MULLE Development Information

Jens Eliasson

January 19, 2007

Abstract

This document describes how to get started with software development for the MULLE platform.
Created with LaTeX.

Contents

Chapter 1

Introduction

The MULLE software consists of a number of different components. Firstly; a Hardware Abstraction Layer (HAL) is responsible for hiding hardware-specific functionality. This enables the software easily be ported to other micro controllers or operating systems. On top of the HAL is the light-weight TCP/IP stack lwIP. lwIP brings memory management, IP, TCP, UDP and various other building blocks. For wireless communication, the MULLE is equipped with a Bluetooth module, and hence needs a Bluetooth stack. The stack in use on the MULLE is the light-weight Bluetooth stack lwBT, which acts as a network interface to lwIP, allowing a multitude of protocols to be used on top of the Bluetooth layer.

Chapter 2

Compilers

To build the MULLE software, a compiler is needed. The one currently supported is the IAR Embedded Workbench from IAR Systems. gcc is supported in emulated mode on the Win32 and Linux platforms. Work is in progress to support gcc for the M16C micro controller as well.

2.1 Starting with IAR 3.10

The first step is to open the workspace file located in the top level in the software folder. This file is named *eisplatform_WS.eww*. Start the IAR Embedded Workbench and choose File→Open→Workspace. Browse to the location of the software folder and open the file *eisplatform_WS.eww*. This workspace is now open. To select the right target, choose *mulle_no_rtos* from the drop-down menu in the *Workspace* window.

The workspace, Fig. 2.1, contains a number of different containers. Below is an short description of each of them:

i2c Files for interfacing the I2C bus. The code currently supports realtime-clocks, EEPROMs and port expanders.

khboard_config Not in use on the MULLE.

lwBT All files for the basic Bluetooth functionality.

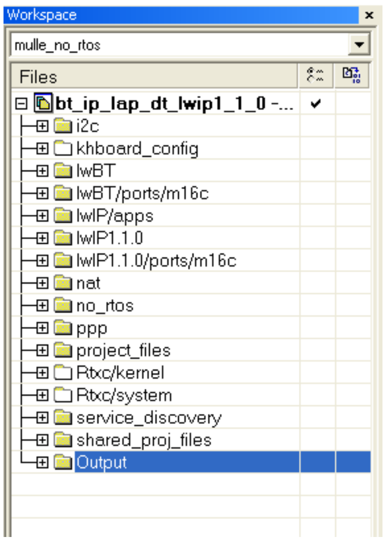
lwBT/ports/m16c Hardware specific files for lwBT, such as UART communication.

lwIP1.1.0 lwIP version 1.1.0 .

lwIP1.1.0/ports/m16c Hardware specific files for lwIP.

nat The NAT (Network Address Translation) layer.

no_rtos HAL files when RTXC operating system is not used.



```
bt_ip_lap_dt_lwip1_1_0 -...  ✓
├ i2c
├ khboard_config
├ lwBT
├ lwBT/ports/m16c
├ lwIP/apps
├ lwIP1.1.0
├ lwIP1.1.0/ports/m16c
├ nat
├ no_rtos
├ ppp
├ project_files
├ Rtxc/kernel
├ Rtxc/system
├ service_discovery
├ shared_proj_files
└ Output
```

Figure 2.1: IAR workspace view

ppp EISLAB PPP implementation.

project_files Includes the intialization files, main function and more.

Rtxc/kernel Files for the RTXC Real-time Operating system

Rtxc/system Files for the RTXC Real-time Operating system

service_discovery Files for the Zeroconf/mDNS-SD (Bonjour) service discovery protocol.

shared_proj_files Various files whick controls the behaviour of the MULLE.

Output Files generated by the compiler.

2.2 Starting with gcc

Support for the GNU C Compiler is ongoing, but not finished yet. When full gcc support is finished, more information will be added here. Currently, compiling software for the M16C microcontroller is fully working, but the M16C-Flasher tool does not seem the be able to read the produced .mot file generated by the linker when building large projects. Smaller projects, such as the TinyTimber kernel seems to be working though.

Chapter 3

The Software

Now that we have introduced to main blocks, it is time to go into detail. This chapter will cover the directory structure and some important files that are commonly used/modified. In Fig 3 the root folder is shown.

3.1 lwIP files

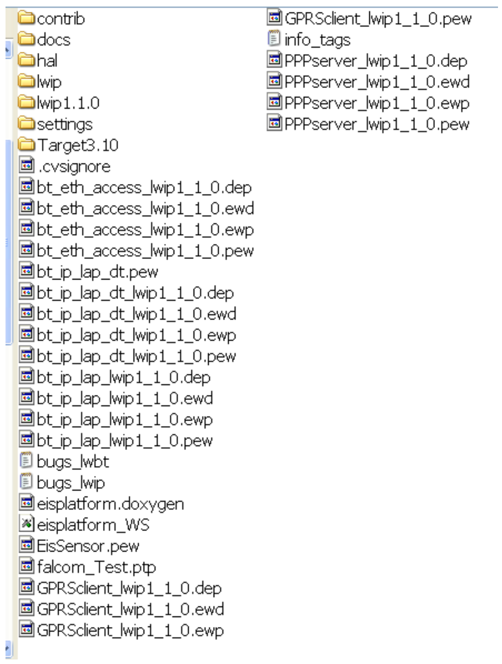
In the two folders *lwip* and *lwip1.1.0* are different versions of the lwIP package. For more information regarding lwIP visit the lwIP homepage at http://savannah.nongnu.org/projects/lwip/ .

3.2 lwBT files

The lwBT files can be found in *contrib/plugin/src/netif/lwbt/*. These are normally don't modified since othrer control files are used to control the Bluetooth stack. To modify the Bluetooth behaviour, change the control files instead.

3.3 Control files

In this category we can find the system main main maintask.c and the Bluetooth control file bt_ip_*.c. There are also some header files which are used to setup a configuration, such as which sensor that is used, memory sizes and so on. The proj_arch.h file is the most important file since it control how the whole software is compiled. Three other commonly changed files are lwipopts.h, lwbtopts.h ant natopts.h which in detail controls lwIP, lwBT and the NAT functionality.



```
contrib                           GPRSclient_lwip1_1_0.pew
docs                              info_tags
hal                               PPPserver_lwip1_1_0.dep
lwip                              PPPserver_lwip1_1_0.ewd
lwip1.1.0                         PPPserver_lwip1_1_0.ewp
settings                          PPPserver_lwip1_1_0.pew
Target3.10
.cvsignore
bt_eth_access_lwip1_1_0.dep
bt_eth_access_lwip1_1_0.ewd
bt_eth_access_lwip1_1_0.ewp
bt_eth_access_lwip1_1_0.pew
bt_ip_lap_dt.pew
bt_ip_lap_dt_lwip1_1_0.dep
bt_ip_lap_dt_lwip1_1_0.ewd
bt_ip_lap_dt_lwip1_1_0.ewp
bt_ip_lap_dt_lwip1_1_0.pew
bt_ip_lap_lwip1_1_0.dep
bt_ip_lap_lwip1_1_0.ewd
bt_ip_lap_lwip1_1_0.ewp
bt_ip_lap_lwip1_1_0.pew
bugs_lwbt
bugs_lwip
eisplatform.doxygen
eisplatform_WS
EisSensor.pew
falcom_Test.ptp
GPRSclient_lwip1_1_0.dep
GPRSclient_lwip1_1_0.ewd
GPRSclient_lwip1_1_0.ewp
```

Figure 3.1: The root folder

3.4 Applications

All applications are located in *contrib/apps/*. The important ones are the web server *httpd* and the sensor deamon *sensord*. If a new sensor type is to be added to the MULLE, the *sensord.c/.h* files must be modified.

3.5 IP packet flow

This section will outline how an IP packet is traversing the Bluetooth- and IP-stack. For incoming packets, either using PAN or LAP, the first interesting file is either uart_h4.c or uart_bcsp.c (depedning on if the Bluetooth module is using H4 or BCSP). All Bluetooth traffic passes through phybusif_input(). Here the UART bytestream is converted to packets, and sent to the HCI layer via hci_input(). When HCI has recognised the packet as an ACL data packet, it is sent to L2CAP, or in the case of a HCI command or event, it stays in the HCI layer. When a data packet is sent into L2CAP using l2cap_input in l2cap.c, L2CAP will forward it the the BNEP layer (in case of PAN) or RFCOMM (in the case of PPP (LAP or DUN profiles)). If PAN is used, the packet will be sent to the BNEP layer via bnep_input(). If the packet is a BNEP command or response, it will stop here, otherwise it is forwarded to bnep_frame_input(). IP packets will be sent to ip_input()

or nat_input() (when NAT is used on a PAN netif). In case of multicast or broadcast Ethernet packets, BNEP will forward them to all BNEP links. On the other hand, if PPP is used, packets will be passed from L2CAP to RFCOMM using rfcomm_input(). RFCOMM will recognise it as beeing a PPP packet and will pass it into the PPP layer (via get_ppp_frame() in bt_ip_*.c and ppp_input() in ppp.c). The IP layer will send UDP packets to udp_input() in udp.c and TCP packets to tcp_input() in tcp_in.c and tcp.c. When the packet has reached the UDP or TCP layer, it is sent to the application recieve callback function.

3.6 Ostmark link protocol

To communicate with the sensor deamon on a MULLE, a simple TCP-based protocol is used. This protocol comes in different versions depending on how the MULLE is configured. Howeverm in the future, all MULLEs will use the same version of the protocol. This section describes the version that must be used by all new MULLEs.

3.6.1 Header layout

All communication is performed by sending packets over a reliable TCP connection. A packet header is five bytes long, and consists of a one byte type code, a two byte length field and two (currently) unused bytes.

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |     type      |      length (little endian)     |  unused0    |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |    unused1    |  (payload, if len > 0)                        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

A C struct implementation might look like this:

```
struct eis_packet {
  uint8 type;
  uint16 len;
  uint16 unused;
};
```

Remember that some C-compilers use byte-stuffing (i.e. the type might be stored as a 4 byte integer for performance reasons). Make sure that sizeof(struct eis_packet) equals 5.

Chapter 4

The build process

4.1 Building a firmware

To build a firmware for the MULLE, simply press F7 or choose Project→Make. This will create a new firmware called *mulle_no_rtos.mot* located in *Target3.10/bt_ip_lap_dt_1_1_0/mulle/no_rtos/Exe/* which can be downloaded into the MULLE using the M16C Flasher program.

4.2 Configuring using proj_arch.h

To change settings, the first thing to do is to modify the *proj_arch.h* file. To modify communication settings, look for the following section:

```
/*
 * Interface stuff
 */
#define USE_PPP_NO_LWBT     0   /* DO NOT CHANGE THIS Macro ! */
#define USE_ETH             0
#define USE_LWBT            1
#define USE_LAP             0
#define USE_BNEP            1
#define USE_SDD             1   /* mDNS Service discovery */
#define USE_I2C             1
```

For the MULLE USE_ETH must be 0. All other (except USE_PPP_NO_LWBT of course) can be either 0 or 1. Normally we want Bluetooth functionality so USE_LWBT, USE_LAP and USE_BNEP should be set to 1. If support for service discovery is wanted, set USE_SDD to 1 also. For I2C support, set to 1 (default).

The correct sensor type can be selected in the

```
/*
```

```
 * Sensor stuff
 */
#define NONIN_FORMAT1       0
#define NONIN_FORMAT2       0
#define ICS3031             0
#define LM61                1
#define BP104               0
#define ADXL_320            0
#define ADXL_321            0
#define ADXL_322            0
#define DS600               0
#define LASSEN_SQ           0
#define PULSESIM            0
```

section. The MULLE with built-in temperature sensor is the DS600, the IR-sensor is called BP104 and the accelerometer-MULLE is ADXLXXX, where XXX depends on the accelerometer input range. The

```
/*
 * Appz stuff
 */
#define USE_CENTRAL_SERVER  1
#define LWIP_HTTPD          1
#define LWIP_NTPD           0
```

section controls which applications are compiled, normally we want the web server so make sure that the LWIP_HTTPD define is 1. The USE_CENTRAL_SERVER define tells weather the MULLE should connect to the EISLAB public server or not. This is also normally set to 1.

For the Bluetooth to work, the correct settings must be used. Check the

```
/*
 * Defines checking
 */

...

#if USE_LWBT
  #define USE_BCSP            1
  #define USE_BCSP_LE         1
  #define USE_H4              0
  #define USE_NAT             1
  #define USE_NATPMP          0
#else // USE_LWBT == 0
  #define USE_BCSP            0
  #define USE_H4              0
  #define USE_NAT             0
#endif // USE_LWBT
```

Normally the Bluetooth-module is set to BCSP with Link-Establishment, so USE_BCSP and USE_BCSP_LE should both be set to 1, while USE_H4 should be 0. NAT are also used by default, so set USE_NAT to 1 as well. However, NAT-PMP (NAT Port-mapping protocol) are rarely used so set to 0 if uncertain.

4.3 Using debug options

To modify debug settings, the file *proj_debug.h* must be altered.

```
/* comment out a line for no debugging */
#define SHOW_MEM_ERROR
#define LWIP_DEBUG
//#define LWIP_NOASSERT
#define LWBT_DEBUG
#define SENSORD_DEBUG

//#define VJ_DEBUG           DBG_ON  /* Debugging VJ TCP header compression */
//#define HTTPD_LWIP_DEBUGF DBG_ON
#define BT_IP_DEBUG       DBG_ON
//#define CS89X0IF_DEBUG    DBG_ON
#define NTP_DEBUG       DBG_OFF
#define I2C_DEBUG       DBG_ON
```

To completely disable all debugging, uncomment the line LWIP_DEBUG. This turns off the lwIP debugging functionality, and hence all other debugging as well. To control individual layers, see lwipopts.h and lwbtopts.h respectively.

4.4 Downloading the new firmware

To download the new firware, make sure that the MULLE is connected to the computer. Place a jumper between Vcc and CNVss, which will set the MULLE in re-programming mode. Turn on power. Then start the M16C-Flasher application and press "Connect". When the text "Connected OK" appears, press "Prog" and select the new .mot file that we just built. When the new firmware is downloaded, turn off power, remove jumper and press "Terminal" (make sure that UART baud rate is set to 9600 or 57600, depending on software configuration). Now turn on power again. The following text should appear:

```
EISLAB platform MULLE started.

TCP/IP initialized.
Reseting BT module
Bluetooth initialized.
Bluetooth interface is up!
Application(s) started.
bt_ip_start:
Successful HCI_RESET.
...
successful HCI_WRITE_PAGE_TIMEOUT.
Initialization done.
LAP % SPP intialized
pan_init: PAN IP 10.0.0.1
PAN intialized
successful HCI_SET_EVENT_FILTER.
successful HCI_WRITE_SCAN_ENABLE.
Discover other Bluetooth devices.
```

All lines starting with "successful HCI_" means that the microcontroller is communicating with the Bluetooth module. In the end, the Bluetooth stack is performing an Inquiry, trying to find other nearby Bluetooth devices such as phones or access-points. However, if the control files bt_ip_pan_gn.c or bt_ip_lap.c are used, no scan will be performed. Instead, the MULLE will be awaiting connections initiated by a user.