CSTARTUP

The assembly file `CSTARTUP.S39` is a highly target-specific, vital part of the run-time model. On processor reset, execution passes to the run-time system routine `CSTARTUP`, which normally performs the following:

Initializes C file-level and static variables except `__no_init` variables.

Calls the start of the user program in `main()`.

`CSTARTUP` is also responsible for receiving and retaining control if the user program exits, whether through `exit` or `abort`.

To initialize hardware before the initializing of variables, define `__LOW_INIT` before assembling `CSTARTUP`. Place all init code in a C function named:

```
void __low_level_init()
```

Since `__low_level_init` is called before any initializing code, your code should not depend on variables having been initialized.

See below how to include your `CSTARTUP` in your project.

Customizing CSTARTUP using the command line

Do not modify `CSTARTUP` unless required by your application. If you need to modify it, the overall procedure for creating a modified copy of `CSTARTUP` and adding it to your project is as follows:

Copy the assembly source file `cstartup.s39`, which is supplied in the `iccpic\` directory, to your project directory.

Make any required modifications and save the file under the same name. To call `__low_level_init`, add `-D__LOW_INIT` to the command line.

Copy the linker file you intend to use from the `iccpic\` directory. At the end of the file, locate the section 'Select the 'C' library…'. On the line stating which library to use, `clXXX.r39`, put `-C` before the `clXXX.r39`. This causes the whole library to be loaded as library, so we can override the `CSTARTUP` contained in the library.

`CSTARTUP` includes two files, `IO.INC` and `MACROS.INC`. Replace `IO.INC` with the full name of the file you intend to use, e.g. `IO17C43.INC`.

Make sure the include path points to the `apic\` directory and the `clib\` directory using the –I option.

Assemble your copy of `CSTARTUP` using target options that match your selected compiler options. Also define a symbol for the appropriate target family model.

The assembler option `-v[m|h]` is used for choosing between the mid-range or high-end instruction set.

The symbol `RTVER` has to be set on the commandline to match the current version of the Runtime Model. The current Runtime Model Version can be found at the beginning of the iccpic.txt file.

For example, if you have compiled for the PIC17C43 processor (`--cpu=17C43` or `-v17C43`), assemble `cstartup.s39` with the command:

```
apic cstartup -v1 -D__LOW_INIT -DRTVER="\"8\""
```

This will create an object module file named `cstartup.r39`.

Link your code using the modified linker command file.

Customizing CSTARTUP using the IAR Embedded Workbench

Copy the assembly source file `cstartup.s39`, which is supplied in the `iccpic\` directory, to your project directory.

Make any required modifications and save the file under the same name.

Copy the linker file you intend to use from the `iccpic\` directory. At the end of the file, locate the section 'Select the 'C' library…'. On the line stating which library to use, `clXXX.r39`, put `-C` before the `clXXX.r39`. This causes the whole library to be loaded as library, so we can override the `CSTARTUP` contained in the library.

Define the assembler symbol `RTVER="8"` using the #define tab in the APIC category in the IAR Embedded Workbench Project Options Dialog. The actual number can be found at the beginning of the iccpic.txt file under Runtime Model Version.

If you want to call the function `__low_level_init()`, you need to define `__LOW_INIT` as well.

`CSTARTUP` includes two files, `IO.INC` and `MACROS.INC`. Replace `IO.INC` with the full name of the file you intend to use, e.g. `IO17C43.INC`.

Make sure the include path points to the `apic\` directory and the `clib\` directory using the Include tab under the APIC options.

Assemble your copy of `cstartup.s39` using the same processor configuration as you have specified for your project.

This will create an object module file named `cstartup.r39`.

Add the customized `CSTARTUP` module to your project.

Rebuild your project.

Maintaining library files

The IAR XLIB Librarian command `REPLACE-MODULES` allows you to permanently replace the original `CSTARTUP` with your customized version. See *The IAR XLIB Librarian* in *PICMicro Assembler, Linker , and Librarian Programming Guide* for detailed information.

Variable and I/O initialization

In some applications you may want to initialize I/O registers, or omit the default initialization of data segments performed by `CSTARTUP`.

You can do the latter by defining the symbol `IGNORE_SEG_INIT` before assembling `CSTARTUP`.

Customizing __low_level_init from the command line

In most cases you can use the `__low_level_init` module provided with the product. If your application requires that you modify it, the overall procedure for creating a modified copy of `__low_level_init` is as follows:

Copy `lowinit.c`, by default located in the `iccpic` directory, to your project directory.

Make any required modifications, including the code necessary for the initializations. Save the file using the same filename.

Compile the customized version of `lowinit.c` using the same processor option and memory model as for your project.

Link the file to the rest of your code.

Customizing __low_level_init in the IAR Embedded Workbench

In most cases you can use the `__low_level_init` module provided with the product. If your application requires that you modify it, the overall procedure for creating a modified copy of `__low_level_init` is as follows:

Copy `lowinit.c`, by default located in the `iccpic` directory, to your project directory.

Make any required modifications, including the code necessary for the initializations. Add the customized version of `lowinit.c` to your project.

Compile the customized routine using the same processor configuration and memory model as for the project.

Rebuild the project.