RTE-V821-PC SUPPLEMENTARY MANUAL NO.1 (REV.1.0)

1. APPLICATION USING MASKABLE INTERRUPT

This manual describes a method of developing an application using a maskable interrupt on the RTE-V821-PC and its restrictions.

1.1. INTERRUPT VECTOR

The interrupt vector area of FFFF-FE00H to FFFF-FFFFH of the V821 is a ROM area and cannot be rewritten. For the Multi monitor ROM, an alternative vector area is provided on SRAM. At a vector at any address between FFFF-FE00H to FFFF-FFFFH, a branch instruction to the alternative vector area is placed. When an interrupt of exception code FEE0H occurs, for example, the CPU interrupt function causes a branch to FFFF-FEE0H. At this address, a branch instruction to the location of an offset FFE0H in the alternative vector area is placed. A user program can branch to an interrupt handler upon an interrupt if the alternative vector area is rewritten like the original vector area. This program is different from usual V821 programs, which need not set (rewrite) the vector area that is generally fixed as a ROM area. The Multi program on the RTE-V821-PC must rewrite a vector before permitting an interrupt.

The alternative vector area extends from FF00-0000H to FF00-01FFH on SRAM. (The SRAM image is actually shown and can be referenced and modified from other addresses. An interrupt vector that is first referenced by the CPU branches to any address between FF00-0000H to FF00-01FFH.) When the interrupt of exception code FFE0H described above occurs, the branch instruction to the target interrupt handler is written at FF00-00E0H.

For the V821, the CPU contains a cache memory. Therefore, the CPU must flush the cache after rewriting a vector. If this operation is forgotten, the vector before the alternative vector is rewritten is used.

A sample program for rewriting an alternative vector is given below. (The relative address from the interrupt handler to the alternative vector area is within 26 bits, in this example.)

```
void SetAJump(int addr,  int jmpdest)  /* ←Vector setting routine */
/* int addr;             address where we're storing the 'jr' */
/* int jmpdest;          address where the 'jr' jumps to */
{
   int offset;
   unsigned inst;
   unsigned int *p ;

   offset = jmpdest - addr;
   inst = 0xa8000000 /* 'jr' opcode */ | (offset & 0x3ffffff);
   *((UINT16 *)(addr     )) = (inst >> 16) & 0xffff    ;
   *((UINT16 *)(addr + 2)) = (inst       ) & 0xffff    ;

   cache_clr() ;          /* cache flash ←Provided by ASM */
}
.....
void __interrupt IntEntry()  /* ←Interrupt handler */
{
.......
}
.....
main()
{
........
          SetAJump((int)((0xfffffee0-0xfffffe00)+0xfe000000) ,(int)IntEntry) ;
          /*               ↑Original vector address of the target interrupt */
```

1.2. GENERAL RESTRICTIONS AND NOTES

This section gives restrictions and notes on debugging an application using a maskable interrupt.

1) If an interrupt occurs before its alternative vector is set or after an incorrect alternative vector is set, a break occurs at that location of interrupt of the program. This is because the initial value of the alternative vector is a branch instruction to the break handler on monitor ROM.

2) If a relative address from the alternative vector area to the interrupt handler exceeds 26 bits, the following is needed to make a correct branch to the interrupt handler: At least one or more register values must be destroyed; Alternatively, a branch relay point must be created.

3) The alternative vector area is protected as an area managed by the ROM monitor. This area cannot be rewritten by downloading a program. Another possible rewriting method is to define the vector area on the source program as an independent section and to allocate the section to the alternative vector area by specifying corresponding parameters at a link. This method, however, is not practical because the program cannot be normally downloaded.

4) Flush the cache memory built-in to the CPU immediately after rewriting the alternative vector area. If this operation is forgotten, the vector before the alternative vector is rewritten is used.

5) All peripherals including interrupt relationships are initialized only when the reset switch on the board is pressed. When a program is reloaded and executed after the same program is started, the effect of the previous program execution remains on the peripherals. In order to avoid this problem, follow these steps when executing a program that uses peripherals from the beginning after executing the same program: (1) Disconnect rteserv. (2) Press the reset button of the RTE-V821-PC. (3) Re-connect rteserv.

6) The EI (interrupt permitted) state must be set after the DI (interrupt inhibited) state is set at the beginning of a program and after peripherals and vectors are set.

1.3. RESTRICTIONS AND NOTES ON USING A BREAKPOINT

A breakpoint can be set in an interrupt handler to break interrupt handling. The subsequent part of interrupt handling can be single-stepped. When using a breakpoint in this way, keep in mind the following restrictions and precautions:

1) During a break, no maskable interrupts are accepted.

2) The single step function sets a temporary breakpoint in the next instruction. While a user program in the EI (interrupt permitted) state is being single-stepped, an interrupt can be accepted.

3) The interrupt handler cannot be left by single-stepping. (To be more specific, the last step "}" of the interrupt handler cannot be single-stepped). The iret instruction cannot be single-stepped, either.

4) The return function of the debugger cannot cause a return from an interrupt handler to the original routine.