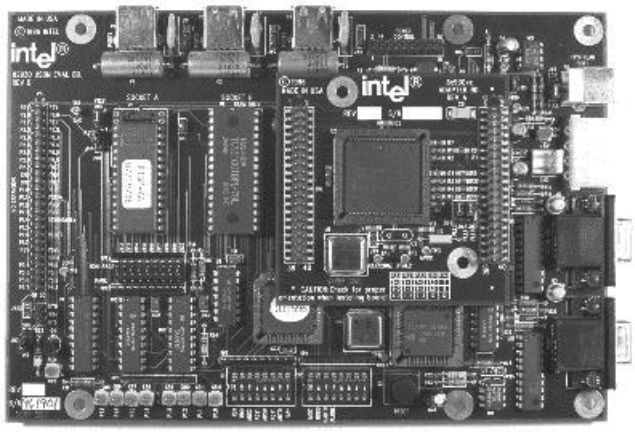
OVERVIEW



Intel provides the USBM Evaluation board for developers interested in the Universal Serial Bus (USB). The USBM includes the 8x930Ax USB device and the RISM251 monitor.

The Keil C251 Compiler, A251 Assembler, and dScope-251 Debugger fully support the Intel 8x930Ax USB device and the RISM251 monitor.

This application note describes a number of configuration details regarding the Intel board, the RISM monitor, and the Keil tools.

If you have questions this application note does not address, contact our technical support group at support@keil.com.

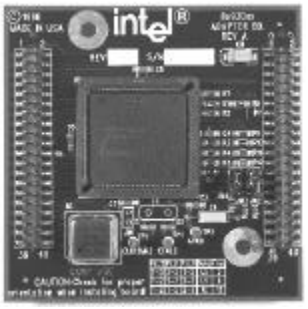
USBM EVALUATION BOARD CONFIGURATION

The USBM Evaluation Board from Intel lets you evaluate the 8x930Ax and 8x930Hx USB devices. The USBM includes the following:

- An EPROM in SOCKET A that contains the RISM251 monitor.
- A 128 Kbyte RAM in SOCKET B that is used for your target program and memory.
- The 8x930 device that is the 251-compatible USB part.
- DIP switches for simple board configuration.
- Serial port jacks for the internal 8x930 UART and for a 16550 external UART.
- LEDs on P1 that you may use to prove that the board really does something.
- USB upstream and downstream jacks.
- A RESET button.
- A green power LED.

There are a few things you should check out and know about the board before you begin.

8X930XX ADAPTER BOARD (DAUGHTER CARD)



The USBM evaluation board was originally designed for the 82930 USB device. This chip is now obsolete and should not be used. Intel released a daughter card that plugs into the USBM board on top of the 82930 socket. To install the daughter card, you must perform the following steps.

1. Unplug the power supply from the USBM board.

2. Remove the 82930 devices from the PLCC socket. We recommend using a chip extractor to accomplish this. However, you may also use a tiny screwdriver, a dentists' pick, or a car key (for a really small car).

3. Remove the RISM monitor EPROM from SOCKET A.

4. Plug the daughter card into the USBM making sure the IDC connectors on the bottom of the daughter card align with the IDC pins on the USBM.

5. Plug the new RISM monitor (ours is labeled RISM-AA 07/19/96) into SOCKET A. If your EPROM is a 28-pin device, make sure pin 14 is plugged into pin 16 of SOCKET A.

This completes installation of the daughter card. Once again, DO NOT USE the 82930 device and DO NOT USE the USBM without the daughter card and the new RISM monitor.

8X930AX AND 8X930HX DEVICES

In the prototyping stages of chip development, a number of 8x930 devices were manufactured that were only marginally functional or not functional at all. In developing software for the 930, we have noticed that some chips work and some do not. Make sure you are using the latest version of the chips available.

POWER SUPPLY

The USBM requires an external, regulated power supply. Be **extremely** careful connecting power to the USBM. Make careful note of the polarity of the plug since it is very easy to plug it in backwards. GND and +5V are clearly marked on the board. If the polarity of the plug is not correct, you may destroy the board.

DIP SWITCHES

The USBM has 2 sets of DIP switches and a number of jumpers you must configure. Typically, these are preset from the factory, but you should verify them anyway.

S1 Settings

S1-1 RD1	S1-2 RD0	S1-3 ADS2	S1-4 ADS1	S1-5 ADS0	S1-6 ADT1	S1-7 ADT0	S1-8 EA*
on	on	on	off	off	off	on	on

S2 Settings

S2-1 MOD1	S2-2 MOD0	S2-3 UARTC	S2-4 ALONE	S2-5	S2-6	S2-7	S2-8
on	off	on	on	off	off	off	off

JUMPER SETTINGS

The following jumpers on the USBM must be installed:

Jumper	Description
JW5	P12V
JW6	SHDN*
JW7	FULL
JW8	P5V
JW9	Short pin 1 and pin 2
JW10	Short pin 1 and pin 2
JW12	NON-PAGE
JW13	NON-PAGE
JW14	NON-PAGE

Jumper	Description
JW15	NON-PAGE
JW16	NON-PAGE
JW17	NON-PAGE
JW18	NON-PAGE
JW19	NON-PAGE
JW20	NON-PAGE
JW22	INT
JW24	INT0

In addition to the jumpers on the USBM, the following jumpers on the daughter card must be installed.

Jumper	Description
JW2	Short pin 2 and pin 3
JW3	Short pin 1 and pin 2
JW4	Short pin 1 and pin 2

EXTERNAL UART (16550)

The external 16550 UART at U14 uses the serial port connector at J2 to connect to your PC. This is the same chip used in the PC for serial communications and it should work well with any PC. This is the UART we recommend using.

Parameter	Specification
Baud Rate	19,200
Serial Jack	J2
Label	UART

When you debug using the external UART, the on-chip UART of the 8x930 is available for your target program to use.

ON-CHIP UART (8X930AX)

The on-chip UART uses the serial port connector at J1 to connect to your PC. Even though the on-chip UART will work, we recommend that you use the external UART instead.

Parameter	Specification
Baud Rate	9,600
Serial Jack	J1
Label	SERIAL I/O

Your target application may not use the on-chip UART when you debug using RISM251. This is not a limitation with the Keil MON251 monitor.

STAND-ALONE PROGRAMS

When you create stand-alone programs for the 930, you do not need to take any considerations for monitor programs or other obtrusive debugging tools. The following table lists several parameters that should be obvious from the User's Manuals for these chips.

Parameter	Specification
Reset Vector	Address 0xFF0000
Interrupt Vector	Address 0xFF0003
DATA Memory	0x000000-0x00007F
EDATA Memory	0x000000-0x00041F

RISM251 PROGRAMS

The RISM251 (Reduced Instruction Set Monitor 251) is the Intel monitor for the 251 and for the 8x930Ax and Hx. RISM is already programmed in the EPROM supplied with the board. RISM communicates with a PC-hosted debugger using the serial port of the PC and a UART on the USBM (either external or on-chip). Using RISM, you may download code, view memory, and single-step through your program.

Debugging programs using RISM requires that you first make a few changes to your program. The following table lists the parameters for debugging with RISM.

Parameter	Specification
Reset Vector	Address 0x004000
Interrupt Vector	Address 0x004003
DATA Memory	0x000000-0x00001F and 0x000040-0x00007F
EDATA Memory	0x000000-0x00001F and 0x000040-0x00041F
Reserved Memory	0x000020-0x00003F

MEMORY REQUIREMENTS

RISM is **not** a totally unobtrusive monitor. It does use some resources of the CPU. Specifically, you must be concerned with memory that RISM uses. Although this memory area seems to change with each revision, the area from 0x20 to 0x3F in the DATA area seems to be fairly consistent.

First, you must inform the linker that the space from 0x000020 to 0x00003F is reserved. The linker should not locate any variables in this area. Refer to the L251 Linker Configuration section below.

Second, you must modify the startup code so that the internal memory is **not** cleared. By default, the startup code for the Keil tools clears all internal memory from 0x000000 to 0x00041F. This includes the area from 0x000020 to 0x00003F. Clearing this area causes RISM to crash. Refer to the Startup Code Configuration section below.

RESET VECTOR (PROGRAM LOAD ADDRESS)

On reset, the 930 begins executing code at address 0xFF0000. When you burn an EPROM with your target program, this is where it must start.

When debugging using RISM, the monitor must begin at this address. Since two programs cannot occupy the same address space, your target program must be relocated. RISM assumes that your program loads at address 0x004000 and redirects the reset vector to that location. If you write relocatable programs, it is very easy to change the starting address using the Keil L251 Linker. Refer to the L251 Linker Configuration section below.

INTERRUPT VECTORS

The interrupt vector table for the 930 is located starting at address 0xFF0003. Each interrupt is offset by eight bytes. For example, interrupt 0 is at address 0xFF0003, interrupt 1 is at address 0xFF000B, interrupt 2 is at address 0xFF0013, and so on.

When debugging with RISM, the monitor redirects the interrupt vectors to your target program at offset 0x004000. For example, interrupt 0 is at address 0x004003, interrupt 1 is at address 0x00400B, interrupt 2 is at address 0x004013, and so on. If you write relocatable programs, it is very easy to change the starting address using the Keil L251 Linker. Refer to the L251 Linker Configuration section below.

MONITOR CAVEATS

Debugging using any MONITOR program has certain limitations that are not present in a simulator or a real-time emulator.

- The MONITOR is a software application that runs on your target hardware. You must be careful to work within the boundaries set by the MONITOR. For example, if the MONITOR uses a certain area of memory, your program must avoid that memory area.
- Stepping over code that changes the stack point will probably cause the monitor to crash. This is true because the monitor uses the hardware stack for its calls.

RISM CAVEATS

Debugging using RISM has certain limitations that you must consider before you begin debugging your program.

- RISM uses data memory from 0x000020 to 0x0003F. Your program must avoid this memory area.
- RISM uses the serial interrupt (at the highest priority level) to communicate with the host PC. You cannot debug other interrupts that run at the highest priority level.
- RISM expects that your target program starts at 0x004000 and that the interrupt vector table starts at 0x004003. You must take steps to locate your reset and interrupt vectors appropriately.

APBUILDER NOTES

The Intel ApBuilder program helps you get started generating code for the 930. Unfortunately, the code emitted by ApBuilder is not relocatable. This is OK if you don't mind changing your program each time you switch between debugging with RISM and making a stand-alone program in EPROM.

By writing relocatable programs, you can use the Keil L251 linker/locator to relocate your program automatically. Then, it becomes easy to switch between debugging with RISM and making a stand-alone program.

The following notes will help to convert the ApBuilder output

1. Copy REG930AX.INC from the Keil \C251\ASM directory and include it instead of 930AXREG.INC. 930AXREG.INC is not complete and its HEX constants do not include the requires starting digit (0-9).

2. Add the following lines to REG930AX.INC. These are 16-bit SFRs that are used by the program.

```
RXCNT DATA 0E6h
TXCNT DATA 0F6h
```

3. In ENUM_AX.ASM, change the include directive to be path insensitive. This makes examples that can be located in any directory.

4. In ENUM_AX.ASM, change the **`ORG 00:40??`** statements to **`CSEG AT 00??h`** statements. This does several good things:

   - It tells the assembler that the following stuff is a **CODE** segment (CSEG).

   - It gets rid of those pesky "Error 37" messages.

   - Rather than specifying an absolute address, it specifies an offset within the **CODE** segment. Why is this a big deal? Well, if you develop the code for the RISM board, you want the code to start at 00:4000h. However, if you develop a stand-alone program for your own hardware, you want it to start at 0FF:0000h. If you use absolute addresses in your program, then you have to make a RISM version and a stand-alone version of the program. You don't need to do that with the Keil tools if you make a relocatable program.

   - The linker can **easily** relocate the program to **any** location.

5. In ENUM_AX.ASM, put comments (;) in front of all of the COMMENT blocks so that the assembler doesn't try to interpret it as program stuff.

6. In ENUM_AX.ASM, locate the STATE and PHASE variable declarations and move them out of the middle of the program code. We suggest that you copy the following code above the RESET vector stuff and declare the STATE and PHASE variables to use **DATA** memory. This is the fastest way to go. However, since RISM uses some amount of **DATA** memory you may want to put these variables in **EDATA**.

```
    ;MYSEG   SEGMENT   DATA    ; DATA segment definitions
     MYSEG   SEGMENT   EDATA   ; EDATA segment definitions
     RSEG    MYSEG

    ; ---------
    ; Variables
    ; ---------
    ; STATE variable -- set in Check_State
    ; if 1, in control read
    ; if 2, control write
    ; if 3, no data control write
    STATE:   DS  1    ; reserve 1 byte

|    ; PHASE variable
    ; if 1, in setup phase
    ; if 2, in data phase
    ; if 3, in status phase
    PHASE:   DS  1    ; reserve 1 byte
```

KEIL TOOLS CONFIGURATION

There are essentially four components of the Keil C251 software that you must configure to use RISM and the Intel USBM evaluation board. They are:

- Startup Code,
- C251 Compiler,
- A251 Assembler,
- L251 Linker/Locator.

STARTUP CODE CONFIGURATION

The startup code for the Keil compiler is pre-assembled and stored in the library. If you do not explicitly include startup code in your project, the L251 linker will automatically include the startup code from the library.

Usually, the default startup code is acceptable for most programs. However, the default startup code causes RISM to crash. This is due to the memory clearing performed by the startup code.

To create programs for use with RISM, you must modify the startup code to leave memory intact. This is a 2-step process.

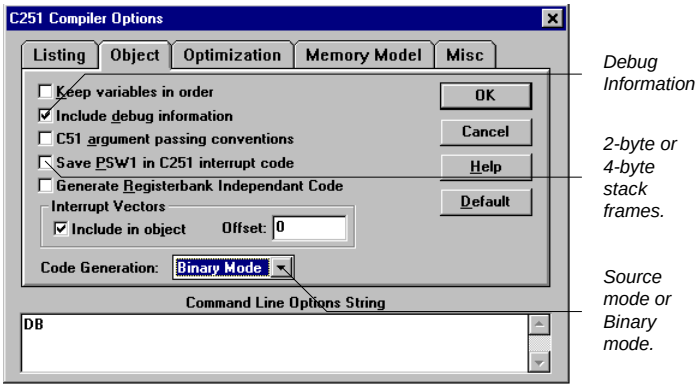
1. Copy the startup code (START251.A51) from the \C251\LIB directory to your project directory.
2. Edit your copy of START251.A51 and change EDATALEN from 420h to 0h as shown below.

```
;------------------------------------------------------------------------------
;
;  User-defined Power-On Zero Initialization of Memory
;
;  With the following EQU statements the zero initialization of memory
;  at processor reset can be defined:
;
;                            ; the absolute start of EDATA memory is always 0
EDATALEN     EQU   0H        ; the length of EDATA memory in bytes.
                             ; put 0h here to use RISM-251
;
XDATASTART   EQU   10000H    ; the absolute start-address of XDATA memory
XDATALEN     EQU   0         ; the length of XDATA memory in bytes.
;
HDATASTART   EQU   10000H    ; the absolute start-address of HDATA memory
HDATALEN     EQU   0         ; the length of HDATA memory in bytes.
;
;  Note:  The EDATA space overlaps physically the DATA, IDATA, BIT and EBIT
;         areas of the 251 CPU.
;
;------------------------------------------------------------------------------
```

C251 COMPILER CONFIGURATION

Configuring the Keil C251 compiler is easy using the µVision integrated development environment. There are only three items you must configure for the compiler.

- Include or Exclude Debug Information.
- Interrupt Stack Frame Size (2 bytes or 4 bytes).
- Binary Mode or Source Mode.



Debug Information

To include debug information like line numbers and symbolic information in the object file that C251 generates, make sure to check the **Include debug information** check box. There is really no reason to exclude debugging information, so this box should always be checked.

Interrupt STack Frame Size

The 930 chip can be configured to save 2 bytes or 4 bytes on the stack when an interrupt occurs. When saving 2 bytes, the lower 2 bytes of the program counter are saved. When saving 4 bytes, the lower 3 bytes of the program counter and the **PSW1** register are saved.

The C251 Compiler must know the frame size supported by your hardware so that it can take appropriate steps in the interrupt code it generates. For example, if the CPU saves only 2 bytes on the stack, the compiler must generate code to save the PSW register on the stack. However, if the CPU already saves the PSW, there is no need for the compiler to generate redundant code and waste space and time in your program.

If your CPU is configured to save only 2 bytes in an interrupt, you must check the **Save PSW1 in C251 interrupt code** check box. This instructs the C251 compiler to generate code to push and pop PSW1.

If your CPU is configured to save all 4 bytes in an interrupt, make sure the **Save PSW1 in C251 interrupt code** check box is unchecked.

Note that the Intel USBM evaluation board configures the CPU to save all 4 bytes in an interrupt.

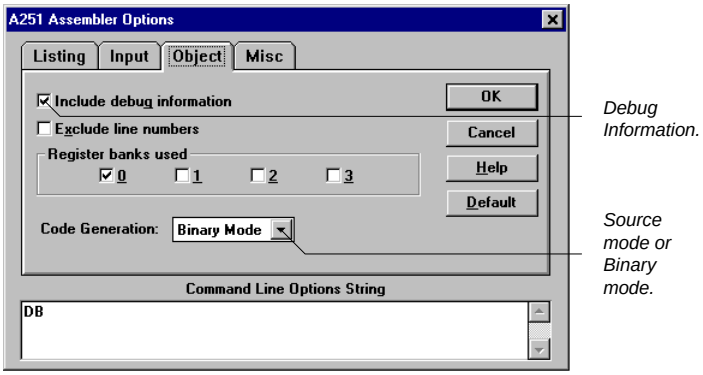
Binary Mode and Source Mode

The DIP switch and jumper setting specified above for the Intel USBM evaluation board specify binary mode. If you want to use source mode, you must change the appropriate DIP switches and the **Code Generation** method in the C251 Compiler Options dialog box.

A251 ASSEMBLER CONFIGURATION

Configuring the Keil A251 assembler is easy using the µVision integrated development environment. There are only two items you must configure.

- Include or Exclude Debug Information.
- Binary Mode or Source Mode.



Debug Information

To include debugging information like line numbers and symbolic information in the object file that A251 generates, make sure to check the **Include debug information** check box. There is really no reason to exclude debugging information, so this box should always be checked.

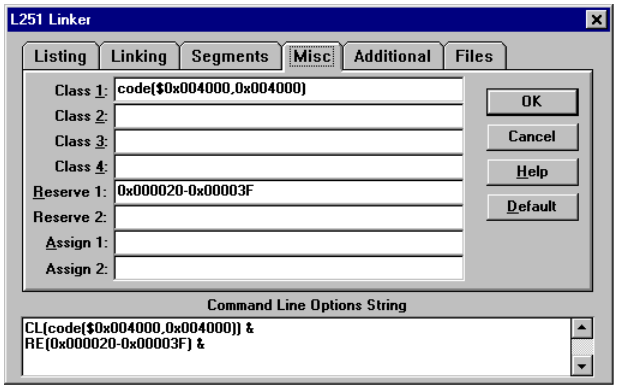
Binary Mode and Source Mode

The DIP switch and jumper setting specified above for the Intel USBM evaluation board specify binary mode. If you want to use source mode, you must change the appropriate DIP switches and the **Code Generation** method in the A251 Assembler Options dialog box.

L251 LINKER CONFIGURATION

To link your programs to use RISM, you must configure the linker to relocate the reset and interrupt vectors to start at 0x0040000 and you must reserve the memory used by RISM.

Controls in the L251 Linker Options dialog box under the Misc tab let you set segment class offsets and reserve memory areas.



Setting the Code Offset

To set the offset for the **CODE** class, you must enter the following in one of the **Class 1**, **Class 2**, **Class 3**, or **Class 4** input lines.

```
code($0x004000, 0x004000)
```

In this command, **`$0x004000`** tells the linker that the program counter starts at 0x004000. This is only used by the debugger to properly locate the program counter for non-standard programs (standard programs start at 0xFF0000).

The second **0x004000** tells the linker to offset all absolute **CODE** segments by 0x004000. In other words, code generated by the following assembly code…

```
CSEG at 0
NOP
```

is located at 0x004000 rather than at offset 0.

By default, the **CODE** segment is assumed to have an offset of 0xFF0000 (since that is where the 930 begins executing code after reset) and the program counter is assumed to begin at 0xFF0000.

To change your project to create a stand-alone program that will not use RISM, you may remove the **CODE** class setup shown above.

Reserving Memory

To reserve an area of memory you must enter the starting and ending addresses separated by a dash ('-') in either the **Reserve 1** or **Reserve 2** input line. To reserve the memory area from 0x000020 to 0x00003F (which is required by RISM), you must enter the following:

```
0x000020-0x00003F
```

To change your project to create a stand-alone program that will not use RISM, you may remove the memory reservation shown above.

CONCLUSION

Using relocatable programs makes configuring your USB programs for stand-alone operation and for RISM debugging relatively easy.

In the USA:
Keil Software, Inc.
16990 Dallas Parkway, Suite 120
Dallas, TX 75248-1903
USA

Sales: 800-348-8051
Phone: 972-735-8052
FAX: 972-735-8055

E-mail: sales.us@keil.com
support.us@keil.com

Internet: http://www.keil.com/

In Europe:
Keil Elektronik GmbH
Bretonischer Ring 15
D-85630 Grasbrunn b. Munchen
Germany

Phone: (49) (089) 45 60 40 - 0
FAX: (49) (089) 46 81 62

E-mail: sales.intl@keil.com
support.intl@keil.com