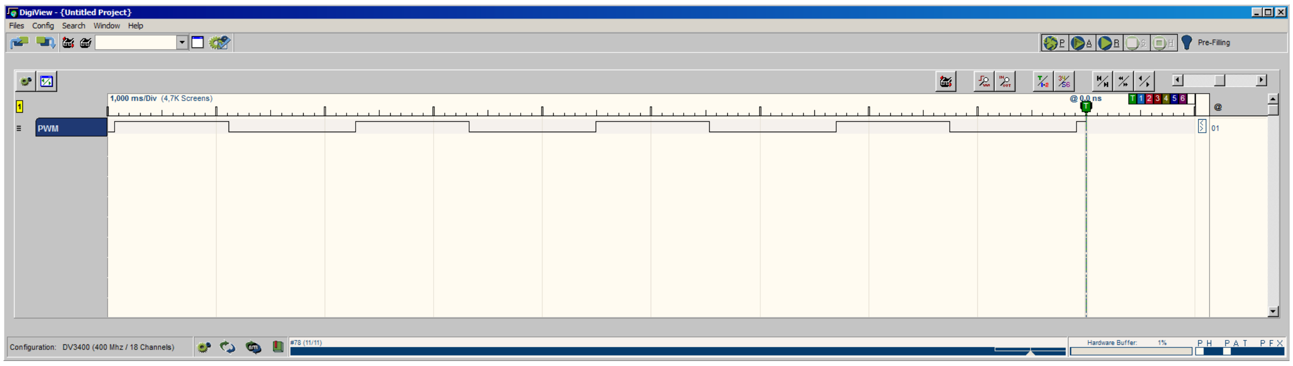
Graphic 9: PWM configuration

All signals are defined now. Next step is to analyze some signals.

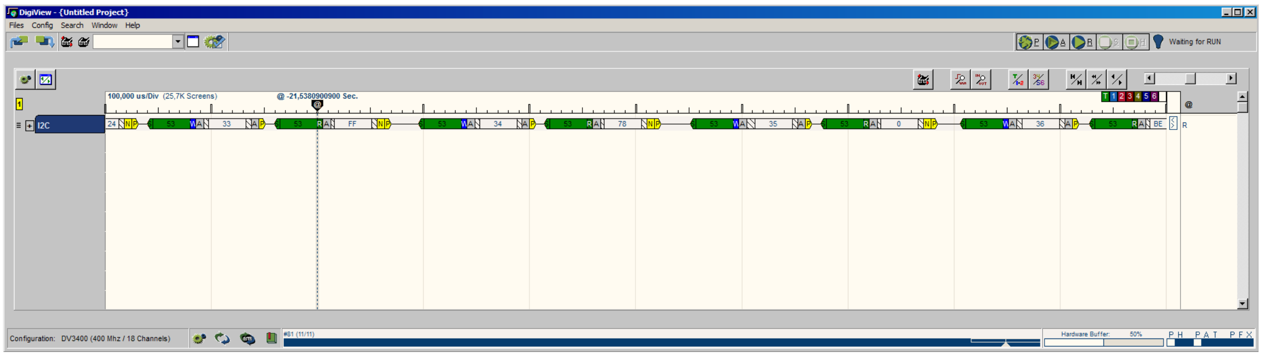
5.4 Analyze Signals

After preparing the device and setup the software you can start to analyze the signals. Therefore you just have to click on the “Run Once” button in the right upper corner. From this time the program records the signals. Then you can stop this process by clicking on the “Stop” button next to the button before.

The signals sequence is shown for the period between start and stop. For I2C it is a line which tells you the meaning of both lines SCL and SDA together. For a Boolean value you can for example measure the frequency.


DigiView - {Untitled Project}
Files Config Search Window Help
Pre-Filling
1,000 ms/Div (4,7K Screens)
@ 0,0 ns
PWM
Configuration: DV3400 (400 Mhz / 18 Channels)
Hardware Buffer: 1%

Graphic 10: Example for a PWM signal


DigiView - {Untitled Project}
Files Config Search Window Help
Waiting for RUN
100,000 us/Div (25,7K Screens)
@ -21,5380900900 Sec.
I2C
Configuration: DV3400 (400 Mhz / 18 Channels)
Hardware Buffer: 50%

Graphic 11: Example for an I2C transmission

5.5 Clearing the Window

It is possible to delete the measurements, if you want to do the next one. Therefore you only have to click in the lower left corner on the gear-wheel and then on *Clear Capture History*. The measurements will be deleted and you can start your next analysis.

Maybe you set up or delete some markers. Then you can right-click on the markers and choose *Clear Marker*. There you can delete just one of them or all together.

6 Linux ARM

6.1 Introduction

This document describes how to work with the Linux on the ARM processor of the DE1-SoC board. As host system a Linux machine is used. The following software is also required.

- Altera SoC Embedded Design Suite
- Altera Embedded Command Shell
- SSH / SFTP
- minicom

6.2 Start-up Script

On the system exists a start-up script file to configure the system after boot up. It is located at /home/root/scripts/startup.sh. At the moment it only sets the default IP address of the system to 192.168.100.2.

6.3 Connect with the System

There are two possible ways to connect with the Linux on the ARM processor of the DE1-SoC board. One way is via Ethernet connection and SSH, the other is the serial connection over the UART to USB port.

6.4 SSH and SFTP

To connect to the board via SSH, a direct Ethernet connection between board and computer is necessary. The network configuration for the host system are as followed:

IP address:	**192.168.100.1**
Subnet mask:	**255.255.255.0**

To log in following account is used:

Username:	**root**

Password: root

The network configuration sets the default IP address to 192.168.100.2 through the start-up script.

Now it is possible to connect to the board. With the following command

```
ssh root@192.168.100.2
```

you get access to a shell session.

With the command

```
sftp root@192.168.100.2
```

you can upload or download files.

To upload a file enter

```
put <path-to-file>
```

and to download a file enter

```
get <path-to-file>
```

6.5 Serial Connection via UART to USB

Another way to connect to the board is via the UART to USB port. Therefor minicom is used with the following configuration:

Device	/dev/ttyUSB0
Baud	115200 bps
Data bits	8
Parity bit	None
Stop bit	1
Hardware/Software flow control	off

To save this configuration do as follow

- Open a shell window
- Enter “sudo minicom -s sockit”
- Select “Serial port setup”
- Now you can select the parameter you want to change by enter the expected key for example A to

change the serial device

- After the configuration is done, press ESC to go back to the main menu.
- Here select “Save setup as sockit”.

If this procedure was successful you can simply open a connection by entering

sudo minicom sockit

To log in following account is used:

Username: root

Password: not required

This serial connection does not provide uploading and downloading from files. It only provides access to a shell session.

6.6 Cross-Compile Programs

To cross-compile programs for the Linux on the ARM system, start the Altera Embedded Command Shell by entering

<installation-path>/altera/13.1/embedded/embedded_command_shell.sh

The script sets the important path for the building tools and the libraries.

Now you can cross-compile your C programs by using the arm-linux-gnueabihf-gcc. For example

arm-linux-gnueabihf-gcc -o demoProg demoProg.c

will compile the demoProg.c for the ARM architecture.

6.7 Linux SD Card

The Linux SD card consists of two partitions, which can be mounted directly on a Linux system. The first partition is 20 MB large and contains the following files

File	Description
soc_system.rbf	Raw Binary File, contains the default SoPC configuration
socfpga.dtb	Device Tree Binary
u-boot.scr	U-boot script to configure FPGA
zImage	Compressed Linux kernel image

To replace a file you can do it through drag and drop.

The second partition is 1 GB large and homes the root partition. Here you can access all files of the Linux system.

The Device Tree Binary has to be replace if you want to connect new periphery with the ARM processor. To create a new .dtb file you need the .sopcinfo file of your project.

Open an embedded command shell and change to your project folder by entering

```
cd <your-project-path>

<installation-path>/altera/13.1/embedded/embedded_command_shell.sh
```

After that run the Device Tree Generator with

```
sopc2dts --input <your_sopcfile>.sopcinfo --output socfpga.dts --board soc_system_board_info.xml
--board hps_clock_info.xml
```

Now you have a Device Tree Source file which is needed to obtain the Device Tree Binary. Therefor enter

```
dtc -I dts -O dtb -o socfpga.dtb socfpga.dts
```

For more information about creating and modifying the Linux SD image, see the documentation on rock-etboards.org.

7 SoPC

7.1 Introduction

In this section it is described how to put the DE1-SoC into operation with the current SoPC. Therefore the following two tools are needed:

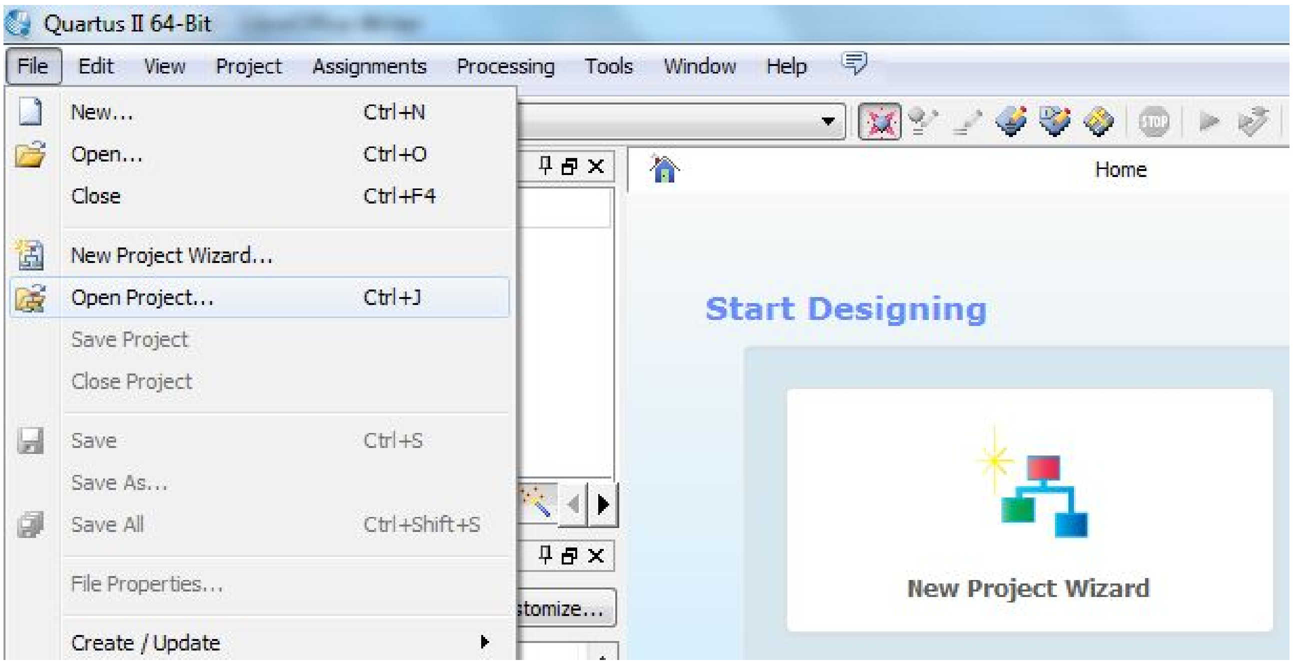
- Altera Quartus II (Web Edition or Subscription Edition)
- Altera QSYS

Also you should have the latest version of the SoPC which is currently (01/2015) NIOS_MCAPI_Base_v07.

7.2 Open Project

Before you open the project files you should first unzip the NIOS-MCAPI_Base_v07.zip into a folder onto your PC.

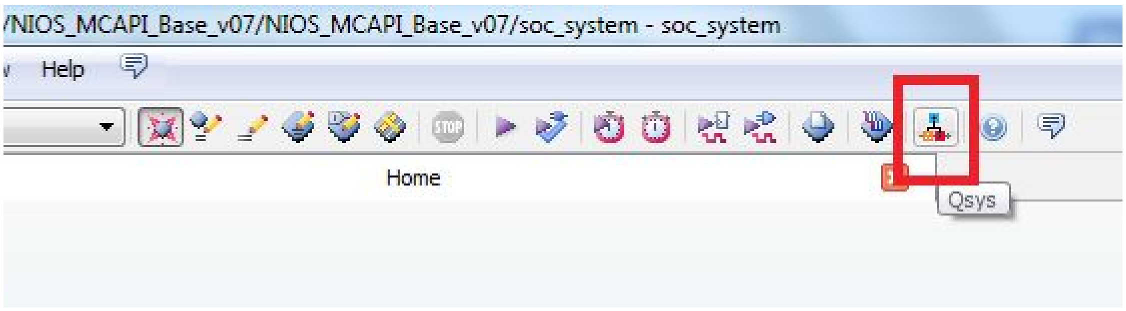
After that you should start up Quartus II. Then just click on File->Open Project....



Graphic 12: Quartus file menu

In the following dialogue you have to select the .qpf file in this example soc_system.qpf. This will open the Quartus II project.

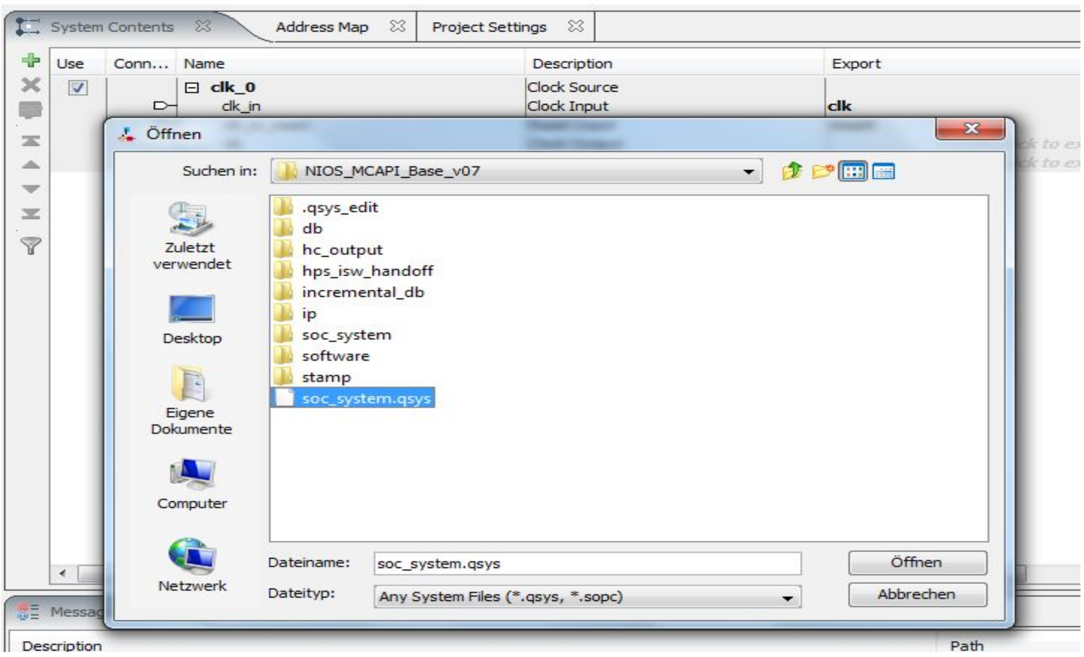
First we want to edit the SoPC. Therefore an extra tool called QSYS that is integrated into Quartus II is used. To start up QSYS you have to click on the icon shown below.


/NIOS_MCAPI_Base_v07/NIOS_MCAPI_Base_v07/soc_system - soc_system
Help
Home
Qsys

Graphic 13: Quartus icon bar

After that the splash screen of QSYS should show up and the program should be ready to use in a few seconds.

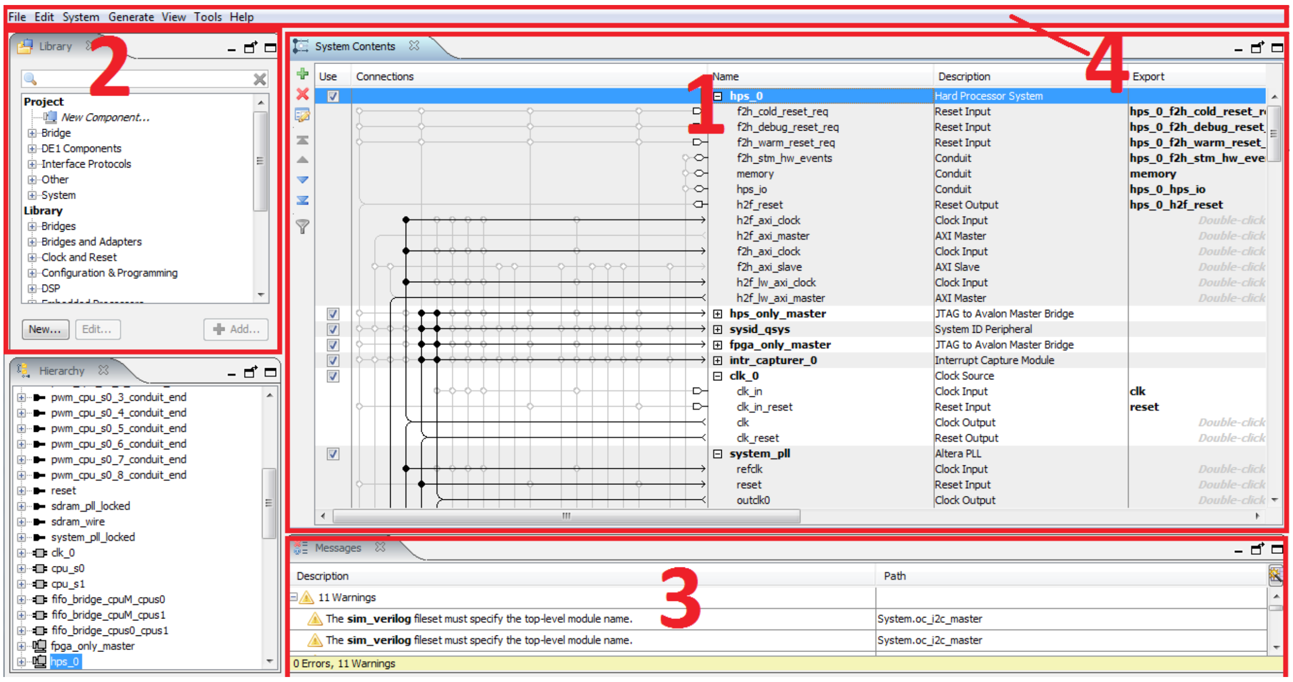
It shows an opening dialogue in which you are asked to select a .qsys file which is the QSYS project file. In our case the right file is the soc_system.qsys.



Graphic 14: SoPC open dialogue

After a click on the open button QSYS loads the project.

In the standard view of QSYS there are 4 essential areas that are important for the work with the system.



Use	Connections	Name	Description	Export
✓		hps_0	Hard Processor System	
		f2h_cold_reset_req	Reset Input	hps_0_f2h_cold_reset_r
		f2h_debug_reset_req	Reset Input	hps_0_f2h_debug_reset
		f2h_warm_reset_req	Reset Input	hps_0_f2h_warm_reset
		f2h_stm_hw_events	Conduit	hps_0_f2h_stm_hw_eve
		memory	Conduit	memory
		hps_io	Conduit	hps_0_hps_io
		h2f_reset	Reset Output	hps_0_h2f_reset
		h2f_axi_clock	Clock Input	Double-click
		h2f_axi_master	AXI Master	Double-click
		f2h_axi_clock	Clock Input	Double-click
		f2h_axi_slave	AXI Slave	Double-click
		h2f_lw_axi_clock	Clock Input	Double-click
		h2f_lw_axi_master	AXI Master	Double-click
✓		hps_only_master	JTAG to Avalon Master Bridge	
✓		sysid_qsys	System ID Peripheral	
✓		fpga_only_master	JTAG to Avalon Master Bridge	
✓		intr_capturer_0	Interrupt Capture Module	
✓		clk_0	Clock Source	
		clk_in	Clock Input	clk
		clk_in_reset	Reset Input	reset
		clk	Clock Output	Double-click
		clk_reset	Reset Output	Double-click
✓		system_pll	Altera PLL	
		refclk	Clock Input	Double-click
		reset	Reset Input	Double-click
		outclk0	Clock Output	Double-click

Hierarchy:
- pwm_cpu_s0_3_conduit_end
- pwm_cpu_s0_4_conduit_end
- pwm_cpu_s0_5_conduit_end
- pwm_cpu_s0_6_conduit_end
- pwm_cpu_s0_7_conduit_end
- pwm_cpu_s0_8_conduit_end
- reset
- sdram_pll_locked
- sdram_wire
- system_pll_locked
- clk_0
- cpu_s0
- cpu_s1
- fifo_bridge_cpuM_cpus0
- fifo_bridge_cpuM_cpus1
- fifo_bridge_cpus0_cpus1
- fpga_only_master
- hps_0

Description	Path
11 Warnings	
The **sim_verilog** fileset must specify the top-level module name.	System.oc_i2c_master
The **sim_verilog** fileset must specify the top-level module name.	System.oc_i2c_master

0 Errors, 11 Warnings

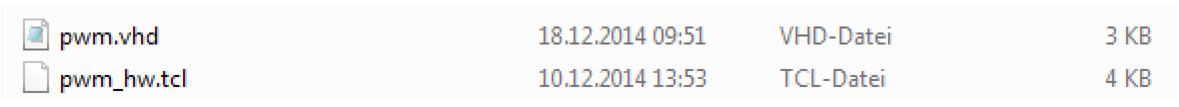
Graphic 15: QSYS overview

- Area 1 shows a list of the components contained by the system. It also visualizes the wiring between the components.
- Area 2 shows a list of the components that are available for the use in the project in QSYS.
- Area 3 shows warning messages and errors that are generated by QSYS.
- Area 4 is the menu bar that can be used to save, open and generate projects and much more.

7.3 Add IP-Core to Component Library

Often controllers that are needed for the project are not contained in the standard library of QSYS.
In this section it is described how to add a so called IP-Core to the library.
In this example a very simple IP-Core that implements a PWM controller is used.

First you should copy the needed files to a directory in you Quartus II project directory for example NIOS_MCAPI_Base_07/ip/pwm. If the directories don't exist in you project till now just create them. The files that are needed for our Component are shown below.

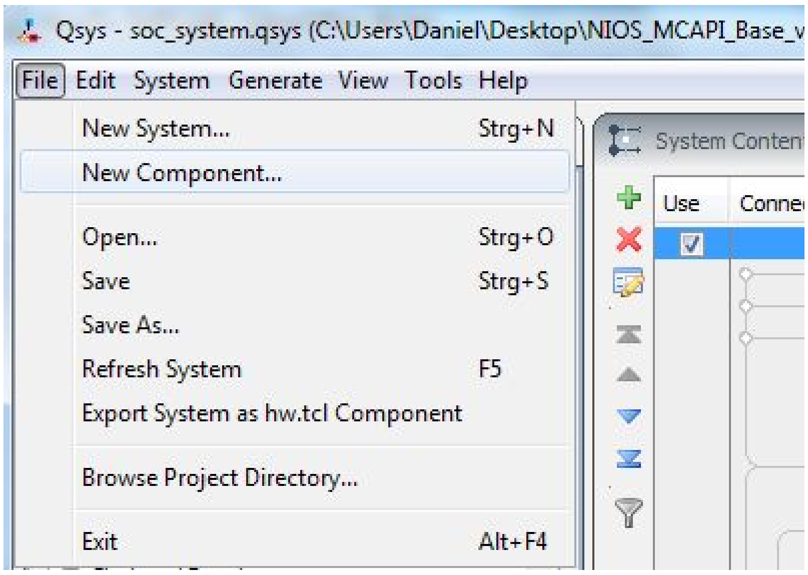


pwm.vhd	18.12.2014 09:51	VHD-Datei	3 KB
pwm_hw.tcl	10.12.2014 13:53	TCL-Datei	4 KB

Graphic 16: IP-Core files

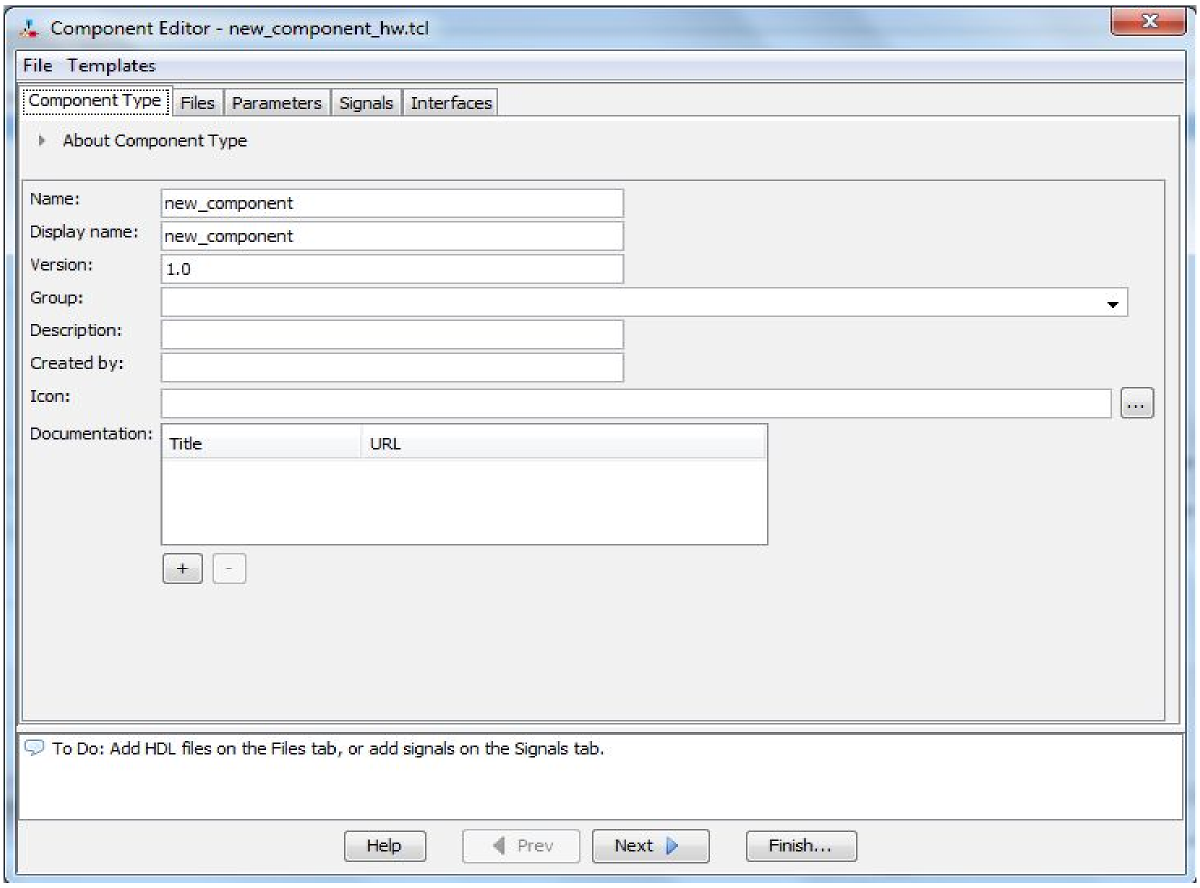
The .vhd file is the VHDL description of the IP-Core. The .tcl file contains information and settings on the component that can be loaded into QSYS.

To add a new component just click on File->New Component....


Qsys - soc_system.qsys (C:\Users\Daniel\Desktop\NIOS_MCAPI_Base_v
File Edit System Generate View Tools Help
New System... Strg+N
New Component...
Open... Strg+O
Save Strg+S
Save As...
Refresh System F5
Export System as hw.tcl Component
Browse Project Directory...
Exit Alt+F4
System Conten
Use
Conne

Graphic 17: QSYS file menu

The click opens the following dialogue window.



Name:	new_component
Display name:	new_component
Version:	1.0
Group:	
Description:	
Created by:	
Icon:	

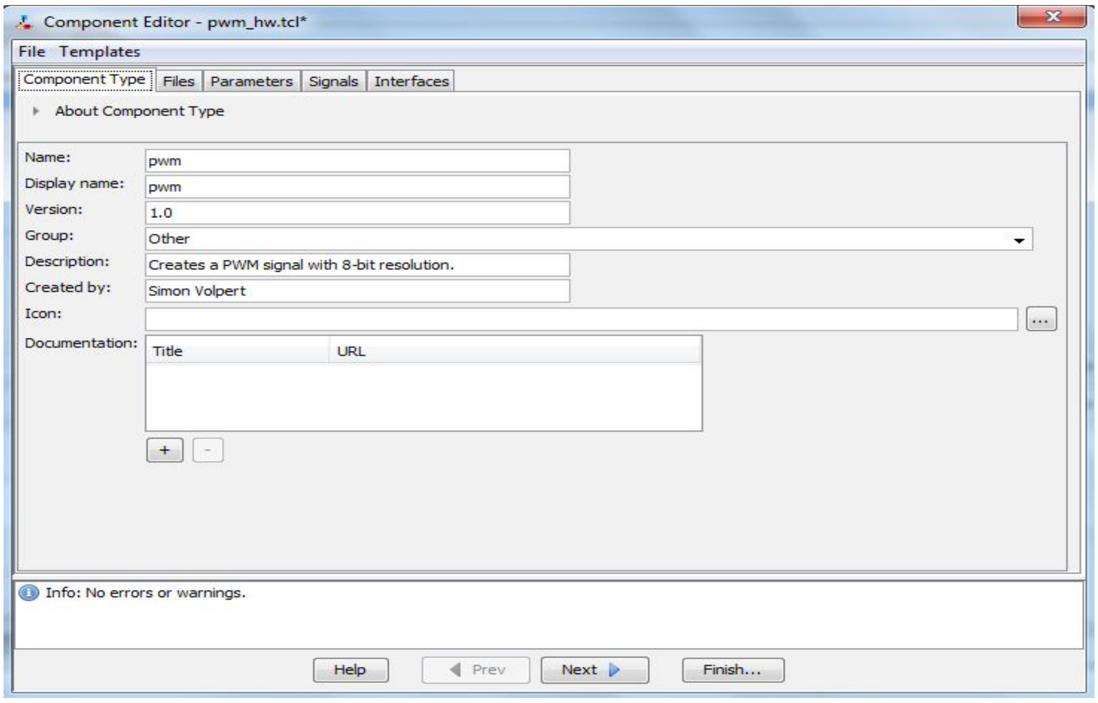
Documentation:

Title	URL

To Do: Add HDL files on the Files tab, or add signals on the Signals tab.

Graphic 18: QSYS component editor

In the different register cards multiple settings have to be defined. In our case we already have a .tcl file that contains this information and can be loaded into the Component Editor.
To open the .tcl file just click on File->open in the menu bar. In the opening dialogue just navigate to the .tcl file into your project directory and click the open button.
After that the settings should be loaded into the Component Editor.

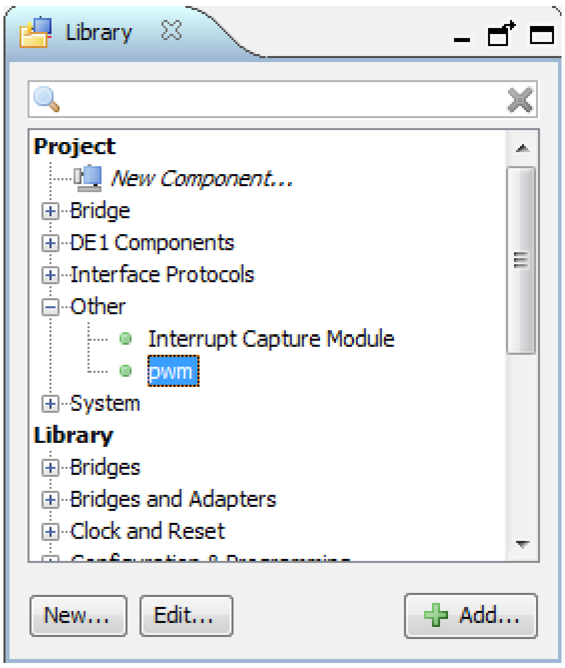


Name:	pwm
Display name:	pwm
Version:	1.0
Group:	Other
Description:	Creates a PWM signal with 8-bit resolution.
Created by:	Simon Volpert
Icon:	
Documentation:	Title URL

Info: No errors or warnings.

Graphic 19: QSYS component editor

If you don't have to adjust any settings as in our example you just have to click Finish... to add the compo-

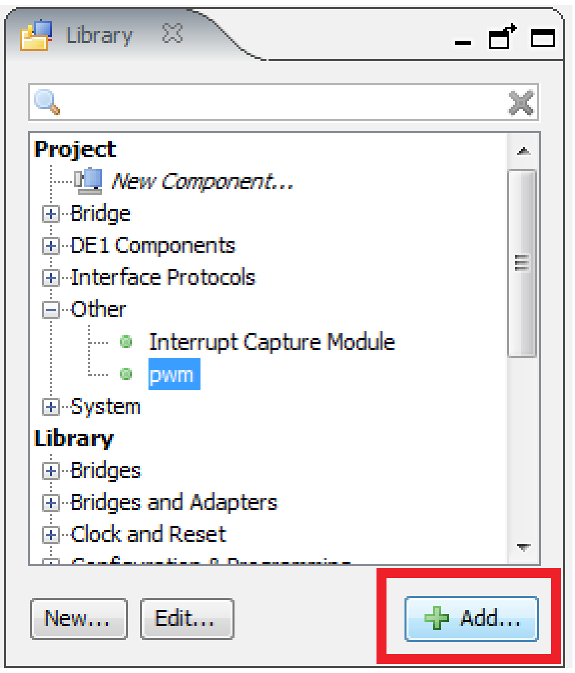


Graphic 20: QSYS component library

nent to the library. You should see the IP-Core into the group you have selected under the Display name you entered. In our case the group "Other" and the display name "pwm".

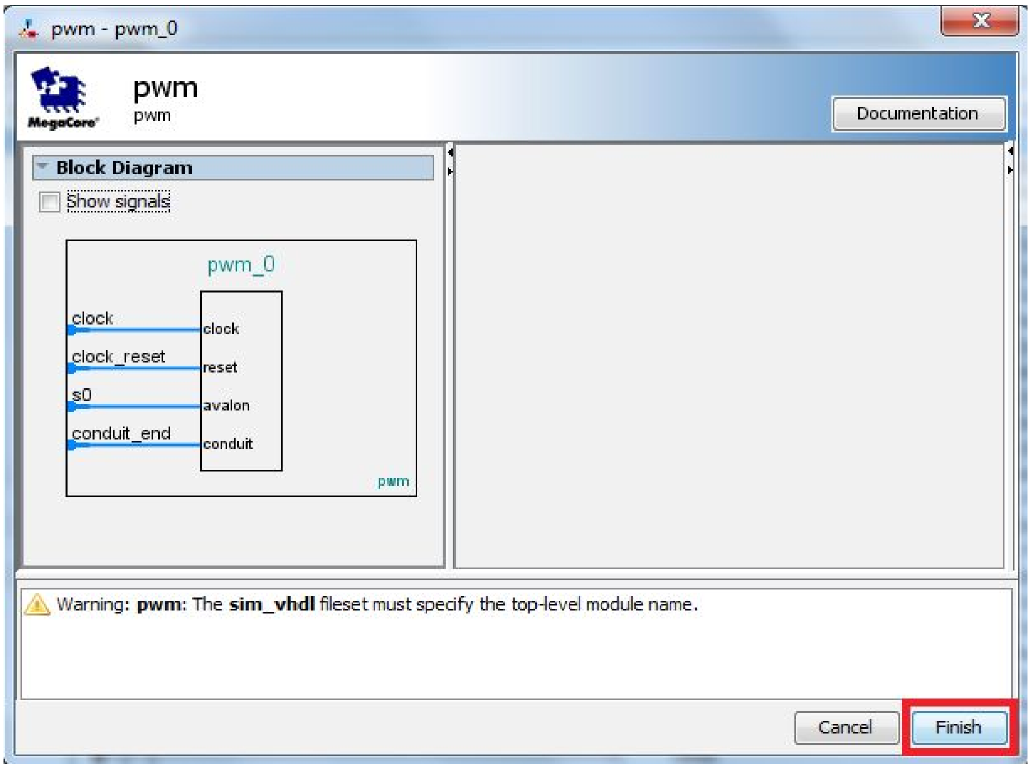
7.4 Add Component to the System

Also the example to add a component to the system uses the imported pwm component as an example component.
As the first step you have to search you component in the library. In this case it can be found in the group "Other" with the display name "pwm". To add the component just mark it by clicking on it and click Add....



Graphic 21: QSYS component library

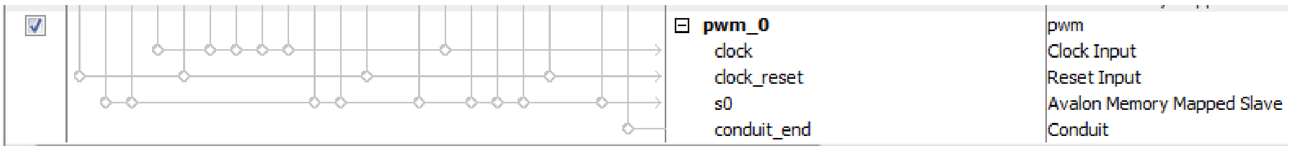
The dialogue that is shown below should appear, which often offers additional settings for the component. In our case we just click Finish... because there are no settings that have to be adjusted to add the component.



Warning: **pwm**: The **sim_vhdl** fileset must specify the top-level module name.

Graphic 22: PWM controller settings

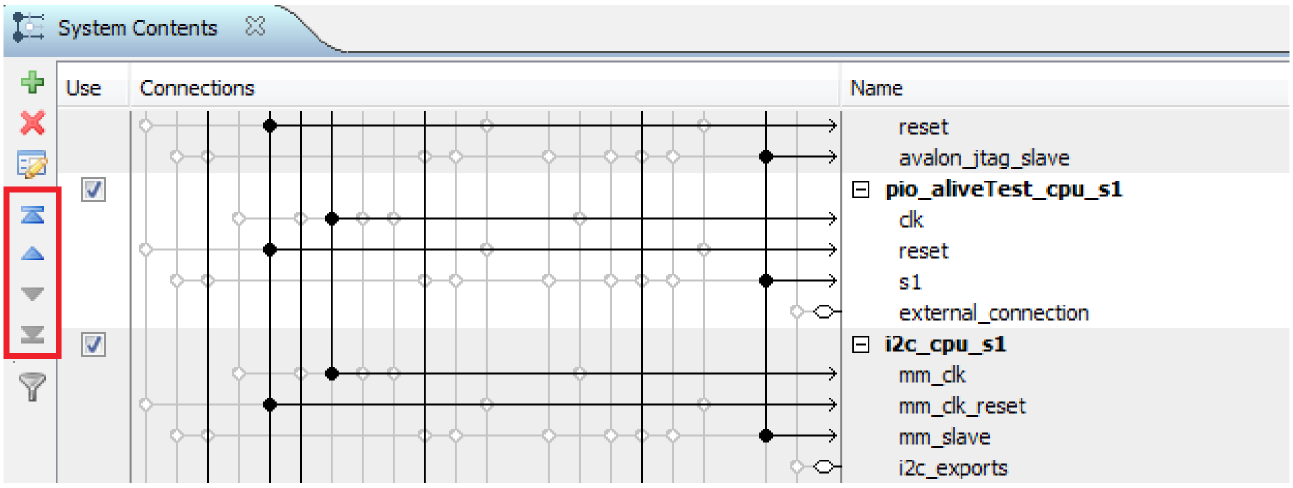
The component now should appear in the component overview.



pwm_0	pwm
clock	Clock Input
clock_reset	Reset Input
s0	Avalon Memory Mapped Slave
conduit_end	Conduit

Graphic 23: QSYS component overview

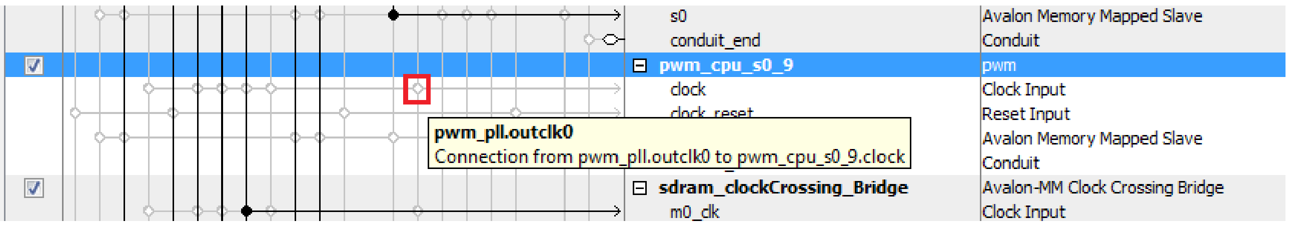
Maybe the component which appeared at the bottom of the component overview has to be moved up in



Graphic 24: Move components in QSYS

order to keep a better overview on you components.
To move the component in the component view you just have to click on the following buttons.
To rename the component right-click on the component and click rename. After that you can enter a new display name. In our case we name the component as pwm_cpu_s0_9.

After that the component has to be wired. In our case a clock signal, a reset signal and a data signal.
To activate a connection between two components just click on the nodes that are displayed left the display names.


s0 Avalon Memory Mapped Slave
conduit_end Conduit
pwm_cpu_s0_9 pwm
clock Clock Input
clock_reset Reset Input
Avalon Memory Mapped Slave
Conduit
sdram_clockCrossing_Bridge Avalon-MM Clock Crossing Bridge
m0_clk Clock Input

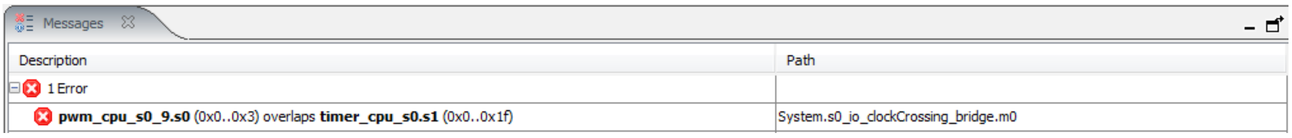
pwm_pll.outclk0
Connection from pwm_pll.outclk0 to pwm_cpu_s0_9.clock

Graphic 25: Wire component

If you add new IO components in our system you should make the following connections:

- Clock input -> system_pll.outclk1 or for the pwms pwm_pll.outclk0
- Reset input -> clk_0.clk_reset
- Data signal -> s0_io_clockCrossing_bridge.m0

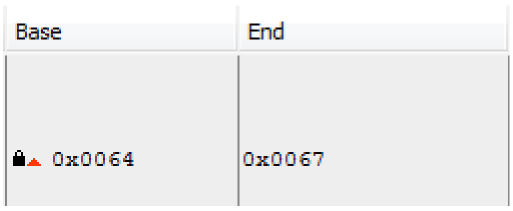
After the all signals are wired there is maybe a warning message that the address spaces of some components do overlap.


Messages

Description	Path
1 Error	
pwm_cpu_s0_9.s0 (0x0..0x3) overlaps **timer_cpu_s0.s1** (0x0..0x1f)	System.s0_io_clockCrossing_bridge.m0

Graphic 26: Addressspace error

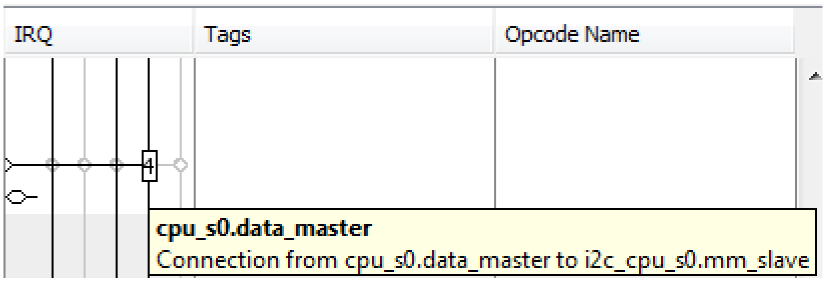
To solve this problem you should click the following menu item of the menu bar: System -> Assign Base Addresses. The error message should disappear after that. If you want to keep the addresses of other components stable you should look these by clicking on the small lock icons beside each components address which can be found in the "Base" column.



Base	End
0x0064	0x0067

Graphic 27: Lock base address

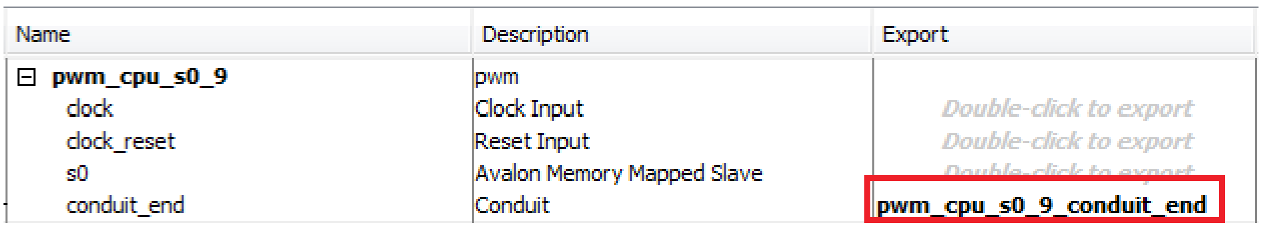
Optionally you also could connect an interrupt signal to a processor and assign interrupt priorities using the "IRQ" column. This is not necessary in this case. The IRQ column is shown in the picture below.



cpu_s0.data_master
Connection from cpu_s0.data_master to i2c_cpu_s0.mm_slave

Graphic 28: Define interrupt connections

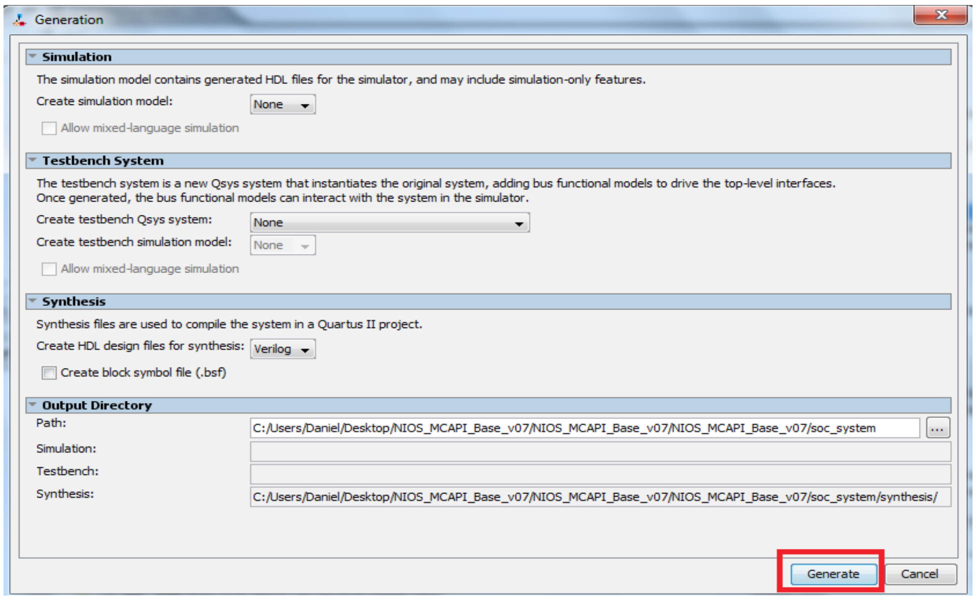
Another important thing you have to do is to export signals that shall be wired on IOs of the FPGA for example if the PWM controller output signal should be available on a IO header of your board.
To achieve this you just have to Double-click on the signal that has to be exported into the "Export" column.



Name	Description	Export
⊟ **pwm_cpu_s0_9**	pwm	
clock	Clock Input	Double-click to export
clock_reset	Reset Input	Double-click to export
s0	Avalon Memory Mapped Slave	Double-click to export
conduit_end	Conduit	**pwm_cpu_s0_9_conduit_end**

Graphic 29: Export output signal

After you finished these steps you should save you result by clicking on File -> Save in the menu bar.
Now you system is ready for generation. To generate the system just click on
Generate -> Generate... in the menu bar. The Generation dialogue should open and offer the output settings which should be already appropriate in our case. So you jsut have to click on the "Generate" button.


Generation

Simulation

The simulation model contains generated HDL files for the simulator, and may include simulation-only features.

Create simulation model: None

Allow mixed-language simulation

Testbench System

The testbench system is a new Qsys system that instantiates the original system, adding bus functional models to drive the top-level interfaces. Once generated, the bus functional models can interact with the system in the simulator.

Create testbench Qsys system: None

Create testbench simulation model: None

Allow mixed-language simulation

Synthesis

Synthesis files are used to compile the system in a Quartus II project.

Create HDL design files for synthesis: Verilog

Create block symbol file (.bsf)

Output Directory

Path: C:/Users/Daniel/Desktop/NIOS_MCAPI_Base_v07/NIOS_MCAPI_Base_v07/NIOS_MCAPI_Base_v07/soc_system

Simulation:

Testbench:

Synthesis: C:/Users/Daniel/Desktop/NIOS_MCAPI_Base_v07/NIOS_MCAPI_Base_v07/NIOS_MCAPI_Base_v07/soc_system/synthesis/

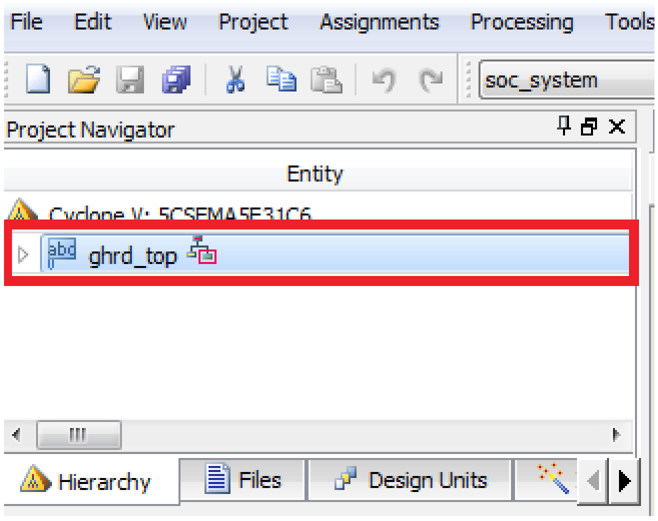
Graphic 30: Generate system

A progress dialogue should appear that shows information, warnings and errors during the generation process. After the dialogue shows that the Generation has finished just close it and switch back to Quartus II.

7.5 Wire a Signal to a FPGA Pin

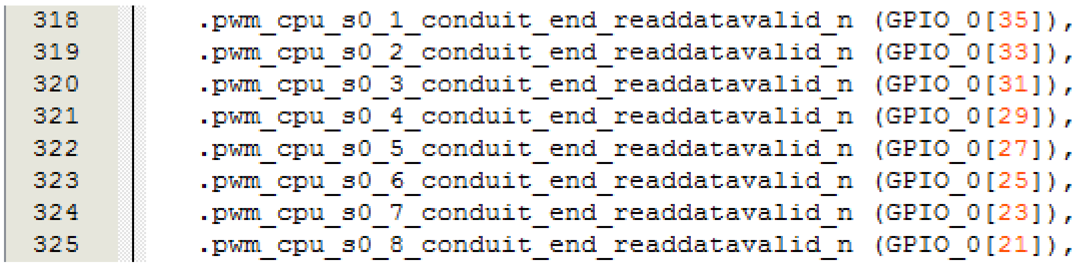
In this chapter it is explained how to wire a signal to a FPGA. In this example we want to wire the PWM controllers output to a pin on the expansion headers of our board.

First double-click on the top level entity of you project in Quartus II.



Graphic 31: Open top-level entity

The entity opens and shows the connections of the already wired exported signals.



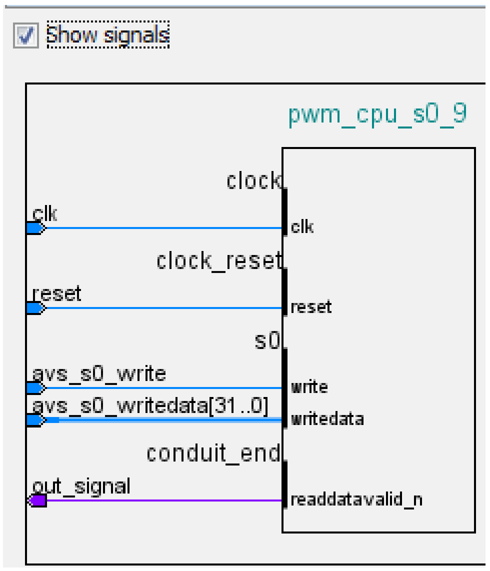
```
    .pwm_cpu_s0_1_conduit_end_readdatavalid_n (GPIO_0[35]),
    .pwm_cpu_s0_2_conduit_end_readdatavalid_n (GPIO_0[33]),
    .pwm_cpu_s0_3_conduit_end_readdatavalid_n (GPIO_0[31]),
    .pwm_cpu_s0_4_conduit_end_readdatavalid_n (GPIO_0[29]),
    .pwm_cpu_s0_5_conduit_end_readdatavalid_n (GPIO_0[27]),
    .pwm_cpu_s0_6_conduit_end_readdatavalid_n (GPIO_0[25]),
    .pwm_cpu_s0_7_conduit_end_readdatavalid_n (GPIO_0[23]),
    .pwm_cpu_s0_8_conduit_end_readdatavalid_n (GPIO_0[21]),
```

Graphic 32: Exported signals

The basic format of one line is .<export_signal_name> (node_name).
For a list of the available node names see the pin planner in Quartus II. In our case all GPIO pins are already imported into the project.

To find out what is the signal name of you export signal you have multiple possibilities. You can just go into QSYS, right-click on a component and click Edit....

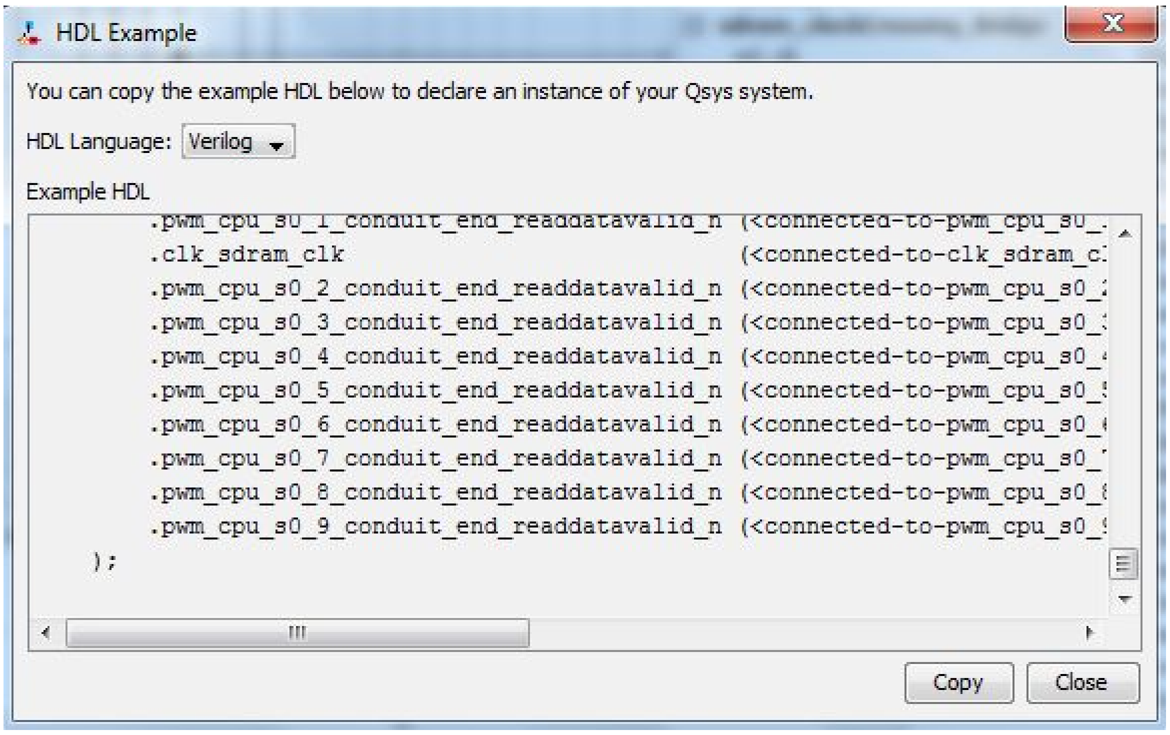
To show all signals set the check-box Show signals. In our example the resulting view would be:



Graphic 33: Signal details

The name of the output signal in this case is pwm_cpu_s0_9_conduit_end_readdatavalid_n.

Alternatively stay in QSYS and click on Generate -> HDL Example... in the menu bar.



You can copy the example HDL below to declare an instance of your Qsys system.

HDL Language: Verilog

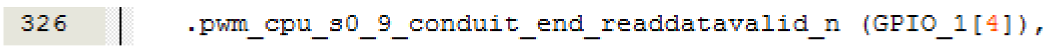
Example HDL

```
        .pwm_cpu_s0_1_conduit_end_readdatavalid_n (<connected-to-pwm_cpu_s0_
        .clk_sdram_clk                            (<connected-to-clk_sdram_c
        .pwm_cpu_s0_2_conduit_end_readdatavalid_n (<connected-to-pwm_cpu_s0_
        .pwm_cpu_s0_3_conduit_end_readdatavalid_n (<connected-to-pwm_cpu_s0_
        .pwm_cpu_s0_4_conduit_end_readdatavalid_n (<connected-to-pwm_cpu_s0_
        .pwm_cpu_s0_5_conduit_end_readdatavalid_n (<connected-to-pwm_cpu_s0_
        .pwm_cpu_s0_6_conduit_end_readdatavalid_n (<connected-to-pwm_cpu_s0_
        .pwm_cpu_s0_7_conduit_end_readdatavalid_n (<connected-to-pwm_cpu_s0_
        .pwm_cpu_s0_8_conduit_end_readdatavalid_n (<connected-to-pwm_cpu_s0_
        .pwm_cpu_s0_9_conduit_end_readdatavalid_n (<connected-to-pwm_cpu_s0_
    );
```

Graphic 34: QSYS HDL example

QSYS generates and example HDL file in which the export signal names that should be used are contained.

Now remember or copy the signal name and go back to Quartus II. Enter the following code into your top level entity.



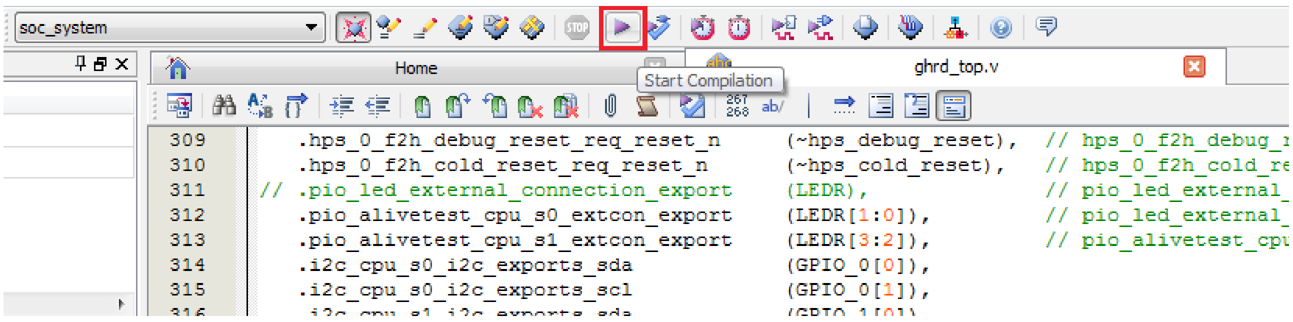
```
326 |    .pwm_cpu_s0_9_conduit_end_readdatavalid_n (GPIO_1[4]),
```

Graphic 35: Wire new output signal

With this code the output of our PWM controller is wired to pin 4 on the second expansion header of our board.

7.6 Generate the System

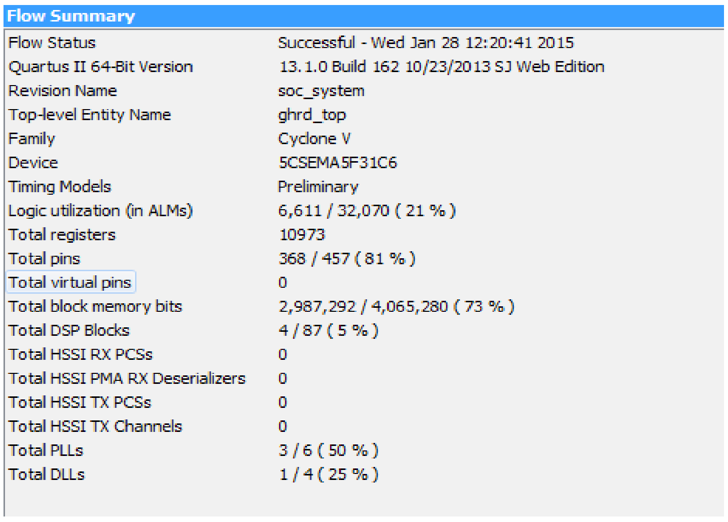
To generate the output files that are required to program the board just click on the "Start Compilation" button.



```
    .hps_0_f2h_debug_reset_req_reset_n      (~hps_debug_reset),  // hps_0_f2h_debug_r
    .hps_0_f2h_cold_reset_req_reset_n       (~hps_cold_reset),   // hps_0_f2h_cold_re
//  .pio_led_external_connection_export     (LEDR),              // pio_led_external_
    .pio_alivetest_cpu_s0_extcon_export     (LEDR[1:0]),         // pio_led_external_
    .pio_alivetest_cpu_s1_extcon_export     (LEDR[3:2]),         // pio_alivetest_cpu
    .i2c_cpu_s0_i2c_exports_sda             (GPIO_0[0]),
    .i2c_cpu_s0_i2c_exports_scl             (GPIO_0[1]),
```

Graphic 36: Start compilation

The "Compilation Report" view should open that shows the progress of the compilation. After the gener-



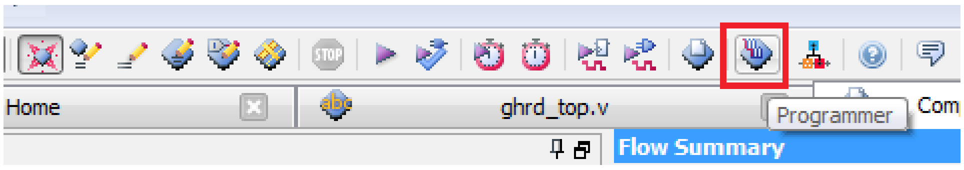
Flow Summary	
Flow Status	Successful - Wed Jan 28 12:20:41 2015
Quartus II 64-Bit Version	13.1.0 Build 162 10/23/2013 SJ Web Edition
Revision Name	soc_system
Top-level Entity Name	ghrd_top
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Preliminary
Logic utilization (in ALMs)	6,611 / 32,070 (21 %)
Total registers	10973
Total pins	368 / 457 (81 %)
Total virtual pins	0
Total block memory bits	2,987,292 / 4,065,280 (73 %)
Total DSP Blocks	4 / 87 (5 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI TX Channels	0
Total PLLs	3 / 6 (50 %)
Total DLLs	1 / 4 (25 %)

Graphic 37: Compilation report

ation finished it just shows a summary of the results of the compilation.

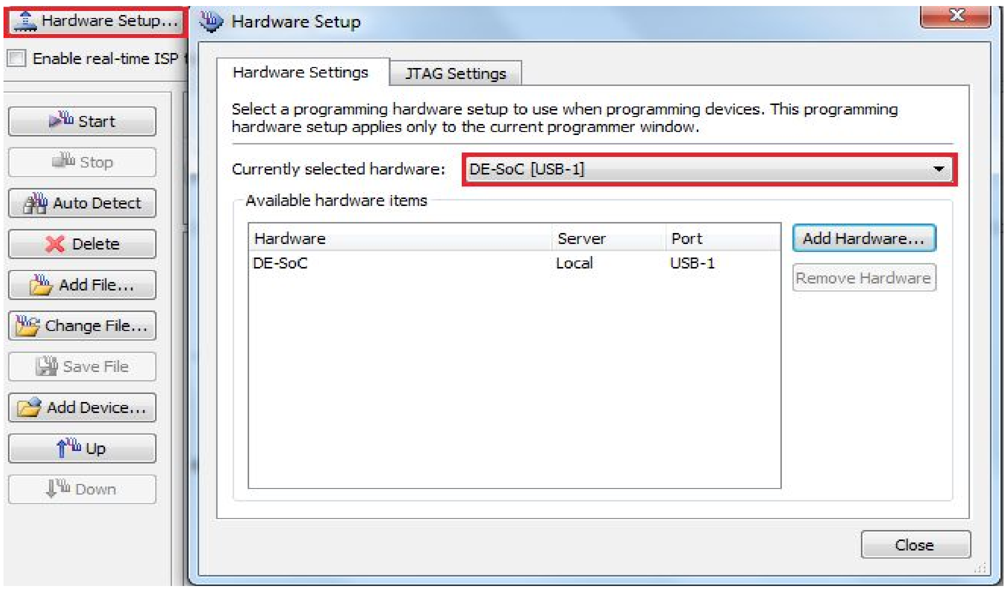
7.7 Program the Board

To program the board you should first open the Quartus II Programmer by clicking on the following icon.



Graphic 38: Open programmer

The Programmer should open after this and at this point you should connect and power up your board. In the Programmer you should first configure the connection hardware by clicking on Hardware Setup.



Select a programming hardware setup to use when programming devices. This programming hardware setup applies only to the current programmer window.

Currently selected hardware: DE-SoC [USB-1]

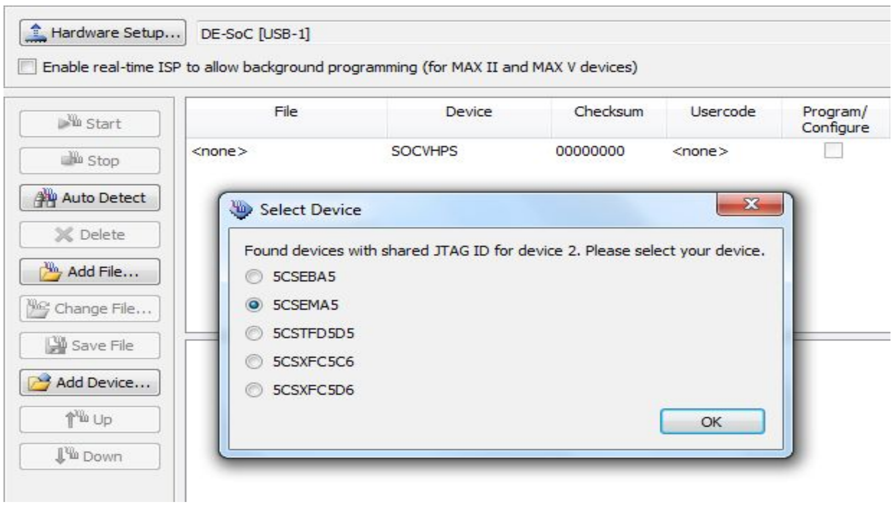
Available hardware items

Hardware	Server	Port
DE-SoC	Local	USB-1

Graphic 39: Setup connection

In the "Hardware Setup" dialogue you just have to select the hardware that should be used for the connection. In this case this is "DE-SoC". To finish the configuration just click on the "Close" button.

Afterwords you should should click on "Auto Detect"to search for connected devices. The following dialogue should appear.



File	Device	Checksum	Usercode	Program/ Configure
<none>	SOCVHPS	00000000	<none>	☐

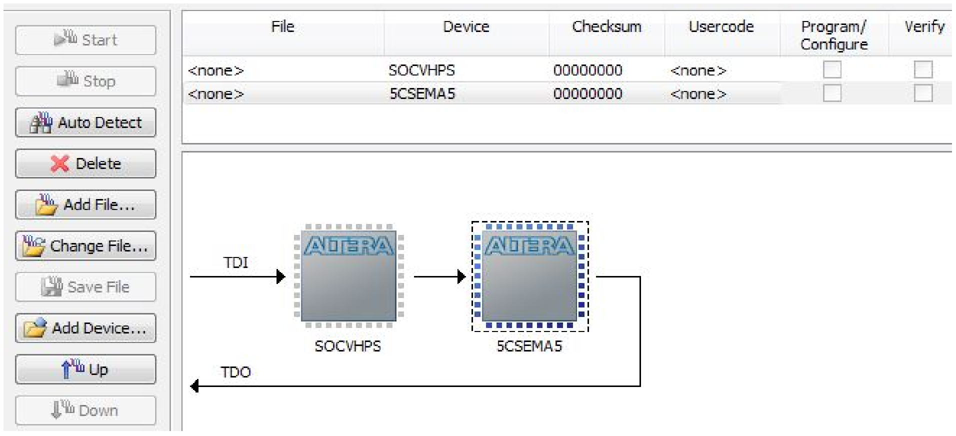
Found devices with shared JTAG ID for device 2. Please select your device.

- ○ 5CSEBA5
- ● 5CSEMA5
- ○ 5CSTFD5D5
- ○ 5CSXFC5C6
- ○ 5CSXFC5D6

Graphic 40: Auto detect hardware

You have to select the type of you FPGA and click "OK". In our case the right type is 5CSEMA5.

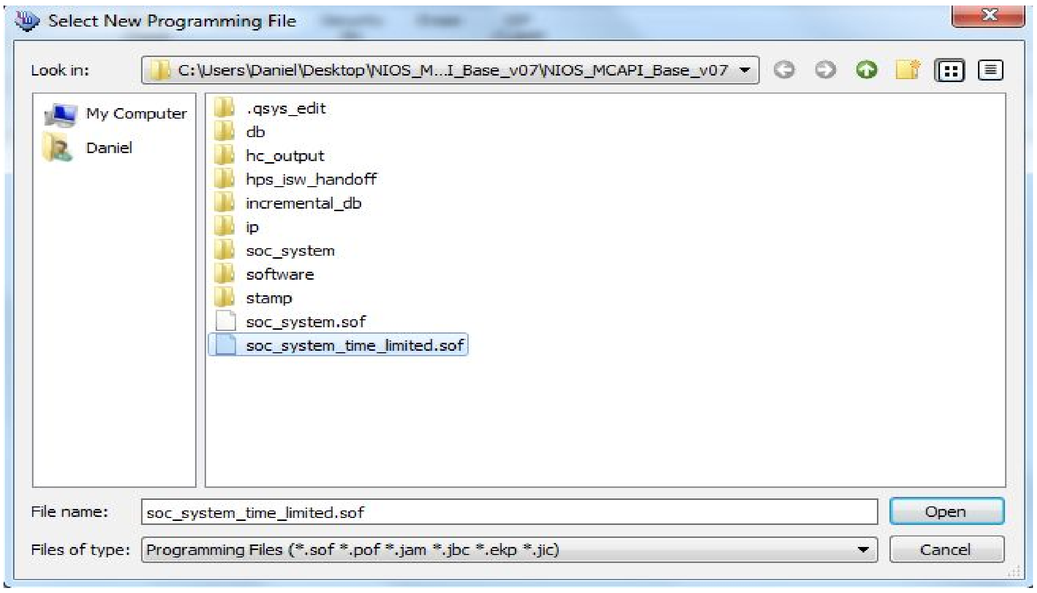
Your Programmer dialogue should now look like this:



File	Device	Checksum	Usercode	Program/ Configure	Verify
<none>	SOCVHPS	00000000	<none>	☐	☐
<none>	5CSEMA5	00000000	<none>	☐	☐

Graphic 41: Select FPGA

To select an output file of Quartus II for programming your device click on the FPGA device in this case 5CSEMA5 to select it for configuration. Then click on Change File... to open the file selection dialogue.


Select New Programming File

Look in: C:\Users\Daniel\Desktop\NIOS_M...I_Base_v07\NIOS_MCAPI_Base_v07

My Computer
Daniel

.qsys_edit
db
hc_output
hps_isw_handoff
incremental_db
ip
soc_system
software
stamp
soc_system.sof
soc_system_time_limited.sof

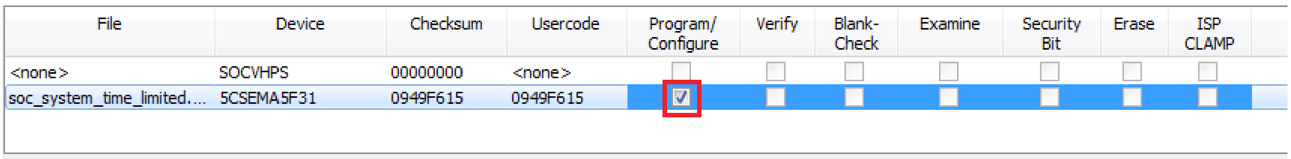
File name: soc_system_time_limited.sof
Files of type: Programming Files (*.sof *.pof *.jam *.jbc *.ekp *.jic)

Open
Cancel

Graphic 42: Select output file

You should select the fitting .sof file that contains your generated system. In our case this is the soc_system_time_limited.sof. If your system doesn't contain any IP-Cores that are limited to the subscription version of Quartus II or you use this version of Quartus II you may also can select the non time limited version. Just click open to finish the file selection.

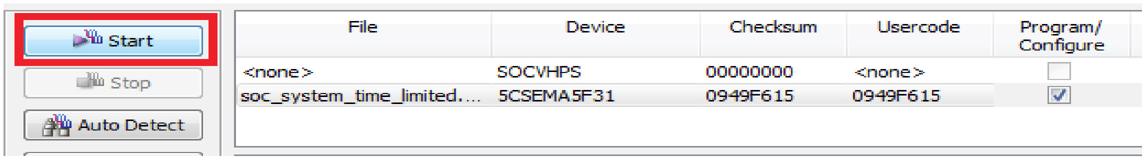
Before programming the device you now have to set the check-box in the column "Program/Configure".



File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine	Security Bit	Erase	ISP CLAMP
<none>	SOCVHPS	00000000	<none>	☐	☐	☐	☐	☐	☐	☐
soc_system_time_limited....	5CSEMA5F31	0949F615	0949F615	✓	☐	☐	☐	☐	☐	☐

Graphic 43: Choose program / configure

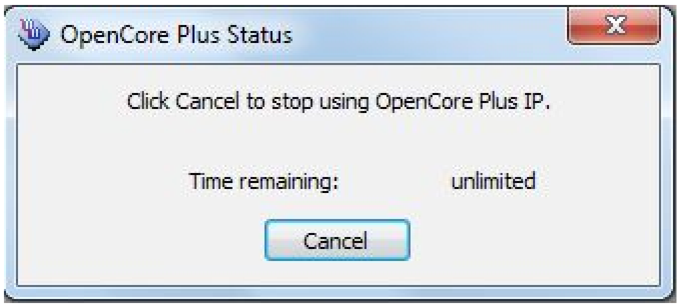
After that just click on the button "Start". After that the programming should start. Its progress can be seen in the progress bar.



File	Device	Checksum	Usercode	Program/ Configure
<none>	SOCVHPS	00000000	<none>	☐
soc_system_time_limited....	5CSEMA5F31	0949F615	0949F615	✓

Graphic 44: Start programming

After the programming, in case that you used the time limited sof, a dialogue opens that has to be kept open in order to keep the system in the FPGA. Also the connection to the development system has to be kept. Otherwise you could also terminate the connection.


OpenCore Plus Status

Click Cancel to stop using OpenCore Plus IP.

Time remaining: unlimited

Graphic 45: Time limited programming dialogue

Now the system is ready for being programmed from NIOS-II EDS.

8 Software Development

8.1 Introduction

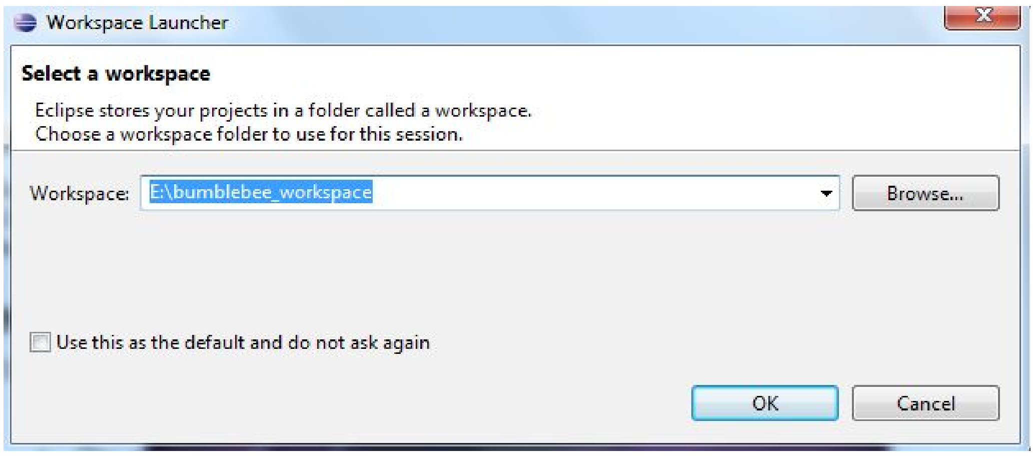
In this Howto the development of software for the project is described. It is split in two parts. One part that explains how to develop software for the Nios II processors and one that explains how to develop software for the ARM processor used in this project.

8.2 NIOS-II Software Development

To follow this Howto you need the following things:

- Nios II EDS (IDE provided by Altera)
- The current SoPC version
- The C source code provided in the appendix of the project report

Before starting Nios II EDS you should unzip the archive that contains the current SoPC version to your hard drive. In our case this is the NIOS_MCAPI_Base_v07.zip.
After that just start up Nios II EDS from your start menu. You should see the Eclipse splash screen and should be asked to select a workspace. Here you can choose an existing workspace or just create a new directory.


Workspace Launcher

Select a workspace

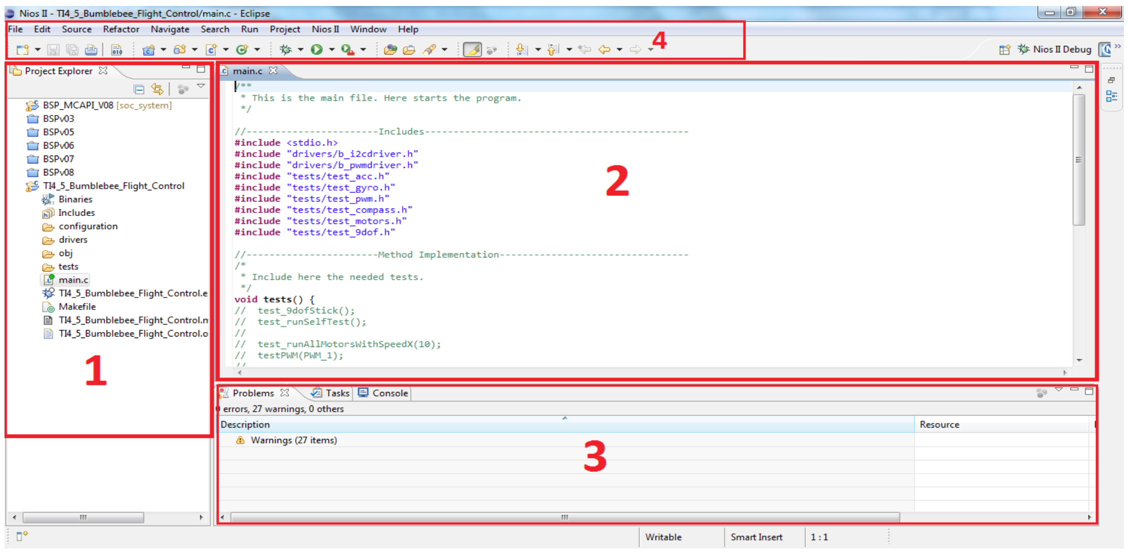
Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: E:\bumblebee_workspace

Graphic 46: Select workspace

After you found an appropriate path just click the "OK" button.

Into the Nios II EDS IDE there are 4 important areas that are needed to execute most actions that are required to develop new software modules.


Nios II - TI4_5_Bumblebee_Flight_Control/main.c - Eclipse
File Edit Source Refactor Navigate Search Run Project Nios II Window Help
Nios II Debug
Project Explorer
BSP_MCAPI_V08 [soc_system]
BSPv03
BSPv05
BSPv06
BSPv07
BSPv08
TI4_5_Bumblebee_Flight_Control
Binaries
Includes
configuration
drivers
obj
tests
main.c
TI4_5_Bumblebee_Flight_Control.e
Makefile
TI4_5_Bumblebee_Flight_Control.n
TI4_5_Bumblebee_Flight_Control.o
main.c
1
2
3
4
Problems
Tasks
Console
Description
Resource
Writable
Smart Insert
1 : 1

```
/**
 * This is the main file. Here starts the program.
 */

//----------------------Includes-------------------------------------------
#include <stdio.h>
#include "drivers/b_i2cdriver.h"
#include "drivers/b_pwmdriver.h"
#include "tests/test_acc.h"
#include "tests/test_gyro.h"
#include "tests/test_pwm.h"
#include "tests/test_compass.h"
#include "tests/test_motors.h"
#include "tests/test_9dof.h"

//----------------------Method Implementation-------------------------------
/*
 * Include here the needed tests.
 */
void tests() {
//  test_9dofStick();
//  test_runSelfTest();
//
//  test_runAllMotorsWithSpeedX(10);
//  testPWM(PWM_1);
```

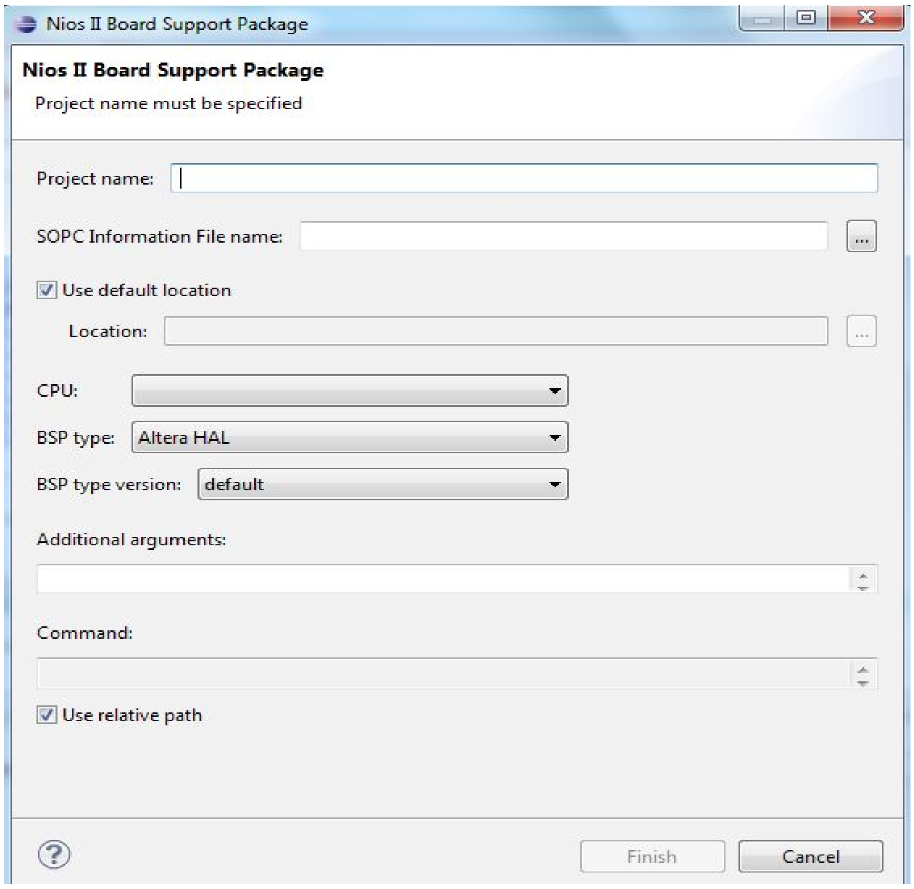
```
0 errors, 27 warnings, 0 others
Warnings (27 items)
```

Graphic 47: Nios II EDS overview

- Area 1 is the project explorer which offers an overview over the available projects as well the directory and file structure of a project.
- Area 2 is the code editor in which the code of the header and source files can be edited.
- Area 3 displays different information in various registers. Examples are the warnings and errors or the console output of the Nios II application.
- Area 4 contains shortcuts to often used functions as well as the menu bar that offers all functions available in the IDE.

8.3 Create a BSP

As a first step you should create a Board Support Package for your SoPC. To do this just click File -> New -> Nios II Board Support Package in the menu bar. After this the following dialogue should open:


Nios II Board Support Package

Nios II Board Support Package

Project name must be specified

Project name:

SOPC Information File name:

[✓] Use default location

Location:

CPU:

BSP type: Altera HAL

BSP type version: default

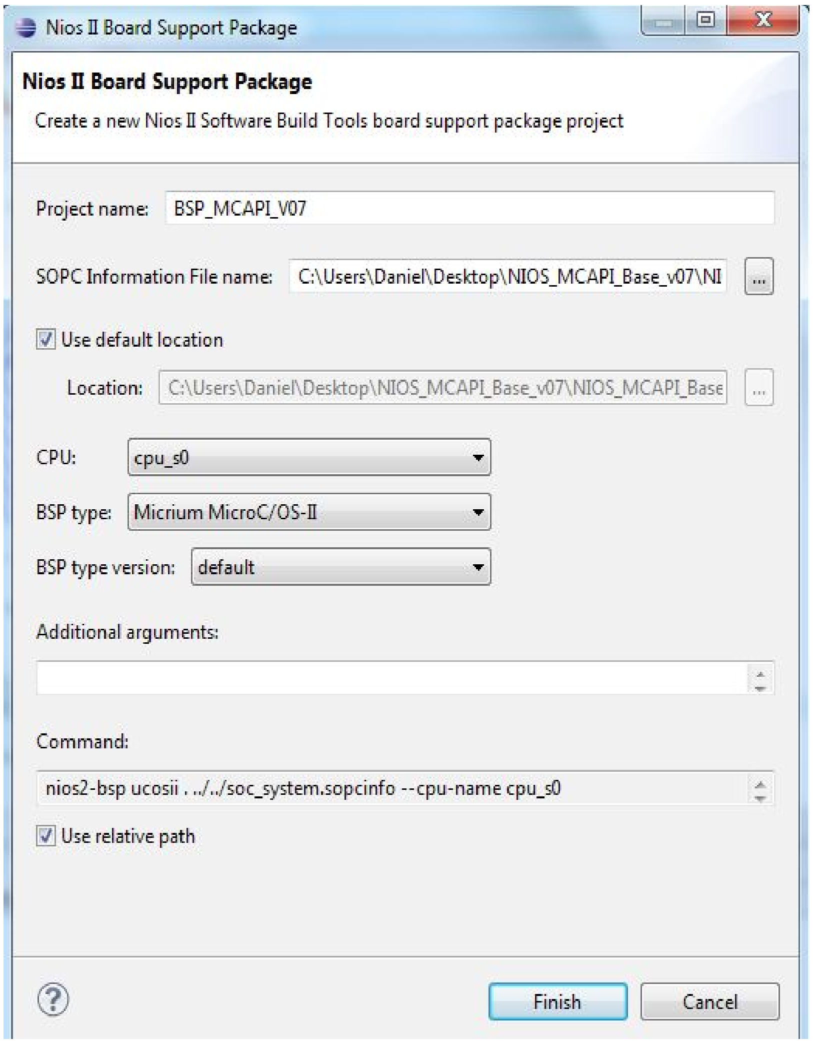
Additional arguments:

Command:

[✓] Use relative path

Graphic 48: New board support package

Here you should first enter a project name, in our case BSP_MCAPI_V07.
After that just select the .sopcinfo file that should be contained in your SoPC Quartus II project directory.
After the information file was loaded optionally select the CPU for which the BSP should be used if your system contains more than one CPU. You should also select Micrium MicroC/OS-II as BSP type to use MicroC/OS-II as a real-time operating system for your project.
Optionally you could also change the path that is used to store the project. Otherwise the BSP will be stored in a sub directory of your Quartus II project.
After configuration the dialogue should look like this.


Nios II Board Support Package

Nios II Board Support Package

Create a new Nios II Software Build Tools board support package project

Project name: BSP_MCAPI_V07

SOPC Information File name: C:\Users\Daniel\Desktop\NIOS_MCAPI_Base_v07\NI

☑ Use default location

Location: C:\Users\Daniel\Desktop\NIOS_MCAPI_Base_v07\NIOS_MCAPI_Base

CPU: cpu_s0

BSP type: Micrium MicroC/OS-II

BSP type version: default

Additional arguments:

Command:

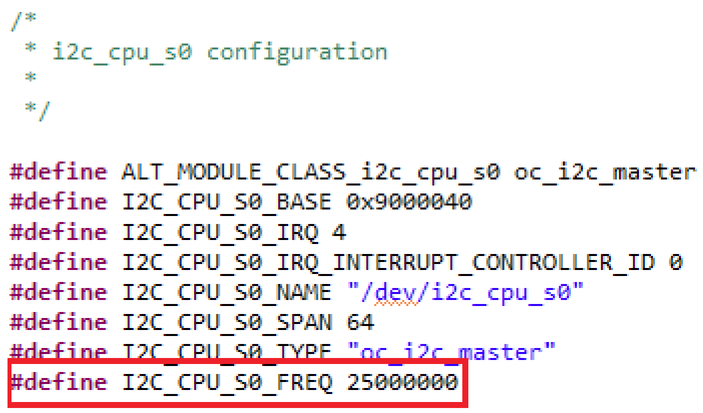
```
nios2-bsp ucosii . ../../soc_system.sopcinfo --cpu-name cpu_s0
```

☑ Use relative path

Graphic 49: New board support package

To generate the BSP you now should just click the "Finish" button.

Important:
With the current version of the SoPC and foregone versions there is an error in the BSP generation process. If the system has an I2C controller of the type oc_i2c_master as a component not every define regarding to the controller is made in the system.h automatically.
To fix this error open the BSP into the project explorer and open the file "system.h".
Search for every occurring block that describes the I2C controller and add the following.



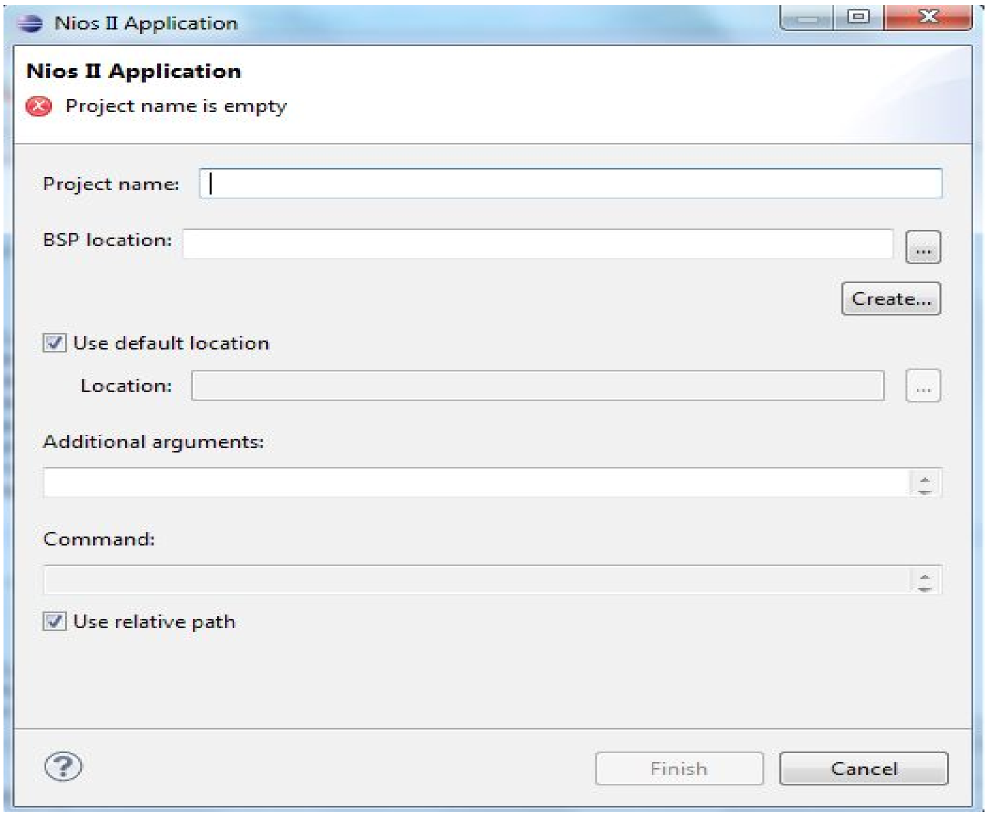
```
/*
 * i2c_cpu_s0 configuration
 *
 */

#define ALT_MODULE_CLASS_i2c_cpu_s0 oc_i2c_master
#define I2C_CPU_S0_BASE 0x9000040
#define I2C_CPU_S0_IRQ 4
#define I2C_CPU_S0_IRQ_INTERRUPT_CONTROLLER_ID 0
#define I2C_CPU_S0_NAME "/dev/i2c_cpu_s0"
#define I2C_CPU_S0_SPAN 64
#define I2C_CPU_S0_TYPE "oc_i2c_master"
#define I2C_CPU_S0_FREQ 25000000
```

Graphic 50: Add additional define

The FREQ parameter has to be set to 25000000 because the IO components are connected to a clock signal with 25MHz.

8.4 Create NIOS-II Application

After you created an appropriate Board Support Package you can create a Nios II Application. Therefore click File -> New -> Nios II Application. The following dialogue should appear.



Nios II Application

Nios II Application

Project name is empty

Project name:

BSP location:

Create...

[✓] Use default location

Location:

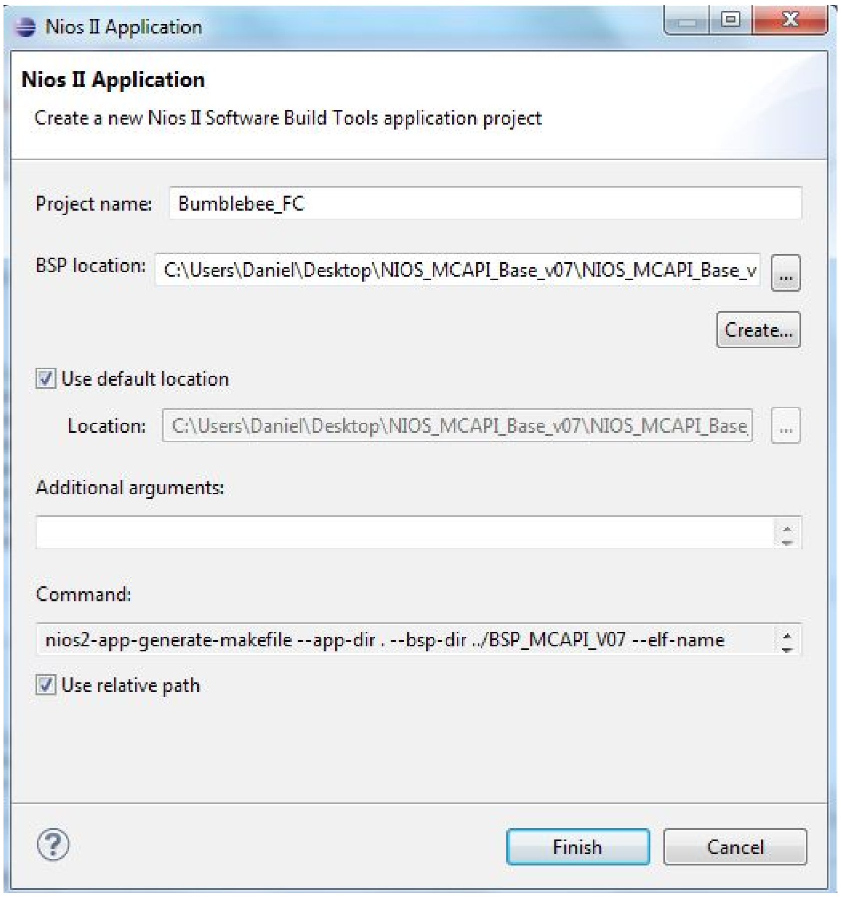
Additional arguments:

Command:

[✓] Use relative path

Graphic 51: New Nios II application

You have to enter a fitting project name here. Also you have to select the location of the BSP you created before.
After you entered your setting the dialogue should approximately look like this.


Nios II Application

Nios II Application

Create a new Nios II Software Build Tools application project

Project name: Bumblebee_FC

BSP location: C:\Users\Daniel\Desktop\NIOS_MCAPI_Base_v07\NIOS_MCAPI_Base_v

Create...

[✓] Use default location

Location: C:\Users\Daniel\Desktop\NIOS_MCAPI_Base_v07\NIOS_MCAPI_Base

Additional arguments:

Command:

```
nios2-app-generate-makefile --app-dir . --bsp-dir ../BSP_MCAPI_V07 --elf-name
```

[✓] Use relative path

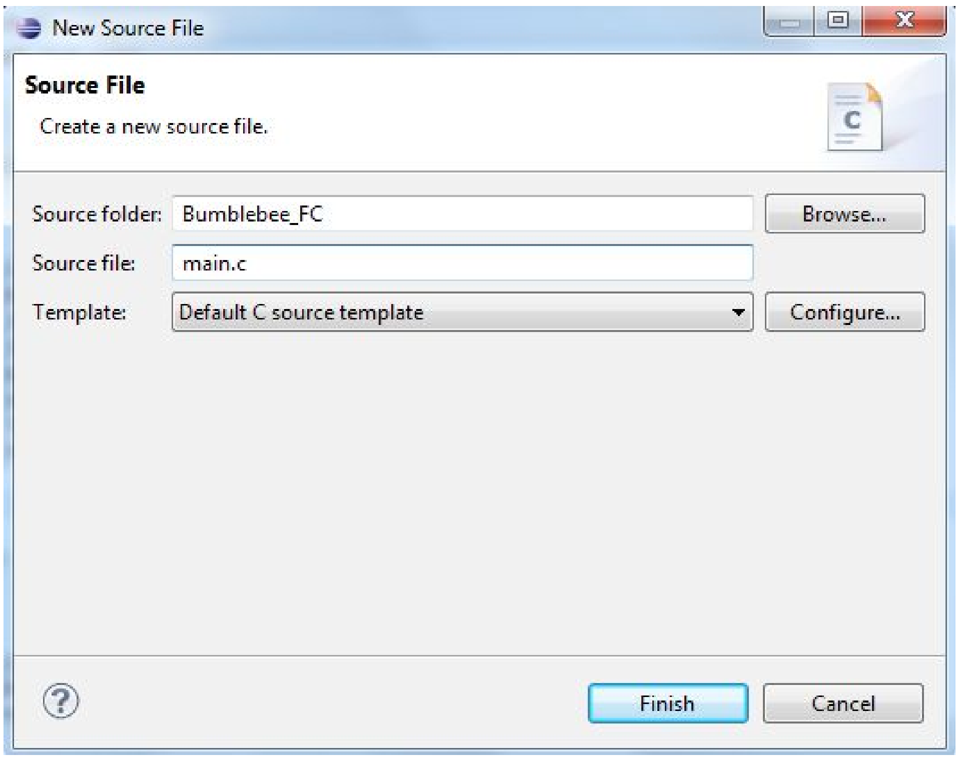
Finish Cancel

Graphic 52: New Nios II application

Optionally you could define a different project location that is located in your workspace.
To create the project just click "Finish". The new project should appear in the Project Explorer.

8.5 Create Source Code

If you want to start from scratch you now can just create your source files into your project. To create a simple program just right-click on your project and select New -> Source File. The following dialogue will open:



New Source File

Source File

Create a new source file.

Source folder: Bumblebee_FC

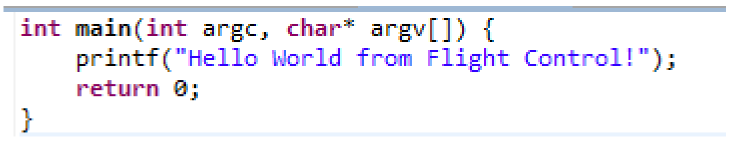
Source file: main.c

Template: Default C source template

Graphic 53: Create new source file

Enter your file-name in this case main.c and click "Finish".

To enter code just double-click on your new file and it opens into the Code Editor.
In our case just a main method with a printf function is added to create a simple hello world program.

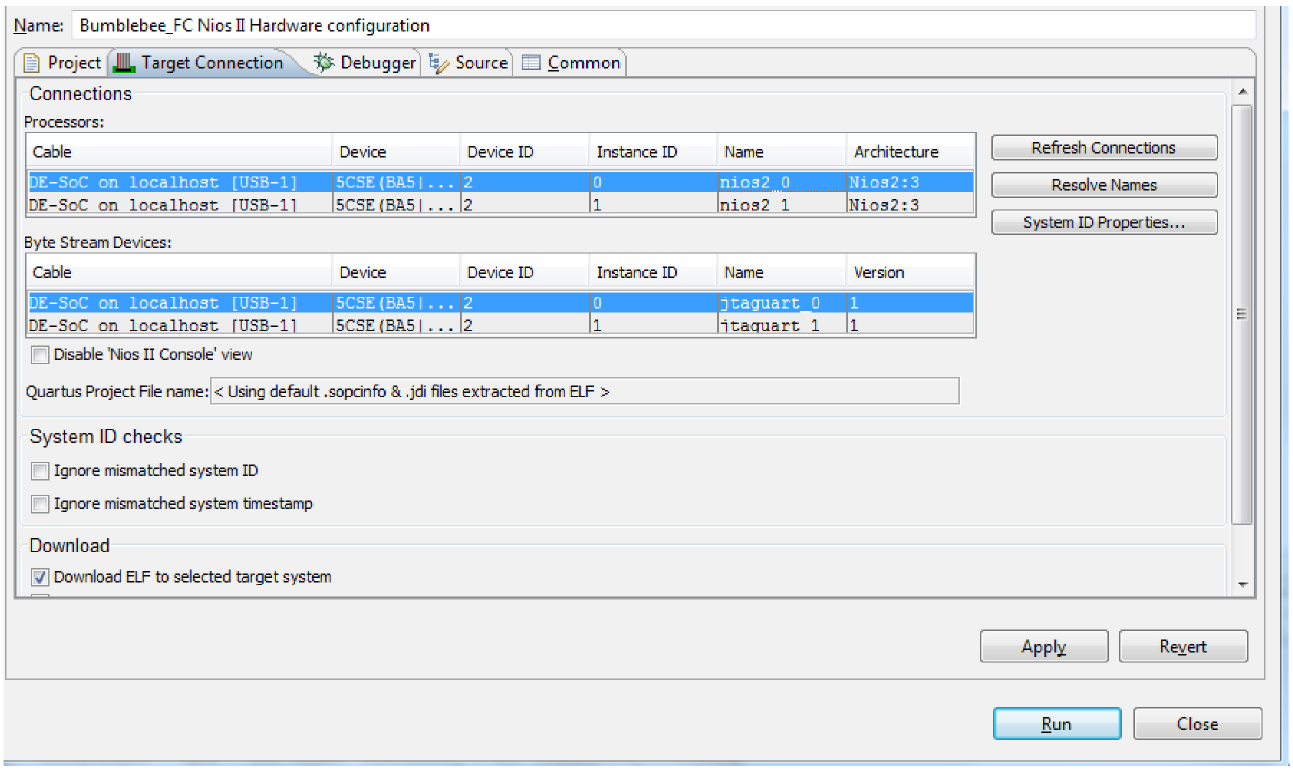


```c
int main(int argc, char* argv[]) {
    printf("Hello World from Flight Control!");
    return 0;
}
```

Graphic 54: Add source code

To run the program just right-click on your project and select Run As -> Nios II Hardware.
If the project wasn't built till now it should be built automatically.

A dialogue that looks as follows should appear.



Name: Bumblebee_FC Nios II Hardware configuration

Connections

Processors:

Cable	Device	Device ID	Instance ID	Name	Architecture
DE-SoC on localhost [USB-1]	5CSE(BA5|...	2	0	nios2_0	Nios2:3
DE-SoC on localhost [USB-1]	5CSE(BA5|...	2	1	nios2_1	Nios2:3

Byte Stream Devices:

Cable	Device	Device ID	Instance ID	Name	Version
DE-SoC on localhost [USB-1]	5CSE(BA5|...	2	0	jtaguart_0	1
DE-SoC on localhost [USB-1]	5CSE(BA5|...	2	1	jtaguart_1	1

☐ Disable 'Nios II Console' view

Quartus Project File name: < Using default .sopcinfo & .jdi files extracted from ELF >

System ID checks

☐ Ignore mismatched system ID

☐ Ignore mismatched system timestamp

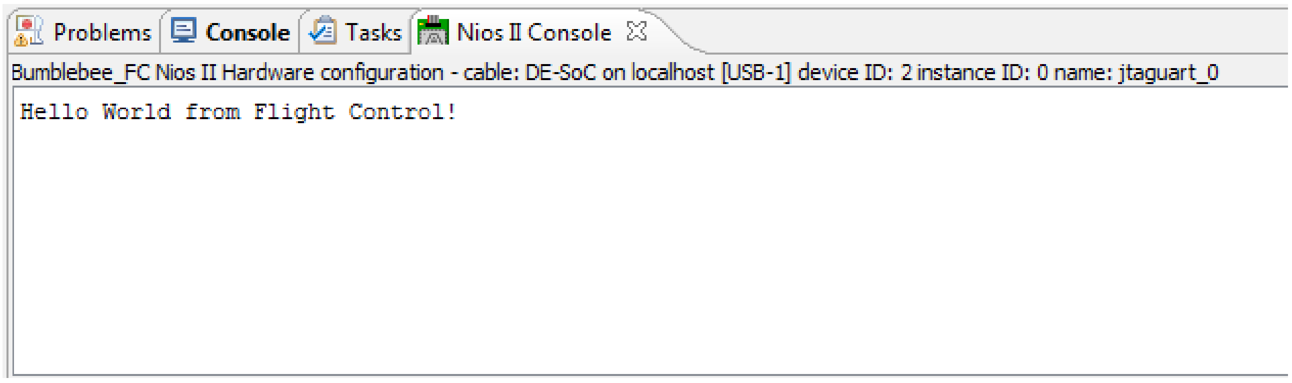
Download

☑ Download ELF to selected target system

Graphic 55: Setup connection

Here the Target Connection register should be selected to configure the target processor.
If there are no connections displayed make sure the board is connected and powered up. Then press the "Refresh Connections" button until the processors appear. If this doesn't work it is maybe required to select one or both of the check-boxes "Ignore mismatched system ID" or "Ignore mismatched system timestamp". To run the program select the processor and press "Run".

If the program works you should get the following output into the console area.

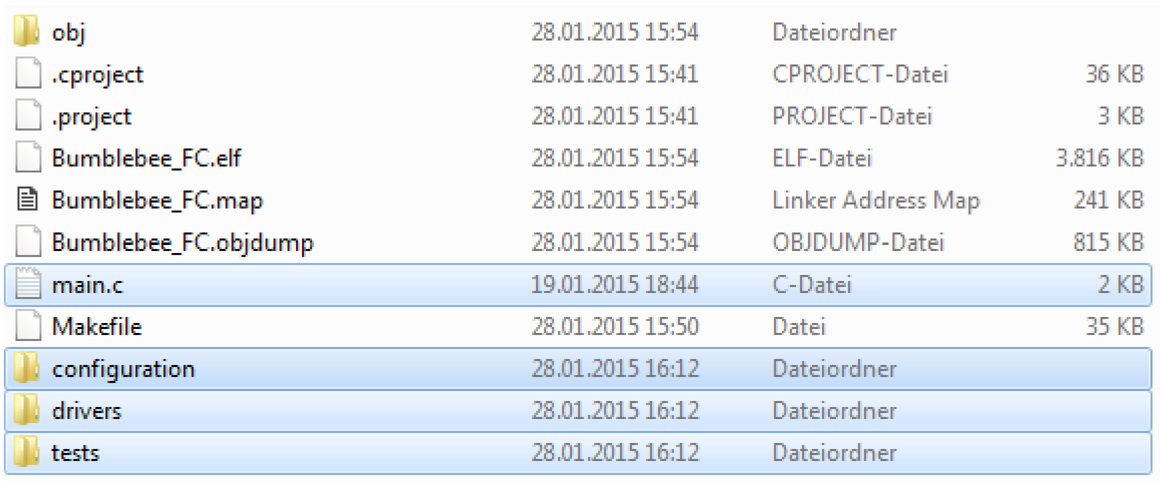


```
Bumblebee_FC Nios II Hardware configuration - cable: DE-SoC on localhost [USB-1] device ID: 2 instance ID: 0 name: jtaguart_0
Hello World from Flight Control!
```

Graphic 56: Console output

8.6 Copy Source Files to Project

To use source files that were created for this project by others just take the source and header files and copy them into your project directory. This can be simply done by using the standard file browser of your operating system.



obj	28.01.2015 15:54	Dateiordner	
.cproject	28.01.2015 15:41	CPROJECT-Datei	36 KB
.project	28.01.2015 15:41	PROJECT-Datei	3 KB
Bumblebee_FC.elf	28.01.2015 15:54	ELF-Datei	3.816 KB
Bumblebee_FC.map	28.01.2015 15:54	Linker Address Map	241 KB
Bumblebee_FC.objdump	28.01.2015 15:54	OBJDUMP-Datei	815 KB
main.c	19.01.2015 18:44	C-Datei	2 KB
Makefile	28.01.2015 15:50	Datei	35 KB
configuration	28.01.2015 16:12	Dateiordner	
drivers	28.01.2015 16:12	Dateiordner	
tests	28.01.2015 16:12	Dateiordner	

Graphic 57: Copy source files

To update these changes into the Nios II EDS IDE right-click on your project and click "Refresh". After that you should be able to build the project and run it.