

Body Area Network

Demo Test Report

haobo.gao

April 23, 2019

Foxconn ZZDC

Introduce

Our Design

我们 BAN 的原本设计是这样的：

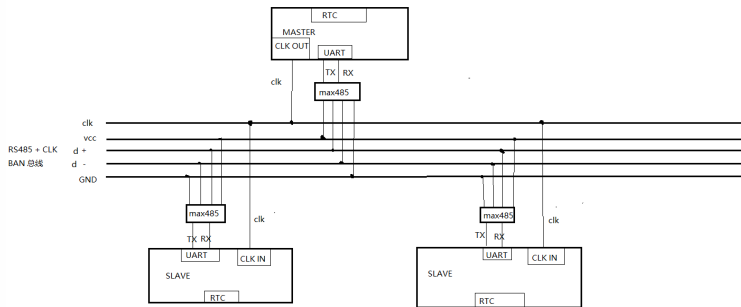


Figure: BAN

1. 由 RS485 增添一根 clk 线形成 BAN 总线的基本硬件线路。
2. clk 用于产生 ms 级别的脉冲波,用于时钟的同步。
3. d+ d- 是差分处理后的数据信号。

图2 假设了挂载在总线上的设备有一个主机,两个从机。

max485 是一型 RS232 转 RS485 的芯片,可以把串口数据转化为差分信号,使其符合 **RS-485** 电气规范。

STM32 方面需要实现串口的驱动,使确保串口可以实时的发送数据。接收数据。

本地时钟用于本地计数,同步时钟用于同步 BAN 上设备的时钟。

- RTC, 本地时钟。每个设备都有个本地时钟。计时粒度为 100us.
- CLK, 同步时钟。BAN 总线上的设备使用时钟线来完成时钟同步。主设备的定时器产生同步脉冲,从设备的定时器捕获同步脉冲完成同步。

在 demo 中, 为了克服硬件条件的限制, 我们:

- 没有使用 max485 来完成 TTL 信号转差分信号.
- 使用另外的一个定时器来模拟 RTC 的功能。
- 只能模拟一个主设备, 一个从设备的情况。
- 使用串口发送开始信号

Demo 连线

下图是 demo 的连线和结构图：

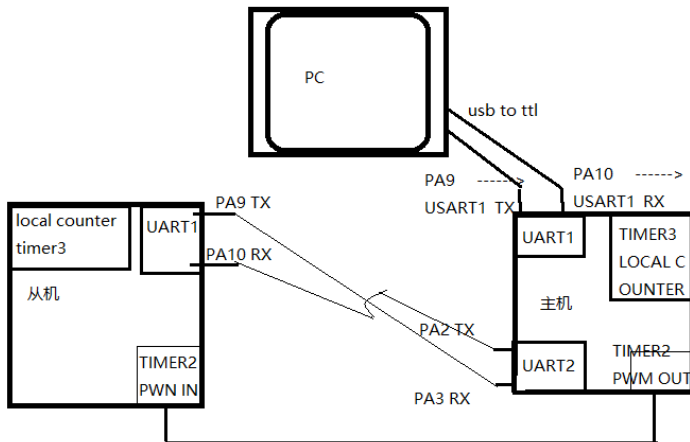


Figure: demo

workflow

Demo 工作流程: 两个指令

1. 主机, 从机上电。外设驱动初始化。从机进入 PWM 捕获状态, 等待事件的发生。
2. 主机在接收到 PC 的 "begin" 指令的时候。
 - a. 先转发给从机
 - b. 开始 timer3 本地计数 (local counter).
 - c. 开始使用 timer2 产生 PWM 波。
3. 当从机接收到 "begin" 指令, 且从机的 timer2 捕获到 clk 上的第一个上升沿。
 - a. timer3 本地计数 (local counter).
 - b. 进入 PWM 捕获同步状态。
4. "report" 指令:
 - a. 主机接收到 "report" 指令后, 上报自己的时间戳给 PC。
 - b. 从机接收到 "report" 指令后, 上报自己的时间戳给主机, 主机转发给 PC。

主从机中：

- timer2 的时钟产生或捕获 PWM，用于同步。是 BAN 系统中的大粒度(8ms)时钟。
- timer3 的时钟仅用于小粒度 (100us) 的时间戳测量。

对于从机来说，每捕获到一个上升沿，为了时间同步，需要强制使从机的 `jiffies`, `jiffies_pwm` 增加一个捕获周期(8ms)。

Demo 工作流程: 同步细节

local_tm_t 为主从机代码中描述时间的类型:

```
1 typedef struct local_time_struct{
2     unsigned char sec:7;    //秒
3     unsigned char min:7;    //分
4     unsigned char hour:5;   //时
5     unsigned int  day;      //天
6     unsigned int  jiffies;  //本地时间戳
7     unsigned int  jiffies_pwm; //pwm 时间戳
8 } local_tm_t;
```

- 主机使用 timer2 产生的 PWM 波来作为同步信号,从机捕获 pwm 上升沿。这个方波上升沿间隔固定且精确为 4ms 或者 8ms, 可以客制化,在方波的上升沿,会去触发 jiffies_pwm 累加。
- 主从都使用 timer3 每隔 100us 产生一个中断,在中断中 jiffies 累加,到 10000(1s)超时进位。作为本地时钟。

从机上升沿捕获中断

tick_sync 每隔 8ms 会被调用一次,用于同步本地时间戳:

```
1  /* every 8ms 80*100us */
2  void tick_sync(void)
3  {
4      if( dev.local_tm.jiffies_pwm + 80 < 10000 ){
5          dev.local_tm.jiffies_pwm = dev.local_tm.jiffies_pwm +
              80;
6      }
7      if( dev.local_tm.jiffies_pwm + 80 >= 10000 ){
8          dev.evt.sec_need_update=1;
9          dev.local_tm.jiffies_pwm = 80 - (10000 - dev.local_tm.
              jiffies_pwm);
10     }
11     //同步
12     dev.local_tm.jiffies = dev.local_tm.jiffies_pwm; ^ ^ I ^ ^ I //
        强等 同步
13 }
```

本地计数超时处理

tick_sync 每隔 100us 会被调用一次,用于小粒度计数:

```
1  /*
2   每 100us 进入这个中断一次。
3  */
4  void Tick(void)
5  {
6      dev.local_tm.jiffies ++;
7      if( dev.local_tm.jiffies % 80 == 0){
8          //GPIO A5 产生 方波 用于和 同步线的方波做对比
9          HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,1-HAL_GPIO_ReadPin(
10             GPIOA,GPIO_PIN_5));
11      }
12      /* one second passed */
13      if(dev.local_tm.jiffies >= 10000 ){
14          dev.local_tm.jiffies = 0;
15      }
16  }
```

这里从机的 GPIOA5 有相对于时钟同步线的方波输出。是一个测试点。

TEST Data

串口打印的时间戳

在"report" 指令发送到主机后,有如下的代码段会被执行:

```
1  /*
2   report the jiffes to the PC
3  */
4  void report_jiffes(void)
5  {
6      printf("master_rj:\t%d\r\n", dev.local_tm.jiffies);
7      printf("master_rj:\t%d\r\n", dev.local_tm.jiffies);
8      printf("master_rj_PWM:\t%d\r\n", dev.local_tm.
9              jiffies_pwm);
10     printf("master_rj_PWM:\t%d\r\n", dev.local_tm.
11            jiffies_pwm);
12 }
```

这段代码会去汇报 jiffies(**本地时钟**) 和 jiffies_pwm **同步时钟**的值给 PC.

串口打印的时间戳

```
MASTER 0 0:0:50 jiffes:0
MASTER 0 0:0:51 jiffes:0
MASTER 0 0:0:52 jiffes:0
MASTER 0 0:0:53 jiffes:0
MASTER 0 0:0:54 jiffes:0
MASTER 0 0:0:55 jiffes:0
MASTER 0 0:0:56 jiffes:0
master rj: 6119
master rj: 6126
master rj_PWM: 6080
master rj_PWM: 6080
slave:rj:80

MASTER 0 0:0:57 jiffes:0
MASTER 0 0:0:58 jiffes:0
```

Figure: report jiffies

在输入"report" 命令后,log 如上图(红圈里),结合前面的代码,可以知道:

- 上述代码中为了打印 log , printf 的执行用时大约为 $(6126-6119=7)*100\mu s$ 。这里的 printf 是移植的无阻塞,轻量级,全功能的 printf .
- 从机和主机有传输延时。

波形: 未同步

我们使

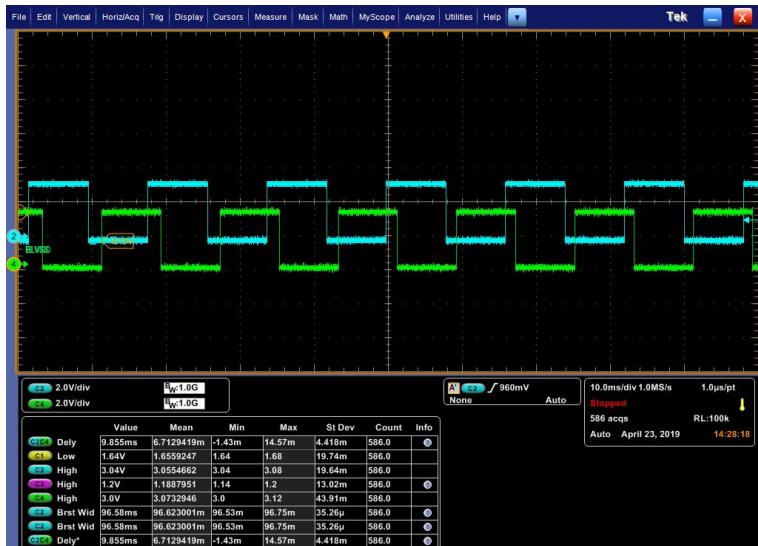
- 本地时钟每 8ms 使 GPIOA5 电平状态反转. 这个测试通道记为 CH_LOCAL.
- 同步线由主机产生半周期为 8ms 的同步方波。这个测试通道记为 CH_SYNC .
- 在从机的捕获同步线上升沿的中断处理函数中,使时钟不同步,也就是注释掉如下代码:

```
1 dev.local_tm.jiffies = dev.local_tm.jiffies_pwm;^^I^^I  
   //强等使 同步
```

下列一些截图展示了未同步的情况下的截图。

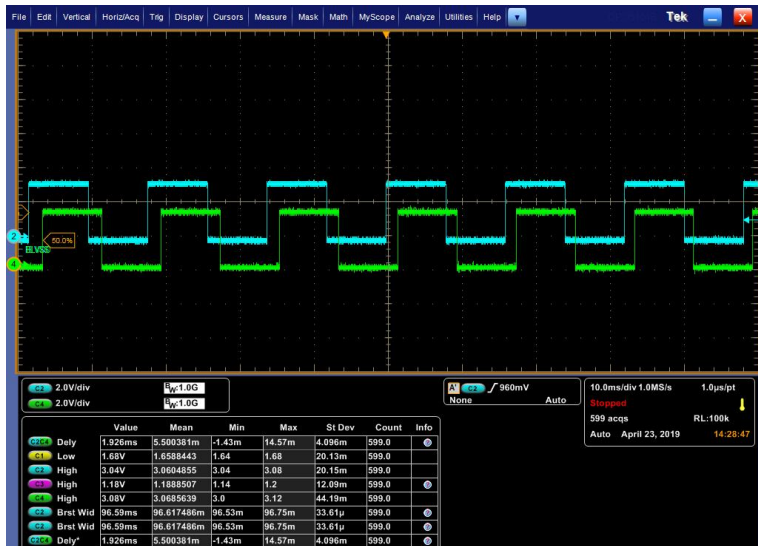
波形: 未同步

未同步的波形一:



波形: 未同步

未同步的波形二:



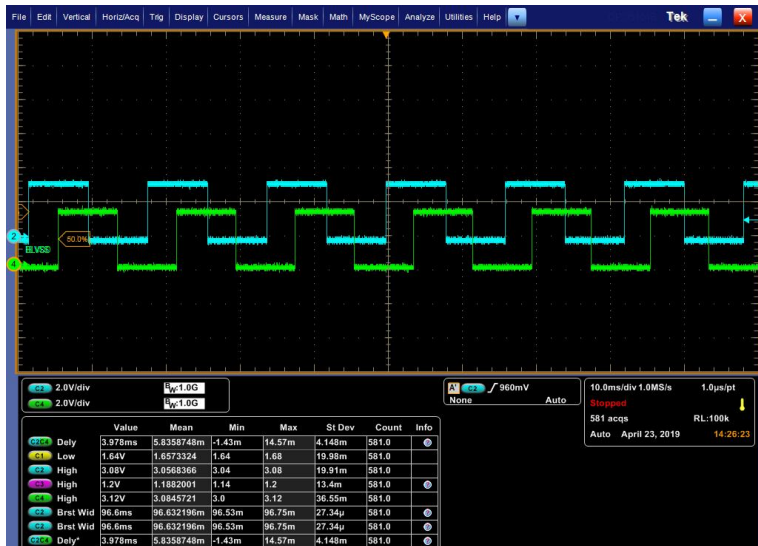
波形: 未同步

未同步的波形三:



波形: 未同步

未同步的波形四:



未同步波形的一些说明

把这个文档返回到未同步波形一 (15), 然后用键盘方向键的右键去翻页 (从未同步波形一至未同步波形四)。这个时候屏幕上呈现的动画效果比较能复原当时示波器上的样子。

由于未同步本地时钟, 所以每次误差累加, 导致了这样的动态效果。

我们使

- 本地时钟每 8ms 使 GPIOA5 电平状态反转. 这个测试通道记为 CH_LOCAL.
- 同步线由主机产生半周期为 8ms 的同步方波。这个测试通道记为 CH_SYNC .
- 在从机的捕获同步线上升沿的中断处理函数中,使时钟同步,也就是取消注释如下代码:

```
1 dev.local_tm.jiffies = dev.local_tm.jiffies_pwm;^^I^^I  
   //强等使 同步
```

下列截图展示了同步的情况下的截图。

波形: 同步

同步的波形:



同步波形的一些说明

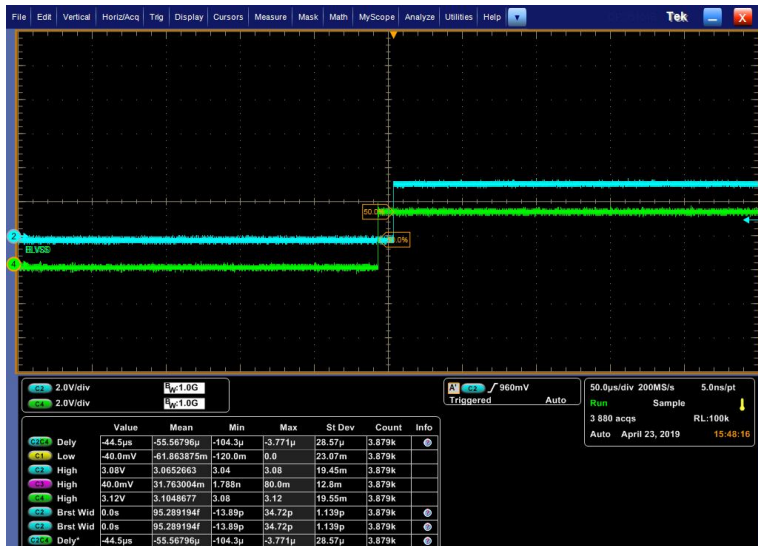
由于每次同步线上的上升沿被从机捕获后,会触发同步中断,在中断处理中,去消除每次周期的本地时钟误差。

所以同步的波形整体看起来固定,相位比较一致。但是:

如果我们放大示波器的粒度,会发现存在误差。下列是一些同步的误差。示波器的粒度已经为 $50\mu\text{s}$ 我们关注同步的前后时刻。

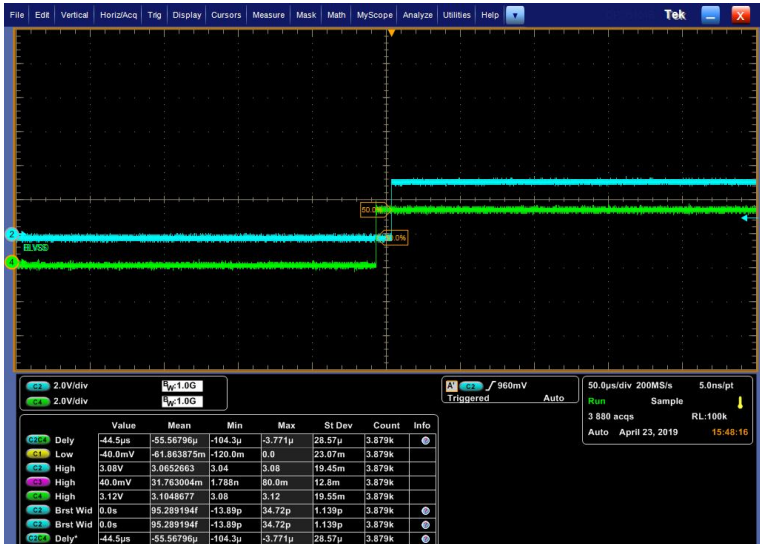
波形: 同步

同步误差:



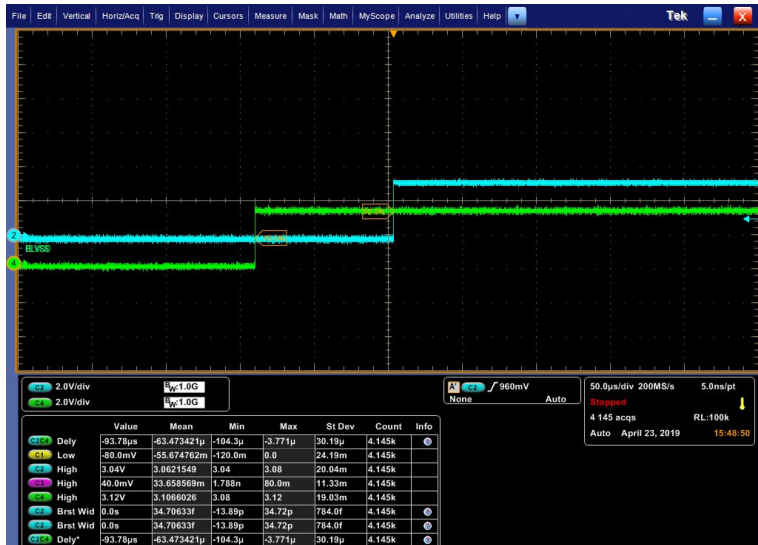
同步误差

同步误差:



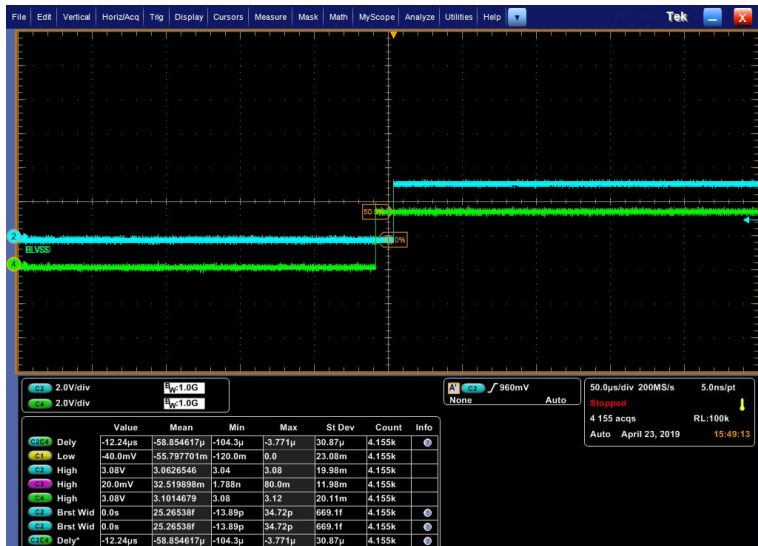
同步误差

同步误差:



同步误差

同步误差:





	Value	Mean	Min	Max	St Dev	Count	Info
Delay	-12.24µs	-58.854617µ	-104.3µ	-3.771µ	30.87µ	4.155k	

Figure: report jiffies

上图可以看出取样 4.155k 次, CH_LOCAL 相比 CH_SYNC 的相位差距最大为 $-3.771 - (-104.3) = 100.529\mu\text{s}$

串口的数据传输会产生一些延迟(尽管发送时无阻塞发送,在中断中接收,直接读寄存器)。串口传输耗时在一个停止位,无检验位,的情况下每字节传输耗时 = $\text{baudrate}/10\text{bit}$. 本例中我们使用的 baudrate 是 $115200*2$ 。

纯软件的 CRC 校验必然带来一定的延时,如果需要校验,在本项目中需要实现 CRC 硬件外设的驱动。

demo 同步后的误差经过多次测试,其中有变换主从角色,最大的误差在 500us 以内。最常见的是 300us 以内。

本次测试代码逻辑不复杂,实时性的处理都在中断上下文,而 STM32F4 使用 Cortex M4 的 Nested Vectored Interrupt Controller (NVIC) 实时性非常高,应该少有优化空间。