文档版本	V1.0.0
发布日期	20221026

APT32F110x 基于 CSI 库 ETCB 应用指南

目录

1 概述

本文介绍了在APT32F110x中使用ETCB的应用范例。

2. 适用的硬件

该例程使用于 APT32F110x 系列学习板

3. 应用方案代码说明

基于 APT32F110x 完整的库文件系统，可以对 ETCB 进行配置。

3.1 ETCB 概述

- **硬件配置：**

ETCB 模块是在收到一个 IP 的某个事件后，会触发另一个 IP 的相应动作。该模块能有效的减少 CPU 处理器中断请求的时间，从而节省 CPU 资源的占用，降低 CPU 的负载。

该模块总共有 12 个通道。每个通道都可以由一个源来触发另一个目标。通道 0 可以由多个源触发一个目标，通道 1 和通道 2 可以用一个源来触发多个目标。如果需要使用到 DMA 的触发功能，只能使用通道 8~11，并且在通道 8 的配置寄存器中使能 DMA 功能。

- **注意：**

a) 如果某个 IP 的事件源一直不停的以一个非常高的频率触发，那么 ETB 模块有可能会丢失一部分触发信号。

b) 事件源输出触发后，ETB 模块需要一个时钟处理事件转发，即一个时钟后事件到目标模块。

c) 一个目标只能被一个通道选择，如果 2 个或者多个通道都选择了同一个目标，那么序号小的目标有更高的优先级。

- **模块框图：**

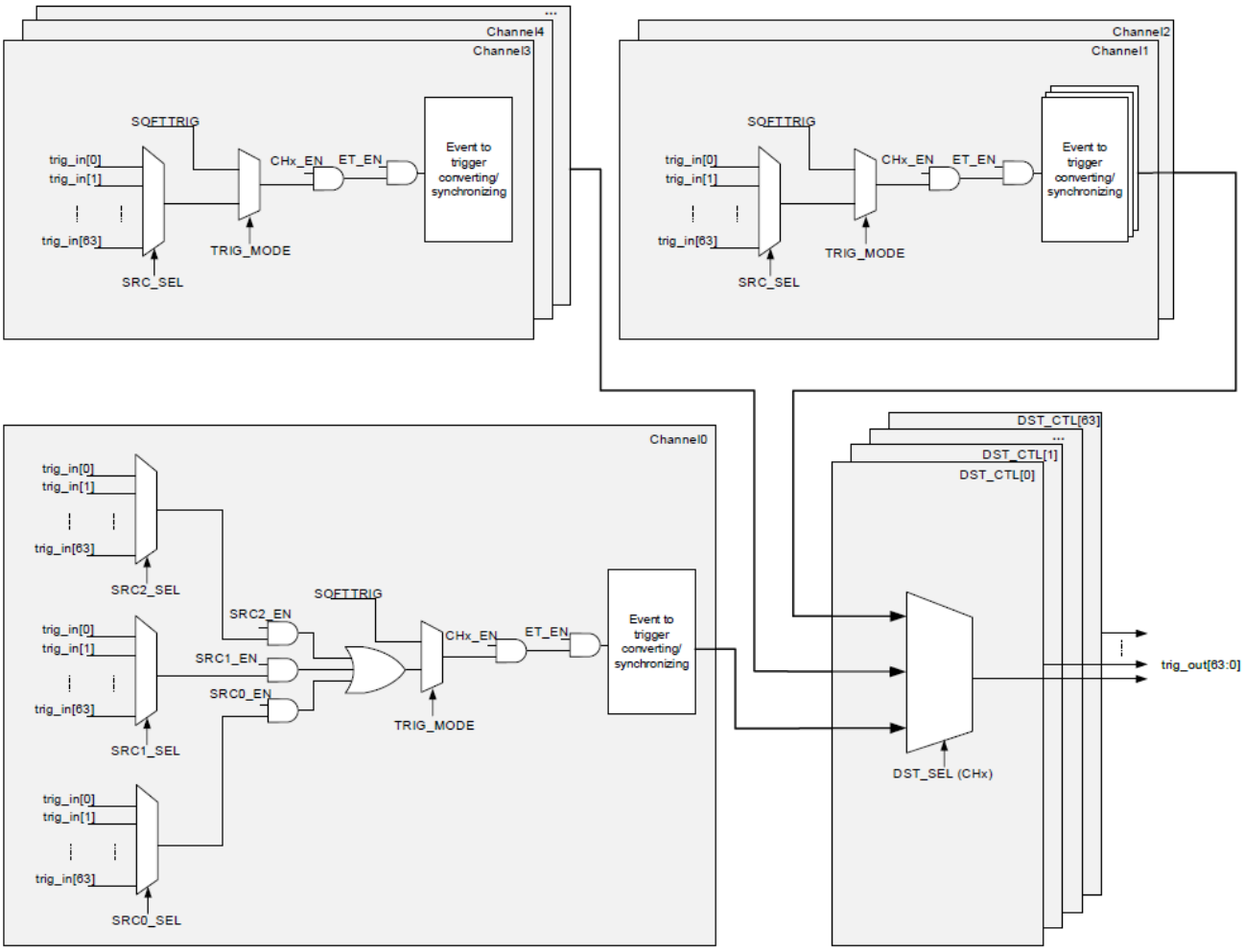

图 3.1.1 ETCB 框图

3.2 ETCB 事件对应表

事件源都是来自片上各 IP 模块。当 IP 在工作时，这些事件就会产生，而并不需要相应的中断使能。事件序号与 IP 的对应关系如下表格。(参见 APT32F110X 系列使用手册 ETCB 章节 Table8-1 事件对应表)

源序号	事件源	目标序号	目标事件
0 （0H）	LPT_TRGOUT0	0 （0H）	LPT_SYNCIN0
1 （1H）	RSVD	1 （1H）	RSVD

2 （2H）	RSVD	2 （2H）	BT0_SYNCIN0
3 （3H）	RSVD	3 （3H）	BT0_SYNCIN1
4 （4H）	EXI_ TRGOUT0	4 （4H）	BT1_SYNCIN0
5 （5H）	EXI_ TRGOUT1	5 （5H）	BT1_SYNCIN1
6 （6H）	EXI_ TRGOUT2	6 （6H）	ADC_SYNCIN0
7 （7H）	EXI_TRGOUT3	7 （7H）	ADC_SYNCIN1
8 （8H）	RTC_SYNC0	8 （8H）	ADC_SYNCIN2
9 （9H）	RTC_SYNC1	9 （9H）	ADC_SYNCIN3
10 （AH）	BT0_TRGOUT	10 （AH）	ADC_SYNCIN4
11 （BH）	BT1_TRGOUT	11 （BH）	ADC_SYNCIN5
12 （CH）	RSVD	12 （CH）	BT1_SYNCIN1
13 （DH）	RSVD	13 （DH）	BT1_SYNCIN2
14 （EH）	EPT0_TRGOUT0	14 （EH）	CMP0_SYNCIN
15 （FH）	EPT0_TRGOUT1	15 （FH）	CMP1_SYNCIN
16 （10H）	EPT0_TRGOUT2	16 （10H）	EPT0_SYNCIN0
17 （11H）	EPT0_TRGOUT3	17 （11H）	EPT0_SYNCIN1
18 （12H）	RSVD	18 （12H）	EPT0_SYNCIN2
19 （13H）	EXI_TRGOUT4	19 （13H）	EPT0_SYNCIN3
20 （14H）	EXI_TRGOUT5	20 （14H）	EPT0_SYNCIN4

21 （15H）	ADC_TRGOUT0	21 （15H）	EPT0_SYNCIN5
22 （16H）	ADC_TRGOUT1	22 （16H）	RSVD
23 （17H）	RSVD	23 （17H）	RSVD
24 （18H）	CMP0_TRGOUT	24 （18H）	GPTA0_SYNCIN0
25 （19H）	CMP1_ TRGOUT	25 （19H）	GPTA0_SYNCIN1
26 （1AH）	RSVD	26 （1AH）	GPTA0_SYNCIN2
27 （1BH）	RSVD	27 （1BH）	GPTA0_SYNCIN3
28 （1CH）	RSVD	28 （1CH）	GPTA0_SYNCIN4
29 （1DH）	RSVD	29 （1DH）	GPTA0_SYNCIN5
30 （1EH）	GPTA0_ TRGOUT 0	30 （1EH）	RSVD
31 （1FH）	GPTA0_ TRGOUT 1	31 （1FH）	RSVD
32 （20H）	GPTA1_ TRGOUT 0	32 （20H）	GPTA1_SYNCIN0
33 （21H）	GPTA1_ TRGOUT 1	33 （21H）	GPTA1_SYNCIN1
34 （22H）	RSVD	34 （22H）	GPTA1_SYNCIN2
35 （23H）	RSVD	35 （23H）	GPTA1_SYNCIN3
36 （24H）	RSVD	36 （24H）	GPTA1_SYNCIN4
37 （25H）	RSVD	37 （25H）	GPTA1_SYNCIN5
38 （26H）	GPTB0_ TRGOUT 0	38 （26H）	RSVD
39 （27H）	GPTB0_ TRGOUT 1	39 （27H）	RSVD

40 （28H）	GPTB1_ TRGOUT 0	40 （28H）	GPTB0_SYNCIN0
41 （29H）	GPTB1_ TRGOUT 1	41 （29H）	GPTB0_SYNCIN1
42 （2AH）	RSVD	42 （2AH）	GPTB0_SYNCIN2
43 （2BH）	RSVD	43 （2BH）	GPTB0_SYNCIN3
44 （2CH）	TKEY_ TRGOUT	44 （2CH）	GPTB0_SYNCIN4
45 （2DH）	RSVD	45 （2DH）	GPTB0_SYNCIN5
46 （2EH）	RSVD	46 （2EH）	RSVD
47 （2FH）	RSVD	47 （2FH）	RSVD
48 （30H）	I2C0_TXSRC	48 （30H）	GPTB1_SYNCIN0
49 （31H）	I2C0_RXSRC	49 （31H）	GPTB1_SYNCIN1
50 （32H）	RSVD	50 （32H）	GPTB1_SYNCIN2
51 （33H）	RSVD	51 （33H）	GPTB1_SYNCIN3
52 （34H）	RSVD	52 （34H）	GPTB1_SYNCIN4
53 （35H）	RSVD	53 （35H）	GPTB1_SYNCIN5
54 （36H）	RSVD	54 （36H）	RSVD
55 （37H）	RSVD	55 （37H）	RSVD
56 （38H）	UART0_TXSRC	56 （38H）	RSVD
57 （39H）	UART0_RXSRC	57 （39H）	TKEY_SYNCIN
58 （3AH）	UART1_TXSRC	58 （3AH）	RSVD

59 （3BH）	UART1_RXSRC	59 （3BH）	RSVD
60 （3CH）	UART2_TXSRC	60 （3CH）	DMA_CH0
61 （3DH）	UART2_RXSRC	61 （3DH）	DMA_CH1
62 （3EH）	USART0_TXSRC	62 （3EH）	DMA_CH2
63 （3FH）	USART0_RXSRC	63 （3FH）	DMA_CH3

3.3 一对一触发模式

选择内部主频 48MHz 作为系统时钟，可在 user_demo.c 文件的 etcb_one_trg_one_demo0() 中进行配置。通过 GPIO 的外部中断事件触发 BT 模块，在中断里翻转 IO 输出。

- **编程要点：**
  1. 配置对应模块的功能以及触发事件
  2. 配置 ETCB 对应事件源和目标事件

```
int etcb_one_trg_one_demo0(void)
{
    int iRet = 0;
    volatile uint8_t ch;
    csi_etb_config_t tEtbConfig;                                    //ETB 参数配置结构体

    csi_pin_set_mux(PA01,PA01_INPUT);
    csi_pin_pull_mode(PA01, GPIO_PULLUP);                           //PA01 上拉
    csi_pin_irq_mode(PA01,EXI_GRP1, GPIO_IRQ_FALLING_EDGE);         //PA01 下降沿产生中断
    csi_exi_set_evtrg(EXI_TRGOUT1, TRGSRC_EXI1, 0);
    csi_pin_irq_enable(PA01, EXI_GRP1, ENABLE);

    csi_bt_start_sync(BT0, 1000);
    csi_pin_set_mux(PA19, PA19_OUTPUT);
    csi_bt_set_sync(BT0,BT_TRG_SYNCIN0, BT_TRG_ONCE, ENABLE);
    csi_bt_set_evtrg(BT0, 0, BT_TRGSRC_PEND);

    tEtbConfig.byChType = ETB_ONE_TRG_ONE;          //单个源触发单个目标
    tEtbConfig.bySrcIp   = ETB_EXI_TRGOUT1;         //EXI1 触发输出 0 作为触发源
```

```
        tEtbConfig.byDstIp =   ETB_BT0_SYNCIN0;          //BT0 同步输入作为目标事件
        tEtbConfig.byTrgMode = ETB_HARDWARE_TRG;

        csi_etb_init();
        ch = csi_etb_ch_alloc(tEtbConfig.byChType);      //自动获取空闲通道号,ch >= 0 获取成功
        if(ch < 0)
            return -1;                                   //ch < 0,则获取通道号失败
        iRet = csi_etb_ch_config(ch, &tEtbConfig);

        return iRet;
    }

/************中断处理函数**********************/
__attribute__((weak)) void bt_irqhandler(csp_bt_t *ptBtBase)
{
    // ISR content ...
    volatile uint32_t wMisr = csp_bt_get_isr(ptBtBase);

    if(wMisr & BT_PEND_INT)                          //PEND interrupt
    {
        csp_bt_clr_isr(ptBtBase, BT_PEND_INT);

        csi_pin_toggle(PA19);
    }

    if(wMisr & BT_CMP_INT)                           //CMP interrupt
    {
        csp_bt_clr_isr(ptBtBase, BT_CMP_INT);
    }

    if(wMisr & BT_OVF_INT)                           //OVF interrupt
    {
        csp_bt_clr_isr(ptBtBase, BT_OVF_INT);
    }

    if(wMisr & BT_EVTRG_INT)                         //EVTRG interrupt
    {
        csp_bt_clr_isr(ptBtBase, BT_EVTRG_INT);
    }
}
```

- **代码说明：**

1. **csi_pin_set_mux(): -----** 设置 PIN 脚功能。

2. **csi_pin_pull_mode():** ----- 设置 PIN 脚上拉/下拉功能。
3. **csi_pin_irq_mode():** ----- 配置 PIN 脚中断模式。
4. **csi_exi_set_evtrg():** ----- 配置外部中断事件触发输出。
5. **csi_pin_irq_enable():** ----- 使能 PIN 中断。
6. **csi_bt_start_sync():** ----- 启动 BT 同步触发功能。
7. **csi_bt_set_sync():** ----- 配置 BT 外部同步触发输入。
8. **csi_etb_ch_alloc():** ----- 申请一个 ETB 通道。
9. **csi_etb_ch_config():** ----- 配置 ETB 通道。
10. **csi_etb_init():** ----- 初始化 ETB 模块

- **函数参数说明：**

1. **csi_pin_set_mux(pin_name_e ePinName, pin_func_e ePinFunc)；**

   ePinName：pin 脚名字

   ePinFunc：pin 脚功能

2. **csi_pin_pull_mode(pin_name_e ePinName, csi_gpio_pull_mode_e ePullMode)；**

   ePinName：pin 脚名字

   ePullMode：上拉/下拉模式

3. **csi_pin_irq_mode(pin_name_e ePinName, csi_exi_grp_e eExiGrp, csi_gpio_irq_mode_e eTrgEdge)；**

   ePinName：pin 脚名字

   eExiGrp：外部中断组

   eTrgEdge：中断触发边沿模式

4. **csi_exi_set_evtrg(csi_exi_trgout_e eTrgOut, csi_exi_trgsrc_e eExiTrgSrc, uint8_t**

byTrgPrd)；

eTrgOut：输出通道选择

eExiTrgSrc：EXI 触发源

byTrgPrd：EXI 事件触发计数周期

5. **csi_pin_irq_enable(pin_name_e ePinName, csi_exi_grp_e eExiGrp, bool bEnable)；**

ePinName：pin 脚名字

eExiGrp：外部中断组

bEnable：使能/禁止中断

6. **csi_bt_start_sync(csp_bt_t *ptBtBase, uint32_t wTimeOut)；**

ptBtBase：BT 寄存器结构体指针，指向 BT 基地址。

wTimeOut：计数器溢出时间，即定时时间，单位 us。

7. **csi_bt_set_sync(csp_bt_t *ptBtBase,csi_bt_trgin_e eTrgin, csi_bt_trgmode_e eTrgMode, bool bAutoRearm)；**

ptBtBase：BT 寄存器结构体指针，指向 BT 基地址。

eTrgin：同步触发输入通道

eTrgMode：两种触发模式，连续触发、一次性触发

bAutoRearm：一次性触发模式下有效。ENABLE：清除通道状态，并允许新的触发；

DISABLE：无效

8. **csi_etb_ch_alloc(csi_etb_ch_type_e eChType)；**

eChType：ETB 通道类型

9. **csi_etb_ch_config(csi_etb_chid_e eChId, csi_etb_config_t *ptConfig)；**

eChId：ETB 通道 ID 号

ptConfig：ETB 通道配置参数结构体指针

ptConfig->bySrcIp:触发源 0 事件

ptConfig->bySrcIp1:触发源 1 事件

ptConfig->bySrcIp2:触发源 2 事件

ptConfig->byDstIp:触发目标 0 事件

ptConfig->byDstIp1:触发目标 1 事件

ptConfig->byDstIp2:触发目标 2 事件

ptConfig->byTrgMode:触发模式

ptConfig->byChType:通道类型

- **波形输出：**

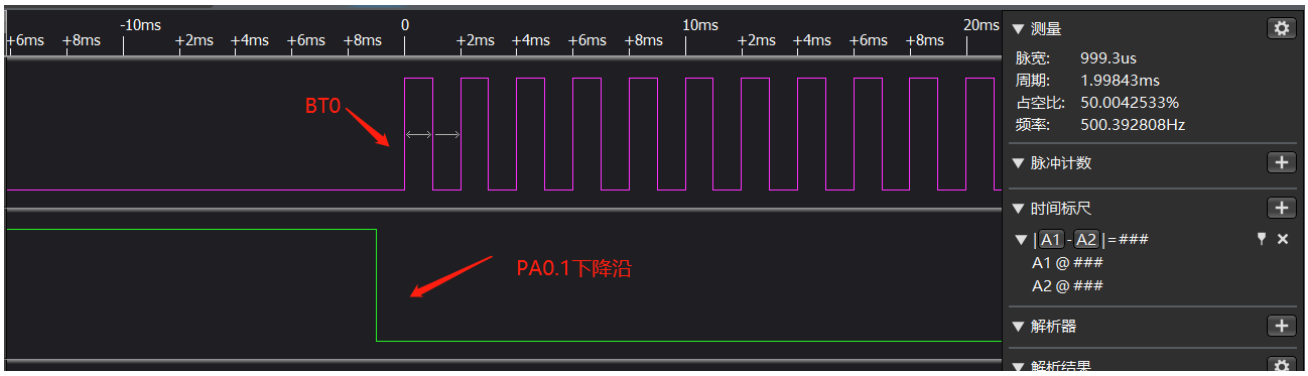

图 3.3.1 BT 输出波形

3.4 一对多触发模式

选择内部主频 48MHz 作为系统时钟，通过 GPIO 的外部中断事件触发 BT 模块和 LPT 模块产生，在中断里翻转 IO 输出。(可在 user_demo.c 文件的 etcb_one_trg_more_demo ()中进行配置)

```
int etcb_one_trg_more_demo(void)
{
    int iRet = 0;
    volatile uint8_t ch;
    csi_etb_config_t tEtbConfig;                    //ETB 参数配置结构体
```

```
    csi_pin_set_mux(PA01,PA01_INPUT);
    csi_pin_pull_mode(PA01, GPIO_PULLUP);                                   //PA01 上拉
    csi_pin_irq_mode(PA01,EXI_GRP1, GPIO_IRQ_FALLING_EDGE);                 //PA01 下降沿产生中断
    csi_exi_set_evtrg(EXI_TRGOUT1, TRGSRC_EXI1, 0);
    csi_pin_irq_enable(PA01, EXI_GRP1, ENABLE);

    csi_bt_start_sync(BT0, 1000);
    csi_pin_set_mux(PA19, PA19_OUTPUT);
    csi_bt_set_sync(BT0,BT_TRG_SYNCIN0, BT_TRG_ONCE, ENABLE);

    csi_pin_set_mux(PA04, PA04_OUTPUT);
    csi_lpt_start_sync(LPT,LPT_CLK_ISCLK,5);
    csi_lpt_set_sync(LPT, LPT_TRG_SYNCIN0, LPT_SYNC_ONCE, ENABLE);

    tEtbConfig.byChType = ETB_ONE_TRG_MORE;
    tEtbConfig.bySrcIp   = ETB_EXI_TRGOUT1;
    tEtbConfig.byDstIp =   ETB_BT0_SYNCIN0;
    tEtbConfig.byDstIp1 = ETB_LPT_SYNCIN0;
    tEtbConfig.byDstIp2 = ETB_DST_NOT_USE;
    tEtbConfig.byTrgMode = ETB_HARDWARE_TRG;

    csi_etb_init();
    ch = csi_etb_ch_alloc(tEtbConfig.byChType);          //自动获取空闲通道号,ch >= 0  获取成功
    if(ch < 0)
        return -1;                                                          //ch < 0,则获取通道号失败
    iRet = csi_etb_ch_config(ch, &tEtbConfig);

    return iRet;
}
/*****************BT 中断处理函数*****************************/
__attribute__((weak)) void bt_irqhandler(csp_bt_t *ptBtBase)
{
    // ISR content ...
    volatile uint32_t wMisr = csp_bt_get_isr(ptBtBase);

    if(wMisr & BT_PEND_INT)                                 //PEND interrupt
    {
        csp_bt_clr_isr(ptBtBase, BT_PEND_INT);

        csi_pin_toggle(PA19);
    }

    if(wMisr & BT_CMP_INT)                                  //CMP interrupt
```

```
        {
                csp_bt_clr_isr(ptBtBase, BT_CMP_INT);
        }

        if(wMisr & BT_OVF_INT)                                  //OVF interrupt
        {
                csp_bt_clr_isr(ptBtBase, BT_OVF_INT);
        }

        if(wMisr & BT_EVTRG_INT)                        //EVTRG interrupt
        {
                csp_bt_clr_isr(ptBtBase, BT_EVTRG_INT);
        }
}

/******************LPT 中断处理函数*************************/
__attribute__((weak)) void lpt_irqhandler(csp_lpt_t *ptLptBase)
{
    volatile uint32_t wMisr = csp_lpt_get_misr(ptLptBase);

        if(wMisr & LPT_TRGEV_INT)
        {
                csp_lpt_clr_misr(ptLptBase, LPT_TRGEV_INT);
        }
        if(wMisr & LPT_MATCH_INT)
        {
                csp_lpt_clr_misr(ptLptBase, LPT_MATCH_INT);
        }
        if(wMisr & LPT_PEND_INT)
        {
                csp_lpt_clr_misr(ptLptBase, LPT_PEND_INT);
                csi_pin_toggle(PA04);
        }
}
```

- **代码说明：**

1. **csi_pin_set_mux():** ----- 设置 PIN 脚功能。
2. **csi_pin_pull_mode():** ----- 设置 PIN 脚上拉/下拉功能。
3. **csi_pin_irq_mode():** ----- 配置 PIN 脚中断模式。
4. **csi_exi_set_evtrg():** ----- 配置外部中断事件触发输出。

5. **csi_pin_irq_enable():** ----- 使能 PIN 中断。
6. **csi_bt_start_sync():** ----- 启动 BT 同步触发功能。
7. **csi_bt_set_sync():** ----- 配置 BT 外部同步触发输入。
8. **csi_etb_ch_alloc():** ----- 申请一个 ETB 通道。
9. **csi_etb_ch_config():** ----- 配置 ETB 通道。
10. **csi_etb_init():** ----- 初始化 ETB 模块
11. **csi_lpt_start_sync():** ----- 启动 LPT 同步触发功能
12. **csi_lpt_set_sync():** ----- 设置 LPT 同步触发功能

- **函数参数说明：**

1. **csi_pin_set_mux(pin_name_e ePinName, pin_func_e ePinFunc)；**

   ePinName：pin 脚名字

   ePinFunc：pin 脚功能

2. **csi_pin_pull_mode(pin_name_e ePinName, csi_gpio_pull_mode_e ePullMode)；**

   ePinName：pin 脚名字

   ePullMode：上拉/下拉模式

3. **csi_pin_irq_mode(pin_name_e ePinName, csi_exi_grp_e eExiGrp, csi_gpio_irq_mode_e eTrgEdge)；**

   ePinName：pin 脚名字

   eExiGrp：外部中断组

   eTrgEdge：中断触发边沿模式

4. **csi_exi_set_evtrg(csi_exi_trgout_e eTrgOut, csi_exi_trgsrc_e eExiTrgSrc, uint8_t byTrgPrd)；**

   eTrgOut：输出通道选择

eExiTrgSrc：EXI 触发源

byTrgPrd：EXI 事件触发计数周期

5. csi_pin_irq_enable(pin_name_e ePinName, csi_exi_grp_e eExiGrp, bool bEnable)；

ePinName：pin 脚名字

eExiGrp：外部中断组

bEnable：使能/禁止中断

6. csi_bt_start_sync(csp_bt_t *ptBtBase, uint32_t wTimeOut)；

ptBtBase：BT 寄存器结构体指针，指向 BT 基地址。

wTimeOut：计数器溢出时间，即定时时间，单位 us。

7. csi_bt_set_sync(csp_bt_t *ptBtBase,csi_bt_trgin_e eTrgin, csi_bt_trgmode_e eTrgMode, bool bAutoRearm)；

ptBtBase：BT 寄存器结构体指针，指向 BT 基地址。

eTrgin：同步触发输入通道

eTrgMode：两种触发模式，连续触发、一次性触发

bAutoRearm：一次性触发模式下有效。ENABLE：清除通道状态，并允许新的触发；

DISABLE：无效

8. csi_etb_ch_alloc(csi_etb_ch_type_e eChType)；

eChType：ETB 通道类型

9. csi_etb_ch_config(csi_etb_chid_e eChId, csi_etb_config_t *ptConfig)；

eChId：ETB 通道 ID 号

ptConfig：ETB 通道配置参数结构体指针

ptConfig->bySrcIp:触发源 0 事件

ptConfig->bySrcIp1:触发源 1 事件

ptConfig->bySrcIp2:触发源 2 事件

ptConfig->byDstIp:触发目标 0 事件

ptConfig->byDstIp1:触发目标 1 事件

ptConfig->byDstIp2:触发目标 2 事件

ptConfig->byTrgMode:触发模式

ptConfig->byChType:通道类型

10. **csi_lpt_start_sync(csp_lpt_t *ptLptBase, csi_lpt_clksrc_e eClk, uint32_t wTimeMs);**

    ptLptBase：LPT寄存器结构体指针

    eClk：时钟选择

    wTimeMs： 定时时间,单位 ms

11. **csi_lpt_set_sync(csp_lpt_t *ptLptBase, csi_lpt_trgin_e eTrgin, csi_lpt_trgmode_e eSyncMode, bool bARearmEnable);**

    ptLptBase：LPT寄存器结构体指针

    eTrgin：同步触发输入，LPT_TRG_SYNCIN0有效

    eSyncMode：模式选择，连续模式和一次模式

    bARearmEnable： 0：禁止硬件自动 REARM，1：周期结束时，自动 REARM

- **输出波形：**

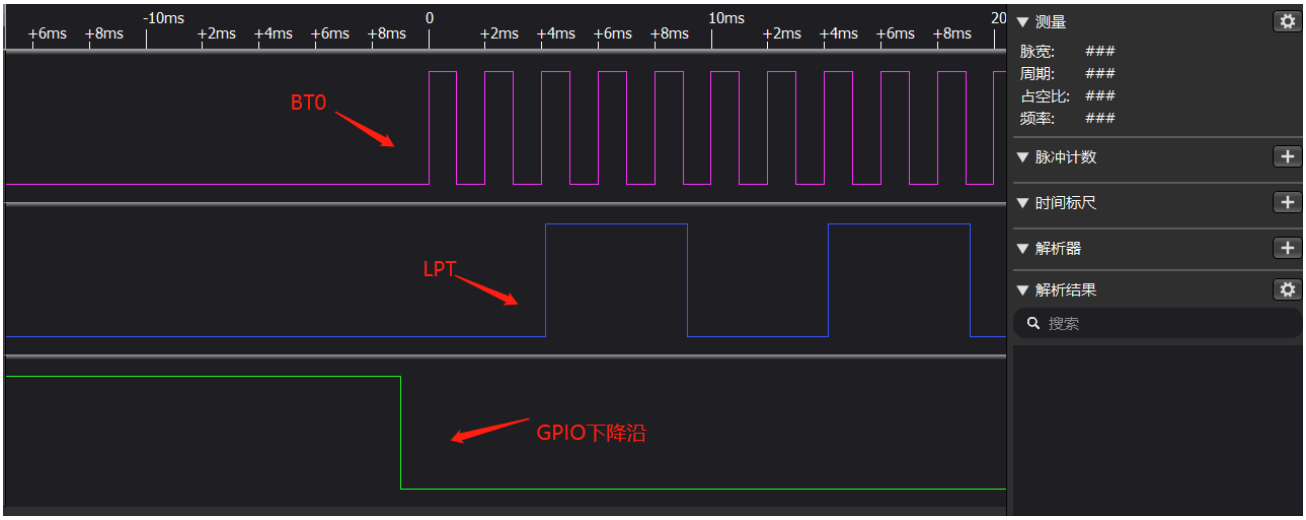

图 3.4.1 BT&LPT 输出波形

4. 程序下载和运行

1．将目标板与仿真器连接，分别为 VDD、SCLK、SWIO、GND

2. 程序编译后仿真运行

3. 通过示波器或逻辑分析仪可验证如图 3.3.1 图 3.4.1 输出波形