# Using the PIC18F4620 or PIC18F4520



Historically, MIOS was developed to run on a core module stuffed with a PIC18F452. Recently, the PIC18F4620 has become available. It is near code-compatible with the 452, but features a significant increase in RAM/EEPROM/Codespace. See the PIC18F4620 page for details.

The following are intructions on converting old apps, and developing new apps, to run on the PIC18F4620. Small changes to the procedure make it compatible with the PIC18F4520 also.

## **OS Layers**

MIOS v1.9b or above is required. You will need to download the MIOS source from The uCApps.de Download Page or Directly. I recommend checking the first link for the latest version, as the '4620 is current in beta.

The Bootloader and MIOS recompile steps which follow should not be necessary for most cases of '4620 use, as these components are now available precompiled and packaged in a zip file hosted on uCApps.de Instructions follow for reference only, or for '4520 use.

## **Bootloader**

Bootloader v1.2, which is packaged with MIOS v1.9 and up, will need to be recompiled as follows:

- Extract the MIOS source files from the zip
- Edit bootloader\main.asm
- Change

#define PIC\_DERIVATIVE\_TYPE 0

То

#define PIC\_DERIVATIVE\_TYPE 1



Last update: 2011/11/18 12:14



## **MIOS**

The MIOS Operating System itself must also be compiled, as follows:

- Edit src\mios.h from the MIOS source files
- Change

#define PIC\_DERIVATIVE\_TYPE 0

То

#define PIC DERIVATIVE TYPE 1

- Compile the project Fix Me! wiki link
- also see: Compiling the MIDIbox source on Linux using GPASM (Linux, Mac) instead of MPLAB (Windows)
- Upload the hex file with MIOS Studio Fix Me! wiki link

Please note that the above instructions should work for PIC18F4520 also. The only difference is that the PIC DERIVATIVE TYPE should be '2', not '1'. This stands for all of the following instructions.

# **Application Layer**

Once your PIC18F4620 has the Bootloader burned onto it, and MIOS uploaded, you are ready to upload your application. A few modifications may be required:

## **Migration**

If you have an existing ASM-based application, which is designed for MIOS v1.8 or lower, then you will need to migrate the application to support MIOS v1.9

- Extract the 'migration' folder from MIOS source zip file
- Overwrite the files contained in the source of your application.

Take note that this may overwrite customisations you have made to your application, so please take a backup first, and a copy for comparison with the new files.

## **ASM**

If your application is either:

- 1. a freshly migrated application (as above)
- 2. a brand new ASM-based project based on a skeleton >= v1.9
- 3. an ASM-based application which already requires MIOS v1.9 or greater (like MBSID v1.7303)

Then the following steps are required:

- Edit mios.h in the source of your application
- Change

#define PIC DERIVATIVE TYPE 0

То

#define PIC\_DERIVATIVE\_TYPE 1

- Compile the project Fix Me! wiki link
- Upload the hex file with MIOS Studio Fix Me!

### C

If your application is C-based, then the following steps are required. Some are optional recommendations, as noted.

#### Note on compile errors

When compiling your C-based application, you may see an error such as this:

```
Linking project
warning: processor mismatch in "_output\mios_wrapper.o"
```

This error is caused by SDCC compiling the application for the PIC18F452. Fortunately, the code is

compatible between the two chips, so this error can be ignored. Thanks to bill, for having the guts to put the app on his PIC, and confirm that this was the case;)

## **Header and Library**

In the case that you should need to take advantage of the additional EEPROM on the newer PICs, the following alterations to the library and header are necessary:

- Edit pic18f452.c in the source of your application
- Change

```
sfr at 0xfa9 EEADR;
sfr at 0xfab RCSTA;
```

Tο

```
sfr at 0xfa9 EEADR;
sfr at 0xfaa EEADRH;
sfr at 0xfab RCSTA;
```

- Edit pic18f452.h in the source of your application
- Change

```
extern __sfr __at 0xfa9 EEADR;
extern __sfr __at 0xfab RCSTA;
```

To

```
extern __sfr __at 0xfa9 EEADR;
extern __sfr __at 0xfaa EEADRH;
extern __sfr __at 0xfab RCSTA;
```

Note that the filenames stay as pic18f452.\*, regardless of the PIC model we are actually using. For our purposes, SDCC considers the '4620 to be the same as a '452.

## **C-Wrapper**

The C-Wrapper will need to be edited as follows:

- In the source of your application, edit mios\_wrapper\mios.h
- Change

```
#define PIC_DERIVATIVE_TYPE 0
```

To

```
#define PIC_DERIVATIVE_TYPE 1
```

If you want to use this function, you may want to apply a small fix to the DEC2BCD Helper:

- In the source of your application, edit mios wrapper\mios wrapper.asm
- Change

To

```
global _MIOS_HLP_Dec2BCD //The low byte is already in W

movff FSR0L, FSR2L //These guys
movff PREINC2, MIOS_PARAMETER1 //Put the high byte in

MIOS_PARAMETER1. Yay!

goto MIOS_HLP_Dec2BCD
```

## **Linker Script**

Modifications should be made to the linker script in order to take advantage of the additional capabilities of the 4620/4520. If you are using a standard, PIC18F452-based application, these steps should not be necessary. These procedures are intended for applications being developed which will require the additional capabilities of the newer PICs.

#### **Extend Codepage**

Both the 4620 and 4520 have extended code memory. To utilise this fully, make the following alterations:

- In the source of your application, edit project.lkr
- Change

CODEPAGE	NAME=page	START=0x3000	END=0×7FFF	
----------	-----------	--------------	------------	--

To

CODEPAGE NAME=page	START=0x3000	END=0xFFFF
--------------------	--------------	------------

If you are using a GLCD with your PIC18F4620 the GLCD Font of the MIOS will be overwritten by this change though, as it lies in the range of 0x7C00-0x7FFF. There are <u>two</u> approaches to prevent this (also see forum thread <a href="http://www.midibox.org/forum/index.php?topic=7540.0">http://www.midibox.org/forum/index.php?topic=7540.0</a>):

• To leave out the Font space in the PIC's code memory, change

CODEPAGE NA	AME=page	START=0x3000	END=0×7FFF
-------------	----------	--------------	------------

To

CODEPAGE	NAME=page0	START=0x3000	END=0x7BFF	
CODEPAGE	NAME=page1	START=0x8000	END=0xFFFF	

**or** try this solution from TK:

- copy the mios\_glcd\_font.inc file from the MIOS release into your application directory, rename it to mios\_glcd\_font.asm
- add following code to the file header:

```
LIST P=PIC18F4620, R=DEC

DEFAULT_FONT code

FONT_ENTRY MACRO width, height, x0, char_offset

dw ((height) << 8) | (width), ((char_offset) << 8) | (x0)

ENDM
```

- change the "org" (start address) from 0x7cfc to 0xfcfc
- add a "END" at the file footer
- add the new .asm file to the MAKEFILE.SPEC (behind the MK SET OBJ statement)
- change the font pointer within the Init() routine:

```
void Init(void)
{
MIOS_GLCD_FontInit(0xfcfc);
}
```

• if you are working under MacOS or Linux, type "perl tools/mkmk.pl MAKEFILE.SPEC; make", under DOS just type "make"

The first approach has the advantage, that it isn't required to upload the font again and again with each program update. The second approach that new fonts can be inserted into the project in a similar way. Please see the forum article mentioned above on instructions how to use labels in combination with fonts.

#### **Add Databanks**

In order to give our application the ability to recognise all that lovely, lovely RAM in the newer '4620 and '4520 PICs, one or a mixture of the following options is required:

#### **Standard Bank Size**

- In the source of your application, edit project.lkr
- Change

DATABANK NAME=miosra	am_u START=0x380	END=0×5FF	PROTECTED
ACCESSBANK NAME=access	ssfr START=0xF80	END=0×FFF	PROTECTED

To

DATABANK DATABANK	NAME=miosram_u NAME=gpr6	START=0x380 START=0x600	END=0x5FF END=0x6FF	PROTECTED
DATABANK	NAME=gpr7	START=0x700	END=0x7FF	
DATABANK	NAME=gpr8	START=0×800	END=0x8FF	
DATABANK	NAME=gpr9	START=0×900	END=0x9FF	
DATABANK	NAME=gpr10	START=0×A00	END=0xAFF	
DATABANK	NAME=gpr11	START=0×B00	END=0xBFF	
DATABANK	NAME=gpr12	START=0xC00	END=0xCFF	
DATABANK	NAME=gpr13	START=0xD00	END=0xDFF	
DATABANK	NAME=gpr14	START=0×E00	END=0xEFF	
DATABANK	NAME=gpr15	START=0×F00	END=0xF7F	
ACCESSBANK	NAME=accesssfr	START=0xF80	END=0xFFF	PROTECTED

#### **Extended Bank Capacity**

**Be Careful doing this!!** This can do weird things like make variables always = 0 where the memory that variable is stored in, crosses boundaries of 256b blocks. Do not make banks any larger than required, and try to let them begin at round numbers (100, 200, 300,...). This still might not even work. - stryd\_one

The above change will enable SDCC to allocate the variables in your application to any of the specified banks above. The very observant among you may have noticed that these banks are 256 bits each.... So what happens if you want to use a variable which is greater than 256 bits in size, such as a large array, or string of characters? For this, you will need to create a bank of extended size, and you will need to direct your application to use that bank to store your large variable.

In order to create memory banks of extended capacity, it is necessary to section off a greater range than those given above. A good way to go about this is to combine two or more of the default banks. The following are examples of this.

Making a single, 512-bit bank:

DATABANK NAME=miosram_u // DATABANK NAME=gpr6 // Remove this bank	START=0x380 START=0x600	END=0x5FF END=0x6FF	PROTECTED
// DATABANK NAME=gpr7 // And remove this bank	START=0x700	END=0×7FF	
DATABANK NAME=gpr67	START=0x600	END=0×7FF	
<pre>// And create this one out DATABANK NAME=gpr8 DATABANK NAME=gpr9</pre>	START=0x800 START=0x900	END=0×8FF END=0×9FF	

Last	update:	2011	/11	/12	12.14
Last	upuate.	2011	/	, 10	12.14

DATABANK	NAME=gpr10	START=0×A00	END=0xAFF	
DATABANK	NAME=gpr11	START=0×B00	END=0xBFF	
DATABANK	NAME=gpr12	START=0×C00	END=0xCFF	
DATABANK	NAME=gpr13	START=0×D00	END=0xDFF	
DATABANK	NAME=gpr14	START=0×E00	END=0xEFF	
DATABANK	NAME=gpr15	START=0×F00	END=0xF7F	
ACCESSBANK	NAME=accesssfr	START=0xF80	END=0xFFF	PROTECTED

Note that the START of the bank is the same as the START of the first bank removed, and the END of the bank, is the same as the END of the last bank removed.

This can be extended into larger ranges, and multiple customised ranges, as below:

DATABANK NAME=miosram_u // DATABANK NAME=gpr6 // Remove this bank,			PROTECTED
<pre>// DATABANK NAME=gpr7 // And remove this bank,</pre>	START=0×700	END=0x7FF	
DATABANK NAME=gpr67	START=0x600	END=0x7FF	
// And create this 512-bit	bank out of the two	256-bit banks.	
DATABANK NAME=gpr8	START=0x800	END=0x8FF	
DATABANK NAME=gpr9	START=0×900	END=0x9FF	
DATABANK NAME=gpr10	START=0xA00	END=0xAFF	
// DATABANK NAME=gpr11	START=0xB00	END=0×BFF	
// Remove this bank,			
// DATABANK NAME=gpr12	START=0xC00	END=0xCFF	
// And remove this bank,			
// DATABANK NAME=gpr13	START=0xD00	END=0xDFF	
<pre>// And remove this bank,</pre>			
// DATABANK NAME=gpr14	START=0×E00	END=0×EFF	
<pre>// And remove this bank!</pre>			
DATABANK NAME=gpr1114	START=0xB00	END=0xEFF	
// And create this 1024-bi	t (1 Kilobit) bank o	ut of the four 3	256-bit banks.
DATABANK NAME=gpr15	START=0xF00	END=0×F7F	
ACCESSBANK NAME=accesssfr	START=0xF80	END=0xFFF	PROTECTED

Or of course you could make the whole lot into one bank if you wanted to. This is NOT recommended though!:

	miosram_u START=0x380 gpr615 START=0x600 2.5kilobits!!	END=0×5FF END=0×F7F	PROTECTED
ACCESSBANK NAME=	accesssfr START=0xF80	END=0×FFF	PROTECTED

#### **Add Sections**

In order to assist in the use of these memory banks, we can give create 'sections' with names, and those names can be referenced in our code later on. I will use the 2nd example above, to demonstrate:

```
DATABANK
                                                                PROTECTED
           NAME=miosram u
                           START=0x380
                                                END=0x5FF
DATABANK
           NAME=qpr67
                           START=0x600
                                                END=0x7FF
// And create this 512-bit bank out of the two 256-bit banks.
DATABANK
          NAME=qpr8
                          START=0x800
                                                END=0x8FF
DATABANK
           NAME=qpr9
                           START=0x900
                                                END=0x9FF
DATABANK
           NAME=gpr10
                           START=0xA00
                                                END=0xAFF
DATABANK
           NAME=gpr1114
                           START=0xB00
                                                END=0xEFF
// And create this 1024-bit (1 Kilobit) bank out of the four 256-bit banks.
DATABANK
           NAME=gpr15
                           START=0xF00
                                                END=0xF7F
ACCESSBANK NAME=accesssfr START=0xF80
                                                END=0xFFF
                                                                PR0TECTED
SECTION
           NAME=CONFIG
                           ROM=config
// This SECTION entry will already exist in the file. Do NOT alter this
line!
SECTION
           NAME=gpr8
                        RAM=gpr8
// This creates a SECTION called 'gpr8' which references the normal 256-bit
bank 'gpr8'
SECTION
           NAME=b512
                        RAM=qpr67
// This creates a SECTION called 'b512' which references our 512-bit bank
SECTION
           NAME=b1024
                        RAM=gpr1114
// This creates a SECTION called 'b1024' which references our 1kb bank
```

You may create as many or as few sections as you require for your application.

## **Application Code**

Once these sections are created, you can use them within your application, by forcing a variable to be stored within that section. This is done using the 'udata' pragma statement with the following syntax:

```
#pragma udata section_name variable_name
```

For example, referencing the above section:

```
#pragma udata b512 MIDI_Table  // This means "store a variable named
'MIDI_Table' in the SECTION named 'b512'
unsigned char MIDI_Table[512];  // Declare the array named
'MIDI_Table', and now it will be stored in 'b512'
```

It is recommended to avoid this wherever possible. Allow the compiler to allocate the memory

#### Last update: 2011/11/18 12:14

wherever possible!

Check An answer to the extended databanks problem for more info!!!!!!!!!

• Compile the project Fix Me! wiki link

• Upload the hex file with MIOS Studio



Still reading? Shouldn't you be writing code right now?;)

From:

http://wiki.midibox.org/ - MIDIbox

Permanent link:

http://wiki.midibox.org/doku.php?id=using\_pic18f4620

Last update: 2011/11/18 12:14

