Time Keeper A Module User's Guide

Version: V1.0

Product Overview: Innovati's Time Keeper A Module is designed to provide versatile time and date related features. As it contains a fully integrated weekday mapping function, when the present day is inputted, the corresponding weekday can be determined automatically. In addition, it provides a secondary time function that can be setup by the user, and 8 additional countdown timers that can meet the requirements of the user for multiple timer functions. A calibration function is provided, which can help control the time error of each day to be within 0.08 seconds. Please use "**TimeKeeperA**" as the module object name in program.

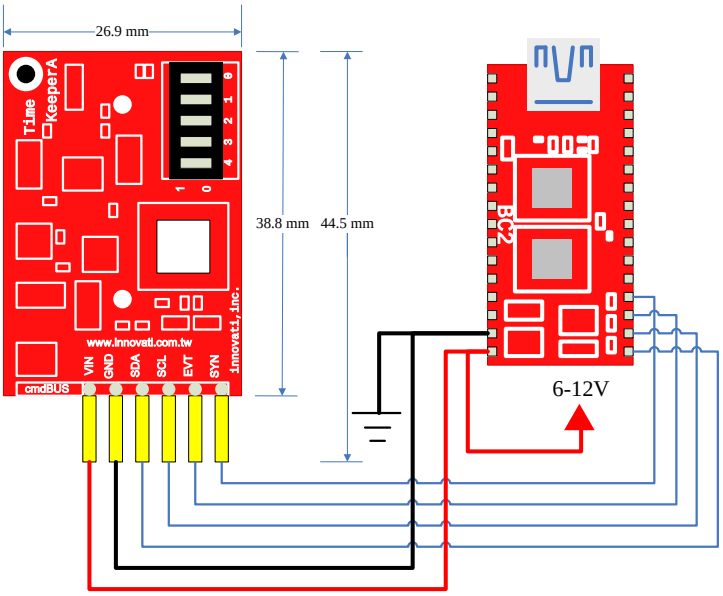
Application:

- It can be used together with the LCD Module for displaying the time as a simple electronic clock.
- It can work with other of modules as the scheduler.
- It can be easily used for calendar applications and provide various notifications for versatile schedules.
- It can be connected to switches and used as regular timers for various appliances or to perform scheduled activations.

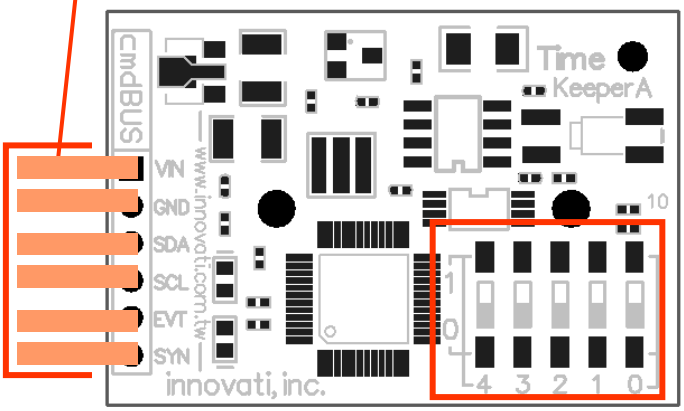
Product Features:

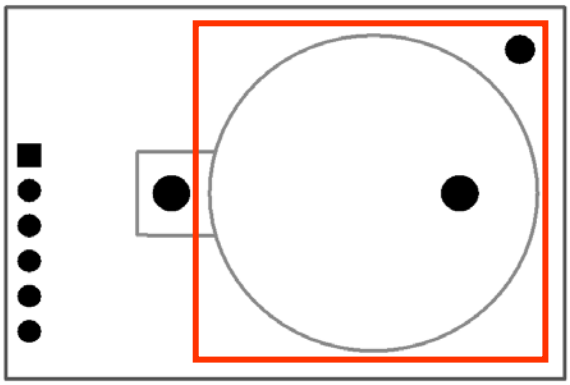
- The year, month, date, weekday, hour, minute, second within AD 2000~2099 can be automatically counted.
- Provide the time display in both 24- and 12-hour format.
- Five subsidiary timers are provided for setting the hour, minute and second.
- Versatile notifications, which allows the modes such as every second, every minute, every hour, every day, every week, every month, etc., and several modes to exist at the same time.
- The 8 timers allow the user to set count down operations with the value range from the unit of days to the unit of seconds.
- The accuracy can be finely adjusted to reduce the time error. The most accurate condition can be as low as 3.052ppm.

Connection: Directly setup 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.



cmdBUS pin. Connect these pins to the corresponding pins on the BASIC Commander. Then the Timer module can be controlled through the BASIC Commander. While connecting the pin, connect Vin to the Vin pin on the BASIC Commander. If the pins are incorrectly connected, the module may be damaged.





Product Specifications:
Operating current : Approximately 9 mA

Precautions for Operations:
Please use CR2032 type batteries. When replacing with other batteries, please check their operating precautions.

Absolute Maximum Ratings:
Operating Temperature : 0°C~70°C (excluding the batteries)
Storage Temperature : -50 °C~125°C

Commands and Events:
The following tables list all the unique commands and events provided with the Time Keeper 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
Time and Date Setting Commands	
Set12Hour(*Hour, AMPM*)	Set the hour of the Time Keeper module specified by the byte value ***Hour***, ranging from 1 to 12.The byte value ***AMPM*** with value 0 represents AM and 1 represents PM.
SetDate(*Date*)	Set the date of the Time Keeper module specified by the byte value ***Date*** ranging from 1 to 31. If the date that does not exist in the corresponding month is set, the day will automatically be set to 1.
SetHour(*Hour*)	Set the hour of the Time Keeper module specified by the byte value ***Hour*** ranging from 0 to 11 for 12-hour format and 0 to 23 for the 24-hour format.
SetMinute(*Minute*)	Set the minute of the Time Keeper module specified by the byte value ***Minute*** ranging from 0 to 59.
SetMonth(*Month*)	Set the month of the Time Keeper module specified by the byte value ***Month*** ranging from 1 to 12.
SetTimeAndDate (*Year, Month, Date, Hour, Minute, Second*)	According to the current date and time of the Time Keeper module by the byte values ***Year***, ***Month***, ***Day***, ***Hour***, ***Minute*** and ***Second.*** The input range of ***Year*** is 0~99, which represents AD 2000~2099.
SetSecond(*Second*)	Set the second of the Time Keeper module specified by the byte value ***Second*** ranging from 0 to 59.
SetSubTime(*SubID, Hour, Minute, Second*)	Set the subsidiary timer specified by ***SubID,*** ranging from 0 to 4, for setting the hour, minute, second by the byte variables ***Hour***, ***Minute*** and ***Second***.

SetYear(*Year*)	Set the year of the Time Keeper module by the byte variable ***Year***, ranging from 0~99, which represents AD 2000~2099.
Time Reading Commands	
Get12Hour(*Hour, AMPM*)	Get the current time in the 12-hour format and store the hour value in the byte variable ***Hour***, and represent the morning or afternoon by ***AMPM***. If ***AMPM*** has value 0 represents morning, and 1 represents afternoon.
GetDate(*Date*)	Get the date and store it in the byte variable ***Date***.
GetHour(*Hour*)	Get the hour and store it in the byte variable ***Hour***.
GetMinute(*Minute*)	Get the minute and store it in the byte variable ***Minute***.
GetMonth(*Month*)	Get the month and store it in the byte variable ***Month***.
GetTimeAndDate(*Year, Month, Date, Weekday, Hour, Minute, Second*)	Get the current time and store the values of the year, month, day, weekday, hour, minute, and second in the byte variables ***Year***, ***Month***, ***Date***, ***Weekday***, ***Hour***, ***Minute*** and ***Second***, respectively. The weekday is stored in a way that Sunday is represented by 0 and Monday through Saturday are represented by 1~6.
GetSecond(*Second*)	Get the current time and store the value of second in the byte variable ***Second***.
GetSubTime(*ID, Hour, Minute, Second*)	Get the subsidiary timer specified by ***ID***, ranging from 0 to 4, and then store the values of the hour, minute and second in the byte variables ***Hour***, ***Minute*** and ***Second***.
GetWeekDay(*Weekday*)	Get the current weekday and store in the byte variable ***Weekday***.
GetYear(*Year*)	Get the current year and store in the byte variable ***Year***.
Event Related Commands	
DailyAlarmOn(*AlarmID*)	Enable the daily alarm specified by the byte value ***AlarmID***, ranging from 0 to 7.
DailyAlarmOff(*AlarmID*)	Disable the daily alarm specified by the byte value ***AlarmID***, ranging from 0 to 7.
DisableDailyEvent()	Disable the event **DailyEvent**.
DisableHourlyEvent()	Disable the event **HourlyEvent**.
DisableMinutelyEvent()	Disable the event **MinutelyEvent**.
DisableSecondlyEvent()	Disable the event **SecondlyEvent**.
EnableDailyEvent()	Enable the event **DailyEvent**.
EnableHourlyEvent()	Enable the event **HourlyEvent**.
EnableMinutelyEvent()	Enable the event **MinutelyEvent**.
EnableSecondlyEvent()	Enable the event **SecondlyEvent**.
GetDailyAlarm(*AlarmID, Hour, Minute*)	Get the settings of the daily alarm specified by the byte value ***AlarmID***, ranging from 0 to 7. Store the values of the hour and minute in the byte variables ***Hour*** and

	Minute.
GetHourlyAlarm(*AlarmID*, *Minute*)	Get the settings of the hourly alarm specified by ***AlarmID***, ranging from 0 to 7. The value of minute is stored in the byte variable ***Minute***.
GetMonthAlarm(*AlarmID*, *Date*, *Hour*, *Minute*)	Get the settings of the monthly alarm specified by the byte value ***AlarmID***, ranging from 0 to 7. The values of the date, hour, and minute are stored in the byte variables ***Date***, ***Hour*** and ***Minute***.
GetWeeklyAlarm(*AlarmID*, *Weekday*, *Hour*, *Minute*)	Get the settings of the weekly alarm specified by ***AlarmID***, ranging from 0 to 7. The values of the weekday, hour and minute are stored in the byte variables ***Weekday***, ***Hour*** and ***Minute***.
HourlyAlarmOff(*AlarmID*)	Disable the hourly alarm specified by the byte variable ***AlarmID***, ranging from 0 to 7.
HourlyAlarmOn(*AlarmID*)	Activate the hourly alarm specified by the byte variable ***AlarmID***, ranging from 0~7.
MonthlyAlarmOff(*AlarmID*)	Disable the monthly alarm specified by the byte variable ***AlarmID***, ranging from 0~7.
MonthlyAlarmOn(*AlarmID*)	Activate the monthly alarm specified by the byte variable ***AlarmID***, ranging from 0~7.
SetDailyAlarm(*AlarmID*, *Hour*, *Minute*)	Set the daily alarm specified by the byte value ***AlarmID***, ranging from 0 to 7, by the byte values ***Hour*** and ***Minute***.
SetHourlyAlarm(*AlarmID*, *Minute*)	Set the the hourly alarm specified by the byte value ***AlarmID***, ranging from 0 to 7, by the byte value ***Minute***.
SetMonthlyAlarm(*AlarmID*, *Date*, *Hour*, *Minute*)	Set the monthly alarm specified by the byte value ***AlarmID***, ranging from 0 to 7, by the byte value ***Date***, ***Hour*** and ***Minute***.
SetWeeklyAlarm(*AlarmID*, *Weekday*, *Hour*, *Minute*)	Set the weekly alarm specified by the byte value ***AlarmID***, ranging from 0 to 7, by the byte value ***Weekday***, ***Hour*** and ***Minute***.
WeeklyAlarmOff(*AlarmID*)	Disable the weekly alarm specified by ***AlarmID***, ranging from 0 to 7.
WeeklyAlarmOn(*AlarmID*)	Activate the weekly alarm specified by ***AlarmID***, ranging from 0 to 7.
Timer Commands	
CountDownTimerOn(*TimerID*)	Activate the timer specified by ***TimerID***, ranging from 0 to 7.
CountDownTimerOff(*TimerID*)	Disable the timer specified by ***TimerID***, ranging from 0 to 7.
GetCountDownTimer(*TimerID*, *Day*, *Hour*, *Minute*, *Second*)	Get the time of the count-down timer specified by ***TimerID***, ranging from 0 to 7. The remaining days, hours, minutes and seconds are stored in ***Day***, ***Hour***, ***Minute*** and ***Second***.

SetCountDownTimer(*TimerID, Day, Hour, Minute, Second*)	Set the time of the count-down timer specified by ***TimerID***, ranging from 0 to 7. The remaining days, hours, minutes and seconds are set by the byte value ***Day***, ***Hour***, ***Minute*** and ***Second***.
Calibration and Reset Commands	
ResetTimeAndDate()	Reset the Time Keeper module to its default value.
GetAdjustment(*AdjValue*)	Get the fine adjustment value and store it in the variable ***AdjValue***.
SetClockAdj(*AdjValue*)	Set the fine adjustment value by the value ***AdjValue***. Refer to Appendix 3 for the detailed settings.

Event	Description
Count-down Timer Events	
CountDownTimer0Event : **CountDownTimer7Event**	After **CountDownTimerOn(*TimerID*)** command is executed, when the timer counts down to 0, the corresponding event will be triggered.
Alarm Events	
MonthlyAlarm0Event : **MonthlyAlarm7Event**	After **MonthlyAlarmOn(*AlarmID*)** command is executed, when the time reaches the preset date, hour and minute, the corresponding event will be triggered monthly.
WeeklyAlarm0Event : **WeeklyAlarm7Event**	After **WeeklyAlarmOn(*AlarmID*)** command is executed, when the time reaches the preset weekday, hour and minutes, the corresponding event will be triggered weekly.
DailyAlarm0Event : **DailyAlarm7Event**	After **DailyAlarmOn(*AlarmID*)** command is executed, when the time reaches the preset hour and minute, the corresponding event will be triggered daily.
HourlyAlarm0Event : **HourlyAlarm7Event**	After **HourlyAlarmOn(*AlarmID*)** command is executed, when the time reaches the preset minute, the corresponding event will be triggered hourly.
Periodical Events	
DailyEvent	After **EnableDailyEvent** command is executed, when the value of date is changed, the **DailyEvent** will be triggered. In other words, the event will be triggered every midnight.
HourlyEvent	After **EnableHourlyEvent** command is executed, when the value of hour is changed, the **HourlyEvent** will be triggered. In other words, the event will be triggered every sharp hour.
MinutelyEvent	After the **EnableMinutelyEvent** command is executed, when the value of minute is changed, the **MinutelyEvent** will be triggered. In other words, the event will be triggered every sharp minute.
SecondlyEvent	After **EnableSecondlyEvent** command is executed, when the value of second is changed, the **SecondlyEvent** will be

	triggered. In other words, the event will be triggered every sharp second.

Example Program:

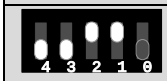
```
Peripheral MyTime As TimeKeeperA @ 0  '  Set module number is 0

Dim CurYear As Byte                    ' Store the current value of year
Dim CurMonth As Byte                   ' Store the current value of month
Dim CurDay As Byte                     ' Store the current value of day
Dim CurWeek As Byte                    ' Store the current value of weekday
Dim CurHour As Byte                    ' Store the current value of hour
Dim CurMinute As Byte                  ' Store the current value of minute
Dim CurSecond As Byte                  ' Store the current value of second
Dim SecondCnt As Byte                  ' Count the display time

Sub Main()                             ' Main program
    MyTime.SetTimeAndDate(7, 9, 17, 15, 47, 0)       ' Set the current time
    SecondCnt=0
    MyTime.EnableSecondlyEvent()    ' Enable the second event to be activated every second

' The following loop will be exited after time displayed at least 5 times.
    Do
    Loop Until SecondCnt>5

    MyTime.DisableSecondlyEvent()   ' Cancel the second event
    MyTime.SetMinute(48)            ' Set the minute to 48
    MyTime.SetSecond(55)            ' Set the second to 55
    MyTime.SetHourlyAlarm(0, 49)    ' Set Alarm #0 with a notification at 49 minutes in every hour
    SecondCnt=0
    MyTime.HourlyAlarmOn(0)         ' Activate Alarm #0 for hourly notifications.

' The following loop will be exited only when the hourly notification is activated.
    Do
    Loop Until SecondCnt>0

    MyTime.HourlyAlarmOff(0)        ' Disable the hourly notification of Alarm 0
    MyTime.SetCountDownTimer(0, 0, 0, 0, 3)   ' Set Timer #0 for the count down of 3 seconds
    SecondCnt=0
    MyTime.CountDownTimerOn(0)      ' Activate Timer #0 to execute the count down operation

' The following loop will be exited only when the count down operation is completed
    Do
    Loop Until SecondCnt>0
```

```
    MyTime.CountDownTimerOff(0)     ' Disable Timer #0
End Sub

Event MyTime.SecondlyEvent()
    MyTime.GetTimeAndDate(CurYear,CurMonth,CurDay,CurWeek,CurHour,CurMinute,CurSecond)
                                                                        ' Get the current time
    Debug CurYear, “/”, CurMonth, “/”, CurDay, “ “, CurHour, “:”, CurMinute, “:”, CurSecond, CR
    SecondCnt+=1
End Event

Event MyTime.HourlyAlarm0Event()
    Debug “It is 49 minutes now.”, CR
    SecondCnt+=1
End Event

Event MyTime.CountDownTimer0Event()
    Debug “Count down 3 seconds.”, CR
    SecondCnt+=1
End Event
```

Appendix

1. Known Problems:
   - When using SetHour to set the hour, if the Pause command is used to temporarily stop for a short time interval and then the hour value is acquired by GetHour, sometimes an incorrect value will occur.

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

3. Time Calibration Formula:

   To use SetClockAdj to perform a fine calibration of the time difference for a more accurate time setting, use the SetTimeAndDate command to set the Time Keeper module to the available precise time (such as from the time reporting station) and then record the Time to perform the setting operation. Then keep the Time Keeper module unchanged for a time period and observe the difference between the time read from the Time Keeper module and the current time. The parameters can be calculated as following.

   Total test time = setting time – currently measured time (in seconds)
   Current time difference = time measured by Module – actual time (in seconds)
   PPM value = (Current time difference/Total test time) * 1000000 (Note 1)

   If the time different is positive, the input value should be 128 minus the nearest integer of (PPM value/3.052).
   If the time difference is negative, the input value should be the positive nearest integer of (PPM value/3.052) + 1.

   Note 1:
   The allowed range for the fine adjustment is limited in a way that the PPM value should be within the range from -195.3 to 192.2. If the calculated time difference exceeds this range, it is not possible to adjust the accurate time with the fine adjustment value.