Lab 1: Hello World, Blinking LED

Getting used to the lab platform and development environment

Farhang Nemati, March 2016

In this lab you will learn how to build and debug programs using Visual Studio and VisualGDB for Linux platforms (Raspbbery Pi in the lab). In the lab you will write a simple “Hello World” program and run it on the Raspberry Pi. You will also write a program that makes a LED to blink. Please read document “General Instructions” first!

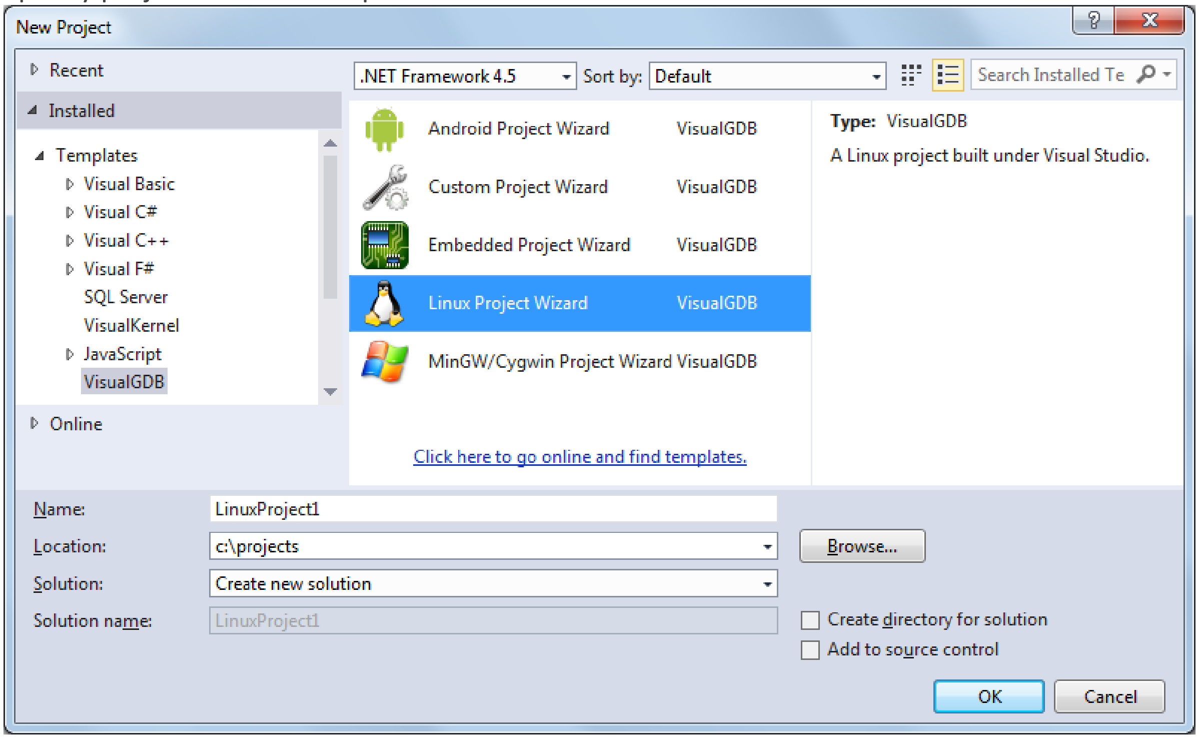
Connecting and Running the Raspberry Pi

Please be careful when working with the hardware; a failure in connecting and running the hardware may destroy it. Ask the lab assistant for help!

First of all you have to connect the Pi to network (e.g., using yellow network cables). Then connect the power supply adapter to the Pi and connect it to electricity. Wait until Pi is up and running. Please notice that VisualGDB is installed on only 15 computers in the lab. Make sure that you are working with a computer where VisualGDB has been installed on. Now follow the following steps to build the “Hello World!” program:

A. Create “Hello World!” Project

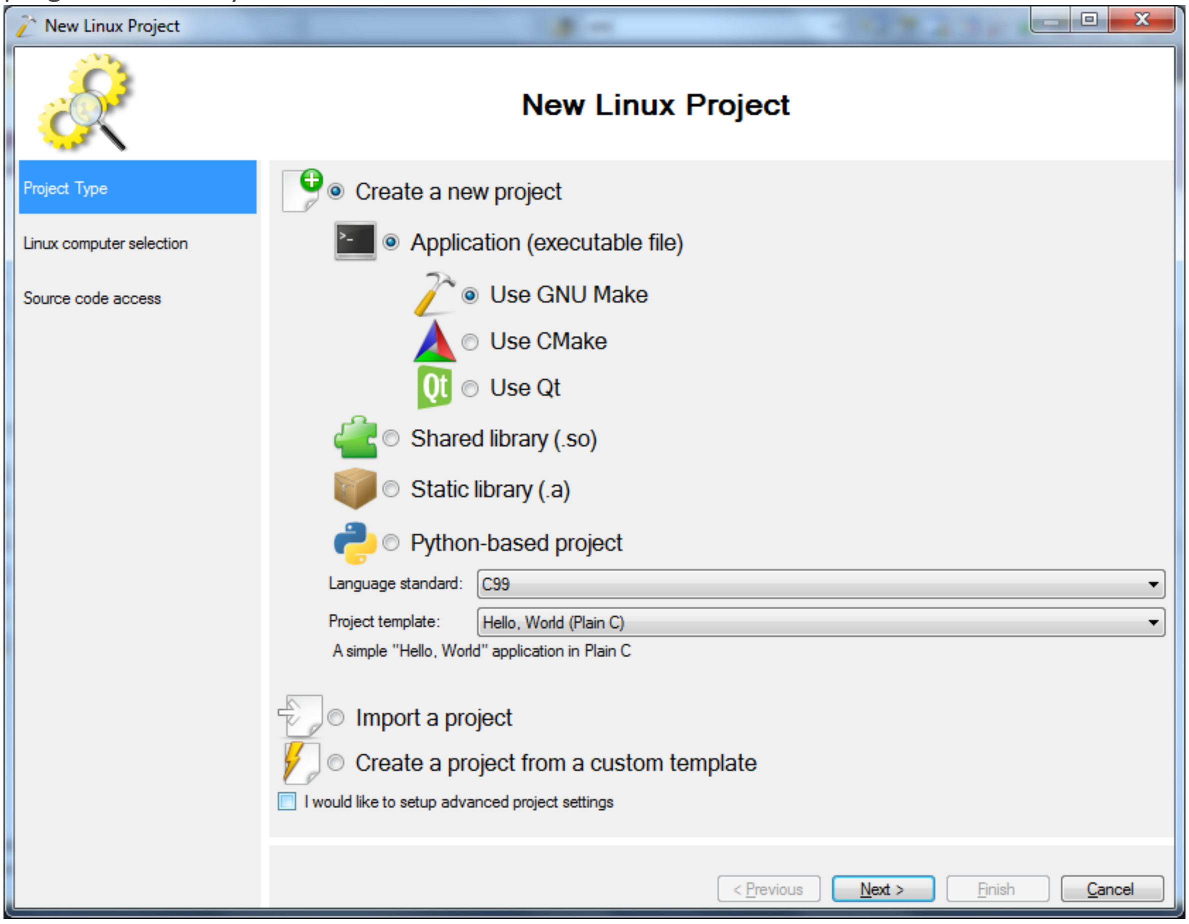
1. In Windows Visual Studio, select “File->New project”. Then select “VisualGDB->Linux Project Wizard”. Specify project location and press “OK”.


New Project
Recent
Installed
Templates
Visual Basic
Visual C#
Visual C++
Visual F#
SQL Server
VisualKernel
JavaScript
VisualGDB
Online
.NET Framework 4.5
Sort by: Default
Search Installed Te
Android Project Wizard VisualGDB
Custom Project Wizard VisualGDB
Embedded Project Wizard VisualGDB
Linux Project Wizard VisualGDB
MinGW/Cygwin Project Wizard VisualGDB
Click here to go online and find templates.
Browse...
Create directory for solution
Add to source control
OK
Cancel

Type: VisualGDB
A Linux project built under Visual Studio.

Name:	LinuxProject1
Location:	c:\projects
Solution:	Create new solution
Solution name:	LinuxProject1

2. The VisualGDB Linux Project Wizard will start. As you are writing a simple "Hello World!" program, keep "Create a new project". Change "Project template" to "Hello, World (Plain C)". You can also write the program in C++ if you are familiar with C++. Press "Next" to continue.


New Linux Project

New Linux Project

- Project Type
- Linux computer selection
- Source code access

[✓ Create a new project]

[✓ Application (executable file)]

[✓ Use GNU Make]

Use CMake

Use Qt

Shared library (.so)

Static library (.a)

Python-based project

Language standard: C99

Project template: Hello, World (Plain C)

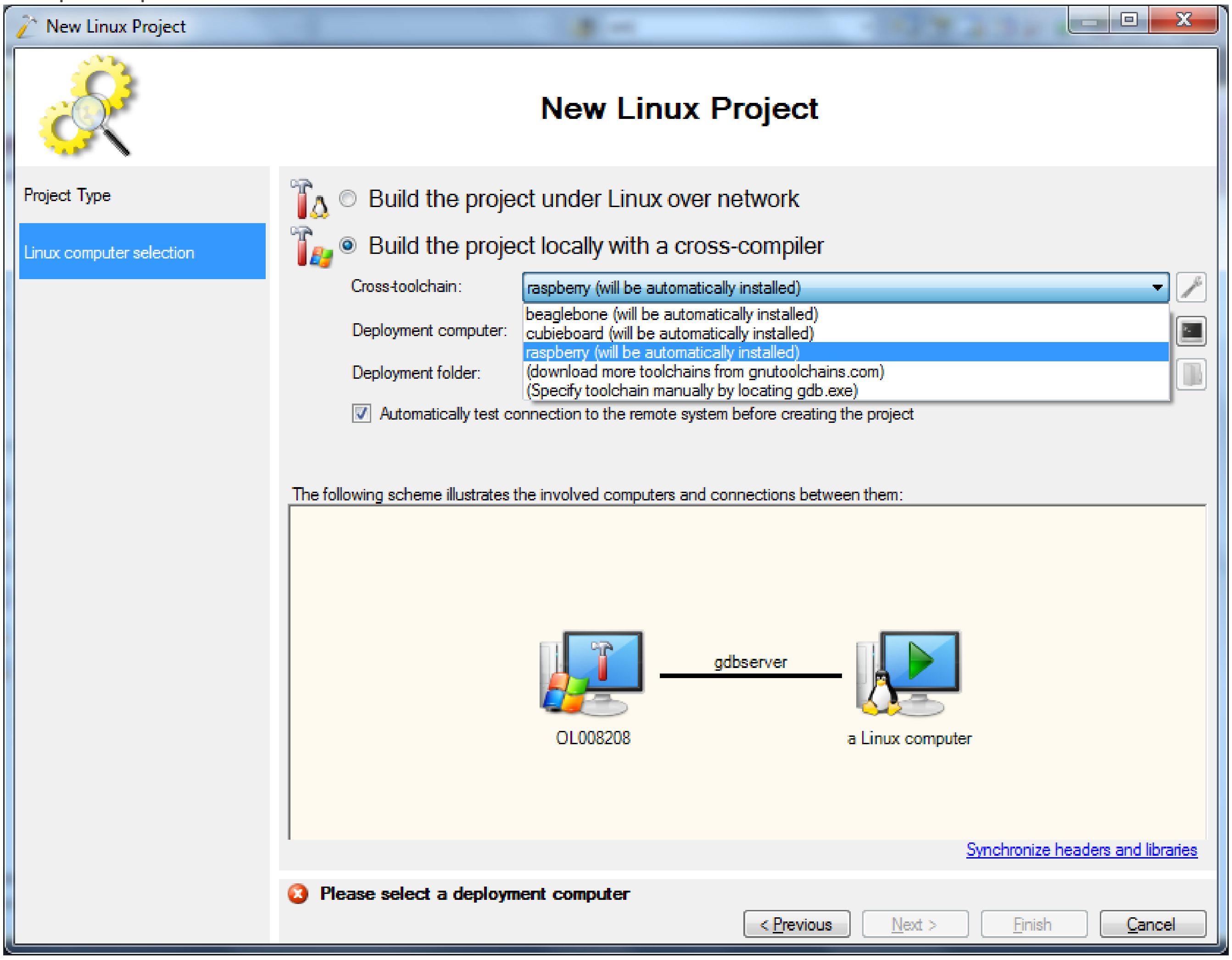
A simple "Hello, World" application in Plain C

Import a project

Create a project from a custom template

I would like to setup advanced project settings

3. You can either use the native Linux on the Pi for compiling your project or compile it locally (using a cross-compiler that VisualGDB will install for you) on Window and deploy (in the folder specified in "Deployment folder") it on Pi. Here we focus on the latter option. Thus select "Build the project locally with a cross-compiler". In "Cross-toolchain" select "raspberry …". VisualGDB then asks you to install the toolchain. Accept and push the "Install" button.



( ) Build the project under Linux over network
(●) Build the project locally with a cross-compiler

Cross-toolchain: raspberry (will be automatically installed)
- beaglebone (will be automatically installed)
- cubieboard (will be automatically installed)
- raspberry (will be automatically installed)
- (download more toolchains from gnutoolchains.com)
- (Specify toolchain manually by locating gdb.exe)

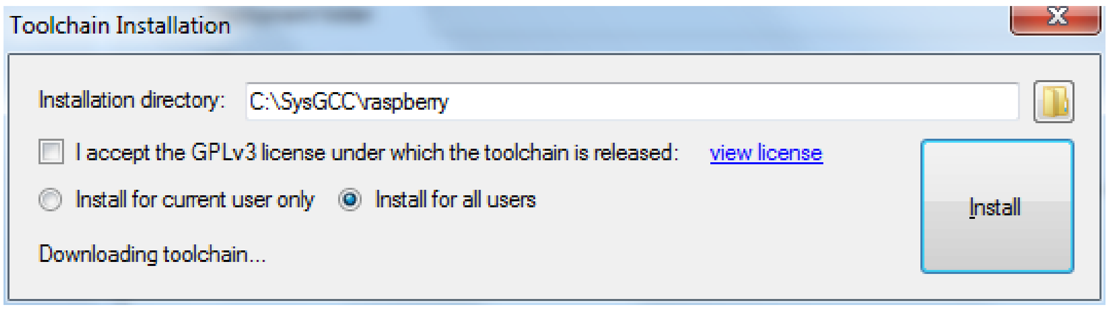
Deployment computer:

Deployment folder:

[✓] Automatically test connection to the remote system before creating the project

The following scheme illustrates the involved computers and connections between them:

Please select a deployment computer



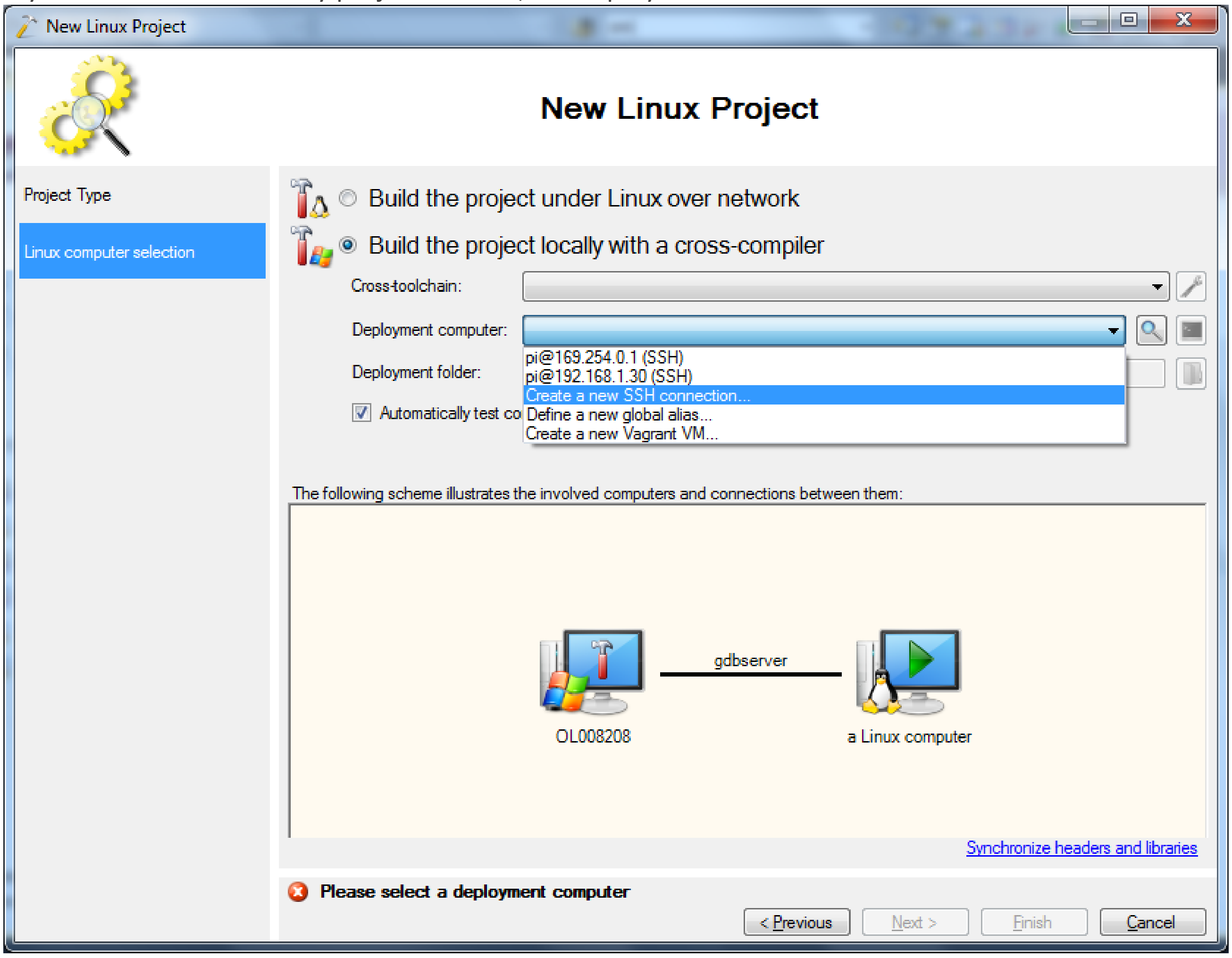
Installation directory: C:\SysGCC\raspberry

[ ] I accept the GPLv3 license under which the toolchain is released:

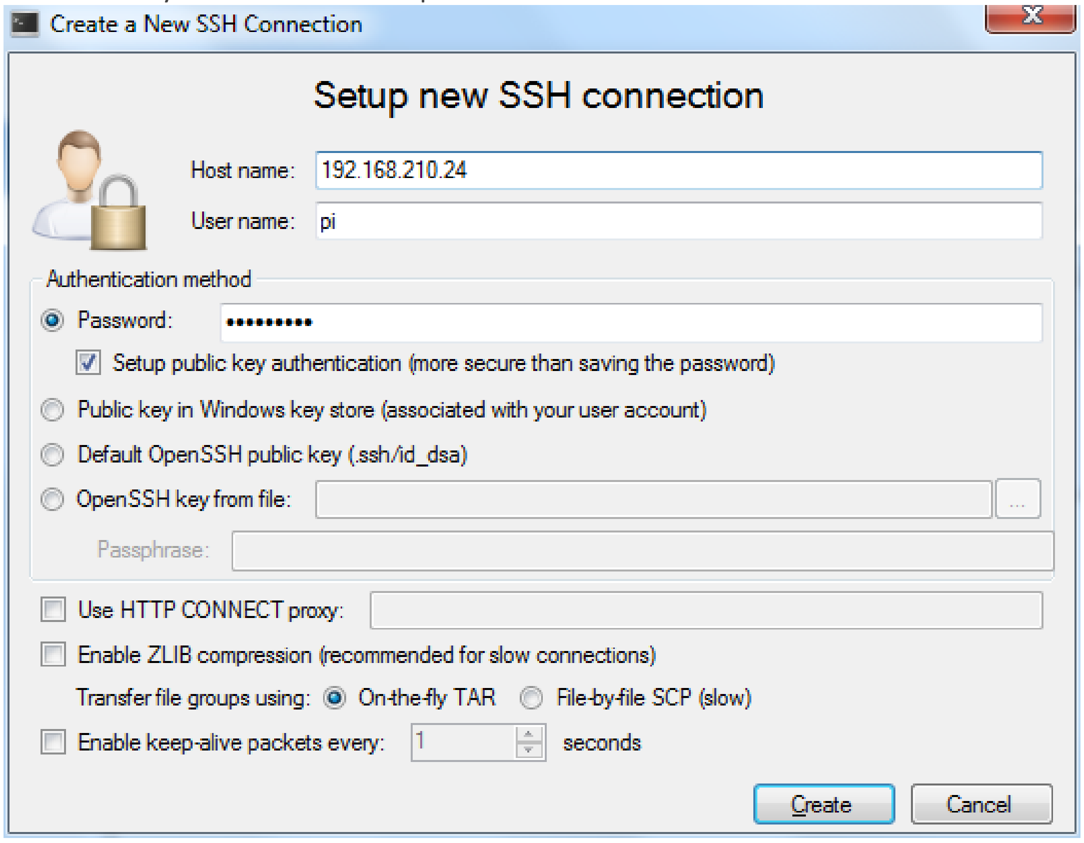
( ) Install for current user only (●) Install for all users

Downloading toolchain...

If you have not created any projects before, in “Deployment folder” select “Create a new SSH connection”.


New Linux Project
New Linux Project
Project Type
Linux computer selection
Build the project under Linux over network
Build the project locally with a cross-compiler
Cross-toolchain:
Deployment computer:
pi@169.254.0.1 (SSH)
pi@192.168.1.30 (SSH)
Create a new SSH connection...
Define a new global alias...
Create a new Vagrant VM...
Deployment folder:
Automatically test co
The following scheme illustrates the involved computers and connections between them:
gdbserver
OL008208
a Linux computer
Synchronize headers and libraries
Please select a deployment computer
< Previous
Next >
Finish
Cancel

4. Provide the IP address of your Raspberry Pi; write **192.168.210.X** where **X** is written on the board. Enter "pi" as user name and "raspberry" as password. It is recommended to check the "setup public key" checkbox, so that VisualGDB will automatically generate a public/private key pair, store it in your Windows account's key container and setup the remote Linux machine to use it.



Create a New SSH Connection

Setup new SSH connection

Host name: 192.168.210.24

User name: pi

Authentication method

[✓ Password:] ••••••••••

[✓] Setup public key authentication (more secure than saving the password)

[ ] Public key in Windows key store (associated with your user account)

[ ] Default OpenSSH public key (.ssh/id_dsa)

[ ] OpenSSH key from file:

Passphrase:

[ ] Use HTTP CONNECT proxy:

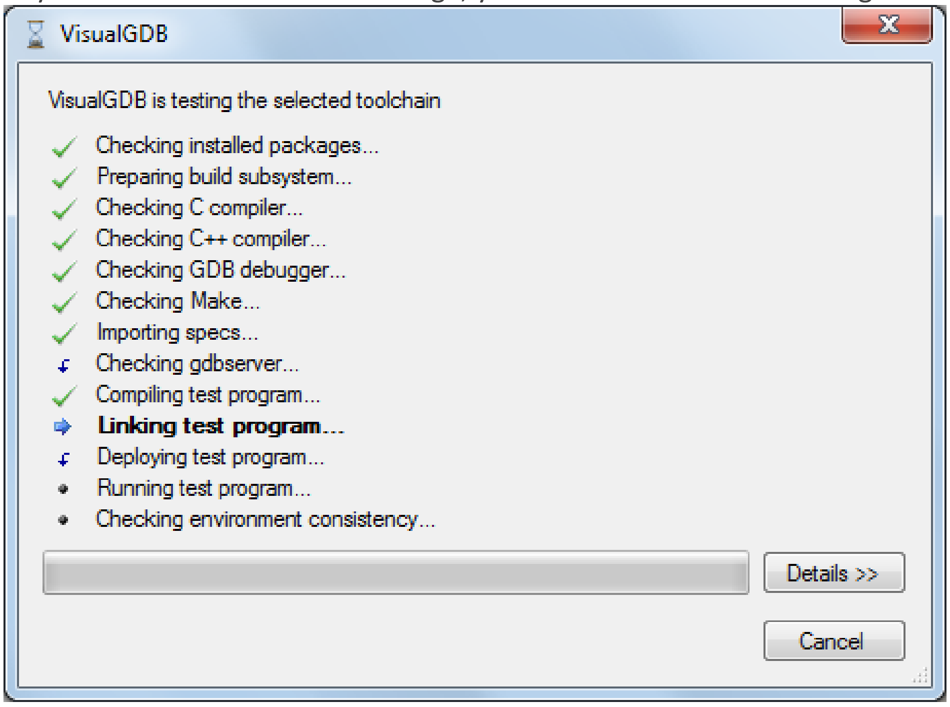
[ ] Enable ZLIB compression (recommended for slow connections)

Transfer file groups using: [✓ On-the-fly TAR] File-by-file SCP (slow)

[ ] Enable keep-alive packets every: 1 seconds

If you don't enable public key authentication, VisualGDB will remember your password for this connection. The stored passwords are encrypted using a key stored in your Windows account. Thus, the password will only be available once you login using your Windows account.

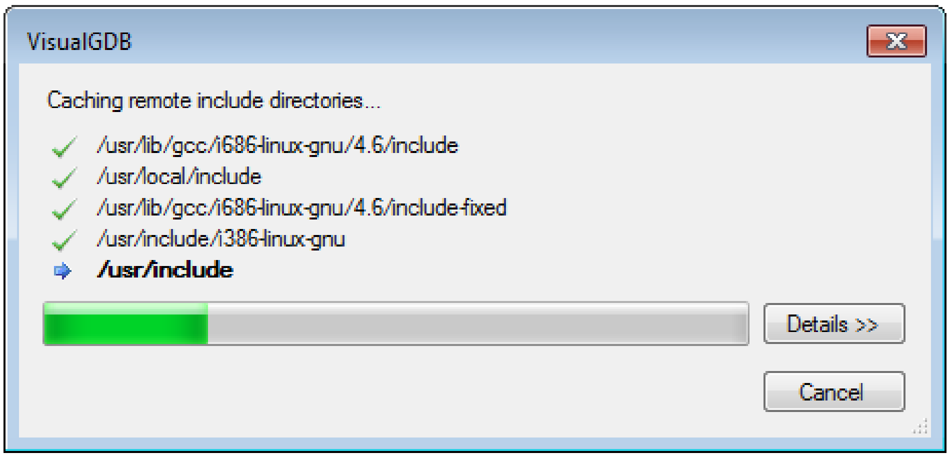
5. When you press “Next”, VisualGDB will test your toolchain by trying to compile and run a trivial program. If any errors are detected at this stage, you will see a detailed error log with further information.



VisualGDB is testing the selected toolchain

✓ Checking installed packages...
✓ Preparing build subsystem...
✓ Checking C compiler...
✓ Checking C++ compiler...
✓ Checking GDB debugger...
✓ Checking Make...
✓ Importing specs...
Checking gdbserver...
✓ Compiling test program...
Linking test program...
Deploying test program...
● Running test program...
● Checking environment consistency...

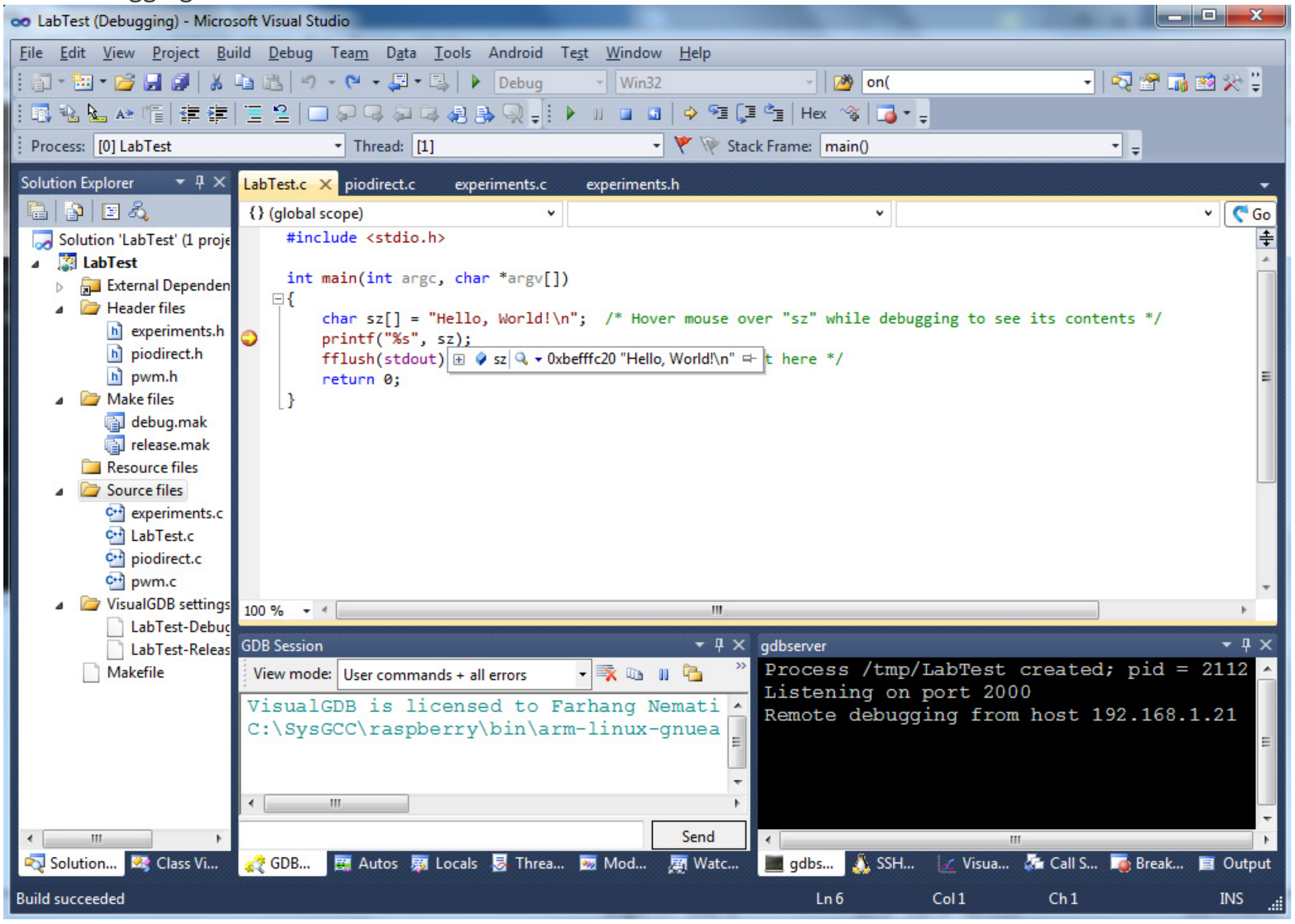
6. Press Finish to complete the wizard. If you are setting up your first project using this Raspberry Pi, VisualGDB will make local caches of the include directories to make them available through IntelliSense.



Caching remote include directories...

✓ /usr/lib/gcc/i686-linux-gnu/4.6/include
✓ /usr/local/include
✓ /usr/lib/gcc/i686-linux-gnu/4.6/include-fixed
✓ /usr/include/i386-linux-gnu
/usr/include

7. The project has been created. You can build your project now. Set a breakpoint on the “printf()” line and start debugging with F5.



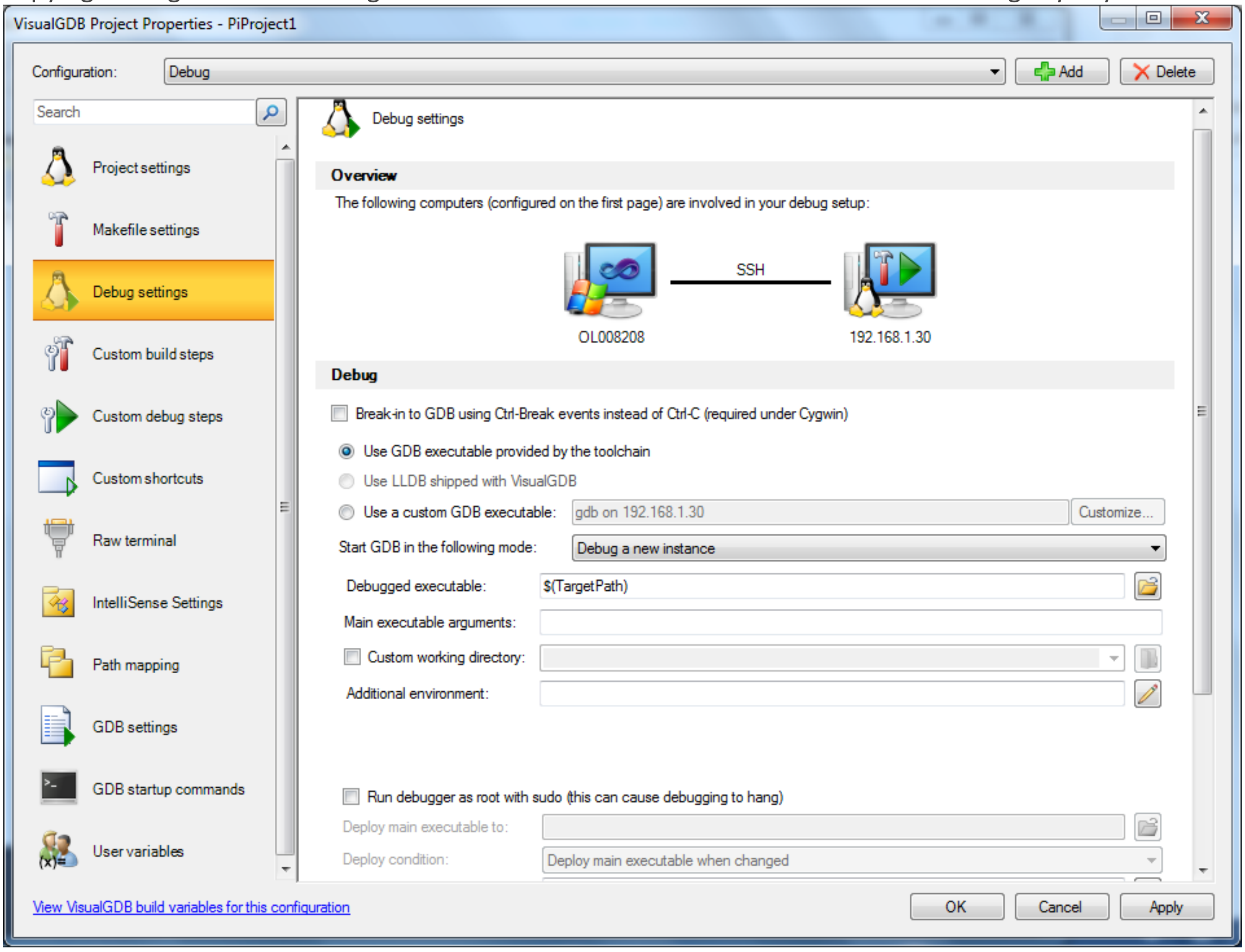
```
#include <stdio.h>

int main(int argc, char *argv[])
{
    char sz[] = "Hello, World!\n";  /* Hover mouse over "sz" while debugging to see its contents */
    printf("%s", sz);
    fflush(stdout)  sz  0xbefffc20 "Hello, World!\n"  t here */
    return 0;
}
```

```
VisualGDB is licensed to Farhang Nemati
C:\SysGCC\raspberry\bin\arm-linux-gnuea
```

```
Process /tmp/LabTest created; pid = 2112
Listening on port 2000
Remote debugging from host 192.168.1.21
```

8. You can always change various debugging settings by right-clicking at the project in Solution Explorer and selecting "VisualGDB Project Properties". The properties window allows changing various settings and also copying settings between configurations. Use the search field on the left to find settings by keywords:


VisualGDB Project Properties - PiProject1

Configuration: Debug

Debug settings

Overview

The following computers (configured on the first page) are involved in your debug setup:

Debug

- [ ] Break-in to GDB using Ctrl-Break events instead of Ctrl-C (required under Cygwin)
- [✓ Use GDB executable provided by the toolchain]
- Use LLDB shipped with VisualGDB
- Use a custom GDB executable: gdb on 192.168.1.30

Start GDB in the following mode: Debug a new instance

Debugged executable: $(TargetPath)

Main executable arguments:

- [ ] Custom working directory:

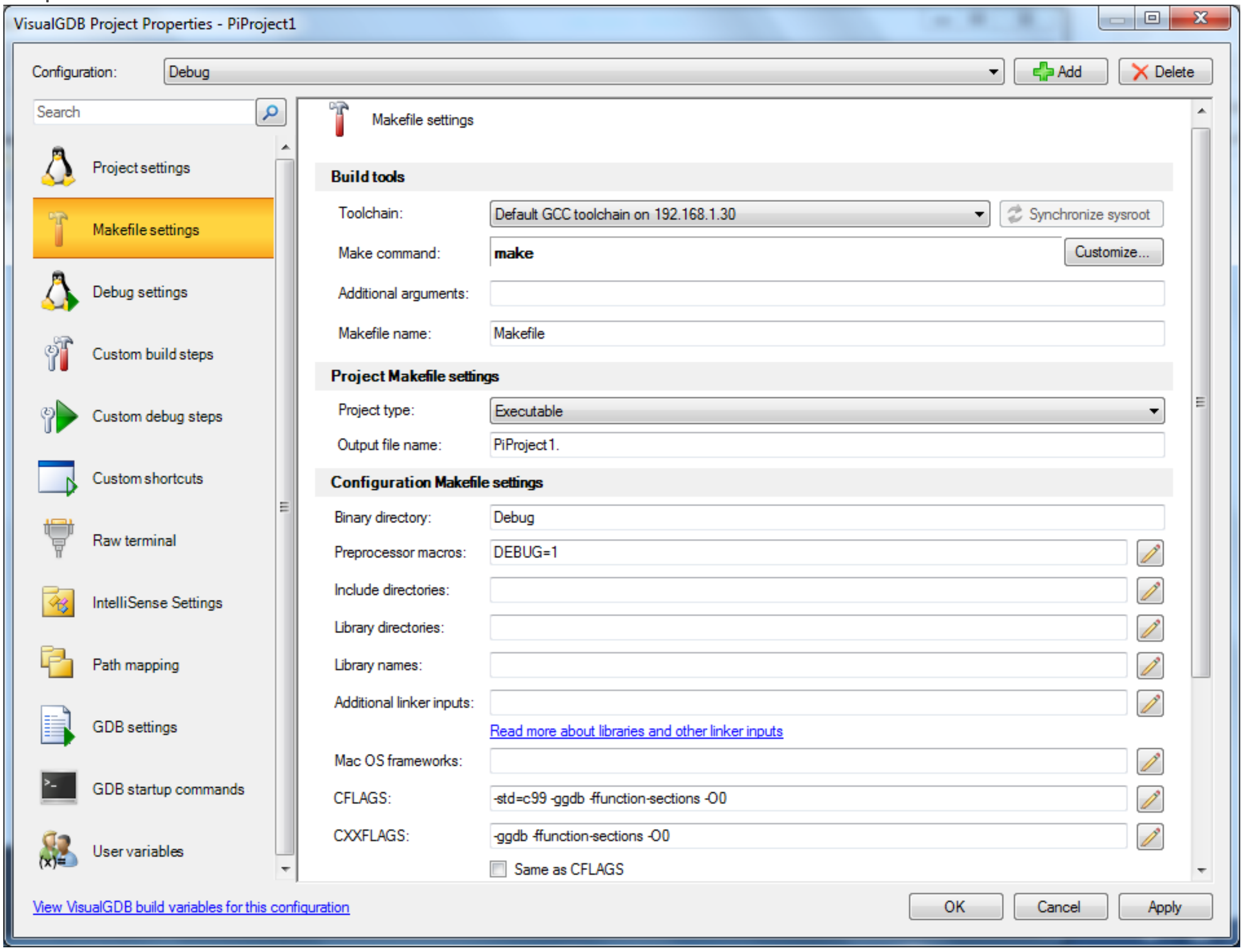
Additional environment:

- [ ] Run debugger as root with sudo (this can cause debugging to hang)

Deploy main executable to:

Deploy condition: Deploy main executable when changed

9. VisualGDB uses GNU make to build your project. You can easily customize various GCC settings, such as additional include directories or compiler flags, on the Makefile Settings page of VisualGDB Project Properties:



Configuration: Debug

Makefile settings

Build tools

Toolchain:	Default GCC toolchain on 192.168.1.30
Make command:	make
Additional arguments:	
Makefile name:	Makefile

Project Makefile settings

Project type:	Executable
Output file name:	PiProject1.

Configuration Makefile settings

Binary directory:	Debug
Preprocessor macros:	DEBUG=1
Include directories:	
Library directories:	
Library names:	
Additional linker inputs:	
Mac OS frameworks:	
CFLAGS:	-std=c99 -ggdb -ffunction-sections -O0
CXXFLAGS:	-ggdb -ffunction-sections -O0

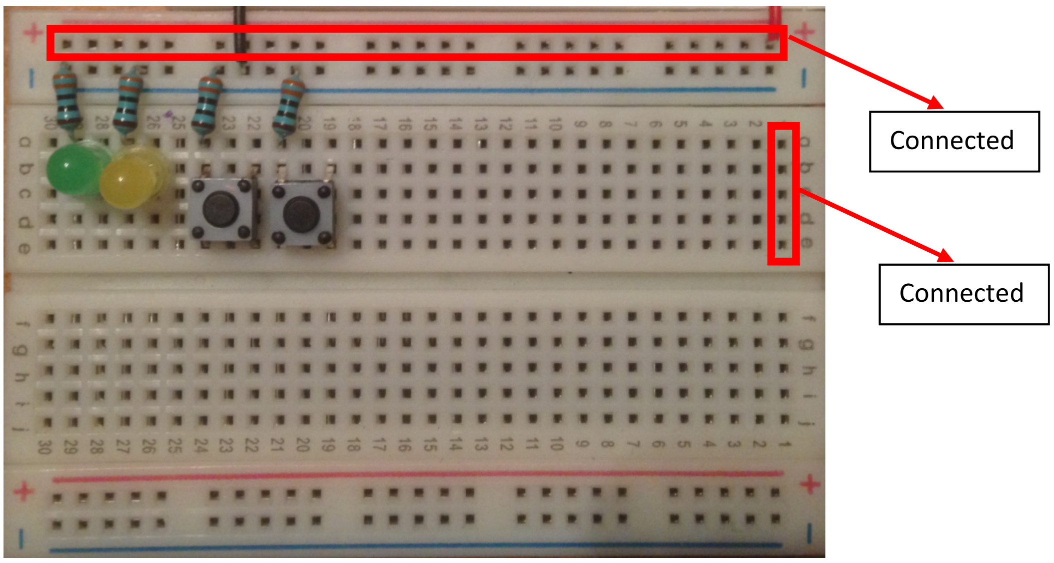
☐ Same as CFLAGS

Once you click “Apply” or “OK”, VisualGDB will automatically check the new settings and update IntelliSense configuration accordingly.

B. Blinking LED

In previous section you learned how to create a project and run/debug it on the Pi using Visual Studio and VisualGDB. Now you are going to write a program that makes a LED blink. The negative pin of the LED has to be connected to GDN and its positive pin has to be connected to a GPIO pin which will be set as “output” pin. To ease working with the Pi we have attached a breadboard next to it. The GND pins are connected to GND of Pi and the positive pins of the breadboard are connected to 3.3V pin on GPIO header of the Pi.

A breadboard makes it easy and convenient to connect devices to a Raspberry Pi. A Breadboard usually consists of 4 parts (called strips); 2 power supply strips on the sides (called Bus strips), and 2 parts in the middle (the main areas) to connect devices. A bus strip is used to supply power to electronic devices connected to the Pi. All pins along each positive column (red line) and negative column (blue line) are interconnected. On each of the middle parts the pins on each row are connected. However the 2 parts in middle are not connected to each other.



1. We have already installed 2 LEDs and 2 buttons (You will practice with the buttons in the next lab) on the breadboard next to the Pi. The negative pins of the LEDs are already connected to the GND. You need to connect the positive pin of a LED that you want to use in this lab to a GPIO pin (preferably connect it to a pin with a number as its name, e.g., 17). Use a Male-Male jump wire for the connection. Ask the lab assistant for helping to connect the LED if you are not sure how to do it. Please notice that failure in connecting and working with pins can destroy them.

2. Create a project like the one for “Hello World!” program.

3. Download *piodirect.h* and *piodirect.c* from Blackboard (at “Kursmaterial/Labs/Helper Code”) and add them to your project.

4. In the program create an output GPIO pin with output option. Use the same pin name as parameter for the create function that you have connected to the LED to. For example if you have connected the LED to pin 17 you have to enter _17 as pin in function create(). When the output pin is high the LED will turn on and when it is low it will turn off.

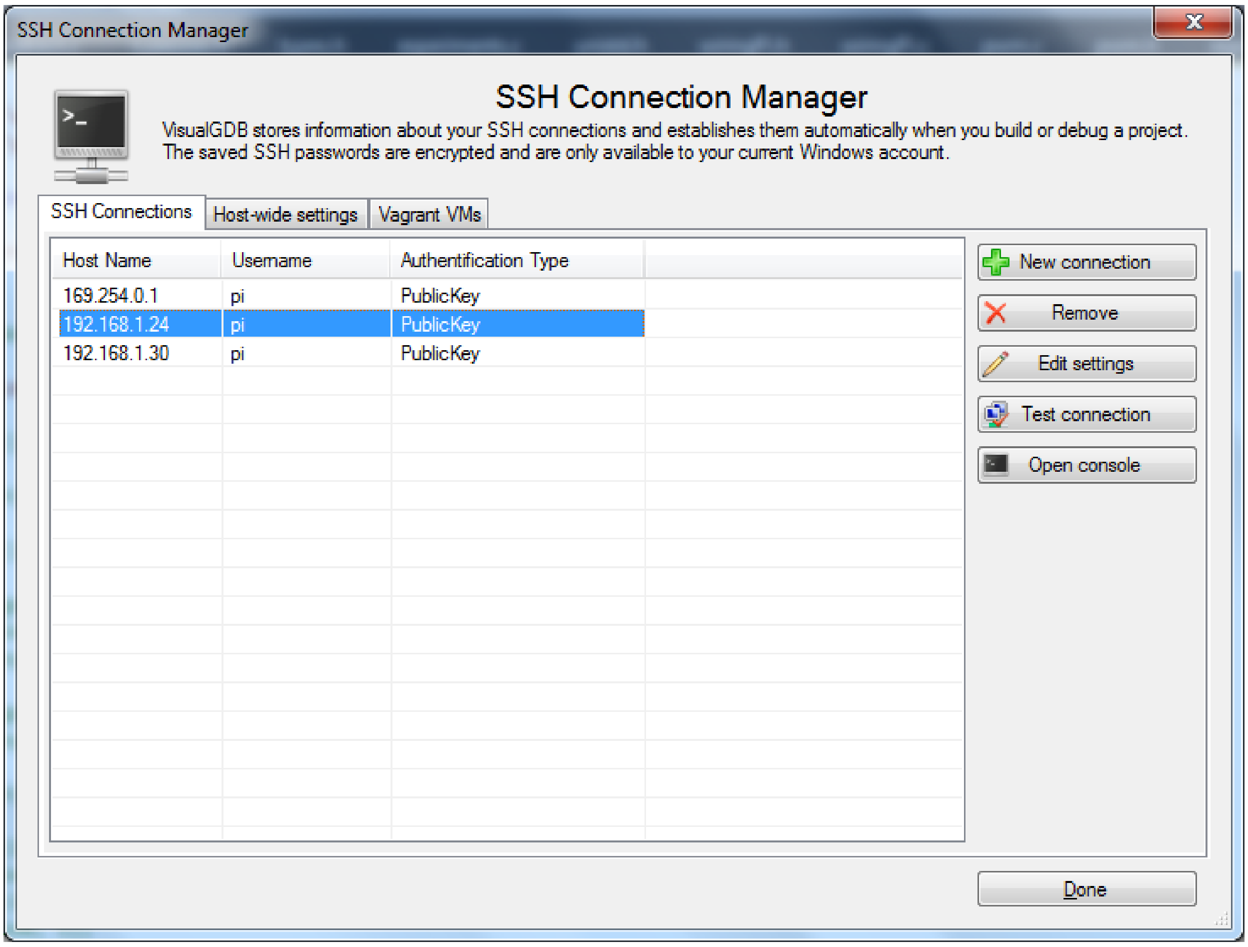
5. The program has to execute in following sequence 10 times:
   Turns on the LED, waits for 1 second, turns it off, wait for 1 second.
   Hint: use function *sleep(sec)* to delay for *sec* seconds.

6. Don’t forget to call function destroy() on the pin before exiting the program.

Running the Program

To run the Blinking LED program on the Pi, you have to first connect to the Pi by opening a terminal. Perform the following steps:

1. In Visual Studio select "Tools->SSH Host Manager".
2. In the opened window select the host name of the Pi you are working with (for example 192.168.210.12).



VisualGDB stores information about your SSH connections and establishes them automatically when you build or debug a project.
The saved SSH passwords are encrypted and are only available to your current Windows account.

Host Name	Username	Authentification Type
169.254.0.1	pi	PublicKey
192.168.1.24	pi	PublicKey
192.168.1.30	pi	PublicKey

3. Push the "Open console" button.
4. Change to the folder that your program is deployed in (the default is /tmp if you didn't change it).
5. To access GPIOs the program has to run as root (use *sudo*). Assuming that you have given name "IOPrj" to your project, in the folder use the following command:

```
pi@navio-rpi /tmp $ sudo ./IOPrj
```

Examination

To pass the lab hand in your code (only in Blackboard!) and demonstrate the program for the lab assistant in one of the lab sessions. The lab assistant may ask you questions related to lab task.