



emBRICK[®] - EPC

BeagleboneBrick Starterkit-1
Local Bus / Beaglebone Black

Starter Kit - Local-Bus

Rev. 7

emBRICK® is developed and supported by



IMACS GmbH

Alfred-Nobel-Straße 2

D – 55411 Bingen am Rhein

www.imacs-gmbh.com

www.embrick.de

support@embrick.de

Hotline: +49 (0) 7154 80 83 - 15

IMACS GmbH reserves the right to make changes without further notice to any products herein. IMACS GmbH makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does IMACS GmbH assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters which may be provided in IMACS GmbH data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals” must be validated for each customer application by customer’s technical experts. IMACS GmbH does not convey any license under its patent rights nor the rights of others.

copyright © IMACS GmbH 2017. All rights reserved.

Reproduction, in part or whole, without the prior written consent of IMACS GmbH is prohibited.

Content

1. The emBRICK® Mission	4
1.1 Basic Characteristics	4
1.2 Typical Applications.....	4
1.3 Programming, Host Plattform, Connectivity	4
2. Introduction.....	5
2.1 About this Manual.....	5
2.2 References / Manual Overview.....	5
2.3 Before you start.....	6
2.4 Package contents	6
2.5 Additionally required components	6
2.6 The Hardware.....	7
2.6.1 Communication structure.....	7
2.7 The Software.....	8
3. Mounting and wiring.....	9
4. Hands on Software – C Programming on Debian	10
4.1 Preparation.....	10
4.1.1 Install Debian	10
4.1.2 Configure the Keyboard	11
4.1.3 Download the StarterKit Software	12
4.2 Setup the Development environment.....	13
4.2.1 Install the C-Programming IDE for Beaglebone Black.....	13
4.2.2 Enable SPI.....	13
4.3 Unzip the Z-BeagleboneBrick_VX.XX file.....	14
4.4 Load and compile the Sample Application	15
4.5 Start and explore the Functionality of "Appl"	17
4.6 Create your own application	18
4.6.1 Example.....	18
4.6.2 IO-Assignment	18
4.6.3 IO-Functions.....	19
4.7 Exterded Example	20
5. Hands on Software with radCASE.....	22
5.1 Installing Debian for radCASE	22
5.2 Install the radCASE software.....	22
5.2.1 Cross compiler for radCASE	22
5.2.2 Install radCASE	23
5.2.3 Board Support Package	23
5.2.4 Executing the radCASE project and Visualization	23
6. Extendend Informations for brickBUS® Stack	25
7. Appendix	26
7.1 Debian installation guide	26
7.2 Serial Port.....	27
7.2.1 Activate the serial port.....	27
7.2.2 Testing the serial port	27
7.3 GPIOs über die Kommandozeile steuern.....	27
7.4 I2C.....	27

7.5 Starting a program at boot time	27
7.6 WLAN.....	27
7.7 Bluetooth.....	27
7.8 E-Mail.....	27
7.9 Webserver.....	27
7.9.1 Apache-Webserver installieren.....	27
7.9.2 Common Gate Interface (CGI) und Python.....	27
7.10 Configure static IP address	27
7.11 Using the Beaglebone Black as a TCP-RS232-Bridge.....	29
7.11.1 Software and Configuration	29
7.11.2 Test the Connection.....	29
7.11.3 Deactivate checksum.....	29

1. The emBRICK® Mission

The idea of *emBRICK®* is to create **individual** electronic **controlsystems** by **assembling** (patching) small embedded **boards** (bricks) together via a simple interface (*brickBUS®*). Therefore we call this class of controllers simple **EPC** = Embedded PatchboardController.

emBRICK® combines in a perfect way the **cost-efficient** and **tailored** characteristics of a dedicated embedded system with the **ready to use** and **flexibility** of a PLC system.

To ensure a high acceptance, it is an **open** and **free** system. Besides buying existing devices everyone can develop his own components, to realize easily his individually tailored, cost-efficient and industrial-suited measure and control system.

For more information and products visit www.embrick.de or see the detailed manuals.

1.1 Basic Characteristics

- **free** - also for commercial use in own appliances (for pure EMS with a small licence fee)
- **open** - supplying reference schematics, protocol source code, samples and starterkits
- **third...half price** compared to common control systems (complete system view)
- scaleable local and remote topologies, **1...>1000 I/Os**, up to **1ms update**, deterministic
- **low own power** consumption, average 50mW/slave module in operation (outputs inactive)
- global and sector specific modules for **direct connection** of various **sensors and actors**
- using **common, low cost** standard μ Cs / **components**
- **easy installation**, no configuration necessary, simple plug modules together and use
- works with / programmable by **various established**, well known **platforms / languages**

1.2 Typical Applications

- Small, medium and largesize **measure and control systems**
- **Sectoral purpose**, with direct sensor/actor interface
- **Autonomous single box** control solutions i.e. with HMI and communication interfaces
- **Rapid hardware prototyping** system for control and measuring applications
- **PLC replacement**(i.e. with a Soft-PLC, IPC or an Embedded controller)
- Medium and large size **distributed IO-systems** (i.e. building automation)
- Physical front-end for **Internet of things** (IOT)

For more details see *Product_Catalogue* and *Application_Manual*.

1.3 Programming, Host Platforms, Connectivity

emBRICK® can be adapted to every platform with almost every footprint/performance. For master units, implementations are currently available for/with the following systems:

Computer platforms..... PC, Embedded-PC, Modul-PC, Raspberry Pi, Beaglebone Black

μ Controller platforms ARM-Ax, ARM-Cortex-Mx, Microchip PIC24/PIC24

OS/RTOS..... Windows, Linux, FreeRTOS, proprietary

Programming languages..... C,C++, IEC61131, UML (with implementing expansion)

Develop. environments MSVC,radCASE, Enterprise Architect, CODESYS, Cocox, MPLab, Geany (Raspberry Pi, Beaglebone Black), every other C/C++ IDE

Host Interfaces..... Ethernet, WLAN, CAN, RS232, RS485, others planed

For slave modules, actual Microchip PIC16/24 is used. Others (i.e. Cortex-M0) are planed.

2. Introduction

2.1 About this Manual

This manual leads step by step from the hardware mousing and software installation, to start up the emBRICK® adapter starterkit, running the delivered sample application and create your own applications.

Furtermore it is used as an open reference plattform for the brickBUS® local-master protocol stack.

2.2 References / Manual Overview

For *emBRICK®* and *brickBUS®* the following documents are available. Before reading this document it is recommended to read them in the given order:

- [System Manual](#) (*embrick_System-Manual_#.pdf*) ... the basic system manual that contains the idea, the intention and the basic technical concept of *emBRICK®/ brickBUS®* like mechanics, electronics and communication protocol. It includes the glossary for all other documents.
- [Application Examples](#)..... (*emBRICK_Application-Examples_#.pdf*) ... overview of typical *emBRICK®* device configurations and sample constellations for different industrial applications. It gives an idea how to use *emBRICK®* as an alternative to a normal PLC or an individual PCB / embedded system.
- [Product Catalogue](#) (*emBRICK_Product-Catalogue_#.pdf*) ... contains the overviews and detailed datasheets of all IMACS-available *emBRICK®* components and products. This includes electrical and mechanical characteristics, terminal assignment and notes about their usage.
- [Programmers Manual](#) (*emBRICK_Programmers-Manual_#.pdf*) ... is the manual for application software programmers when using established programming systems like Embedded-IDEs, Soft-PLCs, CASE-Tools but also native C/C++-coding.
- [FAQ Manual](#) (*emBRICK_FAQ-Manual_#.pdf*) ... contains answers to the most frequently asked questions about *emBRICK®* and its usage.
- Developers Manual..... is the manual for system developers, who like to create their own slave modules or master adaptations. It includes all technical details specifications of *brickBUS®* and also sample schematics and code samples of the software stacks. This document is only available on request from IMACS GmbH and needs the agreement on the *emBRICK®* free license conditions. Please contact support@embrick.de.

2.3 Before you start

- Check if you have received all starterkit parts, see [Package contents](#)
- Check if you need anything else which doesn't come with the starterkit in [Additionally required components](#)
- Follow the instructions in [Mounting and wiring](#) for starterkit assembly
- If you don't know what software to download read [The Software](#)

2.4 Package contents

The BeagleboneBrick Starterkit-1 contains:

- Adapter board (Z-BeagleboneBrick-##)
- One slave module (CAE_P-2Rel4Di2Ai-0#)
- Carrier Board (CAE_Y-CHBoc200)

Note: If you want to try out radCASE with the starterkit the adapter board for the Beaglebone Black must be the fully populated version (Z-BeagleboneBrick-01) or else the software, which is generated by radCASE will not work.

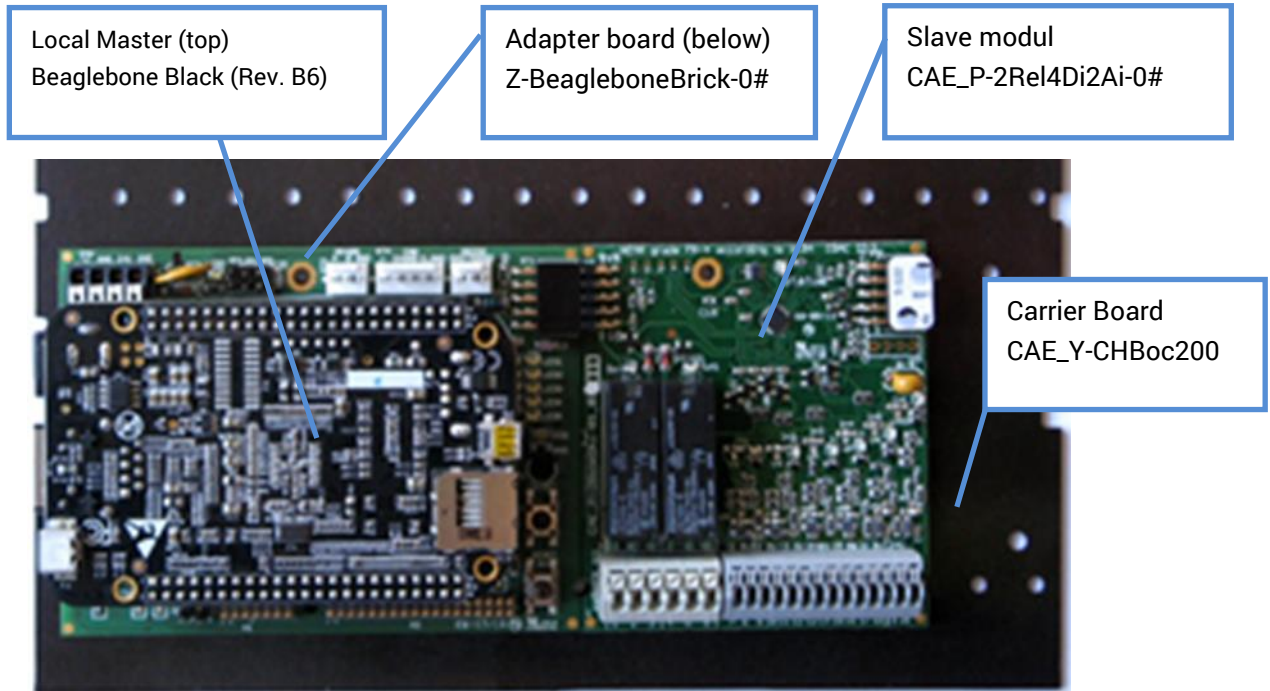
2.5 Additionally required components

To operate the starter kit, the following additional items are required:

- Beaglebone Black, Rev. B6 or later (with SD card and preinstalled Debian). If your Beaglebone Black is already flashed with Debian Version 7.5-2014-05-14 or higher, you won't need the SD card.
- DC power supply 24V, > 500mA
- Monitor with HDMI port or suitable adapter. Note: Not all adapters will work with Beaglebone Black immediately – in some cases, a separate reset of the Beaglebone Black after a power-on is necessary. The monitor does not work with the custom image for radCASE (if you want to try radCASE with Beaglebone Black). The signals are relayed to the LCD interface instead.
- USB keyboard, USB mouse and a powered USB-Hub
- RJ45 cable for LAN connection and internet access for diverse downloads

2.6 The Hardware

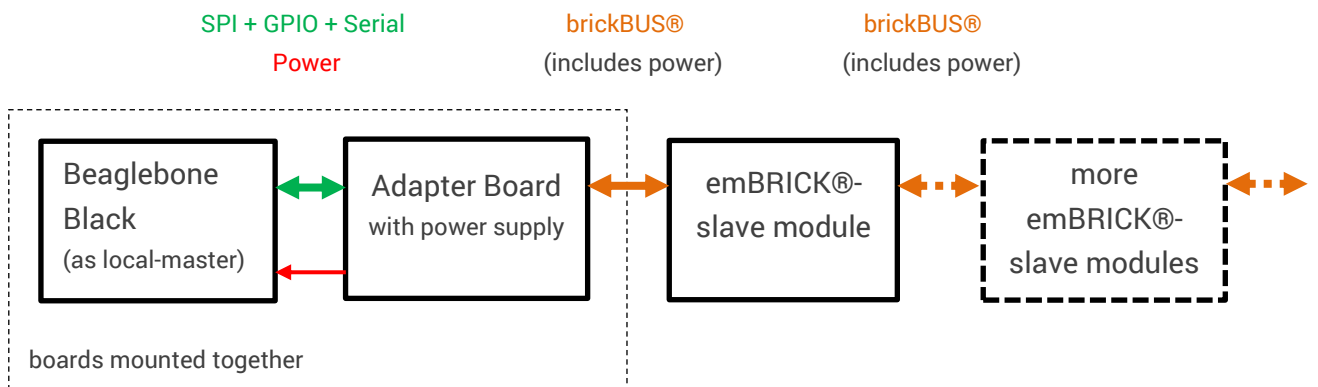
The adapter board is an electric coupler between the Beaglebone Black and an emBRICK®-*String*. The *String* consists of one or more *slave modules* which can be controlled via *brickBUS®*.



Picture 1

2.6.1 Communication structure

The Beaglebone Black acts as *local master* that exchanges data with the slave-modules via *brickBUS®*.



Picture 2

For detailed information, please read the description of the modules in the [System Manual](#).

2.7 The Software

There are two Board Support Packages available for download at embrick.de for this starterkit.

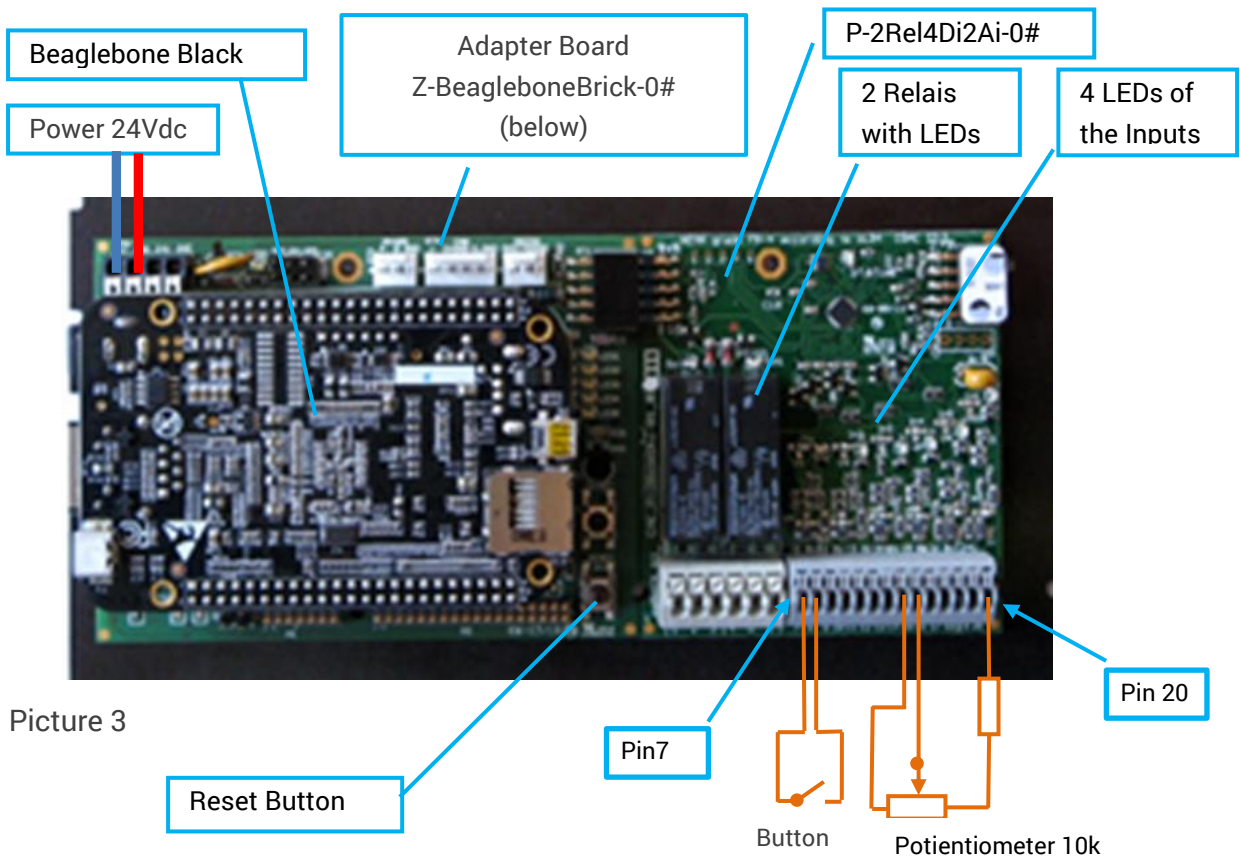
The BSP Z-Beaglebone-V## contains sources and examples in the programming language C. It includes an application, which you can compile and execute directly on the Beaglebone Black. If you want to try out Beaglebone Black and emBrick this way please use this package and go to the chapter Hands on Software – C Programming on Debian

The BSP Z-Beaglebone_rC-V## contains a test version of the radCASE software with an example project. If you want to try out radCASE with Beaglebone Black and emBrick please use this package and go to chapter Hands on Software with radCASE

3. Mounting and wiring

Please connect the *adapter board* to the *slave module* in the order shown in the picture below.

For more detailed information about the components (terminal assignment, electrical data etc.) refer to the [Product Catalogue](#).



Picture 3

1. There should be two spacers on the adapter board, which prevent faulty mounting. **If not you need to make sure you mount the Beaglebone Black correctly!** The LAN connector must face the short side with the power connector. In addition, the shape of the Beaglebone Black is printed onto the adapter board, which shows the correct mounting position. Mount the Beaglebone Black on top of the Z-BeagleboneBrick-0# adapter board. **Wrong installation can cause permanent damage to either device!**
2. Connect the adapter board (Z-BeagleboneBrick-0#) with the *slave module* (P-2Rel4Di2Ai-01) in the shown order.
3. Mount the boards onto the carrier board.
4. Connect LAN cable (with internet access), HDMI monitor (Note: The monitor does not work with the custom image for radCASE. The signals are relayed to the LCD interface instead.), keyboard and mouse to the Beaglebone Black USB Port.
5. Connect an external 24VDC (power off) to the Z-BeagleboneBrick-0#. Please refer to the *Product Catalogue* if you are unsure about the correct installation. Search for "Z-BeagleboneBrick-0#" and "CAE_P-2Rel4Di2Ai-0#".
6. Recommended: With three additional electronic parts, you can test the starter kit functionality.
 - a) Connect a potentiometer (10kOhm) via a series resistor (22kOhm) on pin 14 (ground), pin 15 (analog input), pin 19 (24V) on slave module.
 - b) Connect a button on I/Os pin 7 (digital input) and pin 8 (ground).

Result: Now the hardware is properly set up. Next up is the software installation.

4. Hands on Software – C Programming on Debian

4.1 Preparation

This section summarizes which releases/versions of hard and software is needed throughout this tutorial so that you can check if you have the proper hard or software if you encounter any troubles.

There are two sample applications with emBrick modules. If you execute the sample applications, a line for every module will be displayed.

For example: **module 01 ID:2-181, Mod-Vers: 13, Prot-Vers: 11**

- "module 01" means that the module is the first module connected to the Beaglebone Black adapter board.
- "2-181" is the ID of the module
- "Mod-Vers: 13" is the version of the module
- "Prot-Vers: 11" is the protocol version of the module. Each module should have the same or higher protocol version which is currently 11.

The first sample application includes the following emBrick module:

Module Name	ID	Module Version	Prot.-Version
CAE_P-2Rel4Di2Ai	5-131	18	11

The second extended sample application needs the following modules:

Module Name	ID	Module Version	Prot.-Version
CAE_B-4DimLEDI	4-566	12	11
CAE_B-6Moti2LDR	4-411	16	11
CAE_P-2Rel4Di2Ai	5-131	18	11
CAE_G-8Di8DO	2-181	13	11

If you start a sample application (in section 4.5) it should display the same version numbers or higher for the respective module or the sample application might not work as described.

Software used for the tutorials:

- Geany 1.22
- GNU Make 3.81
- GCC 4.6.3 (GNU Compiler Collection)

If you use software older than in the list above you might encounter problems while using the sample applications.

4.1.1 Install Debian

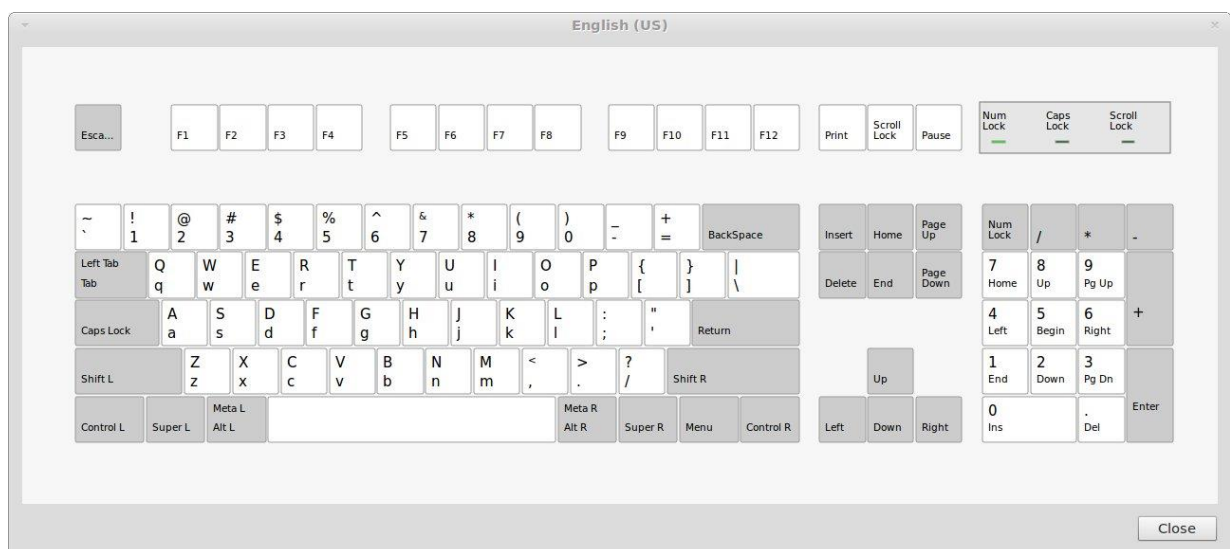
Turn on the 24VDC power supply. The Beaglebone Black will boot and display a desktop on the screen.

If your Beaglebone comes with an already preinstalled Debian Wheezy with Version 7.5 or 7.8 the examples and instruction in this manual should work. It could work with newer image releases but this is not guaranteed. You might have difficulties typing commands because your keyboard has a non-English layout. Please use Picture 4 below if you don't know how your keyboard corresponds to the English layout.

If you start Debian for the first time your mouse and keyboard might not be recognized. The reason for that could be that your USB hub isn't powered. If it is powered, try to press the reset button on the adapter board (see picture 3). If mouse and keyboard still aren't recognized try to switch mouse and keyboard to other connectors of the USB hub.

4.1.2 Configure the Keyboard

If you have an English keyboard layout you don't need to do anything since the Beaglebone Black is already preconfigured for this layout. In case you want another keyboard layout you have to go through the following steps.



Picture 4

1. Execute the two following commands in the LXTerminal. These commands will install the necessary software packages from the debian repositories and therefore need internet connection. These packages might be already installed. Try to install them anyway to be sure.

```
# sudo apt-get install console-data
```

2. During the installation a menu will be displayed in which you select your keyboard layout. As an example the german layout will be configured. Select the following menu options.
 - Select keymap from full list
 - pc / quertz / German / Standard / latin – no dead keys

3. Then install the second package like follows.

```
# sudo apt-get install keyboard-configuration
```

4. Open another configuration menu.

```
# sudo dpkg-reconfigure keyboard-configuration
```

Select the following menu options. This configuration is an example for the German keyboard layout.

- Generic 105-key (Intl) PC
- Other
- German
- German
- The default for the keyboard layout
- No compose key
- No

5. If you aborted the configuration in step 2 execute the command

```
# sudo dpkg-reconfigure console-data
```

Again, configure the layout like in step 2.

6. Now you have to reboot Debian with the command

```
# sudo reboot
```

Result: The keyboard is configured for your language.

4.1.3 Download the StarterKit Software

1. Start the Browser (usually you can find it in the menu in the lower left corner). This will start the Chromium Browser. Internet connection is required. If the message “Chromium can not be run as root” reboot Debian, this will start Debian without root rights. Then try to start the browser again.
2. Open the emBrick download page and download the board support package
3. The zip file should be now in /home/debian/Downloads. You will need it later.

4.2 Setup the Development environment

4.2.1 Install the C-Programming IDE for Beaglebone Black

To write and compile your own applications you need a C-IDE. We recommend using Geany IDE. The Beaglebone Black will need access to internet for the installation.

If you already have Geany installed on your Beaglebone Black you can ignore this step.

Install the *Geany IDE* step by step:

- Open a LXTerminal (Symbol in lower left corner → Accessories → LXTerminal)
- Type in LXTerminal: **sudo apt-get install geany**. This will install the Geany IDE.

Result: Now you can create and edit C-applications and compile them.

4.2.2 Enable SPI

Note: The instructions below were tested with Debian Wheezy 7.5 and 7.8. Other Version of Debian might work but this is not guaranteed.

The emBrick modules are controlled via SPI and this section shows how to do that.

Open the LXTerminal (Symbol in lower left corner → Accessories → LXTerminal) and type the following command.

```
# echo 'BB-SPIDEV0' | sudo tee /sys/devices/bone_capemgr.* /slots
```

The asterisk is a number which will be different for every installation. You can just hit the TAB-key after you typed the dot before the asterisk and the right number will be added.

If you get an error message which says “No such file or directory” the file BB-SPIDEV0.dtbo might be missing. You can copy this file from the /home/beagle/emBone directory (or the directory you choose for the support package. see 4.3) to the directory /lib/firmware and then try to execute the command **echo 'BB-SPIDEV0' | sudo tee /sys/devices/bone_capemgr.* /slots** again.

This will activate the SPI0 only temporary i.e. as long as Debian isn't rebooted. To make it permanent open LXTerminal and type the command **sudo leafpad /boot/uEnv.txt**. The leafpad text editor will pop up with the file uEnv.txt opened. Add the following line at the end of the file

```
optargs=capemgr.enable_partno=BB-SPIDEV0
```

and save the file and close the leafpad editor.

Result: Now the SPI0 of the Beaglebone Black is enabled.

4.3 Unzip the Z-BeagleboneBrick_VX.XX file

The emBone software package contains the *brickBUS@* protocol-stack, a simple hardware abstraction layer for the Beaglebone Black and a prepared small sample application to test the starter kit which can also be used as a template for own applications.

In this step the package you downloaded earlier will be unzipped to `/home/beagle`.

1. Execute the following command in the LXTerminal which will install unzip from the Debian repositories, therefore an internet connection is needed.

```
# sudo apt-get install unzip
```

2. Go to the `/home/debian/Downloads` directory by executing the following command.

```
# cd /home/debian/Downloads
```

3. To unzip the zip file `Z-BeagleboneBrick_VX.XX.zip` execute the following command. This creates the directory `/home/beagle/emBone` where the sourcefiles are stored. The parameter `-d` defines the directory where the zip file is unzipped.

```
# sudo unzip Z-BeagleboneBrick_VX.XX.zip -d /home/beagle
```

The subdirectory `Sources` contains the following files:

- `Appl.c` : The main source file containing the `main()` function
- `bB_EasyAPI.c` / `bB_EasyAPI.h` : a bundle of functions to access the brickBUS in different formats
- `bB_master.c` / `bBDefines.h`: The brickBUS Stack implementation. This is a general file used in different constellations.
- `brickbus.c` / `brickbus.h` : The hardware application layer containing the hardware-/platform specific interface functions
- `bBConfigs.h` : a set of macros to connect the brickBUS Stack in `bB_master.c` with the HAL in `brickbus.c`

The subdirectory `Examples` contains examples for a customer specific version of `Appl.c`

The file `BB-SPIDEV0-00A0.dtbo` is a cape Mgr file, see <http://elinux.org/Capemgr>

Result: Now you are ready to compile and start the sample application.














4.4 Load and compile the Sample Application

The main source file of the sample application is the *Appl.c*. It shows how to use the functions with a very simple API.

1. Open the LXTerminal
2. Type the command **sudo geany**. This opens geany with root rights so that it won't complain if it tries to compile the source code or execute an application.
3. Create a project with Project→ New. Give the project the name emBrick or any other name you like and click create.
4. Next click File→Open from the menu and then navigate to /home/beagle/emBone/Sources or wherever you copied the starterkit sources and select all files with Ctrl+A. Click Open.
5. Set up the commands for compiling and executing the programm.

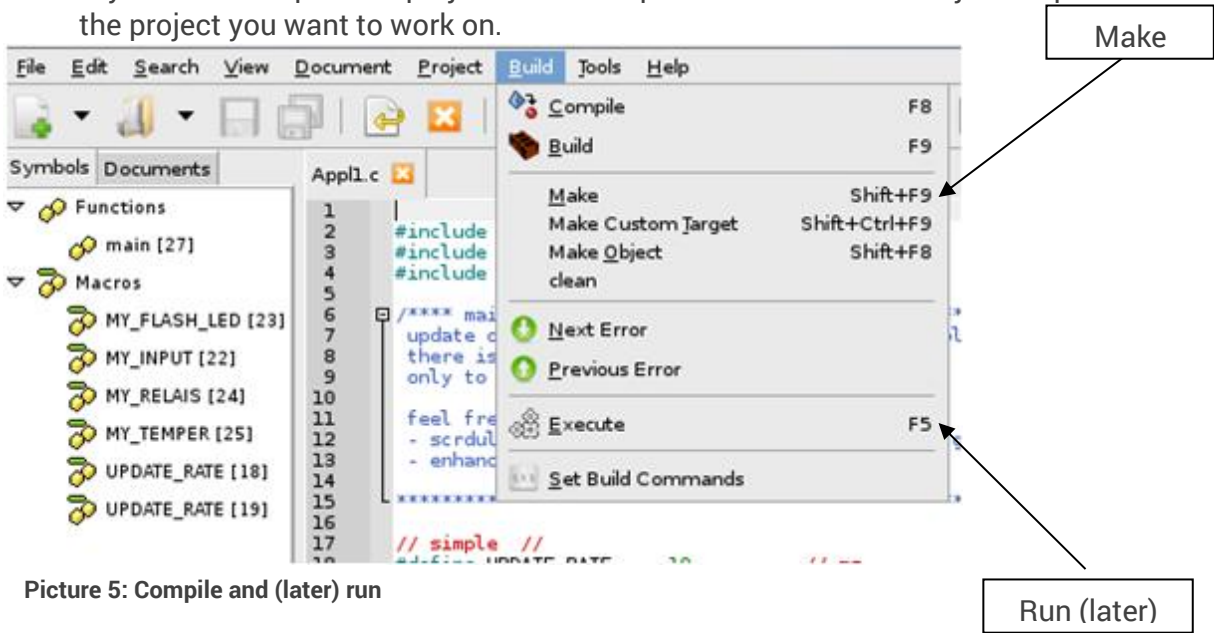
Note: You probably only need to set up the Execute command, because the other commands are probably already set by default. If a command is not set up you have to set it up like described below.

Go to Build→Set Build Commands, you will see a dialog. In the section "C commands" click the 3. button and enter "Clean and Make" and then enter "make clean; make " in the text field to the right of the button you just clicked. In the section "Independent commands" click the 1. button and enter "Make" and then enter "make" in the text field to the right of the button you just clicked. In the section "Execute commands" the 1. Button should already be named "Execute" in the text field right of this button enter "sudo ./Appl and click OK to save.

#	Label	Command	Working directory	Reset
C commands				
1.	<u>C</u> ompile	gcc -Wall -c "%f"		
2.	<u>B</u> uild	gcc -Wall -o "%e" "%f" -l bcm28		
3.	Clean and Make	make clean; make		
Error regular expression:				
Independent commands				
1.	<u>M</u> ake	make		
2.	Make Custom <u>T</u> arget	make		
3.	Make <u>O</u> bject	make %e.o		
4.	Clean	make clean		
Error regular expression:				
<i>Note: Item 2 opens a dialogue and appends the response to the command.</i>				
Execute commands				
1.	<u>E</u> xecute	sudo ./Appl		
2.				
<i>%d, %e, %f, %p are substituted in command and directory fields, see manual for details.</i>				
				 

6. You can now compile the sample application by "Build"→"Make".
7. If you want to recompile the application completely regardless if the code changed or not execute Build→Clean and Make to do that.
8. And execute the application with Build→Execute.

9. If you want to open the project at a later point in time select Project→Open and select the project you want to work on.



Picture 5: Compile and (later) run

Result: The sample application is now compiled into the executable file *App1* for the Beaglebone Black.

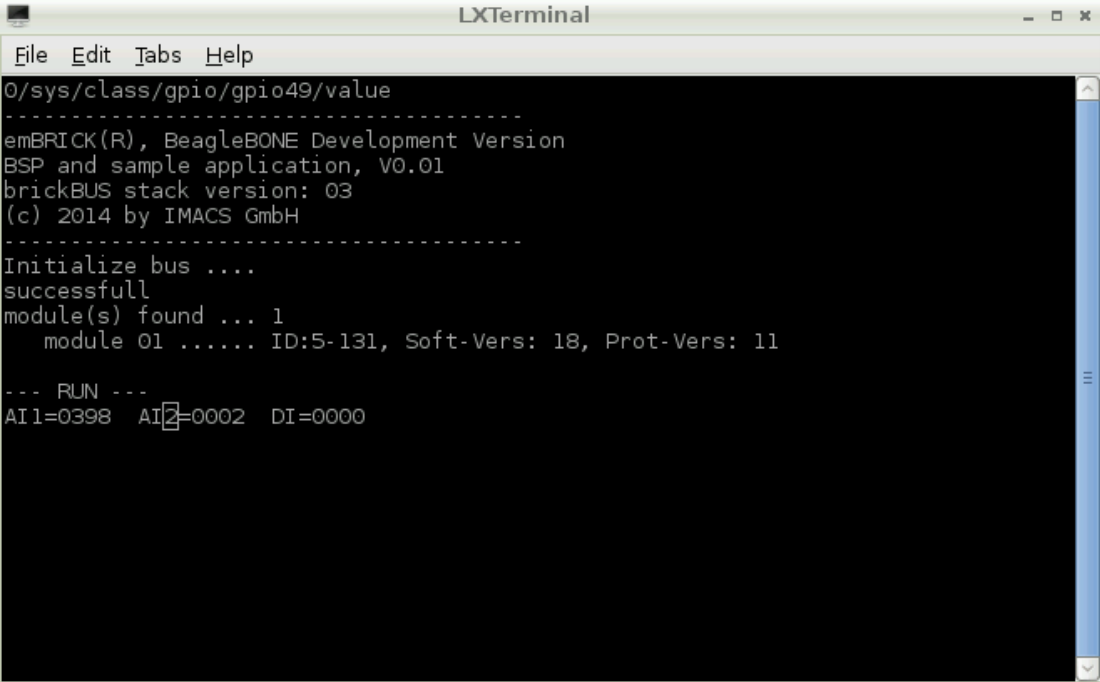
4.5 Start and explore the Functionality of "Appl"

Here we start/run the above compiled (and now executable) sample application *Appl*.

Start the Application via Geany IDE: -Menu → Build → Execute. The shown screen display appears and the relay 1 starts alternating.

Note: As mentioned above, the application uses simple threads with i.e. `sleep()` functions and console output. There is no claim about realtime and fast screen output. Therefore the permanent alternating relay could be shortly halted and the update of the input states/values is significantly delayed on the display. Feel free to modify and send us your enhancements.

With the starter kit (and the one I/O board) the application shows this screen:



```

LXTerminal
File Edit Tabs Help
O/sys/class/gpio/gpio49/value
-----
emBRICK(R), BeagleBONE Development Version
BSP and sample application, V0.01
brickBUS stack version: 03
(c) 2014 by IMACS GmbH
-----
Initialize bus ...
successfull
module(s) found ... 1
  module 01 ..... ID:5-131, Soft-Vers: 18, Prot-Vers: 11

--- RUN ---
AI1=0398 AI2=0002 DI=0000

```

Picture 6

The state of bus initialisation and the number of found slave-modules are shown. For every slave-module one additional line is printed with module-ID, the module software version and the *brickBUS@* protocol version.

The permanent execution loop in *Appl.c*, function *bb_Appl()* starts with ...

- updating the two analog inputs 1 and 2 (AI1, AI2), 0...11V = 0...1023 digits
- updating the four digital inputs 1..4 ("DI=xxxx"). Note: n-switchng input, 1=closed
- alternating the relay 1 with approx. 1s
- triggers LED1 (sign of life, you can see that the program runs)
- triggers relay 2 by the input state of DI1 (closed button forces activation of relay)

To stop the execution, press CTRL-C

For more details about the hardware, see *Product Catalogue* (search for: "Z-BeagleboneBrick-0#", "P-2Rel4Di2Ai-0#"). For more details about the software see source code of *Appl.c*, the other files of *emBerry* and the scheduling information in chapter 6.

4.6 Create your own application

The complete sample application consists of multiple hierarchical files that are included and linked together. The easiest and fastest way to write your own application is to use and modify the sample application. For this purpose, you can start by changing the file **Appl.C**. Once you have acquired some knowledge in the *emBRICK@* system you can also change **bb_EasyAPI.c()**.

Please follow the listed steps:

Note: Only LED1 can be used for testing purposes. LED2 and LED3 are already occupied.

1. Unzip the **Z-BeagleboneBrick_VX.XX.zip** again in **another** folder, i.e. */home/beagle/myappl*.
2. Open the **Appl.c** with Geany-IDE like before. (see section 4.4)
3. Application file **Appl.c**, the inside content of **bb_Appl()**.
4. Write your own application code inside the **bb_Appl()**. For this, use the I/O-functions and I/O-assignment listed below.
5. Start the compilation and execution of your application as described in 4.5

4.6.1 Example

Currently, the application controls the relay 00 so that it alternates every second. Now we change the application so that relay 01 also alternates every second.

- Unzip the **Z-BeagleboneBrick_VX.XX.zip** again in another folder, i.e. */home/beagle/example*.
- Open the **Appl.c** with Geany-IDE like before. (see section 4.4)
- Create a new definition for the second Relay **"#define MY_FLASH_RELAIS01 1,1,0,1"**
- Write the command **"bb_putBit(MY_FLASH_RELAIS01, (counter & 0x10) ? 1:0);"** inside the **bb_Appl()**.
- Start the compilation and execution of your application as described in 4.4 and 4.5
- Now the relay 01 and relay 02 alternates every second.

4.6.2 IO-Assignment

The data of the I/O modules are organized in a byte buffer for each module (a separate one of in- and out-data). To access this data, you need to know the ...

bus number..... (here always 1 because we have only one local bus),

slave number (1...),..... position of module in string,

byte position (0...) relative position/offset of the data inside the module image. For details of each module refer to [Product Catalogue](#), chapter 6.x.x.7, "process data image"

bit position (0..7) **only** in case of a bit access, indicates the bit in the selected byte

4.6.3 IO-Functions

The actual data access is performed by 6 simple functions that differ in the direction (read/write) and the data width (bit, byte, word).

read: `bB_getBit()`, `bB_getByte()`, `bB_getWord()`

write: `bB_putBit()`, `bB_putByte()`, `bB_putWord()`

Tipp: To avoid the manual input of the single digits in the function parameter, create a macro definition for each I/O you like to use that contains these digits.

For example:

```
#define MY_BUTTON          1,1,0,1    // Node 1, Module 1, Byte 0, Bit 1
```

this allows the coding: `bB_getBit(MY_BUTTON)` instead of `bB_getBit(1,1,0,1)`

Of course there are more functions to control the *brickBUS*® stack and access to process and condition data. Read more about in the next chapter or the [Programmers Manual](#).

The *bB_EasyAPI* is one possibility to access the stack. It is recommended to write your own set of API functions or modify the stack itself according to your requirements.

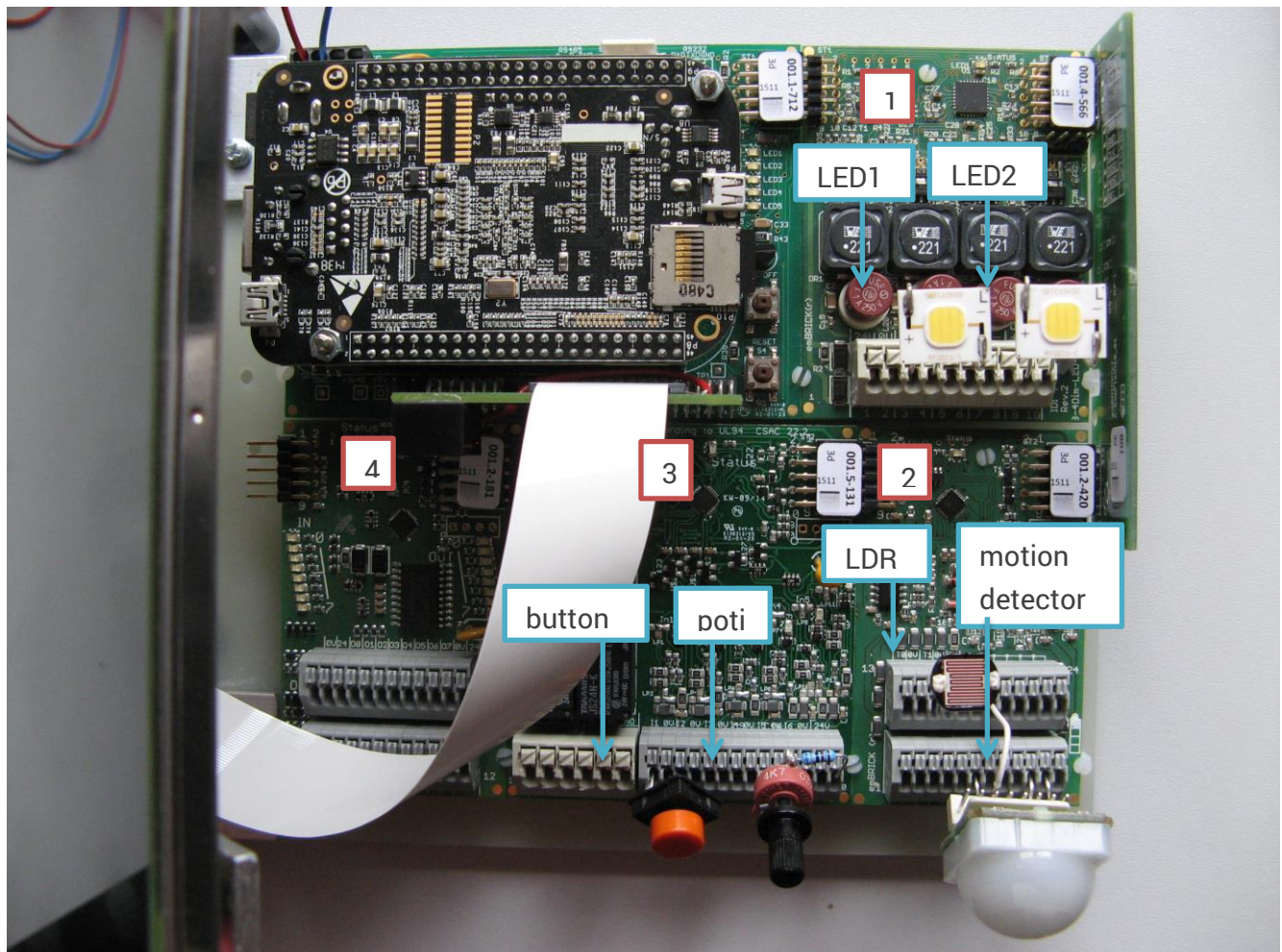
4.7 Externded Example

For this Example, you will need the following emBrick modules and components:

1. B-4DimLED1 with two LEDs which are connected to terminal 3 ("+"), terminal 4 ("-") and terminal 9 ("+"), terminal 10 ("-") respectively. You also need to connect a 24V power supply to terminal 1 ("24") and terminal 2 ("0").
2. B-6Moti2LDR with a LDR connected to terminal 15 ("T2"), terminal 16 ("0V") and a motion detector connected to terminal 11 ("I6"), terminal 12 ("0V").
3. P-2Rel4Di2Ai with a button connected to terminal 7 ("I1"), terminal 8 ("0V") and a potentiometer connected to terminal 17 ("I6"), terminal 18 ("0V").
4. G-8Di8Do (no components)

Please connect these modules in this order. The first module is closest to the Beaglebone Black, see the numbers in the red boxes.

It should look something like this



The needed Software for this example is in the Starterkit you downloaded earlier. "Appl_Demo2.c" is the one you need here. Copy the file "Appl_Demo2.c" in the folder Source which is also in the emBone folder. In the folder Source delete the file "Appl.c" and rename the copied file "Appl_Demo2.c" to "Appl.c". The easiest way to do that is to execute the two following commands:

```
# cd /home/beagle/emBone/Sources  
# cp ../Examples/Appl_Demo2.c Appl.c
```

Next, compile and run the downloaded Example like you did in chapter 4.4.

Here is what the application does:

The LED1 adapts its brightness to the ambient brightness. Try to cover the LDR to see how the brightness of LED1 changes. The motion detector triggers a relay if it detects any movement. You can control the brightness of LED2 with the potentiometer. The potentiometer also controls the switching frequency of the second relay which can be turned on and off with the button. The output LEDs on the 8Di8Do are representing a binary number which is counted up continuously.

5. Hands on Software with radCASE

5.1 Installing Debian for radCASE

You will need a custom Debian version for the Beaglebone Black if you want to use radCASE. The Debian image is downloadable here: http://embrick.de/downloads/remotemaster/beagleboneblack/Z-Beaglebone_rC_image1.img.zip

If you have downloaded the Debian image, you have to write it to a micro SD card like described in Debian installation guide, but only apply steps 3 to 5. Please do not try to flash the eMMC (internal flash drive) of the BBB with this image, it doesn't work with this image. Just put the SD card in the slot and turn the BBB on.

For the next step, you have to know the IP address. The BBB connects via DHCP to the local network by default. To find out the IP address you can use the program [fing](#).

Alternatively, if you are using a display the IP address is shown for a short time during boot up.

Now log in with [Putty](#) to the BBB with the following parameters:

Host Name: IP address of the BBB

Connection type: SSH

Port: 22

When you connect to the BBB you will be asked for login and password.

The login is **root** and password is **root123!**

When you are logged in execute the following command:

```
echo i2c-dev >> /etc/modules
```

This will activate the i2c devices.

5.2 Install the radCASE software

5.2.1 Cross compiler for radCASE

Download the compiler from

http://releases.linaro.org/14.09/components/toolchain/binaries/gcc-linaro-arm-linux-gnueabihf-4.9-2014.09_win32.zip

Unzip the compiler to <linarocompiler>, where <linarocompiler> can be any directory on your PC.

Next you need to create the environment variable LINARO_GCC with value <linarocompiler>\bin.

5.2.2 Install radCASE

You will need to install the evaluation version of radCASE. To do that, go to the internet page <http://www.radcase.com/try-it/> and register for “radCASE-Test”. If you are registered, you will receive an eMail. Follow the instruction in the eMail for installation. Once radCASE is installed you should read the chapter “First Steps” in the document “Getting Started”, which you can access via Help > Documentation in the radCASE IDE.

5.2.3 Board Support Package

After radCASE is installed, download the board support package for radCASE from embrick.de. It should contain the directories Common, rc_libObj and Soft.obj. Unzip the zip file and copy Common, rc_libObj and Soft.obj in a directory of your choice.

Next, create an environment variable with the name OSDL_SYS (if you didn't already) which points to the rc_libObj directory. Please make sure that there is no trailing “\” at the end of the variable.

After you have done the steps above go to the the Soft.obj\OSDL and doble click Gen_Neutral.rad. The radCASE editor should open the project file.

To generate the complete project (design, simulation, embedded software) go to “Build” in the menu and then select “Complete Generation”.

5.2.4 Executing the radCASE project and Visualization

If the radCASE project is successfully compiled you can find the compiled binary rcprogramm for the target in the directory Soft.obj\CTR\Target\target. This file must be transferred to the BBB file system. First, you have to find out the IP address of the BBB.

The BBB connects via DHCP to the local network by default. To find out the IP address you can use the program [fing](#).

Alternatively, if you are using a display the IP address is shown for a short time during boot up.

When you know the IP address open the batch file set_ip.bat in Soft.obj\CTR\Target and change the variable BBB_IP to the IP address of your BBB. Next, go to “Build” and select “Transfer to controller”. The radCASE editor will then transfer the generated program via Ethernet to the BBB and starts it.

If you want to use the visualization, the program must be executed while you try to connect the visualization with BBB. In order to execute the visualization on the PC you have to do at least the design compilation, which should already be done if you run a complete generation of the project. If you try to start a visualization without a design compilation, a dialog will pop up as a reminder.

You can start the visualization from the menu by selecting “Build” and then “Start visualization” or by clicking on the eye symbol. Navigate in the visualization to “settings” then again

“settings” and lastly “interface”. This will open a dialog in which you set the IP address for the visualization. This is the same address, which you entered for transferring the generated program to BBB.

If the visualization connection is successful, a green “on” symbol is shown in the lower left corner of the window.

Please read the “Quick Start Guide” for further information. You can access it via Help > Documentation. There is also the “Reference Manual” which can be helpful if you have question regarding designing/programming with radCASE.

6. Extendend Informations for brickBUS® Stack

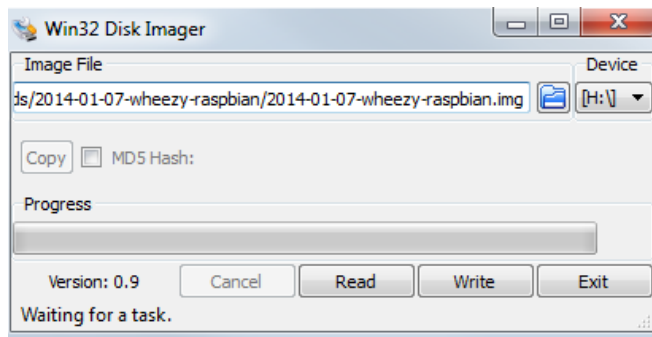
Currently you can only get this information on request.

Please contact: support@embrick.de

7. Appendix

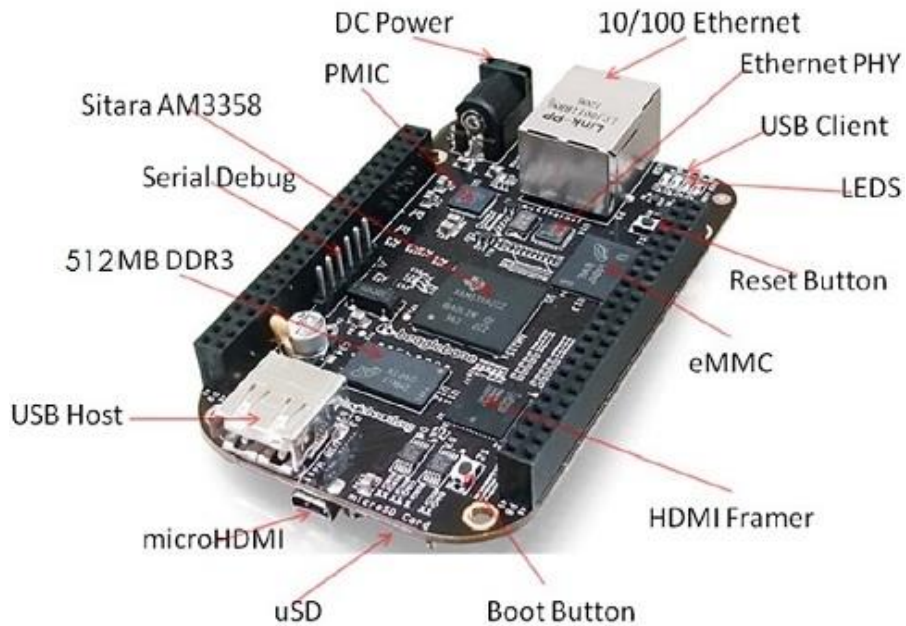
7.1 Debian installation guide

1. Download the latest version (last tested: BBB-eMMC-flasher-debian-7.5-2014-05-14-2gb) on your PC from beagleboard.org/latest-images
2. Please make sure that the file you downloaded is named BBB-eMMC-flasher-7.5-2014-05-14-2gb.
3. Unzip the file that you just downloaded . To extract the image you need to install [7zip](#). Right click on the file → 7zip → Extract Here. You will end up with a file ending with .img. This .img file can only be written to your SD card by special disk imaging software.
4. Download and install the [Win32Diskimager](#) software.
5. Writing Debian to the SD card.
 - a. Plug your SD card into your PC.
 - b. Start the Win32DiskImager (you find it in Start->Program).
 - c.



Picture 6: Win32 Disk Manager

- d. If the SD card (Device) you are using isn't found automatically then click on the drop down box and select it. **Caution: Make sure you choose the right drive otherwise another data storage could be overwritten and the data would be lost.**
- e. In the Image File box, choose the Debian .img file that you downloaded.
- f. Click Write and press Y-Button (on message) to permit the installation.
6. Flashing the the Beaglebone Black.
 - g. When the Disk Imager is finished insert the SD card into your (powered-down) board, hold down the USER/BOOT button (see picture below) or reset button on Adapter Board (if board is already mouted) and apply power, either by the USB cable (USB Client) or 5V adapter (DC Power).
 - h. You'll need to wait while the programming occurs (LEDs start to flash). When the flashing is complete, all 4 USRx LEDs will be steady on or off. The latest Debian flasher images automatically power down the board upon completion.



7.10 Configure static IP address

First you need to gather some information

7. Log in to the Beaglebone Black like described in section 2.3.

```
# ifconfig
```

The following output will be shown:

```
eth0 Link encap:Ethernet HWaddr 7c:66:9d:6d:02:ef
      inet addr:192.168.100.121 Bcast:192.168.100.255 Mask:255.255.255.0
      inet6 addr: fe80::7e66:9dff:fe6d:2ef/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
      RX packets:5654 errors:0 dropped:0 overruns:0 frame:0
      TX packets:250 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:2752213 (2.6 MiB) TX bytes:35650 (34.8 KiB)
      Interrupt:56
```

Write down the following information:

```
Inet addr: 192.168.100.121
Bcast: 192.168.100.255
Mask: 255.255.255.0
```

You will need it later.

8. Next execute the following command

```
# netstat -nr
```

You will get the following output:

Kernel IP routing table

Destination	Gateway	Genmask	Flags	MSS	Window	irtt	Iface
0.0.0.0	192.168.100.100	0.0.0.0	UG	0	0	0	eth0
192.168.7.0	0.0.0.0	255.255.255.252	U	0	0	0	usb0
192.168.100.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0

Look at the rows where Iface is "eth0". Write down the Gateway and Destination.

Gateway: 192.168.100.100

Destination: 192.168.100.0

Now we have what we needed to configure the static IP address.

9. Open the the file /etc/network/interfaces with nano with the command. Nano is a small text editor which comes with the Debian distribution.

```
# nano /etc/network/interfaces
```

10. Add the following lines at the end of the file

```
auto eth0
```

```
iface eth0 inet static
```

```
address 192.168.100.110
```

```
netmask 255.255.255.0
```

```
gateway 192.168.100.100
```

```
network 192.168.100.0
```

```
broadcast 192.168.100.255
```

The address can be any of the form 192.168.100.xxx except 192.168.100.255 (broadcast address), 192.168.100.100 (gateway address) and 192.168.100.0 (network)

Of course your addresses may be different.

11. Press CTRL+X when you're finished this will quit nano, but ask you if you want to save. Press the Y key and then Enter to save and quit.

12. Now you have to reboot by executing the following command

```
# reboot
```