How to use Xcode2 as IDE on a Mac

While you probably need a PC to burn the MBHP-Loader into the PIC, you can stick to your beloved Macintosh while coding your C-Applications!

And more than that: you are able to use Apple's powerful Xcode IDE and don't leave that except for sending the .syx file! You can even debug your MIOS C-Applications with the MIOS C Simulator - Debugger - although this is in a very early development state, please contribute code! *audiocommander.*

First, go and install GPUUTILS and SDCC on your mac if not already done!

If you have already SDCC running and calling “sdcc -v” in terminal gives you the info you're running 2.5. or higher, you have the most complicated step already completed!

Now, let's go on to real cool stuff and switch to Xcode.

I presume, you have **downloaded the MIOS-C-Skeleton**, unpacked it in an appropriate folder and you're ready to start coding your own nifty MIOS-App.

About Targets and Makefiles

Okay, if you're a pro, you can skip this, else some words for the basic understanding.

To generate the myapplication.syx file that we can upload to the Core, a process of different steps has to be processed. This process is normally handled by a script called “Makefile”:

- (SDCC) Preprocess the sources (⇒ main.c & otherfiles.c)
- (SDCC) Compile the sources (⇒ output/*.asm)
- (SDCC) Assemble to objects (⇒ output/*.o)
- (GAPSM / GPLINK) Link objects to executeable (⇒ *.hex)
- (hex2syx.pl) Convert to SysEx (⇒ *.syx)

If you're using the ACSim, you need a different script; now the GCC compiler is used instead of SDCC because the executeable should run on the Macintosh and not on the box, so the process above would look like this:

- (GCC) Preprocess the sources (⇒ main.c & otherfiles.c)
- (GCC) Compile the sources (⇒ build/myApp.build/*.asm)
- (GCC) Assemble to objects (⇒ build/myApp.build/*.o)
- Link objects to binary executeable (⇒ build/Debug/*.bin)

Okay, now you could either sit down and write your own Makefile (uuhh, that's for the real nerds only), or you do it the easy way by calling some scripts! In Xcode (and any other IDE) you can compile/assemble/link as many executeables out of your sources as you like, by setting up different

targets for each application. So once you've set up a target it can easily be selected by a drop-down menu. Don't forget to choose the right target before compiling your sources ;)

But targets aren't only useful to generate executeables, they can also be used to generate the Makefile!

As you can see, the Makefile invokes several installed applications, which makes the file not cross-platform compliant. So before we can compile the real application, we need to generate our own Makefile, which can be easily done by adapting the included MAKEFILE.SPEC and setting up an extra target.

This is what we will need to do:

- Create a new project
- Set up a new target: Makefile (run once and whenever you add more object sources to the MAKEFILE.SPEC)
- Set up a new target: Application.syx
- Set up a new target: Application_debug (optional if ACSim is used)

Creating a new project

Create a new, empty project from the File-Menu. Select your MIOS-application folder.

Now drag your files into the project. This should be mainly main.h and main.c, but it won't hurt if you add all the files in your directory.

Edit the file “MAKEFILE.SPEC” to your needs. That means, if you have additional classes, you should add them to “MK_ADD_OBJ myfile.c”

Setting up the make target

Because things change a lot, we're making it easy: we set up a “makefile”-target, so with a press of button, your new makefile is generated!

Note, that you can skip this step if your makefile will never change. Neverthess, the procedure is the same as setting up your main target, so you should read it anyway :)

- go to “Project > New Target” and choose “Shell Script Target”. Name the target “Makefile”
- open the Target-Tree, open the target “Makefile” and double-click on “Run Script”
- you'll find now:

```
# shell script goes here
```

```
exit
```

type this:

```
# shell script goes here

PATH=/usr/local/bin:$PATH
cd "$SRCROOT"
./tools/mkmk.pl "$SRCROOT/MAKEFILE.SPEC"
echo Makefile replaced.

exit
```

for Xcode 3 you need to strip the quotes:

```
# shell script goes here

PATH=/usr/local/bin:$PATH
cd $SRCROOT/tools
./mkmk.pl "$SRCROOT/MAKEFILE.SPEC"
echo Makefile replaced.

exit
```

If you get an error like “line x: ./mkmk.pl: Permission denied”, you need to give the perl script execute rights. Easiest way to do this is open your favorite FTP program and set “Executeable” or use the Terminal:

```
chmod 750 <drag mkmk.pl in terminal window, path should appear)
```

Don't forget to close the panel or switch forth and back to the next window item (“Comments”), because sometimes the changes are not recognized.

You're done!

Once you select “makefile” as your primary target and press “build”, the pearl tool ./mkmk.pl is invoked, generates your Makefile, deletes the unnecessary “makefile.bat” and moves the new makefile in your directory root.

If you want to, you can now add “$(SRCROOT)/MAKEFILE.SPEC” to the “Input Files” and “$(SRCROOT)/Makefile” to the “Output Files”. Then Xcode2 checks if any of the input files is newer than the output files and will start the build only if necessary.

Setting up the primary application target

Now this is basically the same as before. Repeat the steps above, but name your target like your

application. The bash script would look like this in the “Run Script”:

```
# shell script goes here

PATH=/usr/local/bin:$PATH

cd "$SRCROOT"
make

exit
```

For XCode 3.0 (under Leopard) do not forget to remove the quotes around the $SRCROOT (the variable points to your project directory).

As you see, the only trick is, to invoke the makefile.

Don't forget to choose your new current target, your application. Otherwise only your makefile is generated again :)

Easy, isn't it?

Debugging with Xcode

go to the ACSim page and grab the debug sources. Note, that these are far from being complete. If you add functions in your own project, please update the files here in the wiki!

Add the debug-files to your Xcode project, and adapt what needs to be adapted (defines, includes etc...).

Now add a new target, choose “Carbon Shell Tool” and name it “MIOS_Debug” or whatever you like.

Now you have to add the Carbon Framework (Right Mouse on the Folder “Libraries”, Add Framework, Search for Carbon).

At this point you just have to make sure, that “ACSim_console.c”, “ACSim_mios.c” and “ACSim_toolbox.c” are the only files that is included for this target. If you open the debug target, you see “Compile Sources (3)”. If there are more files than “debug.c”, delete them from this target.

Important: If you debug your C-Application, you need to make inputs from the command-line, which can't be done from the debugger-console. Launch the “Standard I/O” Window from the “Debug” Menu. This console-window is only available in debug-mode!

If you have questions, feel free to post them here:
http://www.midibox.org/forum/index.php?topic=6527.0