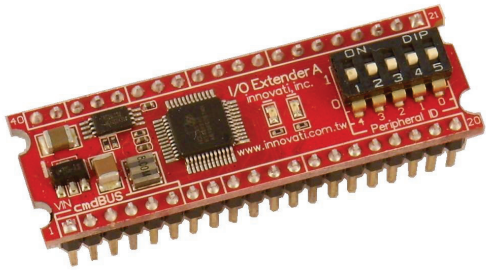
I/O Extender A Module User's Guide

Version: V1.1



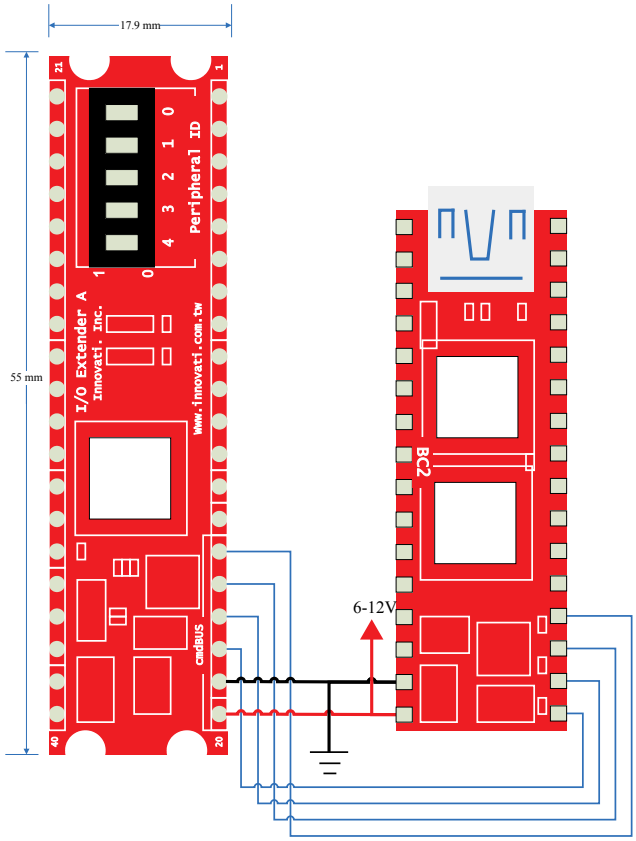
Product Overview: Using Innovati's I/O Extender A Module, users can obtain 3 additional ports (total of 24 pins) in addition to those already in the BASIC Commander. Each additional pin will have a function and performance similar to the BASIC Commander pins and can use similar commands with the proper module name. In addition, additional functions are also available, such as for event detection of pin signal variations, input pulse counting etc. Please use "**IOExtenderA**" as the module object name in program.

Application:

- Add the functions to independently control the display status of several indicators at the same time.
- Provide a detection function for signal variations on each pin.
- With the A/D converter pins, an analog signal can be converted into a digital signal for follow-up processing.

Product Features:

- There are 3 ports, each with 8 pins, giving a total of 24 pins, providing the capability for extended applications.
- Provides independent signal width measurement and signal pulse counting. Measurement accuracy can be down to microseconds (μs).
- Provides an independent variable frequency output.
- 8-channel analog to digital (A/D) measurement inputs.
- Provides high voltage event detection on 8 pins and low voltage event detection on another 8 pins. It is possible to detect the voltage variation events on 16 pins at the same time.



Connection: Directly setups the ID switches to the required number, and then connect the cmdBUS cable to the corresponding pins on the BASIC Commander (shown in the following figure). Then the required operations can be performed through the BASIC Commander. DC power (6~12V) and ground should be connected to VIN and GND pin.

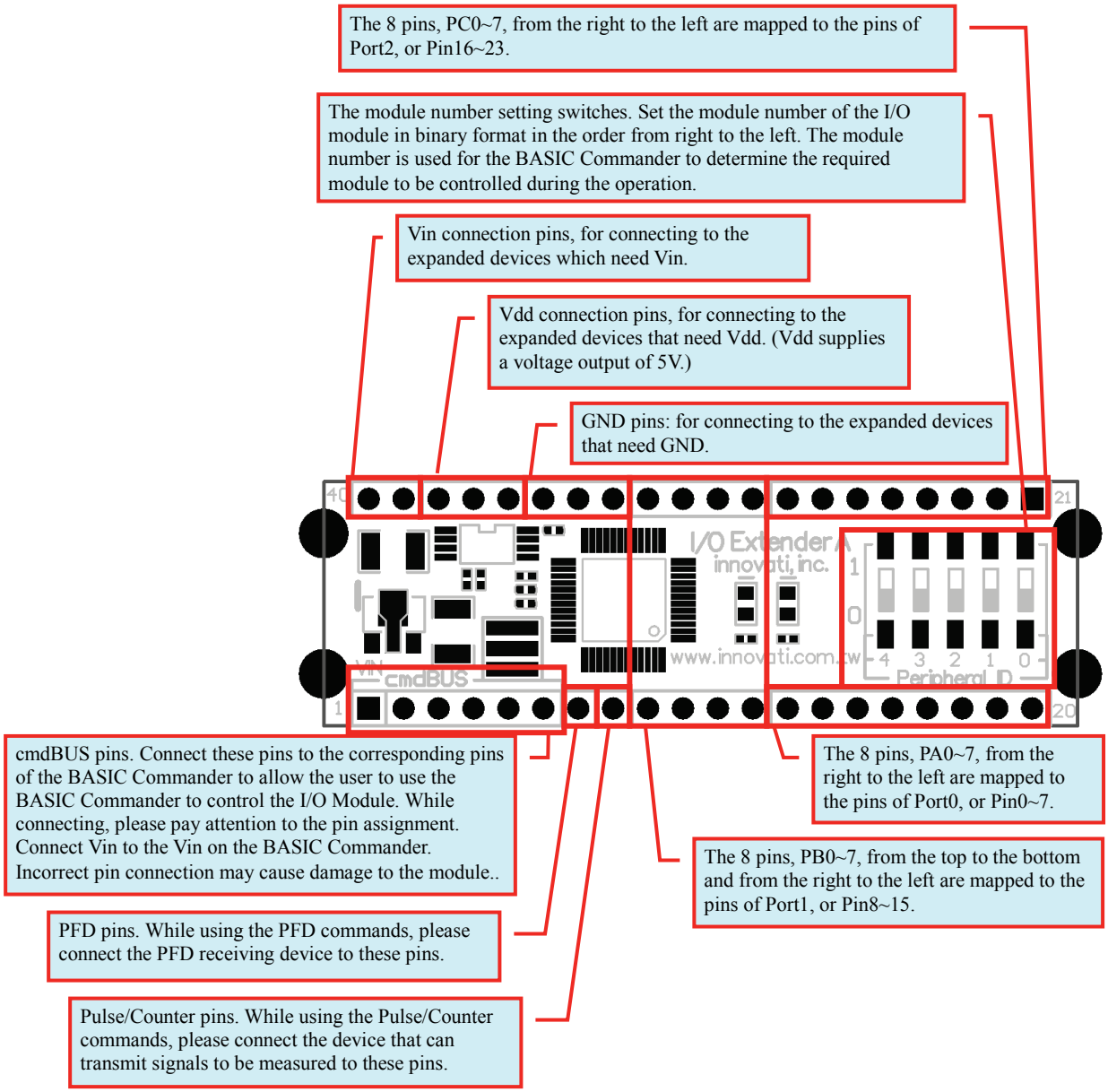


Figure 1: Module pin assignment and switches

Electronic Characteristics

Symbol	Parameter	Test Conditions (V_{IN}=7.5V)	Min.	Typ.	Max.	Unit
I_{DD}	Operating Current	No I/O	—	4	—	mA
V_{IL}	Input Low Voltage for I/O Ports.	—	0	—	1.5	V
V_{IH}	Input High Voltage for I/O Ports.	—	3.5	—	5	V

V_{OH}	I/O Port output high voltage	No load	—	5	—	V
V_{OL}	I/O Port output low voltage	No load	—	0	—	V
I_{OL}	I/O Port Sink Current	V_{load}=0.1V_{OH}	10	20	—	mA
I_{OH}	I/O Port Source Current	V_{load}=0.9V_{OH}	-5	-10	—	mA
V_{AD}	A/D Input Voltage	—	0	—	5	V
I_{ADC}	Additional Power Consumption if A/D Converter is Used	—	—	1.5	3	mA

Table 1: Electronic Characteristics (ambient temperature 25 °C)

Absolute Maximum Ratings:

Operating Temperature : 0°C ~ 70°C

Storage Temperature : -50°C ~ 125°C

Commands and Events:

The following tables list all the unique commands and events provided with the I/O Extender A module. Note that essential words in the commands will be written in **bold** type and ***italics*** in bold type. The bold type word must be written exactly as shown, whereas the italic bold type words must be replaced with the user values. Note that the innoBASIC language is case-insensitive.

Command Format	Description
I/O Port Read/Write Commands	
High(*Pin*)	Output a high voltage on the pin specified by the byte variable ***Pin*** ranging from 0 to 23. (See Note 1.)
In(*Pin*, *Value*)	Get the voltage on the pin specified by the byte variable ***Pin*** ranging from 0 to 23 and store it in the byte variable ***Value***. (See Note 2.)
Low(*Pin*)	Output a low voltage on the pin specified by the byte variable ***Pin*** ranging from 0 to 23. (See Note 1.)
PulseOut(***Pin***, ***Mode***, ***Width***)	Output a pulse with a time interval specified by the word variable ***Width*** in millisecond (ms) on the pin specified by the byte variable ***Pin*** ranging from 0 to 23. If ***Mode*** is 0, output a low voltage pulse. If ***Mode*** is 1, output a high voltage pulse. (See Note

	3.)
ReadPort(*Port*, *Value*)	Read the port specified by the byte variable ***Port*** and then store the result in the byte variable ***Value***. (See Note 4.)
ReadPort0(*Value*) **ReadPort1(*Value*)** **ReadPort2(*Value*)**	Read the port 0, 1 or 2 and store the result in the byte variable ***Value***. (See Note 4.)
Toggle(*Pin*)	Invert the output voltage on the pin specified by the byte variable ***Pin*** ranging from 0 to 23. (See Note 1.)
TogglePort0() **TogglePort1()** **TogglePort2()**	Invert the output levels on each pin of port 0, 1 or 2. (See Note 4.)
WritePort(*Port*, *Value*)	Write the byte variable ***Value*** to the pins of the port specified by the byte variable ***Port***. (See Note 4.)
WritePort0(*Value*) **WritePort1(*Value*)** **WritePort2(*Value*)**	Write the byte variable ***Value*** to the pins of port 0, 1 or 2. (See Note 4.)
I/O Port Direction Setting Commands	
GetDirPort(*Port*, *Result*)	Get the I/O modes of the port specified by the byte variable ***Port*** and store the value in the byte variable ***Result***. Each bit with value 0 represents the output mode and 1 represents the input mode.
GetDirPort0(*Result*) **GetDirPort1(*Result*)** **GetDirPort2(*Result*)**	Get the I/O modes of the port 0, 1 or 2 and store the result in the byte variable ***Result***. Each bit with value 0 represents the output mode and 1 represents the input mode.
SetDirPort(*Port*, *Dir*)	Set the port specified by the byte variable ***Port*** with the I/O directions specified by the byte variable ***Dir***.
SetDirPort0(*Dir*) **SetDirPort1(*Dir*)** **SetDirPort2(*Dir*)**	Set the port 0, 1 or 2 with the I/O directions specified by the byte variable ***Dir***.
Event Enabling/Disabling Commands	
DisablePinLowEvent1() : **DisablePinLowEvent8()**	Disable the pin low event 1 to 8.
DisablePinHighEvent1() : **DisablePinHighEvent8()**	Disable the pin high event 1 to 8.
DisablePort0ChangeEvent() **DisablePort1ChangeEvent()** **DisablePort2ChangeEvent()**	Disable port 0, 1 or 2 level change event.

DisablePort0MatchEvent() **DisablePort1MatchEvent()** **DisablePort2MatchEvent()**	Disable port 0, 1 or 2 match event.
DisablePort0LowEvent() **DisablePort1LowEvent()** **DisablePort2LowEvent()**	Disable port 0, 1 or 2 low event.
DisablePort0HighEvent() **DisablePort1HighEvent()** **DisablePort2HighEvent()**	Disable port 0, 1 or 2 high event.
EnablePinLowEvent1(*Pin*, *Repeat*) : **EnablePinLowEvent8(*Pin*, *Repeat*)**	Enable the pin low event 1 to 8 for pin specified by the byte variable ***Pin*** ranging from 0 to 23 with the repeat event triggering time specified by the word variable ***Repeat*** in milliseconds (ms). If the ***Repeat*** value is 0, the event will be triggered once.
EnablePinHighEvent1(*Pin*, *Repeat*) : **EnablePinHighEvent8(*Pin*, *Repeat*)**	Enable the pin high event 1 to 8 for pin specified by the byte variable ***Pin*** ranging from 0 to 23 with the repeat event triggering time specified by the word variable ***Repeat*** in milliseconds (ms). If the ***Repeat*** value is 0, the event will be triggered once.
EnablePort0ChangeEvent(*BitMask*) **EnablePort1ChangeEvent(*BitMask*)** **EnablePort2ChangeEvent(*BitMask*)**	Enable the port change event specified by "1" bit(s) in the byte variable ***BitMask.*** If voltage level change from either low to high or high to low on the ***BitMask*** specified pin(s) of port 0, 1 or 2 is detected, the corresponding event will be activated.
EnablePort0MatchEvent(*Value*, *Repeat*) **EnablePort1MatchEvent(*Value*, *Repeat*)** **EnablePort2MatchEvent(*Value*, *Repeat*)**	Enable the port 0, 1 or 2 match event specified by the byte variable ***Value*** with the repeat event triggering time specified by the word variable ***Repeat*** in milliseconds (ms). If the ***Repeat*** value is 0, the event will be triggered once.
EnablePort0LowEvent(*BitMask*) **EnablePort1LowEvent(*BitMask*)** **EnablePort2LowEvent(*BitMask*)**	Enable the port low event specified by "1" bit(s) in the byte variable ***BitMask.*** If low voltage on the ***BitMask*** specified pin(s) of port 0, 1 or 2 is detected, the corresponding event will be activated.
EnablePort0HighEvent(*BitMask*) **EnablePort1HighEvent(*BitMask*)** **EnablePort2HighEvent(*BitMask*)**	Enable the port high event specified by "1" bit(s) in the byte variable ***BitMask.*** If high voltage on the ***BitMask*** specified pin(s) of port 0, 1 or 2 is detected, the corresponding event will be activated.

***Status*=GetPort0LowEventStatus()** ***Status*=GetPort1LowEventStatus()** ***Status*=GetPort2LowEventStatus()**	Get the port 0, 1 or 2 low event status and store the result in the byte variable ***Status***. A "1" bit in the ***Status*** indicates a logic low level on the corresponding pin(s) has been detected. After execution of this command, the status value will be cleared to 0 automatically.
***Status*=GetPort0HighEventStatus()** ***Status*=GetPort1HighEventStatus()** ***Status*=GetPort2HighEventStatus()**	Get the port 0, 1 or 2 high event status and store the result in the byte variable ***Status***. A "1" bit in the ***Status*** indicates a logic high level on the corresponding pin(s) has been detected. After execution of this command, the status value will be cleared to 0 automatically.
***Status*=GetPort0ChangeEventStatus()** ***Status*=GetPort1ChangeEventStatus()** ***Status*=GetPort2ChangeEventStatus()**	Get the port 0, 1 or 2 change event status and store the result in the byte variable ***Status***. A "1" bit in the ***Status*** indicates a bit logic level change on the corresponding pin(s) has been detected. After execution of this command, the status value will be cleared to 0 automatically.
PFD (Programmable Frequency Divider) Commands	
PFDOff()	Disable PFD output.
PFDOn()	Enable PFD output.
SetPFD(*Freq*)	Set the output frequency on PFD pin by a Word variable ***Freq***. (See Note 5.)
Pulse/Counter Commands	
CounterOff()	Stop the counter
CounterOn(*Mode*, *Value*, *EventEn*)	According to the value of the byte variable ***Mode***, there are four different modes described as follows: 0 : Within the time specified by the word variable ***Value*** (in millisecond, ms), count the low to high voltage level transients on the counter pin. 1 : Within the time specified by the word variable ***Value*** (in millisecond, ms), count the high to low voltage transients on the counter pin. 2 : Count the number of low to high voltage transients on the counter pin. When the number reaches the value specified by the word variable ***Value***, the counter stops counting. 3 : Count the number of high to low voltage transients on the counter pin. When the number reaches the value specified by the word variable ***Value***, the counter stops counting.

	When the condition of each mode is encountered, if the value of the byte variable ***EventEn*** is set as 1, the event **CountStopEvent** will be activated. (See Note 6 and 7.)
***Status*=GetCounter(*Count*)**	Get the status of the current counter saved in the byte variable ***Status*** and the counted value saved in the word variable ***Count***. If the value of ***Status*** is 1, it means that the counter has stopped, otherwise the counter is still in counting.
***Status*=GetPulseWidth(*Width*)**	Get the status of the pulse measurement. If the value of the byte variable ***Status*** is 1, it means that the measurement stops, and the measured value of the pulse is stored in the word variable ***Width*** in microseconds (μs) , otherwise the counter is still in measuring.
PulseInOff()	Stop the pulse measurement.
PulseInOn(*Mode*, *EventEn*)	To measure the input pulse. If the byte variable ***Mode*** is 0, the low pulse will be measured, 1, the high pulse will be measured. When the pulse measurement is complete, if the value of the byte variable ***EventEn*** is set as 1, the **PulseMeasureEndEvent** will be activated. If the pulse is longer than 65535 microseconds (μs), the **PulseOverflowEvent** will be activated and the counter will overflow to 0 and continue counting. (See Note 7.)
ADC (Analog to Digital Converter) Commands	
GetADC (*ChanNum*,*Value*)	Get the ADC value stored it in the word variable ***Value*** ranging from 0 to 1023 of the channel specified by the byte variable ***ChanNum***.
SetADC (*ChanNums*)	Set the ADC channels by the byte variable ***ChanNums*** as follows: 0 : CH0~CH7 Disabled 1 : CH0 Enabled 2 : CH0~CH1 Enabled 3 : CH0~CH2 Enabled 4 : CH0~CH3 Enabled 5 : CH0~CH4 Enabled 6 : CH0~CH5 Enabled 7～8 : CH0~CH7 Enabled Note that select 7 or 8 has the same effect of enabling all CH0~CH7 channels. CH0~CH7 are mapped to PA0~PA7, respectively.

Timer Commands	
GetTimer(*Timer*)	Read the value of the timer and store in a word variable ***Timer***. Note that the value of the timer will be reset to 0 once the timer is started.)
SetTimer(*Mode*, *Timer*)	Set the mode of the timer specified by the byte variable ***Mode*** with value 1 for repetition mode or value 0 for single mode. Set the time interval to be counted by a Word variable ***Timer*** in millisecond (ms).
TimerOff	Stop the timer.
TimerOn	Start the timer.

Note 1:

This command will set the pin to be in the output mode.

Note 2:

This command will set the pin to be in the input mode.

Note 3:

If the original output voltage is already high, when the request for to output a high voltage pulse is made, the module will send out a short low voltage signal in advance and then change to a high voltage.

Note 4:

This command will not change the I/O mode of the pin. Use the SetDirPort or related commands to change the I/O mode.

Note 5:

After the new PFD value is set, it is necessary to execute PFDOn() one more time to generate the required signal.

Note 6:

In mode 0 and 1, if the counter has reached 65535 before the specified time has elapsed the counter will automatically stop and the event CountStopEvent will be activated at the same time. In mode 2 and 3, if the target value is set as 0, when the counter has reached 65535, the counter stops and the event CountStopEvent will be activated at the same time.

Note 7:

This function will take effect at least 200 μs after this command is executed.

Event	Description
CounterStopEvent()	**CounterOn** command is executed with the event enabled, and the specified condition encountered.
PinLowEvent1() **:** **PinLowEvent 8()**	After the corresponding event enabling command is executed, a low voltage on the specified pin is detected, the corresponding event will be triggered.
PinHighEvent1() **:**	After the corresponding event enabling command is executed, a high voltage on the specified pin is detected, the corresponding

PinLowEvent 8()	event will be triggered.
Port0ChangeEvent() **Port1ChangeEvent()** **Port2ChangeEvent()**	After the corresponding event enabling command is executed, a voltage change on the specified pin(s) of the port is detected, the corresponding event will be triggered.
Port0HighEvent() **Port1HighEvent()** **Port2HighEvent()**	After the corresponding event enabling command is executed, a high voltage on the specified pin(s) of the port is detected, the corresponding event will be triggered.
Port0LowEvent() **Port1LowEvent()** **Port2LowEvent()**	After the corresponding event enabling command is executed, a low voltage on the specified pin(s) of the port is detected, the corresponding event will be triggered.
Port0MatchEvent() **Port1MatchEvent()** **Port2MatchEvent()**	After the corresponding event enabling command is executed, a matched value on the specified pins of the ports are detected, the corresponding event will be triggered.
PortNumberErrorEvent()	If the pin number (0~23) or the port number (0~2) given exceeds the allowed range, the corresponding event will be triggered.
PulseInOverflowEvent()	If the **PulseInOn** command is executed and the detected pulse is longer than 65535 microsecond (μs), the corresponding event will be triggered.
PulseMeasureEndEvent	If the **PulseInOn** is executed and the detected signal voltage changes from the voltage value to be detected, the corresponding Event will be triggered.
TimerOverFlowEvent()	If the **TimerOn** command is executed and the specified time has elapsed, the corresponding event will be triggered.

Example Program:

```
Peripheral myIO As IOExtenderA @ 0          'Set module number to 0

Dim PinLevel As Byte                        'Store pin status
Dim CountStop As Byte                       'Store counter - Stop the status
Dim ChValue As Word                         'Store the analog to digital converted value
Dim TimeUp As Byte                          'Store the status when time is up

Sub Main()
myIO.High(0)                                'Output a high voltage on Pin 0
myIO.Low(1)                                 'Output a low voltage on Pin 1
myIO.In(2, PinLevel)                        'Get the input voltage on Pin 2
myIO.PulseOut(3, 1, 2)                      'Output 2ms high voltage waveform on Pin 3
myIO.Toggle(0)                              'Invert the output voltage on Pin 0 (from original high voltage into low voltage)

myIO.SetDirPort1(0)                         'Set all the pins of Port 1 to be used as outputs
```

```
myIO.EnablePinLowEvent1(16, 0)                'Detect a low going transition on Pin 16
myIO.DisablePinLowEvent1()                    'Disable the low voltage detection

myIO.SetPFD(1000)                             'Set PFD to output 1k Hz waveform
myIO.PFDOn()                                  'Enable PFD output
myIO.PFDOff()                                 'Disable PFD output

CountStop=0
myIO.CounterOn(0, 2000, 1)                    'Count the number of positive edges on the Counter Pin within 2 seconds

'       The following loop will be exited only when the counter stops
Do
Loop Until CountStop>0

myIO.CounterOff()                             'Disable the counter

myIO.SetADC (1)                               'Enable PA0 as the input for ADC conversion
myIO.GetADC (0, ChValue)                      'Get the converted digital value from the analog input on CH0 and store in ChValue

myIO.SetTimer(0, 1000)                        'Set the timer to count down for 1 second
TimeUp=0
myIO.TimerOn()                                'Activate the timer

' The following loop will be exited only when the timer is up
Do
Loop Until TimeUp>0

myIO.TimerOff()                               'Disable the timer
End Sub

Event MyIO.PinLowEvent1()
Debug “Appear when the low voltage is detected!”, CR
End Event

Event MyIO.CounterStopEvent()
Debug “Counter stops!”, CR
CountStop=1
End Event

Event MyIO.TimerOverflowEvent()
Debug “Timer is up!”, CR
TimeUp=1
End Event
```

Appendix

1. Known Problems:

None

2. Module ID Setting Table:

	0		8		16		24
	1		9		17		25
	2		10		18		26
	3		11		19		27
	4		12		20		28
	5		13		21		29
	6		14		22		30
	7		15		23		31