

HT-IDE3000 User's Guide

April 2010

Copyright © 2009 by HOLTEK SEMICONDUCTOR INC. All rights reserved. Printed in Taiwan. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form by any means, electronic, mechanical photocopying, recording, or otherwise without the prior written permission of HOLTEK SEMICONDUCTOR INC

NOTICE

The information appearing in this User's Guide is believed to be accurate at the time of publication. However, Holtek assumes no responsibility arising from the use of the specifications described. The applications mentioned herein are used solely for the purpose of illustration and Holtek makes no warranty or representation that such applications will be suitable without further modification, nor recommends the use of its products for application that may present a risk to human life due to malfunction or otherwise. Holtek's products are not authorized for use as critical components in life support devices or systems. Holtek reserves the right to alter its products without prior notification. For the most up-to-date information, please visit our web site at <http://www.holtek.com.tw>.

Contents

Part I Integrated Development Environment	1
Chapter 1 Overview and Installation	3
HT-IDE Development Environment.....	3
Holtek In-Circuit Emulator (HT-ICE & e-ICE)	5
HT-ICE Interface Card.....	5
MCU Programmer	5
MCU Adapter Card	6
System Configuration.....	6
Installation	7
System Requirement	7
Hardware Installation	8
Software Installation	8
Chapter 2 Quick Start.....	13
Step 1 : Create a New Project with the CodeWizard	13
Step 2 : Build the Project.....	13
Step 3 : Programming the MCU Device.....	14
Step 4 : Transmit Code to Holtek.....	14
Chapter 3 Menu – File/Edit/View/Tools/Options	17
Start the HT-IDE3000 System.....	17
File Menu.....	20
Edit Menu	20

View Menu	21
Tools Menu	22
Configuration Option	23
Diagnose	23
Writer	24
Library Manager	25
Editor	26
Voice & Flash Download	27
LCD Simulator	28
Virtual Peripheral Manager	28
Options Menu	28
Project Settings	29
Editor Settings	33
Language	34
Chapter 4 Menu - Project	37
Create a New Project	37
Step1: Project Location	38
Step2: Project Option	39
Step3: Project Deployment	40
Open and Close a Project	41
Manage the Source Files of a Project	41
To Add a Source File to the Project	42
To Delete a Source File from the Project	42
To Move a Source File Up or Down	42
Build a Project's Task Files	43
To Build a Project Task File	44
To Rebuild a Project Task File	44
Assemble/Compile	44
To Assemble or Compile a Program	44
Print Option Table Command	45
Backup/Restore Project	45
Chapter 5 Menu - Debug	47
Reset the HT-IDE3000 System	48
To Reset from the HT-IDE3000 Commands	49

Emulation of Application Programs.....	49
To Emulate the Application Program.....	50
To Stop Emulating the Application Program	50
To Run the Application Program to a Line.....	50
To Directly Jump to a Line of an Application Program.....	50
Single Step.....	51
Breakpoints.....	52
Breakpoint Features	52
Description of Breakpoint Items	53
How to Set Breakpoints.....	55
Trace the Application Program.....	57
Initiating the Trace Mechanism	57
Stopping the Trace Mechanism	59
Trace Start/Stop Setup.....	59
Trace Record Format	62
Debugger Command Mode.....	64
Enter/Quit the Command Mode	64
Functions Supported by the Command Mode	65
Log File Format	72
HT-COMMAND Error Messages.....	73
Chapter 6 Menu - Window	75
Window Menu Commands	76
Chapter 7 Simulation.....	85
Start the Simulation	85
Chapter 8 MCU Programming	87
Introduction	87
Installation	88
Adapter Card	88
Programming an MCU Device with the EverPro K1000	89
Run the EverPro K1000 Software.....	89
EverPro K1000 Programming Functions	90
EverPro K1000 Additional Functions.....	91

Part II Development Language and Tools..... 97

Chapter 9 Assembly Language and Cross Assembler 99

 Notational Conventions 99

 Statement Syntax..... 100

 Name 100

 Operation..... 101

 Operand..... 101

 Comment..... 101

 Assembly Directives 101

 Conditional Assembly Directives..... 101

 File Control Directives 102

 Program Directives 104

 Data Definition Directives..... 107

 Macro Directives..... 109

 Assembly Instructions 111

 Name 111

 Mnemonic..... 112

 Operand, Operator and Expression..... 112

 Miscellaneous 114

 Forward References 114

 Local Labels..... 114

 Reserved Assembly Language Words..... 115

 Cross Assembler Options..... 116

 Assembly Listing File Format..... 116

 Source Program Listing..... 116

 Summary of Assembly..... 117

 Miscellaneous 117

Chapter 10 Cross Linker..... 119

 What the Cross Linker Does 119

 Cross Linker Options..... 119

 Libraries 119

 Section Address 120

 Generate Map File 120

 Map File 120

Cross Linker Task File and Debug File.....	122
Part III Utilities.....	125
Chapter 11 Library Manager.....	127
What the Library Manager Does.....	127
To Setup the Library Files.....	127
Create a New Library File	128
Add a Program Module into a Library File	129
Delete a Program Module from a Library File	129
Extract a Program Module from Library and Create An Object File	129
Object Module Information.....	130
Chapter 12 LCD Simulator.....	131
Introduction	131
LCD Panel Configuration File	131
Relationship Between the Panel File and the Current Project.....	132
Selecting the HT-LCDS	132
LCD Panel Picture File	133
Setup the LCD Panel Configuration File.....	134
Setup the Panel Configurations	134
Select the Patterns and Their Positions.....	135
Add a New Pattern.....	135
Delete a Pattern	136
Change the Pattern.....	136
Change the Pattern Position.....	137
How to Add a User-define Matrix.....	137
Define the Pattern Using the Panel Editor.....	138
Add New Pattern Items Using a Batch File	138
Selecting Color for an LCD Panel.....	139
Setting Pattern Color for VFD Panel	139
Simulating the LCD.....	140
Stop the Simulation.....	141
Chapter 13 Virtual Peripheral Manager.....	143

Introduction	143
The VPM Window	143
VPM Menu	144
File Menu.....	144
Function Menu.....	145
The VPM Peripherals.....	148
LED	148
Button/Switch.....	148
Seven Segment Display.....	149
Quick Start Example	151
Scanning Light.....	151
Chapter 14 Hi-Tech C MCU Converter	154
Hi-Tech C MCU Converter Function.....	154
Using the Hi-Tech C MCU Converter.....	154
Part IV Appendix.....	157
Appendix A Reserved Words Used By Cross Assembler.....	159
Reserved Assembly Language Words.....	159
Instruction Sets.....	161

Part I

Integrated Development Environment

Chapter 1

Overview and Installation

1

To ease the process of application development, the importance and availability of supporting tools for microcontrollers cannot be underestimated. To support its range of MCUs, Holtek is fully committed to the development and release of easy to use and fully functional tools for its full range of devices. The overall development environment is known as the HT-IDE, while the operating software is known as the HT-IDE3000. The software provides an extremely user friendly Windows based approach for program editing and debugging while the HT-ICE and e-ICE emulator hardware provides full real time emulation with multi functional trace, stepping and breakpoint functions. With a complete set of interface cards for its full device range and regular software Service Pack updates, the HT-IDE development environment ensures that designers have the best tools to maximize efficiency in the design and release of their microcontroller applications.

HT-IDE Development Environment

The Holtek Integrated Development Environment, otherwise known as the HT-IDE, is a high performance integrated development environment designed around Holtek's series of 8-bit MCU devices. Incorporated within the system is the hardware and software tools necessary for rapid and easy development of applications based on the Holtek range of 8-bit MCUs. The key component within the HT-IDE system is the HT-ICE or e-ICE In-Circuit Emulator, capable of emulating the Holtek 8-bit MCU in real time, in addition to providing powerful debugging and trace features. The new e-ICE includes an actual MCU for more effective simulation purposes.

As for the software, the HT-IDE3000 provides a friendly workbench to ease the process of

application program development, by integrating all of the software tools, such as editor, Cross Assembler, Cross Linker, library and symbolic debugger into a user friendly Windows based environment. In addition, the HT-IDE3000 provides a software simulator which is capable of simulating the behavior of Holtek's 8-bit MCU range without connection to the HT-ICE. All fundamental functions of the HT-ICE hardware are valid for the simulator.

More detailed information on the HT-IDE3000 development system is contained within the HT-IDE3000 User's Guide. Installed in conjunction with the HT-IDE3000 and to ensure that the development system contains information on new microcontrollers and the latest software updates, Holtek provides regular HT-IDE3000 Service Packs. These Service Packs, which can be downloaded from the Holtek website, do not replace the HT-IDE3000 but are installed after the HT-IDE3000 system software has been installed.

Some of the special features provided by the HT-IDE3000 include::

→ **Emulation**

- Real-time program instruction emulation

→ **Hardware**

HT-ICE

- Easy installation and usage
- Either internal or external oscillator
- Breakpoint mechanism
- Trace functions and trigger qualification supported by trace emulation chip
- MCU writer hardware integrated within the HT-ICE
- Printer port for connecting the HT-ICE to a host computer
- I/O interface card for connecting the user's application board to the HT-ICE

e-ICE

- Easy installation and usage
- Either internal or external oscillator
- Breakpoint mechanism
- USB cable for connecting the e-ICE to a host computer
- 2.54mm standard needle for connecting the user's application board to the e-ICE

→ **Software**

- Windows based software utilities
- Source program level debugger (symbolic debugger)
- Workbench for multiple source program files (more than one source program file in one application project)
- All tools are included for the development, debug, evaluation and generation of the final application program code (mask ROM file and OTP file)

- Library for the setting up of common procedures which can be linked at a later date to other projects.
- Simulator can simulate and debug programs without connection to the HT-ICE hardware
- Virtual Peripheral Manager (VPM) simulates the behavior of the peripheral devices.
- LCD simulator simulates the behavior of the LCD panel.

Holtek In-Circuit Emulator (HT-ICE & e-ICE)

Developed alongside the Holtek 8-bit microcontroller device range, the Holtek ICE is a fully functional in-circuit emulator for Holtek's 8-bit microcontroller devices. Incorporated within the system are a comprehensive set of hardware and software tools for rapid and easy development of user applications. Central to the system is the in-circuit hardware emulator, capable of emulating all of Holtek's 8-bit devices in real-time, while also providing a range of powerful debugging and trace facilities. Regarding software functions, the system incorporates a user-friendly Windows based workbench which integrates together functions such as program editor, Cross Assembler, Cross Linker and library manager. In addition, the system is capable of running in software simulation mode without connection to the HT-ICE hardware.

HT-ICE Interface Card

The interface cards supplied with the HT-ICE can be used for most applications, however, it is possible for the user to omit the supplied interface card and design their own interface card. By including the necessary interface circuitry on their own interface card, the user has a means of directly connecting their target boards to the CN1 and CN2 connectors of the HT-ICE.

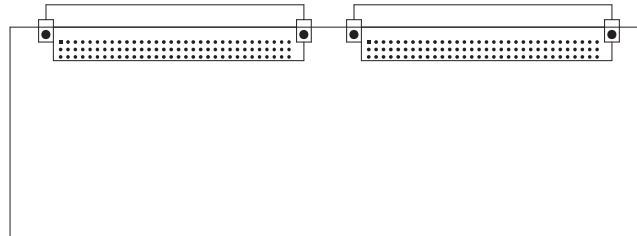


Fig 1-1

MCU Programmer

Holtek's MCU devices are fully supported by a range of programmers. For engineering level MCU device programming, Holtek supplies its stand alone programming tool which provides a quick and efficient means for low volume MCU programming. The HT-ICE

In-Circuit Emulators has integrated a writer as part of the hardware package, facilitating complete design, debug and MCU device programming all within the HT-ICE. More programmers from other suppliers are available which provide more efficient and higher volume production capability. Refer to our website for further suppliers information.

MCU Adapter Card

The Holtek MCU programmers are supplied with a standard Textool chip socket. The OTP Adapter Card is used to connect the Holtek MCU programmers to the various sizes of available MCU chip packages that are unable to use this supplied socket.

System Configuration

The HT-IDE system configuration is shown below, in which the host computer is a Pentium compatible machine with Windows 2000/XP or later. Note that if Windows 2000/XP or later systems are used, then the HT-IDE3000 software must be installed in the Supervisor Privilege mode.

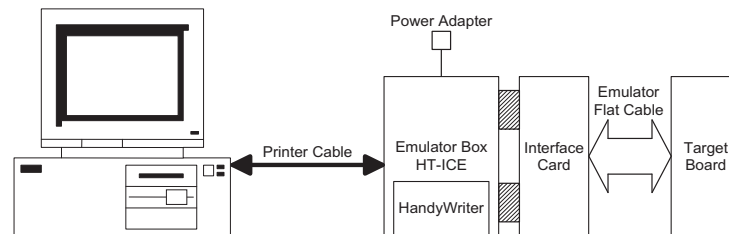


Fig 1-2

The HT-IDE system contains the following hardware components :

HT-ICE

- The HT-ICE box contains the emulator box with 1 printer port connector for connecting to the host machine, I/O signal connector and one power-on LED
- I/O interface card for connecting the target board to the HT-ICE box
- Power Adapter, output 16V
- 25-pin D-type printer cable
- Integrated MCU writer

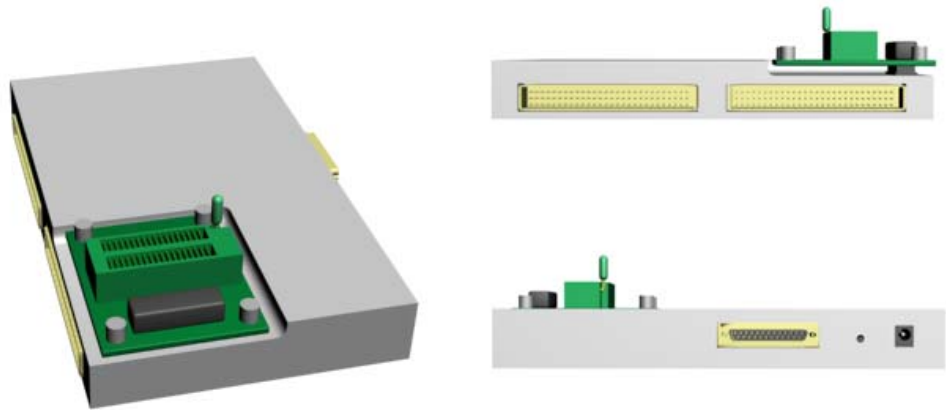


Fig 1-3

e-ICE

- The e-ICE basically consists of two boards, a mother board, known as the MEV, and into which is plugged a device daughter board, known as the DEV.
- 5-pin Mini-B USB cable

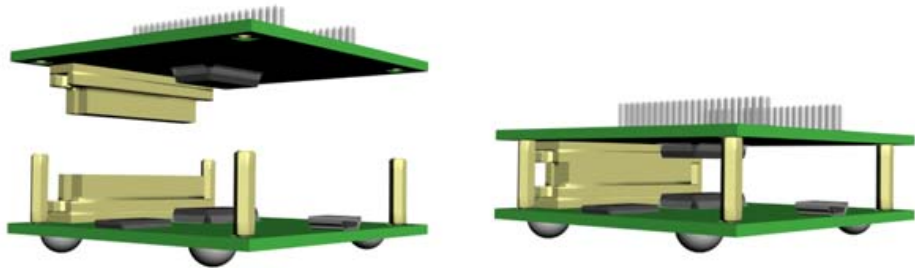


Fig 1-4

Installation

System Requirement

The hardware and software requirements for installing HT-IDE3000 system are as follows:

- PC/AT compatible machine with Pentium or higher CPU
 - SVGA color monitor
 - At least 256M RAM for best performance
 - CD ROM drive (for CD installation)
 - At least 200M free disk space
 - Parallel or USB port to connect PC and ICE
 - Windows XP/Vista/7
- * Windows XP/Vista/7 are trademarks of Microsoft Corporation

Hardware Installation

Holtek provides two kinds of ICE for the user to choose, as follows:

HT-ICE

- Step 1
Plug the power adapter into the power connector of the HT-ICE
- Step 2
Connect the target board to the HT-ICE by using the I/O interface card or flat cable
- Step 3
Connect the HT-ICE to the host machine using the printer cable. The LED on the HT-ICE should now be lit, if not, there is an error and your dealer should be contacted.

Caution: Exercise care when using the power adapter. Do not use a power adapter whose output voltage is not 16V, otherwise the HT-ICE may be damaged. It is strongly recommended that only the power adapter supplied by Holtek be used. First plug the power adapter to the power connector of the HT-ICE.

e-ICE

- Step 1
Install the correct DEV board for the MCU to be emulated
- Step 2
Use the USB cable to connect the e-ICE to the PC. The LED on the HT-ICE should now be lit, if not, there is an error and your dealer should be contacted.

Software Installation

- Step 1
First click on the IDE3000 install icon to start the Holtek HT-IDE3000 installation.

- Step 2

Press the <Next> button to continue setup or press <Cancel> button to abort.

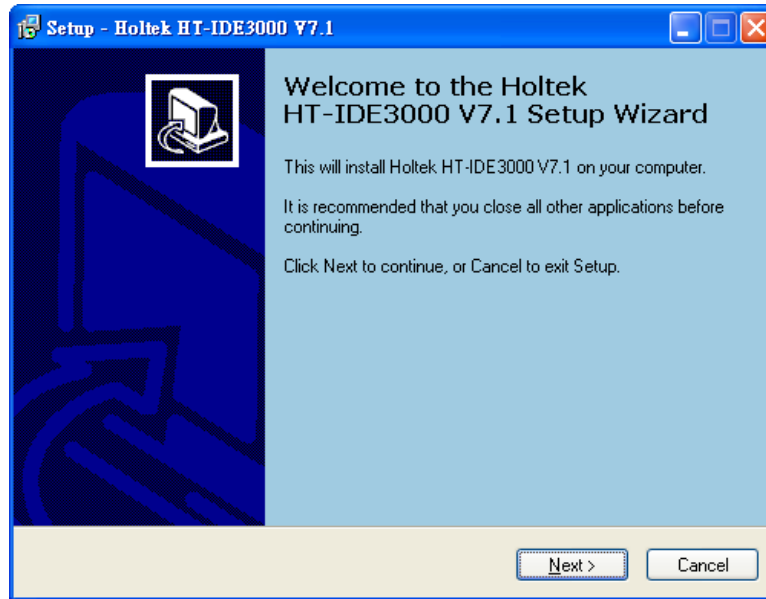


Fig 1-5

- Step 3

The following dialogue will be shown to ask the user to enter a directory name.

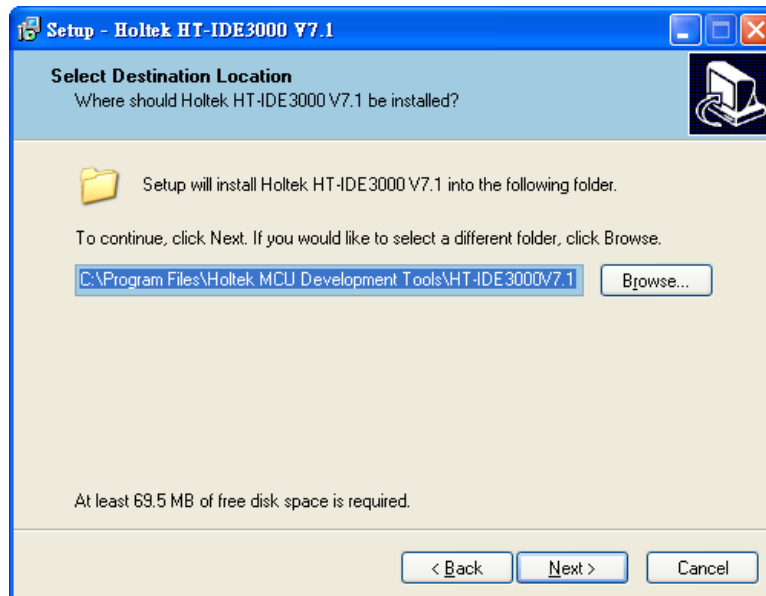


Fig 1-6

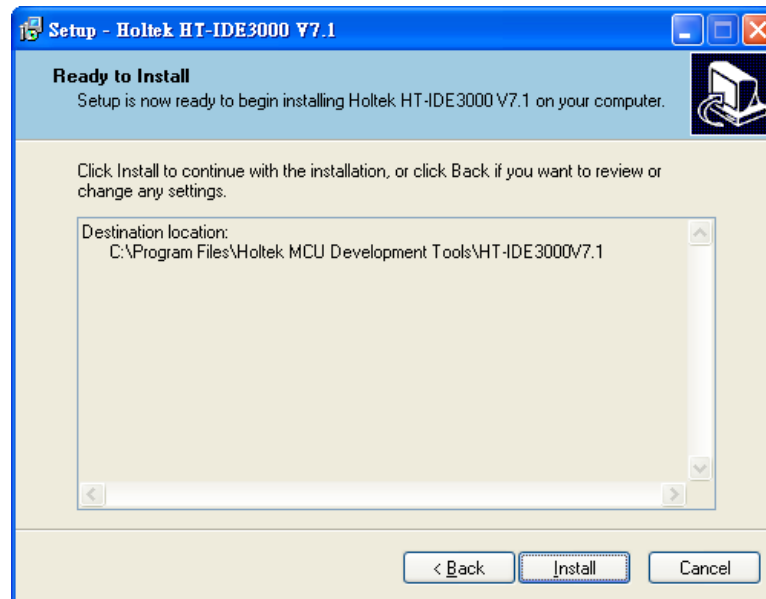


Fig 1-7

- Step 4
Specify the path you want to install the HT-IDE3000 to and click the <Next> button.
- Step 5
SETUP will copy all files to the specified directory.

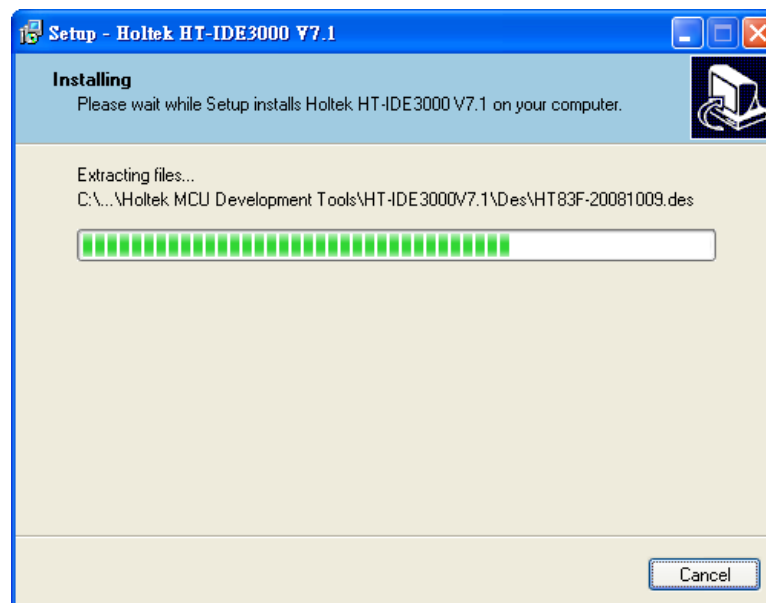


Fig 1-8

- Step 6

If the process is successful the following dialogue box will be shown.

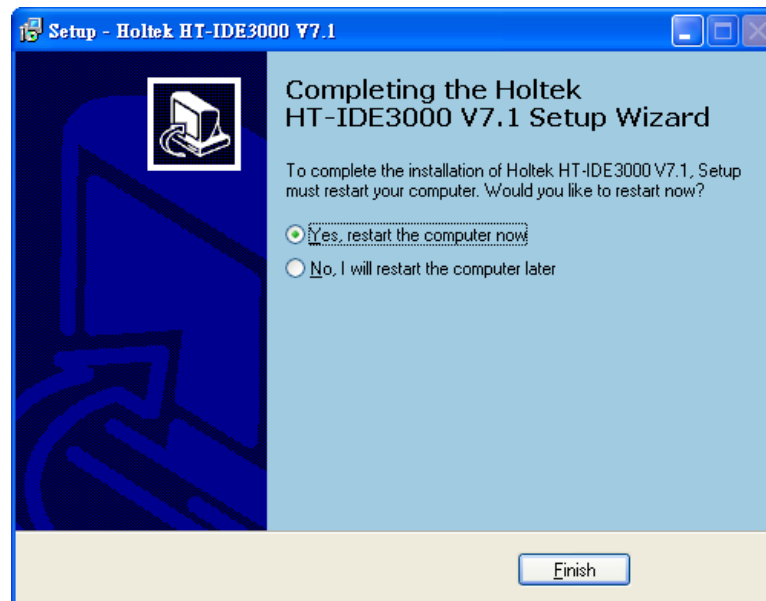


Fig 1-9

- Step 7

Press the Finish button and restart the computer system, after which the HT-IDE3000 can be run.

Chapter 2

Quick Start

2

This chapter gives a brief description of using HT-IDE3000 to develop an application project.

Step 1 : Create a New Project with the CodeWizard

- Click on the Project menu and select New command
- Enter your project name and select an MCU from the combo box
- Choose the file type that is either .ASM or .C.
- Click on the Next button and the system will ask you to setup the configuration options
- Setup all configuration options and click on the OK button.
- Finally, click OK when you have confirmed the Project Setting options.

Step 2 : Build the Project

- Click on Project menu and select the Build command
- The system will assemble/compile all source files in the project
 - If there are errors in the programs, double click on the error message line and the system will prompt you to the position where the error has occurred.
 - If all the program files are error free, the system will create a Task file and download it to the HT-ICE for debug.
- These steps can be repeated until the program is fully debugged.

Step 3 : Programming the MCU Device

- Build the project to create the .OTP file
- Click on the Tools menu and select the Writer command to program the OTP devices

Step 4 : Transmit Code to Holtek

- Click on the Project menu and select the Print Option Table command
- Send the .COD file and the Option Approval Sheet to Holtek

The Programming and data flow is illustrated by the following diagram:

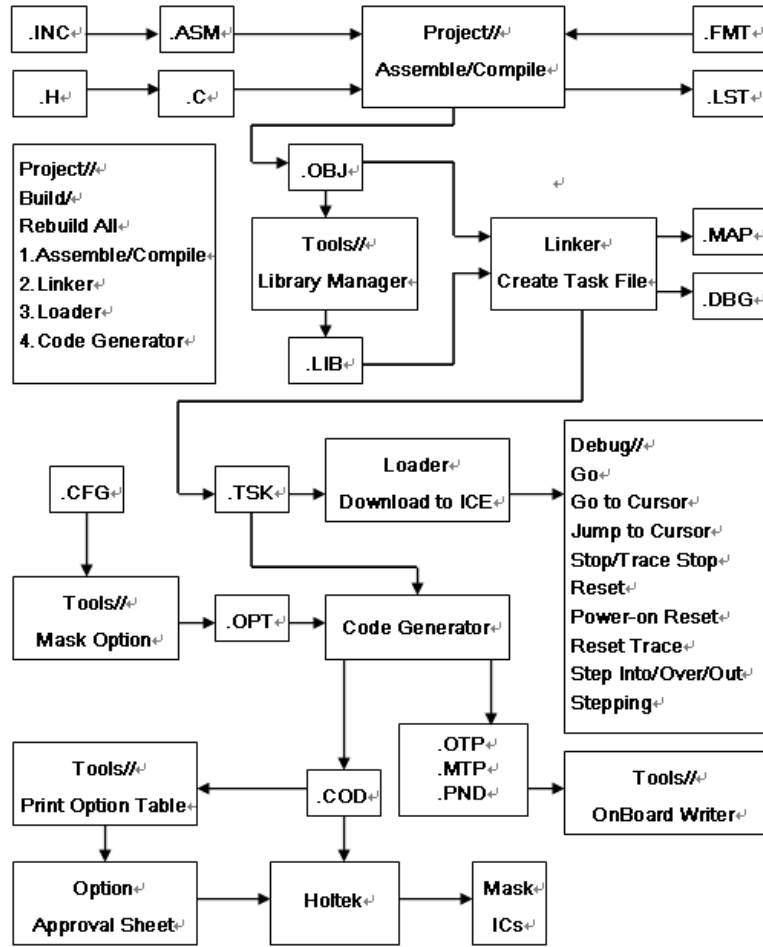


Fig 2-1

Chapter 3

Menu –

File/Edit/View/Tools/Options

3

This chapter describes some of the menus and commands of the HT-IDE3000. Other menus are described in the Project, Debug and Window chapters.

Start the HT-IDE3000 System

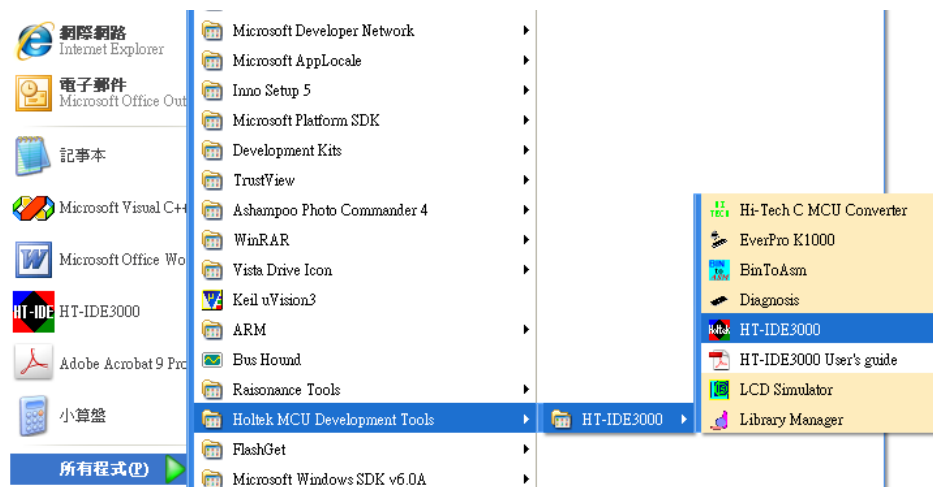


Fig 3-1

- Click the Start Button, select Programs and select Holtek HT-IDE3000
 - Click the HT-IDE3000 icon

- If the last project you worked on HT-IDE3000 is in emulation mode (using HT-ICE), then Fig 3-2 will be displayed if one of the following conditions occurs.
 - No connection between the HT-ICE and the host machine or connection fails.
 - The HT-ICE is powered off.



Fig 3-2

If “YES” is selected and the connection between the HT-ICE and the host machine has been made, then Fig 3-3 will be displayed, the HT-IDE3000 will enter the emulation mode and the HT-ICE begins to function.



Fig 3-3

- If the last project you work on HT-IDE3000 is in simulation mode (using Simulator), then Fig 3-4 will be displayed to indicate that HT-IDE3000 will enter the simulation mode.



Fig 3-4

The HT-IDE3000 software includes File, Edit, View, Project, Build, Debug, Tools, Options, Window and Help menus. The following sections describe the functions and commands of each menu.

A dockable toolbar, below the menu bar (Fig 3-5), contains icons that correspond to, and assist the user with more convenient execution of frequently used menu

commands. When the cursor is placed on a toolbar icon, the corresponding command name will be displayed alongside. Clicking on the icon will cause the command to be executed.

A Status Bar, in the bottom line (Fig 3-5), displays the emulation or simulation present status and the resulting command status. In the status bar, the field (PC=0001H) displays the Program Counter while in debugging process (Debug menu).

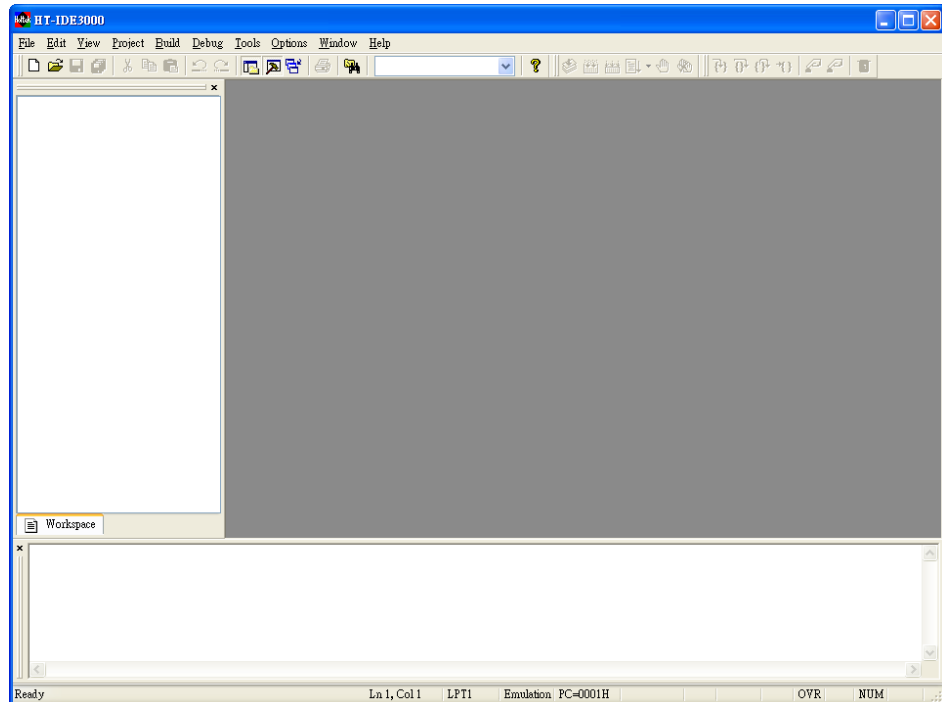


Fig 3-5

The Status Bar contains information that may be useful during program debug. The Program Counter is used during program execution and indicates the actual present Program Counter value while the row and column indicators are used to show the present cursor position when using the program editor.

File Menu



The File menu provides file processing commands, the details behind which are shown in the following list along with the corresponding toolbar icons.

- New
Create a new file
- Open
Open an existing file
- Close
Close the current active file
- Save
Write the active windows data to the active file
- SaveAs
Write the active windows data to the specified file
- Save All
Write all windows data to the corresponding opened files
- Print
Print active data to the printer
- Print Setup
Setup printer
- Recent Files
List the most recently opened and closed four files
- Exit
Exit from HT-IDE3000 and return to Windows

Edit Menu



- Undo
Cancel the previous editing operation

- Redo
Cancel the previous Undo operation
- Cut
Remove the selected lines from the file and place onto the clipboard
- Copy
Place a copy of the selected lines onto the clipboard
- Paste
Paste the clipboard information to the present insertion point
- Delete
Delete the selected information
- Select All
Select the entire document
- Find
Search the specified word from the editor active buffer
- Find Next
Find the next occurrence of the specified text
- Find Previous
Find the previous occurrence of the specified text
- Find in Files
Search for a string in multiple files
- Replace
Replace the specified source word with the destination word in the editor active buffer
- Go To...
Moves to a specified location
- Read Only
Read only mode

View Menu

The View menu provides the following commands to control the window screen of the HT-IDE3000. (Refer to Fig 3-6)

- Full Screen
Toggles Full Screen Mode on/off
- Toolbar

Display the toolbar information on the window. The toolbar contains some groups of buttons whose function is the same as that of the command in each corresponding menu item. When the mouse cursor is placed on a toolbar button, the corresponding function name will be displayed next to the button. If the mouse is clicked, the command will be executed. Refer to the corresponding chapter for the functionality of each button. The Toggle Breakpoint button will set the line specified by the cursor as a breakpoint (highlighted). The toggle action of this button will clear the breakpoint function if previously set.

- Status Bar
Displays the status bar information on the window.
- Display Line Numbers
Toggle line numbering on and off in your code.
- Cycle Count
Count instruction cycles accumulatively. Press the reset button to clear the cycle count. The Hex and Dec buttons are used to change the radix of the count, hexadecimal or decimal. The maximum cycle count is 65535.

Note: There is a slight difference of maximum cycle count between two kinds of ICE, the maximum cycle count of e-ICE can up to 4294967295 while HT-ICE can only count to 65535.

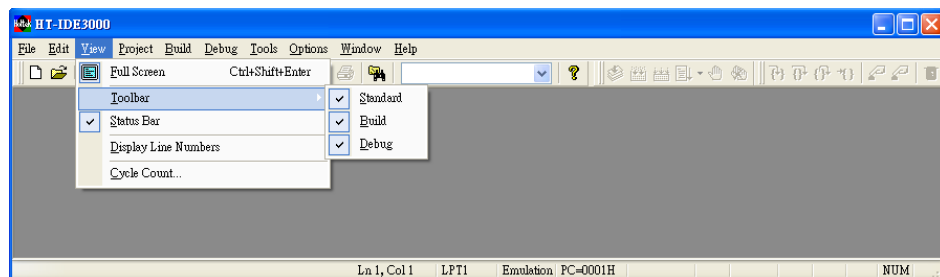


Fig 3-6

Tools Menu

The Tools menu provides the special commands to facilitate user application debug. These commands are Configuration Option, Diagnose, Writer, Library Manager, Voice

tools and LCD Simulator and virtual peripheral manager.

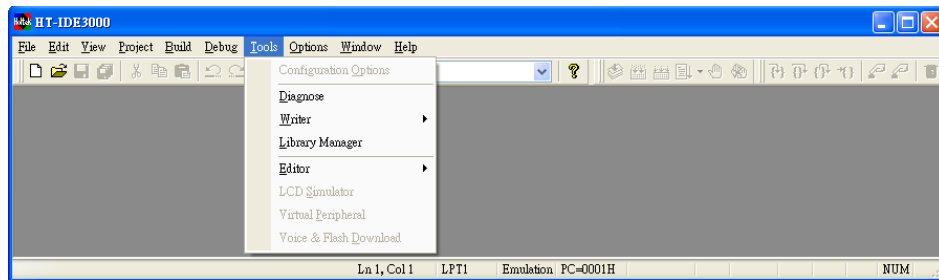


Fig 3-7

Configuration Option

This command generates an option file used by the Build command in the Project menu. The contents of the option file depend upon the specified MCU. This command allows options to be modified after creation of the project.

→ **Choosing the Clock Source**

When creating a new project or modifying the configuration options, it is necessary to choose an internal or external clock source for ICE.

If an internal clock source is used, the system application frequency has to be specified. The HT-IDE3000 system will calculate a frequency which can be supported by the HT-ICE, one which will be the most approximate value to the specified system frequency. Whenever the calculated frequency is not equal to the specified frequency, a warning message and the specified frequency along with the calculated frequency will be displayed. Confirmation will then be required to confirm the use of the calculated frequency or to specify another system frequency. Otherwise an external clock source is the only option. No matter which kind of clock source is chosen, the system frequency must be specified.

Note: More information about choosing the clock source for e-ICE, please refer to the e-ICE User's Guide.

Diagnose

This command (Fig 3-8) helps to check whether the HT-ICE is working correctly. There are a total of 9 items for diagnosis. Multiple items can be selected by clicking the check box and pressing the Test button, or press the Test All button to diagnose all items. These items are listed below.

- MCU resource option space

- Diagnose the MCU options space of the HT-ICE.
- Code space
 - Diagnose the program code memory of the HT-ICE.
- Trace space
 - Diagnose the trace buffer memory of the HT-ICE.
- Data space
 - Diagnose the program Data Memory of the HT-ICE.
- System space
 - Diagnose the system Data Memory of the HT-ICE.
- I/O EV 0
 - Diagnose the I/O EV-chip in socket 0 of the HT-ICE.
- I/O EV 1
 - Diagnose the I/O EV-chip in socket 1 of the HT-ICE.
- I/O EV 2
 - Diagnose the I/O EV-chip in socket 2 of the HT-ICE.
- I/O EV 3
 - Diagnose the I/O EV-chip in socket 3 of the HT-ICE.

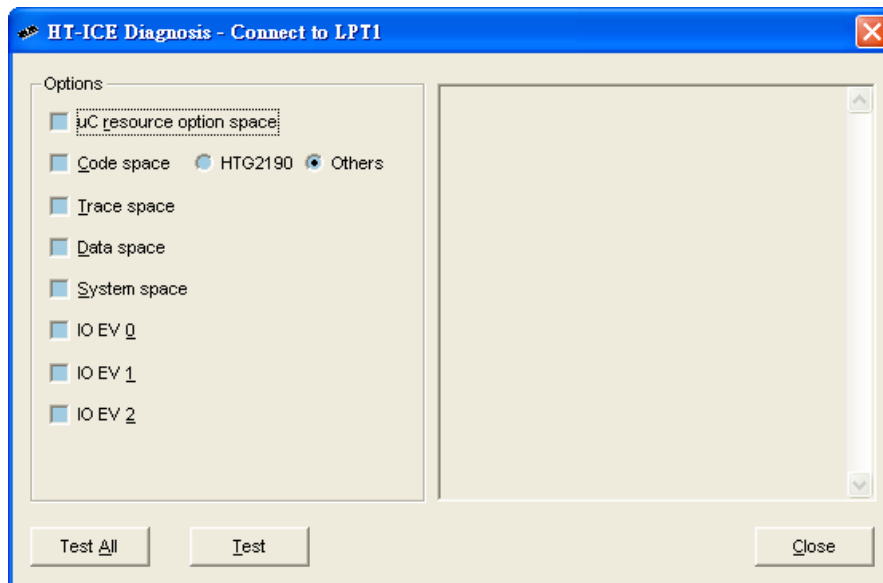


Fig 3-8

Writer

The Writer command under the Tools menu controls the OTP/MTP programming

functions of the HT-ICE built-in writer. Within this command, the sub command EverPro K1000 is used to program most of OTP/MTP type MCUs. However, this command is not applicable for the other external stand-alone writer which is known as the e-Writer (HOPE3000). Visit the Holtek website for the relevant information.

Library Manager

The Library Manager command, in Fig 3-9, supports the library functions. Program codes used frequently can be compiled into library files and then included in the application program by using the Project command in the Options menu. (Refer to the Cross Linker options item in the Options menu, Project command). The functions of Library Manager are:

- Create a new library file or modify a library file
- Add/Delete a program module into/from a library file
- Extract a program module from a library file, and create an object file

Part III gives more details on the library manager.

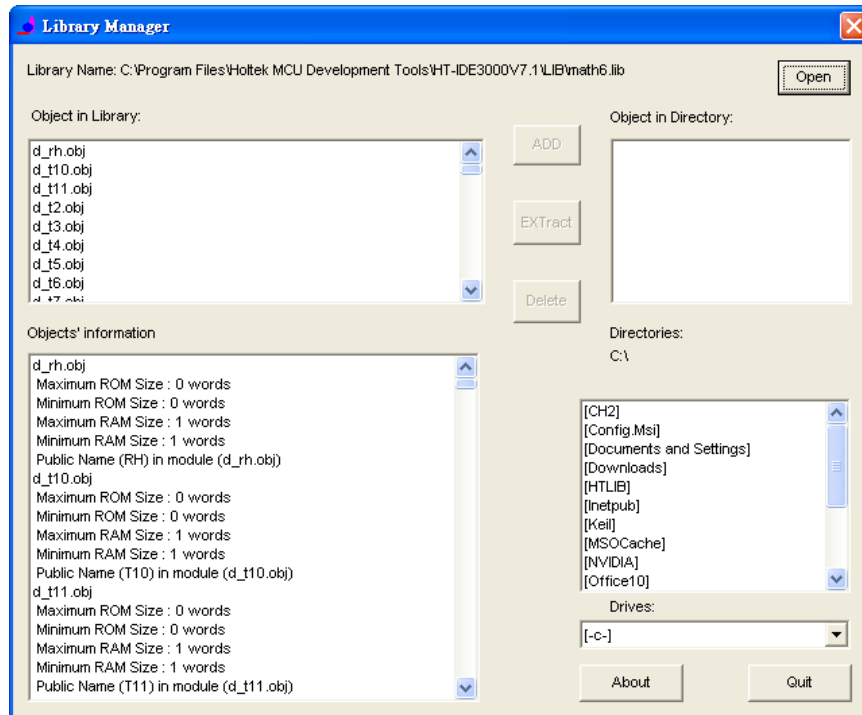


Fig 3-9

Editor

- **Voice ROM Editor**
 - Holtek provides a VROM Editor for the user to arrange the voice code for the specific MCU (eg. The HT86 series)

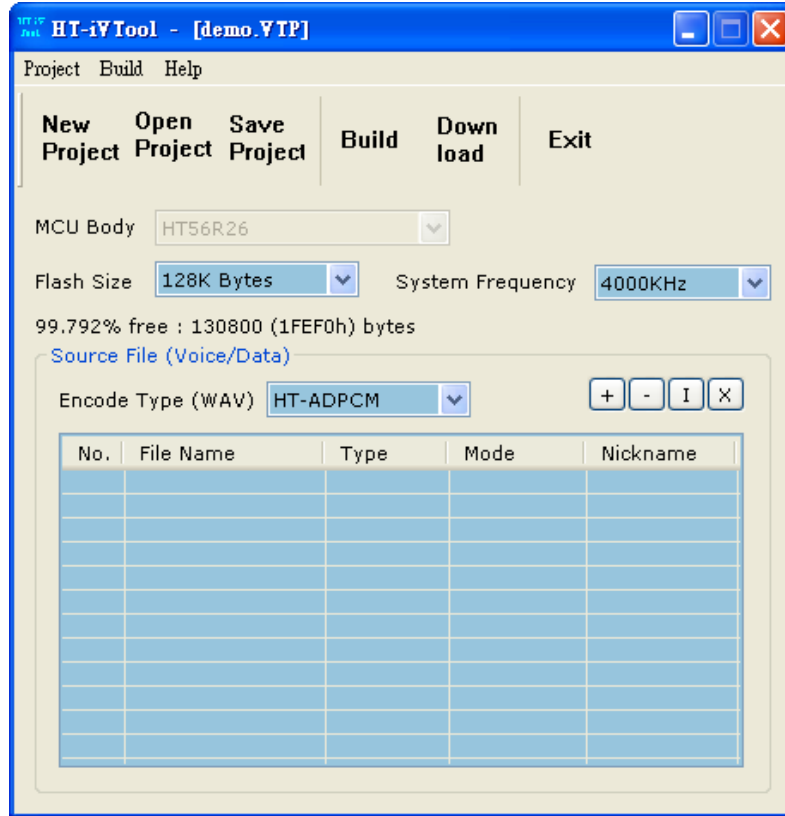


Fig 3-10

- **Data Editor**
 - Some Holtek MCUs (eg. the HT48E series) include internal EEPROM memory. The Data EEPROM Editor provides an interface for the user to arrange the data and download/upload the data to/from the HT-ICE.

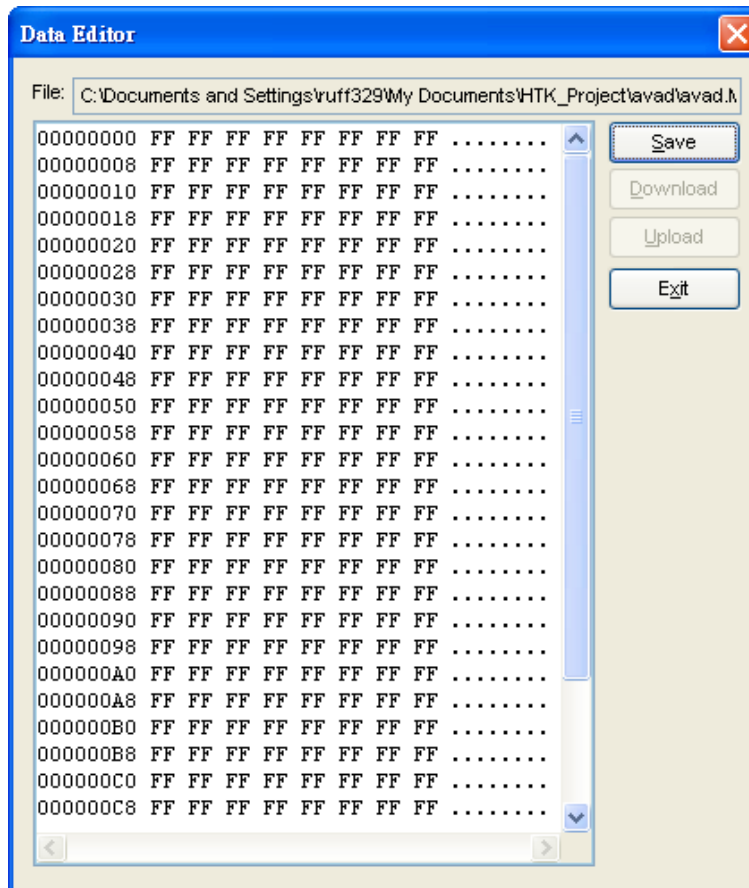


Fig 3-11

Voice & Flash Download

The Voice & Flash Download downloads the contents of a specified voice data file with the extension .VOC or .DAT to the ICE for emulation or burn the voice data to SPI Flash by e-Writer. It also uploads from ICE VROM or SPI Flash saving the data to a specified .VOC or .DAT file. Fig 3-12 displays the dialog box which shows the name of the download voice .VOC, which was generated by the VROM editor. The File Size box, below the File Path box, displays the voice ROM size in bytes for microcontroller device in the current project. Ensure that the voice file .VOC has been generated by the VROM editor before downloading.

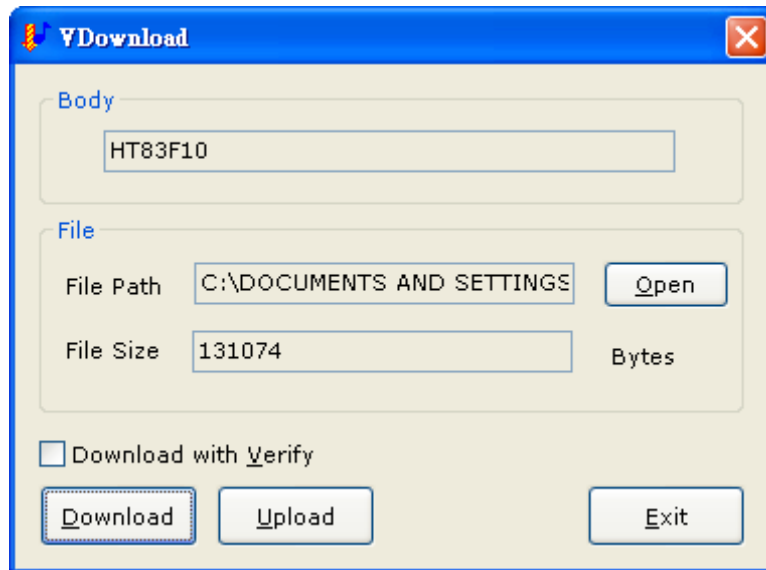


Fig 3-12

LCD Simulator

The LCD simulator, HT-LCDS, provides a mechanism to simulate the output of the LCD driver. According to the designed patterns and the control programs, the HT-LCDS displays the patterns on the screen in real time. Part III gives more details on the LCD simulator.

Virtual Peripheral Manager

The Virtual Peripheral Manager (VPM) provides a mechanism to simulate certain peripheral devices. It can only be used when the HT-IDE3000 is in the simulation mode.

Options Menu

The Options menu (Fig 3-13) provides the following commands which can set the working parameters for other menus and commands.

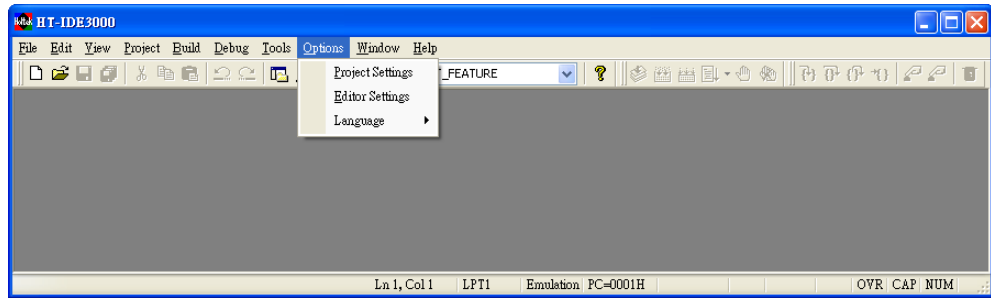


Fig 3-13

Project Settings

→ Project Option

The Project Option sets the default parameters used by the Build command in the Project menu. During development, the project options may be changed according to the needs of the application. According to the options set, the HT-IDE3000 will generate a proper task file for these options when the Build command of the Project menu is issued. The dialog box (Fig 3-14) is used to set the Project options.

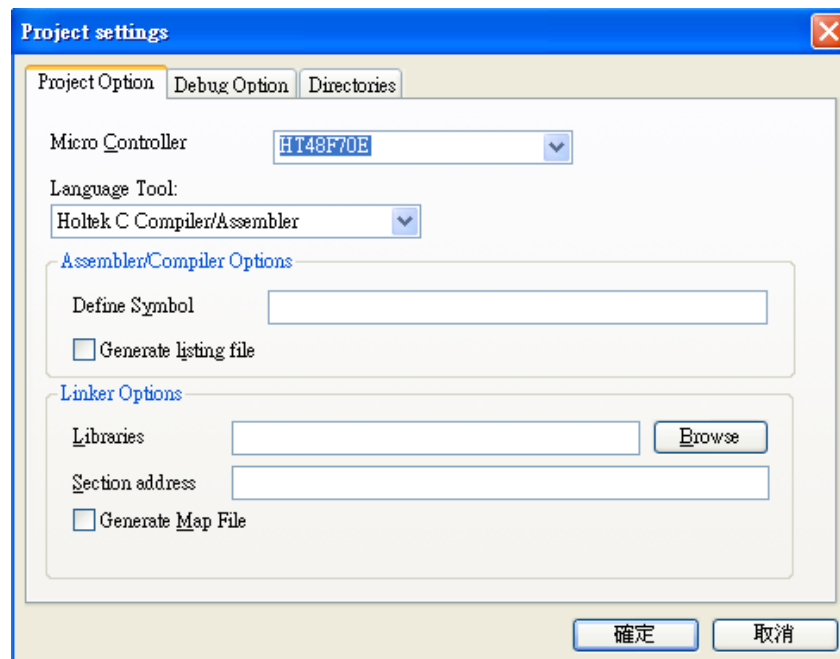


Fig 3-14

Note: Before issuing the Build command, ensure that the project options are set correctly.

- **Micro Controller**

The chosen MCU for this project is selected here. Use the scroll arrow to browse the available MCUs and select the appropriate one.

- **Language Tool**

Holtek permits Third Parties to provide C compilers for Holtek's MCUs. Here the Hi-Tech language tool can be selected as an alternative choice.

- **Assembler/Compiler Options**

The command line options of the Cross Assembler. Define symbol allows users to define values for specified symbols used in assembly programs. The syntax is as follows:

symbol1[=value1] [,symbol2 [=value2] [...]]

For example :

debugflag=1, newver=3

The check box of the Generate listing file is used to check if the source program listing file has been generated.

- **Linker options**

To specify the options of the Cross Linker. Libraries are used to specify the library files referred by Cross Linker. For example :

libfile1, libfile2

Library files can be selected by clicking the Browse button.

Section address is used to set the ROM/RAM addresses of the specified sections, for example :

codesec=100, datasec=40

The check box of the Generate map file is used to check if the map file of Cross Linker is generated.

→ **Debug Command**

This command sets the options used by the Debug menu. The dialog box (Fig 3-14) lists all the debug options with check boxes. By selecting the options and pressing the OK button, the Debug menu can then obtain these options during the debugging process.

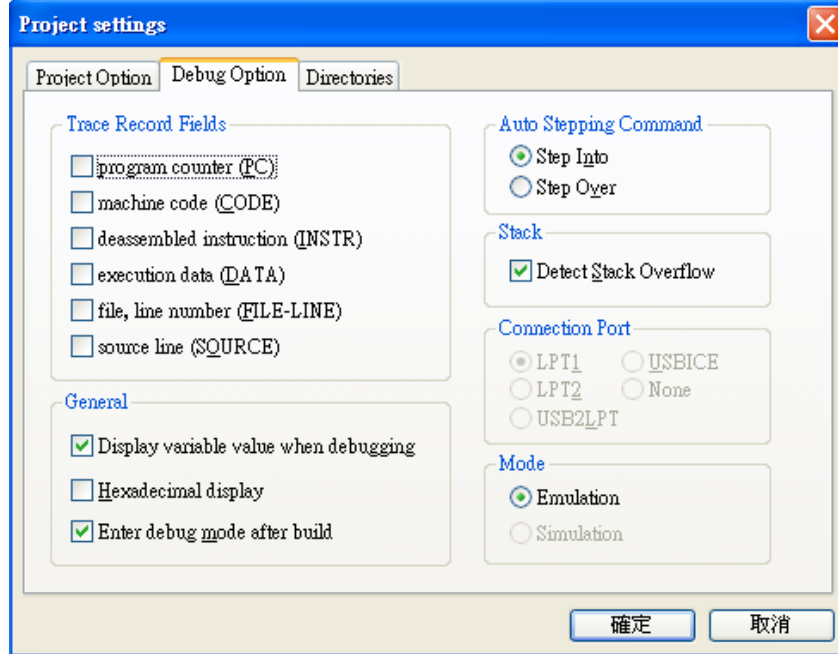


Fig 3-15

- Trace Record Fields

This location specifies the information to be displayed when issuing the Trace List command, contained within the Window menu. For each source file instruction, the information will be displayed in the same order as that of the items in the dialog box, from top to the bottom. If no item has been selected, the next selected item will be moved forward. The default trace list will display the file name and line number only. The de-assembled instruction is obtained from the machine code, and the source line is obtained from the source file. The execution data is the read data if the execution is a read operation only, and it is the written data if the execution is a write only or read and write operation. The external signal status has no effect if the simulation mode is selected.
- General

Several items are used to display certain actions when in the debug mode, such as displaying variable values, hexadecimal displays and entering the debug mode after a Build process.

- Auto Stepping Command
Selects the automatic call procedure step option, namely Step Into or Step Over. Only one option can be selected.
- Stack
Uncheck this Detect Stack Overflow box if you do not want the system to show a message while detecting a stack overflow.
- Connection Port
Display the PC connection port for the ICE. The connection port has no effect if the simulation mode is selected.
- Mode
Selects the HT-IDE3000 working mode as either simulation or emulation mode. If the HT-ICE is connected to the host machine and powered on, the HT-IDE3000 can be selected to be either in emulation or simulation mode.

→ **Directories Command**

The command sets the default search path and directories for saving files. (Fig 3-16)

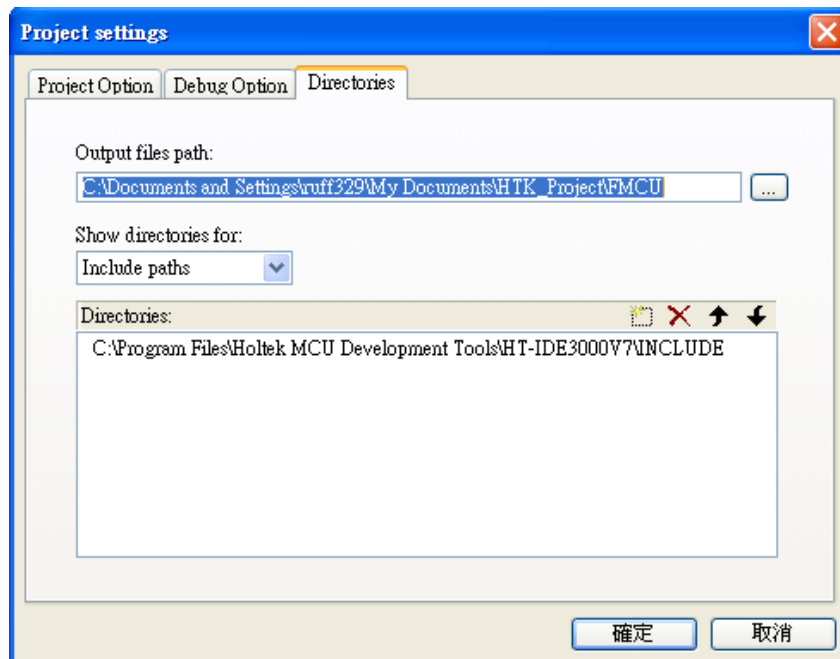


Fig 3-16

- Include files path
The search path referred to by the Cross Assembler to search for the included files.

- Library files path
The search path referred to by the Cross Linker to search for the library files.
- Output files path
The directory for saving the output files of the Cross Assembler (.obj, .lst) and Cross Linker (.tsk, .map, .dbg)

Editor Settings

→ Editors

This command sets the editor options such as tab size and the Undo command count. The Save Before Assemble option will save the file before assembly. The Maximum Undo Count is the maximum allowable counts of consecutive undo operations.

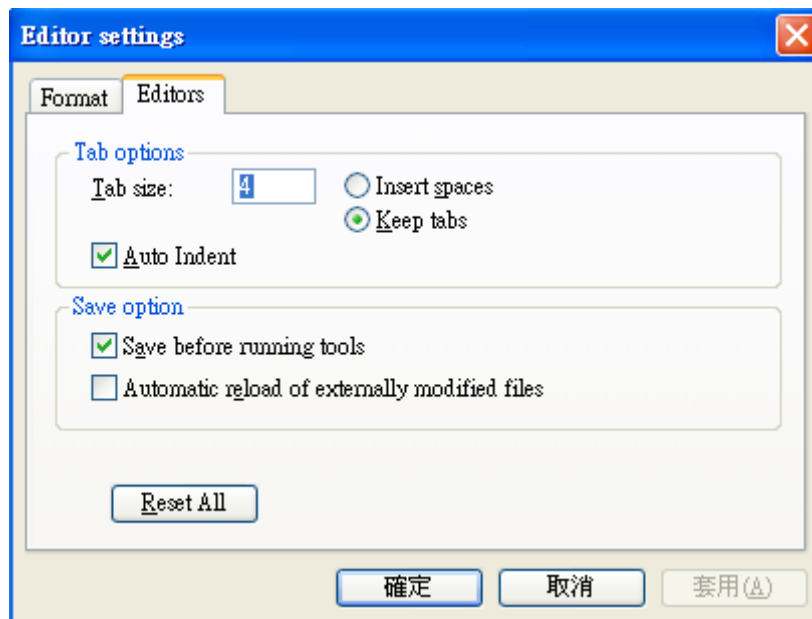


Fig 3-17

→ Format

This command sets the foreground and background colours for the specified category. From the available options (Fig 3-17), Text Selection is used for the Edit menu, Current line, Breakpoint Line, trace Line and Stack Line are for the Debug menu and Error line is for the Assembler output.

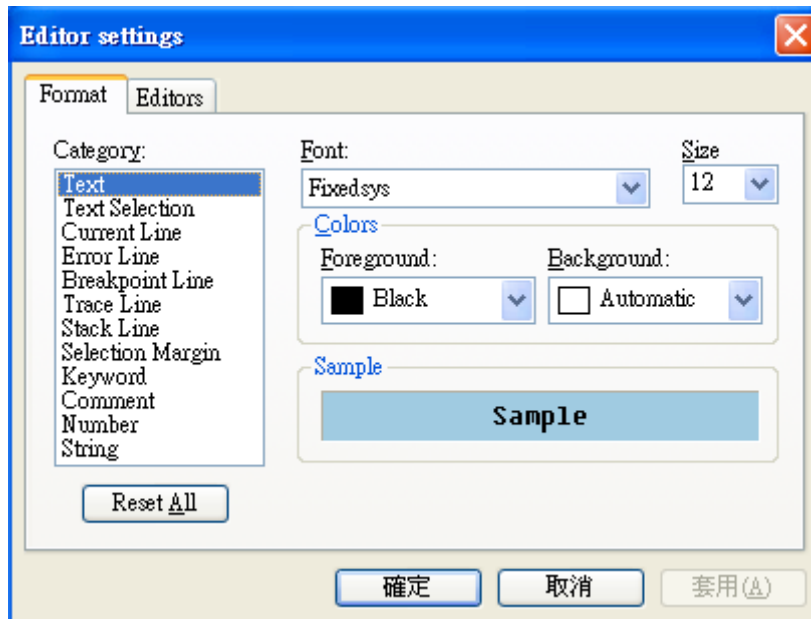


Fig 3-18

Language

This command changes the language of the user interface. 'Default' is the language of the operation system. After changing the language, you must restart the HT-IDE3000 to take it effect.

Chapter 4

Menu - Project & Build

4

The HT-IDE3000 provides an example Project, which will assist first time users in quickly familiarizing themselves with project development. It should be noted that from the standpoint of the HT-IDE3000 system, a working unit is a project with each user application described by a unique project.

When developing an HT-IDE3000 application for the first time, the development steps, as described earlier, are recommended.

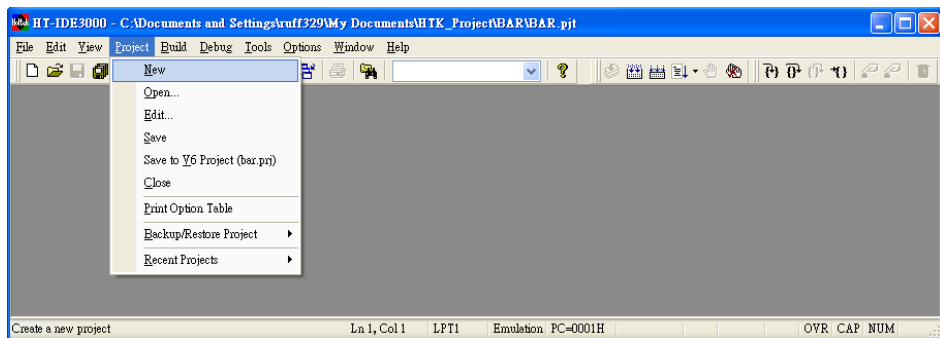


Fig 4-1

Create a New Project



In the Project menu (Fig 4-1), select the New command to create a new project. This command will call the CodeWizard to assist users to create a new project.

Note: The project name is a file name with the extension .PJT and .PJTX.

CodeWizard flowchart

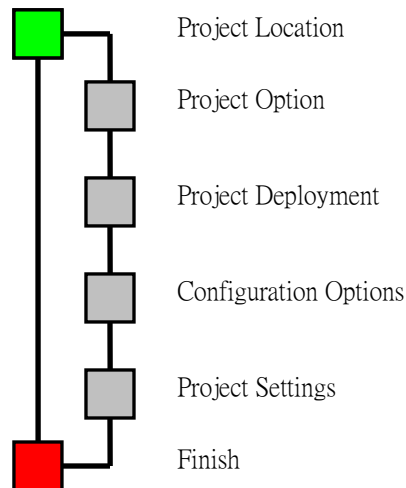


Fig 4-2

Step1: Project Location

This step will require the user to input a project name and select a Microcontroller, see Fig. 4-3. Users can access all of their folders and saved files to select an already existing project or can instead input a new project name. Additionally users can select the required microcontroller for their project and also select the compiler tools. If the user wishes to construct an empty project in advance then the More project settings box should be unchecked.

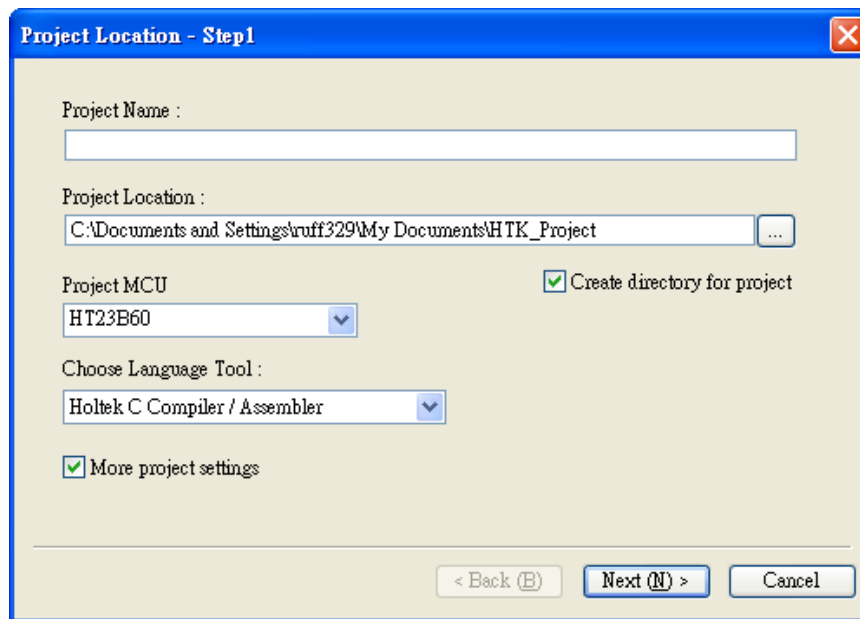


Fig 4-3

Step2: Project Option

The second step is to select whether assembly files or C-language files are to be used in the project.

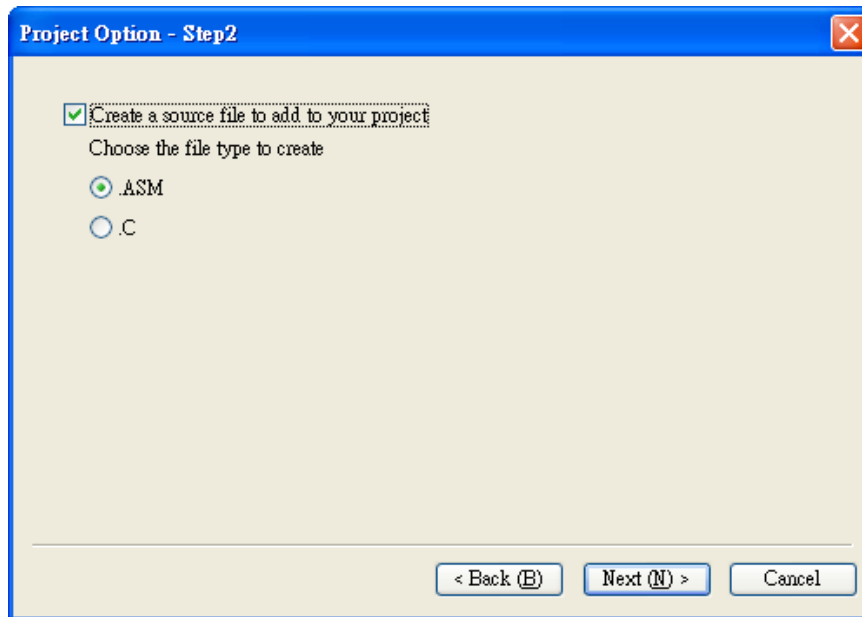


Fig 4-4

Step3: Project Deployment

This step is to change the source program File name, program section and data section.

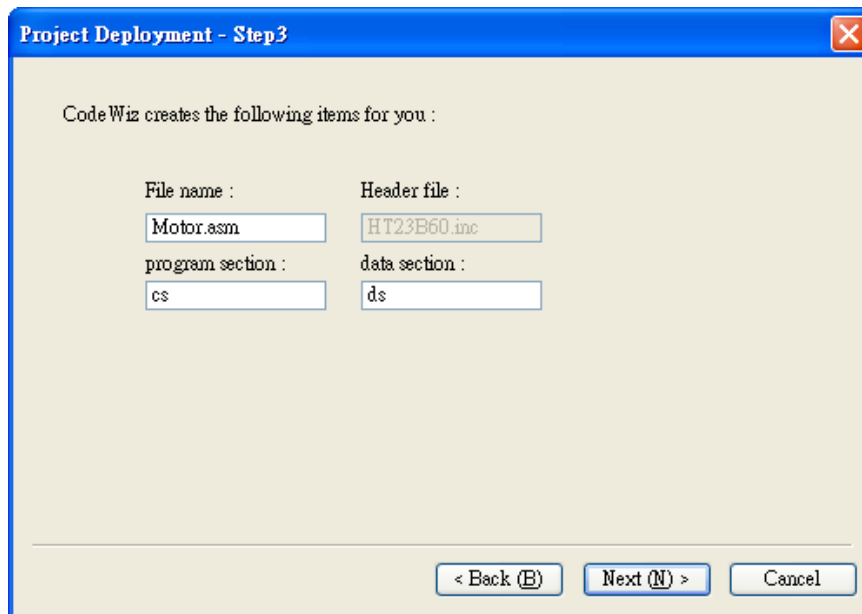


Fig 4-5

Finally is the Configuration Option and Project Setting operation, for this consult the related chapters.

Open and Close a Project

The HT-IDE3000 can work with only one project at a time, which is the opening project, at any time. If a project is to be worked upon, the project should first be opened, by using the Open command of the Project menu (Fig 4-1). Then, insert the project name directly or browse the directories and select a project name. Use the Close command to close the project.

Note: When opening a project, the current project is closed automatically. Within the development period, i.e. during editing, setting options and debugging etc., ensure that the project is in the open state. This is shown by the displaying of the project name of the opening project on the title of the HT-IDE3000 window. Otherwise, the results are unpredictable. The HT-IDE3000 will retain the opening project information if the system exits from the HT-IDE3000 without closing the opening project. This project will be opened automatically the next time the HT-IDE3000 is run.

Manage the Source Files of a Project

Use the Edit command to add or remove source program files from the opened project. The order, from top to bottom, of each source file in the list box, is the order of the input files to the Cross Linker. The Cross Linker processes the input files according to the order of these files in the box. Two buttons, namely [Move Up] and [Move Down], can be used to adjust the order of a source file in the project. Fig 4-3 is the dialog box of the Project menu's Edit command.

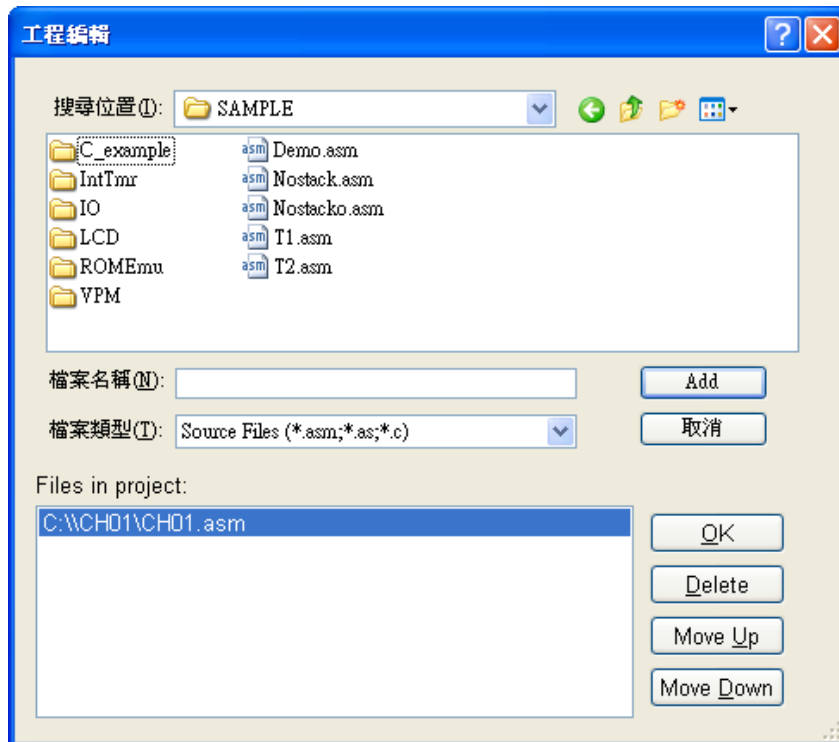


Fig 4-6

To Add a Source File to the Project

Choose a source file name from the file list box. Double-click the selected file name or choose the Add button to add the source files to the project. When the selected source file has been added, This file name is displayed on the list box of the Files in project.

To Delete a Source File from the Project

- Choose the file to be deleted from the project
- Click the Delete button

Note: Deleting the source files from the project does not actually delete the file but refers to the removal of the file information from the project.

To Move a Source File Up or Down

- Choose the file to be moved in the list box (Files in project), by moving the cursor to

this file and clicking the mouse button

- Click the [Move Up] button or the [Move Down] button

Build a Project's Task Files

Be sure that the following tasks have been completed before building a new project:

- The project has been opened
- The project options have all been set
- The project source files have been added
- The MCU options have been set (refer to the Tools menu chapter)

There are two commands related to the building of a project file, the Build command and the Rebuild all command.

The Project menu's Build command performs the following operations:

- Assemble or compile all the source files of the current project, by calling the Cross Assembler or C compiler depends on the file extension .asm or .C
- Link all the object files generated by the Cross Assembler or C compiler, and generate a task file and a debugging file.
- Load the task file into the HT-ICE if it is powered-on
- Display the source program of the execution entry point on the active window (the HTIDE3000 refers to the source files, the task file and the debugging file for emulation)

Note: The Build command may or may not execute the above tasks as the execution is dependent on the creation date/time of all corresponding files. The rules are:

- If the creation date/time of a source file is later than that of its object file, then the Cross Assembler or C compiler is called to assemble, compile this source file and to generate a new object file.
 - If one of the task s object files has a later creation date/time than that of the task file, then the Cross Linker is called to link all object files of this task and to generate a new task file.
-

The Build command downloads the task file into the HT-ICE automatically whether there is an action or not.

The Rebuild All command carries out the same task as the Build command. The

difference is that the Rebuild All command will execute the task immediately without first checking the creation date/time of the project files.

The result message of executing a Build or Rebuild All command are displayed on the Output window. If an error occurs in the processing procedure, the actions following it are skipped, and no task file is generated, and no download is performed.

To Build a Project Task File

- Click the Open command of the Project menu to open the project
- Click either the Build command of the Project menu or the Build button on the toolbar (Fig 4-1) to start building a project

To Rebuild a Project Task File

- Click the Open command of the Project menu to open the project
- Click either the Rebuild All command of the Project menu or the Rebuild all button on the toolbar (Fig 4-1) to start building a project once the project task has been built successfully, emulation and debugging of the application program can begin (refer to the HT-IDE3000 menu - Debug chapter).

Assemble/Compile

To verify the integrity of application programs, this command can be used to assemble or compile the source code and display the result message in the Output window.

To Assemble or Compile a Program

- Use the File menu to open the source program file to be assembled or compiled
- Either select the Assemble/Compile command of the Project menu or click the Assemble button on the toolbar to assemble/compile this program file

If the opened file has an .asm file extension name, the Cross Assembler will execute the assembly process. If the file has a .C extension then the Holtek C compiler will compile the program.

If no errors are detected, an object file with extension .OBJ is generated and stored in the directory which is specified in the Output Files Path (refer to Options menu, Directories command). If an error occurs and a corresponding message displayed on the Output

window, one of the following commands can be used to move the cursor to the error line:

- Double-click the left button of the mouse or
- Select the error message line on the Output window, and press the <Enter> key

Print Option Table Command

This command will print the current active option file to the specified printer. A printer may be selected where the options file is to be printed out. It is recommended to use a different printer port from the port which is connected to the HT-ICE.

If both the printer and the HT-ICE are using the same printer port, issuing this command will cause the loss of all debug information and corresponding data. After the printing job has finished, the user should proceed to the very beginning of the development procedure and use the Build command of the Project menu if further emulation/debugging of the application program is required.

Backup/Restore Project

The Backup Project command will use PROJECT_DATE_VERSION format to compress the current project, and also allow users to add some project description in [Description] editing box if necessary.

The Restroe Project command will restore the compressed project which selected in the backup list box currently.

Chapter 5

Menu - Debug

5

In the development process, the repeated modification and testing of source programs is an inevitable procedure. The HT-IDE3000 provides many tools not only to facilitate the debugging work, but also to reduce the development time. Included are functions such as single stepping, symbolic breakpoints, automatic single stepping, trace trigger conditions, etc.

After the application program has been successfully constructed, (refer to the chapter on Build a project's task files) the first execution line of the source program is displayed and highlighted in the active window (Fig 5-1). The HT-IDE3000 is now ready to accept and execute the debug commands.

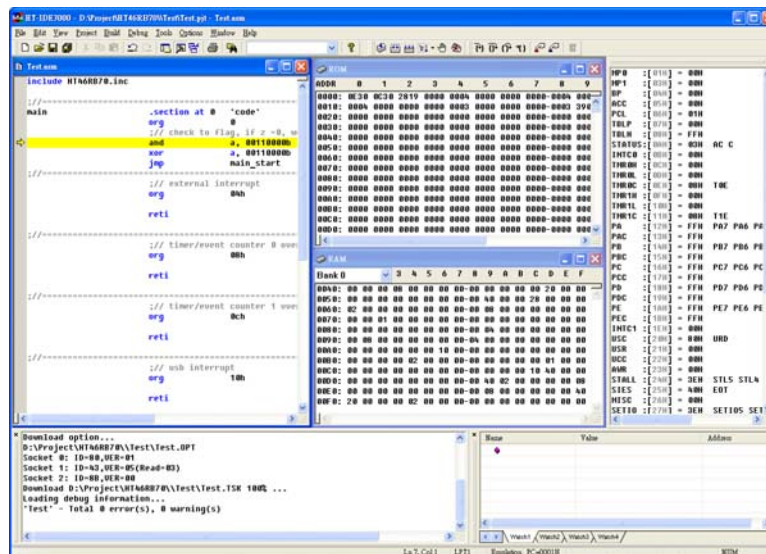


Fig 5-1

Reset the HT-IDE3000 System

There are 4 kinds of reset methods in the HT-IDE3000 system:

- Power-on reset (POR) by plugging in the power adapter or pressing the reset button on the HT-ICE
- Reset from the target board
- Software reset command in the HT-IDE3000 Debug menu (Fig 5-2)
- Software power-on reset command in the HT-IDE3000 Debug menu (Fig 5-2)

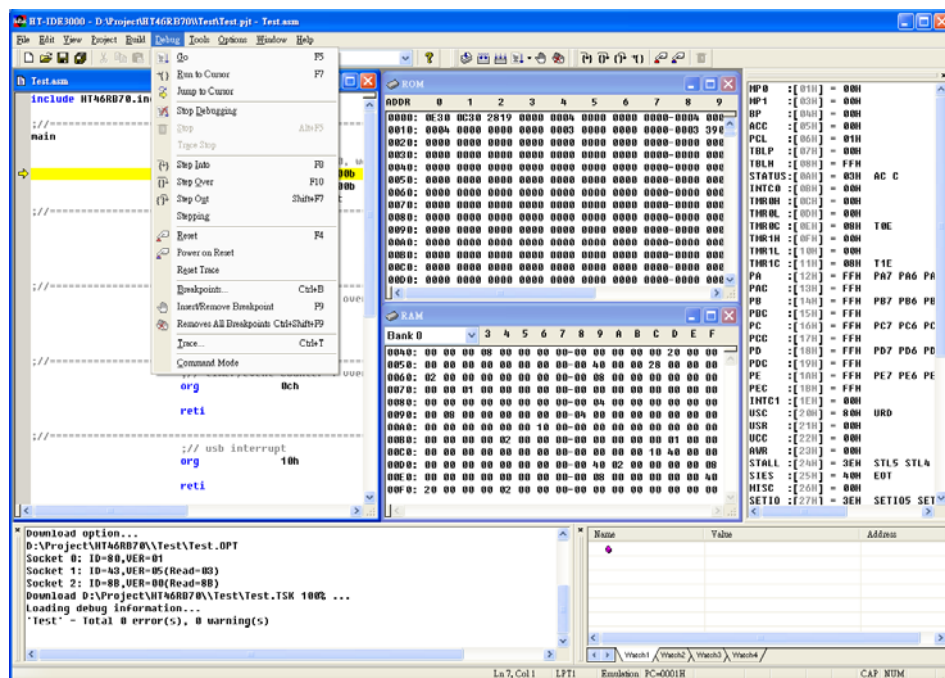


Fig 5-2



The effects of the above 4 types of reset are listed in table 5-1.

Reset Item	Power-On Reset	Target Board Reset	Software Reset Command	Software Power-On Reset Command
Clear Registers	(*)	(*)	(*)	(*)
Clear Options	Yes	No	No	No
Clear PD , TO	Yes	No	No	Yes
PC Value	(**)	0	0	0
Emulation Stop	(**)	No(***)	Yes	Yes
Check Stand-Alone	Yes	No	No	No

Table 5-1

Note: (*): Refer to the Data Book of the corresponding MCU for the effects of registers under the different resets.

(**): The PC value is 0 and the emulation stops.

(***): If the reset is from the target board, the MCU will start emulating the application after the reset is completed.

PC - Program Counter

PDF - Power Down Flag

TO - Time-out Flag

To Reset from the HT-IDE3000 Commands

- Either choose the Reset command from Debug menu or click the Reset button on the toolbar to execute a software reset
- Either choose the Power-on Reset command from the Debug menu or click the Power-on Reset button to execute a software power on reset

Emulation of Application Programs

After the application program has been successfully written and assembled, the Build or Rebuild command should be executed. If successful, the first executable line of the source program will be displayed and highlighted on the active window (Fig 5-1). At this point, emulation of the application program can begin by using the HT-IDE3000 debug

commands.

Note: During emulation of an application program, the corresponding project has to be open.

To Emulate the Application Program

- Choose the Go command from the Debug menu
 - or press the hot key F5
 - or press the Go button on the toolbar

Other windows can be activated during emulation. The HT-IDE3000 system will automatically stop the emulation if a break condition is met. Otherwise, it will continue emulating until the end of the application program. The Stop button on the toolbar is illuminated with a red color while the HT-ICE is in emulation. Pressing this button will stop the emulation process.

To Stop Emulating the Application Program

There are three methods to stop the emulation, shown as follows:

- Set the breakpoints before starting the emulation
- Choose the Stop command of the Debug menu or press the hot key Alt+F5
- Press the Stop button on the toolbar

To Run the Application Program to a Line

The emulation may be stopped at a specified line when debugging a program. The following methods provide this function. All instructions between the current point and the specified line will be executed except the conditional skips. Note however that the program may not stop at the specified line due to conditional jumps or other situations.

- Move the cursor to the stopped line (or highlight this line)
- Choose the Go to Cursor command of the Debug menu
 - or press the hot key F7
 - or press the Go to Cursor button on the toolbar

To Directly Jump to a Line of an Application Program

It is possible to jump directly to a line, if the result of executed instructions between the current point and the specified line are not important. This command will not change the contents of Data Memory, registers and status except for the Program Counter. The specified line is the next line to be executed.

- Move the cursor to the appropriate line or highlight this line
- Choose Jump to Cursor command of the Debug menu

Single Step

The execution results of some instructions in the above section may be viewed and checked. It is also possible to view the execution results one instruction at a time, i.e., in a step-by-step manner. The HT-IDE3000 provides two step modes, namely manual mode and automatic mode.

In the manual mode, the HT-IDE3000 executes exactly one step command each time the single- step command is executed. In the automatic mode, the HT-IDE3000 executes single step commands continuously until the emulation stop command is issued, using the Stop command of the Debug menu. In the automatic mode, all user specified breakpoints are discarded and the step rate can be set from FAST, 0.5, 1, 2, 3, 4 to 5 seconds. There are 3 step commands, namely Step Into, Step Over and Step Out.

- The Step Into command executes exactly one instruction at a time, however, it will enter the procedure and stop at the first instruction of the procedure when it encounters a CALL procedure instruction.
- The Step Over command executes exactly one instruction at a time, however upon encountering a CALL procedure, will stop at the next instruction after the CALL instruction instead of entering the procedure. All instructions of this procedure will have been executed and the register contents and status may have changed.
- The Step Out command is only used when inside a procedure. It executes all instructions between the current point and the RET instruction (including RET), and stops at the next instruction after the CALL instruction.

Note: The Step Out command should only be used when the current pointer is within a procedure or otherwise unpredictable results may happen.

The two step commands, Step Into and Step Over, in the automatic mode are set using the Debug sub-menu of the Options menu

- To start automatic single step mode

- Choose the Stepping command from the Debug menu also choose the stepping speed (the step command is set in the Debug command from the Options menu)
- To end automatic single step mode
Choose the Stop command from the Debug menu
 - To change automatic single step command for the automatic mode
 - Choose the Debug command from the Options menu
 - Choose the Step Into or the Step Over command in the Stepping command box
 - To start Step Into
Choose the Step Into command from the Debug menu
or press the hot key F8
or press the Step Into button on the toolbar
 - To start Step Over
Choose the Step Over command of the Debug menu
or press the hot key F10
or press the Step Over button on the toolbar
 - To start Step Out
Choose the Step Out command of the Debug menu
or press the hot key Shift+F7
or press the Step Out button on the toolbar

Breakpoints

The HT-IDE3000 provides a powerful breakpoint mechanism which accepts various forms of conditioning including program address, source line number and symbolic breakpoint, etc

Breakpoint Features

The following are the main features of the HT-IDE3000 breakpoint mechanism:

- Any breakpoint will be recorded in the breakpoints list box after it is set, however this breakpoint may not be immediately effective. It can be set to be effective later, as long as it is not deleted, i.e. still in the breakpoints list box.
- Breakpoints of address or data, in binary form with don't-care bits, are permitted.
- When an instruction is set to be an effective breakpoint, the ICE will stop at this instruction, but will not execute it, i.e. this instruction will become the next one to be

executed. Although an instruction is an effective breakpoint, the ICE may not stop at this instruction due to execution flow or conditional skips. If an effective breakpoint is in the Data Space (RAM), the instruction that matches this conditional breakpoint data will always be executed. The ICE will stop at the next instruction.

-
- Note:** 1. The HT-ICE can only have a maximum of 3 breakpoints active at the same time, while e-ICE can enjoy up to 65536 effective breakpoints.
2. It is acceptable to set breakpoints in Free Run mode for HT-ICE, however, e-ICE is not.
-

Description of Breakpoint Items

A breakpoint consists of the following descriptive items. It is not necessary to set all items, Fig 5-3:

- Space
The location of the breakpoint, either Program Code space or Data space.
- Location
The actual location of the breakpoint. The next paragraph will give the location format.
- Content
The data content of breakpoint. This item is effective only when the Space is assigned to the Data space. The Read and Write check box are used for executing conditions of the breakpoint.

Note: The breakpoint in data space is only available for HT-ICE but e-ICE.

→ Format of Description Items – Location

The allowed formats of Location items are:

- Absolute address (in code space or data space) with 4 format types, namely decimal, hexadecimal (suffix with “H” or “h” or prefix with 0x), binary and don’t-care bits. For example

20, 14h , 0x14, 00010100b , 10xx0011

represents decimal 20, hexadecimal 14h/0x14, binary 00010100b and don't-care bits 4 and 5 respectively.

Note: Don't-care bits must be in binary format.

- Line number with or without source file name, the format is:

[source_file_name!].line_number

where the source_file_name is a name of the optional source file. If there is no file name, the current active file is assumed. The exclamation point "!" is necessary only when a source file name is specified. The dot . must prefix the line number which is decimal.

Example:

C:\HIDE\USER\GE.ASM!.42

sets the breakpoint at the 42nd line of the file GE.ASM in directory \HIDE\USER of drive C.

Example:

.48

sets the breakpoint at the 48th line of the current active file.

- Program symbol with or without the source file name. The format is

[source_file_name!].symbol_name

All are the same as the line number location format except that the line_number is replaced with symbol_name. The following program symbols are acceptable:

- Label name
- Section name
- Procedure name
- Dynamic data symbols defined in data section

→ **Format of Description Items – Content**

The format of the content and external signals have five digital number options, similar to the format of Location absolute address. These four types of number are decimal, hexadecimal, binary and don't-care bits.

→ **Format of Breakpoints List Box**

The Breakpoints list box contains all the breakpoints that have been added, including effective breakpoints and non-effective breakpoints. The Add button should be used to add new breakpoints to the list box, and the Delete button to remove breakpoints from the

list box. The format of each breakpoint in the list box is as follows:

<status> {<space and read/write>, <location>, <data content>, <external signal>}

where <status> is effective status. “+” is effective (enabled) and “-” is non-effective (disabled). <space and read/write> is the space type and operating mode. “C” is the code space, “D/R” is the data space with read, “D/W” is the data space with write, “D/RW” is the data space with read and write.

<location>, <data content> and <external signal> have the same data format as the input form respectively.

How to Set Breakpoints

There are four methods to set/enable a breakpoint, one is by using the Breakpoint command from the Debug menu, the others are by using the Toggle Breakpoint button on the toolbar, double clicking on the gray bar in the edit window, or pressing key F9. The rules of the breakpoint mechanism are as follows:

- If the breakpoint to be set is not in the Breakpoints list box (Fig 5-3), then the descriptive items must be designated first, then added to the Breakpoints list box.
- As long as the breakpoint exists in the list box, it can be made effective by Enabling the breakpoint if it fails to be initially effective.
- Press the OK button for confirmation. Otherwise, all changes here will not be effective.
- When using the Toggle Breakpoint button on the toolbar, the cursor should first be moved to the breakpoint line, and then the Toggle Breakpoint button pressed. If an effective breakpoint is to be changed to a non-effective breakpoint, this can be achieved by merely pressing the Toggle breakpoint button.

→ To Add a Breakpoint

- Choose the Breakpoint command from the Debug menu (or press the hot key Ctrl+B)
A breakpoint dialog box is displayed (Fig 5-3)
- Designate the descriptive items of the breakpoint
Set Space, Location items
Set Content item and Read/Write check box if Space is the data space
- Press the Add button to add this breakpoint to the Breakpoints list box.
- Press the OK button to confirm

Note: If the total count of the effective breakpoints is less than 3, the newly added one will take effect automatically after it has been added.

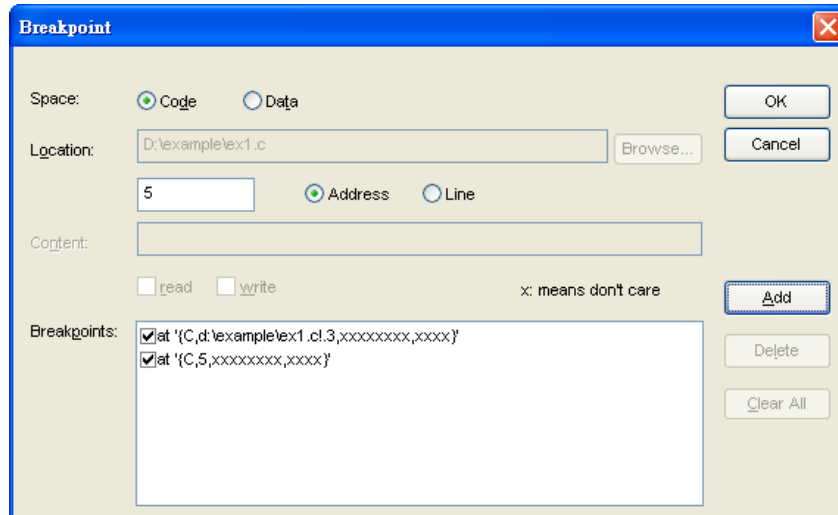


Fig 5-3

→ **To Delete a Breakpoint**

- Choose the Breakpoint command from the Debug menu or press the hot key Ctrl+B
A breakpoint dialog box is displayed (Fig 5-3)
- Choose or highlight the breakpoint to be deleted from the Breakpoints list box
- Press the Delete button to delete this breakpoint from the Breakpoints list box
- Press the OK button to confirm

→ **To Delete all Breakpoints**

- Choose the Breakpoint command from the Debug menu or press the hot key Ctrl+B
A breakpoint dialog box is displayed (Fig 5-3)
- Choose the Clear All button to delete all breakpoints from the Breakpoints list box
- Press the OK button to confirm
- You can also click the Clear All Breakpoint button on the toolbar to accomplish this task.

→ **To Enable (Disable) a Breakpoint**

- Choose the Breakpoint command from the Debug menu or press the hot key Ctrl+B
A breakpoint dialog box is displayed (Fig 5-3)
- Choose the disabled (enabled) breakpoint from the Breakpoints list box
- Press the Enable (Disable) button, to enable or disable this breakpoint
- Press the OK button to confirm

Trace the Application Program

The HT-IDE3000 provides a powerful trace mechanism which records the execution processes and all relative information when the HT-IDE3000 is emulating the application program. The trace mechanism provides qualifiers to filter specified instructions and trigger conditions in order to stop the trace recording. It also provides a method to record a specified count of the trace records before or after a trigger point.

Note: When the HT-IDE3000 starts emulating (refer to the section on Emulation of the Application Programs), the trace mechanism will begin to record the executing instructions and relative information automatically, but not vice versa.

Note: Only the HT-ICE support the trace mechanism function.

Initiating the Trace Mechanism

The basic requirement for initializing the trace mechanism is to set the Trace Mode with or without Qualify. The Trace Mode defines the trace scope of the application program and Qualify defines the filter conditions of the trace recording.

The available Trace Modes are:

- Normal
Sets the trace scope to all application programs and is the default mode.
- Trace Main
Sets the trace scope to all application programs except the interrupt service routine programs.
- Trace INT
Sets the trace scope to all interrupt service routine programs.

According to Qualify, the trace mechanism decides which instructions and what corresponding information should be recorded in the trace buffer during the emulation process. The rule is that an instruction will be recorded if its information and status satisfy one of the enabled qualifiers. The format of Qualify is the same as that of the breakpoint. If all program steps are required to be recorded, then No Qualify is needed (do not set the Qualify). The default is No Qualify.

In contrast to the Trace Mode and Qualify, which specify the conditions of trace recording, both the Trigger Mode and Forward Rate specify the conditions to stop the trace recording.

The Trigger Mode specifies the kind of trigger point, and is a standard used to determine the location of the stop trace point. The Forward Rate specifies the trace scope between the trigger point and the stop trace point.

The available Trigger Modes are:

- No Trigger
No stopping of the trace recording condition. This is the default case.
- Trigger at Condition A
The trigger point is at condition A.
- Trigger at Condition B
The trigger point is at condition B.
- Trigger at Condition A or B
The trigger point is at either condition A or condition B.
- Trigger at Condition B after A
The trigger point is at condition B after condition A has occurred.
- Trigger when meeting condition A for k times
The trigger point is when condition A has occurred k times.
- Trigger at Condition B after meeting A for k times
The trigger point is at condition B after condition A has occurred for k times.

Condition A and Condition B specify the trigger conditions. The format of condition A or B is the same as that of the breakpoint.

The Loop Count specifies the number of occurrences of the specified condition A. It is used only when the Trigger Mode is from one of the last two modes in the above list.

The Forward Rate specifies the approximate rate of the trace recording information between the trigger point and stop trace point in the whole trace buffer. The trigger point

divides the trace buffer into two parts, before and after trigger point. The forward rate is used to limit the trace recording scope after the trigger point. The percentage is adjustable between 0 and 100%.

Note: It is not necessary for the trace recording scope to be equal to the forward rate. If a breakpoint is met before reaching the trace recording scope or a trace stop command (refer to: Stopping the trace mechanism) is issued, the trace recording will be stopped.

A Qualify list box records and displays all qualifiers used by the Trace Mode. Up to 20 qualifiers can be added into the list box and up to 6 qualifiers can be effective. A Qualifier can be disabled or deleted from the list box. The format of each qualifier in the Qualify list box has the same format as the breakpoint in the Breakpoints list box (refer to the section on Breakpoints, Format of breakpoints list box)

Stopping the Trace Mechanism

There are 3 methods to stop the trace recording mechanism :

- Set the trigger point (Trigger Mode) and Forward Rate as shown above
- Set breakpoints to stop the the emulation and the trace recording.
- Issue a Trace Stop command from the Debug menu (Fig 5-2) to stop the trace recording.

Fig 5-4 lists all the requirements to use the trace mechanism. This is the result of the Trace command from the Debug menu.

Trace Start/Stop Setup

→ **To Set the Trace Mode**

- Choose the Trace command from the Debug Menu
A Trace dialog box is displayed as in Fig 5-4.
- Choose a trace mode from the Trace Mode pull-down list box
- Press the OK button

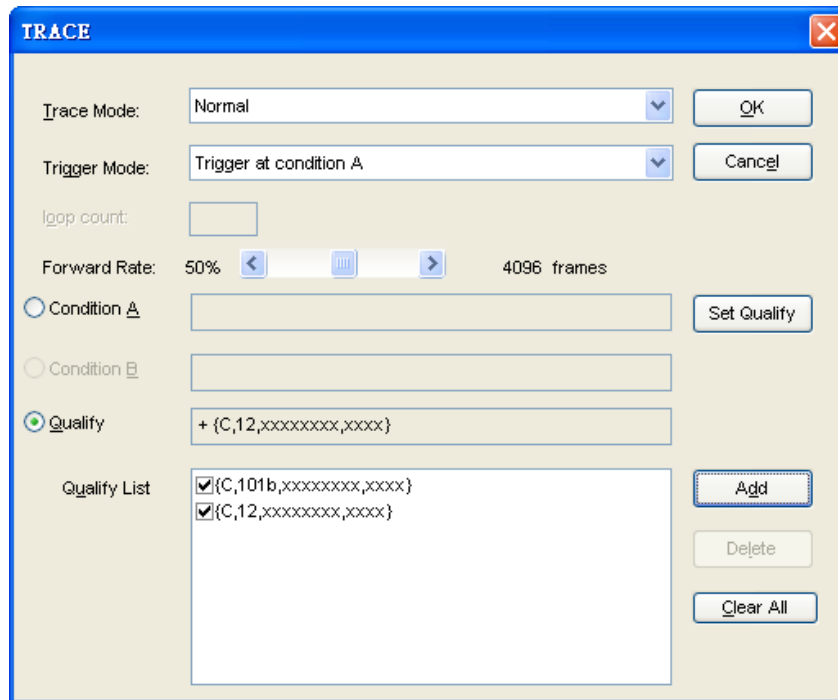


Fig 5-4

- **To Set the Trigger Mode**
 - Choose the Trace command from the Debug Menu
A Trace dialog box is displayed as in Fig 5-4.
 - Choose a trigger mode from the Trigger Mode pull-down list box
 - press the OK button

- **To Change the Forward Rate**
 - Choose the Trace command from the Debug Menu
A Trace dialog box is displayed as in Fig 5-4
 - Use the Forward Rate scroll bar to specify the desired rate
 - Press the OK button

- **To Setup the Condition A/Condition B**
 - Choose the Trace command of the Debug Menu
A Trace dialog box is displayed as Fig 5-4.
 - Press Condition A/Condition B radio button
 - Press the Set Condition button
A Set Qualify dialog box is displayed as in Fig 5-5.
 - Enter the conditional information

- Press the OK button to close the Set Condition dialog box
- Press the OK button to close the Trace dialog box

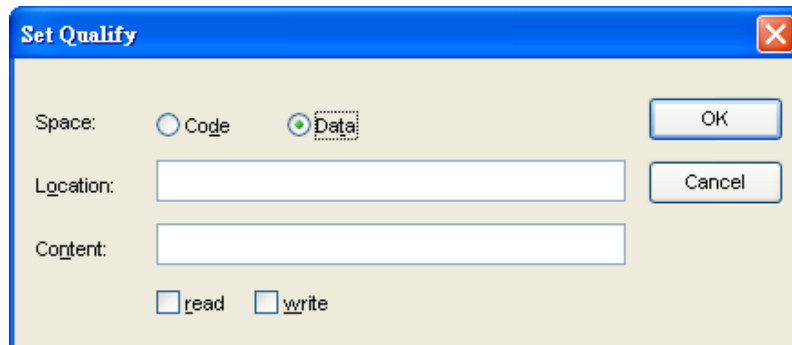


Fig 5-5

→ **To Add a Trace Qualify Condition**

- Choose the Trace command from the Debug Menu
A Trace dialog box is displayed as in Fig 5-4.
- Press the Qualify radio button
- Press the Set Qualify button
A Set Qualify dialog box is displayed as in Fig 5-5.
- Enter the qualifier information
- Press the OK button to close the Set Qualify dialog box
- Press the Add button to add the qualifiers into the Qualify list box below
- Press the OK button to close the Trace dialog box

→ **To Delete a Trace Qualify Condition**

- Choose the Trace command from the Debug Menu
A Trace dialog box is displayed as in Fig 5-4.
- Choose the qualify line to be deleted from the Qualify list box
- Press the Delete button
- Press the OK button to confirm

→ **To Delete All Qualify Conditions**

- Choose the Trace command from the Debug Menu
A Trace dialog box is displayed as in Fig 5-4.
- Press the Clear All button
- Press the OK button to confirm

Note: If there is no qualifier, all instructions are qualified by default.

→ To Enable (Disable) a Trace Qualify Condition

- Choose the Trace command from the Debug Menu
A Trace dialog box is displayed as in Fig 5-4
- Choose the disabled (enabled) qualifier line to be enabled (disabled) from the Qualify list box
- Press the Enable (disable) button
- Press the OK button to confirm

Note: At most, 6 trace qualifications can be enabled at the same time. The e-ICE is limited to the Normal Mode, the trace range is for the whole application program.

Trace Record Format

Once the trace qualify and trigger conditions have been setup, those instructions which satisfy the qualify conditions will be recorded in the trace buffer. The Trace List command of the Window menu provides the functions to view and check the trace record information, used for debugging the program. The trace record fields may not all be displayed on the screen except for the sequence number. These fields are dependent upon the settings in the Debug sub-menu from the Options menu. The text enclosed by the parentheses are the headings shown in the Trace List command of the Window menu. Fig 5-6 and Fig 5-7 illustrate the contents of the trace list under the different debug options.

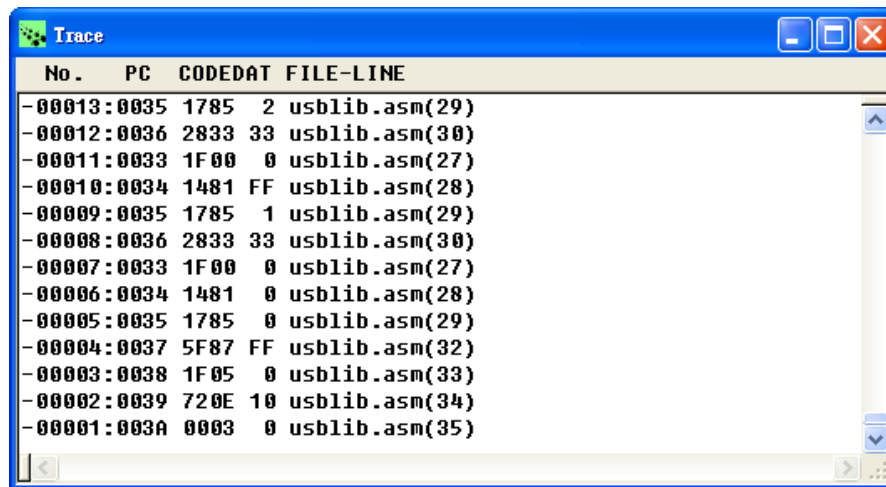


Fig 5-6

- Sequence number (No)

For any of the trigger modes, the sequence number of a trigger point is +0. The trace records before and after the trigger point are numbered using negative and positive line numbers respectively. If all the fields of the Trace Record Fields (in the Debug Option of Option menu) are selected, the result is as shown in Fig 5-7. If No trigger mode is selected or the trigger point has not yet occurred, the sequence number starts from -00001 and decreases 1 sequentially for the trace records (Fig 5-6).

- Program count (PC)
The program count of the instruction in this trace record.
- Machine code (CODE)
The machine code of this instruction.
- Disassembled instruction (INSTRUCTION)
The disassembled mnemonic instruction is disassembled using an HT-IDE3000 utility.
- Execution data (DAT)
The data content to be executed (read/write).
- Source file name with a line number (FILE-LINE)
The source file name and the line number of this instruction.
- Source file (SOURCE)
The source line statement (including symbols).

All the above fields are optional except the sequence number which is always displayed.

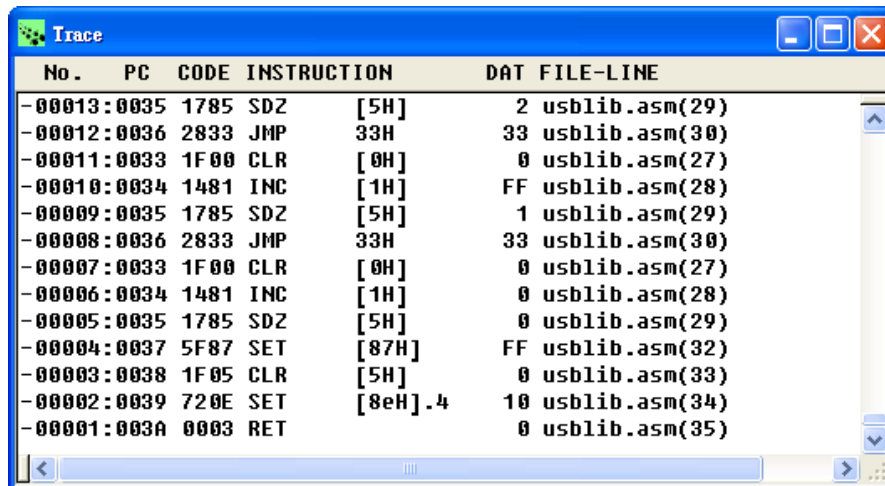


Fig 5-7

Note: To set the trace record fields use the Debug command of the Options menu.
To view the trace record fields use Trace List command of the Window menu.

→ **Clear the Trace Buffer**

The trace buffer can be cleared by issuing the Reset Trace command. Hereafter, the trace information will be saved from the beginning of the trace buffer. Note that both the Reset command and the Power-On Reset command also clear the trace buffer.

Debugger Command Mode

In addition to the windows based debugging mode, the HT-IDE3000 provides an alternative debugging mode, named the Command Mode. Under this mode, the user, in addition to obtaining the same functions as the menu-driven windows based debugging mode, also has access to additional debugging functions. These added functions include the ability to save the debugging history into a log file in order to execute these debugging commands automatically again as well as the ability to execute the previous debugging command without rewriting the command.

Enter/Quit the Command Mode

→ **Enter to Command Mode**

From the Debug Menu of the HT-IDE3000 select “Command Mode” command. When the command mode has been entered a new screen will appear where commands can be entered after the “HT8>” prompt on the second line. (Fig 5-8)

→ **Command Mode Window**

- The Command Mode Title bar shows the name of the present project file.
- Any command can be entered after the “HT8>” prompt on the command line.
- When the command is entered the full command syntax will be displayed on the bottom status bar.
- After the command has been entered at the “HT8> xxxx” prompt, the next line will display the result of the command execution. (Fig 5-9)
Another “HT8>” prompt will then be displayed where another command can be entered.

→ **Quit from the Command Mode**

To quit from the Command Mode the normal windows exit method can be used or a Q[quit] command can be entered at the command prompt.

Functions Supported by the Command Mode

The following table shows the complete list of debugging statements supported by the Command Mode

Command	Function Description	Command Syntax
!	Execute a previous command	! dd
;	Comment	;
BP	Breakpoint Commands	BP{-C -D -E -L}[list]*] → list=11 12...
BP	Breakpoint Set	BP S[,RW],Location [,Data][,Ext Sig]
DB	Dump Program Memory	DB[bank.address[,range]]
DR	Dump Data Memory	DR[bank]address[,range]
FA	Fill string	FA {bank.address symbol}list. → list=11 12...
FB	Fill bytes	FB {bank.address symbol}list → list=11 12...
GO	Free run or run to the specified address	GO [address]
JP	Jump to specified address directly	JP address
H	Help	H
HIS	History of commands	HIS
LF	Load and execute a log file	LF [-V] [LogFileName]
LP	Load project	LP ProjectName
Q	Quit	Q
R	Reset	R
POR	Power on reset	POR
S	Single Step (Into/Over/Out)	S [-I -V -O] (default option: -I
TR	Trace list	TR [-L][length]
W	Open/Write/Close a Log file	W {-S -C}[LogFileName]

In the debugging command syntax, if large brackets exist, this indicates that a parameter must be inserted otherwise an error will occur. Parameters are separated by a | symbol.

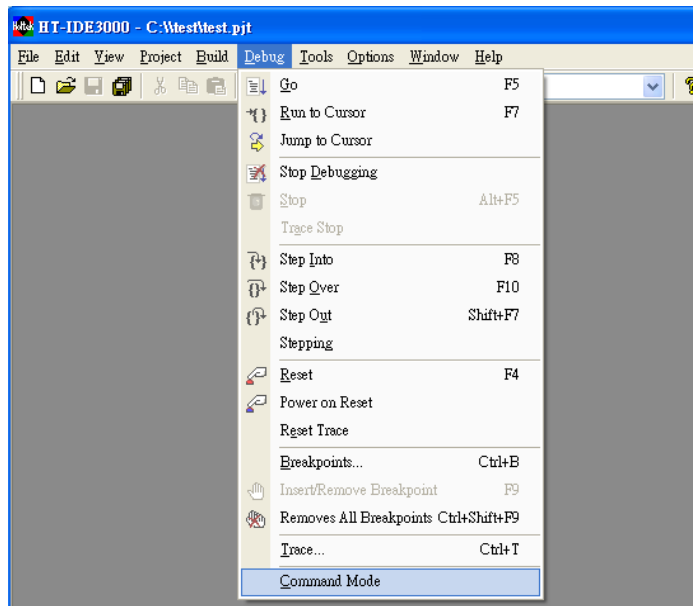


Fig 5-8

→ **Breakpoint Commands**

There are two breakpoint commands, their command syntax and function is as follows :

- BP - Breakpoint Clear/Enable/Disable/List

Syntax : BP [-C|-D|-E|-L] [list]*

Parameter -C is the clear breakpoint parameter. This will delete the indicated breakpoint or clear all the breakpoints shown in the Breakpoint Box. Within the list there can be from 1~20 numbers

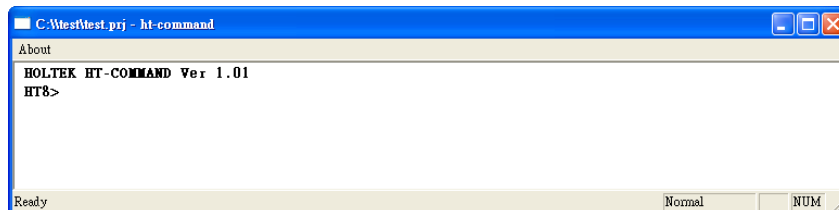


Fig 5-9

which represent the breakpoints already setup. This means that more than one can be selected. For example, the three numbers 1 3 8 each separated by a space, indicates that the 1st, 3rd and 8th, breakpoints will be cleared. This has the same operation as the Delete function within the Debug/Breakpoint window. The star symbol * means that all the breakpoints already setup will be cleared. It has the same operation as the Clear All function within the Debug/Breakpoint window.

Parameter -D will change all the indicated breakpoints to non-active, however the breakpoints will still remain shown in the Breakpoint Box. This command is the same as the Disable function within the Debug/Breakpoint window. The star * has the same operation as that described above.

Parameter -E will change all the indicated breakpoints to active. , This command is the same as the Enable function within the Debug/Breakpoint window. The star * has the same operation as that described above.

Parameter -L will display all the presently setup breakpoints in the window, the format is consistent with the contents of the Debug/Breakpoint window, where the first column shows the breakpoint number. The user can refer to this breakpoint number to setup the required numbers in the BP -C, BP -D, BP -E statement.

-
- Note:** 1. BP-L this parameter does not require list |
 2. The HT-IDE3000 can only have a maximum of 3 breakpoints active at the same time.
-

If no C, D, E or L parameters are given then the Breakpoint command will be of the following type:

- BP – Breakpoint Set

Syntax : BP S[,RW] ,Location [,Data][,Ext Sig]

The parameter within the brackets is optional however under certain conditions it must be specified.

S denotes a Space, where a choice can be made between C or D. The letter C indicates that the breakpoint is set in Program Code Memory, while D indicates that the breakpoint is set in the Data Memory (RAM)

If D is chosen to replace S then the read/write option [,RW] must also be specified.

The user can choose from R or W or RW. This is because if the breakpoints are set in the Data Memory then the choice exists for the breakpoint to be activated on either a read, a write or both a read and write. If C is chosen to replace S, which indicates program code, then it is not necessary to setup RW.

The “Location” parameter sets the position of the breakpoint, its format is:

[SourceFileName!].LineNumber or [SourceFileName!].SymbolName

If no SourceFileName is specified then the already opened source file will be taken as the default.

If D is chosen to replace S then the “Data” parameter must be setup. The breakpoint is setup at the specified location in the Data Memory and will initiate a break when a

read or write with the specified data occurs.

Ext Sig is a parameter that can be chosen, for its use consult the HT-IDE3000 User's Guide.

→ **Comment Command**

- Syntax : ; comment string

This command is provided to give an explanation to the Log file. Any characters found after the ; will have no functional effect.

→ **Dump Command**

- Syntax : DB bank.address ,range

DB range

DB

This command will display in the window, the contents of the specified program memory area. This area is specified by indicating the address, as well as the range and bank. The data is in hex format. If the address is specified but the bank number is not specified then the bank number will be taken as that of the current bank. If neither address nor bank number is specified the bank number will be taken as that of the current bank number and the address will be taken as that of the present Program Counter. If the range is not specified then the range value will be taken as 16 words. The range is not allowed to exceed one bank (2000h). An example of this statement would be 1.0f00 which would indicate that the bank number is 1 and the address value is 0f00h.

- Syntax : DR bank.address ,range

DR address

This command will display in the window, the contents of the specified area of Data Memory. This data area is specified by its address, range and bank. The data is displayed in hex format. If the range is not specified then it will be set to 16 bytes. The range is not allowed to exceed one bank (100h) and the bank address is expressed in hex format.

→ **Fill Command**

This command changes the contents of the Data Memory

- Syntax : FB {bank.address | symbol} ,list

Will write the bytes specified in the list into a Data Memory area at the specified bank number and at the specified start address or symbol. Either a bank.address or symbol name can be used. Also the list can be more than one byte, however at least one

blank must be used as a delimiter. All values are specified in hex format. The list range cannot cross over a bank boundary.

- Syntax : FA {bank.address | symbol} ,string

FA has the same function as FB except that the data is supplied in ASCII the user can chose one of the following symbol formats:

.var
filename!.var
path\ filename!.var

Note: If path contains spaces then the name must be included in quotation marks otherwise an error condition will occur.

- Example : FA "d:\tmp\test cmd\test1.asm!.count" , "test1"

→ **Go/Jump Commands**

- Syntax : GO [address]

If an address is specified the program will free run until the specified address is encountered. If the address is not specified the program will run to the end or until an active breakpoint is encountered.

- Syntax : JP address

Will force a direct jump to the specified address. Note that an address must be specified.

→ **Help Command**

- Syntax : H

This command will list in the window all of the debugging commands, their syntax and description.

→ **History Command**

- Syntax : HIS

This command will display in the window the last 20 commands, not including the HIS command, that were executed. At the same time the first column will display the command sequence numbers in succession.

- Syntax : !dd

dd is the displayed command sequence number in the above mentioned HIS command. This command will execute the previously executed command again. By writing the sequence number and adding a "!" the same command can be executed again reducing the need to re-input commands and parameters. If no command sequence number is indicated the last command will be executed.

→ Load Commands

- Syntax : LF [-V] [LogFileName]

This command will load and execute all the Debugging Commands in the Log File, specified by the LogFileName

If no LogFileName is specified, then the same name as the current Project File name will be taken as the filename.

Parameter -V indicates that the command line and the execution result should be displayed in the window.

If LF has no -V option, then the result record will be placed in a logfile of the same name with a .res file extension name.

Log file is created using the W command. The contents can be modified by using the File and Edit function within the HT-IDE3000.

However these contents must contain the correct Debugging Commands otherwise an error condition will occur, the execution will stop and return to the prompt sign

-
- Note:** 1.If spaces are included in the LogFileName then the name must be included within quotation marks otherwise an error condition will occur.
2.The logfile cannot contain the LF, W or Q commands.
-

→ Quit Command

- Syntax : Q

This command will end the Command Mode and return to the present window.

-
- Note:** 1.This command has no effect in the Command Log file.
2.After quitting from the command mode all the files opened by “LF” and “W -S” will be closed and the execution of commands will stop.
-

→ Reset Commands

- Syntax : R

The function of this command is the same as the Debug/Reset command

- Syntax : POR

The function of this command is the same as the Debug/Power-On Reset command

→ Step Commands

There are 3 kinds of Single Step commands; which after execution will display the contents of the PC, STATUS and ACC

- Syntax : S {-I | -O | -V}

Single Step Command.

- I is Step Into, which has the same function as Debug/Step Into
 - V is Step Over, which has the same function as Debug/Step Over
 - O is Step Out, which has the same function as Debug/Step Out
- If no option has been setup the default condition will be “S -V”

→ **Trace Command**

- Syntax : TR [-L] [length]

The trace command will display the contents of the trace buffer in the window. Parameter -L indicates that all records will be displayed, which include Sequence number, Program count, Machine code, Disassembled instructions, Execution data, External signal, source file name with line number and source file.

If the -L parameter is not supplied, then the default condition will only display Sequence number, Program count, Machine code, Disassembled instructions and source file name with line number. The parameter “length” indicates the length of the displayed trace. The trace display will begin from sequence number 0 and trace back with the specified length. The length can also specify the length to trace forward. To do this the forward rate must first be setup in the system. The default length value is 5.

The Trace mode, qualify conditions and forward rate etc. parameters are directly setup within the HT-IDE3000 window, the command mode does not support these functions.

→ **Write Command**

- Syntax : W [-S | -C] [LogFileName]

This command will write the debugging commands and its corresponding results into the Log File. The Log File will terminate whenever a W -C or Q command is encountered or if the command mode is terminated.

-S will create a Log File in which all following commands and results will be written

-C will close the previously created Log File, no further commands will be written into the Log File

If the indicated Log File is already saved, then the system will require confirmation before overwriting and continuing with the next step. It is not necessary to add a file extension name.

If the Log File name does not exist, then the file name will take the same name as the project with an added .CMD file extension name.

-
- Note:** 1.If spaces are included in the LogFileName then the name must be included within quotation marks otherwise an error condition will occur.
- 2.After executing the W -S command the LF or W -S command cannot be executed.
-

Log File Format

The Log File is a text file that can be modified by any text editor including the editor contained within the HT-IDE3000. This editor can be accessed by selecting Edit from the main menu. Its format is that every Debugging command will occupy one line.

command: `W -S LogFileName` will clear the contents of the Log File, and after write the new commands and results.

If the command string, has been created by the “`W -S`”command then note that prompt signs will also be written into the Log File. However, the next time it is read by the debugger command these previously written prompt signs will be ignored automatically. For the case where the command strings are generated using an editor, note that it is not necessary to enter any prompt signs into the Log File.

If the Log File has been created by the “`W -S`”command then before each command execution result a “`;`” will be automatically inserted making the execution result into an annotated note. In this way when the next upload is executed only the command string will be executed, the result string will be ignored.

HT-COMMAND Error Messages

Error Message	Description
Invalid Command	The command just entered is illegal
Can not find HT-IDE	The present environment is not the HTIDE3000
Syntax error	The input syntax is incorrect
No project for debug	No project file has been opened in the HT-IDE3000
ROM bank Out of range	The Program Memory dump has exceeded its range
RAM bank Out of range	The Data Memory dump has exceeded its range
Can not run xxx command in emulation mode	The xxx command cannot be executed
Can not run xxx command in load file mode	The xxx command cannot be executed
Can not run xxx command in write file mode	The xxx command cannot be executed
Unterminated string	The character string definition requires balanced quotes
No Command in history buffer	History buffer empty
Open xxx log file error	Cannot open the log file
Close xxx log file error	Cannot close the log file
Read xxx log file error	Cannot read the log file
Write xxx log file error	Cannot write to the log file
Not in emulation status	Before executing this command first enter emulation mode
Sources have been modified , please rebuild	The original source file has been modified requiring the files to be rebuilt
Stop by user	User has stopped execution
Get PC failed	Reading the value of the Program Counter has failed
Stack overflow	The stack has exceeded its capacity
No debug info	The setup breakpoints have no debug information

Error Message	Description
Cannot find the symbol	The indicated symbol cannot be found
Cannot find the register	The indicated register cannot be found

Chapter 6

Menu - Window

6

The HT-IDE3000 provides various kinds of windows which assist the user to emulate or simulate application programs. These windows (as shown in Fig 6-1) include program Data Memory (RAM), program code memory (ROM), Trace List, Register, Watch, Stack, Program, Output, etc.

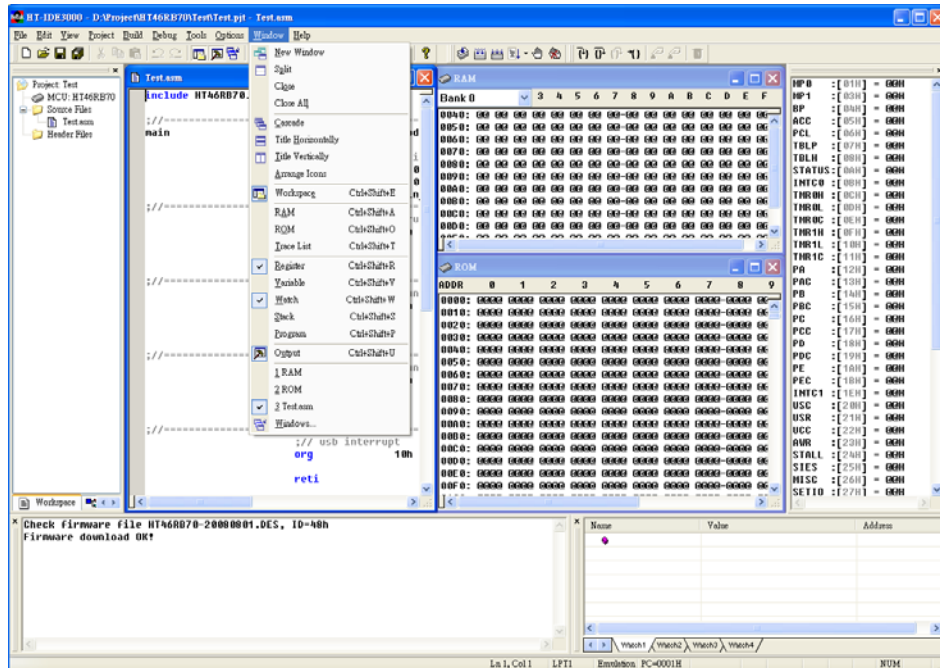


Fig 6-1

Window Menu Commands

- **Workspace**
 The Workspace window lists out all of the source files in the project. As shown in Fig. 6-2, here chosen source files can be quickly selected. Files can be added or removed here.

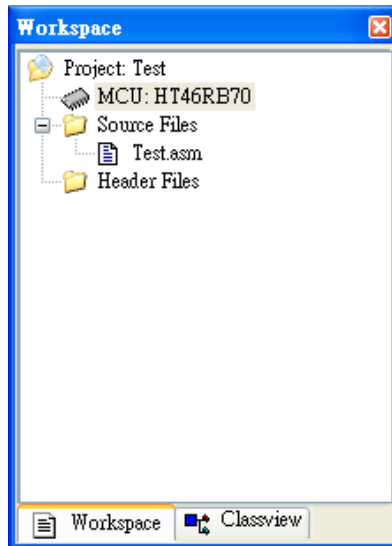


Fig 6-2

- **RAM**
 The RAM window display the contents of the program Data Memory space as shown in Fig 6-3 The address spaces of the registers are not included in the RAM window because they are displayed in the register window. The contents of the RAM window can be modified directly for debugging purpose. The address displayed vertically is the base address while the horizontal single digit address is the offset. All the digits are displayed in hexadecimal format.

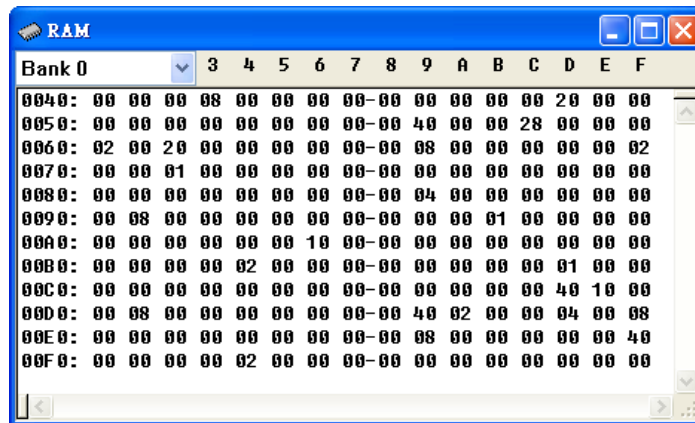


Fig 6-3

- ROM
The ROM window displays the contents of the program code memory space as shown in Fig 6-4. The ROM address range is from 0 to last address where the last address depends upon the MCU selected in the project. The horizontal and vertical scrollbars can be used to view any address in the ROM window. The contents in ROM window are displayed in hexadecimal format and cannot be modified.

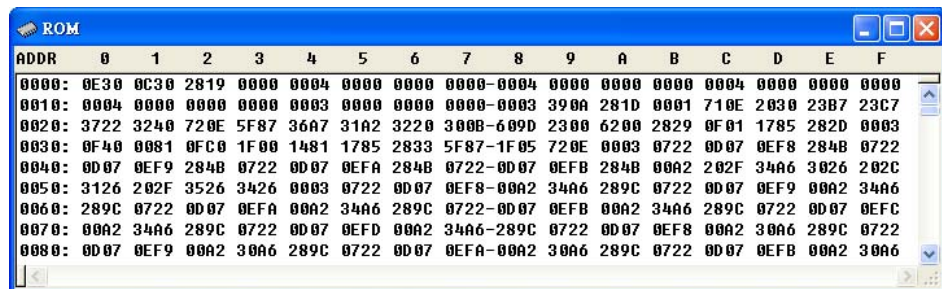


Fig 6-4

- Trace List
The Trace List window displays the trace record information as shown in Fig 6-5. The contents of the trace record can be defined in the Debug command in the Options menu. Double click the trace record in the Trace List window will activate the source file window and the cursor will stop at the corresponding line.

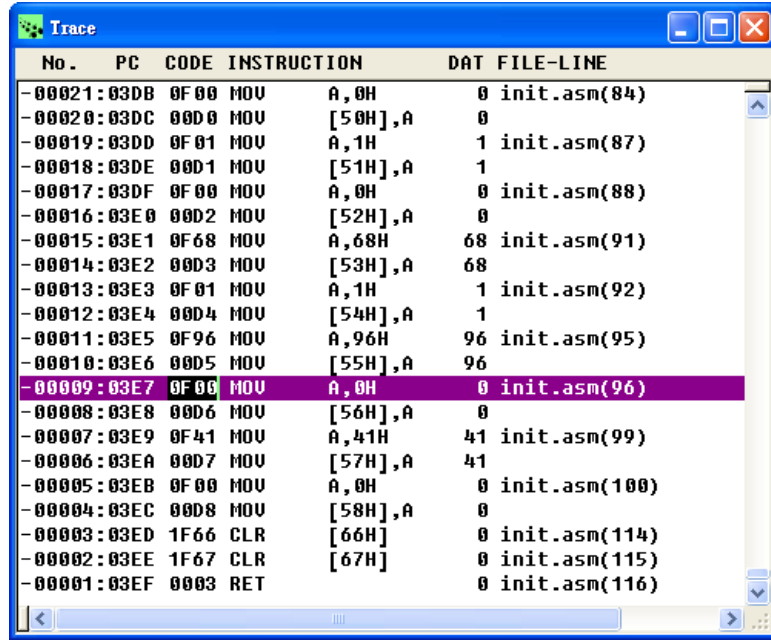


Fig 6-5

- Register

The Register window displays all the registers defined in the MCU selected in the project. Fig 6-6 shows an example of the Register window of HT48C70-1. The contents of the Register window can be modified for debugging. Note that the Register window is dockable.

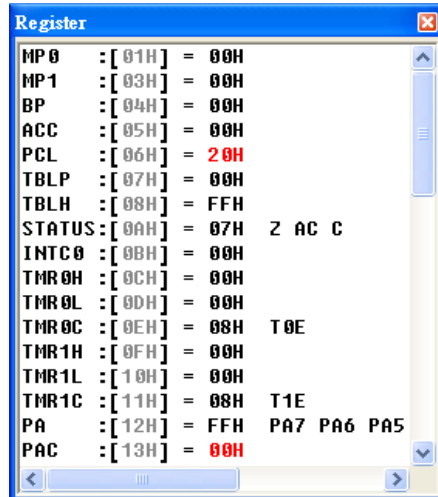


Fig 6-6

- Watch

The Watch window displays the memory addresses and contents of the specified symbols defined in the data sections, i.e., in the RAM space. The format of the symbol is:

[source_file_name!].symbol_name

The contents of the registers can also be displayed by first typing a period then typing the symbol name or register name and pressing the Enter key. The memory address and contents of the specified symbol or register will be displayed to the right of the symbol as shown in the following format:

:[address]=data contents

Note that both address and data are displayed in hexadecimal format as shown in Fig 6-7. The symbol and their corresponding data will be saved by the HT-IDE3000 and displayed the next time the Watch window is opened. The symbols can be deleted from Watch window by pressing the delete key. Note that the Watch window is dockable.

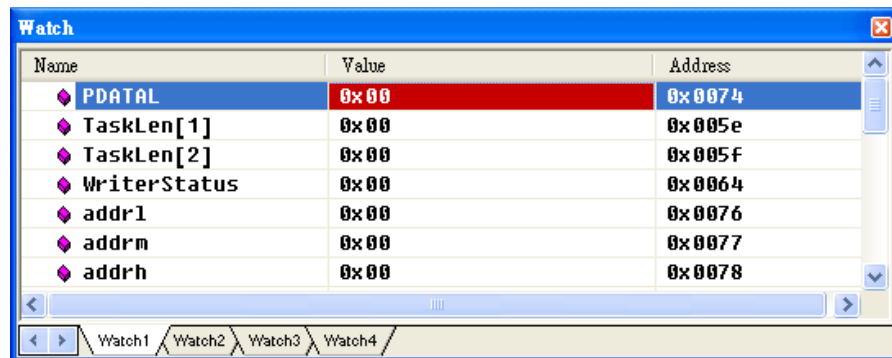


Fig 6-7

- Stack

The Stack window displays the contents of the stack buffer for the MCU selected in the current project. The maximum stack level is dependent upon the MCU selected. Fig 6-8 shows an example of the Stack window. The growth of the stack is numbered from 0. The number is increased by 1 for a push operation (CALL instruction or interrupt) and decreased by 1 for a pop operation (RET or RETI instructions). The top stack line is highlighted. E.g. The 01: shown in Fig 6-8 is the top stack line. While executing a RET or RETI instruction, the program line number specified in the top stack line (134 in this example) will be used as the next instruction line to be executed.

Also, the line above the top stack line (00: in this example) will be used as the new top stack line. If there is no stack line anymore, no line in the Stack window will be highlighted. The format of the stack line is:

Stack_level: program_counter source_file_name(line_number)

where the stack_level is the level number of the stack, program_counter is the hexadecimal return address of the calling procedure or the program address of the interrupted instruction, source_file_name is the complete name of the source file containing the calling or interrupted instruction, and line_number is the decimal line number of the instruction after the call instruction or interrupted instruction in the source file.

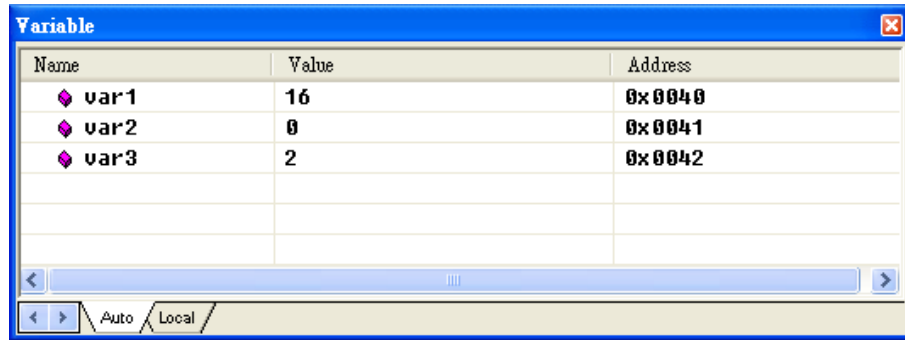


Fig 6-8

- Variable

This window can view and modify variable values, including two tabs:

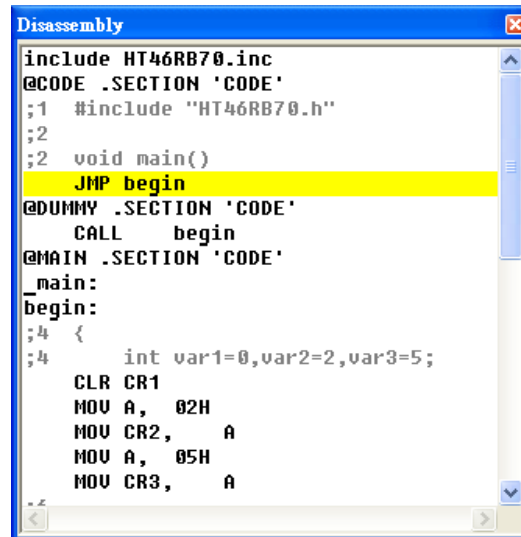
- Auto tab, this tab can observe and set both local variables and global variables related to the current function.
- Local tab, this tab can observe and set local variables related to the current function.



Name	Value	Address
var1	16	0x0040
var2	0	0x0041
var3	2	0x0042

Fig 6-9

- Program
The Program window displays the program code memory or ROM in disassembly format. The address range is from 0 to last address where the last address depends upon the MCU selected in the project.
- Disassembly
The Disassembly Window shows mixed high-level source code and its associated assembler code. Each instruction is marked with code coverage indicators that show execution status.



```

include HT46RB70.inc
@CODE .SECTION 'CODE'
;1 #include "HT46RB70.h"
;2
;2 void main()
JMP begin
@DUMMY .SECTION 'CODE'
CALL begin
@MAIN .SECTION 'CODE'
_main:
begin:
;4 {
;4 int var1=0,var2=2,var3=5;
CLR CR1
MOV A, 02H
MOV CR2, A
MOV A, 05H
MOV CR3, A
    
```

Fig 6-10

- Output
The Output window shows the system messages from the HT-IDE3000 when the Build/Rebuild All commands are executing. By double clicking on the error message

line, the window containing the source file will be displayed and the corresponding line containing the error highlighted.

Chapter 7

Menu - Simulation

7

The HT-IDE3000 provides a simulation mechanism for debugging application programs. The HT-IDE3000 simulator provides the same functions as the HT-ICE, but does not require the actual presence of the HT-ICE to function. In the HT-IDE3000, all the debugging and window functions for the HT-ICE are valid for the simulator. In addition, the simulator provides an interface for the input and output ports. Although the simulator provides many functions, some hardware characteristics of the MCU cannot be simulated. It is therefore recommended that emulation is carried out on the application program using the HT-ICE before manufacture of the masked IC.

Some MCU series support emulation mode only and some support simulation mode.

Note: Some MCU series support simulation mode, e.g. HT48R10/30/50/70-1

Start the Simulation

Upon entering the HT-IDE3000, two situations may occur. The first is when a project has already been opened, and the second is when no project has been opened. In the first case, the working mode of the HT-IDE3000 depends upon the working mode of this project. In the latter case, the working mode will be in simulation. Even if the working mode of a project is in emulation, it can be changed by the user to be in simulation. In addition, the working mode of the HT-IDE3000 will be in simulation when the following situations occur.

- No connection between the HT-ICE and the host machine or when the connection

fails.

- The HT-ICE is powered off.

The Debug command in the Option menu provides the function to set the working mode of the HT-IDE3000. Fig 7-1 displays the contents of the Debug command.

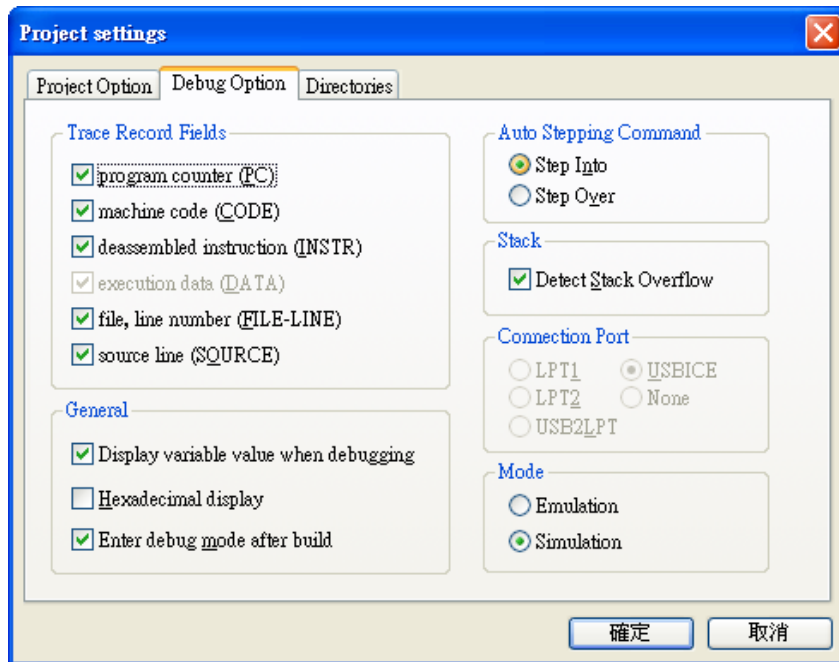


Fig 7-1

In addition to MCU simulator, Holtek provides a Virtual Peripheral Manager (VPM) which enables the user to directly drive and monitor the simulation of inputs and outputs on PC.

Part III gives more details on the VPM.

Chapter 8

MCU Programming

8

Introduction

The MCU Writer is a writer to program both OTP (One-Time Programmable) and MTP (Multi-Times Programmable) MCU devices. All of the Holtek OTP and MTP devices can be programmed using this writer. The advantages of this writer are in its small size and ease of installation and simple operation.

The newest versions of the HT-ICE hardware emulator include an integrated writer for convenient user operation – see Fig. 8-1.

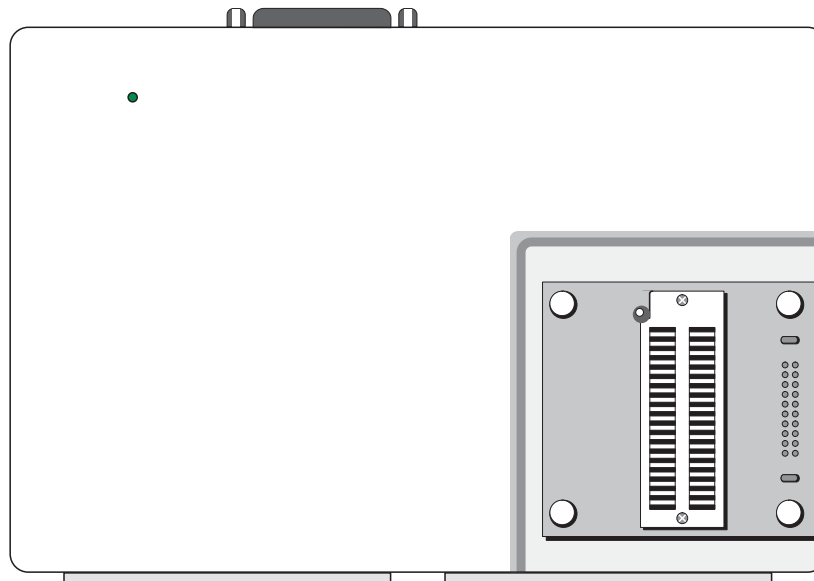


Fig 8-1

Installation

Since the MCU writer is built-in on the HT-ICE box, after the completion of HT-ICE installation, the MCU programming function is ready to be used within the HT-IDE3000 software with no further installation procedure needed. Refer to Chapter 1 — Overview and Installation.

Adapter Card

The HT-ICE emulator is shipped with a 40-pin TEXTTOOL Adapter Card. If the device package format doesn't match with this Adapter Card, the user will need to change the Adapter Card. Refer to other Holtek Technical Documents or visit our website for further information on selecting Adapter Cards.

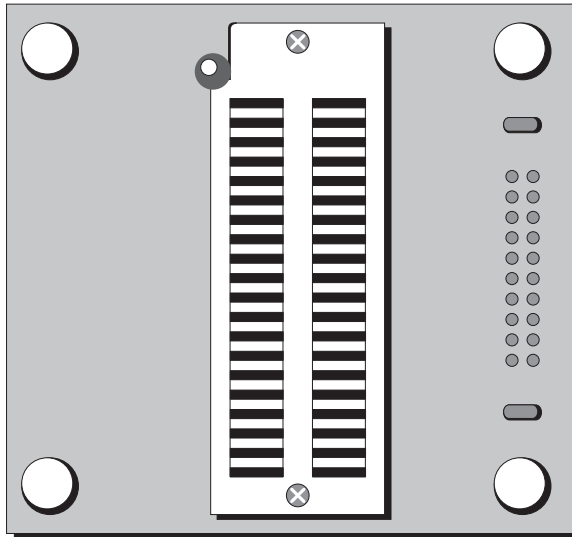


Fig 8-2

Programming an MCU Device with the EverPro K1000

Run the EverPro K1000 Software

Run the EverPro K1000 software under the Holtek Development System icon in the main Windows programs menu as shown in the Fig 8-3 below:

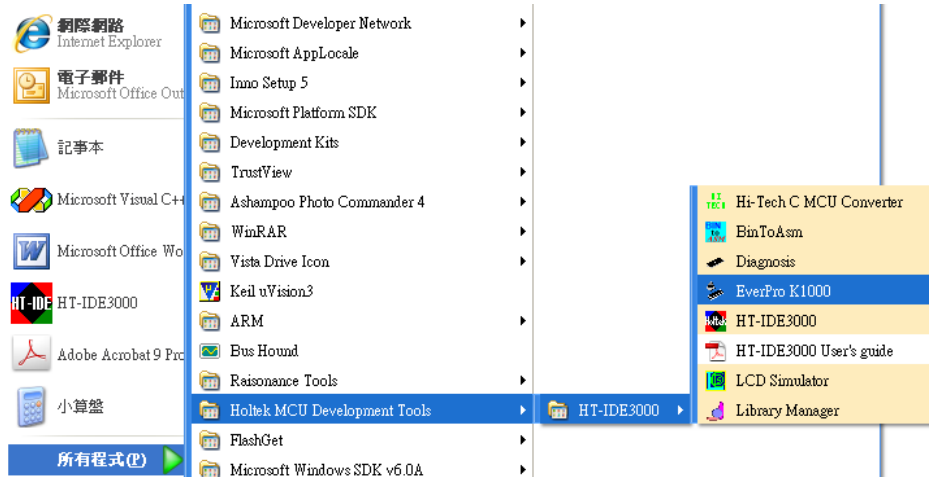


Fig 8-3

Or launch the EverPro K1000 from the HT-IDE3000 as show int the Fig8-4 below:

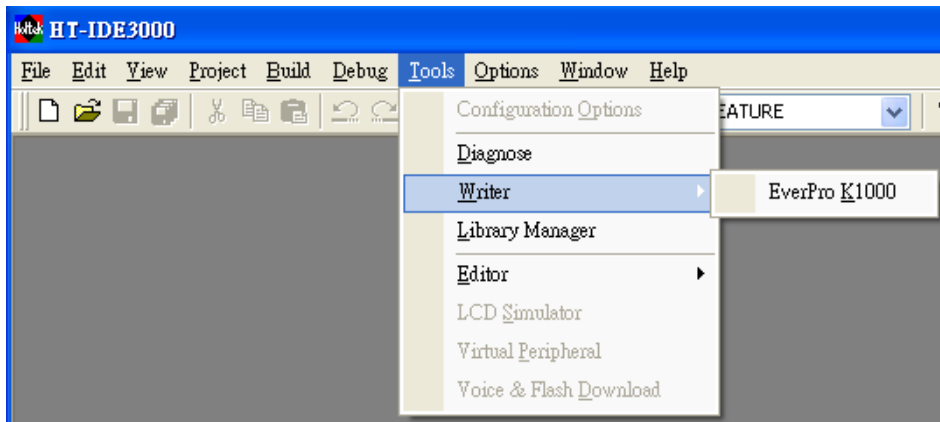


Fig 8-4

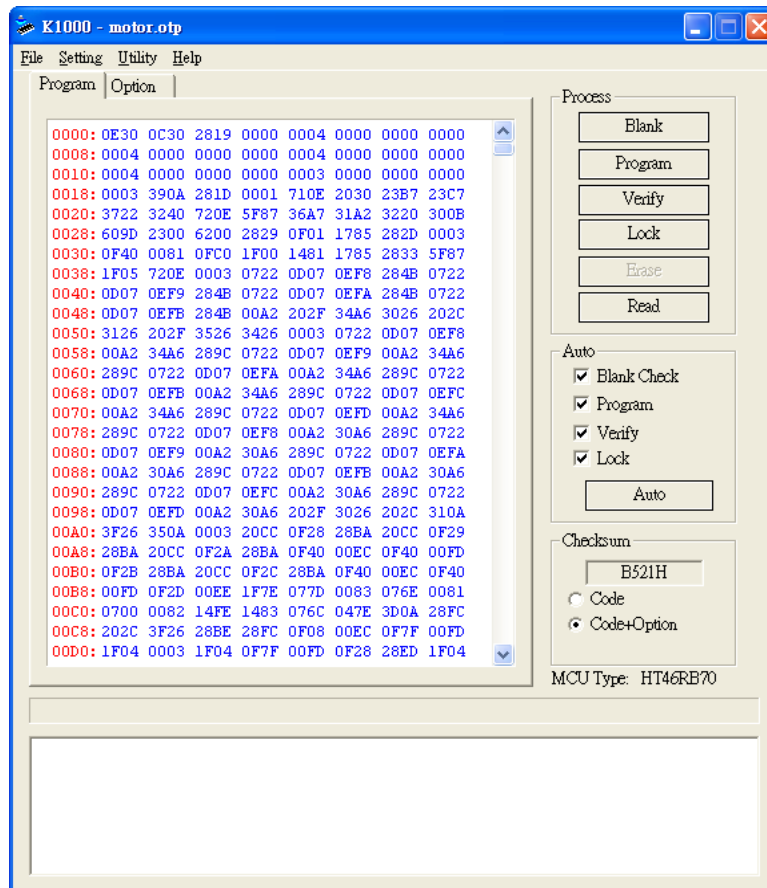


Fig 8-5

EverPro K1000 Programming Functions

Fig 8-5 shows the internal functions of the EverPro K1000. The 7 buttons shown at the right hand side of this window each represent an instruction, the function is explained below:

- Blank Check
Check that the presently loaded MCU device has not previously been written to. The results of this check will be displayed on the EverPro K1000 display. If the device is not empty, the memory area that has been written to will also be shown on the display.
- Program
The function is to place the program data in the PC ram memory into the OTP device
- Verify
The contents of the presently loaded MCU device will be read and checked that it is

the same as the data loaded into the PC ram memory, the results of which will be displayed on the EverPro K1000 Writer display.

- **Read**

This instruction will read out the contents of the MCU device presently loaded into the MCU writer and store them in the PC ram memory. This instruction will also cause the file checksum to be displayed underneath the “Auto” button. If required, this data can also be stored in a file with the .OTP or .MTP file suffix.

- **Erase**

This command can erase the Program and Data area of the MTP device.

- **Lock**

This instruction will implement the protect function in the MCU device preventing the contents of this IC from being read. After programming an MCU device, this instruction can then be used to protect the contents.

- **Auto**

This instruction will execute in order the four instructions Blank Check, Program, Verify and Lock. If any of the instructions do not execute correctly, the process will be halted and the following instruction not executed.

EverPro K1000 Additional Functions

→ **File/Open...**

This opens a file with an .OTP or .MTP suffix, which will load the program contents into the PC ram memory. This data will be accessed when programming the relevant MCU device. After selecting “Open”, the file dialogue box will be displayed from which the correct folder and file name can be chosen. The file content will be displayed in the message window after being opened, and the checksum of the opened file will be shown underneath the “Auto” button.

→ **File/Save**

Save updates the current file by overwriting the last save of the file.

→ **File/SaveAs...**

Save As lets you save the current file in OTP or MTP format under a different file name.

→ **Setting/Programming Setting...**

These commands allow the setup of detailed programming operations and other setup functions as shown in Fig. 8-6.

- **MCU Type**
Select the MCU type. If the OTP device has not stored its part number then the user can manually select the MCU type.
- **Programming Field**
Specify programming field, including Program, Option, Data and Voice four areas.
- **Read Field**
Specify read field, including Program, Option, Data and Voice four areas.
- **Verify Field**
Specify verify field, including Program, Option, Data and Voice four areas.
- **Blank Field**
Specify blank field, including Program, Option, Data and Voice four areas.
- **Erase Field**
Specify erase field, including Program, Option, and Data three areas (Only support MTP series microcontroller).
- **Lock Field**
Specify lock field, including Program and Data areas.
- **Programming Mode**
Either Parallel Mode or Serial Mode can be selected.
- **Check ID**
Checks the device ID before each operation.

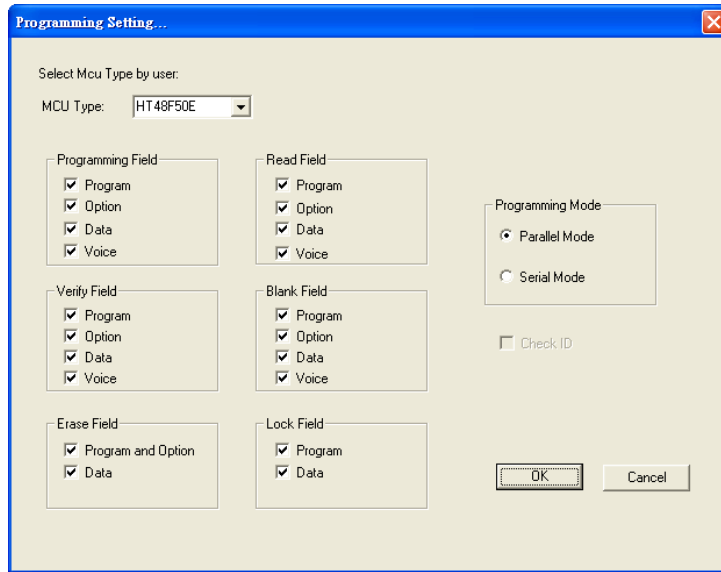


Fig 8-6

→ **Setting/Language**

English, Traditional Chinese or Simplified Chinese can be selected here.

→ **Utility/Read Option**

This command can read the Option contents – can also be used for locked devices.

→ **Utility/Partial Lock...**

This command can lock indicated and partial memory sections as shown in Fig. 8-6. After selecting the required area select OK to continue operation.

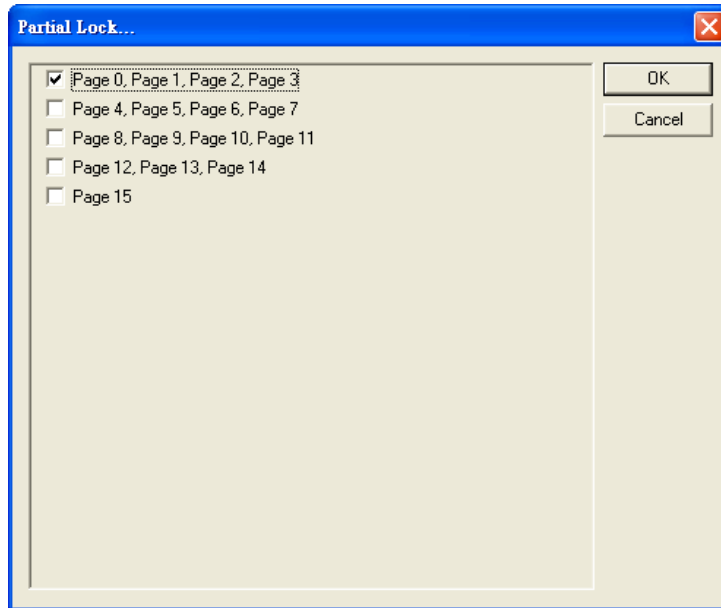


Fig 8-7

→ Utility/Partial Program...

This command will program indicated and partial memory sections as shown in Fig. 8-8. The red area indicates the selected area which will be programmed.

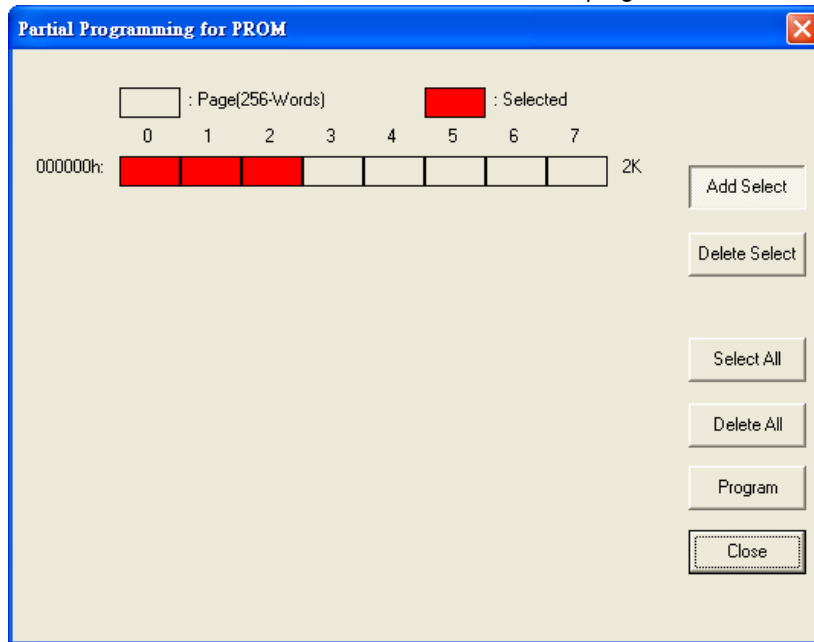


Fig 8-8

→ **Utility/Page Erase...**

Can erase indicated pages of memory as shown in Fig. 8-9. Only supports MTP type devices.

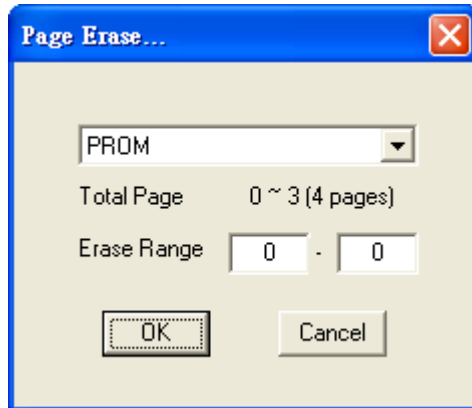


Fig 8-9

→ **Utility/Print Option Table**

Print option table.

Part II

Development Language and Tools

Chapter 9

Assembly Language and Cross Assembler

9

Assembly-Language programs are written as source files. They can be assembled into object files by the Holtek Cross Assembler. Object files are combined by the Cross Linker to generate a task file.

A source program is made up of statements and look up tables, giving directions to the Cross Assembler at assembly time or to the processor at run time. Statements are constituted by mnemonics (operations), operands and comments.

Notational Conventions

The following list describes the notations used by this document.

Example of convention	Description of convention
[<i>optional items</i>]	Syntax elements that are enclosed by a pair of brackets are optional. For example, the syntax of the command line is as follows: HASM [<i>options</i>] <i>filename</i> [:]
	In the above command line, options and semicolon; are both optional, but filename is required, except for the following case:

	Brackets in the instruction operands. In this case, the brackets refer to memory address.
<i>{choice1 choice2}</i>	Braces and vertical bars stand for a choice between two or more items. Braces enclose the choices whereas vertical bars separate the choices. Only one item can be chosen.
	Three dots following an item signify that more items with the same form may be entered. For example, the directive PUBLIC has the following form:
Repeating elements...	PUBLIC <i>name1</i> [, <i>name2</i> [...]]
	In the above form, the three dots following <i>name2</i> indicate that many names can be entered as long as each is preceded by a comma.

Statement Syntax

The construction of each statement is as follows:

[name][operation][operands][,comment]

- All fields are optional.
- Each field (except the comment field) must be separated from other fields by at least one space or one tab character.
- Fields are not case-sensitive, i.e., lower-case characters are changed to upper-case characters before processing.

Name

Statements can be assigned labels to enable easy access by other statements. A name consists of the following characters:

A~Z a~z 0~9 ? - @

with the following restrictions :

- 0~9 cannot be the first character of a name

- ? cannot stand alone as a name
- Only the first 31 characters are recognized

Operation

The operation defines the statement action of which two types exist, directives and instructions. Directives give directions to the Cross Assembler, specifying the manner in which the Cross Assembler is to generate the object code at assembly time. Instructions, on the other hand, give directions to the processor. They are translated to object code at assembly time, the object code in turn controls the behavior of the processor at run time.

Operand

Operands define the data used by directives and instructions. They can be made up of symbols, constants, expressions and registers.

Comment

Comments are the descriptions of codes. They are used for documentation only and are ignored by the Cross Assembler. Any text following a semicolon is considered a comment.

Assembly Directives

Directives give direction to the Cross Assembler, specifying the manner in which the Cross Assembler generates object code at assembly time. Directives can be further classified according to their behavior as described below.

Conditional Assembly Directives

The conditional block has the following form:

```
IF
  statements
[ELSE
  statements]
ENDIF
```

→ **Syntax**
IF *expression*
IFE *expression*

- Description
The directives **IF** and **IFE** test the expression following them.
The **IF** directive grants assembly if the value of the expression is true, i.e. non-zero.
The **IFE** directive grants assembly if the value of the expression is false, i.e. zero.

- Example


```
IF      debugcase
        ACC1      equ 5
        extern   username: byte
ENDIF
```

In this example, the value of the variable `ACC1` is set to 5 and the `username` is declared as an external variable if the symbol `debugcase` is evaluated as true, i.e. nonzero.

→ **Syntax**

IFDEF *name*
IFNDEF *name*

- Description
The directives **IFDEF** and **IFNDEF** test whether or not the given name has been defined. The **IFDEF** directive grants assembly only if the name is a label, a variable or a symbol. The **IFNDEF** directive grants assembly only if the name has not yet been defined. The conditional assembly directives support a nesting structure, with a maximum nesting level of 7.

- Example


```
IFDEF      buf_flag
        buffer DB 20 dup (?)
ENDIF
```

In this example, the buffer is allocated only if the `buf_flag` has been previously defined.

File Control Directives

→ **Syntax**

INCLUDE *file-name*
or
INCLUDE *file-name*

- Description
This directive inserts source codes from the source file given by `file-name` into the current source file during assembly. Cross Assembler supports at most 7 nesting levels.
- Example


```
INCLUDE    macro.def
```

In this example, the Cross Assembler inserts the source codes from the file `macro.def`

into the current source file.

→ **Syntax**

PAGE *size*

- Description

This directive specifies the number of the lines in a page of the program listing file. The page size must be within the range from 10 to 255, the default page size is 60.

- Example

```
PAGE 57
```

This example sets the maximum page size of the listing file to 57 lines.

→ **Syntax**

.LIST

.NOLIST

- Description

The directives **.LIST** and **.NOLIST** decide whether or not the source program lines are to be copied to the program listing file. **.NOLIST** suppresses copying of subsequent source lines to the program listing file. **.LIST** restores the copying of subsequent source lines to the program listing file. The default is **.LIST**.

- Example

```
.NOLIST
mov a, 1
mov b1, a
.LIST
```

In this example, the two instructions in the block enclosed by **.NOLIST** and **.LIST** are suppressed from copying to the source listing file.

→ **Syntax**

.LISTMACRO

.NOLISTMACRO

- Description

The directive **.LISTMACRO** causes the Cross Assembler to list all the source statements, including comments, in a macro. The directive **.NOLISTMACRO** suppresses the listing of all macro expansions. The default is **.NOLISTMACRO**.

→ **Syntax**

.LISTINCLUDE

.NOLISTINCLUDE

- Description

The directive **.LISTINCLUDE** inserts the contents of all included files into the program listing. The directive **.NOLISTINCLUDE** suppresses the addition of included files. The default is **.NOLISTINCLUDE**.

→ **Syntax**

MESSAGE *'text-string'*

- Description

The directive **MESSAGE** directs the Cross Assembler to display the text-string on the screen. The characters in the *text-string* must be enclosed by a pair of single quotation marks.

→ **Syntax**

ERRMESSAGE *'error-string'*

- Description

The directive **ERRMESSAGE** directs the Cross Assembler to issue an error. The characters in the *error-string* must be enclosed by a pair of single quotation marks.

Program Directives

→ **Syntax (comment)**

; text

- Description

A comment consists of characters preceded by a semicolon (;) and terminated by an embedded carriage-return/line-feed.

→ **Syntax**

name **.SECTION** [*align*] [*combine*] *'class'*

- Description

The **.SECTION** directive marks the beginning of a program section. A program section is a collection of instructions and/or data whose addresses are relative to the section beginning with the name which defines that section. The name of a section can be unique or be the same as the name given to other sections in the program. Sections with the same complete names are treated as the same section. The optional align type defines the alignment of the given section. It can be one of the following:

BYTE uses any byte address (the default align type)

WORD uses any word address

PARA uses a paragraph address

PAGE uses a page address

For the CODE section, the byte address is in a single instruction unit. **BYTE** aligns the section at any instruction address, **WORD** aligns the section at any even instruction address, **PARA** aligns the section at any instruction address which is a multiple of 16, and **PAGE** aligns the section at any instruction address with a multiple of 256.

For **DATA** sections, the byte address is in one byte units (8 bits/byte). **BYTE** aligns the section at any byte address, **WORD** aligns the section at any even address, **PARA** aligns the section at any address which is a multiple of 16, and **PAGE** aligns the

section at any address which is a multiple of 256. The optional combine type defines the way of combining sections having the same complete name (section and class name). It can be any one of the following:

- COMMON
Creates overlapping sections by placing the start of all sections with the same complete name at the same address. The length of the resulting area is the length of the longest section.
- AT *address*
Causes all label and variable addresses defined in a section to be relative to the given address. The address can be any valid expression except a forward reference. It is an absolute address in a specified ROM/RAM bank and must be within the ROM/RAM range.

If no *combine* type is given, the section is combinative, i.e., this section can be concatenated with all sections having the same complete name to form a single, contiguous section. The *class* type defines the sections that are to be loaded in the contiguous memory. Sections with the same class name are loaded into the memory one after another. The class name **CODE** is used for sections stored in ROM, and the class name **DATA** is used for sections stored in RAM. The complete name of a section consists of a section name and a class name. The named section includes all codes and data below (after) it until the next section is defined.

→ **Syntax**

ROMBANK *banknum section-name [,section-name,...]*

- Description
This directive declares which sections are allocated to the specified ROM bank. The banknum specifies the ROM bank, ranging from 0 to the maximum bank number of the destination MCU. The *section-name* is the name of the section defined previously in the program. More than one section can be declared in a bank as long as the total size of the sections does not exceed the bank size of 8K words. If this directive is not declared, bank 0 is assumed and all CODE sections defined in this program will be in bank 0. If a CODE section is not declared in any ROM bank, then bank 0 is assumed.

→ **Syntax**

RAMBANK *banknum section-name [,section-name,...]*

- Description
This directive is similar to **ROMBANK** except that it specifies the RAM bank, the size of RAM bank is 256 bytes.

→ **Syntax**

END

- Description

This directive marks the end of a program. Adding this directive to any included file should be avoided.

→ **Syntax**

ORG *expression*

- Description

This directive sets the location counter to expression. The subsequent code and data offsets begin at the new offset specified by expression. The code or data offset is relative to the beginning of the section where the directive ORG is defined. The attribute of a section determines the actual value of offset, absolute or relative.

- Example

```
ORG 8
mov A, 1
```

In this example, the statement `mov A, 1` begins at location 8 in the current section.

→ **Syntax**

PUBLIC *name1* [, *name2* [, ...]]

EXTERN *name1.type* [, *name2.type* [, ...]]

- Description

The **PUBLIC** directive marks the variable or label specified by a name that is available to other modules in the program. The **EXTERN** directive, on the other hand, declares an external variable, label or symbol of the specified name and type. The type can be one of the three types: **BYTE**, **BIT** (these three types are for data variables), and **NEAR** (a label type and used by `call` or `jmp`).

- Example

```
PUBLIC start, setflag
EXTERN tmpbuf:byte
CODE .SECTION 'CODE'
start:
    mov    a, 55h
    call  setflag
    ...
setflag proc
    mov    tmpbuf, a
    ret
setflag endp
end
```

In this example, both the label `start` and the procedure `setflag` are declared as public variables. Programs in other sources may refer to these variables. The variable `tmpbuf` is also declared as external. There should be a source file defining a byte that is named `tmpbuf` and is declared as a public variable.

→ **Syntax**

name **PROC**

name **ENDP**

- Description

The **PROC** and **ENDP** directives mark a block of code which can be called or jumped to from other modules. The **PROC** creates a label name which stands for the address of the first instruction of a procedure. The Cross Assembler will set the value of the label to the current value of the location counter.

- Example

```
toggle PROC
mov      tmpbuf, a
mov      a, 1
xorm     a, flag
mov      a, tmpbuf
ret
toggle ENDP
```

→ **Syntax**

[*label*.] **DC** *expression1* [, *expression2* [...]]

- Description

The **DC** directive stores the value of *expression1*, *expression2* etc. in consecutive memory locations. This directive is used for the CODE section only. The bit size of the result value is dependent on the ROM size of the MCU. The Cross Assembler will clear any redundant bits; *expression1* has to be a value or a label. This directive may also be employed to setup the table in the code section.

- Example

```
table: DC      0128H, 025CH
```

In this example, the Cross Assembler reserves two units of ROM space and also stores 0128H and 025CH into these two ROM units.

Data Definition Directives

An assembly language program consists of one or more statements and comments. A statement or comment is a composition of characters, numbers, and names. The assembly language supports integer numbers. An integer number is a collection of binary, octal, decimal, or hexadecimal digits along with an optional radix. If no radix is given, the Cross Assembler uses the default radix (decimal). The table lists the digits that can be used with each radix.

Radix	Type	Digits
B	Binary	01
O	Octal	01234567
D	Decimal	0123456789

H Hexadecimal 0123456789ABCDEF

→ **Syntax**

```
[name] DB value1 [,value2 [,...]]
[name] DBIT
[name] DB repeated-count DUP(?)
```

- Description

These directives reserve the number of bytes specified by the repeated-count or reserve bytes only. *value1* and *value2* should be ? due to the microcontroller RAM . The Cross Assembler will not initialize the RAM data. **DBIT** reserves a bit. The content ? denotes uninitialized data, i.e., reserves the space of the data. The Cross Assembler will gather every 8 **DBIT** together and reserve a byte for these 8 **DBIT** variables.

- Example

```
DATA            .SECTION        'DATA'
tbuf            DB ?
flag1           DBIT
sbuf            DB ?
cflag           DBIT
```

In this example, the Cross Assembler reserves byte location 0 for tbuf, bit 0 of location 1 for flag1, location 2 for sbuf and bit 1 of location 1 for cflag.

→ **Syntax**

```
name LABEL {BIT|BYTE|WORD}
```

- Description

The name with the data type has the same address as the following data variable

- Example

```
lab1            LABEL        WORD
d1              DB ?
d2              DB ?
```

In this example, d1 is the low byte of lab1 and d2 is the high byte of lab1.

→ **Syntax**

```
name EQU expression
```

- Description

The **EQU** directive creates absolute symbols, aliases, or text symbols by assigning an expression to name. An absolute symbol is a name standing for a 16-bit value; an alias is a name representing another symbol; a text symbol is a name for another combination of characters. The name must be unique, i.e. not having been defined previously. The expression can be an integer, a string constant, an instruction mnemonic, a constant expression, or an address expression.

- Example

```
accreg EQU 5
bmove EQU mov
```

In this example, the variable `accreg` is equal to 5, and `bmove` is equal to the instruction `mov`.

Macro Directives

Macro directives enable a block of source statements to be named, and then that name to be re-used in the source file to represent the statements. During assembly, the Cross Assembler automatically replaces each occurrence of the macro name with the statements in the macro definition.

A macro can be defined at any place in the source file as long as the definition precedes the first source line that calls this macro. In the macro definition, the macro to be defined may refer to other macros which have been previously defined. The Cross Assembler supports a maximum of 7 nesting levels.

→ **Syntax**

```
name MACRO [dummy-parameter [...]]
      statements
      ENDM
```

The Cross Assembler supports a directive `LOCAL` for the macro definition.

→ **Syntax**

```
name LOCAL dummy-name [...]
```

- Description

The **LOCAL** directive defines symbols available only in the defined macro. It must be the first line following the **MACRO** directive, if it is present. The `dummy-name` is a temporary name that is replaced by a unique name when the macro is expanded. The Cross Assembler creates a new actual name for `dummy-name` each time the macro is expanded. The actual name has the form `??digit`, where `digit` is a hexadecimal number within the range from 0000 to FFFF. A label should be added to the **LOCAL** directive when labels are used within the **MACRO/ENDM** block. Otherwise, the Cross Assembler will issue an error if this **MACRO** is referred to more than once in the source file.

In the following example, `tmp1` and `tmp2` are both dummy parameters, and are replaced by actual parameters when calling this macro. `label1` and `label2` are both declared **LOCAL**, and are replaced by `??0000` and `??0001` respectively at the first reference, if no other **MACRO** is referred. If no **LOCAL** declaration takes place, `label1` and `label2` will be referred to labels, similar to the declaration in the source program. At the second reference of this macro, a multiple define error message is displayed.

```
Delay MACRO tmp1, tmp2
      LOCAL label1, label2
      mov    a, 70h
      mov    tmp1, a
```

```

label1:
    mov     tmp2, a
label2:
    clr     wdt1
    clr     wdt2
    sdz     tmp2
    jmp     label2
    sdz     tmp1
    jmp     label1
ENDM

```

The following source program refers to the macro Delay ...

```

; T.ASM
; Sample program for MACRO.
.ListMacro
Delay MACRO tmp1, tmp2
    LOCAL  label1, label2
    mov    a, 70h
    mov    tmp1, a
label1:
    mov    tmp2, a
label2:
    clr    wdt1
    clr    wdt2
    sdz    tmp2
    jmp    label2
    sdz    tmp1
    jmp    label1
ENDM

data .section 'data'
BCnt db ?
SCnt db ?

code .section at 0 'code'
Delay BCnt, SCnt
end

```

The Cross Assembler will expand the macro Delay as shown in the following listing file. Note that the offset of each line in the macro body, from line 4 to line 17, is 0000. Line 24 is expanded to 11 lines and forms the macro body. In addition the formal parameters, tmp1 and tmp2, are replaced with the actual parameters, BCnt and SCnt, respectively.

```

File: T.asm           Holtek Cross-Assembler  Version 2.80           Page 1

1 0000                ; T.ASM
2 0000                ; Sample program for MACRO.
3 0000                .ListMacro
4 0000                Delay MACRO  tmp1, tmp2
5 0000                  LOCAL  label1, label2
6 0000                  mov    a, 70h
7 0000                  mov    tmp1, a
8 0000                label1:
9 0000                  mov    tmp2, a
10 0000               label2:
11 0000                  clr    wdt1
12 0000                  clr    wdt2
13 0000                  sdz    tmp2
14 0000                  jmp    label2
15 0000                  sdz    tmp1
16 0000                  jmp    label1
17 0000                ENDM
18 0000
19 0000                data .section 'data'
20 0000 00             BCnt db ?
21 0001 00             SCnt db ?
22 0002
23 0000                code .section at 0 'code'
24 0000                Delay BCnt, SCnt
24 0000 0F70          1      mov    a, 70h
24 0001 0080          R1     mov    BCnt, a
24 0002               1      ??0000:
24 0002 0080          R1     mov    SCnt, a
24 0003               1      ??0001:
24 0003 0001          1      clr    wdt1
24 0004 0005          1      clr    wdt2
24 0005 1780          R1     sdz    SCnt
24 0006 2803          1      jmp    ??0001
24 0007 1780          R1     sdz    BCnt
24 0008 2802          1      jmp    ??0000
25 0009                end

0 Errors

```

Assembly Instructions

The syntax of an instruction has the following form :

[name:] mnemonic [operand1 [,operand2]] [; comment]

where

- name:* → label name
- mnemonic* → instruction name (keywords)
- operand1* → registers
memory address
- operand2* → registers
memory address
immediate value

Name

A name is made up of letters, digits, and special characters, and is used as a label.

Mnemonic

Mnemonic is an instruction name dependent upon the type of the MCU used in the source program.

Operand, Operator and Expression

Operands (source or destination) are the argument defining values that are to be acted on by instructions. They can be constants, variables, registers, expressions or keywords. When using the instruction statements, care must be taken to select the correct operand type, i.e. source operand or destination operand. The dollar sign \$ is a special operand, namely the current location operand.

An expression consists of many operands that are combined to describe a value or a memory location. The combined operators are evaluated at assembly time. They can contain constants, symbols, or any combination of constants and symbols that are separated by arithmetic operators.

Operators specify the operations to be performed while combining the operands of an expression. The Cross Assembler provides many operators to combine and evaluate operands. Some operators work with integer constants, some with memory values, and some with both. Operators handle the calculation of constant values that are known at the assembly time. The following are some operators provided by the Cross Assembler.

- Arithmetic operators + - * / % (MOD)
- SHL 和 SHR operators
 - Syntax
 - expression* **SHR** *count*
 - expression* **SHL** *count*

The values of these shift bit operators are all constant values. The expression is shifted right SHR or left SHL by the number of bits specified by count. If bits are shifted out of position, the corresponding bits that are shifted in are zero-filled. The following are such examples:

```
mov A, 01110111b SHR 3 ; result ACC=00001110b
mov A, 01110111b SHL 4 ; result ACC=01110000b
```

- Bitwise operators NOT、AND、OR、XOR
 - Syntax
 - NOT** *expression*
 - expression1* **AND** *expression2*
 - expression1* **OR** *expression2*

expression1 **XOR** *expression2*

NOT is a bitwise complement.

AND is a bitwise AND.

OR is a bitwise inclusive OR.

XOR is a bitwise exclusive OR.

- OFFSET operator

- Syntax

OFFSET *expression*

The **OFFSET** operator returns the offset address of an expression. The expression can be a label, a variable, or other direct memory operand. The value returned by the **OFFSET** operator is an immediate operand.

- LOW、MID 和 HIGH operator

- Syntax

LOW *expression*

MID *expression*

HIGH *expression*

The **LOW/MID/HIGH** operator returns the value of an expression if the result of the expression is an immediate value. The **LOW/MID/HIGH** operators will then take the low/middle/ high byte of this value. But if the expression is a label, the **LOW/MID/HIGH** operator will take the values of the low/middle/high byte of the program count of this label.

- BANK operator

- Syntax

BANK *name*

The **BANK** operator returns the bank number allocated to the section of the name declared. If the name is a label then it returns the rom bank number. If the name is a data variable then it returns the ram bank number. The format of the bank number is the same as the BP defined. For more information of the format please refer to the data sheets of the corresponding MCUs. (Note: The format of the BP might be different between MCUs.)

Example 1:

```
mov  A, BANK start
mov  BP, A
jmp  start
```

Example 2:

```
mov  A, BANK var
mov  BP, A
mov  A, OFFSET var
mov  MP1, A
mov  A, R1
```

- Operator precedence

Precedence	Operators
1(Highest)	(), []
2	+ , - (unary), LOW, MID, HIGH, OFFSET, BANK
3	*, /, %, SHL, SHR
4	+ , - (binary)
5	>(greater than), >=(greater than or equal to), <(less than), <= (less than or equal to)
6	= = (equal to), !=(not equal to)
7	! (bitwise NOT)
8	& (bitwise AND)
9(Lowest)	(bitwise OR), ^ (bitwise XOR)

Miscellaneous

Forward References

The Cross Assembler allows reference to labels, variable names, and other symbols before they are declared in the source code (forward named references). But symbols to the right of EQU are not allowed to be forward referenced.

Local Labels

Alocal label is a label with a fixed form such as \$number. The number can be 0~29. The function of a local label is the same as a label except that the local label can be used repeatedly. The local label should be used between any two consecutive labels and the same local label name may used between other two consecutive labels. The Cross Assembler will transfer every local label into a unique label before assembling the source file. At most 30 local labels can be defined between two consecutive labels.

Example

```

Label1:                                ; label
      $1:                               ;; local label
            mov a, 1
            jmp $3
      $2:                               ;; local label
            mov a, 2
            jmp $1
      $3:                               ;; local label
            jmp $2
Label2:                                ; labe1
      Jmp $1
      $0:                               ;; local label
            jmp Label1
      $1:                               ;; local label
            jmp $0
Label3:

```

Reserved Assembly Language Words

The following tables list all reserved words used by the assembly language.

- Reserved Names (directives, operators)

\$	DUP	INCLUDE	NOT
*		LABEL	OFFSET
+	ELSE	.LIST	OR
-	END	.LISTINCLUDE	ORG
.	ENDIF	.LISTMACRO	PAGE
/	ENDM	LOCAL	PARA
=	ENDP	LOW	PROC
?	EQU	MACRO	PUBLIC
[]	ERRMESSAGE	MESSAGE	RAMBANK
AND	EXTERN	MID	ROMBANK
BANK	HIGH	MOD	.SECTION
BYTE	IF	NEAR	SHL
DB	IFDEF	.NOLIST	SHR
DBIT	IFE	.NOLISTINCLUDE	WORD
DC	IFNDEF	.NOLISTMACRO	XOR

- Reserved Names (instruction mnemonics)

ADC	HALT	RLCA	SUB
ADCM	INC	RR	SUBM
ADD	INCA	RRA	SWAP
ADDM	JMP	RRC	SWAPA
AND	MOV	RRCA	SZ
ANDM	NOP	SBC	SZA
CALL	OR	SBCM	TABRDC
CLR	ORM	SDZ	TABRDL
CPL	RET	SDZA	XOR
CPLA	RETI	SET	XORM
DAA	RL	SIZ	
DEC	RLA	SIZA	
DECA	RLC	SNZ	

- Reserved Names (registers names)

A	WDT	WDT1	WDT2
---	-----	------	------

Cross Assembler Options

The Cross Assembler options can be set via the Options menu Project command in HTIDE3000. The Cross Assembler Options is located on the center part of the Project Option dialog box, as shown in Fig 3-12.

The symbols could be defined in the Define Symbol edit box.

→ **Syntax**

symbol1 [= value 1] [,symbol2 [= value2] [,...]]

- Example

debugflag=1, newver=3

The check box of the Generate listing file is used to decide whether the listing file should be generated or not. If the check box is checked, the listing file will be generated. Otherwise, it won't be generated.

Assembly Listing File Format

The Assembly Listing File contains the source program listing and summary information. The first line of each page is a title line which include company name, the Cross Assembler version number, source file name, date/time of assembly and page number.

Source Program Listing

Each line in the source program has the following syntax:

line-number offset [code] statement

- *Line-number* is the number of the line starting from the first statement in the assembly source file (4 decimal digits).
- The 2nd field – *offset* – is the offset from the beginning of the current section to the code (4 hexadecimal digits)
- The 3rd field – *code* – is present only if the statement generates code or data (two hexadecimal 4-digit data)

The code shows the numeric value in hexadecimal if the value is known at assembly time. Otherwise, a proper flag will indicate the action required to compute the value.

The following two flags may appear behind the code field.

- **R** → relocatable address (Cross Linker must resolve)
- **E** → external symbol (Cross Linker must resolve)

The following flag may appear before the code field

= → EQU or equal-sign directive

The following 2 flags may appear in the code field

--- → section address (Cross Linker must resolve)

nn[xx] → DUP expression: nn DUP(?)

- The 4th field – statement – is the source statement shown exactly as it appears in the source file, or as expanded by a macro. The following flags may appear before a statement.

– n → Macro-expansion nesting level

– C → line from INCLUDE file

- Summary

```

0          1          2          3          4          5          6
123456789012345678901234567890123456789012345678901234567890...
IIII  oooo  hhhh  hhhh  EC  source-program-statement
                        Rn
  
```

IIII → line number (4 digits, right alignment)

oooo → offset of code (4 digits)

hhhh → two 4-digits for opcode

E → external reference

C → statement from included file

R → relocatable name

n → Macro-expansion nesting level

Summary of Assembly

The total warning number and total error number is the information provided at the end of the Cross Assembler listing file.

Miscellaneous

If any errors occur during assembly, each error message and error number will appear directly below the statement where the error occurred.

→ Example of Assembly Listing File

File: SAMPLE.ASM Holtek Cross-Assembler Version 2.86 Page 1

```

1 0000          page 60
2 0000          message 'Sample Program 1'
3 0000
4 0000          .listinclude
5 0000          .listmacro
6 0000
7 0000          #include "sample.inc"

1 0000          C pa    equ    [12h]
2 0000          C pac   equ    [13h]
3 0000          C pb    equ    [14h]
4 0000          C pbc   equ    [15h]
5 0000          C pc    equ    [16h]
6 0000          C pcc   equ    [17h]
7 0000          C

8 0000
9 0000          extern extlab : near
10 0000         extern extb1 : byte
11 0000
12 0000         clrpb macro
13 0000         clr pb
14 0000         endm
15 0000
16 0000         clrpa macro
17 0000         mov a, 00h
18 0000         mov pa, a
19 0000         clrpb
20 0000         endm
21 0000
22 0000         data .section 'data'
23 0000 00          b1    db ?
24 0001 00          b2    db ?
25 0002 00          bit1  dbit
26 0003
27 0000         code .section 'code'
28 0000 0F55        mov a, 055h
29 0001 0080        R mov b1, a
30 0002 0080        E mov extb1, a
31 0003 0FAA        mov a, 0aah
32 0004 0093        mov pac, a
33 0005             clrpa
33 0005 0F00        1 mov a, 00h
33 0006 0092        1 mov [12h], a
33 0007             1 clrpb
33 0007 1F14        2 clr [14h]
34 0008 0700        R mov a, b1
35 0009 0F00        E mov a, bank extlab
36 000A 0F00        E mov a, offset extb1
37 000B 2800        E jmp  extlab
38 000C
39 000C 1234 5678    dw  1234h, 5678h, 0abcdh, 0ef12h
   ABCD EF12
40 0010             end

0 Errors

```

Chapter 10

Cross Linker

10

What the Cross Linker Does

The Cross Linker creates task files from the object files generated by the Cross Assembler or the C compiler. The Cross Linker combines both code and data in the object files and searches the named libraries to resolve external references to routines and variables. It also locates the code and data sections at the specified memory address or at the default address, if no explicit address is specified. Finally, the Cross Linker copies both the program codes and other information to the task file. It is this task file that is loaded by the Holtek IDE Holtek Integrated Development Environment, into the Holtek HT-ICE In-Circuit Emulator, for debugging. The libraries included by the Cross Linker were generated by the Holtek library manager.

Cross Linker Options

The options specify and control the tasks performed by Cross Linker. In chapter 3, Option Menu, Project command provides a dialog box, Cross Linker Options, to specify these options to the Cross Linker. These options are:

Libraries

- Syntax
libfile 1[,libfile2...]

This option informs the Cross Linker to search the specified library files if the input

object files refer to a procedure or variable which is not defined in any of the object files. If a module of a library file contains the referred procedure or variable, then only this module, not the whole library file will be included in the output task file. (refer to Chapter 13 Library Manager)

Section Address

- Syntax
`section_name=address[,section_name=address]...`

This option specifies the address of the sections; section_name is the name of the section that is to be addressed. The section_name must be defined in at least one input object file, otherwise a warning will occur. The address is the specified address whose format is xxxx in hexadecimal format.

Generate Map File

The check box of this option is to specify whether the map file is generated or not.

Map File

The map file lists the names and loads the addresses and lengths of all sections in a program as well as listing the messages it encounters. The Cross Linker gives the address of the program entry point at the end of the map file. The map file also lists the names and loads addresses of all public symbols. The names and file names of the external symbols or procedures are recorded in the map file if no corresponding public symbol or procedure can be found. The contents of the map file are as follows :

```
Holtek (R) Cross Linker Version 8.1
Copyright (C) HOLTEK Semiconductor Inc. 2007-2008.All
rights reserved.
Input Object File: C:\SAMPLE\T2.OBJ
Input Library File: C:\Program Files\Holtek MCU Development
Tools\HT-IDE3000V7.1\LIB\MATH6.LIB
```

```
SECTION
      Start  End      Length  Class  Name
      0000h  0002h    0003h   CODE   @CODE (C:\Documents
and
Documents\HTK_Project\C-INTR-TEST\C-INTR-TEST.OBJ)
      0003h  0003h    0001h   CODE   STARTSEC (C:\Documents
and
Settings\panwei\My
```

```

Documents\HTK_Project\C-INTR-TEST\C-INTR-TEST.OBJ
0004h 0004h 0001h CODE @@ExtISR (C:\Documents
and Settings\panwei\My
Documents\HTK_Project\C-INTR-TEST\C-INTR-TEST.OBJ)
0005h 000fh 000bh CODE @main (C:\Documents and
Settings\panwei\My Documents\HTK_Project\C-INTR-TEST\C-INTR-TEST.OBJ)
0010h 0017h 0008h CODE @ExtISR (C:\Documents
and Settings\panwei\My
Documents\HTK_Project\C-INTR-TEST\C-INTR-TEST.OBJ)
0040h 0041h 0002h DATA CTMPDATA (C:\Program
Files\Holtek MCU Development Tools\HT-IDE3000V7.1\LIB\MATH6.LIB)
0048h 0048h 0001h DATA @ExtISR (C:\Documents
and Settings\panwei\My
Documents\HTK_Project\C-INTR-TEST\C-INTR-TEST.OBJ)

```

```

Local Sections
Start End Length Class Name
0043h 0044h 0002h LOCAL _funa (C:\Documents and
Settings\panwei\My Documents\HTK_Project\C-INTR-TEST\C-INTR-TEST.OBJ)
0042h 0042h 0001h LOCAL _funb (C:\Documents and
Settings\panwei\My Documents\HTK_Project\C-INTR-TEST\C-INTR-TEST.OBJ)
0044h 0047h 0004h LOCAL _func (C:\Documents and
Settings\panwei\My Documents\HTK_Project\C-INTR-TEST\C-INTR-TEST.OBJ)
0042h 0043h 0002h LOCAL _fund (C:\Documents and
Settings\panwei\My Documents\HTK_Project\C-INTR-TEST\C-INTR-TEST.OBJ)
0048h 0048h 0000h LOCAL @DUMMY (C:\Documents and
Settings\panwei\My Documents\HTK_Project\C-INTR-TEST\C-INTR-TEST.OBJ)
0042h 0047h 0006h LOCAL _main (C:\Documents and
Settings\panwei\My Documents\HTK_Project\C-INTR-TEST\C-INTR-TEST.OBJ)

```

```

Independent Local Sections
Start End Length Class Name
0048h 0048h 0000h ILOCAL _ExtISR (C:\Documents and
Settings\panwei\My Documents\HTK_Project\C-INTR-TEST\C-INTR-TEST.OBJ)

```

Public Symbols Information

```

Address Public by Name
0040h V1A
0041h V1S

```

```

Address Public by Value
0040h V1A
0041h V1S

```

ROM Usage Statistics

```

Size Used Percentage
0800h 0018h 1%

```

RAM Usage Statistics

```

Size Used Percentage
0040h 0009h 14%

```

```
Call Tree
  _funa
    _funb
      _func
        _fund
          @DUMMY
            _main
```

HLINK: Program entry point at section '@CODE' (address 0) of file 'C:\Documents and Settings\panwei\My Documents\HTK_Project\C-INTR-TEST\C-INTR-TEST.OBJ'

Total 0 Error(s), Total 0 Warning(s)

Cross Linker Task File and Debug File

One of the Cross Linker's output files is the task file which consists of two parts, a task header and binary code. The task header contains the Cross Linker version, the MCU name and the ROM code size. The binary code part contains the program codes. The other Cross Linker output file is the debug file which contains all information referred to by the Holtek IDE debugging program. This information includes source file names, symbol names and line numbers as defined in the source files. The Holtek IDE will refer to the symbolic debugging function information. This file should not be deleted unless the debugging procedure is completed, otherwise the Holtek IDE will be unable to support the symbolic debugging function.

Part III

Utilities

In addition to the previously discussed general purpose 8-bit MCU development tools, Holtek also supplies several other utilities for its range of special purpose Voice and LCD MCU devices by supplying all the necessary tools and step by step guide for relevant simulation of voice synthesis and tone generator applications as well as the tools for real time hardware LCD panel simulation. This part contains all the information needed to program and debug relevant applications quickly and efficiently.

Chapter 11

Library Manager

11

What the Library Manager Does

The Library Manager provides functions to process the library files. The library files are utilized in the creation of the output file by the Cross Linker. A library is a collection of one or more object modules which are assembled or compiled and ready for linking. It stores the modules that other programs may require for execution.

By using the Library Manager, library files can be created. Object files including common routines may be added to the library files. Before creating these object files, the names of all common routines must be made public by using the assembly directive PUBLIC (refer to the chapter on Assembly Language and Cross Assembler). The Cross Assembler generates the output object file (.OBJ) while the Library Manager adds this object file into the specified library file. When the Cross Linker has found unresolved names in a program during the linking process, it will search the library files for these unresolved names, and extracts a copy of the module containing that name. If an unresolved name has been found in this library module, the module will be linked to the program.

To Setup the Library Files

The Library Manager provides the following functions :

- Create new library files
- Add/Delete a program module to/from a library file

- Extract a program module from a library file, and create an object file

To select use the Tools Menu and the Library Manager command as shown in Fig 11-1.

Fig 11-2 shows the dialogue box for processing the functions of the Library Manager.

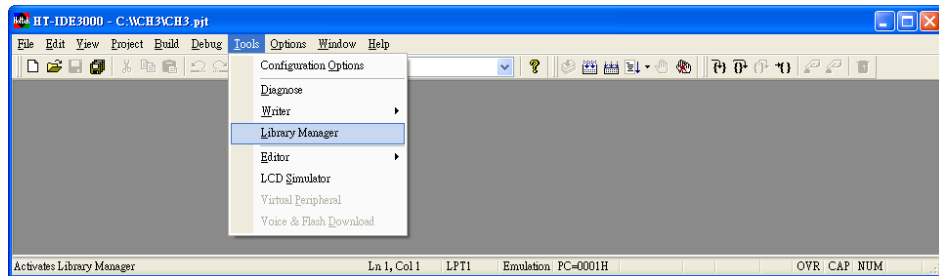


Fig 11-1

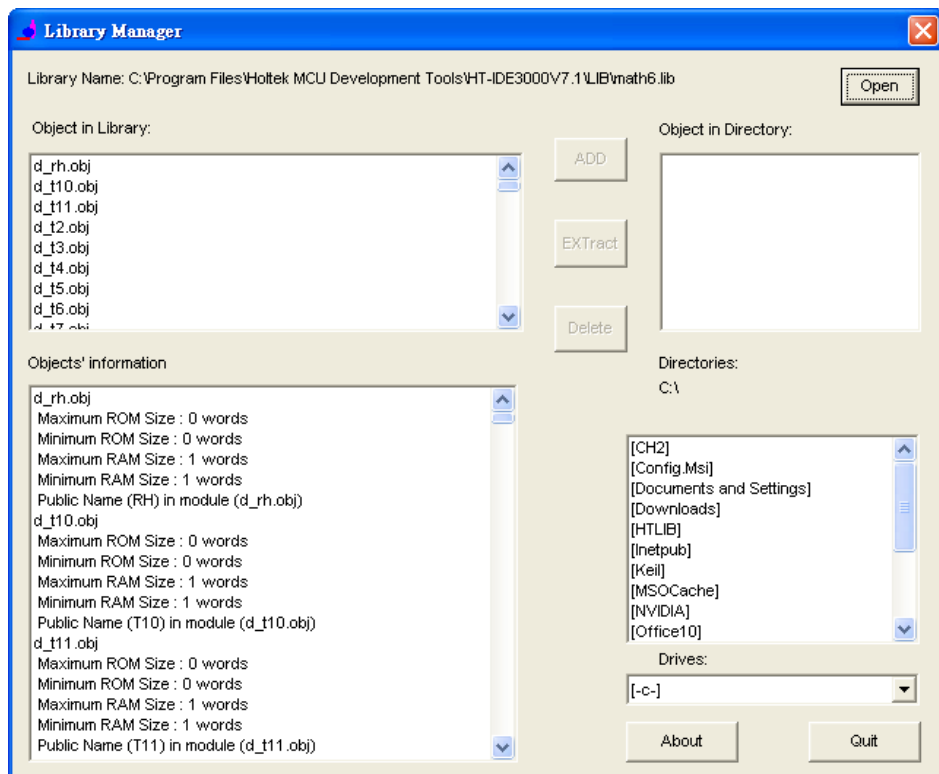


Fig 11-2

Create a New Library File

Press Open button, Fig 11-3 is displayed

Type in a new library file name and press the OK button, Fig 11-4 is displayed for confirmation. If the Yes button is chosen, a new library file will be created but will not contain any program modules.

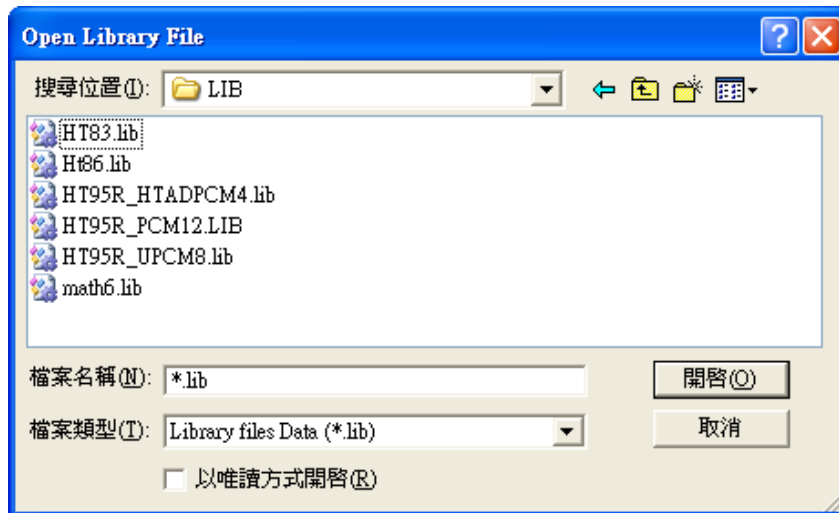


Fig 11-3



Fig 11-4

Add a Program Module into a Library File

Select an object module from the “Object in Directory” box, and press the [ADD] button to add this object module into this library file.

Delete a Program Module from a Library File

Select an object module from the “Object In Library” box, and press the [Delete] button to delete this object module from the library file.

Extract a Program Module from Library and Create An Object File

Select an object module from the “Object in Library” box, and press [Extract] button. A

file will then be created with the same name and same content as the selected object module. It is displayed on the “Object in Directory” box.

Object Module Information

Press the Open button, Fig 11-3 is displayed. Select a library file from the box below the File Name box, press OK button. From Fig 11-2, all the object modules of the selected library file are listed in the “Object in Library” box. The following information about each object module is also listed in the “Objects’ Information” box :

- **Maximum ROM size**
The maximum size used by this object module program code. Dependent upon the code section align type.
- **Minimum ROM size**
The minimum actual size used by this object module program code
- **Maximum RAM size**
The maximum size used by this object module program data. It depends on the data section align type.
- **Minimum RAM size**
The minimum, actual size used by this object module program data.
- **Public Name**
The names of all public symbols in this object module.

Chapter 12

LCD Simulator

12

Introduction

The Holtek LCD simulator, known as the HT-LCDS, provides a mechanism allowing users to simulate the output of LCD drivers. According to the user designed patterns and the control programs, the HT-LCDS displays the patterns on the screen in real time. It facilitates the development process even if the actual LCD hardware panel is unavailable. Note that if the current project's microcontroller does not support LCD functions, these commands are disabled.

LCD Panel Configuration File

Before starting the LCD simulation, an LCD panel configuration file must first be setup. The HT-LCDS will obtain the LCD data and display LCD patterns on the screen according to the LCD panel configuration file. The HT-LCDS cannot simulate the LCD action if this file is absent. For microcontrollers possessing an LCD driver, the corresponding panel configuration file has to be setup for LCD simulation. The LCD simulator command within the Tools menu will then be enabled to setup the panel configuration file and for simulation (Fig12-1). The LCD panel configuration file contains two kinds of data, panel configuration data and pattern information, which users can setup using the HT-LCDS.

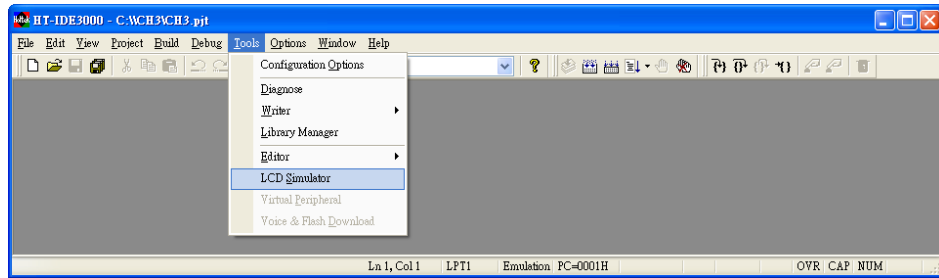


Fig 12-1

Relationship Between the Panel File and the Current Project

By default, the panel configuration file has the same file name as the current project name except for the extension name, which is .lcd. The HT-LCDS assumes this file to be the corresponding panel configuration file of the current project. The panel configuration file is generated by the HT-LCDS File menu, New command or the New button on the toolbar. A different file name from the current project name can be assigned to the panel configuration file by clicking File menu, Save command or Save button on the toolbar.

When the HT-LCDS begins simulation, it references the current active panel configuration file to obtain its simulation information. The LCD panel configuration file is activated by selecting the New or Open command of the HT-LCDS File menu. The file name of the LCD panel configuration file may be the same as the current project name or a different name can be chosen.

Selecting the HT-LCDS

When selected from within the Tools menu, the LCD simulator as shown in Fig 12-2 is displayed if the corresponding panel configuration file of the current project exists. The file name of each bitmap pattern is shown at the specified COM/SEG position of the table. At the same time, these patterns are shown on the above panel screen. If the corresponding panel configuration file does not exist within the project directory, both the panel screen and the COM/SEG table will not be displayed. Fig 12-3 shows the HTLCDS menu bar information.

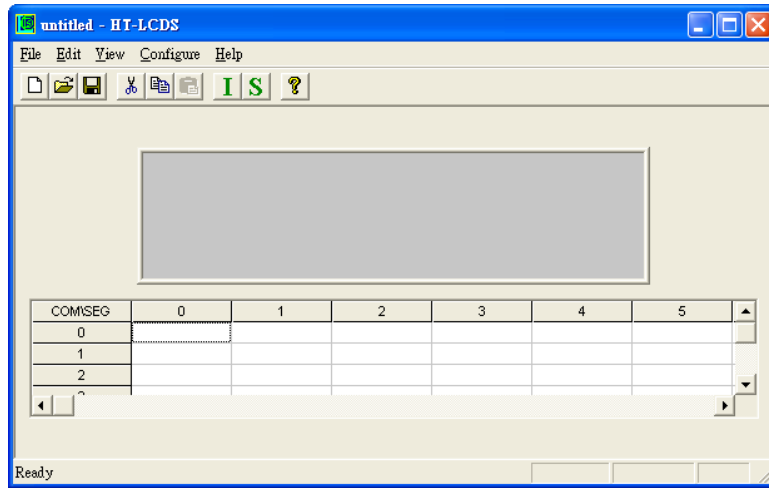


Fig 12-2

The Fig below shows the HT-LCDS menu bar information.

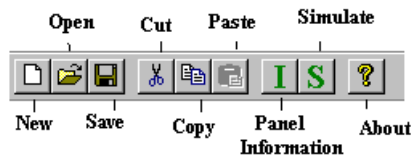


Fig 12-3

- New : create a new panel configuration file
- Open : open an existing panel configuration file
- Save : save the panel configuration file
- Cut : delete a pattern
- Copy : copy a pattern to the clipboard
- Paste : add the copied pattern to the panel
- I : panel information dialog
- S : enter the LCD simulation mode

LCD Panel Picture File

The LCD panel picture (pattern) file is a bitmap file (.bmp) which represents the practical patterns and their positions on the panel. The bitmap file can be created using any bitmap editor and provides another method of setting up the LCD panel pattern information by using the HT-LCDS Edit menu, Panel Editor command. The bitmap file is optional, users

can setup the LCD panel pattern information even if the LCD panel picture file is absent.

Setup the LCD Panel Configuration File

The following two steps are used to setup a panel configuration file:

- Setup the panel configurations, including the segment and common number of the LCD driver as well as the width and height size of the panel in pixels. Also, the directory of the panel configuration file and the dot matrix mode can be selected.
- Select the patterns and their positions. This will setup the relationship between the patterns and the COM/SEG positions.

Setup the Panel Configurations

To setup the panel configurations by selecting the HT-LCDS File menu, New command. The Panel Configuration dialog box (Fig 12-4) will be displayed. Setup the correct LCD driver data, COM/SEG number, Width, Height and Directory of the pattern, then press the [OK] button. After setting up the panel configuration, the system returns to Fig 12-2 for pattern selection.

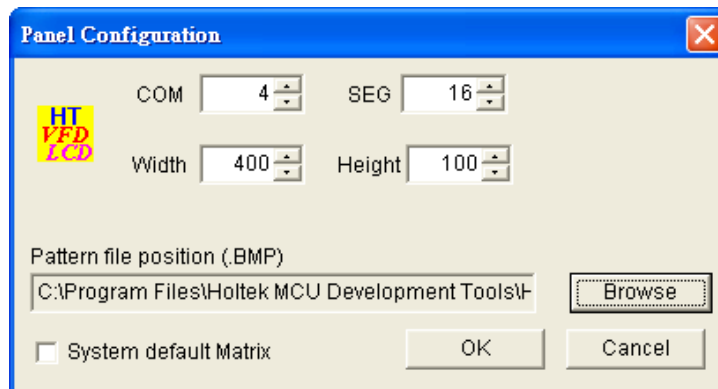


Fig 12-4

The panel configurations include:

- The default number of the LCD driver for this microcontroller is displayed when Fig12-4 is displayed. To ensure that these numbers are the same as the actual setting number of the LCD driver for the micro controller.
- Width and Height. These are the size of the panel screen in pixels and can be changed to adjust the panel screen.
- Panel configuration file directory. Select the directory where the panel configuration

file is stored using the browse button or setup to have the same directory as the project.

- Dot Matrix Mode. To simulate dot matrix type LCD panels. Fig 12-5 shows the dot matrix screen.

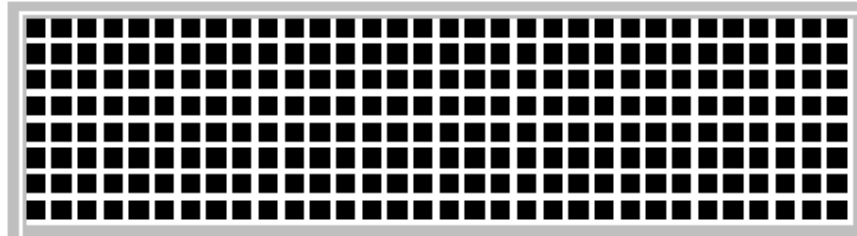


Fig 12-5

Note: It is important not to set different COM or SEG number from the actual corresponding LCD driver numbers, otherwise unpredictable results will occur.

Select the Patterns and Their Positions

The following methods show the steps of selecting the patterns and their positions

- To create a new panel configuration file using the HT-LCDS File menu New command. After having set the panel configuration, Fig 12-2 is displayed. The user then has to select the patterns from the Pattern Information dialog box (Fig 12-6) and set the COM/SEG positions. The section, Add a new pattern, describes the procedure in detail.
- To open an existing panel configuration file using the HT-LCDS File menu Open command. The patterns are displayed as shown on the panel screen in Fig 12-2 and the pattern file names are displayed as shown in the Fig12-2 COM/SEG table position. Users can add/delete/change the pattern information, including the pattern file and pattern positions.
- To open a panel picture file using the HT-LCDS Edit menu Panel Editor command. If this panel picture file has been setup already, then it is not necessary to select the patterns, it is only necessary to select the pattern positions. The section, Define the pattern using the Panel Editor, describes the procedure in detail.

Add a New Pattern

- Move the cursor to a COM/SEG position on the grid as shown in Fig 12-2 and double click the mouse. The Pattern Information dialog box, as shown in Fig 12-6, is

displayed. All the pattern files (.bmp) in the project's directory are listed in the Pattern List box. The Size field is the bitmap size of the selected pattern, Com and Seg fields are the numbers of the selected COM/SEG position of this pattern. None of these three fields can be modified.

- Select a pattern, a bitmap file, from the Pattern List box, or click the Browse button to change to another directory and select a pattern from that directory. The HT-LCDS uses 2-color bitmap files as the image source of patterns. The Preview-window zooms into the selected pattern.
- Set the X/Y positions in the panel screen for the selected pattern.
- Press the [OK] button and return to Fig 12-2, then click the File menu, Save command or click the Save button on the toolbar. The panel file has now been created or modified.

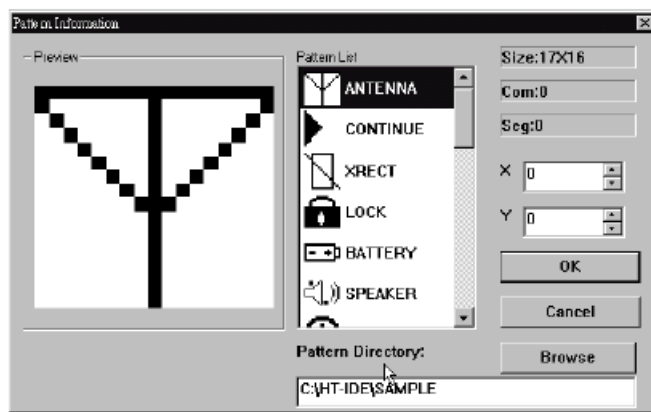


Fig 12-6

Delete a Pattern

- As shown in Fig 12-2, select the COM/SEG position of the pattern to be deleted and press the [Delete] key or click the Cut button on the toolbar.

Change the Pattern

- Delete the selected pattern first, then add a new pattern to change the pattern.
- Alternatively, as shown in Fig 12-2, select the COM/SEG position of the selected pattern and double click the mouse. The Pattern Information dialog box, as shown in Fig 12-6, is displayed. Select a pattern from the Pattern List box and press the [OK] button.

Change the Pattern Position

- As shown in Fig 12-2, use the Select-Drag-Drop method to move the pattern directly onto the panel screen.
- Alternatively, as shown in Fig 12-2, double click the COM/SEG position of the selected pattern. The Pattern Information dialog box, in Fig 12-6, is displayed. Set the X, Y value of the new position and press the [OK] button.

When the above operations have been completed and the system has returned to that shown in Fig 12-2, click the HT-LCDS File menu, Save command or click the Save button on the toolbar. The panel file has now been created or modified.

How to Add a User-define Matrix

The HT-LCDS supports a mapping strategy (File menu, Import user matrix command) which can help define a new matrix if the COM/SEG number is not equal to the ROW/COL number of the LCD panel. For example, Assume there is an LCD panel of 2 COMs and 6 SEGs, and assuming this LCD panel is a 3 ROWs 4 COLs matrix, as shown in the following mapping

COM0-SEG0	COM0-SEG1	COM0-SEG2	COM0-SEG3
COM1-SEG0	COM1-SEG1	COM1-SEG2	COM1-SEG3
COM0-SEG4	COM0-SEG5	COM1-SEG4	COM1-SEG5

A definition file for the above matrix can be defined as follows,

```

; MATRIX.DEF
; Comment line
ROW=3
COLUMN=4
; mapping syntax : ROW,COL=>COM,SEG
0 , 0 => 0 , 0 ; Map Row0 co10 to COM0 SEG0
0 , 1 => 0 , 1 ; Map Row0 co11 to COM0 SEG1
0 , 2 => 0 , 2 ; Map Row0 co12 to COM0 SEG2
0 , 3 => 0 , 3 ; Map Row0 co13 to COM0 SEG3
1 , 0 => 1 , 0 ; Map Row1 co10 to COM1 SEG0
1 , 1 => 1 , 1 ; Map Row1 co11 to COM1 SEG1
1 , 2 => 1 , 2 ; Map Row1 co12 to COM1 SEG2
1 , 3 => 1 , 3 ; Map Row1 co13 to COM1 SEG3
2 , 0 => 0 , 4 ; Map Row2 co10 to COM0 SEG4
2 , 1 => 0 , 5 ; Map Row2 co11 to COM0 SEG5
2 , 2 => 1 , 4 ; Map Row2 co12 to COM1 SEG4
2 , 3 => 1 , 5 ; Map Row2 co13 to COM1 SEG5

```

Define the Pattern Using the Panel Editor

The HT-LCDS supports a full panel edit interface to define the LCD panel patterns. If a panel picture file has been drawn already, then it is not necessary to set all pattern files in the panel respectively. The only requirement is to select the pattern positions.

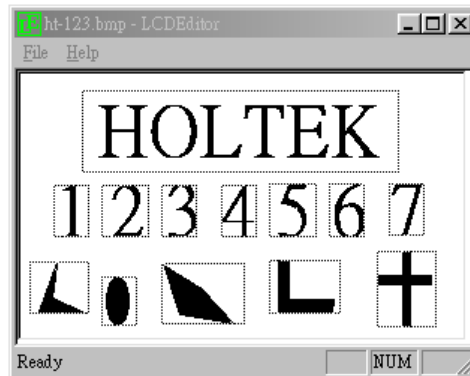


Fig 12-7

The following steps select the pattern positions for all the patterns in the LCD panel

- Invoke the Panel Editor by selecting the Edit command, Panel Editor command after having set the panel configuration
- Select the File menu, Open command in the Panel Editor to open the panel picture file (.bmp)

Note: Supports 2-color .BMP only

- The panel will be displayed in the window as in Fig 12-7
- Select the pattern for each COM/SEG by using double-click or drag-and-drop methods. The Save Pattern dialog box will be displayed after which the pattern information can be entered.
- Repeat the above step for all patterns in the panel.
- After having set the pattern information for all patterns, return to the Panel Editor window and save all the settings using the File menu Save command.
- Exit the Panel Editor and return to the HT-LCDS, the panel will now display the new settings.

Add New Pattern Items Using a Batch File

The HT-LCDS provides a method to add pattern items from a batch file using the Edit menu and Add Item Batch command. The batch file is a text file with an extension

name .BTH. All the pattern items in the batch file will define the pattern file name and its positions. After selecting a batch file using the Edit menu's Add Item Batch command, the HT-LCDS adds all patterns depicted in the batch file at the specified positions of the panel. The following is an example of a .BTH file.

```

; this is a comment line.
; item syntax : BMPfile.bmp, COM, SEG, X, Y
CRYSTAL.BMP,    0, 2, 120, 30
FION.BMP,       2, 3, 200, 50
CLIN.BMP,       3, 2, 130, 90
STEVE.BMP,      4, 4, 20,  40
    
```

Selecting Color for an LCD Panel

The HT-LCDS provides a palette dialog, as shown in Fig.12-8, for selecting the colors of the panel using the HT-LCDS Configure menu and Set Panel Color command.

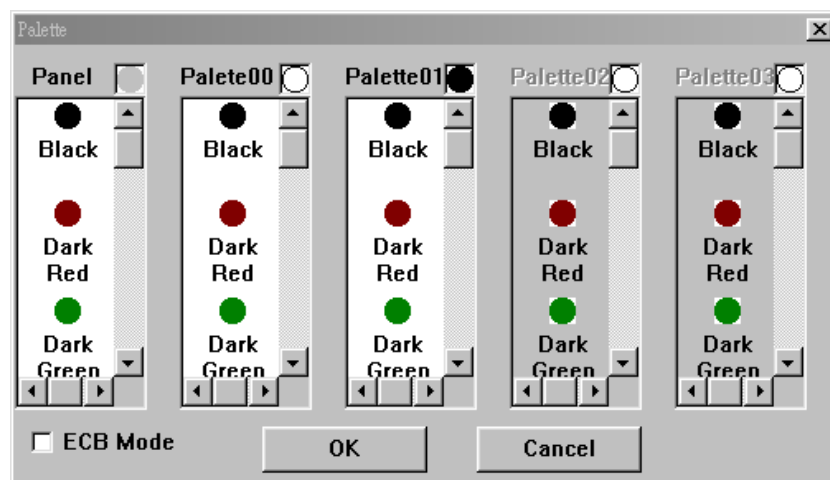


Fig 12-8

Note: The ECB mode is for HTG21x0 color LCD only.

Setting Pattern Color for VFD Panel

The HT-LCDS provides an interface, as shown in Fig.12-9, for setting the color of each pattern for Holtek's VFD MCU (eg. HT49CVX series) Select Configure menu and execute the Set VFD pattern Color command to accomplish this setting.

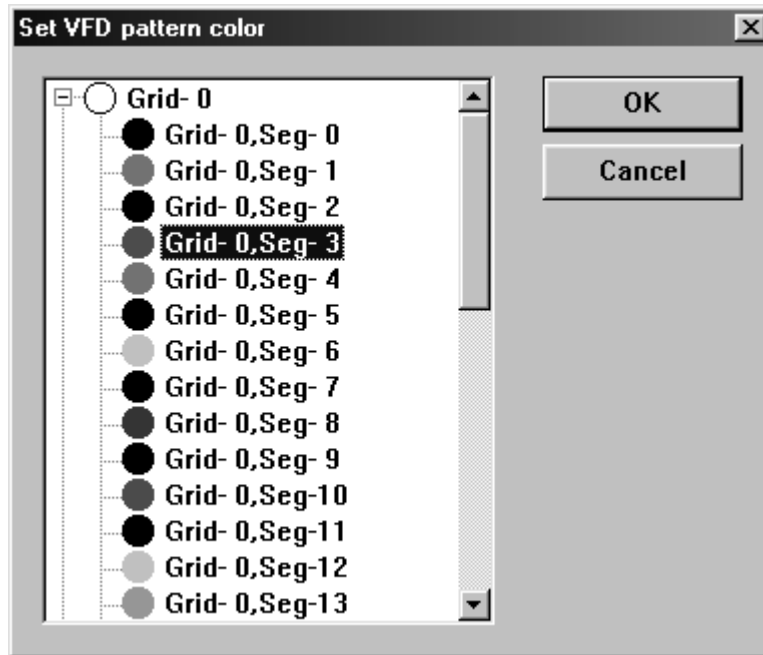


Fig 12-9

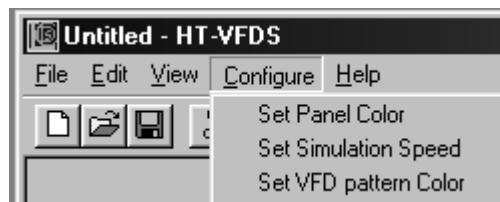


Fig 12-10

Simulating the LCD

Before starting the LCD simulation, ensure that the HT-LCDS refers to the correct panel configuration file. Enter the HT-LCDS environment by selecting the Tools menu, LCD Simulator command as shown in Fig 12-1 and Fig 12-2.

- Click once the S button on the toolbar allowing the HT-LCDS to begin LCD simulation while referring to the corresponding panel configuration file.
- Open a panel configuration file which is not the corresponding panel configuration file of the current project and click the S button on the toolbar. The HT-LCDS will then

begin LCD simulation while referring to the opened panel configuration file. When the HT-LCDS begins simulation, a window as shown in Fig 12-11 will be displayed while the most recent LCD patterns will be displayed on the panel screen.

Stop the Simulation

Double click the title bar of the LCD simulation window to make the HT-LCDS return to the edit mode.

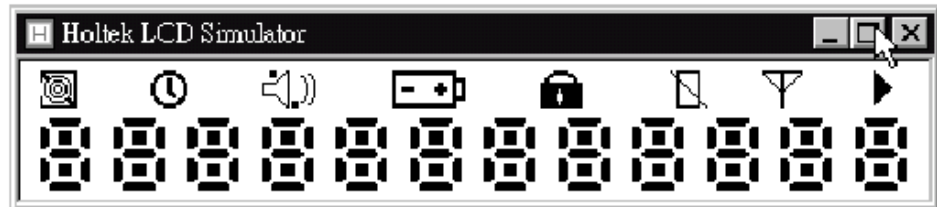


Fig 12-9

Chapter 13

Virtual Peripheral Manager

13

Introduction

In most practical applications the chosen MCU is connected to some forms of external hardware to implement the necessary user functions, however the inclusion of this external hardware in the simulation process is usually outside the scope of most MCU simulators. To overcome this problem, Holtek has developed a Virtual Peripheral Manager, or VPM, which enables the user to add a range of external peripheral devices to the MCU project. Used in conjunction with the HT-IDE simulator, the VPM enables the user to directly drive and monitor the inputs and outputs of these external hardware devices allowing for more efficient debugging and implementation of user applications.

The VPM Window

Fig 13-1 shows a practical example of a VPM window. As in most Windows applications the VPM window incorporates a toolbar for the function menus and a status bar to indicate program information with the main screen area displaying the peripherals or devices which have been added to the project.

The peripherals added to the project are known as components in the VPM. Components can be selected by clicking the mouse left button on the component required. Within this document the selected component will be referred to as the current component. By double clicking on the current component a connect dialog box will be displayed which

permits the necessary connections to be made between the component and the MCU. By clicking the right mouse key, on certain current components a configuration" dialog box will be displayed allowing attributes to be setup for that particular component.

In the status bar there are four fields, Mode, Current Component, Time and Cycle. The Mode field indicates whether the VPM is currently in configuration mode or running mode. The Current Component field shows the name of the current component. The Time field and Cycle field show the total execution time and cycle count respectively while the VPM is in running mode.

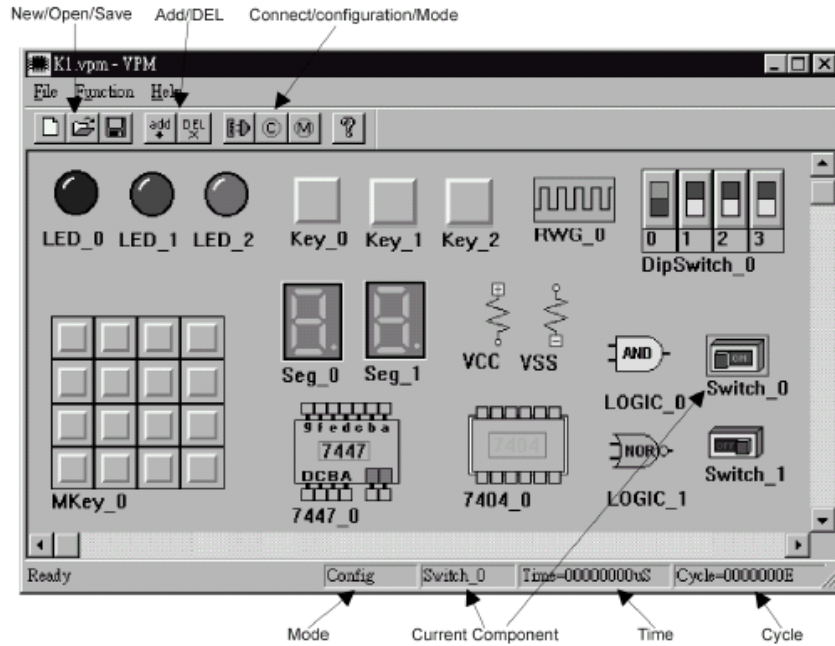


Fig 13-1

VPM Menu

File Menu

There are six functions in the File menu as shown in Fig 13-2. Three of the main functions can also be found on the toolbar as shown in Fig 13-3.



Fig 13-2



Fig 13-3

- **New**
Create a new VPM project. Each time the VPM is entered the system automatically creates a new project.
- **Open**
Open an existing VPM project.
- **Save**
Save current project to file.
- **Save As**
Save current project with another file name to file.
- **Exit**
Exit VPM and return to Windows.

Function Menu

There are five functions in the Function menu as shown in Fig 13-4. All of these functions can also be found on the toolbar as shown in Fig 13-5.



Fig 13-4

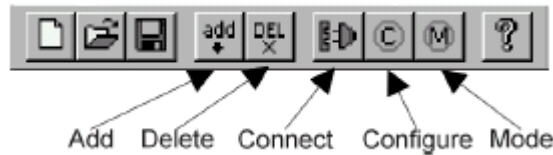


Fig 13-5

→ **Add**

Add a new peripheral to the project.

Click the Add button on the toolbar. An Add Peripheral dialog will be displayed as shown in Fig 13-6. Select the peripheral desired and click the OK button.

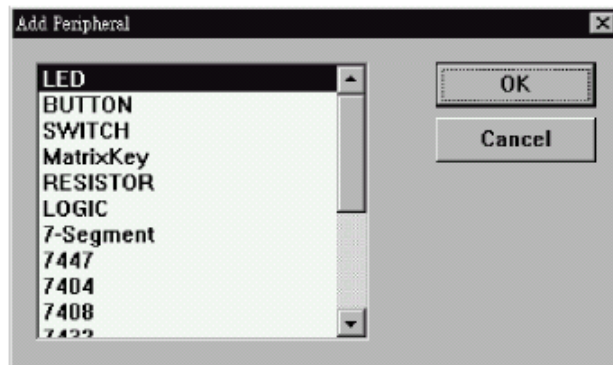


Fig 13-6

→ **Delete**

Delete a peripheral from the project. Select the component to be deleted and click the Del button. The selected component will be removed from the project.

→ **Connect**

Select a component and click the Connect button on the toolbar. A Connect Dialog will be displayed like Fig. 13-7. The connection status of the current component will be displayed in Connect status list box. The Connect/Disconnect button can be used again to adjust

the connection status between components.

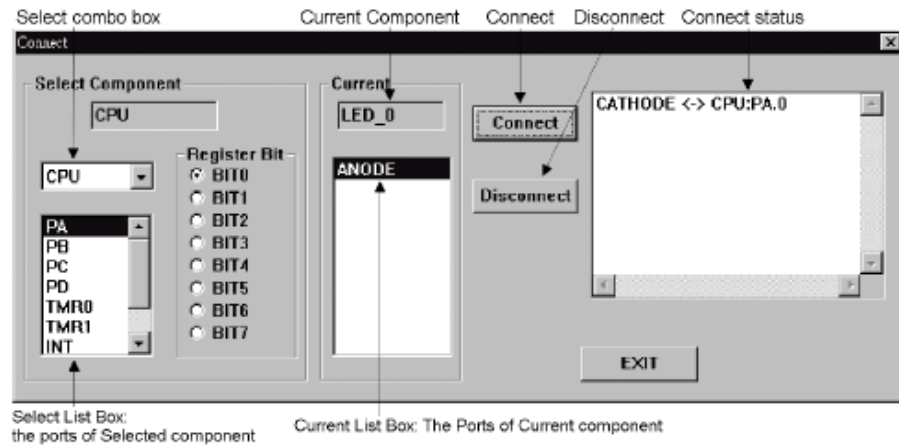


Fig 13-7

As an example, Fig. 13-7 shows the Connect dialog box for an LED component named LED_0. In this example, the current component is LED_0. The Select combo box will display all the components in this project that can be connected to LED_0. The Select List Box will display all the ports of the selected component. The Register Bit shows the port information details. The peripheral of an LED has two pins, one anode and one cathode. In this example, LED_0 s CATHODE pin has been connected to the CPU Port A bit0.

→ **CONFIG**

Some peripherals include some user adjustable attribute options. To do this the component should first be selected and then the Configure button pressed. If the component has attribute options, the Configuration Dialog box will be displayed. Fig. 13-8 shows an example of an LED configuration dialog box.



Fig 13-8

→ **Mode**

There has two modes, configuration mode and running mode. By clicking on the mode button, or selecting mode item from the function menu, the system will toggle the VPM between these two modes. In the configuration mode, the virtual external circuit can be edited using the Add/Del/CONFIG functions. In the running mode, the VPM will display the operations of these components according to their specific configurations in addition to displaying the Holtek IDE MCU simulation results.

The VPM Peripherals

LED



Fig 13-9

The LED has two pins, one cathode and one anode. When the cathode =0 and the anode =1, the LED will be illuminated. The LED has a colour option as shown in the configuration dialog box.

Button/Switch



Fig 13-10

The BUTTON/SWITCH has two options, the debounce time and the switch status when in the open position. The debounce time units are in milliseconds. The BUTTON has a non-latching momentary operation while the SWITCH has a latching non-momentary operation. The DipSwitch peripheral offers a means of providing multiple switches in a single package, the size of which is adjustable.



Fig 13-11

Seven Segment Display

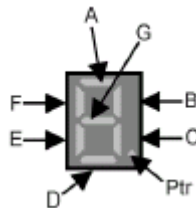


Fig 13-12

A seven segment display is formed from eight individual leds known as A, B, C, D, E, F, G and ptr. Each of these individual leds is connected to an input pin of the same name and also to a common pin. This common pin can be either a cathode (-) or an anode (+) connection which determines the polarity of the display.

→ **Resistor**



Fig 13-13

The resistors exist to provide a pull-up or pull-down function and are connected to either VCC or VSS respectively. The required configuration is set using their respective configuration dialog box.

→ **Logic gate**

Logic gates are provided to give a total of six logic functions.

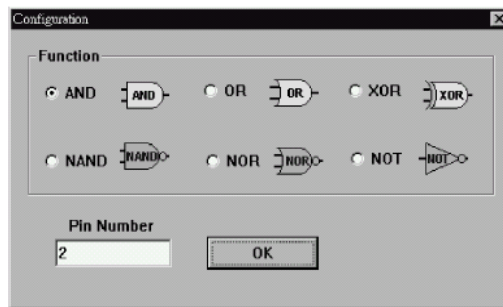


Fig 13-14

Select a logic gate using the add function. If the logic gate that is displayed is not the required one, pressing the right key on the mouse will display a range of logic gates as shown in the figure. The desired logic gate can then be selected. The Pin Number input area determines the number of input pins to each gate. The value set here is reflected in the number of pins available in the connect dialog box.

→ **Matrix key**

The Matrix key provides a standard matrix key peripheral device, the size of which can be setup from the configuration dialog box. The debounce time can be set for the matrix switches with the units in milliseconds. Note that the columns of the matrix are either connected to VCC or VSS, an option which is set in the attribute dialog box of the matrix peripheral.

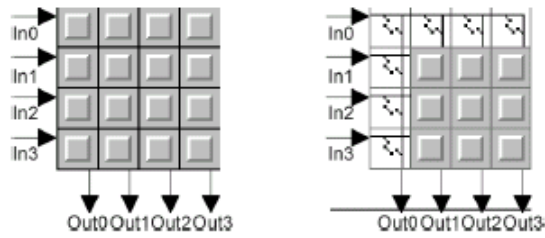


Fig 13-15

If, for example, the user sets up the matrix key with row = 4 and column =4, there will be 4 input pins or rows and 4 output pins or columns.

→ **Rectangle wave generator**



Fig 13-16

The rectangle wave generator is used to generate rectangular waves, the frequency of which is dependent upon the MCU frequency. In the attribute dialog box of this peripheral the cycle input dictates how many instruction cycles are required for an input waveform transition. If for example the cycle value is set to 2, then every 2 machine cycles the rectangular waveform generator input will toggle. The period of this input is therefore twice the cycle value. Note that if the rectangular wave generator is selected and the left key double clicked to display the connect dialog box, the generator can only connect to one device. However if the devices to be connected to are selected and their connect dialog box displayed then more than one device can be connected to the same wave generator. If more than one pin on the MCU is to be connected to the same wave generator then it is necessary to add further wave generators to achieve this.

Quick Start Example

From the examples provided in the Holtek IDE3000 User's Guide, one has been chosen as a practical example to illustrate how to construct a virtual external circuit.

Scanning Light

→ **From within the HT-IDE3000 System**

- Create a new project and select the HT48C10-1 MCU (Project/New)

- Add the source file scanning.asm to the project (Project/Edit) The file can be found in the Holtek IDE\SAMPLE\IO
- Change the Holtek IDE to simulation mode.(Options/Debug/Mode)
- Build the project.(Project/Build)

→ **From within the VPM**

- Create a new VPM project.
- Add 8 LEDs to the project by repeatedly clicking the Add button and selecting LED 8 times
- Add a resistor to the project - click the Add button and select RESISTOR just added and double click the mouse left button - then setup the resistor s name with VCC
- Connect all of the LED anode pins to VCC and connect all of the LED's cathode pins to bit n of PA on the MCU (n=0-7). The following shows how to connect LED_0's anode to VCC and its cathode to bit0 of PA on the MCU
 - Click the mouse left button on LED_0 to select it
 - Click the mouse right button on LED_0 to display the connect dialog box as shown as Fig 13-18
 - Connect the cathode of LED_0 to PA bit0 on the MCU
 - Repeat the above to setup all other LED_n connections
- Push the Mode button to change the VPM mode from configuration mode to running mode

→ **From within the HT-IDE3000**

Start the debug operations — the output results for the LEDs will be shown in the VPM window.

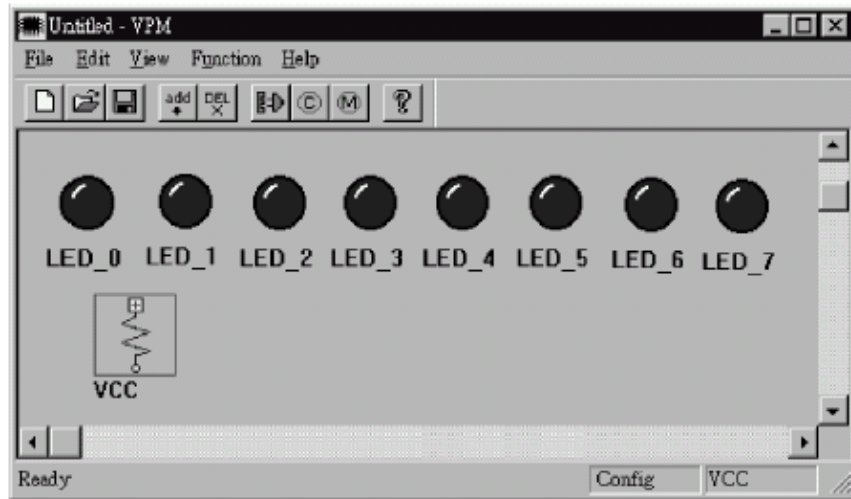


Fig 13-17

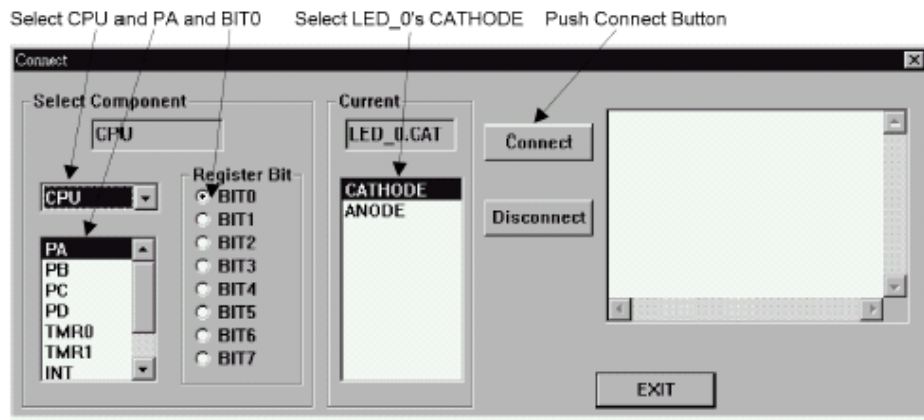


Fig 13-18

Chapter 14

Hi-Tech C MCU Converter

14

Hi-Tech C MCU Converter Function

This is a Holtek MCU plug-in for the Hi-Tech C compiler environment which can support the newer Holtek MCUs.

Using the Hi-Tech C MCU Converter

→ **Execute the Hi-Tech C MCU Converter Program**

First select the main Holtek MCU Development Tools area, and then select the Hi-Tech C MCU Converter as shown in Fig. 14-1.

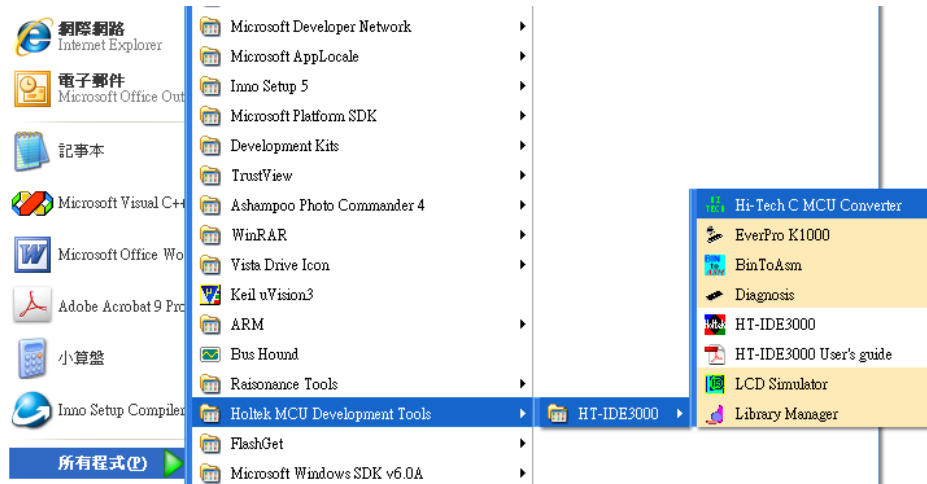


Fig 14-1

→ **Setup the HT-IDE3000 and HI-TECH shortcut**

The user needs to install the HI-TECH C for the Holtek MCU software. Then select the HT-IDE3000 and HI-TECH install shortcut as shown in Fig. 14-2. Then select Merge to activate the converter after which a successful convert message will be shown.

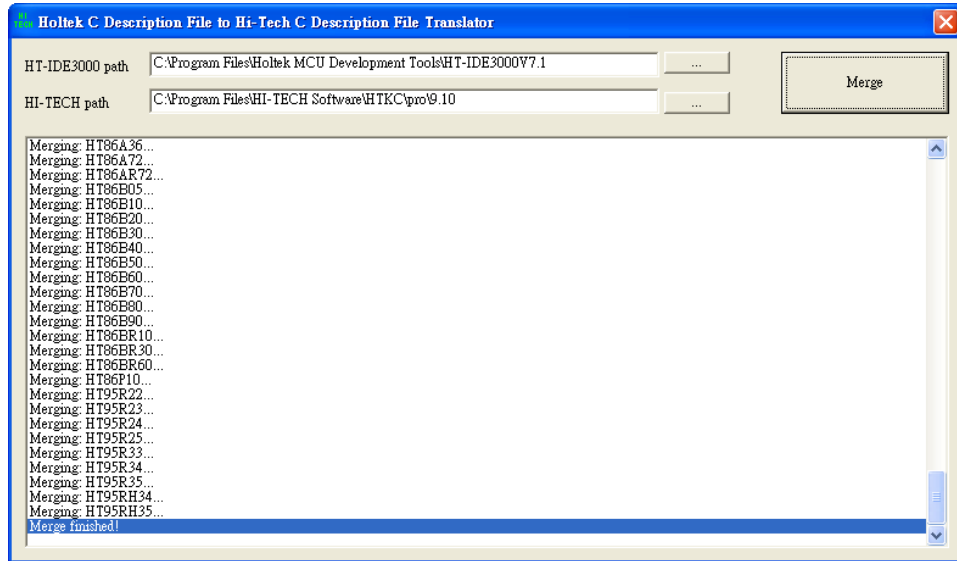


Fig 14-2

Part IV

Appendix

Appendix A



Reserved Words

Used By Cross Assembler

Reserved Assembly Language Words

The following table lists all reserved words used by the assembly language.

- Reserved Names (directives, operators)

\$	DUP	INCLUDE	NOT
*		LABEL	OFFSET
+	ELSE	.LIST	OR
-	END	.LISTINCLUDE	ORG
.	ENDIF	.LISTMACRO	PAGE
/	ENDM	LOCAL	PARA
=	ENDP	LOW	PROC
?	EQU	MACRO	PUBLIC
[]	ERRMESSAGE	MESSAGE	RAMBANK

AND	EXTERN	MID	ROMBANK
BANK	HIGH	MOD	.SECTION
BYTE	IF	NEAR	SHL
DB	IFDEF	.NOLIST	SHR
DBIT	IFE	.NOLISTINCLUDE	WORD
DC	IFNDEF	.NOLISTMACRO	XOR

- Reserved Names (instruction mnemonics)

ADC	HALT	RLCA	SUB
ADCM	INC	RR	SUBM
ADD	INCA	RRA	SWAP
ADDM	JMP	RRC	SWAPA
AND	MOV	RRCA	SZ
ANDM	NOP	SBC	SZA
CALL	OR	SBCM	TABRDC
CLR	ORM	SDZ	TABRDL
CPL	RET	SDZA	XOR
CPLA	RETI	SET	XORM
DAA	RL	SIZ	
DEC	RLA	SIZA	
DECA	RLC	SNZ	

- Reserved Names (registers names)

A	WDT	WDT1	WDT2
---	-----	------	------

Instruction Sets

Arithmetic Instructions

ADD A,[m]	Add Data Memory to ACC
ADDM A,[m]	Add ACC to Data Memory
ADD A,x	Add immediate data to ACC
ADC A,[m]	Add Data Memory to ACC with carry
ADCM A,[m]	Add ACC to Data Memory with carry
SUB A,x	Subtract immediate data from ACC
SUB A,[m]	Subtract Data Memory from ACC
SUBM A,[m]	Subtract Data Memory from ACC with result in Data Memory
SBC A,[m]	Subtract Data Memory from ACC with carry
SBCM A,[m]	Subtract Data Memory from ACC with carry and result in Data Memory
DAA [m]	Decimal adjust ACC for addition with result in Data Memory

Logic Operation Instructions

AND A,[m]	AND Data Memory to ACC
OR A,[m]	OR Data Memory to ACC
XOR A,[m]	Exclusive-OR Data Memory to ACC
ANDM A,[m]	AND ACC to Data Memory
ORM A,[m]	OR ACC to Data Memory
XORM A,[m]	Exclusive-OR ACC to Data Memory
AND A,x	AND immediate data to ACC
OR A,x	OR immediate data to ACC
XOR A,x	Exclusive-OR immediate data to ACC
CPL [m]	Complement Data Memory
CPLA [m]	Complement Data Memory with result in ACC

Increment & Decrement Instructions

INCA [m]	Increment Data Memory with result in ACC
INC [m]	Increment Data Memory
DECA [m]	Decrement Data Memory with result in ACC
DEC [m]	Decrement Data Memory

Rotate Instructions

RRA [m]	Rotate Data Memory right with result in ACC
RR [m]	Rotate Data Memory right
RRCA [m]	Rotate Data Memory right through carry with result in ACC
RRC [m]	Rotate Data Memory right through carry
RLA [m]	Rotate Data Memory left with result in ACC
RL [m]	Rotate Data Memory left
RLCA [m]	Rotate Data Memory left through carry with result in ACC
RLC [m]	Rotate Data Memory left through carry

Data Move Instructions

MOV A,[m]	Move Data Memory to ACC
MOV [m],A	Move ACC to Data Memory
MOV A,x	Move immediate data to ACC

Bit Operation Instructions

CLR [m].i	Clear bit of Data Memory
SET [m].i	Set bit of Data Memory

Branch Instructions

JMP addr	Jump unconditionally
SZ [m]	Skip if Data Memory is zero
SZA [m]	Skip if Data Memory is zero with data movement to ACC
SZ [m].i	Skip if bit i of Data Memory is zero
SNZ [m].i	Skip if bit i of Data Memory is not zero
SIZ [m]	Skip if increment Data Memory is zero
SDZ [m]	Skip if decrement Data Memory is zero
SIZA [m]	Skip if increment Data Memory is zero with result in ACC
SDZA [m]	Skip if decrement Data Memory is zero with result in ACC
CALL addr	Subroutine call
RET	Return from subroutine
RET A,x	Return from subroutine and load immediate data to ACC
RETI	Return from interrupt

Table Read Instructions

TABRDC [m]	Read ROM code (current page) to Data Memory and TBLH
TABRDL [m]	Read ROM code (last page) to Data Memory and TBLH

Miscellaneous Instructions

NOP	No operation
CLR [m]	Clear Data Memory
SET [m]	Set Data Memory
CLR WDT	Clear Watchdog Timer
CLR WDT1	Pre-clear Watchdog Timer
CLR WDT2	Pre-clear Watchdog Timer
SWAP [m]	Swap nibbles of Data Memory
SWAPA [m]	Swap nibbles of Data Memory with result in ACC
HALT	Enter Power Down Mode