V850ES/Sx1, V850ES/Sx2, V850ES/Sx3 Microcontrollers Usage Restrictions	Document No.	ZBG-BF-09-0015	1/4
	Date issued	November 9, 2009	
	Issued by	Car Audio Solution Development Group Automotive Systems Division Microcomputer Operations Unit NEC Electronics Corporation	
Related documents: V850ES/SG1 Hardware User's Manual: U17329EJ2V0UD00 (2nd edition) V850ES/SG2, V850ES/SG2-H Hardware User's Manual: U16541EJ5V1UD00 (5th edition) V850ES/SJ2, V850ES/SJ2-H Hardware User's Manual: U16603EJ5V1UD00 (5th edition) V850ES/SG3 Hardware User's Manual: U17728EJ3V1UD00 (3rd edition) V850ES/SJ3 Hardware User's Manual: U17790EJ3V1UD00 (3rd edition) V850ES Architecture User's Manual: U15943EJ3V0UM00 (3rd edition)	Notification classification	√	Usage restriction
			Upgrade
			Document modification
			Other notification

1. Affected products

All V850ES/Sx1, V850ES/Sx2, and V850ES/Sx3 microcontrollers

For the names of the specific products in each series, see attachment 1.

2. Notification

The following restriction for the V850ES/Sx1, V850ES/Sx2, and V850ES/Sx3 microcontrollers has been found.

No. 3 Restriction on executing a `mul` or `mulu` instruction

Description:

The following occur if a specific instruction sequence (sequence 1 or 2 below) is executed.

- The result of executing a multiplication instruction is not stored in the relevant general-purpose register.
- As a result of executing an `ld` instruction for a mis-aligned address, the data at an incorrect address is read and stored in the relevant general-purpose register.

Sequence 1:

In the following instruction sequence, the RAM is read by one of the instructions in (2) at the same time as the RAM is accessed by a DMA transfer:

(1) `ld` or `sld`: A load instruction for the internal ROM

(2) `ld` or `sld`: A load instruction for the internal RAM

(3) `mul` or `mulu`: An instruction that multiplies word data and whose result is truncated to 32 bits[Note 1]

... **Note 2**

(4) `ld` or `sld`: A load instruction for a mis-aligned address in the internal ROM or RAM

Notes 1. For a `mul` or `mulu` instruction, the operation described in this restriction occurs if `r0` is specified for the third operand (`reg3`), or the same register is specified for the second operand (`reg2`) and third operand (`reg3`), as shown below:

`mul reg1, reg2, reg3` (`reg3` = `r0` or `reg2` = `reg3`)

`mul imm9, reg2, reg3` (`reg3` = `r0` or `reg2` = `reg3`)

`mulu reg1, reg2, reg3` (`reg3` = `r0` or `reg2` = `reg3`)

`mulu imm9, reg2, reg3` (`reg3` = `r0` or `reg2` = `reg3`)

For a program written in C, the CA850 does not generate the `mul` and `mulu` instructions if the same register is specified for the `reg2` and `reg3` operands.

2. This restriction applies if it takes 2 clock cycles or less between the instructions in (3) and (4).

This restriction does not apply in any of the following cases:

- DMA is not used to transfer data to or from the internal RAM.
- The data read by the load instruction in (1) is used to specify the target address of the load instruction in (2).
- The data read by the load instruction in (2) is referenced by the multiplication instruction in (3).
- The data obtained by the multiplication instruction in (3) is used to specify the target address of the load instruction in (4).
- The data obtained by the multiplication instruction in (3) is referenced by an instruction executed between the instructions in (3) and (4).
- At least one of the following instructions is executed between the instructions in (3) and (4):
  - A multiplication instruction (`mul`, `mulh`, `mulhi`, `mulu`)
  - A bit manipulation instruction (`clr1`, `not1`, `set1`, `tst1`)
  - A special instruction (`callt`, `dispose`, `switch`)
- The instruction in (4) is a load instruction that accesses the memory in bytes (`ld.b`, `ld.bu`, `sld.b`, or `sld.bu`).
- The instructions in (1) to (4) are located in an external memory or the internal RAM.

Sequence 2:

In the following instruction sequence, access by the instruction in (1) ends at the same time as the instruction in (2) accesses the internal RAM:

(1) `ld` or `sld`: A load instruction for an external memory or a CAN peripheral I/O register

... **Note 1**

(2) `ld` or `sld`: A load instruction for the internal RAM

(3) `mul` or `mulu`: An instruction that multiplies word data and whose result is truncated to 32 bits[Note 2]

... **Note 3**

(4) `ld` or `sld`: A load instruction for a mis-aligned address in the internal ROM or RAM

Notes 1. This restriction applies if a different instruction is executed between the load instructions in (1) and (2), and the access by the instruction in (1) ends at the same time as the instruction in (2) accesses the internal RAM.

2. For a `mul` or `mulu` instruction, the operation described in this restriction occurs if `r0` is specified for the third operand (`reg3`), or the same register is specified for the second operand (`reg2`) and third operand (`reg3`), as shown below:

`mul reg1, reg2, reg3` (`reg3` = `r0` or `reg2` = `reg3`)

`mul imm9, reg2, reg3` (`reg3` = `r0` or `reg2` = `reg3`)

`mulu reg1, reg2, reg3` (`reg3` = `r0` or `reg2` = `reg3`)

`mulu imm9, reg2, reg3` (`reg3` = `r0` or `reg2` = `reg3`)

For a program written in C, the CA850 does not generate the `mul` and `mulu` instructions if the same register is specified for the `reg2` and `reg3` operands.

3. This restriction applies if it takes 2 clock cycles or less between the instructions in (3) and (4).

This restriction does not apply in any of the following cases:

- An external bus and CAN controller are not used.
- The data read by the load instruction in (1) is used to specify the target address of the load instruction in (2).
- The data read by the load instruction in (1) is referenced by an instruction executed between the instructions in (1) and (2).
- The data read by the load instruction in (2) is referenced by the multiplication instruction in (3).
- The data obtained by the multiplication instruction in (3) is used to specify the target address of the load instruction in (4).
- The data obtained by the multiplication instruction in (3) is referenced by an instruction executed between the instructions in (3) and (4).
- At least one of the following instructions is executed between the instructions in (3) and (4):
  - A multiplication instruction (`mul`, `mulh`, `mulhi`, `mulu`)
  - A bit manipulation instruction (`clr1`, `not1`, `set1`, `tst1`)
  - A special instruction (`callt`, `dispose`, `switch`)
- The instruction in (4) is a load instruction that accesses the memory in bytes (`ld.b`, `ld.bu`, `sld.b`, or `sld.bu`).
- The instructions in (1) to (4) are located in an external memory or the internal RAM.

Workaround:

Action for systems being developed and to be developed in future:

We regard this as a restriction related to the CPU's features and do not intend to correct the microcontroller. Instead, the CA850 compiler will be modified to automatically prevent instructions to which this restriction applies from being generated. This workaround does not work for instructions in assembly code, so the CA850 outputs a message for such instructions. How NEC Electronics will provide the update for the compiler is shown below.

If you are using another compiler, contact an NEC Electronics sales representative.

- If your compiler is the NEC Electronics compiler CA850:

  The CA850 will be upgraded to add the above workaround.

  The following table shows the version and release schedule for the C compiler package CA850, and the software package SP850 that includes the CA850.

Product Name	Current CA850 Version	Version of CA850 After Upgrade	Language	Release Schedule
SP850, CA850	V3.40	V3.42	Japanese	November 9, 2009
			English	November 9, 2009

Action for already-developed systems:

Use the interview sheet in attachment 3 to check whether the restriction applies.

Application of this restriction to embedded software products:

This restriction applies to NEC Electronics real-time OSs and middleware as follows:

- Real-time OSs:

  RX850: The restriction does not apply.

  RX850 Pro: The restriction does not apply.

  RX850V4: The restriction does not apply.

- Middleware:

  GOFAST: The restriction does not apply.

  JPEG: The restriction does not apply.

For products other than the above, contact an NEC Electronics sales representative. For third-party products, contact the vendor of the product.

Modification:

The device will not be corrected, so regard this restriction as a specification.

Use the latest compiler to avoid this restriction.

3. Development environment required to work around this restriction

Use NEC Electronics compiler CA850 V3.42 or later.

Note that the outputs from the above version might differ from the outputs from the older versions.

If you are using another compiler, contact an NEC Electronics sales representative.

4. Document revision history

Document Number	Issued on	Description
ZBG-BF-05-0004	July 8, 2005	1st edition. Restriction nos. 1 and 2
ZBG-BF-09-0015	November 9, 2009	Restriction no. 3

V850ES/Sx1, V850ES/Sx2, and V850ES/Sx3 Microcontrollers

▼ V850ES/SG1

μPD703249Y, μPD703252Y, μPD703253Y

▼ V850ES/SG2

μPD703260(Y), μPD703261(Y), μPD703262(Y), μPD703263(Y), μPD70F3261(Y), μPD70F3263(Y), μPD703270(Y), μPD703271(Y), μPD703272(Y), μPD703273(Y), μPD70F3271(Y), μPD70F3273(Y), μPD703280(Y), μPD703281(Y), μPD703282(Y), μPD703283(Y), μPD70F3281(Y), μPD70F3283(Y), μPD703262HY, μPD703263HY, μPD70F3263HY, μPD703272HY, μPD703273HY, μPD70F3273HY, μPD703282HY, μPD703283HY, μPD70F3283HY

▼ V850ES/SJ2

μPD703264(Y), μPD703265(Y), μPD703266(Y), μPD70F3264(Y), μPD70F3266(Y) , μPD703274(Y), μPD703275(Y), μPD703276(Y), μPD70F3274(Y), μPD70F3276(Y), μPD703284(Y), μPD703285(Y), μPD703286(Y), μPD703287(Y), μPD703288(Y), μPD70F3284(Y), μPD70F3286(Y), μPD70F3288(Y), μPD703265HY, μPD703266HY, μPD70F3266HY, μPD703275HY, μPD703276HY, μPD70F3276HY, μPD703285HY, μPD703286HY, μPD70F3286HY, μPD703287HY, μPD703288HY, μPD70F3288HY

▼ V850ES/SG3

μPD70F3333, μPD70F3334, μPD70F3335, μPD70F3336, μPD70F3340, μPD70F3341, μPD70F3342, μPD70F3343, μPD70F3350, μPD70F3351, μPD70F3352, μPD70F3353

▼ V850ES/SJ3

μPD70F3344, μPD70F3345, μPD70F3346, μPD70F3347, μPD70F3348, μPD70F3354, μPD70F3355, μPD70F3356, μPD70F3357, μPD70F3358, μPD70F3364, μPD70F3365, μPD70F3366, μPD70F3367, μPD70F3368

V850ES/Sx1, V850ES/Sx2, and V850ES/Sx3 Microcontroller Usage Restrictions

No.	Restriction	Nickname	Product	**Note**	Development Environment in Which the Restriction Can Be Worked Around
1	Restriction on continuous write access to WDTM2 register in watchdog timer 2	V850ES/SG1 V850ES/SG2 V850ES/SJ2 V850ES/SG3 V850ES/SJ3	All products	△	–
2	Restrictions on rewriting to A/D control register and external/timer trigger input during A/D conversion		All products	△	–
3	Restriction on executing a `mul` or `mulu` instruction		All products	△	Category: Compiler Product name: CA850 Version: V3.42 or later

Note Whether or not the restriction applies

Remark The meaning of each symbol for **Note** is as follows:

–: Restriction does not apply
○: Restriction is already corrected
×: Restriction applies (correction is planned)
△: Restriction applies (correction is not planned)

No. 1 Restriction on continuous write access to WDTM2 register in watchdog timer 2

Description:

When a write access is made to the WDTM2 register twice or more after a reset, the CPU judges that the program is inadvertently looping and forcibly generates an overflow signal.

However, an overflow signal cannot be generated during a continuous write to WDTM2.

The overflow signal is generated after the continuous write is suspended.

Workaround:

Do not write to the WDTM2 register three times or more immediately after a reset until the next reset is applied.

No. 2 Restrictions on rewriting to A/D control register and external/timer trigger input during A/D conversion

Description:

(1) After A/D conversion is enabled, if a reconversion trigger is input[Note 2] at the same time as the stabilization time[Note 1] for A/D converter setup ends, the stabilization time is inserted again for 64 clock cycles. Furthermore, if the reinserted stabilization time ends at the same time as another reconversion trigger is input[Note 2], the stabilization time will be inserted again. Although the stabilization time is inserted again, conversion ends normally.

(2) In normal conversion mode, if a reconversion trigger [Note 2] is input at the same time as conversion in one-shot select conversion mode or external/timer trigger select conversion mode ends (prior to the wait time)[Note 1], the conversion that should restart does not restart, and conversion ends while A/D conversion remains enabled (ADA0CE = 1) but is stopped (ADA0M0.ADA0EF = 0). (No A/D conversion end interrupt occurs, and the conversion result is not stored).

To continue A/D conversion in one-shot select conversion mode, A/D conversion must be stopped (ADA0CE = 1 → 0) and then enabled (ADA0CE = 0 → 1). If the selected trigger signal is input and the wait time and stabilization time have elapsed when the A/D converter is using an external or timer trigger, conversion starts.

(3) In normal conversion mode, if an A/D conversion ends (prior to the wait time)[Note 1] at the same time as a reconversion trigger[Note 2] is input, the conversion that should restart does not restart and conversion ends while A/D conversion remains enabled (ADA0CE = 1) and A/D conversion is running (ADA0M0.ADA0EF = 1). (No A/D conversion end interrupt occurs, and the conversion result is not stored).

If a reconversion trigger[Note 2] occurs in this state, conversion starts again.

(4) In high-speed conversion mode and software conversion start trigger mode, if one-shot select conversion or one-shot scan conversion ends[Note 1] when a reconversion trigger[Note 2] is input, the conversion that should restart does not restart and conversion ends while A/D conversion remains enabled (ADA0CE = 1) but is stopped (ADA0M0.ADA0EF = 0). (No A/D conversion end interrupt occurs, and the conversion result is not stored). In this state, if 1 is written to the ADA0CE bit, or if A/D conversion is stopped (ADA0CE = 1 → 0) and then enabled again (ADA0CE = 0 → 1), conversion starts again.

Correspondence Between A/D Converter Operation Mode and Restriction

Conversion Trigger	Conversion Mode	High-Speed Conversion Mode (Without Intermittent Operation)				Normal Conversion Mode (with Intermittent Operation)			
		(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
Software trigger	Continuous select	X	–	–	√	X	√	X	–
	Continuous scan	X	–	–	√	X	√	X	–
	One-shot select	X	–	–	X	X	X Note 4	X Note 4	–
	One-shot scan	X	–	–	X	X	√ Note 3	X	–
External/ timer trigger	Select	X	–	–	√	X	X Note 4	X Note 4	–
	Scan	X	–	–	√	X	√ Note 3	X	–

X: Restriction applies, √: Restriction does not apply, –: Not relevant

Notes 1. For the timing details, see the user's manual for each product.

2. There are two types of reconversion trigger.

(a) Write to an A/D conversion register (ADA0M0, ADA0M2, ADA0S, ADA0PFM, or ADA0PFT)

(b) External trigger (detection of the ADTRG pin's edge) or timer trigger (TMP2 compare match interrupt request signal)

* If a reconversion trigger occurs during A/D conversion, the conversion is suspended and then restarts.

3. This restriction applies if ADA0S = 00H in scan mode.

4. Either (2) or (3) occurs depending on the internal status when the end of conversion conflicts with the input of the reconversion trigger.

Workaround:

1. If the external trigger or timer trigger is selected, set high-speed conversion mode. Do not input a trigger during the stabilization time that is inserted only once after A/D conversion is enabled (ADA0CE bit = 0 → 1).
2. Before writing to an A/D control register[Note] in normal conversion mode, or during one-shot conversion in high-speed conversion mode, stop A/D conversion (ADA0M0.ADA0CE = 0), and then enable A/D conversion (ADA0M0.ADA0CE = 1).

Note A/D control register: ADA0M0, ADA0M2, ADA0S, ADA0PFT, and ADA0PFM registers

No. 3 Restriction on executing a `mul` or `mulu` instruction

Description:

The following occur if a specific instruction sequence (sequence 1 or 2 below) is executed.

- The result of executing a multiplication instruction is not stored in the relevant general-purpose register.
- As a result of executing an `ld` instruction for a mis-aligned address, the data at an incorrect address is read and stored in the relevant general-purpose register.

Sequence 1:

In the following instruction sequence, the RAM is read by one of the instructions in (2) at the same time as the RAM is accessed by a DMA transfer:

(1) `ld` or `sld`: A load instruction for the internal ROM
(2) `ld` or `sld`: A load instruction for the internal RAM
(3) `mul` or `mulu`: An instruction that multiplies word data and whose result is truncated to 32 bits[Note 1]
... **Note 2**
(4) `ld` or `sld`: A load instruction for a mis-aligned address in the internal ROM or RAM

Notes 1. For a `mul` or `mulu` instruction, the operation described in this restriction occurs if `r0` is specified for the third operand (`reg3`), or the same register is specified for the second operand (`reg2`) and third operand (`reg3`), as shown below:

`mul reg1, reg2, reg3` (`reg3` = `r0` or `reg2` = `reg3`)
`mul imm9, reg2, reg3` (`reg3` = `r0` or `reg2` = `reg3`)
`mulu reg1, reg2, reg3` (`reg3` = `r0` or `reg2` = `reg3`)
`mulu imm9, reg2, reg3` (`reg3` = `r0` or `reg2` = `reg3`)

For a program written in C, the CA850 does not generate the `mul` and `mulu` instructions if the same register is specified for the `reg2` and `reg3` operands.

2. This restriction applies if it takes 2 clock cycles or less between the instructions in (3) and (4).

This restriction does not apply in any of the following cases:

- DMA is not used to transfer data to or from the internal RAM.
- The data read by the load instruction in (1) is used to specify the target address of the load instruction in (2).
- The data read by the load instruction in (2) is referenced by the multiplication instruction in (3).
- The data obtained by the multiplication instruction in (3) is used to specify the target address of the load instruction in (4).
- The data obtained by the multiplication instruction in (3) is referenced by an instruction executed between the instructions in (3) and (4).
- At least one of the following instructions is executed between the instructions in (3) and (4):
  - A multiplication instruction (`mul`, `mulh`, `mulhi`, `mulu`)
  - A bit manipulation instruction (`clr1`, `not1`, `set1`, `tst1`)
  - A special instruction (`callt`, `dispose`, `switch`)

- The instruction in (4) is a load instruction that accesses the memory in bytes (`ld.b`, `ld.bu`, `sld.b`, or `sld.bu`).
- The instructions in (1) to (4) are located in an external memory or the internal RAM.

Sequence 2:

In the following instruction sequence, access by the instruction in (1) ends at the same time as the instruction in (2) accesses the internal RAM:

(1) `ld` or `sld`: A load instruction for an external memory or a CAN peripheral I/O register

... **Note 1**

(2) `ld` or `sld`: A load instruction for the internal RAM

(3) `mul` or `mulu`: An instruction that multiplies word data and whose result is truncated to 32 bits[Note 2]

... **Note 3**

(4) `ld` or `sld`: A load instruction for a mis-aligned address in the internal ROM or RAM

Notes 1. This restriction applies if a different instruction is executed between the load instructions in (1) and (2), and the access by the instruction in (1) ends at the same time as the instruction in (2) accesses the internal RAM.

2. For a `mul` or `mulu` instruction, the operation described in this restriction occurs if `r0` is specified for the third operand (`reg3`), or the same register is specified for the second operand (`reg2`) and third operand (`reg3`), as shown below:

`mul reg1, reg2, reg3` (`reg3` = `r0` or `reg2` = `reg3`)

`mul imm9, reg2, reg3` (`reg3` = `r0` or `reg2` = `reg3`)

`mulu reg1, reg2, reg3` (`reg3` = `r0` or `reg2` = `reg3`)

`mulu imm9, reg2, reg3` (`reg3` = `r0` or `reg2` = `reg3`)

For a program written in C, the CA850 does not generate the `mul` and `mulu` instructions if the same register is specified for the `reg2` and `reg3` operands.

3. This restriction applies if it takes 2 clock cycles or less between the instructions in (3) and (4).

This restriction does not apply in any of the following cases:

- An external bus and CAN controller are not used.
- The data read by the load instruction in (1) is used to specify the target address of the load instruction in (2).
- The data read by the load instruction in (1) is referenced by an instruction executed between the instructions in (1) and (2).
- The data read by the load instruction in (2) is referenced by the multiplication instruction in (3).
- The data obtained by the multiplication instruction in (3) is used to specify the target address of the load instruction in (4).
- The data obtained by the multiplication instruction in (3) is referenced by an instruction executed between the instructions in (3) and (4).

- At least one of the following instructions is executed between the instructions in (3) and (4):
  - A multiplication instruction (`mul`, `mulh`, `mulhi`, `mulu`)
  - A bit manipulation instruction (`clr1`, `not1`, `set1`, `tst1`)
  - A special instruction (`callt`, `dispose`, `switch`)
- The instruction in (4) is a load instruction that accesses the memory in bytes (`ld.b`, `ld.bu`, `sld.b`, or `sld.bu`).
- The instructions in (1) to (4) are located in an external memory or the internal RAM.

Workaround:

We regard this as a restriction related to the CPU's features and do not intend to correct the microcontroller. Instead, the CA850 compiler will be modified to automatically prevent instructions to which this restriction applies from being generated. This workaround does not work for instructions in assembly code, so the CA850 outputs a message for such instructions. How NEC Electronics will provide the update for the compiler is shown below.

If you are using another compiler, contact an NEC Electronics sales representative.

- If your compiler is the NEC Electronics compiler CA850:

  The CA850 will be upgraded to add the above workaround.

  The following table shows the version and release schedule for the C compiler package CA850, and the software package SP850 that includes the CA850.

Product Name	Current CA850 Version	Version of CA850 After Upgrade	Language	Release Schedule
SP850, CA850	V3.40	V3.42	Japanese	November 9, 2009
			English	November 9, 2009

No. 3 Restriction on executing a `mul` or `mulu` instruction

• First judgment: Judgment based on product usage conditions

Check the conditions under which you are using the product to see whether the restriction applies to the product. If the restriction might apply, perform a second judgment. If the restriction is judged to be not applicable, subsequent checking is not necessary.

■ Checking the usage conditions

Select **Yes** or **No** for whether each feature from (1) to (3) below is used. If the product does not incorporate a feature, select **Not relevant** for the feature. If there is no item for which **Yes** is selected, the restriction does not apply.

(1) Data is not transferred to or from the internal RAM using DMA.

☐ Yes ☐ No ☐ Not relevant

(2) An on-chip CAN controller is used.

☐ Yes ☐ No ☐ Not relevant

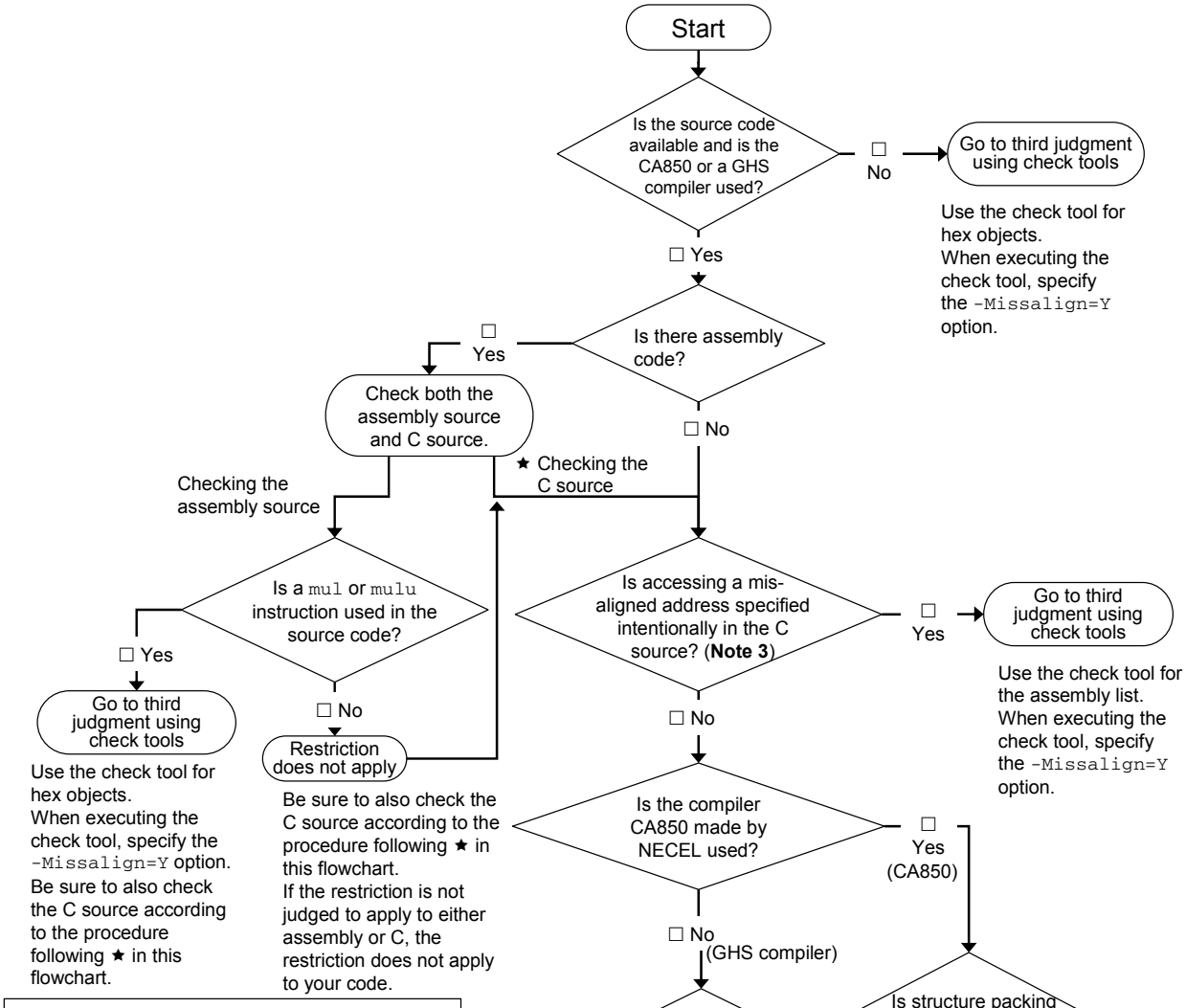
(3) An external bus interface is used.

☐ Yes ☐ No ☐ Not relevant

- **Second judgment: Judgment based on compiling conditions**

Interview Sheet (for second judgment)

Use the following flowchart to judge whether the restriction applies. If the restriction might apply, a third judgment using check tools is required.



Note 1: When using the CA850
The restriction might apply if either the following conditions is satisfied:

1. `#pragma pack(1)` or `#pragma pack(2)` is specified in the source code.

2. `-Xpack=1` or `-Xpack=2` is specified as a compiler option.

Note 2: When using a GHS compiler
The restriction might apply if either the following conditions is satisfied:

1. `#pragma pack(1)` or `#pragma pack(2)` is specified in the source code and `-misalign_pack` is specified as a compiler option.

2. `-pack=1` or `-pack=2` and `-misalign_pack` are specified as compiler options.

Note 3: A mis-aligned address might be accessed if a pointer to a `char` is cast to a pointer to an `int` to reference the `int`, as shown below.

```
func()
{
  char  *ptr_char="abcdef" ;
  int    data,  *ptr_int ;

    ptr_int= (int *)
(ptr_char+2);
   data = *ptr_int;
}
```