To our customers,

## Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

# Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   “Specific”: Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

RENESAS

# Application Note

# V850ES/Hx2

## 32-bit Single-Chip Microcontrollers

## Flash Memory Programming (Programmer)

**μPD70F3700**
**μPD70F3701**
**μPD70F3702**
**μPD70F3703**
**μPD70F3704**
**μPD70F3706**
**μPD70F3707**
**μPD70F3709**
**μPD70F3710**
**μPD70F3711**
**μPD70F3712**

Document No. U18215EJ1V0AN00 (1st edition)
Date Published August 2006 NS CP(K)

**[MEMO]**

## NOTES FOR CMOS DEVICES

### ① VOLTAGE APPLICATION WAVEFORM AT INPUT PIN

Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between $V_{IL}$ (MAX) and $V_{IH}$ (MIN) due to noise, etc., the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between $V_{IL}$ (MAX) and $V_{IH}$ (MIN).

### ② HANDLING OF UNUSED INPUT PINS

Unconnected CMOS device inputs can be cause of malfunction. If an input pin is unconnected, it is possible that an internal input level may be generated due to noise, etc., causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using pull-up or pull-down circuitry. Each unused pin should be connected to $V_{DD}$ or GND via a resistor if there is a possibility that it will be an output pin. All handling related to unused pins must be judged separately for each device and according to related specifications governing the device.

### ③ PRECAUTION AGAINST ESD

A strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it when it has occurred. Environmental control must be adequate. When it is dry, a humidifier should be used. It is recommended to avoid using insulators that easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors should be grounded. The operator should be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with mounted semiconductor devices.

### ④ STATUS BEFORE INITIALIZATION

Power-on does not necessarily define the initial status of a MOS device. Immediately after the power source is turned ON, devices with reset functions have not yet been initialized. Hence, power-on does not guarantee output pin levels, I/O settings or contents of registers. A device is not initialized until the reset signal is received. A reset operation must be executed immediately after power-on for devices with reset functions.

### ⑤ POWER ON/OFF SEQUENCE

In the case of a device that uses different power supplies for the internal operation and external interface, as a rule, switch on the external power supply after switching on the internal power supply. When switching the power supply off, as a rule, switch off the external power supply and then the internal power supply. Use of the reverse power on/off sequences may result in the application of an overvoltage to the internal elements of the device, causing malfunction and degradation of internal elements due to the passage of an abnormal current.

The correct power on/off sequence must be judged separately for each device and according to related specifications governing the device.

### ⑥ INPUT OF SIGNAL DURING POWER OFF STATE

Do not input signals or an I/O pull-up power supply while the device is not powered. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements.

Input of signals during the power off state must be judged separately for each device and according to related specifications governing the device.

- **The information in this document is current as of August, 2006. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC Electronics sales representative for availability and additional information.**
- No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.
- NEC Electronics does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC Electronics products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of a customer's equipment shall be done under the full responsibility of the customer. NEC Electronics assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.
- NEC Electronics products are classified into the following three quality grades: "Standard", "Special" and "Specific".

  The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of an NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.

  "Standard": Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment and industrial robots.

  "Special": Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime systems, safety equipment and medical equipment (not specifically designed for life support).

  "Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.

The quality grade of NEC Electronics products is "Standard" unless otherwise expressly specified in NEC Electronics data sheets or data books, etc. If customers wish to use NEC Electronics products in applications not intended by NEC Electronics, they must contact an NEC Electronics sales representative in advance to determine NEC Electronics' willingness to support a given application.

(Note)

(1) "NEC Electronics" as used in this statement means NEC Electronics Corporation and also includes its majority-owned subsidiaries.

(2) "NEC Electronics products" means any product developed or manufactured by or for NEC Electronics (as defined above).

M8E 02. 11-1

# INTRODUCTION

**Target Readers**

This application note is intended for users who understand the functions of the V850ES/Hx2 and who will use this product to design application systems.

**Purpose**

The purpose of this application note is to help users understand how to develop dedicated flash memory programmers for rewriting the internal flash memory of the V850ES/Hx2.

The sample programs and circuit diagrams shown in this document are for reference only and are not intended for use in actual design-ins.

Therefore, these sample programs must be used at the user's own risk. Correct operation is not guaranteed if these sample programs are used.

**Organization**

This manual consists of the following main sections.

- Flash memory programming
- Programmer operating environment
- Basic programmer operation
- Command/data frame format
- Description of command processing
- UART communication mode
- 3-wire serial I/O communication mode with handshake supported (CSI + HS)
- 3-wire serial I/O communication mode (CSI)
- Flash memory programming parameter characteristics

**How to Read This Manual**

It is assumed that the reader of this manual has general knowledge in the fields of electrical engineering, logic circuits, and microcontrollers.

☐ To learn more about the V850ES/Hx2's hardware functions:

→ See the user's manual of each V850ES/Hx2 product.

**Conventions**

| | |
|---|---|
| Data significance: | Higher digits on the left and lower digits on the right |
| Active low representation: | $\overline{xxx}$ (overscore over pin or signal name) |
| **Note**: | Footnote for item marked with **Note** in the text |
| **Caution**: | Information requiring particular attention |
| **Remark**: | Supplementary information |
| Numeral representation: | Binary.....xxxx or xxxxB |
| | Decimal.....xxxx |
| | Hexadecimal .....xxxxH |

**Related Documents** The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such.

**Device-related documents**

| Document Name | Document Number |
|---|---|
| V850ES/HE2 User's Manual | U17720E |
| V850ES/HF2 User's Manual | U17719E |
| V850ES/HG2 User's Manual | U17718E |
| V850ES/HJ2 User's Manual | U17717E |
| V850ES Architecture User's Manual | U15943E |

**CONTENTS**

# CHAPTER 1 FLASH MEMORY PROGRAMMING

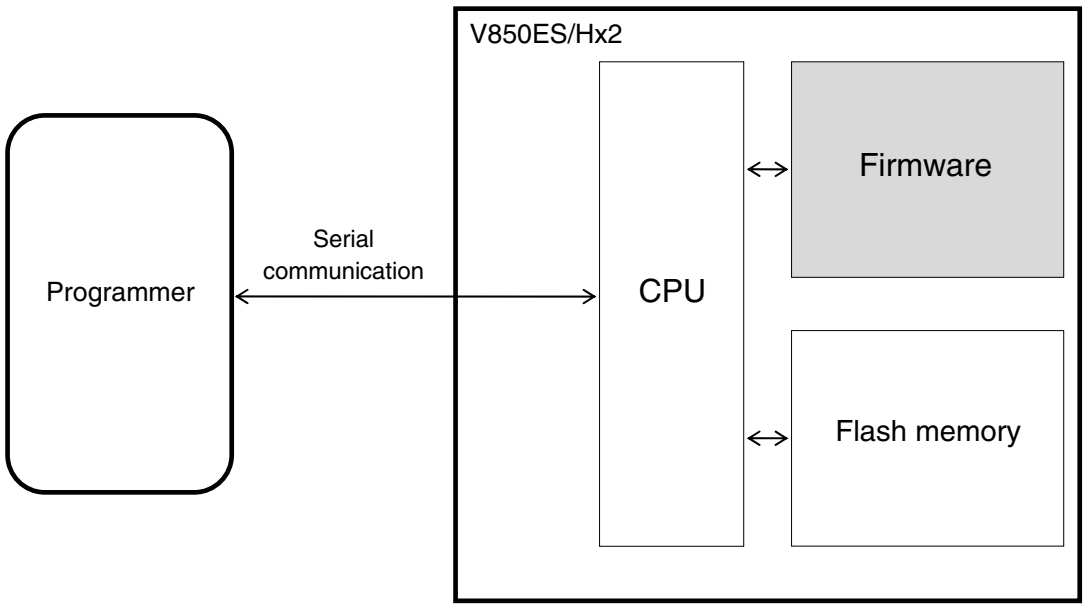
To rewrite the contents of the internal flash memory of the V850ES/Hx2, a dedicated flash memory programmer (hereafter referred to as the “programmer”) is usually used.

This Application Note explains how to develop a dedicated programmer.

## 1.1 Overview

The V850ES/Hx2 incorporates firmware that controls flash memory programming. The programming to the internal flash memory is performed by transmitting/receiving commands between the programmer and the V850ES/Hx2 via serial communication.

**Figure 1-1. System Outline of Flash Memory Programming in V850ES/Hx2**



## 1.2 System Configuration

Examples of the system configuration for programming the flash memory are illustrated in Figure 1-2.

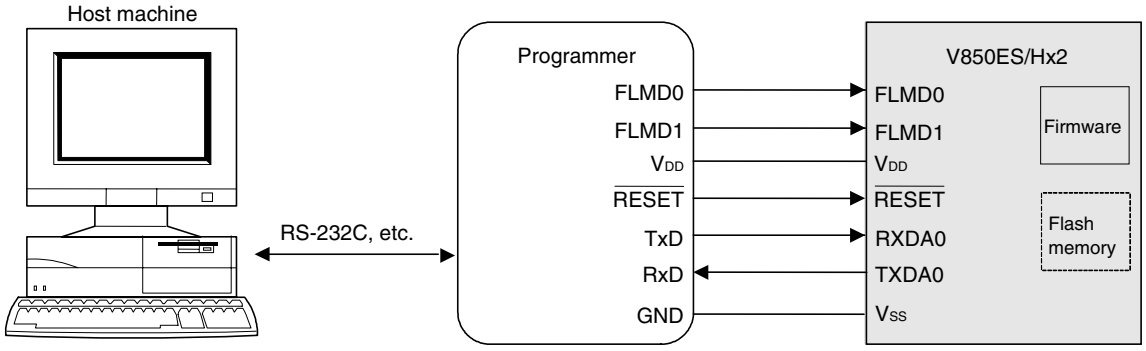
These figures illustrate how to program the flash memory with the programmer, under control of a host machine.

Depending on how the programmer is connected, the programmer can be used in a standalone mode without using the host machine, if a user program has been downloaded to the programmer in advance.

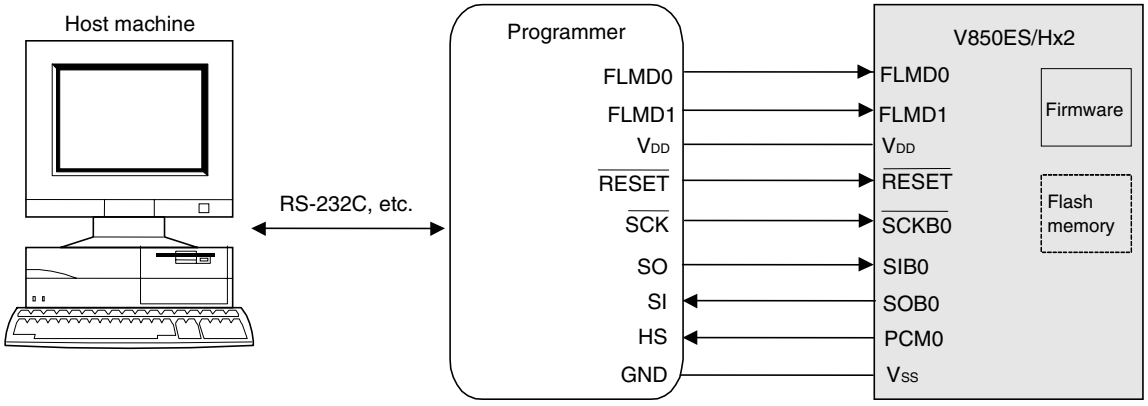
For example, NEC Electronics' flash memory programmer PG-FP4 can execute programming either by using the GUI software with a host machine connected or by itself (standalone).

**Figure 1-2. System Configuration Examples**

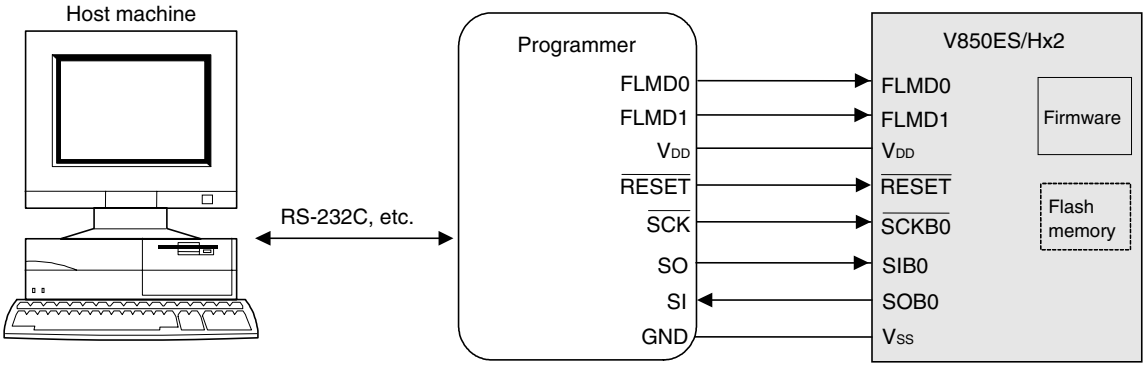
**(1) UART communication mode (LSB-first transfer)**




**(2) 3-wire serial I/O communication mode with handshake supported (CSI + HS) (MSB-first transfer)**



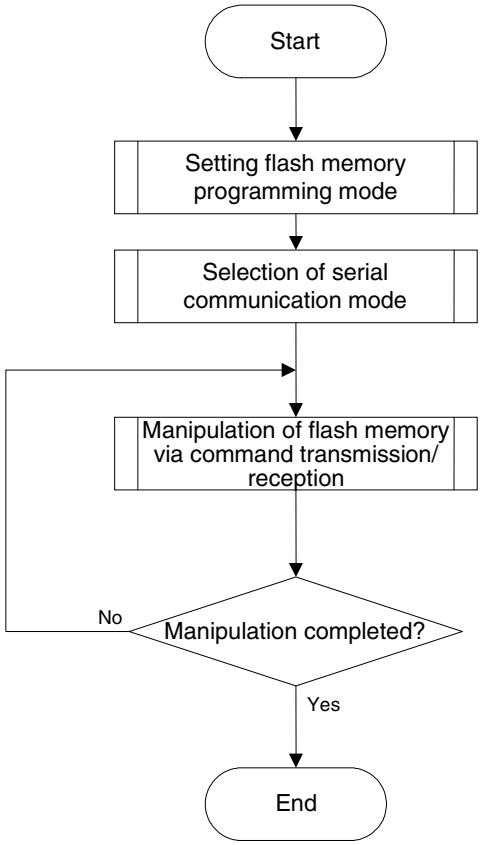

**(3) 3-wire serial I/O communication mode (CSI) (MSB-first transfer)**



## 1.3 Programming Overview

To rewrite the contents of the flash memory with the programmer, the V850ES/Hx2 must first be set to the flash memory programming mode. After that, select the mode for communication between the programmer and the V850ES/Hx2, transmit commands from the programmer via serial communication, and then rewrite the flash memory. The flowchart of programming is illustrated in Figure 1-3.

**Figure 1-3. Programming Flowchart**




### 1.3.1 Setting flash memory programming mode

Supply a specific voltage to the flash memory programming mode setting pins (FLMD0 and FLMD1) in the V850ES/Hx2 and release a reset; the flash memory programming mode is then set.

### 1.3.2 Selecting serial communication mode

To select a serial communication mode, generate pulses by changing the voltage at flash memory programming mode setting pin (FLMD0) between the $V_{DD}$ voltage and GND voltage in the flash memory programming mode, and determine the communication mode according to the pulse count.

### 1.3.3 Manipulating flash memory via command transmission/reception

The flash memory incorporated in the V850ES/Hx2 has functions to rewrite the flash memory contents. The flash memory manipulating functions shown in Table 1-1 are available.

**Table 1-1. Outline of Flash Memory Functions**

| Function | Outline |
|---|---|
| Erase | Erases the flash memory contents. |
| Write | Writes data to the flash memory. |
| Verify | Compares the flash memory contents with data for verify. |
| Acquisition of information | Reads information related to the flash memory. |

To control these functions, the programmer transmits commands to the V850ES/Hx2 via serial communication. The V850ES/Hx2 returns the response status for the commands. The programming to the flash memory is performed by repeating these series of serial communications.

## 1.4 Information Specific to V850ES/Hx2

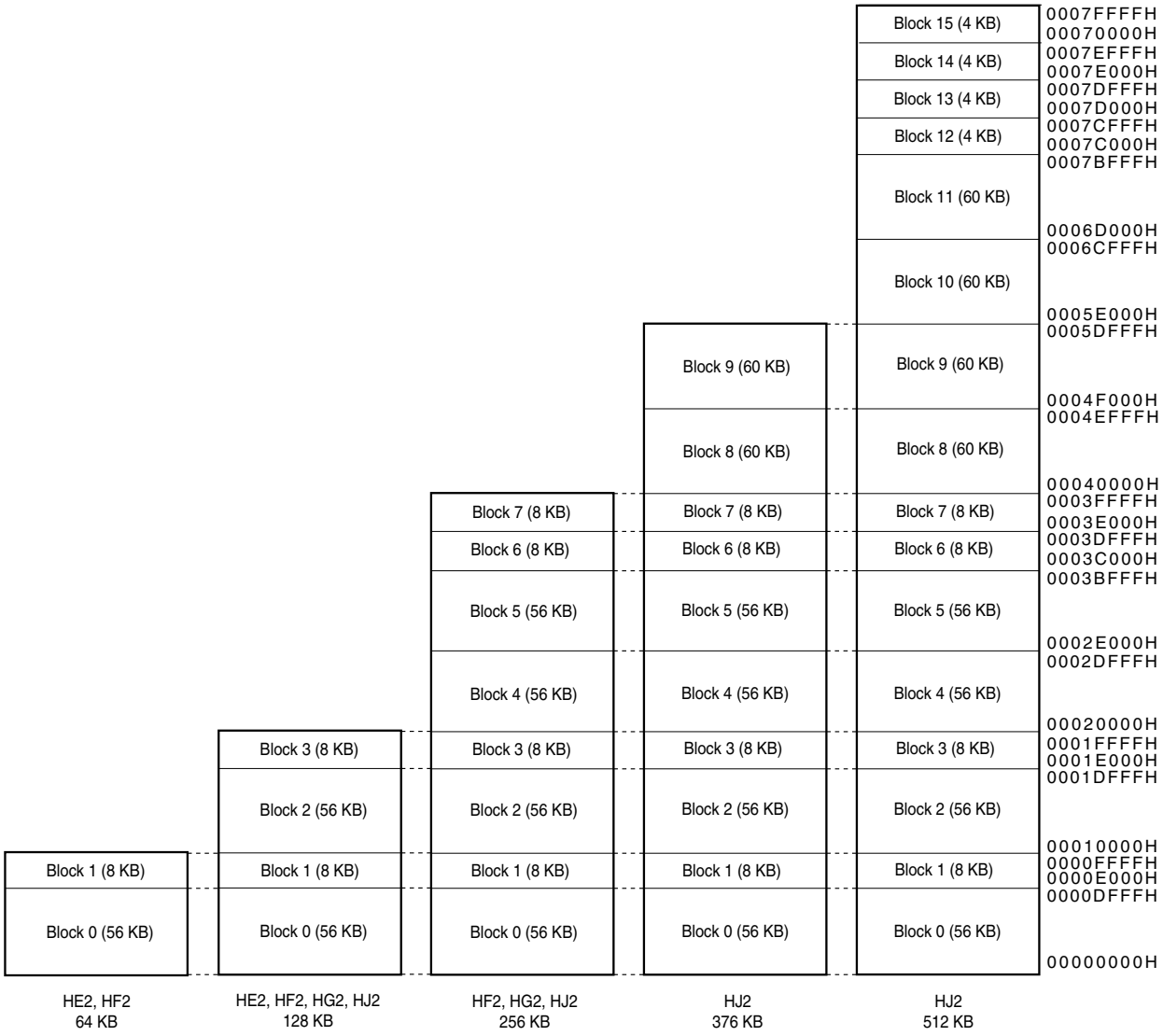
The programmer must manage product-specific information (such as a device name and memory information).

Table 1-2 shows the flash memory size of the V850ES/Hx2 and Figure 1-4 shows the configuration of the flash memory.

**Table 1-2. Flash Memory Size of V850ES/Hx2**

| Device Name | | Flash Memory Size |
|---|---|---|
| V850ES/HE2 | μPD70F3700 | 64 KB |
| | μPD70F3701 | 128 KB |
| V850ES/HF2 | μPD70F3702 | 64 KB |
| | μPD70F3703 | 128 KB |
| | μPD70F3704 | 256 KB |
| V850ES/HG2 | μPD70F3706 | 128 KB |
| | μPD70F3707 | 256 KB |
| V850ES/HJ2 | μPD70F3709 | 128 KB |
| | μPD70F3710 | 256 KB |
| | μPD70F3711 | 376 KB |
| | μPD70F3712 | 512 KB |

**Figure 1-4. Flash Memory Configuration**



# CHAPTER 2 PROGRAMMER OPERATING ENVIRONMENT

## 2.1 Programmer Control Pins

Table 2-1 lists the pins that the programmer must control to implement the programmer function in the user system. See the following pages for details on each pin.

**Table 2-1. Pin Description**

| Programmer | | | V850ES/Hx2 | Mode for Communication with Target System | | |
|---|---|---|---|---|---|---|
| Signal Name | I/O | Pin Function | Pin Name | CSI | CSI + HS | UART |
| FLMD0 | Output | Output of signal level to set programming mode and output of pulse to select communication mode | FLMD0 | ○ | ○ | ○ |
| FLMD1 | Output | Output of signal level to set programming mode | FLMD1 | ○ | ○ | ○ |
| $V_{DD}$ | Output | $V_{DD}$ voltage generation/monitoring | $V_{DD}$ | △ | △ | △ |
| GND | – | Ground | $V_{SS}$ | ○ | ○ | ○ |
| CLK | Output | Operating clock output to V850ES/Hx2 | – | ×Note | ×Note | ×Note |
| $\overline{\text{RESET}}$ | Output | Programming mode switching trigger | $\overline{\text{RESET}}$ | ○ | ○ | ○ |
| SO | Output | Command transmission to V850ES/Hx2 | SIB0 | ○ | ○ | × |
| SI | Input | Response status and data reception from V850ES/Hx2 | SOB0 | ○ | ○ | × |
| $\overline{\text{SCK}}$ | Output | Serial clock supply to V850ES/Hx2 | $\overline{\text{SCKB0}}$ | ○ | ○ | × |
| HS (handshake) | Input | Handshake signal reception for serial communication with V850ES/Hx2 | PCM0 | × | ○ | × |
| TxD | Output | Command transmission to V850ES/Hx2 | RXDA0 | × | × | ○ |
| RxD | Input | Response status and data reception from V850ES/Hx2 | TXDA0 | × | × | ○ |

**Note** No clock can be supplied from the CLK pin of the programmer. Mount an oscillator circuit onto the target system to supply the clock.

**Remark** ○: Be sure to connect the pin.
×: The pin is not connected.
△: The pin does not have to be connected if the signal is generated in the user system.

For the voltage of the pins controlled by the programmer, refer to the user's manual of the device that is subject to flash memory programming.

## 2.2 Details of Control Pins

### 2.2.1 Flash memory programming mode setting pins (FLMD0, FLMD1)

The FLMD0 and FLMD1 pins are used to control the operating mode of the V850ES/Hx2. The V850ES/Hx2 operates in flash memory programming mode when a specific voltage is supplied to these pins and a reset is released.

The mode for the serial communication between the programmer and the V850ES/Hx2 is determined by controlling the voltage at the FLMD0 pin between $V_{DD}$ and GND and outputting pulses, after reset. Refer to **Table 2-3** in **2.5 Selecting Serial Communication Mode** for the relationship between the FLMD0 pulse counts and communication modes.

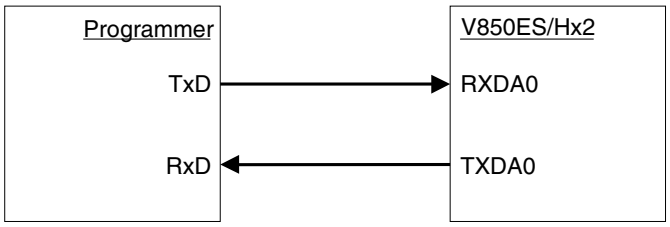
### 2.2.2 Serial interface pins (TxD, RxD, SI, SO, $\overline{SCK}$, HS)

The serial interface pins are used to transfer the flash memory writing commands between the programmer and the V850ES/Hx2.

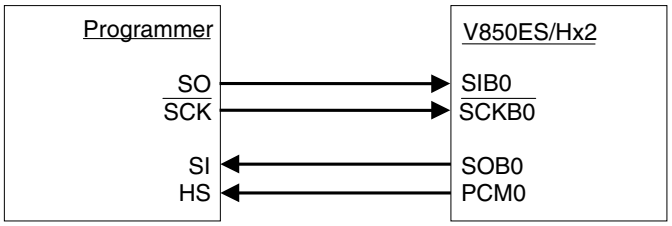
With the V850ES/Hx2, the communication mode can be selected from UART, CSI + HS, and CSI. The following figures illustrate the connection of pins used in each communication mode.

**Figure 2-1. Serial Interface Pins**

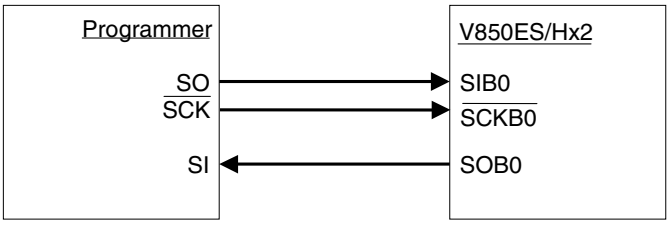
**(1) UART communication mode**




**(2) 3-wire serial I/O communication mode with handshake supported (CSI + HS)**



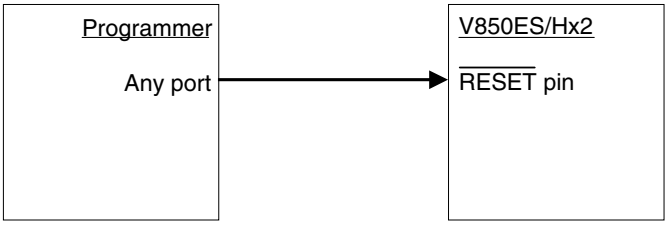

**(3) 3-wire serial I/O communication mode (CSI)**



### 2.2.3 Reset control pin ($\overline{\text{RESET}}$)

The reset control pin is used to control the system reset for the V850ES/Hx2 from the programmer. The flash memory programming mode can be selected when a specific voltage is supplied to the FLMD0 and FLMD1 pins and a reset is released.

**Figure 2-2. Reset Control Pin**




### 2.2.4 Clock control pin (CLK)

The clock control pin is not used.

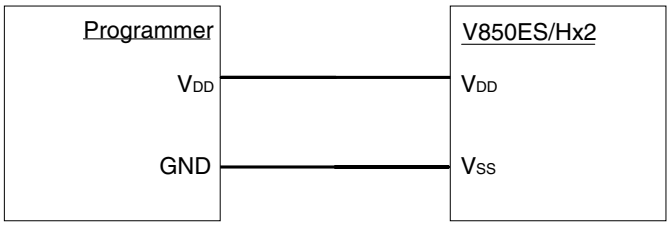
No clock can be supplied from the CLK pin of the programmer. Mount an oscillator circuit onto the target system to supply the clock.

### 2.2.5 $V_{DD}$/GND control pins

The $V_{DD}$ control pin is used to supply power to the V850ES/Hx2 from the programmer. Connection of this pin is not necessary when it is not necessary to supply power to the V850ES/Hx2 from the programmer. However, this pin must be connected regardless of whether the power is supplied from the programmer when the dedicated programmer is used, because the dedicated programmer monitors the power supply status of the V850ES/Hx2.

The GND control pin must be connected to $V_{SS}$ of the V850ES/Hx2 regardless of whether the power is supplied from the programmer.

**Figure 2-3. $V_{DD}$/GND Control Pin**



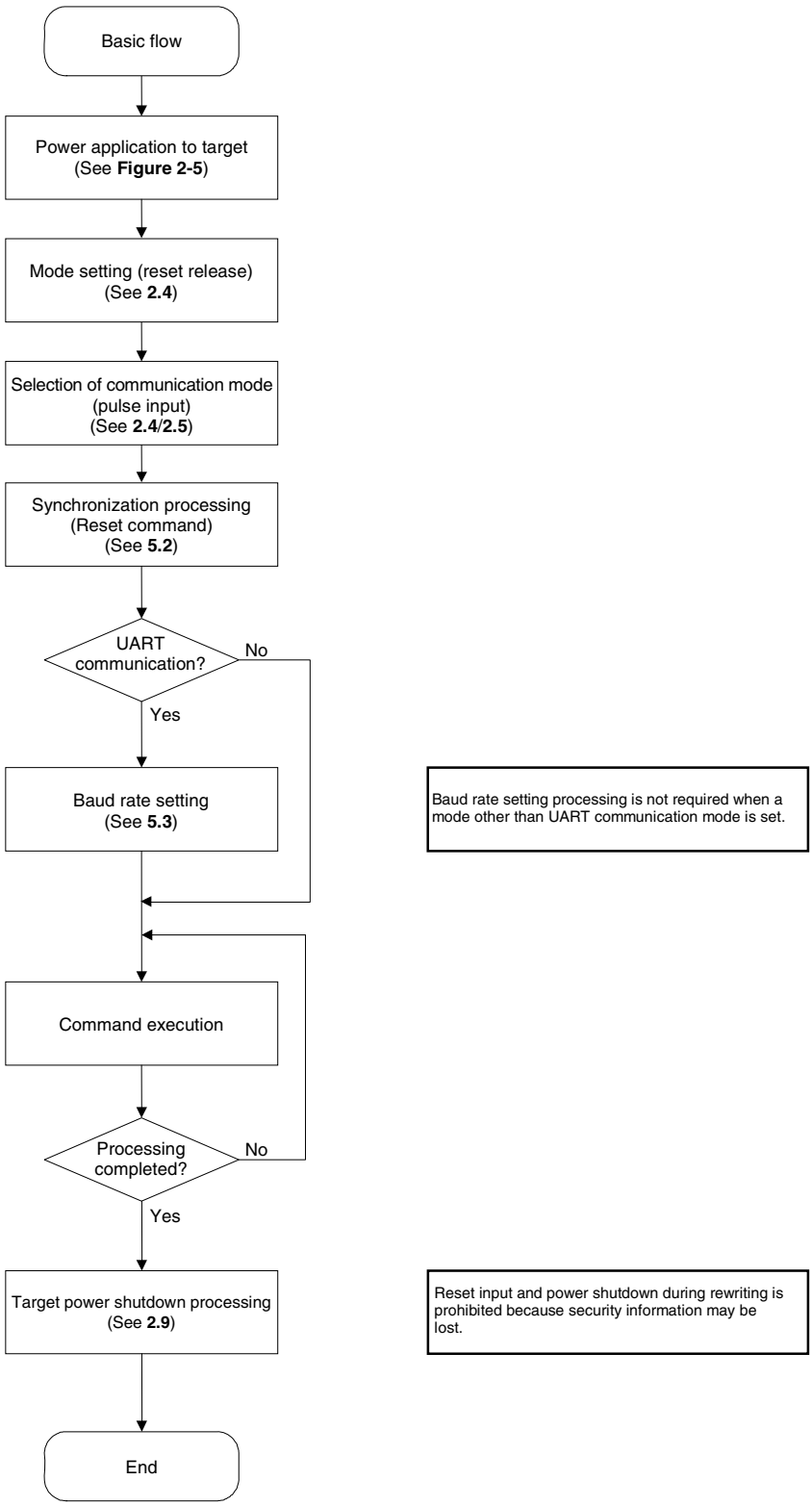

### 2.2.6 Other pins

For the connection of the pins that are not connected to the programmer, refer to the chapter describing the flash memory in the user's manual of each device.

## 2.3 Basic Flowchart

The following illustrates the basic flowchart for performing flash memory rewriting with the programmer.

**Figure 2-4. Basic Flowchart for Flash Memory Rewrite Processing**

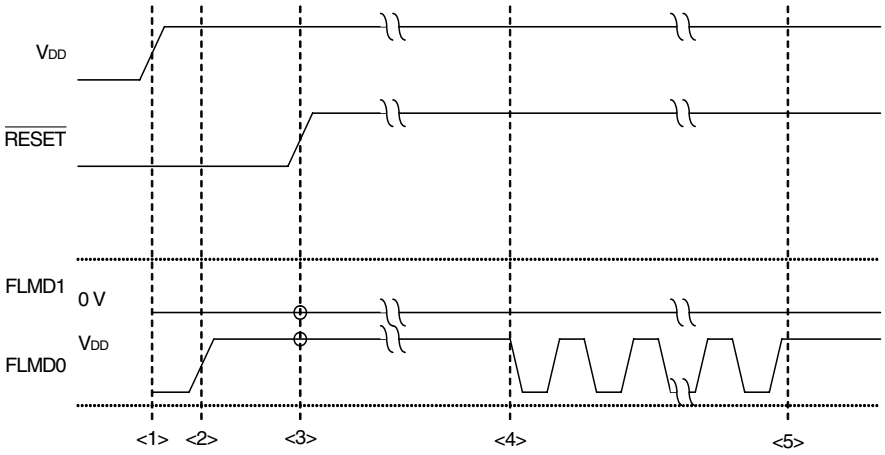


## 2.4 Setting Flash Memory Programming Mode

To rewrite the contents of the flash memory with the programmer, the V850ES/Hx2 must first be set to the flash memory programming mode by supplying a specific voltage to the flash memory programming mode setting pins (FLMD0, FLMD1) in the V850ES/Hx2, then releasing a reset.

The following illustrates a timing chart for setting the flash memory programming mode and selecting the communication mode.

**Figure 2-5. Setting Flash Memory Programming Mode and Selecting Communication Mode**



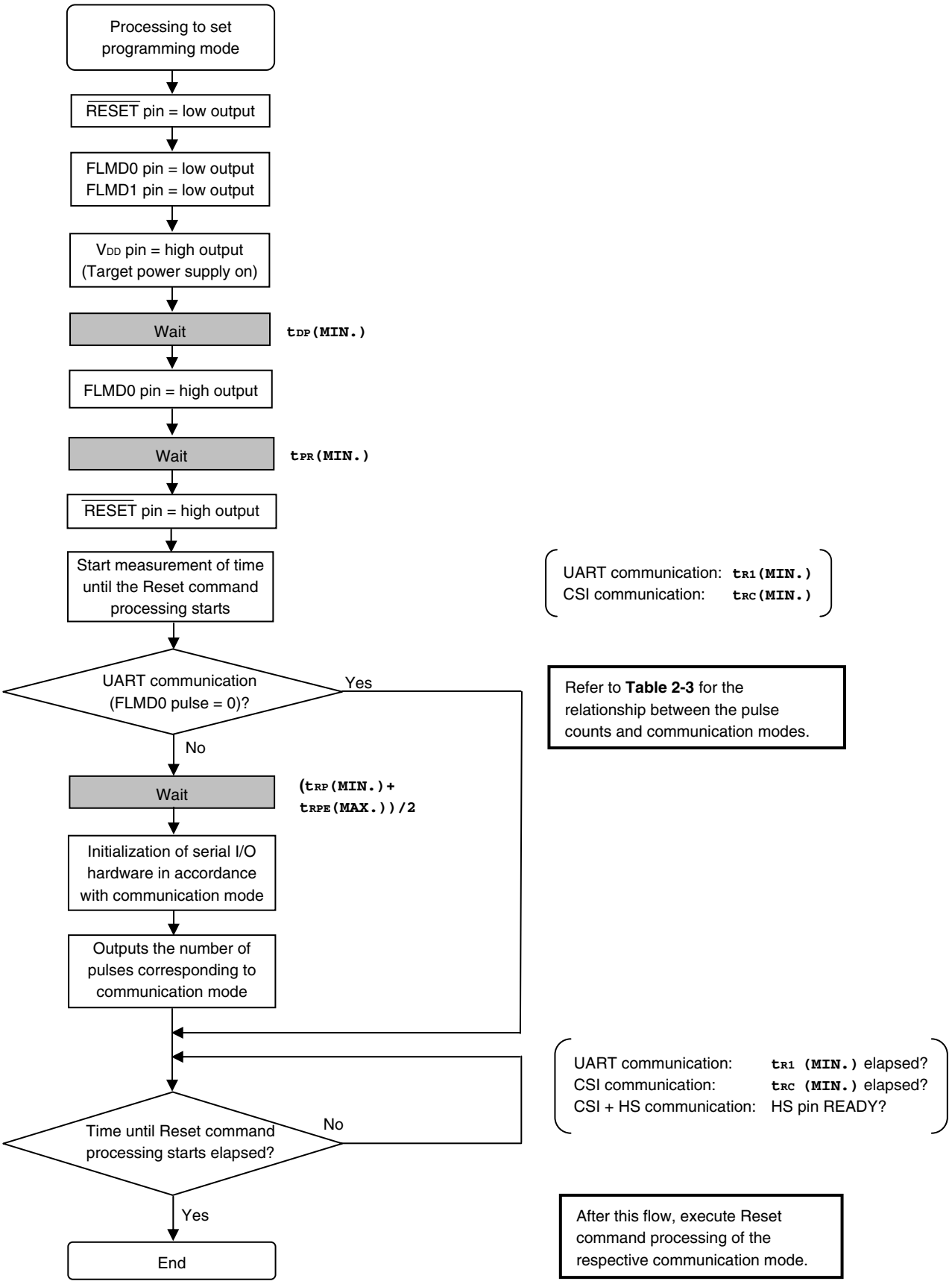

<1>: Power application ($V_{DD}$)
<2>: FLMD0 = high level, FLMD1 = low level
<3>: Reset release (mode setting)
<4>: Pulse output starts
<5>: Pulse output ends

The relationship between the settings of the FLMD0 and FLMD1 pins after reset release and the operating mode is shown below.

**Table 2-2. Relationship Between Settings of FLMD0 and FLMD1 Pins After Reset Release and Operating Mode**

| FLMD0 | FLMD1 | Operating Mode |
|---|---|---|
| Low (GND) | Any | Normal operating mode |
| High ($V_{DD}$) | Low (GND) | Flash memory programming mode |
| High ($V_{DD}$) | High ($V_{DD}$) | Setting prohibited |

### 2.4.1 Mode setting flowchart



### 2.4.2 Sample program

The following shows a sample program for mode setting.

```
/*************************************************************/
/*                                                           */
/*      connect to Flash device                              */
/*                                                           */
/*************************************************************/

void    fl_con_dev(void)
{
extern  void   init_fl_uart(void);
extern  void   init_fl_csi(void);

  int    n;
  int    pulse;

  SRMK0 = true;       // disable UART Rx INT.
  UARTE0 = false;     // disable UART H.W.

  switch (fl_if){     // preset pulse count by I/F
        default:
        case   FLIF_UART:   pulse = PULSE_UART;       break;
        case   FLIF_CSI:    pulse = PULSE_CSI;        break;
        case   FLIF_CSI_HS: pulse = PULSE_CSIHS;      break;

  }

  pFL_RES       = low;        // RESET = low
  pmFL_FLMD0    = PM_OUT;     // FLMD0 = Low output
  pFL_FLMD0     = low;
  pmFL_FLMD1    = PM_OUT;     // FLMD1 = Low output
  pFL_FLMD1     = low;
  FL_VDD_HI();                // VDD = high

  fl_wait(tDP);               // wait

  pFL_FLMD0     = hi;         // FLMD0 = high
  fl_wait(tPR);               // wait

  pFL_RES       = hi;         // RESET = high
  start_flto(tRC);            // start "tRC" wait timer
  fl_wait((tRP+tRPE)/2);      // wait

  if (fl_if == FLIF_UART){
        init_fl_uart();       // Initialize UART h.w.(for Flash device control)
        UARTE0 = true;        // enable UART h.w.
        SRIF0 = false;        // clear UART Rx IRQ flag
        SRMK0 = false;        // enable UART Rx INT.
  }
  else{
        init_fl_csi();        // Initialize CSI h.w.
  }
  for (n = 0; n < pulse; n++){// pulse output

        pFL_FLMD0 = low;
```

```
        fl_wait(tPW);
        pFL_FLMD0 = hi;
        fl_wait(tPW);
  }

  while(!check_flto())       // timeout tRC ?
        ;                    // no

  // start RESET command proc.
}
```

## 2.5 Selecting Serial Communication Mode

The communication mode is determined by inputting a pulse to the FLMD0 pin in the V850ES/Hx2 after reset release.

The high- and low-levels of the FLMD0 pulse are $V_{DD}$ and GND, respectively.

The following table shows the relationship between the number of FLMD0 pulses (pulse counts) and communication modes that can be selected with the V850ES/Hx2.

**Table 2-3. Relationship Between FLMD0 Pulse Counts and Communication Modes**

| Communication Mode | FLMD0 Pulse Counts | Port Used for Communication |
|---|---|---|
| UART (UART0) | 0 | TXDA0 (P30), RXDA0 (P31) |
| 3-wire serial I/O (CSIB0) | 8 | SOB0 (P41), SIB0 (P40), $\overline{\text{SCKB0}}$ (P42) |
| 3-wire serial I/O with handshake supported (CSIB0 + HS) | 11 | SOB0 (P41), SIB0 (P40), $\overline{\text{SCKB0}}$ (P42), HS (PCM0) |
| Setting prohibited | Others | – |

## 2.6 UART Communication Mode

The RxD and TxD pins are used for UART communication. The communication conditions are as shown below.

**Table 2-4. UART Communication Conditions**

| Item | Description |
|---|---|
| Baud rate | Selectable from 9,600, 19,200, 31,250, 38,400, 76,800, and 153,600 bps (default: 9,600 bps) |
| Parity bit | None |
| Data length | 8 bits (LSB first) |
| Stop bit | 1 bit |

The programmer always operates as the master device during CSI communication, so the programmer must check whether the processing by the V850ES/Hx2, such as writing or erasing, is normally completed. On the other hand, the status of the master and slave is occasionally exchanged during UART communication, so communication at the optimum timing is possible without assigning one pin like CSI + HS communication.

**Caution Set the same baud rate to the master and slave devices when performing UART communication.**

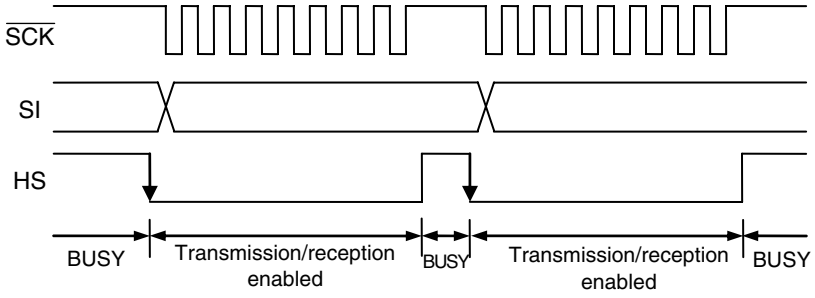
## 2.7 3-Wire Serial I/O Communication Mode with Handshake Supported (CSI + HS)

In the CSI + HS communication mode, the timing for communication of commands or data is optimized. In addition to the SI, SO and $\overline{SCK}$ pins, the HS (handshake) pin is used for implementing effective communication.

The level of the HS pin signal falls (low level) when the V850ES/Hx2 is ready for transmitting or receiving data. The programmer must check the falling edge of the HS pin signal (low level) before starting transmission/reception of commands or data to the V850ES/Hx2.

The communication data format is MSB-first, in 8-bit units. Keep the clock frequency 2.5 MHz or lower.

**Figure 2-6. Timing Chart of CSI + HS Communication**




## 2.8 3-Wire Serial I/O Communication Mode (CSI)

The $\overline{SCK}$, SO and SI pins are used for CSI communication. The programmer always operates as the master device, so communication may not be performed normally if data is transmitted via the SCK pin while the V850ES/Hx2 is not ready for transmission/reception.

The communication data format is MSB-first, in 8-bit units. Keep the clock frequency 2.5 MHz or lower.

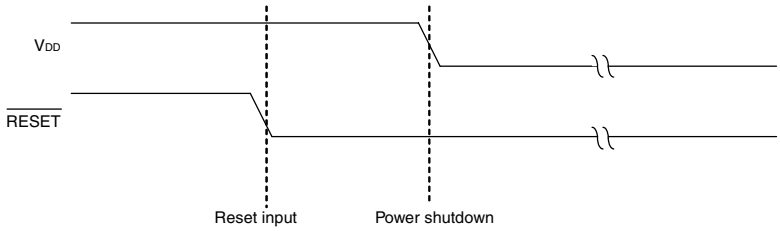
## 2.9 Shutting Down Target Power Supply

After each command execution is completed, shut down the power supply to the target after setting the $\overline{RESET}$ pin to low level, as shown below.

Set other pins to Hi-Z when shutting down the power supply to the target.

**Caution Shutting down the power supply and inputting a reset during command processing are prohibited.**

**Figure 2-7. Timing for Terminating Flash Memory Programming Mode**



## 2.10 Manipulation of Flash Memory

The flash memory incorporated in the V850ES/Hx2 has functions to manipulate the flash memory, as listed in Table 2-5. The programmer transmits commands to control these functions to the V850ES/Hx2, and checks the response status sent from the V850ES/Hx2, to manipulate the flash memory.

**Table 2-5. List of Flash Memory Manipulating Functions**

| Classification | Function Name | Description |
|---|---|---|
| Erase | Chip erase | Erases the entire flash memory area. Clears the security flag. |
| | Block erase | Erases a specified block in the flash memory. |
| Write | Write | Writes data to a specified area in the flash memory. |
| Verify | Verify | Compares data acquired from a specified address in the flash memory with data transmitted from the programmer, on the V850ES/Hx2 side. |
| Blank check | Block blank check | Checks the erase status of a specified area in the flash memory. |
| Information acquisition | Silicon signature acquisition | Acquires writing protocol information. |
| | Version acquisition | Acquires version information of the V850ES/Hx2 and firmware. |
| | Status acquisition | Acquires the current operating status. |
| | Checksum acquisition | Acquires checksum data of a specified area. |
| Security | Security setting | Sets security information. |
| Other | Reset | Detects synchronization in communication. |

## 2.11 Command List

The commands used by the programmer and their functions are listed below.

**Table 2-6. List of Commands Transmitted from Programmer to V850ES/Hx2**

| Command Number | Command Name | Function |
|---|---|---|
| 70H | Status | Acquires the current operating status (status data). |
| 00H | Reset | Detects synchronization in communication. |
| 90H | Oscillating Frequency Set | Specifies the oscillation frequency of the V850ES/Hx2. |
| 9AH | Baud Rate Set | Sets baud rate when UART communication mode is selected. |
| 20H | Chip Erase | Erases the entire flash memory area. |
| 22H | Block Erase | Erases a specified area in the flash memory. |
| 40H | Programming | Writes data to a specified area in the flash memory. |
| 13H | Verify | Compares the contents in a specified area in the flash memory with data transmitted from the programmer. |
| 32H | Block Blank Check | Checks the erase status of a specified block in the flash memory. |
| C0H | Silicon Signature | Acquires V850ES/Hx2 information (part number, flash memory configuration, etc.). |
| C5H | Version Get | Acquires version information of the V850ES/Hx2 and firmware. |
| B0H | Checksum | Acquires checksum data of a specified area. |
| A0H | Security Set | Sets security information. |
| 50H | Read | Reads data of a specified area in the flash memory. |

## 2.12 Status List

The following table lists the status codes the programmer receives from the V850ES/Hx2.

**Table 2-7. Status Code List**

| Status Code | Status | Description |
|---|---|---|
| 04H | Command number error | Error returned if a command not supported or invalid frame is received |
| 05H | Parameter error | Error returned if command information (parameter) is invalid |
| 06H | Normal acknowledgment (ACK) | Normal acknowledgment |
| 07H | Checksum error | Error returned if data in a frame transmitted from the programmer is abnormal |
| 08H | WWV1 error | Write error |
| 0BH | EWV1 error | Erase error |
| 0CH | EWV2 error | Erase error |
| 0DH | EWV3 error | Erase error |
| 0EH | Verify error | A verify error has occurred for the data of the frame transmitted from the programmer |
| 0FH | Verify error | A verify error has occurred for the data of the frame transmitted from the programmer |
| 10H | Protect error | Error returned if an attempt is made to execute processing that is prohibited by the Security Set command |
| 11H | EWV4 error | Internal verify error/blank error |
| 13H | Compaction search error | Erase error |
| 15H | Negative acknowledgment (NACK) | Negative acknowledgment |
| 16H | Sequencer error | Error returned if an error occurs in flash control macro |
| FFH | Processing in progress (BUSY) | Busy response[Note] |

**Note** During CSI communication, 1-byte “FFH” may be transmitted, as well as “FFH” as the data frame format.

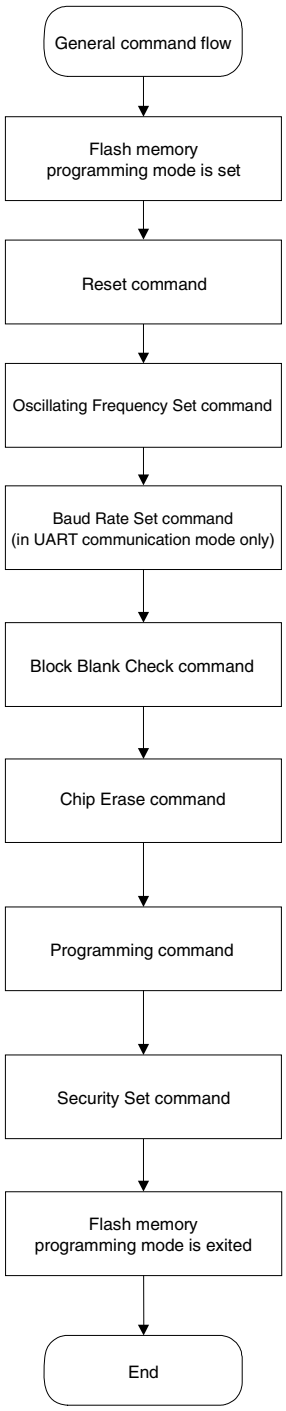
Reception of a checksum error or NACK is treated as an immediate abnormal end in this manual. When a dedicated programmer is developed, however, the processing may be retried without problem from the wait immediately before transmission of the command that results a checksum error or NACK or after BUSY status check via the HS pin. In this event, limiting the retry count is recommended for preventing infinite repetition of the retry operation.

Although not listed in the above table, if a time-out error (BUSY time-out, HS pin time-out, or time-out in data frame reception during UART communication) occurs, it is recommended to shutdown the power supply to the V850ES/Hx2 (refer to **2.9 Shutting Down Target Power Supply**) and then connect the power supply again.

# CHAPTER 3 BASIC PROGRAMMER OPERATION

Figure 3-1 illustrates the general command execution flow when flash memory rewriting is performed with the programmer.

**Figure 3-1. General Command Execution Flow at Flash Memory Rewriting**




**Note** It is recommended to perform security settings to disable read as a default in programmer specification.

**Remark** The Verify command and Checksum command can also be supported.

# CHAPTER 4 COMMAND/DATA FRAME FORMAT

The programmer uses the command frame to transmit commands to the V850ES/Hx2. The V850ES/Hx2 uses the data frame to transmit write data or verify data to the programmer. A header, footer, data length information, and checksum are appended to each frame to enhance the reliability of the transferred data.

The following shows the format of a command frame and data frame.

**Figure 4-1. Command Frame Format**

| SOH<br>(1 byte) | LEN<br>(1 byte) | COM<br>(1 byte) | Command information (variable length)<br>(Max. 255 bytes) | SUM<br>(1 byte) | ETX<br>(1 byte) |
|---|---|---|---|---|---|

**Figure 4-2. Data Frame Format**

| STX<br>(1 byte) | LEN<br>(1 byte) | Data (variable length)<br>(Max. 256 bytes) | SUM<br>(1 byte) | ETX or ETB<br>(1 byte) |
|---|---|---|---|---|

**Table 4-1. Description of Symbols in Each Frame**

| Symbol | Value | Description |
|---|---|---|
| SOH | 01H | Command frame header |
| STX | 02H | Data frame header |
| LEN | – | Data length information (00H indicates 256).<br>Command frame: COM + command information length<br>Data frame: Data field length |
| COM | – | Command number |
| SUM | – | Checksum data for a frame<br>Obtained by sequentially subtracting all of calculation target data from the initial value (00H) in 1-byte units (borrow is ignored). The calculation targets are as follows.<br>Command frame: LEN + COM + all of command information<br>Data frame: LEN + all of data |
| ETB | 17H | Footer of data frame other than the last frame |
| ETX | 03H | Command frame footer, or footer of last data frame |

The following shows examples of calculating the checksum (SUM) for a frame.

**[Command frame]**

No command information is included in the following example of a Status command frame, so LEN and COM are targets of checksum calculation.

| SOH | LEN | COM | SUM | ETX |
|---|---|---|---|---|
| 01H | 01H | 70H | Checksum | 03H |
| | Checksum calculation targets | | | |

For this command frame, checksum data is obtained as follows.

00H (initial value) – 01H (LEN) – 70H (COM) = 8FH (Borrow ignored. Lower 8 bits only.)

The command frame finally transmitted is as follows.

| SOH | LEN | COM | SUM | ETX |
|---|---|---|---|---|
| 01H | 01H | 70H | 8FH | 03H |

**[Data frame]**

To transmit a data frame as shown below, LEN and D1 to D4 are targets of checksum calculation.

| STX | LEN | D1 | D2 | D3 | D4 | SUM | ETX |
|---|---|---|---|---|---|---|---|
| 02H | 04H | FFH | 80H | 40H | 22H | Checksum | 03H |
| | checksum calculation targets | | | | | | |

For this data frame, checksum data is obtained as follows.

00H (initial value) – 04H (LEN) – FFH (D1) – 80H (D2) – 40H (D3) – 22H (D4)
= 1BH (Borrow ignored. Lower 8 bits only.)

The data frame finally transmitted is as follows.

| STX | LEN | D1 | D2 | D3 | D4 | SUM | ETX |
|---|---|---|---|---|---|---|---|
| 02H | 04H | FFH | 80H | 40H | 22H | 1BH | 03H |

When a data frame is received, the checksum data is calculated in the same manner, and the obtained value is used to detect a checksum error by judging whether the value is the same as that stored in the SUM field of the receive data. When a data frame as shown below is received, for example, a checksum error is detected.

| STX | LEN | D1 | D2 | D3 | D4 | SUM | ETX |
|---|---|---|---|---|---|---|---|
| 02H | 04H | FFH | 80H | 40H | 22H | 1AH | 03H |

↑ Should be 1BH, if normal

## 4.1 Command Frame Transmission Processing

Read the following chapters for details on flowcharts of command processing to transmit command frames, for each communication mode.

- For the UART communication mode, read **6.1 Flowchart of Command Frame Transmission Processing**.
- For the 3-wire serial I/O communication mode with handshake supported (CSI + HS), read **7.1 Flowchart of Command Frame Transmission Processing**.
- For the 3-wire serial I/O communication mode (CSI), read **8.1 Flowchart of Command Frame Transmission Processing**.

## 4.2 Data Frame Transmission Processing

The write data frame (user program), verify data frame (user program), and security data frame (security flag) are transmitted as a data frame.

Read the following chapters for details on flowcharts of command processing to transmit data frames, for each communication mode.

- For the UART communication mode, read **6.2 Flowchart of Data Frame Transmission Processing**.
- For the 3-wire serial I/O communication mode with handshake supported (CSI + HS), read **7.2 Flowchart of Data Frame Transmission Processing**.
- For the 3-wire serial I/O communication mode (CSI), read **8.2 Flowchart of Data Frame Transmission Processing**.

## 4.3 Data Frame Reception Processing

The status frame, silicon signature data frame, version data frame, and checksum data frame are received as a data frame.

Read the following chapters for details on flowcharts of command processing to receive data frames, for each communication mode.

- For the UART communication mode, read **6.3 Flowchart of Data Frame Reception Processing**.
- For the 3-wire serial I/O communication mode with handshake supported (CSI + HS), read **7.3 Flowchart of Data Frame Reception Processing**.
- For the 3-wire serial I/O communication mode (CSI), read **8.3 Flowchart of Data Frame Reception Processing**.

# CHAPTER 5 DESCRIPTION OF COMMAND PROCESSING

## 5.1 Status Command

### 5.1.1 Description

This command is used to check the operation status of the V850ES/Hx2 after issuance of each command such as write or erase.

After the Status command is issued, if the Status command frame cannot be received normally in the V850ES/Hx2 due to problems based on communication or the like, the status setting will not performed in the V850ES/Hx2. As a result, a busy response (FFH), not the status frame, may be received. In such a case, retry the Status command.

### 5.1.2 Command frame and status frame

Figure 5-1 shows the format of a command frame for the Status command, and Figure 5-2 shows the status frame for the command.

**Figure 5-1. Status Command Frame (from Programmer to V850ES/Hx2)**

| SOH | LEN | COM | SUM | ETX |
|---|---|---|---|---|
| 01H | 01H | 70H (Status) | Checksum | 03H |

**Figure 5-2. Status Frame for Status Command (from V850ES/Hx2 to Programmer)**

| STX | LEN | Data | | | SUM | ETX |
|---|---|---|---|---|---|---|
| 02H | n | ST1 | … | STn | Checksum | 03H |

**Remarks 1.** ST1 to STn: Status #1 to Status #n

**2.** The length of a status frame varies according to each command (such as write or erase) to be transmitted to the V850ES/Hx2.

Read the following chapters for details on flowcharts of processing sequences between the programmer and the V850ES/Hx2, flowcharts of command processing, and sample programs for each communication mode.

- The Status command is not used in the UART communication mode.
- For the 3-wire serial I/O communication mode with handshake supported (CSI + HS), read **7.4 Status Command**.
- For the 3-wire serial I/O communication mode (CSI), read **8.4 Status Command**.

**Caution After each command such as write or erase is transmitted in UART communication, the V850ES/Hx2 automatically returns the status frame within a specified time. The Status command is therefore not used.**

**If the Status command is transmitted in UART communication, the Command Number Error is returned.**

## 5.2 Reset Command

### 5.2.1 Description

This command is used to check the establishment of communication between the programmer and the V850ES/Hx2 after the communication mode is set.

When UART is selected as the mode for communication with the V850ES/Hx2, the same baud rate must be set in the programmer and V850ES/Hx2. However, the V850ES/Hx2 cannot detect its own operating frequency so the baud rate cannot be set. It makes detection of the operating frequency in the V850ES/Hx2 possible by sending “00H” twice at 9,600 bps from the programmer, measuring the low-level width of “00H”, and then calculating the average of two sent signals. The baud rate can consequently be set, which enables synchronous detection in communication.

### 5.2.2 Command frame and status frame

Figure 5-3 shows the format of a command frame for the Reset command, and Figure 5-4 shows the status frame for the command.

**Figure 5-3. Reset Command Frame (from Programmer to V850ES/Hx2)**

| SOH | LEN | COM | SUM | ETX |
|---|---|---|---|---|
| 01H | 01H | 00H (Reset) | Checksum | 03H |

**Figure 5-4. Status Frame for Reset Command (from V850ES/Hx2 to Programmer)**

| STX | LEN | Data | SUM | ETX |
|---|---|---|---|---|
| 02H | 1 | ST1 | Checksum | 03H |

**Remark** ST1: Synchronization detection result

Read the following chapters for details on flowcharts of processing sequences between the programmer and the V850ES/Hx2, flowcharts of command processing, and sample programs for each communication mode.

- For the UART communication mode, read **6.4 Reset Command**.
- For the 3-wire serial I/O communication mode with handshake supported (CSI + HS), read **7.5 Reset Command**.
- For the 3-wire serial I/O communication mode (CSI), read **8.5 Reset Command**.

## 5.3 Baud Rate Set Command

### 5.3.1 Description

This command is used to change the baud rate for UART (default value: 9,600 bps).

After the Baud Rate Set command is executed, the Reset command must be executed to confirm synchronization at the new baud rate.

The Baud Rate Set command is valid only in the UART communication mode. Data for setting the baud rate is represented as a 1-byte numeric value.

The V850ES/Hx2 ignores the Baud Rate Set command if it is transmitted in modes other than the UART communication mode.

### 5.3.2 Command frame and status frame

Figure 5-5 shows the format of a command frame for the Baud Rate Set command, and Figure 5-6 shows the status frame for the command.

**Figure 5-5. Baud Rate Set Command Frame (from Programmer to V850ES/Hx2)**

| SOH | LEN | COM | Command Information | SUM | ETX |
|---|---|---|---|---|---|
| 01H | 02H | 9AH (Baud Rate Set) | D01 | Checksum | 03H |

**Remark** D01: Baud rate selection value

| D01 Value | 03H | 04H | 05H | 06H | 07H | 08H |
|---|---|---|---|---|---|---|
| Baud rate (bps) | 9600 | 19200 | 31250 | 38400 | 76800 | 153600 |

**Figure 5-6. Status Frame for Baud Rate Set Command (from V850ES/Hx2 to Programmer)**

| STX | LEN | Data | SUM | ETX |
|---|---|---|---|---|
| 02H | 01H | ST1 | Checksum | 03H |

**Remark** ST1: Synchronization detection result

Read the following chapters for details on flowcharts of processing sequences between the programmer and the V850ES/Hx2, flowcharts of command processing, and sample programs for each communication mode.

- For the UART communication mode, read **6.5 Baud Rate Set Command**.
- The Baud Rate Set command is not used in the 3-wire serial I/O communication mode with handshake supported (CSI + HS).
- The Baud Rate Set command is not used in 3-wire serial I/O communication mode (CSI).

## 5.4 Oscillating Frequency Set Command

### 5.4.1 Description

This command is used to set oscillation frequency data in the V850ES/Hx2.

Specify the frequency of the clock that is actually input to the X1 pin of the V850ES/Hx2.

The V850ES/Hx2 automatically sets the multiply rate of the CPU operation clock, based on the clock frequency specified with this command. Therefore, note that the reference clock for wait calculation varies before and after execution of this command.

### 5.4.2 Command frame and status frame

Figure 5-7 shows the format of a command frame for the Oscillating Frequency Set command, and Figure 5-8 shows the status frame for the command.

**Figure 5-7. Oscillating Frequency Set Command Frame (from Programmer to V850ES/Hx2)**

| SOH | LEN | COM | Command Information | | | | SUM | ETX |
|---|---|---|---|---|---|---|---|---|
| 01H | 05H | 90H (Oscillating Frequency Set) | D01 | D02 | D03 | D04 | Checksum | 03H |

**Remark** D01 to D04: Oscillation frequency = $(D01 \times 0.1 + D02 \times 0.01 + D03 \times 0.001) \times 10^{D04}$ (Unit: kHz)
Settings can be made from 10 kHz to 100 MHz, but set the value according to the specifications of each device when actually transmitting the command. D01 to D03 hold unpacked BCDs, and D04 holds a signed integer.

Setting example: To set 6 MHz
D01 = 06H
D02 = 00H
D03 = 00H
D04 = 04H
Oscillation frequency = $6 \times 0.1 \times 10^4$ = 6,000 kHz = 6 MHz

Setting example: To set 10 MHz
D01 = 01H
D02 = 00H
D03 = 00H
D04 = 05H
Oscillation frequency = $1 \times 0.1 \times 10^5$ = 10,000 kHz = 10 MHz

**Figure 5-8. Status Frame for Oscillating Frequency Set Command (from V850ES/Hx2 to Programmer)**

| STX | LEN | Data | SUM | ETX |
|---|---|---|---|---|
| 02H | 01H | ST1 | Checksum | 03H |

**Remark** ST1: Oscillation frequency setting result

Read the following chapters for details on flowcharts of processing sequences between the programmer and the V850ES/Hx2, flowcharts of command processing, and sample programs for each communication mode.

- For the UART communication mode, read **6.6 Oscillating Frequency Set Command**.
- For the 3-wire serial I/O communication mode with handshake supported (CSI + HS), read **7.6 Oscillating Frequency Set Command**.
- For the 3-wire serial I/O communication mode (CSI), read **8.6 Oscillating Frequency Set Command**.

## 5.5 Chip Erase Command

### 5.5.1 Description

This command is used to erase the entire contents of the flash memory. In addition, all of the information that is set by security setting processing can be initialized by chip erase processing, as long as erasure is not prohibited by the security setting (see **5.13 Security Set Command**).

### 5.5.2 Command frame and status frame

Figure 5-9 shows the format of a command frame for the Chip Erase command, and Figure 5-10 shows the status frame for the command.

**Figure 5-9. Chip Erase Command Frame (from Programmer to V850ES/Hx2)**

| SOH | LEN | COM | SUM | ETX |
|---|---|---|---|---|
| 01H | 01H | 20H<br>(Chip Erase) | Checksum | 03H |

**Figure 5-10. Status Frame for Chip Erase Command (from V850ES/Hx2 to Programmer)**

| STX | LEN | Data | SUM | ETX |
|---|---|---|---|---|
| 02H | 01H | ST1 | Checksum | 03H |

**Remark** ST1: Chip erase result

Read the following chapters for details on flowcharts of processing sequences between the programmer and the V850ES/Hx2, flowcharts of command processing, and sample programs for each communication mode.

- For the UART communication mode, read **6.7 Chip Erase Command**.
- For the 3-wire serial I/O communication mode with handshake supported (CSI + HS), read **7.7 Chip Erase Command**.
- For the 3-wire serial I/O communication mode (CSI), read **8.7 Chip Erase Command**.

## 5.6 Block Erase Command

### 5.6.1 Description

This command is used to erase the contents of blocks with the specified number in the flash memory, as long as erasure is not prohibited by the security setting (see **5.13 Security Set Command**).

### 5.6.2 Command frame and status frame

Figure 5-11 shows the format of a command frame for the Block Erase command, and Figure 5-12 shows the status frame for the command.

**Figure 5-11. Block Erase Command Frame (from Programmer to V850ES/Hx2)**

| SOH | LEN | COM | Command Information | SUM | ETX |
|---|---|---|---|---|---|
| 01H | 02H | 22H (Block Erase) | BLK | Checksum | 03H |

**Remark** BLK: Number of block to be erased

**Figure 5-12. Status Frame for Block Erase Command (from V850ES/Hx2 to Programmer)**

| STX | LEN | Data | SUM | ETX |
|---|---|---|---|---|
| 02H | 01H | ST1 | Checksum | 03H |

**Remark** ST1: Block erase result

Read the following chapters for details on flowcharts of processing sequences between the programmer and the V850ES/Hx2, flowcharts of command processing, and sample programs for each communication mode.

- For the UART communication mode, read **6.8 Block Erase Command**.
- For the 3-wire serial I/O communication mode with handshake supported (CSI + HS), read **7.8 Block Erase Command**.
- For the 3-wire serial I/O communication mode (CSI), read **8.8 Block Erase Command**.

## 5.7 Programming Command

### 5.7.1 Description

This command is used to transmit data by the number of written bytes after the write start address and the write end address are transmitted. This command then writes the user program to the flash memory and verifies it internally.

The write start/end address can be set only in the block start/end address units.

If both of the status frames (ST1 and ST2) after the last data transmission indicate ACK, the V850ES/Hx2 firmware automatically executes internal verify. Therefore, the Status command for this internal verify must be transmitted.

### 5.7.2 Command frame and status frame

Figure 5-13 shows the format of a command frame for the Programming command, and Figure 5-14 shows the status frame for the command.

**Figure 5-13. Programming Command Frame (from Programmer to V850ES/Hx2)**

| SOH | LEN | COM | Command Information | | | | | | SUM | ETX |
|---|---|---|---|---|---|---|---|---|---|---|
| 01H | 07H | 40H (Programming) | SAH | SAM | SAL | EAH | EAM | EAL | Checksum | 03H |

**Remark** SAH, SAM, SAL: Write start addresses
EAH, EAM, EAL: Write end addresses

**Figure 5-14. Status Frame for Programming Command (from V850ES/Hx2 to Programmer)**

| STX | LEN | Data | SUM | ETX |
|---|---|---|---|---|
| 02H | 01H | ST1 (a) | Checksum | 03H |

**Remark** ST1 (a): Command reception result

### 5.7.3 Data frame and status frame

Figure 5-15 shows the format of a frame that includes data to be written, and Figure 5-16 shows the status frame for the data.

**Figure 5-15. Data Frame to Be Written (from Programmer to V850ES/Hx2)**

| STX | LEN | Data | SUM | ETX/ETB |
|---|---|---|---|---|
| 02H | 00H to FFH (00H = 256) | Write Data | Checksum | 03H/17H |

**Remark** Write Data: User program to be written

**Figure 5-16. Status Frame for Data Frame (from V850ES/Hx2 to Programmer)**

| STX | LEN | Data | | SUM | ETX |
|---|---|---|---|---|---|
| 02H | 02H | ST1 (b) | ST2 (b) | Checksum | 03H |

**Remark** ST1 (b): Data reception check result
ST2 (b): Write result

### 5.7.4 Completion of transferring all data and status frame

Figure 5-17 shows the status frame after transfer of all data is completed.

**Figure 5-17. Status Frame After Completion of Transferring All Data (from V850ES/Hx2 to Programmer)**

| STX | LEN | Data | SUM | ETX |
|---|---|---|---|---|
| 02H | 01H | ST1 (c) | Checksum | 03H |

**Remark** ST1 (c): Internal verify result

Read the following chapters for details on flowcharts of processing sequences between the programmer and the V850ES/Hx2, flowcharts of command processing, and sample programs for each communication mode.

- For the UART communication mode, read **6.9 Programming Command**.
- For the 3-wire serial I/O communication mode with handshake supported (CSI + HS), read **7.9 Programming Command**.
- For the 3-wire serial I/O communication mode (CSI), read **8.9 Programming Command**.

## 5.8 Verify Command

### 5.8.1 Description

This command is used to compare the data transmitted from the programmer with the data read from the V850ES/Hx2 (read level) in the specified address range, and check whether they match.

The verify start/end address can be set only in the block start/end address units.

### 5.8.2 Command frame and status frame

Figure 5-18 shows the format of a command frame for the Verify command, and Figure 5-19 shows the status frame for the command.

**Figure 5-18. Verify Command Frame (from Programmer to V850ES/Hx2)**

| SOH | LEN | COM | Command Information | | | | | | SUM | ETX |
|---|---|---|---|---|---|---|---|---|---|---|
| 01H | 07H | 13H (Verify) | SAH | SAM | SAL | EAH | EAM | EAL | Checksum | 03H |

**Remark** SAH, SAM, SAL: Verify start addresses
EAH, EAM, EAL: Verify end addresses

**Figure 5-19. Status Frame for Verify Command (from V850ES/Hx2 to Programmer)**

| STX | LEN | Data | SUM | ETX |
|---|---|---|---|---|
| 02H | 01H | ST1 (a) | Checksum | 03H |

**Remark** ST1 (a): Command reception result

### 5.8.3 Data frame and status frame

Figure 5-20 shows the format of a frame that includes data to be verified, and Figure 5-21 shows the status frame for the data.

**Figure 5-20. Data Frame of Data to Be Verified (from Programmer to V850ES/Hx2)**

| STX | LEN | Data | SUM | ETX/ETB |
|---|---|---|---|---|
| 02H | 00H to FFH (00H = 256) | Verify data | Checksum | 03H/17H |

**Remark** Verify Data: User program to be verified

**Figure 5-21. Status Frame for Data Frame (from V850ES/Hx2 to Programmer)**

| STX | LEN | Data | | SUM | ETX |
|---|---|---|---|---|---|
| 02H | 02H | ST1 (b) | ST2 (b) | Checksum | 03H |

**Remark** ST1 (b): Data reception check result
ST2 (b): Verify result[Note]

**Note** Even if a verify error occurs in the specified address range, ACK is always returned as the verify result. The status of all verify errors are reflected in the verify result for the last data. Therefore, the occurrence of verify errors can be checked only when all the verify processing for the specified address range is completed.

Read the following chapters for details on flowcharts of processing sequences between the programmer and the V850ES/Hx2, flowcharts of command processing, and sample programs for each communication mode.

- For the UART communication mode, read **6.10 Verify Command**.
- For the 3-wire serial I/O communication mode with handshake supported (CSI + HS), read **7.10 Verify Command**.
- For the 3-wire serial I/O communication mode (CSI), read **8.10 Verify Command**.

## 5.9 Block Blank Check Command

### 5.9.1 Description

This command is used to check if a block in the flash memory, with a specified block number, is blank (erased state).

### 5.9.2 Command frame and status frame

Figure 5-22 shows the format of a command frame for the Block Blank Check command, and Figure 5-23 shows the status frame for the command.

**Figure 5-22. Block Blank Check Command Frame (from Programmer to V850ES/Hx2)**

| SOH | LEN | COM | Command Information | SUM | ETX |
|---|---|---|---|---|---|
| 01H | 02H | 32H (Block Blank Check) | BLK | Checksum | 03H |

**Remark** BLK: Number of block to be checked for blank

**Figure 5-23. Status Frame for Block Blank Check Command (from V850ES/Hx2 to Programmer)**

| STX | LEN | Data | SUM | ETX |
|---|---|---|---|---|
| 02H | 01H | ST1 | Checksum | 03H |

**Remark** ST1: Block blank check result

Read the following chapters for details on flowcharts of processing sequences between the programmer and the V850ES/Hx2, flowcharts of command processing, and sample programs for each communication mode.

- For the UART communication mode, read **6.11 Block Blank Check Command**.
- For the 3-wire serial I/O communication mode with handshake supported (CSI + HS), read **7.11 Block Blank Check Command**.
- For the 3-wire serial I/O communication mode (CSI), read **8.11 Block Blank Check Command**.

## 5.10 Silicon Signature Command

### 5.10.1 Description

This command is used to read the write protocol information (silicon signature) of the device.

If the programmer supports a programming protocol that is not supported in the V850ES/Hx2, for example, execute this command to select an appropriate protocol in accordance with the values of the second and third bytes.

### 5.10.2 Command frame and status frame

Figure 5-24 shows the format of a command frame for the Silicon Signature command, and Figure 5-25 shows the status frame for the command.

**Figure 5-24. Silicon Signature Command Frame (from Programmer to V850ES/Hx2)**

| SOH | LEN | COM | SUM | ETX |
|---|---|---|---|---|
| 01H | 01H | C0H (Silicon Signature) | Checksum | 03H |

**Figure 5-25. Status Frame for Silicon Signature Command (from V850ES/Hx2 to Programmer)**

| STX | LEN | Data | SUM | ETX |
|---|---|---|---|---|
| 02H | 01H | ST1 | Checksum | 03H |

**Remark** ST1: Command reception result

### 5.10.3 Silicon signature data frame

Figure 5-26 shows the format of a frame that includes silicon signature data.

**Figure 5-26. Silicon Signature Data Frame (from V850ES/Hx2 to Programmer)**

| STX | LEN | Data | | | | SUM | ETX |
|---|---|---|---|---|---|---|---|
| 02H | n | VEN | EXT | FNC | Invalid data | Checksum | 03H |

**Remarks 1.** n (LEN): Data length
VEN: Vendor code (NEC: 10H)
EXT: Extension code
FNC: Function information
INVALID DATA: Invalid data of 90 to 198-byte length.

**2.** For the vendor code (VEN), extension code (EXT) and function information (FNC), the highest bit is used as an odd parity. The following shows an example.

**Table 5-1. Example of Silicon Signature Data**

| Field | Contents | Length (Byte) | Example of Silicon Signature Data[Note] | Actual Value |
|---|---|---|---|---|
| VEN | Vendor code (NEC) | 1 | 10H (00010000B) | 10H |
| EXT | Extension code (fixed) | 1 | 4FH (01001111B) | 4FH |
| FNC | Function information (fixed) | 1 | 40H (01000000B) | 40H |

**Note** The shaded bit is an odd parity (the value to adjust the number of “1” in a byte).

Read the following chapters for details on flowcharts of processing sequences between the programmer and the V850ES/Hx2, flowcharts of command processing, and sample programs for each communication mode.

- For the UART communication mode, read **6.12 Silicon Signature Command**.
- For the 3-wire serial I/O communication mode with handshake supported (CSI + HS), read **7.12 Silicon Signature Command**.
- For the 3-wire serial I/O communication mode (CSI), read **8.12 Silicon Signature Command**.

## 5.11 Version Get Command

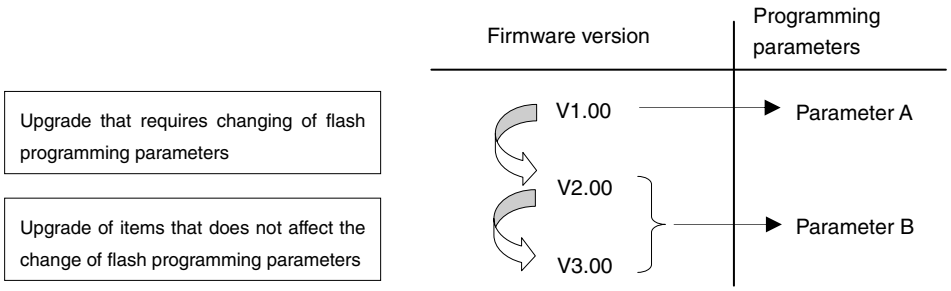
### 5.11.1 Description

This command is used to acquire information on the V850ES/Hx2 device version and firmware version.

Use this command when the programming parameters must be changed in accordance with the V850ES/Hx2 firmware version.

**Caution The firmware version may be updated during firmware update that does not affect the change of flash programming parameters (at this time, update of the firmware version is not reported).**

**Example** Firmware version and reprogramming parameters




### 5.11.2 Command frame and status frame

Figure 5-28 shows the format of a command frame for the Version Get command, and Figure 5-29 shows the status frame for the command.

**Figure 5-28. Version Get Command Frame (from Programmer to V850ES/Hx2)**

| SOH | LEN | COM | SUM | ETX |
|---|---|---|---|---|
| 01H | 01H | C5H (Version Get) | Checksum | 03H |

**Figure 5-29. Status Frame for Version Get Command (from V850ES/Hx2 to Programmer)**

| STX | LEN | Data | SUM | ETX |
|---|---|---|---|---|
| 02H | 01H | ST1 | Checksum | 03H |

**Remark** ST1: Command reception result

### 5.11.3 Version data frame

Figure 5-30 shows the data frame of version data.

**Figure 5-30. Version Data Frame (from V850ES/Hx2 to Programmer)**

| STX | LEN | Data | | | | | | SUM | ETX |
|---|---|---|---|---|---|---|---|---|---|
| 02H | 06H | DV1 | DV2 | DV3 | FV1 | FV2 | FV3 | Checksum | 03H |

**Remark** DV1: Integer of device version
DV2: First decimal place of device version
DV3: Second decimal place of device version
FV1: Integer of firmware version
FV2: First decimal place of firmware version
FV3: Second decimal place of firmware version

Read the following chapters for details on flowcharts of processing sequences between the programmer and the V850ES/Hx2, flowcharts of command processing, and sample programs for each communication mode.

- For the UART communication mode, read **6.13 Version Get Command**.
- For the 3-wire serial I/O communication mode with handshake supported (CSI + HS), read **7.13 Version Get Command**.
- For the 3-wire serial I/O communication mode (CSI), read **8.13 Version Get Command**.

## 5.12 Checksum Command

### 5.12.1 Description

This command is used to acquire the checksum data in the specified area.

For the checksum calculation start/end address, specify a fixed address in block units (2 KB) starting from the top of the flash memory.

Checksum data is obtained by sequentially subtracting data in the specified address range from the initial value (00H) in 1-byte units.

### 5.12.2 Command frame and status frame

Figure 5-31 shows the format of a command frame for the Checksum command, and Figure 5-32 shows the status frame for the command.

**Figure 5-31. Checksum Command Frame (from Programmer to V850ES/Hx2)**

| SOH | LEN | COM | Command Information | | | | | | SUM | ETX |
|---|---|---|---|---|---|---|---|---|---|---|
| 01H | 07H | B0H (Checksum) | SAH | SAM | SAL | EAH | EAM | EAL | Checksum | 03H |

**Remark** SAH, SAM, SAL: Checksum calculation start addresses
EAH, EAM, EAL: Checksum calculation end addresses

**Figure 5-32. Status Frame for Checksum Command (from V850ES/Hx2 to Programmer)**

| STX | LEN | Data | SUM | ETX |
|---|---|---|---|---|
| 02H | 01H | ST1 | Checksum | 03H |

**Remark** ST1: Command reception result

### 5.12.3 Checksum data frame

Figure 5-33 shows the format of a frame that includes checksum data.

**Figure 5-33. Checksum Data Frame (from V850ES/Hx2 to Programmer)**

| STX | LEN | Data | | SUM | ETX |
|---|---|---|---|---|---|
| 02H | 02H | CK1 | CK2 | Checksum | 03H |

**Remark** CK1: Higher 8 bits of checksum data
CK2: Lower 8 bits of checksum data

Read the following chapters for details on flowcharts of processing sequences between the programmer and the V850ES/Hx2, flowcharts of command processing, and sample programs for each communication mode.

- For the UART communication mode, read **6.14 Checksum Command**.
- For the 3-wire serial I/O communication mode with handshake supported (CSI + HS), read **7.14 Checksum Command**.
- For the 3-wire serial I/O communication mode (CSI), read **8.14 Checksum Command**.

## 5.13 Security Set Command

### 5.13.1 Description

This command is used to perform security settings (enable or disable of write, block erase, and chip erase). By performing these settings with this command, rewriting of the flash memory by an unauthorized party can be restricted.

**Caution Once the security setting is performed, additional setting to the security flags or changing of the setting from disable to enable will no longer be possible. If such settings are attempted, a Write error (1CH) will occur. To re-set the security flag, all the security flags must be initialized by executing the Chip Erase command (the Block Erase command cannot be used to initialize the security flags). If chip erase has been disabled, however, chip erase itself will be impossible and so the settings cannot be erased from the programmer. Re-confirmation of security setting execution is therefore recommended before disabling chip erase, due to this programmer specification.**

### 5.13.2 Command frame and status frame

Figure 5-34 shows the format of a command frame for the Security Set command, and Figure 5-35 shows the status frame for the command.

The Security Set command frame includes the block number field and page number field but these fields do not have any particular usage, so set these fields to 00H.

**Figure 5-34. Security Set Command Frame (from Programmer to V850ES/Hx2)**

| SOH | LEN | COM | Command Information | | SUM | ETX |
|---|---|---|---|---|---|---|
| 01H | 03H | A0H (Security Set) | 00H (fixed) | 00H (fixed) | Checksum | 03H |

**Remark** BLK, PAG: Fixed to 00H

**Figure 5-35. Status Frame for Security Set Command (from V850ES/Hx2 to Programmer)**

| STX | LEN | Data | SUM | ETX |
|---|---|---|---|---|
| 02H | 01H | ST1 (a) | Checksum | 03H |

**Remark** ST1 (a): Command reception result

### 5.13.3 Data frame and status frame

Figure 5-36 shows the format of a security data frame, and Figure 5-37 shows the status frame for the data.

**Figure 5-36 Security Data Frame (from Programmer to V850ES/Hx2)**

| STX | LEN | Data | | | | SUM | ETX |
|---|---|---|---|---|---|---|---|
| 02H | 04H | FLG | ADH | ADM | ADL | Checksum | 03H |

**Remark** FLG: Security flag
ADH: Reset vector handler address (bits 23 to 16)
ADM: Reset vector handler address (bits 15 to 8)
ADL: Reset vector handler address (bits 7 to 0)

**Figure 5-37. Status Frame for Security Data Writing (from V850ES/Hx2 to Programmer)**

| STX | LEN | Data | SUM | ETX |
|---|---|---|---|---|
| 02H | 01H | ST1 (b) | Checksum | 03H |

**Remark** ST1 (b): Security data write result

### 5.13.4 Internal verify check and status frame

Figure 5-38 shows the status frame for internal verify check.

**Figure 5-38. Status Frame for Internal Verify Check (from V850ES/Hx2 to Programmer)**

| STX | LEN | Data | SUM | ETX |
|---|---|---|---|---|
| 02H | 01H | ST1 (c) | Checksum | 03H |

**Remark** ST1 (c): Internal verify result

The following table shows the contents in the security flag field.

**Table 5-2. Contents of Security Flag Field**

| Item | Contents |
|---|---|
| Bit 7 | Fixed to “1” |
| Bit 6 | |
| Bit 5 | |
| Bit 4 | |
| Bit 3 | |
| Bit 2 | Programming disable flag (1: Enables programming, 0: Disable programming) |
| Bit 1 | Block erase disable flag (1: Enables block erase, 0: Disable block erase) |
| Bit 0 | Chip erase disable flag (1: Enables chip erase, 0: Disable chip erase) |

The following table shows the relationship between the security flag field settings and the enable/disable status of each operation.

**Table 5-3. Security Flag Field and Enable/Disable Status of Each Operation**

| Operating Mode | Flash Memory Programming Mode | | | Self-Programming Mode |
|---|---|---|---|---|
| Command / Security Setting Item | Command Operation After Security Setting<br>√: Execution possible, ×: Execution impossible | | | • All commands can be executed regardless of the security setting values<br>• Only retention of security setting values is possible |
| | Programming | Chip Erase | Block Erase | |
| Disable programming | × | √ | × | |
| Disable chip erase | √ | × | × | |
| Disable block erase | √ | √ | × | |

Read the following chapters for details on flowcharts of processing sequences between the programmer and the V850ES/Hx2, flowcharts of command processing, and sample programs for each communication mode.

- For the UART communication mode, read **6.15 Security Set Command**.
- For the 3-wire serial I/O communication mode with handshake supported (CSI + HS), read **7.15 Security Set Command**.
- For the 3-wire serial I/O communication mode (CSI), read **8.15 Security Set Command**.

## 5.14 Read Command

### 5.14.1 Description

This command is used to read data from the flash memory of the V850ES/Hx2.

The write start/end address can be set only in the block start/end address units.

### 5.14.2 Command frame and status frame

Figure 5-39 shows the format of a command frame for the Read command, and Figure 5-40 shows the status frame for the command.

**Figure 5-39. Read Command Frame (from Programmer to V850ES/Hx2)**

| SOH | LEN | COM | Command Information | | | | | | SUM | ETX |
|---|---|---|---|---|---|---|---|---|---|---|
| 01H | 07H | 50H (Read) | SAH | SAM | SAL | EAH | EAM | EAL | Checksum | 03H |

**Remark** SAH, SAM, SAL: Read start address (start address of the block)
EAH, EAM, EAL: Read end address (end address of the block)

**Figure 5-40. Status Frame for Read Command (from V850ES/Hx2 to Programmer)**

| STX | LEN | Data | SUM | ETX |
|---|---|---|---|---|
| 02H | 01H | ST1 (a) | Checksum | 03H |

**Remark** ST1 (a): Command reception result

### 5.14.3 Data frame and status frame

Figure 5-41 shows the format of a frame that includes data to be read, and Figure 5-42 shows the status frame for the data.

**Figure 5-41. Data Frame of Data to Be Read (from Programmer to V850ES/Hx2)**

| STX | LEN | Data | SUM | ETX/ETB |
|---|---|---|---|---|
| 02H | 00H to FFH (00H = 256) | Read Data | Checksum | 03H/17H |

**Remark** Read Data: Data read from V850ES/Hx2

**Figure 5-42. Status Frame for Read Data (from V850ES/Hx2 to Programmer)**

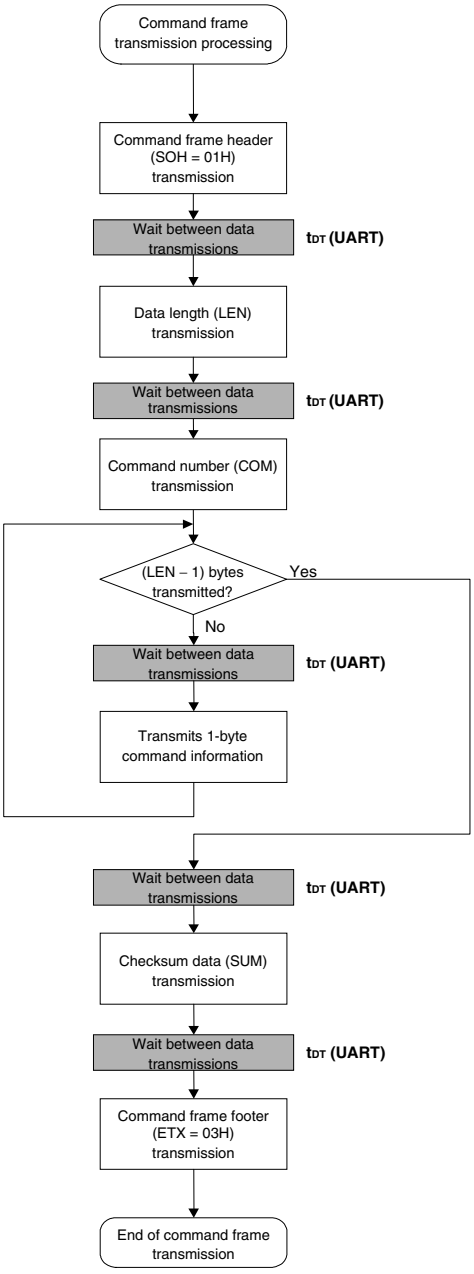
| STX | LEN | Data | SUM | ETX |
|---|---|---|---|---|
| 02H | 00H to FFH (00H = 256) | ST1 (b) | Checksum | 03H/17H |

**Remark** ST1 (b): ACK (06H) or NACK (15H) sent from the programmer for read data

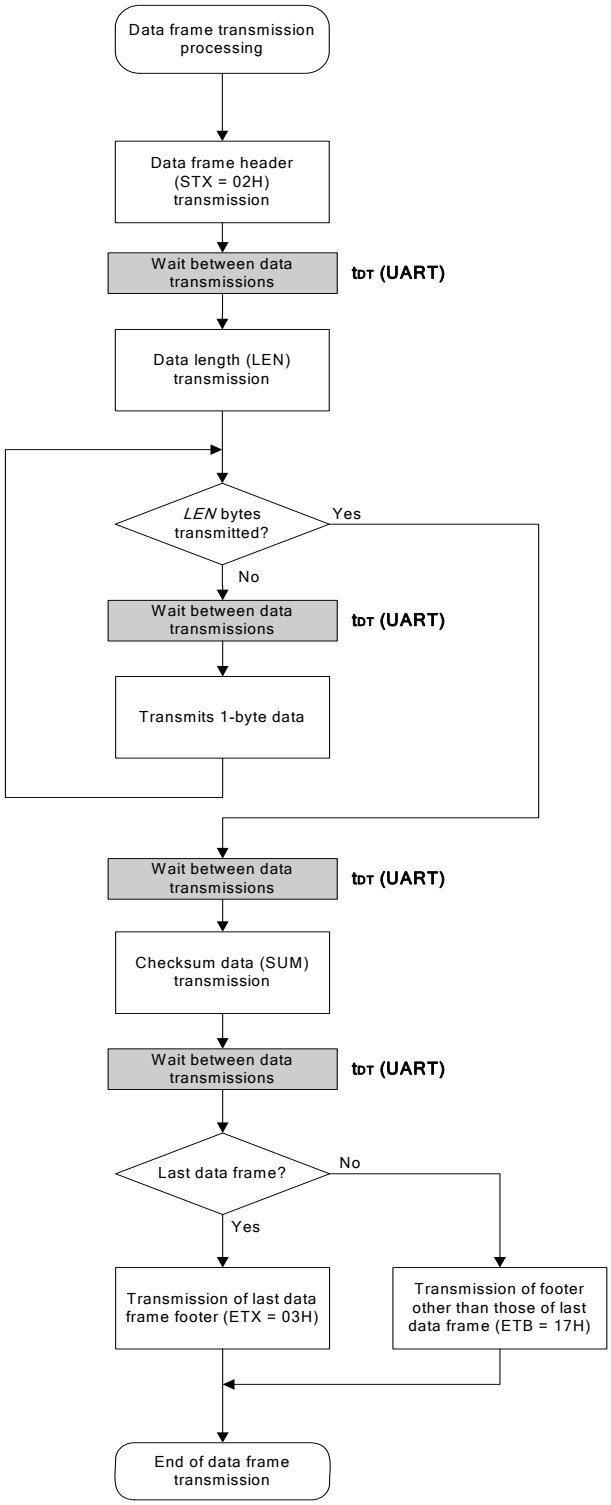
Read the following chapters for details on flowcharts of processing sequences between the programmer and the V850ES/Hx2, flowcharts of command processing, and sample programs for each communication mode.

- For the UART communication mode, read **6.16 Read Command**.
- For the 3-wire serial I/O communication mode with handshake supported (CSI + HS), read **7.16 Read Command**.
- For the 3-wire serial I/O communication mode (CSI), read **8.16 Read Command**.

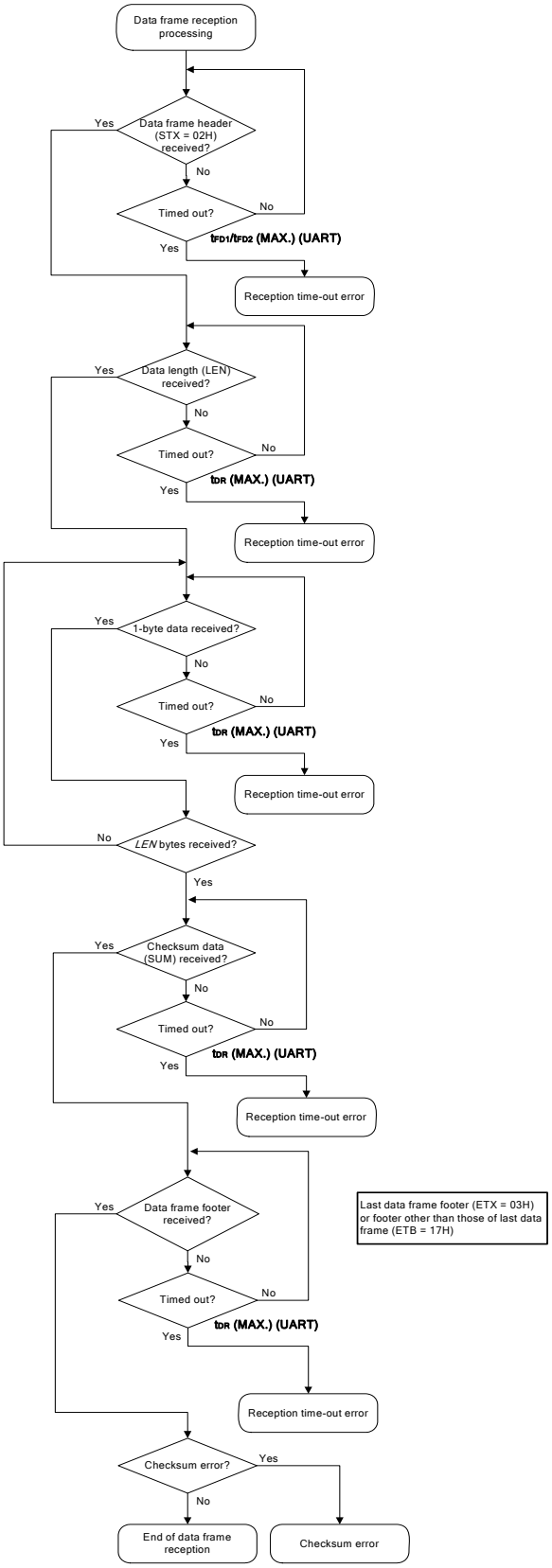
## 6.1 Command Frame Transmission Processing Flowchart



## 6.2 Data Frame Transmission Processing Flowchart



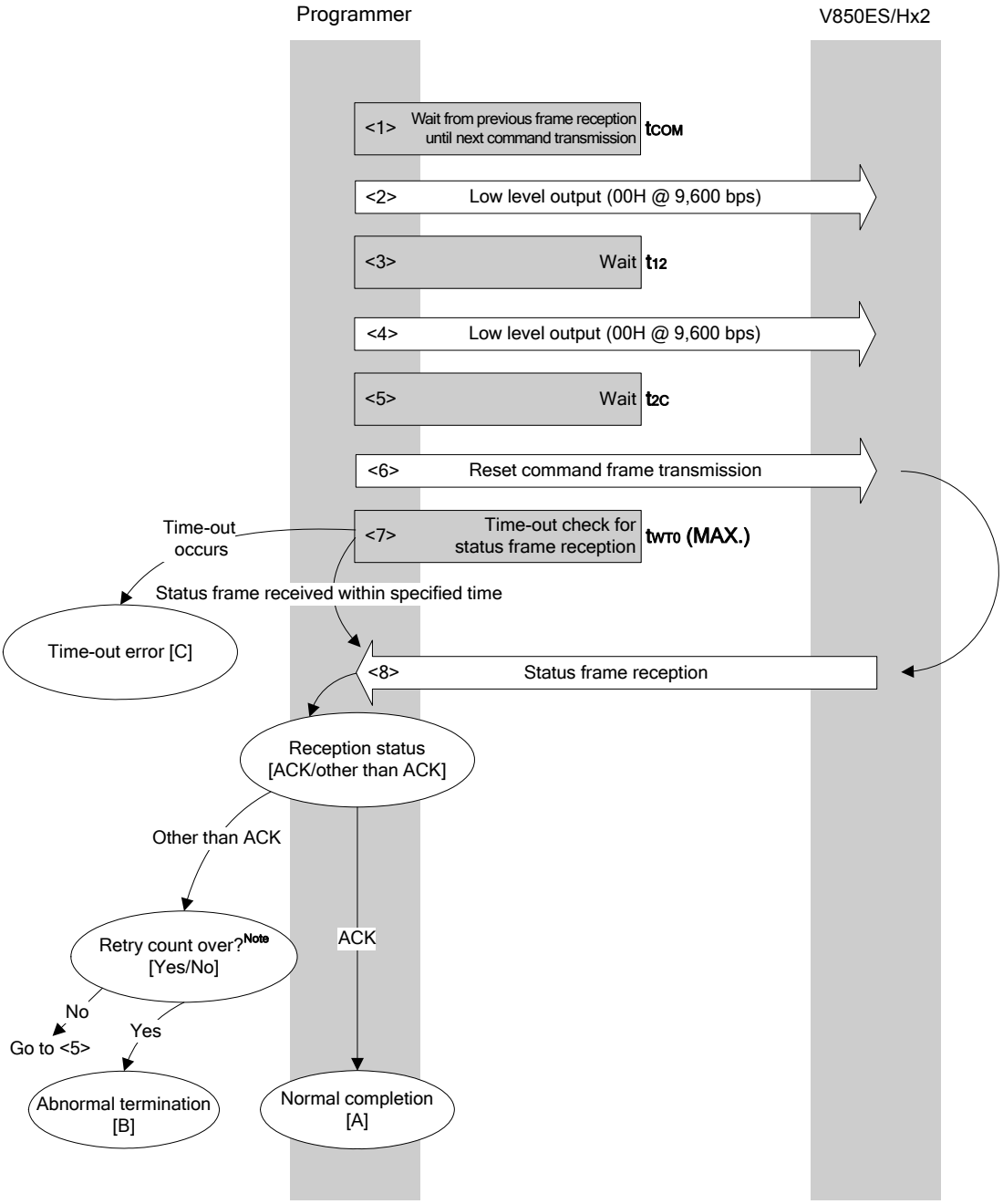
## 6.3 Data Frame Reception Processing Flowchart



## 6.4 Reset Command

### 6.4.1 Processing sequence chart

Reset command processing sequence




**Note** Do not exceed the retry count for the reset command transmission (up to 16 times).

### 6.4.2 Description of processing sequence

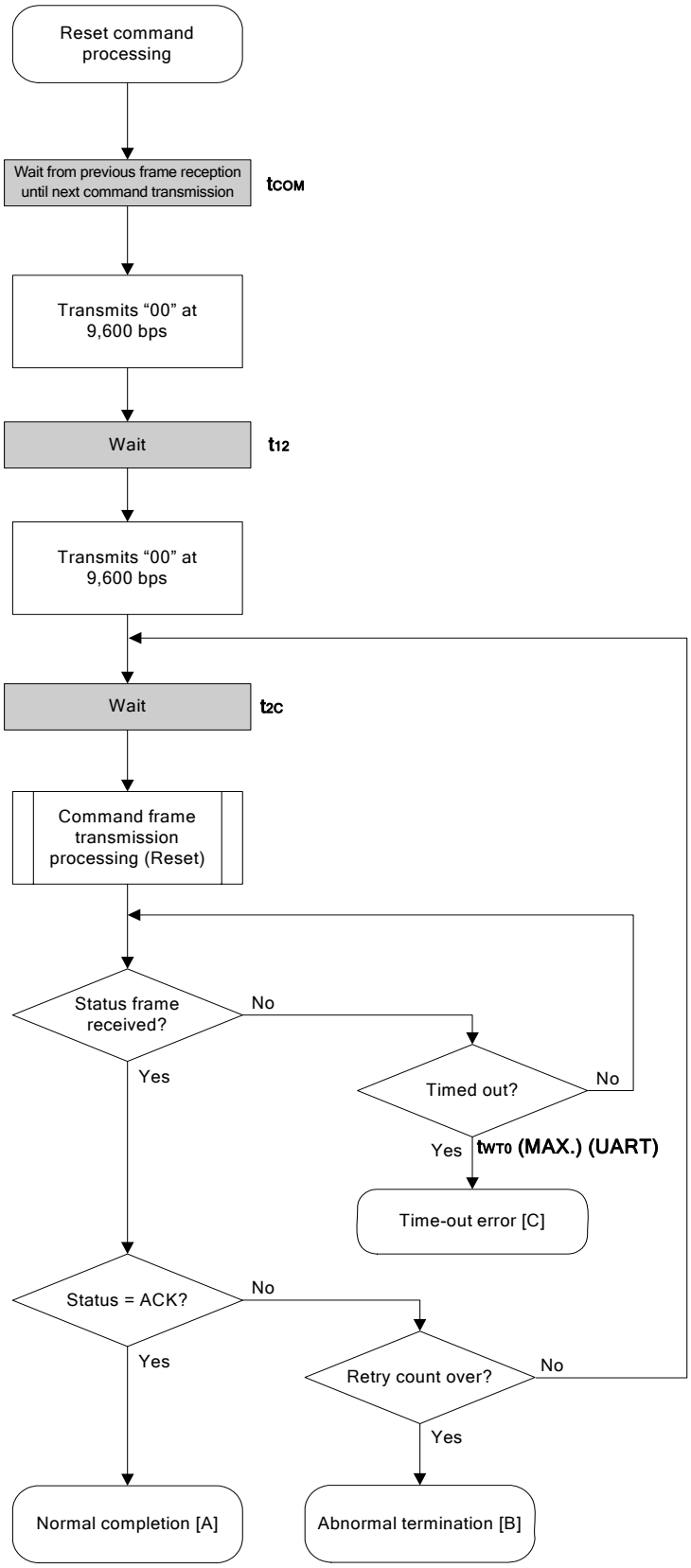
<1> Waits from the previous frame reception until the next command processing starts (wait time $t_{COM}$).
<2> The low level is output (data 00H is transmitted at 9,600 bps).
<3> Wait state (wait time $t_{12}$).
<4> The low level is output (data 00H is transmitted at 9,600 bps).
<5> Wait state (wait time $t_{2C}$).
<6> The Reset command is transmitted by command frame transmission processing.
<7> A time-out check is performed from command transmission until status frame reception.
If a time-out occurs, a time-out error [C] is returned (time-out time $t_{WT0}$ (MAX.)).
<8> The status code is checked.

When ST1 = ACK: Normal completion [A]
When ST1 ≠ ACK: The retry count ($t_{RS}$) is checked.
The sequence is re-executed from <5> if the retry count is not over.
If the retry count is over, the processing ends abnormally [B].

### 6.4.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and synchronization between the programmer and the V850ES/Hx2 has been established. |
| Abnormal termination [B] | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | The status frame was not received within the specified time. |

### 6.4.4 Flowchart



### 6.4.5 Sample program

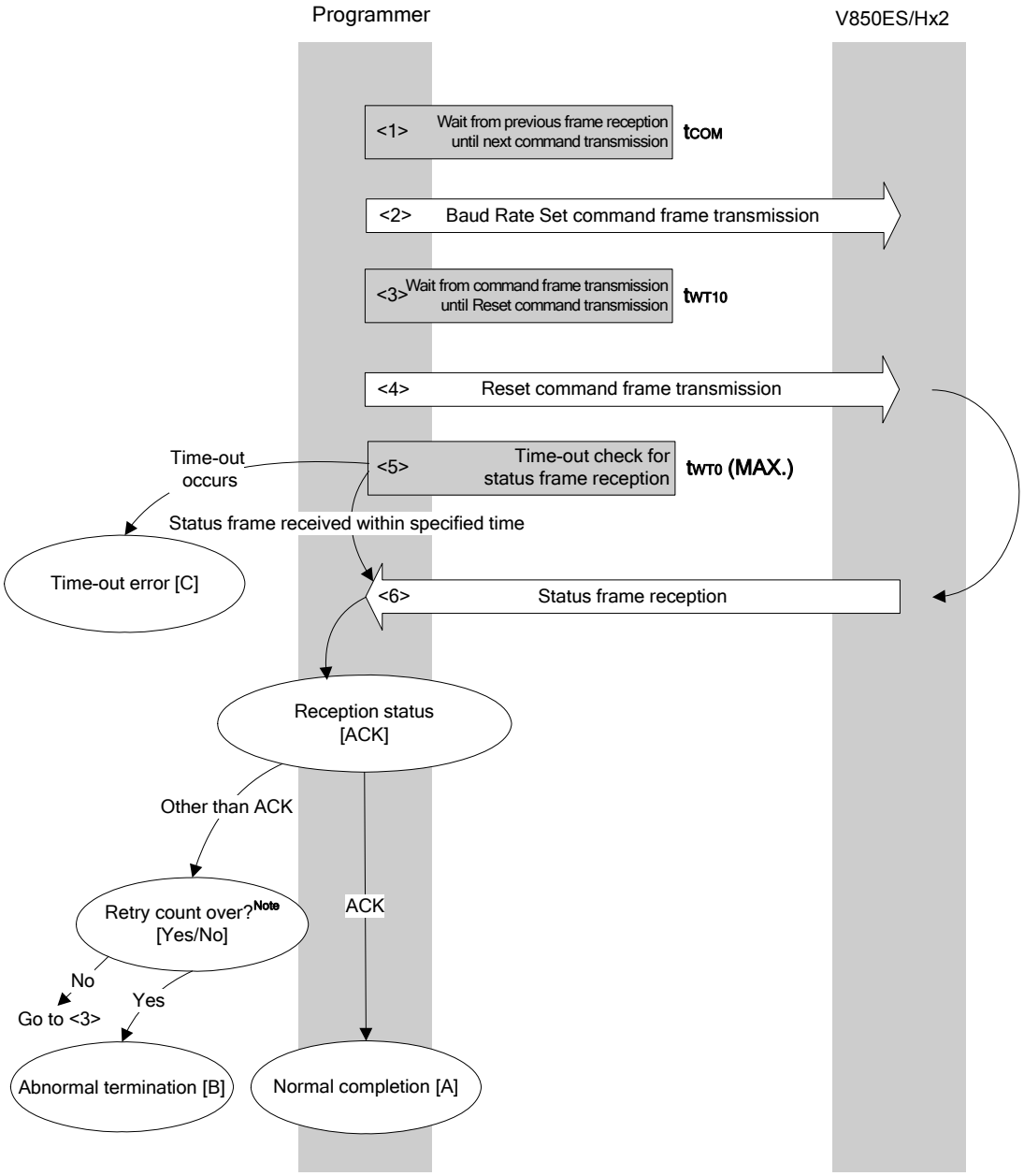
The following shows a sample program for Reset command processing.

```
/************************************************************/
/*                                                          */
/* Reset command                                            */
/*                                                          */
/************************************************************/
/* [r] u16          ... error code                          */
/************************************************************/
u16  fl_ua_reset(void)
{
     u16    rc;
     u32    retry;

     set_uart0_br(BR_9600);     // change to 9600bps

     fl_wait(tCOM_UA);          // wait
     putc_ua(0x00);             // send 0x00 @ 9600bps

     fl_wait(t12);              // wait
     putc_ua(0x00);             // send 0x00 @ 9600bps

     for (retry = 0; retry < tRS; retry++){

            fl_wait(t2C);// wait

            put_cmd_ua(FL_COM_RESET, 1, fl_cmd_prm);       // send RESET command

            rc = get_sfrm_ua(fl_ua_sfrm, tWT0_MAX);
            if (rc == FLC_DFTO_ERR)    // t.o. ?
                   break;              // yes // case [C]

            if (rc == FLC_ACK){        // ACK ?
                   break;              // yes // case [A]
            }
            else{
                   NOP();
            }
            //continue;                      // case [B] (if exit from loop)
     }
//   switch(rc) {
//
//         case   FLC_NO_ERR:   return rc;    break; // case [A]
//         case   FLC_DFTO_ERR: return rc;    break; // case [C]
//         default:             return rc;    break; // case [B]
//   }
     return rc;
}
```

## 6.5 Baud Rate Set Command

### 6.5.1 Processing sequence chart

Baud Rate Set command processing sequence




**Note** Do not exceed the retry count for the reset command transmission (up to 16 times).

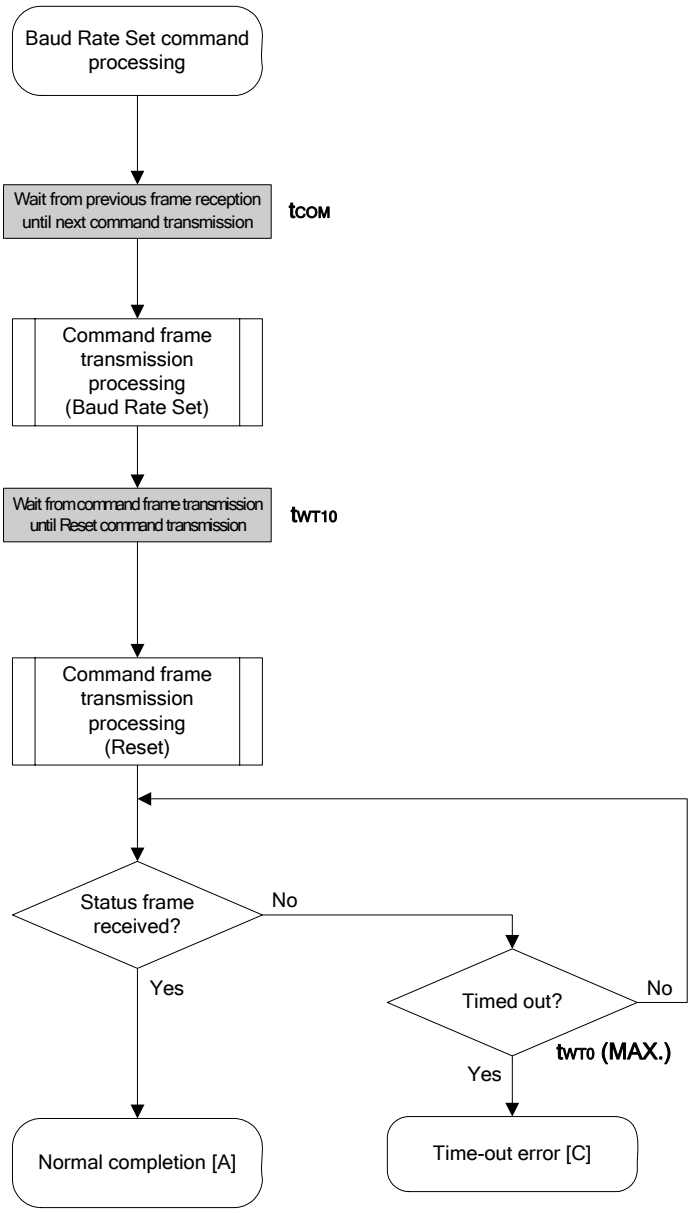
### 6.5.2 Description of processing sequence

<1> Waits from the previous frame reception until the next command transmission (wait time $t_{COM}$).
<2> The Baud Rate Set command is transmitted by command frame transmission processing.
<3> Waits from command transmission until Reset command transmission (wait time $t_{WT10}$).
<4> The Reset command is transmitted by command frame transmission processing.
<5> A time-out check is performed from command transmission until status frame reception.
If a time-out occurs, a time-out error [C] is returned (time-out time $t_{WT0}$ (MAX.)).
<6> Since the status code should be ACK, the processing ends normally [A].

### 6.5.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and the synchronization of the UART communication speed has been established between the programmer and the V850ES/Hx2. |
| Abnormal termination [B] | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Negative acknowledgment (NACK) | 15H | Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | Data frame reception was timed out.<br>With the V850ES/Hx2, this command also results in errors in the following cases.<br>• Command information (parameter) is invalid<br>• The command frame includes the checksum error<br>• The data length of the command frame (LEN) is invalid<br>• The footer of the command frame (ETX) is missing<br>• The Reset command was not detected after setting the baud rate and receiving command frame data for 16 times. |

### 6.5.4 Flowchart



### 6.5.5 Sample program

The following shows a sample program for Baud Rate Set command processing.

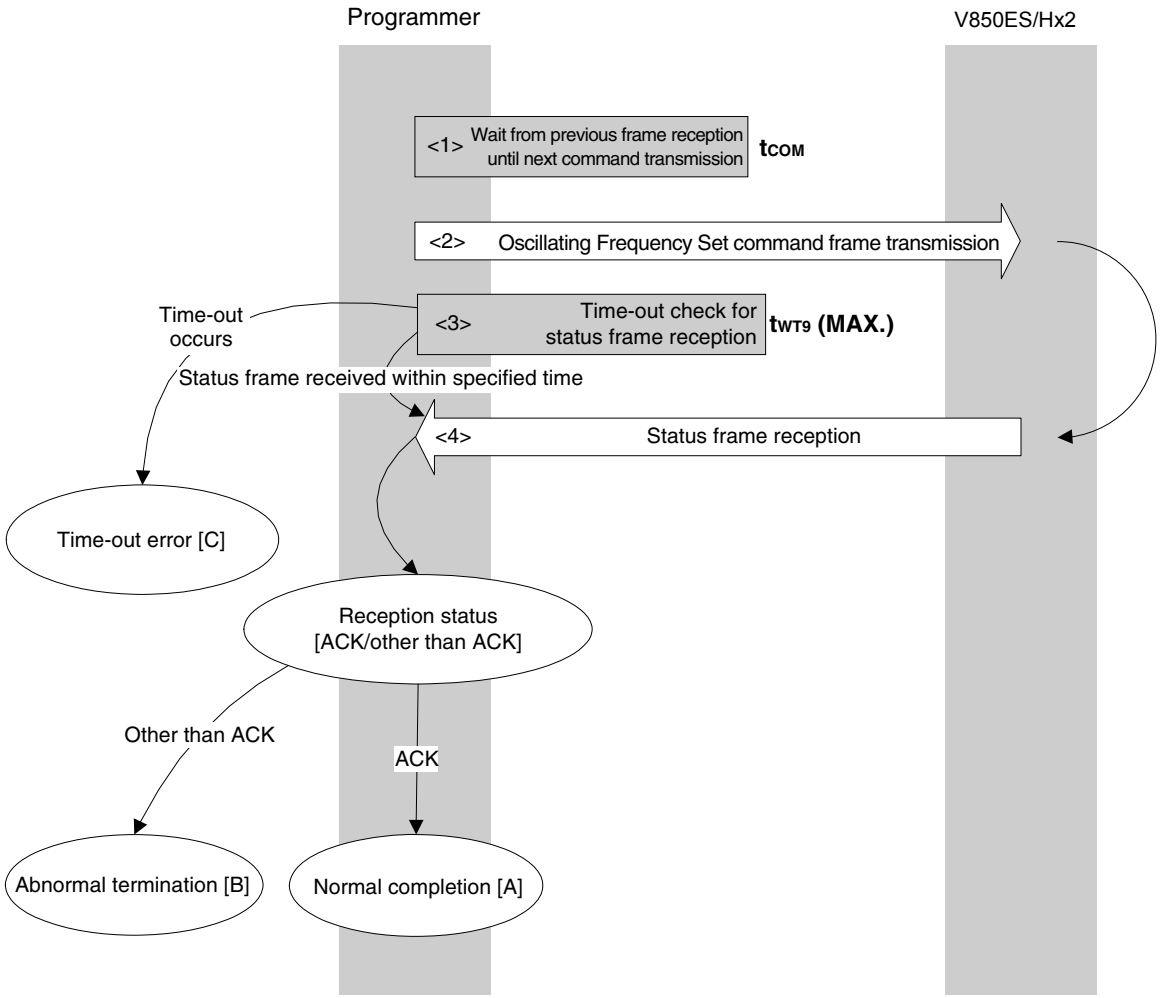
```
/************************************************************/
/*                                                          */
/* Set baudrate command                                     */
/*                                                          */
/************************************************************/
/* [i]  u8 brid ... baudrate ID                             */
/* [r] u16      ... error code                              */
/************************************************************/
u16  fl_ua_setbaud(u8 brid)
{
     u16    rc;
     u8     br;
     u32    retry;

     switch(brid){
            default:
            case   BR_9600:     br = 0x03;    break;
            case   BR_19200:    br = 0x04;    break;
            case   BR_31250:    br = 0x05;    break;
            case   BR_38400:    br = 0x06;    break;
            case   BR_76800:    br = 0x07;    break;
            case   BR_153600:   br = 0x08;    break;
     }
     fl_cmd_prm[0] = br; // "D01"


     fl_wait(tCOM_UA);   // wait before sending command
     put_cmd_ua(FL_COM_SET_BAUDRATE, 2, fl_cmd_prm); // send "Baudrate Set" command

     set_flbaud(brid);          // change baud-rate
     set_uart0_br(brid);        // change baud-rate (h.w.)

     retry = tRS;
     while(1){
            fl_wait(tWT10);

            put_cmd_ua(FL_COM_RESET, 1, fl_cmd_prm);       // send RESET command

            rc = get_sfrm_ua(fl_ua_sfrm, tWT0_MAX);        // get status frame
            if (rc){
                   if (retry--)
                          continue;
                   else
                          return rc;
            }
```

```
            break;        // got ACK !!

    }
//  switch(rc) {
//          case   FLC_NO_ERR:   return rc;    break; // case [A]
//          case   FLC_DFTO_ERR: return rc;    break; // case [C]
//          default:             return rc;    break; // case [B]
//  }

    return rc;
}
```

## 6.6 Oscillating Frequency Set Command

### 6.6.1 Processing sequence chart

Oscillating Frequency Set command processing sequence



### 6.6.2 Description of processing sequence

<1> Waits from the previous frame reception until the next command transmission (wait time $t_{COM}$).

<2> The Oscillating Frequency Set command is transmitted by command frame transmission processing.

<3> A time-out check is performed from command transmission until status frame reception.
If a time-out occurs, a time-out error [C] is returned (time-out time $t_{WT9}$(MAX.)).

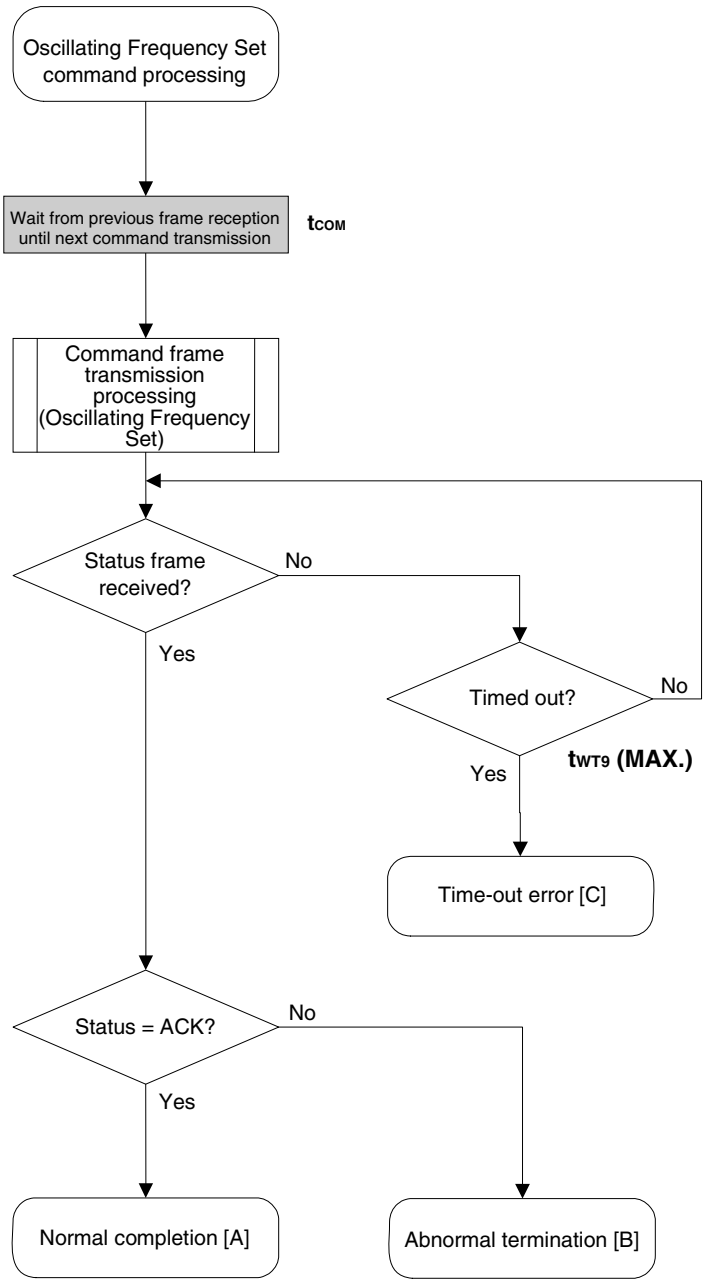
<4> The status code is checked.

When ST1 = ACK: Normal completion [A]
When ST1 ≠ ACK: Abnormal termination [B]

### 6.6.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and the operating frequency was correctly set to the V850ES/Hx2. |
| Abnormal termination [B] | Parameter error | 05H | The oscillation frequency value is out of range. |
| | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | The status frame was not received within the specified time. |

### 6.6.4 Flowchart



### 6.6.5 Sample program

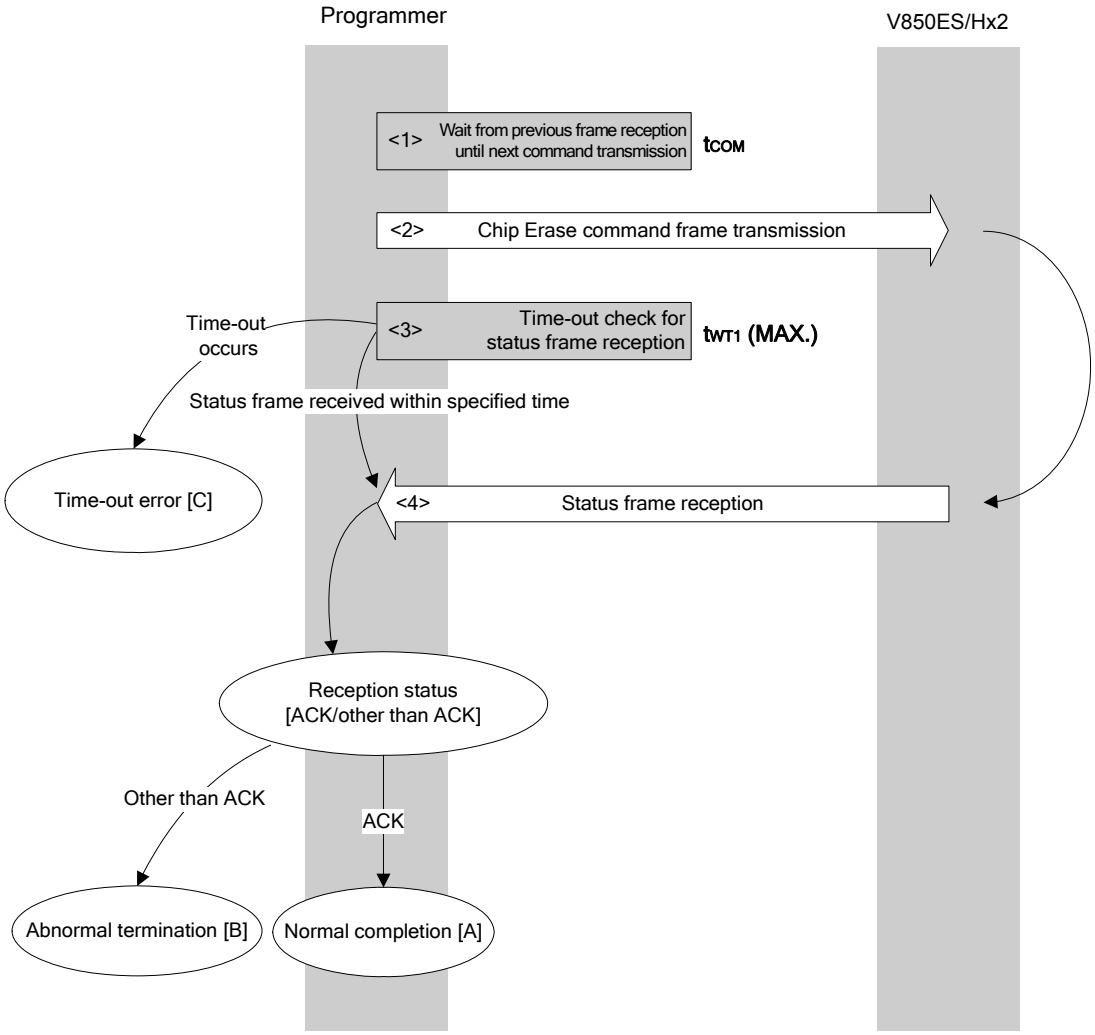
The following shows a sample program for Oscillating Frequency Set command processing.

```
/************************************************************/
/*                                                          */
/* Set Flash device clock value command                     */
/*                                                          */
/************************************************************/
/* [i] u8 clk[4]  ... frequency data(D1-D4)                 */
/* [r] u16        ... error code                            */
/************************************************************/
u16         fl_ua_setclk(u8 clk[])
{
     u16    rc;

     fl_cmd_prm[0] = clk[0];    // "D01"
     fl_cmd_prm[1] = clk[1];    // "D02"
     fl_cmd_prm[2] = clk[2];    // "D03"
     fl_cmd_prm[3] = clk[3];    // "D04"


     fl_wait(tCOM_UA);          // wait before sending command
     put_cmd_ua(FL_COM_SET_OSC_FREQ, 5, fl_cmd_prm);

     rc = get_sfrm_ua(fl_ua_sfrm, tWT9_MAX); // get status frame
//   switch(rc) {
//
//         case   FLC_NO_ERR:   return rc;    break; // case [A]
//         case   FLC_DFTO_ERR: return rc;    break; // case [C]
//         default:             return rc;    break; // case [B]
//   }

     return rc;
}
```

## 6.7 Chip Erase Command

### 6.7.1 Processing sequence chart

Chip Erase command processing sequence



### 6.7.2 Description of processing sequence

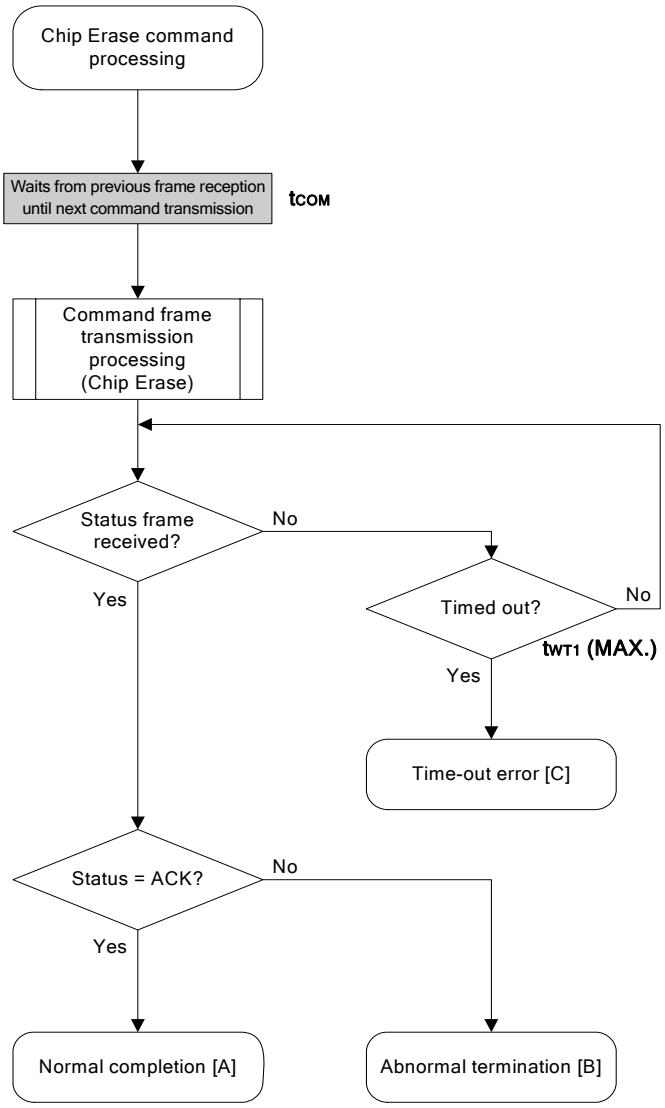
<1> Waits from the previous frame reception until the next command transmission (wait time $t_{COM}$).
<2> The Chip Erase command is transmitted by command frame transmission processing.
<3> A time-out check is performed from command transmission until status frame reception.
If a time-out occurs, a time-out error [C] is returned (time-out time $t_{WT1}$(MAX.)).
<4> The status code is checked.

When ST1 = ACK: Normal completion [A]
When ST1 ≠ ACK: Abnormal termination [B]

### 6.7.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and chip erase was performed normally. |
| Abnormal termination [B] | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Protect error | 10H | Chip erase is prohibited in the security setting. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| | WWV1 error | 08H | An erase error has occurred. |
| | EWV1 error | 0BH | |
| | EWV2 error | 0CH | |
| | EWV3 error | 0DH | |
| | Compaction search error | 13H | |
| | Sequencer error | 16H | A sequencer error has occurred. |
| Time-out error [C] | | – | The status frame was not received within the specified time. |

### 6.7.4 Flowchart



### 6.7.5 Sample program

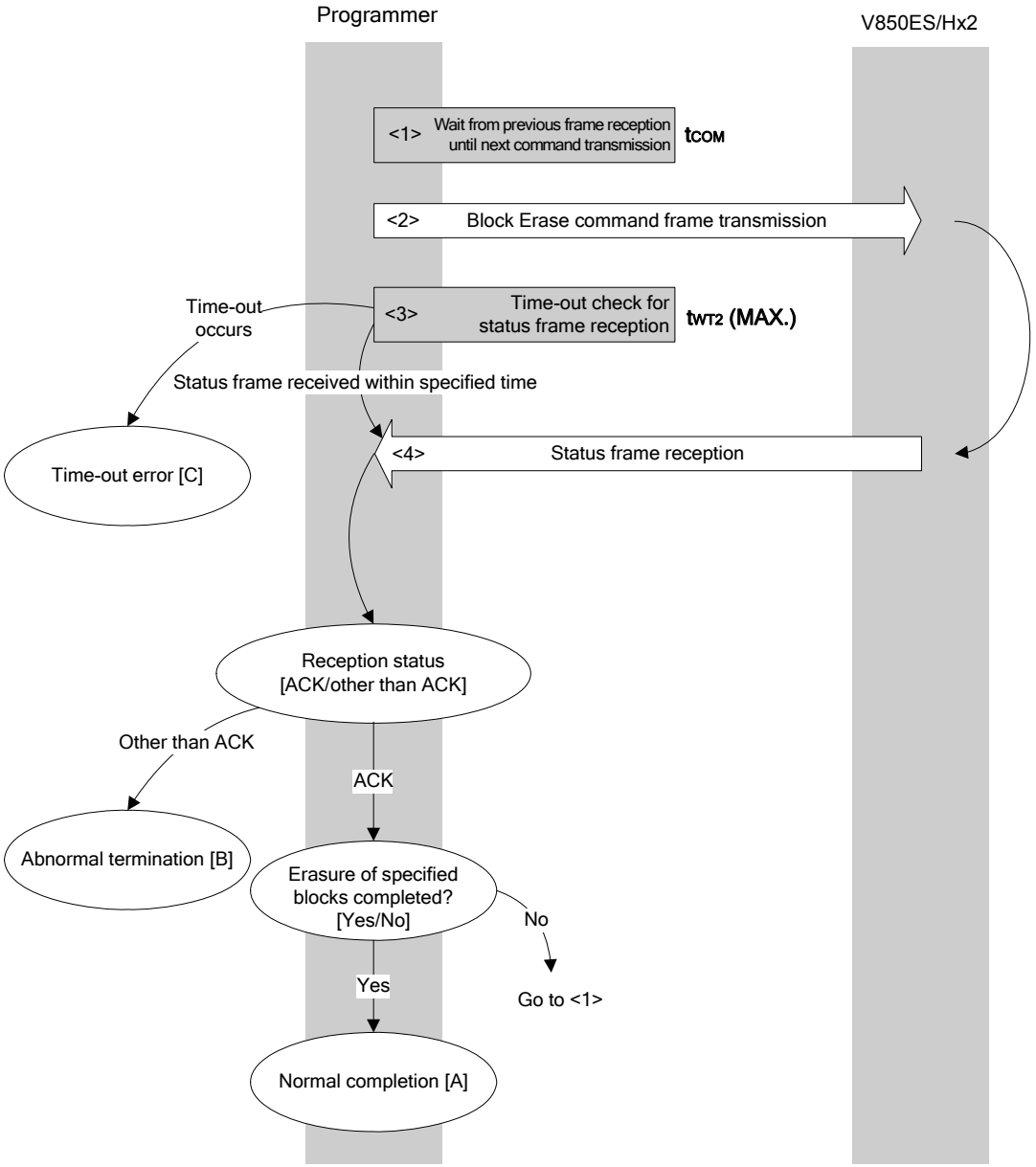
The following shows a sample program for Chip Erase command processing.

```
/************************************************************/
/*                                                          */
/* Erase all(chip) command                                  */
/*                                                          */
/************************************************************/
/* [r] u16  ... error code                                  */
/************************************************************/
u16        fl_ua_erase_all(void)
{
     u16    rc;


     fl_wait(tCOM_UA);          // wait before sending command

     put_cmd_ua(FL_COM_ERASE_CHIP, 1, fl_cmd_prm); // send ERASE CHIP command

     rc = get_sfrm_ua(fl_ua_sfrm, tWT1_MAX); // get status frame
//   switch(rc) {
//
//         case   FLC_NO_ERR:   return rc;    break; // case [A]
//         case   FLC_DFTO_ERR: return rc;    break; // case [C]
//         default:             return rc;    break; // case [B]
//   }
     return rc;

}
```

## 6.8 Block Erase Command

### 6.8.1 Processing sequence chart

Block Erase command processing sequence



### 6.8.2 Description of processing sequence

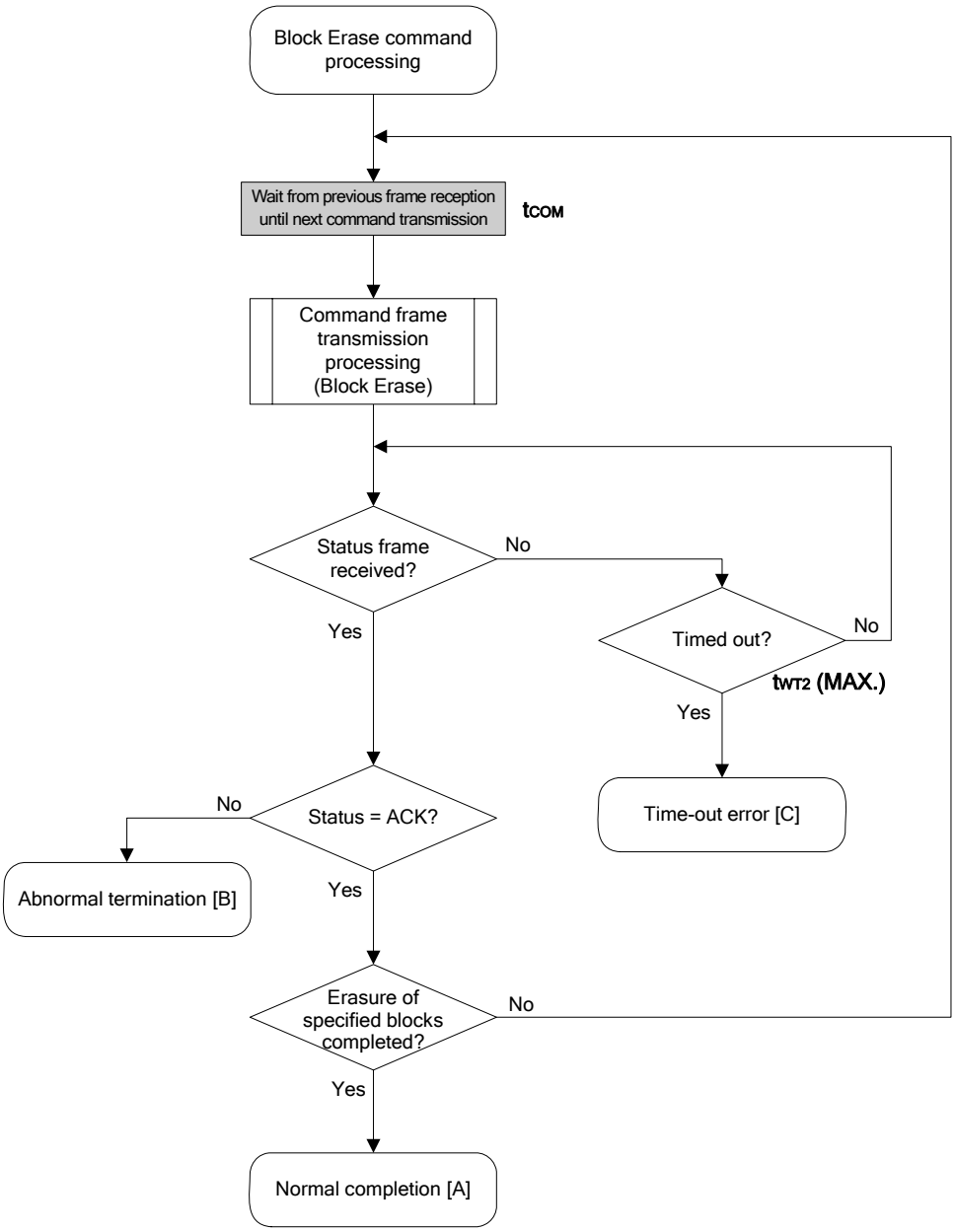
<1> Waits from the previous frame reception until the next command transmission (wait time $t_{COM}$).
<2> The Block Erase command is transmitted by command frame transmission processing.
<3> A time-out check is performed from command transmission until status frame reception.
If a time-out occurs, a time-out error [C] is returned (time-out time $t_{WT2}$(MAX.)).
<4> The status code is checked.

When ST1 = ACK: When the block erase for all of the specified blocks is not yet completed, processing changes the block number and re-executes the sequence from <1>.
When the block erase for all of the specified blocks is completed, the processing ends normally [A].
When ST1 ≠ ACK: Abnormal termination [B]

### 6.8.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and block erase was performed normally. |
| Abnormal termination [B] | Parameter error | 05H | The block number is out of range. |
| | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Protect error | 10H | Write, block erase, or chip erase is prohibited in the security setting. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| | WWV1 error | 08H | An erase error has occurred. |
| | EWV1 error | 0BH | |
| | EWV2 error | 0CH | |
| | EWV3 error | 0DH | |
| | Compaction search error | 13H | |
| | Sequencer error | 16H | A sequencer error has occurred. |
| Time-out error [C] | | – | The status frame was not received within the specified time. |

### 6.8.4 Flowchart



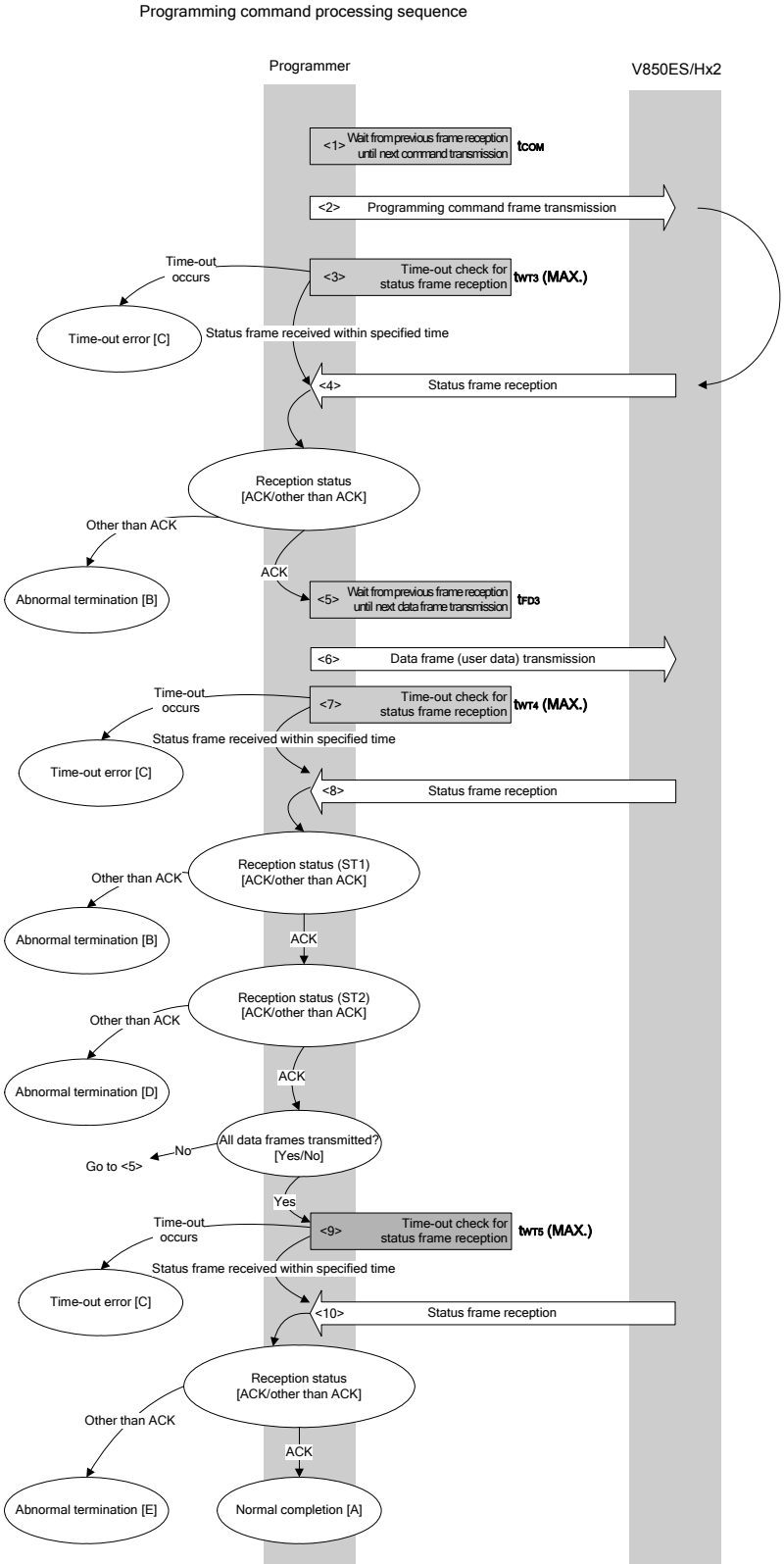
### 6.8.5 Sample program

The following shows a sample program for Block Erase command processing for one block.

```
/*****************************************************************/
/*                                                               */
/* Erase block command                                           */
/*                                                               */
/*****************************************************************/
/* [i] u8 block    ... block number                              */
/* [r] u16         ... error code                                */
/*****************************************************************/
u16         fl_ua_erase_blk(u8 block)
{
     u16    rc;
     u32    wt2_max;

     fl_cmd_prm[0] = block;     // BLK
     wt2_max = get_wt2_max(get_block_size(block));

     fl_wait(tCOM_UA);          // wait before sending command

     put_cmd_ua(FL_COM_ERASE_BLOCK, 2, fl_cmd_prm); // send ERASE CHIP command

     rc = get_sfrm_ua(fl_ua_sfrm, wt2_max);  // get status frame
//   switch(rc) {
//
//         case   FLC_NO_ERR:   return rc;    break; // case [A]
//         case   FLC_DFTO_ERR: return rc;    break; // case [C]
//         default:             return rc;    break; // case [B]
//   }

     return rc;

}
```

## 6.9 Programming Command

### 6.9.1 Processing sequence chart



### 6.9.2 Description of processing sequence

<1> Waits from the previous frame reception until the next command transmission (wait time $t_{COM}$).
<2> The Programming command is transmitted by command frame transmission processing.
<3> A time-out check is performed from command transmission until status frame reception.
If a time-out occurs, a time-out error [C] is returned (time-out time $t_{WT3}$(MAX.)).
<4> The status code is checked.

When ST1 = ACK: Proceeds to <5>.
When ST1 ≠ ACK: Abnormal termination [B]

<5> Waits from the previous frame reception until the next data frame transmission (wait time $t_{FD3}$).
<6> User data is transmitted by data frame transmission processing.
<7> A time-out check is performed from user data transmission until data frame reception.
If a time-out occurs, a time-out error [C] is returned (time-out time $t_{WT4}$(MAX.)).
<8> The status code (ST1/ST2) is checked (also refer to the processing sequence chart and flowchart).

When ST1 ≠ ACK: Abnormal termination [B]
When ST1 = ACK: The following processing is performed according to the ST2 value.
- When ST2 = ACK: Proceeds to <9> when transmission of all data frames is completed.
If there still remain data frames to be transmitted, the processing re-executes the sequence from <5>.
- When ST2 ≠ ACK: Abnormal termination [D]

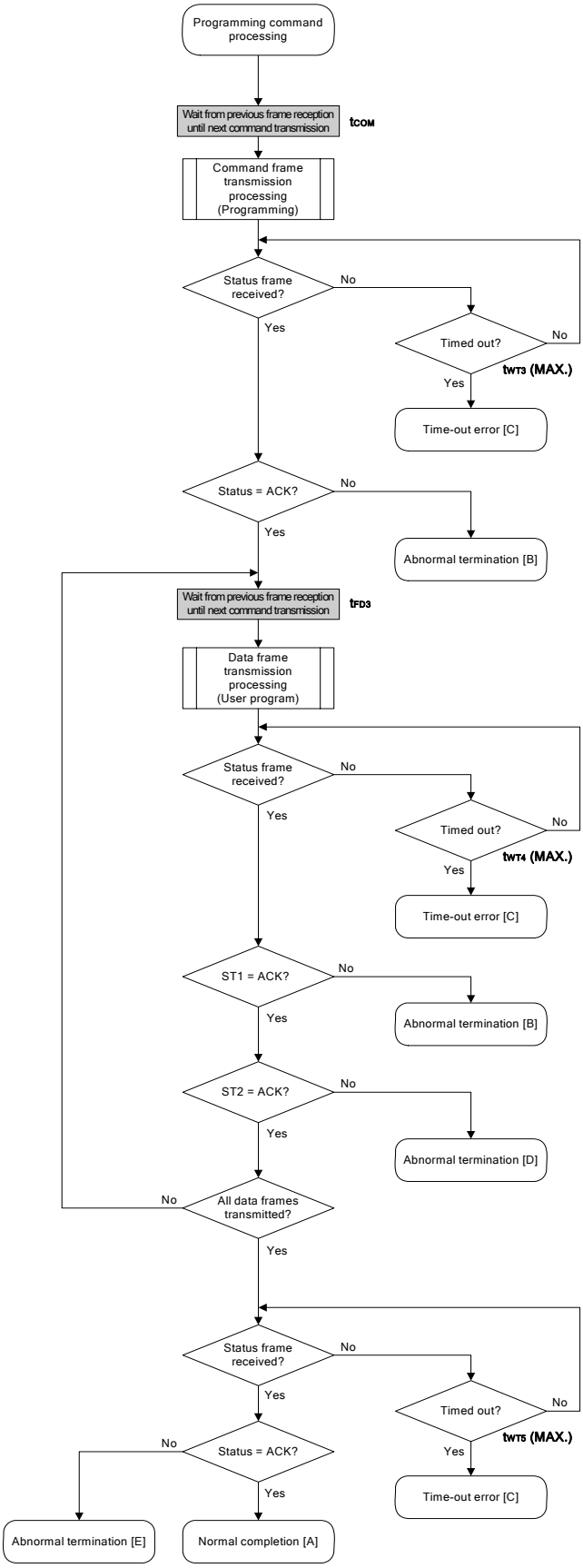
<9> A time-out check is performed until status frame reception.
If a time-out occurs, a time-out error [C] is returned (time-out time $t_{WT5}$(MAX.)).
<10> The status code is checked.

When ST1 = ACK: Normal completion [A]
When ST1 ≠ ACK: Abnormal termination [E]

### 6.9.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and the user data was written normally. |
| Abnormal termination [B] | Parameter error | 05H | The specified start/end address is not the start/end address of the block. |
| | Checksum error | 07H | The checksum of the transmitted command frame or data frame does not match. |
| | Protect error | 10H | Write is prohibited in the security setting. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | The status frame was not received within the specified time. |
| Abnormal termination [D] | WWV1 error | 08H (ST2) | A write error has occurred. |
| | Sequencer error | 16H | A sequencer error has occurred. |
| Abnormal termination [E] | EWV4 error | 11H | An internal verify error has occurred. |
| | Sequencer error | 16H | A sequencer error has occurred. |

### 6.9.4 Flowchart



### 6.9.5 Sample program

The following shows a sample program for Programming command processing.

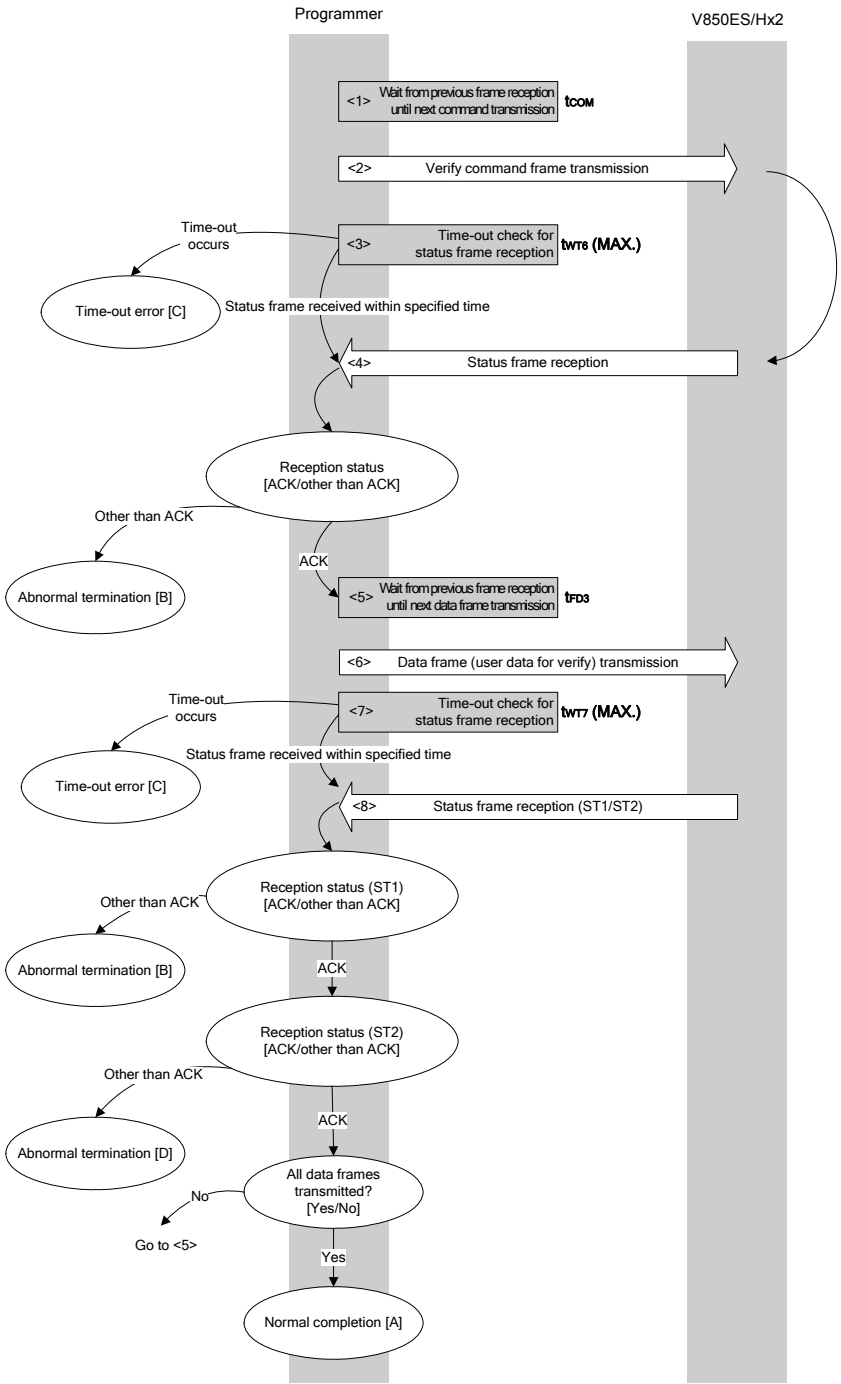
```
/************************************************************/
/*                                                          */
/* Write command                                            */
/*                                                          */
/************************************************************/
/* [i] u32 top     ... start address                        */
/* [i] u32 bottom  ... end address                          */
/* [r] u16         ... error code                           */
/************************************************************/

#define           fl_st2_ua     (fl_ua_sfrm[OFS_STA_PLD+1])

u16  fl_ua_write(u32 top, u32 bottom)
{
     u16    rc;
     u32    send_head, send_size;
     bool   is_end;
     u32    wt5_max;

     /*********************************************/
     /*     set params                            */
     /*********************************************/
     set_range_prm(fl_cmd_prm, top, bottom); // set SAH/SAM/SAL, EAH/EAM/EAL
     wt5_max = get_wt5_max(bottom - top + 1);

     /*********************************************/
     /*     send command & check status           */
     /*********************************************/
     fl_wait(tCOM);             // wait before sending command

     put_cmd_ua(FL_COM_WRITE, 7, fl_cmd_prm); // send "Programming" command

     rc = get_sfrm_ua(fl_ua_sfrm, tWT3_MAX); // get status frame
     switch(rc) {
            case   FLC_NO_ERR:               break; // continue
     //     case   FLC_DFTO_ERR: return rc;  break; // case [C]
            default:             return rc;  break; // case [B]
     }

     /*********************************************/
     /*     send user data                        */
     /*********************************************/
     send_head = top;

            while(1){
```

```
        // make send data frame
        if ((bottom - send_head) > 256){  // rest size > 256 ?
              is_end = false;             // yes, not is_end frame
              send_size  = 256;           // transmit size = 256 byte
        }
        else{
              is_end = true;
              send_size = bottom - send_head + 1;     // transmit size = (bottom -
                                                      // send_head)+1 byte

        }
        memcpy(fl_txdata_frm, rom_buf+send_head, send_size);  // set data frame
                                                              // payload
        send_head += send_size;

        fl_wait(tFD3);                          // wait before sending data frame

        put_dfrm_ua(send_size, fl_txdata_frm, is_end); // send user data

        rc = get_sfrm_ua(fl_ua_sfrm, tWT4_MAX);        // get status frame
        switch(rc) {
                    case   FLC_NO_ERR:                 break; // continue
                    case   FLC_DFTO_ERR: return rc;    break; // case [C]
                    default:             return rc;    break; // case [B]
        }
        if (fl_st2_ua != FLST_ACK){                    // ST2 = ACK ?
                    rc = decode_status(fl_st2_ua);     // No
                    return rc;                         // case [D]
        }
        if (is_end)
                    break;

    }
    /*********************************************/
    /*     Check internally verify               */
    /*********************************************/
    rc = get_sfrm_ua(fl_ua_sfrm, wt5_max);             // get status frame again

    switch(rc) {
//          case   FLC_NO_ERR:   return rc;    break; // case [A]
            case   FLC_DFTO_ERR: return rc;    break; // case [C]
            default:             return rc;    break; // case [E]
    }

    return rc;
}
```

## 6.10 Verify Command

### 6.10.1 Processing sequence chart

Verify command processing sequence



### 6.10.2 Description of processing sequence

<1> Waits from the previous frame reception until the next command transmission (wait time $t_{COM}$).
<2> The Verify command is transmitted by command frame transmission processing.
<3> A time-out check is performed from command transmission until status frame reception.
If a time-out occurs, a time-out error [C] is returned (time-out time $t_{WT6}$(MAX.)).
<4> The status code is checked.

When ST1 = ACK: Proceeds to <5>.
When ST1 ≠ ACK: Abnormal termination [B]

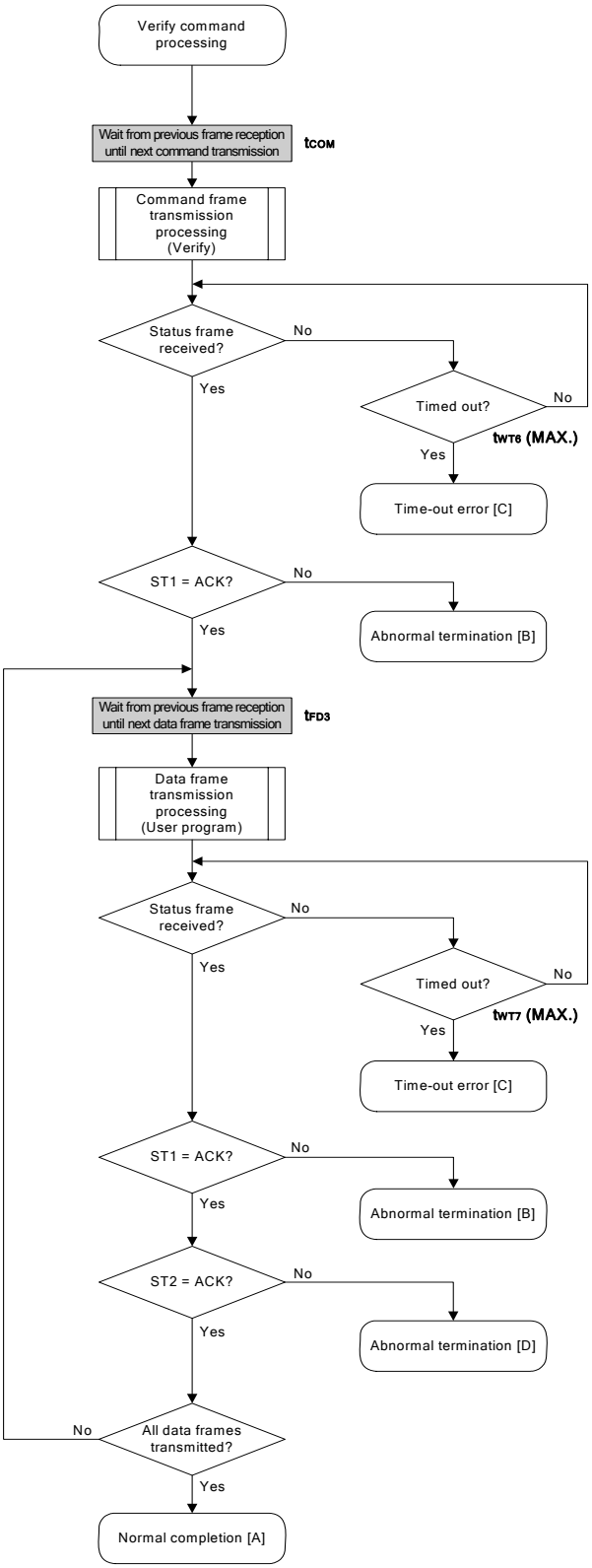
<5> Waits from the previous frame reception until the next data frame transmission (wait time $t_{FD3}$).
<6> User data for verifying is transmitted by data frame transmission processing.
<7> A time-out check is performed from user data transmission until status frame reception.
If a time-out occurs, a time-out error [C] is returned (time-out time $t_{WT7}$(MAX.)).
<8> The status code (ST1/ST2) is checked (also refer to the processing sequence chart and flowchart).

When ST1 ≠ ACK: Abnormal termination [B]
When ST1 = ACK: The following processing is performed according to the ST2 value.
- When ST2 = ACK: If transmission of all data frames is completed, the processing ends normally [A].
  If there still remain data frames to be transmitted, the processing re-executes the sequence from <5>.
- When ST2 ≠ ACK: Abnormal termination [D]

### 6.10.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and the verify was completed normally. |
| Abnormal termination [B] | Parameter error | 05H | The specified start/end address is not the start/end address of the block. |
| | Checksum error | 07H | The checksum of the transmitted command frame or data frame does not match. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | The status frame was not received within the specified time. |
| Abnormal termination [D] | Verify error | 0FH (ST2) | The verify has failed, or another error has occurred. |
| | Sequencer error | 16H | A sequencer error has occurred. |

### 6.10.4 Flowchart



### 6.10.5 Sample program

The following shows a sample program for Verify command processing.

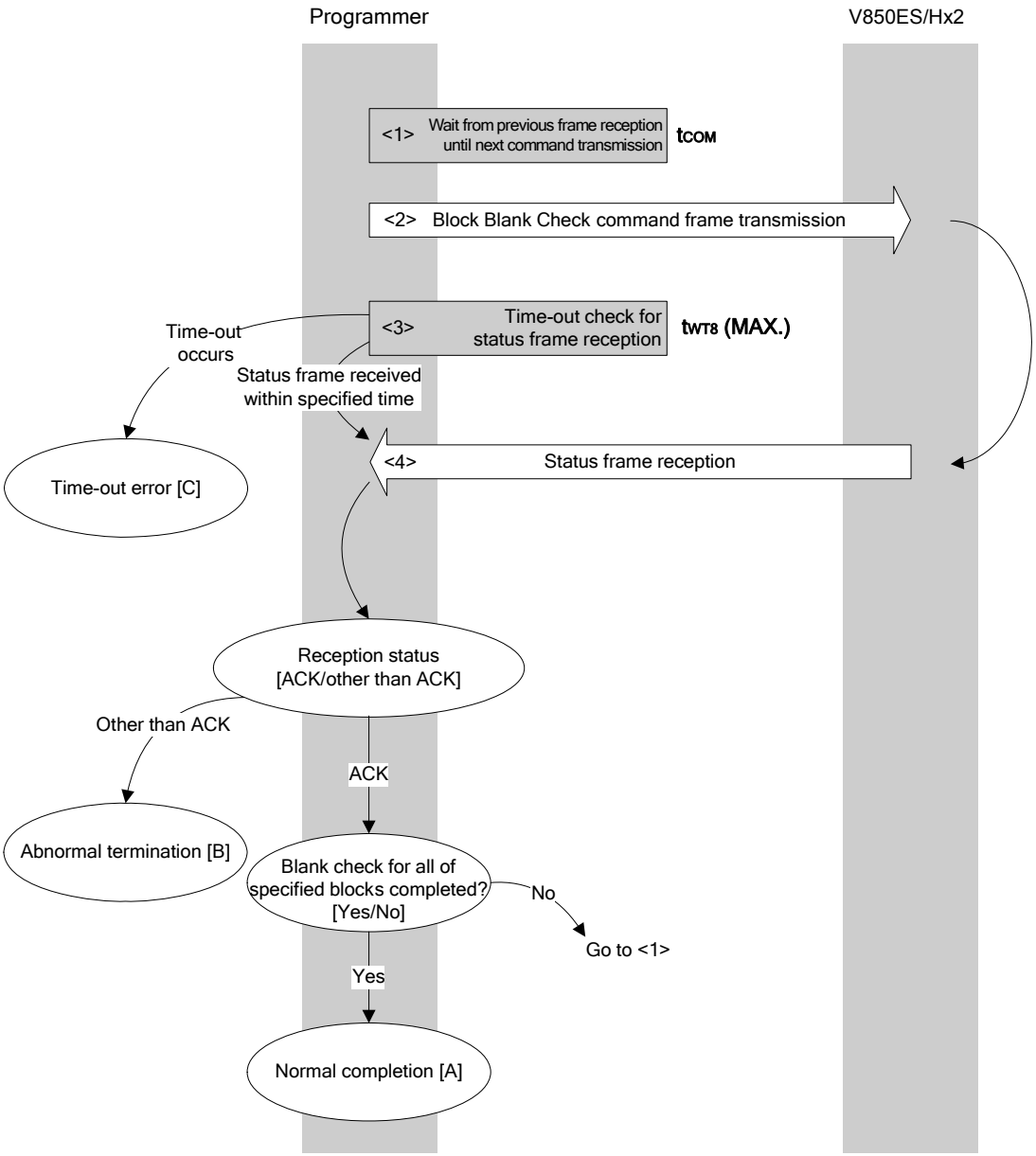
```
/*****************************************************************/
/*                                                               */
/*      Verify command                                           */
/*                                                               */
/*****************************************************************/
/*      [i] u32 top   ... start address                          */
/*      [i] u32 bottom      ... end address                      */
/*      [r] u16             ... error code                       */
/*****************************************************************/
u16     fl_ua_verify(u32 top, u32 bottom, u8 *buf)
{
  u16   rc;
  u32   send_head, send_size;
  bool  is_end;

  /*********************************************/
  /*    set params                             */
  /*********************************************/
  set_range_prm(fl_cmd_prm, top, bottom); // set SAH/SAM/SAL, EAH/EAM/EAL


  /*********************************************/
  /*    send command & check status            */
  /*********************************************/

  fl_wait(tCOM_UA);          // wait before sending command

  put_cmd_ua(FL_COM_VERIFY, 7, fl_cmd_prm);      // send VERIFY command

  rc = get_sfrm_ua(fl_ua_sfrm, tWT6_MAX);        // get status frame
  switch(rc) {
        case   FLC_NO_ERR:               break; // continue
  //    case   FLC_DFTO_ERR: return rc;  break; // case [C]
        default:           return rc;    break; // case [B]
  }

  /*********************************************/
  /*    send user data                         */
  /*********************************************/
  send_head = top;

  while(1){

        // make send data frame
        if ((bottom - send_head) > 256){        // rest size > 256 ?
              is_end = false;                   // yes, not is_end frame
              send_size  = 256;                 // transmit size = 256 byte
        }
        else{
              is_end = true;
              send_size = bottom - send_head + 1;     // transmit size = (bottom
- send_head)+1 byte

        }
        memcpy(fl_txdata_frm, buf+send_head, send_size);     //  set  data  frame
payload
        send_head += send_size;
```

```
            fl_wait(tFD3);
            put_dfrm_ua(send_size, fl_txdata_frm, is_end); // send user data

            rc = get_sfrm_ua(fl_ua_sfrm, tWT7_MAX);        // get status frame
            switch(rc) {
                  case   FLC_NO_ERR:                break; // continue
            //    case   FLC_DFTO_ERR: return rc;   break; // case [C]
                  default:             return rc;   break; // case [B]
            }
            if (fl_st2_ua != FLST_ACK){               // ST2 = ACK ?
                  rc = decode_status(fl_st2_ua);      // No
                  return rc;                  // case [D]
            }
            if (is_end)                 // send all user data ?
                  break;                // yes
            //continue;
      }
      return FLC_NO_ERR;   // case [A]
}
```

## 6.11 Block Blank Check Command

### 6.11.1 Processing sequence chart

Block Blank Check command processing sequence



### 6.11.2 Description of processing sequence

<1> Waits from the previous frame reception until the next command transmission (wait time $t_{COM}$).

<2> The Block Blank Check command is transmitted by command frame transmission processing.

<3> A time-out check is performed from command transmission until status frame reception.
If a time-out occurs, a time-out error [C] is returned (time-out time $t_{WT8}$(MAX.)).

<4> The status code is checked.

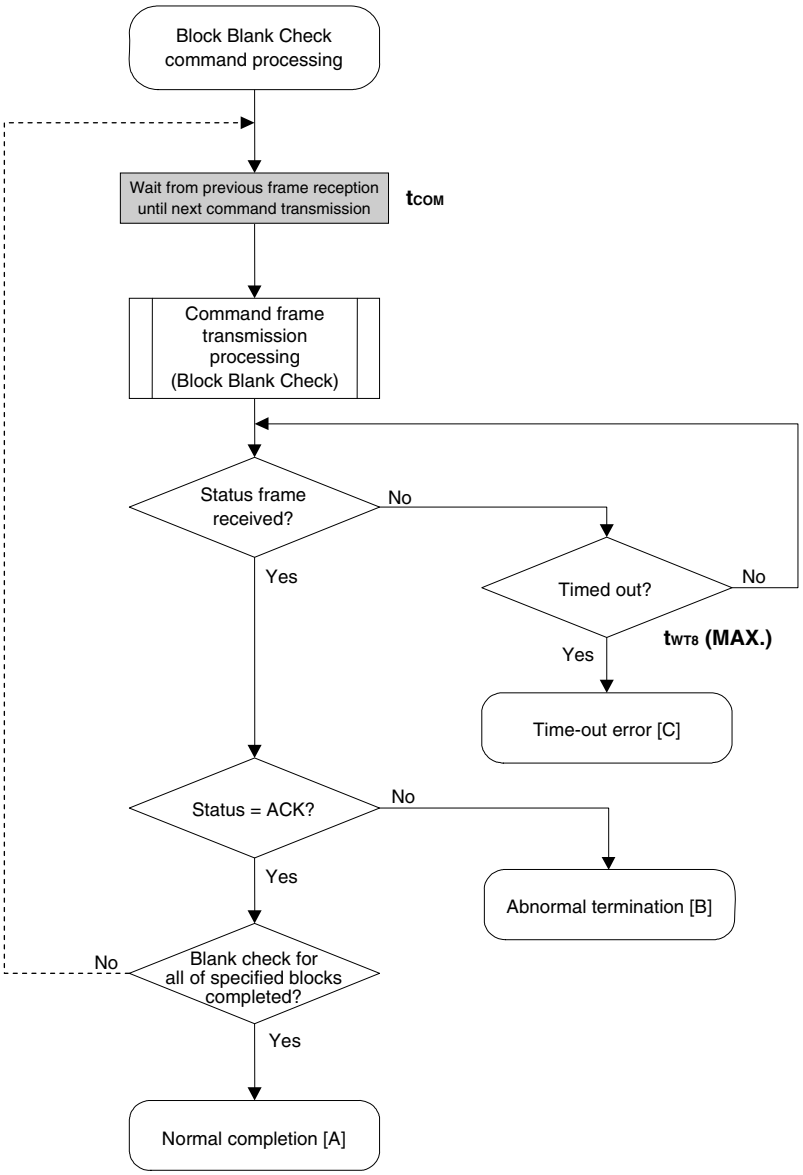
When ST1 = ACK: If the blank check for all of the specified blocks is not yet completed, processing changes the block number and re-executes the sequence from <1>.
If the blank check for all of the specified blocks is completed, the processing ends normally [A].

When ST1 ≠ ACK: Abnormal termination [B]

### 6.11.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and all of the specified blocks are blank. |
| Abnormal termination [B] | Parameter error | 05H | The block number is out of range. |
| | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| | EWV4 error | 11H | The specified block in the flash memory is not blank. |
| | Sequencer error | 16H | A sequencer error has occurred. |
| Time-out error [C] | | – | The status frame was not received within the specified time. |

### 6.11.4 Flowchart



### 6.11.5 Sample program

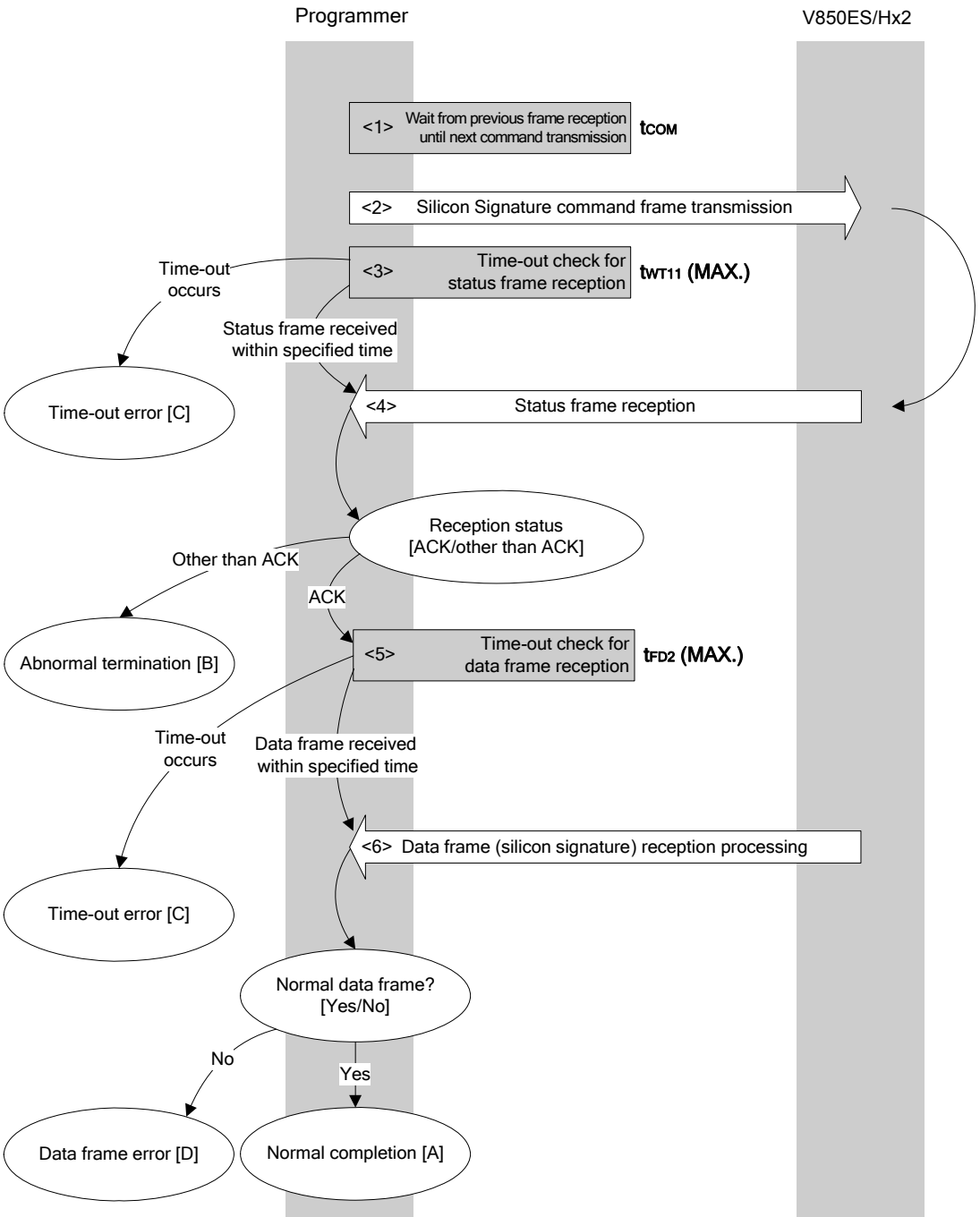
The following shows a sample program for Block Blank Check command processing for one block.

```
/************************************************************/
/*                                                          */
/* Block blank check command                                */
/*                                                          */
/************************************************************/
/* [i]  u8 block  ... block number                          */
/* [r] u16        ... error code                            */
/************************************************************/
u16         fl_ua_blk_blank_chk(u8 block)
{
     u16    rc;
     u32    wt8_max;

     fl_cmd_prm[0] = block;     // "BLK"
     wt8_max = get_wt8_max(get_block_size(block));

     fl_wait(tCOM_UA);                // wait before sending command

     put_cmd_ua(FL_COM_BLOCK_BLANK_CHK, 2, fl_cmd_prm);


     rc = get_sfrm_ua(fl_ua_sfrm, wt8_max);         // get status frame
//   switch(rc) {
//
//         case          FLC_NO_ERR:   return rc;    break; // case [A]
//         case          FLC_DFTO_ERR: return rc;    break; // case [C]
//         default:                    return rc;    break; // case [B]
//   }
     return rc;

}
```

## 6.12 Silicon Signature Command

### 6.12.1 Processing sequence chart

Silicon Signature command processing sequence



### 6.12.2 Description of processing sequence

<1> Waits from the previous frame reception until the next command transmission (wait time $t_{COM}$).

<2> The Silicon Signature command is transmitted by command frame transmission processing.

<3> A time-out check is performed from command transmission until status frame reception.
If a time-out occurs, a time-out error [C] is returned (time-out time $t_{WT11}$(MAX.)).

<4> The status code is checked.

When ST1 = ACK: Proceeds to <5>.
When ST1 ≠ ACK: Abnormal termination [B]

<5> A time-out check is performed until data frame (silicon signature data) reception.
If a time-out occurs, a time-out error [C] is returned (time-out time $t_{FD2}$(MAX.)).

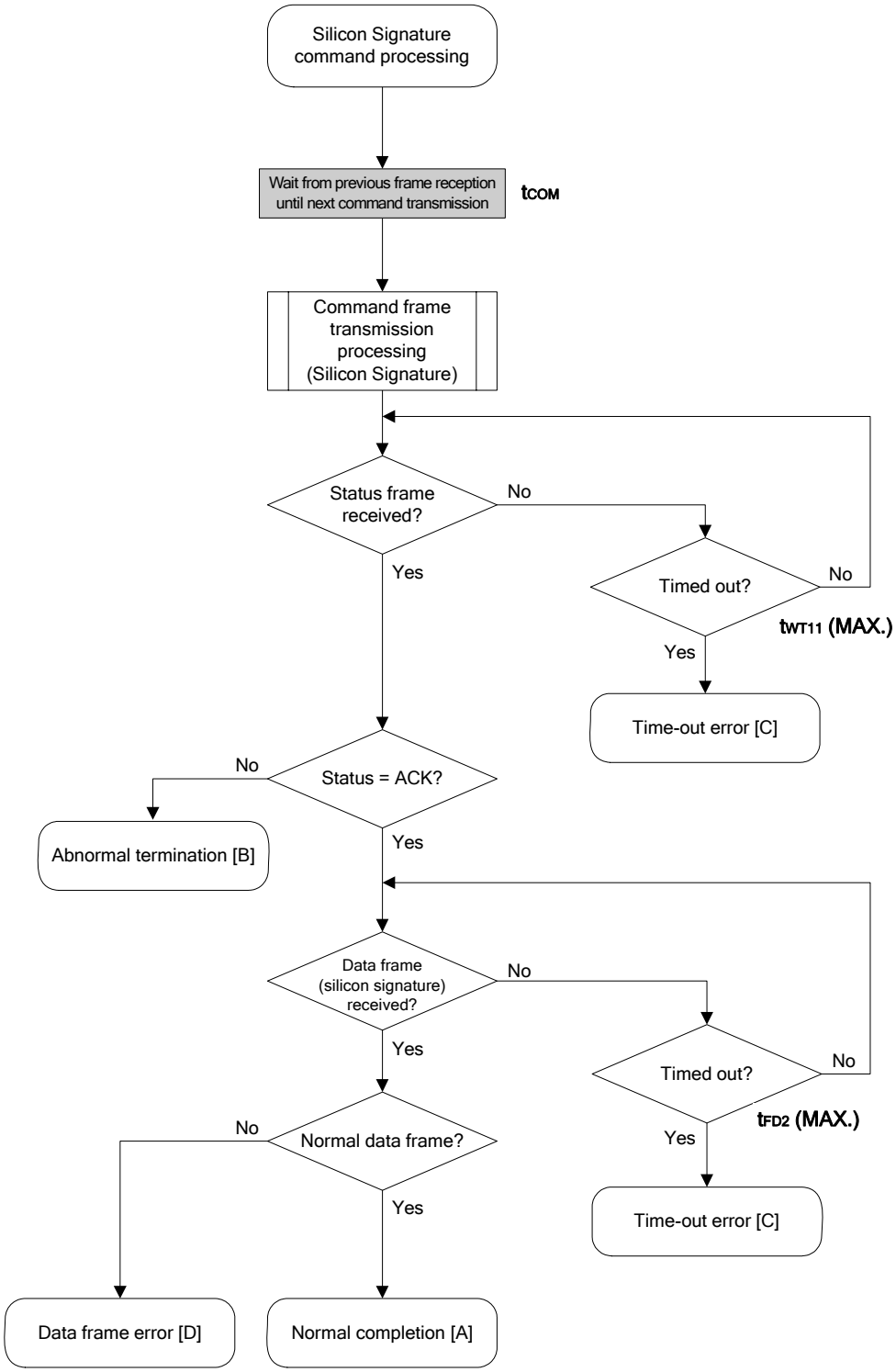
<6> The received data frame (silicon signature data) is checked.

If data frame is normal: Normal completion [A]
If data frame is abnormal: Data frame error [D]

### 6.12.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and the silicon signature was acquired normally. |
| Abnormal termination [B] | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | The status frame or data frame was not received within the specified time. |
| Data frame error [D] | | – | The checksum of the data frame received as silicon signature data does not match. |

### 6.12.4 Flowchart



### 6.12.5 Sample program

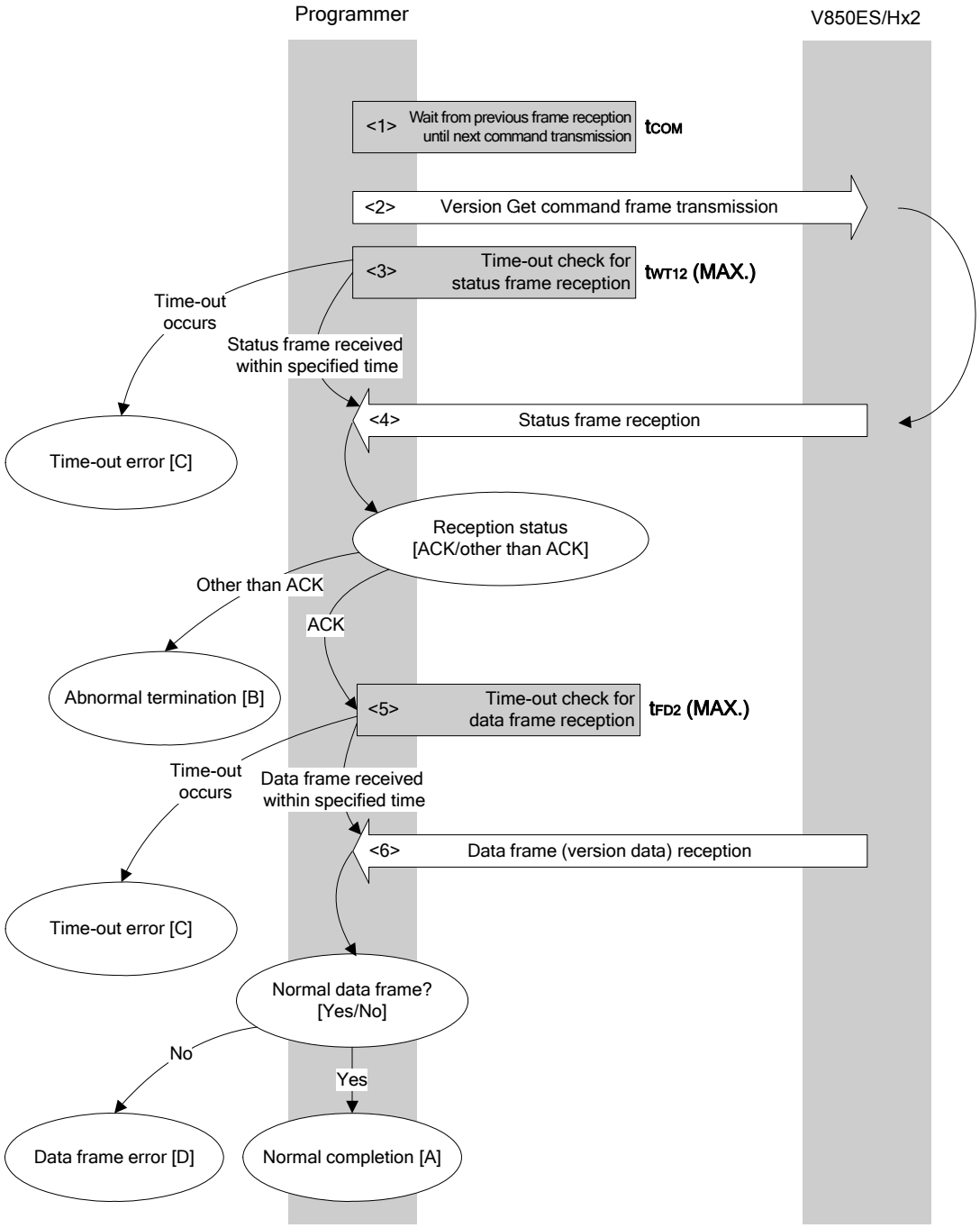
The following shows a sample program for Silicon Signature command processing.

```
/************************************************************/
/*                                                          */
/* Get silicon signature command                            */
/*                                                          */
/************************************************************/
/* [i] u8 *sig  ... pointer to signature save area          */
/* [r] u16      ... error code                              */
/************************************************************/
u16        fl_ua_getsig(u8 *sig)
{
     u16    rc;


     fl_wait(tCOM_UA);          // wait before sending command

     put_cmd_ua(FL_COM_GET_SIGNATURE, 1, fl_cmd_prm); // send GET SIGNATURE command

     rc = get_sfrm_ua(fl_ua_sfrm, tWT11_MAX);         // get status frame
     switch(rc) {
           case   FLC_NO_ERR:               break; // continue
     //    case   FLC_DFTO_ERR: return rc;  break; // case [C]
           default:             return rc;  break; // case [B]
     }


     rc = get_dfrm_ua(fl_rxdata_frm, tFD2_MAX);     // get status frame
     if (rc){                          // if error
           return rc;                  // case [D]
     }
     memcpy(sig, fl_rxdata_frm+OFS_STA_PLD, fl_rxdata_frm[OFS_LEN]);
                                                    // copy Signature data
     return rc;                        // case [A]
}
```

## 6.13 Version Get Command

### 6.13.1 Processing sequence chart

Version Get command processing sequence



### 6.13.2 Description of processing sequence

<1> Waits from the previous frame reception until the next command transmission (wait time $t_{COM}$).

<2> The Version Get command is transmitted by command frame transmission processing.

<3> A time-out check is performed from command transmission until status frame reception.
If a time-out occurs, a time-out error [C] is returned (time-out time $t_{WT12}$(MAX.)).

<4> The status code is checked.

When ST1 = ACK: Proceeds to <5>.
When ST1 ≠ ACK: Abnormal termination [B]

<5> A time-out check is performed until data frame (version data) reception.
If a time-out occurs, a time-out error [C] is returned (time-out time $t_{FD2}$(MAX.)).

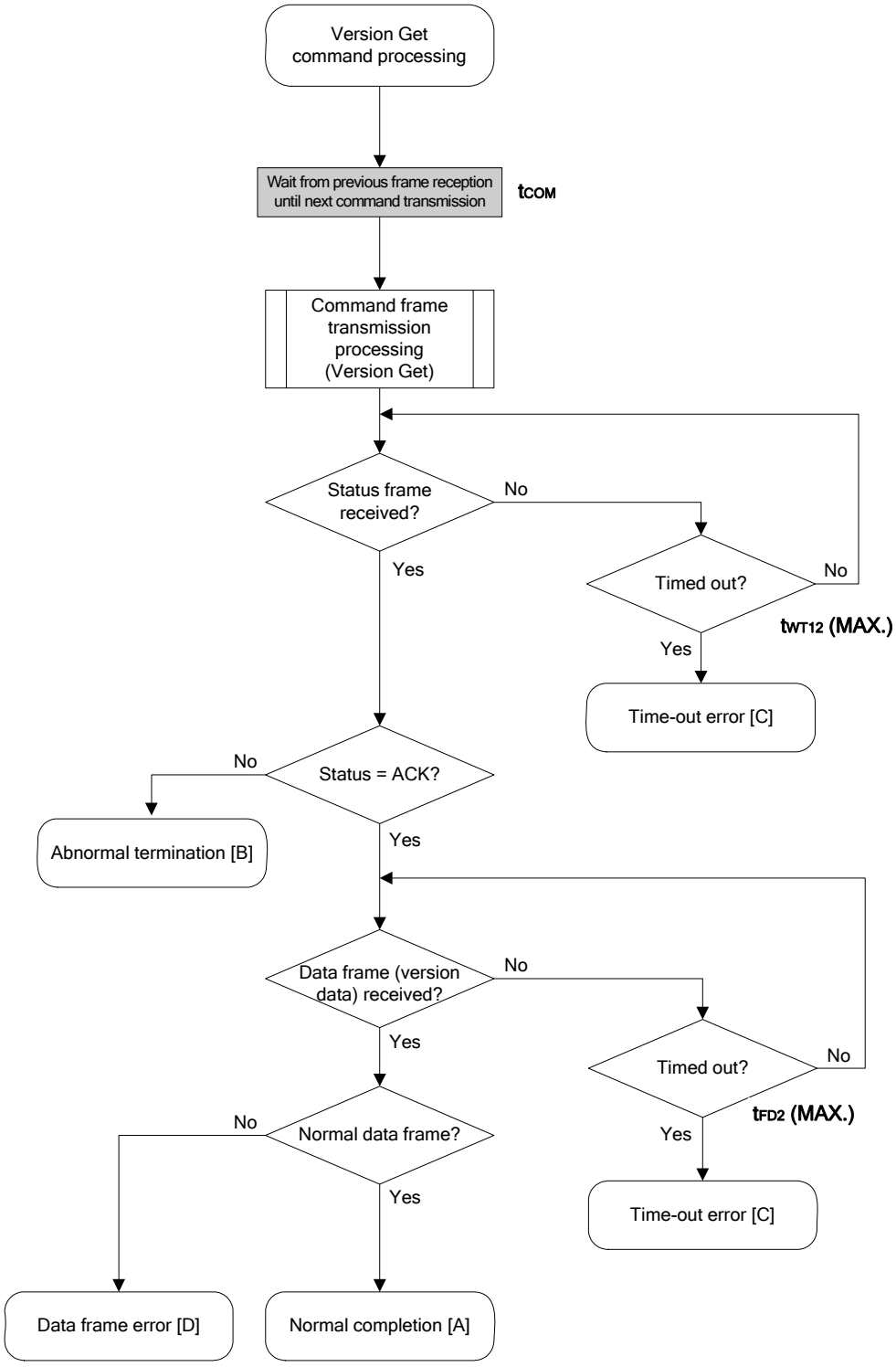
<6> The received data frame (version data) is checked.

If data frame is normal: Normal completion [A]
If data frame is abnormal: Data frame error [D]

### 6.13.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and version data was acquired normally. |
| Abnormal termination [B] | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | The status frame or data frame was not received within the specified time. |
| Data frame error [D] | | – | The checksum of the data frame received as version data does not match. |

### 6.13.4 Flowchart



### 6.13.5 Sample program

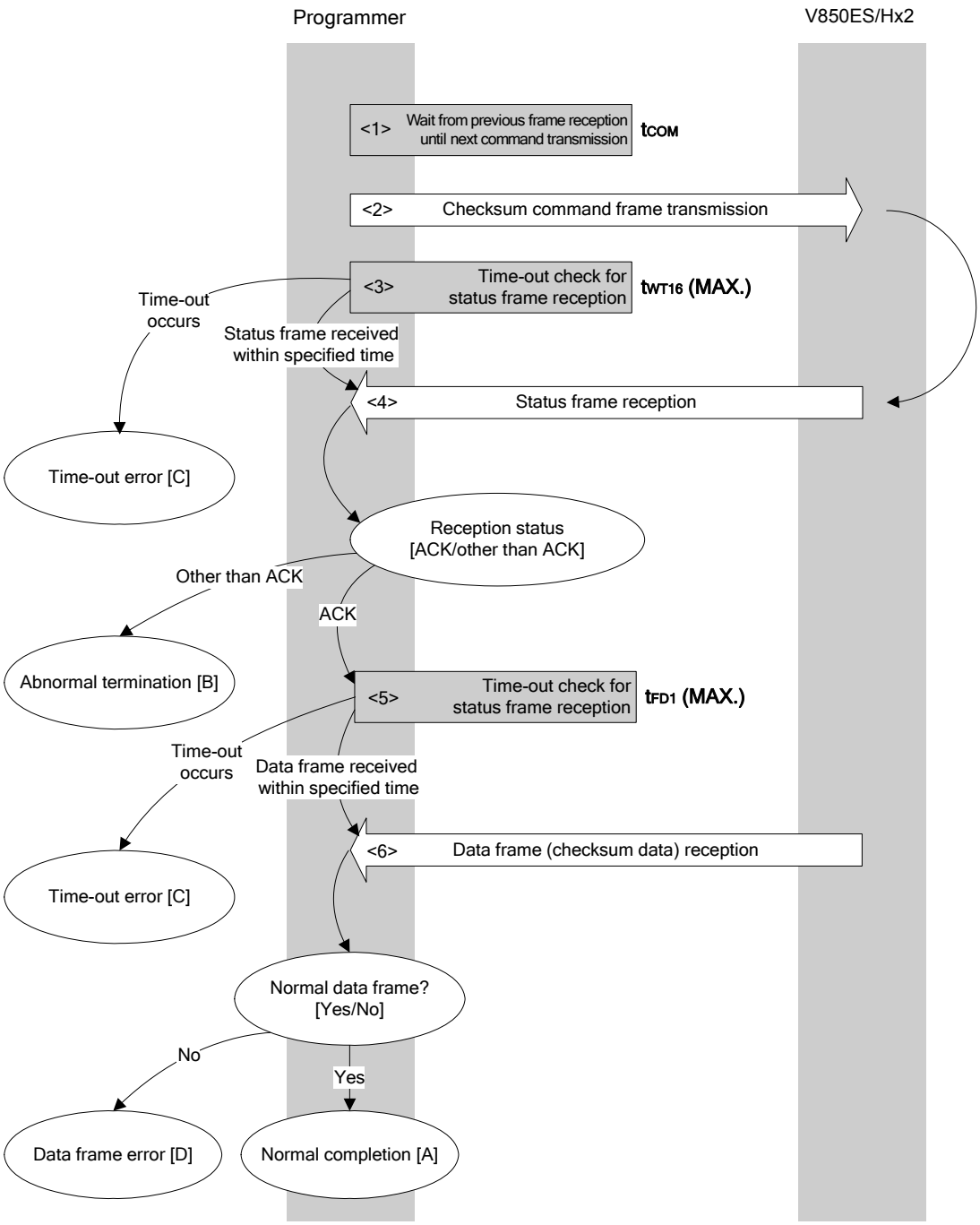
The following shows a sample program for Version Get command processing.

```
/************************************************************/
/*                                                          */
/* Get device/firmware version command                      */
/*                                                          */
/************************************************************/
/* [i] u8 *buf ... pointer to version date save area        */
/* [r] u16     ... error code                               */
/************************************************************/
u16         fl_ua_getver(u8 *buf)
{
     u16    rc;


     fl_wait(tCOM_UA);                // wait before sending command


     put_cmd_ua(FL_COM_GET_VERSION, 1, fl_cmd_prm); // send GET VERSION command

     rc = get_sfrm_ua(fl_ua_sfrm, tWT12_MAX);       // get status frame
     switch(rc) {
           case   FLC_NO_ERR:                       break; // continue
     //    case   FLC_DFTO_ERR:       return rc;    break; // case [C]
           default:                   return rc;    break; // case [B]
     }

     rc = get_dfrm_ua(fl_rxdata_frm, tFD2_MAX);     // get data frame
     if (rc){
           return rc;                 // case [D]
     }

     memcpy(buf, fl_rxdata_frm+OFS_STA_PLD, DFV_LEN);// copy version data
     return rc;                       // case [A]
}
```

## 6.14 Checksum Command

### 6.14.1 Processing sequence chart

Checksum command processing sequence



### 6.14.2 Description of processing sequence

<1> Waits from the previous frame reception until the next command transmission (wait time $t_{COM}$).

<2> The Checksum command is transmitted by command frame transmission processing.

<3> A time-out check is performed from command transmission until status frame reception.
If a time-out occurs, a time-out error [C] is returned (time-out time $t_{WT16}$(MAX.)).

<4> The status code is checked.

When ST1 = ACK: Proceeds to <5>.
When ST1 ≠ ACK: Abnormal termination [B]

<5> A time-out check is performed until data frame (checksum data) reception.
If a time-out occurs, a time-out error [C] is returned (time-out time $t_{FD1}$(MAX.)).

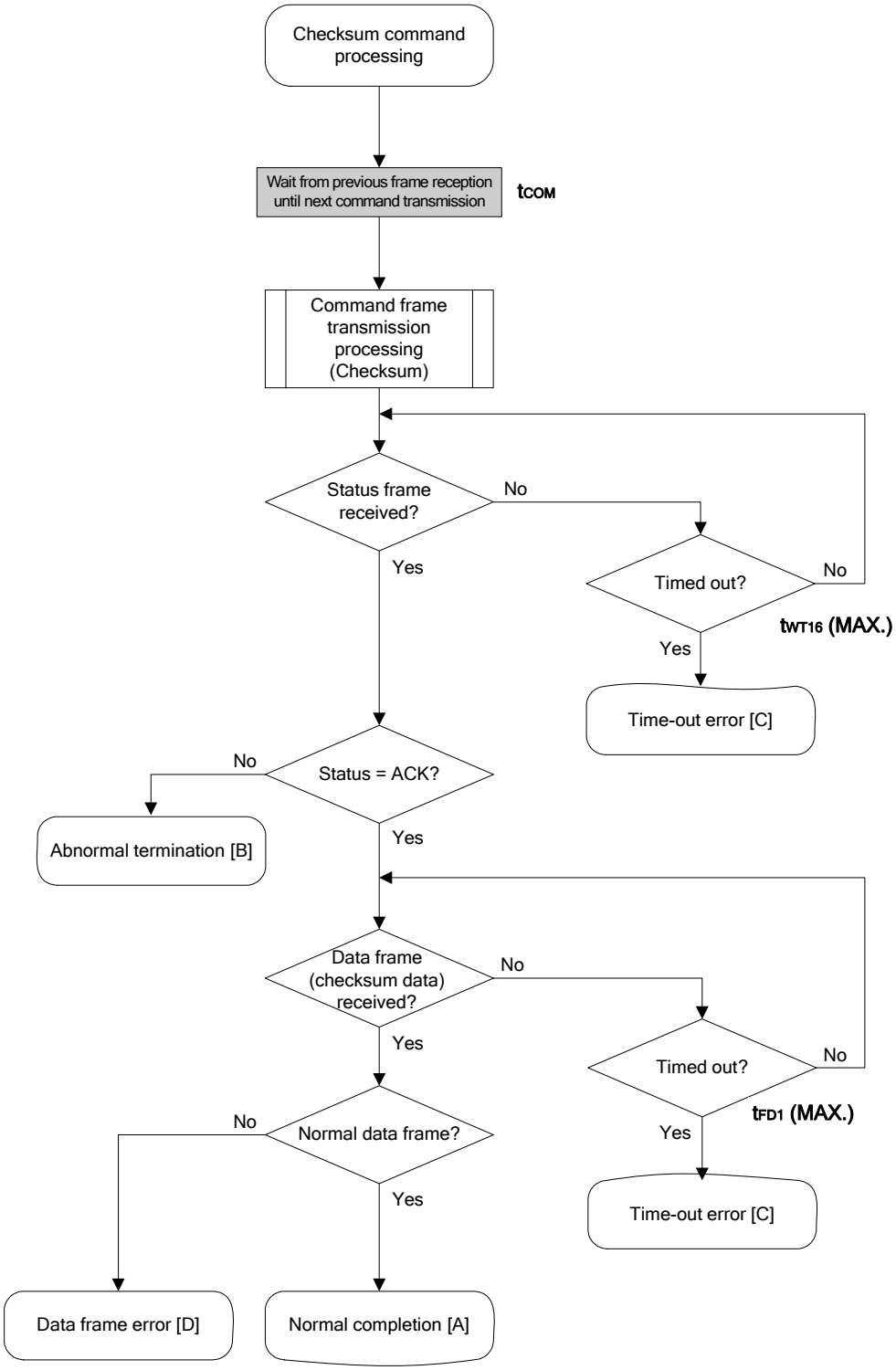
<6> The received data frame (checksum data) is checked.

If data frame is normal: Normal completion [A]
If data frame is abnormal: Data frame error [D]

### 6.14.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and checksum data was acquired normally. |
| Abnormal termination [B] | Parameter error | 05H | The specified start/end address is not the start/end address of the block. |
| | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | The status frame or data frame was not received within the specified time. |
| Data frame error [D] | | – | The checksum of the data frame received as version data does not match. |

### 6.14.4 Flowchart



### 6.14.5 Sample program

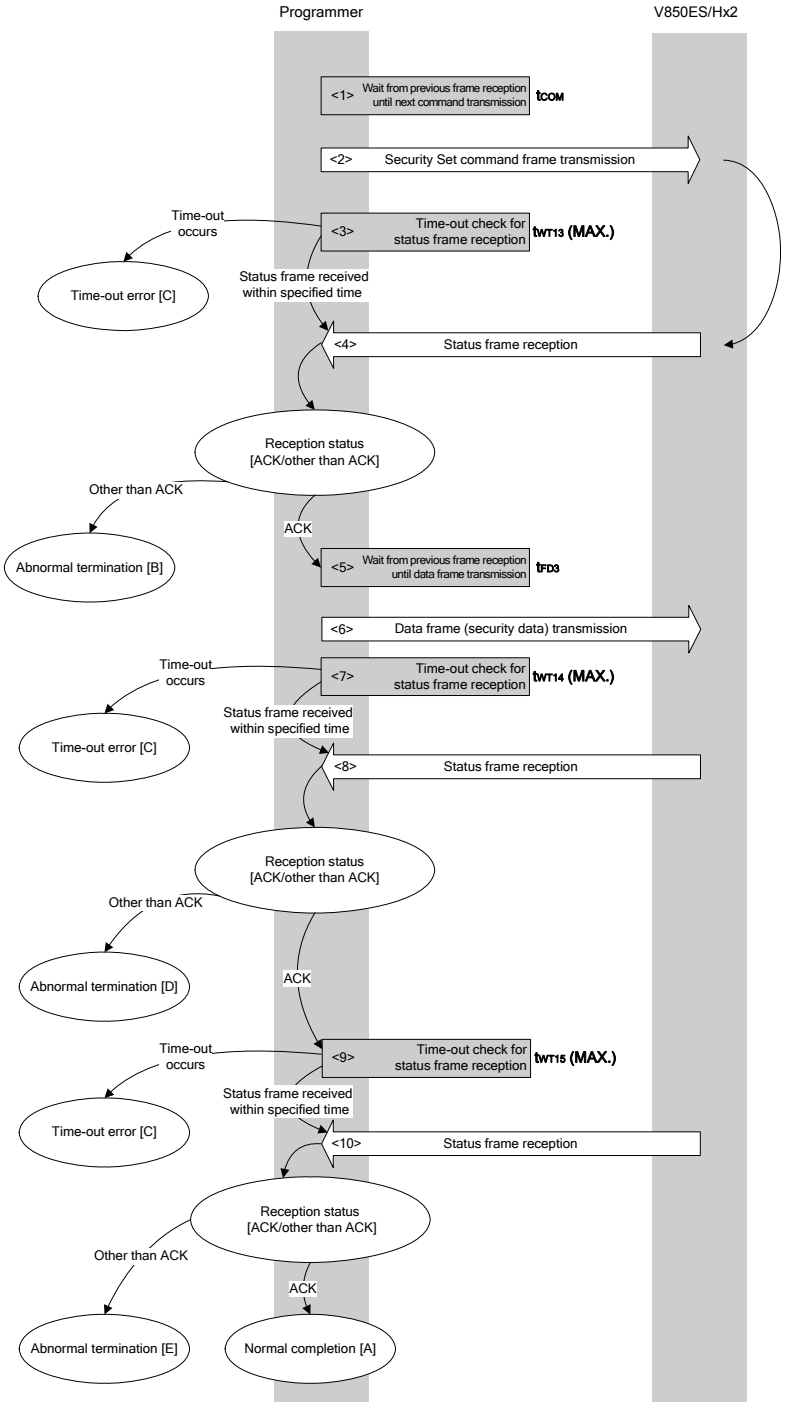
The following shows a sample program for Checksum command processing.

```
/************************************************************/
/*                                                          */
/* Get checksum command                                     */
/*                                                          */
/************************************************************/
/* [i] u16 *sum    ... pointer to checksum save area        */
/* [i] u32 top     ... start address                        */
/* [i] u32 bottom  ... end address                          */
/* [r] u16         ... error code                           */
/************************************************************/
u16         fl_ua_getsum(u16 *sum, u32 top, u32 bottom)
{
     u16    rc;

     /*********************************************/
     /*     set params                            */
     /*********************************************/
     // set params
     set_range_prm(fl_cmd_prm, top, bottom); // set SAH/SAM/SAL, EAH/EAM/EAL


     /*********************************************/
     /*     send command                          */
     /*********************************************/

     fl_wait(tCOM_UA);                // wait before sending command

     put_cmd_ua(FL_COM_GET_CHECK_SUM, 7, fl_cmd_prm); // send GET VERSION command

     rc = get_sfrm_ua(fl_ua_sfrm, tWT16_MAX);       // get status frame
     switch(rc) {
            case   FLC_NO_ERR:                      break; // continue
     //     case   FLC_DFTO_ERR:       return rc;    break; // case [C]
            default:                   return rc;    break; // case [B]
     }

     /*********************************************/
     /*     get data frame (Checksum data)        */
     /*********************************************/
     rc = get_dfrm_ua(fl_rxdata_frm, tFD1_MAX);     // get status frame
     if (rc){                                                // if no error,
            return rc;                          // case [D]
     }

     *sum = (fl_rxdata_frm[OFS_STA_PLD] << 8) + fl_rxdata_frm[OFS_STA_PLD+1];
                                                             // set SUM data
     return rc;                                 // case [A]

}
```

## 6.15 Security Set Command

### 6.15.1 Processing sequence chart

Security Set command processing sequence



### 6.15.2 Description of processing sequence

<1> Waits from the previous frame reception until the next command transmission (wait time $t_{COM}$).
<2> The Security Set command is transmitted by command frame transmission processing.
<3> A time-out check is performed from command transmission until status frame reception.
If a time-out occurs, a time-out error [C] is returned (time-out time $t_{WT13}$ (MAX.)).
<4> The status code is checked.

When ST1 = ACK: Proceeds to <5>.
When ST1 ≠ ACK: Abnormal termination [B]

<5> Waits from the previous frame reception until the next data frame transmission (wait time $t_{FD3}$).
<6> The data frame (security setting data) is transmitted by data frame transmission processing.
<7> A time-out check is performed until status frame reception.
If a time-out occurs, a time-out error [C] is returned (time-out time $t_{WT14}$ (MAX.)).
<8> The status code is checked.

When ST1 = ACK: Proceeds to <9>.
When ST1 ≠ ACK: Abnormal termination [D]

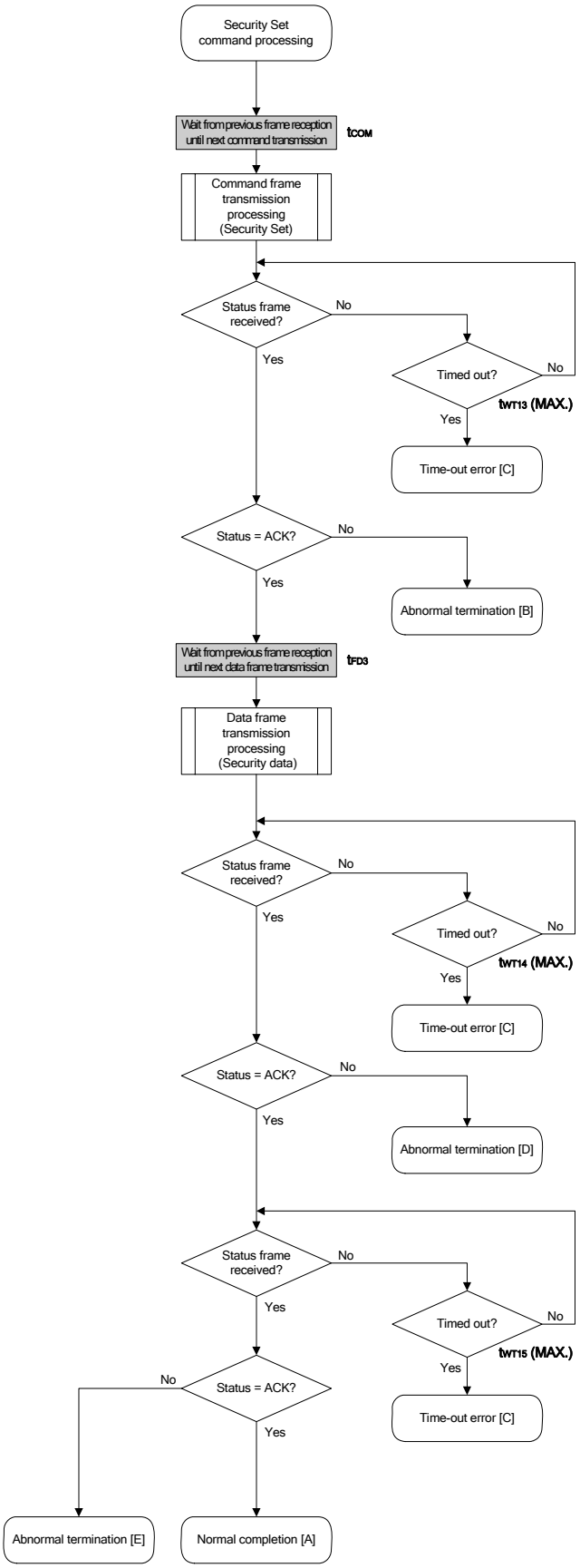
<9> A time-out check is performed until status frame reception.
If a time-out occurs, a time-out error [C] is returned (time-out time $t_{WT15}$ (MAX.)).
<10> The status code is checked.

When ST1 = ACK: Normal completion [A]
When ST1 ≠ ACK: Abnormal termination [E]

### 6.15.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and security setting was performed normally. |
| Abnormal termination [B] | Parameter error | 05H | Command information (parameter) is not 00H. |
| | Checksum error | 07H | The checksum of the transmitted command frame or data frame does not match. |
| | Protect error | 10H | The ID codes do not match. |
| | Negative acknowledgment (NACK) | 15H | Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | The status frame or data frame was not received within the specified time. |
| Abnormal termination [D] | WWV1 error | 08H | • Security data is already set.<br>• A security data write error has occurred. |
| | Sequencer error | 16H | A sequencer error has occurred. |
| Abnormal termination [E] | EWV4 error | 11H | An internal verify error has occurred. |
| | Sequencer error | 16H | A sequencer error has occurred. |

### 6.15.4 Flowchart



### 6.15.5 Sample program

The following shows a sample program for Security Set command processing.

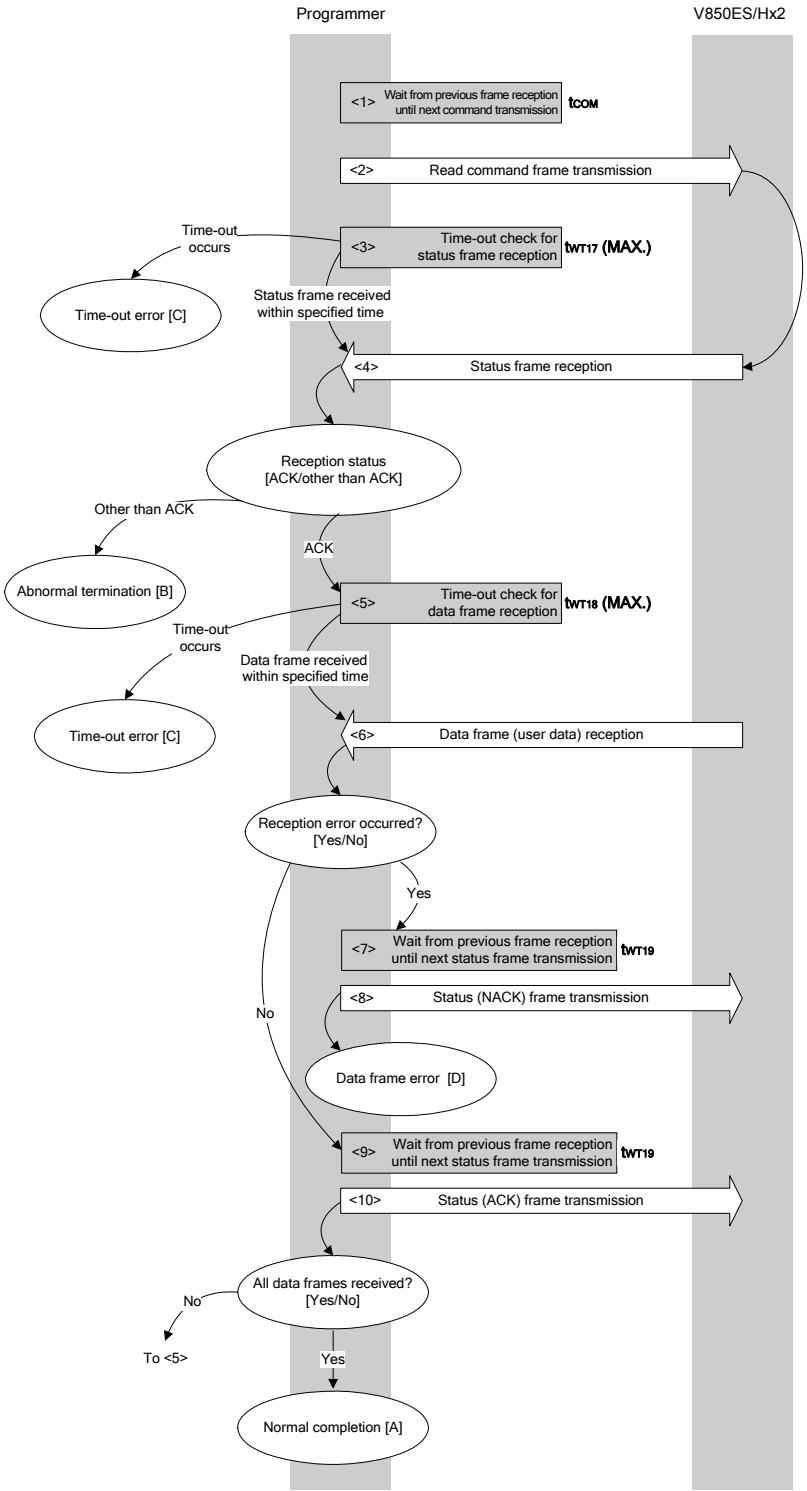
```
/************************************************************/
/*                                                          */
/*       Set security flag command                          */
/*                                                          */
/************************************************************/
/*       [i] u8 scf    ... Security flag data               */
/*       [r] u16             ... error code                 */
/************************************************************/
u16     fl_ua_setscf(u8 scf, u32 vect)
{
  u16    rc;

  /*********************************************/
  /*     set params                            */
  /*********************************************/
  fl_cmd_prm[0] = 0x00;                    // "BLK" (must be 0x00)
  fl_cmd_prm[1] = 0x00;                    // "PAG" (must be 0x00)

  fl_txdata_frm[0] = (scf|= 0b11110000);    // "FLG" (upper 4bits must be '1' (to make
sure))
  fl_txdata_frm[1] = (u8)(vect >> 16);     // "ADH"
  fl_txdata_frm[2] = (u8)(vect >>  8);     // "ADM"
  fl_txdata_frm[3] = (u8) vect;            // "ADL"


  /*********************************************/
  /*     send command                          */
  /*********************************************/
  fl_wait(tCOM_UA);          // wait before sending command

  put_cmd_ua(FL_COM_SET_SECURITY, 3, fl_cmd_prm);

  rc = get_sfrm_ua(fl_ua_sfrm, tWT13_MAX);       // get status frame
  switch(rc) {
        case   FLC_NO_ERR:               break; // continue
  //    case   FLC_DFTO_ERR: return rc;  break; // case [C]
        default:             return rc;  break; // case [B]
  }


  /*********************************************/
  /*     send data frame (security setting data)  */
  /*********************************************/

  fl_wait(tFD3);
  put_dfrm_ua(4, fl_txdata_frm, true);    // send securithi setting data

  rc = get_sfrm_ua(fl_ua_sfrm, tWT14_MAX);       // get status frame
  switch(rc) {
        case   FLC_NO_ERR:               break; // continue
  //    case   FLC_DFTO_ERR: return rc;  break; // case [C]
        default:             return rc;  break; // case [B]
  }

  /*********************************************/
  /*     Check internally verify               */
  /*********************************************/
  rc = get_sfrm_ua(fl_ua_sfrm, tWT15_MAX);       // get status frame
//      switch(rc) {
//
//            case   FLC_NO_ERR:   return rc;    break; // case [A]
//            case   FLC_DFTO_ERR: return rc;    break; // case [C]
//            default:             return rc;    break; // case [B]
```

```
    //      }
      return rc;
}
```

## 6.16 Read Command

### 6.16.1 Processing sequence chart

Read command processing sequence



### 6.16.2 Description of processing sequence

<1> Waits from the previous frame reception until the next command transmission (wait time $t_{COM}$).
<2> The Read command is transmitted by command frame transmission processing.
<3> A time-out check is performed from command transmission until status frame reception.
If a time-out occurs, a time-out error [C] is returned (time-out time $t_{WT17}$(MAX.)).
<4> The status code is checked.

When ST1 = ACK: Proceeds to <5>.
When ST1 ≠ ACK: Abnormal termination [B]

<5> A time-out check is performed until reception of the data frame reception result (user data).
If a time-out occurs, a time-out error [C] is returned (time-out time $t_{WT19}$(MAX.)).
<6> The received data frame is checked.

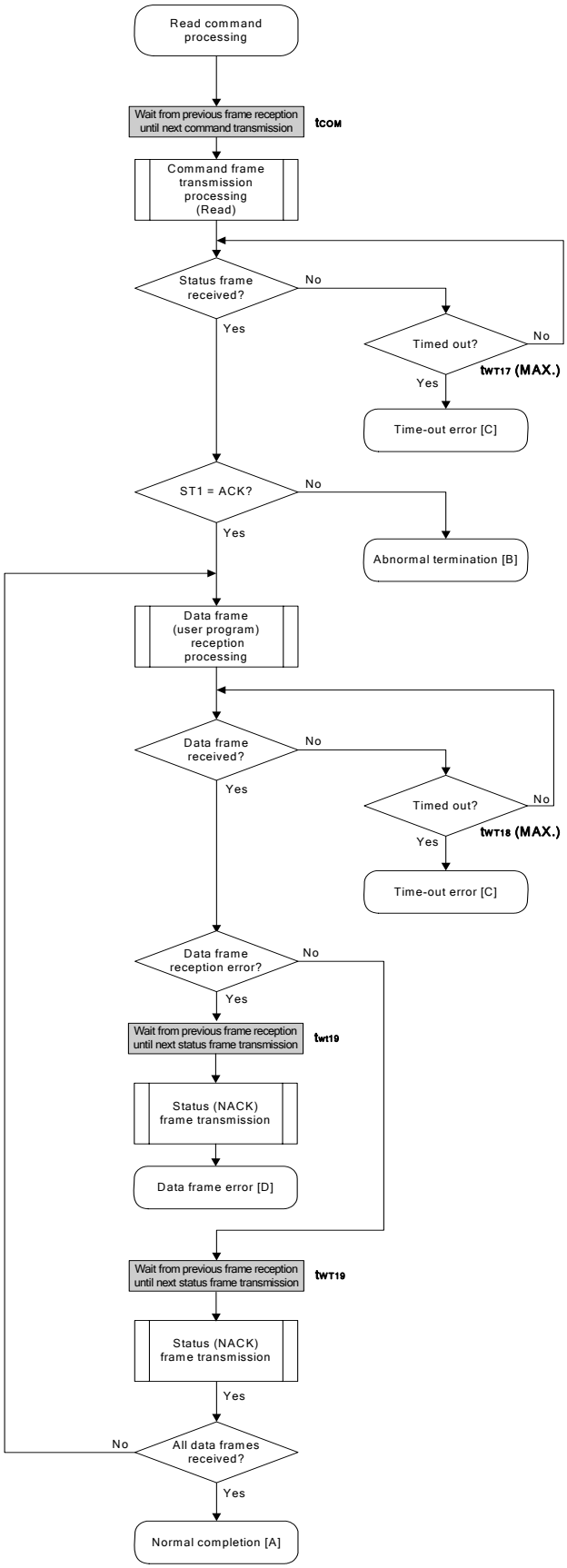
If data frame is normal: Proceeds to <9>.
If data frame is abnormal: Proceeds to <7>.

<7> Waits from the previous frame reception until the next status (NACK) frame transmission (wait time $t_{WT19}$).
<8> The NACK frame is transmitted by data frame transmission processing.
→ A data frame error [D] is returned.
<9> Waits from the previous frame reception until the next status (ACK) frame transmission (wait time $t_{WT19}$).
<10> The ACK frame is transmitted by data frame transmission processing.
When reception of all data frames is completed, normal completion [A] is returned.
If there still remain data frames to be received, the processing re-executes the sequence from <5>.

### 6.16.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and read data was set normally. |
| Abnormal termination [B] | Parameter error | 05H | The specified start/end address is not the start/end address of the block. |
| | Checksum error | 07H | The checksum of the transmitted command frame or data frame does not match. |
| | Protect error | 10H | Read is prohibited in the security setting. |
| | Negative acknowledgment (NACK) | 15H | Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | The status frame or data frame was not received within the specified time. |
| Data frame error [D] | | – | The checksum of the data frame received as read data does not match. |

### 6.16.4 Flowchart



### 6.16.5 Sample program

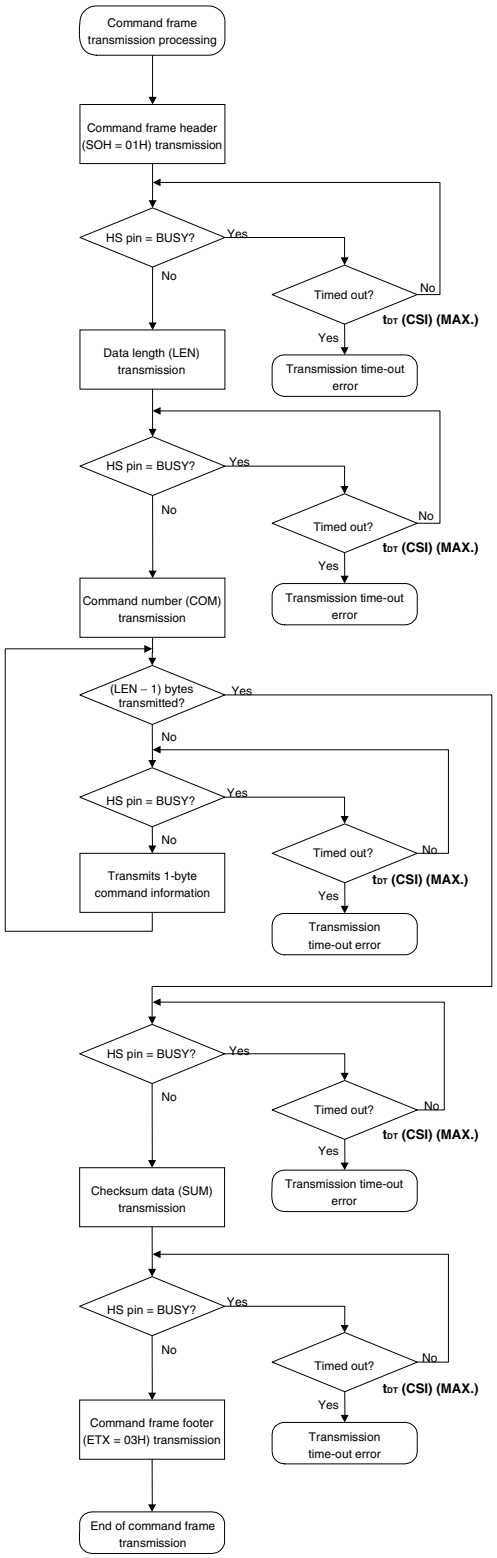
The following shows a sample program for Read command processing.

```
/*************************************************************/
/*                                                           */
/*       Read command                                        */
/*                                                           */
/*************************************************************/
u16      fl_ua_read(u32 top, u32 bottom)
{
  u16    rc;
  u32    read_head;
  u16    len;
  u8     hooter;

  /*********************************************/
  /*     set params                             */
  /*********************************************/
  set_range_prm(fl_cmd_prm, top, bottom); // set SAH/SAM/SAL, EAH/EAM/EAL


  /*********************************************/
  /*     send command & check status            */
  /*********************************************/
  fl_wait(tCOM_UA);   // wait before sending command

  put_cmd_ua(FL_COM_READ, 7, fl_cmd_prm);

  rc = get_sfrm_ua(fl_ua_sfrm, tWT17_MAX);       // get status frame
  switch(rc) {
        case   FLC_NO_ERR:               break;
  //    case   FLC_DFTO_ERR: return rc;  break; // case [C]
        default:             return rc;  break; // case [B]
  }

  /*********************************************/
  /*     receive user data                      */
  /*********************************************/
  read_head = top;

  while(1){

        rc = get_dfrm_ua(fl_rxdata_frm, tWT18_MAX);    // get ROM data from FLASH

        switch(rc) {
              case   FLC_NO_ERR:               break; // continue
              case   FLC_DFTO_ERR: return rc;  break; // case [C]
        //    case   FLC_RX_DFSUM_ERR:
              default:                                 // case [B]

                    fl_wait(tWT19);
                    put_sfrm_ua(FLST_NACK);            // send status(NACK) frame
                    return rc;
                    break;

        }
```

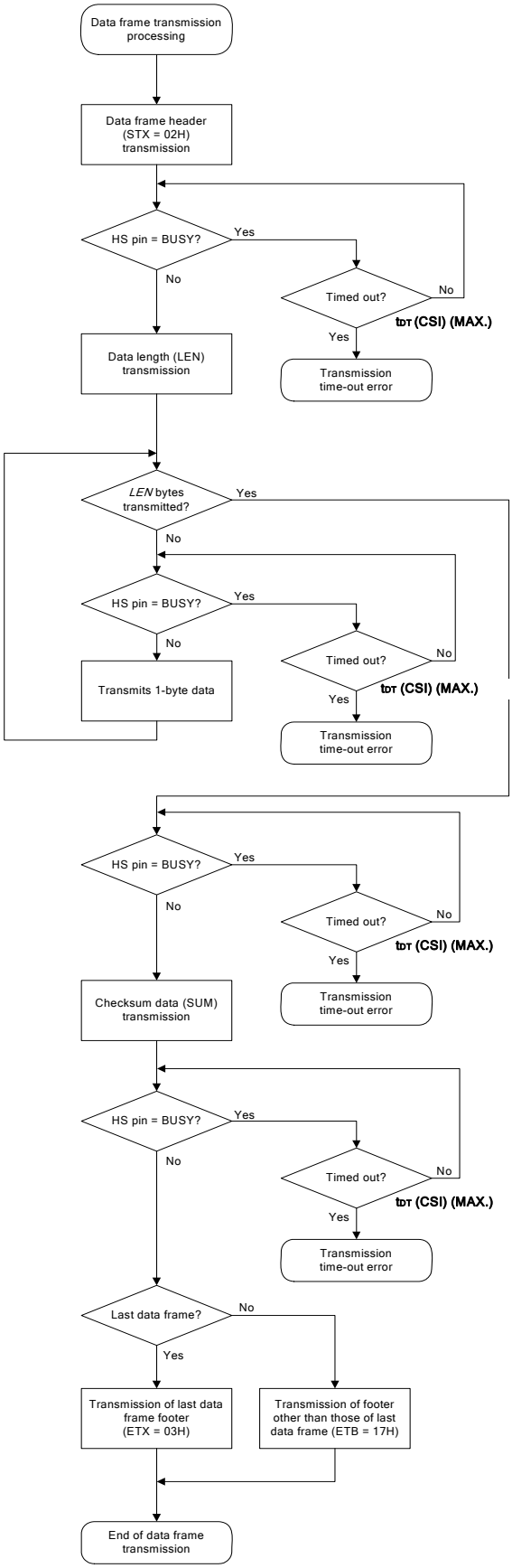
```
        fl_wait(tWT19);
        put_sfrm_ua(FLST_ACK);

        /*********************************************/
        /*     save ROM data                         */
        /*********************************************/
        if ((len = fl_rxdata_frm[OFS_LEN]) == 0)       // get length
              len = 256;

        memcpy(read_buf+read_head, fl_rxdata_frm+2, len); // save to external RAM

        read_head += len;

        /*********************************************/
        /*     end check                             */
        /*********************************************/
        hooter = fl_rxdata_frm[len + 3];
        if (hooter == FL_ETB)                    // end frame ?
              continue;                  // no
        break;                           // yes
  }

  return FLC_NO_ERR;

}
```

# CHAPTER 7 3-WIRE SERIAL I/O COMMUNICATION MODE WITH HANDSHAKE SUPPORTED (CSI + HS)

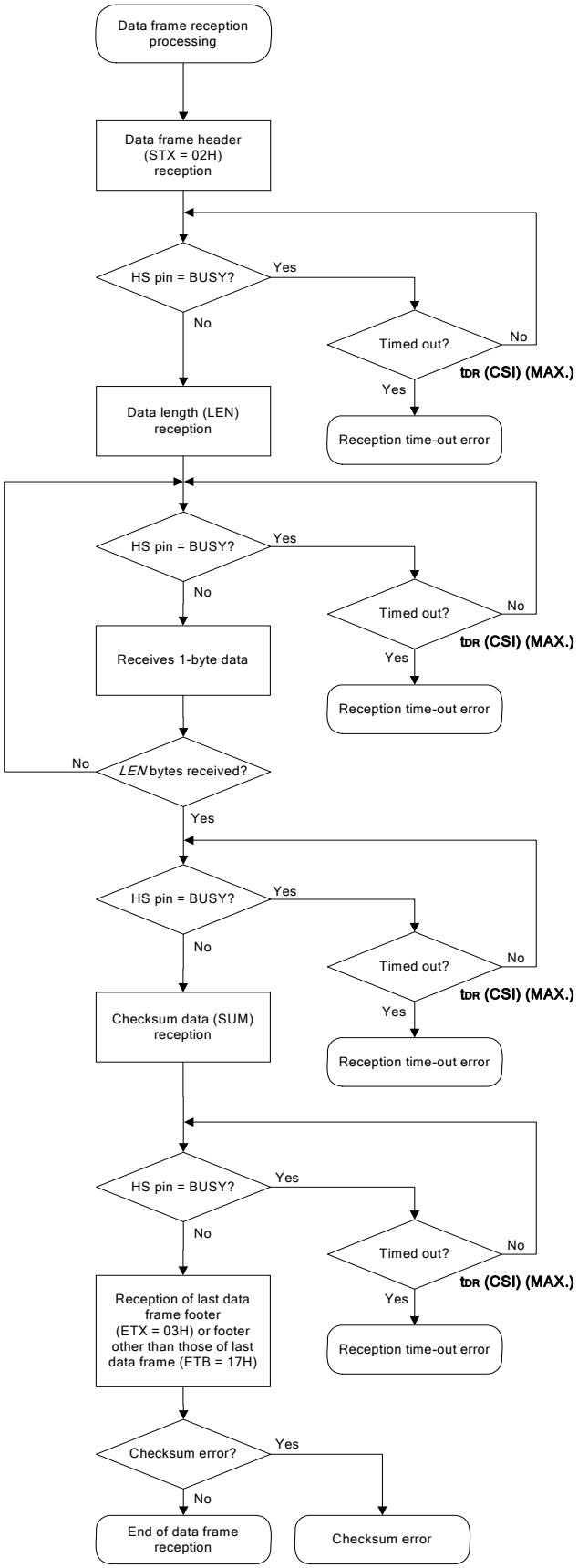
## 7.1 Command Frame Transmission Processing Flowchart



## 7.2 Data Frame Transmission Processing Flowchart



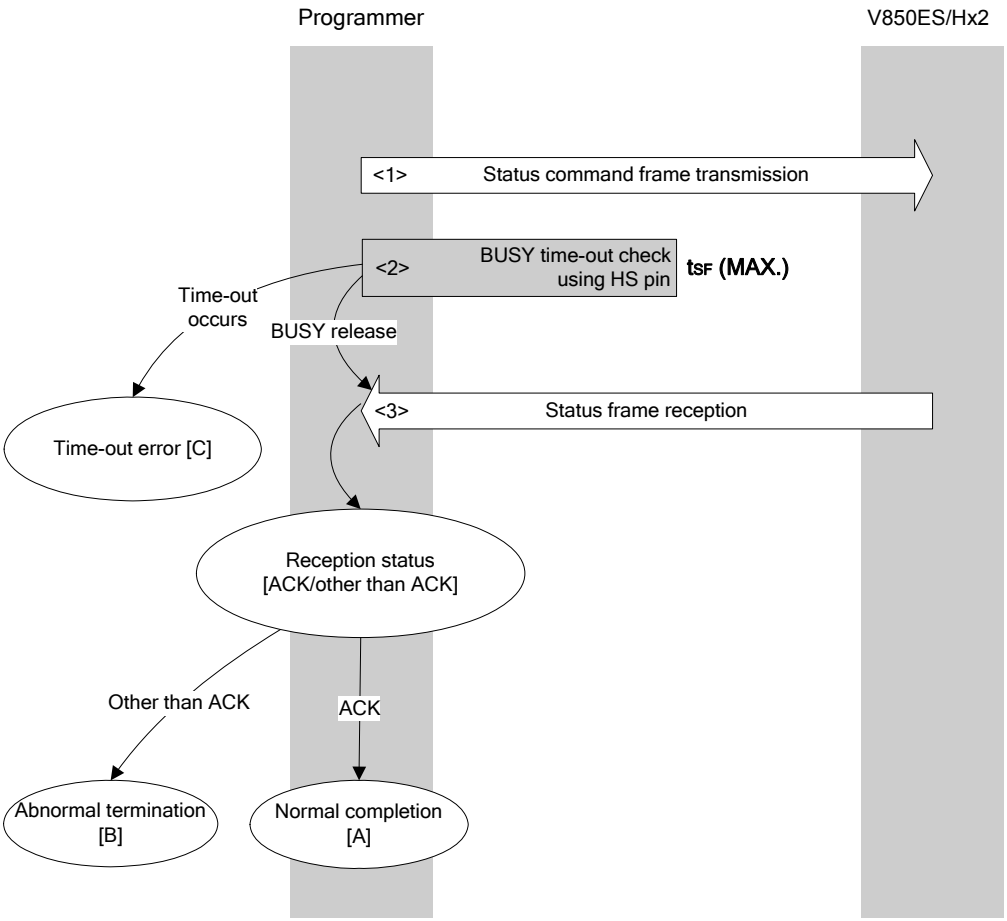
## 7.3 Data Frame Reception Processing Flowchart



## 7.4 Status Command

### 7.4.1 Processing sequence chart

Status command processing sequence



### 7.4.2 Description of processing sequence

<1> The Status command is transmitted by command frame transmission processing.

<2> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{SF(MAX.)}$).

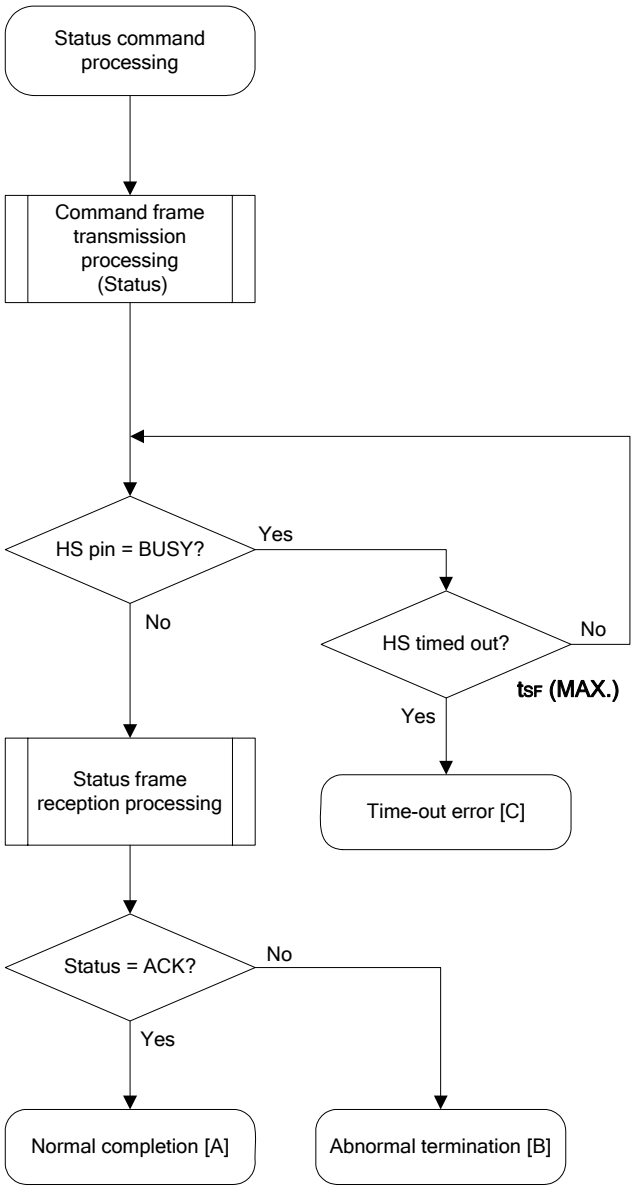
<3> The status code is checked.

When ST1 = ACK: Normal completion [A]
When ST1 ≠ ACK: Abnormal termination [B]

### 7.4.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The status frame transmitted from the V850ES/Hx2 has been received normally. |
| Abnormal termination [B] | Command error | 04H | An unsupported command or abnormal frame has been received. |
| | Parameter error | 05H | Command information (parameter) is invalid. |
| | Checksum error | 07H | The data of the frame transmitted from the programmer is abnormal. |
| | WWV1 error | 08H | Write error |
| | EWV1 error | 0BH | Erase error |
| | EWV2 error | 0CH | |
| | EWV3 error | 0DH | |
| | Verify error | 0EH | A verify error has occurred for the data of the frame transmitted from the programmer. |
| | | 0FH | |
| | Protect error | 10H | An attempt was made to execute processing prohibited by the Security Set command. |
| | EWV4 error | 11H | Internal verify error/blank error |
| | Compaction search error | 13H | Erase error |
| | Negative acknowledgment (NACK) | 15H | Negative acknowledgment |
| | Sequencer error | 16H | A sequencer error has occurred. |
| Time-out error [C] | | – | Processing timed out due to the busy status at the HS pin. |

### 7.4.4 Flowchart



### 7.4.5 Sample program

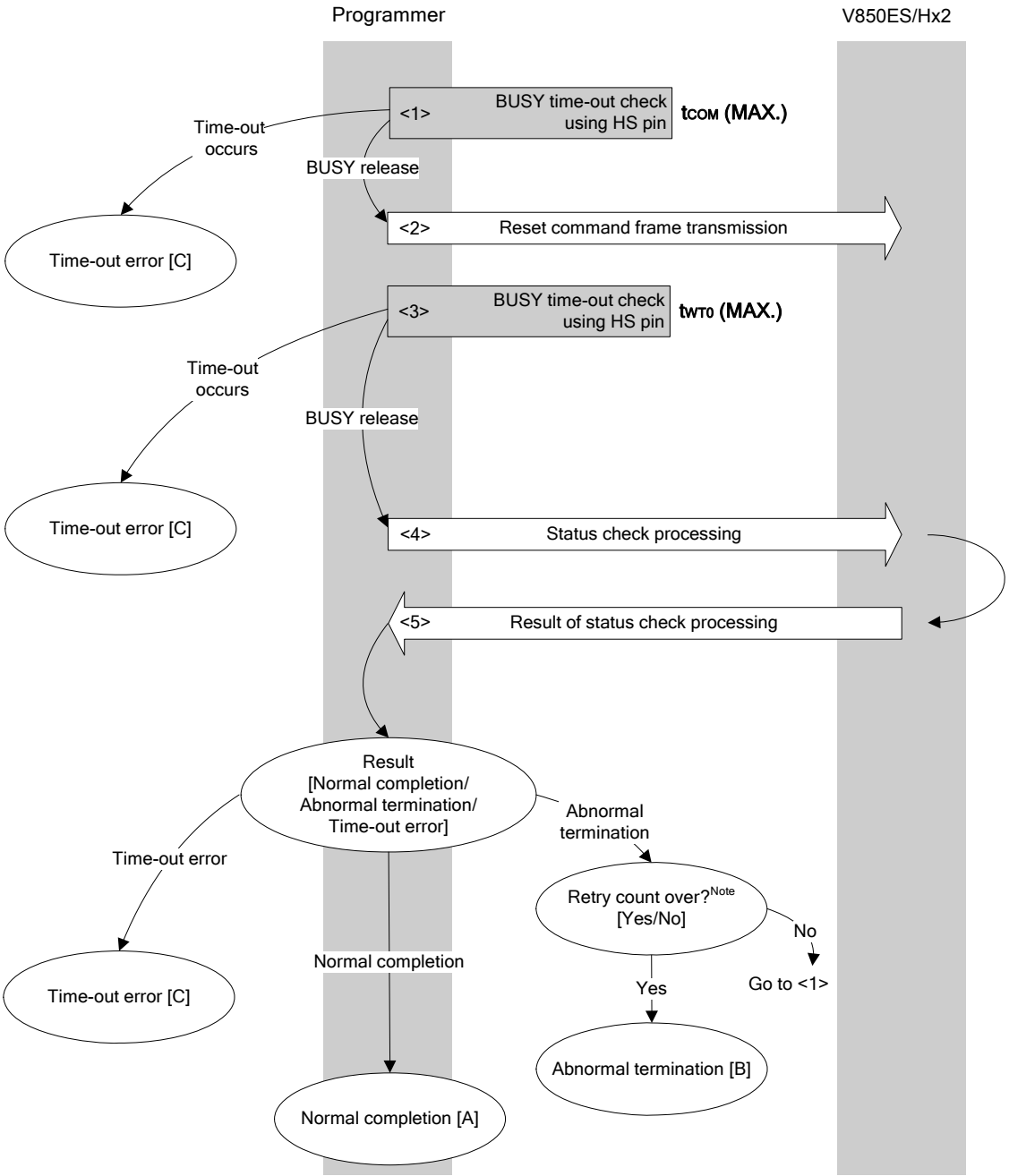
The following shows a sample program for Status command processing.

```
/*************************************************************/
/*                                                           */
/*  Get status command (CSI-HS)                             */
/*                                                           */
/*************************************************************/
/*  [r] u16       ... decoded status or error code           */
/*                                                           */
/*  (see fl.h/fl-proto.h &                                   */
/*           definition of decode_status() in fl.c)          */
/*************************************************************/
static u16 fl_hs_getstatus(void)
{
     u16    rc;
     u32    retry = 0;



     rc = put_cmd_hs(FL_COM_GET_STA, 1, fl_cmd_prm);      // send "Status" command
     if (rc)
            return rc;    // case [C]

     if (hs_busy_to(tSF_MAX))          // HS-Busy t.o. ?
            return FLC_HSTO_ERR;       // t.o. detected  : case [C]

     if (rc = get_sfrm_hs(fl_rxdata_frm))
            return rc;           // case [C] or [B(checksum error)]


     rc = decode_status(fl_st1);      // decode return code

     return rc;          // case [A] or [B]
}
```

## 7.5 Reset Command

### 7.5.1 Processing sequence chart

Reset command processing sequence




**Note** Do not exceed the retry count for the reset command transmission (up to 16 times).

### 7.5.2 Description of processing sequence

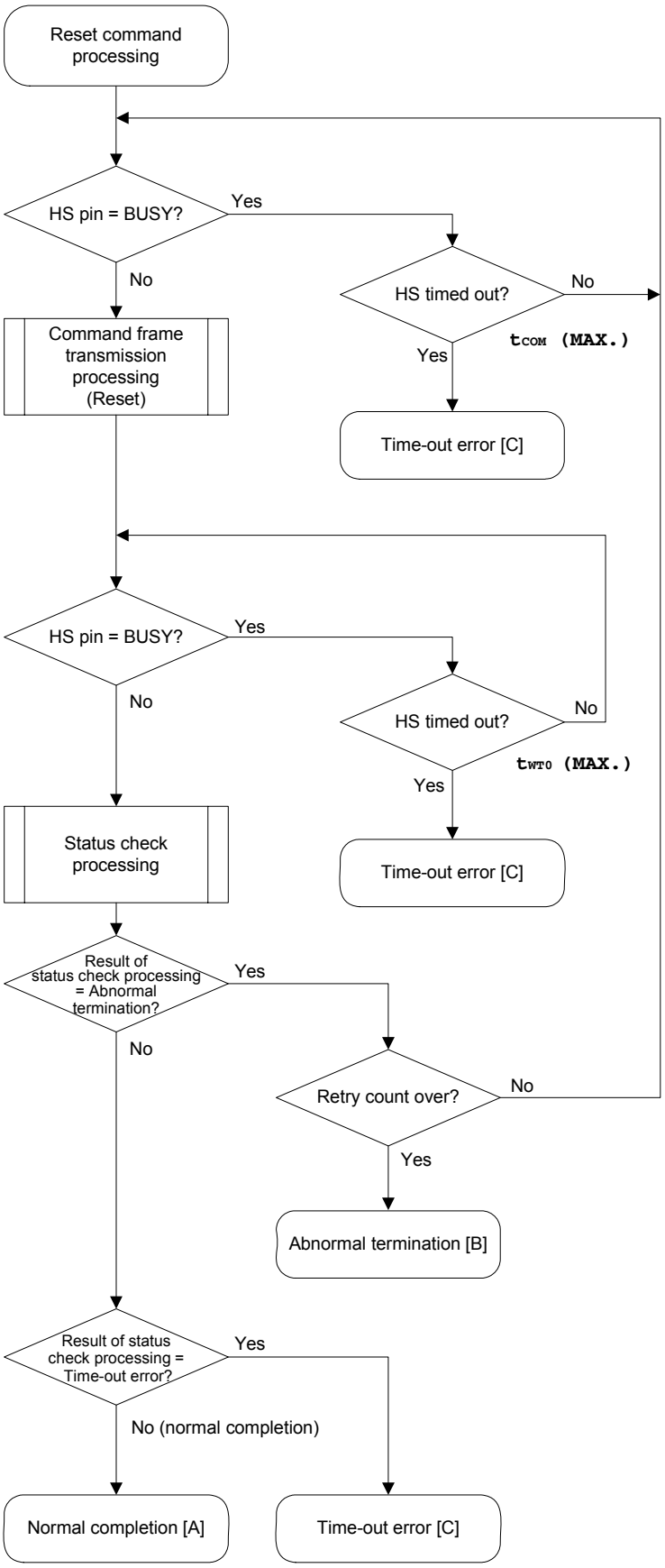
<1> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{COM(MAX.)}$).
<2> The Reset command is transmitted by command frame transmission processing.
<3> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{WT0(MAX.)}$).
<4> The status frame is acquired by status check processing.
<5> The following processing is performed according to the result of status check processing.

When the processing ends normally: Normal completion [A]
When the processing ends abnormally: The sequence is re-executed from <1> if the retry count is not over.
If the retry count is over, the processing ends abnormally [B].
When a time-out error occurs: A time-out error [C] is returned.

### 7.5.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and synchronization between the programmer and the V850ES/Hx2 has been established. |
| Abnormal termination [B] | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | Status check processing timed out.<br>Processing timed out due to the busy status at the HS pin. |

### 7.5.4 Flowchart



### 7.5.5 Sample program

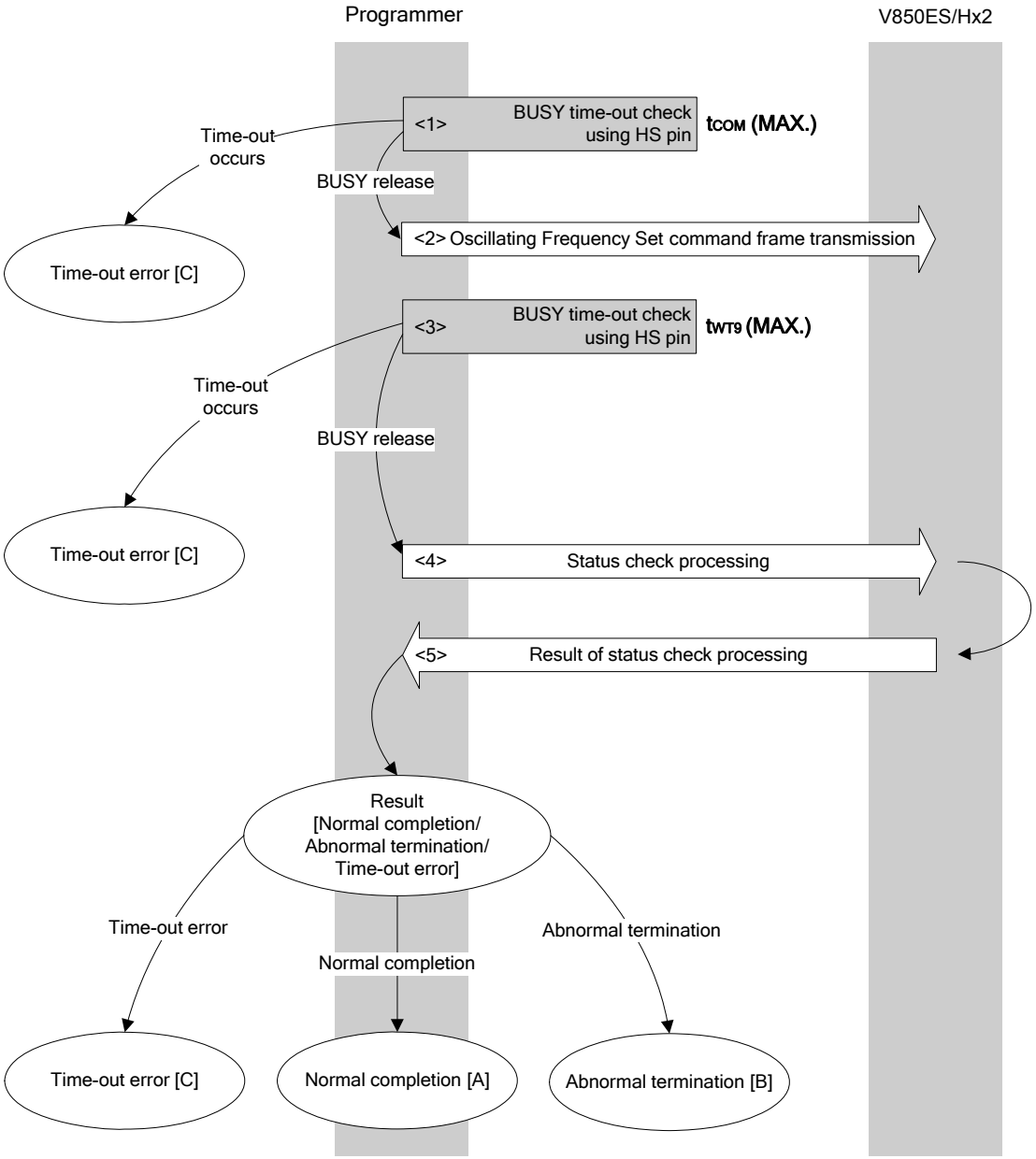
The following shows a sample program for Reset command processing.

```
/************************************************************/
/*                                                          */
/* Reset command (CSI-HS)                                   */
/*                                                          */
/************************************************************/
/* [r] u16             ... error code                       */
/************************************************************/
u16         fl_hs_reset(void)
{
     u16    rc;
     u32    retry;

     for (retry = 0; retry < tRS; retry++){

            if (hs_busy_to(tCOM_MAX))
                   return FLC_HSTO_ERR;              // t.o. detected :case [C]

            rc = put_cmd_hs(FL_COM_RESET, 1, fl_cmd_prm);  // send "Reset" command
            if (rc)
                   return rc;          // case [C]

            if (hs_busy_to(tWT0_MAX))
                   return FLC_HSTO_ERR;              // t.o. detected :case [C]

            rc = fl_hs_getstatus();    // get status frame
            if (rc == FLC_ACK)         // ST1 = ACK ?
                   break;              // case [A]
            //continue;                // case [B] (if exit from loop)

     }
//   switch(rc) {
//          case   FLC_NO_ERR:   return rc;    break; // case [A]
//          case   FLC_HSTO_ERR: return rc;    break; // case [C]
//          default:             return rc;    break; // case [B]
//   }
     return rc;
}
```

## 7.6 Oscillating Frequency Set Command

### 7.6.1 Processing sequence chart

Oscillating Frequency Set command processing sequence



### 7.6.2 Description of processing sequence

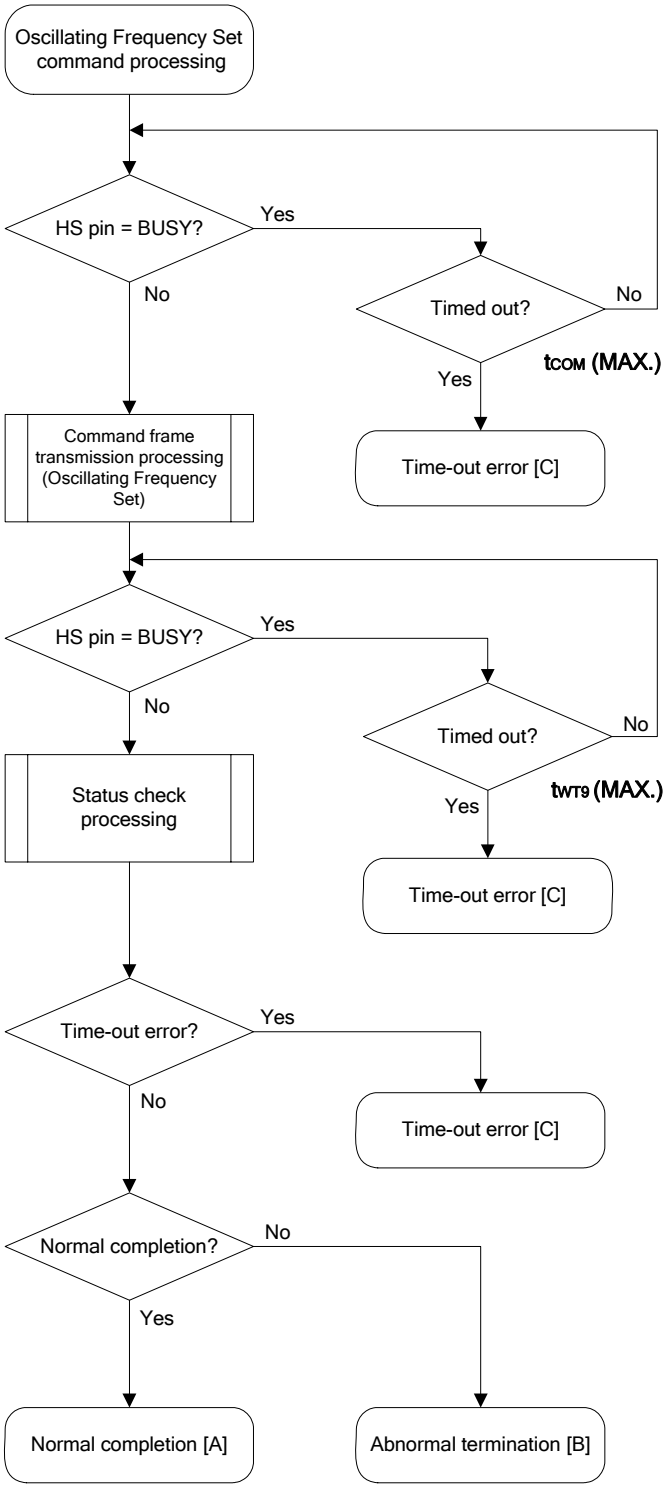
<1> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{COM(MAX.)}$).
<2> The Oscillating Frequency Set command is transmitted by command frame transmission processing.
<3> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{WT9(MAX.)}$).
<4> The status frame is acquired by status check processing.
<5> The following processing is performed according to the result of status check processing.

When the processing ends normally: Normal completion [A]
When the processing ends abnormally: Abnormal termination [B]
When a time-out error occurs: A time-out error [C] is returned.

### 7.6.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and the operating frequency was correctly set to the V850ES/Hx2. |
| Abnormal termination [B] | Parameter error | 05H | The oscillation frequency value is out of range. |
| | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | Processing timed out due to the busy status at the HS pin. |

### 7.6.4 Flowchart



### 7.6.5 Sample program

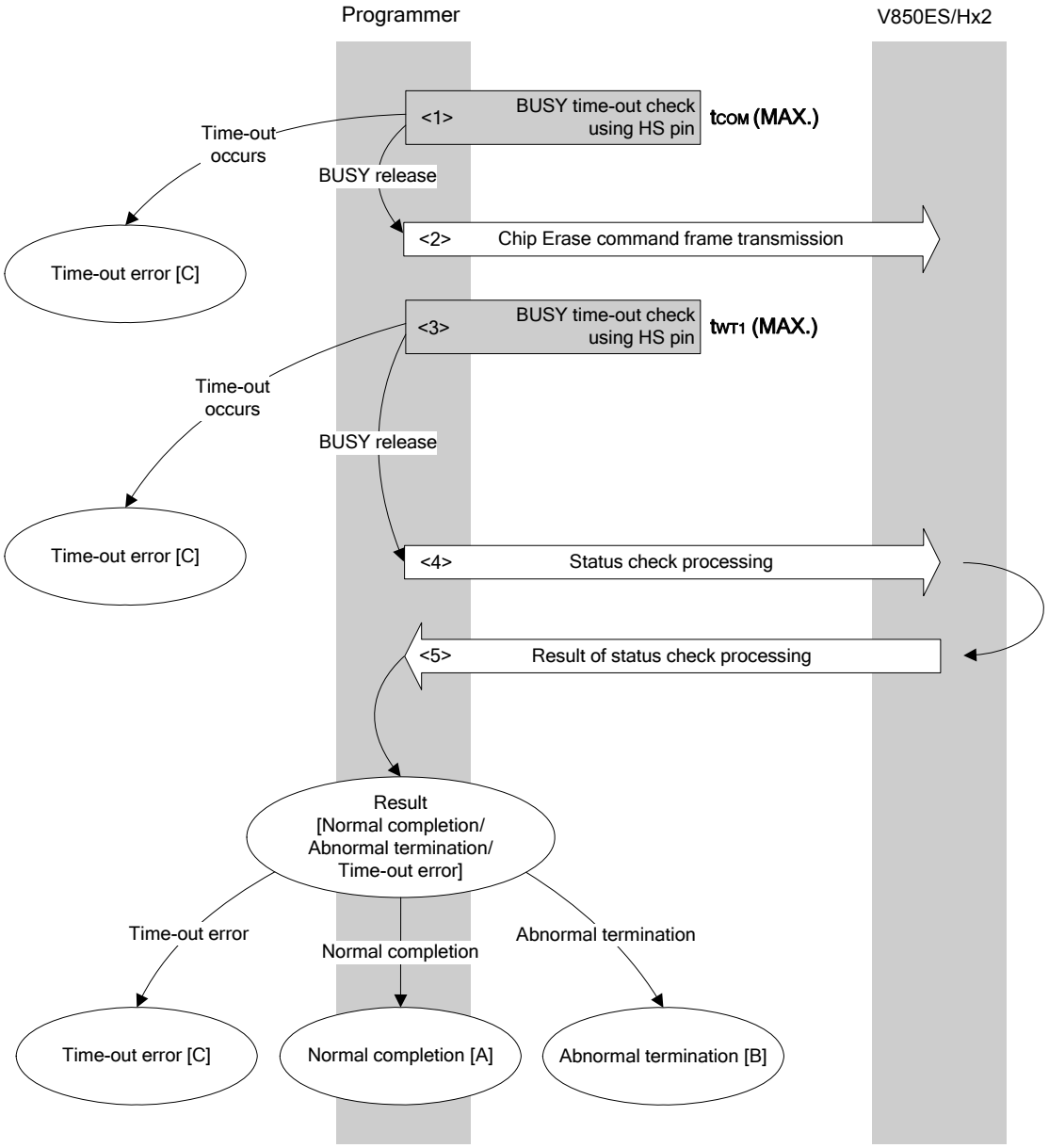
The following shows a sample program for Oscillating Frequency Set command processing.

```
/************************************************************/
/*                                                          */
/* Set Flash device clock value command (CSI-HS)            */
/*                                                          */
/************************************************************/
/* [i] u8 clk[4]  ... frequency data(D1-D4)                 */
/* [r] u16        ... error code                            */
/************************************************************/
/************************************************************/
u16        fl_hs_setclk(u8 clk[])
{
     u16    rc;

     fl_cmd_prm[0] = clk[0];    // "D01"
     fl_cmd_prm[1] = clk[1];    // "D02"
     fl_cmd_prm[2] = clk[2];    // "D03"
     fl_cmd_prm[3] = clk[3];    // "D04"


     if (hs_busy_to(tCOM_MAX))
            return FLC_HSTO_ERR;       // t.o. detected :case [C]

     if (rc = put_cmd_hs(FL_COM_SET_OSC_FREQ, 5, fl_cmd_prm))
                                        // send "Oscilating Frequency Set" command
            return rc;                  // case [C]

     if (hs_busy_to(tWT9_MAX))
            return FLC_HSTO_ERR;       // t.o. detected :case [C]


     rc = fl_hs_getstatus();           // get status frame
//   switch(rc) {
//          case   FLC_NO_ERR:   return rc;    break; // case [A]
//          case   FLC_HSTO_ERR: return rc;    break; // case [C]
//          default:             return rc;    break; // case [B]
//   }
     return rc;
}
```

## 7.7 Chip Erase Command

### 7.7.1 Processing sequence chart

Chip Erase command processing sequence



### 7.7.2 Description of processing sequence

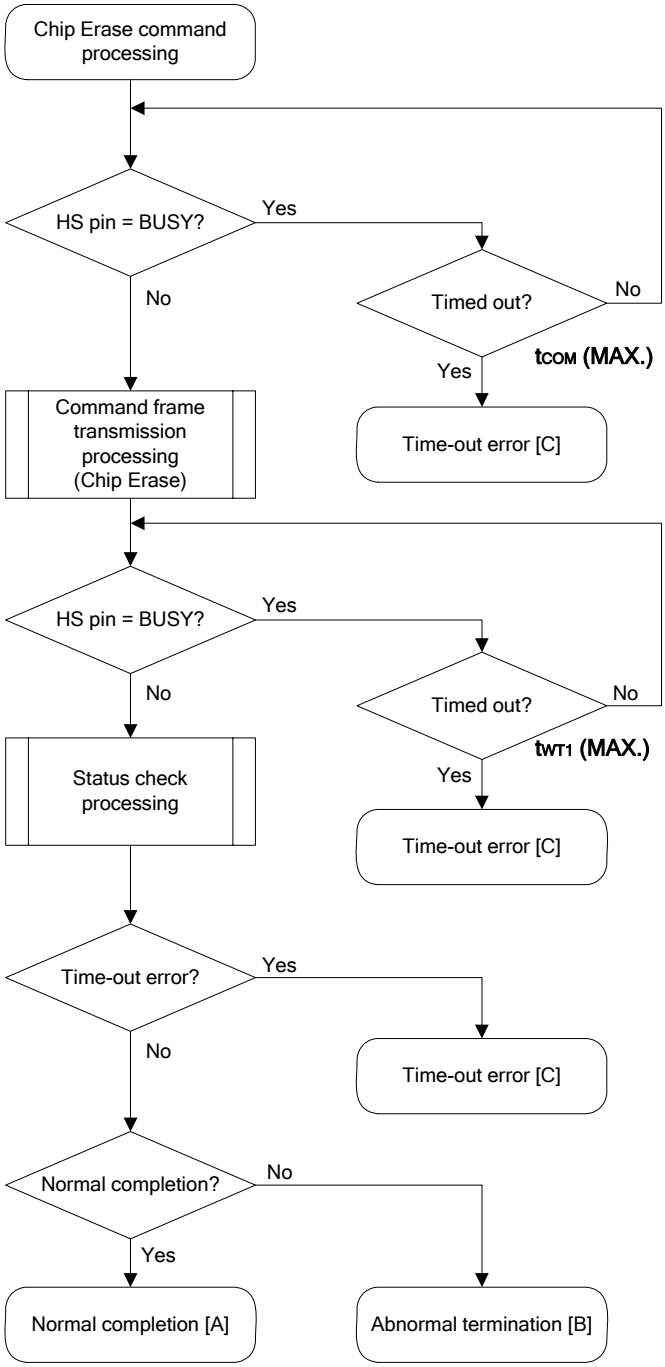
<1> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{COM(MAX.)}$).
<2> The Chip Erase command is transmitted by command frame transmission processing.
<3> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{WT1(MAX.)}$).
<4> The status frame is acquired by status check processing.
<5> The following processing is performed according to the result of status check processing.

When the processing ends normally: Normal completion [A]
When the processing ends abnormally: Abnormal termination [B]
When a time-out error occurs: A time-out error [C] is returned.

### 7.7.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and chip erase was performed normally. |
| Abnormal termination [B] | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Protect error | 10H | Chip erase is prohibited in the security setting. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| | WWV1 error | 08H | An erase error has occurred. |
| | EWV1 error | 0BH | |
| | EWV2 error | 0CH | |
| | EWV3 error | 0DH | |
| | Compaction search error | 13H | |
| | Sequencer error | 16H | A sequencer error has occurred. |
| Time-out error [C] | | – | Processing timed out due to the busy status at the HS pin. |

### 7.7.4 Flowchart



### 7.7.5 Sample program

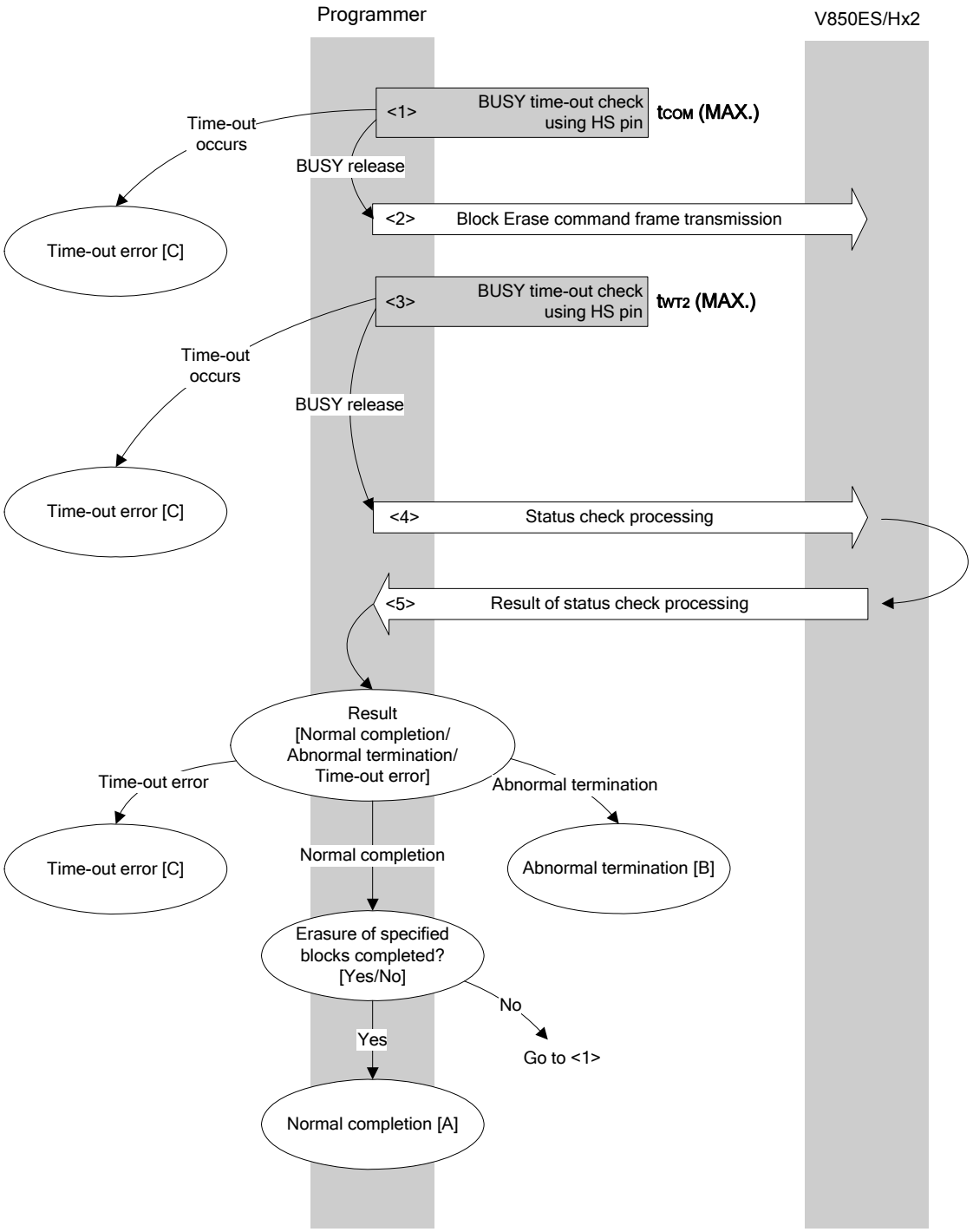
The following shows a sample program for Chip Erase command processing.

```
/*****************************************************************/
/*                                                               */
/* Erase all(chip) command (CSI-HS)                              */
/*                                                               */
/*****************************************************************/
/* [r] u16         ... error code                                */
/*****************************************************************/
u16         fl_hs_erase_all(void)
{
     u16    rc;

     if (hs_busy_to(tCOM_MAX))
            return FLC_HSTO_ERR;              // t.o. detected

     if (rc = put_cmd_hs(FL_COM_ERASE_CHIP, 1, fl_cmd_prm))
                                              // send "Chip Erase" command
            return rc;                        // case [C]

     if (hs_busy_to(tWT1_MAX))
            return FLC_HSTO_ERR;              // case [C]

     rc = fl_hs_getstatus();                  // get status frame
//   switch(rc) {
//          case   FLC_NO_ERR:   return rc;    break; // case [A]
//          case   FLC_HSTO_ERR: return rc;    break; // case [C]
//          default:             return rc;    break; // case [B]
//   }
     return rc;
}
```

## 7.8 Block Erase Command

### 7.8.1 Processing sequence chart

Block Erase command processing sequence



### 7.8.2 Description of processing sequence

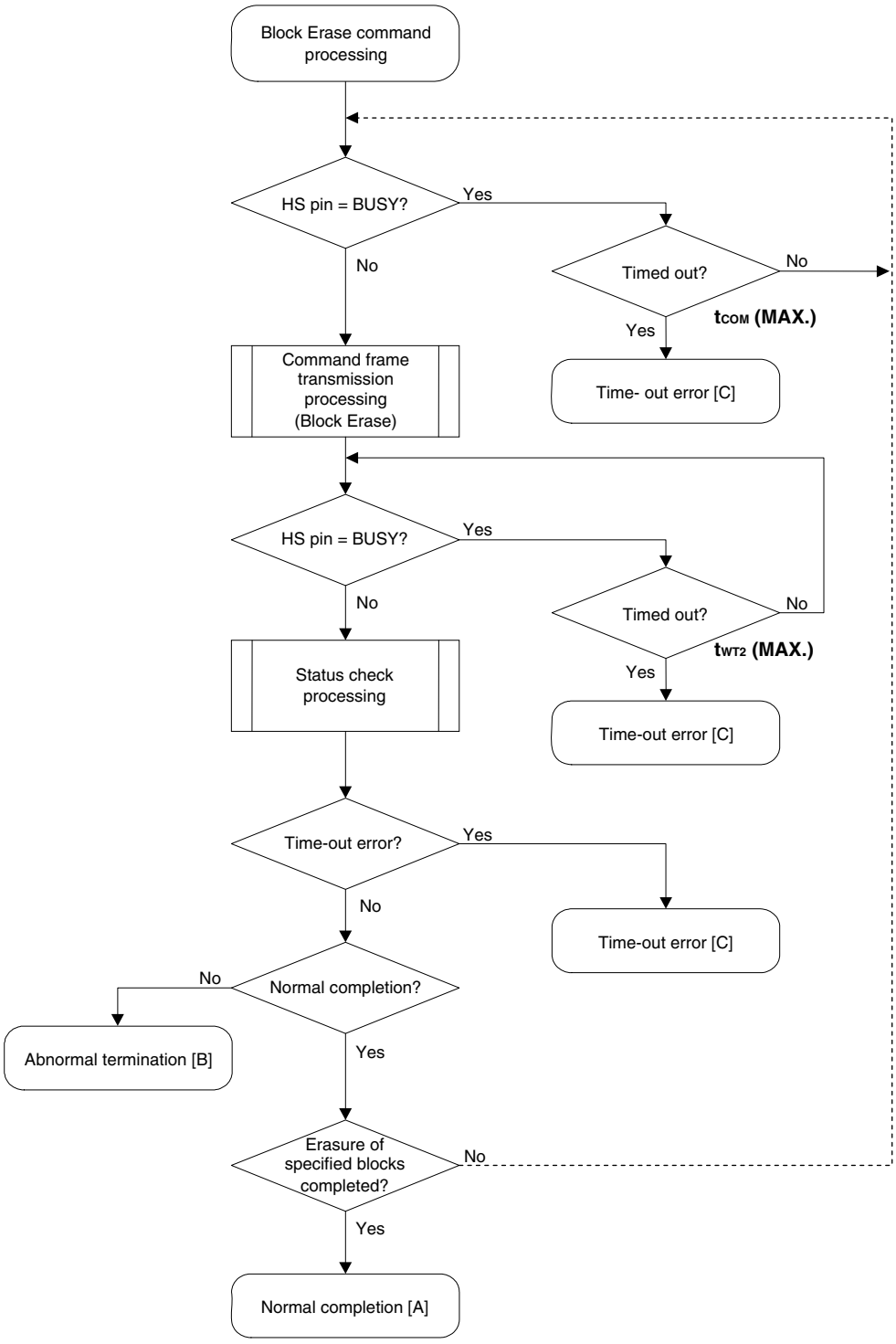
<1> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{COM(MAX.)}$).
<2> The Block Erase command is transmitted by command frame transmission processing.
<3> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{WT2(MAX.)}$).
<4> The status frame is acquired by status check processing.
<5> The following processing is performed according to the result of status check processing.

When the processing ends normally: When the block erase for all of the specified blocks is not yet completed, processing changes the block number and re-executes the sequence from <1>.
When the block erase for all of the specified blocks is completed, the processing ends normally [A].
When the processing ends abnormally: Abnormal termination [B]
When a time-out error occurs: A time-out error [C] is returned.

### 7.8.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and block erase was performed normally. |
| Abnormal termination [B] | Parameter error | 05H | The block number is out of range. |
| | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Protect error | 10H | Chip erase is prohibited in the security setting. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| | WWV1 error | 08H | An erase error has occurred. |
| | EWV1 error | 0BH | |
| | EWV2 error | 0CH | |
| | EWV3 error | 0DH | |
| | Compaction search error | 13H | |
| | Sequencer error | 16H | A sequencer error has occurred. |
| Time-out error [C] | | – | Processing timed out due to the busy status at the HS pin. |

### 7.8.4 Flowchart



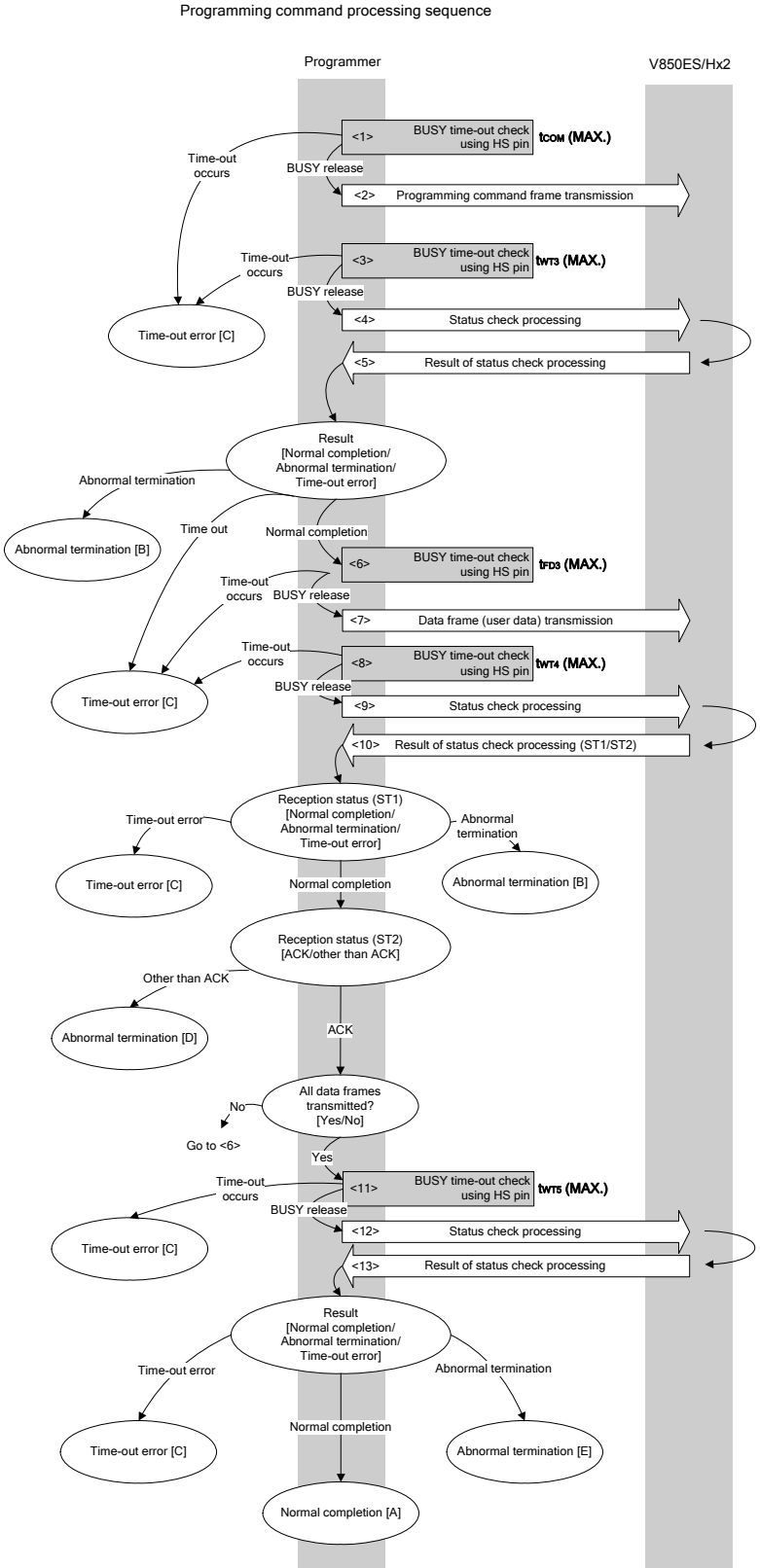
### 7.8.5 Sample program

The following shows a sample program for Block Erase command processing for one block.

```
/************************************************************/
/*                                                          */
/* Erase block command (CSI-HS)                             */
/*                                                          */
/************************************************************/
/* [i] u8 block   ... block number                          */
/* [r] u16        ... error code                            */
/************************************************************/
u16        fl_hs_erase_blk(u8 block)
{
     u16    rc;
     u32    wt2_max;

     fl_cmd_prm[0] = block;           // set param (BLK)
     wt2_max = get_wt2_max(get_block_size(block));

     if (hs_busy_to(tCOM_MAX))
            return FLC_HSTO_ERR;             // t.o. detected :case [C]

     if (rc = put_cmd_hs(FL_COM_ERASE_BLOCK, 2, fl_cmd_prm))
                                              // send "Block Erase" command
            return rc;                        // case [C]

     if (hs_busy_to(wt2_max))
            return FLC_HSTO_ERR;             // t.o. detected :case [C]

     rc = fl_hs_getstatus();                  // get status frame
//   switch(rc) {
//          case   FLC_NO_ERR:   return rc;    break; // case [A]
//          case   FLC_HSTO_ERR: return rc;    break; // case [C]
//          default:             return rc;    break; // case [B]
//   }
     return rc;
}
```

## 7.9 Programming Command

### 7.9.1 Processing sequence chart



### 7.9.2 Description of processing sequence

<1> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{COM(MAX.)}$).
<2> The Programming command is transmitted by command frame transmission processing.
<3> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{WT3(MAX.)}$).
<4> The status frame is acquired by status check processing.
<5> The following processing is performed according to the result of status check processing.

When the processing ends normally: Proceeds to <6>.
When the processing ends abnormally: Abnormal termination [B]
When a time-out error occurs: A time-out error [C] is returned.

<6> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{FD3(MAX.)}$).
<7> User data is transmitted by data frame transmission processing.
<8> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{WT4(MAX.)}$).
<9> The status frame is acquired by status check processing.
<10> The following processing is performed according to the result of status check processing (status code (ST1/ST2)) (also refer to the processing sequence chart and flowchart).

When ST1 = abnormal termination: Abnormal termination [B]
When ST1 = time-out error: A time-out error [C] is returned.
When ST1 = normal completion: The following processing is performed according to the ST2 value.
- When ST2 ≠ ACK: Abnormal termination [D]
- When ST2 = ACK: Proceeds to <11> when transmission of all of the user data is completed.
If there still remain user data to be transmitted, the processing re-executes the sequence from <6>.

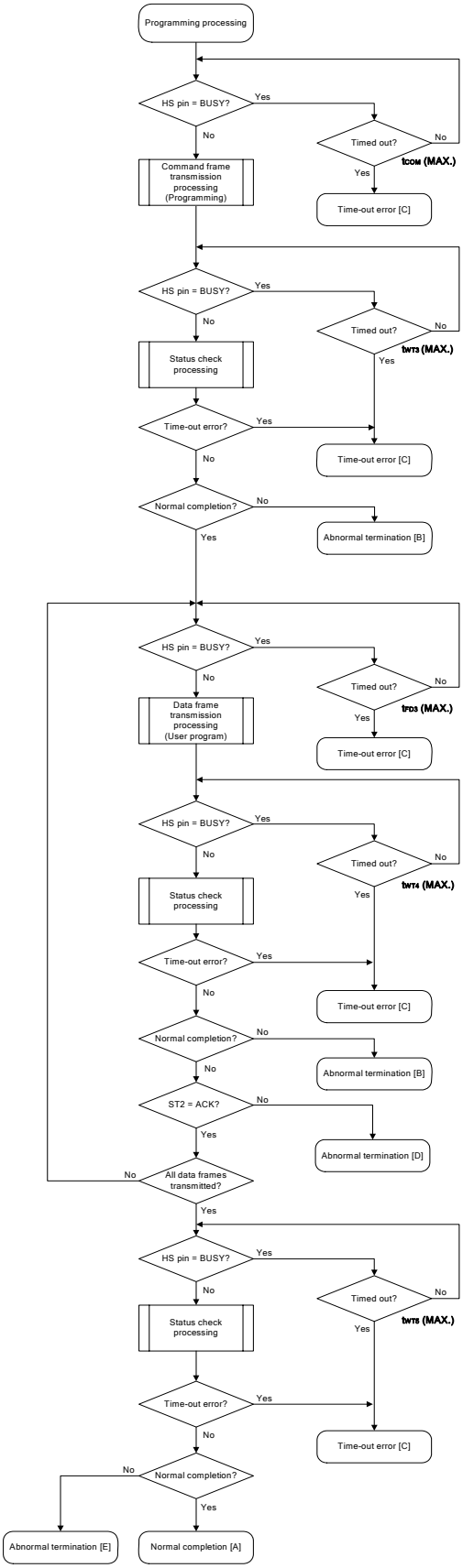
<11> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{WT5(MAX.)}$).
<12> The status frame is acquired by status check processing.
<13> The following processing is performed according to the result of status check processing.

When the processing ends normally: Normal completion [A]
(Indicating that the internal verify check has performed normally after completion of write)
When the processing ends abnormally: Abnormal termination [E]
(Indicating that the internal verify check has not performed normally after completion of write)
When a time-out error occurs: A time-out error [C] is returned.

### 7.9.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and the user data was written normally. |
| Abnormal termination [B] | Parameter error | 05H | The specified start/end address is not the start/end address of the block. |
| | Checksum error | 07H | The checksum of the transmitted command frame or data frame does not match. |
| | Protect error | 10H | Write is prohibited in the security setting. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | Processing timed out due to the busy status at the HS pin. |
| Abnormal termination [D] | WWV1 error | 08H (ST2) | A write error has occurred. |
| | Sequencer error | 16H | A sequencer error has occurred. |
| Abnormal termination [E] | EWV4 error | 11H | An internal verify error has occurred. |
| | Sequencer error | 16H | A sequencer error has occurred. |

### 7.9.4 Flowchart



### 7.9.5 Sample program

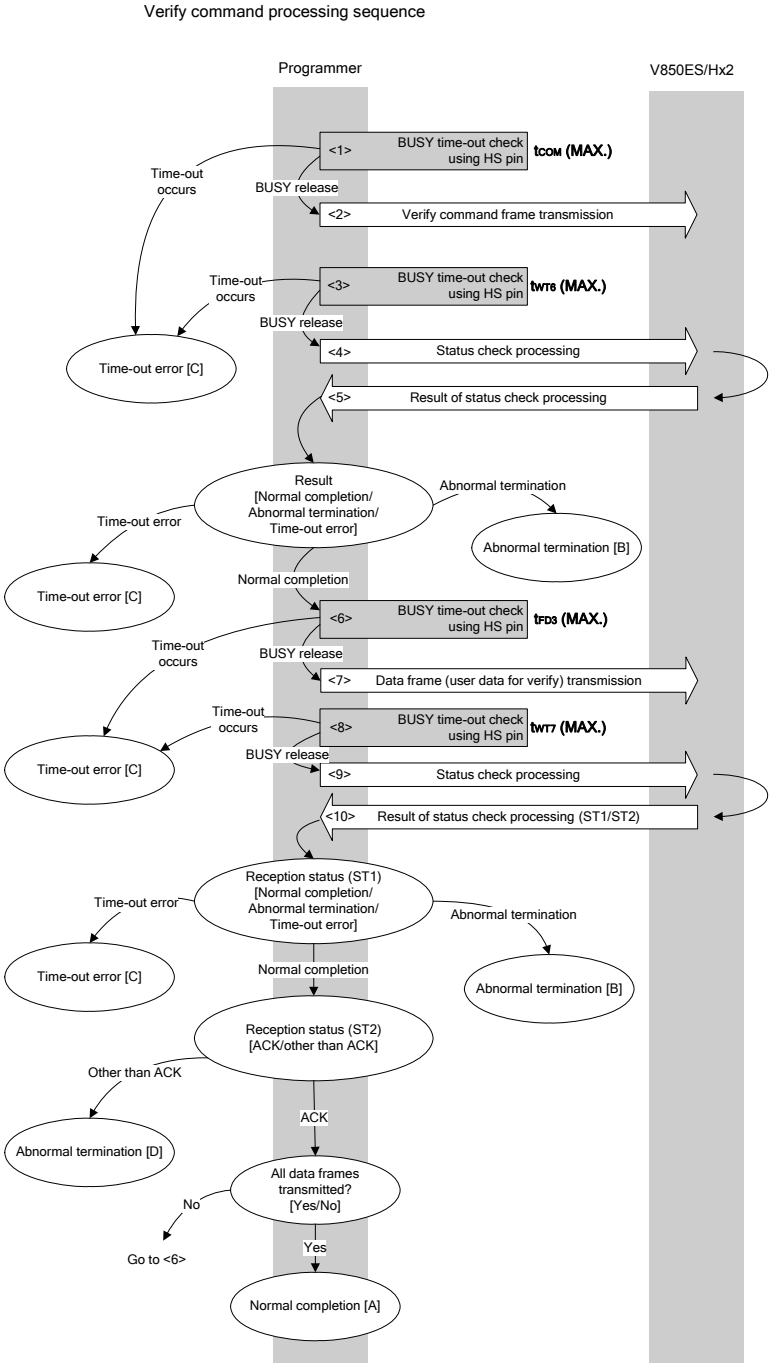
The following shows a sample program for Programming command processing.

```
/************************************************************/
/*                                                          */
/* Write command (CSI-HS)                                   */
/*                                                          */
/************************************************************/
/* [i] u32 top     ... start address                        */
/* [i] u32 bottom  ... end address                          */
/* [r] u16         ... error code                           */
/************************************************************/
u16        fl_hs_write(u32 top, u32 bottom)
{
     u16    rc;
     u32    send_head, send_size;
     bool   is_end;
     u32    wt5_max;


     /*********************************************/
     /*      set params                           */
     /*********************************************/
     set_range_prm(fl_cmd_prm, top, bottom); // set SAH/SAM/SAL, EAH/EAM/EAL
     wt5_max = get_wt5_max(bottom - top + 1);


     /*********************************************/
     /*      send command & check status          */
     /*********************************************/
     if (hs_busy_to(tCOM_MAX))
            return FLC_HSTO_ERR;       // t.o. detected

     if (rc = put_cmd_hs(FL_COM_WRITE, 7, fl_cmd_prm))  // send "Programming" command
            return rc;                                  // t.o. detected

     if (hs_busy_to(tWT3_MAX))
            return FLC_HSTO_ERR;              // t.o. detected

     rc = fl_hs_getstatus();                  // get status frame
     switch(rc) {
            case   FLC_NO_ERR:                break; // continue
     //     case   FLC_HSTO_ERR: return rc;   break; // case [C]
            default:             return rc;   break; // case [B]
     }

     /*********************************************/
     /*      send user data                       */
     /*********************************************/
     send_head = top;

     while(1){
            // make send data frame
            if ((bottom - send_head) > 256){  // rest size > 256 ?
                   is_end = false;            // yes, not end frame
                   send_size  = 256;          // transmit size = 256 byte
```

```
            }
            else{
                  is_end = true;
                  send_size = bottom - send_head + 1;
                                    // transmit size = (bottom - send_head)+1 byte

            }
            memcpy(fl_txdata_frm, rom_buf+send_head, send_size);
                                    // set data frame payload
            send_head += send_size;

            if (hs_busy_to(tFD3_MAX))  // t.o. check before sending data frame
                  return FLC_HSTO_ERR; // t.o. detected

            if (rc = put_dfrm_hs(send_size, fl_txdata_frm, is_end))
                                          // send user data
                  return rc;              // error detected

            if (hs_busy_to(tWT4_MAX))
                  return FLC_HSTO_ERR;    // t.o. detected


            rc = fl_hs_getstatus();       // get status frame
            switch(rc) {
                  case   FLC_NO_ERR:              break; // continue
            //    case   FLC_HSTO_ERR: return rc; break; // case [C]
                  default:             return rc; break; // case [B]
            }
            if (fl_st2 != FLST_ACK){              // ST2 = ACK ?
                  rc = decode_status(fl_st2);     // No
                  return rc;                      // case [D]
            }
            if (is_end)                   // send all user data ?
                  break;                  // yes



      }
      /*********************************************/
      /*      Check internally verify                */
      /*********************************************/
      if (hs_busy_to(wt5_max))
            return FLC_HSTO_ERR;          // t.o. detected

      rc = fl_hs_getstatus();             // get status frame
//    switch(rc) {
//          case   FLC_NO_ERR:   return rc;   break; // case [A]
//          case   FLC_HSTO_ERR: return rc;   break; // case [C]
//          default:             return rc;   break; // case [B]
//    }
      return rc;

}
```

## 7.10 Verify Command

### 7.10.1 Processing sequence chart



### 7.10.2 Description of processing sequence

<1> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{COM(MAX.)}$).
<2> The Verify command is transmitted by command frame transmission processing.
<3> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{WT6(MAX.)}$).
<4> The status frame is acquired by status check processing.
<5> The following processing is performed according to the result of status check processing.

When the processing ends normally: Proceeds to <6>.
When the processing ends abnormally: Abnormal termination [B]
When a time-out error occurs: A time-out error [C] is returned.

<6> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{FD3(MAX.)}$).
<7> User data for verifying is transmitted by data frame transmission processing.
<8> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{WT7(MAX.)}$).
<9> The status frame is acquired by status check processing.
<10> The following processing is performed according to the result of status check processing (status code (ST1/ST2)) (also refer to the processing sequence chart and flowchart).

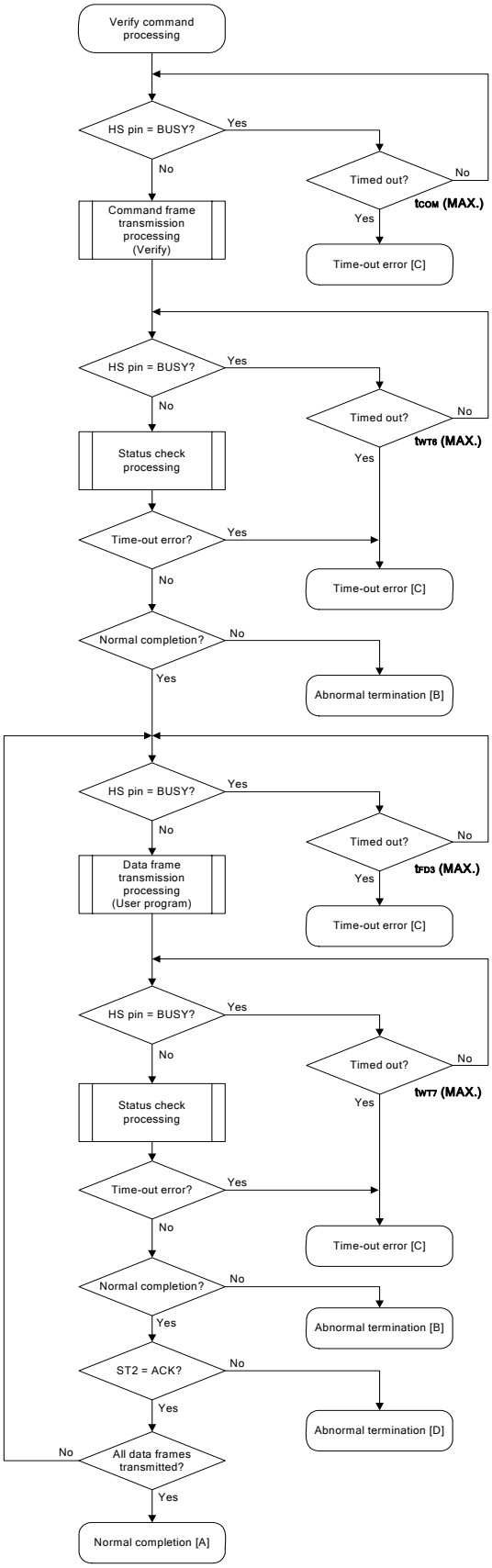
When ST1 = abnormal termination: Abnormal termination [B]
When ST1 = time-out error: A time-out error [C] is returned.
When ST1 = normal completion: The following processing is performed according to the ST2 value.

- When ST2 = ACK: If transmission of all data frames is completed, the processing ends normally [A].
  If there still remain data frames to be transmitted, the processing re-executes the sequence from <6>.
- When ST2 ≠ ACK: Abnormal termination [D]

### 7.10.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and the verify was completed normally. |
| Abnormal termination [B] | Parameter error | 05H | The specified start/end address is not the start/end address of the block. |
| | Checksum error | 07H | The checksum of the transmitted command frame or data frame does not match. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | Processing timed out due to the busy status at the HS pin. |
| Abnormal termination [D] | Verify error | 0FH (ST2) | The verify has failed, or another error has occurred. |
| | | 0EH | |
| | Sequencer error | 16H | A sequencer error has occurred. |

### 7.10.4 Flowchart



### 7.10.5 Sample program

The following shows a sample program for Verify command processing.

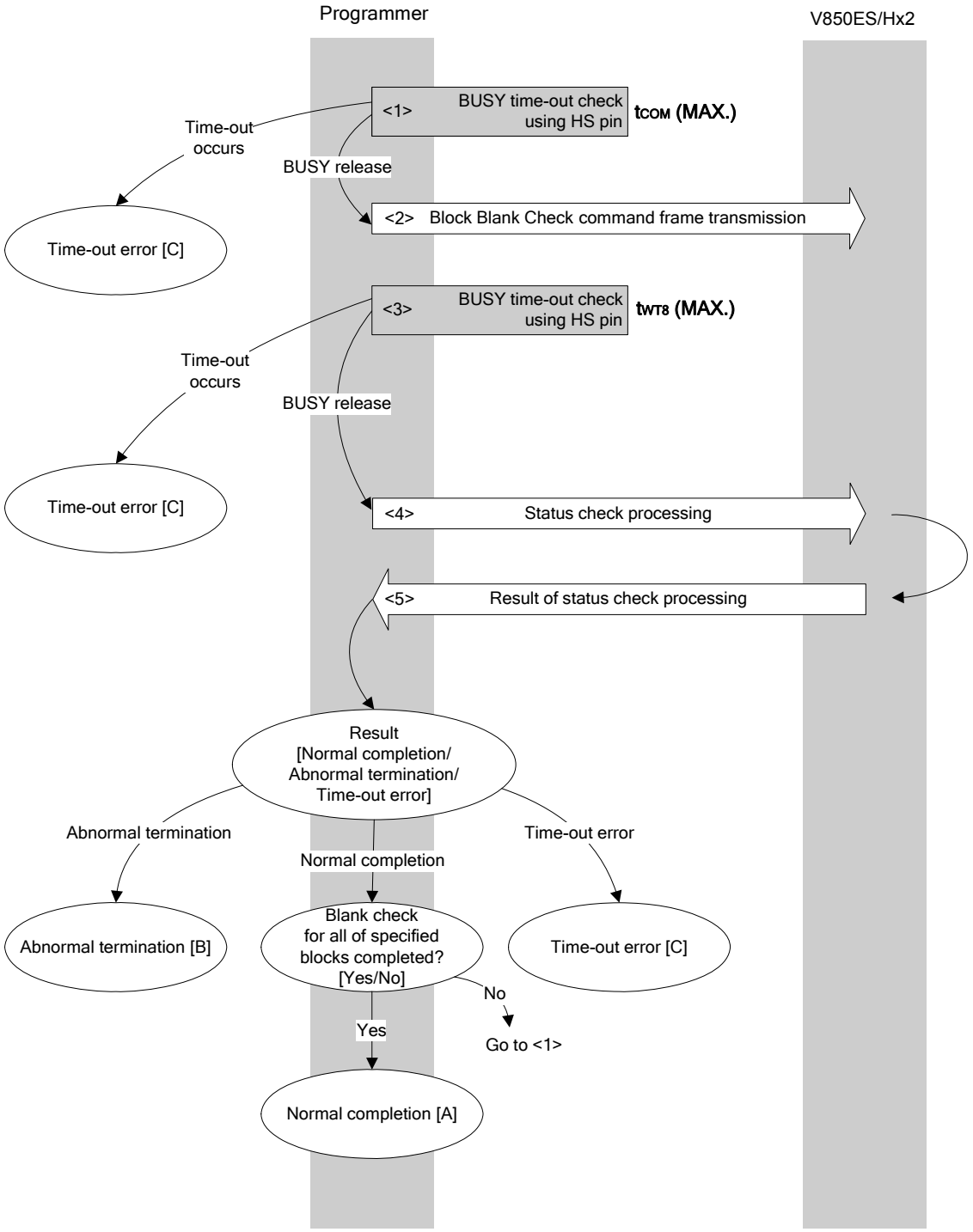
```
/************************************************************/
/*                                                          */
/*   Verify command (CSI-HS)                                */
/*                                                          */
/************************************************************/
/*   [i] u32 top   ... start address                        */
/*   [i] u32 bottom      ... end address                    */
/*   [r] u16             ... error code                     */
/************************************************************/
u16  fl_hs_verify(u32 top, u32 bottom, u8 *buf)
{
     u16    rc;
     u32    send_head, send_size;
     bool   is_end;

     /*********************************************/
     /*     set params                            */
     /*********************************************/
     set_range_prm(fl_cmd_prm, top, bottom); // set SAH/SAM/SAL, EAH/EAM/EAL


     /*********************************************/
     /*     send command & check status           */
     /*********************************************/

     if (hs_busy_to(tCOM_MAX))
            return FLC_HSTO_ERR;       // t.o. detected

     if (rc = put_cmd_hs(FL_COM_VERIFY, 7, fl_cmd_prm))    // send "Verify" command
            return rc;                 // error detected

     if (hs_busy_to(tWT6_MAX))
            return FLC_HSTO_ERR;       // t.o. detected

     rc = fl_hs_getstatus();                 // get status frame
     switch(rc) {
            case   FLC_NO_ERR:                break; // continue
     //     case   FLC_HSTO_ERR: return rc;   break; // case [C]
            default:             return rc;   break; // case [B]
     }


     /*********************************************/
     /*     send user data                        */
     /*********************************************/
```

```
      send_head = top;

      while(1){

            // make send data frame
            if ((bottom - send_head) > 256){        // rest size > 256 ?
                  is_end = false;                   // yes, not end frame
                  send_size  = 256;                 // transmit size = 256 byte
            }
            else{
                  is_end = true;
                  send_size = bottom - send_head + 1;     // transmit size = (bottom
- send_head)+1 byte

            }
            memcpy(fl_txdata_frm, buf+send_head, send_size);     //  set  data  frame
payload
            send_head += send_size;

            if (hs_busy_to(tFD3_MAX))
                  return FLC_HSTO_ERR;       // t.o. detected

            if (rc = put_dfrm_hs(send_size, fl_txdata_frm, is_end))     //  send  user
data
                  return rc;                 // error detected


            if (hs_busy_to(tWT7_MAX))
                  return FLC_HSTO_ERR;       // t.o. detected

            rc = fl_hs_getstatus();          // get status frame
            switch(rc) {
                  case   FLC_NO_ERR:                break; // continue
            //    case   FLC_HSTO_ERR: return rc;   break; // case [C]
                  default:             return rc;   break; // case [B]
            }
            if (fl_st2 != FLST_ACK){               // ST2 = ACK ?
                  rc = decode_status(fl_st2);      // No
                  return rc;                       // case [D]
            }
            if (is_end)               // send all user data ?
                  break;              // yes
      }
      return FLC_NO_ERR;   // case [A]
  }
```

## 7.11 Block Blank Check Command

### 7.11.1 Processing sequence chart

Block Blank Check command processing sequence



### 7.11.2 Description of processing sequence

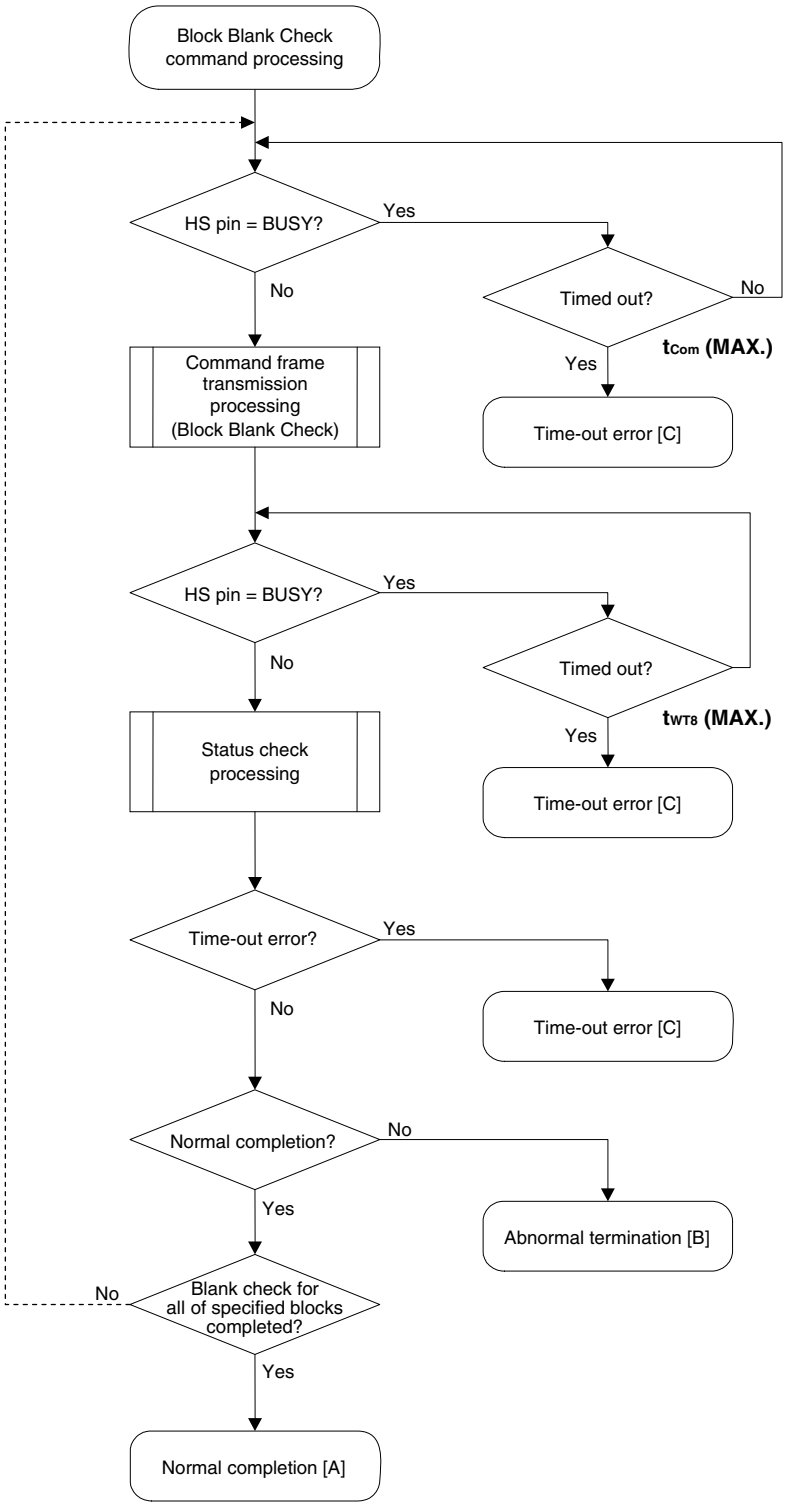
<1> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{COM(MAX.)}$).
<2> The Block Blank Check command is transmitted by command frame transmission processing.
<3> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{WT8(MAX.)}$).
<4> The status frame is acquired by status check processing.
<5> The following processing is performed according to the result of status check processing.

When the processing ends abnormally: Abnormal termination [B]
When the processing ends normally: If the blank check for all of the specified blocks is completed, the processing ends normally [A].
If the blank check for all of the specified blocks is not yet completed, processing changes the block number and re-executes the sequence from <1>.
When a time-out error occurs: A time-out error [C] is returned.

### 7.11.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and all of the specified blocks are blank. |
| Abnormal termination [B] | Parameter error | 05H | The block number is out of range. |
| | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| | EWV4 error | 11H | The specified block in the flash memory is not blank. |
| | Sequencer error | 16H | A sequencer error has occurred. |
| Time-out error [C] | | – | Processing timed out due to the busy status at the HS pin. |

### 7.11.4 Flowchart



### 7.11.5 Sample program

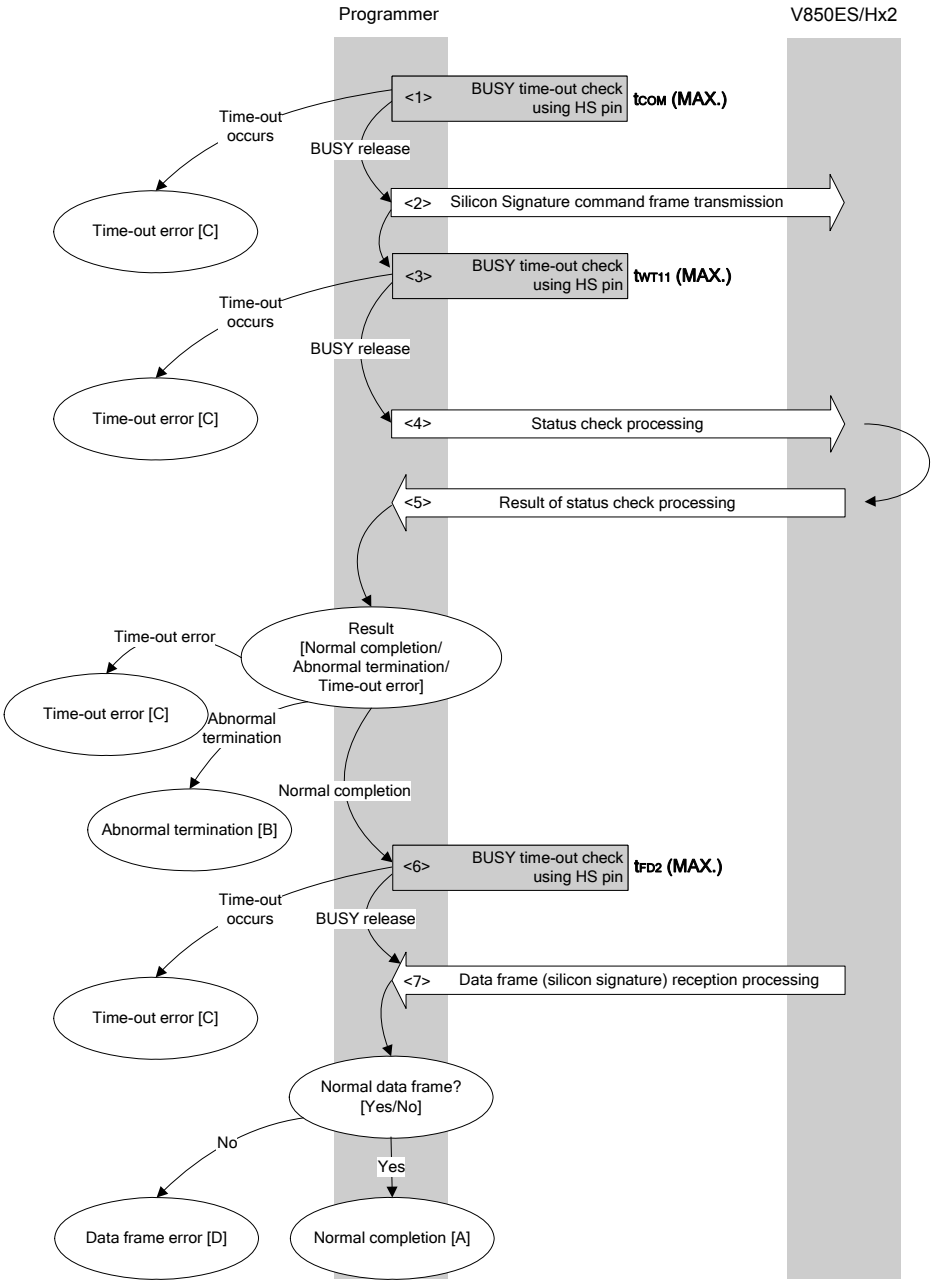
The following shows a sample program for Block Blank Check command processing for one block.

```
/*****************************************************************/
/*                                                               */
/* Block blank check command (CSI-HS)                            */
/*                                                               */
/*****************************************************************/
/* [i] u16 block  ... block number                               */
/* [r] u16        ... error code                                 */
/*****************************************************************/
u16        fl_hs_blk_blank_chk(u8 block)
{
     u16    rc;
     u32    wt8_max;

     fl_cmd_prm[0] = block;     // "BLK"
     wt8_max = get_wt8_max(get_block_size(block));


     if (hs_busy_to(tCOM_MAX))
            return FLC_HSTO_ERR;       // t.o. detected :case [C]

     if (rc = put_cmd_hs(FL_COM_BLOCK_BLANK_CHK, 2, fl_cmd_prm))
                                       // send "Block Blank Check" command
            return rc;                 // case [C]

     if (hs_busy_to(wt8_max))
            return FLC_HSTO_ERR;       // t.o. detected :case [C]

     rc = fl_hs_getstatus();           // get status frame
//   switch(rc) {
//          case   FLC_NO_ERR:   return rc;    break; // case [A]
//          case   FLC_HSTO_ERR: return rc;    break; // case [C]
//          default:             return rc;    break; // case [B]
//   }
     return rc;
}
```

## 7.12 Silicon Signature Command

### 7.12.1 Processing sequence chart

Silicon Signature command processing sequence



### 7.12.2 Description of processing sequence

<1> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{COM}$(MAX.)).
<2> The Silicon Signature command is transmitted by command frame transmission processing.
<3> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{WT11}$(MAX.)).
<4> The status frame is acquired by status check processing.
<5> The following processing is performed according to the result of status check processing.

When the processing ends normally: Proceeds to <6>.
When the processing ends abnormally: Abnormal termination [B]
When a time-out error occurs: A time-out error [C] is returned.

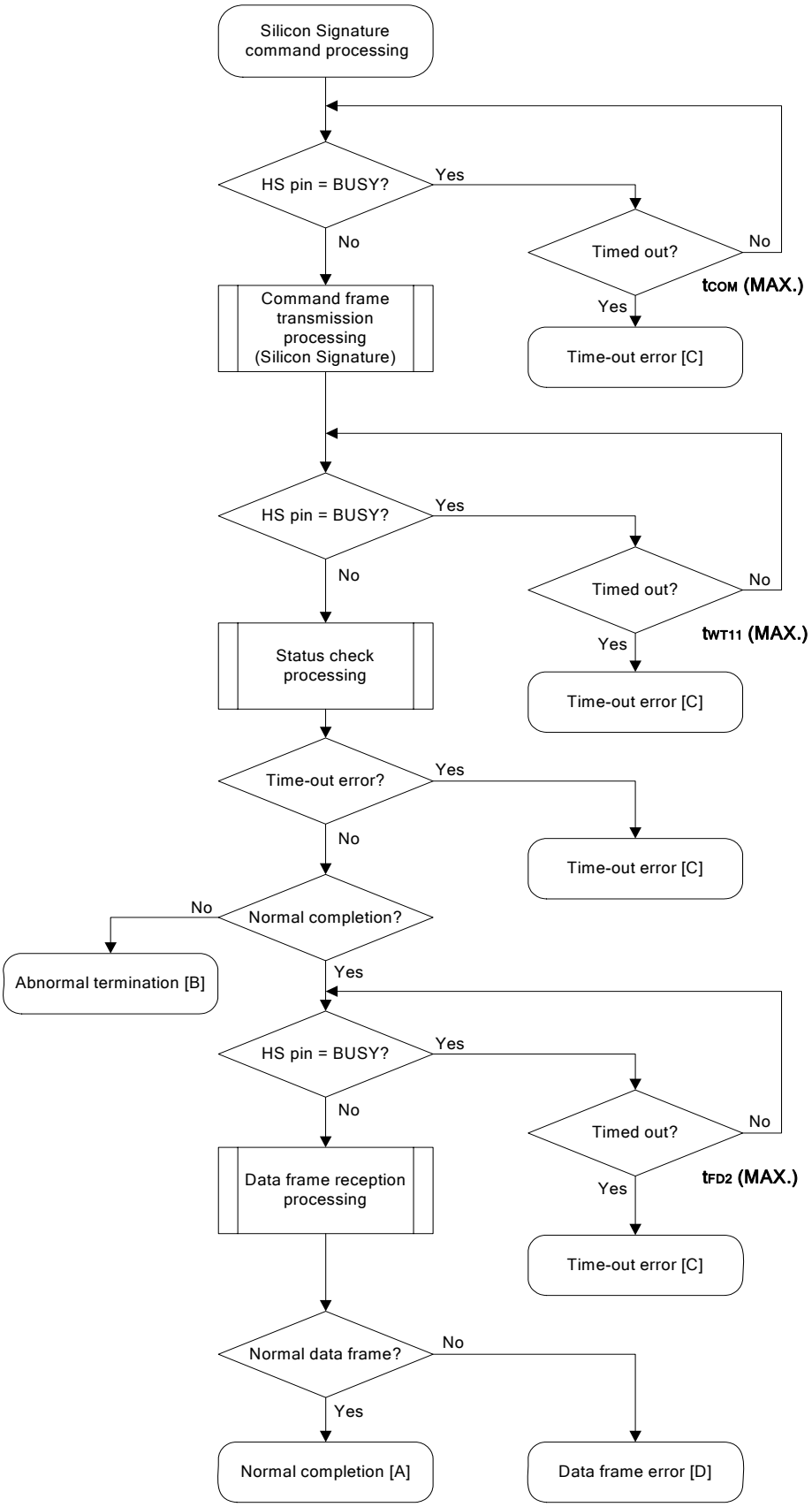
<6> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{FD2}$(MAX.)).
<7> The received data frame (silicon signature data) is checked.

If data frame is normal: Normal completion [A]
If data frame is abnormal: Data frame error [D]

### 7.12.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and the silicon signature was acquired normally. |
| Abnormal termination [B] | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | Processing timed out due to the busy status at the HS pin. |
| Data frame error [D] | | – | The checksum of the data frame received as silicon signature data does not match. |

### 7.12.4 Flowchart



### 7.12.5 Sample program

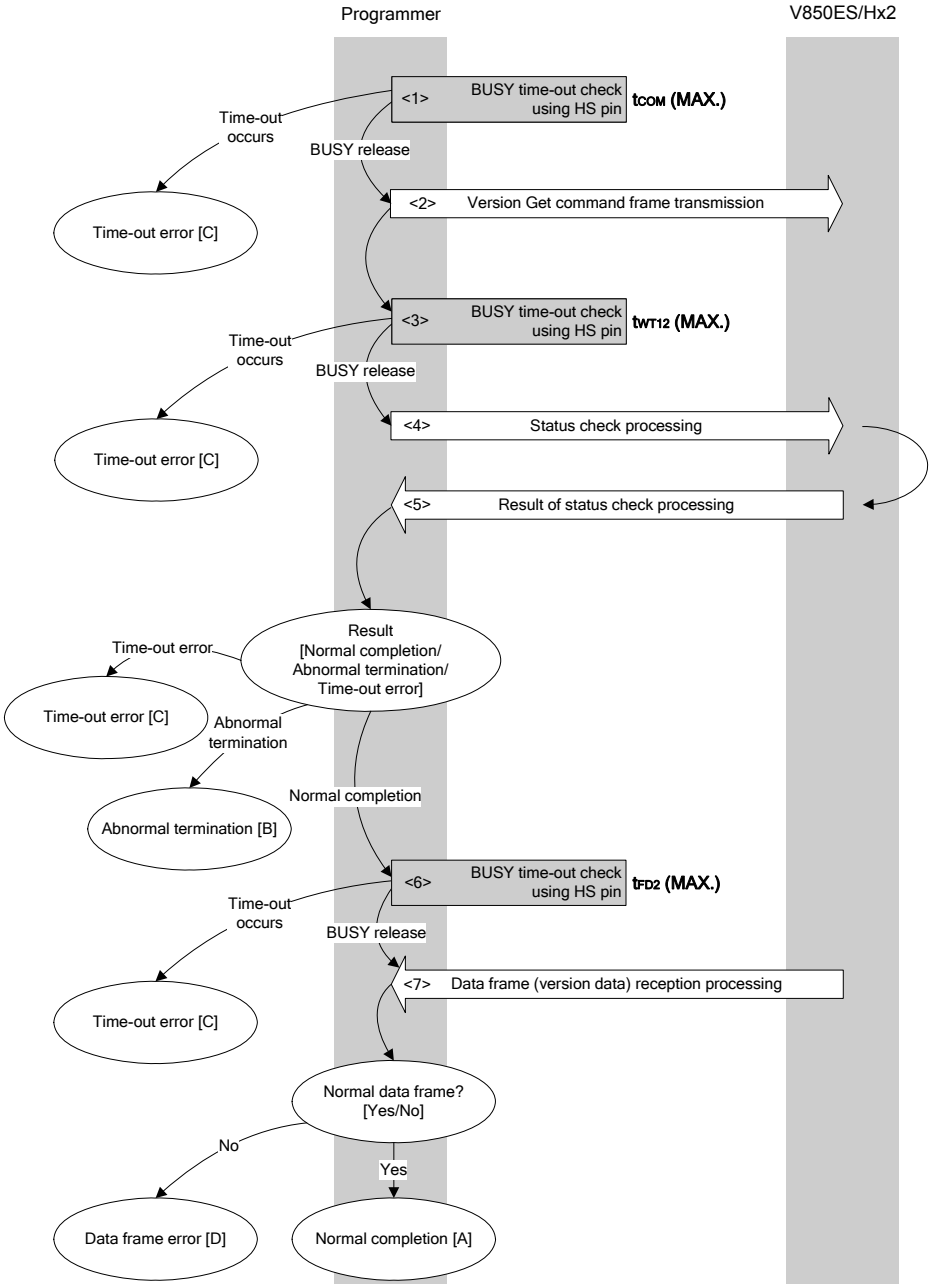
The following shows a sample program for Silicon Signature command processing.

```
/*****************************************************************/
/*                                                               */
/* Get silicon signature command (CSI-HS)                        */
/*                                                               */
/*****************************************************************/
/* [i] u8 *sig... pointer to signature save area                 */
/* [r] u16    ... error code                                     */
/*****************************************************************/
u16        fl_hs_getsig(u8 *sig)
{
     u16    rc;

     if (hs_busy_to(tCOM_MAX))
            return FLC_HSTO_ERR;       // t.o. detected :case [C]

     if (rc = put_cmd_hs(FL_COM_GET_SIGNATURE, 1, fl_cmd_prm))
                                       // send "Silicon Signature" command
            return rc;                 // error detected :case [C]

     if (hs_busy_to(tWT11_MAX))
            return FLC_HSTO_ERR;       // t.o. detected :case [C]


     rc = fl_hs_getstatus();           // get status frame
     switch(rc) {
            case   FLC_NO_ERR:                 break; // continue
     //     case   FLC_HSTO_ERR: return rc;    break; // case [C]
            default:             return rc;    break; // case [B]
     }


     if (hs_busy_to(tFD2_MAX))
            return FLC_HSTO_ERR;       // t.o. detected :case [C]

     rc = get_dfrm_hs(fl_rxdata_frm);  // get signature data

     switch(rc) {
            case   FLC_NO_ERR:                 break; // continue
     //     case   FLC_HSTO_ERR: return rc;    break; // case [C]
            default:             return rc;    break; // case [D]
     }

     memcpy(sig, fl_rxdata_frm+OFS_STA_PLD, fl_rxdata_frm[OFS_LEN]);
            // copy Signature data

     return rc;                        // case [A]

}
```

## 7.13 Version Get Command

### 7.13.1 Processing sequence chart

Version Get command processing sequence



### 7.13.2 Description of processing sequence

<1> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{COM(MAX.)}$).
<2> The Version Get command is transmitted by command frame transmission processing.
<3> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{WT12(MAX.)}$).
<4> The status frame is acquired by status check processing.
<5> The following processing is performed according to the result of status check processing.

When the processing ends normally: Proceeds to <6>.
When the processing ends abnormally: Abnormal termination [B]
When a time-out error occurs: A time-out error [C] is returned.

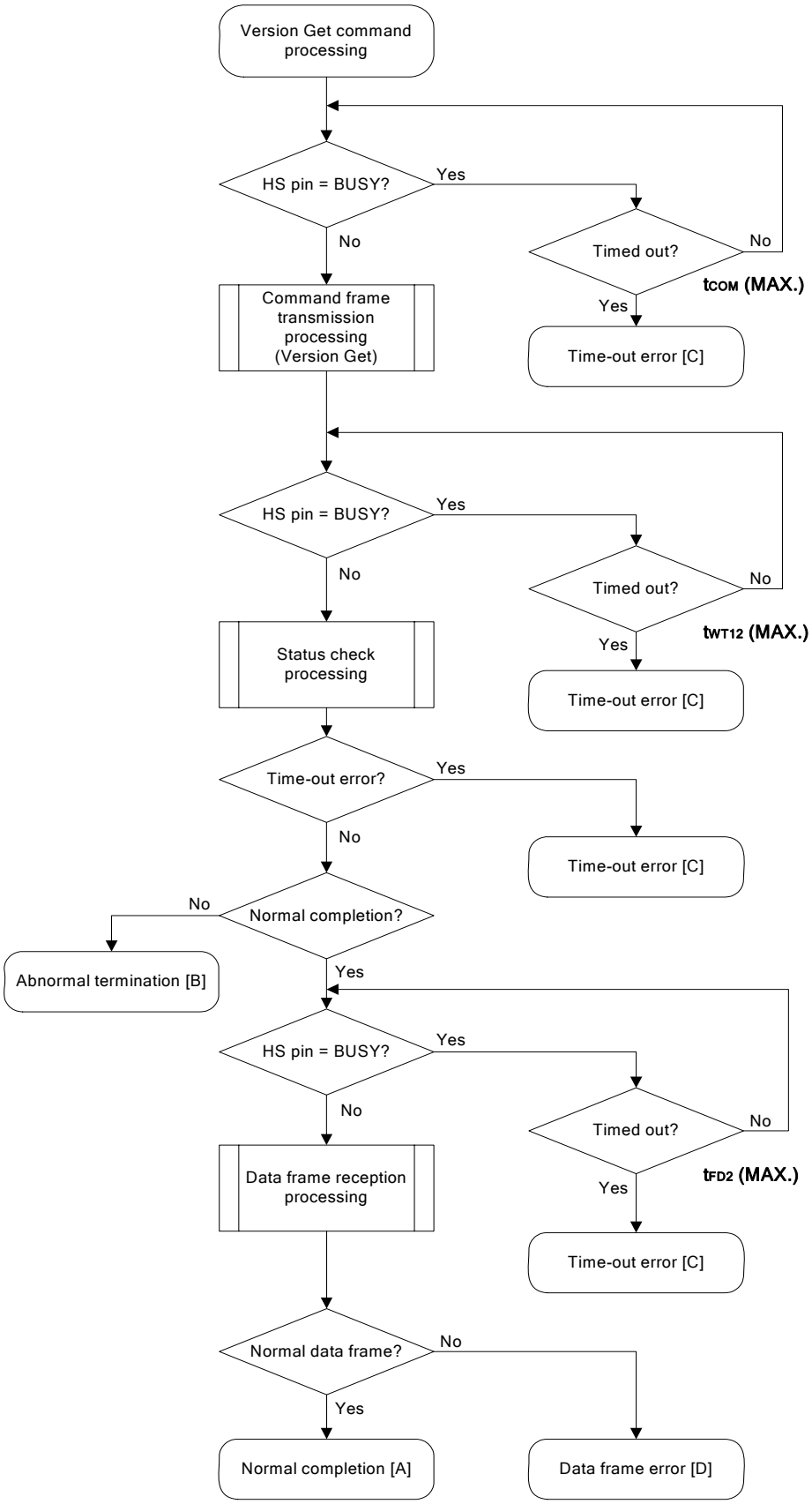
<6> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{FD2(MAX.)}$).
<7> The received data frame (version data) is checked.

If data frame is normal: Normal completion [A]
If data frame is abnormal: Data frame error [D]

### 7.13.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and version data was acquired normally. |
| Abnormal termination [B] | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | Processing timed out due to the busy status at the HS pin. |
| Data frame error [D] | | – | The checksum of the data frame received as version data does not match. |

### 7.13.4 Flowchart



### 7.13.5 Sample program

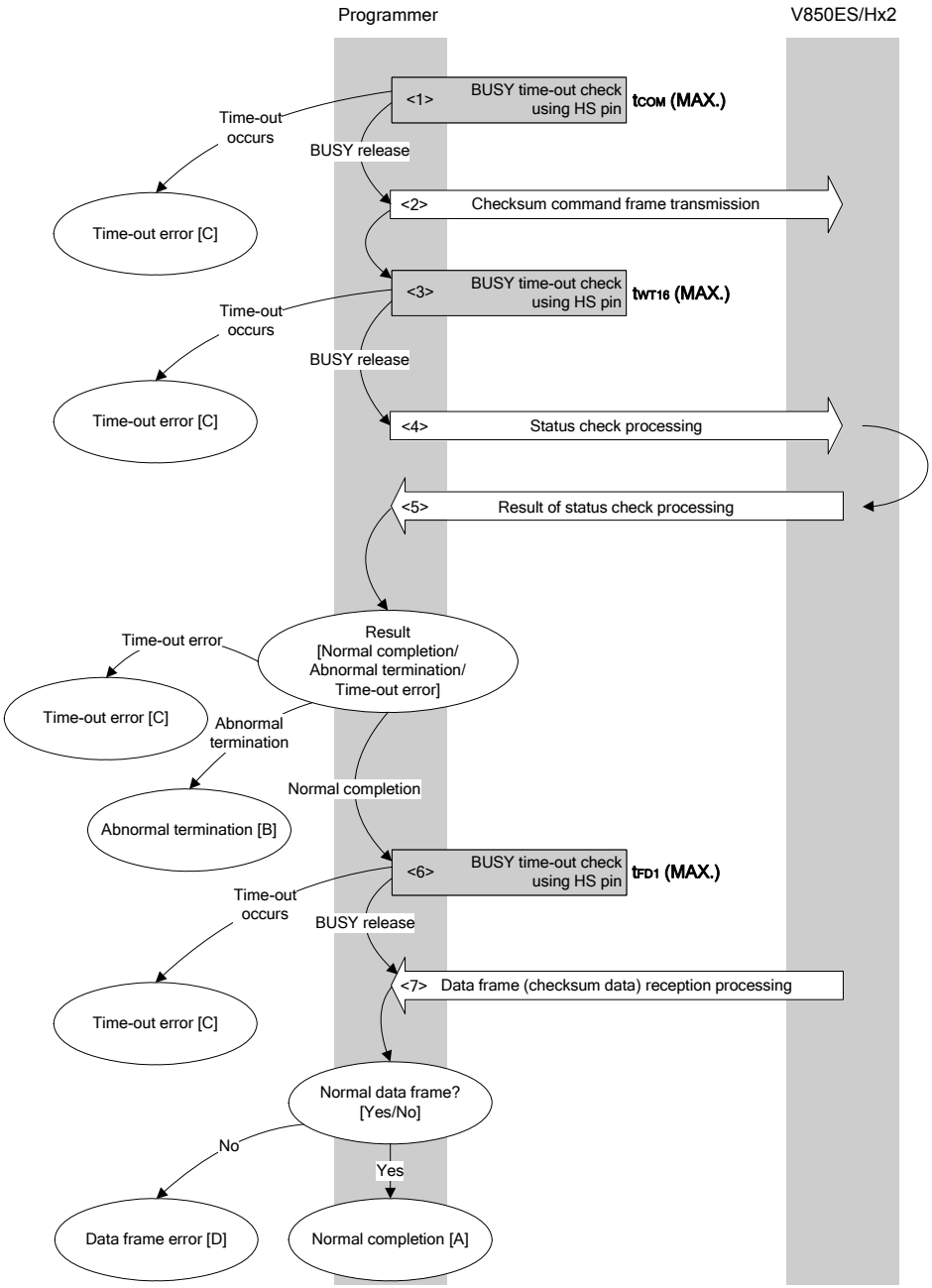
The following shows a sample program for Version Get command processing.

```
/************************************************************/
/*                                                          */
/* Get device/firmware version command (CSI-HS)             */
/*                                                          */
/************************************************************/
/* [i] u8 *buf     ... pointer to version date save area    */
/* [r] u16         ... error code                           */
/************************************************************/
u16        fl_hs_getver(u8 *buf)
{
     u16    rc;

     if (hs_busy_to(tCOM_MAX))
            return FLC_HSTO_ERR;       // t.o. detected :case [C]

     if (rc = put_cmd_hs(FL_COM_GET_VERSION, 1, fl_cmd_prm))
                                       // send "Version Get" command
            return rc;                 // error detected :case [C]

     if (hs_busy_to(tWT12_MAX))
            return FLC_HSTO_ERR;       // t.o. detected :case [C]

     rc = fl_hs_getstatus();           // get status frame
     switch(rc) {
            case   FLC_NO_ERR:                break; // continue
     //     case   FLC_HSTO_ERR: return rc;   break; // case [C]
            default:             return rc;   break; // case [B]
     }

     if (hs_busy_to(tFD2_MAX))
            return FLC_HSTO_ERR;       // t.o. detected :case [C]

     rc = get_dfrm_hs(fl_rxdata_frm);        // get signature data
     switch(rc) {
            case   FLC_NO_ERR:                break; // continue
     //     case   FLC_HSTO_ERR: return rc;   break; // case [C]
            default:             return rc;   break; // case [D]
     }
     memcpy(buf, fl_rxdata_frm+OFS_STA_PLD, DFV_LEN);// copy version data
     return rc;                        // case [A]

}
```

## 7.14 Checksum Command

### 7.14.1 Processing sequence chart

Checksum command processing sequence



### 7.14.2 Description of processing sequence

<1> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{COM(MAX.)}$).
<2> The Checksum command is transmitted by command frame transmission processing.
<3> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{WT16(MAX.)}$).
<4> The status frame is acquired by status check processing.
<5> The following processing is performed according to the result of status check processing.

When the processing ends normally: Proceeds to <6>.
When the processing ends abnormally: Abnormal termination [B]
When a time-out error occurs: A time-out error [C] is returned.

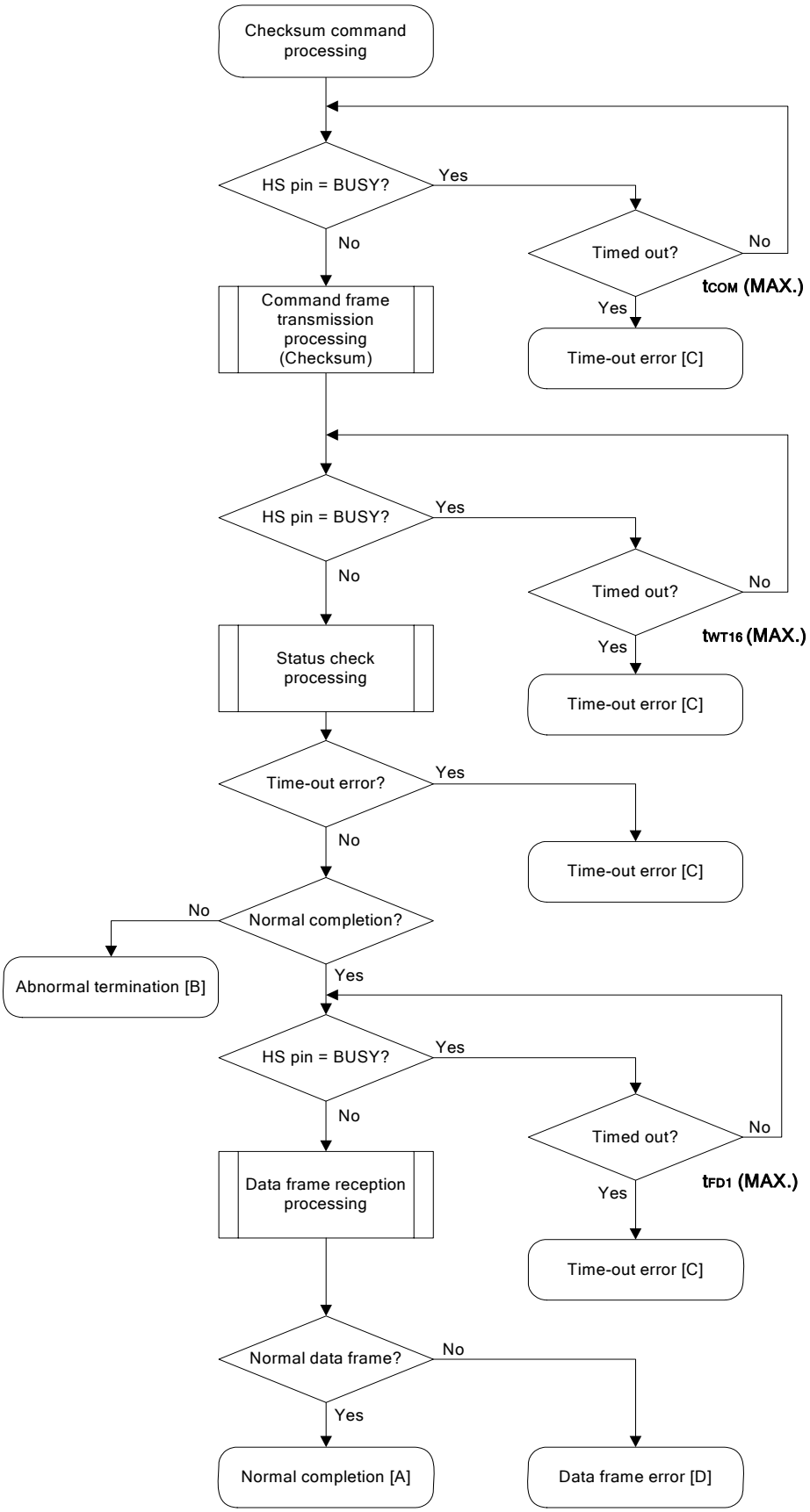
<6> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{FD1\ (MAX.)}$).
<7> The received data frame (checksum data) is checked.

If data frame is normal: Normal completion [A]
If data frame is abnormal: Data frame error [D]

### 7.14.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and checksum data was acquired normally. |
| Abnormal termination [B] | Parameter error | 05H | The specified start/end address is not the start/end address of the block. |
| | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | Processing timed out due to the busy status at the HS pin. |
| Data frame error [D] | | – | The checksum of the data frame received as version data does not match. |

### 7.14.4 Flowchart



### 7.14.5 Sample program

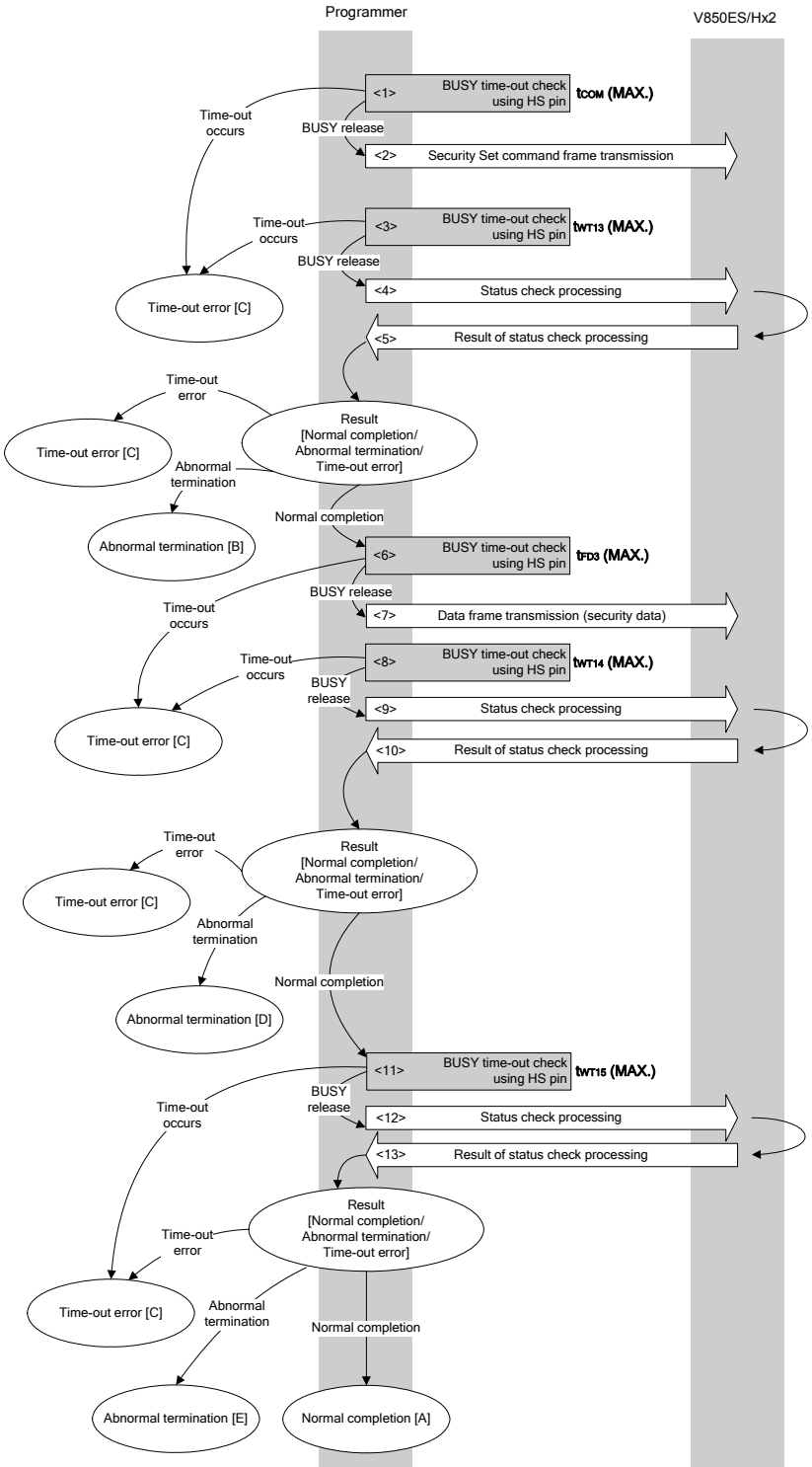
The following shows a sample program for Checksum command processing.

```
/************************************************************/
/*                                                          */
/* Get checksum command (CSI-HS)                            */
/*                                                          */
/************************************************************/
/* [i] u16 *sum    ... pointer to checksum save area        */
/* [i] u32 top     ... start address                        */
/* [i] u32 bottom  ... end address                          */
/* [r] u16         ... error code                           */
/************************************************************/
u16        fl_hs_getsum(u16 *sum, u32 top, u32 bottom)
{
     u16    rc;

     // set params
     set_range_prm(fl_cmd_prm, top, bottom); // set SAH/SAM/SAL, EAH/EAM/EAL

     if (hs_busy_to(tCOM_MAX))
            return FLC_HSTO_ERR;             // t.o. detected :case [C]

     if (rc = put_cmd_hs(FL_COM_GET_CHECK_SUM, 7, fl_cmd_prm))
                                              // send "Checksum" command
            return rc;                        // error detected :case [C]

     if (hs_busy_to(tWT16_MAX))
            return FLC_HSTO_ERR;             // t.o. detected :case [C]

     rc = fl_hs_getstatus();                  // get status frame
     switch(rc) {
            case   FLC_NO_ERR:                break; // continue
     //     case   FLC_HSTO_ERR: return rc;   break; // case [C]
            default:             return rc;   break; // case [B]
     }


     if (hs_busy_to(tFD1_MAX))
            return FLC_HSTO_ERR;             // t.o. detected :case [C]

     rc = get_dfrm_hs(fl_rxdata_frm);         // get signature data

     switch(rc) {
            case   FLC_NO_ERR:                break; // continue
     //     case   FLC_HSTO_ERR: return rc;   break; // case [C]
            default:             return rc;   break; // case [D]
     }

     *sum = (fl_rxdata_frm[OFS_STA_PLD] << 8) + fl_rxdata_frm[OFS_STA_PLD+1];
                                    // set SUM data
     return rc;                     // case [A]
}
```

## 7.15 Security Set Command

### 7.15.1 Processing sequence chart

Security Set command processing sequence



### 7.15.2 Description of processing sequence

<1> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{COM(MAX.)}$).
<2> The Security Set command is transmitted by command frame transmission processing.
<3> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{WT13(MAX.)}$).
<4> The status frame is acquired by status check processing.
<5> The following processing is performed according to the result of status check processing.

| | |
|---|---|
| When the processing ends normally: | Proceeds to <6>. |
| When the processing ends abnormally: | Abnormal termination [B] |
| When a time-out error occurs: | A time-out error [C] is returned. |

<6> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{FD3(MAX.)}$).
<7> The data frame (security setting data) is transmitted by data frame transmission processing.
<8> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{WT14(MAX.)}$).
<9> The status frame is acquired by status check processing.
<10> The following processing is performed according to the result of status check processing.

| | |
|---|---|
| When the processing ends normally: | Proceeds to <11>. |
| When the processing ends abnormally: | Abnormal termination [D] |
| When a time-out error occurs: | A time-out error [C] is returned. |

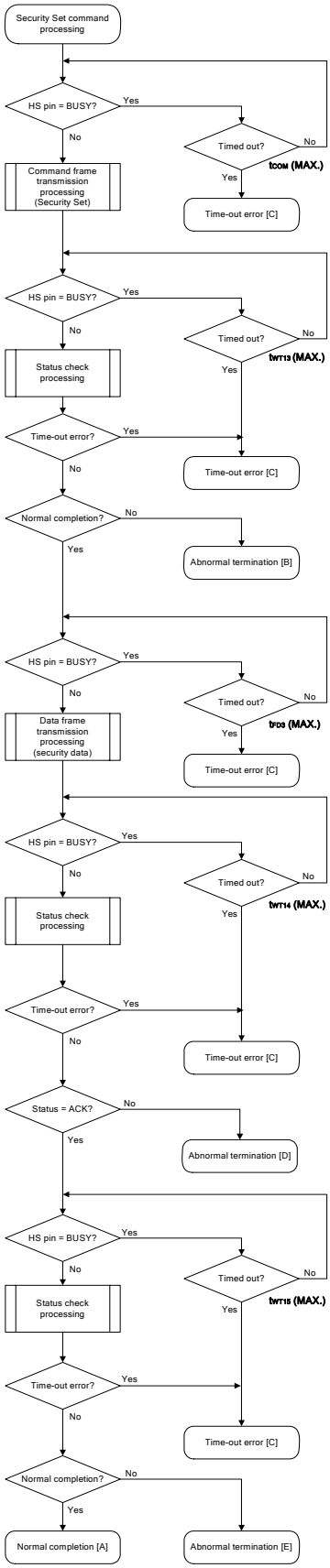
<11> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{WT15(MAX.)}$).
<12> The status frame is acquired by status check processing.
<13> The following processing is performed according to the result of status check processing.

| | |
|---|---|
| When the processing ends normally: | Normal completion [A] |
| When the processing ends abnormally: | Abnormal termination [E] |
| When a time-out error occurs: | A time-out error [C] is returned. |

### 7.15.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and security setting was performed normally. |
| Abnormal termination [B] | Parameter error | 05H | The command information (parameter) is not 00H. |
| | Checksum error | 07H | The checksum of the transmitted command frame or data frame does not match. |
| | Protect error | 10H | The ID codes do not match. |
| | Negative acknowledgment (NACK) | 15H | Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | Processing timed out due to the busy status at the HS pin. |
| Abnormal termination [D] | WWV1 error | 08H | • Security data is already set.<br>• A security data write error has occurred. |
| | Sequencer error | 16H | A sequencer error has occurred. |
| Abnormal termination [E] | EWV4 error | 11H | An internal verify error has occurred. |
| | Sequencer error | 16H | A sequencer error has occurred. |

### 7.15.4 Flowchart



### 7.15.5 Sample program

The following shows a sample program for Security Set command processing.

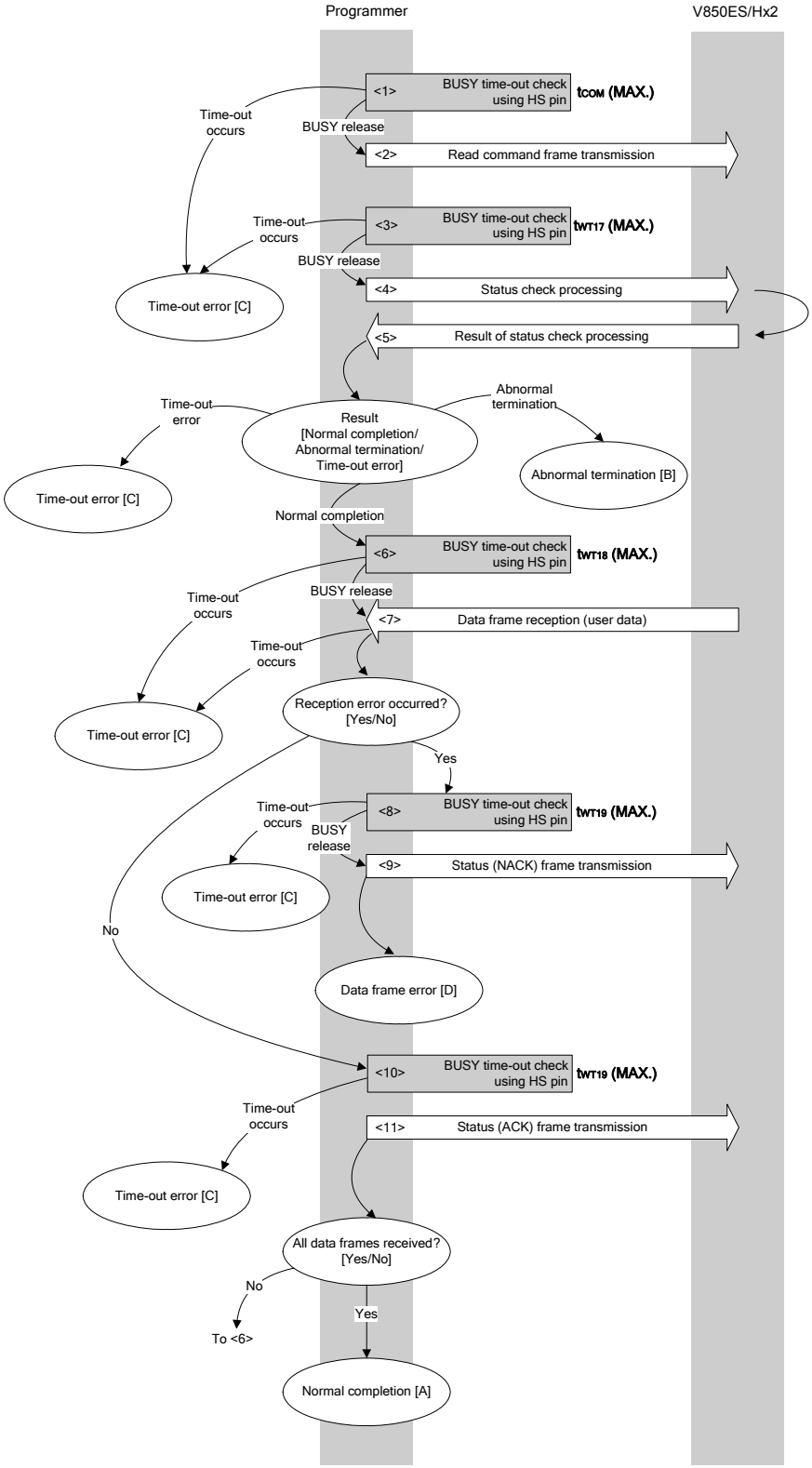
```
/************************************************************/
/*                                                          */
/*   Set security flag command (CSI-HS)                     */
/*                                                          */
/************************************************************/
/*   [i] u8 scf    ... Security flag data                   */
/*   [r] u16             ... error code                     */
/************************************************************/
u16  fl_hs_setscf(u8 scf, u32 vect)
{
     u16    rc;

     /*******************************************************/
     /*     set params & data frame                         */
     /*******************************************************/
     fl_cmd_prm[0] = 0x00;                  // "BLK" (must be 0x00)
     fl_cmd_prm[1] = 0x00;                  // "PAG" (must be 0x00)

     fl_txdata_frm[0] = (scf|= 0b11110000);  // "FLG" (upper 4bits must be '1' (to
make sure))
     fl_txdata_frm[1] = (u8)(vect >> 16);    // "ADH"
     fl_txdata_frm[2] = (u8)(vect >>  8);    // "ADM"
     fl_txdata_frm[3] = (u8) vect;           // "ADL"


     /*********************************************/
     /*     send command                          */
     /*********************************************/
     if (hs_busy_to(tCOM_MAX))
           return FLC_HSTO_ERR;       // t.o. detected :case [C]

     if (rc = put_cmd_hs(FL_COM_SET_SECURITY, 3, fl_cmd_prm))    //   send   "Security
Set" command
           return rc;                 // error detected :case [C]

     if (hs_busy_to(tWT13_MAX))
           return FLC_HSTO_ERR;       // t.o. detected :case [C]

     rc = fl_hs_getstatus();          // get status frame
     switch(rc) {
           case   FLC_NO_ERR:                break; // continue
     //    case   FLC_HSTO_ERR: return rc;   break; // case [C]
           default:             return rc;   break; // case [B]
     }
```

```
    /*********************************************/
    /*   send data frame (security setting data)    */
    /*********************************************/
    if (hs_busy_to(tFD3_MAX))
          return FLC_HSTO_ERR;       // t.o. detected :case [C]

    if (rc = put_dfrm_hs(4, fl_txdata_frm, true))  // send securithi setting data
          return rc;                 // error detected :case [C]

    if (hs_busy_to(tWT14_MAX))
          return FLC_HSTO_ERR;       // t.o. detected :case [C]

    rc = fl_hs_getstatus();          // get status frame
    switch(rc) {
          case   FLC_NO_ERR:               break; // continue
    //    case   FLC_HSTO_ERR:return rc;   break; // case [C]
          default:           return rc;    break; // case [B]
    }


    /*********************************************/
    /*     Check internally verify                */
    /*********************************************/
    if (hs_busy_to(tWT15_MAX))
          return FLC_HSTO_ERR;       // t.o. detected

    rc = fl_hs_getstatus();          // get status frame again
//  switch(rc) {
//        case   FLC_NO_ERR:  return rc;   break; // case [A]
//        case   FLC_HSTO_ERR:return rc;   break; // case [C]
//        default:           return rc;    break; // case [B]
//  }
    return rc;
}
```

## 7.16 Read Command

### 7.16.1 Processing sequence chart

Read command processing sequence



### 7.16.2 Description of processing sequence

<1> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{COM(MAX.)}$).

<2> The Read command is transmitted by command frame transmission processing.

<3> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{WT17(MAX.)}$).

<4> The status frame is acquired by status check processing.

<5> The following processing is performed according to the result of status check processing.

| | |
|---|---|
| When the processing ends normally: | Proceeds to <6>. |
| When the processing ends abnormally: | Abnormal termination [B] |
| When a time-out error occurs: | A time-out error [C] is returned. |

<6> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{WT18(MAX.)}$).

<7> The data frame (user data) in the flash memory is received by data frame reception processing.

| | |
|---|---|
| When the processing ends normally: | Proceeds to <10>. |
| When an error such as checksum error occurs: | Proceeds to <8>. |
| When a time-out error occurs: | A time-out error [C] is returned. |

<8> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{WT19(MAX.)}$).

<9> The NACK frame is transmitted by data frame transmission processing.
A data frame error [D] is returned.

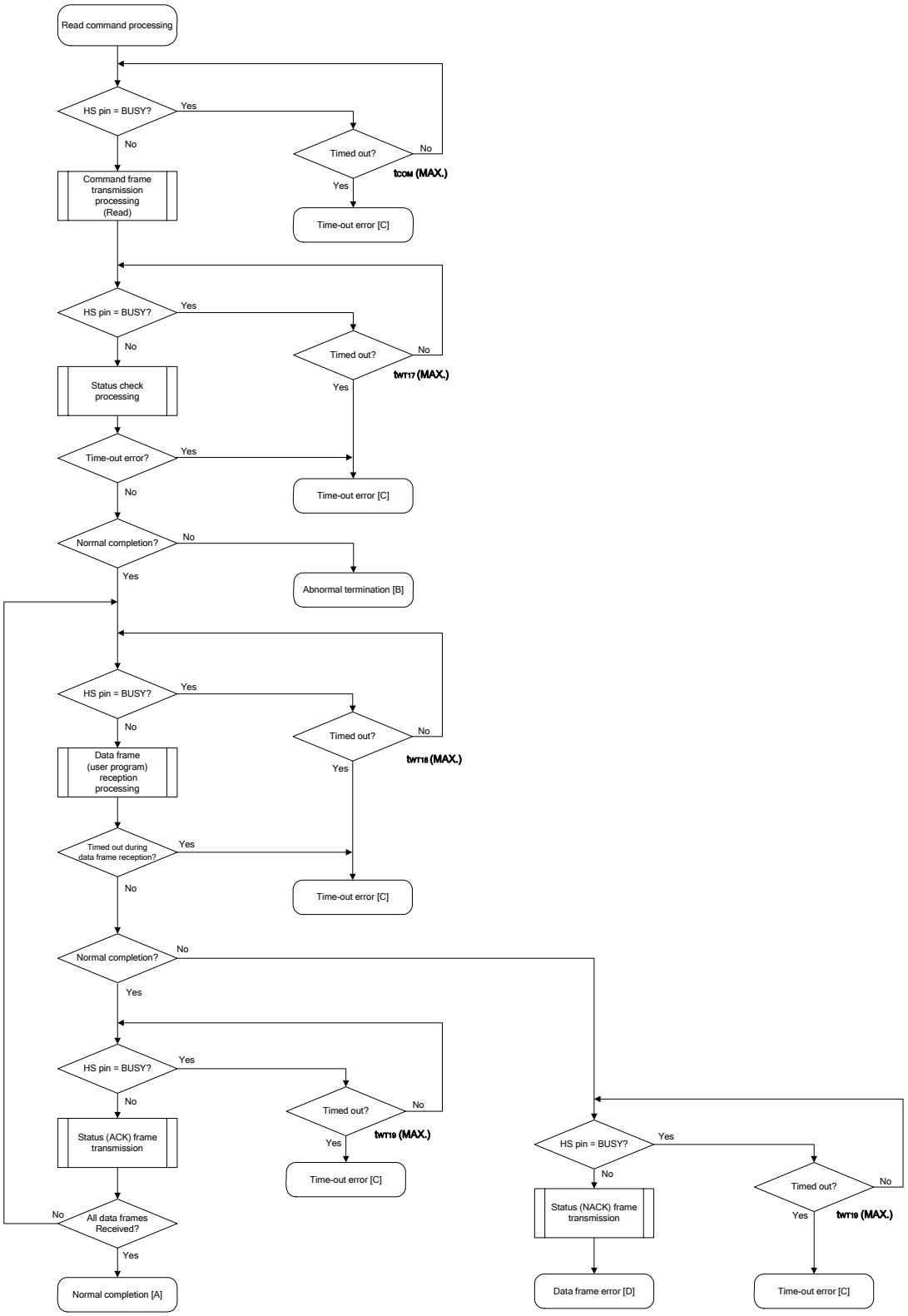
<10> A V850ES/Hx2 BUSY status is checked using the HS pin.
If a BUSY time-out occurs, a time-out error [C] is returned (time-out time $t_{WT19(MAX.)}$).

<11> The ACK frame is transmitted by data frame transmission processing.
When reception of all data frames is completed, the normal completion status [A] is returned.
If there still remain data frames to be received, the sequence is re-executed from <6>.

### 7.16.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and the read data was set normally. |
| Abnormal termination [B] | Parameter error | 05H | The specified start/end address is not the start/end address of the block. |
| | Checksum error | 07H | The checksum of the transmitted command frame or data frame does not match. |
| | Protect error | 10H | Read is prohibited in the security setting. |
| | Negative acknowledgment (NACK) | 15H | Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | Processing timed out due to the busy status at the HS pin. |
| Data frame error [D] | | – | The checksum of the data frame received as read data does not match. |

### 7.16.4 Flowchart



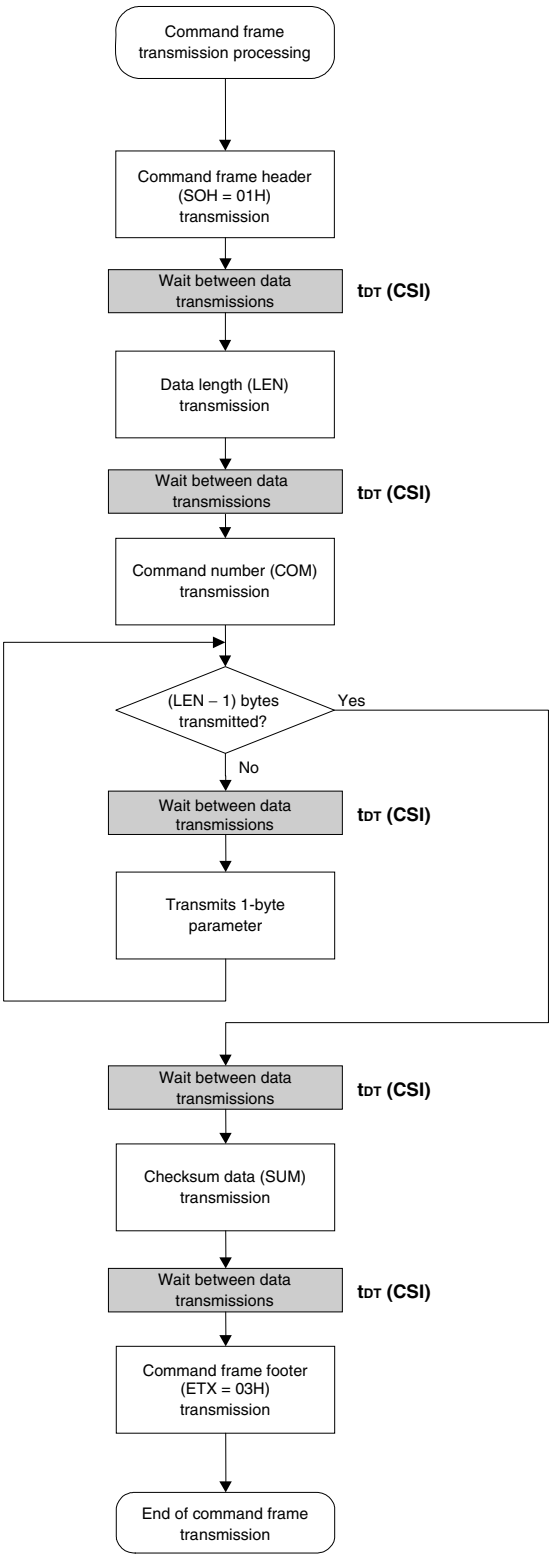
### 7.16.5 Sample program

The following shows a sample program for Read command processing.

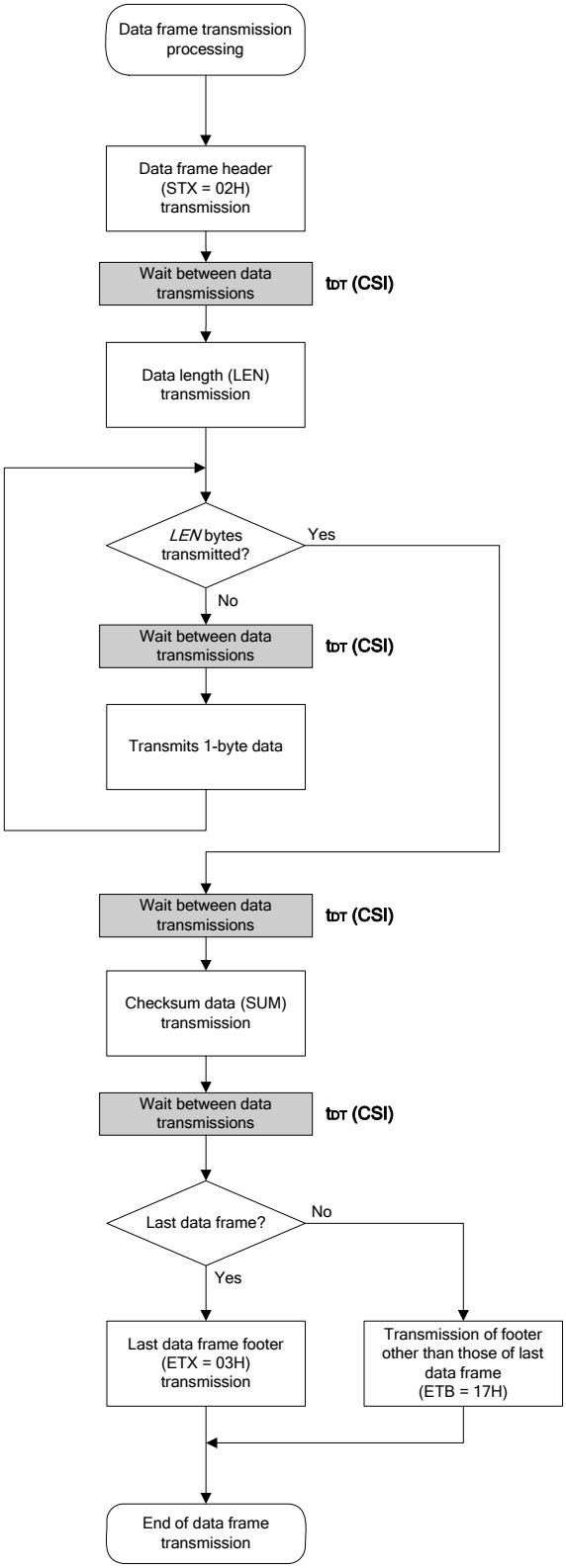
```
/************************************************************/
/*                                                          */
/*      Read command                                        */
/*                                                          */
/************************************************************/
u16     fl_hs_read(u32 top, u32 bottom)
{
  u16   rc;
  u32   read_head;
  u16   len;
  u8    hooter;

  /*********************************************/
  /*    set params                             */
  /*********************************************/
  set_range_prm(fl_cmd_prm, top, bottom); // set SAH/SAM/SAL, EAH/EAM/EAL


  /*********************************************/
  /*    send command & check status            */
  /*********************************************/
  if (hs_busy_to(tCOM_MAX))
        return FLC_HSTO_ERR;       // t.o. detected :case [C]

  if (rc = put_cmd_hs(FL_COM_READ, 7, fl_cmd_prm))
        return rc;

  if (hs_busy_to(tWT17_MAX))
        return FLC_HSTO_ERR;       // t.o. detected :case [C]

  rc = fl_hs_getstatus();          // get status frame
  switch(rc) {
        case   FLC_NO_ERR:               break; // continue
  //    case   FLC_HSTO_ERR: return rc;  break; // case [C]
        default:           return rc;    break; // case [B]
  }

  /*********************************************/
  /*    receive user data                      */
  /*********************************************/
  read_head = top;

  while(1){

        if (hs_busy_to(tWT18_MAX))
              return FLC_HSTO_ERR;       // t.o. detected :case [C]

        rc = get_dfrm_hs(fl_rxdata_frm);  // get ROM data from FLASH
        switch(rc) {
              case   FLC_NO_ERR:               break; // continue
              case   FLC_HSTO_ERR: return rc;  break; // case [C]

        //    case   FLC_RX_DFSUM_ERR:
```

```
                default:                          // case [D]

                        if (hs_busy_to(tWT19_MAX))
                                return FLC_HSTO_ERR;       // t.o. detected

                        put_sfrm_hs(FLST_NACK);            // send status(NACK) frame
                        return rc;
                        break;

        }
        if (hs_busy_to(tWT19_MAX))
                return FLC_HSTO_ERR;       // t.o. detected

        put_sfrm_hs(FLST_ACK);             // send status(ACK) frame

        /*********************************************/
        /*     save ROM data                          */
        /*********************************************/
        if ((len = fl_rxdata_frm[OFS_LEN]) == 0)       // get length
                len = 256;

        memcpy(read_buf+read_head, fl_rxdata_frm+2, len); // save to external RAM

        read_head += len;

        /*********************************************/
        /*     end check                              */
        /*********************************************/
        hooter = fl_rxdata_frm[len + 3];
        if (hooter == FL_ETB)              // end frame ?
                continue;                  // no
        break;                             // yes
  }

  return FLC_NO_ERR;
}
```

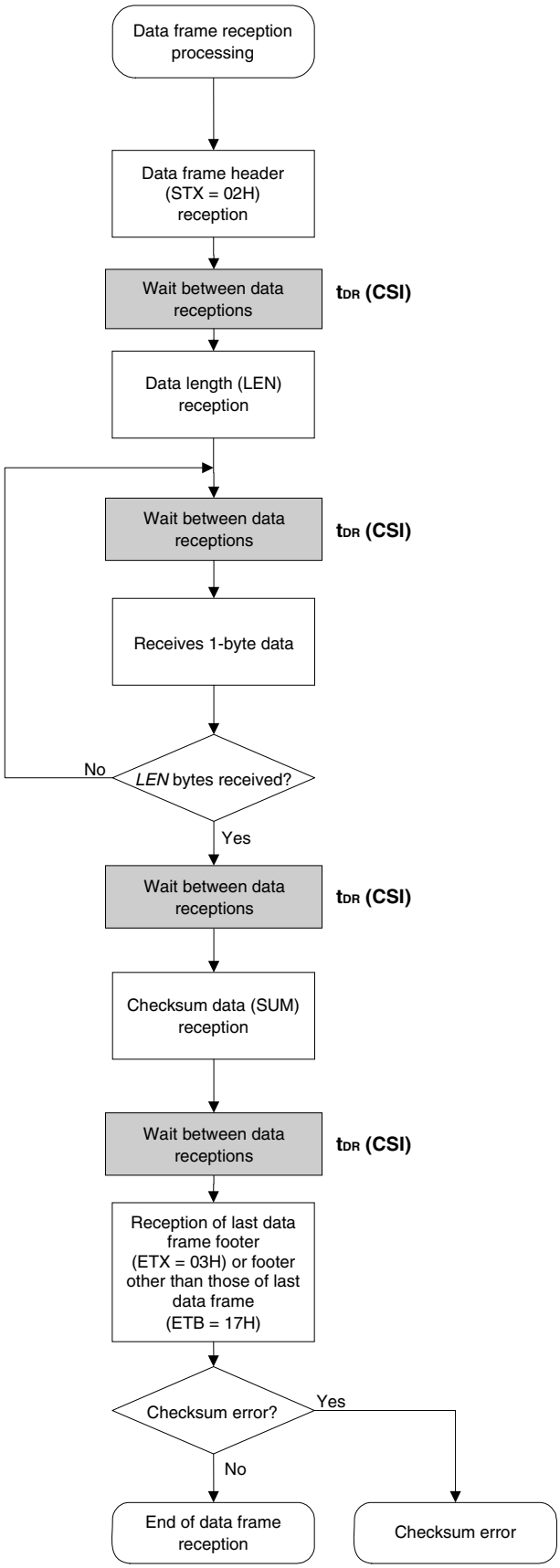
## 8.1 Command Frame Transmission Processing Flowchart



## 8.2 Data Frame Transmission Processing Flowchart



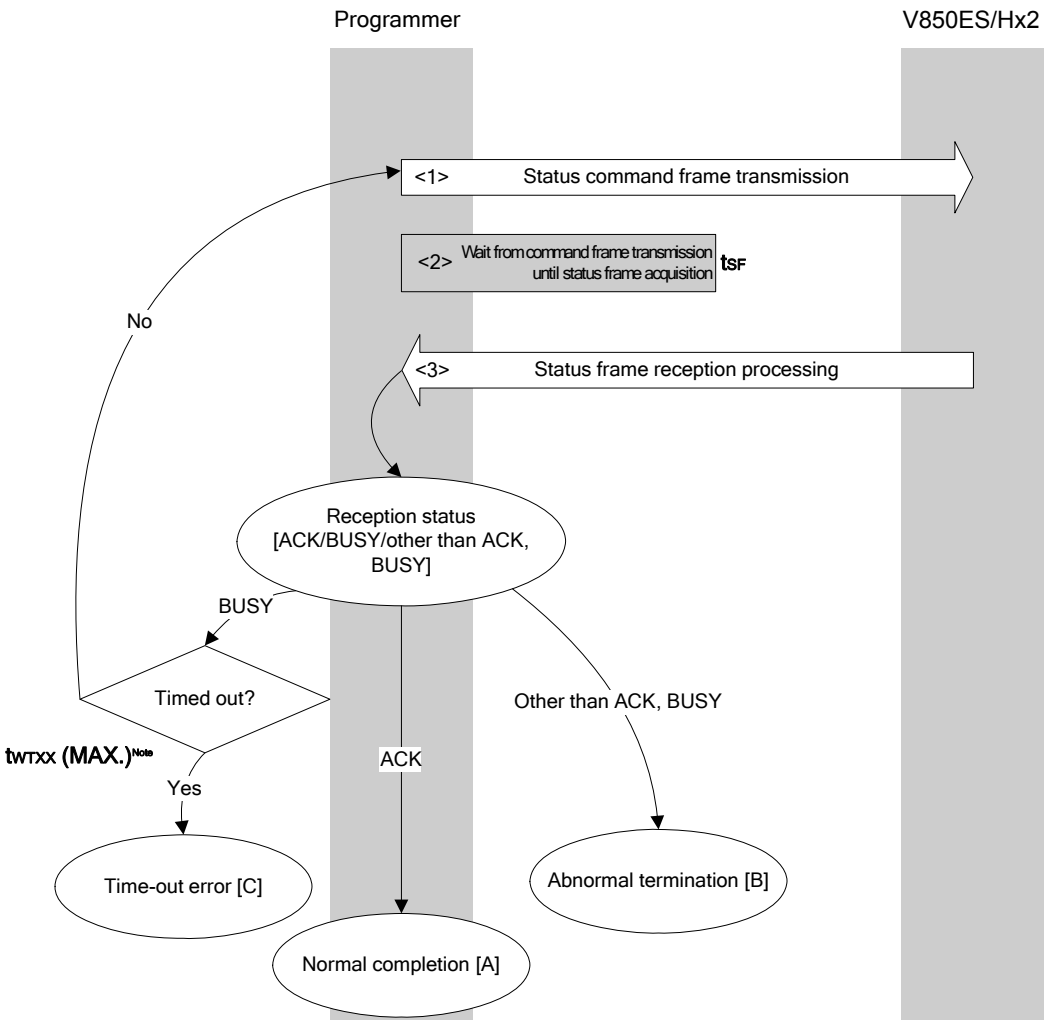
## 8.3 Data Frame Reception Processing Flowchart



## 8.4 Status Command

### 8.4.1 Processing sequence chart

Status command processing sequence




**Note** Applied specifications differ depending on the command executed.

### 8.4.2 Description of processing sequence

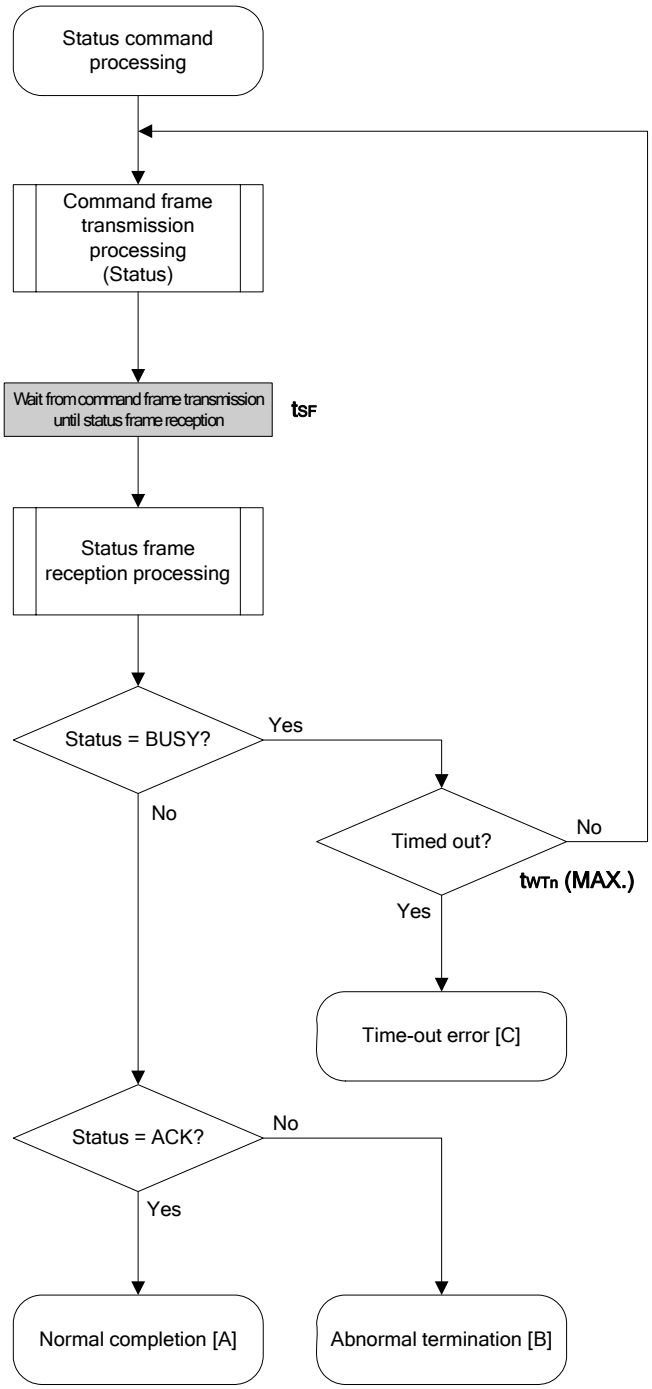
<1> The Status command is transmitted by command frame transmission processing.
<2> Waits from command transmission until status frame reception (wait time $t_{SF}$).
<3> The status code is checked.

When ST1 = ACK: Normal completion [A]
When ST1 = BUSY: A time-out check is performed. The time-out time ($t_{WTn(MAX.)}$) is given as a parameter for this processing.
If the processing is not timed out, the sequence is re-executed from <1>.
If a time-out occurs, a time-out error [C] is returned.
When ST1 ≠ ACK, BUSY: Abnormal termination [B]

### 8.4.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The status frame transmitted from the V850ES/Hx2 has been received normally. |
| Abnormal termination [B] | Command error | 04H | An unsupported command or abnormal frame has been received. |
| | Parameter error | 05H | Command information (parameter) is invalid. |
| | Checksum error | 07H | The data of the frame transmitted from the programmer is abnormal. |
| | WWV1 error | 08H | Write error |
| | EWV1 error | 0BH | Erase error |
| | EWV2 error | 0CH | |
| | EWV3 error | 0DH | |
| | Verify error | 0EH | A verify error has occurred for the data of the frame transmitted from the programmer. |
| | | 0FH | |
| | Protect error | 10H | An attempt was made to execute processing prohibited by the Security Set command. |
| | EWV4 error | 11H | Internal verify error/blank error |
| | Compaction search error | 13H | Erase error |
| | Negative acknowledgment (NACK) | 15H | Negative acknowledgment |
| | Sequencer error | 16H | A sequencer error has occurred. |
| Time-out error [C] | | – | After command transmission, the specified time has elapsed but a BUSY response is still returned. |

### 8.4.4 Flowchart



### 8.4.5 Sample program

The following shows a sample program for Status command processing.

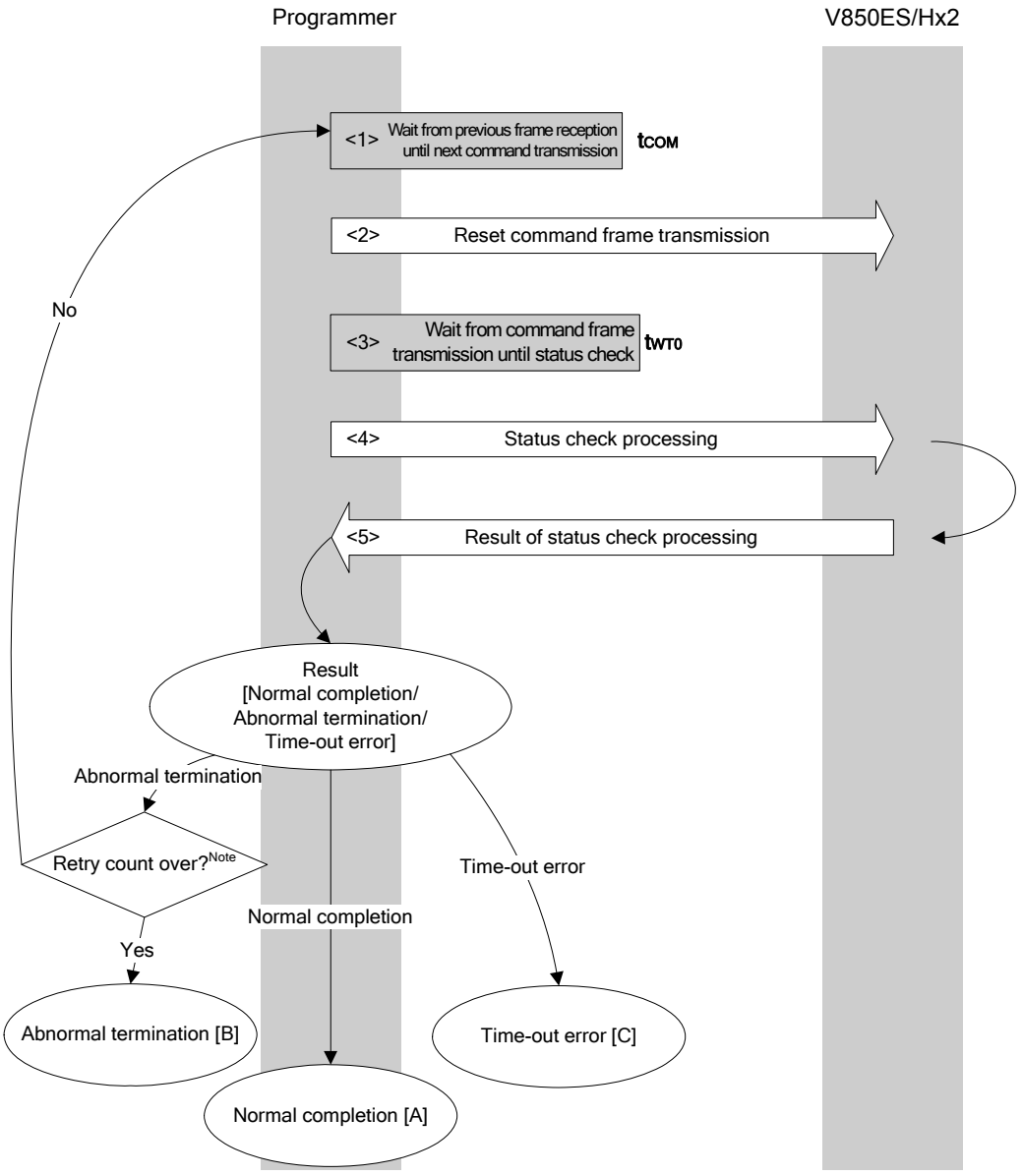
```
/************************************************************/
/*                                                          */
/* Get status command (CSI)                                 */
/*                                                          */
/************************************************************/
/* [r] u16         ... decoded status or error code         */
/*                                                          */
/* (see fl.h/fl-proto.h &                                   */
/*         definition of decode_status() in fl.c)           */
/************************************************************/
static u16 fl_csi_getstatus(u32 limit)
{
     u16    rc;

     start_flto(limit);

     while(1){

            put_cmd_csi(FL_COM_GET_STA, 1, fl_cmd_prm);    // send "Status" command frame
            fl_wait(tSF);                                  // wait

            rc = get_sfrm_csi(fl_rxdata_frm);              // get status frame

            switch(rc){
                   case   FLC_BUSY:
                          if (check_flto())                // time out ?
                                 return FLC_DFTO_ERR;      // Yes, time-out // case [C]
                          continue;                        // No, retry

                   default:                                // checksum error
                          return rc;

                   case   FLC_NO_ERR:                      // no error
                          break;

            }
            if (fl_st1 == FLST_BUSY){  // ST1 = BUSY
                   if (check_flto())          // time out ?
                          return FLC_DFTO_ERR; // Yes, time-out // case [C]
                   continue;                  // No, retry
            }

            if (fl_rxdata_frm[OFS_LEN] == 2 && fl_st1 == FLST_ACK && fl_st2 ==
            FLST_BUSY){
                   if (check_flto())          // time out ?
                          return FLC_DFTO_ERR; // Yes, time-out // case [C]
                   continue;
            }

            break;                     // ACK or other error (but BUSY)
     }

     rc = decode_status(fl_st1);       // decode status to return code
```

```
//   switch(rc) {
//
//         case   FLC_NO_ERR:   return rc;    break; // case [A]
//         default:             return rc;    break; // case [B]
//   }
     return rc;

}
```

## 8.5 Reset Command

### 8.5.1 Processing sequence chart

Reset command processing sequence




**Note** Do not exceed the retry count for the reset command transmission (up to 16 times).

### 8.5.2 Description of processing sequence

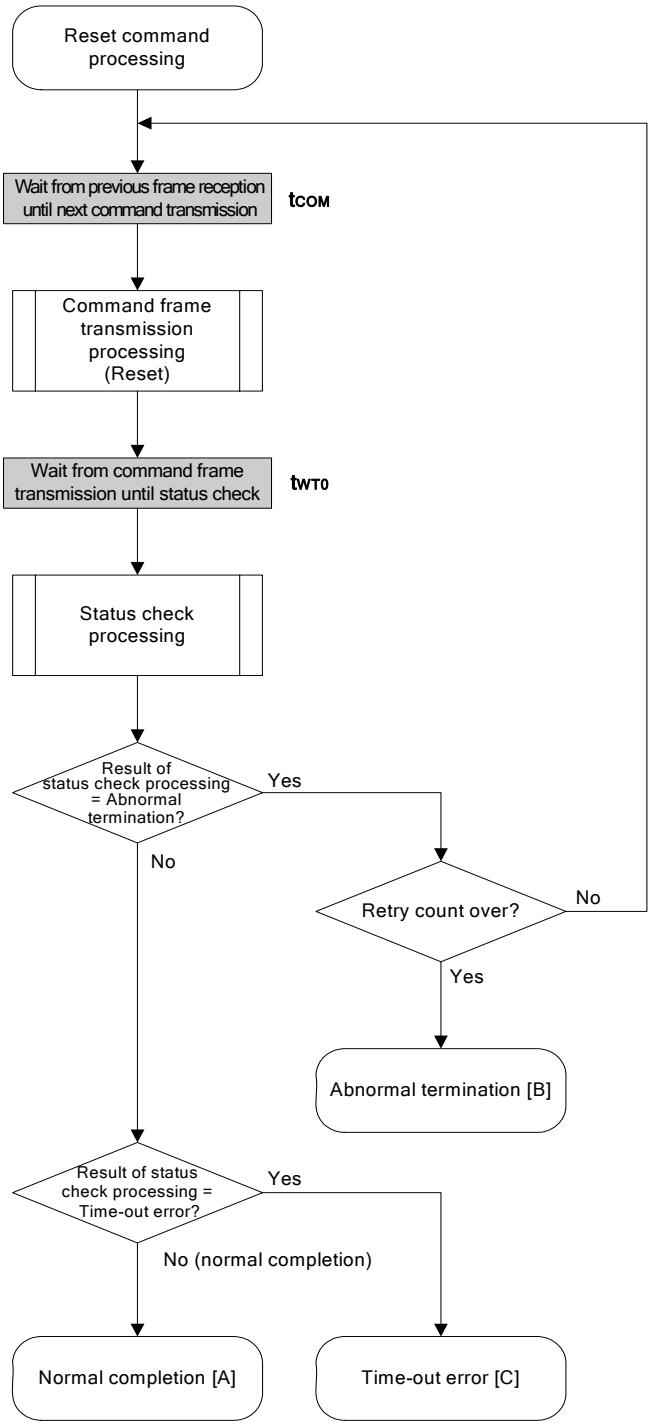
<1> Waits from the previous frame reception until the next command transmission (wait time $t_{COM}$).
<2> The Reset command is transmitted by command frame transmission processing.
<3> Waits from command transmission until status check processing (wait time $t_{WT0}$(MAX.)).
<4> The status frame is acquired by status check processing.
<5> The following processing is performed according to the result of status check processing.

When the processing ends normally: Normal completion [A]
When the processing ends abnormally: The sequence is re-executed from <1> if the retry count is not over.
If the retry count is over, the processing ends abnormally [B].
When a time-out error occurs: A time-out error [C] is returned.

### 8.5.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and synchronization between the programmer and the V850ES/Hx2 has been established. |
| Abnormal termination [B] | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | Status check processing timed out. |

### 8.5.4 Flowchart



### 8.5.5 Sample program

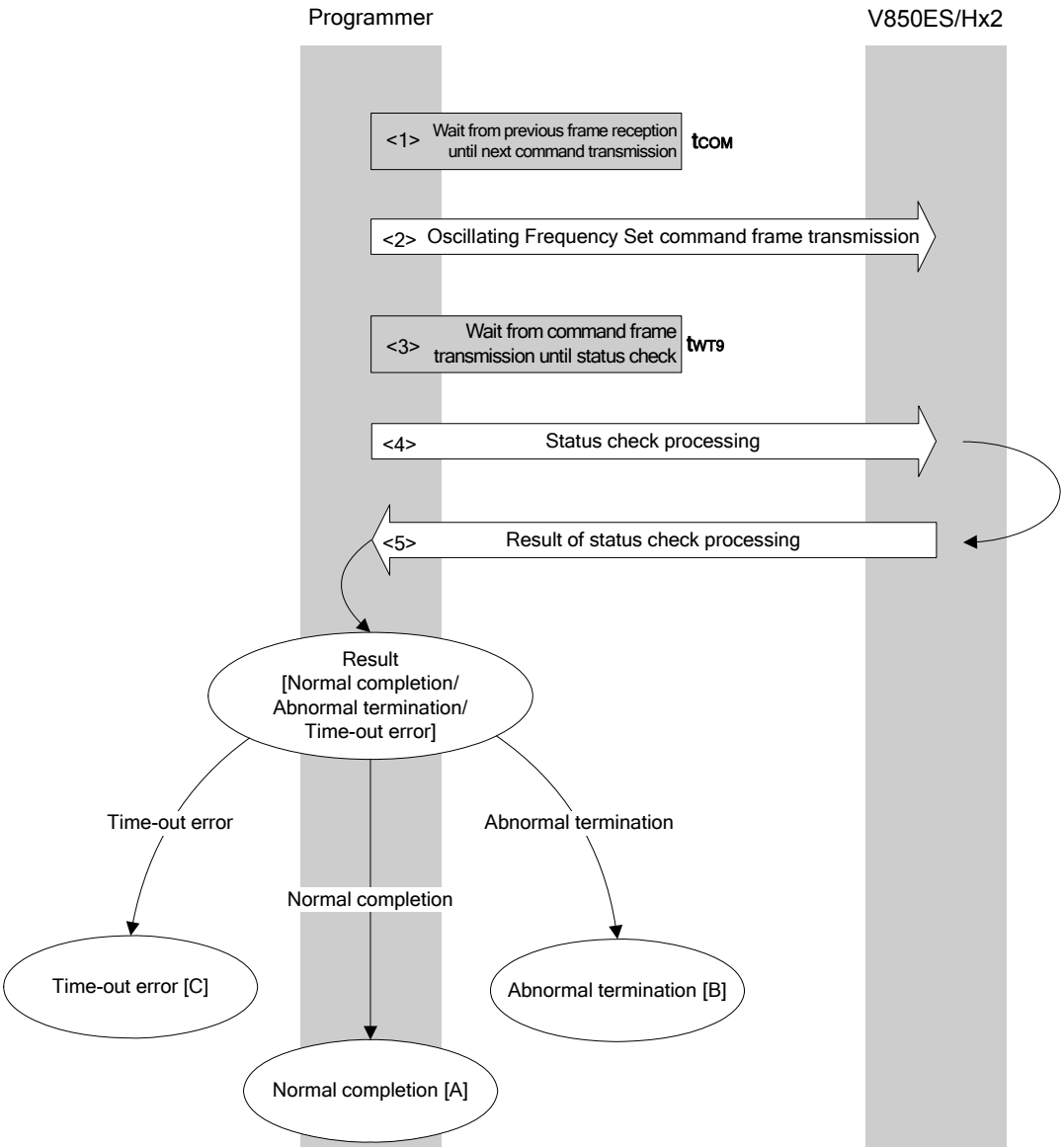
The following shows a sample program for Reset command processing.

```
/************************************************************/
/*                                                          */
/* Reset command (CSI)                                      */
/*                                                          */
/************************************************************/
/* [r] u16        ... error code                            */
/************************************************************/
u16        fl_csi_reset(void)
{
     u16    rc;
     u32    retry;

     for (retry = 0; retry < tRS; retry++){

            fl_wait(tCOM_CSI);               // wait before sending command frame

            put_cmd_csi(FL_COM_RESET, 1, fl_cmd_prm);   // send "Reset" command frame

            fl_wait(tWT0);

            rc = fl_csi_getstatus(tWT0_MAX);        // get status

            if (rc == FLC_DFTO_ERR)          // timeout error ?
                  break;                     // yes // case [C]
            if (rc == FLC_ACK)               // Ack ?
                  break;                     // yes // case [A]
            //continue;                             // case [B] (if exit from loop)
     }
//   switch(rc) {
//
//         case   FLC_NO_ERR:   return rc;    break; // case [A]
//         case   FLC_DFTO_ERR: return rc;    break; // case [C]
//         default:             return rc;    break; // case [B]
//   }
     return rc;
}
```

## 8.6 Oscillating Frequency Set Command

### 8.6.1 Processing sequence chart

Oscillating Frequency Set command processing sequence



### 8.6.2 Description of processing sequence

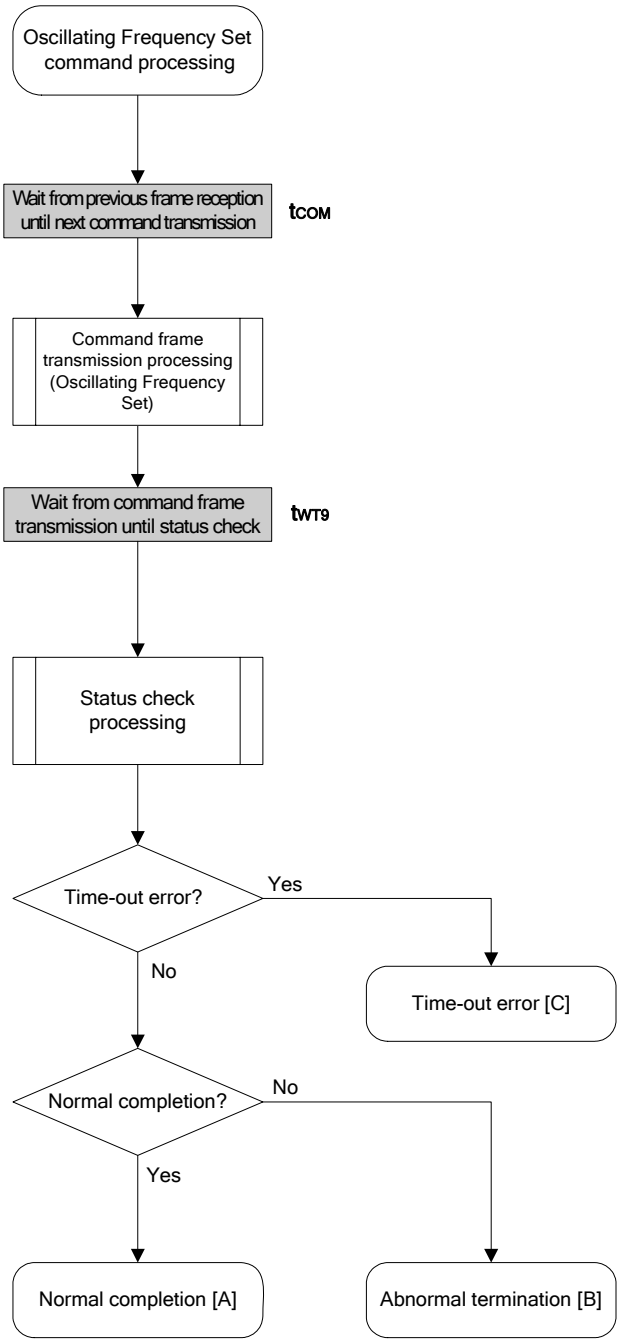
<1> Waits from the previous frame reception until the next command transmission (wait time $t_{COM}$).
<2> The Oscillating Frequency Set command is transmitted by command frame transmission processing.
<3> Waits from command transmission until status check processing (wait time $t_{WT9}$(MAX.)).
<4> The status frame is acquired by status check processing.
<5> The following processing is performed according to the result of status check processing.

When the processing ends normally: Normal completion [A]
When the processing ends abnormally: Abnormal termination [B]
When a time-out error occurs: A time-out error [C] is returned.

### 8.6.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and the operating frequency was correctly set to the V850ES/Hx2. |
| Abnormal termination [B] | Parameter error | 05H | The oscillation frequency value is out of range. |
| | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | The status frame was not received within the specified time. |

### 8.6.4 Flowchart



### 8.6.5 Sample program

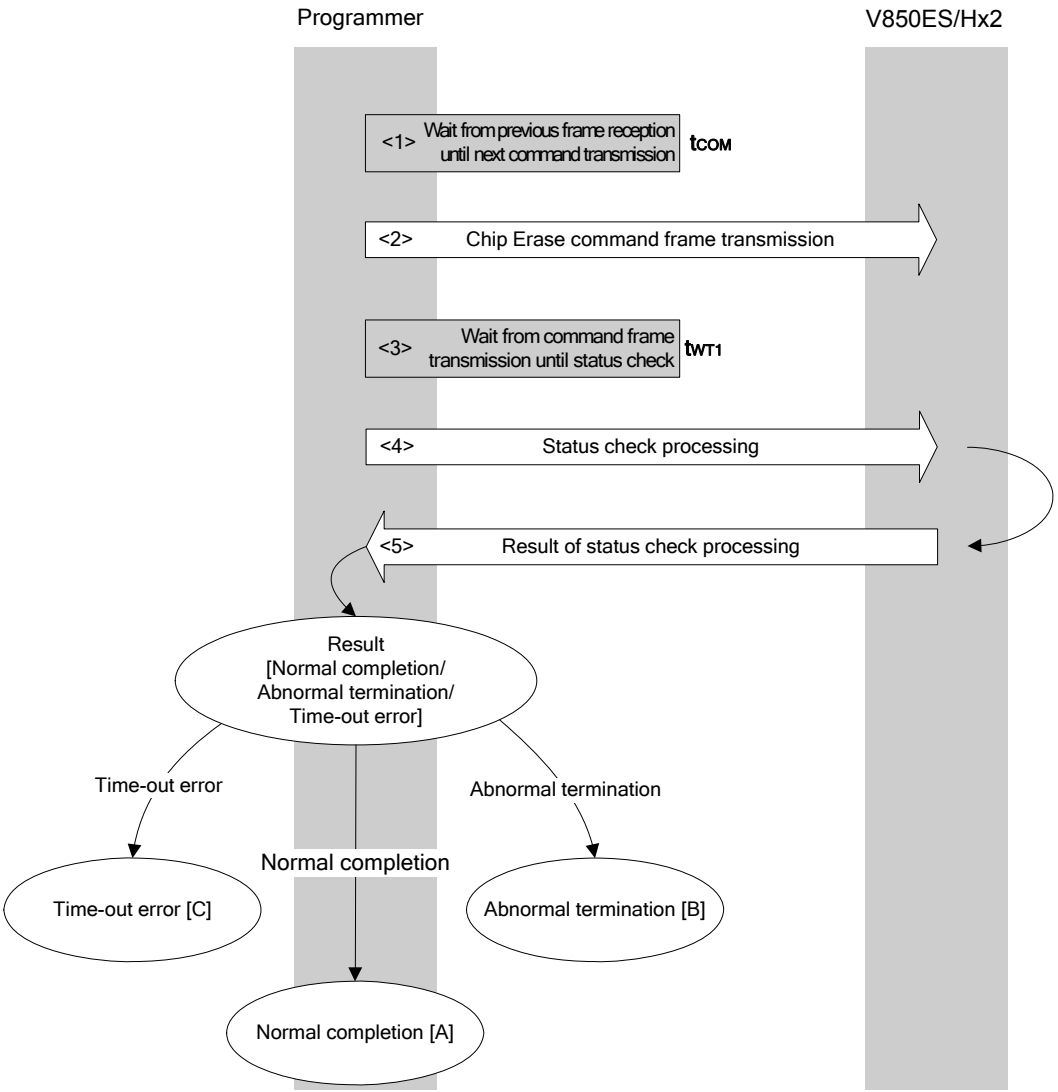
The following shows a sample program for Oscillating Frequency Set command processing.

```
/************************************************************/
/*                                                          */
/* Set Flash device clock value command (CSI)               */
/*                                                          */
/************************************************************/
/* [i] u8 clk[4]  ... frequency data(D1-D4)                 */
/* [r] u16        ... error code                            */
/************************************************************/
u16        fl_csi_setclk(u8 clk[])
{
     u16    rc;

     fl_cmd_prm[0] = clk[0];    // "D01"
     fl_cmd_prm[1] = clk[1];    // "D02"
     fl_cmd_prm[2] = clk[2];    // "D03"
     fl_cmd_prm[3] = clk[3];    // "D04"


     fl_wait(tCOM_CSI);                     // wait before sending command frame

     put_cmd_csi(FL_COM_SET_OSC_FREQ, 5, fl_cmd_prm);
                                    // send "Oscillation Frequency Set" command

     fl_wait(tWT9);

     rc = fl_csi_getstatus(tWT9_MAX);       // get status frame
//   switch(rc) {
//
//         case   FLC_NO_ERR:         return rc;    break; // case [A]
//         case   FLC_DFTO_ERR:       return rc;    break; // case [C]
//         default:                   return rc;    break; // case [B]
//   }
     return rc;
}
```

## 8.7 Chip Erase Command

### 8.7.1 Processing sequence chart

Chip Erase command processing sequence



### 8.7.2 Description of processing sequence

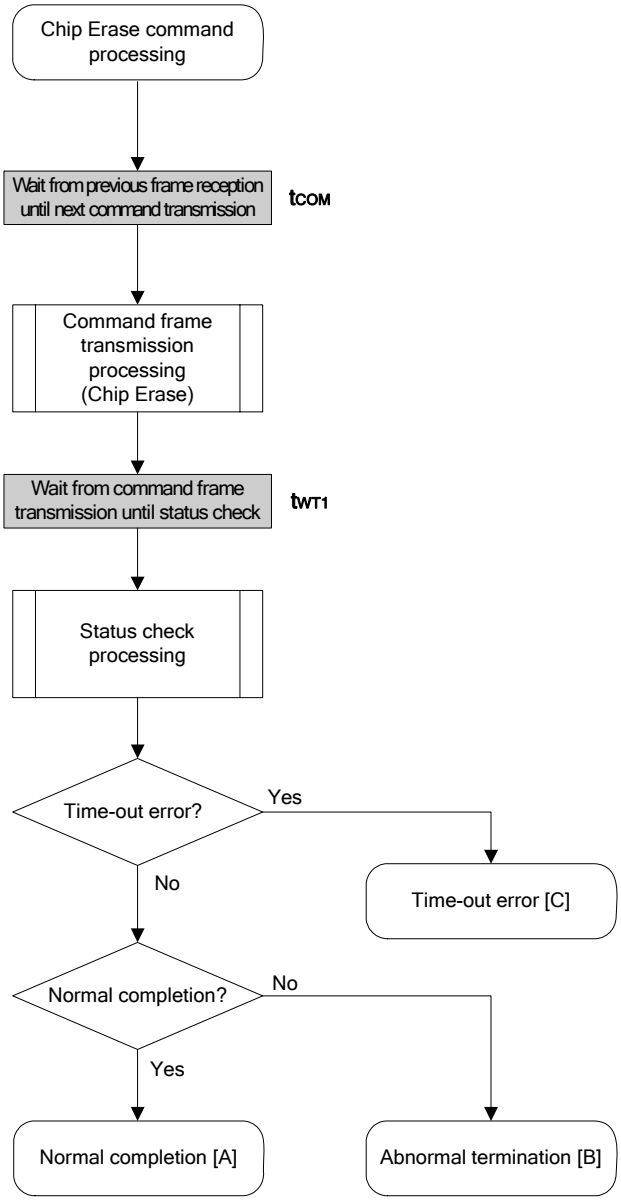
<1> Waits from the previous frame reception until the next command transmission (wait time $t_{COM}$).
<2> The Chip Erase command is transmitted by command frame transmission processing.
<3> Waits from command transmission until status check processing (wait time $t_{WT1}$(MAX.)).
<4> The status frame is acquired by status check processing.
<5> The following processing is performed according to the result of status check processing.

When the processing ends normally: Normal completion [A]
When the processing ends abnormally: Abnormal termination [B]
When a time-out error occurs: A time-out error [C] is returned.

### 8.7.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and chip erase was performed normally. |
| Abnormal termination [B] | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Protect error | 10H | Chip erase is prohibited in the security setting. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| | WWV1 error | 08H | An erase error has occurred. |
| | EWV1 error | 0BH | |
| | EWV2 error | 0CH | |
| | EWV3 error | 0DH | |
| | Compaction search error | 13H | |
| | Sequencer error | 16H | A sequencer error has occurred. |
| Time-out error [C] | | – | The status frame was not received within the specified time. |

### 8.7.4 Flowchart



### 8.7.5 Sample program

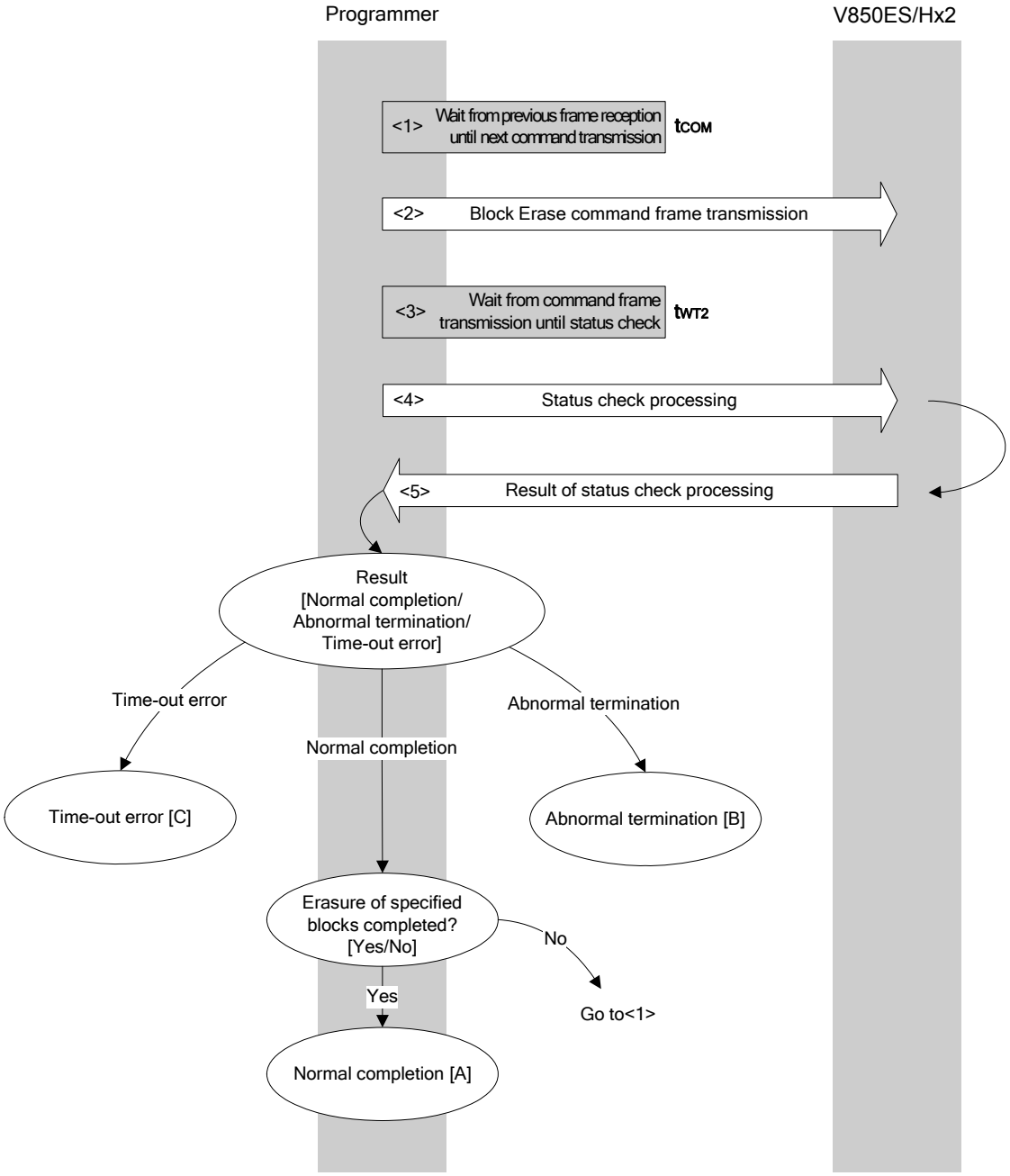
The following shows a sample program for Chip Erase command processing.

```
/************************************************************/
/*                                                          */
/* Erase all(chip) command (CSI)                            */
/*                                                          */
/************************************************************/
/* [r] u16         ... error code                           */
/************************************************************/
u16        fl_csi_erase_all(void)
{
     u16    rc;


     fl_wait(tCOM_CSI);                      // wait before sending command frame

     put_cmd_csi(FL_COM_ERASE_CHIP, 1, fl_cmd_prm); // send "Chip Erase" command

     fl_wait(tWT1);

     rc = fl_csi_getstatus(tWT1_MAX);        // get status frame
//   switch(rc) {
//
//         case   FLC_NO_ERR:         return rc;    break; // case [A]
//         case   FLC_DFTO_ERR:       return rc;    break; // case [C]
//         default:                   return rc;    break; // case [B]
//   }
     return rc;

}
```

## 8.8 Block Erase Command

### 8.8.1 Processing sequence chart

Block Erase command processing sequence



### 8.8.2 Description of processing sequence

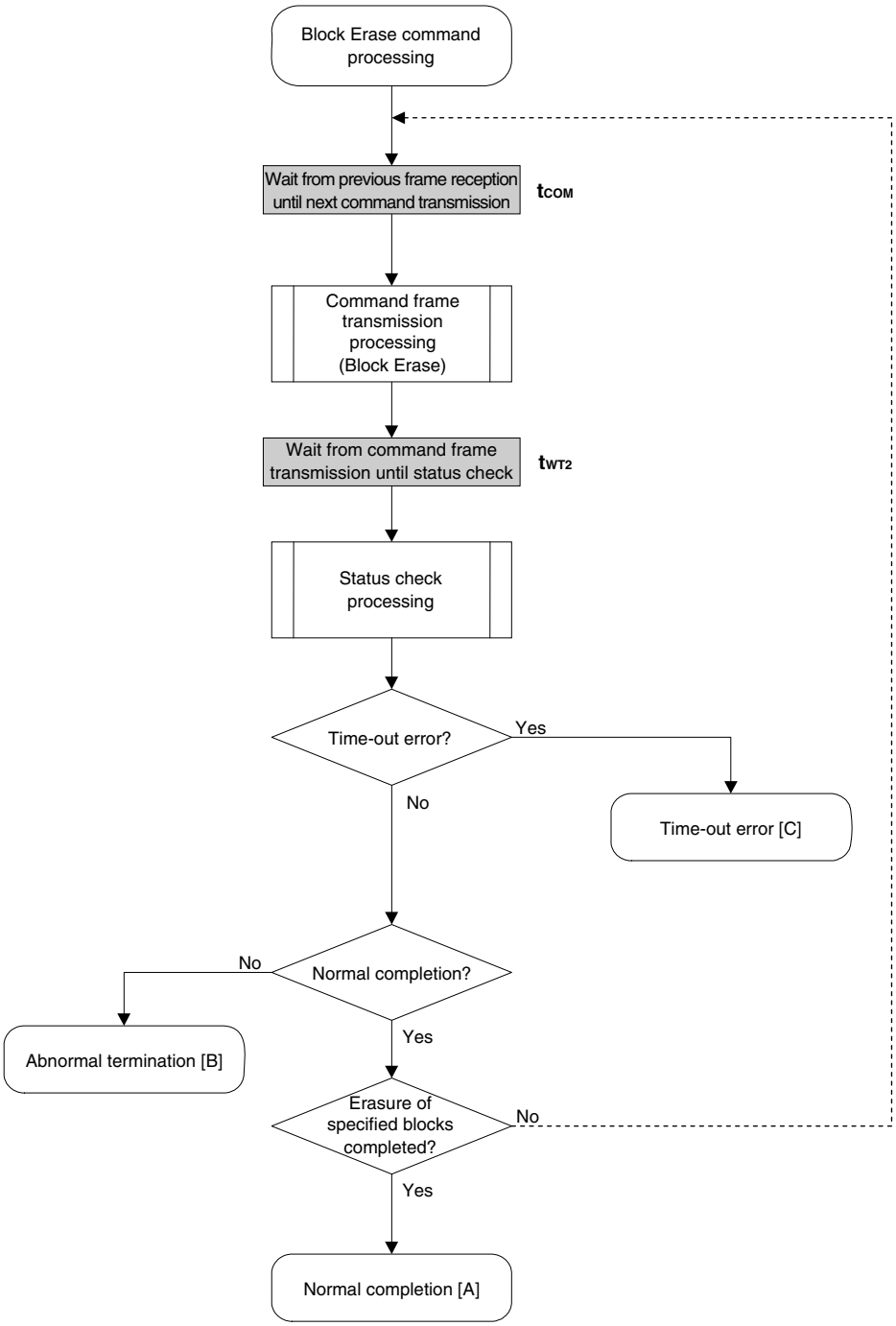
<1> Waits from the previous frame reception until the next command transmission (wait time $t_{COM}$).
<2> The Block Erase command is transmitted by command frame transmission processing.
<3> Waits until status frame acquisition (wait time $t_{WT2}$ (MAX.)).
<4> The status frame is acquired by status check processing.
<5> The following processing is performed according to the result of status check processing.

When the processing ends normally: When the block erase for all of the specified blocks is not yet completed, processing changes the block number and re-executes the sequence from <1>.
When the block erase for all of the specified blocks is completed, the processing ends normally [A].
When the processing ends abnormally: Abnormal termination [B]
When a time-out error occurs: A time-out error [C] is returned.

### 8.8.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and block erase was performed normally. |
| Abnormal termination [B] | Parameter error | 05H | The block number is out of range. |
| | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Protect error | 10H | Chip erase is prohibited in the security setting. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| | WWV1 error | 08H | An erase error has occurred. |
| | EWV1 error | 0BH | |
| | EWV2 error | 0CH | |
| | EWV3 error | 0DH | |
| | Compaction search error | 13H | |
| | Sequencer error | 16H | A sequencer error has occurred. |
| Time-out error [C] | | – | The status frame was not received within the specified time. |

### 8.8.4 Flowchart



### 8.8.5 Sample program

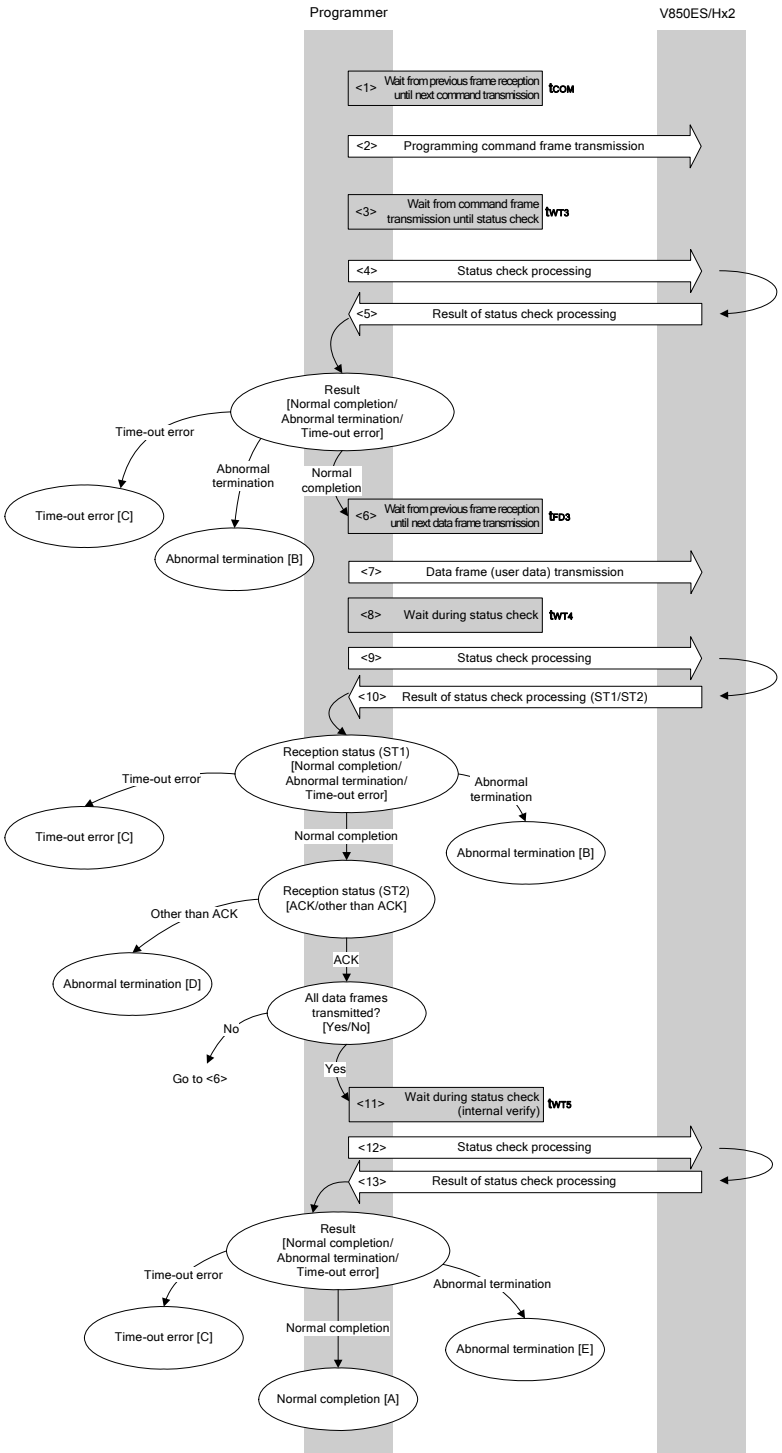
The following shows a sample program for Block Erase command processing for one block.

```
/************************************************************/
/*                                                          */
/* Erase block command (CSI)                                */
/*                                                          */
/************************************************************/
/* [i] u8 block    ... block number                         */
/* [r] u16         ... error code                           */
/************************************************************/
u16         fl_csi_erase_blk(u8 block)
{
     u16    rc;
     u32    wt2, wt2_max;

     fl_cmd_prm[0] = block;     // set params
     wt2     = get_wt2(get_block_size(block));
     wt2_max = get_wt2_max(get_block_size(block));

     fl_wait(tCOM_CSI);                      // wait before sending command frame

     put_cmd_csi(FL_COM_ERASE_BLOCK, 2, fl_cmd_prm);    // send "Block Erase" command

     fl_wait(wt2);


     rc = fl_csi_getstatus(wt2_max);  // get status frame
//   switch(rc) {
//
//         case   FLC_NO_ERR:         return rc;    break; // case [A]
//         case   FLC_DFTO_ERR:       return rc;    break; // case [C]
//         default:                   return rc;    break; // case [B]
//   }
     return rc;
}
```

## 8.9 Programming Command

### 8.9.1 Processing sequence chart

Programming command processing sequence



### 8.9.2 Description of processing sequence

<1> Waits from the previous frame reception until the next command transmission (wait time $t_{COM}$).
<2> The Programming command is transmitted by command frame transmission processing.
<3> Waits from command transmission until status check processing (wait time $t_{WT3}$ (MAX.)).
<4> The status frame is acquired by status check processing.
<5> The following processing is performed according to the result of status check processing.

When the processing ends normally: Proceeds to <6>.
When the processing ends abnormally: Abnormal termination [B]
When a time-out error occurs: A time-out error [C] is returned.

<6> Waits until the next data frame transmission (wait time $t_{FD3}$ (MAX.)).
<7> User data to be written to the V850ES/Hx2 flash memory is transmitted by data frame transmission processing.
<8> Waits from data frame (user data) transmission until status check processing (wait time $t_{WT4}$ (MAX.)).
<9> The status frame is acquired by status check processing.
<10> The following processing is performed according to the result of status check processing (status code (ST1/ST2)) (also refer to the processing sequence chart and flowchart).

When ST1 = abnormal termination: Abnormal termination [B]
When ST1 = time-out error: A time-out error [C] is returned.
When ST1 = normal completion: The following processing is performed according to the ST2 value.
- When ST2 ≠ ACK: Abnormal termination [D]
- When ST2 = ACK: Proceeds to <11> when transmission of all of the user data is completed.
  If there still remain user data to be transmitted, the processing re-executes the sequence from <6>.

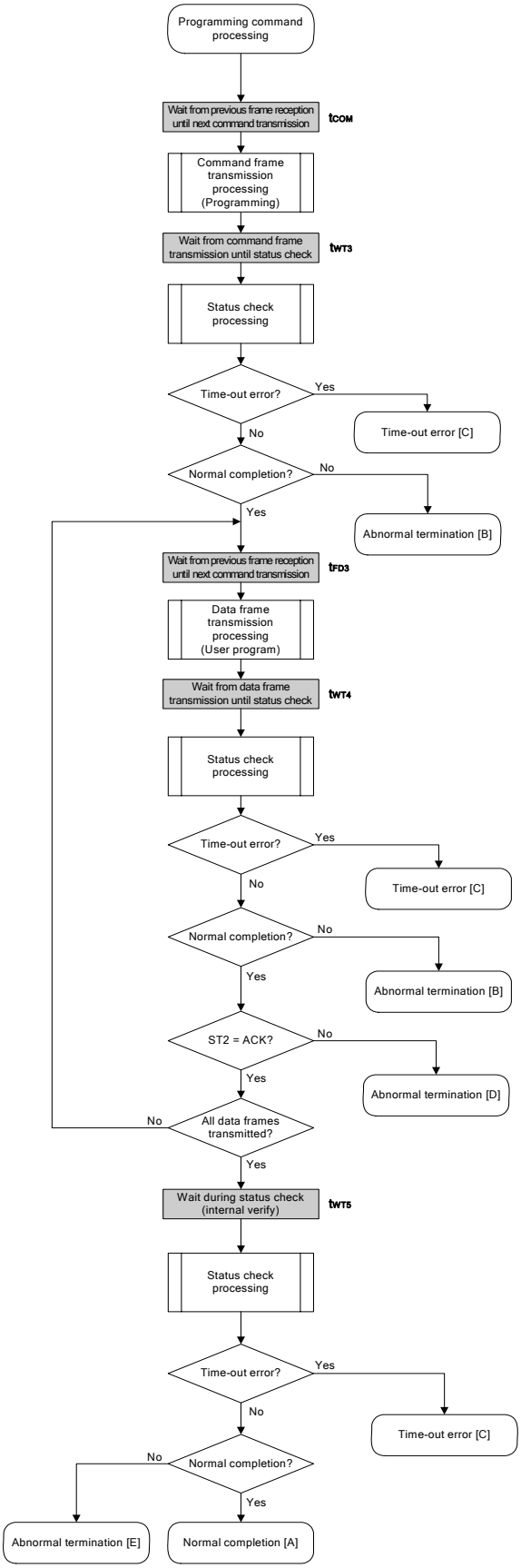
<11> Waits until status check processing (time-out time $t_{WT5}$ (MAX.)).
<12> The status frame is acquired by status check processing.
<13> The following processing is performed according to the result of status check processing.

When the processing ends normally: Normal completion [A]
(Indicating that the internal verify check has performed normally after completion of write)
When the processing ends abnormally: Abnormal termination [E]
(Indicating that the internal verify check has not performed normally after completion of write)
When a time-out error occurs: A time-out error [C] is returned.

### 8.9.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and the user data was written normally. |
| Abnormal termination [B] | Parameter error | 05H | The specified start/end address is not the start/end address of the block. |
| | Checksum error | 07H | The checksum of the transmitted command frame or data frame does not match. |
| | Protect error | 10H | Write is prohibited in the security setting. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | The status frame was not received within the specified time. |
| Abnormal termination [D] | WWV1 error | 08H (ST2) | A write error has occurred. |
| | Sequencer error | 16H | A sequencer error has occurred. |
| Abnormal termination [E] | EWV4 error | 11H | An internal verify error has occurred. |
| | Sequencer error | 16H | A sequencer error has occurred. |

### 8.9.4 Flowchart



### 8.9.5 Sample program

The following shows a sample program for Programming command processing.

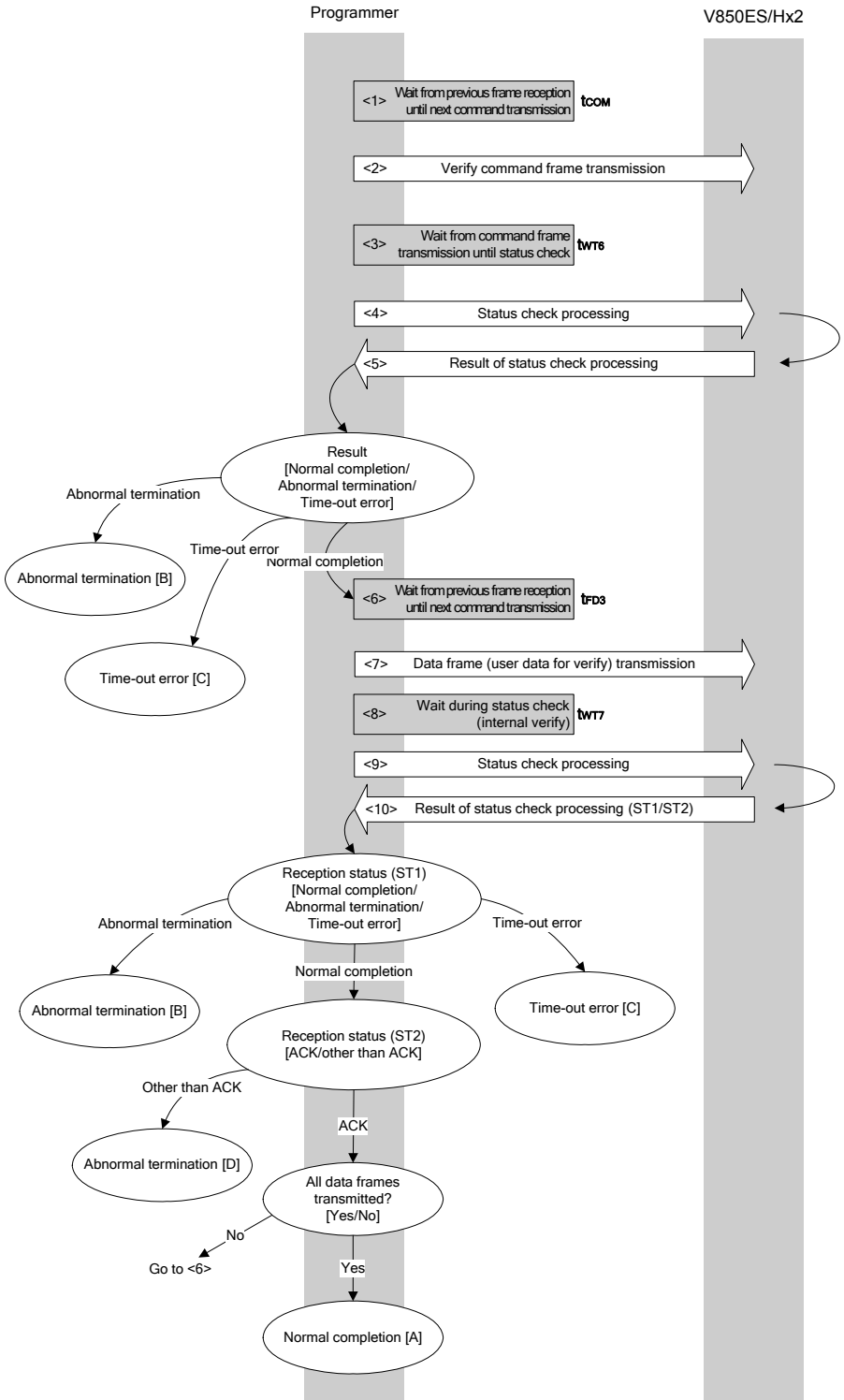
```
/************************************************************/
/*                                                          */
/* Write command (CSI)                                      */
/*                                                          */
/************************************************************/
/* [i] u32 top     ... start address                        */
/* [i] u32 bottom  ... end address                          */
/* [r] u16         ... error code                           */
/************************************************************/
u16        fl_csi_write(u32 top, u32 bottom)
{
     u16    rc;
     u32    send_head, send_size;
     bool   is_end;
     u32    wt5, wt5_max;

     // set params
     set_range_prm(fl_cmd_prm, top, bottom); // set SAH/SAM/SAL, EAH/EAM/EAL
     wt5     = get_wt5(bottom - top + 1);
     wt5_max = get_wt5_max(bottom - top + 1);


     /*********************************************/
     /*      send command & check status          */
     /*********************************************/

     fl_wait(tCOM_CSI);
     put_cmd_csi(FL_COM_WRITE, 7, fl_cmd_prm);      // send "Programming" command
     fl_wait(tWT3);

     rc = fl_csi_getstatus(tWT3_MAX);               // get status frame
     switch(rc) {
           case   FLC_NO_ERR:                       break; // continue
     //    case   FLC_DFTO_ERR:       return rc;    break; // case [C]
           default:                   return rc;    break; // case [B]
     }

     /*********************************************/
     /*      send user data                       */
     /*********************************************/
     send_head = top;

     while(1){

           if ((bottom - send_head) > 256){         // rest size > 256 ?
                 is_end = false;                    // yes, not end frame
                 send_size  = 256;                  // transmit size = 256 byte
           }
           else{
```

```
                is_end = true;
                send_size = bottom - send_head + 1;
                                // transmit size = (bottom - send_head)+1 byte
        }

        memcpy(fl_txdata_frm, rom_buf+send_head, send_size);
                                                    // set data frame payload
        send_head += send_size;


        fl_wait(tFD3);                          // wait before sending data frame
        put_dfrm_csi(send_size, fl_txdata_frm, is_end);
                                                // send data frame (user data)
        fl_wait(tWT4);                          // wait


        rc = fl_csi_getstatus(tWT4_MAX);        // get status frame
        switch(rc) {
                case   FLC_NO_ERR:                      break; // continue
        //      case   FLC_DFTO_ERR:       return rc;   break; // case [C]
                default:                   return rc;   break; // case [B]
        }
        if (fl_st2 != FLST_ACK){                // ST2 = ACK ?
                rc = decode_status(fl_st2);     // No
                return rc;                      // case [D]
        }

        if (is_end)                         // send all user data ?
                break;                      // yes
        //continue;
    }
    /*********************************************/
    /*     Check internally verify                */
    /*********************************************/

    fl_wait(wt5);                           // wait

    rc = fl_csi_getstatus(wt5_max);             // get status frame
//  switch(rc) {
//          case   FLC_NO_ERR:         return rc;   break; // case [A]
//          case   FLC_DFTO_ERR:       return rc;   break; // case [C]
//          default:                   return rc;   break; // case [E]
//  }
    return rc;

}
```

## 8.10 Verify Command

### 8.10.1 Processing sequence chart

Verify command processing sequence



### 8.10.2 Description of processing sequence

<1> Waits from the previous frame reception until the next command transmission (wait time $t_{COM}$).
<2> The Verify command is transmitted by command frame transmission processing.
<3> Waits from command transmission until status check processing (wait time $t_{WT6}$(MAX.)).
<4> The status frame is acquired by status check processing.
<5> The following processing is performed according to the result of status check processing.

When the processing ends normally: Proceeds to <6>.
When the processing ends abnormally: Abnormal termination [B]
When a time-out error occurs: A time-out error [C] is returned.

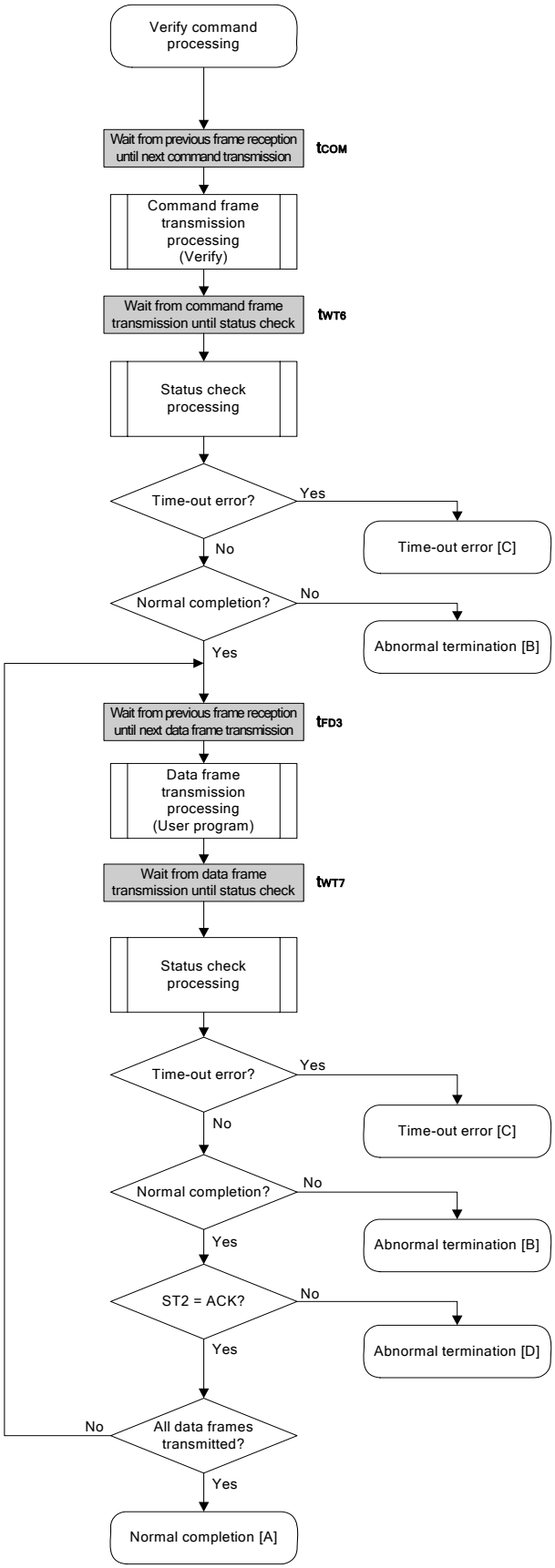
<6> Waits from the previous frame reception until the next data frame transmission (wait time $t_{FD3}$).
<7> User data for verifying is transmitted by data frame transmission processing.
<8> Waits from data frame transmission until status check processing (wait time $t_{WT7}$(MAX.)).
<9> The status frame is acquired by status check processing.
<10> The following processing is performed according to the result of status check processing (status code (ST1/ST2)) (also refer to the processing sequence chart and flowchart).

When ST1 = abnormal termination: Abnormal termination [B]
When ST1 = time-out error: A time-out error [C] is returned.
When ST1 = normal completion: The following processing is performed according to the ST2 value.
- When ST2 ≠ ACK: Abnormal termination [D]
- When ST2 = ACK: If transmission of all data frames is completed, the processing ends normally [A].
  If there still remain data frames to be transmitted, the processing re-executes the sequence from <6>.

### 8.10.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and the verify was completed normally. |
| Abnormal termination [B] | Parameter error | 05H | The specified start/end address is not the start/end address of the block. |
| | Checksum error | 07H | The checksum of the transmitted command frame or data frame does not match. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | The status frame was not received within the specified time. |
| Abnormal termination [D] | Verify error | 0FH (ST2) | The verify has failed, or another error has occurred. |
| | | 0EH | |
| | Sequencer error | 16H | A sequencer error has occurred. |

### 8.10.4 Flowchart



### 8.10.5 Sample program

The following shows a sample program for Verify command processing.

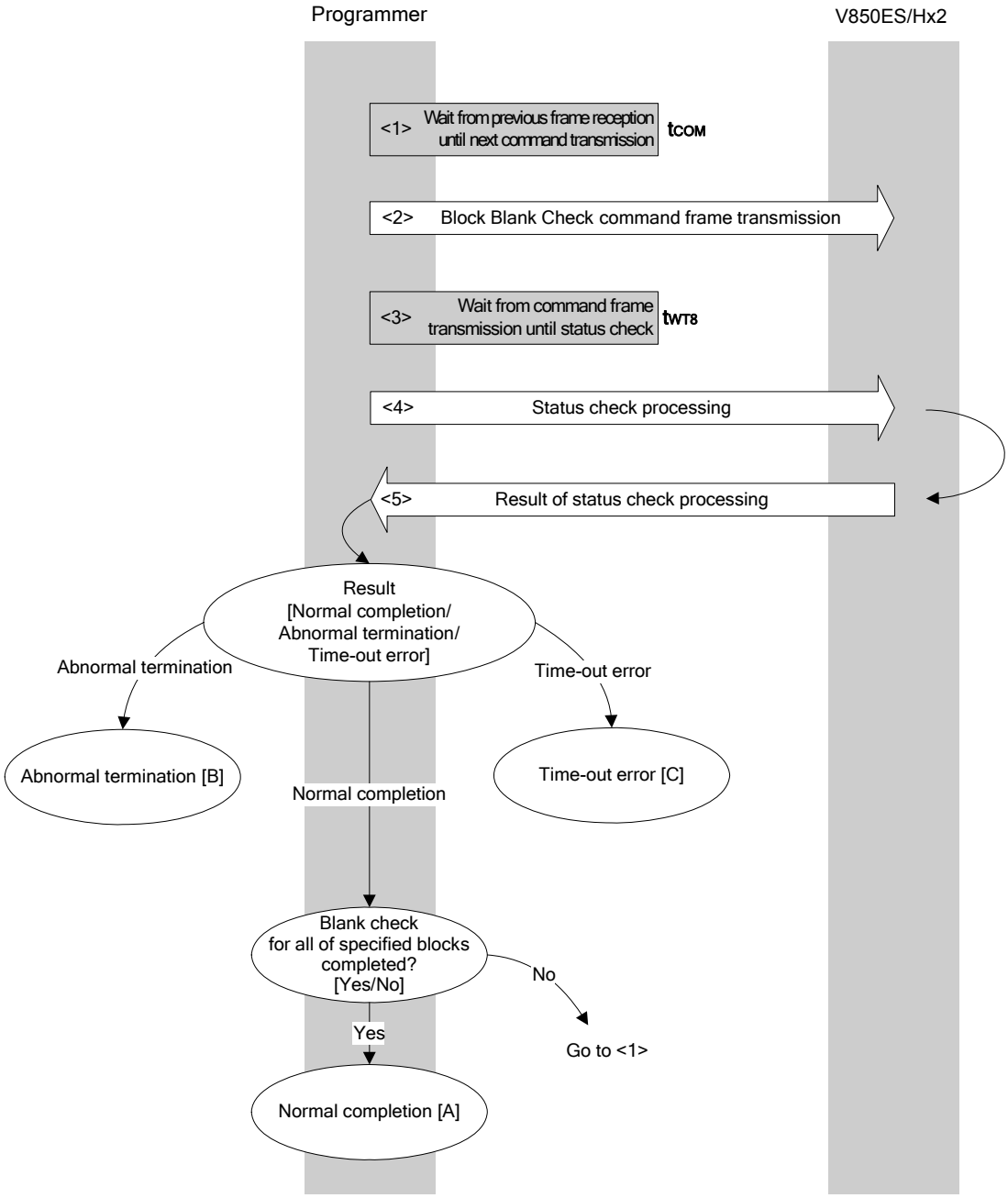
```
/************************************************************/
/*                                                          */
/*   Verify command (CSI)                                   */
/*                                                          */
/************************************************************/
/*   [i] u32 top   ... start address                        */
/*   [i] u32 bottom      ... end address                    */
/*   [i] u8  *buf  ... pointer to verify data buffer        */
/*   [r] u16             ... error code                     */
/************************************************************/
u16  fl_csi_verify(u32 top, u32 bottom, u8 *buf)
{
     u16    rc;
     u32    send_head, send_size;
     bool   is_end;

     // set params
     set_range_prm(fl_cmd_prm, top, bottom); // set SAH/SAM/SAL, EAH/EAM/EAL


     /*********************************************/
     /*     send command & check status           */
     /*********************************************/
     fl_wait(tCOM_CSI);
     put_cmd_csi(FL_COM_VERIFY, 7, fl_cmd_prm);     // send "Verify" command
     fl_wait(tWT6);

     rc = fl_csi_getstatus(tWT6_MAX);        // get status frame
     switch(rc) {
            case   FLC_NO_ERR:               break; // continue
     //     case   FLC_DFTO_ERR: return rc;  break; // case [C]
            default:             return rc;  break; // case [B]
     }


     /*********************************************/
     /*     send user data                        */
     /*********************************************/
     send_head = top;

     while(1){

            if ((bottom - send_head) > 256){        // rest size > 256 ?
                   is_end = false;                  // yes, not end frame
                   send_size  = 256;                // transmit size = 256 byte
```

```
            }
            else{
                  is_end = true;
                  send_size = bottom - send_head + 1;     // transmit size = (bottom
- send_head)+1 byte

            }

            memcpy(fl_txdata_frm, buf+send_head, send_size);     //  set  data  frame
payload
            send_head += send_size;

            fl_wait(tFD3);                          // wait before sending data frame
            put_dfrm_csi(send_size, fl_txdata_frm, is_end);      // send data frame
            fl_wait(tWT7);                          // wait

            rc = fl_csi_getstatus(tWT7_MAX);        // get status frame
            switch(rc) {
                  case   FLC_NO_ERR:                break; // continue
            //    case   FLC_DFTO_ERR: return rc;   break; // case [C]
                  default:             return rc;   break; // case [B]
            }
            if (fl_st2 != FLST_ACK){                // ST2 = ACK ?
                  rc = decode_status(fl_st2);       // No
                  return rc;                        // case [D]
            }

            if (is_end)               // send all user data ?
                  break;              // yes
            //continue;

      }
      return FLC_NO_ERR;  // case [A]

  }
```

## 8.11 Block Blank Check Command

### 8.11.1 Processing sequence chart

Block Blank Check command processing sequence



### 8.11.2 Description of processing sequence

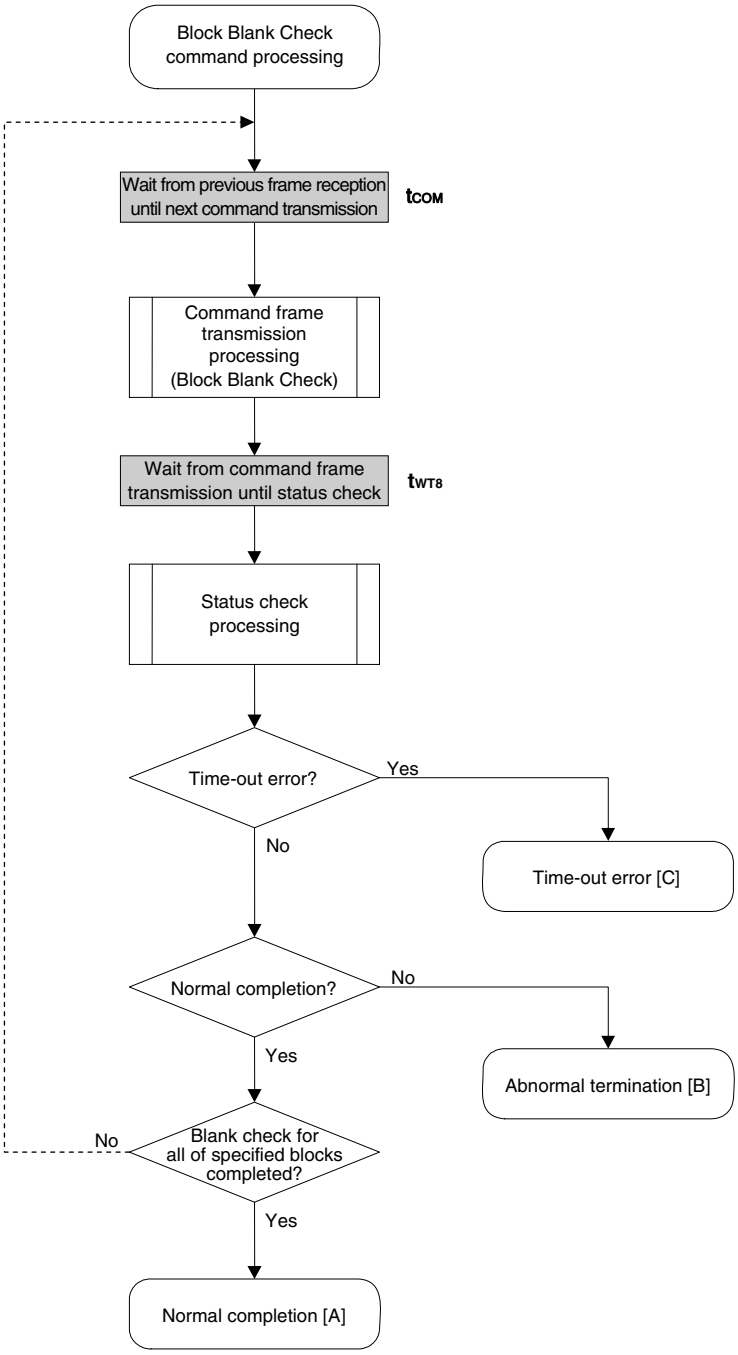
<1> Waits from the previous frame reception until the next command transmission (wait time $t_{COM}$).
<2> The Block Blank Check command is transmitted by command frame transmission processing.
<3> Waits from command transmission until status check processing (wait time $t_{WT8}$ (MAX.)).
<4> The status frame is acquired by status check processing.
<5> The following processing is performed according to the result of status check processing.

When a time-out error occurs: A time-out error [C] is returned.
When the processing ends abnormally: Abnormal termination [B]
When the processing ends normally: If the blank check for all of the specified blocks is not yet completed, processing changes the block number and re-executes the sequence from <1>.
If the blank check for all of the specified blocks is completed, the processing ends normally [A].

### 8.11.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and all of the specified blocks are blank. |
| Abnormal termination [B] | Parameter error | 05H | The block number is out of range. |
| | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| | EWV4 error | 11H | The specified block in the flash memory is not blank. |
| | Sequencer error | 16H | A sequencer error has occurred. |
| Time-out error [C] | | – | The status frame was not received within the specified time. |

### 8.11.4 Flowchart



### 8.11.5 Sample program

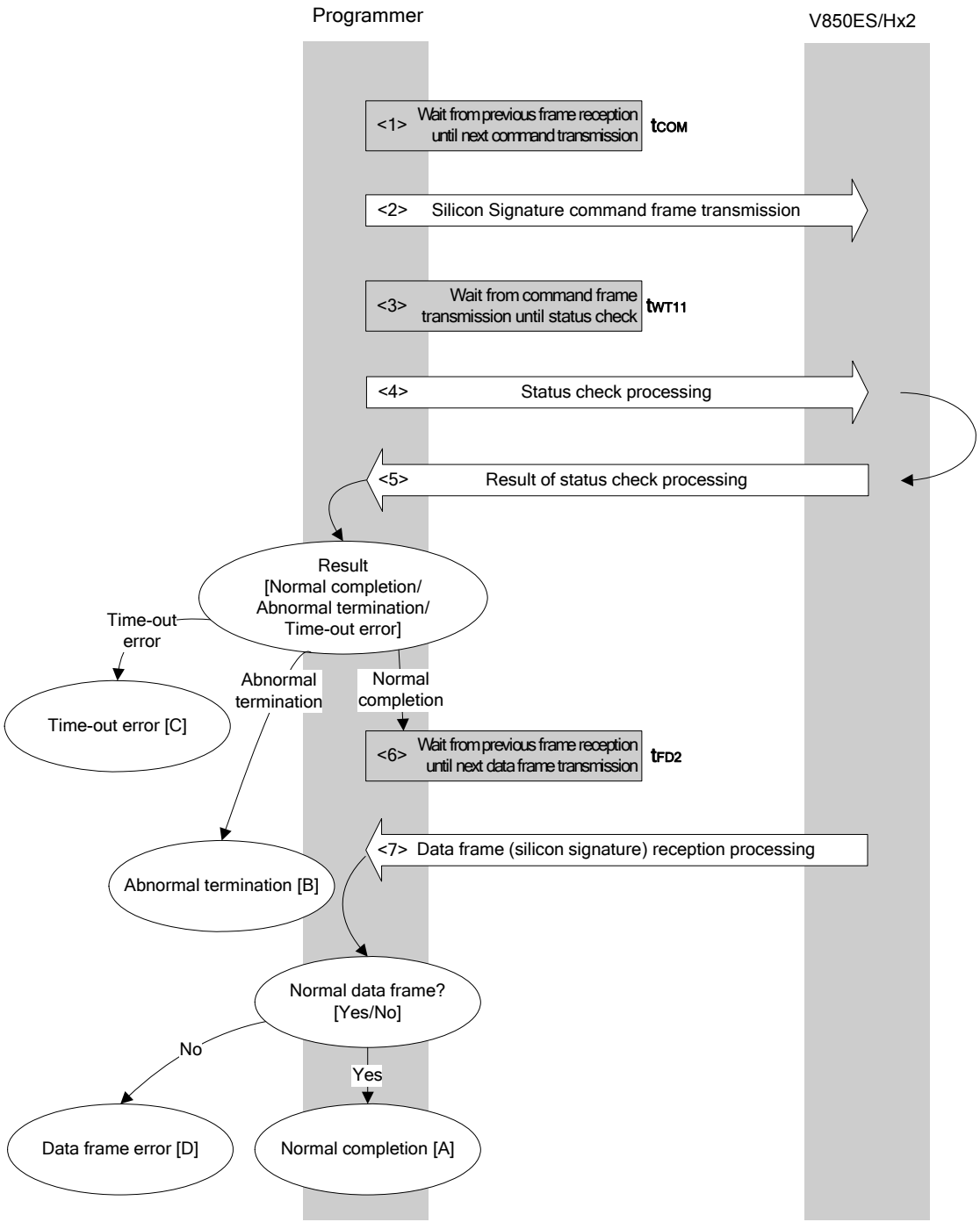
The following shows a sample program for Block Blank Check command processing for one block.

```
/************************************************************/
/*                                                          */
/* Block blank check command (CSI)                          */
/*                                                          */
/************************************************************/
/* [i]  u8 block  ... block number                          */
/* [r] u16        ... error code                            */
/************************************************************/
u16        fl_csi_blk_blank_chk(u8 block)
{
     u16    rc;
     u32    wt8, wt8_max;

     fl_cmd_prm[0] = block;     // "BLK"
     wt8     = get_wt8(get_block_size(block));
     wt8_max = get_wt8_max(get_block_size(block));


     fl_wait(tCOM_CSI);                      // wait before sending command frame

     put_cmd_csi(FL_COM_BLOCK_BLANK_CHK, 2, fl_cmd_prm);
                                             // send "Block Blank Check" command

     fl_wait(wt8);

     rc = fl_csi_getstatus(wt8_max);   // get status frame
//   switch(rc) {
//
//         case   FLC_NO_ERR:         return rc;    break; // case [A]
//         case   FLC_DFTO_ERR:       return rc;    break; // case [C]
//         default:                   return rc;    break; // case [B]
//   }
     return rc;
}
```

## 8.12 Silicon Signature Command

### 8.12.1 Processing sequence chart

Silicon Signature command processing sequence



### 8.12.2 Description of processing sequence

<1> Waits from the previous frame reception until the next command transmission (wait time $t_{COM}$).
<2> The Silicon Signature command is transmitted by command frame transmission processing.
<3> Waits from command transmission until status check processing (wait time $t_{WT11(MAX.)}$).
<4> The status frame is acquired by status check processing.
<5> The following processing is performed according to the result of status check processing.

When the processing ends normally: Proceeds to <6>.
When the processing ends abnormally: Abnormal termination [B]
When a time-out error occurs: A time-out error [C] is returned.

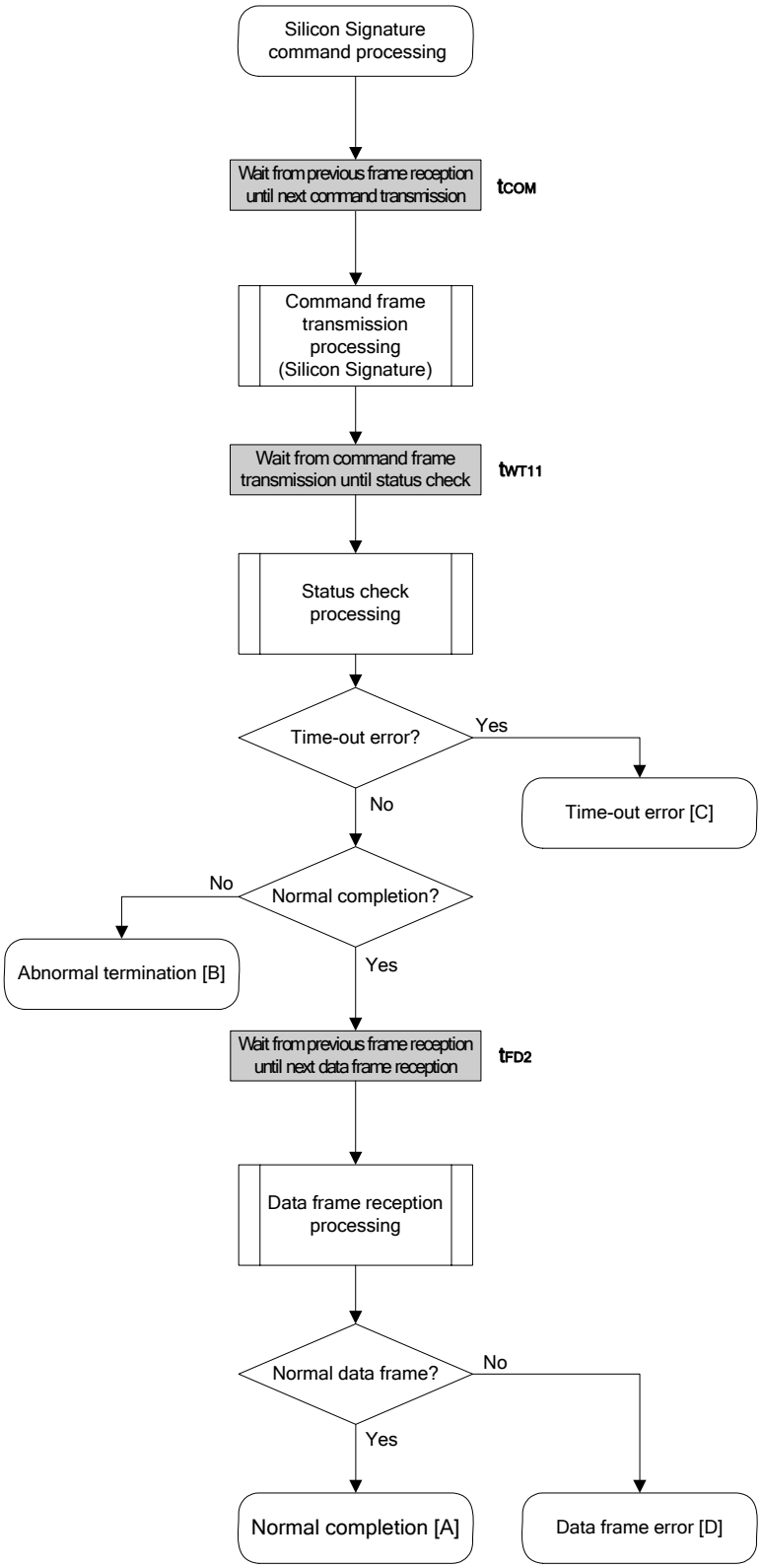
<6> Waits from the previous frame reception until the next command transmission (wait time $t_{FD2}$).
<7> The received data frame (silicon signature data) is checked.

If data frame is normal: Normal completion [A]
If data frame is abnormal: Data frame error [D]

### 8.12.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and the silicon signature was acquired normally. |
| Abnormal termination [B] | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | The status frame was not received within the specified time. |
| Data frame error [D] | | – | The checksum of the data frame received as silicon signature data does not match. |

### 8.12.4 Flowchart



### 8.12.5 Sample program

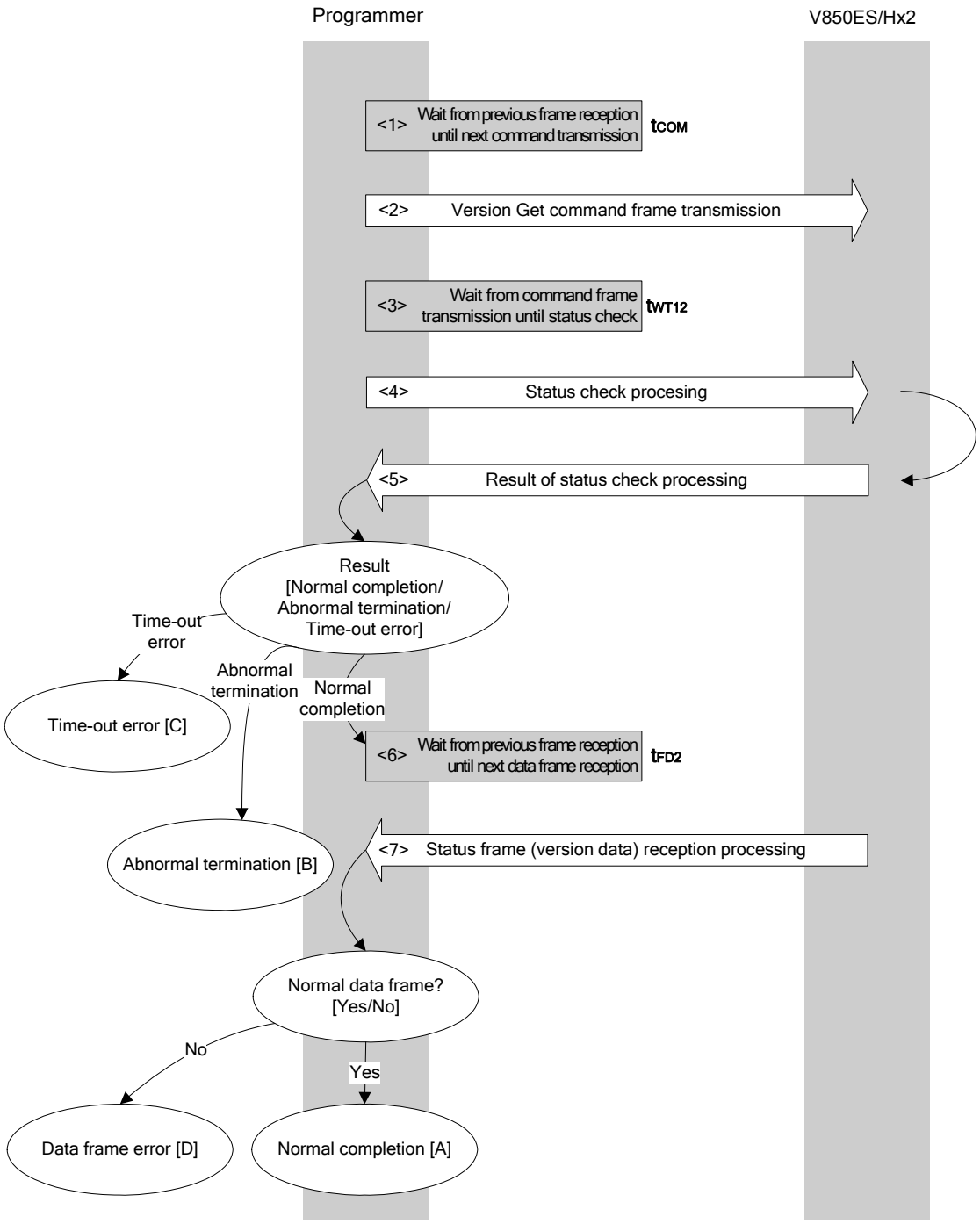
The following shows a sample program for Silicon Signature command processing.

```
/************************************************************/
/*                                                          */
/* Get silicon signature command (CSI)                      */
/*                                                          */
/************************************************************/
/* [i] u8 *sig    ... pointer to signature save area        */
/* [r] u16        ... error code                            */
/************************************************************/
u16        fl_csi_getsig(u8 *sig)
{
     u16    rc;

     fl_wait(tCOM_CSI);                      // wait before sending command frame

     put_cmd_csi(FL_COM_GET_SIGNATURE, 1, fl_cmd_prm);
                                             // send "Silicon Signature" command

     fl_wait(tWT11);

     rc = fl_csi_getstatus(tWT11_MAX);       // get status frame
     switch(rc) {
           case   FLC_NO_ERR:                        break; // continue
     //    case   FLC_DFTO_ERR:       return rc;     break; // case [C]
           default:                   return rc;     break; // case [B]
     }

     fl_wait(tFD2_SIG);                      // wait before getting data frame

     rc = get_dfrm_csi(fl_rxdata_frm);       // get data frame (signature data)

     if (rc){                                                // if no error,
           return rc;                        // case [D]
     }
     memcpy(sig, fl_rxdata_frm+OFS_STA_PLD, fl_rxdata_frm[OFS_LEN]);
                                                             // copy Signature data
     return rc;                              // case [A]
}
```

## 8.13 Version Get Command

### 8.13.1 Processing sequence chart

Version Get command processing sequence



### 8.13.2 Description of processing sequence

<1> Waits from the previous frame reception until the next command transmission (wait time $t_{COM}$).
<2> The Version Get command is transmitted by command frame transmission processing.
<3> Waits from command transmission until status check processing (wait time $t_{WT12(MAX.)}$).
<4> The status frame is acquired by status check processing.
<5> The following processing is performed according to the result of status check processing.

When the processing ends normally: Proceeds to <6>.
When the processing ends abnormally: Abnormal termination [B]
When a time-out error occurs: A time-out error [C] is returned.

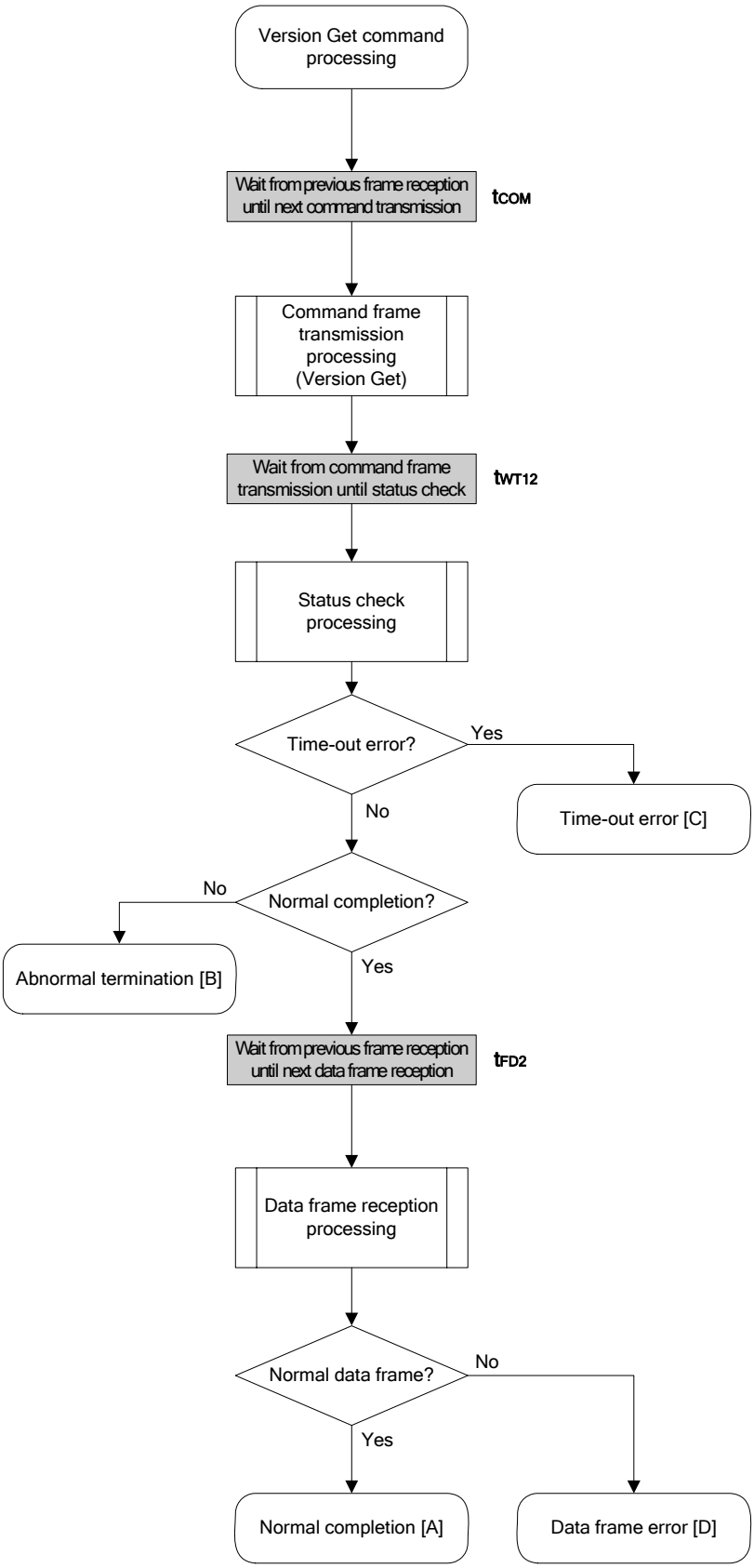
<6> Waits from the previous frame reception until the next command transmission (wait time $t_{FD2}$).
<7> The received data frame (version data) is checked.

If data frame is normal: Normal completion [A]
If data frame is abnormal: Data frame error [D]

### 8.13.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and version data was acquired normally. |
| Abnormal termination [B] | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | The status frame was not received within the specified time. |
| Data frame error [D] | | – | The checksum of the data frame received as version data does not match. |

### 8.13.4 Flowchart



### 8.13.5 Sample program

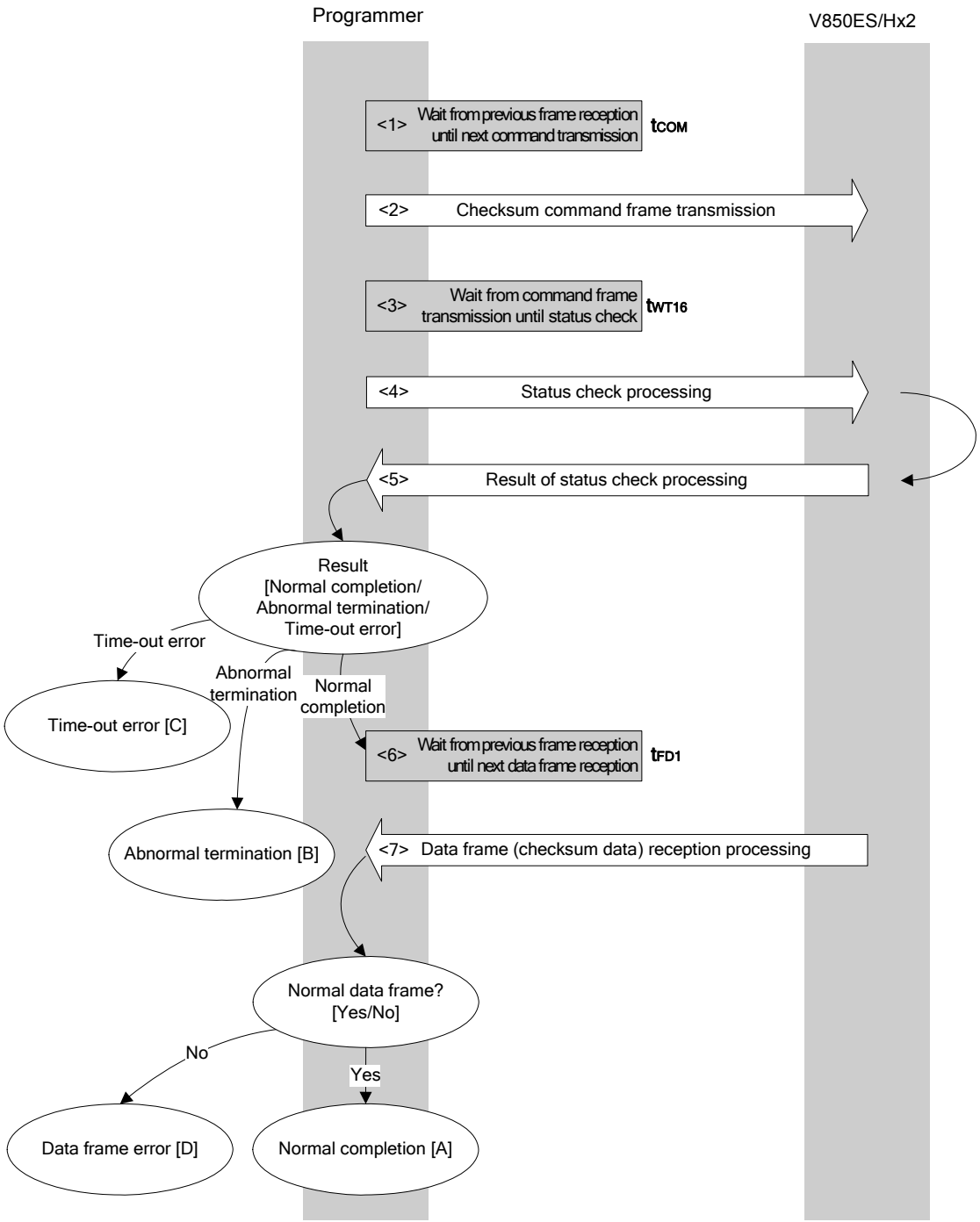
The following shows a sample program for Version Get command processing.

```
/************************************************************/
/*                                                          */
/* Get device/firmware version command (CSI)                */
/*                                                          */
/************************************************************/
/* [i] u8 *buf    ... pointer to version date save area     */
/* [r] u16        ... error code                            */
/************************************************************/
u16        fl_csi_getver(u8 *buf)
{
     u16    rc;


     fl_wait(tCOM_CSI);                      // wait before sending command frame

     put_cmd_csi(FL_COM_GET_VERSION, 1, fl_cmd_prm);    // send "Version Get" command

     fl_wait(tWT12);

     rc = fl_csi_getstatus(tWT12_MAX);       // get status frame
     switch(rc) {
           case   FLC_NO_ERR:                      break; // continue
     //    case   FLC_DFTO_ERR:       return rc;   break; // case [C]
           default:                   return rc;   break; // case [B]
     }

     fl_wait(tFD2_VG);                       // wait before getting data frame

     rc = get_dfrm_csi(fl_rxdata_frm);       // get version data

     if (rc){                                            // if no error,
           return rc;                        // case [D]
     }
     memcpy(buf, fl_rxdata_frm+OFS_STA_PLD, DFV_LEN);// copy version data
     return rc;                              // case [A]
}
```

## 8.14 Checksum Command

### 8.14.1 Processing sequence chart

Checksum command processing sequence



### 8.14.2 Description of processing sequence

<1> Waits from the previous frame reception until the next command transmission (wait time $t_{COM}$).
<2> The Checksum command is transmitted by command frame transmission processing.
<3> Waits from command transmission until status check processing (wait time $t_{WT16(MAX.)}$).
<4> The status frame is acquired by status check processing.
<5> The following processing is performed according to the result of status check processing.

When the processing ends normally: Proceeds to <6>.
When the processing ends abnormally: Abnormal termination [B]
When a time-out error occurs: A time-out error [C] is returned.

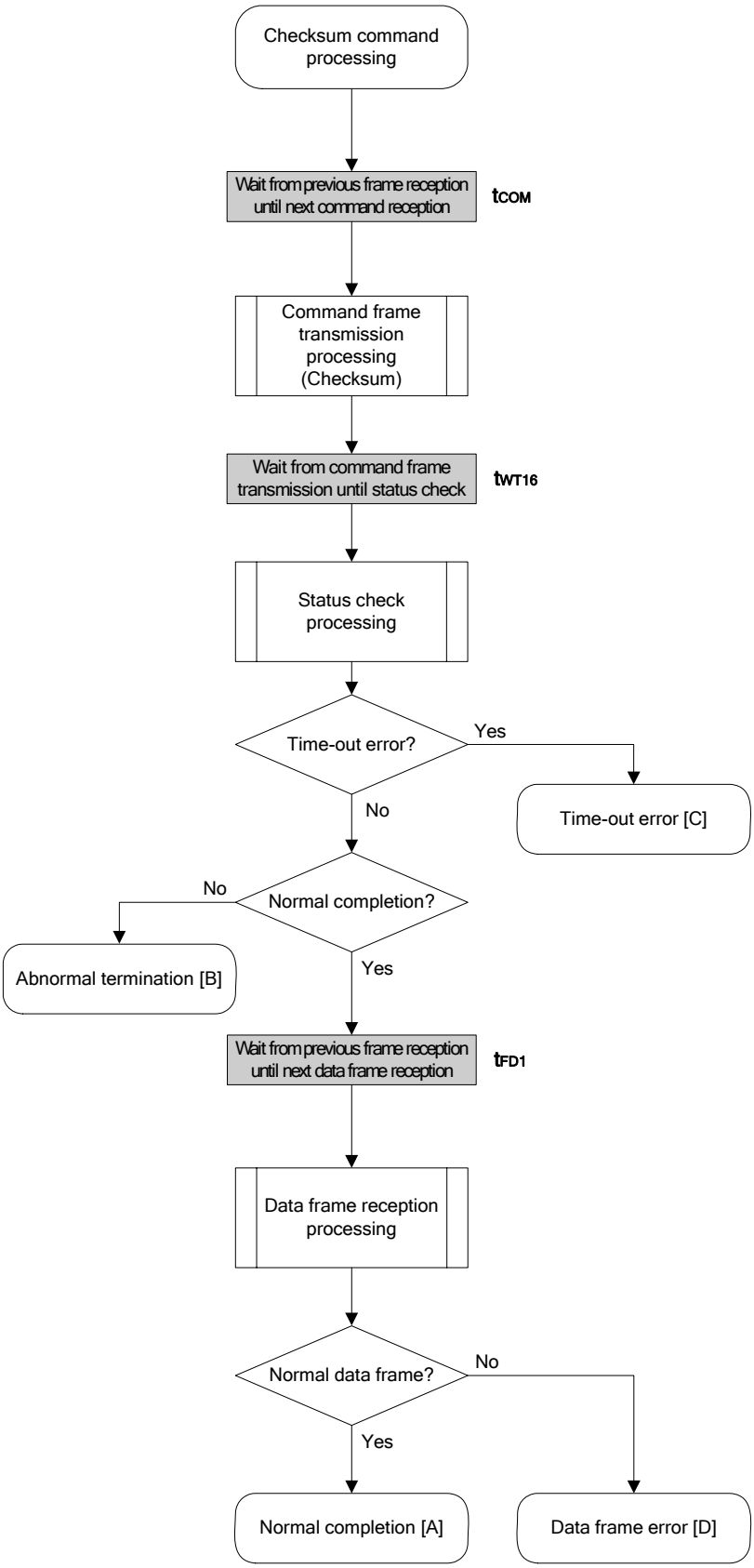
<6> Waits from the previous frame reception until the next command transmission (wait time $t_{FD1}$).
<7> The received data frame (checksum data) is checked.

If data frame is normal: Normal completion [A]
If data frame is abnormal: Data frame error [D]

### 8.14.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and checksum data was acquired normally. |
| Abnormal termination [B] | Parameter error | 05H | The specified start/end address is not the start/end address of the block. |
| | Checksum error | 07H | The checksum of the transmitted command frame does not match. |
| | Negative acknowledgment (NACK) | 15H | • A command other than the Status command was received during processing.<br>• Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | The status frame was not received within the specified time. |
| Data frame error [D] | | – | The checksum of the data frame received as version data does not match. |

### 8.14.4 Flowchart



### 8.14.5 Sample program

The following shows a sample program for Checksum command processing.

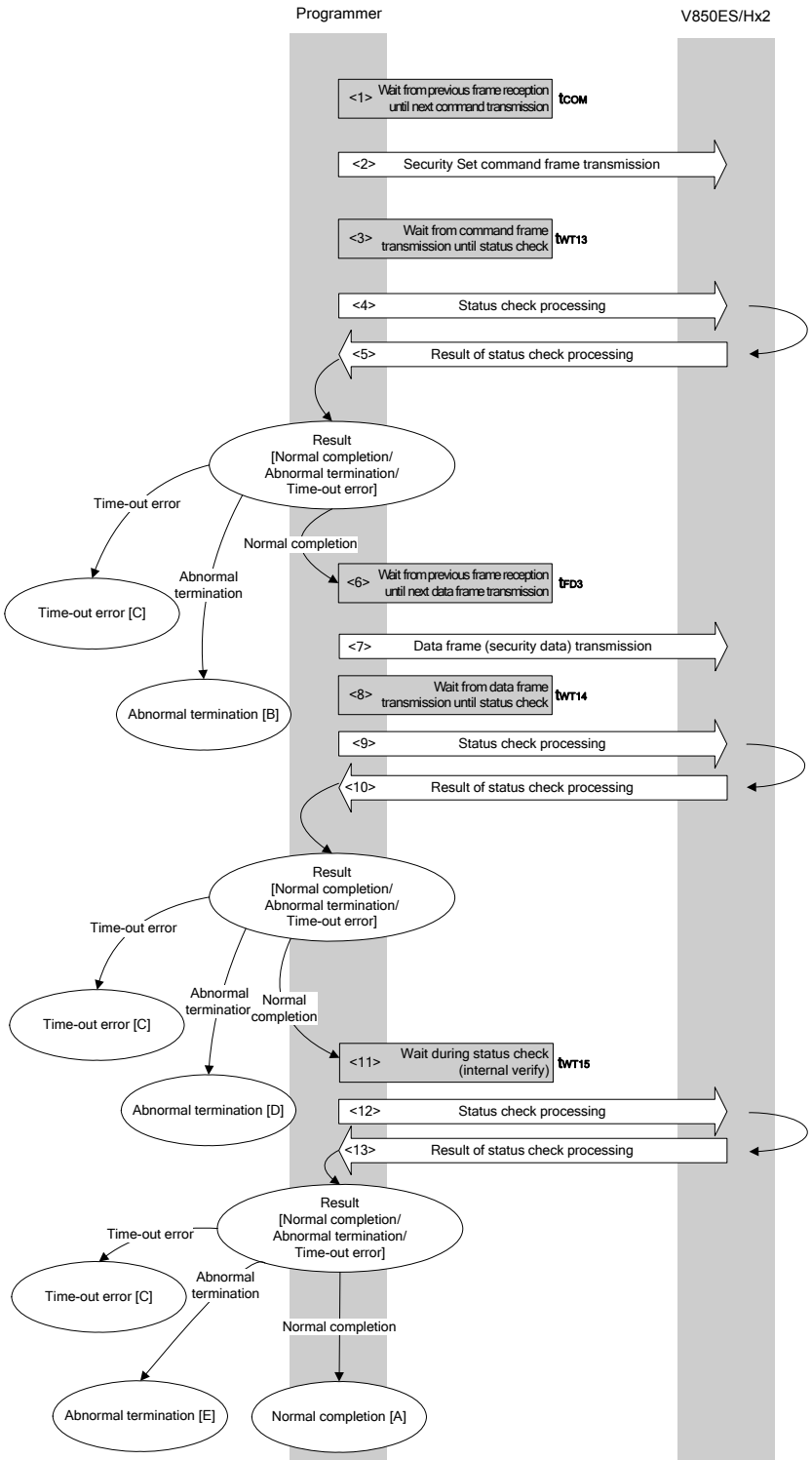
```
/************************************************************/
/*                                                          */
/* Get checksum command (CSI)                               */
/*                                                          */
/************************************************************/
/* [i] u16 *sum    ... pointer to checksum save area        */
/* [i] u32 top     ... start address                        */
/* [i] u32 bottom  ... end address                          */
/* [r] u16         ... error code                           */
/************************************************************/
u16         fl_csi_getsum(u16 *sum, u32 top, u32 bottom)
{
     u16    rc;
     u32    fd1;

     /*********************************************/
     /*      set params                           */
     /*********************************************/
     // set params
     set_range_prm(fl_cmd_prm, top, bottom); // set SAH/SAM/SAL, EAH/EAM/EAL
     fd1 = get_fd1(bottom - top + 1);

     /*********************************************/
     /*      send command                         */
     /*********************************************/
     fl_wait(tCOM_CSI);                      // wait before sending command frame

     put_cmd_csi(FL_COM_GET_CHECK_SUM, 7, fl_cmd_prm);     // send "Checksum" command

     fl_wait(tWT16);

     rc = fl_csi_getstatus(tWT16_MAX);       // get status frame
     switch(rc) {
           case   FLC_NO_ERR:                break; // continue
     //    case   FLC_DFTO_ERR: return rc;   break; // case [C]
           default:             return rc;   break; // case [B]
     }


     /*********************************************/
     /*      get data frame (Checksum data)       */
     /*********************************************/
     fl_wait(fd1);

     rc = get_dfrm_csi(fl_rxdata_frm); // get data frame(version data)
```

```
    if (rc){                          // if error,
          return rc;                  // case [D]
    }

    *sum = (fl_rxdata_frm[OFS_STA_PLD] << 8) + fl_rxdata_frm[OFS_STA_PLD+1];
                                                              // set SUM data
    return rc;                        // case [A]
}
```

## 8.15 Security Set Command

### 8.15.1 Processing sequence chart

Security Set command processing sequence



### 8.15.2 Description of processing sequence

<1> Waits from the previous frame reception until the next command transmission (wait time $t_{COM}$).
<2> The Security Set command is transmitted by command frame transmission processing.
<3> Waits from command transmission until status check processing (wait time $t_{WT13}$ (MAX.)).
<4> The status frame is acquired by status check processing.
<5> The following processing is performed according to the result of status check processing.

When the processing ends normally: Proceeds to <6>.
When the processing ends abnormally: Abnormal termination [B]
When a time-out error occurs: A time-out error [C] is returned.

<6> Waits from the previous frame reception until the data frame transmission (wait time $t_{FD3}$).
<7> The data frame (security setting data) is transmitted by data frame transmission processing.
<8> Waits from data frame transmission until status check processing (wait time $t_{WT14}$ (MAX.)).
<9> The status frame is acquired by status check processing.
<10> The following processing is performed according to the result of status check processing.

When the processing ends normally: Proceeds to <11>.
When the processing ends abnormally: Abnormal termination [D]
When a time-out error occurs: A time-out error [C] is returned.

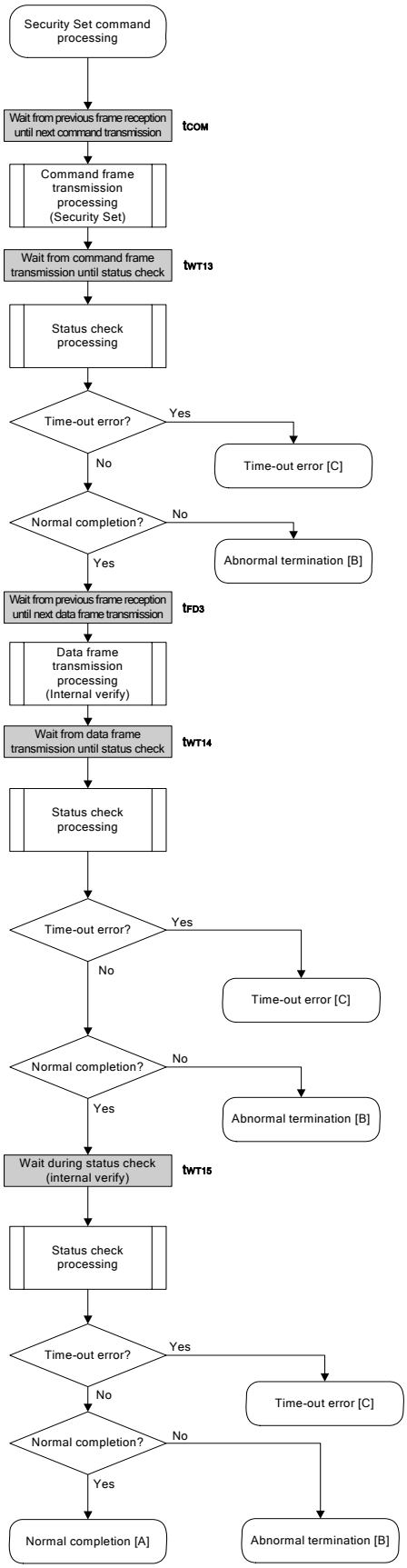
<11> Waits until status acquisition (completion of internal verify) (wait time $t_{WT15}$ (MAX.)).
<12> The status frame is acquired by status check processing.
<13> The following processing is performed according to the result of status check processing.

When the processing ends normally: Normal completion [A]
When the processing ends abnormally: Abnormal termination [E]
When a time-out error occurs: A time-out error [C] is returned.

### 8.15.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and security setting was performed normally. |
| Abnormal termination [B] | Parameter error | 05H | The command information (parameter) is not 00H. |
| | Checksum error | 07H | The checksum of the transmitted command frame or data frame does not match. |
| | Protect error | 10H | The ID codes do not match. |
| | Negative acknowledgment (NACK) | 15H | Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | The status frame was not received within the specified time. |
| Abnormal termination [D] | WWV1 error | 08H | • Security data is already set.<br>• A security data write error has occurred. |
| | Sequencer error | 16H | A sequencer error has occurred. |
| Abnormal termination [E] | EWV4 error | 11H | An internal verify error has occurred. |
| | Sequencer error | 16H | A sequencer error has occurred. |

### 8.15.4 Flowchart



### 8.15.5 Sample program

The following shows a sample program for Security Set command processing.

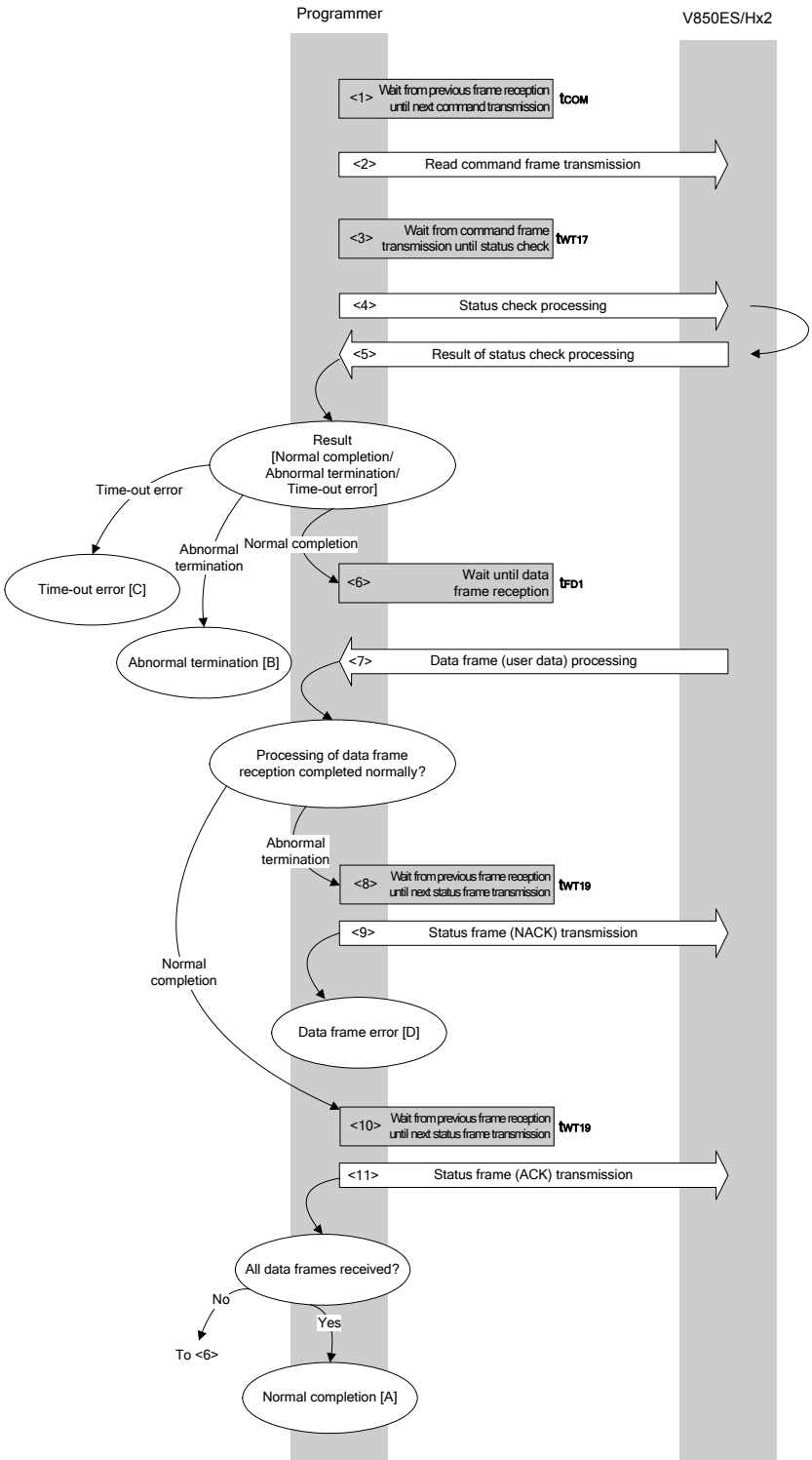
```
/************************************************************/
/*                                                          */
/*   Set security flag command (CSI)                        */
/*                                                          */
/************************************************************/
/*   [i] u8 scf    ... Security flag data                   */
/*   [r] u16             ... error code                     */
/************************************************************/
u16  fl_csi_setscf(u8 scf, u32 vect)
{
     u16    rc;

     /******************************************************/
     /*     set params & data frame                        */
     /******************************************************/
     fl_cmd_prm[0] = 0x00;                  // "BLK" (must be 0x00)
     fl_cmd_prm[1] = 0x00;                  // "PAG" (must be 0x00)

     fl_txdata_frm[0] = (scf|= 0b11110000);  // "FLG" (upper 4bits must be '1' (to
make sure))
     fl_txdata_frm[1] = (u8)(vect >> 16);    // "ADH"
     fl_txdata_frm[2] = (u8)(vect >>  8);    // "ADM"
     fl_txdata_frm[3] = (u8) vect;           // "ADL"


     /*********************************************/
     /*     send command                          */
     /*********************************************/
     fl_wait(tCOM_CSI);                     // wait before sending command frame

     put_cmd_csi(FL_COM_SET_SECURITY, 3, fl_cmd_prm);     //   send   "Security   Set"
command

     fl_wait(tWT13);                  // wait

     rc = fl_csi_getstatus(tWT13_MAX);// get status frame
     switch(rc) {
          case   FLC_NO_ERR:               break; // continue
     //   case   FLC_DFTO_ERR:return rc;   break; // case [C]
          default:           return rc;    break; // case [B]
     }

     /*********************************************/
     /*   send data frame (security setting data)    */
     /*********************************************/
```

```
      fl_wait(tFD3);                          // wait before getting data frame

      put_dfrm_csi(4, fl_txdata_frm, true);   // send data frame(Security data & reset
vector)

      fl_wait(tWT14);

      rc = fl_csi_getstatus(tWT14_MAX);       // get status frame
      switch(rc) {
            case   FLC_NO_ERR:                break; // continue
      //    case   FLC_DFTO_ERR: return rc;   break; // case [C]
            default:             return rc;   break; // case [B]
      }


      /*********************************************/
      /*     Check internally verify                */
      /*********************************************/
      fl_wait(tWT15);

      rc = fl_csi_getstatus(tWT15_MAX);       // get status frame
  //  switch(rc) {
  //
  //        case   FLC_NO_ERR:   return rc;   break; // case [A]
  //        case   FLC_DFTO_ERR: return rc;   break; // case [C]
  //        default:             return rc;   break; // case [B]
  //  }
      return rc;
  }
```

## 8.16 Read Command

### 8.16.1 Processing sequence chart

Read command processing sequence



### 8.16.2 Description of processing sequence

<1> Waits from the previous frame reception until the next command transmission (wait time $t_{COM}$).
<2> The Read command is transmitted by command frame transmission processing.
<3> Waits from command transmission until status check processing (wait time $t_{WT17}$).
<4> The status frame is acquired by status check processing.
<5> The following processing is performed according to the result of status check processing.

When the processing ends normally: Proceeds to <6>.
When the processing ends abnormally: Abnormal termination [B]
When a time-out error occurs: A time-out error [C] is returned.

<6> Waits from the previous frame reception until the data frame reception (wait time $t_{WT18}$).
<7> The data frame (user data) is received by data frame reception processing.
The following processing is performed according to the result of reception processing.

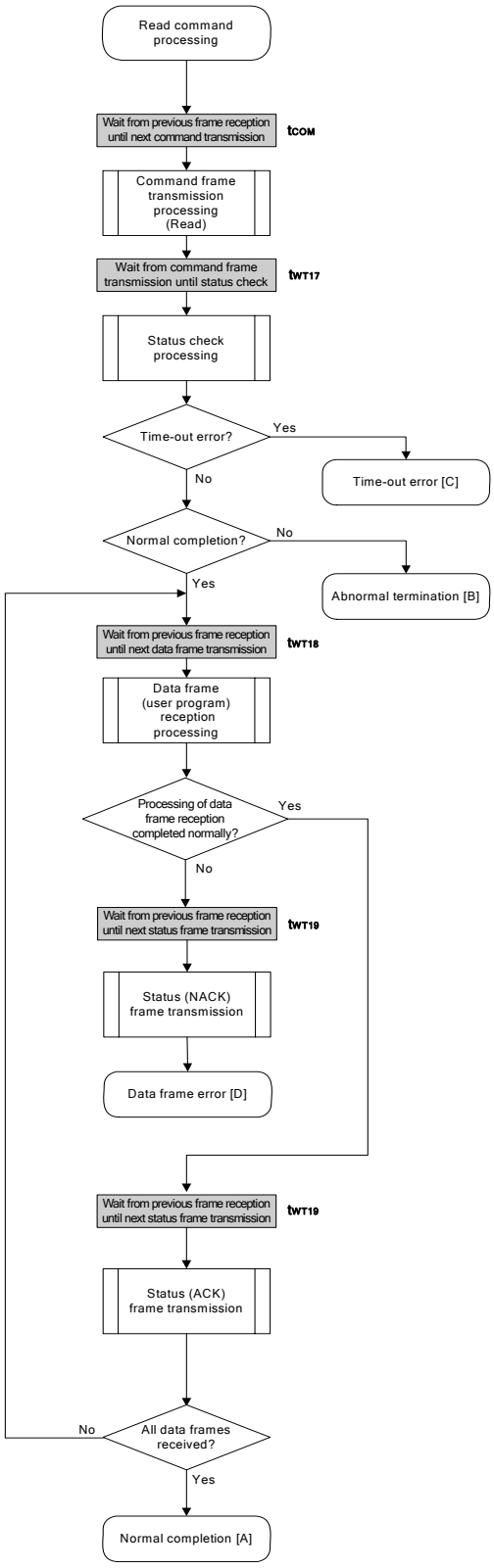
When the processing ends normally: Proceeds to <10>.
When the processing ends abnormally: Proceeds to <8>.

<8> Waits from the previous frame reception until the next status (NACK) frame transmission (wait time $t_{WT19}$).
<9> The NACK frame is transmitted by data frame transmission processing.
A data frame error [D] is returned.
<10> Waits from the previous frame reception until the next status (ACK) frame transmission (wait time $t_{WT19}$).
<11> The ACK frame is transmitted by data frame transmission processing.
When reception of all data frames is completed, the normal completion status [A] is returned.
If there still remain data frames to be received, the sequence is re-executed from <5>.

### 8.16.3 Status at processing completion

| Status at Processing Completion | | Status Code | Description |
|---|---|---|---|
| Normal completion [A] | Normal acknowledgment (ACK) | 06H | The command was executed normally and read data was set normally. |
| Abnormal termination [B] | Parameter error | 05H | The specified start/end address is not the start/end address of the block. |
| | Checksum error | 07H | The checksum of the transmitted command frame or data frame does not match. |
| | Protect error | 10H | Read is prohibited in the security setting. |
| | Negative acknowledgment (NACK) | 15H | Command frame data is abnormal (such as invalid data length (LEN) or no ETX). |
| Time-out error [C] | | – | The status frame or data frame was not received within the specified time. |
| Data frame error [D] | | – | The checksum of the data frame received as read data does not match. |

### 8.16.4 Flowchart



### 8.16.5 Sample program

The following shows a sample program for Read command processing.

```
/************************************************************/
/*                                                          */
/*      Read command (CSI)                                  */
/*                                                          */
/************************************************************/
/*      [i] u32 top   ... start address                     */
/*      [i] u32 bottom      ... end address                 */
/*      [r] u16             ... error code                  */
/************************************************************/
u16     fl_csi_read(u32 top, u32 bottom)
{
  u16   rc;
  u32   read_head;
  u16   len;
  u8    hooter;

  /*********************************************/
  /*    set params                             */
  /*********************************************/

  set_range_prm(fl_cmd_prm, top, bottom); // set SAH/SAM/SAL, EAH/EAM/EAL

  /*********************************************/
  /*    send command & check status            */
  /*********************************************/
  fl_wait(tCOM_CSI);        // wait before sending command

  put_cmd_csi(FL_COM_READ, 7, fl_cmd_prm);      // send "Read" command

  fl_wait(tWT17);           // wait

  rc = fl_csi_getstatus(tWT17_MAX);// get status frame
  switch(rc) {
              case  FLC_NO_ERR:               break; // continue
//            case  FLC_DFTO_ERR:return rc;   break; // case [C]
              default:          return rc;    break; // case [B]
  }

  /*********************************************/
  /*    receive user data                      */
  /*********************************************/
  read_head = top;

  while(1){
        fl_wait(tWT18);

        rc = get_dfrm_csi(fl_rxdata_frm);// get ROM data from FLASH
        switch(rc) {
              case  FLC_NO_ERR:               break; // continue
        //    case  FLC_RX_DFSUM_ERR:
              default:                        // case [D]
                    fl_wait(tWT19);
                    put_sfrm_csi(FLST_NACK);  // send status(NACK) frame
                    return rc;
                    break;

        }

        fl_wait(tWT19);
        put_sfrm_csi(FLST_ACK);         // send status(ACK) frame

        /*********************************************/
        /*    save ROM data                          */
```

```
        /*********************************************/
        if ((len = fl_rxdata_frm[OFS_LEN]) == 0)       // get length
              len = 256;

        memcpy(read_buf+read_head, fl_rxdata_frm+2, len); // save to external RAM

        read_head += len;

        /*********************************************/
        /*     end check                              */
        /*********************************************/
        hooter = fl_rxdata_frm[len + 3];
        if (hooter == FL_ETB)                   // end frame ?
              continue;                         // no
        break;                                  // yes
  }

  return FLC_NO_ERR;
}
```

# CHAPTER 9 FLASH MEMORY PROGRAMMING PARAMETER CHARACTERISTICS

This chapter describes the parameter characteristics between the programmer and the V850ES/Hx2 in the flash memory programming mode.

Be sure to refer to the user's manual of the V850ES/Hx2 for electrical specifications when designing a programmer.

<Operating clock (fx)>

The internal clock of the V850ES/Hx2 is changed according to the value of the oscillation frequency (fx) specified with the Oscillation Frequency Set command by the programmer.

4.0 MHz ≤ fx ≤ 5.0 MHz: $f_{XX} = f_X \times 4$

Therefore, it is obtained by assigning fx until $t_{WT9}$ of the Oscillation Frequency Set command and assigning fxx after $t_{WT9}$.

## 9.1 Flash Memory Programming Mode Setting Time

| Parameter | Symbol | MIN. | TYP. | MAX. |
|---|---|---|---|---|
| $V_{DD}$↑ to FLMD0/FLMD1↑ | $t_{DP}$ | 1 ms | | |
| FLMD0/FLMD1↑ to $\overline{\text{RESET}}$↑ | $t_{PR}$ | 2 ms | | |
| Count start time from $\overline{\text{RESET}}$↑ to FLMD0[Note 1] | $t_{RP}$ | 66,920/fx | | |
| Count finish time from $\overline{\text{RESET}}$↑ to FLMD0[Note 1] | $t_{RPE}$ | | | 139,566/fx |
| FLMD0 counter high-level/low-level width | $t_{PW}$ | 10 μs | | 100 μs |
| Wait for Reset command (CSI/CSI + HS) | $t_{RC}$ | 148,103/fx | | 3 s |
| Wait for low-level data 1 (UART) | $t_{R1}$ | 148,103/fx | | 3 s |
| Wait for low-level data 2 (UART) | $t_{2C}$ | 30,000/fx | | 3 s |
| Wait for Read command (UART) | $t_{12}$ | 30,000/fx | | 3 s |
| Width of low-level data 1/2[Note 2] | $t_{L1}$, $t_{L2}$ | | **Note 2** | |
| FLMD0 counter rise/fall time | – | | | 1 μs |

**Notes 1.** (66,920/fx + 139,566/fx)/2 is recommended as the standard value for the FLMD0 pulse input timing.

**2.** The low-level width is the same as the 00H data width at 9,600 bps.

## 9.2 Programming Characteristics

| Wait | Condition | Symbol | Serial I/F | MIN. | MAX. |
|---|---|---|---|---|---|
| Between data frame transmission/reception | Data frame reception | $t_{DR}$ | CSI | 125/$f_X$ | 3 s |
| | | | UART | 125/$f_X$ | 3 s |
| | Data frame transmission | $t_{DT}$ | CSI | 127/$f_X$ | 3 s |
| | | | UART | 0 | 3 s |
| From Status command frame reception until status frame transmission | – | $t_{SF}$ | CSI | **Note 1** | |
| From status frame transmission until data frame transmission (1) | – | $t_{FD1}$ | CSI | 55,920/$f_X$ + 33,802/$f_X \times N$ | 3 s |
| | | | UART | 0 | 3 s |
| From status frame transmission until data frame transmission (2) | – | $t_{FD2}$ | CSI | 2,998/$f_X$ | 3 s |
| | | | UART | 0[Note 2] | 3 s |
| From status frame transmission until data frame reception | – | $t_{FD3}$ | CSI | 168/$f_X$ | 3 s |
| | | | UART | 168/$f_X$ | 3 s |
| From status frame transmission until command frame reception | – | $t_{COM}$ | CSI | 154/$f_X$ | 3 s |
| | | | UART | 120/$f_X$ | 3 s |

**Notes 1.** $t_{SF}$ MIN. values and MAX. values for respective commands

| Command | MIN. | MAX. |
|---|---|---|
| Reset, Read, Oscillating Frequency Set, Checksum, Silicon Signature, Version Get | 241/$f_X$ | 3 s |
| Chip Erase | 936/$f_X$ | 3 s |
| Block Erase | 807/$f_X$ | 3 s |
| Programming | 12,960/$f_X$ | 3 s |
| Verify | 4,584/$f_X$ | 3 s |
| Block Blank Check | 12,960/$f_X$ | 3 s |
| Security Set | 826/$f_X$ | 3 s |

**2.** When continuous reception is enabled for the programmer

**Remarks 1.** N: Command execution block size (KB)

**2.** The waits are defined as follows.

<$t_{DR}$, $t_{FD3}$, $t_{COM}$>

The V850ES/Hx2 is readied for the next communication after the MIN. time has elapsed after completion of the previous communication.

The programmer must transmit the next data between the MIN. and MAX. time after completion of the previous communication.

<$t_{DT}$, $t_{SF}$, $t_{FD1}$, $t_{FD2}$>

The V850ES/Hx2 is readied for the next communication after the MIN. time has elapsed after completion of the previous communication.

The programmer must receive the next data between the MIN. and MAX. time after completion of the previous communication.

| Command | Symbol | Serial I/F | MIN. | MAX. |
|---|---|---|---|---|
| Reset | $t_{WT0}$ | CSI | $199/f_X$ | 3 s |
| | | UART | **Note 1** | 3 s |
| Chip Erase | $t_{WT1}$ | – | **[$\mu$PD70F3700, 70F3701]** $32,088,940/f_{CX} + 820.8$ ms | **[$\mu$PD70F3700, 70F3701]** $550,375,712/f_{CX} + 16,242/f_X + 35,410$ ms |
| | | | **[$\mu$PD70F3702, 70F3703, 70F3704, 70F3706, 70F3707]** $64,177,880/f_{CX} + 1,641.6$ ms | **[$\mu$PD70F3702, 70F3703, 70F3704, 70F3706, 70F3707]** $1,313,350,208/f_{CX} + 19,130/f_X + 70,820$ ms |
| | | | **[$\mu$PD70F3709, 70F3710, 70F3711, 70F3712]** $128,355,760/f_{CX} + 3,283.2$ ms | **[$\mu$PD70F3709, 70F3710, 70F3711, 70F3712]** $3,477,095,552/f_{CX} + 24,906/f_X + 141,640$ ms |
| Block Erase | $t_{WT2}$ | – | $(5.2 \text{ ms} + 125,147/f_{CX}) + (6.25 \text{ ms} + 246,784/f_{CX}) \times N$ | $25,570/f_X + (282.9 \text{ ms} + 1,836,104/f_{CX}) + (267.8 \text{ ms} + 3,619,584/f_{CX}) \times N$ |
| Programming | $t_{WT3}$ | CSI | $12,768/f_X$ | 3 s |
| | | UART | **Note 1** | 3 s |
| | $t_{WT4}$ | – | $971.3\ \mu s + 29,818/f_{CX}$ | $49.5 \text{ ms} + 367,079/f_{CX}$ |
| | $t_{WT5}$ | CSI | $98,400/f_{CX} \times N$ | $123,000/f_{CX} \times N$ |
| | | UART | **Note 1** | $123,000/f_{CX} \times N$ |
| Verify | $t_{WT6}$ | CSI | $407/f_{CX}$ | 3 s |
| | | UART | **Note 1** | 3 s |
| | $t_{WT7}$ | CSI | $28,128/f_X$ | 3 s |
| | | UART | **Note 1** | 3 s |
| Block Blank Check | $t_{WT8}$ | – | $44,904/f_{CX} \times N$ | $(811,400 + 2,777,554 \times N)/f_{CX} + 21,611/f_X$ |
| Oscillating Frequency Set | $t_{WT9}$ | CSI | $6,199/f_X + 2^{13}/f_X$ | 3 s |
| | | UART | **Note 1** | 3 s |
| Baud Rate Set | $t_{WT10}$ | UART | $4,488/f_X$ | 3 s |
| Silicon Signature | $t_{WT11}$ | CSI | $557/f_X$ | 3 s |
| | | UART | **Note 1** | 3 s |
| Version Get | $t_{WT12}$ | CSI | $570/f_X$ | 3 s |
| | | UART | **Note 1** | 3 s |
| Security Set | $t_{WT13}$ | CSI | $461/f_X$ | 3 s |
| | | UART | **Note 1** | 3 s |
| | $t_{WT14}$ | – | $16,114/f_X + 364.3\ \mu s + 11,295/f_{CX}$ | $17,266/f_X + 49.5 \text{ ms} + 367,079/f_{CX}$ |
| | $t_{WT15}$ | CSI | $11,834,400/f_{CX}$ | $147,930,960/f_{CX}$ |
| | | UART | **Note 1** | $147,930,960/f_{CX}$ |
| Checksum | $t_{WT16}$ | CSI | $691/f_X$ | 3 s |
| | | UART | **Note 1** | 3 s |
| Read | $t_{WT17}$ | CSI | $12,903/f_X$ | 3 s |
| | | UART | **Note 1** | 3 s |
| | $t_{WT18}$ | CSI | $21,466/f_X$ | 3 s |
| | | UART | **Note 1** | 3 s |
| | $t_{WT19}$ | – | **Note 2** | **Note 2** |

**Note 1.** Reception must be enabled for the programmer before command transmission.

**2.** Time until the programmer transmits “ACK”.

**Remarks 1.** N: Command execution block size (K byte)

$f_{CX}$: $f_{XX}$

**2.** The waits are defined as follows.

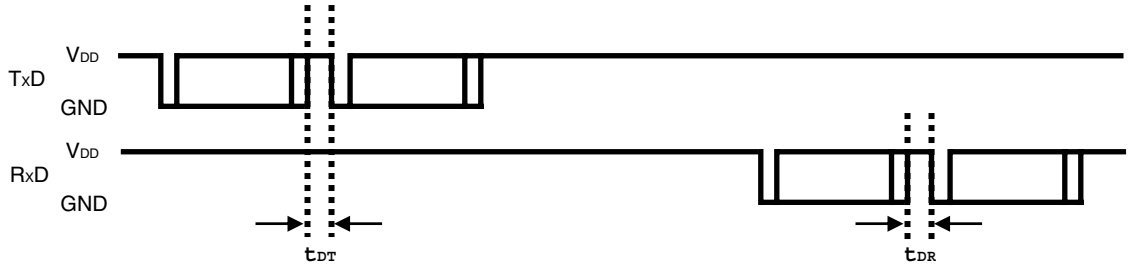
<$t_{WT0}$ to $t_{WT16}$>

The V850ES/Hx2 completes command processing within the MIN. to MAX. time.

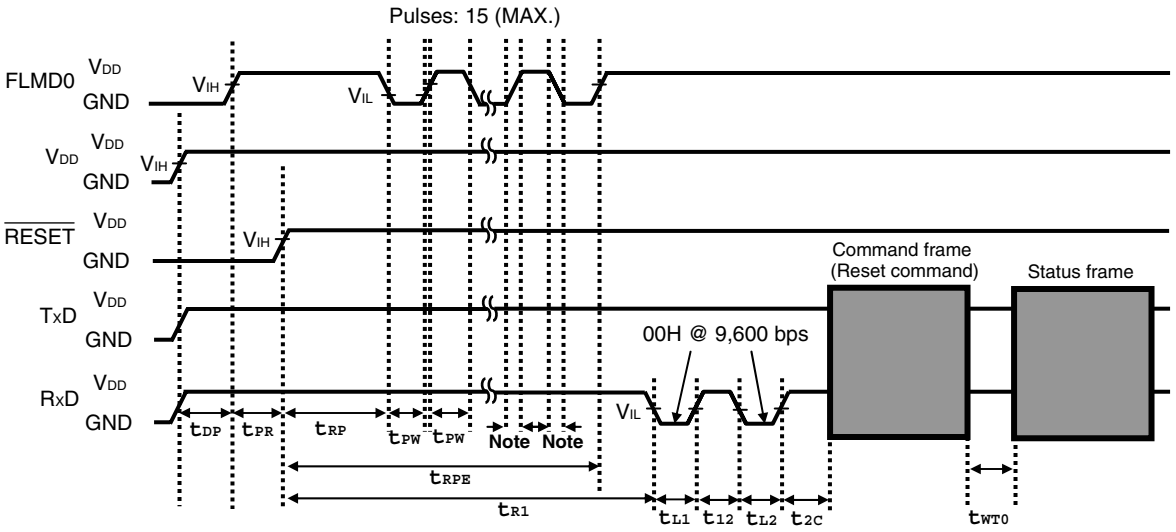
The programmer must repeat the status check until the MAX. time is elapsed.

## 9.3 UART Communication Mode

- Data frame

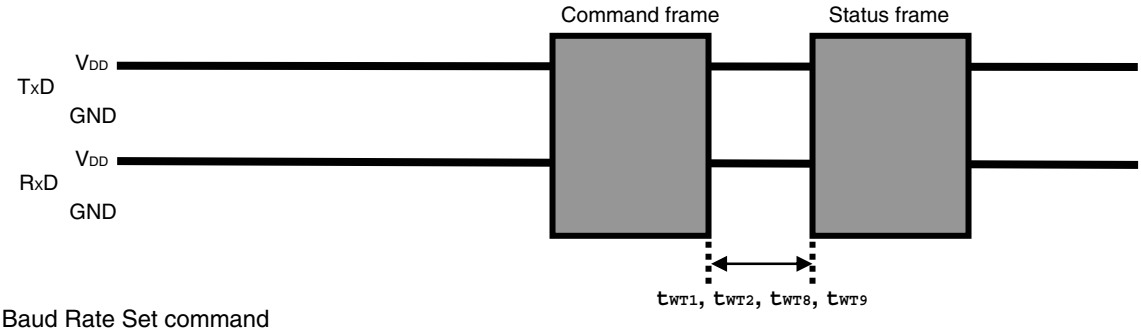



- Programming mode setting/Reset command

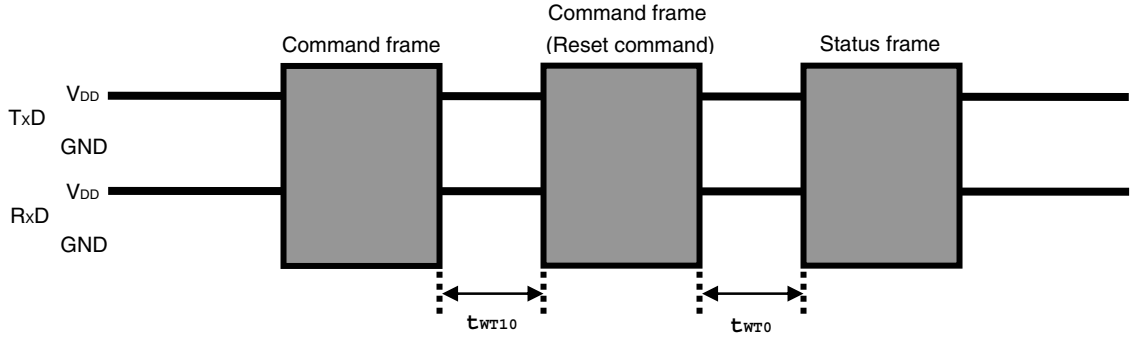



**Note** FLMD0 counter rise/fall time

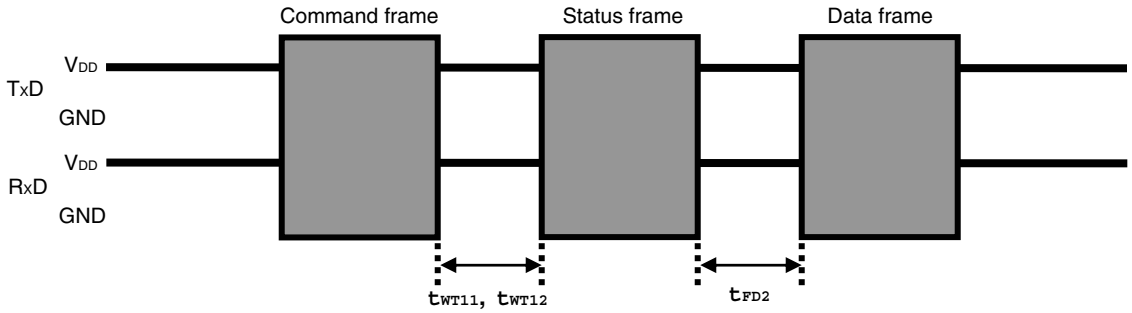
- Chip Erase command/Block Erase command/Block Blank Check command/Oscillating Frequency Set command



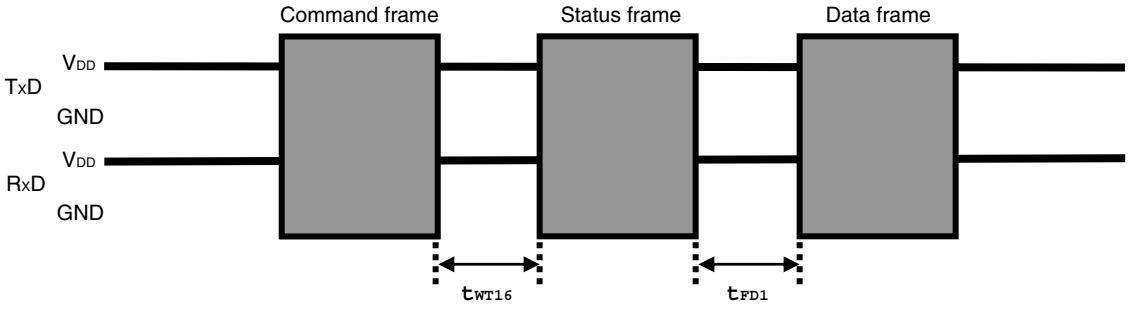

- Baud Rate Set command



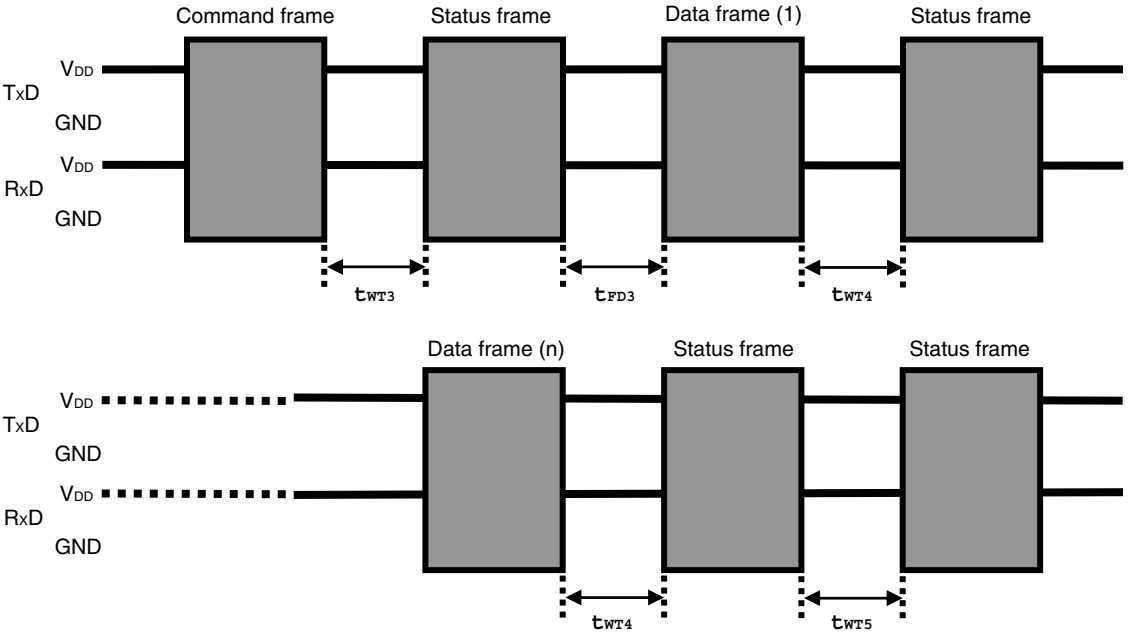
• Silicon Signature command/Version Get command



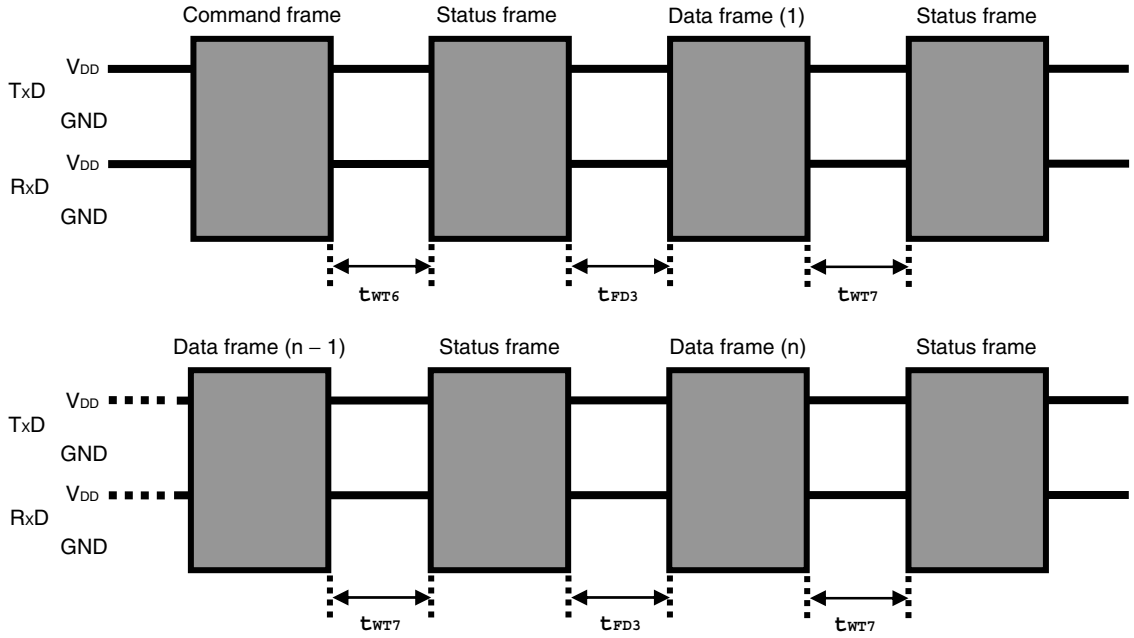

• Checksum command



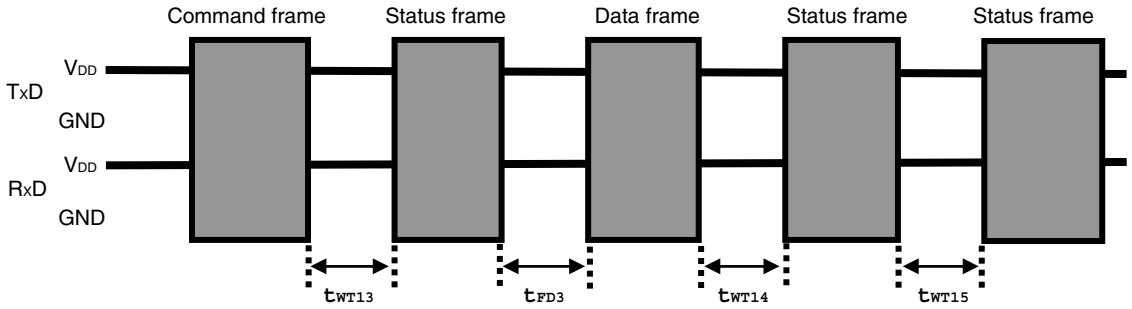

• Programming command



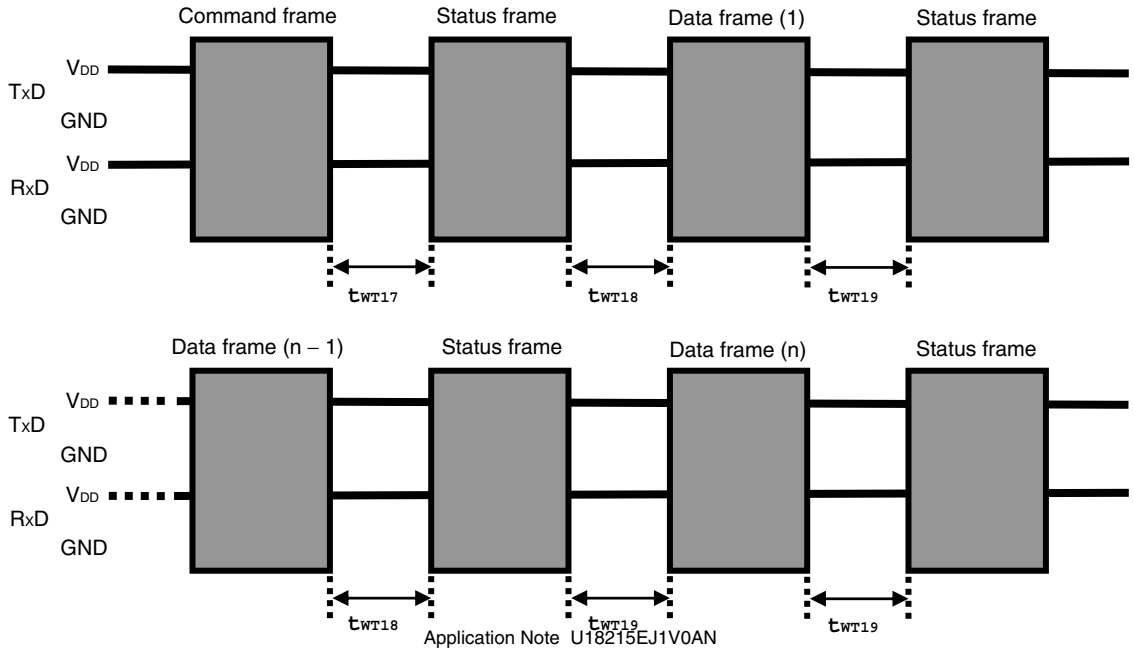
• Verify command



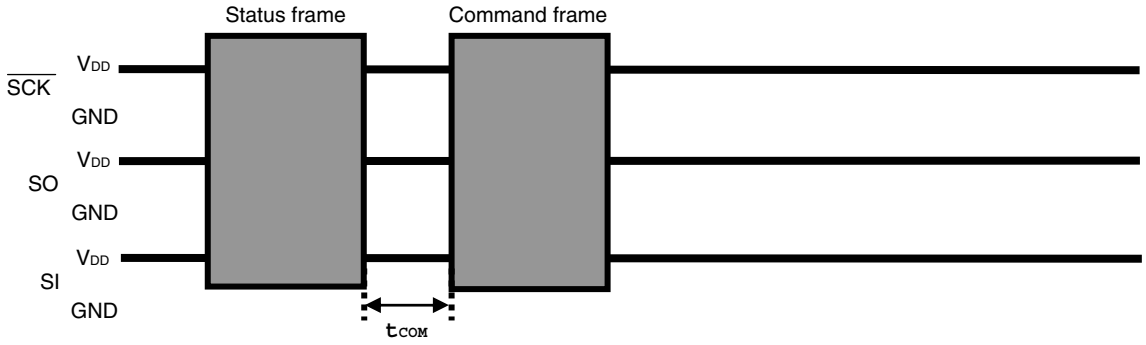

• Security Set command




• Read command

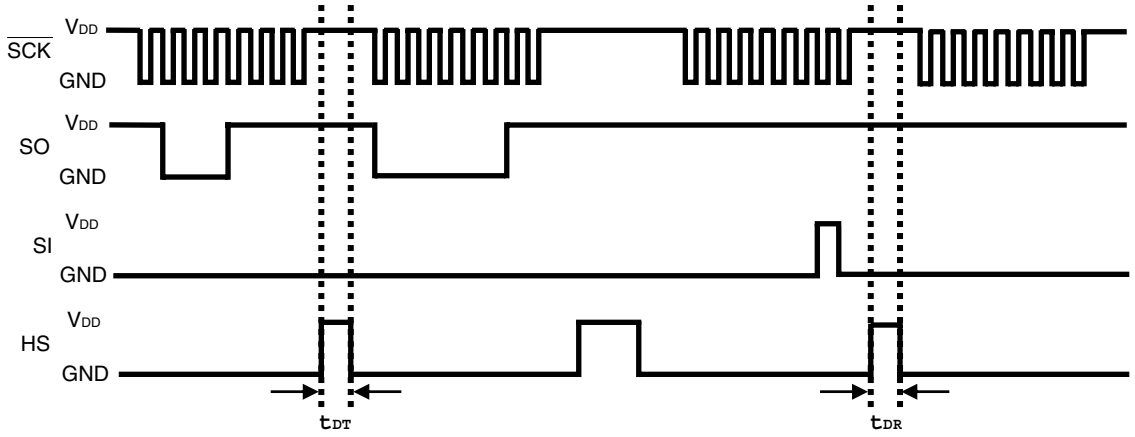


- Wait before command frame transmission

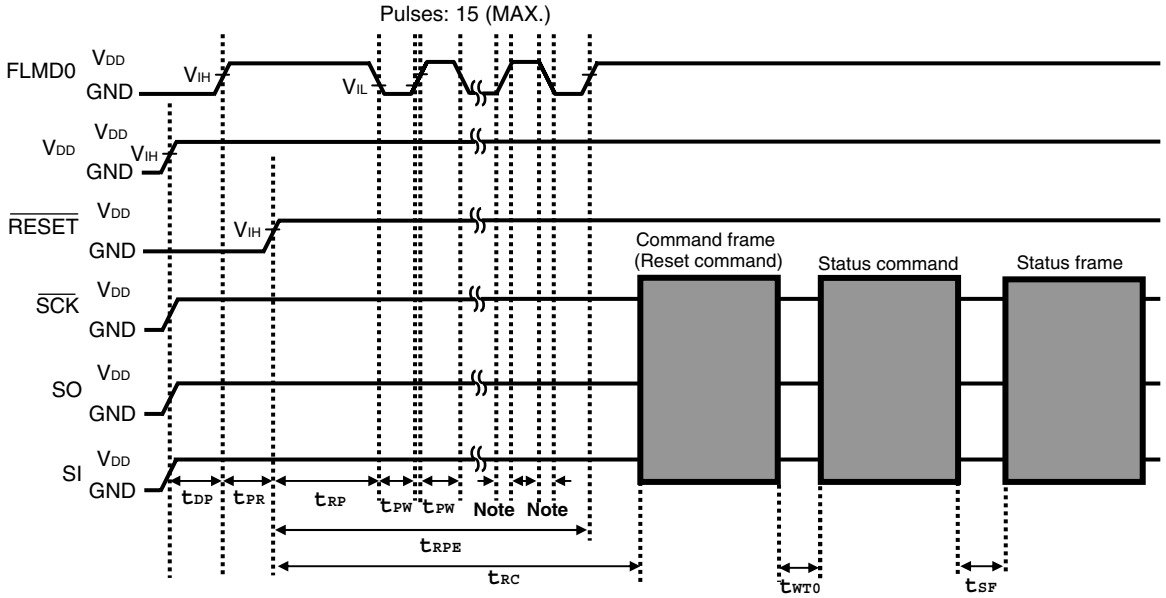


## 9.4 3-Wire Serial I/O Communication Mode

- Data frame

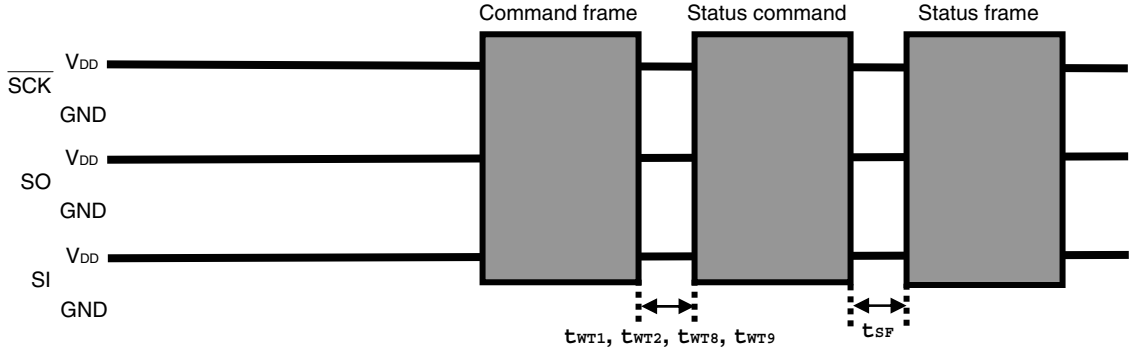



- Programming mode setting/Reset command

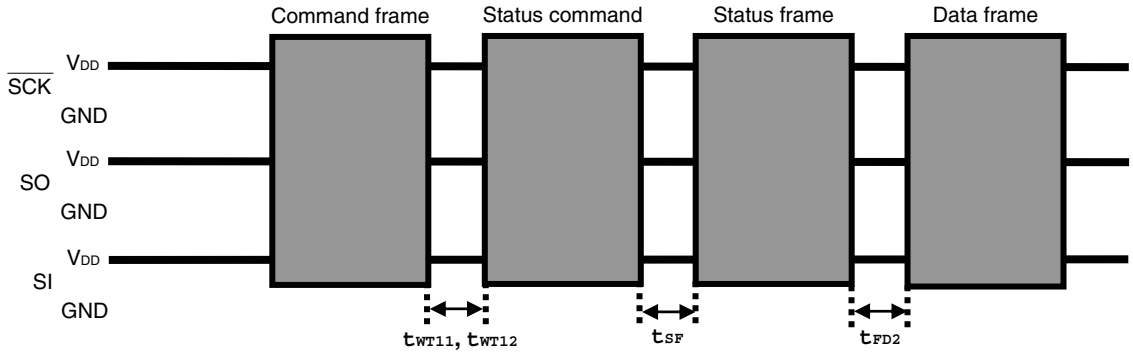



**Note** FLMD0 counter rise/fall time

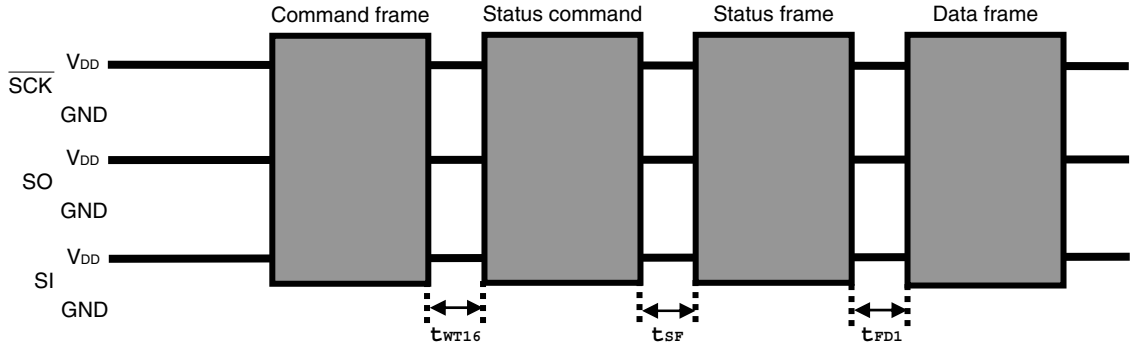
- Chip Erase command/Block Erase command/Block Blank Check command/Oscillating Frequency Set command



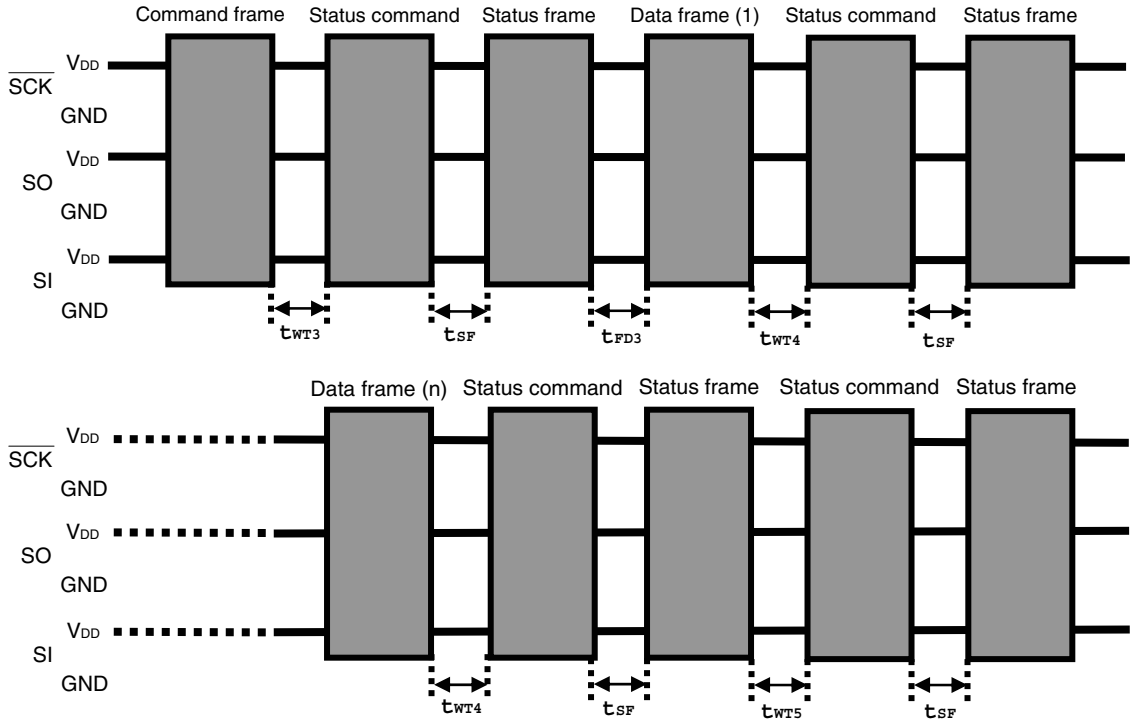
• Silicon Signature command/Version Get command



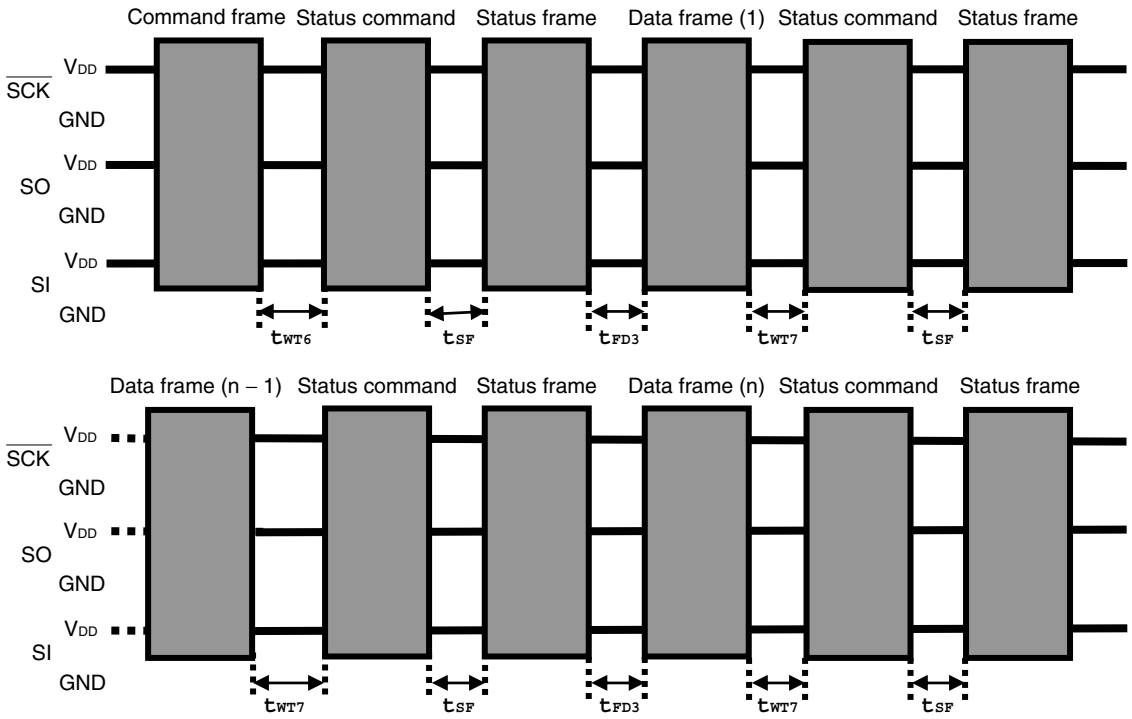

• Checksum command



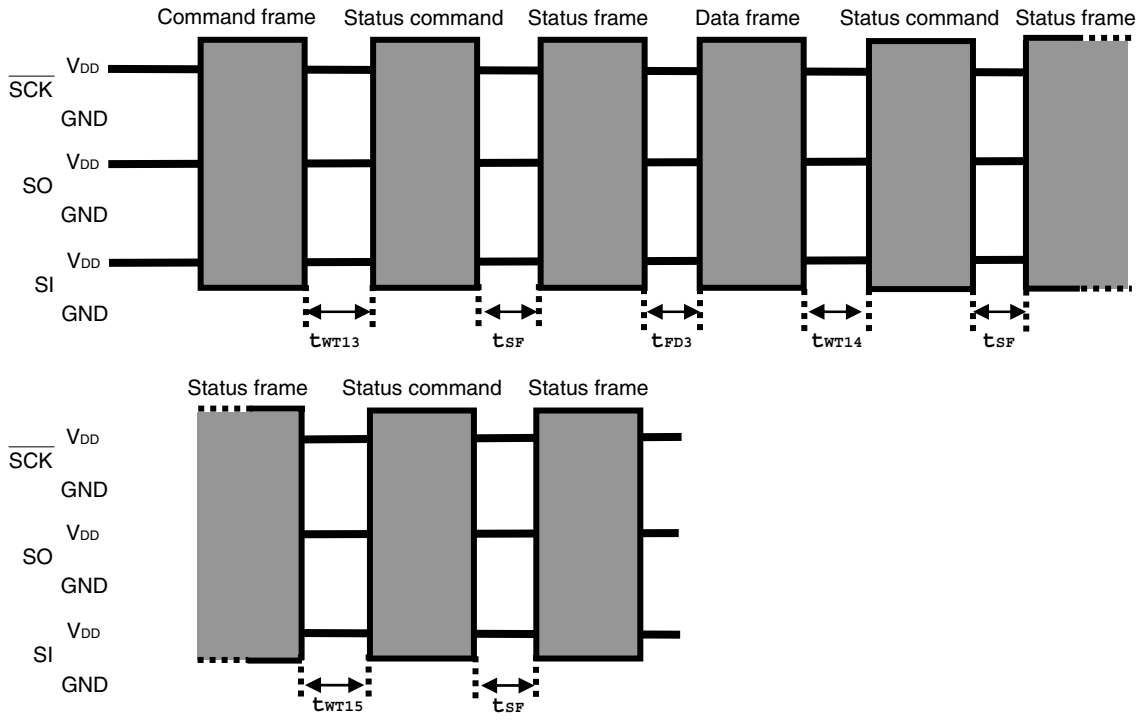

• Programming command



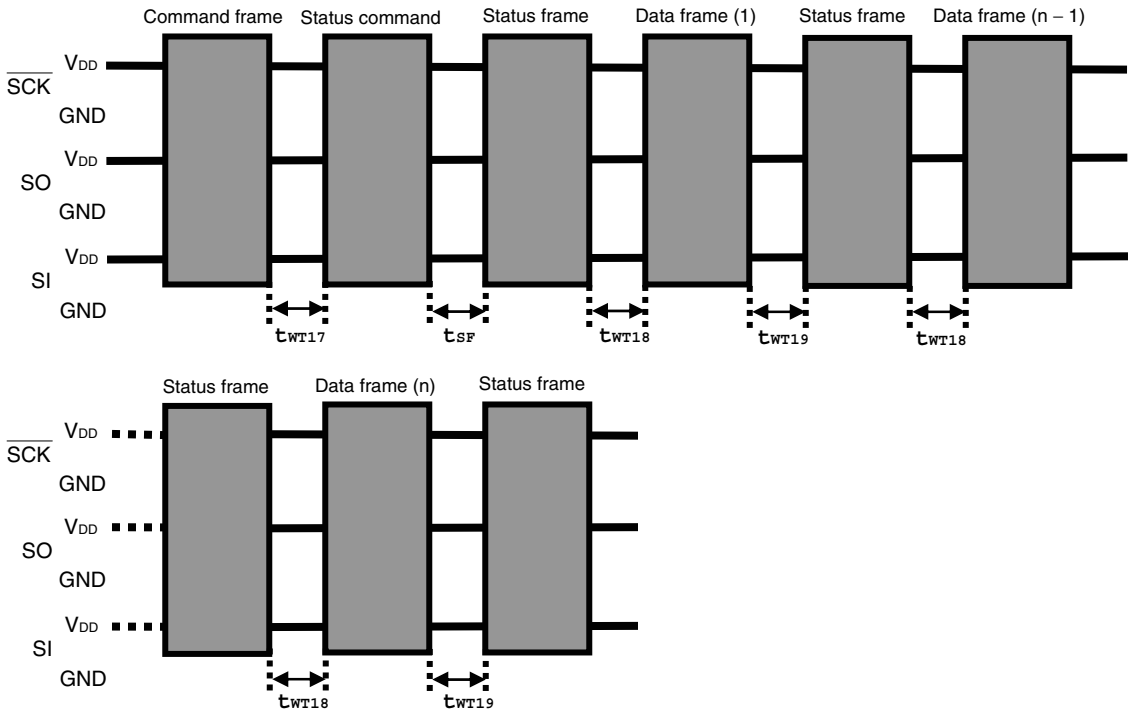
• Verify command



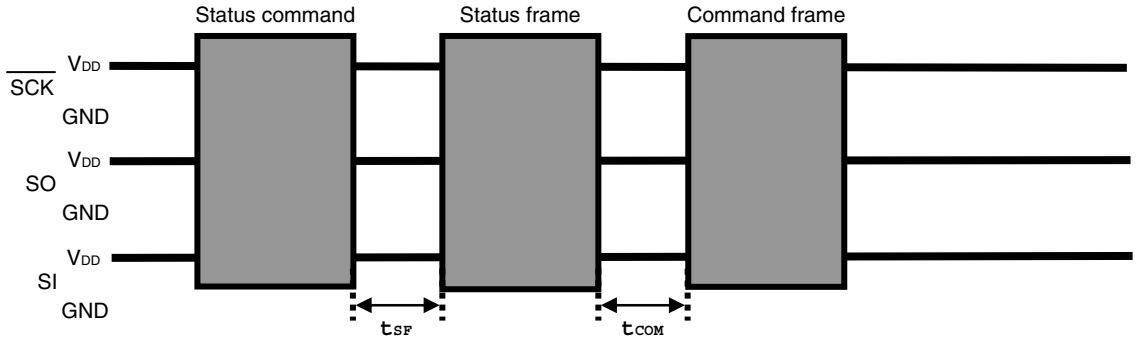

• Security Set command



• Read command




• Wait before command frame transmission



# APPENDIX A CIRCUIT DIAGRAM (REFERENCE)

Figures A-1 and A-2 show circuit diagrams of the programmer and the V850ES/Hx2, for reference.

**Figure A-1. Reference Circuit Diagram of Programmer and V850ES/Hx2 (Main board)**

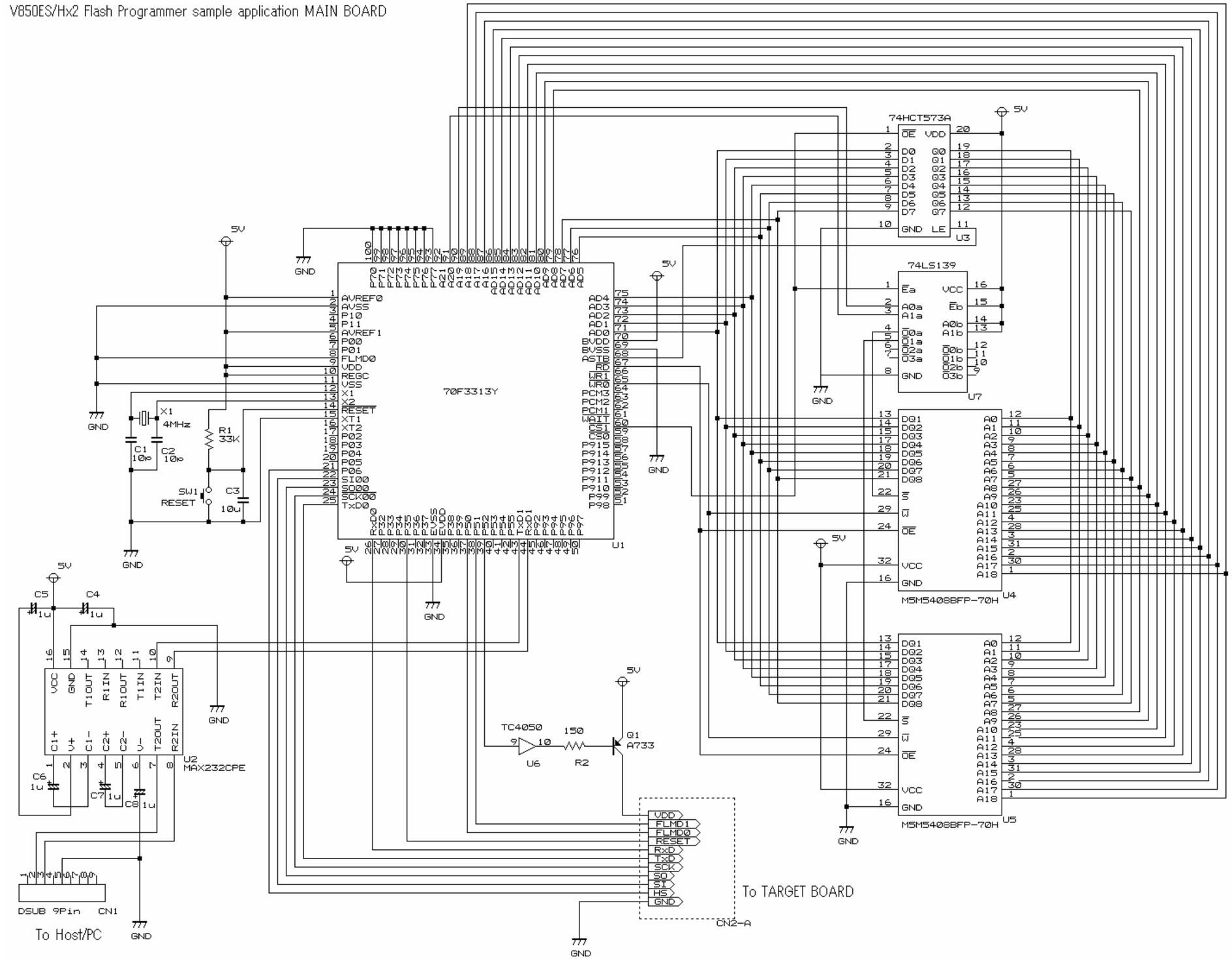
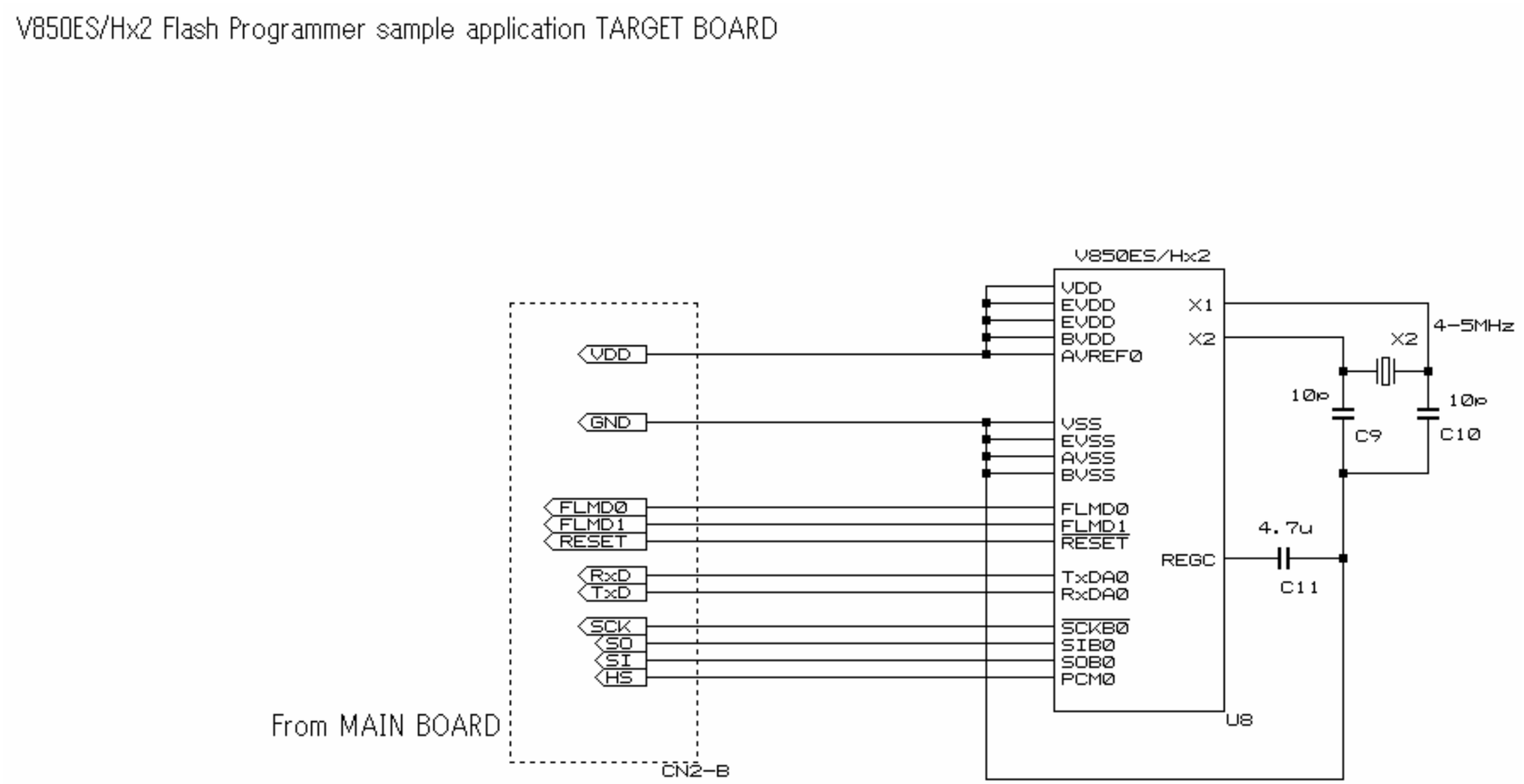


Figure A-2. Reference Circuit Diagram of Programmer and V850ES/Hx2 (Target board)



*For further information,*
*please contact:*


**NEC Electronics Corporation**
1753, Shimonumabe, Nakahara-ku,
Kawasaki, Kanagawa 211-8668,
Japan
Tel: 044-435-5111
http://www.necel.com/

**[America]**

**NEC Electronics America, Inc.**
2880 Scott Blvd.
Santa Clara, CA 95050-2554, U.S.A.
Tel: 408-588-6000
800-366-9782
http://www.am.necel.com/

**[Europe]**

**NEC Electronics (Europe) GmbH**
Arcadiastrasse 10
40472 Düsseldorf, Germany
Tel: 0211-65030
http://www.eu.necel.com/

**Hanover Office**
Podbielski Strasse 166 B
30177 Hanover
Tel: 0 511 33 40 2-0

**Munich Office**
Werner-Eckert-Strasse 9
81829 München
Tel: 0 89 92 10 03-0

**Stuttgart Office**
Industriestrasse 3
70565 Stuttgart
Tel: 0 711 99 01 0-0

**United Kingdom Branch**
Cygnus House, Sunrise Parkway
Linford Wood, Milton Keynes
MK14 6NP, U.K.
Tel: 01908-691-133

**Succursale Française**
9, rue Paul Dautier, B.P. 52180
78142 Velizy-Villacoublay Cédex
France
Tel: 01-3067-5800

**Sucursal en España**
Juan Esplandiu, 15
28007 Madrid, Spain
Tel: 091-504-2787

**Tyskland Filial**
Täby Centrum
Entrance S (7th floor)
18322 Täby, Sweden
Tel: 08 638 72 00

**Filiale Italiana**
Via Fabio Filzi, 25/A
20124 Milano, Italy
Tel: 02-667541

**Branch The Netherlands**
Steijgerweg 6
5616 HS Eindhoven
The Netherlands
Tel: 040 265 40 10

**[Asia & Oceania]**

**NEC Electronics (China) Co., Ltd**
7th Floor, Quantum Plaza, No. 27 ZhiChunLu Haidian
District, Beijing 100083, P.R.China
TEL: 010-8235-1155
http://www.cn.necel.com/

**NEC Electronics Shanghai Ltd.**
Room 2509-2510, Bank of China Tower,
200 Yincheng Road Central,
Pudong New Area, Shanghai P.R. China P.C:200120
Tel: 021-5888-5400
http://www.cn.necel.com/

**NEC Electronics Hong Kong Ltd.**
12/F., Cityplaza 4,
12 Taikoo Wan Road, Hong Kong
Tel: 2886-9318
http://www.hk.necel.com/

**Seoul Branch**
11F., Samik Lavied'or Bldg., 720-2,
Yeoksam-Dong, Kangnam-Ku,
Seoul, 135-080, Korea
Tel: 02-558-3737

**NEC Electronics Taiwan Ltd.**
7F, No. 363 Fu Shing North Road
Taipei, Taiwan, R. O. C.
Tel: 02-8175-9600

**NEC Electronics Singapore Pte. Ltd.**
238A Thomson Road,
#12-08 Novena Square,
Singapore 307684
Tel: 6253-8311
http://www.sg.necel.com/