F&eIT Series

Isolated Analog Input Module

# ADI12-8(FIT)GY

# User's Manual

CONTEC CO.,LTD.

# Check Your Package

Thank you for purchasing the CONTEC product.

The product consists of the items listed below.

Check, with the following list, that your package is complete. If you discover damaged or missing items, contact your retailer.

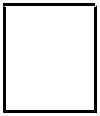
Product Configuration List

- Module …1
- First Step Guide …1
- CD-ROM [F&eIT Series Setup Disk] *1…1
- Interface connector plugs …2

*1 The CD-ROM contains various software and User's Manual (this manual)


Module


Interface connector plugs


First step guide


CD-ROM
[F&eIT Series Setup Disk]

# Copyright



# Trademarks

# Table of Contents

# 1. Introduction

Congratulations on your recent purchase of an Insulator Digital Input Module.
By converting external analog voltage signals into digital data, the ADI12-8(FIT)GY can process them inside the F&eIT series controller module < CPU-CAxx(FIT)GYGY, CPU-SBxx(FIT)GY etc >.
The insulation between external signals and the Controller Module permits the use of the Controller Module without compromising the communications features of the latter.

Please read this manual carefully to create application programs and configure the system, such as setting the switches and connecting it to external devices.

## Features

- The input range is common to different channels, and can be selected from four input ranges, including unipolar and bipolar ranges.
- The ability to accommodate differential input permits the accurate measurement of voltage values over long distances from the signal source and even under a considerable potential difference.
- A rotary switch allows you to set device IDs to help you keep track of device numbers.
- The system incorporates a screwless connector plug that allows you to easily attach and detach wires without using any special tools.
- Similar to other F&eIT series products, the system, in the module itself, incorporates a 35-mm DIN rail mounting mechanism as a standard item. A connection to a controller module can be effected on a lateral, stack basis in a unique configuration, which permits a simple, smart system configuration without the need for a backplane board.

# Functions and control method by controller connected

The ADI12-8(FIT)GY can be connected to a variety of controllers.

| | | |
|---|---|---|
| Micro Controller Unit | : | CPU-SBxx(FIT)GY |
| I/O Controller Module | : | CPU-CAxx(FIT)GY |
| Monitoring & Control Server Unit | : | SVR-MMF2(FIT) |
| Monitoring & Control Server Unit | : | SVR-MMF(FIT)GY |
| Isolated Analog Input Module for USB | : | ADI12-8(USB)GY |
| I/O Controller Module with USB | : | CPU-CA10(USB)GY |

The functions and control of the ADI12-8(FIT)GY vary with the controller to which the ADI12-8(FIT)GY is connected.

**Functions available with each controller connected**

| | CPU-SBxx(FIT)GY | CPU-CAxx(FIT)GY | SVR-MMF2(FIT) | SVR-MMF(FIT)GY | ADI12-8(USB)GY | CPU-CA10(USB)GY |
|---|---|---|---|---|---|---|
| Software input range setting | o | o | *1 | *1 | o | o |
| A/D conversion with software command | o | o | | | o | o |
| Continuous A/D conversion based on internal sampling | o*2 | | | | o*3 | |
| Interrupt function | o | | | | | |
| Device ID setting range | 0 - 7 | 0 - 7 | 0 - 7 | 0 - 7 | 1 - 3 | 0 - 7 |

*1 For the function available, refer to the reference manual for the SVR-MMF2(FIT), SVR-MMF(FIT)GY.

*2 Sampling timer setting: 10 to 1,073,741,824 μsec.

*3 Using the sampling timer built in the ADI12-8(USB)GY.
The setting range is from 1000 to 1,073,741,000 μsec.

**Control method by controller connected**

| | | CPU-SBxx(FIT)GY | CPU-CAxx(FIT)GY | SVR-MMF2(FIT) | SVR-MMF(FIT)GY | ADI12-8(USB)GY | CPU-CA10(USB)GY |
|---|---|---|---|---|---|---|---|
| Control using the I/O address map | | o | | | | | |
| Control using the memory address map | | | o | | | | |
| Control via the Windows driver * | FIT Protocol | | o | | | | |
| | API-CAP(W32) | | o | | | | |
| | API-SBP(W32) | o | | | | | |
| | API-USBP(WDM) | | | | | o | o |
| Control over the web | | | | o | o | | |

* The API-SBP(W32) is included in the development kit DTK-SBxx(FIT)GY; the other drivers are bundled with each controller.

**Control using the I/O address map**

When connected to the CPU-SBxx(FIT)GY, the ADI12-8(FIT)GY can receive I/O instructions directly from the controller module. For details, see Chapter 4 “Using the I/O Address Map”.

**Control using the memory address map**

When connected to the CPU-CAxx(FIT)GY, the ADI12-8(FIT)GY can be accessed from the host computer over the network.

The ADI12-8(FIT)GY is assigned with its device ID in the memory managed by the controller module. The application running on the host computer controls the module by reading/writing the memory managed by the controller module. For details, see Chapter 5 “Using the Memory Address Map”.

**Control via the Windows driver**

For the functions and settings available when using the Windows driver, refer to the reference manual and online help for each module.

**Control over the web**

You can monitor collected data and manage the log over the web. You can use your familiar browser to easily make various settings. For details, refer to the reference manual for the SVR-MMF2(FIT), SVR-MMF(FIT)GY.

## Limited One-Year Warranty

CONTEC products are warranted by CONTEC CO., LTD. to be free from defects in material and workmanship for up to one year from the date of purchase by the original purchaser.

Repair will be free of charge only when this product is returned freight prepaid with a copy of the original invoice and a Return Merchandise Authorization to the distributor or the CONTEC group office, from which it was purchased.

This warranty is not applicable for scratches or normal wear, but only for the electronic circuitry and original products. The warranty is not applicable if the device has been tampered with or damaged through abuse, mistreatment, neglect, or unreasonable use, or if the original invoice is not included, in which case repairs will be considered beyond the warranty policy.

## How to Obtain Service

For replacement or repair, return the device freight prepaid, with a copy of the original invoice. Please obtain a Return Merchandise Authorization Number (RMA) from the CONTEC group office where you purchased before returning any product.

*** No product will be accepted by CONTEC group without the RMA number.**

## Liability

The obligation of the warrantor is solely to repair or replace the product. In no event will the warrantor be liable for any incidental or consequential damages due to such defect or consequences that arise from inexperienced usage, misuse, or malfunction of this device.

## Handling Precautions

Take the following precautions when handling this module.

-Do not modify the module. CONTEC will bear no responsibility for any problems, etc., resulting from modifying this module.

-Do not use or store the equipment in a hot or cold place, or in a place that is subject to severe temperature changes. (Operating temperature range: 0 - 50°C)

-Do not use or store the equipment in a place subject to direct sunlight or near a heating device, such as a stove.

-Do not use or store the equipment in a dusty or humid place. (Operating humidity range: 10 - 90%RH, No condensation)

-As this product contains precision electronic components, do not use or store in environments subject to shock or vibration.

-Do not use or store the product near equipment generating a strong magnetic field or radio waves.

-If you notice any strange odor or overheating, please unplug the power cord immediately.

-In the event of an abnormal condition or malfunction, please consult the dealer from whom the equipment was purchased.

-To avoid electric shock, please do not touch the system with a wet hand.

-Do not open the module casing. CONTEC will disclaim any responsibility for equipment whose casing has been opened.

-To prevent damage, please do not subject the module to impact or bend it.

-To prevent contact malfunction, please do not touch the metallic pins on the external module connector.

-The module contains switches that need to be properly set. Before using the module, please check its switch settings.

-To avoid malfunction, please do not change the module switch settings in an unauthorized manner.

-"Do not operate the device module when the power for the Controller Module is on. To avoid malfunction, please be sure to turn off the power for the Controller Module."

**FCC PART 15Class A Notice**

NOTE

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in commercial environment.
This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference at his own expense.

WARNING TO USER

Change or modifications not expressly approved the manufacturer can void the user's authority to operate this equipment.

# About the Manual

This manual consists of the following chapters:

| | |
|---|---|
| Chapter 1 | Introduction |
| Chapter 2 | Module Nomenclature and Settings<br>Explains the nomenclature of the components of the I/O Controller Module and their operations. |
| Chapter 3 | Connecting to an External Device<br>Explains interface connectors and external I/O circuits. |
| Chapter 4 | Functioning as a CPU-SBxx(FIT)GY Module<br>Explains I/O port bit assignements and the definitions of the bits when the Module is used as a CPU-SBxx(FIT)GY module. |
| Chapter 5 | Functioning as a CPU-CAxx(FIT)GY Module<br>Explains the module settings area, the information area, and the I/O data area when the Module is used as a CPU-CAxx(FIT)GY module. |
| Chapter 6 | System Reference<br>Explains module specifications and circuit diagrams. |

# 2. Module Nomenclature and Settings

## Nomenclature of Module Components

Figure 2.1. shows the names of module components. In the figure, the indicated switch settings represent factory settings.

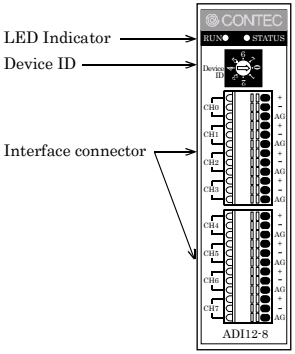



**Figure 2.1. Names of module components**

# Setting a Device ID

The controller module distinguishes and keeps track of the modules that are connected to it by assigning device IDs to them. Each module, therefore, should be assigned a unique ID.

A Device ID can be assigned in a 0 - 7 range, so that a maximum of eight modules can be distinguished.

To connect the ADI12-8(FIT)GY to the ADI12-8(USB)GY, assign a device ID between 1 and 3.

The factory setting for the Device ID is [0].

## Setup Method

A device ID can be set by turning the rotary switch on the device face.
To set a device ID, turn the switch knob.

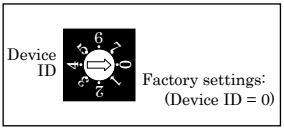



**Figure 2.2. Setting a Device ID**

# LED Indicator

RUN: Indicates that the ADI12-8(FIT)GY can be controlled from the Controller Module. (green)

STATUS: This light comes on when an A/D conversion error occurs. (red)

# 3. Connecting to an External Device

# Interface Connector

## How to Connect an Interface Connector

When connecting the Module to an external device, you can use the supplied connector plug. When wiring the Module, strip off approximately 7 - 8 mm of the covering for the cable, and insert the bare wire by pressing the orange button on the connector plug. Releasing the orange button after the wire is inserted to fix the cable. Compatible wires are AWG 28 - 20.

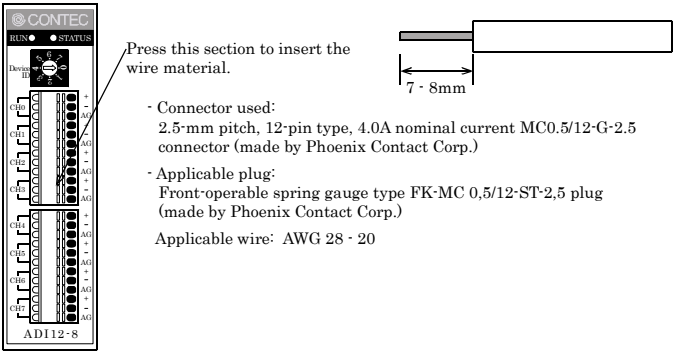



**Figure 3.1. Connecting an interface connector and connectors that can be used**

***Note!***

*Removing the connector plug by grasping the cable can break the wire.*

# Signal Layout on the Interface Connector

The Module can be connected to an external device using a 12-pin (1 group) connector that is provided on the Module face.

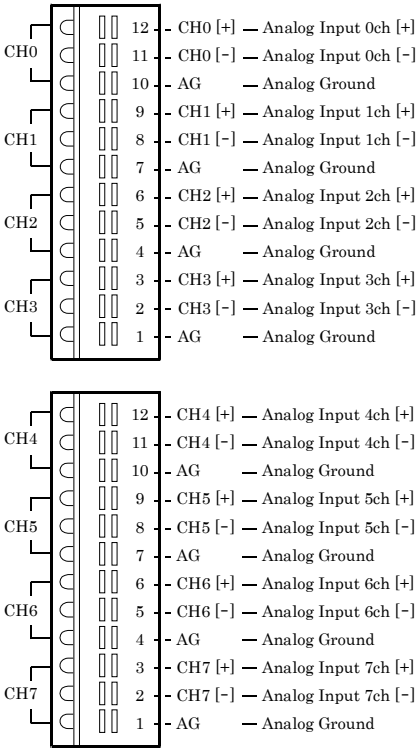



**Figure 3.2. Signal layout on the interface connector**

## Example of connecting a current input

To measure a current using the ADI12-8(FIT)GY, you need to convert the current into a voltage using a resistor.

By connecting a 250Ω resistor between the [+] and [-] inputs, the ADI12-8(FIT)GY can measure currents from a 0 - 20mA current output device in a voltage input range from 0 to 5 V. Note that, if the inserted resistor has a considerable error, it adversely affects converted data, preventing accurate measurement. You should therefore use a precision resistor (±0.1 %) on the connector side.

Note also that, when there are more than one current source, no potential difference must exist between their respective GND points.

The ADI12-8(FIT)GY is insulated between its internal CPU and the external device but not between the analog input channels and thus uses a common analog ground.

If an affecting potential difference exists between the channels, insert an insulator such as an insulating transducer between the channels.

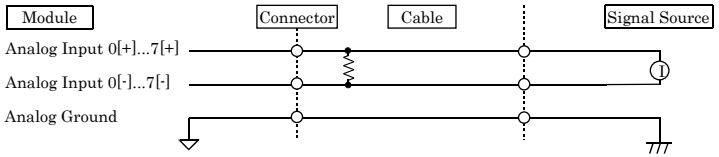



**Figure 3.5. Current input connection**

***Notes!***

- *Converted data is undefined with no analog ground connected.*
- *If the connection cable is affected by noise, the analog input can be inaccurate. Route the connection cable apart from noise sources.*
- *The analog signal input to the [+] and [-] inputs must not exceed the maximum input voltage with reference to the analog ground of the module. Exceeding the input voltage can damage the module.*
- *Converted data is undefined when either of the [+] and [-] input terminals is left unconnected. Connect both of the [+] and [-] input terminals of the channel not connected to the signal source to the analog ground.*

# 4. Using the I/O Address Map

## Starting I/O Address

When connected to a CPU-SBxx(FIT)GY, the ADI12-8(FIT)GY can directly receive I/O commands from the controller module. Depending on how the Device ID is set, the I/O addresses indicated below will be used exclusively by the ADI12-8(FIT)GY.

Because the address bus on which the I/O address space is specified is not fully decoded in continued 16 bits, four starting I/O addresses exist in each Device ID.

If the Device ID is set to 0h, one of the four addresses (0800h, 0840h, 0880h, or 08C0h) will be used as a starting I/O address.

**Table 4.1. List of starting I/O addresses.**

| ID No. | Occupied I/O address | | | |
|---|---|---|---|---|
| 0 | 0800h - 081Fh(recommend) | 0840h - 085Fh | 0880h - 089Fh | 08C0h - 08DFh |
| 1 | 1800h - 181Fh(recommend) | 1840h - 185Fh | 1880h - 189Fh | 18C0h - 18DFh |
| 2 | 2800h - 281Fh(recommend) | 2840h - 285Fh | 2880h - 289Fh | 28C0h - 28DFh |
| 3 | 3800h - 381Fh(recommend) | 3840h - 385Fh | 3880h - 389Fh | 38C0h - 38DFh |
| 4 | 4800h - 481Fh(recommend) | 4840h - 485Fh | 4880h - 489Fh | 48C0h - 48DFh |
| 5 | 5800h - 581Fh(recommend) | 5840h - 585Fh | 5880h - 589Fh | 58C0h - 58DFh |
| 6 | 6800h - 681Fh(recommend) | 6840h - 685Fh | 6880h - 689Fh | 68C0h - 68DFh |
| 7 | 7800h - 781Fh(recommend ) | 7840h - 785Fh | 7880h - 789Fh | 78C0h - 78DFh |

For detailed specifications on the I/O space that is managed by the controller module, see the controller module manual.

# List of I/O Address Maps

**Input Port**

| Starting I/O address | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| input +0 (00h) | Product Category | | | | Revision Data | | | |
| | 0 | 0 | 1 | 0 | Revision Data3 | Revision Data2 | Revision Data1 | Revision Data0 |
| +1 (01h) | Product ID Number | | | | | | | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| +2 (02h) | Interrupt Status | | | | | | | |
| | Enable | Status | 0 | 0 | 0 | IRQ9 | IRQ7 | IRQ5 |
| +3 (03h) | (Not available) | | | | | | | |
| ⋮ | | | | | | | | |
| +16 (10h) | Analog input Data (Lower) | | | | | | | |
| | Conversion Data7 | Conversion Data6 | Conversion Data5 | Conversion Data4 | Conversion Data3 | Conversion Data2 | Conversion Data1 | Conversion Data0(LSB) |
| +17 (11h) | Analog input Data (Upper) | | | | | | | |
| | 0 | 0 | 0 | 0 | Conversion Data11(MSB) | Conversion Data10 | Conversion Data9 | Conversion Data8 |
| +18 (12h) | (Not available) | | | | | | | |
| +21 (15h) | | | | | | | | |
| +22 (16h) | Analog Input Status | | | | | | | |
| | 0 | 0 | Sampling Clock Error | Sampling Clock Input | 0 | Data Over Write Error | 0 | Data Read Enable |
| +23 (17h) | (Not available) | | | | | | | |
| ⋮ | | | | | | | | |
| +31 (1Fh) | | | | | | | | |

**Figure 4.1. Input port**

**Output Port**

| Starting I/O address output | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| +0 (00h) | (Not allowed) | | | | | | | |
| +1 (01h) | Interrupt Data | | | | | | | |
| +2 (02h) | Enable | N/A | N/A | N/A | N/A | IRQ9 Data | IRQ7 Data | IRQ5 Data |
| +3 (00h) | (Not allowed) | | | | | | | |
| ⋮ | | | | | | | | |
| +17 (11h) | | | | | | | | |
| | Channel Data | | | | | | | |
| +18 (12h) | N/A | N/A | N/A | N/A | N/A | Channel Data2 | Channel Data1 | Channel Data0 |
| +19 (13h) | (Not allowed) | | | | | | | |
| +21 (15h) | | | | | | | | |
| | Status Reset | | | | | | | |
| +22 (16h) | N/A | N/A | Sampling Clock Error | Sampling Clock Input | N/A | Data Over Write Error | N/A | Data Read Enable |
| +23 (17h) | (Not allowed) | | | | | | | |
| | Command | | | | | | | |
| +24 (18h) | Command Data7 | Command Data6 | Command Data5 | Command Data4 | Command Data3 | Command Data2 | Command Data1 | Command Data0 |
| +25 (19h) | (Not allowed) | | | | | | | |
| ⋮ | | | | | | | | |
| +27 (1Bh) | | | | | | | | |
| | Setting Data 0 | | | | | | | |
| +28 (1Ch) | Setting Data07 | Setting Data06 | Setting Data05 | Setting Data04 | Setting Data03 | Setting Data02 | Setting Data01 | Setting Data00 |
| | Setting Data 1 | | | | | | | |
| +29 (1Dh) | Setting Data15 | Setting Data14 | Setting Data13 | Setting Data12 | Setting Data11 | Setting Data10 | Setting Data09 | Setting Data08 |
| | Setting Data 2 | | | | | | | |
| +30 (1Eh) | Setting Data23 | Setting Data22 | Setting Data21 | Setting Data20 | Setting Data19 | Setting Data18 | Setting Data17 | Setting Data16 |
| | Setting Data 3 | | | | | | | |
| +31 (1Fh) | Setting Data31 | Setting Data30 | Setting Data29 | Setting Data28 | Setting Data27 | Setting Data26 | Setting Data25 | Setting Data24 |

**Figure 4.2. Output port**

# Specifications Common to F&eIT Products

The regions with starting I/O addresses +0h - +Fh are maps that are common to all modules in the F&eIT series.

## Product Information

| Starting I/O address input | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| +0 (00h) | Product Category | | | | Revision Data | | | |
| | 0 | 0 | 1 | 0 | Revision Data3 | Revision Data2 | Revision Data1 | Revision Data0 |
| +1 (01h) | Product ID Number | | | | | | | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 4.3. Product information**

-Revision Data [D3 - D0]:
This is product update information, subject to change without notice, that is managed by CONTEC.

-Product Category [D7 - D4]:
This is a module function classification code. For the ADI12-8(FIT)GY, the code is "2h".

**Table 4.2. Product Category**

| Code | Function |
|---|---|
| 0 | Extension BUS |
| 1 | Digital input-output |
| 2 | Analog input-output |
| 3 | Counter |
| 4 | Serial communication |
| 5 | GPIB |
| 6-F | Reserved |

-Products ID Number [D7 - D0]:
This is the product ID within the same product category. For the ADI12-8(FIT)GY, the product ID is "0h".

Following are examples of the initialization that is performed in high-level languages:

Microsoft C

```
ProductID = inp( ADR+1 );
```

Microsoft QBASIC

```
ProductID = INP( ADR+1 )
```

*ADR is the starting I/O address for the ADI12-8(FIT)GY.

**Interrupt Status**

This is a common port on which the interrupt status requested by the Module can be verified. Although in this example values are assigned centered on the status concerning interrupt levels, information on interrupt sources varies from module to module.

| Starting I/O address | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| input +2 (02h) | Interrupt Status | | | | | | | |
| | Enable | Status | 0 | 0 | 0 | IRQ9 | IRQ7 | IRQ5 |

**Figure 4.4. Interrupt status**

-Enable [D7]:
This verifies the interrupt source enabled/disabled status.
The value "1" indicates that a hardware interrupt on the controller module is enabled.
This bit indicates an interrupt request status in the module.
When IRQ5, IRQ7, or IRQ9 is "1", this bit will also be "1".

-IRQ* [D2 - D0]:
These bits allow you to verify the interrupt level that is currently set. The current interrupt level is indicated as "1".

Following are examples of the initialization that is performed in high-level languages:

Microsoft C

```
IrqStatus = inp( ADR+2 );
```

Microsoft QBASIC

```
IrqStatus = INP( ADR+2 )
```

**Setting an Interrupt Level**

| Starting I/O address | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| | Interrupt Data | | | | | | | |
| output +2 (02h) | Enable | N/A | N/A | N/A | N/A | IRQ9 Data | IRQ7 Data | IRQ5 Data |

**Figure 4.5. Setting an interrupt level**

-Enable [D7]:
This bit enables an interrupt source.

-IRQ* [D2 - D0]:
The interrupt level used by the module is set in these bits.

Following are examples of initialization settings that can be effected in high-level languages:

The interrupt level to be used is assigned to IRQ5.

Microsoft C

```
outp( ADR+2, 0x81 );
```

Microsoft QBASIC

```
OUT ADR+2, &H81
```

# Overview of the Sampling Function

When a start-sampling command is issued, analog input signals are converted into 12-bit digital data at a maximum rate of 10µsec/ch + 20µsec under pre-set sampling conditions.

Two sampling modes are supported:

- Software mode: Stores data from a channel specified by a data-fetch command into an internal read buffer.
- Clock mode: Stores data from a specified channel into an internal read buffer in synchronization with a programmable timer.

Two channel specification methods are supported:

- Single channel: Can read data from a specified channel.
- Multi-channel: Can read data from multiple channel, beginning with channel 0, up to a specified channel.

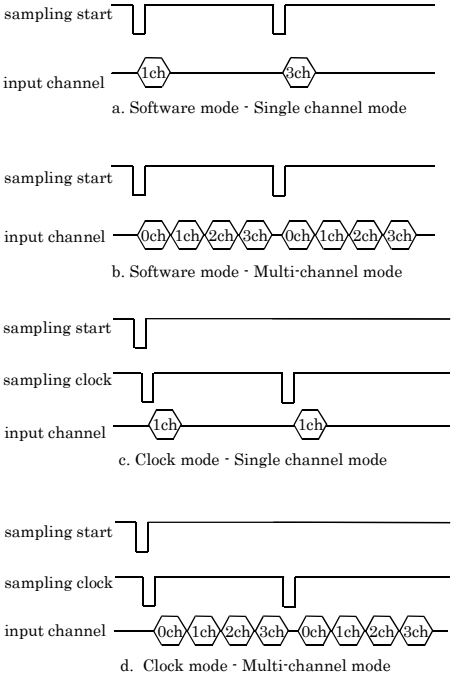



**Figure 4.6. Basic analog input operation**

The sampling operation can be checked by monitoring the status.

An interrupt can also be generated as the status changes.

Figure 4.7. illustrates the analog input procedure.

The initialization must be performed prior to any sampling.

In the next step, sampling conditions (operating mode, input range, and so forth) must be set.

In the final step, the start command is issued, and the conversion data is input.

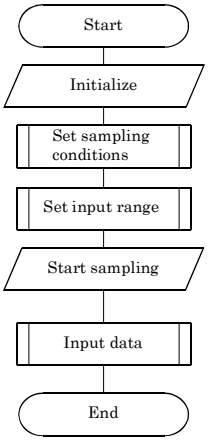



**Figure 4.7. Analog input procedure**

**Initialization**

This step initializes the analog input function.

This command clears all settings and status, and returns the module to the "initialized state", which is the same as when the power is turned on and the RESET button is pressed.

During the initialization, the control port assumes the following state:

| Starting I/O address | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| output +24 (18h) | Command | | | | | | | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 4.8. Initialization**

Following are examples of the initialization that is effected in high-level languages:

Microsoft C

```
outp( ADR+24, 0x0 );
```

Microsoft QBASIC

```
OUT ADR+24, &H0
```

**Setting sampling conditions**

This step sets sampling conditions.

In terms of procedures, a sampling condition setup command is issued, and then settings data is output.

| Starting I/O address | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| output +24 (18h) | Command | | | | | | | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

| output +28 (1Ch) | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| | Sampling Setting | | | | | | | |
| | 0 | 0 | 0 | 0 | 0 | Channel Mode | Sampling Clock | Sampling Mode |

**Figure 4.9. Setting sampling conditions**

-Channel mode [D2]:

Set the mode in which the sampling is to be performed.
Select either the "single-channel mode", in which only one channel is specified, or the "multi-channel mode", in which two or more channels are specified.

Channel Mode [0]: Single *Initialized state
[1]: Multi

-Sampling clock [D1]:

This option should be set when the clock mode is selected in the specification of a sampling mode.

Sampling Clock [0]: Internal Clock * Initialized state
[1]: Reserved

-Sampling mode [D0]:

This step sets the conversion operation.
Specify either the "software mode", in which a specified channel is sampled once, or the "clock mode", in which sampling is performed periodically according to clock signals.

Sampling Mode [0]: Software Command * Initialized state
[1]: Clock

Following are examples in which sampling conditions are specified in high-level languages:

Microsoft C

```
outp( ADR+24, 0x2 );
outp( ADR+28, ConditionData );
```

Microsoft QBASIC

```
OUT ADR+24, &H2
OUT ADR+28, ConditionData
```

**Input range-setting**

The input range refers to the voltage range in which analog signals are input.

All channels are set on a common basis, and the input data is converted into digital signals with a 12-bit resolution.

The input range-setting control port assumes the following state:

| Starting I/O address | | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|
| output +24 (18h) | Command | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| output +28 (1Ch) | Range Setting | Range Data7 | Range Data6 | Range Data5 | Range Data4 | Range Data3 | Range Data2 | Range Data1 | Range Data0 |

**Figure 4.10. Setting an input range**

**Table 4.3. Input range and settings data**

| Range | Input range |
|---|---|
| 00h | ±10 V |
| 01h | ±5 V |
| 02h - 7Fh | Not decied |
| 80h | 0 - 10 V |
| 81h | 0 - 5 V |
| 82h and above | Not decied |

Following are examples in which an input range is set in high-level languages:

The input range is set to 0 - 10V:

Microsoft C

```
outp( ADR+24, 0x3 );
outp( ADR+28, 0x80 );
```

Microsoft QBASIC

```
OUT ADR+24, &H3
OUT ADR+28, &H80
```

**Setting an Internal Sampling Clock**

When either the "clock mode" or the "internal sampling clock" is selected as a sampling condition, this option allows you to set a sampling cycle (clock data). In the initial state, the clock data is undefined. Clock data must be set when an internal sampling clock is used.

Clock data is set in 250-nsec increments. The allowable range is 10, 000nsec - 1, 073, 741, 824, 000nsec (approximately 17 minutes and 54 seconds), which corresponds to the setting data 39 - 4, 294, 967, 295.

The relationship between a clock cycle and setting data can be expressed as follows:

$$Clock\ data = \frac{Sampling\ clock}{250} - 1$$

where the sampling clock is specified in units of *nsec*.
Sampling clock values must satisfy the following expression:

$$\text{Sampling clock} \geq 10000\text{nsec} \times \text{Number of specified channels} + 20\mu\text{sec}\ (10\mu\text{sec})$$

Sampling in accurate cycles cannot be performed if the specified value is shorter than the conversion time for a specified number of channels.

The control port for setting an internal sampling clock assumes the following state:

| Starting I/O address | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| output +24 (18h) | Command | | | | | | | |
| | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| output +28 (1Ch) | Timer Data 0 | | | | | | | |
| | Timer Data07 | Timer Data06 | Timer Data05 | Timer Data04 | Timer Data03 | Timer Data02 | Timer Data01 | Timer Data00 |
| +29 (1Dh) | Timer Data 1 | | | | | | | |
| | Timer Data15 | Timer Data14 | Timer Data13 | Timer Data12 | Timer Data11 | Timer Data10 | Timer Data09 | Timer Data08 |
| +30 (1Eh) | Timer Data 2 | | | | | | | |
| | Timer Data23 | Timer Data22 | Timer Data21 | Timer Data20 | Timer Data19 | Timer Data18 | Timer Data17 | Timer Data16 |
| +31 (1Fh) | Timer Data 3 | | | | | | | |
| | Timer Data31 | Timer Data30 | Timer Data29 | Timer Data28 | Timer Data27 | Timer Data26 | Timer Data25 | Timer Data24 |

**Figure 4.11. Setting an internal sampling clock**

Following are examples in which an internal sampling clock is set in high-level languages:

Microsoft C

```
outp( ADR+24, 0x4 );
outp( ADR+28, ClockData0 );
outp( ADR+29, ClockData1 );
outp( ADR+30, ClockData2 );
outp( ADR+31, ClockData3 );
```

Microsoft QBASIC

```
OUT ADR+24, &H4
OUT ADR+28, ClockData0
OUT ADR+29, ClockData1
OUT ADR+30, ClockData2
OUT ADR+31, ClockData3
```

**Starting a sampling process**

If the sampling mode is the single-channel mode, specify the channel through which data is to be converted.

In the case of the multi-channel mode, specify the upper limit (1 or greater) on the channels through which data is to be converted.

Example: Specifying "4ch" results in the sampling of channels 0 - 4ch.

The control port for commencing a sampling process assumes the following state:

| Starting I/O address | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| output +18 (12h) | Channel Data | | | | | | | |
| | N/A | N/A | N/A | N/A | N/A | Channel Data2 | Channel Data1 | Channel Data0 |

**Figure 4.12. Starting the sampling process**

If the sampling mode is the software command, the specified channel is sampled only once.

In the case of an internal sampling clock, the internal sampling clock starts simultaneously with the commencement of the first sampling.

If a sampling process is started with the sampling mode set to the clock mode, and then the sampling process is restarted, any conversion data that has accumulated up to that point and the analog input status are reset, and a new sampling operation is started.

The sampling stops when the module initialization command or the sampling condition-setting command is executed.

Following are examples in which a sampling process is started in high-level languages:

Microsoft C

```
outp( ADR+18, 0x4 );
```

Microsoft QBASIC

```
OUT ADR+18, &H4
```

**Input of conversion data**

Conversion data should be input only after a verification is made that conversion data is stored in a register.

Conversion data cannot be input from a register during a conversion operation.

The figure on the right shows procedures by which conversion data is input.

Conversion data is in the offset binary form. The relationship between conversion data and input voltages is indicated by the following expression:

$$Data = \frac{(Voltage + Offset)}{Span} \times 2^{12}$$

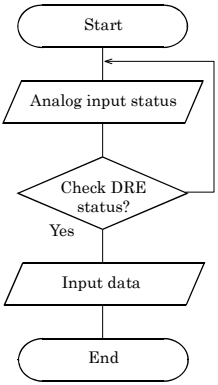



**Figure 4.13. Conversion data input procedures**

**Table 4.4. Input range**

| Input range | Offset | Span | Input range | Offset | Span |
|---|---|---|---|---|---|
| -10V - +10V | 10 | 20 | 0V - +10V | 0 | 10 |
| -5V - +5V | 5 | 10 | 0V - +5V | 0 | 5 |

**Table 4.5. Example of conversion data for a ±10V range**

| Input voltage (+/-10V range) | 12-bit conversion data Offset binary |
|---|---|
| +9.995V | 0FFF h |
| : | : |
| 0.005V | 0801 h |
| 0.000V | 0800 h |
| -0.005V | 07FF h |
| : | : |
| -10.000V | 0000 h |

The control port by which conversion data is input assumes the following state:

| Starting I/O address | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| input +22 (16h) | Interrupt Status | | | | | | | |
| | 0 | 0 | Sampling Clock Error | Sampling Clock Input | 0 | Data Over Write Error | 1 | Data Read Enable |

| | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| input +16 (10h) | Analog Input Data (Lower) | | | | | | | |
| | Conversion Data7 | Conversion Data6 | Conversion Data5 | Conversion Data4 | Conversion Data3 | Conversion Data2 | Conversion Data1 | Conversion Data0(LSB) |
| +17 (11h) | Analog Input Data (Upper) | | | | | | | |
| | 0 | 0 | 0 | 0 | Conversion Data11(MSB) | Conversion Data10 | Conversion Data9 | Conversion Data8 |

**Figure 4.14. Input control port for conversion data**

Details on analog input and the interrupt status will be described in the following section.

Following are examples in which a sampling process is started in high-level languages:

Following are examples in which conversion data is input in high-level languages.

Microsoft C

```
while( !(inp( ADR+22 ) &2 );
LowerAiData = inp( ADR+16 );
UpperAiData = inp( ADR+17 );
```

Microsoft QBASIC

```
WHILE(( INP( ADR+22 )AND 2 ) = 0 ) : WEND
LowerAiData = INP( ADR+16 )
UpperAiData = INP( ADR+17 )
```

**Details on the Analog Input Status**

The analog input status shows the status of an A/D conversion operation.

| Starting I/O address input +22 (16h) | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| | Analog Input Status | | | | | | | |
| | 0 | 0 | Sampling Clock Error | Sampling Clock Input | 0 | Data Over Write Error | 0 | Data Read Enable |

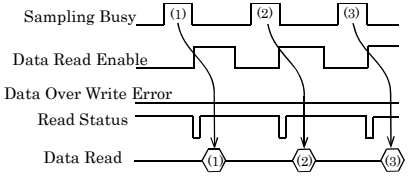
**Figure 4.15. Analog input status**

-Data Read Enabled Status (DRE) [D0]:

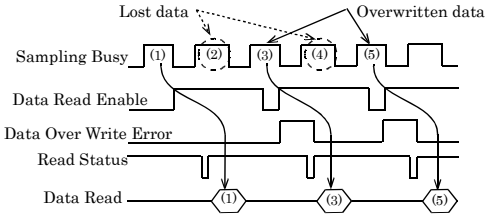
This indicates that conversion data is available that can be read.
If the channel mode is the single-channel mode, when any readable conversion data is stored, this status is set to [1]. In the case of the multi-channel mode, this status is set to [1] when conversion data equal to the number of specified channels is available.
When readable conversion data is depleted, this status is cleared. *

-Data Overwrite Error Status (DOWE) [D2]:

When this status is set to [1], it indicates that the data input interval is greater than the sampling clock interval during a clock-mode operation, and therefore the readable conversion data is being overwritten. If this status is detected, you need to either increase the sampling clock interval or reduce the READ processing time.
This status is cleared when there is no longer data to be overwritten.*




a. Normal timing




b. With "Data Overwrite Error" set

**Figure 4.16. Set/reset timing for the data overwrite error status**

-Sampling Clock Input Status (SCI) [D4]:

This status is set to [1] when a sampling clock is entered after a start-sampling command is issued in the clock mode. The status is cleared when it is reset. *

-Sampling Clock Error Status (SCE) [D5]:
This status is set to [1] when a sampling clock in entered during a sampling operation in the clock mode. The status is cleared when it is reset, and any sampling clocks that are entered during the sampling operation will be ignored. *
* The various status indicators are also cleared to [0] under the following conditions:

-When the initialization command is issued

-When a sampling-condition-setting command is issued

-When a start-sampling command is issued (except when the status is the "busy sampling status")

**Details on Resetting the Status**

This step clears the analog input status.

| Starting I/O address | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| output | Status Reset | | | | | | | |
| +22 (16h) | 0 | 0 | Sampling Clock Error | Sampling Clock Input | 0 | Data Over Write Error | 0 | |

**Figure 4.17. Status reset**

-Data Read Error Status Clear (DRE) [D0]:
Setting the value [1] clears the data read error status.

-Data Over Write Error Status Clear (DOWE) [D2]:
Setting the value [1] clears the data over write error status.

-Sampling Clock Input Status Clear (SCI) [D4]:
Setting the value [1] clears the sampling clock input status.

-Sampling Clock Error Status Clear (SCE) [D5]:
Setting the value [1] clears the sampling clock error status.

**Interrupt Function**

This option allows you to use the hardware interrupt function. For interrupt levels, a level that is set by the Module will be used. When using the interrupt function, you can pre-select one of the following status conditions as an interrupt source (multiple settings allowed):

**Table 4.6. Interrupt function**

| Status | Explanation |
|---|---|
| Data Read Enable | Data Read Enable status set |
| Data Overwrite Error | Data Overwrite Error status set |
| Sampling Clock Input | Sampling clock is input (internally) |
| Sampling Clock Error | Sampling Clock Error status set |

* More on this status later

An interrupt request signal is generated simultaneously with the setting of the status that is specified as an interrupt source. If two or more interrupt sources are specified, you can specify a specific interrupt signal generation source by entering a status in the interrupt handler.

**Setting an Interrupt Source**

This option allows you to specify an interrupt signal generation source.

The control port that sets an interrupt source assumes the following state:

| Starting I/O address | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| output +24 (18h) | Command | | | | | | | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

| | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| output +28 (1Ch) | Interrupt Source | | | | | | | |
| | 1 | 1 | Sampling Clock Error | Sampling Clock Input | 1 | Data Over Write Error | 1 | Data Read Enable |

**Figure 4.18. Interrupt sources**

When the value [1] is output as an interrupt source, the control port is masked; when the value [0] is output, the port is set as an interrupt source.

[1] Masked * Initialized state

[0] Interrupt Request Enable

Following are examples in which a timer cycle is set in high-level languages:

Microsoft C

```
outp( ADR+24, 0x1 );
outp( ADR+28, InterruptFactor );
```

Microsoft QBASIC

```
OUT ADR+24, &H1
OUT ADR+28,InterruptFactor
```

# List of Commands

Following is a list of ADI12-8(FIT)GY commands that are issued to "Output port +24":

**Table 4.7. List of commands**

| No. | HEX | Function | Data length |
|---|---|---|---|
| 00 | 0 | Initialization | 0-bit |
| 01 | 1 | Interrupt source mask | 8-bit |
| 02 | 2 | Sampling settings | 8-bit |
| 03 | 3 | Input range | 8-bit |
| 04 | 4 | Internal sampling clock | 32-bit |
| 05 | 5 | Timer start | 0-bit |
| 06 | 6 | Timer stop | 0-bit |

# Examples

## Software Mode

**Flowchart**

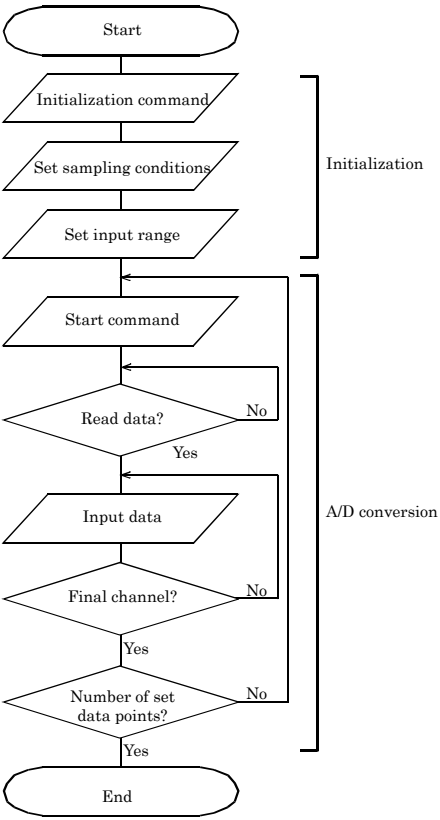



**Figure 4.19. Software mode**

**Sample program**

```
/*=========================================================
   Sample program 1

      DEVICE ID:        0
      Mode:             Software Mode, Multi-Channel
      Channel:          0 to 7ch
      Range:            -10 to 10V
      Internal Clock:   N/A
      Interrupt:        N/A

=========================================================*/
#include <stdio.h>
#include <conio.h>

/* ----- Constant ----------------------------------- */
#define  ADR      0x0800                  /* I/O address */

/* ----- Prototype ---------------------------------- */
void  main( void );

/* ----- Main --------------------------------------- */
void  main( void )
{
   unsigned char     UpperData, LowerData;
   unsigned int      i, j;

   outp( ADR+0x18, 0x00 );                  /* Initialize */
   outp( ADR+0x18, 0x02 );              /* Sampling Mode */
   outp( ADR+0x1c, 0x04 );                   /* Software */
   outp( ADR+0x18, 0x03 );                      /* Range */
   outp( ADR+0x1c, 0x00 );                 /* -10 to 10V */

   for(i = 0; i < 10; i++) {
      outp( ADR+0x12, 0x7 );/* Channel data & Conversion Start */
      while( !( inp( ADR+0x16 ) & 0x01 ) );
      for(j = 0; j < 8; j++) {
         LowerData = (unsigned char)inp( ADR+0x10 );
         UpperData = (unsigned char)inp( ADR+0x11 );
         printf("%01dch %02x%02x  ", j, UpperData, LowerData);
      }
   }
}

/* ----------------------------------- End of file ----*/
```

# Clock Mode (No Interrupts)

**Flowchart**

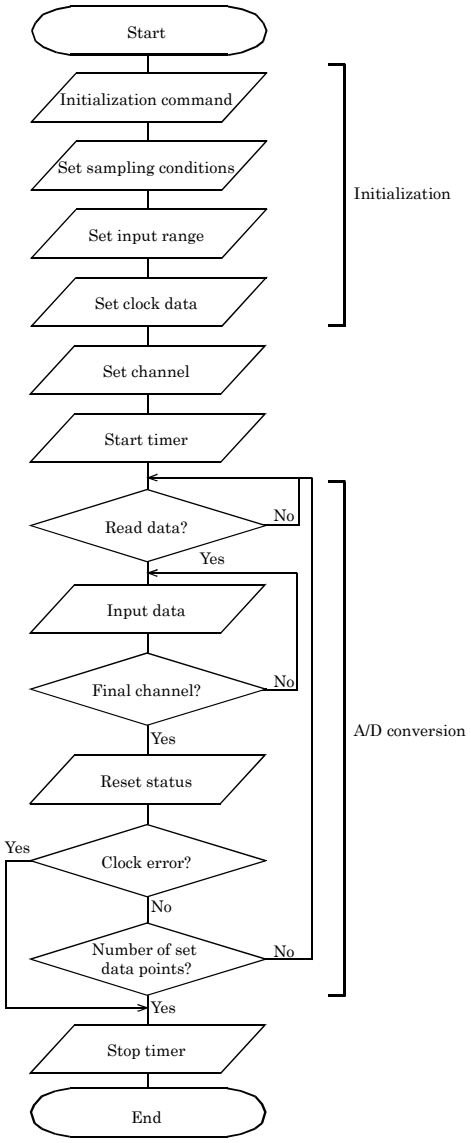



**Figure 4.20. Clock mode (no interrupts)**

**Sample program**

```
/*=======================================================
   Sample program 2

      DEVICE ID:        0
      Mode:             Clock Mode, Multi-Channel
      Channel:          0 to 7ch
      Range:            -10 to 10V
      Internal Clock:   250msec (250ns x 1,000,000)
      Interrupt:        N/A

======================================================= */
#include <stdio.h>
#include <conio.h>

/* ----- Constant ------------------------------------- */
#define  ADR         0x0800              /* I/O address */
#define  NUM         10                /* Sampling Times */

/* ----- Prototype ------------------------------------ */
void  main( void );

/* ----- Main ----------------------------------------- */
void  main( void )
{
   unsigned char     UpperData, LowerData, sts;
   unsigned int      i, j;
   float             VDAT;

   outp( ADR+0x18, 0x00 );                /* Initialize */
   outp( ADR+0x18, 0x02 );              /* Sampling Mode */
   outp( ADR+0x1c, 0x05 );                    /* Clock */
   outp( ADR+0x18, 0x03 );                    /* Range */
   outp( ADR+0x1c, 0x00 );                /* -10 to 10V */
   outp( ADR+0x18, 0x04 );                /* Timer Data */
   outp( ADR+0x1c, 0x3f );          /* 250ns x 1,000,000 */
   outp( ADR+0x1d, 0x42 );
   outp( ADR+0x1e, 0x0f );
   outp( ADR+0x1f, 0x00 );

   outp( ADR+0x12, 0x07 );               /* Channel data */
   outp( ADR+0x18, 0x05 );/* Conversion Start & Timer Start */

   for(i = 0; i <NUM; i++) {
      do {
         sts = (unsigned char)inp( ADR+0x16 );
      } while( !( sts & 0x01 ) );

      for(j = 0; j < 8; j++) {
       LowerData = (unsigned char)inp( ADR+0x10 );
       UpperData = (unsigned char)inp( ADR+0x11 );
       VDAT = (UpperData*0x100+LowerData)*20.0f/4096.0f-10.0f;
       printf("%01dch Input Data:%02x%02x Input Voltage:%7.3fV\n",
j, UpperData, LowerData, VDAT);
      }

      sts = (unsigned char)inp( ADR+0x16 );
```

```
            outp( ADR+0x16, sts & 0x10 );      /* Status reset */

            if( sts & 0x20 ) {
               i = NUM;
               printf("\nClock Error\n");
            } else printf("\n");
         }
         outp( ADR+0x18, 0x06 );                /* Timer Stop */
      }

      /* ------------------------------------ End of file --- */
```

# Clock Mode (with Interrupts)

**Flowchart**

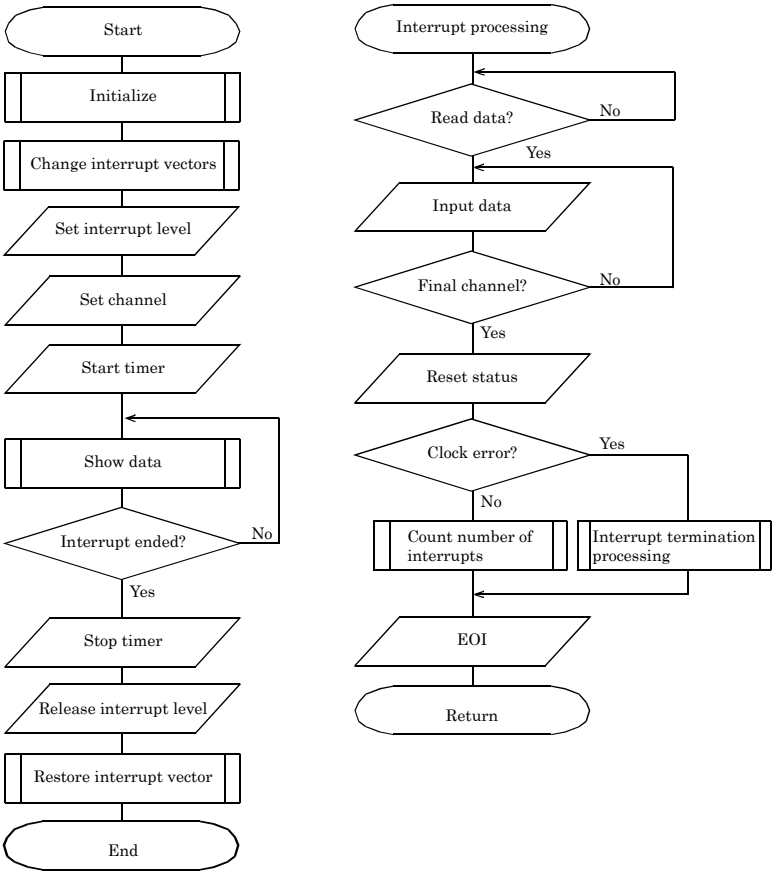



**Figure 4.21. Clock mode (with interrupts)**

```
/*=======================================================
   Sample program 3

      DEVICE ID:        0
      Mode:             Clock Mode, Multi-Channel
      Channel:          0 to 3ch
      Range:            -10 to 10V
      Internal Clock:   1sec (250ns x 4,000,000)
      Interrupt:        IRQ5 10 times

=======================================================
*/
#include <stdio.h>
#include <conio.h>
#include <dos.h>

/* ----- Constant ------------------------------------ */
#define           ADR   0x0800            /* I/O address */
#define           CH                         /* Channels */
#define           NUM   10             /* Sampling Times */
#define           IRQ5  0                      /* IRQ5 */
#define           IRQ7  1                      /* IRQ7 */
#define           IRQ9  2                      /* IRQ9 */

volatile unsigned int   AdData[CH][NUM];     /* A/D Data */
volatile int         intcnt = 0;    /* interrupt counter */
volatile int         IrqLevel = IRQ5; /* interrupt level */
int                  OrgMasterImr, OrgSlaveImr;
                                          /* original IMR */
unsigned char        IntVector[3] = { 0x0d, 0x0f, 0x71 };
                                       /* interrupt vector */
unsigned char        PicMask[3] = { 0xdf, 0x7f, 0xfd };
                                            /* mask bit */
unsigned char        IsrClear[3] = { 0x65, 0x67, 0x61 };
                                          /* ISR clear */
unsigned char        IntEnable[3] = { 0x81, 0x82, 0x84 };
                                     /* interrupt enable */

/* ----- Prototype ----------------------------------- */
void  main( void );
void  Initialize( void );                  /* initialize */
void  ChgVect( void );                  /* change vector */
void  ResVect( void );                 /* restore vector */
void  _interrupt _far inthandler( void );
                                       /* interrupt handler */
void  ( _interrupt _far *OrgVect)();
                          /* original interrupt vector */


/* ----- Initialize ---------------------------------- */
void  Initialize( void )
{
   outp( ADR+0x18, 0x00 );                 /* Initialize */
   outp( ADR+0x18, 0x02 );              /* Sampling Mode */
   outp( ADR+0x1c, 0x05 );                       /* Clock */
   outp( ADR+0x18, 0x03 );                       /* Range */
   outp( ADR+0x1c, 0x00 );                 /* -10 to 10V */
   outp( ADR+0x18, 0x04 );                 /* Timer Data */
```

```
   outp( ADR+0x1c, 0xff );          /* 250ns x 4,000,000 */
   outp( ADR+0x1d, 0x08 );
   outp( ADR+0x1e, 0x3d );
   outp( ADR+0x1f, 0x00 );
   outp( ADR+0x18, 0x01 );           /* Interrupt Factor */
   outp( ADR+0x1c, 0xef );/* Sampling Clock Input Mask OFF */
}


/* ----- change vector -------------------------------- */
void  ChgVect( void )
{
   OrgVect = _dos_getvect( IntVector[IrqLevel] );
   _disable();
   _dos_setvect( IntVector[IrqLevel], inthandler );
   if ( IrqLevel > IRQ7 ) {        /* IMR and mask clear */
      outp( 0x21, ( OrgMasterImr = inp( 0x21 ) ) & 0xfb );
      outp( 0xa1, ( OrgSlaveImr = inp( 0xa1 ) ) &
         PicMask[IrqLevel] );
      outp( 0x20, 0x62 );          /* ISR clear (master) */
      outp( 0xa0, IsrClear[IrqLevel] );/* ISR clear (slave) */
   } else {                        /* IMR and mask clear */
      outp( 0x21, ( OrgMasterImr = inp( 0x21 ) ) &
         PicMask[IrqLevel] );
      outp( 0x20, IsrClear[IrqLevel] );     /* ISR clear */
   }
   _enable();                                 /* enable */
}


/* ----- restore vector ------------------------------- */
void  ResVect( void )
{
   _disable();                                /* disable */
   if ( IrqLevel > IRQ7 ) {               /* restore IMR */
      outp( 0x21, OrgMasterImr );
      outp( 0xa1, OrgSlaveImr );
   } else
      outp( 0x21, OrgMasterImr );
   _dos_setvect( IntVector[IrqLevel], OrgVect );
                                          /* restore orgvect */
   _enable();                                 /* enable */
}


/* ----- interrupt handler ---------------------------- */
void  _interrupt _far inthandler( void )
{
   unsigned int      i;
   unsigned char     UpperData, LowerData, sts;
   _enable();                                 /* enable */
   sts = (unsigned char)inp( ADR+0x16 );
   if ( sts & 0x01 ){
      for(i = 0; i < CH; i++) {
         LowerData = (unsigned char)inp( ADR+0x10 );
         UpperData = (unsigned char)inp( ADR+0x11 );
```

```
            AdData[i][intcnt] = UpperData*0x100+LowerData;
         }
         intcnt++;
      }
      sts = (unsigned char)inp( ADR+0x16 );
      outp( ADR+0x16, sts & 0x10 );         /* Status reset */
      if( sts & 0x20 )  intcnt = 32767;
      _disable();                                  /* disable */
      if ( IrqLevel > IRQ7 ) {                        /* EOI */
         outp( 0xa0, 0x20 );
         outp( 0xa0, 0x0b );
            if ( !inp( 0xa0 ) )            outp( 0x20, 0x20 );
      } else         outp( 0x20, 0x20 );
}
/* ----- main ---------------------------------------- */
void  main( void )
{
      unsigned int   i, j;
      float          Volt;
      Initialize();                             /* initialize */
      ChgVect();                              /* change vector */
      outp( ADR+0x2, IntEnable[IrqLevel] ); /* interrupt level */
      outp( ADR+0x12, CH-1 );                 /* Channel data */
      outp( ADR+0x18, 0x05 );/* Conversion Start & Timer Start */
      while( intcnt <= NUM )
         printf("interrupt count = %02d \n", intcnt);
      printf("\n\n");
      outp( ADR+0x18, 0x06 );                   /* Timer Stop */
      outp( ADR+0x2, 0x0 );                /* interrupt level */
      ResVect();                             /* restore vector */
      for(j = 0; j < NUM; j++) {
         for(i = 0; i < CH; i++) {
            Volt = AdData[i][j]*20.0f/4096.0f-10.0f;
            printf("%01dch %04x %7.3fV   ", i, AdData[i][j],
Volt);
         }
      }
      printf("\n\n");
      if( intcnt == 32767 )  printf("Sampling Clock Error\n");
}
/* ------------------------------------ End of file --- */
```

# 5. Using the Memory Address Map

When connected to a CPU-CAxx(FIT)GY, the ADI12-8(FIT)GY can be accessed by a host computer through a network. In addition, the Module can be allocated to the memory controlled by the Controller Module according to a given Device ID. Applications running on the host computer control the I/O modules by reading/writing the memory that is controlled by the Controller Module.

For detailed specifications on the memory controlled by the Controller Module, see the Controller Module manual.

Following is an explanation of the memory areas necessary for the use of the ADI12-8(FIT)GY: the "module settings area", the "module information area", and the "basic input data area".

**Module Settings Area**

This area controls the settings and how the module is started.

The module becomes available when the necessary settings are written into this area and the module activation option is set in the [module startup register].

**Module Information Area**

The current module settings are stored in this area.

When the Module is started, the contents of the Module Settings Area are copied to the Module Information Area. By reading this area, you can verify the current module settings.

**Basic Input Data Area**

Basic input data is read in this area.

# Module Settings Area

A module settings area, which is a 128-byte (80h) area beginning with address 301000h and corresponding to a given Device ID, is where the settings for the given device are written.

The starting address can be determined according to the following expression:

Starting address **= 301000h + 80h x (Device ID)**

**Table 5.1. Starting address**

| Address (h) | Area | Item | Size | Access type | Initial value (h) | Initial settings |
|---|---|---|---|---|---|---|
| Starting address+00 | Module-specific information | Module type (category) | 1 | R | 02 | ADI12-8(FIT)GY |
| Starting address+01 | | Module type (serial No.) | 1 | R | 00 | |
| Starting address+02 | | System-reserved (revision No.) | 1 | R | None | |
| Starting address+03 | | Supported functions | 1 | R | 01 | Basic Input |
| Starting address+04 | | Number of basic input channels | 1 | R | 08 | 8 channels |
| Starting address+05 | | Basic input data size | 1 | R | 02 | 2 bytes |
| Starting address+06 | | Number of basic output channels | 1 | R | 00 | 0 channel |
| Starting address+07 | | Basic output data size | 1 | R | 00 | 0 byte |
| Starting address+08 | | Input channel settings address | 1 | R | 20 | 20h |
| Starting address+09 | | Input channel settings data size | 1 | R | 06 | 6 bytes |
| Starting address+0A | | Output channel settings address | 1 | R | 50 | 50h |
| Starting address+0B | | Output channel settings data size | 1 | R | 06 | 6 bytes |
| Starting address+0C to Starting address+0F | | Reserved | 4 | R | None | |
| Starting address+10 | Common to channels | Module startup register | 1 | R/W | 00 | |
| Starting address+11 | | Error status | 1 | R | 00 | |
| Starting address+12 | | Analog input resolution | 1 | R | 10 | 12-bit Analog input resolution |
| Starting address+13 | | Analog input range | 1 | R/W | 00 | -10V - +10V |
| Starting address+14 to Starting address+1F | | Reserved | 12 | R | None | |
| Starting address+20 to Starting address+7F | Channel settings | Reserved | 96 | R | None | |

**Module-specific information**

-Module type (category)
The ADI12-8(FIT)GY belongs to the analog module (02h) category.

-Module type (serial No.)
The ADI12-8(FIT)GY is an analog module with a serial No. 0 (00h).

-Supported functions
The ADI12-8(FIT)GY supports the basic input function (01h).
The basic input data takes analog input values.

-Number of basic input channels
The number of basic input channels for the ADI12-8(FIT)GY is 8 (08h).
Eight analog input channels are provided.

-Basic input data size
The basic input data size for the ADI12-8(FIT)GY is 2 (02h) bytes.
This is a 16-bit data area, of which 12 bits are used by the ADI12-8(FIT)GY.

-Number of basic output channels
The ADI12-8(FIT)GY does not take basic output data (00h).

-Basic input data size
The ADI12-8(FIT)GY does not take basic output data (00h).

-Input channel settings address
The ADI12-8(FIT)GY does not have channel-specific settings.
This field is provided for compatibility with other device modules.

-Input channel settings data size
The ADI12-8(FIT)GY does not have channel-specific settings.
This field is provided for compatibility with other device modules.

-Output channel settings address
The ADI12-8(FIT)GY does not have channel-specific settings.
This field is provided for compatibility with other device modules.

-Output channel settings data size
The ADI12-8(FIT)GY does not have channel-specific settings.
This field is provided for compatibility with other device modules.

**Items Common to Modules**

-Module startup register

Setting the module startup option (01h) causes the device module to be started. Setting the module startup option when the module is being started causes the module to be restarted.

00h: No operation
01h: Module startup

-Error status

The error status bits, which are not reflected in the module settings area, always remain [00h].

The error status on a module is stored in the module information area.

-Analog input resolution

The analog input resolution capacity of the ADI12-8(FIT)GY is fixed at 12 bits (0Ch).

-Analog input range

This field sets an analog input range.

**Table 5.2. Analog input range**

| Setting value of analog input range (h) | Analog input range |
|---|---|
| 00 | -10V - +10V |
| 01 | -5V - +5V |
| 32 | 0V - +10V |
| 33 | 0V - +5V |

**Channel Settings**

The ADI12-8(FIT)GY does not have channel-specific settings.

This field is provided for compatibility with other device modules.

# Module Information Area

The module information area is a 128-byte (80h) area beginning with address 300000h and corresponding to a given Device ID.

The starting address can be determined according to the following expression:

Starting address **= 300000h + 80h x (Device ID)**

**Table 5.3. Module information area**

| Address (h) | Area | Item | Size | Access type | Initial value (h) |
|---|---|---|---|---|---|
| Starting address + 00 | Module-specific information | Module type (category) | 1 | R | 02 |
| Starting address + 01 | | Module type (serial No.) | 1 | R | 00 |
| Starting address + 02 | | System-reserved (revision No.) | 1 | R | None |
| Starting address + 03 | | Supported functions | 1 | R | 01 |
| Starting address + 04 | | Number of basic input channels | 1 | R | 08 |
| Starting address + 05 | | Basic input data size | 1 | R | 02 |
| Starting address + 06 | | Number of basic output channels | 1 | R | 00 |
| Starting address + 07 | | Basic output data size | 1 | R | 00 |
| Starting address + 08 | | Input channel settings address | 1 | R | 20 |
| Starting address + 09 | | Input channel settings data size | 1 | R | 06 |
| Starting address + 0A | | Output channel settings address | 1 | R | 50 |
| Starting address + 0B | | Output channel settings data size | 1 | R | 06 |
| Starting address + 0C to Starting address + 0F | | Reserved | 4 | R | None |
| Starting address + 10 | Common to channels | Module startup register | 1 | R/W | 00 |
| Starting address + 11 | | Error status | 1 | R | 00 |
| Starting address + 12 | | Analog input resolution | 1 | R | 10 |
| Starting address + 13 | | Analog input range | 1 | R/W | 00 |
| Starting address + 14 to Starting address + 1F | | Reserved | 12 | R | None |
| Starting address + 20 to Starting address + 7F | Channel settings | Reserved | 96 | R | None |

When the module is started, the contents of the module setting area are stored in the module information area, with the exception of the [Module Startup Register] and the [Error Status].

-Module startup register

This register holds the module operating status.
Therefore, the fact that the module is shut down simply indicates that the module has not been started.

00h : Module shutdown
01h : Module operating

-Error status

This register stores the error status of the module.
The error status register is reset when the module is restarted.

00h : Normal status
21h : Module timeout

The module timeout status (21h) is an error status that does not usually occur, and indicates that an error occurred during an A/D conversion process. This status will be reset when the module is restarted. When the module timeout status is on, the integrity of analog input values cannot be guaranteed.

## Basic Input Data Area

The basic input data area, which is a 128-byte (80h) area beginning with address 304000h, corresponds to a given Device ID.

The starting address can be determined according to the following expression:

Starting address **= 304000h + 80h x (Device ID)**

**Table 5.4. Basic input data area**

| Address (h) | Area | Item | Size | Access type |
|---|---|---|---|---|
| Starting address+00<br>- Starting address+01 | CH0 | Analog input value | 2 | R |
| Starting address+02<br>- Starting address+03 | CH1 | Analog input value | 2 | R |
| Starting address+04<br>- Starting address+05 | CH2 | Analog input value | 2 | R |
| Starting address+06<br>- Starting address+07 | CH3 | Analog input value | 2 | R |
| Starting address+08<br>- Starting address+09 | CH4 | Analog input value | 2 | R |
| Starting address+0A<br>- Starting address+0B | CH5 | Analog input value | 2 | R |
| Starting address+0C<br>- Starting address+0D | CH6 | Analog input value | 2 | R |
| Starting address+0E<br>- Starting address+0F | CH7 | Analog input value | 2 | R |
| Starting address+10<br>- Starting address+7F | Reserved | | 112 | R |

**Analog input value**

Analog input values are stored as Little Endians.

**Table 5.5. Analog input value**

| | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| +00h | A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 |
| +01h | 0 | 0 | 0 | 0 | A11 | A10 | A9 | A8 |

The A/D conversion is performed on a channel-by-channel basis, with a timing that reads analog input values; the results are stored two bytes at a time. For this reason although there is data compatibility between the high and low bytes, there is no synchronization between channels.

**Conversion formula:**

**input voltage (V) = analog input value x span / $2^{12}$ – offset**

**Table 5.6. Conversion coefficients**

| Analog input value | Offset | Span |
|---|---|---|
| -10V - +10V | 10 | 20 |
| -5V - +5V | 5 | 10 |
| 0V - +10V | 0 | 10 |
| 0V - +5V | 0 | 5 |

**Table 5.7. Analog input range: an example of a conversion in the -10V - +10V range**

| Input voltage (V) | Analog input value (h) |
|---|---|
| +9.995 | 0FFF |
| : | : |
| 0.005 | 0801 |
| 0.000 | 0800 |
| -0.005 | 7FF |
| : | : |
| -10.000 | 0000 |

***Notes!***

- *Analog input values contain data that is valid during the operation of the module. When the module is shut down, the analog input values are undefined.*
- *An analog input value is 2 bytes per channel. In order to maintain compatibility between the high and low bytes, the data should be loaded in a single READ operation.*

# Examples

**Flowchart**

Following is an example in which the ADI12-8(FIT)GY is installed at Device ID :0:

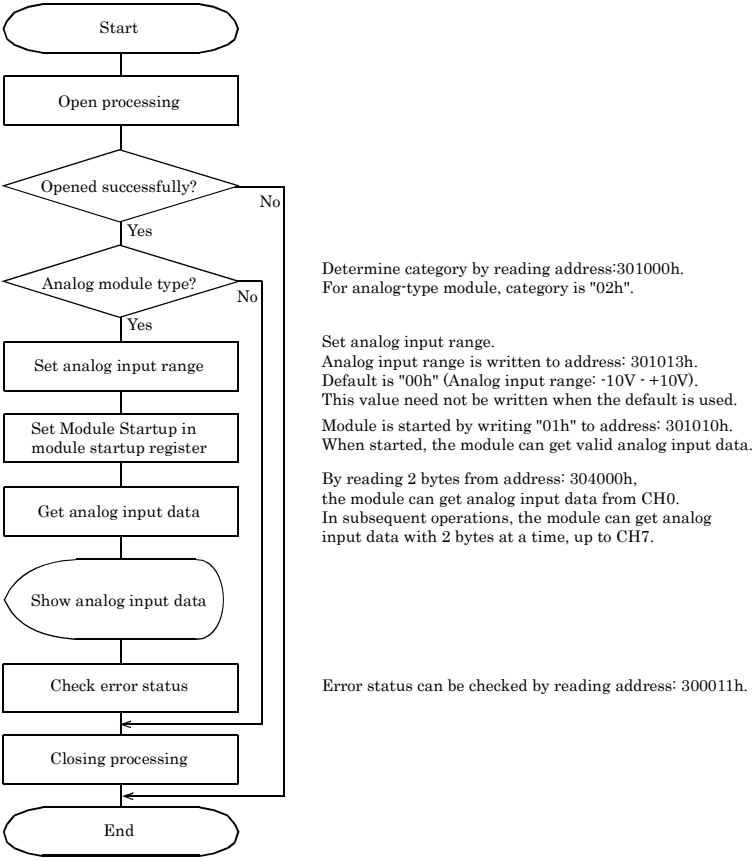



**Figure 5.1. Installed at Device ID:0**

**Sample program**

```
/*=======================================================
   F&eIT I/F Sample Program

      DEVICE ID:     0
      Channel:       0 to 7ch
      Range:         -10 to 10V
======================================================= */
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "Fit.h"

/* Address(common) */
#define FIT_IO                          (0x00300000)
#define FIT_IO_DEVICE_INFOR             (0x0000)
#define FIT_IO_DEVICE_CONFIG            (0x1000)
#define FIT_IO_INPUT                    (0x4000)
#define FIT_IO_OUTPUT                   (0x5000)

#define FIT_IO_DEVICE_SIZE              (0x0080)

#define FIT_PRODUCT_CATEGORY            (0x00)

#define FIT_MODULE_START                (0x10)
#define FIT_ERROR_STATUS                (0x11)

/* Information(Common) */
#define FIT_PRODUCT_DIGITAL             (0x01)
#define FIT_PRODUCT_ANALOG              (0x02)
#define FIT_PRODUCT_COUNTER             (0x03)

#define FIT_MODULE_START_OFF            (0x00)
#define  FIT_MODULE_START_ON            (0x01)

/* Address(AIO) */
#define FIT_AIO_AI_BIT                  (0x12)
#define FIT_AIO_AI_RANGE                (0x13)
#define FIT_AIO_AI_MODE                 (0x14)
#define FIT_AIO_AO_BIT                  (0x1A)
#define FIT_AIO_AO_RANGE                (0x1B)

/* Information(AIO) */
#define FIT_AIO_RANGE_PM10              (0)
#define FIT_AIO_RANGE_PM5               (1)
#define FIT_AIO_RANGE_P10               (50)
#define FIT_AIO_RANGE_P5                (51)
#define FIT_AIO_RANGE_P20MA             (100)
#define FIT_AIO_RANGE_P4TO20MA          (101)
```

```
/* Sample */
#define  FIT_SAMPLE_IP_ADDRESS        "192.168.132.211"
#define FIT_SAMPLE_PORT               (0x5007)
#define FIT_SAMPLE_DEVICE_ID          (0)

int main(void)
{
   DWORD dwIpAddress;
   DWORD dwVaBase;
   DWORD dwVaOffset;
   WORD  hHandle;
   WORD  wStatus;
   BYTE  byCategory;
   BYTE  byRange;
   BYTE  byModuleStart;
   BYTE  byData[0x80];
   BYTE  byChCount;
   BYTE  byErrorStatus;

   /* Open */
   dwIpAddress = FIT_IpChenge((BYTE
*)FIT_SAMPLE_IP_ADDRESS);
   hHandle = FIT_Open((BYTE *)&dwIpAddress, FIT_SAMPLE_PORT,
NULL);
   if (hHandle == 0) {
      printf("Error! FIT_Open = %04X(H)\n", hHandle);
      return 1;
   }

   /* Offset Address */
   dwVaOffset = FIT_IO_DEVICE_SIZE * FIT_SAMPLE_DEVICE_ID;

   /* Read 'Category' */
   dwVaBase = FIT_IO + FIT_IO_DEVICE_CONFIG;
   wStatus = FIT_Read(hHandle, dwVaBase + dwVaOffset +
FIT_PRODUCT_CATEGORY, 1, &byCategory);
   if (wStatus != 0) {
      printf("Error! FIT_Read = %04X(H)\n", wStatus);
      FIT_Close(hHandle);
      return 1;
   }
   if (byCategory != FIT_PRODUCT_ANALOG) {
      printf("Error! Category = %02X(H)\n", byCategory);
      FIT_Close(hHandle);
      return 1;
   }

   /* Write 'A/D Range' */
   byRange = FIT_AIO_RANGE_PM10;      /* Range:-10 to 10V */
   wStatus = FIT_Write(hHandle, dwVaBase + dwVaOffset +
FIT_AIO_AI_RANGE, 1, &byRange);
   if (wStatus != 0) {
      printf("Error! FIT_Write = %04X(H)\n", wStatus);
   }
```

```
   /* rite 'Module Start' */
   byModuleStart = FIT_MODULE_START_ON;
   wStatus = FIT_Write(hHandle, dwVaBase + dwVaOffset +
FIT_MODULE_START, 1, &byModuleStart);
   if (wStatus != 0) {
      printf("Error! FIT_Write = %04X(H)\n", wStatus);
   }

   /* Read 'A/D Data' */
   dwVaBase = FIT_IO + FIT_IO_INPUT;
   wStatus = FIT_Read(hHandle, dwVaBase + dwVaOffset, 2 * 8,
(BYTE *)&byData[0]);
   if (wStatus != 0) {
      printf("Error! FIT_Read = %04X(H)\n", wStatus);
   }
   for (byChCount = 0; byChCount < 8; byChCount++) {
      printf("A/D CH%d Data:%02X%02X\n", byChCount,
byData[byChCount * 2 + 1], byData[byChCount * 2]);
   }

   /* ead 'Error Status' */
   dwVaBase = FIT_IO + FIT_IO_DEVICE_INFOR;
   wStatus = FIT_Read(hHandle, dwVaBase + dwVaOffset +
FIT_ERROR_STATUS, 1, &byErrorStatus);
   if (wStatus != 0) {
      printf("Error! FIT_Read = %04X(H)\n", wStatus);
   }
   if (byErrorStatus != 0x00) {
      printf("Error! Error Status = %02X(H)\n",
byErrorStatus);
   }

   /* Close */
   FIT_Close(hHandle);

   return 0;
}
```

# 6. System Reference

## Block Diagram

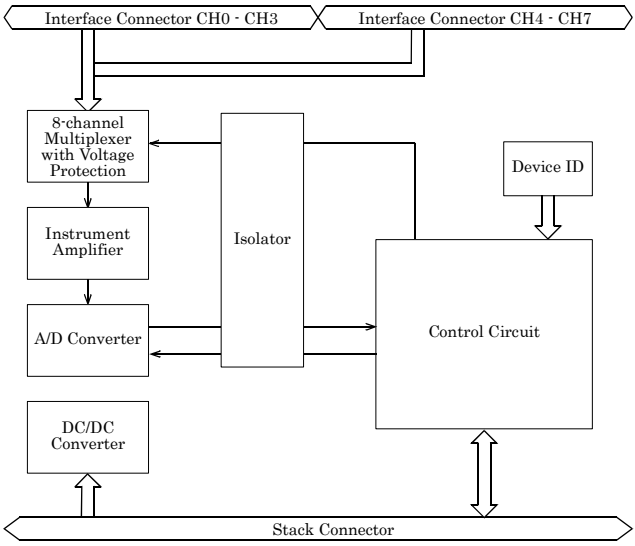



**Figure 6.1. Circuit block diagram**

# Specifications

**Table 6.1. Specifications**

| Item | | Specifications |
|---|---|---|
| Analog input section | | |
| | Input format | Bus-isolated voltage input |
| | Input range | Bipolar ±10V, ±5V Unipolar 0 - 10V, 0 - 5V |
| | Maximum input voltage | ±20V |
| | Input impedance | 1MΩ (Min.) |
| | Input channel | Differential input, 8 channels |
| | Resolution | 12-bit |
| | Non-linear error *1 | ±3LSB |
| | Conversion rate | Number of conversion channels x 10μsec + 20μsec |
| | Data buffer | 8-Word |
| | Interrupt | Either IRQ5 or IRQ7 or IRQ9 *2 |
| | Internal sampling timer | 10μsec - 1,073,741,824μsec *3 |
| Common section | | |
| | Internal power consumption | 5VDC±5% 350mA (Max.) |
| | Maximum distance of signal extension | 1.5m |
| | External dimensions (mm) | 25.2 (W) x 64.7 (D) x 94.0 (H) (exclusive of protrusions) |
| | Weight (module itself) | 100g |
| | Module connection method | Stack connection by the connector that is provided with the side of module |
| | Module installation method | One-touch connection to 35mm DIN rails (standard connection mechanism provided in the system) |
| | Applicable wire | AWG 28 - 20 |
| | Applicable plug | FK- MC 0,5/12-ST-2,5 (made by Phoenix Contact Corp.) |

*1 When the environment temperature is near 0ºC or 50ºC, the non-linearity error may become larger.

*2 Available only when the ADI12-8(FIT)GY is connected to the CPU-SBxx(FIT)GY.

*3 When the ADI12-8(FIT)GY is connected to the ADI12-8(USB)GY, the sampling timer built in the ADI12-8(USB)GY is used. The setting range is from 1,000 - 1,073,741,824μsec.

***Notes!***

- *When connecting one of the modules to a controller module, the internal power consumption should be taken into account. If the total current exceeds the capacity of the power supply unit, the integrity of the operation cannot be guaranteed. For further details, please see the Controller Module manual.*
- *Depending upon the specific controller module that is used, some of the functions are not supported.*

**Table 6.2. Installation Environment Requirements**

| Parameter | | Requirement description |
|---|---|---|
| Operating temperature | | 0 - 50°C |
| Storage temperature | | -10 - 60°C |
| Humidity | | 10 - 90% RH (No condensation) |
| Floating dust particles | | Not to be excessive |
| Corrosive gases | | None |
| Line-Noise resistance | Line-noise | AC line/2kV, Signal line/1kV (IEC1000-4-4Level 3, EN61000-4-4Level 3) |
| | Static electricity resistance | Contact discharge/4kV (IEC1000-4-2Level 2, EN61000-4-2Level 2)<br>Atmospheric discharge/8kV (IEC1000-4-2Level 3, EN61000-4-2Level 3) |
| Vibration resistance | Sweep resistance | 10 - 57Hz/semi-amplitude 0.15mm, 57 - 150Hz/2.0G<br>80minutes each in X, Y, and Z directions (JIS C0040-compliant, IEC68-2-6-compliant) |
| Impact resistance | | 15G, half-sine shock for 11ms in X, Y, and Z directions<br>(JIS C004-compliant, IEC68-2-27-compliant) |

# External Dimensions

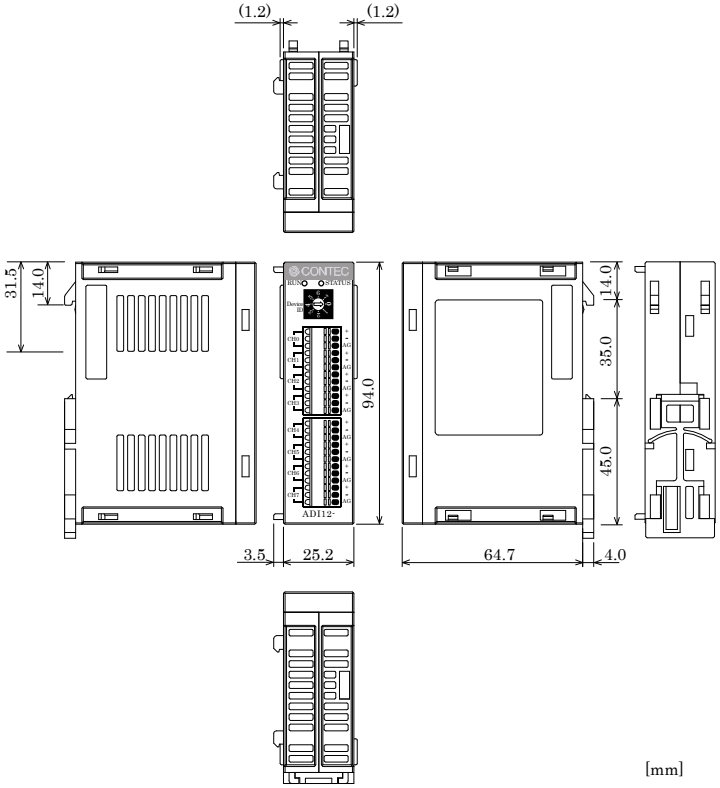



**Figure 6.2. External dimensions**

# ADI12-8(FIT)GY

## User's Manual

CONTEC CO., LTD.
May 2008 Edition
3-9-31, Himesato, Nishiyodogawa-ku, Osaka 555-0025, Japan
Japanese http://www.contec.co.jp/
English http://www.contec.com/
Chinese http://www.contec.com.cn/



[08062001]
[05092008_rev8]
Management No. A-40-606
Parts No. LZU3843