Application Note

展示 TI C2000™ CLA 的独特功能的软件示例

Himanshu Chaudhary and Aravindhan Karuppiah

摘要

实现超高性能计算和高效处理对于解决当今复杂的实时控制问题至关重要。实时控制系统是闭环控制系统，其中使用一个紧凑的时间窗口来收集数据、处理该数据并更新系统以满足性能目标。TI 的控制律加速器 (CLA) 旨在与 C28x CPU 并行执行实时控制算法，使 C2000 器件的计算性能加倍。本应用报告将讨论 CLA 的一些特性，并使用一些简单的软件示例进行演示。这些独立示例作为 C2000Ware 的一部分提供，可快速用于探索和评估 CLA 的功能。

内容

插图清单

表格清单

商标

1 引言

该 CLA 是完全可编程的独立 32 位浮点 CPU，专为优化数学密集型计算而设计，可显著提升控制算法的性能。与执行指令和处理中断的标准传统处理器不同，CLA 实际上是任务驱动状态机，最多可支持 8 个用户定义的任务。除了提供计算能力外，CLA 的独特之处还包括提供最小延迟并轻松访问主要控制外设。因此，CLA 非常适合用于实现快速控制环路，从而释放 C28x 上的带宽以运行额外的控制环路并执行其他与诊断和通信相关的任务。本应用报告的后续各节将详细讨论 CLA 的这些独特功能，并通过 C2000Ware 包 [2] 中提供的一些简单软件示例进行演示。更多有关 CLA 架构和指令集的详细信息，请参阅 [1]、[3]。

本文档中讨论的示例，都可以在安装 C2000Ware v3.01.00.00 或最新版本后在以下目录中找到：

- C:\ti\c2000\C2000Ware_<version_number>\driverlib\f28004x\examples\cla
- C:\ti\c2000\C2000Ware_<version_number>\libraries\math\CLAmath\c28\examples
- C:\ti\c2000\ C2000Ware_<version_number>\libraries\control\DCL\c28\examples

讨论的示例工程包括：

- cla_ex4_pwm_control
- cla_ex5_adc_just_in_time
- cla_ex6_cpu_offloading
- cla_ex7_shared_resource_handling

2 CLA 直接访问主要外设

大多数实时控制算法用于对系统执行三个主要任务：激励、采样和控制。对系统进行激励涉及更新 PWM 寄存器，对系统进行采样涉及访问 ADC 结果寄存器，而对系统进行控制涉及控制环路数学计算。CLA 作为独立的数学处理器，还能够访问用于控制应用的所有主要外设（如 EPWM、ADC、ECAP、EQEP、CMPSS 等）的寄存器。因此，CLA 可以执行采样和驱动以及计算控制逻辑，并且能够在没有任何 C28x 参与的情况下独立执行整个控制任务。

“cla_ex4_pwm_control”示例展示了如何通过 CLA 直接控制 PWM 信号输出。此示例的方框图如图 2-1 所示。在此示例中，EPWM1 配置为以 100KHz 的固定频率在两个通道上生成互补信号，而 EPWM4 配置为以 10KHz 的频率触发周期性的 CLA 控制任务。CLA 任务 1 采用一个非常简单的逻辑来改变 EPWM1 输出的占空比，方法是针对每次迭代将其增加 0.1，同时还需要将其保持在 0.1-0.9 的范围内。下面的代码序列说明了如何在 CLA 任务中按原样使用现有的 C28x driverlib API（作为 C2000Ware 的一部分提供）来更新 EPWM 寄存器，从而避免与 CLA 相关的任何额外软件开发工作。CLA 任务也可以采用类似的方式访问其他共享外设的主要寄存器。请注意，无法在 .cla 文件的开头初始化 CLA 全局变量，因此，该示例还说明了在专用 CLA 任务（CLA 任务 8，在初始化时由 C28x 软件触发）内初始化所有 CLA 全局变量的系统方法。

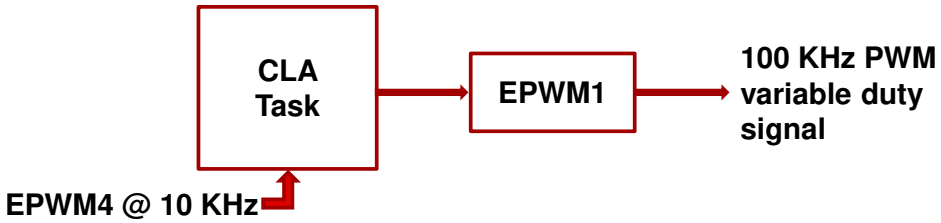

图 2-1. 使用 CLA 直接进行 PWM 控制

```
__attribute__((interrupt)) void Cla1Task1 ( void )
{
    //
    // Uncomment this to debug the CLA while connected to the debugger
    //
    __mdebugstop();
    //
    // Write to the COMPA register to realize a particular duty value
    //
    EPWM_setCounterCompareValue(EPWM1_BASE, EPWM_COUNTER_COMPARE_A,
                                (uint16_t)(duty * EPWM1_PERIOD + 0.5f));

    //
    // Update duty value and use the limiter
    //
    duty += 0.1f;
    duty = (duty > 0.9f) ? 0.1f : duty;
    //
    // Clear EPWM4 interrupt flag so that next interrupt can come in
    //
    EPWM_clearEventTriggerInterruptFlag(EPWM4_BASE);
}
```

3 CLA 的低中断延迟

在任何实时控制应用中，采样到输出的延迟（定义为检测、处理和驱动之间经过的时间）是系统的一项重要考虑因素。CLA 的低延迟架构在增加总体系统吞吐量的同时减少了这个采样到输出的时间。之所以可以实现这一点，是因为 CLA 以任务为导向，而非中断驱动状态机，并且不使用中断与硬件同步。相反，该器件支持多达八个独立任务，每个任务都映射到硬件事件，例如计时器，或 ADC 上的数据可用性，等等。在 CLA 上启动的任务会在不涉及任何中断或嵌套的情况下运行至完成，因此不必考虑传统上基于中断的处理器通常涉及到的任何上下文切换开销。所以，CLA 处理数据时几乎没有延迟，最终会降低采样到输出的延迟并实现更快的系统响应。图 3-1 举例说明了任务驱动状态机 (TDM) 和一个中断驱动状态机 (IDM) 间的区别。

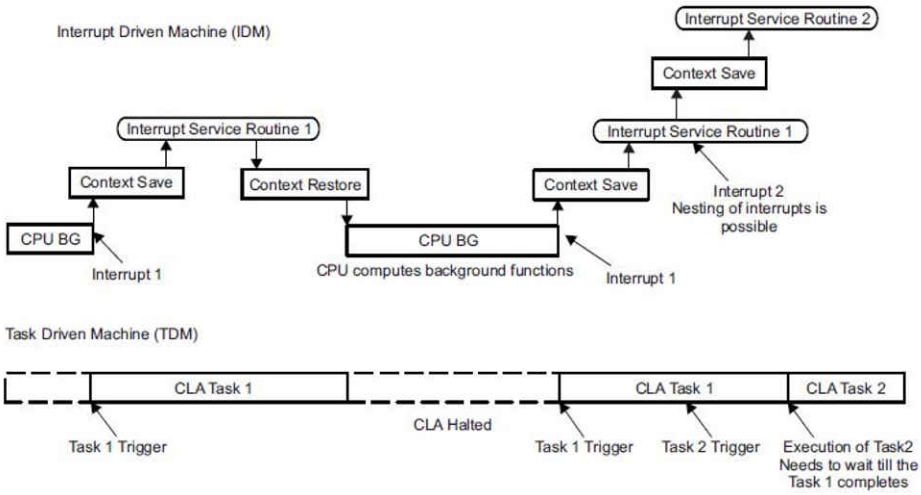

图 3-1. 中断驱动状态机与任务驱动状态机

CLA 的低中断响应可与 TI 内部 ADC 的提前中断特性结合使用，进一步降低采样到输出的延迟。ADC 可以配置为在转换完成前在采样结束时生成提前中断脉冲。ADC 生成的这个提前中断脉冲可用于触发 CLA 任务，此任务将使得 CLA 能够在 ADC 结果寄存器中有转换结果时立即读取结果。这种即时采样与 CLA 的低中断响应相结合，可实现更快的系统响应和更高频率的控制环路。转换前的可用时间可有效地用于 CLA 任务内任何必要的预处理步骤，如图 3-2 所示。为了实现即时读取，可以根据 N 周期 ADC 转换的 CLA 流水线活动来计算读取请求应该放置在哪条确切指令中。如图 3-3 所示，N-2 指令将即时进入 R2 阶段以读取结果寄存器。对于标准的 12 位 ADC 配置和时钟分频器为 4 的情况，N 为 42。要根据 ADC 的配置确定正确的 N 值，请参阅器件特定的数据表 [4]。

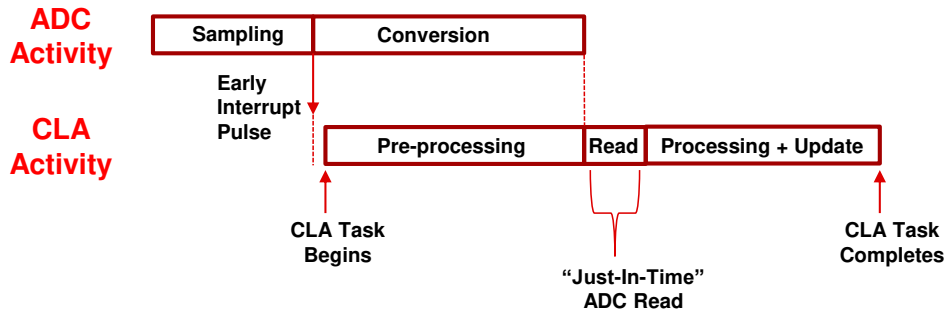

图 3-2. ADC 生成的用于触发 CLA 任务的提前中断

ADC Activity	CLA Activity	F1	F2	D1	D2	R1	R2	E	W
Sample									
Sample									
...									
Sample									
Conversion(Cycle 1)	Interrupt Received								
Conversion(Cycle 2)	Task Startup								
Conversion(Cycle 3)	Task Startup								
Conversion(Cycle 4)	I(Cycle 4)	I(Cycle 4)							
Conversion(Cycle 5)	I(Cycle 5)	I(Cycle 5)	I(Cycle 4)						
Conversion(...)	...	...	...	...	...	...	...		
Conversion(Cycle N-6)	I(Cycle N-6)	I(Cycle N-6)	I(Cycle N-7)	I(Cycle N-8)	I(Cycle N-9)	I(Cycle N-10)	I(Cycle N-11)		
Conversion(Cycle N-5)	I(Cycle N-5)	I(Cycle N-5)	I(Cycle N-6)	I(Cycle N-7)	I(Cycle N-8)	I(Cycle N-9)	I(Cycle N-10)		
Conversion(Cycle N-4)	I(Cycle N-4)	I(Cycle N-4)	I(Cycle N-5)	I(Cycle N-6)	I(Cycle N-7)	I(Cycle N-8)	I(Cycle N-9)		
Conversion(Cycle N-3)	I(Cycle N-3)	I(Cycle N-3)	I(Cycle N-4)	I(Cycle N-5)	I(Cycle N-6)	I(Cycle N-7)	I(Cycle N-8)		
Conversion(Cycle N-2)	Read RESULT	Read RESULT	I(Cycle N-3)	I(Cycle N-4)	I(Cycle N-5)	I(Cycle N-6)	I(Cycle N-7)		
Conversion(Cycle N-1)			Read RESULT	I(Cycle N-3)	I(Cycle N-4)	I(Cycle N-5)	I(Cycle N-6)		
Conversion(Cycle N-0)				Read RESULT	I(Cycle N-3)	I(Cycle N-4)	I(Cycle N-5)		
Conversion Complete					Read RESULT	I(Cycle N-3)	I(Cycle N-4)		
RESULT Latched						Read RESULT	I(Cycle N-3)		
RESULT Available							Read RESULT		

图 3-3. 提前中断脉冲的 CLA 流水线活动

“cla_ex5_adc_just_in_time”示例利用上述概念，即使在非常高的采样频率下也能“即时”读取 ADC 数据。如图 3-4 所示，EPWM1 配置为生成频率为 1MHz 的 PWM 输出信号，该信号也用于在每个周期触发 ADC 采样。该示例还利用了 TI 的 5 类 ADC 中新增的特性，允许根据编程的 OFFSET 值将提前中断脉冲延迟几个周期。因此，ADCA 配置为对通道 0 上的输入进行采样，并在 S/H + 偏移周期结束时生成提前中断。此中断用于触发 CLA 控制任务。该 CLA 任务采用控制逻辑根据读取的 ADC 值更新 PWM 输出的占空比。提前中断特性和 CLA 的低中断延迟搭配使用，使得应用能够执行任何必要的前期工作，以便能够在 ADC 结果可用时立即对其进行处理，并且还能在下一个中断到达之前完成对 PWM 输出的更新。因此，所有三个步骤（采样、处理和驱动）都在 1MHz 周期内完成。如 CLA 任务的以下代码片段所示，为了进行说明，采用了 3 点移动平均滤波器来模拟处理序列，并且在读取 ADC 结果之前执行了少量表示为预处理代码的滤波序列步骤以便利用转换前的可用时间。

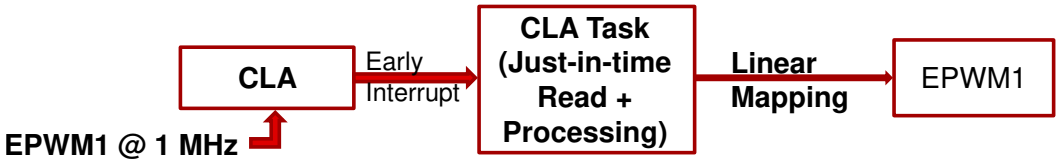

图 3-4. “即时”ADC 读取示例展示

```
//
// Pre-processing for implementing moving average filter, takes 13 cycles
// This is just to illustrate how cycles can be utilized to do some pre-
// processing before ADC result latches.Based on the cycles taken by
// pre-processing code, ADC interrupt offset need to be programmed
//
data_read_total = data_read + data_read_prev;
data_read_prev2 = data_read_prev;
data_read_prev = data_read;
//
// Reading ADC just-in-time
//
data_read = HWREGH(ADCARESULT_BASE + ADC_RESULTx_OFFSET_BASE + ADC_SOC_NUMBER0);
//
// "data_read_total" stores the cumulative sum of current and last 2 data elements
//
data_read_total += data_read;
//
// Taking average of 3 elements, normalizing for 12-bit and mapping to output duty
// linearly in the range  0.1-0.9
// duty = 0.1 + (0.9-0.1) * ((data_read_total / 3) / 2^12 )
//
duty = 0.1f + (data_read_total / (15360.0f));
//
// Writing to the COMPA register for realizing computed duty value
//
HWREGH(EPWM1_BASE + EPWM_O_CMPA + 0x1U) = (uint16_t)(duty * EPWM1_PERIOD + 0.5f);
```

ADC 的提前中断 OFFSET 值需要根据预处理所消耗的周期进行调整，以便“即时”读取 ADC 数据。此示例根据示例标头中显示的计算，使用 OFFSET 值 20。下面显示了此 ADC 配置的编程序列。实际用例可能涉及不同的预处理步骤，因此需要相应地对中断 OFFSET 值进行编程。

```
//
// Set pulse positions to early
//
ADC_setInterruptPulseMode(ADCA_BASE, ADC_PULSE_END_OF_ACQ_WIN);
//
// Set interrupt offset delay as 20 cycles based on the calculation
// shown in example header
//
ADC_setInterruptCycleOffset(ADCA_BASE, 20);
```

4 CLA 强大的数学计算能力

CLA 还为 C2000 器件提供强大的 32 位浮点处理能力，并显著提升了控制算法中常用的典型数学函数的性能。性能强大的 CLA 指令集支持在单周期中使用并行加法或减法运算进行浮点乘法，还支持在单周期中计算平方根反比。为了方便使用 CLA 进行软件开发，一个广泛的常用浮点数学函数集合（表 4-1 列出了其中的一些函数）被打包到称为“CLA 数学”的单个库中，这个库作为 C2000Ware 的一部分提供。这个源代码库包括几个专为 CLA 架构编写的可由 C 调用的汇编数学函数。

除了基本的数学例程之外，TI 还提供了数字控制库（即 DCL，作为 C2000Ware 的一部分提供），包括在 CLA CPU 上进行良好实现的标准控制例程，其中一些例程如表 4-1 所示。这些可由 C 调用的汇编控制例程可以在 CLA 应用任务中调用，以便在 CLA CPU 上实现数字控制器。除了库源代码，还提供了一些示例向用户展示如何将库集成到其工程中以及如何使用数学例程或控制例程。这些示例可在简介一节所述的示例目录中找到，可用于探索和评估 CLA 的计算能力。

表 4-1. CLA 数学和控制库例程

库	例程	说明	周期数
CLA 数学	CLAcos	在 CLA 上计算余弦	28
	CLAsin	在 CLA 上计算正弦	28
	CLAacos	在 CLA 上计算反余弦	24
	CLAasin	在 CLA 上计算反正弦	22
	CLAatan	在 CLA 上计算反正切	41
	CLAlog10	在 CLA 上计算对数（以 10 为底）	29
	CLAexp	在 CLA 上计算指数	41
	CLAdiv	在 CLA 上计算浮点除法	13
	CLAisqrt	在 CLA 上计算平方根反比	14
	CLAsqrt	在 CLA 上计算平方根	16
DCL	DCL_runPID_L1	在 CLA 上运行理想形式的 PID 控制器	53
	DCL_runPID_L2	在 CLA 上运行并行形式的 PID 控制器	45
	DCL_runPI_L1	在 CLA 上运行理想形式的 PI 控制器	34
	DCL_runDF13_L1	在 CLA 上运行 DF13 完全补偿器	61
	DCL_runDF13_L2	在 CLA 上运行 DF13 即时补偿器	20
	DCL_runDF13_L3	在 CLA 上运行 DF13 部分补偿器	58

5 将快速控制环路卸载到 CLA

各种实时控制应用涉及在单个器件上实现多个控制环路。然而，从处理器带宽的角度来讲，将多个控制系统集成在单个控制器上，而同时保持低系统成本，仍然具有挑战性。CLA 是与 C28x 主内核完全并行的处理器，可为 C28x 系列带来并发的控制环路执行。CLA 有其自身的程序和数据总线，并且独立于 MCU 上的主内核之外执行。如节 2 和节 3 所述，CLA 的独特之处包括提供最小延迟并轻松访问主要控制外设，使 CLA 能够完全从 C28x 卸载快速控制算法任务。将控制任务卸载到 CLA 还有其他好处，例如执行中的抖动降低和控制环路的确定性操作。之所以可以实现这一点，是因为 CLA 以任务为导向，而非由中断服务驱动的状态机，并且 CLA 上的任务不能被中断，从而保证了控制环路的确定性。在管线型 CPU 中，如果 CPU 在 ISR 被接收时正在执行分支类型语句，ISR 可被延迟“n”个数量周期。但是，这对于 CLA CPU 来说不是问题，因为它本质上是任务驱动的，会在空闲状态下等待，直到周期性任务触发才开始执行。因此，将快速控制任务卸载到 CLA 并在 C28x 上运行其余任务有助于提高总体系统性能，同时减少执行中的抖动。

“cla_ex6_cpu_offloading”示例说明了当涉及多个需要多个 CPU (C28x) 带宽的控制任务和后台任务时，如何以较好的方式将控制环路从 C28x 卸载到 CLA。图 5-1 显示在该示例中仿真了两个控制环路。较快的环路（环路 1）以 200KHz 的频率运行，而较慢的环路（环路 2）以 20KHz 的频率运行。这两个环路都使用 PI 控制器来控制单个 PWM 输出的占空比，但贡献权重不同：较快的环路占 80%，较慢的环路占 20%。两个环路的输入由 ADCA 和 ADCB 进行采样，每个环路都有多个 SOC 可用于滤除输入中的任何噪声。主环路中还有一个连续运行的后台任务，可根据用户配置的开关“system_OFF”禁用或启用整个系统，包括 PWM 输出和控制环路。请注意，CCS 调试器时钟不能用于对 CLA 例程进行分析，因此该示例中采用了基于 GPIO 的分析技术来分析这两个任务的表现。已使用 GPIO2 和 GPIO 3 来实现此目的。

图 5-2 显示了在没有使用 CLA 的情况下，两个控制任务全部都在 C28x 上运行时的流程图。在这种情况下，总 CPU (C28x) 利用率超过可调度的利用率限制 (UB)，因此在这种情况下系统是可调度的。通过观察图 5-3 中显示的分析波形也可以进一步证实这一点。请注意，在 GPIO3 上没有观察到切换，这清楚地表明优先级较低的环路 2 任务没有机会完成，后台任务也不会完成。

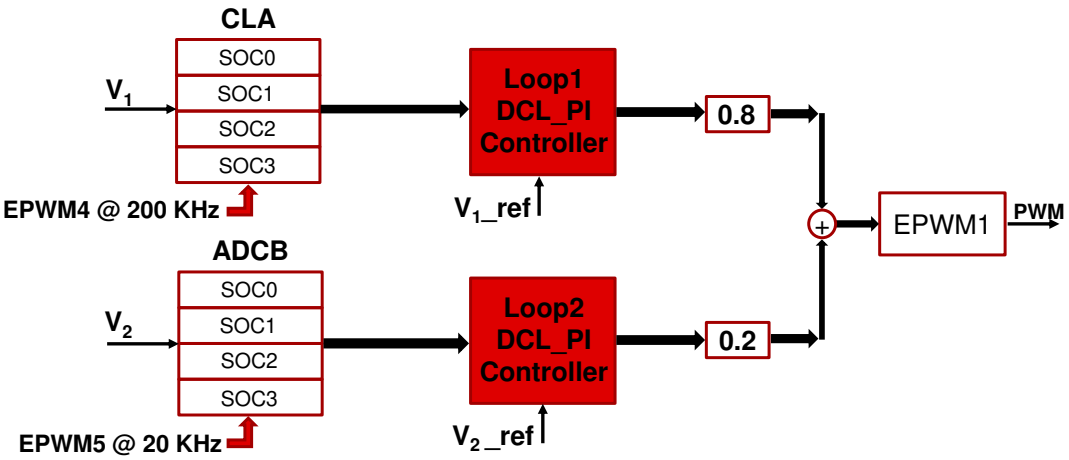

图 5-1. 双控制环路示例展示

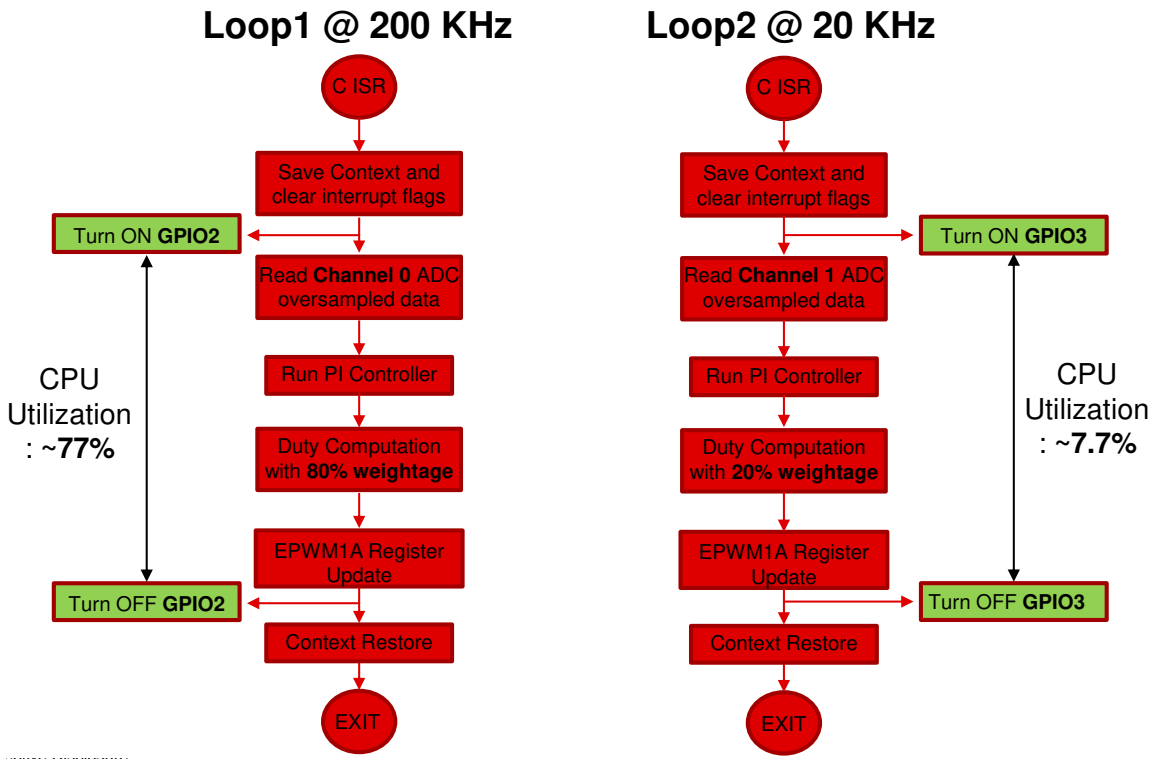

图 5-2. 两个任务都在 C28x 上运行的流程图

图 5-3. C28x 上运行的两个任务的分析波形

由于系统不可通过 C28x 进行调度，因此可以将其中一项控制任务卸载到 CLA 以满足系统要求。由于 CLA 提供非常低的中断延迟，因此最好将快速控制任务卸载到 CLA，这样也会释放 C28x 上的最大带宽，从而可将带宽用于执行后台任务和其他系统任务。图 5-4 所示为当较高频率环路 1 任务卸载到 CLA 时这两个任务的流程图。通过将 CLA 用于并发环路执行，用于控制任务的 C28x 利用率下降到大约 7.7%，从而可以正确执行其他后台任务。将任务卸载到 CLA 可使系统在这种情况下完全可调度，从图 5-5 中显示的分析波形也可以明显看出这一点。该示例允许用户通过将工程构建选项中的预定义符号“run_loop1_cla”更新为 1，快速方便地将环路 1 任务从 C28x 卸载到 CLA。

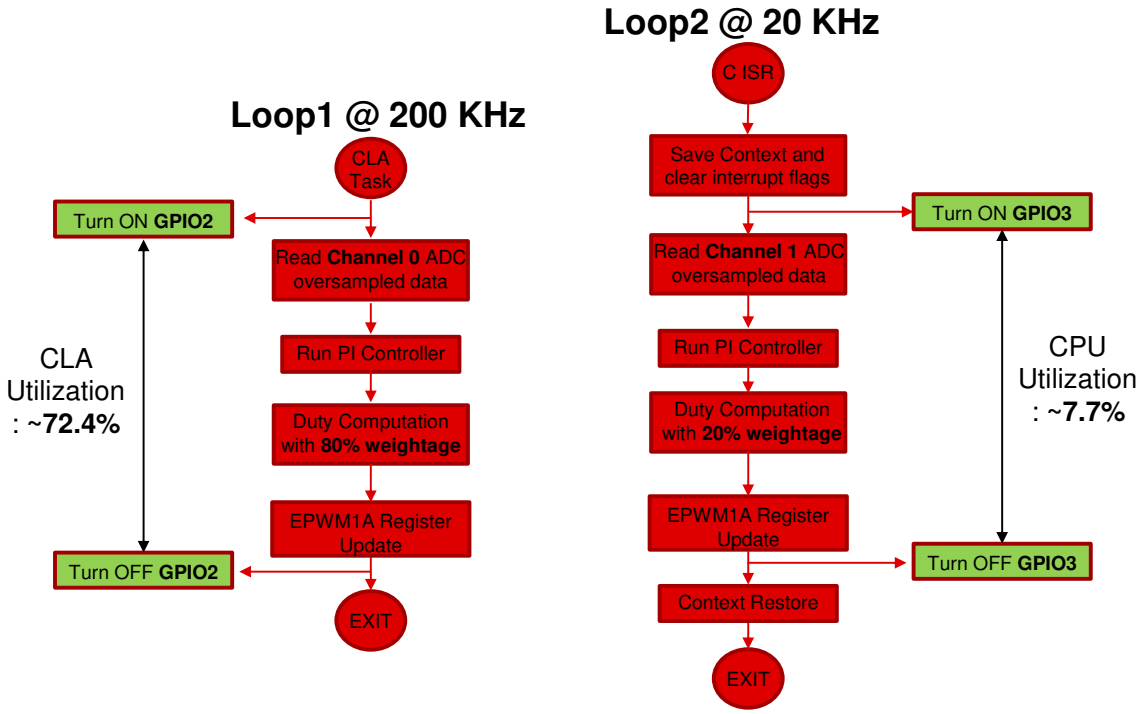

图 5-4. 将环路 1 任务卸载到 CLA 的流程图

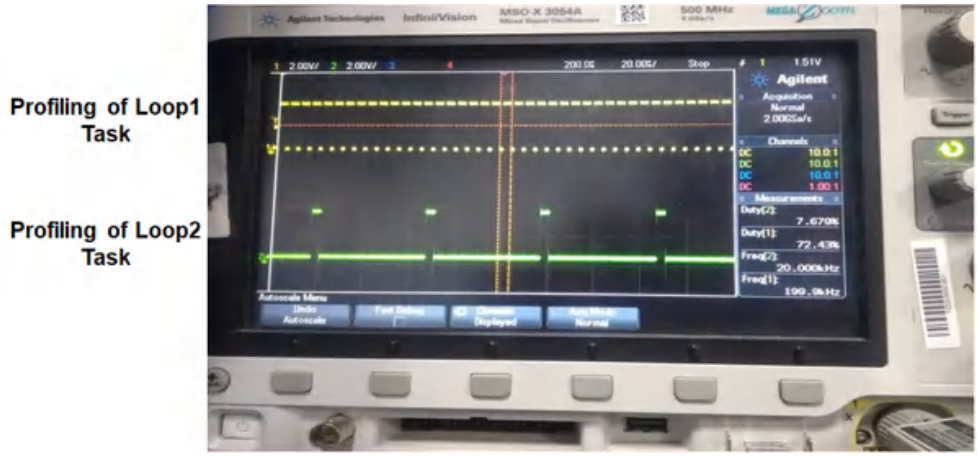

图 5-5. 将环路 1 任务卸载到 CLA 的分析波形

5.1 跨 C28x/CLA 处理共享资源

使用 CLA 可以有效地从 C28x 卸载控制任务，并在 C2000 器件上实现并发控制环路执行，同时具有前面几节讨论的许多其他优势。但必须注意的是，仍然会在它们之间共享外设，并且对共享寄存器进行并发读取-修改-写入操作会产生数据争用条件，最终导致数据冲突或功能不正确。理想情况下，最好避免 CLA 和 C28x 在运行时同时对同一外设进行任何更新，但如果无法避免，则必须小心处理共享资源的冲突。

“cla_ex7_shared_resource_handling”示例说明了这种情况，其中 C28x 和 CLA 以不同频率独立执行对同一 (AQCSFRC) 寄存器的并发读取-修改-写入操作，这会在 C28x 和 CLA 之间产生竞态条件，并可能导致两者其中一个引发的更新丢失或被覆盖。这是一个标准的临界区问题，可以使用互斥等软件握手机制进行处理，但大多数实时控制应用程序对时间敏感，无法承受额外的软件周期开销。此示例建议使用一种基于硬件的替代技术来巧妙地调度 CLA 和 C28x 任务，从而避免对共享资源的访问发生重叠。基于硬件的调度技术利用了 EPWM 模块的可编程相移机制。

如 图 5-6 所示，C28x ISR 和 CLA 任务分别以 10KHz 和 100KHz 独立运行。C28x ISR 由 EPWM4 周期性触发，并通过软件控制 AQCSFRC 的 CSFB 位来切换 EPWM1B 输出。CLA 任务由 EPWM5 触发，并通过软件控制 AQCSFRC 的 CSFA 位来切换 EPWM1A 输出（请参阅器件 TRM [1] 了解该寄存器的更多详细信息）。因此，在此过程中，C28x 和 CLA 对 AQCSFRC 寄存器执行了重叠的读取-修改-写入操作，如图 5-7 中的分析波形所示。因此，由于 CLA 而对 AQCSFRC 进行的更新会被覆盖，根据从图 5-7 的 EPWM1A 输出波形中观察到的尖峰可以明显看出这一点。

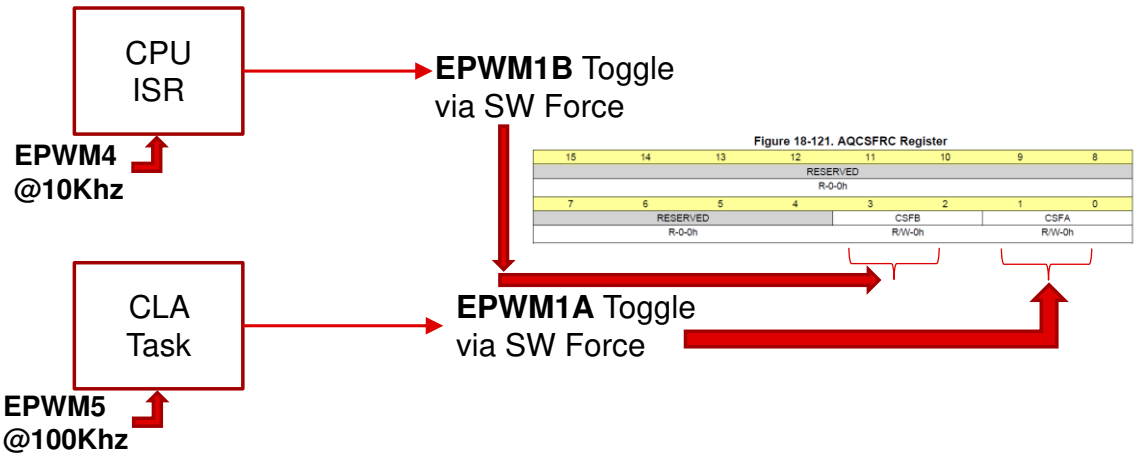

图 5-6. C28x/CLA 的并发读取-修改-写入

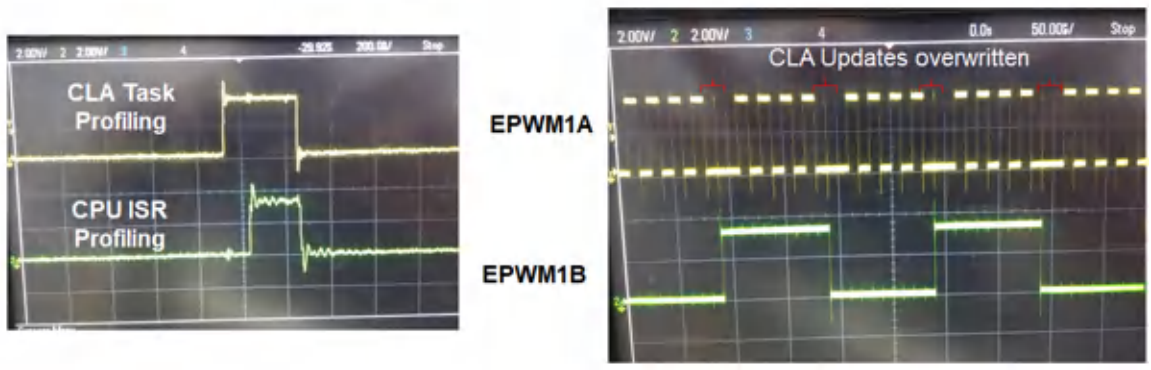

图 5-7. 禁用相移时的分析波形和输出波形

为解决上述问题，可以使用 EPWM 模块的相移机制（如图 5-8 所示）来高效地调度 CLA 任务和 C28x ISR。EPWM4 在发生每个零事件时生成同步脉冲，并向 EPWM5 提供 20 个周期的相移。这样 CLA 任务和 C28x ISR 都会以原始频率（100KHz 和 10KHz）运行，但 CLA 任务比 C28x ISR 超前 20 个周期的相位偏移，如图 5-9 中的分析波形所示。始终不会发生对 AQCSFRC 进行并发读取-修改-写入操作的情况，EPWM1A 和 EPWM1B 输出的行为方式符合要求，不会出现任何失真，如图 5-9 所示。因此，建议的基于硬件的调度技术有助于避免 C28x 和 CLA 之间的数据竞态条件，还有助于通过避免任何同步访问来实现两个处理引擎的真正并行执行，从而尽可能提高器件的整体性能。

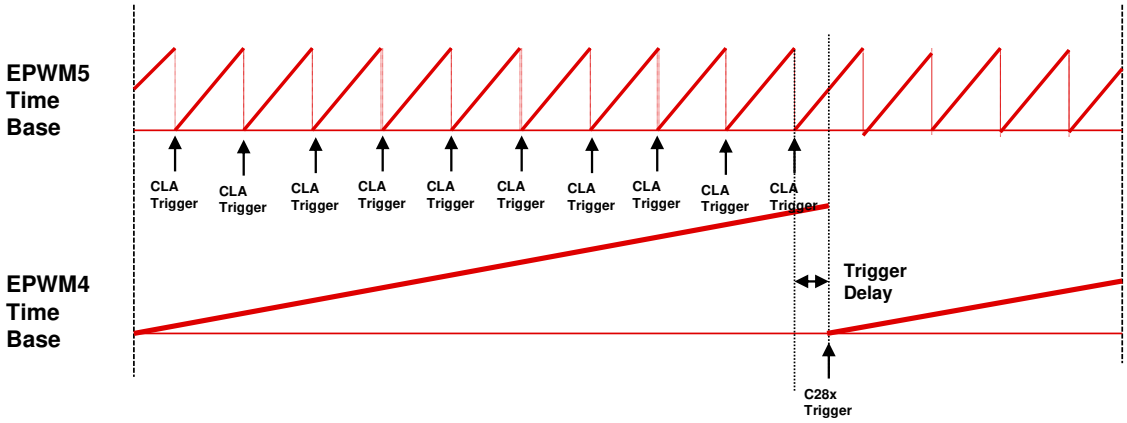

图 5-8. 基于 EPWM 的相移技术

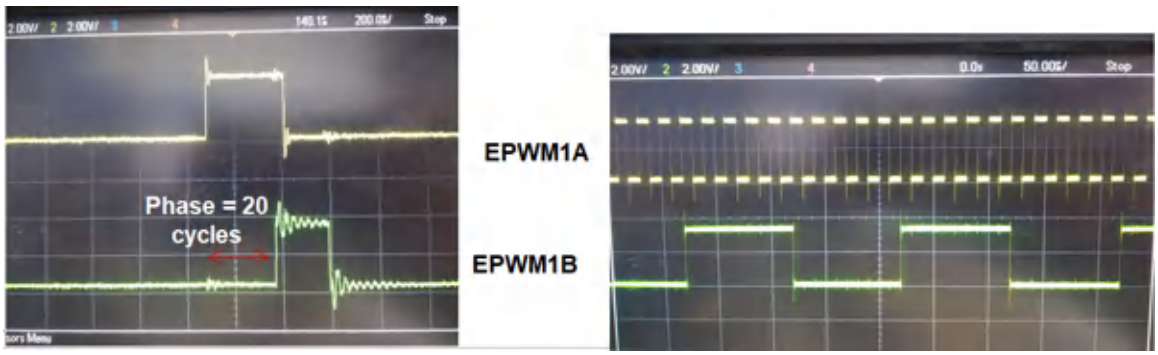

图 5-9. 启用相移时的分析波形和输出波形

6 总结

TI 的控制律加速器 (CLA) 提供的差异化特性支持在 C2000 器件上高效执行并发控制环路。CLA 经过专门设计，旨在提高实时 MCU 上的控制密集型数学例程的性能。CLA 的低延迟任务驱动架构非常独特，可降低采样到输出的延迟，这一点在控制应用中非常重要。对主要控制外设的直接访问和强大的浮点处理能力使 CLA 能够完全从主 CPU (C28x) 卸载控制任务，从而释放其带宽以执行其他系统任务。CLA 为 C2000 器件提供了额外的处理功能，并提高了器件的整体性能。本报告中讨论的用于调度 CLA 任务的相移机制可用于从器件中提取最大处理带宽。与基于硬件的控制律实现方案相比，CLA 的另一个主要优势是灵活性。CLA 是完全可编程的，开发人员可以自由修改他们的控制系统，而无需花费时间和高昂的成本重新设计基于硬件的解决方案。与 C28x 类似，C2000 C 编译器 [5] 允许使用 C 语言对 CLA 进行编程，因此在 CLA 上移植现有算法或开发更新的算法非常方便。本应用报告中讨论的各种软件示例演示了 CLA 的主要功能，可用作在相关应用中采用 CLA 这些独特功能的参考。这些示例非常易于使用，除了标准的 TI ControlCard 之外，不需要任何特殊的硬件平台来探索和评估 CLA 的性能。除这些示例外，还有各种 Digital Power SDK [6] 解决方案展示了如何使用 CLA 来降低各种数字电源解决方案 [7] 中的总体 C28x 负载。更多有关使用 CLA 进行软件开发和调试的详细信息，请参阅 [8]。

7 参考文献

1. 德州仪器 (TI)：*TMS320F28004x 微控制器技术参考手册*
2. 适用于 C2000 MCU 的 C2000Ware
3. CLA 实践技术讲座
4. 德州仪器 (TI)：*具有连接管理器的 TMS320F2838x 微控制器数据表*
5. CLA C 编译器
6. 适用于 C2000 MCU 的 Digital Power SDK
7. CLA 在谷底开关升压功率因数校正 (PFC) 参考设计中的应用
8. *C2000™ CLA 软件开发指南*

8 修订历史记录

Changes from Revision * (May 2020) to Revision A (November 2022)	Page
• 更新了整个文档中的表格、图和交叉参考的编号格式	1

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265