Brief Introduction to Application Development with the Digilent chipKIT Max32 Development Board and the Microchip Toolchain

Paulo Pedreiras, Pedro Fonseca
Aveiro University
Department of Electronics, Telecommunications and Informatics
pbrp@ua.pt, pf@ua.pt

2021/09/07

Summary

This document reviews briefly the process of creating an application to be executed on the Digilent chipKIT Max32 Development Platform using the Microchip Toolchain (MPLAB IDE X and XC32 compiler). In addition to the steps necessary to create the project, the techniques for uploading the applications to the micro-controller and using programmers with debugging capabilities are also briefly presented.

History

Version	Date	Description
1.0	07/02/2013	Initial Release
1.1	15/03/2013	Addition of section with the multiOS (Java) PC application
1.2	07/12/2014	Addition of makefile (contribution of Ricardo Marau)
1.3	02/22/2017	Installing PLIB (XC32 1.4 or higher)
1.4	18/05/2018	Additional details about PLIB installation added
1.5	21/05/2018	New section on using a programmer
1.6	07/02/2019	Adapted to the Digilent Chipkit board Plib dependency removed Additional information on debugging
1.7	07/09/2021	Minor bug fixes and editing

Contents

1 Requirements

- MPLAB X IDE and XC32, both available from Microchip, should be installed.
  - MPLAB X IDE:
    `http://www.microchip.com/pagehandler/en-us/family/mplabx/#downloads`
  - XC32 compiler:
    `http://www.microchip.com/mplab/compilers`
- These programs are available for Linux, Windows and Mac OS and can be installed as freeware.
- The bootloader tools (PC application and bootloader itself) can be found at:
  `http://ppedreiras.av.it.pt/resources.htm` [1]
- This document is based on MPLAB X 5.10 and XC32 v2.15, but the procedure is similar for other versions

2 Creating a new project

1. Execute MPLab IDE
2. File + New project
3. Select "Microchip Embedded" + "Standalone project" + "Next" (Figure 1)

[1] The bootloader tools are included for information, only. They are not required if you are using the chipKIT Max32 with the chipKIT programmer.

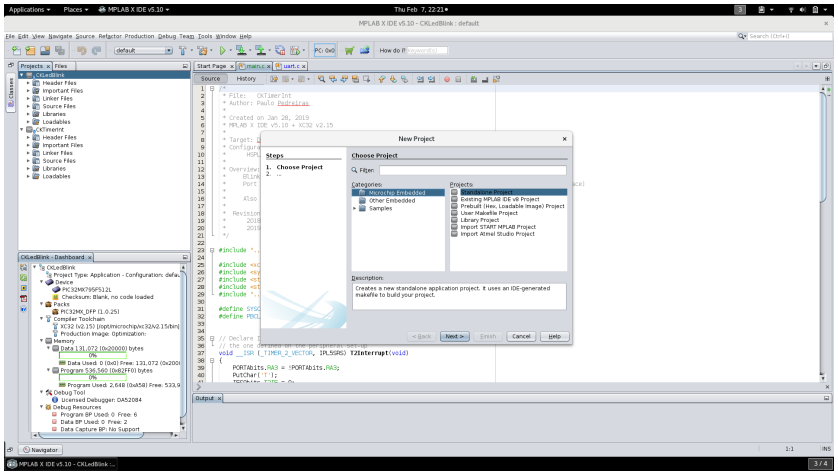

Figure 1: Create the project

4. Select PIC32MX795F512L microcontroller and “Next” button (Figure 2)

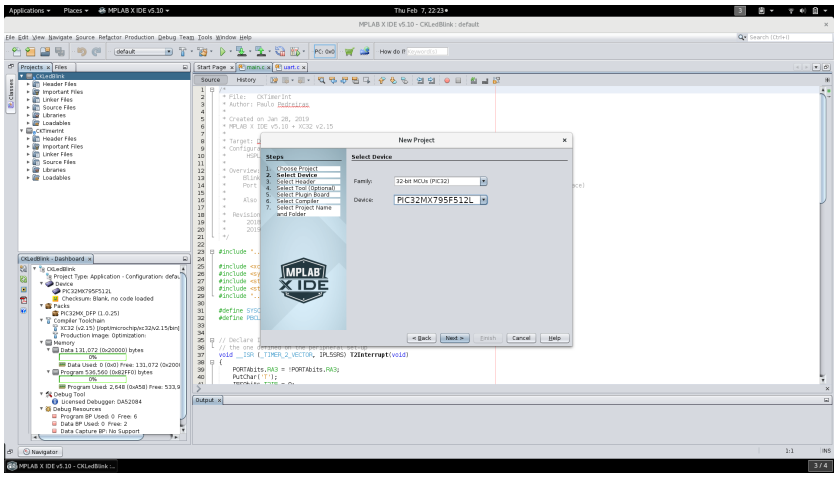

Figure 2: Select the microcontroller model. **Note - Must be an exact match!**

5. Leave the option in “Hardware Tools” to the default value, if using a Bootloader, or select the programmer.
   The Chipkit programmer shows up at the bottom, under “Licensed Debugger”. These options can be changed latter.
   Continue with “Next” (Figure 3)

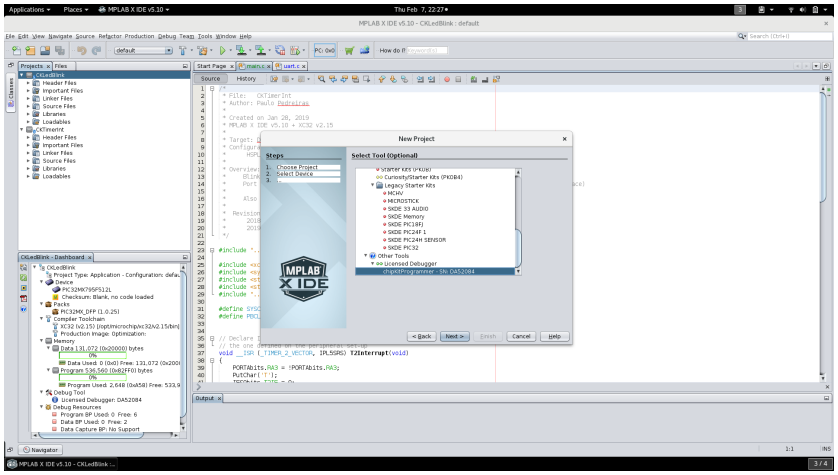

Figure 3: Select the programming hardware (if any)

6. Select the XC32 compiler and do “Next” (Figure 4)

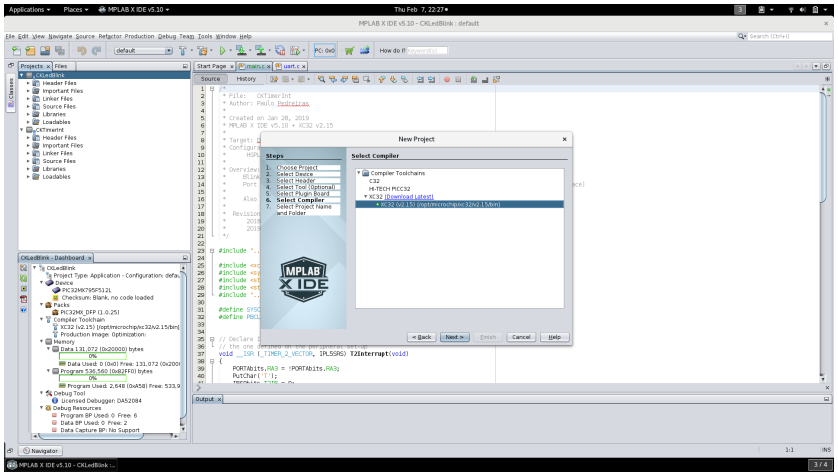

Figure 4: Select the compiler

7. Name the project (box “Project name”) and select a suitable folder for storing it (Figure 5)

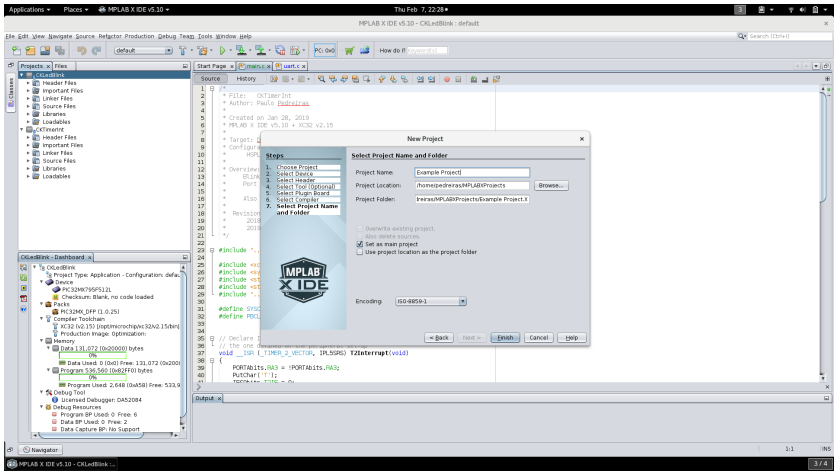

Figure 5: Type project name

8. If using the bootloader, **!!!and only in this case!!!**, copy to the project folder the file "linker-ck.ld", click with the right mouse button on "Linker Files", select "Add Existing Item" and then navigate to the project folder and select the linker file (linker-ck.ld) (Figure 6)

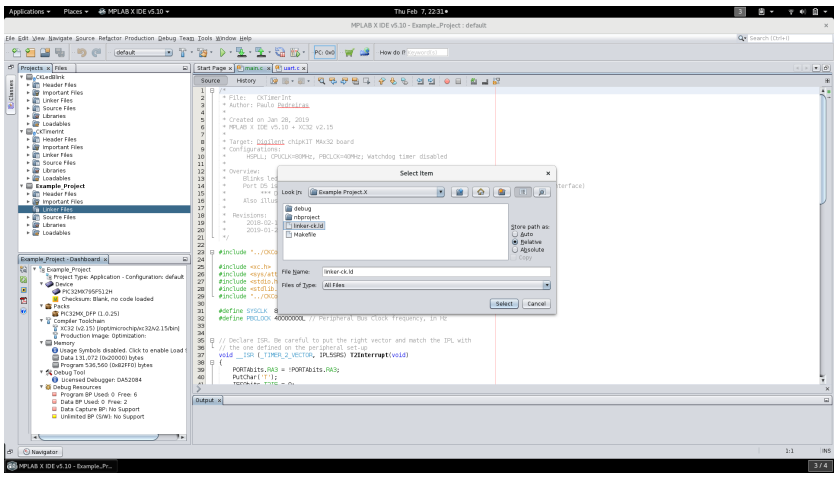

Figure 6: Copying the linker script file. **!!!Only if using a bootloader!!!**

MPLAB X IDE automatically creates a folder with the same project name, to which is added extension "X" (e.g. Example_Project.X).

9. Add Files ".c" ("Source Files" sub-folder) and ".h" (sub-folder "Header

Files"), as required.

These files can be created from scratch (using the right button on the appropriate sub-folder and choosing the "New" option) or be copied from other projects. In this latter case the files must be added to the project in a manner similar to that for the linker script.

10. The source files may be at any folder and added to the project at will. E.g. a personal set of libraries can be created by the programmer and used in several projects just by adding them to the project as explained above.

3 Editing and compiling

3.1 Microchip's Toolchain

Once the project is created, you should get a view similar to the one reported in Fig. 7

Using this IDE (based on NetBeans http://netbeans.org/) is quite intuitive. The screen is divided in four main parts:

1. **Projects**: managing projects (add, delete, and associate files with projects)
2. **Dashboard/Navigator**: variables, definitions, functions and header files used in the project, or project information (compiler, memory usage, ...)
3. **Editor**: text Editor
4. **Output**: activity output (compilation results, programmer messages, ...)

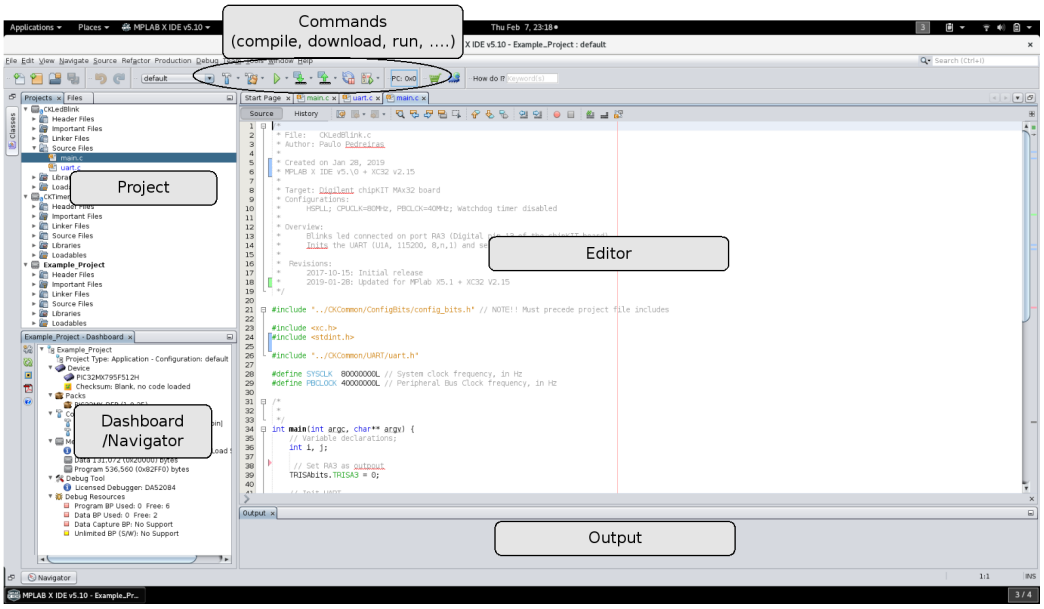

Figure 7: IDE organization

On top there is a set of menus and buttons. There are numerous functions (copy / paste text, search, formatting, recording, etc.). It is advised to read the documentation on the IDE, in particular the "MPLAB® X IDE User's Guide"(https://www.microchip.com/mplab/mplab-x-ide, "Documentation" tab).

3.2 Basic editor and Makefile

Creating applications can also be performed using a basic text editor. In this case, it is desirable to automate the process of compiling and linking in order to avoid typing long command lines, which in addition to consuming time, can induce errors, some of them little obvious and that can lead to wasting significant time (e.g. forget to compile a "c" library, leading to the inclusion of old code).

The tool of choice in Linux environments to automate such processes is the "Make" utility. This tool allows automating entirely the process of compiling and linking, having as input a file, usually called "Makefile", which defines how the build should be carried out (compiler command line, application name, files that compose the project, ...).

In "http://ppedreiras.av.it.pt/resources/detpic32-bootloader/blinker.tgz"

you can find a sample “Makefile”, developed by Ricardo Marau(rmarau@gmail.com), which can be easily adapted to new projects.

4 Loading an application on the development board using a bootloader

By default, the compiler creates on the host computer an “.hex” file that must then be load into the development board. By default, the “.hex” file is built in folder “[PROJ_FOLDER]/dist/default/production/” with name “PROJ_NAME.X.production.hex”, where “PROJ_FOLDER” is the folder where the project was created and “PROJ_NAME” is the name given to the project.

There are two possibilities for loading the “.hex” into the development board. One requires the existence of a bootloader, which must be previously programmed on the development board, and the second one uses the programmer directly. This section describes the use of the bootloader. As mentioned before, the bootloader is not required if you are using a programmer, such as PICKit3 from Microchip or Digilent’s chipKIT programmer.

4.1 Microchip bootloader

Microchip provides a bootloader that can be adapted to several PIC32-based boards, such as Digilent chipKIT Max32 or the DETPIC32 board. This software, which includes the source code, can be downloaded from *ww1.microchip.com/downloads/en/AppNotes/AN1388_Source_Code_011112.zip*

The Microchip software includes a PC application, built with Visual C.Net for Windows OS (“PIC32UBL.exe”). This application can also be used under Linux via Wine (Windows Emulator, http://www.winehq.org/)

If using Linux, you must:

1. Set up a COM port:
   in “ /.wine/dosdevices/”create a link between “com1” and the linux device
   E.g. ln -s /dev/ttyUSB0 com1

2. Add yourself to the dialout group to get appropriate permissions:
   sudo usermod -a -G dialout $USER
   where $USER is your username

3. After these settings, simply type
   "$wine [PATH_TO_BOOTLOADER]/PIC32UBL.exe", where "PATH_TO_BOOTLOADER
   is the folder where the "PIC32UBL.exe" is stored.

Regardless of being using Windows or Linux, when the the above program is executed you see the window shown in Fig. 8.

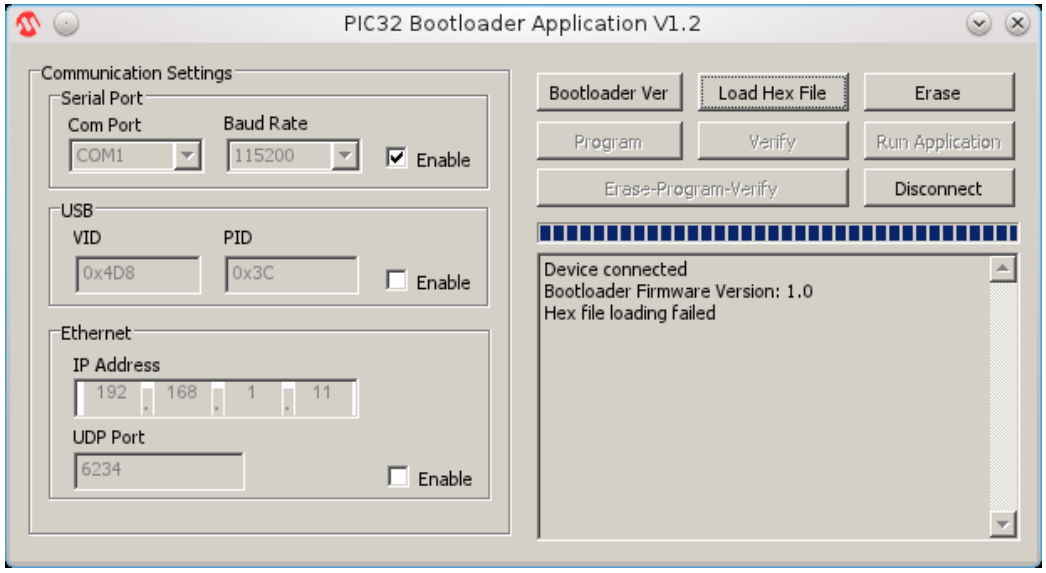

Figure 8: Bootloader interface

The loading process consists of the following steps:

1. Press the "Connect" button

2. Press the reset button and the following message should appear:
   Device connected
   Bootloader Firmware Version: x.y

3. Press the "Load Hex File" button and select the ".hex" file intended (see note above about the local where ".hex" files are placed by the compiler). You should see the message:
   Hex file loaded successfully

4. Press the button "Erase-Program-Verify"
   You should see the message:
   Flash Erased
   Programming completed
   Verification successful

5. Press the button “run application”or make reset the board to run the application

6. After this process you should press the “Disconnect” button.

4.2 Java PC application

It is also available a Java application, which runs on Windows, Linux and Mac OS. This variant has the advantage of running on the command line, automatically opening and closing the communication port, thus being more practical.

Its use is very simple:

```
\[PATH\_TO\_JAVA\] java -jar bl.jar -p COM\_INT -b -e -r -l XXXXXX.hex
```

where:

- COM_INT: identifying the serial interface as seen by the operating system
- - b: returns the version of the bootloader
- -e: erase the memory
- -r: runs the program
- -lXXXXXX.hex: file identification charging (Intel hex format)
- -h: help

For instance:

```
$/usr/java/jre1.7.0\_15/bin/java -jar bl.jar -p /dev/ttyUSB0 \
        -b -e -r -l blink.hex
```

5 Loading and debugging programs using a programmer

The process of loading, and particularly debugging, application code is greatly simplified by using a programmer. Programmers are hardware devices that communicate between the host computer, running MPLab IDE, and the micro-controller board. There are several programmers available for the PIC micro-controllers. They can be either made by Microchip (such as ICD3, ICD4 or PICKit 3), which are the originals, or by other manufacturers (such as Digilent's chipKit). All of them follow the same communication protocol, called ICSP (In-Circuit Serial Programming) and, in many cases, are interchangeable. ICSP allows to program a device when it is assembled to the board ("In-Circuit") using a serial line.

The programmer connects to the development board by a specific connector. Usually, this is either a RJ-11 connector, for ICD3 and ICD4 programmers, or a 5 or 6 pin header connector, for PICKit3 or chipKit. Figure 9 illustrates the connection of a chpKIT PGM to the chipKIT Max32 development board. Note that ICSP communicates through pins PGECx and PGEDx, so you should take care of what you connect to these lines, as it may interfere with the ISCP hardware and cause ISCP to malfunction. To be on the safe side, do not connect anything to these pins. Further details on ICSP and its use can be found in the PIC32MX Reference Manual.

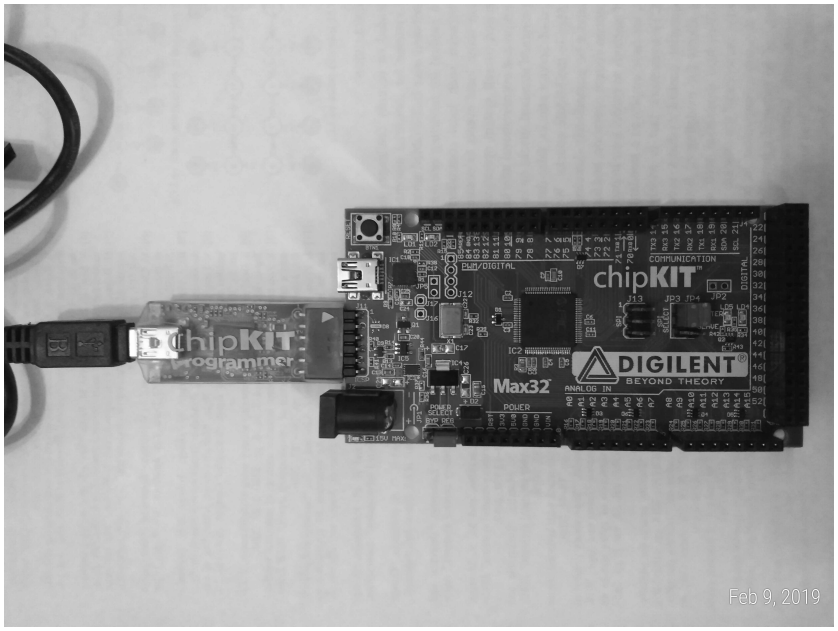

Figure 9: Connecting the chipKIT programmer to the chipKIT Dev Board

The first step is, evidently, write and compile the application code. In MPLAB X, there are diverse build options and, correspondingly, buttons, shortcuts and menu items to build (production build) a project and to debug (debug build) a project.

The production build, as the name implies, is intended for code that goes to production, i.e., that is terminated. It generates the more efficient code. Conversely, when performing a debug build, the IDE will set the configuration bit to allow debugging of the project by a debug tool, such as the MPLAB ICD or the chipKIT PGM. In MPLAB X IDE, memory is reserved for your debugger (if selected) only when you perform a debug build. Moreover, a debug build also sets a preprocessor macro called `__DEBUG`. This macro is not defined for production builds and can be used select code developed specifically for debugging (conditional compiling) by using `#ifdef` directives, etc.

Compiling and uploading the application code to the development board can be done through the programming toolbar.

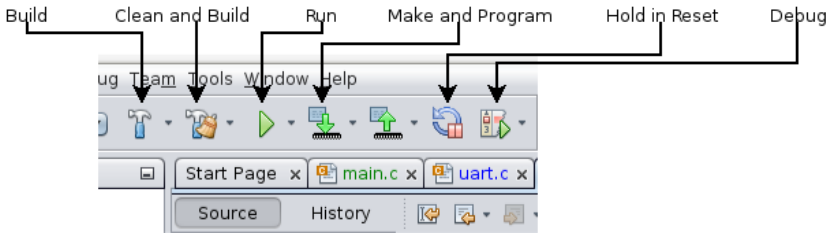

Figure 10: Build and Debug toolbar elements

- The **Build** button allows to compile and link the project (F11). It identifies and compiles only files that have been modified, to improve the build process efficiency. The button allows to build for production (default) or for debugging.
- The **Clean and Build** button is similar to the Build button, except that it compiles all files from scratch (Shift + F11).
- The **Run** button builds the project, loads the ".hex" file on the development board and starts its execution.
- The **Make and Program** button builds and downloads the ".hex" file to the development board. It allows selecting the Production of Debug modes.
- The **Hold in Reset** button toggles the device between Reset and running.
- The **Debug** button allows to build, program and execute an application in debug mode.

Note that the above commands operate on the "Main Project". You can set the main project by right-clicking on the project name on the Project Window.

Most of the above commands are self-explanatory. We will now give a few more hints on debugging.

Debugging is considerably facilitated when using a programmer. It allows controlling the execution of the code in the microcontroller, namely define break points, start and stop the program execution, executing the program set by step, examine the contents of variables, memory and special function registers, among many other possibilities. The utilization of these functionalities is fairly intuitive. We will illustrate its use through a few examples of

typical operations.

The easiest way of starting a debug session consists in issuing the "Debug" command, as explained above, to build, load and start execution of the application. After doing that, the debug toolbar becomes accessible. This toolbar allows the following actions (left to right):

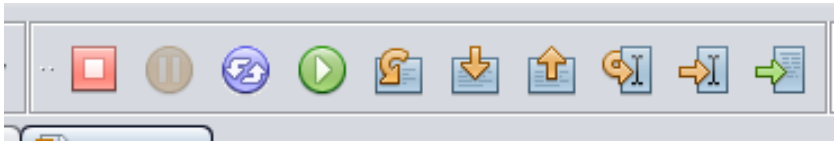

Figure 11: Debug control toolbar

- **Stop the debug session:** interrupts the debug session.
- **Pause execution:** suspends the program execution. It can be resumed at any time, using the continue of step commands (see below)
- **Reset the execution:** resets the program execution to the initial state
- **Continue the execution:** resumes program execution, when in the suspended state
- **Step over:** executes the line. If the line calls a function, the function is completely executed before control returns top the debugger
- **Step into:** executes the line. If the line calls a function, the function is executed step by step
- **Step out:** the code is executed until the function returns to its caller
- **Run to cursor:** the program is executed until reaching the line in which the cursor is placed
- **Set PC at cursor:** the program will execute at the line where the cursor is placed
- **Focus cursor at PC:** the cursor is moved to the PC

A breakpoint can be inserted at any line by clicking of the line number, on the left side of the editor window. For example, if we want to debug the CKLedBlink example, we can insert a breakpoint at line 54, so the execution

gets interrupted each time the output state is toggled. After inserting the breakpoint and starting a debug session, we get into the scenario shown in Figure 12.

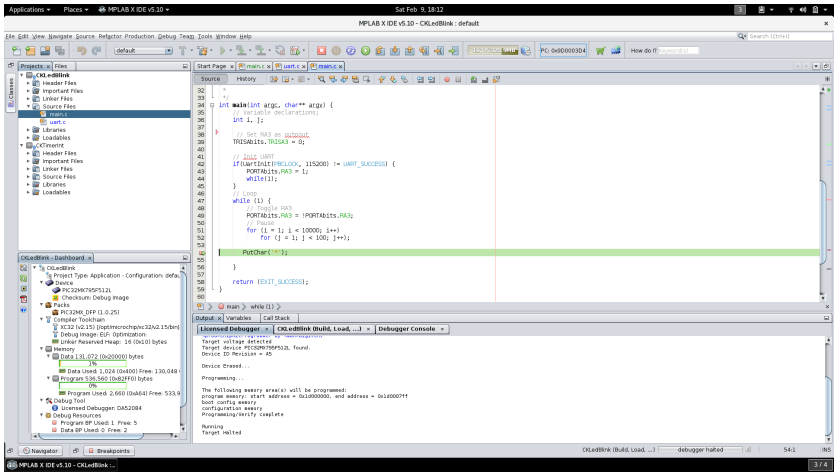

Figure 12: Starting a debug session

At this point, issuing the **Step into** command, the execution resumes and is suspended at the first line of the called function (*PutChar()*), as shown in Figure 13.

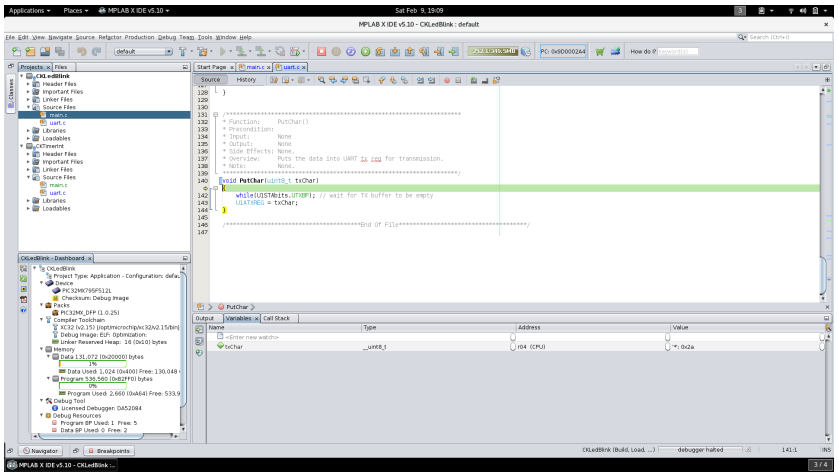

Figure 13: Stepping into a function

At this point you can execute the function step by step or e.g. use the command **Step out** to execute the PutChar function until it ends. In any

case the execution is again suspended at the line that follows the call of *PutChar()*, as shown in Figure 14

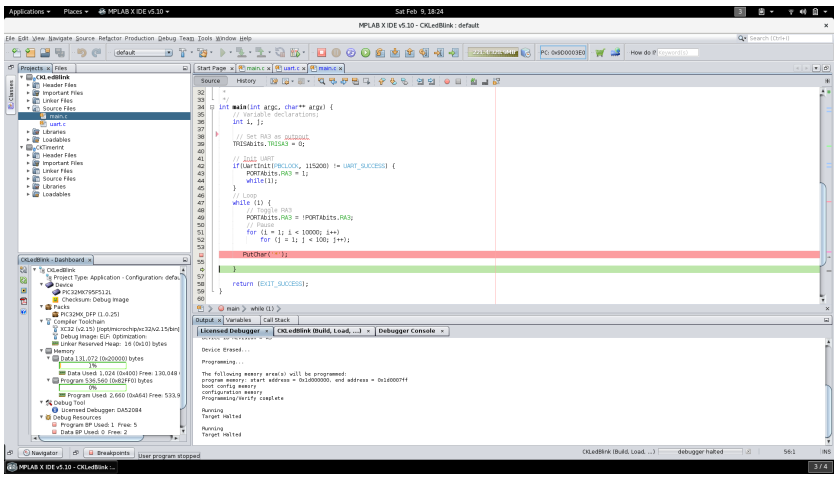

Figure 14: Stepping out of a called function

At this point if you issue the **Continue** command, the execution will continue and will stop again at line 54, due to the breakpoint.

Now lets remove the breakpoint from line 54 and add a new one at line 52 ("for" loop in "j" variable). Then resume the execution. After it stops at the breakpoint, place the cursor over variable "i" and wait a few moments. It will show up a context box that shows the variable address and its value, as shown in Figure 15.

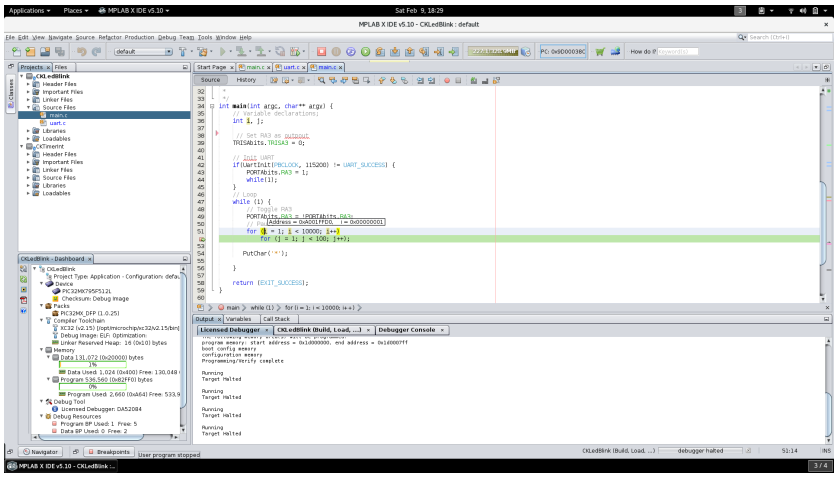

Figure 15: Showing variable info by cursor placement

It is possible to show permanently the contents of variables, SFRs, etc. This is called a “Watch” and shows up bellow the editor, selecting the “Variables” tab. For adding a variable watch right click on the variable and from the context menu choose “New Watch”, as shown in Figure 16. Then confirm with OK. The watch is added to the “Variables” tab. The variable value is automatically updated as you step on the program.

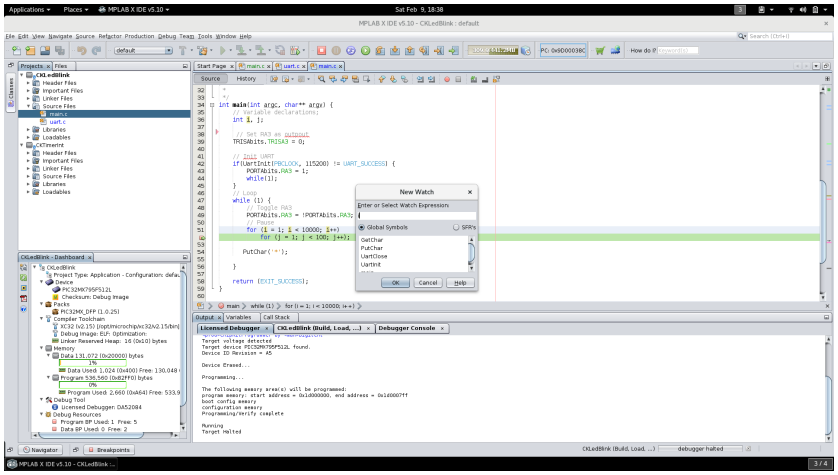

Figure 16: Adding a watch

Adding a watch to a SFR is done in the same way. You can use the shortcut CTRL + SHIFT + F9. This allows e.g. to check the contents of a Timer register, the status of an interrupt flag or port, etc.

The debugger is an extremely valuable tool for programmers, and can save significant amounts of development time. You are advised to look learn about it. Additional information is available e.g. at the ”MPLAB X User’s Guide“ (http://ww1.microchip.com/downloads/en/devicedoc/50002027d.pdf), chapters 3 and 4.

6 Acknowledgments

The original port of Microchip’s bootloader for DETPIC32 platform was made by:

- Diego Mendes - diego.mendes@ua.pt
- Cristóvão Cruz - cac@ua.pt

- Rómulo Antão - romantao@gmail.com

The “makefile” and associated example was developed by:
Ricardo Marau - rmarau@gmail.com