EOSAL library

Operating system and hardware abstraction layer.

11.9.2019/Pekka Lehtikoski

Table of Contents

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

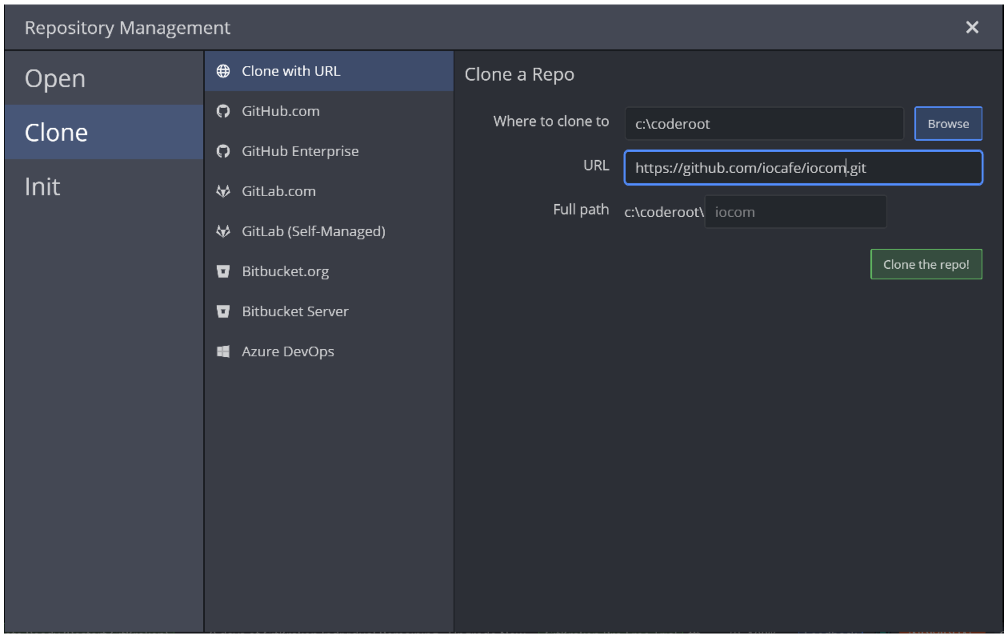
The library code is stored in:
https://github.com/iocafe/iocom

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. Also repository /coderoot/pins will be needed to run example codes (HW independent access to GPIO, ADC/DAC, PWM, etc. We plan eventually to support other root folders than /coderoot, but at least for now using other root folder would require editing a lot of paths and be wasted effort.

Recommendation: GitKraken is 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). GitKraken instructions can be found at https://support.gitkraken.com/start-here/guide/.

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



Where to clone to: c:\coderoot
URL: https://github.com/iocafe/iocom.git
Full path: c:\coderoot\ iocom

Cloning iocom repository with GIT Kraken on Windows

1.3 Using vmware virtual machines for development

190828, updated 14.9.2019/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 Maintain backup of your virtual machine

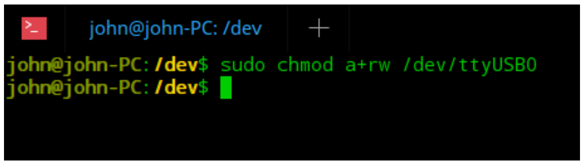
Keep back up of your virtual machine, this is very likely to become handy at some point. When working with a bunch of new tools and code, it is only too easy to install or configure something erroneous. Working backup will then save the day. To create a back up, shut down your virtual machine and use 7-Zip, etc, to compress whole virtual machine folder. I recommend to keep at least two backups: Latest proven stable version, and version currently working on.

1.3.2 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.

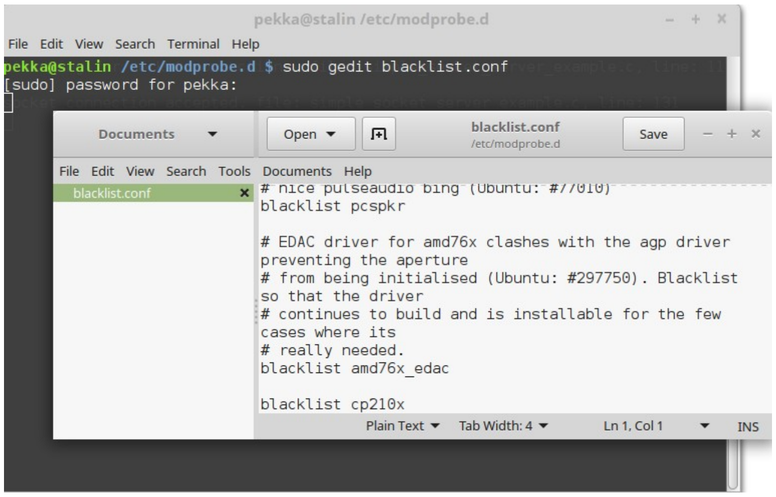
Virtual machines run typically Linux operating system. Windows is more problematic because license would need to be bought separately for each copy of virtual machine and this makes it impractical to use windows virtual machines for sharing development environments.

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:



```
john@john-PC:/dev$ sudo chmod a+rw /dev/ttyUSB0
john@john-PC:/dev$ 
```

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.


pekka@stalin /etc/modprobe.d
File Edit View Search Terminal Help
Documents
Open
blacklist.conf
/etc/modprobe.d
Save
File Edit View Search Tools Documents Help
blacklist.conf
Plain Text
Tab Width: 4
Ln 1, Col 1
INS

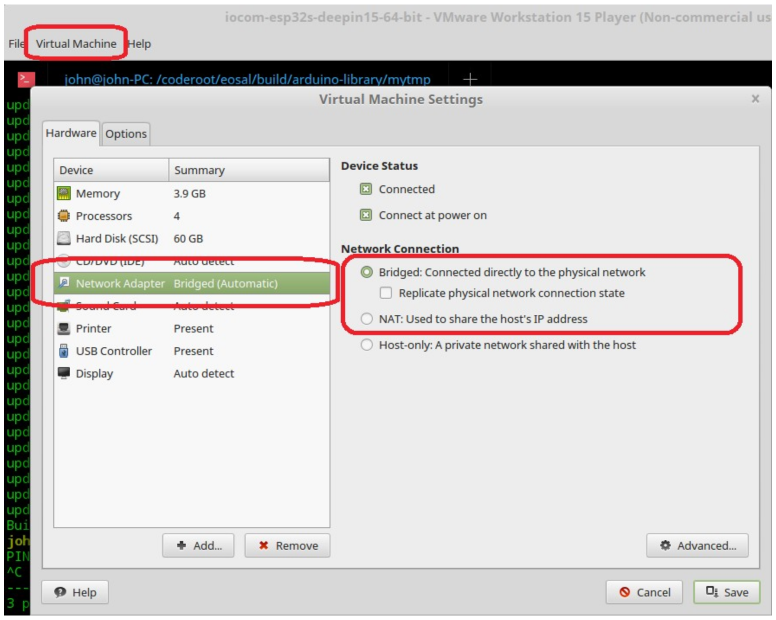
```
pekka@stalin /etc/modprobe.d $ sudo gedit blacklist.conf
[sudo] password for pekka:
```

```
# nice pulseaudio bing (Ubuntu: #77010)
blacklist pcspkr

# EDAC driver for amd76x clashes with the agp driver preventing the aperture
# from being initialised (Ubuntu: #297750). Blacklist so that the driver
# continues to build and is installable for the few cases where its
# really needed.
blacklist amd76x_edac

blacklist cp210x
```

1.3.3 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.


iocom-esp32s-deepin15-64-bit - VMware Workstation 15 Player (Non-commercial us
File
Virtual Machine
Help
john@john-PC: /coderoot/eosal/build/arduino-library/mytmp
Virtual Machine Settings
Hardware
Options
Add...
Remove
Advanced...
Help
Cancel
Save

Device	Summary
Memory	3.9 GB
Processors	4
Hard Disk (SCSI)	60 GB
CD/DVD (IDE)	Auto detect
Network Adapter	Bridged (Automatic)
Sound Card	Auto detect
Printer	Present
USB Controller	Present
Display	Auto detect

Device Status
[✓] Connected
[✓] Connect at power on

Network Connection
[✓] Bridged: Connected directly to the physical network
[ ] Replicate physical network connection state
NAT: Used to share the host's IP address
Host-only: A private network shared with the host

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. Extensions

190921, updated 21.9.2019/pekka
X…

2.1 Persistent parameters – osal_persistent

190921, updated 21.9.2019/pekka
Micro-controllers store persistent board configuration parameters in EEPROM or flash. On Windows/Linux these parameters are usually saved in files within OS file system. Several different implementations are needed because flash and EEPROM use on micro-controllers is usually platform/micro-controller/board dependent.

2.1.1 Persistent storage functions

Include "eosalx.h" to use persistent storage functions. Define OSAL_OSAL_PERSISTENT_SUPPORT is nonzero if persistent storage is available, value 1, 2.. selects the used implementation for the platform.

Initialize persistent storage access. Call this first. prm can be OS_NULL if not needed.

```
void os_persistent_initialze(
    osPersistentParams *prm);
```

Load parameter structure identified by block number from persistent storage. Load all parameters when micro controller starts, not during normal operation. If data cannot be loaded, leaves the block as is. Returned value maxes at block_sz.

```
os_memsz os_persistent_load(
    osPersistentBlockNr block_nr,
    os_uchar *block,
    os_memsz block_sz);
```

Save parameter structure to persistent storage and identigy it by block number.

```
osalStatus os_persistent_save(
    osPersistentBlockNr block_nr,
    os_uchar *block,
    os_memsz block_sz);
```

2.1.2 Parameters are saved and loaded as blocks

Data is saved and loaded as memory blocks. When storing data in flash or EEPROM, we do not need to care about portability of the data format from micro-controller architecture to another. Structure alignment will not change and data will not be copied from big 32 bit MCU to 16 bit, etc. So we can present a set of parameters as C structure:

```
typedef
{
    os_short version;  // Have this always first, even if you would not need it
    os_char ip[MY_IP_SZ];
    os_char subnet[MY_IP_SZ];
    os_char gateway[MY_IP_SZ];
    os_boolean dhcp;
}
MyNetConfParams;
```

Load parameter to C structure as below. If the data cannot be found or is corrupted in storage, the os_persisent_load() leaves the parameter structure unchanged. If it should be zeroed, call os_memclear(&prm, sizeof(prm)) before.

```
#define MY_PRM_BLOCK_NR OS_PBNR_APP_1
MyNetConfParams prm;
os_memsz sz;
sz = os_persistent_load(MY_PRM_BLOCK_NR, &prm, sizeof(prm));
if  (sz == sizeof(prm)) {
    os_console_write(“success, exact version match”);
}
else if (sz > 0)
{
    os_console_write(“success, not same version (may work, may not)”);
}
```

When parameters have changed, we need to save those. Set version number before saving, this is likely to make things easier later on.

```
prm.version = 1;
if (os_persistent_save(MY_PRM_BLOCK_NR, &prm, sizeof(prm))) {
  os_debug_error(“saving my parameters failed”);
}
```

2.1.3 New versions of parameter structure

Often we need to add parameters to existing set, change some array size, etc. We do not want to reconfigure a board at software update. Version number should be incremented when saving and can be checked when loading.

2.1.4 Implementation

Flash and EEPROM hardware use is typically micro-controller specific, and common library functionality can rarely be used. On Arduino there is valuable effort to make this portable, but even with Arduino we cannot necessarily use the portable code: To be specific we cannot use Arduino EEPROM emulation relying on flash with secure flash program updates over TLS.

On raw metal or custom SPI connected EEPROMS we always end up with micro-controller or board specific code.

On Windows or Linux this is usually simple, we can just save board configuration in file system. Except if we have read only file system on Linux.

The "persistent" directory contains persistent storage interface header osal_persistent.h, which declares how the persistent storage access functions look like. The actual implementations are in subdirectories for different platforms, boards and setups.

The subdirectories named after platform, like "arduino", "linux" or "windows" contain either the persistent storage implementations for the platform or include C code from "shared" directory, when a generic implementation can be used. The "metal" subdirectory is used typically with micro-controller without any operating system.

Warning: Do not use micro-controller flash to save any data which changes during normal run time operation, it will eventually burn out the flash (death of the micro controller).

2.2 Serialization and streams

191026, 26.10.2019/pekka

Data is typically a stored in computer’s memory is almost always organized differently that what is needed to save the data to file or transfer it over network connection. For the latter we need a flat stream of bytes. This conversion of data format is called serialization, or stream writer and reader.

- Independent of computer architecture
- Extensible, we can add information and modify types without losing compatibility.
- Compresses to use minimal system resources

Serialization, as are stream implementations, are implemented as eosal extensions and can be included or excluded by compile time defines.

Streams
Files, sockets (TCP or TLS) and serial ports can be treated as streams. To be able to use any of these from serialization code, data is written to/read from stream trough stream interface. There is also stream buffer class which implements the stream interface: This allows serialization in memory.

Integer serialization
When an integer is serialized, number of bytes written to stream depends on integer value, not on integer data type -> Small numbers take less space than large ones. This is computer architecture independent, since generated data has always same byte order and size doesn't depend on data type size.

Float serialization
Floating points are split as two integers, mantissa and exponent. These are serialized as two integers. This makes serialization format independent on internal float presentation (in practice most computers use IEEE 754, but byte and word order do vary).

JSON serialization
The JSON is used as hierarchical and extensible presentation of data. JSON data is normally plain text, and quite inefficient as such. Thus it is routinely compressed as packed binary JSON. Binary JSON is suited for internal data presentation for multiple purposes, like presenting IOCOM data structures, saving parameters, etc. Plain text JSON is good for editing and communicating with systems by other manufacturers.

2.2.1 Integer serialization

191026, 26.10.2019/pekka

Serialization format is packed by integer value. Packed format is type and architecture independent , number of bytes generated depends only on value. Maximum integer size is 64 bits.

Bits in first byte byte NNNNSxxx:

- NNNN number of follower bytes.
- S sign, 0 for positive, 1 for negative.
- xxx Least significant 3 data bits.

Follower bytes hold the remaining data bits.

Required defines:
```
OSAL_SERIALIZE_SUPPORT = 1
```

Serializing an integer
The osal_intser_writer() function packs C integer x into buffer using serialization format. The function returns number of bytes written to the buffer.

```
os_int osal_intser_writer(
    os_char *buf,
    os_long x);
```

Reading an integer
The osal_intser_reader() function converts integer from serialization format into 64 bit C integer x.
The function returns number of bytes read from the buffer.

```
os_int osal_intser_reader(
    os_char *buf,
    os_long *x);
```

2.2.2 Float serialization

191026, 26.10.2019/pekka

Floating points can be split as two integers, mantissa and exponent. Computer's internal floating point presentation doesn't effect to values of mantissa and exponent.

Required defines:
```
OSAL_SERIALIZE_SUPPORT = 1
```

Splitting float or double into mantissa and exponent

To make serialization indepent on computer's floating point presentation, we break a C floating point type into two integers, mantissa and exponent. Then we serialize those.

```
void osal_float2ints(
    os_float x,
    os_long *m,
    os_short *e);

void osal_double2ints(
    os_double x,
    os_long *m,
    os_short *e);
```

Merging mantissa and exponent into float or double

A C floating point type can be constructed from mantissa and exponent:

```
os_boolean osal_ints2float(
    os_float *x,
    os_long m,
    os_short e);

os_boolean osal_ints2double(
    os_double *x,
    os_long m,
    os_short e);
```

2.2.3 JSON serialization

191026, 26.10.2019/pekka

The JSON is used as hierarchical and extensible presentation of data. JSON plain text is routinely compressed as packed binary JSON.

Convert JSON as packed binary

JSON as text has about 50% overhead from text format. To nip this off from data transfers and micro-controller storage, the JSON is packed as architecture independent binary file. Binary JSON is suited for internal data presentation for multiple purposes, like presenting IOCOM data structures, saving parameters, etc. The binary file can be further turned into C code and complied with micro-controller application.

Required defines:
```
OSAL_SERIALIZE_SUPPORT = 1
OSAL_JSON_TEXT_SUPPORT = 1
```

Function:
```
osalStatus osal_compress_json(
    osalStream compressed,
    os_char *json_source,
    os_char *skip_tags,
    os_int flags);
```

Convert packed binary to plain JSON text

To export data, the packed binary can be converted back to plain text JSON. Plain text JSON is good for editing and communicating with systems by other manufacturers.

Required defines:
```
OSAL_SERIALIZE_SUPPORT = 1
OSAL_JSON_TEXT_SUPPORT = 1
```

Function:
```
osalStatus osal_uncompress_json(
    osalStream uncompressed,
    os_char *compressed,
    os_memsz compressed_sz,
    os_int flags);
```

Accessing data in packed binary

Once packed binary JSON is in memory, regardless if it is as “const” C code in micro-controller’s flash or received trough communication, it can be accessed.

Required defines:
```
OSAL_SERIALIZE_SUPPORT = 1
```

Functions:
```
/* Create indexer to access compressed data easily.
 */
osalStatus osal_create_json_indexer(
    osalJsonIndex *jindex,
    os_char *compressed,
    os_memsz compressed_sz,
    os_int flags);

/* Release indexer when no longer in use.
 */
void osal_release_json_indexer(
    osalJsonIndex *jindex);

/* Get next JSON item from the indexer.
 */
osalStatus osal_get_json_item(
    osalJsonIndex *jindex,
    osalJsonItem *item);
```

JSON item data structure:
```
/** Information about single JSON item.
 */
typedef struct osalJsonItem
{
    /** One of: OSAL_JSON_START_BLOCK, OSAL_JSON_END_BLOCK, OSAL_JSON_VALUE_STRING,
        OSAL_JSON_VALUE_INTEGER or OSAL_JSON_VALUE_FLOAT.
     */
    osalJsonElementCode code;

    /** Tag name is name in double quotes before colon..
     */
    const os_char *tag_name;

    /** Recursion level in JSON.
     */
    os_int depth;

    /** Primitive value of the item, for codes OSAL_JSON_VALUE_STRING,
        OSAL_JSON_VALUE_INTEGER or OSAL_JSON_VALUE_FLOAT.
     */
    union
    {
        os_long l;          /* For code OSAL_JSON_VALUE_INTEGER */
        os_double d;        /* For OSAL_JSON_VALUE_FLOAT */
        const os_char *s;   /* For OSAL_JSON_VALUE_STRING */
    }
    value;
}
osalJsonItem;
```

Example, , access compressed JSON data:

```
osalJsonIndex jindex;
osalJsonItem item;

s = osal_create_json_indexer(&jindex, ... )
if (s) error...

while (!(s = osal_get_json_item(&jindex, &item))
{
    switch (item.code)
    {
        case OSAL_JSON_START_BLOCK:
            printf ("%s\n", item.tag_name);
            break;

        ...
    }
    ....
}
osal_release_json_indexer(&jindex);
```

3. Metals – micro-controller hardwares

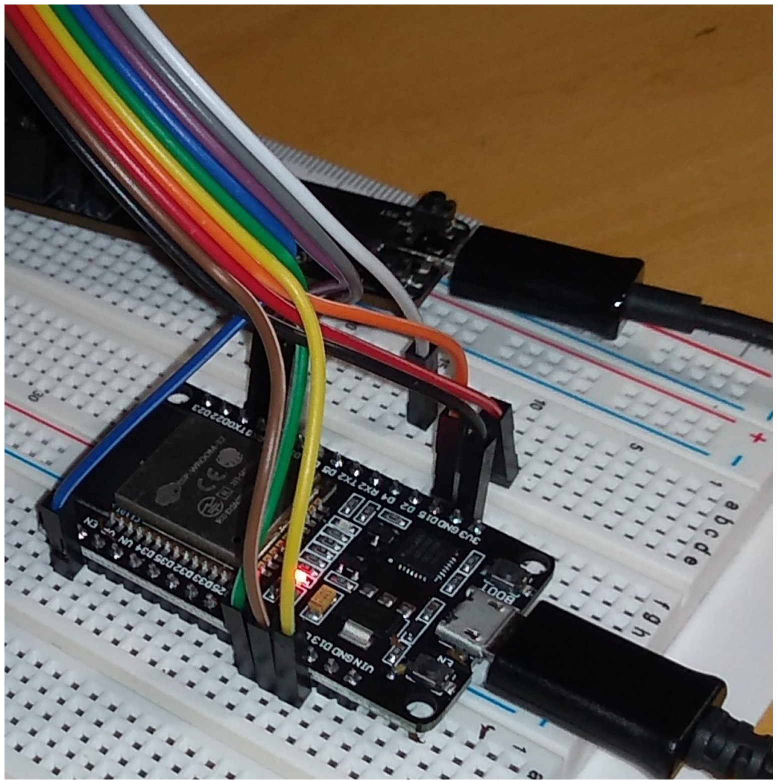
190911, updated 11.9.2019/pekka
X…

3.1 ESP32 with Visual Studio Code, Platform IO, Arduino, and Esp-Prog/JTAG

190915, updated 15.9.2019/pekka
A lot of acronyms in the title. There are many options how to do micro-controller development, and this is one combination. The goal here was to evaluate this choice for the development environment.

3.1.1 Hardware

The micro-controller development board: MELIFE ESP32 Development Board (from Amazon).
External USB/JTAG debugger: ESP-Prog (from Grid Connect).



ESP32 development board and ESP-Prog on background

It is a bit of effort to find correct pins to connect. Initially, I powered the ESP32 trough Esp-Prog. This didn't provide enough current, causing "brownouts" at ESP when Wifi started up. Solved by disconnecting red +3.3V wire between ESP32 and Esp-Proj, and connecting power to ESP32 with it's own USB connector. I am unsure how will this connect the grounds of the two USB cables together? Anyhow after this hardware worked fine.

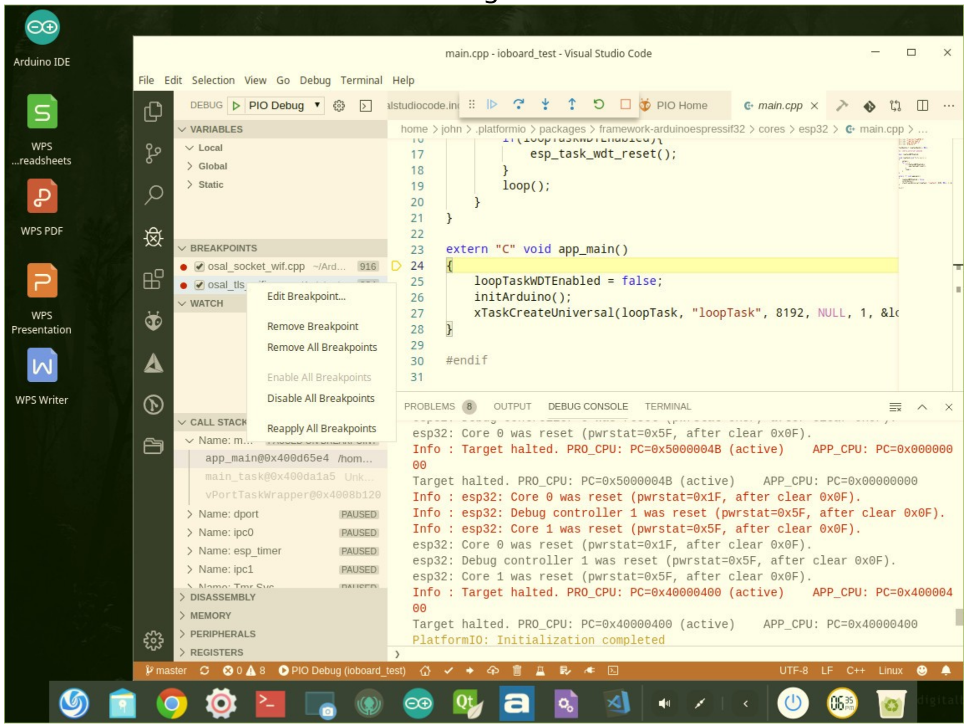
3.1.2 ESP-idf – the ESP32 support and development

The first thing needed to develop for ESP32 is esp-idf by Espressif, the chip maker. It is solid and good, and can be used to build, load and monitor ESP32 code as it. But it is not a full-fledged development environment with a debugger, other components are needed. If one wants a GUI, choices are Eclipse or Visual Studio code. I vote for Visual Studio Code, even it is tricky to set up it is still easier than Eclipse and works with Platform IO

3.1.3 Arduino libraries – nice to use and pain to debug

This is a love/hate relationship. I find Arduino libraries easiest way to write micro-controller code which is at least

“about portable” from micro-controller to another. Way of distributing libraries as .zip files containing sources is also good, and the best part is that the application stays small, simple and readable. Hardware dependent code is not bundled in.
But then. It is an environment which has not given any taught on debugging, which is painful to put it nicely. Options that have worked for me are using gdb -tui from the command line (hackaday) or using Visual Studio Code + Platform IO + GDB. Both have their own tricks and are not easy to set up or use, but get the job done. As a preference I think Visual Studio Code is better of the two, still it is painful to get to work if one is not intimately familiar with the Visual Studio Code and Platform IO configuration.



```
            esp_task_wdt_reset();
        }
        loop();
    }
}

extern "C" void app_main()
{
    loopTaskWDTEnabled = false;
    initArduino();
    xTaskCreateUniversal(loopTask, "loopTask", 8192, NULL, 1, &lo
}

#endif
```

```
esp32: Core 0 was reset (pwrstat=0x5F, after clear 0x0F).
Info : Target halted. PRO_CPU: PC=0x5000004B (active)    APP_CPU: PC=0x000000
00
Target halted. PRO_CPU: PC=0x5000004B (active)    APP_CPU: PC=0x00000000
Info : esp32: Core 0 was reset (pwrstat=0x1F, after clear 0x0F).
Info : esp32: Debug controller 1 was reset (pwrstat=0x5F, after clear 0x0F).
Info : esp32: Core 1 was reset (pwrstat=0x5F, after clear 0x0F).
esp32: Core 0 was reset (pwrstat=0x1F, after clear 0x0F).
esp32: Debug controller 1 was reset (pwrstat=0x5F, after clear 0x0F).
esp32: Core 1 was reset (pwrstat=0x5F, after clear 0x0F).
Info : Target halted. PRO_CPU: PC=0x40000400 (active)    APP_CPU: PC=0x400004
00
Target halted. PRO_CPU: PC=0x40000400 (active)    APP_CPU: PC=0x40000400
PlatformIO: Initialization completed
```

Visual Studio Code/Platform IO/GDB debugging in Deepin virtual machine

3.1.4 TLS – secure transport

The environment comes with Mbed TLS by default. It worked easily > I was happy. The IOCOM library traffic flows nicely from ESP32 to Linux or Windows (or to anything else) trough secure socket. I have still not implemented device identification. So even the communication is encrypted, device and control computer cannot be certain with whom they communicate. This is on to do list.

3.1.5 vmware virtual machines and other easier choices

Setting up micro-controller development environment like this one takes quite a bit of time and Googling. There are many setting, software version requirements, etc, which need to be sorted out. This makes me think that a good way for a new person to get started is to download ready virtual machine for the specific setup. Even if one wants to set up native development environment, the virtual machine can be used as working reference. Then some tools are relatively easy, especially Arduino IDE is simple and easy (without debugging), no need for virtual machine. Atollic True Studio for SMT32 chips is also easy, but tends to bundle much HW dependent code in.
When you get ESP32 make sure that it has USB connector which can be used for debugging (get ESP-WROVER-KIT), no need to fiddle with wires and pins.

4. Build tools, debuggers, IDEs

190911, updated 11.9.2019/pekka
X…

4.1 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.

4.1.1 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.

4.2 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.

4.2.1 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).

4.2.2 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/min
18"
E_ROOT=”/coderoot”
```

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

4.2.3 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".

4.3

4.4 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.

4.4.1 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.

4.4.2 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.

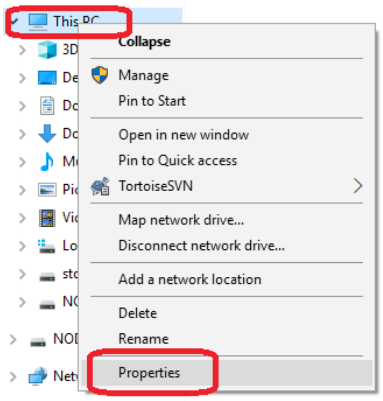
3. Debugging doesn't always work if network connection is not up to Microsoft "specs" (what ever those are). In practice this means that all works when at home, but when traveling I often need to turn off Wifi networking. Otherwise debugging doesn't start, but Visual studio just locks up.

4.5 Windows build environment

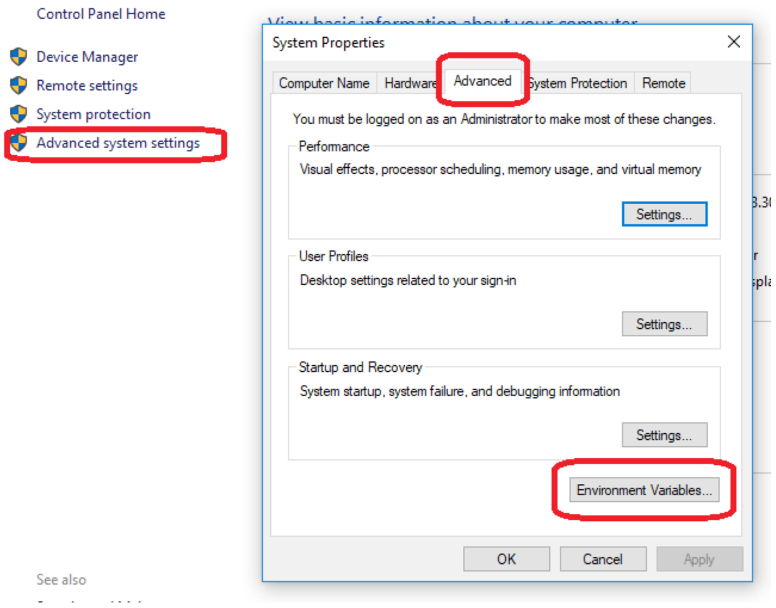
180911 notes 11.9.2018/pekka

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.



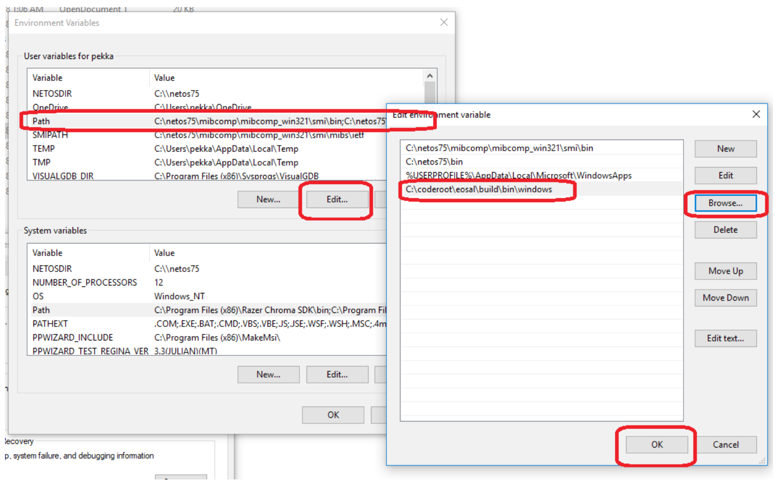
You must be logged on as an Administrator to make most of these changes.

Visual effects, processor scheduling, memory usage, and virtual memory

Desktop settings related to your sign-in

System startup, system failure, and debugging information

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



User variables for pekka

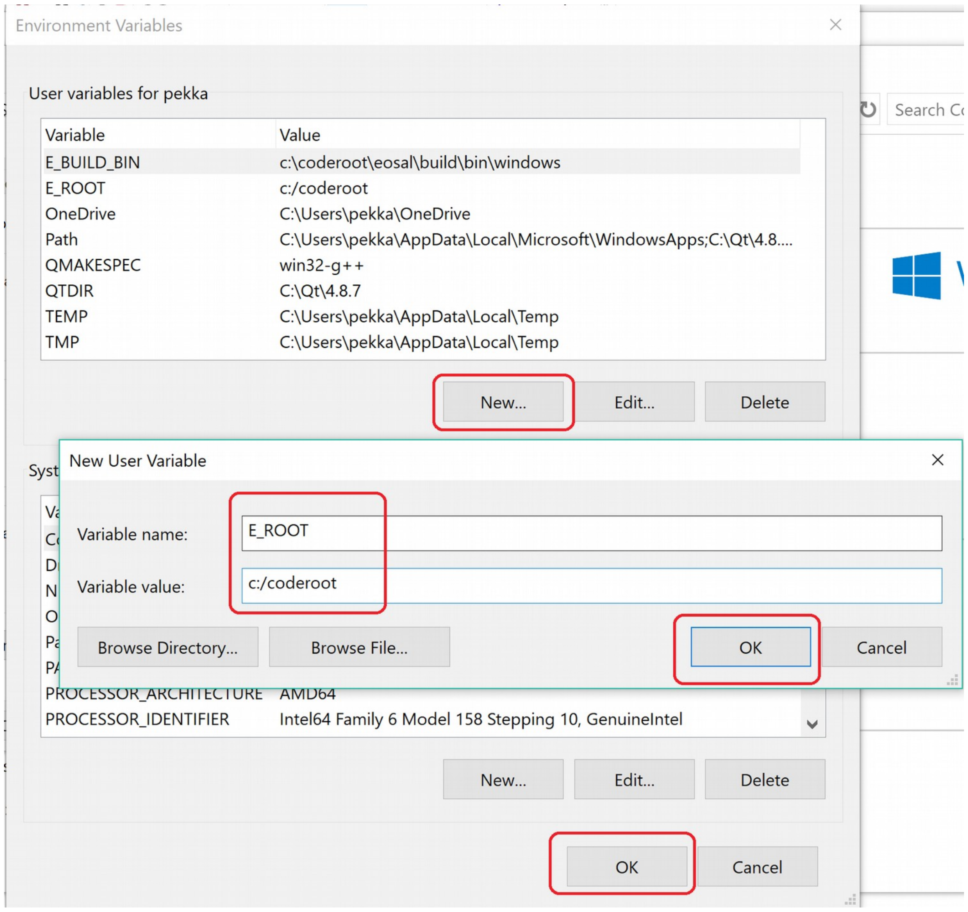
Variable	Value
NETOSDIR	C:\\netos75
OneDrive	C:\Users\pekka\OneDrive
Path	C:\netos75\mibcomp\mibcomp_win321\smi\bin;C:\netos75\
SMIPATH	C:\netos75\mibcomp\mibcomp_win321\smi\mibs\ietf
TEMP	C:\Users\pekka\AppData\Local\Temp
TMP	C:\Users\pekka\AppData\Local\Temp
VISUALGDB_DIR	C:\Program Files (x86)\Sysprogs\VisualGDB

System variables

Variable	Value
NETOSDIR	C:\\netos75
NUMBER_OF_PROCESSORS	12
OS	Windows_NT
Path	C:\Program Files (x86)\Razer Chroma SDK\bin;C:\Program Fil
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.4m
PPWIZARD_INCLUDE	C:\Program Files (x86)\MakeMsi\
PPWIZARD_TEST_REGINA_VER	3.3(JULIAN)(MT)

```
C:\netos75\mibcomp\mibcomp_win321\smi\bin
C:\netos75\bin
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\coderoot\eosal\build\bin\windows
```

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



User variables for pekka

Variable	Value
E_BUILD_BIN	c:\coderoot\eosal\build\bin\windows
E_ROOT	c:/coderoot
OneDrive	C:\Users\pekka\OneDrive
Path	C:\Users\pekka\AppData\Local\Microsoft\WindowsApps;C:\Qt\4.8....
QMAKESPEC	win32-g++
QTDIR	C:\Qt\4.8.7
TEMP	C:\Users\pekka\AppData\Local\Temp
TMP	C:\Users\pekka\AppData\Local\Temp

New User Variable

Variable name:	E_ROOT
Variable value:	c:/coderoot

PROCESSOR_ARCHITECTURE	AMD64
PROCESSOR_IDENTIFIER	Intel64 Family 6 Model 158 Stepping 10, GenuineIntel

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.

4.5.1 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".

4.6 Building OpenSSL on Windows VS 2019

190909, updated 9.9.2019/pekka

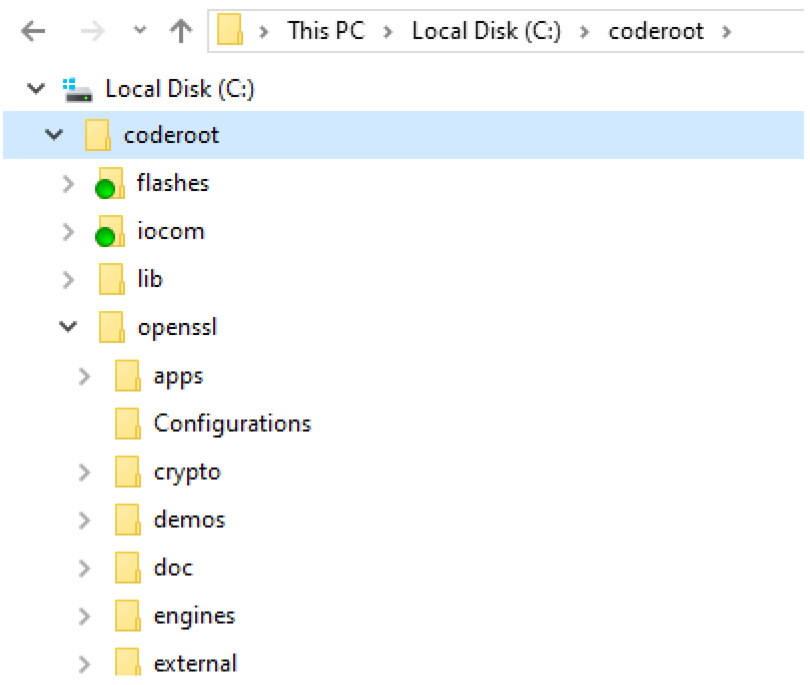
The OpenSLL library is used to provide transport layer security on both Linux and Windows. This paper describes how to take OpenSSL to use for Visual Studio 2019 development. The resulting build will be part of eosal repository, do redoing this is necessary only when taking new OpenSSL version to use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

4.6.1 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.

4.6.2 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:



4.6.3 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.

4.6.4 Configure OpenSSL

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

4.6.5 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

4.6.6 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.

4.6.7 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.

4.7 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.

4.7.1 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.

4.7.2 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.

4.7.3 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.

4.8 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.

4.8.1 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.

4.8.2 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.