

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.

Application Note

78K0/Lx3

Sample Program (16-bit $\Delta\Sigma$ -Type A/D Converter)

Conversion Result Accuracy Correction

This application note introduces methods which enable the 16-bit $\Delta\Sigma$ -type A/D converter mounted in the 78K0/LE3 and 78K0/LF3 microcontrollers to be used with better accuracy. When using the 16-bit $\Delta\Sigma$ -type A/D converter in differential input mode, the conversion result error includes a specific characteristic. By using this error characteristic and removing it using software, the $\Delta\Sigma$ -type A/D converter can be used with better accuracy.

Target devices

- 78K0/LE3 microcontroller
- 78K0/LF3 microcontroller

CONTENTS

CHAPTER 1 OVERVIEW	3
CHAPTER 2 A/D CONVERSION RESULT CORRECTION ALGORITHM.....	7
2.1 5-Point Correction	7
2.2 Temperature Correction	10
CHAPTER 3 CIRCUIT DIAGRAM	12
3.1 Circuit Diagrams	12
3.1.1 Circuit diagram when not using DAC.....	12
3.1.2 Circuit diagram when using DAC	13
3.2 Peripheral Hardware	14
CHAPTER 4 SOFTWARE	15
4.1 File Configuration	15
4.2 Internal Peripheral Functions to Be Used	16
4.3 Initial Settings and Operation Overviews	17
4.3.1 Operation when not using DAC	17
4.3.2 Operation when using DAC	20
4.4 UART Transmission Data Format.....	23
4.5 Flowcharts.....	24
CHAPTER 5 SETTING METHOD.....	43
5.1 Initial Settings of Peripherals to Be Used.....	43
5.2 A/D Conversion Processing.....	45
CHAPTER 6 OPERATION CHECK EXAMPLE USING DEVICE.....	46
6.1 5-Point Correction	46
6.2 Temperature Correction	47
CHAPTER 7 RELATED DOCUMENTS.....	48
APPENDIX A PROGRAM LIST.....	49
APPENDIX B REVISION HISTORY	87

• **The information in this document is current as of June, 2008. 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).

CHAPTER 1 OVERVIEW

This application note introduces methods which enable the 16-bit $\Delta\Sigma$ -type A/D converter mounted in the 78K0/LE3 and 78K0/LF3 microcontrollers to be used with better accuracy.

When using the 16-bit $\Delta\Sigma$ -type A/D converter in differential input mode, the conversion result includes a specific characteristic. Particularly, as shown in Figure 1-1, inflection points exist in the conversion result at specific input voltages (5 points). By using this characteristic and removing it using software (5-point correction), the $\Delta\Sigma$ -type A/D converter can be used with better accuracy.

There is also a specific characteristic in the conversion result when the temperature has changed. Particularly, as shown in Figure 1-2, the gain changes due to the temperature change. By using this characteristic and removing them using software (temperature correction), the $\Delta\Sigma$ -type A/D converter can be used at better accuracy.

This application note describes the above-mentioned methods in detail and provides a sample program.

Figure 1-1. A/D Conversion Result Characteristic

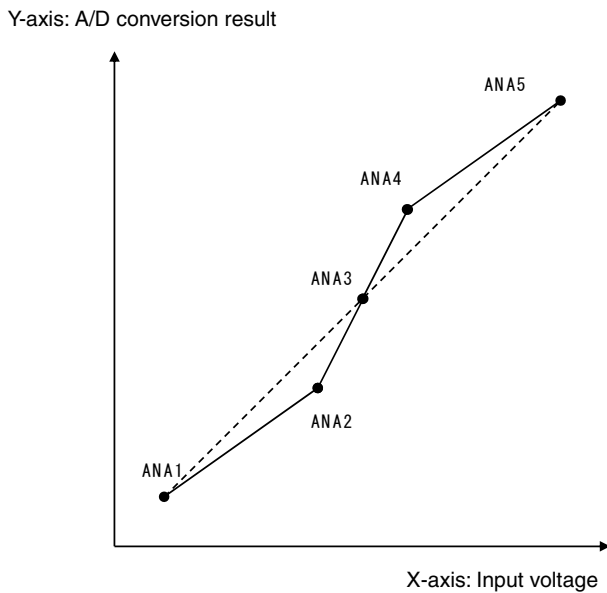
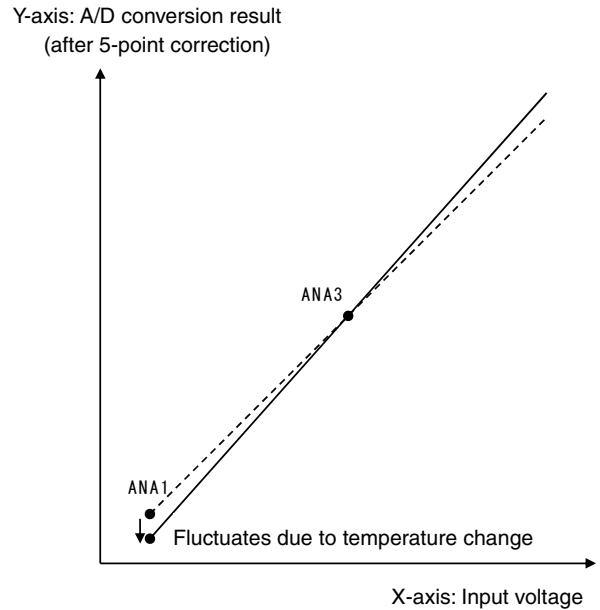


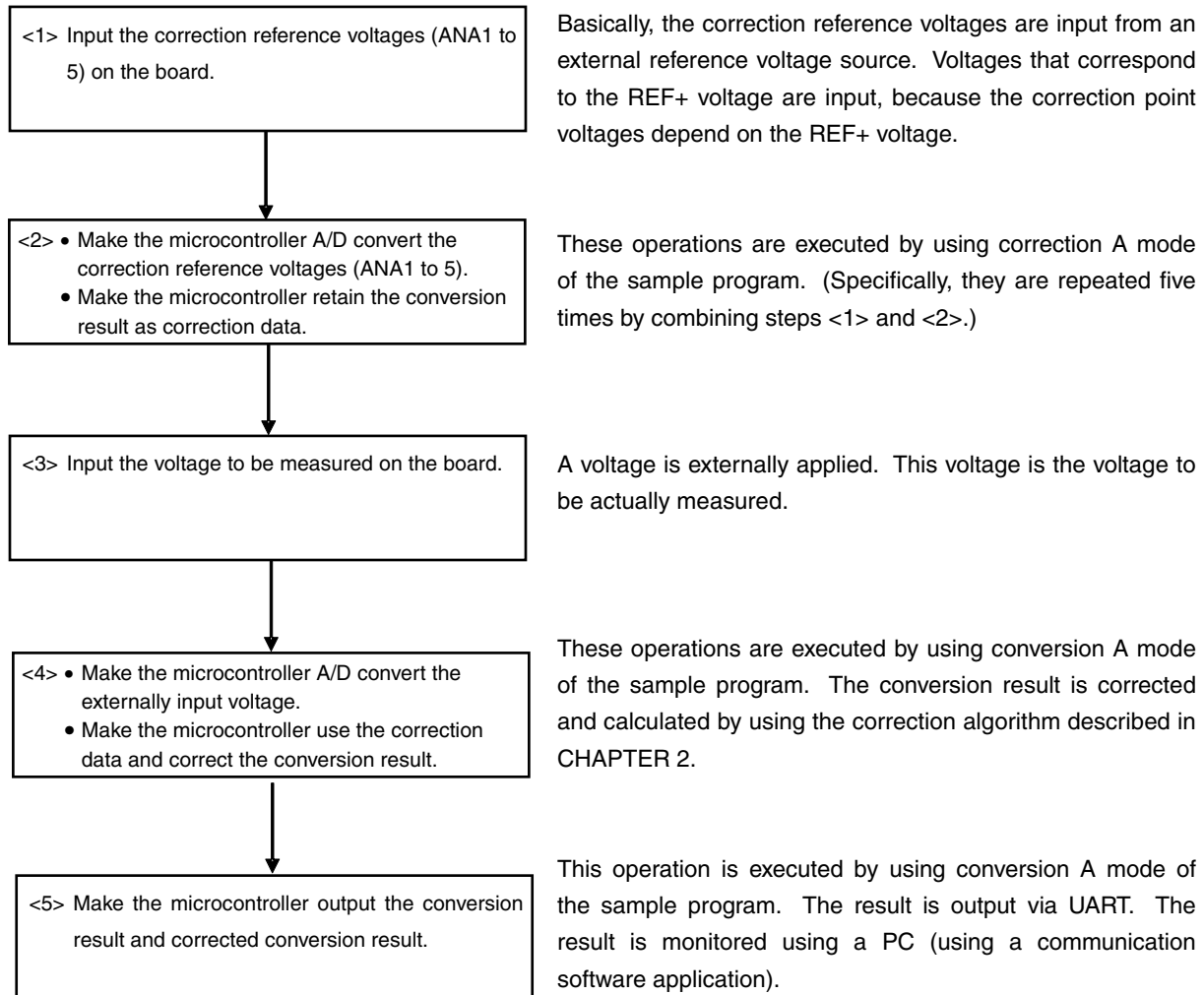
Figure 1-2. A/D Conversion Result Characteristic upon Temperature Change



Caution The methods described in this document do not guarantee accuracy improvement. Use them after implementing thorough evaluation in an actual application.

An operation procedure overview and use of the sample program are described below.

The basic operation procedure is performed in the order of steps <1> to <5>, shown below.



Correction B mode and conversion B mode are provided in this sample program, in addition to correction A mode and conversion A mode, mentioned above. Correction B mode and conversion B mode are used to output the voltage setting to an external DAC-IC before A/D conversion. By mounting a DAC-IC on the board and using these modes, evaluation using the voltage from the DAC-IC can be performed.

Hereinafter, “when not using a DAC” refers to externally inputting a voltage without using the voltage from the DAC-IC, and “when using a DAC” refers to using the voltage from the DAC-IC.

The processing contents of the sample program are described below.

(1) Main contents of the initial settings of the peripherals to be used

The contents of the initial settings of the peripherals to be used are as follows.

- Disabling interrupts
- Setting the CPU and peripheral hardware clock to 10 MHz via high-speed system clock operation
- Setting serial interface UART6 for data transmission
- Setting 8-bit timer H0 as the interval timer (basic timer: about 100 μ s) for the following timing
 - Wait for stabilization of the circuit after setting the DACs (about 500 μ s)
- Setting 8-bit timer H1 as the interval timer for key scan (about 10 ms cycle)
- Setting the 16-bit $\Delta\Sigma$ -type A/D converter
- Setting ports
- Enabling interrupts

(2) Contents of main processing

Key scan processing and event processing are called. Furthermore, the current mode is identified and one among conversion A1 mode control processing, conversion A2 mode control processing, conversion B mode control processing, correction A mode control processing, and correction B mode control processing is called.

Key scan processing is called in a cycle of about 10 ms.

(3) Contents of key scan processing

This processing detects key inputs.

It scans the seven key input pins about every 10 ms. It removes chattering upon three matches and encodes the current key statuses only if matches occur.

(4) Contents of event processing

The mode (conversion A1, conversion A2, conversion B, correction A, or correction B) or the processing execution status (during standby, during execution, or during stop processing) is switched according to key code when a key code change has been detected. If the processing execution status is “During standby”, the mode can be switched.

(5) Contents of conversion A1 mode control processing

In conversion A1 mode, a single conversion operation is performed assuming external voltage input. The following processing is executed once by pressing the progress key.

- Analog input A/D conversion
- A/D conversion result correction^{Note 1}
- UART transmission^{Note 2}

Notes 1. A/D conversion result correction is executed only if 5-point correction data has been normally measured in correction A mode or correction B mode.

2. See **4.4 UART Transmission Data Format** for the UART transmission data format.

(6) Contents of conversion A2 mode control processing

In conversion A2 mode, successive conversion operations are performed assuming external voltage input. By pressing the progress key, the following processing is executed until the stop key is pressed.

- Analog input A/D conversion
- A/D conversion result correction^{Note 1}
- UART transmission^{Note 2}

(7) Contents of conversion B mode control processing

In conversion B mode, successive conversion operations are performed assuming voltage input from a DAC. The set conversion start and conversion end values are read by pressing the progress key. Next, the following processing is executed while the DAC setting values from the conversion start value to the conversion end value, which have been read, are incremented. Conversion B mode control processing ends when the following processing ends after setting the conversion end value to the DAC, or when the stop key is pressed.

- DAC output setting
- Analog input A/D conversion
- A/D conversion result correction^{Note 1}
- UART transmission^{Note 2}

(8) Contents of correction A mode control processing

In correction A mode, 5-point correction data is measured assuming external voltage input. The following processing is performed every time the progress key is pressed, and correction A mode control processing ends if the following processing has been executed five times or if the stop key has been pressed.

- Analog input A/D conversion
- UART transmission^{Note 2}

An error display LED will be lit if the 5-point A/D conversion result is not measured correctly.

(9) Contents of correction B mode control processing

In correction B mode, 5-point correction data is measured assuming voltage input from a DAC. The main processing is as follows. The following processing is successively performed by pressing the progress key, and correction B mode control processing ends if the following processing has been executed five times or if the stop key has been pressed. The DAC output setting values are calculated in advance via initialization processing.

- DAC output setting
- Analog input A/D conversion
- UART transmission^{Note 2}

An error display LED will be lit if the 5-point A/D conversion result is not measured correctly.

Notes 1. A/D conversion result correction is executed only if 5-point correction data has been normally measured in correction A mode or correction B mode.

2. See 4.4 **UART Transmission Data Format** for the UART transmission data format.

CHAPTER 2 A/D CONVERSION RESULT CORRECTION ALGORITHM

This chapter describes the 5-point correction and temperature correction algorithms to be used in this sample program.

2.1 5-Point Correction

An overview of 5-point correction processing is given below.

- <1> Five units of 5-point correction data, which have been matched with the voltage to be used are measured from each sample in advance.
- <2> The I/O characteristics of the 16-bit $\Delta\Sigma$ -type A/D converter are approximated to four functions, based on the 5-point correction data.
- <3> The approximation functions of the 16-bit $\Delta\Sigma$ -type A/D converter I/O characteristics are used to correct the conversion result of the 16-bit $\Delta\Sigma$ -type A/D converter.

(1) 5-point correction data measurement

Five units of 5-point correction data, which have been matched with the voltage to be used are measured from each sample in advance. Figure 2-1 shows the I/O characteristic model of the 16-bit $\Delta\Sigma$ -type A/D converter. ana1, ana2, ana3, ana4, and ana5, shown in the figure, are I/O signal values. code1, code2, code3, code4, and code5, shown in the figure, are the output codes for I/O signals ana1, ana2, ana3, ana4, and ana5. The 16-bit $\Delta\Sigma$ -type A/D converter is used to measure code1 to code5 at ana1 to ana5 in 16-bit resolution. ana1 to ana5 are calculated using the following expressions. code1 to code5 are measured at 25°C.

$$\text{ana5: } 0.9 \times (\text{REF+})$$

$$\text{ana4: } 0.82 \times (\text{REF+}) - 0.91^{\text{Note}}$$

$$\text{ana3: } 0.5 \times (\text{REF+})$$

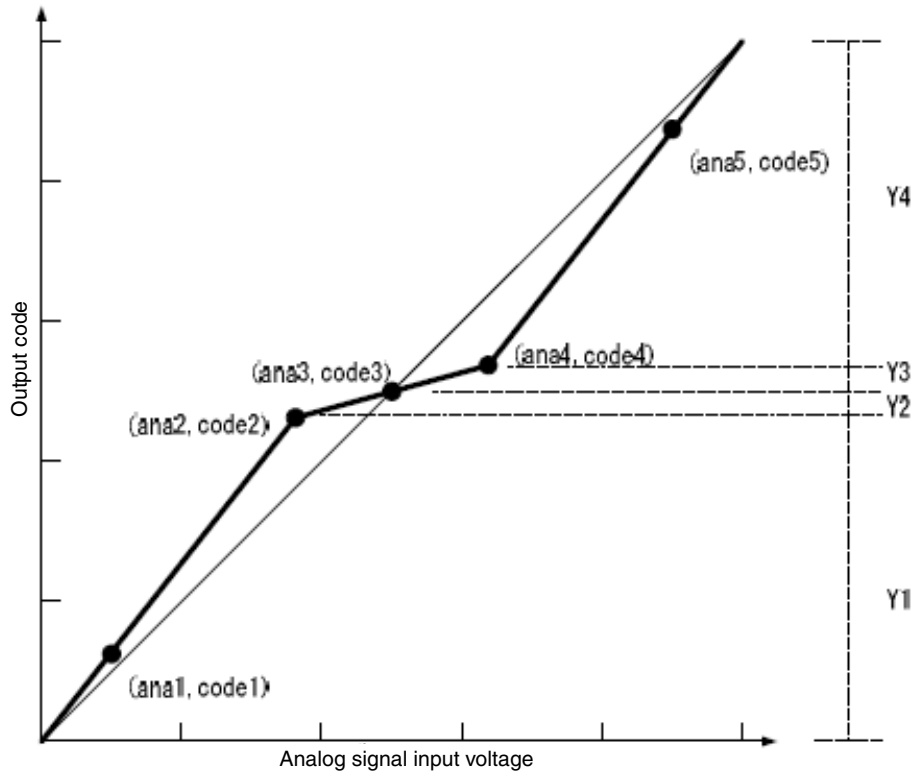
$$\text{ana2: } (\text{REF+}) - (0.82 \times (\text{REF+}) - 0.91)^{\text{Note}}$$

$$\text{ana1: } 0.1 \times (\text{REF+})$$

- (REF+) is the positive-side voltage of the reference pin and has the same potential as AV_{REF} .
- AV_{DD} is an analog power supply voltage.
- Output codes code1 to code5 are calculated by A/D converting input signals ana1 to ana5.

Note If $AV_{\text{REF}} = (\text{REF+}) < 2.84375 \text{ V}$, ana2 > ana4. In this case, the values of ana2 and ana4 are reversed and 5-point correction is performed.

Figure 2-1. 16-bit $\Delta\Sigma$ -Type A/D Converter I/O Characteristic Model



(2) Approximating 16-bit $\Delta\Sigma$ -type A/D converter I/O characteristic to functions

The I/O characteristics of the 16-bit $\Delta\Sigma$ -type A/D converter are approximated to four functions, based on the 5-point correction data. As shown in Table 2-1, the I/O characteristic of each output code interval Y1, Y2, Y3, and Y4 (see Figure 2-1) is approximated to a function (inclination and intercept) using the 5-point correction data in sets of two points.

Table 2-1. 16-bit $\Delta\Sigma$ -Type A/D Converter I/O Characteristic Approximation Functions

Interval	Subject Output Codes	Approximation Functions	
		Inclination	Intercept
Y4	code4 or later	$a4 \times A_{IN} + b4$	$a4 = \frac{code5 - code4}{ana5 - ana4}$ $b4 = \frac{ana4 \times code5 - ana5 \times code4}{ana4 - ana5}$
Y3	code3 to code4	$a3 \times A_{IN} + b3$	$a3 = \frac{code4 - code3}{ana4 - ana3}$ $b3 = \frac{ana3 \times code4 - ana4 \times code3}{ana3 - ana4}$
Y2	code2 to code3	$a2 \times A_{IN} + b2$	$a2 = \frac{code3 - code2}{ana3 - ana2}$ $b2 = \frac{ana2 \times code3 - ana3 \times code2}{ana2 - ana3}$
Y1	code2	$a1 \times A_{IN} + b1$	$a1 = \frac{code2 - code1}{ana2 - ana1}$ $b1 = \frac{ana1 \times code2 - ana2 \times code1}{ana1 - ana2}$

Remark A_{IN} : Analog input voltage

(3) A/D conversion result correction

The approximation functions of the 16-bit $\Delta\Sigma$ -type A/D converter I/O characteristics are used to correct the A/D conversion result. As shown in Table 2-2, the A/D conversion result is corrected according to the output code values and then output. In principle, the same result is obtained for the interval boundary values (code2, code3, and code4), regardless of whichever correction expression is used.

If $\text{code1} \leq \text{A/D conversion result} \leq \text{code5}$, 5-point correction can be performed.

Table 2-2. Correction Output Arithmetic Expressions

Interval	Output Codes Subject to 5-Point Correction	5-Point Correction Outputs	5-Point Correction Outputs (Expanded by Assigning a1 to a4 and b1 to b4)
Y4	code4 or later	$\frac{\text{A/D conversion result} - b4}{a4}$	$\frac{\text{A/D conversion result} (\text{ana5} - \text{ana4}) + \text{ana4code5} - \text{ana5code4}}{\text{code5} - \text{code4}}$
Y3	code3 to code4	$\frac{\text{A/D conversion result} - b3}{a3}$	$\frac{\text{A/D conversion result} (\text{ana4} - \text{ana3}) + \text{ana3code4} - \text{ana4code3}}{\text{code4} - \text{code3}}$
Y2	code2 to code3	$\frac{\text{A/D conversion result} - b2}{a2}$	$\frac{\text{A/D conversion result} (\text{ana3} - \text{ana2}) + \text{ana2code3} - \text{ana3code2}}{\text{code3} - \text{code2}}$
Y1	code2	$\frac{\text{A/D conversion result} - b1}{a1}$	$\frac{\text{A/D conversion result} (\text{ana2} - \text{ana1}) + \text{ana1code2} - \text{ana2code1}}{\text{code2} - \text{code1}}$

2.2 Temperature Correction

An overview of temperature correction processing is given below.

- <1> One point of temperature correction data is measured before starting A/D conversion.
- <2> The inclination of the 5-point correction result according to the temperature is calculated, based on the temperature correction data.
- <3> The 5-point correction result is temperature-corrected, based on the calculated inclination.

Caution The outcome of correction via this temperature correction is not quantified.

(1) Temperature correction data measurement

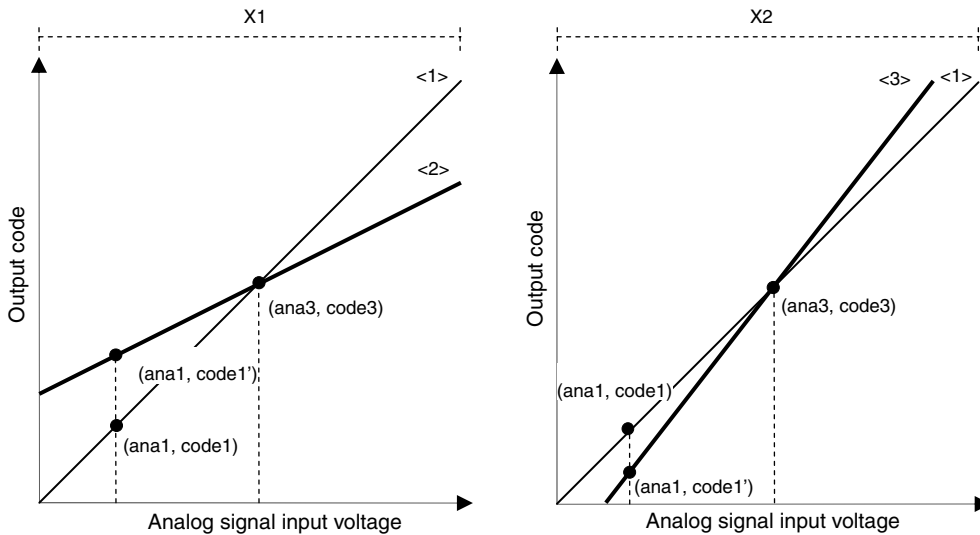
The 16-bit $\Delta\Sigma$ -type A/D converter is used to measure the output code at ana1 in 16-bit resolution and 5-point correction is performed for the A/D value before starting A/D conversion. This value is code1'.

(2) Calculating inclination of 5-point correction result

The reference temperature A/D value (code1) changes up to code1' due to a temperature change at ana1. At this time, the A/D conversion value characteristic is a straight line going through (ana1, code1') and (ana3, code3).

The actual inclination of the 5-point correction result is calculated based on code1' measured in (1), ana1, ana3, and code3, which has been corrected in advance. code1 and code3 are values resulting from performing 5-point correction for code1 and code3, which were measured in 2.1 (1).

Figure 2-2. 5-Point Correction Result Inclination Model



- <1>: Data without temperature errors
- <2>: Actual 5-point correction result (when the inclination decreases due to temperature errors)
- <3>: Actual 5-point correction result (when the inclination increases due to temperature errors)

(3) Temperature-correcting 5-point correction result

Temperature correction is performed by calculating the inclination of the 5-point correction result with temperature errors, according to code1, code3, and code1', and thus approximating the inclination to an inclination of ideal values.

Table 2-2 shows the expressions to be used for actual temperature correction operation. Two expressions for the cases of X1 and X2 are used so that negative values do not result midway during the calculation. See **Figure 2-2** for each case.

Table 2-2. Temperature Correction Arithmetic Expressions

Interval	Inclination of 5-Point Correction Result According to Temperature Errors	Temperature-Corrected Output
X1	Small	$\frac{5\text{-point correction result} (code3 - code1) - code3 (code1' - code1)}{code3 - code1'}$
X2	Large	$\frac{5\text{-point correction result} (code3 - code1) + code3 (code1 - code1')}{code3 - code1'}$

CHAPTER 3 CIRCUIT DIAGRAM

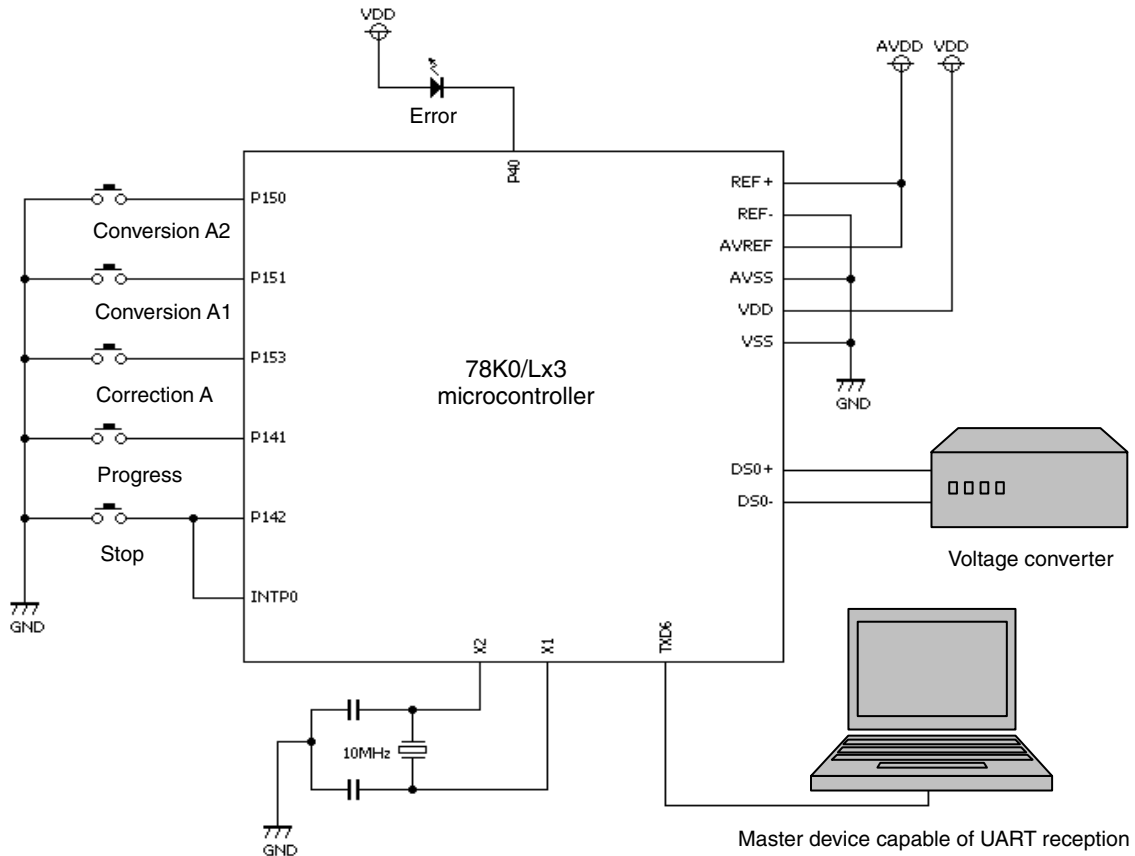
This chapter describes the circuit diagrams and peripheral hardware when using this sample program.

3.1 Circuit Diagrams

3.1.1 Circuit diagram when not using DAC

The circuit diagram when not using a DAC for the $\Delta\Sigma$ -type A/D converter analog input is shown below.

Figure 3-1. Connection When Not Using DAC

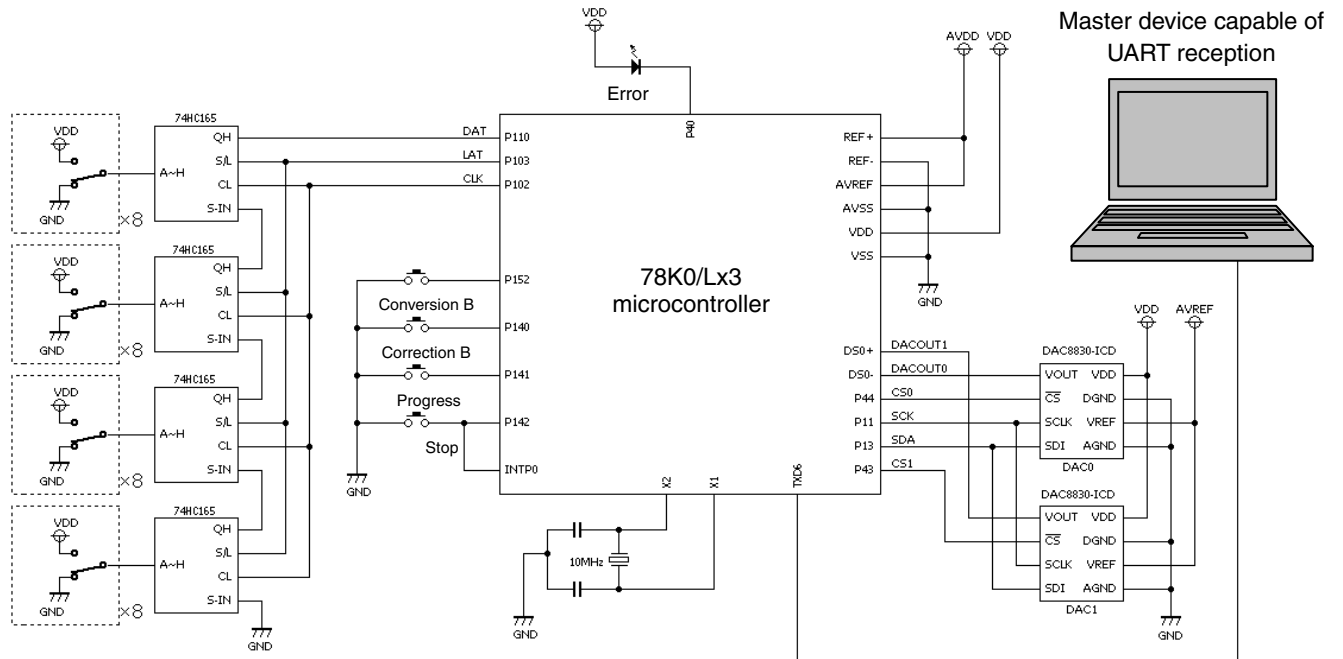


- Cautions**
1. Use the circuit within a voltage range of $2.7\text{ V} \leq AV_{REF} = REF+ \leq V_{DD} \leq 5.5\text{ V}$.
 2. Make the AV_{SS} pin the same potential as V_{SS} and connect it directly to GND.
 3. Leave unused port function pins except those in the circuit diagram open (unconnected), because they are all output ports.
 4. Connect DS0- and DS0+ to a voltage converter for which analog input can be performed within a range of REF- to REF+.
 5. Connect the TXD6 pin to a device that is capable of UART reception.

3.1.2 Circuit diagram when using DAC

The circuit diagram when using a DAC for the $\Delta\Sigma$ -type A/D converter analog input is shown below. DAC8830ICD made by Texas Instruments is used as the DACs in this sample program.

Figure 3-2. Connection When Using DAC



- Cautions 1.** Use the circuit within a voltage range of $2.7\text{ V} \leq AV_{REF} = REF+ \leq V_{DD} \leq 5.5\text{ V}$.
- 2.** Make the AV_{SS} pin the same potential as V_{SS} and connect it directly to GND.
- 3.** Leave unused port function pins except those in the circuit diagram open (unconnected), because they are all output ports.
- 4.** Connect $DS0-$ and $DS0+$ to a voltage converter for which analog input can be performed within a range of $REF-$ to $REF+$.
- 5.** Connect the TxD6 pin to a device that is capable of UART reception.

3.2 Peripheral Hardware

The peripheral hardware to be used is shown below.

(1) Crystal resonators

10 MHz crystal resonators are connected to X1 and X2.

(2) UART communication device (TxD6)

A UART reception device is connected to the TxD6 pin.

(3) Key switches (P140 to P142, P150 to P153)

When not using a DAC, five key switches are connected to the key input ports (P150, P151, P153, P141, P142).

When using a DAC, four key switches are connected to the key input ports (P152, P140 to P142).

(4) 16-bit $\Delta\Sigma$ -type A/D converter analog input pins (DS0–/DS0+)

When not using a DAC for analog input, the DS0– and DS0+ pins are connected to a voltage converter to which analog input can be performed within a range of AV_{SS} to AV_{REF} .

When using a DAC for analog input, the DS0– and DS0+ pins are connected to the DAC output pins. Two units of DAC8830ICD are used in this sample program.

(5) 8-bit serial shift registers (P110, P102, P103)

When using a DAC, four 8-bit serial shift registers (74HC165) are connected to set the conversion start value (16 bits) and conversion end value (16 bits). P110, P103, and P102 are used for the data line, latch, and clock line, respectively.

(6) Error output LED (P40)



An error output LED is connected to P40 for displaying errors of 5-point correction operation coefficients.

CHAPTER 4 SOFTWARE


This chapter describes the file configuration of compressed files to be downloaded, internal peripheral functions of the microcontroller to be used, initial settings and operation overviews of the sample program, UART transmission data format, and flowcharts.

4.1 File Configuration

The file configuration of the compressed files to be downloaded is shown below.

File Name	Description	Compressed (*.zip) File Included	
			
main.c	Source file of hardware initialization processing of the microcontroller, main processing, key scan processing, event processing, conversion A1 mode control processing, conversion A2 mode control processing, conversion B mode control processing, correction A mode control processing, and correction B mode control processing ^{Note}	●	●
K0Lx3_DSAD.prw	Work space file for integrated development environment PM+		●
K0Lx3_DSAD.prj	Project file for integrated development environment PM+		●

Note The source file to be downloaded includes processing of both when not using and when using a DAC for A/D conversion analog input. Delete unnecessary processing according to the operation environment.

Remark  : Only the source file is included.



 : The files to be used with integrated development environment PM+ are included.

4.2 Internal Peripheral Functions to Be Used

The following peripheral functions provided in the microcontroller are used in this sample program.

- For measuring the key scan cycle timing:
8-bit timer H1 is used as an interval timer of a cycle of about 10 ms ($f_{PRS}/2^{12} \times 23$).
- For measuring the peripheral circuit stabilization time after setting the DAC:
8-bit timer H0 is used as an interval timer with a reference time of a cycle of about 100 μ s ($f_{PRS}/2^{10}$).
- For A/D conversion:
DS0– and DS0+ of the 16-bit $\Delta\Sigma$ -type A/D converter are used.
- For A/D conversion result output:
Serial interface UART6 is used.

4.3 Initial Settings and Operation Overviews

In this sample program, the clock frequency is selected and 8-bit timer H0, 8-bit timer H1, serial interface UART6, the 16-bit $\Delta\Sigma$ -type A/D converter, and I/O ports are set in the initial settings.

Operation overviews are described in **4.3.1 Operation when not using DAC** and **4.3.2 Operation when using DAC**.

This sample program includes the processing of both when not using and when using a DAC for A/D conversion analog input. Delete unnecessary processing according to the operation environment and reference the **4.3.1 Operation when not using DAC**, or **4.3.2 Operation when using DAC**.

4.3.1 Operation when not using DAC

Variables are initialized and key scan processing, event processing, conversion A1 mode control processing, conversion A2 mode control processing, and correction A mode control processing are called after the initial settings of the peripherals to be used have been completed. Key scan processing is called in a cycle of about 10 ms. Furthermore, conversion A1 mode control processing, conversion A2 mode control processing, and correction A mode control processing are called according to the current mode and system status.

(1) Key scan processing

The key statuses are scanned from the seven key input ports. Chattering is removed three times and the key statuses are encoded.

(2) Event processing

When a key code has changed, the mode (conversion A1, conversion A2, or correction A) or the conversion execution status (during conversion execution, during standby, or stopped) is switched according to key code. The mode can be switched only if the conversion status is "During standby".

(3) Conversion A1 mode control processing

The following processing is executed every time the progress key is pressed. The conversion operation ends by pressing the stop key.

- A/D conversion using the 16-bit $\Delta\Sigma$ -type A/D converter
- 5-point correction (executed only if the 5-point correction data have been measured)
- Temperature correction (executed only if 5-point correction has been completed and temperature errors exist)
- Outputting the A/D conversion, 5-point correction, and temperature correction results via UART^{Note}

Temperature correction is performed if temperature errors exist in the 5-point correction data (code1) measured immediately after conversion A1 mode has been entered. Consequently, the ana1 value must be input to the analog input when entering conversion A1 mode.

The A/D conversion, 5-point correction, and temperature correction results are output via UART, but if 5-point correction and temperature correction are not performed, the corresponding sections of the transmit data will be "*****"^{Note}.

Note See **4.4 UART Transmission Data Format** for details of the UART transmit data.

(4) Conversion A2 mode control processing

The following processing is successively performed by pressing the progress key. The conversion operation ends by pressing the stop key.

- A/D conversion using the 16-bit $\Delta\Sigma$ -type A/D converter
- 5-point correction (executed only if the 5-point correction data have been measured)
- Temperature correction (executed only if 5-point correction has been completed and temperature errors exist)
- Outputting the A/D conversion, 5-point correction, and temperature correction results via UART^{Note}

Temperature correction is performed if temperature errors exist in the 5-point correction data (code1) measured immediately after conversion A2 mode has been entered. Consequently, the ana1 value must be input to the analog input when entering conversion A2 mode.

The A/D conversion, 5-point correction, and temperature correction results are output via UART, but if 5-point correction and temperature correction are not performed, the corresponding sections of the transmit data will be ^{Note}*****.

(5) Correction A mode control processing

In correction A mode, 5-point correction data is measured. The following processing is performed once by pressing the progress key. The conversion operation ends by pressing the stop key. Furthermore, the conversion operation ends when the following processing is performed five times.

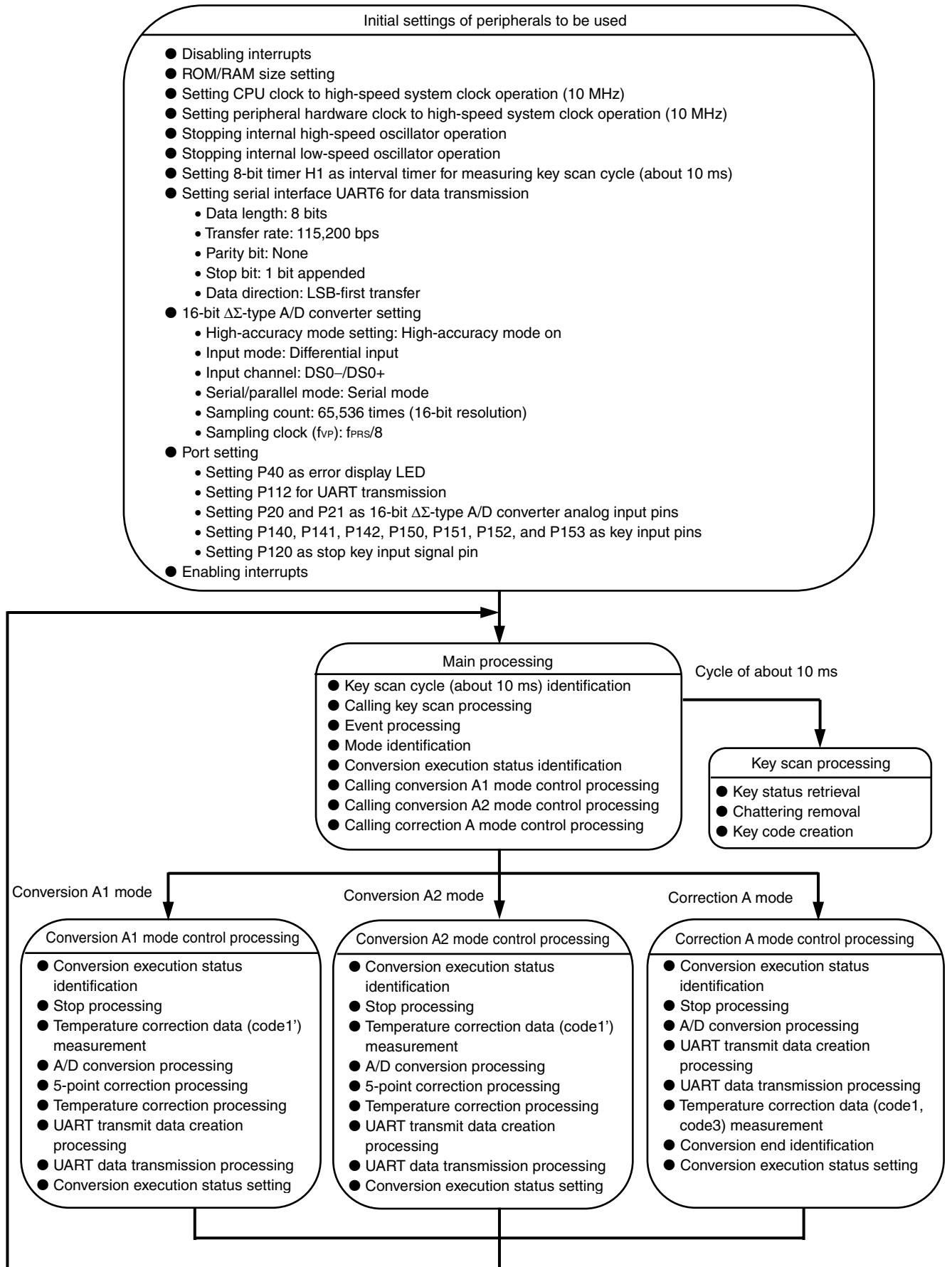
- A/D conversion using the 16-bit $\Delta\Sigma$ -type A/D converter
- Outputting the A/D conversion result via UART^{Note}

code1 to code5 are sequentially measured each time the progress key is pressed. Consequently, ana1 to ana5 must be sequentially input to the analog input.

An error display LED is lit if the sizes of 5-point correction data code1 to code 5 are not in the order of code1 < code3 < code5. code1 and code3 are corrected via 5-point correction and saved as temperature correction data if the 5-point correction data have been correctly measured.

Note See **4.4 UART Transmission Data Format** for details of the UART transmit data.

Details are shown in the following state transition diagram (state chart).



4.3.2 Operation when using DAC

Variables are initialized and key scan processing, event processing, conversion B mode control processing, and correction B mode control processing are called after the initial settings of the peripherals to be used have been completed. Key scan processing is called in a cycle of about 10 ms. Furthermore, conversion B mode control processing and correction B mode control processing are called according to the current mode and system status.

(1) Key scan processing

The key statuses are scanned from the seven key input ports. Chattering is removed three times and the key statuses are encoded.

(2) Event processing

When a key code has changed, the mode (conversion B or correction B) or the conversion execution status (during conversion execution, during standby, or stopped) is switched according to key code. The mode can be switched only if the conversion status is "During standby".

(3) Conversion B mode control processing

When conversion B mode is entered, the conversion start and end values are read from 74HC165 and the following processing is successively performed by pressing the progress key. The conversion operation ends if the stop key is pressed or if the values set for DAC output become equal to the conversion end values, which have been read.

- DAC output setting
- A/D conversion using the 16-bit $\Delta\Sigma$ -type A/D converter
- 5-point correction (executed only if the 5-point correction data have been measured)
- Temperature correction (executed only if 5-point correction has been completed and temperature errors exist)
- Outputting the A/D conversion, 5-point correction, and temperature correction results via UART^{Note}

The values to be set to the DAC are incremented every time A/D conversion is performed. The values from the conversion start value to the conversion end value, which have been read, are sequentially set for DAC output and A/D converted. If the conversion end value is smaller than the conversion start value, the DAC setting values are rounded and the values from the conversion start value to the conversion end value are set for DAC output.

Temperature correction is performed if temperature errors exist in the 5-point correction data (code1) measured immediately after conversion B mode has been entered. code1 is measured by setting ana1 to the DAC.

The A/D conversion, 5-point correction, and temperature correction results are output via UART, but if 5-point correction and temperature correction are not performed, the corresponding sections of the transmit data will be "*****"^{Note}.

Note See 4.4 UART Transmission Data Format for details of the UART transmit data.

(4) Correction B mode control processing

In correction B mode, 5-point correction data is measured. The following processing is successively performed five times by pressing the progress key. The conversion operation ends by pressing the stop key or when the following processing has been completed five times.

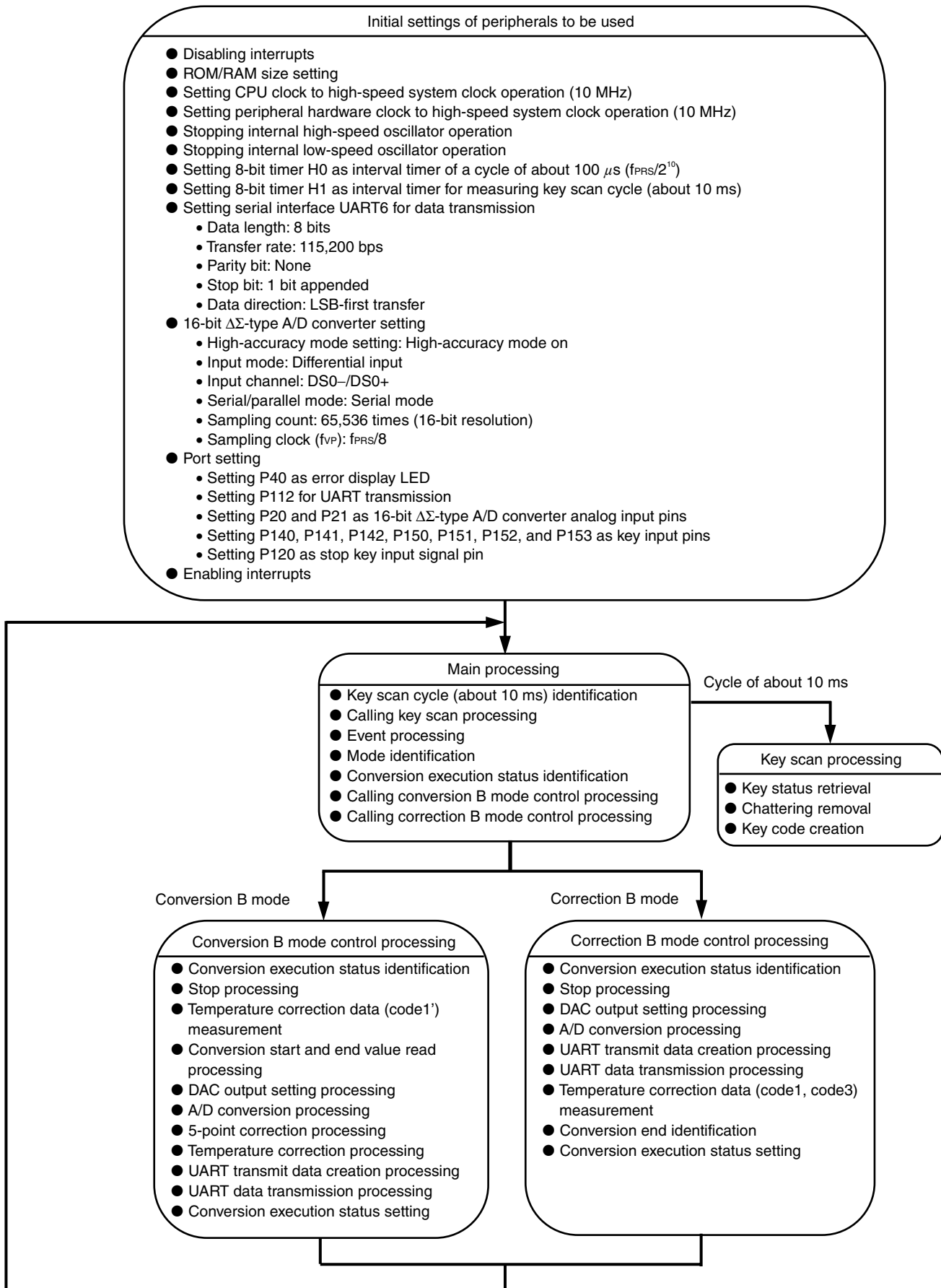
- DAC output setting
- A/D conversion using the 16-bit $\Delta\Sigma$ -type A/D converter
- Outputting the A/D conversion result via UART^{Note}

ana1 to ana5 are sequentially set for DAC output and code1 to code5 are measured. An error display LED is lit if the sizes of code1 to code5 are not in the order of code1 < code3 < code5. code1 and code3 are corrected via 5-point correction and saved as temperature correction data if the 5-point correction data have been correctly measured.

The A/D conversion, 5-point correction, and temperature correction results are output via UART, but if 5-point correction and temperature correction are not performed, the corresponding sections of the transmit data will be "*****"^{Note}. The conversion operation ends by pressing the stop key.

Note See 4.4 **UART Transmission Data Format** for details of the UART transmit data.

Details are shown in the following state transition diagram (state chart).



4.4 UART Transmission Data Format

The data to be transmitted via UART6 is described below.

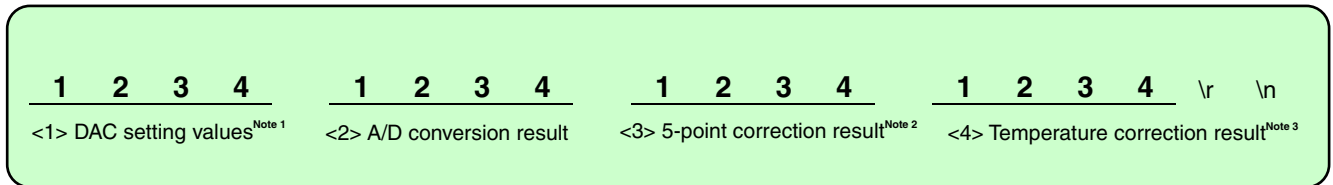
The UART settings are as follows.

- Baud rate: 115,200 bps
- Data character length: 8 bits
- Parity bit: Not output
- Number of stop bits: 1

Data transmission is performed once for every A/D conversion. The length of data transmitted at one time is 21 bytes. The DAC setting values, A/D conversion result, 5-point correction result, and temperature correction result are converted into ASCII codes in hexadecimal and then transmitted.

Figure 4-1 shows the contents of the data.

Figure 4-1. UART Transmission Data Format

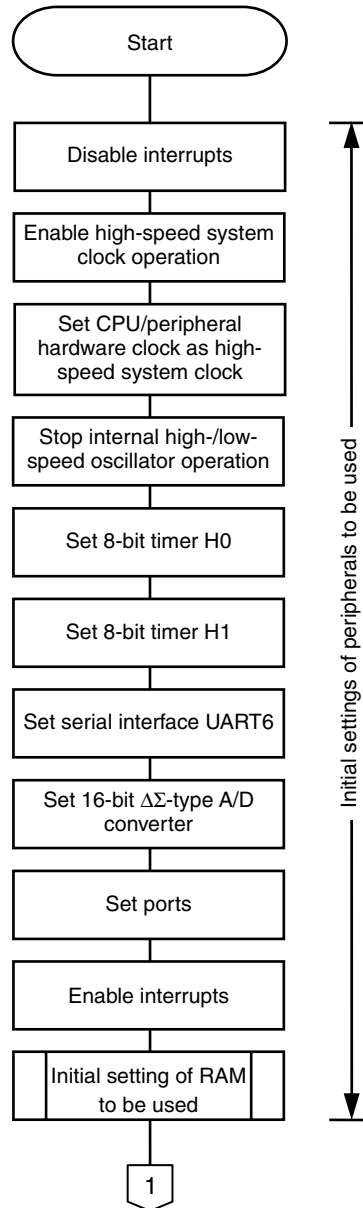


- Notes**
1. When not using a DAC, the data becomes "*****".
 2. When not performing 5-point correction, the data becomes "*****".
 3. When not performing temperature correction, the data becomes "*****".

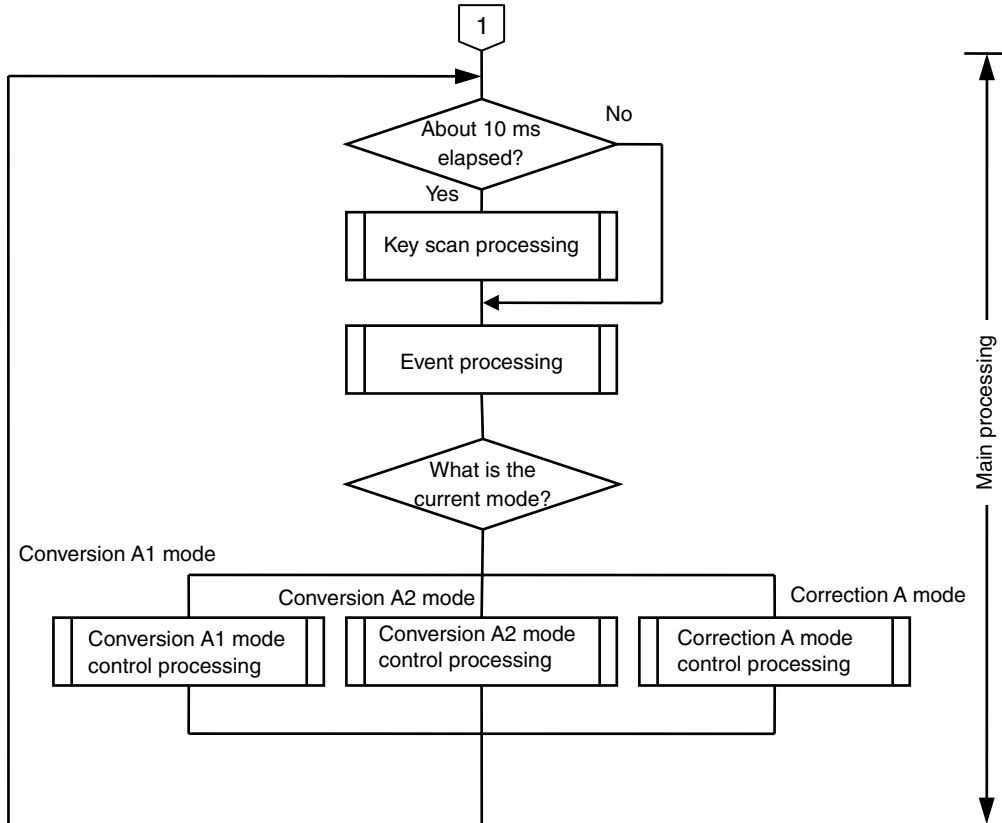
4.5 Flowcharts

The flowcharts of this sample program are shown below.

<Initialization processing after reset release>



● Main processing when not using a DAC



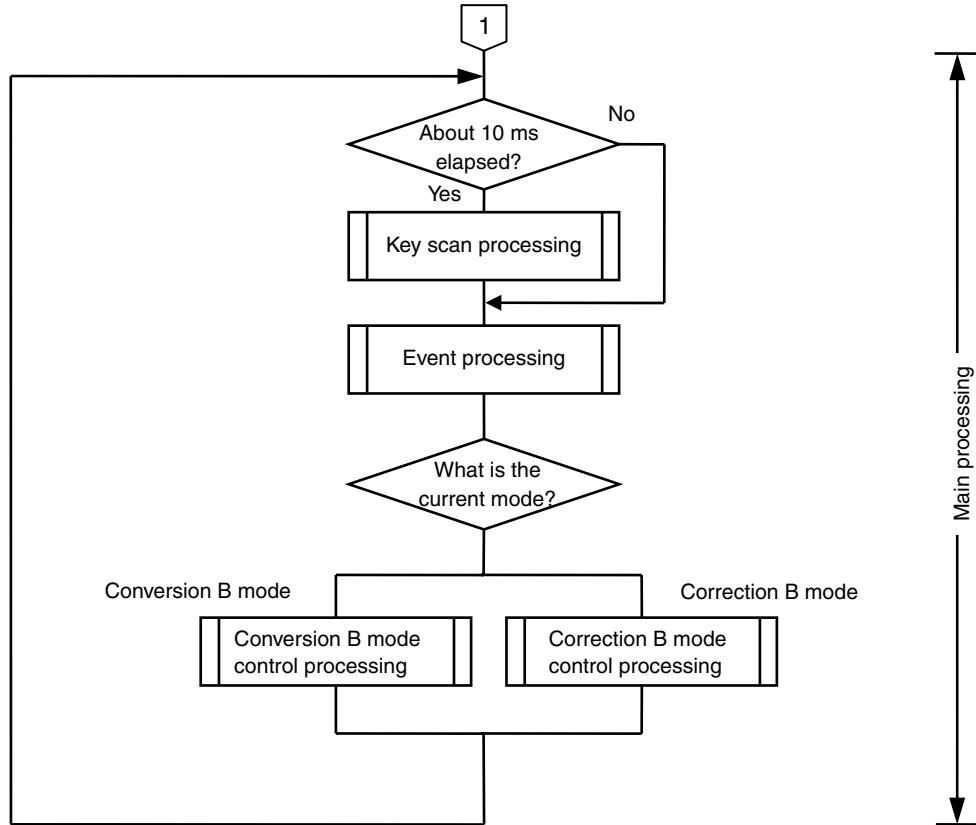
Cautions 1. Set the option byte via the RA78K0 linker options. See the RA78K0 Assembler Package User's Manual for the setting method.

The following items can be set using the option byte.

- Watchdog timer operation
- LVI setting upon reset release (upon power activation)
- On-chip debug operation control

2. This sample program includes the processing of both when using and not using a DAC for analog input. Delete unnecessary processing according to the operation environment.

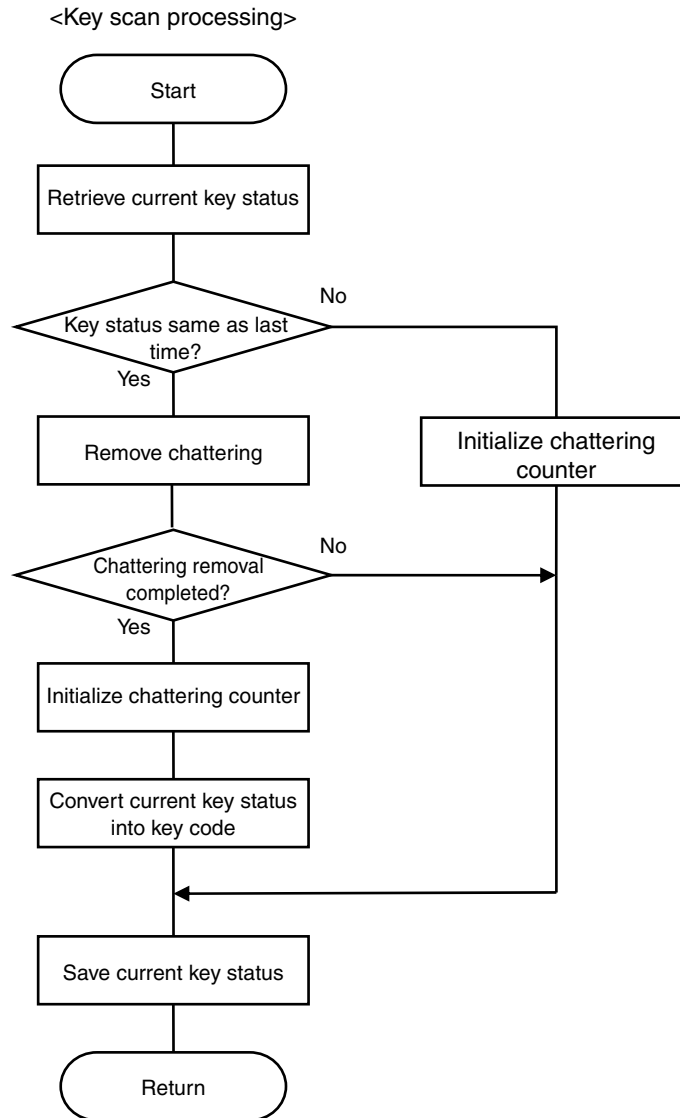
● Main processing when using a DAC



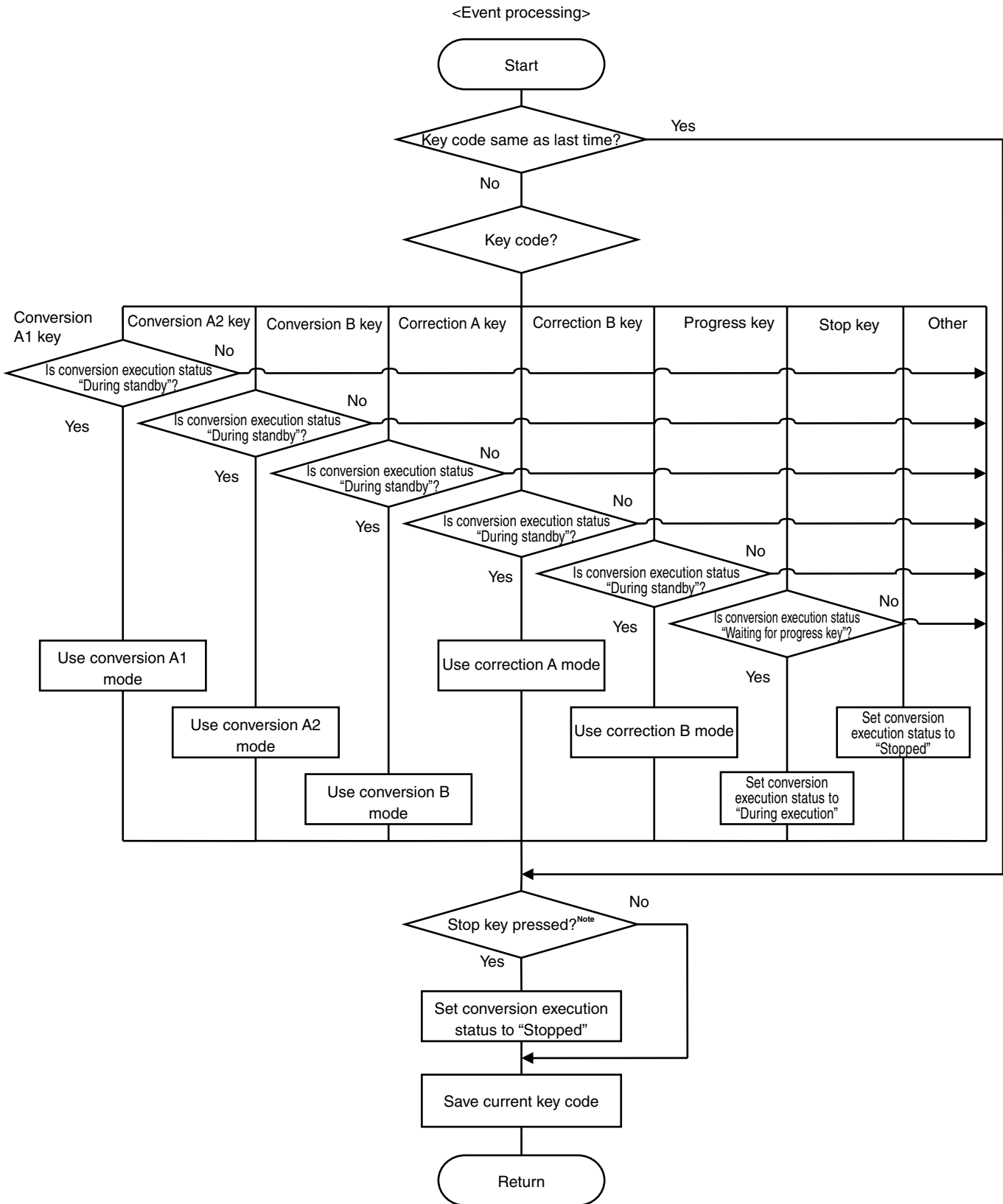
Cautions 1. Set the option byte via the RA78K0 linker options. See the RA78K0 Assembler Package User's Manual for the setting method.

The following items can be set using the option byte.

- Watchdog timer operation
 - LVI setting upon reset release (upon power activation)
 - On-chip debug operation control
2. This sample program includes the processing of both when using and not using a DAC for analog input. Delete unnecessary processing according to the operation environment.

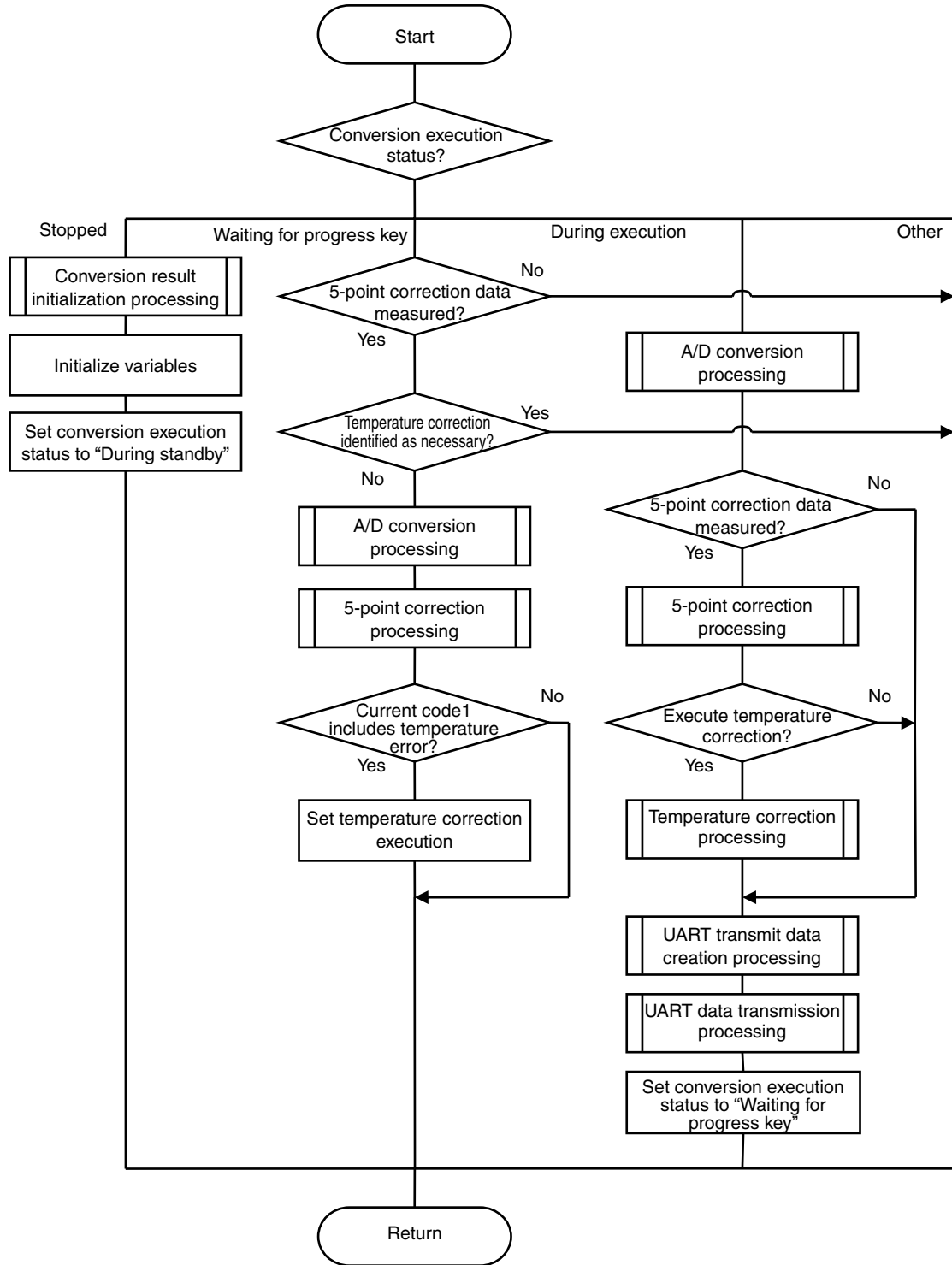


Caution Pressing a key multiple times is invalid.

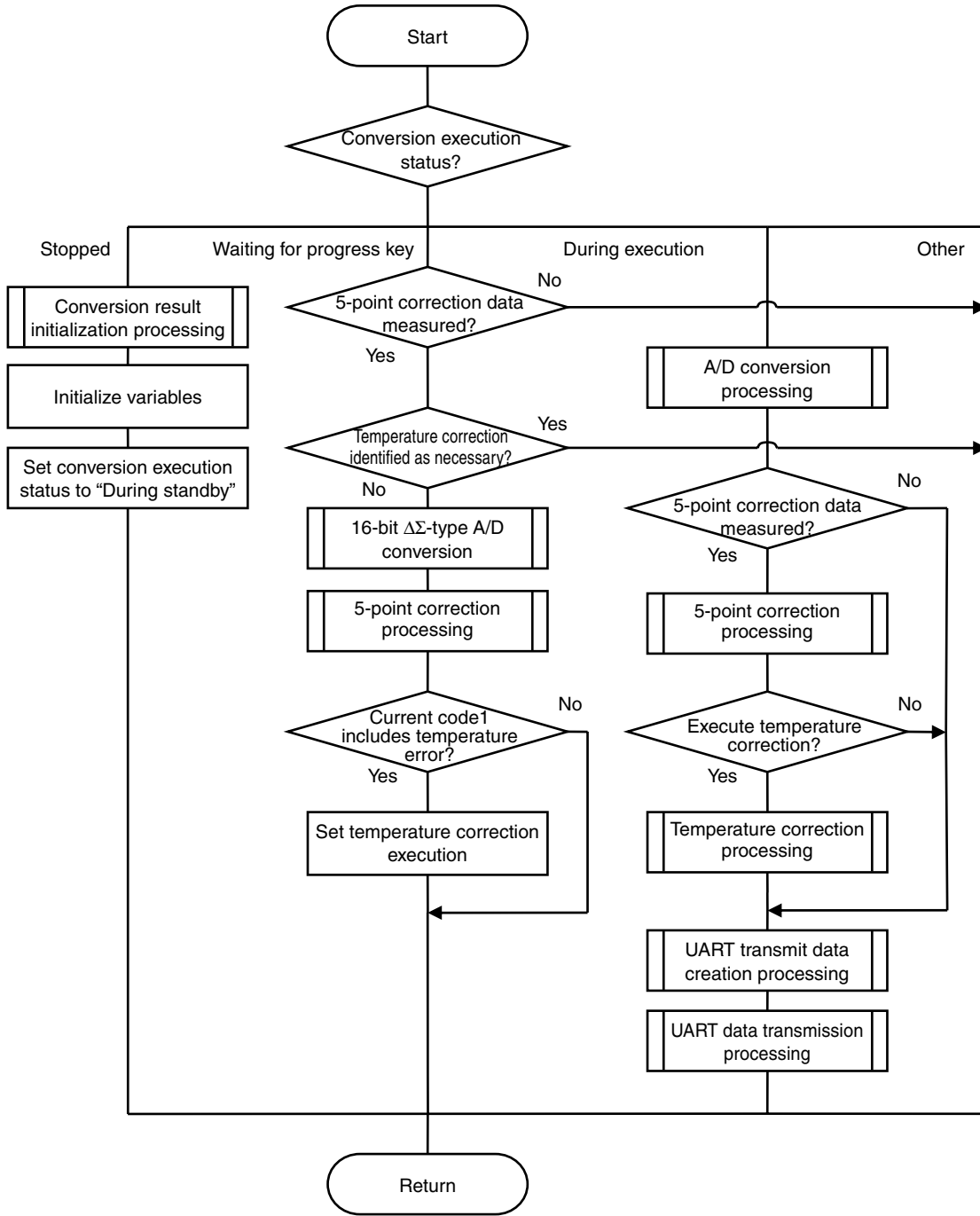


Note Use the interrupt request flag (PIF0) to determine whether the stop key has been pressed. Key scan processing is executed in a cycle of about 10 ms, but may be delayed due to A/D conversion processing.

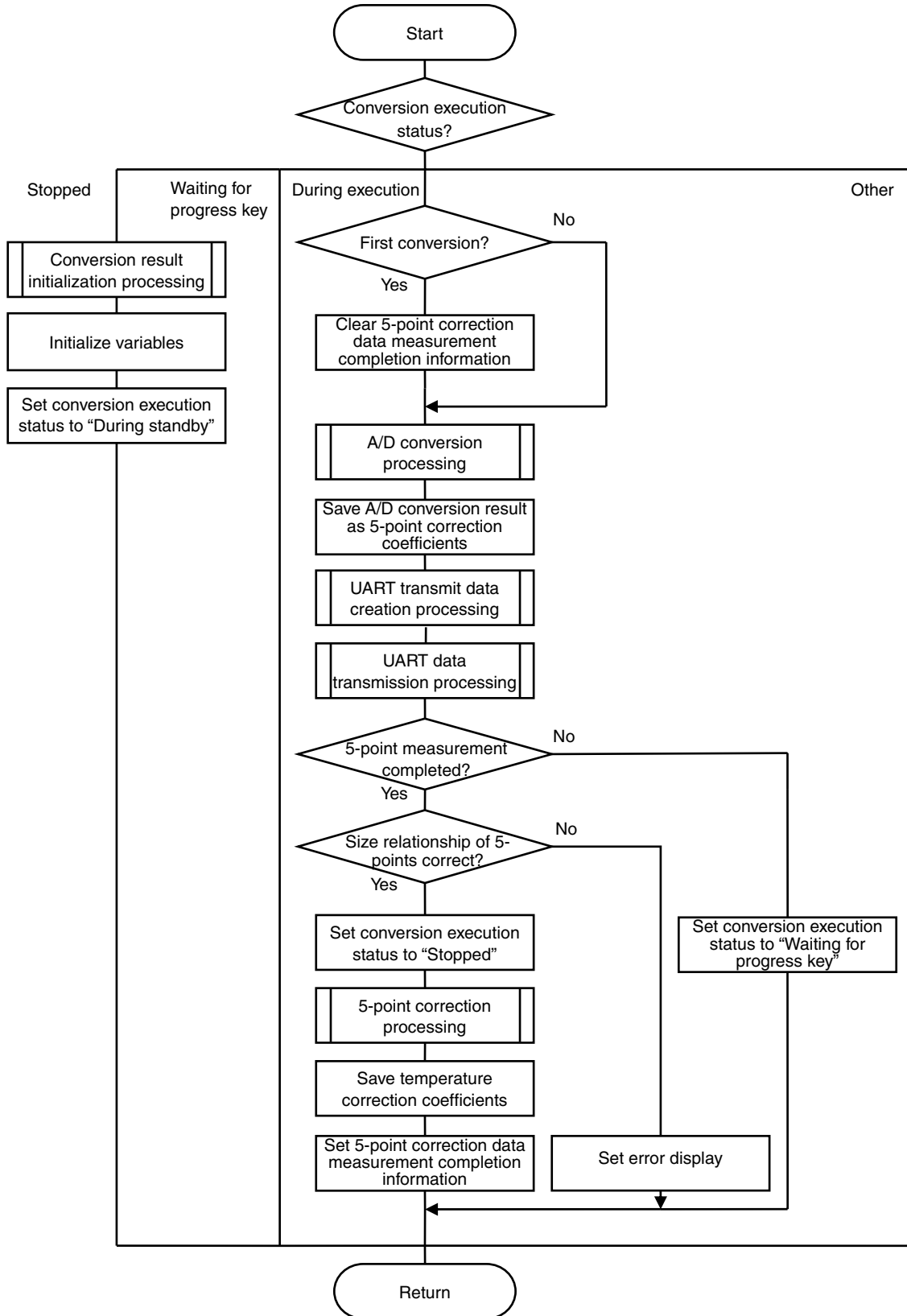
<Conversion A1 mode control processing>



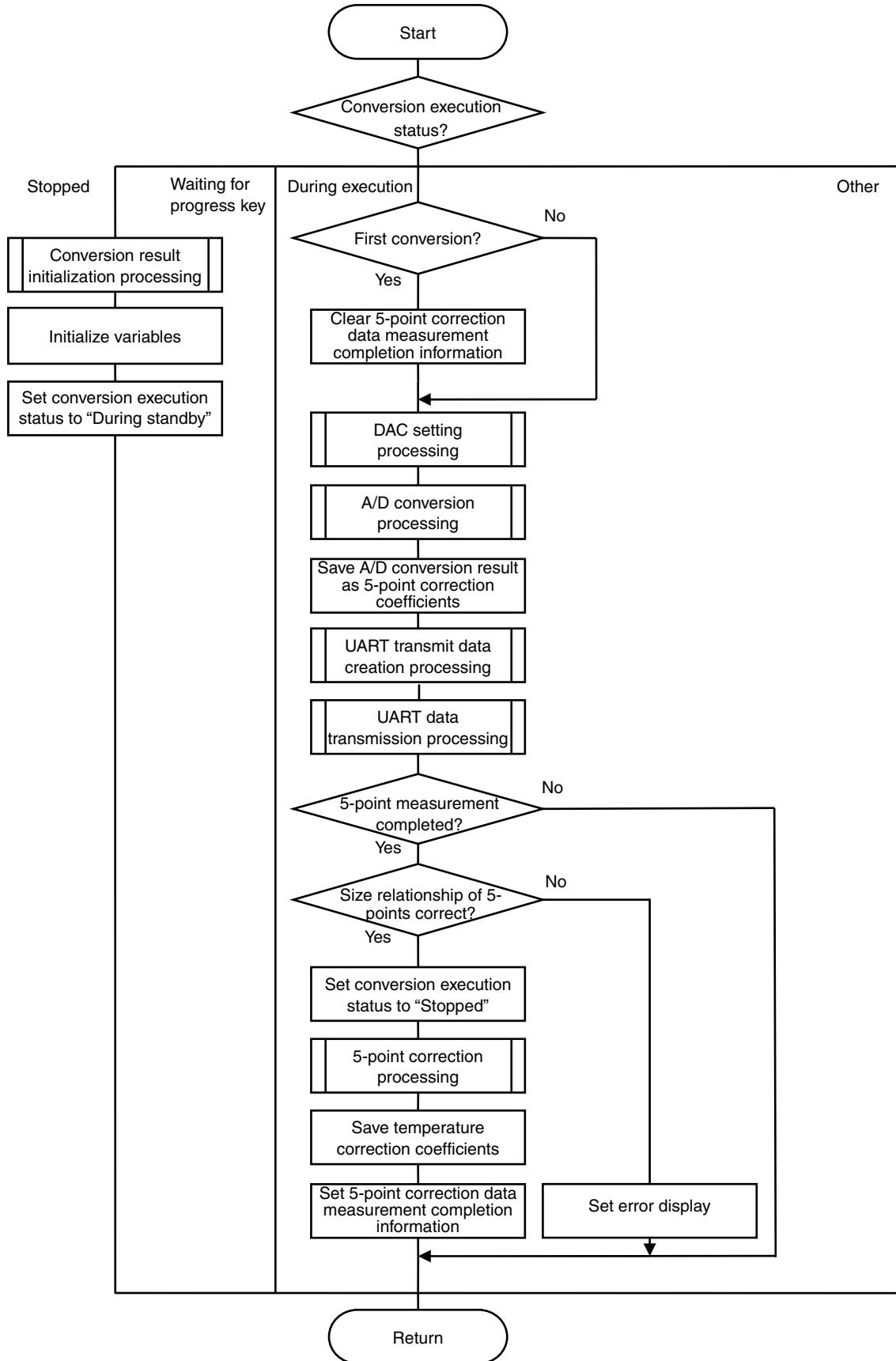
<Conversion A2 mode control processing>



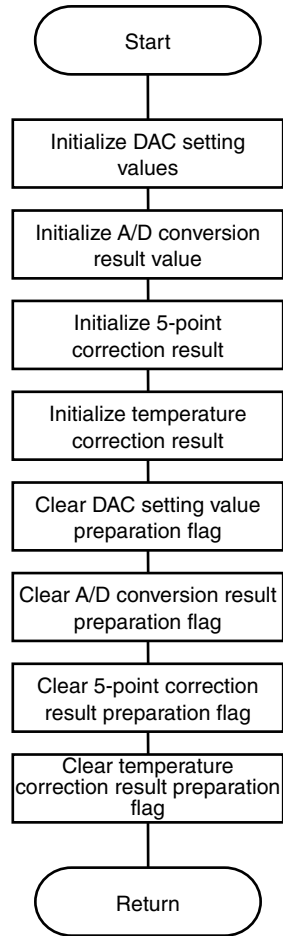
<Correction A mode control processing>



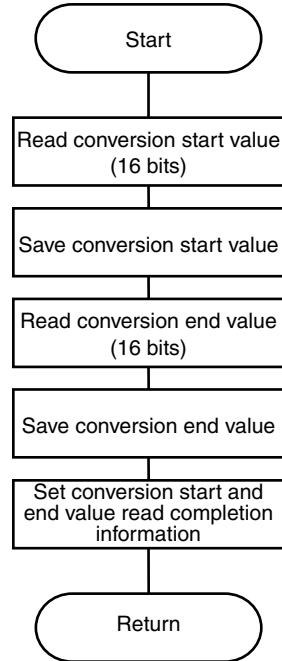
<Correction B mode control processing>



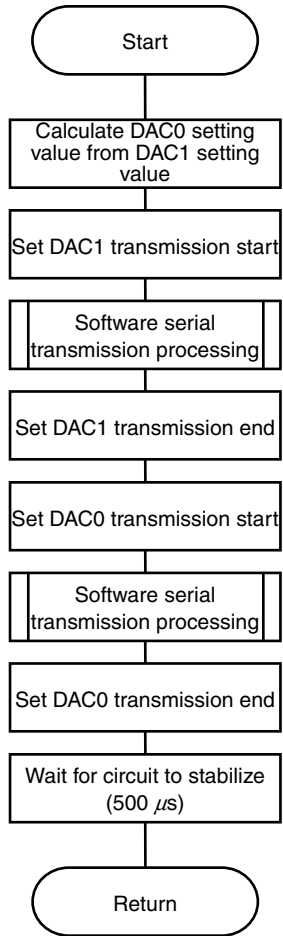
<Conversion result initialization processing>



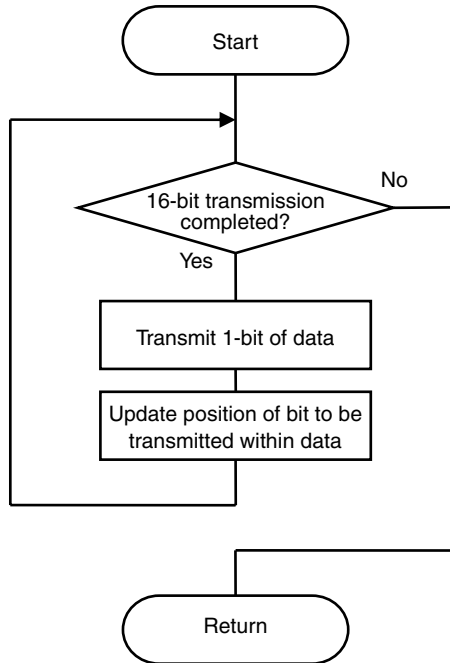
<Conversion start and end value read processing>



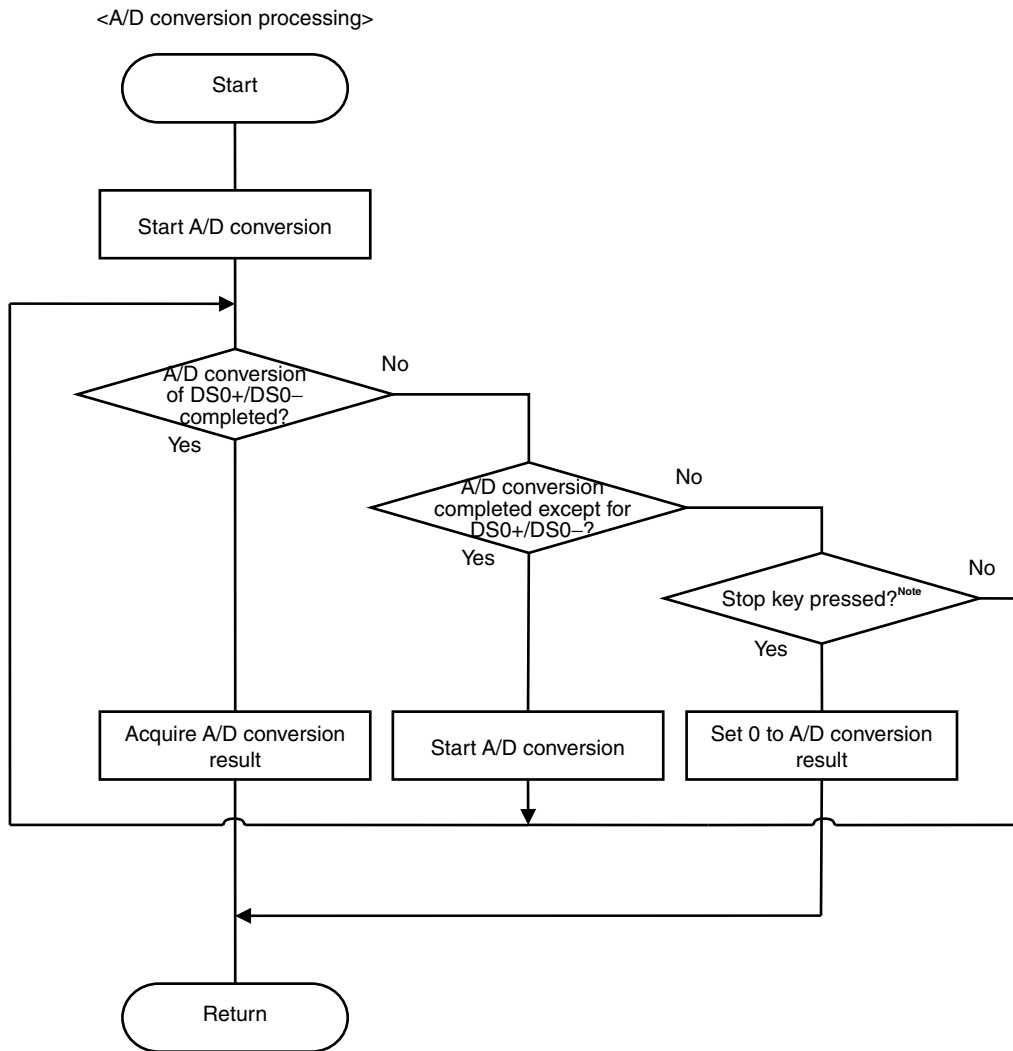
<DAC setting processing>



<Software serial transmission processing>

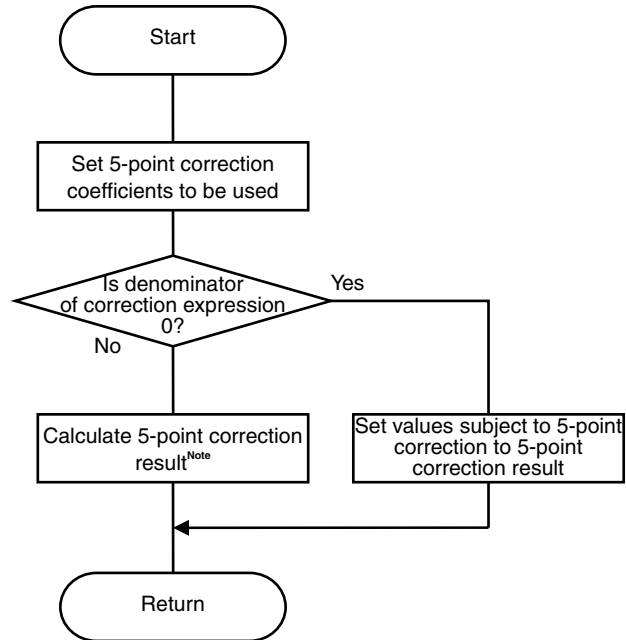


Caution Data is sequentially transmitted from the MSB, bit by bit.



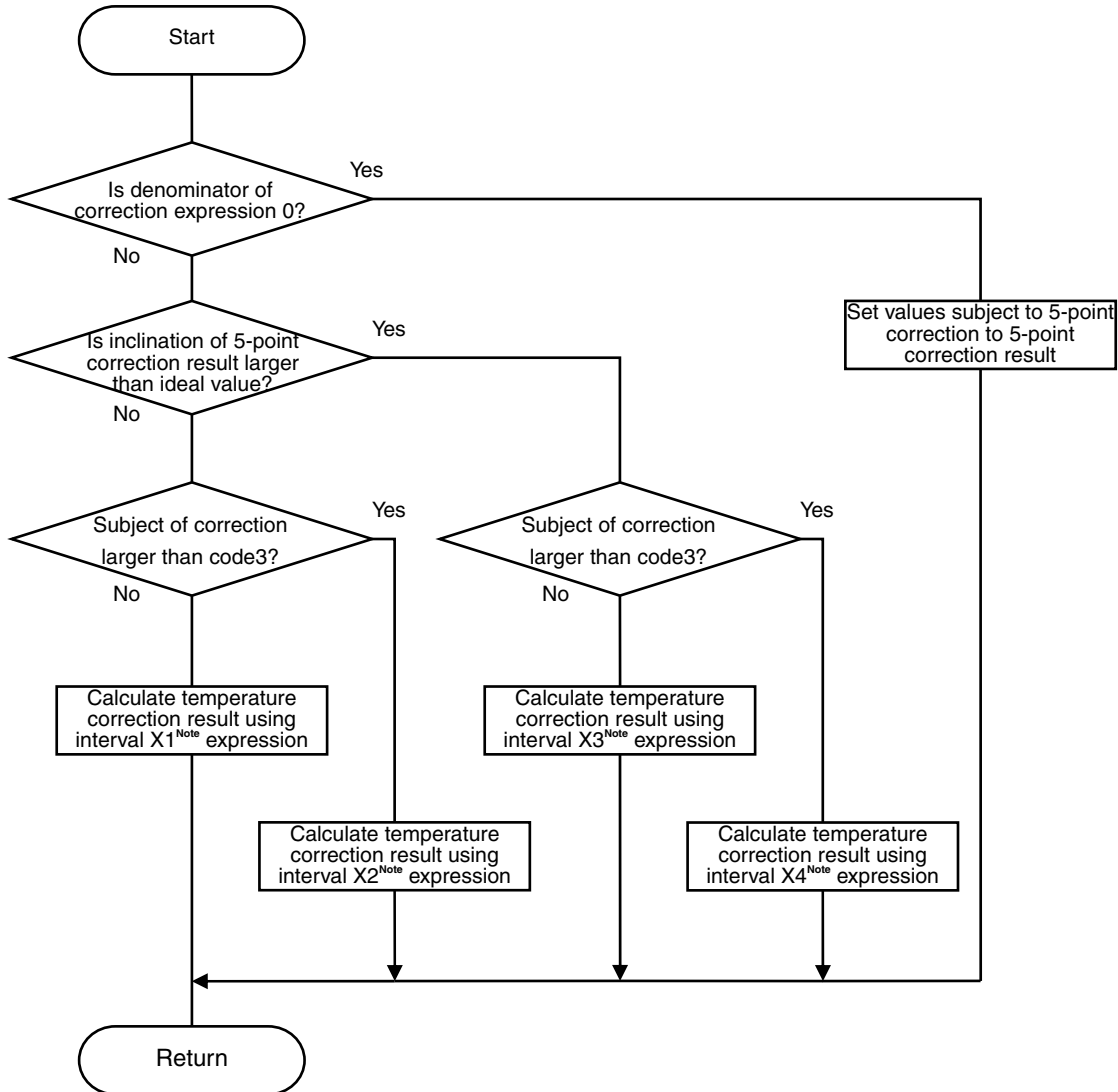
Note Use the interrupt request flag (PIF0) to determine whether the stop key has been pressed. Key scan processing is executed in a cycle of about 10 ms, but may be delayed due to A/D conversion processing.

<5-point correction processing>



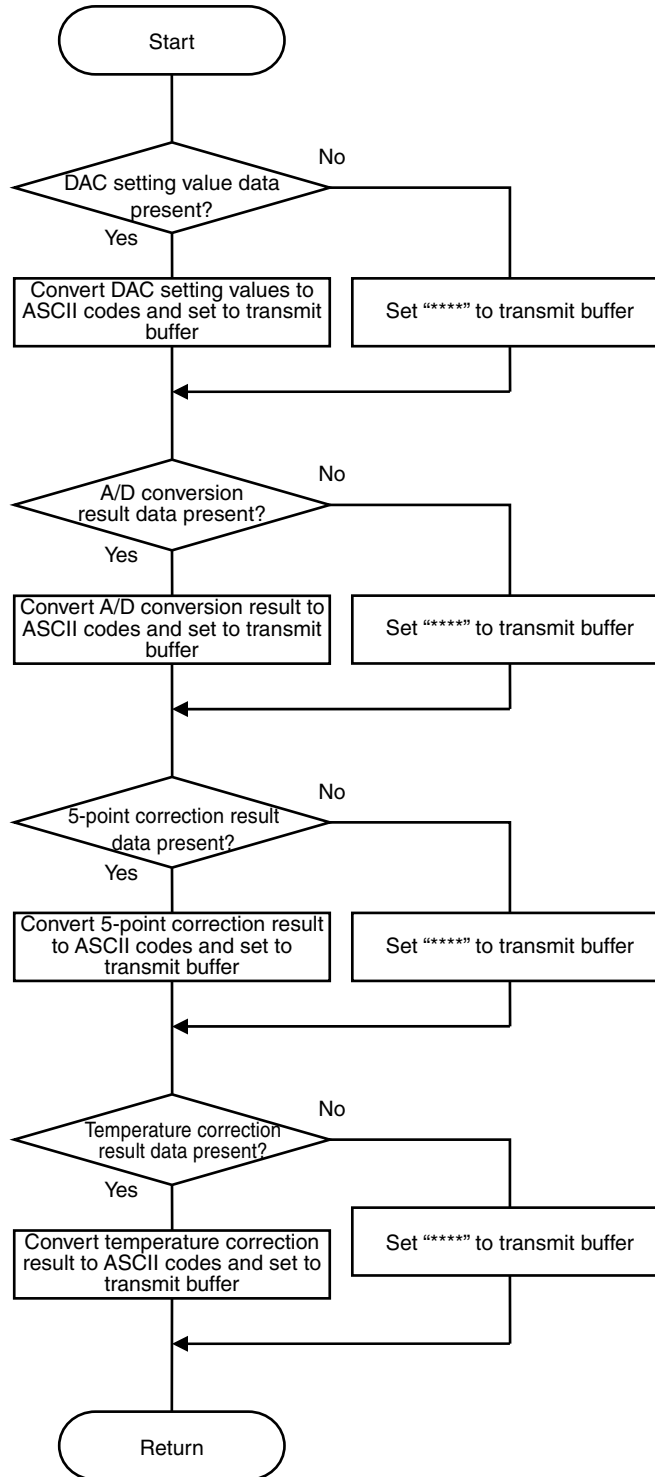
Note See 2.1 5-Point Correction for details of the 5-point correction arithmetic expression.

<Temperature correction processing>



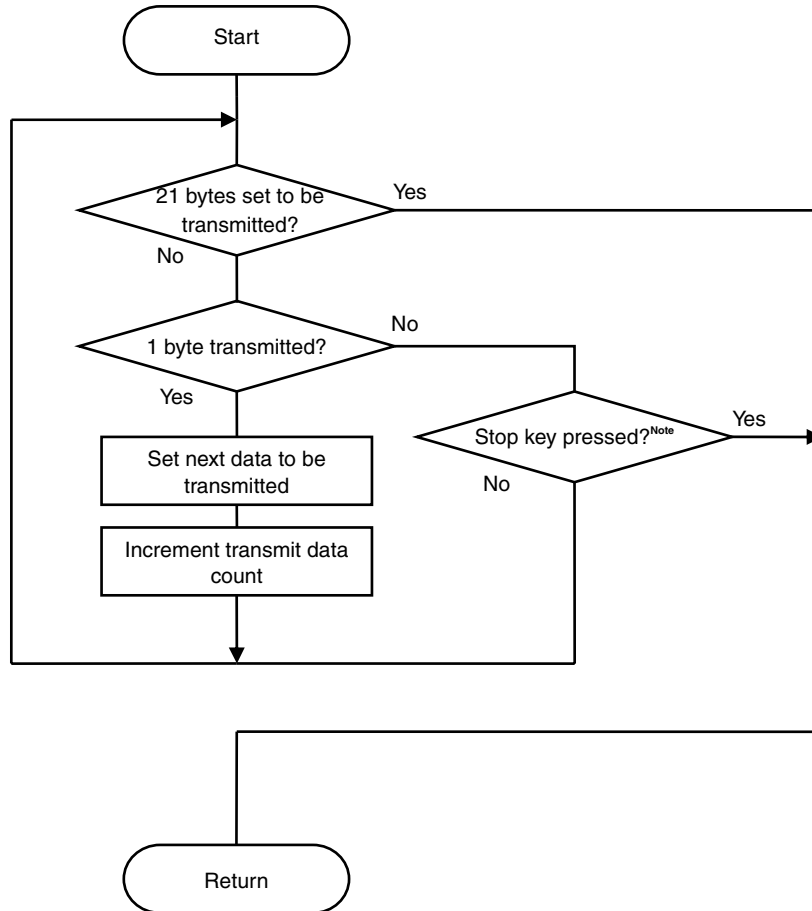
Note See 2.2 Temperature Correction for details of the temperature correction arithmetic expression.

<UART transmission data creation processing>



Caution See 4.4 UART Transmission Data Format for details of the UART transmit data.

<UART data transmission processing>



Note Use the interrupt request flag (PIF0) to determine whether the stop key has been pressed. Key scan processing is executed in a cycle of about 10 ms, but may be delayed due to A/D conversion processing.

Caution The data is sequentially transmitted from the MSB, bit by bit.

CHAPTER 5 SETTING METHOD

This chapter describes $\Delta\Sigma$ A/D processing.

See the user's manual (78K0/LE3, 78K0/LF3) of each product for details of register setting methods.

5.1 Initial Settings of Peripherals to Be Used

Initial settings of the following SFRs are performed in the initialization processing after reset release.

- <1> A/D port configuration register 0 (ADPC0)
Specifies pins to be used as analog inputs.
- <2> Port mode register 2 (PM2)
Sets analog input ports to input mode.
- <3> A/D converter power supply (ADDPON)
Turns on the A/D converter power supply.
- <4> 16-bit $\Delta\Sigma$ -type A/D converter control register 1 (ADDCTL1)
Sets the sampling count, sampling clock, and operation mode.
- <5> 16-bit $\Delta\Sigma$ -type A/D converter control register 0 (ADDCTL0)
Sets A/D operation to $\Delta\Sigma$.
- <6> Leave an interval of at least 5 μ s between turning on the A/D converter power supply and enabling operation.
Waits for the A/D power supply to stabilize.
- <7> Conversion operation enable bit (ADDCE)
Enables A/D conversion operation.
- <8> A/D conversion completion interrupt request flag (DSADIF)
Clears the interrupt request flag, because it is used for A/D conversion completion.

```

/*-----
16-bit ΔΣ-type A/D converter
-----*/
<1>..... ADPC0 = 0b00000000;          /* A/D port configuration register 0 */
          /*|||++++----- ADPC03/ADPC02/ADPC01/ADPC00: Set P20 to P27 as
analog inputs (ΔΣ-type) */
          /*++++----- <Fix to 0000> */

<2>..... PM2 = 0b11111111;          /* Set PM2 I/O */
          /*|||++----- PM20: Use input (1) as DS0- */
          /*|||++----- PM21: Use input (1) as DS0+ */
          /*|||++----- PM22: Use input (1) as DS1- */
          /*|||++----- PM23: Use input (1) as DS1+ */
          /*||+----- PM24: Use input (1) as DS2- */
          /*||+----- PM25: Use input (1) as DS2+ */
          /*|+----- PM26: Use input (1) as REF- */
          /*+----- PM27: Use input (1) as REF+ */

          P2 = 0b00000000;          /* Set initial P2 value */
          /*+++++----- P27/P26/P25/P24/P23/P22/P21/P20: Lo(0) */

<3>..... ADDPON = 1;                /* Turn on A/D converter power supply */
<4>..... ADDCTL1=0b01100111;        /* 16-bit ΔΣ-type A/D converter control register 1 */
          /*|||++++----- ADDN2/ADDN1/ADDN0: Specify sampling count
(resolution) to 65,536 times (16 bits) */
          /*||++----- <Fix to 00> */
          /*|+----- ADDTS: Set serial mode */
          /*++----- ADDFS1/ADDFS0: Select fPRS/8 as sampling clock */

<5>..... ADDCTL0=0b10110000;        /* 16-bit ΔΣ-type A/D converter control register 0 */
          /*|||++----- ADDS1/ADDS0: Specify DS0+/DS0- as analog input */
          /*|||++----- <Fix to 00> */
          /*||+----- AINMOD: Specify differential input as input mode */
          /*|+----- HAC: Set high-accuracy mode to on */
          /*|+----- ADDCE: Stop conversion */
          /*+----- ADDPON: Turn on A/D converter power supply */

<6>..... for( i = 20; i < 0; i--);  /* Leave interval of at least 5 us between turning on
A/D converter power supply and enabling operation */
<7>..... ADDCE = 1;                  /* Enable conversion operation */
<8>..... DSADIF = 0;                 /* Clear A/D conversion completion interrupt
request flag */

```

5.2 A/D Conversion Processing

The following operations are performed in $\Delta\Sigma$ A/D conversion processing.

An A/D conversion operation for the voltage applied to analog input channel 0 (DS0+/DS0-) is performed. See the user's manual (78K0/LE3, 78K0/LF3) of each product for details of A/D conversion.

- <1> A/D conversion operation is enabled.
- <2> The A/D conversion end timing is determined upon an A/D conversion completion interrupt request, so the interrupt request is cleared.
- <3> The conversion result is read after waiting for the end of A/D conversion, because A/D conversion is always operated in this application.

The conversion end channel (ADDSTR) is not required to be checked, because only channel 0 is used in this application. However, whether ADDSTR is the target channel, channel 0, is checked just in case.

Re-conversion is performed if ADDSTR is another channel.

- <4> A/D conversion loops until the end of A/D conversion in this application, so external interrupts are checked to end A/D conversion midway.
- <5> The A/D conversion result is returned as return values.

```

/*****
16-bit  $\Delta\Sigma$ -type A/D conversion
-----
 $\Delta\Sigma$ -type A/D conversion is started and A/D conversion result is returned.
*****/
static unsigned short fn_GetAD(void)
{
    unsigned short ret = 0;          /* A/D conversion result return value */

    /* Start A/D conversion */
    ADDCE = 1;                      /* Enable conversion operation */
    DSADIF = 0;                     /* Clear A/D conversion completion
    interrupt request */

    /* Acquire A/D conversion result */
    while(1){ /* Wait until A/D conversion is completed or stop key is pressed */
        if(( DSADIF )&&( ADDSTR == 0 )){ /* If A/D conversion of DS0+
is completed */
            ret = ADDCR;                /* Read A/D conversion result
*/
            break;
        }else if( ( DSADIF )&&( ADDSTR != 0 )){ /* If value of non-
target channel is converted */
            ADDCE = 1;                /* Redo A/D conversion */
            DSADIF = 0;
        }else if(PIF0){ /* If stop key is pressed */
            ADDCE = 0;                /* Stop A/D conversion
operation */
            break;                    /* Abort A/D conversion */
        }else
            ;
    }

    return ( ret );                  /* Return A/D conversion result */
}

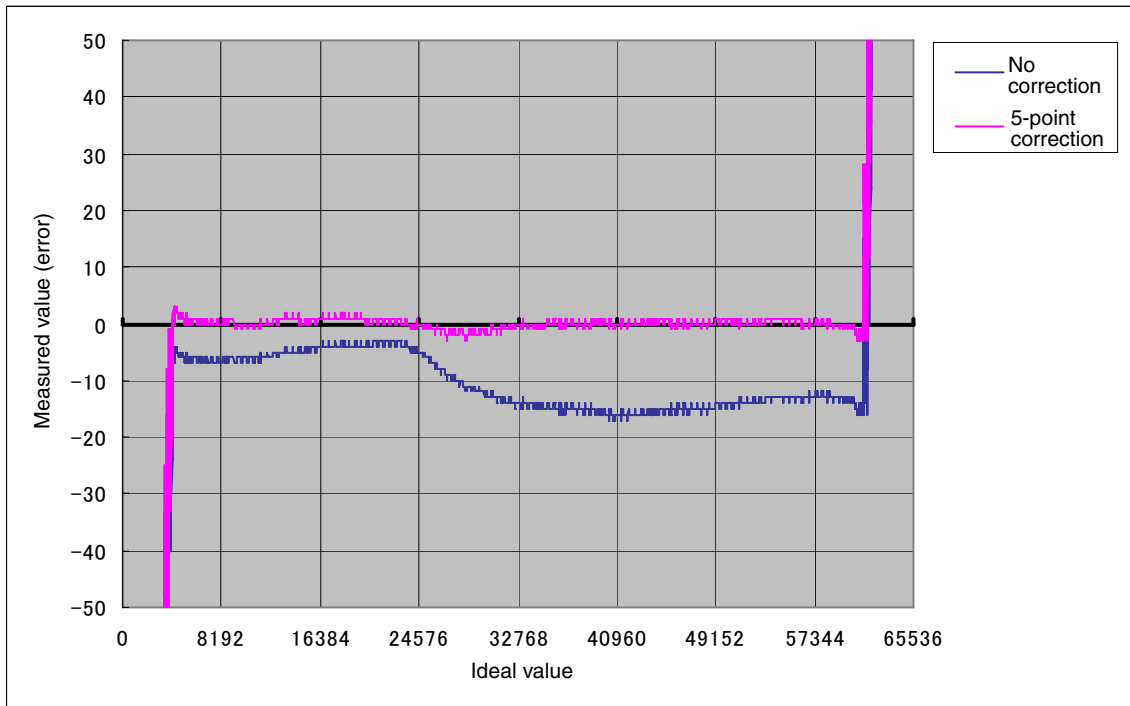
```

CHAPTER 6 OPERATION CHECK EXAMPLE USING DEVICE

This chapter describes examples of A/D conversion value correction.

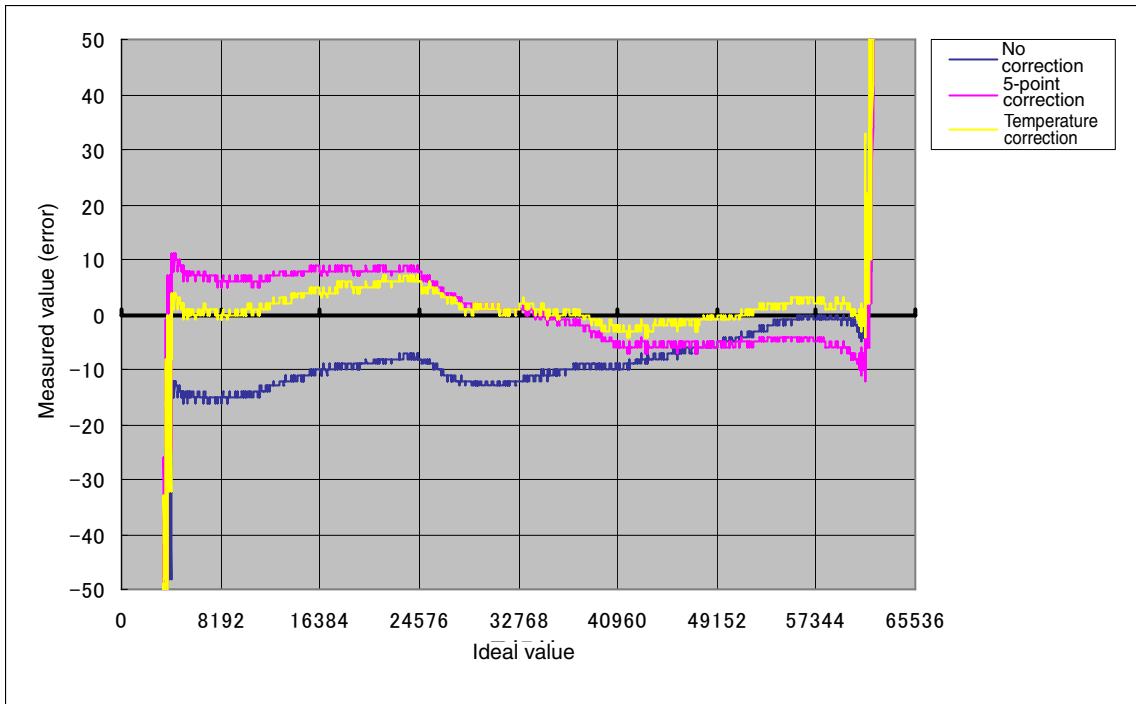
6.1 5-Point Correction

An example of 5-point correction is shown below.



6.2 Temperature Correction

An example of temperature correction performed at 0°C, based on the values measured at 25°C, is shown below.



CHAPTER 7 RELATED DOCUMENTS

Document Name		Japanese/English
78K0/LE3 User's Manual		PDF
78K0/LF3 User's Manual		PDF
78K/0 Series Instructions User's Manual		PDF
RA78K0 Assembler Package User's Manual	Language	PDF
	Operation	PDF
CC78K0 C Compiler User's Manual	Language	PDF
	Operation	PDF
PM+ Project Manager User's Manual		PDF

APPENDIX A PROGRAM LIST

The 78K0/LF3 microcontroller source program is shown below as a program list example.

● main.c (C language version)

```
/******
```

```
    NEC Electronics    78K0/Lx3 Series
```

```
*****
```

```
    78K0/LF3 Series    Sample program
```

```
*****
```

```
    16-bit  $\Delta\Sigma$ -type ADC accuracy correction AN development
```

```
*****
```

```
<<History>>
```

```
    2008.1.--          Release
```

```
*****
```

```
<<Overview>>
```

In this sample program, the 16-bit $\Delta\Sigma$ -type A/D converter is used to A/D convert analog inputs and A/D values are corrected. 5-point correction and temperature correction are performed for the measured A/D values.

5-point correction is performed by deriving an approximation function of an A/D conversion I/O characteristic from five units of 5-point correction data measured according to the voltage used.

Temperature correction corrects errors in the 5-point correction result due to temperature changes, using the acquired temperature correction data and 5-point correction data.

This sample program consists of the five modes of conversion A1, conversion A2, conversion B, correction A, and correction B.

```
<<Conversion A1 mode>>
```

Voltages input from other than the DAC are A/D converted once every time the progress key is pressed and the A/D values are corrected.

```
<<Conversion A2 mode>>
```

By pressing the progress key, voltages input from other than the DAC are A/D converted until the stop key is pressed and the A/D values are corrected.

```
<<Conversion B mode>>
```

The conversion start and end values are read, an output voltage is set to the DAC, the analog input from the DAC is A/D converted, and the A/D values are corrected. This processing is successively performed until the setting value reaches the end value while incrementing the output voltage to be set to the DAC from the start value.

```
<<Correction A mode>>
```

Voltages input from other than the DAC are A/D converted once every time the progress key is pressed. This processing is performed five times and the five units of correction data to be used for 5-point correction operation are acquired.

<<Correction B mode>>

The value derived from the AVDD voltage is set to the DAC and the analog input from the DAC is A/D converted. This processing is performed five times and the five units of correction data to be used for 5-point correction operation are acquired.

The conversion A1, conversion A2, and correction A modes are used to A/D convert analog inputs from other than the DAC. The conversion B and correction B modes are used to A/D convert the analog input from the DAC. Delete unnecessary processing according to the operation environment.

```

*****/
#pragma SFR                                /* SFR names can be described at the C source
level */
#pragma DI                                  /* DI instructions can be described at the C
source level */
#pragma EI                                  /* EI instructions can be described at the C
source level */

/*=====

        Define the ports

=====*/
/* Key input ports */
#define P_KEY0  P15      /* Key (conversion A1, conversion A2, conversion B, correction
A) input port */
#define P_KEY1  P14      /* Key (correction B, progress, stop) input port */

/* Software serial interface */
#define P_CS0   P4.4     /* Chip select 0: For DAC0 */
#define P_CS1   P4.3     /* Chip select 1: For DAC1 */
#define P_SCK   P1.1     /* Clock: Common to DAC1 and DAC0*/
#define P_SDA   P1.3     /* Data: Common to DAC1 and DAC0 */

/* Parallel-to-serial input for HD74HC165 communication */
#define P_CLK   P11.0    /* Clock */
#define P_DAT   P10.3    /* Data */
#define P_LAT   P10.2    /* Latch */

/* Error display */
#define P_NOERR P4.0

/*=====

        Define the structure

```

```

=====*/
struct st_Result {          /* Structure storing processing result */
    unsigned short ushDAC;          /* DAC setting values */
    unsigned short ushAD;          /* A/D conversion result */
    unsigned short ushCorrect;     /* 5-point correction result */
    unsigned short ushHeatCorr;    /* Temperature correction result */
    unsigned char bDAC_ready       :1; /* DAC setting value preparation flag
*/
    unsigned char bAD_ready        :1; /* A/D conversion result preparation
flag */
    unsigned char bCorrect_ready   :1; /* 5-point correction result
preparation flag */
    unsigned char bHeatCorr_ready  :1; /* Temperature correction result
preparation flag */
};

```

```

/*=====
Function prototype declaration

```

```

=====*/
static void fn_Init(void);
static void fn_Init_st_Result(struct st_Result *result);
static void fn_KeyScan(void);
static void fn_KeyEvent(void);
static void fn_Control_ConvertA1(void);
static void fn_Control_ConvertA2(void);
static void fn_Control_ConvertB(void);
static void fn_Control_CorrectA(void);
static void fn_Control_CorrectB(void);
static void fn_ImportStartEnd(void);
static void fn_SetDAC(unsigned short DAC1_value);
static void fn_SSI0_16bit(unsigned short value);
static unsigned short fn_GetAD(void);
static unsigned short fn_CorrectADresult( unsigned short ADresult );
static unsigned short fn_HeatCorrect(unsigned short ushNowCode1, unsigned short
ushCode);
static void fn_SetSendData(struct st_Result s_Result, unsigned char *p_ucTxBuffer);
static void fn_ADResultOut(unsigned char ucTxBuffer[16]);

```

```

/*=====
Define the ROM

```

```

=====*/
/* For  $\Delta\Sigma$ -type A/D converter initialization */
#define ADD0TCR0          (*(volatile unsigned char*) 0x0FA26)
#define ADD0TCR1          (*(volatile unsigned char*) 0x0FA27)

```

APPENDIX A PROGRAM LIST

```

#define ADD0TCR2          (*(volatile unsigned char*) 0x0FA28)
#define ADD0TCR3          (*(volatile unsigned char*) 0x0FA29)

/*=====

        Define the RAM

=====*/
#define ANA_AVDD          5          /* AVDD = 5V */
#define ANA_REFP          ANA_AVDD  /* REFP = AVDD */
#define ANA_MIN           0x0000    /* Minimum digital voltage value */
#define ANA_MAX           0xFFFF    /* Maximum digital voltage value */
#define ANA_THRESHOLD     2.84375   /* Threshold when ana2 > ana4 */

#define CONVERT_B_STEP    1          /* Perform successive conversion in 1H
intervals when converting from start value to end value */

#define TXDATA_SIZE       21        /* Display data size */
/* Display contents: DAC setting values (four digits) SP A/D conversion result (four
digits) SP 5-point correction result (four digits) SP temperature correction result
(four digits) \r\n */

#define READY_NO          0          /* No data */
#define READY_OK          1          /* Data is present */

#define CKEYCHAT          3          /* Key scan chattering removal count */
static unsigned char ucKeyCode;    /* Key code */
#define KYCODE_OFF        0          /* No keys are pressed */
#define KYCODE_CONVERT_A1 1          /* Conversion A1 key is pressed */
#define KYCODE_CONVERT_A2 2          /* Conversion A2 key is pressed */
#define KYCODE_CONVERT_B  3          /* Conversion B key is pressed */
#define KYCODE_CORRECT_A  4          /* Correction A key is pressed */
#define KYCODE_CORRECT_B  5          /* Correction B key is pressed */
#define KYCODE_GO         6          /* Progress key is pressed */
#define KYCODE_STOP       7          /* Stop key is pressed */
#define KYCODE_INVALID    9          /* Multiple keys are pressed (invalid)
*/

static unsigned char ucSystemMode; /* System status */
#define MODE_STANDBY      0          /* Standby status/Waiting for key input
*/
#define MODE_CONVERT_A1  1          /* Single conversion mode (voltage
input from other than DAC) */
#define MODE_CONVERT_A2  2          /* Successive conversion mode (voltage
input from other than DAC)*/
#define MODE_CONVERT_B   3          /* Successive conversion mode (voltage
input from DAC)*/
#define MODE_CORRECT_A   4          /* 5-point correction mode (voltage
input from other than DAC)*/

```

```

#define MODE_CORRECT_B          5          /* 5-point correction mode (voltage
input from DAC) */

static unsigned char ucState;          /* A/D conversion/correction execution
status */
#define STATE_STANDBY          0          /* During standby */
#define STATE_WAIT_GO          1          /* Waiting for progress key */
#define STATE_EXEC              2          /* During execution */
#define STATE_STOP              3          /* Stopped/End of processing */

#define CMP00_AFTER_DAC        (500/100)  /* 8-bit TMH1 count: Waiting for
circuit to stabilize after setting DAC */

static unsigned char uc5CodeReady;     /* 5-point correction data measurement
completion status */
#define CODE5_NOT_COMP          0          /* Measuring 5-point correction data is
not completed */
#define CODE5_COMPLETE         1          /* Measuring 5-point correction data is
completed */

static float fAna[5];                  /* 5-point correction output operation
coefficient */
static unsigned short ushCode[5];      /* 5-point correction output operation
coefficient */
static unsigned short ushCorrCode1;    /* Temperature correction output
operation coefficient code1 correction result */
static unsigned short ushCorrCode3;    /* Temperature correction output
operation coefficient code3 correction result */

#define JDG_HEATCORR_NO        0          /* Temperature correction is not
required */
#define JDG_HEATCORR_EX        1          /* Temperature correction is required
*/
#define JDG_NOT_YET            2          /* Necessity of temperature correction
is not identified */

static unsigned short ushStart;         /* A/D conversion start value */
static unsigned short ushEnd;          /* A/D conversion end value */
static unsigned char ucImportState;     /* A/D conversion start/end value
retrieval status */
#define IMPORT_NO              0          /* Retrieving A/D conversion start/end
value is not completed */
#define IMPORT_OK              1          /* Retrieving A/D conversion start/end
value is completed */

```

/******

Initialization processing after reset release

*****/

```

void hdwinit(void)
{

    unsigned short i;          /* Work area */

    DI();                      /* Disable interrupts */
/*-----
    Set the ROM/RAM size
-----*/

    Note that the setting values differ depending on the model.
    Enable setting of the model used. (uPD78F0495 by default)
-----*/

    /* Settings when uPD78F0461 or uPD78F0491 is used */
    /*IMS = 0x04;              /* Set ROM size */
    /*IXS = 0x0C;              /* Set internal expansion RAM size */

    /* Settings when uPD78F0462 or uPD78F0492 is used */
    /*IMS = 0xC6;              /* Set ROM size */
    /*IXS = 0x0C;              /* Set internal expansion RAM size */

    /* Settings when uPD78F0463 or uPD78F0493 is used */
    /*IMS = 0xC8;              /* Set ROM size */
    /*IXS = 0x0C;              /* Set internal expansion RAM size */

    /* Settings when uPD78F0464 or uPD78F0494 is used */
    /*IMS = 0xCC;              /* Set ROM size */
    /*IXS = 0x0A;              /* Set internal expansion RAM size */

    /* Settings when uPD78F0465 or uPD78F0495 is used */
    IMS = 0xCF;                /* Set ROM size */
    IXS = 0x0A;                /* Set internal expansion RAM size */

/*-----
    Set the clock frequency
-----*/

    The clock frequency is set to enable operation using the high-speed system
    clock.
-----*/

    OSCCTL = 0b01000000;       /* Clock operation mode */
    /* ||| |++++----- <Fix to 0> */
    /* ||| |+----- OSCSELS: Input port */
    /* || |+----- <Fix to 0> */
    /* ++----- EXCLK/OSCSEL: X1 oscillation mode */

    MOC = 0;                   /* X1 oscillator operation */

    while (OSTC.0 == 0)        /* Wait for oscillation stabilization time of
X1 oscillator to elapse */

    MCM = 0b00000101;         /* Select supply clock */

```



```

/*|||||+|+----- XSEL/MCM0: */
/*||||| |           Main system clock (fXP) = fXH */
/*||||| |           Peripheral hardware clock (fPRS) = fXH */
/*||||| +----- MCS: Read Only */
/*+++++----- <Fix to 0> */

PCC = 0b0000000;          /* Select CPU clock (fCPU) */
/*|||+|+++----- CSS/PCC2/PCC1/PCC0: */
/*||| |             CPU clock (fCPU) = fXP */
/*||| +----- <Fix to 0> */
/*||+----- CLS: Main system clock */
/*++----- <Fix to 0> */

RCM = 0b0000011;          /* Select CPU clock (fCPU) */
/*|||||+|+----- LSRSTOP: Stop internal low-speed oscillator
*/

/*|||||+|+----- RSTOP: Stop internal high-speed oscillator */
/*|++++----- <Fix to 0> */
/*+----- RSTS: Read Only */

/*-----
Set UART6
-----*/

CKSR6 = 0b00000000;      /* Select UART6 base clock */
/*|||||++++----- TPS63-60: Base clock (fXCLK6) = fPRS */
/*++++----- <Fix to 0> */

/* Set baud rate clock division value */
BRGC6 = 43;              /* Baud rate: 115200 bps ← 116279 bps(ERR: 0.94%) */

ASIM6 = 0b01000101;      /* Select UART6 operation mode */
/*|||||+|+----- ISRM6: Interrupt INTSR6 when reception error
occurs */

/*|||||+|+----- SL6: Number of stop bits = 1 */
/*|||||+|+----- CL6: Data length = 8 */
/*||+|+----- PS61-60: No parity */
/*||+----- RXE6: Disable receive operation */
/*|+----- TXE6: Enable transmit operation */
/*+----- POWER6: Disable internal operating clock
operation */

ASICL6 =0b00010110;      /* Select start bit and TxD6 output reversal */
/*|||||+|+----- TXDLV6: Normal TxD6 output */
/*|||||+|+----- DIR6: Start bit LSB */
/*||+|+----- SBL62-60: Unused */
/*||+----- SBTT6: Unused */
/*|+----- SBRT6: Read Only */
/*+----- SBRF6: Unused */

```

```

PF1 = 0b00000000;          /* Set to use P16 */
/*+|++|+++----- <Fix to 000000> */
/* |  +----- PF13: Unused */
/* +----- PF16: Unused (when using K0/LF3) */
/* +----- <Fix to 0> (when using K0/LE3) */

ISC = 0b00001000;          /* Control input switching (when using LF3) */
/*|||||+----- ISC0: Unused */
/*|||||+----- ISC1: Use TI000 input as it is (normal
operation) */
/*|||||+----- ISC2: Unused */
/*|||||+----- ISC3: Enable RxD6/P113 input */
/*|||++----- ISC5-4: TxD6 = P112, RxD6 = P113 */
/*++----- <Fix to 0> */

POWER6 = 1;                /* Enable internal operating clock operation */

/*-----
Set the transmission
-----*/
PM11.2 = 0;                 /* P112 = TxD6 */
P11.2 = 1;                  /* P112 = Hi */

/*-----
Set the reception
-----*/
PM11.3 = 1;                 /* P113 = RxD6 */
PU11.3 = 1;                 /* Connect internal pull-up, because receive
operation is not performed */

/*-----
Set the 100 us interval timer
-----
Set the wait time measurement timer (the interval time can be set in 100 us
units).
- Circuit stabilization wait time after setting the DAC (500 us)
-----*/
TMHMD0 = 0b01000000;        /* Timer clock selection register */
/*|||||+----- TOEN1: Disable timer output */
/*|||||+----- TOLEV0: Timer output level = Unused */
/*|||++----- TMMD01/00: Timer operation = Interval */
/*|+++----- CKS02/01/00: Count clock fPRS/2^10 (9.77 kHz
when fPRS = 10 MHz) */
/* +----- TMHE0: Disable timer operation (enable by
setting CMP00 when using timer) */

TMIFH0 = 0;                 /* Clear interrupt request */
TMMKH0 = 1;                 /* Disable interrupts */

/*-----

```

```

Set the 10 ms interval
-----
Use TMH1 and create a 10 ms interval
-----*/
TMHMD1 = 0b01000000;          /* Timer clock selection register */
/* |||||+----- TOEN1: Disable timer output */
/* |||||+----- TOLEV1: Timer output level = Unused */
/* |||++----- TMMD11/10: Timer operation = Interval */
/* |+++----- CKS12/11/10: Count clock fPRS/2^12 (2.44 kHz
when fPRS = 10 MHz) */
/* +----- TMHE1: Disable timer operation (enable after
setting timer) */
CMP01 = 24-1;                /* 10 ms interval: (fPRS/2^12)*0.01[sec]=24.4
*/

TMHE1 = 1;                   /* Start timer operation */
TMIFH1 = 0;                  /* Clear interrupt request */
TMMKH1 = 1;                  /* Disable interrupts */

/*-----
Set the DAC setting software serial interface
-----*/
PM1 = 0b00000000;           /* Set P1 I/O *//* Exclude the comment only
when operating with the K0/LF3 */
/* |||||+----- PM10: Unused (0) */
/* |||||+----- PM11: Use output (0) as SCK */
/* ||||+----- PM12: Unused (0) */
/* |||+----- PM13: Use output (0) as SDA */
/* ++++----- PM17/PM16/PM15/PM14: Unused (0) */
P1 = 0b00000010;           /* Initial P1 value *//* Exclude the comment
only when operating with the K0/LF3*/
/* |||||+----- P10: Unused (0) */
/* |||||+----- P11: Hi(1) */
/* ||||+----- P12: Unused (0) */
/* |||+----- P13: Lo(0) */
/* ++++----- P17/P16/P15/P14: Unused (0) */
/* PM1 = 0b11100000; */ /* Set P1 I/O *//* Exclude the comment only
when operating with the K0/LE3 */
/* |||||+----- PM10: Unused (0) */
/* |||||+----- PM11: Use output (0) as SCK */
/* ||||+----- PM12: Unused (0) */
/* |||+----- PM13: Use output (0) as SDA */
/* ||+----- PM14: Unused (0) */
/* ++++----- <Fix to 111> */
/* P1 = 0b00000010; */ /* Initial P1 value *//* Exclude the comment
only when operating with the K0/LE3 */
/* |||||+----- P10: Unused (0) */
/* |||||+----- P11: Hi(1) */
/* ||||+----- P12: Unused (0) */
/* |||+----- P13: Lo(0) */

```

```

/*|||+----- P14: Unused (0) */
/*+++----- <Fix to 000> */
PM4 = 0b00000000; /* Set P4 I/O *//* Exclude the comment only
when operating with the K0/LF3 */
/*|||+++----- PM42/PM41/PM40: Unused (0) (Set P40 for error
output afterward) */
/*|||+----- PM43: Use output (0) as CS1 */
/*||+----- PM44: Use output (0) as CS0 */
/*+++----- PM47/PM46/PM45: Unused (0) */
P4 = 0b00011000; /* Initial P4 value *//* Exclude the comment
only when operating with the K0/LF3 */
/*|||+++----- PM42/PM41/PM40: Unused (0) (Set P40 for error
output afterward) */
/*|||+----- PM43: Hi(1) */
/*||+----- PM44: Hi(1) */
/*+++----- PM47/PM46/PM45: Unused (0) */
/* PM4 = 0b11100000; */ /* Set P4 I/O *//* Exclude the comment only
when operating with the K0/LE3 */
/*|||+++----- PM42/PM41/PM40: Unused (0) (Set P40 for error
output afterward) */
/*|||+----- PM43: Use output (0) as CS1 */
/*||+----- PM44: Use output (0) as CS0 */
/*+++----- <Fix to 111> */
/* P4 = 0b00011000; */ /* Initial P4 value *//* Exclude the comment
only when operating with the K0/LE3 */
/*|||+++----- PM42/PM41/PM40: Unused (0) (Set P40 for error
output afterward) */
/*|||+----- PM43: Hi(1) */
/*||+----- PM44: Hi(1) */
/*+++----- <Fix to 000> */

/*-----
16-bit  $\Delta\Sigma$ -type A/D converter
-----*/
ADPC0 = 0b00000000; /* A/D port configuration register 0 */
/*|||+++----- ADPC03/ADPC02/ADPC01/ADPC00: Set P20 to P27
as analog inputs ( $\Delta\Sigma$ -type) */
/*++++----- <Fix to 0000> */

PM2 = 0b11111111; /* Set PM2 I/O */
/*|||+++----- PM20: Use input (1) as DS0- */
/*|||+++----- PM21: Use input (1) as DS0+ */
/*|||+++----- PM22: Use input (1) as DS1- */
/*|||+++----- PM23: Use input (1) as DS1+ */
/*||+----- PM24: Use input (1) as DS2- */
/*|+----- PM25: Use input (1) as DS2+ */
/*+----- PM26: Use input (1) as REF- */
/*----- PM27: Use input (1) as REF+ */
P2 = 0b00000000; /* Set initial P2 value */
/*++++----- P27/P26/P25/P24/P23/P22/P21/P20: Lo(O) */

```

```

        ADDPON = 1;                                /* Turn on A/D converter power supply */

        ADDCTL1=0b01100111;                       /* 16-bit ΔΣ-type A/D converter control
register 1 */
        /*|||||+++----- ADDN2/ADDN1/ADDN0: Specify sampling count
(resolution) to 65,536 times (16 bits) */
        /*|||||+++----- <Fix to 00> */
        /*|+----- ADDTS: Set serial mode */
        /*+++----- ADDFS1/ADDF0: Select fPRS/8 as sampling clock
*/

        ADDCTL0=0b10110000;                       /* 16-bit ΔΣ-type A/D converter control
register 0 */
        /*|||||+++----- ADDS1/ADDS0: Specify DS0+/DS0- as analog input
*/
        /*|||||+++----- <Fix to 00> */
        /*|+----- AINMOD: Specify differential input as input
mode */
        /*|+----- HAC: Set high-accuracy mode to on */
        /*+----- ADDCE: Stop conversion */
        /*+++----- ADDPON: Turn on A/D converter power supply */

        /* Initialize SFR */
        ADD0TCR0 = 0b10000000;
        ADD0TCR0 = 0b11000000;
        ADD0TCR1 = 0b00100000;
        ADD0TCR2 = 0b01100000;
        ADD0TCR3 = 0b00010011;

        for( i = 20; i < 0; i--);                /* Leave interval of at least 5 us between
turning on A/D converter power supply and enabling operation */
        ADDCE = 1;                                /* Enable conversion operation */
        DSADIF = 0;                                /* Clear A/D conversion completion interrupt
request flag */

/*-----
        Set key scan
-----

        Set the ports for key scan
        Allocate ports
        P150 : Conversion A1    P140 : Correction B
        P151 : Conversion A2    P141 : Progress key
        P152 : Conversion B    P142 : Stop
        P153 : Correction A

-----*/
        PM15 = 0b11111111;                        /* Set P15 I/O */
        /*|||||++++----- PM153/PM152/PM151/PM150: Input (1) */
        /*++++----- <Fix to 0000> */
        P15 = 0b00000000;                        /* Initial P15 value */

```

```

/* ||| |++++----- P153/P152/P151/P150: Lo(O) */
/*++++----- <Fix to 0000> */
PU15 = 0b00001111;          /* Set P15 internal pull-up connection */
/* ||| |++++----- PU153/PU152/PU151/PU150: Connect (1) internal
pull-up resistor */
/*++++----- <Fix to 0000> */

PM14 = 0b11111111;         /* Set P14 I/O */
/* ||| |++++----- PM143/PM142/PM141/PM140: Input (1) */
/*++++----- <Fix to 1111> */
P14 = 0b00001111;         /* Set initial P14 value */
/* ||| |++++----- P143/P142/P141/P140: Hi(1) */
/*++++----- <Fix to 0000> */
PU14 = 0b00001111;         /* Set P14 internal pull-up connection */
/* ||| |++++----- PU143/PU142/PU141/PU140: Connect (1) internal
pull-up resistor */
/*++++----- <Fix to 0000> */

/* Set interrupt using STOP key */
PM12 = 0b11111111;         /* Set P12 I/O */
/* ||| | | | | | +----- P120: Input (1) INTPO */
/*++++----- <Fix to 0000000> */
P12 = 0b00000001;         /* Set initial PM12 value */
/* ||| | | | | | +----- PM120: Lo(0) */
/*++++----- <Fix to 0000000> (when using K0/LF3) */
/*++++----- Not settable (0) (when using K0/LE3) */
PU12 = 0b00000001;         /* Set P12 internal pull-up connection */
/* ||| | | | | | +----- PU120: Connect (1) internal pull-up resistor
*/
/*++++----- <Fix to 0000000> */

EGP = 0b00000000;         /* External interrupt rising edge enable
register */
/* ||| | | | | | +----- EGP0: Enable falling-edge detection according
to EGN0 */
/* | | | | | | | | +----- EGP5/EGP4/EGP3/EGP2/EGP1: Unused */
/*++----- <Fix to 00> */
EGN = 0b00000001;         /* External interrupt falling edge enable
register */
/* ||| | | | | | +----- EGN0: Enable falling-edge detection according
to EGP0*/
/* | | | | | | | | +----- EGN5/EGN4/EGN3/EGN2/EGN1: Unused */
/*++----- <Fix to 00> */

PMK0 = 1;                  /* Disable interrupt servicing */
PIF0 = 0;                  /* Clear interrupt request */

/*-----
Set the parallel-to-serial input for HD74HC165 communication
-----*/

```

```

P10 = 0b00000000;          /* Set initial P10 value */
/* |||||++----- P101/P100: Lo(O) */
/* |||||++----- P103/P102: Lo(O) */
/*++++----- <Fix to 0000> */
PM10 = 0b11111000;        /* Set P10 I/O */
/* |||||++----- PM101/PM100: Unused (0) */
/* |||||+----- PM102: Output (0) = LAT */
/* |||||+----- PM103: Input (1) = DAT */
/*++++----- <Fix to 1111> */
PU10 = 0b00001000;        /* Set P10 internal pull-up connection */
/* |||||++----- PU101/PU100: Unused (0) */
/* |||||+----- PU102: Unused (0) */
/* |||||+----- PU103: Connect (1) internal pull-up resistor
*/

/*++++----- <Fix to 0000> */

P11.0 = 0;                /* P110: Lo(O) */
PM11.0 = 0;               /* PM110: Output (0) = CLK */

/*-----
Set the error output port
-----*/
PM4.0 = 0;                /* PM40: Use output (0) for error output */
P4.0 = 1;                 /* P40: Hi(1) */

/*-----
Initialize unused ports
-----*/
/* P3 */
PM3 = 0b11100000;         /* Set P3 I/O */ /* Exclude the comment only
when operating with the K0/LF3 */
/* |||++++----- PM34/PM33/PM32/PM31/PM30: Unused (0) */
/*++++----- <Fix to 111> */
P3 = 0b00000000;         /* Set initial P3 value */ /* Exclude the
comment only when operating with the K0/LF3 */
/* |||++++----- P34/P33/P32/P31/P30: Lo(O) */
/*++++----- <Fix to 000> */
/* PM3 = 0b11100001; */ /* Set P3 I/O */ /* Exclude the comment only
when operating with the K0/LE3 */
/* |||||+----- <Fix to 1> */
/* |||++++----- PM34/PM33/PM32/PM31: Unused (0) */
/*++++----- <Fix to 111> */
/* P3 = 0b00000000; */ /* Set initial P3 value */ /* Exclude the
comment only when operating with the K0/LE3 */
/* |||||+----- <Fix to 0> */
/* |||++++----- P34/P33/P32/P31: Lo(O) */
/*++++----- <Fix to 000> */

/* P8 */

```

```

PM8 = 0b11110000;          /* Set P8 I/O */
/* ||| |++++----- PM83/PM82/PM81/PM80: Unused (0) */
/*++++----- <Fix to 1111> */
P8 = 0b00000000;          /* Set initial P8 value */
/* ||| |++++----- P83/P82/P81/P80: Lo(0) */
/*++++----- <Fix to 0000> */

/* P9 */
PM9 = 0b11110000;          /* Set P9 I/O */ /* Exclude the comment only
when operating with the K0/LF3 */
/* ||| |++++----- PM93/PM92/PM91/PM90: Unused (0) */
/*++++----- <Fix to 1111> */
P9 = 0b00000000;          /* Set initial P9 value */ /* Exclude the
comment only when operating with the K0/LF3 */
/* ||| |++++----- P93/P92/P91/P90: Lo(0) */
/*++++----- <Fix to 0000> */

/* P11 */
PM11.1 = 0;                /* PM111: Unused (0) */
P11.1 = 0;                 /* P111: Lo(0) */

/* P13 */
PM13 = 0b11110000;          /* Set P13 I/O */ /* Exclude the comment only
when operating with the K0/LF3 */
/* ||| |++++----- PM130/PM133/PM132/PM131: Unused (0) */
/*++++----- <Fix to 1111> */
P13 = 0b00000001;          /* Set initial P13 value */ /* Exclude the
comment only when operating with the K0/LF3 */
/* ||| |++++----- P133/P132/P131/P130: Lo(0) */
/*++++----- <Fix to 0000> */

PFALL = 0b00000000;          /* Set output of port shared with segment */
/* ++|++|+----- PF15ALL/PF14ALL/PF11ALL/PF10ALL/PF08ALL: Use
for other than segment output */
/* | +-------- PF13ALL/PF09ALL: Use for other than segment
output (when using K0/LF3) */
/* | +-------- <Fix to 00> (when using K0/LE3) */
/*+----- <Fix to 0> */

EI();                       /* Enable interrupts */
}

/*****

Variable initialization processing

-----

*****/
static void fn_Init(void)

```



```

{
    unsigned char i;                /* Initialization work variable */
    float temp;

    /* Key code */
    ucKeyCode = KYCODE_OFF;        /* Key off */

    /* System status */
    ucSystemMode = MODE_STANDBY;   /* Standby */

    /* A/D conversion/correction execution status */
    ucState = STATE_STANDBY;       /* During standby */

    /* 5-point correction data measurement completion status */
    uc5CodeReady = CODE5_NOT_COMP; /* Measuring 5-point correction data is
not completed */

    /* Set voltage values of correction output operation coefficients ana1 to ana5
using digital values */
    fAna[0] = ((0.1) * ANA_REFP)*(ANA_MAX/ANA_REFP);
    fAna[1] = (ANA_AVDD - (0.82*ANA_AVDD) + 0.91)*(ANA_MAX/ANA_REFP);
    fAna[2] = (0.5 * ANA_REFP)*(ANA_MAX/ANA_REFP);
    fAna[3] = (0.82 * ANA_AVDD - 0.91)*(ANA_MAX/ANA_REFP);
    fAna[4] = (0.9 * ANA_REFP)*(ANA_MAX/ANA_REFP);

    if( ANA_AVDD < ANA_THRESHOLD ){          /* If AVDD = REFP < 2.84375 V */
        temp = fAna[1];                      /* Replace value because ana2 > ana4 */
        fAna[1] = fAna[3];
        fAna[3] = temp;
    }

    /* Correction output operation coefficients code1 to code5 */
    for( i = 0; i < 5; i++){
        ushCode[i] = ANA_MIN;
    }

    /* Retrieve A/D conversion start/end value */
    ushStart = ANA_MIN;                    /* Initialize A/D conversion start
value at minimum value */
    ushEnd = ANA_MAX;                      /* Clear A/D conversion end value */
    ucImportState = IMPORT_NO;             /* Retrieving A/D conversion start/end
value is not completed */
}

/*****

    st_Result type variable initialization processing

*****/
static void fn_Init_st_Result(struct st_Result *result)

```

```

{
    (*result).ushDAC = ANA_MIN;          /* Initialize DAC setting
values */
    (*result).ushAD = ANA_MIN;          /* Initialize A/D conversion
result */
    (*result).ushCorrect = ANA_MIN;     /* Initialize 5-point
correction result */
    (*result).ushHeatCorr = ANA_MIN;    /* Initialize temperature
correction result */
    (*result).bDAC_ready = READY_NO;    /* No DAC setting value data */
    (*result).bAD_ready = READY_NO;     /* No A/D conversion result
data */
    (*result).bCorrect_ready = READY_NO; /* No 5-point correction result
data */
    (*result).bHeatCorr_ready = READY_NO; /* No temperature correction
result data */
}

/*****

Main loop

*****/

void main(void)
{
    hdwinit();          /* Initialize hardware */
    fn_Init();          /* Initialize variables or the like */

    while(1)
    {
        if( TMIFH1 )          /* Processing every 10 ms */
        {
            TMIFH1 = 0;
            fn_KeyScan();     /* Retrieve key */
        }
        fn_KeyEvent();       /* Event processing using key code */
        switch( ucSystemMode ){ /* Branch using current mode */
        case MODE_CONVERT_A1: /* Conversion A1 mode control
processing */
            fn_Control_ConvertA1();
            break;
        case MODE_CONVERT_A2: /* Conversion A2 mode control
processing */
            fn_Control_ConvertA2();
            break;
        case MODE_CONVERT_B: /* Conversion B mode control processing
*/
            fn_Control_ConvertB();
            break;

```

```

        case MODE_CORRECT_A:          /* Correction A mode control processing
*/
            fn_Control_CorrectA();
            break;
        case MODE_CORRECT_B:          /* Correction B mode control processing
*/
            fn_Control_CorrectB();
            break;
        default:
            break;
    }
}

/*****

Key scan processing

-----

The key status is retrieved and chattering is removed.
Key scan cycle: 10 ms
*****/
static void fn_KeyScan(void)
{
    static unsigned char LastKeyState = 0;
    static unsigned char ChatCounter = 0;
    unsigned char NewKeyState;

    NewKeyState = ( P_KEY1 & 0b00001111 ) << 4 ;          /* Retrieve current key
status */
    NewKeyState = NewKeyState | ( P_KEY0 & 0b00001111 );

    if( LastKeyState == NewKeyState ){                      /* When key status does
not change */
        ChatCounter--;
        if( ChatCounter == 0 ){
            ChatCounter = CKEYCHAT;                          /* Initialize
chattering counter */

            /* Set key code according to key status */
            if( NewKeyState == 0b11111111 ){                  /* All keys are off */
                ucKeyCode = KYCODE_OFF;                       /* Key code: Key off */
            } else if( NewKeyState == 0b11111110 ){           /* Only conversion A1
key is on */
                ucKeyCode = KYCODE_CONVERT_A1;               /* Key code: Conversion
A1 */
            } else if( NewKeyState == 0b11111101 ){         /* Only conversion A2
key is on */
                ucKeyCode = KYCODE_CONVERT_A2;               /* Key code: Conversion
A2 */
            }
        }
    }
}

```

```

        } else if( NewKeyState == 0b11111011 ){ /* Only conversion B
key is on */
                                ucKeyCode = KYCODE_CONVERT_B; /* Key code: Conversion
B */
        } else if( NewKeyState == 0b11110111 ){ /* Only correction A
key is on */
                                ucKeyCode = KYCODE_CORRECT_A; /* Key code: Correction
A */
        } else if( NewKeyState == 0b11101111 ){ /* Only correction B
key is on */
                                ucKeyCode = KYCODE_CORRECT_B; /* Key code: Correction
B */
        } else if( NewKeyState == 0b11011111 ){ /* Only progress key is
on */
                                ucKeyCode = KYCODE_GO; /* Key code: Progress
*/
        } else if( NewKeyState == 0b10111111 ){ /* Only stop key is on
*/
                                ucKeyCode = KYCODE_STOP; /* Key code: Stop */
        } else { /* Multiple keys are on
*/
                                ucKeyCode = KYCODE_INVALID; /* Key code: Keys are
invalid */
        }
    }
}
else{ /* When key status has changed */
    ChatCounter = CKEYCHAT; /* Initialize chattering counter */
}
LastKeyState = NewKeyState; /* Save current key status */
}

```

/******

Key event processing

 The mode or conversion execution status is transitioned by using keys.
 *****/

```

static void fn_KeyEvent(void)
{
    static unsigned char ucLastKeyCode;

    if( ucKeyCode != ucLastKeyCode ){
        switch ( ucKeyCode ){
            case KYCODE_CONVERT_A1: /* If conversion A1 key is pressed */
                if( ucState == STATE_STANDBY ){ /* If
conversion/correction is not performed */

```

```

        ucSystemMode = MODE_CONVERT_A1; /* Go to single
conversion mode (voltage input from other than DAC) */
        ucState = STATE_WAIT_GO;      /* Set progress key
wait state */
    }
    break;

    case KYCODE_CONVERT_A2:          /* If conversion A2 key is pressed */
        if( ucState == STATE_STANDBY ){          /* If
conversion/correction is not performed */
            ucSystemMode = MODE_CONVERT_A2; /* Go to successive
conversion mode (voltage input from other than DAC) */
            ucState = STATE_WAIT_GO;          /* Set progress key
wait state */
        }
        break;

    case KYCODE_CONVERT_B:          /* If conversion B key is pressed */
        if( ucState == STATE_STANDBY ){          /* If
conversion/correction is not performed */
            ucSystemMode = MODE_CONVERT_B; /* Go to successive
conversion mode (voltage input from DAC) */
            ucState = STATE_WAIT_GO;          /* Set progress key
wait state */
        }
        break;

    case KYCODE_CORRECT_A:          /* If correction A key is pressed */
        if( ucState == STATE_STANDBY ){          /* If
conversion/correction is not performed */
            ucSystemMode = MODE_CORRECT_A; /* Go to 5-point
correction mode (voltage input from other than DAC) */
            ucState = STATE_WAIT_GO;          /* Set progress key
wait state */
        }
        break;

    case KYCODE_CORRECT_B:          /* If correction B key is pressed */
        if( ucState == STATE_STANDBY ){          /* If
conversion/correction is not performed */
            ucSystemMode = MODE_CORRECT_B; /* Go to 5-point
correction mode (voltage input from DAC) */
            ucState = STATE_WAIT_GO;          /* Set progress key
wait state */
        }
        break;

    case KYCODE_GO:                  /* If progress key is pressed */
        if( ucState == STATE_WAIT_GO ){          /* If waiting for
progress key */

```

```

                                ucState = STATE_EXEC;           /* Set conversion
execution wait state */
                                }
                                break;

                                case KYCODE_STOP:                /* If stop key is pressed */
                                ucState = STATE_STOP;           /* Go to stop
processing */
                                break;

                                /* case KYCODE_INVALID: */       /* If key is invalid */
                                default:
                                ;                               /* Do nothing */
                                }
                                }
                                if(PIF0){                       /* If stop key is
pressed */
                                ucState = STATE_STOP;           /* Go to stop
processing */
                                PIF0 = 0;                       /* Clear interrupt
request */
                                }
                                ucLastKeyCode = ucKeyCode;      /* Update previous key
code */
                                }

```

A/D conversion: Conversion A1 mode control processing

Single conversion mode (voltage input from other than the DAC) processing is controlled.

* Used only if the DAC is not used for A/D conversion analog input

```

static void fn_Control_ConvertA1(void)
{
    struct st_Result sResult;                               /* Store conversion result */
    static unsigned char ucTxBuffer[TXDATA_SIZE];          /* Display data buffer */
    static unsigned short ushNowCode1;                     /* code1 re-acquired at current
temperature */
    static unsigned char HeatCorrJdg;                       /* Information on necessity of
temperature correction */

    switch( ucState ){ /* Branch according to execution status */
    case STATE_STOP: /* Stop processing */
        fn_Init_st_Result( &sResult );                    /* Initialize
conversion result */
        ucState = STATE_STANDBY;                           /* Set operation status
to "Standby" */

```

```

        ushNowCode1 = ANA_MIN;                                /* Initialize code1 of
current temperature */
        HeatCorrJdg = JDG_NOT_YET;                          /* Necessity of
temperature correction is not identified */
        break;

    case STATE_WAIT_GO:    /* Wait for progress key */
        if(( uc5CodeReady == CODE5_COMPLETE )&&( HeatCorrJdg == JDG_NOT_YET ))
        {
            /* If 5-point correction data has been measured and necessity
of temperature correction is unidentified, re-acquire code1 and perform identification
*/
            ushNowCode1 = fn_GetAD();                          /* Re-acquire
anal A/D conversion result */
            ushNowCode1 = fn_CorrectADresult( ushNowCode1 );/* Correct
current code1 */
            if( ushNowCode1 != ushCorrCode1 )                  /* If
temperature change exists */
                HeatCorrJdg = JDG_HEATCORR_EX;                /* Set to
execute temperature correction */
            else
                HeatCorrJdg = JDG_HEATCORR_NO;                 /* Set so as
not to execute temperature correction */
        }
        break;

    case STATE_EXEC:    /* Execute conversion processing */
        sResult.bDAC_ready = READY_NO;                          /* No DAC setting value
data */
        sResult.ushAD = fn_GetAD();                            /* Acquire A/D
conversion result */
        sResult.bAD_ready = READY_OK;                          /* Preparing A/D
conversion result data is completed */

        if( uc5CodeReady == CODE5_COMPLETE )                  /* If measuring 5-point
correction data is completed */
        {
            sResult.ushCorrect = fn_CorrectADresult( sResult.ushAD );
/* Correct 5-point of A/D conversion result */
            sResult.bCorrect_ready = READY_OK;                 /* Preparing 5-point
correction result data is completed */
            if( HeatCorrJdg == JDG_HEATCORR_EX )              /* If temperature
correction is required */
            {
                sResult.ushHeatCorr = fn_HeatCorrect( ushNowCode1,
sResult.ushCorrect);/* Correct temperature */
                sResult.bHeatCorr_ready = READY_OK;/* Preparing
temperature correction result data is completed */
            }
            else

```

```

        sResult.bHeatCorr_ready = READY_NO; /* No 5-point
correction result data */
    }
    else /* If preparing 5-point
correction data is not completed */
    {
        sResult.bCorrect_ready = READY_NO; /* No 5-point
correction result data */
        sResult.bHeatCorr_ready = READY_NO; /* No 5-point
correction result data */
    }

    fn_SetSendData( sResult, ucTxBuffer ); /* Set conversion
result output */
    fn_ADResultOut( ucTxBuffer ); /* Output data */
    ucState = STATE_WAIT_GO; /* Wait for progress
key and perform next conversion */
    break;

default:
    break;
}
}

```

A/D conversion: Conversion A2 mode control processing

```

-----
    Successive conversion mode (voltage input from other than the DAC) processing
is controlled.
    * Used only if the DAC is not used for A/D conversion analog input.
*****/
static void fn_Control_ConvertA2(void)
{
    struct st_Result sResult; /* Store conversion result */
    static unsigned char ucTxBuffer[TXDATA_SIZE]; /* Display data buffer */
    static unsigned short ushNowCode1; /* code1 re-acquired at current
temperature */
    static unsigned char HeatCorrJdg; /* Information on necessity of
temperature correction */

    switch( ucState ){ /* Branch according to execution status */
    case STATE_STOP: /* Stop processing */
        fn_Init_st_Result( &sResult ); /* Initialize
conversion result */
        ucState = STATE_STANDBY; /* Set operation status
to "Standby" */
        ushNowCode1 = ANA_MIN; /* Initialize code1 of
current temperature */

```



```

        HeatCorrJdg = JDG_NOT_YET;                                /* Necessity of
temperature correction is not identified */
        break;

    case STATE_WAIT_GO:    /* Wait for progress key */
        if(( uc5CodeReady == CODE5_COMPLETE )&&( HeatCorrJdg == JDG_NOT_YET ))
        {
            /* If measuring 5-point correction data is completed and
necessity of temperature correction is unidentified, re-acquire code1 and perform
identification */
                ushNowCode1 = fn_GetAD();                        /* Re-acquire
anal A/D conversion result */
                ushNowCode1 = fn_CorrectADresult( ushNowCode1 );/* Correct
current code1 */
                if( ushNowCode1 != ushCorrCode1 )                /* If
temperature change exist */
                    HeatCorrJdg = JDG_HEATCORR_EX;              /* Set to
execute temperature correction */
                else
                    HeatCorrJdg = JDG_HEATCORR_NO;              /* Set so as
not to execute temperature correction */
        }
        break;

    case STATE_EXEC:    /* Execute conversion processing */
        sResult.bDAC_ready = READY_NO;                            /* No DAC setting value
data */
        sResult.ushAD = fn_GetAD();                               /* Acquire A/D
conversion result */
        sResult.bAD_ready = READY_OK;                            /* Preparing A/D
conversion result data is completed */

        if( uc5CodeReady == CODE5_COMPLETE )                    /* If measuring 5-point
correction data is completed */
        {
            sResult.ushCorrect = fn_CorrectADresult( sResult.ushAD );
/* Correct 5-point of A/D conversion result */
            sResult.bCorrect_ready = READY_OK;                  /* Preparing 5-point
correction result data is completed */
            if( HeatCorrJdg == JDG_HEATCORR_EX )                /* If temperature
correction is required */
            {
                sResult.ushHeatCorr = fn_HeatCorrect( ushNowCode1,
sResult.ushCorrect);/* Correct temperature */
                sResult.bHeatCorr_ready = READY_OK;/* Preparing
temperature correction result data is completed */
            }
            else
                sResult.bHeatCorr_ready = READY_NO;/* No 5-point
correction result data */
        }
    }

```

```

        else                                     /* If measuring 5-point
correction data is not completed */
        {
            sResult.bCorrect_ready = READY_NO;    /* No 5-point
correction result data */
            sResult.bHeatCorr_ready = READY_NO;   /* No 5-point
correction result data */
        }
        fn_SetSendData( sResult, ucTxBuffer );    /* Set conversion
result output */
        fn_ADResultOut( ucTxBuffer );            /* Output data */
        /* Do not change A/D execution status, because conversion is also
performed next time */
        break;

    default:
        break;
}
}

```

A/D conversion: Conversion B mode control processing

 Successive conversion mode (voltage input from the DAC) processing is controlled.

* Used only if the DAC is used for A/D conversion analog input.

```

static void fn_Control_ConvertB(void)
{
    static struct st_Result sResult;             /* Store conversion result */
    static unsigned char ucTxBuffer[TXDATA_SIZE]; /* Display data buffer */
    static unsigned short ushNowCode1;          /* code1 re-acquired at current
temperature */
    static unsigned char HeatCorrJdg;           /* Information on necessity of
temperature correction */
    unsigned char temp;

    switch( ucState ){                          /* Branch according to execution status */
    case STATE_STOP:                             /* Stop processing */
        fn_Init_st_Result( &sResult );          /* Initialize
conversion result */
        ushStart = ANA_MIN;                       /* Clear A/D conversion
start value */
        ushEnd = ANA_MAX;                         /* Clear A/D conversion
end value */
        ucImportState = IMPORT_NO;               /* Clear A/D conversion
start/end value retrieval status */
    }
}

```

```

        ucState = STATE_STANDBY;                /* Set operation status
to "Standby" */
        ushNowCode1 = ANA_MIN;                 /* Initialize code1 of
current temperature */
        HeatCorrJdg = JDG_NOT_YET;            /* Necessity of
temperature correction is not identified */
        break;

        case STATE_WAIT_GO:    /* Wait for progress key */
            if( ucImportState == IMPORT_NO )    /* If A/D conversion
start/end value is not retrieved */
            {
                fn_ImportStartEnd();           /* Retrieve conversion
start/end value */
                sResult.ushDAC = ushStart;     /* Initialize DAC
setting values */
            }
            if(( uc5CodeReady == CODE5_COMPLETE )&&( HeatCorrJdg == JDG_NOT_YET ))
            { /* If 5-point correction data has been measured and necessity
of temperature correction is unidentified, re-acquire code1 and perform identification
*/
                fn_SetDAC((unsigned short)fAna[1-1] ); /* Set anal to
DAC output */
                ushNowCode1 = fn_GetAD();       /* Re-acquire
anal A/D conversion result */
                ushNowCode1 = fn_CorrectADresult( ushNowCode1 ); /* Correct
current code1 */
                if( ushNowCode1 != ushCorrCode1 ) /* If
temperature change exists */
                    HeatCorrJdg = JDG_HEATCORR_EX; /* Set to
execute temperature correction */
                else
                    HeatCorrJdg = JDG_HEATCORR_NO; /* Set so as
not to execute temperature correction */
            }
            break;

        case STATE_EXEC:    /* Execute conversion processing */
            if( ucImportState == IMPORT_NO )    /* If start/end value
is not yet retrieved */
            {
                ucState = STATE_WAIT_GO;       /* Re-read start/end
value */
                return;
            }

            fn_SetDAC( sResult.ushDAC );       /* Set DAC output */
            sResult.bDAC_ready = READY_OK;     /* Preparing DAC
setting value data is completed */

```

```

        sResult.usAD = fn_GetAD();                /* Acquire A/D
conversion result */
        sResult.bAD_ready = READY_OK;           /* Preparing A/D
conversion result data is completed */

        if( uc5CodeReady == CODE5_COMPLETE )    /* If measuring 5-point
correction data is completed */
        {
            sResult.usCorrect = fn_CorrectADresult( sResult.usAD );
/* Correct A/D conversion result */
            sResult.bCorrect_ready = READY_OK;   /* Preparing 5-point
correction result data is completed */
            if( HeatCorrJdg == JDG_HEATCORR_EX ) /* If temperature
correction is required */
            {
                sResult.usHeatCorr = fn_HeatCorrect( ushNowCode1,
sResult.usCorrect);/* Correct temperature */
                sResult.bHeatCorr_ready = READY_OK; /* Preparing
temperature correction result data is completed */
            }
            else
                sResult.bHeatCorr_ready = READY_NO; /* No 5-point
correction data measurement result data */
        }
        else                                     /* If 5-point
correction has not been completed */
        {
            sResult.bCorrect_ready = READY_NO;   /* No 5-point
correction result data */
            sResult.bHeatCorr_ready = READY_NO; /* No 5-point correction
result data */
        }
        fn_SetSendData( sResult, ucTxBuffer );  /* Set conversion
result output */
        fn_ADResultOut( ucTxBuffer );          /* Output data */

        /* Update DAC setting values */
        /*if( sResult.usDAC == ANA_MAX )        */ /* If conversion up to
maximum value is completed */
        /*      sResult.usDAC = 0;             */ /* Continue conversion
from minimum value */
        /*else*/
        /*      sResult.usDAC++;               */
        /*if(( ushEnd == ANA_MAX )&&( sResult.usDAC == 0 ))*/ /* If conversion
up to conversion end value is completed */
        /*      ucState = STATE_STOP;         */ /* End conversion */
        /*else if( sResult.usDAC == ( ushEnd + 1 ))*/
        /*      ucState = STATE_STOP;         */ /* End successive conversion
*/

```

```

        /* Update DAC setting values */
        for( temp = 0; temp < CONVERT_B_STEP; temp++ )
        {
            sResult.ushDAC ++;
            if( sResult.ushDAC == ( ushEnd + 1 ))
            {
                ucState = STATE_STOP;          /* End successive
conversion */

                break;
            }
        }
        break;

    default:
        break;
}

/*****

A/D conversion: Correction A mode control processing

-----

5-point correction mode (voltage input from other than the DAC) is controlled.
* Used only if the DAC is not used for A/D conversion analog input.
*****/
static void fn_Control_CorrectA(void)
{
    static struct st_Result sResult;          /* Store conversion result */
    static unsigned char ucTxBuffer[TXDATA_SIZE]; /* Display data buffer */
    static unsigned char ADCCount = 0;        /* Conversion count */

    switch( ucState ){ /* Branch according to execution status */
    case STATE_STOP: /* Stop processing */
        fn_Init_st_Result( &sResult );      /* Initialize conversion result
*/

        ADCCount = 0; /* Clear conversion count */
        ucState = STATE_STANDBY; /* Set operation status to
"Standby" */

        break;

    case STATE_WAIT_GO: /* Wait for progress key */
        break;

    case STATE_EXEC: /* Execute conversion processing */
        if( ADCCount == 0 )
            uc5CodeReady = CODE5_NOT_COMP; /* Measuring 5-point correction
data is not completed */

        sResult.bDAC_ready = READY_NO; /* No DAC setting value data */
    }
}

```

```

        sResult.ushAD = fn_GetAD();           /* Acquire A/D conversion
result */
        ushCode[ADCount] = sResult.ushAD;    /* Save A/D conversion result
*/
        sResult.bAD_ready = READY_OK;       /* Preparing A/D conversion
result data is completed */

        sResult.bCorrect_ready = READY_NO;   /* No 5-point correction result
data */
        sResult.bHeatCorr_ready = READY_NO; /* No 5-point correction result
data */

        fn_SetSendData( sResult, ucTxBuffer ); /* Set conversion result output
*/
        fn_ADResultOut( ucTxBuffer );        /* Output data */

        ADCount++;                           /* Update DAC setting values */

        if( ADCount >= 5 ){                  /* If converting all 5 points
is completed */
            ucState = STATE_STOP;            /* End conversion */
            if(( ushCode[0] < ushCode[2] ) /* If size relation among code1,
code3, and code5 is correct */
                && ( ushCode[2] < ushCode[4] )){
                uc5CodeReady = CODE5_COMPLETE; /* Report completion of
5-point correction data measurement */
                P_NOERR = 1;                  /* No error display */
                ushCorrCode1 = fn_CorrectADresult( ushCode[1-1] );
/* Correct and save code1 for temperature correction */
                ushCorrCode3 = fn_CorrectADresult( ushCode[3-1] );
/* Correct and save code3 for temperature correction */
            }
            else{                             /* If size relation among code1
to code5 is not correct */
                uc5CodeReady = CODE5_NOT_COMP; /* Measuring 5-point
correction data is not completed */
                P_NOERR = 0;                  /* Set error display */
            }
        }
        else{                                 /* If correcting 5 points is
not completed */
            ucState = STATE_WAIT_GO;         /* Wait for progress key and
perform next conversion */
        }
        break;

    default:
        break;
}

```

```

}

/*****

A/D conversion: Correction B mode control processing

-----

5-point correction mode (voltage input from the DAC) is controlled.
* Used only if the DAC is used for A/D conversion analog input.
*****/
static void fn_Control_CorrectB(void)
{
    static struct st_Result sResult;           /* Store conversion result */
    static unsigned char ucTxBuffer[TXDATA_SIZE]; /* Display data buffer */
    static unsigned char ADCCount = 0;        /* Conversion count */

    switch( ucState ){ /* Branch according to execution status */
    case STATE_STOP: /* Stop processing */
        fn_Init_st_Result( &sResult ); /* Initialize conversion result
*/
        ADCCount = 0; /* Clear conversion count */
        ucState = STATE_STANDBY; /* Set operation status to
"Standby" */
        break;

    case STATE_WAIT_GO: /* Wait for progress key */
        break;

    case STATE_EXEC: /* Execute conversion processing */
        if( ADCCount == 0 )
            uc5CodeReady = CODE5_NOT_COMP; /* Measuring 5-point correction
data is not completed */

        sResult.ushDAC = (unsigned short)fAna[ADCCount];
        fn_SetDAC( sResult.ushDAC ); /* Set DAC output */
        sResult.bDAC_ready = READY_OK; /* Preparing DAC setting value
data is completed */

        sResult.ushAD = fn_GetAD(); /* Acquire A/D conversion
result */
        ushCode[ADCCount] = sResult.ushAD; /* Save A/D conversion result
*/
        sResult.bAD_ready = READY_OK; /* Preparing A/D conversion
result data is completed */

        sResult.bCorrect_ready = READY_NO; /* Preparing 5-point correction
result data is completed */
        sResult.bHeatCorr_ready = READY_NO; /* No 5-point correction result
data */

```

```

        fn_SetSendData( sResult, ucTxBuffer ); /* Set conversion result output
*/
        fn_ADResultOut( ucTxBuffer );          /* Output data */

        ADCCount++;                            /* Update DAC setting values */
        if( ADCCount >= 5 ){                  /* If converting all 5 points
is completed */
            ucState = STATE_STOP;             /* End conversion */
            if(( ushCode[0] < ushCode[2] ) /* If size relation among code1,
code3, and code5 is correct */
                && ( ushCode[2] < ushCode[4] )){
                uc5CodeReady = CODE5_COMPLETE; /* Report completion of
5-point correction data measurement */
                P_NOERR = 1;                   /* No error display */
                ushCorrCode1 = fn_CorrectADresult( ushCode[1-1] );
/* Correct and save code1 for temperature correction */
                ushCorrCode3 = fn_CorrectADresult( ushCode[3-1] );
/* Correct and save code3 for temperature correction */
            }
            else{                               /* If size relation among code1
to code5 is not correct */
                uc5CodeReady = CODE5_NOT_COMP; /* Measuring 5-point
correction data is not completed */
                P_NOERR = 0;                   /* Set error display */
            }
        }
        break;

    default:
        break;
}
}

```

Retrieving A/D conversion start and end values from HD74HC165

 The A/D conversion start value (16 bits) and end value (16 bits), which have been set by using a switch, are read.

* Used only if the DAC is used for A/D conversion analog input.

```

static void fn_ImportStartEnd(void)
{
    unsigned short temp;          /* Temporary variable for data retrieval */
    unsigned char count;         /* Counter for data retrieval */

    /*-----
    A/D conversion start value retrieval

```



```

-----*/
temp = 0;
P_LAT = 1;                               /* Start data retrieval */
for( count = 16; count != 0; count-- ){ /* Read 16 bits of A/D conversion start
value from MSB */
    temp = ( temp << 1 );
    if( P_DAT )                           /* If retrieved data is "1", set 1 to
corresponding bit */
        temp++;
    P_CLK = 0;
    P_CLK = 1;                             /* Request shifting of next data */
}
ushStart = temp;                           /* Set A/D conversion start value */

/*-----
    A/D conversion end value retrieval
-----*/
temp = 0;
for( count = 16; count != 0; count-- ){ /* Read 16 bits of A/D conversion start
value from MSB */
    temp = ( temp << 1 );
    if( P_DAT )                           /* If retrieved data is "1", set 1 to
corresponding bit */
        temp++;
    P_CLK = 0;
    P_CLK = 1;                             /* Request shifting of next data */
}
ushEnd = temp;                             /* Set A/D conversion end value */

/* End all data retrieval */
P_LAT = 0;
P_CLK = 0;

ucImportState = IMPORT_OK;                /* Retrieval has been completed */
}

```

/*-----

Set the DAC

```

-----
The output voltages are set to DAC1 and DAC2.
* Used only if the DAC is used for A/D conversion analog input.
*****/
static void fn_SetDAC(unsigned short DAC1_value)
{
    unsigned short DAC0_value;             /* DAC0 setting data */

    /* Calculate value to be set to DAC0 */
    DAC0_value = ~DAC1_value;             /* Reverse DAC1 setting value */
}

```

```

        if(( DAC0_value == ANA_MAX ) || ( DAC0_value == ANA_MIN ))
            ; /* Set reverse value of DAC1 for
minimum/maximum setting */
        else
            DAC0_value += 1; /* Set "reverse value DAC1 + 1" for
settings other than minimum/maximum setting */

/*-----
                Set to DAC1
-----*/
/* Transmit setting to DAC1 */
P_CS1 = 0; /* Start transmission */
fn_SSIO_16bit( DAC1_value ); /* Output data */
P_CS1 = 1; /* End transmission */

/*-----
                Set to DAC0
-----*/
/* Transmit setting to DAC0 */
P_CS0 = 0; /* Start transmission */
fn_SSIO_16bit( DAC0_value ); /* Output data */
P_CS0 = 1; /* End transmission */

/* Wait until DAC stabilizes after serial transmission */
CMP00 = CMP00_AFTER_DAC; /* Set 500 us */
TMIFH0 = 0; /* Clear interrupt request */
TMHE0 = 1; /* Start timer operation */
while( !TMIFH0 ) ; /* Wait for 500 us to elapse */
TMHE0 = 0; /* Stop timer operation */
}

```

Software serial transmission

```

-----
Data (16 bits) is transmitted to the DAC via 3-wire serial communication.
* Used only if the DAC is used for A/D conversion analog input.
*****/
static void fn_SSIO_16bit(unsigned short value)
{
    unsigned short mask = 0b1000000000000000; /* Mask data */

    for( ; mask != 0; mask >>= 1 ) /* Transmit bit by bit from MSB */
    {
        P_SCK = 0; /* SCK: LOW */
        if(( value & mask ) == 0 ) /* Is transmit data at low level? */
            P_SDA = 0; /* Yes: Output transmit data (low
level) */
        else

```

```

        P_SDA = 1;          /* No: Output transmit data (high
level) */
        P_SCK = 1;          /* SCK: HIGH (DAC reads transmit data)
*/
    }
}

/*****

16-bit  $\Delta\Sigma$ -type A/D conversion

-----

 $\Delta\Sigma$ -type A/D conversion is started and A/D conversion result is returned.
*****/
static unsigned short fn_GetAD(void)
{
    unsigned short ret = 0;      /* A/D conversion result return value */

    /* Start A/D conversion */
    ADDCE = 1;                   /* Enable conversion operation
*/
    DSADIF = 0;                  /* Clear A/D conversion
completion interrupt request */

    /* Acquire A/D conversion result */
    while(1){                    /* Wait until A/D conversion is completed or stop key is
pressed */
        if(( DSADIF )&&( ADDSTR == 0 )){          /* If A/D conversion of DS0+ is
completed */
            ret = ADDCR;                          /* Read A/D conversion result
*/
            break;
        }else if(( DSADIF )&&( ADDSTR != 0 )){    /* If value of non-target
channel is converted */
            ADDCE = 1;                            /* Redo A/D conversion */
            DSADIF = 0;
        }else if(PIF0){                          /* If stop key is pressed */
            ADDCE = 0;                            /* Stop A/D conversion
operation */
            break;                                /* Abort A/D conversion */
        }else
            ;
    }

    return ( ret );              /* Return A/D conversion result
*/
}

/*****

```

5-point correction

 The A/D conversion result is corrected by using the following arithmetic expression for correction output.

If the denominator of the operation is 0, the A/D conversion result is returned as it is without performing a 5-point correction operation.

$$\frac{\text{A/D conversion result} - b_n}{a_n}$$

* Description of the correction coefficients

$$a_n = \frac{\text{codeA} - \text{codeB}}{\text{anaA} - \text{anaB}}$$

$$b_n = \frac{\text{anaB} \times \text{codeA} - \text{anaA} \times \text{codeB}}{\text{anaB} - \text{anaA}}$$

n: 1 to 4

n = 1 : anaA = ana2, anaB = ana1, codeA = code2, codeB = code1

n = 2 : anaA = ana3, anaB = ana2, codeA = code3, codeB = code2

n = 3 : anaA = ana4, anaB = ana3, codeA = code4, codeB = code3

n = 4 : anaA = ana5, anaB = ana4, codeA = code5, codeB = code4

```

*****/
static unsigned short fn_CorrectADresult( unsigned short ADresult )
{
    float  ret = ADresult;          /* Operation result: Initial value is value
before correction */
    unsigned short  temp;
    unsigned char   n;

    /* Identify coefficient to be used */
    if( ADresult >= ushCode[4-1] )      /* If A/D conversion result is code4 or
later */
        n = 4;
    else if(( ADresult >= ushCode[3-1] )) /* If A/D conversion result is code3 to
code4 (including code3) */
        n = 3;
    else if(( ADresult > ushCode[2-1] )) /* If A/D conversion result is code2 to
code3 */
        n = 2;
    else                                /* If A/D conversion result is code2 or
earlier */
        n = 1;

    /* Correct A/D conversion result */
    temp = ushCode[n] - ushCode[n-1];

```

```

        if(temp != 0){          /* If denominator of operation is 0, return A/D value
as it is without performing correction operation */
            ret = (float)ADresult*(fAna[n] - fAna[n-1]) + fAna[n-
1]*(float)ushCode[n] - fAna[n]*(float)ushCode[n-1];
            ret = ret / temp;
        }

        return ( unsigned short )( ret );      /* Return integer portion (16 bits) of
correction result */
    }

```

/******

Temperature correction

This function is used to perform temperature correction.

The 5-point correction result is corrected by converting the inclination of 5-point correction including temperature errors to an inclination of ideal values.

Operation is performed separately for the following two cases.

<1> The inclination of the 5-point correction result is smaller than that of the ideal values (high temperature)

<2> The inclination of the 5-point correction result is greater than that of the ideal values (low temperature)

If the denominator of the operation is 0, the A/D result is returned as it is without performing a temperature correction operation.

*****/

```

static unsigned short fn_HeatCorrect(unsigned short ushNowCode1, unsigned short
ushCode){

    unsigned long ret = ushCode;          /* Temperature correction result */
    unsigned short temp;                  /* Operation work area */

    temp = ushCorrCode3 - ushNowCode1;
    if( temp != 0 ){                      /* If denominator is 0, return correction
target as it is without performing correction operation */
        if( ushNowCode1 > ushCorrCode1 ) /* <1> */
        {
            ret = (unsigned long)ushCode*(unsigned long)(ushCorrCode3 -
ushCorrCode1);
            ret = ret - (unsigned long)ushCorrCode3*(unsigned
long)(ushNowCode1 - ushCorrCode1);
        }
        else                               /* <2> */
        {
            ret = (unsigned long)ushCode*(unsigned long)(ushCorrCode3 -
ushCorrCode1);
            ret = ret + (unsigned long)ushCorrCode3*(unsigned
long)(ushCorrCode1 - ushNowCode1);
        }
    }
}

```

```

    }
    ret = ret / temp;
}

return ( unsigned short )( ret );    /* Return correction result */

}

/*****

Create the data to be output via UART

-----

The argument values are converted to data to be output via UART and set to the
transmit buffer.
*****/
static void fn_SetSendData(struct st_Result s_Result, unsigned char *p_ucTxBuffer)
{
    unsigned char temp;    /* Data creation work area */
    unsigned char n;

    /* Set transmit data of DAC setting values */
    if( s_Result.bDAC_ready )
    {
        /* If transmit data of DAC setting values exists */
        for( n = 0; n < 4; n++ ){
            /* Extract digits to be converted to transmit data */
            temp = (unsigned char)((s_Result.ushDAC & ( 0xf000 >> 4*n )) >>
(4*(3 - n)));
            if( temp >= 0xa )    /* If transmit data is 0xa to 0xf */
                p_ucTxBuffer[n] = temp + ( 'A' - 0xa );    /*
Convert to "A" to "F" */
            else    /* If transmit data is 0x0 to 0x9 */
                p_ucTxBuffer[n] = temp + '0';    /*
Convert to "0" to "9" */
        }
    }else{ /* If transmit data of DAC setting values does not exist */
        for( n = 0; n < 4; n++ )
            p_ucTxBuffer[n] = '*';    /* Output "*" in place of DAC
setting values */
    }

    p_ucTxBuffer[4] = ' ';    /* SP between data */

    /* Set transmit data of A/D conversion values */
    if( s_Result.bAD_ready )
    {
        /* If transmit data of A/D setting values exists */
        for( n = 0; n < 4; n++ ){
            /* Extract digits to be converted to transmit data */
            temp = (unsigned char)((s_Result.ushAD & ( 0xf000 >> 4*n )) >>
(4*(3 - n)));

```

```

        if( temp >= 0xa )          /* If transmit data is 0xa to 0xf */
            p_ucTxBuffer[n + 5] = temp + ( 'A' - 0xa );          /*
Convert to "A" to "F" */
        else                        /* If transmit data is 0x0 to 0x9 */
            p_ucTxBuffer[n + 5] = temp + '0';                    /*
Convert to "0" to "9" */
    }
}
}else{ /* If transmit data of A/D setting values does not exist */
    for( n = 0; n < 4; n++ )
        p_ucTxBuffer[n + 5] = '*';          /* Output "*" in place
of DAC setting values */
    }

    p_ucTxBuffer[9] = ' ';          /* SP between data */

    /* Set 5-point correction transmit data */
    if( s_Result.bCorrect_ready )
    {          /* If 5-point correction transmit data exists */
        for( n = 0; n < 4; n++ ){
            /* Extract digits to be converted to transmit data */
            temp = (unsigned char)((s_Result.ushCorrect & ( 0xf000 >>
4*n )) >> (4*(3 - n)));
            if( temp >= 0xa )          /* If transmit data is 0xa to 0xf */
                p_ucTxBuffer[n + 10] = temp + ( 'A' - 0xa );          /*
Convert to "A" to "F" */
            else                        /* If transmit data is 0x0 to 0x9 */
                p_ucTxBuffer[n + 10] = temp + '0';                    /*
Convert to "0" to "9" */
        }
    }
}else{ /* If 5-point correction transmit data does not exist */
    for( n = 0; n < 4; n++ )
        p_ucTxBuffer[n + 10] = '*';          /* Output "*" in place
of DAC setting values */
    }

    p_ucTxBuffer[14] = ' ';          /* SP between data */

    /* Set temperature correction result transmit data */
    if( s_Result.bHeatCorr_ready )
    {          /* If temperature correction result transmit data exists */
        for( n = 0; n < 4; n++ ){
            /* Extract digits to be converted to transmit data */
            temp = (unsigned char)((s_Result.ushHeatCorr & ( 0xf000 >>
4*n )) >> (4*(3 - n)));
            if( temp >= 0xa )          /* If transmit data is 0xa to 0xf */
                p_ucTxBuffer[n + 15] = temp + ( 'A' - 0xa );          /*
Convert to "A" to "F" */
            else                        /* If transmit data is 0x0 to 0x9 */
                p_ucTxBuffer[n + 15] = temp + '0';                    /*
Convert to "0" to "9" */
        }
    }
}

```

```

    }
}else{ /* If temperature correction result transmit data does not exist */
    for( n = 0; n < 4; n++ )
        p_ucTxBuffer[n + 15] = '*'; /* Output "*" in place
of DAC setting values */
    }

    p_ucTxBuffer[19] = '\r'; /* Carriage return */
    p_ucTxBuffer[20] = '\n'; /* Line feed */
}

/*****

Output the conversion result via UART6

-----

The transmit buffer contents are transmitted via UART.
*****/
static void fn_ADResultOut(unsigned char *p_TxBuffer)
{
    unsigned char ucTxBufferCounter = 0; /* Transmit counter */

    /* Start transmit operation */
    STIF6 = 1; /* Set interrupt request */

    while( ucTxBufferCounter < TXDATA_SIZE ){ /* Wait for completion of UART6
transmission */
        if( STIF6 ){
            STIF6 = 0; /* Clear interrupt request */

            TXB6 = p_TxBuffer[ucTxBufferCounter]; /* TxD6: Transmit data
*/
            ucTxBufferCounter++; /* Update transmit counter */
        }

        if(PIF0){ /* If stop key is pressed */
            STIF6 = 1; /* Set interrupt request */
            break;
        }
    }
}
}

```


APPENDIX B REVISION HISTORY

Edition	Date Published	Page	Revision
1st edition	July 2008	–	–

*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

Podbielskistrasse 166 B
30177 Hannover
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. 52
78142 Velizy-Villacoublay Cédex
France
Tel: 01-3067-5800

Sucursal en España

Juan Esplandiú, 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/>

Shanghai Branch

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/>

Shenzhen Branch

Unit 01, 39/F, Excellence Times Square Building,
No. 4068 Yi Tian Road, Futian District, Shenzhen,
P.R.China P.C:518048
Tel:0755-8282-9800
<http://www.cn.necel.com/>

NEC Electronics Hong Kong Ltd.

Unit 1601-1613, 16/F., Tower 2, Grand Century Place,
193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: 2886-9318
<http://www.hk.necel.com/>

NEC Electronics Taiwan Ltd.

7F, No. 363 Fu Shing North Road
Taipei, Taiwan, R. O. C.
Tel: 02-8175-9600
<http://www.tw.necel.com/>

NEC Electronics Singapore Pte. Ltd.

238A Thomson Road,
#12-08 Novena Square,
Singapore 307684
Tel: 6253-8311
<http://www.sg.necel.com/>

NEC Electronics Korea Ltd.

11F., Samik Lavied'or Bldg., 720-2,
Yeoksam-Dong, Kangnam-Ku,
Seoul, 135-080, Korea
Tel: 02-558-3737
<http://www.kr.necel.com/>