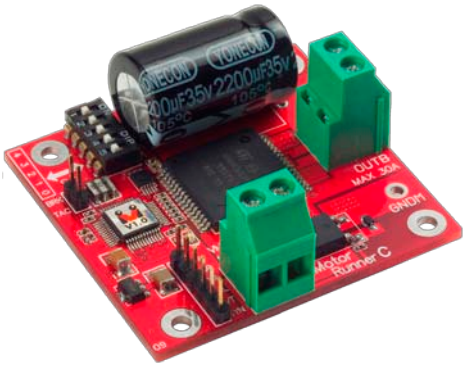
Motor Runner C

User's Guide

Single DC Motor Controller Module

Version: V1.0



Product Overview: Innovati's Motor Runner C Module can freely control a single DC motor through simple commands. It can dynamically change the rotation speed of the motor at any time, and get the current status of the motor, including the rotation speed or the direction. Compared with Motor Runner A and Motor Runner B, it can withstand a higher voltage and a higher current.

Application:

- Control the driving of the motor for moving a model car forwards or backwards.
- Dynamically control the rotation speed of a motor in the equipment that needs the feedback of rotation speed.
- It can be connected with a small fan for controlling the blowing strength.

Product Features:

- Control the rotation direction and speed of a single motor with simple commands
- It can withstand a maximum continuous current up to ±30A.
- It can withstand a maximum voltage up to 35V.
- Internal PWM current control at a fixed frequency of 10KHz.
- Provide the automatic shut down protection against overheat (150°C).
- Provide the protection against current overload.
- Provide the crossover-current protection and the under-voltage lockout (UVLO) protection.
- With the brake command, it can rapidly stop the motion of the motor.
- Provide 256-step rotation speed variation.
- By using the commands, it is easy to obtain the current status of the rotation speed or direction of the motor.
- Provide the connector for external stop signal. With a simple connection to an external button, the user can stop the motion of the motor by simply pressing the button.
- Provide connection pins for the motor rotation speed signal which allows the user to connect the module to a motor with a tachometer to obtain a more accurate rotation speed of the motor in real time through the commands.
- The tachometer can be configured for 13 different sensing frequencies.
- With the connection of a tachometer, the user can directly set the rotation speed through the commands for rotation speed settings so that the module can control the motor to accelerate (decelerate) to the required rotation speed and maintain at a constant rotation speed.
- With the connection of a tachometer, the user can directly set the rotation counts through the counter commands so that the module can control the motor to stop once the required number of rotation counts is reached.
- Provide the counter events. When the required number of counts is reached, the SBC will be notified by the event and perform the follow-up operations after the counter is reached.
- It provides the error alarm event. After the error status is clear, the module can rapidly recover the previous state through the related commands.

Connection: Set the ID switch to the required number directly, and then connect the cmdBUS to the corresponding pins on the BASIC Commander so that the user can perform the required operations through the BASIC Commander.Connect the motor input pins OUTA and OUTB to the corresponding pins on the motor to be controlled and then connect the motor power pins VM and GND to the power supply which can provide the power required for the motor. During the operation by the commands, if the rotation direction of the motor is opposite to the direction specified by the command, it means that the pins OUTA and OUTB are connected reversely. The user can exchange the connection between OUTA and OUTB or swap the forward and backward commands in the program.

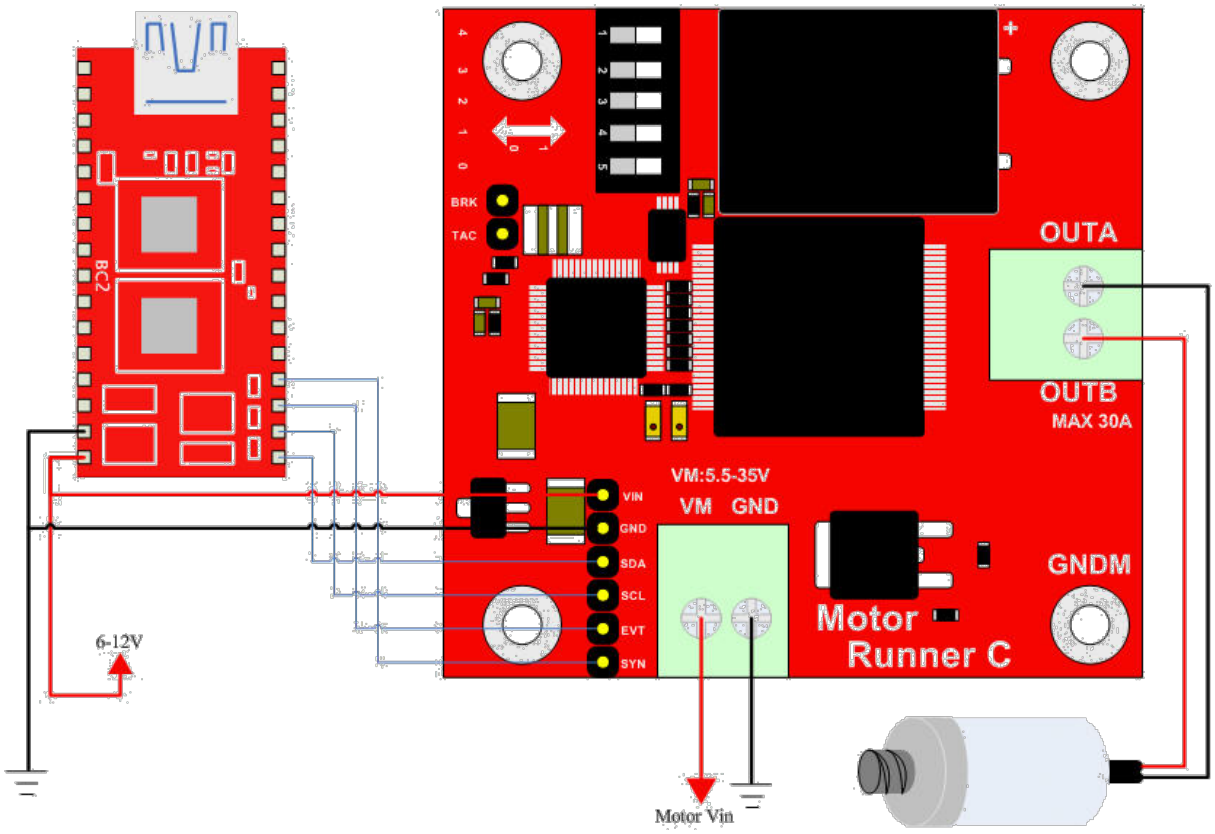
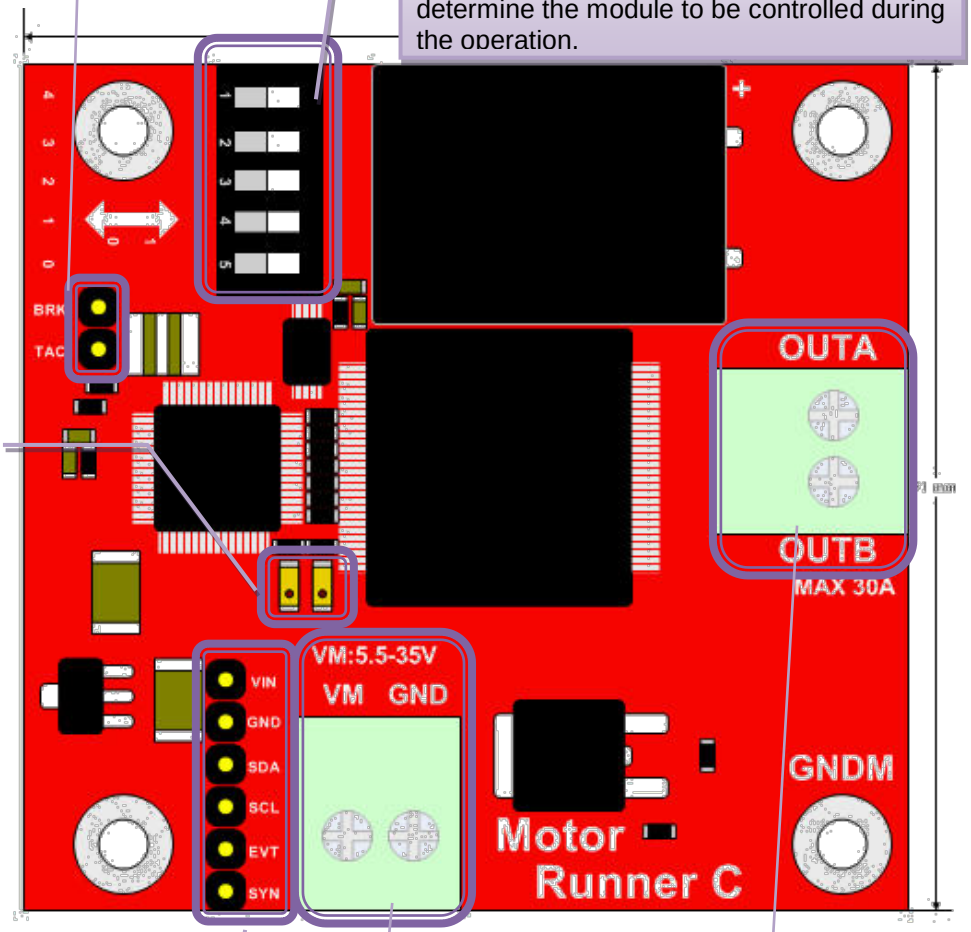


Figure 1 Example of the connection of the motor module

Product Specifications:

BRK is connected with a button so as to allow the user to stop the motor by pressing the button. The other terminal of the button should be grounded.
TAC is the connection pin for tachometer. After the tachometer is connected, the rotation speed control related commands can be used.

Module ID Setting Switch: The module ID of the LCD module can be configured with the binary digits from the right to the left. This ID number allows the BASIC Commander to determine the module to be controlled during the operation.



The right orange light is the data transmission indicator. A blinking light indicates that the module is transmitting/receiving data. The left green light is the event indicator. A blinking light indicates that an event is generated.

Pins for cmdBUS: Connect these pins to the corresponding pins on the BASIC Commander for controlling the Motor Runner C module through the BASIC Commander. While connecting, please notice the pin assignment. Connect Vin to the Vin on the BASIC Commander. Incorrect pin connection may cause damage to the module.)

Motor power connection pins. Please connect the power supply dedicated for the motor to these corresponding pins.

Motor connection pins for determining the rotation direction and speed of the motor. Please note that if the pins M+ and M- are connected reversely, the rotation direction will be opposite the direction specified by the command.

Figure 2 Description of pins and switches on the module

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
I_{DD}	Operating Current	7.5	No I/O	—	5.4	—	mA
f_{pwm}	PWM Output frequency	—	—	0	—	10	kHz

Table 1: Characteristics of the operating current (room temperature at 25 °C)

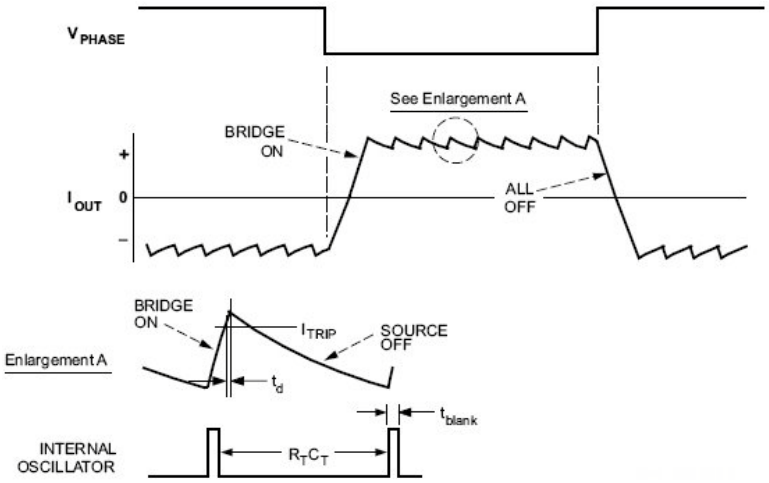
Test Condition: T_A=25°C, V_M=5V-35V

Characteristic	Symbol	Test Conditions	Limits			
			Min.	Typ.	Max.	Units
Load Supply Voltage Range	V_M	Operating	5.5	-	35	V
Thermal Shutdown Temp.	T_J	V_{IN} = 3.25V	150	170	200	°C
Thermal Shutdown Hysteresis.	T_J		7	15	-	°C

Table 2: Electrical Characteristics of a Motor

Overheat Protection: The overheat protection circuit is used for automatically breaking the circuit while the internal temperature up to 165°C inside the drive IC is detected. At this time, the motor will stop the operation. When the temperature is decreased by 8°C, the protection circuits will automatically recover the circuit connection and the motor will continue the previous operations.

Current Limit Protection: Please refer to the right figure. When the H-bridge starts to output, the current will increase as the motor rotates increasingly. When the current value exceeds I_{TRIP} (as the indication shown in the Enlargement A in the lower right figure), the H-bridge output will be stopped. After the next clock of the internal oscillator is sent out (as the INTERNAL OSCILLATOR shown in the lower right figure) and then the current transmission will be continued. In such a way, the operation is repeatedly limited within the range shown in the figure.



Precautions for Operations:

The Motor Module provides one set of connection pins for only one motor. Please make sure that the connected motor is a DC motor.

The heat dissipation fin is not installed on the module at the factory before delivery. Under the low current operations in an well-ventilated environment, the module can work normally. However, when a high current passes through the module, or the high heat cannot be dissipated through ordinary convection, it is recommended to attach the heat dissipation fin on the module. The following table shows the approximate

time that the module can work normally under a higher current without the heat dissipation fin at the room temperature (25°C) in a well-ventilated environment:

Current (A)	Time to overheat protection (Sec)
10	>300
12	~107
15	~40
18	~20

Table 3: Current vs. Overheat Protection Time (without heat dissipation fin)

Operating Temperature of the Module: 0 °C ~ 70 °C (Please confirm the operating temperature for the motor separately)
Storage Temperature of the Module: -50°C ~125°C

Commands And Events:

The following list shows the commands dedicated for controlling the Motor Runner C module. The command name and parameters which should be input are shown in bold or bold-italic typefaces. The words in bold typeface should not be changed while being input. The words in bold-italic typefaces can be filled with parameters in properly defined format by the user. Please note that the words in uppercase or lowercase are regarded as the same word while entering the command in the innoBASIC Workshop.

Before executing the command for Motor Runner C, please define the corresponding parameters and the module ID at the beginning of the program, for example:

Peripheral *ModuleName* As MotorRunnerC @ ModuleID

Command Format	Command Function
Commands for Configuring the Motor Speed	
Forward(*Speed*)	Set the motor to perform the forward rotation operation at a speed specified by the value of ***Speed***. The value of ***Speed*** should be within 0~255 (the higher the value of ***Speed*** is, the faster the motor rotates).
Backward(*Speed*)	Set the motor to perform the backward rotation operation at a speed specified by the value of ***Speed***. The value of ***Speed*** should be within 0~255 (the higher the value of ***Speed*** is, the faster the motor rotates).
SetSpdDC(*Speed)*	Set the rotation speed of the motor module at a value specified by ***Speed***. The motor module still maintains the rotation in the original direction. The input value of ***Speed*** should be within 0~255 (the higher the value of ***Speed*** is the faster the motor rotates).
Commands for Configuring the Rotation Direction of the Motor	
SetDir(*Dir*)	Set the motor to rotate in the direction specified by ***Dir***. If the input value of ***Dir*** is 0, the motor rotates in the forward direction; 1 for the rotation in backward direction.
Stop the Motion of the Motor	
Brake()	Set the motor module to rapidly stop the rotation.
Stop()	Set the motor module to stop the rotation.
Commands for Retrieve the Status of Settings	

GetSpdDC(*Speed*)	Get the preset rotation speed of the motor and store it in ***Speed***. The returned value of ***Speed*** will be within 0~255 (the higher the value of ***Speed*** is, the faster the motor rotates).
GetDir(*Dir*)	Get the preset rotation direction of the motor and store it in ***Dir*** (0 for forward direction and 1 for backward direction).
Commands for Setting the Tachometer and Retrieving the Rotation Speed	
SetTACHPeriod(*Period*)	Set the motor module to count the rotation speed at a time interval specified by ***Period***. The input value of ***Period*** is defined as follows: 0: 16 ms 1: 32 ms 2: 64 ms 3: 125 ms 4: 250 ms 5: 500 ms 6: 1 s 7: 2 s 8: 4 s 9: 8 s 10: 15 s 11: 30 s 12: 60 s The motor module will count the number of pulses measured by the tachometer within the specified time interval. A shorter time interval means a faster update of the measurement value; however, a larger counting error. A longer time interval means a slower update of the measurement value; however, a smaller counting error.
GetTACHPeriod(*Period*)	Get the preset time interval for the motor to retrieve the rotation speed and store it in ***Period***. The returned value of ***Period*** will be within 0~12 which represents the time interval as defined in the previous command.
***Status* = TACHIn(*Speed*)**	Get the update status of the tachometer and store it in ***Status*** (0 represents that no new counting value is measured since the last update; 1 represents that the rotation speed has been updated). Meanwhile, the number of pulses per minute retrieved from the tachometer is returned and stored in ***Speed***. The returned value of ***Speed*** will be within 0~4294836225. Please note that the time interval for updating the returned value and the accuracy may vary depending on the time interval for the pulse counting. *1
Commands for Configuring and Retrieving the Rotation Speed (To perform the commands in this section, it is required to connect a tachometer for correct operation.)	
SetSpdCtrl(*Dir*, *Speed*)	Set the speed control operation. If the input value of ***Dir*** is 0, it means forward rotation; 1 for backward rotation. The input value of ***Speed*** should be within 0~245756250 (the larger number, the faster the rotation speed). If the motor cannot reach the specified rotation speed, the motor will rotate at its maximum speed. Please note that the rotation speed control will not be in effect after this command is executed. It is required to execute the command **SpdCtrlOn**() to enable the rotation speed control. The final rotation speed may have error to some degree

	depending on the measurement time interval specified by the command **SetTACHPeriod** and the time required to reach the target rotation speed may also different.*1
***Status* = GetSpdCtrl(*Dir*, *Speed*)**	Get the status of the rotation speed control operation and store it in ***Status*** (0 represents that the speed control is disabled; 1 means that the speed control is enabled). Meanwhile, the rotation direction is stored in ***Dir***. If the returned value of ***Dir*** is 0, it means the motor rotates forward; 1 for backward. The rotation speed is stored in ***Speed***. The returned value of ***Speed*** will be within 0~245756250.*1
SpdCtrlOn()	Enable the speed control operation according to the rotation speed settings. Please execute the command **SetSpdCtrl** in advance to configure the required rotation speed settings before execute this command.
SpdCtrlOff()	Disable the rotation speed control operation.
***Status* = GetSpdCtrlStatus()**	Get the status of the rotation speed control operation and store it in ***Status*** (0 represents that the target speed is not reached or the speed control operation is not enabled; 1 represents that the target speed is reached).
Commands for Configuring and Retrieving the Counting Operation (To perform the commands in this section, it is required to connect a tachometer for correct operation.)	
SetCount(*Mode*, *Count*)	Set the counter control operation. If the input value of ***Mode*** is 0, when the counter reaches the target number of counts, it will execute the **Stop** command to stop the motor. If it is 1, when the counter reaches the target number of count, it will execute the **Brake** command to stop the motor. The input value of ***Count*** should be within 0~65535 which represents the target number of counts. Please note that the module will not start the counter operation after this command is executed. It is required to execute the command **CountOn**() to enable the counter operation.
***Status*= GetCount(*Mode*, *Count*)**	Get the status of the counter control operation and store it in ***Status*** (0 represents that the counter control operation is not enabled; 1 represents that the counter control is enabled). The stop mode information is stored in ***Mode***. If the returned value of ***Mode*** is 0, when the counter reaches the target number of counts, it will execute the **Stop** command to stop the motor. If it is 1, when the counter reaches the target number of count, it will execute the **Brake** command to stop the motor. The number of count is stored in ***Count***. The returned value of ***Count*** will be within 0~65535.
CountOn()	Enable the counter control operation according to the counter control settings. Please execute the command **SetCount** in advance to configure the required counter control settings before executing this

	command.
CountOnWithEvent()	Enable the counter control operation according to the counter control settings. After the counter reaches the target number of counts, the event **CountFinishEvent** will be generated. Please execute the command **SetCount** in advance to configure the required counter control settings before executing this command.
CountOff()	Disable the counter control operation.
***Status* = GetCountStatus()**	Get the status of the counter control operation and store it in ***Status*** (0 represents that the target number of counts is not reached or the counter control operation is not enabled; 1 represents that the target number of counts is reached).
Commands for Retrieving the Error Status and Restoring to the Default Settings	
***Status* = GetBrakeButStatus()**	Get the status of the Brake button and store it in ***Status***. The returned value 0 represents that the Brake button is not activated; 1 represents that the Brake button is activated. Please note that after the Brake button is activated, this returned value will remain as 1 until the command **ClrBrakeButStatus** is executed. When the status is 1, all the motor activation commands will have no effect.
ClrBrakeButStatus()	Clear the status of the Brake button.
***Status* = GetFaultStatus()**	Get the status of the fault detection and store it in ***Status*** (The returned value 0 represents that no fault is detected; 1 represent that a fault is detected) *2
EnFaultStop()	Enable the function of automatically stopping the motor when a fault is detected. The fault includes the motor stop due to IC overheat or the current limiting operation due to instantaneous overcurrent. After this command is executed, when a fault occurs, it is necessary to execute the command **ClearFault()** to re-start the motor again. The default value is **DisFaultStop().**
DisFaultStop()	Disable the automatic motor stop operation when a fault is detected. After this command is executed, when a fault occurs, the module will automatically execute the command **ClearFault()** and continue the previously preset motor operation without stopping the motor. The user should determine whether to stop the motor or not by him/her-self. The default value is **DisFaultStop()**. The user can determine whether the fault continues to occur according to the **FaultProtectionEvent.**
ClearFault()	Clear the fault. When the command **EnFaultStop()** is executed, please make sure that the cause of the fault is solved in advance before executing this command

	to continue the operation of the motor.
RestoreStatus()	Restore the setting at the time the fault occurs. When a fault occurs in the module, it will automatically store all the settings at the moment. After this command is executed, it can restore all the setting to the values at that moment.
EnFaultEvent()	Enable the fault notification event. The default value is **EnFaultEvent().**
DisFaultEvent()	Disable the fault notification event. The default value is **EnFaultEvent().**

Table 4 :Command Table

***1 Please note that the setting value and the returned value will have different maximum values depending on the setting value of Period.**

Period=0 ➔ 245756250
Period=1 ➔ 122878125
Period=2 ➔ 61439063
Period=3 ➔ 31456800
Period=4 ➔ 15728400
Period=5 ➔ 7864200
Period=6 ➔ 3932100
Period=7 ➔ 1966050
Period=8 ➔ 983025
Period=9 ➔ 491513
Period=10 ➔ 262140
Period=11 ➔ 131070
Period=12 ➔ 65535

***2 Under the condition of DisFaultStop(), after the FaultEvent is received, the system will automatically perfoem the ClearFault() operation. So a returned value of 0 will be retrieved when the command GetFaultStatus() is executed.**

Event	Activation Condition
CountFinishEvent	When the command **CountOnWithEvent** is activated for the counter control operation and the counter reaches the preset number of counts.
FaultProtectionEvent	When a fault occurs and is detected.

Table 5 :Event Table

Demonstration Program:

```
Peripheral myMotor As MotorRunnerC @ 0      ' Set the module ID as 0

Sub Main()
   Debug CLS
   MyMotor.Forward(200)       ' Set the motor to rotate forward at the speed of 200
   Pause 3000
   MyMotor.Stop()             ' Stop the Motion of the Motor
   Pause 3000
   MyMotor.Backward(200)      ' Set the motor to rotate backward at the speed of 200
   Pause 3000
   MyMotor.SetDir(0)          ' Set the motor to rotate in the forward direction
   Pause 3000
   MyMotor.SetSpdDC(150)      ' Change the rotation speed of the motor to 150
   Pause 3000
   MyMotor.Brake()            ' Stop the motor rapidly
End Sub
```

Appendix

1. Known Problem:

2. List of the Configuration of the Module ID Switch:

	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