Raisonance Tools STM8/ST7

Getting Started

Document version
24 March 2017

Contents

1. Introduction

Ride7 is the Raisonance brand integrated development environment (IDE), designed for the development of ST7 and STM8 projects from beginning to end.

The RKit-STM8 plug-in for Ride7 provides the tools necessary to build ST7 and STM8-based projects:

- Compile chain (Assembler, C compiler and linker) for building applications.
- Software simulator for validating applications.
- Hardware debugger for debugging with an RLink and associated platforms (Open4, EvoPrimer, REva).
- RFlasher GUI and command-line interfaces for programming the Flash of ST7/STM8 devices through RLink during production.
- (ST7 only) RBuilder wizard to help users build their applications from scratch with minimum knowledge of the ST7 architecture and peripherals. This builder takes users through peripheral configuration and generates the necessary code in a start-up project.

1.1 Purpose of this manual

This guide should be used by anyone who is interested in building ST7/STM8 projects using Ride7.

1.2 Scope of this manual

It is assumed that the reader already has experience with at least one programming language, such as C, C++, JScript, VBScript or Python. This guide does not describe the basics of procedural programming such as functions, conditional branching and looping.

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 IoTize 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
Support Email:	support@raisonance.com

1.4 Raisonance brand microcontroller application development tools

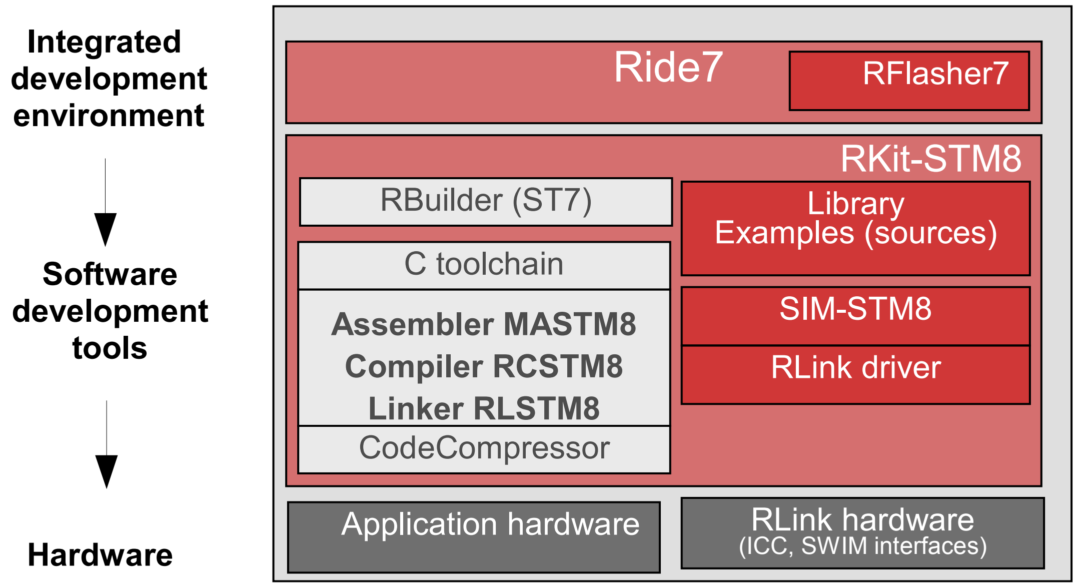
February 1, 2017, Raisonance became the brand under which the company IoTize sells its microcontroller hardware and software application development tools.

All Raisonance branded products regardless of their date of purchase or distribution are licensed to users, supported and maintained by IoTize 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. Ride7 and RKit-STM8 overview

RKit-STM8 defines the software and hardware tools for creating, compiling and debugging applications. For a full range of STM8 and ST7 microcontrollers.
The following table shows the set of tools that compose the RKit-STM8:



2.1 RKit-STM8

There are three versions of the RKit-STM8 (see Section 8 for details).

- **Basic** license includes programming and debugging of STM8/ST7 with no limitations. Compiler up to 2KB code. Activated using RLink/REva/Open4/EvoPrimer.
- **Lite** license includes programming and debugging of STM8/ST7 with no limitations. Compiler up to 32KB code. Activated using Serial Key, Dongle.
- **Enterprise** license includes programming, debugging and compiling of STM8/ST7 with no limitations. Activated using Serial Key, Dongle.

RKit-STM8 comes supplied with a number of ST7/STM8-specific tools from Raisonance:

- 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 ST7/STM8, ARM and uPSD.
- Compile Chain: Raisonance C toolchain, composed of C Compiler, Assembler and Linker which allows you to write applications in C and/or assembler.
- Simulator: Simulates core (including the entire memory space) and most peripherals. Complex peripherals (USB, CAN) and some less common peripherals are not simulated.
- CodeCompressor: A 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...).
- RBuilder: A GUI that configures the ST7 peripherals used by the application, and generates the corresponding source code (in C) for the peripherals, using the ST firmware library.
- RFlasher7: A graphical user interface for Flash programming.
- RLink support: Ride7 can communicate with the RLink, which is a USB hardware dongle that allows the user to program the ST7/STM8 on an application board and debug the application while it is running on the ST7/STM8. It uses the ICC or SWIM protocol. For more information refer to chapter “Debugging with hardware tools”.

Each tool mentioned above has a dedicated user manual (please refer to the appropriate manual for more details) apart from Ride7, the simulator and RLink.

2.2 Third party tools used in conjunction with RKit-STM8

RBuilder is a 100% Raisonance product. However it uses the ST7 libraries for most of the code it generates. This implies that as the ST library evolves, so will the code generated by RBuilder (starting from the same set of options). See http://www.st.com/

2.3 Supported derivatives

RKit-STM8 supports most of the existing STM8 and ST7 (those with ICC not ISP) derivatives to various degrees. An up-to-date list of supported derivatives and software simulation limitations can be seen in Ride7 when creating a new ST7/STM8 project (Project | New project): Browsing through the list of available devices displays some detailed characteristics in the “description field”.

2.4 Installing Raisonance Tools for STM8/ST7

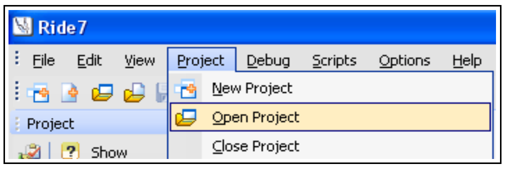
When installing Ride7 and RKit-STM8, make sure you have the latest software version from the Raisonance website: http://www.raisonance.com/download/

2.5 Example projects

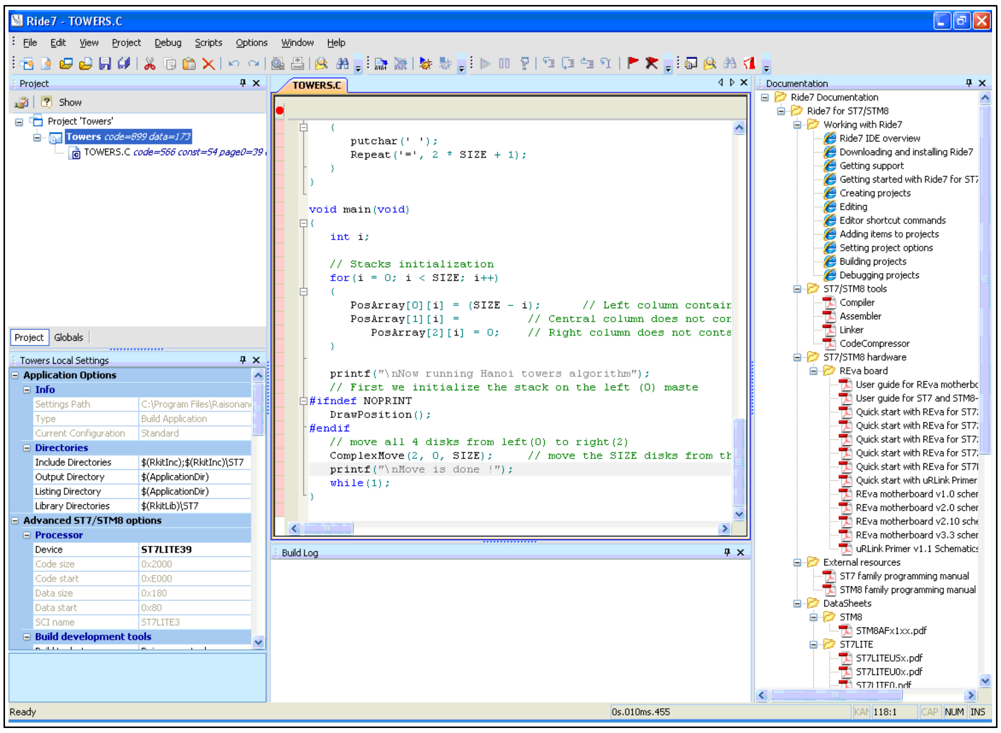
RKit-STM8 contains example projects written in C or assembler that are ready to run. Each example is described in the comments in the application's source files.

To open a project, for example Towers:

1. Click on **Project > Open Project** in **Ride7**.
2. Navigate to the Ride installation directory, then in “Examples\STM8\C\Towers”.
3. Click on **Towers.rprj** and **Open**.



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



```
    {
        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 contair
        PosArray[1][i] =                // Central column does not cor
          PosArray[2][i] = 0;    // Right column does not conte
    }

    printf("\nNow running Hanoi towers algorithm");
    // First we initialize the stack on the left (0) maste
#ifndef NOPRINT
    DrawPosition();
#endif
    // move all 4 disks from left(0) to right(2)
    ComplexMove(2, 0, SIZE);     // move the SIZE disks from th
    printf("\nMove is done !");
    while(1);
}
```

Towers Local Settings

Application Options	
Info	
Settings Path	C:\Program Files\Raisonan
Type	Build Application
Current Configuration	Standard
Directories	
Include Directories	$(RkitInc);$(RkitInc)\ST7
Output Directory	$(ApplicationDir)
Listing Directory	$(ApplicationDir)
Library Directories	$(RkitLib)\ST7
Advanced ST7/STM8 options	
Processor	
Device	**ST7LITE39**
Code size	0x2000
Code start	0xE000
Data size	0x180
Data start	0x80
SCI name	ST7LITE3
Build development tools	

3. Register the Raisonance tools for STM8/ST7

The new Ride7 and RKit-STM8 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 register your RKit-STM8 using your RKit-STM8 license (Serial Key or Dongle).

Unregistered software functions for a 30-day evaluation period with full features of the Enterprise version. After 30 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-STM8 software, then validate its operation (test compilation, connection to RLink and to target CPU, etc.).
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 key (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 (purchasers of RLink, REva, Open4 EvoPrimer).
4. Register the software by selecting the menu item **Help > License**...

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. If necessary users can always activate versions of the STM8 software that were released during their one-year support and maintenance contract for that product.

3.2 Register using a serial key

1. Launch Ride7
2. Select Help > License...
3. Select Serial Key Activation, click on Next
4. Paste the your Serial Key in the provided field
5. Click on Next
6. Click on Get Activation code online

3.3 Register using a dongle

1. Launch Ride7
2. Connect your RLink, REva, EvoPrimer, Open4 or USB dongle to a USB port on the PC
3. Select **Help > License...**
4. Dongle activation for USB dongles, click on **Next**

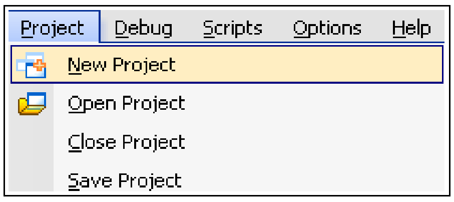
Ride detects your hardware and reads its serial number.

5. Click on **Get Activation code online**

These registration procedures are also detailed on www.support.raisonance.com.

4. Building a new project

4.1 Creating a new project



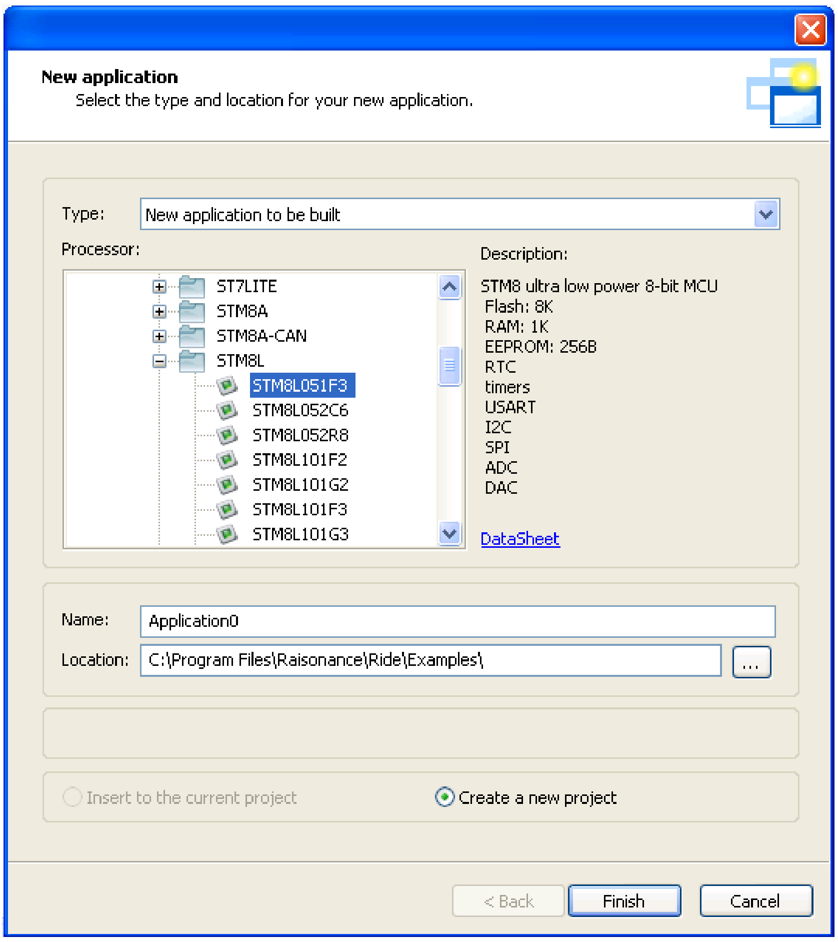
1. Click on the **Project** tab in the Ride7 menu.
2. In the drop down menu, click on **New Project**.

4.2 Creating a STM8 project

4.2.1 Creating the application

To create the settings, follow the order below:

1. Select the Type of application to be built.
2. Select the type of Processor e.g. STM8L051F3.
3. Create a Name for the project e.g. `Application0`
4. Select a directory Location e.g. *C:\Program Files\Raisonance\ride\examples\*
5. Click on Finish to generate your application. Your application project is now created.



New application

Select the type and location for your new application.

Type: New application to be built

Processor:

- ST7LITE
- STM8A
- STM8A-CAN
- STM8L
  - STM8L051F3
  - STM8L052C6
  - STM8L052R8
  - STM8L101F2
  - STM8L101G2
  - STM8L101F3
  - STM8L101G3

Description:

STM8 ultra low power 8-bit MCU
Flash: 8K
RAM: 1K
EEPROM: 256B
RTC
timers
USART
I2C
SPI
ADC
DAC

DataSheet

Name: Application0

Location: C:\Program Files\Raisonance\Ride\Examples\

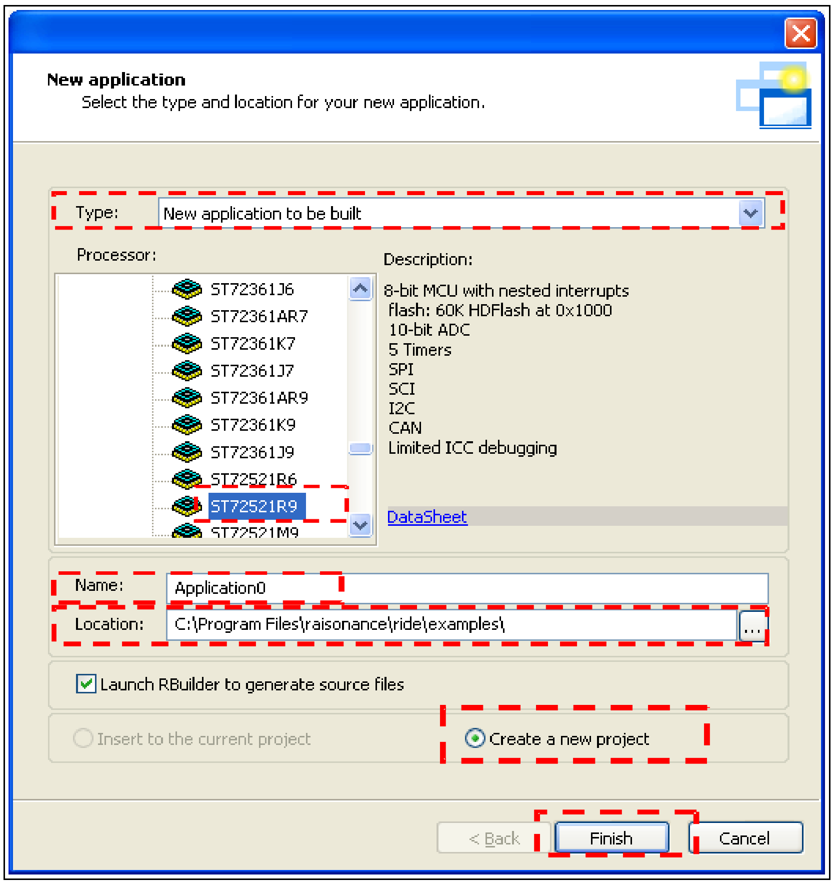
Insert to the current project [✓ Create a new project]

4.3 Creating a ST7 project

4.3.1 Creating the settings for the new application

To create the settings, follow the order below:

1. Select the **Type** of application to be built.
2. Select the type of **Processor** e.g. ST72521R9.
3. Create a **Name** for the project e.g. **`Application0`**
4. Select a directory **Location** e.g. *C:\Program Files\Raisonance\ride\examples\*
5. Tick **Launch RBuilder to generate source files**.
6. Select the radio button **Create a new project**.
7. Press **Finish**.



New application
Select the type and location for your new application.

Type: New application to be built

Processor:
ST72361J6
ST72361AR7
ST72361K7
ST72361J7
ST72361AR9
ST72361K9
ST72361J9
ST72521R6
ST72521R9
ST72521M9

Description:
8-bit MCU with nested interrupts
flash: 60K HDFlash at 0x1000
10-bit ADC
5 Timers
SPI
SCI
I2C
CAN
Limited ICC debugging

DataSheet

Name: Application0
Location: C:\Program Files\raisonance\ride\examples\

✓ Launch RBuilder to generate source files

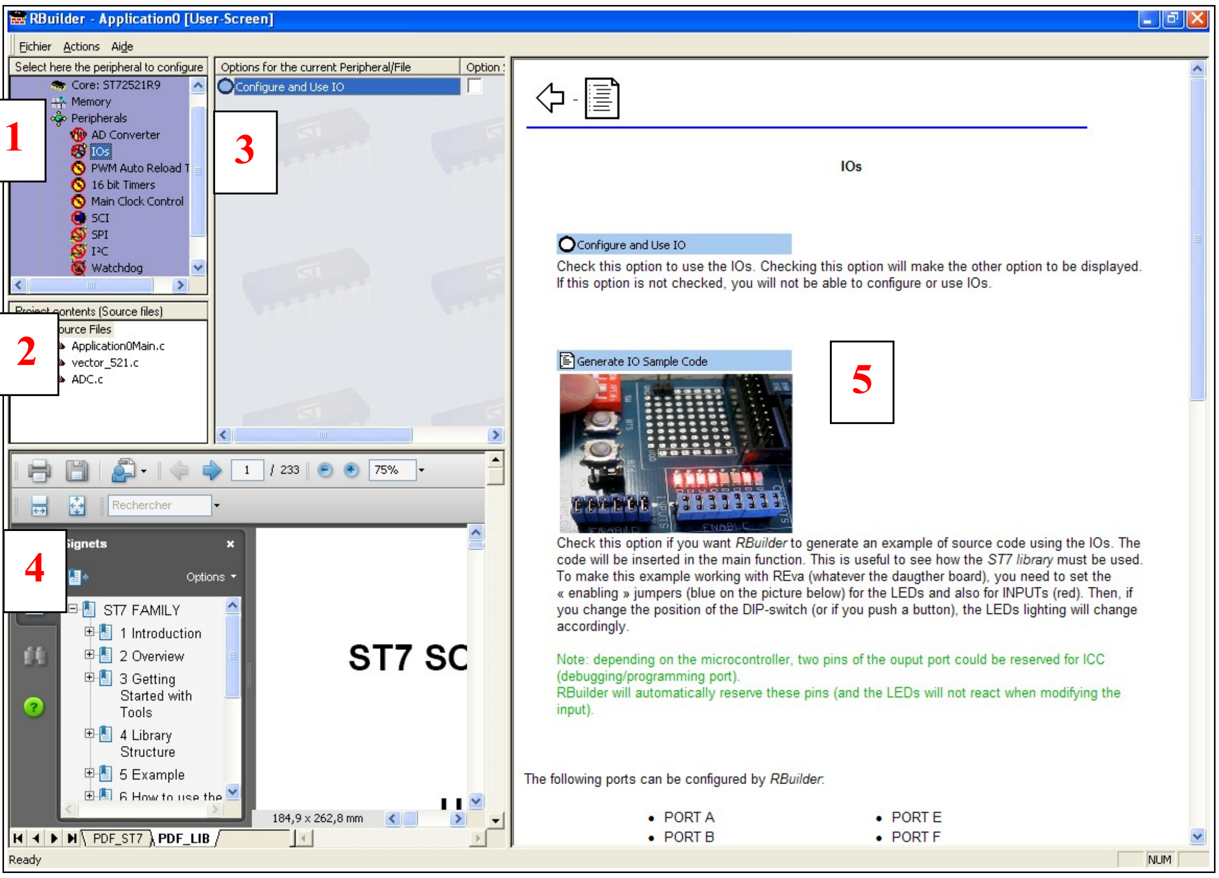
Insert to the current project [✓ Create a new project]

4.3.2 RBuilder start-up

RBuilder is an application builder based on the STM8/ST7 software library.
With RBuilder you can:

- Configure a project.
- Generate a project skeleton containing all the necessary code.
- Add your own code.
- **Execute** the application.

When the project settings have been set and validated with **Finish**, the following RBuilder start-up screen appears:



Check this option to use the IOs. Checking this option will make the other option to be displayed. If this option is not checked, you will not be able to configure or use IOs.

Check this option if you want *RBuilder* to generate an example of source code using the IOs. The code will be inserted in the main function. This is useful to see how the *ST7 library* must be used. To make this example working with REva (whatever the daugther board), you need to set the « enabling » jumpers (blue on the picture below) for the LEDs and also for INPUTs (red). Then, if you change the position of the DIP-switch (or if you push a button), the LEDs lighting will change accordingly.

Note: depending on the microcontroller, two pins of the ouput port could be reserved for ICC (debugging/programming port).
RBuilder will automatically reserve these pins (and the LEDs will not react when modifying the input).

The following ports can be configured by *RBuilder*.

- PORT A
- PORT B
- PORT E
- PORT F

The screen is divided into 5 different zones:

- **Zone 1** displays the peripherals associated with the microcontroller you are using. This view is called the **project tree**. To use and configure a peripheral, select it in the tree and the options associated with the selected peripheral are displayed in Zone 4.
- **Zone 2** shows you which file will be added to your project. This list changes while you are adding/removing peripherals to your project.
- **Zone 3** displays the datasheet of your microcontroller and the ST7 software library manual.
- **Zone 4** displays the options related to the selected peripheral. This view is called the **Option View**.
- **Zone 5** displays the online help which guides you through the configuration of your project.

4.3.3 Configuring the peripherals from RBuilder

Once you are in RBuilder, you can configure the peripherals for your project.

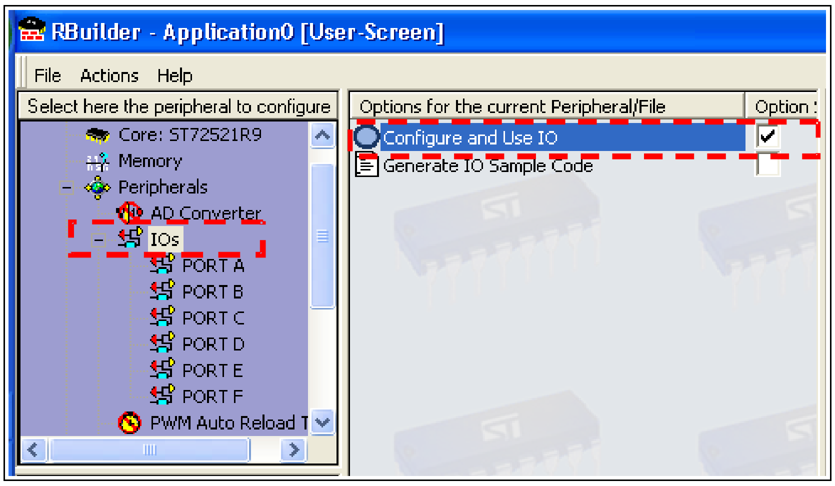
Some peripherals, like the I/O ports, offer the option to generate sample code. If you check this option, sample code is placed in the main function.

Example 1 Port A: I/O ports

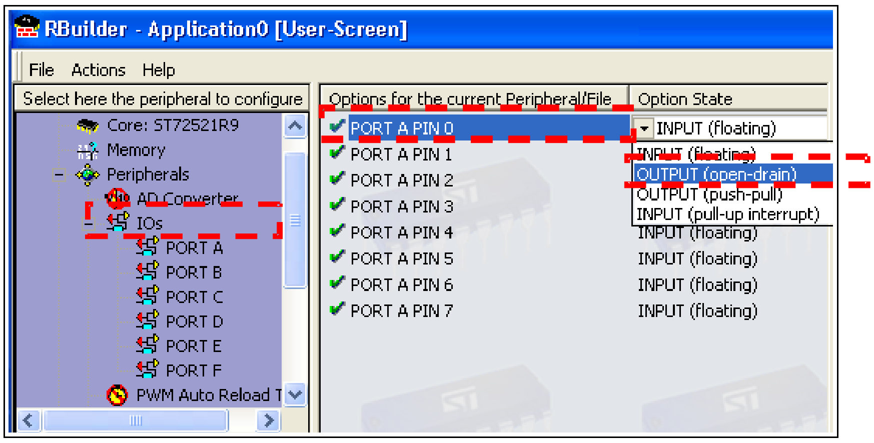
This example shows how to use the basic library functions related to the associated peripheral(s).

1. To add I/O support to your project, click on the I/O peripheral on the left (in the project tree), as shown in the screen shot below. The peripheral will be unblocked and the ‘no entry’ sign disappears.
2. Tick the **Option** check box of each peripheral you would like to add.
3. In the example below, tick the check box **Configure and Use IO**
4. Then click on **Options for the current Peripheral/File** view.

5. The available options for the peripheral appear and, for I/Os, the list of available ports is added in the project tree. If no port is selected, only one option is available for I/Os. However, when a specific port is selected in the project tree, more options are displayed.



6. Click on **Port A** to show the definable settings. Each pin of the PORT is easily configured using a combo box which lets you specify if the pin is either a floating INPUT, a pull-up interrupt INPUT, an open-drain OUTPUT or a push-pull OUTPUT as shown in the next screen shot.



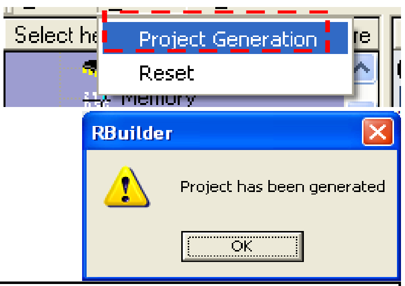
Configure other peripherals

To select other peripherals, simply select the desired peripheral in the project tree as in the I/O example above, tick the **Option** checkbox and configure as you wish.

4.3.4 Generating your project

Once you have configured your project:

1. Select **Project Generation** in the **Actions** menu:
2. After doing this, RBuilder generates the sample/prototype application code based on the project configuration, and notifies you when generation is complete.
3. RBuilder then closes and returns to the main Ride7 window.
4. You can now compile the project.



Project has been generated

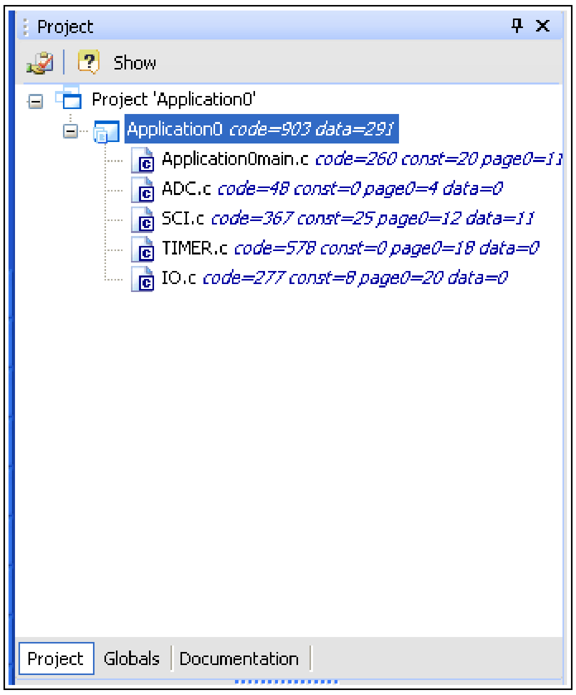
Warning: Once the project is generated, there is no way to go back to modify one option and generate it again. If you find out that you were mistaken about an option, then you have to restart the project creation process from the beginning.

4.3.5 Adding your application code

Once you are back in Ride7, source files will be added to your project. Most of the files in your project are generated from the STM8/ST7 software library according to the configuration options that you specified in RBuilder. These files should not need to be edited.

Two additional files are generated: *myprojectmain.c* and *myprojectint.c*

- ***myprojectmain.c*** is the main file of the project. In this file, RBuilder generates the initialization function for all peripherals, the `main()` function and any user requested functions (for example `putchar` and `getchar` for the SCI.



```
Project 'Application0'
  Application0 code=903 data=291
    Application0main.c code=260 const=20 page0=11
    ADC.c code=48 const=0 page0=4 data=0
    SCI.c code=367 const=25 page0=12 data=11
    TIMER.c code=578 const=0 page0=18 data=0
    IO.c code=277 const=8 page0=20 data=0
```

- ***myprojectint.c*** contains all the interrupt functions. This file is generated only if the user selected the use of interrupts in RBuilder. Otherwise it is not part of the project.

4.3.6 Main file

The following is an example of a main file, generated by RBuilder, for which the following peripherals have been selected:

- I/Os

- Timer
- SCI
- ADC

In this file, you can see:

- The `PeriphInit` function (**blue arrow**) performs initialization for all peripherals according to the options you selected in RBuilder.
- The `main` function (**green arrow**) is made of a call to `PeriphInit` to initialize all peripherals, followed by an infinite loop that contains the application code. RBuilder gives you the possibility to generate sample code for almost every peripheral. This example contains code for the ADC, the SCI and I/Os. You just need to add your specific code at the end of the function (**red arrow**).
- The `TERMIO_PutChar` function (**brown arrow**) is generated when you select **Generate a putchar function** in RBuilder's SCI options section.

```
/*============================================================== */
/* Project:        Application0 */
/* File:           Application0main.c */
/* Organization:     */
/* Author:             (initial version generated by RBuilder) */
/* Date:           5/6/2008 */
/*============================================================== */
#include "ST7lib_config.h"
#include <stdio.h>
/* --------------------------------------------------------------------------- */
/* Function:       PeriphInit */
/* Purpose:        Periph configuration at RESET */
/* Date:           5/6/2008 */
/* --------------------------------------------------------------------------- */
void PeriphInit ( void ) {

   /* A/D Converter initialization */
   ADC_Init(ADC_DEFAULT);
   ADC_Select_Channel(3);

   /* SCI initialization */
   SCI_Init(SCI_DEFAULT_PARAM1, SCI_DEFAULT_PARAM2);
     /* NOTE: Baud rate calculation is performed for Fcpu= 6000000 Hz */
   SCI_Select_Baudrate(0x0);
   SCIERPR =  0x27;  /* Set RX Baudrate (real= 9615)  */
   SCIETPR =  0x27;  /* Set TX Baudrate (real= 9615)  */
   SCI_Mode(SCI_TX_ENABLE|SCI_RX_ENABLE);

   /* 16-bit timers initialization */
   TIMERA_Init(TIMER_FCPU_4);   /* Timer A */

   /* I/O Port initialization */
   IO_Init();
   IO_Input(IO_FLOATING_IT, IO_PORT_B, 0x88);
   IO_Input(IO_PULL_UP_IT, IO_PORT_B, 0x77);
   IO_Output(IO_PUSH_PULL, IO_PORT_F, 0x87);
   IO_Input(IO_FLOATING_IT, IO_PORT_F, 0x88);

}/* end of  PeriphInit */

/* ---------------------------------------------------------------------------- */
/* Function:       main() */
/* Purpose:        main routine */
/* Date:           5/6/2008 */
/* ---------------------------------------------------------------------------- */
void main ( void ) {

   /* Configure the internal peripherals */
   PeriphInit();
   while ( 1 ){
     {
        /* * * * Sample code for the ADC * * *
       This code works with the REva evaluation board. It performs the following:
           a. Measurement of the voltage set by the potentiometer
           b. Translation to a value that fits into the range [0,7]
           c. Copy of the resulting value to the Port F (LEDs as a gauge). */
        int Conv_Data1;
        const char array_Val_Gauge [8] = {0xFE, 0xFC, 0xF8, 0xF0, 0xE0, 0xC0, 0x80, 0x00};
        ADC_Select_Channel(3);                    /* Select channel 3  */
        ADC_Enable();                             /* Start conversion ADON bit is set */
        while ( !ADC_Test_Conversn_Complete() ); /* Wait till conversion completes */
        Conv_Data1 = ADC_Conversn_Read();         /* Read converted value */
        if ( !Conv_Data1 ) {
           IO_ByteWrite(IO_PORT_F, 0xff);    /* Write Port F: all LEDs switched off */
        }
        else {
           Conv_Data1 >>= 7;                   /* Keep the 3 Most Significant Bits */
           IO_ByteWrite(IO_PORT_F, array_Val_Gauge[Conv_Data1]); /* Write Port F */
        }
        ADC_Disable();
     }
     /* Sample code for SCI */
     puts("Hello world");
     /* Insert your code here... */

   }
}/* end of main */
```

```
/* ------------------------------------------------------------------------- */
/* Function:       TERMIO_PutChar */
/* Purpose:        Putchar in polling mode on SCI */
/* Date:           5/6/2008 */
/* ------------------------------------------------------------------------- */
void TERMIO_PutChar(char c){
   if (c == '\n'){
      SCI_PutByte ('\r');
      while (!SCI_IsTransmitCompleted());
   }
   SCI_PutByte (c);
   while (!SCI_IsTransmitCompleted());
}

/* ------------------------------------------------------------------------- */
/* Function:       TERMIO_GetChar */
/* Purpose:        Getchar in polling mode on SCI */
/* Date:           5/6/2008 */
/* ------------------------------------------------------------------------- */
unsigned char TERMIO_GetChar(void){
   unsigned char c;
   while(SCI_IsReceptionCompleted()!=SCI_RECEIVE_OK);
   c = SCI_GetByte();
   return(c);
}
```

4.4 Adding your code to the project

Now that you have a project, you need to add your application code.

1. To add your source files, select **Project > Add item**.
   This opens a file selection window where you add your source files in C or assembler.
2. To rebuild the application, just press **F9** or select **Project** > **Make All.**
3. You can now debug your application with the simulator.

If you don't have source files, you can create your own, by selecting **File > New … > Source File**
You can now enter your code. To save it, go to **File > Save**

5. Debugging with simulator

5.1 About the STM8/ST7 simulator

RKit-STM8 provides a simulator capable of simulating the most common STM8/ST7 peripherals. The simulator lets you check your code as well as the interaction between your code and the peripheral, before you debug with an in-circuit debugger or emulator.

5.2 STM8/ST7 derivatives supported by RKit-STM8

RKit-STM8 supports most of the existing STM8/ST7 derivatives to various degrees. The up-to-date list of supported derivatives and the limitations of the software simulation can be seen in the **Target Options** in Ride7.

5.3 Peripherals and main file used in example project

For the rest of this document we use a project that uses the following peripheral I/Os:

- PORT A Pin 1 > Output (push/pull)
- PORT A Pin 2 > Output (push/pull)
- PORT A Pin 3 > Output (push/pull)
- PORT A Pin 4 > Output (push/pull)

The following code has been written in the main function:

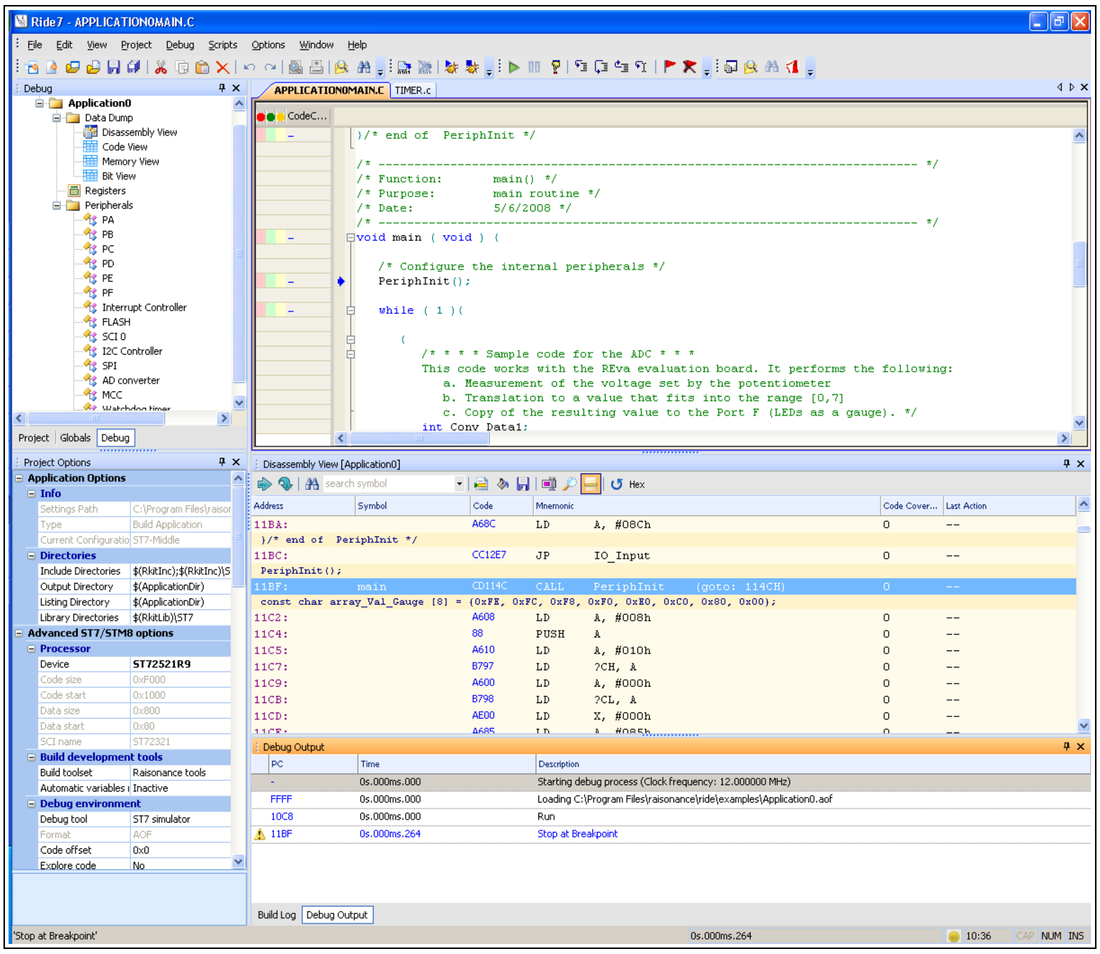
```
int toggle = 0; /*This declaration goes at the beginning of the main*/

IO_Write(IO_PORT_A, IO_PIN_0, (toggle == 0)?IO_DATA_HIGH:IO_DATA_LOW);
IO_Write(IO_PORT_A, IO_PIN_0, (toggle == 1)?IO_DATA_HIGH:IO_DATA_LOW);
IO_Write(IO_PORT_A, IO_PIN_0, (toggle == 2)?IO_DATA_HIGH:IO_DATA_LOW);
IO_Write(IO_PORT_A, IO_PIN_0, (toggle == 3)?IO_DATA_HIGH:IO_DATA_LOW);

if(++toggle==4)
      toggle = 0;
```

So you can easily build this project to use the simulator, as described in the chapter "*Building a new project in RBuilder*".

5.4 Launch the simulator

Before launching the simulation, you must configure the debugger.

To launch the simulator, type **CTRL-D**. If your project has not been built, it will be built automatically before the simulator launches. Otherwise, the simulator launches immediately and you are now in the simulator. Your Ride7 window looks like the following:



```
}/* end of  PeriphInit */

/* ---------------------------------------------------------------------------- */
/* Function:        main() */
/* Purpose:         main routine */
/* Date:            5/6/2008 */
/* ---------------------------------------------------------------------------- */
void main ( void ) {

   /* Configure the internal peripherals */
   PeriphInit();

   while ( 1 ){

      {
         /* * * * Sample code for the ADC * * *
         This code works with the REva evaluation board. It performs the following:
            a. Measurement of the voltage set by the potentiometer
            b. Translation to a value that fits into the range [0,7]
            c. Copy of the resulting value to the Port F (LEDs as a gauge). */
         int Conv_Data1;
```

Project Options

Application Options	
Info	
Settings Path	C:\Program Files\raisor
Type	Build Application
Current Configuratio	ST7-Middle
Directories	
Include Directories	$(RkitInc);$(RkitInc)\S
Output Directory	$(ApplicationDir)
Listing Directory	$(ApplicationDir)
Library Directories	$(RkitLib)\ST7
Advanced ST7/STM8 options	
Processor	
Device	ST72521R9
Code size	0xF000
Code start	0x1000
Data size	0x800
Data start	0x80
SCI name	ST72321
Build development tools	
Build toolset	Raisonance tools
Automatic variables i	Inactive
Debug environment	
Debug tool	ST7 simulator
Format	AOF
Code offset	0x0
Explore code	No

```
Address   Symbol    Code     Mnemonic                                    Code Cover...  Last Action
11BA:               A68C     LD     A, #08Ch                             0              --
 }/* end of  PeriphInit */
11BC:               CC12E7   JP     IO_Input                             0              --
 PeriphInit();
11BF:     main      CD114C   CALL   PeriphInit     (goto: 114CH)         0              --
 const char array_Val_Gauge [8] = {0xFE, 0xFC, 0xF8, 0xF0, 0xE0, 0xC0, 0x80, 0x00};
11C2:               A608     LD     A, #008h                             0              --
11C4:               88       PUSH   A                                    0              --
11C5:               A610     LD     A, #010h                             0              --
11C7:               B797     LD     ?CH, A                               0              --
11C9:               A600     LD     A, #000h                             0              --
11CB:               B798     LD     ?CL, A                               0              --
11CD:               AE00     LD     X, #000h                             0              --
```

Debug Output

```
PC     Time             Description
-      0s.000ms.000     Starting debug process (Clock frequency: 12.000000 MHz)
FFFF   0s.000ms.000     Loading C:\Program Files\raisonance\ride\examples\Application0.aof
10C8   0s.000ms.000     Run
11BF   0s.000ms.264     Stop at Breakpoint
```

```
'Stop at Breakpoint'                  0s.000ms.264     10:36   CAP  NUM  INS
```

The screen shot above shows:

1. **Application0main.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.

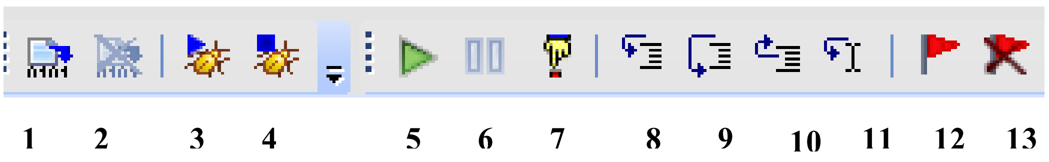
5.5 Debug options

Choose the debugging environment:

- ST7 simulator or STM8 simulator for software debugging.
- RLink ST7 or RLink STM8 for hardware debugging. (see Chapters 6 and 7).

5.6 Debug controls

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



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**: On a function call in a line of the C source code, this button steps over the called function. (F8)
10. **Step out Exit functions** (Shift + F7)
11. **Run to** (Ctrl Q)
12. **Toggle breakpoint** (F5)
13. Clear all breakpoints

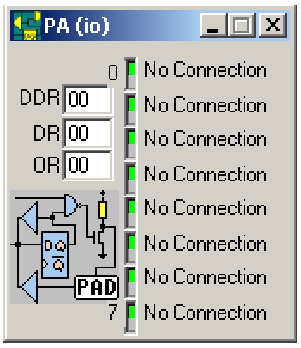
5.7 Peripheral status view

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 (see section 4.4).

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.

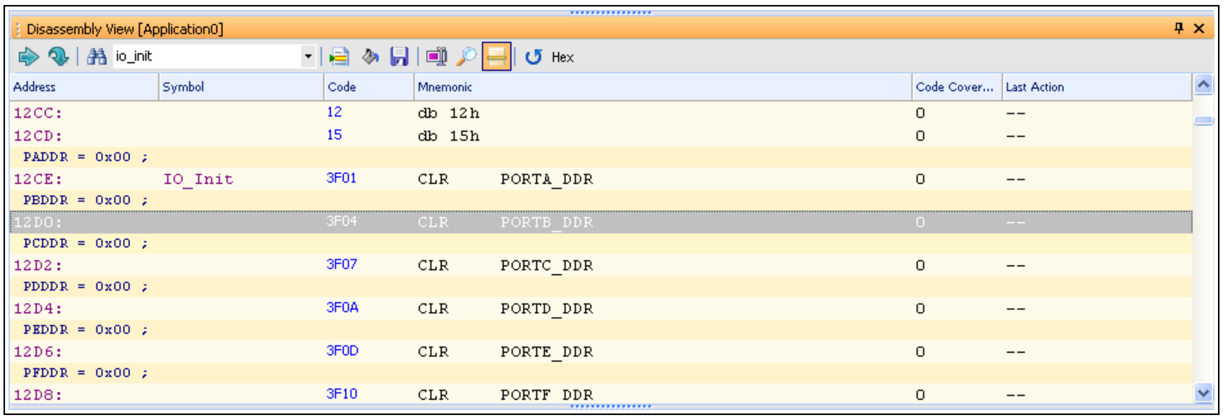


5.8 Breakpoints

You can set a breakpoint either in the source file or in the code view.

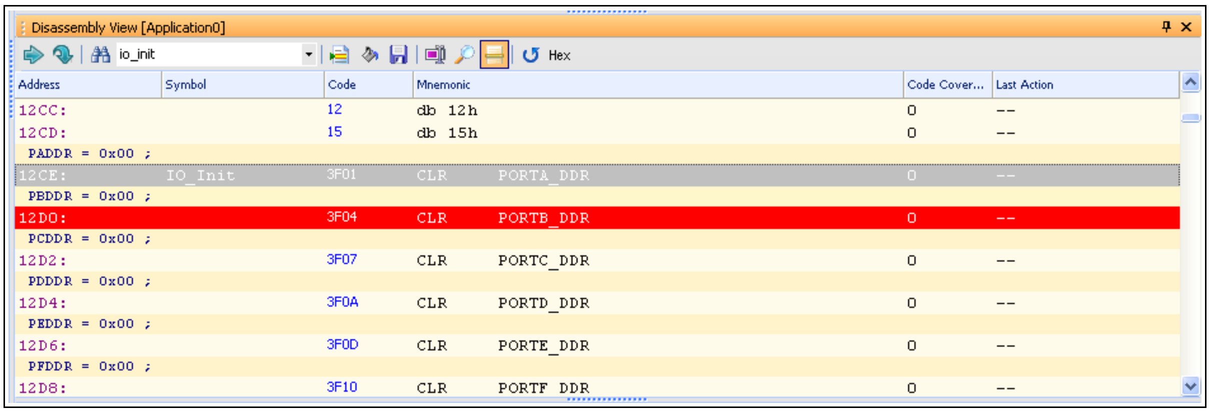
5.8.1 Code view

In the code view, first select the line on which you want to stop. The line becomes grey:


Disassembly View [Application0]

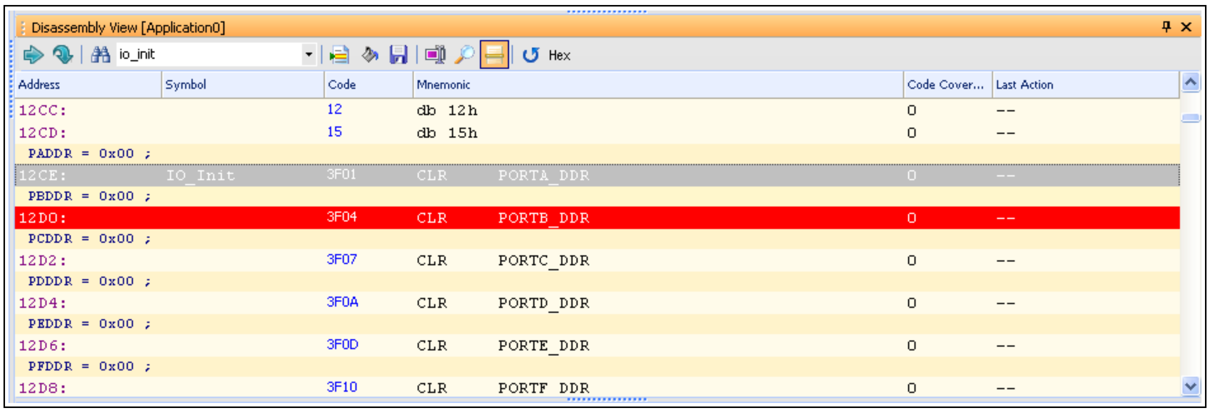
Address	Symbol	Code	Mnemonic	Code Cover...	Last Action
12CC:		12	db 12h	0	--
12CD:		15	db 15h	0	--
PADDR = 0x00 ;					
12CE:	IO_Init	3F01	CLR PORTA_DDR	0	--
PBDDR = 0x00 ;					
12D0:		3F04	CLR PORTB_DDR	0	--
PCDDR = 0x00 ;					
12D2:		3F07	CLR PORTC_DDR	0	--
PDDDR = 0x00 ;					
12D4:		3F0A	CLR PORTD_DDR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

Then click on the **Toggle Breakpoint** button and the line becomes red:


Disassembly View [Application0]

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	--

This means that a breakpoint has been set on this line.

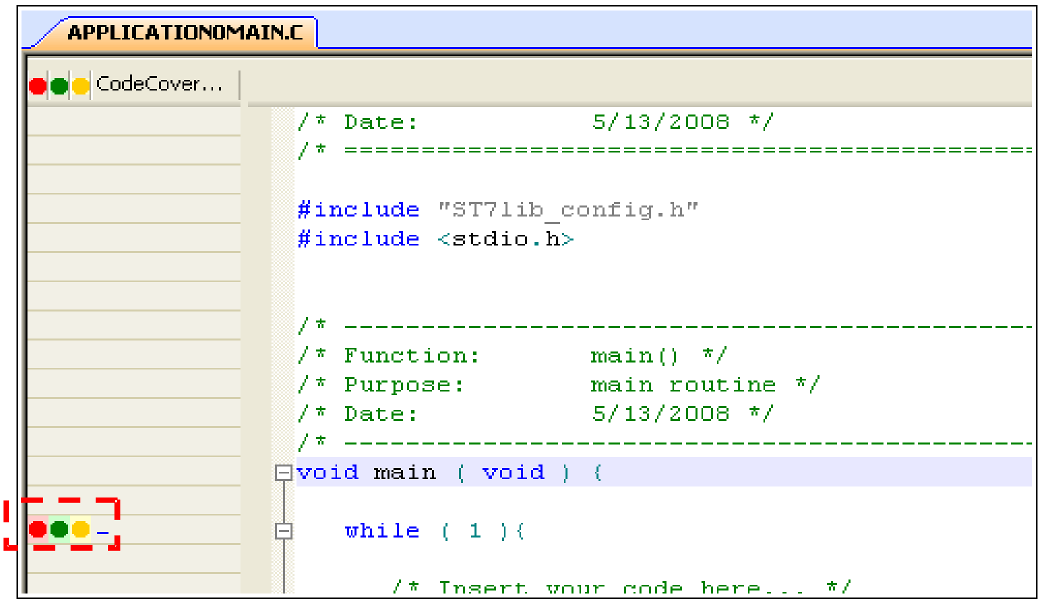

Disassembly View [Application0]

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	--

5.8.2 Source view

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

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:



```
/* Date:              5/13/2008 */
/* ============================================

#include "ST7lib_config.h"
#include <stdio.h>


/* --------------------------------------------
/* Function:          main() */
/* Purpose:           main routine */
/* Date:              5/13/2008 */
/* --------------------------------------------
void main ( void ) {

    while ( 1 ){

        /* Insert your code here... */
```

6. Debugging with hardware tools

RKit-STM8 can be used with a number of hardware debug tools (in addition to the Raisonance simulator):

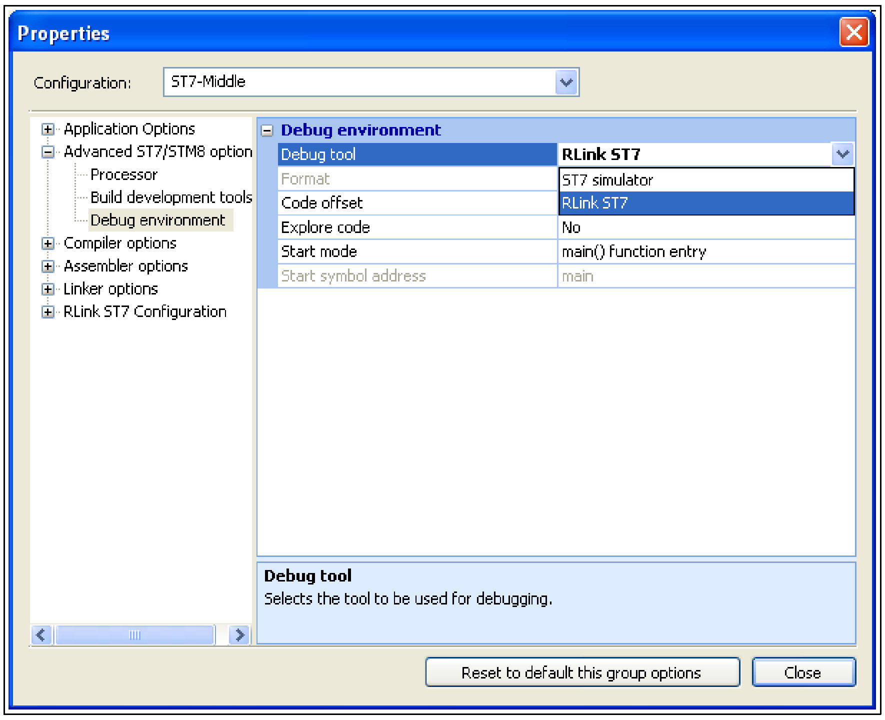
- RLink ST7 for in-circuit debugging ST7 applications using the ICC (not ISP) protocol.
- RLink STM8 for in-circuit debugging STM8 applications using the SWIM protocol.

Note: 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. Refer to the previous chapter and get familiar with the simulator before starting to work with the hardware debug tools.

6.1 Selecting hardware debug tool

In the **Advanced STM8/ST7 > Debug environment > Debug tool** menu of Ride7 choose your target hardware debugger. Select the tool that corresponds to your debug hardware e.g. **RLink_ST7**. You should select RLink if you are using:

- RLink connected to the target STM8/ST7 on your application board via an ICC/SWIM connector,
- REva evaluation board, which includes an embedded RLink.
- Primer stick (ST7UltraLite, ST7FOX, STM8A, ...), which includes an embedded RLink.



Configuration: ST7-Middle

Debug environment	
Debug tool	RLink ST7
Format	
Code offset	
Explore code	No
Start mode	main() function entry
Start symbol address	main

Debug tool
Selects the tool to be used for debugging.

6.2 RLink programming (ICP) and debugging (ICD) features

The RLink is a USB interface device designed by Raisonance. It allows In-Circuit-Programming (ICP) and In-Circuit-Debugging (ICD) of various microcontrollers, including all the ST7 and STM8 devices supported by RKit-STM8. (see the up-to-date list in the **Target Options**).

Note: ST7 devices with HDFlash and no debug module can be programmed, but debugging through ICC is very limited with them. (See Section 5.6.9 Limitations for ST7 HDFlash targets without debug module for more information).

With the ST7 devices, RLink uses the In-Circuit-Communication (ICC) protocol from ST to perform ICP and ICD. RLink versions up to V2.02 feature the standard 10-point ICC connector, as defined by ST, directly on the RLink. Version 3.0 and earlier versions require an adaptor that converts the 24-point RLink connector to a 10-point ICC connector.

Note: ST7 devices that do not support the ICC protocol are not supported by the Raisonance tools. These devices use the ISP or other protocols. Please contact ST for information about tools supporting these devices.

With the STM8 devices, RLink uses the Serial Wire Interface Module (SWIM) protocol from ST to perform ICP and ICD. RLink versions 3.0 and earlier require an adaptor to feature the standard 4-point SWIM connector as defined by ST. RLink V2.02 and older versions cannot be used with STM8.

6.2.1 RLink USB driver

Windows automatically recognizes when an RLink is plugged in, then it must associate the RLink with a USB driver.

The USB driver should be installed before you plug the RLink. Unless you have specified otherwise, it is installed along with Ride7. If the USB driver has not been installed, launch the program RlinkUSBInstall.exe. For standard installations of Ride7, it is located in:

C:\Program Files\Raisonance\Ride\Driver\RLinkDrv\RLinkUSBInstall.exe

After running this program, when you plug an RLink in, Windows recognizes it automatically.

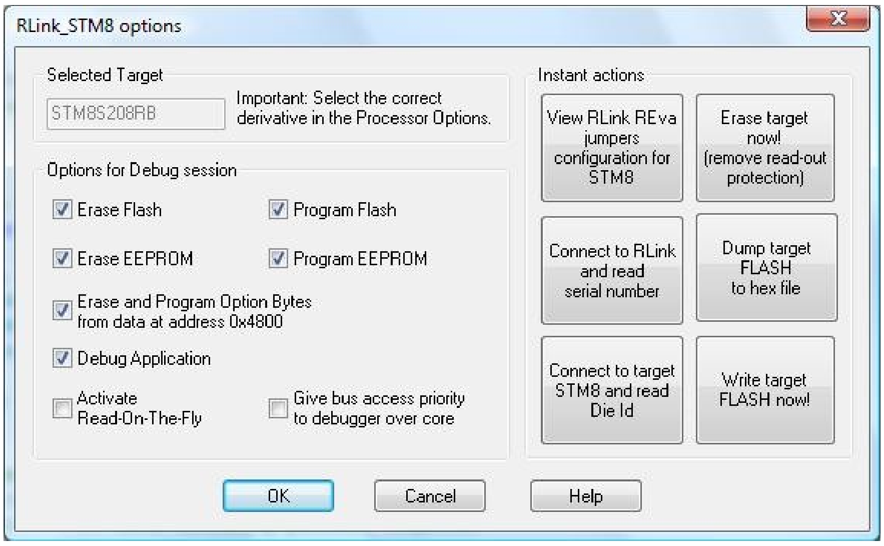
6.3 REva board

The REva demonstration board includes an RLink. The whole board can be powered by the USB through the RLink. The target chip is placed on an interchangeable daughter board that can include different targets. For Ride7, (and for Windows and the USB driver) there is no difference between operating the REva and using an RLink with any other application board featuring the ICC or SWIM connector.

For more information about the demo board itself (schemes, etc.), see the appropriate documentation.

6.4 Configuring RLink STM8

After selecting RLink STM8 as your debugging tool (see section “Selecting hardware debug tool”), click on the **Advanced Options** button to open the **RLink_STM8 options** dialog box shown below:


RLink_STM8 options

Selected Target: STM8S208RB

Important: Select the correct derivative in the Processor Options.

Options for Debug session

- [✓] Erase Flash
- [✓] Program Flash
- [✓] Erase EEPROM
- [✓] Program EEPROM
- [✓] Erase and Program Option Bytes from data at address 0x4800
- [✓] Debug Application
- [ ] Activate Read-On-The-Fly
- [ ] Give bus access priority to debugger over core

Instant actions
View RLink REva jumpers configuration for STM8
Erase target now! (remove read-out protection)
Connect to RLink and read serial number
Dump target FLASH to hex file
Connect to target STM8 and read Die Id
Write target FLASH now!
OK
Cancel
Help

1. It is critical to check that the **Selected Target** corresponds to your chip.
   If not, you must go back to the Ride7 main screen and change the **Device selection**.
2. Select the memory regions that you want Ride to Erase and Program before debugging.
3. To debug your application, confirm that Debug Application is checked.
   Uncheck the Debug Application option if you want to use RLink as a simple programmer, e.g. if you want to run the application on the STM8 without debugging it.
4. Select if you want to Activate Read-On-The-Fly. This option will make debugging more comfortable by updating the memory views during CPU execution.
5. The Read-On-The-Fly option, but also the “normal” debugging communication can create bus access conflicts if your application makes heavy load on the bus. That can result in slow or stalled debugging. To solve this you can select the Give bus access priority to debugger over core option. But you must be aware that this might impair the real-time execution feature of the debugger by slowing down the application's execution. So you should only activate this option if you have slow or stalling debugger.

6.4.1 RLink STM8 Instant actions

This section of the **RLink_STM8 options** dialog box carries out the instant actions listed below without leaving this dialog box. This is useful for testing connections and retrieving information from the RLink and your ST7, as well as for programming the ST7 and its option bytes.

- **Connect to RLink and read serial number** is useful for checking that RLink is working and properly connected and that the USB driver is correctly installed. It also allows you to read the RLink serial number, which you will be asked for if you contact our support team.
- **Erase target now!** allows you to completely erase the target's Flash (writing 0x00), option bytes and EEPROM (if your STM8 features any). This is the correct way to remove the read-out protection from a protected device.
- **Dump target FLASH to hex file** reads the contents of the Flash and writes it in a file in hex format whose name is derived from the current application's name with the extension *.hex* (*<application name>.hex*).
- **Write target FLASH now!** programs the Flash with the current application's hex file generated by the linker. Then, launches the execution in user mode.

6.4.2 Jumpers for RLink STM8 using ADP

If you are using a V3.0 RLink or earlier, which features a 24-point connector, you must use an adaptor (ADP) for connecting to STM8 devices. These adaptors convert the 24-point RLink connector into a 4-point SWIM connector. These ADPs contain jumpers that you may need to set, or not, depending on your situation.

All versions of ADPs ST7-STM8 allow connection to STM8 devices.

Version 1.X can only connect to STM8 devices with a power voltage as low as 2.2V.

Version 2.x allows to connect to STM8 devices with a power as low as 1.6V.

All versions also allow to connect to ST7 devices.

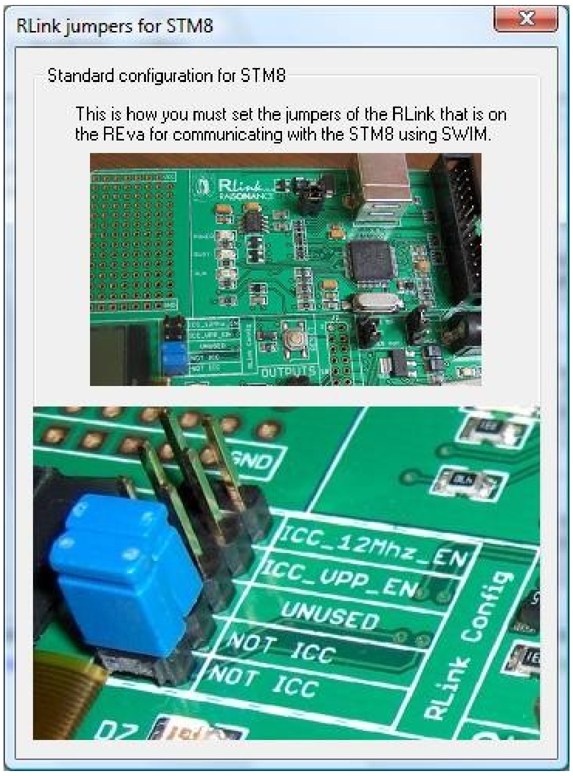
These ADPs feature four jumpers:

- Jumper “SWIM” for selecting SWIM or ICC protocol. (select SWIM for STM8)
- Jumper “PW5V” for transferring the 5V power from RLink to the target STM8 board. Use this if your board has no power and you want to power it from the RLink. !!! ONLY DO THIS IF YOUR TARGET BOARD IS 5V-COMPLIANT !!!
- Jumper “Adapt” which should be used if your board's power supply is lower than 3V. This jumper allows to reduce the value of the ADP's pull-up resistor on the SWIMDATA signal.
- Jumper “12MHz” is for ST7 only. Do not set it when connecting to STM8 devices.

6.4.3 Jumpers for STM8 REva board

If you are using a REva board, you must make sure that the jumpers are set correctly on the RLink part of the board.

To do this, click on **View RLink jumpers configuration for STM8**. The following illustrations showing the STM8 configuration for the RLink jumpers are displayed:


RLink jumpers for STM8

Standard configuration for STM8

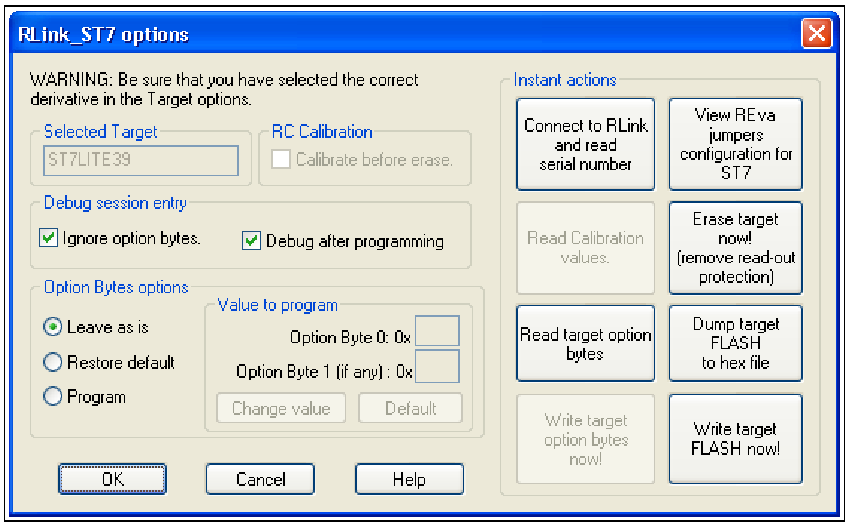
This is how you must set the jumpers of the RLink that is on the REva for communicating with the STM8 using SWIM.

Note: Should the pictures in this documentation and in Ride7 be different, please assume that those shown in Ride7 are correct. If you purchased RLink as part of a STM8 kit (such as the REva board for STM8), then the jumpers should already be correctly set. For this reason, you should only need to adjust these jumpers if they were accidentally unplugged, or if you are using an RLink that was configured for another protocol, such as ICC.

6.5 Configuring RLink ST7

After selecting RLink ST7 as your debugging tool (see section 6.1.Selecting hardware debug tool), click on the **Advanced Options** button to open the RLink ST7 **options** dialog box shown below.

1. It is critical to check that the **Selected Target** corresponds to your chip.
   If not, you must go back to the Ride7 main screen and change the **Device selection**.


RLink_ST7 options

WARNING: Be sure that you have selected the correct derivative in the Target options.

Selected Target: ST7LITE39

RC Calibration: Calibrate before erase.

Debug session entry: [✓] Ignore option bytes. [✓] Debug after programming

Option Bytes options: [✓ Leave as is] Restore default Program

Value to program: Option Byte 0: 0x Option Byte 1 (if any) : 0x

Change value
Default
OK
Cancel
Help
Instant actions
Connect to RLink and read serial number
View REva jumpers configuration for ST7
Read Calibration values.
Erase target now! (remove read-out protection)
Read target option bytes
Dump target FLASH to hex file
Write target option bytes now!
Write target FLASH now!

2. Next, indicate the preferred reset method to use when establishing in-circuit communication with the target ST7 by checking or unchecking the “Ignore Option Bytes” checkbox. See the ICC section of your device Datasheet for more information about this.
   If you do not know, just leave the default: If the first method fails, Ride7 will try the other, the only drawback for you is that the reset will take a little more time (less than half a second more).
   Be careful that the external clock is provided (either from RLink or another external source) if both your device's default option bytes and current option bytes select the external clock.
3. To debug your application, confirm that **Debug** is checked.
   Uncheck the **Debug** option if you want to use RLink as a simple programmer, e.g. if you want to try the application on the ST7 without debugging it.

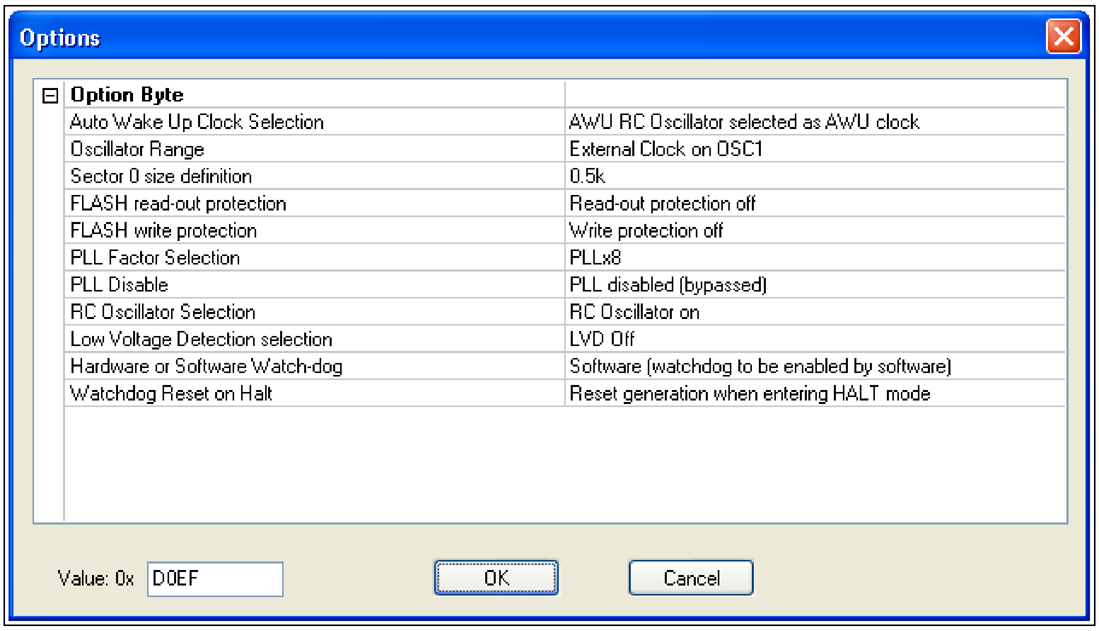
Note:
If the Debug option is checked, the application will be patched (to add code such as reset and trap vectors for debugging) and is only executed if it is driven by Ride7 through RLink.
If the Debug option is unchecked, then launching the debug session will simply program the unpatched code to your ST7 and start execution. This is useful when using RLink and you do not have the source code.

6.5.1 Option bytes

Next, you need to configure how the ST7's option bytes are handled.

- **Leave as is**: the option bytes are not programmed. They retain their current values.
- **Restore default**: tells Ride7 to erase the option bytes and restore the default values before loading the Flash and debugging. If you want to know what value will be loaded when selecting this option, then click the **Default** button, and you will see the default values in the **Value to program** section. These default values are NOT the factory settings. They are the values that are best for debugging and they are not fit for production. (minimal protection, etc.)
- **Program** tells Ride7 to erase the option bytes prior to programming the Flash memory, and to program the option bytes with the value that you have specified in the **Value to program** section. The option bytes are programmed after the Flash memory has been programmed.
- **Value to program**: This is the value that will be written in the option bytes if the **Program** option is selected. You can change the value by typing the value that you want in the **Option Byte 0** and the **Option Byte 1** fields (refer to your device datasheet for the meanings of these values), or by selecting **Change value** and configuring the options controlled by each option byte in the **Options** dialog box (shown below).

Warning: Some option byte values such as those controlling read-out, debug and re-write protection will prevent any further re-programming of your ST7. Be careful when setting the option byte values and refer to your device Datasheet for complete descriptions of the option byte values for your ST7.



Option Byte	
Auto Wake Up Clock Selection	AWU RC Oscillator selected as AWU clock
Oscillator Range	External Clock on OSC1
Sector 0 size definition	0.5k
FLASH read-out protection	Read-out protection off
FLASH write protection	Write protection off
PLL Factor Selection	PLLx8
PLL Disable	PLL disabled (bypassed)
RC Oscillator Selection	RC Oscillator on
Low Voltage Detection selection	LVD Off
Hardware or Software Watch-dog	Software (watchdog to be enabled by software)
Watchdog Reset on Halt	Reset generation when entering HALT mode

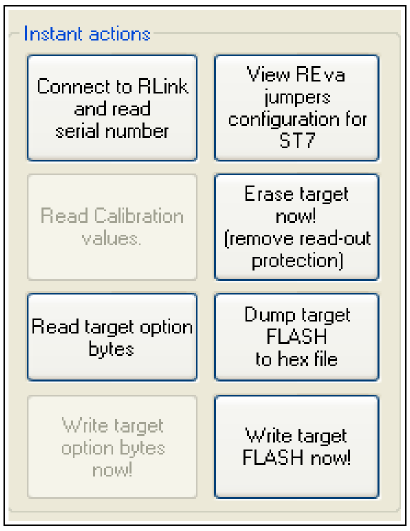
Value: 0x D0EF

The **Options** dialog box shows the meaning of each bit of the option bytes. This can help prevent errors resulting from typing the wrong value to program to the option bytes. Click on the field on the right to select option byte settings from a drop-down list of possible settings.

Finally, clicking **Default** restores the default value in the edit fields if you think that you might have typed in an incorrect value.

Note: **Restore default** is the same as **Program** with the default value, but it's faster!

6.5.2 Instant actions


Instant actions
Connect to RLink and read serial number
View REva jumpers configuration for ST7
Read Calibration values.
Erase target now! (remove read-out protection)
Read target option bytes
Dump target FLASH to hex file
Write target option bytes now!
Write target FLASH now!

This section of the **RLink_ST7 options** dialog box carries out the instant actions listed below without leaving this dialog box. This is useful for testing connections and retrieving information from the RLink and your ST7, as well as for programming the ST7 and its option bytes.

- **Connect to RLink and read serial number** is useful for checking that RLink is working and properly connected and that the USB driver is correctly installed. It also allows you to read the RLink serial number, which you will be asked for if you contact our support team.
- **Read Calibration values** reads current values for calibration, address and frequencies. (for ST7FOX devices only)
- **Read target option bytes** allows you to read the option bytes currently written in the chip. Use this also to test the connections and power of the target ST7.
- **Write target option bytes now!** allows to program the option bytes without leaving the configuration window. When programming them, Ride7 takes into account the settings in the **Option bytes options** section of the dialog box (see the previous section). It will do nothing if the "Leave as is" option is selected. It will erase and write the default value if the **Restore default** option is selected, and it will program the value displayed in the edit fields if **Program** is selected.
- **Erase target now!** allows you to completely erase the target's Flash (writing 0xFF), option bytes (restoring the default value) and EEPROM (if your ST7 features any). This is the correct way to remove the read-out protection from a protected device.
- **Dump target FLASH to hex file** reads the contents of the Flash and writes it in a file in hex format whose name is derived from the current application's name with the extension .hex (*<application name>.hex*).
- **Write target FLASH now!** programs the Flash with the current application's hex file generated by the linker. Then, launches the execution in user mode. When using this instant action, the code is not patched for debug, even if the **Debug option** is checked.

6.5.3 RC calibration

This setting applies only to the ST7 Fox family.

At each erase, calibration automatically starts.

Current values for calibration, address and frequencies can be read via the **Instant actions** menu.

6.5.4 Jumpers for RLink ST7 using an adapter

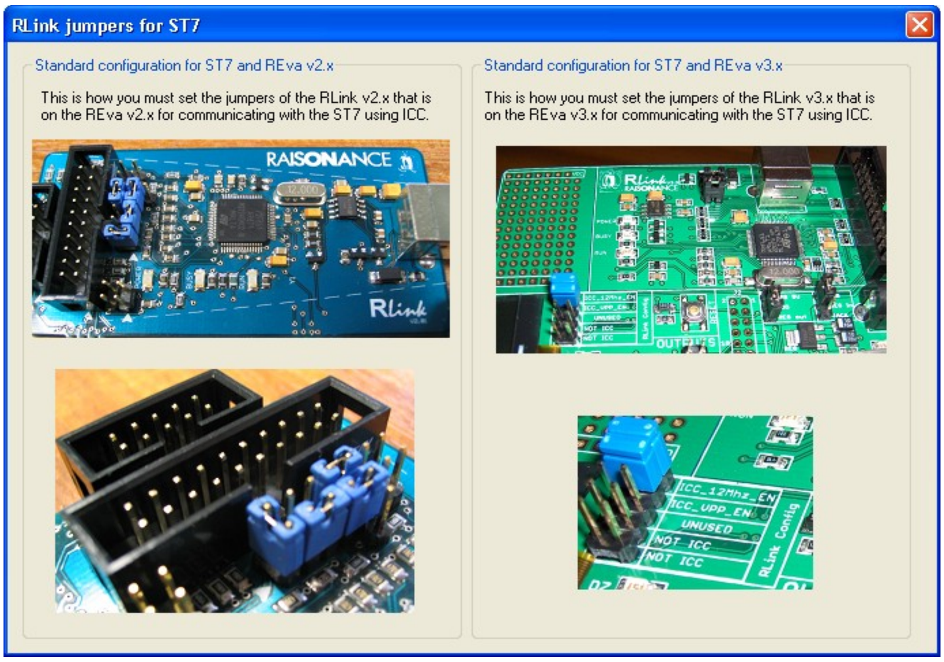
If you are using the version 3.0 of RLink or earlier, which feature a 24-point connector, you must use an adaptor (ADP) to convert the 24-point RLink connector into a 10-point ICC connector to connect to ST7 devices. These ADPs contain jumpers that you may want to set, depending on your situation.

ADPs ICC-ST7 and ST7-STM8 all allow to connect to ST7 devices. Some of them also allow to connect to STM8 devices. Depending on the versions, these ADPs feature up to four jumpers:

- Jumper “SWIM” for selecting SWIM or ICC protocol.
- Jumper “PW5V” for transferring the 5V power from RLink to the target ST7 board. Use this if your board has no power and you want to power it from the RLink. !!! ONLY DO THIS IF YOUR TARGET BOARD IS 5V-COMPLIANT !!!
- Jumper “Adapt” which is for STM8 and should not be set when connecting to ST7.
- Jumper “12MHz” for sending the 12MHz clock to the target ST7. Use it if your target ST7 board does not feature a clock (crystal or oscillator) and you want RLink to provide one.

6.5.5 Jumpers for ST7 REva board

If you are using a REva board, you must make sure that the jumpers are set correctly on the RLink part of the board. To do this, click on **View RLink jumpers configuration for ST7**. The following illustrations showing the ST7 configuration for the RLink jumpers is displayed:



Standard configuration for ST7 and REva v2.x

This is how you must set the jumpers of the RLink v2.x that is on the REva v2.x for communicating with the ST7 using ICC.

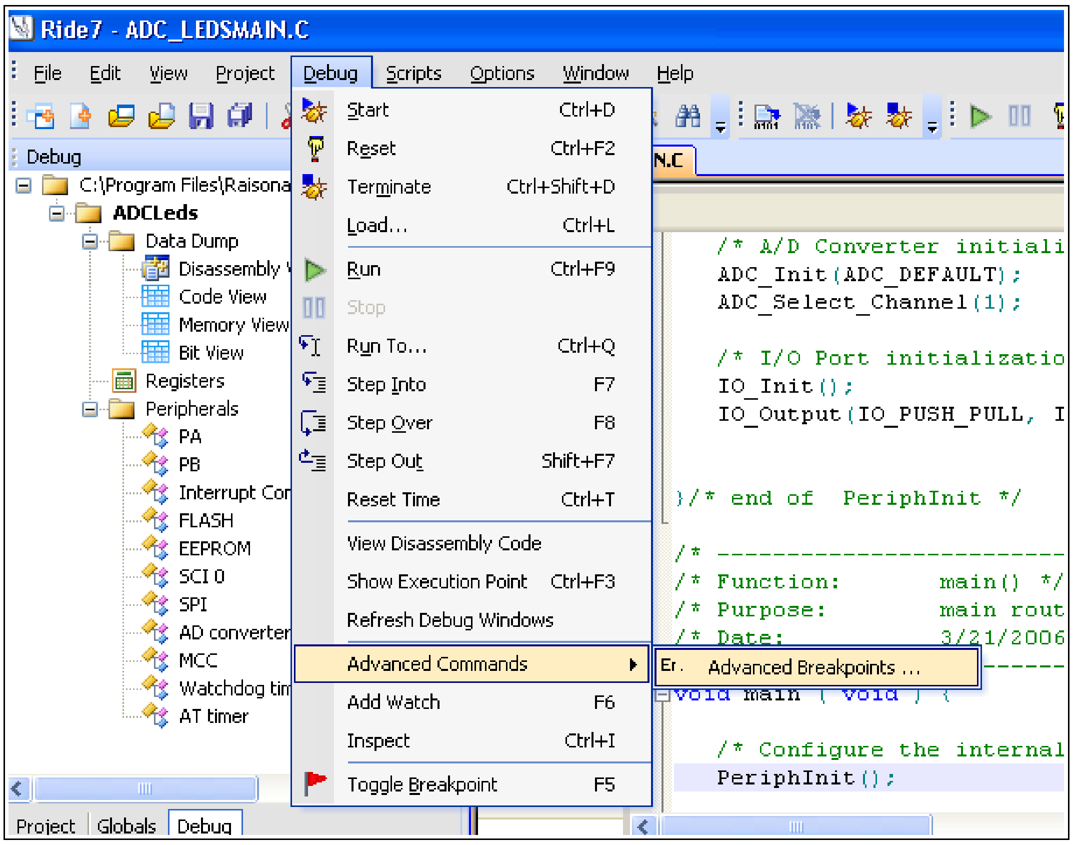
Standard configuration for ST7 and REva v3.x

This is how you must set the jumpers of the RLink v3.x that is on the REva v3.x for communicating with the ST7 using ICC.

Note: If the pictures in this documentation and in Ride7 are different, please assume that those shown in Ride7 are correct. If you purchased RLink as part of an ST7 kit (such as the REva board for ST7), then the jumpers should already be correctly set. For this reason, you should only need to adjust these jumpers if they were accidentally unplugged, or if you are using an RLink that was configured for another protocol, such as JTAG.

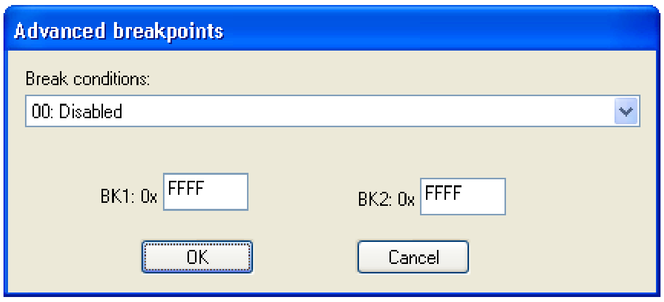
6.5.6 Advanced breakpoints

The interface for operating the debugger is exactly the same as for the simulator. However, there is one feature that is specific to the ST7 devices with a debug module: the advanced breakpoints.



```
    /* A/D Converter initiali
    ADC_Init(ADC_DEFAULT);
    ADC_Select_Channel(1);

    /* I/O Port initializatio
    IO_Init();
    IO_Output(IO_PUSH_PULL, I


}/* end of  PeriphInit */

/* -------------------------
/* Function:        main() */
/* Purpose:         main rout
/* Date:            3/21/2006
                    ---------
void main ( void ) {

    /* Configure the internal
    PeriphInit();
```

Once the debug session has started, if your ST7 device features a debug module, click on **Debug > Advanced Commands > Advanced Breakpoints** to open the **Advanced breakpoints** window which allows you to select the break conditions and addresses. (If this option does not appear, then your device probably does not have a debug module. You can check this on your device *Datasheet*.)



Break conditions: 00: Disabled

BK1: 0x FFFF   BK2: 0x FFFF

The advanced breakpoints configuration is discarded upon reset.

Note: If you use the debug module to set advanced breakpoints, the debugger cannot use them and may not be able to set standard breakpoints (if the target ST7 has HDFlash or if the breakpoint is in sector 0 of an XFlash device, then Ride7 cannot set the breakpoint). To let Ride7 use the debug module for standard breakpoints again, you must go to the Advanced breakpoints window and disable them.

6.6 Hints and troubleshooting

6.6.1 Example projects

The examples in the Ride7 directory REva folder are configured for use with a REva evaluation board, which includes an RLink. For standard ST7 installations they are found at:

C:\Program files\ Raisonance\Ride\Examples\ST7\REva.

For standard STM8 installations they are found at: *C:\Program files\ Raisonance\Ride\Examples\STM8\REva.*

These examples can be used with other demonstration and evaluation boards with a standard SWIM or ICC connector and the RLink. These examples can be compiled using the C toolchain during the evaluation period (30 days, no code size limitation) or with a purchased compiler license (Lite or Enterprise). Some examples are too big to be compiled with the demo version. These particular examples include a precompiled version of the application that you can download, run and debug, but you will not be able to modify and recompile it.

Before using an example, look at it and make sure that the jumpers on the REva board are set correctly (enables for the LEDs, buttons, SCI, EEPROM, etc.). Usually, there is some important information in comments at the beginning of the main file (i.e. the file that contains the `main` function).

6.6.2 Testing USB driver, connections and power supplies

To test the USB driver installation and RLink operation, use **Connect to RLink** instant action. RLink appears in Windows' device manager under the **Jungo** section if it is correctly recognized in Windows XP, NT and later. It appears under the RLinkWinUSBClass section in Windows Vista and earlier.

To test the connections and power of the target board and STM8/ST7, use the **Read option bytes** instant check. This operation requires RLink to connect to the target STM8/ST7, ensuring that it is powered, correctly connected to RLink, and that the rest of the application board does not interfere with the communication between RLink and the STM8/ST7 (see below).

6.6.3 Debug pins

The STM8 uses the RST and SWIMDATA pins. The ST7 uses the RST, ICCCLK and ICCDATA.

These pins communicate between the RLink and the target STM8/ST7. So you must ensure that the rest of the system (i.e. the other components on the board) does **not** use them. This also means that your application cannot use these pins if you plan to debug it with RLink.

These pins' addresses depend on the target STM8/ST7. Please refer to Chapter 6 Building an ICD-compliant application board and your *STM8/ST7 device Datasheet* for more information.

6.6.4 Handling option bytes for ST7

Here are the suggested ways to handle the option bytes, depending on the state of your project:

- While you are still debugging the application, you will probably re-program the target ST7 quite often, but the option bytes values will not change much. Since you are always using the same device, there is no need to re-program the option bytes every time. In this case, program them once using the **Write target option bytes now!** instant action, and then select the **Leave as is** option for the debugging sessions.
- When you are in the pre-production phase (debugging is complete and you are programming multiple ST7s), select **Program**, so as to program the option bytes for each ST7 device (and uncheck the **Debug** option in the actions for debug session).
- Also, remember that if you protect the Flash memory against read-out (or write) with the option bytes, then you will not be able to debug. So even if you already know that your final application will be protected against read-out, you should not do so during the development phases of the project when you still need to debug.

6.6.5 Handling option bytes for STM8

In STM8 devices, the option bytes are simply mapped in the memory space (see the device datasheet from ST to know the addresses and contents of the option bytes in your particular STM8 derivative). The option bytes are accessed along with the Flash during the programming phases.

To program them, your compile chain must generate some data at these addresses in the application file. Then the option bytes are programmed whenever you program the application to Flash. See how it is done in the *STM8_OB.asm* file in this example:

<RideInstallDirectory>\Examples\STM8\REva\STM8S208RB\Toggle\...

If you do not want to program them, just leave these addresses unreferenced in the application.

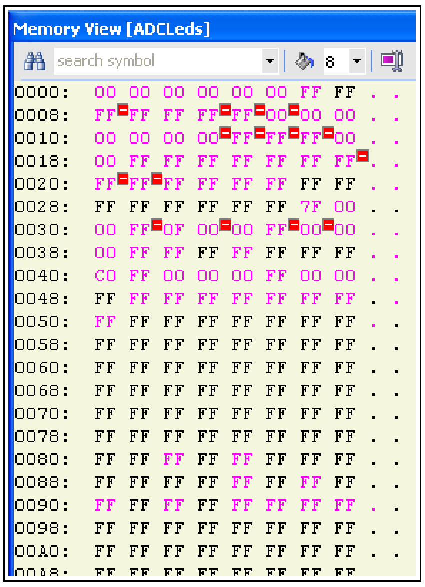
If you want to read them, just read the Flash out to a file using the **Instant actions** or RFlasher or *STM8_pgm.exe*. The option bytes are dumped along with the Flash and are seen in the resulting file at the corresponding addresses.

6.6.6 Command-line programming tool

In the Ride7 binaries directory (for standard installations *C:\Program Files\Raisonance\Ride\Bin*), you find programs named *ST7_pgm.exe and STM8_pgm.exe*. These executable files erase and program the STM8/ST7 connected to the RLink. Call one of them in a DOS prompt with no arguments in order to see the help that explains the command line arguments.

6.6.7 Protected addresses

You will notice, in the Code view, that some values have a red square with a white dash next to them. These are "protected addresses". These values are never updated, and you cannot modify them. This means that Ride7 never reads them because they are special peripheral registers that are modified whenever they are read. Any access would interfere with the execution of the application.



```
0000:  00 00 00 00 00 00 FF FF . .
0008:  FF FF FF FF FF 00 00 00 . .
0010:  00 00 00 00 FF FF FF 00 . .
0018:  00 FF FF FF FF FF FF FF . .
0020:  FF FF FF FF FF FF FF FF . .
0028:  FF FF FF FF FF FF 7F 00 . .
0030:  00 FF 0F 00 00 FF 00 00 . .
0038:  00 FF FF FF FF FF FF FF . .
0040:  C0 FF 00 00 00 FF 00 00 . .
0048:  FF FF FF FF FF FF FF FF . .
0050:  FF FF FF FF FF FF FF FF . .
0058:  FF FF FF FF FF FF FF FF . .
0060:  FF FF FF FF FF FF FF FF . .
0068:  FF FF FF FF FF FF FF FF . .
0070:  FF FF FF FF FF FF FF FF . .
0078:  FF FF FF FF FF FF FF FF . .
0080:  FF FF FF FF FF FF FF FF . .
0088:  FF FF FF FF FF FF FF FF . .
0090:  FF FF FF FF FF FF FF FF . .
0098:  FF FF FF FF FF FF FF FF . .
00A0:  FF FF FF FF FF FF FF FF . .
```

6.6.8 Limitations of RLink compared to the emulators

- Debugging STM8/ST7 devices through SWIM/ICC uses 5 bytes of stack and some I/O pins (SWIMDATA for STM8, ICCDATA and ICCCLK for ST7).
- It also uses about 195 bytes of Flash memory for ST7 devices with HDFlash, and ST7 devices with XFlash that do not feature the advanced version of the ROM-monitor (like the ST7Lite0. See the device datasheet).

6.6.8.1 ST7 monitor code

- Monitor code is loaded in the Flash along with the user application.
- It is loaded at the highest address where the application contains enough contiguous 0xFFs.

- The size of the monitor is 0xCC bytes.
- If the application does not contain enough contiguous blank bytes, then debug is not possible.

6.6.8.2 Watchdog and reset

- Activation of watchdog makes the monitor crash. You must disable watchdog when debugging.
- The **RESET** signal is controlled by the RLink during the whole debugging process. Therefore, the **RESET** button on the target board (if there is one) will probably not work and might even make the debugger crash if it is pressed. You should not use it.
- You can reset the CPU by using the **RESET** command in Ride7.

6.6.8.3 Limitations for ST7 XFlash targets without debug module (ST72F26x,ST7FLITE0/Sx)

(ST72F26x, ST7FLITE0x, ST7FLITESx):

- ICCCLK and ICCDATA lines communicate between the RLink and the target ST7. These lines are reserved and should not be used by the application being debugged.
- 5 stack bytes are used for communication between the RLink and the target. These stack bytes cannot be used by the application.
- A TRAP vector points to the dedicated RLink-loaded code that manages breakpoints. A breakpoint is a TRAP instruction patched into the application code. For this reason, the TRAP vector and TRAP instruction are reserved for RLink.
- During debugging, peripherals continue to run even when the application has stopped.
- It is not possible to set a breakpoint in sector 0. For this reason, sector 0 should be configured to its smallest size (0.5Kb) with the option byte.
- Pressing **Stop** while debugging in Ride7 is the same as a reset, and returns the program counter to the `main` function.
- Additional limitation for ST7FLITE0x and ST7FLITES2/5: 195 bytes in Flash memory are reserved for code that manages communication between the RLink and the target ST7. These bytes cannot be used by the application.
  This code is loaded when the application is programmed to the ST7.
  The programming algorithm places the code as high as possible in Flash memory.
  This code never overwrites the interrupt vectors. If there is not enough space for it, Ride7 returns an error message indicating that the user cannot debug the application.

6.6.8.4 Limitations for ST7 XFlash targets with debug module (ST7FLITE1/2/3x)

(ST7FLITE1x, ST7FLITE2x, ST7FLITE3x):

- ICCCLK and ICCDATA lines communicate between the RLink and the target ST7. These lines are reserved and should not be used by the application being debugged.
- 5 stack bytes are used for communication between the RLink and the target. These stack bytes cannot be used by the application.
- A TRAP vector points to the dedicated RLink-loaded code that manages breakpoints. A breakpoint is a TRAP instruction patched into the application code. For this reason, the TRAP vector and TRAP instruction are reserved for RLink.
- During debugging, peripherals continue to run even when the application has stopped.
- It is possible to set up to 2 breakpoints on sector 0, or 1 advanced breakpoint.

6.6.8.5 Limitation for ST7 HDFlash targets

- 5 stack bytes are used for communication between the RLink and the target. These stack bytes cannot be used by the application.
- 195 bytes in Flash memory are reserved for code that manages communication between RLink and the target ST7.
  These bytes cannot be used by the application.
  This code is loaded when the application is programmed to the ST7.
  The programming algorithm places the code as high as possible in Flash memory.
  This code never overwrites the interrupt vectors. If there is not enough space for it, Ride7 returns an error message indicating that the user cannot debug the application.

6.6.9 Limitations for ST7 HDFlash targets without debug module

Flash breakpoints

- Breakpoints in Flash are made by adding TRAP instructions in the application code at compile time. You can use the "Breakpoint" macro defined in *st7lib_conf.h* to do this easily.
- The trap interrupt cannot be used by the application when debugging.
- The trap ISR should be defined and do nothing, in order to avoid breakpoints doing a reset when just running the application (not debugging).
- RAM breakpoints can be set and cleared normally.

Step

- Stepping needs dynamic breakpoints. Therefore, it does not work with HDFlash devices without a debug module.
- Ride7 implements a half-simulated stepping feature called “virtual emulation”. This mode is automatically activated whenever you request a step (at C or assembler level) while debugging on an HDFlash target without a debug module. In this mode, Ride7 simulates the ST7 core just as if you were debugging using pure software simulation. But it updates the real CPU data and registers accordingly, and therefore the peripherals (GPIO, etc.) actually run on the real device, making it possible to interact with the rest of the system.
- This simulation mode is NOT real-time, and disables the interrupts if they were enabled. After that, when the program is run again using the **GO** command, the interrupts are re-enabled (if needed), and program execution resumes in the real CPU, providing real-time emulation again.

Code in RAM

- Executing a function in RAM is the best way to debug it.
- Look at the compile and link documentation for how to place code in RAM. In a future versions of Ride7, we will add a simple syntax in the compiler to easily place a function in RAM memory.

Stop

The **Stop** command needs an interrupt enabled (depends on the target), and additional configuration:

Stop for 361 and 561

- The ei1 interrupt on PB5 is used.
- The monitor automatically enables ei1, but it lets the user application enable the interrupts.
- The application must enable the interrupts using RIM for **STOP** to work.
- If the application disables ei1 or changes PB5 configuration, **STOP** will not work.
- Beware of `IO_Init` function that disables it.
- We advise to place a breakpoint after peripheral init, which enables the break on PB5 again.
- ei1 should be kept at the highest IT level.
- The other ei1 interrupts can still be used, but the monitor will add 2 instructions before entering the ISR (BTJF then JP).

Stop for 63B

- The external IT ITi is used.
- The user application has to use RIM for enabling interrupts and to enable the STOP.
- The user must plug a wire between ICCDATA and the signal associated with the enabled IT.
- It must be one of the 4 falling-edge-sensitive signals (PA6, PA7, PB6, PB7).
- Debugging prevents the application from using the ITi, even with the 7 other signals associated with this interrupt. Therefore, it is not possible to debug applications using ITi with this monitor.

Stop for 321, 324, 521

- External IT ei2 is used for stopping the execution.

- The user application has to use RIM for enabling interrupts and STOP.
- The user application has to configure one of the pins associated with ei2 (PB0,1,2,3) to generate IT on falling edge.
- The user must plug a wire between ICCDATA and one of the signal associated with the enabled IT.
- It must also configure the signal as input with interrupt on falling edge.
- The selected IT cannot be used for any other purpose than debugging.
- The user can choose to disable the STOP feature, in order to use the 4 PB0,1,2,3 pins and/or ei2 in his application. To do this, just don't plug the wire on ICCDATA.

7. ICD-compliant application board

In-Circuit Debugging (ICD) and In-Circuit Programming (ICP) are applications of the SWIM or ICC protocol developed by ST for STM8 and ST7 microcontrollers. With the necessary connection hardware, these protocols allow you to read and write to your STM8/ST7's Flash memory, and control the running of your application on your microcontroller.

These protocols are used by RLink and the other STM8/ST7 in-circuit debuggers and programmers.

To take advantage of SWIM/ICC, you must integrate a SWIM or ICC connector into your application hardware when you start developing your application. To help you implement ICP and ICD in the development of your application, this chapter provides a summary of points you should consider when installing an ICD connector.

7.1 SWIM connector for STM8

To connect to your application board for ICP and ICD, you must install a 4-pin SWIM connector and ensure the appropriate connections to your STM8.

This connector receives the SWIM cable and relays the signals required for ICP and ICD to your STM8.

The table below describes the SWIM connector and its pins usage:

Connector pin (and pin number)	**STM8 pin (see pin number on device datasheet)**	**Function**
VDD_APPLI (1)	VDD	Device power supply
SWIMDATA (2)	SWIMDATA	SWIM Input/Output serial data pin
GND (3)	GND	Ground
SWIMRESET (4)	RESET	Device reset

7.1.1 SWIMDATA pin

This pin transfers data between the RLink and the target STM8 microcontroller.

As soon as the programmer’s SWIM connector is connected to the application board, the SWIMDATA pin should not be used by other application devices, even if a SWIM session is not in progress. The RLink and its ADP include 50 Ohms serial resistors that provide a limited protection. However, there is no guarantee that this protection will be enough to prevent damage to occur in case of electrical conflict.

7.1.2 SWIMRESET pin

This pin resets the target STM8 microcontroller from the host PC through the RLink.

During a SWIM session, you must ensure that the RLink controls the STM8’s RESET pin so that no external reset is generated by the application board. This can lead to a conflict if the application reset circuitry signal exceeds 5mA (push-pull output or pull-up resistor <1KΩ). To avoid such conflicts, a Schottky diode can be used to isolate the application reset circuit.

You can place a capacitor on the RESET pin, but it should not be too large; if the rising time of the reset signal is too long, RLink might think that the device is not connected or not operational and programming or debugging is impossible. 0.1µF should be fine. Larger values should be avoided.

7.1.3 VDD_APPLI pin

This pin is used by tools with a power supply follower, such as RLink. This connection is needed by the power supply follower to detect the supply voltage and power the RLink's I/Os accordingly.

7.2 ICC connector for ST7

To connect to your application board for ICP and ICD, you must install a 10-pin, HE-10 type connector (also called an ICC connector) and ensure the appropriate connections to your ST7.

This connector receives the ICC cable and relays the signals required for ICP and ICD to your ST7.

The table below describes the ICC connector and its pins usage:

Connector pin (and pin number)	**ST7 pin (see pin number on device datasheet)**	**Function**
GND (1, 3, 5, 10)	GND	Ground
VDD_APPLI (7)	VDD	Device power supply
ICCDATA (2)	ICCDATA	ICC input/output serial data pin
ICCCLK (4)	ICCCLK	ICC input serial clock pin
ICCRESET (6)	RESET	Device reset
ICCSEL/VPP (8)	ICCSEL/VPP	Programming voltage and ICC selection
ICCOSC (9)	OSC1 or OSCIN	Main clock input for external clocking (optional)

7.2.1 ICCDATA and ICCCLK pins

These pins transfer data between the RLink and the target microcontroller.

As soon as the programmer's ICC connector is connected to the application board, the ICCDATA and ICCCLK pins must not be used by other application devices, even if an ICC session is not in progress. The RLink and its ADP include 50 Ω serial resistors that provide a limited protection. However, there is no guarantee that this protection will be enough to prevent damage occurring in the case of electrical conflict.

For an ST7 without an ICCSEL pin, during normal operation the ICCCLK pin must be pulled-up internally or externally (10KΩ pull-up required in noisy environments). This is to avoid entering ICC mode unintentionally during a reset.

7.2.2 ICCRESET pin

This pin resets the target microcontroller from the host PC through the RLink.

During an ICC session, you must ensure that the RLink controls the ST7's RESET pin so that no external reset is generated by the application board. This can lead to a conflict if the application reset circuitry signal exceeds 5mA (push-pull output or pull-up resistor <1KΩ). To avoid such conflicts, a Schottky diode can be used to isolate the application reset circuit.

You can place a capacitor on the RESET pin, but it should not be too large; if the rising time of the reset signal is too long, RLink might think that the device is not connected or not operational, making it impossible to program and debug. 0.1µF should be fine. Larger values should be avoided.

7.2.3 ICCSEL/VPP pin

This pin is used on certain ST7 derivatives to supply a 12V programming voltage and/or to enter ICC (program) mode. The application should include a pull-down resistor not smaller than 10KΩ[1].

7.2.4 ICCOSC pin

This pin can be used for the RLink to provide an external clock to the target microcontroller.

If the clock is not provided by the application, or if the application clock source is not programmed in the option byte then the ICCOSC pin of the ICC connector must be connected to the ST7's OSC1 or OSCIN pin.

This connection allows you to start communication with your ST7 for in-circuit debugging and/or programming using the **Ignore Option bytes** option. When doing so, your RLink provides a clock source to initiate communication with the ST7.

The RLink provides a clock source at a frequency of 12MHz.

For ST7 devices with multi-oscillator capability, when the ICCOSC pin is connected, the OSC2 pin should be grounded.

7.2.5 VDD_APPLI pin

This pin is used by tools with a power supply follower, such as RLink. This connection is needed by the power supply follower to detect the supply voltage and power the RLink's I/Os accordingly.

8. Raisonance solutions for STM8/ST7 upgrades

8.1 RKit-STM8-Lite versus RKit-STM8-Enterprise

The RKit-STM8 capabilities are determined by a software-based license and are independent of the hardware that is used with them. Software licenses are node-locked licenses - specific to a computer. However, a dongle-based option is available when ordering.

- The RKit-STM8-Enterprise software license allows access to all features.
- The RKit-STM8-Lite has a limited build size of 32 Kbytes of output code.

8.2 RKit and RLink options

RKit-STM8-Lite

- All supported STM8A/L/S sub-families
- Raisonance compiler toolchain
- GUI interface for compiler control
- Project manager
- Debug run control, breakpoints and all views
- Full programming GUI
- Support via forums, email with standard priority

RKit-STM8-Enterprise (All features of the “Lite”, plus...)

- All supported ST7
- Script-based controls (C, C++, JScript)
- Control of version management tools (CVS, ...)
- Automatic C formatter
- Calculator (standard & hex)
- Comment stripper
- Unix line converter
- Support via forums, email with high priority

RLink-STD

- Debugging – No code size limitation
- Programming – No code size limitation

RLink-PRO

- Debugging – No code size limitation
- Programming – No code size limitation

9. Conformity

ROHS Compliance (Restriction of Hazardous Substances)

IoTize products are certified to comply with the European Union RoHS Directive (2002/95/EC) which restricts the use of six hazardous chemicals in its products for the protection of human health and the environment.

The restricted substances are as follows: lead, mercury, cadmium, hexavalent chromium, polybrominated biphenyls (PBB), and polybrominated diphenyl ethers (PBDE).

CE Compliance (Conformité Européenne)

IoTize products are certified to comply with the European Union CE Directive.

In a domestic environment, the user is responsible for taking protective measures from possible radio interference the products may cause.

FCC Compliance (Federal Communications Commission)

IoTize products are certified as Class A products in compliance with the American FCC requirements. In a domestic environment, the user is responsible for taking protective measures from possible radio interference the products may cause.

WEEE Compliance (The Waste Electrical & Electronic Equipment Directive)

IoTize disposes of its electrical equipment according to the WEEE Directive (2002/96/EC).

Upon request, KOLABS can recycle customer’s redundant products.

For more information on conformity and recycling, please visit the IoTize website *www.iotize.com*

10. Glossary

Term	Description
ADC	Analog Digital Converter
ADP	RLink adaptor
CodeCompressor	Post link code optimization tool
HDFlash	High Density Flash
I/O	Input/Output
ICC	In-Circuit Communication
ICD	In Circuit Debugging
ICP	In Circuit Programming
RBuilder	Application builder that allows users to configure device peripherals and out put the required C code automatically for their applications. Code is based on libraries provided by the manufacturer.
REva	Raisonance evaluation platform – modular evaluation boards with main evaluation board (motherboard) and daughter boards featuring different microcontrollers
RFlasher	Raisonance Flasher: Programming interface for user-friendly flash programming
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.
SCI	Serial Communication Interface
SWIM	Serial Wire Interface Module
XFlash	Extended Flash: Flash memory based on EEPROM technology

11. Index

Alphabetical Index

12. History

Date	Modification
Jul 2008	Initial version
Jan 2009	Some file names updated.
Feb 2009	STM8 information added. (only ST7 before)
Jan 2010	Corrected some references to menu entries. Cleaned up some pictures.
Feb 2010	Added description of new RLink STM8 options.
Jun 2011	Added description of new registration process.
06 Nov 2012	Put in new template
02 Apr 2013	Modified sections 1.4, 2.1, 3.2, 3.3, 8.1, 8.2.