Infrared Remote Control and Saving Last Speed Setting

About this document

Scope and purpose

This application note provides examples of how to use the iMOTION™ script language to realize infrared remote control and FLASH data storage functions on iMOTION™ devices with Motion Control Engine (MCE).

Intended audience

This document is intended for customers who would like to use iMOTION™ devices in home appliance applications.

Table of contents

1 Infrared (IR) Remote Control

1.1 Introduction

The Infrared (IR) remote control is a wireless, non-contact control technology with strong anti-interference ability, reliable information transmission, low power consumption, low cost, easy to realize, and other significant advantages. The IR remote is widely used by a variety of electronic equipment, especially household appliances.

The IR remote control has two parts: One is the transmitter part that uses an IR light emitting diode to emit modulated IR light waves; another is the IR receiver part that converts IR light from the IR transmitter into a corresponding electrical signal.

The FW5.1 or later release of the iMOTION™ motion control engine (MCE) provides the script engine, which supports IR remote control function that consists of a plug-in of the scripting engine and script APIs. The IR Interface supports the following protocols: NEC, NEC Extended, and RC5 Phillips. For the signal receiving pin, users can use pin RX0, RX1, or VSP, but not all devices support these three pins. If the IR transmitter supports these protocols and IR receiver connects to the correct pin, users can use the MCE to decode IR commands directly. For more information, users can refer to datasheets of relevant devices and the functional reference manual[1].

1.2 Overview

The hardware is equipped with an IR receiver circuit, which is used to receive IR signals and convert IR signals into pulse electrical signals. According to the defined protocol format, the firmware decodes the received IR commands. With the decoded IR command, users can call the corresponding IR APIs to get the variables and parameters required by the MCE using the script. Figure 1 shows the IR remote control layers. The physical layer will process the signal received from the IR transmitter and send it to the IR pin. The data link layer and network layer will decode the signal from the IR pin and define the address and command. The application layer is implemented by the script, allowing users to obtain the address and command information from the network layer. For more information about the hardware, users can refer to the user guide[3].

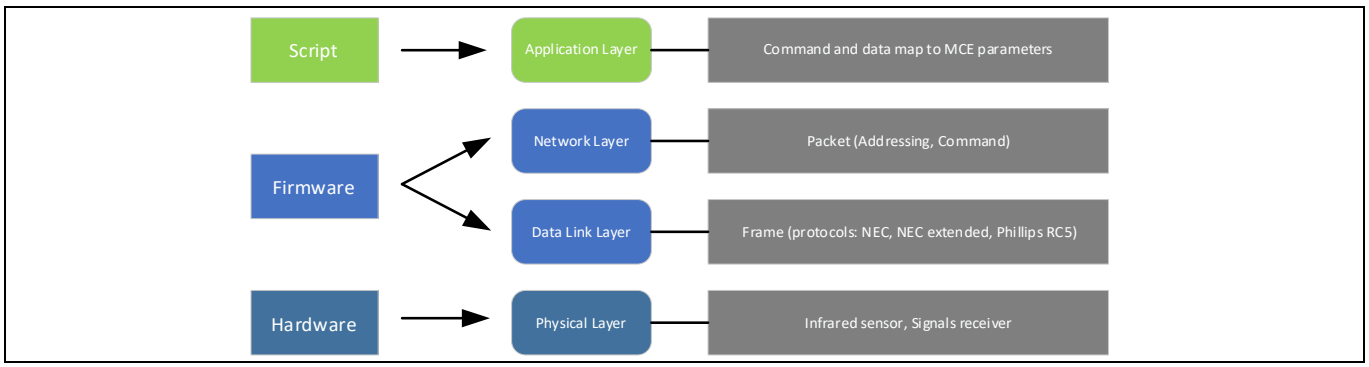


Figure 1 IR Remote Control Layers

1.3 IR Protocol

The latest MCE supports three IR protocols: NEC, NEC Extended, and RC5 Phillips. This section introduces the characteristics of the three protocols.

1.3.1 NEC Protocol

The NEC IR transmission protocol uses pulse distance encoding of the message bits. Each pulse burst (RC transmitter ON) is 562.5µs in length, at a carrier frequency of 38kHz. Logical bits are transmitted as follows:

- Logical '0' – a 562.5µs pulse burst followed by a 562.5µs space, with a total transmit time of 1.125ms.
- Logical '1' – a 562.5µs pulse burst followed by a 1.6875ms space, with a total transmit time of 2.25ms.

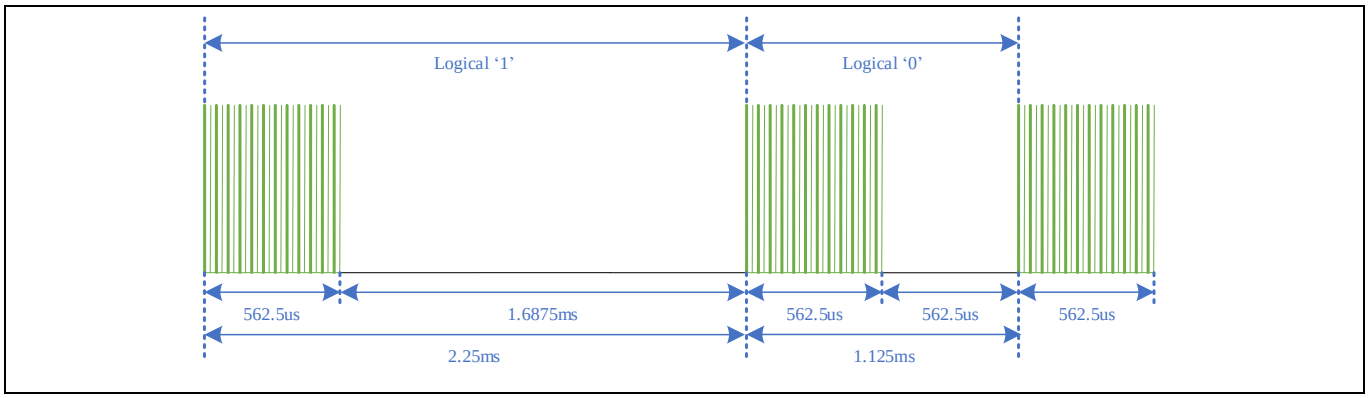


Figure 2 NEC Logical Define

When a key is pressed on the remote controller, the message transmitted consists of the following:

- a 9ms leading pulse burst (16 times the pulse burst length used for a logical data bit)
- a 4.5ms space
- the 8-bit address for the receiving device
- the 8-bit logical inverse of the address
- the 8-bit command
- the 8-bit logical inverse of the command
- a final 562.5µs pulse burst to signify the end of message transmission.

Figure 3 shows an example using the NEC IR transmission protocol. The four bytes of data bits are each sent least significant bit first. for an address of 00h (00000000b) and a command of ADh (10101101b).

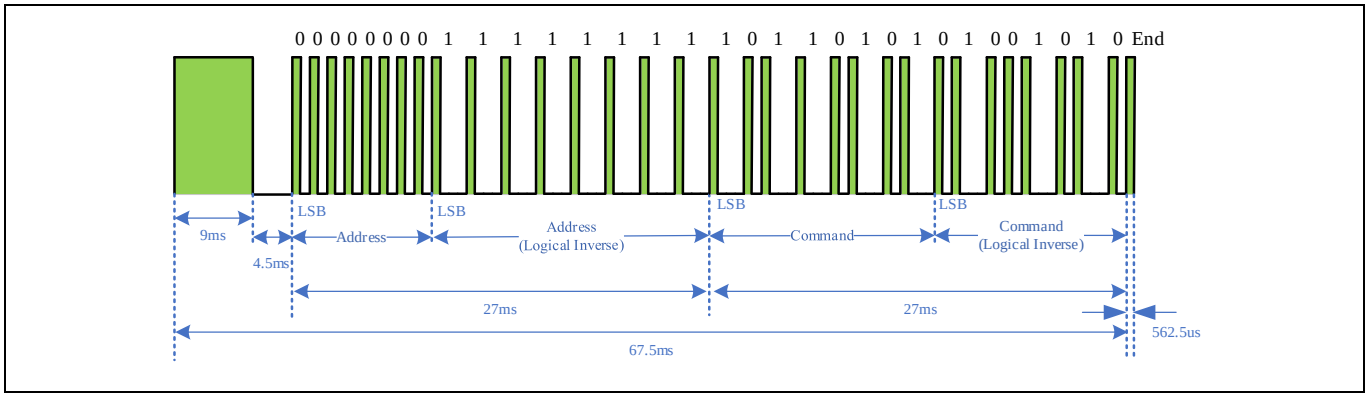


Figure 3 Example message frame using the NEC IR transmission protocol.

Infrared (IR) Remote Control

Occasionally, a user must modify the speed increment or decrement. When the button on the remote controller is held down, a repeat code is generated, typically occurring approximately 40ms after the pulse burst that indicated the completion of the message. Subsequently, this repeat code will be transmitted at regular intervals of 108ms until the key is eventually released. The repeat code consists of the following, as shown in Figure 4:

- a 9ms leading pulse burst
- a 2.25ms space
- a 562.5µs pulse burst to mark the end of the space

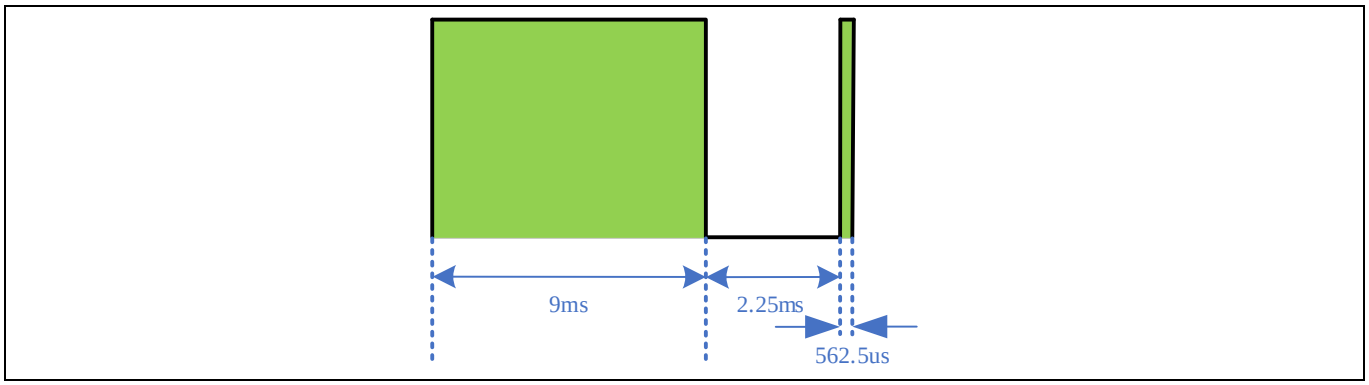


Figure 4 Repeat code contain

Figure 5 shows two repeat codes after an initial message is sent of the NEC IR transmission protocol.

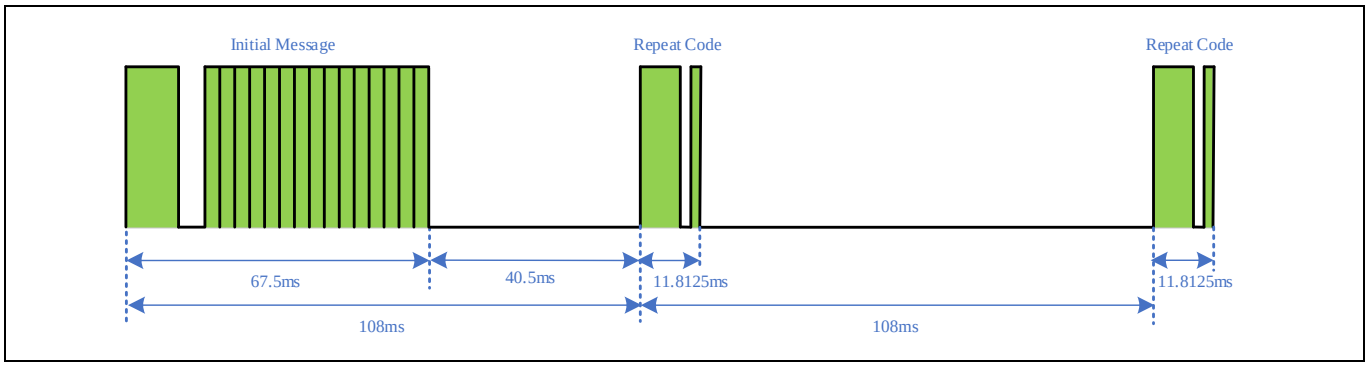


Figure 5 Repeat code example

1.3.2 NEC Extended Protocol

The difference between NEC Extended and NEC is that NEC Extended uses the address logical inverse byte as part of the address code. Conswequently, the address code of the NEC Extended protocol is changed from 8 bits to 16 bits. Except for the difference in the number of bits of the address code, the other contents of the NEC Extended protocol are the same as those of the standard NEC protocol. Figure 6 shows the example for an address of FF00h (1111111100000000b) and a command of ADh (10101101b).

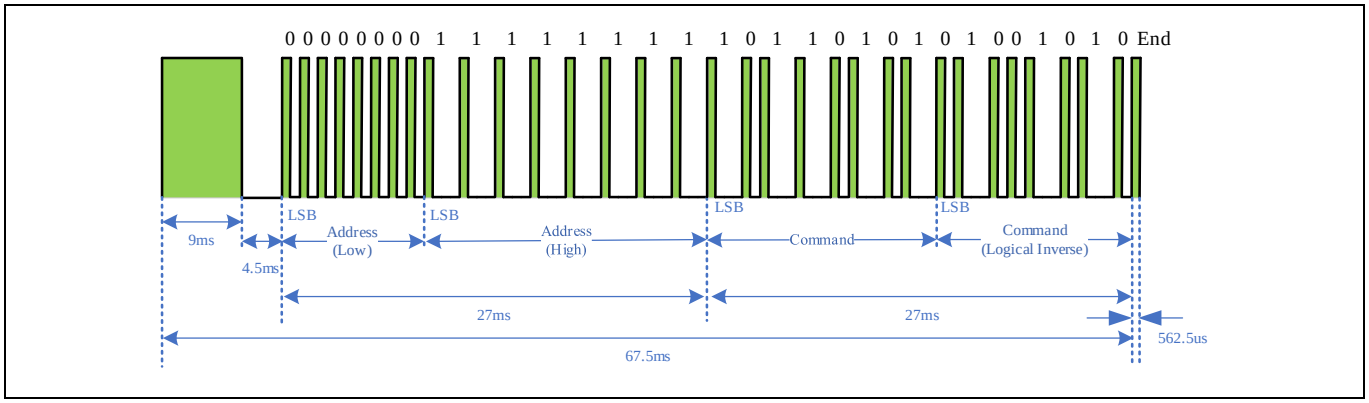


Figure 6 NEC Extended example

1.3.3 RC5 Philips Protocol

The RC5 Philips IR transmission protocol uses pulse distance encoding of the message bits. Each pulse burst (RC transmitter ON) is 889µs in length, at a carrier frequency of 36kHz. Logical bits are transmitted as follows:

- Logical '0' – a 889µs pulse burst followed by a 889µs space, with a total transmit time of 1.778ms.
- Logical '1' –a 889µs space followed by a 889µs pulse burst, with a total transmit time of 1.778ms.

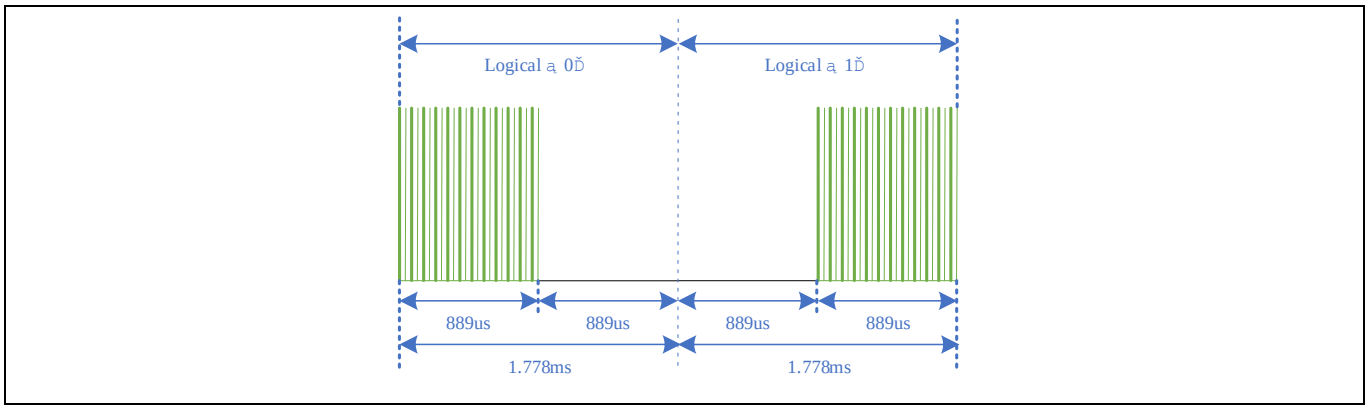


Figure 7 RC5 Philips Logical Define

Table 1 shows the RC5 Philips data packet. The message transmitted consists of the following:

- 1 Start Bit: logical 1
- 1 Field Bit: logical 1
- 1 TR Bit: When the remote controller button is released and pressed again, this bit will reverse (0 → 1, 1 → 0). In this way, the receiver can identify whether the key has been pressed or repeatedly pressed
- the 5-bit address
- the 6-bit command

Table 1 RC5 Philips data packet

Start Bit	Field Bit	TR Bit	Address Bits	Command Bits
1	1	1	5	6

Figure 8 shows an example using the RC5 Philips IR transmission protocol. The address bits and command bits are transmitted in the order of their significance, with the most significant bit being sent first. In this particular scenario, the address being transmitted is 01000b, and the command being transmitted is 001000b.

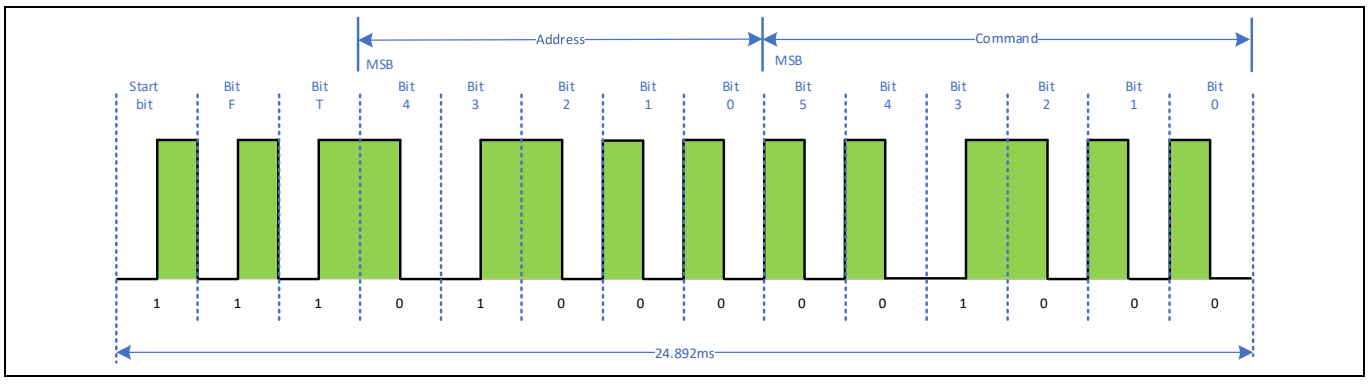


Figure 8 RC5 Philips example

2 FLASH Data Storage

2.1 Introduction

In some applications, the system may not work normally due a change in power input, such as sudden power failure. When incidents like this occur, it is important for the system to store the current operation data at the time of power failure. This allows for the system to automatically resume operation at it's previous state once power is restored.

The latest MCE firmware provides FLASH data storage function, which can easily be used through the provided APIs.

2.2 FLASH Data Storage function

All script supported variable types can be stored, but they must be defined with the keyword 'flash', for example:

```
flash uint8_t  FlashVar1;
flash uint16_t FlashVar2;
flash int32_t  FlashVar3;
```

Up to 160 bytes of flash variables can be stored when called by the Flash_Write(). Users have to ensure that this function is called when the motor is in the STOP or IDLE state. Because all interrupt executions of the MCE are prohibited during FLASH data storage, it is recommended to call when the motor is stopped to avoid some unexpected behaviors, such as a lack of motor control. That said, users must consider the flash write lifetime. Assuming that the system runs for 8 hours every day and needs to store data three times, it can be used around 15 years based on a 50000 erase cycles. Table 2 provides an explanation for the APIs of the flash data storage plug-in. For the APIs, more details information can find in the functional reference manual[1].

Table 2 Flash data storage APIs

API name	Brief description
Flash_Write()	Writes all "flash" type variables to flash
Flash_Erase()	Erases all data in allocated storage
Flash_GetWriteCount()	Returns amount of times flash has been written over lifetime
Flash_GetStatus()	Returns status from flash driver

3 Ceiling Fan Example Design

3.1 Requirement

This chapter will show the user how to utilize the infrared control and FLASH data storage functions. The purpose of the IR function is to provide users with a simple, convenient, and reliable way to control the motor via an application. We will use a ceiling fan application as an example to explain in detail how to attain the IR control function through scripting. We will also show users how to use a remote controller to achieve various speed command responses and employ the FLASH data storage function to save and retrieve the previous speed setting. Before normal operation, if a fault occurs, the system will attempt to automatically clear the fault for 5 times. During this period, the system will only operate normally if the fault disappears. Below are the fundamental system design prerequisites:

1. Respond to the 3 different speed setting commands and map the speed input commands to the relevant target speed settings.
2. Respond to the repeat command to increase and decrease speed as needed.
3. Respond to the power button press on the matching remote by toggling between the ON or OFF state. When switching to the ON state, it should resume from the last used speed setting. If there is no previously used speed setting is available, then it will start at the lowest speed setting.
4. Continuously monitor AC status by checking DC bus voltage level. Respond to AC brown-out event by initiating storing current speed setting into FLASH.
5. Upon start-up, restore last power status. If it is in ON status, then restore last saved speed setting.

3.2 IR Interface APIs

The MCE firmware is designed with the corresponding IR program interface in mind. In the application process users only need to call directly, which improves the ease of use and flexibility of the program. The APIs of the IR Interface plug-in are summarized in Table 3. More details can be found in the functional reference manual [1].

Table 3 IR APIs List

API name	Brief description
IR_DriverInit()	Initializes IR Driver based on key parameters
IR_DriverDeinit()	De-initializes IR Driver
IR_RxBuffer()	Returns most recent transmission
IR_GetStatus()	Returns status
IR_RxCommand()	Returns “Command” section of transmission
IR_RxAddress()	Returns “Address” section of transmission
IR_RxRepeats()	Returns numbers of transmissions repeated
IR_RxReceived()	Returns true if transmission has been received
IR_RxRepeating()	Returns true if transmission has not been fully received

3.3 IR Command Decode

This paragraph shows users how to best use specific IR controls. By using the remote controller to send commands, the IR receiver obtains the signal and uses the IR interface APIs to decode its specific content.

According to the analysis of requirements and the characteristics of this ceiling fan application, the script can be divided into three state machines. This includes the state machine for IR decode, the state machine for motor running, and the state machine for DC bus voltage monitoring. In the next chapter, we will discuss each state machine in detail.

In this ceiling fan application the remote controller has 6 buttons, where buttons 1 to 3 allow for 3 different speeds respectively. The 4 button is the speed accelerate command and the 5 button is the speed decelerate command. The other button is the power button, which can switch the start and stop commands of the motor. If the user presses a numbered button, then the MCE executes the desired speed command. If the power button is pressed, then the MCE needs to judge the current running state of the motor. If the motor is in the running state, then the button command should be resolved into a stop command; Conversely, if the motor is in the stop state, the button command should be interpreted as a start command. At this point, the motor has been running at the last executed setting speed. If there is no valid setting speed, the motor will run at the lowest valid speed.

In the script, the MCE uses the count value speed 'APP_MOTOR0.TargetSpeed' instead of the actual target speed. The relationship between the count value and the target speed is expressed as follows:

$$APP\_MOTOR0.TargetSpeed = \frac{TargetSpeed}{MaxSpeed} \times 16383$$

In this equation, the count 16383 represents the maximum speed count and the MaxSpeed is configurated in the iMOTION™ Solution Designer (iSD).

The relationship between the remote controller, the target speed setting, and the variables in the script for this design is listed inTable 4.

Table 4 IR control command, variables in script, and target speed setting

Remote button	IR control command	Target speed (rpm)	APP_MOTOR0.TargetSpeed(count)
Power	Power OFF	0	0
	Power ON	Last speed setting or 125	Last speed setting or 5319
1	Speed_1	125	5319
2	Speed_2	190	8085
3	Speed_3	255	10851
4	Speed accelerate	Current speed+ speed step (Max 385)	Current speed+ speed step (Max 16383)
5	Speed decelerate	Current speed - speed step (Min 125)	Current speed - speed step (Min 5319)

3.4 IR Control and FLASH Data Storage State Machine

3.4.1 IR decode state machine

The main work of this part is that the controller receives the IR signal and it decodes the IR power command and speed target. If the IR instruction changes (the previous command is not equal to the command received now), then users should ensure the CmdUpdate_Flag is active. This flag is used when the Flash_Write() is called to avoid repeated calls when the instruction is not updated.
The update of the IR control instruction requires a time delay to suppress multiple start/stops when the power button is pressed. Considering the APIs call time is 50ms, the delay time should not be less than 50ms. On the

other hand, if the delay time is too long, it will affect the timely response of commands. It is recommended that users set delay time between 50ms and 500ms.

Figure 9 shows the IR decode state machine. When the ceiling fan is powered on, the IR system starts to work. Because the execution period of the script is 50ms and the command delay time is 500ms, a counter with a cycle of 10 is set here. When the counter value reaches 10, enter the 'IR read' state. Once users have entered the IR read state, the counter value will be cleared to 0, and the IR command will be read at the same time. The power status will then be judged; if the previous one is the Power OFF, then this one is the Power ON, and vice versa.

Then the speed command will be parsed and the corresponding target speed will be set according to the different speed commands

It is necessary to store the speed and power state information each time to determine whether the next command is different with the current command.

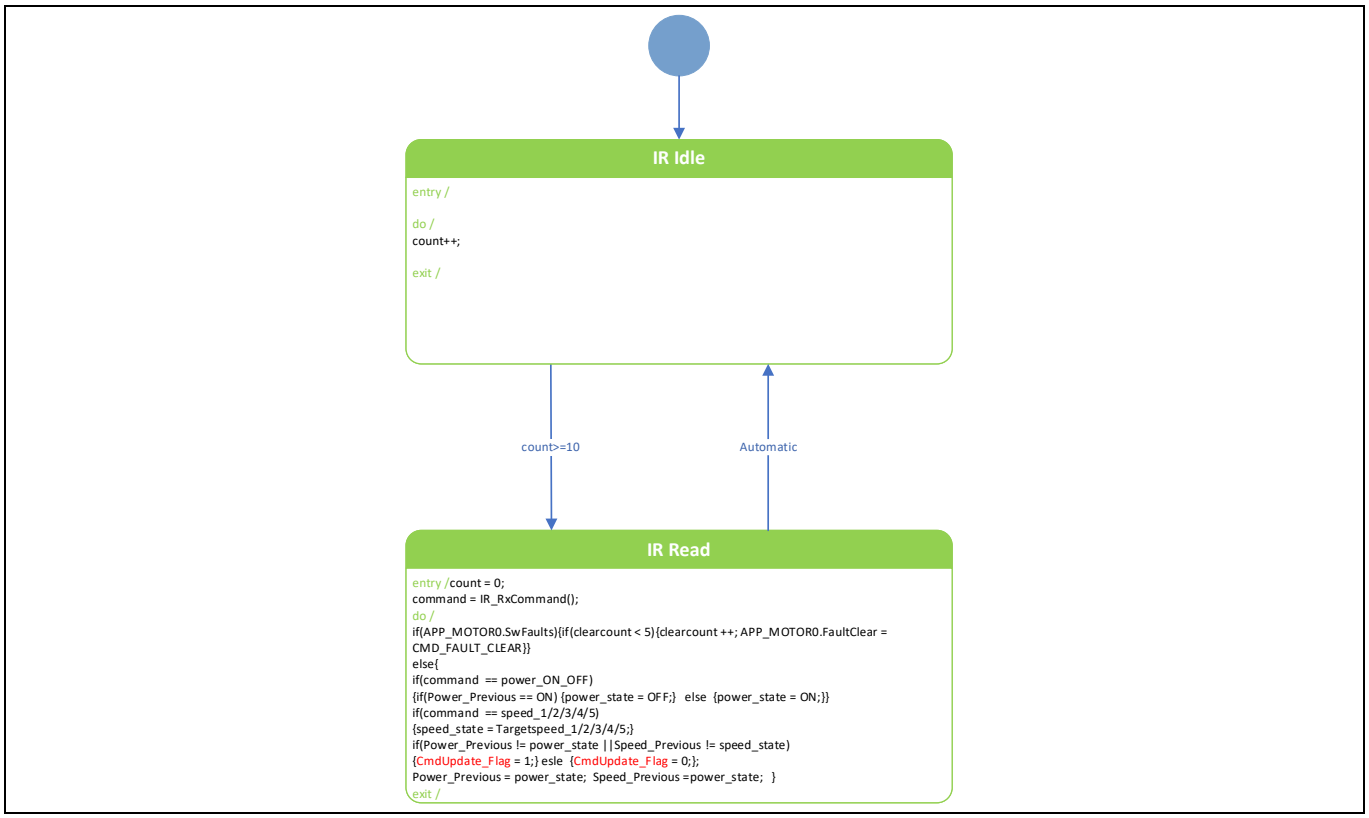


Figure 9 IR decode state machine

Table 5 shows the input and output parameters for the IR decode state machine. The output parameters are sent to the motor running state machine.

Table 5 IR decode state machine input and output parameters

Input parameters	Output parameters
NA	Power_state
NA	Speed_state
NA	CmdUpdate_Flag

3.4.2 Motor running state machine

Figure 10 shows the motor running state machine. When the IR receives the Power ON command, the ceiling fan starts to run at the target speed. During running state, the variable CmdUpdate_Flag will be monitored at all times to determine whether new instructions are received. If new instructions are received, the variable FlashWrite_Flag will be set to 1. The motor will stop running when the Power OFF command is received. If the brown-out event occurs, the system will issue a fault signal and the motor will also stop running. In the stop state, the command is set as OFF. If the motor sequence state is STOP or FAULT, a brown-out event has occured, and the FlashWrite_Flag is 1, we will need to store the power state and speed state. At that point, the script will call the Flash_Write() function to store the data. After this, clearing the FlashWrite_Flag ensures the Flash_Write() function is called only once due to the limited FLASH write life cycles.

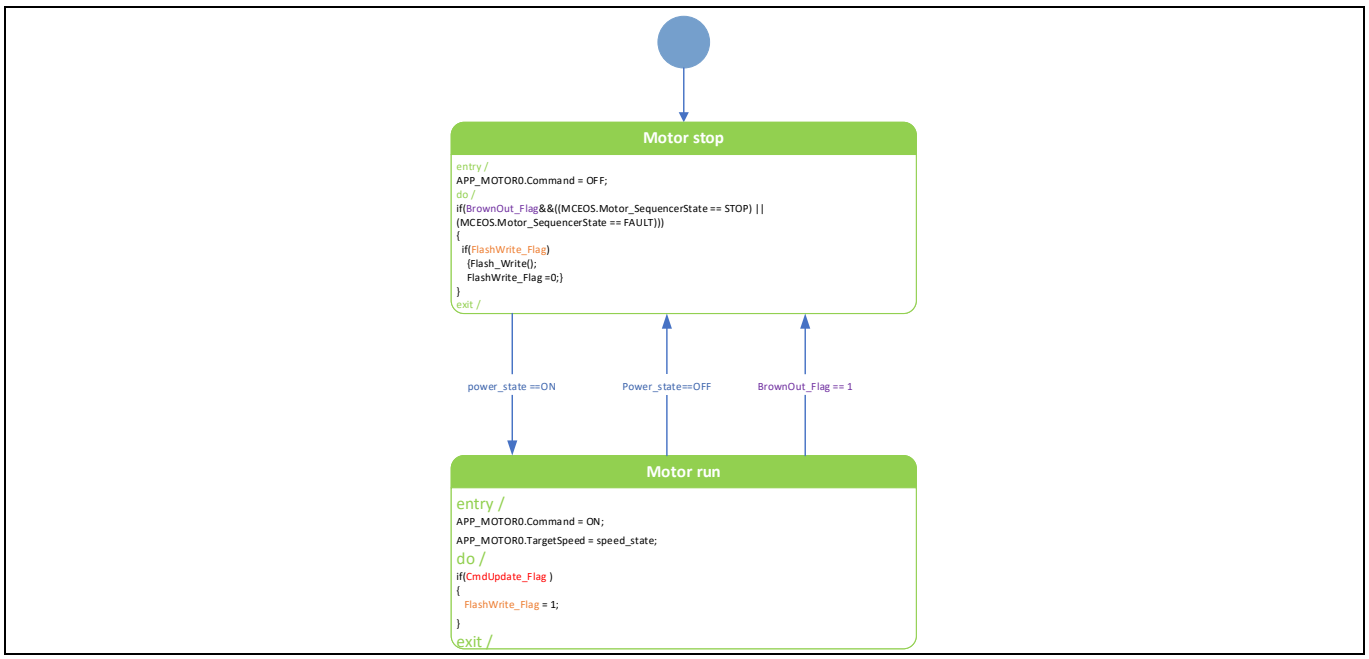


Figure 10 Motor running state machine

Table 6 shows the input and output parameters for the motor running state machine. The input parameters ‘Power_state’,’ Speed_state’, and ‘CmdUpdate_Flag’ are received from the IR decode state machine. The input parameter ‘BrownOut_Flag’ is received from the DC bus state machine.

Table 6 Motor running state machine input and output parameters

Input parameters	Output parameters
Power_state	NA
Speed_state	NA
CmdUpdate_Flag	NA
BrownOut_Flag	NA

3.4.3 DC bus state machine

According to the application requirements, the MCE should store the last speed and motor run state setting when the AC brown-out event occurrence. For this ceiling application, since no AC input signal is sampled, the DC bus voltage signal is used instead.

The iSD sets the undervoltage protection value. In order to reserve enough power supply voltage to enable data storage to be completed successfully, we recommend setting a software protection value that is greater than the undervoltage protection voltage. The value of this threshold should fully consider the design of the system hardware. The system should meet the requirements of providing sufficient voltage to enable data storage when the brown-out occurs. If the DC bus voltage remains below this value for a certain duration, the script will activate the brown-out flag and issue a stop command. Simultaneously, by setting a hysteresis threshold, when the voltage exceeds this threshold and remains consistently higher for a specific period, the normal power supply logic will be reinstated, restoring the initial state. The hysteresis setting is used to avoid unexpected actions that may occur when the voltage value fluctuates at the threshold. The hysteresis value suggestion should be larger than the voltage fluctuation. Figure 11 shows the relationship.

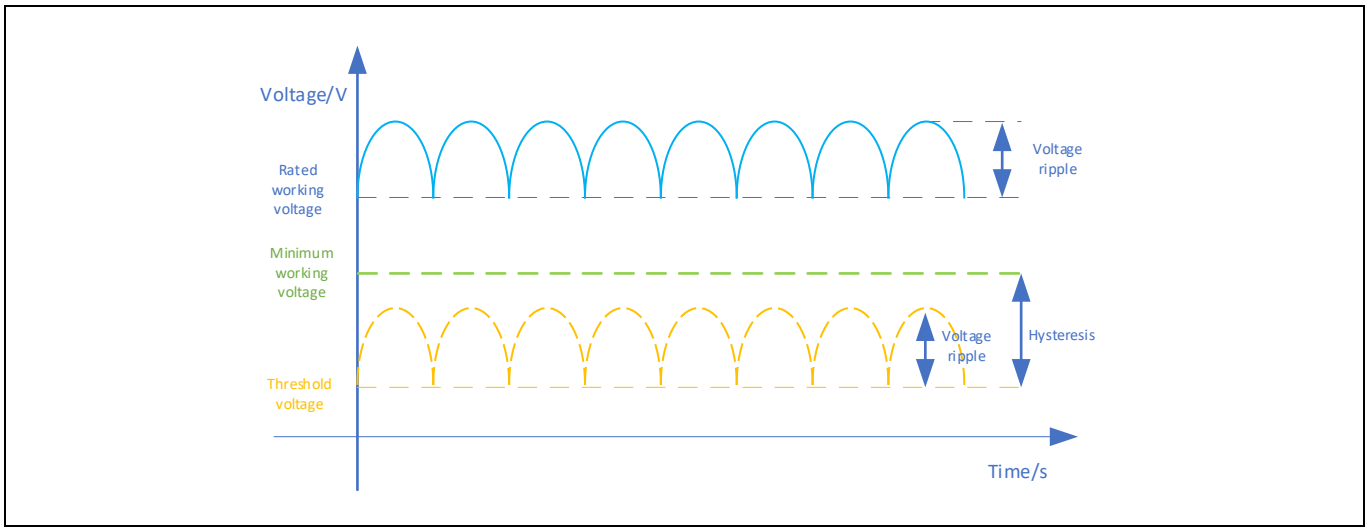


Figure 11 DC Voltage hysteresis diagram

Figure 12 shows the DC bus state machine. The ceiling fan works normally and it enters this state when the voltage is normal. First, it will check whether there is a valid power state. If the data shows that there is no valid data in the storage area, then power state is set to OFF and the target speed is set to the lowest speed. During this state, the script will monitor the DC bus voltage variable FB_MEASURE.VdcFilt constantly and set the BrownOut_Flag as 0. Once the FB_MEASURE.VdcFilt is detected to be below the threshold for a continuous 10 execution cycles (the number of cycles can be configured using the variable 'DC_Time_Out'), a brown-out event has occurred, and the system will enter the DC bus abnormal state machine. The DC bus abnormal state machine will set the BrownOut_Flag as 1. This flag variable is a key parameter used for the Flash_Write() function.

When the FB_MEASURE.VdcFilt exceeds the threshold, plus the hysteresis, for a continuous period of 10 execution cycles, it can be inferred that the input power has recovered and returned to the normal state of the DC bus machine.

There are two variables, “threshold” and “hysteresis”, that define the reasonable value. Since the DC bus voltage reflects the AC input voltage, the lowest voltage that can work should be considered here. In the same way, the hysteresis value should be considered to avoid the normal fluctuation of the DC bus waveform.

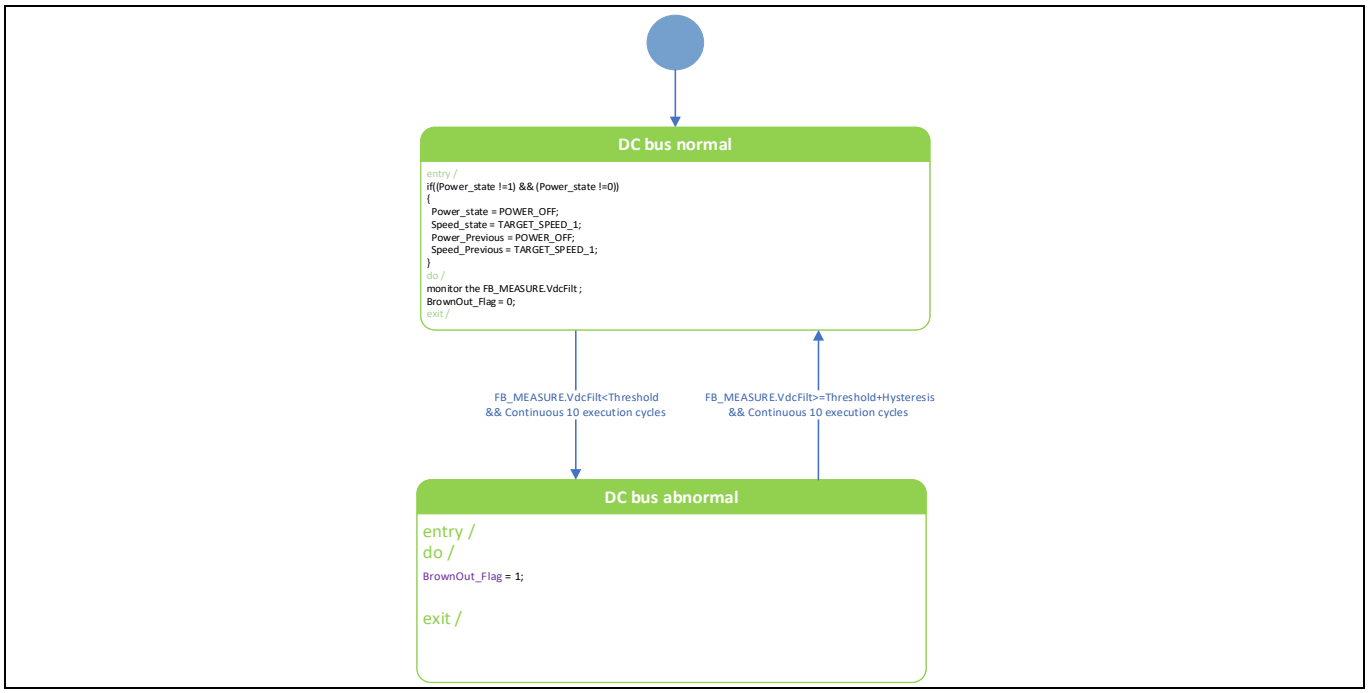


Figure 12 DC bus state machine

Table 7 shows the input and output parameters for the motor running state machine. The output parameter 'BrownOut_Flag' is sent to the motor running state machine.

Table 7 DC bus state machine input and output parameters

Input parameters	Output parameters
NA	BrownOut_Flag

3.5 Script Implementation

3.5.1 Code for iSD

Code Listing 1 shows the global parameters define and Code Listing 2 shows the IR remote control how to realize the process.

Code Listing 1 IR Remote Control Interface Script Code for iSD (Global.mcs)

```
/*************************************************************/
/*Global variables*/
/*************************************************************/
/* status variables */
int32_t IR_status;
int32_t IR_DriverStatus;
/* data and address variables */
uint8_t command;
uint8_t Repeating;
uint8_t mask_cnt;
int32_t CMD_STATE;
uint8_t FlashWriteFlag;
```

Code Listing 1 IR Remote Control Interface Script Code for iSD (Global.mcs)

```
013         uint8_t BrownOut_Flag;
014         uint8_t CmdUpdate_Flag;
015         uint8_t DCabnormal_count;
016         uint8_t DCnormal_count;
017         uint8_t FlashWrite_Flag;
018         uint8_t FlashStatus;
019         uint8_t FlashEmpty;
020         uint8_t FlashInvalid;
021         uint8_t FlashWriteError;
022         uint8_t FaultClearCount;
023         CONST int CMD_SPEED_1 = 26;
024         CONST int CMD_SPEED_2 = 2;
025         CONST int CMD_SPEED_3 = 3;
026         CONST int CMD_ACCELERATE = 6;
027         CONST int CMD_DECELERATE = 5;
028         CONST int CMD_ON_OFF = 1;
029         CONST int TARGET_SPEED_1 = 5319;    //5319 =
   125rpm/385rpm*16383
030         CONST int TARGET_SPEED_2 = 8085;
031         CONST int TARGET_SPEED_3 = 10851;
032         CONST int MAX_SPEED = 16383;
033         CONST int MIN_SPEED = 5319;
034         CONST int VDC_THRESHOLD = 2313;//150;      //Vdc_Max 42.5V
   2313 = 24V/42.5V*4096    150 for test
035         CONST int VDC_Hysteresis  = 10; //144;      //1.5V   2313 =
   1.5V/42.5V*4096
036         CONST int STEP_SPEED  = 100;
037         CONST int STOP_State = 1;
038         CONST int FAULT_State  = 5;
039         CONST int MASK = 0;
040         CONST int READY = 1;
041         CONST int MASK_TIME_OUT = 10;
042         CONST int STOP = 1;    //Motor_SequencerState
043         CONST int POWER_ON = 1;
044         CONST int POWER_OFF = 0;
045         CONST int DC_Time_Out = 10;
046         CONST int CMD_FAULT_CLEAR = 1; //CLEARE FAULT
047         flash uint16_t Power_state;    // Flash storage data
048         flash uint16_t Speed_state;    // Flash storage data
049         flash uint16_t Power_Previous;
050         flash uint16_t Speed_Previous;
```

Code Listing 2 IR Remote Control Interface Script Code for iSD (Task0.mcs)

```
001         /*Task1 init function*/
002         Script_Task1_init()
003         {
004               IR_DriverDeInit();
005               /* Initialize IR Interface */
006               /* channel (0-RX0, 1-RX1, 2-VSP),  rxinvert, protocol
   (0:RC5, 1:NEC, 2:NECextended), address  */
007               IR_DriverStatus = IR_DriverInit(2,1,1,128);
```

Code Listing 2 IR Remote Control Interface Script Code for iSD (Task0.mcs)

```
008                FlashStatus = Flash_GetStatus();
009                FlashEmpty = FlashStatus & 0x01;
010                FlashInvalid = FlashStatus & 0x02;
011                FlashWriteError = FlashStatus & 0x04;
012                CMD_STATE = READY;
013                mask_cnt = 0;
014                APP_MOTOR0.TargetSpeed = 0;
015                BrownOut_Flag = 0;
016                CmdUpdate_Flag = 0;
017                FlashWrite_Flag = 0;
018                FaultClearCount = 0;
019                APP_MOTOR0.FaultClear = CMD_FAULT_CLEAR;
020                if((Power_state !=1) && (Power_state !=0))
021                {
022                       Power_state = POWER_OFF;
023                       Speed_state = TARGET_SPEED_1;
024                       Power_Previous = POWER_OFF;
025                       Speed_Previous = TARGET_SPEED_1;
026                }
027         }
028         /*************************************************************/
029         /*Task1 function*/
030         Script_Task1()
031         {
032                if(APP_MOTOR0.SwFaults)
033                {
034                       if(FaultClearCount < 5)
035                       {
036                          APP_MOTOR0.FaultClear = CMD_FAULT_CLEAR;
037                          FaultClearCount = FaultClearCount + 1;
038                       }
039                }
040                else
041                {
042                       FaultClearCount = 0;
043                      IR_status = IR_RxReceived();
044                      Repeating = IR_RxRepeating();
045                }
046                if(CMD_STATE == MASK)
047                {
048                       mask_cnt = mask_cnt + 1;
049                       if(mask_cnt == MASK_TIME_OUT)
050                       {
051                              CMD_STATE = READY;
052                              mask_cnt = 0;
053                       }
054                }
055                /* Received Data valid */
056                if(IR_status || Repeating)
057                {
058                       command = IR_RxCommand();
059                       if(CMD_STATE == READY)
060                       {
061                              if(command == CMD_ON_OFF)
```

Code Listing 2 IR Remote Control Interface Script Code for iSD (Task0.mcs)

```
062                                 {
063                                       if(Power_state == POWER_OFF)
064                                       {
065                                             Power_state = POWER_ON;
066                                       }
067                                       else
068                                       {
069                                             Power_state = POWER_OFF;
070                                       }
071                                 }
072                           }
073                           if(Power_state == POWER_ON)
074                           {
075                                if(command == CMD_SPEED_1)
076                               {
077                                   Speed_state = TARGET_SPEED_1;
078                                }
079                                if(command == CMD_SPEED_2)
080                                {
081                                   Speed_state = TARGET_SPEED_2;
082                                }
083                                if(command == CMD_SPEED_3)
084                                {
085                                   Speed_state = TARGET_SPEED_3;
086                                }
087                                if(command == CMD_ACCELERATE)
088                                {
089                                   if(Repeating)
090                                   {
091                                     Speed_state = Speed_state + STEP_SPEED;
092                                     if(Speed_state >= MAX_SPEED)
093                                     {
094                                         Speed_state = MAX_SPEED;
095                                     }
096                                   }
097
098                                }
099                                if(command == CMD_DECELERATE)
100                                {
101                                   if(Repeating)
102                                   {
103                                     Speed_state = Speed_state - STEP_SPEED;
104                                     if(Speed_state <= MIN_SPEED)
105                                       {
106                                         Speed_state = MIN_SPEED;
                                       }
107                                    }
108                                }
109                           }
110                           CMD_STATE = MASK;
111                     }
112                     if((Power_Previous != Power_state) || (Speed_Previous!=
   Speed_state))
113                     {
```

Code Listing 2 IR Remote Control Interface Script Code for iSD (Task0.mcs)

```
                        CmdUpdate_Flag = 1;
                }
                else
                {
                        CmdUpdate_Flag = 0;
                }
                if(CmdUpdate_Flag)
                {
                        FlashWrite_Flag = 1;
                }
                Power_Previous = Power_state;
                Speed_Previous = Speed_state;
                if(FB_MEASURE.VdcFilt <= VDC_THRESHOLD )
   //VDC_THRESHOLD
                {
                        DCnormal_count = 0;
                        DCabnormal_count = DCabnormal_count + 1 ;
                        if(DCabnormal_count >= DC_Time_Out)
                        {
                                DCabnormal_count = 0;
                                BrownOut_Flag = 1;
                        }
                }
                if(FB_MEASURE.VdcFilt >= (VDC_THRESHOLD +
   VDC_Hysteresis))
                {
                        DCabnormal_count = 0;
                        DCnormal_count = DCnormal_count + 1 ;
                        if(DCnormal_count >= DC_Time_Out)
                        {
                                DCnormal_count = 0;
                                BrownOut_Flag = 0;
                        }
                }
                if(BrownOut_Flag)
                {
                        APP_MOTOR0.Command = POWER_OFF;
                }
                else
                {
                        APP_MOTOR0.Command = Power_state;
                        APP_MOTOR0.TargetSpeed = Speed_state;
                }
                if(BrownOut_Flag && ((MCEOS.Motor_SequencerState ==
   STOP_State) || (MCEOS.Motor_SequencerState == FAULT_State)))
                //if(BrownOut_Flag)
                {

                        if(FlashWrite_Flag)
                        {
                                Flash_Write();
                                FlashWrite_Flag = 0;
                        }
                }
```

Code Listing 2 IR Remote Control Interface Script Code for iSD (Task0.mcs)

```
165         }
```

In this application, the call cycle of the APIs requires 50ms. Therefore, the overall script execution cycle must be at least 50ms. However, it is not advisable to set it excessively long as it would adversely impact the program's execution efficiency. The script consists of 68 lines of instructions, provided that the execution of all 68 lines is finished within the designated execution period. This example is set to the execution step as 70. For more settings, please refer to functional reference manual[1].

Figure 13 shows script task build output information. Figure 14 shows the script setting information.

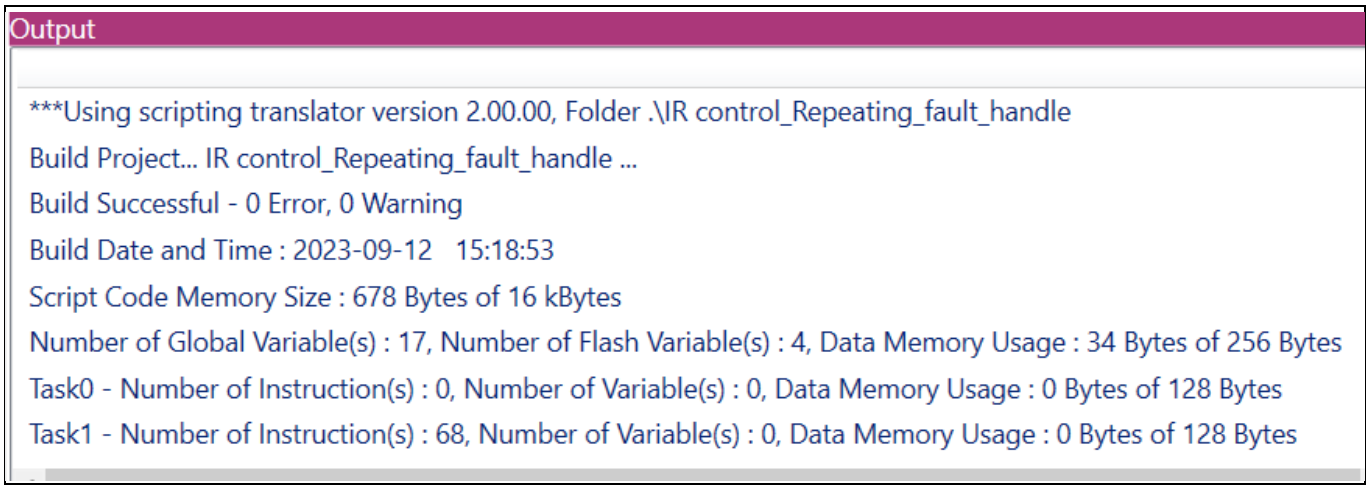


Figure 13 Task bulid information

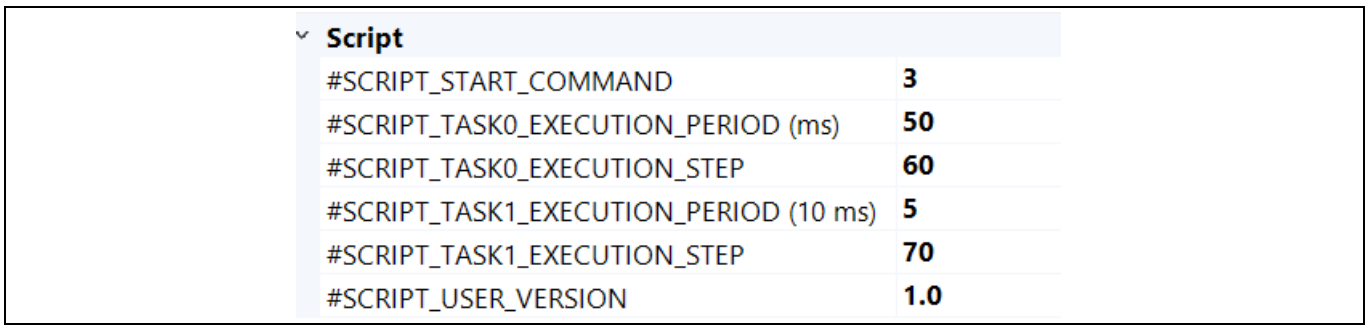


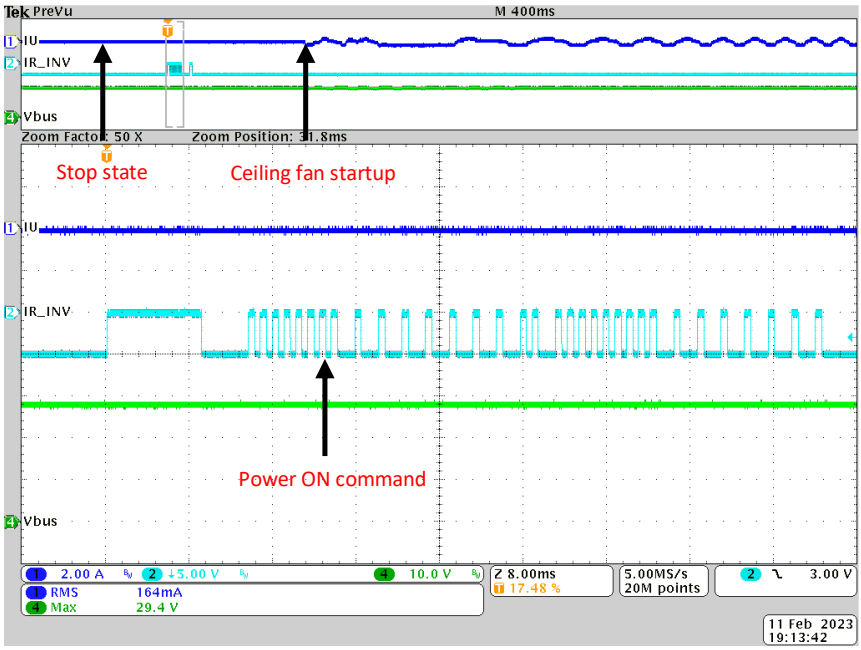
Figure 14 The execution period and the step for the IR control

4 Test Result

All the functions in this document are implemented by the script language code and tested with the evaluation board REF-SHA35IMD111TSYS. For more information, please refer to the evaluation board user guide[3].

4.1 IR command test

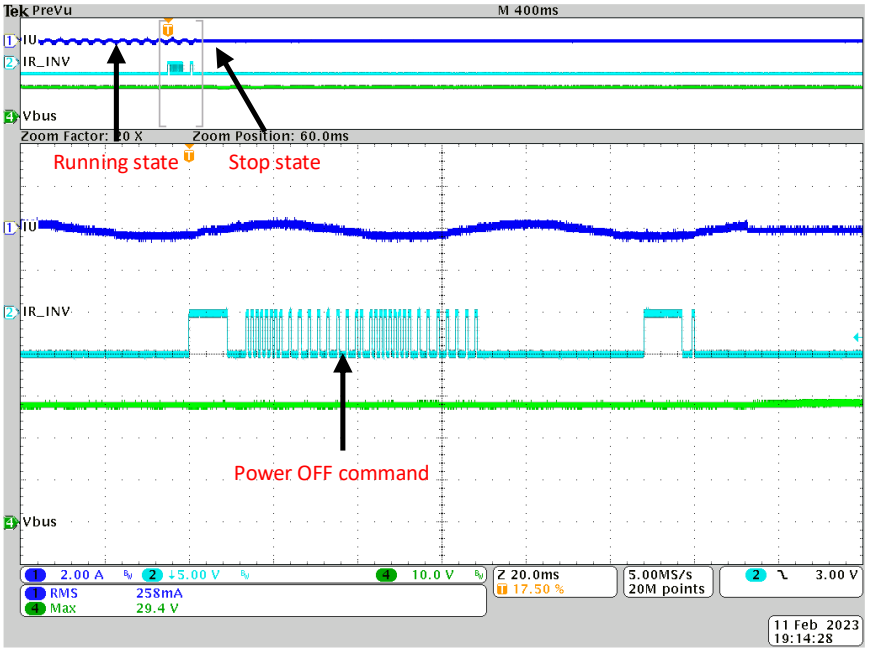
Figure 15 and Figure 16 show the IR command of Power ON and OFF. Figure 17 shows the speed command change status.



CH1: Phase U current
CH2: IR_INV command signal
CH4: Vbus voltage

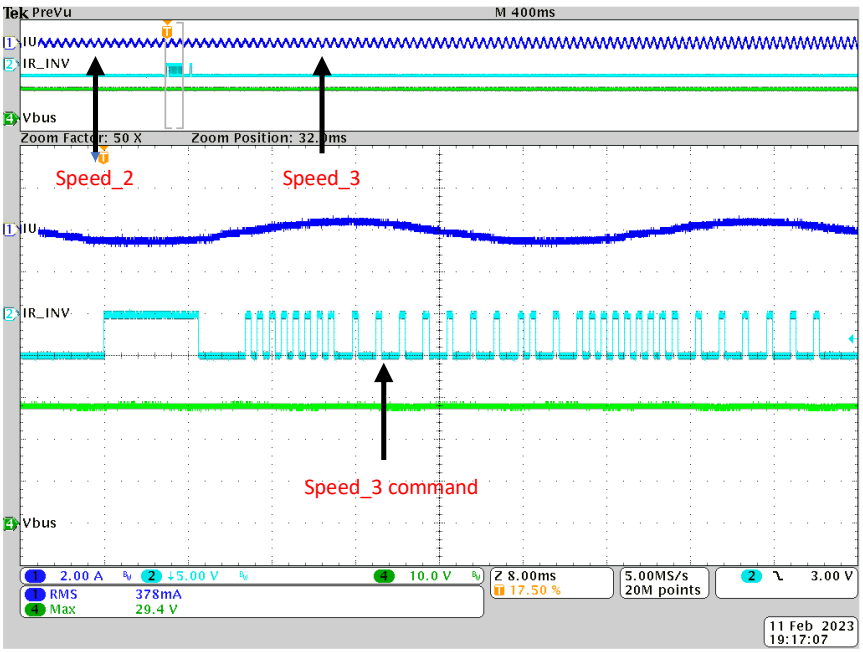
Figure 15 IR command Power ON

Test Result



CH1: Phase U current
CH2: IR_INV command signal
CH4: Vbus voltage

Figure 16 IR command Power OFF

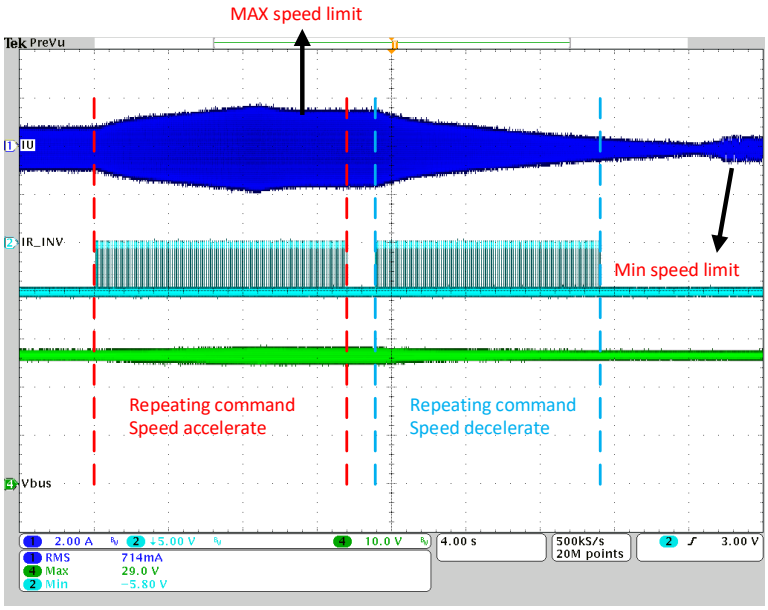


CH1: Phase U current
CH2: IR_INV command signal
CH4: Vbus voltage

Figure 17 IR Speed command change test

Test Result

Figure 18 shows the repeating command for the speed accelerate and speed decelerate functions.

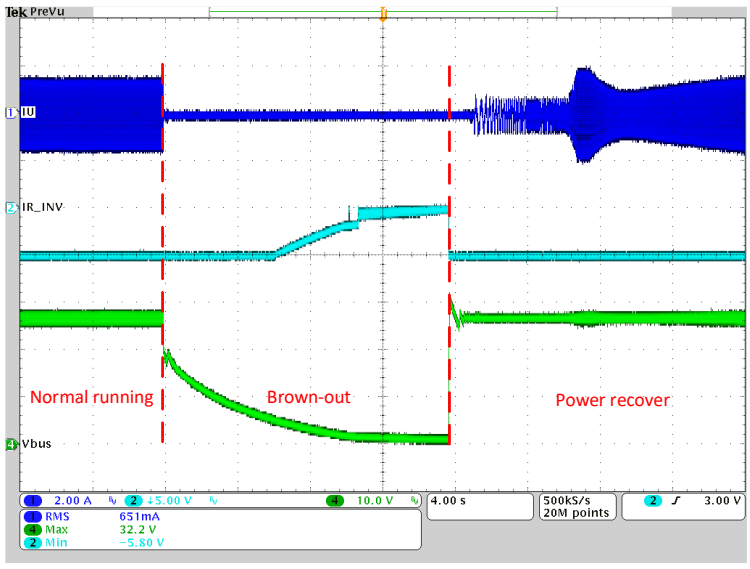


CH1: Phase U current
CH2: IR_INV command signal
CH4: Vbus voltage

Figure 18 Repeating command for speed accelerate and speed decelerate

4.2 Last speed setting test

Figure 19 shows the flash data storage function. At first, the ceiling fan was in normal running. Then the brown-out event occurred and the ceiling would stop, storing the power and speed state. Finally, when the power recovered, the ceiling fan resumed from the last speed setting.



CH1: Phase U current
CH2: IR_INV command signal
CH4: Vbus voltage

Figure 19 resume from the last speed setting after power recover

5 Summary

5.1 The difference between task0 and task1

Task0 is executed in an interrupt program, while task1 is executed in a background program. Whether the script is placed in task0 or task1, users should comprehensively consider the CPU load of the whole program. We recommend that the CPU load should not be higher than 90% when executing the entire program, otherwise, the script function may cause CPU overload, thus affecting the operation of the MCE and PFC.
Figure 20 shows an example of CPU load. The example in this application note is using script Task 0 because it uses 70.9% CPU load. It is recommended to use script in task1 when CPU load is higher than 90%.

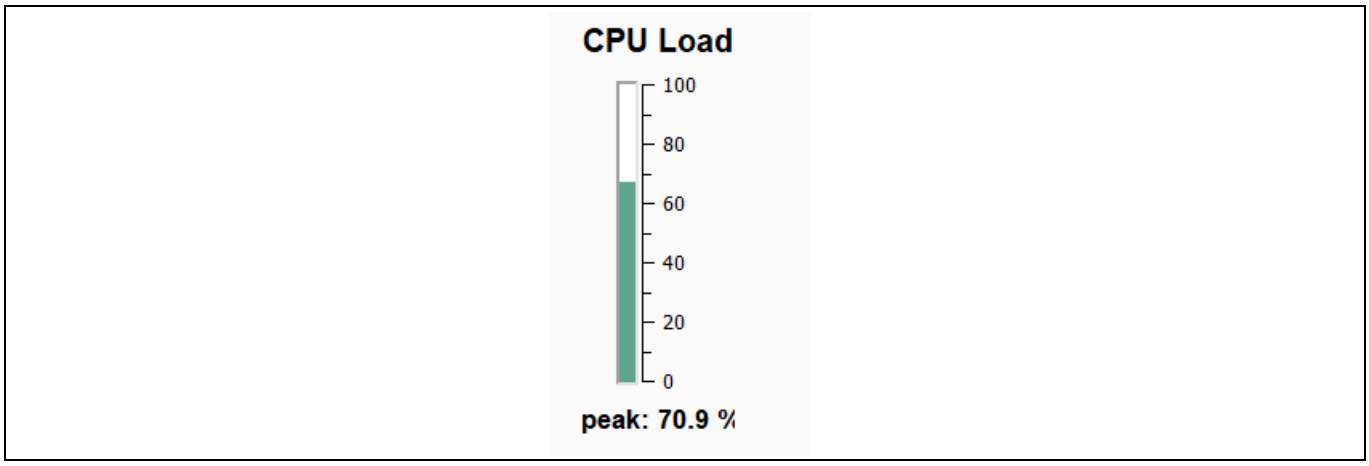


Figure 20 CPU load

5.2 The delay time for the remote control command

As explained in section 3.4.1, the APIs associated with IR control require 50ms to be called within the MCE. Consequently, the scripting design's delay time does not necessarily have to be faster than this value. However, it should not be excessively long, considering both user experience and program execution efficiency. The delay time needs to be carefully evaluated in a comprehensive manner by the user.

6 Reference

[1] Functional Reference Manual iMOTION™ Motion Control Engine

[2] How to Use iMOTION™ Script Language

[3] REF-SHA35IMD111TSYS User guide Full-featured starter kit for low voltage ceiling fan motor designs

[4] IMI111T - iMOTION™ IPM for motor control data sheet

Table of contents

Revision history

Document version	Date of release	Description of changes
1.0	2023-10-17	Initial release

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

Edition 2023-10-17

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference
AN-2023-03

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.