Version N° 1.1

LaurTec

LTlib 5

User Manual

Author : *Mauro Laurenti*

License

Index

Introduction....4
The path from LTlib 4 to LTlib 5....4
Software versions Free, Maker and PRO....5
LTlib file organization....6
LTlib.h file....7
LTlib configuration files....8
Create a new project....9
Change a project from one architecture to another....12
Use a different configuration file....13
Add a new device to the library....15
Use MCC with LTlib....19
Bibliography....21
History....22

Introduction

LTlib library makes programming with the MCUs easier than before. It integrates both MCU peripheral libraries as well as external IC libraries. This makes implementing an application as easy as connecting building blocks. The IC libraries span from IO extenders, LCD drivers, memories, data converters and sensors. LTlib is independent from Microchip peripheral libraries as well from the standard libraries that may come with the compilers. As it is now, the library supports PIC microcontrollers from the following architectures:

- PIC with 8 bit architecture.
- PIC with 16bit architecture.
- PIC with 32 bit architecture.

The path from LTlib 4 to LTlib 5

LTlib 4 was first introduced in February 2016. The main goal was to make the previous PIC18 library independent from the Microchip Peripheral library. This was done by introducing the module_xxxx libraries to cover and support the internal peripherals. Furthermore the IC libraries got integrated with the peripheral libraries, making the IC initialization easier. Indeed, by each IC initialization it was possible to initialize the MCU as well. Each supported MCU got a configuration file, out of which it was possible to provide, beside the MCU configuration, also the MCU characteristics used by the LTlib library.

LTlib 5, inherited all that features, further extending it. The library architecture around LTlib 4 made easy extending it to other MCUs, by keeping the same configuration file and basic code. All the IC libraries got supported by extending the module_xxxx files. The configuration files have been reorganized to make their customization easier. This allow adding new MCUs with less effort.

As code clean up, to support easy of use, by LTlib 5, all the C18 code compatibility has been removed, supporting XC compilers only. Each code and example has been re-compiled supporting XC compilers from version 2.x, thus supporting C99 standard by default. This was needed since LTlib 4 was developed for compiler versions smaller than 2.x.

To improve the code standards, all the code support now data format according to ANSI standard, such as uint16_t instead of simple int. This allows keeping a known variable size by changing MCU architecture. Indeed by moving codes by 8, 16 and 32 bits MCU architectures, subtle bugs may occur if that precaution would not have been taken.

Those changes allow writing a code that may easily be compiled for 8, 16 and 32 bits MCUs.

Software versions Free, Maker and PRO

The activities made by LaurTec, among which LTlib, always have the main goal to support education applications for free, without charge. Thus the library, with minor limitations is offered also for free. The library is offered in three different versions:

- Free
- Maker
- PRO

The Free version can be used without charges for non commercial applications, as specified in details within the header file of each library. It includes 8 bits architecture only.

The Maker version offers some additional libraries. As the free version is not intended for commercial applications. That version can be requested with a simple donation. No minimum donation is needed, just offer the coffee you want let me drink to remain awake while coding. The maker version is intended to support further development of the library and payback the working hours behind it. Professors may request for free the maker version and offer the same rights to the students that will attend the class. So for the students there would be no need to spend any money to get the libraries used during the class. They can bring their maker version at home and keep programming. The Maker version includes 8 bits architecture only.

PRO version, it is based on the maker version but may have additional libraries. Major difference is that it can be used for commercial purposes and you would have direct support. PRO version is for sale and you would need to request a quote for it. 8,16 and 32 bits architecture can be requested separately.

LTlib file organization

LTlib does not need any installation, you just need to download it and copy the folder within your preferred working path. The library name is LTlib_v_5.x.x where the x.x denotes the subversion. This allows using multiple versions of the library without overwriting older ones. Each project can be linked with a specific version of the library. The folder is organized as shown in Figure 1.

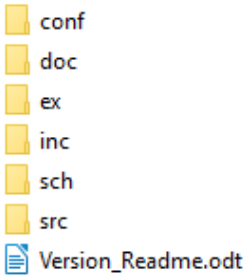

Figure 1: *LTlib organization.*

The content in each folder is as follow:

- **conf**
  Configuration files related to each supported devices. Within that folder you can find also the hardware files related to the boards developed by LaurTec. The hardware files have definitions for LEDs, buttons or similar hardware, specific to a board.

- **doc**
  LTlib documentation. This includes the high level information and not the library documentation. The library documentation can be found inside each header file.

- **ex**
  Code examples. Each library has a folder with one or multiple examples that show the basic library usage. Examples are offered for 8, 16 and 32 bits.

- **inc**
  Include files for each library and LTlib.h. The header file represents the documentation for each library. The format is compatible with Doxigen, thus it could be extracted and navigated by extracting it in HTML format. The documentation inside the doc folder does not provide the extracted documentation from the header files.

- **sch**
  It contains simple schematic to properly use a specific library. Referring to the datasheet is always recommended, since a proper schematic may differ by each application.

- **src**
  It contains the source code of each library.

- **src/modules_xxx**
  It contains the source code for the peripheral libraries. The available codes are architecture depended, thus you have one folder per MCU architecture 8, 16 and 32 bits.

LTlib.h file

The configuration file LTlib.h , that can be found inside the `inc` directory, is a key part of the LTlib library. LTlib.h file must be included inside each project. The file contains the main settings that are related to the MCU, such as the clock frequency, compiler settings and supported devices. Below there is a simple code cut out from the file:

```
//*****************************************************
//           COMPILER AND MCU INFO
//*****************************************************

#if defined (_XC_H_) || defined (__XC_H)
    #define COMPILER_XC
#endif

#ifdef __XC8
    #define COMPILER_XC8
#endif

#if defined (__PIC32C) || defined (__PIC32M)
    #define COMPILER_XC32
#endif

#ifdef COMPILER_XC
    #include <xc.h>
#endif

//*****************************************************
//             LIBRARY TYPE DEFINITIONS
//*****************************************************

#include "LTlib_types.h"

//*****************************************************
//             SYSTEM & MODULE CLOCKS
//*****************************************************

#ifndef SYSTEM_CLOCK
    #define SYSTEM_CLOCK 20000000
#endif

#ifndef I2C_CLOCK
    #define I2C_CLOCK SYSTEM_CLOCK
#endif

#ifndef UART_CLOCK
    #define UART_CLOCK SYSTEM_CLOCK
#endif

#ifndef SPI_CLOCK
```

```
    #define SPI_CLOCK SYSTEM_CLOCK
#endif
```

It is possible to see that also the compiler is cheeked here. This allows to properly initialize the LTlib library.

Many parameters inside the LTlib library support being changed without the need of changing the file.
LTlib.h already offers that feature as other IC libraries. Each parameter that can be changed is typically wrapped as follow:

```
#ifndef UART_CLOCK
    #define UART_CLOCK SYSTEM_CLOCK
#endif
```

In this way if there is no previous definition of the parameter, the default one is used. To set the UART_CLOCK to a different value it is required to define it before the LTlib file is called the first time.

LTlib configuration files

The other key configuration files are the ones related to each supported device, these can be found within the `conf` directory. Each device that is supported has a configuration file. The configuration file is automatically loaded by LTlib.h depending on the device that gets selected once a new project is created, via MPLAB X IDE. There is indeed no need to specify to the library the used device, since that information gets automatically retrieved by the project information, upon its creation or once the device gets changed.

Each device configuration file is made of three parts:

- **LTlib configurations**
  It contains the list of supported peripheral modules. In this way, if a specific module library is used, it can check if the module is supported.

- **Module Settings**
  It contains all the module information. Such as number of IO, ADC channels, number of UARTS and so on. For each module, there is also the count for it and the pin location in case specific pins may need to get properly initialized.

- **MCU configurations**
  These are the standard MCU configuration required by the compiler and the selected MCU. It is a list of `#pragma config` associated with each configuration. The configuration related to each MCU is provided within the XC compiler documentation.

Create a new project

Using LTlib is quite easy. Since no installation is required, the only thing which is needed is to update the IDE include paths.

The first thing to be done is to create a new project. If you have already a new one and you want to start using LTlib, works fine as well.

Afterward, you need to update the project proprieties. Just select your project in the navigation pane, and right click on it, then select *Proprieties*. If you are working with MPLAB X and XC8 you will get the window as the one in Figure 4.

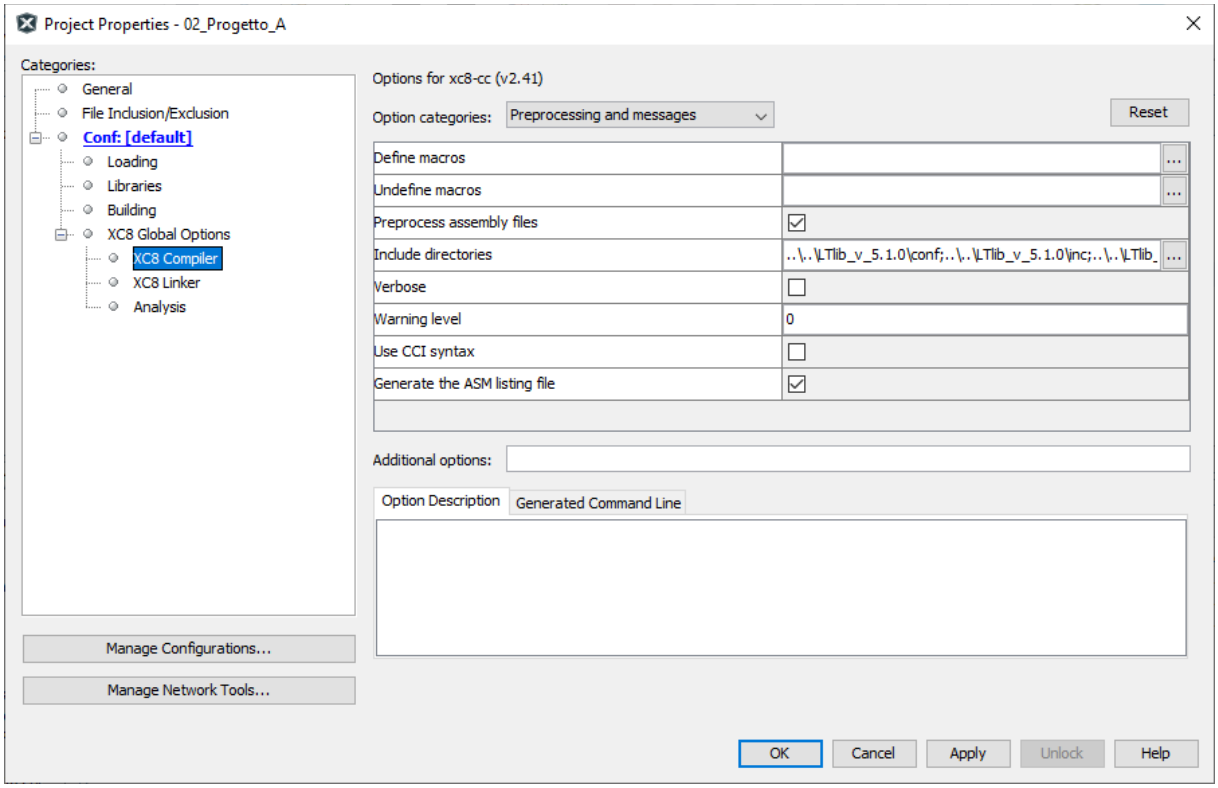

Figure 2: *Project Proprieties window - XC8.*

Among the global options on the left side, select XC8 Compiler, while on the right side update the include directories field. You can than add the following paths:

- conf
- inc
- src
- src\modules_PIC_8_bits

the include paths show that the module source code is architecture dependent, thus beside the `src` path, it is required to add the specific folder containing the module libraries for the specific architecture that is used.

LTlib 5, is compiled with the warning level 0 rather than -3, thus it is recommended to

change it to 0. Keeping the level as -3, may show some additional Messages and Warnings depending on the library that is used.

If the project is based on the MPLAX IDE and the compiler XC16, the steps are the same, but the Project Proprieties window is slightly different, as shown in Figure 3.

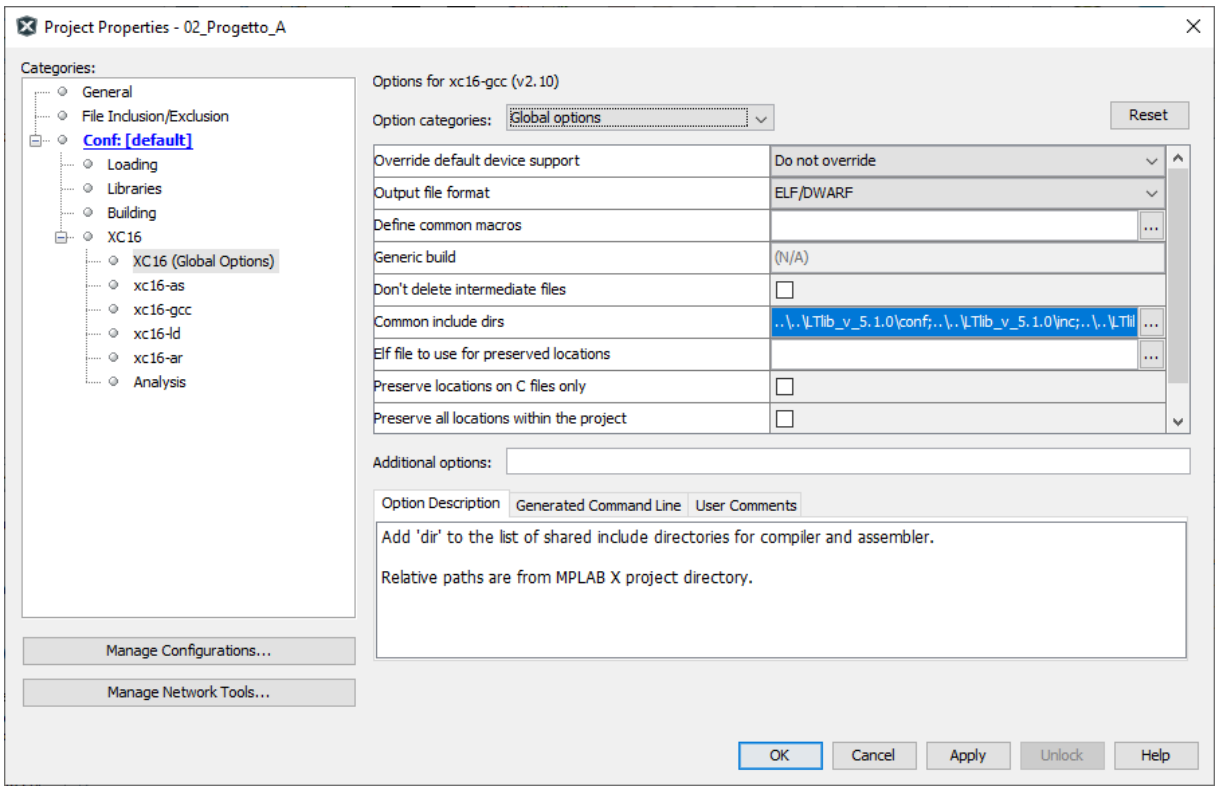

Figure 3: *Project Proprieties window - XC16.*

In this case to set the paths, you need to select the *XC16 (Global Options)* on the left side and the *Common include dirs* field on the right side. You can than add the following paths:

- conf
- inc
- src
- src\modules_PIC_16_bits

This time as well, the module libraries path is architecture dependent.

If the project is based on the MPLAX IDE and the compiler XC32, the steps are the same, but the Project Proprieties window is slightly different, as shown in Figure 4.

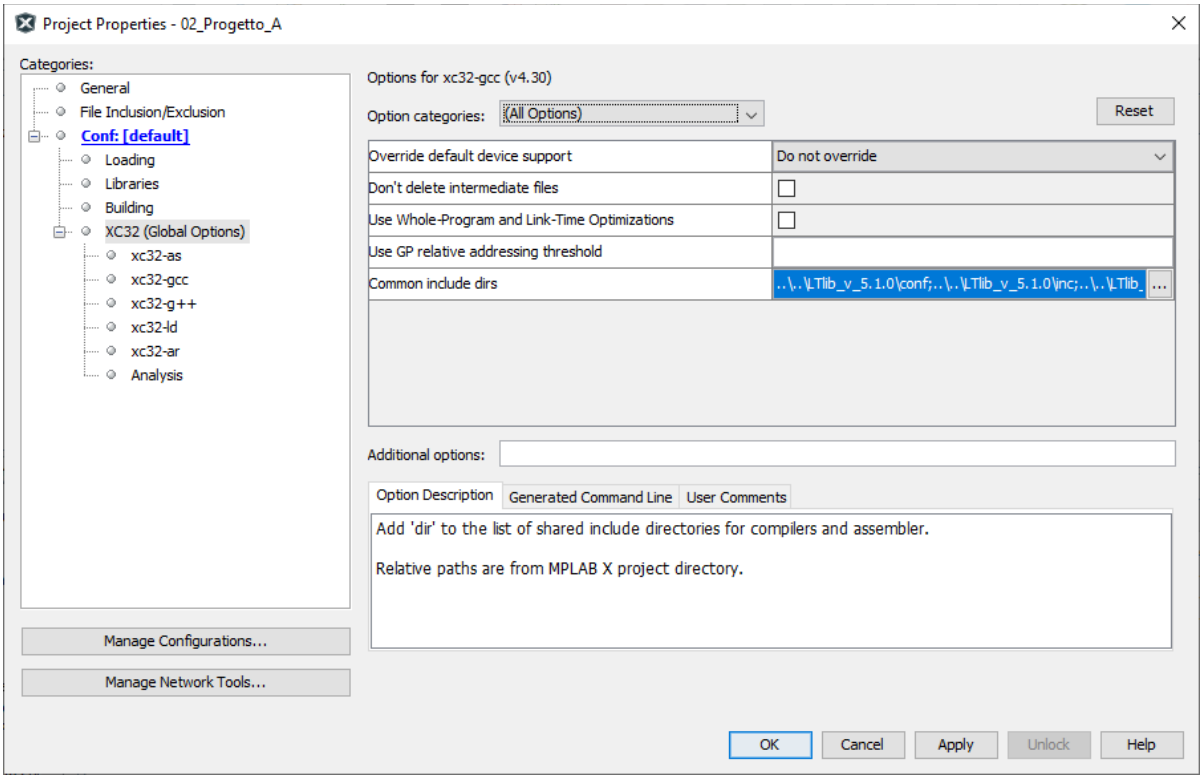

Figure 4: *Project Proprieties window - XC32.*

In this case to set the path, you need to select the *XC32 (Global Options)* on the left side and the *Common include dirs* field on the right side. You can than add the following paths:

- conf
- inc
- src
- src\modules_PIC_32_bits

This time as well, the module libraries path is architecture dependent.

Change a project from one architecture to another

As it has been shown during the creation of a new project, the only thing that change between the architectures is that the module libraries path is changed. This is a key change while switching the project from one MCU architecture to another. Nevertheless before doing it, the following steps are required.

Select the Project Proprieties – configuration window and set the new MCU Device first, Figure 5 shows the case of moving from PIC18 (XC8) to PIC24F (XC16). Once you have selected the new MCU from the Device field, the *Compiler Toochains* window on the bottom side, gets updated with the new available compilers. Select the one that apply to your use case and press the *Apply* button.

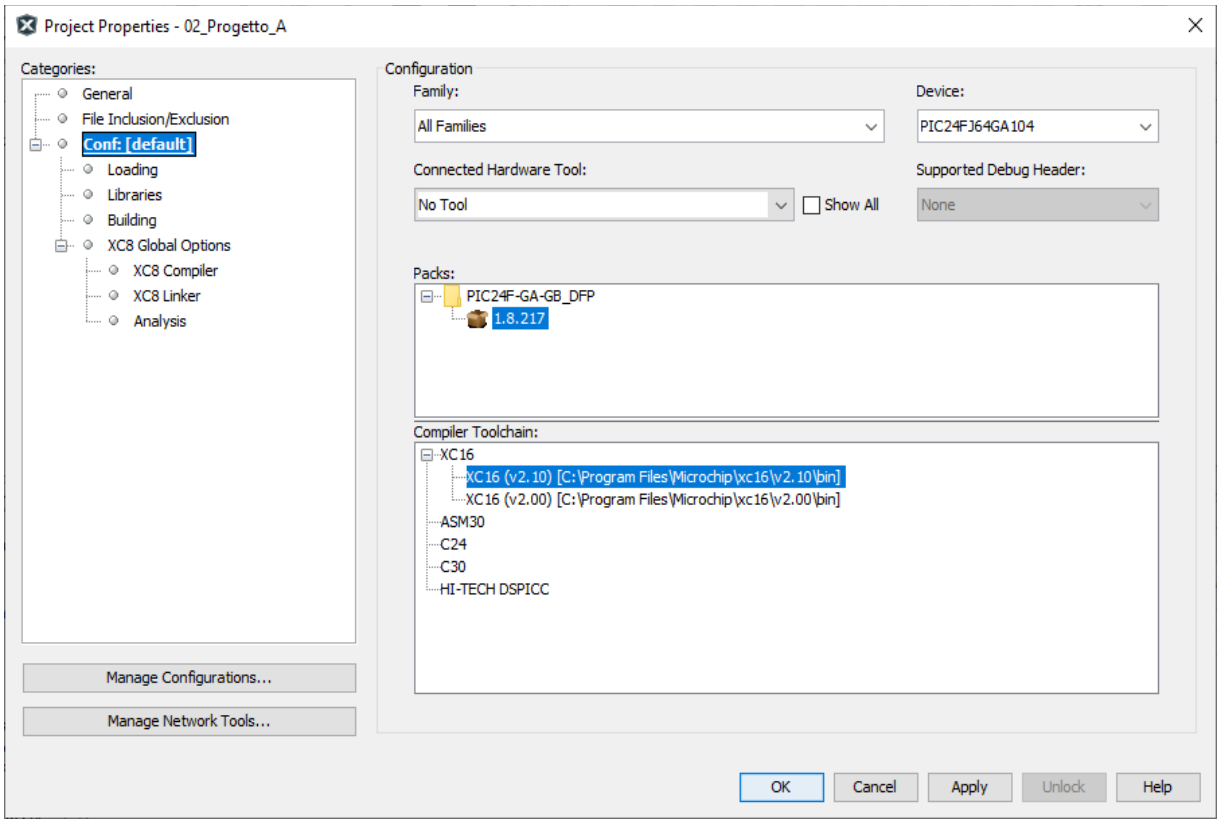

Figure 5: *Project Proprieties window – new MCU selection.*

Only after you apply the new MCU and compiler, the left side of the window is updated with the new proprieties and settings for the new *Tool chain*. In particular, the left side, after applying the new settings, will be updated as shown in Figure 6.

From that point, the included path are removed, thus you need to insert it again as it has been shown on the previous paragraph.

At this point, you are ready to go, programming with the new architecture.

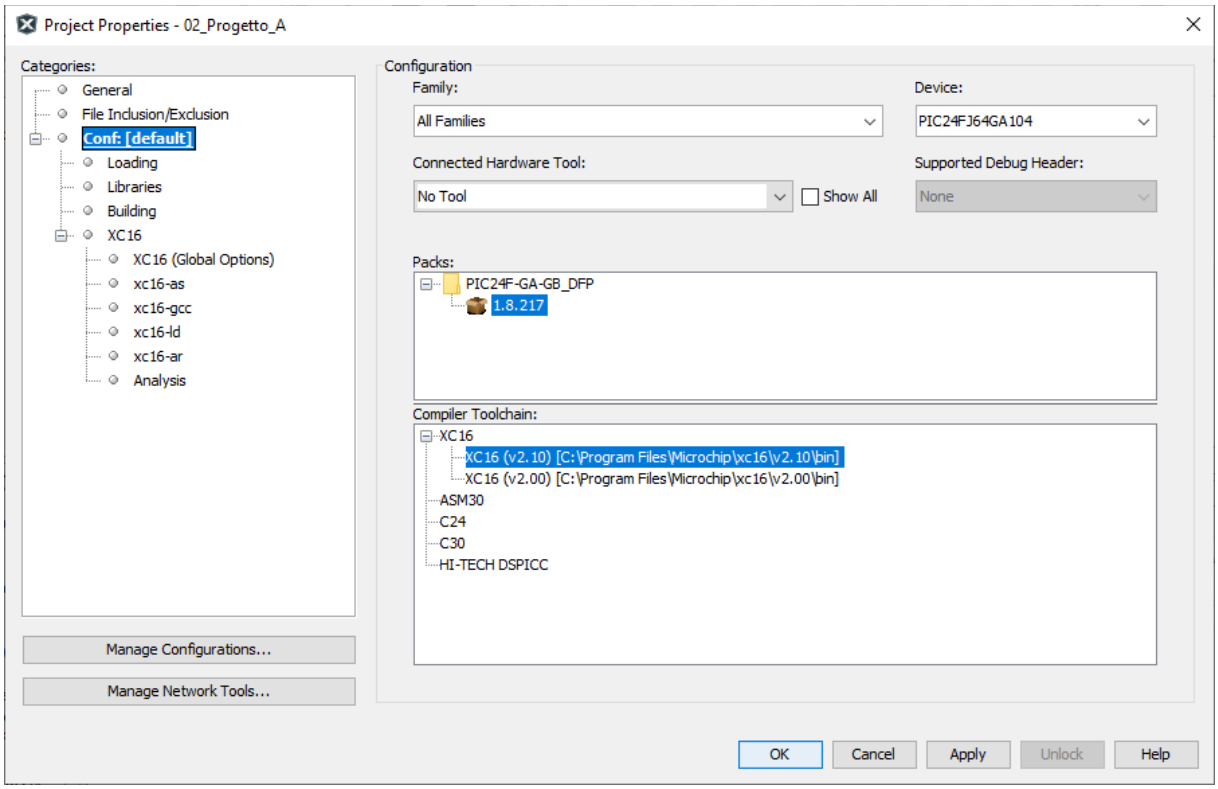

Figure 6: *Project Proprieties window – after the new Toolchain.*

Use a different configuration file

LTlib 5 comes with the configuration file for each device that is supported. Configurations are made to be changed, thus the configuration file can be changed as well. There are 4 major ways to do that with pro and cons, as shown below.

- Change directly the configuration file.
  You can update directly the configuration file, but this means loosing the original file content. Nevertheless is the quickest and easiest ways to do it. To get the original file you can always download LTlib again.

- Copy and paste the configuration file of interest and rename it with your project name or similar name. In this way you do not loose the original library file.
  The new configuration file is not automatically loaded by the library. To properly call the configuration file you need to change the LTlib.h file with your new file name. For instance if you have created a new configuration file for the PIC18F46K22 you can change the following section of the LTlib.h file:

```
#ifdef _18F46K22
    #include "PIC18F46K22_config.h"
    #define CHIP_SUPPORTED
#endif
```

the file name PIC18F46K22_config.h, must be updated with your new

configuration file name. By each new project you will create with the PIC18F46K22 the new configuration file is loaded automatically.

- Another option for loading a new configuration without changing the LTlib.h file would be to create two copies for the configuration file, so that you have the original one plus the two copies, as shown below:

  PIC18F46K22_config.h
  PIC18F46K22_config_Copy_1.h
  PIC18F46K22_config_Copy_2.h

  you can then rename one of the copy as xxx_Original and the second one with your project name. This second file is the one that you would change based on your configuration needs.

  PIC18F46K22_config.h
  PIC18F46K22_config_Original.h
  PIC18F46K22_config_Your_Project.h

  By creating a new project, only the PIC18F46K22_config.h is loaded, thus you can change the content of the PIC18F46K22_config.h by including the configuration you want. For instance:

```
#include "PIC18F46K22_config_Your_Project.h"
```

  All the configuration information can be deleted, you just write the single code line as before. Indeed the configuration file PIC18F46K22_config_Original.h has the original settings and nothing has been lost.

  With this change the new configuration file will be automatically loaded. If a new configuration is needed you can add it by creating a third file and update the `#include` option within the main configuration file.

- Last but not least, all the configuration files are within the `conf` directory. Copy the configuration you need and add it within your project folder. Do not add the `conf` path within the project paths, so that the configuration will be loaded from the project root path and not from the `conf` directory. You can change the local configuration file as you need, without the need of creating new files. The original configuration files are preserved.

Add a new device to the library

LTlib 5 comes with the support of the MCUs that are typically used within the LaurTec projects. This means that the library may not support directly the MCU you need. This is in general not a big problem since many MCUs share similar module architectures. If specific modules are not supported or different, you may need to update the module libraries and it may get more complicated unless you are an experienced programmer. In the following section I will cover only the case where you do not need to update the module libraries. Nevertheless the following steps would apply to the second case as well. So, you would need to follow first the following steps, and if you do not need to change the module libraries you are ready to go, while if you need to change it, you would do it after the updates shown below.

Let us analyze few easy Use Cases

Use Case A

The easiest use case is when you need to add the same device with another voltage level. For instance you need the PIC18LF46K22 while the library supports the PIC18F46K22 without L. In this case just copy and paste the configuration file PIC18F46K22_config.h and rename it PIC18LF46K22_config.h , by adding the L. Open the configuration file and update the definition Header from:

```
#ifndef PIC18F45K22_CONFIG_H
#define PIC18F45K22_CONFIG_H
```

to:

```
#ifndef PIC18LF45K22_CONFIG_H
#define PIC18LF45K22_CONFIG_H
```

Afterward you would need to update the LTlib.h file, so that once you will create a new project with the PIC18LF46K22, it would automatically load the new configuration file. If you will forget that step, by selecting the PIC18LF46K22 and compiling the project, you will get the error:

```
#error LTlib is not tested on the microcontroller you have selected
```

The new MCU must be added within the compiler group that can be found within the LTlib.h file. Indeed each compiler has a list of MCUs that are supported.

For the PIC18LF46K22 you can search for the COMPILER_XC8 and add the following line after any MCU of the group, as shown below.

```
#ifdef _18LF46K22
    #include "PIC18LF46K22_config.h"
    #define CHIP_SUPPORTED
 #endif
```

After that modification you can compile new projects with the new MCU.

Use Case B

Let's assume we want now to add the PIC18F23K22. The new MCU is similar to the PIC18F46K22 since it belongs to the same family but it has a different package. This will reflect in having less IO, probably different number of ADC channels and sometime less communication modules. With other MCUs packages you may have the other way around, such as more pins and more ADC channels.

As case A, we can start by copy and paste the configuration file of the PIC18F46K22_config.h, since it is the most similar MCU. We can then rename it PIC18F23K22_config.h.

This time the configuration must be really checked and updated, since we have a different package. The configuration file contains all the peripheral settings that are supported:

```
#define IO_LIBRARY_SUPPORTED
#define UART_LIBRARY_SUPPORTED
#define SPI_LIBRARY_SUPPORTED
#define I2C_LIBRARY_SUPPORTED
#define EEPROM_LIBRARY_SUPPORTED
#define ADC_LIBRARY_SUPPORTED
#define FLASH_LIBRARY_SUPPORTED
```

in this case they are the same, otherwise we should remove or eventually add it, depending on the available modules. The right definition name to be added can be found on similar MCU configuration files or by opening the module library of interest. Indeed the definition name is checked within each module library file.

Afterward it is needed to go to each peripheral setting and double check it. In particular the pin definitions may change.

For the IO section we would need to update it from the following code:

```
//****************************************
//                IO
//****************************************
#define  NUMBER_OF_IO_PORTS 5

#define  PULL_UP_RESISTORS_AVAILABLE
#define  PULL_UP_ENABLE_BIT INTCON2bits.RBPU
#define  PULL_UP_SINGLE_BIT_ENABLE
#define  PULL_UP_ENABLE_REGISTER_B WPUB
```

into:

```
//*****************************************
//                IO
//*****************************************
#define  NUMBER_OF_IO_PORTS 3

#define  PULL_UP_RESISTORS_AVAILABLE
#define  PULL_UP_ENABLE_BIT INTCON2bits.RBPU
#define  PULL_UP_SINGLE_BIT_ENABLE
#define  PULL_UP_ENABLE_REGISTER_B WPUB
```

If MCLR is used as IO, since it is mapped to PORT E, you need to keep 5 as port count and not 3. Otherwise you will not be able to access PORTE via module_IO library. Keeping 3 still make possible to access the port via direct access of the register PORTE, thus without using the LTlib library. This possibility remains at any time, but mixing the code too much may lead portability problems.

The communication modules require some care since the reduced IO pins may imply that the pins are mapped differently. An example is the UART module 2 which is not on PORT D but PORT B, thus the configuration must be updated. Reduced IO may also imply that less communication modules may be available. LTlib XC8 supports, as it is today, up to two modules per communication type so 2x UART, 2x SPI and 2x I2C. On different architecture the supported communications channel may be different.

Once all the configurations are matched for the new device, it is important to double check if the number of ports you have selected with NUMBER_OF_IO_PORTS, it is supported by the library module_IO. You can do that by checking the module_IO.c file of the architecture you are using. For the specific case, it is possible to see that there is the block:

```
#if (NUMBER_OF_IO_PORTS == 3)
```

which contains all the initialization for the 3 ports use case; thus we are fine for the PIC18F23K22 too.

Once all the configurations are updated, it is possible to update the LTlib file by adding:

```
#ifdef _18F23K22
    #include "PIC18LF46K22_config.h"
    #define CHIP_SUPPORTED
 #endif
```

Use Case C

A similar approach can be used for devices that are part of the same family but with a different memory size. In that case, beside the updates shown before, you have to make sure to update the following sessions as well, related to the Flash and EEPROM memory.

```
//****************************************
//                EEPROM
//****************************************
#define EEPROM_AVAILABLE

#define EEPROM_MODULE_SIZE 0x3FF

//****************************************
//                FLASH
//****************************************
#define FLASH_AVAILABLE

#define FLASH_ERASE_BLOCK 64
#define FLASH_WRITE_BLOCK 64

#define DEVICE_ID_ADDRESS 0x3FFFFE
#define DEVICE_ID_BYTES 0x02

#define DEVICE_REVISION_ADDRESS 0x3FFFFE
#define DEVICE_REVISION_BYTES 0x01
```

Use MCC with LTlib

If the device is not supported by LTlib, creating a new configuration file is an option. On the other hand you may not want to use the configurations out of the LTlib files at all, and use MCC or your configuration instead. This use case still let you benefit of the rest of the LTlib libraries that support different external ICs.

LTlib supports the option of working together with MCC by simply declaring the following line before LTlib.h gets included.

```
#define MCC_USED

#include <LTlib.h>
```

by doing that, LTlib will be set as follow:

- Use _XTAL_FREQ that is defined by MCC, thus the clock for the different peripherals will use it.
- All the MCU configurations defined via `#pragma` , inside LTlib, are ignored. The ones used by MCC will be used instead.
- Each library that automatically initializes the IC peripheral, will be ignored, thus via MCC you have to make sure that you define the pins as the library will use it.

You can still change the LTlib used pin by modifying the standard configuration file that gets loaded, MCC_default_config.h, nevertheless the MCC I/O settings and the configuration file must match.

If you use MCC you still get the following warnings from the LTlib library:

```
#warning (LTlib) LTlib is not tested on the microcontroller you have
selected
#warning (LTlib) LTlib has loaded the MCC default config file.
```

Index

1
16bit architecture....4
3
32 bit architecture....4
8
8 bit architecture....4
A
ANSI....4
C
C99....4
CHIP_SUPPORTED....16
Common include dirs....12
Compiler Toochains....12
COMPILER_XC8....15
conf....6, 9 e segg.
D
doc....6
Doxigen....6
E
ex....6
F
Free version....5
G
Global Options....12
H
HTML....6
I
I2C....17
IDE include paths....9
inc....6, 9 e segg.
L
LCD....4
LTlib.h....7
M
Maker version....5
MCC....19
MCLR....17
module_IO.c....17
module_xxxx....4
N
NUMBER_OF_IO_PORTS....17
P
PORTE....17
PRO....5
PRO version....5
Project Proprieties....12
R
RBPU....17
S
sch....6
SPI....17
src....9 e segg.
src/modules_xxx....7
src\modules_PIC_16_bits....10
src\modules_PIC_32_bits....11
src\modules_PIC_8_bits....9
T
Tool chain....12
U
UART....17
UART_CLOCK....8
W
WPUB....16
X
XC compilers....4
XC16....10
XC32....10 e seg.
XC8....9
XC8 Compiler....9
_
_XTAL_FREQ....19
#
#pragma config....8

Bibliography

[1] www.LaurTec.it: official site where you can download the LTlib software upgrades.

History

Date	Version	Author	Description
6. August 2023	1.1	Mauro Laurenti	• Updated GUI screenshot to MPLAB X 6.10. • Added: Use MCC with LTlib.
7. March. 2020	1.0	Mauro Laurenti	Original version.