Modulation Index Control using iMOTION™ Script Language

Modulation Index Control for Fan Application

About this document

Scope and purpose

Modulation index control is a kind of open loop control, which is widely used for fan application. This document describes how to realize modulation index control function by using iMOTION™ script launguage.

Intended audience

This document targets the customers who would like to control fan motor by modulation index control.

Table of contents

About this document **1**
Table of contents **1**
1 Introduction **2**
2 Implementation of Modulation Index Control **3**
2.1 Introduction 3
2.2 High Level Block Diagram 3
2.3 Implementation by Script Language 4
2.3.1 Capturing Vsp Analog Input 4
2.3.2 Capturing Frequency and Duty Input 5
2.3.3 Generation of Target Modulation Index 5
2.3.4 Ramp Block Implementation 7
2.3.5 Current Reference Definition 8
2.3.6 State Machine 9
2.3.7 Rotor Lock Protection 9
2.3.7.1 With Hall Sensor 9
2.3.7.2 Without Hall Sensor 10
2.4 Control Flowchart 11
3 Example **12**
3.1 Configuration for FW5.x (iMOTION™ Solution Designer) 12
3.2 FW1.3.7 (MCEWizard & Designer) 15
3.3 Operation Example 16
4 Appendix: Example Code **18**
4.1 Code Example for Vsp Analog Input (FW5.x) 18
4.2 Code Example for Vsp Analog Input (FW1.3.7) 20
4.3 Code Example for Frequency Input (FW1.3.7) 23
5 References **26**
Revision history **27**

1 Introduction

Modulation index control is widely used for fan application (air conditioner, HVAC system, air purifier, etc.). This is a kind of open-loop duty control.

Figure 1 depicts the system block diagram for modulation index control. There are five connections between host controller and motor system. Two power distributions (DCBUS and Vcc for gate driver), two signal input/output (control input and PGOUT) and one ground connection. Typically, Vcc=15 V, and power supply for iMOTION™ controller (3.3~5.0 V) is generated by voltage regulator in the motor system.

The host controller sends modulation index information by analog or digital signal. iMOTION™ device receives the signal and translates it into modulation index output to inverter. iMOTION™ device sends speed information to host controller via PGOUT signal. Host controller controls motor speed by using this PGOUT signal. In this case, speed control loop of iMOTION™ is not used, and motor speed depends on the load applied to the motor.

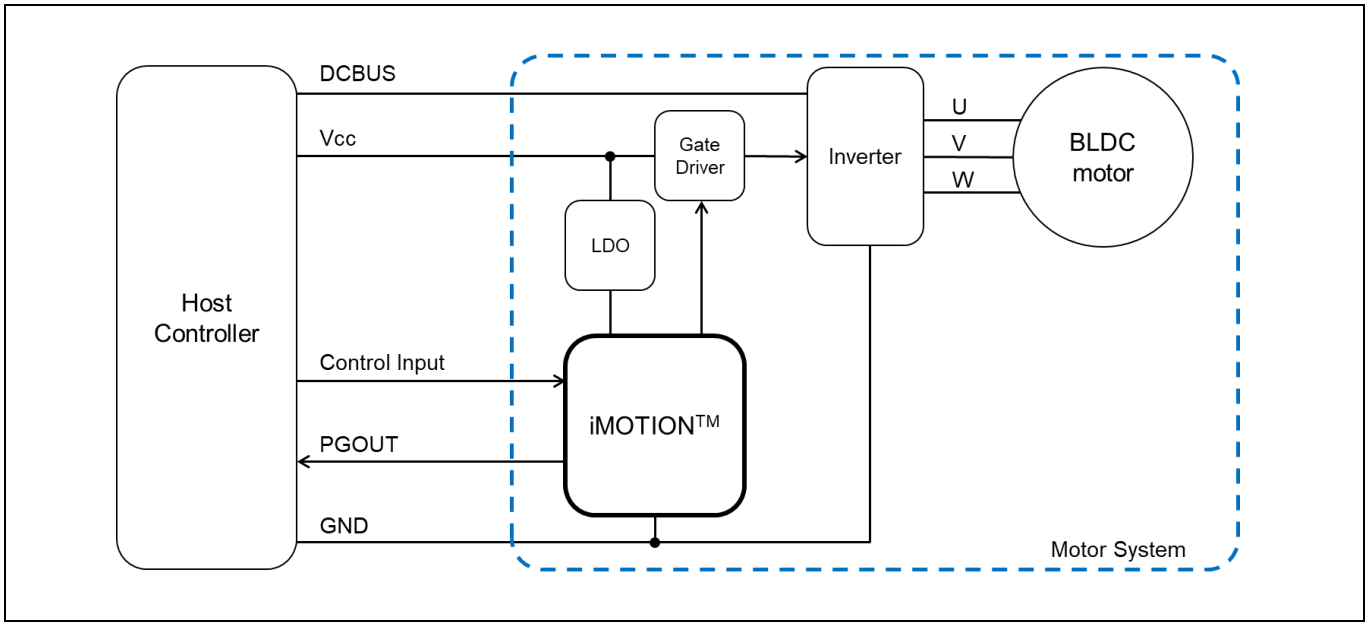


Figure 1 System Block Diagram for Modulation Index Control

2 Implementation of Modulation Index Control

2.1 Introduction

Standard control block diagram of MCE (motion control engine) is depicted in Figure 2. This is a well-established cascaded control structure with an outer speed loop and inner current control loop. Speed reference is input by external input signals (analog Vsp voltage, frequency, ducy and UART communication) through ramp block[1]**Error! Reference source not found.**.

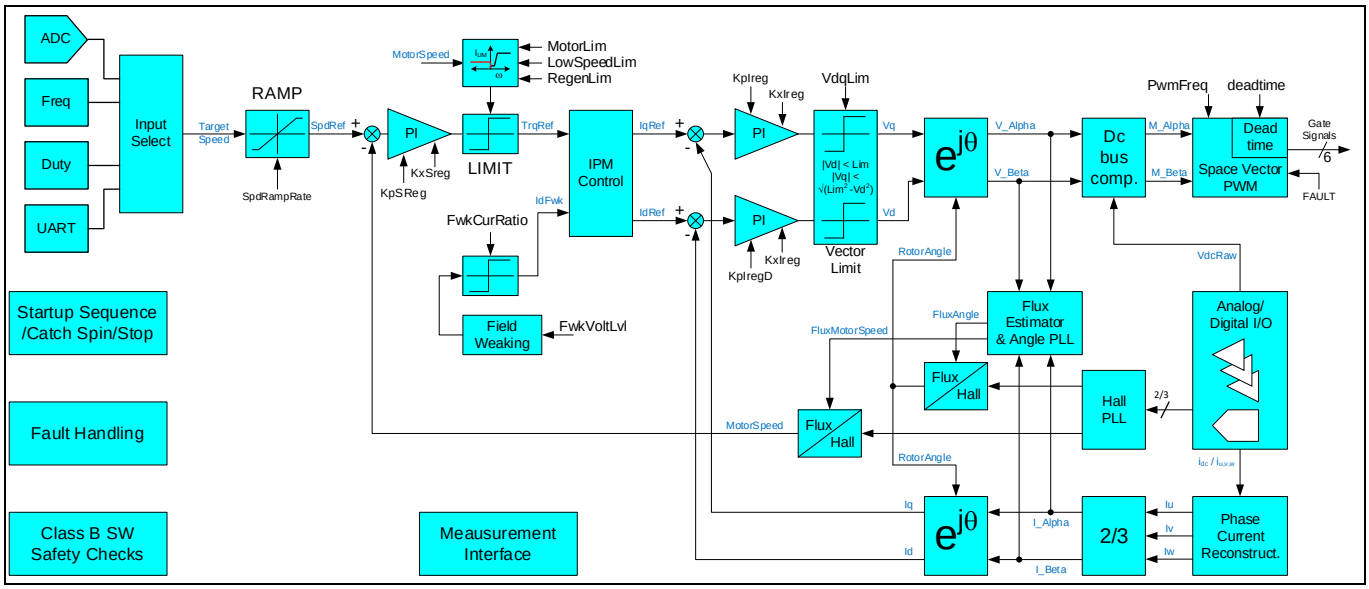


Figure 2 Control Block Diagram of MCE with Speed Control Loop

In modulation index control scheme, speed control is done by host controller by using PGOUT signals from iMOTION™ device, so speed control loop is not used.

In speed control, speed reference is SpdRef, which is generated in ramp block by TargetSpeed and SpdRampRate. For modulation index control, reference of modulation index should be input to the system. In the following sections, design concept for modulation index control will be described.

2.2 High Level Block Diagram

Figure 3 depicts control block diagram for modulation index control. Compared to Figure 2, outer speed control loop does not exist, and reference of modulation index and current are generated in the input block implemented by script language of iMOTION™ function. In order to use this control scheme, it is necessary to use current control instead of speed control (CtrlModeSelect = 1).

Script sets reference of q-axis current (I_qRef_Ext) as MotorLim variable, and reference of d-axis current (I_dRef_Ext) as 0. So current regulator tries to generate q-axis voltage (V_q) and d-axis voltage (V_d) so that motor current becomes MotorLim value. After current regulator, V_q and V_d are limited by VdqLim variable in vector limit block. Then output voltage is equal when VdqLim value is small enough, and saturated by current limit (MotorLim) when VdqLim is large enough. Script function sets VdqLim as reference of modulation index.

As for external reference of modulatrion index control input, Vsp analog, frequency and duty input are applicable for this control scheme.

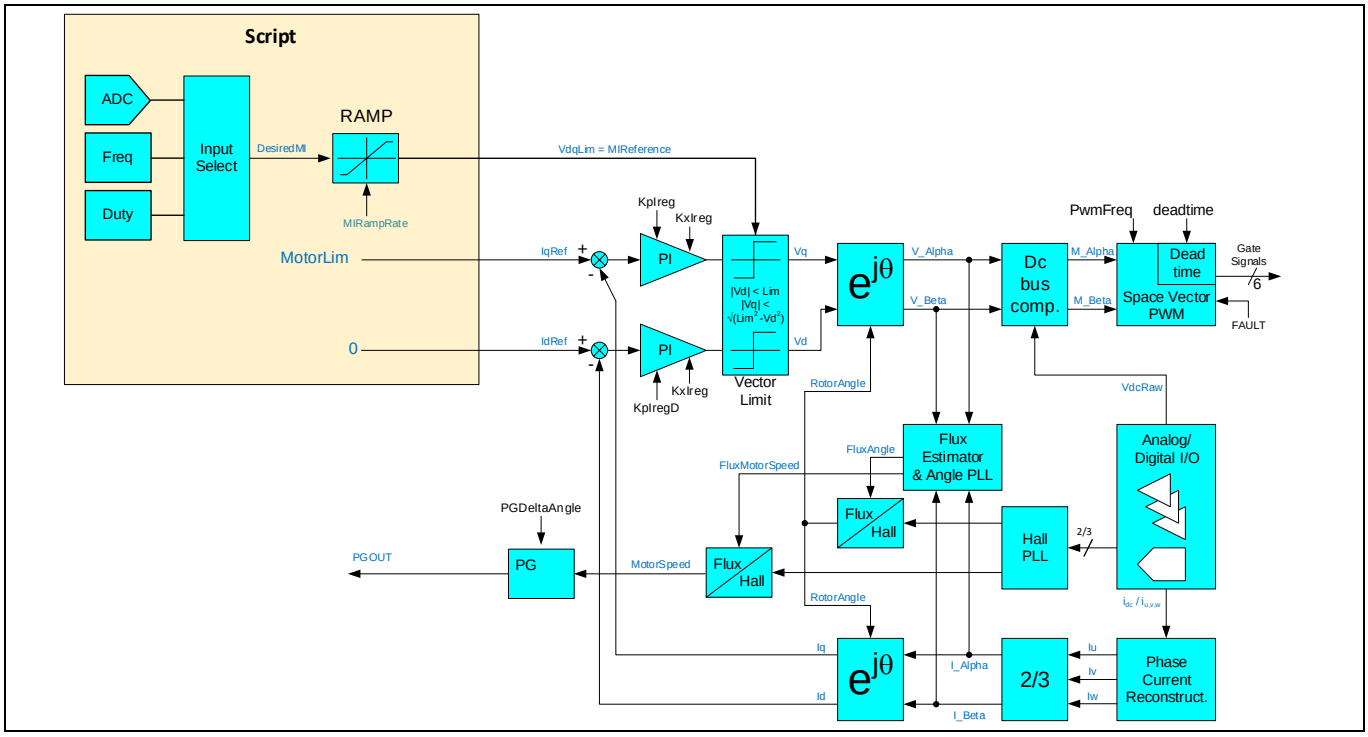


Figure 3 Control Block Diagram for Modulation index Control

2.3 Implementation by Script Language

2.3.1 Capturing Vsp Analog Input

Code Listing 1 shows example code to capture Vsp analog input. In this sample code, Vsp analog input value is acquired to read ADC[0] value. Low pass filter is applied to the acquired analog value. Global variable `VSPFilt` can be used in the script as VSP analog input value.

Sampling period of Vsp analog $T_s = 1\ ms$ by MCE. In this implementation, $\alpha = \frac{2^6-1}{2^6} = 0.984$, and filter's time constant $\tau = -\frac{T_s}{Ln(\alpha)} = 63.5\ ms$. For detail of low pass filter, please refer to **Error! Reference source not found.**.

Code Listing 1 Input Block Implementation for FW5.x

```
001   uint16_t VSPFilt;
002   Script_Task0_init()
003   {
004     int32_t VSPMultiplyDEN;
005     VSPMultiplyDEN = 0;
006   }
007   Script_Task0()
008   {
009     VSPMultiplyDEN = VSPMultiplyDEN + FB_ADC.adc_result[0];
010     VSPFilt = VSPMultiplyDEN >> 6;
011   }
```

Example code in Code Listing 1 is for FW5.x. For FW1.3.7, please convert "`int32_t`" and "`uint16_t`" to "`int`", and "`FB_ADC.adc_result[0]`" to "`ADC_Result0`".

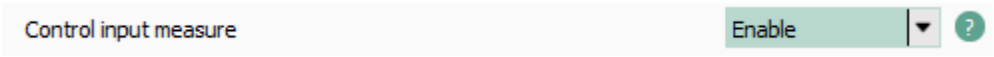
2.3.2 Capturing Frequency and Duty Input

Since frequency and duty inputs are updated at every 10 ms, task1 (10 ms execution period) is used to capture these inputs. Captured value is stored in the variable shown in Table 1.

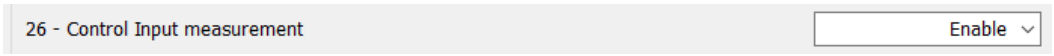
Table 1 Variable Name for Control Input

Input	FW5.x (Solution Designer)	FW1.3.7 (MCEWizard)
Frequency	`FB_CONTROLINPUT.FrequencyInput`	`ControlFreq`
Duty	`FB_CONTROLINPUT.DutyInput`	`ControlDuty`

In order to use this function, it is necessary to enable Control Input Measurement in Solution Designer or MCEWizard. In Solution Designer, please select "Enable" in Motor Control Configuration → Application → Control Input Measure in Configuration Wizard.



In MCEWizard, please select "Enable" in Q26 in Advanced Mode.



2.3.3 Generation of Target Modulation Index

After capturing analog or digital input, it is necessary to convert the input value to target of modulation index, `Target_MI`. The relationship between the input and `Target_MI` is shown in Figure 4, which is similar to the relationship between the control input and `TargetSpeed` described in [1] and **Error! Reference source not found.**. In the case of Figure 4, x-axis is Vsp analog input, but similar discussion can be applied for frequency and duty input.

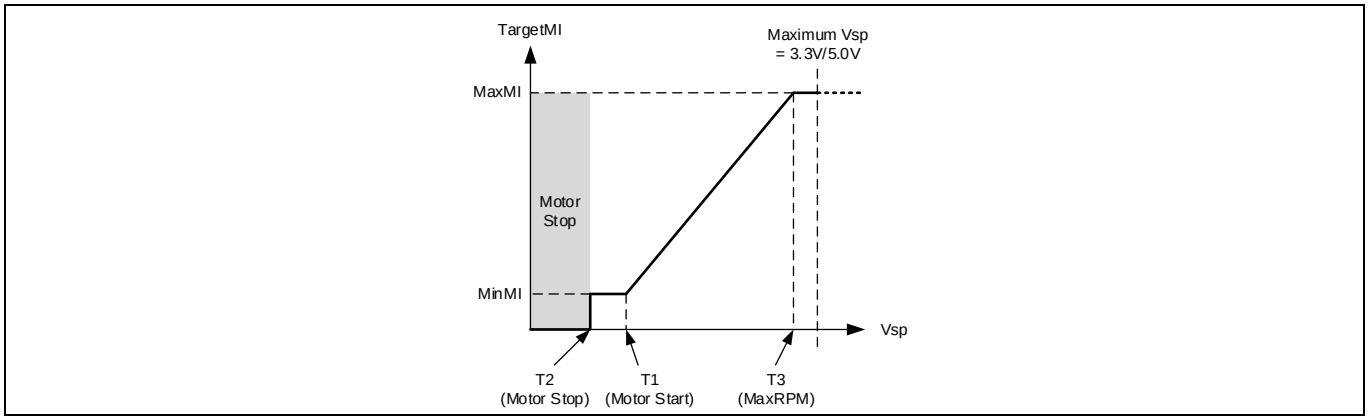


Figure 4 Target_MI Setting

When Vsp analog voltage is increased from 0 but smaller than `T1`, motor stops.

When Vsp analog voltage value reaches `T1`, `Target_MI` is set at minimum modulation index value (`Min_MI`) and motor starts rotation. `Target_MI` is linearly increased as Vsp is increased. When Vsp analog voltage value reaches `T3`, `Target_MI` reaches maximum value of modulation index (`Max_MI`), and is not changed if Vsp is larger than `T3`.

Implementation of Modulation Index Control

When Vsp analog voltage is decreased and reaches to T1, Target_MI is set at Min_MI and stay at this value until Vsp reaches to T2. If Vsp becomes smaller than T2, Target_MI is set at 0 and motor stops. In the script, T1, T2 and T3 should be specified as ADC counts. For example, if T1=1.5[V] and Vdd=3.3[V], T1 as count is $\frac{1.5}{3.3} \times 4095 = 1861$.

Min_MI should be determined so that the motor rotates smoothly at low speed.

If overmodulation is not allowed, Max_MI should be equal or smaller than 4307. If overmodulation is allowed, Max_MI should be equal or smaller than 4974.

Code Listing 2 shows the example code to convert analog or digital input to Target_MI for FW5.x. Since iMOTION™ Script Language can handle only integer type variable, pseudo fixed point operation is done by using bit shift operation (e.g. <<16). For FW1.3.7, please use "int" as variable declaration for all variables. In this example code, update of Target_MI is done in task1, which is executed at 10 ms period.

Code Listing 2 Target_MI Implementation for FW5.x

```
001       uint16_t Target_MI; /* Global variable */
002
003       Script_Task1_Init()
004       {
005         int32_t VSPMISlopeQ16;
006         uint16_t Max_MI, Min_MI;
007         uint16_t T1, T2, T3;
008
009         Max_MI = 4974;
010         Min_MI = 497;
011         T1 = 1720;  /* 42% of Vdd */
012         T2 = 1474;  /* 36% of Vdd */
013         T3 = 3890;  /* 95% of Vdd */
014         VSP2MISlopeQ16 = ((Max_MI - Min_MI) << 16) / (T3 - T1);
015       }
016       Script_Task1()
017       {
018         Target_MI = (((VSPFilt - T1) * VSP2MISlopeQ16) >> 16) +
    Min_MI;
019       }
```

Example code in Code Listing 2 is for FW5.x. For FW1.3.7, please convert "int32_t" and "uint16_t" to "int"

2.3.4 Ramp Block Implementation

In order to avoid sudden accleration or deceleration, ramp block for modulation index is implemented. The ramp block generates reference value of modulation index (`MIReference`) applied to the system, which is equal to `VdqLim`.

Variable `MIRampRate` is used for increment of MIReference in task1 period. If `MIRampRate`=10, modulation index is increased or decreased at 1000 counts/s because execution period of task1 is 10 ms.

If `MITarget` is larger than `MIRefence`, modulation index is increased (=acceleration). If `MITarget` is smaller than `MIReference`, modulation index is decreased (=deceleration).

Code Listing 3 shows the example code of ramp block implementation. This code is included in task1(`Script_Task1()`).

Code Listing 3 Ramp Block Implementation in Script_Task1() for FW5.x

```
001       if (MIReference < Target_MI)
002       {
003         MIReference = MIReference + MIRampRate;
004         if (MIReference > Target_MI)
005         {
006           MIReference = Target_MI;
007         }
008       }
009       else
010       {
011         if (MIReference > Target_MI)
012         {
013           MIReference = MIReference - MIRampRate;
014           if (MIReference < Target_MI)
015           {
016             MIReference = Target_MI;
017           }
018         }
019       }
020       FB_CURRENTREGULATOR.VdqLim = MIReference;
```

Example code in Code Listing 3 is for FW5.x. For FW1.3.7, please convert "`FB_CURRENTREGULATOR.VdqLim`" to "`VdqLim`" in line 020.

2.3.5 Current Reference Definition

As shown in Figure 3, reference for current regluator is generated by script. Code Listing 4 and Code Listing 5 show the example code of current reference definition. Reference of d-axis current (`IdRef_Ext`) is set at 0, and q-axis current (`IqRef_Ext`) at `MotorLim`. Since sign of `IqRef_Ext` depends on rotation direction, variable "`Direction`" is defined to specify rotation direction. For CW (clockwise) direction, Direction=1. For CCW (counter clockwise) direction, Direction=-1.

Code Listing 4 Current Reference Definition in Script_Task1() for FW5.x

```
001     APP_MOTOR0.IdRef_Ext = 0;
002     APP_MOTOR0.IqRef_Ext = Direction * FB_LIMIT_SPEED.MotorLim;
```

Code Listing 5 Current Reference Definition in Script_Task1() for FW1.3.7

```
001     IdRef_Ext = 0;
002     IqRef_Ext = Direction * MotorLim;
```

GPIO can be used to define Direction. Code Listing 6 shows the example code to define rotation direction in `Script_Task1_Init()`. In this case, GPIO[6] is used to define rotation direction. If GPIO[6] is pulled up, `Direction` = 1. If GPIO[6] is pulled down, `Direction` = -1. Fow FW1.3.7, please convert "`FB_GPIO.GPIO_Status.GPIO6`" to "`GPIO6_IN`".

Code Listing 6 Definition of rotation direction in Script_Task1_Init() for FW5.x

```
001     if (FB_GPIO.GPIO_Status.GPIO6 == 0)
002     {
003       Direction = -1;
004     }
005     else
006     {
007       Direction = 1;
008     }
```

2.3.6 State Machine

Global variable `MotorState` is defined to represent status of motor running condition. Table 2 shows the detail of the state machine. There are three states; MOTORSTOP (stop), MOTORRUN (run) and MOTORDECEL (deceleration). MOTORDECEL is necessary for smooth motor stop. Before motor stops, modulation index is gradually decreased to Min_MI then Command=0 is issued, in order to avoid vibration or acoursitc noise due to sudden deceleration.

Table 2 MotorState Description and Transition Event

State #	State	State Function	Transition Event	Next MotrState
0	MOTORSTOP	Motor stops	`VspFilt > T1`	MOTORRUN
1	MOTORRUN	Motor runs	`VspFilt < T2`	MOTORDECEL
2	MOTORDECEL	Motor decelerates before motor stops.	`MIReference < Min_MI`	MOTORSTOP

2.3.7 Rotor Lock Protection

MCE supports rotor lock protection, but this function cannot be used in current control mode (CtrlModeSelect = 1), because speed PI regulator output "`TrqRef`" is used to detect rotor lock.

For modulation index control, other method can be used to detect rotor lock protection. It depends on whether hall sensor is used or not.

2.3.7.1 With Hall Sensor

When hall sensors are used to detect rotor angle, a 'Hall Timeout' Fault can be used to detect rotor lock.

When the motor control state machine is in ‘MOTORRUN’ state, if the time interval between the two sequential Hall transition events is longer than a threshold T_{zf}, then it is considered as an Hall zero frequency fault, and accordingly, bit [2] of the variable ‘HallStatus’ is asserted. The threshold T_{zf} is calculated following this equation $T_{zf} = 4096 \times T_{PWM}$. Once the time interval between the two sequential Hall transition events is shorter than the threshold T_{zf}, this fault is automatically cleared.

The equivalent motor speed that would trigger Hall zero frequency fault consistently with 2 or 3 digital Hall sensor configurations can be calculated as follows:

$$\omega_{zf_3Hall}(rpm) = \frac{1}{4096 \times T_{PWM}} \times \frac{1}{6} \times \frac{60}{pole_pair}$$

If this Hall zero frequency fault lasts as long as $T_{HallTimeOut}$, then a ‘Hall Timeout’ Fault is confirmed. The threshold $T_{HallTimeOut}$ can be configured using the parameter ‘HallTimeoutPeriod’ following this equation: $T_{HallTimeOut} = HallTimeoutPeriod \times 10ms$.

2.3.7.2 Without Hall Sensor

When hall sensor is not used and sensorless control is applied, a 'Flux PLL Out-of-Control" Fault can be used to detect rotor lock.

When the Flux PLL is locked to the correct rotor angle, Flx_M, which represent the flux of the permanent magnet of the motor, should be a DC value normalized at 2048 counts. Instead, if the PLL is not locked to correct rotor angle, Flx_M becomes either unstable or its value is far off from 2048 counts. Flux PLL out-of-control protection is the mechanism designed to detect this fault condition.

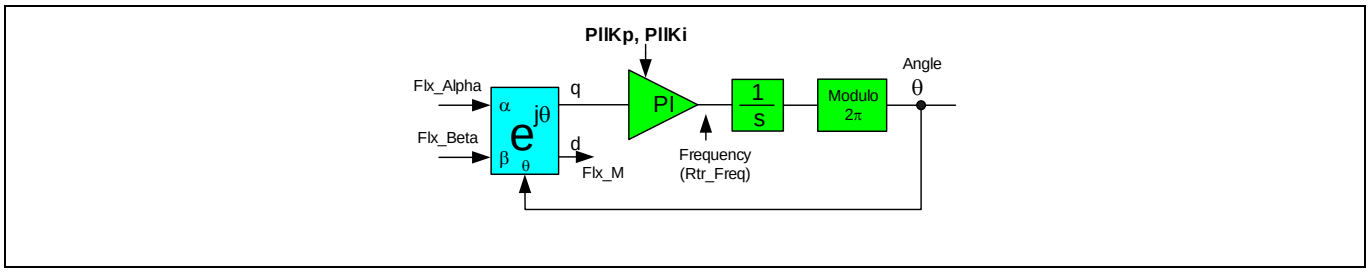


Figure 5 Simplified block diagram of a Flux PLL

The MCE keeps monitoring Flx_M, within certain time slot (configured by 'FluxFaultTime' parameter), if Flx_M value is below 512 or above 8192, and if this happens in 8 continuous time slots (each slot time is equal to FluxFaultTime/8), flux PLL is considered "out-of-control". See Figure 6 for details.

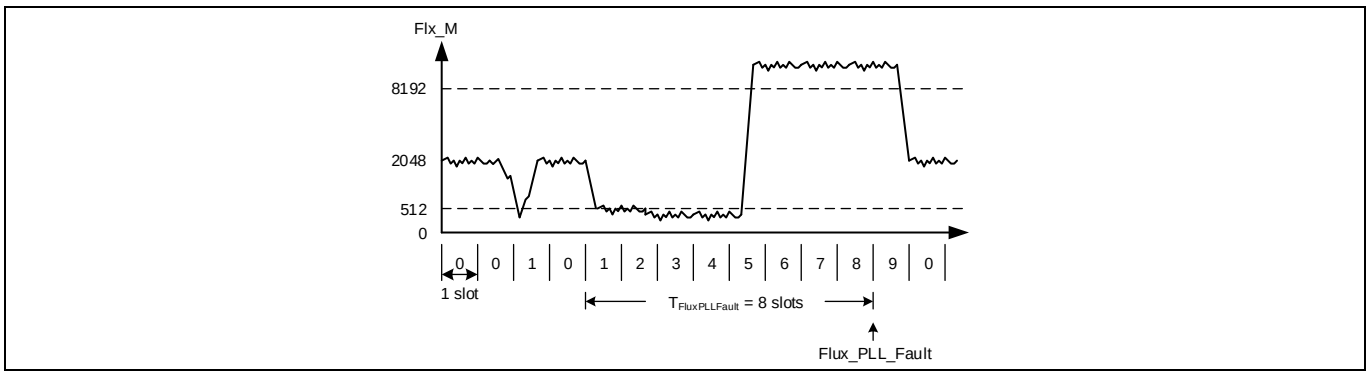


Figure 6 Flux PLL Out-of-Control Protection

If the Flux PLL out-of-control fault is confirmed, then it will be reported by setting the bit 4 in FaultFlags motor variable, and the motor speed loop gets reset. If the bit 4 in 'FaultEnable' motor dynamic parameter is set, then this fault will be reflected in 'SwFaults' motor variable and the motor state machine will shift to FAULT state causing the motor to stop running. If this bit is not set, then the corresponding bit in 'SwFaults' variable will be masked by 'FaultEnable' parameter, so that this fault will not be reflected in 'SwFaults' variable, and the motor state machine will not shift to FAULT state. This protection is also able to detect phase loss condition.

The PLL out-of-control fault response time can be configured by setting motor parameter 'FluxFaultTime'. The valid range of its value is from 0 to 65535. The value of 1 corresponds to 0.01 seconds. The default value is set to 800, which corresponds to a response time of 8 seconds.

2.4 Control Flowchart

Figure 7 shows overall control flowchart of modulation index control. State machine transition and reference of modulation index control are handled in this flow.

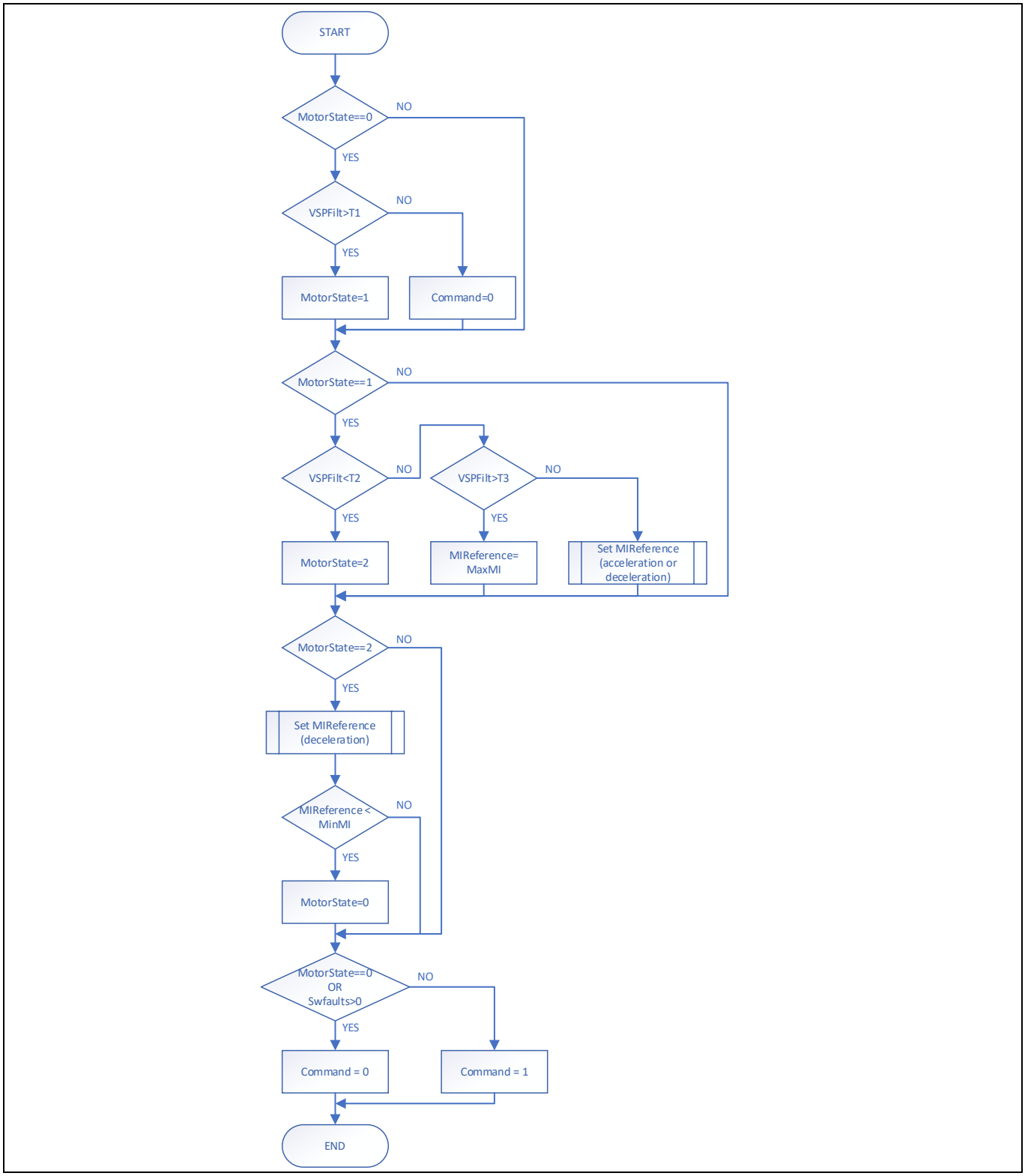


Figure 7 Modulation Index Control Flow Chart

3 Example

3.1 Configuration for FW5.x (iMOTION™ Solution Designer)

It is necessary to set system configuration by Configuration Wizard.

In "Motor Control Configuration", please select "Current Control" in Motor Control Modes.

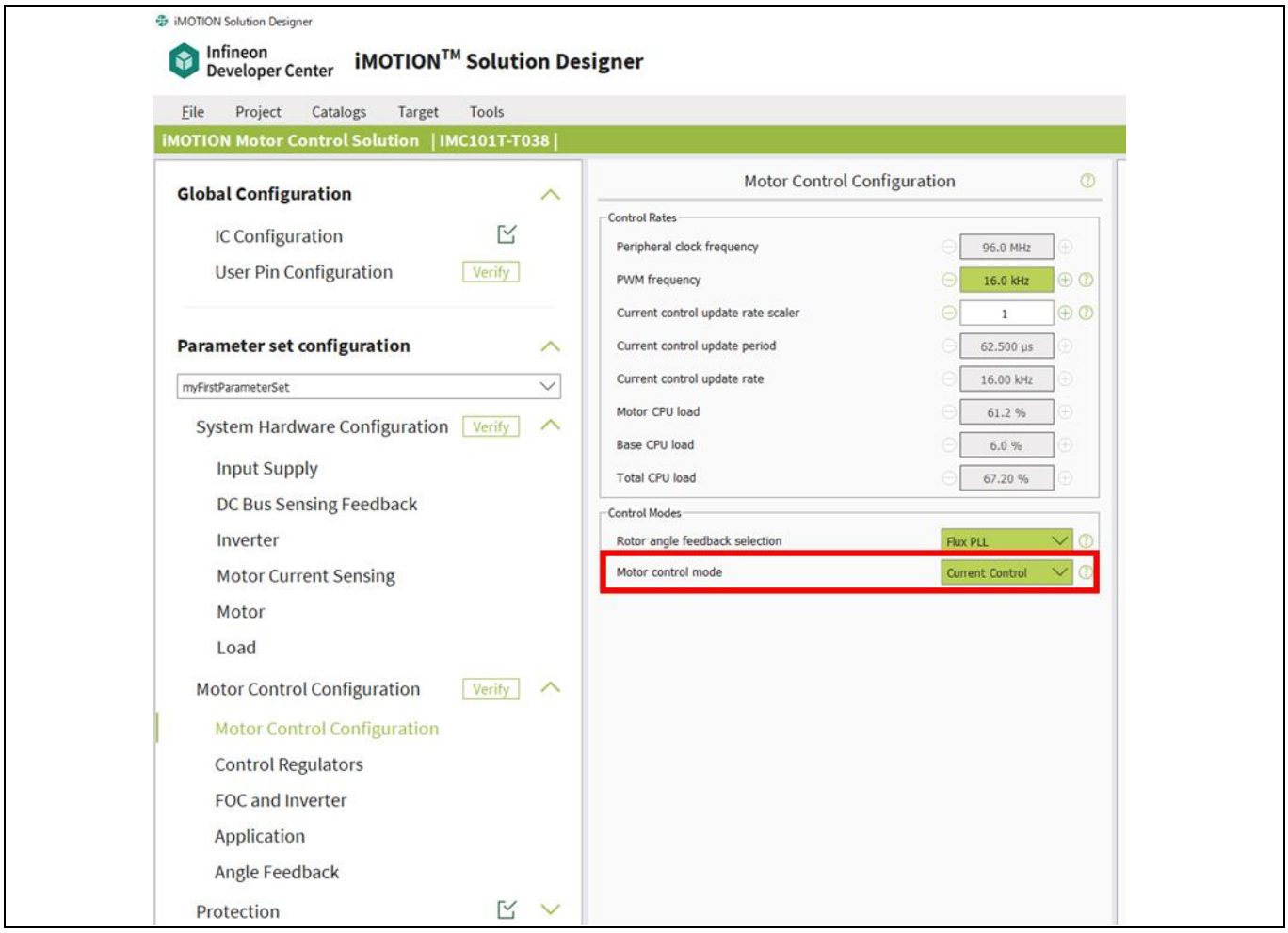


Figure 8 Motor Control Configuration

In "Application", please configure followings:

① Command input: UART

② Control Input Measure: Enable

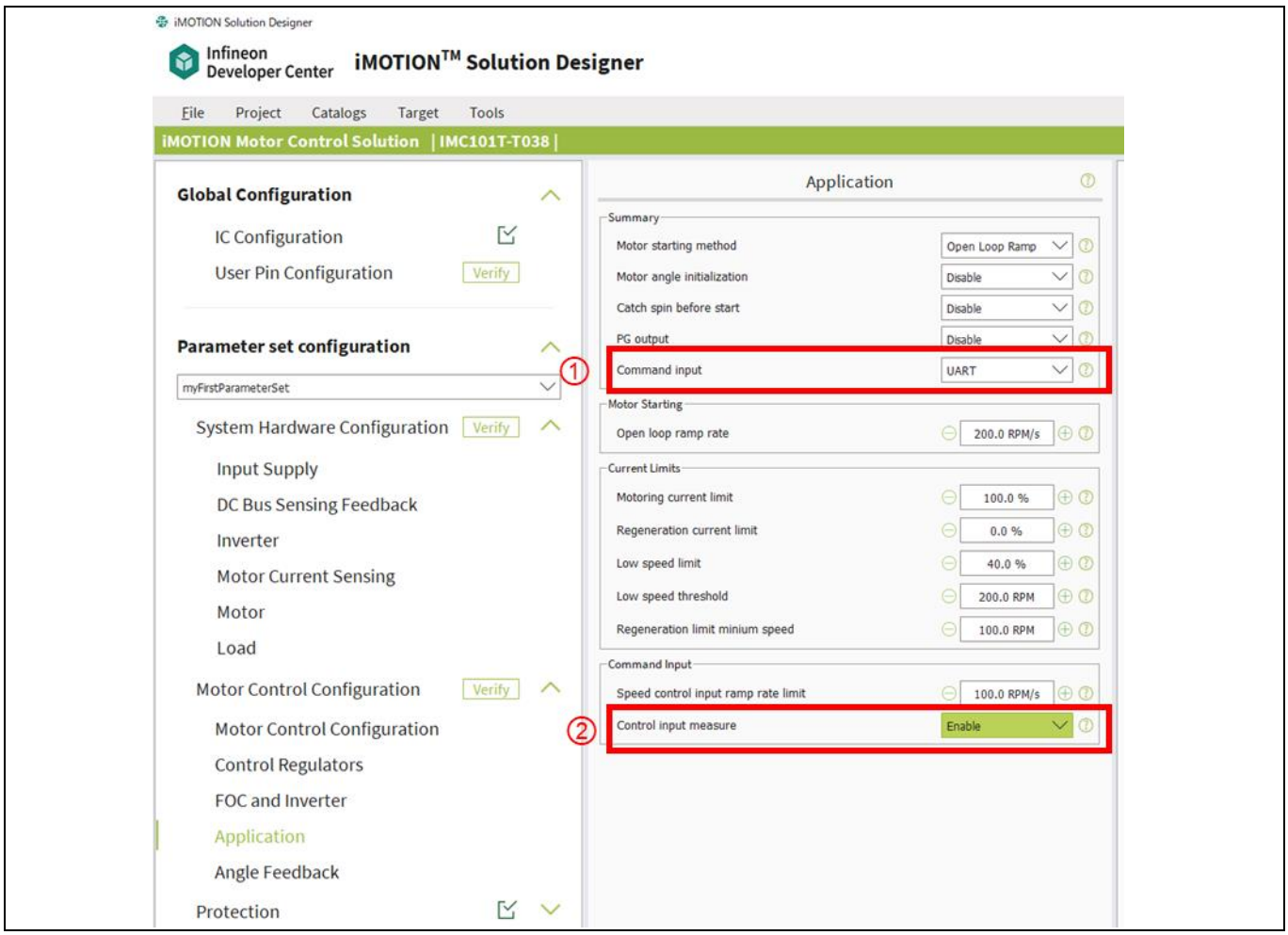


Figure 9 Application Setting

If rotation direction is defined by DIR pin setting, or VSP analog input is used, it is necessary to configure user pin usage. Please check DIR pin from GPIO list and set "Input" to use DIR pin. For VSP analog input, please check AIN[0]/VSP.

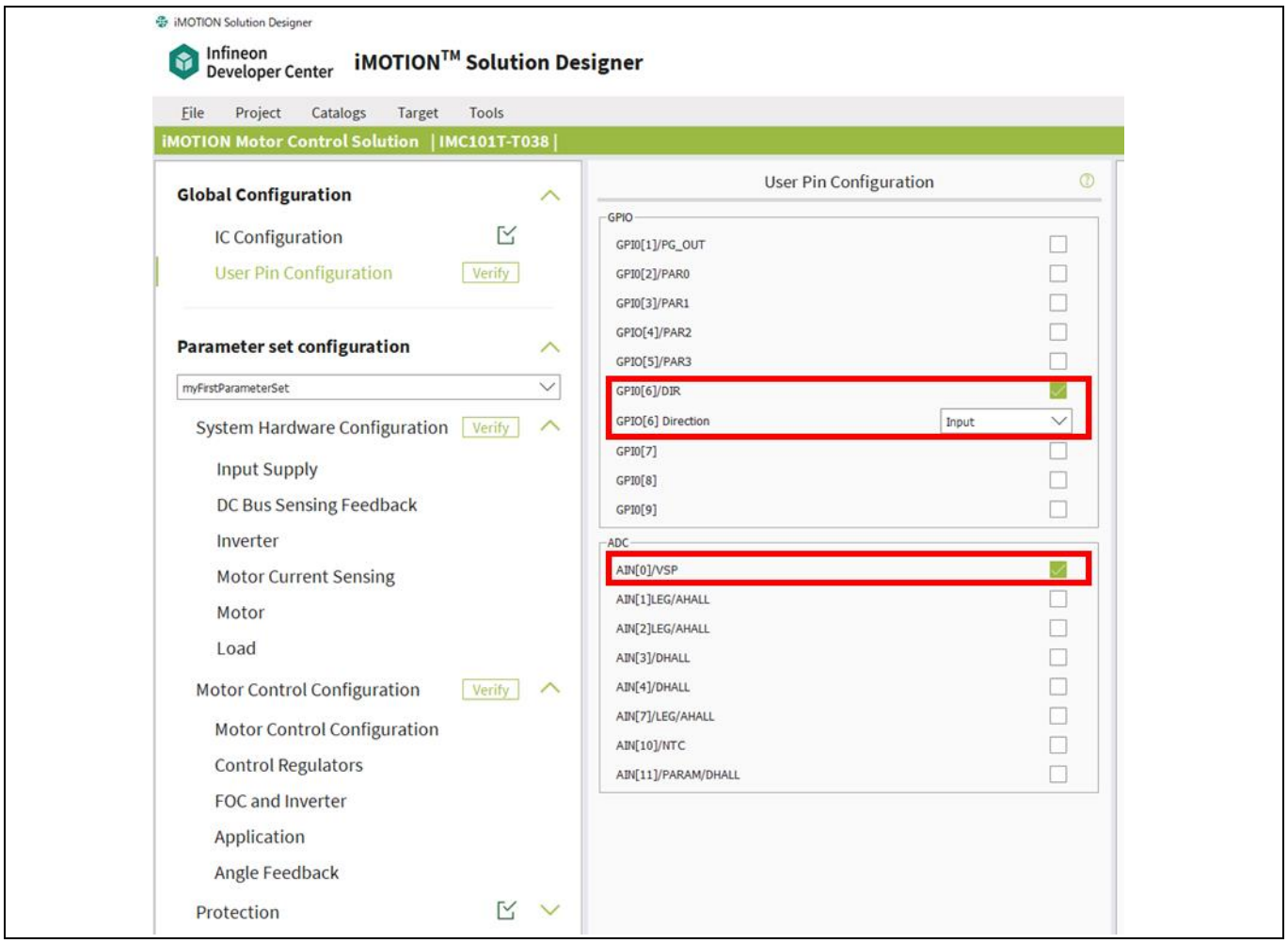


Figure 10 User Pin Configuration

3.2 FW1.3.7 (MCEWizard & Designer)

In "Options" Page, please configure following items as shown in Figure 11.

① Script Function Enable/Disable: Enable

② Application Control Interface: UART

③ Motor Control Mode: Current Control

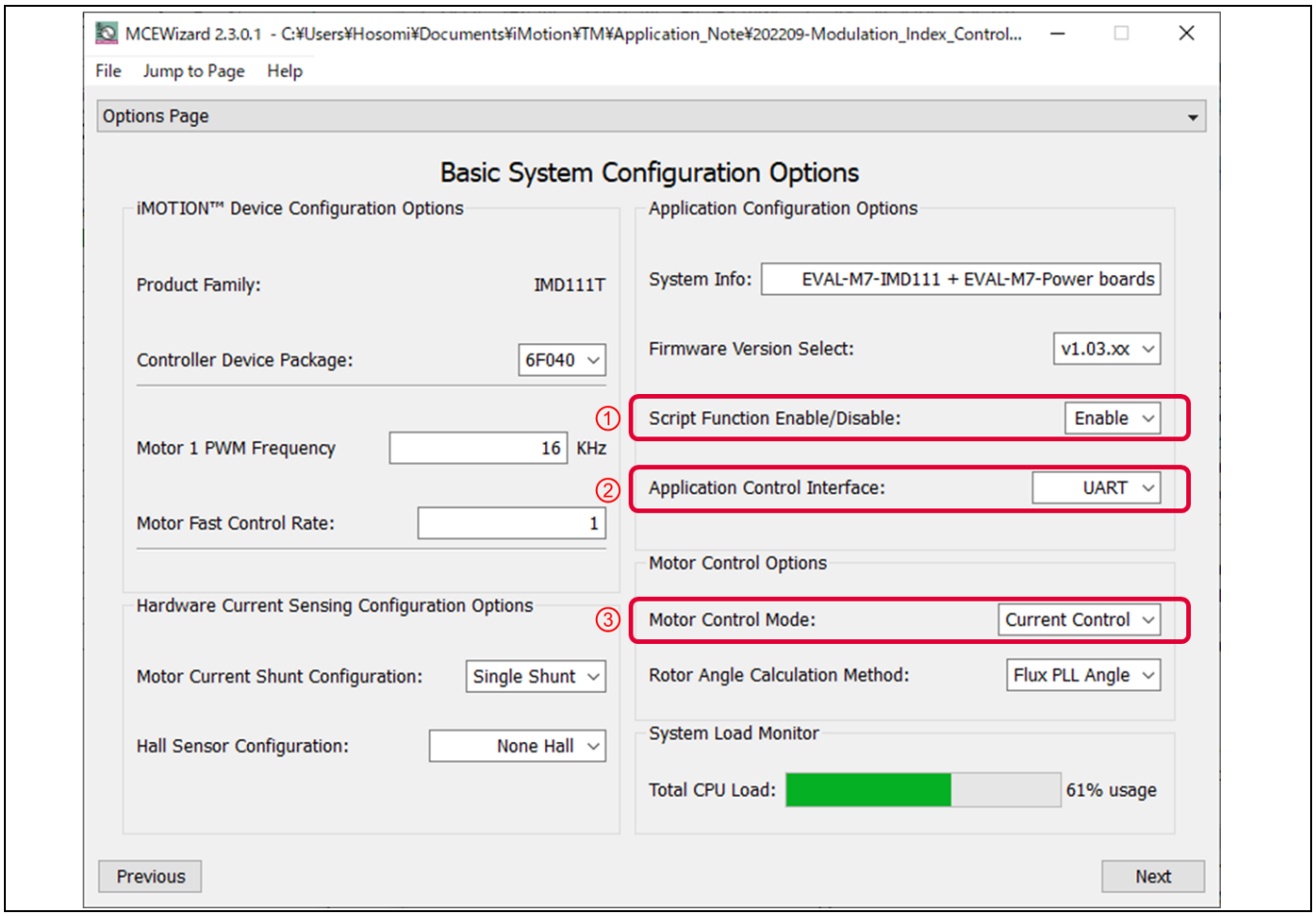


Figure 11 Settings in Options Page

In "Advanced Mode" page, please configure Control Input measurement as "Enable" as shown in Figure 12.

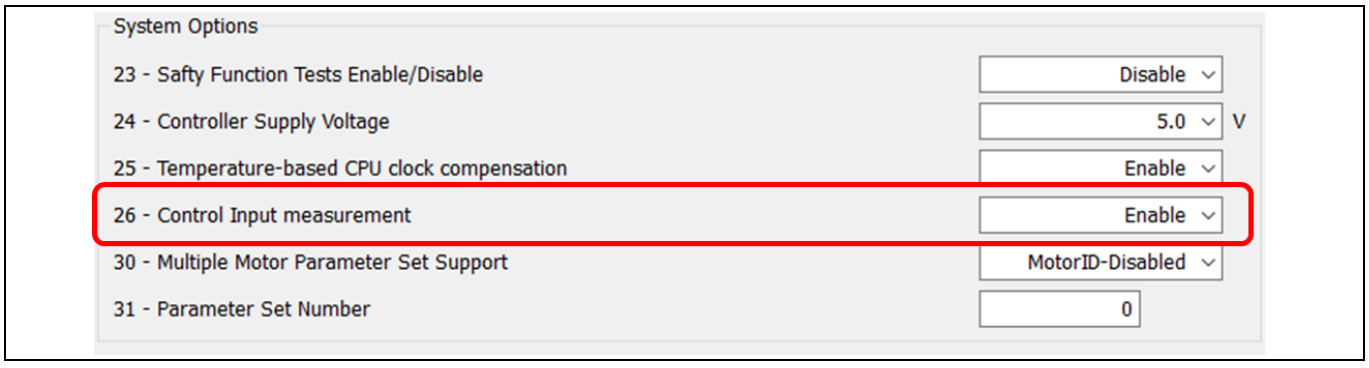


Figure 12 Control Input Measurement Setting in Advanced Mode – System

If rotation direction is defined by DIR pin setting, or VSP analog input is used, it is necessary to configure user pin usage in Script Edit Page/Script Configurator. Please set GPIO6/DIR as "Input" to use DIR pin. For VSP analog input, please check AIN[0]/VSP.

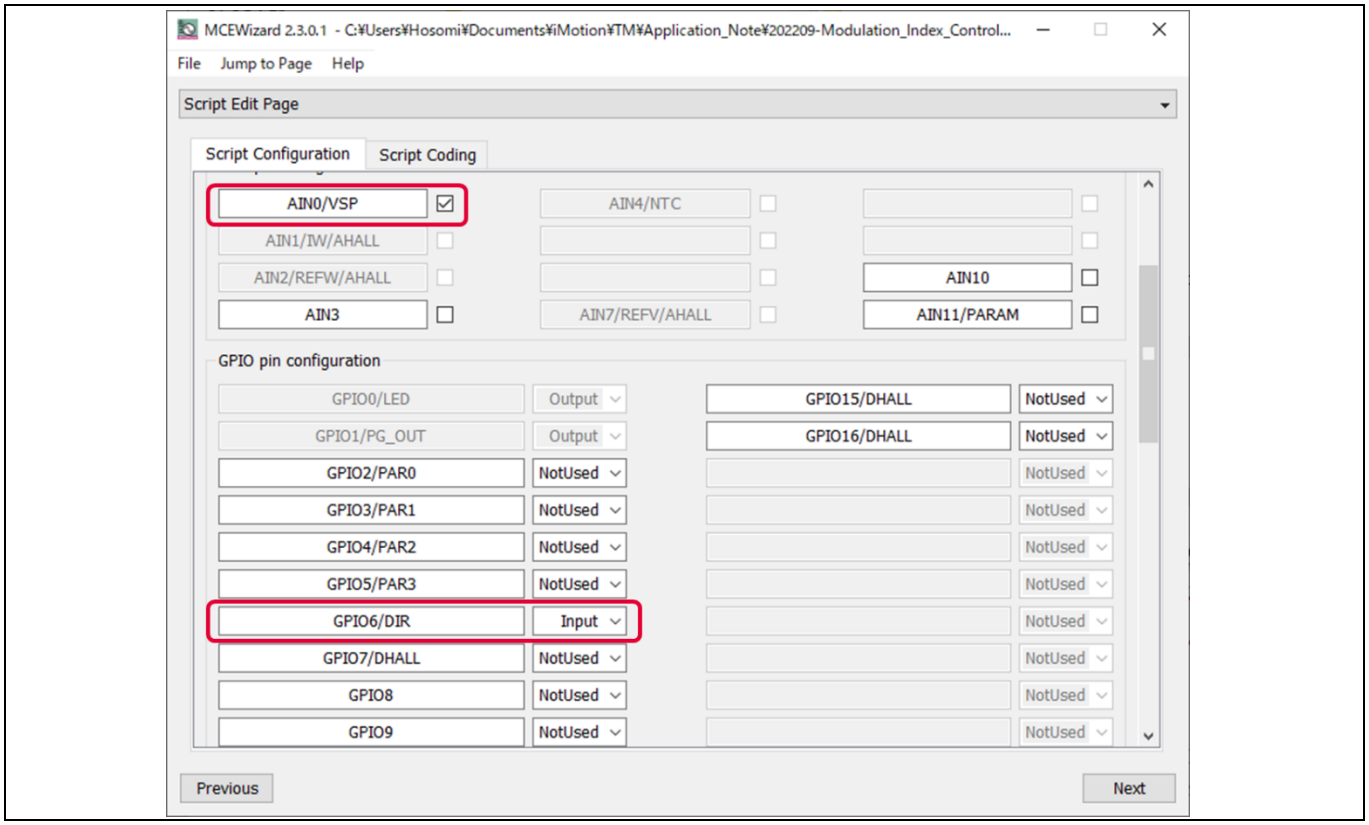


Figure 13 GPIO and ADC Pin Configuration in Script Edit Page

3.3 Operation Example

Figure 14 shows motor start up behavior when Vsp analog voltage is applied. When Vsp analog voltage reaches motor start threshold (T1=1720), MotorState changes from 0 to 1 and motor starts running. MIReference is gradually increased according to MI_RampRate, and reaches target modulation index value.

Figure 15 shows motor stop behavior when Vsp analog voltage is dropped. VSPFilt is gradually decreased, and MotorState is changed from 1 to 2 when VSPFilt reaches deceleration criteria (T2=1474). MIReference is decreased according to MI_DecelRate and without regard to VSPFilt. When MI_Reference reaches minimum modulation index value (Min_MI), then MI_Reference becomes zero and motor stops. This behavior can avoid accorstic noise or vibration due to sudden motor stop.

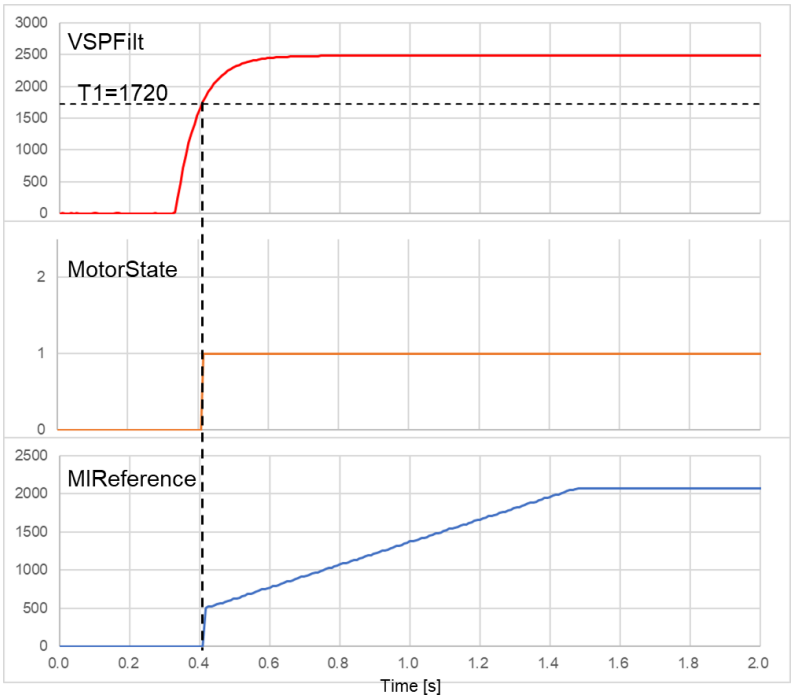


Figure 14 Start Up Behavior

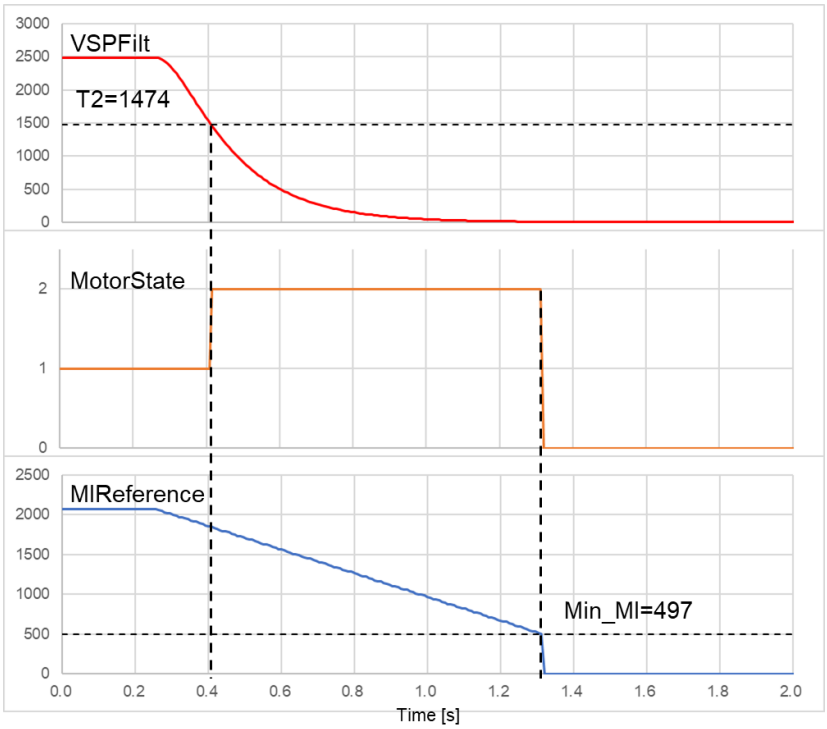


Figure 15 Stop Behavior

Code examples are shown in Code Listing 8 (Vsp analog input) and Code Listing 9 (Frequency input). For duty input, please modify lines 029 to 031, and 044 in Code Listing 9.

4 Appendix: Example Code

These example codes are for VSP analog input. If frequency or duty input is used, please modify the code as follows.

(1) Delete codes in task0 init function and task0 function.

(2) Add one line in the beginning of task1 function. It depends on FW version and input type. Please refer to Table 3

Table 3 Frequency and Duty Input

	FW5.x	FW1.3.7
Frequency	VSPFilt=FB_CONTROLINPUT.FrequencyInput	VSPFilt=ControlFreq
Duty	VSPFilt=FB_CONTROLINPUT.DutyInput	VSPFilt=ControlDuty

4.1 Code Example for Vsp Analog Input (FW5.x)

Code Listing 7 Code Example of Modulation Index Control by Vsp Analog Input (FW5.x)

```
001       uint8_t MotorState;
002       uint16_t int VSPFilt;
003       uint16_t int MIReference;
004       uint16_t int Target_MI;
005       uint16_t int T1;
006       uint16_t int T2;
007       uint16_t int T3;
008       uint16_t Min_MI;
009       uint16_t Max_MI;
010       uint16_t MI_RampRate;
011       uint16_t MI_DecelRate;
012       int8_t Direction;
013
014       /*Task0 init function*/
015       Script_Task0_init()
016       {
017         int VSPMultiplyDEN;
018         int ExecutionTimeCapture0;
019
020         VSPFilt = 0;
021         VSPMultiplyDEN = 0;
022       }
023       /*Task0 script function*/
024       Script_Task0()
025       {
026         VSPMultiplyDEN = VSPMultiplyDEN + (FB_ADC.adc_result[0] -
   VSPFilt);
027         VSPFilt = VSPMultiplyDEN >> 6;
028         ExecutionTimeTask0 = MCEOS.RunTimeCounter -
   ExecutionTimeCapture0;
029       }
030
031       /*Task1 init function*/
032       Script_Task1_init()
033       {
```

Code Listing 7 Code Example of Modulation Index Control by Vsp Analog Input (FW5.x)

```
034         int VSP2MISlopeQ16;            // MI Slope for VSP
035         int ExecutionTimeCapture;
036
037         Max_MI = 4974;
038         Min_MI = 497;
039         T1=1720;
040         T2=1474;
041         T3=3890;
042         MotorState = 0;
043         MI_RampRate = 10;
044         MI_DecelRate = 10;
045         FB_LIMIT_SPEED.MotorLim = 4096;
046         VSP2MISlopeQ16 = ((Max_MI - Min_MI) << 16) / (T3 - T1);
047       }
048       /*Task1 script function*/
049       Script_Task1()
050       {
051         APP_MOTOR0.IqRef_Ext = Direction * FB_LIMIT_SPEED.MotorLim;
052         APP_MOTOR0.IdRef_Ext = 0;
053
054         if (MotorState == 0)
055         {
056           if (VSPFilt > T1)
057           {
058             MIReference = Min_MI;
059             MotorState = 1;
060           }
061         }
062
063         if (MotorState == 1)
064         {
065           if (VSPFilt < T2)
066           {
067             MotorState = 2;
068           }
069           else
070           {
071             if (VSPFilt > T3)
072             {
073               Target_MI = Max_MI;
074             }
075             else
076             {
077               if (VSPFilt < T1)
078               {
079                 Target_MI = Min_MI;
080               }
081               else
082               {
083                 Target_MI = (((VSPFilt - T1) * VSP2MISlopeQ16) >>
    16) + Min_MI;
084                 if (Target_MI > Max_MI)
085                 {
086                   Target_MI = Max_MI;
```

Code Listing 7 Code Example of Modulation Index Control by Vsp Analog Input (FW5.x)

```
087                   }
088                 }
089               }
090               if (MIReference < Target_MI)
091               {
092                 MIReference = MIReference + MI_RampRate;
093                 if (MIReference > Target_MI)
094                 {
095                   MIReference = Target_MI;
096                 }
097               }
098               else
099               {
100                 if (MIReference > Target_MI)
101                 {
102                   MIReference = MIReference - MI_RampRate;
103                   if (MIReference < Target_MI)
104                   {
105                     MIReference = Target_MI;
106                   }
107                 }
108               }
109             }
110           }
111
112           if (MotorState == 2)
113           {
114             MIReference = MIReference - MI_DecelRate;
115             if (MIReference < Min_MI)
116             {
117               MotorState = 0;
118               MIReference = 0;
119             }
120           }
121
122           FB_CURRENTREGULATOR.VdqLim = MIReference;
123           if ((MotorState == 0) || (APP_MOTOR0.SwFaults > 0 ))
124           {
125             APP_MOTOR0.Command = 0;
126           }
127           else
128           {
129             APP_MOTOR0.Command = 1;
130           }
131         }
```

4.2 Code Example for Vsp Analog Input (FW1.3.7)

Code Listing 8 Code Example of Modulation Index Control by Vsp Analog Input (FW1.3.7)

```
int MotorState; /* 0: stop, 1: run, 2: deceleration */
int VSPFilt; /* Filtered Vsp analog volgate */
int MIReference;
int Target_MI;
int T1, T2, T3;
int Min_MI, Max_MI;
int MI_RampRate
int MI_DecelRate;
int Direction;

/*Task0 init function*/
Script_Task0_init()
{
  int VSPMultiplyDEN;
  VSPFilt = 0;
}

/*Task0 script function*/
Script_Task0()
{
  VSPMultiplyDEN = VSPMultiplyDEN + (ADC_Result0 - VSPFilt);
  VSPFilt = VSPMultiplyDEN >> 6;
 }

/*Task1 init function*/
Script_Task1_init()
{
  int VSP2MISlopeQ16;

  Max_MI = 4974;
  Min_MI = 497;
  T1=1720; // 42%
  T2=1474; // 36%
  T3=3890; // 95%
  MotorState = 0;
  MI_RampRate = 10;
  MI_DecelRate = 10;
  MotorLim = 4096;
  Direction = 1;
  VSP2MISlopeQ16 = ((Max_MI - Min_MI) << 16) / (T3 - T1);
}

Script_Task1()
{
  IqRef_Ext =  Direction * MotorLim;
  IdRef_Ext = 0;

  if (MotorState == 0)
  {
    if (VSPFilt > T1)
    {
      MIReference = Min_MI;
      MotorState = 1;
    }
```

Code Listing 8 Code Example of Modulation Index Control by Vsp Analog Input (FW1.3.7)

```
055           }
056           if (MotorState == 1)
057           {
058             if (VSPFilt < T2)
059                {
060                  MotorState = 2;
061                }
062                else
063                {
064                  if (VSPFilt > T3)
065                {
066                Target_MI = Max_MI;
067              }
068              else
069              {
070                if (VSPFilt < T1)
071                {
072                  Target_MI = Min_MI;
073                }
074                else
075                {
076                  Target_MI = ((VSPFilt - T1) * VSP2MISlopeQ16) >> 16 +
   Min_MI;
077                  if (Target_MI > Max_MI)
078                  {
079                    Target_MI = Max_MI;
080                  }
081                }
082              }
083              if (MIReference < Target_MI)
084              {
085                MIReference = MIReference + MI_RampRate;
086                if (MIReference > Target_MI)
087                {
088                  MIReference = Target_MI;
089                }
090              }
091              else
092              {
093                if (MIReference > Target_MI)
094                {
095                  MIReference = MIReference - MI_RampRate;
096                  if (MIReference < Target_MI)
097                  {
098                    MIReference = Target_MI;
099                  }
100                }
101              }
102            }
103          }
104
105          if (MotorState == 2)
106          {
107            MIReference = MIReference - MI_DecelRate;
```

Code Listing 8 Code Example of Modulation Index Control by Vsp Analog Input (FW1.3.7)

```
  if (MIReference < Min_MI)
  {
    MotorState = 0;
    MIReference = 0;
    }
  }

  VdqLim = MIReference;

  if ((MotorState == 0) || (SwFaults > 0 ))
  {
    Command = 0;
  }
  else
  {
    Command = 1;
  }
}
```

4.3 Code Example for Frequency Input (FW1.3.7)

Code Listing 9 Code Example of Modulation Index Control by Frequency Input (FW1.3.7)

```
int MotorState; /* 0: stop, 1: run, 2: deceleration */
int VSPFilt; /* Filtered Vsp analog volgate */
int MIReference;
int Target_MI;
int T1, T2, T3;
int Min_MI, Max_MI;
int MI_RampRate
int MI_DecelRate;
int Direction;

/*Task0 init function*/
Script_Task0_init()
{
}

/*Task0 script function*/
Script_Task0()
{
}

/*Task1 init function*/
Script_Task1_init()
{
  int VSP2MISlopeQ16;

  VSPFilt = 0;
  Max_MI = 4974;
  Min_MI = 497;
  T1=2000; // 200 Hz
```

Code Listing 9 Code Example of Modulation Index Control by Frequency Input (FW1.3.7)

```
030           T2=1500; // 150 Hz
031           T3=9500; // 950 Hz
032           MotorState = 0;
033           MI_RampRate = 10;
034           MI_DecelRate = 10;
035           MotorLim = 4096;
036           Direction = 1;
037           VSP2MISlopeQ16 = ((Max_MI - Min_MI) << 16) / (T3 - T1);
038         }
039
040         Script_Task1()
041         {
042           IqRef_Ext = Direction * MotorLim;
043           IdRef_Ext = 0;
044           VSPFilt = ControlFreq;
045
046           if (MotorState == 0)
047           {
048             if (VSPFilt > T1)
049             {
050               MIReference = Min_MI;
051               MotorState = 1;
052             }
053           }
054           if (MotorState == 1)
055           {
056             if (VSPFilt < T2)
057                {
058                  MotorState = 2;
059                }
060                else
061                {
062                  if (VSPFilt > T3)
063                {
064                Target_MI = Max_MI;
065             }
066             else
067             {
068               if (VSPFilt < T1)
069                {
070                  Target_MI = Min_MI;
071                }
072                else
073                {
074                  Target_MI = ((VSPFilt - T1) * VSP2MISlopeQ16) >> 16 +
   Min_MI;
075                  if (Target_MI > Max_MI)
076                  {
077                    Target_MI = Max_MI;
078                  }
079                }
080             }
081             if (MIReference < Target_MI)
082             {
```

Code Listing 9 Code Example of Modulation Index Control by Frequency Input (FW1.3.7)

```
083               MIReference = MIReference + MI_RampRate;
084               if (MIReference > Target_MI)
085               {
086                 MIReference = Target_MI;
087               }
088             }
089             else
090             {
091               if (MIReference > Target_MI)
092               {
093                 MIReference = MIReference - MI_RampRate;
094                 if (MIReference < Target_MI)
095                 {
096                   MIReference = Target_MI;
097                 }
098               }
099             }
100           }
101         }
102
103         if (MotorState == 2)
104         {
105           MIReference = MIReference - MI_DecelRate;
106           if (MIReference < Min_MI)
107           {
108             MotorState = 0;
109             MIReference = 0;
110             }
111           }
112
113           VdqLim = MIReference;
114
115           if ((MotorState == 0) || (SwFaults > 0 ))
116           {
117             Command = 0;
118           }
119           else
120           {
121             Command = 1;
122           }
123         }
```

5 References

[1] iMOTION™ Motion Control Engine Functional Reference Manual

[2] iMOTION™ Motion Control engine Software Reference Manual

[3] AN2018-27 How to use iMOTION™ Script Language

[4] United States Patent US 11,557,998 B2

Revision history

Document version	Date of release	Description of changes
1.0	2023-12-25	First release

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

Edition 2023-12-25

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2024 Infineon Technologies AG. All Rights Reserved.

Do you have a question about this document?

Email: erratum@infineon.com

Document reference
AN2023-08

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.