

***emBRICK®* - EPC**

RaspberryBrick Starterkit-1
Local Bus / Raspberry Pi

Starter Kit - Local-Bus

Rev. 6

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) 71 54 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 2016. All rights reserved.

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

Content

1. The emBRICK® Mission	3
1.1 Basic Characteristics	3
1.2 Typical Applications.....	3
1.3 Programming, Host Platforms, Connectivity.....	3
2. Introduction.....	4
2.1 About this manual.....	4
2.2 References / manual overview	4
2.3 Package content	5
2.4 Separate required components	5
2.5 The Hardware	7
2.5.1 Communication structure.....	7
2.6 The Software	8
3. Mounting and wiring.....	9
4. Hands on Software - Raspbian.....	10
4.1 Preparation.....	10
4.2 Setup the Development environment	10
4.2.1 Install the Raspberry Pi C-Programming IDE	10
4.2.2 Install the Raspberry GPIO library bcm2835.....	10
4.3 Download the emBerrySupportPackage	12
4.4 Load and compile the Sample Application	12
4.5 Start and explore the Functionality of "Appl"	13
4.6 Create your own application	15
4.6.1 Example.....	15
4.6.2 IO-Assignment	15
4.6.3 IO-Functions.....	16
5. Extendend Informations for brickBUS® Stack	17
6. Extended Example	18
7. Appendix	21
7.1 Rasbian installation guide	21
7.2 Configuration of Rasbian.....	22
7.3 Connect to RaspberryPi with Xming	23
7.4 Exchanging files with WinSCP	24

1. The emBRICK® Mission

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

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 easily realize 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 large sized **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, BeagleBoneBlack

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

OS/RTOS Windows, Linux, FreeRTOS, proprietary

Programming languages C, C++, IEC61131, UML (by implementing expansion)

Developing environments MSVC, radCASE, Enterprise Architect, CODESYS, Coocox, MPLab, Geany (Raspberry Pi), every other C/C++ IDE

Host Interfaces Ethernet, WLAN, CAN, RS232, RS485, others planned

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

2. Introduction

2.1 About this manual

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

Furthermore it is used as an open reference platform 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..... 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 Package content

The Raspberry Starterkit-1 contains:

- Adapter board(Z-RaspberryBrick-##)
- One slave module (CAE_P-2Rel4Di2Ai-0#)
- Carrier Board (CAE_Y-CHBoc200)
- Sample application runs on Raspberry Pi with Raspbian(Version 2014-01-07) and Raspberry Pi 2 B with Raspbian(Version 2015-05-05) as operating system.

2.4 Separate required components

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

- Raspberry Pi, Rev. B (with SD-card and installed Raspbian)
- DC powersupply 24V, > 500mA
- Some wires and electronic components for own experiments (if you need)

... and furthermore the common items to run the Raspberry Pi:

- Monitor with HDMI connection or an adpter for HDMI (note: not all adapters will work with Raspberry Pi immediatelly - sometimes a separate reset of the Raspberry Pi after a power-on is necessary)
- USB keyboard and USB mouse
- RJ45 cabel for LAN connection and internet access for diverse downloads
- A soldering iron to connect the female header on the Raspberry Pi. So that the reset button is working.

2.5 Before you start

This section summarizes which releases/versions of hard and software is needed throughout the tutorials 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 ist he first module connected to the RaspberryPI adapter (CAE_Z-RasberryBrick-02).
- "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_G-8Di8DO	2-181	13	11
CAE_B-8TEMP	4-001	15	11
CAE_P-6Rel5DiPow	5-302	15	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.

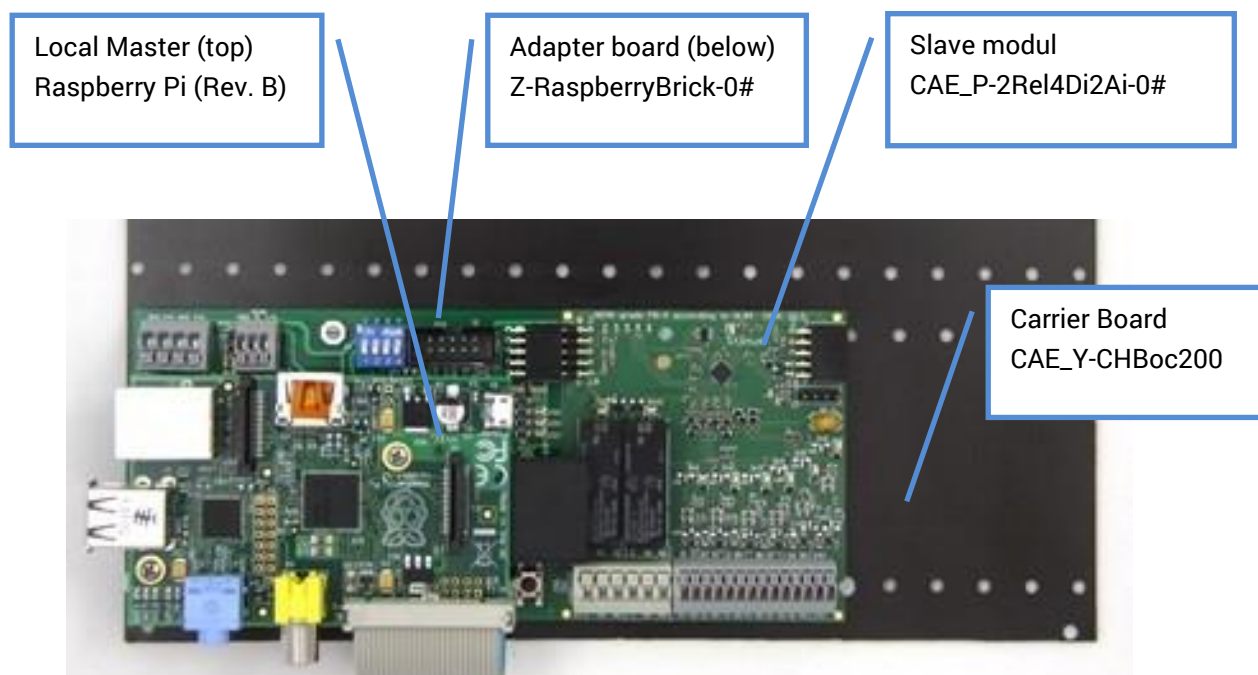
Software used for the tutorials:

- Geany 1.22
- GNU Make 4.0
- GCC 4.6.3 (GNU Compiler Collection)
- Bcm2835 1.45 (for GPIOs)

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

2.6 The Hardware

The adapter board is only an electric coupler between Raspberry Pi and an emBRICK®-*String*. The *String* consists of one or several *slave-modules*. *Slave-modules* receive the commands/data from the *brickBUS®* and control the I/Os.



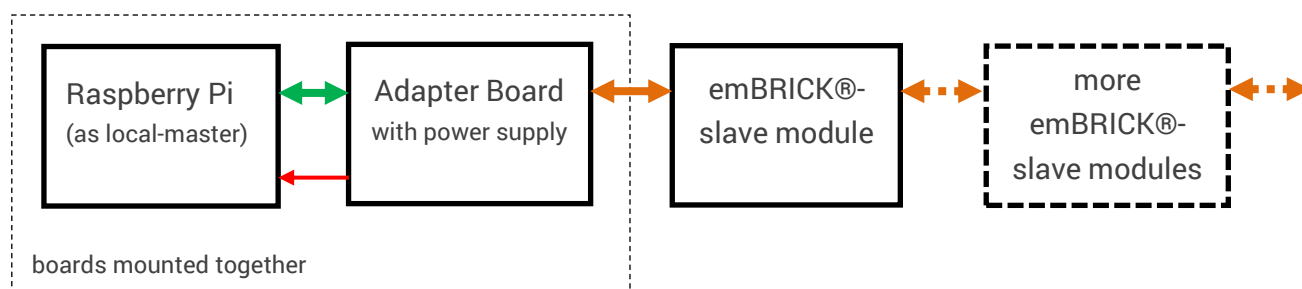
Picture 1

2.6.1 Communication structure

The application on the Raspberry Pi is the *local-master* that sends/receives commands/data to the slave-modules via *brickBUS®*.

SPI + GPIO + Serial brickBUS®brickBUS®

Power



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

2.7 The Software

For this starter kit a Board Support Packages (BSP) is available:

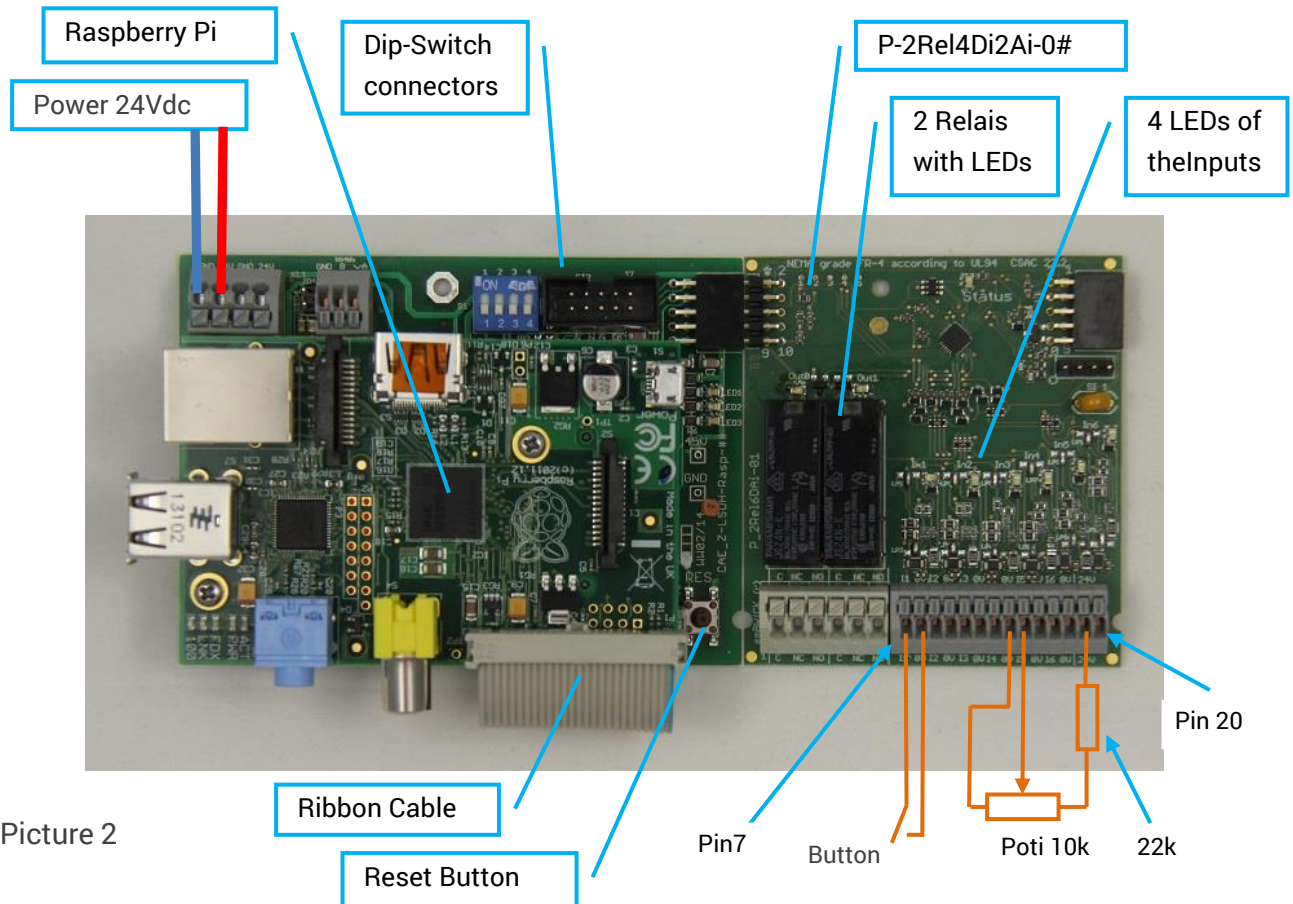
BSP_Z-RaspberryBrick-Vxx

The board support package (BSP) includes the *brickBUS@* protokol stack, the low level driver and a small sample application. The scheduling is done with simple threading methods, offered by the OS and standard C/C++ and a GPIO library.

Note: To keep the software simple, this sample has no claim of a real time behaviour or a fast/modern HMI. Of course it is possible (and recommended) to schedule *brickBUS@* stack with a realtime kernel to reach a definite deterministic behaviour or use a powerful graphic library. Feel free to realize this and send us feedback.

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 itself (terminal assignment, electrical data etc.) refer to the [Product Catalogue](#).



Picture 2

1. At first you have to solder the female header into the Raspberry Pi, without this the reset button won't work.
2. Mount the Raspberry Pi on top of the Z-RaspberryBrick-0# and connect them via the ribbon cable. Please be careful with the reset connector.
3. Connect the adapter board (Z-RaspberryBrick-0#) with the *slave module* (P-2Rel4Di2Ai-01) in the shown order.
4. Mount the boards onto the carrier board.
5. Connect LAN cable (with internet access), HDMI monitor, keyboard and mouse to the Raspberry USB Port. See picture chapter 7.1).
6. Connect an external 24VDC (power off) to the Z-RaspberryBrick-0#. Please refer to *Product Catalogue*. Search for "Z-RaspberryBrick-0#" and "CAE_P-2Rel4Di2Ai-0#".
7. 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).
 - b) Connect a button on I/Os pin 7 (digital input) and pin 8 (ground).

Result: Now the hardware is mounted and wired to start the software installation.

4. Hands on Software - Raspbian

Tip: Open this PDF-document directly on your Raspberry Pi to use the links in this document for easier navigation.

4.1 Preparation

To continue, you need a Raspberry Pi with a ready installed normal "Raspbian" as the operating system (on the SD-Card). For a quick reference see chapter 7.1. Furthermore you need a working internet connection.

4.2 Setup the Development environment

4.2.1 Install the Raspberry Pi C-Programming IDE

To write and compile your own applications you need a C-IDE (integrated development environment). Here we use Geany IDE.

8. Install the *GeanyIDE* step by step:
 - a. Boot up your Raspberry.
 - b. After the operating system has started - open the LXTerminal (Desktop→LXTerminal).
 - c. Type in LXTerminal: **sudo apt-get install geany** <enter> and confirm the installation a few seconds later with <Y>.

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

4.2.2 Install the Raspberry GPIO library bcm2835

You need a library for access to GPIOs of the Raspberry Pi (see [GPIO library bcm2835](#)).

You can download the file directly from [airspayce](#) or use the library from the starterkit folder.

9. Download the file [bcm2835-1.45.tar.gz](#) (click on this link **or** use the *NetSurf* browser on the Raspberry Pi with the URL <http://www.airspayce.com/mikem/bcm2835/bcm2835-1.45.tar.gz>). Download folder *home/pi* is automatically suggested.
10. Now you can find the downloaded file *bcm2835-1.45.tar.gz* in folder *home/pi* (Start→Accessories→File Manager). Unzip the downloaded file (right mouse-click on the *bcm2835-1.45.tar.gz* and choose "Extract here").
11. Open the unzipped folder *bcm2835-1.45* and find the *configure* file. To make *configure* executable right mouse-click on *configure* and choose "Properties". In the file properties dialog navigate to "Permissions" and click the check box "Make the file executable", if it is not checked. You might find a different permission dialog in which instead of the check box there is a dropdown menu "Execute" where you select "Anyone".
12. Start the LXTerminal from *bcm2835-1.45* folder (key F4) and give in the following command one by one:
 - a. **sudo ./configure**<enter> ... and wait till execution stops (you might need to make "configure" executable with **sudo chmod +x configure**)
 - b. **sudo make**<enter> ... and wait till execution stops
 - c. **sudo make check**<enter> ... and wait till execution stops
 - d. **sudo make install**<enter> ... and wait till execution stops
 - e. If you are using the Raspberry Pi 2 you have to enable the device tree support.

Type **sudo raspi-config** in the terminal. A menu opens in which you select "Advanced options" > "Device Tree" > "Yes" and then reboot so that the change can take effect.

Note: There might be a later version than 1.45 of bcm2835 which you should use.

Result: Now you can access the GPIO of the Raspberry Pi, e.g. it is possible to access the SPI.

4.3 Download the emBerrySupportPackage

The emBerry software package contains the *brickBUS@protocoll-stack*, a simple hardware abstraction layer for the Raspberry Pi and a prepared small sample application to test the starter kit and to be used as a template for own applications.

In this step the package will be downloaded from the Web and installed on your Raspberry Pi.

13. Download the zipped board support package Z-RaspberryBrick_# from www.embrick.de/starterkit
14. Unzip it into */home/pi* (or any other desired folder). This will create the folder */home/pi/emberry* with all necessary files inside. About a detailed description of the files inside, refer to chapter 5 [Extendend Informations](#).

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

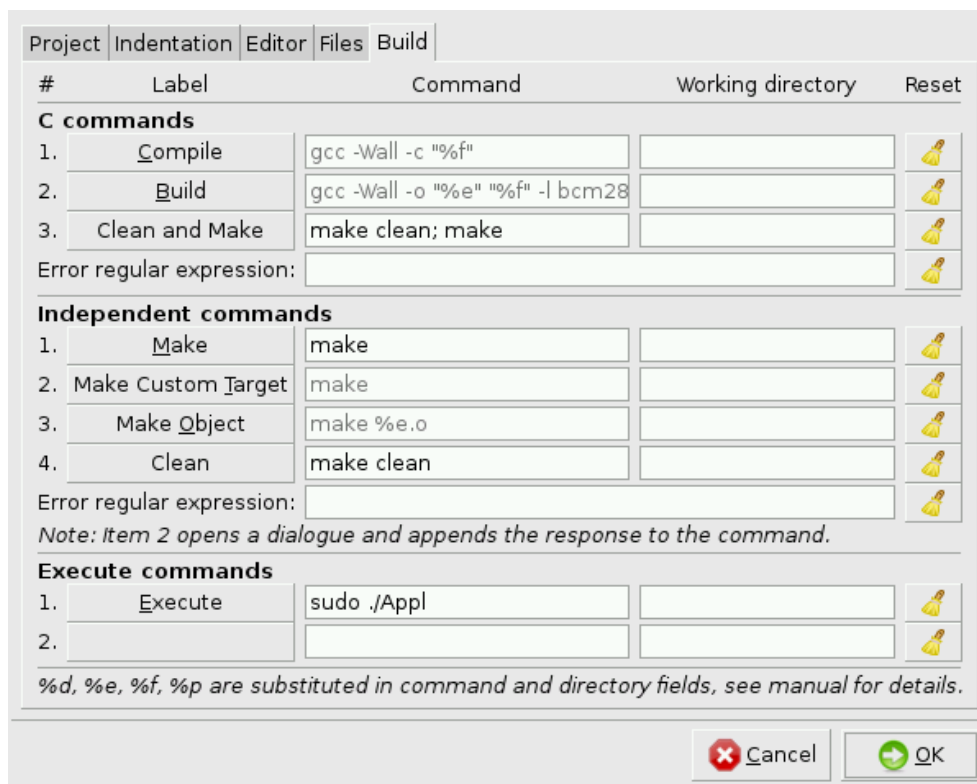
4.4 Load and compile the Sample Application

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

15. Open Geany and create a project with Project→New. Give the project the name emBrick or any other name you like and click create.
16. Next click File→Open from the menu and then navigate to */home/pi/emberry/Sources* or wherever you copied the starterkit sources and select all files with Ctrl+A. Click Open.
17. Set up the commands for compiling and executing the programm.

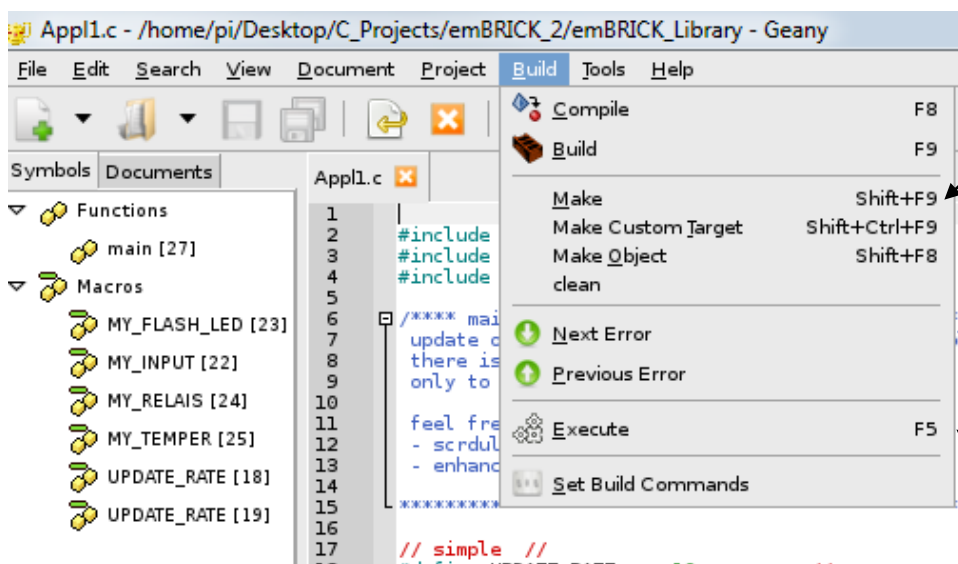
Note: You probably only need to set up the Execute command, because the other commands are probably already set by default. If a command is not set up you have to set it up like described below.

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.



18. You can now compile the sample application by "Build"→"Make".
19. If you want to recompile the application completely regardless if the code changed or not execute Build→Clean and Make to do that.
20. And execute the application with Build→Execute.
21. 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.

Make



Picture 3: Compile and (later) run

Run (later)

Result: The sample application is now compiled into the executable file *Appl* for the Raspberry Pi.

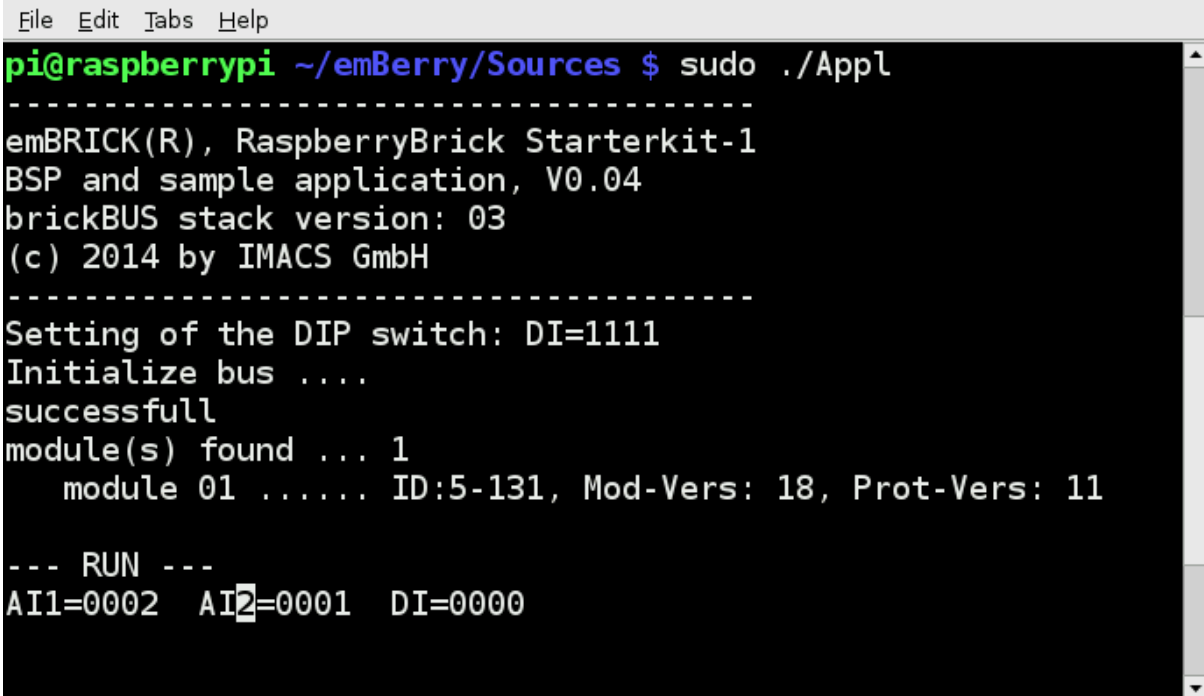
4.5 Start and explore the Functionality of "Appl"

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

22. Start the Application via Geany IDE: Menu→ Build →Execute. The below shown terminal window appears and the relay 1 starts alternating.

Note: As mentioned above, the application uses simple threads, 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 I/O board) the application shows this screen:



```
File Edit Tabs Help
pi@raspberrypi ~/emBerry/Sources $ sudo ./Appl
-----
emBRICK(R), RaspberryBrick Starterkit-1
BSP and sample application, V0.04
brickBUS stack version: 03
(c) 2014 by IMACS GmbH
-----
Setting of the DIP switch: DI=1111
Initialize bus ....
successfull
module(s) found ... 1
    module 01 ..... ID:5-131, Mod-Vers: 18, Prot-Vers: 11
-----
--- RUN ---
AI1=0002 AI2=0001 DI=0000
```

Picture 4

After the header, the state of DIP switch (on adapter board), the state of bus initialization 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.

Afterwards 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-switching input, 1=closed)
- alternates the relay 1 with approx. 1s
- alternates 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-RaspberryBrick-0#", "P-2Rel4Di2Ai-0#"). For more details about the software see source code of *Appl.c*, the other files of *emBerry* and information of scheduling in chapter 5.

4.6 Create your own application

The complete sample application contains multiple hierarchical files that are included and linked together. The easiest and fastest way to write an own application is to use and modify the sample application. For this purpose, you must only change one file: **Appl.C**. If you have deeper knowledge of the *emBRICK®* system, you can also change **bb_EasyAPI.c()**.

Please follow the listed steps:

Note:

- 1) The Geany-IDE reminds the previous opened files and probably open them automatically again. Be sure not to mix the files of the different folders.
- 2) For your test, you can take only LED1. The LED2 and LED3 are already occupied.

23. Unzip the **BSP_Z-RaspberryBrick-V#** again in **another** folder, i.e. **/home/pi/myappl**.
24. Open the **Appl.c** with Geany-IDE like before **as root** (in window on top with a click on Tools → "Open Current Folder as Root"). This is necessary to change the **Appl.c**.
25. In the sample application file **Appl.c**, delete content of the function **bb_Appl()**.
26. Write your own application code inside the function **bb_Appl()**. For this, use the I/O-functions and I/O-assignment listed below.
27. Start the compilation and execution of your application as described in 4.5

4.6.1 Example

Currently, the application controls the relay 1 so that it alternates every second. Now we change the application so that relay 2 also alternates every second.

- Unzip the **BSP_Z-RaspberryBrick-V#** again in another folder, i.e. **/home/pi/example**.
- Open the **Appl.c** with Geany-IDE like before. Open the **Appl.c** with Geany-IDE like before **as root** (in window on top with a click on Tools → "Open Current Folder as Root"). This is necessary to change the **Appl.c**.
- 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 4.5
- Now relay 1 and relay 2 alternate 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 and differs in the direction (reading/writing) and the data width (bit, byte, word).

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

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

In case you want to know more about those functions refer to the comments/description inside the files ***bB_EasyAPI.h*** and ***bB_EasyAPI.c***, where they are defined and implemented.

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 special requirements.

5. Extendend Informations for brickBUS® Stack

Currently you will get this information only on request.

Please contact: support@embrick.de

6. Extended Example

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

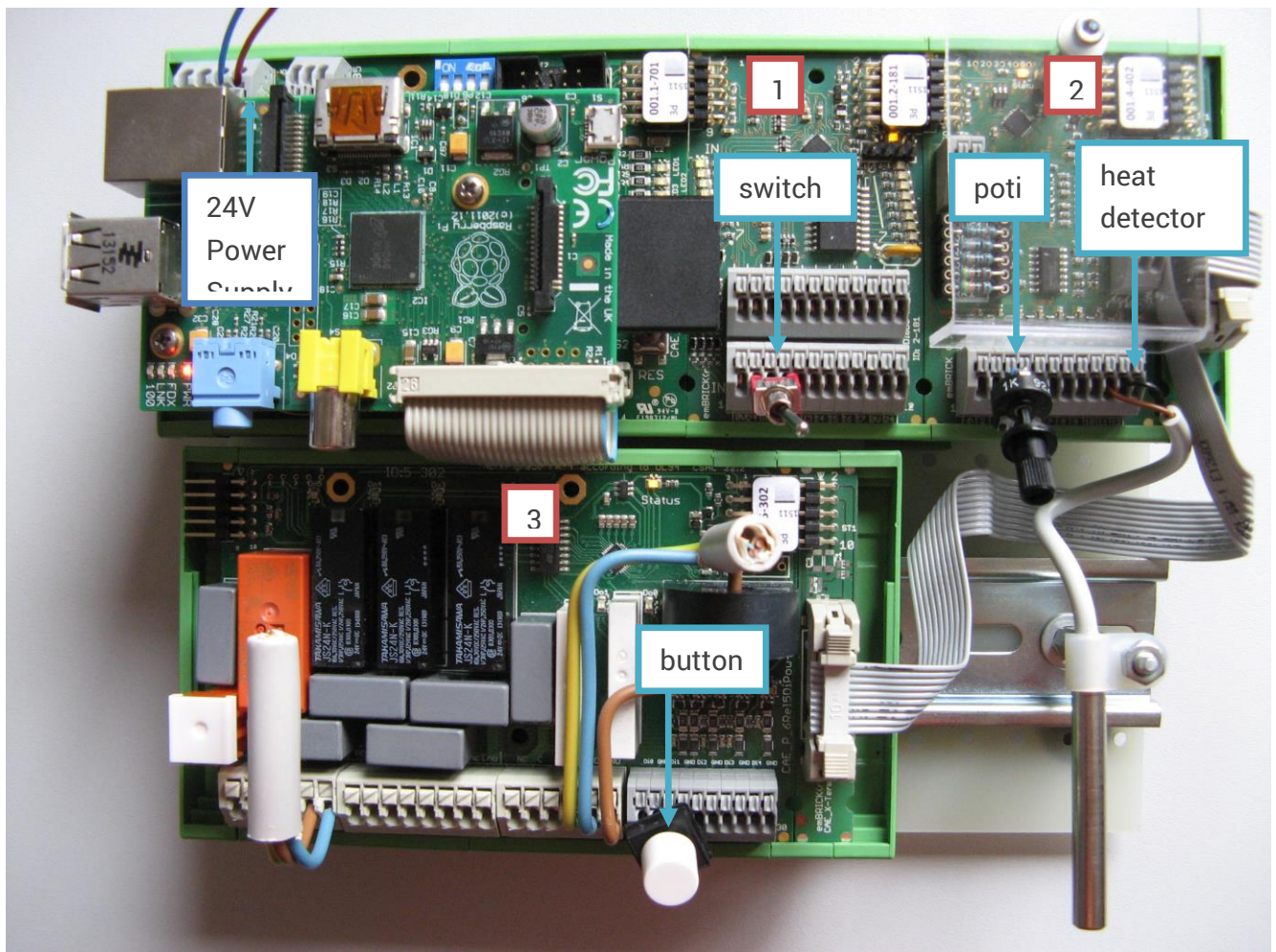
1. 8Di8Do with a switch which is connected to the terminal 3 ("I1") and terminal 2 ("24").
Here you will also need a 24V power supply. Connect the 24V of the power supply to terminal 24 ("24") and the ground of the power supply to terminal 1 ("0V").
2. 8Temp with a 1,5kOhm resistor in line with a 1kOhm potentiometer connected to terminal 3 ("T1") and terminal 2 ("0V"). Also connect a KTY 81-210 temperature sensor to terminal 10 ("T8") and terminal 11 ("0V").
3. 6Rel5DiPow with a button connected to terminal 21 ("DI1") and to terminal 22 ("GND").

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

You can look up the terminals of the modules in the [Product Catalogue](#).

The next step is to connect a power supply of 24V to the RaspberryPi adapter board. You can just use the same power supply wich you use for the module 8Di8Do. Connect GND and 24V of the power supply to GND and 24V on the adapter board. Make sure that the power supply also supplies up to 500mA.

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 em-Bone 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 in the LXTerminal:

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

Next, compile and run the downloaded Example with Build→Clean and Make in Geany. This will delete the object files and and binary from the last compilation. To start the sample application click Build→Execute in the menu. This of coarse works only if you already did the necessary steps in section 4.4.

Here is an alternative for compiling and executing the sample application if for some reason this doesn't work.

Open a LXTerminal and execute the following commands.

Go to the sample application sources

```
# cd /home/beagle/emBone/Sources
```

Delete the previously compiled sample application and object files (with ending ".o")

```
# sudo make clean
```

Compile the new application

```
# sudo make
```

Make the compiled Application executable

```
# sudo chmod +x Appl
```

Execute the sample application

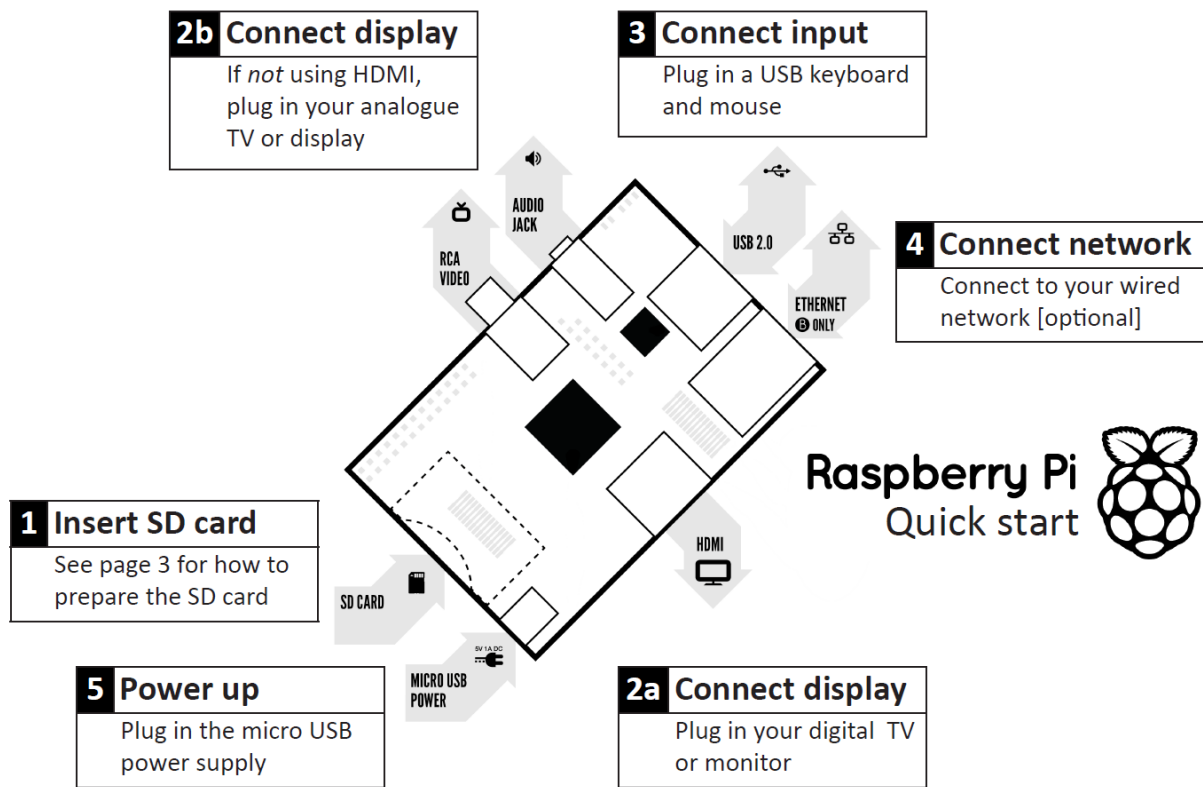
```
# sudo ./Appl
```

Here is what the application does:

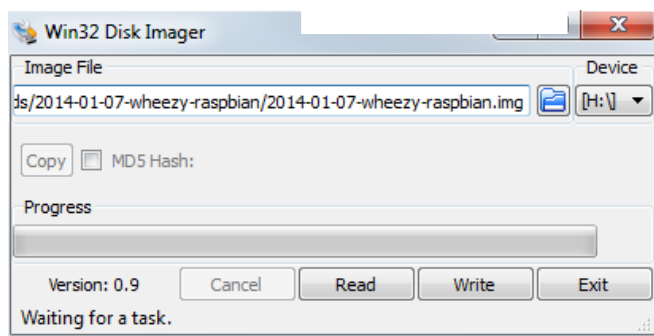
On the 8Di8Do module you should see a chaser light which you can start and stop with with the switch. The potentiometer controls the frequency of the chasing lights. The button turns on and off the relays which should alternate like a chasing light. If the temperature sensor gets warmer the relays alternate faster an vise versa.

7. Appendix

7.1 Rasbian installation guide



1. Download the current version of the Raspian image on your PC: <http://www.raspberrypi.org/downloads>. (last tested version is 2015-05-05-raspbian-wheezy)
2. Unzip the file that you just downloaded (Right click on the file and choose "Extract all"). Follow the instructions—you will end up with a file ending in .img. This .img file can be written to a SD card via special disk imaging software.
3. Download and install the [Win32Diskimager](#) software.
4. Writing Raspbian to the SD card.
 - a) Plug your SD card into your PC.
 - b) Start the Win32Disk (... ->Program).



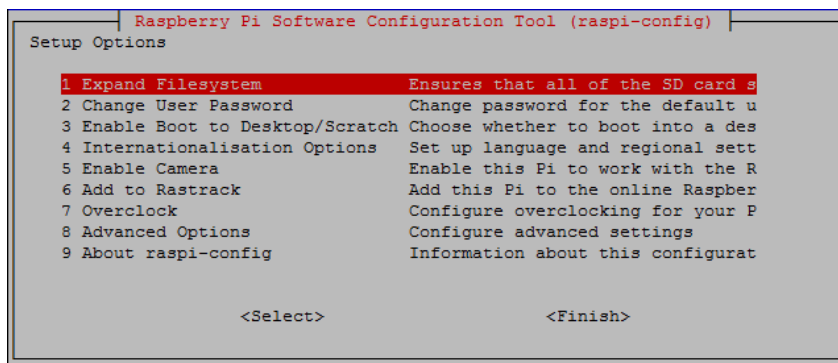
Picture: Win32bDisk Manager

- c) If the SD card (Device) you are using isn't found automatically then click on the drop down box and select it.
- d) In the Image File box, choose the Raspbian .img file that you downloaded.
- e) Click Write and press Y-Button(on message) to permit the installation.
- f) After a few minutes you will have a SD card that you can use in your Raspberry Pi.

7.2 Configuration of Rasbian

Raspberry starts with a configuration menu (at the first start). If you have already configured Raspberry, then skip this chapter.

1. Insert the SD card with Rasbian image into Raspberry.
2. Switch the power on. You will see the following picture.



Picture: Raspi-Config

2. In menu 4 "Internationalisation Options" set up your:

Locale/1->Change Locale->en_GB.UTF-8 UTF-8(choose with space and jump with tab).

Confirm your selection with OK(in next dialoge).

Timezone/2->Change Timezone-> choose your land-> choose your city (with same time).

Keyboard Layout/2 Change Keyboard Layout-> Generic 105-key (Intl) PC-> choose your language->choose:the default for the keyboard layout->choose:no compose key->choose no terminate the X server.

3. In menu 3 "Enable Boot to desktop/Scratch" set up the boot option:

Desktop Log in as user 'pi' at the graphical desktop.

4. Reboot the Raspberry (Finish->Reboot-OK).

Or optionally: Click on <Finish> and type: `sudo shutdown now -r`

Tip: You can always call the configuration menu with the order in the LXTerminal, if you want to change the configuration settings:

sudo raspi-config<enter>

Remember your account data (default):

Username: pi

Password: raspberry

7.3 Connect to RaspberryPi with Xming

This section explains how to connect to RaspberryPi via ethernet to get a virtual monitor on your PC.

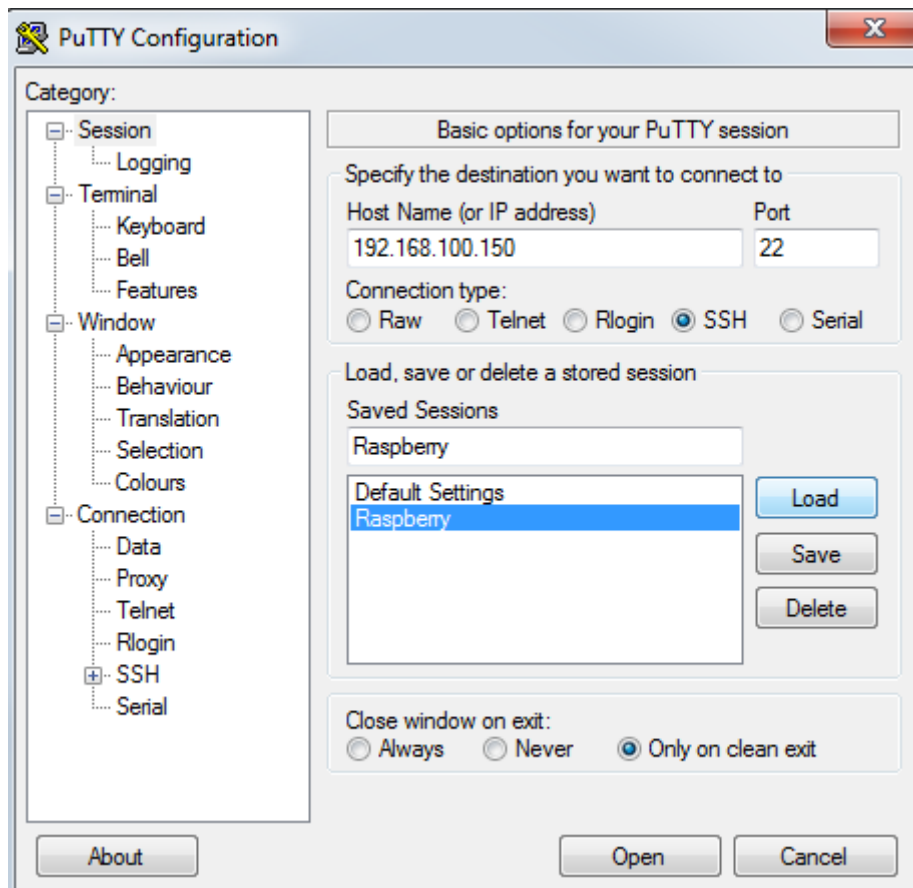
1. Download Xming from <http://sourceforge.net/projects/xming/> and install it (if not installed already)
2. Download Putty from <http://www.putty.org/> (if not installed already)
3. Configure the Raspberry (you need to do that only once)
 - a. Open a LXTerminal and execute the command below

```
sudo nano /etc/network/interfaces
```

- b. Change the line "iface eth0 inet dhcp" in the file to

```
iface eth0 inet static
    address 192.168.1.2 (!Attention this IP must not be occupied)
    netmask 255.255.255.0
    gateway 192.168.1.1
```

- c. Save the changes and exit nano:
Press Ctrl+X
Enter Y to save the change
Hit Enter
 - d. Connect the RaspberryPi to a local network
 - e. Restart the RaspberryPi
4. Start a X-Server with Xming. If you are not familiar with Xming, just start XLaunch (you should find it in the start menu after the installation) and click the forward button and than the finish button in the last dialog.
 5. Start PUTTY, put in the host Name or the IP-Adresse of the RaspberryPi, Port 22 and in SHH→X11 activate Enable X11 forwarding (click the check box). If you want to save this configuration, then click on Save, but you have to enter a session name first (right below "Saved Sessions").



- Once Putty started and you will be asked for the login name and password (usually the login name is "pi" and the password "raspberrypi"). Enter the login name and password. If the login was successful type in the following command in the command line:

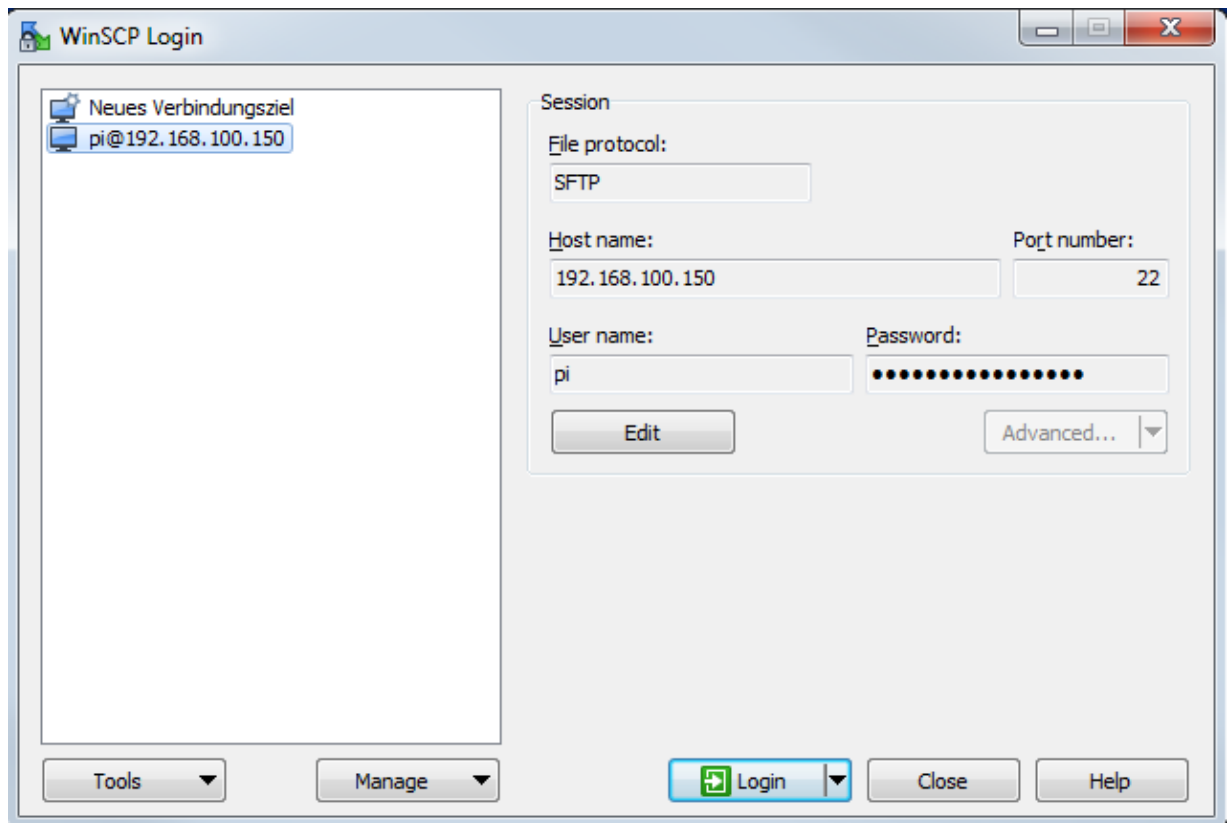
`lxsession&`

Xming should open a window which represents the RaspberryPi desktop (it could take a while).

You should see a bar at the bottom or top. To the left hand side of the bar (in the corner of the screen) is a menu button. In Menu→Programming you will find the Geany IDE if you installed it (see 4.2). In Menu→Accessories you can find a text editor, the terminal in which you can execute commands which are mentioned throughout the tutorial, and the file manager which you can use to browse the file system of the RaspberryPi.

7.4 Exchanging files with WinSCP

- Download and install WinSCP from <https://winscp.net/eng/download.php>
- Select the file protocol, enter the host name of the RaspberryPi (can be an IP address), username and password. You may want to save your session details Press Save button. Press Login to connect.



If you want to change files in the system directories like /etc you have to log in as root.

