Raisonance Tools for the 8051 family

Getting Started

Document version
16 October 2012

Contents

1. Introduction

The Raisonance 8051 Development Kits are a complete solution to creating software for the 8051 family of microcontrollers. The Development Kits comprise many different tools that allow projects ranging from simple to highly complex to be developed with relative ease.

Raisonance has been developing embedded tools since 1988 and has built up many years of experience. You will find that with the Raisonance Development Kits you can rely on tools that have been tested by real users over a long period of time.

The Raisonance integrated development environment (Ride7) is designed to develop 80C51 projects from beginning to end. It provides the tools necessary to build 8051-based projects:

- Compile chain (Assembler, C compiler and linker) for building applications.
- Software simulator for validating applications.
- Hardware debugger for debugging with the RLink USB dongle.

1.1 Purpose of this manual

This manual has been written to introduce the first time user to the Raisonance Development Kits and guide them through many of the features available. With the aid of this manual users should be able to quickly understand the tools, how they interact with each other and how to start developing their own projects.

1.2 Scope of this manual

This manual does not cover the features in great detail or cover advanced features, but provides a familiarity to the tools that will provided a basis for using more complex features.

It is assumed that the user is familiar with Windows and has at least some familiarity with the 8051 microcontroller family and the C programming language.

Please see Raisonance's relevant application notes for details of the hardware used for debugging.

1.3 Additional help or information

If you want additional help or information, if you find any errors or omissions, or if you have suggestions for improving this manual, go to the KEOLABS' site for Raisonance microcontroller development tools www.raisonance.com, or contact the microcontroller support team.

Microcontroller website:	www.raisonance.com
Support extranet site:	support-raisonance.com (software updates, registration, bugs database, etc.)
Support Forum:	forum.raisonance.com/index.php
Support Email:	support@raisonance.com

1.4 Raisonance brand microcontroller application development tools

January 1, 2012, Raisonance became the brand under which the company KEOLABS sells its microcontroller hardware and software application development tools.

All Raisonance branded products regardless of their date of purchase or distribution are licensed to users, supported and maintained by KEOLABS in accordance with the companies' standard licensing maintenance and support agreements for its microcontroller application development tools. For information about these standard agreements, go to:

Support and Maintenance Agreement:	http://www.raisonance.com/warranty.html
End User License Agreement:	http://www.raisonance.com/software-license.html

2. Development tools overview

The following is a list of the tools included in the Development Kit with a short overview of each one:

Tool	**Overview**
ANSI C compiler RC-51	ANSI compliant compiler that takes source files and generates object files. Extensions to the C language are used to enable features of the microcontroller to be used or controlled.
Assembler MA-51	Takes source files written in assembler and generates object files. Controls are included to enable features of the microcontroller to be used or controlled.
Linker/Locator LX-51	Combines the object files generated by the Compiler and Linker and produces a different kind of object file. The Linker also decides where certain types of Data and Code are located in memory.
Object-to-HEX Converter	Converts an object file generated by the linker and generates an Intel Hex file, compatible with most device programmers.
Ride7	Integrated development environment which is the interface for all the other tools. Provides an editor, a project manager (no need for a Makefile) and a debug user interface that can be used either with the simulator or with most available hardware-debugging tools. It can be used by several microcontroller families, including 80C51, STM8/ST7, ARM and more. The simulator simulates the core (including the entire memory space) and most peripherals. Complex peripherals (USB, CAN) and some less common peripherals are not simulated.
Library Manager LIB-51	Takes object files generated by the Compiler or Assembler and creates a library to be included in other projects.

2.1 Conventions used in this manual

File | New	Refers to the menu item "New" on the File menu.
while(1);	(bold, monospaced type) User input.
filename	Replace the italicized text with the item it represents.
[ ]	Items inside [and] are optional.

2.2 Other tools

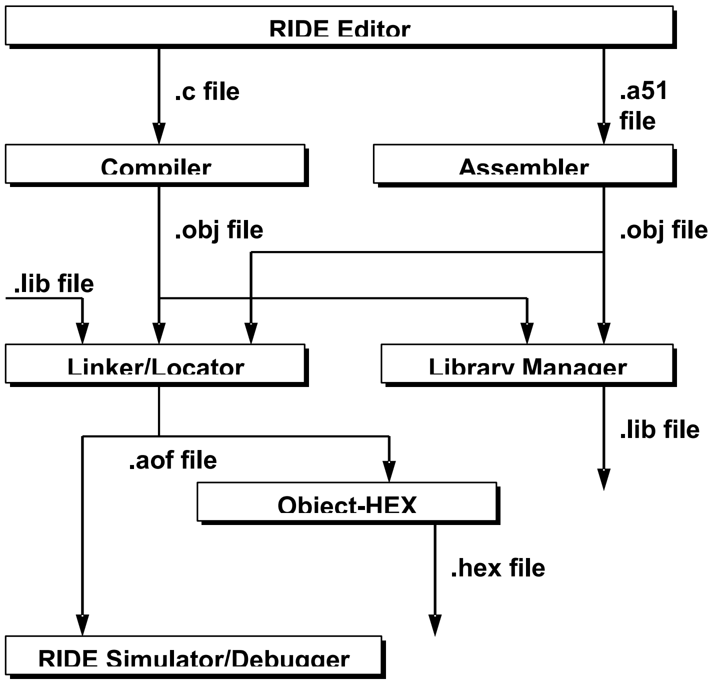
The development kit supports a number of tools from Raisonance:

- CodeCompressor: Post link optimization tool that can reduce code size. It accepts as input any executable code, whether from assembler, C or any other source (libraries...).
- RLink support: Ride7 can communicate with the RLink, which is a USB hardware dongle that allows the user to program the 8051 on an application board and debug the application while it is running on the 8051. It uses the ICC or SWIM protocol. For more information refer to chapter "Debugging with hardware tools".
- KR-51: A real time kernel for the 8051 microcontroller family. Placed in the heart of an application, it provides efficient task scheduling. It recognizes tasks written in either assembly, PL/M-51 or the C-51 language.

Each tool mentioned above has a dedicated user manual (please refer to the appropriate manual for more details).

2.3 The relationship between the tools

In addition to being an editor, simulator and debugger, Ride7 also controls and automates the entire build process. By selecting a single menu item, Ride7 will execute the correct tools to generate either a library file or an Absolute Object File (.aof) and Intel HEX File.



This diagram shows the relationship between the tools.

1. Ride7 provides an editor which allows the user to generate C source files (.c extension) and Assembler source files (.a51 extension).
2. Each source file is translated using the appropriate tool. The Compiler translates C source files. The Assembler translates assembler source files. Each tool generates a relocatable object file (.obj extension) . If a project has more than one C source file or more than one Assembler source file, then the Compiler and Assembler are executed as many times as required.
3. If a library file is being generated then the Library Manager takes all the relocatable object files and combines them into a library file (.lib extension). The library file may then be linked in with other projects.
4. The Linker/Locator takes relocatable object files and library files and links them together resolving external references. The Linker/Locator then locates variables and code to specific addresses in the memory map. The Linker/Locator generates a single .aof. It also generates the same file with no extension.
5. The Absolute Object File may then be used by the simulator or debugger in Ride7, as the file can contain debugging information. Alternatively the .aof may be used with In-Circuit Emulators.
6. The Object-HEX converter tool converts a .aof into an Intel HEX file (.hex extension) which is a representation of the pure binary code generated, without debugging information. The Intel HEX File is accepted by virtually all device programmers.

Note: Each relocatable object file is referred to as a module. Each module must have a unique name. For example the source file *foo.c* generates the relocatable object file *foo.obj*. The module name is therefore “`foo`”. However the source file *foo.a51* also generates the relocatable object file *foo.obj*. The result is two modules with the same name. Therefore each source file must have a unique name, regardless of whether it is a C source file or an Assembler source file.

2.4 Listing files

Some of the tools generate text files, collectively referred to as listing files, in addition to the files shown in the diagram. These listing files aid the user in understanding how the tools processed the input files and in tracking down problems.

The Compiler and Assembler generate a listing file (.lst extension) for each source file they translate. The listing file contains such information as the assembly code generated, a symbol table, the memory requirements of the module and how the tool was invoked.

The Linker generates a listing file commonly referred to as the map file (.m51 extension). This file will be referred to as the map file in the rest of this manual. The map file contains a list of input modules and libraries, a memory map of the project, a summary of the memory requirements, a call tree and a symbol table. Only one map file is generated as the Linker is only executed once.

2.5 Summary of file extensions

The following table provides a summary of the file extensions used.

File	File Extension
Project	.prj
Compiler source file	.c
Assembler source file	.a51
Compiler and assembler object files	.obj
Compiler and assembler listing files	.lst
Linker object file	.aof
Linker map file	.m51
Intel hex file	.hex
Library file	.lib

Note: The file extensions used may be modified or additional file extensions supplied.
In Ride7 choose **Options | Tools** then select the relevant tool and click on **Edit**. Fields are provided to enter the input and output file extensions.

2.6 Supported derivatives

The development kit supports most of the existing 80C51 derivatives. An up-to-date list of supported derivatives and software simulation limitations can be seen in Ride7 when creating a new 80C51 project (**Project | New project**): browsing through the list of available devices displays some detailed characteristics in the "description field".

2.7 Installing Raisonance Tools for 8051

When installing the development kit, make sure you have the latest software version from the Raisonance website: http://www.raisonance.com/download/

If the software is being installed from CD then the installation program should automatically run when the CD is inserted. If the CD autorun feature is turned off or you have downloaded the software from a web site then the software may be installed simply by running *install.exe*.

The following is the directory structure placed into the installation folder

Folder	**Description**
BIN	Contains executable files and associated library files that are required by the tools.
DOC	Contains manuals for each of the tools and for various evaluation boards.
EXAMPLES	Contains example projects for use with Ride7. The examples are subdivided into categories.
HELP	Contains the on-line help files used by Ride7.
INC	Contains include files that may be used by users in projects. Some of the include files define Special Function Registers for various derivatives. Include files for the standard C libraries may also be found in this folder. In addition some of the standard C library functions, such as memory allocation and low-level I/O and the startup code may be found in the Sources sub-folder. This folder is divided into different sub-folder corresponding to each 8051 manufacturers.
LIB	Contains libraries that the Compiler and Linker/Locator may use during compilation of a source file or linking of a project. The libraries include such things as routines to perform mathematical operations on floating point numbers, and the standard C library functions.

Note: The compiler will look in the INC folder for header files. If the Special Function Register header file for the particular device you are using is not present in the INC folder then you can easily create your own and place it in the INC folder.

1. Find the header file of the device that is closest to the device you are using.

2. Copy it and rename the copy to the name of the device.

3. Using the device datasheet, add in the missing Special Function Register declarations using the same format as shown in the file.
4. Include the header file in your C source files inside "<" and ">", for example #include <reg999.h>

2.8 Example projects

Example projects written in C or assembler are ready to run.

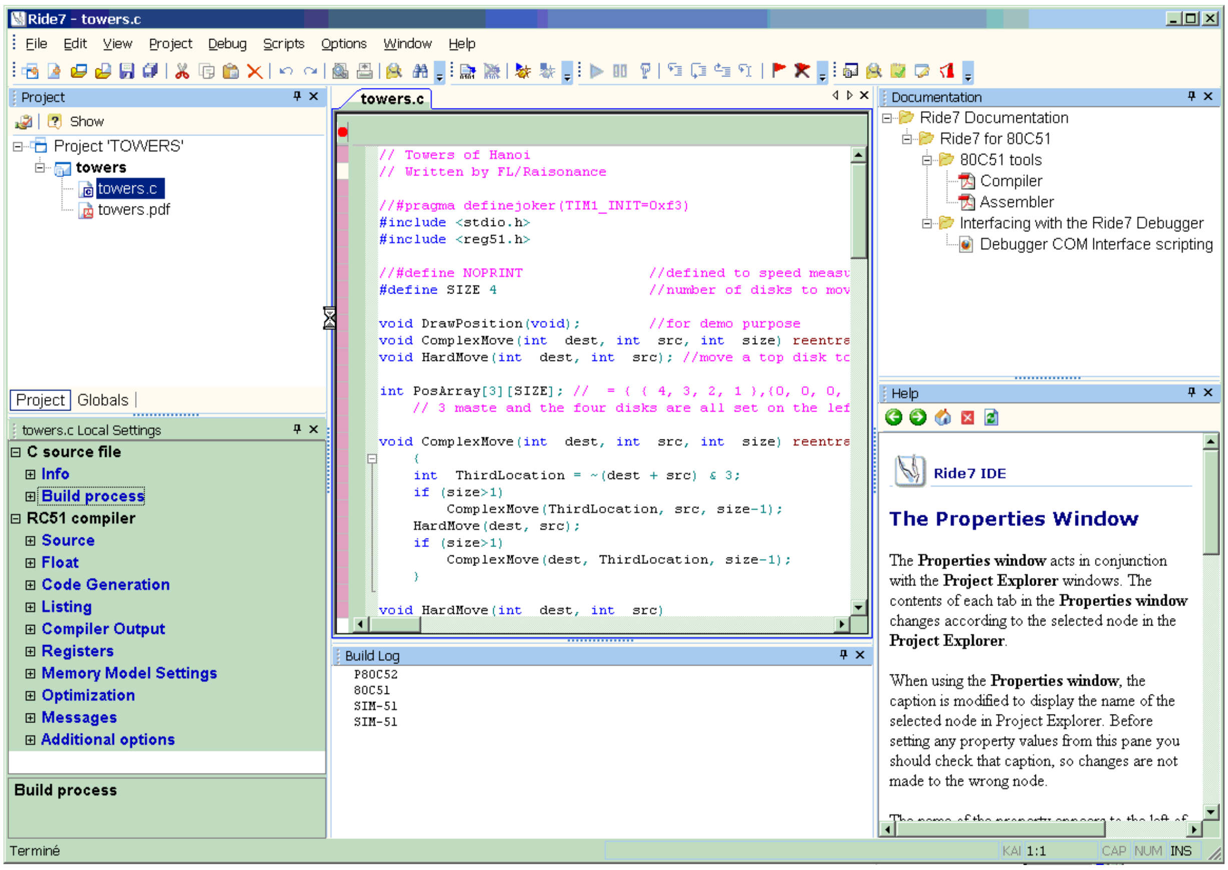
Each example is described in the comments in the application's source files.

To open a project, for example Towers:

- Click on **Project | Open Project** in **Ride7.**

- Click on **Towers.rprj** and **Open**

Towers opens. In this example project, you can modify the debug values then build to view the result.



```
// Towers of Hanoi
// Written by FL/Raisonance

//#pragma definejoker(TIM1_INIT=0xf3)
#include <stdio.h>
#include <reg51.h>

//#define NOPRINT                 //defined to speed measu
#define SIZE 4                    //number of disks to mov

void DrawPosition(void);          //for demo purpose
void ComplexMove(int  dest, int  src, int  size) reentra
void HardMove(int  dest, int  src); //move a top disk tc

int PosArray[3][SIZE]; //  = { { 4, 3, 2, 1 },{0, 0, 0,
    // 3 maste and the four disks are all set on the lef

void ComplexMove(int  dest, int  src, int  size) reentra
    {
    int  ThirdLocation = ~(dest + src) & 3;
    if (size>1)
        ComplexMove(ThirdLocation, src, size-1);
    HardMove(dest, src);
    if (size>1)
        ComplexMove(dest, ThirdLocation, size-1);
    }

void HardMove(int  dest, int  src)
```

```
P80C52
80C51
SIM-51
SIM-51
```

The Properties Window

The **Properties window** acts in conjunction with the **Project Explorer** windows. The contents of each tab in the **Properties window** changes according to the selected node in the **Project Explorer**.

When using the **Properties window**, the caption is modified to display the name of the selected node in Project Explorer. Before setting any property values from this pane you should check that caption, so changes are not made to the wrong node.

2.9 Ride6 project import

To import a Ride6 project:

- Click on **Project | Open Project** in **Ride7**.
- Change the **file type** to Ride6 projects (*.prj)
- Select the project to import and **Open**

Ride7 reads the Ride6 project and creates a Ride7 project (*.rprj). The applications are opened as Ride7 applications (*.rapp) and therafter any changes are done to the Ride7 files.

3. Register the new Raisonance tools for 8051

The new Ride7 and RKit-51 must be registered to operate correctly. Registration requires a Raisonance software product license that is under a valid support contract (a standard support contract expires one year after the date of purchase).

To activate the software you must first register your RKit- 51 using your RKit- 51 license (Serial number or Dongle).

Unregistered software functions for a 7-day evaluation period with full features of the Enterprise version. After 7 days the software can no longer be used. If this occurs, contact info@raisonance.com.

3.1 Install and activate the new software

Perform these steps to install your new software:

1. Remove old versions of Ride7 and RKits.
2. Install the new Ride7 software, then the RKit- 51 software.
3. Launch **Ride7** and if the Serialization Choice popup does not appear, select **Help > License**. Activate your software by providing one of the following:
   a. Software serial number (purchasers of RKit-xxx-Enterprise or RKit-xxx-Lite tool sets).
   b. Hardware dongle (purchasers of RKit-xxx-Enterprise or RKit-xxx-Lite tool sets).
   c. RLink serial number (this option is not applicable to the RKit-51).
4. Register the software by selecting the menu item **Help > License**... and follow the instructions.

Note: During the registration process, you will be notified if your product's support contract has expired. This notification includes information about how to renew your tool's support contract.

You can verify that registration was successful by closing and restarting Ride7, then checking in the **About Ride7** window that you are not in "Demonstration version" for RKit or Ride7. Validate its operation (test compilation, connection to RLink and target CPU, etc.). If you have any problems please contact technical support (***support-mcu@raisonance.com***).

3.2 Register using a serial key

You can use this procedure for RKit-STM8-Enterprise software licenses (Serial Number or Dongle).

If your product was purchased:

- less than one year ago, you will need to provide your proof of purchase.
- more than one year ago, you will need to purchase the (annual) support extension.

Perform these steps to register using a serial key:

1. Contact Raisonance (info@raisonance.com) to obtain a **Serial Key**. They need your Software Serial Number.
2. Log on as admin, ensure you have internet access, a working default browser and a working email address.
3. Open Ride7 or RFlasher7. If it does not open automatically, click **Help->About Ride7**.
4. Click the **Change License** button. Select the **Serial Activation** method, click **Next**.
5. Enter the **Serial Key** provided to you. Click **Next**.
6. Connect to the internet and click **Get Activation code online** to open your browser on the Raisonance server.
7. Fill in the form and click **Generate and Send Activation code**.
8. Check your server for an email from Raisonance Support team with the **Activation Code**.

9. Copy the **Activation Code** from the email to the Ride7 window. Click **Finish**.

3.3 Register using a dongle

1. Launch Ride7 or RFlasher7.
2. Connect your USB dongle to a USB port on the PC (make sure the LED lights up, otherwise you must install the driver).
3. Select Help > License...
4. Select Dongle activation, click on Next. Ride7 detects your hardware and reads its serial number.
5. Click on Get Activation code online. This opens a browser window to the Raisonance Support extranet. In this form:
    a. Confirm your username and email.
    b. Click on Generate and Send Activation code. An e-mail with the Activation code is sent to you automatically.
    c. Copy and paste the Activation code into the field provided in Ride7.
    d. Click on Close.

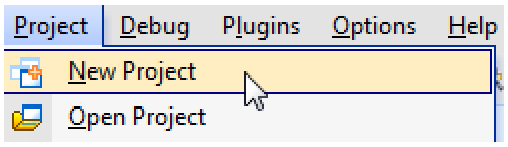
4. Build a new project

4.1 Start Ride7

Starting Ride7 is very easy. Select **Start | Programs | Raisonance Tools | Ride7**.
After a few moments the main window opens.

4.2 Creating a new project

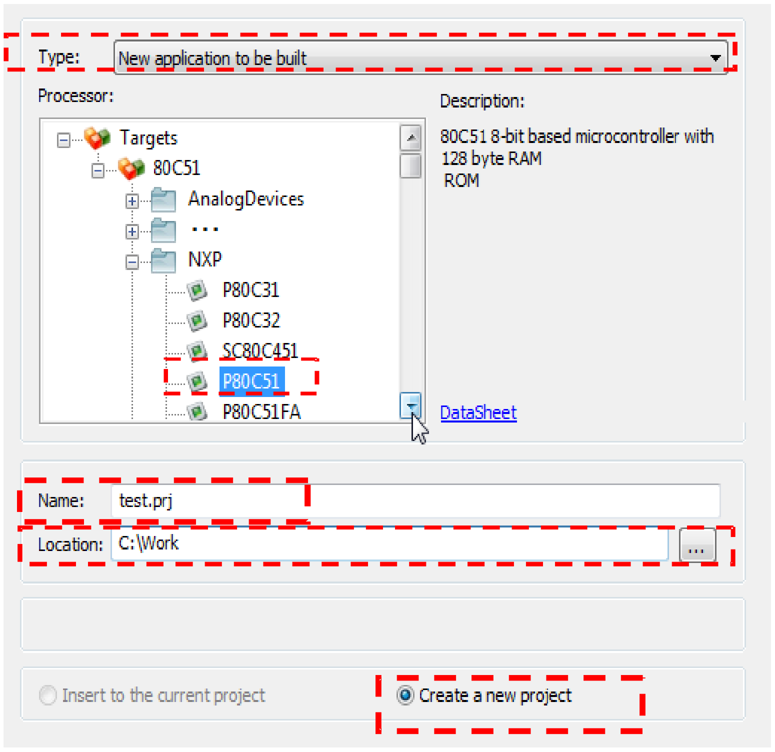
1. Click on the **Project** tab in the Ride7 menu.
2. In the drop down menu, click on **New Project**.



4.3 Creating the settings for the new application

To create the settings:

1. Select application **Type**.
2. Select the type of **Processor**
3. Create a project **Name**
4. Select a directory **Location**
5. Select the radio button **Create a new project**.
6. Press **Finish**.
7. The Ride7 window appears.



Type: New application to be built

Processor:

Description:

80C51 8-bit based microcontroller with 128 byte RAM ROM

Name: test.prj

Location: C:\Work

( ) Insert to the current project [✓ Create a new project]

The Project window acts like a project manager, showing which source files are in the project and giving instant access to each one in both the editor and debugger.

This entry represents the project as a whole.

A .aof file is the result of building the project.

Note: The .aof file always takes its name from the name of the project.
In this case the project is called “Test” so the .aof file will be called “test.aof”.
Likewise the generated Intel hex file will also be named after the project (“test.hex”).

4.4 Create and add a source file

The next step we will take is to create a new, basic source file and add it to the project. The following section will show how to build the project. We will then know that all the tools are working and you will then have a starting point for all future projects.

To create a new source file choose **File | New | Source Files**.

Enter the following into the new window:

```
void main(void)

{

  while(1);

}
```

To save the source file:

- Choose **File | Save**. A standard Save As window will open.
- Enter *main.c* into the **Filename** field.
- Click on **Save**.

To add the file to the project:

- Right click on *test.prj* in the Project window.
- Select **Add I Item**
- Select *main.obj* in the File Open window.
- Click on **Open**.

4.5 Build the project

To build the project simply click on the **Play All** project button on the toolbar (or **Project | Build Project**).

Once the project has been built, the Make window will show the result of the build process in tree form.

Note: If the Compiler or Linker generated any warnings or errors then it is possible to view them in the Make window. In the case of Compiler warnings or errors, double clicking on them in the Make window will take you to the relevant point in the source code.

⚠ **Warning:** Once the project is generated, there is no way to go back to modify one option and generate it again. If you find out that you were mistaken about an option, then you have to restart the project creation process from the beginning.

4.6 Add your application code

Before we take a look at the simulator we need to add some more code. Simulating an infinite loop is not very exciting, unless you happen to like infinite loops a great deal. The Raisonance compiler features language extensions that allow aspects specific to microcontrollers to be used in C. One of those language extensions gives the ability to declare Special Function Registers (SFRs) so they may be read from and written to.

1. To save you from entering the SFR declarations every time you create a new project they are commonly placed in header files, with one header file per derivative. You can examine the contents of this header file to see how SFRs are declared using the language extensions.
   a. Enter the following line at the top of the main.c source file before the main function

   ```
   #include <reg51.h>
   ```

   b. Select the header file name (for example “reg51.h”) with the mouse,
   c. Right click and a menu will pop-up, choose **Open**:
2. Below the #include add the following line:

   ```
   unsigned char counter = 0;
   ```

3. We will add two functions. The first, called `init`, initializes a timer and the timer interrupt. The second function, called `timerisr`, is the Interrupt Service Routine for the timer. The code for both these functions is different for all three microcontroller families, so please refer to the correct section below for the tools you are using. Add the following code before the main function, but after the line you just added declaring the counter variable:

   ```
   void timerisr(void) interrupt 1

   {

     TF0 = 0;    // clear overflow flag

     counter++;

   }

   void init(void)

   {

     TMOD = 0x02;      // 8-bit auto-reload timer

     ET0 = 1;          // enable timer interrupt

     EA = 1;           // global interrupt enable

     TR0 = 1;          // run timer

   }
   ```

4. Add the following line inside the main function, just before the `while(1);`

   ```
   init();
   ```

5. Save the source file by choosing **File | Save** or clicking on the **Save** button.
6. Build the project by choosing **Project | Make Project** or clicking on the **Make All** button
7. If “+” signs appear next to the Compiler or Linker items in the Make window, click on the “+” sign to expand the tree and view the warnings. Double-click on the Compiler warnings to jump to the relevant point in the source code and fix the problem.

Once you have the project successfully built we are ready to start the debugger.

4.7 Automatic relocation of global variables

80C51 architecture allows placement of data in “data” memory space, which gives much more efficient code. The data memory space is very small (typically 128 bytes of data), and for a large project it is not always obvious which variables should be located in this memory space.

The RC51, in coordination with the RL51 linker, allows automatic selection of which Xdata **global** variables should be assigned to it. Depending on their size and their usage count within the whole application, the most used variables will be relocated to data, producing the smallest code.

As the linker only deals with Xdata variables, the Auto Relocation process is only available for Large and Huge memory models where variables without dedicated space are located in Xdata.

☺**Tip: Reduce your code footprint using Auto Relocation mode**

The “Auto Relocation mode” is a project-wide optimization that can help reduce your code footprint. Make sure you activate Auto Relocation mode for your large projects, in order to automatically benefit from this feature.

To select the "*Auto Relocation mode*":

1. Ensure your project is functional (compiles and links without error).
2. In Ride7, select **Project | Properties**.
3. Select **Memory Model Settings | Memory Model | Large or Huge**
4. Select **80C51 Options | 80C51 tools Sets | Automatic variables relocation**; set it to **active.** This performs (upon next build) a specific (and silent) compilation and link pass that produces the "best list" of variables to be relocated to the zero page. This list of relocated variables is passed to the compiler.
5. Subsequent builds of the project automatically relocate the variables according to the produced “Auto Relocation File”.

4.7.1 Internal workings of the Auto Relocation mode

During the first build, the linker evaluates the amount of data memory space still available after linking and provides a list of relocatable Xdata variables. These variables are selected because they have the highest number of accesses, relocating them in the “data memory space” produces the best code size reduction. This list is output by the linker in a text file called the AOF file, with an extra extension ".areloc" (for example: "myproject.aof.areloc").

During the second build, Ride forwards the list of variables to be considered as "data" to the compiler. This forced relocation is done using the SPACEDEF directive (see in the list of the 80C51 compiler directives).

Note: Local variables

Automatic relocation handles only **global** variables. Local variables are always relocated in the default memory class (Xdata for Large and Huge memory spaces).

☺**Tip: Reading your *.m51* file**

Read your *.m51* file (produced by the linker) to check for proper variables memory placement.

5. Debugging and simulating

The debug interface is used for both simulating and debugging. From a user interface point of view, basic debugging functions (stopping and resuming CPU execution, setting a breakpoint, single-stepping, checking memory and registers, etc.) are identical, whether you are using the simulator or a hardware debug tool. It is advisable to get familiar with the simulator before starting to work with the hardware debug tools.

You can select the tool that corresponds to your debug requirements from **Project Properties | 80C51 | Options | Debug environment | Debug tool**

5.1 Hardware debug

The 8051 development kit can be used with a number of hardware debug tools.

5.2 Simulation debug

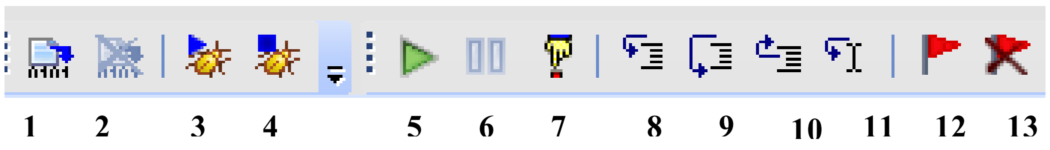
The 8051 development kit provides a simulator which lets you check your code, and the interaction between your code and the peripheral, before you debug with an in-circuit debugger or emulator.

5.3 Launch the debugger

Before running the debugger, you must configure the debugger interface.
To launch the debugger, type **CTRL-D**. If your project has not been built, it will be built automatically before the debugger launches. Otherwise, the debugger launches immediately.

5.4 Debugger toolbar

The debugging (software or hardware) is controlled by the debugger toolbar:



1. **Make**: Build the project (F9)
2. **Cancel Make**: Stop building the project
3. **Start debug session** (Ctrl D)
4. **Stop debug session** (Shift + Ctrl D)
5. **Run debug** (Ctrl F9)
6. **Pause debug**
7. **Reset**: Press this button to reset the application. (Ctrl F2)
8. **Step into**: On a function call in a line of the C source code, this button steps into the called function. If it is not a function call, it goes to the next line in the source code. (F7)
9. **Step over**: This button steps over a function call in a line of the C source code. (F8)
10. **Step out Exit functions** (Shift + F7)
11. **Run to** (Cntrl Q)
12. **Toggle breakpoint** (F5)
13. **Clear all breakpoints**

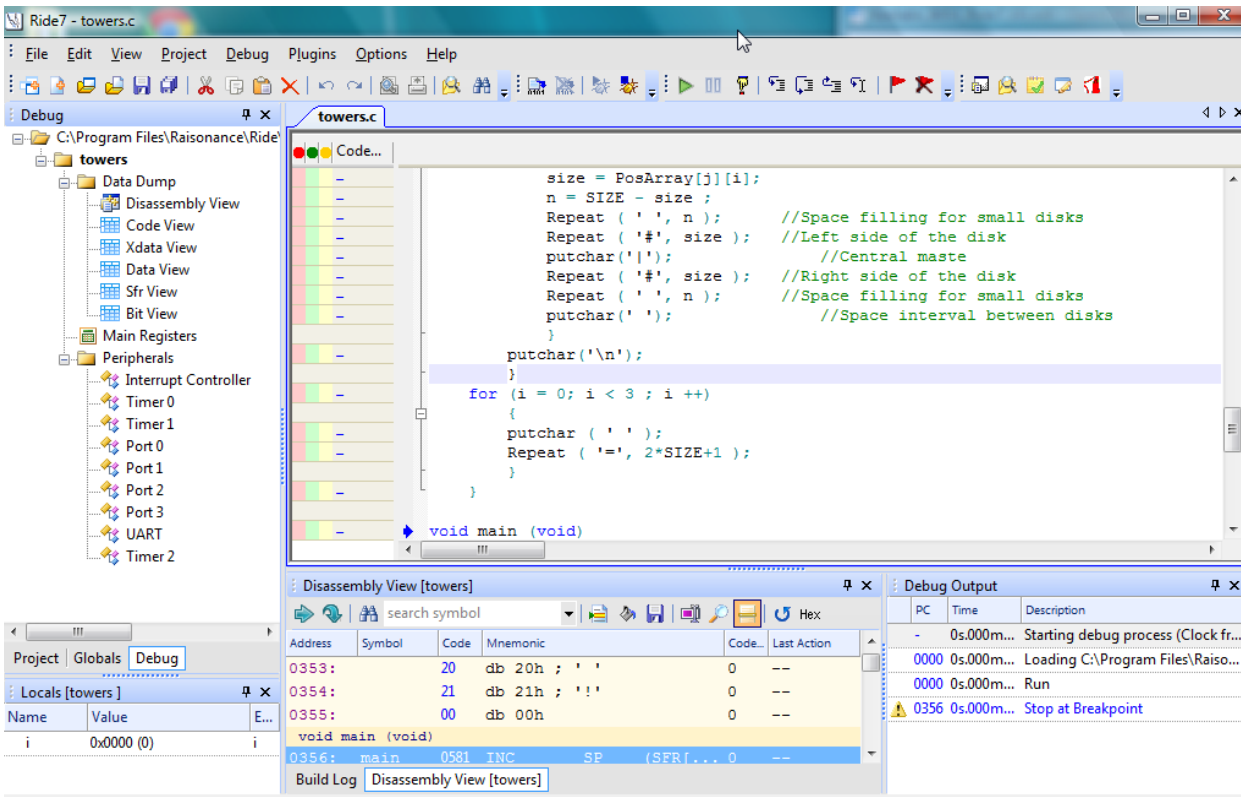
5.5 Ride7 window

Open Program files/Raisonance/Ride/Examples/80C51/General/C/Towers.
Your Ride7 window looks like the following:

1. **towers.c:** The source file as edited in C language or in assembly language.
   The Code window that shows you the instruction to be executed by the simulator.
2. **Disassembly View:** This shows an image of the code in the Flash memory of the target.
3. **Debug peripheral tree**.
4. **Project options window: Application** and **Advanced Options**.
5. **Debug Output**.
6. **Toolbar** which allows the user to control the simulation. (more information in the next section)

The following columns are available in the **Code window** -(Disassembly view):

- **Address:** The address where the instruction is located.
- **Symbol:** The name of the symbol, if a symbol is located at this address.
- **Code:** The byte-code located at this address.
- **Mnemonic:** The mnemonic corresponding to the byte-code.
- **Code Coverage:** The number of times the byte-code at this address has been executed.



```
            size = PosArray[j][i];
            n = SIZE - size ;
            Repeat ( ' ', n );       //Space filling for small disks
            Repeat ( '#', size );    //Left side of the disk
            putchar('|');                //Central maste
            Repeat ( '#', size );    //Right side of the disk
            Repeat ( ' ', n );       //Space filling for small disks
            putchar(' ');                //Space interval between disks
            }
        putchar('\n');
        }
    for (i = 0; i < 3 ; i ++)
        {
        putchar ( ' ' );
        Repeat ( '=', 2*SIZE+1 );
        }
    }

void main (void)
```

Locals [towers]

Name	Value	E...
i	0x0000 (0)	i

Disassembly View [towers]

```
Address  Symbol  Code  Mnemonic                Code...  Last Action
0353:            20    db 20h ; ' '            0        --
0354:            21    db 21h ; '!'            0        --
0355:            00    db 00h                  0        --
 void main (void)
0356:    main    0581  INC        SP   (SFR[... 0        --
```

Debug Output

	PC	Time	Description
	-	0s.000m...	Starting debug process (Clock fr...
	0000	0s.000m...	Loading C:\Program Files\Raiso...
	0000	0s.000m...	Run
⚠	0356	0s.000m...	Stop at Breakpoint

-

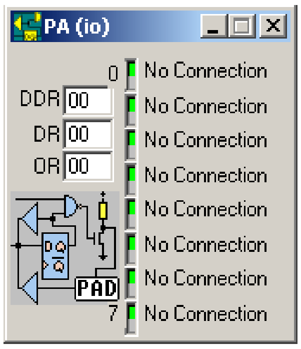
5.6 Peripheral status view

To view the status of a peripheral, you must open it by clicking on the corresponding item in the Debug peripheral tree.

For example, to simulate Port A (PA), double click on the **PA** icon in the Debug peripherals tree.

A **PORT A** view appears which shows the state of each pin of the port and lets you modify the registers:

- Green indicates a value of one and red a value of zero.
- By clicking on the LED it is possible to connect each pin of the port to a Net, to VCC, the Ground or no connection.



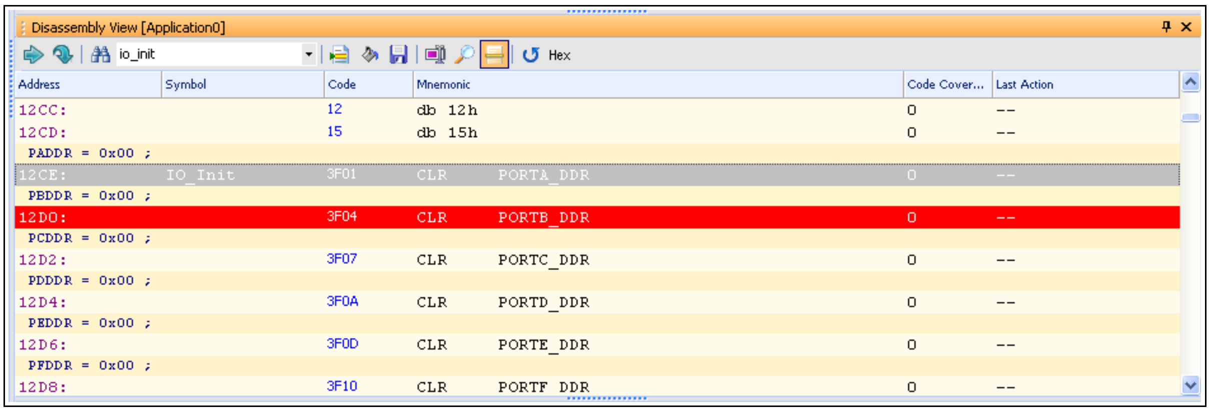
```
PA (io)
0   No Connection
DDR 00   No Connection
DR  00   No Connection
OR  00   No Connection
    No Connection
    No Connection
    No Connection
7   No Connection
```

5.7 Breakpoints

You can set breakpoints either in the source file or in the disassembly view.

5.7.1 Disassembly view

In this view, select the line on which you want to stop (the line becomes grey). Then click on the **Toggle Breakpoint** button (or F5) and the line becomes red. This means that a breakpoint has been set on this line.

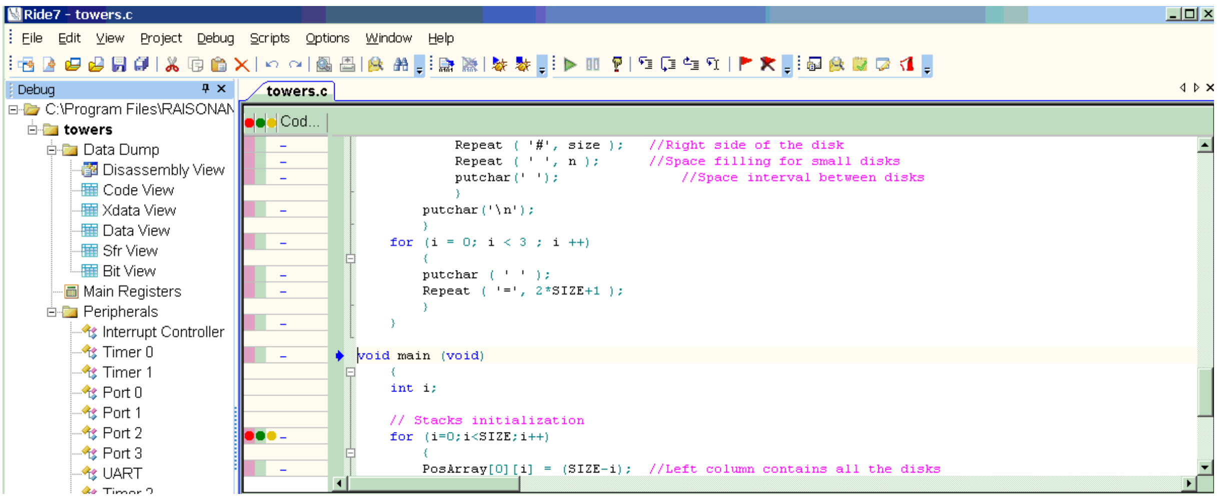


```
Disassembly View [Application0]
io_init   Hex
Address   Symbol    Code   Mnemonic                    Code Cover...  Last Action
12CC:               12     db  12h                     0              --
12CD:               15     db  15h                     0              --
 PADDR = 0x00 ;
12CE:     IO_Init   3F01   CLR      PORTA_DDR          0              --
 PBDDR = 0x00 ;
12D0:               3F04   CLR      PORTB_DDR          0              --
 PCDDR = 0x00 ;
12D2:               3F07   CLR      PORTC_DDR          0              --
 PDDDR = 0x00 ;
12D4:               3F0A   CLR      PORTD_DDR          0              --
 PEDDR = 0x00 ;
12D6:               3F0D   CLR      PORTE_DDR          0              --
 PFDDR = 0x00 ;
12D8:               3F10   CLR      PORTF_DDR          0              --
```

The application stops running when this line is reached and the line turns pink.

5.7.2 Source view

You can use the same procedure to set a breakpoint on a line of source code, or you can click on the pink square in the margin next to the instruction. When you click on the pink square, a red dot appears, indicating that a breakpoint has been set:



```
            Repeat ( '#', size );   //Right side of the disk
            Repeat ( ' ', n );      //Space filling for small disks
            putchar(' ');               //Space interval between disks
            }
        putchar('\n');
        }
    for (i = 0; i < 3 ; i ++)
        {
        putchar ( ' ' );
        Repeat ( '=', 2*SIZE+1 );
        }
    }

void main (void)
    {
    int i;

    // Stacks initialization
    for (i=0;i<SIZE;i++)
        {
        PosArray[0][i] = (SIZE-i);  //Left column contains all the disks
```

5.8 Watchpoints

Watchpoints display variable values permanently on the screen. To create one you must add the variable to the Watch window.

1. Open the Watch window by choosing **View | Debug Windows | View Watch**.
2. With the pointer over the Watch window press the right mouse button.
3. Choose **Add** from the menu that pops up.
4. Enter the variable into the expression field and click on **OK**

The Watch window will now show the current value of the variable in decimal, with the hexadecimal equivalent in parentheses.

If you click on the **GO** button to run to the next breakpoint, the value of variable turns red to indicate that it has changed.

6. Glossary

Term	Description
CodeCompressor	Post link code optimization tool
I/O	Input/Output
ICC	In-Circuit Communication
Ride7	Raisonance Integrated Development Environment
RLink	Hardware tool for in-circuit debugging and programming of a target microcontroller mounted on an application board. Supports interface via JTAG, ICC and SWIM protocols.
SWIM	Serial Wire Interface Module

7. Index

8. History

Date	Description
23 Mar 10	Initial version in new template for 80C51 with Ride7.
20 Jun 11	Added chapter about registering new Raisonance tools.
24 Jul 12	Modified cover page, final page and section 1.3 Additional help or information for KEOLABS
16 Oct 12	Updated screens and paths.