



远-T1L 发射模块使用说明

程序详解、1527 类编码原理、二次开发和移植注意事项

1. 解决什么问题

本文面向第一次接触无线发射模块和单片机发码程序的客户。阅读目标是能看懂例程为什么这样写、波形为什么这样发、客户要改代码时应该改哪里。

学习阶段	客户应该能做到
先跑通	T1L 接到 DEMO 板, 按 KEY2 后串口提示发射, 接收端看到 11 22 33。
看懂波形	知道同步头、逻辑 0、逻辑 1、24bit 数据、重复帧分别是什么。
看懂程序	能说清 main.c、ALL.H、main_uart.c、rf.c 各自负责什么。
能改代码	会修改默认码值、按键数量、重复次数、发射间隔和目标 MCU 接口。
能排问题	能用串口和示波器判断是按键没触发、DAT 没波形、频率不一致, 还是接收端问题。

最小成功标准

第一次调试只追求一件事: T1L 模块接好、程序下载、串口有“无线发射 112233”提示, 并且配套接收端或开发助手能看到 11 22 33。先证明能发、能收、频率一致, 再做按键业务和功耗优化。

一句话理解 T1L

远-T1L 发射模块不负责“生成编码”。它只看 DAT 脚: DAT 为高时发射载波, DAT 为低时关闭载波。真正的编码由 MCU 按时间输出高低电平来完成。



2. 远-T1L 模块关键参数

项目	规格书参数/说明	初学者要注意
用途	小家电遥控专用 ASK/OOK 发射模块。	适合遥控器、开关量、简单无线按键。
工作频率	315MHz 或 433.92MHz 可选。	发射和接收频率必须一致。
工作电压	2.0-5.0V, 典型 3V; 极限 VDD -0.3 至 5.5V。	第一次建议用 DEMO 板默认 3.3V。
工作电流	约 9mA@3V, 30% 调制占空比。	电池和 LDO 要能承受发射瞬间电流。
待机电流	约 0.1uA。	低功耗产品还要让 MCU 睡眠。
输出功率	约 12dBm@3V, 2.4V 约 8dBm, 5V 约 14dBm。	距离不足先查天线, 不要先盲目升压。
调制方式	ASK/OOK。	DAT 高低电平控制有无载波。
传输速率	0.1-9.6kbps。	本例程 400us/1200us 时序在范围内。
天线阻抗	50 欧姆。	不接天线不要测试距离。
数据输入	DAT 为 3-5V 电平输入。	9-12V 编码 IC 输出必须串 51K。
尺寸	10.54 x 6.86 x 2.0mm。	布局时预留天线和焊接空间。
温度范围	-40 至 +85 degC。	户外/小家电环境也要验证整机。



T1L 引脚: 图为模块背面, D (DAT), V (VCC), G (GND), A (ANT)

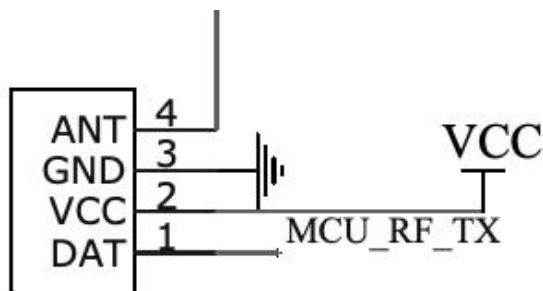
电源和 DAT 保护

T1L 的 VCC 不要超过规格, DAT 推荐 3V 电平。若使用 9-12V 编码 IC/MCU 直接驱动 DAT, 必须在 DAT 与编码 IC/MCU 之间串 51K 电阻保护模块。

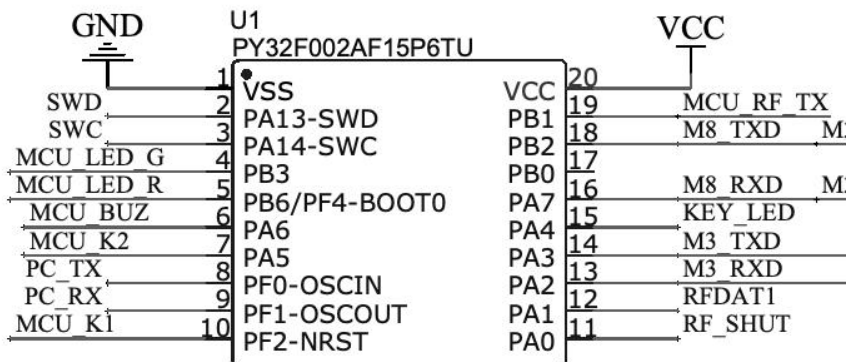


3. MCU_DEMO 板硬件连接

网络/器件	原理图连接	客户说明
M6 远-T1L 座	DAT/VCC/GND/ANT	T1L 插接或焊接位置。
MCU_RF_TX	PY32 PB1 -> M6 DAT	例程发射波形输出脚。
VCC	默认 R6=0R 接 V33, R7=NC	默认 3.3V。要改 5V 必须确认模块和 MCU IO。
Type-C + CH340N	USB 转串口	用于 9600bps 调试输出。
SWD	PA13/PA14	用于下载和调试 PY32F002A。
KEY2	原理图 KEY2 -> MCU_K2 -> PA6	客户包整理版工程使用 PA6 作为 KEY2 触发。
LED2/PB3	MCU_LED_G	发送时闪烁提示。



发射模块座 M6: DAT 接 MCU_RF_TX, VCC/GND/ANT 独立引出。



PY32F002A 周边关键网络: PB1 为 MCU_RF_TX; KEY2 网络在当前原理图上接 PA6。

按键脚位说明

工程已按 MCU_DEMO 原理图将 KEY2 触发脚校正为 PA6。原例程里曾写 PA5 低电平触发; 如使用不同批次 DEMO 或自己的 PCB, 以实际原理图为准, 确认 p_key2 宏对应的 IO 能被按键拉低。

4. ASK/OOK 和 1527 类编码原理

ASK/OOK 可以先按最简单的方式理解: DAT 高电平时, T1L 发射无线载波; DAT 低电平时, T1L 关闭载波。接收端看到的不是“字节”, 而是一串高低电平脉冲。接收端再根据脉冲宽度还原出 0 和 1。

概念	在本例程中的含义	为什么重要
同步头	高 400us, 低约 12400us。	低电平时间很长, 接收端用它判断一帧开始。
逻辑 1	高 1200us, 低 400us。	高电平长、低电平短。
逻辑 0	高 400us, 低 1200us。	高电平短、低电平长。
一帧	同步头 + 24bit 数据。	默认 0x11 0x22 0x33 共 24 位。
重复帧	TX1527() 默认重复 25 次。	无线容易受干扰, 接收端常要求连续两帧一致。
MSB first	每个字节从最高位 bit7 开始发。	示波器波形顺序要按 bit7 到 bit0 对照。



文字版时序图

一帧数据的结构:

同步头

H 400us + L 12400us

第 1 字节 d0, 共 8bit, 按 bit7 -> bit0 发送

第 2 字节 d1, 共 8bit, 按 bit7 -> bit0 发送

第 3 字节 d2, 共 8bit, 按 bit7 -> bit0 发送

bit=1: H 1200us + L 400us

bit=0: H 400us + L 1200us

为什么不要只发一帧

无线环境有噪声, 接收端可能误判出一帧假数据。工程上通常要求连续两帧一致才输出。发射端重复发送多帧, 是为了让接收端有足够机会完成确认。

5. 默认编码 11 22 33 是怎样发出去的

例程调用 TX1527(0x11, 0x22, 0x33)。这不是字符串“112233”, 而是 3 个十六进制字节。每个字节 8 位, 3 个字节一共 24 位。

十六进制	二进制	发送顺序
0x11	00010001	先发 bit7=0, 最后发 bit0=1。
0x22	00100010	先发 bit7=0, 最后发 bit0=0。
0x33	00110011	先发 bit7=0, 最后发 bit0=1。

字节	位	值	DAT 输出波形
d0=0x11	bit7	0	高 400us -> 低 1200us
d0=0x11	bit6	0	高 400us -> 低 1200us
d0=0x11	bit5	0	高 400us -> 低 1200us
d0=0x11	bit4	1	高 1200us -> 低 400us
d0=0x11	bit3	0	高 400us -> 低 1200us



d0=0x11	bit2	0	高 400us -> 低 1200us
d0=0x11	bit1	0	高 400us -> 低 1200us
d0=0x11	bit0	1	高 1200us -> 低 400us
d1=0x22	bit7	0	高 400us -> 低 1200us
d1=0x22	bit6	0	高 400us -> 低 1200us
d1=0x22	bit5	1	高 1200us -> 低 400us
d1=0x22	bit4	0	高 400us -> 低 1200us
d1=0x22	bit3	0	高 400us -> 低 1200us
d1=0x22	bit2	0	高 400us -> 低 1200us
d1=0x22	bit1	1	高 1200us -> 低 400us
d1=0x22	bit0	0	高 400us -> 低 1200us
d2=0x33	bit7	0	高 400us -> 低 1200us
d2=0x33	bit6	0	高 400us -> 低 1200us
d2=0x33	bit5	1	高 1200us -> 低 400us
d2=0x33	bit4	1	高 1200us -> 低 400us
d2=0x33	bit3	0	高 400us -> 低 1200us
d2=0x33	bit2	0	高 400us -> 低 1200us
d2=0x33	bit1	1	高 1200us -> 低 400us
d2=0x33	bit0	1	高 1200us -> 低 400us

示波器怎么对照

按键后先找最长的低电平，那就是同步头。同步头后面开始数 24 个数据位：长高短低是 1，短高长低是 0。按上表数完，应得到 00010001 00100010 00110011。



6. 初学者应该按什么顺序看源码

先后顺序	文件	先看什么	暂时不用纠结什么
1	程序说明.txt	硬件准备、PB1 发射、默认编码。	文字里原始 PA5 说明, 客户包已按原理图整理为 PA6。
2	ALL.H	p_rf_tx、p_key2、LED、GPIO 输入输出宏。	其它复用模板宏。
3	main.c	初始化流程、按键判断、TX1527 调用。	低功耗预留函数。
4	main_uart.c	USART1 9600 和 TIM1 1us 定时。	LPTIM/STOP 等低功耗预留代码。
5	rf.c	WAIT_US、TX_SYN、TX_BYTE、TX1527。	文件前半部分保留的接收解码函数。
6	py32f0xx_it.c	SysTick/USART/TIM 中断入口。	EXTI 接收相关入口, 发射入门可先忽略。

源码里为什么有接收函数

工程复用了远系列收发模板, rf.c 中保留 REC_RF_us_INT()、CHK_RF_DECODE() 等接收解码函数。做 T1L 发射入门时, 主线只看 TX_SYN()、TX_BYTE()、TX1527()。

7. 工程环境和初始化

项目	内容	初学者说明
工程路径	80_示例工程和安装包/.../MDK-ARM/Project.uvprojx	用 Keil 打开这个文件。
目标芯片	PY32F002Ax5	DEMO 板 MCU。
Keil 目标	PY32F002Ax5_Project	不要随意改 target。
宏定义	PY32F002Ax5, USE_FULL_LL_DRIVER	工程使用 PY32 LL 库。
Pack	客户包带 Puya.PY32F0xx_DFP.1.1.0.pack	若提示 1.2.3 缺失, 可安装更高版本或重新选择已安装 Pack。
输出	CreateHexFile=1	编译后生成 HEX, 便于量产烧录。
串口	USART1, 9600bps, 8N1	用于 printf 调试。
定时器	TIM1 预分频 8-1, ARR=50000-1	8MHz 时钟下 TIM1 每 1us 加 1。



8. main.c: 程序从哪里开始

main.c 主流程

```
int main(void)
{
    APP_SystemClockConfig();    // 系统时钟配置为 HSI 8MHz
    RCC_INI();                  // 打开 GPIO/USART/TIM1 时钟
    GPIO_INI();                 // 配置输入输出脚
    APP_ConfigUsart(USART1);    // 串口 9600, 8N1
    SET_PF1_for_UART_or_GPIO(1); // 把串口映射到 DEMO 板 USB
    APP_ConfigTIM1();           // TIM1 配成 1us 计数
    systick_init();             // SysTick 每 10ms 中断一次
    myDelay_ms(50);
    LL_GPIO_SetOutputPin(p_led1);
    printf("蜂鸟无线 RF433 发射测试\r\n");

    while (1) {
        if (p_key2 == 0) {
            TX1527(0x11, 0x22, 0x33);
        }
        myDelay_ms(100);
    }
}
```

代码	作用	初学者注意
APP_SystemClockConfig()	把系统时钟设为内部 HSI, SystemCoreClock=8000000。	后面 TIM1 的 1us 计时依赖 8MHz。
RCC_INI()	打开 GPIOA/GPIOB/GPIOF、USART1、TIM1 时钟。	外设没开时钟, GPIO/串口/定时器都不会工作。
GPIO_INI()	按 ALL.H 中的宏配置输入输出。	p_rf_tx 和 p_key2 是否正确, 主要看 ALL.H。
APP_ConfigUsart()	配置 9600bps 串口。	串口没有输出时先查 CH340 和波特率。



SET_PF1_for_UART_or_GPIO (1)	把 PF0/PF1 配成 USART1 复用。	让 printf 能从 Type-C 串口出来。
APP_ConfigTIM1()	让 TIM1 以 1us 为单位计数。	发码时所有 400/1200/12400us 都靠它。
if (p_key2 == 0)	检测按键是否按下。	按键是上拉输入, 按下接地, 所以按下为 0。
TX1527(0x11,0x22,0x33)	发送默认 24bit 编码。	客户改码值通常先改这里。
myDelay_ms(100)	每次循环后等 100ms。	避免按键按住时过于密集地重复触发。

9. ALL.H: 引脚宏怎么看

资料包 ALL.H 关键宏

<pre>#define GPIOA_INPUT (1<<1 1<<5 1<<6) // PA6=KEY2, PA5 保留为输入 #define p_key2 pA_in_(6) // 读 PA6 电平, 按下为 0 #define GPIOB_INPUT (1<<0 1<<2) #define GPIOB_OUTPUT (0xff-GPIOB_INPUT) #define p_led2 GPIOB,1<<3 #define p_rf_tx GPIOB,1<<1 // PB1 -> MCU_RF_TX -> T1L DAT</pre>		
宏	含义	客户常改场景
p_rf_tx	发射 DAT 输出脚, 当前为 PB1。	客户换 PCB 时, 把它改成自己的 DAT IO。
p_key2	触发按键输入脚, 当前为 PA6。	客户换按键脚时改这里, 同时把该脚加入输入宏。
GPIOA_INPUT	GPIOA 哪些脚配置为输入。	p_key2 改到 PAX 时, 必须包含 1<<x。
GPIOB_OUTPUT	GPIOB 除输入脚外配置为输出。	p_rf_tx 在 PB1, 所以 PB1 必须是输出。
p_led2	发送时闪烁的 LED。	没有 LED 时可删除或换脚。



10. main_uart.c: TIM1 为什么是 1us

发射编码最怕时间不准。例程用 TIM1 做微秒计数, WAIT_US(400) 就表示等待 400us。

TIM1 关键配置

```

TIM1CountInit.Prescaler = 8 - 1;
TIM1CountInit.Autoreload = 50000 - 1;
LL_TIM_Init(TIM1, &TIM1CountInit);
LL_TIM_EnableCounter(TIM1);

```

参数	本例程数值	解释
系统时钟	8MHz	APP_SystemClockConfig() 设置。
Prescaler	8-1	8MHz / 8 = 1MHz。
计数周期	1us	1MHz 计数频率意味着每 1us 加 1。
Autoreload	50000-1	最多可连续计约 50ms, 足够覆盖 12.4ms 同步低电平。
移植注意	按新时钟重算	如果系统时钟不是 8MHz, Prescaler 必须重算。

修改主频后的公式

如果定时器时钟为 timer_clock_hz, 要让 TIM 每 1us 计数一次, 则 $\text{Prescaler} = \text{timer_clock_hz} / 1000000 - 1$ 。STM32 还要注意 APB 分频后 TIM 时钟可能翻倍。

11. rf.c: WAIT_US() 微秒等待

WAIT_US() 原理

```

void WAIT_US(u16 t)
{
    TIM1->CNT = 0;
    while (TIM1->CNT < t);
}

```



代码	作用	初学者注意
TIM1->CNT = 0	把 TIM1 当前计数清零。	每次等待都从 0 开始, 比较好理解。
while (TIM1->CNT < t)	一直等到计数值达到 t。	因为 TIM1 是 1us 计数, 所以 t=400 就是约 400us。
函数期间 MCU 忙等	等待时 CPU 不做其它事。	发码时间短, 入门例程可以接受; 复杂产品可改用定时器输出。

12. rf.c: TX_SYN() 同步头

同步头函数

```
void TX_SYN(void)
{
    LL_GPIO_SetOutputPin(p_rf_tx);
    WAIT_US(400);
    LL_GPIO_ResetOutputPin(p_rf_tx);
    WAIT_US(400 * 31);
}
```

步骤	DAT 电平	持续时间	接收端看到什么
1	高	400us	短高电平。
2	低	12400us	很长低电平, 作为一帧开始标志。

为什么同步低电平这么长

普通数据位的低电平最长 1200us, 而同步头低电平约 12400us, 差别很大。接收端只要发现一个明显超长低电平, 就知道下一段开始是 24bit 数据。



13. rf.c: TX_BYTE() 怎样发送 8 个 bit

TX_BYTE() 原代码逻辑

```
void TX_BYTE(u8 d)
{
    u8 i;
    for (i = 0; i < 8; i++) {
        if ((d & 0x80) != 0) {
            LL_GPIO_SetOutputPin(p_rf_tx);
            WAIT_US(1200);
            LL_GPIO_ResetOutputPin(p_rf_tx);
            WAIT_US(400);
        } else {
            LL_GPIO_SetOutputPin(p_rf_tx);
            WAIT_US(400);
            LL_GPIO_ResetOutputPin(p_rf_tx);
            WAIT_US(1200);
        }
        d <<= 1;
    }
}
```

代码	中文解释	为什么这样写
for (i=0; i<8; i++)	一个字节有 8 位, 所以循环 8 次。	每循环一次只发送 1 个 bit。
d & 0x80	检查当前字节最高位 bit7 是否为 1。	0x80 的二进制是 10000000。
高 1200us / 低 400us	当前位是 1。	长高短低表示逻辑 1。
高 400us / 低 1200us	当前位是 0。	短高长低表示逻辑 0。
d <<= 1	发送完最高位后, 把下一个 bit 移到最高位。	这样下一轮仍然用 d&0x80 判断。
MSB first	从 bit7 发到 bit0。	示波器对照时不要从最低位开始数。

d <<= 1 的直观理解

以 d = 0x11 = 00010001 为例:



第 1 轮: d&0x80 看 bit7 = 0, 发 0, 然后 d 左移
第 2 轮: 看 bit6 = 0, 发 0, 然后 d 左移
第 3 轮: 看 bit5 = 0, 发 0, 然后 d 左移
第 4 轮: 看 bit4 = 1, 发 1, 然后 d 左移
...
第 8 轮: 看 bit0 = 1, 发 1

14. rf.c: TX1527() 发送完整一组码

TX1527() 原代码逻辑

```
void TX1527(u8 d0, u8 d1, u8 d2)
{
    u8 i;
    printf("\r\n 无线发射%02X%02X%02X", d0, d1, d2);
    for (i = 0; i < 25; i++) {
        LL_GPIO_TogglePin(p_led2);
        TX_SYN();
        TX_BYTE(d0);
        TX_BYTE(d1);
        TX_BYTE(d2);
    }
    printf("   发射结束", NULL);
    LL_GPIO_ResetOutputPin(p_led2);
}
```

代码

作用

初学者注意



printf("无线发射%02X%02X%02X")	把本次发射的 3 个字节打印到串口。	看到无线发射 112233, 说明程序至少进入了发射函数。
for (i=0; i<25; i++)	重复发送 25 帧。	初次调试要多发几帧, 方便接收端确认。
LL_GPIO_TogglePin(p_led2)	每帧翻转 LED。	LED 闪烁说明正在发射。
TX_SYN()	先发同步头。	接收端靠同步头定位一帧开始。
TX_BYTE(d0/d1/d2)	连续发 3 个字节, 共 24bit。	默认就是 0x11、0x22、0x33。
ResetOutputPin(p_led2)	发完后关闭 LED。	结束后回到空闲状态。

15. 主流程和发射流程图

主流程

主程序流程:

上电

|

v

初始化时钟/GPIO/串口/TIM1/SysTick

|

v

串口打印“蜂鸟无线 RF433 发射测试”

|

v

循环检测 p_key2

|

+++ p_key2 != 0: 未按下, 继续等待

|

+++ p_key2 == 0: 调用 TX1527(0x11,0x22,0x33)

发射流程

TX1527 发射流程:



重复 25 次:

1. LED 翻转
2. TX_SYN() 发同步头
3. TX_BYTE(0x11) 发第 1 字节
4. TX_BYTE(0x22) 发第 2 字节
5. TX_BYTE(0x33) 发第 3 字节

发完后:

LED 关闭, 串口打印“发射结束”

16. 常见二次开发怎么改

需求	推荐修改位置	示例	注意事项
改默认码值	main.c	TX1527(0x12,0x34,0x56);	接收端应看到 12 34 56。
增加 2 个按键	ALL.H + main.c	if(KEY1 按下)发 A1; if(KEY2 按下)发 A2;	每个按键先消抖。
增加 4/8 个按键	main.c 建立按键表	按键号 -> 三字节码值。	不要在多个 if 中重复发太久。
改变重复次数	rf.c 中 for(i=0;i<25;i++)	改为 5、10、30 等。	少了可能收不到, 多了会更耗电。
改变发射间隔	main.c 的 myDelay_ms(100)	改为 200ms 或按键释放后再允许下一次。	防止长按连续狂发。
上电自动发一次	main.c 初始化后调用一次 TX1527()	printf 后 TX1527(...);	量产测试可用, 产品上谨慎使用。
串口输入码值后发射	READ_UART() + main.c	解析 AT+TX=112233。	先做输入合法性判断。
换 MCU	重写 GPIO 和 WAIT_US	保留 TX_SYN/TX_BYTE 思路。	先用示波器验 400/1200/12400us。

多按键伪代码

两个按键示例思路:

```
if (KEY1_IS_DOWN()) {
```



```
TX1527(0x11, 0x22, 0x31);  
}  
  
if (KEY2_IS_DOWN()) {  
    TX1527(0x11, 0x22, 0x32);  
}
```

建议: 按键先消抖, 发射完成后等待按键释放, 再允许下一次发射。

17. 与接收端联调的正确顺序

1. 先用市面成熟 1527 遥控器验证接收端, 确认接收模块、频率、天线和解码程序没有问题。
2. 再用 T1L + PY32 例程发默认码 11 22 33, 看接收端是否输出相同码值。
3. 如果接收不到, 用示波器先看 PB1/DAT 是否有规律同步头和数据脉冲。
4. 有波形但收不到时, 重点查频率、天线、接收端门限、码型和重复帧。
5. 没有波形时, 重点查程序是否运行、按键是否触发、p_key2 是否对应实际 IO、TIM1 是否 1us。



18. 初学者常见误区

误区	正确理解	怎么避免
以为 T1L 自己会编码	T1L 只调制 DAT, 编码由 MCU 产生。	先看 PB1/DAT 波形。
频率不一致	315 和 433.92MHz 不能互通。	发射/接收模块下单和测试都核对频率。
不接天线测距离	天线直接影响效果。	先接天线近距离成功, 再测距离。
只发一帧	接收端常要求连续两帧一致。	初次调试重复 25-30 帧。
改主频不改 TIM1	WAIT_US 依赖 1us 计数。	改时钟后重新计算 Prescaler。
按键 IO 不一致	代码读的脚必须和原理图一致。	查 p_key2 宏和实际按键网络。
同时调发射和接收	问题范围太大。	先成熟遥控验证接收, 再调 MCU 发射。
DAT 高压直连	9-12V 可能损坏 T1L。	高压编码 IC 输出串 51K。
看到串口就认为无线一定发出	串口只能说明程序进入发射函数。	还要用示波器或开发助手确认。

19. 产品设计建议

项目	建议
电源	T1L 发射瞬间约 9mA@3V, 电池和 LDO 要能承受瞬态; 靠近模块放 0.1uF + 10uF 去耦。
地线	T1L GND 与 MCU GND 必须可靠连接, 避免长细地线造成波形毛刺。
DAT	默认空闲低电平。走线短、远离大电流和电感负载; 高压编码 IC 输出串 51K。
天线	预留匹配和实测空间; 塑料外壳、手握位置、金属件都会影响距离。
按键	做硬件或软件消抖。发射期间可屏蔽重复触发, 避免连续占空比过高。
低功耗	按键唤醒后发码, 发完关闭不必要外设再睡眠; T1L 自身待机很低, 但 MCU 也要睡。
量产	建议用开发助手/信号助手做频率、码值、信号强度和天线一致性抽检。



20. 交付前确认

确认项	通过标准
工程可编译	指定 Keil/Pack 环境下 0 Error, 生成 HEX。
串口提示	上电打印测试信息, 触发后打印无线发射 112233。
DAT 波形	PB1/DAT 有同步头和 24bit 数据脉冲。
接收码值	近距离接收端稳定看到 11 22 33。
电源	发射时 VCC 不明显下跌; 供电未超规格。
天线	已焊接或接入合适天线, 距离测试有记录。
按键	p_key2 与实际按键 IO 一致, 按键无误触发。
边界	明确本例程是固定码发射示例, 不含学习码管理、不含产品业务逻辑。

附录 A: 给源码保留的注释

放在 rf.c 开头的注释。

注释

```
/*  
* 远-T1L 发射模块 DAT 输出说明  
* 硬件连接: PY32F002A PB1 -> MCU_RF_TX -> T1L DAT  
* 空闲电平: 低电平  
* 计时方式: TIM1 每 1us 计数一次, WAIT_US() 用于产生精确高低电平时间  
* 默认编码: TX1527(0x11, 0x22, 0x33)  
* 帧结构: 同步头 + 24bit 数据; 当前示例重复发送 25 帧  
* 逻辑 1: 高 1200us, 低 400us  
* 逻辑 0: 高 400us, 低 1200us  
* 注意事项: 修改系统主频后, 必须重新确认 TIM1 是否仍为 1us 计数  
*/
```