ALL ABOUT THE DIOLAN PLUS BOOTLOADER.

The Diolan bootloader is an open source bootloader originally written for the PIC18F2455. The home of this bootloader is here. http://www.diolan.com The credit for bulk of the work for this bootloader rightly belongs to the original author at Diolan and so to much credit for releasing it as open source under the GNU licence. Details on the scope of the original work can be found at the above web site as too can some documentation required for using the host side software. This is not covered by this documentation as there are no differences when using the host side software.

The **Diolan Plus bootloader firmware** modification is chiefly the work of James Robertson but with important work also done by Ian Lesnet of dangerous prototypes fame.
http:www.dangerousprototypes.com

Ian was responsible for altering the PIC18 code so that it does not require the use of the PIC18 extended instruction set. Ian also made a few more minor changes to make it compatible with the PIC18F2550 as that was the target PIC of choice for dangerous prototypes.

After several people expressed interest in porting the Diolan bootloader to the PIC18F14K50 on the microchip forum the idea took seed in the mind of James Robertson who then ported the work of Ian to become the Diolan Plus bootloader.

So what has changed from the original version in the plus version?

Chiefly these are the differences.

- The Diolan Plus version now easily supports all the USB PIC18 family (except the J-series). These may be added later.
- Supported parts by name: 18F14K50, 18F13K50, 18F2450, 18F2455, 18F2458, 18F2550, 18F2553, 18F4450, 18F4455, 18F4458, 18F4550, 18F4553
- The port came from the dangerous prototypes version. As such it does not implement the xtea encryption as this generally is not required for hacker and hobbiest level work that this port is aimed at.
- The plus version is half the code size of the original version. This is mostly because it does not implement the xtea encryption. The bootloader will now fit in the for the purpose boot block of all the target PICs.
- The code along with the linker script was adjusted to be compatible with either the standard PIC18 instruction set or the extended instruction set.
- The code has been reorganized somewhat to be modular and clearly separate user settings for ease of use and maintainability across many applications.
- The code has been altered to allow for the **watchdog timer** to be enabled in the config words without interfering with the bootloader operation.
- The code has been designed to assist newbies in setting up the bootloader and effectively also

becomes a tutorial on the PIC18 USB oscillator settings. This is something that has confused many a newbie.

- A single custom for the bootloader linker script has been provided and this is compatible with all target PICs and operating modes.

What hasn't changed.

- Like the original version the Plus version is a FULL SPEED HID device. LOW SPEED is not supported (or desirable.)
- The same PC side host software is used as there is no change to the bootloader protocol. The same documentation for the old Diolan bootloader is still applicable to the plus version and will need to be downloaded from the Diolan web site.

GETTING STARTED

The prerequisite assumed is that you have MPLAB 8.x with the MPASM and MPLINK tools installed. Please consult the microchip documentation for details as installing MPLAB etc. is out of the scope for this documentation.

An MPLAB project file is provided in the Diolan Plus package. Using this project file is simple.

To start unpack all the files into a folder of your choice. Click on **plus_bootloader.mcp** to fire it up in MPLAB.

In MPLAB under **configure -> settings**, choose your target PIC.

At this time skip ahead and read about the three USER_ files that you may need to configure. Once that is taken care of build the project and program it into your target PIC. All going well you now should have a working bootloader.

USER_ FILES HOW TO...

Once we have MPLAB set up and our project loaded there are just a few more details to take care of before we build and burn. To do those we goto the three **USER_** files and make any required alterations. Remember, only the three **USER_** (.inc) files need to altered to fully customize the Diolan Plus bootloader.

VID and PID SETTINGS (USER_vid_pid.inc)

(VID = Vendor ID, PID = Product ID)

All USB devices require a VID/PID pair. The **USER_vid_pid.inc** file is solely used to define the VID/PID pair that the bootloader will enumerate with. Typically, with a microchip product the VID is 0x04d8 and a suitable PID can be applied for directly from microchip for small (< 10,000 units) productions. Some open source houses already have a PID allocated for use with this bootloader. If you are associated with them and ask nicely they may let you use it, keeping in mind that the sub licence for the PID is limited to only 10,000 units.

CONFIG WORDS (USER_configwords.inc)

Config words may vary project by project. They allow for the PIC to be customized to the hardware and application. Because the bootloader uses the same config word settings as the main target application some care must be exercised in selecting config settings that are compatible with both. Fortunately, in most cases there is little area of conflict as usually they both require the same USB settings and run on the same hardware.

The Plus version bootloader is written to be more flexible in dealing with user config settings in that it now will work with both the legacy instruction set and the extended instruction set. For now we are only going to concern ourselves with the legacy mode instruction set . The extended instruction set mode is an advanced topic and will be covered later.

Additionally the Plus version also tolerates the watchdog timer being enabled making it virtually fully compatible with all user firmware settings. Really, the plus version completes the original Diolan bootloader.

The **USER_configwords.inc** file contains two templates for suitable config word settings. One template for the PIC18F13K50/14K50 and the other applies to all other supported PIC18s.

As per usual you the user will alter the config word settings as required by your end application.

However, within these templates there are sections clearly marked as **DO NOT ALTER.** This means, in clearer English, **DO NOT ALTER!** There are also a few single lines that are similarly marked and again **DO NOT ALTER.**

All the config word settings that are ***essential*** to the bootloader operating correctly are ***quarantined*** and especial care has going into recoding the original bootloader so that it will remain compatible with any and all non-quarantined config word settings that the user may require.

In addition to the quarantined config word settings there are sections marked as **"STRONGLY RECOMMEND."** Please follow this advice and do not alter these settings **unless you really know what you are doing.** Altering any of these settings will put the bootloader at very much greater risk of accidental "bricking." If you are an advanced user you will no doubt understand this and know all the caveats about rewriting config words and leaving the boot block unprotected etc.

If you do not understand the above caveats then that is no problem as all the hard work has been done for you. Just leave the settings as is.

NOTE! No claim is made that the quarantined and recommended are the only correct config word settings required to get the bootloader to work. Consult the PIC data sheet for further settings and how they relate to your particular hardware.

For further newbie assistance the **USER_configwords.inc** contains some convenience features. One of the great puzzles confronting a newbie with what all those oscillator settings should be and what crystals can be used. The leg work here has been done for you and all that is required is for you to read within **USER_configwords.inc** what crystal frequencies are suitable for the target USB PIC family and for you to choose one and enter its two digit numeric value right at spot where you see the **NN** below. (This setting is found at the top of the **USER_configwords.inc** file.)

CRYSTAL equ NN ;(Mhz)

Once you have selected your crystal frequency and entered the value as shown above the assembler directives will choose the right config word settings for you. However the point of this is not for you to be lazy about what is happening here, rather you might look at both the directives and the config word settings, a long with the data sheets, and learn two things at once.

Further to this the **USER_configwords.inc** with also help you configure the CPU clock speed that you required and in a very similar fashion. Again all the available options are listed for the different PIC families and again all that is required is for you to enter the value of your required CPU speed right where it says:

CPUSPEED equ NN ;(Mhz)

(Note for advanced users. Radix does not apply here as the values are in fact only being used as labels.)

ALL OTHER SETTINGS

With regard to all other config word settings, these are determined by the requirements of the target application and not the bootloader. While not every possible combination of these optional settings has been tested, it is the intent by design that they will work, (or not work!) with the bootloader equally as well as they will do with the end application. That is the same requirements and caveats apply equally and they are not bootloader specific and any way.

BOOTLOADER ENTRY (USER_POR_EntryMacro.inc)

This is our third and last USER_ file and where we enter our bootloader entry code.

As this USER_ file's name suggests, the entry code at reset is a **MPASM MACRO** that is included in the boot.asm main source code file. The reason for making it a MPASM macro in a separate USER file is to maintain the code base in an easy to use and port format. With this method, all the bootloader source files other than those prefixed with USER_ in the file name are consistent across all applications and therefore can be maintained and updated in a modular fashion.

Basic Instructions for the entry macro:

You bootloader entry macro is just that. ***Yours.*** As such it is entire up to you to write the code to enter the bootloader. There are numerous methods used like testing for a “bootstrap” I/O pin, testing that two I/O pins follow each other or using a EEPROM mark left there by the user application code.

(Note! Not all the PICs supported by the plus version have EEPROM. I.E PIC18F2450, PIC18F4450. If and when the plus version supports the J series of PICs, they too do not have EEPROM.)

You can consult the original Diolan bootloader documentation for details on using the EEPROM mode and there are many examples on the web for other possible methods.

Once you have your code ready we are going to make that code a **MACRO** with a standard name that will be included in the file **boot.asm.**

So to begin, create a file called **USER_POR_EntryMacro.inc** or use the existing template file provided.

Cut and paste this template into this file :

```
PORentrytest macro ; Start of macro (This line not indented)

        ;  Your code here to decide what to run bootloader or application
        ;  For simply testing the bootloader you can leave this empty but you will not be able to run
any  user code of course...

        ;  If bootloader required then

        bra     bootloader                  ; Always in range of a relative branch

        ; else

        goto    APP_RESET_VECTOR            ; Run Application FW @ 0x800

        endm ; End of macro
```

The important things about this file are:

File must be named **USER_POR_EntryMacro.inc**
Macro must be called: **PORentrytest** and begin with **PORentrytest macro**
Macro ends with the line: **endm**
Must **NOT** have a END assembler directive in the file.
The label to branch to if bootloader is to be invoked is: “bootloader” (without the quotes.)
You can use the label **APP_RESET_VECTOR** to branch (or goto) the main user application @0x0800 if the bootloader entry test failed. It is recommended that you use this label.

Once that is completed, it is time to build and program the bootloader into your PIC. Hopefully the above instructions where clear and complete enough for you to have success first go. If not then carefully review what you have done and try again.

ADVANCED TOPIC USING THE EXTENDED INSTRUCTION MODE

The original Diolan bootloader use the PIC18's extended instruction mode and this was the only option available. Because there are no separate config words for the bootloader and user app, this forced the user app to also use the extended instruction set mode and as this mode while providing eight new instructions does so at the price of eliminating the lower part of the access bank.

For most uses this is just not a good trade off and the majority of code written for the PIC18 only uses the standard instruction set along with the lower access bank. The Diolan Plus version of the bootloader was rewritten to use the standard instruction set and give the access bank back to the user.

However, what if you have a rare case where in fact you want to use the extended instruction set and with a PIC18 not supported by the standard Diolan bootloader?

Fortunately it is still possible to use the extended instruction set mode and there are only a couple of things you need to do.

- Enable the **XINST** in the config word.
- Force MPASM and MPLINK to the extend mode.

The first point is obvious and needs no further explanation. For the second point the only way the author found to force MPASM to compile for the extended instruction set in MPLAB was to directly edit the .mcp file.

Open the .mcp project file in a text editor and look for this line:

suite_state=

Change this to:

suite_state=extended-mode

If you are using MPASM on the command line then the extended instruction set is enabled via a command line switch. **/Y**

HOW THE EXTENDED MODE WORKS WITH THE DIOLAN PLUS BOOTLOADER

This is just a short technical note for more advanced users who may be wondering how the Diolan Plus bootloader can work with either the standard instruction set or the extended instruction set without alteration. Here's how.

Firstly to work with the standard instruction set no extended instructions are used. These instructions were removed by Ian Lesnet at Dangerous Prototypes. As the Diolan Plus bootloader only uses the standard instruction set, and this is a subset of the extended instruction set, we now have instruction code that is compatible with both modes.

That still leaves the problem with the missing access bank. To over come this we initialize the BSR SPR to point to bank 0 and we force MPLINK to encode the PIC18 instructions with the BANKED bit. So it is the linker that is cleaver enough to do the hard work for us.

To get the linker to do this, the define for the lower access bank is removed from the linker script. Now it has to treat all RAM locations other than the upper access area as being BANKED. As we have pointed the BSR to bank 0 that contains all our RAM usage the problem is solved.