Contents

Building Pulse Counter 1
Visual Studio 2013 1
Boost Build V2 system 1
Compiling Boost Library 1
Boost Library Naming 2
Deploying Library for use 3
Wiring Pi 3

Compilers

Install the gnu tool chain for the raspberry pi,
http://sysprogs.com/files/gnutoolchains/raspberry/raspberry-gcc4.9.2-r2.exe

The destination folder needs to be: C:\SysGCC\Raspberry\4.9.2

Visual studio 2013 is required if you want to debug/test the code on windows. If you do not have visual studio 2013 installed then comment out the visual studio configuration (using msvc) in:

PulseCounter\boost\boost-build-2014.10\share\boost-build\user-config.jam

If you do have it installed but not on C drive then you will need to modify the above file.

Building Pulse Counter

Visual Studio 2013

Visual studio files are included for easy compilation, debugging and testing on windows.

Boost Build V2 system

The boost build system compiles the program for both the arm compiler and visual studio environment.

The generated files are in the bin directory.

In the PulseCounter\PulseCounter directory, run the build_arm.bat batch file. This should use boost build to build the PulseCounter binary for the raspberry pi. You can the transfer this file to the pi to run.

A build_win32.bat file exists to build the application for windows using boost build as well.

Compiling Boost Library

If you want to use a different compiler then these tips should allow you to build the boost libraries with the correct swtiches.

Due to long path issues with arm compiler, the Boost library needs to be moved to the top of a drive so the paths are not too long, otherwise some of the compilations fail to generate .o files.

Add your compiler to: …\boost-build-2014.10\share\boost-build\user-config.jam

Compilers are already added for gcc-4.9.2(arm) and vs2013

Unpack boost to a c:\boostsrc

Configure your compiler environment in the PATH variable.

Follow example for gcc-4.9.1 and msvc in "BoostBuildEnvironment.bat"

Arm:

In order to compile for the Arm platform on windows we need to provide a few more command line switches to get it going, eg. target-os and threadapi

When your environment (boost build and compiler paths) is configured you can execute the following (change toolset to yours):

Example Gcc-4.9.1

b2 --build-dir=release --build-type=complete --without-mpi --without-python --without-context --without-coroutine --without-mpi --without-test --without-graph --without-graph_parallel target-os=linux ***toolset***=gcc-4.9.1 threadapi=pthread stage

Example VS2013

b2 --build-dir=release link=static --build-type=complete stage --without-mpi --without-python toolset=msvc-12

Boost Library Naming

In order to choose the right binary for your build configuration you need to know how Boost binaries are named. Each library filename is composed of a common sequence of elements that describe how it was built. For example, `libboost_regex-vc71-mt-d-1_34.lib` can be broken down into the following elements:

`lib`
: *Prefix*: except on Microsoft Windows, every Boost library name begins with this string. On Windows, only ordinary static libraries use the `lib` prefix; import libraries and DLLs do not.[4]

`boost_regex`
: *Library name*: all boost library filenames begin with `boost_`.

`-vc71`
: *Toolset tag*: identifies the toolset and version used to build the binary.

`-mt`
: *Threading tag*: indicates that the library was built with multithreading support enabled. Libraries built without multithreading support can be identified by the absence of `-mt`.

`-d`

ABI tag: encodes details that affect the library's interoperability with other compiled code. For each such feature, a single letter is added to the tag:

Key	Use this lib
s	linking statically to the C++ standard library and compiler runtime support libraries.
g	using debug versions of the standard and runtime support libraries.
y	using a special debug build of Python.
d	building a debug version of your code.[5]
p	using the STLPort standard library rather than the default one supplied with your compiler.

For example, if you build a debug version of your code for use with debug versions of the static runtime library and the STLPort standard library in “native iostreams” mode, the tag would be: `-sgdpn`. If none of the above apply, the ABI tag is ommitted.

`-1_34`

Version tag: the full Boost release number, with periods replaced by underscores. For example, version 1.31.1 would be tagged as "-1_31_1".

`.lib`

Extension: determined according to the operating system's usual convention. On most unix-style platforms the extensions are `.a` and `.so` for static libraries (archives) and shared libraries, respectively. On Windows, `.dll` indicates a shared library and `.lib` indicates a static or import library. Where supported by toolsets on unix variants, a full version extension is added (e.g. ".so.1.34") and a symbolic link to the library file, named without the trailing version number, will also be created.

Deploying Library for use

The libraries will exist in the stage/lib structure.

The headers will be in the boost directory.

Copy the newly compiled boost libraries (headers already exist for other compiled libraries) to the deployment area. Make sure the libs are in their own lib area for the compiler and the libraries are named in the same manner as the libraries for the other compilers.

Wiring Pi

Cd wiringpi\wiringpi

Makesure compiler is in path, should contain make.exe as well.

Modify Makefile like:

```
CC      = C:\SysGCC\raspberry\bin\arm-linux-gnueabihf-g++.exe
AR      = C:\SysGCC\raspberry\bin\arm-linux-gnueabihf-ar.exe
RANLIB  = C:\SysGCC\raspberry\bin\arm-linux-gnueabihf-gcc-ranlib.exe
```

Then;

```
make static
```