Power calculation and constant-power control

Implemented by iMOTION™ script language

About this document

Scope and purpose

This document provides the constant-power control algorithm based on the iMOTION™ 2.0 script language and the power calculation method.

Intended audience

This document is intended for those who would like to use the script language option to implement the power calculation and the constant-power control.

Table of contents

About this document.................................................................................................................. 1
Table of contents.......................................................................................................................... 1
1 Introduction ................................................................................................................................. 2
2 Motor power calculation ............................................................................................................. 3
   2.1 Introduction............................................................................................................................. 3
   2.2 Implementation......................................................................................................................... 3
   2.3 Test results .............................................................................................................................. 5
      2.3.1 No over-modulation condition ......................................................................................... 6
      2.3.2 Over-modulation condition ............................................................................................. 8
3 Constant-power control............................................................................................................... 11
   3.1 Introduction............................................................................................................................. 11
   3.2 Implementation......................................................................................................................... 12
      3.2.1 Parameters/variables for power control ......................................................................... 12
      3.2.2 Implementation ............................................................................................................... 13
   3.3 Test result................................................................................................................................. 20
   3.4 Limitation ............................................................................................................................... 22
4 References ............................................................................................................................... 23
Revision history ............................................................................................................................. 24
Introduction

1. Introduction

The latest software release of the iMOTION™ motion control engine (MCE) includes script language support offering users the possibility to customize system-level functionalities without affecting the motor and PFC control algorithm. The script language option enables additional control loops along with the reading and writing of the motor control and PFC parameters and variables, which allows users to take advantage of the analog and digital resources that are not used by motor and/or PFC control. It is also scalable for any functional extension in the future.

In this document (AN2020-20), motor power calculation and constant-power control implemented by script code are introduced. Those functions are frequently used in the motor control application, especially fan, pump, compressor and vacuum cleaner applications.

Fan/pump applications often require constant-flow control. When the motor output power is constant, the motor speed reflects the air/flow resistance. One solution is to control the cubic volume of air flow by the closed loop based on the power feedback, which is closely associated to the air flow mass.

The power feedback is provided by internal information of the d/q-axis voltage and current, and described in detail in this application note.

All the functions in this document are implemented by the script language code, and tested with the EVAL-C101-A evaluation board, in which the IMC101T-T038 device is embedded. For more information about script language, please refer to the reference documents [2][5].
Power calculation and constant-power control
Implemented by iMOTION™ script language

Motor power calculation

2 Motor power calculation

There are several methods for calculating motor power (shaft power, inverter power, etc.). In this chapter, the power calculation is done by the inverter power; details about principles, implementation and test results are introduced.

2.1 Introduction

The basic scheme of power control in this example is based on the motor input power derived from the estimate of the motor voltage and motor current. In terms of controlling the target physical flow of fluid or air mass through the associated mechanical load (i.e. propeller, or impeller or blade) should be directly equivalent to the motor shaft power, which is the product of motor shaft torque and speed. Thus, the difference between the motor shaft power and motor input power, i.e., the permanent magnet motor loss, has to be considered for more accurate control. However, this difference can be negligible or ignored if the control aim is to establish a linear control of cubic volume of air/fluid mass, and not be concerned about the absolute value of power.

The formula below is the basic power calculation formula, which is derived by Ohm’s law and Joule’s law.

\[ P = VI \]

Where, \( P \) is the power, \( V \) is the voltage, and \( I \) is the current.

Similarly, the power calculation formula of the motor driver is

\[ P = \frac{3}{2} (V_d I_d + V_q I_q) \]

Where,

- \( I_d, I_q \): d-axis/q-axis stator current;
- \( V_d, V_q \): d-axis/q-axis stator voltage.

All the variables here are given in the rotor reference frame.

2.2 Implementation

In this section, the details on how to implement the formula by script language are introduced.

All the variables in the formula have been defined as registers in the MCE software; for further details, see Reference [2]. The corresponding registers in the MCE software are \( V_d, V_q, I_dFilt \) and \( I_qFilt \) (the \( I_d/I_q \) after the low pass filter).

Please note that the registers here do not represent real physical values, but have internal scaling. Therefore, the coefficient is needed to transfer the result to real world values. The script language only supports 32-bit integers, so all data in the floating-point format need to be transferred to fixed-point format.

It is recommended for users to learn some basic knowledge of the fixed-point number and Q format representations, which will help them to understand the code.

There are two variables defined in the script code, PowerScl and PwrRslt. The variable PowerScl needs to be calculated first, and the calculated power result is saved in the variable PwrRslt. The details are listed below.
## Power calculation and constant-power control

**Implemented by iMOTION™ script language**

### Motor power calculation

#### PowerScl

| Scaling or notation | \[
| INT \left( \frac{3}{2} \left( I_{\text{rated}} \cdot \sqrt{2} \cdot \frac{2048+4096}{\text{DCBusScl} \cdot 4307 \cdot \sqrt{3}} \cdot 100 \right) \right) \]
| Type | Variable, needs to be calculated offline.
| Description | PowerScl is the scaler to transfer the calculated results to the actual physical value. Where, 
\( I_{\text{rated}} \): the rated current of the motor that is typed in the MCEWizard
\( \text{DCBusScl} \): the DC bus feedback scaling whose unit is counts/V, which can be found in the MCEWizard, as shown in Figure 1.

For example: if \( I_{\text{rated}} = 1.09 \text{ A} \), \( \text{DCBusScl} = 112.81 \text{ counts/V} \), the PowerScl equals 2304.

More information about PowerScl is listed here. \( I_d \) or \( I_q \) \( \times \) \( \frac{I_{\text{rated}}}{4096} \) converts the \( I_d \) or \( I_q \)'s unit to the real physical unit A; \( V_d \) or \( V_q \) \( \times \) \( \frac{2048}{4307} \) converts the \( V_d \) or \( V_q \)'s unit to ADC counts, and dividing by \( \text{DCBusScl} \) converts its unit to the real physical unit V; 4096 is the coefficient to compensate right shifting (line 001 and 002 in the Code Listing 1); \( 3 \cdot \sqrt{2}/(\sqrt{3} \cdot 3) \) is the coefficient of the coordinate transformation; 100 magnifies the result 100 times so that the resolution of PwrRslt is 0.01 W.

#### PwrRslt

| Scaling or notation | 1 = 0.01 W
| Type | Variable
| Description | This variable represents the calculated power results, whose resolution is 0.01 W. It is calculated by the formula:
\[
PwrRslt = \frac{\text{PowerScl} \cdot (V_d \cdot I_{\text{d filt}} + V_q \cdot I_{\text{q filt}})}{2^{12}}\]

Where:
\( V_d, V_q \): registers of d-axis/q-axis stator voltage in the MCE;
\( I_{\text{d filt}}, I_{\text{q filt}} \): registers of filtered d-axis/q-axis stator current in the MCE.

Scaling is often used, since the script only supports 32-bit signed integers. If you use multiplication, you need to confirm that overflow will not occur. Scaling can be changed if the range and accuracy do not meet requirements.

**Note:** If DC bus compensation is disabled in the MCEWizard, the PwrRslt needs to be compensated by multiplying the coefficient \( \frac{V_{\text{dc Fil}}}{2048} \), where VdcFilt is an MCE register representing the \( V_{\text{dc}} \) voltage after LPF.
Power calculation and constant-power control
Implemented by iMOTION™ script language

Motor power calculation

Figure 1  DC bus feedback scaling

The calculated script codes are shown in Code Listing 1 and Code Listing 2. The details on how to configure the script code are introduced in Section 3.2.2 and Reference [2][9].

Code Listing 1  Power calculation code with DC bus compensation enabled

```c
001  DPwr = (IdFilt*Vd)>>12;  //Q12
002  QPwr = (IqFilt*Vq)>>12;  //Q12
003  TempVar = (PowerScl * (QPwr+DPwr))>>12;
005  //LPF Ts 1 ms (2.5 Hz - 3 db)
006  PwrMultiDEN= PwrMultiDEN + (TempVar - PwrRslt);
007  PwrRslt = PwrMultiDEN >> 6;
```

Code Listing 2  Power calculation code with DC bus compensation disabled

```c
001  DPwr = (IdFilt*Vd)>>12;  //Q12
002  QPwr = (IqFilt*Vq)>>12;  //Q12
003  TempVar = (PowerScl * (QPwr+DPwr))>>12;
005  //LPF Ts 1 ms (2.5 Hz - 3 db)
006  PwrMultiDEN= PwrMultiDEN + (TempVar - PwrRslt);
007  PwrRslt = PwrMultiDEN >> 6;
```

2.3  Test results

In this test example, a DC 24 V fan is used. The related motor parameters are as follows:

- Maximum DC volatege: 36 V
- Rated motor current $I_{\text{rated}}$: 1.2 A
- Rated motor speed: 3000 RPM
Power calculation and constant-power control
Implemented by iMOTION™ script language

Motor power calculation

The drive board is the internal evaluation board EVAL-C101T-A, which is the internal board, the DCBusScl of this board:

- DCBusScl: 112.81 counts/V

PowerScl needs to be calculated offline, as in the following example for a motor.

\[
\text{PowerScl} = \text{INT} \left( \frac{3}{2} \left( \frac{I_{\text{rated}} \cdot \sqrt{2}}{\text{DCBusScl}} \cdot \frac{2048 \cdot 4096}{4307 \cdot \sqrt{3} \cdot 100} \right) \right) = 2536
\]

In a high-speed region, when over-modulation occurs, the output voltage and the modulation index are no longer linearly correlated. An adjustment needs to be made. In this section, both test results are shown.

2.3.1 No over-modulation condition

Test results are shown in Table 1 to Table 3, and Figure 2 to Figure 4. The error is calculated by the formula below.

\[
\text{Error} = \frac{\text{Calculated power} - \text{Measured power}}{\text{Measured power}} \times 100\%
\]

<table>
<thead>
<tr>
<th>Motor speed (RPM)</th>
<th>Measured power (W)</th>
<th>Calculated power (W)</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.92</td>
<td>0.91</td>
<td>-1.1%</td>
</tr>
<tr>
<td>2000</td>
<td>3.96</td>
<td>4.05</td>
<td>2.3%</td>
</tr>
<tr>
<td>2500</td>
<td>7.02</td>
<td>7.18</td>
<td>2.2%</td>
</tr>
<tr>
<td>3000</td>
<td>11.56</td>
<td>11.65</td>
<td>0.8%</td>
</tr>
</tbody>
</table>

![Motor speed vs power with DC bus 20V](image)

Figure 2 Test results with DC bus at 20 V
Power calculation and constant-power control
Implemented by iMOTION™ script language

Motor power calculation

Table 2  Test results with DC bus at 25 V

<table>
<thead>
<tr>
<th>Motor speed (RPM)</th>
<th>Measured power (W)</th>
<th>Calculated power (W)</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.98</td>
<td>0.95</td>
<td>3.5%</td>
</tr>
<tr>
<td>2000</td>
<td>4.09</td>
<td>4.13</td>
<td>-0.9%</td>
</tr>
<tr>
<td>2500</td>
<td>7.13</td>
<td>7.25</td>
<td>-1.7%</td>
</tr>
<tr>
<td>3000</td>
<td>11.60</td>
<td>11.75</td>
<td>-1.3%</td>
</tr>
</tbody>
</table>

Figure 3  Test results with DC bus at 25 V

Table 3  Test results with DC voltage at 30 V

<table>
<thead>
<tr>
<th>Motor speed (RPM)</th>
<th>Measured power (W)</th>
<th>Calculated power (W)</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.96</td>
<td>0.95</td>
<td>-1.6%</td>
</tr>
<tr>
<td>2000</td>
<td>4.05</td>
<td>4.15</td>
<td>3.2%</td>
</tr>
<tr>
<td>2500</td>
<td>6.95</td>
<td>7.20</td>
<td>2.6%</td>
</tr>
<tr>
<td>3000</td>
<td>11.30</td>
<td>11.88</td>
<td>3.1%</td>
</tr>
</tbody>
</table>
2.3.2 Over-modulation condition

If over-modulation is enabled, the register VdqLim in the MCE will be set at 4974; otherwise its value is 4307. If over-modulation occurs, it means that modulation exceeds the linear range of the space vector pulse width modulation (SVPWM).

If the modulation index ranges from 4307 to 4974, the formula of Section 2.3 introduces an error, which is caused by over-modulation. As shown in Table 4 and Figure 5, the introduced error is 11% when the modulation index is 4974, so additional measures need to be implemented in this situation.

One method for adjusting the calculated power result is given here when over-modulation occurs. The method is to multiply a coefficient to the power results if over-modulation occurs. The coefficient can be calculated by the formula $\frac{4307}{\text{modulation index}}$. Table 5 and Figure 6 are the results after the coefficient has been adjusted. The error will be more acceptable.

There are different ways to calculate the modulation index in terms of DC compensation settings. If the DC bus compensation is enabled, the modulation index can be obtained by $\frac{\text{MotorVoltage} \times 2048}{\text{VdcFilt}}$. If the DC bus compensation is disabled, the modulation index is the same value as the \textit{MotorVoltage}. Here \textit{MotorVoltage} and \textit{VdcFilt} are the MCE registers \[2\] that represent the motor voltage and DC bus voltage, respectively.
Power calculation and constant-power control
Implemented by iMOTION™ script language

Motor power calculation

Table 4  Power calculation results under over-modulation conditions

<table>
<thead>
<tr>
<th>Modulation index</th>
<th>Motor speed (RPM)</th>
<th>Measured power (W)</th>
<th>Calculated power (W)</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1476*</td>
<td>1000</td>
<td>0.80</td>
<td>0.82</td>
<td>-3.1%</td>
</tr>
<tr>
<td>2991*</td>
<td>2000</td>
<td>3.67</td>
<td>3.75</td>
<td>-2.2%</td>
</tr>
<tr>
<td>3820*</td>
<td>2500</td>
<td>6.52</td>
<td>6.75</td>
<td>-3.6%</td>
</tr>
<tr>
<td>4307*</td>
<td>2823</td>
<td>9.01</td>
<td>9.06</td>
<td>-0.6%</td>
</tr>
<tr>
<td>4400</td>
<td>2853</td>
<td>9.38</td>
<td>9.53</td>
<td>-1.6%</td>
</tr>
<tr>
<td>4500</td>
<td>2888</td>
<td>9.68</td>
<td>9.87</td>
<td>-1.9%</td>
</tr>
<tr>
<td>4600</td>
<td>2895</td>
<td>9.89</td>
<td>10.22</td>
<td>-3.4%</td>
</tr>
<tr>
<td>4700</td>
<td>2910</td>
<td>9.97</td>
<td>10.51</td>
<td>-5.4%</td>
</tr>
<tr>
<td>4974</td>
<td>2942</td>
<td>10.14</td>
<td>11.26</td>
<td>-11.0%</td>
</tr>
</tbody>
</table>

Note: * at this value, no over-modulation occurs

Figure 5  Power calculation results under over-modulation conditions
Power calculation and constant-power control
Implemented by iMOTION™ script language

Motor power calculation

Table 5  Power calculation results after adjusting under the over-modulation conditions

<table>
<thead>
<tr>
<th>Modulation index</th>
<th>MotorSpeed (RPM)</th>
<th>Measured power (W)</th>
<th>Adjusted power (W)</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1476*</td>
<td>1000</td>
<td>0.80</td>
<td>0.82</td>
<td>-3.1%</td>
</tr>
<tr>
<td>2991*</td>
<td>2000</td>
<td>3.67</td>
<td>3.75</td>
<td>-2.2%</td>
</tr>
<tr>
<td>3820*</td>
<td>2500</td>
<td>6.52</td>
<td>6.75</td>
<td>-3.6%</td>
</tr>
<tr>
<td>4307*</td>
<td>2823</td>
<td>9.01</td>
<td>9.06</td>
<td>-0.6%</td>
</tr>
<tr>
<td>4400</td>
<td>2853</td>
<td>9.38</td>
<td>9.33</td>
<td>0.6%</td>
</tr>
<tr>
<td>4500</td>
<td>2888</td>
<td>9.68</td>
<td>9.45</td>
<td>2.4%</td>
</tr>
<tr>
<td>4600</td>
<td>2895</td>
<td>9.89</td>
<td>9.57</td>
<td>3.2%</td>
</tr>
<tr>
<td>4700</td>
<td>2910</td>
<td>9.97</td>
<td>9.63</td>
<td>3.4%</td>
</tr>
<tr>
<td>4974</td>
<td>2942</td>
<td>10.14</td>
<td>9.75</td>
<td>3.9%</td>
</tr>
</tbody>
</table>

Note: * at this value, no over-modulation occurs

Figure 6  Adjusted results under the over-modulation conditions
3 Constant-power control

3.1 Introduction

Some of the applications such as pumps and fans require the constant-power control or the constant air/water flow control algorithm. Instead of regulating speed or torque, it requires a control of the air flow or water flow to the desired value. In such applications, the constant-power control is essential, and this chapter shows an example of constant-power control which is implemented by the script language.

Figure 7 Constant-power control block diagram

Figure 7 shows the block diagram of the constant-power control. It utilizes FOC (field-oriented control) of the existing iMOTION™ firmware except for speed control. The script-based task is added, and performs the major control loop on top of the FOC, in lieu of the speed control loop, which regulates the power based on the power feedback.
### 3.2 Implementation

#### 3.2.1 Parameters/variables for power control

This section introduces the parameters and variables in the demo code. The explanation can help users to read the demo code more easily.

Variable PowerRslt in Section 2.2 is taken as the power reference, and a typical PI controller is used to implement the constant-power control algorithm. Five additional, defined variables are used in the algorithm, which are listed and described below. These variables need to be set before the code is compiled according to the user’s system configuration.

Note: If those parameters are defined as the global variables in the script code, they can be written/read by MCEDesigner online when tuning the motor.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxPowerMulti100</td>
<td>Motor’s maximum power (Watt)*100</td>
</tr>
<tr>
<td>Scaling or notation</td>
<td>Variable, needs to be calculated offline</td>
</tr>
<tr>
<td>Type</td>
<td>This variable sets the maximum power of the motor, its value equals motor maximum power (Watt)*100. For example, if the maximum power of the fan is 15 W, this variable is set to 1500. To be easily read, the PwrRslt is set as the Power (Watt)*100 format, but TargetPower is in Q12 format, so ( \frac{\text{PwrRslt} \times 2^{12}}{\text{MaxPowerMulti100}} ) will transfer the PwrRslt to Q12 format.</td>
</tr>
</tbody>
</table>
Power calculation and constant-power control
Implemented by iMOTION™ script language

**Constant-power control**

<table>
<thead>
<tr>
<th>TargetPower</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scaling or notation</strong></td>
</tr>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>KpPreg/KxPreg</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scaling or notation</strong></td>
</tr>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ErrLim</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scaling or notation</strong></td>
</tr>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
</tbody>
</table>

The demo code uses some motor parameters defined in the MCE. Users can refer to them in the Chapter “Register description” from Reference [2]. Here is a short description:

<table>
<thead>
<tr>
<th>CtrlModeSelect</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IdRef_Ext</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IqRef_Ext</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
</tr>
</tbody>
</table>

### 3.2.2 Implementation

The script program consists of the following parts [3]:

- **Set Commands**: Defines script-user version and script-task execution period
- **Functions**: Script code should be written inside four predefined functions including Script_Task0_init(), Script_Task0(), Script_Task1_init() and Script_Task1()
- **Variables and Parameters**
- **Statements and Expressions**: Each individual statement must end with a semicolon
Power calculation and constant-power control

Implemented by iMOTION™ script language

Constant-power control

- Comments: Starts with a slash asterisk /* and ends with an asterisk slash */ for multiple line comments, starts with double slash // for single line comments.

The script engine supports 2 independent tasks, namely, Task 0 and Task 1, running concurrently. The user-script program runs repeatedly on a configurable interval within Task 0 or Task 1 loop. The shortest possible execution period is 1 ms for Task 0, and 10 ms for Task 1. The execution period for each task can be configured to the multiples of 1 ms for Task 0 or 10 ms for Task 1 in the script code. Task 0 has higher priority than Task 1. The task execution period can be configured using “SCRIPT_TASK0_EXECUTION_PERIOD” and “SCRIPT_TASK1_EXECUTION_PERIOD” in the Set Commands section. Other parameters corresponding to the task execution period that need to be configured include “SCRIPT_TASK0_EXECUTION_STEP” and “SCRIPT_TASK1_EXECUTION_STEP.” These represent the number of lines to be executed for the shortest possible period of each task, which are 1 ms and 10 ms respectively.

All the functions in this demo code run in Task 0. Since the minimum execution period of Task 1 is 10 ms, long periods will cause too much delay. This will result in a poor dynamic response and even instability. In this demo code, for example, “SCRIPT_TASK0_EXECUTION_PERIOD” sets 2 and “SCRIPT_TASK0_EXECUTION_STEP” sets 50, which means that the execution period for this task is 2 ms, and 50 lines of instructions are executed every 1 ms.

There are some factors that need to be considered in order to determine those 2 settings.

1. Function minimum loop time

Loop time ensures that the function runs normally. Using the LPF design, for example, according to the Nyquist theorem, the sampling frequency needs to be at least twice as high as the frequency of interest to realize effective attenuation. Therefore, choosing the sampling period of 1 ms (1 kHz) for the LPF would be effective for the signal frequency ranging up to 500 Hz. If the sampling period is 2 ms, the effective frequency would be down to 250 Hz.

In this demo for example, the function works normally when the “SCRIPT_TASK0_EXECUTION_PERIOD” is set to 1~5. However, the lower the period setting, the faster the dynamic response. The period is set to 2 ms in the demo, whereas 1 ms will consume more CPU resources causing a higher CPU workload.

2. Numbers of instructions

The following table “Code Listing 3” shows a section of the compiled script object file (.ldf), in which the number of instructions for Task 0 is 30 (line 008). Therefore, the “SCRIPT_TASK0_EXECUTION_STEP” should be set to greater than 30 to ensure that the entire loop of Task 0 is completed during each minimum execution period of 1 ms. Since the “SCRIPT_TASK0_EXECUTION_PERIOD” in this demo is 2 ms, the “SCRIPT_TASK0_EXECUTION_STEP,” set to greater than half of 30, is feasible, which will enable the entire loop of Task 0 to be completed during the execution period of 2 ms. First the 1 ms script engine executes the specific lines of script, and the rest of the code is done in the next 1 ms. In the demo code, “SCRIPT_TASK0_EXECUTION_STEP” is set at 20.

<table>
<thead>
<tr>
<th>Code Listing 3</th>
<th>Section of compiled script object file for constant-power control</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>%-----------------------------------------------------------------</td>
</tr>
<tr>
<td>002</td>
<td>% Script Object File</td>
</tr>
<tr>
<td>003</td>
<td>%-----------------------------------------------------------------</td>
</tr>
<tr>
<td>004</td>
<td>% SCRIPT_USER_VERSION : 001.000</td>
</tr>
<tr>
<td>005</td>
<td>% DATE &amp; TIME : 23.08.2019 14:50:03</td>
</tr>
<tr>
<td>006</td>
<td>% SIZE : 445 Bytes</td>
</tr>
<tr>
<td>007</td>
<td>% Total Number of Lines : 190</td>
</tr>
<tr>
<td>008</td>
<td>% Task0 - Number of Instructions : 30</td>
</tr>
<tr>
<td>009</td>
<td>% Task1 - Number of Instructions : 0</td>
</tr>
</tbody>
</table>
3. CPU Load

The CPU resource is prioritized for the implementation of the motor and PFC control algorithm. The script engine is designed to take advantage of the spare CPU resources for executing the script program. The priority of executing the script program is lower than that of the motor and PFC control algorithm, so that the performance of the control algorithm will not be affected. However, CPU usage needs to be carefully evaluated before the script function is enabled. The “SCRIPT_TASK0_EXECUTION_PERIOD” should meet the time requirement of the application/function. The setting should also ensure that the CPU load does not exceed 95%. If the CPU overload occurs, the script function will not work as expected.

There are two methods used to evaluate the CPU load. Details can be found in Section 3 of Reference [5], where the most common method is introduced.

The CPU load status can be obtained by reading the system parameter ‘CPU Load’ using MCEDesigner [4]. Its minimum resolution is 0.1% [2]. The following Figure 9 shows that the register CPU Load is 682, which means the result is 68.2%. The more complicated the script code is, the higher the demanded CPU load. It is recommended that the actual CPU load does not exceed 95% to guarantee the script execution.

![Figure 9](image)

**Figure 9**  Reading register CPU load in the MCEDesigner

The script engine supports two kinds of 32-bit unsigned integer variables, which are global variables and local variables. The maximum number of global variables supported by the script engine is 30, and the maximum number of local variables for each task is 24. The intercommunication between Task 0 and Task 1 can be implemented by using global variables. Only global variables are accessible from the MCEDesigner or user UART interface. It is recommended that users define a variable as a global type if they intend to read its value during the running time using MCEDesigner [4].

The variables defined in lines 010-015 of the Code Listing 4 are global variables; instructions on how to import them into the MCEDesigner are listed below. When the global variables are imported into the MCEDesigner, they can be written/read online by users.
Power calculation and constant-power control
Implemented by iMOTION™ script language

Constant-power control

a) After the MCEWizard compiles the script code, a map file will be generated in the same folder as script code whose suffix is “.map”.

b) Open the MCEDesigner file whose suffix is “.irc” and import the map file by File→Import register map, as shown in Figure 10.

c) The defined global variables can then be found in the default group of both Read/Write registers in the motor1 window and source list of the trace setup in the system window. Detail are shown in Figure 11.

Note: The variables in the MCEDesigner are divided into two variables, which are low 16 bits with suffix “_L” and high 16 bits with suffix “_H”, since the MCEDesigner only support 16-bit integers.
Power calculation and constant-power control

Implemented by iMOTION™ script language

Constant-power control

Figure 11  Global variables in the MCEDesigner

The variables defined in the initialized function are local variables, which can only be read/written by the defined task itself. Lines 020–043 in the demo code are local variable definitions, and lines 045–061 are variable initialization. Both global and local variables can be initialized here, and both functions Script_Task0/1_init() are only called once after startup.

The function Script_Task0/1() are the script functions which are executed periodically by the setting period. In this demo, lines 071–078 are the power calculation with d/q axis voltage and current. The LPF in lines 076-077 are commented out, since the phase lag of LPF will affect the stability of the constant-power control if users only use the power calculation without power control. LPF can be added here to make the power result more constant. Lines 085-098 are the conditions that determine whether the motor starts or stops. DC bus voltage is chosen here to set the flag. Users can set the different conditions based on their own application requirements. Lines 099-145 are the codes of the constant-power control section. There are three sub-sections. The first section is the error calculation and limitation; the variable ErrLim is used to limit the power error, so the power can ramp up with the specific slope. The next section is the PI control of power controller; the integral value is limited by the variable IntegralLim. The last section is the control output, is limited; the reference torque of the current controller is set here.

Code Listing 4  Demo of the constant-power control

```c
001  #SET SCRIPT_USER_VERSION (1.00) /*Script version value should be 255.255*/
002  #SET SCRIPT_TASK0_EXECUTION_PERIOD (2) /*Script execution time for Task0 in ms, maximum value 65535*/
003  #SET SCRIPT_TASK1_EXECUTION_PERIOD (1) /*Script execution time for Task1 in 10mS, maximum value 65535*/
004  #SET SCRIPT_START_COMMAND (0x3) /* Start command, Task0 : Bit0, Task1 : Bit1; if bit is set, script executes after init */
005  #SET SCRIPT_TASK0_EXECUTION_STEP (20) /* Script Task0 step, This defines number of lines to be executed every 1 ms*/
```
Power calculation and constant-power control
 Implemented by iMOTION™ script language

Constant-power control

Code Listing 4  Demo of the constant-power control

```
#SET SCRIPT_TASK1_EXECUTION_STEP (30) /* Script Task1 step,  
   This defines number of lines to be executed every 10 ms*/

/*Global Variable Definition*/
int IqLPF,IdLPF;
int PowerRslt;//1 = 0.01 W
int TargetPower;//Q14,16384 means max power
int PowerCtrlModeEn;
int KpPreg,KxPreg;//Q12
int ViewVar;

/*Glabal Variable Definition*/
int IqLPF,IdLPF;  
int PowerRslt;//1 = 0.01 W
int TargetPower;//Q14,16384 means max power
int PowerCtrlModeEn;
int KpPreg,KxPreg;//Q12
int ViewVar;

/*Constant-power Control Variable*/
int TargetTrqTemp,TrqLimit;
int Power_Integral;
int MaxPowerMulti100;
int PowerQ12;
int PowerErr;
int IntegralLim;
int ErrLim;
int TimeCnt;

/*Constant-power Control Variable intializing*/
MaxPowerMulti100 = 1500;
ErrLim = 50;
```

Power calculation and constant-power control
 Implemented by iMOTION™ script language

Constant-power control

Code Listing 4  Demo of the constant-power control

```
#SET SCRIPT_TASK1_EXECUTION_STEP (30) /* Script Task1 step,  
   This defines number of lines to be executed every 10 ms*/

/*Global Variable Definition*/
int IqLPF,IdLPF;
int PowerRslt;//1 = 0.01 W
int TargetPower;//Q14,16384 means max power
int PowerCtrlModeEn;
int KpPreg,KxPreg;//Q12
int ViewVar;

/*Glabal Variable Definition*/
int IqLPF,IdLPF;  
int PowerRslt;//1 = 0.01 W
int TargetPower;//Q14,16384 means max power
int PowerCtrlModeEn;
int KpPreg,KxPreg;//Q12
int ViewVar;

/*Constant-power Control Variable*/
int TargetTrqTemp,TrqLimit;
int Power_Integral;
int MaxPowerMulti100;
int PowerQ12;
int PowerErr;
int IntegralLim;
int ErrLim;
int TimeCnt;

/*Constant-power Control Variable intializing*/
MaxPowerMulti100 = 1500;
ErrLim = 50;
```

Power calculation and constant-power control
 Implemented by iMOTION™ script language

Constant-power control

Code Listing 4  Demo of the constant-power control

```
#SET SCRIPT_TASK1_EXECUTION_STEP (30) /* Script Task1 step,  
   This defines number of lines to be executed every 10 ms*/

/*Global Variable Definition*/
int IqLPF,IdLPF;
int PowerRslt;//1 = 0.01 W
int TargetPower;//Q14,16384 means max power
int PowerCtrlModeEn;
int KpPreg,KxPreg;//Q12
int ViewVar;

/*Glabal Variable Definition*/
int IqLPF,IdLPF;  
int PowerRslt;//1 = 0.01 W
int TargetPower;//Q14,16384 means max power
int PowerCtrlModeEn;
int KpPreg,KxPreg;//Q12
int ViewVar;

/*Constant-power Control Variable*/
int TargetTrqTemp,TrqLimit;
int Power_Integral;
int MaxPowerMulti100;
int PowerQ12;
int PowerErr;
int IntegralLim;
int ErrLim;
int TimeCnt;

/*Constant-power Control Variable intializing*/
MaxPowerMulti100 = 1500;
ErrLim = 50;
```
Power calculation and constant-power control
Implemented by iMOTION™ script language

Code Listing 4  Demo of the constant-power control

```plaintext
055    TrqLimit = 4500;  //4096 means 100% Torque
056
057    //set zero
058    IdRef_Ext = 0;
059    TimeCnt = 0;
060    Power_Integral = 0;
061    PwrMultiDEN = 0;
062
063 }
064 /*******************************************************************/
065    /*Task0 script function*/
066    Script_Task0()
067    {
068        //========================================
069        // P = 3/2*(VdId+VqIq)
070        //========================================
071        DPwr = (IdFilt*Vd)>>12;  //Q12
072        QPwr = (IqFilt*Vq)>>12;  //Q12
073        TempVar = (PowerScl * (QPwr+DPwr))>>12;
074        //LPF Ts 1ms (2.5Hz -3db)
075        // PwrMultiDEN= PwrMultiDEN + (TempVar - PowerRslt);
076        PowerRslt = PwrMultiDEN >> 6;
077        PowerRslt = TempVar;
078
079    /*Constant-power Control*/
080    //monitor the DC bus voltage, if it is higher than the
081    //level, motor will start.
082        if (VdcFilt > 1500)
083            {
084                TimeCnt = TimeCnt + 1;
085                if(TimeCnt > 500)
086                    {
087                        Command = 1 ;  //start the motor
088                        PowerCtrlModeEn = 1;  //
089                        TimeCnt = 501;
090                    }
091                }
092        else
093            {
094                Command = 0;
095                TimeCnt = 0;
096                PowerCtrlModeEn=0;
097                Power_Integral=0;
098                
099            if(PowerCtrlModeEn == 1)
100                {
101                    //calculate the power error
102                    PowerQ12 = (PowerRslt<<12)/MaxPowerMulti100;  //to
103                    PowerErr = TargetPower-PowerQ12;//Q12
104                    //limit error input
105                    if(PowerErr>ErrLim)
```
Power calculation and constant-power control
Implemented by iMOTION™ script language

Constant-power control

Code Listing 4  Demo of the constant-power control

```
106 {  
107   PowerErr = ErrLim;  
108 }  
109 else  
110 {  
111   if(PowerErr < (0-ErrLim))  
112   {  
113     PowerErr = 0-ErrLim;  
114   }  
115 }  
116 //PI  
117 IntegralLim = 1<<24; //limit 2^24  
118 Power_Integral = Power_Integral + PowerErr*KxPreg;  
119 if(Power_Integral > IntegralLim)  
120 {  
121   Power_Integral = IntegralLim;  
122 }  
123 else  
124 {  
125   if (Power_Integral < 0 )  
126   {  
127     Power_Integral = 0;  
128   }  
129 }  
130 TargetTrqTemp =  
131   (PowerErr*KpPreg+Power_Integral)>>12; //convert to Q12  
132   //Limit trqref, you could also limit it by motorlim  
133 if (TargetTrqTemp > TrqLimit)  
134 {  
135   TargetTrqTemp = TrqLimit;  
136 }  
137 else  
138 {  
139   if(TargetTrqTemp<0)  
140   {  
141     TargetTrqTemp = 0;  
142   }  
143 }  
144 IqRef_Ext = TargetTrqTemp;  
145 }  
146 }
```

3.3 Test result

The function of the constant-power control algorithm is verified by the setup, which is the same as in Section 2.3. The step response from 7.5 W to 15 W by constant-power control is tested, and the waveform is shown in Figure 12 and Figure 13.

In these two figures, yellow is for the variable TargetPower, which represents the power reference; and green is the filtered result of the variable PowerRslt. The period of execution of the task is 2 ms.
Power calculation and constant-power control
Implemented by iMOTION™ script language

Constant-power control

Figure 12  Step response with KpReg = 1000 and KxPreg = 100 (transition time is 1.35 s)

Figure 13  Step response with KpReg = 3200 and KxPreg = 800 (transition time is 0.4 s)
3.4 Limitation

The following items need to be considered when applying the power control.

1. There is a limit in the applicable speed range. In general, it is inaccurate to measure power at a speed which is less than 5% of the maximum speed.

2. The type of power measurement and control needs to be understood and properly applied for each application, i.e., whether motor shaft power, input inverter power or output inverter power are to be controlled. If input power needs to be calculated or controlled, compensation is required by taking into account the inverter loss, the auxiliary power loss, the loss of passive components, etc. A motor loss needs to be considered if more accurate motor shaft power information is required. Motor efficiency could vary depending on the motor magnet type, structure of motor, etc. However, a percent loss can be incorporated if the absolute value becomes necessary.
Power calculation and constant-power control
Implemented by iMOTION™ script language

References

4 References


## Power calculation and constant-power control

Implemented by iMOTION™ script language

### Revision history

<table>
<thead>
<tr>
<th>Document version</th>
<th>Date of release</th>
<th>Description of changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>V 1.0</td>
<td>2020-11-24</td>
<td>Initial release</td>
</tr>
</tbody>
</table>
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).

Please note that this product is not qualified according to the AEC Q100 or AEC Q101 documents of the Automotive Electronics Council.

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 car reasonably be expected to result in personal injury.