AN204945

F²MC-8FX Family MB95330 Series 120° Hall Sensor/Sensorless DC Inverter Control Softune C Library

Associated Part Family: MB95330 Series

This application note describes the implementation of 120° conduction hall sensor/sensorless brushless DC motor control using the provided F2MC-8L/8FX SOFTUNE C library and the Cypress MB95F330 8-bit microcontroller.

Contents

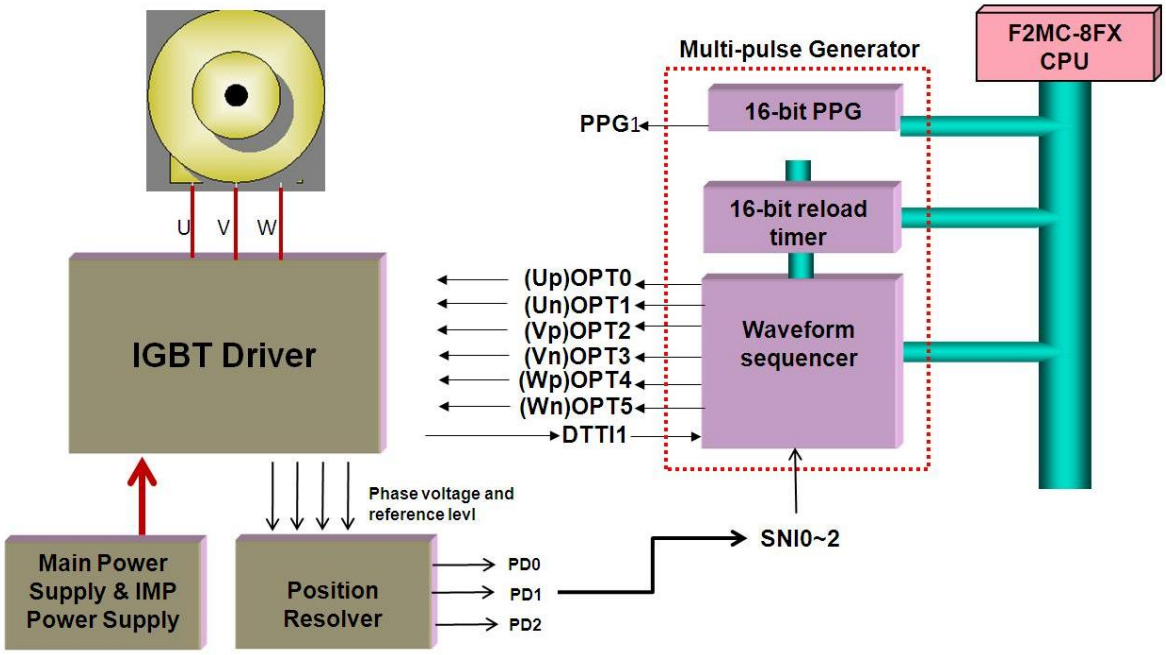
1 Introduction

This document describes the implementation of 120° conduction hall sensor/sensorless brushless DC motor control using the provided F2MC-8L/8FX SOFTUNE C library and the Cypress MB95F330 8-bit microcontroller. The operation principles, specification, library installation, library function description and operation of library functions are included._MB95F330 series 8-bit Micro-controller can be used to control the operation of a 3-phase brushless DC motor using the 120° conduction inverter control solution.

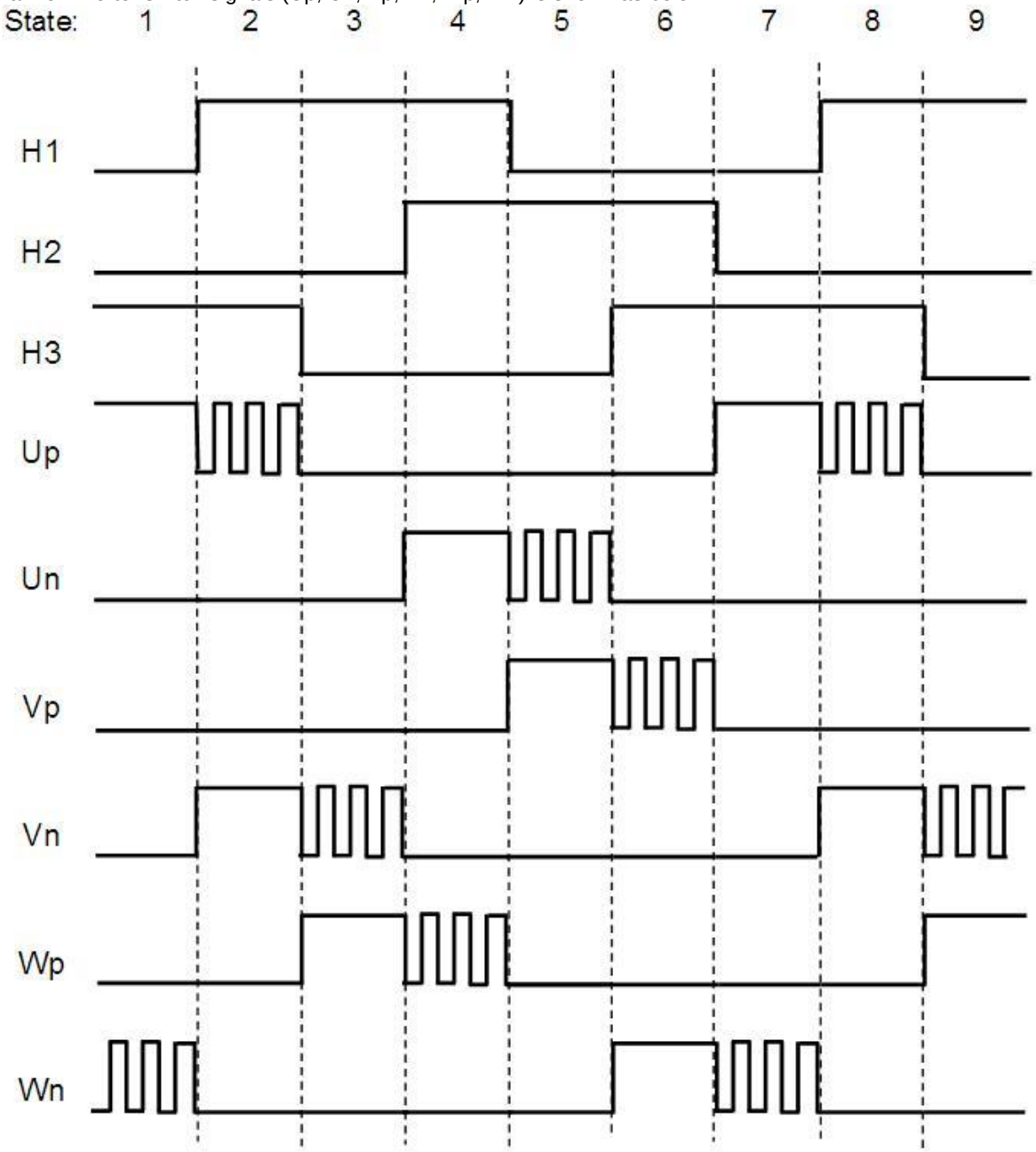
2 Operation Principles and Theory

2.1 Hall Sensor Drive

Below is the brief working principle for MCU to drive motor with hall sensor. A multi-pulse generator outputs six switch signals to drive IGBT inverter. Three channel hall sensor signals are detected by MCU input capture to achieve motor position. One channel over-current signal is output by IGBT inverter to MCU to protect the whole system.


Multi-pulse Generator
F2MC-8FX CPU
16-bit PPG
PPG1
16-bit reload timer
Waveform sequencer
(Up)OPT0
(Un)OPT1
(Vp)OPT2
(Vn)OPT3
(Wp)OPT4
(Wn)OPT5
DTTI1
SNI0~2
U
V
W
IGBT Driver
Phase voltage and reference levl
Main Power Supply & IMP Power Supply
Position Resolver
PD0
PD1
PD2

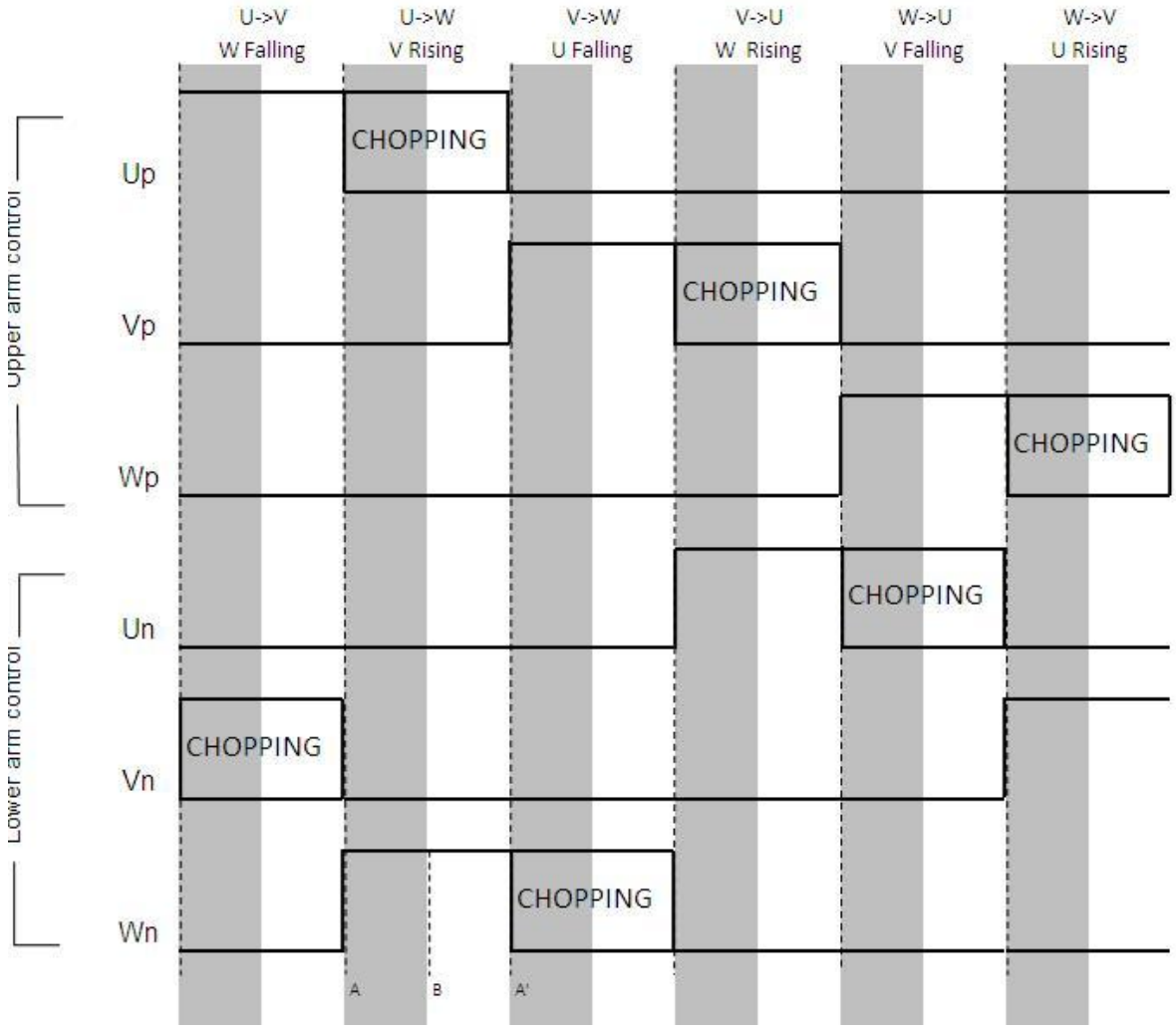
One electrical cycle is divided into 6 states. The relationship between three channel hall sensor signals (H1, H2, H3) and six channel inverter switch signals (Up, Un, Vp, Vn, Wp, Wn) is shown as below:



2.2 Sensorless Drive

2.2.1 Sensorless Startup

The suggested startup method is forced startup. The following is the driving pattern. The marker A and A' are the state change, while A – B is the position detect mask-off period used to mask off unwanted interrupt when the back EMF is very weak during startup.



2.2.2 Normal Run

The normal run consists of 12 different driving patterns and 6 different states. The following shows the relationship between the driving patterns and the expected interrupts from the position detection circuit.

Marker explanation:

A: position detection interrupt

B: change state

C: change chopping-arm

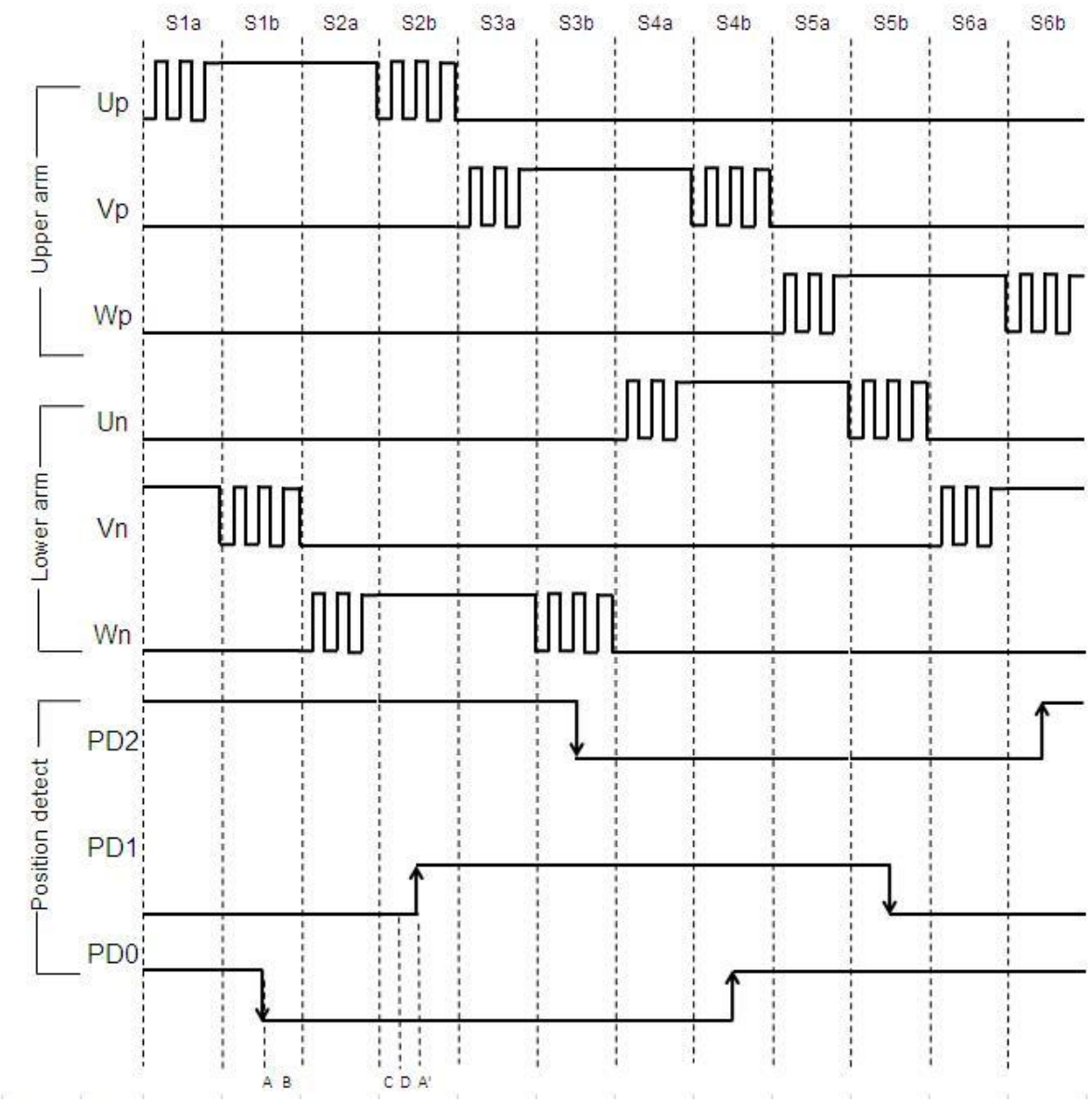
D: position detection interrupt enable

A': next position detection interrupt

A – B: commutation delay

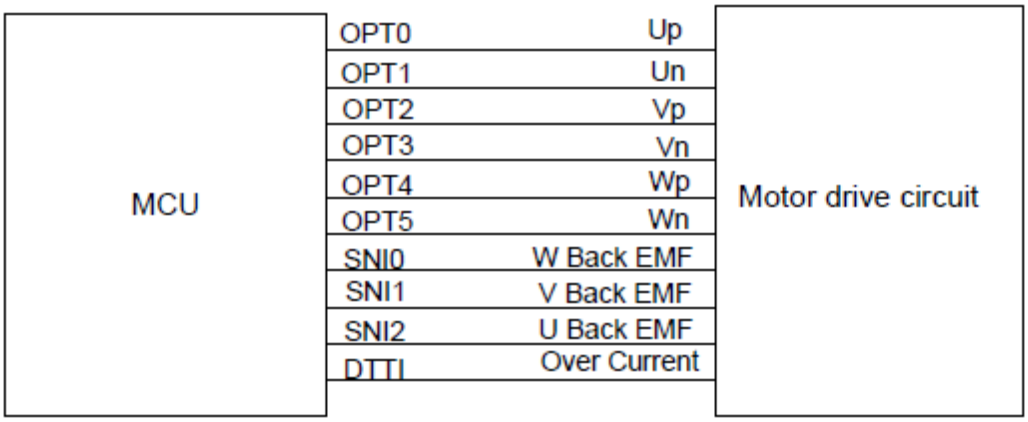
B – C: change arm delay

C – D: change arm mask-off period

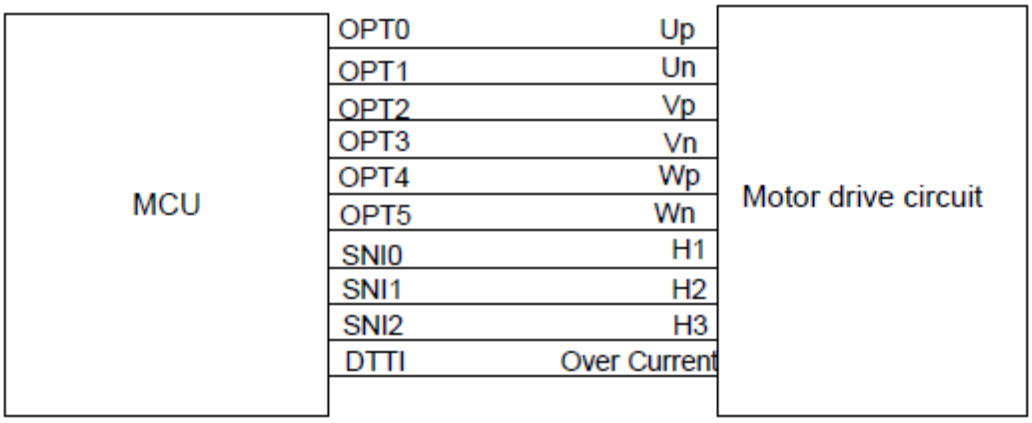

S1a
S1b
S2a
S2b
S3a
S3b
S4a
S4b
S5a
S5b
S6a
S6b
Upper arm
Up
Vp
Wp
Lower arm
Un
Vn
Wn
Position detect
PD2
PD1
PD0
A B
C D A'

3 HW Description

The library is used for MB95F330 and MB95F390 series MCU. This library only uses the multi-pulse generator of MCU and it uses three interruptions: 16-bit reload timer ch.1, MPG (writing timer or compare match) and MPG (DTTI). The16-bit reload timer ch.1 and the MPG (writing timer or compare match) interruptions are closed in the library. The MPG (DTTI) interruption is open. In order to use this library more easily, the connection between MCU and motor drive circuit should be as below.



Back EMF Drive



Hall Sensor Drive

Note:

OPT0, OPT1, OPT2, OPT3, OPT4, OPT5, SNI0, SNI1, SNI2, DTTI: MCU pin.

Up: U phase upper arm inverter switch signal.

Un: U phase lower arm inverter switch signal.

Vp: V phase upper arm inverter switch signal.

Vn: V phase lower arm inverter switch signal.

Wp: W phase upper arm inverter switch signal.

Wn: W phase lower arm inverter switch signal.

H1, H2, H3: Hall sensor signal.

Over current: Over current signal.

W Back EMF: W phase back electromotive force.

V Back EMF: V phase back electromotive force.

U Back EMF: U phase back electromotive force.

Over Current: Over current protection signal.

4 Library Installation

4.1 Components

The library package contains 3 files

File name	Usage
MB95F330 Motor Drive V0.2.0.lib	Library file, contains all function modules
DTTI.c	Deal DTTI interrupt
Motor.h	Header file, contains prototypes of the modules and global variables
myvect.h	Header file, contains the interrupt vector table declaration

4.2 Procedure

There are 3 steps to begin using the Motor.lib C library.

- In F2MC-8L/8FX SOFTUNE, after creation of a new project, use PROJECT –> ADD MEMBER to add MB95F330 Motor Drive V0.2.0.lib and DTTI.c as a member.
- Include Motor.h header file into C main program for external references.
- Include myvect.h header file into the module which uses directive #pragma to generate the interrupt vector table.

Thus, a project including Lib file is ready for the caller program.

5 Library Functions and External Variables

There are 5 global variables in the library:

- Rotation_Direction
- Start_Motor
- Driver_Mode
- Motor_State
- Drive_Level

There are 9 functional modules for library control:

- Motor_Init
- Sensor_Less_Start
- Motor_Parm
- Motor_Set_Change_Speed
- Motor_Stop
- Sensor_Less_Normal_Work
- Hall_Sensor_Start
- Hall_Sensor_Normal_Work
- Motor_Get_Speed

5.1 Function Syntax

Syntax	extern void Motor_Init(void);
Description	Initialize MCU resources to be ready for start and stop commands. • Initialize port configuration. • Initialize multi-function timer resources. • Initialize speed check timer. • Initialize interrupt. • Initialize motor state to MOTOR_READY.
Input parameters	Void
Return	Void

Syntax	extern void Sensor_Less_Start(unsigned short start_duty_on, unsigned short start_period, unsigned short normal_duty_on, unsigned short normal_period);
Description	Start motor from reset with sensorless drive • Motor_State will be MOTOR_NORMAL or MOTOR_FAILURE • Startup and normal run parameters are initialized.
Input parameters	start_duty_on : startup carrier frequency duty on duration in 125ns unit Start_period : startup carrier period in 125ns period unit Normal_duty_on : carrier duty on duration when startup changes to normal run, in 125ns unit normal_duty : carrier period in normal run mode
Return	Void
Example	Sensor_Less_Start(400, 1600, 200, 800); 60us on time during startup = 400 x 125ns => 60000 5kHz carrier frequency => 1600 x 125ns startup carrier period, 25us on time just after startup = 200 x 125ns => 25000 10kHz carrier frequency => 800 x 125ns normal run carrier period

Syntax	extern void Motor_Parm(unsigned long speed_con, unsigned short csd, unsigned short cad,unsigned short camaskt, unsigned short stmaskt);
Description	Define runtime parameters with sensorless drive. • Define speed constant for speed checking • Define commutation delay duration • Define the duration between change-state and change-arm • Define the mask-off period just after change-arm • Define the mask-off period during startup
Input parameters	speed_con= 6000000 / (2us x number of pole pair) csd, in x100 electric angle cad, in x100 electric angle camaskt, in x100 electric angle stmaskt, in 1us unit
Return	Void
Example	Motor_Parm(15000000, 0, 200,200, 2000); 2 pole pair => 60 / (2us x 2) = 15000000 0 change state delay after back EMF zero crossing => 0 2 change-arm delay after back EMF zero crossing => 200 After change arm, mask time =>200 During startup, 2ms = 2000 x 1us => 2000

Syntax	extern void Motor_Set_Change_Speed(unsigned short speed);
Description	Set or change target rotational speed in RPM whenever sensorless drive or hall sensor drive is used.
Input parameters	speed in RPM
Return	Void
Example	Motor_Set_Change_Speed(6000); Set target speed to 6000rpm.

Syntax	extern void Motor_Stop(void);
Description	Stop motor without brake. • All driving outputs are inactivated. • Speed checking timer is stopped. • Multi-function timer is reset. • Input capture edge detection are disabled.
Input parameters	Void
Return	Void

Syntax	extern void Sensor_Less_Normal_Work(void);
Description	Control motor running normally with sensorless drive. • Count change arm time.
Input parameters	Void
Return	Void

Syntax	extern void Hall_Sensor_Start(unsigned short duty_on, unsigned short period);
Description	Start motor from reset with hall sensor drive. • Motor_State will be MOTOR_NORMAL or MOTOR_FAILURE • Parameters are initialized
Input parameters	duty_on :Carrier frequency duty on duration in 125ns unit period : Carrier period in 125ns period unit
Return	Void
Example	Hall_Sensor_Start (150, 800); 18.75us on time during startup = 150 x 125ns => 150 10kHz carrier frequency => 800 x 125ns startup carrier period,

Syntax	extern void Hall_Sensor_Normal_Work(void);
Description	Control motor running normally with hall sensor drive. • Count motor speed. • Control motor speed. • Check hall sensor signal and change arm.
Input parameters	Void
Return	Void

Syntax	extern unsigned int Motor_Get_Speed(void);
Description	Get motor actual speed.
Input parameters	Void
Return	Motor actual speed in RPM.

5.2 External Variables

Variable	extern unsigned char Motor_State
Description	Motor operation mode
Value	MOTOR_READY, 1 : motor ready for accepting start command MOTOR_START, 2 : motor in startup stage MOTOR_NORMAL, 3 : motor in normal run stage MOTOR_FAILURE, 4 : motor which cannot run MOTOR_START_FAILURE, 5 : motor start failed OVER_CURRENT, 6: motor over current

Variable	extern unsigned char Rotation_Direction
Description	Motor running direction
Value	ANTICLOCKWISE, 0: motor anticlockwise running CLOCKWISE, 1: motor clockwise running.

Variable	extern unsigned char Driver_Mode
Description	Motor drive method
Value	HALL_SENSOR, 0: hall sensor drive SENSOR_LESS, 1: sensorless drive.

Variable	extern unsigned char Start_Motor
Description	Start motor signal
Value	FALSE, 0: the motor cannot be started. TRUE, 1: the motor can be started.

Variable	extern unsigned char Drive_Level
Description	Drive motor level choice
Value	Drive_High, 0: high level drive. Drive_Low, 1: low level drive.

6 Usage of Library Functions

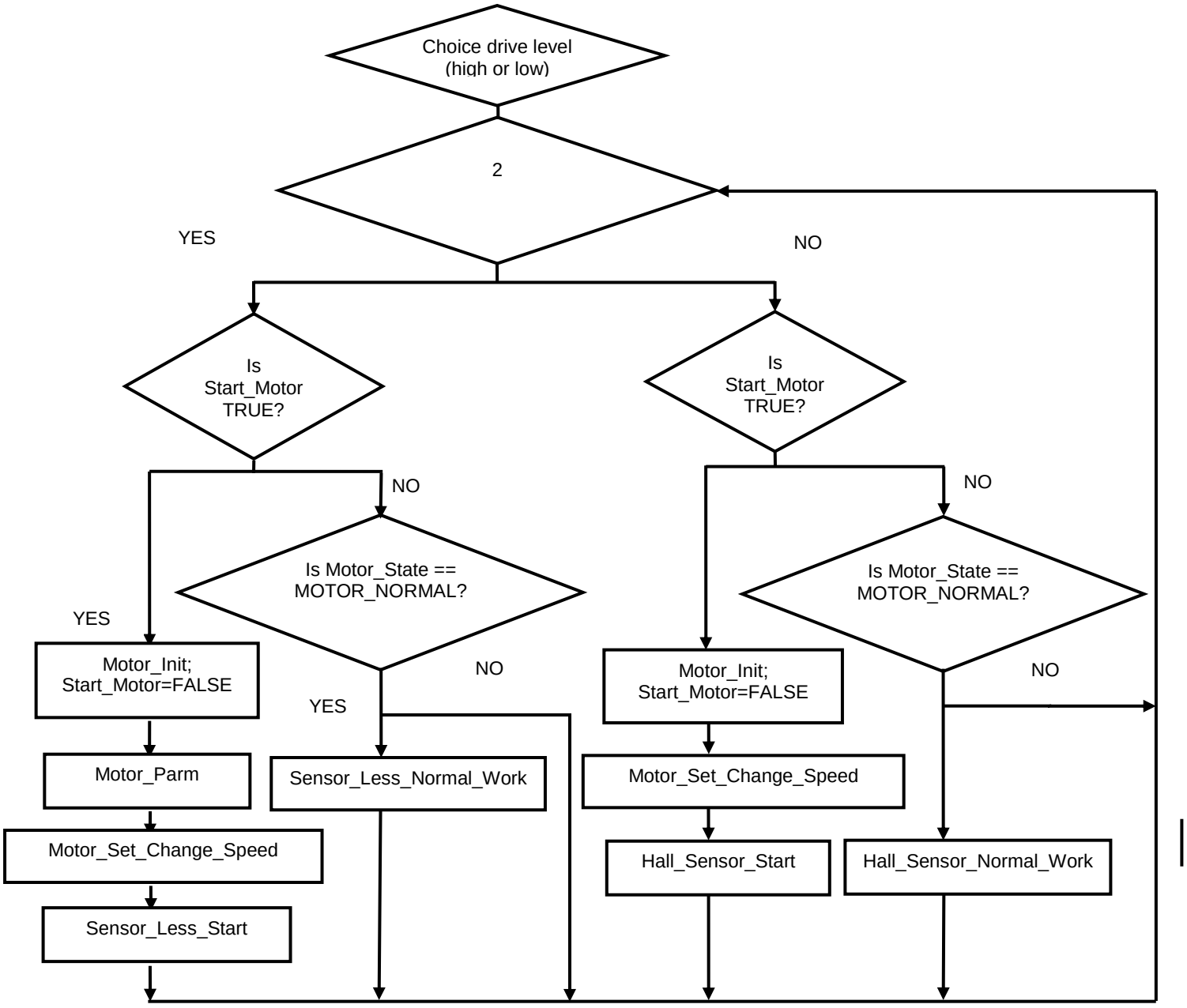
In general, user should follow the following steps to control the motor:

- Set global variables with suitable values.
- Initialize the MCU resource.
- Start the motor with suitable startup speed.
- Modify motor synchronous speed, accelerating speed and decelerating speed by changing values of the global variables.
- Stop the motor.

6.1 Operation Flow

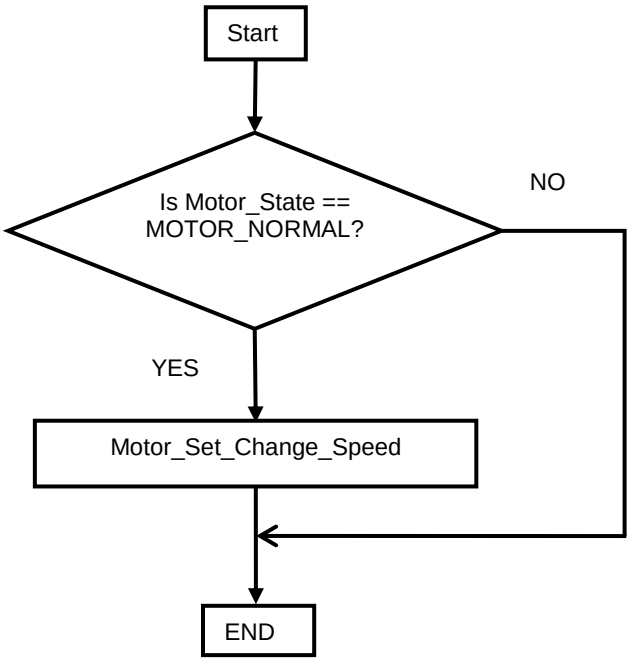
6.1.1 Start Motor

This can be done by calling the following successively using appropriate parameters.


Choice drive level
(high or low)
2
YES
NO
Is
Start_Motor
TRUE?
Is
Start_Motor
TRUE?
NO
NO
Is Motor_State ==
MOTOR_NORMAL?
Is Motor_State ==
MOTOR_NORMAL?
YES
YES
NO
NO
Motor_Init;
Start_Motor=FALSE
Motor_Init;
Start_Motor=FALSE
Motor_Parm
Sensor_Less_Normal_Work
Motor_Set_Change_Speed
Motor_Set_Change_Speed
Hall_Sensor_Start
Hall_Sensor_Normal_Work
Sensor_Less_Start

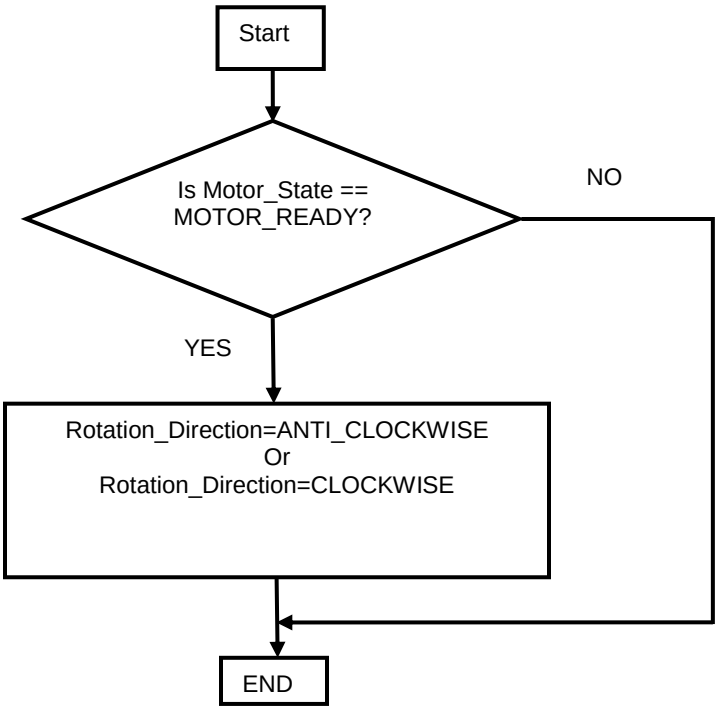
6.1.2 Change Motor Speed

To change motor speed, please ensure that the motor is running under normal status. The following flow chart shows how to change the motor speed:



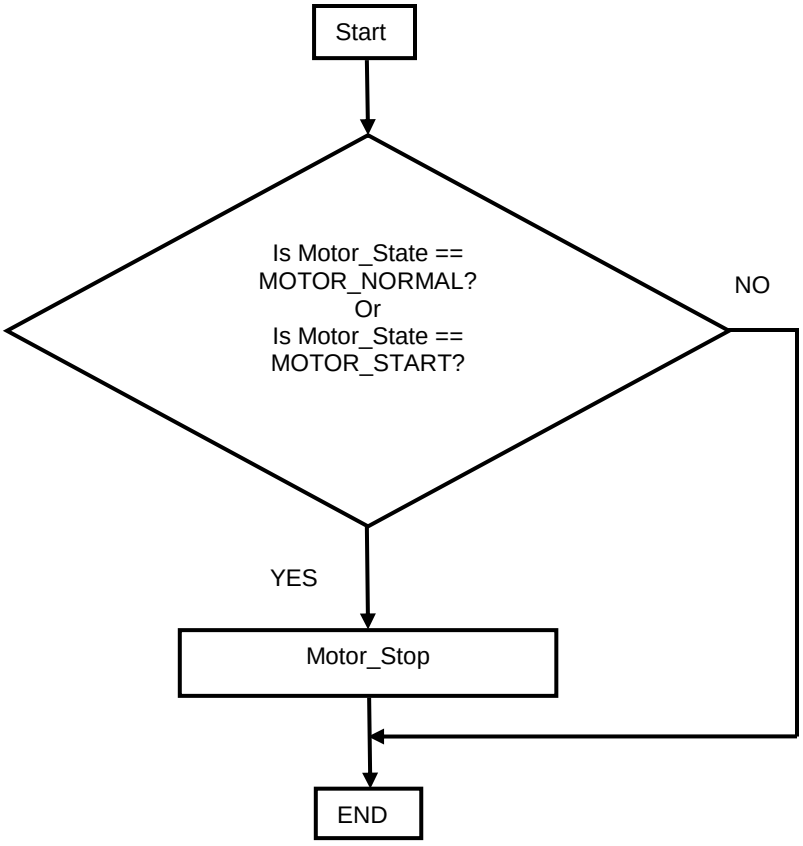
6.1.3 Set Motor Rotation Direction

To set motor rotation direction, please ensure that the motor is under ready status. The following flow chart shows how to set the motor rotation direction.



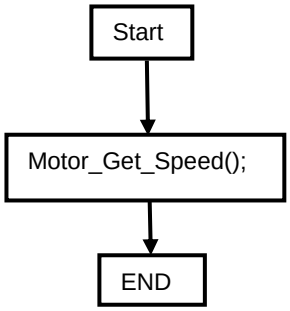
6.1.4 Stop Motor

To stop a motor, please ensure that the motor is under normal or startup status. The following flow chart shows how to stop the motor.



6.1.5 Get Motor Speed

The following flow chart shows how to get motor actual speed.



6.1.6 Adjust Parameter

When we use this library to drive a new motor, there are some parameters need to adjust.

Because of different motors need different parameters.

Adjust carrier frequency

We can use the two functions to adjust motor carrier frequency.

```
Hall_Sensor_Start(unsigned short duty_on, unsigned short period);
```

We can adjust period for motor carrier frequency when we choose hall sensor drive.

```
Sensor_Less_Start( unsigned short start_duty_on,unsigned short start_period,unsigned short normal_duty_on,unsigned short normal_period);
```

When we choose sensorless drive, we can change start_period for adjusting the motor start carrier frequency and we change normal_period for adjusting motor normal run carrier frequency. It should keep the start duty as same as normal duty, otherwise the motor can't start normally.

Adjust start

When we choose sensorless drive, maybe we should adjust some parameters to keep motor start normally. There are two functions need to adjust.

```
Sensor_Less_Start( unsigned short start_duty_on,unsigned short start_period,unsigned short normal_duty_on,unsigned short normal_period);
Motor_Parm(unsigned long speed_con,unsigned short csd, unsigned short cad,unsigned short camaskt, unsigned short stmaskt);
```

We know when we just start motor. There is no Back-EMF can checked. So we use a timer to help motor change arm and bring Back-EMF. But how long we set timer and how large duty need? If the time of the timer is large or small, the motor can't run normally. If the duty is large, the motor will over current and can't start. If the duty is small, the Back-EMF of motor is very weak and we can't check. Usually, we fix one parameter, adjust another one. For example, we fixed start_duty_on(start duty) , then we change stmaskt(timer). We also can fix stamaskt(timer), then we change start_duty_on(start duty). In order to keep motor can change start state to normal state, When we changed start_duty_on, we should change normal_duty_on to keep the start duty as same as normal duty.

Set speed

Because of different motors have different numbers of pole pair.

So we should change this function for counting motor speed.

```
Motor_Parm(unsigned long speed_con,unsigned short csd, unsigned short cad,unsigned short camaskt, unsigned short stmaskt);
```

We should change speed_con according to numbers of pole pair by the follow formula.

speed_con= 6000000 / (2x number of pole pair)

Improve efficiency

Sometimes, we find the efficiency is very low. We can adjust three parameters of Motor_Parm function.

```
Motor_Parm(unsigned long speed_con,unsigned short csd, unsigned short cad,unsigned short camaskt, unsigned short stmaskt);
```

7 Sample Program

Motor.zip is a sample project containing source code which can drive a sensorless brushless or hall sensor DC motor with motor EV Board (PN: MB2146-440-E V1.2). Please refer to Motor EV Board MB2146-440-E HW User Manual.

Tested configuration:

DC motor: Fulling FL28BL26-15V-8006AF

Number of phases: 3

Number of poles: 4

Supply voltage: 15VDC

Minimum tested speed: 1000rpm

Maximum tested speed: 7000rpm

MCU work load: 8%~30% (Motor speed from 1000 rpm to 7000 rpm with sensorless drive);

2%~10% (Motor speed from 1000 rpm to 7000 rpm with hall sensor drive);

8 Additional Information

For more information on how to use MB9595330 EV Board, BGM adaptor and SOFTUNE, please refer to Motor EV Board MB2146-440-E HW User Manual or visit Websites:

http://www.cypress.com/documentation/application-notes/mb95330-mb2146-440-e-motor-evb-user-manual.

For more information on softune library, visit our website:

http://www.cypress.com/documentation/software-and-drivers/softune-ide-development-support-tools

http://www.cypress.com/documentation/software-and-drivers/softune-workbench

Document History

Document Title: AN204945 - F²MC-8FX Family MB95330 Series 120° Hall Sensor/Sensorless DC Inverter Control Softune C Library

Document Number: 002-04945

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	–	HUAL	11/20/2009	First draft.
			11/30/2009	Modify
			01/11/2010	Modify
			08/03/2010	Modify 4.1
			03/20/2010	Update motor drive waveform and back waveform
			12/07/2010	Add HW description Add DTTI.c file Add drive level choice function Add get motor speed function Add how to adjust parameter Add more motor fail indicate
*A	5265773	CHMA	09/13/2016	Migrated Spansion Application Note “MCU-AN-500067-E-14” to Cypress format.
*B	5844483	AESATMP9	08/04/2017	Updated logo and copyright.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709