Titania (AMB3626) ??????????

> 模組型號: Würth Titania 2607011111100x (AMB3626)
> 硬體相關手冊章節: Manual-um-titania-260701111100x (rev4.5) - 第 2 章、第 3 章、第 7 章
> 主要參考標準: EN 300 220 (歐盟無線設備規範)

? ??????????

- [ ] VCC 已穩定供電 (2.0–3.6 V, 建議 3.3 V)
- [ ] /RESET 腳位已釋放 (被 RC 電路或軟體拉高)
- [ ] **TRX_DISABLE** 接 **GND** (一定要)
- [ ] UART TX/RX 已接到 STM32
- [ ] /CONFIG 腳位接到 STM32 GPIO (選用, 但建議接)
- [ ] /RTS 腳位接到 STM32GPIO 或監測 (強烈建議)
- [ ] 沒有無線訊號、 UART 也空閒時, 才改設定
- [ ] 改完非揮發參數後, 務必 **CMD_RESET_REQ**

? ?????????

1. VCC ? /RESET ?

腳位定義:

腳位	標號	用途	電壓	說明
VCC	1	電源正	+3.3 V (範圍 2.0–3.6 V)	模組主電源
GND	2 (多個)	地	0 V	必須與 MCU 共地

腳位	標號	用途	電壓	說明
/RESET	某腳	模組重置	開漏輸出或 RC 延遲	上電時 必須先低、再高

上電時序（極其重要）：

時間線：

T=0 ms: VCC 開始上升

T=0~2 ms: VCC 還在上升，/RESET 必須保持 LOW（不要放高）

T=2 ms: VCC 穩定在 3.3 V

T=2~10 ms: /RESET 開始被 RC 電路或軟體拉高

T=10~50 ms: Titania 內部初始化進行中

T=50 ms+: 模組完成啟動，UART + /CONFIG 開始工作

⚠◌ 如果 /RESET 在 VCC 還不穩時就被拉高，模組會受損或無法啟動。

建議實作：

- 硬體 **RC** 延遲方案（最簡單）：

```
VCC ──[10kΩ]──┬──/RESET
              ├──[100nF]──┴──GND
              └──外部按鍵到 GND（選用）

效果：上電時，/RESET 會被 100nF 電容延遲 ~10~50 ms 才慢慢被拉高，
      自動保證 VCC 先穩定再放開 /RESET。
```

- 軟體方案（ **MCU** 控制）：

```
void Titania_PowerUp(void)
{
    // 上電時 /RESET 已經被拉低（可能通過外部電路或預設狀態）
    HAL_GPIO_WritePin(RESET_PORT, RESET_PIN, GPIO_PIN_RESET);  // 確保低
    HAL_Delay(10);   // 等 VCC 穩定
    HAL_GPIO_WritePin(RESET_PORT, RESET_PIN, GPIO_PIN_SET);    // 拉高
    HAL_Delay(100);  // 等模組啟動完成
}
```

---

2. TRX_DISABLE ??????????

功能： 當 TRX_DISABLE 被拉高時，模組無法傳送無線訊號（ RX 仍可用）。用來省功耗或避免干擾。

設定參數時的用途：雖然不是「必須」讓它 HIGH，但強烈建議在整個參數設定過程中保持 **LOW**（讓無線 **TX** 可用，但我們不會在設定時送訊號）。

狀態	說明
LOW（接 **GND**）	無線 TX/RX 都啟用（正常工作模式）
HIGH	無線 TX 被禁用，模組停止傳送（僅 RX）
浮接	不允許，會造成無法預知的行為

建議實作：

```
方案 1（最簡單，出廠推薦）：
  TRX_DISABLE ──→ GND
   （直接接地，永遠 LOW，模組正常工作）


方案 2（後期要動態禁用 TX 時）：
  TRX_DISABLE ──[10kΩ pull-down]──┬──STM32 GPIO（或保留浮接用軟體控制）
                                  └──GND


  HAL_GPIO_WritePin(TRX_DIS_PORT, TRX_DIS_PIN, GPIO_PIN_RESET);  // 允許 TX
  HAL_GPIO_WritePin(TRX_DIS_PORT, TRX_DIS_PIN, GPIO_PIN_SET);    // 禁用 TX
```

---

3. /CONFIG ??Mode Selector?Transparent ? Command?

功能：檢測「下降沿」（HIGH → LOW）來切換模式。

邊緣	狀態轉換
下降沿（HIGH → LOW）	進入或跳出 Command Mode
LOW 狀態保持	停留在 Command Mode
HIGH 狀態保持	Transparent Mode

時序要求（最重要）：

只有在「沒有無線收發、也沒有 UART 資料傳輸」的安全區間才能切模式。手冊建議看 **/RTS** 線：

- `/RTS = HIGH`（模組忙，有 UART 活動）→ 不要切模式
- `/RTS = LOW`（模組空閒）→ 可以安全地做下降沿切模式

```
void Titania_EnterCommandMode(GPIO_TypeDef *cfg_port, uint16_t cfg_pin,
                              GPIO_TypeDef *rts_port, uint16_t rts_pin)
{
    // 1. 等待 /RTS = LOW（模組空閒），timeout 100 ms
    uint32_t timeout = 100;
    while (HAL_GPIO_ReadPin(rts_port, rts_pin) == GPIO_PIN_SET && timeout--)
    {
        HAL_Delay(1);
    }

    // 2. 確認 /CONFIG 已經 HIGH（停留 Transparent Mode）
    HAL_GPIO_WritePin(cfg_port, cfg_pin, GPIO_PIN_SET);
    HAL_Delay(10);

    // 3. 做下降沿
    HAL_GPIO_WritePin(cfg_port, cfg_pin, GPIO_PIN_RESET);
    HAL_Delay(5);

    // 4. 保持 LOW 一段時間（表示「停留 Command Mode」）
    //  （不需要立即拉回 HIGH）

    HAL_Delay(50);  // 給模組時間完成模式切換
}

void Titania_ExitCommandMode(GPIO_TypeDef *cfg_port, uint16_t cfg_pin)
{
    // 1. /CONFIG 保持 LOW 一段時間（已在 Command Mode 中）
    // 2. 再做一次下降沿，跳出 Command Mode

    HAL_GPIO_WritePin(cfg_port, cfg_pin, GPIO_PIN_SET);   // 拉 HIGH
    HAL_Delay(10);
    HAL_GPIO_WritePin(cfg_port, cfg_pin, GPIO_PIN_RESET); // 拉 LOW（下降沿）
    HAL_Delay(5);

    // 3. 拉 HIGH 回 Transparent Mode
    HAL_GPIO_WritePin(cfg_port, cfg_pin, GPIO_PIN_SET);
    HAL_Delay(50);
}
```

4. UART ?????TX?RX?/RTS?/CTS?

最小硬體連接（必須）：

訊號	**Titania** 腳	**STM32** 腳	說明
UTXD	Titania TX	STM32 RXx	模組發送，MCU 接收
URXD	Titania RX	STM32 TXx	MCU 發送，模組接收
GND	GND	GND	必須共地

選用但強烈建議的流控腳位：

訊號	**Titania** 腳	**STM32** 腳	說明
/RTS	模組 RTS	STM32 GPIO 輸入	模組輸出，LOW = 可以發送，HIGH = 忙碌
/CTS	模組 CTS	STM32 GPIO 輸出	若不用，直接接 CTS flow control GND

為什麼要監測 **/RTS**？

- `/RTS = HIGH`：模組 UART buffer 滿或正在無線收發，MCU 不應再丟資料。
- `/RTS = LOW`：安全區間，可以發 UART 命令或改設定。

建議實作：

```
#define UART_RX_PIN     GPIO_PIN_10  // STM32 PA10 (USART1 RX)
#define UART_TX_PIN     GPIO_PIN_9   // STM32 PA9 (USART1 TX)
#define RTS_PIN         GPIO_PIN_11  // STM32 PA11 (輸入，讀取 Titania /RTS)
#define CTS_PIN         GPIO_PIN_12  // STM32 PA12 (輸出，若不用直接接 GND)

void Titania_UART_Init(void)
{
    // UART1 初始化為 9600 8N1
    huart1.Instance = USART1;
    huart1.Init.BaudRate = 9600;
    huart1.Init.WordLength = UART_WORDLENGTH_8B;
    huart1.Init.StopBits = UART_STOPBITS_1;
    huart1.Init.Parity = UART_PARITY_NONE;
    huart1.Init.Mode = UART_MODE_TX_RX;
    HAL_UART_Init(&huart1);

    // GPIO 設定: /RTS 為輸入（讀模組狀態）
```

```
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    GPIO_InitStruct.Pin = RTS_PIN;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    // /CTS 為輸出且拉低（若不用 CTS flow，直接 LOW）
    GPIO_InitStruct.Pin = CTS_PIN;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    HAL_GPIO_WritePin(GPIOA, CTS_PIN, GPIO_PIN_RESET);
}

HAL_StatusTypeDef Titania_SendUARTSafe(const uint8_t *data, uint16_t len)
{
    // 先等 /RTS = LOW（模組空閒），timeout 100 ms
    uint32_t timeout = 100;
    while (HAL_GPIO_ReadPin(GPIOA, RTS_PIN) == GPIO_PIN_SET && timeout--)
    {
        HAL_Delay(1);
    }
    if (timeout == 0) return HAL_TIMEOUT;

    // 發送
    return HAL_UART_Transmit(&huart1, (uint8_t *)data, len, 100);
}
```

5. ANT??????

最小設置:　　不需要　　MCU 控制，但　　PCB layout 要遵守手冊規則:

- 天線線路　　旁邊要有「淨空區」（通常　　10~15 mm），不要拉數位線穿過。
- 天線匹配電路　　按手冊　　reference design 來，不要亂改。
- 若用外部天線，焊接要牢固，斷線會影響通訊距離。

? ??????????

?????

```
START
  ↓
[1] 上電 & /RESET 釋放
  └→ TRX_DISABLE = LOW (允許無線 TX/RX)
  └→ 等 100 ms 模組啟動完成
  ↓
[2] 初始化 STM32 UART (9600 8N1)
  ↓
[3] 監測 /RTS, 等它變 LOW (模組空閒)
  ↓
[4] 進入 Command Mode (/CONFIG 下降沿)
  └→ 等 /CONFIG 穩定在 LOW 50 ms
  ↓
[5] 透過 UART 發送 CMD_SET_REQ (改 baudrate/channel/NetID…)
  └→ 等 50~200 ms 回應
  ↓
[6] 檢查回應 Status (0x00 = OK)
  ├─ OK → 步驟 [7]
  └─ FAIL → 重試或 debug
  ↓
[7] 再監測 /RTS = LOW
  ↓
[8] 發送 CMD_RESET_REQ (軟重啟)
  └→ 等 500 ms 模組重啟完成
  ↓
[9] 改 STM32 UART 波特率 (新的 baudrate)
  ↓
[10] 退出 Command Mode (/CONFIG 再做下降沿)
  └→ 拉 HIGH, 再下降沿, 再拉 HIGH
  ↓
[11] 回到 Transparent Mode, 可以正常收發無線訊號
  ↓
DONE
```

? ??????????????

```c
#include "stm32f1xx_hal.h"

// □□□□
#define RESET_PORT      GPIOA
#define RESET_PIN       GPIO_PIN_0
#define TRX_DIS_PORT    GPIOA
#define TRX_DIS_PIN     GPIO_PIN_1
#define CONFIG_PORT     GPIOA
#define CONFIG_PIN      GPIO_PIN_2
#define RTS_PORT        GPIOA
#define RTS_PIN         GPIO_PIN_3

extern UART_HandleTypeDef huart1;

// □□ Checksum
static uint8_t CalcChecksum(uint8_t *buf, uint8_t len)
{
    uint8_t cs = 0;
    for (uint8_t i = 0; i < len; i++)
        cs ^= buf[i];
    return cs;
}

// □□ UART □□□□□ Checksum□
static HAL_StatusTypeDef Titania_SendCommand(uint8_t cmd,
                                             const uint8_t *data,
                                             uint8_t data_len)
{
    uint8_t frame[64];
    uint8_t idx = 0;

    frame[idx++] = 0x02;  // Start
    frame[idx++] = cmd;
    frame[idx++] = data_len;
    for (uint8_t i = 0; i < data_len; i++)
        frame[idx++] = data[i];

    uint8_t cs = CalcChecksum(frame, idx);
    frame[idx++] = cs;
```

```
    return HAL_UART_Transmit(&huart1, frame, idx, 100);
}

// 等待 /RTS = LOW（模組空閒）
static HAL_StatusTypeDef Titania_WaitReady(uint32_t timeout_ms)
{
    while (HAL_GPIO_ReadPin(RTS_PORT, RTS_PIN) == GPIO_PIN_SET && timeout_ms--)
    {
        HAL_Delay(1);
    }
    return (timeout_ms == 0) ? HAL_TIMEOUT : HAL_OK;
}

// 進 Command Mode
static void Titania_EnterCmdMode(void)
{
    Titania_WaitReady(100);
    HAL_GPIO_WritePin(CONFIG_PORT, CONFIG_PIN, GPIO_PIN_SET);   // HIGH
    HAL_Delay(10);
    HAL_GPIO_WritePin(CONFIG_PORT, CONFIG_PIN, GPIO_PIN_RESET); // 下降沿
    HAL_Delay(50);
}

// 退出 Command Mode
static void Titania_ExitCmdMode(void)
{
    Titania_WaitReady(100);
    HAL_GPIO_WritePin(CONFIG_PORT, CONFIG_PIN, GPIO_PIN_SET);   // HIGH
    HAL_Delay(10);
    HAL_GPIO_WritePin(CONFIG_PORT, CONFIG_PIN, GPIO_PIN_RESET); // 下降沿
    HAL_Delay(5);
    HAL_GPIO_WritePin(CONFIG_PORT, CONFIG_PIN, GPIO_PIN_SET);   // 回 HIGH
    HAL_Delay(50);
}

// 上電與初始化
void Titania_PowerUp(void)
{
    // 1. GPIO 初始化
    GPIO_InitTypeDef GPIO_InitStruct = {0};
```

```
    GPIO_InitStruct.Pin = RESET_PIN | TRX_DIS_PIN | CONFIG_PIN;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = RTS_PIN;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    // 2. □□ /RESET □□TRX_DISABLE □□/CONFIG □
    HAL_GPIO_WritePin(RESET_PORT, RESET_PIN, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(TRX_DIS_PORT, TRX_DIS_PIN, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(CONFIG_PORT, CONFIG_PIN, GPIO_PIN_SET);
    HAL_Delay(10);

    // 3. □□ /RESET
    HAL_GPIO_WritePin(RESET_PORT, RESET_PIN, GPIO_PIN_SET);
    HAL_Delay(100);  // □□□□□
}

// □□□□□□
typedef struct
{
    uint32_t baudrate;  // 1200~115200
    uint8_t  channel;   // 0~4
    uint16_t netid;     // NetID
    uint16_t addr;      // Addr LSB
} TitaniaSettings;

HAL_StatusTypeDef Titania_Configure(const TitaniaSettings *settings)
{
    uint8_t payload[10];
    uint8_t resp[10];
    HAL_StatusTypeDef st;

    // 1. □ Command Mode
    Titania_EnterCmdMode();
    HAL_Delay(100);
```

```
// 2. 改 UART Baudrate（非揮發）
payload[0] = 0x50;  // UART_Baudrate mem pos
payload[1] = 0x04;  // 4 bytes
payload[2] = (uint8_t)(settings->baudrate & 0xFF);
payload[3] = (uint8_t)((settings->baudrate >> 8) & 0xFF);
payload[4] = (uint8_t)((settings->baudrate >> 16) & 0xFF);
payload[5] = (uint8_t)((settings->baudrate >> 24) & 0xFF);

Titania_SendCommand(0x09, payload, 6);
HAL_Delay(100);
HAL_UART_Receive(&huart1, resp, 5, 200);

if (resp[3] != 0x00) return HAL_ERROR;  // Status check

// 3. 改頻道（非揮發）
payload[0] = 0x2A;  // PHY_DefaultChannel mem pos
payload[1] = 0x01;  // 1 byte
payload[2] = settings->channel;

Titania_SendCommand(0x09, payload, 3);
HAL_Delay(100);
HAL_UART_Receive(&huart1, resp, 5, 200);

if (resp[3] != 0x00) return HAL_ERROR;

// 4. 改 NetID（需要補上正確的 mem pos）
// payload[0] = 0xXX;  // MAC_DefaultDestNetID mem pos (TODO)
// payload[1] = 0x02;  // 2 bytes
// payload[2] = (uint8_t)(settings->netid & 0xFF);
// payload[3] = (uint8_t)(settings->netid >> 8);
// Titania_SendCommand(0x09, payload, 4);
// HAL_UART_Receive(&huart1, resp, 5, 200);

// 5. 軟重啟讓設定生效
Titania_SendCommand(0x05, NULL, 0);
HAL_Delay(100);
HAL_UART_Receive(&huart1, resp, 5, 500);

// 6. 改 STM32 UART 波特率
```

```
    huart1.Init.BaudRate = settings->baudrate;
    HAL_UART_Init(&huart1);
    HAL_Delay(100);

    // 7. 退出 Command Mode
    Titania_ExitCmdMode();

    return HAL_OK;
}

// 使用範例
int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_USART1_UART_Init();  // 先 9600

    // 上電
    Titania_PowerUp();
    HAL_Delay(200);

    // 設定參數
    TitaniaSettings cfg = {
        .baudrate = 57600,
        .channel = 2,
        .netid = 0x0001,
        .addr = 0x0005
    };
    Titania_Configure(&cfg);

    while (1)
    {
        // 主應用程式...
    }
}
```

? ???? & ??

問題	原因	解決方案
上電後模組不回應	/RESET 在 VCC 未穩時被拉高，或 /RESET 沒被正確拉高	檢查 RC 延遲電路、或用示波器看 /RESET 時序
無法進 Command Mode	/CONFIG 下降沿時機不對，或被拉高時間不足	確保 /RTS = LOW 後再做下降沿，並保持 LOW 50 ms+
改參數後 UART 通訊失敗	沒有做 CMD_RESET_REQ、或 STM32 UART 波特率沒同步改	一定要 reset，再改 STM32 baud rate
頻道、NetID 沒生效	memory position 填錯，或 status byte 不是 0x00	對照手冊第 8 章確認 memory position；檢查 Checksum
/RTS 一直高	UART buffer 滿，或模組忙碌	等待、或檢查有沒有大量無線訊號干擾

? ??????????

設定項目	需進 **Command Mode**	需要 **Reset**	**UART** 切換	安全區間條件
UART_Baudrate	□	□	□ □□□ STM32 baud	/RTS=LOW
PHY_DefaultChannel	□	□	□ □□□	/RTS=LOW
MAC_DefaultDestNetID	□	□	□ □□□	/RTS=LOW
MAC_DefaultDestAddrLSB	□	□	□ □□□	/RTS=LOW
PHY_PAPower	□	□	□ □□□	/RTS=LOW
CMD_FACTORY_RESET	□	□	□ □□□ STM32 baud □ 9600	/RTS=LOW
CMD_SET_CHANNEL_REQ (揮發)	□	□	□ □□□	/RTS=LOW

? ?????????

- ☐ PCB 佈局按手冊 reference layout
- ☐ VCC 穩壓、旁路電容（ 100 nF + 1 µF）
- ☐ /RESET 有 RC 延遲或軟體控制
- ☐ TRX_DISABLE 接 GND（正常模式）
- ☐ UART TX/RX 正確連接且沒反接

- [ ] /RTS 接到 GPIO 或監測
- [ ] /CONFIG 接到 GPIO (可進 Command Mode)
- [ ] 天線線路淨空、不穿過數位線
- [ ] 第一次設定時用邏輯分析儀監看時序確認
- [ ] 改完參數後驗證設定有沒有真的生效（ CMD_GET_REQ 讀回來確認）

? ??

- **Würth Titania Manual:** Manual-um-titania-260701111100x (rev4.5) - 第 2、3、7、8 章
- **HAL Reference:** STM32 HAL Driver 官方文檔
- **EN 300 220:** 歐盟無線設備規範

最後更新: 2026-03-10

作者備註: 本硬體指南強調「時序」與「狀態機」，設定參數時務必遵守 /RTS、/CONFIG 的時序要求。

Revision #3
Created 2026-04-01 02:06:03 UTC by TaipeiTechRacing
Updated 2026-04-11 14:34:06 UTC by AI Agent