phyBOARD®-Wega AM335x

Application Guide

Document No.:	**L-792e_1**
SBC Prod. No..:	**PB-00802-xxx**
CB PCB No.:	**1405.1, 1405.2**
SOM PCB No.:	**1397.1**

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-00802-xxx Single Board Computer (SBC) in the following referred to as phyBOARD-Wega AM335x. The manual specifies the phyBOARD-Wega AM335x's design and function. Precise specifications for the Texas Instruments AM335x microcontrollers can be found in the Texas Instrumenten's AM335x 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^{2}C 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-Wega 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 Texas Instruments AM335x 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-CD-02 /phyBOARD-Wega AM335x
SMT	Surface mount technology
SOM	System on Module; used in reference to the PCL-051 /phyCORE-AM335x 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-Wega AM335x 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 AM335x 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 hyBOARD-Wega AM335x 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 AM335x-PD14.1-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-Wega AM335x 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-Wega AM335x 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-Wega, 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-Wega is the phyCORE-AM335x System on Module (SOM). As with all SBCs of the phyBOARD® family the SOM is directly soldered onto the carrier board PCB for routing of signals from the SOM to applicable I/O interfaces. This “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-Wega. 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-Wega, 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-Wega AM335x Development Kit:	**KPB-00802-xxx**
phyBOARD-Wega AM335x SBC:	**PB-00802-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-Wega-AM335x.html or
http://www.phytec.eu/europe/support/faq/faq-phyBOARD-Wega-AM335x.html

Other Products and Development Support

Aside of the new phyBOARD® family, 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-Wega AM335x

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 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.

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.

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-Wega for phyCORE-AM335x is a low-cost, feature-rich software development platform supporting the Texas Instruments AM335x microcontroller. Moreover, due to the numerous standard interfaces the phyBOARD-Wega AM335x can serve as bedrock for your application. At the core of the phyBOARD-Wega is the PCL-051/phyCORE-AM335x System On Module (SOM) in a direct solder form factor, containing the processor, DRAM, NAND Flash, power regulation, supervision, transceivers, and other core functions required to support the AM335x processor. Surrounding the SOM is the PBA-CD-02/phyBOARD-Wega carrier board, adding power input, buttons, connectors, signal breakout, and Ethernet connectivity amongst other things.

The PCL-051 System On Module is a connector-less, BGA style variant of the PCM-051/phyCORE-AM335x SOM. Unlike traditional PHYTEC SOM products that support high density connectors, the PCL-051 SOM is directly soldered down to the phyBOARD-Wega using PHYTEC's Direct Solder Connect technology. This solution offers an ultra-low cost Single Board Computer for the AM335x processor, while maintaining most of the advantages of the SOM concept.

Adding the phyCORE-AM335x SOM into your own design is as simple as ordering the connectored version (PCM-051) and making use of our phyCORE Carrier Board reference schematics.

1.1.1 Features of the phyBOARD-Wega AM335x

The phyBOARD-Wega AM335x supports the following features :

- Developed in accordance with PHYTEC's new SBCplus concept (*Preface*)
- PHYTEC’s phyCORE-AM335x SOM with Direct Solder Connect (DSC)
- Pico ITX standard dimensions (100 mm × 72 mm)
- Boot from MMC or NAND Flash
- Max. 1 GHz core clock frequency
- Three different power supply options (5 V via 3.5 mm combicon or micro USB connector; or 12 V – 24 V through external power module)
- Two RJ45 jacks for 10/100 Mbps Ethernet
- One USB host interface brought out to an upright USB Standard-A connector, or at the expansion connector[1]
- One USB OTG interface available at an USB Micro-AB connector at the back side, or at the expansion connector[1]
- One Secure Digital / Multi Media Memory Card interface brought out to a Micro-SD connector at the back side

[1] : **Caution!** There is no protective circuit for the USB interface brought out at the expansion connector.

- CAN interface at 2×5 pin header 2.54 mm
- Audiocodec with Stereo Line In and Line Out (2×3 pin header 2.54 mm) and mono speaker (2-pole Molex SPOX)
- RS-232 transceiver supporting UART1 incl. handshake signals with data rates of up to 1 Mbps (2×5 pin header 2.54 mm)
- Reset-Button
- Audio/Video (A/V) connectors
- Expansion connector with different interfaces
- Backup battery supply for RTC with Gold cap (lasts approx. 17 ½ days)

1.1.2 Block Diagram

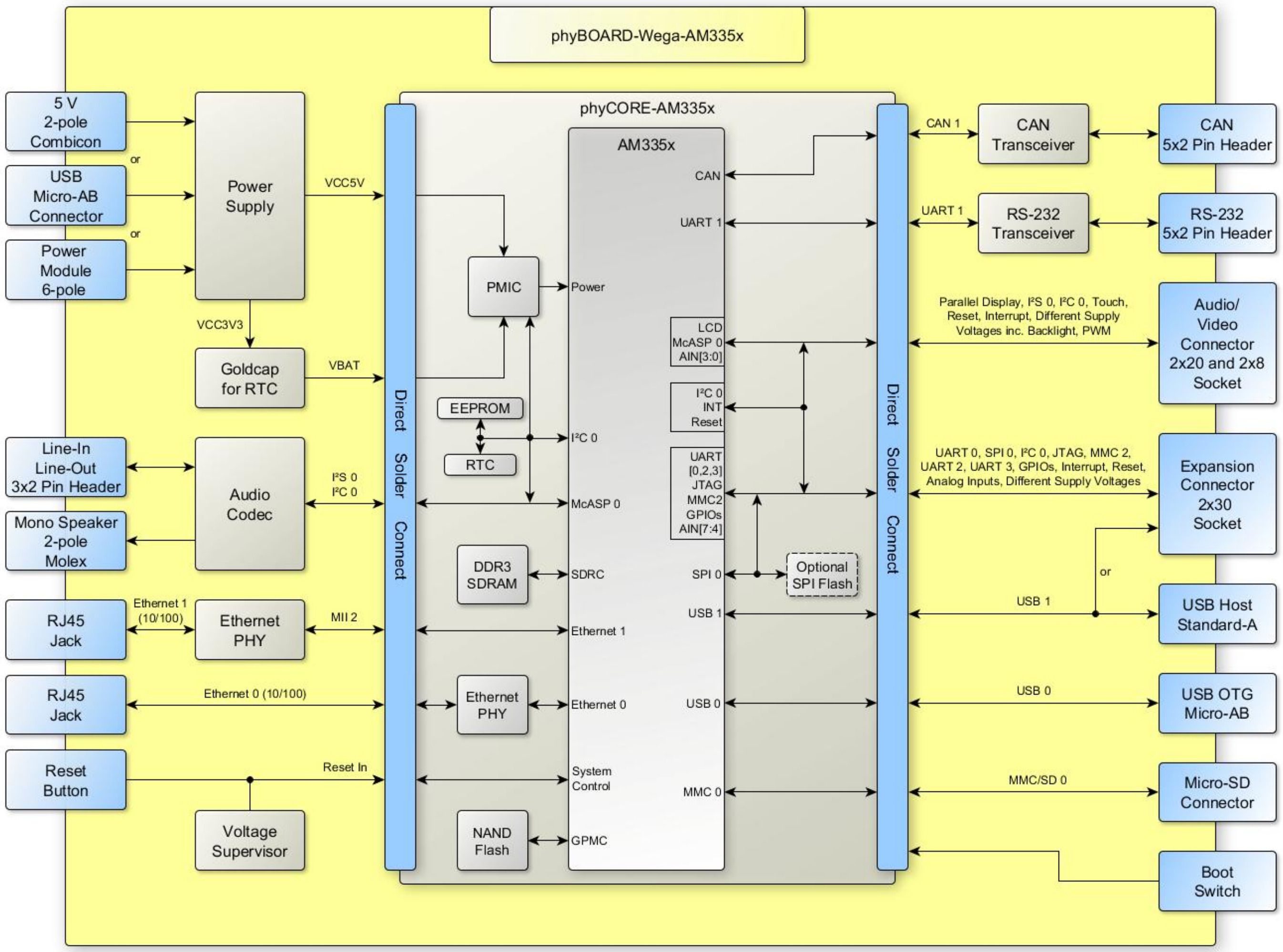


*Figure 1:	Block Diagram of the phyBOARD-Wega AM335x*

1.1.3 View of the phyBOARD-Wega AM335x

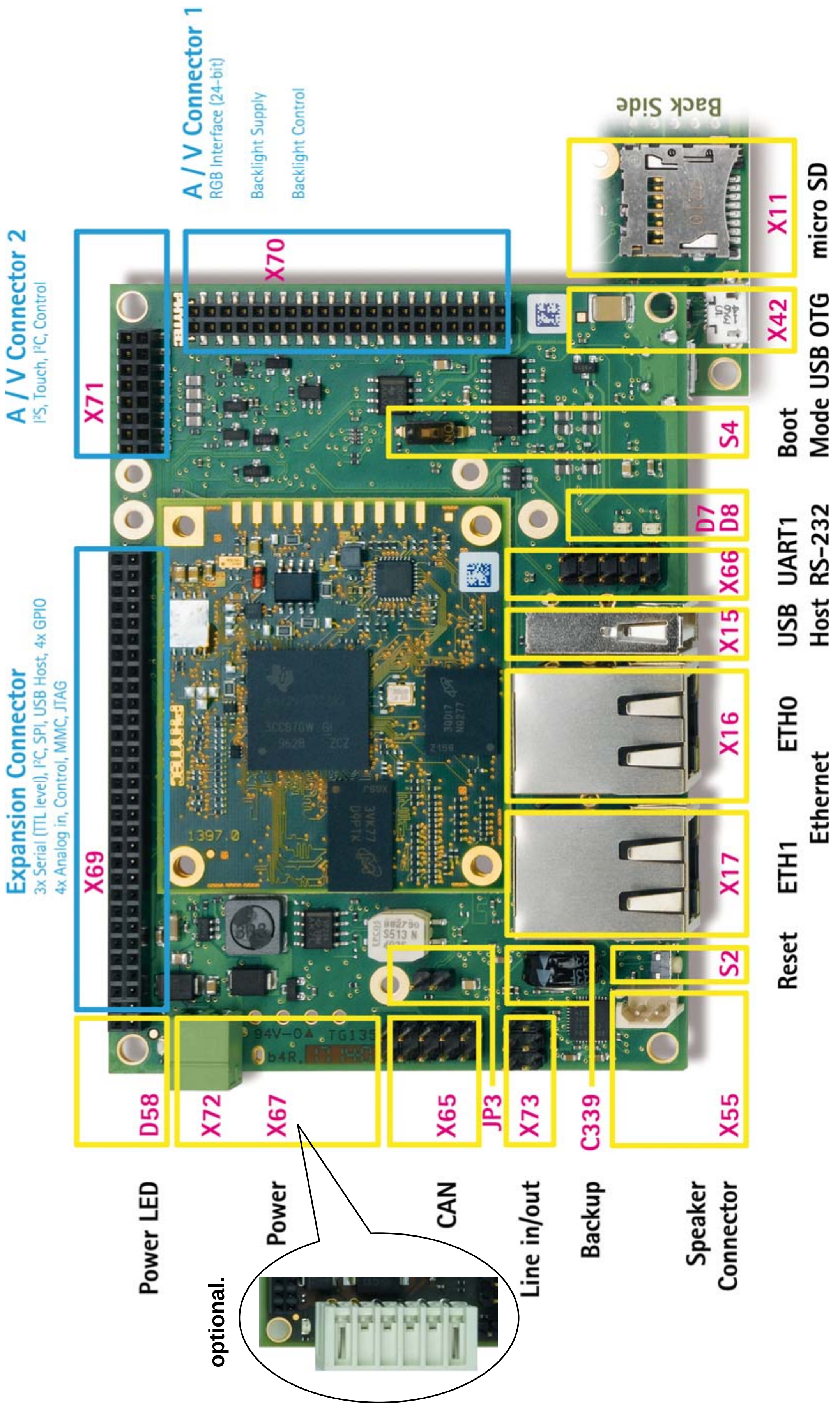


Figure 2: View of the phyBOARD-Wega AM335x

1.2 Software Overview

1.2.1 Ubuntu

Ubuntu - which you is used as operating system for our virtual machine hard disk image - 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 *14.04.2* and was released on 20 February 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.

⚠	To ensure successful introduction to the development with the phyBOARD-Wega AM335x we strongly recommend continuing with our preconfigured virtual machine hard disk image based on *Ubuntu*. Nonetheless, if you want to use your already existing environment *section 4.5 "Setup your own Linux Host PC*" explains how to modify your system to get the same experience as with our virtual machine. Please be advised that use of your already existing environment is on your own risk. We can not guarantee to successfully support you with your own environment.

2.1 Working with Eclipse

Now we start developing our own applications with the help of *Eclipse*. First we take a look on the C programming language before we go on with programming a *QT* project. At the end of this chapter we explain how to execute your written programs automatically when booting the target.

2.1.1 Programming in the C/C++ Perspective

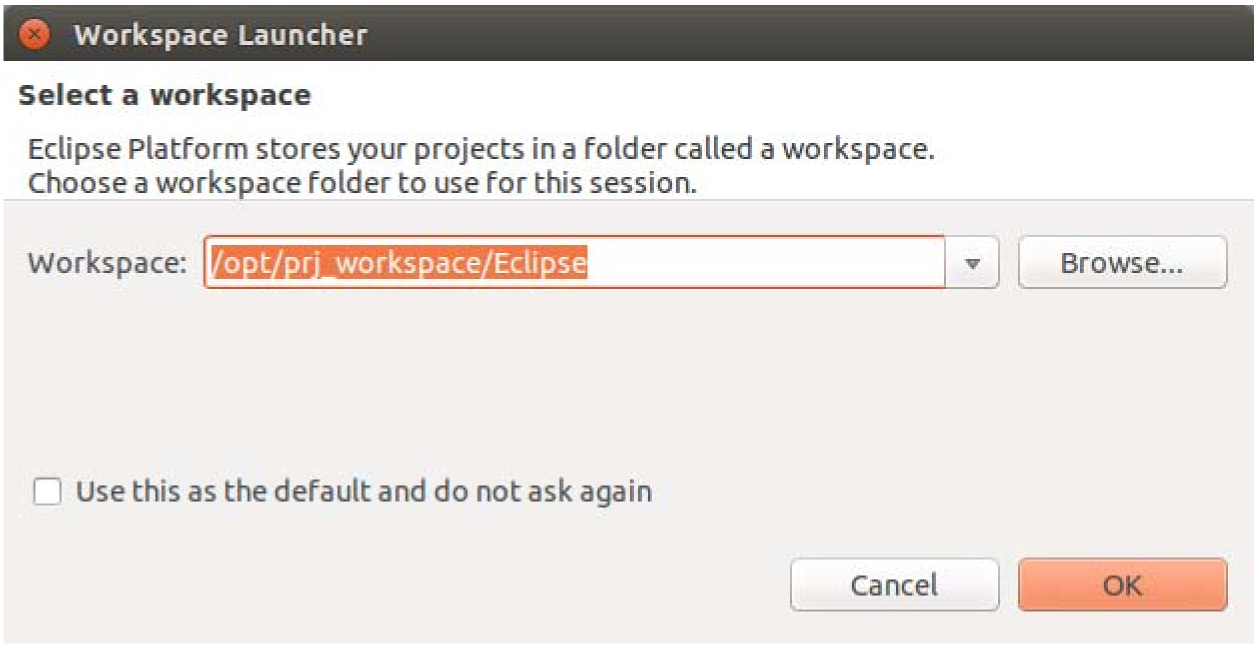
We are starting with the *C/C++* workbench. Therefore you will import an existing Eclipse project into your workspace. The imported example project will be compiled with the cross compiler. After that, you will copy and execute the newly created program on the target.

2.1.1.1 Work with the Demo Project

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

Eclipse

- Confirm the workspace directory with ***OK***.

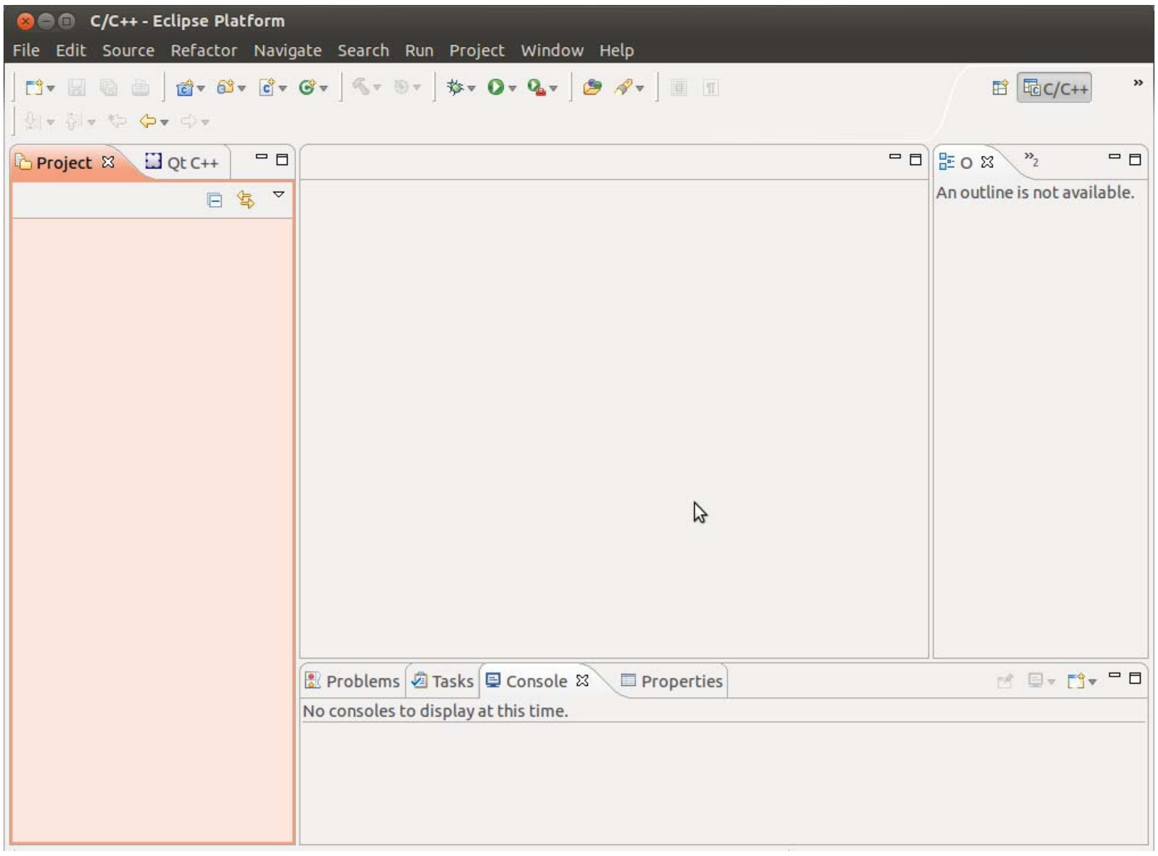

Workspace Launcher

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.

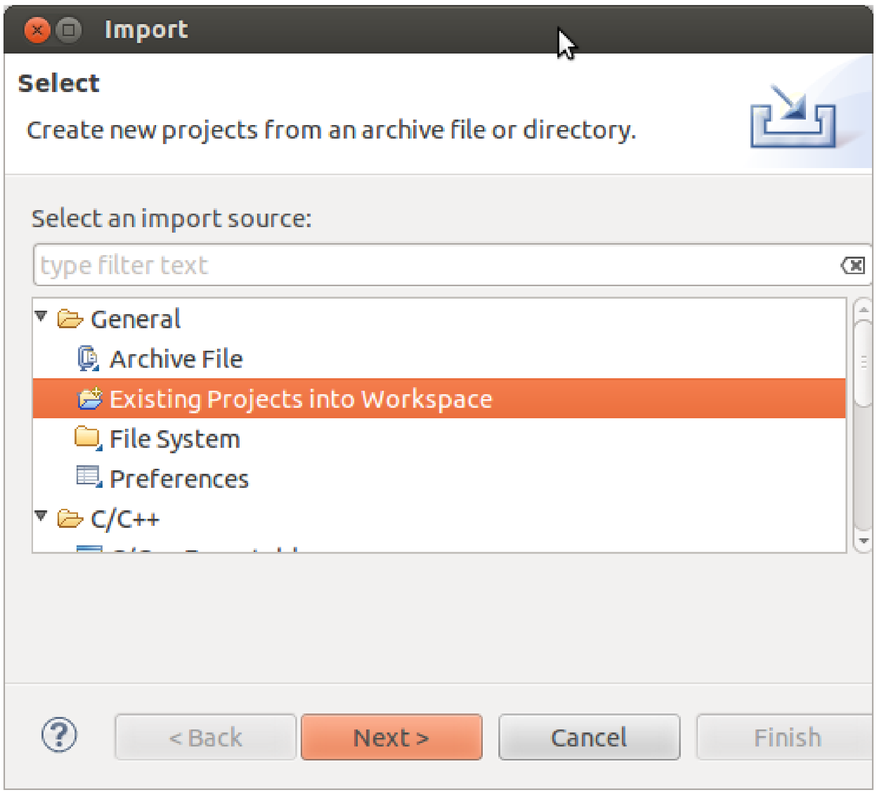


An outline is not available.

No consoles to display at this time.

First we will import an existing project.

- Select ***File*** ▶ ***Import*** from the menu bar.
- Expand ***General*** and select ***Existing Projects into Workspace.***
- Click ***Next***.



Import

Select

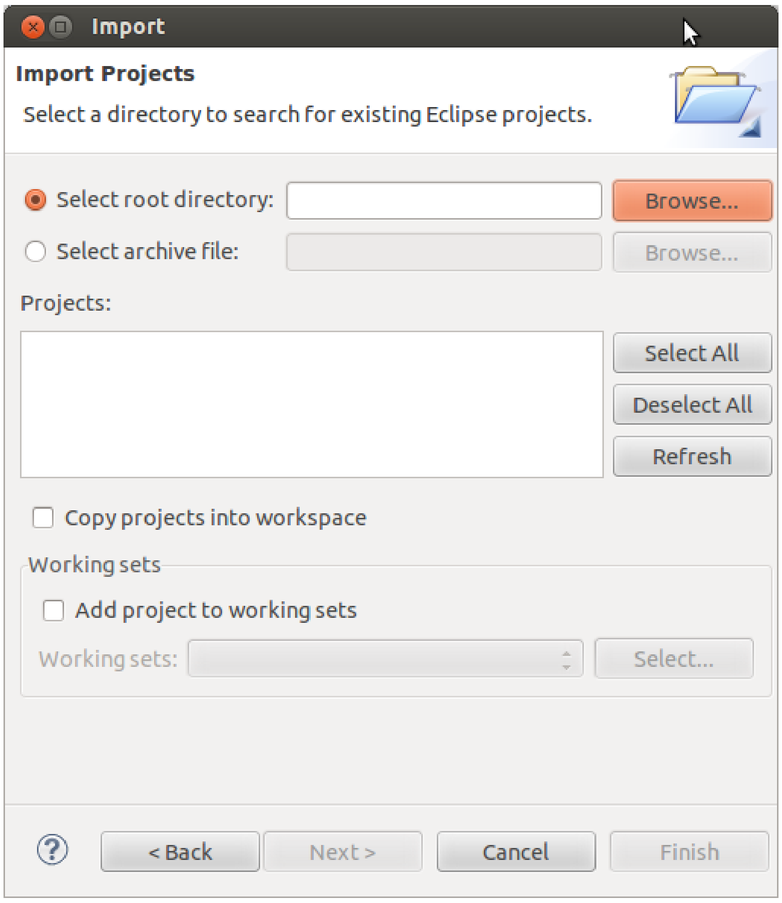
Create new projects from an archive file or directory.

Select an import source:

type filter text

- General
  - Archive File
  - Existing Projects into Workspace
  - File System
  - Preferences
- C/C++

- 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

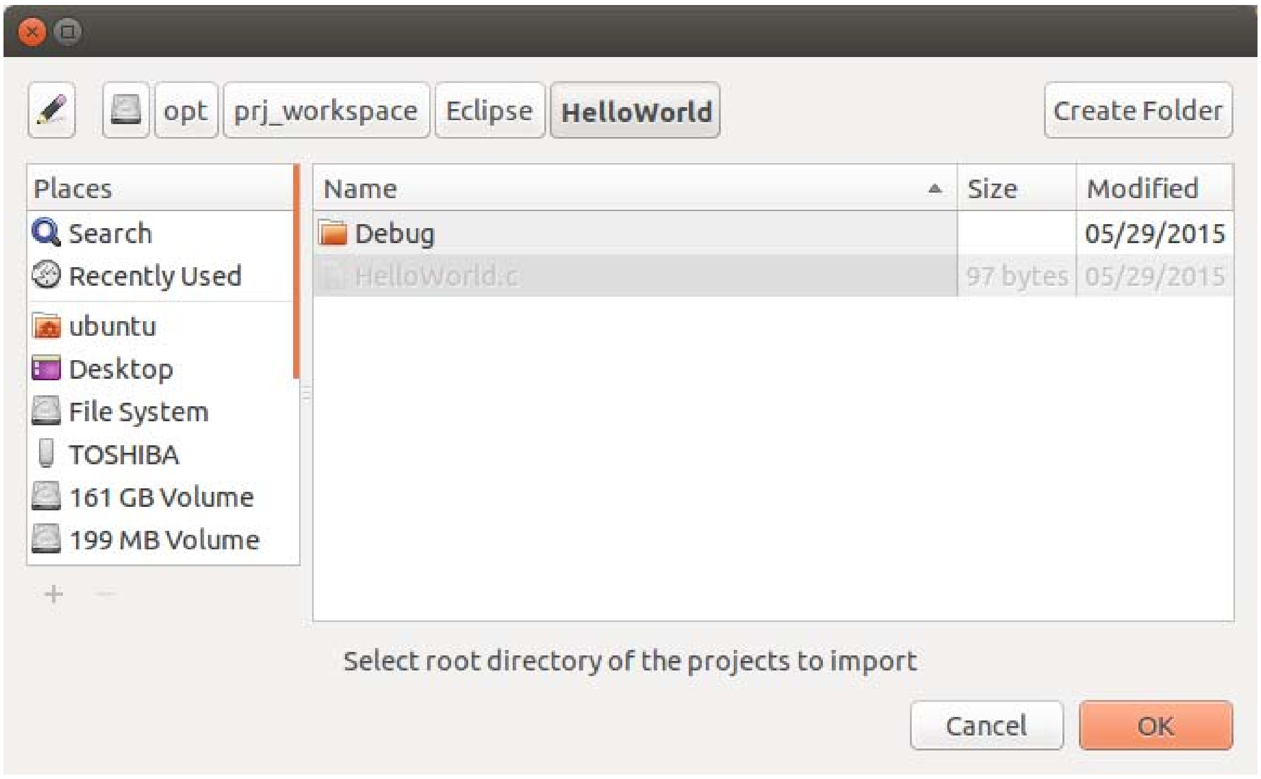
Working sets

- [ ] 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

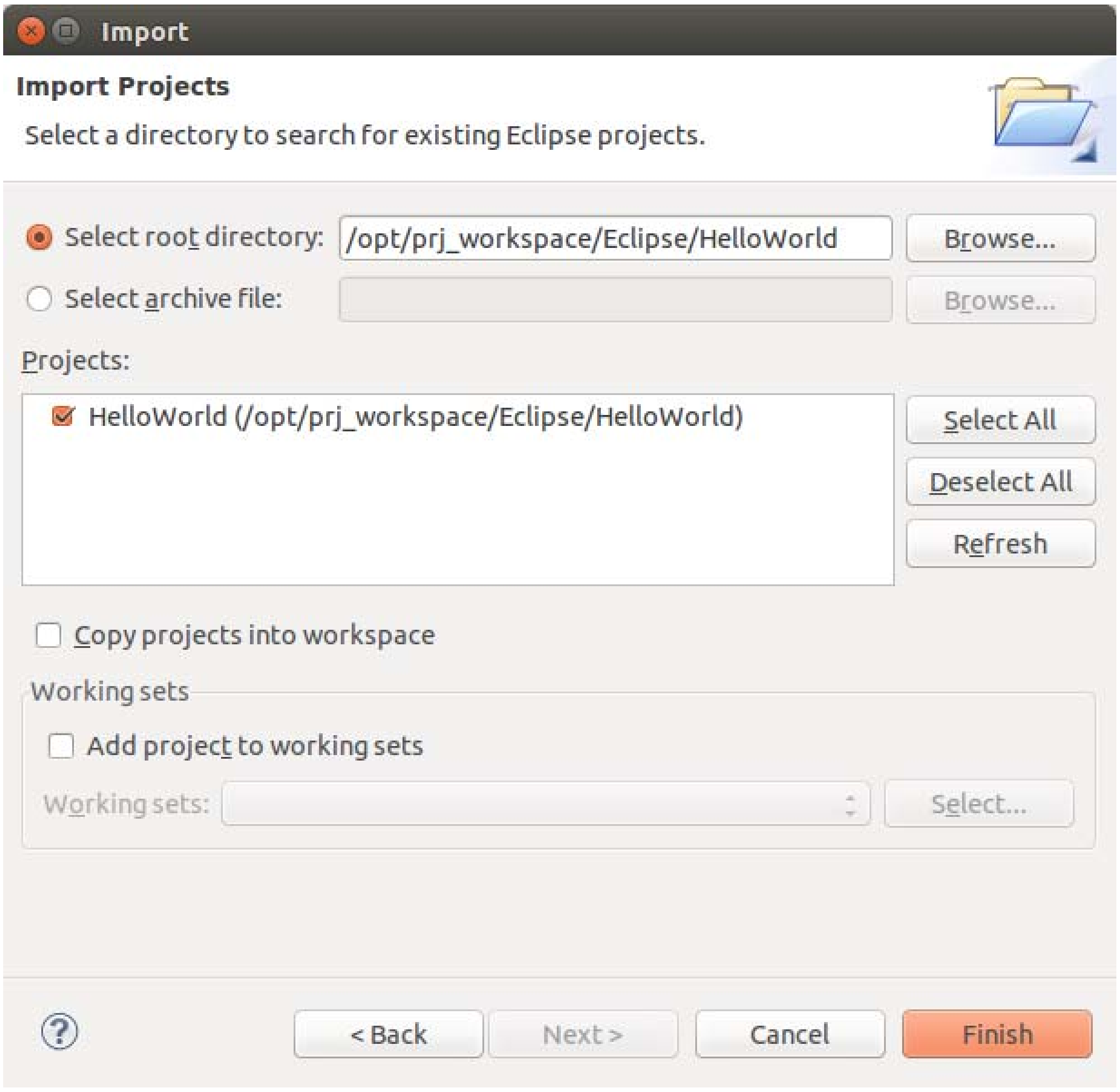
Name	Size	Modified
Debug		05/29/2015
HelloWorld.c	97 bytes	05/29/2015

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 [Browse...]

( ) Select archive file: [Browse...]

Projects:

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

[ ] Copy projects into workspace

Working sets

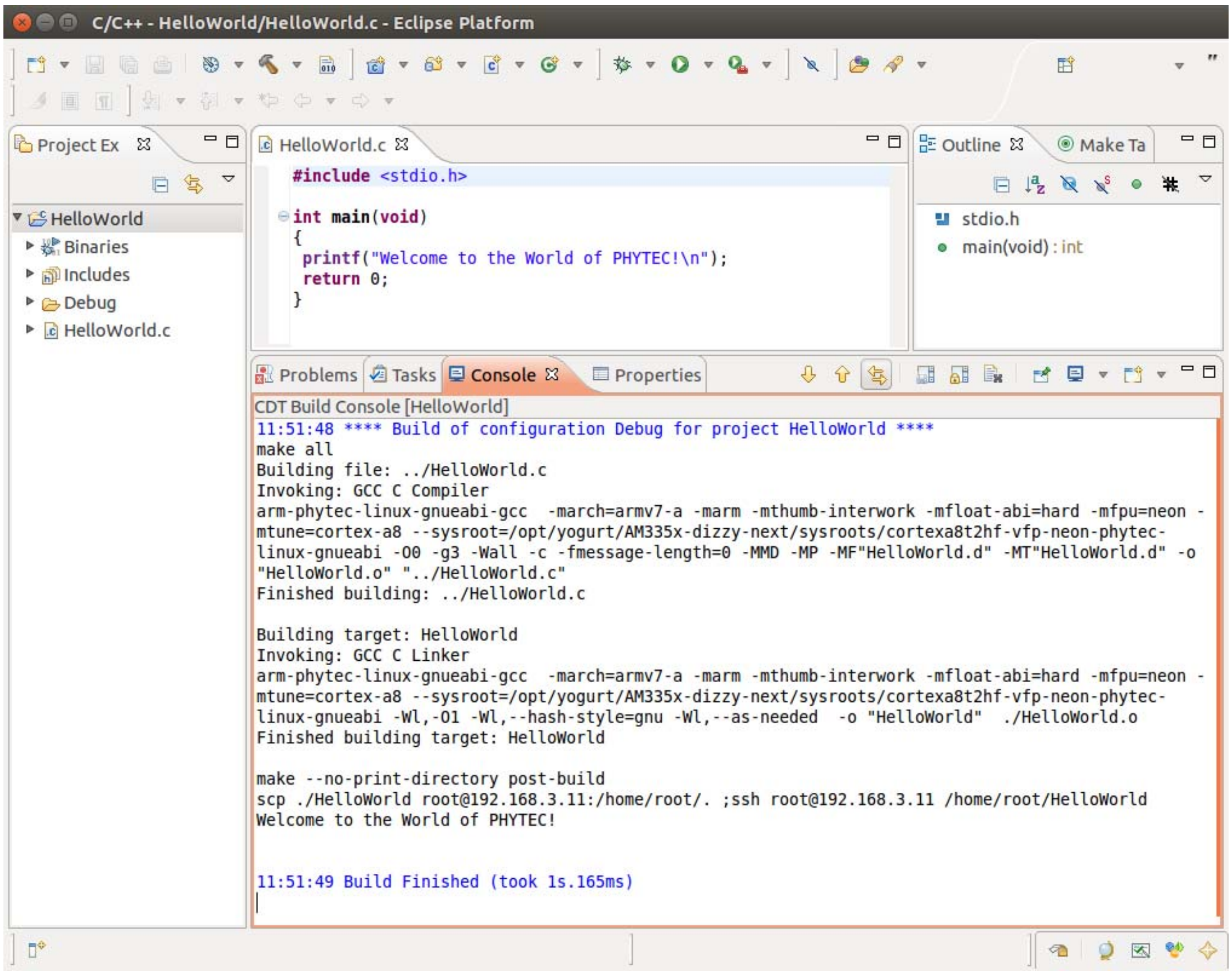
[ ] Add project to working sets

Working sets: [Select...]

- 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 a content in the *Console* window similar to the following image:



```
#include <stdio.h>

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

```
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.

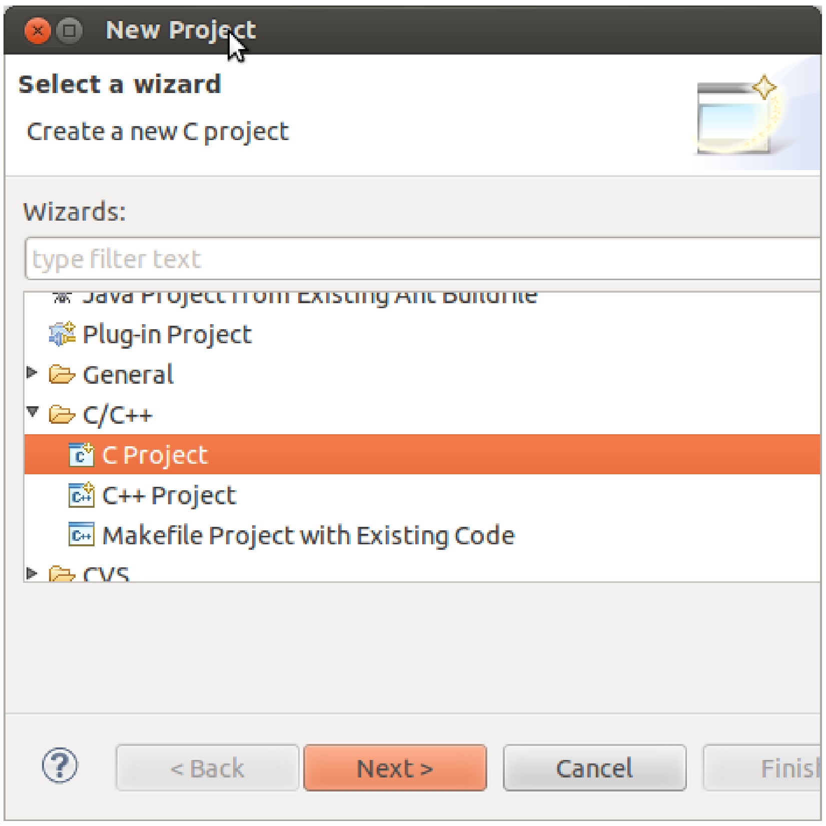
2.1.1.2 Creating a New Project

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***.


New Project

Select a wizard

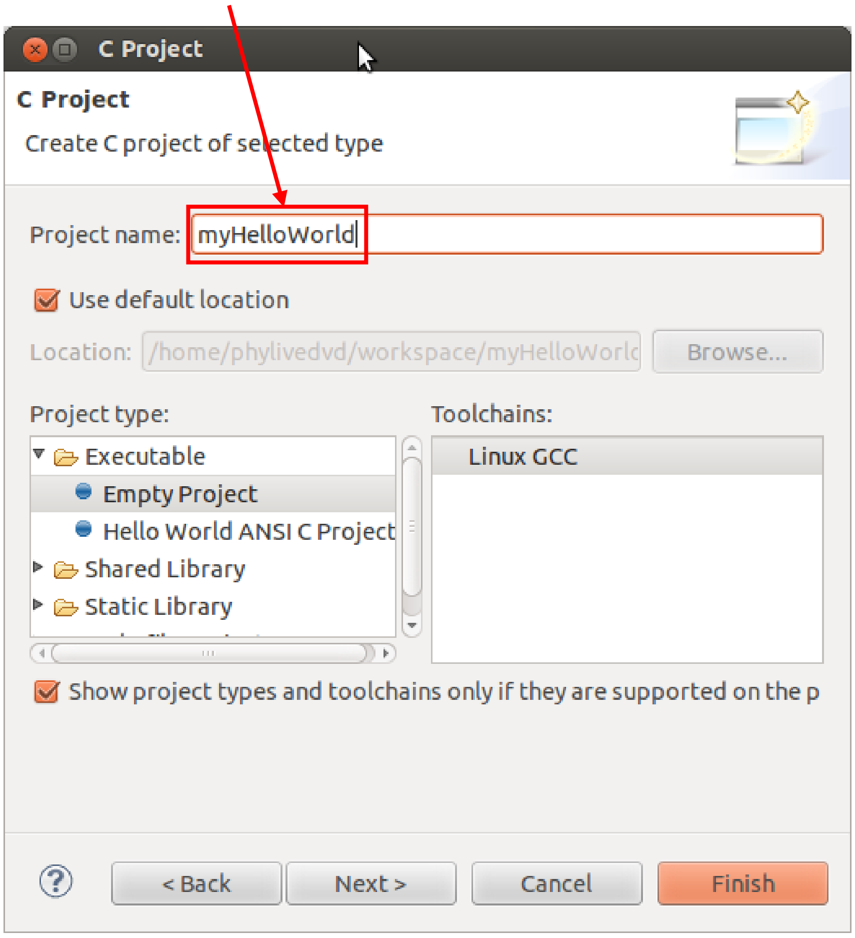
Create a new C project

Wizards:

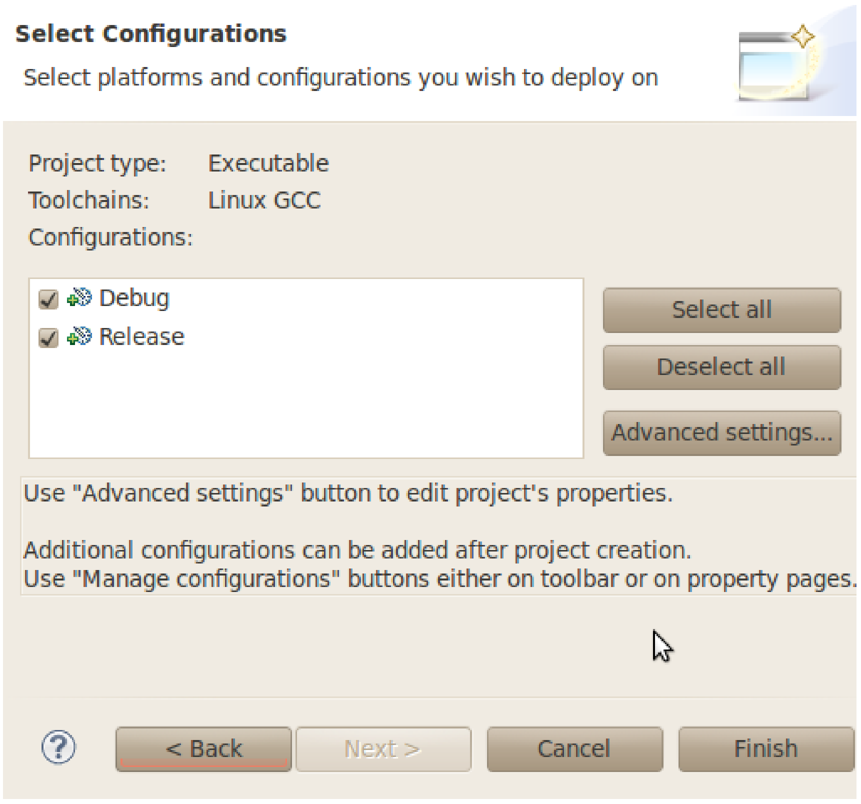
type filter text

Java Project from Existing Ant Buildfile
Plug-in Project
General
C/C++
C Project
C++ Project
Makefile Project with Existing Code
CVS

- 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/myHelloWorlc
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***.

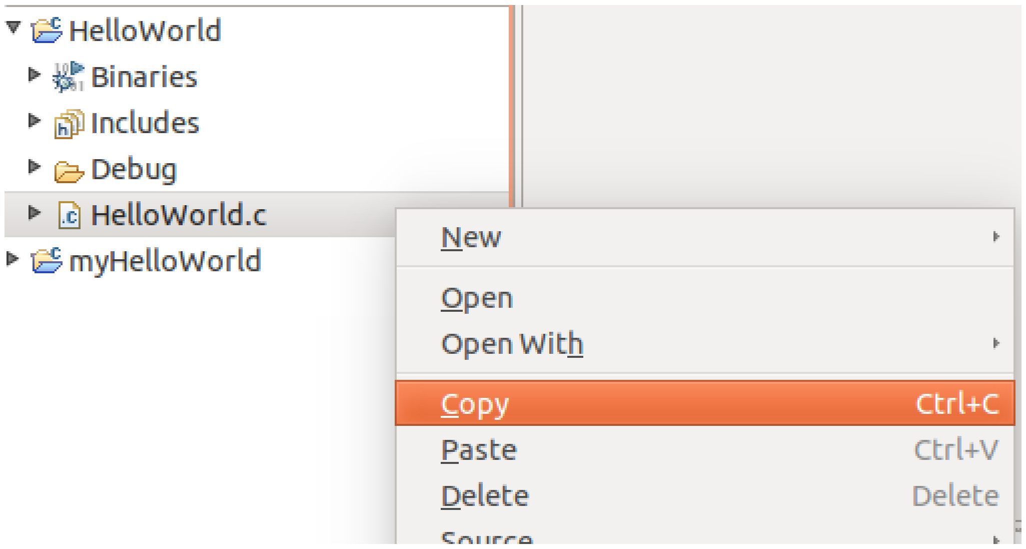


Use "Advanced settings" button to edit project's properties.

Additional configurations can be added after project creation.
Use "Manage configurations" buttons either on toolbar or on property pages.

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.

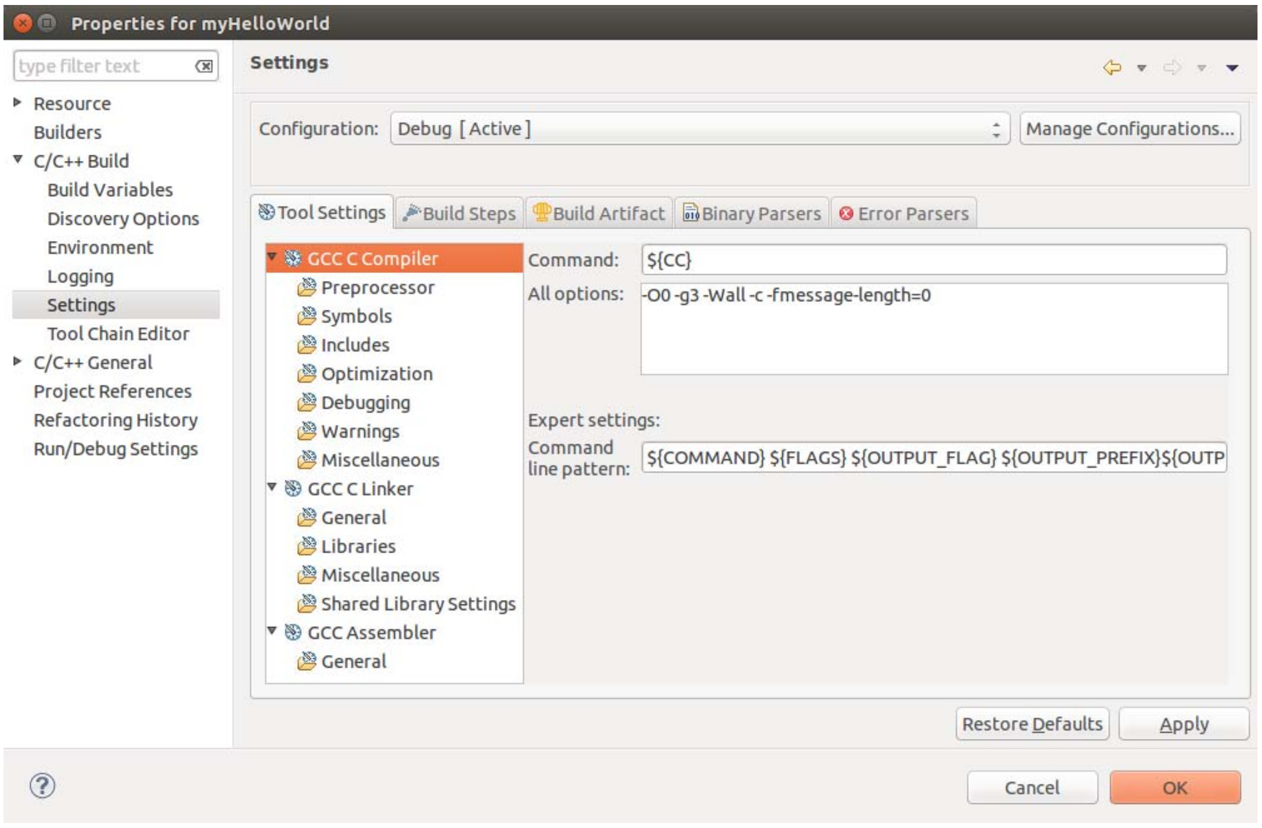
If *Build Automatically* from the *Project* menu is selected, the *HelloWorld* application will now be compiled and created with the standard *GCC C/C++* compiler suitable for your host machine. You will find the executable file, which can only run on your host system, in the *workspace/myHelloWorld/Debug*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

To compile your project for the phyCORE-AM335x 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]

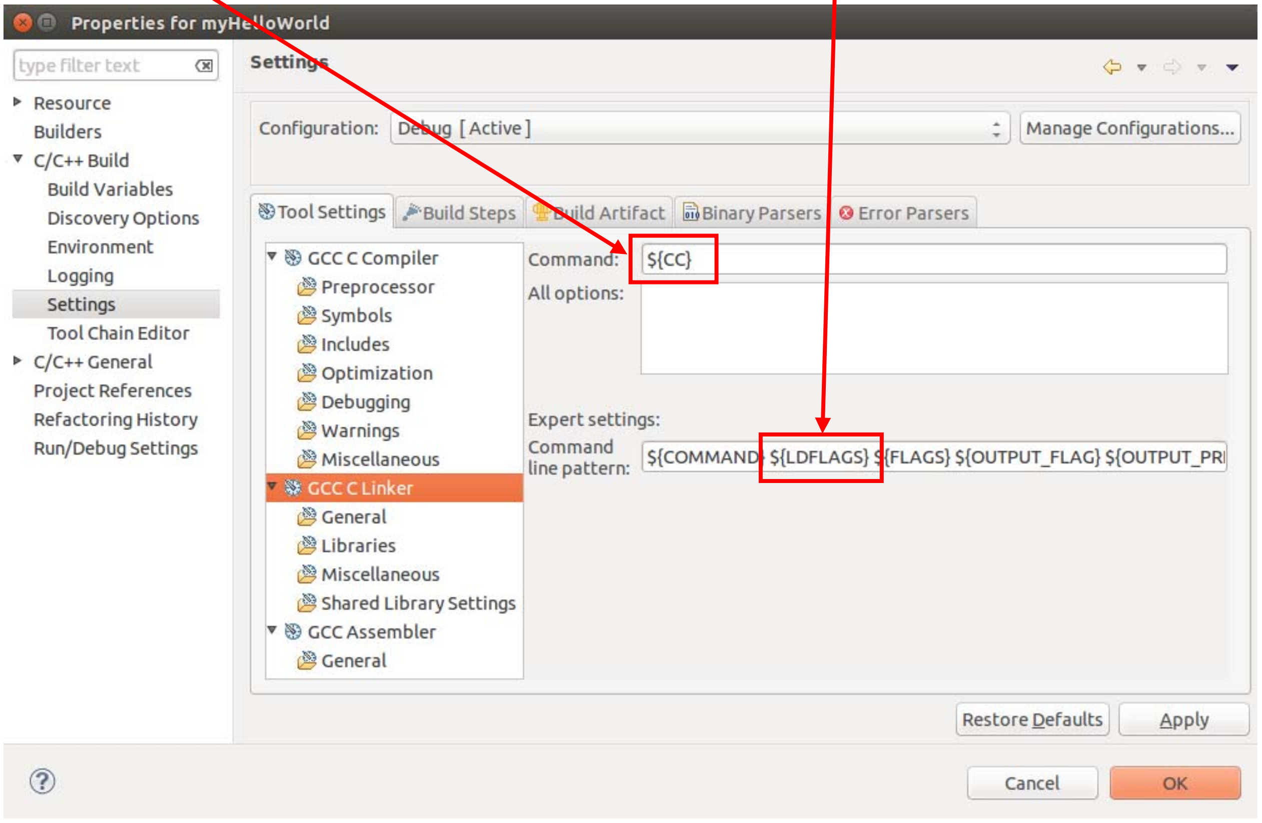
Command: ${CC}

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]

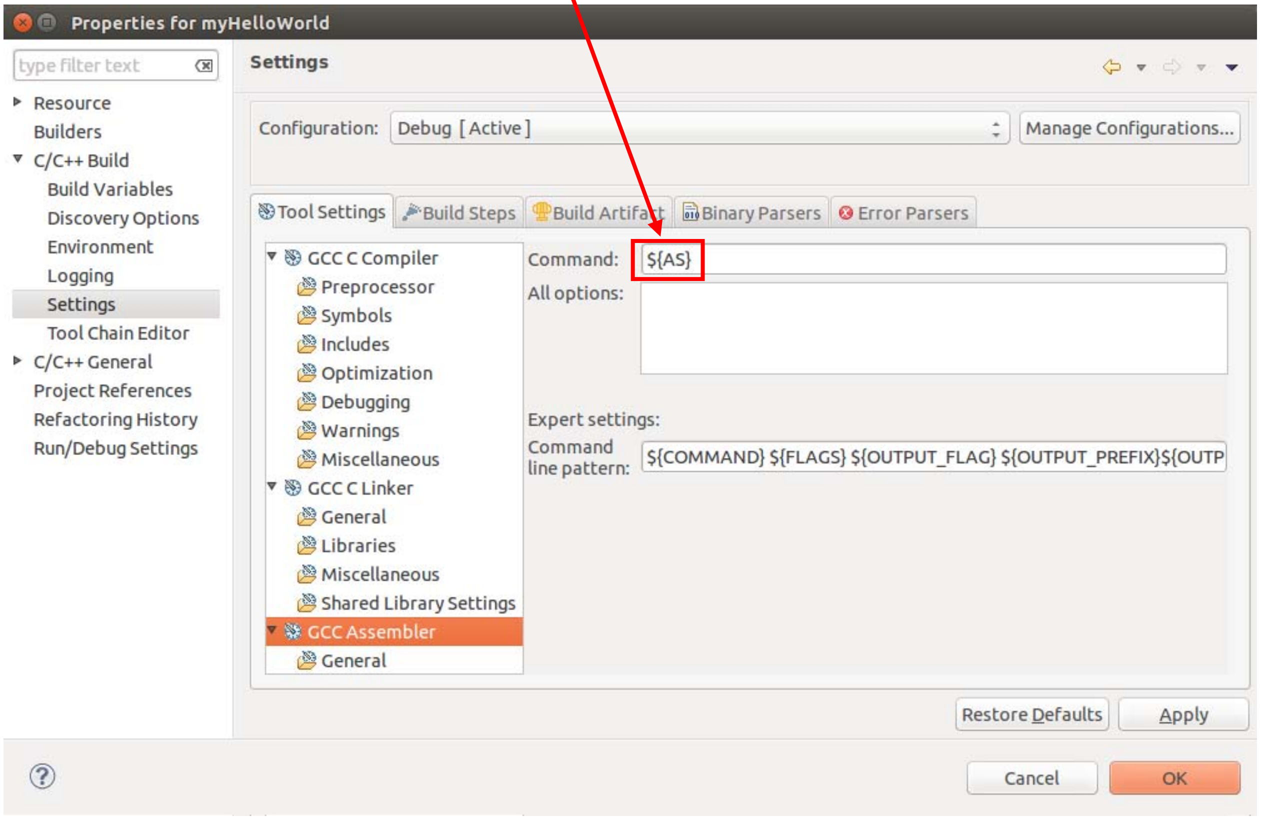
Command: ${CC}

All options:

Expert settings:

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

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


Properties for myHelloWorld

type filter text

- Resource
- Builders
- C/C++ Build
  - Build Variables
  - Discovery Options
  - Environment
  - Logging
  - Settings
  - Tool Chain Editor
- C/C++ General
- Project References
- Refactoring History
- Run/Debug Settings

Settings

Configuration: Debug [Active]

Tool Settings | Build Steps | Build Artifact | Binary Parsers | Error Parsers

- GCC C Compiler
  - Preprocessor
  - Symbols
  - Includes
  - Optimization
  - Debugging
  - Warnings
  - Miscellaneous
- GCC C Linker
  - General
  - Libraries
  - Miscellaneous
  - Shared Library Settings
- GCC Assembler
  - General

Command: ${AS}

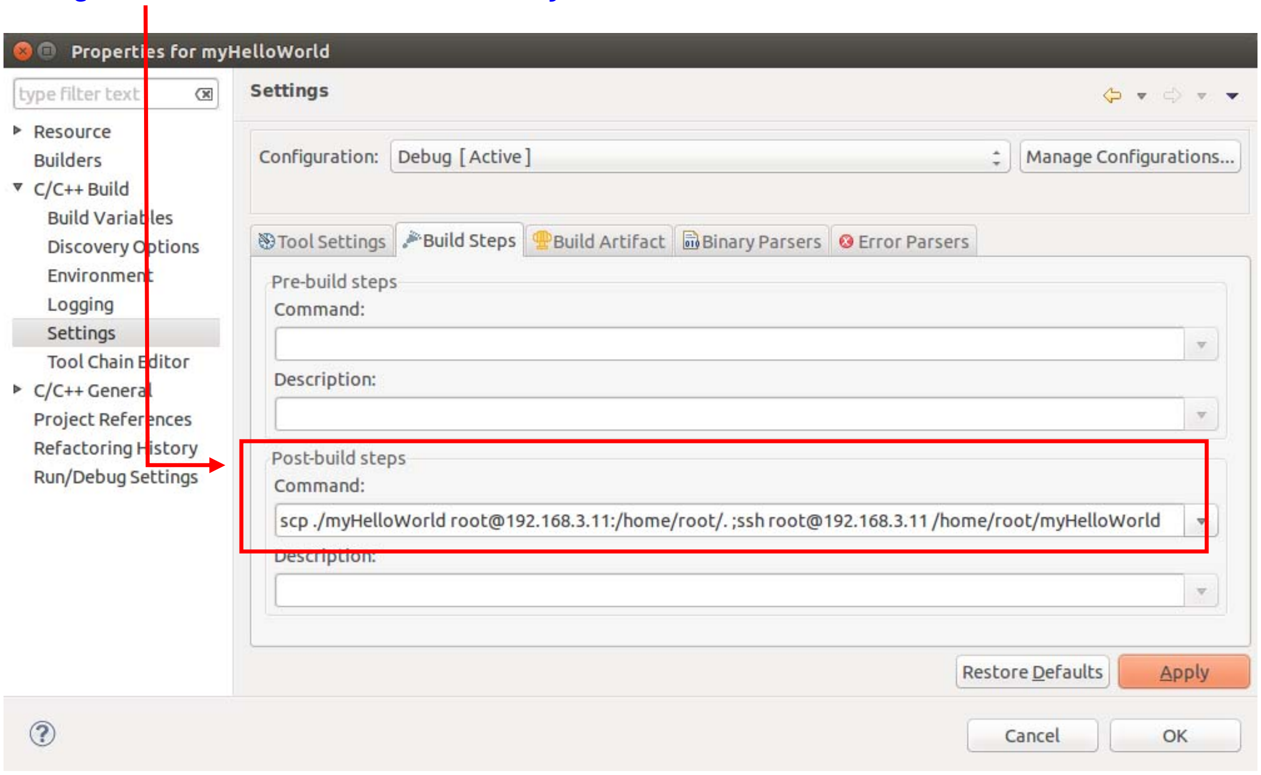
All options:

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`



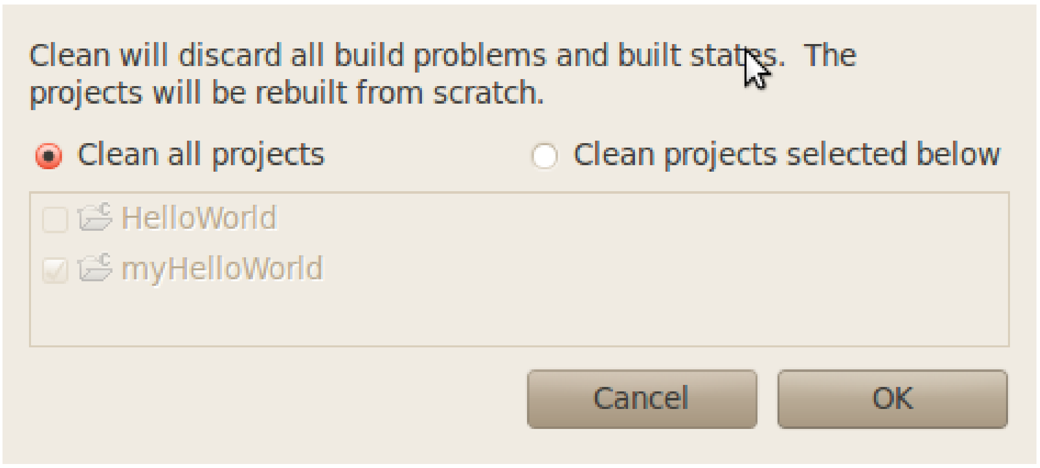
Configuration: Debug [Active]

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

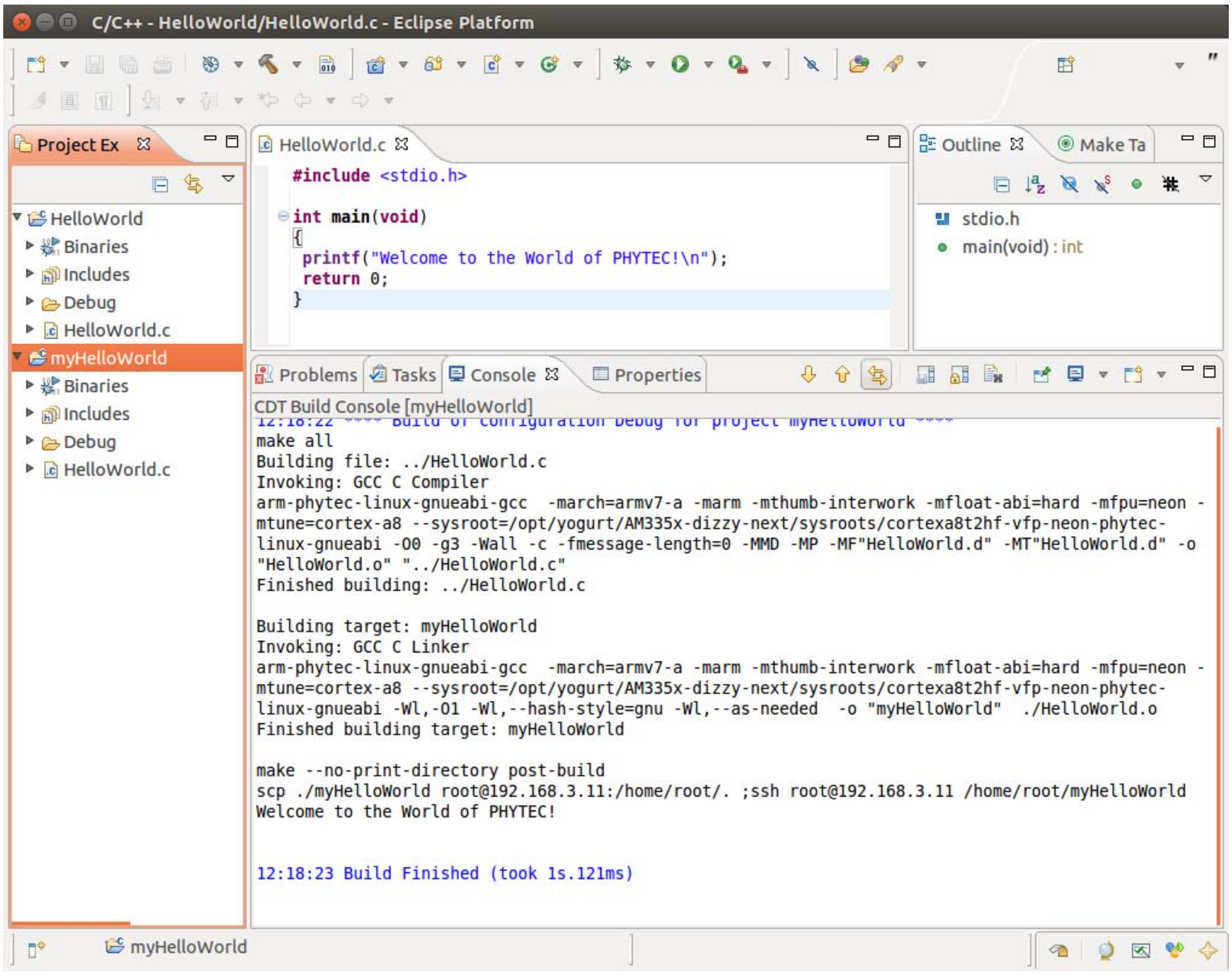
HelloWorld
myHelloWorld

- Confirm with ***OK***.

The project will be rebuilt.

- Select the *Console* tab.

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



```
#include <stdio.h>

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

```
12:18:22 **** Build of configuration Debug for project myHelloWorld ****
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: myHelloWorld
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 "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!


12:18:23 Build Finished (took 1s.121ms)
```

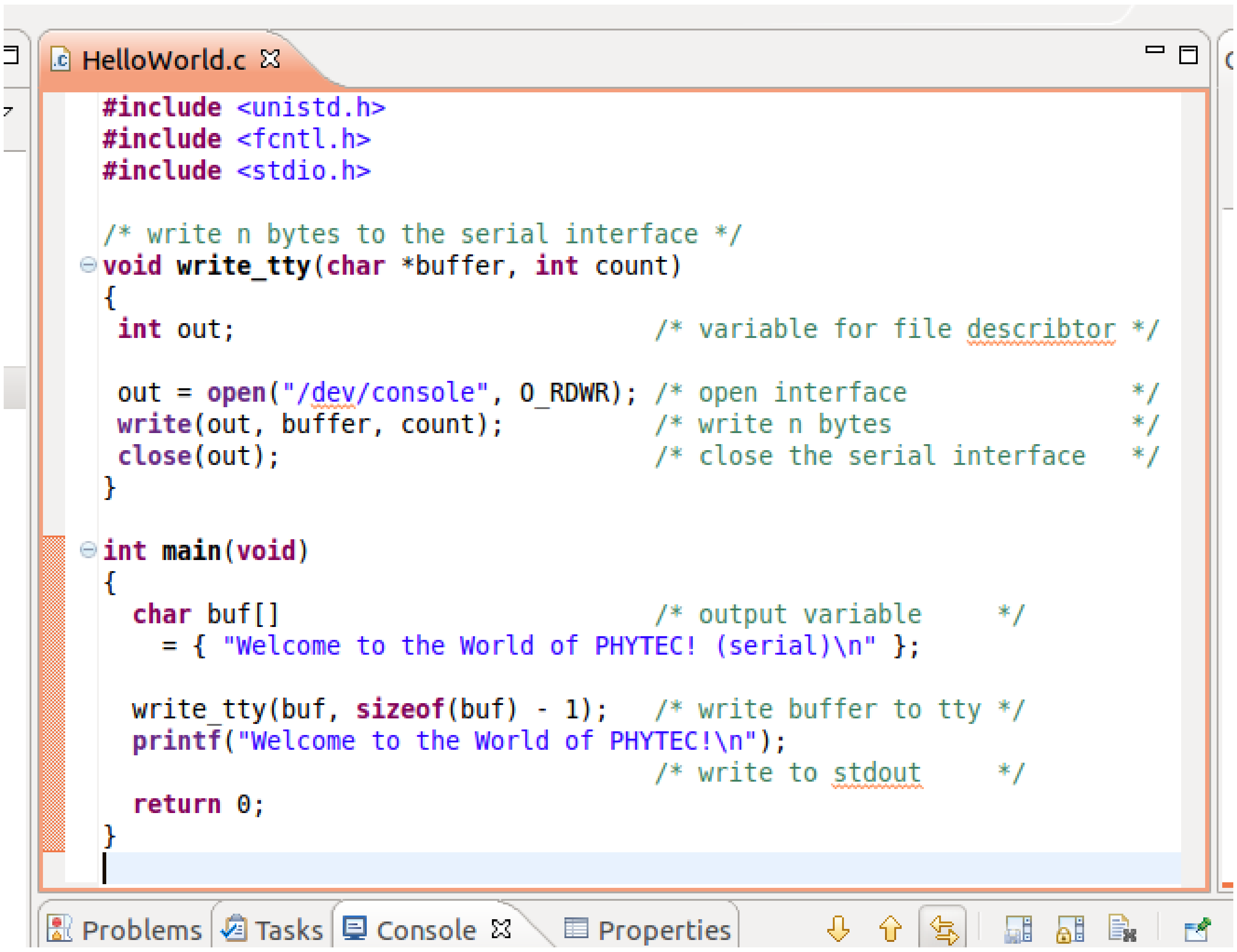
You have successfully created your first own project with the *Eclipse* IDE. You have configured the project to create an application for your target platform.

2.1.1.3 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-Wega AM335x, 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()*.

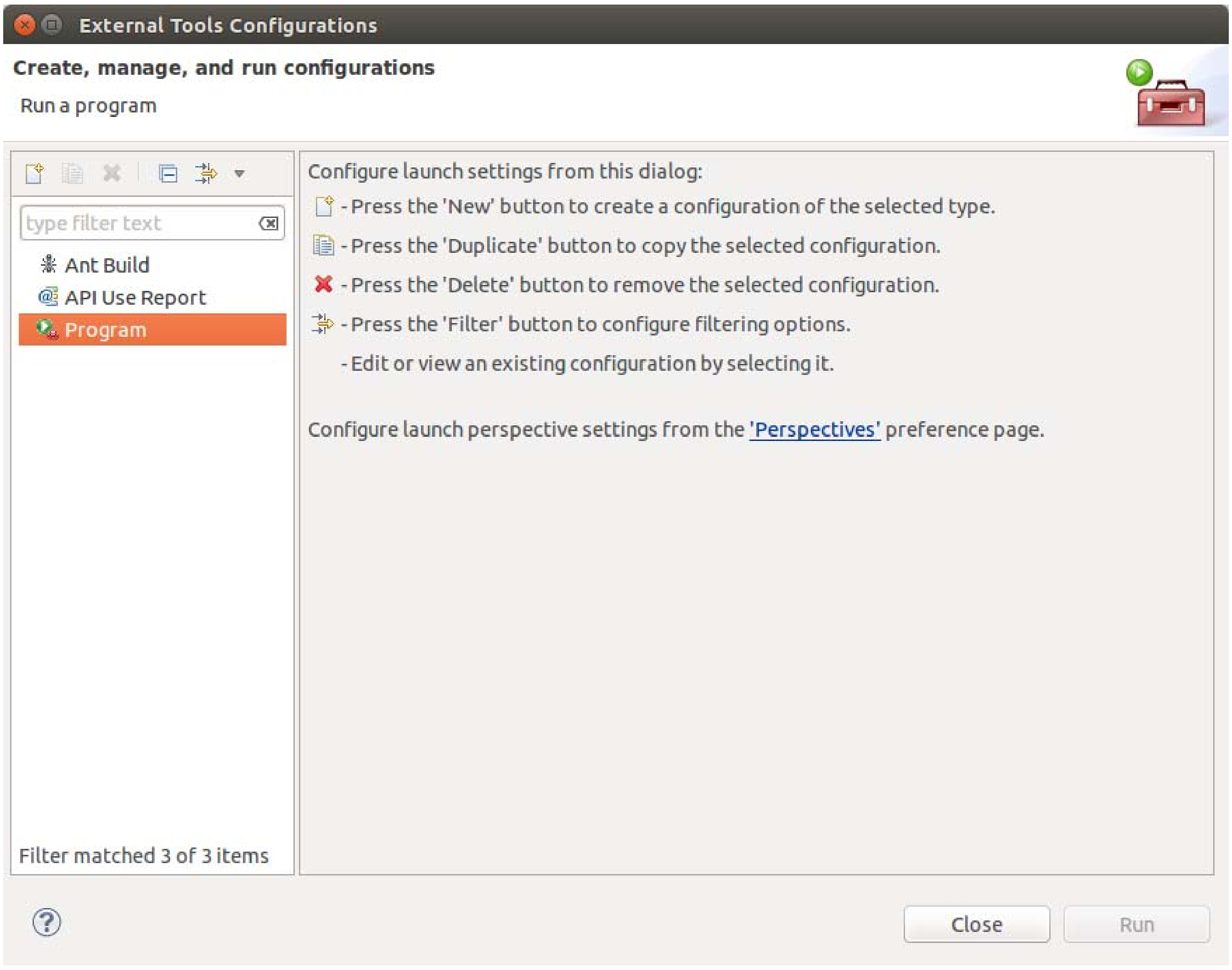
This procedure is quite typical for Linux, because Linux treats everything as a file.

2.1.1.4 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.

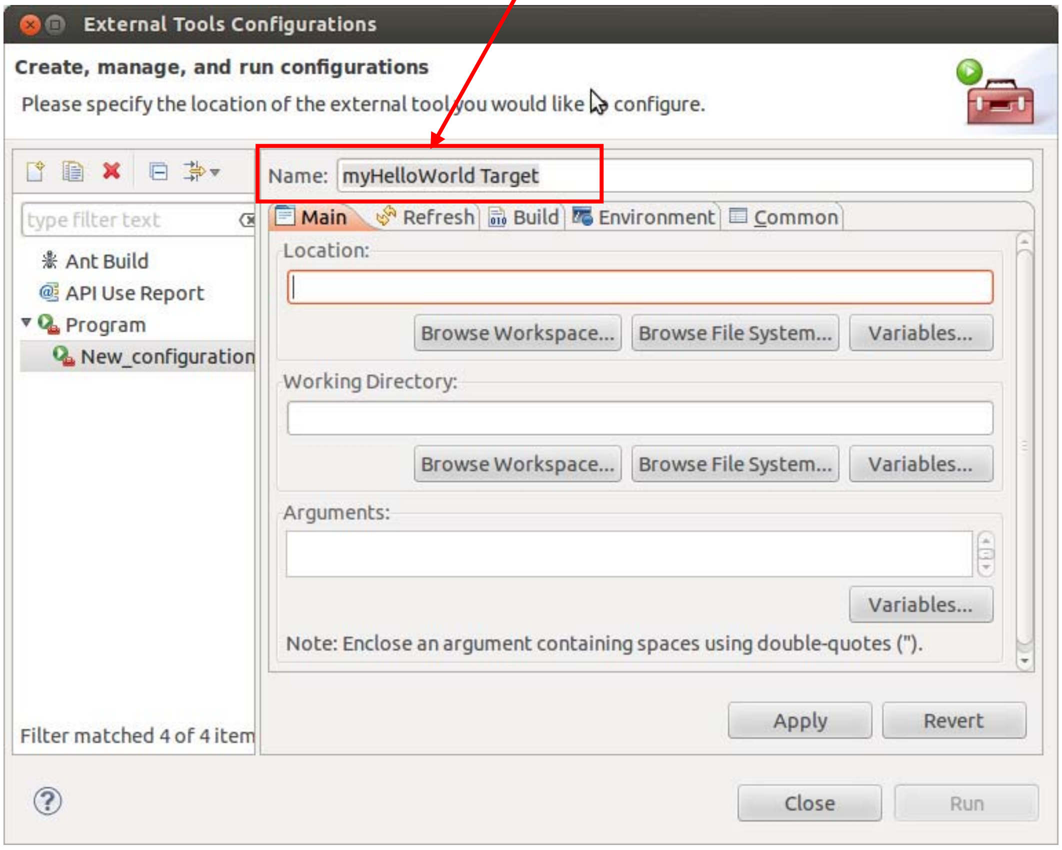

External Tools Configurations
Create, manage, and run configurations
Run a program
type filter text
Ant Build
API Use Report
Program
Filter matched 3 of 3 items
Close
Run

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:

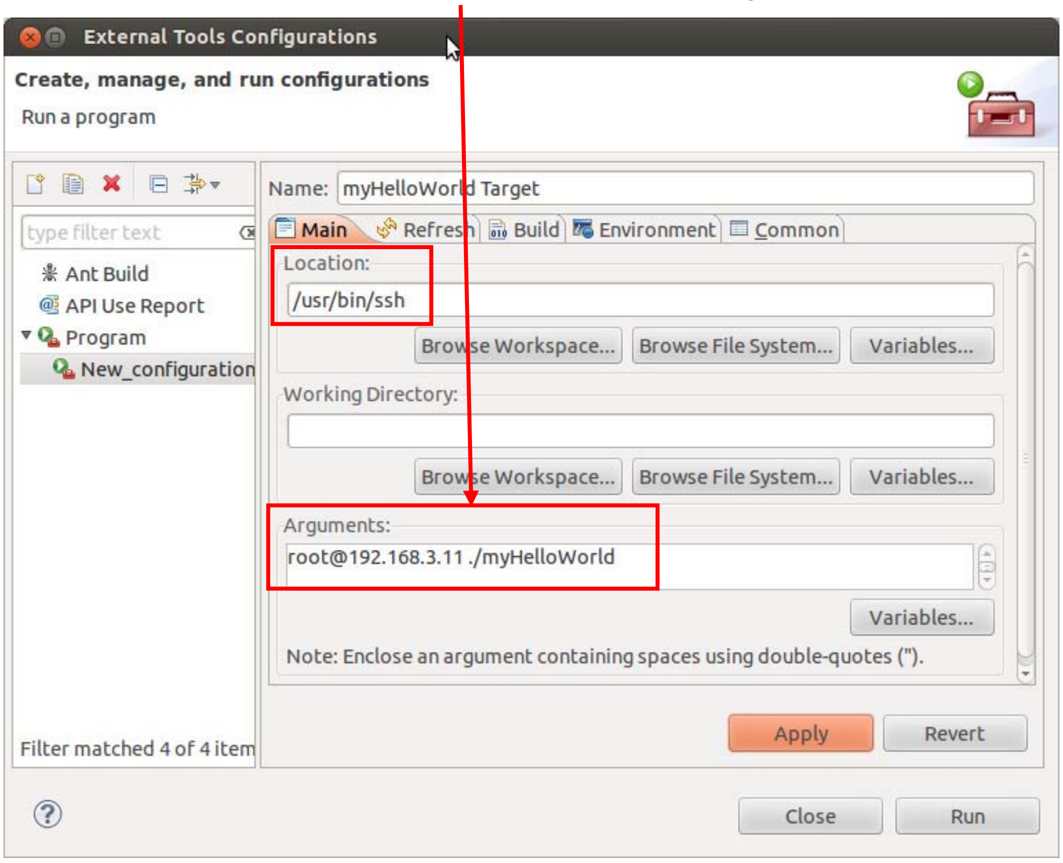
Working Directory:

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.

2.1.2 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.

2.1.2.1 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.

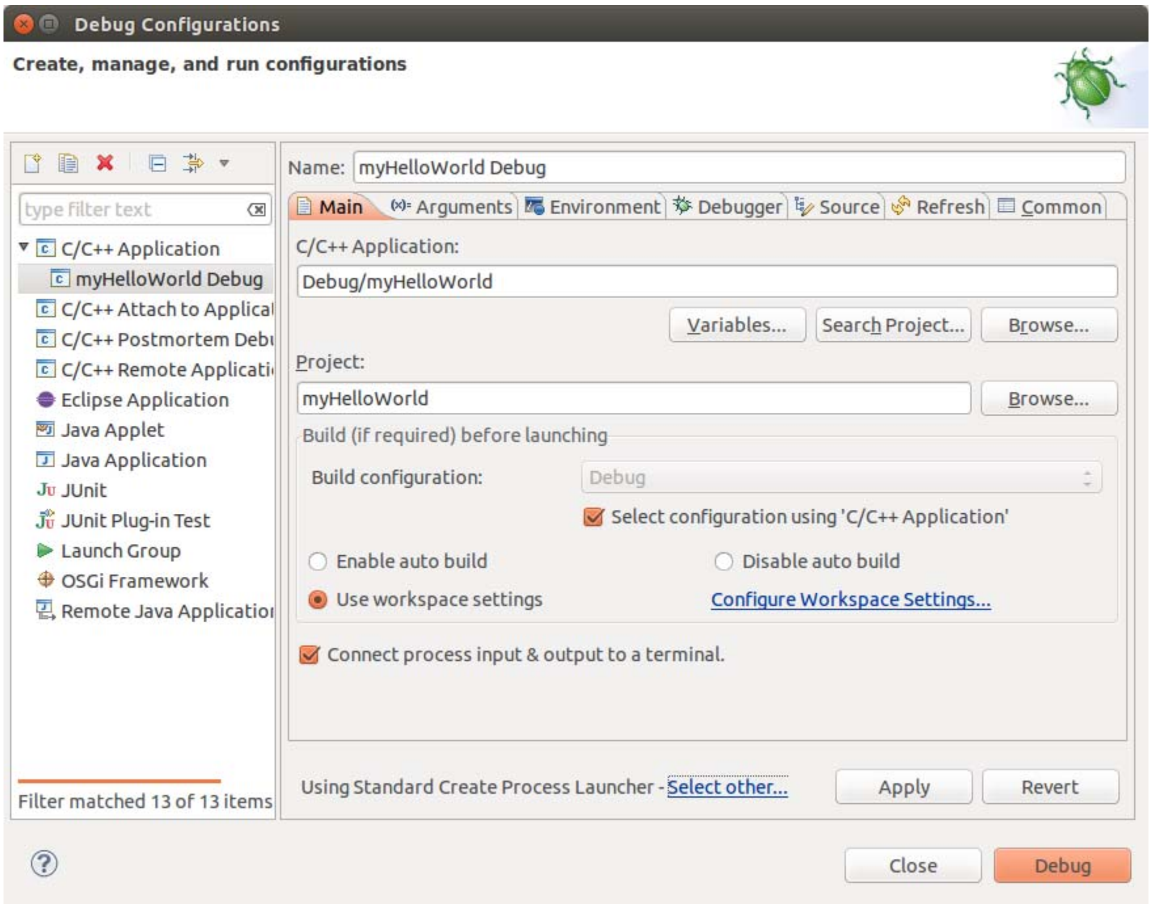
2.1.2.2 Configuring and Starting the Debugger in Eclipse

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).



Name: myHelloWorld Debug

C/C++ Application: Debug/myHelloWorld

Project: myHelloWorld

Build (if required) before launching

Build configuration: Debug

☑ Select configuration using 'C/C++ Application'

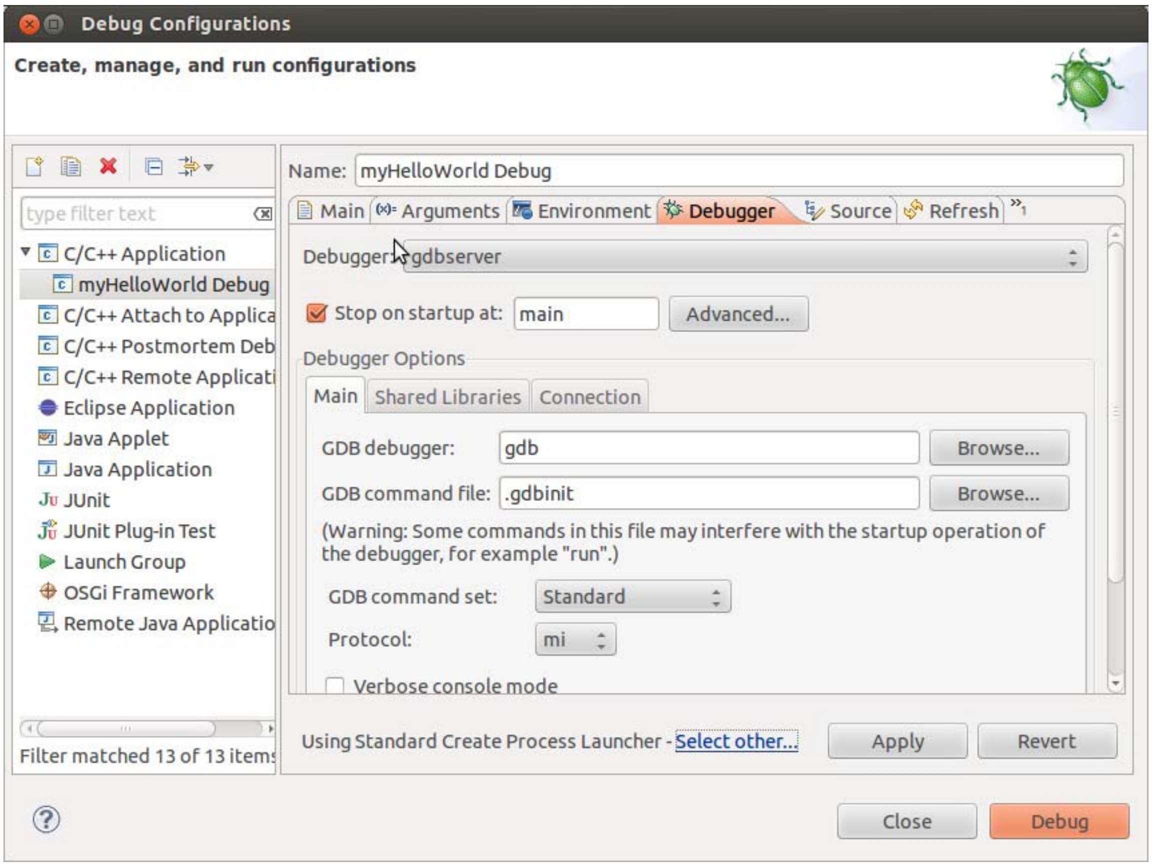
○ Enable auto build ○ Disable auto build

● Use workspace settings

☑ Connect process input & output to a terminal.

Using Standard Create Process Launcher

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



Name: myHelloWorld Debug

Debugger: gdbserver

☑ Stop on startup at: main

Debugger Options

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".)

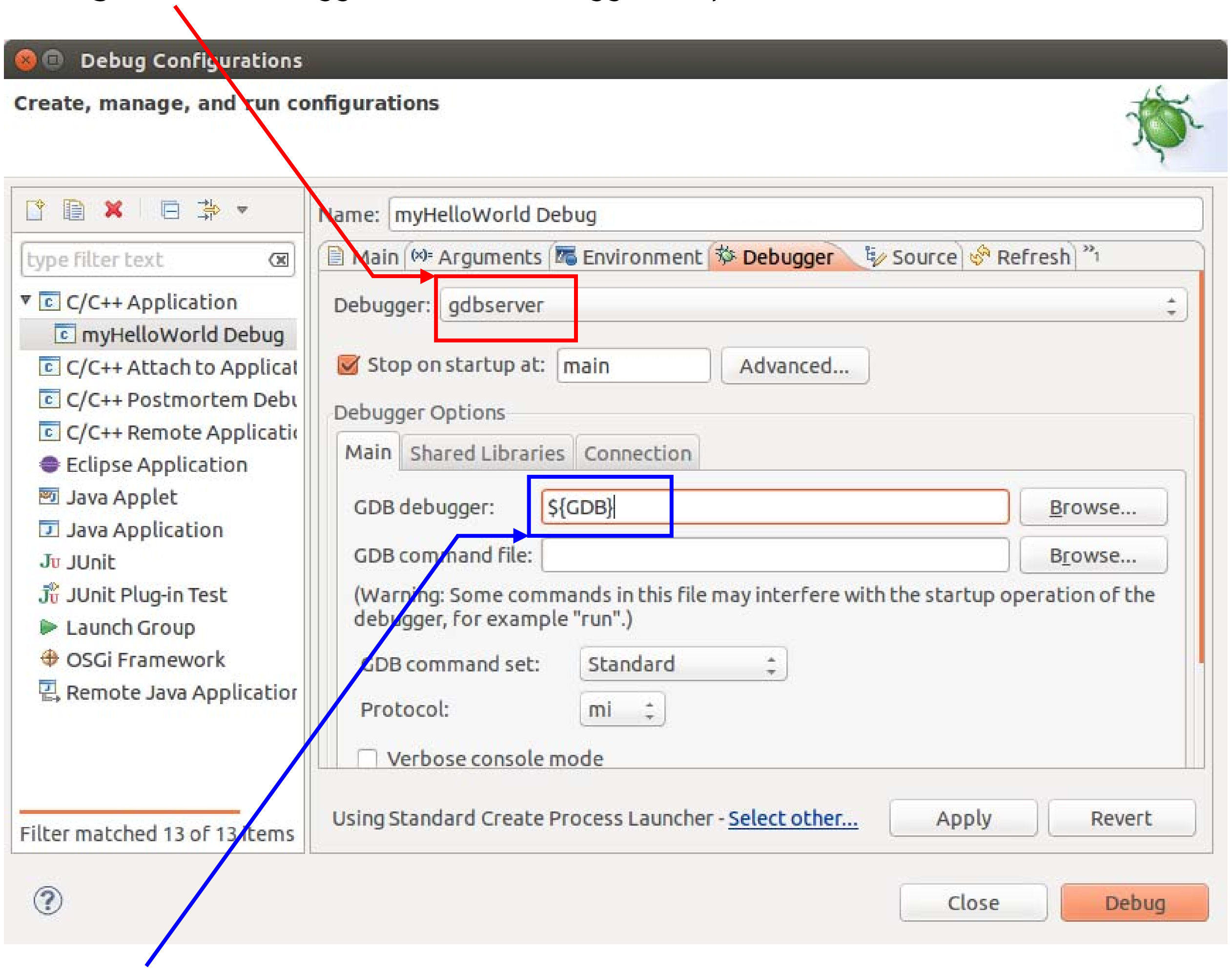
GDB command set: Standard

Protocol: mi

☐ Verbose console mode

Using Standard Create Process Launcher

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



Debug Configurations

Create, manage, and run configurations

Name: myHelloWorld Debug

Debugger: gdbserver

☑ Stop on startup at: main

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

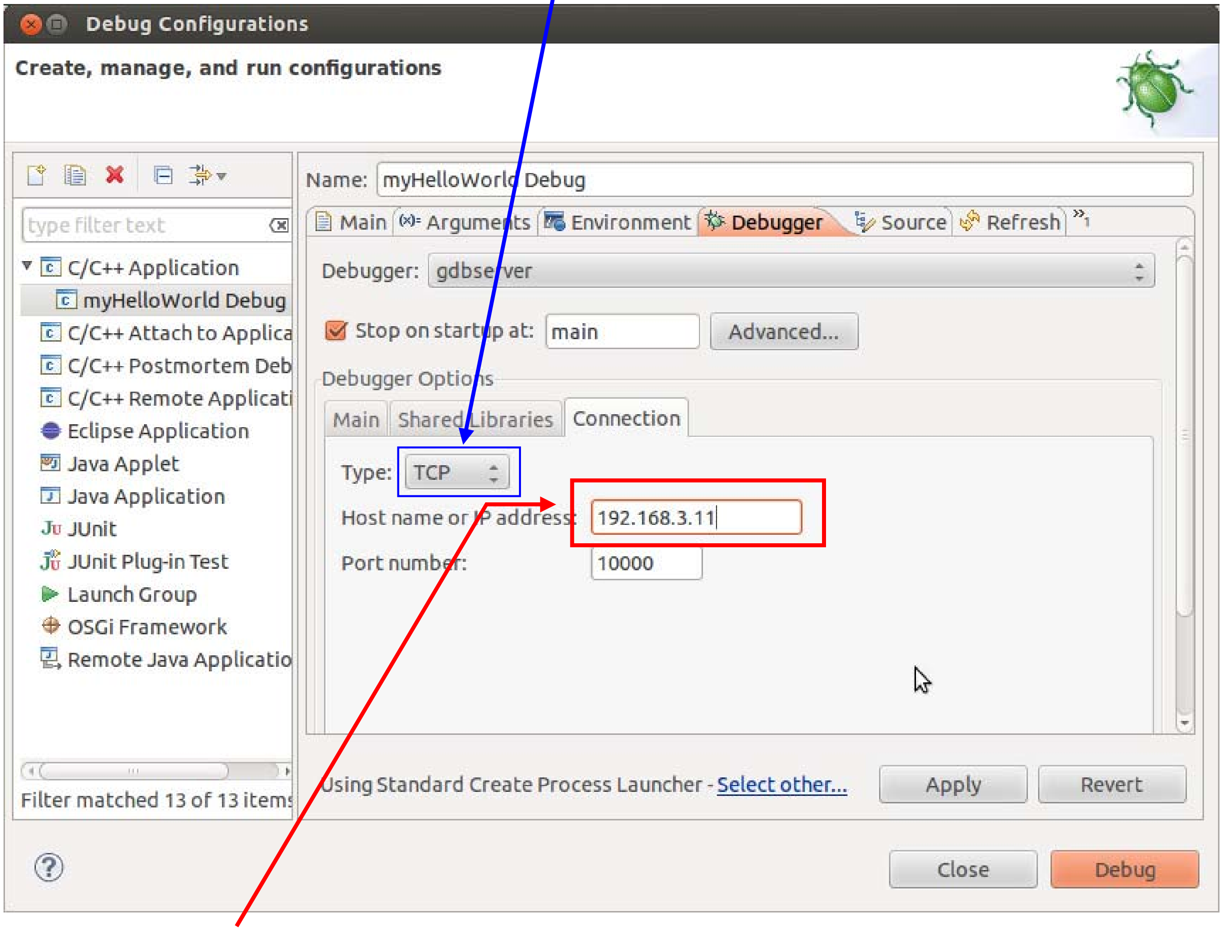
Protocol: mi

☐ Verbose console mode

Using Standard Create Process Launcher - Select other...

- 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

Debugger Options: Connection

Type: TCP

Host name or IP address: 192.168.3.11

Port number: 10000

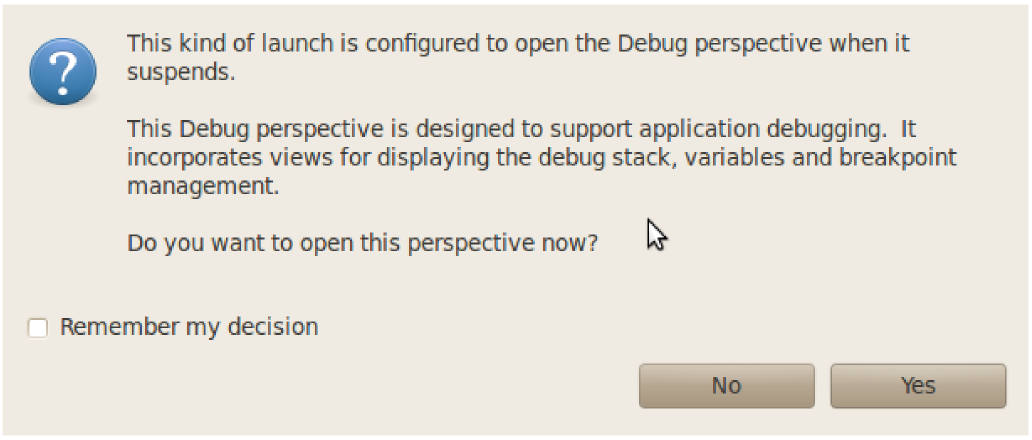
- Enter `192.168.3.11` (the target's IP address) in the *Host name* input field.

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.

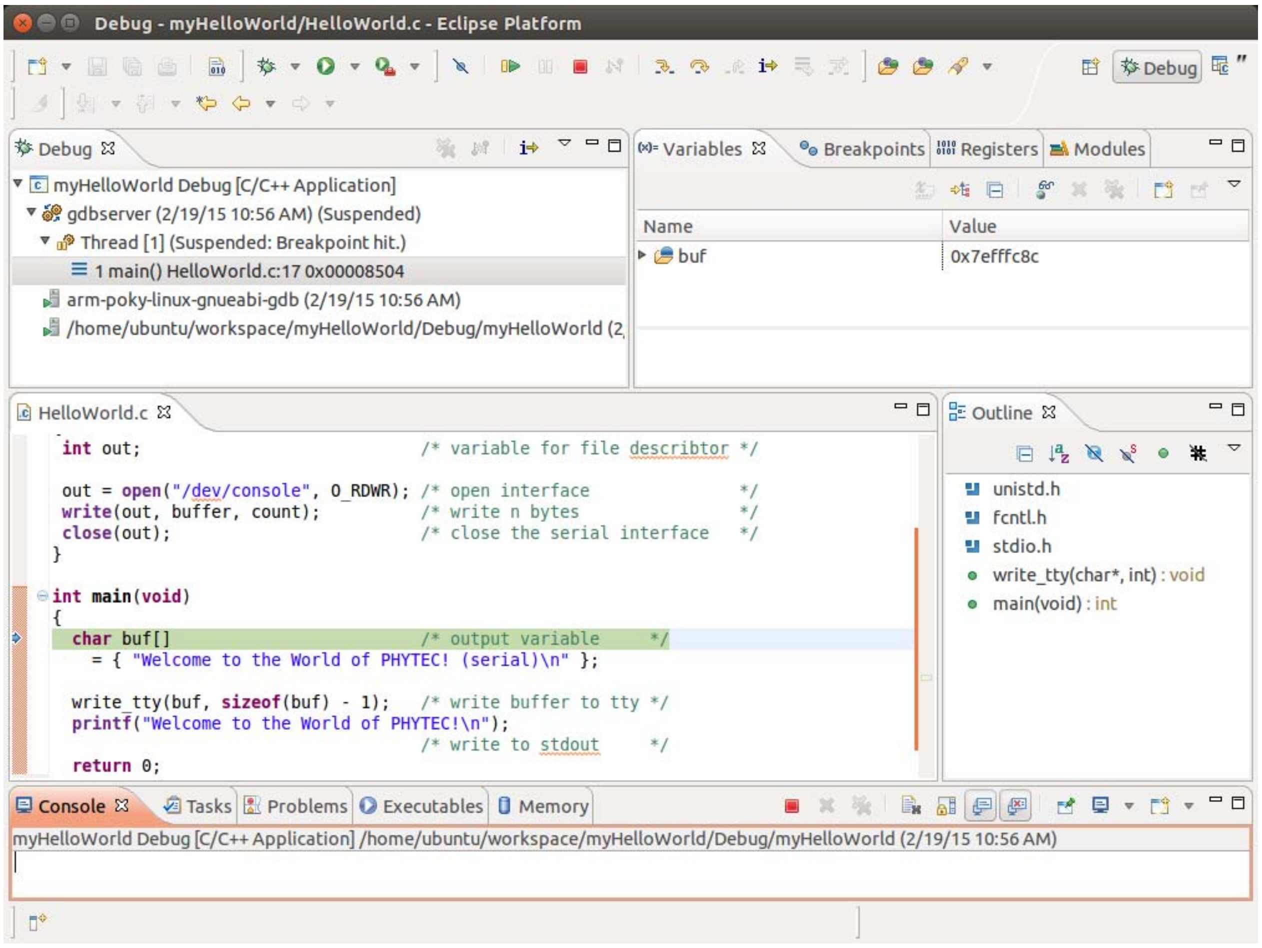


This kind of launch is configured to open the Debug perspective when it suspends.

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?

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.



```
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,
```

Name	Value
buf	0x7efffc8c

```
 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;
```

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

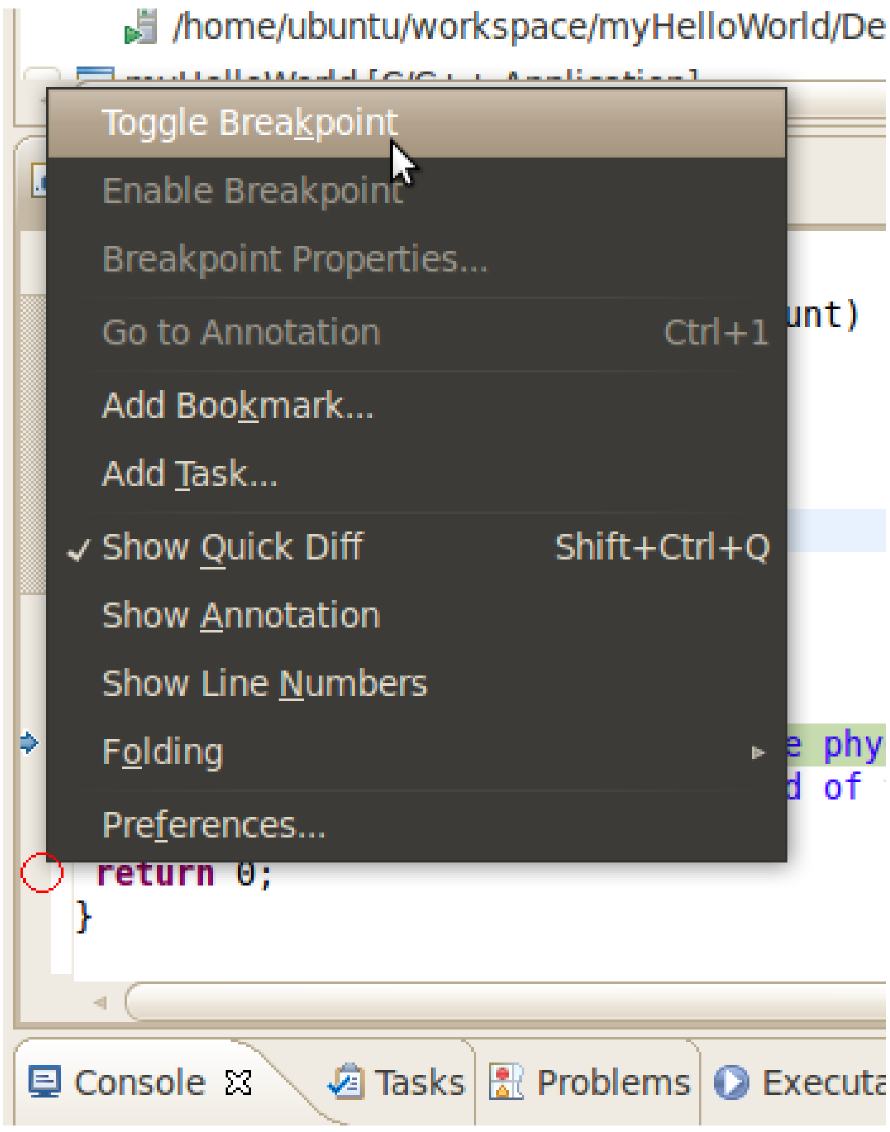
```
myHelloWorld Debug [C/C++ Application] /home/ubuntu/workspace/myHelloWorld/Debug/myHelloWorld (2/19/15 10:56 AM)
```

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.

2.1.2.3 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.

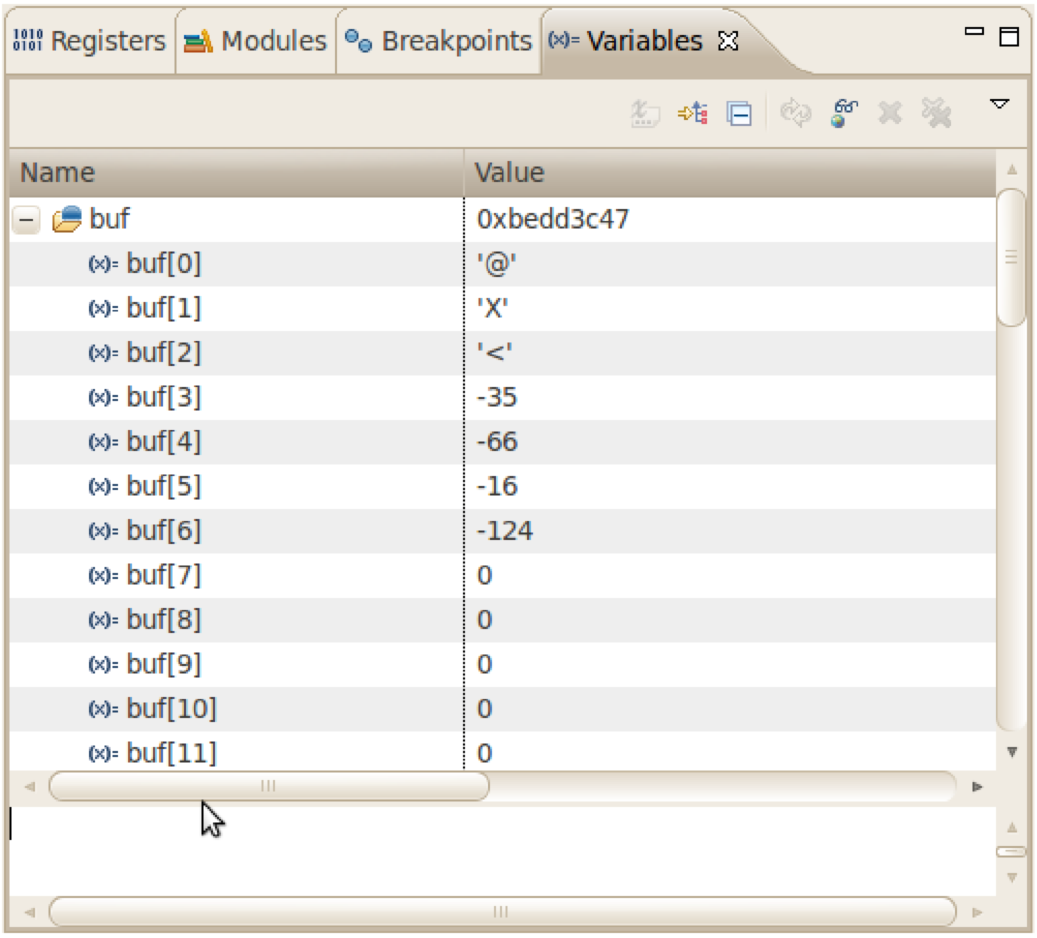


```
return 0;
}
```

2.1.2.4 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.

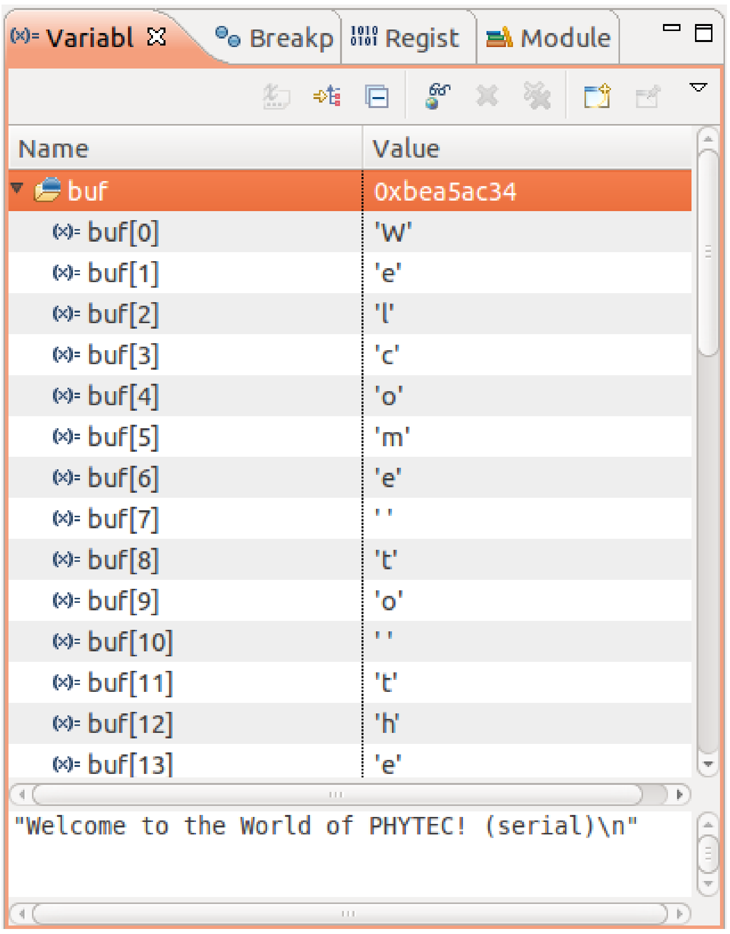
- Expand ***buf*** in the *Variables* window.



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***.



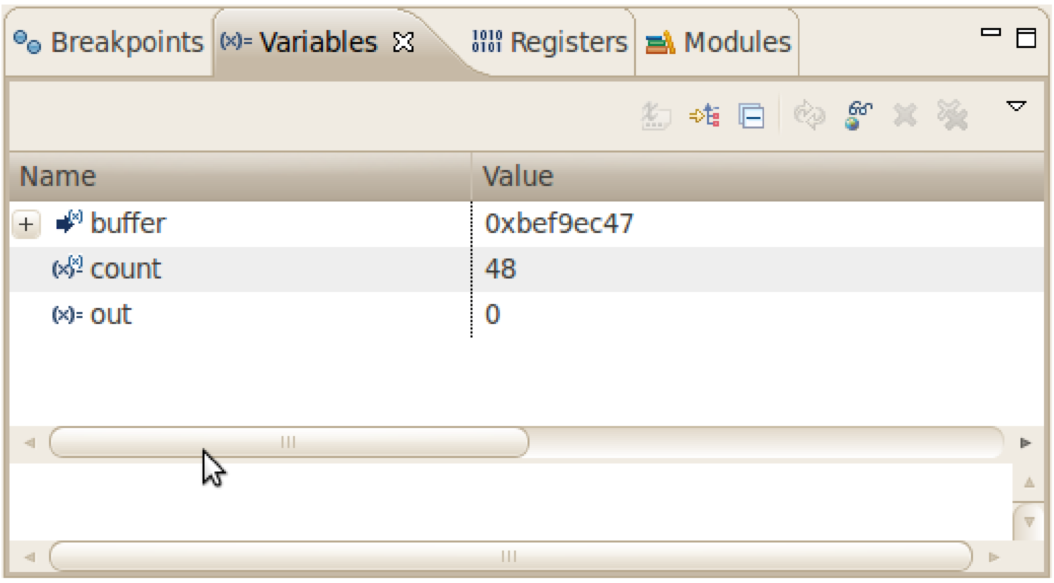
Name	Value
buf	0xbea5ac34
buf[0]	'W'
buf[1]	'e'
buf[2]	'l'
buf[3]	'c'
buf[4]	'o'
buf[5]	'm'
buf[6]	'e'
buf[7]	' '
buf[8]	't'
buf[9]	'o'
buf[10]	' '
buf[11]	't'
buf[12]	'h'
buf[13]	'e'

```
"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

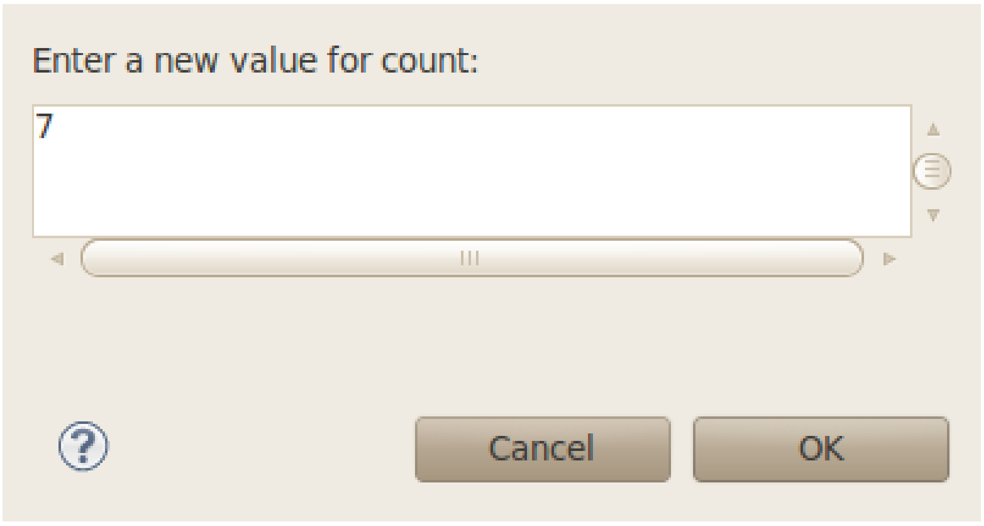
- Click on the variable ***buffer***.

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.

2.1.2.5 Stepping through and Changing Variable Contents

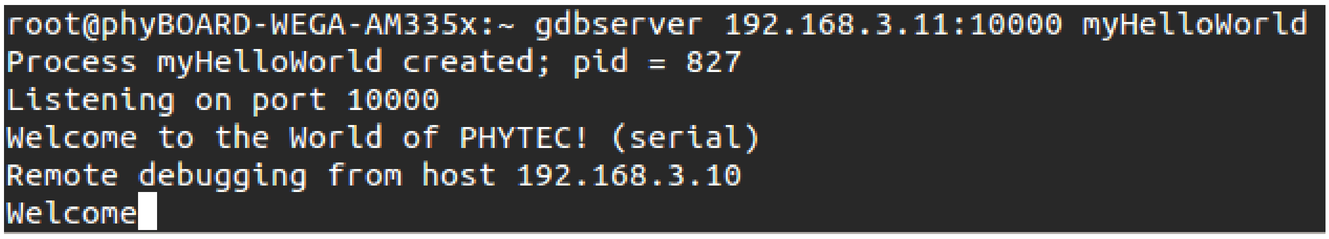
In this section you will change the value of a variable. At the end of this part you will see the effect of this change.

- 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***.



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

- Switch to *Microcom*.



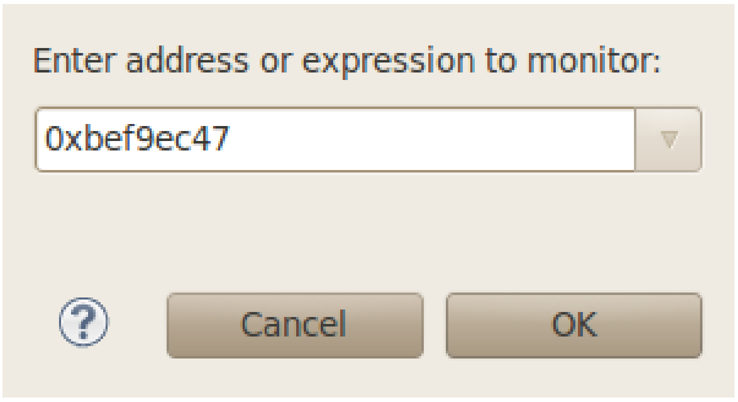
```
root@phyBOARD-WEGA-AM335x:~ gdbserver 192.168.3.11:10000 myHelloWorld
Process myHelloWorld created; pid = 827
Listening on port 10000
Welcome to the World of PHYTEC! (serial)
Remote debugging from host 192.168.3.10
Welcome
```

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

2.1.2.6 Using the Memory Monitor

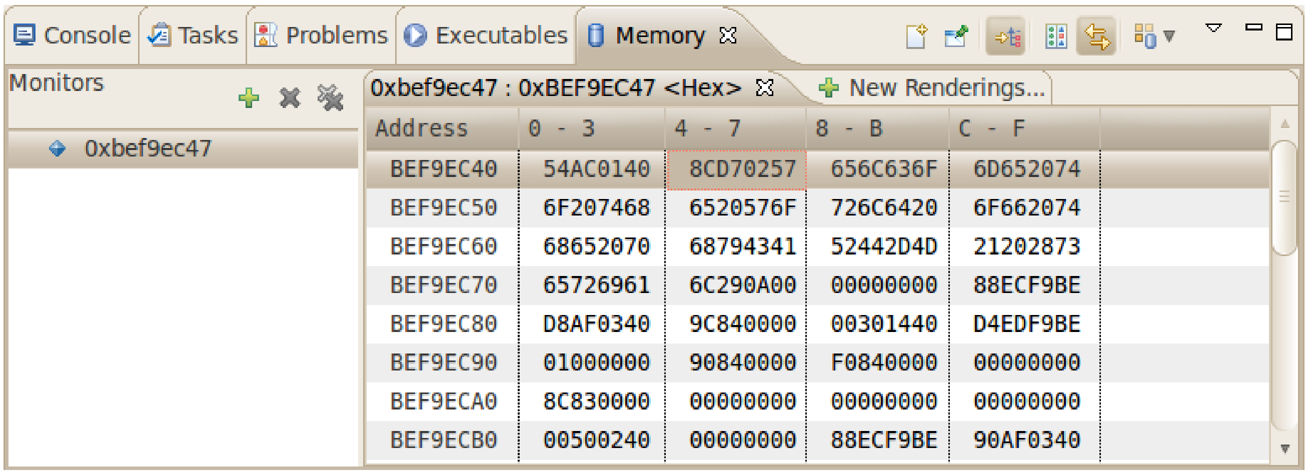
In the last section of this chapter you will use the memory monitor to control the content at a memory address.

- 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.


Enter address or expression to monitor:

0xbef9ec47

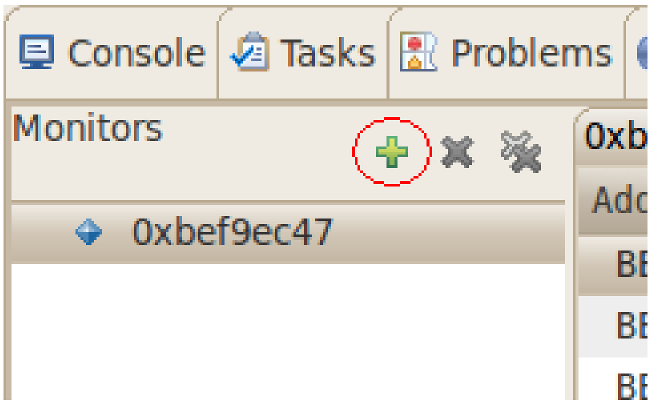
- Change the size of the window.



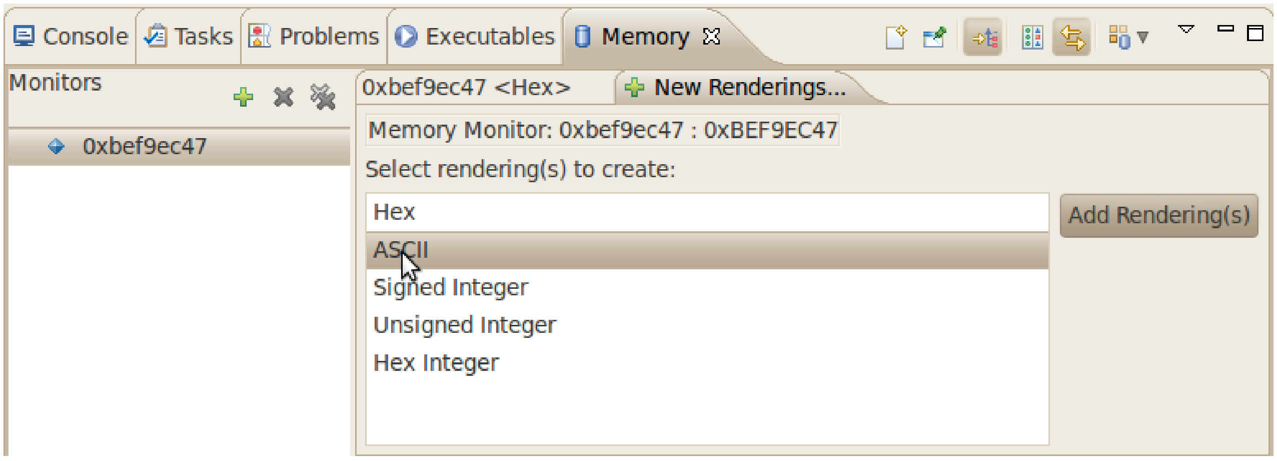
0xbef9ec47 : 0xBEF9EC47 <Hex>

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)***.

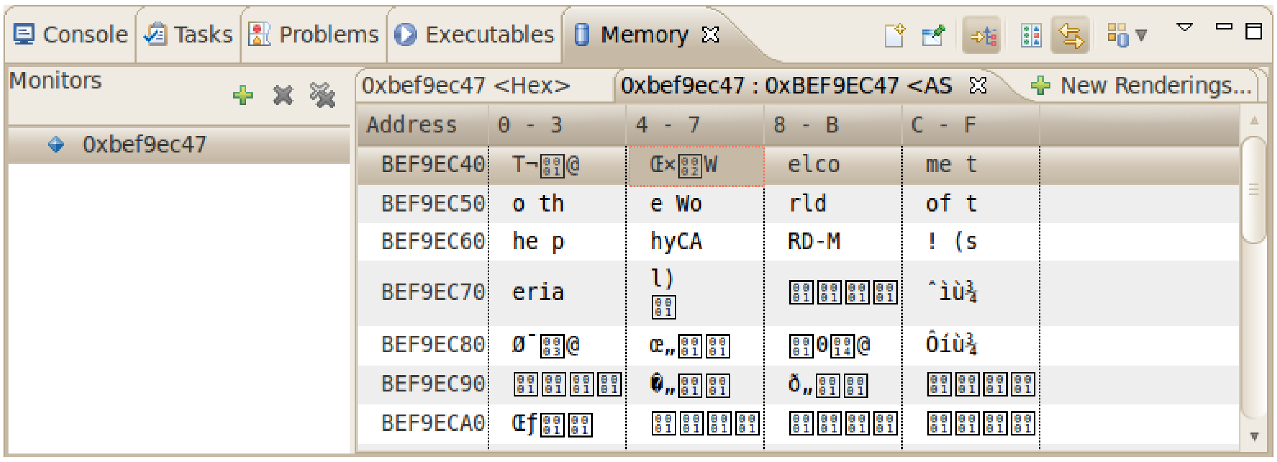


Memory Monitor: 0xbef9ec47 : 0xBEF9EC47

Select rendering(s) to create:

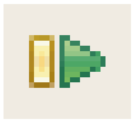
- Hex
- ASCII
- Signed Integer
- Unsigned Integer
- Hex Integer

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

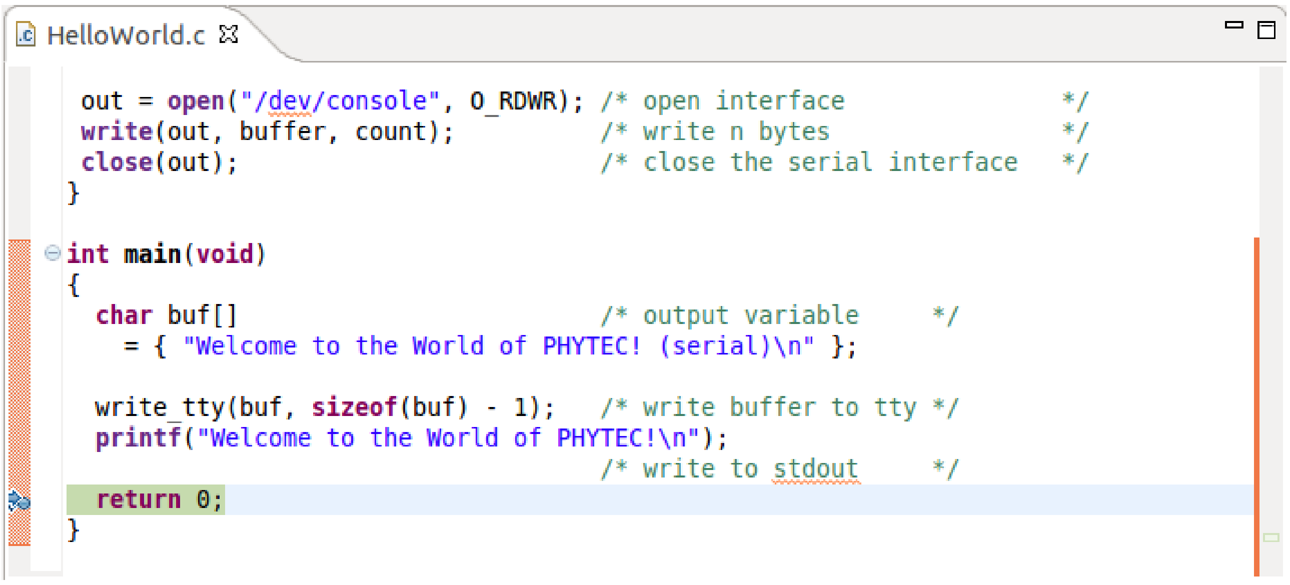


```
Address    0 - 3     4 - 7     8 - B     C - F
BEF9EC40   T¬▯@      Œx▯W      elco      me t
BEF9EC50   o th      e Wo      rld       of t
BEF9EC60   he p      hyCA      RD-M      ! (s
BEF9EC70   eria      l)▯       ▯▯▯▯      ˆìù¾
BEF9EC80   ø¯▯@      œ„▯▯      ▯0▯@      Ôíù¾
BEF9EC90   ▯▯▯▯    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

2.2 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.

2.2.1 Stop the Running Qt Demo on the Target

A *Qt* demo application starts automatically by default after the target has booted completely (*section 3.2.14*). 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`

2.2.2 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.

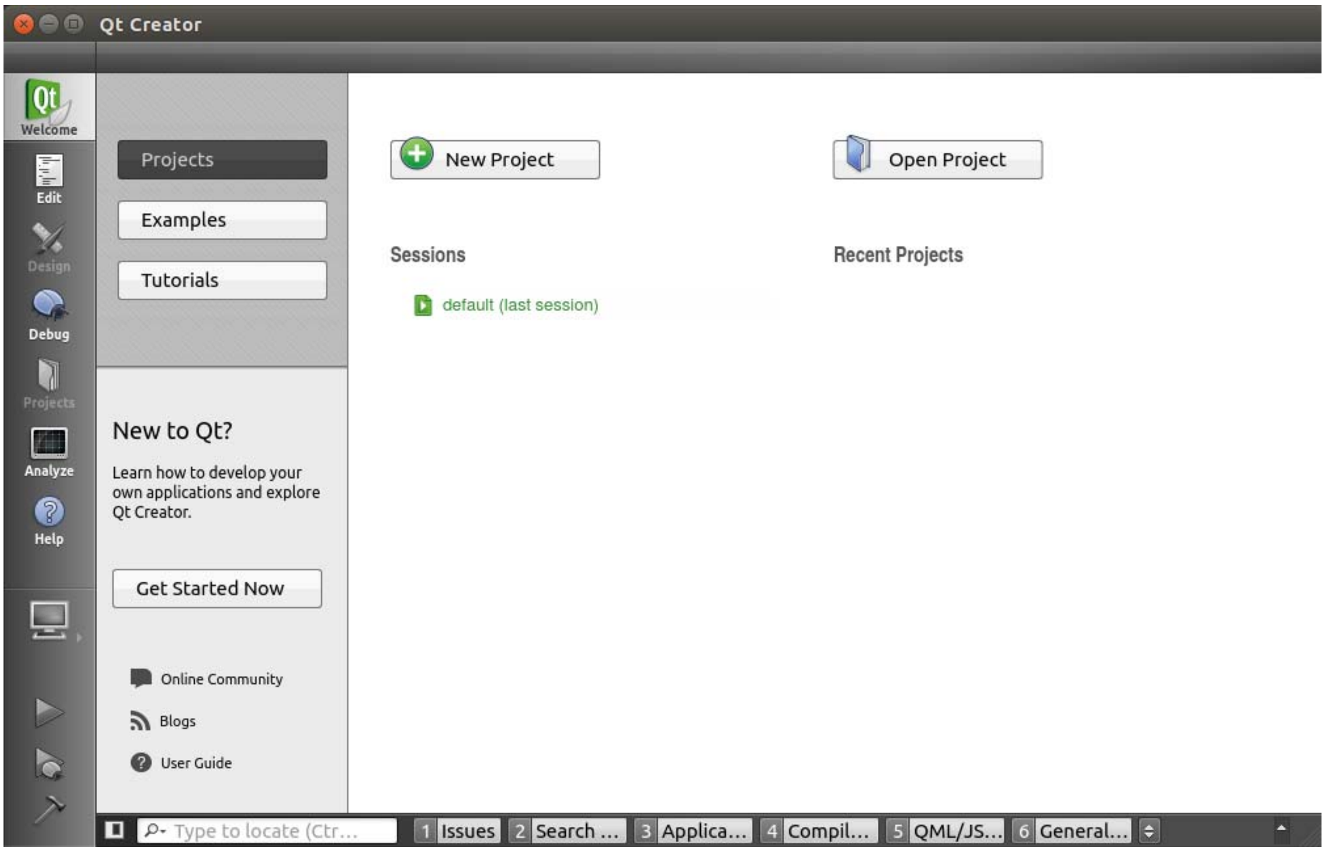
- Open a terminal.

Terminal

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

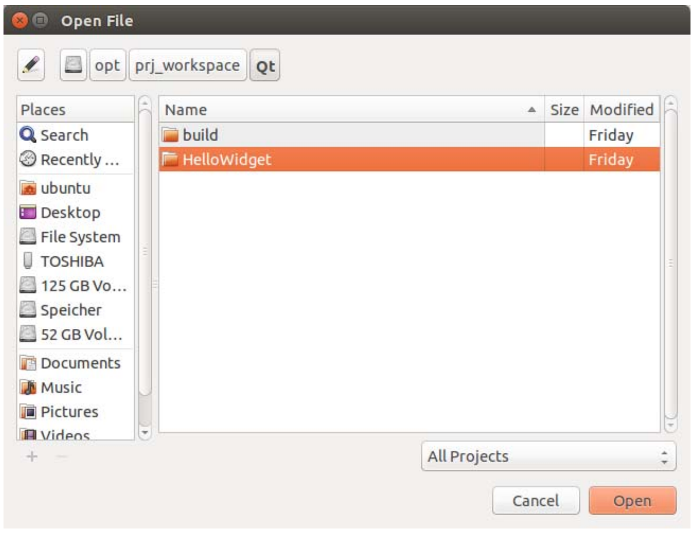
```
/usr/bin/qtcreator.sh &
```

The user interface of *Qt Creator* appears:



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

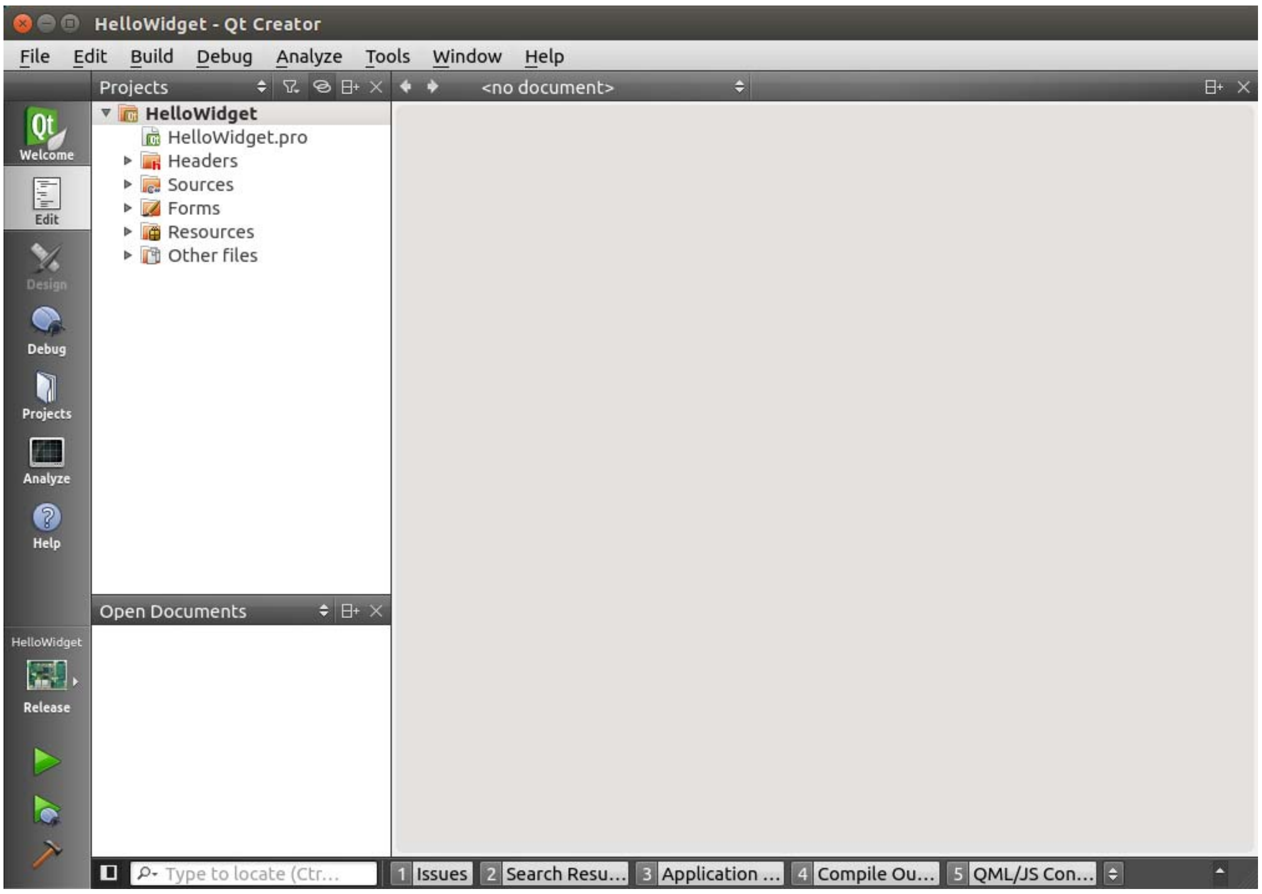
This opens a dialog in which the path to */opt/prj_workspace/Qt* is set automatically.



Name	Size	Modified
build		Friday
HelloWidget		Friday

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

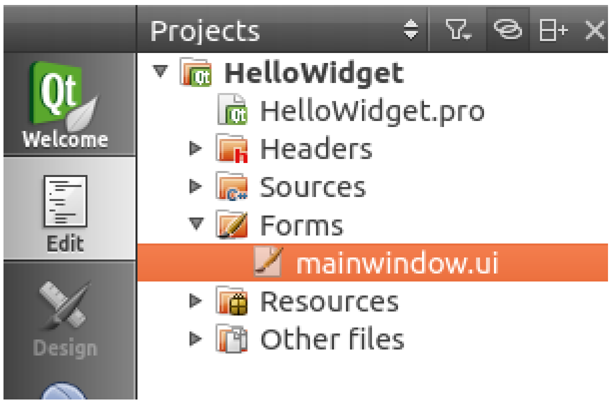
The *HelloWidget* project is now imported into your *Qt Creator* workspace.



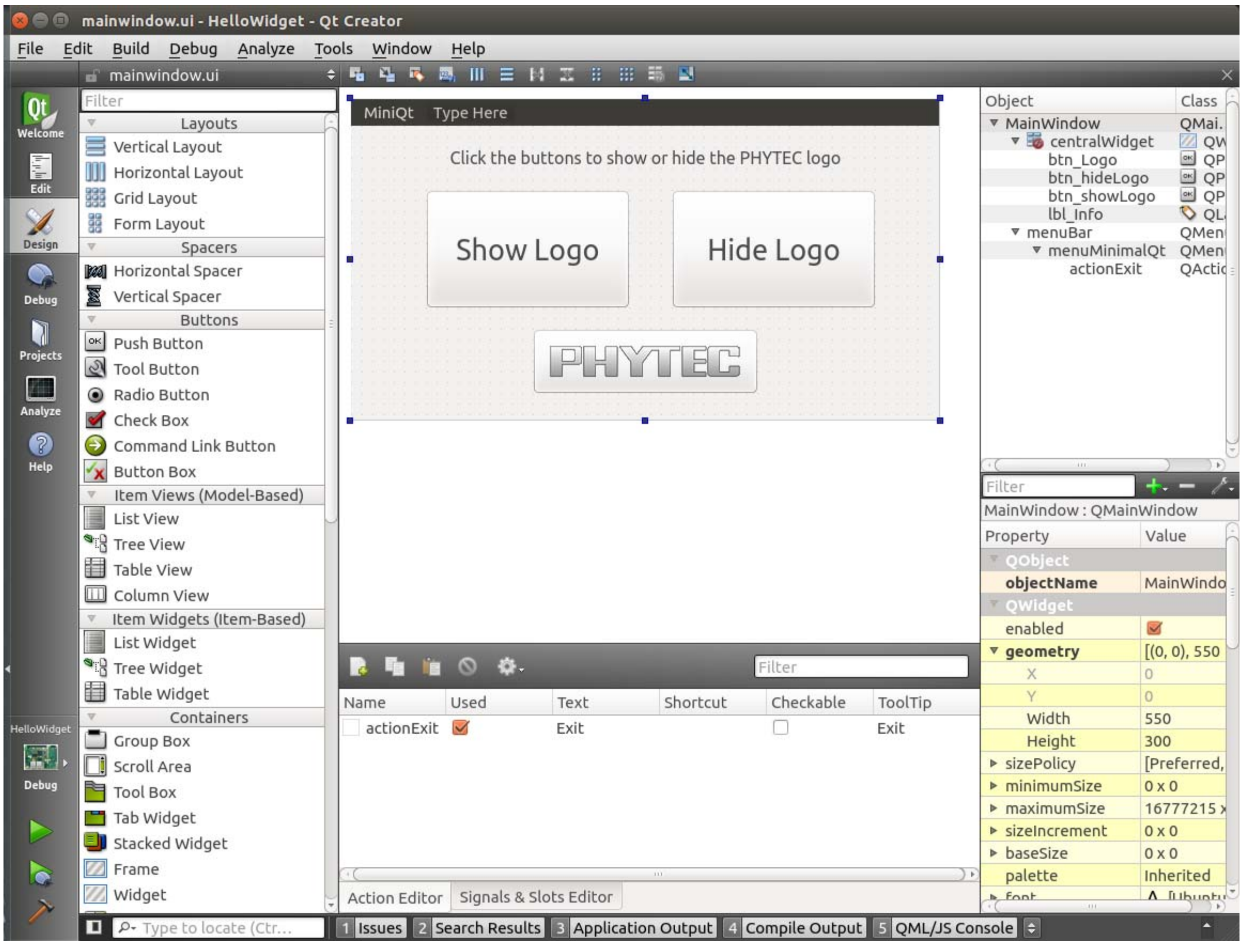
2.2.3 Work with the Demo Application

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

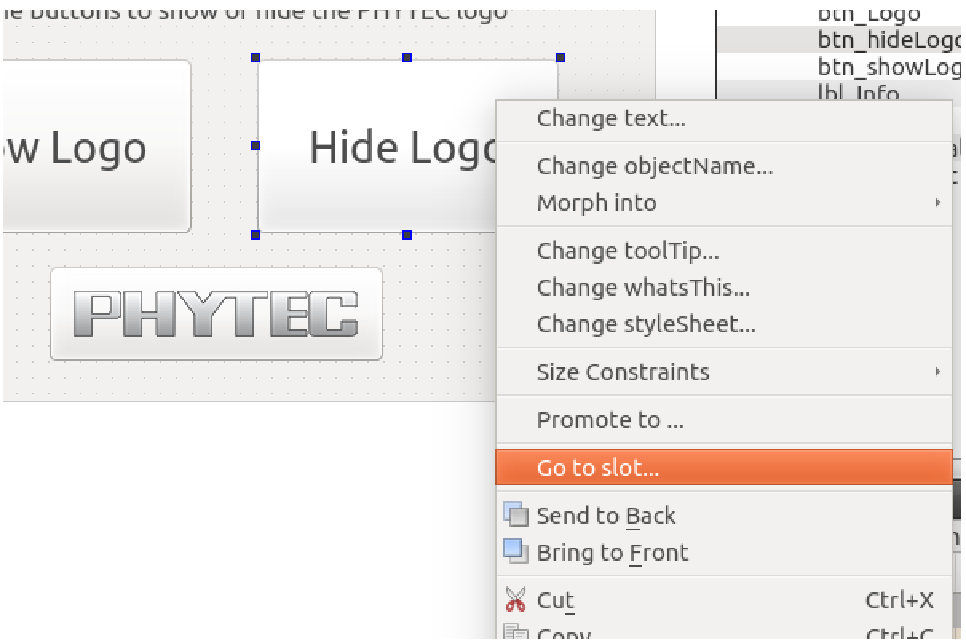
- Expand the folder ***Forms*** under the project sidebar on the left side and double-click ***mainwindow.ui***.



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.

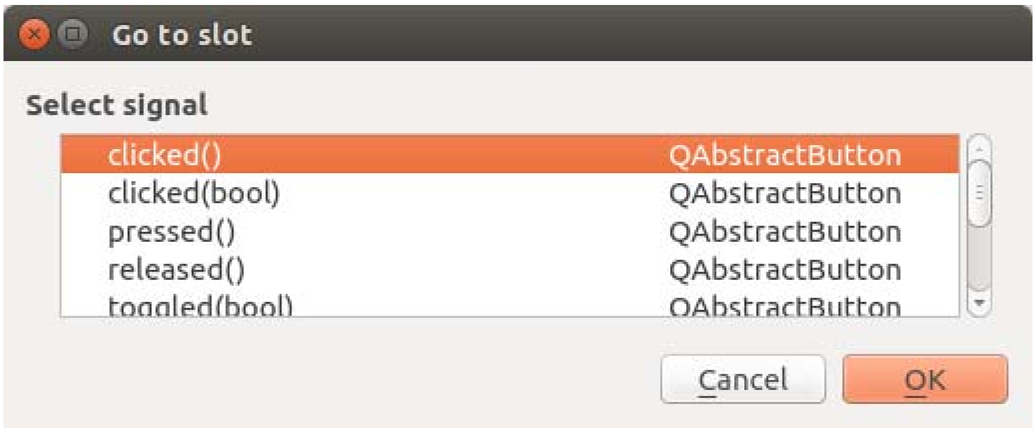

mainwindow.ui - HelloWidget - Qt Creator
File Edit Build Debug Analyze Tools Window Help
MiniQt Type Here
Click the buttons to show or hide the PHYTEC logo
Show Logo
Hide Logo
PHYTEC
Action Editor
Signals & Slots Editor
MainWindow : QMainWindow

- 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.***

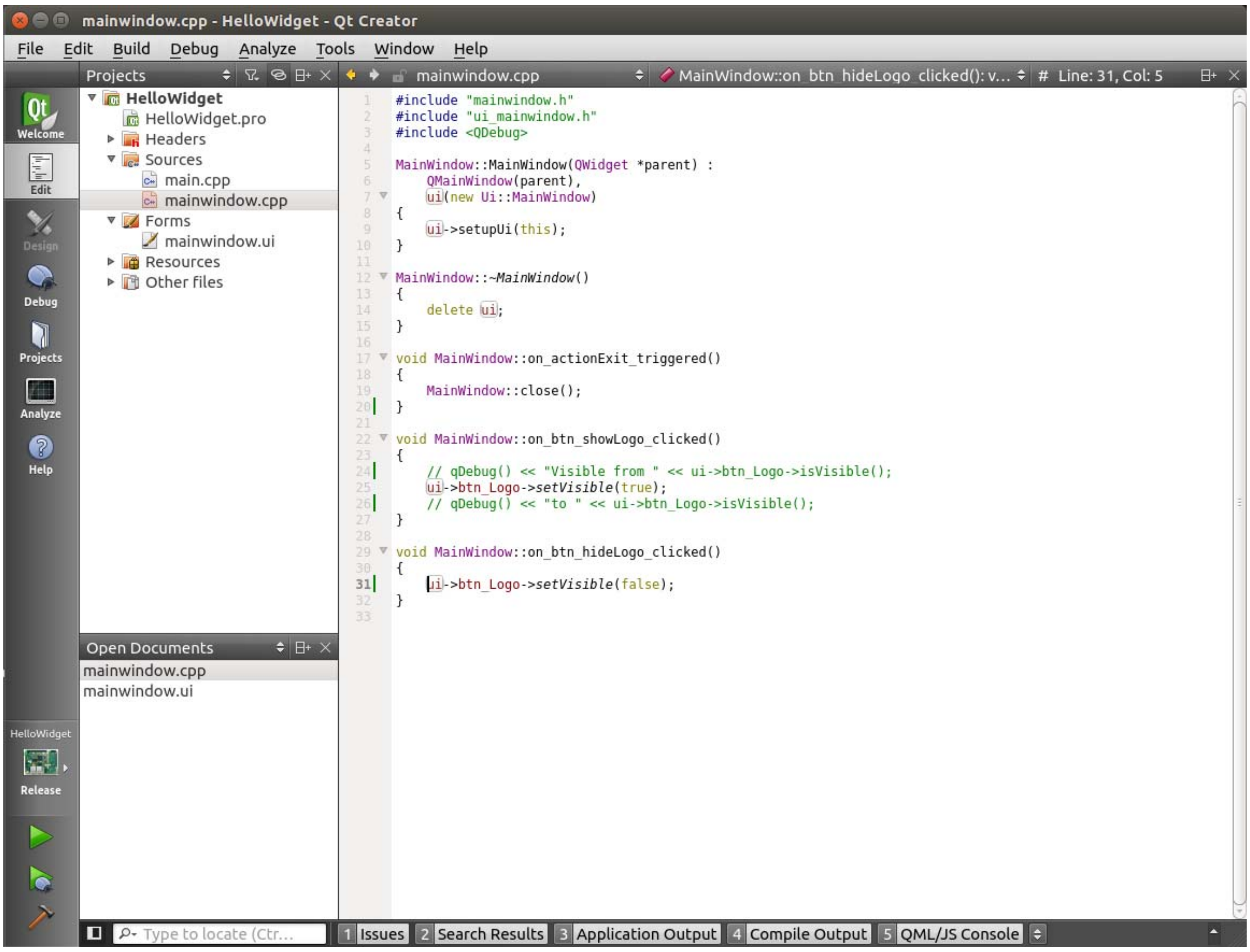


Go to slot

Select signal

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.

2.2.4 Compile and Run the Demo Application on the Target

Now we want to compile and run the demo application on the phyBOARD-Wega. 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 (X15) on the phyBOARD.

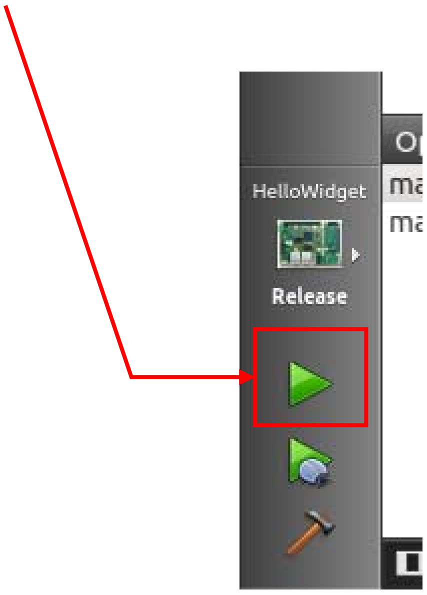
The correct Qt settings are already preset in the virtual machine, so the project can be build directly.

	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.

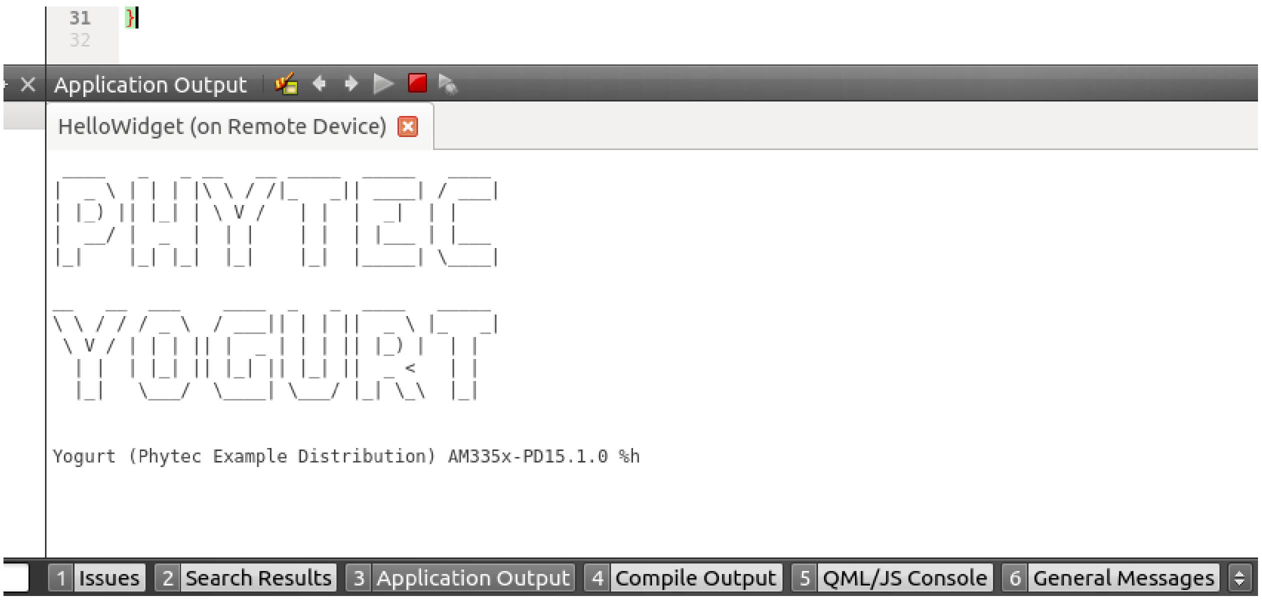
	Be sure that the target is connected via Ethernet and is powered on. As described before do not forget to stop all actually running *Qt* applications on the target.

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.



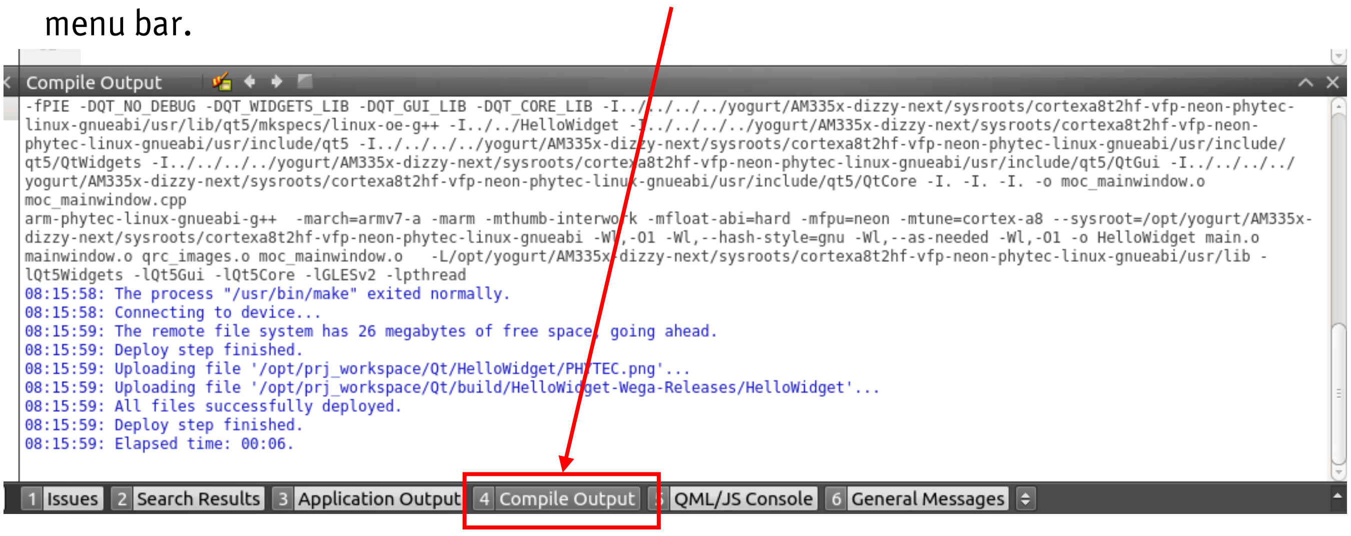
After the target is successfully deployed on to the target the *Application Output* frame is shown in the frame under the *mainwindow.cpp* source code where you can see the prompt from the phyBOARD-Wega.



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

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


Yogurt (Phytec Example Distribution) AM335x-PD15.1.0 %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.

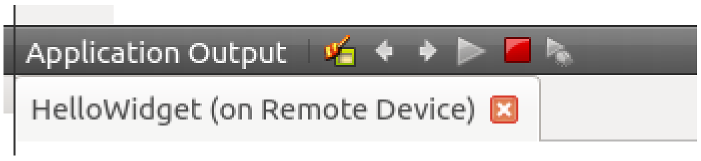
- To see the different build steps click ***Compile output*** which can be found in the bottom menu bar.



```
-fPIE -DQT_NO_DEBUG -DQT_WIDGETS_LIB -DQT_GUI_LIB -DQT_CORE_LIB -I../../../../yogurt/AM335x-dizzy-next/sysroots/cortexa8t2hf-vfp-neon-phytec-
linux-gnueabi/usr/lib/qt5/mkspecs/linux-oe-g++ -I../../HelloWidget -I../../../../yogurt/AM335x-dizzy-next/sysroots/cortexa8t2hf-vfp-neon-
phytec-linux-gnueabi/usr/include/qt5 -I../../../../yogurt/AM335x-dizzy-next/sysroots/cortexa8t2hf-vfp-neon-phytec-linux-gnueabi/usr/include/
qt5/QtWidgets -I../../../../yogurt/AM335x-dizzy-next/sysroots/cortexa8t2hf-vfp-neon-phytec-linux-gnueabi/usr/include/qt5/QtGui -I../../../../
yogurt/AM335x-dizzy-next/sysroots/cortexa8t2hf-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 -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 -Wl,-O1 -o HelloWidget main.o
mainwindow.o qrc_images.o moc_mainwindow.o   -L/opt/yogurt/AM335x-dizzy-next/sysroots/cortexa8t2hf-vfp-neon-phytec-linux-gnueabi/usr/lib -
lQt5Widgets -lQt5Gui -lQt5Core -lGLESv2 -lpthread
08:15:58: The process "/usr/bin/make" exited normally.
08:15:58: Connecting to device...
08:15:59: The remote file system has 26 megabytes of free space, going ahead.
08:15:59: Deploy step finished.
08:15:59: Uploading file '/opt/prj_workspace/Qt/HelloWidget/PHYTEC.png'...
08:15:59: Uploading file '/opt/prj_workspace/Qt/build/HelloWidget-Wega-Releases/HelloWidget'...
08:15:59: All files successfully deployed.
08:15:59: Deploy step finished.
08:15:59: Elapsed time: 00:06.
```

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.



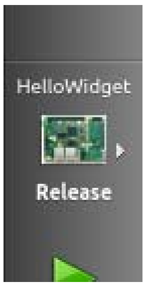
	You have successfully imported and built a *Qt* project with *Qt Creator*. You have also learned how to download and execute your application on the target.

2.2.5 Compile and Run the Demo Application on the Host

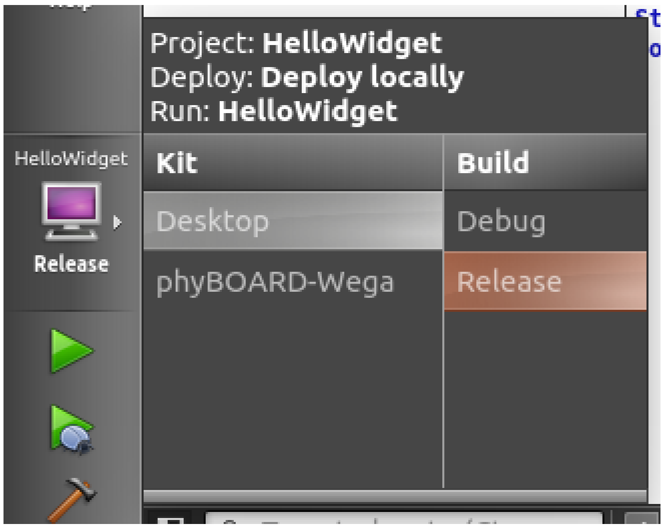
In some cases you do not want to compile and run the application on the target. For example if the phyBOARD-Wega is not connected to the host or if you only changed some user interface relevant things and you do not want to copy the application and all the necessary resources to the target. Running the 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 the following steps:

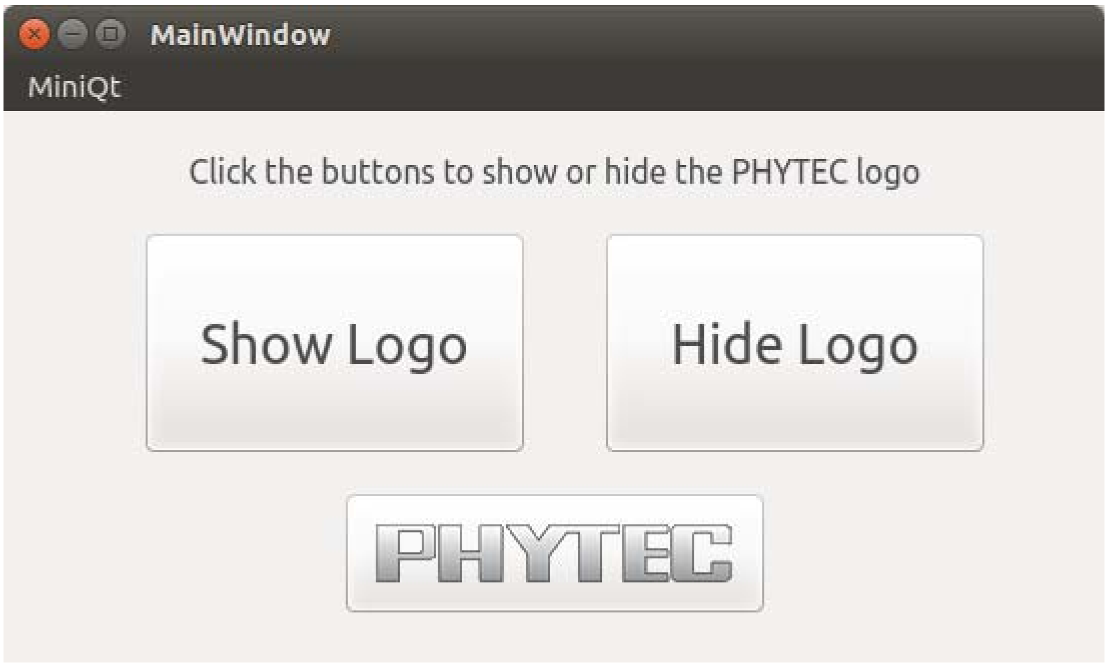
- In the gray bar on the left click the button with the small phyBOARD-Wega picture.



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



- 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.

2.2.6 Debugging the Demo Application

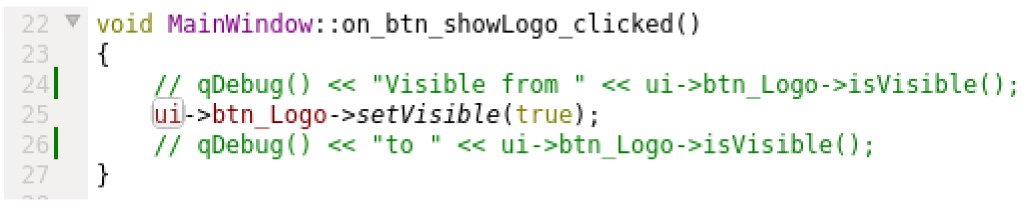
We finish the Qt Creator chapter by showing how to debug the *HelloWidget* demo application.

2.2.6.1 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*.

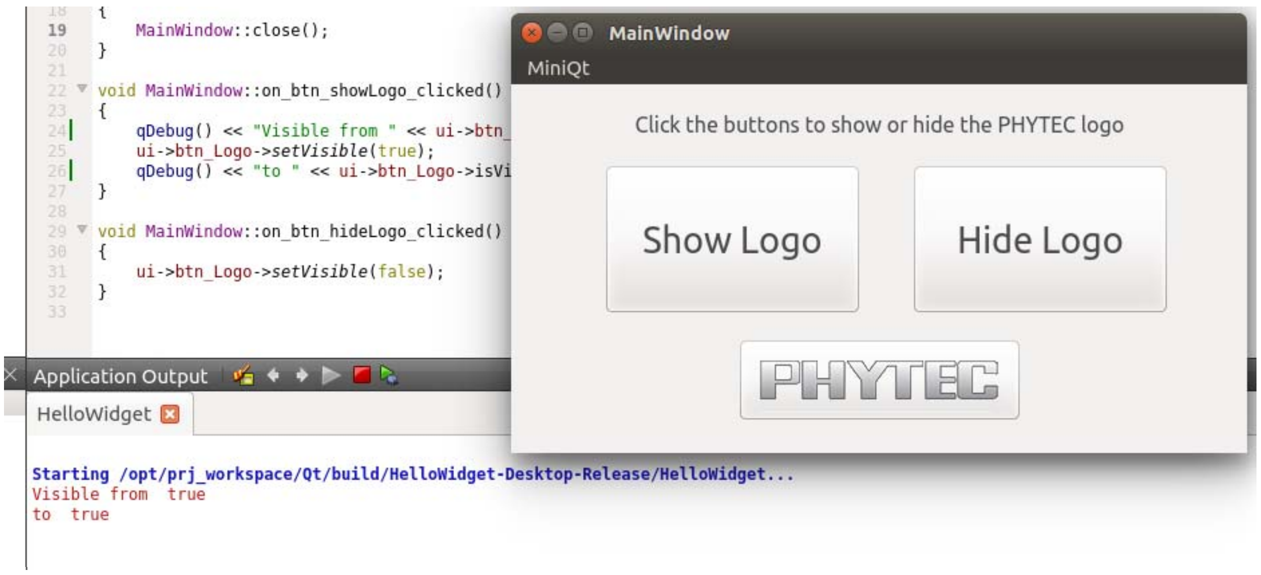
In this file you will also see two out-commented lines in the function *on_btn_showLogo_clicked()* which start with a *qDebug()* command. This is an example how to make use of qDebug().



```
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);
}
```

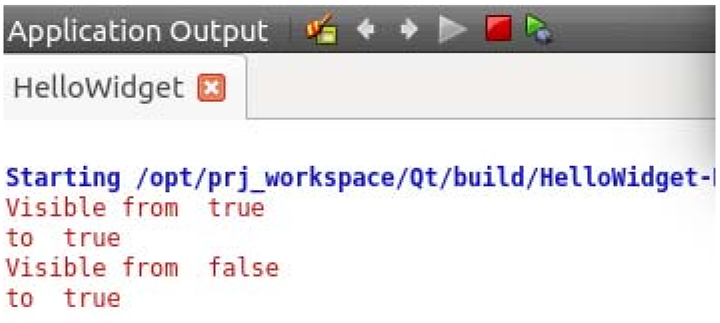
```
Starting /opt/prj_workspace/Qt/build/HelloWidget-Desktop-Release/HelloWidget...
Visible from  true
to  true
```

- 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.*



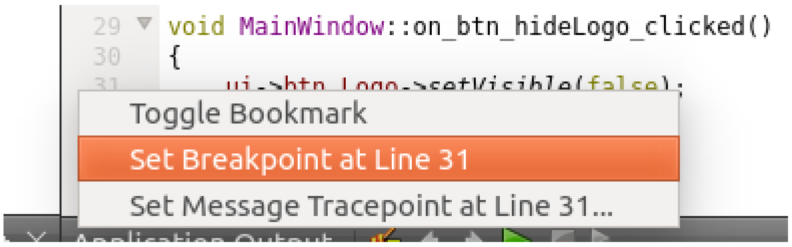
```
Starting /opt/prj_workspace/Qt/build/HelloWidget-
Visible from  true
to  true
Visible from  false
to  true
```

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

2.2.6.2 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.

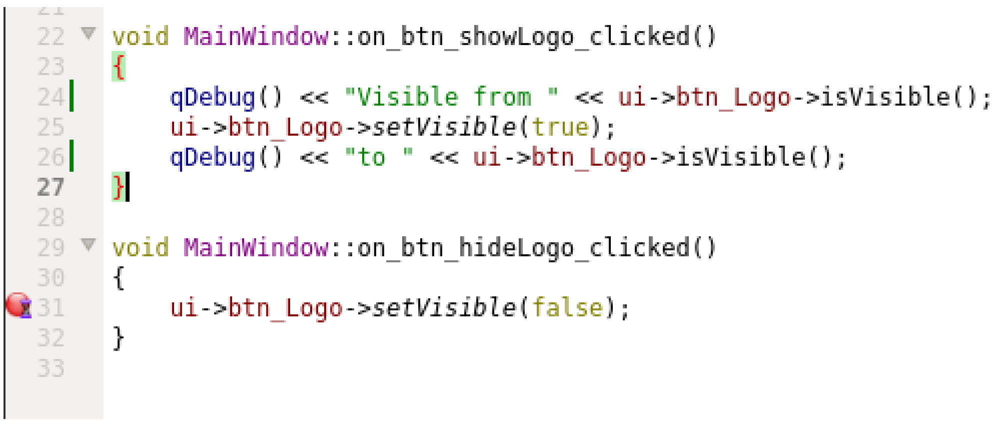


```
void MainWindow::on_btn_hideLogo_clicked()
{
    ui->btn_Logo->setVisible(false);
```

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

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

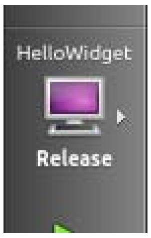
You now see a red filled circle with a small sand glass in front of this line.



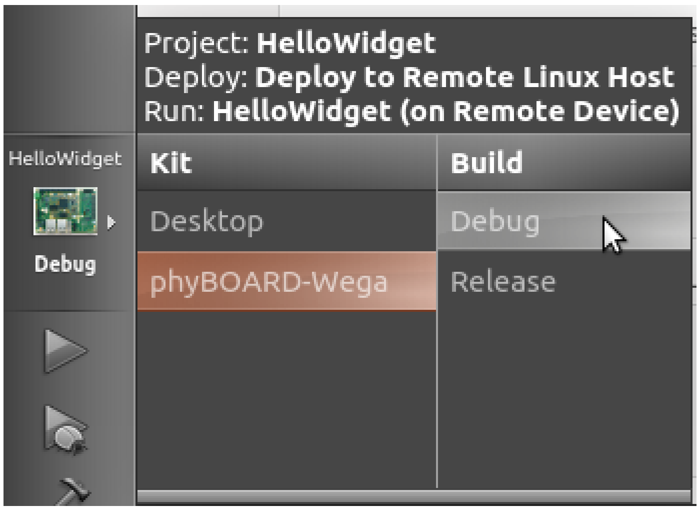
```
22  void MainWindow::on_btn_showLogo_clicked()
23  {
24      qDebug() << "Visible from " << ui->btn_Logo->isVisible();
25      ui->btn_Logo->setVisible(true);
26      qDebug() << "to " << ui->btn_Logo->isVisible();
27  }
28
29  void MainWindow::on_btn_hideLogo_clicked()
30  {
31      ui->btn_Logo->setVisible(false);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

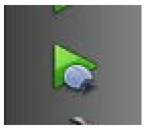


- In the opened context menu select ***phyBOARD-Wega*** as *Kit* and ***Debug*** as *Build*.

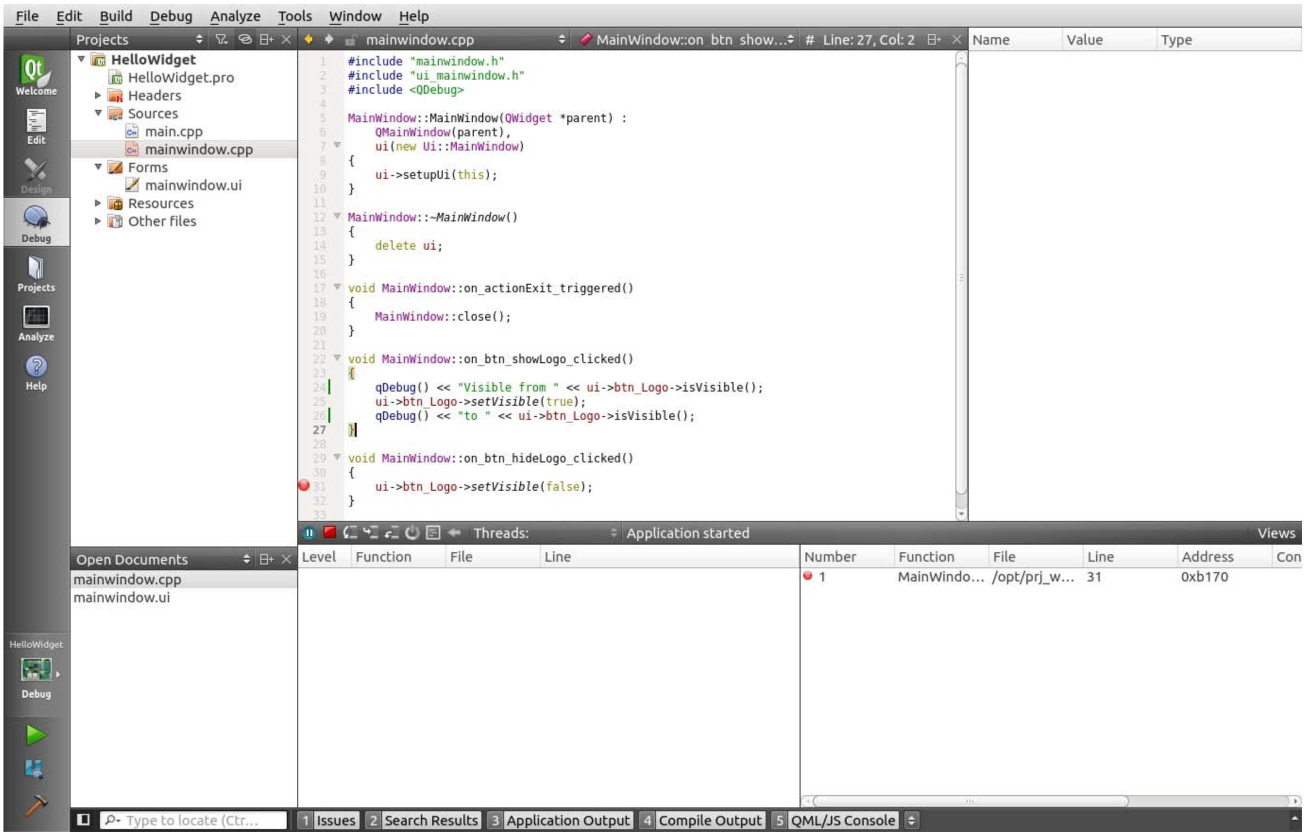


Be sure that the target is connected via Ethernet to the host. If you use our PEB-AV-01 with a connected monitor you must also connect an USB mouse to use the demo application.

- 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);
}
```

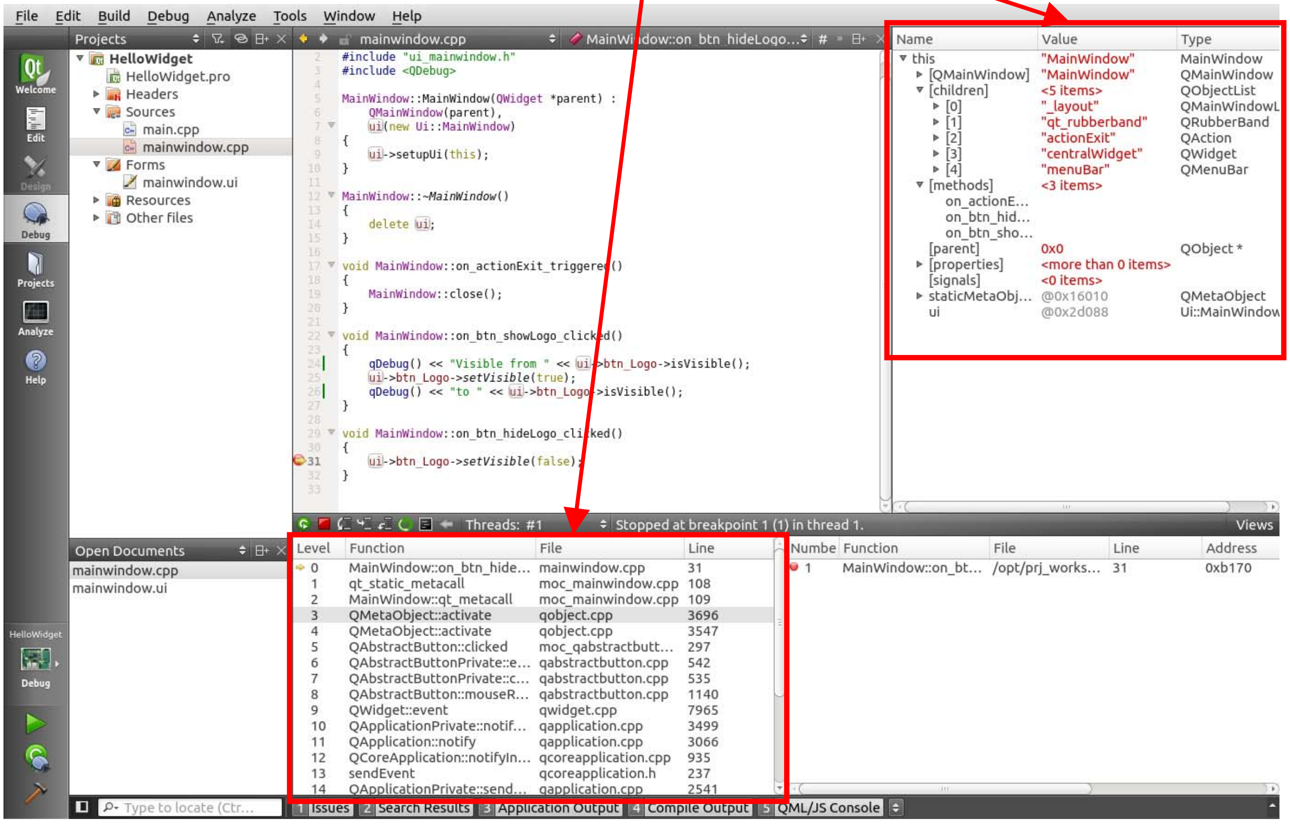
Level	Function	File	Line

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);
}
```

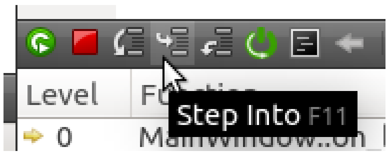
Name	Value	Type
this	"MainWindow"	MainWindow
[QMainWindow]	"MainWindow"	QMainWindow
[children]	<5 items>	QObjectList
[0]	"_layout"	QMainWindowL
[1]	"qt_rubberband"	QRubberBand
[2]	"actionExit"	QAction
[3]	"centralWidget"	QWidget
[4]	"menuBar"	QMenuBar
[methods]	<3 items>	
on_actionE…		
on_btn_hid…		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

Threads: #1   Stopped at breakpoint 1 (1) in thread 1.

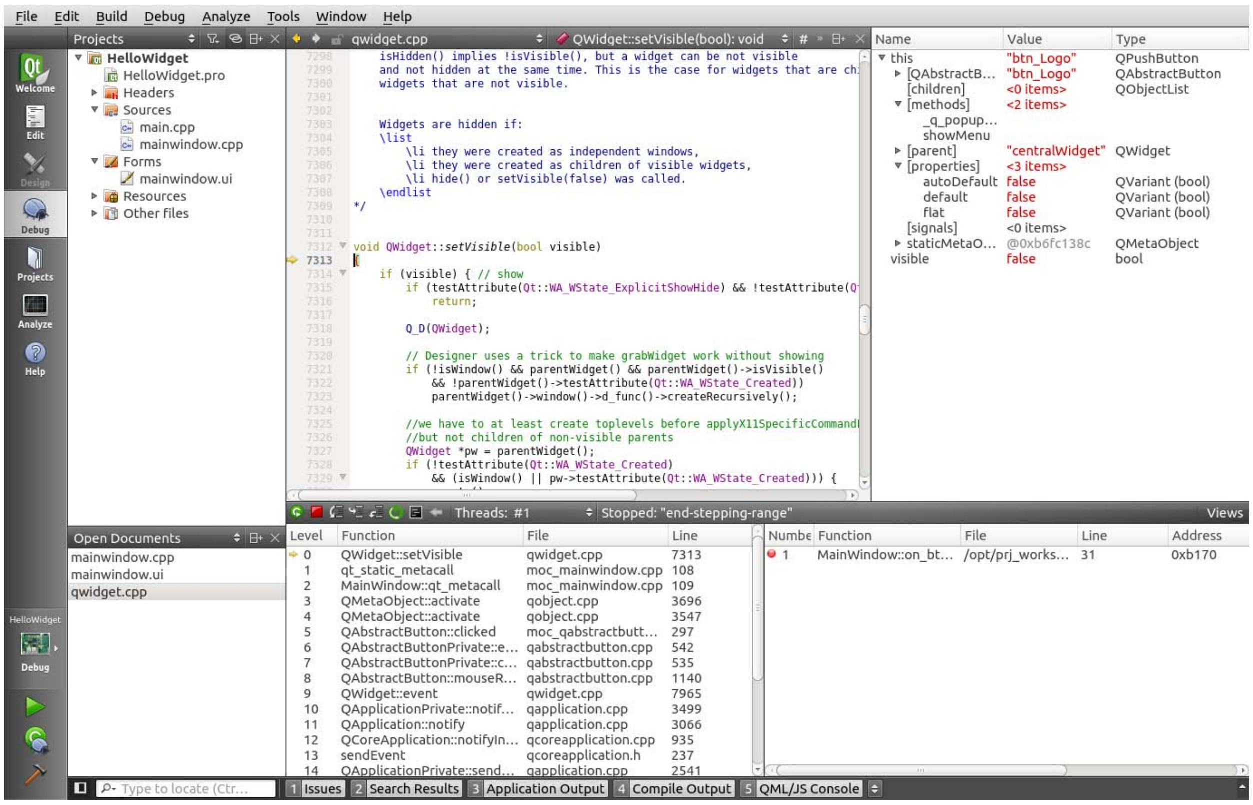
Level	Function	File	Line
0	MainWindow::on_btn_hide…	mainwindow.cpp	31
1	qt_static_metacall	moc_mainwindow.cpp	108
2	MainWindow::qt_metacall	moc_mainwindow.cpp	109
3	QMetaObject::activate	qobject.cpp	3696
4	QMetaObject::activate	qobject.cpp	3547
5	QAbstractButton::clicked	moc_qabstractbutt…	297
6	QAbstractButtonPrivate::e…	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))) {
```

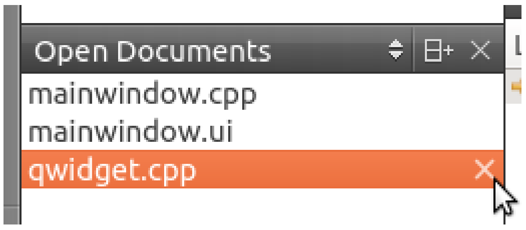
Name	Value	Type
this	"btn_Logo"	QPushButton
[QAbstractB...	"btn_Logo"	QAbstractButton
[children]	<0 items>	QObjectList
[methods]	<2 items>	
_q_popup...		
showMenu		
[parent]	"centralWidget"	QWidget
[properties]	<3 items>	
autoDefault	false	QVariant (bool)
default	false	QVariant (bool)
flat	false	QVariant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

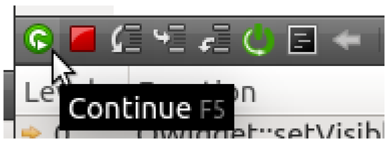
Numbe	Function	File	Line	Address
1	MainWindow::on_bt...	/opt/prj_works...	31	0xb170

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.

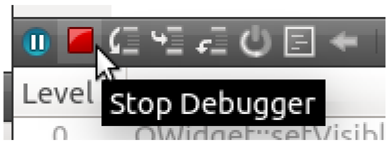


- Continue the stopped application by clicking on *Continue*.



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-Wega AM335x, 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-Wega 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-Wega AM335x.

3 Accessing the phyBOARD-Wega Features

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

3.1 Overview of the phyBOARD-Wega Peripherals

The phyBOARD-Wega 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-Wega. *Figure 2* highlights the location of each connector for easy identification.

Reference Designator	Description	See Section
X11	Secure Digital / Multi Media Card (Micro-slot)	*3.2.7*
X15	USB host connector (USB 2.0 Standard-A)	*3.2.4*
X16	Ethernet 0 connector (RJ45 with speed and link LED)	*3.2.3*
X17	Ethernet 1 connector (RJ45 with speed and link LED)	
X42	USB On-The-Go connector (USB Micro-AB)	*3.2.4*
X55	Mono Speaker output (2-pole Molex SPOX 2.5 mm pitch)	*3.2.5*
X65	CAN connector (2×5 pin header 2.54 mm pitch)	*3.2.6*
X66	RS-232 with RTS and CTS (UART1; 2×5 pin header 2.54 mm pitch)	*3.2.2*
X67	Power supply 5 V only (via 6-pole WAGO male header, or 2-pole PHOENIX MINI COMBICON base strip)	*3.2.1.1*
X69	Expansion connector (2×30 socket connector 2 mm pitch)	*3.2.11*
X70	A/V connector #1 (2×20 socket connector 2 mm pitch)	*3.2.8*
X71	A/V connector #2 (2×8 socket connector 2 mm pitch)	
X72	Optional 5 V power supply via USB Micro-AB connector	*3.2.1.1*
X73	Stereo Line Out and Line In connector (2×3 pin header 2.54 mm pitch)	*3.2.5*

Table 2: phyBOARD-Wega 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-Wega is populated with three LEDs to indicate the status of the USB VBUS voltages, as well as of the power supply voltage.

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

LED	Color	Description	See Section
D7	green	Indicates presence of VBUS1 at the USB host interface	*3.2.4*
D8	green	Indicates presence of VBUS0 at the USB OTG interface	*3.2.4*
D58	red	3.3 V voltage generation of the phyBOARD-Wega	*3.2.1.2*

Table 3: phyBOARD-Wega LEDs Descriptions

3.1.3 Switches

The phyBOARD-Wega is populated with two switches, one to reset the phyBOARD-Wega 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
S2	Reset Button	*3.2.9*
S4	Boot Switch	*3.2.8*

Table 4: phyBOARD-Wega Switches Description

3.1.4 Jumpers

The phyBOARD-Wega comes pre-configured with one removable jumper (JP) and 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 *section 4.6* and contact our sales team if you need jumper configurations different from the default configuration.

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

Figure 2 shows the location of jumper JP3.

Switch	Description	See Section
JP3	CAN Termination	*3.2.6*

Table 5: phyBOARD-Wega 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-Wega SBC

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

3.2.1 Power Supply

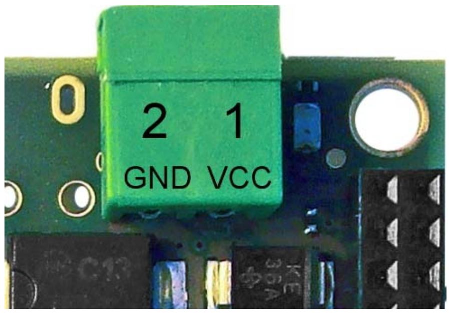
Do not change modules or jumper settings while the phyBOARD-Wega is supplied with power!

3.2.1.1 Power Connectors (X67 and X72)

The phyBOARD-Wega is available with three 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 (X67) suitable for a single 5 V supply voltage, or
2. an USB Micro-AB connector (X72) to connect a standard 5 V USB power supply, or
3. a 6-pole WAGO male header (X67) 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-Wega, the particular interfaces enabled while executing software, as well as whether an optional expansion board is connected to the carrier board.



PHOENIX MINI COMBICON base strip

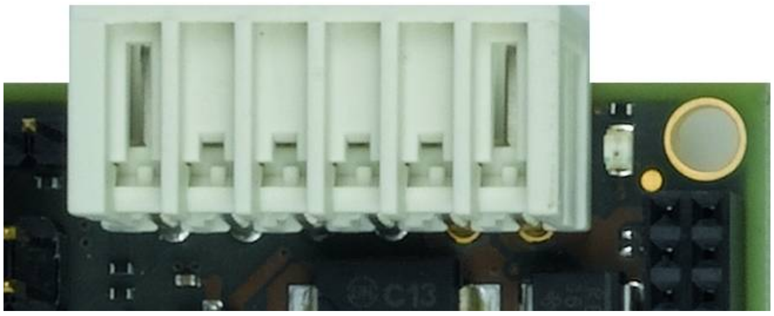

WAGO male header 6-pole

Figure 3: Power Supply Connectors

3.2.1.1.1 PHOENIX 2-pole MINI COMBICON Base Strip (X67)

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 3 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 X67

3.2.1.1.2 USB Micro-AB (X72)

If your board provides an USB Micro-AB female connector (X72) at the upper side of the board a standard USB Micro power supply with +5 V DC can be used to supply the phyBOARD-Wega.

⚠	Do not confuse the USB Micro connector on the upper side of the board with the one on the back side of the board which provides USB OTG connectivity. The USB Micro connector on the upper side is exclusively used for power supply and has no other USB functionality!

3.2.1.1.3 WAGO 6-pole Male Header (X67)

If a WAGO 6-pole male header is mounted on your board (*Figure 2* and *Figure 3*) 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 X67

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 D58

The red LED D58 right next to the power connector (*Figure 2*) indicates the presence of the 3.3 V supply voltage generated from the 5 V input voltage.

3.2.1.3 VBAT and RTC

The phyBOARD-Wega features an external RTC mounted on the phyCORE-AM335x module. It is used for real-time or time-driven applications. To backup the RTC on the module, a Gold cap (C339) (*Figure 2*) is placed on the phyBOARD-Wega. This voltage source is connected to the backup voltage pin VBAT_IN_4RTC (A2) of the phyCORE-AM335x and supplies the RTC and some critical registers of the Power Management IC when the primary system power, VCC5V_IN, is removed. The backup supply lasts approximately 17 ½ days.

3.2.2 UART Connectivity (X66 and X69)

The AM335x SOM supports up to 6 so called UART units. On the phyBOARD-Wega the TTL level signals of UART0 (the standard console), UART2 and UART3 are routed to expansion connector X69. UART1 is available at pin header connector X66 at RS-232 level.

	The Evaluation Board (PEB-EVAL-01) delivered with the kit plugs into the expansion connector and allows easy use of the standard console (UART0) which is required for debugging. Please find additional information on the Evaluation Board in the Application Guide for phyBOARD Expansion Boards (L-793e).

Further information on the expansion connector can be found in *section 4.6.4.*

Pin header connector X66 is located next to the USB host connector (*Figure 4)* and provides the UART1 signals of the AM335x at RS-232 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 8* shows the signal mapping of the RS-232 level signals at connector X66.

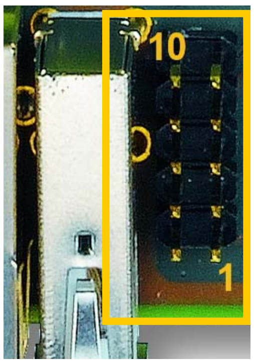
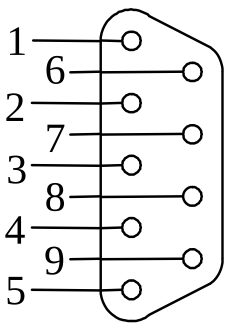


Figure 4: RS-232 Interface Connector (X66)

Pin	Signal	Pin	Signal
1	NC	2	NC
3	UART1_RXD_RS232	4	UART1_RTS_RS232
5	UART1_TXD_RS232	6	UART1_CTS_RS232
7	NC	8	NC
9	GND	10	NC

Table 8: Pin Assignment of RS-232 Interface Connector X66

An adapter cable is included in the phyBOARD-Wega AM335x Kit to facilitate the use of the UART1 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 5:	GND

Figure 5: RS-232 Connector Signal Mapping

3.2.2.1 Software Implementation

Only */dev/ttyO0* for UART0 and */dev/ttyO1* for UART1 have been implemented within the BSP.

The standard console UART0 is accessible as */dev/ttyO0* and is mainly used for debugging and control of software updates.

UART1 can be accessed as */dev/ttyO1*. It is intended for user applications.

3.2.3 Ethernet Connectivity (X16 and X17)

The Ethernet interfaces of the phyBOARD-Wega are accessible at two RJ45 connectors X16 (Ethernet 0) and X17 (Ethernet 1).

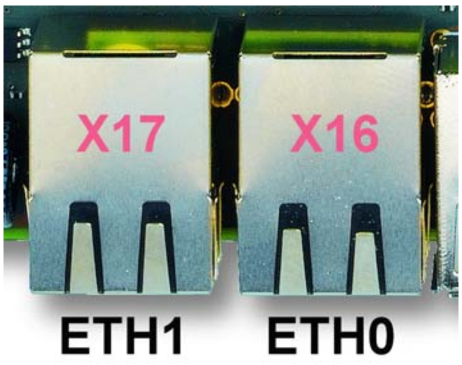


Figure 6: Ethernet Interfaces at Connectors (X16 and X17)

Both Ethernet interfaces are configured as 10/100Base-T networks. The LEDs for LINK (green) and SPEED (yellow) indication are integrated in the connector. Both Ethernet transceivers support HP Auto-MDIX, 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 two 10/100 Mbit Ethernet interfaces are being provided as network interfaces *eth0* with static IP 192.168.3.11 and *eth1* with static IP 192.168.4.11 by default.

Both interfaces offer a standard Linux network port which can be programmed using the BSD socket interface.

3.2.4 USB Connectivity (X15 and X42)

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

USB0 is accessible at connector X42 (USB Micro-AB) located at the back side of the phyBOARD-Wega. It 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.

USB1 is accessible on the top at connector X15 (USB Standard-A) and is configured as USB host.

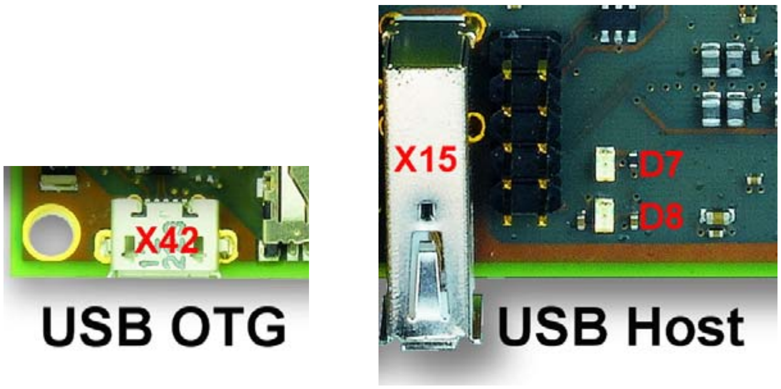


Figure 7: Components supporting the USB Interfaces

LED D8 displays the status of USB0_VBUS and LED D7 the status of USB1_VBUS.

Numerous jumpers allow to configure the USB interfaces according to your needs. Please refer to *section 4.6.1* for more information.

3.2.4.1 Software Implementation

3.2.4.1.1 USB Host

The AM335x 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-Wega behave like an USB flash drive:

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

fdisk is started and expects commands. Create a new partition with the following *fdisk* shortcut keys:

N ► p ► 1 ► <ENTER> ► <ENTER> ► w

```
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 file system (replace *sdc1* by the device the computer really shows):

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

Using USB OTG as host is supported by the BSP with the module *g_zero,* but the BSP does not provide support for using USB OTG as host.

3.2.5 Audio Interface (X55 and X73)

The audio interface provides a method of exploring AM335x's audio capabilities. The phyBOARD-Wega is populated with an audio codec at U35. The audio codec is connected to the AM335x's McASP0 interface to support stereo line input and stereo line output at connector X73. In addition to that the phyBOARD-Wega has one direct mono speaker output (1 W) at the Molex connector X55.

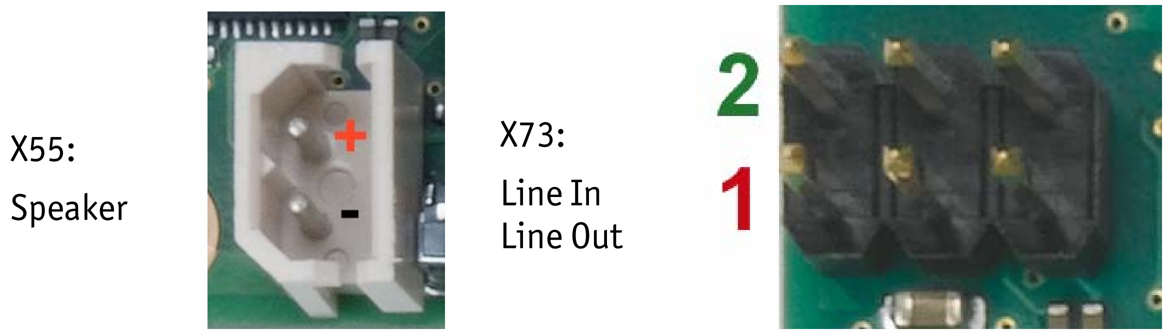


Figure 8: Audio Interfaces at Connectors (X55 and X73)

Pin	Signal	Pin	Signal
1	LINE_IN_L	2	LINE_IN_R
3	AGND	4	AGND
5	LINE_OUT_L	6	LINE_OUT_R

Table 9: Pin Assignment of Audio Connector X73

Pin	Signal	Description
1	SPOP	Class-D positive differential output
2	SPOM	Class-D negative differential output

Table 10: Pin Assignment of Audio Connector X55

The audio codec can be configured via I²C interface I²C0 at address 0x18.

For additional audio applications the McASP0 interface of the AM335x including the signals X_MCASP0_AHCLKX, X_I2S_CLK, X_I2S_FRM, X_I2S_ADC and X_I2S_DAC are routed to the A/V connector X71 (please refer to *section 4.6.2* for additional information on the A/V connector).

Please refer to the audio codec's reference manual for additional information regarding the special interface specification.

3.2.5.1 Software Implementation

Audio support on the module is done via the I^{2}S interface and controlled via I^{2}C.

On the phyBOARD-Wega the audio codec's registers can be accessed via the I2C0 interface at address 0x18 (7-bit MSB addressing).

To check if your soundcard driver is loaded correctly and what the device is called, type:

```
aplay -L
```

A wav file can be copied to the board using scp and then played through the sound interface.

```
aplay -vv file.wav
```

Not all *wave* formats are support by the sound interface. Use *Audacity* on your host to convert any file into *44100:S16_LE wave* format which should be supported on all platforms.

To identify channel numbers run *speaker-test*:

```
speaker-test -c 2 -t wav
```

An external audio source can be connected to the input, in order to record a sound file which then can be played back.

```
arecord -c 2 -r 441000 -f S16_LE test.wav;
aplay test.wav
```

Inspect your soundcards capabilities with:

```
alsamixer
```

You should see a lot of options as the audio-ICs have a lot of features you can play with. It might be better to open *alsamixer* via *ssh* instead of the serial console, as the console graphical effects could be better. You have either *mono* or *stereo* gain controls for all the mix points. *"MM"* means the feature is muted, which can be toggled by hitting ***m***.

3.2.6 CAN Connectivity (X65, JP3)

The Controller Area Network (CAN) bus offers a low-bandwidth, prioritized message fieldbus for serial communication between microcontrollers. It efficiently supports distributed real time control with a high level of security. The DCAN module of the AM335x implements the CAN protocol according to the CAN 2.0B protocol specification and supports bitrates up to 1 Mbit/s.

The CAN1 interface of the phyBOARD-Wega AM335x is accessible at connector X65 (2×5 pin header, 2.54 mm pitch).

Jumper JP3 can be installed to add a 120 Ohm termination resistor across the CAN data lines if needed.

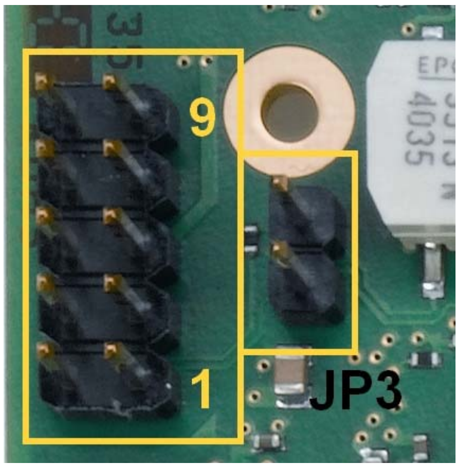


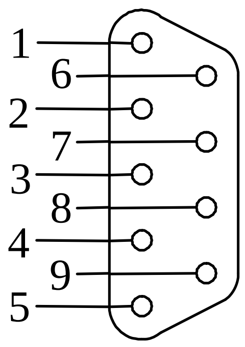
Figure 9: Components supporting the CAN Interface

Table 11 below shows the signal mapping of the CAN1 signals at connector X65.

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

Table 11: Pin Assignment of CAN Connector X65

An adapter cable is included in the phyBOARD-Wega AM335x 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

Depending on the muxing options a second CAN interface (CAN0) is available on expansion port X69 (*section 4.6.4*). CAN0 (TX and RX) can be used instead of UART0 (RX and TX) or MMC2 (DAT1 and DAT2).

3.2.6.1 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.

The configuration is done with the *systemd* configuration file */lib/systemd/system/can0.service*. For a persistent change of the default bitrates change the configuration in *meta-yogurt/recipes-core/systemd/systemd/can0.service* instead and rebuild the root filesystem.

To get the information on *can0* (which represents AM335x's CAN1 routed to X65) 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 about using and options.

3.2.7 Secure Digital Memory Card/ MultiMedia Card (X11)

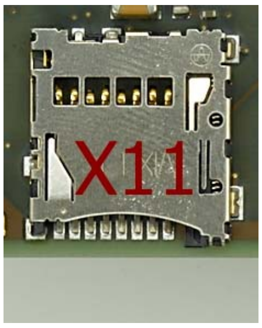


Figure 11: SD / MM Card interface at connector(X11)

The phyBOARD-Wega provides a standard microSDHC card slot at X11 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/ Push-out of card.

DIP switch S4 allows to toggle between NAND boot and boot from SD card. In order to boot from SD card S4 must be switched ON (refer to *section 3.2.8* for further information).

3.2.7.1 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 not hot pluggable, so 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.

3.2.8 Boot Mode (S4)

The pyhBOARD-Wega has two defined boot sequences which can be selected with DIP switch S4.

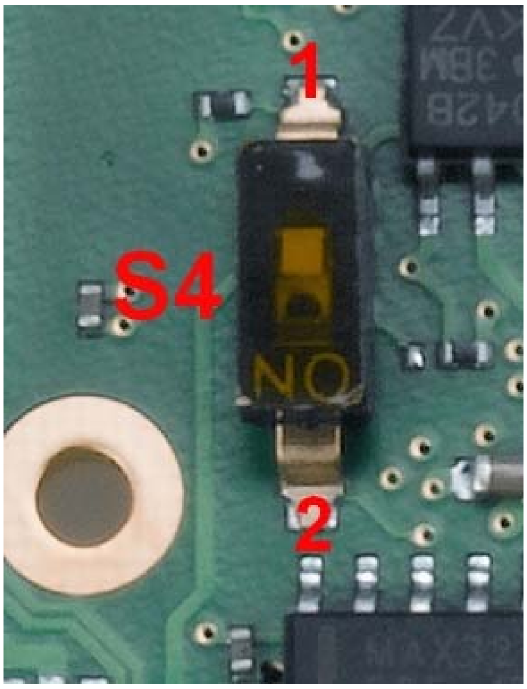


Figure 12: Boot Switch (S4)

Boot Mode	Description
Boot mode 1 (S4 = OFF)	SYSBOOT[4:0] = 10011b → NAND, NANDI2C, MMC0, UART0
Boot mode 2 (S4 = ON)	SYSBOOT[4:0] = 10111b → MMC0, SPI0, UART0, USB

Table 12: Boot Switch Configuration (S4)

3.2.9 System Reset Button (S2)

The phyBOARD-Wega is equipped with a system reset button at S2. Pressing this button will toggle the X_nRESET_IN pin (X64A11) of the phyCORE SOM low, causing the module to reset. Additionally, the reset signal nRESET_OUT is generated on the module to also reset the peripherals on the carrier board.

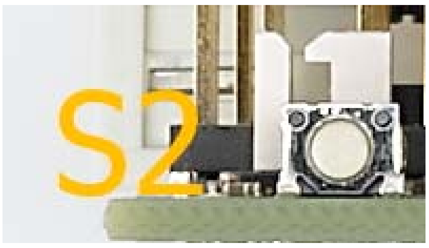


Figure 13: System Reset Button (S2)

3.2.10 Audio/Video connectors (X70 and X71)

The Audio/Video (A/V) connectors X70 and X71 provide an easy way to add typical A/V functions and features to the phyBOARD-Wega. 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.

For further information of the A/V connectors see *chapter 4.6.2*. Information on the expansion boards available for the A/V Connectors can be found in the Application Guide for phyBOARD-Wega Expansion Boards (L-793e).

3.2.11 Expansion connector (X69)

The expansion connector X69 provides an easy way to add other functions and features to the phyBOARD-Wega. Standard interfaces such as JTAG, UART, MMC2, SPI and I^2C as well as different supply voltages and some GPIOs and analog inputs are available at the expansion female connector.

For further information of the expansion connector and the pinout see *chapter 4.6.4*. Information on the expansion boards available for the expansion connector can be found in the Application Guide for phyBOARD-Wega Expansion Boards (L-793e).

3.2.12 Addressing the RTC

The RTC RV-4162-C7 on the phyCORE-AM335x 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.

Please note that in case you need to use RTC's interrupt, you need to shortcut pin 37 *x_intr1* and pin 40 *x_intr_RTCn* of the expansion connector X69. But this can only be done if you do not need this interrupt line for a touch controller.

3.2.13 CPU Core Frequency Scaling

The phyBOARD-Wega AM335x 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. In case you have an AM3354 or AM3359 with a maximum of 800 MHz the result will be:

```
300000 600000 720000 800000.
```

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

You can decrease the maximum frequency (e.g. to 720000)

```
echo 720000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
```

or increase the minimum frequency (e.g. to 600000)

```
echo 600000 > /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 behavior 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 600000). Type: `echo 600000 > /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.

3.2.14 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 2.2.1* 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 virtual machine
- A description how to update the Bootloader and how to write the Kernel and Root File System into the Flash
- Detailed information on the different interfaces and features of the phyBOARD-Wega 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 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 virtual machine under */opt/PHYTEC_BSPs/phyBOARD-WEGA/*. All necessary tools and programs are already pre-installed to directly start with *Yocto*.

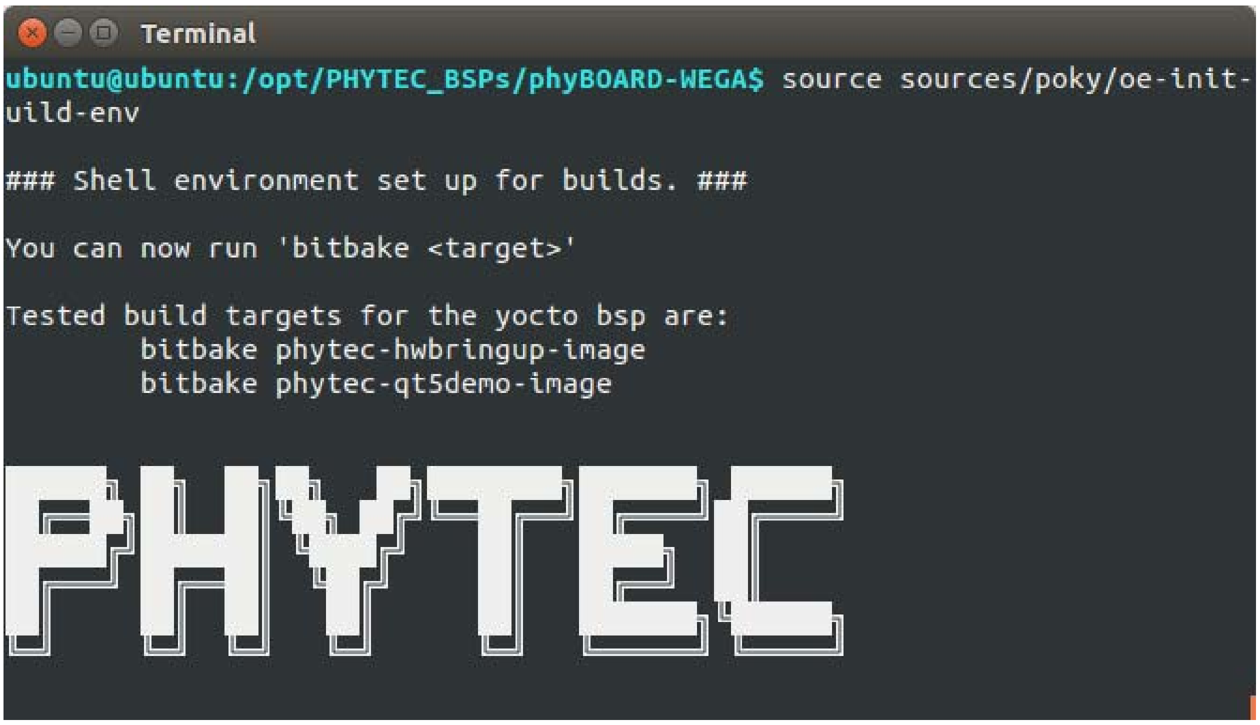
Open a new terminal if you have not already done and change to the directory of the pre-installed BSP:

Terminal

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

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`



```
ubuntu@ubuntu:/opt/PHYTEC_BSPs/phyBOARD-WEGA$ source sources/poky/oe-init-
uild-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
```

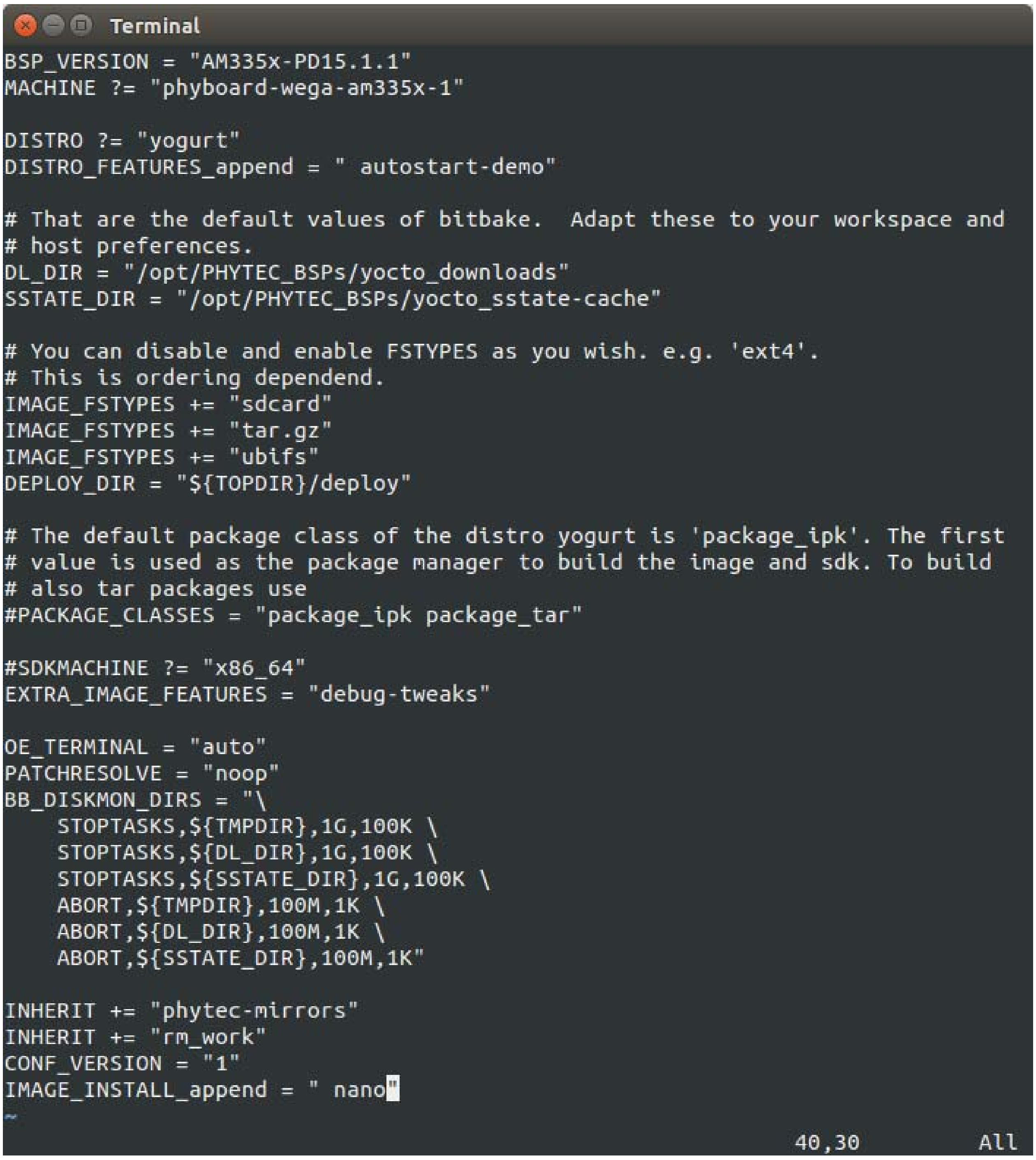
The environment is set and we are now in the build folder. In this example we want to add the *nano editor* to the image. Usually new packages are downloaded from an appropriate server prior to building the image unless they are stored locally. In the virtual machine the package for the *nano editor* 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-wega-am335x-1*. This is the correct setting for the *PEB-AV-01* HDMI display adapter. If you use the PEB-AV-02 LVDS/LCD adapter set the variable to: `phyboard-wega-am335x-2`



```
BSP_VERSION = "AM335x-PD15.1.1"
MACHINE ?= "phyboard-wega-am335x-1"

DISTRO ?= "yogurt"
DISTRO_FEATURES_append = " autostart-demo"

# That are the default values of bitbake.  Adapt these to your workspace and
# host preferences.
DL_DIR = "/opt/PHYTEC_BSPs/yocto_downloads"
SSTATE_DIR = "/opt/PHYTEC_BSPs/yocto_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"

#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"
CONF_VERSION = "1"
IMAGE_INSTALL_append = " nano"
~
                                                              40,30         All
```

- 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-WEGA/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.

Among other images you will find the kernel named *zImage*, the device tree named *zImage-am335x-wega-rdk.dtb* and the root file system named *phytec-qt5demo-image-phyboard-wega-am335x-1.ubifs* in the directory *deploy/images/phyboard-wega-am335x-1/*.

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

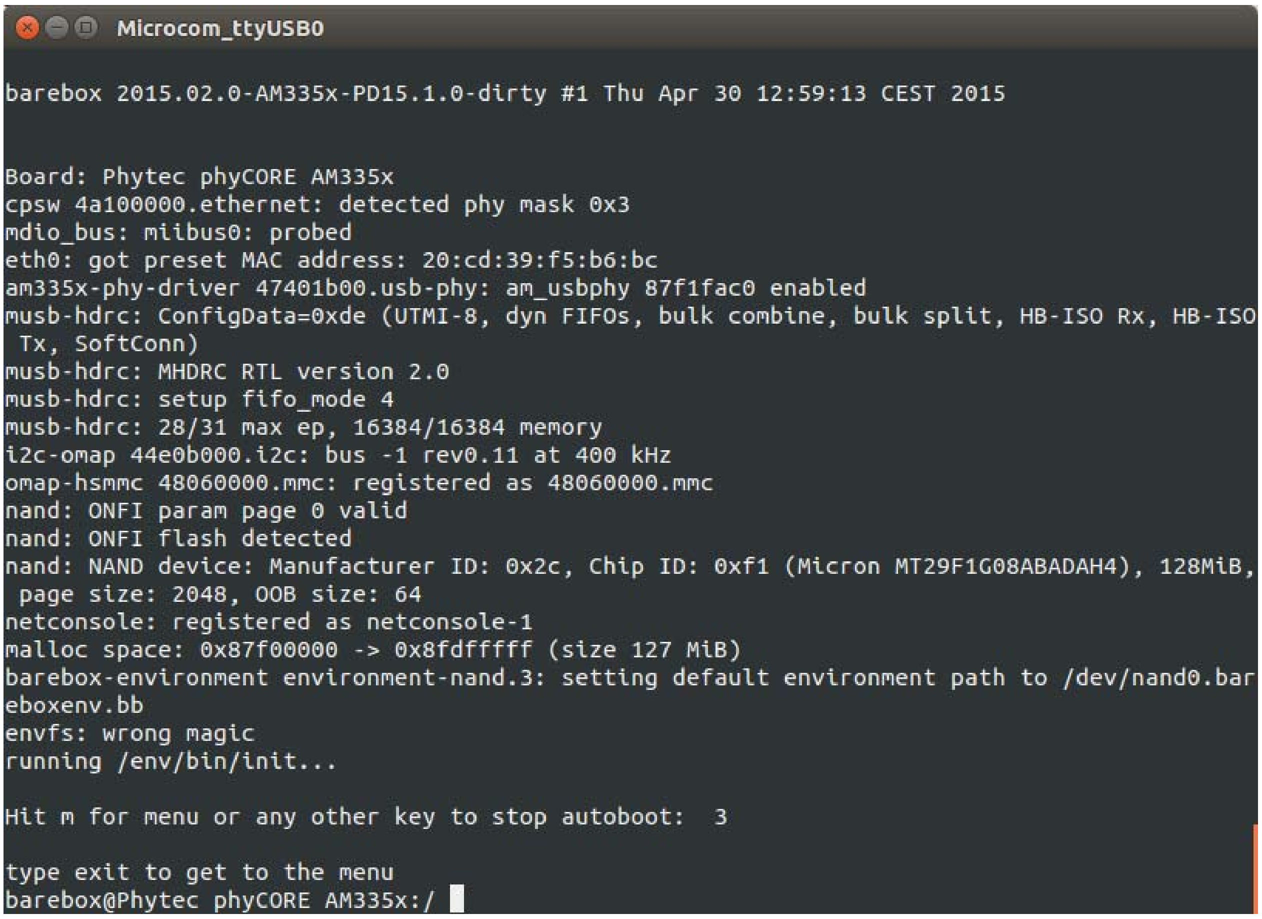
In this section you will find a description on how to write the newly created root filesystem into the phyBOARD-Wega's flash memory. Before the image can be written into the flash, it must be uploaded to the phyBOARD-Wega from a TFTP server. This will be done from the command line of the boot loader. 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-WEGA/build/deploy/images/phyboard-wega-am335x-1/*

- 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 2015.02.0-AM335x-PD15.1.0-dirty #1 Thu Apr 30 12:59:13 CEST 2015


Board: Phytec phyCORE AM335x
cpsw 4a100000.ethernet: detected phy mask 0x3
mdio_bus: miibus0: probed
eth0: got preset MAC address: 20:cd:39:f5:b6:bc
am335x-phy-driver 47401b00.usb-phy: am_usbphy 87f1fac0 enabled
musb-hdrc: ConfigData=0xde (UTMI-8, dyn FIFOs, bulk combine, bulk split, HB-ISO Rx, HB-ISO
 Tx, SoftConn)
musb-hdrc: MHDRC RTL version 2.0
musb-hdrc: setup fifo_mode 4
musb-hdrc: 28/31 max ep, 16384/16384 memory
i2c-omap 44e0b000.i2c: bus -1 rev0.11 at 400 kHz
omap-hsmmc 48060000.mmc: registered as 48060000.mmc
nand: ONFI param page 0 valid
nand: ONFI flash detected
nand: NAND device: Manufacturer ID: 0x2c, Chip ID: 0xf1 (Micron MT29F1G08ABADAH4), 128MiB,
 page size: 2048, OOB size: 64
netconsole: registered as netconsole-1
malloc space: 0x87f00000 -> 0x8fdfffff (size 127 MiB)
barebox-environment environment-nand.3: setting default environment path to /dev/nand0.bar
eboxenv.bb
envfs: wrong magic
running /env/bin/init...

Hit m for menu or any other key to stop autoboot:   3

type exit to get to the menu
barebox@Phytec phyCORE AM335x:/
```

- 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=192.168.3.10
  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 type:
  ```
  ubiformat /dev/nand0.root
  ubiattach /dev/nand0.root
  ubimkvol /dev/ubi0 root 0
  cp /mnt/tftp/phytec-qt5demo-image-phyboard-wega-am335x-1.ubifs
  /dev/ubi0.root
  ```

- Enter `boot` to boot the phyBOARD-Wega with the new root filesystem.

- 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*) and root file system (*phytec-qt5demo-image-phyboard-wega-am335x-*.ubifs*) from the */tftpboot* directory.

🦊	All files are also downloadable from our ftp server, or 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 you want to flash it, this chapter shows how to do it. In case that your phyBOARD-Wega AM335x does not start anymore because you damaged its software during the previous chapter, you are right here, too. In the latter case you will find all original images needed in the virtual machine.

⚠	Make sure that the BSP version matches the hardware version of the phyBOARD-Wega **and** the display adapter connected. Please visit http://www.phytec.eu and navigate to "Support" / "FAQ/Download" / "phyBOARDs – Single Board Computer" / "phyBOARD-Wega" 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-Wega AM335x 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 AM335x does not use file systems. Instead it is hard coded at which sectors of the SD card the AM335x 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-phywega-am335x-*.sdcard of=/dev/<DEVICE>
```

4.4.2 Flashing the Bootloader

Use a bootable SD card in order to boot into Barebox.

Without any valid bootloader in NAND, the board will automatically try to use the SD card for booting. Otherwise you need to force booting from SD card by switching DIP switch S4 to ***ON*** (*3.2.8*).

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
```

- Flash x-loader MLO by typing:
  ```
  barebox_update -t MLO.nand /boot/MLO
  ```

- 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 get a native environment with the same modifications as in our virtual machine hard disk image or for developers who want to use their own existing environment.
We will give an overview of the modifications which we made to the *Ubuntu* version used in the virtual machine hard disk image for the phyBOARD-Wega AM335x in comparison to the original *Ubuntu*. In the following we distinguish between optional and essential modifications. So you can see easily which changes are important to execute this Application Guide in case you do not want to use our modified *Ubuntu* version. The following step-by-step instruction is based on the use of *Ubuntu* 14.04.2.

⚠	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 aware of the complexity of actions necessary.

4.5.1 Essential settings

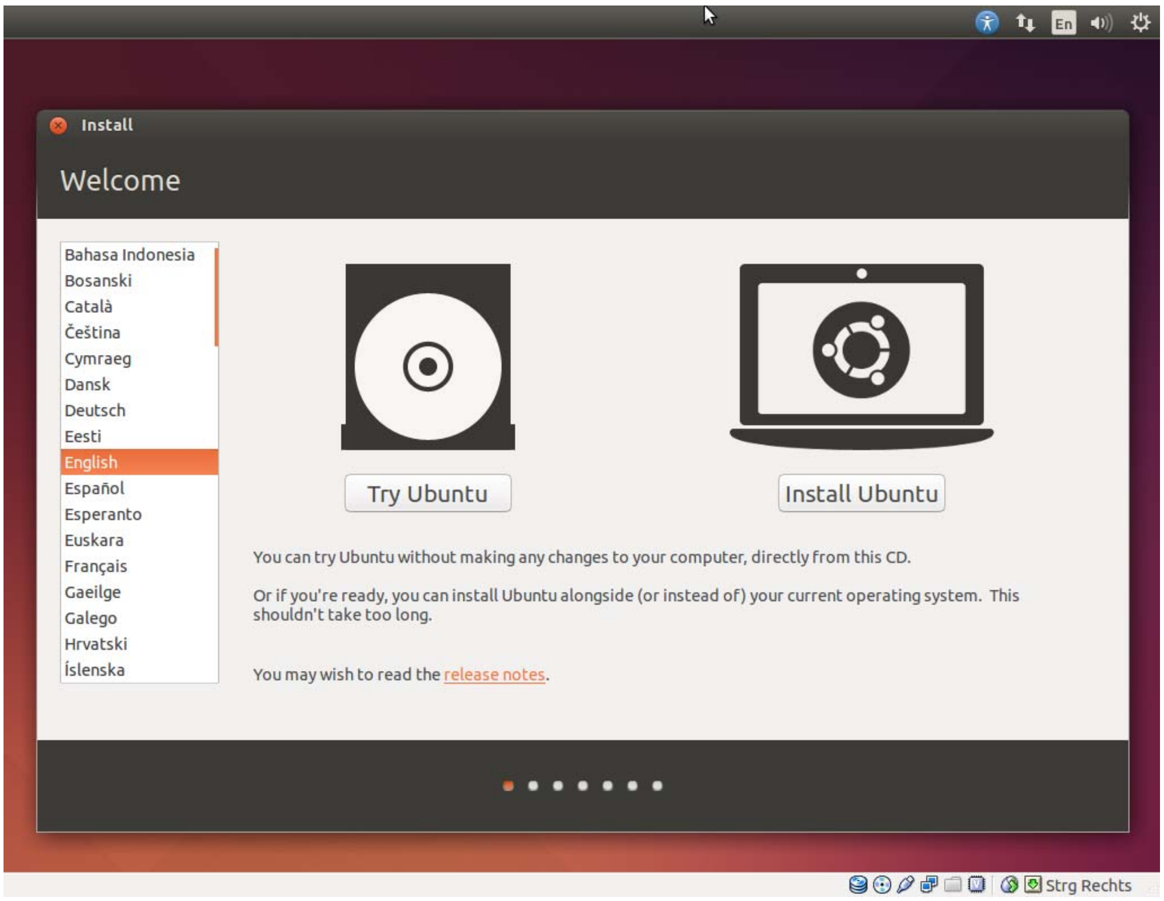
In the following section you get a short instruction about the important settings which are essential to guarantee the execution of the examples in this Application Guide.

4.5.1.1 Installing Ubuntu

Before we can start with the modification we must first install *Ubuntu*. *Ubuntu* version 14.04.2 64-bit, which we use, can be downloaded from *www.ubuntu.com*

⚠	We recommend to have at least 80 GB of free disk space available to install all required packages.

- Boot *Ubuntu* from the created bootable medium.
- Select ***Install Ubuntu*** in the first *Welcome* window.

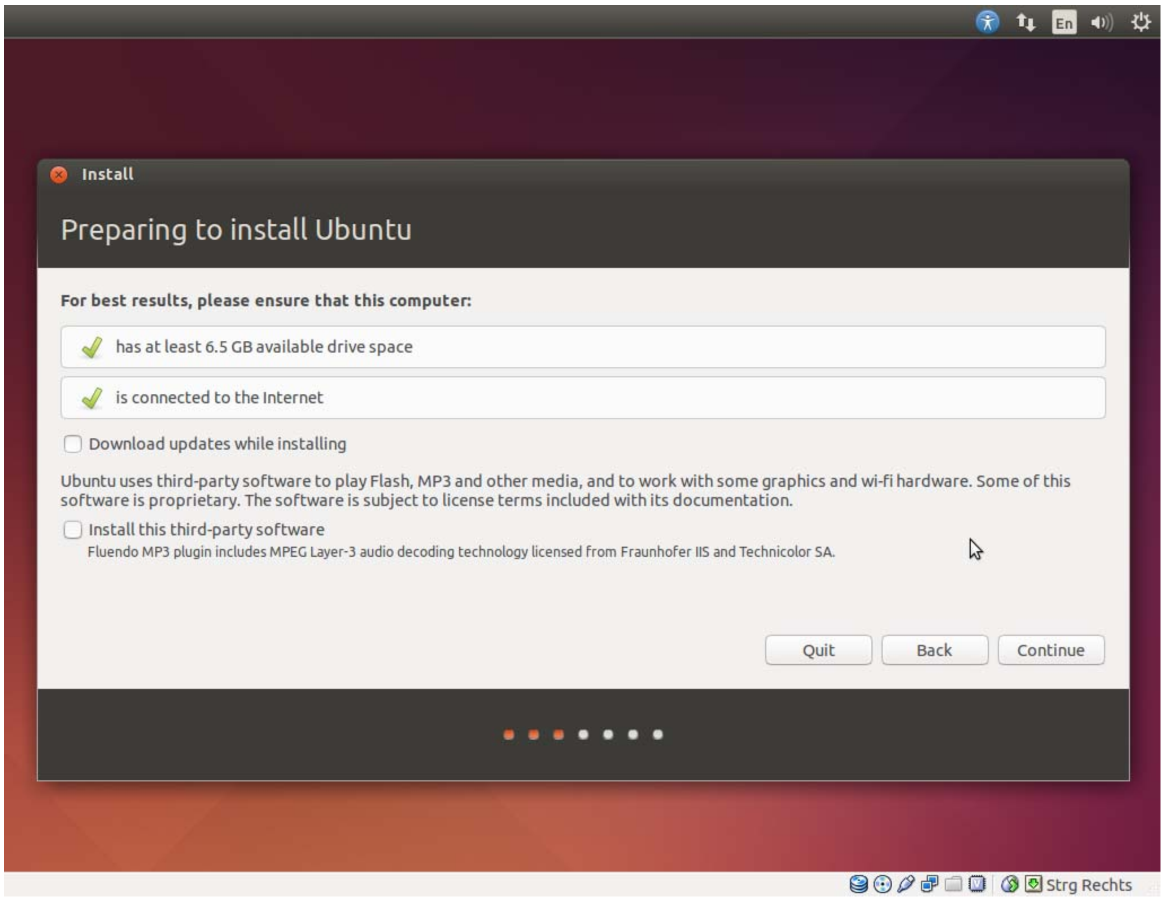


You can try Ubuntu without making any changes to your computer, directly from this CD.

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.

- Chosen one and 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*.

4.5.1.2 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.

4.5.1.3 Set the Git Configuration

The Board Support Package is strongly 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 the following two commands to directly set them
  ```
  git config --global user.email “your_email@example.com”
  git config --global user.name “name surname”
  ```

4.5.1.4 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 virtual machine hard disk image.

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

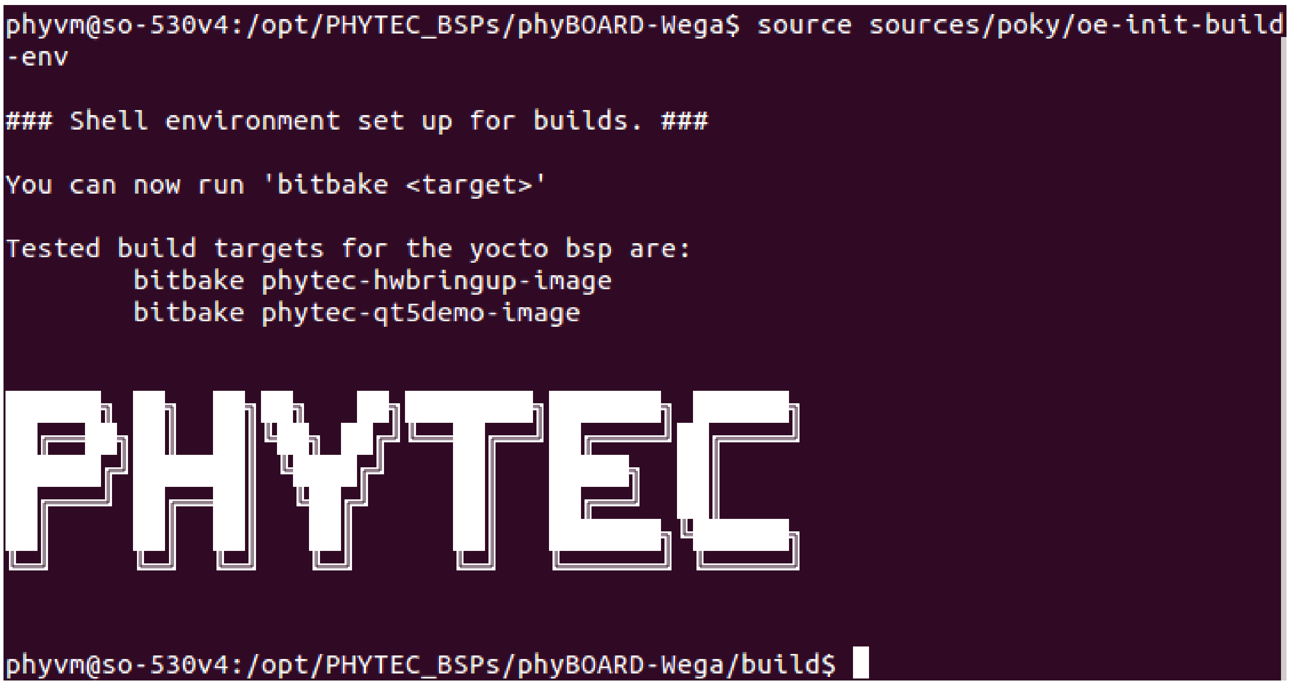
The *phyLinux* script is a basic management tool for PHYTEC *Yocto* BSP releases. It is mainly a tool to get started with the BSP structure. You can get all the BSP sources without the need of interacting with repo or git.

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

During the execution of the init command, you need to choose your processor platform (am335x), PHYTEC's BSP release number (PD15.1.1) and the hardware you are working on (phyboard-wega-am335x-1, or phyboard-wega-am335x-2).

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.

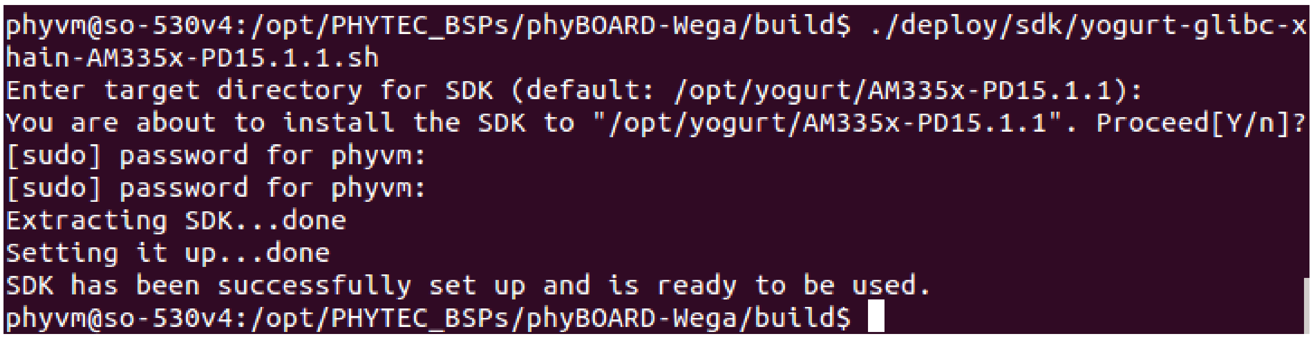
- Type:
  ```
  source sources/poky/oe-init-build-env
  ```



```
phyvm@so-530v4:/opt/PHYTEC_BSPs/phyBOARD-Wega$ 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

phyvm@so-530v4:/opt/PHYTEC_BSPs/phyBOARD-Wega/build$
```

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`



```
phyvm@so-530v4:/opt/PHYTEC_BSPs/phyBOARD-Wega/build$ ./deploy/sdk/yogurt-glibc-x
hain-AM335x-PD15.1.1.sh
Enter target directory for SDK (default: /opt/yogurt/AM335x-PD15.1.1):
You are about to install the SDK to "/opt/yogurt/AM335x-PD15.1.1". Proceed[Y/n]?
[sudo] password for phyvm:
[sudo] password for phyvm:
Extracting SDK...done
Setting it up...done
SDK has been successfully set up and is ready to be used.
phyvm@so-530v4:/opt/PHYTEC_BSPs/phyBOARD-Wega/build$
```

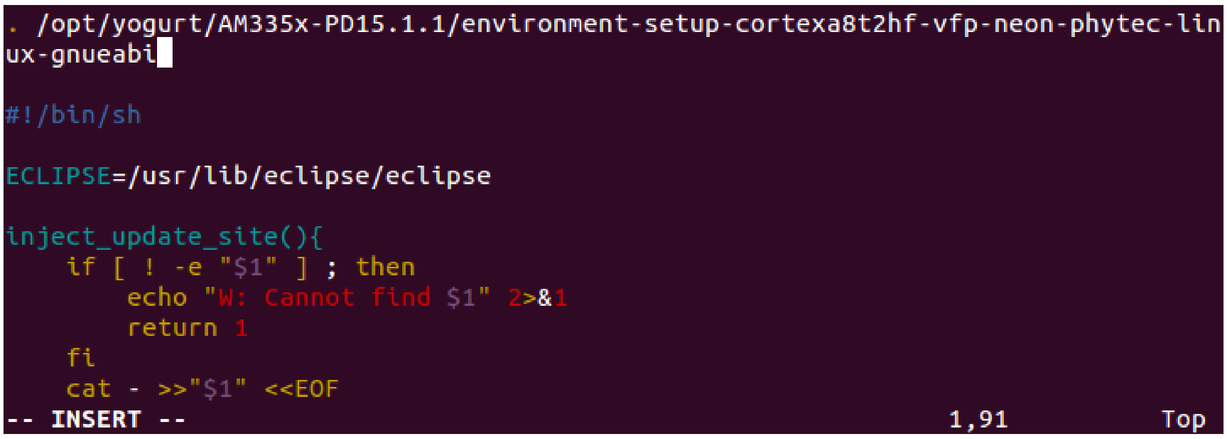
4.5.1.5 Setting up Eclipse and Integrate Plug-ins

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 virtual machine:
  `mkdir -p /opt/prj_workspace/Eclipse`

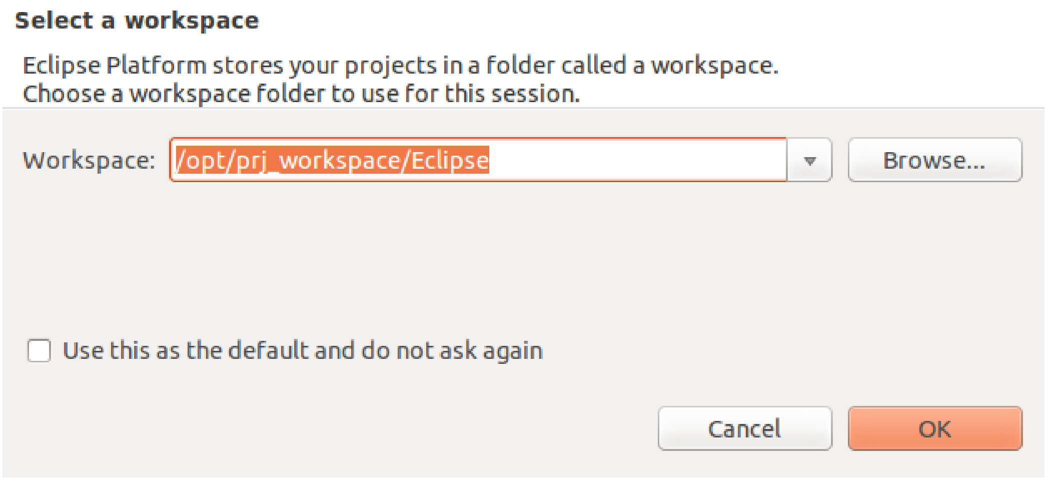
```
sudo chmod -R 777 /opt/prj_workspace/
```

- 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/AM335x-PD15.1.1/environment-setup-cortexa8t2hf-vfp-neon-phytec-linux-gnueabi
  ```



```
. /opt/yogurt/AM335x-PD15.1.1/environment-setup-cortexa8t2hf-vfp-neon-phytec-lin
ux-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
-- INSERT --                                                  1,91          Top
```

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

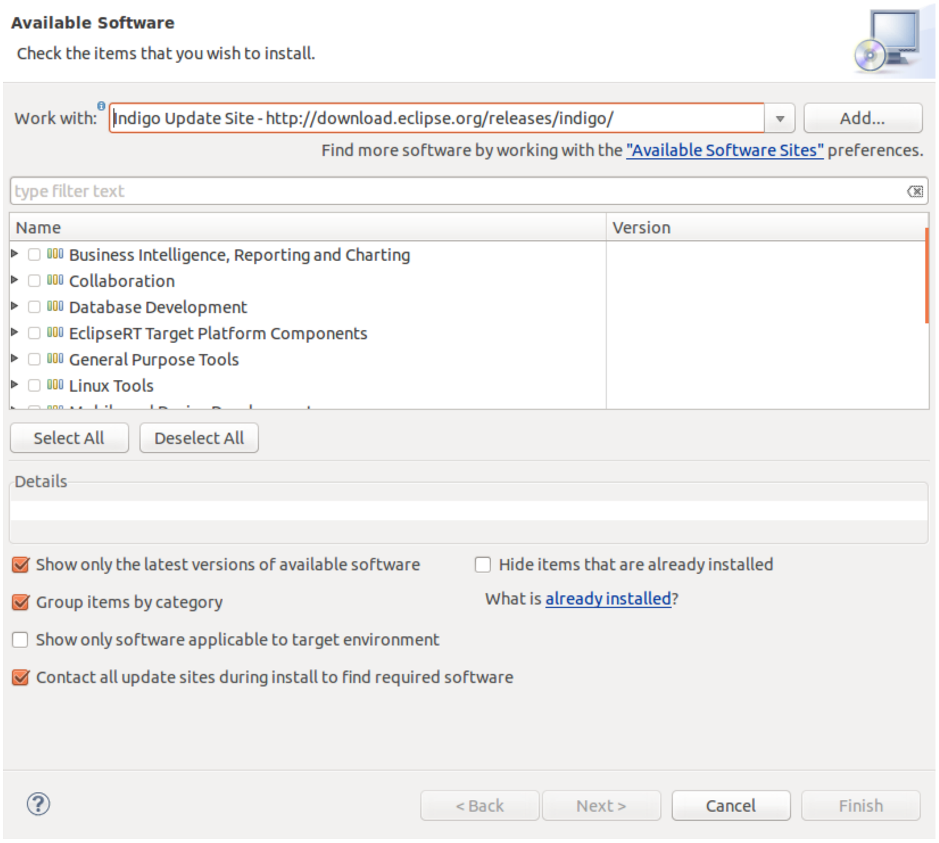
Workspace: /opt/prj_workspace/Eclipse

☐ Use this as the default and do not ask again

- 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 ***Indigo Update Site***.



Available Software

Check the items that you wish to install.

Work with: Indigo Update Site - http://download.eclipse.org/releases/indigo/

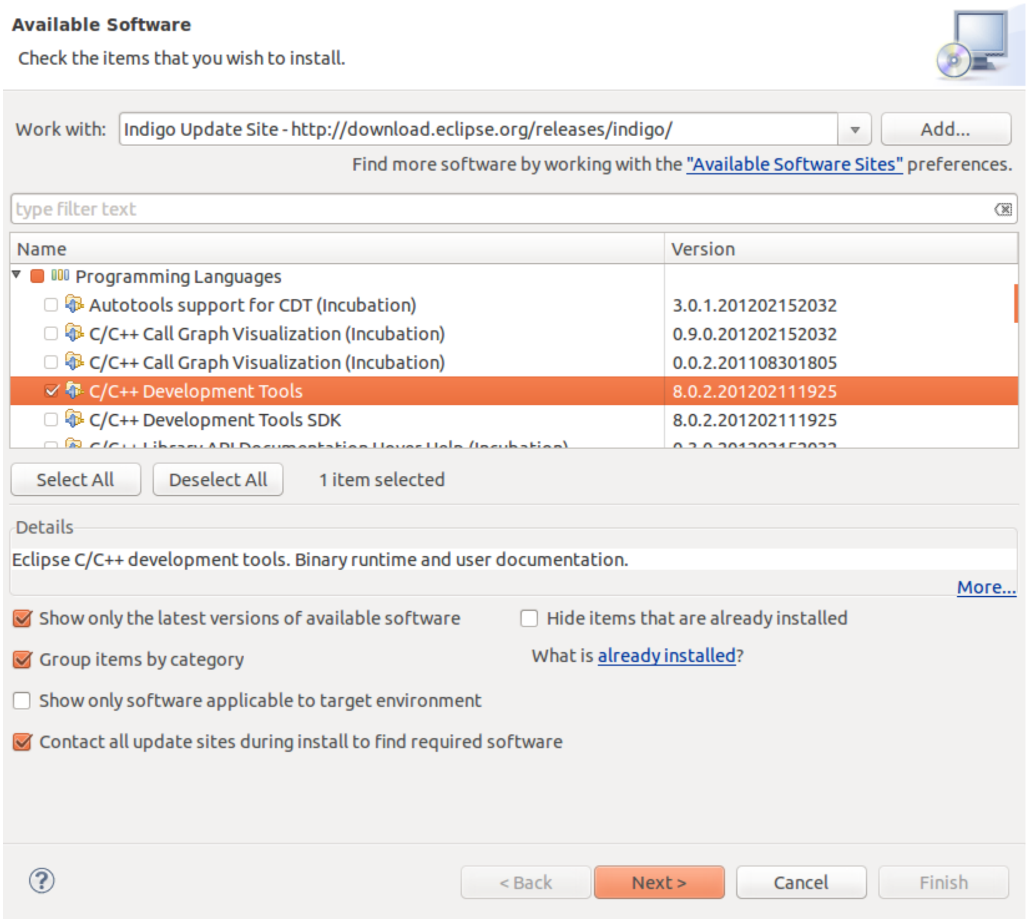
Find more software by working with the "Available Software Sites" preferences.

Name	Version
☐ Business Intelligence, Reporting and Charting	
☐ Collaboration	
☐ Database Development	
☐ EclipseRT Target Platform Components	
☐ General Purpose Tools	
☐ Linux Tools	

Details

☑ 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

- The available plug-ins appear in the selection area below now. Expand ***Programming Languages*** and check ***C/C++ Development Tools***. Click ***Next***.



Available Software

Check the items that you wish to install.

Work with: Indigo Update Site - http://download.eclipse.org/releases/indigo/

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

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

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

- 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`

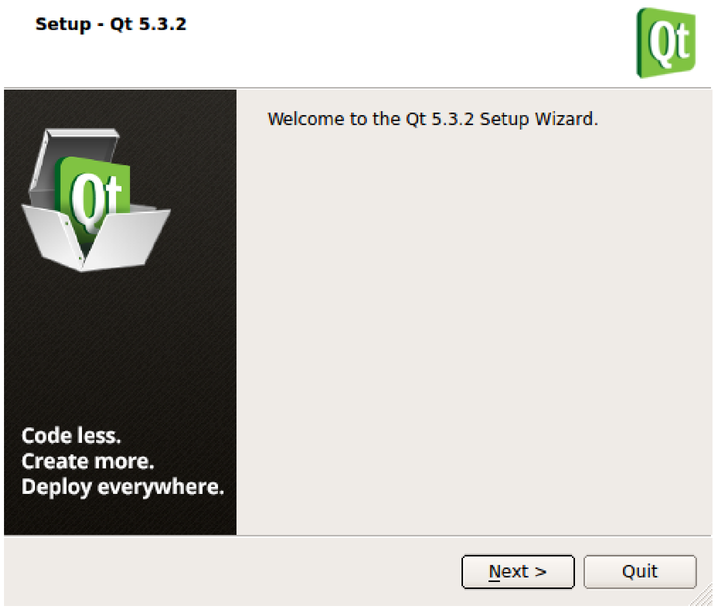
	Congratulations! You have successfully integrated the *CDT* plug-in in *Eclipse* and now you can start programming in *C/C++*. In the next section you will find a short introduction on how to install and setup the *Qt Creator*.

4.5.1.6 Install and Setup *Qt Creator*

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

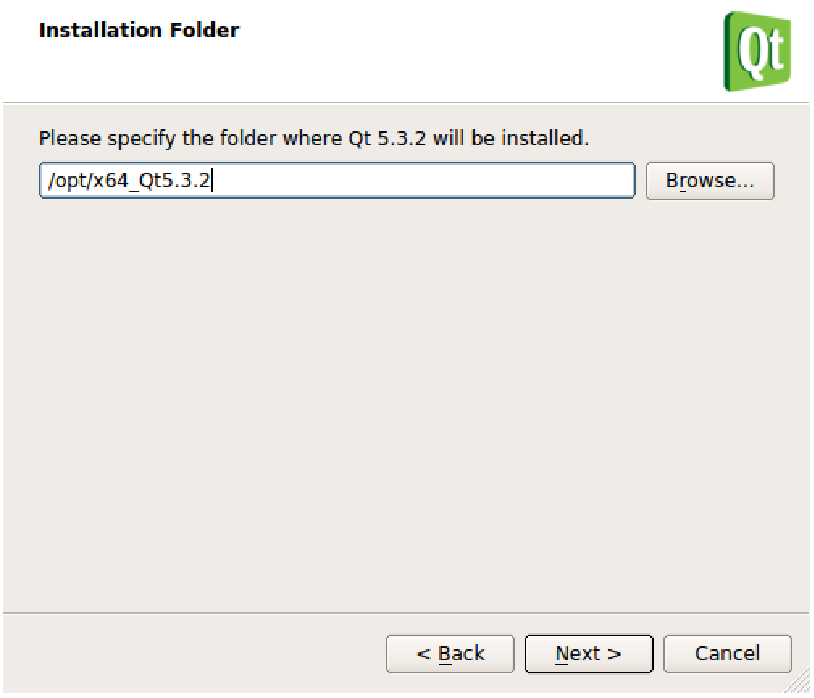
- Download *Qt Creator* with following command:
  `wget http://download.qt.io/official_releases/qt/5.3/5.3.2/qt-opensource-linux-x64-5.3.2.run`
- Set execute priviliges:
  sudo `chmod +x qt-opensource-linux-x64-5.3.2.run`
- Run the *QtCreator* installation routine
  `./qt-opensource-linux-x64-5.3.2.run`

This opens a window for the installation.


Setup - Qt 5.3.2

Welcome to the Qt 5.3.2 Setup Wizard.

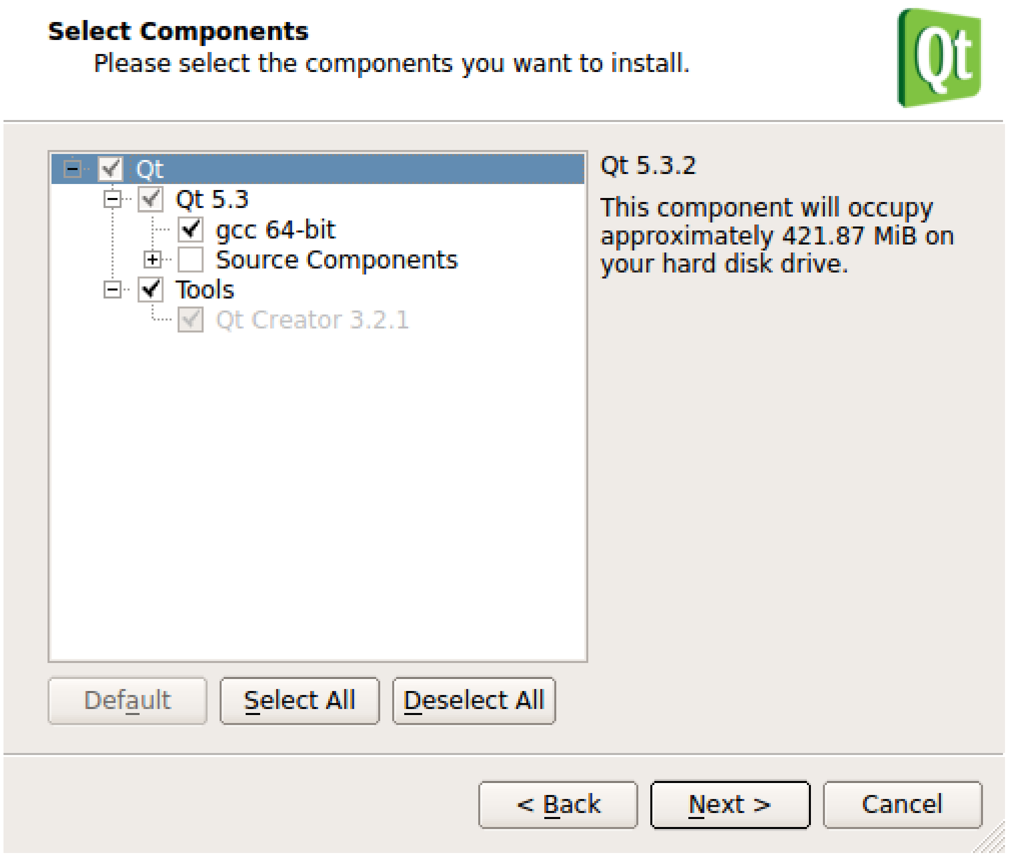
- Step trough the installation and if you are asked set following installation directory:
  `/opt/x64_Qt5.3.2`


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.

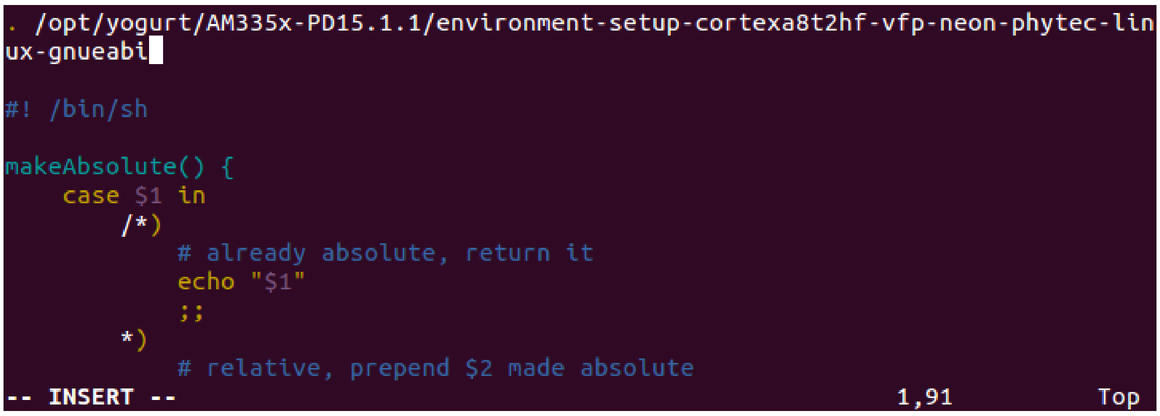
Qt 5.3.2

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 easy 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/AM335x-PD15.1.1/environment-setup-cortexa8t2hf-vfp-neon-phytec-linux-gnueabi`

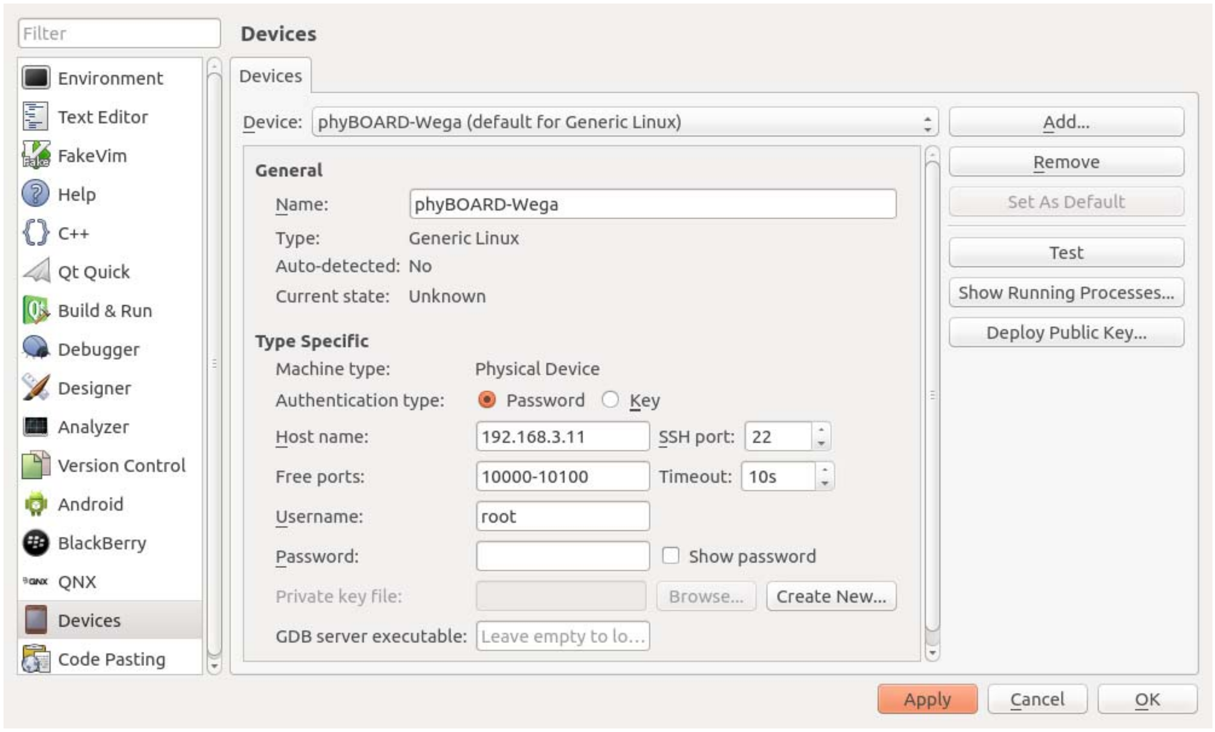


```
. /opt/yogurt/AM335x-PD15.1.1/environment-setup-cortexa8t2hf-vfp-neon-phytec-lin
ux-gnueabi

#! /bin/sh

makeAbsolute() {
    case $1 in
        /*)
            # already absolute, return it
            echo "$1"
            ;;
        *)
            # relative, prepend $2 made absolute
-- INSERT --                                                1,91          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-Wega as device with following inputs by opening
  Tools --> Options --> Devices --> Add--> Generic Linux Device:

Name:	phyBOARD-Wega
IP:	192.168.3.11
Username:	root

  Keep the *password* empty



Device: phyBOARD-Wega (default for Generic Linux)

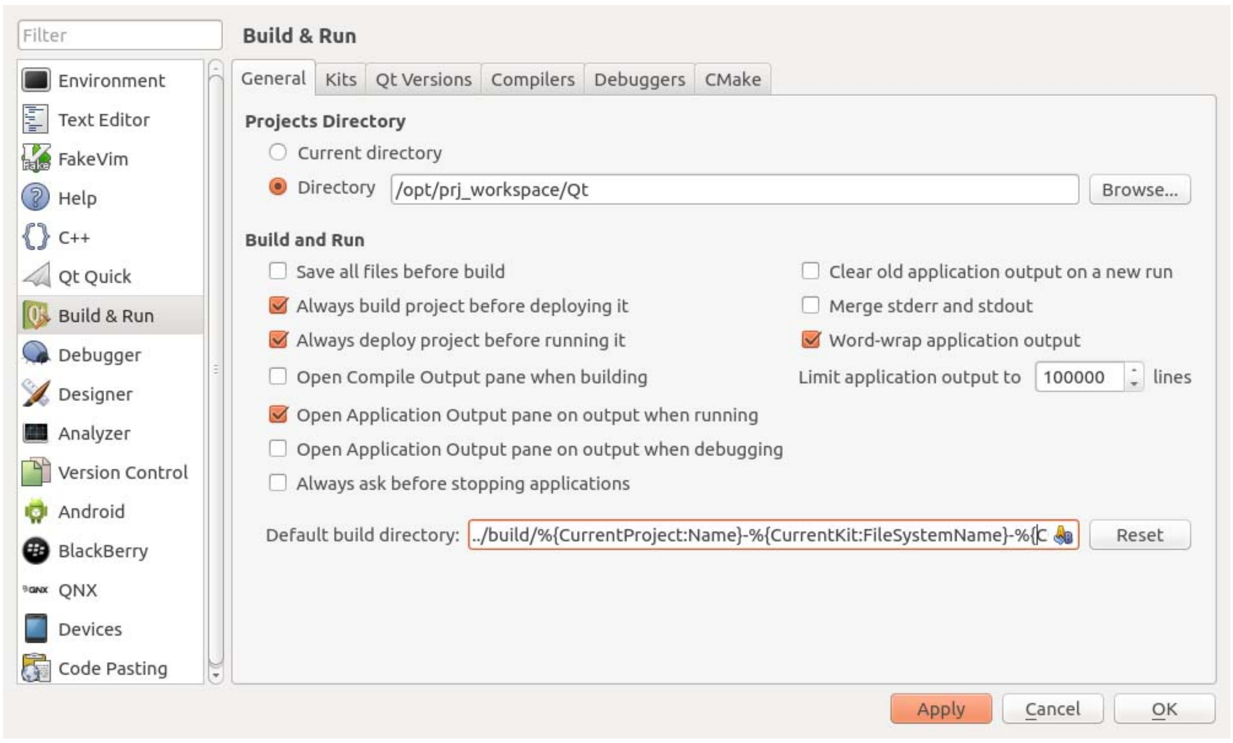
General

Name: phyBOARD-Wega
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 lo...

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



Filter

Environment, Text Editor, FakeVim, Help, C++, Qt Quick, Build & Run, Debugger, Designer, Analyzer, Version Control, Android, BlackBerry, QNX, Devices, Code Pasting

Build & Run

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

Projects Directory

○ Current directory
● Directory /opt/prj_workspace/Qt Browse...

Build and Run

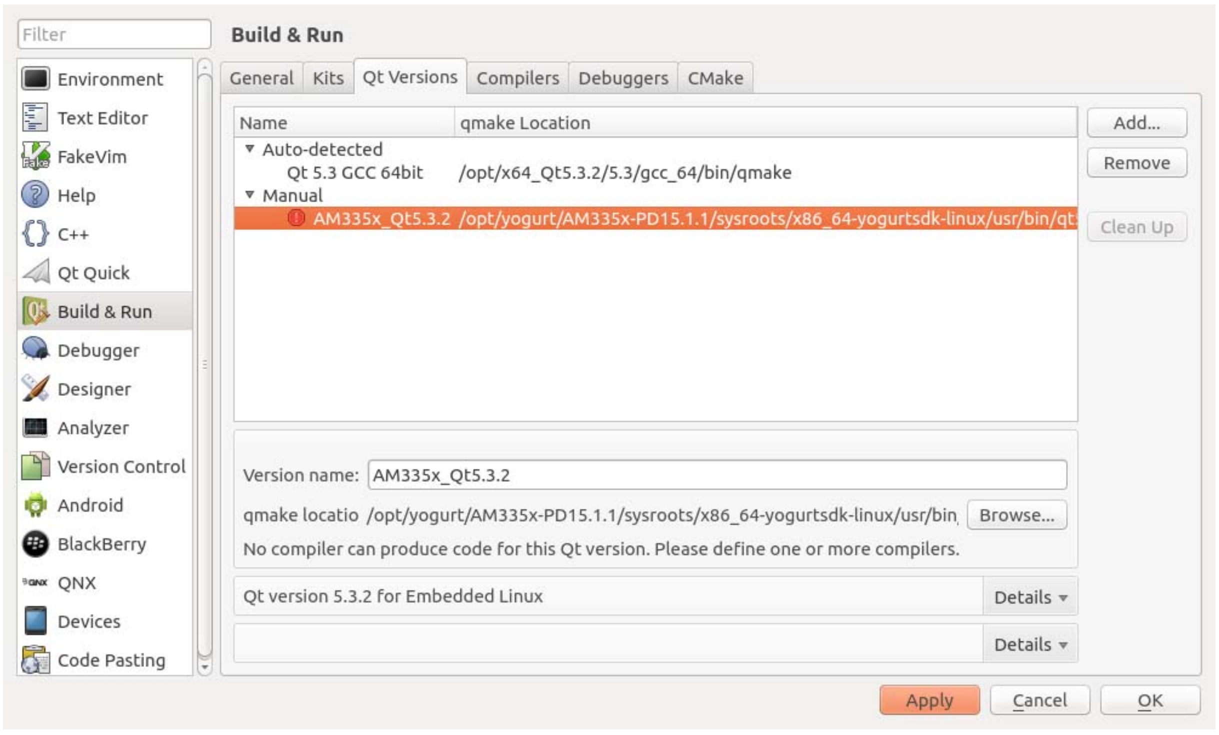
☐ Save all files before build
☑ Always build project before deploying it
☑ Always deploy project before running it
☐ Open Compile Output pane when building
☑ Open Application Output pane on output when running
☐ Open Application Output pane on output when debugging
☐ Always ask before stopping applications

☐ 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 Reset

Apply | Cancel | OK

- Select tab ***Qt Versions*** and click ***Add...*** to add a new one**.**
  `Name: AM335x_Qt5.3.2`
  `qmake Location: /opt/yogurt/AM335x-PD15.1.1/sysroots/x86_64-yogurtsdk-linux/usr/bin/qt5/qmake`



Filter

Environment, Text Editor, FakeVim, Help, C++, Qt Quick, Build & Run, Debugger, Designer, Analyzer, Version Control, Android, BlackBerry, QNX, Devices, Code Pasting

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	
AM335x_Qt5.3.2	/opt/yogurt/AM335x-PD15.1.1/sysroots/x86_64-yogurtsdk-linux/usr/bin/qt

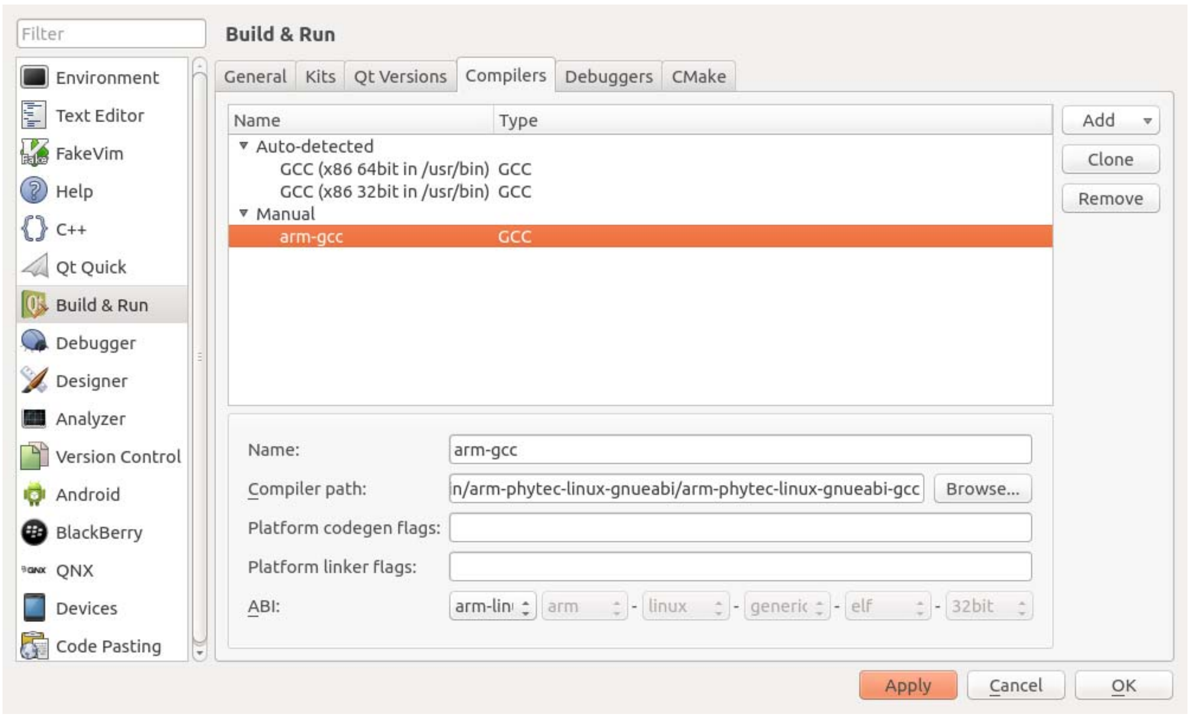
Add... | Remove | Clean Up

Version name: AM335x_Qt5.3.2
qmake locatio /opt/yogurt/AM335x-PD15.1.1/sysroots/x86_64-yogurtsdk-linux/usr/bin Browse...
No compiler can produce code for this Qt version. Please define one or more compilers.

Qt version 5.3.2 for Embedded Linux Details ▾
Details ▾

Apply | Cancel | OK

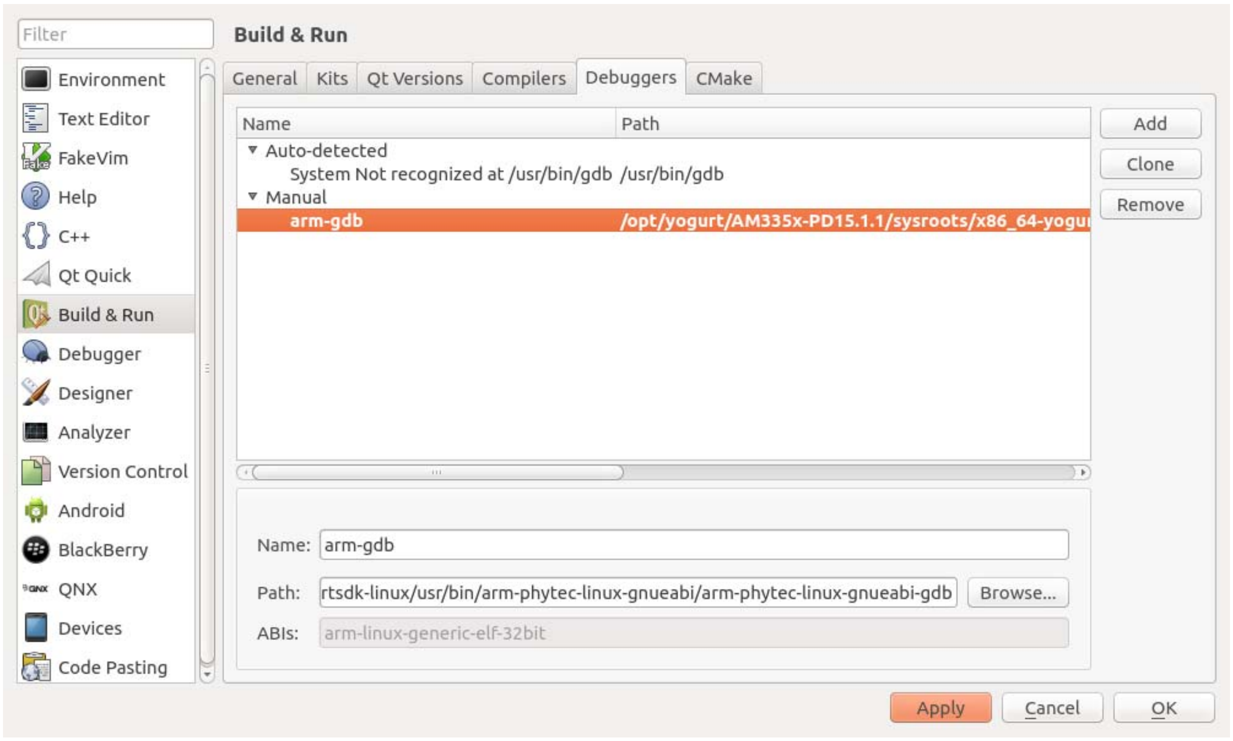
- Select tab ***Compilers*** and click on ***Add*** select ***GCC***
  `Name: arm-gcc`
  `Path:/opt/yogurt/AM335x-PD15.1.1/sysroots/x86_64-yogurtsdk-linux/usr/bin/arm-phytec-linux-gnueabi/arm-phytec-linux-gnueabi-gcc`



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

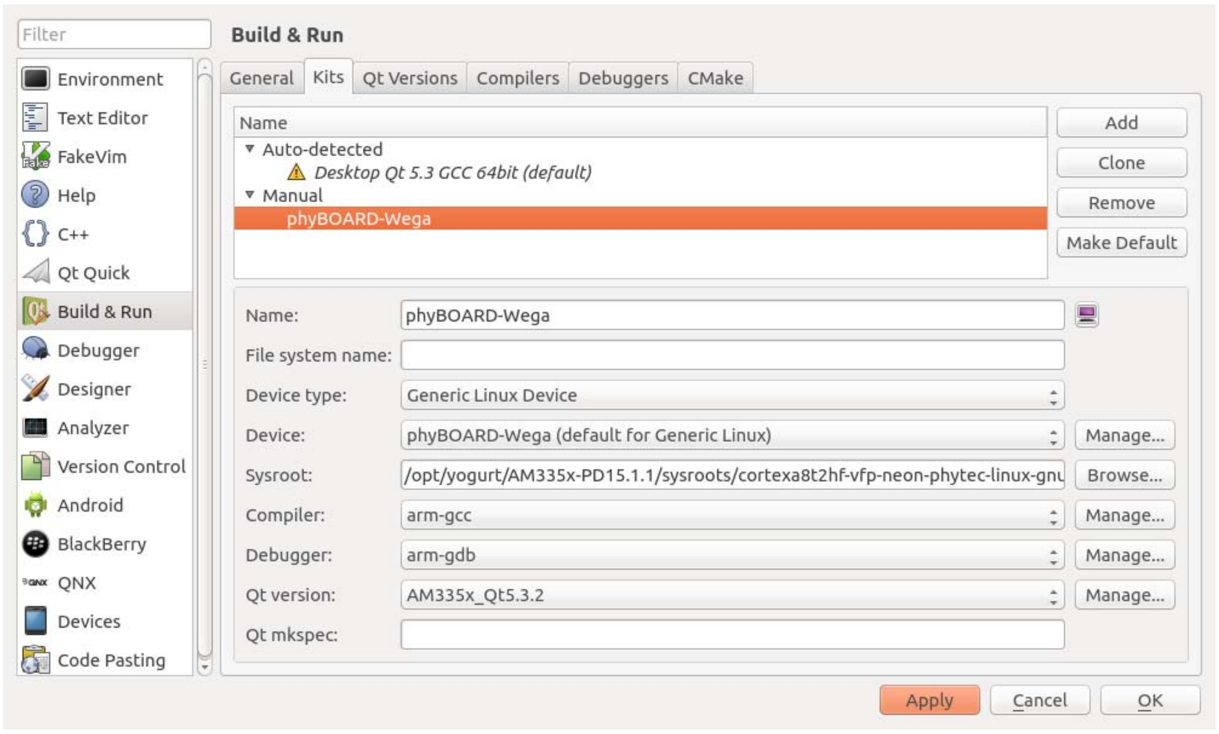
- Select tab ***Debuggers*** an click on ***Add***
  `Name: arm-gdb`
  `Path: :/opt/yogurt/AM335x-PD15.1.1/sysroots/x86_64-yogurtsdk-linux/usr/bin/arm-phytec-linux-gnueabi/arm-phytec-linux-gnueabi-gdb`



Name	Path
▾ Auto-detected	
System Not recognized at /usr/bin/gdb	/usr/bin/gdb
▾ Manual	
arm-gdb	/opt/yogurt/AM335x-PD15.1.1/sysroots/x86_64-yogu

Name: arm-gdb
Path: rtsdk-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-WEGA
Device Type: Generic Linux Device
Sysroot: /opt/yogurt/AM335x-PD15.1.1/sysroots/cortexa8t2hf-vfp-neon-
phytec-linux-gnueabi
Compiler: arm-gcc
Debugger: arm-gdb
Qt version: AM335x_Qt5.3.2
```



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

Name: phyBOARD-Wega
File system name:
Device type: Generic Linux Device
Device: phyBOARD-Wega (default for Generic Linux)
Sysroot: /opt/yogurt/AM335x-PD15.1.1/sysroots/cortexa8t2hf-vfp-neon-phytec-linux-gnu
Compiler: arm-gcc
Debugger: arm-gdb
Qt version: AM335x_Qt5.3.2
Qt mkspec:

- If you generate a new Project you can use this Kit to cross compile your application.

4.5.1.7 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-Wega AM335x 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 mandatory for a successful operation of this Application Guide. They only simplify the handling and the look of the system. Because of that the modifications are only listed without further 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 USB Connectivity (X15 and X42)

Numerous jumpers allow to configure the USB interfaces 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 USB configurations described below.

4.6.1.1 Combining the Overcurrent Signals (J78 and J77)

To save one GPIO of the controller for other purposes jumper J78 allows to connect the overcurrent signals of both USB interfaces USB0 and USB1 to one single GPIO (GPIO3_18). If the two overcurrent signals need to be evaluated separately, the OC signal of USB0 can be connected to GPIO3_19 and the OC signal of USB1 to GPIO3_18.

The following table shows the available configurations:

J78	Description
1+2	Separate OC signals for USB1 (nUSB1_OC_GPIO3_18) and USB0 (nUSB0_OC_GPIO3_19)
2+3	**One OC signal (nUSB1_OC_GPIO3_18)for both USB interfaces**

Table 13: USBOC Configuration

⚠	If J78 is set to 1+2, J77 also has to be set to 1+2.

4.6.1.2

4.6.1.3 Rerouting the USB Interfaces to different Connectors (J72 – J75, J79 and J80)

For later expansion boards one of the two USB interfaces can be routed to the expansion connector (X69). The following table shows all possible configurations.

Mode	J72	J73	J74	J75	J79	J80
USB1 at USB-A connector X15 and USB0 at USB-OTG connector X42	**1+2**	**1+2**	**1+2**	**1+2**	**nm**	**nm**
USB1, USB1_VBUS and X_USB1_ID at expansion connector X69[2,3] USB0 at USB-OTG connector X42	2+3	2+3	1+2	1+2	1+2[4]	1+2
USB0, USB0_VBUS and USB0_ID at expansion connector X69[2,3] USB1 at USB-A connector X15	1+2	1+2	2+3	2+3	2+3	2+3

Table 14: USB Routing Configuration

4.6.2 I^{2}C Connectivity

The I^{2}C interface of the AM335x is available at different connectors on the phyBOARD-Wega. The following table provides a list of the connectors and pins with I^{2}C connectivity.

Connector	Location
Expansion connector X69	pin 11 (X_I2C0_SDA); pin 13 (X_I2C0_SCL)
A/V connector X71	pin 16 (X_I2C0_SDA); pin 15 (X_I2C0_SCL)

Table 15: I^{2}C Connectivity

To avoid any conflicts when connecting external I^{2}C devices to the phyBOARD-Wega the addresses of the on-board I^{2}C devices must be considered. *Table 16* lists the addresses already in use. The table shows only the default address.

[2]: Caution! There is no protective circuit for the USB interfaces brought out at the expansion connector

[3]: Caution! The voltage level of the USB ID signal X_USB_ID_EXP, is 1.8 V. Steady state voltages above 2.1 V applied to this signal may damage the AM335x.

[4]: Note: The ID pin of USB1 is hardwired. To use the function at the expansion connector R399 must be removed, too.

Board	Prod. No.	Device	Address used (7 MSB)
phyCORE-AM335x	PCL-051	EEPROM	0x52
		RTC	0x68
		PMIC	0x2D, 0x12
phyBOARD-Wega	PBA-CD-02	Audio	0x18
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 16: I^2C Addresses in Use

4.6.2.1 Software Implementation

The driver for I^2C can be accessed by its API within the kernel. If you connect a chip to I^2C, you should implement its driver into the kernel, as well. 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^2C devices. It is also possible to access the I^2C 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*.

4.6.2.1.1 EEPROM

It is possible to read/write directly to the device:

```
    /sys/class/i2c-adapter/i2c-0/0-0052/eeprom
```

e.g.: Fill with zeros

```
    dd if=/dev/zero of=/sys/class/i2c-adapter/i2c-0/0-0052/eeprom
    bs=4096 count=1
```

4.6.3 Audio/Video Connectors (X70 and X71)

The Audio/Video (A/V) connectors X70 and 71 provide an easy way to add typical A/V functions and features to the phyBOARD-Wega. 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. The pinout of the A/V connectors is shown in *Table 17* and *Table 18.*

The A/V connector is intended for use with phyBOARD Expansion Boards[5], and to add specific audio/video connectivity with custom expansion boards.

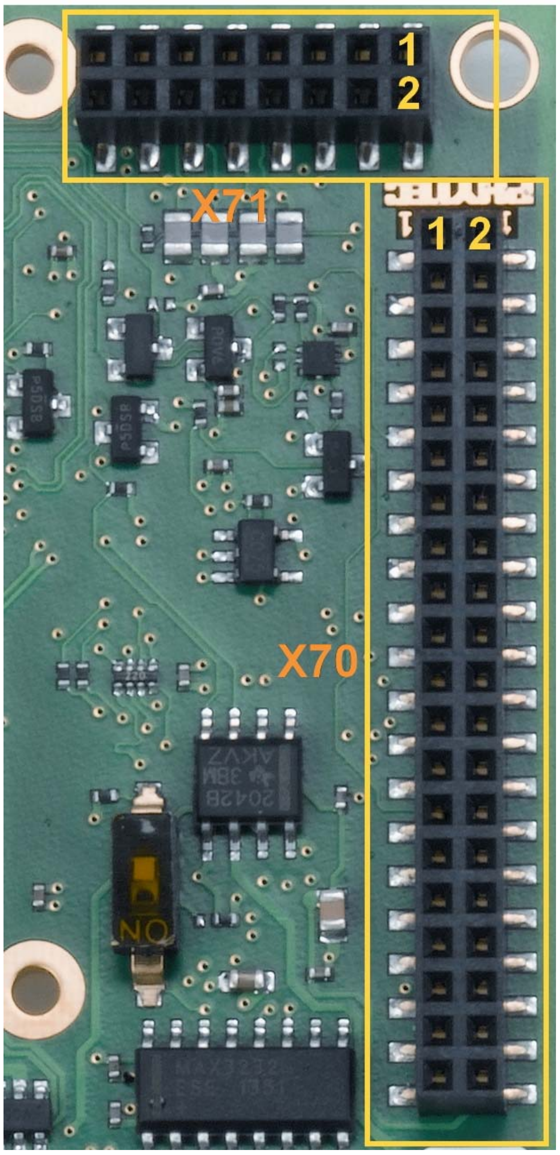


Figure 14: Audio/Video Connectors (X70 and X71)

[5]: Please find additional information on phyBOARD-Wega AM335x Expansion Boards in the corresponding application guide (L-793e).

Pin #	Signal Name	Type	SL	Description
1	GND	-	-	Ground
2	X_LCD_D21	OUT	3.3V	LCD D21
3	X_LCD_D18	OUT	3.3 V	LCD D18
4	X_LCD_D16	OUT	3.3 V	LCD D16
5	X_LCD_D0	OUT	3.3 V	LCD D0
6	GND	-	-	Ground
7	X_LCD_D1	OUT	3.3 V	LCD D1
8	X_LCD_D2	OUT	3.3 V	LCD D2
9	X_LCD_D3	OUT	3.3 V	LCD D3
10	X_LCD_D4	OUT	3.3 V	LCD D4
11	GND	-	-	Ground
12	X_LCD_D22	OUT	3.3 V	LCD D22
13	X_LCD_D19	OUT	3.3 V	LCD D19
14	X_LCD_D5	OUT	3.3 V	LCD D5
15	X_LCD_D6	OUT	3.3 V	LCD D6
16	GND	-	-	Ground
17	X_LCD_D7	OUT	3.3 V	LCD D7
18	X_LCD_D8	OUT	3.3 V	LCD D8
19	X_LCD_D9	OUT	3.3 V	LCD D9
20	X_LCD_D10	OUT	3.3 V	LCD D10
21	GND	-	-	Ground
22	X_LCD_D23	OUT	3.3 V	LCD D23
23	X_LCD_D20	OUT	3.3 V	LCD D20
24	X_LCD_D17	OUT	3.3 V	LCD D17
25	X_LCD_D11	OUT	3.3 V	LCD D11
26	GND	-	-	Ground
27	X_LCD_D12	OUT	3.3 V	LCD D12
28	X_LCD_D13	OUT	3.3 V	LCD D13
29	X_LCD_D14	OUT	3.3 V	LCD D14
30	X_LCD_D15	OUT	3.3 V	LCD D15
31	GND	-	-	Ground
32	X_LCD_PCLK	OUT	3.3 V	LCD Pixel Clock
33	X_LCD_BIAS_EN	OUT	3.3 V	LCD BIAS
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[6]
40	VCC5V	OUT	5.0 V	5 V power supply

Table 17: Pin Assignment of PHYTEC A/V connector X70

Pin #	Signal Name	Type	SL	Description
1	X_I2S_CLK	I/O	3.3 V	I²S Clock
2	X_I2S_FRM	I/O	3.3 V	I²S Frame
3	X_I2S_ADC	I/O	3.3 V	I²S Analog-Digital converter (microphone)
4	X_I2S_DAC	I/O	3.3 V	I²S Digital-Analog converter (speaker)
5	X_AV_INT_GPIO1_30	I/O	3.3 V	A/V interrupt; GPIO1_30
6	nUSB0_OC_GPIO3_19 or X_MCASP0_AHCLKX_GPIO3_21	I/O	3.3 V	GPIO3_19 or McASP0 high frequency clock (see below)
7	GND	-	-	Ground
8	nRESET_OUT	OUT	3.3 V	Reset
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_I2C0_SCL	I/O	3.3 V	I²C Clock
16	X_I2C0_SDA	I/O	3.3 V	I²C Data

Table 18: Pin Assignment of PHYTEC A/V connector X71

Jumper J77 connects either signal X_MCASP0_AHCLKX_GPIO3_21 or signal nUSB1_OC_GPIO3_19 to pin 6 of X71. The following table shows the available configurations:

J77	Description
1+2	X_MCASP0_AHCLKX_GPIO3_21
2+3	**nUSB1_OC_GPIO3_19**

Table 19: A/V Jumper Configuration J77

[6] Voltage level is not specified and depends on the connected power module and the voltage attached.

⚠	If J77 is set to 2+3, J78 also has to be set to 2+3.

4.6.3.1 Software Implementation

4.6.3.1.1 Framebuffer

The framebuffer 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.

You can check your framebuffer resolution with the command:

```
fbset
```

⚠	If the device /dev/fb0 is not available, verify that the BSP version matches the hardware version of the phyBOARD-Wega **and** the display adapter connected. Please visit http://www.phytec.eu and navigate to “Support” / “FAQ/Download” / “phyBOARDs – Single Board Computer” / “phyBOARD-Wega” and see FAQ “Choosing the right Linux-BSP version” for more information.

4.6.3.1.2 Brightness

The PWM signal at pin 38 allows to change the brightness of the display attached to the A/V connector.

If you use the you can change the brightness as follows:

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.

4.6.3.1.3 Touch

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.

4.6.3.1.4 I^{2}C Connectivity

Please refer to *section 4.6.2* for complete information on the I^{2}C connectivity.

4.6.3.1.5 Audio I^{2}S

Audio support on the module is done via the I^{2}S interface and controlled via I^{2}C.

On the phyBOARD-Wega the audio codec's registers can be accessed via the I2C0 interface at address 0x18 (7-bit MSB addressing).

4.6.3.1.6 User programmable GPIOs

Two pins of the A/V connector X71 are dedicated as GPIO (*Table 18*). These signals are also available/used on the corresponding expansion-boards, e.g. PEB-AV-02. For more information please look at the Expansion Boards Application Guide (L-793e).

⚠	The BSP delivered with the phyBOARD-Wega supports the GPIOs according to the configuration done in correspondence to the expansion board installed on delivery. Thus the GPIOs might not be available if they are needed to support functions of the expansion board. In order to apply the GPIOs for other purposes after removal of the expansion board the BSP must be exchanged, too. This allows to easily adapt the BSP if an expansion board is attached, removed, or exchanged, thus allowing to release the GPIOs for other purposes.

With the appropriate BSP / BSP configuration the GPIOs (GPIO1_30, GPIO3_19, or GPIO3_21) are available as input by default.

The following table lists all GPIOs, their location, their number and their default usage.

Pin #	GPIO Name	GPIOno	Default Usage	Comment
5	GPIO1_30	62	IN	
6	GPIO3_19 or GPIO3_21	115 or 117	IN	selectable with jumper J77

Table 20: GPIOs available at A/V Connector X71

Please refer to chapter *4.6.4.1.5* for information on how to use the GPIO pins.

4.6.4 Expansion Connector (X69)

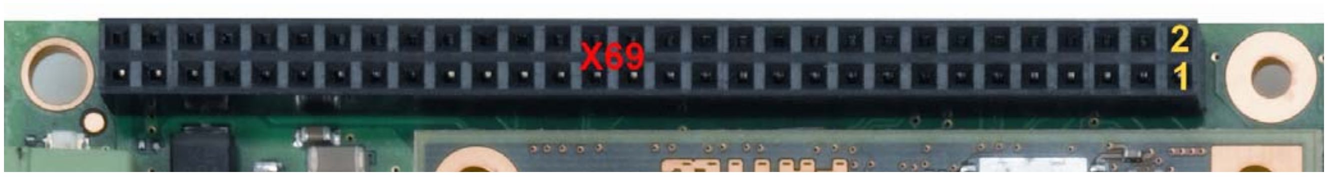


Figure 15: Expansion Connector (X69)

The expansion connector X69 provides an easy way to add other functions and features to the phyBOARD-Wega. 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-Wega AM335x Expansion Boards[7], and to add specific functions with custom expansion boards

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

[7]: Please find additional information on phyBOARD-Wega AM335x Expansion Boards in the corresponding application guide (L-793e).

Pin #	Signal Name	Type	SL	Description
1	VCC3V3	OUT	3.3 V	3.3 V power supply
2	VCC5V	OUT	5.0 V	5 V power supply
3	VDIG1_1P8V	OUT	1.8 V	1.8 V power supply (max. 300 mA)
4	GND	-	-	Ground
5	X_SPI0_CS0	OUT	3.3 V	SPI 0 chip select 0
6	X_SPI0_MOSI	OUT	3.3 V	SPI 0 master output/slave input
7	X_SPI0_MISO	IN	3.3 V	SPI 0 master input/slave output
8	X_SPI0_CLK	OUT	3.3 V	SPI 0 clock output
9	GND	-	-	Ground
10	X_UART0_RXD	IN	3.3 V	UART 0 receive data (standard debug interface)
11	X_I2C0_SDA	I/O	3.3 V	I2C0 Data
12	X_UART0_TXD	OUT	3.3 V	UART 0 transmit data (standard debug interface)
13	X_I2C0_SCL	I/O	3.3 V	I2C0 Clock
14	GND	-	-	Ground
15	X_JTAG_TMS	IN	3.3 V	JTAG Chain Test Mode Select signal
16	X_nJTAG_TRST	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_USB_DP_EXP	I/O	3.3 V	USB data plus (for USB0 or USB1)[9; 10]
22	X_USB_DM_EXP	I/O	3.3 V	USB data minus (for USB0 or USB1)[9; 10]
23	nRESET_OUT	OUT	3.3 V	Reset
24	GND	-	-	Ground
25	X_MMC2_CMD	I/O	3.3 V	MMC command
26	X_MMC2_DAT0	I/O	3.3 V	MMC data 0
27	X_MMC2_CLK	I/O	3.3 V	MMC clock
28	X_MMC2_DAT1	I/O	3.3 V	MMC data 1
29	GND	-	-	Ground
30	X_MMC2_DAT2	I/O	3.3 V	MMC data 2
31	X_UART2_RX_GPIO3_9	I/O	3.3 V	UART 2 receive data; GPIO3_9[8]
32	X_MMC2_DAT3	I/O	3.3 V	MMC data 3
33	X_UART2_TX_GPIO3_10	I/O	3.3 V	UART 2 transmit data; GPIO3_10[8]
34	GND	-	-	Ground

35	X_UART3_RX_GPIO2_18	I/O	3.3 V	UART 3 receive data; GPIO2_18[8]
36	X_UART3_TX_GPIO2_19	I/O	3.3 V	UART 3 transmit data; GPIO2_19[8]
37	X_INTR1_GPIO0_20	I/O	3.3 V	Interrupt 1; GPIO0_20
38	X_GPIO0_7	I/O	3.3 V	GPIO0_7
39	X_AM335_EXT_WAKEUP	IN	3.3 V	External wakeup
40	X_INT_RTCn	OUT	3.3 V	Interrupt from the RTC
41	GND	-	-	Ground
42	X_GPIO3_7_nPMOD_PWRFAIL	I/O	3.3 V	GPIO3_7; **Caution!** Also connected to power fail signal through R415.
43	nRESET_IN	IN	3.3 V	Push-button reset
44	X_GPIO1_31	I/O	3.3 V	GPIO1_31
45	X_AM335_NMIn	IN	3.3 V	AM335x non-maskable interrupt
46	GND	-	-	Ground
47	X_AIN4	IN	1.8 V	Analog input 4
48	X_AIN5	IN	1.8 V	Analog input 5
49	X_AIN6	IN	1.8 V	Analog input 6
50	X_AIN7	IN	1.8 V	Analog input 7
51	GND	-	-	Ground
52	X_GPIO_CKSYNC	I/O	3.3 V	GPIO Clock Synchronization
53	X_USB_ID_EXP	IN	1.8 V	USB port identification (for USB0 or USB1)[9]
54	USB_VBUS_EXP	OUT	5.0 V	USB bus voltage (for USB0 or USB1) [9][10]
55	X_USB1_CE	OUT	3.3 V	USB 1 charger enable
56	GND	-	-	Ground
57	VCC_BL	OUT	NS	Backlight power supply[11]
58	X_PB_POWER	IN	5.0 V	Power On for Power Management IC for AM335x
59	GND	-	-	Ground
60	VCC5V_IN	IN	5.0 V	5 V input supply voltage

Table 21: Pin Assignment of PHYTEC Expansion Connector X69

[8]: These pins are configured as GPIO pins. To use them as UART interface the pin muxing must be changed and additional software development is required.
[9]: Jumpers J72 – J75, J79 and J80 allow to configure the USB interface at the expansion connector (*Table 14*).
[10] : **Caution!** There is no protective circuit for the USB interface brought out at the expansion connector X69.
[11]: Voltage level is not specified and depends on the connected power module and the voltage attached.

⚠	If the SPI-NOR Flash on the phyCORE-AM335x is populated, the SPI signals on the expansion port can not be used.

4.6.4.1 Software Implementation

4.6.4.1.1 UART Connectivity

Only */dev/tty00* for UART0 and */dev/tty01* for UART1 (at X66) have been implemented within the BSP.

The standard console UART0 is accessible as */dev/tty00* and is mainly used for debugging and control of software updates.

Usage of */dev/tty02* for UART2, or */dev/tty03* for UART3 requires changing of the pin muxing and additional software development.

4.6.4.1.2 USB Connectivity

Depending on the configuration of jumpers J72 – JJ75, J79 and J80 either USB0, or USB1 can be connected to the expansion connector X69 (*section 4.6.1.2*). Both USB interfaces can be used as USB host as well as USB OTG interface. Please refer to *section 3.2.4.1* for information about the software implementation.

⚠	**Caution!** There is no protective circuit for the USB interface brought out at the expansion connector X69.

⚠	**Caution!** The voltage level of the USB ID signal X_USB_ID_EXP, is 1.8 V. Steady state voltages above 2.1 V applied to this signal may damage the AM335x.

4.6.4.1.3 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*.

4.6.4.1.4 I^2C Connectivity

Please refer to *section 4.6.2* for complete information on I^2C connectivity.

4.6.4.1.5 User programmable GPIOs

Eight pins of the expansion connector are dedicated as GPIO (*Table 21*). These signals are also available on the corresponding expansion-boards, e.g. PEB-EVAL-01. For more information please look at the Expansion Boards Application Guide (L-793e).

⚠	The BSP delivered with the phyBOARD-Wega supports the GPIOs according to the configuration done in correspondence to the expansion board installed on delivery. Thus the GPIOs might not be available if they are needed to support functions of the expansion board. In order to apply the GPIOs for other purposes after removal of the expansion board the BSP must be exchanged, too. This allows to easily adapt the BSP if an expansion board is attached, removed, or exchanged, thus allowing to release the GPIOs for other purposes.

With the appropriate BSP / BSP configuration the GPIOs (GPIO0_20, GPIO0_7, GPIO1_31, GPIO3_7 and GPIO3_10) are available as input by default.

The following table lists all GPIOs, their location, their number and default usage.

Pin #	GPIO Name	GPIOno	Default Usage
31	GPIO3_9	105	LED3 out
33	GPIO3_10	106	IN
35	GPIO2_18	82	LED1 out
36	GPIO2_19	83	LED2 out
37	GPIO0_20	20	IN
38	GPIO0_7	7	IN
42	GPIO3_7	103	IN
44	GPIO1_31	63	IN

Table 22: GPIOs available at Expansion Connector X69

To use the GPIOs switch to the appropriate directory by entering

```
cd /sys/class/gpio
```

Then you must request the GPIO from the kernel and export the control to userspace by typing

```
echo GPIOno > export
```

Now you can read the values with
`cat` *`GPIOno`*`/value`

Example:
```
cd /sys/class/gpio
echo 106 > export
cat gpio106/value
```

will read the value at pin 33 (GPIO3_10) at the expansion connector. To get the next values, just retype
`cat gpio106/value`

In order to use the GPIOs not used by the LED driver as output you can change the direction after you have requested the GPIO by entering
`echo out >` *`GPIOno`*`/direction`

Now you can write to the GPIO with
`echo 1 >` *`GPIOno`*`/value` and
`echo 0 >` *`GPIOno`*`/value`

Example:
```
cd /sys/class/gpio
echo 63 > export
echo out > 63/direction
echo 1 > 63/value
```

will result in a high level at pin 44 (GPIO1_31) at the expansion connector.

A different procedure is necessary for the GPIOs already requested by the LED driver (GPIO2_18, GPIO2_19 and GPIO3_9). For setting and resetting these LED outputs on the expansion connector switch to the appropriate directory listed in the following table:

Pin #	Signal Name	Path
31	GPIO3_9	`/sys/class/leds/user_led_green`
35	GPIO2_18	`/sys/class/leds/user_led_red`
36	GPIO2_19	`/sys/class/leds/user_led_yellow`

Table 23: GPO Pins and Device Path

and enter:
`echo 255 > brightness` for a high level at the pin, and
`echo 0 > brightness` for a low level at the pin

5 Revision History

Date	Version #	Changes in this manual
25.11.2014	Manual L-792e_0	Preliminary edition. Describes the phyBOARD-Wega AM335x SOM (PCB 1397.0) with phyBOARD-Wega- Carrier Board (PCB 1405.0).
12.08.2015	Manual L-792e_1	First edition. Describes the phyBOARD-Wega AM335x SOM (PCB 1397.0) with phyBOARD-Wega- Carrier Board (PCB 1405.1).

Index

Document: **phyBOARD-Wega AM335x**
Document number: **L-792e_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 2014

Ordering No. L-792e_1
Printed in Germany