How to Use iMOTION™ Script Language

如何使用 iMOTION™ Script 脚本语言

关于本文

范围和目的

此应用笔记提供了如何在运动控制引擎（MCE）上使用 iMOTION™ 脚本语言（script language）的指南。它包括如何实现低通滤波器（LPF）、2 档速度选择的接口、基于母线电压的目标速度定制、母线欠压保护，以及动态电机电流限制。

目标读者

本文旨在帮助那些希望了解如何使用 iMOTION™ 脚本语言的用户，实现系统启动行为的定制，目标速度配置文件的定制，以及系统故障处理的定制。

目录

Table of Contents

1 脚本语言概述

1.1 简介

iMOTION™ 运动控制引擎 (Motion Control Engine)的最新软件版本包括一个脚本引擎，为用户提供了在不影响电机和 PFC 控制算法的前提下，定制系统级功能的可能性。脚本引擎是一个轻量级的虚拟机，它可以读写所有的电机控制和 PFC 的寄存器变量，允许用户定制电机控制和 PFC 控制没有使用的模拟和数字硬件资源，用于客户功能拓展。典型的脚本语言应用包括系统启动过程定制，速度曲线规划，参数配置以及特殊故障处理。

- CPU 的资源优先给电机控制和 PFC 控制算法使用，脚本引擎利用空闲的 CPU 资源来执行脚本语言，故脚本语言执行的优先级要低于电机控制和 PFC 控制算法。强烈建议检查实际运行中的 CPU 负荷，以保证 CPU 资源的合理分配。
- 脚本引擎支持两个独立并行的任务，任务 0（Task0）和任务 1（Task1），用户的脚本代码在 Task0 和 Task1 中循环运行，间隔时间可配置。Task0 和 Task1 最短的执行周期分别时 1 ms 和 10 ms，用户可以在脚本代码里给每个任务配置更长的执行周期，但必须分别是 1 ms 和 10 ms 的整数倍。Task0 的优先级要高于 Task1。
- iMOTION™ 脚本语言是一种解释型的语言，它需要先被编译成一种伪代码（bytecode），然后在 MCE 的虚拟机中来直接执行。

1.2 脚本开发流程

典型的脚本代码开发流程从 MCEWizard（或者其他文档编辑器）开始，编写代码并保存为后缀为.mcs 的文件。在 MCEWizard 中配置可用的模数转换器 ADC 或者通用的输入输出管脚（GPIO），再通过 MCEWizard 编译脚本代码，生成后缀名为‘.ldf’的脚本目标文件。ldf 文件包含的信息有 Task0 和 Task1 的脚本指令总数和全局变量列表。最后通过 MCEDesigner[3]下载 ldf 文件到目标 MCE 中，MCEDesigner[3]还可以监控脚本代码中全局变量的实际状态。关于脚本语言及其开发的更多细节信息可以参考附录中的文档 2。

2 脚本应用实例

2.1 两档速度选择接口

2.1.1 对速度选择接口的要求

电机应用中多档速度选择是一个很常见的功能，例如吹风机。此例详述了一个在 IMC101T[1]上利用一个 ADC 管脚实现两档速度选择接口的脚本实现和具体要求。

硬件电路将机械开关的速度选择位置转换为一个介于 0 V 到 5 V 的模拟电压值。当模拟电压值在 0 V 到 1 V 时，要求电机在停机状态，1 V 到 2 V 时工作在低速状态，高于 2 V 时工作在高速状态。为了消除电压在阈值边界附近可能引起的振荡，引入了一个 0.2 V 的滞环。

此例通过一个模拟口检测电压来判断速度档位。

Figure 1 描绘了速度档位和模拟电压之间的关系。蓝线起始从停机状态开始，当模拟电压高于 1 V 时候，状态切换至低速档位。当模拟电压超过 2V 时候，速度由低速档位切换至高速档位。红线起始从高速档位开始，当模拟电压低于 1.8 V，速度从高速档位切换至低速，如果模拟电压低于 0.8 V，则切换至停机。

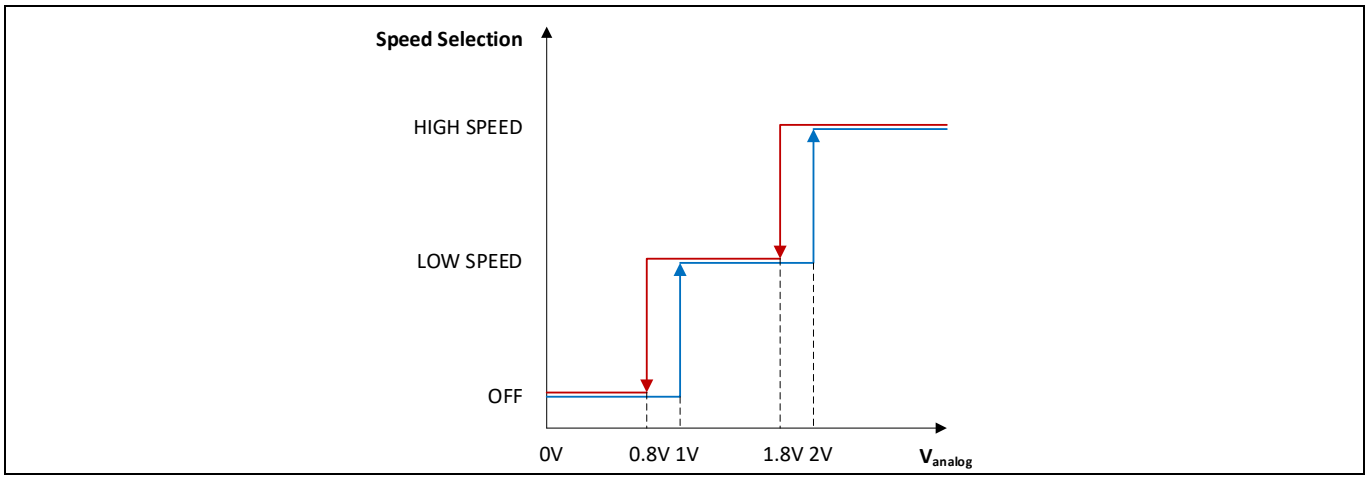


Figure 1 速度档位与模拟电压的关系

2.1.2 用于速度选择的模拟输入管脚

在 2.1.1 节里描述的对速度选择的要求，可以通过脚本引擎使能一个模拟输入管脚方便地实现。在此例中，AIN0 脚被用来作为速度选择电路的接口。使能 AIN0 以后，此 ADC 每 10 ms 采样一次[2]，结果可以通过读取变量 `ADC_Result0` 得到。

已知 ADC 的分辨率为 12 位，AIN0 端口电压和 ADC 转换结果之间的关系可以由以下公式得出，$\text{ADC\_Result0} = INT(V_{AIN0} \cdot \frac{2^{12}-1}{V_{ref}} + 0.5)$,其中V_{ref}为 ADC 的参考电压。如果选择V_{ref}为 5 V，高速和低速指令的电压阈值可以按上述公式计算，结果汇总如下表 Table 1。

Table 1 速度档位与电压阈值

Variable Name	Voltage Threshold	ADC Conversion Result
VLSStart	1 V	819 (ADC Counts)
VLSStop	0.8 V	655 (ADC Counts)
VHSStart	2 V	1638 (ADC Counts)
VHSStop	1.8 V	1474 (ADC Counts)

2.1.3 速度选择状态机

设计一个状态机来选择速度档位，如 Figure 2 所示。用变量 SpeedMode 来表示 3 种可能的速度档位。分别是，停机 Speed_Mode_OFF (SpeedMode = 0)，低速档 Speed_Mode_LOW_SPEED (SpeedMode = 1)，高速档 Speed_Mode_HIGH_SPEED (SpeedMode = 2)。电机从停机状态 Speed_Mode_OFF 开始，这时候目标速度设置为 0，电机处于停机状态。当 VSP 脚的电压高于 VLSStart，电机状态切换至低速档 Speed_Mode_LOW_SPEED。如果电压低于 VLSStop，电机状态切换至停机状态 Speed_Mode_OFF。当 VSP 引脚的电压高于 VHSStart，电机状态切换至高速档 Speed_Mode_HIGH_SPEED，如果电压低于 VHSStop，电机状态切换至低速档 Speed_Mode_LOW_SPEED。当电机工作在高速状态或者低速低速状态时，目标速度会被设定为预先定义好的高速速度值 HighSpeedValue 或者低速速度值 LowSpeedValue 运行。启动/停止电机的操作可以通过电机寄存器 Command 置 1 或者清零来实现。由于脚本引擎可以直接操作电机寄存器，所以 Command 可以被脚本代码直接使用而不需要声明。

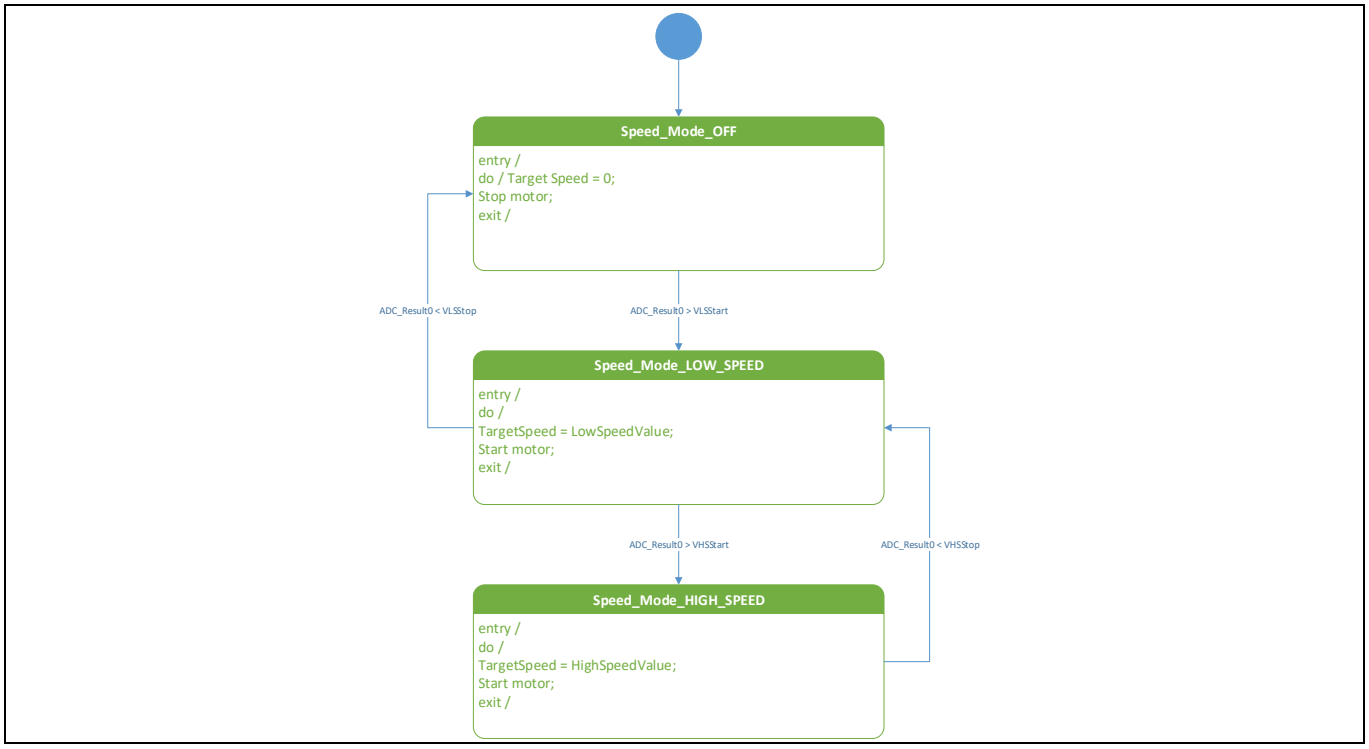


Figure 2 速度选择状态机

2.1.4 速度选择接口的脚本代码实现

Code Listing 1 列出了在 Task 1 里实现 2 档速度选择的源代码。由于速度选择开关不会频繁改变，故建议将 Task1 的执行周期设置为 50 ms。Code Listing 2 中是编译生成的脚本目标文件的一部分，在 009 行显示了 Task1 的代码行数为 17 行。所以每步执行的代码行数需要设置成高于 17，这样每次执行周期才可以

执行完全部代码。在此例中，Task1 的 SCRIPT_TASK1_EXECUTION_PERIOD 设为 5，SCRIPT_TASK1_EXECUTION_STEP 设为 20，从而可以满足目标要求。

此例也可以在 Task0 中实现，这时候 Task0 的执行周期 SCRIPT_TASK0_EXECUTION_PERIOD 需要设置为 50 才为 50ms。

Code Listing 1　　速度选择接口的脚本代码

```
001        /*************************************************************/
002        /*Script user version value, should be 255.255*/
003        #SET SCRIPT_USER_VERSION (1.00)
004        #SET SCRIPT_TASK1_EXECUTION_PERIOD (5)
005        /*Defines number of lines to be executed every 10ms in Task1*/
006        #SET SCRIPT_TASK1_EXECUTION_STEP (20)
007        /*************************************************************/
008        /*Task1 init function*/
009        Script_Task1_init()
010        {
011          int SpeedMode;
012          int VLSStart;
013          int VLSStop;
014          int VHSStart;
015          int VHSStop;
016          int LowSpeedValue;
017          int HighSpeedValue;
018          VLSStart = 819;   // 1V => 819 counts
019          VLSStop = 655;    // 0.8V => 655 counts
020          VHSStart = 1638;  // 2V => 1638 counts
021          VHSStop = 1474;   // 1.8V => 1474 counts
022          LowSpeedValue = 5000;
023          HighSpeedValue = 10000;
024        }
025        /*Task1 script function*/
026        Script_Task1()
027        {
028          if (SpeedMode == 0)      // Speed selection is in OFF state.
029          {
030            TargetSpeed = 0;
031            Command = 0;    // Stop the motor.
032            if (ADC_Result0 > VLSStart)
033            {
034               SpeedMode = 1;      // Shift to LOW_SPEED state.
035            }
036          }
037          if (SpeedMode == 1)      // Speed selection is in LOW_SPEED
   state.
038          {
039            if (ADC_Result0 > VHSStart)
040            {
041               SpeedMode = 2;      // Shift to HIGH_SPEED state.
042            }
043            else
044            {
045              if (ADC_Result0 < VLSStop)
046              {
```

Code Listing 1　　速度选择接口的脚本代码

```
047                                SpeedMode = 0;    // Shift to OFF state.
048              }
049              else   //Stay in LOW_SPEED state.
050              {
051                TargetSpeed = LowSpeedValue;   // Update TargetSpeed.
052                Command = 1;      // Start motor.
053              }
054            }
055          }
056          if(SpeedMode == 2)      // Speed selection is in HIGH_SPEED
   state.
057          {
058            if(ADC_Result0 < VHSStop)
059            {
060              SpeedMode = 1;      // Shift to LOW_SPEED state.
061            }
062            else     // Stay in HIGH_SPEED state.
063            {
064              TargetSpeed = HighSpeedValue;    // Update TargetSpeed.
065              Command = 1;
066            }
067          }
068        }
```

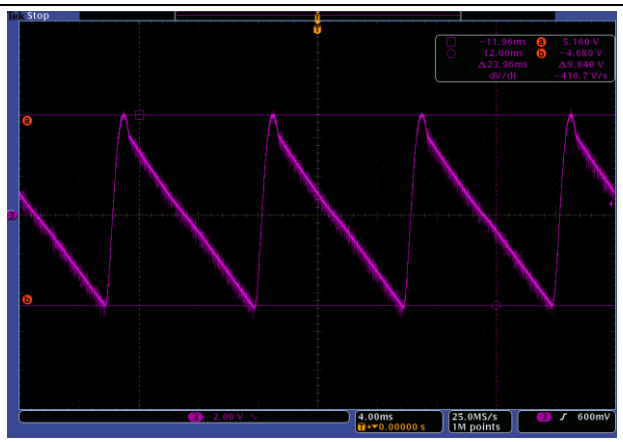
Code Listing 2　　速度选择接口脚本代码编译生成的目标文件中的部分内容

```
001        %-------------------------------------------------------------
002        % Script Object File
003        %-------------------------------------------------------------
004        % SCRIPT_USER_VERSION              : 001.000
005        % DATE & TIME                      : 22.08.2018  22:23:36
006        % SIZE                             : 231 Bytes
007        % Total Number of Lines            : 69
008        % Task0 - Number of Instructions   : 0
009        % Task1 - Number of Instructions   : 17
010        %-------------------------------------------------------------
```

2.2　　母线电压低通滤波器

2.2.1　　母线电压纹波

通常，交流输入前端包括整流桥，接着是大电解电容，将交流电压转换为直流电压。母线电压幅值接近输入交流电压的峰值，而母线电压采样取的就是母线电容两端的电压值。当电机运行时，母线电压包含高频开关纹波和两倍于电源频率的电容充放电的低频纹波。Figure 3 是一个实际运行中的母线电压波形交流部分的波形图，测试平台为 IMC101T 驱动永磁同步电机，转速 19400 RPM，输入电压 125 VAC/50Hz。可以看到，母线电压纹波峰峰值为 9.84 V。



CH3: DC Bus Voltage (AC coupling); V_{pp_CH3}: 9.84 V

Figure 3　　母线电压波形

2.2.2　　母线电压采样

选择 ADC 参考电压为 5 V($V_{ref} = 5\ V$)，则电压采样范围是 0 V 到 5 V。如 Figure 4 所示，母线电压通过由 R1 和 R2 组成的分压器连接到 ADC 引脚，$R_1 = 2\ M\Omega$，$R_2 = 13.3\ K\Omega$，母线电压采样增益 $G_{DCBus\_sensing} = \frac{R2}{R1+R2} = \frac{13.3K\Omega}{2M\Omega+13.3K\Omega} = 0.00661$, VDC 引脚能检测到的母线电压最高至 757 V。

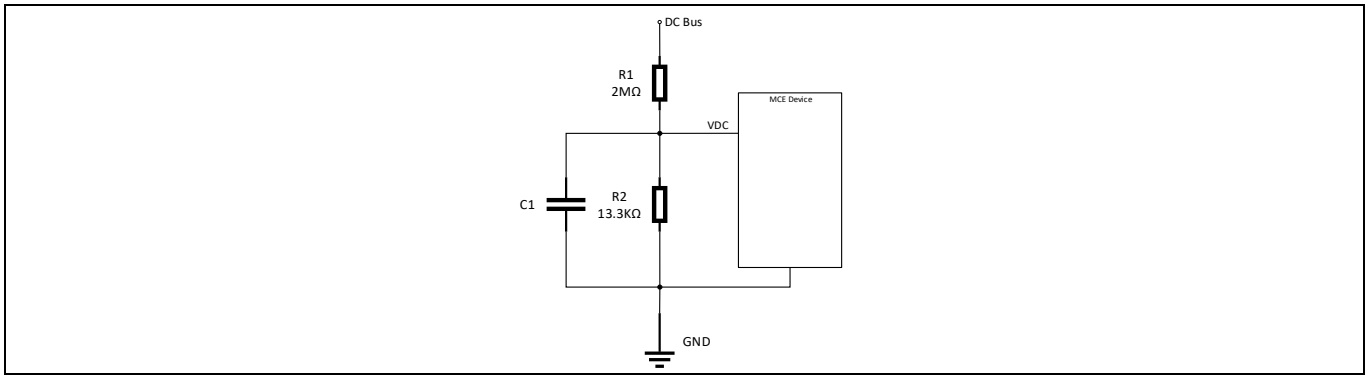


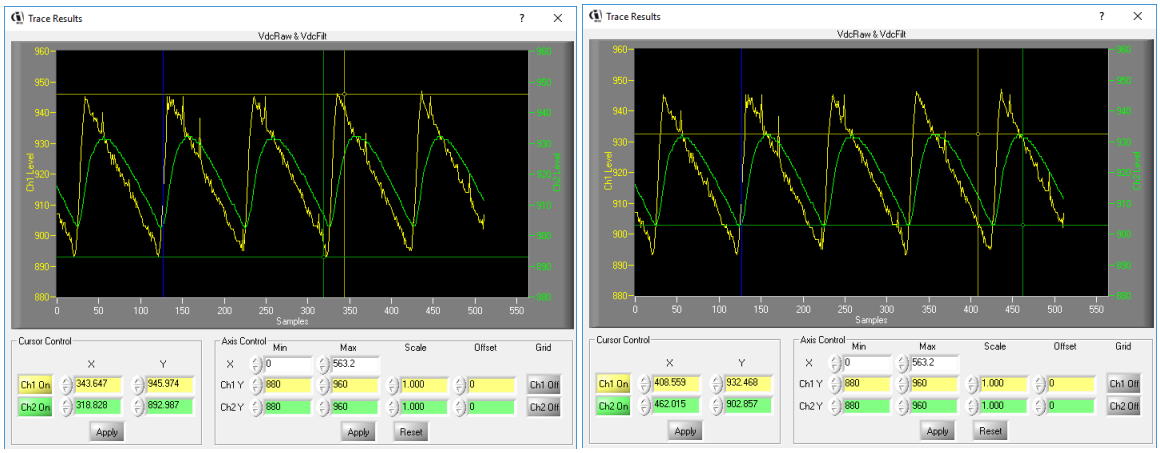
Figure 4　　母线电压检测电路

母线电压在每一个电机 PWM 周期都会采样，并以电机参数 `VdcRaw` 表示，单位为 ADC 的计数值[2]。一个典型的 PWM 周期为 50 µs，`VdcRaw` 经过一个内部数字滤波，结果存储在寄存器 VdcFilt 中[2]。

该处理器的 ADC 的分辨率为 12 位。从母线电压到 ADC 采样的转换公式可以表示为$V_{DCBus_ADC} = INT(V_{DCBus} \cdot G_{DCBus_sensing} \cdot \frac{2^{12}-1}{V_{ref}} + 0.5)$，其中 INT 表示所得数取整。

由于脚本引擎可以直接读取电机控制寄存器，因此寄存器 `VdcRaw` 和 `VdcFilt` 可以直接读取而不需要在脚本代码中进行申明。Figure 5 显示了在 MCEDesigner 窗口中，和 Figure 3 相同条件下，`VdcRaw` 和 `VdcFilt` 的波形[3]。9.84 V 电压纹波通过公式转换为 ADC 计数值应为 53 个，通过 Figure 5 可以看出 `VdcRaw` 结果和公式计算是一致的。可以发现，虽然通过内部低通滤波器，`VdcFilt` 的纹波已经比 `VdcRaw` 衰减了，但是依然有 30 个 ADC 计数值。

为了在系统处于稳定状态时获得母线电压的平均值，需要在脚本代码中增加一个低通滤波器将 `VdcFilt` 的纹波进一步衰减至不超过一个 ADC 计数值。



Ch1: VdcRaw; Ch2: VdcFilt; F_{PWM}: 20kHz; Sample at PWM frequency divided by 2; Trigger Level: 920; V_{in}: 125VAC / 50Hz; V_{pp_CH1}: 53 ADC counts; V_{pp_CH2}: 30 ADC counts

Figure 5　　VdcRaw & VdcFilt 波形截图

2.2.3　　LPF 的设计与实现

考虑到脚本处理单元的资源有限，为此应用选择了一个 1 阶无限冲激响应（IIR）低通滤波器算法。其差分方程如下所示，$y(n) = \alpha \cdot y(n-1) + (1-\alpha) \cdot x(n)$。其中$\alpha$是一个 0-1 之间的系数，$x(n)$和$y(n)$是当前采样时刻的输入值和输出值，$y(n-1)$是上一个采样时刻的输出值。这个滤波器的离散域（z）传递函数可以表示为$H_{LPF}(z) = \frac{1-\alpha}{1-\alpha \cdot z^{-1}}$。假设采样周期为$T_s$，用$z = e^{s \cdot T_s}$代入上式，我们可以得到该滤波器的连续（s）域传递函数为$H_{LPF}(s) = \frac{1-\alpha}{1-\alpha \cdot e^{-sT_s}}$。

VdcFilt 纹波的频率主要为两倍交流电压的频率，一般是 100Hz 或者 120Hz。基于奈奎斯特（Nyquist）采样原理，采样频率至少要大于两倍实际波形的最大频率才可以真实的还原实际波形。Task1 可以支持最低 10 ms 执行周期，对于此例是不够的，故我们需要选用最低执行周期为 1 ms 的 Task0 来实现此低通滤波器。采样频率越高，可以设定的衰减滤波频率越高，因此我们设置$T_s = 1\ ms$，低通滤波器可以衰减的最高频率为 500Hz。

我们用 100Hz 纹波为例，衰减 30 倍对应$20 \cdot log_{10}\left(\frac{1}{30}\right) = -29.5\ dB$，为了实现 100 Hz 衰减$-29.5\ dB$，按照 $H_{LPF}(s)$幅值的计算公式，计算得出α 为 0.98。由于脚本引擎只支持 32 位整型变量，0.98 必须要表示为分数形式。我们定义$\alpha = \frac{\alpha_{NUM}}{\alpha_{DEN}}$，低通滤波器的实现可以用下面 Code Listing 3 里的伪代码来表示。

Code Listing 3　　低通滤波器伪代码

```
011        Y1(n)=Y1(n-1)+(αDEN-αNUM)*(X(n)-Y(n-1));
012        Y(n)=Y1(n)/αDEN;
```

推荐将 α_{DEN}设定成为 2 的 n 次方，这样除法运算可以通过移位来实现。比如我们选择$\alpha_{DEN} = 64$，误差最小的分子$\alpha_{NUM} = 63$，得近似值$\alpha = 0.984$，误差 0.5%。除 64 可以用右移六位来实现。Code Listing 4 列出了实现低通滤波器（LPF）的脚本代码。

Code Listing 4　　低通数字滤波器的脚本代码

```
001        /*************************************************************/
002        /*Script execution time for Task0 in ms, maximum value 65535*/
003        #SET SCRIPT_TASK0_EXECUTION_PERIOD (1)
004        /*Defines number of lines to be executed every 1ms in Task0*/
005        #SET SCRIPT_TASK0_EXECUTION_STEP (2)
```

```
006         /*****************************************************************/
007         /* Global variable definition */
008         int VDCBusLPF;
009         /*****************************************************************/
010         /*Task0 init function*/
011         Script_Task0_init()
012         {
013                 /*Initialize global variable*/
014                 VDCBusLPF = 0;
015                 /* local variable definition */
016                 int VDCBusMultiplyDEN;
017                 /*Initialize local variable*/
018                 VDCBusMultiplyDEN = 0;
019         }
020
021         /*Task0 script function*/
022         Script_Task0()
023         {
024                 // Vdcbus filtering
025                 VDCBusMultiplyDEN = VDCBusMultiplyDEN + (VdcFilt -
    VDCBusLPF);
026                 VDCBusLPF = VDCBusMultiplyDEN >> 6;
}
```

如代码中所示，只有 025 和 026 两行代码进行低通滤波器计算，所以第五行 `SCRIPT_TASK0_EXECUTION_STEP` 设为 2，这样 Task0 就能在 1 ms 内执行完。同时，`VdcFilt` 1 kHz 的采样频率是可以保证的。每个 Task 的有效代码行数可以在相应的目标文件.ldf 中查到。

此例中，滤波器的时间常数$\tau = -\frac{T_s}{Ln(\alpha)} = -\frac{1ms}{Ln(0.984)} = 63\ ms$，截止频率$f_c = \frac{1}{2\pi\tau} = \frac{1}{2\pi \cdot 63ms} = 2.51\ Hz$，在 100Hz 频率的增益为-31.9 dB。通过 Matlab 软件，我们可以做出波特图和阶跃响应曲线，详见 Figure 6 和 Figure 7 **Error! Reference source not found.**。

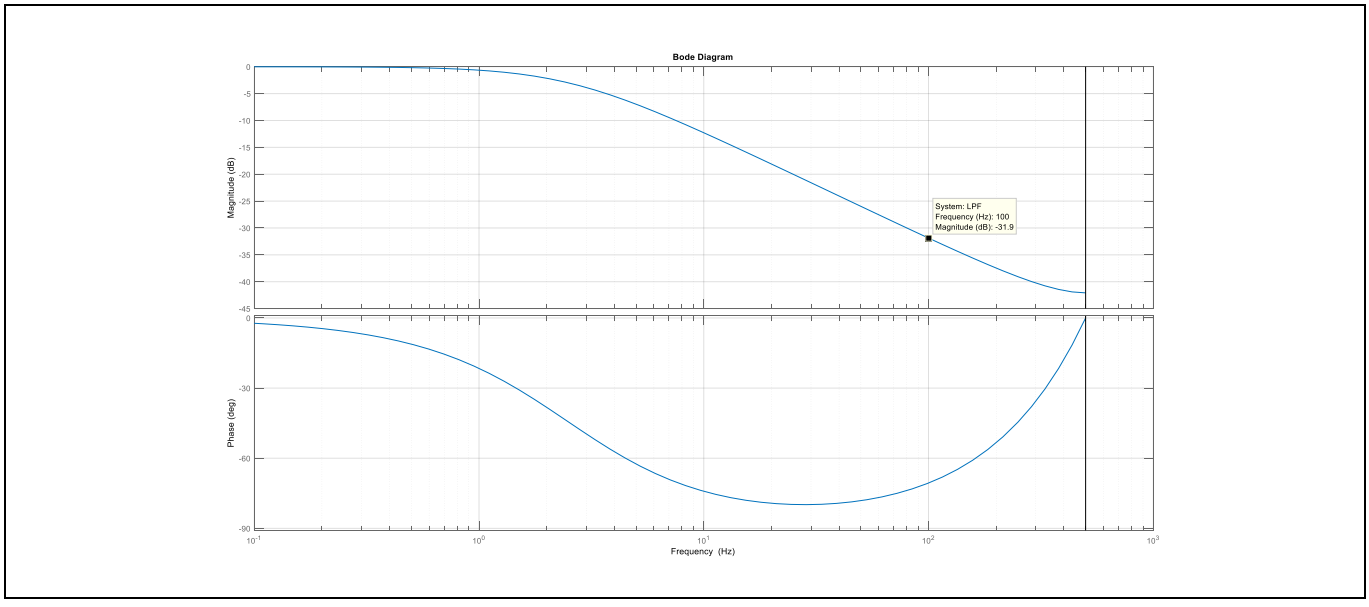


Figure 6　　一阶无限冲激响应滤波器的频率响应曲线

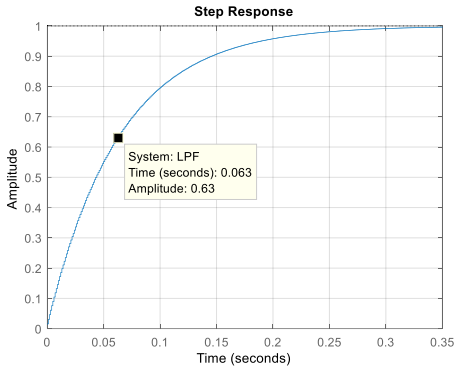
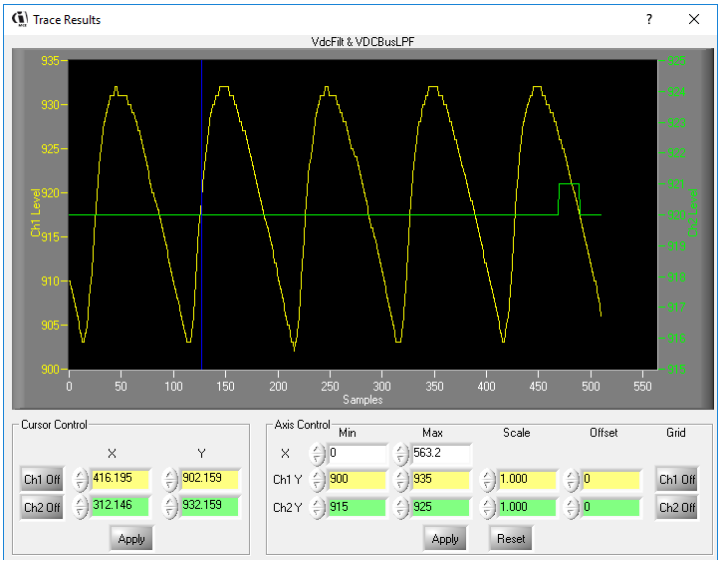


Figure 7 一阶无限冲激响应滤波器的阶跃响应曲线

2.2.4 实测结果

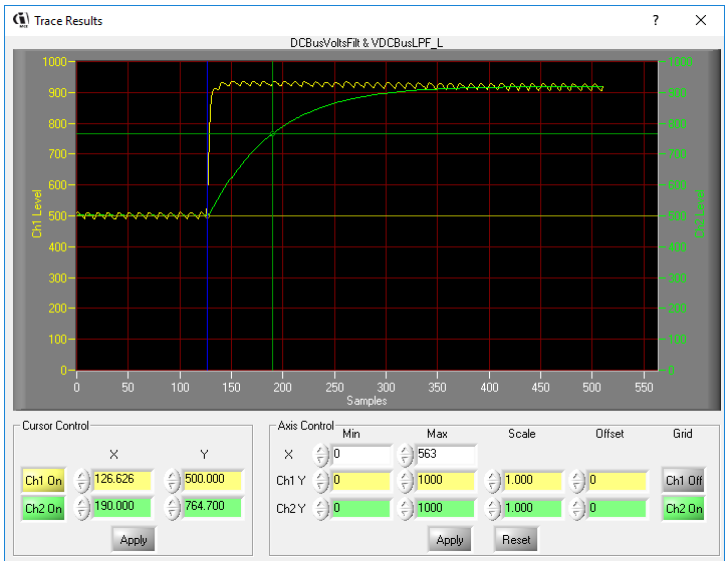
Figure 8 显示了低通滤波器的输入 `VdcFilt` 和输出 `VDCBusLPF_L`（`VDCBusLPF` 的低 16 位）波形。可以看到输出的结果波动不高于 1 个 ADC 计数值。相较于 `VdcFilt` 的 30 个 ADC 计数值，衰减约为-30 dB。

Figure 9 显示了低通滤波器的阶跃响应曲线，V_{in} 从 70 VAC 阶跃至 125VAC。初始的 ADC 计数值约为 500，阶跃稳态以后约为 919，阶跃幅度约为 419。阶跃上升 265 个($419 \cdot (1 - e^{-1}) = 419 \cdot 0.6321 = 265$)ADC 计数值需要的时间约为 63.374 个采样点。由于电机的 PWM 周期位 50 µs，波形的采样比例设定为 PWM 频率除以 20，等于 1 ms。测得的时间常数为$\tau_{measured} = 63.374$ 采样点 $\cdot\, 1\, ms = 63.374\, ms$和理论计算的结果是一致的。



Ch1: `VdcFilt`; Ch2: `VDCBusLPF_L`; F_{PWM}: 20kHz; Sample at PWM frequency divided by 2; Trigger Level: 920; Vin: 125VAC / 50Hz; V_{pp_CH1}: 30 ADC counts; V_{pp_CH2}: 1 ADC count

Figure 8 **`VdcFilt` & `VDCBusLPF_L` 波形**



Ch1: `VdcFilt`; Ch2: `VDCBusLPF_L`; F_{PWM}: 20kHz; Sample at PWM frequency divided by 20; Trigger Level: 600; Vin: step up from 70VAC to 125VAC; △X: 63.374 sample counts; △Y: 264.700 ADC counts

Figure 9　　实测滤波器阶跃响应曲线

2.3　　目标速度整定和母线欠压保护

2.3.1　　对目标速度的要求

吹风机要求基于瞬时的直流母线电压动态整定电机目标速度。我们以一个最高速为 20 kRPM 的 6 极永磁同步电机应用为例，2 档速度选择如 2.1 节中描述，目标速度和母线电压之间的关系由一个二次函数来定义，以下的两个不同系数的函数分别对应高速和低速。

$$TargetSpeed_{HS} = A_h \cdot {V_{DCBus}}^2 + B_h \cdot V_{DCBus} + C_h ;$$

$$TargetSpeed_{LS} = A_l \cdot {V_{DCBus}}^2 + B_l \cdot V_{DCBus} + C_l .$$

目标速度的单位是 RPM，母线电压V_{DCBus}的单位是伏，二次函数里的系数由 Table 2 列出。

Table 2　　目标速度函数里的系数

HIGH SPEED		LOW SPEED	
A_h	-0.159	A_l	-0.572
B_h	132.585	B_l	228.480
C_h	1494.450	C_l	-6153.675

由以上函数计算得出的目标速度需要落在对应的最大和最小限制值之间，Table 3 列出了高速和低速档速度限制范围。

Table 3　　高低速档的速度范围

	HIGH SPEED	LOW SPEED
Max. Target Speed Limit	16200 RPM	19400 RPM
Min. Target Speed Limit	11625 RPM	13537 RPM

另外，还需要母线欠压保护功能，来防止母线电压降低至某一个阈值以下时电机仍然运行。为了防止在欠压阈值附近反复启停，加了一个 5 V 的滞环。Table 4 列出了保护和恢复的阈值。

Table 4　　母线欠压和恢复电平

DC Bus Brown-In Voltage 恢复电平	90 V
DC Bus Brown-Out Voltage 欠压电平	85 V

Figure 10 和 Figure 11 给出了母线电压和目标速度之间的关系。蓝实线表示母线电压从 0 V 开始上升的过程，当母线电压超过 90 V 时，电机开始工作。红色虚线显示母线电压从高点降低，低于 85 V 时电机由运行转到停止的过程。

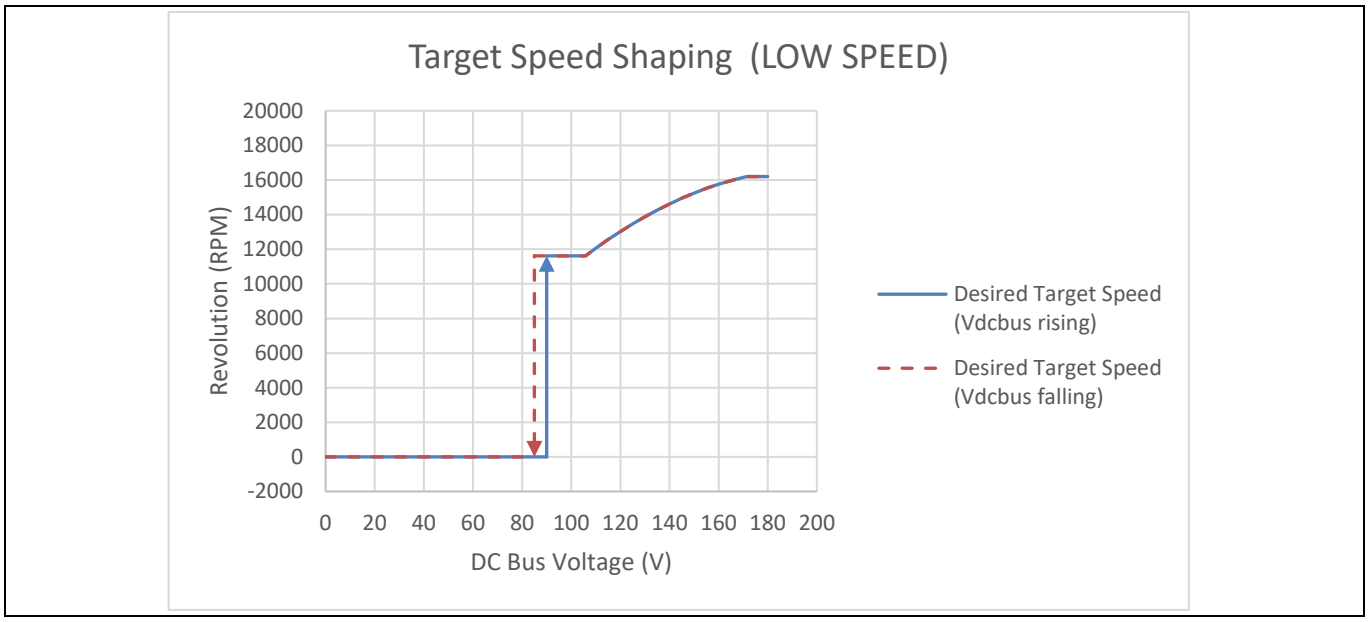


Figure 10　　Target Speed Shaping (LOW SPEED)目标速度整定（低速档）

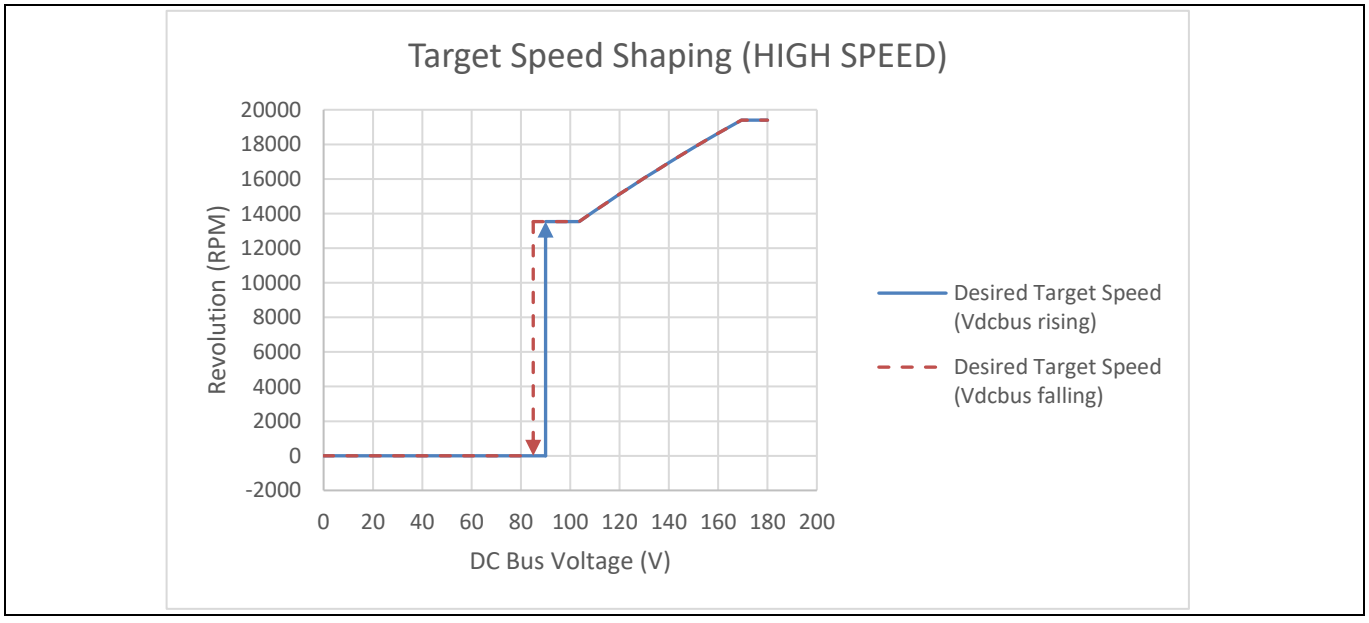


Figure 11　　Target Speed Shaping (HIGH SPEED)目标速度整定（高速档）

2.3.2 母线电压状态机

母线电压保护的状态机如 Figure 12 所示。我们用一个状态变量 DCBusState 来表征 2 个状态，母线电压异常状态 DC_Bus_State_Abnormal (`DCBusState = 0`)和母线电压正常状态 DC_Bus_State_Normal (`DCBusState = 1`)。状态机输入的母线电压信号必须要足够的稳定，因此我们选用 2.2 节中设计的低通滤波器的输出结果 `VDCBusLPF` 作为母线电压保护的输入信号。我们从母线电压异常状态 DC_Bus_State_Abnormal 作为起始状态，如果 VDCBusLPF 大于 `VDCBusBrownIn`，母线电压状态机的状态切换至正常状态 DC_Bus_State_Normal。当 VDCBusLPF 小于 `VDCBusBrownOut`，母线电压状态机的状态切换至异常状态 DC_Bus_State_Abnormal。

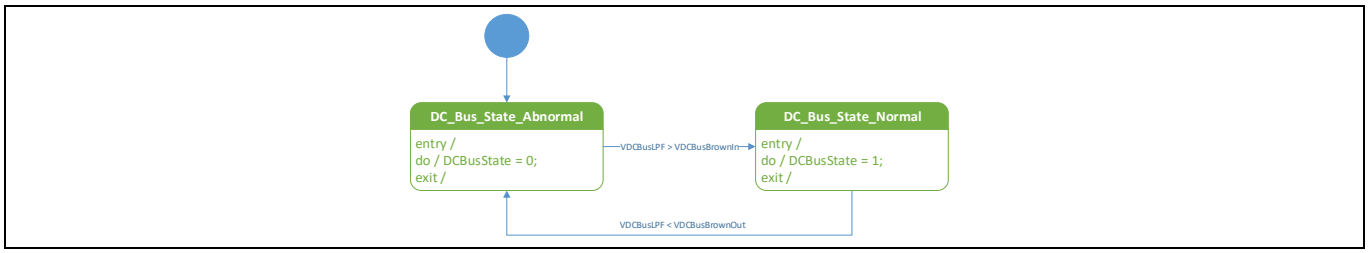


Figure 12 母线电压状态机

计算使用 2.2.2 节中的公式，电压设置参考 Table 4，计算结果显示在 Table 5 中。

Table 5

DC Bus Brown-In Voltage	90 V	`VDCBusBrownIn`	487 (ADC counts)
DC Bus Brown-Out Voltage	85 V	`VDCBusBrownOut`	460 (ADC counts)

2.3.3 目标速度整定计算的定标

2.3.1 节中定义了在不同速度挡位下目标速度$TargetSpeed$和母线电压V_{DCBus}的关系。目标速度的单位是 RPM，母线电压的单位是伏特。在电机控制软件中，目标速度用一个 16 位的有符号整型数来表示，16383 对应电机的最大速度。此例中 16383 对应 20 kRPM。母线电压用 ADC 的采样计数值来表示，转换公式参考 2.2.2 节。2.3.1 节中定义的公式并不能直接在脚本代码中应用，因此为了得到正确的计算结果，我们需要考虑标定因数，如下面公式所示。

$$TargetSpeed_{script} = \left[A \cdot (V_{DCBus_{ADC}} \cdot \frac{V_{ref}}{2^{12}-1} \cdot \frac{1}{G_{DCBus_{sensing}}})^2 + B \cdot (V_{DCBus_{ADC}} \cdot \frac{V_{ref}}{2^{12}-1} \cdot \frac{1}{G_{DCBus_{sensing}}}) + C\right] \cdot \frac{16383}{Speed_{max}}$$

其中$V_{ref} = 5V, G_{DCBus_sensing} = 0.00661$（在 2.2.2 节中表述），$Speed_{max} = 20000$，$A$，$B$，$C$是 2.3.1 节中给出的二次函数中的系数。

我们定义$T_{spd_factor} = \frac{16383}{Speed_{max}} = 0.819$，$V_{DCBus\_factor} = \frac{V_{ref}}{2^{12}-1} \cdot \frac{1}{G_{DCBus_{sensing}}} = 0.185$ 我们可以得到如下公式

$$TargetSpeed_{script} = (A \cdot T_{spd\_factor} \cdot V_{DCBus\_factor}{}^2) \cdot V_{DCBus_{ADC}}{}^2 + (B \cdot T_{spd\_factor} \cdot V_{DCBus\_factor}) \cdot V_{DCBus_{ADC}} + (C \cdot T_{spd\_factor})$$

如果我们定义$A_{script} = A \cdot T_{spd_factor} \cdot V_{DCBus_factor}{}^2$，$B_{script} = B \cdot T_{spd\_factor} \cdot V_{DCBus\_factor}$，$C_{script} = C \cdot T_{spd\_factor} \cdot V_{DCBus\_factor}$，则上式可以简化为

$$TargetSpeed_{script} = A_{script} \cdot V_{DCBus_{ADC}}{}^2 + B_{script} \cdot V_{DCBus_{ADC}} + C_{script}$$

在不同速度等级下公式的系数由 Table 6 列出。

Table 6　　目标速度和母线电压关系方程的浮点系数

HIGH SPEED		LOW SPEED	
A_{h_script}	-0.004	A_{l_script}	-0.016
B_{h_script}	20.074	B_{l_script}	34.593
C_{h_script}	1224.179	C_{l_script}	-5040.783

脚本引擎只支持 32 位有符号整数[2]，在脚本代码里这些浮点数需要用分数形式来表示。举例来讲，如果我们使用一个公共的分母 DEN，那么脚本代码中的目标速度计算通过 Code Listing 5 中的伪代码来实现。

Code Listing 5　　目标速度计算的伪代码

```
001        Speed_Value = A_NUM * VDCBus * VDCBus + B_NUM * VDCBus +
   C_NUM;
002        TargetSpeed = Speed_Value / DEN;
```

兼顾精确度和溢出限制，我们选择公共分母为 65536（Q15.16），这样所有的除法运算可以用右移 16 位来计算。在这种方式下，系数的分子部分计算结果参见 Table 7。

Table 7　　目标速度和母线电压关系方程的定标数（Q15.16）系数

$Denominator$		65536	
HIGH SPEED		**LOW SPEED**	
A_{h_NUM}	-291	A_{l_NUM}	-1049
B_{h_NUM}	1315558	B_{l_NUM}	2267069
C_{h_NUM}	80227767	C_{l_NUM}	-330352746

2.3.4　　目标速度整定和母线欠压保护的脚本代码实现

Code Listing 6 给出了目标速度整定和母线电压欠压保护的脚本源代码，运行在 Task1 中。由于目标速度不需要频繁的更新，建议将循环周期设定为 50 ms。编译生成的目标文件显示 Task1 的代码量为 42，故执行步数需要设定大于 42 来保证代码可以在一个循环周期内执行完。在此例中，我们设定 SCRIPT_TASK1_EXECUTION_PERIOD 为 5，SCRIPT_TASK1_EXECUTION_STEP 为 50 来保证满足设计要求。

此例也可以在 Task0 中实现，这时 SCRIPT_TASK0_EXECUTION_PERIOD 需要设定为 50 来实现 50 ms 的执行周期。

Code Listing 6　　目标速度整定和母线欠压保护脚本代码

```
001        /*************************************************************/
002        /*Script user version value, should be 255.255*/
003        #SET SCRIPT_USER_VERSION (1.00)
004        /*Script execution time for Task0 in ms, maximum value 65535*/
005        #SET SCRIPT_TASK0_EXECUTION_PERIOD (1)
006        /*Defines number of lines to be executed every 1ms in Task0*/
007        #SET SCRIPT_TASK0_EXECUTION_STEP (2)
008        /*Script execution time for Task1 in 10ms, maximum value
   65535*/
009        #SET SCRIPT_TASK1_EXECUTION_PERIOD (5)
010        /*Defines number of lines to be executed every 10ms in Task1*/
011        #SET SCRIPT_TASK1_EXECUTION_STEP (50)
012        /*************************************************************/
```

Code Listing 6 目标速度整定和母线欠压保护脚本代码

```
013         /* Global variable definition */
014         int VDCBusLPF;
015         int DCBusState;
016         int SpeedMode;
017         int SpeedValue;
018         /*************************************************************/
019         /*Task0 init function*/
020         Script_Task0_init()
021         {
022           /*Initialize global variable*/
023           VDCBusLPF = 0;
024           /* local variable definition */
025           int VDCBusMultiplyDEN;
026           /*Initialize local variable*/
027           VDCBusMultiplyDEN = 0;
028         }
029
030         /*Task0 script function*/
031         Script_Task0()
032         {
033           // Vdcbus filtering
034           VDCBusMultiplyDEN = VDCBusMultiplyDEN + (VdcFilt -
    VDCBusLPF);
035           VDCBusLPF = VDCBusMultiplyDEN >> 6;
036         }
037
038         /*Task1 init function*/
039         Script_Task1_init()
040         {
041           /* local variable definition */
042           int VDCBusBrownIn;  // Vdcbus_brown_in = 90V => 487 counts
043           int VDCBusBrownOut; // Vdcbus_brown_out = 85V => 460 counts
044
045           int VLSStart;
046           int VLSStop;
047           int VHSStart;
048           int VHSStop;
049
050           int AlNum;
051           int BlNum;
052           int ClNum;
053           int AhNum;
054           int BhNum;
055           int ChNum;
056           int DenShiftBit;
057
058           int TspdLSMin;
059           int TspdLSMax;
060           int TspdHSMin;
061           int TspdHSMax;
062
063           /*Initialize global variable*/
064           DCBusState = 0;
```

Code Listing 6 目标速度整定和母线欠压保护脚本代码

```
065          SpeedMode = 0;
066          SpeedValue = 0;
067
068          /*Initialize local variable*/
069          VDCBusBrownIn = 487;  // Vdcbus_brown_in = 90V => 487 counts
070          VDCBusBrownOut = 460; // Vdcbus_brown_out = 85V => 460
   counts
071
072          VLSStart = 819;    // Vsp_low_spd_start = 1V => 819 counts
073          VLSStop = 655;     // Vsp_low_spd_stop = 0.8V => 655 counts
074          VHSStart = 1638;   // Vsp_high_spd_start = 2V => 1638 counts
075          VHSStop = 1474;    // Vsp_high_spd_stop = 1.8V => 1474 counts
076
077          AlNum = -1049;
078          BlNum = 2267069;
079          ClNum = -330352770;
080          AhNum = -291;
081          BhNum = 1315558;
082          ChNum = 80227700;
083          DenShiftBit = 16; // Denominator = 2^16 = 65536
084
085          TspdLSMin = 9523; // In LOW_SPEED mode, Target Speed min =
   11625 RPM => 9523 counts.
086          TspdLSMax = 13270;  // In LOW_SPEED mode, Target Speed max =
   16200 RPM => 13270 counts.
087          TspdHSMin = 11089;  // In HIGH_SPEED mode, Target Speed min
   = 13537 RPM => 11089 counts.
088          TspdHSMax = 15892;  // In HIGH_SPEED mode, Target Speed max
   = 19400 RPM => 15892 counts.
089        }
090
091        /*Task1 script function*/
092        Script_Task1()
093        {
094          // DC bus state machine
095          if (DCBusState == 0)  // DCBus is abnormal.
096          {
097            if (VDCBusLPF > VDCBusBrownIn)
098            {
099              DCBusState = 1; // Shift to DCBus normal state.
100            }
101          }
102
103          if (DCBusState == 1)  // DCBus is normal.
104          {
105            if (VDCBusLPF < VDCBusBrownOut)
106            {
107              DCBusState = 0; // Shift to DCBus abnormal state.
108            }
109          }
110
111          // Speed selection state machine
112          if (SpeedMode == 0) // Speed selection is in OFF state.
```

Code Listing 6　　目标速度整定和母线欠压保护脚本代码

```
113          {
114            TargetSpeed = 0;
115            Command = 0;  // Stop the motor.
116
117            if (ADC_Result0 > VLSStart)
118            {
119              SpeedMode = 1;  // Shift to LOW_SPEED state.
120            }
121          }
122
123          if (SpeedMode == 1) // Speed selection is in LOW_SPEED
   state.
124          {
125            if (ADC_Result0 > VHSStart)
126            {
127              SpeedMode = 2;  // Shift to HIGH_SPEED state.
128            }
129            else
130            {
131              if (ADC_Result0 < VLSStop)
132              {
133                SpeedMode = 0;  // Shift to OFF state.
134              }
135              else  //Stay in LOW_SPEED state.
136              {
137                if (DCBusState == 1)  // DC bus voltage is normal.
138                {
139                  // Calculate target speed. Target speed follows 2nd
   order polynomial curve for LS.
140                  SpeedValue = AlNum * VDCBusLPF * VDCBusLPF + BlNum *
   VDCBusLPF + ClNum;
141                  SpeedValue = SpeedValue >> DenShiftBit;
142                  if (SpeedValue > TspdLSMax) // Upper limit check
143                  {
144                    SpeedValue = TspdLSMax;
145                  }
146                  if (SpeedValue < TspdLSMin) // Lower limit check
147                  {
148                    SpeedValue = TspdLSMin;
149                  }
150                  TargetSpeed = SpeedValue; // Update TargetSpeed.
151                  Command = 1;  // Start motor.
152                }
153                else  // DC bus voltage is abnormal.
154                {
155                  TargetSpeed = 0;  // Reset TargetSpeed.
156                  Command = 0;  // Stop motor.
157                }
158              }
159            }
160          }
161
```

Code Listing 6 目标速度整定和母线欠压保护脚本代码

```
162          if(SpeedMode == 2)  // Speed selection is in HIGH_SPEED
   state.
163          {
164            if(ADC_Result0 < VHSStop)
165            {
166              SpeedMode = 1;  // Shift to LOW_SPEED state.
167            }
168            else  // Stay in HIGH_SPEED state.
169            {
170              if (DCBusState == 1)  // DC bus voltage is normal.
171              {
172                // Target speed follows 2nd order polynomial curve for
   HS.
173                SpeedValue = AhNum * VDCBusLPF * VDCBusLPF + BhNum *
   VDCBusLPF + ChNum;
174                SpeedValue = SpeedValue >> DenShiftBit;
175                if (SpeedValue > TspdHSMax) // Upper limit check
176                {
177                  SpeedValue = TspdHSMax;
178                }
179                if (SpeedValue < TspdHSMin) // Lower limit check
180                {
181                  SpeedValue = TspdHSMin;
182                }
183                TargetSpeed = SpeedValue; // Update TargetSpeed.
184                Command = 1;  // Start motor.
185              }
186              else  // DC bus voltage is abnormal.
187              {
188                TargetSpeed = 0;  // Reset TargetSpeed.
189                Command = 0;  // Stop motor.
190              }
191            }
192          }
193        }
```

2.3.5 目标速度整定的测试结果

我们通过测量相电流频率来计算电压从 65 VAC 到 130 VAC 变化时候的实际电机运行速度。低速档位下实际测试结果和计算值之间对比关系列于 Table 8，并绘制在 Figure 13 中。我们可以看到，实际电机速度完全按照我们方程计算的目标速度来运行，误差不超过 1%。实际运行的速度范围满足预先设定的低速档下的最大和最小速度。

Table 8 目标速度和母线电压的测试结果（低速档）

V_{in} (Vrms)	V_{DCbus} (Vdc)	Measured Motor Speed (RPM)	Calculated Target Speed (RPM)	Target Speed Error (%)
64	85.8	11680	11625	0.5%
67	89.6	11740	11625	1.0%
78	105.6	11680	11590	0.8%
80	108.4	11900	11890	0.1%
90	122.0	13216	13204	0.1%

V_{in} (Vrms)	V_{DCbus} (Vdc)	Measured Motor Speed (RPM)	Calculated Target Speed (RPM)	Target Speed Error (%)
64	85.8	11680	11625	0.5%
67	89.6	11740	11625	1.0%
100	135.9	14380	14329	0.4%
110	149.9	15300	15238	0.4%
120	164.0	15980	15926	0.3%
125	171.1	16260	16188	0.4%
126	172.6	16264	16200	0.4%
130	178.3	16264	16200	0.4%

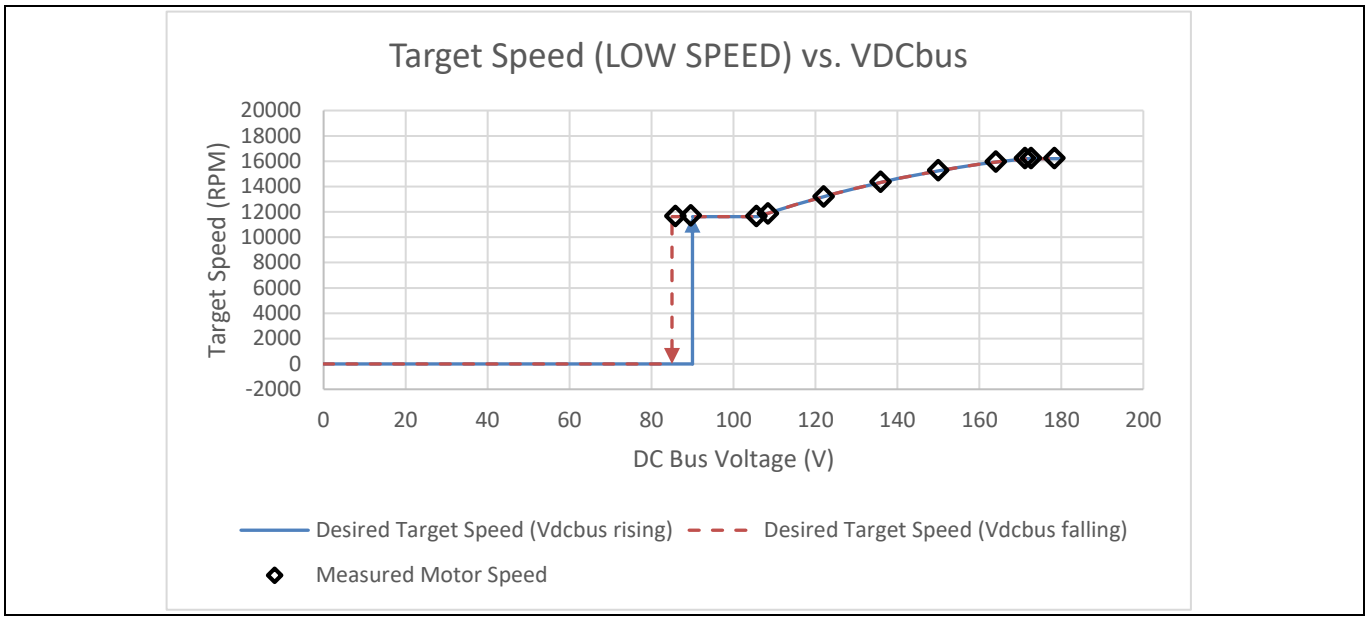


Figure 13 目标速度和母线电压的实测关系曲线（低速档）

Table 9 和 Figure 14 展示了高速档的测试数据，与低速档相同，速度误差低于 1%，实际运行的速度范围满足预先设定的高速档的最大和最小速度。

Table 9 目标速度和母线电压的测试结果（高速档）

V_{in} (Vrms)	V_{DCbus} (Vdc)	Measured Motor Speed (rpm)	Calculated Target Speed (rpm)	Target Speed Error (%)
65	86.3	13100	13537	-3.2%
66	87.9	13260	13537	-2.0%
77	103.7	13600	13537	0.5%
78	105.0	13628	13667	-0.3%
90	121.4	15280	15254	0.2%
100	135.2	16560	16519	0.2%
110	149.0	17762	17728	0.2%
120	162.8	18944	18875	0.4%
124	168.5	19454	19331	0.6%
125	170.0	19458	19400	0.3%
130	177.2	19466	19400	0.3%

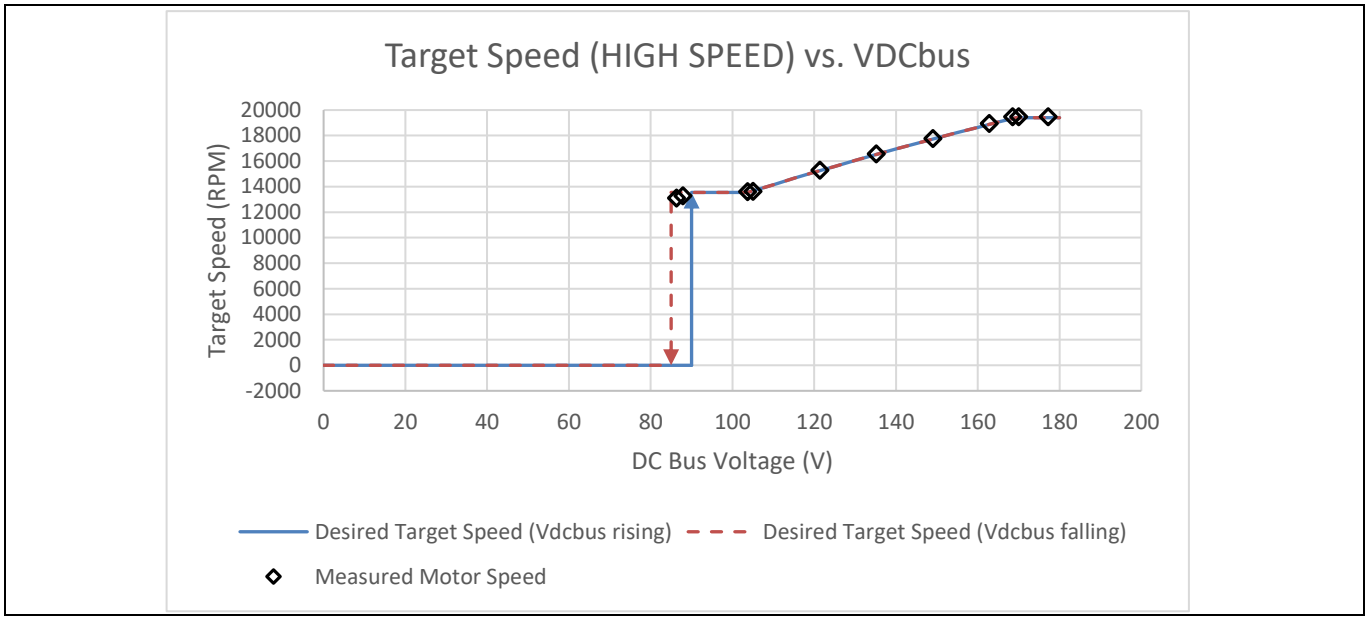


Figure 14 目标速度和母线电压的实测关系曲线（高速档）

2.4 动态电机电流限制

2.4.1 对动态电机电流限制的要求

电机控制内核默认电机电流最大值会限制在 100%额定电流，即 3 A。此吹风机应用需要基于速度选择实现更精确的电流限制，从而收紧转矩控制。在加减速过程中，电机电流限制值又需要放松到原来的设定，100%额定电流，以实现更快的速度响应。当电机停止时，电流限制值也将还原为原始默认设定。细节请参考 Table 10。

Table 10 电机电流限制要求

	Rated Current	Speed Ramp-up / Ramp-down Period	Speed Selection = OFF	Speed Selection = HIGH SPEED	Speed Selection = LOW SPEED
Motor Current Limit	3 A	3 A	3 A	0.6 A	0.38 A

2.4.2 动态电机电流限制算法设计和实现

Figure 15 展示了动态电流限制算法的流程图。初始化时，起始的电流限制值 MotorLim 存储于变量 CurrentLimitOriginal 中。在特定速度档位下，速度处于稳态阶段，最终的电流限制值由速度选择状态机通过寄存器 CurrentLimitTarget 来决定。瞬时的电机电流限制值 CurrentLimitValue 基于目标速度 TargetSpeed 和速度指令给定值 SpdRef 之间差的绝对值来计算。寄存器 CurrentLimitValue 的变化量是由变量 CurrentLimitIncrement 来决定的，在脚本代码中，CurrentLimitIncrement 设定为 100(counts / 10 ms)。由于速度指令给定值 SpdRef 的变化率要小于目标速度 TargetSpeed，如果它们的差值的绝对值大于变量 SpeedDiffThresh (100 counts)，则表示目前处于速度瞬态变化阶段，这时候电机电流限制值逐步增加至起始电流限制值。如果它们的差值的绝对值小于变量 SpeedDiffThresh (100 counts)，这时候表示速度进入稳态阶段，此时电流限制值逐步降低至 CurrentLimitTarget，CurrentLimitTarget 值是由当前的速度档位决定的。CurrentLimitValue 计算 10 ms 更新一次，MotorLim 值也会同步被 CurrentLimitValue 赋值。

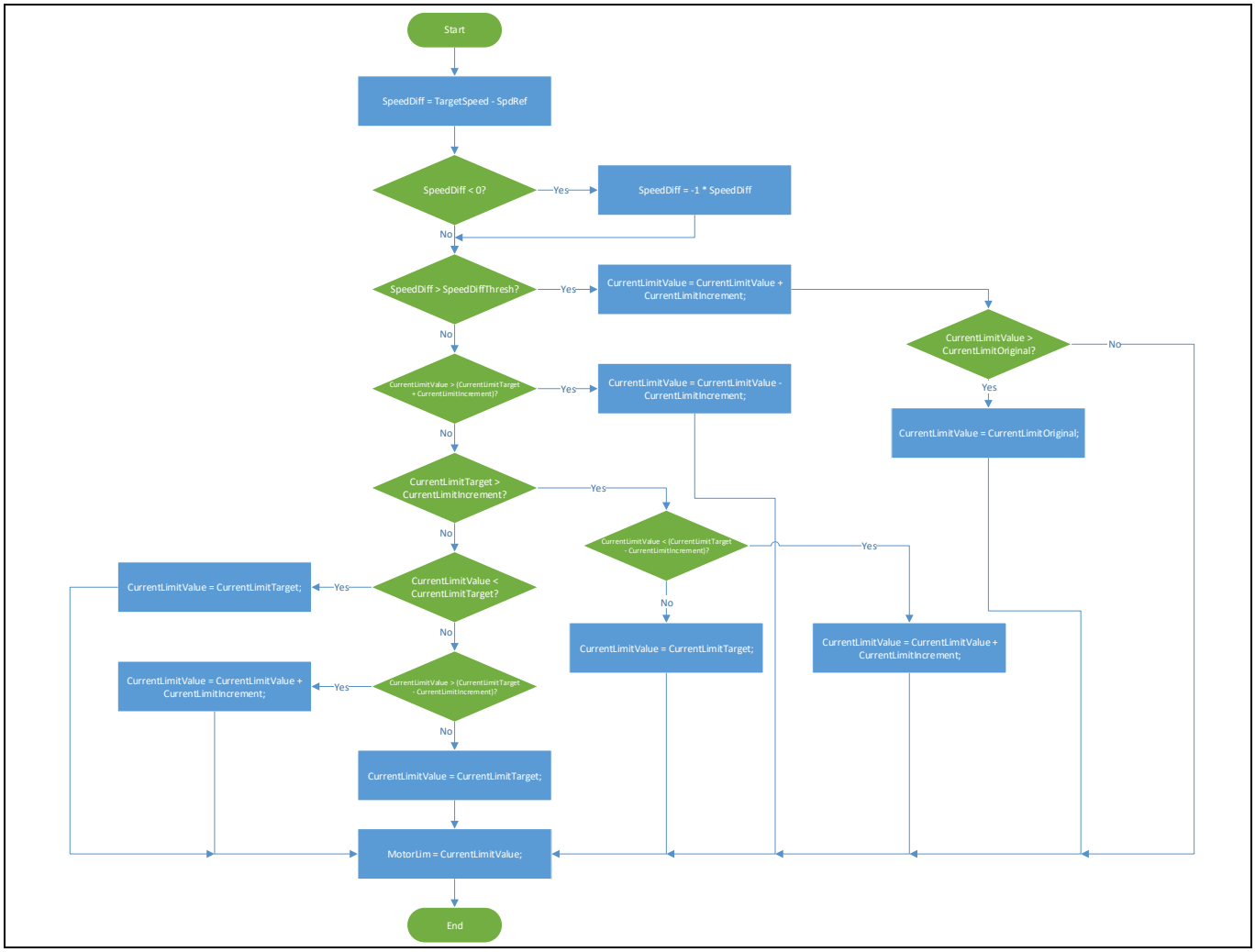


Figure 15　　动态电机电流限制算法的流程图

Code Listing 7 中代码即为动态电流限制的脚本代码，运行于 Task1 中。由于电机电流限制的变化量限制为 100 counts / 10 ms，建议设置 Task1 的循环周期为 10 ms。编译的目标文件显示 Task1 的指令数为 56，故执行步数 SCRIPT_TASK1_EXECUTION_STEP 需要设置大于 56 以保证 Task1 可以在周期内执行完毕。此例中，SCRIPT_TASK1_EXECUTION_PERIOD 设置为 1，SCRIPT_TASK1_EXECUTION_STEP 设置为 60。

同样此例可以在 Task0 中执行，此时 SCRIPT_TASK0_EXECUTION_PERIOD 应该设置为 10 来实现同样的执行周期 10 ms。

Code Listing 7　　动态电机电流限制算法的脚本代码

```
001        /*************************************************************/
002        /*Script user version value, should be 255.255*/
003        #SET SCRIPT_USER_VERSION (1.00)
004        /*Script execution time for Task0 in ms, maximum value 65535*/
005        #SET SCRIPT_TASK0_EXECUTION_PERIOD (1)
006        /*Defines number of lines to be executed every 1ms in Task0*/
007        #SET SCRIPT_TASK0_EXECUTION_STEP (2)
008        /*Script execution time for Task1 in 10ms, maximum value
   65535*/
009        #SET SCRIPT_TASK1_EXECUTION_PERIOD (1)
```

Code Listing 7　　动态电机电流限制算法的脚本代码

```
010         /*Defines number of lines to be executed every 10ms in Task1*/
011         #SET SCRIPT_TASK1_EXECUTION_STEP (60)
012         /*************************************************************/
013         /* Global variable definition */
014         int VDCBusLPF;
015         int DCBusState;
016         int SpeedDiff;
017         int CurrentLimitOriginal;
018         int CurrentLimitValue;
019         int CurrentLimitTarget;
020         int SpeedMode;
021         /*************************************************************/
022         /*Task0 init function*/
023         Script_Task0_init()
024         {
025           /*Initialize global variable*/
026           VDCBusLPF = 0;
027           /* local variable definition */
028           int VDCBusMultiplyDEN;
029           /*Initialize local variable*/
030           VDCBusMultiplyDEN = 0;
031         }
032
033         /*Task0 script function*/
034         Script_Task0()
035         {
036           // Vdcbus filtering
037           VDCBusMultiplyDEN = VDCBusMultiplyDEN + (VdcFilt -
    VDCBusLPF);
038           VDCBusLPF = VDCBusMultiplyDEN >> 6;
039         }
040
041         /*Task1 init function*/
042         Script_Task1_init()
043         {
044           /* local variable definition */
045           int VDCBusBrownIn;  // Vdcbus_brown_in = 90V => 487 counts
046           int VDCBusBrownOut; // Vdcbus_brown_out = 85V => 460 counts
047           int SpeedDiffThresh;
048           int CurrentLimitIncrement;
049           int CurrentLimitLS;
050           int CurrentLimitHS;
051
052           int VLSStart;
053           int VLSStop;
054           int VHSStart;
055           int VHSStop;
056           int LowSpeedValue;
057           int HighSpeedValue;
058
059           /*Initialize global variable*/
060           DCBusState = 0;
061           SpeedDiff = 0;
```

Code Listing 7　　动态电机电流限制算法的脚本代码

```
062          CurrentLimitOriginal = MotorLim;  // Save the original motor
   current limit set in MCEWizard.
063          CurrentLimitValue = CurrentLimitOriginal; // The initial
   value needs to be synced with the original motor current limit set in
   MCEWizard.
064          CurrentLimitTarget = CurrentLimitOriginal;  // The initial
   value needs to be synced with the original motor current limit set in
   MCEWizard.
065          SpeedMode = 0;
066
067          /*Initialize local variable*/
068          VDCBusBrownIn = 487;  // Vdcbus_brown_in = 90V => 487 counts
069          VDCBusBrownOut = 460; // Vdcbus_brown_out = 85V => 460
   counts
070
071          SpeedDiffThresh = 100;  // Set the speed difference
   threshold to 100 counts.
072          CurrentLimitIncrement = 100;  // Motor current limit ramp
   rate = 100 counts / update interval (10 ms).
073          CurrentLimitLS = 519; // low speed motor current limit =
   0.38A => 519 counts
074          CurrentLimitHS = 819; // high speed motor current limit =
   0.6A => 819 counts
075
076          VLSStart = 819;    // Vsp_low_spd_start = 1V => 819 counts
077          VLSStop = 655;     // Vsp_low_spd_stop = 0.8V => 655 counts
078          VHSStart = 1638;   // Vsp_high_spd_start = 2V => 1638 counts
079          VHSStop = 1474;    // Vsp_high_spd_stop = 1.8V => 1474 counts
080
081          LowSpeedValue = 5000;
082          HighSpeedValue = 10000;
083        }
084
085        /*Task1 script function*/
086        Script_Task1()
087        {
088          // DC bus state machine
089          if (DCBusState == 0)  // DCBus is abnormal.
090          {
091            if (VDCBusLPF > VDCBusBrownIn)
092            {
093              DCBusState = 1; // Shift to DCBus normal state.
094            }
095          }
096
097          if (DCBusState == 1)  // DCBus is normal.
098          {
099            if (VDCBusLPF < VDCBusBrownOut)
100            {
101              DCBusState = 0; // Shift to DCBus abnormal state.
102            }
103          }
```

Code Listing 7　　动态电机电流限制算法的脚本代码

```
104          // Calculate the difference between the target speed and the
   speed reference in preparation for motor current limit calculation.
105          SpeedDiff = TargetSpeed - SpdRef; // Find out the difference
   between the speed reference and the target speed.
106          if(SpeedDiff < 0) // The target speed is lower than the
   speed reference.
107          {
108            SpeedDiff = -1 * SpeedDiff; // Takes the absolute value of
   SpeedDiff.
109          }
110          // Calculate motor current limit based on speed reference
   and target speed.
111          if(SpeedDiff > SpeedDiffThresh) // The speed reference is
   more than SpeedDiffThresh counts different from the target speed. We
   need to increase the motor current limit to its original value
   temperarily.
112          {
113            CurrentLimitValue = CurrentLimitValue +
   CurrentLimitIncrement;  // Increase the motor current limit by
   CurrentLimitIncrement until it reaches CurrentLimOriginal.
114            if (CurrentLimitValue > CurrentLimitOriginal) // Upper
   boundary check for CurrentLimitValue.
115            {
116              CurrentLimitValue = CurrentLimitOriginal;
117            }
118          }
119          else  // The speed reference is no more than 100 counts
   different from the target speed. We need to decrease the motor current
   limit to CurrentLimitTarget.
120          {
121            if(CurrentLimitValue > (CurrentLimitTarget +
   CurrentLimitIncrement))  // The motor current limit value at this
   moment is greater than the specified motor current limit by more than
   CurrentLimitIncrement.
122            {
123              CurrentLimitValue = CurrentLimitValue -
   CurrentLimitIncrement;  // Decrease the motor current limit target by
   CurrentLimitIncrement.
124            }
125            else  // The motor current limit target is no more than
   the specified motor current limit by more than CurrentLimitIncrement.
126            {
127              if (CurrentLimitTarget > CurrentLimitIncrement) //
   CurrentLimitTarget is greater than CurrentLimitIncrement. Boundary
   check needed for the following minus operation.
128              {
129                if (CurrentLimitValue < (CurrentLimitTarget -
   CurrentLimitIncrement)) // The motor current limit value at this
   moment is less than the specified motor current limit by more than
   CurrentLimitIncrement.
130                {
```

Code Listing 7　　动态电机电流限制算法的脚本代码

```
131                 CurrentLimitValue = CurrentLimitValue +
   CurrentLimitIncrement;  // Increase the motor current limit target by
   CurrentLimitIncrement.
132               }
133               else  // The motor current limit target falls between
   CurrentLimitTarget - CurrentLimitIncrement and CurrentLimitTarget +
   CurrentLimitIncrement.
134               {
135                 CurrentLimitValue = CurrentLimitTarget; // Set the
   motor current limit target to the specified motor current limit.
136               }
137             }
138             else  // CurrentLimitTarget is no more than
   CurrentLimitIncrement.
139             {
140               if (CurrentLimitValue < CurrentLimitTarget)
141               {
142                 CurrentLimitValue = CurrentLimitTarget; // Set the
   motor current limit target to the specified LOW_SPEED motor current
   limit.
143               }
144               else  // CurrentLimitValue is greater than
   CurrentLimitTarget
145               {
146                 if(CurrentLimitValue > (CurrentLimitTarget -
   CurrentLimitIncrement))  // The motor current limit value at this
   moment is less than the specified LOW_SPEED motor current limit by
   more than CurrentLimitIncrement.
147
148                 {
149                   CurrentLimitValue = CurrentLimitValue +
   CurrentLimitIncrement;  // Increase the motor current limit target by
   CurrentLimitIncrement.
150                 }
151                 else  //The motor current limit value is within the
   range of CurrentLimitTarget and CurrentTarget - CurrentLimitIncrement.
152                 {
153                   CurrentLimitValue = CurrentLimitTarget; // Set the
   motor current limit target to the specified motor current limit.
154                 }
155               }
156             }
157           }
158         }
159         MotorLim = CurrentLimitValue; // Update MotorLim.
160
161         // Speed selection state machine
162         if (SpeedMode == 0) // Speed selection is in OFF state.
163         {
164           TargetSpeed = 0;
165           CurrentLimitTarget = CurrentLimitOriginal;
166           Command = 0;  // Stop the motor.
167
```

Code Listing 7　　动态电机电流限制算法的脚本代码

```
168             if (ADC_Result0 > VLSStart)
169             {
170               SpeedMode = 1;  // Shift to LOW_SPEED state.
171             }
172           }
173
174           if (SpeedMode == 1) // Speed selection is in LOW_SPEED
    state.
175           {
176             if (ADC_Result0 > VHSStart)
177             {
178               SpeedMode = 2;  // Shift to HIGH_SPEED state.
179             }
180             else
181             {
182               if (ADC_Result0 < VLSStop)
183               {
184                 SpeedMode = 0;  // Shift to OFF state.
185               }
186               else  //Stay in LOW_SPEED state.
187               {
188                 if (DCBusState == 1)  // DC bus voltage is normal.
189                 {
190                   TargetSpeed = LowSpeedValue;  // Update TargetSpeed.
191                   CurrentLimitTarget = CurrentLimitLS;
192                   Command = 1;  // Start motor.
193                 }
194                 else  // DC bus voltage is abnormal.
195                 {
196                   TargetSpeed = 0;  // Reset TargetSpeed.
197                   CurrentLimitTarget = CurrentLimitOriginal;  // When
    the target speed is zero, motor current limit is restored back to the
    original limit.
198                   Command = 0;  // Stop motor.
199                 }
200               }
201             }
202           }
203
204           if(SpeedMode == 2)  // Speed selection is in HIGH_SPEED
    state.
205           {
206             if(ADC_Result0 < VHSStop)
207             {
208               SpeedMode = 1;  // Shift to LOW_SPEED state.
209             }
210             else  // Stay in HIGH_SPEED state.
211             {
212               if (DCBusState == 1)  // DC bus voltage is normal.
213               {
214                 // Target speed follows 2nd order polynomial curve for
    HS.
215                 TargetSpeed = HighSpeedValue; // Update TargetSpeed.
```

Code Listing 7　　动态电机电流限制算法的脚本代码

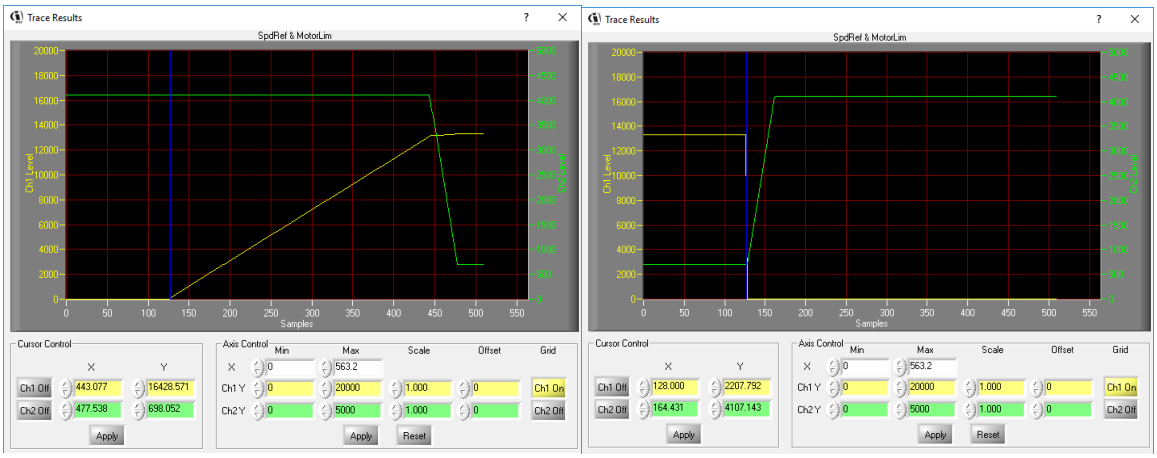
```
216                 CurrentLimitTarget = CurrentLimitHS;
217                 Command = 1;  // Start motor.
218               }
219               else  // DC bus voltage is abnormal.
220               {
221                 TargetSpeed = 0;  // Reset TargetSpeed.
222                 CurrentLimitTarget = CurrentLimitOriginal;  // When
   the target speed is zero, motor current limit is restored back to the
   original limit.
223                 Command = 0;  // Stop motor.
224               }
225             }
226           }
227         }
```

2.4.3　　动态电机电流限制的测试结果

Figure 16 显示了电机电流限制在速度选择从关机状态 OFF 到高速状态 HIGH SPEED 变化时如何动态的切换。当速度模式从关机到高速状态 HIGH SPEED 时候，电机控制器将 MotorLim 存储在变量 CurrentLimitOriginal 中，当电机目标速度指令 SpdRef 接近高速模式的目标速度 targetspeed 时候，进入稳态速度模式，MotorLim 以 100 counts / 10 ms 的斜率逐渐降低，在 330 ms 后稳定在高速模式下的限制值 CurrentLimitHS = 819。当速度模式从高速 HIGH SPEED 切换至关机状态时，速度指令参考 SpdRef 立即清零，这时候 MotorLim 以 100 counts / 10 ms 的斜率逐渐升高，在 330 ms 后稳定在 CurrentLimitOriginal 所设定的值。

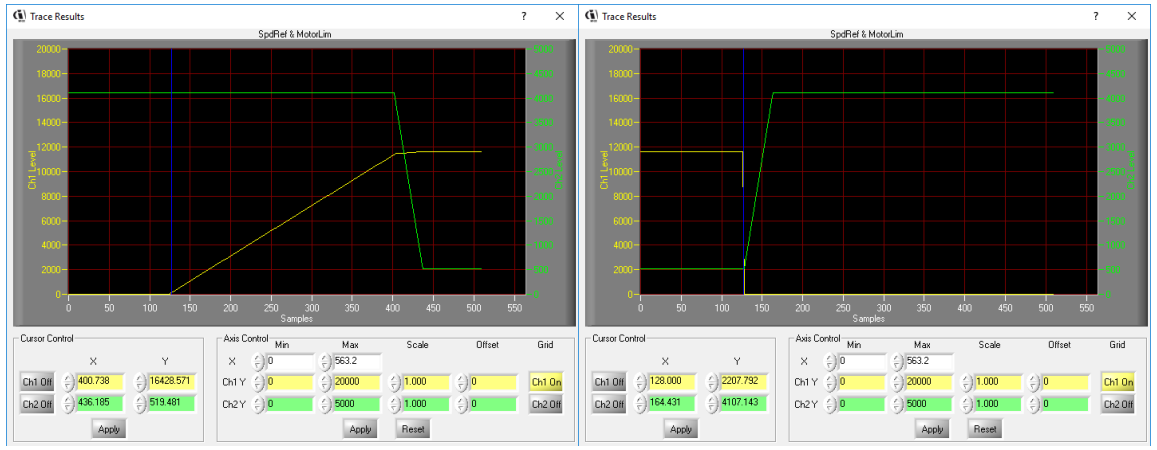
Figure 17 显示了电机电流限制在速度选择从关机状态 OFF 到低速状态 LOW SPEED 变化时如何动态的切换。当速度模式从关机到低速状态 LOW SPEED 时候，电机控制器将 MotorLim 存储在变量 CurrentLimitOriginal 中，当电机目标速度指令 SpdRef 接近低速模式的目标速度 Targetspeed 时候，进入稳态速度模式，MotorLim 以 100 counts / 10 ms 的斜率逐渐降低，在 360 ms 后稳定在低速模式下的限制值 CurrentLimitLS = 519。当速度模式从低速 LOW SPEED 切换至关机状态时，速度指令参考 SpdRef 立即清零，这时候 MotorLim 以 100 counts / 10 ms 的斜率逐渐升高，在 360 ms 后稳定在 CurrentLimitOriginal 所设定的值。

Figure 18 显示了在电机从低速状态 LOW SPEED 切换到高速状态 HIGH SPEED 时电机电流限制的变化情况。当电机从低速状态切换至高速状态时，电机目标速度指令 SpdRef 开始逐渐增加，电机电流限制 MotorLim 以 100 counts / 10 ms 的斜率从 CurrentLimitLS = 519 变化至 CurrentLimitOriginal。当 SpdRef 进入高速稳定状态时候，MotorLim 将按照同样的斜率下降，最终稳定在 CurrentLimitHS = 819。当电机从高速模式切换至低速模式时，电机目标速度指令 SpdRef 开始逐渐降低，电机电流限制 MotorLim 以 100 counts / 10 ms 的斜率从 CurrentLimitHS = 819 变化至 CurrentLimitOriginal。当 SpdRef 进入低速稳定状态时候，MotorLim 将按照同样的斜率下降，最终稳定在 CurrentLimitLS = 519。



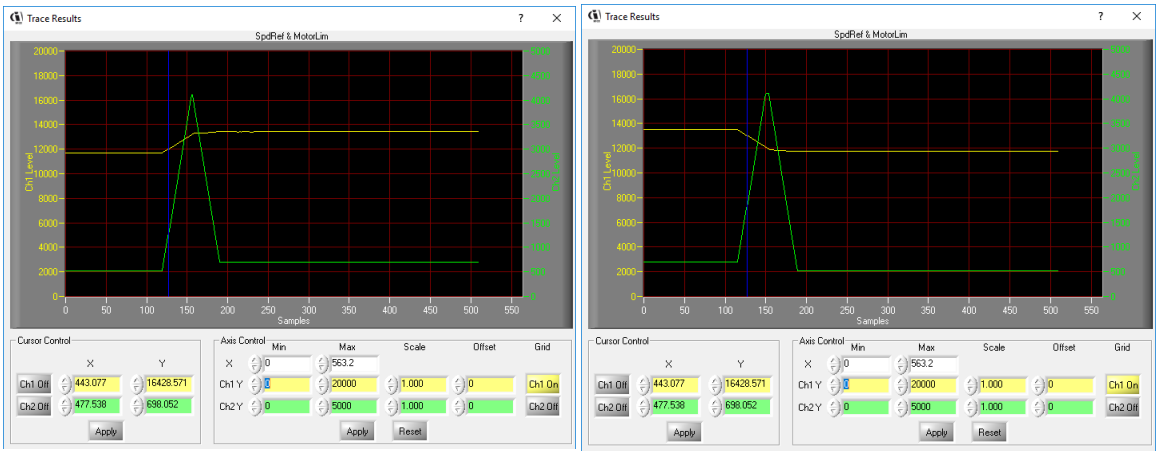
Ch1: SpdRef; Ch2: MotorLim; F_{PWM}: 20kHz; Sample at PWM frequency divided by 200; Trigger Level: 100; Speed Selection: OFF to HIGH SPEED (left) / HIGH SPEED to OFF (right); Vin: 100VAC

Figure 16　动态电机电流限制曲线（停机<->高速档）



Ch1: SpdRef; Ch2: MotorLim; F_{PWM}: 20kHz; Sample at PWM frequency divided by 200; Trigger Level: 100; Speed Selection: OFF to LOW SPEED (left) / LOW SPEED to OFF (right); Vin: 100VAC

Figure 17　动态电机电流限制曲线（停机<->低速档）



Ch1: SpdRef; Ch2: MotorLim; F_{PWM}: 20kHz; Sample at PWM frequency divided by 200; Trigger Level: 12000; Speed Selection: LOW SPEED to HIGH SPEED (left) / LOW SPEED to HIGH SPEED (right); Vin: 100VAC

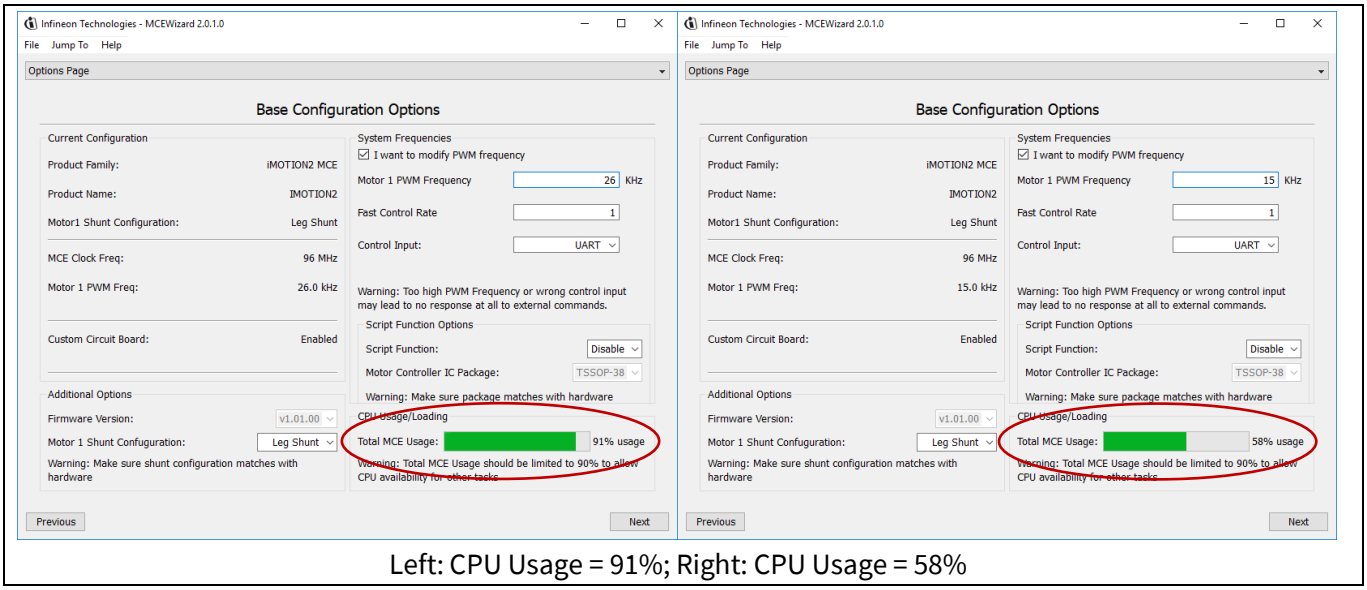
Figure 18　动态电机电流限制曲线（低速档<->高速档）

3 脚本性能评估

3.1 CPU 负荷评估

CPU 资源是优先分配给电机控制和 PFC 控制算法的。脚本引擎则在 CPU 空闲时候来处理脚本代码。由于脚本引擎优先级低于电机控制和 PFC 控制，故它不会影响到控制算法的性能。然而，在使能脚本功能前，还是需要仔细评估 CPU 资源。

电机和 PFC 开关频率的变化以及各种保护功能都会导致估算的 CPU 利用率发生变化。我们可以通过 MCEWizard 来估算 CPU 利用率。如果 CPU 利用率如 Figure 19 左图所示高于 90%，使能脚本功能可能导致 CPU 过载。所以我们推荐在开启脚本功能前，检查 CPU 利用率不高于 90%。

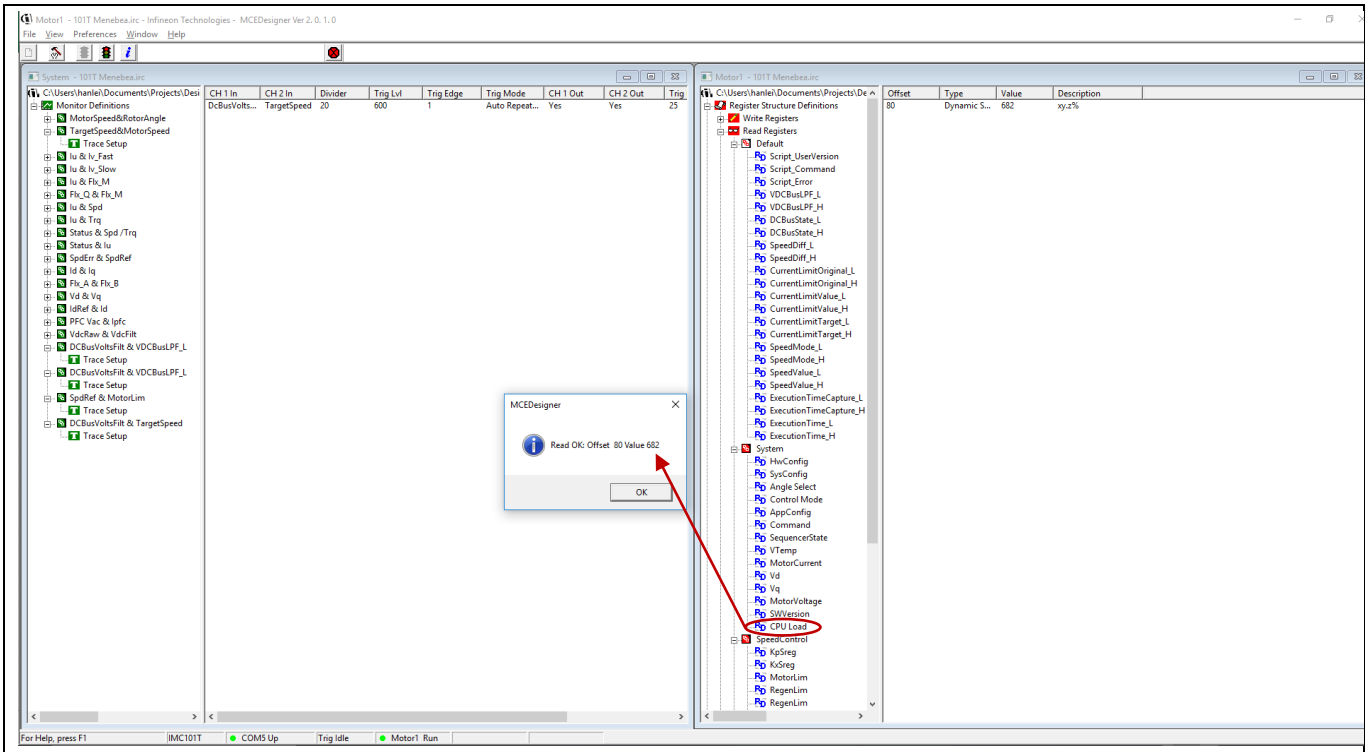


Left: CPU Usage = 91%; Right: CPU Usage = 58%

Figure 19 通过 MCEWizard 评估 CPU 利用率

脚本代码的复杂度，执行周期以及执行步长都会影响到 CPU 的负荷。推荐评估当脚本代码执行时候的 CPU 负载大小，从而保证 CPU 不过载。

CPU 的负荷程度可以通过软件 MCEDesigner[3]读取寄存器 CPU Load[2]来得到，单位数表示 0.1%。Figure 20 显示，使能 2.3 节中低速速度选择模式运行脚本代码的 CPU 负荷为 68.2%。脚本代码越复杂，也就需要更多的 CPU 资源。



Speed Selection: LOW SPEED; Vin: 100VAC

Figure 20　　通过 MCEDesigner 读取 CPU Load 寄存器

3.2　　脚本任务的时序

3.2.1　　脚本任务的时序设置

脚本引擎支持两个独立并行的任务，Task0 和 Task1。Task0 在每 1 ms 一次的系统计时中断中执行，而 Task1 则由后台的循环任务来执行，Task0 优先级要高于 Task1.

用户的脚本代码在 Task0 和 Task1 循环执行。Task0 的执行周期通过参数 `SCRIPT_TASK0_EXECUTION_PERIOD` 来设置，Task0 的最小周期为 1 ms。举例来讲，设定 `SCRIPT_TASK0_EXECUTION_PERIOD` 为 5 表示执行周期为$5 \cdot 1\ ms = 5\ ms$。Task1 的执行周期通过参数 `SCRIPT_TASK1_EXECUTION_PERIOD` 来设置，Task1 的最小周期为 10 ms。举例来讲，设定 `SCRIPT_TASK1_EXECUTION_PERIOD` 为 5 表示执行周期为$5 \cdot 10\ ms = 50\ ms$。

任务在每个执行周期执行的代码条数通过参数 SCRIPT_TASK0_EXECUTION_STEP（Task0）和 SCRIPT_TASK1_EXECUTION_STEP（Task1）来配置[2]。

每个脚本任务的时间设置需要根据实际应用的需求来设定。

3.2.2　　脚本任务的执行时间评估

Task0 和 Task1 的执行时间可以利用 MCE 寄存器 `RunTimeCounter` 来检测。`RunTimeCounter` 是一个每毫秒累加的计时器，它可以由脚本代码直接读取。Code Listing 8 中就是利用此寄存器检测的一个实例，在 Task1 开始时候，读取该寄存器存于变量 `ExecutionTimeCapture`，在任务结束后，再次读取该寄存器，差值存于 `ExecutionTime` 中，作为一个全局变量，`ExecutionTime` 可以由 MCEDesigner 在运行期间读取。

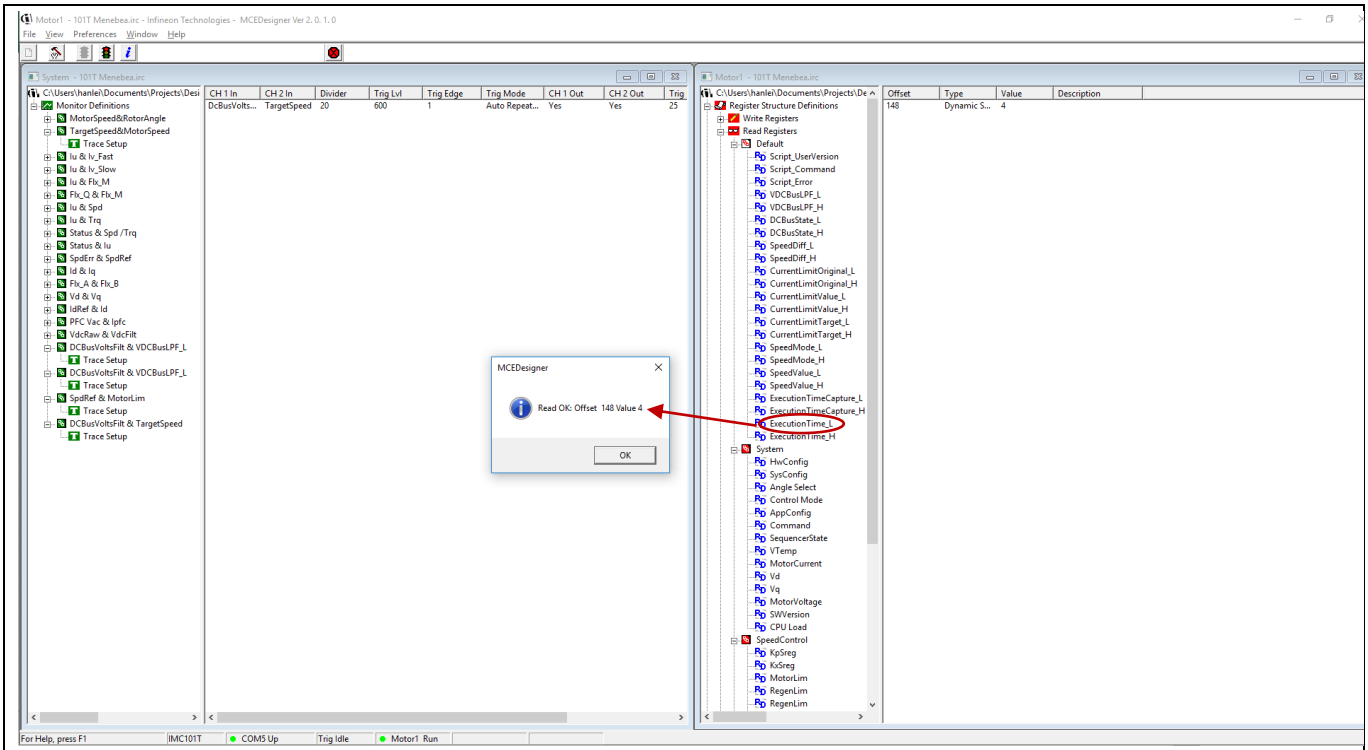
我们用 2.4 节中的脚本代码的例子来评估 Task1 的执行时间，Task1 的执行周期设置为 10 ms。Figure 21 显示，当脚本代码使能并运行在低速模式时候，变量 `ExecutionTime_L`（`ExecutionTime` 的低 16 位）结果为 4。这代表在电机运行时 Task1 的执行时间是 4 ms。由于 Task1 的实际执行时间小于其设定的执行周期，故 Task1 不会超时。

脚本代码越复杂，执行的时间也越长。只要脚本任务的执行时间不超过设定的执行周期，那么任务的执行就不会超时，设定的周期时间是可以保证的。如果脚本任务的执行时间超过设定的执行周期，那么我们设定的周期就不能保证，在这种情况下，任务会接着把没有执行完毕的代码完成才会立即开始执行下一个任务周期，这样实际的任务周期是由脚本任务的实际执行时间来决定。

如果 Task0 的周期设定为 1 ms，由于分辨率的限制，我们是不能通过 `RunTimeCounter` 来估算任务执行时间的。在这种情况下，我们直接通过 CPU 负荷来评估 Task0 的执行状态。如果 CPU 负荷低于 95%，Task0 中设定数目的代码是可以保证在 1 ms 周期的执行完毕的。如果任务没有在 1 ms 内执行完设定条数的代码，则会报 CPU 过载故障，寄存器 `FaultFlags` 的第 10 位会置 1[2],在安全功能关闭的状态下，系统会进入故障状态，而在安全功能打开的状态下，系统会进入安全故障模式（failsafe mode）。

Code Listing 8　　通过 `RunTimeCounter` 测量 `Task1` 的执行时间的脚本代码

```
/*************************************************************/
/* Global variable definition */
int ExecutionTimeCapture;
int ExecutionTime;
/*************************************************************/
/*Task1 script function*/
Script_Task1()
{
      ExecutionTimeCapture = RunTimeCounter;
…
…
…
      ExecutionTime = RunTimeCounter - ExecutionTimeCapture;
}
```



Speed Selection: LOW SPEED; Vin: 100VAC

Figure 21　通过 MCEDesigner 读取 ExecutionTime_L 变量

3.2.3　脚本任务的执行周期评估

Code Listing 9 列出了应用寄存器 `RunTimeCounter` 来检测 Task1 执行时间的脚本代码例子。`RunTimeCounter` 是一个可以由脚本代码直接读取的分辨率为 1 ms 的寄存器。

Code Listing 9　在脚本代码中通过 `RunTimeCounter` 测量 Task1 的执行周期

```
/*************************************************************/
/* Global variable definition */
int LoopExecutionPeriodCapture;
int LoopExecutionPeriod;
/*************************************************************/
/*Task1 script function*/
Script_Task1()
{
  LoopExecutionPeriod = RunTimeCounter -
LoopExecutionPeriodCapture;
  LoopExecutionPeriodCapture = RunTimeCounter;
  …
  …
  …
}
```

4 脚本代码规范和限制

- 脚本引擎支持的最大全局变量数目为 30，每个任务的最大局部变量的数目为 24。Task0 和 Task1 之间交互数据需要通过全局变量。只有全局变量可以被 MCEDesigner 和用户串口通讯读取，所以当用户需要在 MCEDesigner 中读取变量时，该变量需要定义为全局变量[3]。
- 脚本代码最大的容量为 16kB，大概约为 1500 行脚本代码，脚本代码的行数可以由编译生成的目标文件读取。Code Listing 2 的第 6 行就是一个例子。
- 脚本引擎只支持 32 为有符号整型变量，脚本代码中，浮点数需要转换为定点数来处理。转换方法参考 2.3.3 节。
- 脚本引擎支持两个独立并行的任务，分别为 Task0 和 Task1。脚本代码在 Task0 和 Task1 中以设定的周期循环运行。Task0 最短执行周期为 1 ms， Task1 最短执行周期为 10 ms，实际运行周期可以设定为该任务最短执行周期的整数倍。Task0 优先级高于 Task1，实际执行周期的设定需要根据实际系统应用的需求来决定。
- 脚本引擎中模拟输入端口的采样时间为 10 ms 一次，通过奈奎斯特采样理论可以知道，如果模拟信号的频率大于 50 Hz，采样结果是不能反映真实信号的。建议使用模拟低通滤波器将 50Hz 以上的信号衰减以降低信号失真。
- 脚本引擎中通用输入输出端口（GPIO）由 MCE 10 ms 更新一次，任何低于 10 ms 的电平切换将不能正常捕捉到，同样的，快于 10 ms 的端口状态切换也不能通过脚本引擎来实现。最快的 GPIO 电平转换频率是 50 Hz。
- 推荐只在 Task 0 或 Task 1 中进行一次更改 GPIO 端口电平的操作。如果在 Task0 或 Task1 中多次更改 GPIO 的端口电平，只有最后一次操作有效，这是由 MCE 软件的操作机制决定的。比如说：一开始某个 GPIO 端口复位为低电平 0，如果在 Task0 开始时候设置为高电平，然后在 Task0 最后将其复位为低电平，那么实际上这个 GPIO 的电平是不会切换状态的，而是会一直维持在低电平状态。
- 对于执行时间比较敏感的功能，推荐在 Task0 中来执行，其最短执行周期可以设置为 1 ms。对于执行时间要求不高的功能，Task0 和 Task1 都可以执行，建议将执行周期设置为 50 ms。
- Task0 的最小执行周期为 1 ms， 故它所执行的数字滤波器的采样频率可以为 1 kHz，因此 500Hz 带宽以上的信号通过数字滤波器是无法处理的。
- 脚本代码不支持红外通信。
- 由于 GPIO 更新速率的影响，脚本代码不支持可控硅移相控制。
- 脚本代码支持瞬态开关接口（如，继电器）。
- 如果最小时间要求不少于 10 ms，脚本代码可以支持 PLC 接口。
- 脚本代码不支持数字霍尔传感器接口。

5 参考文献

[1] iMOTION™ IMC100 High Performance Motor Control IC Series Datasheet (REV 1.1).

[2] iMOTION™ Motor Control Engine Software Reference Manual (REV 1.1).

[3] MCEDesigner User Guide (REV 2.0.1.0).

更新历史

文档版本	发布时间	变更内容
1.0	12/7/2018	首次发布

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 12/7/2018

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference
AN2018-27 How to Use iMOTION™ Script Language

IMPORTANT NOTICE

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.