AN-04, Using amrForth/BASIC

April 2003, cws/alm

What you need to do to set up your first project.

To start a project in amrFORTH or amrBASIC, you will need to do several things. First you need a project directory. If you haven't yet installed amrFORTH, then go read the installation help on the website, www.amresearch.com/v6. Once the system is installed, you should be able to start the amrFORTH GUI one of the following ways. From your file manager, which might be 'My Computer' or 'File Explorer', move to the amrFORTH home directory. In Windows this will be \amrforth\v6\. In Linux it will be ~/amrforth/v6/. Click on the amrf.bat for Windows or amrf for Linux. A window should appear with 'amrForth v6.x.0.1' or something similar in its title. If this doesn't occur in a Linux system, be sure that the 'amrf' file is executable, via 'chmod +x amrf'. Alternatively you could start from the command line, cd to ~/amrforth/v6/, and start amrf by typing its name.

Now choose the 'File/New Project' menu. Use the dialog to create a project directory. I currently make a directory call 'Projects' off the root directory, and subdivide that by client, then by name of project. Its up to you. It is best to have a Projects directory tree separate from the amrforth tree though. Installing a new version of amrforth may involve erasing the whole directory tree, and your projects would be erased as well.

If you have chosen a new project directory, then you should be in that directory as far as amrforth is concerned. Depending on your window manager, you may see the current path in the title bar of the program's window. Now you need to configure this project. Choose the Options/Processor menu and find the processor you will be using on this project. The Cygnal processors all start with C8051. Currently amrFORTH supports three Cygnal processors: C8051F30x for the f300 family, C8051F31x for the newer f310 family, and C8051F0xx for the larger family including the f017. More will follow as time and experience permits. Next choose the Options/Comm menu, and choose the com port you will be using. If the size of the font is a problem, choose Option/Font and pick a better one. Finally choose Options/Save. Two configuration files will be saved in the current directory, amrfconf.tcl and amrfconf.fs. One is a Tcl source file and the other is a forth source file.

Be aware that there is not really a central amrforth program to be run from an single icon. In order to run amrFORTH/BASIC for a particular project, you should start from that project, so that the configuration files will be found at startup. One way to do this is to always start by clicking the icon in the file manager in that project directory. Another way is to cd to that directory from the command line and run amrf from there manually. Finally you can create a desktop icon that runs amrf in that particular directory. In Linux it probably makes more sense to use the command line. In Windows it might make more sense to use one of the other two methods.

Now you have a place to work. The first thing you need to do in order to actually start working is to create a load file. The amrFORTH and amrBASIC compilers load the file named 'job.fs' in the current directory. If no such file exists, then the compilers won't do anything. So to get started writing your program, you need to create at least the file job.fs. By the way, .fs stands for Forth

Source. Even if you are programming in BASIC, you should think of Forth as an operating system that BASIC runs on top of. 'job.fs' can contain forth style comments. The backslash, \, starts a comment that runs to the end of the line. You might want to put such a comment in the first line, or lines, of job.fs to explain what the project is about. Then each following line should probably start with 'include'. For example if your project uses an LCD your job file might look like this:

```
\ LCD project
include lcddriver.fs
include main.fs

or if you are using BASIC:

\ LCD project
include basic.fs
include main.bas

or even:

\ LCD project
include lcddriver.fs
include basic.fs
include main.fs
```

Then of course you will need to actually create each of the files you want to include (except basic.fs, that's already done for you), and put some actual source code into each one. How you come up with the actual source code is too big a subject for this little paper. Look at the examples in the \amrforth\v6\ subdirectories. You can use your editor of choice. Just be sure the files are saved in your Project directory. You may also use the integrated amrFORTH editor. It is simple, similar to the Windows Notepad editor.

Actually though, you can start working with amrFORTH without writing any source code. You need to create the job.fs file, but you don't have to put any code in it. The compiler will still include the amrFORTH kernel and you can download and test that kernel. Try this. Start up amrf in your new project directory. From the amrFORTH command line type c and press enter, or pull down the Compiler menu and choose Compile. You will see the message 'Compiling Interactive System' on the screen, followed by 'Host Stack = <0> ROM used = nnnn' where nnnn is the number of bytes used by your program, in this case the amrFORTH kernel. Next press d on the command line or pull down the Compiler menu and choose 'Download RS232'. If your amrSTAMP mother board is powered up and the serial line is correctly attached, press SW1, the reset switch, and you should see the following on the GUI screen:

```
Cygnal Downloader.
Downloading rom.bin via RS232
Press RESET on the target board
512 byte pages: 04 03 02 01
Host>
```

When you see `Host>` you can type forth and press return. Hopefully you see 'Target

responds' on the same line, and the next line contains the prompt 'Forth> '. You can type words here and see what words are available. Most of what you see can be directly executed from the command line. Generally what you will do is put some parameters on the stack and run your own test words, then check the stack to see what they did. Be aware that there are a few words in that list that should not be executed from the command line. For example (next) is meant to be compiled by the word next, to make a for next loop. (next) won't do anything sensible from the command line. I just now tried it though and was surprised to find that it didn't cause the target to lock up. One other thing, you might see the word {(s\")} in the words list. This is another word you should not execute from the command line. The actual name of the word is (s") and it is compiled by the forth word s" for an inline string. The braces and the \ are artifacts of the way Tcl is being used to create the words list. There may be a way around this but we haven't found it yet. Tcl is not as clear and simple as forth is, but it has access to the Windows serial ports and a GUI, so we use it.

Many of the words in the words list should be familiar to you if you have used forth. You type number to put them on the data stack. Use .s to see what is currently on the stack. Use + or - to add or subtract numbers on the stack. Experiment with the kernel words just as you would with the words in your application, to see how they affect the stack. One thing you can't do interactively is compile new definitions.

This is a target compiled system. Compiling takes place on the host and the result is downloaded to the target. If you need to make a short definition for debugging, you will need to edit it into a file, compile it and download it. We suggest you put such test definitions at the end of your main program file, or at the end of the job.fs file. Do use them though. You have a live interactive system here. Take advantage of it.