

---

## Technical Notes

# National CR16B Family On-Chip Emulation

## Contents

Contents.....	1
1 Introduction .....	2
2 Emulation options.....	3
2.1 Hardware Options.....	3
2.2 Initialization Sequence .....	4
2.3 JTAG Scan Speed.....	6
3 CPU Setup.....	7
3.1 General Options.....	7
3.2 Debugging options.....	8
3.3 Advanced Options .....	9
4 FLASH programming.....	10
5 Real-Time Memory Access .....	10
6 Access Breakpoints .....	11
7 Getting Started.....	12
8 Troubleshooting.....	12

# 1 Introduction

The CR16 family uses the standard JTAG interface.

There are two interfaces available, depending on the core used. The CR16B core uses the SDI interface; the CR16C uses the SDI+ interface. There are changes in the structure of the interfaces; therefore two different interfaces are available from iSYSTEM.

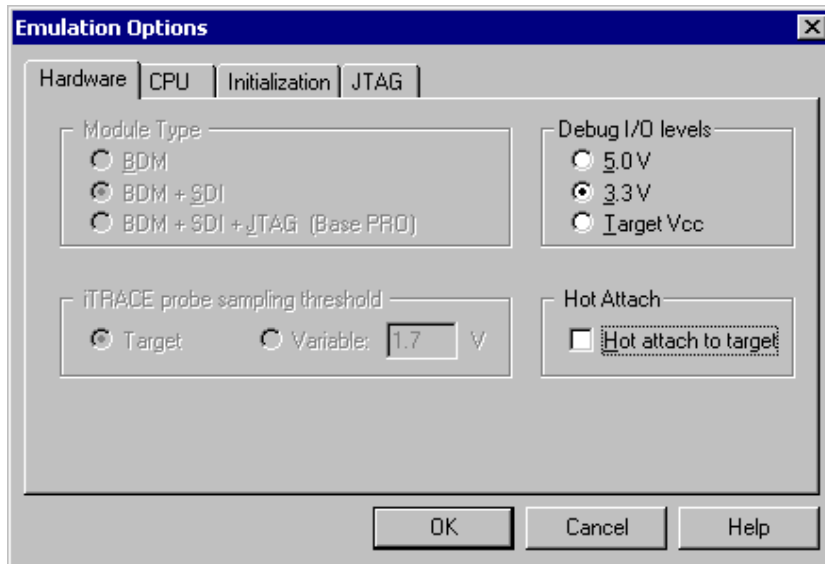
## Debug Features

The CR16B Emulation System features:

- Up to four hardware breakpoints
- Unlimited software breakpoints, including in the internal FLASH
- Access breakpoints
- Real-time access;
- Fast flash programming

## 2 Emulation options

### 2.1 Hardware Options



CR16 Emulation options, Hardware pane

#### **Debug I/O levels**

The development system can be configured in a way that debug (BDM/JTAG) signals are driven at 3.3V, 5V or target voltage levels. When 'Target Vcc' Debug I/O level is selected, a voltage applied to the belonging reference voltage pin (target debug connector) is used as a reference voltage for driving debug (BDM/JTAG) signals. Make sure that the target reference voltage pin is connected when 'Target Vcc' Debug I/O level is selected

#### **Hot Attach**

The JTAG module supports the Hot Attach function. This is a function, which enables the emulator to be connected to a working target device and have all debug functions available.

The procedure for Hot Attach:

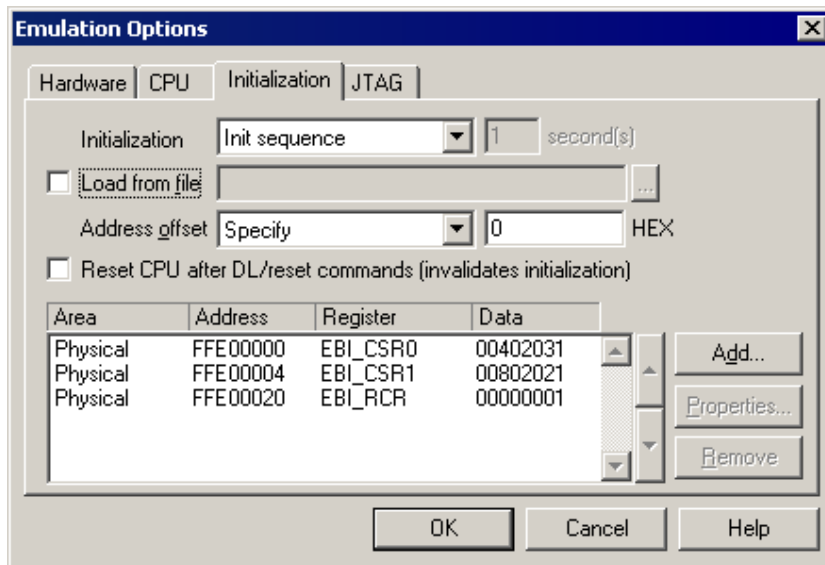
1. The target application should be running.
2. Hot Attach should be selected in the software.
3. A download should be performed, but without the JTAG cable connected. The emulator will be initialized and the ATTACH status will be shown.
4. Connect the JTAG cable.
5. Select the Attach option in the Debug menu. When this option is selected, the emulator tries to communicate through JTAG. If it is successful, it shows the STOP or RUNNING status. At this point, all debug functions are available.
6. When the debugging is finished, the CPU should be set to running and Detach selected from the Debug menu. The status shown is ATTACH. Now the JTAG cable can be safely removed.

## 2.2 Initialization Sequence

Before the flash programming or download can take place, the user must ensure that the memory is accessible. This is very important since there are many applications using memory resources (e.g. external RAM, external flash), which are not accessible after the CPU reset. In that case, the debugger must execute after the CPU reset a so called initialization sequence, which configures necessary CPU chip selects and then the download or flash programming can actually take place. The user must set up the initialization sequence based on his application.

The initialization sequence can be set up in two ways:

1. Set up the initialization sequence by adding necessary register writes directly in the Initialization page within winIDEA.



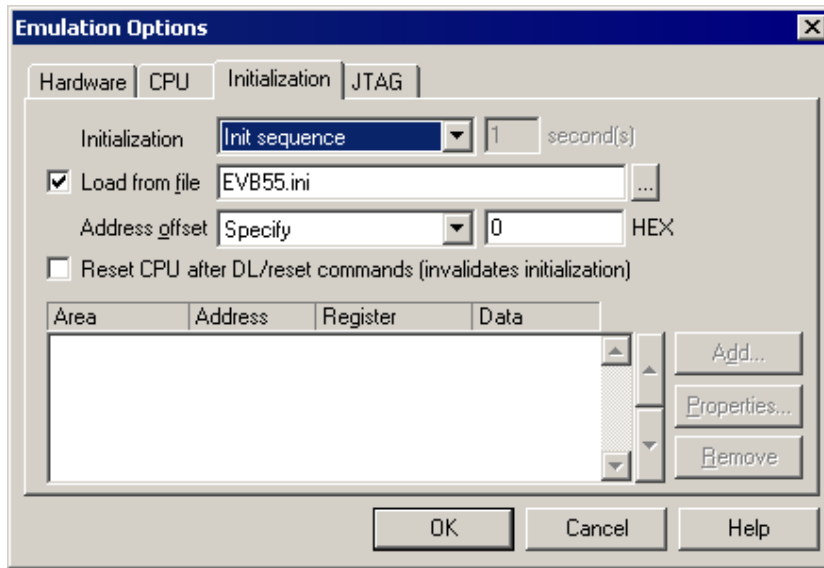
2. winIDEA accepts initialization sequence as a text file with .ini extension. The file must be written according to the syntax specified in the appendix in the hardware user's guide.

Excerpt from EVB55.ini file for the Atmel AT91M55800 CPU (ARM7TDMI):

```
S EBI_CSR0 L 0x00402031 // CS0 - ext. flash, 4 wait states
```

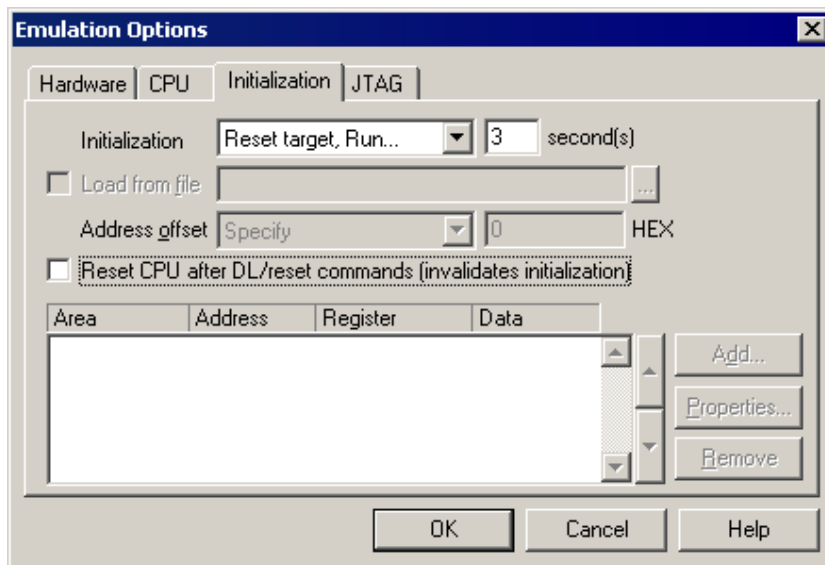
```
S EBI_CSR1 L 0x00802021 // CS1 - ext. SRAM
```

```
S EBI_RCR L 0x00000001 // remap internal RAM
```

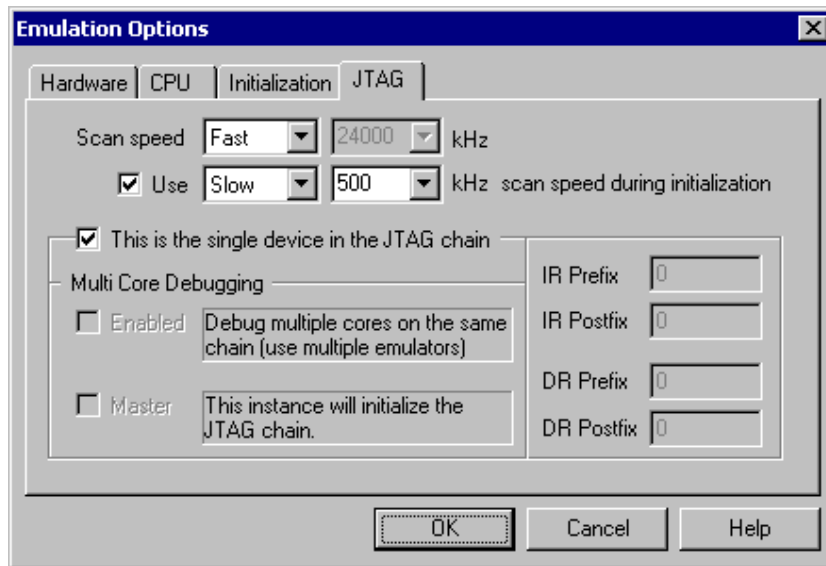


The advantage of the second method is that you can simply distribute your .ini file among different workspaces and users. Additionally, you can easily comment out some line while debugging the initialization sequence itself.

There is also a third method, which can be used too but it's not highly recommended for the start up. The user can initialize the CPU by executing part of the code in the target ROM for X seconds by using 'Reset and run for X sec' option.



## 2.3 JTAG Scan Speed



*JTAG Scan Speed definition*

### ***Scan speed***

The JTAG chain scanning speed can be set to:

- Slow - long delays are introduced in the JTAG scanning to support the slowest devices. JTAG clock frequency varying from 1 kHz to 2000 kHz can be set.
- Fast – the JTAG chain is scanned with no delays.
- Burst – provides the ability to set the JTAG clock frequency varying from 4 MHz to 100 MHz.
- Burst+ - provides the ability to set the JTAG clock frequency varying from 4 MHz to 100 MHz

Slow and Fast JTAG scanning is implemented by means of software toggling the necessary JTAG signals. Burst mode is a mixture of software and hardware based scanning and should normally work except when the JTAG scan frequency is an issue that is when the JTAG scan frequency used by the hardware accelerator is too high for the CPU. In general, selecting an appropriate scan frequency usually depends on scan speed limitations of the CPU. In Burst+ mode, complete scan is controlled by the hardware accelerator, which poses some preconditions, which are not met with all CPUs. Consequentially, Burst+ mode doesn't work for all CPUs.

In general, Fast mode should be used as a default setting. If the debugger works stable with this setting, try Burst or Burst+ mode to increase the download speed. If Fast mode already fails, try Slow mode at different scan frequencies until you find a working setting.

---

Note: Burst and Burst+ modes are implemented for PowerPC and ARM CPUs, including XScale.

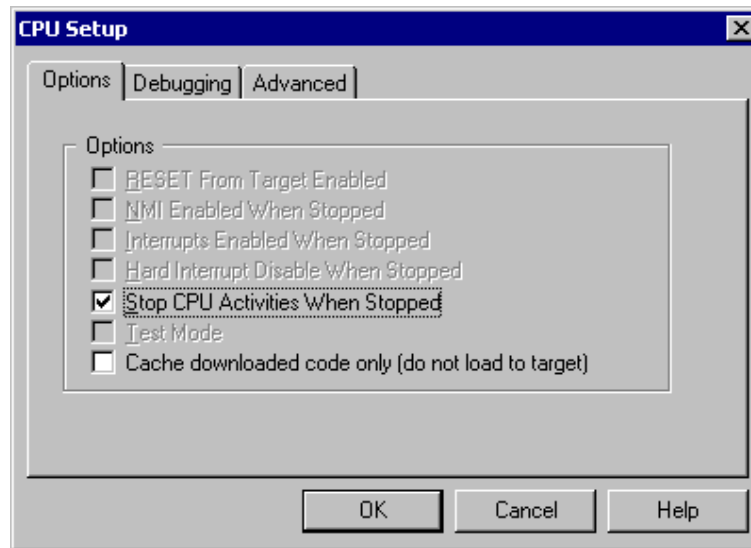
---

### ***Use – Scan Speed during Initialization***

On some systems, slower scan speed must be used during initialization, during which the CPU clock is raised (PLL engaged) and then higher scan speeds can be used in operation. In such case, this option and the appropriate scan speed must be selected.

## 3 CPU Setup

### 3.1 General Options



*CR16 Debugging options dialog*

#### ***Stop CPU Activities When Stopped***

When the option is checked, all internal peripherals like timers and counters are stopped when the application is stopped. Otherwise, timers and counters remain running while the program is stopped. Usually, when the option is checked, the emulation system behaves more consistently while stepping through the program. While being aware of the consequences, it is up to the user whether the option is checked or not.

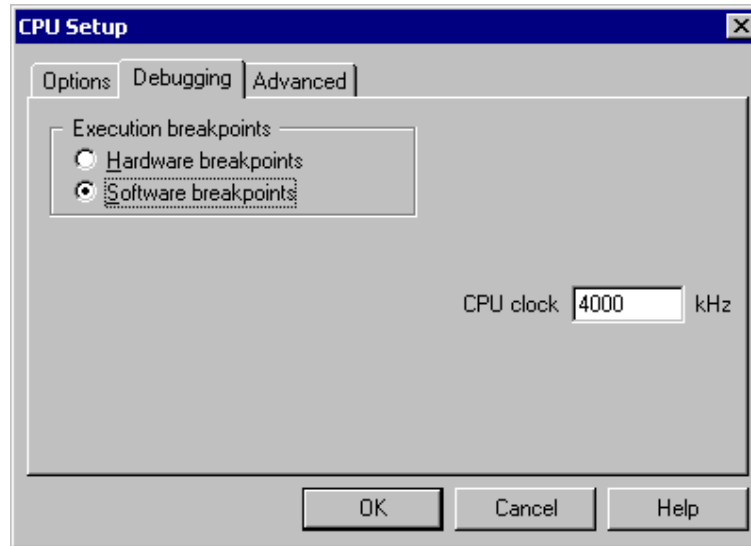
For instance, it's recommended that a timer, which generates interrupts, is stopped when the application is stopped. Otherwise, the CPU would first service all pending interrupts (generated by the timer while the application was stopped) after the application is resumed. Such behaviour is far away from the actual behaviour of the target application.

#### ***Cache Downloaded Code only (do not load to target)***

When this option is checked, the download files will not propagate to the target using standard debug download but the Target download files will.

In cases, where the application is previously programmed in the target or it's programmed through the flash programming dialog, the user may uncheck 'Load code' in the 'Properties' dialog when specifying the debug download file(s). By doing so, the debugger loads only the necessary debug information for high level debugging while it doesn't load any code. However, debug functionalities like ETM and Nexus trace will not work then since an exact code image of the executed code is required as a prerequisite for the correct trace program flow reconstruction. This applies also for the call stack on some CPU platforms. In such applications, 'Load code' option should remain checked and 'Cache downloaded code only (do not load to target)' option checked instead. This will yield in debug information and code image loaded to the debugger but no memory writes will propagate to the target, which otherwise normally load the code to the target.

## 3.2 Debugging options



*CR16 Debugging options dialog*

### ***Execution breakpoints***

#### *Hardware Breakpoints*

Hardware breakpoints are breakpoints that are already provided by the CPU. The number of hardware breakpoints is limited to four. The advantage is that they function anywhere in the CPU space, which is not the case for software breakpoints, which normally cannot be used in the FLASH memory, non-writable memory (ROM) or self-modifying code. If the option 'Use hardware breakpoints' is selected, only hardware breakpoints are used for execution breakpoints.

Note that the debugger, when executing source step debug command, uses one breakpoint. Hence, when all available hardware breakpoints are used as execution breakpoints, the debugger may fail to execute debug step. The debugger offers 'Reserve one breakpoint for high-level debugging' option in the Debug/Debug Options/Debugging' tab to circumvent this. By default this option is checked and the user can uncheck it anytime.

#### *Software Breakpoints*

Available hardware breakpoints often prove to be insufficient. Then the debugger can use unlimited software breakpoints to work around this limitation. Note that the debugger features unlimited software breakpoints in the CR16B internal flash too.

When a software breakpoint is being used, the program first attempts to modify the source code by placing a break instruction into the code. If setting software breakpoint fails, a hardware breakpoint is used instead.

### **Using flash software breakpoints**

A flash device has a limited number of programming cycles. Belonging flash sector is erased and programmed every time when a software breakpoint is set or removed. The debugger sets breakpoints hidden from the user also when a source step is executed. In worst case, a flash may become worn out due to intense and long lasting debugging using flash software breakpoints.

## CPU Clock

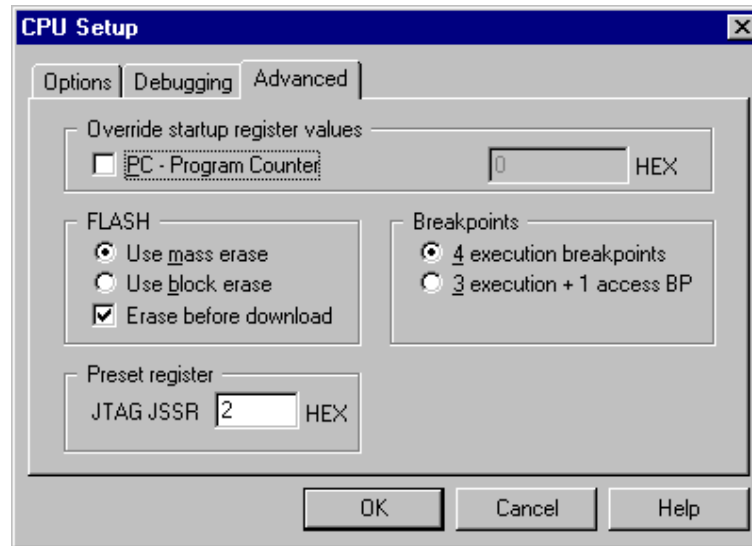
The CPU clock must be set here. A special counter is present on the CPU to ensure the correct programming frequency for the FLASH module, which is generated from the CPU clock. Since the CPU clock cannot be automatically detected, it must be set in this dialog.

---

Note: This setting only affects FLASH programming.

---

## 3.3 Advanced Options



*Advanced BDM Emulation options*

### ***Override startup register values***

This option overrides the default Program Counter reset value with the value set.

### ***Breakpoints***

Hardware breakpoints are breakpoints that are already provided by the CPU. The number of hardware breakpoints is limited to four. Available on-chip debug resources can be configured either all for four execution breakpoints or one access breakpoint and three execution breakpoints. The access breakpoint is configured in the 'Debug/Hardware Breakpoints' dialog.

### ***FLASH***

The way the FLASH is erased can be selected in this menu, whether block erase or mass erase is required. Regardless of the setting the whole FLASH is always erased.

### ***Erase before download***

If this option is selected, the FLASH is erased prior to every debug download. If this option is not selected, the data will be programmed during the debug download without prior erase.

### ***Preset Register – JTAG JSSR***

Certain CPUs have a special JSSR register, available only through JTAG. With this register the CPU can be started in different modes, certain self-test functions can be performed, watchdogs enabled or disabled, etc. The value of the register is set here.

## 4 FLASH programming

The internal CPU FLASH is programmed through standard debug download. The debugger recognizes which code from the download file fits in the internal CPU FLASH. All necessary FLASH programming settings are done in the 'CPU Setup/Advanced' dialog.

Standard FLASH setup dialog is required only for programming external flash devices.

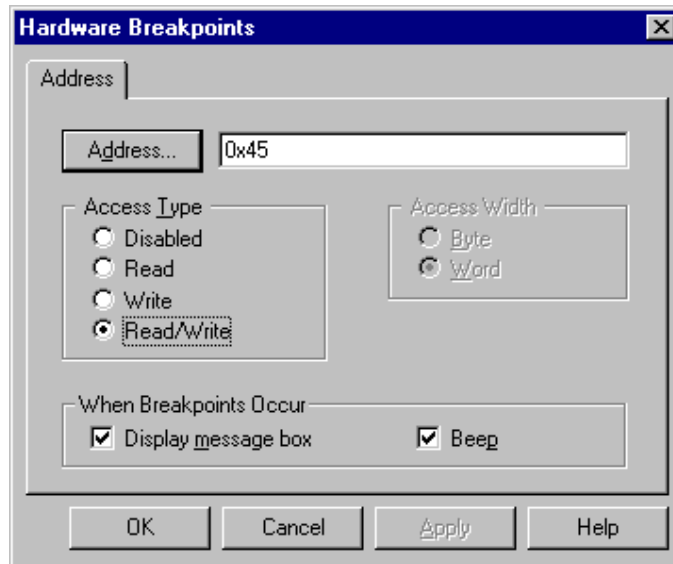
## 5 Real-Time Memory Access

With this type of CPUs, real-time memory access is available. Watch window's *Rt.Watch* panes can be configured to inspect memory without stalling the CPU. Optionally, memory and SFR windows can be configured to use real-time access as well.

Please refer to the Software User's Guide for more information on Real-Time watches.

## 6 Access Breakpoints

Four hardware breakpoints are available which can be used either as 4 execution breakpoints or 3 execution breakpoints and 1 access breakpoint. If a breakpoint operates as access breakpoint, the access type can be defined (read, write or read/write).



*Hardware Breakpoints menu, CR16B*

### ***Address***

The address of the access breakpoint should be entered here.

### ***Access Type***

The Access Type for the Access breakpoint is defined here. The access can be disabled, read only, write only or read/write.

### ***Access Width***

The Access Width can be a byte or a word long. In BDM/JTAG emulation, it is set automatically.

### ***When Breakpoints Occur***

A beep can be issued and/or a message displayed indicating that an access breakpoint has occurred.

## 7 Getting Started

- 1) Connect the system
- 2) Make sure that the target debug connector pinout matches with the one requested by a debug tool. If it doesn't, make some adaptation to comply with the standard connector otherwise the target or the debug tool may be damaged.
- 3) Power up the emulator and then power up the target.
- 4) Execute debug reset
- 5) The CPU should stop on location to which the reset vector points
- 6) Open memory window at internal CPU RAM location and check whether you are able to modify its content.
- 7) If you passed all 6 steps successfully, the debugger is operational and you may proceed to download the code in the internal CPU flash.
- 8) Check the 'Erase before download' option and 'Use mass erase' option in the 'FLASH' section in the 'CPU Setup/Advanced' tab.
- 9) Specify the download in the 'Debug/Files for download/Download files' tab.
- 10) Execute Debug download, which should download the code in the internal CPU flash.

## 8 Troubleshooting

When performing any kind of checksum, remove all software breakpoints since they may impact the checksum result.

Make sure that the power supply is applied to the target JTAG connector when 'Target VCC' is selected for Debug I/O levels in the Hardware/Emulator Options/Hardware tab, otherwise emulation fails or may behave unpredictably.

Try 'Slow' JTAG Scan speed if the debugger cannot connect to the CPU.

---

Disclaimer: iSYSTEM assumes no responsibility for any errors which may appear in this document, reserves the right to change devices or specifications detailed herein at any time without notice, and does not make any commitment to update the information herein.

© iSYSTEM. All rights reserved.