Application Report

智能步进电机驱动器控制

Jose Quiñones

摘要

此文档作为对表 1-1 中所列器件数据表的补充。下表详细介绍的技术能改善对双极步进电机驱动器中集成分度器的实时控制，同时可通过 MSP430 微控制器获取可编程的加速和减速曲线，并实现速度控制和位置控制。

表 1-1. 集成式步进电机驱动器

器件	最大推荐电压 (V)	满量程电流 (A)	Rdson (mΩ) LS+HS FET 典型值 (25°C 时)	控制	失速检测	智能调优	分类
DRV8434S	48	2.5	330	SPI	有	可编程	工业
DRV8434A	48	2.5	330	GPIO	有	固定	工业
DRV8434	48	2.5	330	GPIO	无	固定	工业
DRV8428	33	1	1500	GPIO	无	可编程	工业
DRV8426	33	1.5	900	GPIO	无	可编程	工业
DRV8425	33	2	550	GPIO	无	可编程	工业
DRV8424	33	2.5	330	GPIO	无	可编程	工业
DRV8436	48	1.5	900	GPIO	无	可编程	工业
DRV8889-Q1	45	1.5	900	SPI	有	可编程	汽车
DRV8899-Q1	45	1	1200	SPI	无	可编程	汽车

内容

插图清单

表格清单

商标

所有商标均为其各自所有者的财产。

1 引言和问题说明

驱动步进电机不是一件简单的事情。在直流电机的端子上施加电压，电机便会立即旋转，而对于步进电机而言，必须小心地控制磁场换向，才能实现电机转动。就在不久之前，这种电磁换向是通过对功能强大的微处理器进行编码，进而协调输入功率级的相位和电流信息来实现的。

随着单片集成电路的集成度越来越高，能够更加轻松地将通过代码生成的所有模块放入硬件中。独立 IC 现在甚至无需宝贵的微控制器资源，便可控制换相和微步进等非常复杂的操作。

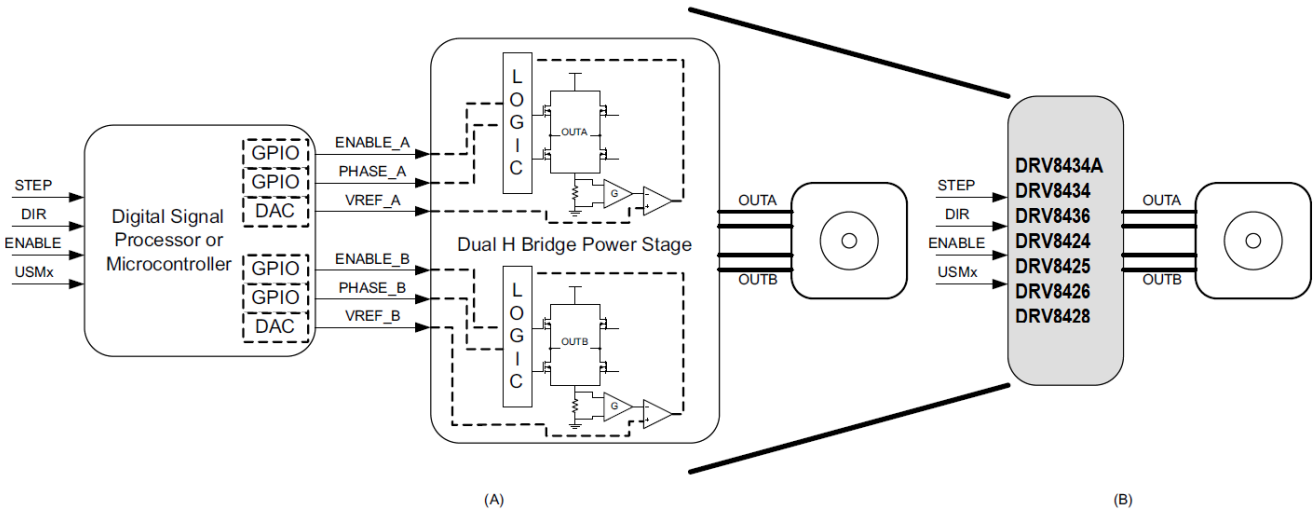


图 1-1. 步进电机控制逻辑和功率级

图 1-1 所示为当微控制器内负责实现步进电机换向的代码与功率级均包含在单芯片解决方案中时的集成度。请注意，在这两种情况中都存在一系列简单的控制信号。STEP 脉冲用于生成步进或微步进；DIR 信号定义旋转的方向；ENABLE 线路决定功率级启用与否；用户模式位用于选择微步进角。

不过，还可以使用微控制器来更好地控制步进电机。速度及位置控制、加速和减速以及归零等操作还需要准确度和精确度，这些可通过微控制器轻松实现。我们必须思考的问题是：与步进电机运动相关的所有参数是否都要由应用处理单元进行计算，或者是否有尺寸更小、更加经济实惠的微控制器可用于处理这些任务？

如果要控制多个步进电机，那么使用更小的微控制器来控制每个驱动器以执行上述操作会非常有利。这样一来，应用处理器可利用其实时资源来合理地协调多个密集应用，而小型微控制器则负责步进电机的复杂控制工作。

本应用手册详细介绍了采用 MSP430F2132 微控制器和 DRV8434/24 器件的方案，其中 DRV8434/24 器件具有内部分度器双极步进电机微步进功率级。这两个器件共同构成了一个模块，该模块既能通过 I^2C 总线接收控制器的命令，又能执行所有的相关操作来控制步进电机的速度和位置。为了充分利用各种可用资源，我们添加了一系列 GPIO 端子，用于为主处理器提供额外的功能。图 1-2 所示为该建议方案的方框图。

虽然该方案中考虑采用由 GPIO 控制的步进电机驱动器，但可调整为采用由 SPI 控制的驱动器。一些新型 TI 步进电机驱动器集成了失速检测功能，因而无需外部归零和行程终端传感器，并能够保护电气和机械系统避免出现磨损。DRV8434S 等步进电机驱动器采用 SPI 控制和数字电路来提供失速检测，而 DRV8434A 采用 GPIO 控制和模拟电路来实现该功能。此外，新型 TI 步进电机驱动器提供集成的电流检测功能，因而无需外部电流检测电阻，这样不仅缩减了成本和 PCB 空间，而且减少了功率损耗和发热。另外，全新器件中的集成智能调优技术还提高了效率，并更大程度地减少了步进电机的可闻噪声。

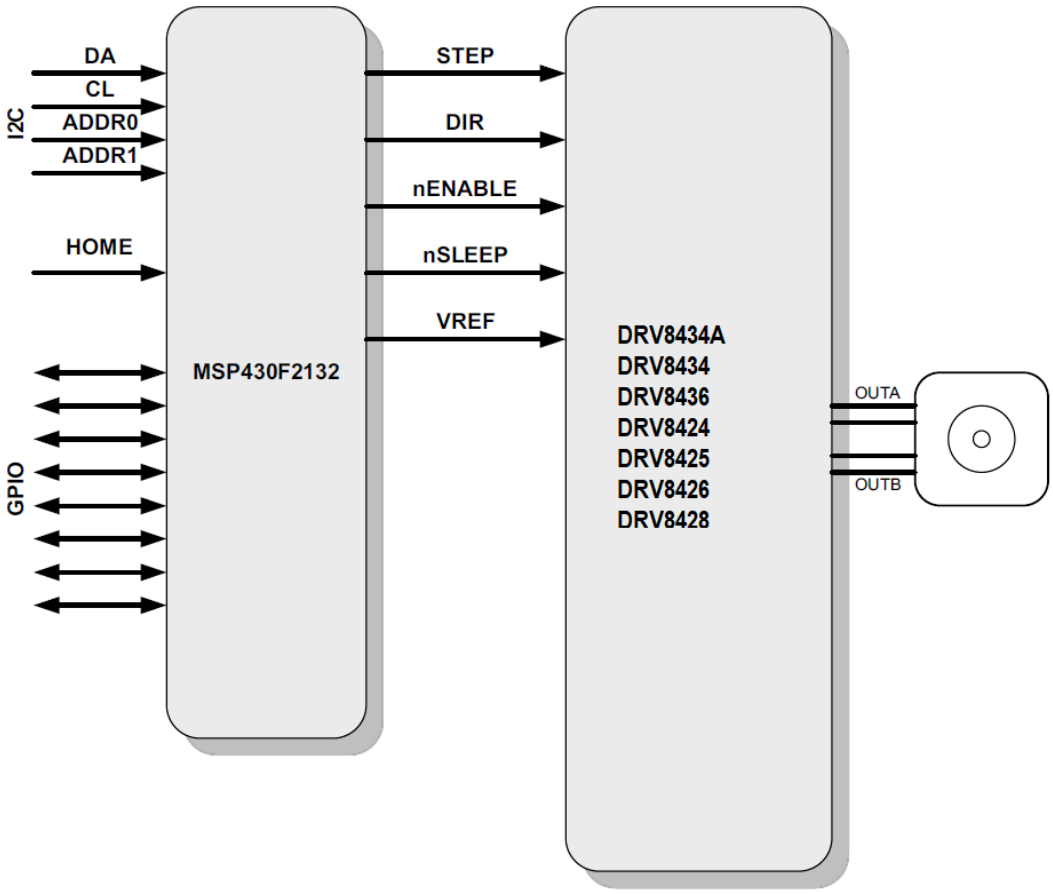


图 1-2. 智能步进电机控制器方框图

2 步进电机控制高级功能

2.1 STEP 驱动：加速、速度控制和减速曲线

步进电机提供了一种无需轴角编码器或旋转变压器等闭环机制即可控制速度的方法。在微步进内部分度器驱动器上，这种开环控制是通过在 STEP 输入端调制频率来实现的。STEP 输入端的每个脉冲都会变为步进电机的一个机械步进运动。因此，可以肯定地说，知道在 STEP 输入端施加的频率大小，便能知道步进电机的实际步进速率。而且只要在应用工作期间将电流、电压和扭矩等正确参数值保持在合理范围内，上述方法始终有效。

不过，我们无法直接对任何指定的步进电机施加任意频率或步进速率。由于定子的旋转磁场与转子的永久磁体背后的相关机制，只有请求的速度小于电机制造商提供的启动频率（以 FS 表示）参数，步进电机才会开始转动。例如，如果特定步进电机的 FS 为 300SPS（每秒步进数），那么当频率为 400SPS 时，电机很可能无法启动。

应用所需的速率大于 FS，因此务必要根据加速曲线调整电机换向，从而以低于最大 FS 的速率启动，然后相应地增加速度，直到达到所需速度。

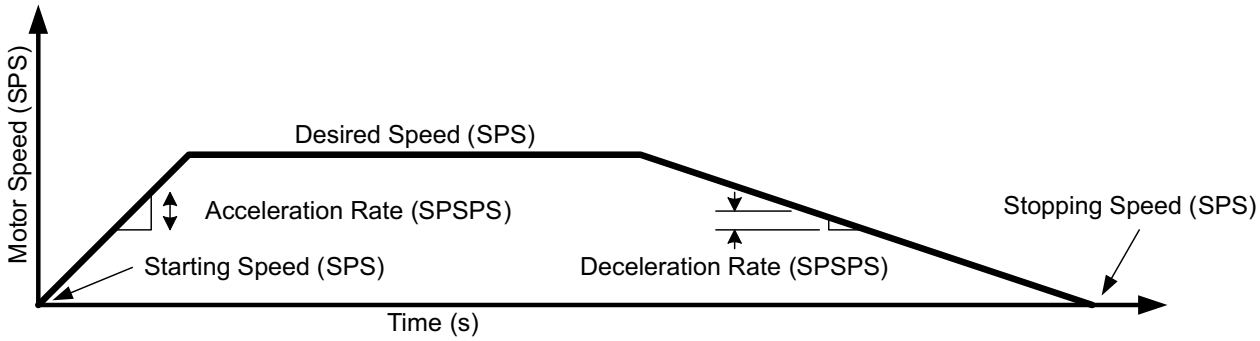


图 2-1. 典型步进电机加速和减速曲线

图 2-1 所示为典型的加速和减速曲线，其中：

启动速度是电机开始转动时所用的 STEP 频率，小于电机的额定 FS。单位为 SPS，其中 STEPS 表示全步进。

加速率是指 STEP 频率每秒增加的变化量，单位为 SPSPS。

所需速度是指应用要求电机旋转的 STEP 频率。它表示加速曲线结束时的 STEP 频率，单位为 SPS。

减速度是指 STEP 频率每秒减少的变化量，单位为 SPSPS。

停止速度是指减速曲线结束及电机停止时的 STEP 频率。本应用手册将停止速度视为与启动速度相同，单位为 SPS。

2.2 电机加速

Start Stepper 命令会首先以 StartingSpeed 变量表示的启动频率发出步进，同时，必须使用计时器以此频率生成 STEP 脉冲。本应用手册使用计时器 A0.2 来设置 STEP 信号，并使用计时器 A0.0 将该信号清零。脉冲宽度为 32 个时钟脉冲，即 2μs。DRV8825 需要宽度至少为 1μs 的 STEP 脉冲，因此通过此方案可生成符合要求的脉冲。

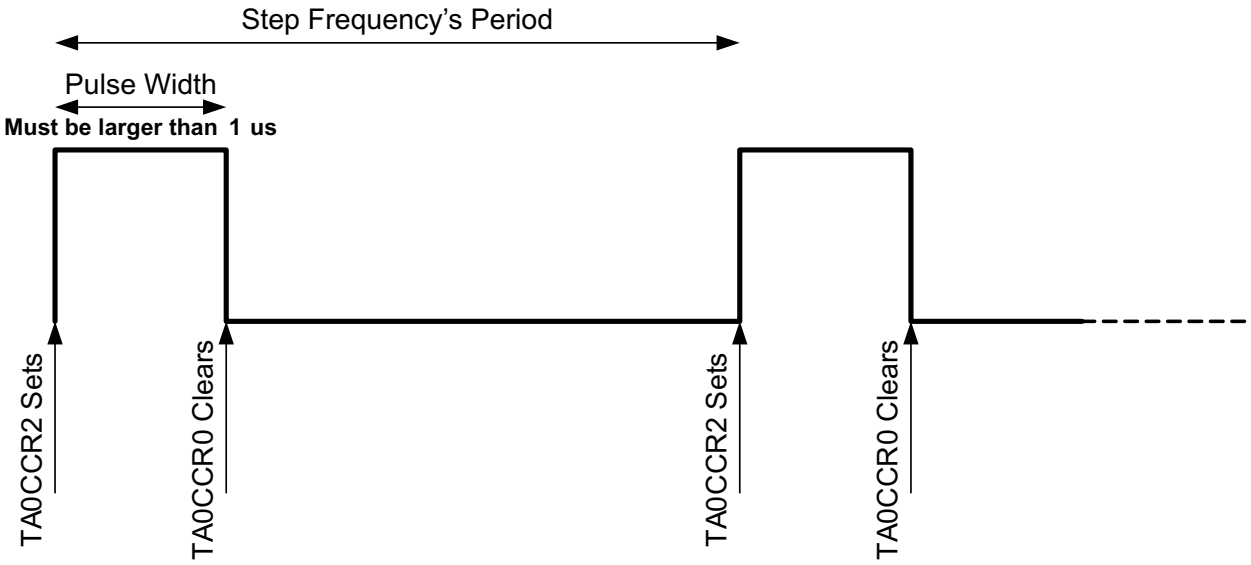


图 2-2. 用于生成 STEP 脉冲的计时器 A0

计时器 A0.2 的中断子例程负责使计时器 A0.0 硬件复位以及对自身进行编程，从而安排生成下一个步进。

另外还需要一个计时器函数来生成速度控制曲线的加速部分。该应用选择的微控制器只有两个可用的计时器，因此必须做好权衡，后文将对此进行简要介绍。

计时器 A1.1 与计时器 A1.0 一起用于生成 PWM 输出，而 PWM 输出将用于生成模数转换器信号，通过添加 RC 滤波器来驱动 DRV8825 的 VREF 模拟输入引脚。然后借助该技术，可实时控制步进电机电流。

尽管所有计时器资源似乎都已被占用，但我们还可利用计时器 A1 的一个功能。如果我们对 PWM 发生器进行编码，从十进制 0 一直计数到十进制 249，那么我们便有 250 个时间单元。当频率为 16MHz 时，每个时间单元等于 62.5ns，因此 250 个时间单元就相当于 15.625μs。通过将这一结果乘以 8，我们可以得到一个相当有用的时基 (125μs)，即每秒 4000 个计时器节拍。

这意味着，我们能够将步进电机速率增加到高达每秒 4000Hz。然后，我们可使用一个简单的数学公式来计算加速时间间隔和加速增量参数。以下代码片段显示了用于计算这两个加速参数的公式：

```
void AccelTimeCompute(unsigned int AccelDecelRate)
{
  if (AccelDecelRate <= 4000)
    {
    AccelerationTime = 4000/AccelDecelRate;
    AccelerationIncrease = 1;
    }
  else
    {
    AccelerationTime = 1;
    AccelerationIncrease = AccelDecelRate/4000;
    }
  tmpAccelerationTime = AccelerationTime;
}
```

图 2-3. 用于计算加速率或减速率时间间隔和加速增量参数的函数

图 2-4 所示为负责协调加速的状态机流程图。如您所见，用于启动电机的命令将配置运动所需的所有参数，其中包括调用 AccelTimeCompute 函数。

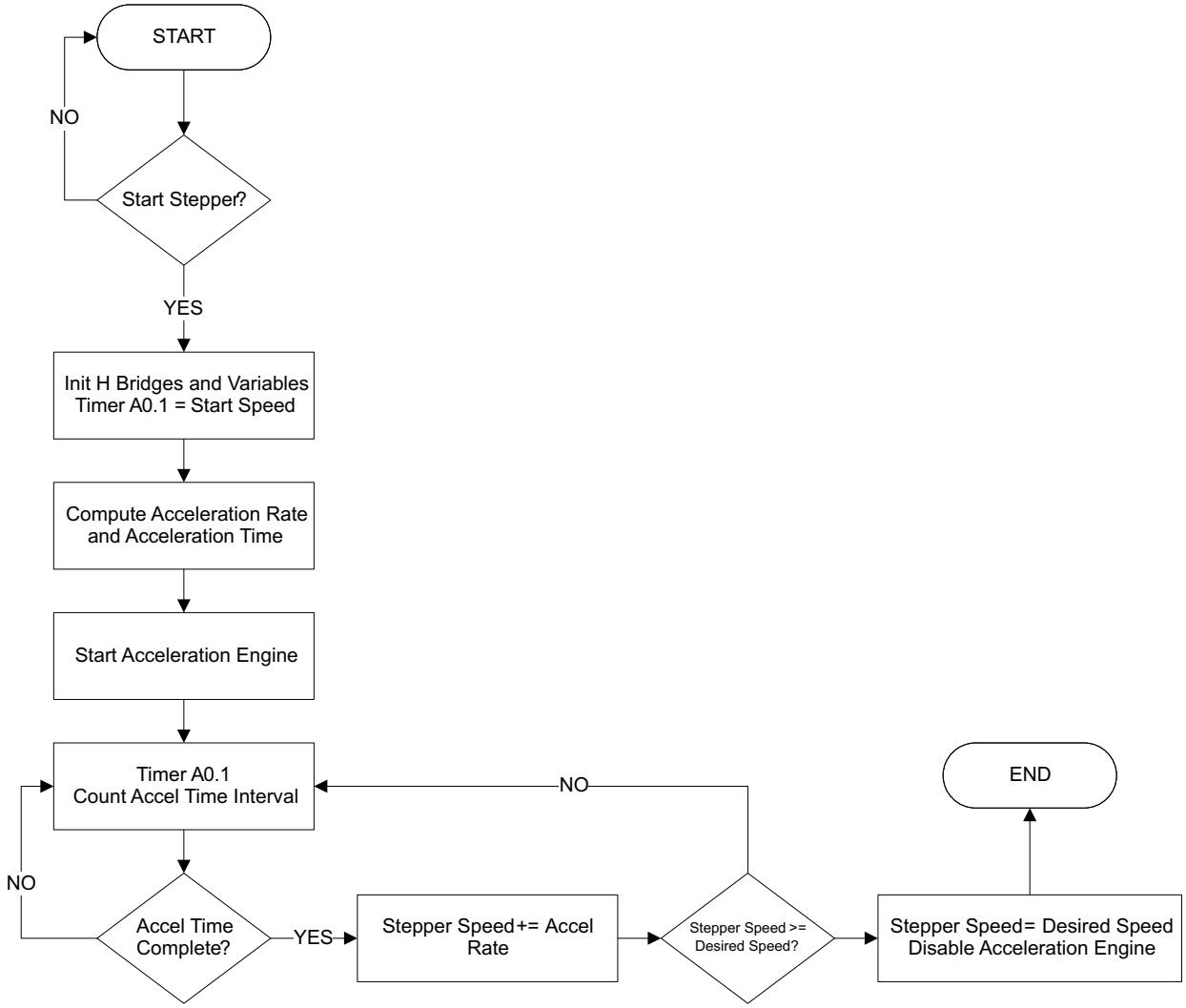


图 2-4. 步进电机转速加速流程图

实际调速发生在 AccelDecel 函数内，该函数从主执行中调用，而主执行则在计时器 A1.0 中断服务例程 (ISR) 内启用。切勿在 ISR 内调用 AccelDecel 函数，因为这会影响实时运算。而在 ISR 内，微控制器睡眠模式会被禁用，这样便可以从主循环恢复执行。

```
int main()
{

//ALL INITIALIZATION OCCURS HERE

__bis_SR_register(GIE);                 // Enable all Interrupts
while (1)
  {
  _BIS_SR(CPUOFF);                      // Enter LPM0
  AccelDecel();
  }
}
```

图 2-5. 每当 ISR 内的代码使微控制器退出睡眠模式时，主函数都会调用 AccelDecel 代码。

```
#pragma vector=TIMER1_A0_VECTOR
__interrupt void Timer1_A0(void)
{
  TimerA0Count += 1;
  if (TimerA0Count == ATBCount)
    {
    TimerA0Count = 0;
    if (tmpAccelerationTime == 0)
      {
      tmpAccelerationTime = AccelerationTime;
       _BIC_SR_IRQ(CPUOFF);                  // Clear LPM0 - jumps to AccelDecel();
      }
    tmpAccelerationTime -= 1;
    }
}
```

图 2-6. 执行加速周期后，计时器 A1.0 ISR 会禁用睡眠模式

```
void AccelDecel(void)
{
  switch (AccelerationState)
 {
 case (NOACC):
  break;
 case (ACCEL):
  ActualStepperSpeed += AccelerationIncrease;
  if (ActualStepperSpeed >= DesiredStepperSpeed)
   {
    ActualStepperSpeed = DesiredStepperSpeed;
    AccelerationState = NOACC;
    TA1CCTL0 &= ~CCIE;       //DISABLE 250 us coordinator interrupt on TA1.0
   }
  SpeedCompute(ActualStepperSpeed);

  break;
 case (DECEL):
  ActualStepperSpeed -= AccelerationIncrease;
  if (ActualStepperSpeed <= DesiredStepperSpeed)
   {
    ActualStepperSpeed = DesiredStepperSpeed;
    AccelerationState = NOACC;
    TA1CCTL0 &= ~CCIE;       //DISABLE 250 us coordinator interrupt on TA1.0
   }
  SpeedCompute(ActualStepperSpeed);

  break;
 case (STOP):
  ActualStepperSpeed -= AccelerationIncrease;
  if (ActualStepperSpeed <= StartStepperSpeed)
   {
    ActualStepperSpeed = StartStepperSpeed;
    AccelerationState = NOACC;
    TA1CCTL0 &= ~CCIE;       //DISABLE 250 us coordinator interrupt on TA1.0
    ExecutedSteps = NumberOfSteps - 1;
   }
  SpeedCompute(ActualStepperSpeed);
  break;
 }
}
```

图 2-7. AccelDecel 函数是负责根据已编程加速或减速曲线调整步进电机转速的状态机代码

2.3 步进电机转速

如前文所述，步进电机转速由计时器 A1.0 和 A1.2 进行精细控制。16 位计数器本身无法重现此拓扑相关应用可能采用的所有可能转速，因此务必要理解该计时器的计时方式。例如，如果计时器 A1 在 16MHz 频率下进行计时，那么可轻松获得较快的步进速率，但却无法生成较慢的速率。

因此，必须通过细分来调制时钟速度。幸运的是，计时器 A 具有一系列分频系数，能够将计时器时钟速度从 16MHz 分频到低至 2MHz。

我们可通过如下公式来计算重现特定步进速率所需的时钟周期数：

$$\text{时钟周期数} = \text{计时器时钟频率}/\text{步进速率 Hz} \quad (1)$$

通过对根据上述公式计算出的结果进行快速分析，我们发现要在每个时钟周期数进行步进速率隔离。在给定的计时器时钟频率条件下，如果得到的时钟周期数量大于 65535，那么便需要另外的分频因子来减少时钟计数，从而避免计时器寄存器发生饱和。

SpeedCompute 函数负责将以 Hz 为单位的步进速率换算为时钟计数。根据具体的速度，计时器 A1 的时钟源会继续进行分频，从而减慢其增长速率。如果所请求的步进速率小于 31，那么该函数会返回错误，这时不会生成任何速率。

尽管可能需要获取小于 31 个全步进的步进速率，但通过采用更快的转速并将 DRV8824/25 器件配置为以 32 度微步进角运行也可实现这一目的。例如，如果已编程的 DesiredSpeed 为 32SPS，并且将 DRV8824/25 配置为以 32 度微步进角运行，那么步进电机将每秒选择 1 个全步进。

```
bool SpeedCompute(unsigned int MotorSpeedInHz)
{
   if (MotorSpeedInHz < 31 )
   {
     return false;
   }
   else if (MotorSpeedInHz < 61 )
   {
     ClockConfigure(8);
     StepperSpeedTMR = 2000000 / MotorSpeedInHz;
     return true;
   }
   else if (MotorSpeedInHz < 123)
   {
     ClockConfigure(4);
     StepperSpeedTMR = 4000000 / MotorSpeedInHz;
     return true;
   }
   else if (MotorSpeedInHz < 245)
   {
     ClockConfigure(2);
     StepperSpeedTMR = 8000000 / MotorSpeedInHz;
     return true;
   }
   else
   {
     ClockConfigure(1);
     StepperSpeedTMR = 16000000 / MotorSpeedInHz;
     return true;
   }
}
```

图 2-8. SpeedCompute 会将以 Hz 为单位的步进速率换算为计时器 A1.1 的时钟计数可用来生成精确的计时信息

```
void ClockConfigure(char Divider)
{
  int tempTA0CTL;
  tempTA0CTL = TA0CTL;
  switch (Divider)
  {
  case 1:
  tempTA0CTL &= ~( BIT7 + BIT6);
  break;

  case 2:
  tempTA0CTL &= ~( BIT7 + BIT6);
  tempTA0CTL |= TA0_ID_DIV2;
  break;

  case 4:
  tempTA0CTL &= ~( BIT7 + BIT6);
  tempTA0CTL |= TA0_ID_DIV4;
  break;

  case 8:
  tempTA0CTL &= ~( BIT7 + BIT6);
  tempTA0CTL |= TA0_ID_DIV8;
  break;
  }
  TA0CTL = tempTA0CTL;
}
```

图 2-9. 在 SpeedCompute 函数中调用 ClockConfigure 函数来修改会影响计时器 A1 时钟速度的分频器

可调用 SpeedCompute 和 ClockConfigure 函数来设置 StartSpeed，并在每次点击加速时设置新的步进电机转速，直到达到 DesiredSpeed。在达到编程的 DesiredSpeed 目标后，步进电机会保持所述转速，直到发出减速曲线命令。

2.4 电机减速

电机的减速曲线概念与加速曲线概念几乎完全相同，不同的是这不是增加步进电机转速，而是使其减速。

可能使用减速曲线的实例有以下两种：在 SpeedUpdate 命令期间或在 StepperStop 命令期间。在此方案中，步进电机的减速曲线会在达到启动速度时或执行已编程的步进数后结束。

另外，之所以选择将减速率保存在不同的存储位置，就是为了可以采用非对称的加速/减速曲线，例如图 2-1 所示曲线。

SpeedCompute 和 ClockConfigure 的使用方式与运动控制曲线的加速部分中的相同。不过，状态机，也即 AccelDecel 函数，配置为执行代码的 DECEL 部分。

2.5 速度调整

在运行时期间，可根据应用需要调整转速并进一步加速或减速。在 SpeedUpdate 命令发出时，固件会进行检查以确认新的 DesiredSpeed 是大于还是小于 ActualStepperSpeed，然后根据此操作的结果相应地配置加减速引擎。

2.6 位置控制：步进数

步进电机的闭环功能不限于准确地控制速度。控制器统计的步进数实际上就是生成的步进数，因此会持续更新步进位置信息。每次执行一个步进时，StepPosition 变量都会根据步进旋转方向进行更新。例如，如果电机沿顺时针方向转动 (DIR = HI)，StepPosition 变量的值会递增，而如果电机沿逆时针方向转动 (DIR = LO)，则 StepPosition 变量的值会递减。

NOTE

电机是沿顺时针还是逆时针方向转动直接取决于电机的接线方式和电机本身。

我们可实时读取 StepPosition 变量的值，从而及时获取电机在任何时刻所处位置的信息。

不过，在本应用手册中，控制器始终采用位置控制模式运行。NumberOfSteps 变量经编程，可保存将要执行的所有步进数。这是一个 32 位变量，因此步进总数可能是个非常大的数值，而这时电机实际采用自由运行模式运行。不过，如果 NumberOfSteps 计数较小，那么在 NumbeOfSteps 总数执行完毕后，电机将会停止。

上述机制是通过统计已执行的步进数并将这个数值与 NumberOfSteps 命令进行比较来实现的。这一过程由计时器 A0.2 ISR 负责，如图 2-10 中所示。

请注意，此 ISR 负责上述的几项任务，这些任务对于生成准确的步进信息而言至关重要。这些操作如下：

1. 通过配置何时执行 STEP 清零操作来生成 STEP 脉冲。计时器 A0.0 负责将 STEP 信号恢复到 LO。
2. 根据当前步进速率 StepperSpeedTMR 配置计时器 A0.2 来生成下一个步进，其中 StepperSpeedTMR 是具有当前速率 (Hz) 的等效计时器。
3. 确定当前生成的步进是不是要执行的最后一个步进。当 ExecutedSteps 等于 NumberOfSteps 时，便会发生此操作。
4. 如果当前生成的步进等于 StepsToStop 的值，便会发出命令让引擎开启减速曲线。
5. 根据电机转动方向更新 StepPosition。

```
#pragma vector=TIMER0_A1_VECTOR
__interrupt void Timer0_A1(void)
{

  switch (TA0IV)
    {
    case TA0CCR1_CCIFG_SET:
    break;
    case TA0CCR2_CCIFG_SET:

    TA0CCR0 = TA0CCR2 + StepPulseWidth;              //2 us at 16 MHz
    TA0CCR2 += StepperSpeedTMR;

    ExecutedSteps += 1;

    if (ExecutedSteps == NumberOfSteps)
      {
      TA0CCTL2 &= ~(CCIE + BIT5 + BIT6 + BIT7);      //Disable Pulse Generation
      }
    else if (ExecutedSteps == StepsToStop)
      {
      AccelTimeCompute(DecelerationRate);
      AccelerationState = STOP;
      TA1CCTL0 |= CCIE;                        //ENABLE 250 us coordinator interrupt on TA1.0
      }

    if (P3IN && DIR)
      {
      StepPosition += 1;
      }
    else
      {
      StepPosition -=1;
      }
    ReadTable[0] = (StepPosition & 0xFF00) >> 8;
    ReadTable[1] = StepPosition & 0xFF;
    break;
  case TA0IFG_SET:
    break;
    }
}
```

图 2-10. 计时器 A0.2 ISR

2.7 步进电机归零

如果我们不知道步进电机的起始位置，而从其某个电气周期的同一相位位置开始运行，实际上是有问题的。对于闭环反馈为相对值或绝对值的任何电机系统来说，都是如此。因此，我们需要从已知的位置开始统计步进数，这一点非常重要。这个位置通常称为 HOME。

本应用手册中包含一个 HOME 传感器输入，该输入具有足够的灵活性，能够适应两种传感器极性（例如 HI 置位或 LO 置位）。典型的 HOME 传感器方案包含一个光学传感器并在步进电机轴处有一个标志。当该标志达到光学传感器的缝隙时，步进电机便会停止，此位置从现在开始称为 HOME。在内部，控制器会将 StepPosition 变量清零。

由于电机可以从任何指定位置启动，因此 HOME 传感器可能处于二者中的一种状态。因此，初始状态为未知。典型的归零方案会调用从 HI 到 LO 或从 LO 到 HI 的转换，具体由应用选定。

为了轻松捕获选定的转换，HOME 传感器被分配至 GPIO 引脚并具有引脚更改中断功能。由于该引脚可配置为在上升沿或下降沿触发 ISR 标志，因此我们可以在任一边沿上捕获。鉴于典型的轮询函数需要某种类型的状态机来滤除错误的转换，此硬件中断很好用，所需的代码量很小。图 2-11 展示了 PORT1 引脚更改中断矢量的 ISR。我们可以看到，此代码负责将 StepPosition 变量清零以及停止电机。

```
#pragma vector=PORT1_VECTOR
__interrupt void PORT1_Change(void)
{
  if (P1IFG && HOMEIN)
  {
    P1IE = 0;
    P1IFG = 0;
    TA0CCTL2 &= ~(CCIE + BIT5 + BIT6 + BIT7);
    StepPosition = 0;
  }
}
```

图 2-11. Port 1 引脚更改中断服务例程

3 I²C 协议与通信引擎

为了对器件参数和步进电机运动引擎曲线进行编程，这里选择了 I²C 协议。根据其设计方式，只需两个通信线路和两个地址选择线路，便可以级联最多四个控制器。

I²C 协议是典型的三字节数据包，其中第一个字节为目标地址，第二个字节为寄存器地址，而第三个字节为数据。这里提供包含 21 个可访问地址的寄存器。

Index	PARAMETER TABLE
0x00	Number Of Steps
0x01	Number Of Steps
0x02	Number Of Steps1
0x03	Number Of Steps0
0x04	StepsToStop
0x05	StepsToStop
0x06	StepsToStop1
0x07	StepsToStop0
0x08	StartSpeed1
0x09	StartSpeed0
0x0A	Accel1
0x0B	Accel0
0x0C	DesiredSpeed1
0x0D	DesiredSpeed0
0x0E	Decel1
0x0F	Decel0
0x10	GPIO CONFIG
0x11	Stepper Config
0x12	GPIO OUT
0x13	Current Duty Cycle
0x14	Start Stepper

图 3-1. 参数表

地址 0x00 至 0x0F 中驻留的变量已经在前面几个部分中讨论过，其他地址则包含参数与操作的组合。

3.1 GPIO 配置

定义了 8 个 GPIO 引脚（0 至 7）的 GPIO 方向，其中配置为 0 时表示输入，配置为 1 时表示输出。

Bit 7							Bit 0
GPIO DIR 7	GPIO DIR 6	GPIO DIR 5	GPIO DIR 4	GPIO DIR 3	GPIO DIR 2	GPIO DIR 1	GPIO DIR 0

3.2 步进电机配置

为步进电机功率级配置控制信号。这些寄存器位直接映射到功率级硬件引脚，因此更改其中任何位的状态均可立即改变功率级输入端的对应引脚。

Bit 7							Bit 0
N/A	ENABLE	MODE2	MODE1	MODE0	N/A	N/A	DIR

3.3 GPIO 输出

写入此地址即对那些已配置为输出的 GPIO 位进行配置。请注意，写入此地址就相当于写入 MSP430 PxOUT 寄存器，因此只有在对应 PxDIR 寄存器中配置为输出的位才会做出相应动作。那些配置为输入的引脚仍用作输入。

Bit 7							Bit 0
GPIO 7	GPIO 6	GPIO 5	GPIO 4	GPIO 3	GPIO 2	GPIO 1	GPIO 0

3.4 当前占空比

寄存器接受 0 至 249 之间的数字，该数值随后会成为 PWM 输出占空比。当前代码不会检查写入的值是否等于或小于 249，当数值大于 249 时，PWM 占空比为 100%。

3.5 启动步进电机

此地址与其说是一个实际的寄存器，还不如说是一项命令。多分支选择语句对数据字节进行解码，然后会执行一项操作。可能的操作包括：

OPCODE	START STEPPER
0x00	START ACCEL
0x01	START STEPPER
0x02	STOP STEPPER
0x03	CHANGE SPEED
0x04	HOME HI
0x05	HOME LO

其中：

START ACCEL 会以 StartSpeed 启动步进电机，然后缓慢加速，直到达到 DesiredSpeed。如果没有收到其他命令，电机会在 NumberOfSteps 执行完毕后停止。

START STEPPER 会以 StartSpeed 启动步进电机，这时不会发生加速。如果没有收到其他命令，电机会在 NumberOfSteps 执行完毕后停止。

STOP STEPPER 会通过减速曲线缓慢减速并停止步进电机。一旦达到 StartSpeed，或 NumberOfSteps 执行完毕，电机便会停止。

CHANGE SPEED 会根据实际速度是大于还是小于新的 DesiredSpeed，对步进电机进行加速或减速。必须在电机运行期间写入新的 DesiredSpeed，此命令才有效。

HOME HI 会以 StartSpeed 启动电机并一直运行，直到在 HOME 传感器输入端观察到转换至 HI 的事件，或者直到 NumberOfSteps 执行完毕。

HOME LO 会以 StartSpeed 启动电机并一直运行，直到在 HOME 传感器输入端观察到转换至 LO 的事件，或者直到 NumberOfSteps 执行完毕。

Start Stepper 命令将会配置 NumberOfSteps、StepsToStop、StartStepperSpeed、DesiredStepperSpeed、AccelerationRate 和 DecelerationRate 等步进引擎参数。然后，它将会对操作参数进行解码并根据收到的命令启动步进电机。USCI RX 和 TX 矢量中断例程负责 I²C 通信。

```
#pragma vector=USCIAB0TX_VECTOR //UCA_TRANSMIT on UART/SPI;
UCB_RECEIVE, UCB_TRANSMIT on I2C
__interrupt void USCI_AB0_Transmit(void)
{
  if (UCB0CTL1 & UCTR)
  {
    UCB0CTL1 &= ~UCTR;
    IFG2 &= ~UCB0TXIFG;
  }
  else
  {
  SerialBuffer[SerialPointer] = UCB0RXBUF;
  SerialPointer += 1;
  if (SerialPointer == SERIAL_BUFFER_LENGTH)
    {
    SerialPointer = 0;
    ParametersTable[ADDRESS] = PARAMETER;

    switch(ADDRESS)
      {
      case GPIO_CONFIG:
        char tempOut;
        P1DIR = PARAMETER & 0xC0;                    //Use 2 MSB's to configure the
GPIO Direction on pins P1.7 and P1.6
        P2DIR = PARAMETER & 0x3F;                    //Use 6 LSB's to configure the
GPIO direction on pins P2.0 to P2.5
      case STEPPER_CONFIG_ADDR:
        tempOut = P3OUT;
        tempOut &= ~(nENABLE + MODE0 + MODE1 + MODE2 + DIR);
        tempOut |= PARAMETER;
        P3OUT = tempOut;        break;
      case GPIO_OUT_ADDR:
        P2OUT = PARAMETER;
        tempOut = P1OUT;
        tempOut &= ~(BIT7 + BIT6);
        tempOut |= (PARAMETER & 0xC0);
        P1OUT = tempOut;
        break;
      case CURRENT_DC_ADDR:
        TA1CCR1 = PARAMETER;
        break;
```

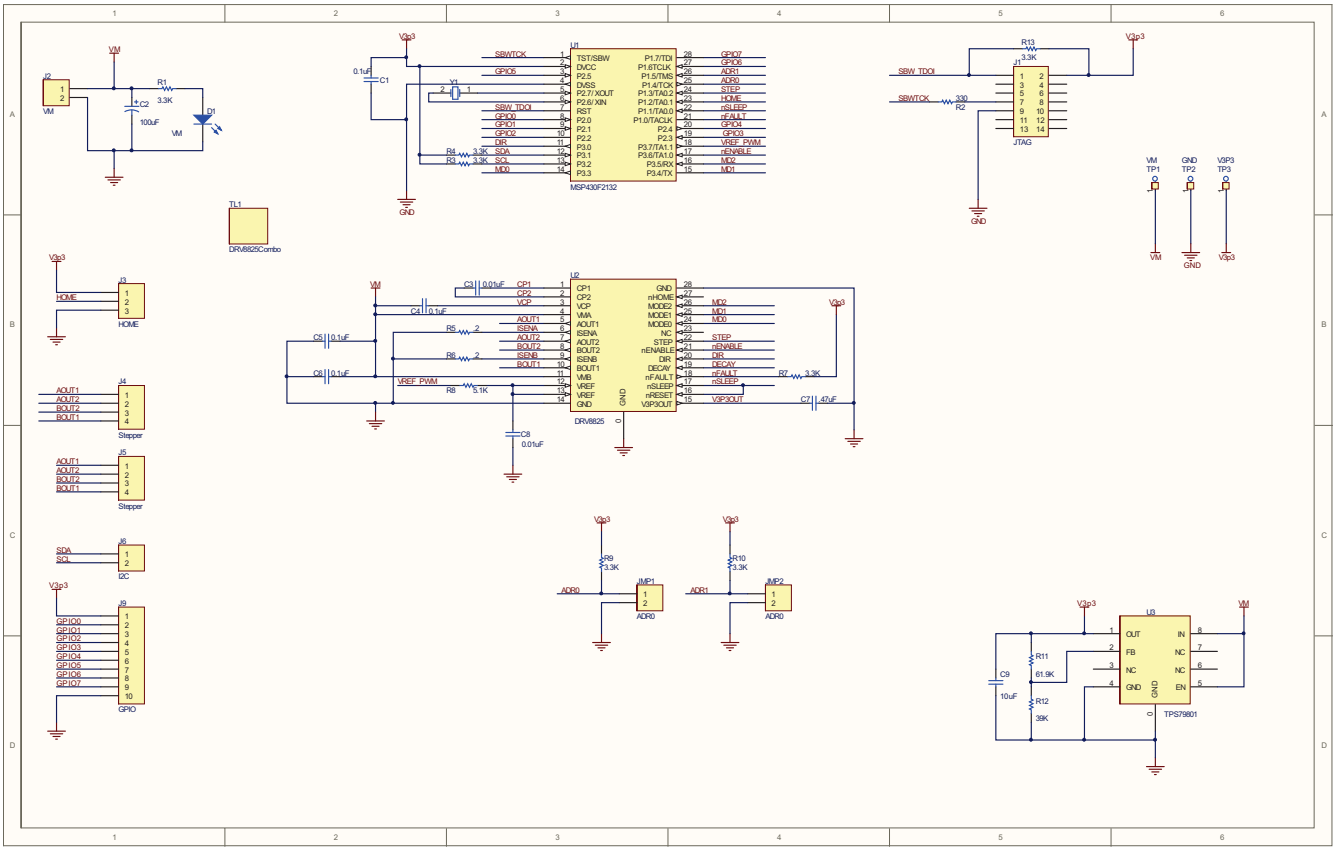
```
    case START_STEPPER_ADDR:
      NumberOfSteps = (ParametersTable[NUMBER_OF_STEPS3_ADDR] << 8) +
(ParametersTable[NUMBER_OF_STEPS2_ADDR]);
      NumberOfSteps *= 65536;
      NumberOfSteps +=  (ParametersTable[NUMBER_OF_STEPS1_ADDR] << 8) +
ParametersTable[NUMBER_OF_STEPS0_ADDR];
      ExecutedSteps = 0;

      StepsToStop = (ParametersTable[STEPS_TO_STOP3_ADDR] << 8) +
(ParametersTable[STEPS_TO_STOP2_ADDR]);
      StepsToStop *= 65536;
      StepsToStop +=  (ParametersTable[STEPS_TO_STOP1_ADDR] << 8) +
ParametersTable[STEPS_TO_STOP0_ADDR];

      StartStepperSpeed = (ParametersTable[START_SPEED1_ADDR] << 8) +
ParametersTable[START_SPEED0_ADDR];

      DesiredStepperSpeed = (ParametersTable[DESIRED_SPEED1_ADDR] << 8) +
ParametersTable[DESIRED_SPEED0_ADDR];

      AccelerationRate = (ParametersTable[ACCEL1_ADDR] << 8) +
ParametersTable[ACCEL0_ADDR];

      DecelerationRate = (ParametersTable[DECEL1_ADDR] << 8) +
ParametersTable[DECEL0_ADDR];
```

```
switch (PARAMETER)
        {
        case START_ACCEL:        // Start stepper motor at Start speed and until
reaching desired speed
          AccelerationState = ACCEL;
          ActualStepperSpeed = StartStepperSpeed;
          AccelTimeCompute(AccelerationRate);
          TA1CCTL0 |= CCIE;       //ENABLE 250 us coordinator interrupt on TA1.0
          //WDTCTL = 0x5A00 + WDTTMSEL + WDTSSEL + BIT1 + BIT0;
          break;
        case START_STEPPER:      // Start stepper motor at start speed. Run at this
rate.
          AccelerationState = NOACC;
          ActualStepperSpeed = StartStepperSpeed;
          break;
        case STOP_STEPPER:       // Stop motor from current speed, through a
deceleration profile and until reaching Start Speed. Then stop and disable stepping
engine
          AccelTimeCompute(DecelerationRate);
          AccelerationState = STOP;
          TA1CCTL0 |= CCIE;       //ENABLE 250 us coordinator interrupt on TA1.0
          break;
        case CHANGE_SPEED:       // Modify current speed up or down to Desired
Speed.
          if (DesiredStepperSpeed >= ActualStepperSpeed)
           {
           AccelTimeCompute(AccelerationRate);
           AccelerationState = ACCEL;
           TA1CCTL0 |= CCIE;       //ENABLE 250 us coordinator interrupt on TA1.0
           }
          else
           {
           AccelTimeCompute(DecelerationRate);
           AccelerationState = DECEL;
           TA1CCTL0 |= CCIE;       //ENABLE 250 us coordinator interrupt on TA1.0
           //WDTCTL = 0x5A00 + WDTTMSEL + WDTSSEL + BIT1 + BIT0;
           }
          break;
```

```
          case HOME_HI:           // Move Stepper at Start Speed until the Home
Sensor becomes HI. Go through LO if already at HI
            AccelerationState = NOACC;
            ActualStepperSpeed = StartStepperSpeed;
            P1IFG &= ~HOMEIN;
            P1IES |= HOMEIN;
            P1IE |= HOMEIN;
            break;
          case HOME_LO:           // Move Stepper at Start Speed until the Home
Sensor becomes LO. Go through HI if already at LO
            AccelerationState = NOACC;
            ActualStepperSpeed = StartStepperSpeed;
            P1IFG &= ~HOMEIN;
            P1IES &= ~HOMEIN;
            P1IE |= HOMEIN;
            break;
          }

        if (SpeedCompute(ActualStepperSpeed))
          {
          P3OUT &= ~(nENABLE);
          P1OUT |= NSLEEP;

          TA0CCR2 += StepperSpeedTMR;
          TA0CCR0 = TACCR2 + StepPulseWidth;          //2 us at 16 MHz
          TA0CCTL2 &= ~CCIFG;
          TA0CCTL2 |= (CCIE + TA0_OUTMOD2_CONF);

          }
        break;

      }
    }
  }
}
```

4 应用原理图

下页展示了 MSP430 和 DRV8811/18/24/25 的组合电路板原理图。该原理图中的步进电机驱动器可替换为 DRV8434A 或 DRV8434/36/24/25/26/28。这些器件均无需外部电流检测电阻。



5 修订历史记录

注：以前版本的页码可能与当前版本的页码不同

将修订版本 A（2014 年 1 月）更改为修订版本 B（2021 年 1 月）

通篇将 DRV8811/18/24/25 替换为 DRV8434/28/26/24

Changes from Revision A (January 2014) to Revision B (January 2021)	Page
• 添加了“集成式步进电机驱动器”表	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