Application Notes
ZHCAA18 – 7.2020

如何使用 C2000 CLB 实现 Traction Inverter 应用中的 PWM 输出保护功能

JOHNSON CHEN/RAYNA WANG EP FAE

摘要

在 Traction inverter 应用中，通常会使用 DSP + CPLD 结构来实现整个系统功能。其中 DSP 负责执行电机控制算法，系统控制和对外通讯等功能，CPLD 用来实现不同故障下的 PWM 快速保护功能。

本文重点介绍如何基于 C2000 CLB 实现原来用 CPLD 实现的 PWM 快速保护功能。

通过本方法可以去除 Traction Inverter 系统中的 CPLD，从而将系统架构从 DSP + CPLD 简化为单芯片 C2000。

本方法已在支持 CLB C2000 的芯片如：TMS320F28386,TMS320F38388,TMS320F28378，TMS320F28379，TMS320F280048，TMS320F280049 等上实现，并进行验证。

Contents

Figures

Tables

1 介绍

在 Traction Inverter 应用中，通常会有不同的故障类型如过流保护(简称 OCP)，过压保护(简称 OVP)，IGBT 驱动故障等，基于不同的故障类型及不同的转速情况， 需要在几百纳秒时间内使 PWM 进入不同的保护状态。Traction Inverter PWM 通常会有下面四种状态：

1. Low side ASC: 三相下桥臂 PWM 100% 占空比开通，三相上桥臂 PWM 全部关断.
2. High side ASC: 三相上桥臂 PWM 100% 占空比开通，三相下桥臂 PWM 全部关断.
3. Shut down All PWM: 6 路 PWM 全部关断.
4. Normal PWM Operation: 没有保护信号，PWM 正常输出。

因此通常会选择 DSP + CPLD 的架构来实现整个系统，如图一，其中 DSP 负责执行电机控制算法，系统控制和对外通讯等功能 ，CPLD 用来实现不同故障下的 PWM 快速保护功能如 ASC，六路 PWM 全关断等。

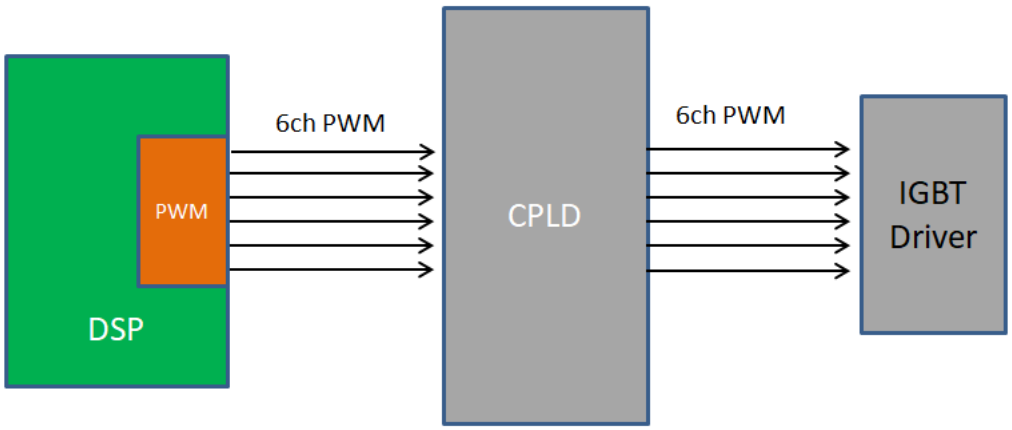

Figure 1. DSP + CPLD Traction inverter 系统架构

CPLD 实现的 PWM 保护功能有下面两个特点：

1. 可以在几百纳秒时间内使 PWM 进入不同的保护状态。
2. 这个保护和 CPU 代码执行是完全独立的，即使 CPU 跑飞，CPLD 也能够保护 PWM 输出，从而让系统进入安全状态

下面将介绍一种基于 C2000 CLB 来实现 Traction Inverter 应用中 PWM 输出保护的方案。

CLB 是可编程的逻辑硬件单元，与 CPU 软件代码执行独立， 即使 CPU 跑飞也不影响 CLB 的 PWM 输出保护功能。

通过 CLB 实现 CPLD 的 PWM 保护功能，可以去除 traction inverter 系统中的 CPLD，从而将系统架构从 DSP + CPLD 简化为单芯片 C2000。

2 使用 CLB 实现 PWM 保护的方法

2.1 具体的 PWM 保护逻辑

通常的保护逻辑如下：

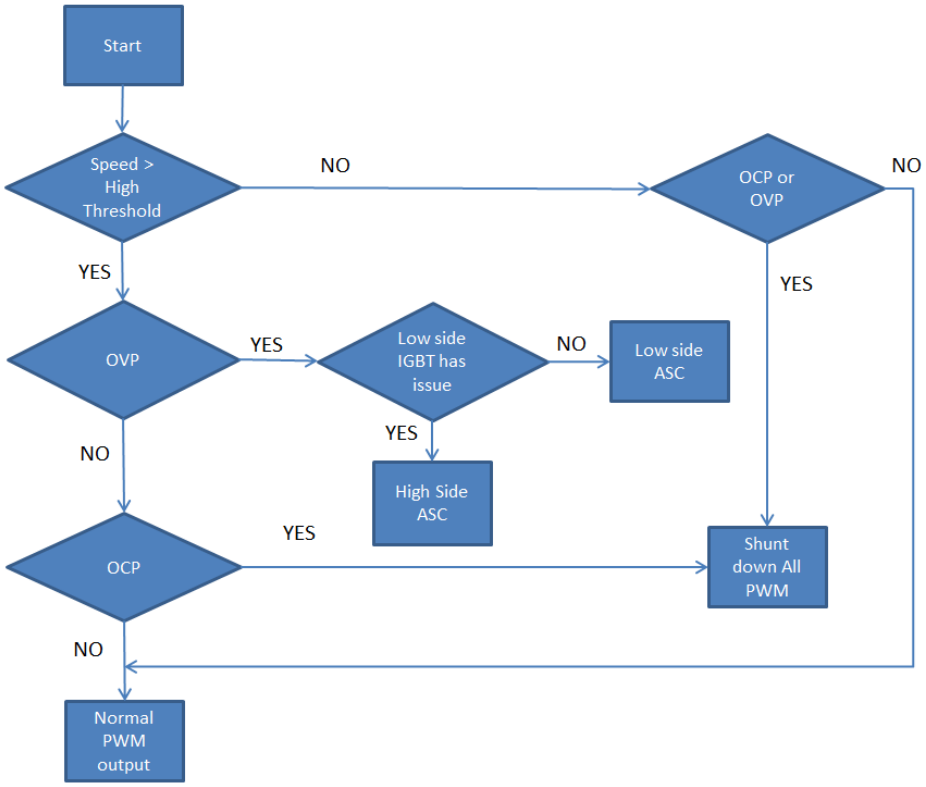

Figure 2. PWM 保护逻辑流程图

上图中相关缩写含义如下：

OCP：过流保护信号 OVP：过压保护信号 ASC:主动短路

2.2 实现方法

输入输出接口如下：

CLB 外部输入 I/O 信号：

- IGBT 错误信号: 逻辑电平, 1: 表示下桥 IGBT 驱动有问题，0:表示 IGBT 驱动有问题没有问题
- OCP 信号: 过流保护信号, 逻辑电平, 1: 过流信号产生, 0: 电流正常.
- OVP 信号: 过压保护信号, 逻辑电平，1: 过压信号产生, 0: 电压正常.

CLB 其他输入信号:

- 转速范围信号: CPU 计算之后提供给 CLB，1—表示速度 > 高速门限, 0--表示速度在正常范围.
- PWMxA 和 PWMxB 信号来自于 ePWMx 模块.

CLB 输出

- PWMxA/ PWMxB

假设使用条件如下：

1. 使用 ePWM1，ePWM2, ePWM3 模块实现电机控制 6 路 PWM 输出，ePWM1A，ePWM2A, ePWM3A 作为三相上桥臂输出，ePWM1B，ePWM2B, ePWM3B 作为三相下桥臂输出
2. OVP 信号输入到 GPIO32
3. OCP 信号输入到 GPIO19
4. IGBT FAULT 信号输入到 GPIO18
5. 假设 PWM 输出高电平有效，开通表示输出高电平，关断表示输出低电平。

下面为 CLB 实现 PWM 保护的方案架构。

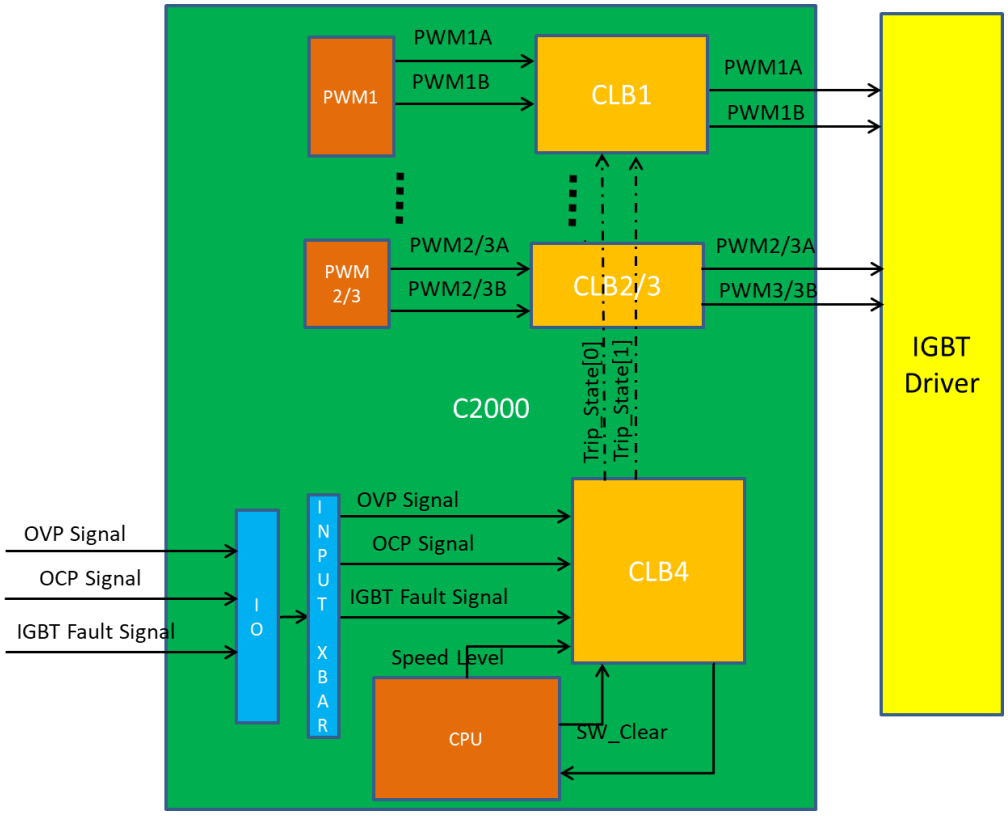

Figure 3. CLB 实现 PWM 保护的方案架构

2.3 CLB TILE 具体资源分配

CLB TILE1 实现 ePWM1 的输出保护及死区插入。

CLB TILE2 实现 ePWM2 的输出保护及死区插入。

CLB TILE3 实现 ePWM3 的输出保护及死区插入。

CLB TILE4 实现下面功能：

1. 对 Speed Level，OVP，OCP，IGBT_FAULT 信号进行逻辑处理，产生相应的保护状态。
2. 对 OVP，OCP 信号进行锁存。

CLB 会产生下面四种 PWM 状态。

Table 1: PWM 保护状态定义

TRIP STATE		
TRIP STATE[1]	TRIP STATE[0]	
0	0	Shut down all PWM
0	1	Active HIGH ASC
1	0	Active LOW ASC
1	1	Normal Operation

CPU 可以通过 CLB TILE4 寄存器 CLB_DBG_OUT 的 28 和 29 位来读取当前的 PWM 保护状态。

关于死区插入:

对于互补输出的 PWM 来说，需要插入死区来保证不同保护状态之间切换的安全，本方法会一直检测 PWMxA/PWMxB 的边缘变化，并插入死区。无论是在正常 PWM 输出模式还是保护状态下， 死区插入功能都会一直工作，确保系统死区正常。FSM0 和 CNT0 用来实现 PWMxA 的死区，FSM1 和 CNT1 用来实现 PWMxB 的死区。

下面为 CLB TILE1 资源使用方法, CLB TILE2 和 CLB TILE3 同理：

Table 2: CLB TILE1 资源使用

Resource	Function	Notes
Inputs		
In0	PWM1A	来自 ePWM1 模块
In1	PWM1B	来自 ePWM1 模块
In2	Trip State Bits from CLB4	CLB4_OUT4 → TRIP_STATE[0]
In3	Trip State Bits from CLB4	CLB4_OUT5 → TRIP_STATE[1]
In4	Not used	
In5	Not used	
In6	Not used	
In7	Not used	
Outputs		
Out0	PWM1A	EPWM1A 输出到 GPIO0
Out1	Not used	
Out2	PWM1B	EPWM1B 输出到 GPIO1
Out3	Not used	
Out4	Not used	
Out5	Not used	
Out6	Not used	
Out7	Not used	
Logic resources		

LUT0	PWMA output based on trip condition	基于 TRIP_STATE[1/0] 位产生输出
LUT1	PWMB output based on trip condition	基于 TRIP_STATE[1/0] 位产生输出
LUT2	Not used	
FSM0	State machine generating the dead time pulse at the falling edge of PWMA	S0 等于 LUT0 输出， S1 反应 PWMA 下降沿产生的死区脉冲
FSM1	State machine generating the dead time pulse at the falling edge of PWMB	S0 等于 LUT1 输出， S1 反应 PWMB 下降沿产生的死区脉冲
FSM2	Not used	
CNT0	Dead time for PWMA	Counter Match1 值在 PWMA 下降沿加载，由 FSM0 来复位和使能。
CNT1	Dead time for PWMB	Counter Match1 值在 PWMB 下降沿加载，由 FSM1 来复位和使能。
CNT2	Not used	
High Level Controller		
HLC	Not used	

CLB TILE4 使用资源和方法如下：

Table 3: CLB TILE 4 资源使用

Resource	Function	Notes
Inputs		
In0	Speed Level	通过 CPU 写 GP_REG[2] 相应位
In1	OVP	通过 InputXbar
In2	OCP	通过 InputXbar
In3	IGBT_FAULT	通过 InputXbar
In4	OVP_sw_clear	通过 CPU 写 GP_REG[2] 相应位
In5	OCP_sw_clear	通过 CPU 写 GP_REG[2] 相应位
In6	Not used	
In7	Not used	
Outputs		
Out0	Not used	
Out1	Not used	
Out2	Not used	
Out3	Not used	
Out4	TRIP_STATE[0]	TRIP_STATE[0] 显示 PWM 保护状态
Out5	TRIP_STATE[1]	TRIP_STATE[0] 显示 PWM 保护状态
Out6	Not used	
Out7	Not used	
Logic resources		
LUT0	Not used	
LUT1	Not used	
LUT2	TRIP_STATE[0]	产生 TRIP_STATE[0] 位
FSM0	Latch OVP	FSM S0 反映 OVP 锁存状态 OVP_sw_clear 位写 1 清除 S0
FSM1	Latch OCP	FSM S0 反映 OCP 锁存状态 OCP_sw_clear 位写 1 清除 S0
FSM2	TRIP_STATE[1]	使用 LUT：产生 TRIP_STATE[1]
CNT0	Not used	
CNT1	Not used	
CNT2	Not used	
High Level Controller		
HLC	Not used	

3 相关参考代码

```
//---------------------------------------------------------------------------

  //变量定义
   uint32_t     Speed = 0;
   uint32_t     Speed_threshold = 0;
   uint32_t     Speed_level;
   uint32_t     OVP_sw_clear = 0;
   uint32_t     OCP_sw_clear = 0;
   uint32_t     PWM_Status;

//---------------------------------------------------------------------------

  //CLB 初始化代码

    CLB_enableCLB(CLB1_BASE);
    CLB_enableCLB(CLB2_BASE);
    CLB_enableCLB(CLB3_BASE);
    CLB_enableCLB(CLB4_BASE);

    initTILE1(CLB1_BASE);
    initTILE2(CLB2_BASE);
    initTILE3(CLB3_BASE);
    initTILE4(CLB4_BASE);

    //configure PWM as CLB input

    // Select Global input instead of local input for all CLB IN
    CLB_configLocalInputMux(CLB1_BASE, CLB_IN0, CLB_LOCAL_IN_MUX_GLOBAL_IN);
    CLB_configLocalInputMux(CLB1_BASE, CLB_IN1, CLB_LOCAL_IN_MUX_GLOBAL_IN);
    CLB_configLocalInputMux(CLB2_BASE, CLB_IN0, CLB_LOCAL_IN_MUX_GLOBAL_IN);
    CLB_configLocalInputMux(CLB2_BASE, CLB_IN1, CLB_LOCAL_IN_MUX_GLOBAL_IN);
    CLB_configLocalInputMux(CLB3_BASE, CLB_IN0, CLB_LOCAL_IN_MUX_GLOBAL_IN);
    CLB_configLocalInputMux(CLB3_BASE, CLB_IN1, CLB_LOCAL_IN_MUX_GLOBAL_IN);

    // Select EPWMxA/B for CLBx, IN0/1
    CLB_configGlobalInputMux(CLB1_BASE, CLB_IN0, CLB_GLOBAL_IN_MUX_EPWM1A);
    CLB_configGlobalInputMux(CLB1_BASE, CLB_IN1, CLB_GLOBAL_IN_MUX_EPWM1B);
    CLB_configGlobalInputMux(CLB2_BASE, CLB_IN0, CLB_GLOBAL_IN_MUX_EPWM2A);
    CLB_configGlobalInputMux(CLB2_BASE, CLB_IN1, CLB_GLOBAL_IN_MUX_EPWM2B);
    CLB_configGlobalInputMux(CLB3_BASE, CLB_IN0, CLB_GLOBAL_IN_MUX_EPWM3A);
    CLB_configGlobalInputMux(CLB3_BASE, CLB_IN1, CLB_GLOBAL_IN_MUX_EPWM3B);

    // Select External for CLBx, IN0/1
    CLB_configGPInputMux(CLB1_BASE, CLB_IN0, CLB_GP_IN_MUX_EXTERNAL);
    CLB_configGPInputMux(CLB1_BASE, CLB_IN1, CLB_GP_IN_MUX_EXTERNAL);
    CLB_configGPInputMux(CLB2_BASE, CLB_IN0, CLB_GP_IN_MUX_EXTERNAL);
    CLB_configGPInputMux(CLB2_BASE, CLB_IN1, CLB_GP_IN_MUX_EXTERNAL);
    CLB_configGPInputMux(CLB3_BASE, CLB_IN0, CLB_GP_IN_MUX_EXTERNAL);
    CLB_configGPInputMux(CLB3_BASE, CLB_IN1, CLB_GP_IN_MUX_EXTERNAL);
```

```
//how to configure PWM output?

CLB_setOutputMask(CLB1_BASE, CLB_OUTPUT_00, true);//output EPWM1A
CLB_setOutputMask(CLB1_BASE, CLB_OUTPUT_02, true);//output EPWM1B
CLB_setOutputMask(CLB2_BASE, CLB_OUTPUT_00, true);//output EPWM2A
CLB_setOutputMask(CLB2_BASE, CLB_OUTPUT_02, true);//output EPWM2B
CLB_setOutputMask(CLB3_BASE, CLB_OUTPUT_00, true);//output EPWM3A
CLB_setOutputMask(CLB3_BASE, CLB_OUTPUT_02, true);//output EPWM3B

//--------------------------------------------------------------------------
 //configure CLB4 outputs as CLB1/2/3 input signals through CLB X-Bar

// Select Global input instead of local input for all CLB IN
CLB_configLocalInputMux(CLB1_BASE, CLB_IN2, CLB_LOCAL_IN_MUX_GLOBAL_IN);//CLB4_OUT4 -> TRIP_STATE[0]
CLB_configLocalInputMux(CLB1_BASE, CLB_IN3, CLB_LOCAL_IN_MUX_GLOBAL_IN);//CLB4_OUT5 -> TRIP_STATE[1]
CLB_configLocalInputMux(CLB2_BASE, CLB_IN2, CLB_LOCAL_IN_MUX_GLOBAL_IN);
CLB_configLocalInputMux(CLB2_BASE, CLB_IN3, CLB_LOCAL_IN_MUX_GLOBAL_IN);
CLB_configLocalInputMux(CLB3_BASE, CLB_IN2, CLB_LOCAL_IN_MUX_GLOBAL_IN);
CLB_configLocalInputMux(CLB3_BASE, CLB_IN3, CLB_LOCAL_IN_MUX_GLOBAL_IN);

//select the CLB_AUXSIGx channels from global inputs
CLB_configGlobalInputMux(CLB1_BASE, CLB_IN2, CLB_GLOBAL_IN_MUX_CLB_AUXSIG0);//CLB4_OUT4 -> TRIP_STATE[0]
CLB_configGlobalInputMux(CLB1_BASE, CLB_IN3, CLB_GLOBAL_IN_MUX_CLB_AUXSIG1);//CLB4_OUT5 -> TRIP_STATE[1]
CLB_configGlobalInputMux(CLB2_BASE, CLB_IN2, CLB_GLOBAL_IN_MUX_CLB_AUXSIG0);
CLB_configGlobalInputMux(CLB2_BASE, CLB_IN3, CLB_GLOBAL_IN_MUX_CLB_AUXSIG1);
CLB_configGlobalInputMux(CLB3_BASE, CLB_IN2, CLB_GLOBAL_IN_MUX_CLB_AUXSIG0);
CLB_configGlobalInputMux(CLB3_BASE, CLB_IN3, CLB_GLOBAL_IN_MUX_CLB_AUXSIG1);

// Inputs set to external input pin
CLB_configGPInputMux(CLB1_BASE, CLB_IN2, CLB_GP_IN_MUX_EXTERNAL);
CLB_configGPInputMux(CLB1_BASE, CLB_IN3, CLB_GP_IN_MUX_EXTERNAL);
CLB_configGPInputMux(CLB2_BASE, CLB_IN2, CLB_GP_IN_MUX_EXTERNAL);
CLB_configGPInputMux(CLB2_BASE, CLB_IN3, CLB_GP_IN_MUX_EXTERNAL);
CLB_configGPInputMux(CLB3_BASE, CLB_IN2, CLB_GP_IN_MUX_EXTERNAL);
CLB_configGPInputMux(CLB3_BASE, CLB_IN3, CLB_GP_IN_MUX_EXTERNAL);

//configure CLB x-bar for CLB4 output
XBAR_setCLBMuxConfig(XBAR_AUXSIG0, XBAR_CLB_MUX13_CLB4_OUT4);
XBAR_setCLBMuxConfig(XBAR_AUXSIG1, XBAR_CLB_MUX15_CLB4_OUT5);//TRIP_STATE[0] TRIP_STATE[1] of CLB4 output

XBAR_enableCLBMux(XBAR_AUXSIG0, XBAR_MUX13);//configure CLB4_OUT4 as CLB_xbar input signals
XBAR_enableCLBMux(XBAR_AUXSIG1, XBAR_MUX15);//configure CLB4_OUT5 as CLB_xbar input signals


//--------------------------------------------------------------------------
//configure IGBT fault/OCP/OVP signals as input through GPIO input X-Bar

// Select Global input instead of local input for all CLB IN
CLB_configLocalInputMux(CLB4_BASE, CLB_IN0, CLB_LOCAL_IN_MUX_GLOBAL_IN);//speed level via. GP_REG[0] by CPU
CLB_configLocalInputMux(CLB4_BASE, CLB_IN1, CLB_LOCAL_IN_MUX_GLOBAL_IN);//OVP via InputXbar
CLB_configLocalInputMux(CLB4_BASE, CLB_IN2, CLB_LOCAL_IN_MUX_GLOBAL_IN);//OCP via InputXbar
CLB_configLocalInputMux(CLB4_BASE, CLB_IN3, CLB_LOCAL_IN_MUX_GLOBAL_IN);//IGBT FAULT via InputXbar
CLB_configLocalInputMux(CLB4_BASE, CLB_IN4, CLB_LOCAL_IN_MUX_GLOBAL_IN);//OVP_sw_clear via. GP_REG[4] by
                                                                         //CPU rising edge
```

```
    CLB_configLocalInputMux(CLB4_BASE, CLB_IN5, CLB_LOCAL_IN_MUX_GLOBAL_IN);//OCP_sw_clear via. GP_REG[5] by
                                                                              //CPU rising edge

    //select the CLB_AUXSIGx channels from global inputs
    CLB_configGlobalInputMux(CLB4_BASE, CLB_IN1, CLB_GLOBAL_IN_MUX_CLB_AUXSIG2);
    CLB_configGlobalInputMux(CLB4_BASE, CLB_IN2, CLB_GLOBAL_IN_MUX_CLB_AUXSIG3);
    CLB_configGlobalInputMux(CLB4_BASE, CLB_IN3, CLB_GLOBAL_IN_MUX_CLB_AUXSIG4);

    //Select External for CLB4, IN1/IN2/IN3
    CLB_configGPInputMux(CLB4_BASE, CLB_IN0, CLB_GP_IN_MUX_GP_REG);
    CLB_configGPInputMux(CLB4_BASE, CLB_IN1, CLB_GP_IN_MUX_EXTERNAL);
    CLB_configGPInputMux(CLB4_BASE, CLB_IN2, CLB_GP_IN_MUX_EXTERNAL);
    CLB_configGPInputMux(CLB4_BASE, CLB_IN3, CLB_GP_IN_MUX_EXTERNAL);
    CLB_configGPInputMux(CLB4_BASE, CLB_IN4, CLB_GP_IN_MUX_GP_REG);
    CLB_configGPInputMux(CLB4_BASE, CLB_IN5, CLB_GP_IN_MUX_GP_REG);

    //configure GPIO32/GPIO19 and GPIO18 as input of InputXbar_customer
    GPIO_setDirectionMode(32,GPIO_DIR_MODE_IN);
    GPIO_setQualificationMode(32,GPIO_QUAL_SYNC);
    GPIO_setPinConfig(GPIO_32_GPIO32);
    GPIO_setDirectionMode(19,GPIO_DIR_MODE_IN);
    GPIO_setQualificationMode(19,GPIO_QUAL_SYNC);
    GPIO_setPinConfig(GPIO_19_GPIO19);
    GPIO_setDirectionMode(18,GPIO_DIR_MODE_IN);
    GPIO_setQualificationMode(18,GPIO_QUAL_SYNC);
    GPIO_setPinConfig(GPIO_18_GPIO18);

    // Configure Input-XBAR INPUT1/2/3 as GPIO32/19/18 respectively
    XBAR_setInputPin(XBAR_INPUT1, 32);
    XBAR_setInputPin(XBAR_INPUT2, 19);
    XBAR_setInputPin(XBAR_INPUT3, 18);

    // Configure CLB-XBAR AUXSIG2/3/4 as INPUT1/2/3 respectively
    XBAR_setCLBMuxConfig(XBAR_AUXSIG2, XBAR_CLB_MUX01_INPUTXBAR1);
    XBAR_setCLBMuxConfig(XBAR_AUXSIG3, XBAR_CLB_MUX03_INPUTXBAR2);
    XBAR_setCLBMuxConfig(XBAR_AUXSIG4, XBAR_CLB_MUX05_INPUTXBAR3);

    XBAR_enableCLBMux(XBAR_AUXSIG2, XBAR_MUX01);
    XBAR_enableCLBMux(XBAR_AUXSIG3, XBAR_MUX03);
    XBAR_enableCLBMux(XBAR_AUXSIG4, XBAR_MUX05);


    CLB_selectInputFilter(CLB1_BASE, CLB_IN4, CLB_FILTER_RISING_EDGE);
    CLB_selectInputFilter(CLB1_BASE, CLB_IN5, CLB_FILTER_RISING_EDGE);
//---------------------------------------------------------------------------
   //读取 CLB PWM 保护状态代码

    PWM_Status = (CLB_getOutputStatus(CLB4_BASE)& 0x30000000)>>28;// CLB4_CLB_DBG_OUT  bit28 / bit29

//---------------------------------------------------------------------------
```

```
//CPU 传递Speed_level信息及清除OVP，OCP 锁存代码。
 //Detect the speed level and input the signal to CLB
 if(Speed > Speed_threshold)
 {
     Speed_level = 1;
     CLB_setGPREG(CLB4_BASE, 0x1);//
 }
 else
 {
     Speed_level = 0;
 }

 if(OVP_sw_clear)
 {
     CLB_setGPREG(CLB4_BASE, (0x10) | Speed_level);
     OVP_sw_clear=0;
 }
 else
 {
     CLB_setGPREG(CLB4_BASE, Speed_level);
 }

 if(OCP_sw_clear)
 {
     CLB_setGPREG(CLB4_BASE, (0x20) | Speed_level);
     OCP_sw_clear=0;
 }
 else
 {
     CLB_setGPREG(CLB4_BASE, Speed_level);
 }

//---------------------------------------------------------------------
```

4 测试结果

测试条件：

a. CCS 10.0.0.00010

b. LAUNCHXL-F28379D LaunchPad™

c. C2000Ware_2_00_00_02

d. Ch1 为 PWM1A 信号，Ch2 为 OVP 信号，Ch3 为 PWM1B 信号，

实际测试波形如下：

下面图 6，图 7，图 8 分别为 High side ASC 保护波形，Low side ASC 保护波形 和 Shut down All PWM 保护波形。

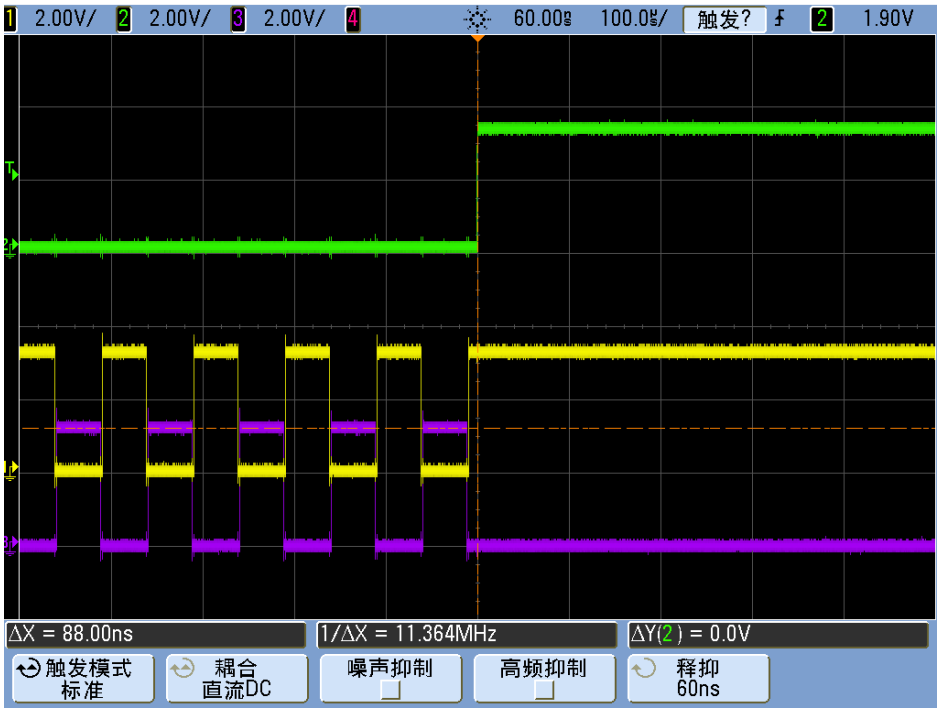

Figure 4. High side ASC 保护波形

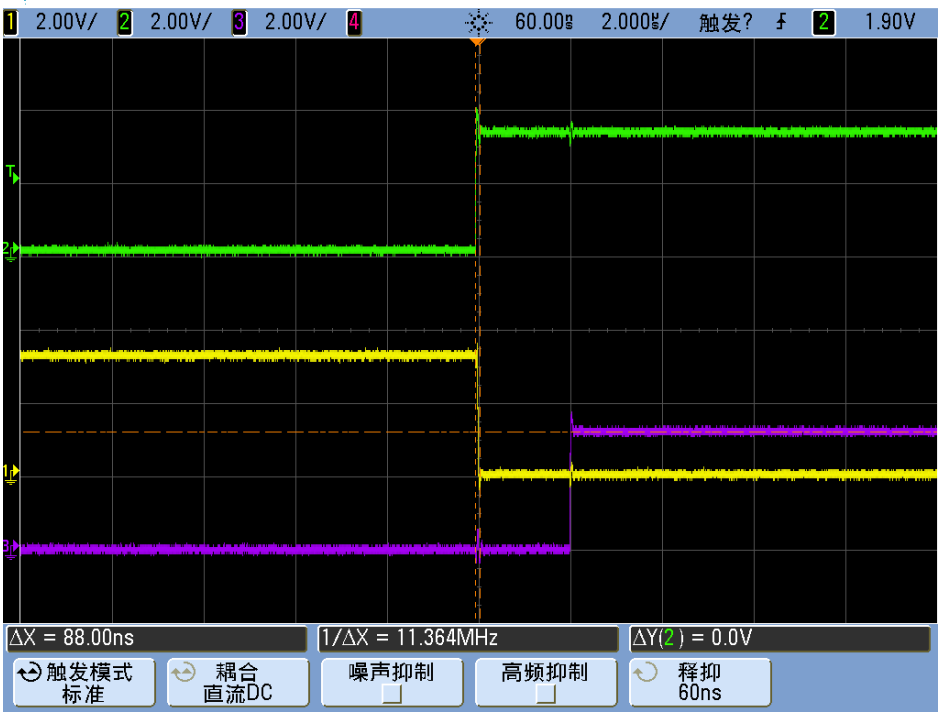

Figure 5. Low side ASC 保护波形

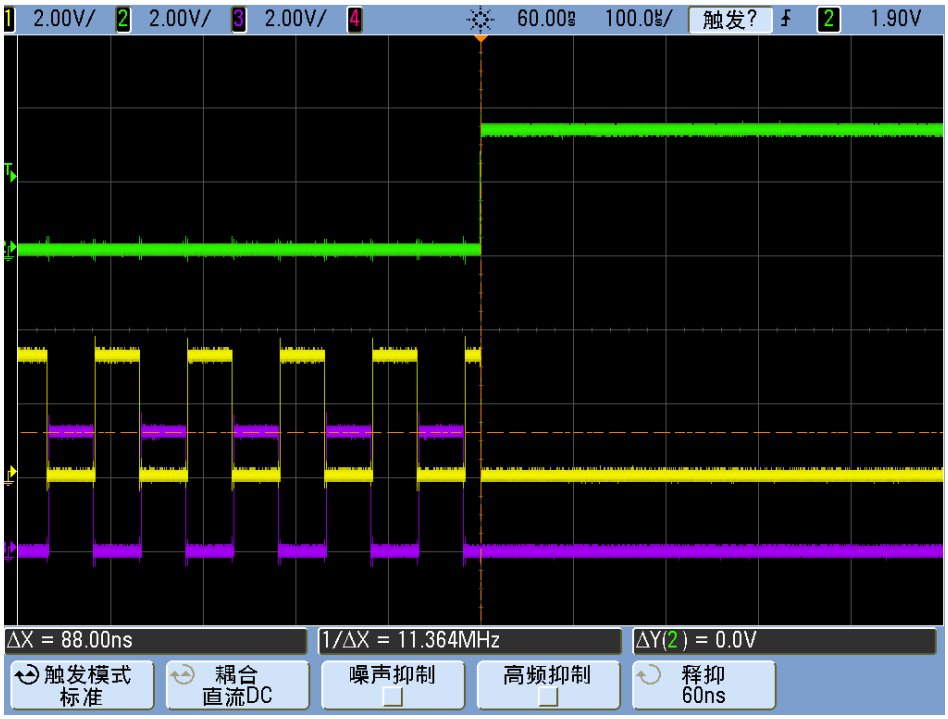

Figure 6. Shut down All PWM 保护波形

从图 9 可以看到，从 OVP 产生到 CLB 把 PWM 关断时间间隔为 88nS。总共时间小于 100nS.，完全可以满足 Traction Inverter 的 PWM 输出保护的响应时间要求。

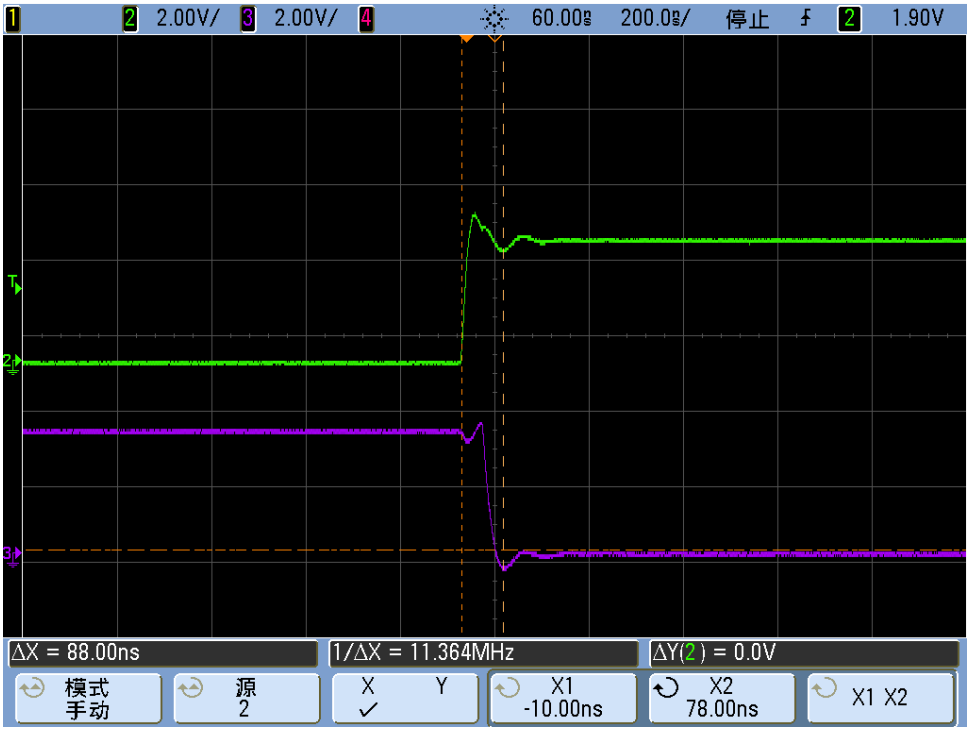

Figure 7.　CLB PWM 关断响应时间

5 注意事项

1. 并非所有 C2000 芯片都有 CLB 模块，要实现本文方案，请选择带有 CLB 模块的 C2000 芯片。
2. 本实现方法共使用了 4 个 CLB TILEs 。
3. CLB tools 的 BOUNDARY 里面配置的输入信号只是仿真时起作用，输入的实际信号并不能在 CLB tools 的 BOUNDARY 里面配置，而是需要在 CPU 的初始代码里配置，具体可参考本文初始化代码。
4. CLB 参考例程路径如下(以 F2837xD 为例)

   C:\ti\c2000\C2000Ware_2_00_00_02\driverlib\f2837xd\examples\cpu1\clb

5. 如需要本方法的 CLB 配置文件，请联系 TI 的相关销售和技术支持。

6 参考文献

1. ***TMS320F28379D Dual-Core Delfino Technical Reference Manual (SPRUHM8H)***
2. ***TMS320F28379S Delfino Microcontrollers Technical Reference Manual (SPRUHX5C)***
3. ***TMS320F280049 MicrocontrollersTechnical Reference Manual (SPRUI33C)***
4. ***CLB Tool Users Guide (SPRUIR8)***

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265