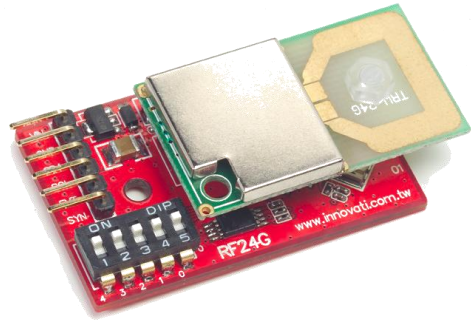


RF24G

無線雙向傳輸模組

版本: V2.0



產品介紹: 利基 RF24G 模組，提供使用者簡單易用的無線雙向傳輸，透過 cmdBUS 與 Arminno 連接，就能以單一指令直接傳送多種格式的資料，透過軟體的動態設定，可以隨時切換收發模式，變更傳輸頻道與識別碼。

應用方向:

- 無線傳輸各種資料。
- 傳送控制訊號，達到無線遙控的功能。
- 同時使用四組 RF24G 模組，完成全雙工通訊。

產品特色:

- 無線傳輸頻率範圍: 2.4 ~ 2.524 GHz。
- 無線傳輸模式: GFSK。
- 能隨時更換模組狀態為接收或發送模式。
- 可以軟體動態切換 125 個頻道。
- 輸出功率: 0 dBm。
- 資料傳輸速率: 250 Kbps。
- 無線傳輸範圍可達約 280 公尺。
- 內建天線無須再外接其他天線。
- 提供 256 組 ID 碼與 Reg 碼讓使用者可做動態設定識別，可以隨時透過軟體更換。
- 可以將想要傳輸的資料先儲存到內建暫存空間，再用指令一次傳送，最多可以儲存 40 Bytes 的資料。
- 簡易的變數傳輸指令，Byte，Word，與 Dword 都可以透過單一指令傳送。
- 提供字串與陣列傳輸指令，可以一次最多傳送 20 個字元或 20 Bytes 長度的陣列。
- 四段傳送強度可動態調整: -20 dBm，-10 dBm，-5 dBm，0 dBm。
- 透過指令可以隨時讀回目前設定值做狀態確認，以及是否有未讀取的接收資料。
- 可透過 cmdBUS 方式，下達指令。

連接方式: 直接將 ID 開關撥至欲設定的編號，再將 cmdBUS 連接至 Arminno 上對應的腳位，就可透過 Arminno 執行操作。

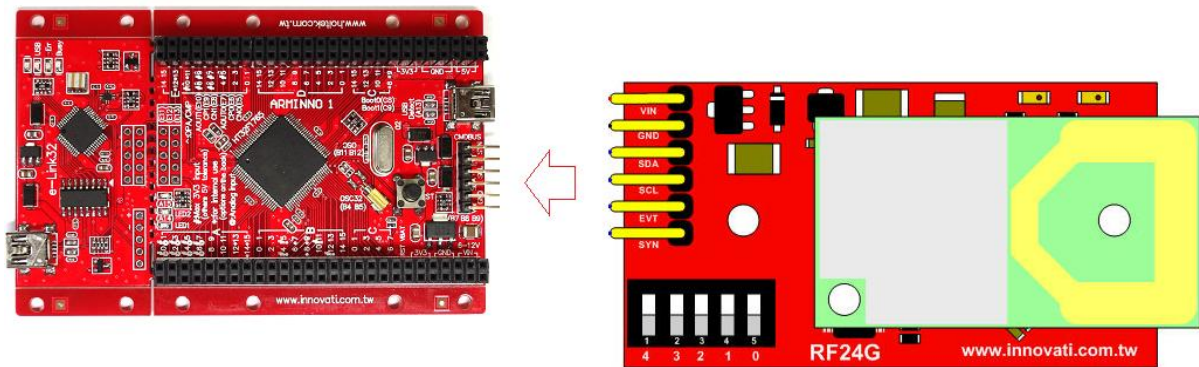
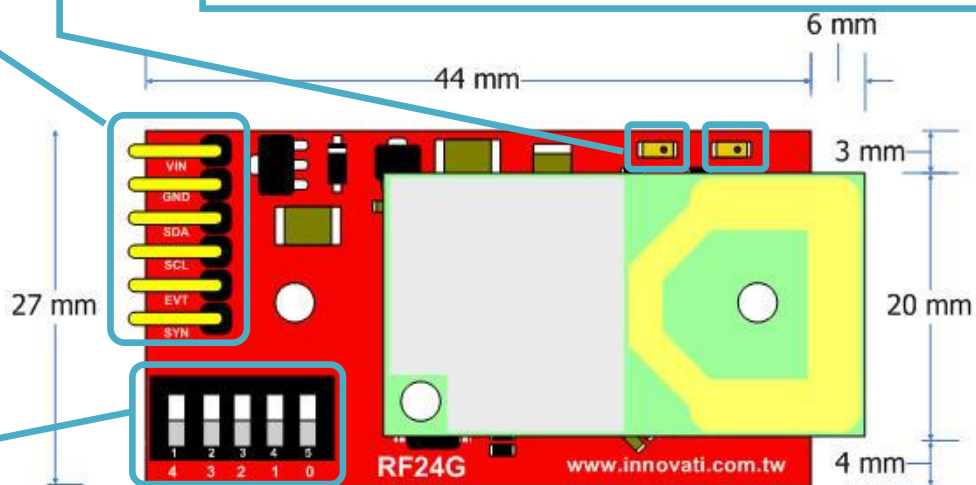


圖 1: 與 Arminno 連接

產品規格:

cmdBUS 接腳，將此處腳位與 Arminno 對應腳位相接，即可透過 Arminno 操控 RF24G 模組(連接時請注意腳位對應，將 Vin 對接 Arminno 上的 Vin 腳位，若是腳位錯誤可能造成模組損毀)。

橘色指令指示燈，閃爍代表模組與 Arminno 正在收送資料



模組編號設定開關，由右至左以二進制設定 RF24G 模組的模組編號，編號可以讓 Arminno 操控時，判斷想要控制的模組(請參考附錄 2)

圖 2: 模組腳位與開關介紹

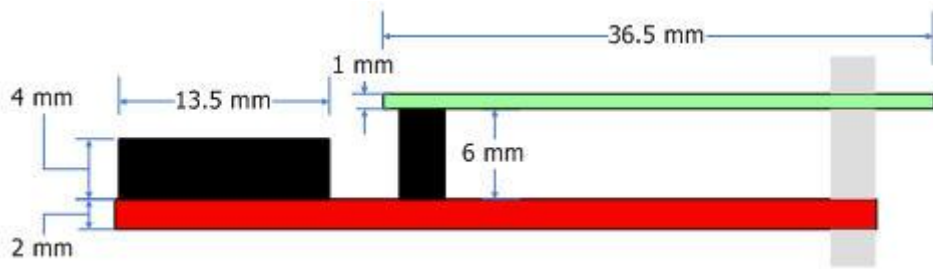


圖 3: 模組側視圖

操作注意事項:

- 無線傳輸時，請不要碰觸天線以免影響資料收送。

模組操作溫度 0 °C ~ 70°C

模組儲存溫度 -40 °C ~ 85°C

指令格式	指令功能
資料收送相關指令…傳送模式	
SendVar(Data)	傳送 <i>Data</i> 的資料， <i>Data</i> 可以為任意參數
SendArray(const void *Array)	傳送 <i>Array</i> 的資料， <i>Array</i> 為一個陣列參數，陣列的總 Byte 數須小於等於 20 Bytes
SendString(const void *String)	傳送 <i>String</i> 的資料， <i>String</i> 為一個字串參數，字串的總字數須小於等於 20 個字元
BufferVar(Data)	將 <i>Data</i> 儲存到傳送資料暫存記憶體中， <i>Data</i> 可以為任意參數 * 1
BufferArray(void* Array)	將 <i>Array</i> 儲存到傳送資料暫存記憶體中， <i>Array</i> 為一個陣列參數，陣列的總 Byte 數須小於等於 20 Bytes * 1
BufferString(void* String)	將 <i>String</i> 儲存到傳送資料暫存記憶體中， <i>String</i> 為一個字串參數，字串的總字數須小於等於 20 個字元 * 1
SendBuffer(void)	將儲存於傳送資料暫存記憶體的所有資料，一次傳送出去
資料收送相關指令…接收模式	
GetVar(&Data)	取得接收資料暫存記憶體中的資料，儲存於 <i>Data</i> 中，需要根據傳送端設定的參數格式，將 <i>Data</i> 設定為對應的參數格式
GetArray(Array)	取得接收資料暫存記憶體中的資料，儲存於 <i>Array</i> 中，需要根據傳送端設定的陣列長度與參數格式，將 <i>Array</i> 設定為對應的陣列值
GetString(String)	取得接收資料暫存記憶體中的資料，儲存於 <i>String</i> 中，需要根據傳送端設定的字串長度，將 <i>String</i>

	設定為對應的長度值
資料收送相關指令…收送共用	
uint8_t Status = GetStatus(void)	<p>發送模式</p> <p>Status = 0: 待命傳送狀態，模組此時可以接受各種傳送指令</p> <p>Status = 1: 資料傳送狀態，模組此時僅能接受模式設定指令，不能執行傳送指令，也不能執行 Config 更改模式或狀態</p> <p>接收模式</p> <p>Status = 0: 接收資料暫存記憶體中沒有收到新的資料</p> <p>Status = 1~40: 代表接收資料暫存記憶體中的資料筆數，亦即 1 代表有一筆資料，最多會有 40 筆資料儲存在記憶體中，若是在有資料未讀取的狀態下，又收到新資料，則就資料會被清空</p>
ClrBuffer(void)	<p>發送模式</p> <p>清除所有傳送資料暫存記憶體中的資料</p> <p>接收模式</p> <p>清除所有接收資料暫存記憶體中的資料，使用 GetStatus 會讀回 0</p>
收送狀態設定相關指令	
SetMode(uint8_t Mode)	<p>根據 Mode 值設定模組為發送或接收模式</p> <p>Mode = 0 → 設定模組為發送模式</p> <p>Mode = 1 → 設定模組為接收模式</p> <p>預設值為 1 (接收模式) *2</p>
SetCh(uint8_t Channel)	<p>根據 Channel 值設定模組為所使用的頻道，Channel 可以設定為 0~124 之件的整數值，預設值為 0 *2</p>
SetRFID(uint8_t ID)	<p>根據 ID 值設定模組所使用的識別碼，ID 可以設定為 0~255 之間的整數值，預設值為 0 *2</p>
SetRegCode(uint8_t Reg)	<p>根據 Reg 值設定模組所使用的註冊碼，Reg 可以設定為 0~255 之間的整數值，預設值為 0 *2</p>
Config(void)	<p>將設定的模式，頻道，識別碼與註冊碼，下載到模組中 *2</p>

*1 傳送資料暫存記憶體最多可儲存 40 Bytes 的資料，當儲存滿 40 Bytes 後再執行 Buffer 相關的指令，會被視為無效指令

*2 發送接收模式，頻道，識別碼與註冊碼，在設定後並不會立刻更新，而是要執行 Config 指令後，才會一次更新此四項資料並啟動設定值

*3 此處的資料遺失，是指接收的資料尚未被讀取，又有接收到新的資料，此時原先在接收資料暫存記憶體的資料會被清除，而以新的資料取代

範例程式:

```
//=====
//    RF24G 發送端範例程式
//=====

#include "arminno.h"

RF24G myT(0);                //    設定模組編號為 0

uint8_t g_bTxReady;          //    宣告傳送狀態參數

int main(void)
{
    //    主程式開始

    uint8_t bTx;              //    宣告傳送資料參數
    myT.SetMode(0);           //    設定模式為發送模式
    myT.SetCh(0);             //    設定傳輸頻道為 0
    myT.SetRFID(0);           //    設定辨識碼為 0
    myT.SetRegCode(0);        //    設定註冊碼為 0
    myT.Config();             //    更新設定值

    //    用 FOR 迴圈重複傳送動作一百次
    for( bTx=0; bTx < 10; bTx++) //    會執行一百次的 FOR 迴圈
    {
        myT.SendVar(bTx);      //    傳送 bTx

    //    用 DO 迴圈等待傳送完畢
        do {
            Pause(10);
            g_bTxReady = myT.GetStatus();
        }while( g_bTxReady == 0);
        printf("%d", bTx);      //    顯示傳送值於終端視窗
        Pause( 1000);           //    等待一段時間讓接收端接收
    }
    printf("Tx Done");         //    顯示傳送結束
}

//=====
//    RF24G 接收端範例程式
//=====

#include "arminno.h"

RF24G myR(31);                //    設定模組編號為 31

uint8_t g_bRxReady;           //    宣告接收狀態參數
```

```

int main(void)
{
    // 主程式開始
    uint8_t bRx; // 宣告接收資料參數

    myR.SetMode(1); // 設定模式為接收模式
    myR.SetCh(0); // 設定傳輸頻道為 0
    myR.SetRFID(0); // 設定辨識碼為 0
    myR.SetRegCode(0); // 設定註冊碼為 0
    myR.Config(); // 更新設定值

































    // 用 DO 迴圈等待接收到最後一筆資料
    do {
    // 用 DO 迴圈等待接收資料
        do {
            Pause(10);
            g_bRxReady = myR.GetStatus();
        }while( g_bRxReady >= 1);
        myR.GetVar(bRx); // 讀取接收值
        printf("%d", bRx); // 顯示接收值於終端視窗
    }while( bRx != 10);
    printf("Rx Done"); // 顯示接收結束
}

```

附錄

1. 已知問題:

2. 模組編號開關對應編號表:

	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