phyBOARD®-Mira i.MX 6

Application Guide

Document No.:	**L-806e_1**
SBC Prod. No.:	**PB-01501-xxx**
CB PCB No.:	**1434.2**
SOM PCB No.:	**1429.2**

Edition: August 2015

	EUROPE	NORTH AMERICA	FRANCE
Address:	PHYTEC Messtechnik GmbH Robert-Koch-Str. 39 D-55129 Mainz GERMANY	PHYTEC America LLC 203 Parfitt Way SW Bainbridge Island, WA 98110 USA	PHYTEC France 17, place Saint-Etienne F-72140 Sillé-le-Guillaume FRANCE
Sales:	+49 6131 9221-32 sales@phytec.de	+1 800 278-9913 sales@phytec.com	+33 2 43 29 22 33 info@phytec.fr
Technical Support:	+49 6131 9221-31 support@phytec.de	+1 206 780-9047 support@phytec.com	support@phytec.fr
Fax:	+49 6131 9221-33	+1 206 780-9135	+33 2 43 29 22 34
Web Site:	http://www.phytec.de http://www.phytec.eu	http://www.phytec.com	http://www.phytec.fr

	INDIA	CHINA
Address:	PHYTEC Embedded Pvt. Ltd. #16/9C, 3rd Main, 3rd Floor, 8th Block, Opp. Police Station Koramangala, Bangalore-560095 INDIA	PHYTEC Information Technology (Shenzhen) Co. Ltd. Suite 2611, Floor 26, Anlian Plaza, 4018 Jin Tian Road Futian District, Shenzhen CHINA 518026
Sales:	+91-80-4086 7046/48 sales@phytec.in	+86-755-3395-5875 sales@phytec.cn
Technical Support:	+91-80-4086 7047 support@phytec.in	support@phytec.cn
Fax:		+86-755-3395-5999
Web Site:	http://www.phytec.in	http://www.phytec.cn

1st Edition August 2015

List of Figures

List of Tables

Conventions, Abbreviations and Acronyms

This hardware manual describes the PB-01501-XXX Single Board Computer (SBC) in the following referred to as phyBOARD-Mira i.MX 6. The manual specifies the phyBOARD-Mira i.MX 6's design and function. Precise specifications for the Freescale Semiconductor i.MX 6 microcontrollers can be found in the Freescale Semiconductor's i.MX 6 Data Sheet and Technical Reference Manual.

Conventions

The conventions used in this manual are as follows:

- Signals that are preceded by an "n", "/", or "#"character (e.g.: nRD, /RD, or #RD), or that have a dash on top of the signal name (e.g.: $\overline{\text{RD}}$) are designated as active low signals. That is, their active state is when they are driven low, or are driving low.
- A "0" indicates a logic zero or low-level signal, while a "1" represents a logic one or high-level signal.
- The hex-numbers given for addresses of I^2C devices always represent the 7 MSB of the address byte. The correct value of the LSB which depends on the desired command (read (1), or write (0)) must be added to get the complete address byte. E.g. given address in this manual 0x41 => complete address byte = 0x83 to read from the device and 0x82 to write to the device.
- Tables which describe jumper settings show the default position in **bold, blue text.**
- Text in *blue italic* indicates a hyperlink within, or external to the document. Click these links to quickly jump to the applicable URL, part, chapter, table, or figure.
- Text in ***bold italic*** indicates an interaction by the user, which is defined on the screen.
- Text in `Consolas` indicates an input by the user, without a premade text or button to click on.
- Text in *italic* indicates proper names of development tools and corresponding controls (windows, tabs, commands etc.) used within the development tool, no interaction takes place.
- White Text on black background shows the result of any user interaction (command, program execution, etc.)

Abbreviations and Acronyms

Many acronyms and abbreviations are used throughout this manual. Use the table below to navigate unfamiliar terms used in this document.

Abbreviation	Definition
A/V	Audio/Video
BSP	Board Support Package (Software delivered with the Development Kit including an operating system (Windows, or Linux) pre-installed on the module and Development Tools)
CB	Carrier Board, used in reference to the phyBOARD-Mira Development Kit Carrier Board
DFF	D flip-flop
DSC	Direct Solder Connect
EMB	External memory bus
EMI	Electromagnetic Interference
GPI	General purpose input
GPIO	General purpose input and output
GPO	General purpose output
IRAM	Internal RAM, the internal static RAM on the Freescale Semiconductor i.MX 6 microcontroller
J	Solder jumper, these types of jumpers require solder equipment to remove and place
JP	Solderless jumper, these types of jumpers can be removed and placed by hand with no special tools
NC	Not Connected
NM	Not Mounted
NS	Not Specified
PCB	Printed circuit board
PDI	PHYTEC Display Interface, defined to connect PHYTEC display adapter boards, or custom adapters
PEB	PHYTEC Expansion Board
PMIC	Power management IC
PoE	Power over Ethernet
PoP	Package on Package
POR	Power-on reset
RTC	Real-time clock
SBC	Single Board Computer, used in reference to the PBA-C(D)-06 /phyBOARD-Mira i.MX 6
SMT	Surface mount technology
SOM	System on Module, used in reference to the PCM-058 /phyCORE-i.MX 6 module
Sx	User button Sx (e.g. S1, S2) used in reference to the available user buttons, or DIP switches on the CB
Sx_y	Switch y of DIP switch Sx, used in reference to the DIP switch on the carrier board
VSTBY	SOM standby voltage input

Table 1: Abbreviations and Acronyms used in this Manual

🪧	At this icon you might leave the path of this Application Guide.
⚠	This is a warning. It helps you to avoid annoying problems.
🦊	You can find useful supplementary information about the topic.
⏱	At the beginning of each chapter you can find information about the time required to read the following chapter.
👍	You have successfully completed an important part of this Application Guide.
🧰	You can find information to solve problems.

Note: The BSP delivered with the phyBOARD-Mira i.MX 6 usually includes drivers and/or software for controlling all components such as interfaces, memory, etc. Therefore programming close to hardware at register level is not necessary in most cases. For this reason, this manual contains no detailed description of the controller's registers. Please refer to the i.MX 6 Technical Reference Manual, if such information is needed to connect customer designed applications.

The BSP is configured according to the hardware configuration including the expansion board delivered with the kit. Thus some functions of the phyBOARD-Mira i.MX 6 might not be available if the corresponding pins and drivers are needed to support an expansion board. If the expansion board is removed, or exchanged the BSP must be exchanged, too.

From BSP version i.MX6-PD14.2-rc1 on it is possible to configure the BSP in regard to the hardware configuration. This allows to easily adapt the BSP if an expansion board is attached, removed, or exchanged.

Preface

As a member of PHYTEC's new phyBOARD® product family the phyBOARD-Mira i.MX 6 is one of a series of PHYTEC System on Modules (SBCs) that offer off-the-shelf solutions for a huge variety of industrial applications. The new phyBOARD® product family consists of a series of extremely compact embedded control engines featuring various processing performance classes. All phyBOARDs are rated for industry, cost optimized and offer long-term availability. The phyBOARD-Mira i.MX 6 is one of currently six industrial-grade carrier boards which are suitable for series production and that have been realized in accordance with PHYTEC's new SBCplus concept. It is an excellent example of this concept.

SBCplus Concept
The SBCplus concept was developed to meet fine differences in customer requirements with little development effort and thus to greatly reduce the time-to-market.

Core of the SBCplus concept is the SBC design library (a kind of construction set) that consists of a great number of function blocks (so-called "building blocks") which are refined constantly. The recombination of these function blocks allows to develop a customer specific SBC within a short time. Thus, PHYTEC is able to deliver production-ready custom Single Board Computers within a few weeks at very low costs.

The already developed SBCs, such as the phyBOARD-Mira, each represent an intersection of different customer wishes. Because of that all necessary interfaces are already available on the standard versions, thus, allowing to integrate them in a large number of applications without modification. For any necessary detail adjustment extension connectors are available to enable adding of a wide variety of functions.

Cost-optimized with Direct Solder Connect (DSC) Technology
At the heart of the phyBOARD-Mira is the phyCORE-i.MX 6 System on Module (SOM). As with other SBCs of the phyBOARD® family the SOM can be directly soldered onto the carrier board PCB for routing of signals from the SOM to applicable I/O interfaces. This optional “Direct Solder Connect” (DSC) of the SOM eliminates costly PCB to PCB connectors, thereby further reducing overall system costs, and making the phyBOARDs ideally suited for deployment into a wide range of cost-optimized and robust industrial applications.

Customized Expandability from PHYTEC
Common interface signals route to standard connector interfaces on the carrier board such as Ethernet, CAN, RS-232, and audio. Due to the easily modifiable phyBOARD design approach (see "*SBCplus concept*"), these plug-and-play interfaces can be readily adapted in customer-specific variants according to end system requirements.

Some signals from the processor populating the SOM also extend to the expansion, and A/V connectors of the phyBOARD-Mira. This provides for customized expandability according to end user requirements. Thus expandability is made easy by available plug-and-play expansion modules from PHYTEC.

- HDMI and LVDS/Parallel Displays
- Power Supply, with broad voltage range
- Industrial I/O (including WLAN)
- Home-Control Board (WiFi, KNX/EIB, I/O)
- M2M Board (GPS, GSM, I/O's)
- Debug Adapter

The default orientation of the expansion bus connectors is parallel and on the top side of the carrier board PCB. However, in custom configurations the connectors can be mounted on the PCB's underside. Connectors in perpendicular orientation can also populate the top or underside of the PCB. This enables maximum flexibility for orientation of expansion modules on the phyBOARD-Mira, as well as integration of the system into a variety of end application physical envelopes and form factors.

Easy Integration of Display und Touch
The phyBOARD and its expansion modules enable easy connection of parallel or LVDS based displays, as well as resistive or capacitive touch screens.

OEM Implementation
Implementation of an OEM-able SBC subassembly as the "core" of your embedded design allows you to focus on hardware peripherals and firmware without expending resources to "re-invent" microcontroller circuitry. Furthermore, much of the value of the phyBOARD® SBC lies in its layout and test.

Software Support
Production-ready Board Support Packages (BSPs) and Design Services for our hardware will further reduce your development time and risk and allow you to focus on your product expertise.

Ordering Information

Ordering numbers:

phyBOARD-Mira i.MX 6 Development Kit:	**KPB-01501-xxx**
phyBOARD-Mira i.MX 6 SBC:	**PB-01501-xxx**

Product Specific Information and Technical Support

In order to receive product specific information on changes and updates in the best way also in the future, we recommend to register at

http://www.phytec.de/de/support/registrierung.html or
http://www.phytec.eu/europe/support/registration.html

For technical support and additional information concerning your product, please visit the support section of our web site which provides product specific information, such as errata sheets, application notes, FAQs, etc.

http://www.phytec.de/de/support/faq/faq-phyBOARD-Mira-i.MX6.html or
http://www.phytec.eu/europe/support/faq/faq-phyBOARD-Mira-i.MX6.html

Other Products and Development Support

Aside of the new phyBOARD® familiy, PHYTEC supports a variety of 8-/16- and 32-bit controllers in two ways:

(1) as the basis for Rapid Development Kits which serve as a reference and evaluation platform

(2) as insert-ready, fully functional OEM modules, which can be embedded directly into the user's peripheral hardware design.

Take advantage of PHYTEC products to shorten time-to-market, reduce development costs, and avoid substantial design issues and risks. With this new innovative full system solution you will be able to bring your new ideas to market in the most timely and cost-efficient manner.

For more information go to:

http://www.phytec.de/de/leistungen/entwicklungsunterstuetzung.html or
www.phytec.eu/europe/oem-integration/evaluation-start-up.html

Declaration of Electro Magnetic Conformity of the PHYTEC phyBOARD-Mira i.MX 6

PHYTEC Single Board Computers (henceforth products) are designed for installation in electrical appliances, or as part of custom applications, or as dedicated Evaluation Boards (i.e.: for use as a test and prototype platform for hardware/software development) in laboratory environments.

Caution!
PHYTEC products lacking protective enclosures are subject to damage by ESD and, hence, may only be unpacked, handled or operated in environments in which sufficient precautionary measures have been taken in respect to ESD-dangers. It is also necessary that only appropriately trained personnel (such as electricians, technicians and engineers) handle and/or operate these products. Moreover, PHYTEC products should not be operated without protection circuitry if connections to the product's pin header rows are longer than 3 m.

PHYTEC products fulfill the norms of the European Union’s Directive for Electro Magnetic Conformity only in accordance to the descriptions and rules of usage indicated in this hardware manual (particularly in respect to the pin header row connectors, power connector and serial interface to a host-PC).

Implementation of PHYTEC products into target devices, as well as user modifications and extensions of PHYTEC products, is subject to renewed establishment of conformity to, and certification of, Electro Magnetic Directives. Users should ensure conformance following any modifications to the products as well as implementation of the products into target systems.

Product Change Management and information in this manual on parts populated on the SOM / SBC

When buying a PHYTEC SOM / SBC, you will, in addition to our HW and SW offerings, receive a free obsolescence maintenance service for the HW we provide.

Our PCM (Product Change Management) Team of developers, is continuously processing, all incoming PCN's (Product Change Notifications) from vendors and distributors concerning parts which are being used in our products.

Possible impacts to the functionality of our products, due to changes of functionality or obsolesce of a certain part, are being evaluated in order to take the right masseurs in purchasing or within our HW/SW design.

Our general philosophy here is: **We never discontinue a product as long as there is demand for it.**

Therefore we have established a set of methods to fulfill our philosophy:

Avoiding strategies

- Avoid changes by evaluating long-livety of parts during design in phase.
- Ensure availability of equivalent second source parts.
- Stay in close contact with part vendors to be aware of roadmap strategies.

Change management in case of functional changes

- Avoid impacts on product functionality by choosing equivalent replacement parts.
- Avoid impacts on product functionality by compensating changes through HW redesign or backward compatible SW maintenance.
- Provide early change notifications concerning functional relevant changes of our products.

Change management in rare event of an obsolete and non replaceable part

- Ensure long term availability by stocking parts through last time buy management according to product forecasts.
- Offer long term frame contract to customers.

Therefore we refrain from providing detailed part specific information within this manual, which can be subject to continuous changes, due to part maintenance for our products.

In order to receive reliable, up to date and detailed information concerning parts used for our product, please contact our support team through the contact information given within this manual.

1 Introduction

1.1 Hardware Overview

The phyBOARD-Mira for phyCORE-i.MX 6 is a low-cost, feature-rich software development platform supporting the Freescale Semiconductor i.MX 6 microcontroller. Moreover, due to the numerous standard interfaces the phyBOARD-Mira i.MX 6 can serve as bedrock for your application. At the core of the phyBOARD-Mira is the PCM-058/phyCORE-i.MX 6 System On Module (SOM), containing the processor, DRAM, NAND Flash, power regulation, supervision, transceivers, and other core functions required to support the i.MX 6 processor. Surrounding the SOM is the PBA-CD-06/phyBOARD-Mira carrier board, adding power input, buttons, connectors, signal breakout, and Ethernet connectivity amongst other peripherals.

The PCM-058 System On Module connects to the phyBOARD-Mira carrier board by use of two high density connectors. The phyBOARD-Mira is also available with the phyCORE-i.MX 6 in a direct solder form factor (PCL-058), a connector-less, BGA style variant of the PCM-058/phyCORE-i.MX 6 SOM. The PCL-058 SOM is directly soldered down to the phyBOARD-Mira using PHYTEC's Direct Solder Connect technology. This solution offers an ultra-low cost Single Board Computer for the i.MX 6 processor, while maintaining most of the advantages of the SOM concept.

Adding the phyCORE-i.MX 6 SOM into your own design is as simple as ordering the connectored version (PCM-058) and making use of our phyCORE Carrier Board reference schematics.

1.1.1 Features of the phyBOARD-Mira i.MX 6

The phyBOARD-Mira i.MX 6 supports the following features :

- Developed in accordance with PHYTEC's new SBCplus concept (*Preface*)
- PHYTEC’s phyCORE-i.MX 6 SOM (optionally with Direct Solder Connect (DSC))
- Pico ITX standard dimensions (100 mm × 72 mm)
- Boot from MMC or NAND Flash
- Max. 1.2 GHz core clock frequency and up to four cores
- Two different power supply options (5 V via 3.5 mm combicon or 12 V – 24 V through external power module)
- One RJ45 jacks for 10/100/1000 Mbps Ethernet
- One USB host interface brought out to an upright USB Standard-A connector, or at the Mini PCI express connector[1]
- One USB OTG interface available at an USB Micro-AB connector, or at the expansion connector[1]

1 : **Caution!** There is no protective circuit for the USB interfaces brought out at the Mini PCI Express connector and the expansion connector.

- One Secure Digital / Multi Media Memory Card interface brought out to a Micro-SD connector at the back side
- CAN interface at 2×5 pin header 2.54 mm
- One HDMI interface brought out to a standard type A connector
- One LVDS interface brought out to a 20 pin FFC connector at the backside, and separate connector for backlight supply and control
- One touch interface at 1x4 pin header 2.54 mm
- One LVDS camera interfaces compatible to PHYTEC phyCAM-S+ camera standard with I^2C for camera control
- One PCI interface brought out to a Mini PCI Express connector, SIM-card signals are also available at the expansion connector
- RS-232 or RS-485 transceiver supporting UART3 incl. handshake signals with data rates of up to 1 Mbps (2×5 pin header 2.54 mm)
- Reset-Button
- One multicolor LED
- Audio/Video (A/V) connectors
- Expansion connector with different interfaces
- RTC
- Backup battery supply for RTC with external 2-pole pin-header or with Gold cap (lasts approx. 11 ½ days)

1.1.2 Block Diagram

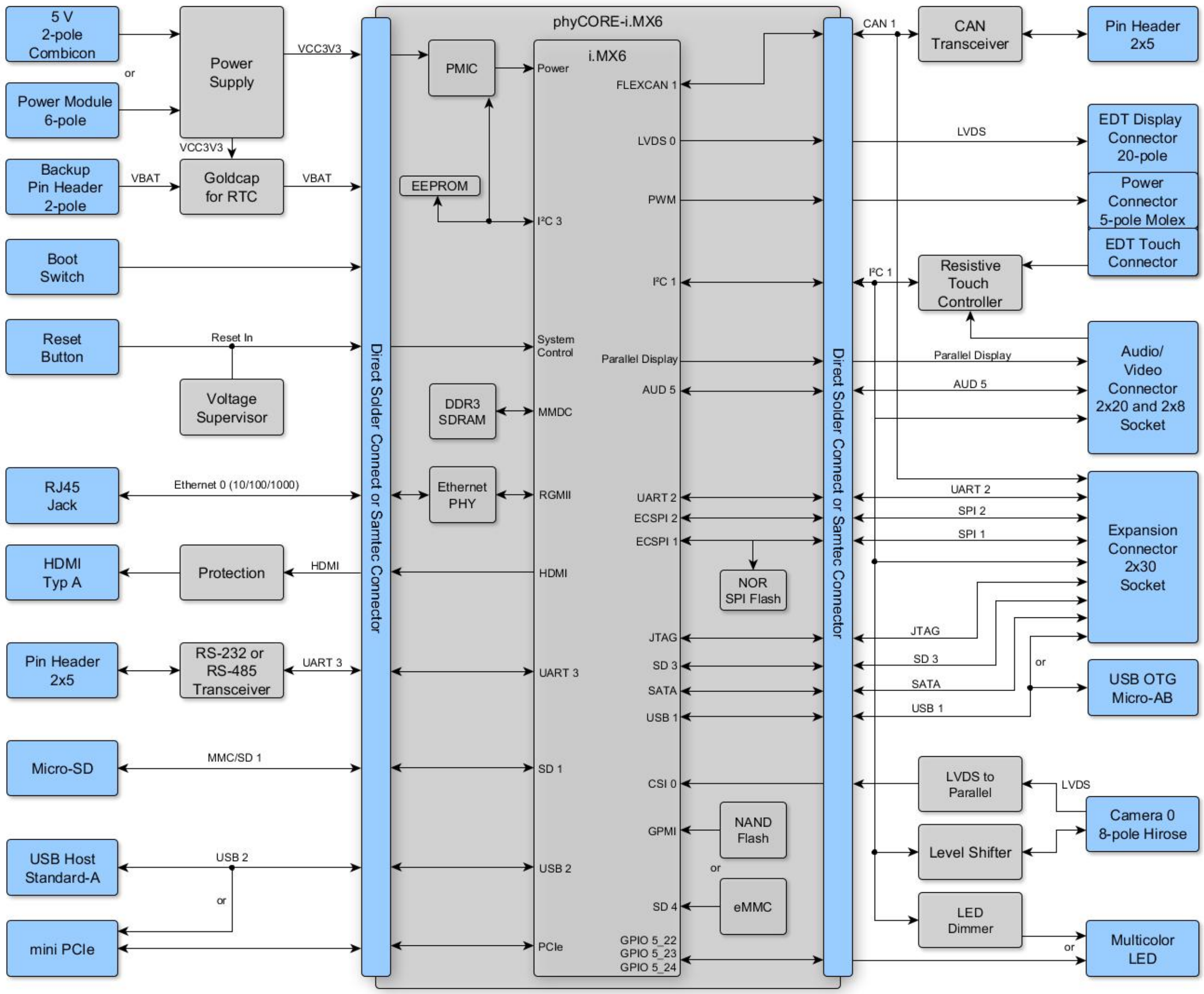


Figure 1: Block Diagram of the phyBOARD-Mira i.MX 6

1.1.3 View of the phyBOARD-Mira i.MX 6

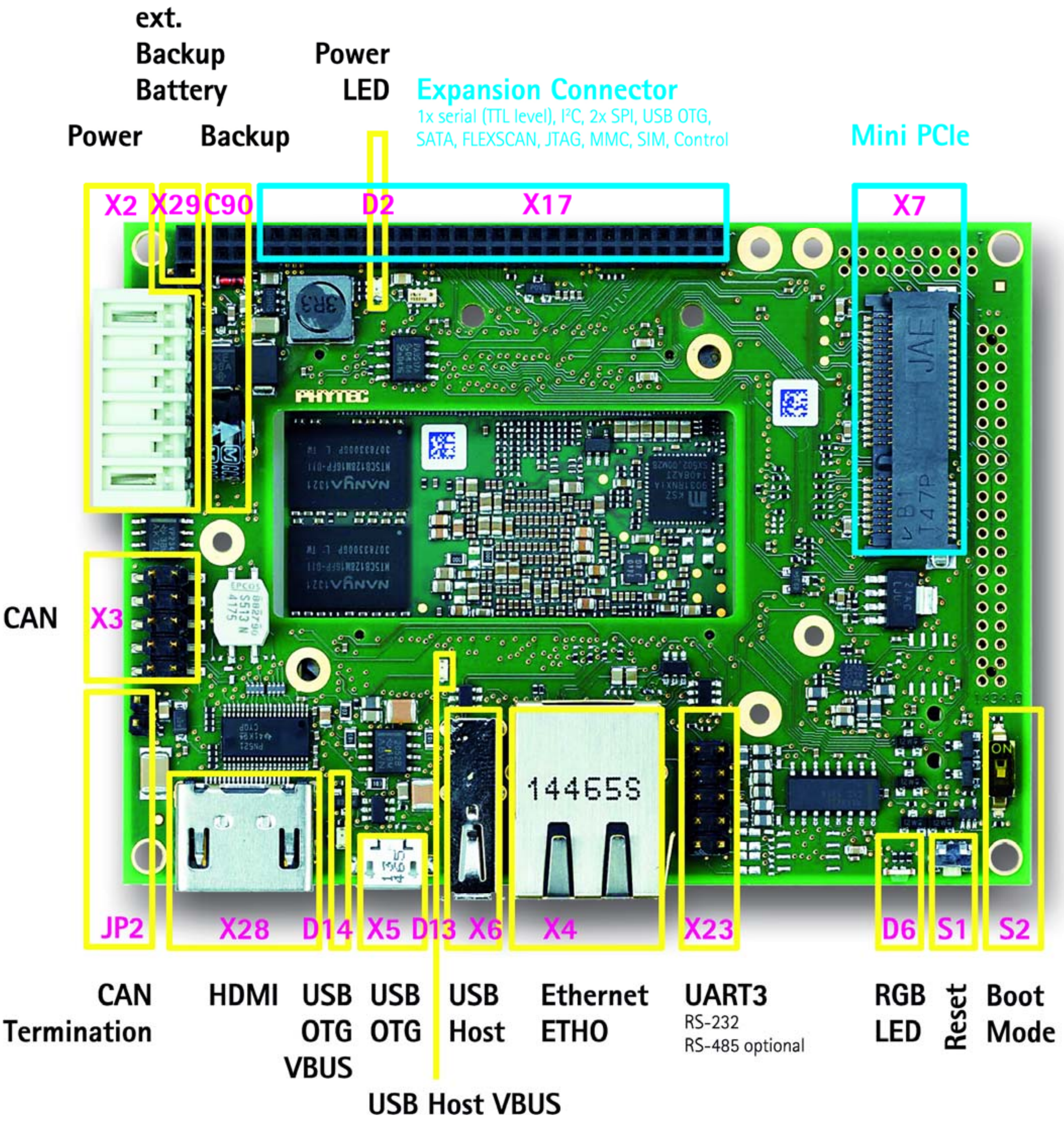


Figure 2: View of the phyBOARD-Mira i.MX 6 (top)

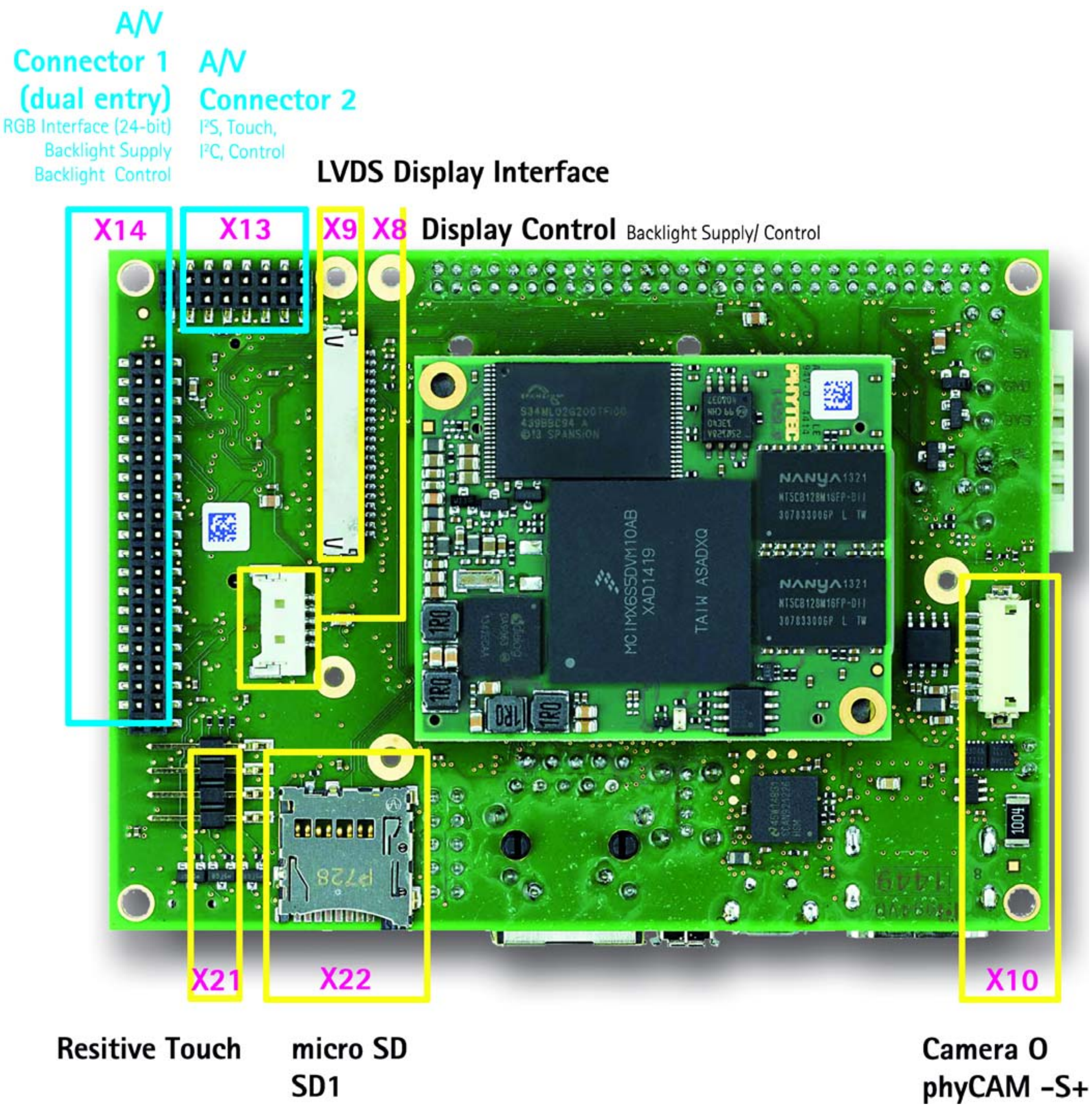


Figure 3: View of the phyBOARD-Mira i.MX 6 (bottom)

1.2 Software Overview

1.2.1 Ubuntu

Ubuntu - which you is used as operating system for our Live System - is a free and open source operating system based on *Debian Linux*. Basically it is designed for desktop use. Web statistics suggest that *Ubuntu* is one of the most popular operating systems in the Linux desktop environment.

The *Ubuntu* release which we deliver is *14.04.3* and was released on 06. August 2015. *Ubuntu* 14.04 code name "*Trusty Tahr*" is designated as a **Long Term Support (LTS)** release and the first stable release was on 17. April 2014. LTS means that it will be supported and updated for five years.

Our *Ubuntu* version comes with *Unity* as desktop environment, *dpkg* as package management system, the update method is based on *APT (Advanced Packaging Tool)* and the user space uses *GNU*.

1.2.2 Eclipse

The *Eclips*e platform provides support for *C/C*++. Because the *Eclipse* platform is only a framework for developer tools, it does not support *C/C*++ directly, instead it uses external plug-ins. This Application Guide shows you how to make use of the *CDT plug-in*.

The *CDT* is an open source project (licensed under the Common Public License) implemented purely in *Java* as a set of plug-ins for the *Eclipse* SDK platform. These plug-ins add a *C/C*++ perspective to the *Eclipse* Workbench that can now support *C/C*++ development with a number of views and wizards, along with advanced editing and debugging support.

1.2.3 Qt Creator

Qt Creator is a cross-platform development environment for the Qt framework. Included are a code editor and a Qt Designer to build graphical user interfaces (GUI). It uses the *GNU C/C*++ compiler.

1.2.4 Yocto Project

The *Yocto* Project is an open source collaboration to create custom Linux-based systems for embedded products regardless of the hardware architecture. We use the *Yocto* Project to create the Board Support Package (BSP) for our hardware.

2 Application Programming

During this chapter you will learn how to build your own C/C++ applications for the target with the help of Eclipse.

We assume that you have first completed our QuickStart Guide successfully.

As all changes on the example projects will be lost if you proceed using the live environment we recommend to now install our modified *Ubuntu* Live System. If you only want to make your own fast experience with our phyBOARD-Mira i.MX 6-Kit you can continue with section *2.2 "Working with Eclipse"*.

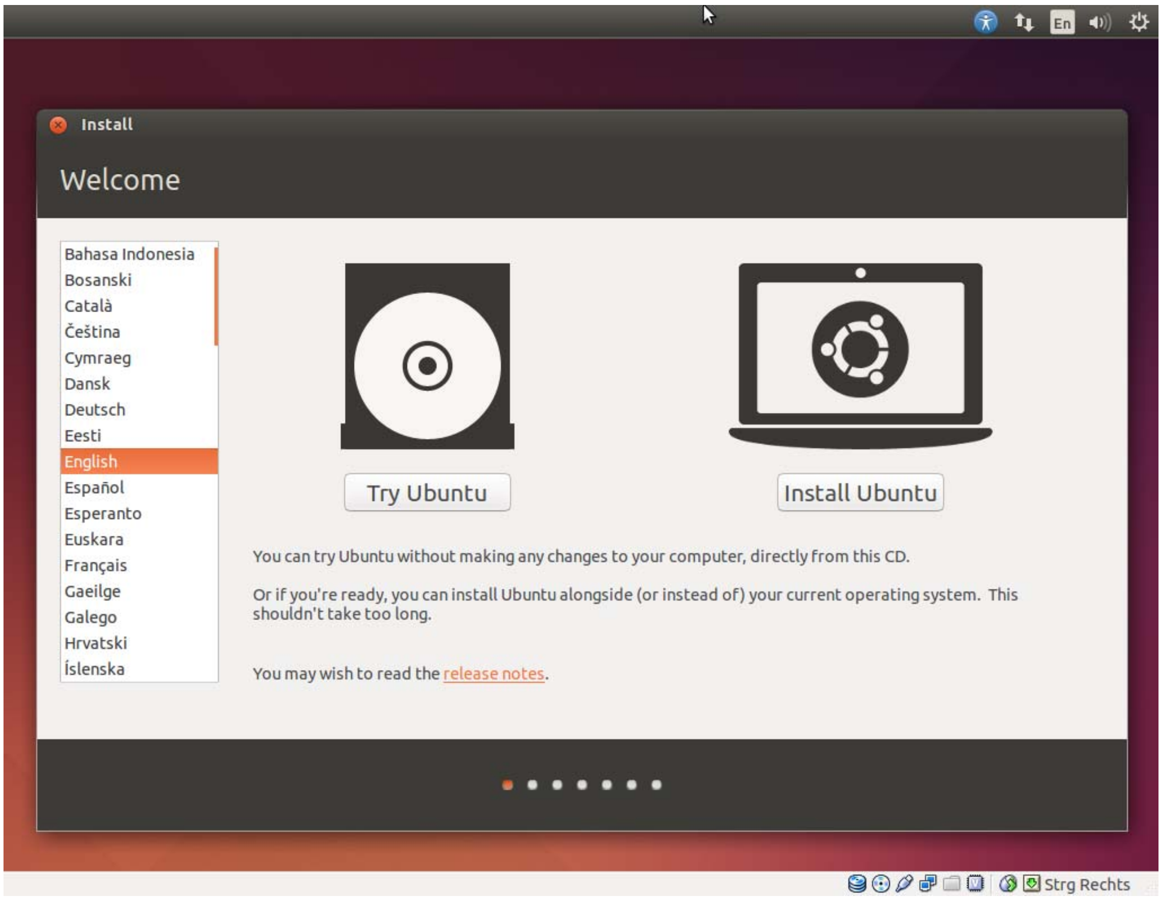
To ensure successful introduction to the development with the phyBOARD-Mira i.MX 6 we strongly recommend continuing with the modified *Ubuntu*, either in a live environment, or completely installed on your PC as described in the next section.
Nonetheless, if you want to use your already existing environment we explain how to modify your system to get the same experience as with our Live System in *section 4.5*.

2.1 Installing our modified Ubuntu Live System

As described above, this step is not needed to successfully finish this chapter but for more in-depth development it is better to install our Live System on your computer or into a virtual machine. If you have purchased our bootable USB flash drive plug it into your PC and restart the PC. Alternatively, you can download a virtual machine hard disk image as explained in the Quickstart Guide. In this case the installation is already done in the virtual machine and you can step to section 2.2.

We recommend to have at least 80 GB of free disk space available to install the customized Live System.

- Boot the modified *Ubuntu* from the bootable USB flash drive.
- Select ***Install Ubuntu*** in the first *Welcome* window.

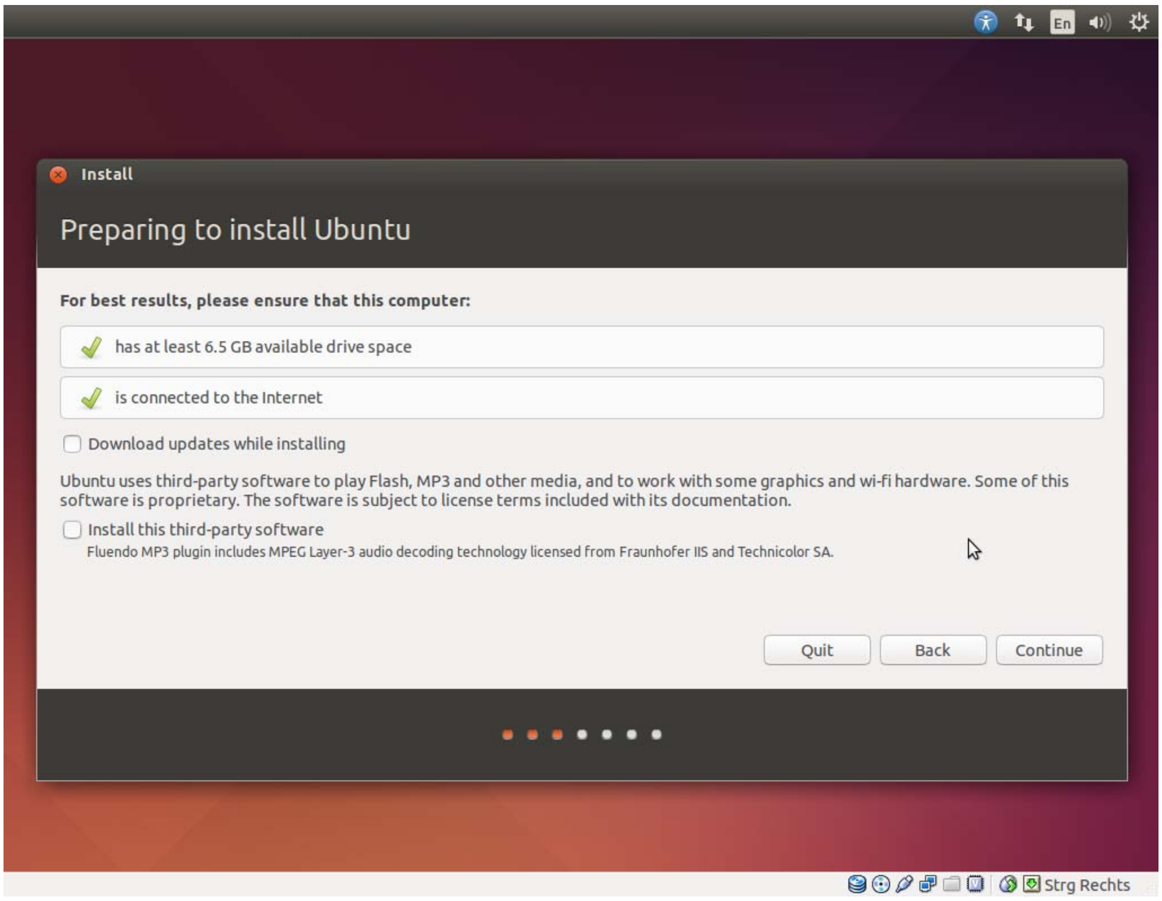


You can try Ubuntu without making any changes to your computer, directly from this CD.

Or if you're ready, you can install Ubuntu alongside (or instead of) your current operating system. This shouldn't take too long.

You may wish to read the release notes.

- The *Preparing to install Ubuntu…* window appears. From *Ubuntu* it is advised that you select ***Download updates while installing*** and ***Install this third-party software now***.



For best results, please ensure that this computer:

✓ has at least 6.5 GB available drive space

✓ is connected to the Internet

☐ Download updates while installing

Ubuntu uses third-party software to play Flash, MP3 and other media, and to work with some graphics and wi-fi hardware. Some of this software is proprietary. The software is subject to license terms included with its documentation.

☐ Install this third-party software

Fluendo MP3 plugin includes MPEG Layer-3 audio decoding technology licensed from Fraunhofer IIS and Technicolor SA.

- Click ***Continue***.
- The *Installation type* window appears. You now have different options how to install *Ubuntu*. Depending on your system you have a number of possibilities that are shown in the dialog. After you have chosen one click ***Continue***.
- The *Install Ubuntu...* window appears. After you have checked the settings you can click ***Install now***.
- While the installation is started *Ubuntu* asks for your location, keyboard layout and login and password details. Please insert this information and wait until the installation is finished.
- Finally you must restart your system after the installation is finished.
- After that the system boots up and you can log into *Ubuntu*. Please configure your network connection now and connect the phyBOARD-Mira-i.XM 6 to your host. How to do that is shown in our QuickStart Guide.

2.2 Working with Eclipse

Now we start developing our own applications with the help of *Eclipse*.

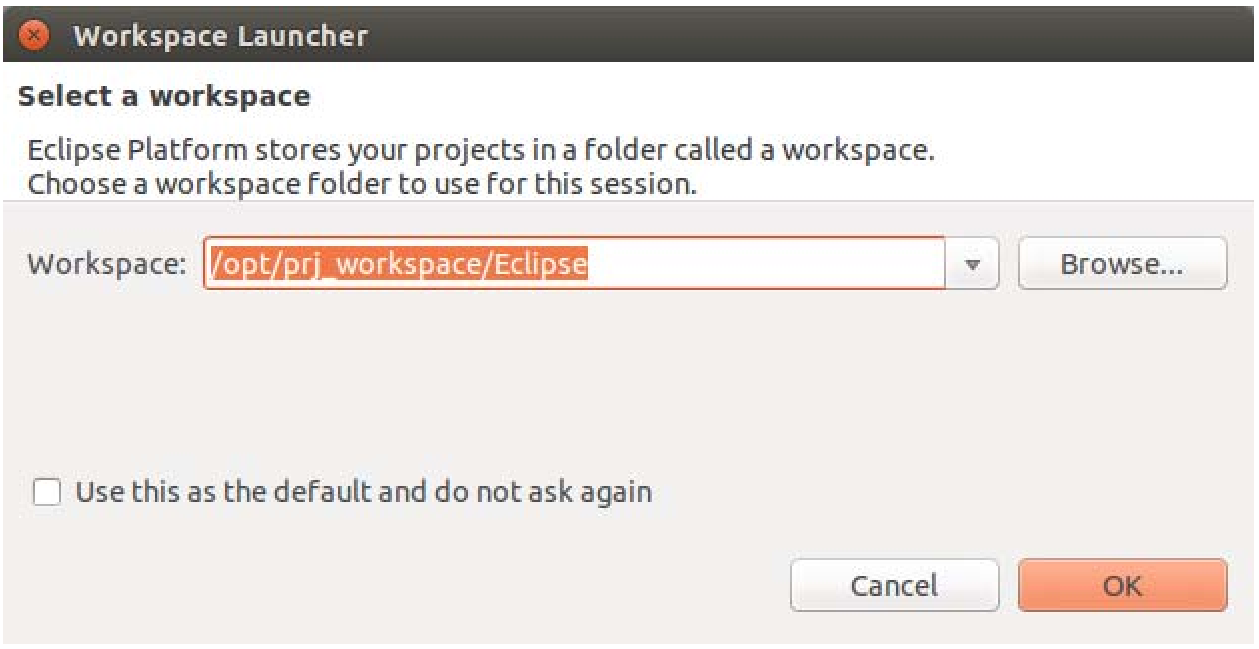
2.2.1 Programming in the C/C++ Perspective

We are starting with the *C/C++* workbench. Therefore you will import an existing Eclipse project into your workspace. The imported example project will be compiled with the cross compiler. After that, you will copy and execute the newly created program on the target.

2.2.1.1 Work with the Demo Project

- Click the **Eclipse icon** to start the application. You can find this icon on your desktop.

- Change the workspace directory to `/opt/prj_workspace/Eclipse` if necessary and confirm with ***OK***.



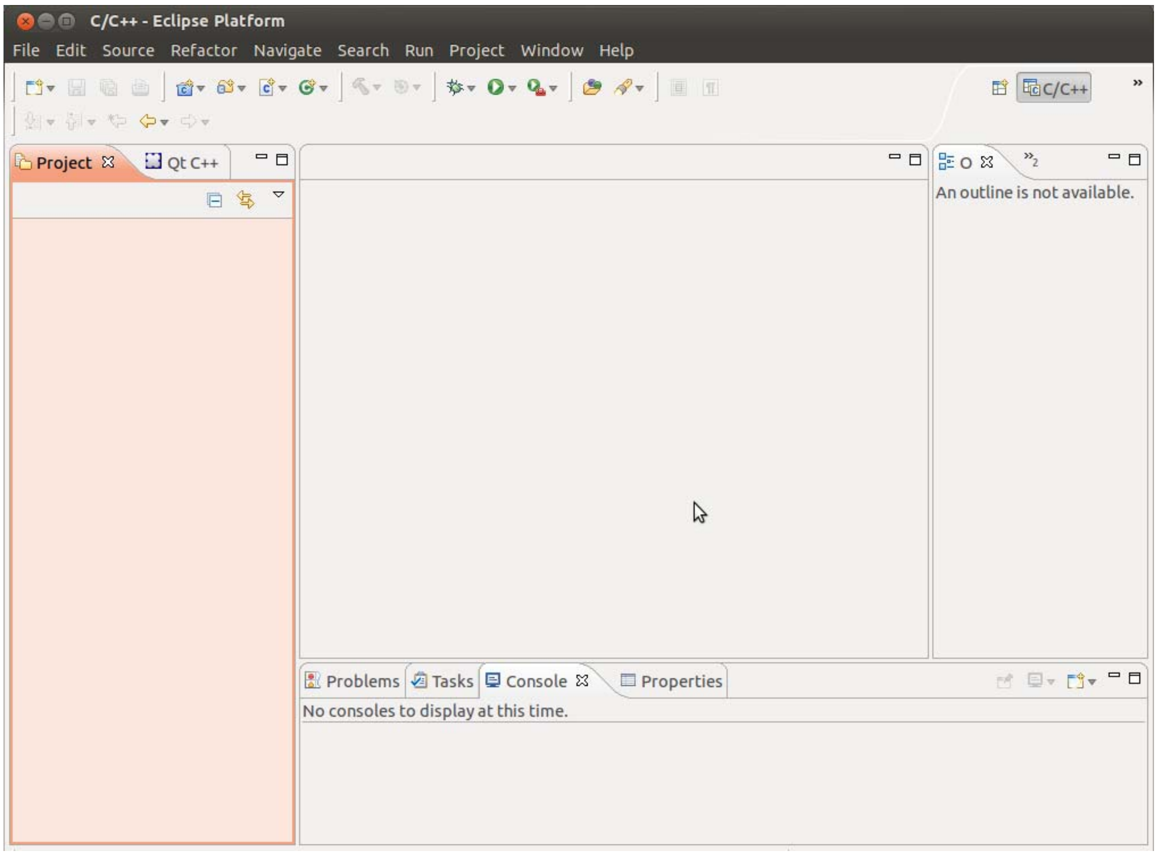
Workspace Launcher

Select a workspace

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

Workspace: /opt/prj_workspace/Eclipse

Now you can see the *Eclipse* workbench.

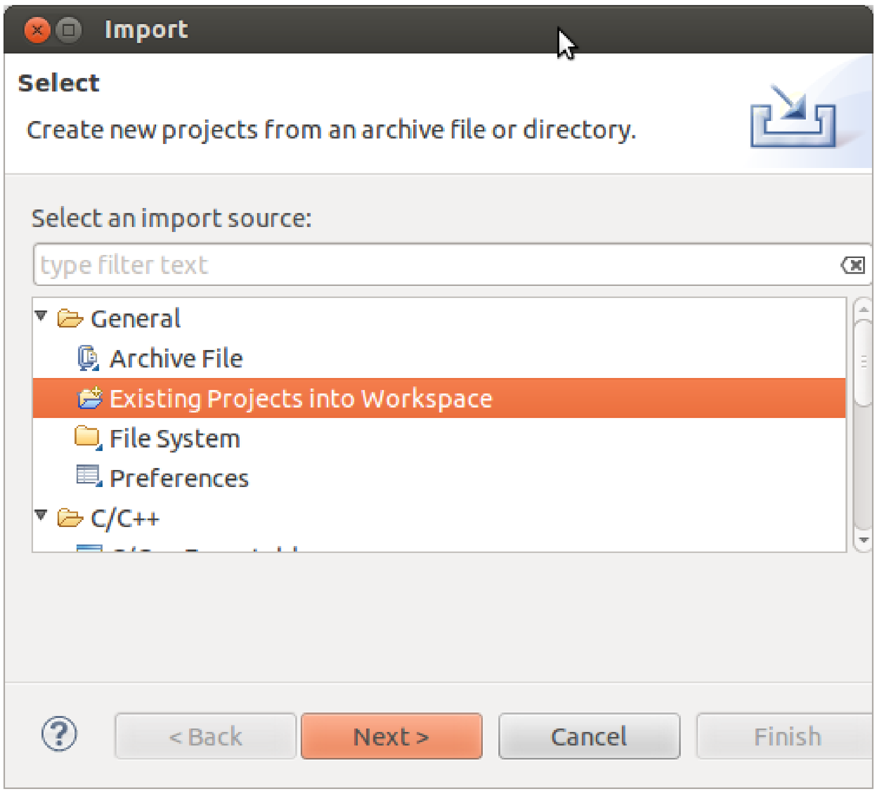


An outline is not available.

No consoles to display at this time.

First we will import an existing project.

- Select ***File* ▶ *Import*** from the menu bar.
- Select ***Existing Projects into Workspace*** and click ***Next***.

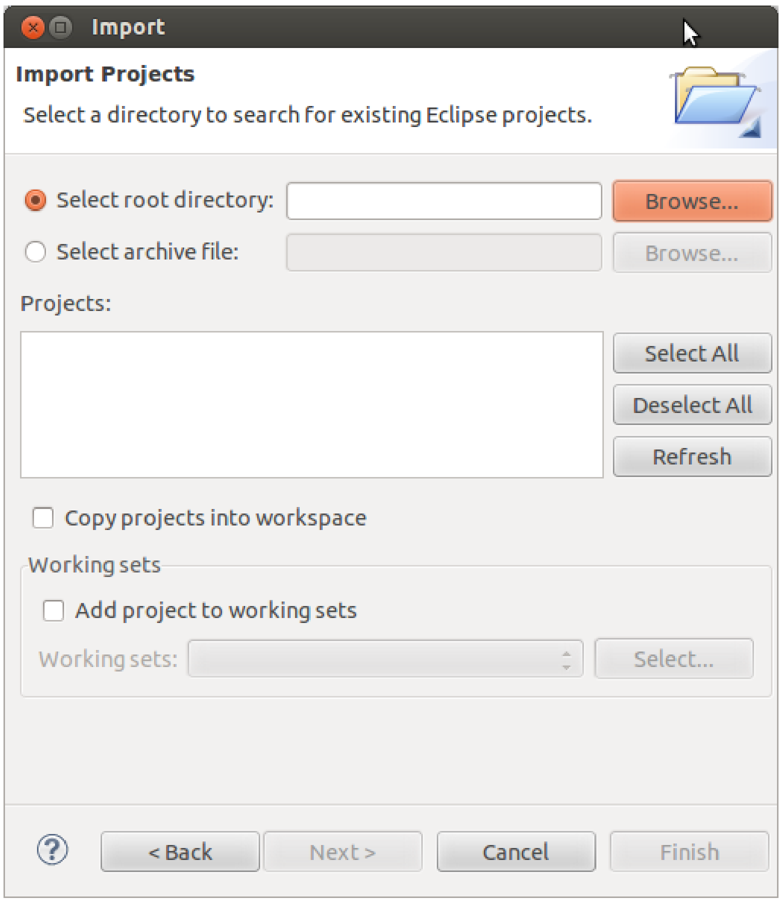


Create new projects from an archive file or directory.

Select an import source:

type filter text

- Select ***Browse***.



Import

Import Projects

Select a directory to search for existing Eclipse projects.

[✓ Select root directory:] Browse...

Select archive file: Browse...

Projects:

Select All / Deselect All / Refresh

[ ] Copy projects into workspace

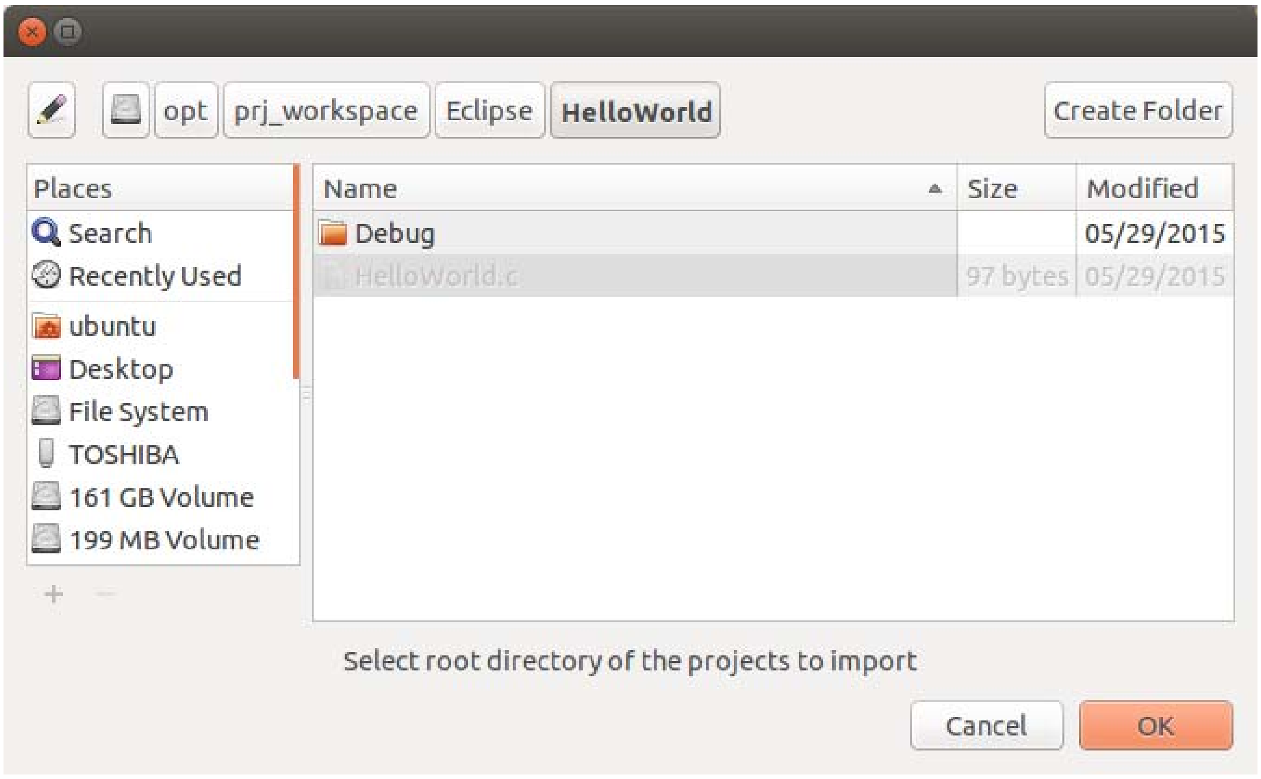
Working sets

[ ] Add project to working sets

Working sets: Select...

< Back | Next > | Cancel | Finish

- Double-click the ***HelloWorld*** directory under */opt/prj_workspace/Eclipse/*.



opt | prj_workspace | Eclipse | **HelloWorld** | Create Folder

Places
Search
Recently Used
ubuntu
Desktop
File System
TOSHIBA
161 GB Volume
199 MB Volume

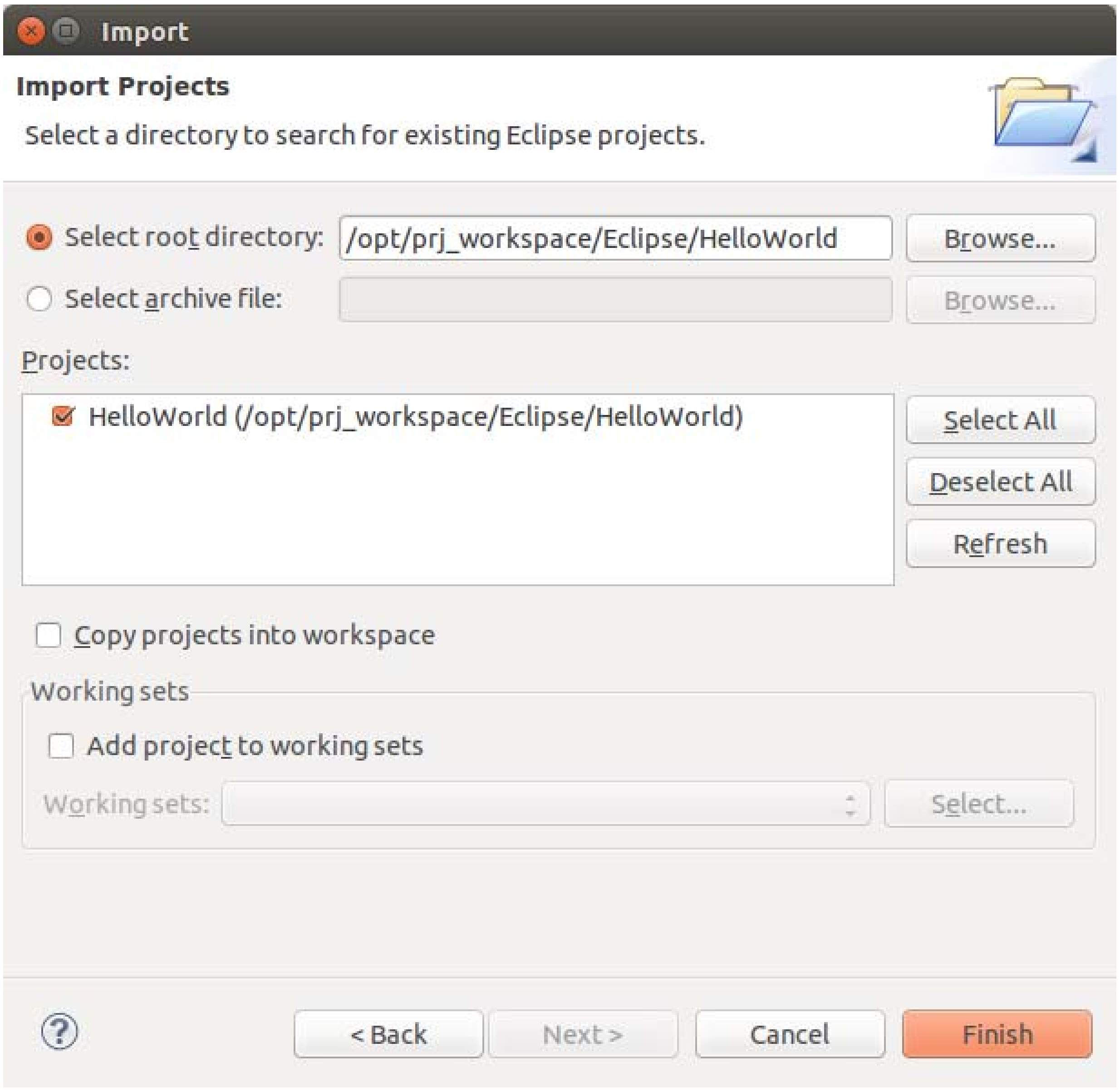
Name	Size	Modified
Debug		05/29/2015
HelloWorld.c	97 bytes	05/29/2015

Select root directory of the projects to import

Cancel | OK

- Click ***OK***.

- Select ***Finish*** to import the project.



Import

Import Projects

Select a directory to search for existing Eclipse projects.

[✓ Select root directory:] /opt/prj_workspace/Eclipse/HelloWorld

Select archive file:

Projects:

✓ HelloWorld (/opt/prj_workspace/Eclipse/HelloWorld)

☐ Copy projects into workspace

Working sets

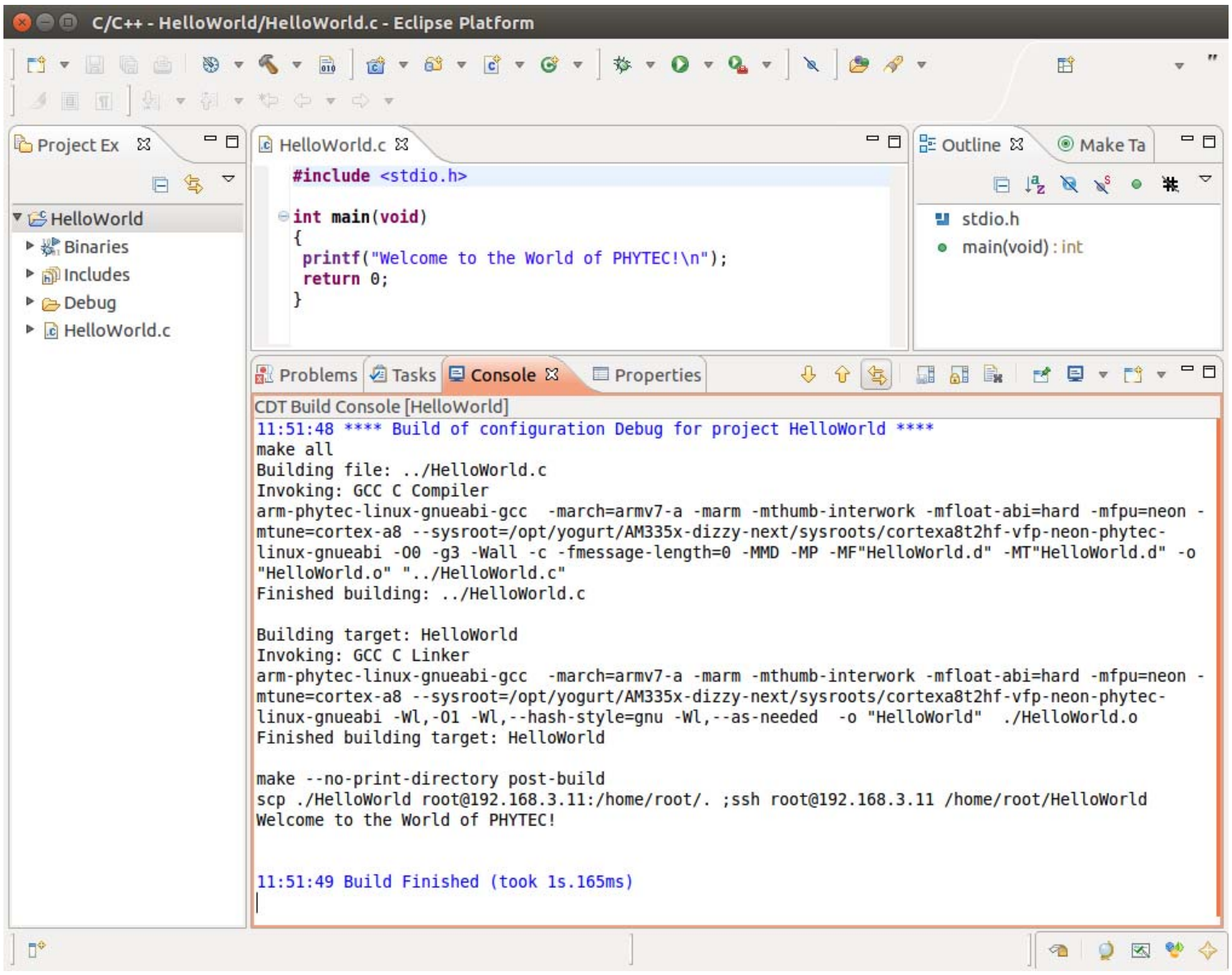
☐ Add project to working sets

Working sets:

- Select ***Project*** ► ***Build Project*** from the menu bar.

The *HelloWorld* program will be compiled and the *HelloWorld* executable is built for the target. Then the *HelloWorld* file is copied to the target using *secure copy*. After the file has been copied to the target, the program is executed on the target using *SSH*.

You will see the following content in the *Console* window:



```
#include <stdio.h>

int main(void)
{
 printf("Welcome to the World of PHYTEC!\n");
 return 0;
}
```

```
CDT Build Console [HelloWorld]
11:51:48 **** Build of configuration Debug for project HelloWorld ****
make all
Building file: ../HelloWorld.c
Invoking: GCC C Compiler
arm-phytec-linux-gnueabi-gcc  -march=armv7-a -marm -mthumb-interwork -mfloat-abi=hard -mfpu=neon -
mtune=cortex-a8 --sysroot=/opt/yogurt/AM335x-dizzy-next/sysroots/cortexa8t2hf-vfp-neon-phytec-
linux-gnueabi -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"HelloWorld.d" -MT"HelloWorld.d" -o
"HelloWorld.o" "../HelloWorld.c"
Finished building: ../HelloWorld.c

Building target: HelloWorld
Invoking: GCC C Linker
arm-phytec-linux-gnueabi-gcc  -march=armv7-a -marm -mthumb-interwork -mfloat-abi=hard -mfpu=neon -
mtune=cortex-a8 --sysroot=/opt/yogurt/AM335x-dizzy-next/sysroots/cortexa8t2hf-vfp-neon-phytec-
linux-gnueabi -Wl,-O1 -Wl,--hash-style=gnu -Wl,--as-needed  -o "HelloWorld"  ./HelloWorld.o
Finished building target: HelloWorld

make --no-print-directory post-build
scp ./HelloWorld root@192.168.3.11:/home/root/. ;ssh root@192.168.3.11 /home/root/HelloWorld
Welcome to the World of PHYTEC!


11:51:49 Build Finished (took 1s.165ms)
```

If you do not get this result verify that you have the target connected to your host, and that the network has been configured as explained in our QuickStart Guide.

You have successfully passed the first steps with the *Eclipse* IDE. You are now able to import existing projects into the *Eclipse* workspace. You can compile an existing project and execute the program on the target.

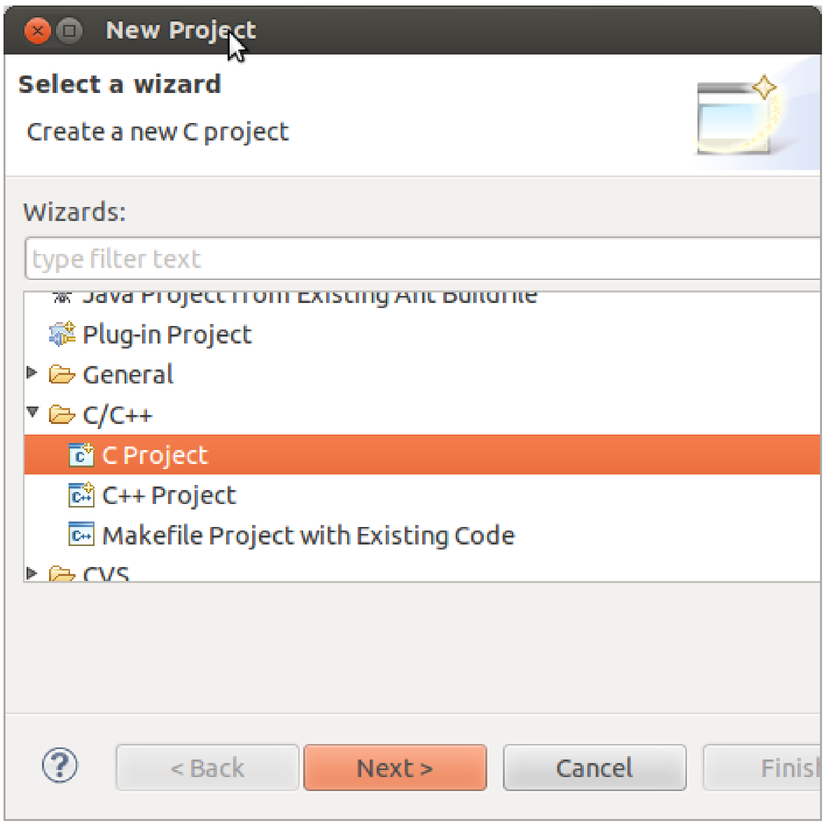
2.2.1.2 Creating a New Project

In this section you will learn how to create a new project with *Eclipse* and how to configure the project for use with the *GNU C/C++* cross development toolchain.

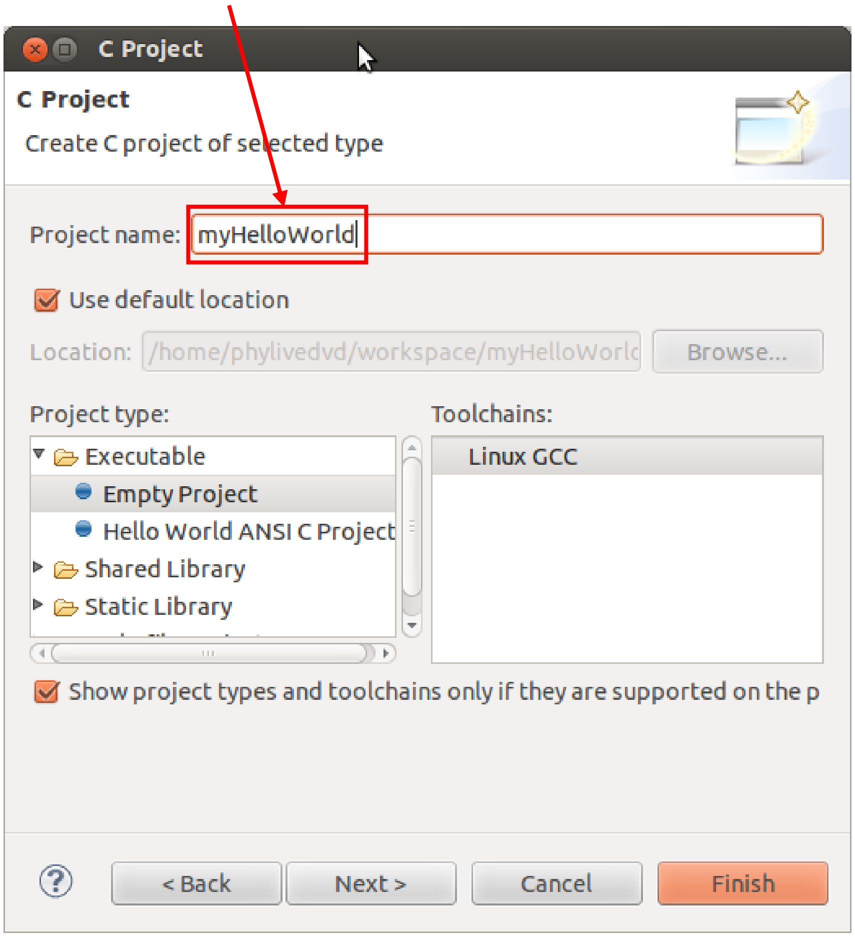
- Open *Eclipse* if it is not already opened.
- Select ***File*** ▶ ***New*** ▶ ***Project*** from the menu bar.

A new dialog opens.

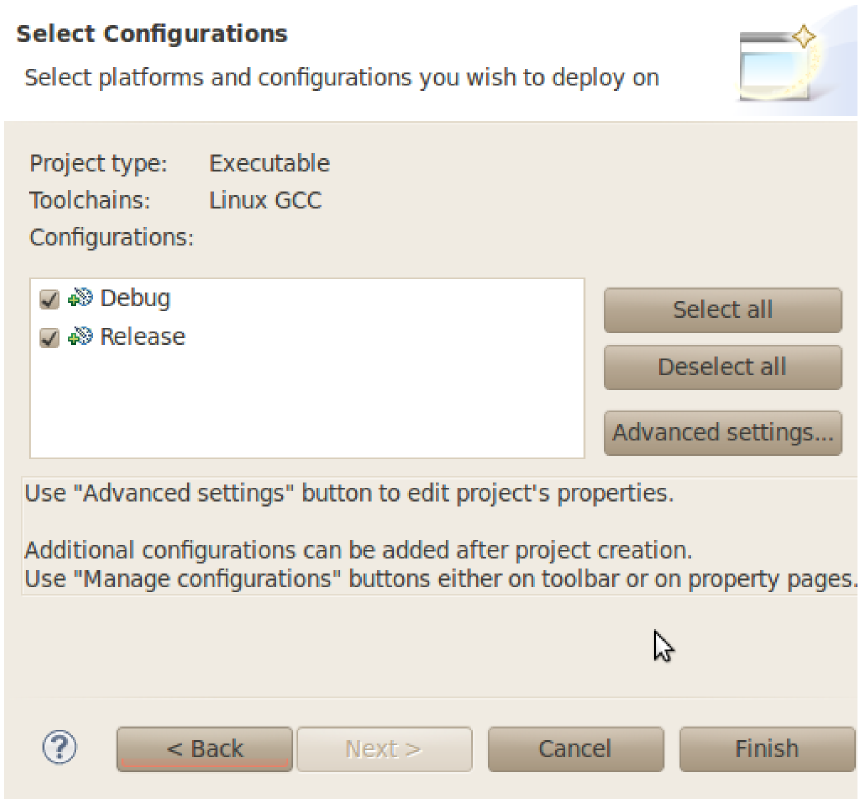
- Select ***C Project*** and click ***Next***.



- Enter the project name `myHelloWorld` and click ***Next***.

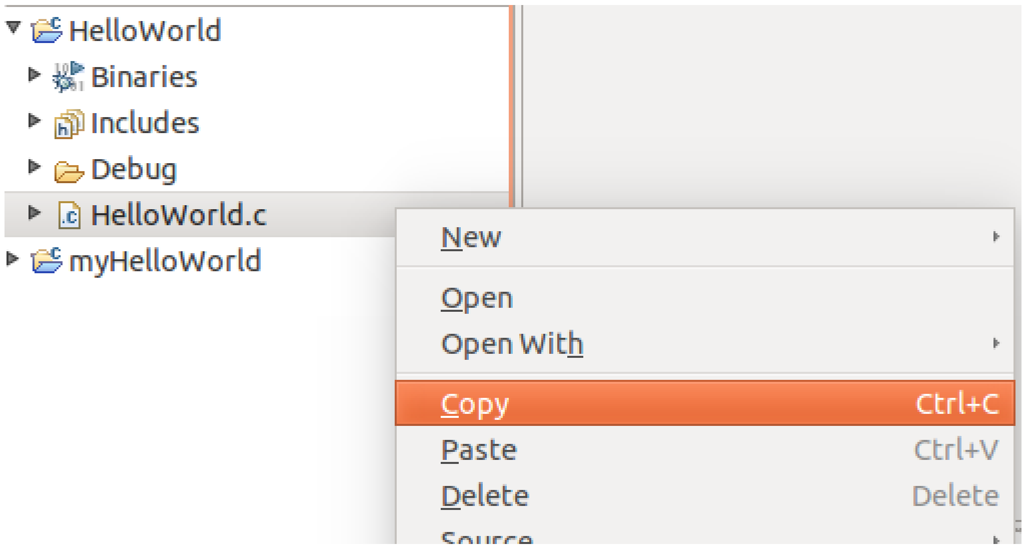

C Project
C Project
Create C project of selected type
Project name: myHelloWorld
Use default location
Location: /home/phylivedvd/workspace/myHelloWorld
Browse...
Project type:
Toolchains:
Executable
Empty Project
Hello World ANSI C Project
Shared Library
Static Library
Linux GCC
Show project types and toolchains only if they are supported on the p
< Back
Next >
Cancel
Finish

- Click ***Finish***.


Select Configurations
Select platforms and configurations you wish to deploy on
Project type: Executable
Toolchains: Linux GCC
Configurations:
Debug
Release
Select all
Deselect all
Advanced settings...
Use "Advanced settings" button to edit project's properties.
Additional configurations can be added after project creation.
Use "Manage configurations" buttons either on toolbar or on property pages.
< Back
Next >
Cancel
Finish

You will see the *C/C++* IDE with the *myHelloWorld* project.

- If the HelloWorld Project is not expanded double-click the ***HelloWorld*** project which we have worked with previously.
- Right-click on ***HelloWorld.c*** in the *HelloWorld* project.
- Select ***Copy***.



- Select the ***myHelloWorld*** project.
- Right-click the ***myHelloWorld*** project.
- Select ***Paste***.
- Double-click on ***HelloWorld.c*** in the *myHelloWorld* project.

If *Build Automatically* from the *Project* menu is selected, the *HelloWorld* application will now be compiled and created with the standard *GCC C/C++* compiler suitable for your host machine. You will find the executable file, which can only run on your host system, in the *workspace/myHelloWorld/Debug* directory.

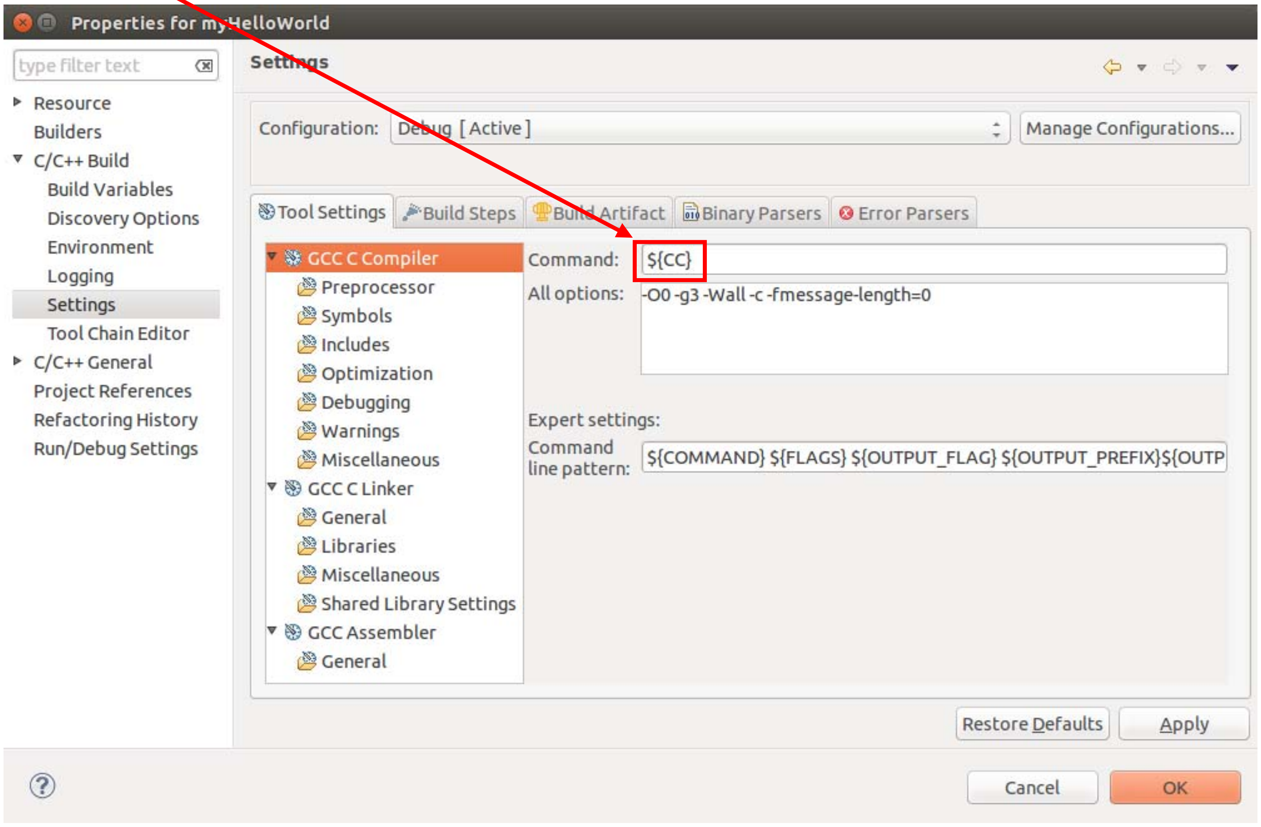
To compile your project for the phBOARD-Mira instead, you will have to use the *GNU C/C++* cross compiler.

- Right-click the ***myHelloWorld*** project and choose ***Properties***.

The *Properties* dialog appears.

- Select ***C/C++ Build ► Settings***.
- Select **GCC C Compiler**.

- Enter `${CC}` into the *Command* input field.



Configuration: Debug [Active]

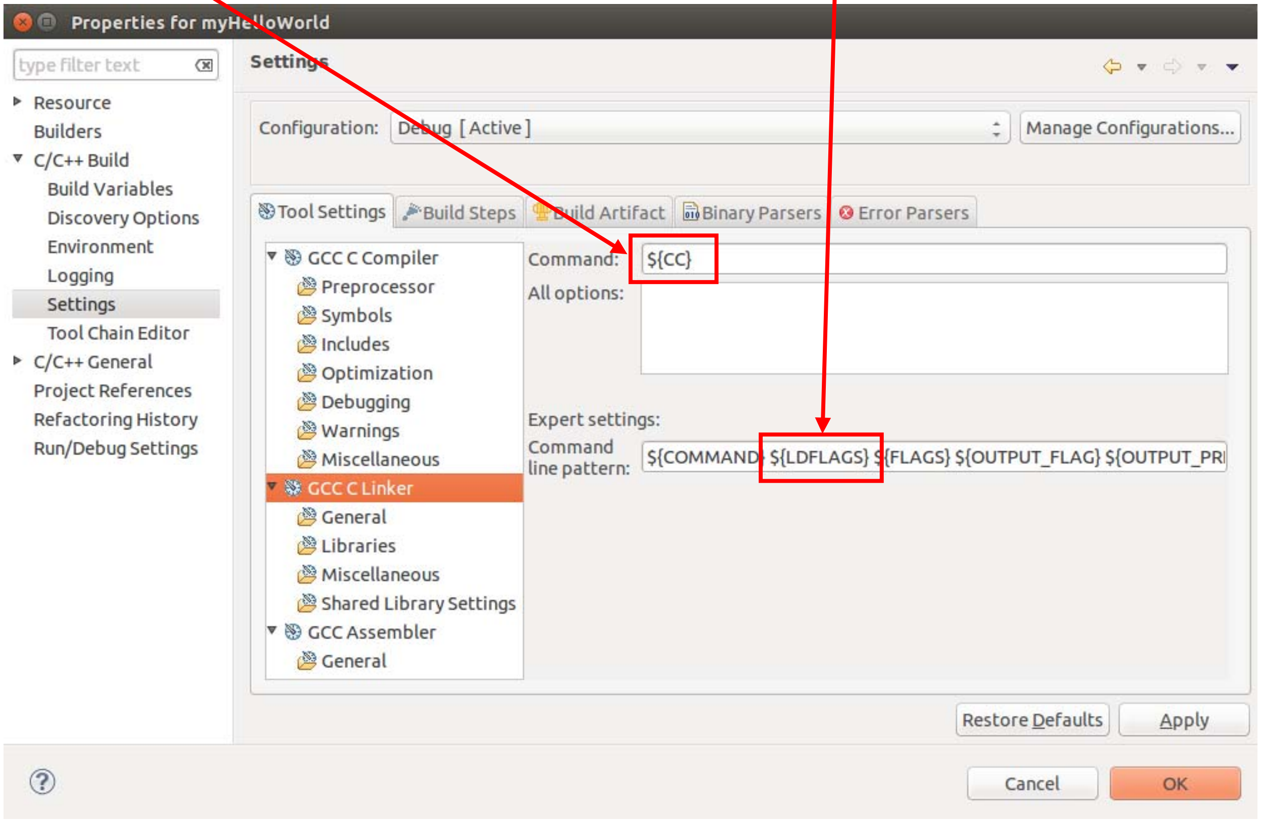
Command: ${CC}

All options: -O0 -g3 -Wall -c -fmessage-length=0

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTP

- Select ***GCC C Linker***.
- Enter `${CC}` into the *Command* input field and add `${LDFLAGS}` in the *Command line pattern* after *${COMMAND}*.



Configuration: Debug [Active]

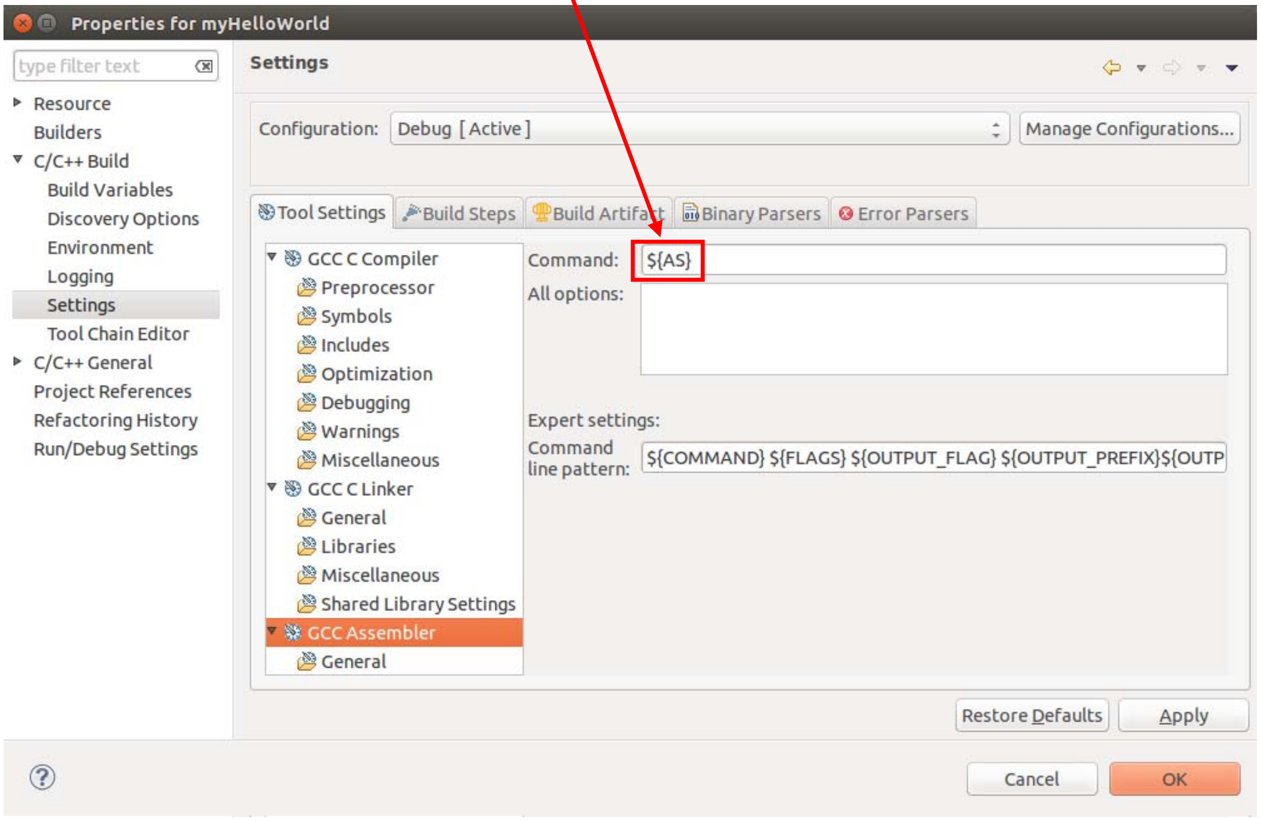
Command: ${CC}

All options:

Expert settings:

Command line pattern: ${COMMAND} ${LDFLAGS} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PRI

- Select ***GCC Assembler***.
- Change the *Command* input field to `${AS}` .



Configuration: Debug [Active]

Command: ${AS}

All options:

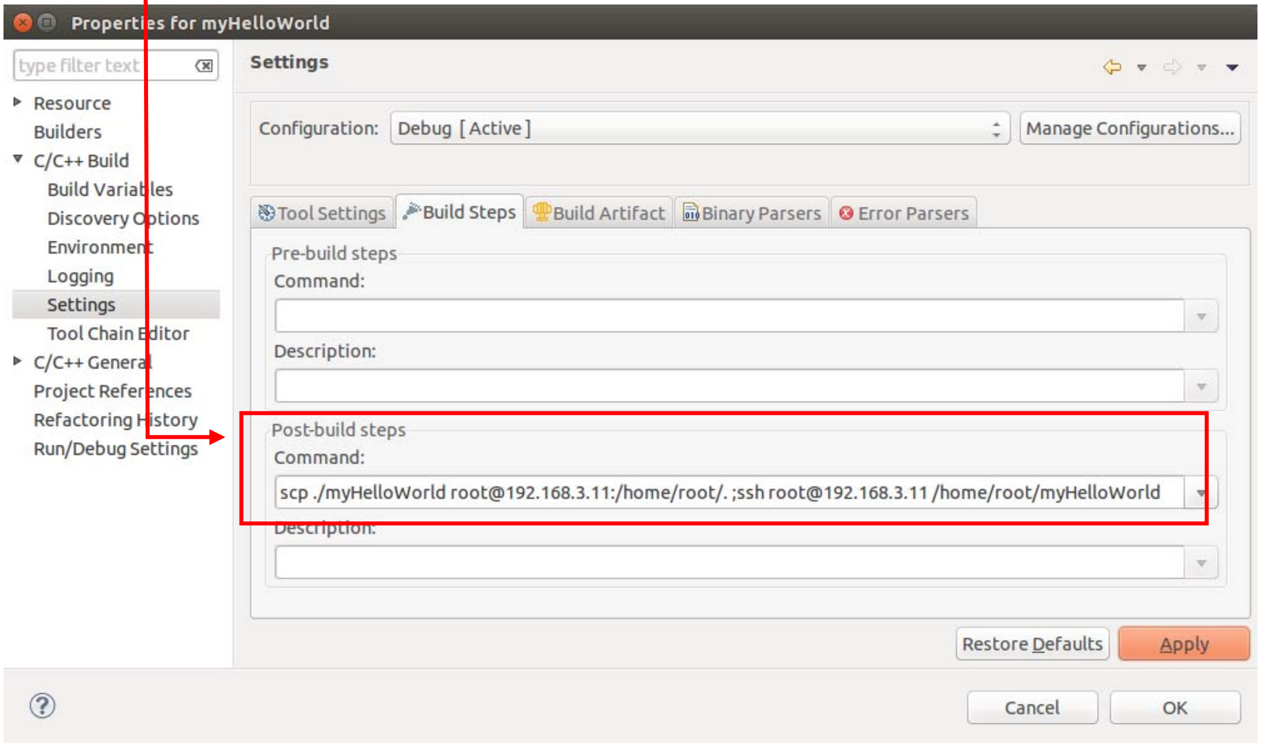
Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTP

- Click ***Apply***.
- Select the *Build Steps* tab.

- Enter the following command in the *Post-build steps Command* input field:

```
scp ./myHelloWorld root@192.168.3.11:/home/root/. ;ssh
root@192.168.3.11 . /home/root/myHelloWorld
```



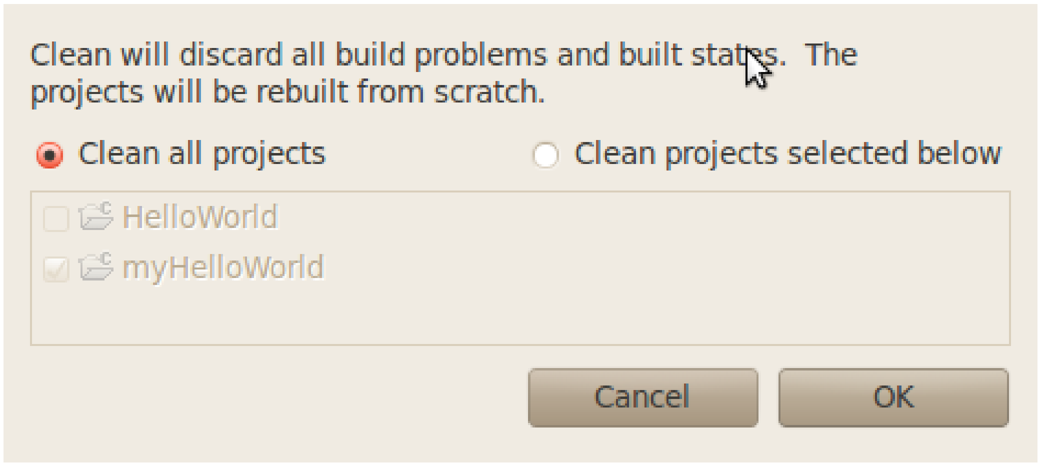
Configuration: Debug [Active]

Pre-build steps
Command:
Description:

Post-build steps
Command: scp ./myHelloWorld root@192.168.3.11:/home/root/. ;ssh root@192.168.3.11 /home/root/myHelloWorld
Description:

⚠	Be sure to enter the **semicolon** before the ssh command. Ensure that the file *myHelloWorld* on the target will have execution rights, because otherwise *ssh* will fail.

- Click ***Apply***.
- Click ***OK***.
- Select ***Project*** ▶ ***Clean*** from the menu bar.


Clean will discard all build problems and built states. The projects will be rebuilt from scratch.

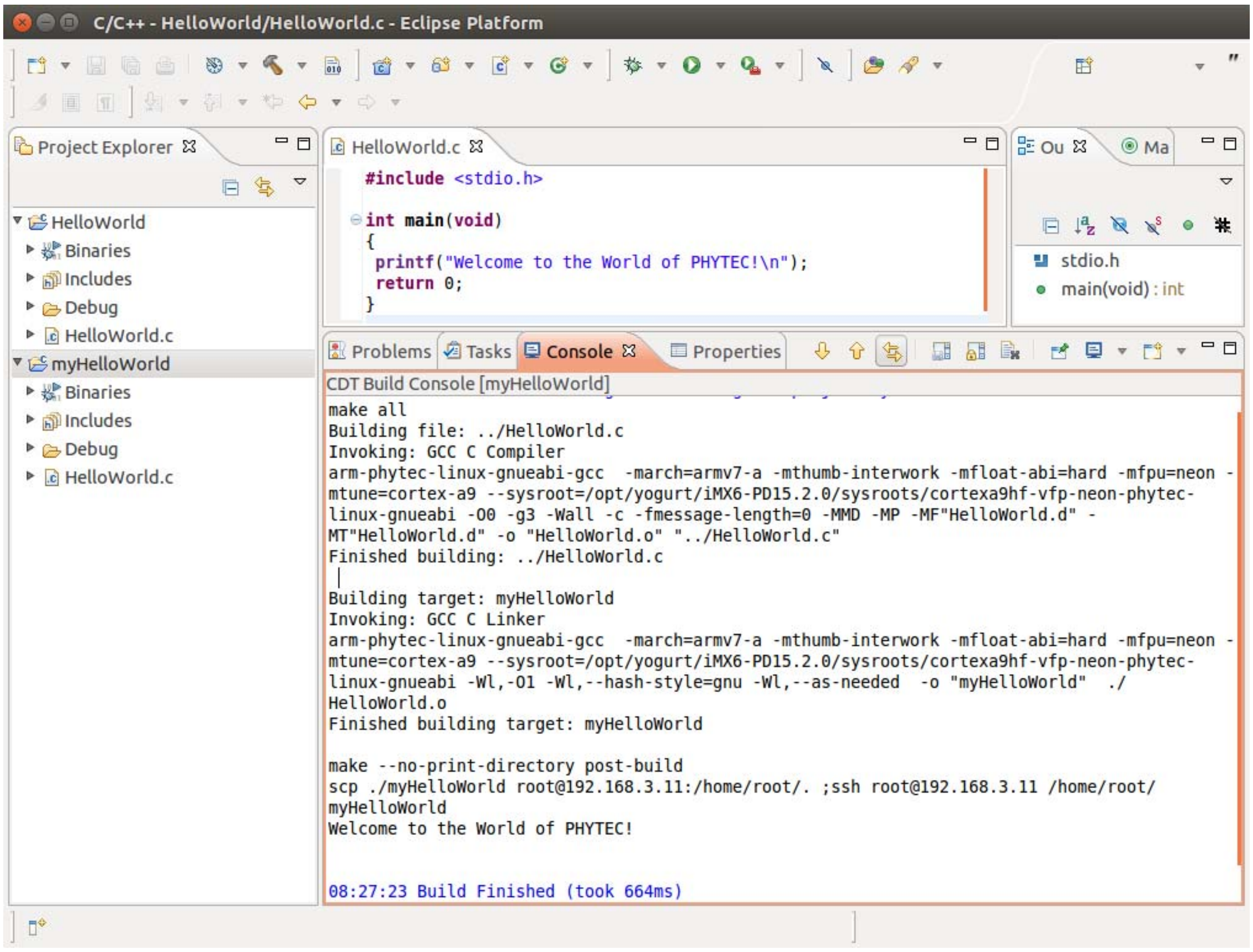
[✓ Clean all projects] Clean projects selected below

- Confirm with ***OK***.

The project will be rebuilt.

- Select the *Console* tab.

If no errors occur while building the project, you will see the following output:



```
#include <stdio.h>

int main(void)
{
 printf("Welcome to the World of PHYTEC!\n");
 return 0;
}
```

CDT Build Console [myHelloWorld]

```
make all
Building file: ../HelloWorld.c
Invoking: GCC C Compiler
arm-phytec-linux-gnueabi-gcc  -march=armv7-a -mthumb-interwork -mfloat-abi=hard -mfpu=neon -
mtune=cortex-a9 --sysroot=/opt/yogurt/iMX6-PD15.2.0/sysroots/cortexa9hf-vfp-neon-phytec-
linux-gnueabi -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"HelloWorld.d" -
MT"HelloWorld.d" -o "HelloWorld.o" "../HelloWorld.c"
Finished building: ../HelloWorld.c

Building target: myHelloWorld
Invoking: GCC C Linker
arm-phytec-linux-gnueabi-gcc  -march=armv7-a -mthumb-interwork -mfloat-abi=hard -mfpu=neon -
mtune=cortex-a9 --sysroot=/opt/yogurt/iMX6-PD15.2.0/sysroots/cortexa9hf-vfp-neon-phytec-
linux-gnueabi -Wl,-O1 -Wl,--hash-style=gnu -Wl,--as-needed  -o "myHelloWorld"  ./
HelloWorld.o
Finished building target: myHelloWorld

make --no-print-directory post-build
scp ./myHelloWorld root@192.168.3.11:/home/root/. ;ssh root@192.168.3.11 /home/root/
myHelloWorld
Welcome to the World of PHYTEC!

08:27:23 Build Finished (took 664ms)
```

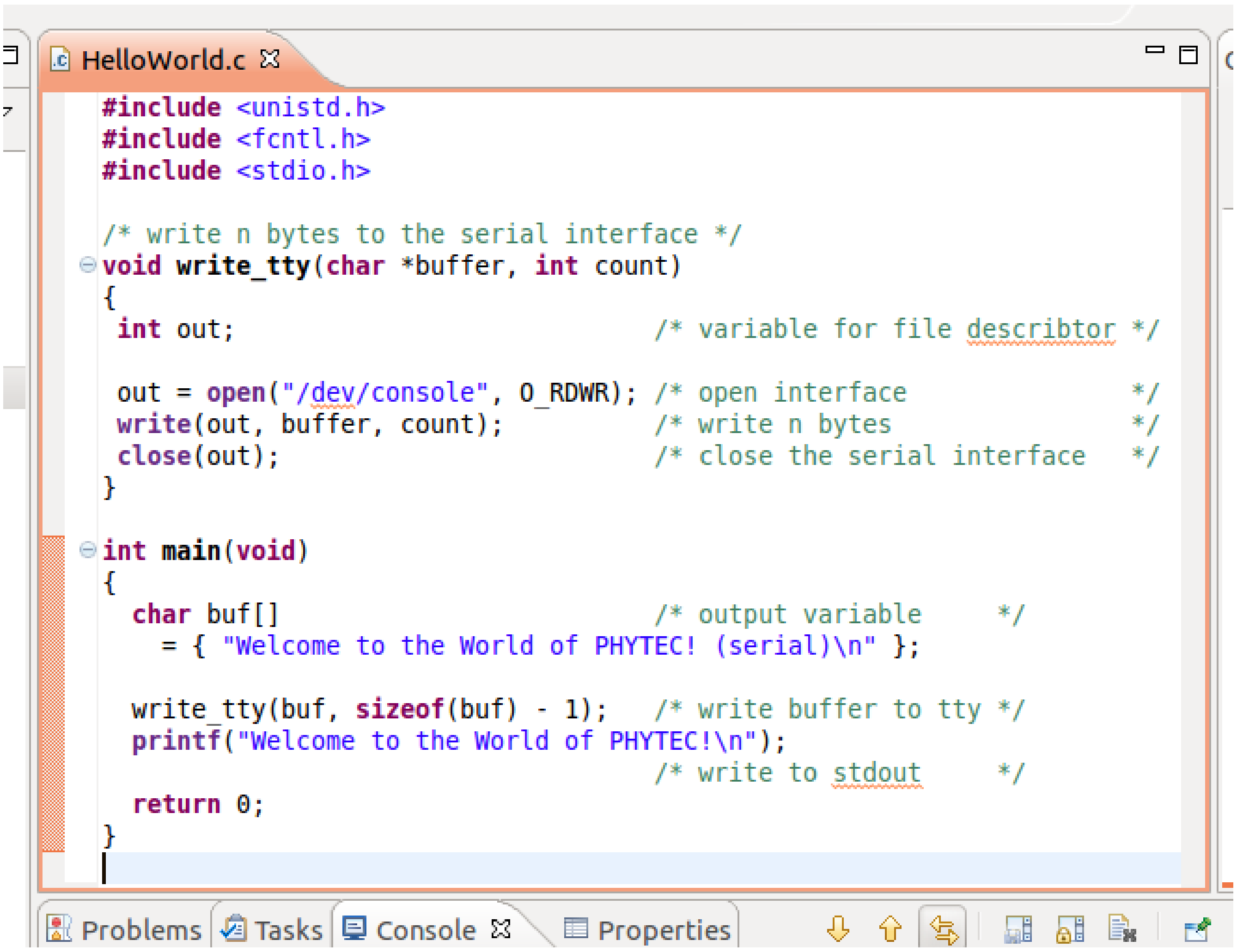
	You have successfully created your first own project with the *Eclipse* IDE. You have configured the project to create an application for your target platform.

2.2.1.3 Modifying the Demo Application

Now we will extend the *myHelloWorld* application. The extended *myHelloWorld* application will write an output to the first serial interface as well as to the standard output.

- Open *Eclipse* if it is not opened yet.
- Double-click ***HelloWorld.c*** in the *myHelloWorld* project.
- First include the following two additional header files:
  ```
  #include <unistd.h>
  #include <fcntl.h>
  ```
- Then add the function *write_tty()*, which writes *n* bytes to the first serial interface (which, on the phyBOARD-Mira i.MX 6, is connected to the system console */dev/console*):
  ```
  void write_tty (char *buffer, int count)
  {
   int out,
   out = open ("/dev/console", O_RDWR),
   write(out, buffer, count),
   close(out),
  }
  ```
- Enter the following two lines in the *main()* function to declare the buffer and call the *write_tty()* function:
  ```
  char buf [] = { "Welcome to the World of PHYTEC! (serial)\n" },
  write_tty(buf, sizeof (buf) - 1),
  ```

In the next screenshot you can see the complete program.



```
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>

/* write n bytes to the serial interface */
void write_tty(char *buffer, int count)
{
 int out;                          /* variable for file describtor */

 out = open("/dev/console", O_RDWR); /* open interface             */
 write(out, buffer, count);        /* write n bytes                */
 close(out);                       /* close the serial interface   */
}

int main(void)
{
  char buf[]                       /* output variable     */
    = { "Welcome to the World of PHYTEC! (serial)\n" };

  write_tty(buf, sizeof(buf) - 1);  /* write buffer to tty */
  printf("Welcome to the World of PHYTEC!\n");
                                   /* write to stdout     */
  return 0;
}
```

- Save your program after changing the code.

The application will be compiled, built, copied to the target and executed.

- Click the ***Microcom icon*** on the desktop.

Microcom

- If you are not logged in, enter `root` and press ***Enter***.
- Type `./myHelloWorld` to start the application.
- You will see the following output:

```
Welcome to the World of PHYTEC! (serial)
Welcome to the World of PHYTEC!
```

- Close *Microcom*.

When you start the application via an *SSH* session, you only see one output line. When you execute the program with *Microcom*, you see two output lines.

The first line is a direct output on the serial interface. You can not see this line in an *SSH* session, because you are connected over a TCP/IP connection to the target. With *Microcom*, however, you have direct access to the serial interface, so you can also see the line that is written to the serial console.

In this section you have changed an existing application. You also learned how to access the serial interface. First you called the function *open()* on the device */dev/console*. The return value of this function was a file descriptor. With the file descriptor you called the function *write()* to send *n* bytes to the device */dev/console*. After that, the file descriptor was closed with the function *close()*.

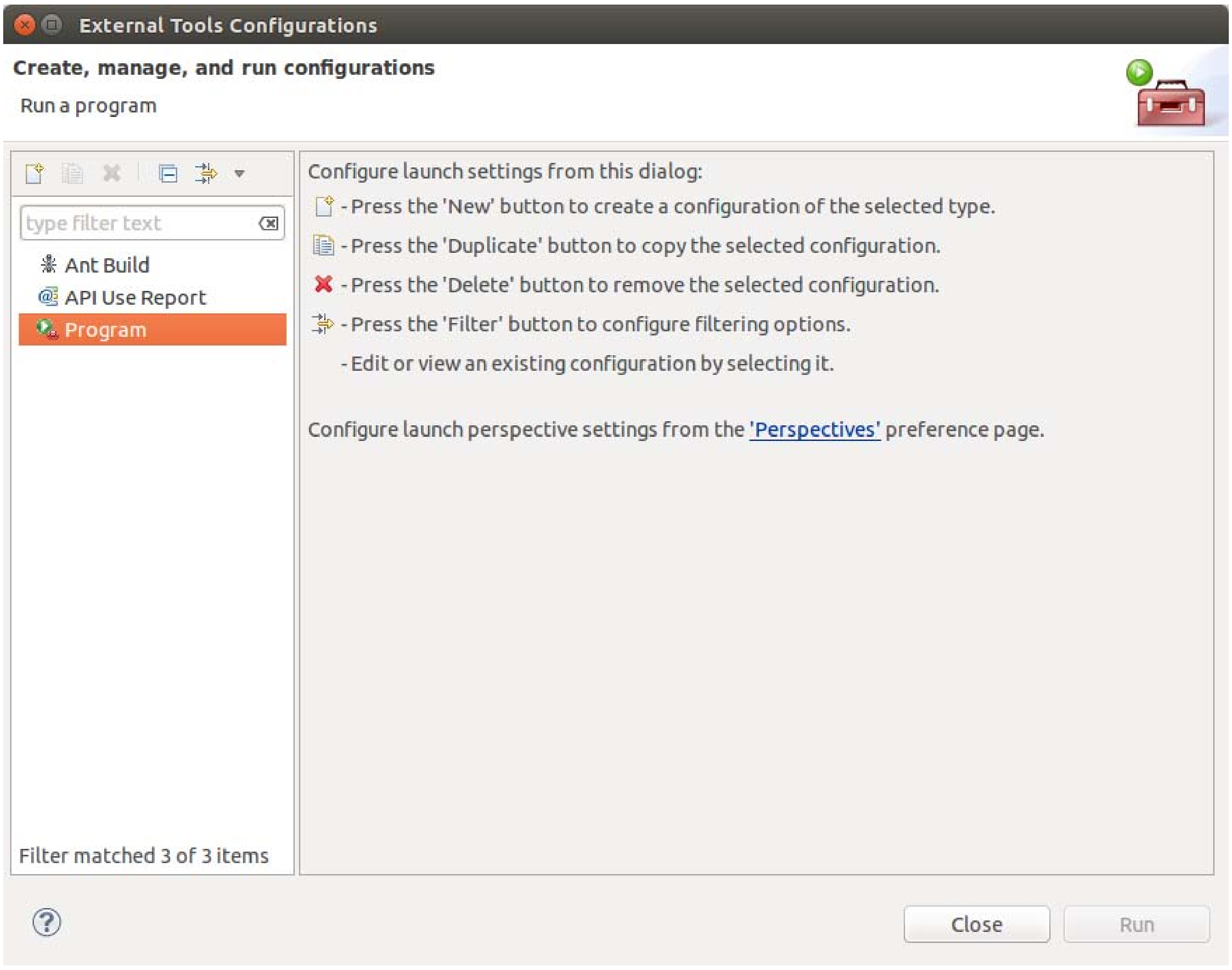
This procedure is quite typical for Linux, because Linux treats everything as a file.

2.2.1.4 Starting a Program out of Eclipse on the Target

In the following you will find another method to start an application out of Eclipse.

After compiling a project in *Eclipse*, the program is copied to the target and directly executed. A program can also be executed on the target without compiling a project. In the following section you will learn how to start a program on the target as an external tool.

- Select ***Run*** ▶ ***External Tools*** ▶ ***External Tools Configurations*** from the menu bar.

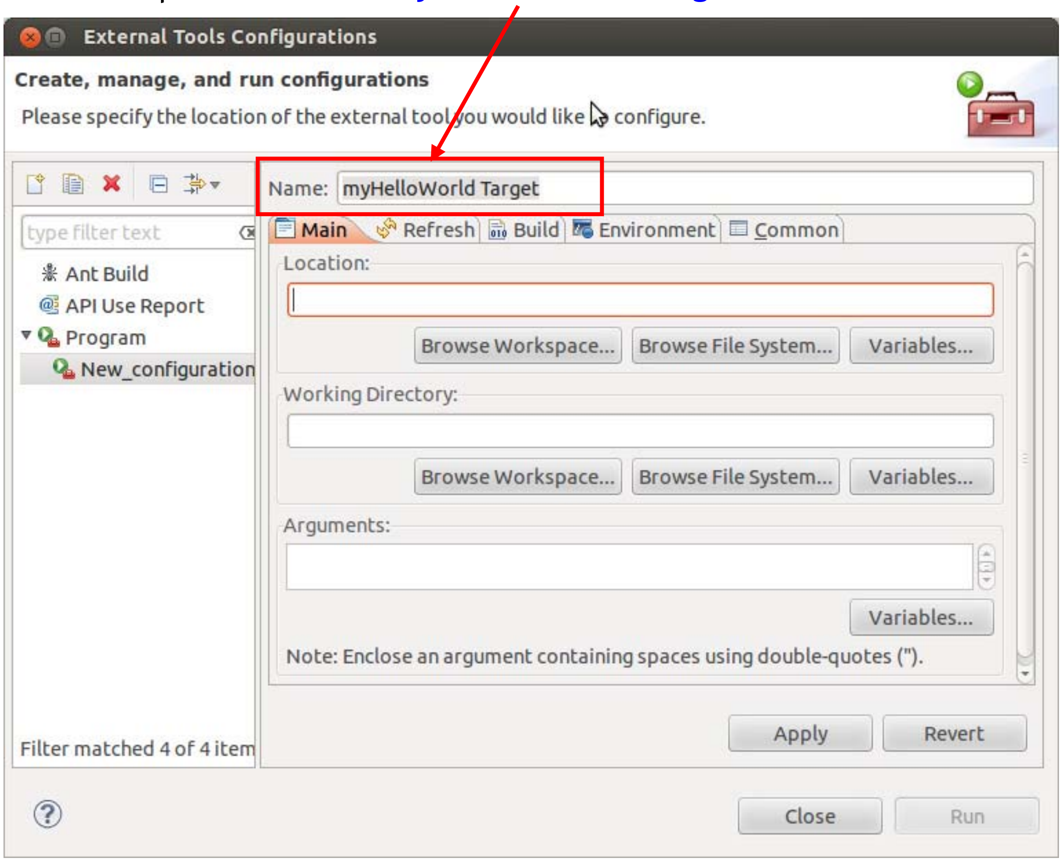

External Tools Configurations
Create, manage, and run configurations
Run a program
type filter text
Ant Build
API Use Report
Program
Filter matched 3 of 3 items
Close
Run

Configure launch settings from this dialog:
- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the 'Perspectives' preference page.

- Double-Click ***Program*** a new program configuration will be opened.

- In the *Name* input field, enter: `myHelloWorld Target`.



External Tools Configurations

Create, manage, and run configurations

Please specify the location of the external tool you would like to configure.

Name: myHelloWorld Target

Location:

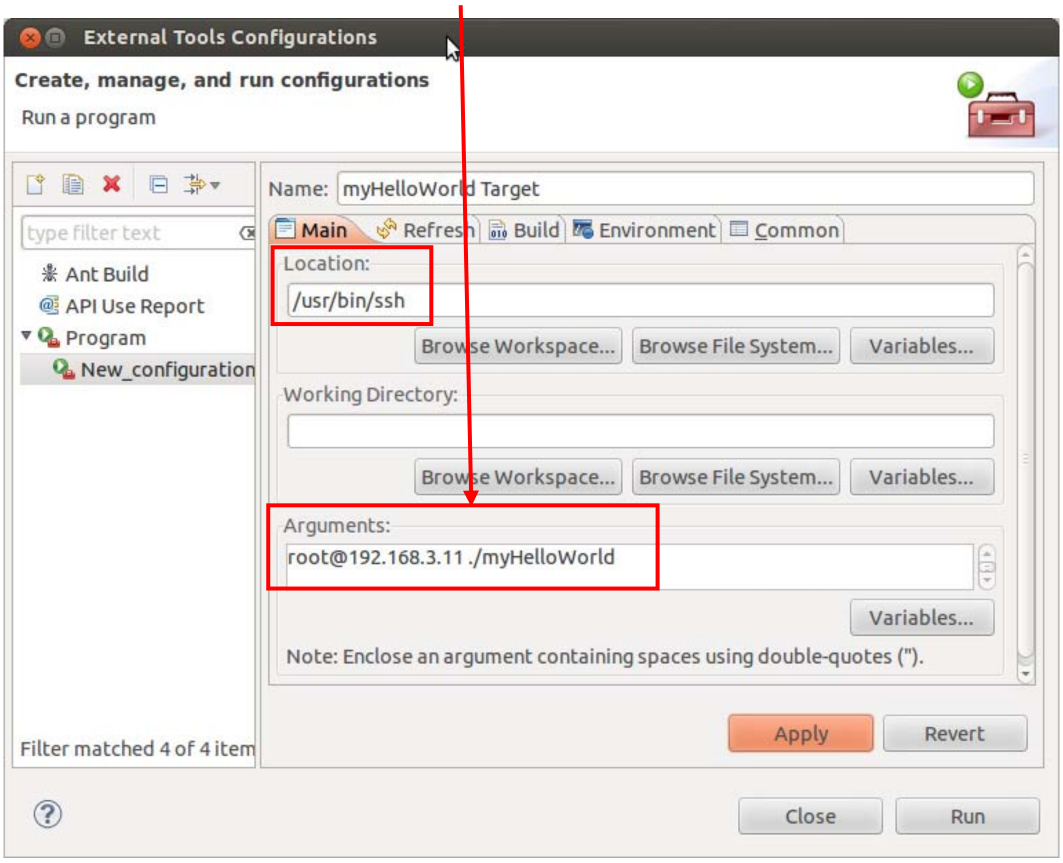
Working Directory:

Arguments:

Note: Enclose an argument containing spaces using double-quotes (").

Filter matched 4 of 4 item

- Enter `/usr/bin/ssh` in the *Location* input field.
- Enter `root@192.168.3.11 ./myHelloWorld` into the *Arguments* field.



External Tools Configurations

Create, manage, and run configurations

Run a program

Name: myHelloWorld Target

Location: /usr/bin/ssh

Working Directory:

Arguments: root@192.168.3.11 ./myHelloWorld

Note: Enclose an argument containing spaces using double-quotes (").

Filter matched 4 of 4 item

- Select ***Apply***.

- Select ***Run***.

Now the program is executed on the target and you will see the output Welcome to the World of PHYTEC! (serial) in the *Microcom* window.

If you want to execute the program the next time, you can use the *Run External Programs* button from the menu bar.

You have successfully created your own *Eclipse* project and you learned how to execute a program on the target.

2.2.2 Debugging an Example Project

In this chapter you will learn using the *GNU debugger GDB* on the host for remote debugging in conjunction with the *GDB* server on the target. *GDB* is the symbolic debugger of the *GNU* project and is arguably the most important debugging tool for any Linux system.

First you will start the *GDB* server on the target. Then you will configure the *Eclipse* platform and start the *GNU* debugger out of *Eclipse* using the *Debug* view.

The *CDT* extends the standard *Eclipse Debug* view with functions for debugging *C/C*++ code. The *Debug* view allows you to manage the debugging and running of a program in the workbench. Using the *Debug* view you will be able to set breakpoints/watchpoints in the code and trace variables and registers. The *Debug* view displays the stack frame for the threads of each target you are debugging. Each thread in your program appears as a node in the tree, and the *Debug* view displays the process for each target you are running.

The *GDB* client is running on the host and is used to control the *GDB* server on the target, which in turn controls the application running on the target. *GDB* client and *GDB* server can communicate over a TCP/IP network connection as well as via a serial interface. In this Application Guide we will only describe debugging via TCP/IP.

2.2.2.1 Starting the GDB Server on the Target

In this passage you will learn how to start the *GDB* server on the target. The *GDB* server will be used to start and control the *myHelloWorld* program.

To debug a program with *GDB*, the program needs extended debugging symbols. These have already been added while building the program.

Microcom

- Open *Microcom*.
- Type `root` and press ***Enter***.
- Start the *GDB* server:
  `gdbserver 192.168.3.11:10000 myHelloWorld`

You have started the *GDB* server on the target. The *GDB* server is now waiting for connections on TCP port 10000.

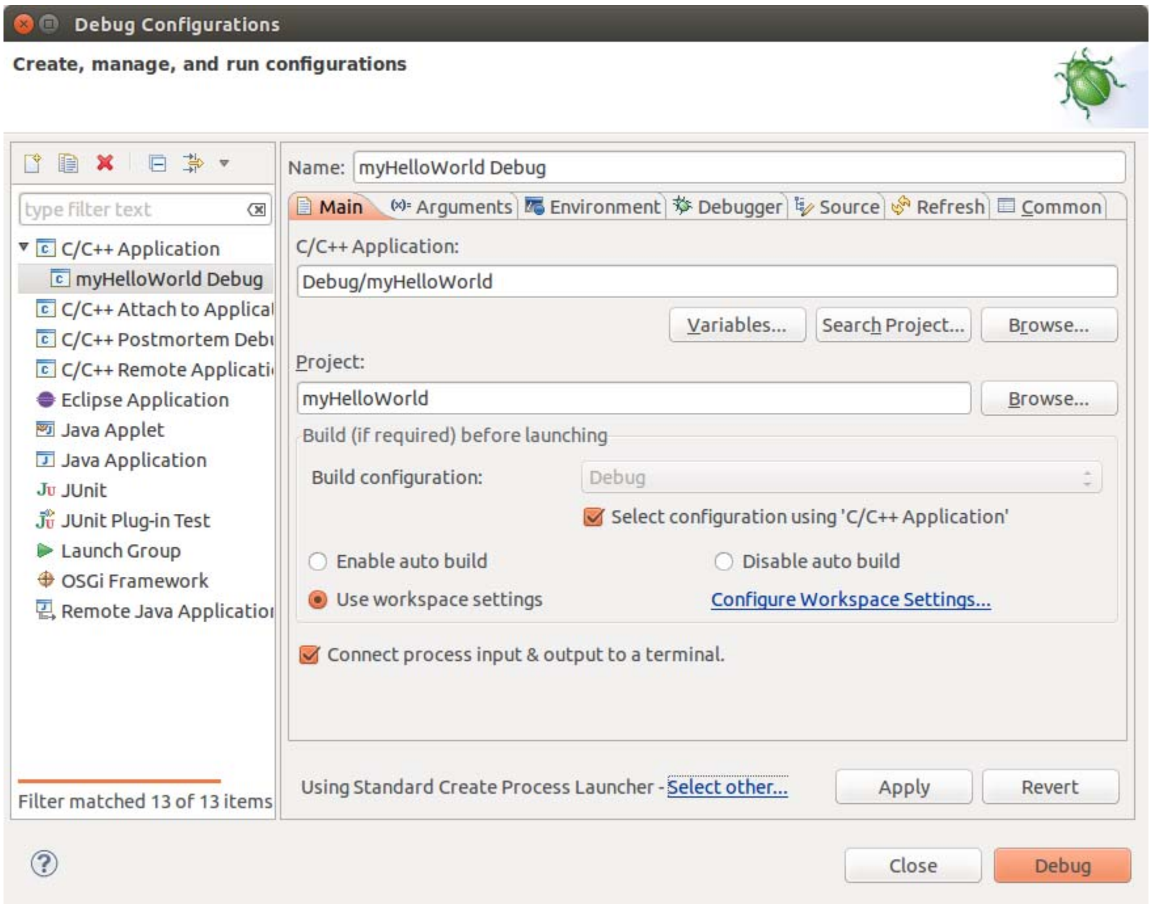
2.2.2.2 Configuring and Starting the Debugger in Eclipse

In this passage you will learn how to configure your project settings to use *Eclipse* with the *GNU* debugger. After the configuration of your project settings, the *GNU* debugger will start and connect to the *GDB* server on the target.

- Start *Eclipse* if the application is not started yet.
- Right-click on the ***myHelloWorld*** project in the *Navigator* window.
- Select ***Debug As*** ▶ ***Debug Configurations***.

A dialog to create, manage and run applications appears.

- Select ***myHelloWorld*** under *C/C++ Application* (to expand it double click on it).



Debug Configurations

Create, manage, and run configurations

Name: myHelloWorld Debug

Main | Arguments | Environment | Debugger | Source | Refresh | Common

C/C++ Application: Debug/myHelloWorld

Project: myHelloWorld

Build (if required) before launching

Build configuration: Debug

Select configuration using 'C/C++ Application'

Enable auto build | Disable auto build

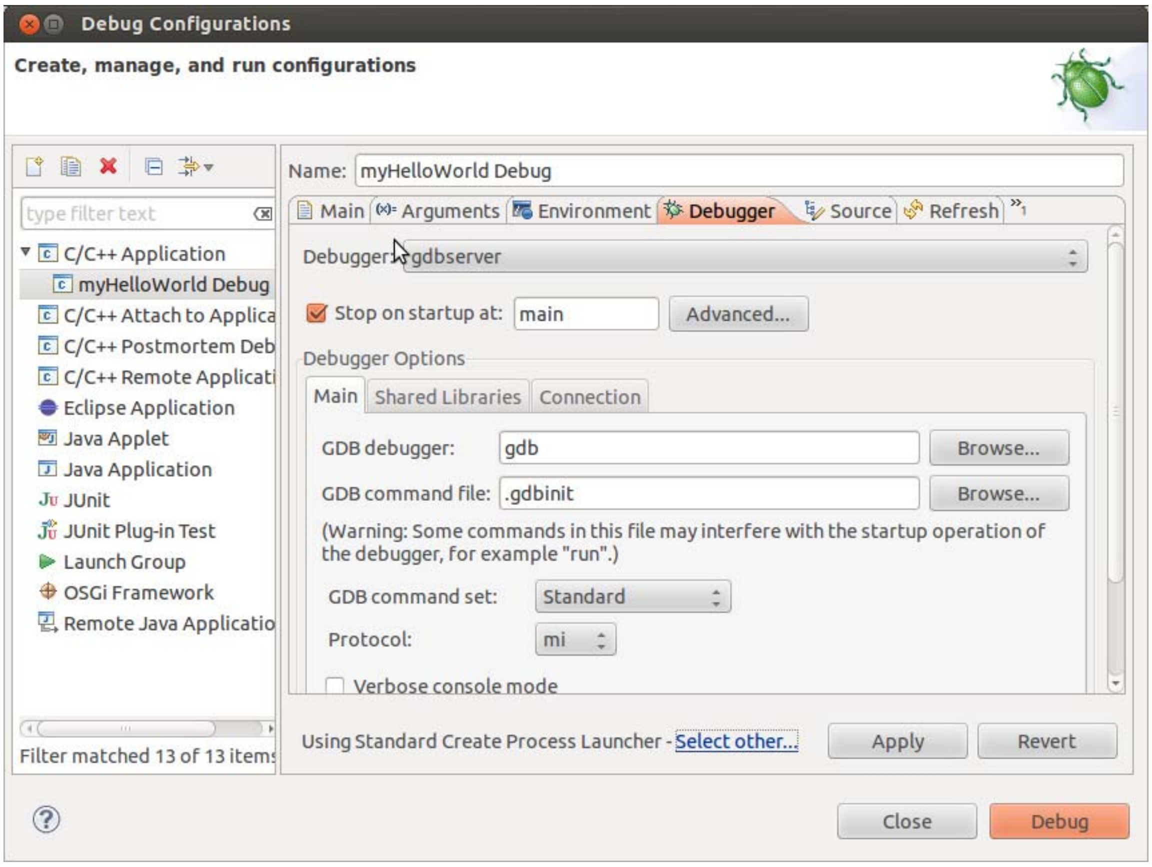
Use workspace settings | Configure Workspace Settings...

Connect process input & output to a terminal.

Filter matched 13 of 13 items

Using Standard Create Process Launcher - Select other...

- Select the ***Debugger*** tab.



Debug Configurations

Create, manage, and run configurations

Name: myHelloWorld Debug

Main | Arguments | Environment | Debugger | Source | Refresh

Debugger: gdbserver

Stop on startup at: main

Debugger Options

Main | Shared Libraries | Connection

GDB debugger: gdb

GDB command file: .gdbinit

(Warning: Some commands in this file may interfere with the startup operation of the debugger, for example "run".)

GDB command set: Standard

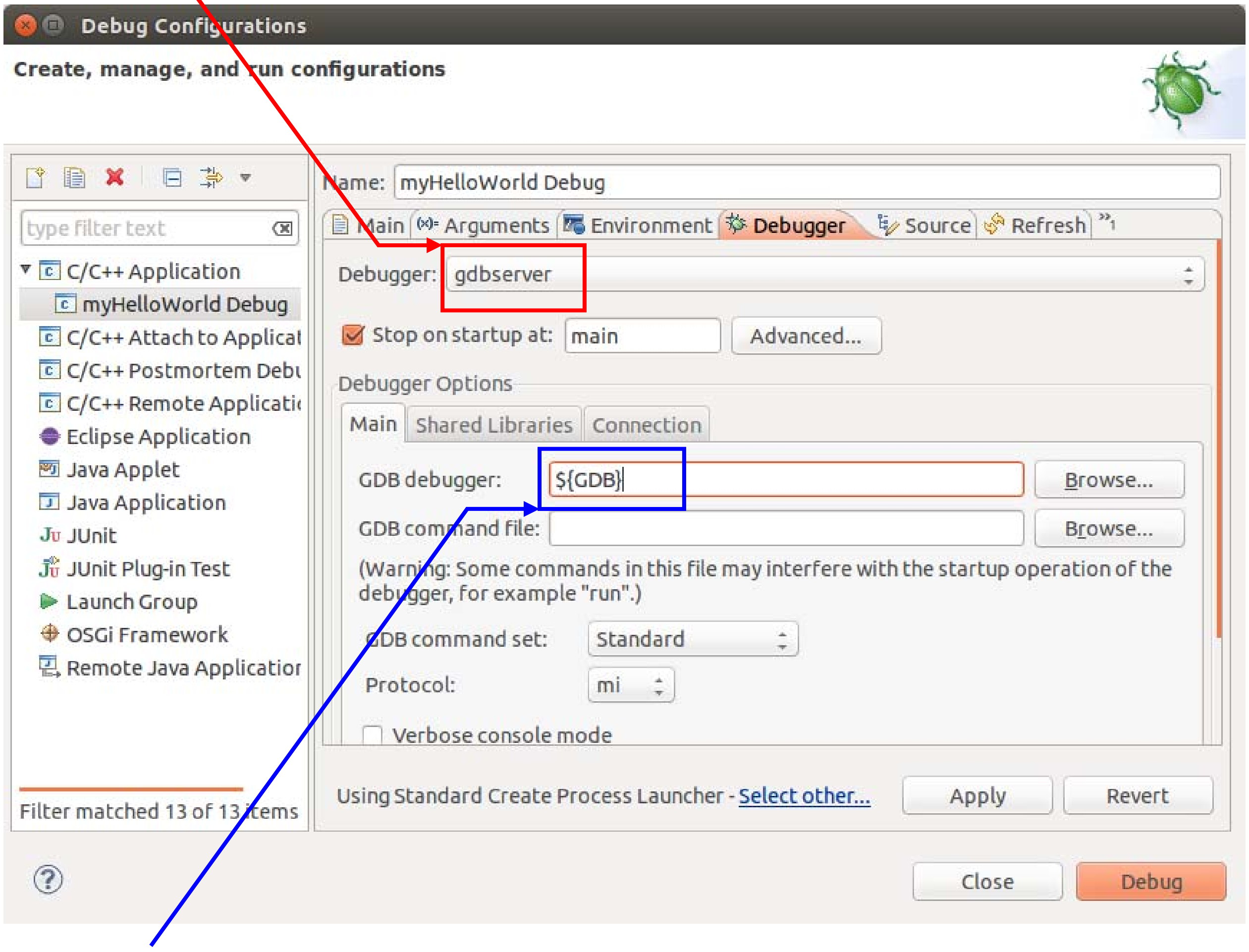
Protocol: mi

Verbose console mode

Filter matched 13 of 13 items

Using Standard Create Process Launcher - Select other...

- Select ***gdbserver*** *Debugger* from the *Debugger* drop-down box.



Debug Configurations

Create, manage, and run configurations

Name: myHelloWorld Debug

Main | Arguments | Environment | Debugger | Source | Refresh

Debugger: gdbserver

☑ Stop on startup at: main

Debugger Options

Main | Shared Libraries | Connection

GDB debugger: ${GDB}

GDB command file:

(Warning: Some commands in this file may interfere with the startup operation of the debugger, for example "run".)

GDB command set: Standard

Protocol: mi

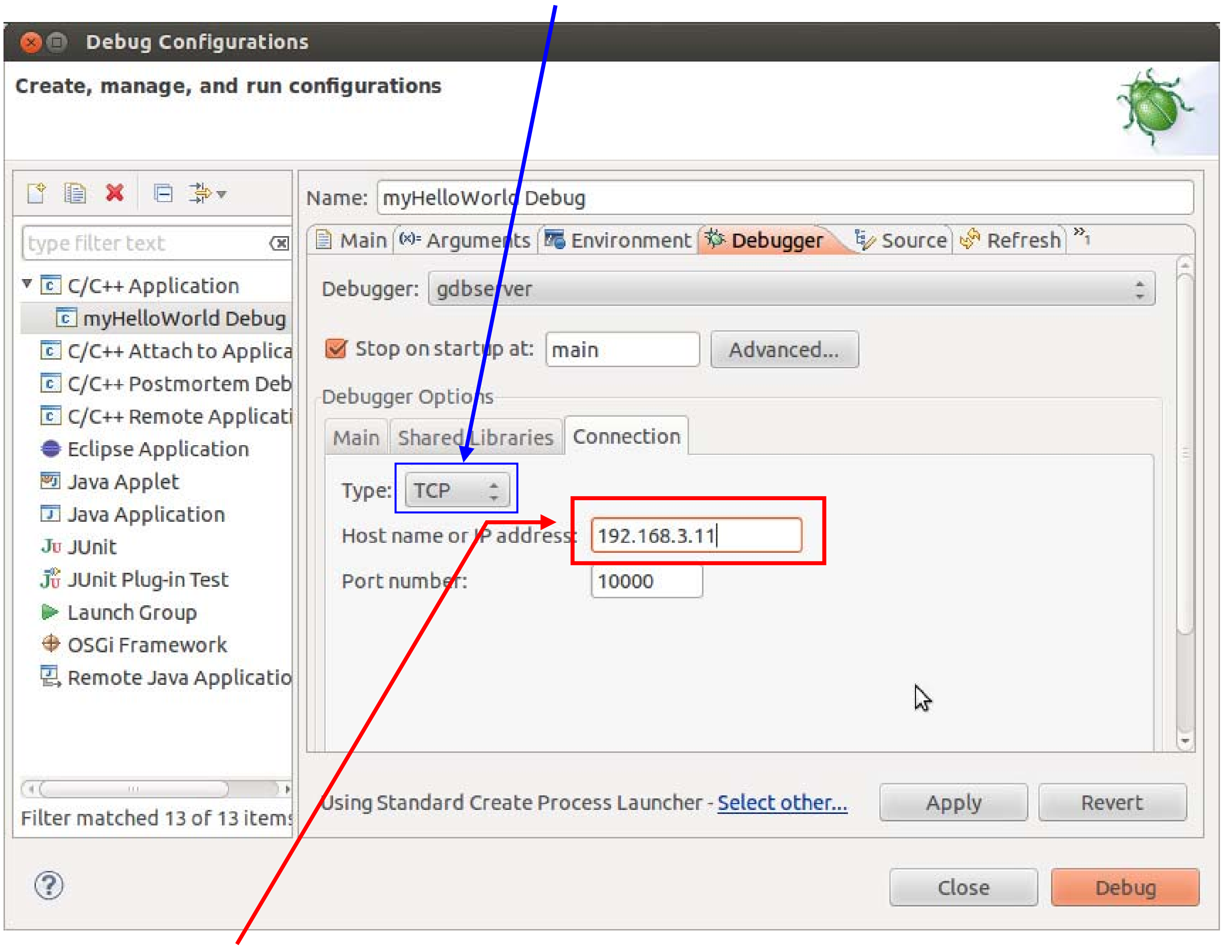
☐ Verbose console mode

Using Standard Create Process Launcher - Select other...

Filter matched 13 of 13 items

- Enter ${GDB} in the *GDB Debugger* field
- Keep the *GDB command file* field empty.

- Select the ***Connection*** tab and select ***TCP*** in the drop-down box.


Debug Configurations

Create, manage, and run configurations

Name: myHelloWorld Debug

Debugger: gdbserver

Stop on startup at: main

Type: TCP

Host name or IP address: 192.168.3.11

Port number: 10000

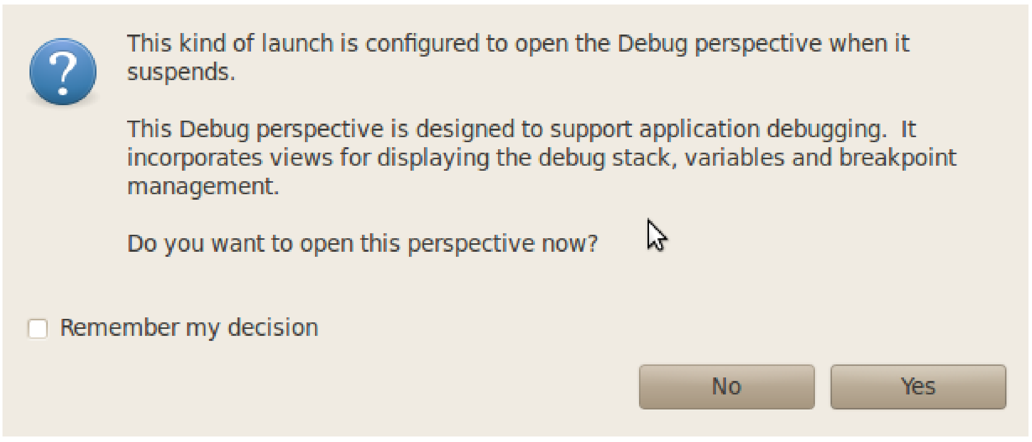
- Enter `192.168.3.11` (the target's IP address) in the *Host name* input field.

The host's *GDB* will connect to this IP address to communicate with the target's *GDB* server.

- Click ***Apply***.
- Click ***Debug***.

A new dialog appears.

- •Select ***Yes*** to switch to the *Debug* perspective.

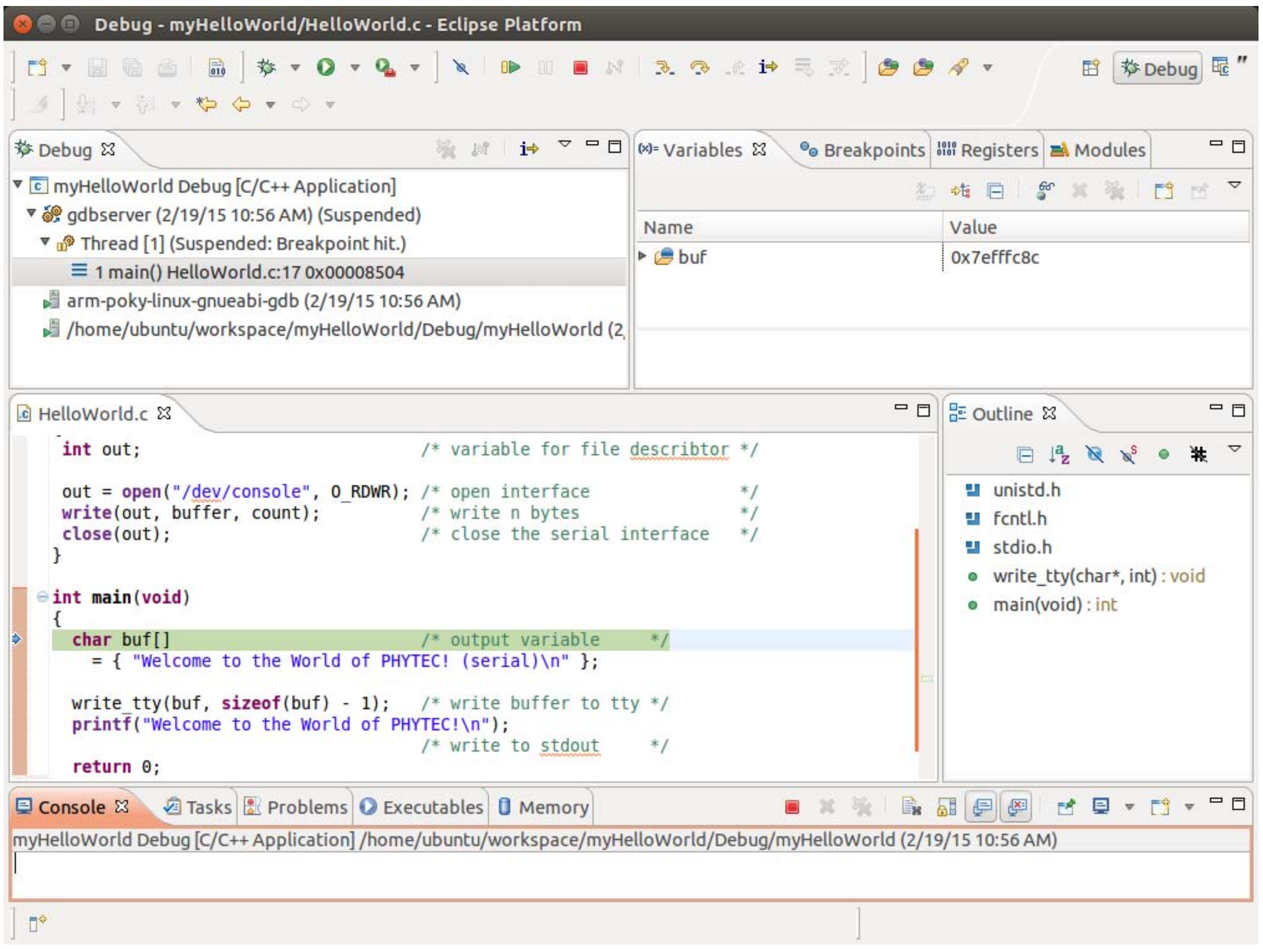

This kind of launch is configured to open the Debug perspective when it suspends.

This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

Do you want to open this perspective now?

Remember my decision

The debug perspective opens and the debugger stops automatically at the first line. The host's *GDB* is now connected to the *GDB* server on the target.


Debug - myHelloWorld/HelloWorld.c - Eclipse Platform

Debug

```
myHelloWorld Debug [C/C++ Application]
  gdbserver (2/19/15 10:56 AM) (Suspended)
    Thread [1] (Suspended: Breakpoint hit.)
      1 main() HelloWorld.c:17 0x00008504
  arm-poky-linux-gnueabi-gdb (2/19/15 10:56 AM)
  /home/ubuntu/workspace/myHelloWorld/Debug/myHelloWorld (2,
```

Variables | Breakpoints | Registers | Modules

Name	Value
buf	0x7efffc8c

HelloWorld.c

```
    int out;                        /* variable for file describtor */

    out = open("/dev/console", O_RDWR); /* open interface           */
    write(out, buffer, count);          /* write n bytes            */
    close(out);                         /* close the serial interface */
}

int main(void)
{
    char buf[]                      /* output variable       */
        = { "Welcome to the World of PHYTEC! (serial)\n" };

    write_tty(buf, sizeof(buf) - 1);  /* write buffer to tty */
    printf("Welcome to the World of PHYTEC!\n");
                                      /* write to stdout      */
    return 0;
```

Outline

```
unistd.h
fcntl.h
stdio.h
write_tty(char*, int) : void
main(void) : int
```

Console | Tasks | Problems | Executables | Memory

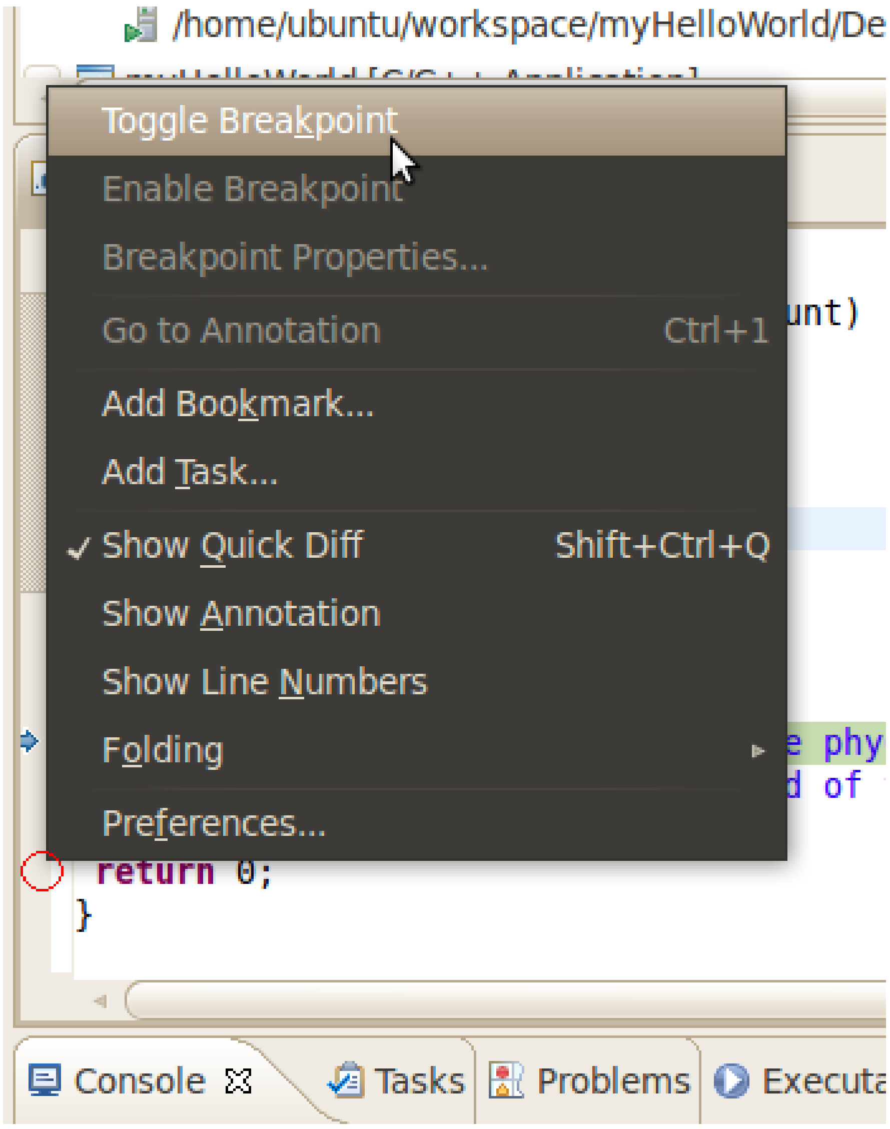
```
myHelloWorld Debug [C/C++ Application] /home/ubuntu/workspace/myHelloWorld/Debug/myHelloWorld (2/19/15 10:56 AM)
```

You have configured your project for remote debugging. You have started the *GNU* debugger in *Eclipse* and connected the host's *GDB* with the target's *GDB* server. You can now start to debug the project.

2.2.2.3 Setting a Breakpoint

Now you will set a breakpoint in your program. The breakpoint will be set on the last line of the function *main()*. If you resume the application, the debugger will stop at this line.

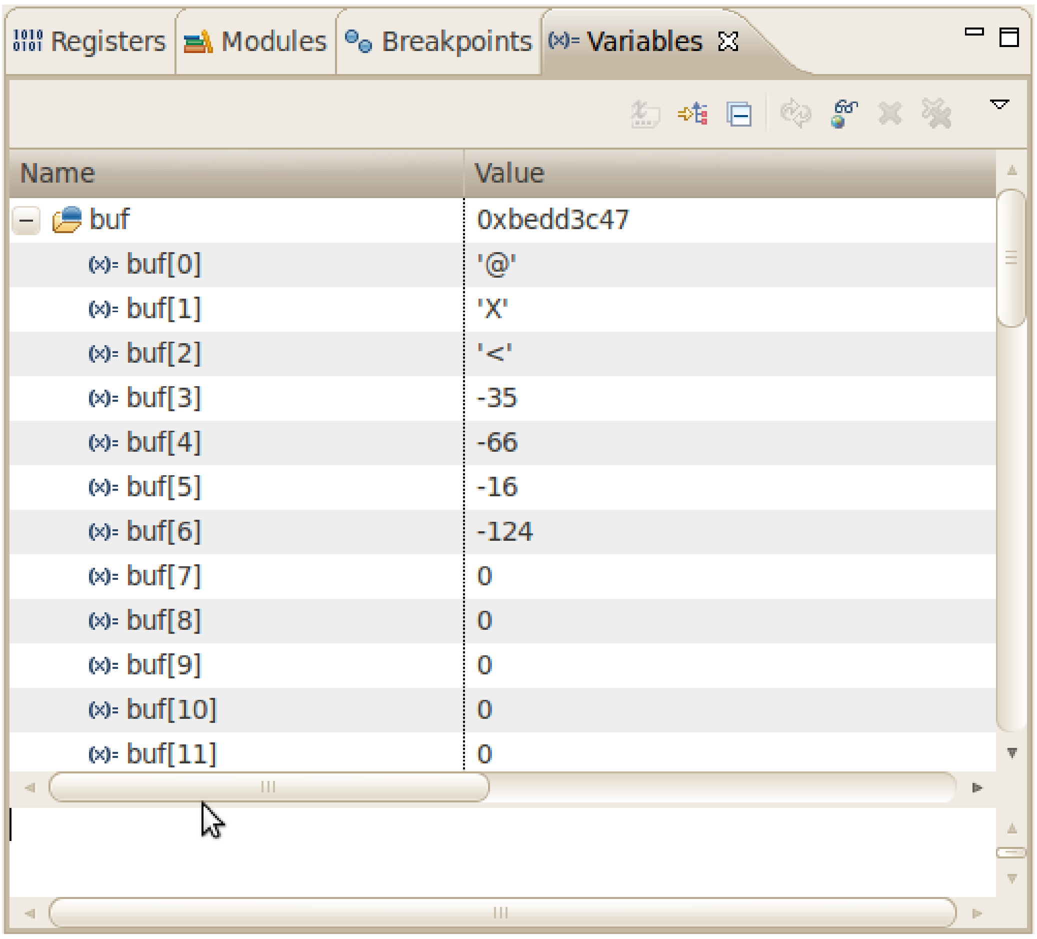
- Select the last line in *main()*.
- Right-click into the small grey border on the left-hand side and select ***Toggle Breakpoint*** to set a new breakpoint.



2.2.2.4 Stepping through and Watching Variable Contents

In this part you will step through the example project with the debugger. You will also learn how to check the content of a variable.

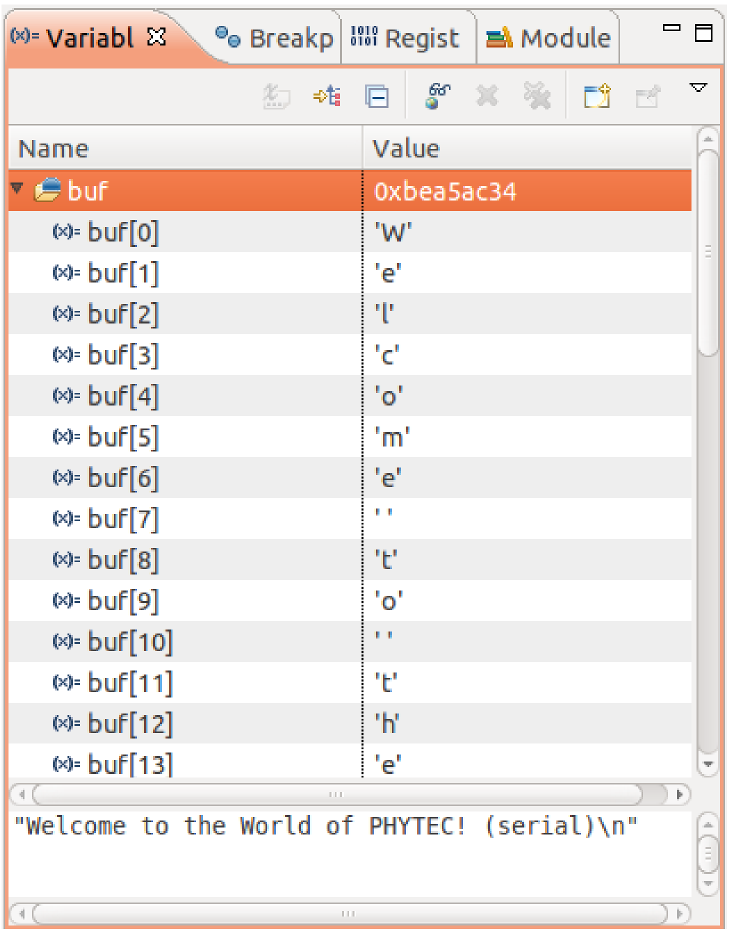
- Expand ***buf*** in the *Variables* window.



Name	Value
buf	0xbedd3c47
buf[0]	'@'
buf[1]	'X'
buf[2]	'<'
buf[3]	-35
buf[4]	-66
buf[5]	-16
buf[6]	-124
buf[7]	0
buf[8]	0
buf[9]	0
buf[10]	0
buf[11]	0

- Click the ***Step Over*** button in the *Debug* window to step to the next line. You will see the content of the *buf* variable in the *Variables* window.

- Click on the variable ***buf***.



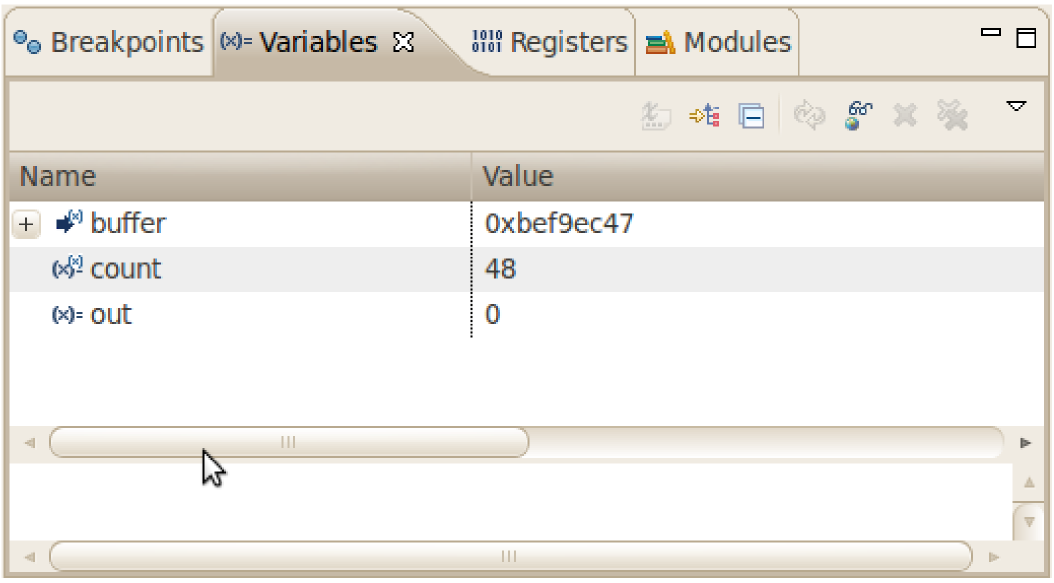
Name	Value
buf	0xbea5ac34
buf[0]	'W'
buf[1]	'e'
buf[2]	'l'
buf[3]	'c'
buf[4]	'o'
buf[5]	'm'
buf[6]	'e'
buf[7]	' '
buf[8]	't'
buf[9]	'o'
buf[10]	' '
buf[11]	't'
buf[12]	'h'
buf[13]	'e'

```
"Welcome to the World of PHYTEC! (serial)\n"
```

- Then click the button ***Step into*** to enter the function *write_tty()*.

- The debugger stops in *write_tty()*.

You will see the following variable window:



Name	Value
buffer	0xbef9ec47
count	48
out	0

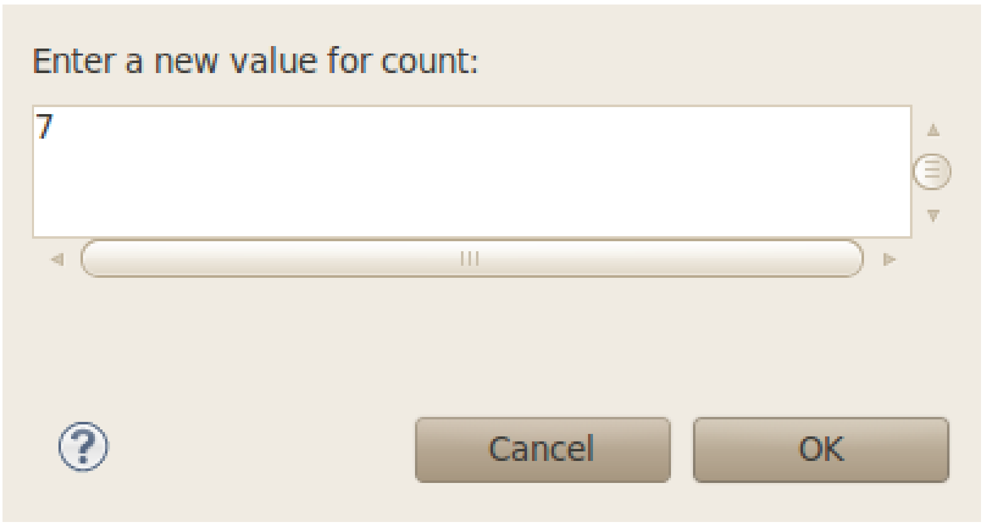
- Click on the variable ***buffer***.

You will probably see a different address on the buffer pointer. Remember which address is shown in your case, you will need this address later.

2.2.2.5 Stepping through and Changing Variable Contents

In this section you will change the value of a variable. At the end of this part you will see the effect of this change.

- Select the ***count*** variable in the *Variables* window.
- Right-click on ***count*** and select ***Change Value***.
- Change the value of count to 7 and click ***OK***.

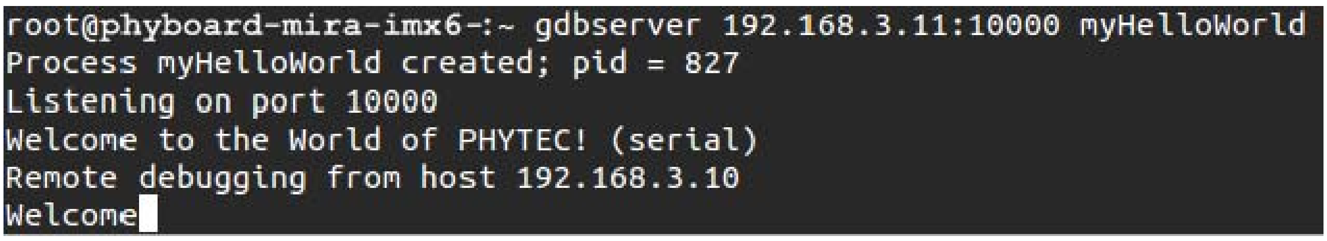


Enter a new value for count:

7

- Open *Microcom* if the application is not already opened.
- Go back to *Eclipse*.
- Click the ***Step Over*** button **twice**.

- Switch to *Microcom*.



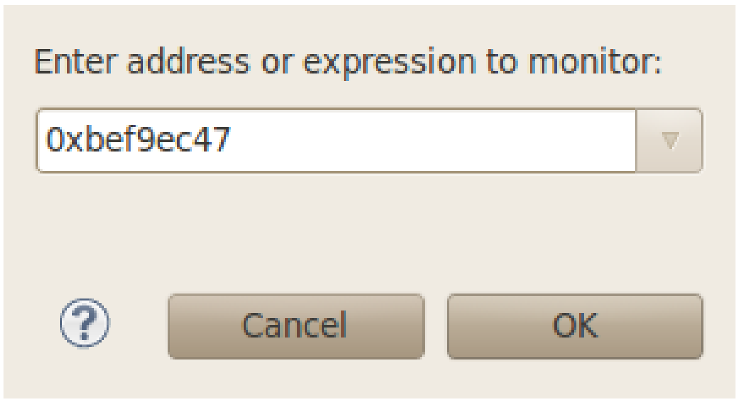
```
root@phyboard-mira-imx6-:~ gdbserver 192.168.3.11:10000 myHelloWorld
Process myHelloWorld created; pid = 827
Listening on port 10000
Welcome to the World of PHYTEC! (serial)
Remote debugging from host 192.168.3.10
Welcome
```

Because we changed the *count* variable to 7 only the first seven characters (*Welcome)* are displayed in the *Microcom* console.

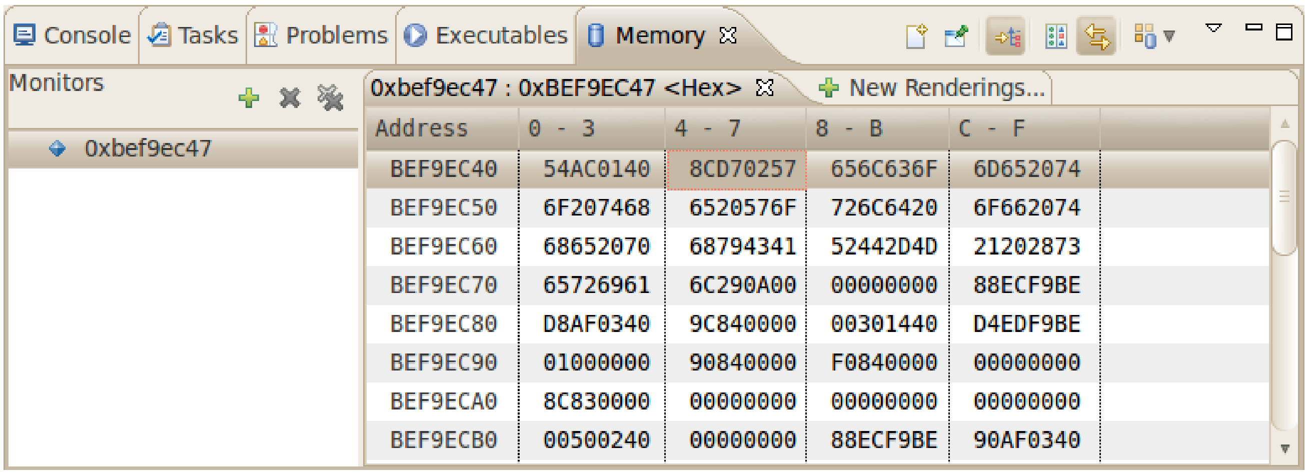
2.2.2.6 Using the Memory Monitor

In the last section of this chapter you will use the memory monitor to control the content at a memory address.

- Select the ***Memory*** tab in the frame where you can find the *Console*.
- Click ***+ Add Memory Monitor***.
- Enter the address of the buffer and click ***OK***. Remember that the variable's address might be different on your system.

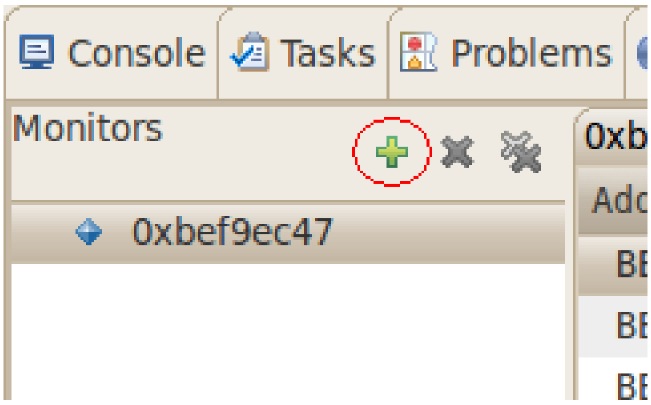


- Change the size of the window.

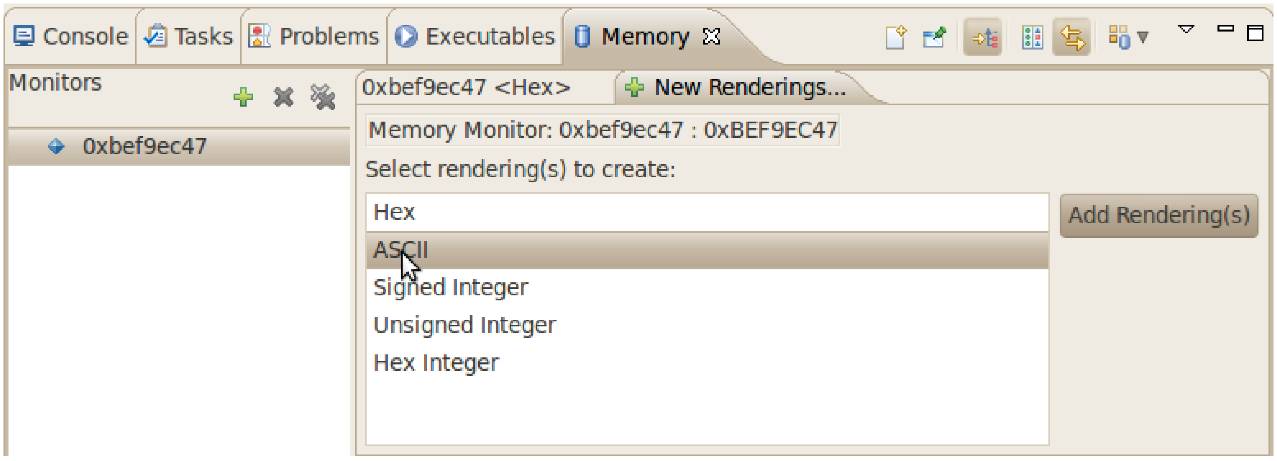


Address	0 - 3	4 - 7	8 - B	C - F
BEF9EC40	54AC0140	8CD70257	656C636F	6D652074
BEF9EC50	6F207468	6520576F	726C6420	6F662074
BEF9EC60	68652070	68794341	52442D4D	21202873
BEF9EC70	65726961	6C290A00	00000000	88ECF9BE
BEF9EC80	D8AF0340	9C840000	00301440	D4EDF9BE
BEF9EC90	01000000	90840000	F0840000	00000000
BEF9ECA0	8C830000	00000000	00000000	00000000
BEF9ECB0	00500240	00000000	88ECF9BE	90AF0340

- Click ***Add Rendering***.



- Select ***ASCII*** and click ***Add Rendering(s)***.

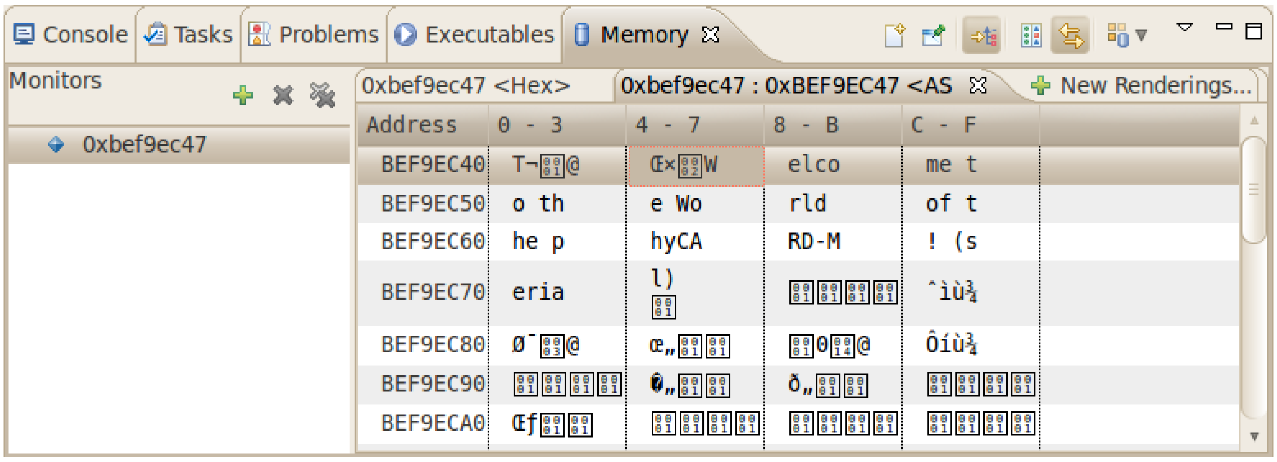


Memory Monitor: 0xbef9ec47 : 0xBEF9EC47

Select rendering(s) to create:

- Hex
- ASCII
- Signed Integer
- Unsigned Integer
- Hex Integer

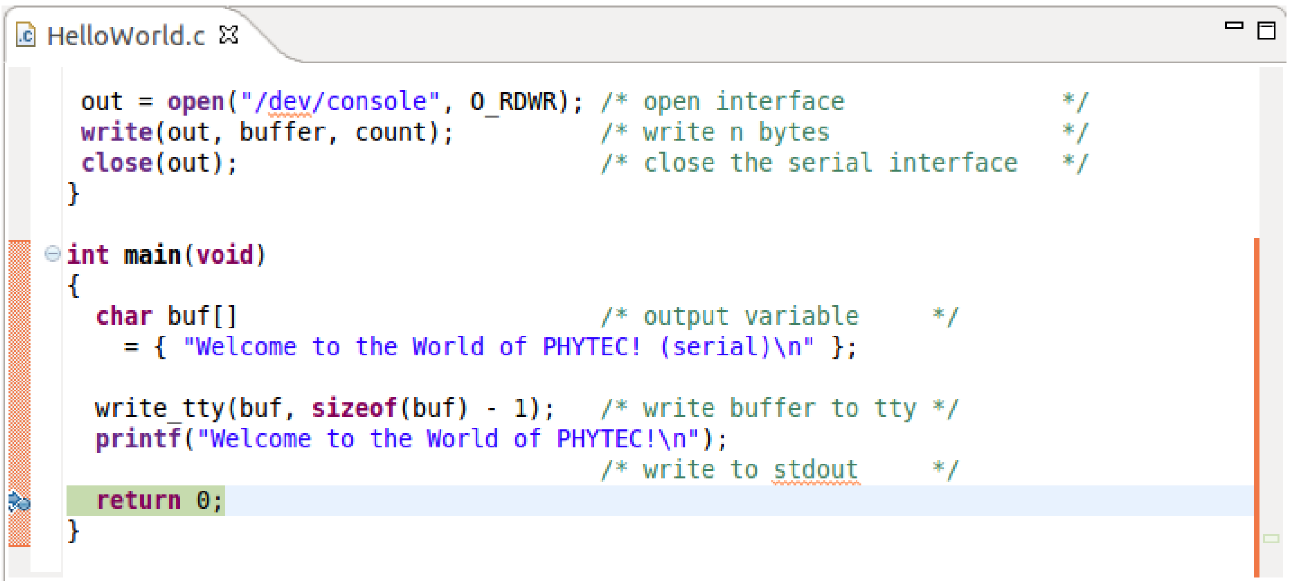
You can see the contents of the variable *buffer* at address *0xbef9ec47* (or at the specific address used on your system).



```
Address    0 - 3   4 - 7   8 - B   C - F
BEF9EC40   T¬▯@    Œx▯W    elco    me t
BEF9EC50   o th    e Wo    rld     of t
BEF9EC60   he p    hyCA    RD-M    ! (s
BEF9EC70   eria    l)▯     ▯▯▯▯    ˆìù¾
BEF9EC80   Ø¯▯@    œ„▯▯    ▯0▯@    Ôíù¾
BEF9EC90   ▯▯▯▯    �„▯▯    ð„▯▯    ▯▯▯▯
BEF9ECA0   Œf▯▯    ▯▯▯▯    ▯▯▯▯    ▯▯▯▯
```

- Now click the ***Resume*** button from the menu bar.

The debugger stops at the breakpoint in the last line of *main()*.



```
   out = open("/dev/console", O_RDWR); /* open interface          */
   write(out, buffer, count);          /* write n bytes           */
   close(out);                         /* close the serial interface   */
  }

 int main(void)
 {
   char buf[]                          /* output variable     */
     = { "Welcome to the World of PHYTEC! (serial)\n" };

   write_tty(buf, sizeof(buf) - 1);    /* write buffer to tty */
   printf("Welcome to the World of PHYTEC!\n");
                                       /* write to stdout     */
   return 0;
 }
```

- Click the ***Resume*** button to end the application.

	You have successfully passed the debugging chapter. You are now able to configure and use *Eclipse* for remote debugging. You can step through a project, watch and change the content of variables, and you can use the memory monitor to view the content at a memory address.

2.3 Working with Qt Creator

In this section we learn how to work with *Qt Creator*. The *Qt* framework provides tools to develop graphical user interfaces. With the help of an example project we show how to compile your own *Qt* based programs and automatically transfer them to the target.

2.3.1 Stop the Running Qt Demo on the Target

A *Qt* demo application starts automatically by default after the target has booted completely (*section 3.2.20*). Before we start compiling and running our example project out of *Qt Creator* we must first close this *Qt* application.

- Open a serial connection with *Microcom*.

Microcom

- After the target is booted login with root and enter the following command to stop the *Qt* application:

```
systemctl stop qt5demo
```

After the *Qt* demo is stopped with the command above we can start to use *Qt Creator*.

If you want to remove the *qt5demo* from the autostart enter the following command:

```
systemctl disable qt5demo
```

2.3.2 Importing the Demo Application

We start with opening the *Qt Creator* in a terminal, because otherwise the correct environment of the toolchain is not set.

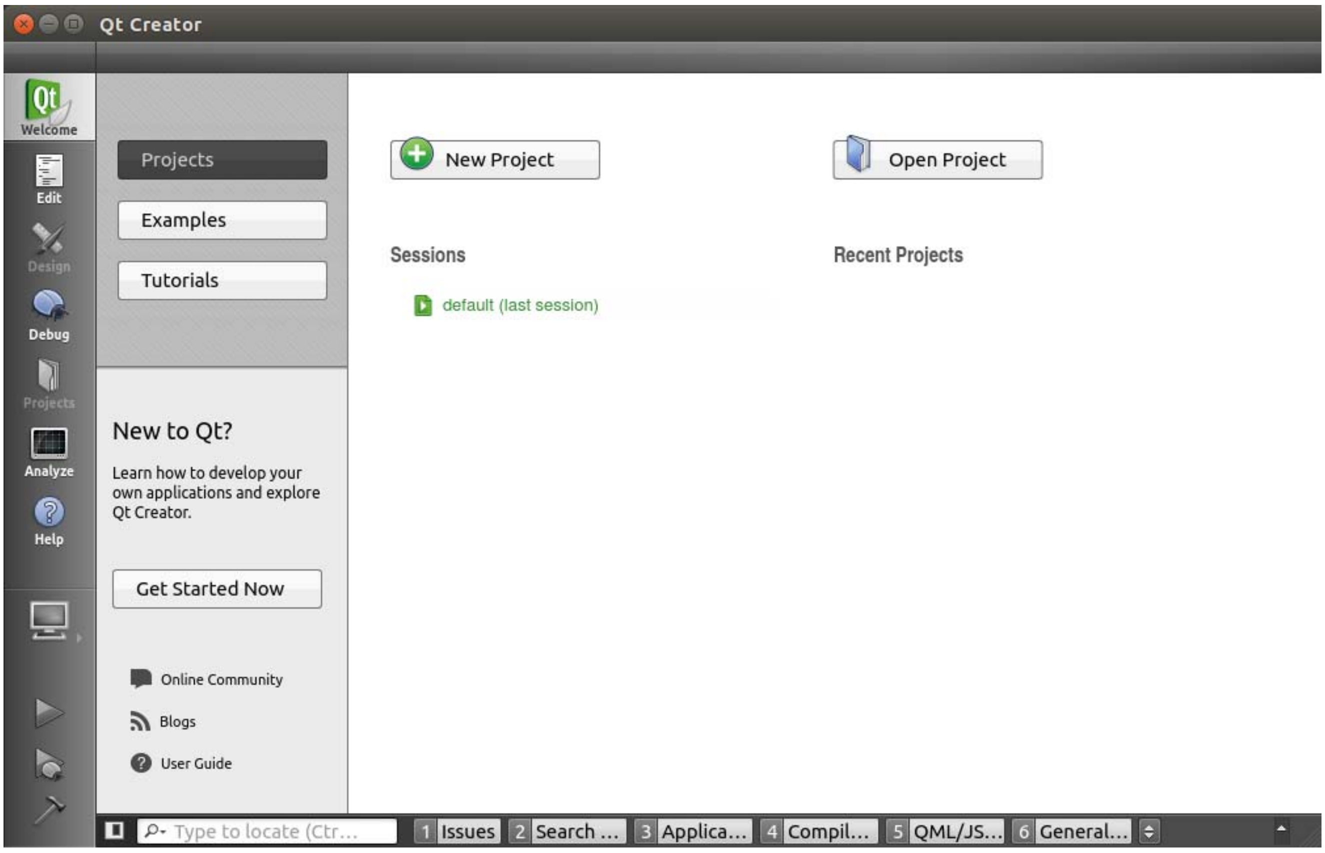
- Open a terminal.

Terminal

- Enter the following command in the terminal to start *Qt Creator*

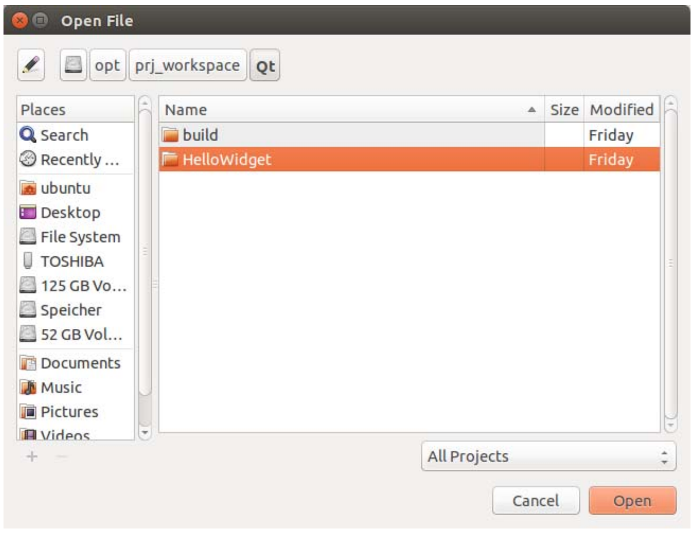
```
/usr/bin/qtcreator.sh &
```

The user interface of *Qt Creator* appears:



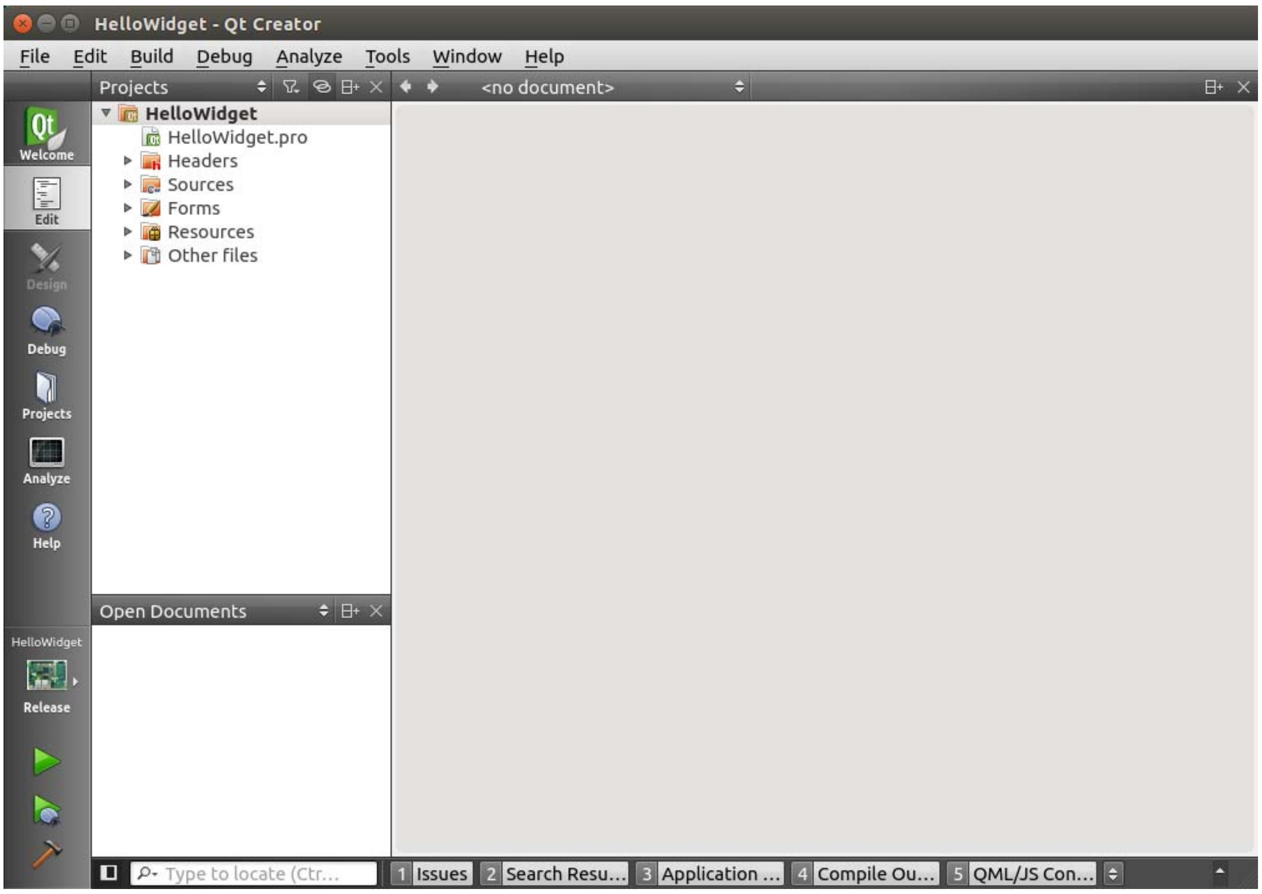
- Now we import the example project by clicking ***Open Project***.

This opens a dialog in which the path to */opt/prj_workspace/Qt* is set automatically.



- Double-click on the ***HelloWidget*** folder.
- Select ***HelloWidget.pro*** and click ***Open.***

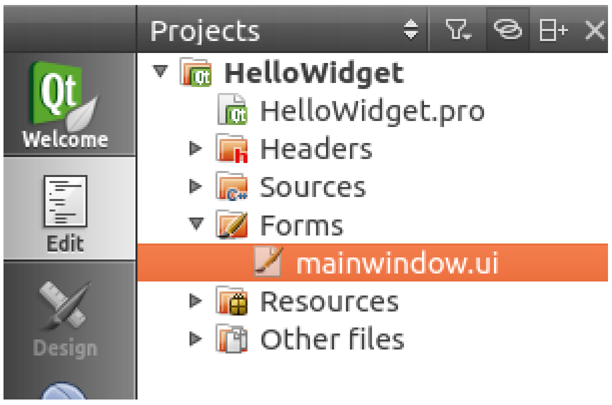
The *HelloWidget* project is now imported into your *Qt Creator* workspace.



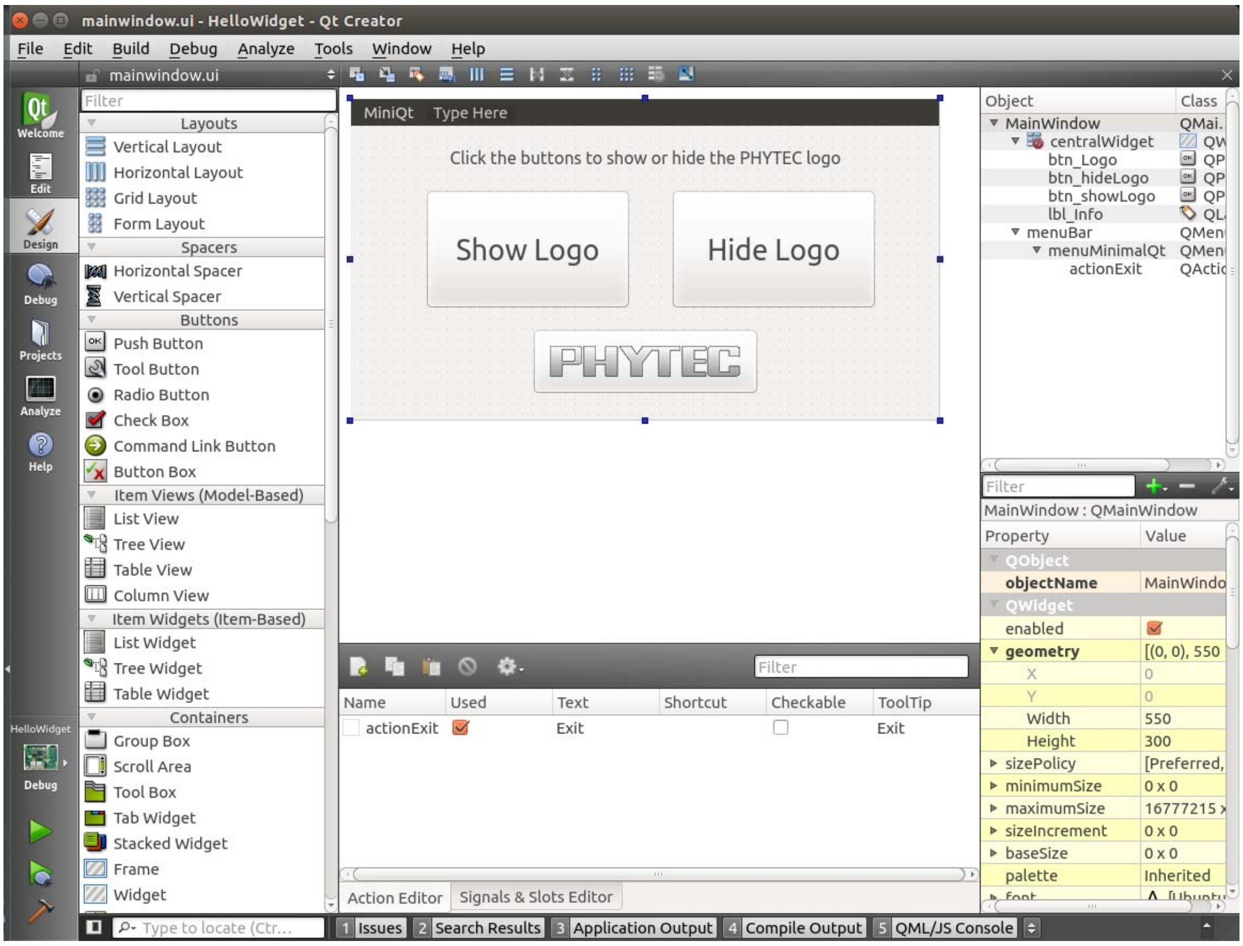
2.3.3 Work with the Demo Application

Our example project is a simple *Qt Widget Application*. First we take a look at the user interface of our example:

- Expand the folder ***Forms*** under the project sidebar on the left side and double-click ***mainwindow.ui***.



Qt Creator opens the design mode and you can see the design of our project, which has a menu bar with an exit action under the menu item *miniQT*, one label and three buttons including one with the PHYTEC logo inside.


mainwindow.ui - HelloWidget - Qt Creator
File Edit Build Debug Analyze Tools Window Help
Welcome Edit Design Debug Projects Analyze Help
Layouts: Vertical Layout, Horizontal Layout, Grid Layout, Form Layout
Spacers: Horizontal Spacer, Vertical Spacer
Buttons: Push Button, Tool Button, Radio Button, Check Box, Command Link Button, Button Box
Item Views (Model-Based): List View, Tree View, Table View, Column View
Item Widgets (Item-Based): List Widget, Tree Widget, Table Widget
Containers: Group Box, Scroll Area, Tool Box, Tab Widget, Stacked Widget, Frame, Widget
MiniQt Type Here
Click the buttons to show or hide the PHYTEC logo
Show Logo
Hide Logo
PHYTEC
Action Editor Signals & Slots Editor
1 Issues 2 Search Results 3 Application Output 4 Compile Output 5 QML/JS Console

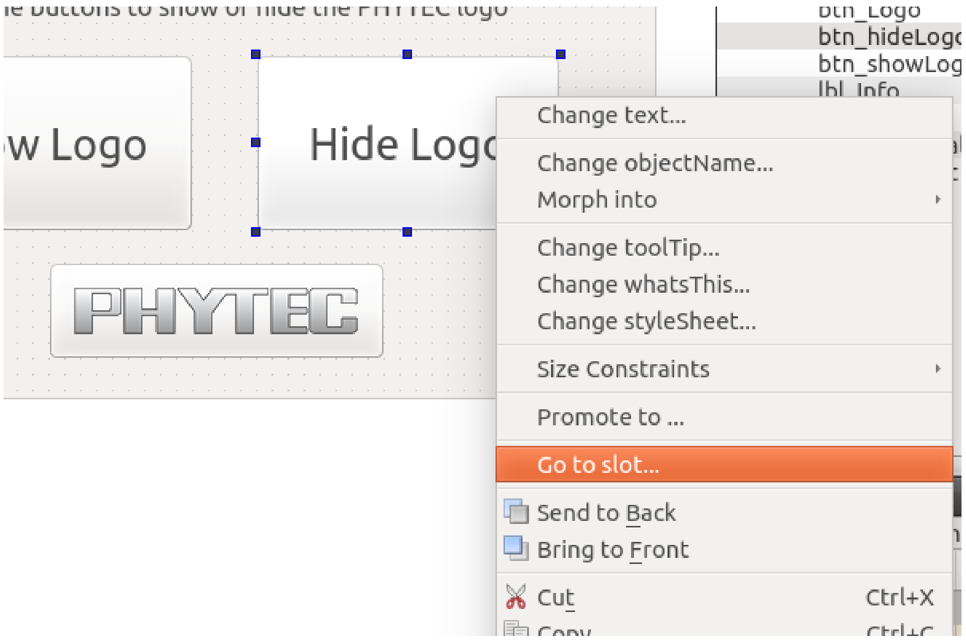
Name	Used	Text	Shortcut	Checkable	ToolTip
actionExit	✓	Exit		☐	Exit

Object	Class
MainWindow	QMai.
centralWidget	QW
btn_Logo	QP
btn_hideLogo	QP
btn_showLogo	QP
lbl_Info	QL
menuBar	QMen
menuMinimalQt	QMen
actionExit	QActic

MainWindow : QMainWindow

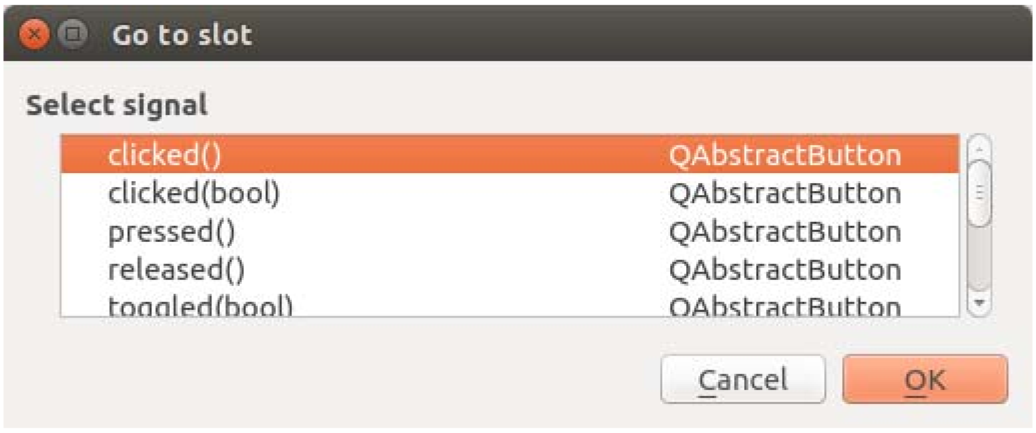
Property	Value
QObject	
objectName	MainWindo
QWidget	
enabled	✓
geometry	[(0, 0), 550
X	0
Y	0
Width	550
Height	300
sizePolicy	[Preferred,
minimumSize	0 x 0
maximumSize	16777215 x
sizeIncrement	0 x 0
baseSize	0 x 0
palette	Inherited

- *Right-click* the *"Hide Logo"* button and select ***Go to slot*** … from the context menu.



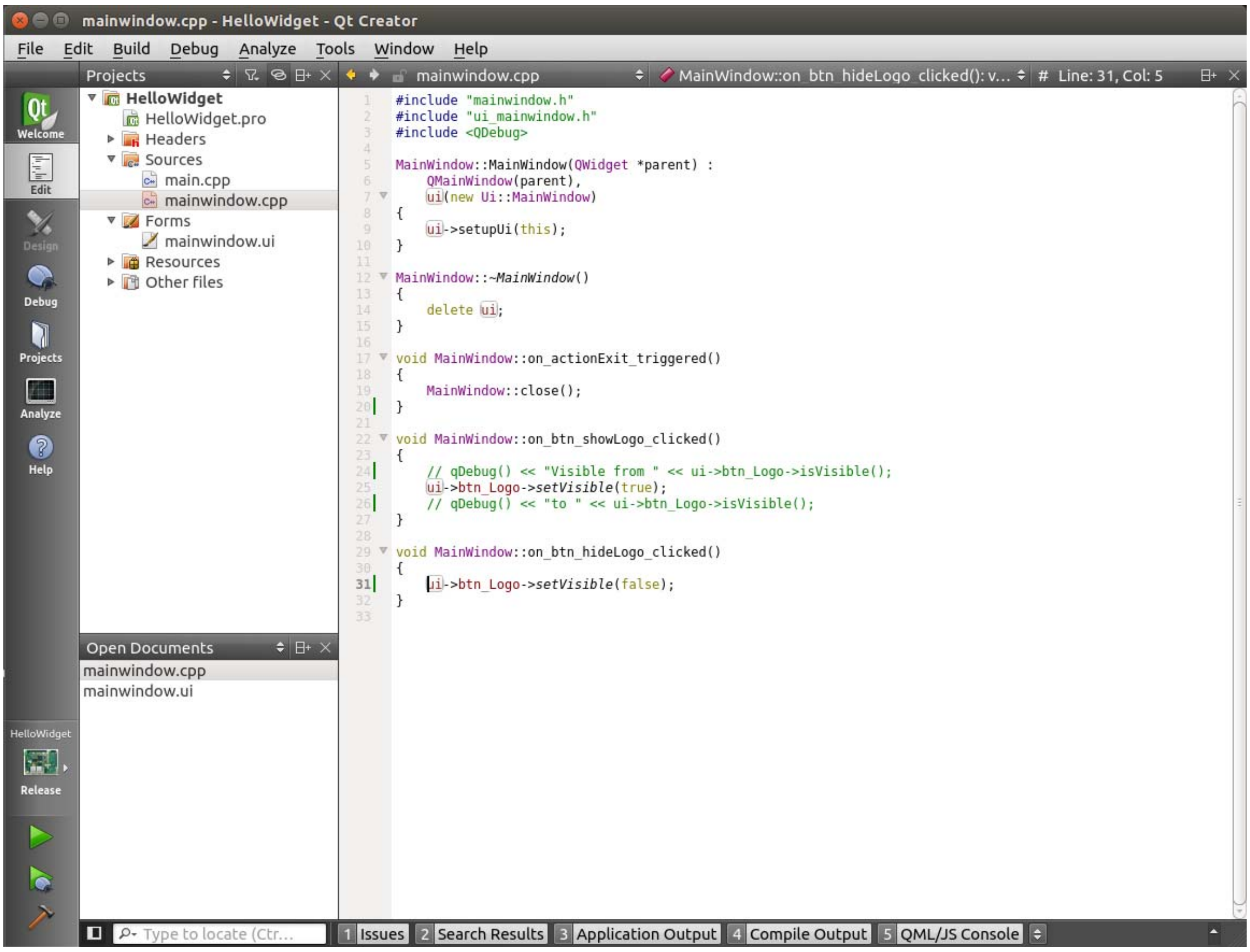
A window opens allowing you to select a signal.

- Select ***clicked()*** and click ***OK.***



clicked()	QAbstractButton
clicked(bool)	QAbstractButton
pressed()	QAbstractButton
released()	QAbstractButton
toggled(bool)	QAbstractButton

Now *Qt Creator* jumps into the *mainwindow.cpp* where you can find the definition of the function *on_btn_hideLogo_clicked()*. You see that clicking this button changes the status of the *btn_Logo* button from visible to hidden.



```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QDebug>

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_actionExit_triggered()
{
    MainWindow::close();
}

void MainWindow::on_btn_showLogo_clicked()
{
    // qDebug() << "Visible from " << ui->btn_Logo->isVisible();
    ui->btn_Logo->setVisible(true);
    // qDebug() << "to " << ui->btn_Logo->isVisible();
}

void MainWindow::on_btn_hideLogo_clicked()
{
    ui->btn_Logo->setVisible(false);
}
```

Next we build and run the example.

2.3.4 Compile and Run the Demo Application on the Target

Now we want to compile and run the demo application on the phyBOARD-Mira. To use and display the demo application connect, an HDMI cable to the HDMI port of the HDMI Adapter *(*PEB-AV-01*)* and connect it to an appropriate display. Also connect an USB wired mouse to the USB host connector (X6) on the phyBOARD.

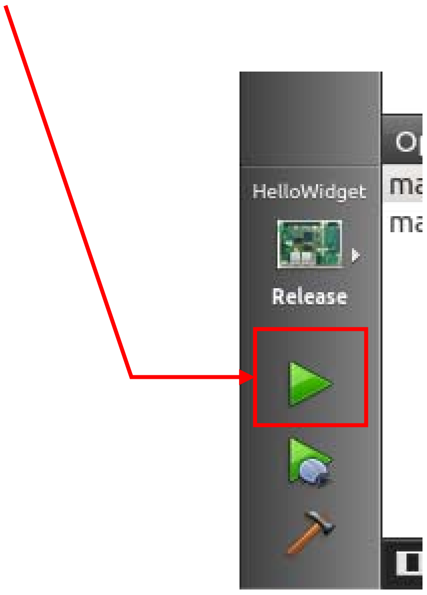
The correct Qt settings are already preset in the Live System, so the project can be build directly.

If you want to check the settings click ***Tools* ▶ *Options...*** in the menu bar**.** A new dialog appears. Click ***Build & Run*** if it is not already selected. Now, you can select different tabs to see the settings for the workspace, compiler, debugger, *Qt* version and a lot other options.

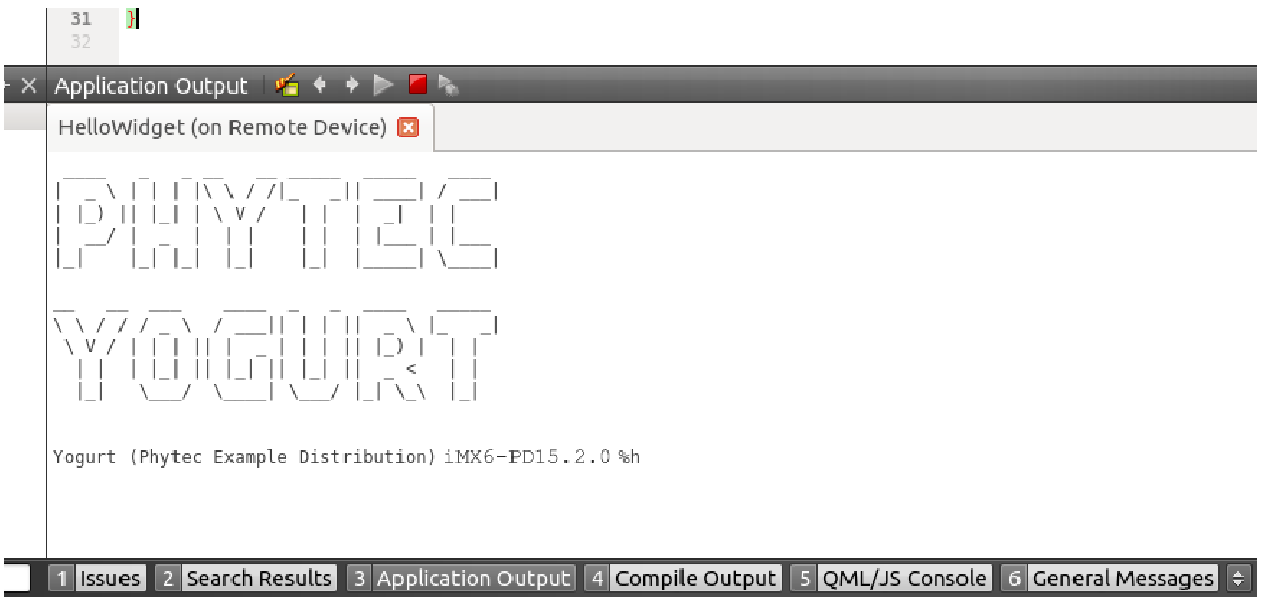
Be sure that the target is connected via Ethernet and is powered on. As described before do not forget to stop all actually running *Qt* applications on the target.

With only one click the project will be built, deployed to the target and executed.

- Click the **green filled triangle** near the bottom of the gray bar on the left.



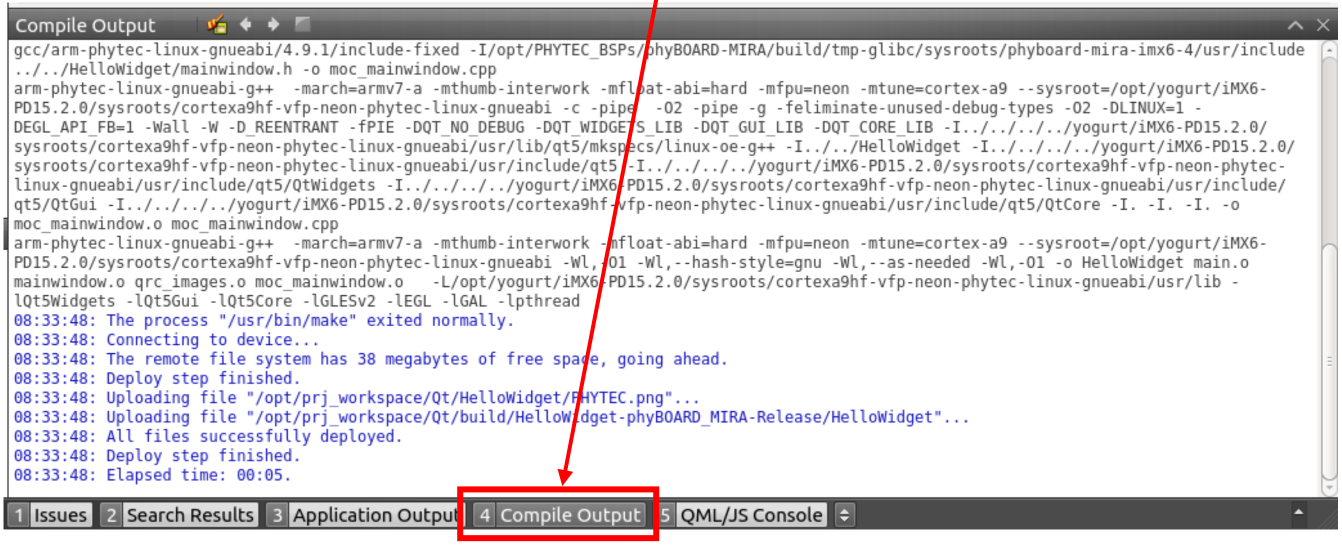
After the target is successfully deployed on to the target the *Application Output* frame is shown in the frame under the *mainwindow.cpp* source code where you can see the prompt from the phyBOARD-Mira.



```
}

 ____  _   ___   _______ _____ ____ 
|  _ \| | | \ \ / /_   _| ____/ ___|
| |_) | |_| |\ V /  | | |  _|| |    
|  __/|  _  | | |   | | | |__| |___ 
|_|   |_| |_| |_|   |_| |_____\____|

__   _____   ____ _   _ ____ _____ 
\ \ / / _ \ / ___| | | |  _ \_   _|
 \ V / | | | |  _| | | | |_) || |  
  | || |_| | |_| | |_| |  _ < | |  
  |_| \___/ \____|\___/|_| \_\|_|  


Yogurt (Phytec Example Distribution) iMX6-PD15.2.0 %h
```

On the connected display you will see the *HelloWidget* application. Click the big button with the PHYTEC logo to enable and disable the Welcome label.

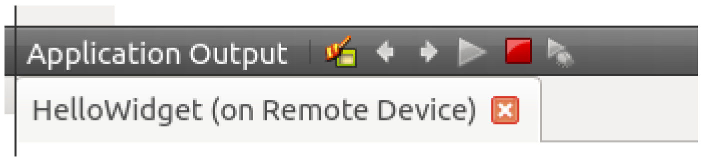
- To see the different build steps click ***Compile output*** which can be found in the bottom menu bar.



```
gcc/arm-phytec-linux-gnueabi/4.9.1/include-fixed -I/opt/PHYTEC_BSPs/phyBOARD-MIRA/build/tmp-glibc/sysroots/phyboard-mira-imx6-4/usr/include
../../HelloWidget/mainwindow.h -o moc_mainwindow.cpp
arm-phytec-linux-gnueabi-g++  -march=armv7-a -mthumb-interwork -mfloat-abi=hard -mfpu=neon -mtune=cortex-a9 --sysroot=/opt/yogurt/iMX6-
PD15.2.0/sysroots/cortexa9hf-vfp-neon-phytec-linux-gnueabi -c -pipe  -O2 -pipe -g -feliminate-unused-debug-types -O2 -DLINUX=1 -
DEGL_API_FB=1 -Wall -W -D_REENTRANT -fPIE -DQT_NO_DEBUG -DQT_WIDGETS_LIB -DQT_GUI_LIB -DQT_CORE_LIB -I../../../../yogurt/iMX6-PD15.2.0/
sysroots/cortexa9hf-vfp-neon-phytec-linux-gnueabi/usr/lib/qt5/mkspecs/linux-oe-g++ -I../../HelloWidget -I../../../../yogurt/iMX6-PD15.2.0/
sysroots/cortexa9hf-vfp-neon-phytec-linux-gnueabi/usr/include/qt5 -I../../../../yogurt/iMX6-PD15.2.0/sysroots/cortexa9hf-vfp-neon-phytec-
linux-gnueabi/usr/include/qt5/QtWidgets -I../../../../yogurt/iMX6-PD15.2.0/sysroots/cortexa9hf-vfp-neon-phytec-linux-gnueabi/usr/include/
qt5/QtGui -I../../../../yogurt/iMX6-PD15.2.0/sysroots/cortexa9hf-vfp-neon-phytec-linux-gnueabi/usr/include/qt5/QtCore -I. -I. -I. -o
moc_mainwindow.o moc_mainwindow.cpp
arm-phytec-linux-gnueabi-g++  -march=armv7-a -mthumb-interwork -mfloat-abi=hard -mfpu=neon -mtune=cortex-a9 --sysroot=/opt/yogurt/iMX6-
PD15.2.0/sysroots/cortexa9hf-vfp-neon-phytec-linux-gnueabi -Wl,-O1 -Wl,--hash-style=gnu -Wl,--as-needed -Wl,-O1 -o HelloWidget main.o
mainwindow.o qrc_images.o moc_mainwindow.o   -L/opt/yogurt/iMX6-PD15.2.0/sysroots/cortexa9hf-vfp-neon-phytec-linux-gnueabi/usr/lib -
lQt5Widgets -lQt5Gui -lQt5Core -lGLESv2 -lEGL -lGAL -lpthread
08:33:48: The process "/usr/bin/make" exited normally.
08:33:48: Connecting to device...
08:33:48: The remote file system has 38 megabytes of free space, going ahead.
08:33:48: Deploy step finished.
08:33:48: Uploading file "/opt/prj_workspace/Qt/HelloWidget/PHYTEC.png"...
08:33:48: Uploading file "/opt/prj_workspace/Qt/build/HelloWidget-phyBOARD_MIRA-Release/HelloWidget"...
08:33:48: All files successfully deployed.
08:33:48: Deploy step finished.
08:33:48: Elapsed time: 00:05.
```

The *Compile output* frame is opened and you can scroll threw through the compile steps.

- If you want to stop the application on the target switch back to the ***Application output*** frame and click the ***red rectangle*** there.



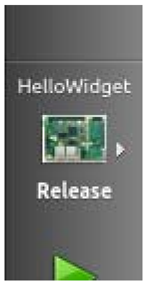
	You have successfully imported and built a *Qt* project with *Qt Creator*. You have also learned how to download and execute your application on the target.

2.3.5 Compile and Run the Demo Application on the Host

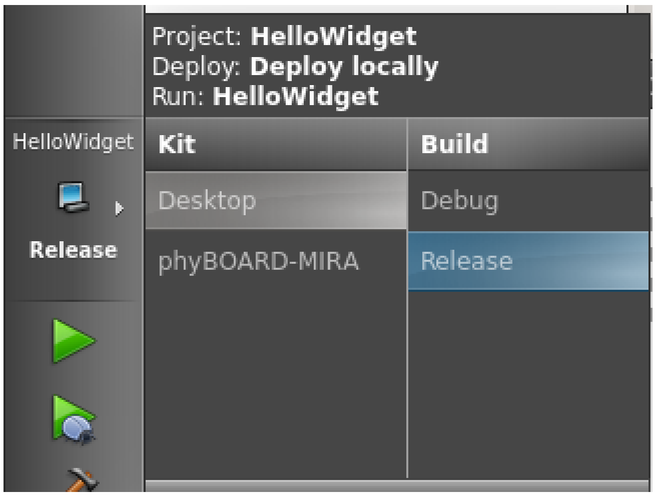
In some cases you do not want to compile and run the application on the target. For example if the phyBOARD-Mira is not connected to the host or if you only changed some user interface relevant things and you do not want to copy the application and all the necessary resources to the target. Running the demo application on the host is faster, but features specific to the target do not work.

To change the target for downloading and running the application do following steps:

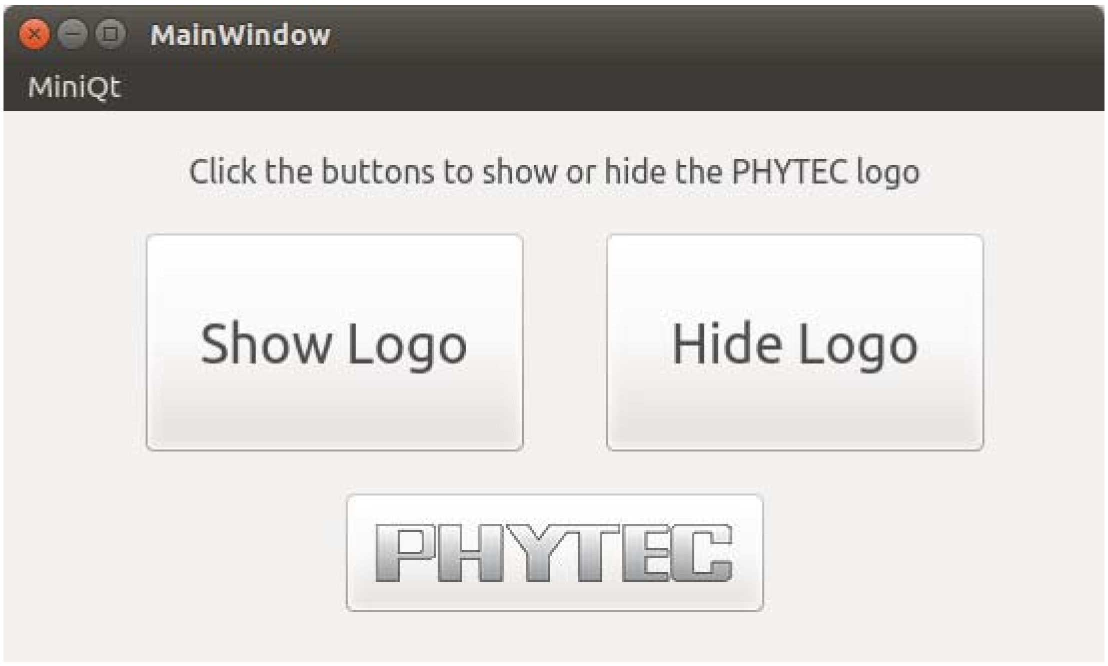
- In the gray bar on the left click the button with the small phyBOARD-Mira picture.



- In the opened context menu select ***Desktop*** as *Kit* and ***Release*** as *Build*.


Project: **HelloWidget**
Deploy: **Deploy locally**
Run: **HelloWidget**

- Click outside of the context menu to close it.
- Now you can start the compilation and execution of the application by clicking the ***green triangle***.
- After the application is compiled a window opens displaying the application running.



- Close the running application.

2.3.6 Debugging the Demo Application

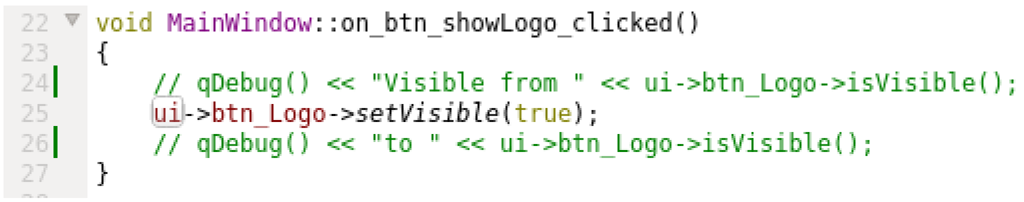
We finish the *Qt* Creator chapter by showing how to debug the *HelloWidget* demo application.

2.3.6.1 Using QDebug for simple Debugging Messages

In our first debugging step we use the *QDebug* class. This class provides an output stream for debugging information. It is used whenever the developer needs to write out debugging or tracing information to a device, file, string or console.

To make use of the *QDebug* functions *QDebug* is already included in the header of the file *mainwindow.cpp*.

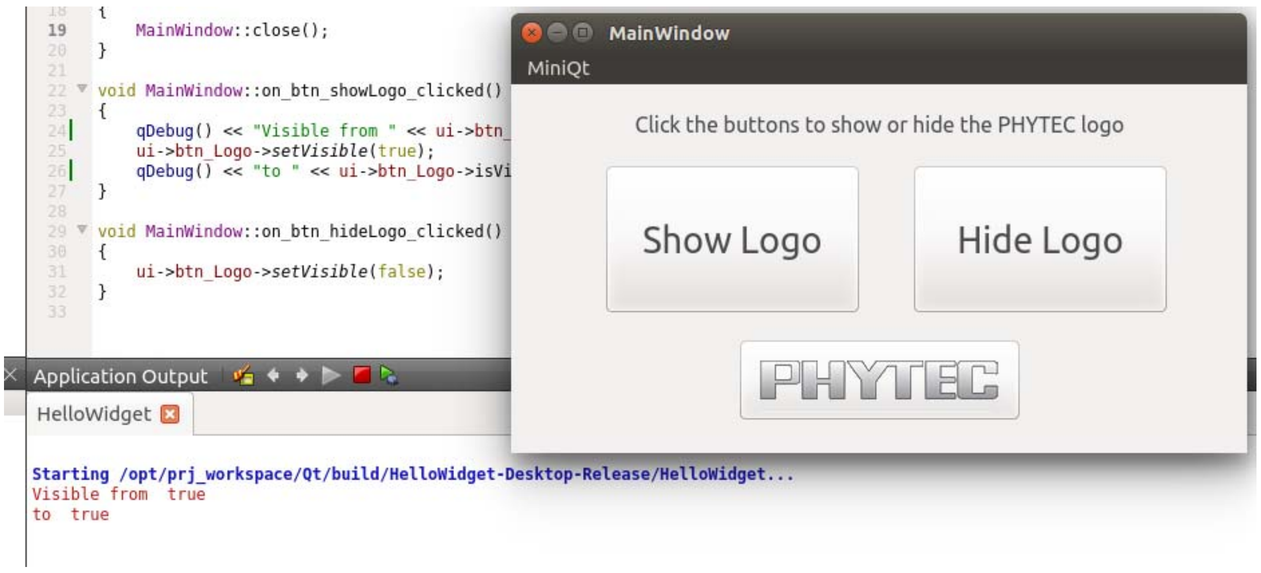
In this file you will also see two out-commented lines in the function *on_btn_showLogo_clicked()* which start with a *qDebug()* command. This is an example how to make use of qDebug().



```
void MainWindow::on_btn_showLogo_clicked()
{
    // qDebug() << "Visible from " << ui->btn_Logo->isVisible();
    ui->btn_Logo->setVisible(true);
    // qDebug() << "to " << ui->btn_Logo->isVisible();
}
```

- Remove the comment flags (//) from both "qDebug lines" to enable the two debug messages.
- Save the changes with ***Ctrl + S*** .
- Open the *Application Output* if it is not already opened.
- Build and run the application on the *Desktop* with the build configuration *Release,* as it was shown in the chapter before.

A window with the running application opens.


```
{
    MainWindow::close();
}

void MainWindow::on_btn_showLogo_clicked()
{
    qDebug() << "Visible from " << ui->btn_
    ui->btn_Logo->setVisible(true);
    qDebug() << "to " << ui->btn_Logo->isVi
}

void MainWindow::on_btn_hideLogo_clicked()
{
    ui->btn_Logo->setVisible(false);
}
```

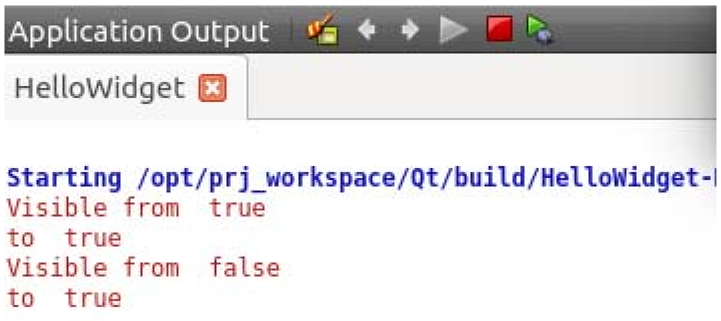
```
Starting /opt/prj_workspace/Qt/build/HelloWidget-Desktop-Release/HelloWidget...
Visible from  true
to  true
```

- Press the *Show Logo* button.

The logo was visible before we pressed the button so we see the appropriate message from qDebug() in the *Application Output*.

- Press the *Hide Logo* button and then the *Show Logo* button.

Now the logo was hidden before we pressed the *Show Logo* button again and we see the appropriate message from qDebug() in the *Application Output*.



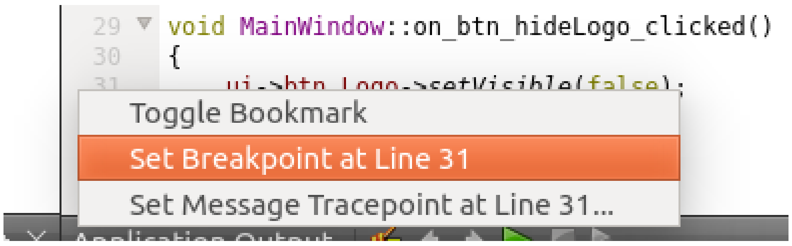
```
Starting /opt/prj_workspace/Qt/build/HelloWidget-
Visible from  true
to  true
Visible from  false
to  true
```

QDebug is a simple way to generate debug information. In the next chapter we use the debugger integrated in *Qt Creator*.

2.3.6.2 Using the integrated Qt Creator Debugger

Before we start the integrated Debugger we first set a breakpoint where the demo application stops.

- Open the *mainwindow.cpp* if it is not already open.
- Right-click in front of line number 31.

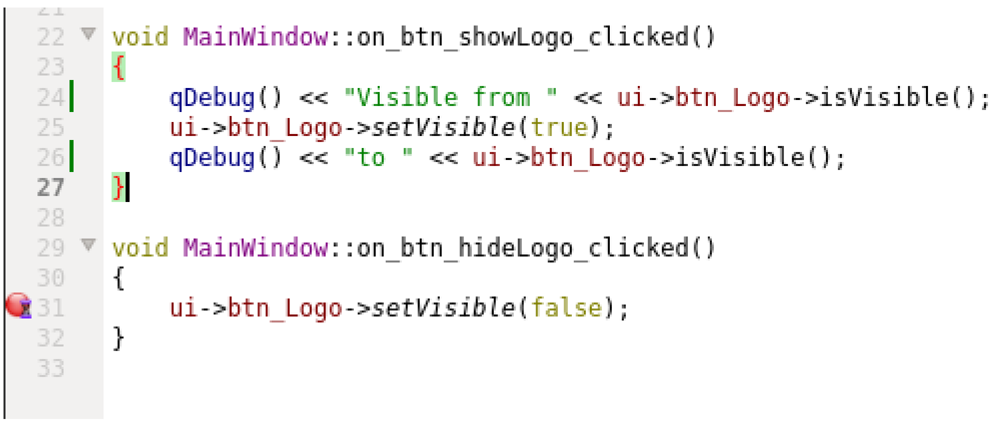


```
29  void MainWindow::on_btn_hideLogo_clicked()
30  {
```

Toggle Bookmark
Set Breakpoint at Line 31
Set Message Tracepoint at Line 31...

- In the context menu click on *Set Breakpoint at Line 31* to add a breakpoint.

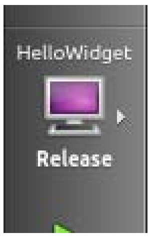
You now see a red filled circle with a small sand glass in front of this line.



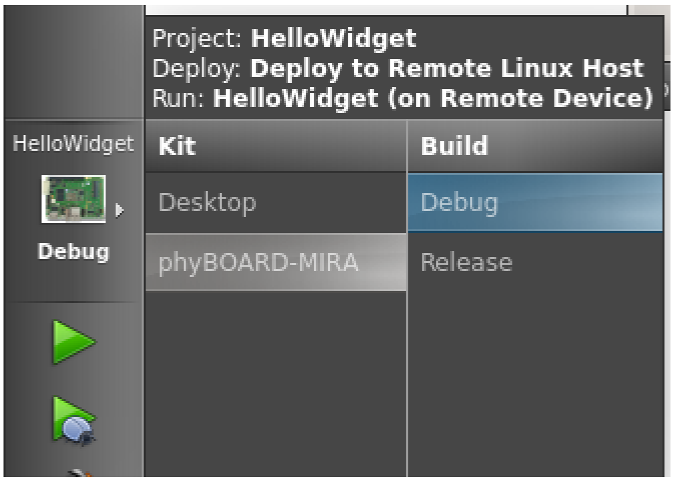
```
22  void MainWindow::on_btn_showLogo_clicked()
23  {
24      qDebug() << "Visible from " << ui->btn_Logo->isVisible();
25      ui->btn_Logo->setVisible(true);
26      qDebug() << "to " << ui->btn_Logo->isVisible();
27  }
28
29  void MainWindow::on_btn_hideLogo_clicked()
30  {
31      ui->btn_Logo->setVisible(false);
32  }
33
```

The next step is to change the build configuration from *Release* to *Debug*.

- In the gray bar on the left click the button with the small Desktop icon.



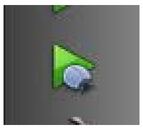
- In the opened context menu select ***phyBOARD-Mira*** as *Kit* and ***Debug*** as *Build*.



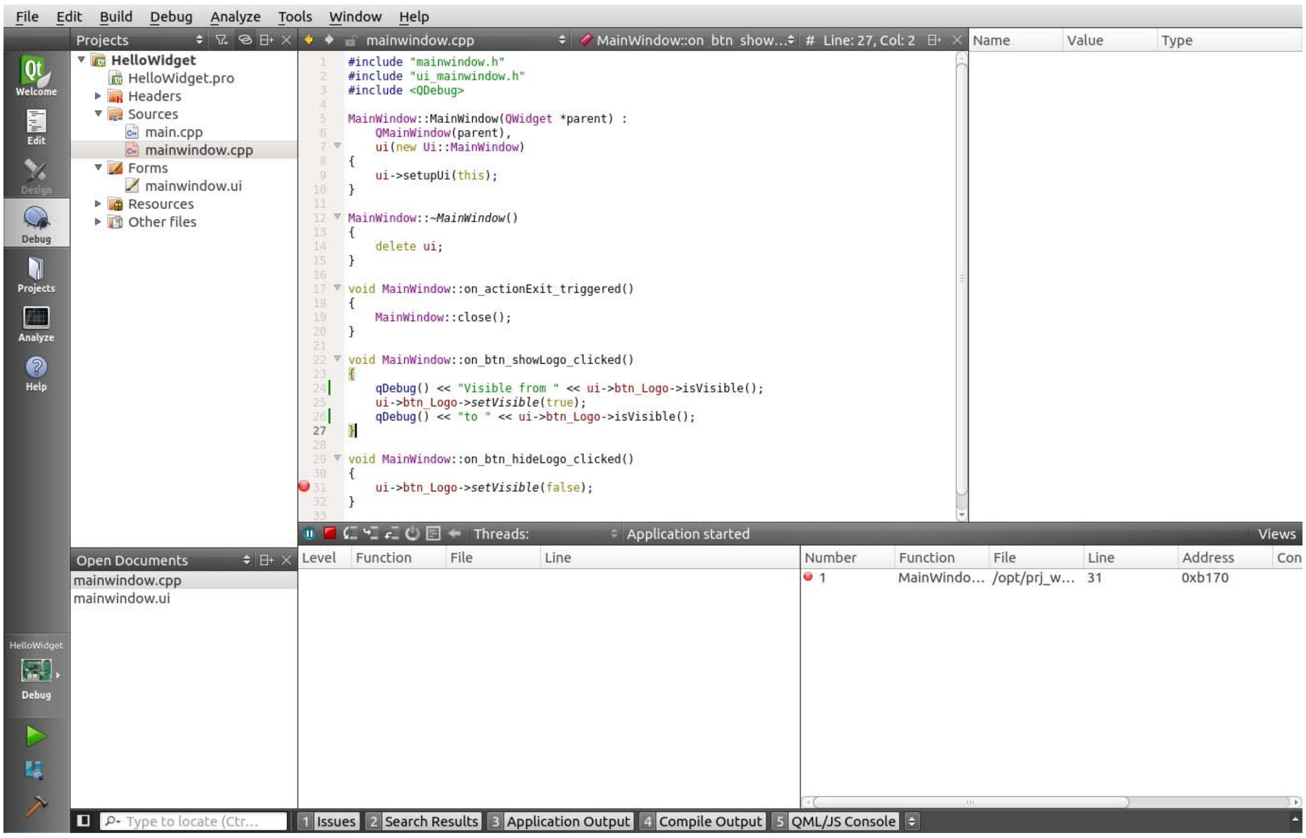
Project: **HelloWidget**
Deploy: **Deploy to Remote Linux Host**
Run: **HelloWidget (on Remote Device)**

Be sure that the target is connected via Ethernet to the host. If you use our PEB-AV-01 with a connected monitor you must also connect an USB mouse to use the demo application.

- Start Debugging by clicking on the green filled triangle with the small magnifier.



The debugger starts and *Qt Creator* changes his view to the Debug mode.



```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QDebug>

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_actionExit_triggered()
{
    MainWindow::close();
}

void MainWindow::on_btn_showLogo_clicked()
{
    qDebug() << "Visible from " << ui->btn_Logo->isVisible();
    ui->btn_Logo->setVisible(true);
    qDebug() << "to " << ui->btn_Logo->isVisible();
}

void MainWindow::on_btn_hideLogo_clicked()
{
    ui->btn_Logo->setVisible(false);
}
```

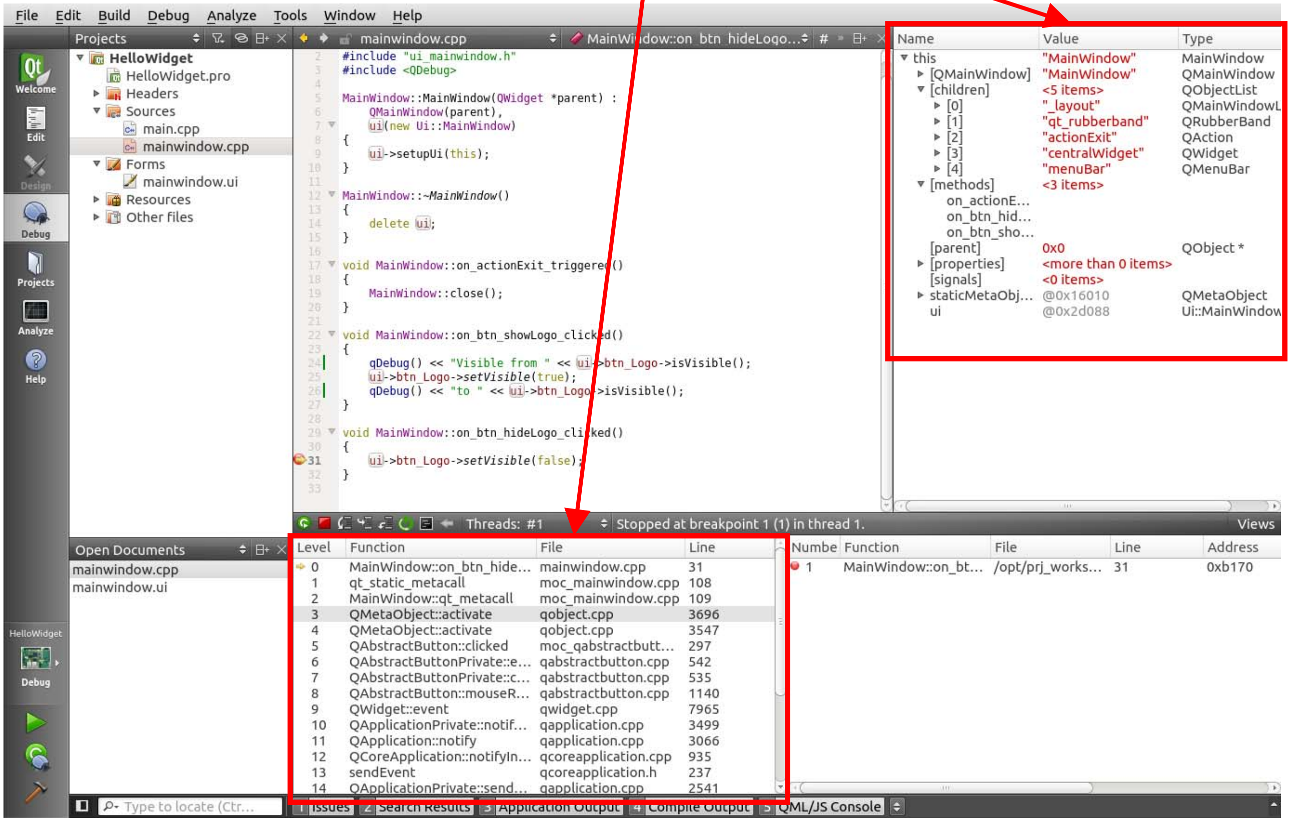
Level	Function	File	Line

Number	Function	File	Line	Address	Con
● 1	MainWindo...	/opt/prj_w...	31	0xb170	

The demo application is shown on the connected display.

- In the running demo application click on the *Hide Logo* button.

When the button is pressed the debugger stops at the created breakpoint, because the function is called. You can now watch the *stack* or the *Locals and Expressions*.



```
#include "ui_mainwindow.h"
#include <QDebug>

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_actionExit_triggered()
{
    MainWindow::close();
}

void MainWindow::on_btn_showLogo_clicked()
{
    qDebug() << "Visible from " << ui->btn_Logo->isVisible();
    ui->btn_Logo->setVisible(true);
    qDebug() << "to " << ui->btn_Logo->isVisible();
}

void MainWindow::on_btn_hideLogo_clicked()
{
    ui->btn_Logo->setVisible(false);
}
```

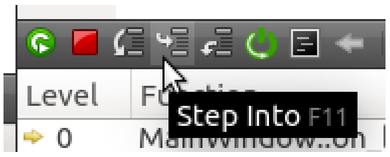
Stopped at breakpoint 1 (1) in thread 1.

Name	Value	Type
this	"MainWindow"	MainWindow
[QMainWindow]	"MainWindow"	QMainWindow
[children]	<5 items>	QObjectList
[0]	"_layout"	QMainWindowL
[1]	"qt_rubberband"	QRubberBand
[2]	"actionExit"	QAction
[3]	"centralWidget"	QWidget
[4]	"menuBar"	QMenuBar
[methods]	<3 items>	
on_actionE…		
on_btn_hid…		
on_btn_sho…		
[parent]	0x0	QObject *
[properties]	<more than 0 items>	
[signals]	<0 items>	
staticMetaObj…	@0x16010	QMetaObject
ui	@0x2d088	Ui::MainWindow

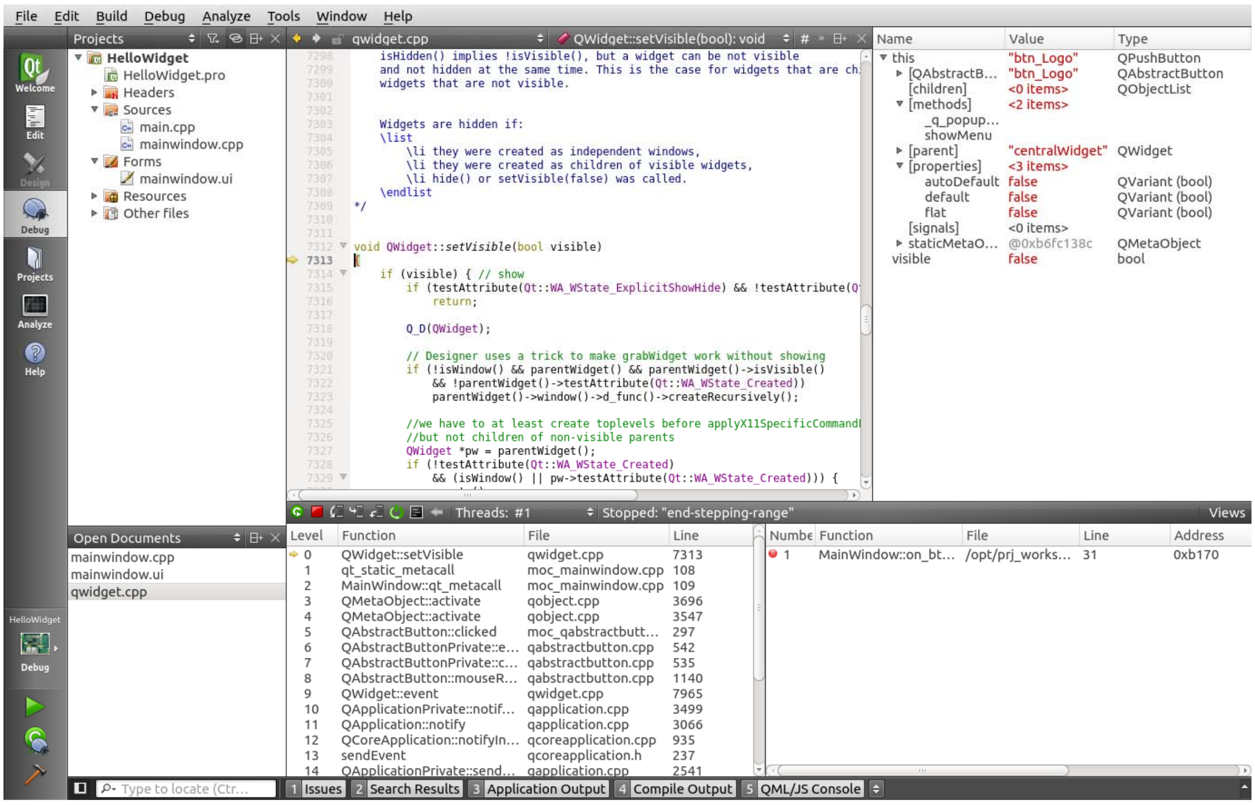
Level	Function	File	Line
0	MainWindow::on_btn_hide…	mainwindow.cpp	31
1	qt_static_metacall	moc_mainwindow.cpp	108
2	MainWindow::qt_metacall	moc_mainwindow.cpp	109
3	QMetaObject::activate	qobject.cpp	3696
4	QMetaObject::activate	qobject.cpp	3547
5	QAbstractButton::clicked	moc_qabstractbutt…	297
6	QAbstractButtonPrivate::e…	qabstractbutton.cpp	542
7	QAbstractButtonPrivate::c…	qabstractbutton.cpp	535
8	QAbstractButton::mouseR…	qabstractbutton.cpp	1140
9	QWidget::event	qwidget.cpp	7965
10	QApplicationPrivate::notif…	qapplication.cpp	3499
11	QApplication::notify	qapplication.cpp	3066
12	QCoreApplication::notifyIn…	qcoreapplication.cpp	935
13	sendEvent	qcoreapplication.h	237
14	QApplicationPrivate::send…	qapplication.cpp	2541

Numbe	Function	File	Line	Address
1	MainWindow::on_bt…	/opt/prj_works…	31	0xb170

- Step into the *setVisible* function by pressing the *Step Into* button.



Now the definition of this function is opened.



```
    isHidden() implies !isVisible(), but a widget can be not visible
    and not hidden at the same time. This is the case for widgets that are ch
    widgets that are not visible.


    Widgets are hidden if:
    \list
        \li they were created as independent windows,
        \li they were created as children of visible widgets,
        \li hide() or setVisible(false) was called.
    \endlist
*/


void QWidget::setVisible(bool visible)
{
    if (visible) { // show
        if (testAttribute(Qt::WA_WState_ExplicitShowHide) && !testAttribute(Q
            return;

        Q_D(QWidget);

        // Designer uses a trick to make grabWidget work without showing
        if (!isWindow() && parentWidget() && parentWidget()->isVisible()
            && !parentWidget()->testAttribute(Qt::WA_WState_Created))
            parentWidget()->window()->d_func()->createRecursively();

        //we have to at least create toplevels before applyX11SpecificCommandl
        //but not children of non-visible parents
        QWidget *pw = parentWidget();
        if (!testAttribute(Qt::WA_WState_Created)
            && (isWindow() || pw->testAttribute(Qt::WA_WState_Created))) {
```

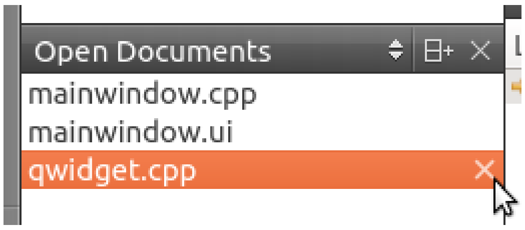
Name	Value	Type
this	"btn_Logo"	QPushButton
[QAbstractB...	"btn_Logo"	QAbstractButton
[children]	<0 items>	QObjectList
[methods]	<2 items>	
_q_popup...		
showMenu		
[parent]	"centralWidget"	QWidget
[properties]	<3 items>	
autoDefault	false	QVariant (bool)
default	false	QVariant (bool)
flat	false	QVariant (bool)
[signals]	<0 items>	
staticMetaO...	@0xb6fc138c	QMetaObject
visible	false	bool

Level	Function	File	Line
0	QWidget::setVisible	qwidget.cpp	7313
1	qt_static_metacall	moc_mainwindow.cpp	108
2	MainWindow::qt_metacall	moc_mainwindow.cpp	109
3	QMetaObject::activate	qobject.cpp	3696
4	QMetaObject::activate	qobject.cpp	3547
5	QAbstractButton::clicked	moc_qabstractbutt...	297
6	QAbstractButtonPrivate::e...	qabstractbutton.cpp	542
7	QAbstractButtonPrivate::c...	qabstractbutton.cpp	535
8	QAbstractButton::mouseR...	qabstractbutton.cpp	1140
9	QWidget::event	qwidget.cpp	7965
10	QApplicationPrivate::notif...	qapplication.cpp	3499
11	QApplication::notify	qapplication.cpp	3066
12	QCoreApplication::notifyIn...	qcoreapplication.cpp	935
13	sendEvent	qcoreapplication.h	237
14	QApplicationPrivate::send...	qapplication.cpp	2541

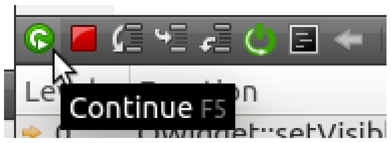
Numbe	Function	File	Line	Address
1	MainWindow::on_bt...	/opt/prj_works...	31	0xb170

As many other calls are following it is recommended to close the opened *qwidget.cpp*.

- Close *qwidget.cpp* in the Open Documents frame by clicking the *X* on the right side.

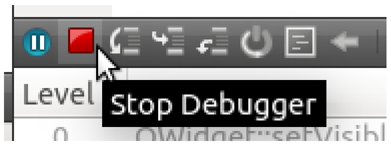


- Continue the stopped application by clicking on *Continue*.



Now the demo application is continued and waiting for user interactions.

- Stop the Debugger by pressing *Stop Debugger*.



	You have successfully finished our short introduction to the Debugger from *Qt Creator*.

Now you know how to work with *Eclipse* and *Qt Creator*. You also learned how to develop and execute an application on the phyBOARD-Mira i.MX 6, so you are well prepared to start your project. The following section will give you detailed information on the different features and interfaces of the phyBOARD-Mira and how to use them within your application.

If your project is more complex, or if you crave more information about working with the BSP, continue with *chapter 4*. *Chapter 4ff* includes step by step instructions on how to modify and download the BSP using *Yocto*. They also include system level information on the phyBOARD- Mira i.MX 6.

3 Accessing the phyBOARD-Mira Features

PHYTEC phyBOARD-Mira is fully equipped with all mechanical and electrical components necessary for the speedy and secure start-up.

3.1 Overview of the phyBOARD-Mira Peripherals

The phyBOARD-Mira is depicted in *Figure 2*. It features many different interfaces and is equipped with the components as listed in *Table 2*, and *Table 3*. For a more detailed description of each peripheral refer to the appropriate chapter listed in the applicable table. *Figure 2* highlights the location of each peripheral for easy identification.

3.1.1 Connectors and Pin Header

Table 2 lists all available connectors on the phyBOARD-Mira. *Figure 2* highlights the location of each connector for easy identification.

Reference Designator	Description	See Section
X1	phyCORE-Connector (2 x Samtec 2x70 pin)	
X2	Power supply 5 V only (via 6-pole WAGO male header, or 2-pole PHOENIX MINI COMBICON base strip)	*3.2.1.1*
X3	CAN connector (2×5 pin header 2.54 mm pitch)	*3.2.5*
X4	Ethernet 0 connector (RJ45 with speed and link LED)	*3.2.3*
X5	USB On-The-Go connector (USB Micro-AB)	*3.2.4*
X6	USB host connector (USB 2.0 Standard-A)	
X7	PCI Express connector (Mini PCI Express)	*3.2.7*
X8	Display backlight supply and control connector (5-pole Molex)	*3.2.11*
X9	Display LVDS connector(20 pin FCC connector 1 mm pitch)	*3.2.10*
X10	Camera phyCAM-S+ connector (8-pole Hirose Board-to-Wire Connector 1.25 mm pitch)	*3.2.8*
X13	A/V connector #1 (2×8 dual entry connector 2 mm pitch)	*3.2.16*
X14	A/V connector #2 (2×20 dual entry connector 2 mm pitch)	
X17	Expansion connector (2×30 socket connector 2 mm pitch)	*3.2.17*
X21	Touch connector (1x4 pin header 2.54 mm pitch)	*3.2.12*
X22	Secure Digital / Multi Media Card (Micro-slot)	*3.2.6*
X23	RS-232 with RTS and CTS, or RS-485 (UART3 2×5 pin header 2.54 mm pitch)	*3.2.2*
X28	HDMI connector (Typ-A)	*3.2.9*
X29	Backup voltage connector (1x2 pin header 2.54 mm pitch)	*3.2.1.3*

Table 2: phyBOARD-Mira Connectors and Pin Headers

⚠	Ensure that all module connections are not to exceed their expressed maximum voltage or current. Maximum signal input values are indicated in the corresponding controller User's Manual/Data Sheets. As damage from improper connections varies according to use and application, it is the user's responsibility to take appropriate safety measures to ensure that the module connections are protected from overloading through connected peripherals.

3.1.2 LEDs

The phyBOARD-Mira is populated with three LEDs to indicate the status of the USB VBUS voltages, as well as of the power supply voltage. The fourth LED is a user programmable RGB-LED.

Figure 2 shows the location of the LEDs. Their function is listed in the table below:

LED	Color	Description	See Section
D2	red	3.3 V voltage generation of the phyBOARD-Mira	*3.2.1.2*
D6	RGB	User programmable RGB-LED	*3.2.13*
D13	green	Indicates presence of VBUS at the USB host interface	*3.2.4*
D14	green	Indicates presence of VBUS at the USB OTG interface	

Table 3: phyBOARD-Mira LEDs Descriptions

3.1.3 Switches

The phyBOARD-Mira is populated with two switches, one to reset the phyBOARD-Mira and another to configure the boot sequence.

Figure 2 shows the location of the switches. Their function is listed in the table below:

Switch	Description	See Section
S1	Reset Button	*3.2.15*
S2	Boot Switch	*3.2.14*

Table 4: phyBOARD-Mira Switches Description

3.1.4 Jumpers

The phyBOARD-Mira comes pre-configured with one removable jumper (JP) and several solder jumpers (J). The jumpers allow flexibility of configuring a limited number of features for development constraint purposes.

⚠	Due to the small footprint of the solder jumpers (J) we do not recommend manual jumper modifications. This might also render the warranty invalid. Because of that only the removable jumper is described in this section. For information on the solder jumpers see *section4.6* and contact our sales team if you need jumper configurations different from the default configuration.

The function of the removable jumper on the phyBOARD-Mira is shown in *Table 5*. More detailed information can be found in the appropriate section.

Figure 2 shows the location of jumper JP2.

Jumper	Description	See Section
JP2	CAN Termination	*3.2.5*

Table 5: phyBOARD-Mira Jumper Description

	Detailed descriptions of the assembled connectors, jumpers and switches can be found in the following chapters.

3.2 Functional Components on the phyBOARD-Mira SBC

This section describes the functional components of the phyBOARD-Mira. Each subsection details a particular connector/interface and associated jumpers for configuring that interface.

3.2.1 Power Supply

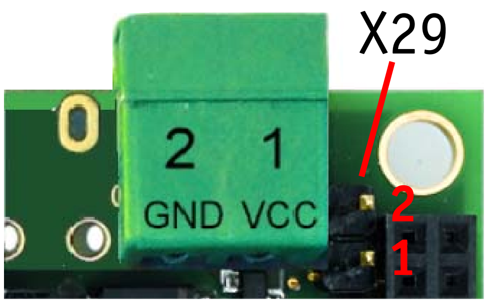
	Do not change modules or jumper settings while the phyBOARD-Mira is supplied with power!

3.2.1.1 Power Connectors (X2)

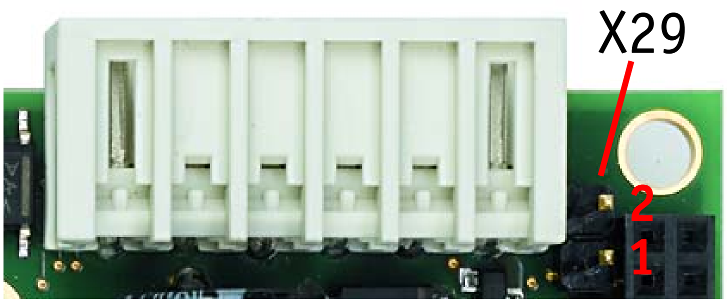
The phyBOARD-Mira is available with two different power supply connectors. Depending on your order you will find one of the following connectors on your SBC:

1. a 2-pole PHOENIX MINI COMBICON base strip 3.5 mm connector (X2) suitable for a single 5 V supply voltage, or
2. a 6-pole WAGO male header (X2) to attach the Power Module for phyBOARDs (PEB-POW-01) which provides connectivity for 12 V – 24 V

The required current load capacity for all power supply solutions depends on the specific configuration of the phyCORE mounted on the phyBOARD-Mira, the particular interfaces enabled while executing software, as well as whether an optional expansion board is connected to the carrier board.



PHOENIX base strip



WAGO male header 6-pole

Figure 4: Power Supply Connectors(X2), Backup Voltage Connector (X29)

3.2.1.1.1 PHOENIX 2-pole MINI COMBICON Base Strip (X2)

The permissible input voltage is +5 V DC if your SBC is equipped with a 2-pole PHOENIX MINI COMBICON base strip. A 5 V adapter with a minimum supply of 1.5 A is recommended to supply the board via the 2-pole base strip.

Figure 4 and the following table show the pin assignment.

Pin	Signal	Description
1	VCC5V_IN	+5 V power supply
2	GND	Ground

Table 6: Pin Assignment of the 2-pole PHOENIX MINI COMBICON Base Strip at X2

3.2.1.1.2 WAGO 6-pole Male Header (X2)

If a WAGO 6-pole male header is mounted on your board (*Figure 2* and *Figure 4*) your board is prepared to connect to a phyBOARD Power Module (PEB-POW-01), or a custom power supply circuitry. The ordering number of the mating connector from WAGO is:
EAN 4045454120610.

Use of the 6-pole connector has the following advantages:

- Higher and wider operate range of the input voltage
- External scaling potential to optimize the electrical output current, by use of customized power modules which match the requirements
- 5 V, 3.3 V and backlight power supply

Pin assignment of the 6–pole WAGO connector:

Pin	Signal	Description
1	VCC5V_IN	+5 V power supply
2	GND	Ground
3	VCC3V3_PMOD	+3.3 V power supply
4	VCC_BL	Backlight power supply (input voltage of power module)
5	PMOD_PWRGOOD	Power good signal (connected to reset nRESET_IN)
6	nPMOD_PWRFAIL	Power fail signal

Table 7: Pin Assignment of the 6-pole WAGO Connector at X2

A detailed description of the Power Module for phyBOARDs can be found in the Application Guide for phyBOARD Expansion Boards (L-793e).

3.2.1.2 Power LED D2

The red LED D2 next to expansion connector X17 (*Figure 2*) indicates the presence of the 3.3 V supply voltage generated from the 5 V input voltage.

3.2.1.3 VBAT, RTC and X29

The phyBOARD-Mira features an external RTC at U12 (*Figure 2*) in addition to the RTC of the Power Management IC mounted on the phyCORE-i.MX 6 module. It is used for real-time or time-driven applications. To backup the RTC a Gold cap (C90) (*Figure 2*) is placed on the phyBOARD-Mira. Alternatively the 2-pole pin header X29 can be used to connect an external battery (max. 3.6 V) to VBAT to feed in the backup voltage.

Figure 4 and the following table show the pin assignment of X29.

Pin	Signal	Description
1	VBAT	Backup Battery voltage
2	GND	Ground

Table 8: Pinout of the 2-pole pin header X29

The backup voltage source (either Gold cap at C90, or external battery via X29) supplies the external RTC at U12. It can also be connected to the backup voltage pin VBAT (A5) of the phyCORE-i.MX 6 via resistor R117[2] to supply some critical registers and the RTC of the Power Management IC on the SOM when the primary system power, VCC5V_IN, is removed. The backup supply for only the RTC (U12) lasts approximately 11½ days.

2: Resistor R117 is not mounted in the standard configuration of the phyBOARD-Mira.

3.2.2 UART Connectivity (X17 and X23)

The i.MX 6 SOM supports up to 5 so called UART units. On the phyBOARD-Mira TTL level signals of UART1 and UART2 (the standard console) are routed to expansion connector X17. UART3 is available at pin header connector X23 at RS-232 level, or optionally at RS-485 level[3].

	The Evaluation Board (PEB-EVAL-01) delivered with the kit plugs into the expansion connector and allows easy use of the standard console (UART2) which is required for debugging. Please find additional information on the Evaluation Board in Application Guide for phyBOARD Expansion Boards (L-793e).

Further information on the expansion connector can be found in *section 4.6.8.*

Pin header connector X23 is located next to the Ethernet connector (*Figure 5)* and provides the UART3 signals of the i.MX 6 at RS-232, or RS-485 level. The serial interface is intended to be used as data terminal equipment (DTE) and allows for a 5-wire connection including the signals RTS and CTS for hardware flow control. *Table 9* shows the signal mapping of the RS-232 and RS-485 level signals at connector X23.

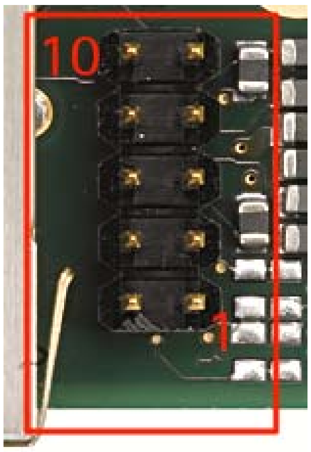


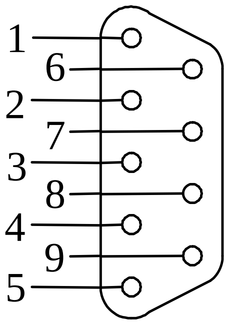
Figure 5: RS-232 or RS-485 Interface Connector X23

Pin	Signal	Pin	Signal
10	NC	9	GND
8	X_UART3_RS485_B[3]	7	X_UART3_RS485_A[3]
6	X_UART3_CTS_RS232	5	X_UART3_TXD_RS232
4	X_UART3_RTS_RS232	3	X_UART3_RXD_RS232
2	NC	1	NC

Table 9: Pin Assignment of RS-232 /RS-485 Interface Connector X23

3: The standard kit comes with an RS-232transceiver installed, if you need an RS-485 interface at X23 instead, please contact our sales team.

An adapter cable is included in the phyBOARD-Mira i.MX 6 Kit to facilitate the use of the UART3 interface. The following figure shows the signal mapping of the adapter.



Pin 2:	RxD-RS232
Pin 7:	RTS-RS232
Pin 3:	TxD-RS232
Pin 8:	CTS-RS232
Pin 4:	A-RS485[3]
Pin 9:	B-RS485[3]
Pin 5:	GND

Figure 6: RS-232 / RS-485 Connector Signal Mapping

3.2.2.1 Software Implementation

Only /dev/ttymxc1 for UART2 and /dev/ttymxc2 for UART3 have been implemented within the BSP.

The standard console UART2 is accessible as /dev/ttymxc1 and is mainly used for debugging and control of software updates.

UART3 can be accessed as /dev/ttymxc2 . It is intended for user applications.

Usage of */dev/ttymxc0* for UART1 requires changing of the pin muxing and additional software development.

3.2.3 Ethernet Connectivity (X4)

The Ethernet 0 interface of the phyBOARD-Mira is accessible at the RJ45 connector X4.

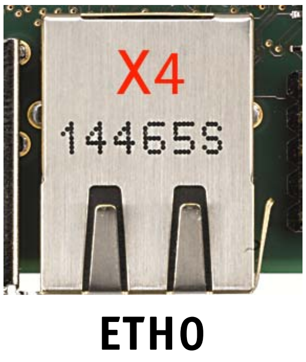


Figure 7: *Ethernet Interface at Connectors X4*

The standard phyBOARD-Mira is equipped with an RJ45 connector supporting a 10/100Base-T network connection. Optionally, it can be ordered with the phyCORE-i.MX 6 module configured for also 1000 Mbps Ethernet and a different RJ45 connector allowing to use the phyBOARD-Mira in a 10/100/1000Base-T network[4]. The LEDs for LINK (green) and SPEED (yellow) indication are integrated in the connector. The Ethernet transceiver supports Auto MDI-X, eliminating the need for the consideration of a direct connect LAN cable, or a cross-over path cable. They detect the TX and RX pins of the connected device and automatically configure the PHY TX and RX pins accordingly.

3.2.3.1 Software Implementation

The i.MX6 SOM features Ethernet, which is being used to provide the network interface eth0 with static IP 192.168.3.11 by default. The interface offers a standard Linux network port at RJ45 connector X4 which can be programmed using the BSD socket interface.

[4]: Please contact our sales team for more information.

3.2.4 USB Connectivity (X5 and X6)

The phyBOARD-Mira provides one USB host and one USB OTG interface.

USB1 is accessible at connector X5 (USB Micro-AB) and is configured as USB OTG. USB OTG devices are capable to initiate a session, control the connection and exchange host and peripheral roles between each other. This interface is compliant with USB revision 2.0.

USB2 is accessible at connector X6 (USB Standard-A) and is configured as USB host.

Both connectors are on the top side of the phyBOARD-Mira and located next to each other (*Figure 8*).

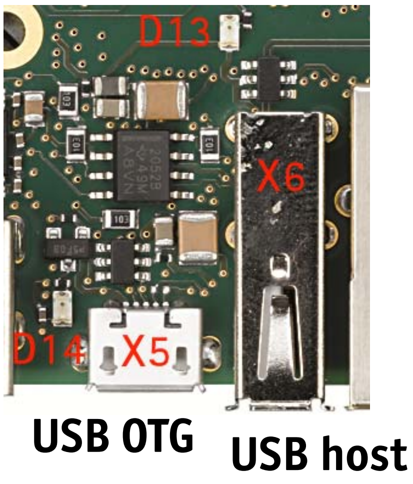


Figure 8: Components supporting the USB Interfaces

LED D14 displays the status of X_USB1_VBUS and LED D13 the status of X_USB2_VBUS.

Numerous jumpers allow to configure the USB interfaces according to your needs. Please refer to *section 4.6.2* for more information.

3.2.4.1 Software Implementation

3.2.4.1.1 USB Host

The i.MX 6 CPU embeds an USB 2.0 EHCI controller that is also able to handle low and full speed devices (USB 1.1).

The BSP includes support for mass storage devices and keyboards. Other USB related device drivers must be enabled in the kernel configuration on demand.

Due to *udev*, connecting various mass storage devices get unique IDs and can be found in */dev/disk/by-id*. These IDs can be used in */etc/fstab* to mount different USB memory devices in a different way.

3.2.4.1.2 USB OTG

In order to activate USB OTG support you need to load an appropriate module with *modprobe*, for example the mass storage gadget *g_mass_storage*, which lets the phyBOARD-Mira behave like a regular USB flash drive including a MS-DOS partition table:

```
dd if=/dev/zero of=/tmp/file.img bs=1M count=64
modprobe g_mass_storage file=/tmp/file.img
```

After connecting the USB OTG port to the USB host port of any other Linux computer, use the following command in order to create a partition table (replace *sdc* by the device the computer really shows):

```
sudo fdisk /dev/sdc
```

Use the following sequence of fdisk commands to create a new partition table and a single FAT32 partition on the USB flash drive: Press `o` to create a empty partition table, press `n` to create a new partition, press `p` to select the partition type as primary, press `1` to use the first partition on the device, press the enter key to select the first available sector and press the enter key again to use the last available sector as the end of the new partition. Now press `t` to toggle the partition type and press `c` to select the partition type W95 FAT32. After that you can write the new partition table to the USB stick by pressing `w`.

To create a FAT32 filesystem execute

```
sudo mkfs.vfat /dev/sdc1
```

You should be able to mount the newly formatted USB flash drive with

```
sudo mount /dev/sdc1 /media
```

To use USB OTG as host you can use the modules g_ether and g_serial. Before you do that unload the previous loaded module e.g.:

```
modprobe -r g_mass_storage
```

To start the Ethernet gadget on the device execute

```
modprobe g_ether
```

You now have an additional Ethernet interface. You can verify this with

```
ifconfig usb0
```

The *g_serial* gadget is used the same way as *g_ether*, but it creates an additional serial interface */dev/ttyGS0*.

3.2.5 CAN Connectivity (X3, JP2)

The Controller Area Network (CAN) bus offers a low-bandwidth, prioritized message fieldbus for serial communication between microcontrollers. The Flexible Controller Area Network (FLEXCAN) module of the iMX 6 implements the CAN protocol according to the CAN 2.0B protocol specification. The first interface (FLEXCAN1) of the Flexible Controller Area Network is accessible at connector X3 (2×5 pin header, 2.54 mm pitch).

Jumper JP2 can be installed to add a 120 Ohm termination resistor across the CAN data lines if needed.

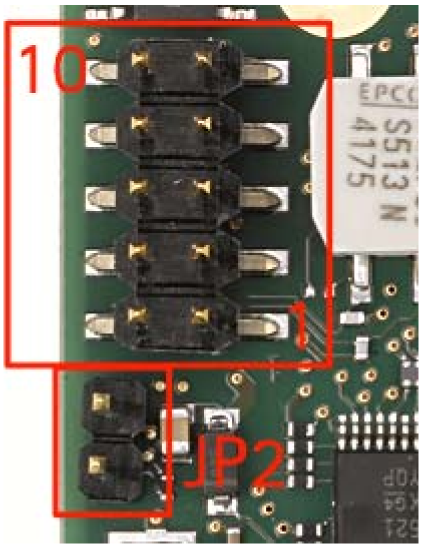


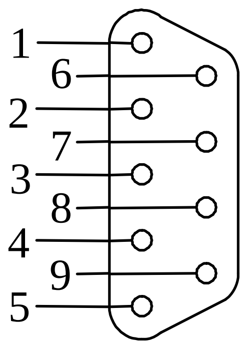
Figure 9: Components supporting the CAN Interface

Table 10 below shows the signal mapping of the CAN1 signals at connector X3.

Pin	Signal	Pin	Signal
10	NC	9	Shield
8	NC	7	NC
6	NC	5	GND
4	X_CANH	3	X_CANL
2	GND	1	NC

Table 10: Pin Assignment of CAN Connector X3

An adapter cable is included in the phyBOARD-Mira i.MX 6 Kit to facilitate the use of the CAN interface. The following figure shows the signal mapping of the adapter.



Pin 6:	GND
Pin 2:	X_CANL
Pin 7:	X_CANH
Pin 3:	GND
Pin 5:	Shield

Figure 10: CAN Connector Signal Mapping

As an alternative option the TTL level signals of FLEXCAN1 can be also routed to expansion port X17 (*4.6.3*).

Depending on the muxing options a second CAN interface (FLEXCAN2) is available at expansion port X17 (*4.6.3*).

3.2.5.1 Software Implementation

Unfortunately, CAN was not designed with the ISO/OSI layer model in mind, so most CAN APIs available throughout the industry do not support a clean separation between the different logical protocol layers, as for example known from Ethernet.

CAN is supported by drivers using the proposed Linux standard CAN framework ”Socket-CAN”, which extends the BSD socket API concept towards CAN bus.

Because of that, using this framework, the CAN interfaces can be programmed with the BSD socket API and the Socket-CAN interface behaves like an ordinary Linux network device, with some additional features special to CAN. Thus, for example, you can use

```
ifconfig -a
```

to see if the interface is up or down, but the given MAC and IP addresses are arbitrary and obsolete.

To get the information on *can0* (which represents i.MX 6’s FLEXCAN1 routed to X3) type

```
ifconfig can0
```

The information will look like the following

```
can0     Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
         inet addr:127.42.23.180  Mask:255.255.255.0
         UP RUNNING NOARP  MTU:16  Metric:1
         RX packets:0 errors:0 dropped:0 overruns:0 frame:0
         TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:10
         RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
         Interrupt:55
```

The output contains a standard set of parameters also shown for Ethernet interfaces, so not all of these are necessarily relevant for CAN (for example the MAC address). The following output parameters contain useful information:

Field	Description
can0	Interface Name
NOARP	CAN cannot use ARP protocol
MTU	Maximum Transfer Unit
RX packets	Number of Received Packets
TX packets	Number of Transmitted Packets
RX bytes	Number of Received Bytes
TX bytes	Number of Transmitted Bytes
errors...	Bus Error Statistics

You can send messages with *cansend* or receive messages with *candump*:

```
cansend can0 123#45.67
candump can0
```

See `cansend --help` and `candump --help` messages for further information on options and usage.

The CAN configuration is done in the *systemd* configuration file */lib/systemd/system/can0.service.*

For a persistent change in the root filesystem you will find the configuration in the BSP under *meta-yogurt/recipes-core/systemd/systemd/can0.service* .

3.2.6 Secure Digital Memory Card/MultiMedia Card (X22)

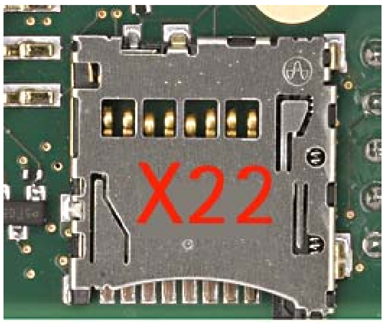


Figure 11: SD/MM Card Interface at Connector X22 (back side)

The phyBOARD-Mira provides a standard microSDHC card slot at X22 for connection to SD/MMC interface cards. It allows easy and convenient connection to peripheral devices like SD- and MMC cards. Power to the SD interface is supplied by inserting the appropriate card into the SD/MMC connector, which features card detection, a lock mechanism and a smooth extraction function by Push-in/-out of card.

DIP switch S2 allows to toggle between NAND boot and boot from SD card. In order to boot from SD card S2 must be switched ON (refer to *section 3.2.14* for further information).

3.2.6.1 Software Implementation

The driver for Micro Secure Digital Cards and Micro Multi Media Cards is accessible as for general purpose block devices. These devices can be used in the same way as any other block devices.

⚠	This kind of devices are hot pluggable, but you must pay attention not to unplug the device while it is still mounted. This may result in data loss.

After inserting an MMC/SD card, the kernel will generate new device nodes in */dev*. The full device can be reached via its */dev/mmcblk0* device node, MMC/SD card partitions will occur in the following way:

```
/dev/mmcblk0p<Y>
```

<Y> counts as the partition number starting from 1 to the max count of partitions on this device.

⚠	These partition device nodes will only occur if the card contains a valid partition table (”hard disk” like handling). If it does not contain one, the whole device can be used for a file system (”floppy” like handling). In this case */dev/mmcblk0* must be used for formatting and mounting.

The partitions can be formatted with any kind of file system and also handled in a standard manner, e.g. the *mount* and *umount* command work as expected.

	The cards are always mounted as being writable. Setting of write-protection of MMC/SD cards is not recognized.

3.2.7 PCIe Connectivity (X7)

The 1-lane PCI express interface of the phyBOARD-Mira i.MX 6 provides PCIe Gen. 2.0 functionality which supports 5 Gbit/s operations. Furthermore the interface is fully backwards compatible to the 2.5 Gbit/s Gen. 1.1 specification. Various control signals are implemented with GPIOs[5]. The PCIe interface is brought out at the Mini PCIe connector X7 shown in the figure below.

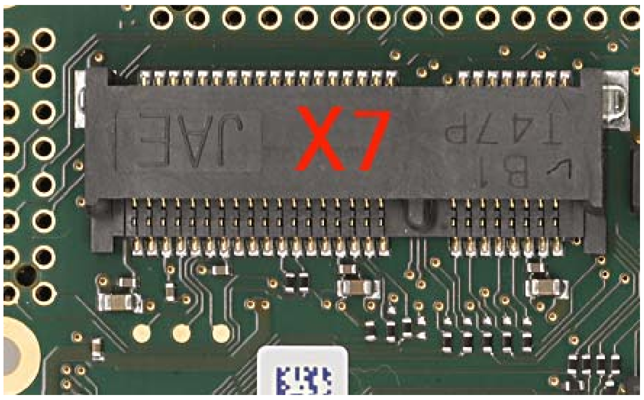


Figure 12: PCIe Interface at Connector X7

The SIM/UIM[6] card signals of a connected PCIe module can be made available at expansion connector X17. Please refer to *Table 28:* PHYTEC Expansion Connector X17 (continued) for more information about the jumper settings.

Soldering jumpers allow to connect the USB host interface to the Mini PCIe connector X7 (*Table 18*).

Please refer to *section 4.6.6* for in-depth information such as pin assignment and signals used to implement special features of the Mini PCIe interface.

3.2.7.1 Software Implementation

To use a specific PCIe device you mostly have to enable the corresponding kernel module in the kernel configuration. You can use the shell command `lspci` to show all recognized PCIe devices.

5: For these pins there is no explicit multiplexing done in the BSP and they are configured with the i.MX6's default values. No further configuration is done and must be implemented by the developer.
6: User Identity Module (UIM) signals

3.2.8 Camera Connectivity (X10)

The SOM on the phyBOARD-Mira provides two types of camera interfaces (parallel and MIPI CSI-2).

Generally, the parallel port can be expanded on the carrier board in three ways:
1. according to the phyCAM-P camera interface standard
2. according to the as phyCAM-S+ camera interface standard
3. as interface for customer parallel cameras

On the phyBOARD-Mira the parallel Camera_0 interface is brought out as phyCAM-S+ camera interface (*Figure 13*) at connector X10 (*Figure 14*). Information on the phyCAM-S+ standard can be found in the phyCAM-manual (L-748).

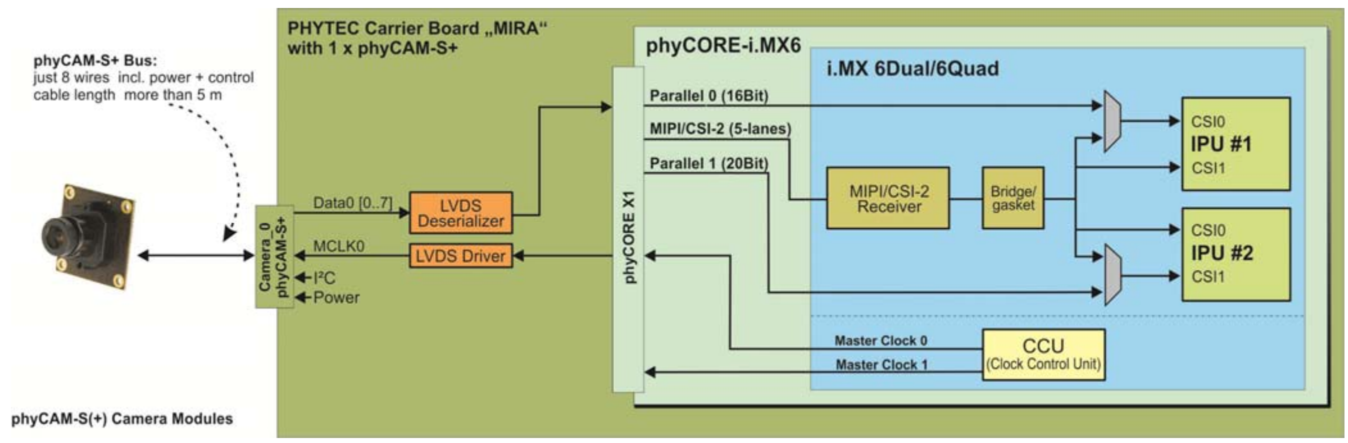


Figure 13: *phyCAM-S+ Camera Interface on the phyBOARD-Mira*

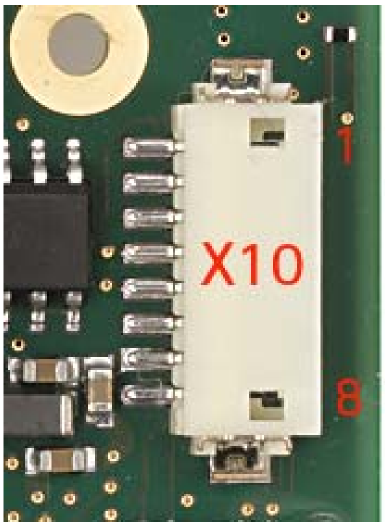


Figure 14: *Camera Interface (phyCAM-S+) at Connector X10*

The table below shows the pinout of connector X10.

Pin #	Signal Name	Description
1	Camera0_L0+	LVDS Input+
2	Camera0_L0-	LVDS Input-
3	Camera0_RXCLK-	LVDS Clock-
4	I2C_SDA_CAMERA	I^{2}C Data
5	I2C_SCL_CAMERA	I^{2}C Clock
6	Camera0_RXCLK +	LVDS Clock+
7	VCC_CAMERA0	Power supply camera (3.3 V)
8	GND	Ground

Table 11: PHYTEC Camera Connector X10

3.2.8.1 Software Implementation

The BSP for the phyBOARD-Mira supports the VM-010-BW-LVDS camera module.

To make use of the camera in our BSP the *gstreamer*framework is implemented in the root filesystem. This multimedia framework allows to create a variety of media-handling components, including simple audio playback, audio and video playback, recording, streaming and editing.

Some example scripts, to get an output from the camera are provided in the directory */home/root/gstreamer_examples*. For example a script to play back a black and white video:

```
cd /home/root/gstreamer_examples
./bwcam-fbdev_640x480.sh
```

Please refer to
ftp://ftp.phytec.de/pub/ImageProcessing/phyBOARD-Mira-i.MX6_linux_PD15.2.0/ to get more information how to make use of *gstreamer* and how to add other phyCAM modules to the BSP.

3.2.9 HDMI Connectivity (X28)

The phyBOARD-Mira i.MX 6 provides a High-Definition Multimedia Interface (HDMI) which is compliant to HDMI 1.4a, DVI 1.0, HDCP 1.4. It supports a maximum pixel clock of up to 340 MHz for up to 720p at 100 Hz and 720i at 200 Hz, or 1080p at 60 Hz and 1080i/720i at 120 Hz HDTV display resolutions, and a graphic display resolution of up to 2048x1536 (QXGA). Audio streams reach a sampling rate of up to 192 kHz. Please refer to the *i.MX 6 Applications Processor Reference Manual* for more information.

The HDMI interface is brought out at a standard HDMI type A connector (X28) on the phyBOARD-Mira i.MX 6 and comprises the following signal groups: three pairs of data signals, one pair of clock signals, an I²C bus which is exclusively for the HDMI interface, the Consumer Electronics Control (CEC) signal and the hot plug detect (HPD) signal. All signals are routed from the phyCORE-Connectore to the HDMI receptacle through an HDMI Transmitter Port Protection and Interface Device. This device provides ESD protection and includes level shifting to shift the I²C interface signals and the hot plug detect signal from IO voltage (VCC3V3) to 5 V. The hot plug detect signal is pulled down to ground at the output of the protection device.

⚠	Ensure that all module connections are not to exceed their expressed maximum voltage or current. Maximum signal input values are indicated in the corresponding controller User's Manual/Data Sheets. As damage from improper connections varies according to use and application, it is the user's responsibility to take appropriate safety measures to ensure that the module connections are protected from overloading through connected peripherals.

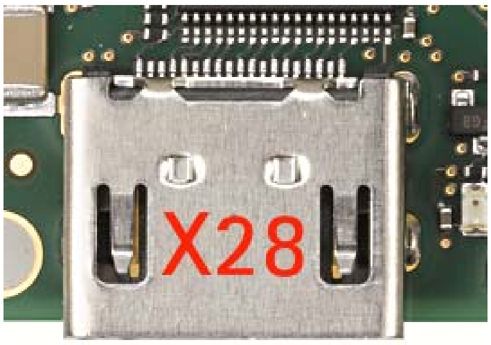


Figure 15: HDMI Interface Connector X28

3.2.9.1 Software Implementation

This driver gains access to the display via device node */dev/fb0* for HDMI. For this BSP the HDMI Interface Connector (X28) is default.

A simple test of the framebuffer feature will show various pictures on the display can be run with:

```
fbtest
```

3.2.10 LVDS Display Connectivity (X9)

Meanwhile there are a few LVDS displays on the market with kind of standardized interfaces. The LVDS display connector X9 is intend to connect these displays with screen diagonals from 7″ up to 12.1″ at different resolutions.

The display connector X9 is a 20-pole receptacle with 1 mm pitch.

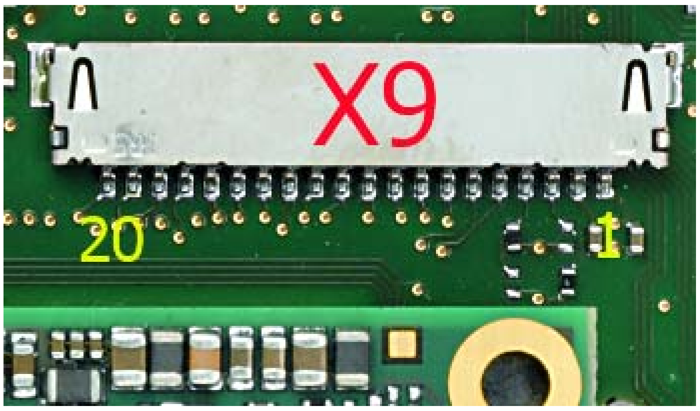


Figure 16: LVDS Display Connector X9

Various solder jumpers and resistors allow to adapt the pin assignment to different displays. If your kit includes a display all jumpers and resistors are preset corresponding to the display.

For more detail information about the pin assignment and how to set up the right configuration for a custom display refer to *section 4.6.5*.

3.2.10.1 Software Implementation

The BSP is already prepared for use with the 10″ AUOG104STN01 display. To activate the LVDS display you must enter the barebox and type the following command:

```
edit /env/config-expansions
```

Following line must be added at the end:

```
. imx6qdl-mira-enable-lvds
```

Save the file with ***CTRL+D*** and ***restart*** the phyBOARD-Mira**.**

This driver gains access to the display via device node /dev/fb0. A simple test of the framebuffer feature can then be run with:

```
fbtest
```

This will show various pictures on the display.

3.2.11 Backlight and Display Control Connector (X8)

In order to support a backlight for the LVDS display X8 provides the supply voltages and control signals necessary.

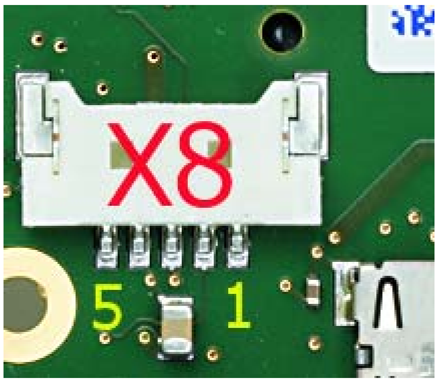


Figure 17: Backlight and Display Control Connector X8

The following table shows the pinout of the 1.25 mm pitch Molex PanelMate^{TM} header.

Pin #	Signal name	ST	SL	Description
1	NC	-	-	Not connected
2	X_PWM1_OUT	O	3.3 V	PWM brightness output
3	X_LVDS_DISP_EN	O	5.0 V	3.3 V power supply display
4	GND	-	-	Ground
5	VCC_BL	O	NS	Backlight power supply

Table 12: Display Power Connector X8 Signal Description

3.2.11.1 Software Implementation

The PWM signal at pin 2 allows to change the brightness of the display attached to the display control connector.

To change the brightness switch to the appropriate directory by entering:

```
cd /sys/class/backlight/backlight.*
```

Now you can enter a new value for the brightness with:

```
echo value > brightness
```

The *value* can be from 0 to 7.

3.2.12 Touch Screen Connectivity (X13, X21)

As many smaller applications need a touch screen as user interface, different provisions are made to connect as well 4- wire resistive touch screens as various capacitive touch screens. A touch screen can be connected either to the dedicated touch connector X21, or to the touch signal inputs of A/V connector X13.

The following table summarizes the different possibilities available to connect a touch screen to one of these connectors.

Connector	Touch Screen
X21	4-wire resistive touch
X13 (pins 9 – 12)	4-wire resistive touch
X13 (pins 15 – 16)	Capacitive touch screen with I^2C interface

Table 13: Touch Screen Connectivity X13, X21

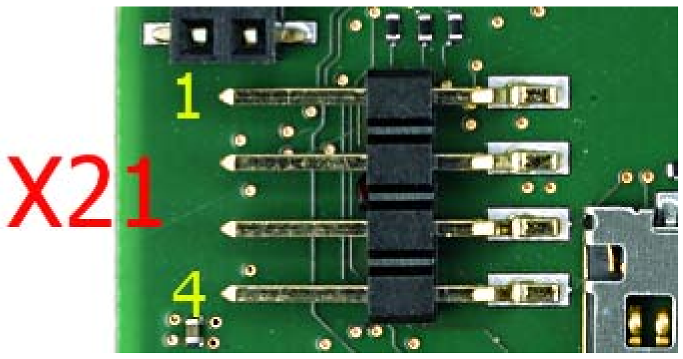


Figure 18: Touch Screen Connectors X21

3.2.12.1 Software Implementation

The resistive touch delivered with the 10" AUOG104STN01 display is implemented in the BSP and will be working when you do the steps from chapter *3.2.10.1*

A simple test of this feature can be run with

```
ts_calibrate
```

to calibrate the touch and

```
ts_test
```

to start a simple application using this feature.

3.2.13 Multicolor (RGB) LED (D6)

The phyBOARD-Mira provides one multicolor (RGB) LED (D6) (*Figure 2*) for user applications. The colors can be controlled with the following GPIOs

Color	Signal	Description
Red	X_CSI0_DAT4	GPIO5_IO22 of the i.MX 6
Green	X_CSI0_DAT5	GPIO5_IO23 of the i.MX 6
Blue	X_CSI0_DAT6	GPIO5_IO24 of the i.MX 6

Table 14: Multicolor LED Configuration

As an option the LED can be controlled with the LED dimmer IC at U6[7]. The LED dimmer can be accessed via I2C1 at address 0X62 and dynamically controls the LED with PWM signals.

3.2.13.1 Software Implementation

The Multicolor LED can be controlled through the *sysfs* interface. There is a folder for each color in */sys/class/leds: blue:user3* for blue, *red:user1* for red and *green:user2* for green.

Each color can be configured independently. To switch on the green LED write *255* in the file *brightness*:

```
echo 255 > /sys/class/leds/blue:user3/brightness
```

To switch it off again, write *0* to the file *brightness*:

```
echo 0 > /sys/class/leds/blue:user3/brightness
```

It is also possible to connect each color to a predefined trigger. To see all available triggers read from the file *trigger* with `cat /sys/class/leds/blue:user3/trigger`. The output may be:

```
none rc-feedback nand-disk [mmc0] timer oneshot heartbeat backlight gpio
```

For the color blue the trigger *mmc0* is used by default.

7: The phyBOARD-Mira in the standard kit is not equipped with the LED dimmer. Please contact our sales team for more information.

3.2.14 Boot Mode (S2)

The pyhBOARD-Mira has two defined boot sequence which can be selected with DIP switch S2.

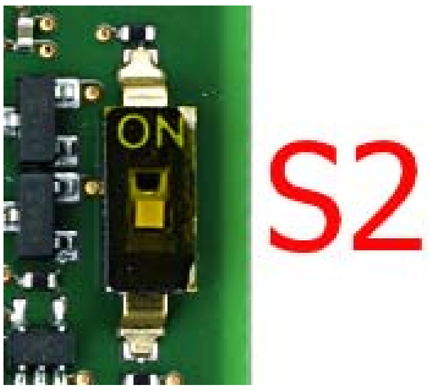


Figure 19: Boot Switch (S2)

Boot Mode	Description
Boot mode 1 (S2 = OFF)	Boot from NAND
Boot mode 2 (S2 = ON)	Boot from SD/MMC 1

Table 15: Boot Switch Configuration (S2)

3.2.15 System Reset Button (S1)

The phyBOARD-Mira is equipped with a system reset button at S1. Pressing this button will toggle the X_nRESET pin (X1D32) of the phyCORE SOM low, causing the module to reset.

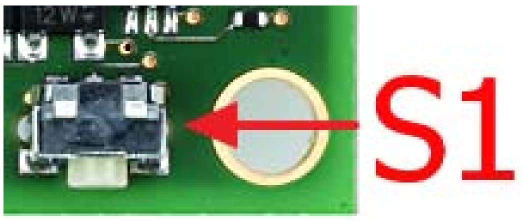


Figure 20: System Reset Button S1

3.2.16 Audio/Video connectors (X13 and X14)

The Audio/Video (A/V) connectors X13 and X14 provide an easy way to add typical A/V functions and features to the phyBOARD-Mira. Standard interfaces such as parallel display, I^{2}S and I^{2}C as well as different supply voltages are available at the two A/V female dual entry connectors. Special feature of these connectors are their connectivity from the bottom or the top.

For further information of the A/V connectors see *chapter 4.6.7*. Information on the expansion boards available for the A/V Connectors can be found in the Application Guide for phyBOARD Expansion Boards (L-793e).

3.2.17 Expansion connector (X17)

The expansion connector X17 provides an easy way to add other functions and features to the phyBOARD-Mira. Standard interfaces such as JTAG, UART, MMC, SPI and I^{2}C as well as different supply voltages and some GPIOs are available at the expansion female connector.

For further information of the expansion connector and the pinout see *chapter 4.6.8*. Information on the expansion boards available for the expansion connector can be found in the Application Guide for phyBOARD Expansion Boards (L-793e).

3.2.18 Addressing the RTC

The RTC RV-4162-C7 on the phyBOARD-Mira can be accessed as */dev/rtc0*. It also has the entry */sys/class/rtc/rtc0* in the sysfs file system, where you can, for example, read the *name*.

Date and time can be manipulated with the *hwclock* tool, using the *-w* (systohc) and *-s* (hctosys) options. For more information about this tool refer to the manpage of *hwclock*.

To set the date first use *date* (see *man date* on the PC) and then run *hwclock -w -u* to store the new date into the RTC.

	As the RTC of the i.MX 6 is not supplied with a backup voltage, it is recommended to use the external RTC on the phyBOARD-Mira, as described above.

3.2.19 CPU Core Frequency Scaling

The phyBOARD-Mira i.MX 6 supports dvfs (dynamic voltage and frequency scaling). Several different frequencies are supported. Type

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies
```

to get a complete list. The result will be:

```
996000 792000 396000
```

The voltages are scaled according to the setup of the frequencies.

You can decrease the maximum frequency (e.g. to 792000)

```
echo 792000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
```

or increase the minimum frequency (e.g. to 792000)

```
echo 792000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq
```

Asking for the current frequency is done with:

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
```

So called governors are selecting one of these frequencies in accordance to their goals, automatically. The governors available can be listed with the following command:

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_governors
```

The result will be:

```
conservative userspace powersave ondemand performance
```

conservative	is much like the *ondemand* governor. It differs in behaviour in that it gracefully increases and decreases the CPU speed rather than jumping to max speed the moment there is any load on the CPU.
userspace	allows the user or userspace program running as root to set a specific frequency (e.g. to 792000). Type: `echo 792000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed`
powersave	always selects the lowest possible CPU core frequency.
ondemand	switches between possible CPU core frequencies in reference to the current system load. When the system load increases above a specific limit it

increases the CPU core frequency immediately. This is the default governor when the system starts up.

performance always selects the highest possible CPU core frequency.

In order to ask for the current governor, type

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

and you will normally get:

```
ondemand.
```

Switching over to another governor (e.g. *userspace*) is done with:

```
echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

For more detailed information about the govenors refer to the *Linux* kernel documentation in:

Documentation/cpu-freq/governors.txt.

3.2.20 Using the Pre-installed Qt Demo Applications

Qt provided by *Digia* is very commonly used for embedded systems and it is supported by this BSP. Please visit http://www.qt.io in order to get all the documentation that is available about this powerful cross-platform GUI toolkit.

Some demos that show the capabilities of Qt version 5.3.2 are included in the BSP. Use of an HDMI monitor requires an USB mouse connected to the board in order to test the demos.

By default the demo will be started automatically after booting (see *section 2.3.1* for information how to stop the demo, or how to remove it from the autostart).

4 System Level Customizing

4.1 About this Section

This section addresses advanced developers who want to configure, build and install a new kernel and file system, design custom expansion boards, or display adapters. It includes the following information:

- Step by step instructions on how to configure, build and install a new kernel and file system using the Live System
- A description how to update the Bootloader and how to write the Kernel and Root File System into the Flash from within your own *Linux* system
- Detailed information on the different interfaces and features of the phyBOARD-Mira at a system level
- How to set up your own Linux host PC

4.2 Software Overview

In *chapter 2* you have learned how to work with *Eclipse* and *Qt Creator*. The following section shows you how to work with the *Yocto Project*.

The *Yocto Project* is an open source collaboration project. The build system used by the Yocto Project is based on the *OpenEmbedded* Project. With this build system you can construct complete Linux images for different platforms and architectures. The *Yocto Project* includes components to design, develop, build, debug, simulate and test your software.

4.3 Getting Started with the BSP

In this chapter you will go through some software topics. First of all we take a look at the BSP and learn how to add a package. After compiling the new images, you will learn how to write the newly created kernel and root file system into the target's flash memory and how to start the images.

4.3.1 Working with Yocto

In this chapter of the Application Guide we describe only the basic *Yocto* usage. For in-depth explanations refer to the *Yocto* Reference Manual.

In this part you will learn how to use the Yocto Project and the Phytec BSP. You can find the pre-build BSP in the Live System under */opt/PHYTEC_BSPs/phyBOARD-MIRA/*. All necessary tools and programs are already pre-installed to directly start with *Yocto*.

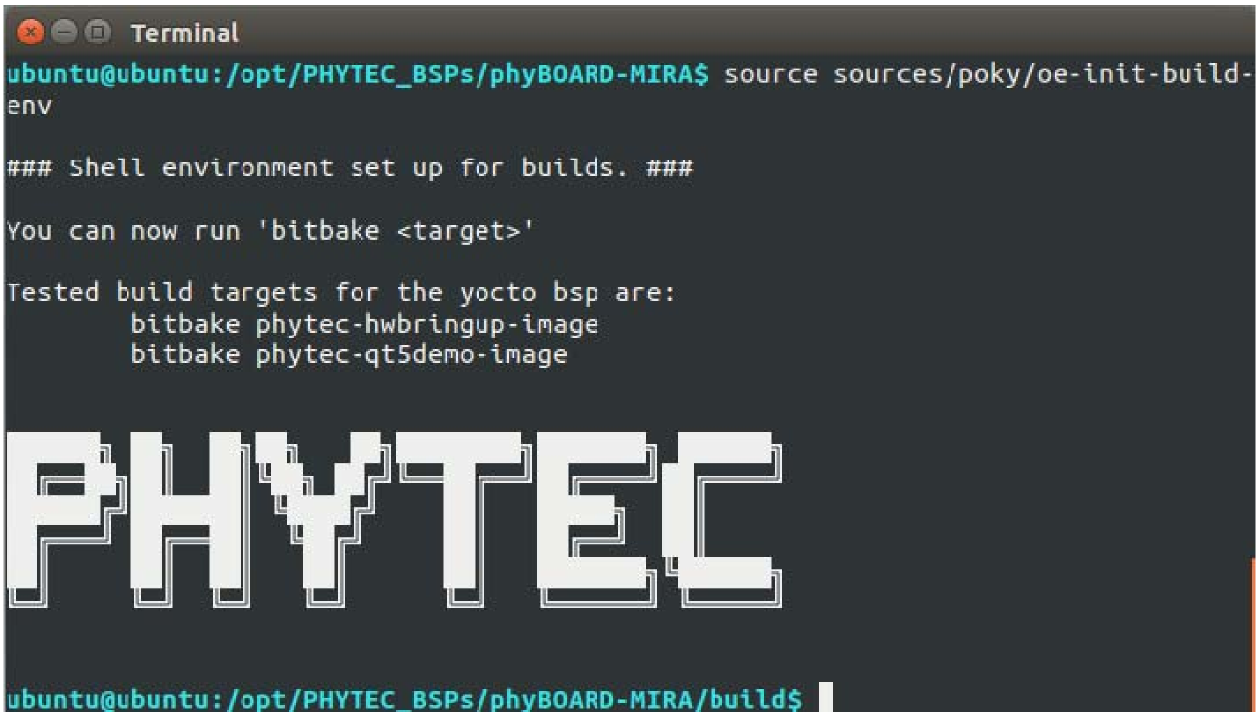
Open a new terminal if you have not already done and change to the directory of the pre-installed BSP:

Terminal

- Click the ***terminal*** icon on your desktop.
- Type the following command to change to the BSP-directory:
  `cd /opt/PHYTEC_BSPs/phyBOARD-MIRA/`

This directory is the start point for our BSP. You will also find further documents in this directory. First we must set the correct environment.

- Enter the following command `source sources/poky/oe-init-build-env`



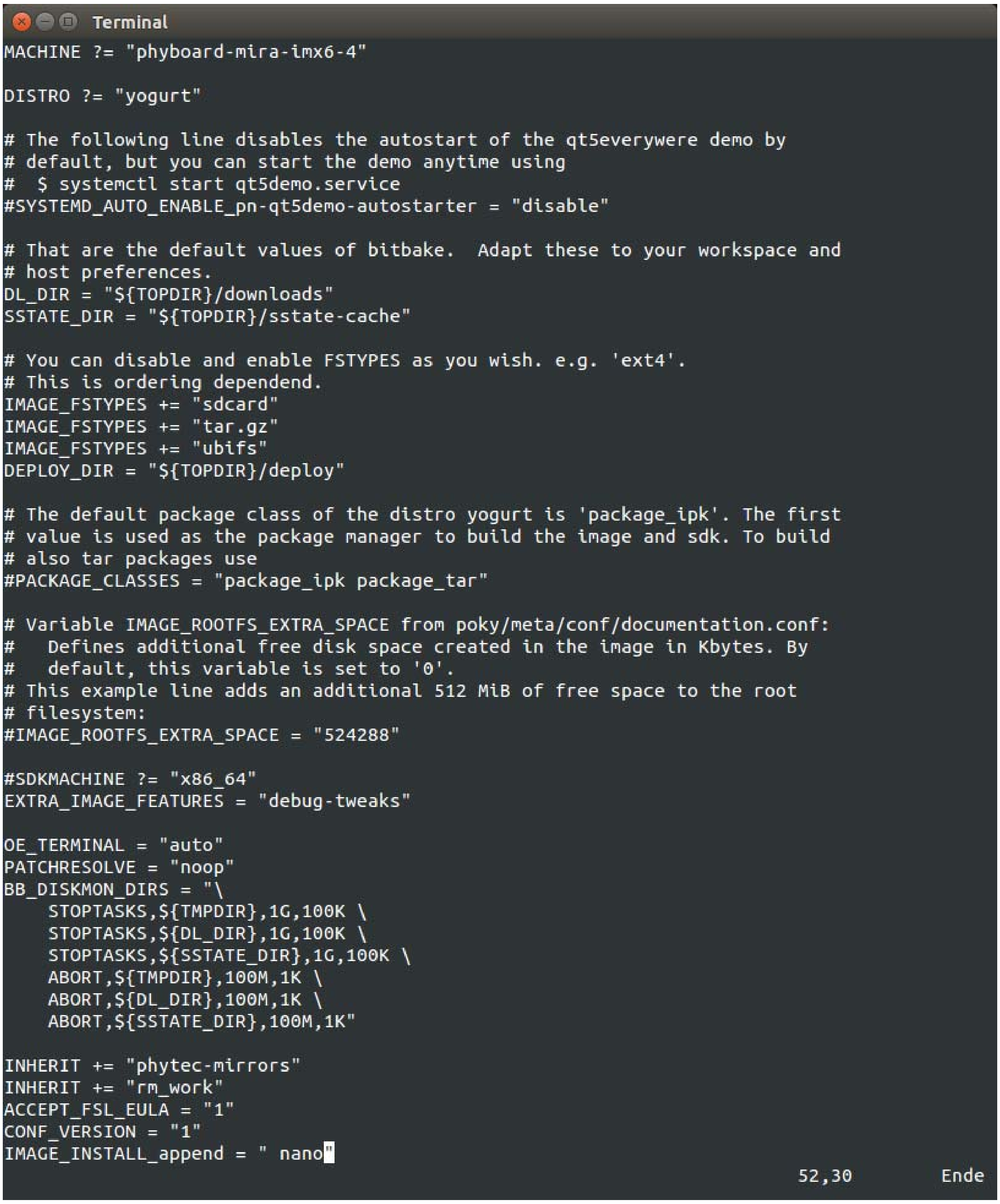
```
Terminal
ubuntu@ubuntu:/opt/PHYTEC_BSPs/phyBOARD-MIRA$ source sources/poky/oe-init-build-
env

### Shell environment set up for builds. ###

You can now run 'bitbake <target>'

Tested build targets for the yocto bsp are:
        bitbake phytec-hwbringup-image
        bitbake phytec-qt5demo-image

PHYTEC

ubuntu@ubuntu:/opt/PHYTEC_BSPs/phyBOARD-MIRA/build$
```

The environment is set and we are now in the build folder. In this example we want to add the *nano editor* to the image. Usually new packages are downloaded from an appropriate server prior to building the image unless they are stored locally. In the Live System the package for the *nano editor* is already pre-downloaded and available locally.

Before we add the package we check if the right configuration is set.

- Open the configuration file with any editor, e.g. *VI,* with the following command: `vi conf/local.conf`
- Check the *MACHINE* variable in this file.

- By default the variable is set to *phyboard-mira-imx6-4*. This is the correct *MACHINE* for the lowcost module.. If you have the PB-01501-001 module set the MACHINE to:
  `phyboard-mira-imx6-3`



```
MACHINE ?= "phyboard-mira-imx6-4"

DISTRO ?= "yogurt"

# The following line disables the autostart of the qt5everywere demo by
# default, but you can start the demo anytime using
#  $ systemctl start qt5demo.service
#SYSTEMD_AUTO_ENABLE_pn-qt5demo-autostarter = "disable"

# That are the default values of bitbake.  Adapt these to your workspace and
# host preferences.
DL_DIR = "${TOPDIR}/downloads"
SSTATE_DIR = "${TOPDIR}/sstate-cache"

# You can disable and enable FSTYPES as you wish. e.g. 'ext4'.
# This is ordering dependend.
IMAGE_FSTYPES += "sdcard"
IMAGE_FSTYPES += "tar.gz"
IMAGE_FSTYPES += "ubifs"
DEPLOY_DIR = "${TOPDIR}/deploy"

# The default package class of the distro yogurt is 'package_ipk'. The first
# value is used as the package manager to build the image and sdk. To build
# also tar packages use
#PACKAGE_CLASSES = "package_ipk package_tar"

# Variable IMAGE_ROOTFS_EXTRA_SPACE from poky/meta/conf/documentation.conf:
#   Defines additional free disk space created in the image in Kbytes. By
#   default, this variable is set to '0'.
# This example line adds an additional 512 MiB of free space to the root
# filesystem:
#IMAGE_ROOTFS_EXTRA_SPACE = "524288"

#SDKMACHINE ?= "x86_64"
EXTRA_IMAGE_FEATURES = "debug-tweaks"

OE_TERMINAL = "auto"
PATCHRESOLVE = "noop"
BB_DISKMON_DIRS = "\
    STOPTASKS,${TMPDIR},1G,100K \
    STOPTASKS,${DL_DIR},1G,100K \
    STOPTASKS,${SSTATE_DIR},1G,100K \
    ABORT,${TMPDIR},100M,1K \
    ABORT,${DL_DIR},100M,1K \
    ABORT,${SSTATE_DIR},100M,1K"

INHERIT += "phytec-mirrors"
INHERIT += "rm_work"
ACCEPT_FSL_EULA = "1"
CONF_VERSION = "1"
IMAGE_INSTALL_append = " nano"
                                                          52,30        Ende
```

- In the same file we can add *nano* by adding the following line at the end of the file (as shown in the previous picture):
  `IMAGE_INSTALL_append = " nano"`
- Save the changes and close the file with ***:x*** and ***Enter.***

You can search for more available packages at:
http://layers.openembedded.org/layerindex/branch/master/recipes/

Be sure that the layer on which the package depends is in our BSP. You can check our integrated layers in the file *conf/bblayers.conf* .
When your needed layer is not in our BSP you must first checkout the layer to the folder /opt/PHYTEC_BSPs/phyBOARD-MIRA/source/ .

Please note that there is no a guarantee that the build process proceeds successfully with all new packages added.

Now you can start building the configured BSP. There are two methods to create the images. You can use either *bitbake phytec-hwbringup-image*, or *bitbake phytec-qt5demo-image*

The first option leads to a faster compilation and smaller images, because all *Qt* relevant features are left out.

- In our example we want to keep the *Qt* relevant features, so we type following command:
  `bitbake phytec-qt5demo-image`

Because of dead links it is possible that an attempt to download a source package fails. PHYTEC cannot guarantee the accessibility of all the web pages that are needed to build a BSP. In case of a failure please check our ftp-server at ftp://ftp.phytec.de/pub/BSP_PACKAGES/EXTERNALS, and download the source package from there into the source directory under */opt/PHYTEC_BSPs/yocto_downloads* manually. After that, please restart the build process by again calling `bitbake phytec-qt5demo-image` or `bitbake phytec-hwbringup-image`.

If the package is also missing on our ftp-server, please send an email to support@phytec.de. We will then take it from our build-server and upload it onto our ftp-server for you.

If you started the Live System from the USB flash drive and did not install it yet, this command might generate some warnings and errors. Nevertheless check if the images were generated. If not, there is not enough memory to build the images and you need to install our Live System now (*section2.1*).

Among other images you will find the kernel named *zImage*, the device tree named *zImage-imx6dl-phytec-mira-rdk-nand.dtb* and the root file system named *phytec-qt5demo-image-phyboard-mira-imx6-4.ubifs* in the directory *deploy/images/phyboard-mira-imx6-4/*.

The three files are only a symbolic link to the correct images with the timestamp of the build date.

4.3.2 Writing the Root Filesystem into the Target's Flash

In this section you will find a description on how to write the newly created root filesystem into the phyBOARD-Mira's flash memory. Before the images can be written into the flash, it must be uploaded to the phyBOARD-Mira from a TFTP server. This will be done from the command line of the boot loader. The partition of the root filesystem will be formatted. Then the image will be downloaded over TFTP to the created flash partition.

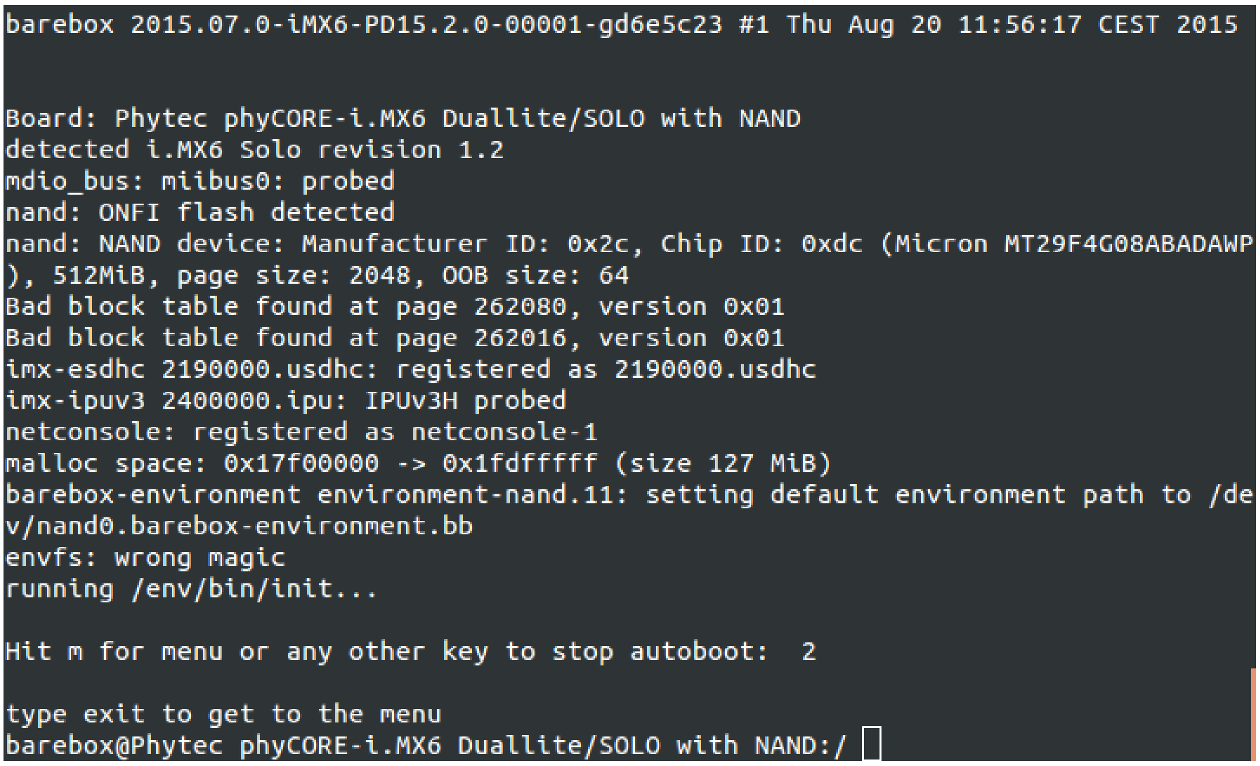
⚠	You should never erase the *Barebox* partition. If this partition is erased, you will not be able to start your target anymore. In such a case, refer to *section 4.4 "Updating the software"*.

⚠	The versions of the *Barebox* and the Linux kernel must match. Therefore the following steps should only be done using the hardware that was shipped together with the Live System and thus already contains the same version of the BSP.

- First open a new terminal window if it is not opened yet. Then change to the directory */opt/PHYTEC_BSPs/phyBOARD-MIRA/build/deploy/images/phyboard-mira-imx6-4/*

- ***Copy*** the root filesystem to the folder */tftpboot*:
  `cp phytec-qt5demo-image-phyboard-mira-imx6-4.ubifs /tftpboot/`

- Open *Microcom* and press the ***RESET*** button on the target.
  You will see the output Hit any key to stop autoboot.
- Press ***any key*** to stop *autoboot*.



```
barebox 2015.07.0-iMX6-PD15.2.0-00001-gd6e5c23 #1 Thu Aug 20 11:56:17 CEST 2015


Board: Phytec phyCORE-i.MX6 Duallite/SOLO with NAND
detected i.MX6 Solo revision 1.2
mdio_bus: miibus0: probed
nand: ONFI flash detected
nand: NAND device: Manufacturer ID: 0x2c, Chip ID: 0xdc (Micron MT29F4G08ABADAWP
), 512MiB, page size: 2048, OOB size: 64
Bad block table found at page 262080, version 0x01
Bad block table found at page 262016, version 0x01
imx-esdhc 2190000.usdhc: registered as 2190000.usdhc
imx-ipuv3 2400000.ipu: IPUv3H probed
netconsole: registered as netconsole-1
malloc space: 0x17f00000 -> 0x1fdfffff (size 127 MiB)
barebox-environment environment-nand.11: setting default environment path to /de
v/nand0.barebox-environment.bb
envfs: wrong magic
running /env/bin/init...

Hit m for menu or any other key to stop autoboot:  2

type exit to get to the menu
barebox@Phytec phyCORE-i.MX6 Duallite/SOLO with NAND:/ 
```

- Check the network settings with following command:
  ```
  ifup eth0
  devinfo eth0
  ```
  The target should return these lines:
  ```
  ipaddr=192.168.3.11
  netmask=255.255.255.0
  gateway=255.255.255.0
  serverip=192.168.3.10
  ```
  If you need to change something, type:
  ```
  edit /env/network/eth0
  ```
  edit the settings, save them by leaving the editor with ***Strg-D***, then type `saveenv` and reboot the board.
- Now we flash the root filesystem. T ype:
  ```
  ubiformat /dev/nand0.root
  ubiattach /dev/nand0.root
  ubimkvol /dev/ubi0 root 0
  cp /mnt/tftp/phytec-qt5demo-image-phyboard-mira-imx6-4.ubifs
  /dev/ubi0.root
  ```
- Enter `boot` to boot the phyBOARD-Mira with the new kernel and root file system.
- After the target has successfully finished booting, type `root` to log in.
- Now we can test the newly installed editor *nano* by trying to open a file with it.
- Enter `nano /etc/profile .`
- Close *nano* by pressing ***CTRL+X***.
- Close *Microcom* after *nano* is closed.

⚠	Troubleshooting: If any problem occurs after writing the kernel or the root filesystem into the flash memory, you can restore the original kernel (*zimage*) from the */tftpboot* directory. The root file system is available the following directory of the Live System: /opt/PHYTEC_BSPs/phyBOARD-MIRA/build/deploy/images/phyboard-mira-imx6-4/phytec-qt5demo-image-phyboard-mira-imx6-4-20150820141326.rootfs.ubifs

🦊	All files are also downloadable from our ftp server, or you can find them on our Linux-phyBOARD-Mira i.MX 6-Kit USB flash drive when you mount it into a running system under */PHYTEC/BSP/*. If you want other versions also check our ftp server: ftp://ftp.phytec.de/pub/Products/

👍	In this section you learned how to prepare the partition of the root filesystem and how to download the root filesystem from a TFTP server into the flash of the target.

4.4 Updating the software using an SD card

If you found a newer BSP on our ftp server ftp://ftp.phytec.de/pub/Products and want to flash it, this chapter shows how to do it. In case that your phyBOARD-Mira i.MX 6 does not start anymore because you damaged its software during the previous chapter, you are right here, too.

⚠	Make sure that the BSP version matches the hardware version of the phyBOARD-Mira-i.MX6. The phyBOARD_Mira is available in a "low cost", or in a "full featured" kit. Please visit http://www.phytec.eu and navigate to “Support” / “FAQ/Download” / “phyBOARDs – Single Board Computer” / “phyBOARD-Mira” and see FAQ “Choosing the right Linux-BSP version” for more information.

4.4.1 Creating a bootable SD Card

In case that your phyBOARD-Mira i.MX 6 does not start anymore due to a damaged bootloader, you need to boot from an SD card. This SD card must be formatted in a special way, because the i.MX 6 does not use file systems. Instead it is hard coded at which sectors of the SD card the i.MX 6 expects the bootloader.

Bitbake, a tool integrated in *Yocto*, builds an image with the ending *.sdcard which fulfills the requirements mentioned above. So we can copy this image to an SD card with the help of *dd*.

All files on the SD card will be erased! Be sure to copy the image to the correct device!

- Get the correct device name with `sudo fdisk -l` or from the last outputs of `dmesg` after inserting the SD card.
- Now use the following command to create your bootable SD card:

```
sudo dd if=phytec-qt5demo-image-phyboard-mira-imx6-*.sdcard
of=/dev/<DEVICE>
```

4.4.2 Flashing the Bootloader

- Set the DIP switch S2 on the base board to ON and insert the SD card to the device.

The partition *boot* on the SD card should automatically be mounted as directory */boot* by the Barebox. Otherwise you can mount it manually using the following Barebox commands:

```
mkdir /boot
mmc0.probe=1
mount /dev/mmc0.0 /boot
```

- Write the Barebox into the Flash by typing:
  ```
  barebox_update -t nand /boot/barebox.bin
  ```

4.4.3 Writing the Kernel / Root File System into Flash

- Flash the Linux kernel by typing:
  ```
  erase /dev/nand0.kernel.bb
  cp /boot/linuximage /dev/nand0.kernel.bb
  ```
- Flash the Oftree by typing:
  ```
  erase /dev/nand0.oftree.bb
  cp /boot/oftree /dev/nand0.oftree.bb
  ```
- Write the root file system into the Flash by typing:
  ```
  ubiformat /dev/nand0.root
  ubiattach /dev/nand0.root
  ubimkvol /dev/ubi0 root 0
  cp /boot/root.ubifs /dev/ubi0.root
  ```

4.5 Setup your own Linux-Host-PC

This chapter is for developers who want to use there own existing environment and not our modified Ubuntu version. It is not needed if you have already installed our modified Ubuntu version like explained in the Application Guide.
We will give an overview of the modifications which we made to the Ubuntu version on the phyBOARD-Mira-iMX6 USB flash drive in comparison to the original Ubuntu in this chapter. In the following we distinguish between optional and essential modifications. So you can see faster which changes are important to execute this Quickstart in case you do not want to use our modified Ubuntu version. You can find a step-by-step instruction of the essential changes in order to modify your own distribution.

⚠	We can not guarantee that the presented changes are compatible to other distributions or versions. If you want to use another distribution, it might take a lot of individual initiative. We do not support other distributions. You should be sure about what you do.

4.5.1 Essential Settings

In the following section you get a short instruction about the important settings which are essential to guarantee the execution of the examples in this Application Guide.

4.5.1.1 Installation of Software Packages

First of all various software packages that are required must be installed using the package manager *APT*.

To gain a better understanding of the packages required, they are installed separately according to their function:

1. Packages which are needed for compiling and building the Board Support Package:
   ```
   sudo apt-get -y install gawk wget git-core diffstat unzip texinfo
   gcc-multilib build-essential chrpath socat libsdl1.2-dev xterm
   ```
2. Packages for development
   ```
   sudo apt-get -y install vim eclipse
   ```
3. Packages to set up the TFTP server:
   ```
   sudo apt-get -y install tftpd-hpa
   ```

During installation some programs may ask for a license agreement. Just go through the steps shown on the screen.

The first preparations are completed. In the next sections you will proceed with building the Board Support Package installation of *Eclipse* and *Qt Creator* and the set up of the TFTP server.

4.5.1.2 Set the Git Configuration

The Board Support Package is heavily based on *Git*. *Git* needs some information from you as a user to be able to identify which changes were done by whom. So at least set the *name* and *email* in your git configuration, otherwise building the BSP generates many warnings.

- Enter following two commands to directly set them
  ```
  git config --global user.email "your_email@example.com"
  git config --global user.name "name surname"
  ```

4.5.1.3 Build the Board Support Package and Install the SDK

In this chapter we build our Board Support Package with the help of the *Yocto Project*.

First we create a folder for the BSP in our example we use the path as it is in our Live System.

- Enter the following commands:
  ```
  sudo mkdir -p /opt/PHYTEC_BSPs/phyBOARD-MIRA/
  sudo chmod -R 777 /opt/PHYTEC_BSPs/
  cd /opt/PHYTEC_BSPs/phyBOARD-MIRA/
  ```
- Download the *phyLinux* script and set execution privileges:
  ```
  wget ftp://ftp.phytec.de/pub/Software/Linux/Yocto/Tools/phyLinux
  chmod +x ./phyLinux
  ```

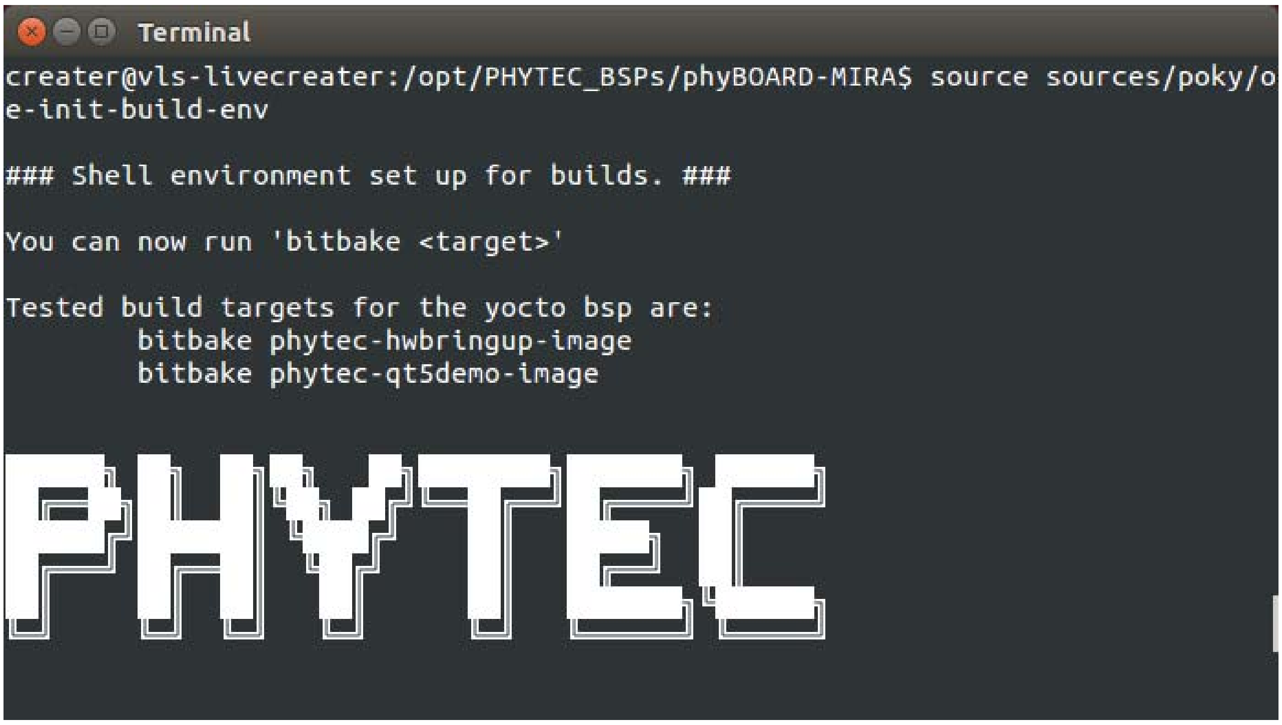
The *phyLinux* script is a basic management tool for PHYTEC Yocto BSP releases. It is mainly a tool to get started with the BSP structure. You can get all the BSP sources without the need of interacting with repo or git.

- Start the phyLinux script
  ```
  ./phyLinux init
  ```

During the execution of the init command, you need to choose your processor platform (iMX6), PHYTEC's BSP release number (PD15.2.0) and the hardware you are working on (phyboard-mira-imx6-4 or phyboard-mira-imx6-3).

After you downloaded all the meta data with *phyLinux*, you have to set up the shell environment variables. This needs to be done every time you open a new shell for starting builds. We use the shell script provided by poky in its default configuration.

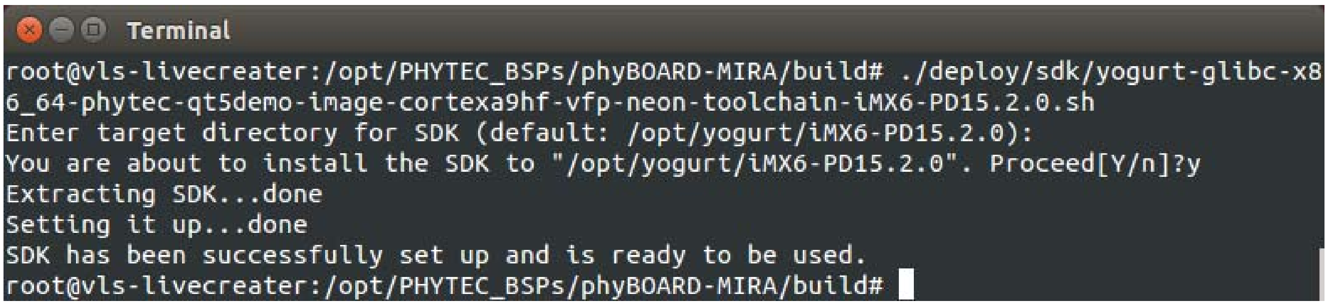
- Type:
  ```
  source sources/poky/oe-init-build-env
  ```



```
creater@vls-livecreater:/opt/PHYTEC_BSPs/phyBOARD-MIRA$ source sources/poky/o
e-init-build-env

### Shell environment set up for builds. ###

You can now run 'bitbake <target>'

Tested build targets for the yocto bsp are:
        bitbake phytec-hwbringup-image
        bitbake phytec-qt5demo-image
```

The current working directory of the shell should change to build/ and you are now ready to build your first images.

- Build the BSP:
  `bitbake phytec-qt5demo-image`
- Generate the SDK needed for *Eclipse* and *Qt Creator*:
  `bitbake phytec-qt5demo-image -c populate_sdk`
- Install the SDK into the recommended default directory:
  ./`deploy/sdk/yogurt-glibc-*.sh`

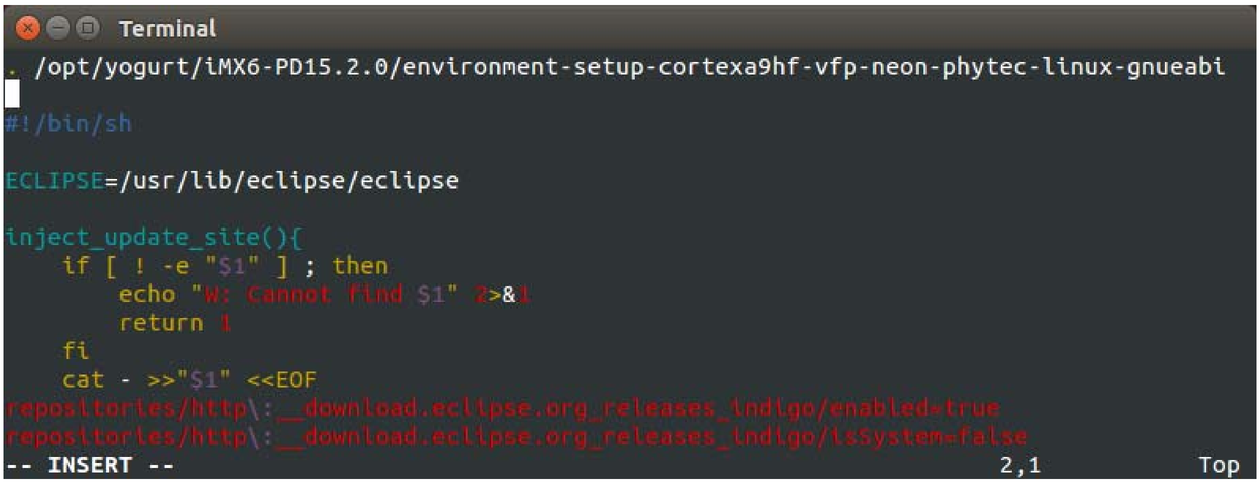


```
root@vls-livecreater:/opt/PHYTEC_BSPs/phyBOARD-MIRA/build# ./deploy/sdk/yogurt-glibc-x8
6_64-phytec-qt5demo-image-cortexa9hf-vfp-neon-toolchain-iMX6-PD15.2.0.sh
Enter target directory for SDK (default: /opt/yogurt/iMX6-PD15.2.0):
You are about to install the SDK to "/opt/yogurt/iMX6-PD15.2.0". Proceed[Y/n]?y
Extracting SDK...done
Setting it up...done
SDK has been successfully set up and is ready to be used.
root@vls-livecreater:/opt/PHYTEC_BSPs/phyBOARD-MIRA/build#
```

4.5.1.4 Set up Eclipse and Integrate Plug-ins

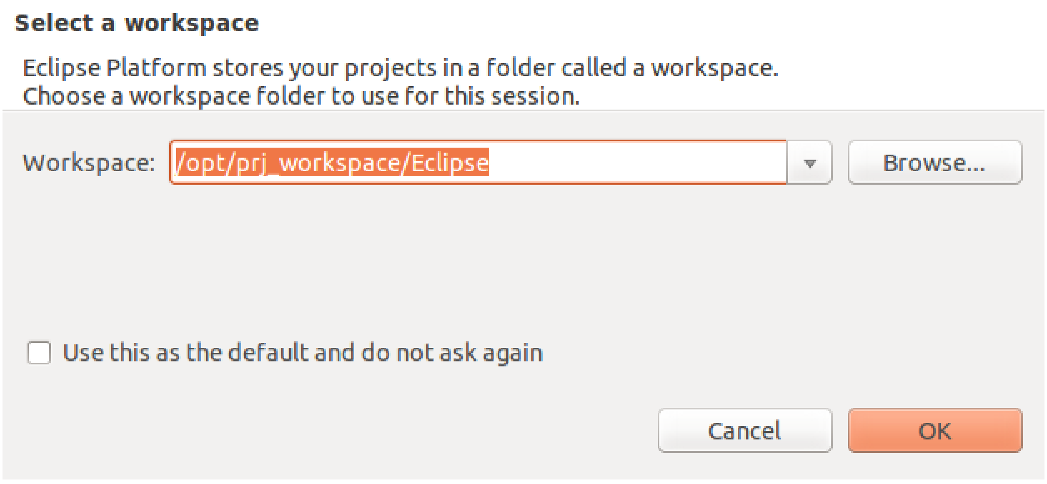
The instructions in this chapter show how to setup *Eclipse* and integrate the C/C++ plug-in. Thus you will be able to assign your own programs, written in *Eclipse*, to the target.

- First of all you have to create a workspace folder, in which you can save your Eclipse projects.
  In this example we create the workspace in the /opt/ directory as it is in the Live System:
  ```
  mkdir -p /opt/prj_workspace/Eclipse
  sudo chmod -R 777 /opt/prj_workspace/
  ```
- Before we start *Eclipse* we must set the correct environment to our SDK:
  ```
  sudo vi /usr/bin/eclipse
  ```
- Enter following line before *#!/bin/sh:*
  ```
  . /opt/yogurt/iMX6-PD15.2.0/environment-setup-cortexa9hf-vfp-neon-
  phytec-linux-gnueabi
  ```



```
. /opt/yogurt/iMX6-PD15.2.0/environment-setup-cortexa9hf-vfp-neon-phytec-linux-gnueabi

#!/bin/sh

ECLIPSE=/usr/lib/eclipse/eclipse

inject_update_site(){
    if [ ! -e "$1" ] ; then
        echo "W: Cannot find $1" 2>&1
        return 1
    fi
    cat - >>"$1" <<EOF
repositories/http\:__download.eclipse.org_releases_indigo/enabled=true
repositories/http\:__download.eclipse.org_releases_indigo/isSystem=false
-- INSERT --                                                    2,1          Top
```

- Afterwards open *Eclipse* and insert the path to the created workspace in the pop-up window:
  ```
  eclipse
  ```
- Enter the path to the created workspace:
  ```
  /opt/prj_workspace/Eclipse
  ```



Select a workspace

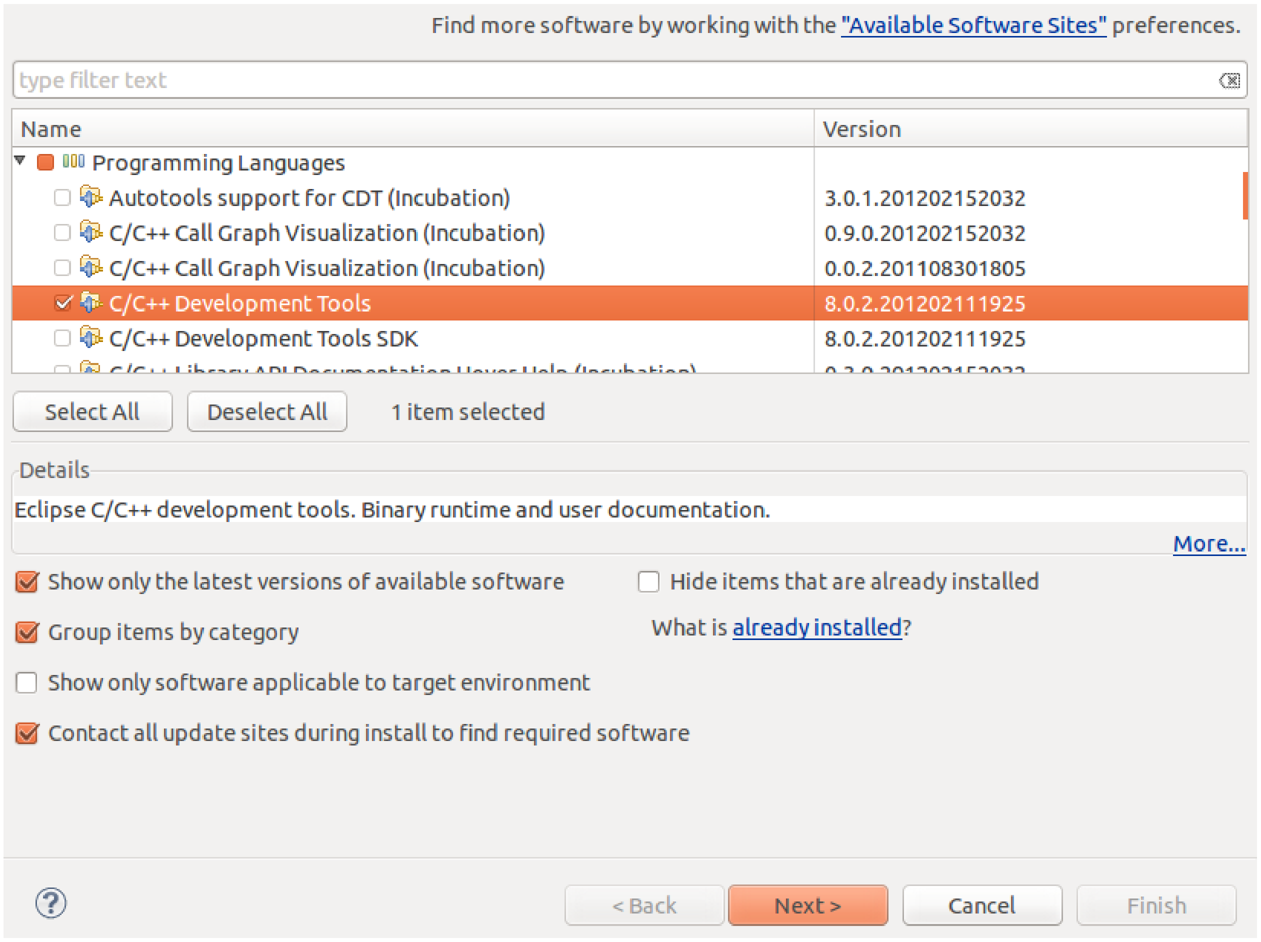
Eclipse Platform stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: /opt/prj_workspace/Eclipse

- After *Eclipse* has started click ***Help*** in the menu bar and then ***Install new Software***.

A window opens to add a plug-in to *Eclipse*.

- In the drop-down menu *Work with* select ***Juno http://download.eclipse.org/releases/juno***
- The available plug-ins appear in the selection area below now. Expand ***Programming Languages*** and check ***C/C++ Development Tools***. Click ***Next***.


Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Version
▾ Programming Languages	
☐ Autotools support for CDT (Incubation)	3.0.1.201202152032
☐ C/C++ Call Graph Visualization (Incubation)	0.9.0.201202152032
☐ C/C++ Call Graph Visualization (Incubation)	0.0.2.201108301805
☑ C/C++ Development Tools	8.0.2.201202111925
☐ C/C++ Development Tools SDK	8.0.2.201202111925

Select All | Deselect All | 1 item selected

Details

Eclipse C/C++ development tools. Binary runtime and user documentation.

More...

☑ Show only the latest versions of available software
☐ Hide items that are already installed
☑ Group items by category
What is already installed?
☐ Show only software applicable to target environment
☑ Contact all update sites during install to find required software

< Back | Next > | Cancel | Finish

- Now the system displays an overview of the installation details. Click ***Next*** to proceed.
- Finally accept the licensing agreement and click ***Finish*** to start the installation of the required software.
- After that start *Eclipse* with the option clean, which cleans any cached data:
  `eclipse --clean`

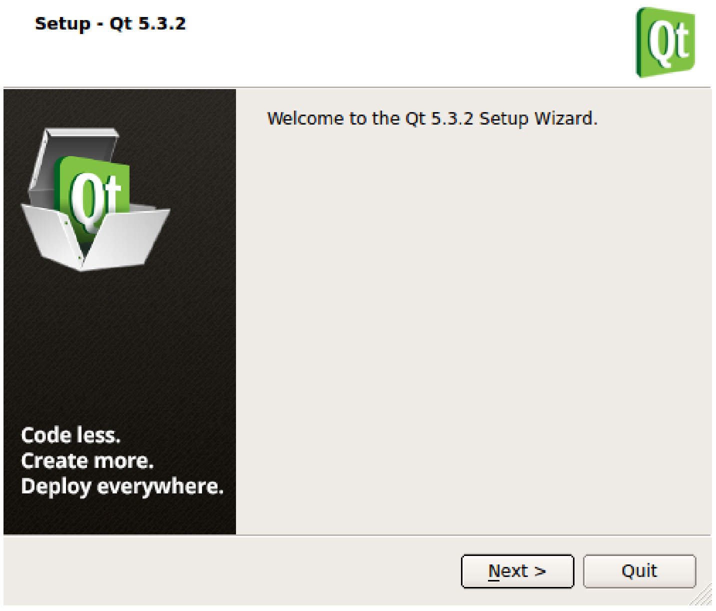
	Congratulations! You have successfully integrated the *CDT* plug-in in *Eclipse* and now you can start programming in *C/C++*. In the next section you will find a short introduction on how to install and setup the *Qt Creator*.

4.5.1.5 Install and Setup *Qt Creator*

Because we want the same qmake version as in our BSP we install *Qt Creator* manually.

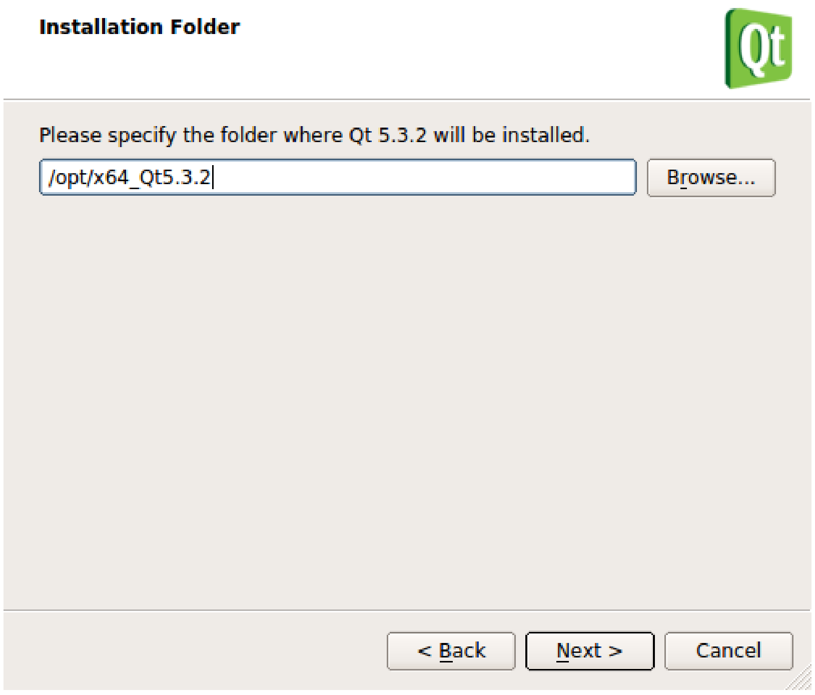
- Download *Qt Creator* with following command:
  `wget http://download.qt.io/official_releases/qt/5.3/5.3.2/qt-opensource-linux-x64-5.3.2.run`
- Set execute priviliges:
  `sudo chmod +x qt-opensource-linux-x64-5.3.2.run`
- Run the *Qt Creator* installation routine
  `./qt-opensource-linux-x64-5.3.2.run`

This opens a window for the installation.



Welcome to the Qt 5.3.2 Setup Wizard.

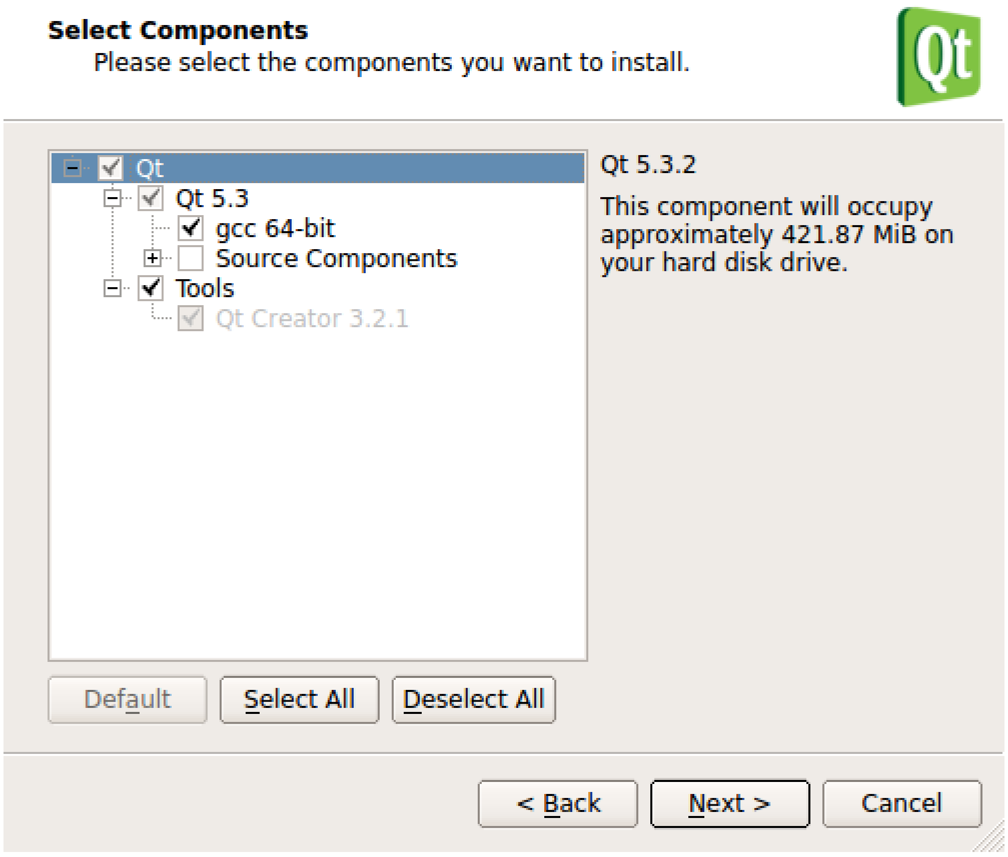
- Step trough the installation and if you are asked set following installation directory:
  `/opt/x64_Qt5.3.2`


Installation Folder

Please specify the folder where Qt 5.3.2 will be installed.

/opt/x64_Qt5.3.2

- In the *Select Components* window keep the default selection of the components and click ***Next***.


Select Components
Please select the components you want to install.

- [✓] Qt
  - [✓] Qt 5.3
    - [✓] gcc 64-bit
    - [ ] Source Components
  - [✓] Tools
    - [✓] Qt Creator 3.2.1

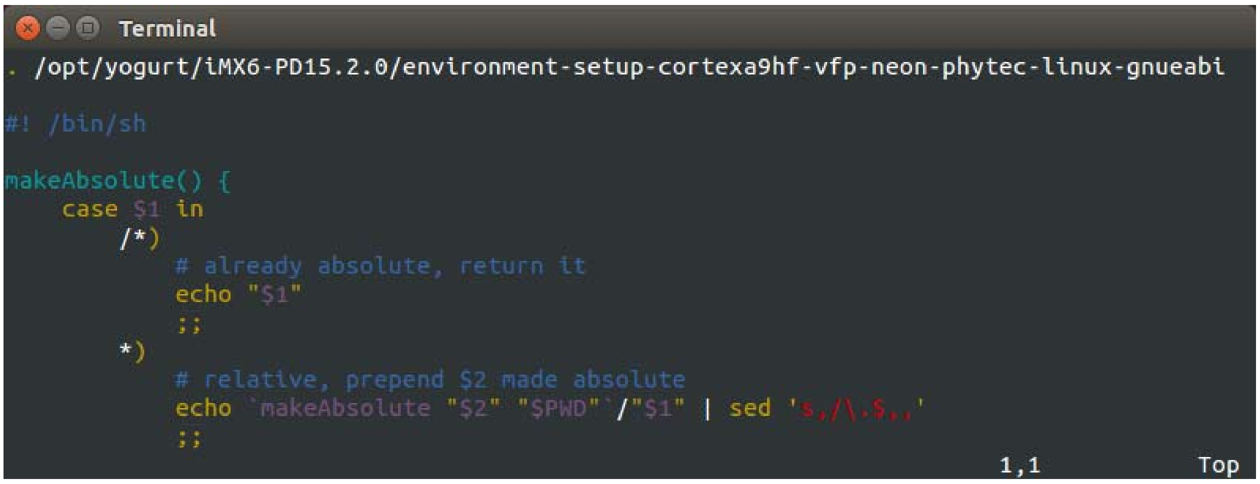
Qt 5.3.2

This component will occupy approximately 421.87 MiB on your hard disk drive.

- Proceed trough the license agreement and start the installation.
- After the installation is finished make a symbolic link to easily start *Qt Creator*:

```
sudo ln -s /opt/x64_Qt5.3.2/Tools/QtCreator/bin/qtcreator.sh
/usr/bin/qtcreator.sh
```

- Set the correct environment to our SDK:
  `sudo vi /usr/bin/qtcreator.sh`
- Enter following line before *#!/bin/sh:*
  `. /opt/yogurt/iMX6-PD15.2.0/environment-setup-cortexa9hf-vfp-neon-phytec-linux-gnueabi`

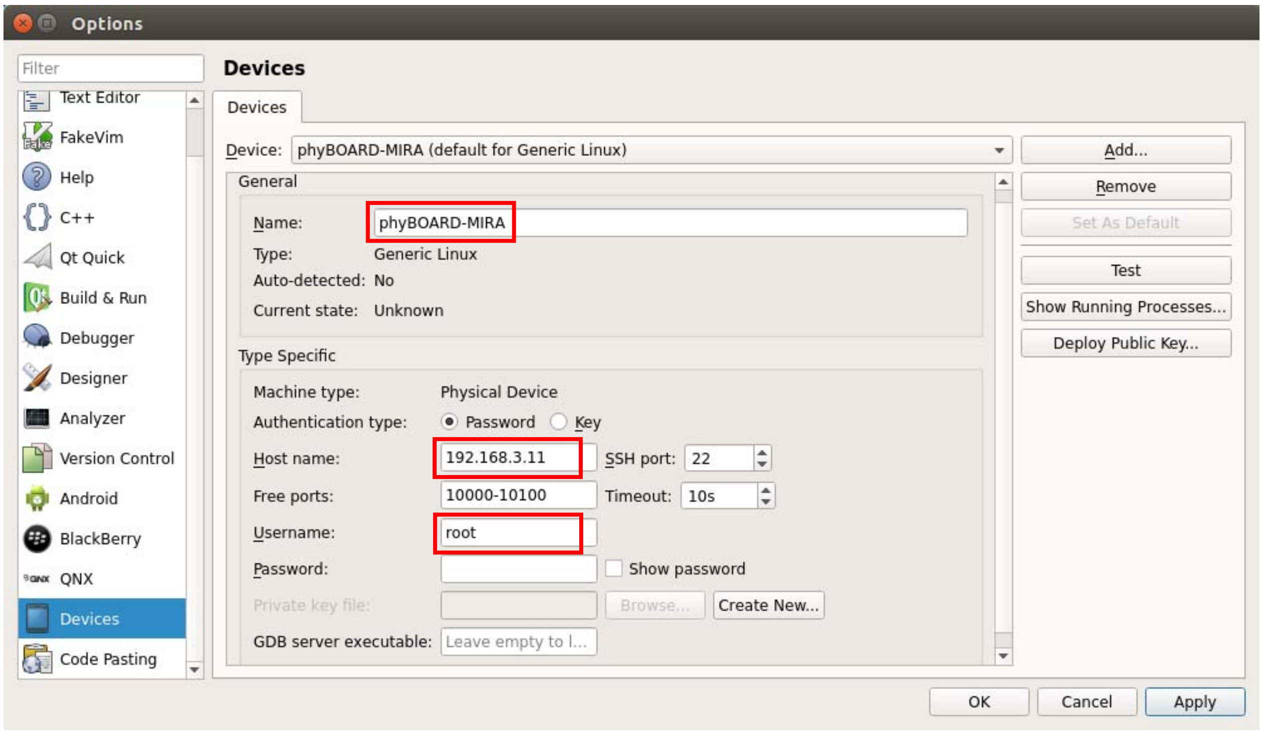


```
. /opt/yogurt/iMX6-PD15.2.0/environment-setup-cortexa9hf-vfp-neon-phytec-linux-gnueabi

#! /bin/sh

makeAbsolute() {
    case $1 in
        /*)
            # already absolute, return it
            echo "$1"
            ;;
        *)
            # relative, prepend $2 made absolute
            echo `makeAbsolute "$2" "$PWD"`/"$1" | sed 's,/\.$,,'
            ;;
                                                                    1,1           Top
```

- Create a workspace for *QtCreator* e.g.
  `mkdir -p /opt/prj_workspace/Qt/build`
- Start *Qt Creator* with the following command in a terminal:
  `/usr/bin/qtcreator.sh`
- Add the phyBOARD-Mira as device with following inputs by opening
  Tools --> Options --> Devices --> Add --> Generic Linux Device:

Name:	phyBOARD-MIRA
IP:	192.168.3.11
Username:	root

  Keep the *password* empty

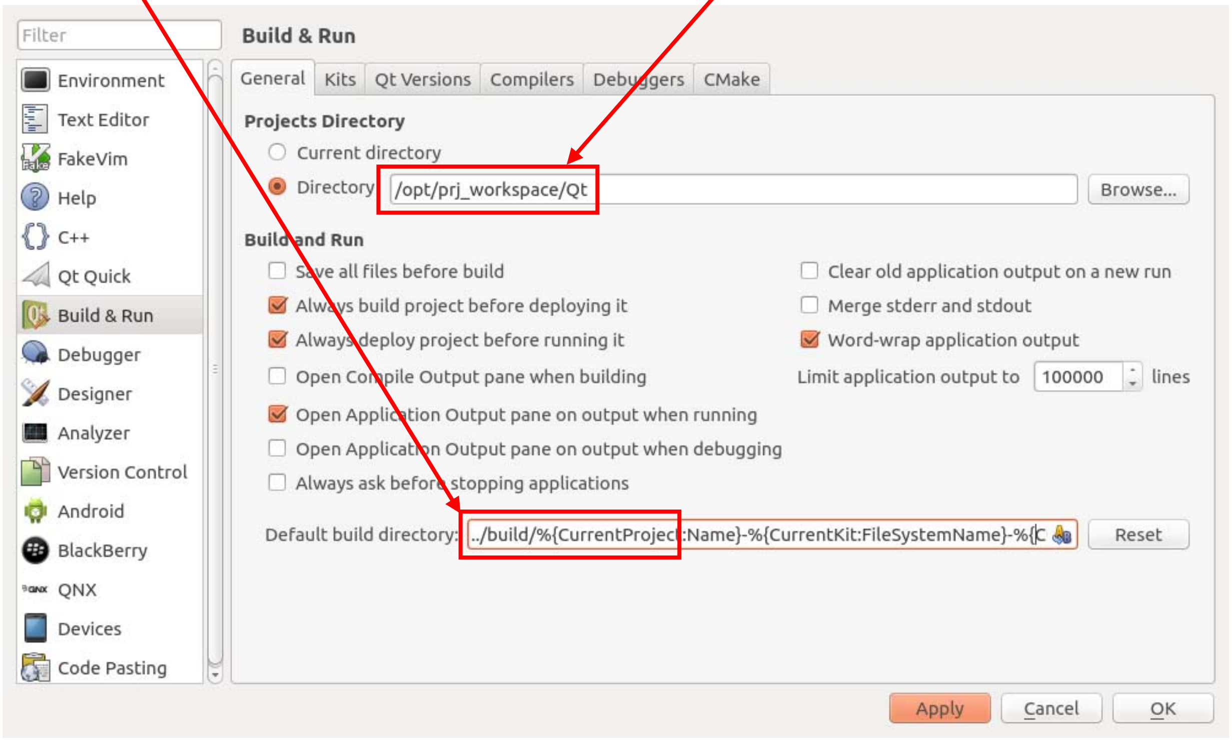


Device: phyBOARD-MIRA (default for Generic Linux)

General
Name: phyBOARD-MIRA
Type: Generic Linux
Auto-detected: No
Current state: Unknown

Type Specific
Machine type: Physical Device
Authentication type: [✓ Password] Key
Host name: 192.168.3.11 SSH port: 22
Free ports: 10000-10100 Timeout: 10s
Username: root
Password: ☐ Show password
Private key file:
GDB server executable: Leave empty to l...

- Set the project directory and build directory
  Tools --> Options --> Build & Run --> General.
  Projects Directory --> Directory: `/opt/prj_workspace/Qt`
  Change the beginning of the entry in the *Default build directory* field:
  `../build/%{CurrentProject:Name}`



Build & Run

General | Kits | Qt Versions | Compilers | Debuggers | CMake

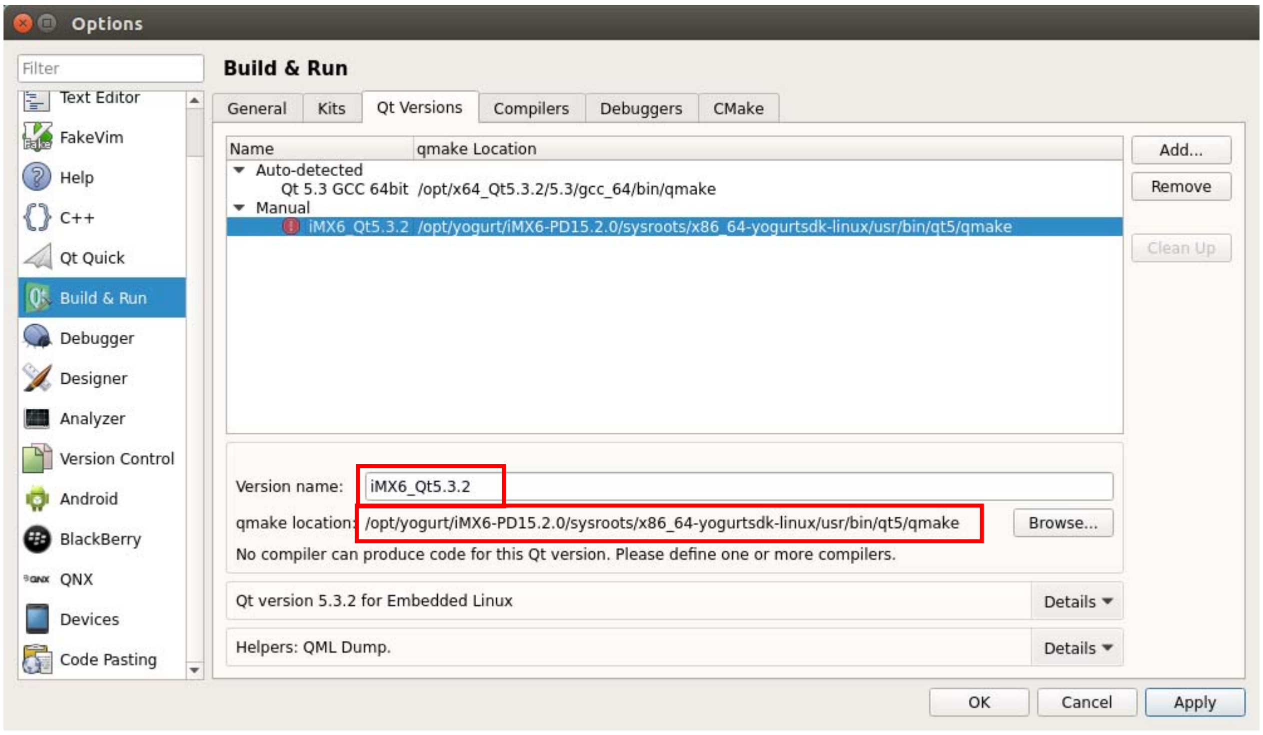
Projects Directory
- Current directory
- Directory: /opt/prj_workspace/Qt

Build and Run
- Save all files before build
- Always build project before deploying it
- Always deploy project before running it
- Open Compile Output pane when building
- Open Application Output pane on output when running
- Open Application Output pane on output when debugging
- Always ask before stopping applications
- Clear old application output on a new run
- Merge stderr and stdout
- Word-wrap application output

Limit application output to 100000 lines

Default build directory: ../build/%{CurrentProject:Name}-%{CurrentKit:FileSystemName}-%{C

- Select tab ***Qt Versions*** and click ***Add...*** to add a new one.
  Name: `iMX6_Qt5.3.2`
  qmake Location: `/opt/yogurt/iMX6-PD15.2.0/sysroots/x86_64-yogurtsdk-linux/usr/bin/qt5/qmake`



Options

Build & Run

General | Kits | Qt Versions | Compilers | Debuggers | CMake

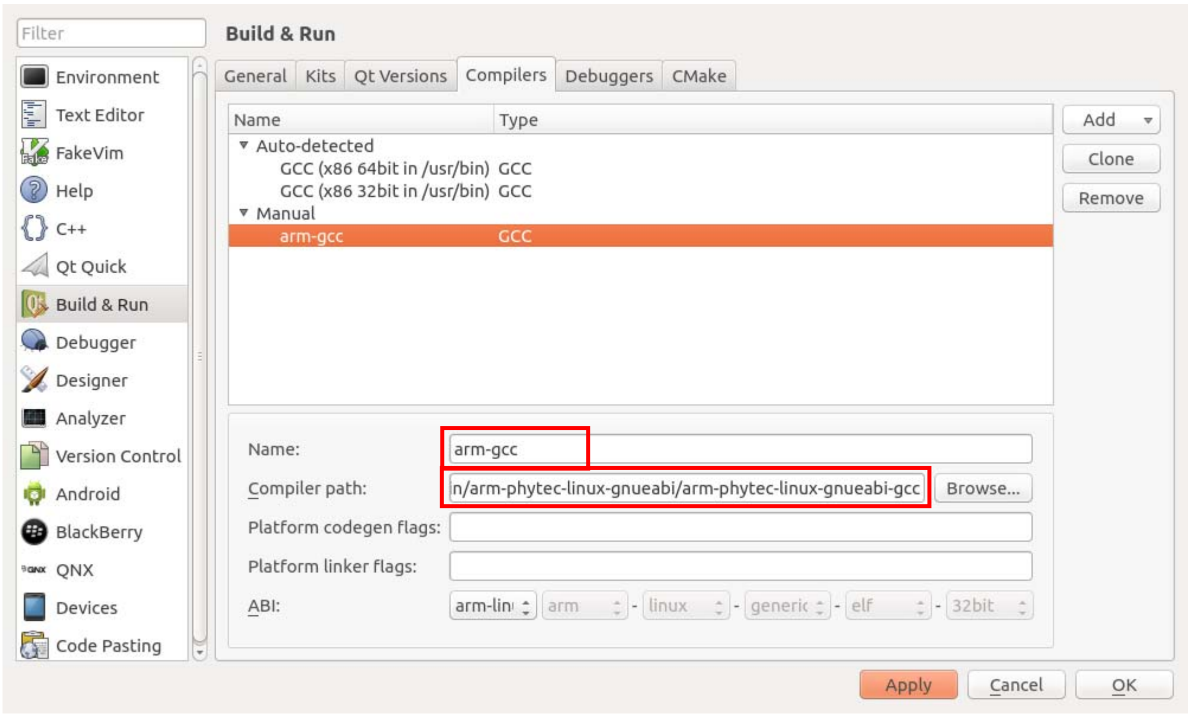
Name	qmake Location
Auto-detected	
Qt 5.3 GCC 64bit	/opt/x64_Qt5.3.2/5.3/gcc_64/bin/qmake
Manual	
iMX6_Qt5.3.2	/opt/yogurt/iMX6-PD15.2.0/sysroots/x86_64-yogurtsdk-linux/usr/bin/qt5/qmake

Version name: iMX6_Qt5.3.2
qmake location: /opt/yogurt/iMX6-PD15.2.0/sysroots/x86_64-yogurtsdk-linux/usr/bin/qt5/qmake
No compiler can produce code for this Qt version. Please define one or more compilers.

Qt version 5.3.2 for Embedded Linux

Helpers: QML Dump.

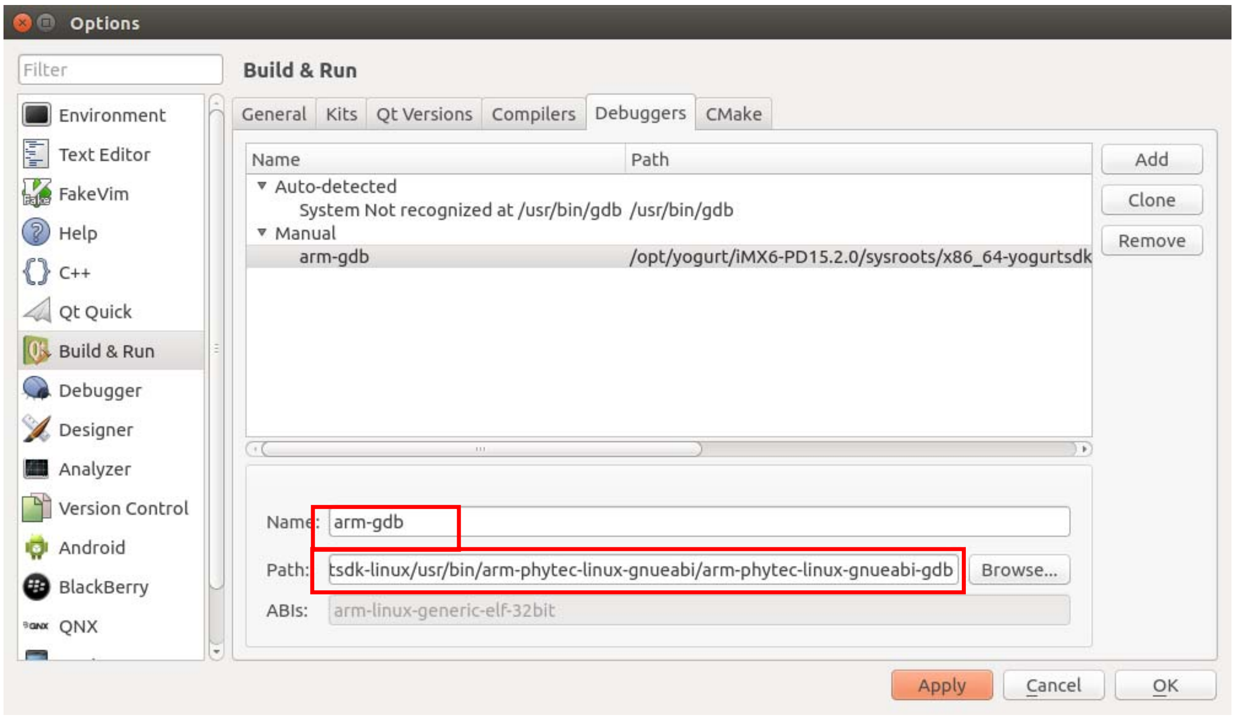
- Select tab ***Compilers*** and click on ***Add*** select ***GCC***
  Name: `arm-gcc`
  Compiler Path: `/opt/yogurt/iMX6-PD15.2.0/sysroots/x86_64-yogurtsdk-linux/usr/bin/arm-phytec-linux-gnueabi/arm-phytec-linux-gnueabi-gcc`



Name	Type
▾ Auto-detected	
GCC (x86 64bit in /usr/bin)	GCC
GCC (x86 32bit in /usr/bin)	GCC
▾ Manual	
arm-gcc	GCC

Name: arm-gcc
Compiler path: n/arm-phytec-linux-gnueabi/arm-phytec-linux-gnueabi-gcc
Platform codegen flags:
Platform linker flags:
ABI: arm-lin - arm - linux - generic - elf - 32bit

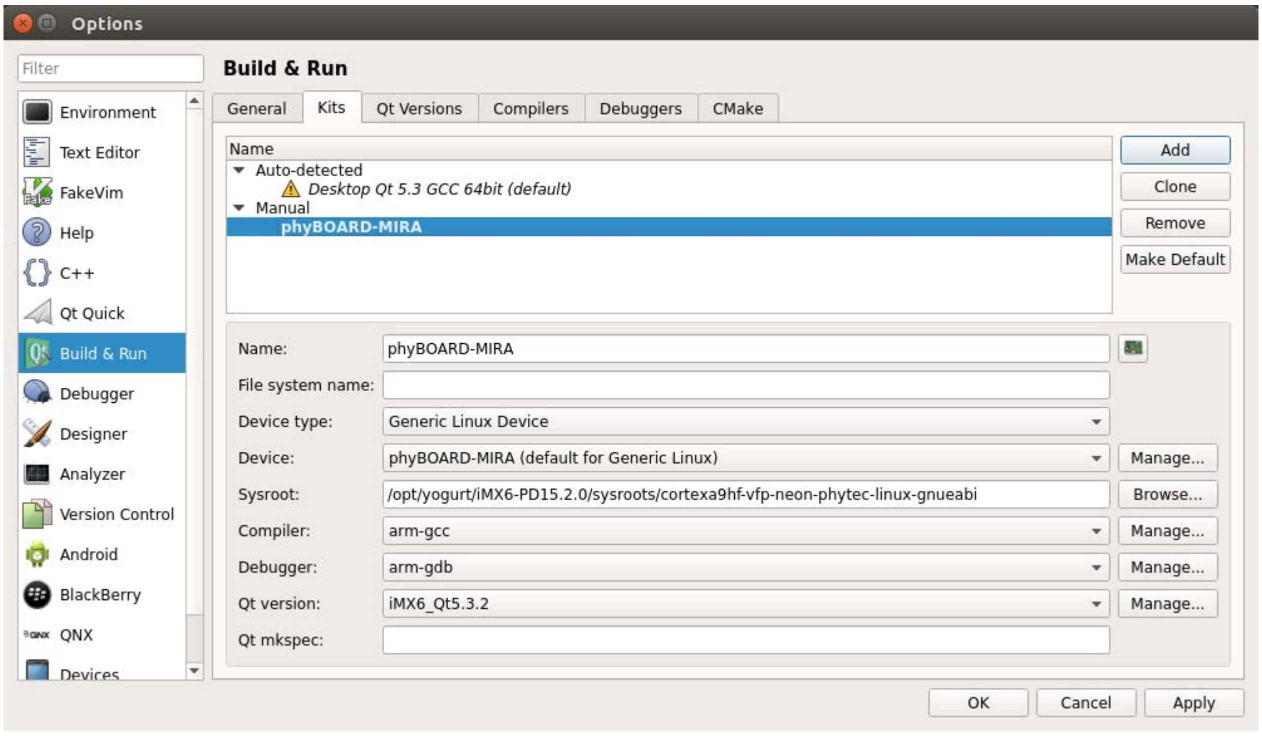
- Select tab ***Debuggers*** an click on ***Add***
  Name: `arm-gdb`
  Path: `/opt/yogurt/iMX6-PD15.2.0/sysroots/x86_64-yogurtsdk-linux/usr/bin/arm-phytec-linux-gnueabi/arm-phytec-linux-gnueabi-gdb`



Name	Path
▾ Auto-detected	
System Not recognized at /usr/bin/gdb	/usr/bin/gdb
▾ Manual	
arm-gdb	/opt/yogurt/iMX6-PD15.2.0/sysroots/x86_64-yogurtsdk

Name: arm-gdb
Path: tsdk-linux/usr/bin/arm-phytec-linux-gnueabi/arm-phytec-linux-gnueabi-gdb
ABIs: arm-linux-generic-elf-32bit

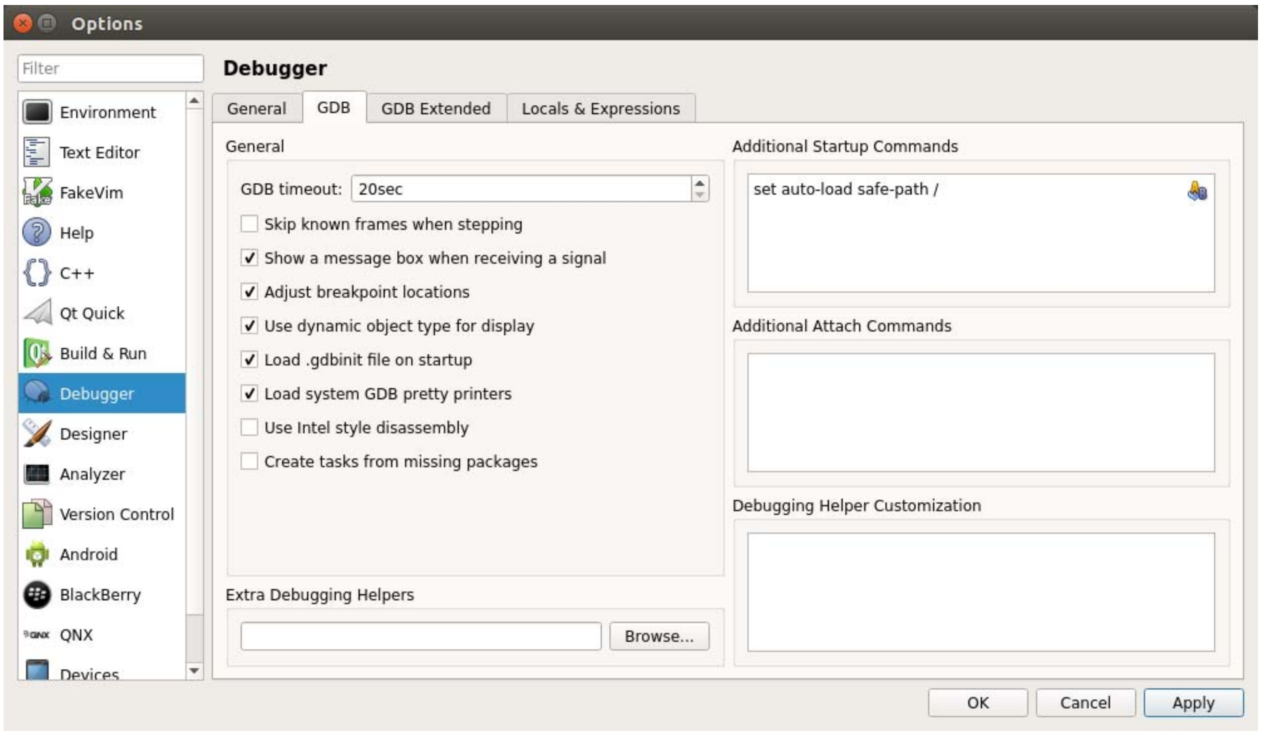
- Select tab ***Kit*** and click on ***Adds***
  Name: `phyBOARD-MIRA`
  Device Type: `Generic Linux Device`
  Sysroot: `/opt/yogurt/iMX6-PD15.2.0/sysroots/cortexa9hf-vfp-neon-phytec-linux-gnueabi`
  Compiler: `arm-gcc`
  Debugger: `arm-gdb`
  Qt version: `iMX6_Qt5.3.2`



Name
Auto-detected
Desktop Qt 5.3 GCC 64bit (default)
Manual
phyBOARD-MIRA

Name:	phyBOARD-MIRA
File system name:	
Device type:	Generic Linux Device
Device:	phyBOARD-MIRA (default for Generic Linux)
Sysroot:	/opt/yogurt/iMX6-PD15.2.0/sysroots/cortexa9hf-vfp-neon-phytec-linux-gnueabi
Compiler:	arm-gcc
Debugger:	arm-gdb
Qt version:	iMX6_Qt5.3.2
Qt mkspec:	

- Select ***Debugger*** on the left side of the window and click on the ***GDB*** tab.
- Enter following command in *the Addition Startup Commands*:
  `set auto-load safe-path /`


Debugger

General | GDB | GDB Extended | Locals & Expressions

General

GDB timeout: 20sec

- [ ] Skip known frames when stepping
- [x] Show a message box when receiving a signal
- [x] Adjust breakpoint locations
- [x] Use dynamic object type for display
- [x] Load .gdbinit file on startup
- [x] Load system GDB pretty printers
- [ ] Use Intel style disassembly
- [ ] Create tasks from missing packages

Extra Debugging Helpers

Additional Startup Commands

```
set auto-load safe-path /
```

Additional Attach Commands

Debugging Helper Customization

- Click on ***Apply*** and close the window with ***OK***.

Now all configurations are done and you can use this Kit to cross compile your application also if you generate a new project.

4.5.1.6 Setting up a TFTP server

In the chapter *"Installation of software packages"* you have installed the required packages to set up a TFTP server. Now we have to change some short settings.

- First change the file */etc/default/tftp-hpa* as follows:
```
TFTP_USERNAME="tftp"
TFTP_DIRECTORY="/tftpboot"
TFTP_ADDRESS="0.0.0.0:69"
TFTP_OPTIONS="--secure"
```
- Then create a folder called */tftpboot*. The TFTP server will access this folder later.
```
mkdir /tftpboot
```
- Finally we have to set the right permissions:
```
chmod 777 /tftpboot
```

⚠	Due to a bug in the current version of the TFTP-hpa daemon the daemon does not start after the system is rebooted. You have to start it in the terminal with the following command:
	`restart tftpd-hpa`
	A permanent workaround is to create a file called *tftp-hpa* under */etc/network/if-up.d/*. Insert the following commands in the file:
	`#!/bin/sh`
	`restart tftpd-hpa`
	After saving the file set the correct permission with
	`chmod 755 /etc/network/if-up.d/tftpd-hpa .`

👍	You have successfully set up the TFTP server. In the future the phyBOARD-Mira i.MX 6 can access to the /tftpboot/ folder to load the images.

4.5.2 Optional Settings

In the following section we list the optional settings. These settings are not needed for a successful operation of this Application Guide. They only simplify the handling and the look of the system. For this reason we show the modifications without big explanation.

- *vim*, the improved *vi* editor is installed.
- Desktop-icons for faster and easier start of required programs are created.
- The following modifications were made to the look of *Ubuntu*:
  – Other wallpaper and associated options are adjusted with the help of **gsettings**
  – The color of the *Gnome* terminal is changed.
  - The scrolling log for the *Gnome* terminal is changed.
- **History-search-backward** and **history-search-forward** in */etc/inputrc* is activated. This allows you to search through your history with the entered string.
- Also there are some scripts that will be executed at the first start after the installation. These scripts ensure that the right permissions are set for the created user.

👍	Congratulations! You have successfully configured your *Ubuntu* to work with.

4.6 System Level Hardware Information

4.6.1 Soldering Jumpers

Numerous jumpers and 0 Ohm resistors allow to configure the phybOARD-Mira according to your needs.

⚠	Due to the small footprint of the jumpers we do not recommend manual jumper modifications. This might also render the warranty invalid. Please contact our sales team if you need one of the configurations described below.

The following table lists all jumpers and resistors, and describes their function.

Jumper / Resistor	Description	See Section
J1	Selecting the reset signal at A/V connector X13	*4.6.7*
J5, J6	Rerouting of the USB OTG interface	*4.6.2*
J9, J10	Rerouting of the USB host interface	
J11	Changing the I^{2}C address of touchscreen controller U5	
J12	Selecting the reset signal at Mini PCIe connector X7	*4.6.6*
J14	Configuring rising, or falling edge strobe for the LVDS Deserializer of the phyCAM-S+ interface	
J17 – J20 R100 – R108	Adapting display connector X9 to different displays	*4.6.5*
J21 – J23, J25, J26J	Routing of the SIM card signals to expansion connector X17	*4.6.6, 4.6.8*
J28	Configuring the HDMI connector's shield signals	*4.6.4*
R110, R111	Rerouting of the FLEXCAN1 interface	*4.6.3.1*

Table 16: Soldering Jumpers on the phyBOARD-Mira

4.6.2 USB Connectivity

4.6.2.1 Rerouting the USB Interfaces to other Connectors (J5, J6, J9 and J10)

For later expansion boards the USB1 OTG interface can be routed to the expansion connector[8] (X17). Moreover the USB2 host interface can be routed to the Mini PCI Express connector[8] (X7). The following table shows all possible configurations.

Mode	J5	J6
USB1 at USB-OTG connector X5	**1+2**	**1+2**
USB1 at expansion connector X17	2+3	2+3

Table 17: USB1 (USB OTG) Routing Configuration

Mode	J9	J10
USB2 at USB-A connector X6	**1+2**	**1+2**
USB2 at Mini PCIe connector X7	2+3	2+3

Table 18: USB2 (USB host) Routing Configuration

4.6.2.2 Configuring the OTG Operating Mode (R10)

Resistor R10 configures the OTG operating mode. By default this resistor is not ,mounted, which leaves the USB_OTG_ID pin floating, and thus configuring the OTG interface as slave, or according to the configuration of the connected USB device. Mounting a 10 k resistor connects the X_USB_OTG_ID pin to GND, forcing the OTG interface into host mode.

4.6.3 CAN Connectivity

4.6.3.1 Rerouting the CAN Interface(R110 and R111)

The first CAN interface (FLEXCAN1) can be routed to expansion connector X17. The signals are located at pin 47 (X_FLEXCAN1_TX) and pin 48 (X_FLEXCAN1_RX) of X17. The following table shows the possible configurations for FLEXCAN1.

Mode	R110	R111	U2
FLEXCAN1 at pin header X3	**n.m.**	**n.m.**	**m.**
FLEXCAN1 at expansion connector X17 (TTL level)[9]	1+2	1+2	n.m.

Table 19: Configurations for CAN interface

8: **Caution!** There is no protective circuit for the USB interfaces brought out at the expansion connector (X17), or the Mini PCI Express connector (X7).

9: The standard kit comes with the CAN transceiver installed at U2. Please contact our sales team, if you need the FLEXCAN1 interface at expansion connector X17.

4.6.3.2 Second CAN Interface at X17

For later expansion boards the second CAN interface (FLEXCAN2) is also available at the expansion connector X17 (*Table 28:*   PHYTEC Expansion Connector X17 (continued)).

Usage of FLEXCAN2 at X17 requires changing of the pin muxing and additional software development.

4.6.4 I^{2}C Connectivity

The I2C1 interface of the i.MX 6 is available at different connectors on the phyBOARD-Mira. The following table provides a list of the connectors and pins with I^{2}C connectivity.

Connector	Location
Mini PCIe connector X7	pin 32 (I2C1_SDA), pin 30 (I2C1_SCL)
phyCAM-S connector X10	pin 4 (I2C1_SDA), pin 5 (I2C1_SCL)
Expansion connector X17	pin 11 (I2C1_SDA), pin 13 (I2C1_SCL)
A/V connector X13	pin 16 (I2C1_SDA), pin 15 (I2C1_SCL)

Table 20: I2C1 Connectivity

To avoid any conflicts when connecting external I^{2}C devices to the phyBOARD-Mira the addresses of the on-board I^{2}C devices must be considered. *Table 21* lists the addresses already in use. The table shows only the default address.

Board	Prod. No.	Device	Address used (7 MSB)
I2C3			
phyCORE-i.MX 6	PCM-058	EEPROM	0x50
		PMIC	0xB0, 0xB1
I2C1			
phyBOARD-Mira	PBA-C-06	Touch controller	0x44
		LED dimmer	0x62
		RTC	0x68
		Mini PCIe	- [10]
		phyCAM-S+	- [10]
AV-Adapter HDMI	PEB-AV-01	HDMI Core	0x70
		CEC Core	0x34
AV-Adapter Display	PEB-AV-02	GPIO Expander	0x41
Evaluation Board	PEB-EVAL-01	EEPROM	0x56
M2M Board	PEB-C-01	GPIO Expander	0x20
		GPIO Expander	0x21
		GPIO Expander	0x22

Table 21: I^{2}C Addresses in Use

10: Depends on the device connected

4.6.4.1 Software Implementation

The driver for I^{2}C can be accessed by its API within the kernel. If you connect a chip to I^{2}C, you should implement its driver into the kernel, as well. The I^{2}C driver offers no /dev/i2c entry in userspace, because using it would mean to place the driver for your chip in userspace, too, what most probably would not be useful. Please note that the BSP already includes a couple of deactivated drivers for various chips. These drivers might be helpful for you even though they are usually not tested. In the following you will find a description of the basic device usage of a few integrated I^{2}C devices.

It is also possible to access the I^{2}C Bus from userland in a rudimentary way. You may use the tools `i2cdetect`, `i2cget`, `i2cset` and `i2cdump` for some quick experiments. To program the character devices */dev/i2c-** directly see the kernel documentation in *Documentation/i2c/dev-interface*.

4.6.5 LVDS connector (X9)

In the following table is a complete overview of the LVDS display connector pin assignment.

Pin #	Signal name	Type	SL	Description
1	VCC3V3	OUT	3.3 V	Power supply display[11]
2	VCC3V3	OUT	3.3 V	Power supply display[11]
3	X_LVDS_CONFIG1	OUT	3.3 V	Display configuration pin 1
4	X_LVDS_CONFIG2	OUT	3.3 V	Display configuration pin 2
5	X_LVDS0_TX0-	OUT	3.3 V	LVDS 0 data channel 0 negative output
6	X_LVDS0_TX0+	OUT	3.3 V	LVDS 0 data channel 0 positive output
7	GND	-	-	Ground
8	X_LVDS0_TX1-	OUT	3.3 V	LVDS 0 data channel 1 negative output
9	X_LVDS0_TX1+	OUT	3.3 V	LVDS 0 data channel 1 positive output
10	GND	-	-	Ground
11	X_LVDS0_TX2-	OUT	3.3 V	LVDS 0 data channel 2 negative output
12	X_LVDS0_TX2+	OUT	3.3 V	LVDS 0 data channel 2 positive output
13	GND	-	-	Ground
14	X_LVDS0_CLK-	OUT	3.3 V	LVDS 0 clock channel negative output
15	X_LVDS0_CLK+	OUT	3.3 V	LVDS 0 clock channel positive output
16	GND	-	-	Ground
17	X_LVDS_CONFIG3	OUT	3.3 V	Display configuration pin 3 or LVDS 0 data channel 3 negative output
18	X_LVDS_CONFIG4	OUT	3.3 V	Display configuration pin 4 or LVDS 0 data channel 3 positive output
19	X_LVDS_CONFIG5	OUT	3.3 V	Display configuration pin 5 or LVDS 0 data channel 3 negative output
20	X_LVDS_CONFIG6	OUT	3.3 V	Display configuration pin 6 or LVDS 0 data channel 3 positive output

Table 22: Pin Assignment LVDS Display Connector X9

Various solder jumpers and resistors allow to adapt the pin assignment to different displays. If your kit includes a display all jumpers and resistors are preset corresponding to the display. The following table gives an overview of possible configurations. Please consider the data sheet of the display used to find the right settings.

[11]: Provided to supply any logic on the display adapter. Max. draw 100 mA

Pin #	Signal name	J or R	Setting	Description
3	X_LVDS_CONFIG1	R101[12]	if mounted	High level
		R103[12]	**if mounted**	**Low level**
4	X_LVDS_CONFIG2	**R100[12]**	**if mounted**	**High level**
		R102[12]	if mounted	Low level
17	X_LVDS_CONFIG3	J17	1+2	High level
			2+3	**X_LVDS0_TX3- connected**
18	X_LVDS_CONFIG4	J18	**closed**	**X_LVDS0_TX3+ connected**
			open	Not connected
19	X_LVDS_CONFIG5	J19	1+2	X_LVDS0_TX3- connected
			2+3	High level if R106[12] is mounted Low level if R108[12] is mounted
20	X_LVDS_CONFIG6	J20	1+2	X_LVDS0_TX3+ connected
			2+3	**High level if R105[12] is mounted** Low level if R107[12] is mounted

Table 23: Configuration for LVDS Display Connector X9

⚠	Due to the small footprint of the jumpers and resistors we do not recommend manual jumper modifications. This might also render the warranty invalid. Please contact our sales team if you need one of the configurations described.

12: All configuration resistors have a value of 10 k.

4.6.6 Mini PCI Express Connector (X7)

In the following table is a complete overview of the Mini PCI Express connector pin assignment.

Pin #	Signal name	Type	SL	Description
1	X_ECSPI2_MOSI/PCIe_nWAKE	IN	3.3 V	PCIe WAKE
2	VCC3V3	OUT	3.3 V	3.3 V power supply
3	X_ECSPI2_SS0/PCIe_COEX1	I/O	3.3 V	Coexistence pins for wireless solutions
4	GND	-	-	Ground
5	X_CSI1_DATA06/PCIe_COEX2	I/O	3.3 V	Coexistence pins for wireless solutions
6	VCC1V5	OUT	1.5 V	1.5 V power supply[13]
7	X_ECSPI2_SCLK/PCIe_nCLKREQ	IN	3.3 V	Clock request support
8	X_SIM_VCC	IN	-	UIM_PWR[14]
9	GND	-	-	Ground
10	X_SIM_IO	I/O	-	UIM_DATA[14]
11	X_PCIe0_CLK-	OUT	DIFF	PCIe0 reference clock -
12	X_SIM_CLK	IN	-	UIM_CLK[14]
13	X_PCIe0_CLK+	OUT	DIFF	PCIe0 reference clock +
14	X_SIM_RST	IN	-	UIM_RESET[14]
15	GND	-	-	Ground
16	X_SIM_VPP	IN	-	UIM_VPP[14]
17	RSVD3	-	-	Not connected
18	GND	-	-	Ground
19	RSVD4	-	-	Not connected
20	X_EIM_DA14/PCIe_nW_DISABLE	OUT	3.3 V	Wireless disable signal
21	GND	-	-	Ground
22	X_ECSPI2_MISO/PCIe_nPERST or X_nRESET (J12 1+2)	OUT	3.3 V	Functional card reset by GPIO or X_nRESET
23	X_PCIe_RXN	IN	DIFF	PCIe receive -
24	VCC3V3	OUT	3.3 V	3.3 V power supply
25	X_PCIe_RXP	IN	DIFF	PCIe receive +
26	GND	-	-	Ground

Table 24: Mini PCI Express Connector X7

13: The 1.5 V voltage can be switched OFF with signal X_CSI1_DATA09/EN_VCC1V5.
14: User Identity Module (UIM) signals

Pin #	Signal name	Type	SL	Description
27	GND	-	-	Ground
28	VCC1V5	OUT	1.5 V	1.5 V power supply
29	GND	-	-	Ground
30	X_I2C1_SCL	I/O	3.3 V	I2C1 clock
31	X_PCIe_TXN	OUT	DIFF	PCIe transmit -
32	X_I2C1_SDA	I/O	3.3 V	I2C1 data
33	X_PCIe_TXP	OUT	DIFF	PCIe transmit +
34	GND	-	-	Ground
35	GND	-	-	Ground
36	X_USB2_DM_PCIe	I/O	DIFF	USB host data -[15, 16]
37	GND	-	-	Ground
38	X_USB2_DP_PCIe	I/O	DIFF	USB host data +[15, 16]
39	VCC3V3	OUT	3.3 V	3.3 V power supply
40	GND	-	-	Ground
41	VCC3V3	OUT	3.3 V	3.3 V power supply
42	TP6	IN	NS	Test point for LED_WWAN
43	GND	-	-	Ground
44	TP7	IN	NS	Test point for LED_WLAN
45	RSVD9	-	-	Not connected
46	TP8	IN	NS	Test point for LED_WPAN
47	RSVD10	-	-	Not connected
48	VCC1V5	OUT	1.5 V	1.5 V power supply[17]
49	RSVD11	-	-	Not connected
50	GND	-	-	Ground
51	RSVD12	-	-	Not connected
52	VCC3V3	OUT	3.3 V	3.3 V power supply
S1	GNDM1	-	-	Ground
S2	GNDM2	-	-	Ground

Table 24: Mini PCI Express Connector X7 (continued)

15: J9 and J10 need to be set to 2+3 to route the USB2 host interface to the Mini PCI Express connector (*Table 18*).
16: **Caution!** There is no protective circuit for the USB interfaces brought out at the Mini PCI Express connector (X7).
17: The 1.5 V voltage can be switched OFF with signal X_CSI1_DATA09/EN_VCC1V5.

4.6.7 Audio/Video connectors (X13 and X14)

The Audio/Video (A/V) connectors X13 and X14 provide an easy way to add typical A/V functions and features to the phyBOARD-Mira. Standard interfaces such as parallel display, I^2S and I^2C as well as different supply voltages are available at the two A/V female dual entry connectors. Special feature of these connectors are their connectivity from the bottom or the top. The pinout of the A/V connectors is shown in *Table 25* and *Table 26.*

The A/V connector is intended for use with phyBOARD Expansion Boards[18], and to add specific audio/video connectivity with custom expansion boards.

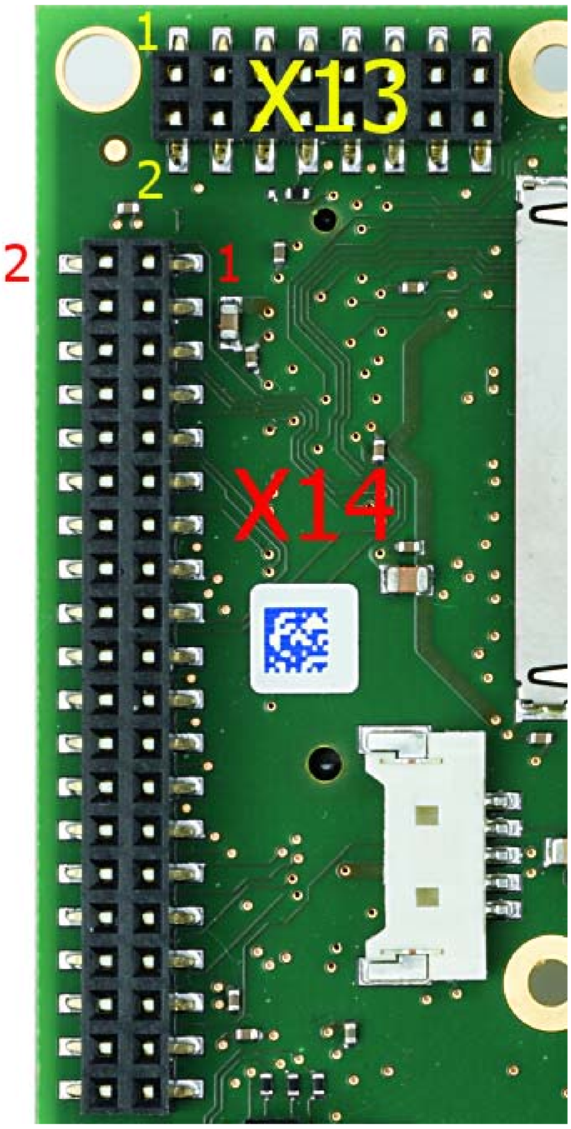


Figure 21: Audio/Video Connectors (X13and X14)

A/V connector X14makes all signals for display connectivity available, while X13 provides signals for audio and touch screen connectivity, as well as an I^2C bus and additional control signals.

18: Please find additional information on phyBOARD Expansion Boards in the corresponding application guide (L-793e).

Pin #	Signal Name	Type	SL	Description
1	GND	-	-	Ground
2	X_LCD_DATA16	OUT	3.3V	LCD Data 16 – red 0
3	X_LCD_DATA17	OUT	3.3 V	LCD Data 17 – red 1
4	X_LCD_DATA18	OUT	3.3 V	LCD Data 18 – red 2
5	X_LCD_DATA19	OUT	3.3 V	LCD Data 19 – red 3
6	GND	-	-	Ground
7	X_LCD_DATA20	OUT	3.3 V	LCD Data 20 – red 4
8	X_LCD_DATA21	OUT	3.3 V	LCD Data 21 – red 5
9	X_LCD_DATA22	OUT	3.3 V	LCD Data 22 – red 6
10	X_LCD_DATA23	OUT	3.3 V	LCD Data 23 – red 7
11	GND	-	-	Ground
12	X_LCD_DATA8	OUT	3.3 V	LCD Data 8 – green 0
13	X_LCD_DATA9	OUT	3.3 V	LCD Data 9 – green 1
14	X_LCD_DATA10	OUT	3.3 V	LCD Data 10 – green 2
15	X_LCD_DATA11	OUT	3.3 V	LCD Data 11 – green 3
16	GND	-	-	Ground
17	X_LCD_DATA12	OUT	3.3 V	LCD Data 12 – green 4
18	X_LCD_DATA13	OUT	3.3 V	LCD Data 13 – green 5
19	X_LCD_DATA14	OUT	3.3 V	LCD Data 14 – green 6
20	X_LCD_DATA15	OUT	3.3 V	LCD Data 15 – green 7
21	GND	-	-	Ground
22	X_LCD_DATA00	OUT	3.3 V	LCD Data 00 – blue 0
23	X_LCD_DATA01	OUT	3.3 V	LCD Data 01 – blue 1
24	X_LCD_DATA02	OUT	3.3 V	LCD Data 02 – blue 2
25	X_LCD_DATA03	OUT	3.3 V	LCD Data 03 – blue 3
26	GND	-	-	Ground
27	X_LCD_DATA04	OUT	3.3 V	LCD Data 04 – blue 4
28	X_LCD_DATA05	OUT	3.3 V	LCD Data05 – blue 5
29	X_LCD_DATA06	OUT	3.3 V	LCD Data 06 – blue 6
30	X_LCD_DATA07	OUT	3.3 V	LCD Data 07 – blue 7
31	GND	-	-	Ground
32	X_LCD_CLK	OUT	3.3 V	LCD Pixel Clock
33	X_LCD_ENABLE	OUT	3.3 V	LCD BIAS ENABLE

Table 25: PHYTEC A/V connector X14

Pin #	Signal Name	Type	SL	Description
34	X_LCD_HSYNC	OUT	3.3 V	LCD Horizontal Synchronization
35	X_LCD_VSYNC	OUT	3.3 V	LCD Vertical Synchronization
36	GND	-	-	Ground
37	GND	-	-	Ground
38	X_PWM1_OUT	OUT	3.3 V	Pulse Width Modulation
39	VCC_BL	OUT	NS	Backlight power supply[19]
40	VCC5V	OUT	5.0 V	5.0 V power supply

Table 25: PHYTEC A/V connector X14 (continued)

Pin #	Signal Name	Type	SL	Description	
1	X_AUD5_TXC	I/O	3.3 V	I²S Clock	
2	X_AUD5_TXFS	I/O	3.3 V	I²S Frame	
3	X_AUD5_RXD	I/O	3.3 V	I²S Analog-Digital converter (microphone)	
4	X_AUD5_TXD	I/O	3.3 V	I²S Digital-Analog converter (speaker)	
5	X_CSI1_DATA07/AV_INT	I/O	3.3 V	A/V interrupt, GPIO3_02[20]	
6	X_CSI1_DATA10/LCD_PWCTRL	OUT	3.3 V	LCD control, GPIO3_22	
7	GND	-	-	Ground	
8	X_nRESET	OUT	3.3 V	Reset[21]	(J1 1+2)
	X_LCD_RESET	OUT	3.3 V	Reset LCD	(**J1 2+3**)
9	TS_X+	IN	1.8 V	Touch X+	
10	TS_X-	IN	1.8 V	Touch X-	
11	TS_Y+	IN	1.8 V	Touch Y+	
12	TS_Y-	IN	1.8 V	Touch Y-	
13	VCC3V3	OUT	3.3 V	3.3 V power supply	
14	GND	-	-	Ground	
15	X_I2C1_SCL	I/O	3.3 V	I2C1 Clock	
16	X_I2C1_SDA	I/O	3.3 V	I2C1 Data	

Table 26: PHYTEC A/V connector X13

Jumper J1 connects either signal X_LCD_RESET or signal X_nRESET to pin 8 of X13. The following table shows the available configurations:

19: Voltage level is not specified and depends on the connected power module and the attached voltage.
20: **Note:** The A/V interrupt must be enabled with signal X_CSI1_DATA05_EN_SWITCH (GPIO3_04).
21: Reset signal depends on J1, please refer to *Table 27* for more information.

J1	Description
1+2	X_nRESET
2+3	**X_LCD_RESET**

Table 27:   A/V Jumper Configuration J1

4.6.7.1 Software Implementation

4.6.7.1.1 Framebuffer

The BSP is already prepared for use with the 7" ET0700 display connected to the A/V connector using the PEB-AV-02 display adapter. To activate the A/V connector with the phyBOARD Expansion Board you must enter the barebox and type the following command:

```
edit /env/config-expansions
```

Following line must be added at the end:

```
. /env/expansions/imx6qdl-phytec-lcd-018-peb-av-02
```

Save the file with ***CTRL+D*** and ***restart*** the phyBOARD-Mira**.**

This driver gains access to the display via device node /dev/fb0 for PHYTEC display connector. A simple test of the framebuffer feature can then be run with:

```
fbtest
```

This will show various pictures on the display.

4.6.7.1.2 Touch

A simple test of this feature can be run with

```
ts_calibrate
```

to calibrate the touch (only needed for resistive touch screens)and

```
ts_test
```

to do a simple application using this feature.

4.6.8 Expansion Connector (X17)

The expansion connector X17 provides an easy way to add other functions and features to the phyBOARD-Mira. Standard interfaces such as UART, SPI and I^{2}C as well as different supply voltages and some GPIOs are available at the expansion female connector.

The expansion connector is intended for use with phyBOARD Expansion Boards[22], and to add specific functions with custom expansion boards.

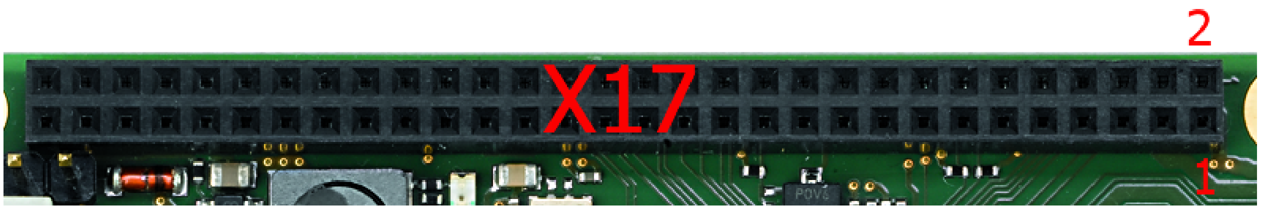


Figure 22: Expansion Connector (X17)

The pinout of the expansion connector is shown in the following table.

Pin #	Signal Name	Type	SL	Description
1	VCC3V3	OUT	3.3 V	3.3 V power supply
2	VCC5V	OUT	5.0 V	5.0 V power supply
3	VCC1V5	OUT	1.5 V	1.5 V power supply[23]
4	GND	-	-	Ground
5	X_ECSPI1_SS0	OUT	3.3 V	SPI 1 chip select 0
6	X_ECSPI1_MOSI	OUT	3.3 V	SPI 1 master output/slave input
7	X_ECSPI1_MISO	IN	3.3 V	SPI 1 master input/slave output
8	X_ECSPI1_SCLK	OUT	3.3 V	SPI 1 clock output
9	GND	-	-	Ground
10	X_UART2_RX_DATA	IN	3.3 V	UART 2 receive data (standard debug interface)
11	X_I2C1_SDA	I/O	3.3 V	I2C 1 Data
12	X_UART2_TX_DATA	OUT	3.3 V	UART 2 transmit data (standard debug interface)
13	X_I2C1_SCL	I/O	3.3 V	I2C 1 Clock
14	GND	-	-	Ground
15	X_JTAG_TMS	IN	3.3 V	JTAG Chain Test Mode Select signal

Table 28: PHYTEC Expansion Connector X17

22: Please find additional information on phyBOARD-Mira i.MX 6 Expansion Boards in the corresponding application guide (L-793e).
23: The 1.5 V voltage can be switched OFF with signal X_CSI1_DATA09/EN_VCC1V5.

16	X_JTAG_TRSTB	IN	3.3 V	JTAG Chain Test Reset	
17	X_JTAG_TDI	IN	3.3 V	JTAG Chain Test Data Input	
18	X_JTAG_TDO	OUT	3.3 V	JTAG Chain Test Data Output	
19	GND	-	-	Ground	
20	X_JTAG_TCK	IN	3.3 V	JTAG Chain Test Clock signal	
21	X_USB1_DP_EXP	I/O	DIFF	USB host data +[24, 25]	
22	X_USB1_DM_EXP	I/O	DIFF	USB host data -[24, 25]	
23	X_nRESET	OUT	3.3 V	Reset	
24	GND	-	-	Ground	
25	X_SD3_CMD	I/O	3.3 V	SD/MMC command	
26	X_SD3_DATA0	I/O	3.3 V	SD/MMC data 0	
27	X_SD3_CLK	I/O	3.3 V	SD/MMC clock	
28	X_SD3_DATA1	I/O	3.3 V	SD/MMC data 1	
29	GND	-	-	Ground	
30	X_SD3_DATA2	I/O	3.3 V	SD/MMC data 2	
31	X_CSI0_DAT11/ECSPI2_SS0	I/O	3.3 V	SPI2 chip select 0, UART1 RX, GPIO5_29	
32	X_SD3_DATA3	I/O	3.3 V	SD/MMC data 3	
33	X_CSI0_DAT10/ECSPI2_MISO	I/O	3.3 V	SPI2 MISO, UART1 TX, GPIO5_28	
34	GND	-	-	Ground	
35	X_SD3_DATA4	I/O	3.3 V	SD/MMC data 4, UART2 RX[26]	
36	X_SD3_DATA5	I/O	3.3 V	SD/MMC data 5, UART2 TX	
37	X_SATA_TXP	OUT	DIFF	SATA transmit positive	
38	X_SD3_DATA6	I/O	3.3 V	SD/MMC data 6	
39	X_SATA_TXN	OUT	DIFF	SATA transmit negative	
40	X_SD3_DATA7	I/O	3.3 V	SD/MMC data 7	
41	GND	-	-	Ground	
42	X_ECSPI2_RDY/nPMON_PWRFAIL	OUT	3.3 V	Power fail signal	(**J26 1+2**)
	X_SIM_VPP	-	-	UIM_VPP	(J26 2+3)
43	X_SATA_RXP	IN	DIFF	SATA receive positive	
44	X_CSI0_DAT8/ECSPI2_SCLK	OUT	3.3 V	SPI 2 clock output	(**J21 1+2**)
	X_SIM_VCC	OUT	-	UIM VCC	(J21 2+3)

Table 28: PHYTEC Expansion Connector X17 (continued)

24: J5 and J6 need to be set to 2+3 to route the USB1 OTG interface to the expansion connector (*Table 17*).
25: **Caution!** There is no protective circuit for the USB interfaces brought out at the expansion connector (X17).
26: **Caution!** If the UART2 signals are redirected to these pins the standard console is not available on the Evaluation Board (PEB-EVAL-01).

45	X_SATA_RXN	IN	DIFF	SATA receive negative	
46	GND	-	-	Ground	
47	X_FLEXCAN1_TX_EXP	OUT	3.3 V	CAN 1 transmit data[27]	
48	X_FLEXCAN1_RX_EXP	IN	3.3 V	CAN 1 receive data[27]	
49	X_USB_OTG_OC/FLEXCAN2_TX	OUT	3.3 V	CAN 2 transmit data	
50	X_USB_OTG_PWR/FLEXCAN2_RX	IN	3.3 V	CAN 2 receive data	
51	GND	-	-	Ground	
52	X_CSI0_DAT9/ECSPI2_MOSI	OUT	3.3 V	SPI 2 MOSI	(**J25 1+2**)
	X_SIM_RST	OUT	-	UIM Reset	(J25 2+3)
53	X_USB1_ID	IN	3.3 V	USB 1 identification	
54	X_USB1_VBUS	OUT	5.0 V	USB 1 bus voltage	
55	X_USB_OTG_CHD_B	OUT	3.3 V	USB 1 charger enable	(**J23 1+2**)
	X_SIM_IO	OUT	-	UIM Data	(J23 2+3)
56	GND	-	-	Ground	
57	VCC_BL	OUT	NS	Backlight power supply[28]	
58	X_ECSPI2_SS1	OUT	3.3 V	SPI 2 chip select 1	(**J22 1+2**)
	X_SIM_CLK	OUT	-	UIM Clock	(J22 2+3)
59	GND	-	-	Ground	
60	VCC5V_IN	IN	5.0 V	5 V input supply voltage	

Table 28: PHYTEC Expansion Connector X17 (continued)

4.6.8.1 Software Implementation

4.6.8.1.1 UART Connectivity

Only /dev/ttymxc1 for UART2 and /dev/ttymxc2 for UART3 (at X23) have been implemented within the BSP.

The standard console UART2 is accessible as /dev/ttymxc1 and is mainly used for debugging and control of software updates.

Usage of */dev/ttymxc0* for UART1 or use of UART2 at pins 35 and 36 of the expansion connector requires changing of the pin muxing and additional software development.

27 : Please refer to section 4.6.3 to make CAN 1 available on expansion connector
28 : Voltage level is not specified and depends on the connected power module and the attached voltage.

4.6.8.1.2 SPI Connectivity

The driver for SPI can be accessed by its API within the kernel. If you connect a chip to SPI, you should implement its driver into the kernel, as well. The SPI driver offers no /dev/spi entry in userspace, because using it would mean to place the driver for your chip in userspace, too, what most probably would not be useful. Please note that the BSP already includes a couple of deactivated drivers for various chips. These drivers might be helpful for you even though they are usually not tested.

If you really want to access SPI from userland, you can use SPIDEV in the linux kernel. See the kernel documentation in *Documentation/spi/spidev*.

4.6.8.1.3 I^{2}C Connectivity

Please refer to *section 4.6.4*.

4.6.8.1.4 User programmable GPIOs

There are different user programmable GPIOs supported in the BSP for the phyBOARD-Mira. The signals are available on the expansion connector or the corresponding expansion-boards, e.g. PEB-EVAL-01. For more information please refer to the Application Guide for phyBOARD Expansion Boards.

For setting and resetting the green user LED on the expansion board (or an LED connected to pin 31 of the expansion connector) just enter:

```
cd /sys/class/leds/user_led_green/
echo 1 > brightness
echo 0 > brightness
```

For setting and resetting the red user LED on the expansion board (or an LED connected to pin 35 of the expansion connector) just enter:

```
cd /sys/class/leds/user_led_red/
echo 1 > brightness
echo 0 > brightness
```

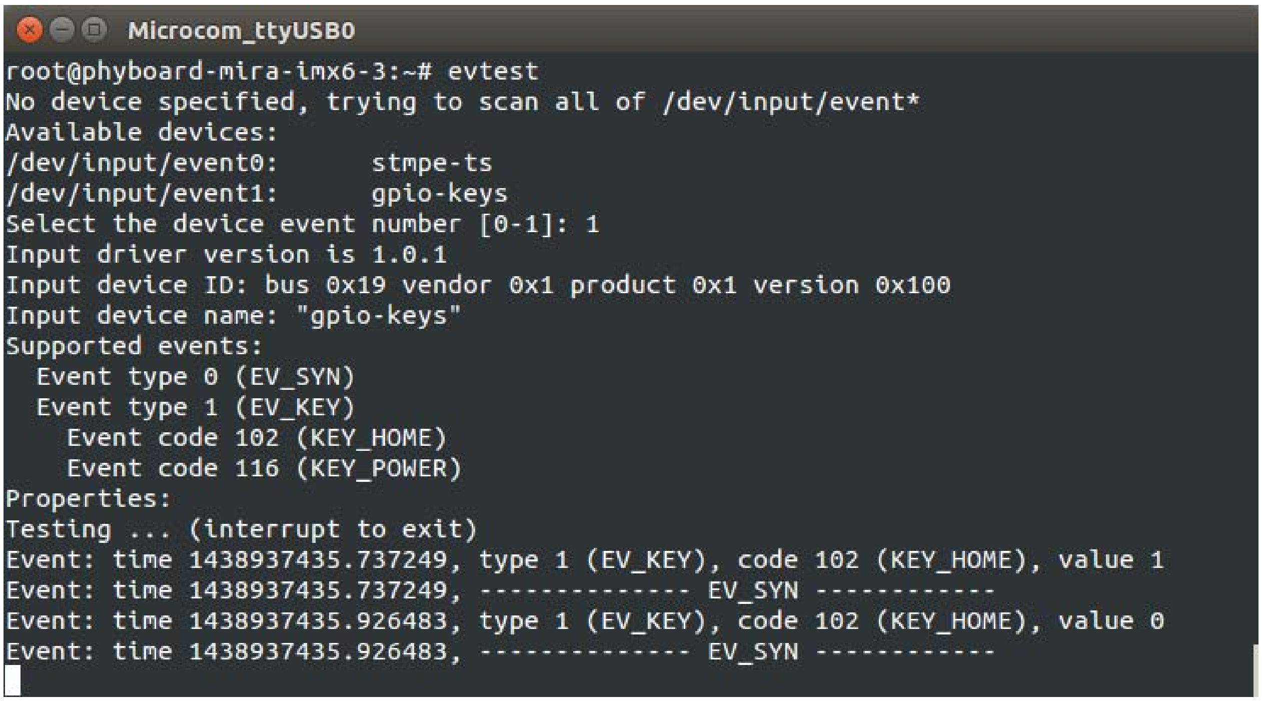
For setting and resetting the yellow user LED on the expansion board (or an LED connected to pin 36 of the expansion connector) just enter:

```
cd /sys/class/leds/user_led_yellow/
echo 1 > brightness
echo 0 > brightness
```

For getting values from the user buttons S2, or S3 on the expansion board (or buttons connected to pin 33 and pin 38 of the expansion connector) just enter

```
evtest
```

This command returns a list of available events and expects an input. After entering the correct number for the *gpio-keys* event you will get information about the input device selected. Now press any button on the Evaluation Board (PEB-EVAL-01) and observe the change at the input device.



```
root@phyboard-mira-imx6-3:~# evtest
No device specified, trying to scan all of /dev/input/event*
Available devices:
/dev/input/event0:      stmpe-ts
/dev/input/event1:      gpio-keys
Select the device event number [0-1]: 1
Input driver version is 1.0.1
Input device ID: bus 0x19 vendor 0x1 product 0x1 version 0x100
Input device name: "gpio-keys"
Supported events:
  Event type 0 (EV_SYN)
  Event type 1 (EV_KEY)
    Event code 102 (KEY_HOME)
    Event code 116 (KEY_POWER)
Properties:
Testing ... (interrupt to exit)
Event: time 1438937435.737249, type 1 (EV_KEY), code 102 (KEY_HOME), value 1
Event: time 1438937435.737249, -------------- EV_SYN ------------
Event: time 1438937435.926483, type 1 (EV_KEY), code 102 (KEY_HOME), value 0
Event: time 1438937435.926483, -------------- EV_SYN ------------
```

⚠	S1 on the Evaluation Board is not supported by the BSP for the phyBOARD-Mira.

5 Revision History

Date	Version #	Changes in this manual
22.02.2015	Manual L-806e_0	Preliminary edition. Describes the phyCORE-i.MX 6 SOM (PCB 1429.0) with phyBOARD-Mira-Carrier Board (PCB 1434.0)
31.08.2015	L-806e_1	Describes the phyCORE-i.MX 6 SOM (PCB 1429.1) with phyBOARD-Mira Carrier Board (PCB 1434.1), the phyBOARD-Mira BSP (PD15.2.0) and the Live System (U-KPB-01501-xxx_SO-536v2)

Index

Document:	**phyBOARD-Mira i.MX 6**
Document number:	**L-806e_1, August 2015**

How would you improve this manual?

Did you find any mistakes in this manual? **page**

Submitted by:

Customer number:	
Name:	
Company:	
Address:	

Return to:

PHYTEC Messtechnik GmbH
Postfach 100403
D-55135 Mainz, Germany
Fax : +49 (6131) 9221-33

Published by

© PHYTEC Messtechnik GmbH 2015

Ordering No. L-806e_1

Printed in Germany