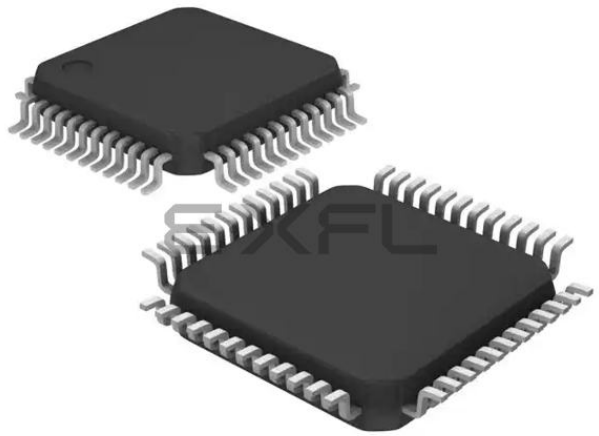


**Welcome to E-XFL.COM**

## What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

## Applications of "Embedded - Microcontrollers"

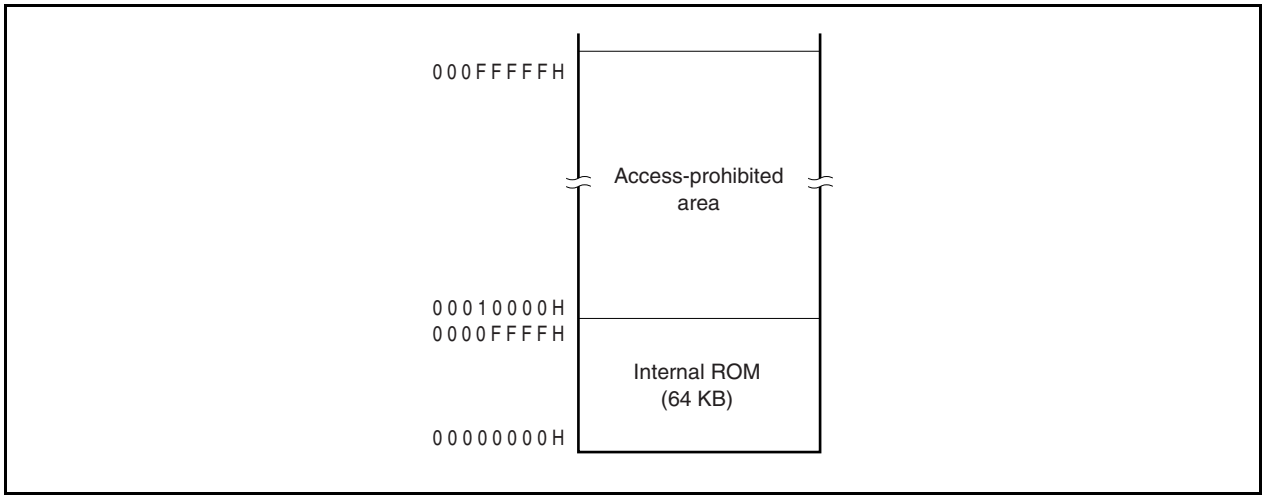
| Details | |
|---|---|
| Product Status | Active |
| Core Processor | V850ES |
| Core Size | 32-Bit Single-Core |
| Speed | 48MHz |
| Connectivity | CANbus, CSI, EBI/EMI, I²C, UART/USART, USB |
| Peripherals | DMA, LVD, PWM, WDT |
| Number of I/O | 32 |
| Program Memory Size | 256KB (256K x 8) |
| Program Memory Type | FLASH |
| EEPROM Size | - |
| RAM Size | 24K x 8 |
| Voltage - Supply (Vcc/Vdd) | 2.85V ~ 3.6V |
| Data Converters | A/D 6x10b; D/A 1x8b |
| Oscillator Type | Internal |
| Operating Temperature | -40°C ~ 85°C (TA) |
| Mounting Type | Surface Mount |
| Package / Case | 48-LQFP |
| Supplier Device Package | - |
| Purchase URL | https://www.e-xfl.com/product-detail/renesas-electronics-america/upd70f3818ga-gam-ax |

**(c) Internal ROM (64 KB)**

64 KB are allocated to addresses 00000000H to 0000FFFFH in the following products.
Accessing addresses 00010000H to 000FFFFFH is prohibited.

- μPD70F3811, 70F3816, 70F3822

**Figure 3-6. Internal ROM Area (64 KB)**

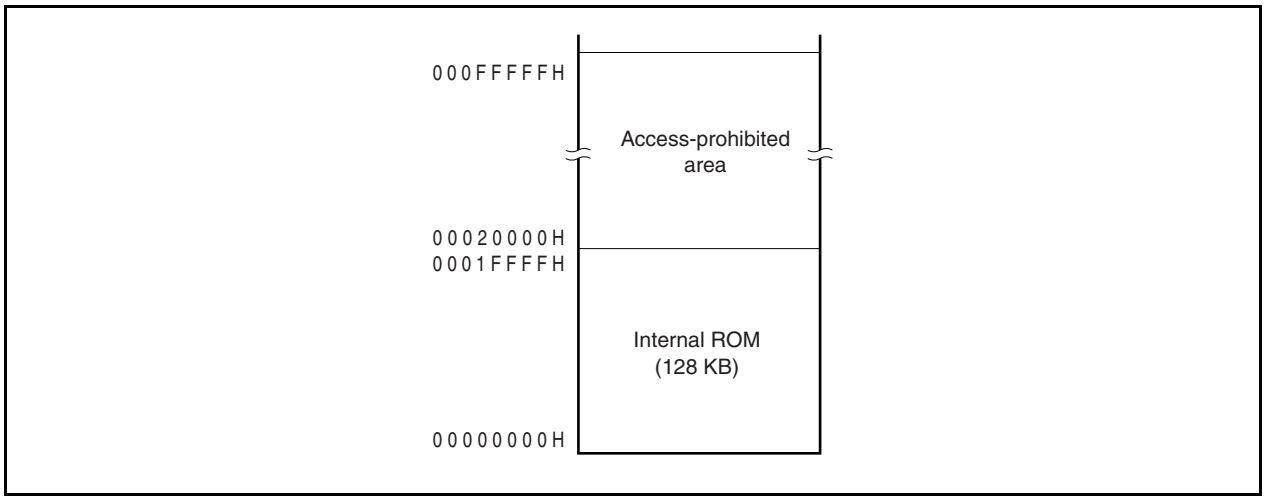



**(d) Internal ROM (128 KB)**

128 KB are allocated to addresses 00000000H to 0001FFFFH in the following products.
Accessing addresses 00020000H to 000FFFFFH is prohibited.

- μPD70F3812, 70F3817, 70F3823

**Figure 3-7. Internal ROM Area (128 KB)**



**(1) Setting data to special registers**

Set data to the special registers in the following sequence.

<1> Disable DMA operation.
<2> Prepare data to be set to the special register in a general-purpose register.
<3> Write the data prepared in <2> to the PRCMD register.
<4> Write the setting data to the special register (by using the following instructions).
- Store instruction (ST/SST instruction)
- Bit manipulation instruction (SET1/CLR1/NOT1 instruction)

(<5> to <9> Insert NOP instructions (5 instructions).)[Note]
<10> Enable DMA operation if necessary.

[Example] With PSC register (setting standby mode)

```
   ST.B r11, PSMR[r0]   ; Set PSMR register (setting IDLE1, IDLE2, and STOP modes).
<1>CLR1 0, DCHCn[r0]    ; Disable DMA operation.  n = 0 to 3
<2>MOV0x02, r10
<3>ST.B r10, PRCMD[r0]  ; Write PRCMD register.
<4>ST.B r10, PSC[r0]    ; Set PSC register.
<5>NOP[Note]            ; Dummy instruction
<6>NOP[Note]            ; Dummy instruction
<7>NOP[Note]            ; Dummy instruction
<8>NOP[Note]            ; Dummy instruction
<9>NOP[Note]            ; Dummy instruction
<10>SET1 0, DCHCn[r0]   ; Enable DMA operation.  n = 0 to 3
(next instruction)
```

There is no special sequence to read a special register.

**Note** Five NOP instructions or more must be inserted immediately after setting the IDLE1 mode, IDLE2 mode, or STOP mode (by setting the PSC.STP bit to 1).

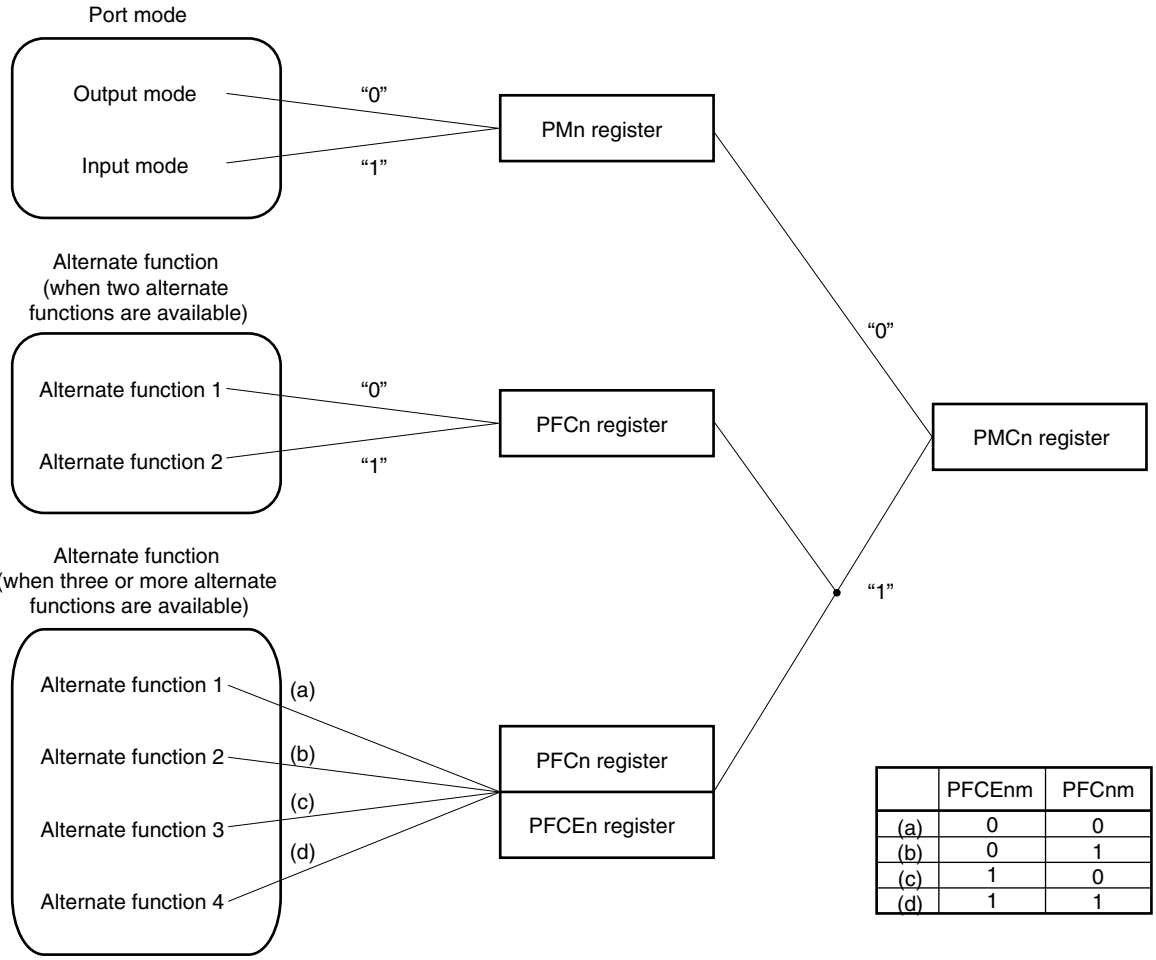
**Cautions 1. When a store instruction is executed to store data in the command register, interrupts are not acknowledged. This is because it is assumed that steps <3> and <4> above are performed by successive store instructions. If another instruction is placed between <3> and <4>, and if an interrupt is acknowledged by that instruction, the above sequence may not be established, causing malfunction.**

**2. Although dummy data is written to the PRCMD register, use the same general-purpose register used to set the special register (<4> in Example) to write data to the PRCMD register (<3> in Example). The same applies when a general-purpose register is used for addressing.**

**(7) Port setting**

Set a port as illustrated below.

**Figure 4-4. Setting of Each Register and Pin Function**




| | PFCEnm | PFCnm |
|---|---|---|
| (a) | 0 | 0 |
| (b) | 0 | 1 |
| (c) | 1 | 0 |
| (d) | 1 | 1 |

**Remark** Set the alternate functions in the following sequence.

<1> Set the PFCn and PFCEn registers.
<2> Set the PMCn register.
<3> Set the INTRn or INTFn register (to specify an external interrupt pin).

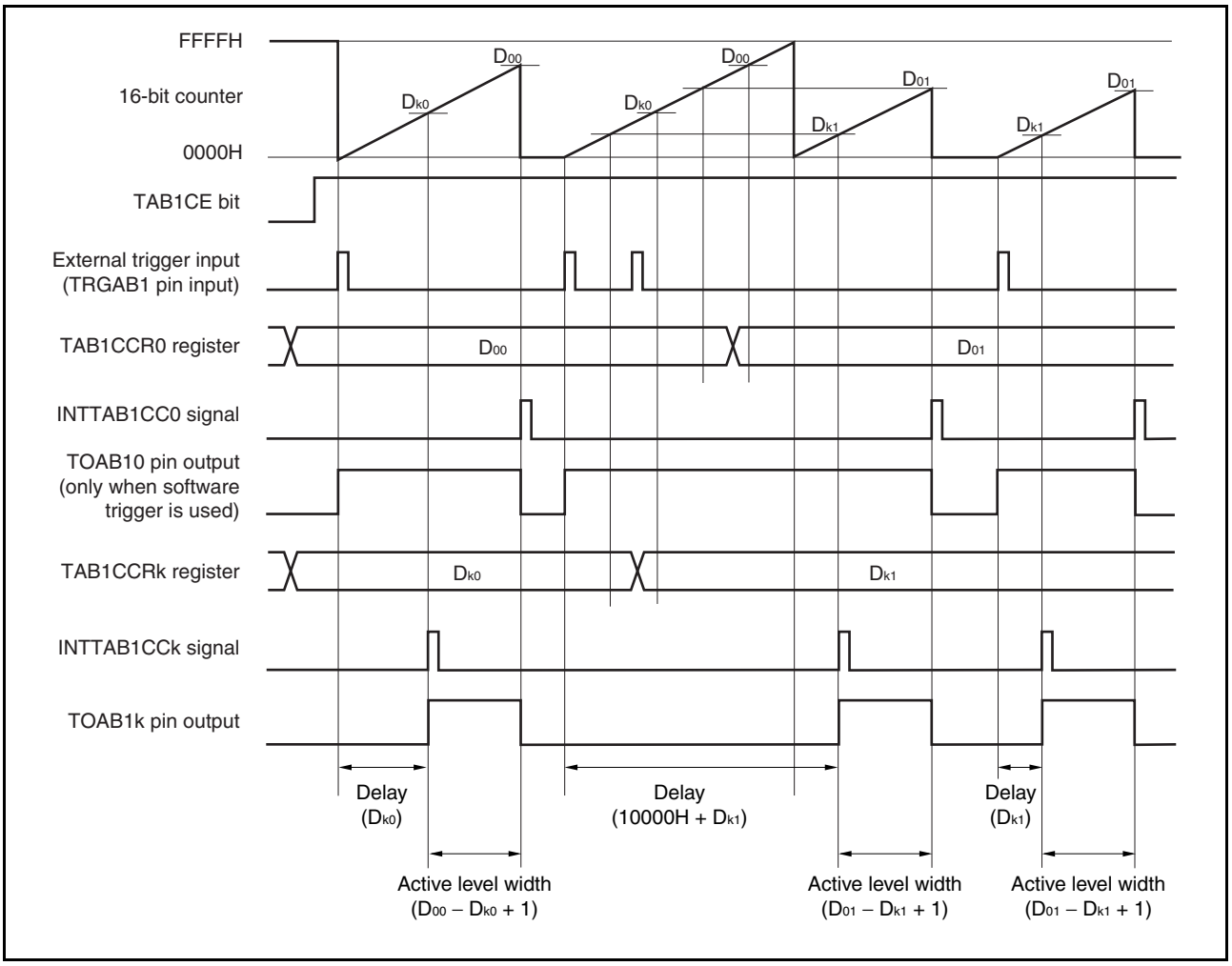
If the PMCn register is set first, an unintended function may be set while the PFCn and PFCEn registers are being set.

### (2) Operation timing in one-shot pulse output mode

#### (a) Notes on rewriting TAB1CCRm register

To change the set value of the TAB1CCRm register to a smaller value, stop counting once, and then change the set value.

If the value of the TAB1CCR0 register is rewritten to a smaller value during counting, the 16-bit counter may overflow.



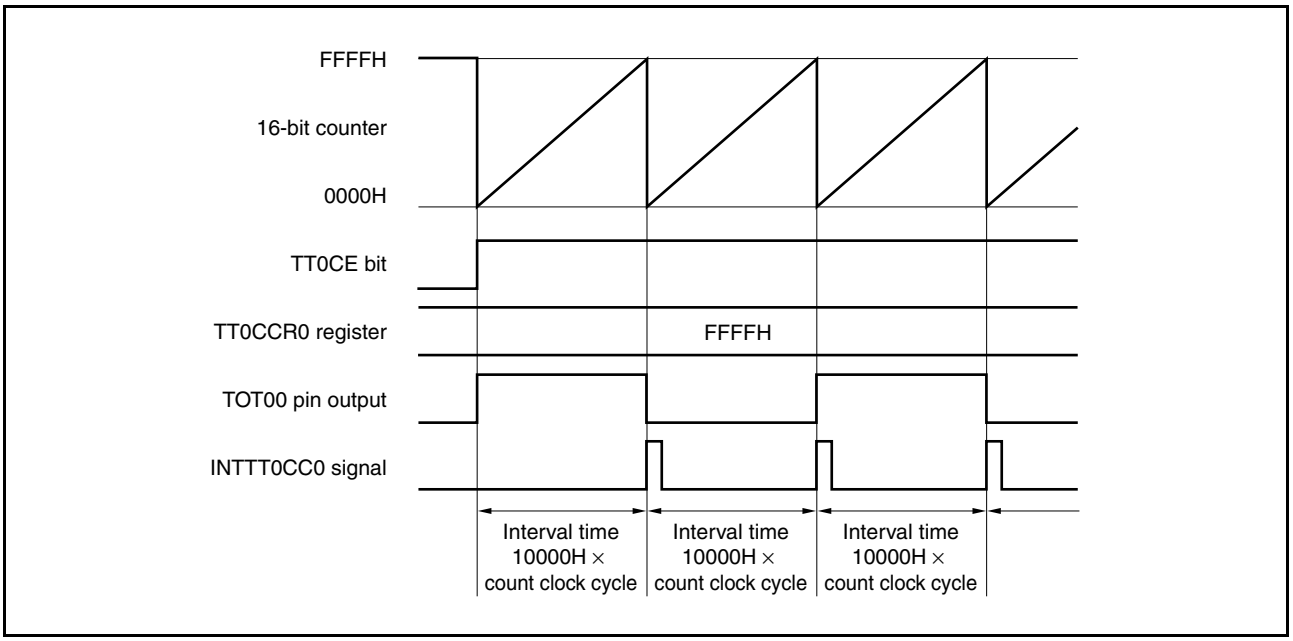

When the TAB1CCR0 register is rewritten from $D_{00}$ to $D_{01}$ and the TAB1CCRk register from $D_{k0}$ to $D_{k1}$ where $D_{00} > D_{01}$ and $D_{k0} > D_{k1}$, if the TAB1CCRk register is rewritten when the count value of the 16-bit counter is greater than $D_{k1}$ and less than $D_{k0}$ and if the TAB1CCR0 register is rewritten when the count value is greater than $D_{01}$ and less than $D_{00}$, each set value is reflected as soon as the register has been rewritten and compared with the count value. The counter counts up to FFFFH and then counts up again from 0000H. When the count value matches $D_{k1}$, the counter generates the INTTAB1CCk signal and asserts the TOAB1k pin. When the count value matches $D_{01}$, the counter generates the INTTAB1CC0 signal, deasserts the TOAB1k pin, and stops counting.

Therefore, the counter may output a pulse with a delay period or active period different from that of the one-shot pulse that is originally expected.

**Remark** k = 1 to 3

**(b) Operation if TT0CCR0 register is set to FFFFH**

If the TT0CCR0 register is set to FFFFH, the 16-bit counter counts up to FFFFH. The counter is cleared to 0000H in synchronization with the next count-up timing. The INTTT0CC0 signal is generated and the output of the TOT00 pin is inverted. At this time, an overflow interrupt request signal (INTTT0OV) is not generated, nor is the overflow flag (TT0OPT0.TT0OVF bit) set to 1.



**(a) Setting procedure**

**(i) Setting of high-impedance control operation**

<1> Set the HZA0DCMn, HZA0DCNn, and HZA0DCPn bits.

<2> Set the HZA0DCEn bit to 1 (enable high-impedance control).

**(ii) Changing setting after enabling high-impedance control operation**

<1> Clear the HZA0DCEn bit to 0 (to stop the high-impedance control operation).

<2> Change the setting of the HZA0DCMn, HZA0DCNn, and HZA0DCPn bits.

<3> Set the HZA0DCEn bit to 1 (to enable the high-impedance control operation again).

**(iii) Resuming output when pins are in high-impedance state**

If the HZA0DCMn bit is 1, set the HZA0DCCn bit to 1 to clear the high-impedance state after the valid edge of the external pin is detected. However, the high-impedance state cannot be cleared unless this bit is set while the input level of the external pin is inactive.

<1> Set the HZA0DCCn bit to 1 (command signal to clear the high-impedance state).

<2> Read the HZA0DCFn bit and check the flag status.

<3> Return to <1> if the HZA0DCFn bit is 1. The input level of the external pin must be checked.
The pin can function as an output pin if the HZA0DCFn bit is 0.

**(iv) Making pin go into high-impedance state by software**

The HZA0DCTn bit must be set to 1 by software to make the pin go into a high-impedance state while the input level of the external pin is inactive. The following procedure is an example in which the setting is not dependent upon the setting of the HZA0DCMn bit.

<1> Set the HZA0DCTn bit to 1 (high-impedance output command).

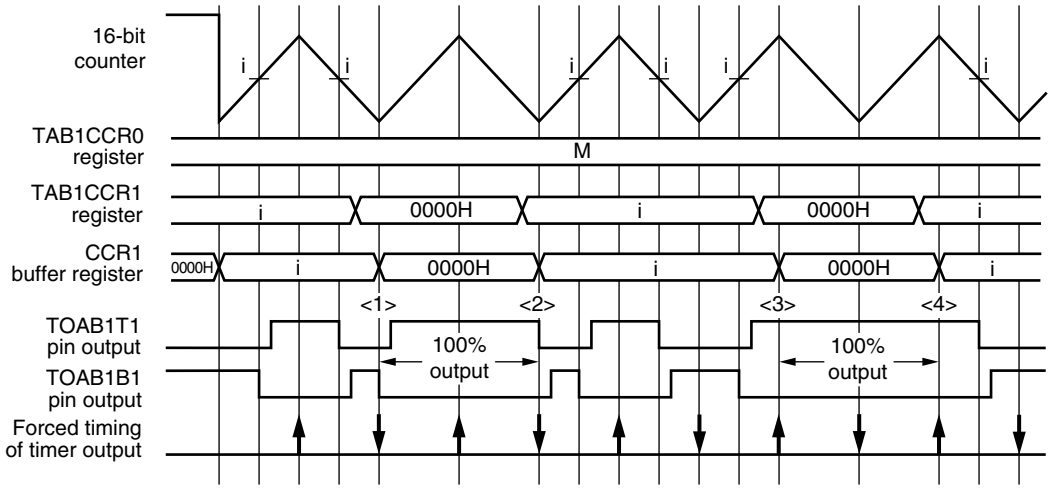
<2> Read the HZA0DCFn bit to check the flag status.

<3> Return to <1> if the HZA0DCFn bit is 0. The input level of the external pin must be checked.
The pin is in a high-impedance state if the HZA0DCFn bit is 1.

However, if the external pin is not used with the HZA0DCPn bit and HZA0DCNn bit cleared to 0, the pin goes into a high-impedance state when the HZA0DCTn bit is set to 1.

**Remark** n = 0, 1

**Figure 10-10. 100% PWM Output Waveform (With Dead Time)**




<1> 100% output is selected by the valley interrupt (with a match with the 16-bit counter).
The valley interrupt forcibly lowers the timer output, but raising the timer output takes precedence when the value of the TAB1CCRm register matches the value of the 16-bit counter. As a result, the 100% output is produced.

<2> 100% output is canceled by the valley interrupt (without a match with the 16-bit counter).
The valley interrupt forcibly lowers the timer output. This cancels the 100% output.

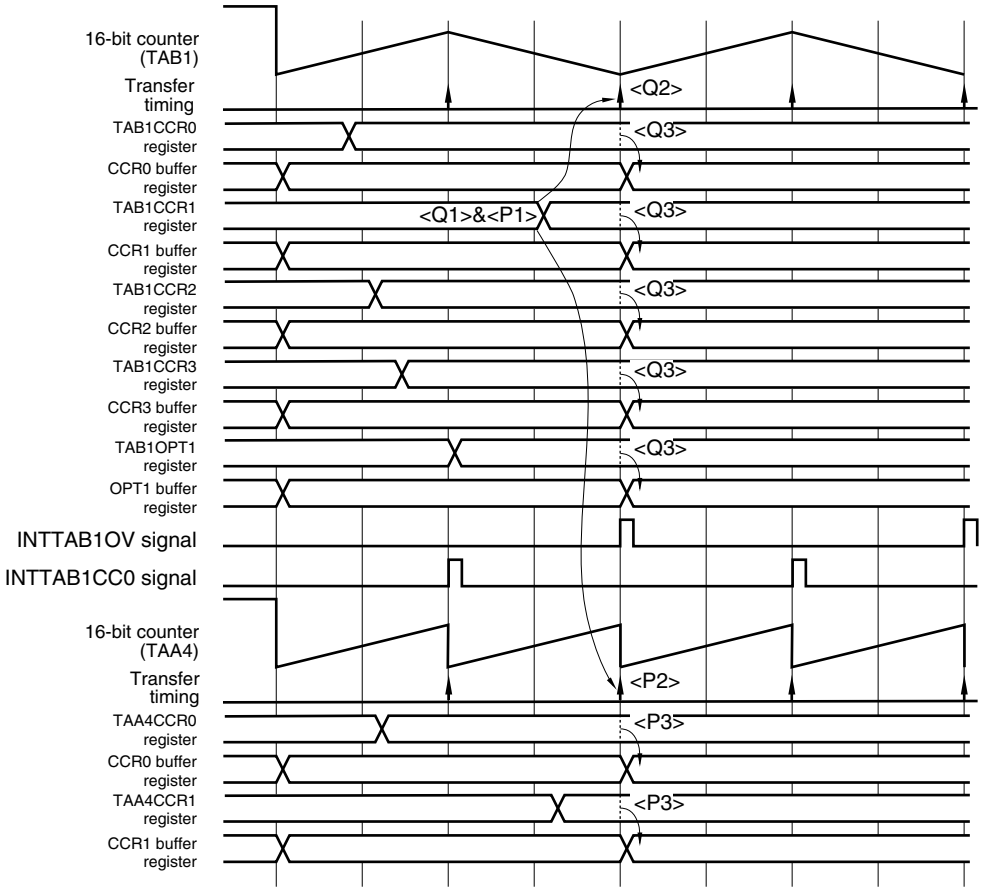
<3> 100% output is selected by the crest interrupt (without a match with the 16-bit counter).
The crest interrupt forcibly raises the timer output. This produces the 100% output.

<4> 100% output is canceled by the crest interrupt (without a match with the 16-bit counter).
The crest interrupt forcibly raises the timer output. This cancels the 100% output.

**Remarks 1.** ↑ means forcible raising and ↓ means forcible lowering.

**2.** m = 1 to 3

**Figure 10-26. Basic Operation in Batch Mode**




[Operation of TAB1]

<Q1> Write the TAB1CCR1 register

<Q2> The target timing is the first transfer timing after a write to the TAB1CCR1 register.

<Q3> The values are transferred all at once at the transfer timing.

[Operation of TAA4]

<P1> Write the TAB1CCR1 register

<P2> The target timing is the first transfer timing after a write to the TAB1CCR1 register.

<P3> The values are transferred all at once at the transfer timing.

**(b) Rewriting TAB1CCR0 register**

When rewriting the TAB1CCR0 register in the batch rewrite mode, the output waveform differs depending on whether transfer occurs at the crest (match between the 16-bit counter value and TAB1CCR0 register value) or at the valley (match between the 16-bit counter value and 0001H). Usually, it is recommended to rewrite the TAB1CCR0 register while the 16-bit counter is counting down, and transfer the register value at the transfer timing of the crest timing.

Figure 10-28 shows an example of rewriting the TAB1CCR0 register while the 16-bit counter is counting up (during period <1> in Figure 10-27). Figure 10-29 shows an example of rewriting the TAB1CCR0 register while the counter is counting down (during period <2> in Figure 10-27).

**Figure 10-27. Basic Operation of 16-Bit Counter**

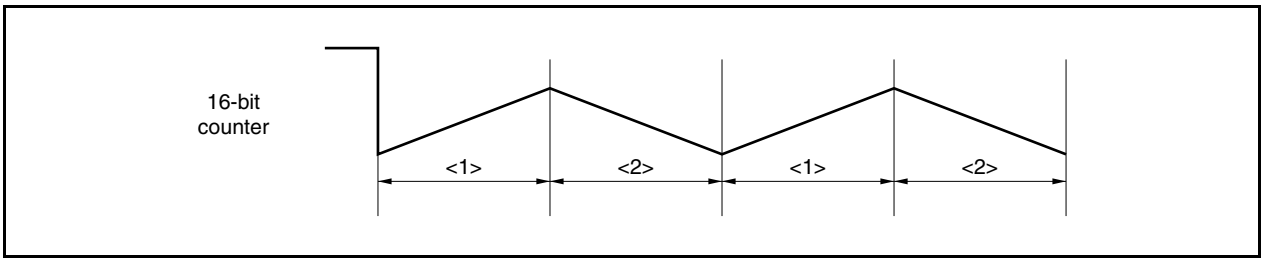
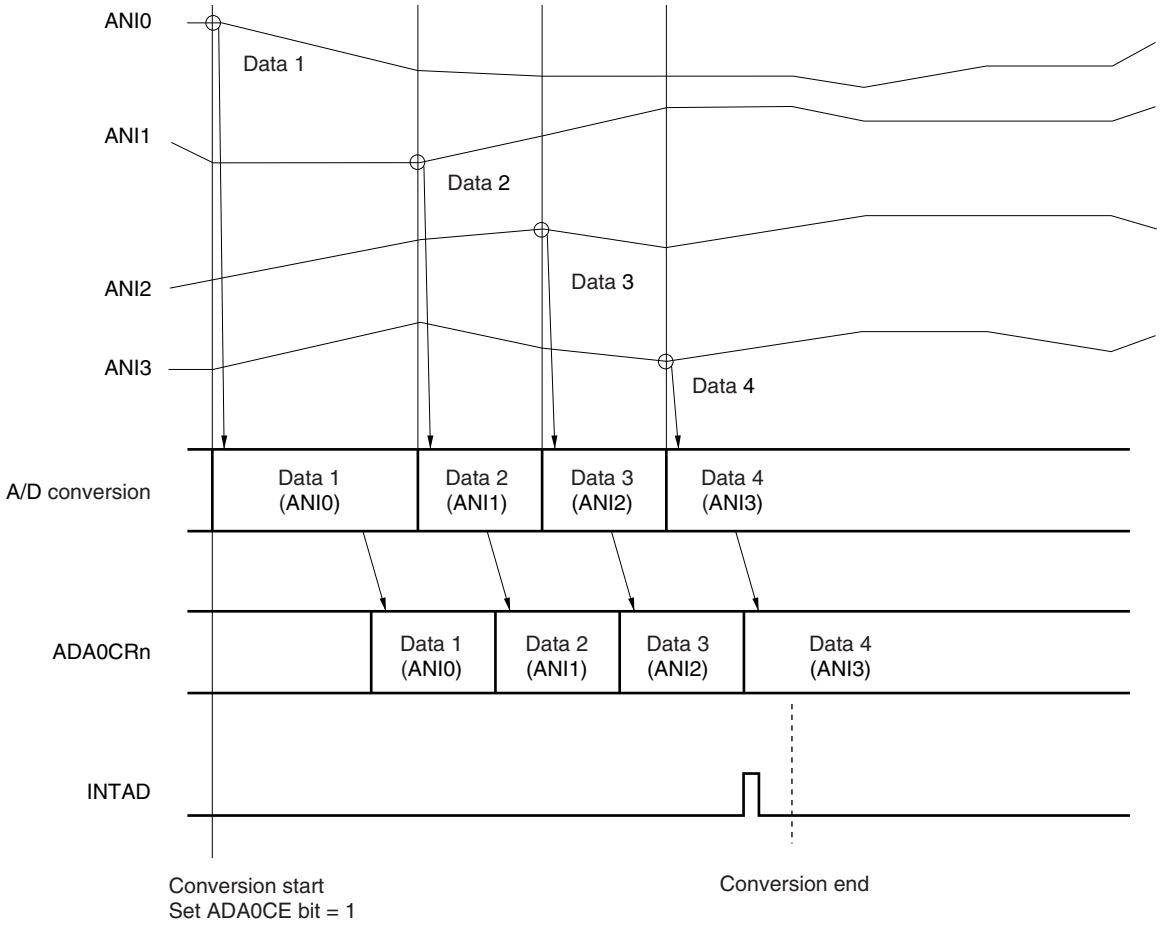
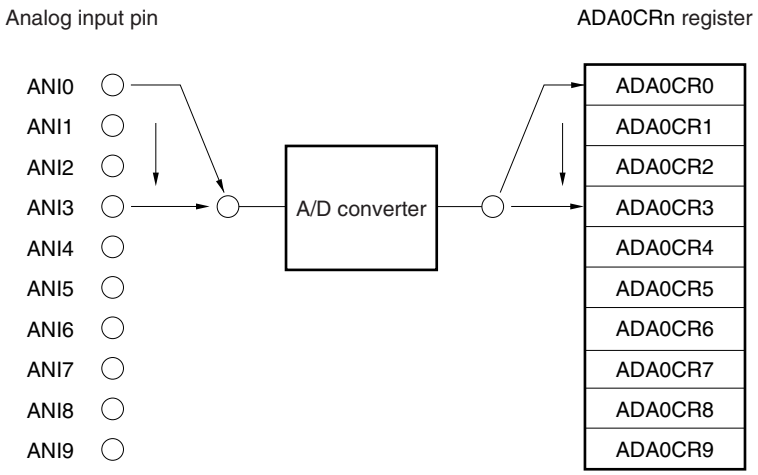


**Figure 14-7. Timing Example of One-Shot Scan Mode Operation (ADA0S Register = 03H)**

**(a) Timing example**



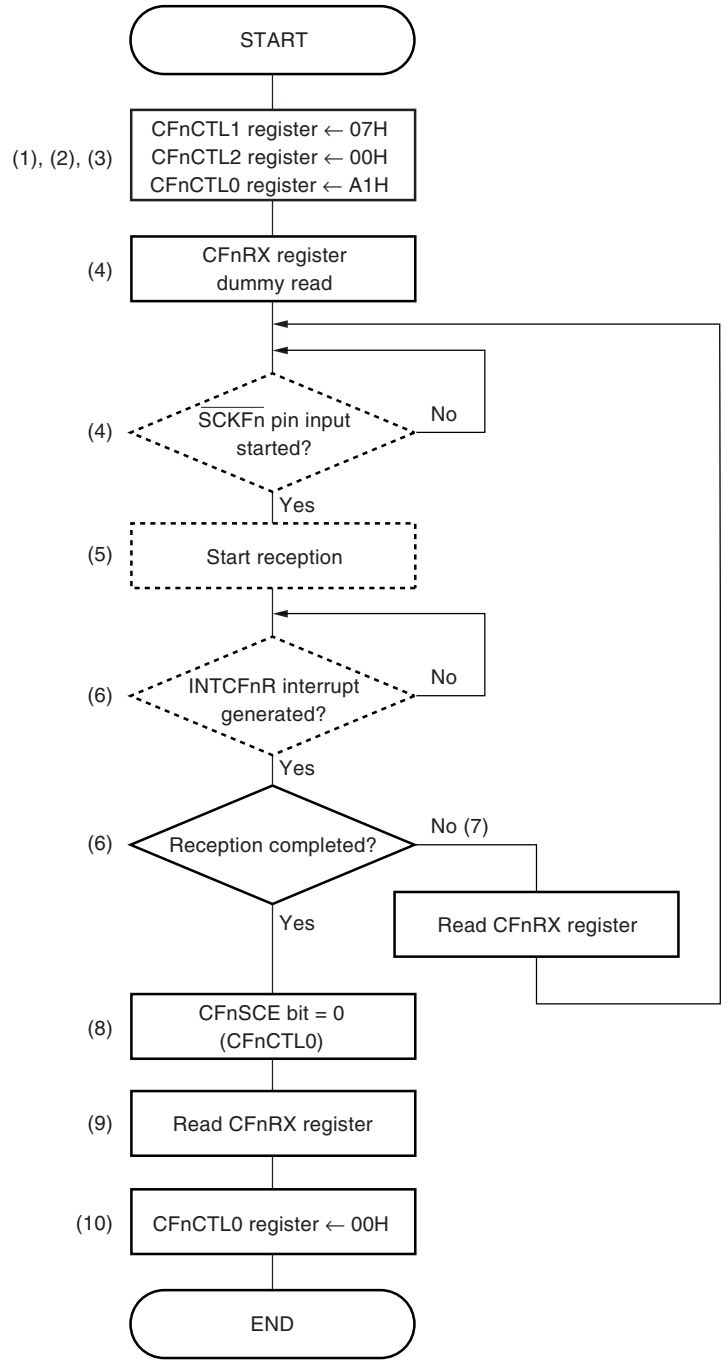

**(b) Block diagram**



### 17.6.5 Single transfer mode (slave mode, reception mode)

MSB first (CFnCTL0.CFnDIR bit = 0), communication type 1 (CFnCTL1.CFnCKP and CFnCTL1.CFnDAP bits = 00), communication clock ($f_{CCLK}$) = external clock ($\overline{SCKFn}$) (CFnCTL1.CFnCKS2 to CFnCTL1.CFnCKS0 bits = 111), transfer data length = 8 bits (CFnCTL2.CFnCL3 to CFnCTL2.CFnCL0 bits = 0000)

#### (1) Operation flow

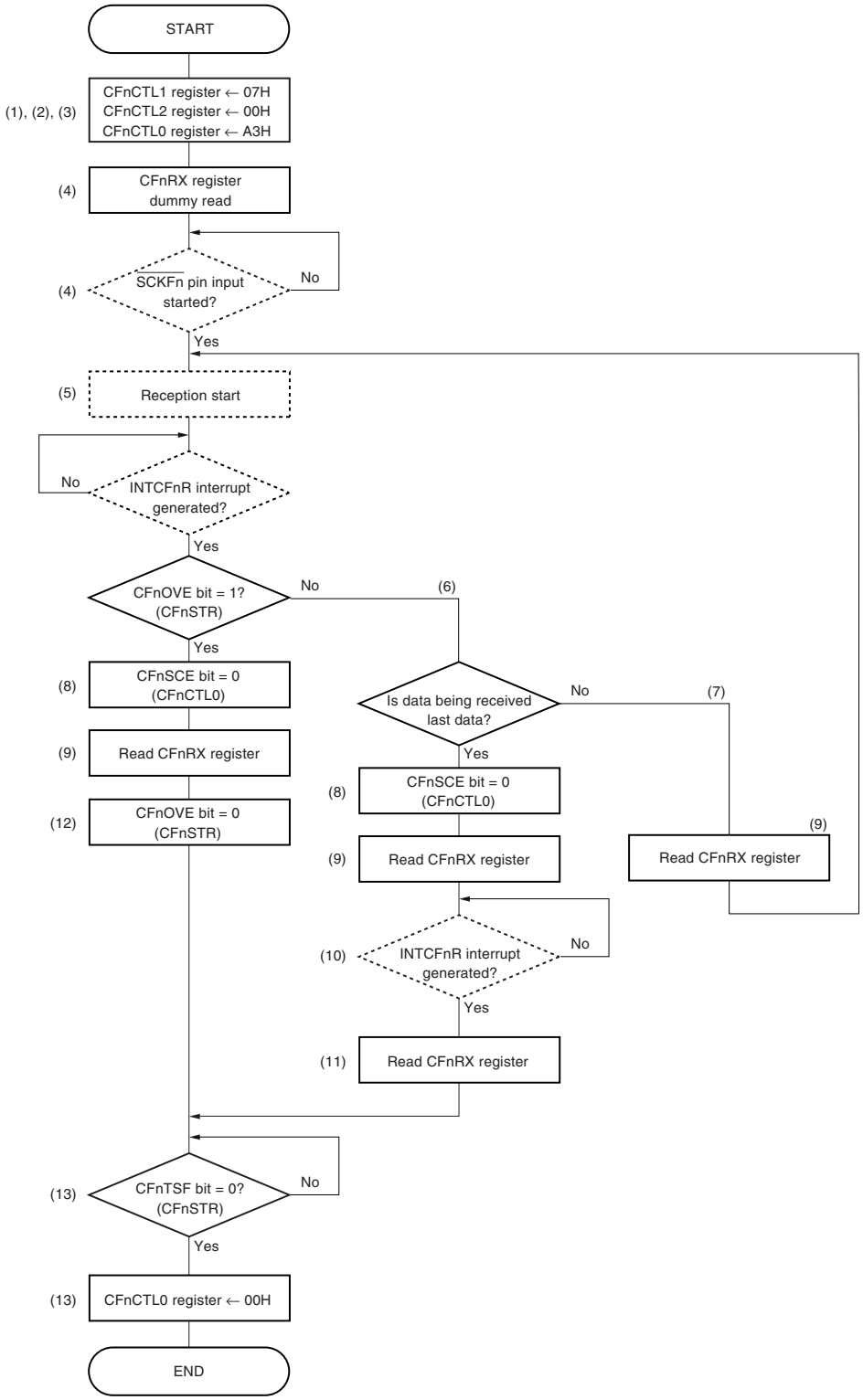



**Remarks 1.** The broken lines indicate the hardware processing.

**2.** The numbers in this figure correspond to the processing numbers in **(2) Operation timing**.

**3.** n = 0, 2 to 4

### (1) Operation flow




**Remarks 1.** The broken lines indicate the hardware processing.

**2.** The numbers in this figure correspond to the processing numbers in **(2) Operation timing**.

**3.** n = 0, 2 to 4

### (2) CAN0 global clock selection register (C0GMCS)

The C0GMCS register is used to select the CAN module system clock.

After reset: 0FH R/W Address: 03FEC002H

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| C0GMCS | 0 | 0 | 0 | 0 | CCP3 | CCP2 | CCP1 | CCP0 |

| CCP3 | CCP2 | CCP1 | CCP0 | CAN module system clock ($f_{CANMOD}$) |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | $f_{CAN}$/1 |
| 0 | 0 | 0 | 1 | $f_{CAN}$/2 |
| 0 | 0 | 1 | 0 | $f_{CAN}$/3 |
| 0 | 0 | 1 | 1 | $f_{CAN}$/4 |
| 0 | 1 | 0 | 0 | $f_{CAN}$/5 |
| 0 | 1 | 0 | 1 | $f_{CAN}$/6 |
| 0 | 1 | 1 | 0 | $f_{CAN}$/7 |
| 0 | 1 | 1 | 1 | $f_{CAN}$/8 |
| 1 | 0 | 0 | 0 | $f_{CAN}$/9 |
| 1 | 0 | 0 | 1 | $f_{CAN}$/10 |
| 1 | 0 | 1 | 0 | $f_{CAN}$/11 |
| 1 | 0 | 1 | 1 | $f_{CAN}$/12 |
| 1 | 1 | 0 | 0 | $f_{CAN}$/13 |
| 1 | 1 | 0 | 1 | $f_{CAN}$/14 |
| 1 | 1 | 1 | 0 | $f_{CAN}$/15 |
| 1 | 1 | 1 | 1 | $f_{CAN}$/16 (Default value) |

**Caution** Make sure that $f_{XX}$ = 32 to 48 MHz.

**Remark** $f_{CAN} = f_{XX}/2$

$f_{CAN}$: CAN clock frequency

$f_{XX}$: Main clock frequency

### (9) CAN0 module error counter register (C0ERC)

The C0ERC register indicates the count value of the transmission/reception error counter.

After reset: 0000H R Address: 03FEC054H

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| C0ERC | REPS | REC6 | REC5 | REC4 | REC3 | REC2 | REC1 | REC0 |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | TEC7 | TEC6 | TEC5 | TEC4 | TEC3 | TEC2 | TEC1 | TEC0 |

| REPS | Reception error passive status bit |
|---|---|
| 0 | The value of the reception error counter is not error passive (< 128) |
| 1 | The value of the reception error counter is in the error passive range (≥ 128) |

| REC6 to REC0 | Reception error counter bit |
|---|---|
| 0 to 127 | Number of reception errors. These bits reflect the status of the reception error counter. The number of errors is defined by the CAN protocol. |

**Remark** The REC6 to REC0 bits of the reception error counter are invalid in the reception error passive status (C0INFO.RECS1, C0INFO.RECS0 bit = 11B).

| TEC7 to TEC0 | Transmission error counter bit |
|---|---|
| 0 to 255 | Number of transmission errors. These bits reflect the status of the transmission error counter. The number of errors is defined by the CAN protocol. |

**Remark** The TEC7 to TEC0 bits of the transmission error counter are invalid in the bus-off status (C0INFO.BOFF bit = 1).

## 20.4 Cautions

**(1) Clock accuracy**

To operate the USB function controller, the internal clock (6 MHz external clock × internal clock multiplied by 8 = 48 MHz internal clock) or external clock (external clock input to UCLK pin ($f_{USB}$ = 48 MHz)) must be used as the USB clock. When the internal clock is used as the USB clock, use a resonator with an accuracy of 6 MHz ±500 ppm (max.). When the external clock is used, apply a clock with an accuracy of 48 MHz ±500 ppm (max.) to the UCLK pin. If the USB clock accuracy drops, the transmission data cannot satisfy the USB rating.

**(2) Stopping the USB clock**

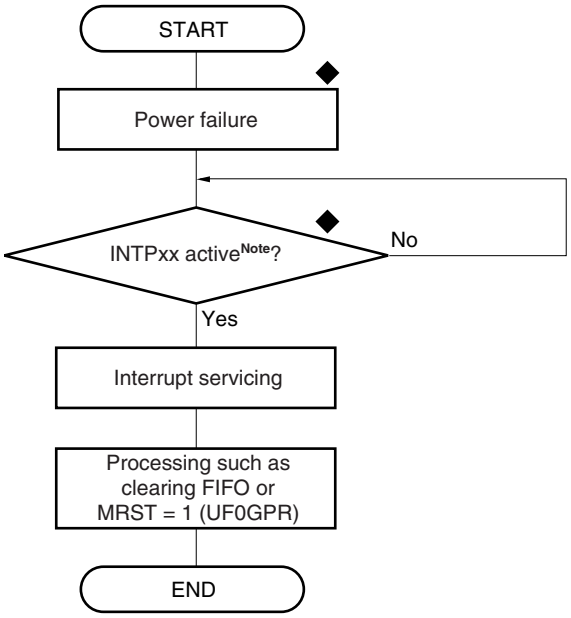
When the main clock ($f_{XX}$) has been selected as the USB function controller clock and it is necessary to stop the USB function controller, be sure to stop the USB function controller (by setting bits 1 and 0 of the UFCKMSK register to 1) first before stopping the main clock ($f_{XX}$).

If the main clock ($f_{XX}$) is stopped without first stopping the USB function controller, a malfunction might occur due to noise in the clock pulse when the main clock ($f_{XX}$) is restarted.

Similarly, when an external clock whose signal is input from the EXCLK pin is selected as the USB function controller clock, measures must be taken to prevent noise from being generated in the clock pulse by the external circuit. If this is not feasible, then the USB function controller must be stopped first before stopping the main clock ($f_{XX}$).

**Figure 20-31. Example of Processing After Power Application/Power Failure (3/3)**

**(b) Processing on power failure**




**Note** INTPxx indicates the external interrupt pins of the V850ES/JC3-H and V850ES/JE3-H (INTP00 to INTP18), and also indicates interrupts input by the external trigger pins (TIAA00, TAA01, TIAA10, TIAA11, TIAA20, TIAA21, TIAB10, TRGAB1, TIT00) of the timer.
Allocate one external interrupt pin to the following applications.

- Detecting disconnection of the connector in the case of self-powered mode (SFPW bit of UF0DSTL register = 1). In this case, monitor the VDD line of the USB connector, and input the result to the external interrupt pin at the edge. Note that the noise elimination time is that of the interrupt input pin, and that of each timer.
- Detecting turning off power from the HUB when the device is mounted on the same board as a HUB chip.

**Remark** ♦: Processing by hardware

**Table 22-2. Interrupt Sources (1/4)**

| Type | Classification | Default Priority | Name | Trigger | Generating Unit | Exception Code | Handler Address | Restored PC | Interrupt Control Register |
|---|---|---|---|---|---|---|---|---|---|
| Reset | Interrupt | – | RESET | $\overline{\text{RESET}}$ pin input/reset input by internal source | RESET | 0000H | 00000000H | Undefined | – |
| Non-maskable | Interrupt | – | NMI Note 1 | NMI pin valid edge input | Pin | 0010H | 00000010H | nextPC | – |
| | | – | INTWDT2 | WDT2 overflow | WDT2 | 0020H | 00000020H | **Note 2** | – |
| Software exception | Exception | – | TRAP0n Note 3 | TRAP instruction | – | 004nH Note 3 | 00000040H | nextPC | – |
| | | – | TRAP1n Note 3 | TRAP instruction | – | 005nH Note 3 | 00000050H | nextPC | – |
| Exception trap | Exception | – | ILGOP/ DBG0 | Illegal opcode/DBTRAP instruction | – | 0060H | 00000060H | nextPC | – |
| Maskable | Interrupt | 0 | INTLVI | Low voltage detection | POCLVI | 0080H | 00000080H | nextPC | LVIIC |
| | | 3 | INTP02 | External interrupt pin input edge detection (INTP02) | Pin | 00B0H | 000000B0H | nextPC | PIC02 |
| | | 6 | INTP05 | External interrupt pin input edge detection (INTP05) | Pin | 00E0H | 000000E0H | nextPC | PIC05 |
| | | 8 | INTP07 | External interrupt pin input edge detection (INTP07) | Pin | 0100H | 00000100H | nextPC | PIC07 |
| | | 9 | INTP08 | External interrupt pin input edge detection (INTP08) | Pin | 0110H | 00000110H | nextPC | PIC08 |
| | | 10 | INTP09 | External interrupt pin input edge detection (INTP09) | Pin | 0120H | 00000120H | nextPC | PIC09 |
| | | 11 | INTP10 | External interrupt pin input edge detection (INTP10) | Pin | 0130H | 00000130H | nextPC | PIC10 |
| | | 12 | INTP11 | External interrupt pin input edge detection (INTP11) | Pin | 0140H | 00000140H | nextPC | PIC11 |
| | | 15 | INTP14 | External interrupt pin input edge detection (INTP14) | Pin | 0170H | 00000170H | nextPC | PIC14 |
| | | 16 | INTP15 | External interrupt pin input edge detection (INTP15) | Pin | 0180H | 00000180H | nextPC | PIC15 |
| | | 17 | INTP16 | External interrupt pin input edge detection (INTP16) | Pin | 0190H | 00000190H | nextPC | PIC16 |

**Notes 1.** V850ES/JE3-H only

**2.** For restoring in the case of INTWDT2, see **22.2.2 (2) From INTWDT2 signal**.

**3.** n = 0 to FH

**Remark** JC3-H: V850ES/JC3-H, JE3-H: V850ES/JE3-H

**Table 22-2. Interrupt Sources (2/4)**

| Type | Classification | Default Priority | Name | Trigger | Generating Unit | Exception Code | Handler Address | Restored PC | Interrupt Control Register |
|---|---|---|---|---|---|---|---|---|---|
| Maskable | Interrupt | 25 | INTTAB1OV[Note 1] | TAB1 overflow | TAB1 | 0210H | 00000210H | nextPC | TAB1OVIC |
| | | 26 | INTTAB1CC0[Note 2] | TAB1 capture 0/ compare 0 match | TAB1 | 0220H | 00000220H | nextPC | TAB1CCIC0 |
| | | 27 | INTTAB1CC1 | TAB1 capture 1/ compare 1 match | TAB1 | 0230H | 00000230H | nextPC | TAB1CCIC1 |
| | | 28 | INTTAB1CC2 | TAB1 capture 2/ compare 2 match | TAB1 | 0240H | 00000240H | nextPC | TAB1CCIC2 |
| | | 29 | INTTAB1CC3 | TAB1 capture 3/ compare 3 match | TAB1 | 0250H | 00000250H | nextPC | TAB1CCIC3 |
| | | 30 | INTTT0OV | TMT0 overflow | TMT0 | 0260H | 00000260H | nextPC | TT0OVIC |
| | | 31 | INTTT0CC0 | TMT0 capture 0/ compare 0 match | TMT0 | 0270H | 00000270H | nextPC | TT0CCIC0 |
| | | 32 | INTTT0CC1 | TMT0 capture 1/ compare 1 match | TMT0 | 0280H | 00000280H | nextPC | TT0CCIC1 |
| | | 33 | INTTT0EC | TMT0 encoder input | TMT0 | 0290H | 00000290H | nextPC | TT0ECIC |
| | | 34 | INTTAA0OV | TAA0 overflow | TAA0 | 02A0H | 000002A0H | nextPC | TAA0OVIC |
| | | 35 | INTTAA0CC0 | TAA0 capture 0/ compare 0 match | TAA0 | 02B0H | 000002B0H | nextPC | TAA0CCIC0 |
| | | 36 | INTTAA0CC1 | TAA0 capture 1/ compare 1 match | TAA0 | 02C0H | 000002C0H | nextPC | TAA0CCIC1 |
| | | 37 | INTTAA1OV | TAA1 overflow | TAA1 | 02D0H | 000002D0H | nextPC | TAA1OVIC |
| | | 38 | INTTAA1CC0 | TAA1 capture 0/ compare 0 match | TAA1 | 02E0H | 000002E0H | nextPC | TAA1CCIC0 |
| | | 39 | INTTAA1CC1 | TAA1 capture 1/ compare 1 match | TAA1 | 02F0H | 000002F0H | nextPC | TAA1CCIC1 |
| | | 40 | INTTAA2OV | TAA2 overflow | TAA2 | 0300H | 00000300H | nextPC | TAA2OVIC |
| | | 41 | INTTAA2CC0 | TAA2 capture 0/ compare 0 match | TAA2 | 0310H | 00000310H | nextPC | TAA2CCIC0 |
| | | 42 | INTTAA2CC1 | TAA2 capture 1/ compare 1 match | TAA2 | 0320H | 00000320H | nextPC | TAA2CCIC1 |
| | | 52 | INTTM0EQ0 | TMM0 compare match | TMM0 | 03C0H | 000003C0H | nextPC | TM0EQIC0 |
| | | 53 | INTTM1EQ0 | TMM1 compare match | TMM1 | 03D0H | 000003D0H | nextPC | TM1EQIC0 |
| | | 54 | INTTM2EQ0 | TMM2 compare match | TMM2 | 03E0H | 000003E0H | nextPC | TM2EQIC0 |
| | | 55 | INTTM3EQ0 | TMM3 compare match | TMM3 | 03F0H | 000003F0H | nextPC | TM3EQIC0 |
| | | 56 | INTCF0R /INTIIC1 | CSIF0 reception completion/ CSIF0 reception error/ IIC1 transfer completion | CSIF0/ IIC1 | 0400H | 00000400H | nextPC | CF0RIC/ IICIC1 |
| | | 57 | INTCF0T | CSIF0 consecutive transmission write enable | CSIF0 | 0410H | 00000410H | nextPC | CF0TIC |

**Notes 1.** When using TAB1 in the 6-phase PWM output mode, functions as the zero match interrupt (TAB1TIOD) request from TMQOP.

**2.** When using TAB1 in the 6-phase PWM output mode, functions as the compare match interrupt (TAB1TICD0) request from TMQOP.

(3/6)

| Mnemonic | Operand | Opcode | Operation | | Execution Clock | | | Flags | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | i | r | l | CY | OV | S | Z | SAT |
| LD.H | disp16[reg1],reg2 | rrrrr111001RRRRR<br>ddddddddddddddd0<br>**Note 8** | adr←GR[reg1]+sign-extend(disp16)<br>GR[reg2]←sign-extend(Load-memory(adr,Halfword)) | | 1 | 1 | Note 11 | | | | | |
| LDSR | reg2,regID | rrrrr111111RRRRR<br>0000000000100000<br>**Note 12** | SR[regID]←GR[reg2] | Other than regID = PSW | 1 | 1 | 1 | | | | | |
| | | | | regID = PSW | 1 | 1 | 1 | × | × | × | × | × |
| LD.HU | disp16[reg1],reg2 | rrrrr111111RRRRR<br>ddddddddddddddd1<br>**Note 8** | adr←GR[reg1]+sign-extend(disp16)<br>GR[reg2]←zero-extend(Load-memory(adr,Halfword) | | 1 | 1 | Note 11 | | | | | |
| LD.W | disp16[reg1],reg2 | rrrrr111001RRRRR<br>ddddddddddddddd1<br>**Note 8** | adr←GR[reg1]+sign-extend(disp16)<br>GR[reg2]←Load-memory(adr,Word) | | 1 | 1 | Note 11 | | | | | |
| MOV | reg1,reg2 | rrrrr000000RRRRR | GR[reg2]←GR[reg1] | | 1 | 1 | 1 | | | | | |
| | imm5,reg2 | rrrrr010000iiiii | GR[reg2]←sign-extend(imm5) | | 1 | 1 | 1 | | | | | |
| | imm32,reg1 | 00000110001RRRRR<br>iiiiiiiiiiiiiiii<br>IIIIIIIIIIIIIIII | GR[reg1]←imm32 | | 2 | 2 | 2 | | | | | |
| MOVEA | imm16,reg1,reg2 | rrrrr110001RRRRR<br>iiiiiiiiiiiiiiii | GR[reg2]←GR[reg1]+sign-extend(imm16) | | 1 | 1 | 1 | | | | | |
| MOVHI | imm16,reg1,reg2 | rrrrr110010RRRRR<br>iiiiiiiiiiiiiiii | GR[reg2]←GR[reg1]+(imm16 ll $0^{16}$) | | 1 | 1 | 1 | | | | | |
| MUL | reg1,reg2,reg3 | rrrrr111111RRRRR<br>wwwww01000100000 | GR[reg3] ll GR[reg2]←GR[reg2]xGR[reg1]<br>**Note 14** | | 1 | 4 | 5 | | | | | |
| | imm9,reg2,reg3 | rrrrr111111iiiii<br>wwwww01001IIII00<br>**Note 13** | GR[reg3] ll GR[reg2]←GR[reg2]xsign-extend(imm9) | | 1 | 4 | 5 | | | | | |
| MULH | reg1,reg2 | rrrrr000111RRRRR | GR[reg2]←GR[reg2]Note 6xGR[reg1]Note 6 | | 1 | 1 | 2 | | | | | |
| | imm5,reg2 | rrrrr010111iiiii | GR[reg2]←GR[reg2]Note 6xsign-extend(imm5) | | 1 | 1 | 2 | | | | | |
| MULHI | imm16,reg1,reg2 | rrrrr110111RRRRR<br>iiiiiiiiiiiiiiii | GR[reg2]←GR[reg1]Note 6ximm16 | | 1 | 1 | 2 | | | | | |
| MULU | reg1,reg2,reg3 | rrrrr111111RRRRR<br>wwwww01000100010 | GR[reg3] ll GR[reg2]←GR[reg2]xGR[reg1]<br>**Note 14** | | 1 | 4 | 5 | | | | | |
| | imm9,reg2,reg3 | rrrrr111111iiiii<br>wwwww01001IIII10<br>**Note 13** | GR[reg3] ll GR[reg2]←GR[reg2]xzero-extend(imm9) | | 1 | 4 | 5 | | | | | |
| NOP | | 0000000000000000 | Pass at least one clock cycle doing nothing. | | 1 | 1 | 1 | | | | | |
| NOT | reg1,reg2 | rrrrr000001RRRRR | GR[reg2]←NOT(GR[reg1]) | | 1 | 1 | 1 | | 0 | × | × | |
| NOT1 | bit#3,disp16[reg1] | 01bbb111110RRRRR<br>ddddddddddddddd | adr←GR[reg1]+sign-extend(disp16)<br>Z flag←Not(Load-memory-bit(adr,bit#3))<br>Store-memory-bit(adr,bit#3,Z flag) | | 3 Note 3 | 3 Note 3 | 3 Note 3 | | | | × | |
| | reg2,[reg1] | rrrrr111111RRRRR<br>0000000011100010 | adr←GR[reg1]<br>Z flag←Not(Load-memory-bit(adr,reg2))<br>Store-memory-bit(adr,reg2,Z flag) | | 3 Note 3 | 3 Note 3 | 3 Note 3 | | | | × | |

(6/6)

| Mnemonic | Operand | Opcode | Operation | Execution Clock | | | Flags | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | i | r | l | CY | OV | S | Z | SAT |
| SUB | reg1,reg2 | rrrrr001101RRRRR | GR[reg2]←GR[reg2]–GR[reg1] | 1 | 1 | 1 | × | × | × | × | |
| SUBR | reg1,reg2 | rrrrr001100RRRRR | GR[reg2]←GR[reg1]–GR[reg2] | 1 | 1 | 1 | × | × | × | × | |
| SWITCH | reg1 | 00000000010RRRRR | adr←(PC+2) + (GR [reg1] logically shift left by 1)<br>PC←(PC+2) + (sign-extend<br>(Load-memory (adr,Halfword))<br>logically shift left by 1 | 5 | 5 | 5 | | | | | |
| SXB | reg1 | 00000000101RRRRR | GR[reg1]←sign-extend<br>(GR[reg1] (7 : 0)) | 1 | 1 | 1 | | | | | |
| SXH | reg1 | 00000000111RRRRR | GR[reg1]←sign-extend<br>(GR[reg1] (15 : 0)) | 1 | 1 | 1 | | | | | |
| TRAP | vector | 0 0 0 0 0 1 1 1 1 1 1 i i i i i<br>0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 | EIPC ←PC+4 (Restored PC)<br>EIPSW ←PSW<br>ECR.EICC ←Interrupt code<br>PSW.EP ←1<br>PSW.ID ←1<br>PC ←00000040H<br>(when vector is 00H to 0FH)<br>00000050H<br>(when vector is 10H to 1FH) | 3 | 3 | 3 | | | | | |
| TST | reg1,reg2 | rrrrr001011RRRRR | result←GR[reg2] AND GR[reg1] | 1 | 1 | 1 | | 0 | × | × | |
| TST1 | bit#3,disp16[reg1] | 11bbb111110RRRRR<br>ddddddddddddddd | adr←GR[reg1]+sign-extend(disp16)<br>Z flag←Not (Load-memory-bit (adr,bit#3)) | 3<br>Note 3 | 3<br>Note 3 | 3<br>Note 3 | | | | × | |
| | reg2, [reg1] | rrrrr111111RRRRR<br>0000000011100110 | adr←GR[reg1]<br>Z flag←Not (Load-memory-bit (adr,reg2)) | 3<br>Note 3 | 3<br>Note 3 | 3<br>Note 3 | | | | × | |
| XOR | reg1,reg2 | rrrrr001001RRRRR | GR[reg2]←GR[reg2] XOR GR[reg1] | 1 | 1 | 1 | | 0 | × | × | |
| XORI | imm16,reg1,reg2 | rrrrr110101RRRRR<br>i i i i i i i i i i i i i i i i | GR[reg2]←GR[reg1] XOR zero-extend (imm16) | 1 | 1 | 1 | | 0 | × | × | |
| ZXB | reg1 | 00000000100RRRRR | GR[reg1]←zero-extend (GR[reg1] (7 : 0)) | 1 | 1 | 1 | | | | | |
| ZXH | reg1 | 00000000110RRRRR | GR[reg1]←zero-extend (GR[reg1] (15 : 0)) | 1 | 1 | 1 | | | | | |

**Notes 1.** ddddddddd: Higher 8 bits of disp9.

**2.** 3 if there is an instruction that rewrites the contents of the PSW immediately before.

**3.** If there is no wait state (3 + the number of read access wait states).

**4.** n is the total number of list12 load registers. (According to the number of wait states. Also, if there are no wait states, n is the total number of list12 registers. If n = 0, same operation as when n = 1)

**5.** RRRRR: other than 00000.

**6.** The lower halfword data only are valid.

**7.** ddddddddddddddddddddd: The higher 21 bits of disp22.

**8.** ddddddddddddddd: The higher 15 bits of disp16.

**9.** According to the number of wait states (1 if there are no wait states).

**10.** b: bit 0 of disp16.

**11.** According to the number of wait states (2 if there are no wait states).