EOSAL library

Operating system and hardware abstraction layer.

11.9.2019/Pekka Lehtikoski

Table of Contents

1. Introduction....2
1.1 Getting started with the library....2
1.2 IOCOM in GitHub....2
1.3 Using vmware virtual machines for development....2
1.3.1 Using USB devices from virtual machine....3
1.3.2 Network considerations....3
1.4 Micro-controller trends....4
1.5 MIT license....4
2. Metals – micro-controller hardwares....6
3. Build tools, debuggers, IDEs....7
3.1 Linux build notes....7
3.1.1 QT creator is good....7
3.2 Linux build environment....7
3.2.1 To set code root directory and variable only for current shell:....7
3.2.2 To set it permanently, and system wide (all users, all processes)....7
3.2.3 Helpful scripts....7
3.4 Windows build notes....8
3.4.1 Build folder organization and .sln files....8
3.4.2 Microsoft Visual Studio 2019 pitfalls (27.6.2019)....8
3.5 Windows build environment....8
3.5.1 Helpful batch files....10
3.6 Building OpenSSL on Windows VS 2019....10
3.6.1 Download OpenSSL source archive....10
3.6.2 Extract OpenSSL sources into a specific path....11
3.6.3 Install Perl and NASM....11
3.6.4 Configure OpenSSL....11
3.6.5 Building OpenSSL....11
3.6.6 Copy OpenSSL libraries to use....12
3.6.7 Do the same for the 32 bit build....12
3.7 Arduino build notes....12
3.7.1 Building Arduino libraries....12
3.7.2 Bringing the libraries into Arduino IDE....13
3.8 Atollic True Studio notes....13
3.8.1 What is great about Atollic True Studio and Cube....13
3.8.2 What is not so great....13
3.8.3 Static libraries and settings....14

1. Introduction

190911, updated 11.9.2019/pekka

The operating system abstraction layer, eosal, provides the separation of powers: hardware and software. The eosal wraps operating system and hardware dependent functionality as function call interface. This interface is is similar for all platforms: Purpose of the operating system abstraction is to separate platform dependent code from the bulk of the library. Thus the IOCOM library can run on multiple platforms, like Windows, Linux and several micro-controllers, and be ported to new systems.

1.1 Getting started with the library

190502, updated 16.7.2019/pekka

I think the best way to get started using the library is to try it out: Glance trough introduction chapter in this document. Check it out from GIT repository. Compile it under Linux or Windows and run/look trough the code examples.

1.2 IOCOM in GitHub

180728, updated 5.9.2019/pekka

The library code is stored in:
https://github.com/iocafe/iocom

The IOCOM library relies on the operating system abstraction layer:
https://github.com/iocafe/eosal

I recommend cloning the git repositories in "/coderoot/iocom" and "/coderoot/eosal" directories, or "c:\coderoot\iocom" and "c:\coderoot\eosal" on Windows. It is possible to use other paths as well, but has not been tested yet, and requires environment variable settings.

Hint: GitKraken is very good graphic front end to GIT. I specifically like how it shows branches and merges of the development tree. It is free for non-commercial use and inexpensive for commercial use (individual license $30/year at time of writing).

See iocafe.org for ready development virtual machines, etc. Or email pekka.lehtikoski@gmail.com

1.3 Using vmware virtual machines for development

190828, updated 28.8.2019/pekka

There are good reasons to use virtual machines in product development. Ready development environment with all tools, settings and repositories can be copied from a developer to another. These can be prepared separately for different operating systems, micro controllers or development tool sets. It is easy to get started or get a tested reference to compare to when setting up your own development environment. These things are great, but there are downsides. USB devices like JTAG/ST-link and serial ports need to be mapped to virtual machine. To test a micro controller communication with PC code, network needs to be mapped, etc. Virtual machines are large files, often tens of gigabytes. Copying these over Internet can be problematic and one needs considerable amount of fast reliable hard disc space to store these. High RAM requirements may also be a problem on low end computers.

I have found vmware to be better for this purpose than virtual box. The vmware workstation-player can be downloaded from vmware's web site. It is free for non commercial hobbyist use, but license needs to be bought to use it for work. At the time of writing this, the license price is around $150 USD.

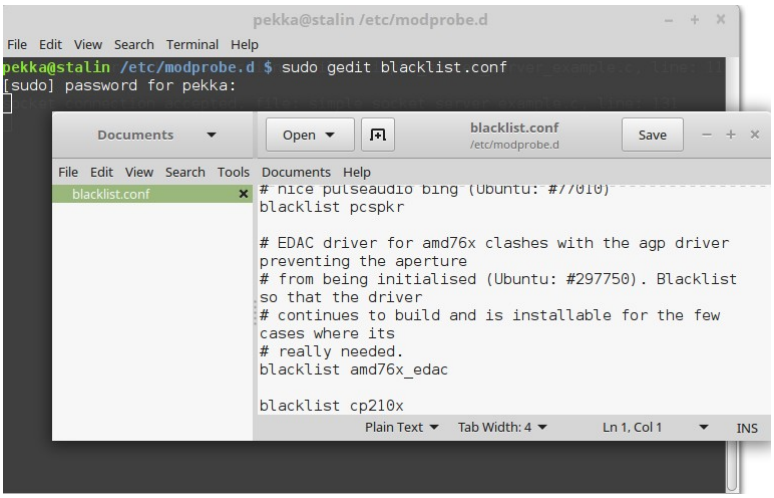
1.3.1 Using USB devices from virtual machine

ST-Link, JTAG and serial ports are nowadays typically USB devices for the development computer. To use these from a virtual machine, these need to be connected to the virtual machine, not to the host computer's operating system.

Virtual machines run typically Linux operating system. Windows is more problematic because license would need to be bought separately for each copy of virtual machine and this makes it impractical to use windows virtual machines for sharing development environments.

In Linux, the development tools run mostly on local user accounts, and serial ports require root privileges by default. To allow anyone to use /dev/ttyUSB0, etc:

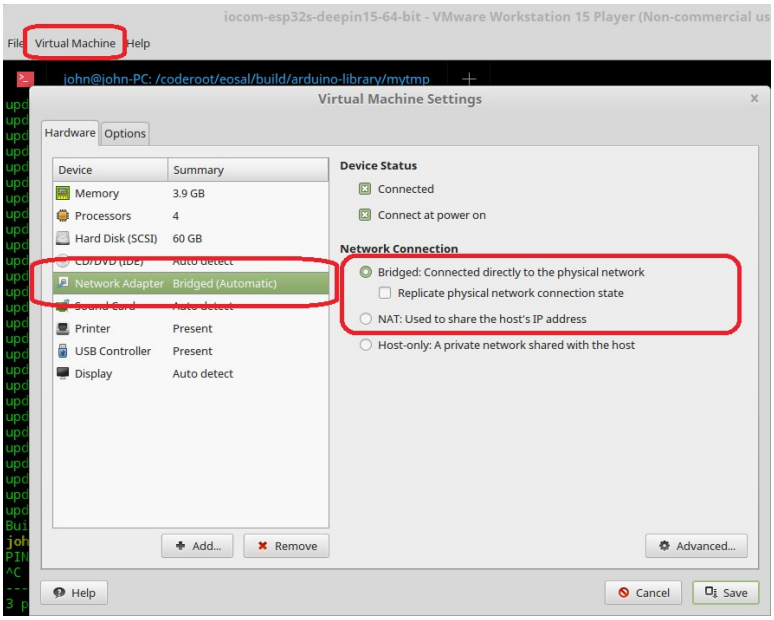
If host computer is also Linux, sometimes it is necessary manually to black list a driver in host operating system to prevent it being loaded. In picture below I have disabled serial port driver cp210xx from the host operating system.



1.3.2 Network considerations

There are basically there ways to approach this.

- Add second network using USB device directly to virtual machine. Map the USB Ethernet or Wi-Fi adapter into virtual machine and just configure it these. Use that network for testing the micro controllers.
- The host's network (either wired or wireless) can also be used to test the devices. If a micro controller is server and software within the virtual machine is client, not much configuration is needed NAT can be used. NAT means that virtual machine shares host's IP address. NAT is especially nice if you use VPN on host computer to connect to corporate network, the virtual machine can share this connection.
- If working other way around where the device is client and it needs to connect to software in virtual machine, set up virtual machine network as "bridged". This means that the virtual machine will appear as completely separate computer from the host computer. Then the virtual machine should pick up network from DHCP, or in none available configure the virtual machine network manually.



1.4 Micro-controller trends

190905, updated 5.9.2019/pekka

- C/C++, mostly plain vanilla C. gcc is dominant and always good + some others are also fine, plain standard C needed.
- Arduino library derivatives like stm32duino and teensyduino simplify and make apps more portable.
- FreeRTOS and RT-Thread (later not tested by me) - useful for some apps - a necessity for multi-core micro-controllers.
- LWIP or uIP Ethernet stack can run on micro-controllers, smart Ethernet chips like WizNET have inbuilt stack and small API lib for micro-controller.
- Use standard TLS - comes as dev library for many micro-controller environments or development tools, OpenSSL to implement for Linux/Windows.
- ARM32 architecture dominant – one assembler to know - one set of build tools compilers - very scalable, from $1 to $20.
- Using “Raspberry PI style” Linux devices is an option. Read only SD file system is required for reliable operation and adds to complexity.

1.5 MIT license

190625, updated 27.6.2019/pekka

Copyright (c) 2019 Pekka Lehtikoski

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT

NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

2. Metals – micro-controller hardwares

190911, updated 11.9.2019/pekka

X...

3. Build tools, debuggers, IDEs

190911, updated 11.9.2019/pekka
X...

3.1 Linux build notes

190611, updated 16.7.2019/pekka
Build is cmake based, so anything which can work with it should do just fine. The cmake is like preprocessor for make files. One writes build instructions in cmake syntax in CmakeLists.txt file, and cmake will generate the make files for actual build system to use. It can generate many different ones. You can as well build with cmake from command line, or use Eclipse, other IDE, etc. QTCreator is an exception, it can use CmakeLists.txt directly. The cmake version should not make much difference, the projects here do not use the new cmake features.

3.1.1 QT creator is good

I have used QT creator 4.0.1, QT 5.6 and cmake for Linux builds. To build a project, start QT creator and open CmakeLists.txt file from project's root folder. QT creator is easy to set up and use, and works well. I appreciate easy debugging and it pointing out the build errors.

3.2 Linux build environment

notes 10.9.2018/pekka
If source code is placed under different folder than */coderoot*, some environment variables, like E_ROOT, need to be set. Replace */coderoot* in this text with path to your source code directory. Setting these can be helpful even if the code is placed in */coderoot*, because build scripts can be accessed from shell without typing path.

3.2.1 To set code root directory and variable only for current shell:

export E_ROOT="/coderoot"
export E_BUILD_BIN="${E_ROOT}/eosal/build/bin/linux"
export E_BIN="${E_ROOT}/bin/mint18"
export PATH=${PATH}:${E_BUILD_BIN}:${E_BIN}

This sets /coderoot as source code root directory and adds build script folder */coderoot/eosal/build/bin/linux* to and */coderoot/bin/mint18* to PATH. You can copy and paste the commands above to shell, but replace "mint18" with your operating system version string (to find out, see which folders you have in coderoot/bin).

3.2.2 To set it permanently, and system wide (all users, all processes)

Use following command to edit system environment: *sudo -H gedit /etc/environment*

Add */coderoot/eosal/build/bin/linux* and */coderoot/bin/min18* (mint18 = your OS version string) to PATH and add line to set the E_ROOT. Edited */etc/environment* could look like:

PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/coderoot/eosal/build/bin/linux:/coderoot/bin/min18"
E_ROOT="/coderoot"

You need to logout from current user and login again so environment variables changes take place.

3.2.3 Helpful scripts

- e-root.sh - This can be started by full path to e-root.sh script, preceded by dot and space: ".

/coderoot/eosal/build/bin/linux/e-root.sh". The script sets environment variables for the calling shell. If E_ROOT is not set, /coderoot is assumed.
- e-clean.sh – Call git to remove non versioned files under current folder from working copy. For example "cd /coderoot/eosal" followed by "e-clean.sh".

3.3

3.4 Windows build notes

190610, updated 27.6.2019/pekka

So far only used build system for Windows has been Visual Studio 2019. There are many other tools the build for Windows: It should be possible to use CMAKE files also on Windows and build with MinGW/Qtcreator, other Visual Studio version, etc, but these are likely to require minor tinkering with build setup and possibly with code.

3.4.1 Build folder organization and .sln files

Each library and example project directory contains subdirectory named build/vs2019. The <projectname>.sln solution file and the <projectname>.vcxproj is project file. All the other files and folders within vs2019 folder are temporary intermediate build and user setting files. For libraries, the solution file contains only reference library's project file. For example projects, the solution file, in addition to example's project file, lists all library projects needed to build the example. To build, open the solution file .sln in Visual Studio 2019, select if you want 32/64 bit compilation and debug/release build.

3.4.2 Microsoft Visual Studio 2019 pitfalls (27.6.2019)

I have used Visual Studio 2019 for Windows development, mostly because it is new. As previous Microsoft libraries, it has nice debugging. Anyhow to me it does have a few surprising disappointments, since Visual Studio 2015 did not have these illnesses:

1. By default it comes with "automatic code formatting" enabled. This will totally mess up your code readability, and is unsustainable for projects with lot of code, long code life time, multiple environments or multiple developers. This must be disabled before using, instructions from Google.

2. Rebuild when using multiple copies of Visual Studio simultaneously to debug multiple executables at same time doesn't work. If these are based on same libraries, something is always locked in other copy of visual Studio by other and prevents the build. It looks like someone has reported problem that door hinges sometimes give a sound, so Microsoft solved the problem by nailing whole door shut. Workaround to debug two executables is to use 64 bit build for other and 32 bit build for the other.

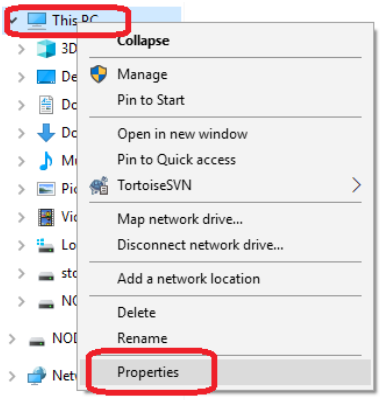
3. Debugging doesn't always work if network connection is not up to Microsoft "specs" (what ever those are). In practice this means that all works when at home, but when traveling I often need to turn off Wifi networking. Otherwise debugging doesn't start, but Visual studio just locks up.

3.5 Windows build environment

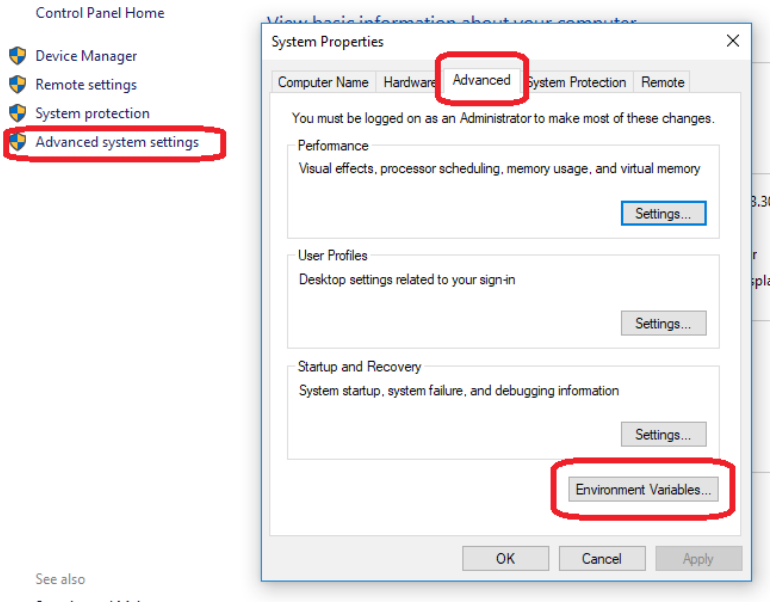
180911 notes 11.9.2018/pekka

If source code is placed under different folder than *c:\coderoot*, some environment variable E_ROOT need to be set. Replace c:*coderoot* in this text with path to your source code directory. Setting these can be helpful even if the code is placed in c:*coderoot*, because build scripts can be accessed from shell without typing path.

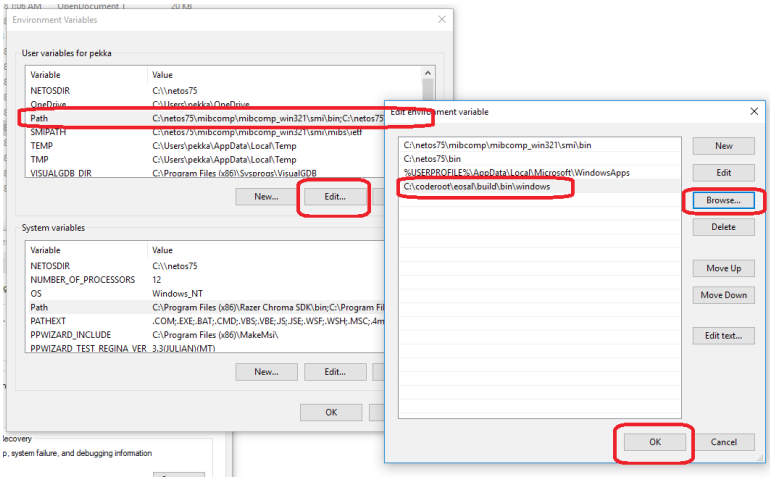
Open "System Properties" dialog from Windows Explorer. Right click "This PC" and select "Properties".



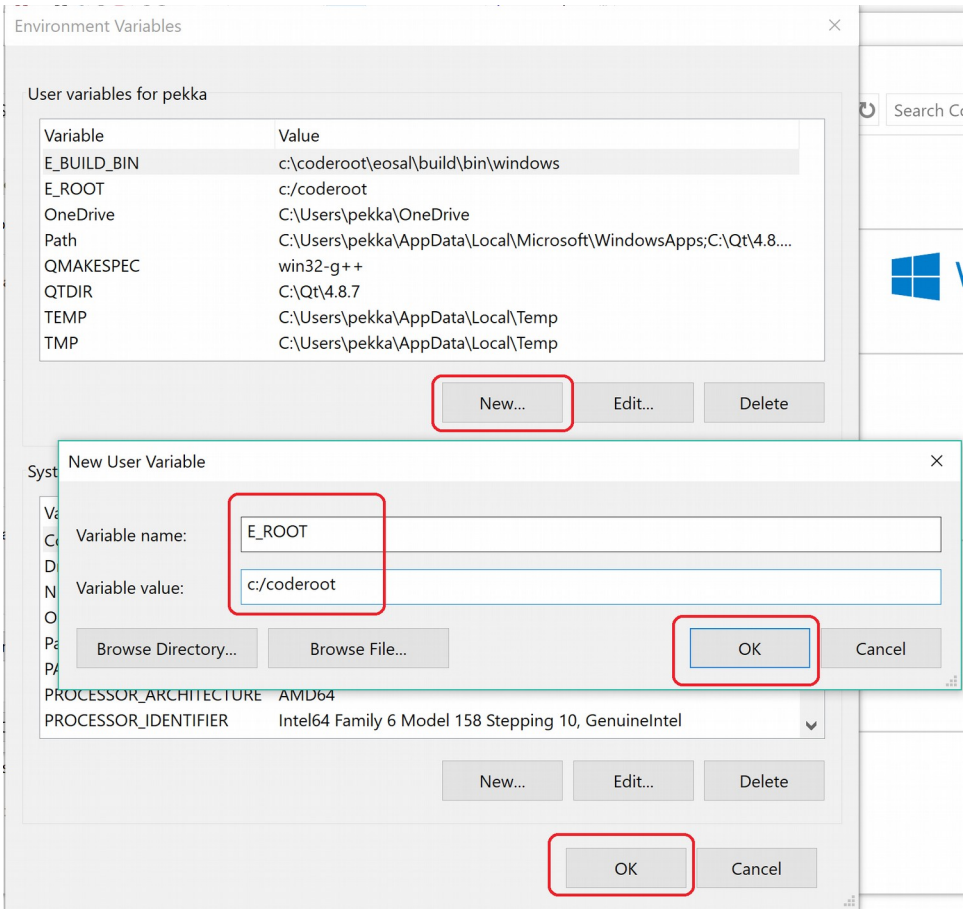
Navigate to “Environment variables” dialog: “Advanced system settings”, “Advanced” tab, “Environment Variables” button.



Add “c:\coderoot\eosal\build\bin\windows” and “c:\coderoot\bin\win64” folders to “Path” environment variable.



Add new “E_ROOT” environment variable with value “c:/coderoot” (notice the forward slash).



It is proper (if not strictly necessary) add also environment variable E_BUILD_BIN with value "c:\coderoot\eosal\build\bin\windows". This example added these only for the current user, but could be set for all users as well.

Notice that you need to sign out and back in or reboot the computer to bring these changes to effect.

3.5.1 Helpful batch files

- e-root.bat - This can be started by full path to e-root.bat file. c:\coderoot\eosal\build\bin\windows\e-root.bat". The batch file sets environment variables for the calling shell. If E_ROOT is not set, c:\coderoot is assumed.
- e-clean.bat – Call git to remove non versioned files under current folder from working copy. For example "cd \coderoot\eosal" followed by "e-clean".
- e-win64-vs2015.bat – Generate solution and work space files for Visual Studio 2015 in current folder. For example "cd \coderoot\eosal\build\cmake" followed by "e-win64-vs2015".

3.6 Building OpenSSL on Windows VS 2019

190909, updated 9.9.2019/pekka

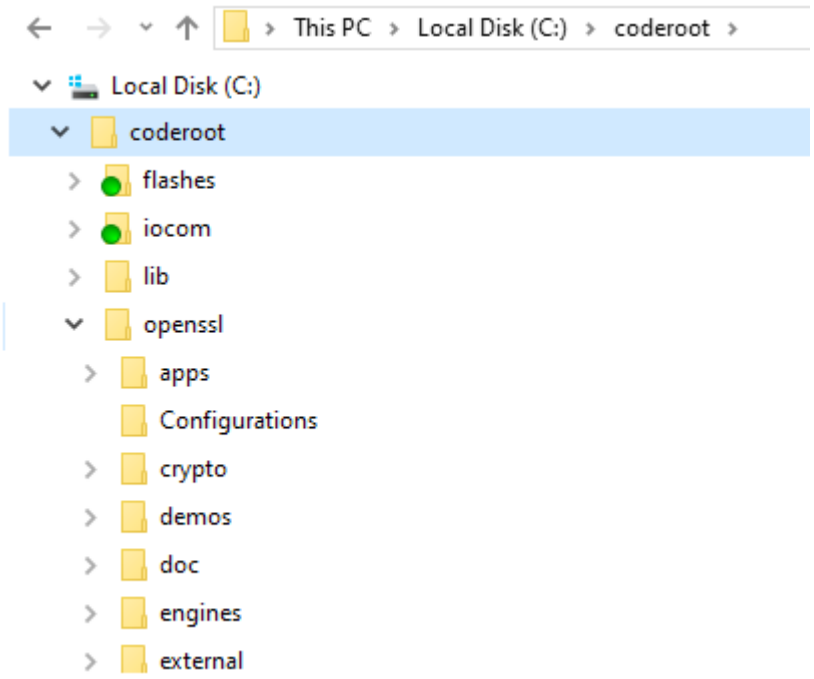
The OpenSLL library is used to provide transport layer security on both Linux and Windows. This paper describes how to take OpenSSL to use for Visual Studio 2019 development. The resulting build will be part of eosal repository, do redoing this is necessary only when taking new OpenSSL version to use or changing OpenSSL configuration used with eosal. Not needed for every computer using OpenSSL.

3.6.1 Download OpenSSL source archive

The OpenSSL sources can be found at https://www.openssl.org/source/. I downloaded the openssl-1.1.1c.tar.gz (prevalent version at 9.9.2019). Something like 7-Zip is needed to extract files from the .gz archive.

3.6.2 Extract OpenSSL sources into a specific path

Extract files from .gz archive into c:\coderoot\openssl folder. Rename openssl directory with version number as plain “openssl” so include paths, etc, work without modification. Resulting file structure should look as in picture below:



3.6.3 Install Perl and NASM

Download and install Perl. I installed “Strawberry Perl 5.30.0.1 (64bit)” from http://strawberryperl.com/.

NASM can be downloaded from https://www.nasm.us/, I used version 2.14.02 with binary installer https://www.nasm.us/pub/nasm/releasebuilds/2.14.02/win64/nasm-2.14.02-installer-x64.exe.

Add C:\Program Files\NASM to PATH.

3.6.4 Configure OpenSSL

For now I have been building with default options, would be better to select only what is needed.

3.6.5 Building OpenSSL

Read the openssl/INSTALL note containing some specific build information common to both the 64 and 32 bit versions. Here we build 64 bit AMD architecture for Windows:

- launch a “x64 Native Tool Command Prompt for VS 2019”
- cd \coderoot\openssl
- nmake clean
- perl Configure -MT -Z7 threads no-deprecated no-shared VC-WIN64A
- perl Configure -MTd -Z7 threads no-deprecated no-shared debug-VC-WIN64A (This could be used to build debug libraries, but those are not used, at least for now)
- nmake

3.6.6 Copy OpenSSL libraries to use

Copy libssl.lib and libcrypto.lib from c:\coderoot\openssl directory to c:\coderoot\eosal\libraries\win64_vs2019. This will move these two libraries to be part of eosal repository, so steps in this paper are not needed in every computer using Visual Studio 2019.

Copy also the “openssl” include directory from c:\coderoot\openssl\include to c:\coderoot\eosal\libraries\include. This will move necessary headers to be part of eosal repository.

3.6.7 Do the same for the 32 bit build

- launch a “x86 Native Tool Command Prompt for VS 2019”
- cd \coderoot\openssl
- nmake clean
- perl Configure -MT -Z7 threads no-deprecated no-shared VC-WIN32
- nmake

Copy libssl.lib and libcrypto.lib from c:\coderoot\openssl directory to c:\coderoot\eosal\libraries\win32_vs2019. Headers were already copied with 64 bit build.

3.7 Arduino build notes

190608, updated 27.6.2019/pekka

Even this is is called Arduino build, the code will not run on UNO, etc. It is simply too heavy and intended for 32 bit micro controllers. I recommend using 32 bit ARM architecture micro-controllers, and my personal favorites of those which I am testing with are STM32F429 Nucleo and Teensy 3.6.

- Teensy 3.6 has really good performance and format that it can be plugged directly into my own PCB. But has two big minuses: lack of debugging support and incomplete LWIP library port.

- STM32F429 Nucleo on the other hand shines where Teensy lacks, but it comes short as development board which doesn’t physically integrate with own electronics nicely. In addition Nucleo cannot be legally reused “as is” as part of commercial project: You have to draw your own PCB and put the STM chip on it. This is something I cannot do at home, but need to order from China.

3.7.1 Building Arduino libraries

An Arduino library is compressed package of source code, not collection of compiled object files. To build eosal (operating system abstraction layer) library.

```
cd /coderoot/eosal/build/arduino-library
mkdir tmp
cd tmp
cmake ..
make
```

Library (or in Arduino’s case .zip file) eosal-master.zip will be created in /coderoot/lib/arduino directory. Remove tmp directory. Then same for iocom library.

```
cd /coderoot/iocom/build/arduino-library
mkdir tmp
cd tmp
cmake ..
make
```

The iocom-master.zip should appear in the same /coderoot/lib/arduino directory.

NOTE: WRITE ABOUT PACKAGES NEEDED. ARDUINO IDE, STMDUINO, TEENSYDUINO VERSIONS. USING TEENSYDUINO SOCKET LIBRARY FOR STM32/WIZ5500. MODIFICATION TO SOCKET LIBRARY. SETTING SERIAL COMMUNICATION TX AND RX TO 256 BYTES.

3.7.2 Bringing the libraries into Arduino IDE

- Delete old version of arduino libraries in use. For example I need to delete eosal-master and iocom-master directories within /home/pekka/Arduino/libraries directory.
- Open an Arduino sketch in Arduino IDE, for example 4_arduinoioboard.ino in /coderoot/iocom/examples/4_arduinoioboard directory.
- Select [Sketch][Include Library][Add .ZIP Library] from Arduino IDE menu and browse to /coderoot/lib/arduino director, select eosal-master.zip and click Ok to bring it in.
- Do the same for the iocom-master.zip.

3.8 Atollic True Studio notes

190714, updated 15.7.2019/pekka

Atollic True Studio is professional eclipse based development environment primarily for STM32 micro-controllers. The IDE is similar to Keil or even QT or Visual Studio, and debugging works very nicely. In addition to IDE, there is Cube. Cube makes assigning micro-controller pins and timing easy (much much easier than reading thick reference manuals and doing it by hand). It generates code to initialize the micro-controller, which can then be used in True Studio project.

Advice: If you are new to programming micro-controllers, starting out cold with building the iocom library with True Studio will be hard. Better to start either with True Studio examples, or even better with STM32duino examples. Way from hello world “blink led”, proceeding step at a time, is likely to yield results faster than time than trying to take it all in with single bite.

3.8.1 What is great about Atollic True Studio and Cube

- Cube is excellent tool for assigning micro-controller pins and timing.
- True Studio IDE has excellent debugging trough IDE. Notice that Nucleo boards come with integrated ST-link USB connector, or if you have board without one, a $6 “ST-Link V2 Mini STM8 STM32 Emulator Downloader” from Amazon works well (connects USB to micro-controller pins).
- Nice professional look and feel. Full functionality without any licensing fees. Experienced IDE users familiar with Eclipse, Kail, QTCreator, Visual Studio, etc. feel quickly at home.
- Integration of LWIP and FreeRTOS to Cube.
- Great ready support and examples for ST Microelectronics ARM32 products.
- Good, although not trouble free, system to keep developers up to date.

3.8.2 What is not so great

- Wants to integrate Cube generated code with application and hardware.
- This easily leads to bloated, non reusable application projects. Extra work is needed to keep IO device application code portable, to separate it from Cube generated files, and to avoid duplicating same Cube and C source files.
- Low abstraction level, chip specific implementations are often needed. For instance Ethernet requires separate implementations for Wiznet and for LWIP, and micro controller chip specific serial port driver code and flash programming. HAL provides some level of portability, but not much. When comparing STMduino/Teensyduino which has APIs for these (others but for flash): This approach allows developer

great control of detail, but also consumes lot of time and requires expertise to get the reliability.
- Much steeper learning curve than STM32duino or TeensyDuino.
- Dependency on Cube versions. Cube and True Studio version updates have caused, and are likely to cause future headaches.
- Cube support for other manufacturers than ST microelectronics practically null. True Studio can still be used for generic ARM builds and should work for debugging.

3.8.3 Static libraries and settings

- Static libraries like iocom, eosal and w5500 have True Studio project in build/atollic folder. This will build the static library in build/atollic/Debug, etc, folder.
- Compiler defines E_OS_metal and STM32L476xx (or other) are set for static libraries. The E_OS_metal indicates that we are compiling for “bare metal” or close to it, anyhow to system without typical operating system (we may still use HAL/CMSIS and FreeRTOS). The STM32L476xx sets micro-controller type. This is needed by HAL.
- Include directories coderoot/eosal, coderoot/iocom allow build with eosal and iocom headers.
- Include directory coderoot/eosal/libraries/WIZnetIOlibrary/Ethernet allows build with WizNET w5500 IO library.
- Include directories related to HAL and CMSIS and come with STM32 Cube. Once support for a specific micro-controller family, lets say STM32F4, is installed within Cube, full copy HAL and CMSIS appears in user’s home repository ~/STM32Cube/Repository/STM32Cube_FW_L4_V1.13.0.