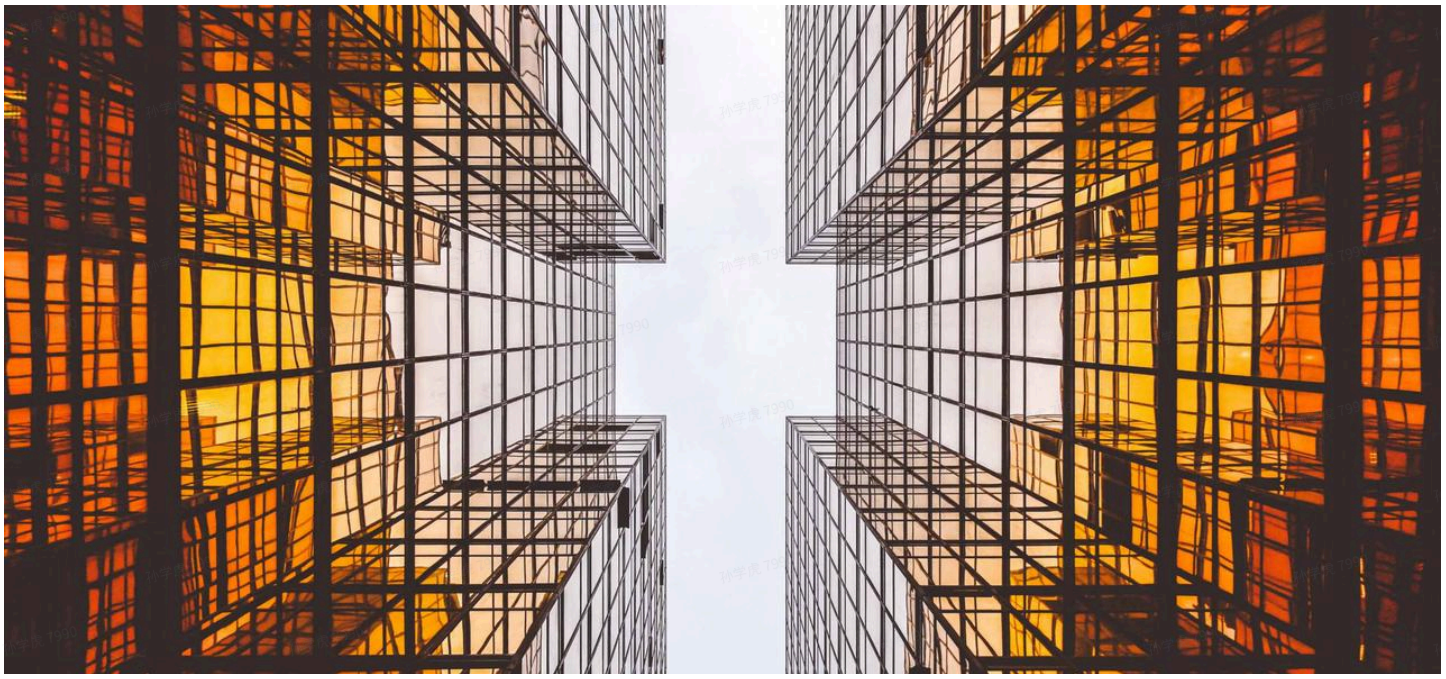


# Break 回调被 GPIO 中断抢占，LBKDIE 丢失的排查和修复

## 问题描述

如下图所示，MCU 在 Break 低电平持续**约 11 bit** 时检出 Break 并进入中断处理（实测 T1 ≈ 11.24 bit / 585 µs，此时主机 ≥13 bit 的 Break 场尚未结束）。在回调开启 GPIO 下降沿中断至关闭 `LBKDIE` 的敏感窗口 **[T1~T5]** 内，若 LIN 总线上产生毛刺（下降沿），将触发高优先级 GPIO 中断抢占并引发两处逆序覆写，使外设陷入永久死锁：

代码块

```
[A] LIN_RX 帧头波形与噪声触发区间（1=隐性，0=显性；约 11b 进 Break 中断）：
                                        |<- T1~T5 ->|
RX=1 ──┐                                              ┌──────────────┐  ┌──┐
       └──┘└──┘└──┘└──┘└──┘└──┘└──┘└──┘└──┘└──┘└──┘└──┘└──┘└──┘└──┘└──┘└──┘
       |  Break Field (>=13b)          ┌┐              |Delimiter| S |b0 |...|Stp|PID
RX=0   └──────────────────────────────┘└──────────────┘ (1~4 b) └──┘└──┘└──┘
       |<------ ~11 bits ------>▲   ▲     ▲     ▲          ▲
                                |   |     |     |          └─ Sync(0x55) 起始沿
                                |   |     |     └─ T5: 关 LBKDIE=0
                                |   |     └─ [!] 毛刺下降沿 -> 立即触发抢占!
                                |   └─ T1: 开 GPIO 下降沿中断
```

```
                                  └─ T0: 低电平约 11b 检出 Break，进 UART0_IRQn

[B] 双中断抢占与逆序覆写时序（时间自上而下 T0 -> T6）:
  UART0_IRQn (Break ISR, Prio 1/2)        GPIO_IRQn (Edge ISR, Prio 0)
  ──────────────────────────────────────────────────────────────────────
  [T0] 约 11b 进中断，调 Callback()
   │   (此时驱动未切状态，State=IDLE!)
  [T1] 切 MUX=GPIO，开下降沿中断
   │   ---------------------------------> [T2] 毛刺下降沿触发高优抢占!
   │        (UART Break ISR Paused)         │   HandleCapture() 校验状态:
   │                                      [T3] 见 State==IDLE (!=SYNC)
   │                                        │   误判返回 LIN_SYNC_ERROR!
   │                                      [T4] 调 GotoIdleState() 恢复:
   │                                        │   - 置 LBKDEN=1, LBKDIE=1
   │   <---------------------------------  └─  - 关 GPIO 中断，切回 UART
  [T5] 回到 Callback 继续执行:
   │   关 LBKDIE=0  ==> [覆写1: T4 刚打开的 LBKDIE=1 被清零!]
  [T6] 返回驱动 ProcessBreakDetect:
   ▼   设 RECV_SYNC ==> [覆写2: T4 刚重置的 IDLE 被改成 RECV_SYNC!]

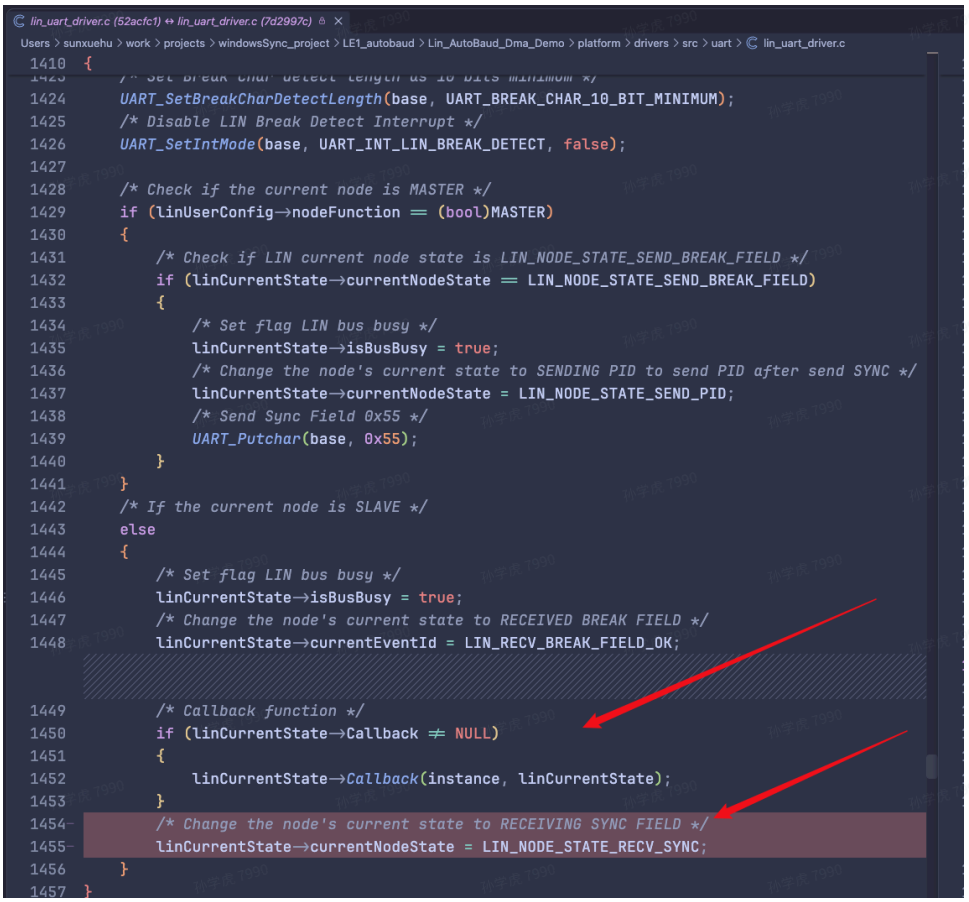
  ──────────────────────────────────────────────────────────────────────
  [死锁终态] LBKDEN=1(只认Break) + LBKDIE=0(不进中断) + GPIO=OFF
             + State=RECV_SYNC(无超时自愈) ===> 永久无响应直至上电复位!

  ──────────────────────────────────────────────────────────────────────
```

## 主要矛盾

- **Break Callback 内执行关闭 LBKDIE 的指令顺序问题：**在 `LinSync_EnableGpioCapture()` 中，先开启 GPIO 下降沿中断（`PCTRL_INT_FALLING_EDGE`），后关闭 Break 中断使能（`UART_SetIntMode(..., UART_INT_LIN_BREAK_DETECT, false)`）。实际上驱动在进入 Break ISR 时已关闭过 `LBKDIE`；应用层将关闭 `LBKDIE` 放在开启高优先级 GPIO 中断之后，一旦在这两步之间被毛刺触发的 GPIO 中断抢占并调用 `GotoIdleState()` 重新打开 `LBKDIE=1`，恢复执行后就会把刚打开的 `LBKDIE` 覆写关断为 0，造成 **LBKDEN=1 且 LBKDIE=0** 的死锁。

## 次要矛盾




- **驱动层“先执行 Callback、后转变 LIN 状态”引发竞态：**这使客户层代码产生两重竞态：
  ① **回调期状态滞后导致误判报错**：Callback 执行期间状态仍为 `IDLE`，开启 GPIO 下降沿中断后一旦被毛刺抢占，GPIO ISR 内 `LinSync_HandleCapture()` 检查 `currentNodeState != RECV_SYNC` 必然误判为 `LIN_SYNC_ERROR` 并触发 `GotoIdleState()`；
  ② **回调返回后状态被驱动强行覆写**：抢占中断内 `GotoIdleState()` 已将状态重置为 `IDLE` 并关闭 GPIO 捕获，但当 `Callback()` 返回驱动时，驱动仍无条件将 `IDLE` 覆写为 `RECV_SYNC`（SDK 1.3.1 的 `TimeoutService` 不处理 `RECV_SYNC` 超时）。

# 板级复现

- **方法一（总线毛刺注入复现）：**主机正常周期发送 LIN 帧，对 LIN 总线注入高频随机毛刺干扰。当毛刺产生的下降沿恰好落在 Break 中断（约 11 bit 检出）后的回调窗口 **[T1~T5]**时，即可复现从机永久无响应。
- **方法二（调试器断点 100% 必现法）：**在 `LinSync_EnableGpioCapture()` 中的 `UART_SetIntMode(LIN_SYNC_UART_BASE, UART_INT_LIN_BREAK_DETECT, false);` 打断点。主机发帧命中该断点时，GPIO 下降沿中断已打开，且总线上紧随其后的 Sync 下降沿已到达并使 GPIO 中断挂起（Pending）。此时删掉断点并按 Go 全速运行：CPU 立即先进入高优先级 GPIO 中断（因 `State == IDLE != RECV_SYNC` 调 `GotoIdleState()` 置 `LBKDEN=1, LBKDIE=1`），返回后执行该语句将 `LBKDIE` 关为 0，100% 必现死锁（在调试器中将 `UART0->INTE` 写为 `0x821` 补上 `LBKDIE=1` 后立即恢复）。

## 复现记录

围绕 `T1/T5` 硬件打点（`PTC14`）对 **[T1, T5]** 敏感窗口开展毛刺遍历（124 次毛刺注入 + 340 个干净帧），实测窗口分布、临界波形与终态如下：

代码块

```
[C] 实测时间轴与 [T1, T5] 指令窗口 (19,200 bps, 1 bit = 52.08 us):
t=0(Break)                T1_hw     T1_mk     T5_hw   T5_mk
──┐                         |         |         |       |
  |   Break (16b, ~833us)   |<-~0.1us>|         |<-1ins>|
  └─────────────────────────┴─────────┴─────────┴───────┴──────────
  |<-- 583.6~587.1 us (med 585.4) -->|         |       |
  |    11.21~11.27 b, jitter ~3.5us  |         |       |
                           |<---- ~0.5 us ---->|       | 真实死锁窗
                                      |<-- 0.78~0.80 -->| 标记高电平(us)
  └─ ~573us(11b) + ~11us SW ->        | T1_hw: SetPinIntSel(FALLING)
                                      | T1_mk: PTC14=1 (PSOR, +4 ins)
                                      | T5_hw: INTE&=~LBKDIE
                                      | T5_mk: PTC14=0 (PCOR, +1 ins)

[D] 5类毛刺命中区间 (dt = 毛刺下降沿 - T1_mk, n=124) 与 PTC14 形态:
                     T1_hw  T1_mk      T5_hw   T5_mk
                       |      |          |       |
 (1) dt<=-0.399us *    |      |          |       |      54/54 通过
 (2) [T1_hw..T1_mk)    |  *   |          |       |       6/6  死锁(T1推迟)
 (3) -0.126~+0.380us   |      |    *     |       |      10/10 死锁(撑宽)
 (4) +0.419~+0.661us   |      |          |   *   |       7/7  恢复(撑宽+双)
 (5) dt>=+0.680us      |      |          |       |*     47/47 拒帧后恢复
                       |      |          |       |
──────────────────────────────────────────────────────────────────
──────
[实测结论] 死锁窗 [T1_hw, T5_hw] 约 0.5 us (窗内 16/16, 窗外 0/108):
 - 死锁终态(16/16): State=0x04(RECV_SYNC) CTRL1=0x40000400(LBKDEN=1)
   INTE=0x21(LBKDIE=0) GPIOA_PCR2=0 PCTRLA_PCR2=0x600 Autobaud=1
 - 恢复验证: 不复位写 INTE=0x821 -> 下帧立即回复, State=0x02(IDLE)
──────────────────────────────────────────────────────────────────
──────
```

## 实测波形实证（图 1～图 4）

**图 1（基线实测）**：实测 **T1 ≈ 585.4 µs (11.24 bit)** 位于 Break 尾段，干净 `[T1_mk, T5_mk]` 脉宽 `0.78~0.80 µs`。




**图 2（必锁两类形态与死锁静默）**：毛刺落入 `[T1_mk, T5_hw]` 时脉宽被 GPIO ISR 撑至 **~17.1 µs**（10/10 死锁）；落入开中断与 `PSOR` 间缝隙时 `T1` 推迟 **~16.5 µs**（6/6 死锁）；死锁后干净帧无回

复且不再出现标记。

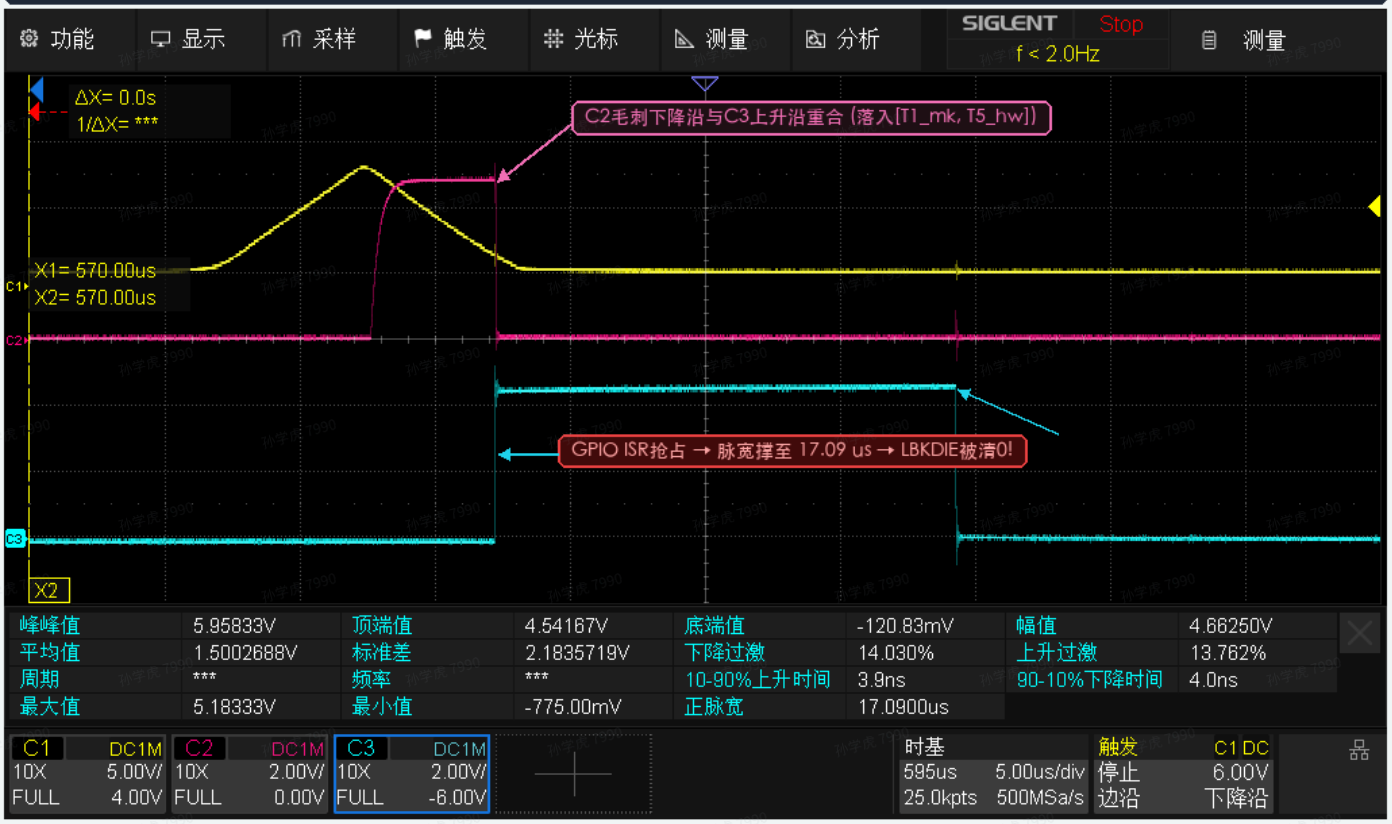
## 图2 死锁实测两类形态（区间②/③, 16/16 必锁）与死锁后永久静默表现

通道: C1(黄)=LIN总线 5V/div(+4V), 6V↓触发　C2(粉)=DUT RX 2V/div(0V), 2.5V判决　C3(青)=PTC14标记 2V/div(-6V)

帧头: 19,200 bps (1b=52.08us) | 16b Break (~833us) | ID 0x3D | 注入毛刺=Break内7us隐性脉冲 (RX约4.5us)

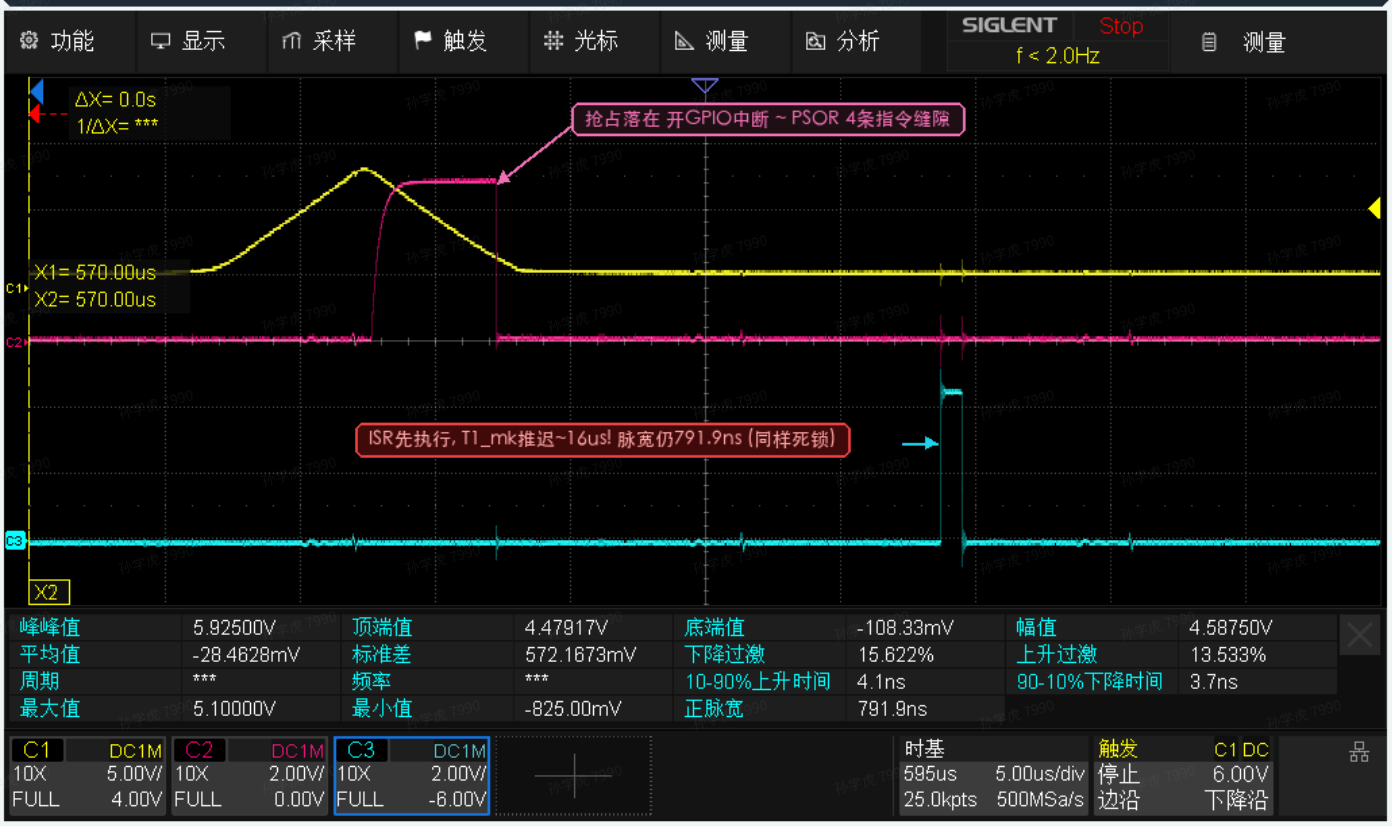
### A. 区间(3) 命中[T1_mk, T5_hw] (10/10 死锁, 5 us/div): 脉宽拉至 17.09 us

死锁: INTE=0x21 / RECV_SYNC



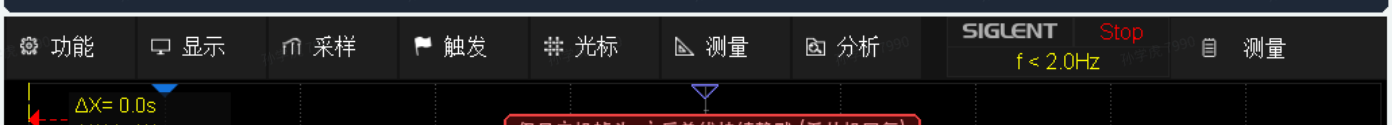

### B. 区间(2) 命中[T1_hw, T1_mk)前缘缝隙 (6/6 死锁, 5 us/div): T1 推迟 ~16 us

死锁: INTE=0x21 / RECV_SYNC




### C. 死锁后发送干净帧 (1 ms/div): 无回复 + C3 无脉冲 (LBKDIE=0)

持续无响应: 0x21 / RECV_SYNC



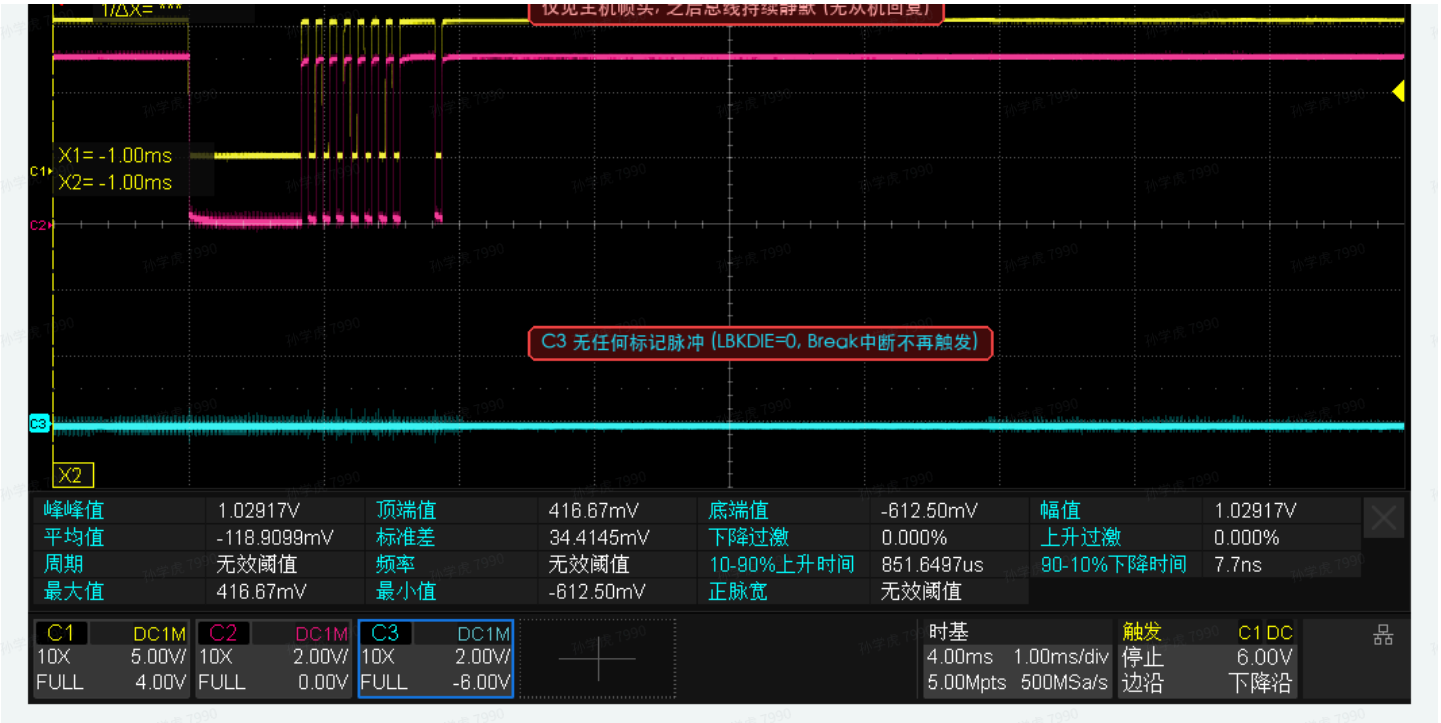


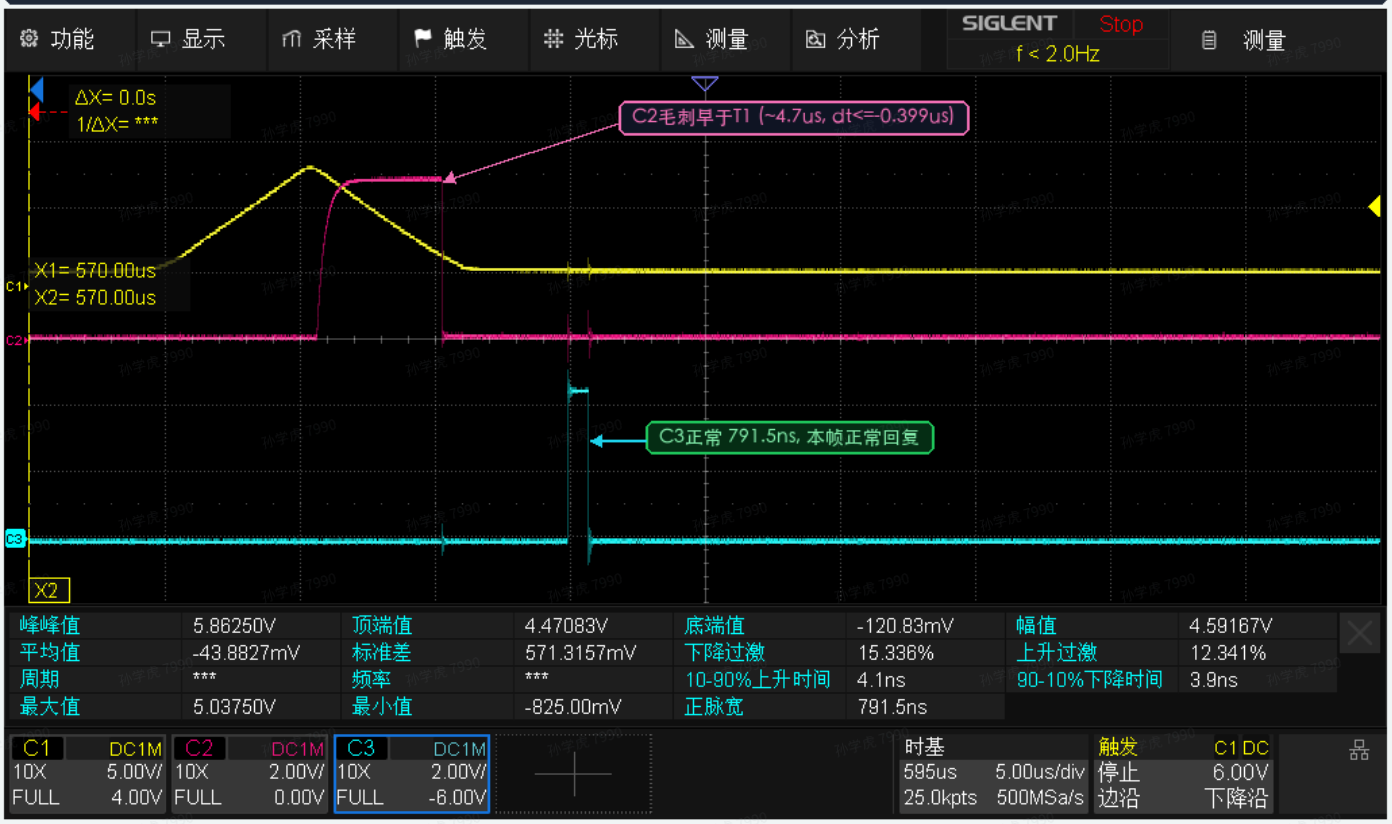

**图 3（边界对照与右界定位）**：`04` 早于 T1（54/54 通过）、`09` 晚于 T5（47/47 恢复）；`08` 晚命中写 `INTE` 后至 `PCOR` 前缝隙，虽同样撑宽至 `~17.1 µs`（扫描中随后可见第二个标记脉冲）并发生 `RECV_SYNC` 覆写，但因 **LBKDIE=1** 未被清零，7/7 均自行恢复——证明 **LBKDIE=0** 覆写是死锁必要条件。

## 图3 窗口边界实证 (区间①/④/⑤)：死锁窗右界位于写 INTE (真实 T5)，LBKDIE=0 是死锁必要条件

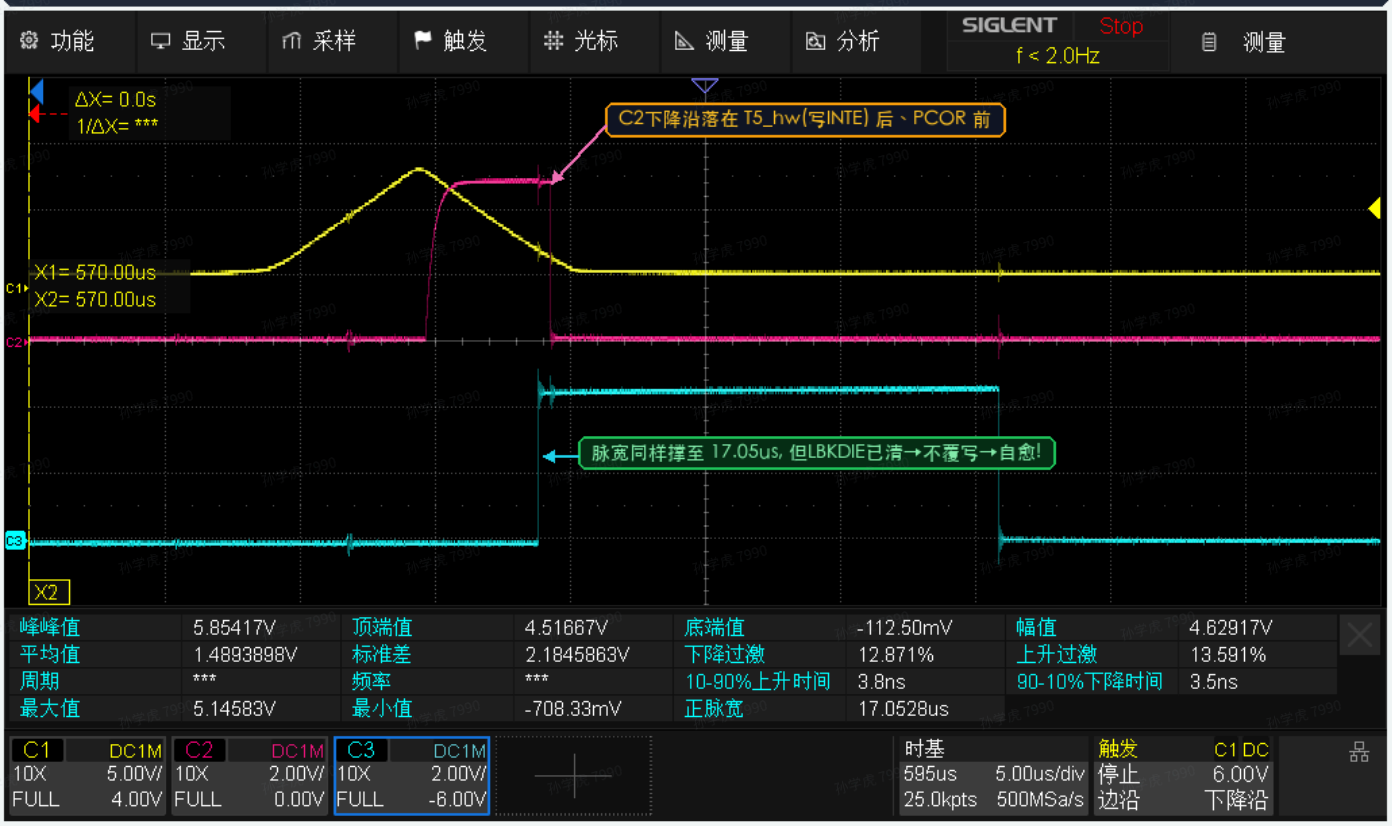
通道: C1(黄)=LIN总线 5V/div(+4V), 6V↓触发　C2(粉)=DUT RX 2V/div(0V), 2.5V判决　C3(青)=PTC14标记 2V/div(-6V)

帧头: 19,200 bps (1b=52.08us) | 16b Break (~833us) | ID 0x3D | 注入毛刺=Break内7us隐性脉冲 (RX约4.5us)

A. 区间(1) 早于 T1 (54/54 通过, 5 us/div): 脉宽 791.5 ns　恢复/通过: 0x821 / IDLE




B. 区间(4) 晚命中 [T5_hw, T5_mk] (7/7 恢复, 5 us/div): 虽撑至 17.05 us 但未锁　不死锁/恢复: 0x821 / IDLE




C. 区间(5) 晚于 T5 (47/47 恢复, 5 us/div): 脉宽 790.4 ns　恢复: 0x821 / IDLE



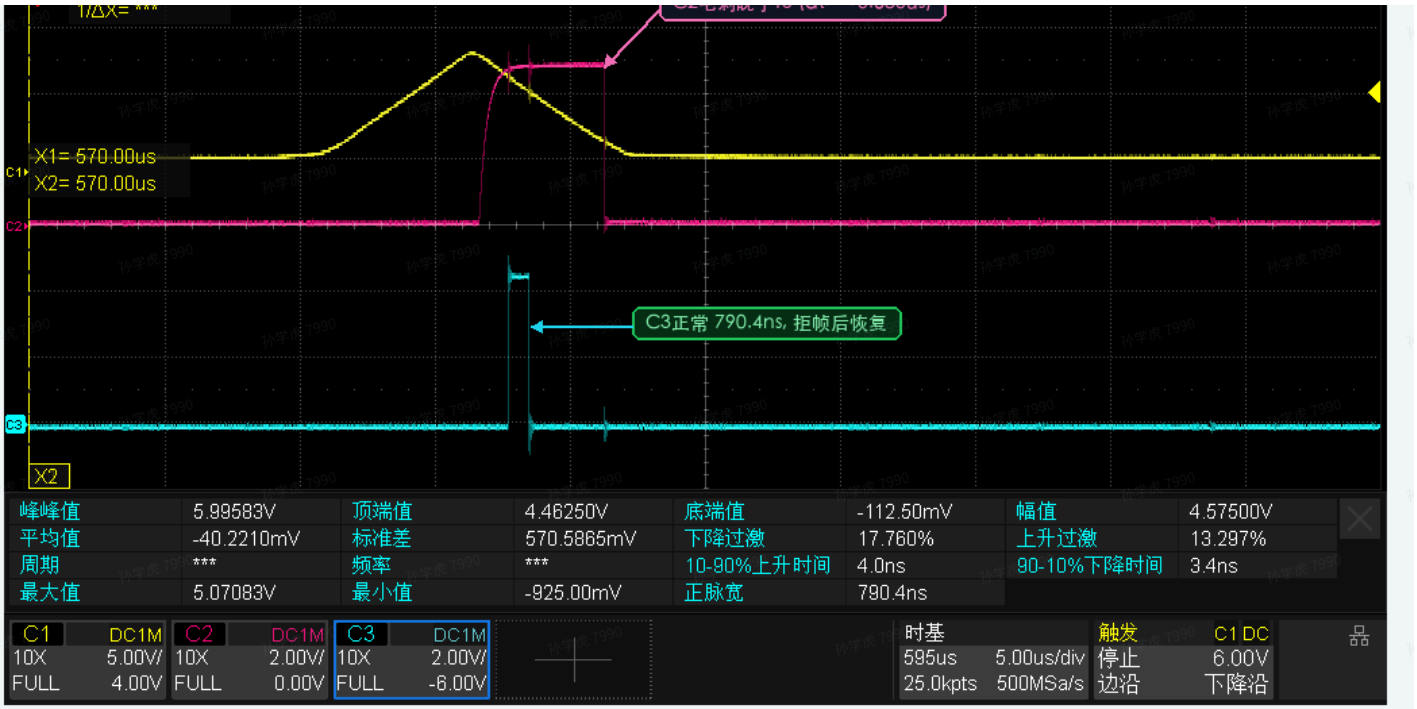
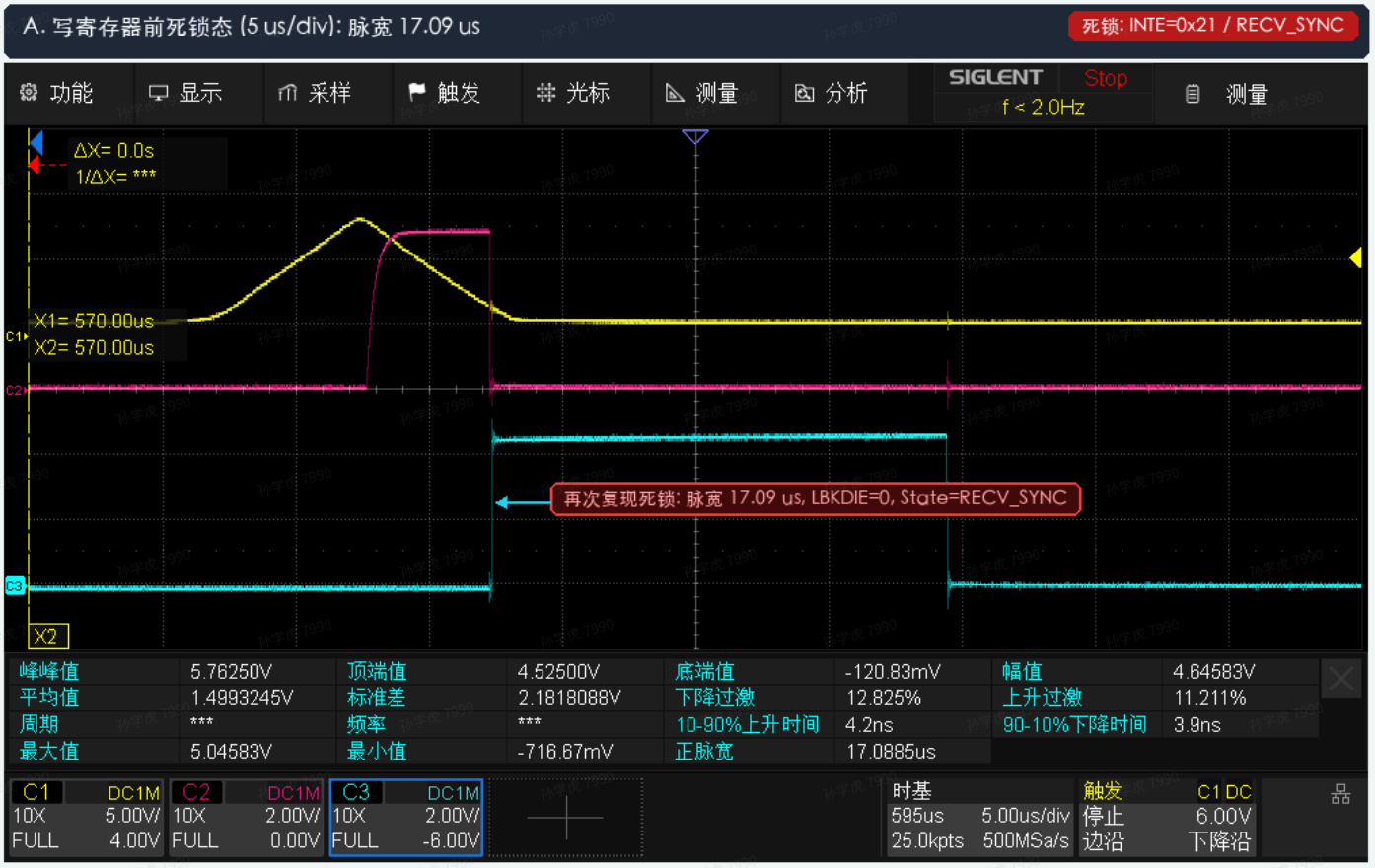


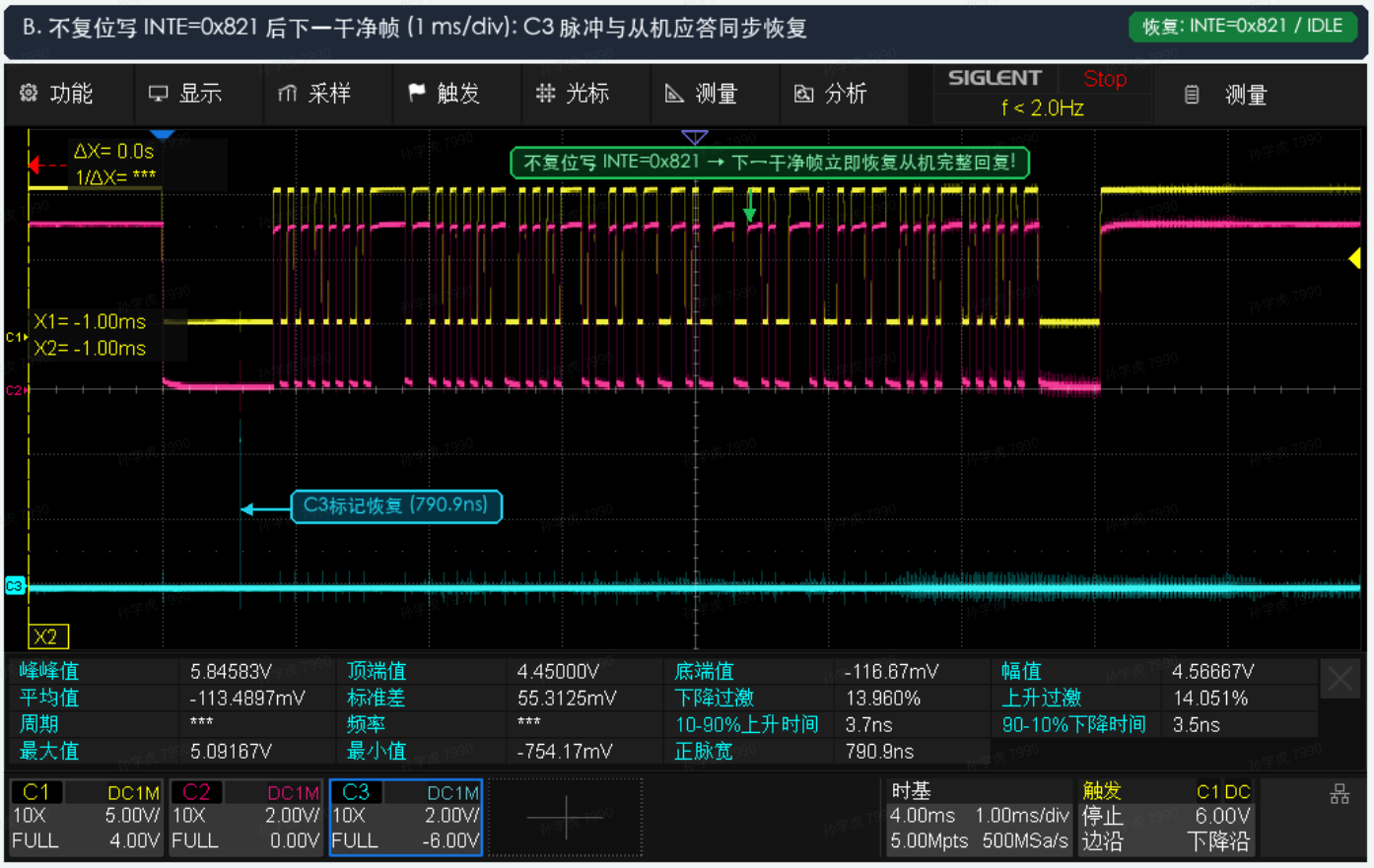

**图 4（不复位写寄存器恢复）**：死锁后保持现场不复位，仅写入 `UART0->INTE = 0x821` 补回 `LBKDIE=1`，下一个干净帧即恢复标记脉冲与完整从机应答，状态回到 **IDLE (0x02)**。

图4 恢复验证：死锁后不复位仅写 UART0->INTE=0x821 (补回 LBKDIE=1)，下帧立即恢复

通道: C1(黄)=LIN总线 5V/div(+4V), 6V↓触发　C2(粉)=DUT RX 2V/div(0V), 2.5V判决　C3(青)=PTC14标记 2V/div(-6V)

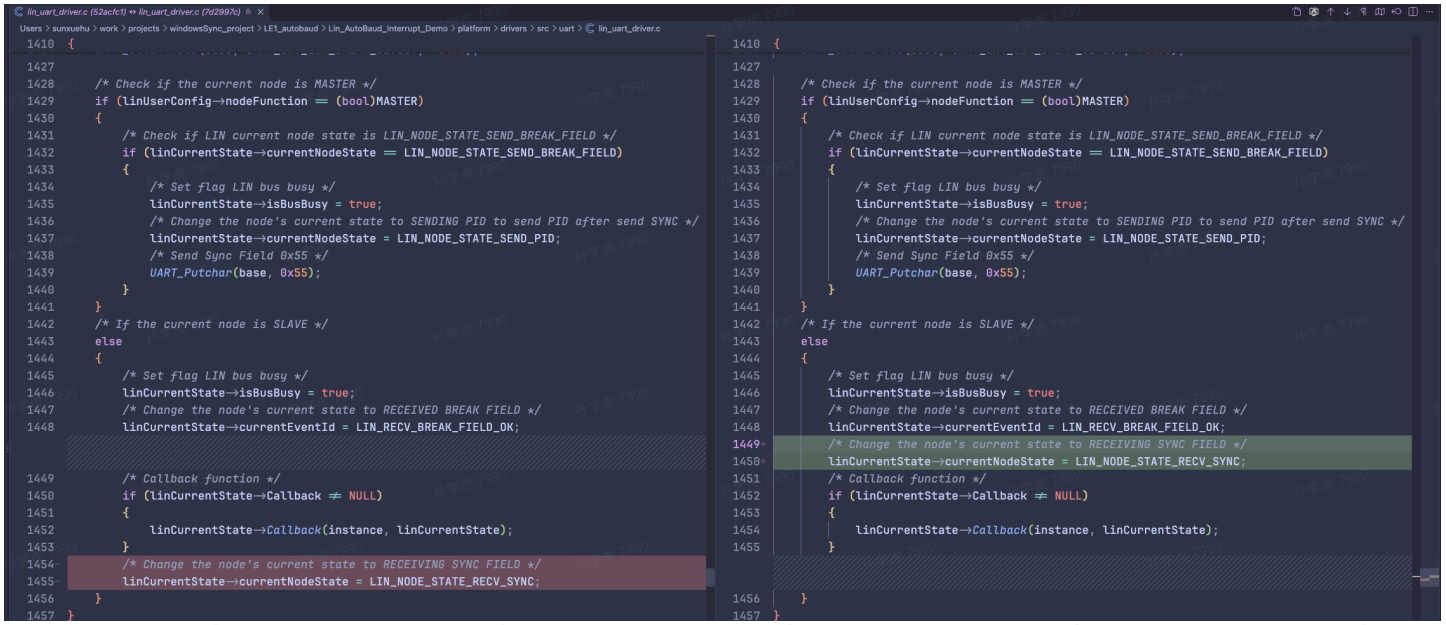
帧头: 19,200 bps (1b=52.08us) | 16b Break (~833us) | ID 0x3D | 注入毛刺=Break内7us隐性脉冲 (RX约4.5us)




# 修复方式

- **1. 解决次要矛盾（消除状态滞后与回调竞态）：**



```
/* Check if the current node is MASTER */
if (linUserConfig->nodeFunction == (bool)MASTER)
{
    /* Check if LIN current node state is LIN_NODE_STATE_SEND_BREAK_FIELD */
    if (linCurrentState->currentNodeState == LIN_NODE_STATE_SEND_BREAK_FIELD)
    {
        /* Set flag LIN bus busy */
        linCurrentState->isBusBusy = true;
        /* Change the node's current state to SENDING PID to send PID after send SYNC */
        linCurrentState->currentNodeState = LIN_NODE_STATE_SEND_PID;
        /* Send Sync Field 0x55 */
        UART_Putchar(base, 0x55);
    }
}
/* If the current node is SLAVE */
else
{
    /* Set flag LIN bus busy */
    linCurrentState->isBusBusy = true;
    /* Change the node's current state to RECEIVED BREAK FIELD */
    linCurrentState->currentEventId = LIN_RECV_BREAK_FIELD_OK;

    /* Callback function */
    if (linCurrentState->Callback != NULL)
    {
        linCurrentState->Callback(instance, linCurrentState);
    }
    /* Change the node's current state to RECEIVING SYNC FIELD */
    linCurrentState->currentNodeState = LIN_NODE_STATE_RECV_SYNC;
}
```

```
/* Check if the current node is MASTER */
if (linUserConfig->nodeFunction == (bool)MASTER)
{
    /* Check if LIN current node state is LIN_NODE_STATE_SEND_BREAK_FIELD */
    if (linCurrentState->currentNodeState == LIN_NODE_STATE_SEND_BREAK_FIELD)
    {
        /* Set flag LIN bus busy */
        linCurrentState->isBusBusy = true;
        /* Change the node's current state to SENDING PID to send PID after send SYNC */
        linCurrentState->currentNodeState = LIN_NODE_STATE_SEND_PID;
        /* Send Sync Field 0x55 */
        UART_Putchar(base, 0x55);
    }
}
/* If the current node is SLAVE */
else
{
    /* Set flag LIN bus busy */
    linCurrentState->isBusBusy = true;
    /* Change the node's current state to RECEIVED BREAK FIELD */
    linCurrentState->currentEventId = LIN_RECV_BREAK_FIELD_OK;
    /* Change the node's current state to RECEIVING SYNC FIELD */
    linCurrentState->currentNodeState = LIN_NODE_STATE_RECV_SYNC;
    /* Callback function */
    if (linCurrentState->Callback != NULL)
    {
        linCurrentState->Callback(instance, linCurrentState);
    }
}
```

· **驱动层（lin_uart_driver.c）**：在 `LIN_UART_DRV_ProcessBreakDetect()` 中将 `linCurrentState->currentNodeState = LIN_NODE_STATE_RECV_SYNC;` 移至 `Callback()` 调用之前。

- **2. 解决主要矛盾（消除 LBKDIE 逆序关断）：**



```
/*!
 */
static void LinSync_ExitAutobaudMode(void)
{
    LinSync_DisableGpioCapture();
    s_captureCount = 0U;
    s_previousCaptureValue = 0U;
    s_isAutobaudMode = false;
}

/*!
 * @brief Switch the LIN RX pin to GPIO falling-edge interrupt mode.
 *
 * Break field detection has already been completed by the UART hardware.
 * From this point onward, GPIO interrupts and pTMR timestamps are used to
 * capture Sync (0x55) falling edges and finish baud-rate estimation before
 * the Sync stop-bit rising edge arrives.
 */
static void LinSync_EnableGpioCapture(void)
{
    s_captureCount = 0U;
    s_previousCaptureValue = 0U;
    PINS_DRV_SetMuxModeSel(LIN_SYNC_RX_PCTRL, LIN_SYNC_RX_PIN, PCTRL_MUX_AS_GPIO);
    PINS_DRV_ClearPinIntFlagCmd(LIN_SYNC_RX_GPIO, LIN_SYNC_RX_PIN);
    PINS_DRV_SetPinIntSel(LIN_SYNC_RX_GPIO, LIN_SYNC_RX_PIN, PCTRL_INT_FALLING_EDGE);
    UART_SetIntMode(LIN_SYNC_UART_BASE, UART_INT_LIN_BREAK_DETECT, false);
}

/*!
 * @brief Disable GPIO edge capture and restore the RX pin mux back to UART.
 */
static void LinSync_DisableGpioCapture(void)
{
```

```
/*!
 */
static void LinSync_ExitAutobaudMode(void)
{
    LinSync_DisableGpioCapture();
    s_captureCount = 0U;
    s_previousCaptureValue = 0U;
    s_isAutobaudMode = false;
}

/*!
 * @brief Switch the LIN RX pin to GPIO falling-edge interrupt mode.
 *
 * Break field detection has already been completed by the UART hardware.
 * From this point onward, GPIO interrupts and pTMR timestamps are used to
 * capture Sync (0x55) falling edges and finish baud-rate estimation before
 * the Sync stop-bit rising edge arrives.
 */
static void LinSync_EnableGpioCapture(void)
{
    s_captureCount = 0U;
    s_previousCaptureValue = 0U;
    PINS_DRV_SetMuxModeSel(LIN_SYNC_RX_PCTRL, LIN_SYNC_RX_PIN, PCTRL_MUX_AS_GPIO);
    PINS_DRV_ClearPinIntFlagCmd(LIN_SYNC_RX_GPIO, LIN_SYNC_RX_PIN);
    PINS_DRV_SetPinIntSel(LIN_SYNC_RX_GPIO, LIN_SYNC_RX_PIN, PCTRL_INT_FALLING_EDGE);
}

/*!
 * @brief Disable GPIO edge capture and restore the RX pin mux back to UART.
 */
static void LinSync_DisableGpioCapture(void)
{
```

中断方式



```
/*!
 * generates one DMA request that copies pTMR->TCV into s_dmaTimerValue[].
 */
static void LinSync_EnableFallingEdgeDmaCapture(void)
{
    uint32_t index;

    s_isAutobaudActive = true;
    s_dmaCaptureDone = false;
    s_dmaCaptureError = false;
    s_baudPrepared = false;
    s_isStopIrqArmed = false;

    for (index = 0U; index < LIN_SYNC_FALLING_EDGE_COUNT; index++)
    {
        s_dmaTimerValue[index] = 0U;
    }

    PINS_DRV_SetMuxModeSel(LIN_SYNC_RX_PCTRL, LIN_SYNC_RX_PIN, PCTRL_MUX_AS_GPIO);
    PINS_DRV_ClearPinIntFlagCmd(LIN_SYNC_RX_GPIO, LIN_SYNC_RX_PIN);
    PINS_DRV_SetPinIntSel(LIN_SYNC_RX_GPIO, LIN_SYNC_RX_PIN, PCTRL_DMA_FALLING_EDGE);
    UART_SetIntMode(LIN_SYNC_UART_BASE, UART_INT_LIN_BREAK_DETECT, false);
}

/*!
 * @brief Disable GPIO DMA/interrupt edge selection.
 *
```

DMA方式

· **删除(main.c)**：`UART_SetIntMode(LIN_SYNC_UART_BASE, UART_INT_LIN_BREAK_DETECT, false);`

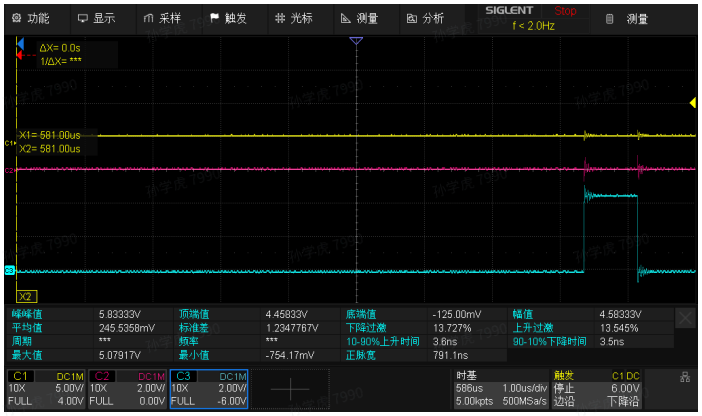
# 测试结果

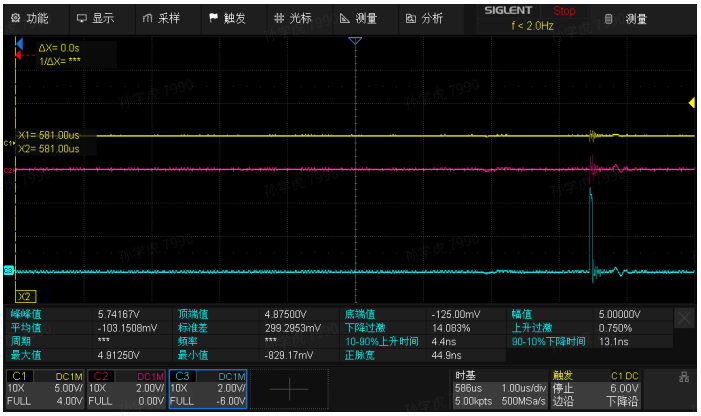
与修复前相同的毛刺矩阵（124 次注入）下，**永久失联从 16 次降为 0 次**，372 个干净帧全部应答；落在原死锁窗口内的 16 次毛刺最多造成当前帧拒帧，下一帧恢复。

| 毛刺下降沿 − T1 | 修复前 | 修复后 |
| --- | --- | --- |
| ⩽ −0.4 μs | 54/54 通过 | 56 次：53 次通过，3 次拒帧后恢复 |
| −0.4～+0.42 μs（原死锁窗口） | 16/16 永久失联 | 16 次：9 次通过，7 次拒帧后恢复，0 次失联 |
| ⩾ +0.42 μs | 54/54 拒帧后恢复 | 52/52 拒帧后恢复 |

**实测波形对比（左：修复前，右：修复后；C1=LIN 总线，C2=LE1 RX，C3=PTC14 标记）：**

**① 干净帧的 [T1, T5] 标记**

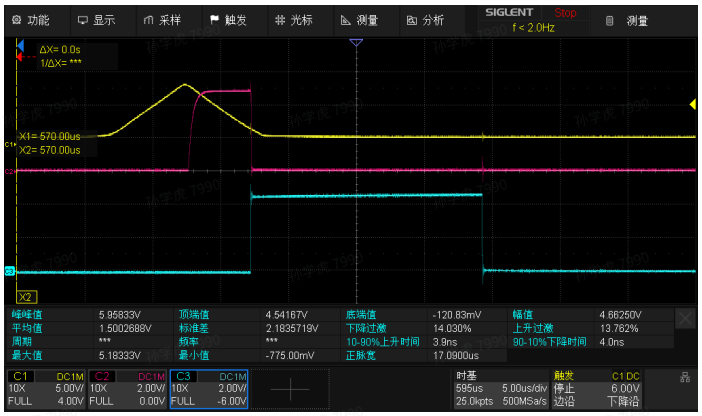




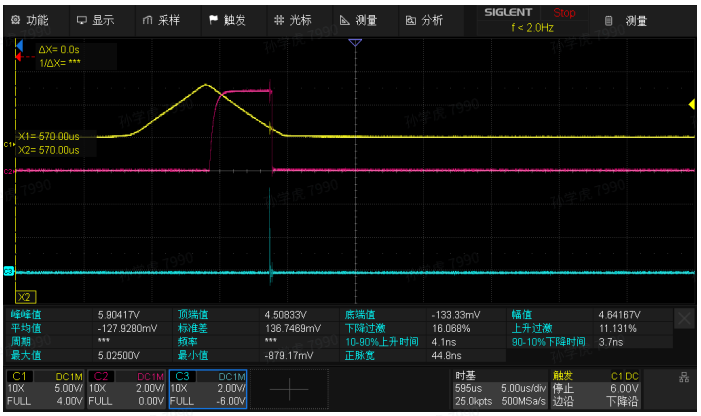
修复前：标记宽 791 ns，开 GPIO 中断后还要执行关 LBKDIE

修复后：标记宽 45 ns，T1 之后已无关 LBKDIE 指令

### ② 毛刺落在 T1 附近

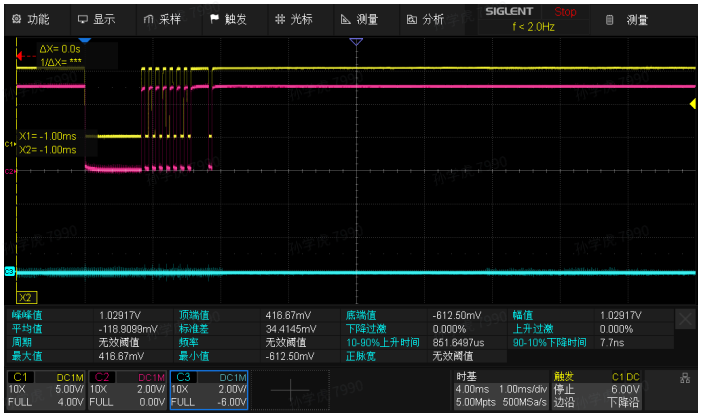


修复前：毛刺命中 [T1, T5]，INTE=0x21，RECV_SYNC 死锁

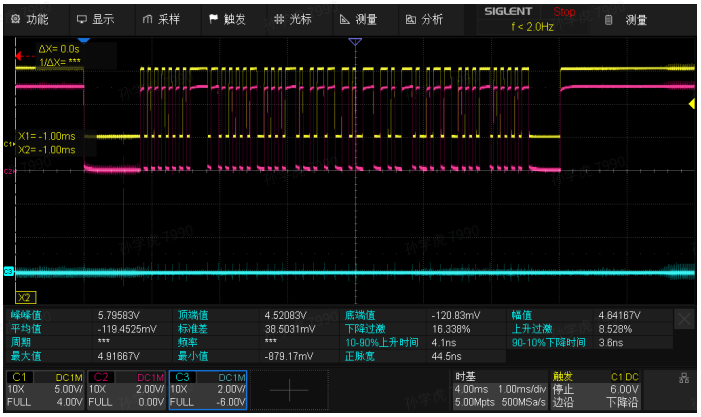


修复后：毛刺同样落在 T1（相差 0.21 µs），INTE=0x821，IDLE

### ③ 毛刺之后的干净帧



修复前：只有帧头，从机无应答



修复后：从机正常应答

- **寄存器快照：**修复后所有帧之后均为 `INTE=0x821`（LBKDIE=1）、State=IDLE；修复前死锁终态为 `INTE=0x21`（LBKDIE=0）、State=RECV_SYNC。
- **PTC14 标记：**[T1_mk, T5_mk] 宽度从 0.78～0.80 µs 缩到 0.04 µs。开 GPIO 下降沿中断之后已没有关 LBKDIE 的指令，逆序覆写的窗口不存在了。
- **测试条件：**中断版；只额外加 PTC14 标记。19,200 bps、16 bit Break、ID 0x3D；毛刺宽 7 µs，584～590 µs 每 0.2 µs 注入一次，每点 4 次，与修复前矩阵相同。