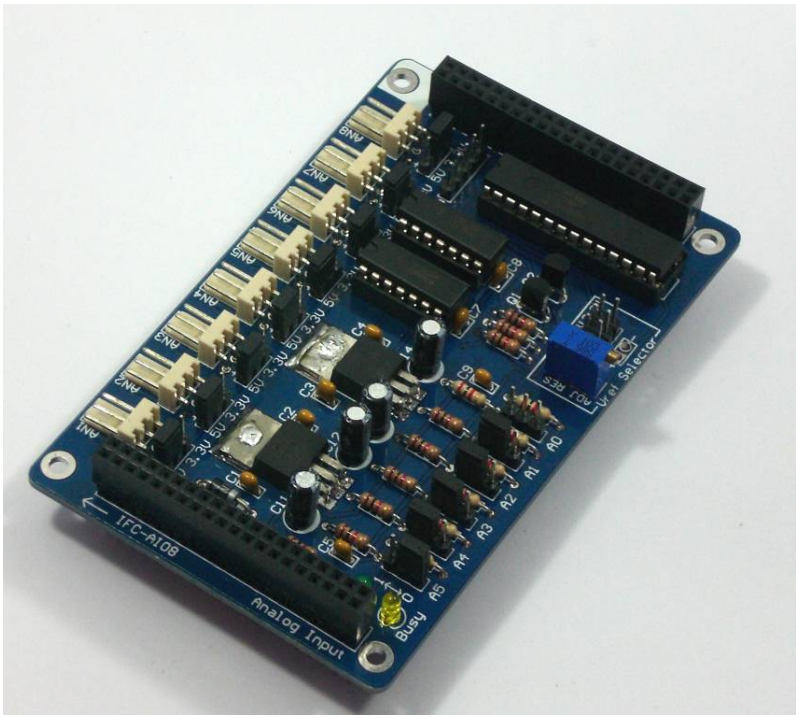
IFC-AI08
Interface Free Controller
Analog Input Card



Card Library Functions for Visual C# Express and Visual Basic Express

V1.0

April 2009

Function Prototype for Analog Input (AI08)

This document explains the function prototype for controlling IFC-AI08 using PC through IFC-CI00. User may also use 'object browser' under Microsoft Visual C# to view the summary, parameter and return value description of IFC-AI08 function prototype. User need to add reference 'ifc_ci.dll' and 'ifc_ai.dll' for IFC-CI00 and IFC-AI08 card in order to control/communicate IFC-AI08 using PC. Please note that before user start the programming, user need to initialize the 'ifc.ifc_ci' and 'ifc.ifc_ai' in order to use the functions to control IFC-AI08. Example of creating a 'ifc.ifc_ci' class called 'ifc1' and 'ifc.ifc_ai' class called ai1:

```
static ifc.ifc_ci ifc1 = new ifc.ifc_ci(74);
ifc.ifc_ai ai1 = new ifc.ifc_ai(ifc1, 4);
```

For 'ifc.ifc_ci' class, user need to specified the COM Port that is connected to IFC-CI00 and for 'ifc.ifc_ai' class, user need to specified the IFC-CI00 in use and also the address for IFC-AI08. Please make sure that the address must be unique and different with other IFC card in the IFC system.

Function Prototype	Example	Summary	Parameter Description	Return Value
void ai_bit_conf(byte *bit_num*)	**ai1.ai_bit_conf**(11110000)	To configure each ADC channel to either 8 bits or 10 bits mode.	*bit_num*: Every bit represents an ADC channel, bit 0 for AN1... and bit 7 for AN8. Value 0 for the representative bit will set the ADC channel to 8 bit mode, and value 1 will set the ADC channel to 10 bit mode. (byte)	
void ai_bit_conf(int *bit_num*)	**ai1.ai_bit_conf**(11000011)	To configure each ADC channel to either 8 bits or 10 bits mode.	*bit_num*: Every bit represent an ADC channel, bit 0 for AN1... and bit 7 for AN8. Value 0 for the representative bit will set the ADC channel to 8 bit mode, and value 1 will set the ADC channel to 10 bit mode. (int)	

ai_cmp2(int *selection1*, int *selection2*)	**ai1.ai_cmp2**(1, 2)	To compare 2 ADC channel conversion result.	*selection1*: First ADC channel to compare, in range of 1 to 8. (int) *selection2*: Second ADC channel to compare, in range of 1 to 8. (int)	Return 0 if selection1 = selection2, return 1 if selection1 > selection2, return 2 if selection2 > selection1. (byte)
ai_cmp2(byte *selection1*, byte *selection2*)	**ai1.ai_cmp2**(1, 2)	To compare 2 ADC channel conversion result.	*selection1*: First ADC channel to compare, in range of 1 to 8. (byte) *selection2*: Second ADC channel to compare, in range of 1 to 8. (byte)	Return 0 if selection1 = selection2, return 1 if selection1 > selection2, return 2 if selection2 > selection1. (byte)
ai_cmphl(int *selection*, bool *highest_lowest*)	**ai_cmphl**(00001111, true)	To compare the selected analog channel and figure out which channel has the highest or lowest conversion result.	*selection*: Every bit represent an ADC channel, bit 0 for AN1, ..., and bit 7 for AN8. Value 1 for the representative bit will determine that the ADC channel is being included in the comparison. (int) *highest_lowest*: True for highest value comparison and false for lowest value comparison. (bool)	Every bit represent an ADC channel, bit 0 for AN1... and bit 7 for AN8. Value 1 determine that the ADC channel has the highest or lowest value, depend on the type of comparison selected. (byte)
ai_cmphl(byte *selection*, bool *highest_lowest*)	**ai_cmphl**(11110000, true)	To compare the selected analog channel and figure out which channel has the highest or lowest conversion result.	*selection*: Every bit represent an ADC channel, bit 0 for AN1, ..., and bit 7 for AN8. Value 1 for the representative bit will determine that the ADC channel is being included in the comparison. (byte) *highest_lowest*: True for highest value comparison and false for lowest value comparison. (bool)	Every bit represent an ADC channel, bit 0 for AN1... and bit 7 for AN8. Value 1 determine that the ADC channel has the highest or lowest value, depend on the type of comparison selected. (byte)

ai_cmphl(int *selection*, int *highest_lowest*)	**ai1.ai_cmphl**(11111111, 0)	To compare the selected analog channel and figure out which channel has the highest or lowest conversion result.	*selection*: Every bit represent an ADC channel, bit 0 for AN1, ..., and bit 7 for AN8. Value 1 for the representative bit will determine that the ADC channel is being included in the comparison. (int) *highest_lowest*: 1 for highest value comparison and 0 for lowest value comparison. (int)	Every bit represent an ADC channel, bit 0 for AN1... and bit 7 for AN8. Value 1 determine that the ADC channel has the highest or lowest value, depend on the type of comparison selected. (byte)
ai_cmphl(byte *selection*, byte *highest_lowest*)	**ai1.ai_cmphl**(11110000, 1)	To compare the selected analog channel and figure out which channel has the highest or lowest conversion result.	*selection*: Every bit represent an ADC channel, bit 0 for AN1, ..., and bit 7 for AN8. Value 1 for the representative bit will determine that the ADC channel is being included in the comparison. (byte) *highest_lowest*: 1 for highest value comparison and 0 for lowest value comparison. (byte)	Every bit represent an ADC channel, bit 0 for AN1... and bit 7 for AN8. Value 1 determine that the ADC channel has the highest or lowest value, depend on the type of comparison selected. (byte)
ai_read(int *an_sel*)	**ai1.ai_read**(1)	To read conversion result of selected ADC channel.	*an_sel*: Selected channel to read, in range of 1 to 8. (int)	Conversion result of selected ADC channel. (int)
ai_read(byte *an_sel*)	**ai1.ai_read**(5)	To read conversion result of selected ADC channel.	*an_sel*: Selected channel to read, in range of 1 to 8. (byte)	Conversion result of selected ADC channel. (int)

void ai_sampling_conf(int *an_sel*, int *average*)	**ai1.ai_sampling_conf**(8, 2000)	To configure sampling rate for each ADC channel. All ADC value within the sampling period will be averaged.	*an_sel*: Selected channel to configure, in range of 1 to 8. (int) *average*: Determine sampling rate of selected ADC channel in range of 1 to 65535. Sampling rate = 10ms x average. (int)	
void ai_sampling_conf(byte *an_sel*, byte *average*)	**ai1.ai_sampling_conf**(1, 200)	To configure sampling rate for each ADC channel. All ADC value within the sampling period will be averaged.	*an_sel*: Selected channel to configure, in range of 1 to 8. (byte) *average*: Determine sampling rate of selected ADC channel in range of 1 to 65535. Sampling rate = 10ms x average. (byte)	
ai_vref_read()	**ai1.ai_vref_read**()	To read the adjusted Vref value through programming besides using multimeter. Please make sure that Vref is being set to ADJ on IFC-AI08.		Vref 10 bit result. Exact Voltage of Vref can be calculated using Vref = (result/1023) x 5V. (int)
ifc_ai(ifc.ifc_ci *ifc_ci*, int *address*)	**ifc.ifc_ai**(ci1, 3)	Initializes a new instance of the ifc.ifc_ai class using the specified ifc.ifc_ci and address for IFC-AI08.	*ifc_ci*: ifc.ifc_ci in use. *address*: Address for IFC-AI08, in range of 0 to 63. (int)	
ifc_ai(ifc.ifc_ci *ifc_ci*, byte *address*)	**ifc.ifc_ai**(ci1, 7)	Initializes a new instance of the ifc.ifc_ai class using the specified ifc.ifc_ci and address for IFC-AI08.	*ifc_ci*: ifc.ifc_ci in use. *address*: Address for IFC-AI08, in range of 0 to 63. (byte)	

Table 1 Function Prototype for Analog Input (AI08)

Prepared by
Cytron Technologies Sdn. Bhd.
19, Jalan Kebudayaan 1A,
Taman Universiti,
81300 Skudai,
Johor, Malaysia.

Tel: *+607-521 3178*
Fax: *+607-521 1861*

URL: *www.cytron.com.my*
Email: *support@cytron.com.my*
sales@cytron.com.my