AN989
APPLICATION NOTE

Getting Started with the ST7 Hiware C Toolchain

by Microcontroller Division Application Team

INTRODUCTION

The purpose of this application note is to explain how to get started with the Hiware C Toolchain and the ST Visual Debugger developed by ST (STVD7) or the Integrated Development Environment (IDE) developed by HIWARE (PANTA) using ST7 microcontrollers.

Metrowerks Europe was founded as HIWARE AG in 1987. Metrowerks Europe has its headquarters in Basel, Switzerland near the French and German border. In addition to their headquarters in Basel, they have a development center in Zurich.

In July 2000, Motorola acquired HIWARE AG which became the European headquarters for Metrowerks, a Motorola company based in Austin, Texas.

Among other things, Hiware develops C and C++ compilers for ST7 microcontrollers.

The following operating systems are supported: Windows 95, 98, NT.

For more information about these debuggers, please refer to the ST Visual Debug User Manual included with the software in a .pdf version or to the PANTA User Manual also installed with the software. A complete on-line help service is also available through the Help menu in the STVD7 program.

Demo versions and documentation for all the tools described in this document can be downloaded from the HIWARE Internet site (http://www.hiware.com).

The files described in this application note, the corresponding source files as well as this application note are provided in a .zip file that may be downloaded from the ST MCU internet site (http://mcu.st.com) or from the “MCU ON CD” CD-ROM (exercise.exe).

Table of Contents

1 C HIWARE TOOLCHAIN

1.1 C COMPILER FEATURES

The ST7 C Compiler fully conforms to the following standards:

– ANSI X3.159-1989 Standard C (ANSI/ISO 9899-1990)

– ANSI C++ Draft (X3J16/95-0087 WG21/N0687).

These standards are verified using independent verification suites.

The main features of the C Compiler for ST7 microcontrollers are listed below:

- CPU-specific and application-specific optimizations for fast, compact and high-quality code
- Capable of producing assembler header files from C/C++ files usable in the assembler
- Full ANSI C support including asm keywords
- __asm and __interrupt functions can be used in strict ANSI mode
- HLI (High Level Inline) assembly can be mixed with C/C++/EC++/cC++ code
- Extended type checking during compilation
- Extended syntax checker
- All compiler predefined macros are logged to a file
- Over 50 different options to adapt the compiler behavior to specific customer requirements
- Unlimited number of command line define directives
- Unlimited number of include directories on the command line
- Possibility of stopping the compiler while compiling
- The compiler directly produces the object file
- Optional capability of semi-automatically producing a make file
- Full access to option settings in the source code using #pragma OPTION directives
- Messages in the source code can be modified by using #pragma MESSAGE directives
- Pragma directives to control the entry/exit code of each single function
- Options and pragma directives to allocate constant objects into ROM
- Compiler can encrypt source files and is able to process encrypted files

1.2 HIWARE DEBUGGER: HI-WAVE

HI-WAVE is a multi-purpose development framework that can be used for various tasks in the embedded system and industrial control world. (See Figure 1.)

Some typical tasks are:

- Simulation and debugging
- Hardware simulation (I/O, board)
- Fast prototyping
- Automated testing
- Building of a target application
- Building of a control application

These modular Debugger and Simulator systems can be customized to best meet the user's needs. The Hi-Wave framework contains tools for debugging, visualization, CPU simulation and I/O device simulation. It is smoothly integrated in the PANTA IDF (Integrated Development Framework).

Hi-Wave was designed to provide a maximum of efficiency, therefore drastically reducing development time and costs:

- Versatile and intuitive Drag & Drop possibilities between almost all components are available
- Seamless switching between Debugger and Editor windows
- Unlimited number of components
- Easy development of user-defined components
- Command line interface
- Special features for embedded debugging
- Fast download performance
- Extendible concept
- Source folding, language-sensitive coloring
- All language types are supported (composed)
- Symbolic enumeration
- Seamless switching between different targets
- Same look and feel for all targets
- Writing and reading to/from files
- Dialog Boxes to manage more complex information
- Shortcut keys for common tasks

- Consistency checks: warns about mismatch between object files and sources
- Supports Metrowerks Europe and ELF/Dwarf Object
- File Format and Motorola S-Records as well
- Online help and context-sensitive help
- In-Place editing and In-Place setting allow changing any value displayed on the screen interactively

1.2.1 HI-WAVE Simulation and Debugging

Several components (i.e. Source Level Debugging, Register, Data) provide powerful simulation and debugging services via simple mouse interactions, pop up menus or dialog boxes. An extract of the summary of these services is listed below:

- Setting various kinds of breakpoints
- Setting various kinds of watchpoints
- Stepping functions
- Edit memory, variables, registers
- Folding, unfolding code and data structures
- Zooming
- User-configurable layout
- Unlimited number of components
- Drag & Drop facilities
- Graphical representation of software and hardware traces
- Real-time kernel awareness
- True-Time simulation
- CPU awareness

HI-WAVE is able to simulate the real-time behavior of your target application in a True-Time fashion. It may be slower than true real-time but it maintains the correct time relations between the events. This simulation is based on the MCU clock cycle and respects all timing aspects, including:

- Instruction execution timing
- Wait states (memory and I/O)
- Timing of instruction prefetch queue
- Interrupt latency

1.2.2 Peripheral Builder

The Peripheral Builder provides a powerful way to generate and test peripheral and I/O device components of any complexity in an efficient and accurate way:

- Generates simulation components of on-chip and on-board peripherals
- Generates simulation components of customer hardware
- Memory-mapped peripheral register
- Independant peripheral registers (latches, external peripherals...)
- Substitution of memory/registers by a peripheral device
- Register block relocation support (if applicable)
- Target CPU independant
- Any number of registers
- Automatic splitting of accesses to register size
- Control of register access time
- Efficient support for mass storage modules (FLASH, EEPROM...)
- Supports visualization
- Derived from powerful base classes
- Well integrated in the common host development tool (Microsoft Visual C++)
- Full debugging with visual tools
- Allows access to all ressources of the Host (file system, network, I/O ports...)
- Fast, pure C/C++ implementation of Peripheral Simulation (no interpreted I/O description language)
- Object-oriented approach
- Dynamic loading and extension
- Subscription/notification mechanism to avoid polling:
  - Memory/register changes
  - Internal states of other peripheral change
  - After fixed time
- Integration of real hardware in simulation using Hardware in the Loop (HIL):
  - External hardware ports/registers (A/D converter, communication...)
  - Sensitive algorithm can be executed in hardware (encryption)
- Access to HI-WAVE engine services and commands

1.2.3 Main Development Tools

- CST7: ANSI C Compiler dedicated and optimized for the ST7 instruction set.
- AST7: ST7 Macro Assembler compatible with the C code at object file level.
- LINKER: Utility able to allocate memory for the merged object and library files.

1.2.3.1 Debugging and Programming Tools

- SIMUST7: ST7 Core/Peripheral simulator for debugging an application.
- BURNER: ABS to S19 File Format Converter for EPROM burning.
- ST7BUG: Hi-Wave Debugger for debugging applications with an ST7 emulator.

1.2.3.2 Enhanced Tools

- LIBMAKER: Librarian utility for creating and maintaining object file libraries.
- DECODER: Disassembler utility with in-line assembler code generation capability.

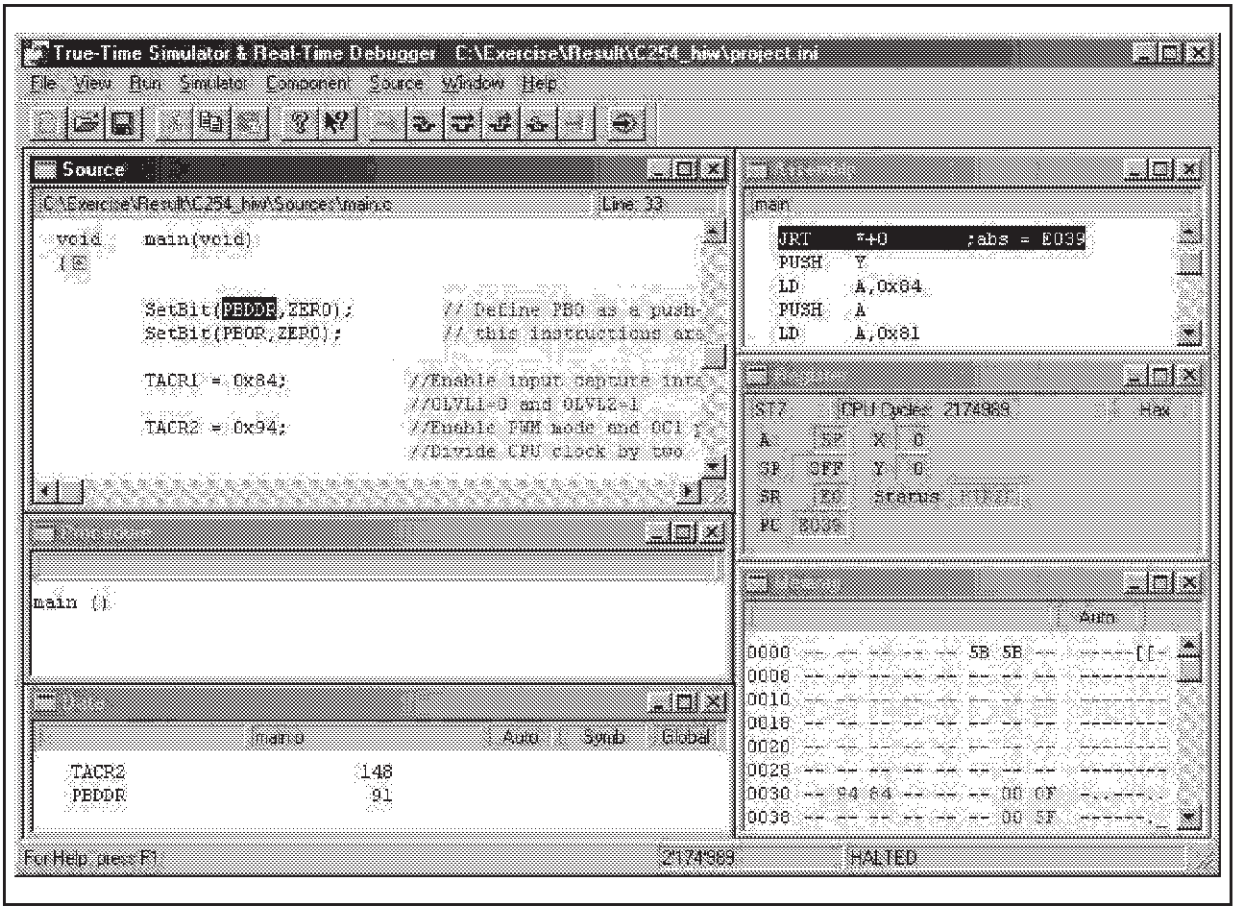
1.2.3.3 Application Management Tool

- MAKER: Makefile processor for C application maintenance.

1.2.3.4 IDE

- IDEA for ST7

Figure 1. HIWAVE Simulator



2 STVD7 AND HIWARE C TOOLCHAIN

2.1 STVD7 DESCRIPTION

The STVD7 is the brand new debugger developed by STMicroelectronics, which replaces the WGDB7 and which is still available at no cost.

This is an IDE (Integrated Development Environment) which means that the same graphical Windows interface can be used for both editing and debugging.

The most recent version of the STVD7 Debugger can be downloaded from the 8-bit MCU Internet site (http://mcu.st.com on the Free Software page) or from the “MCU ON CD” CD-ROM. The WGDB7 is still available on the same Free Software page.

The STVD7 Debugger can be used with 3 different toolchains:

- ST Assembly Toolchain
- COSMIC C Toolchain
- HIWARE C Toolchain

Select the desired toolchain in the “Project Settings” window, as described below.

Please refer to Application Note AN978 (“STVD7 Key Features”) or to the STVD7 User Manual for more information about STVD7 features.

2.2 GETTING STARTED WITH HIWARE AND STVD7

2.2.1 Hiware Files

Default file type definition:

default.env	Hi-Cross/Hi-Wave main environment file
enviro.mak	Application makefile
enviro.prm	Hi-Cross linker parameter file
.c/.h	C source or header file
*.asm	Macro assembler source file
enviro.abs	Absolute executable file
enviro.map	Mapping and statistic information file
.o/.lib	Object or library file
*.lst	Object disassembled file

2.2.2 STVD7 Configurations

Once the STVD7 Debugger has been installed, 3 different icons will be displayed in the Windows Start menu: one for the simulator, one for the development kit and one for the emulator.

Click on the corresponding icon to select the required tool.

The first time the STVD7 program is launched (i.e. any of the 3 icons is selected), the toolchains must be configured (configure only the ones that will be used):

- The ST Assembly Toolchain path is already specified,
- For Hiware, the correct path is C:\Hiware\prog (demo or licensed version),
- For Cosmic, the correct path is C:\Program Files\cosmic software\st7eval\cxst7 (demo version) or C:\COSMIC\ST7 (licenced version), if the default paths were selected during the install procedure.

Click on the Browse button in this window to select the correct path.

Once STVD7 Debugger has been launched, a new workspace (*.wsp) must be created:

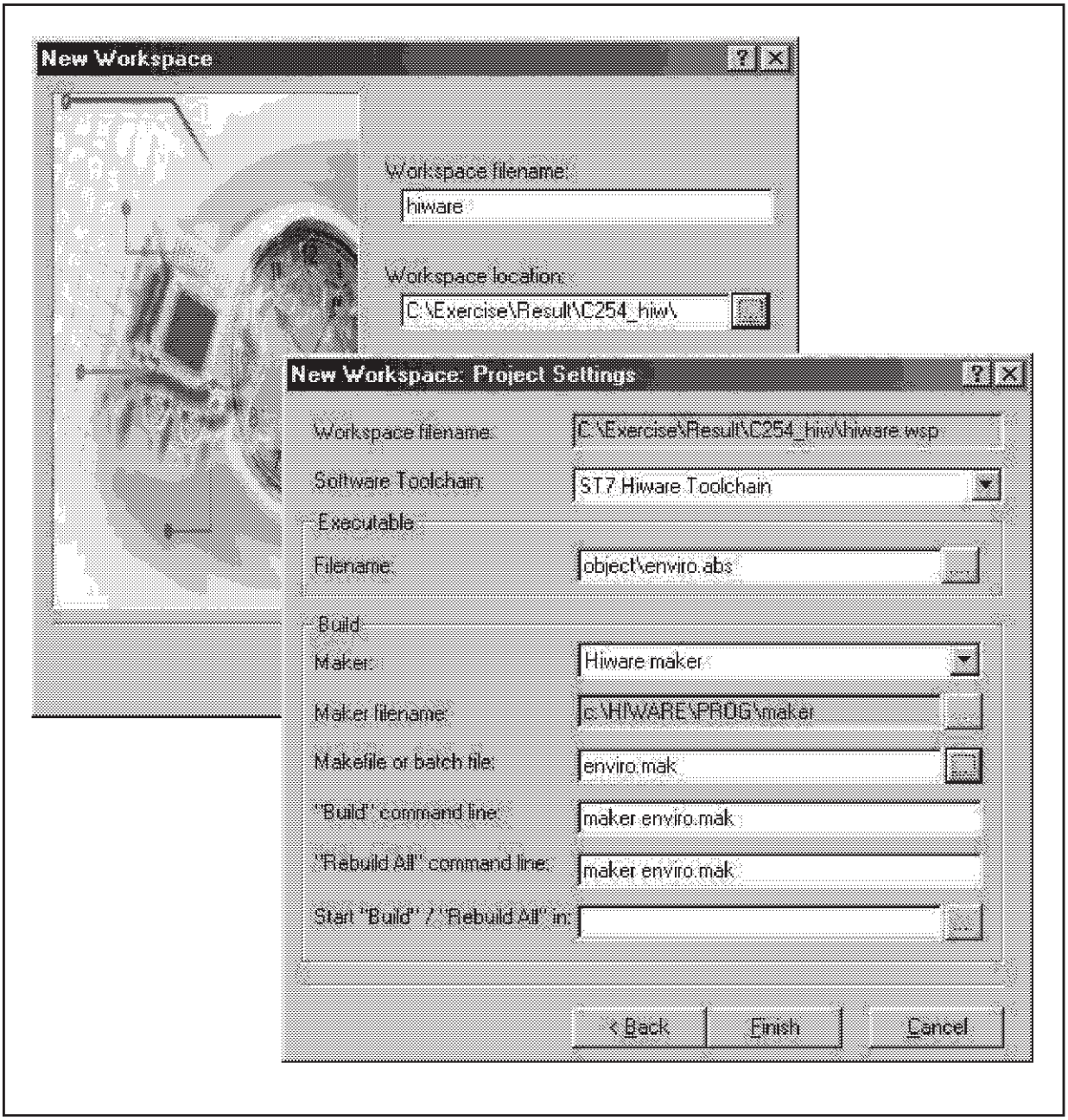
- Click on "***File, New Workspace***" and enter the requested information (see Figure 2.):
  - Workspace Filename: Assign a name to the workspace,
  - Workspace location: Use the Browse function to identify the working directory.

Then, click on ***Next*** and enter the **Project Settings** (see Figure 2.):

- Software Toolchain: Select the Hiware C Toolchain,
- Executable Filename: Name the emulator downloadable file (.abs created by the Linker, enviro.abs here) used to launch the debug session. (If this file already exists, use the browse function to identify its location.) It is also possible to return to this window ("***Project, Project Settings***") and enter this field at a later time.
- Maker: Select the type of maker. A default maker is proposed depending on the toolchain selected .
- Make File or Batch File: Assign a name to the created Maker file (.mak, enviro.mak here), or use the browse function to identify its location. This file will be linked to the Build and Rebuild buttons by default. If other files will be used for the Build and Rebuild commands, enter the correct name instead of the default one listed in the corresponding line.

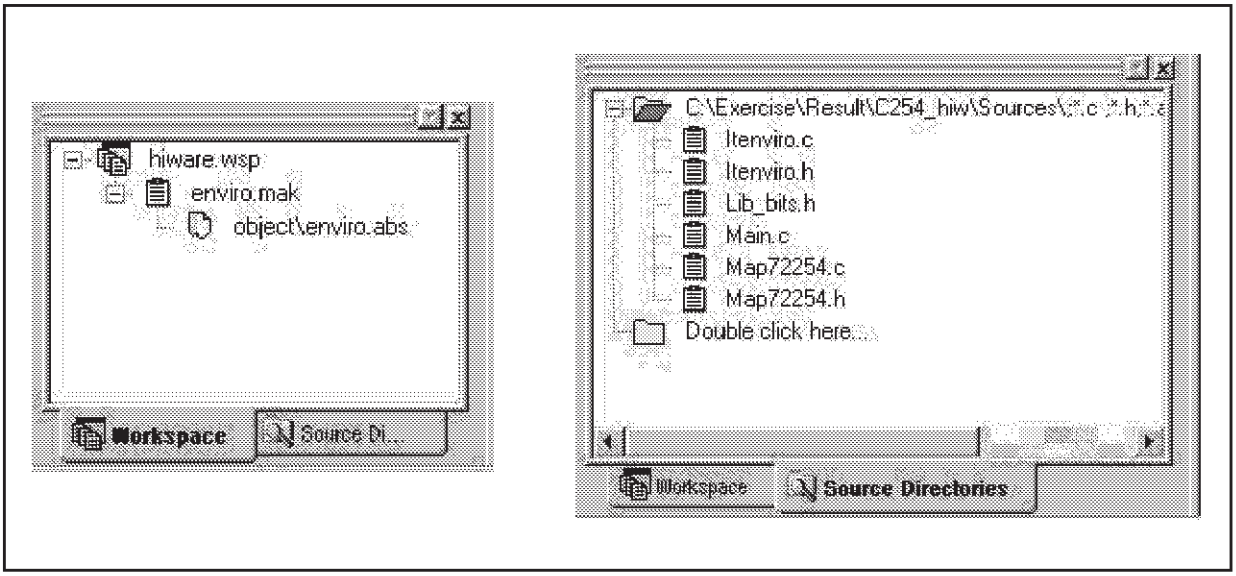
Click on "***Finish***".

Figure 2. STVD7 New Project



- Click on the "*Source Directory*" tab in the Workspace window and double-click on the directory. (See Figure 3.) The working directory is selected by default (in our example, go to the object directory) and the list of source files is displayed in the workspace window. Double-click on the file to be edited. New files can also be created in this window. This workspace window is displayed by default on the left-hand side of the STVD7 window (Edit or Debug modes).

Figure 3. Workspace Window



A new project has now been created.

2.2.3 Default.env File

Since the most recent version of the Hiware C Compiler is delivered with the PANTA IDE, the Hiware toolbox, which was used to create the default.env file, is no longer available. It's now up to the user to write this file as shown in the example attached to this application note:

```
LIBPATH=C:\HIWARE\LIB\ST7C\include
GENPATH=C:\HIWARE\LIB\ST7C\src;C:\ HIWARE\LIB\ST7C\lib;
*C:\Exercise\result\c254_hiw
OBJPATH=C:\Exercise\result\c254_hi w\OBJECT
ABSPATH=C:\Exercise\result\c254_hi w\OBJECT
TEXTPATH=C:\Exercise\result\c254_h iw\OBJECT
COMP=C:\HIWARE\PROG\CST7.EXE
LINK=C:\HIWARE\PROG\linker.EXE
FLAGS=-ml
ERRORFILE=
RESETSTACK=0x17F
```

This file contains all the main paths of the application (tools, source files, library files...) described below.

2.2.4 Sources and Library File Paths (“Paths” tab)

LIBPATH is the file path to the libraries required by the application. Most of them are in a Hiware subdirectory and the corresponding path is then:

```
LIBPATH=C:\HIWARE\LIB\ST7C\include
```

If a file is included in a source file with greater/lower signs (`#include <file.h>`), the compiler searches first in the current directory and then in the LIBPATH directories (not in the GENPATH one).

In this case, only the Hiware libraries are included, but other paths, separated from the first one by a semicolon, can be added.

If a file is included in a source file with double quotes (`#include “file.h”`), the compiler searches first in the current directory, then in the GENPATH (general path) directories and finally in the LIBPATH directories.

```
GENPATH=C:\HIWARE\LIB\ST7C\src; C:\HIWARE\LIB\ST7C\lib;
*C:\Exercise\result\c254_hiw
```

Note: If a directory name starts with a star sign “*”, it means that the whole directory tree is accessed.

2.2.5 Generated File Paths (“Paths” tab)

All object files generated by the C Compiler or the Macro Assembler and all library files built by the Librarian utility are stored in the OBJPATH directory. Unless otherwise specified, all these files will be placed in the current directory. We advise that all the created files be placed in a Object subdirectory (from the object files up to the executable format file)

```
OBJPATH=C:\Exercise\result\c254_hi w\OBJECT
```

The .ABS absolute file generated by the Linker is stored in the ABSPATH directory. Unless otherwise specified, this file will be placed in the current directory.

```
ABSPATH=C:\Exercise\result\c254_hi w\OBJECT
MAPPATH=C:\Exercise\result\c254_hi w\OBJECT
TEXTPATH=C:\Exercise\result\c254_h iw\OBJECT
```

All text files generated by the HIWARE utility (*.lst, *.map...*) are stored in the TEXTPATH directory. Unless otherwise specified, all these files will be placed in the current directory.

2.2.6 System Environment Variables (“Additional” tab)

Environment variables can be defined in the “default.env” main environment file and used in the source and make files of the application.

```
COMP=C:\HIWARE\PROG\CST7.EXE
LINK=C:\HIWARE\PROG\LINKER.EXE
FLAGS = ...
```

If FLAGS and ASMOPTIONS environment variables are set, the Compiler and the Macro Assembler utilities append their contents by default to their command line if the “*default.env*” file is placed in the working directory. These two variables are keywords.

```
FLAGS= -Cc -Cni -N -Ml -Wpd -Ll
ASMOPTIONS=
```

2.2.7 Error File Name Specification

By default, the error file generated by the C Compiler or the Macro Assembler is called “*edout*” and placed in the current directory. This can be modified using the ERROR line in the *“default.env”* file. The desired path and name of the error file must then be specified:

```
ERROR: C:\Exercise\Result\C254_hi w\Object\cbe.err
```

2.2.8 Hiware Makefile (.mak)

The makefile is the input of the HIWARE application maintenance utility. Below is an example of a basic application makefile. This code joined in the example attached to this note:

```
ENV = default.env
CC  = $(COMP) $(FLAGS)
#****************** OBJECT FILES DEFINES ****************************
OBJ_LIST  = map72254.o\
      main.o\
      itenviro.o
#****************** MAIN FILES COMPILE & LINK ************************
ENVIRO.abs : $(ENV) $(OBJ_LIST) ENVIRO.prm
       $(LINK) ENVIRO.prm

main.o : $(ENV) main.c lib_bits.h map72254.h hidef.h
     $(CC) main.c
map72254.o : $(ENV) map72254.c
       $(CC) map72254.c
itenviro.o : $(ENV) itenviro.c map72254.h lib_bits.h
       $(CC) itenviro.c
```

The makefile consists of rules like:

```
target file : {dependency file list}
   {command lines}
```

Tips:

– To rebuild an application without deleting all the object files, insert the "*default.env*" file in all dependency rules. This way, executing the makefile, after saving this file, rebuilds the entire application.

– Use environment variables defined in the "*default.env*" file such as LINK, COMP...

– Macro or dependency lists are line-oriented. To split one line into several lines, the "\" is used as a line break.

– Use the -Lm (file dependencies) and -Lo (list of object files) options to semi-automatically generate the makefile.

2.2.9 Linker Parameter File (prm)

The PRM parameter file is the input of the HIWARE Linker utility. It must contain at least the following commands at all times: LINK, NAMES, STACKSIZE, PLACEMENT and VECTOR ADDRESS (for reset vector):

```
LINK  enviro.abs

NAMES main.o
    map72254.o+
    itenviro.o
    start07.o
    ansi.lib
END

/* STACK INITIALIZATION
********************************* ******************/
STACKTOP 0x017F

/* MEMORY LOCATION SETTING
********************************* ****************/
SECTIONS
      ZRAM     = READ_WRITE 0x0080 TO 0x00FF;
      ROM      = READ_ONLY  0xE000 TO 0xFFDF; /* for 8Ko ROM */
PLACEMENT
      PORT_C   INTO NO_INIT 0x0000 TO 0x0002;
      PORT_B   INTO NO_INIT 0x0004 TO 0x0006;
      PORT_A   INTO NO_INIT 0x0008 TO 0x000A;
      MISC1    INTO NO_INIT 0x0020 TO 0x0020;
      SPI      INTO NO_INIT 0x0021 TO 0x0023;
      WDG      INTO NO_INIT 0x0024 TO 0x0024;
      CRSR     INTO NO_INIT 0x0025 TO 0x0025;
      I2C      INTO NO_INIT 0x0028 TO 0x002E;
      TIMERA   INTO NO_INIT 0x0031 TO 0x003F;
```

```
        MISC2    INTO NO_INIT 0x0040 TO 0x0040;
        TIMERB   INTO NO_INIT 0x0041 TO 0x004F;
        ADC      INTO NO_INIT 0x0070 TO 0x0071;

    DEFAULT_ROM, ROM_VAR, STRINGS INTO ROM;
    DEFAULT_RAM , _ZEROPAGE, _OVERLAP   INTO ZRAM;
END
PRESTART OFF
/* INTERRUPT VECTOR SETTING : ADDRESS - ROUTINE
*******************************
/*INIT main */
VECTOR ADDRESS 0xFFE0 dummy_rt
VECTOR ADDRESS 0xFFE2 dummy_rt
VECTOR ADDRESS 0xFFE4 i2c_rt
VECTOR ADDRESS 0xFFE6 dummy_rt
VECTOR ADDRESS 0xFFE8 dummy_rt
VECTOR ADDRESS 0xFFEA dummy_rt
VECTOR ADDRESS 0xFFEC dummy_rt
VECTOR ADDRESS 0xFFEE timb_rt
VECTOR ADDRESS 0xFFF0 dummy_rt
VECTOR ADDRESS 0xFFF2 tima_rt
VECTOR ADDRESS 0xFFF4 spi_rt
VECTOR ADDRESS 0xFFF6 css_rt
VECTOR ADDRESS 0xFFF8 eit1_rt
VECTOR ADDRESS 0xFFFA eit0_rt
VECTOR ADDRESS 0xFFFC sw_rt
VECTOR ADDRESS 0xFFFE _Startup  /*main*/
```

- **LINK Command**

The LINK command defines the absolute executable output file (ABS) of the linker:

```
LINK  enviro.abs
```

- **NAMES/END Command**

The object and library file lists mentioned between the NAMES and END commands contain the input files to be merged into the memory by the Hiware Linker utility.

```
NAMES main.o
   map72254.o+
   itenviro.o
   start07.o
   ansi.lib
END
```

The plus symbol (“+”) after an object file means that the Linker has to force the memory allocation of all the objects defined in this file. In order to optimize memory space, this option is not defined as the default.

Note: The ST7 hardware register definition object file has to be followed by a “+” sign.

■ STACKSIZE Command

The ST7 stack is defined using the STACKSIZE command.

```
STACKSIZE  0x17F
```

■ SECTIONS Command

The SECTIONS command is optional and allows significant names to be assigned to certain address ranges.

```
SECTIONS
      ROM  = READ_ONLY 0xE000 TO 0xFFDF;
```

■ PLACEMENT/END Commands

The list of correspondence rules put between the PLACEMENT and END commands defines the relationship between the naming of segments, their type and their allocated memory addresses. Some default segments are considered as key words. Some examples are given below:

DEFAULT_RAM, DEFAULT_ROM, _ZEROPAGE, STRING, ROM_VAR and _OVERLAP.

```
PLACEMENT
      HW_REGISTERSINTONO_INIT  0x0000 TO 0x007F;
      DEFAULT_ROM, ROM_VAR, STRINGS INTO ROM;
      DEFAULT_RAM , _ZEROPAGE, _OVERLAP   INTO ZRAM;
```

■ VECTOR ADDRESS Command

The VECTOR ADDRESS command is used to initialize the ST7 reset and interrupt vectors. Each vector contains the address of the function to be executed.

```
VECTOR ADDRESS 0xFFFC sw_rt
VECTOR ADDRESS 0xFFFE _Startup
```

By default, the HIWARE start-up vector is used (the start07.o file is added to object files list in the Linker parameter file). This means that the ST7 reset vector (0xFFFE) has to be linked with the “*_Startup*” function. To customize your own start-up routine, the following two commands have to be set. The following example provides a “*main*” routine as start-up:

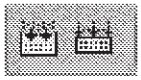
```
INIT main
VECTOR ADDRESS 0xFFFE main
```

CAUTION: When the HIWARE start-up is not used, the RAM is not initialized. This means that all global variables initialized at their definition (`char my_var = 10;`) are not taken into account.

2.2.10 Debug Mode

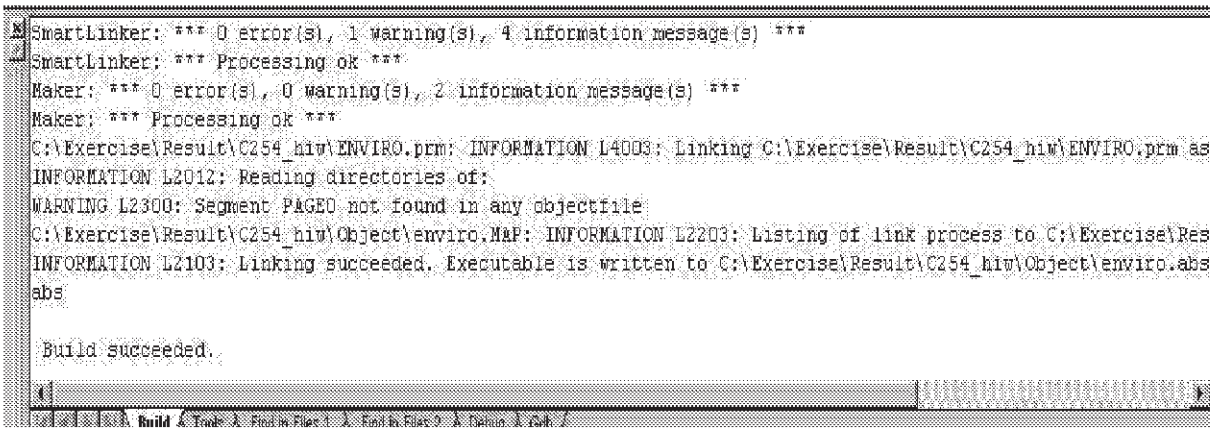
- New source files can of course be created through the "**File, New text file**" menu.
- In order to build when all the application files have been created, click on the Build or Rebuild button on the "**Project, Build or Rebuild**" menu. These 2 icons may be added to the toolbar using the "**Tools, Option**" menu and clicking on "**Project**" in the **Toolbar** tab. This will launch the batch file chosen in the Project Settings window.

Figure 4. Build and Rebuild Buttons



The results of all operations, debug messages and other information (emulator/connection information, run/stop information, warning messages) are displayed in the output window, which is located below the source window. See Figure 5.

Figure 5. Output Window

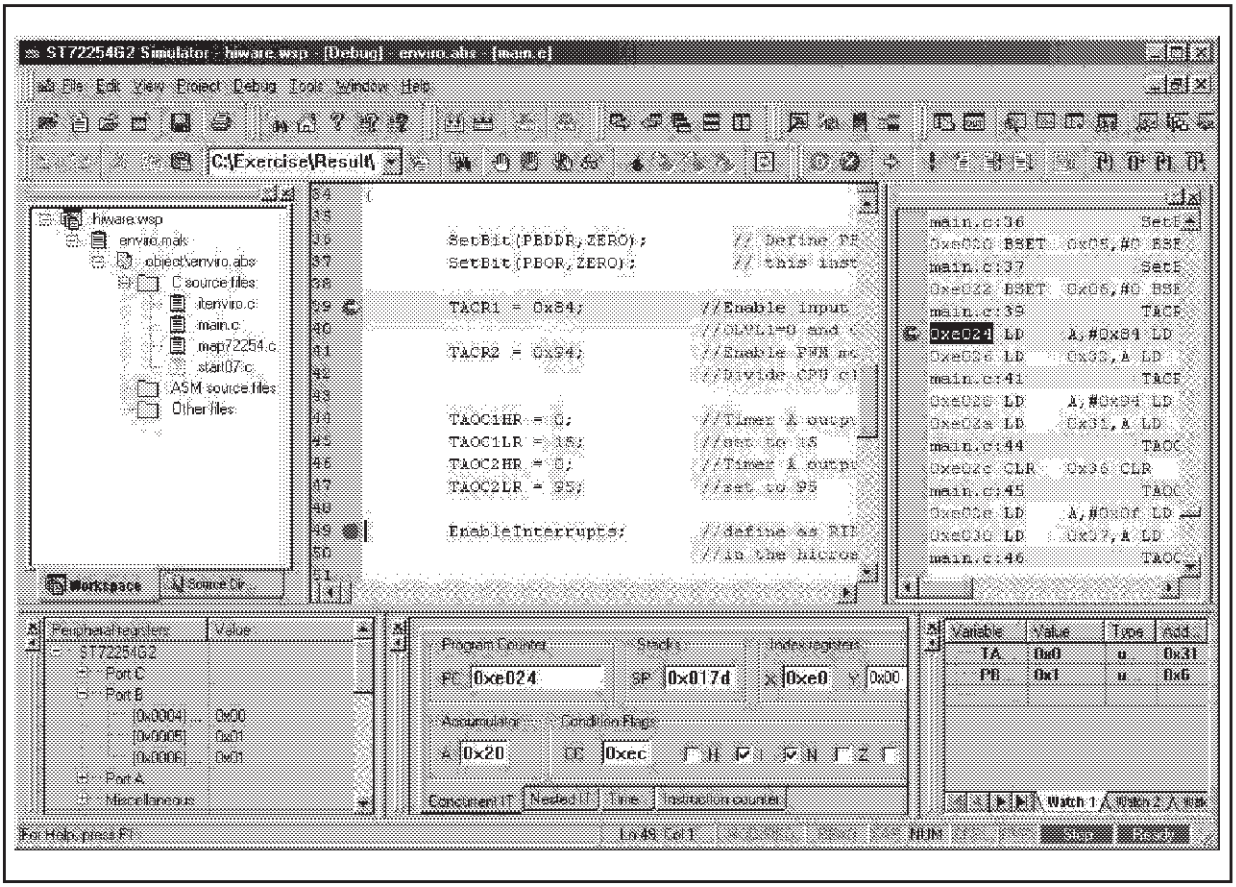


- Detected errors are displayed in the output window. Double-click on the error to go directly to its location in the program. Correct the error, then rebuild.
- When the build is successful, Debug mode can be entered by clicking on the blue icon shown in Figure 6. If the Hiware Startup routine is used, you may be asked for the start07.c file (the startup source file). In this case, there are 2 possibilities:
  - The message can be ignored and the startup routine won't be displayed in the source window,
  - The start07.c file can be copied from C:\HIWARE\LIB\ST7C\src into the working directory.

Figure 6. Debug and Edit Mode Buttons

The Edit mode is very easily differentiated from Debug mode in two ways. First, by the pressed button displayed in the toolbar (the blue D is highlighted when in Debug mode or the red cross in Edit mode). Secondly, in Debug mode, the line reached by the Program Counter is displayed in yellow (Debug environment). See Figure 7.

Figure 7. STVD7

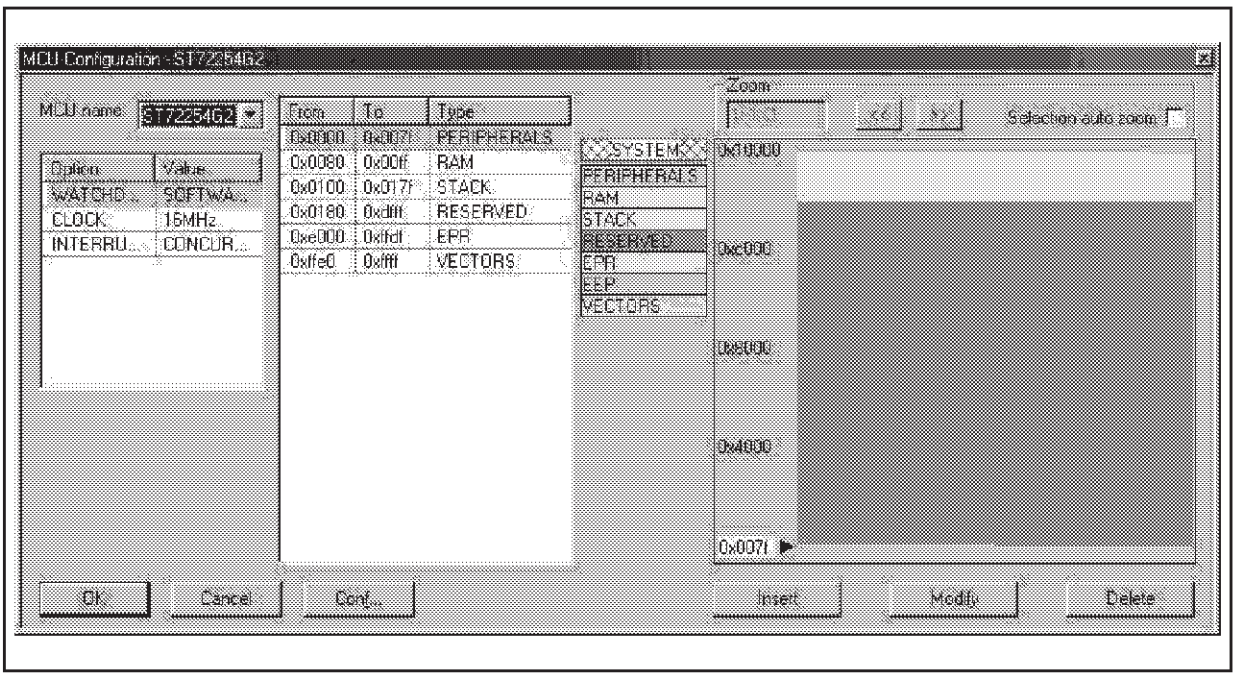


The first time Debug mode is entered, the requested information must be entered in the "**MCU Configuration**" window (see Figure 8.):

- MCU name
- CPU frequency
- LVD (on, off, level)
- Watchdog (hardware or software)...

All possible MCU configurations that are set using the option byte or certain registers when working with the MCU are listed in this window.

Figure 8. MCU Configuration Window



3 HIWARE IDE FOR ST7: PANTA

The HIWARE C Toolchain can also be used with the IDF (Intuitive Development Framework) delivered automatically with the compiler, PANTA. Then for debugging, either the STVD7 or HI-WAVE application can be used.

3.1 PANTA FEATURES

PANTA is a development framework for Embedded Systems with an intuitive user interface.

PANTA is designed to accelerate your development with a maximum of security and easy maintenance.

Due to its modular system, PANTA can be customized to best meet the user's needs. There are Compilers, Macro Assemblers, True-Time Simulators, Real-Time Debuggers, Visualization and I/O Device Simulation available. The built-in SmartLinker, Decoder, Burner, Libmaker and Maker utilities are smoothly integrated.

PANTA provides a global overview and supplies an easier way to manage projects with multiple source files. Projects can be organized in project spaces.

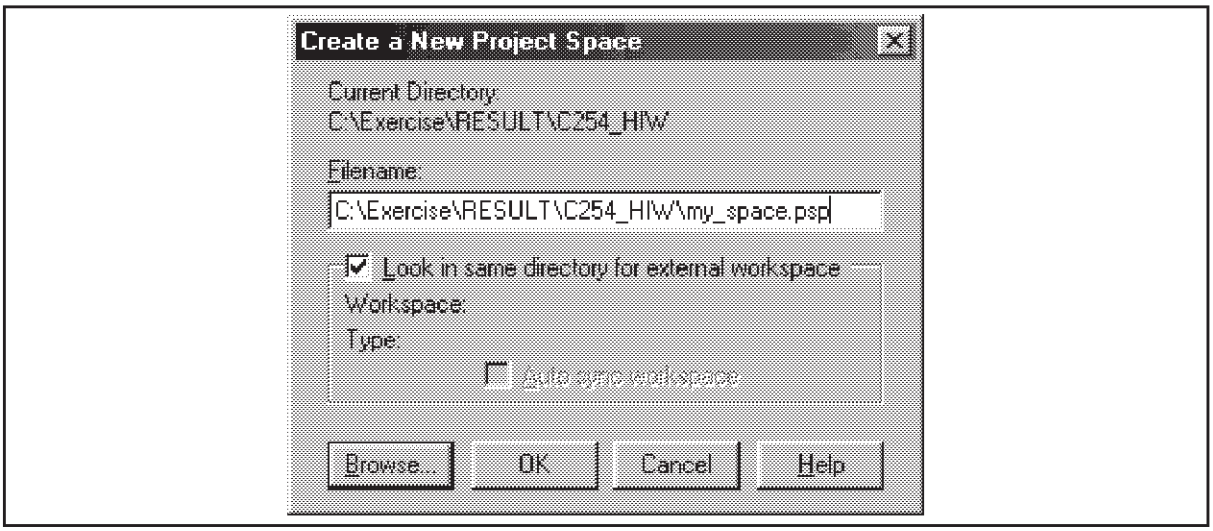
- Better and faster code
- Able to be used right out of the box
- Easy configuration to personal development framework
- Comfortable switching between projects, MCUs and targets
- Powerful Editor
- Sophisticated project management
- Automatic build process
- Build and debug in one step
- CPU-related syntax coloring Snippets for common tasks
- Link to version control Comparison tool (file compare)
- Merging tool
- File Grep
- Template expansion reducing programming time
- Code Wizard
- Synchronized simulation and debugging
- Online Help and context sensitive help
- Easy third party tool integration

3.2 GETTING STARTED WITH HIWARE AND PANTA

To use the PANTA IDE, carry out the following steps:

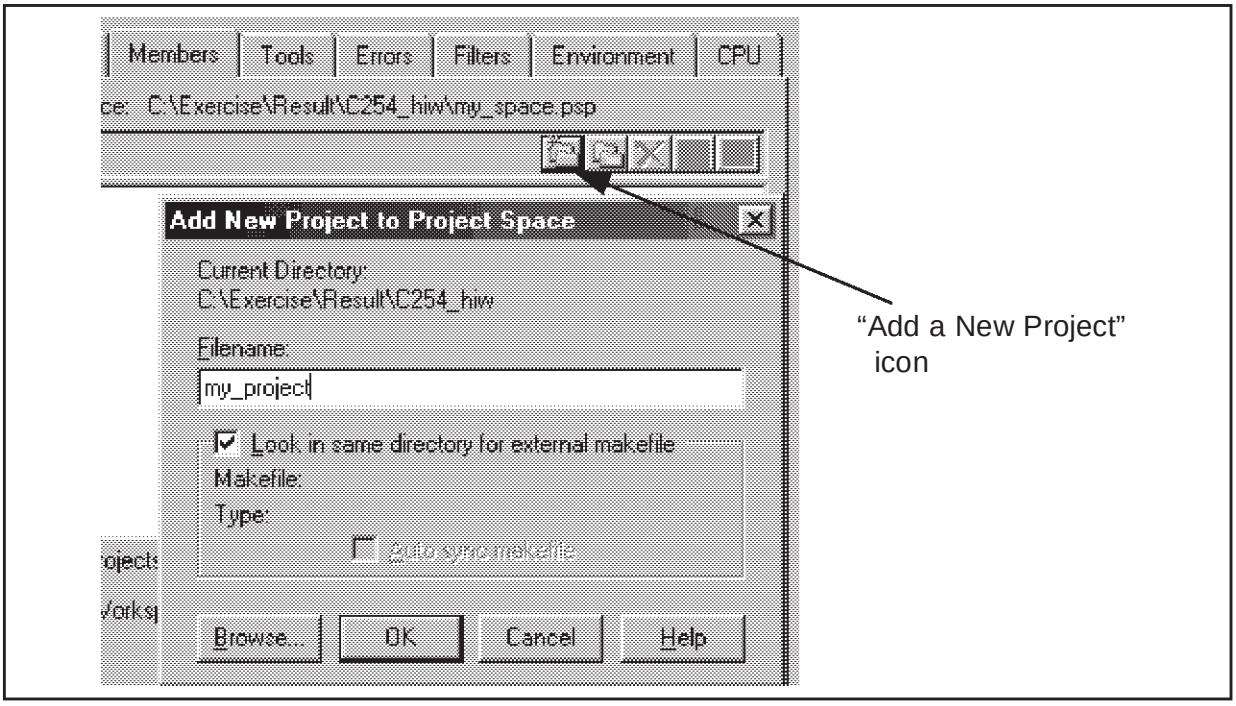
- First, create a project space. (See Figure 14.) A project space is a collection of projects. This space is used to group projects which are linked together. To create this project space, open the PANTA application and click on "*Project, Project Space, New*".
- Enter the path of the new project space. For example: C:\Exercise\Result\C254_hiw \my_space.psp (the directory tree can be selected by clicking on the *Browse* button).

Figure 9. Create a Project Space



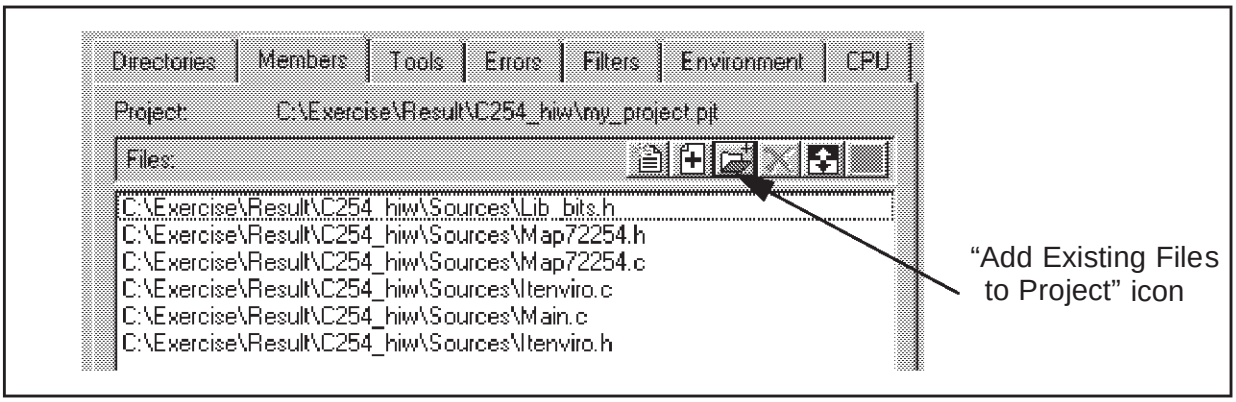
- In the Project Properties window, add a new project (same directory) by clicking on the first icon. (See Figure 10.) Enter a name for the project (for example, my_project.pjt).

Figure 10. New Project



- In the Project Properties window, in the Members sheet, add all the required source files (*.c, *.h, *.prm,...) by clicking on the third icon. (see Figure 11.)

Figure 11. Files Added to Project



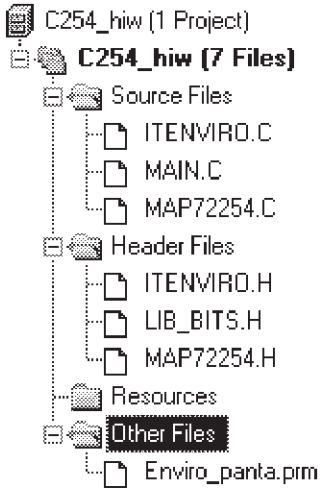
- In the CPU sheet, select ST7, do not modify the MCU Derivate Field default values (unless a specific .prm file is placed in C:\Hiware\templates\st7\prm) and select the desired debugging method (Simulator or ST Micro target interface for the DVP or emulator).
- In the Environment sheet (NB: This step is unnecessary if all the project files are located in the same directory):
  - Add a new "*.lsources*" path for the General Paths by clicking on the white cross icon. (see Figure 12.)

Figure 12. Add a Path to the Project

– Modify the existing path (".” means the current directory) for the Object Paths: "*.\object*"
– Modify the existing path for the Text Paths: "*.\object"*
– Modify the existing path for the Absolute Paths: "*.\object*"
– Add a new path for the Header File Paths: "*.\sources*"

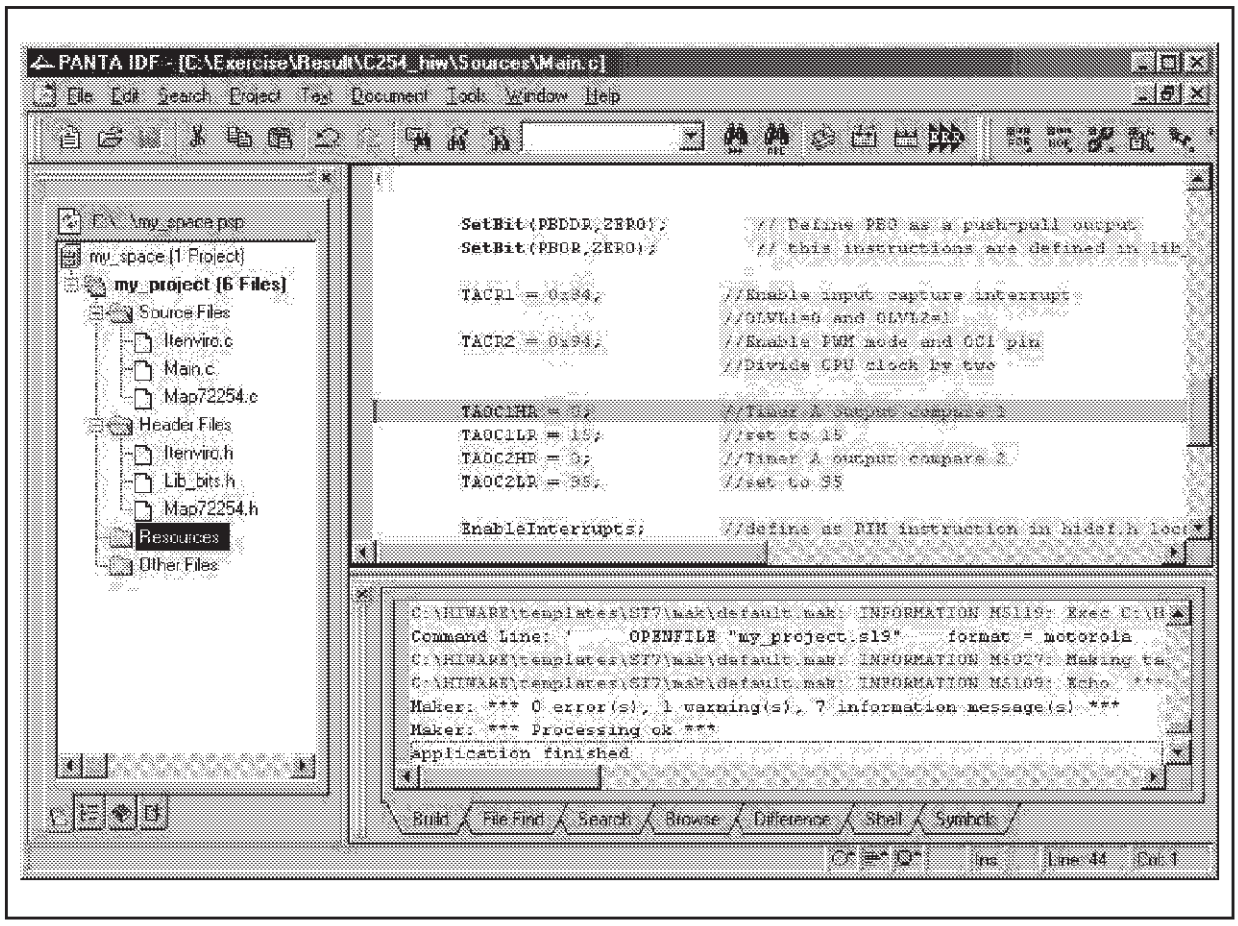
- In the Environment sheet, change the Link Parameter file to "*.\Enviro_panta.prm*".
- Exit the Project Properties window by clicking OK. The left panel display of PANTA IDE should look like Figure 13.

Figure 13. Project Files in PANTA



- Modify the sources, depending to the purpose of this exercise.
- Build by pressing the F7 key or through the "Project, Build" menu. Detected errors are displayed on a dedicated window. Double-click on the error to go directly to its location in the program. Correct the error, then rebuild.

Figure 14. PANTA Main Window



Then, the application can be debugged using the HIWAVE framework (select "*Project, Debug*").

The STVD7 Visual Debugger can also be used for debugging:

- Open the STVD7 application (simulator, DVP or emulator).
- Create a new workspace.
- Select the generated .abs file in the "Project settings" window.
- Choose the debugger you want to use in the Tools in the workspace window and then launch it (STVD7 or ZAP).

The rest is up to you!