CUSTOMER NOTIFICATION

SUD-TT-0027-3-E
June 05, 2002
Koji Nishibayashi, Project Manager Microcomputer Group System LSI Solutions Engineering Div. NEC Electron Devices NEC Corporation

CP(K), O

IE-V850ES-G1 (Control Code: A, B, C)

Operating Precautions

Be sure to read this document before using the product.

Notes on Using IE-V850ES-G1

1. Product Version

Part number: IE-V850ES-G1

Control Code	EVA Chip	Usable Debugger
A	UPD703191AR DS4.5-3 V or higher	ID850 E2.31c or later
B	UPD703191AR DS4.5-3 V or higher	ID850 E2.31c or later
C	UPD703191AR DS4.5-3 V or higher	ID850 E2.31c or later

Use the latest debugger.

2. Product History

No.	Bugs and Changes/Additions to Specification	Control Code [Note]		
		A	B	C
1	The current product case is provisional	×	√	√
2	Restrictions on operating frequency	×	×	×
3	Restrictions on break timing when guard area is fetched	Permanent restriction		
4	ROM contents are rewritten if emulation ROM area is accessed for write.	Permanent restriction		
5	Restrictions on peripheral I/O register illegal break	Permanent restriction		
6	Restrictions on programmable I/O space	Permanent restriction		
7	Hardware break does not occur even if breakpoint is set.	Permanent restriction		
8	Restrictions related to access address during DMA trace	Permanent restriction		
9	Restriction on DBPC and DBPSW access during a break	Permanent restriction		
10	Restriction on DBTRAP instructions	Permanent restriction		
11	Restriction on access data traced by DMA	Permanent restriction		
12	Restriction on peripheral I/O register read access during break	Supported by debugger		
13	Restriction on target operating voltage	×	√	√
14	Improvement of power supply block safety	×	×	√

√: Already corrected
×: Corresponding restriction exists
–: Restriction does not apply

Note The “control code” is the second digit from the left in the 10-digit serial number in the warranty supplied with the in-circuit emulator you purchased (if it has not been upgraded). If the in-circuit emulator has been upgraded, a label indicating the new version is attached to the in-circuit emulator and the x in V-UP LEVEL x on this label indicates the control code.

3. Details of Bugs and Additions to Specification

No. 1 The current product case is provisional.

[Description]

The current product case of the IE-V850ES-G1 is provisional.

[Workaround]

There is no workaround. This bug has been corrected in control code B and later products.

No.2 Restrictions on operating frequency

[Description]

The maximum operating frequency is 20 MHz.

[Workaround]

There is no workaround. Use a frequency of 20 MHz or lower.

No.3 Restrictions on break timing when guard area is fetched

[Description]

When program execution enters the guard area, one instruction in the guard area is executed and then a break occurs.

[Workaround]

There is no workaround. Regard this as a permanent restriction.

No.4 ROM contents are rewritten if emulation ROM area is accessed for write.

[Description]

An illegal break occurs if the emulation ROM area is accessed for write, and the data of ROM is rewritten.

[Workaround]

There is no workaround. Regard this as a permanent restriction.

No.5 Restrictions on peripheral I/O register illegal break

[Description]

Peripheral I/O register illegal break is detected by “address condition + R/W attribute”. However, a break occurs if an 8-bit I/O register is written in halfword units (break does not occur if an 8-bit I/O register is read in halfword units).

[Workaround]

There is no workaround. Regard this as a permanent restriction.

No.6 Restrictions on programmable I/O space

[Description]

a) If a programmable I/O space is mapped to the higher 32 MB area, the programmable I/O space cannot be accessed during break.
b) If the programmable I/O space is accessed during program execution, a peripheral I/O register illegal break occurs.

[Workaround]

a) Map the programmable I/O space to the lower 32 MB area.

b) Map the area where the programmable I/O space exists to the emulation memory or target memory.

Regard this as a permanent restriction.

No.7 Hardware break does not occur even if breakpoint is set.

[Description]

- (Dis)assembly level -

If breakpoints are set for two instructions in a row and a break occurs at the first instruction, a break does not occur at the second instruction in response to the subsequent request for resumption of execution.

```
0x80049c  mov   r9, r10          ← Setting of breakpoint
0x80049e  add    r7, r10         ← Setting of breakpoint
0x8004a0  addi   1, r10, r17
```

- Source level -

If breakpoints are set for two execution statements in a row (of which each is expanded for a single instruction) and a break occurs at the first statement, a break may not occur at the second statement in response to the subsequent request for resumption of execution.

```
10  a = b;    (mov r9,r10)        ← Setting of breakpoint
11  a += c;   (add r7,r10)        ← Setting of breakpoint
```

[Cause]

If resumption of execution is requested at the position where program execution is stopped at a breakpoint, the instruction at the breakpoint is internally executed in one instruction step, and execution is resumed.

Some CPUs execute two instructions at a time, depending on the combination of instructions. In the above (dis)assembly level example, execution is resumed from address 08004a0. Therefore, the breakpoint set at address 0x80049e is not hit.

[Combination of instructions for which two instructions are simultaneously executed]

- Conditions in which “mov + operation instruction” are executed as one instruction

If dst of mov and dst of the operation instruction are the same register, except when that register is r0, in the combination “mov src, dst” and the following instruction:

Format I	satsubr/satsub/satadd/mulh
	or/xor/and
	subr/sub/add
Format II	shr/sar/shl/mulh

This combination of instructions is executed as one instruction only when the mov instruction is at the first position.

- Condition for parallel execution of instructions

(1) Combination of following instructions and br

Format I	nop/mov/not/sld
	satsubr/satsub/satadd/mulh
	or/xor/and/tst
	subr/sub/add/cmp
Format II	mov/satadd/add/cmp
	shr/sar/shl/mulh
Format IV	sld.b/sst.b/sld.h/sst.h/sld.w/sst.w

(2) Combination of following instructions (instructions in 1 excluding those that update flags) and bcc instruction except br

Format I	nop/mov/sld*
	mulh/sxb/sxh/zxb/zxh
Format II	mov/mulh
Format IV	sld.b/sst.b/sld.h/sst.h/sld.w/sst.w

(3) Combination of following instructions and sld

Format I	nop/mov/not
	satsubr/satsub/satadd/mulh
	or/xor/and/tst
	subr/sub/add/cmp
Format II	mov/satadd/add/cmp
	shr/sar/shl/mulh

In (1) to (3) above, parallel execution is performed only when br/bcc/sld instruction is at the second position of the above combination of instructions.

In the following cases, parallel execution is not performed even in the above combination.

a) If the first instruction is the first instruction after branch to non-word align
b) If the second instruction is sld and if writing to the register of ep is not completed (short path is not performed)

<<Example>>
Two instructions are not executed at the same time if instructions are programmed as follows.

```
0x1000          nop
0x1002          nop
0x1004          nop
0x1006          mov r10, ep          ← Setting of breakpoint
0x1008          sld.b 0x8[ep], r11   ← Setting of breakpoint
0x100c          nop
0x100e          nop
```

In this case, the mov instruction at address 0x1006 writes the value of r10 to the ep register. When the sld.b instruction at address 0x1008 is executed, however, WB (write back) of the mov instruction is not completed and therefore, the two instructions are not executed at the same time.

c) If the second instruction is bcc (conditional branch instruction) and flag hazard occurs (if there is a possibility that the flag is updated immediate before or by the preceding instruction)

<<Example>>
Two instructions are not executed at the same time if the instructions are programmed as follows.

```
0x1000      nop
0x1002      nop
0x1004      nop
0x1006      cmp r0, r10      ← Setting of breakpoint
0x1008      bn 0xf0          ← Setting of breakpoint
0x100a      nop
0x100c      nop
```

Because the S flag is changed by the cmp instruction at address 0x1006, the bn instruction that references the S flag and branches must wait for execution of the cmp instruction. Consequently, a flag hazard occurs when the bn instruction is executed, and two instructions are not executed at the same time.

d) If sld is used and the load buffer of both the instructions are in the WB wait status

<<Example>>
Suppose the following instructions are located in memory

```
0x1000          nop
0x1002          nop
```

```
0x1004      ld.w 0x3000[r10], r11
0x1008      ld.w 0x3004[r10], r12
0x100c      mov r8, r9              ← Setting of breakpoint
0x100e      sld.b 0x10[ep], r13     ← Setting of breakpoint
```

At this time, several wait state clocks are inserted if ld.w at addresses 0x1004 and 0x1008 accesses the external memory. When address 0x100e is executed, therefore, WB of the ld.w instruction at addresses 0x1004 and 0x1008 is not completed and "WB wait" status begins. Consequently, the two instructions at addresses 0x100c and 0x100e are not executed at the same time.

*** Formats I, II, and IV are the instruction formats shown in the User's Manual (Architecture) of the processor.**

[Workaround]

- The workaround used when setting software breaks can be applied to the following debugger versions.
  - NEC debugger: ID850 E2.20f and later versions
  - GHS Multi: Use EX85032.DLL version 5.40 or later
- There is no workaround for hardware breaks.
  Regard this as permanent restriction.

* Use the latest debugger.

No.8 Restrictions related to access address during DMA trace

[Description]

If DMA is started while the internal RAM is accessed or the program in the internal RAM is executed, either the source address or destination address of DMA will become 3FFExxxh, indicating an internal RAM address for either of the above or for trace data.

[Workaround]

There is no workaround. Regard this as permanent restriction.

No.9 Restriction on DBPC and DBPSW access during a break

[Description]

Write access to DBPC and DBPSW cannot be performed during a break.
(Read access is possible.)

[Workaround]

There is no workaround. Regard this as permanent restriction.

No.10 Restriction on DBTRAP instructions

[Description]

If a break occurs in the interrupt servicing of a DBTRAP instruction that is executed while a user program is running, the DBPC and DBPSW will be read incorrectly by subsequent

RUN instructions.

[Workaround]

There is no workaround. Regard this as permanent restriction.

No.11 Restriction on access data traced by DMA

[Description]

When data in internal RAM is read by DMA, the read data value is not traced correctly.

* Read address, write data, and write address are traced correctly.

[Workaround]

There is no workaround. Regard this as permanent restriction.

No.12 Restriction on peripheral I/O register read access during break

[Description]

The peripheral I/O register bits that are normally cleared by a read access (e.g. the TC bit of the DCHC register) are also cleared when displayed using peripheral I/O register display in the debugger during a break. (They are cleared when displayed by the debugger even though they are not read by the program.)

[Workaround]

The bit is not reset if the relevant peripheral I/O register is not displayed by debugger.

This restriction can be avoided by using the following debuggers.

Debugger by NEC:	ID850 E2.20f or later
Debugger by GHS (Multi):	SV-V850-WIN32 Rel.4.0.5 or later
(Windows version)	Specify the -X2 option on starting
Debugger by GHS (Multi):	SV-V850-Solaris Rel.4.0.5 or later
(Solaris version)	Specify the -X2 option on starting

* Use the latest debugger.

No.13 Restriction on target operating voltage

[Description]

Emulation cannot be performed when the target operating voltage is 3.5 V or higher.

[Workaround]

There is no workaround. This bug has been corrected in control code B and later products.

No.14 Improvement of power supply block safety

[Description]

The safety of the power supply block has been improved.

(There is no safety problem even if this improvement has not been performed.)

This improvement has been implemented in control code C and later products.

4. Other Cautions

1) Notes on using emulation memory

- The number of waits to be inserted varies depending on the operating frequency of the emulator.
  4 MHz ≤ Operating frequency < 25 MHz: 0 waits
  25 MHz ≤ Operating frequency ≤ 40 MHz: 1 wait
  40 MHz < Operating frequency: 2 waits
  (* The IE-V850ES-G1 does not support an operating frequency of 20 MHz or higher.)
- Set the bus width to 16 or 8 bits. The 32-bit bus cannot be used.
- The emulation memory cannot be mapped to addresses higher than 0x4000000.
- The number of wait cycles for the emulation memory is not affected by the _WAIT signal but is determined by setting of the debugger or setting of the wait control register.
  When setting the number of waits for the emulation memory to the same number as for the actual external memory for the purpose of performance testing, select “Target Wait” using the debugger.
  At this time, if address waits are inserted in the targeted external memory, also add the number of waits inserted in the external memory to the emulation memory.

With ID850

The following three alternatives are available on the configuration screen.

		Emulation Memory Access	External Memory Access
WAIT MASK	Data wait	Fixed to 0 wait	Depends on the DWC register settings. WAIT signal is masked.
	Address wait	Fixed to 0 wait	Depends on the ASC register settings.
	Idle state	Fixed to 0 cycle	Depends on the BCC register settings.
1 WAIT ACCESS	Data wait	Fixed to 1 wait	Depends on the DWC register settings and WAIT signal status.
	Address wait	Fixed to 0 wait	Depends on the ASC register settings.
	Idle state	Fixed to 0 cycle	Depends on the BCC register settings.
TARGET WAIT	Data wait	Depends on the DWC register setting. 1 wait when set to 0 wait.	Depends on the DWC register settings and WAIT signal status.
	Address wait	Fixed to 0 wait	Depends on the ASC register settings.
	Idle state	Depends on the BCC register setting	Depends on the BCC register settings.

With Multi

The wait pin can be masked or unmasked by the “pinmask” command.

		Emulation Memory Access	External Memory Access
WAIT: Mask EMWAIT: Mask	Data wait	Fixed to 0 wait	Depends on the DWC register settings. WAIT signal is masked.
	Address wait	Fixed to 0 wait	Depends on the ASC register settings.
	Idle state	Fixed to 0 cycle	Depends on the BCC register settings.
WAIT: Unmask EMWAIT: Mask	Data wait	Fixed to 1 wait	Depends on the DWC register settings and WAIT signal status.
	Address wait	Fixed to 0 wait	Depends on the ASC register settings.
	Idle state	Fixed to 0 cycle	Depends on the BCC register settings.
WAIT: Unmask EMWAIT: Unmask	Data wait	Depends on the DWC register setting. 1 wait when set to 0 wait.	Depends on the DWC register settings and WAIT signal status.
	Address wait	Fixed to 0 wait	Depends on the ASC register settings.
	Idle state	Depends on the BCC register setting	Depends on the BCC register settings.

2) Pin processing

- This emulator uses the minimum pin processing, giving priority to compatibility with the device. Take adequate workarounds for static electricity if the emulator is used without the target connected.
- Inside the emulator, pin processing is performed by the emulation board. For details, refer to the user’s manual of the emulation board.

3) Power saving

To save power, be sure to insert five NOP instructions after executing the HALT instruction and an instruction that sets the STP bit (PSC register).

a) STP bit (PSC register) setting instruction

```
mov 0x2,r2
movea base_address,r0,r20     ; base_address = FFFF0000H
st.h r11,PRCMD[R20]           ; PRCMD = 01FCH
st.h r11,PSC[R20]             ; PSC = 01FEH
nop  ┐
nop  │
nop  } Insert five NOPs.
nop  │
nop  ┘
```

b) HALT instruction

```
halt
nop  }
nop  }
nop  } Insert five NOPs.
nop  }
nop  }
```

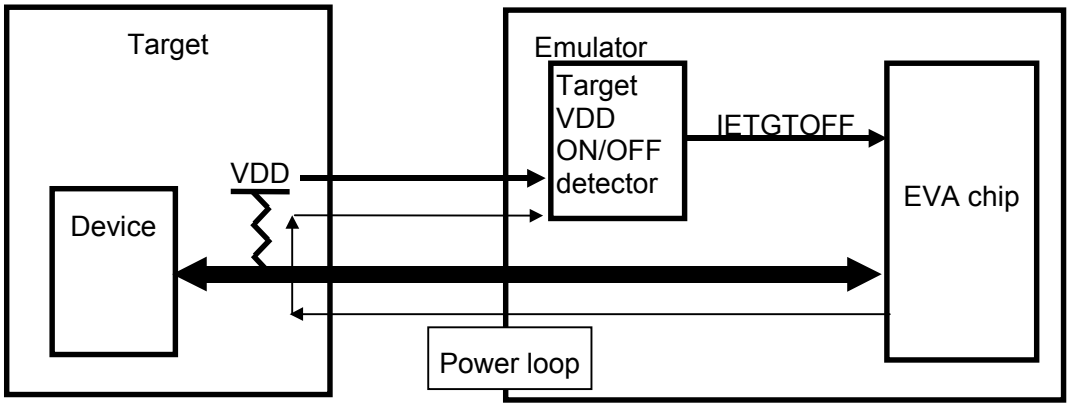
4) Pin control with target power off

When power to the emulator is on and that to the target is off, leakage current may flow from the emulator to the target.

When the target is connected, the emulator always senses the target supply voltage by using a target supply voltage detector, and the emulator is automatically reset when target power is turned off. In this reset status, the external bus signals go into a high-impedance state.

Some external bus signals, however, drive a high level, and a current may leak into VDD of the target via the pull-up resistor of the target.

The target supply voltage detector of the emulator detects this VDD, and the emulator assumes that power is applied to the target. Consequently, reset is cleared and the external bus signals are driven. As a result, a leakage current flows.



5) Notes on trace data

5-1) Trace sequence of access data of LD and ST instructions

If the LD and ST instructions are executed in that order, access of the ST instruction and access of the LD instruction are traced in this order for trace data.

If the LD instruction is the shortest (IRAM access), the read data is written to the same frame as the LD instruction. If the bus cycle is extended because of external memory access, the read data is written to trace after the ST instruction.

For instruction execution (fetch), the instruction that comes first is traced, and relation can be established based on the information on the access validity flag and direction of read/write.

- Details -

Basically, because the data is known in the write cycle, preparation for writing the data to the tracer is made when the ST instruction is executed. In the read cycle, the data is written to the tracer when the read cycle is completed and the data is loaded. If the LD and ST instructions are arranged, therefore, the sequence of only the access data of the trace data may be reversed, like the data of the ST instruction → data of the LD instruction.

Here is a sample program and its trace data:

[Sample program]

```
*    00000700 init    0000              nop
*    00000702 bpc     4056ff03          movhi 0x3ff, r0, r10
*    00000706         8a5e64f0          ori 0xf064, r10, r11
*    0000070A         2d06bb8f0000      mov 0x8fbb, r13
*    00000710         6b6f0000          st.h r13, 0x0[r11]        ; Writes 0x8FBB to address 0x3FFF064
*    00000714 bsc     8a5e66f0          ori 0xf066, r10, r11
*    00000718         206eaa6a          movea 0x6aaa, r0, r13
*    0000071C         6b6f0000          st.h r13, 0x0[r11]        ; Writes 0x6AAA to address 0x3FFF066
*    00000720         207eff00          movea 0xff, r0, r15
*    00000724         2086f00f          movea 0xff0, r0, r16
*    00000728 start   4056ee03          movhi 0x3ee, r0, r10
*    0000072C npb     8a5e40c0          ori 0xc040, r10, r11
*    00000730         2b8f0000          ld.h 0x0[r11], r17        ; Reads 0x0000 from address 0x3EEC040
*    00000734         6b7f0000          st.h r15, 0x0[r11]        ; Writes 0x00FF to address 0x3EEC040
*    00000738         2b970000          ld.h 0x0[r11], r18        ; Reads 0x00FF from address 0x3EEC040
*    0000073C         6b870000          st.h r16, 0x0[r11]        ; Writes 0x0FF0 to address 0x3EEC040
*    00000740         2b9f0000          ld.h 0x0[r11], r19        ; Reads 0x0ff0 from address 0x3EEC040
*    00000744         cb0f0000          set1 0x1, 0x0[r11]        ; SET instruction (RMW) to address 0x3EEC040
*    00000748         2ba70000          ld.h 0x0[r11], r20        ; Reads 0x0FF2 from address 0x3EEC040
*    0000074C         0000              nop
```

[Trace display example of sample program]

```
_    00000    1   00000000  80070007  BRM1                                  00    jr init
_    00001    7   00000700  00004056  BRM1                                  00    nop
_    00002    1   00000702  4056FF03  M1                                    00    movhi 0x3ff, r0, r10
_    00003    1   00000706  8A5E64F0  M1                                    00    ori 0xf064, r10, r11
_    00004    1   0000070A  2D06BB8F  M1                                    00    mov 0x8fbb, r13
_    00005   18   0000070A  2D060000                                        00
_    00006    1   00000710  6B6F0000  M1    03FFF064 8FBB    W  <--------  00    st.h r13, 0x0[r11]
_    00007    1   00000714  8A5E66F0  M1                                    00    ori 0xf066, r10, r11
_    00008   18   00000718  206EAA6A  M1                                    00    movea 0x6aaa, r0, r13
_    00009    1   0000071C  6B6F0000  M1    03FFF066 6AAA    W  <--------  00    st.h r13, 0x0[r11]
_    00010    1   00000720  207EFF00  M1                                    00    movea 0xff, r0, r15
_    00011    1   00000724  2086F00F  M1                                    00    movea 0xff0, r0, r16
_    00012    1   00000728  4056EE03  M1                                    00    movhi 0x3ee, r0, r10
_    00013    1   0000072C  8A5E40C0  M1                                    00    ori 0xc040, r10, r11
_    00014    1   00000730  2B8F0000  M1                                    00    ld.h 0x0[r11], r17
_    00015    1   00000734  6B7F0000  M1    03EEC040 00FF    W  <--------  00    st.h r15, 0x0[r11]
_    00016    1   00000738  2B970000  M1                                    00    ld.h 0x0[r11], r18
_    00017    1   0000073C  6B870000  M1                                    00    st.h r16, 0x0[r11]
_    00018   14                       03EEC040 0000    R                  00
_    00019    2                       03EEC040 0FF0    W                  00
_    00020    1   00000740  2B9F0000  M1                                    00    ld.h 0x0[r11], r19
_    00021   17   00000744  CB0F0000  M1                                    00    set1 0x1, 0x0[r11]
_    00022   14              03EEC040 00FF             R                  00
_    00023   50              03EEC040 0FF0             R                  00
_    00024    1              03EEC040 F0               R                  00
_    00025   36              03EEC040 F2               W                  00
_    00026    2   00000748 2BA70000  M1    03EEC040 0FF2    R  <--------  00    ld.h 0x0[r11], r20
```

5-2) Trace timing of external logic data

It takes the external logic data **the number of clocks required for fetching 8 x 1 times** to be output to the tracer.

- Details -

The external logic data is sampled in synchronization with instruction execution and differs depending on whether a program is stored in internal ROM or external memory, and on the number of wait cycles.

When a program is placed in the external memory, it also differs depending on whether a read/write cycle is inserted in the external memory.

The shortest time is 8 clocks when a program is placed in internal ROM.

The point to be noted is that the external logic data is not sampled every clock.

Because it is not sampled unless instruction execution is performed, a signal that changes after execution of one instruction and before execution of another cannot be traced.

If there is a possibility that an instruction is executed every clock as is the case with internal ROM, therefore, the chance of missing the external logic data decreases. Conversely, if many wait cycles are inserted in the external memory, the chance of missing the external logic data increases.