AN-8205
AMC 库霍尔接口

概括

FCM8531 是一款用于电机控制的特定应用的双核心处理器，由先进电机控制器 (AMC) 处理器和与 MCS®51 兼容型的 MCU 处理器组成。AMC 是专为电机控制设计的核心处理器。它集成了可配置的处理内核和外设电路以实现无传感器磁场定向控制 (FOC) 电机控制。系统控制、用户界面、通信接口和输入/输出接口可通过嵌入式 MCS®51 针对电机应用进行编程。

FCM8531 双核心处理器的优点是，这两个处理器可独立工作且相互补充。AMC 专用于电机控制应用，如电机控制算法、PWM 控制、电流感测、实时过流保护和电机相角计算。嵌入式 MCU 向 AMC 提供电机控制命令，通过通信接口来控制电机。由于复杂电机控制算法在 AMC 中执行，因此，这种方法可减少软件负担并简化控制系统程序。Fairchild 提供电机控制开发系统 (MCDS) 集成式开发环境 (IDE) 和 MCDS 编程套件，以便用户开发软件、执行系统内编程 (ISP) 以及执行在线调试。

典型的 FCM8531 开发环境配置如图 1 所示。应用板可采用 FCM8531 评估板或用户自定义电路板（称为“目标板”）。FCM8531 评估板可与 Fairchild 提供的 MCDS IDE 和 MCDS 编程套件一起使用，帮助您开发电机应用产品。

MCDS IDE 可在 Microsoft® Windows 操作系统上运行，包含项目管理、AMC 库选择、寄存器设置和编译器/链接器/调试器链接等功能，帮助您开发软件。通过使用 Fairchild 提供的 AMC 库，用户可轻松快速开发无传感器的电机驱动，并缩短电机应用的开发时间。本文是 AMC 库的用户指南。有关 *MCDS IDE* 和 *MCDS* 编程套件的详情，请参考 *Fairchild* 网站：

http://www.fairchildsemi.com/applications/motorcontrol/bldc-pmsm-controller.html

AMC 简介

内核处理器 AMC 由多个电机控制模组组成，例如，可配置处理核心、PWM 引擎和角度预测器。根据应用，处理核心可配置合适的 AMC 库以执行不同的电机控制算法，如磁场定向控制 (FOC) 或无传感器控制。

FCM8531 可驱动带霍尔传感器或无传感器库的电机。霍尔接口库是 AMC 库，用于驱动带霍尔传感器的电机。本文主要介绍霍尔接口库的使用，包括控制理论、配置、文件和固件示例。有关用 *AMC* 库实现无传感器控制的更多详细信息，请参考以下网站：http://www.fairchildsemi.com/applications/motorcontrol/bldc-pmsm-controller.html

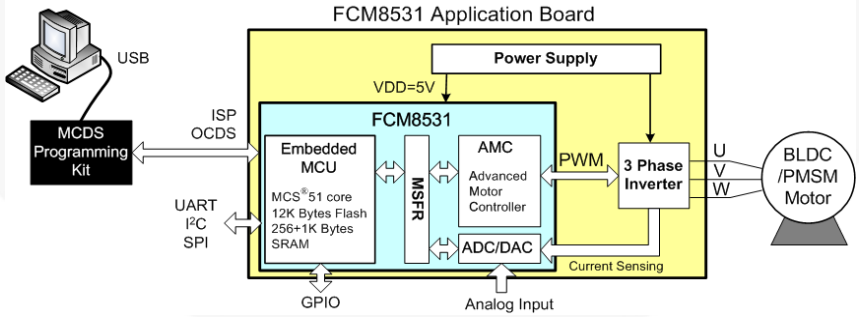

图 1. 典型开发环境

霍尔接口

霍尔接口库通过 MCDS IDE 的项目设置窗口进行选择，如图 2 所示。

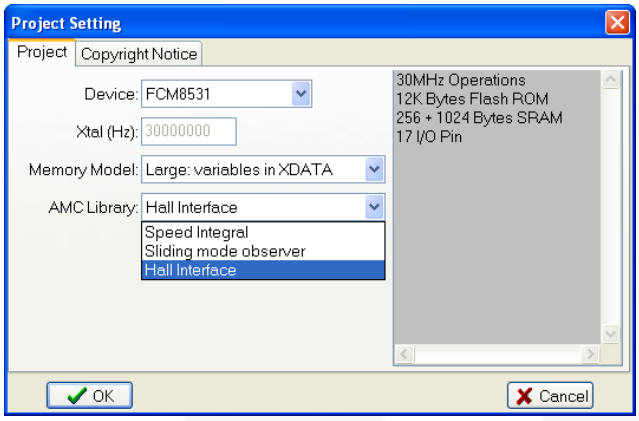

图 2. MCDS IDE：项目设置

控制理论

角度预测器（参见图 3 ）是霍尔接口库的主要处理器；它包括霍尔信号过滤器、锁相环 (PLL)、领先角度位移器和角度编码器。霍尔信号过滤器用于处理霍尔信号和滤除输入信号的噪声。PLL 检测每 60 度的电气角度中霍尔信号的变化，以预测转子位置。角度编码器将 PLL 结果和领先角度位移器设置合计为一个角度，然后 PWM引擎根据角度编码器的角度输出 PWM 信号。欲了解每个模块的更多信息，请参考 AN-8202 — FCM8531 用户手册-硬件说明。

PWM 控制模式可分为两种模式：正弦波模式和方波模式。默认模式是正弦波模式。当电机以正弦波模式启动时，方波 PWM 输出用于初始驱动电机。直至电机的相位被 PLL 锁定时，正弦波 PWM 输出自动转换到驱动电机。

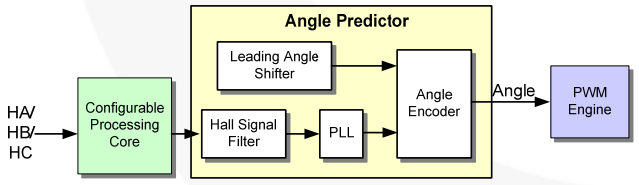

图 3. 角度预测器

配置

FCM8531 用于驱动霍尔传感器电机之前，AMC 需要配置为霍尔接口。在 MCDS IDE 中必须设置一些参数，如角度预测器设置、PWM、霍尔信号保护和霍尔模式。

MCDS IDE 的操作可参考文件：*AN-8207—FCM853 的 MCDS IDE 用户指南 1*。

PWM

PWM 参数如图 4 所示，包括输出波形、SAW 类型、PWM 频率、死区时间等等。

波形：

有两种可选波形：正弦波和方波。

SAW 类型：

有三种可选类型：上-下类型（默认）、向上类型和向下类型。

PWM 频率：

PWM 频率可通过 SPRD 参数进行调节，建议为 15 kHz 至 20 kHz，以获得较大的 SAW 分辨率并避开音频范围。前除器和后除器都选择为 “div 1”。

死区时间：

死区时间需要根据电源规范进行调节。图 4 显示的设置值适用于演示。

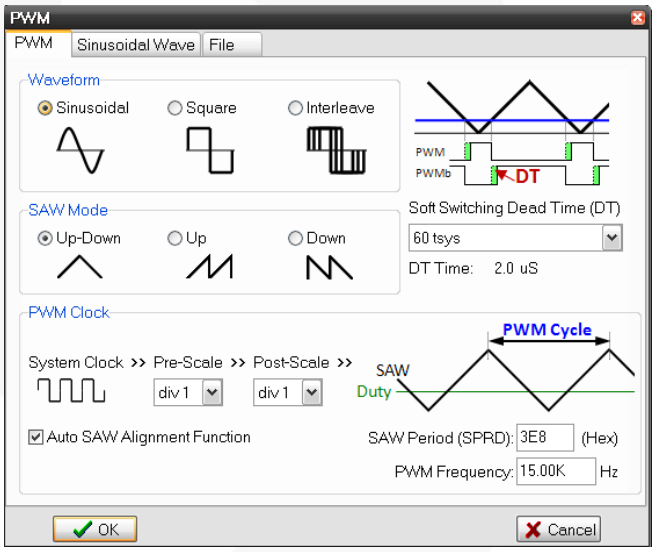

图 4. PWM 设置

霍尔信号

霍尔信号滤波器的选项可在霍尔对话框中设置，如图 5 所示。

霍尔信号中断使能 (EX10)：

当霍尔信号变化时，可选择霍尔信号中断使能 (EX10) 以产生针对 MCU 的中断请求。有三种中断触发器类型：上升沿触发器、下降沿触发器和上升/下降沿触发器。

霍尔反相：

如果霍尔信号的极性不同于 FCM8531 中的定义，霍尔 A/B/C 输入反相能够反转霍尔信号，以确保角度预测器正确。

霍尔信号消隐时间和去抖动时间：

霍尔信号消隐时间和去抖动时间用于防止霍尔信号内噪声引起的 PLL 故障。默认值为上述两个时间的最小值。

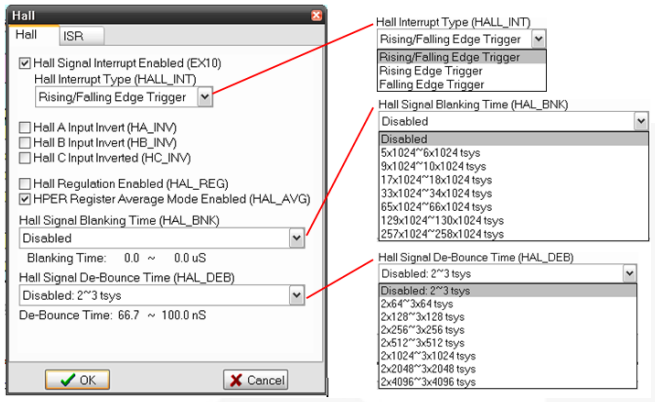

图 5. 霍尔信号设置

保护

设置保护选项，确定在发生霍尔信号错误或超时情况下，是否产生中断请求以通知 MCU。

霍尔错误事件使能：

当 HA、HB 和 HC 信号都是高电平 (111) 或都是低电平 (000) 时，可判断霍尔信号错误并立即关闭 PWM 输出。如果已选择该选项，一旦霍尔信号出现错误，就会产生中断请求 (EX8) 以通知 MCU。

霍尔信号超时：

霍尔周期寄存器 (HPER)（即角度预测器中的一个 20 位定时器），用于计算霍尔信号变化的时间间隔。长时间间隔通常表示电机以极低的速度运转或已经停止运转。如果选择此选项，HPER 的时间会被监控。如果发生超时，HPER 大于 1FFFFh、3FFFFh 或 7FFFFh；产生针对 MCU 的中断请求 (EX8)。

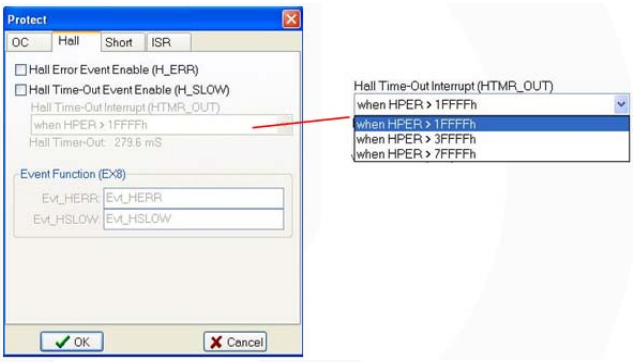

图 6. 保护设置

霍尔信号输入

两种可选霍尔信号输入引脚，如表 1 所示。

表 1. 霍尔信号输入

	配置 1 (40 h)	配置 2 (80 h)
HA	P14	P24
HB	P15	P25
HC	P16	P26

子程序 **btInitial_AMCToHallSignal()** 将 AMC 定义为霍尔接口驱动，并在 AMC 中设置霍尔信号输入选择。在主程序调用子程序之前，需要设置 **_bHall_Type** 的内容，默认值为 **HALL_PINCFG0**。子程序的示例如下。

```
:
:
Initial_Procedure();
//User program start here.(02)
_bHall_Type= HALL_PINCFG0;     //HA=P14, HB= P15, HC=P16
// _bHall_Type= HALL_PINCFG1;  //HA= P24, HB= P25, HC= P26

//User program end here.(02)
EA = 1;
btInit_AMCStatus = btInitial_AMCToHallSignal();   // Initial AMC
:
:
```

子程序 **btInitial_AMCToHallSignal()** 的处理流程通过子程序 **bWriteCmdToAMC()** 来执行，并发送霍尔模式至 AMC：

```
bWriteCmdToAMC(CMD_HALL_PIN, 0, _bHall_Type);
```

通信

通信接口

MCU 和 AMC 之间的通信通过邮箱寄存器进行传输。AMC 通过 MTX0(B0h) -MTX3(B3) 从 MCU 接收控制命令和数据，控制相应的电机；MCU 通过 MRX0(B4h) - MRX3(B7h) 读取 AMC 传送过来的数据，如图 7 所示。

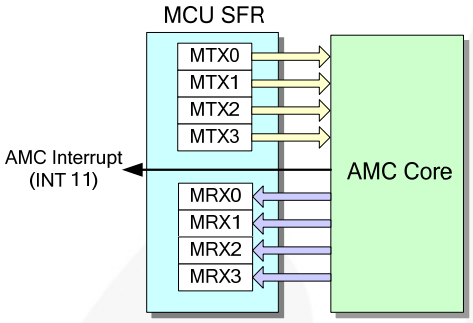

图 7. 通信接口

邮箱寄存器定义

本节介绍了用于通信的每个寄存器。

MCU 数据传输 (MTX0 - MTX3)

MTX0:

b0（触发位）：

传输至 AMC 的命令和数据存储在相关寄存器中；b0 位从 1 转换为 0 可启动传输。

b1–b7（命令）：

存储已传输至 AMC 的命令。

MTX1:

b0–b7（高位数据字节）：

存储已传输至 AMC 的数据的高位字节。

MTX2:

b0–b7（低位数据字节）：

存储已传输至 AMC 的数据的低位字节。

MTX3:

b0 (ForwardFreeRun)：

确定电机正向启动：1 =正向，0 = 非正向。

b1 （反向空闲运行）：

确定电机反向启动：1 = 反向，0 = 非反向。

b2–b7（保留）：

设为 0。

MCU 数据读取 (MRX0 - MRX3)

MRX0:

b0：保留

b1 (AMC_Cmd)：

AMC_Cmd=1：AMC 忙于处理命令，不能接收新命令。

AMC_Cmd=0：AMC 能接收新命令。

b2 (AMC_Cal)：

AMC_Cal=1：AMC 忙碌，不能接收新命令。
AMC_Cal=0：AMC 不忙碌，能接收新命令。

b3 (AMC_Fault)：

AMC_Fault=1：AMC 运行异常。当发生 AMC_ 故障时，MCU 能够执行“重试”或“停止”操作。

AMC_Fault=0：AMC 运行正常。

b4 (STR_Rdy)：

STR_Rdy=1：AMC 已完成启动步骤。MCU 可进行随后的通用电机控制。

STR_Rdy=0：AMC 未完成启动步骤。

b5 (Al_Rdy)：

Al_Rdy=1：AMC已完成对齐步骤；可启动随后的斜升控制。

Al_Rdy=0：AMC 未完成对齐步骤。

b6 (RST_Rdy)：

RST_Rdy=1：AMC 已完成复位操作；MCU 可以开始传输和读取数据。

RST_Rdy=0：AMC 已完成复位操作。

b7：(SAFETY_Warning):

SAFETY_Warning=1：根据 UL60730 规则，AMC 发生故障。

SAFETY_Warning=0：根据 UL60730 规则，AMC 未发生故障。

MRX1:

b0–b7（高位数据字节）：

存储从 AMC 传输的数据的 高位字节。

MRX2:

b0–b7（低位数据字节）：

存储从 AMC 传输的数据的 低位字节。

表 2. 邮箱寄存器定义

字节名称（地址）	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MTX0 (B0h)	命令							触发
MTX1 (B1h)	高位数据字节							
MTX2 (B2h)	低位数据字节							
MTX3 (B3h)	保留							正向
MRX0 (B4h)	保留	RST_Rdy	Al_Rdy	STR_Rdy	AMC_Fault	AMC_Cal	AMC_Cmd	保留
MRX1 (B5h)	高位数据字节							
MRX2 (B6h)	低位数据字节							
MRX3 (B7h)	保留							

通信协议

必须完全执行正确的 通信协议和 MCU 与 AMC 之间的流向，以避免传输错误。通信协议和流向的正确步骤说明和显示如下。

MCU 将命令写入 AMC

步骤 1. 检查 AMC 是否忙碌。

若结果为“是”，则重复执行步骤 1，直到超时。

若结果为“非”，执行步骤 2。

步骤 2. 在相应寄存器中存储命令和数据，然后将 MXT0 的 b0 从 1 转换为 0，以开始传输。

步骤 3. 等待 20 µs。

步骤 4. 检查 AMC 是否已完成处理命令，即：

AMC_Cmd=1.

若结果为“是”，则重复执行步骤 4，直到超时。

若结果为“非”，则传输完成。

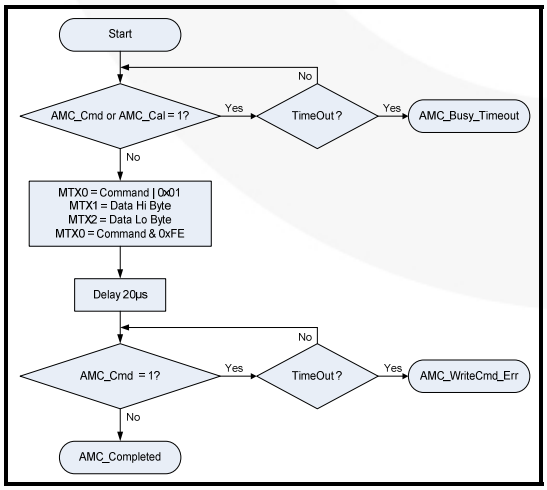

图 8. 写入命令协议

内置函数

为便于使用，当在 MCDS IDE 软件中创建新项目时，生成命令传输函数。该函数用于将数据从 MCU 传输至 AMC，以缩短编码时间。该函数如下所述。

bWriteCmdToAMC(U8 Command，U8 Data_High_Byte，U8 Data_Low_Byte)

其中，

命令：与已传输数据对应的命令。

Data_High_Byte：已传输数据的高位字节。

Data_Low_Byte：已传输数据的低位字节。

示例：

要将 PWM 占空比设置为 0x50，应写入：

bWriteCmdToAMC(CMD_DUTY，0x0，0x50)

执行后；如果函数返回值为 0，则数据传输成功。

命令索引

命令和相关参数如表 3 所示。

表 3. 命令和参数

命令	索引	参数
CMD_RUN	0x10	1：RUN 0：FREE
CMD_DUTY	0x12	占空比 (0~ 511)
CMD_ANGLE_SHIFT	0x14	角度位移值 (0~ 127)
CMD_HALL_PIN	0x80	霍尔接口引脚配置： 0x40：HA/HB/HC= P14/P15/P16 0x80：HA/HB/HC= P24/P25/P26

霍尔接口文件

使用 MCDS IDE 生成新项目和代码后，MCDS IDE 可自动生成 2 个霍尔接口文件。表 4 显示的是这些文件的文件名称和函数。

表 4. 霍尔接口文件

文件名	说明	可编辑
AMC_HallInterface.c	AMC_HallInterface.c 提供通用函数，如 WriteCmdToAMC()、ReadDataFromAMC()、Initial_AMCToSpeedIntegral() 等等。用户不可修改此文件。MCDS IDE 的生成代码函数一旦使用，该文件即被重写。	否
AMC_HallInterface.h	AMC_HallInterface.h 提供所有命令的定义以及读取数据时索引的定义。用户不可修改此文件。MCDS IDE 的生成代码函数一旦使用，该文件即被重写。	否

AMC_HallInterface.c

为便于使用，AMC_HallInterface.c 包含数个通用函数，如表 5 所示。

表 5. AMC_Hallsignal.c 通用函数

函数名称	说明
AMC_Reset( )	复位AMC
btInitial_AMCToHallInterface()	初始化 AMC 并将 AMC 设置为霍尔接口模式 结果： 0：通过 1：AMC 初始化失败
bWriteCmdToAMC（命令、数据 *(H)*、数据 *(L)*）	将数据传输至 AMC。 结果： 00：通过 02：传输错误
延迟10 µs（值）	在 10 µs 延迟时间内等待 AMC
延迟1 ms（值）	在 1 ms 延迟时间内等待 AMC

AMC_HallInterface.c 的内容如下：

线路 13： 将霍尔接口设置为模式 0（默认；HA= P14，HB= P15，HC= P16）

线路 18 ~ 22： 子程序用于复位 AMC

线路 27 ~ 49： 子程序用于初始化 AMC 和检查 AMC 状态

线路 55 ~ 68： 子程序用于发送 HALL MODE 和 TRANS COMPLETE 至 AMC

线路 72 ~ 99： 子程序用于发送一条命令至 AMC 并检查 AMC 状态

线路 104 ~ 110：子程序用于延迟 10 µs

线路 112 ~ 118：子程序用于延迟 1 ms

```
1       /*---------------------------------------------------------------------
2        * Copyright 2012 Fairchild Semiconductor
3        * http://www.fairchildsemi.com
4        *---------------------------------------------------------------------
5        */
6       #include "compiler-define.h"
7       #include "FCM8531.h"
8       #include "MSFR-define.h"
9       #include "Program.h"
10      #include "MCS51.h"
11      #include "AMC_HallInterface.h"
12
13      U8 _bHall_Type= HALL_MODE0;
14
15      //---------------------------------------------------------------------
16      //  AMC Reset Function
17      //---------------------------------------------------------------------
18      void Reset_AMC()
19      {
20          WRITE_MSFR(MSFR_MCNTL, 0x40);
21          WRITE_MSFR(MSFR_MCNTL, 0x00);
22      }
23
24      //---------------------------------------------------------------------
25      //  Initail AMC Procedure
26      //---------------------------------------------------------------------
27      SEG_BIT btInitial_AMCToHallInterface()
28      {
29          SEG_BIT btError_code;
30          U16 wWaitTime;
31
32          Reset_AMC();  // Reset AMC
33
34          Delay1ms(100);
35
36          btError_code = 1;
37          for(wWaitTime = 0; wWaitTime < 3000; wWaitTime++)
38              if((MRX0 & AMC_RESET_READY) == 0x40)
39              {
40                  btError_code = 0;
41                  break;
42              }
43          if(btError_code)   // Check Time-Out
44              return(btError_code);
45
46          btError_code = btTransmit_ParameterToAMC();
47          Delay1ms(1000);
48          return btError_code;
49      }
50
51
52      //---------------------------------------------------------------------
53      //  Deliver Parameter for Hall interface used
54      //---------------------------------------------------------------------
55      SEG_BIT btTransmit_ParameterToAMC()
```

```
56      {
57          U8 bWriteCMD_Status;
58
59          bWriteCMD_Status = bWriteCmdToAMC(CMD_HALL_PIN, 0, _bHall_Type);
60          if(bWriteCMD_Status)
61              return 1;
62
63          bWriteCMD_Status = bWriteCmdToAMC(CMD_TRANS_COMPLETE, 0, 0);
64          if(bWriteCMD_Status)
65              return 1;
66
67          return 0;
68      }
69      //---------------------------------------------------------------------
70      //  MPU write CMD to AMC
71      //---------------------------------------------------------------------
72      U8 bWriteCmdToAMC(U8 bCommand, U8 bData_HB, U8 bData_LB)
73      {
74          U16 wWaitTime;
75          SEG_BIT btAMCStandby_Flag;
76
77          btAMCStandby_Flag = 0;
78          for(wWaitTime = 0; wWaitTime < 600; wWaitTime++)
79              if((MRX0 & (AMC_PROCESSING_CMD | AMC_CALCULATING)) == 0x0)
80              {
81                  btAMCStandby_Flag = 1;
82                  break;
83              }
84          if(!btAMCStandby_Flag)  // Check Time-Out
85              return(AMC_BUSY_TIMEOUT);
86
87          MTX0 = bCommand | MCU_MAILBOX_INTR_STOP;
88          MTX1 = bData_HB;
89          MTX2 = bData_LB;
90          MTX0 = bCommand & MCU_MAILBOX_INTR_START;
91
92          Delay10us(2);
93
94          for(wWaitTime = 0; wWaitTime < 600; wWaitTime++)
95              if((MRX0 & AMC_PROCESSING_CMD) == 0)
96                  return(AMC_COMPLETED);
97
98          return(AMC_WRITE_CMD_ERROR);
99      }
100
101     //---------------------------------------------------------------------
102     //  Delay time routine
103     //---------------------------------------------------------------------
104     void Delay10us(U16 Counter)  //Delay 10us
105     {
106         U16 i, k;
107
108         for(i = 0; i < Counter; i++)
109             for(k = 0; k < 16; k++);
110     }
111
112     void Delay1ms(U16 Counter)  //Delay 1ms
113     {
114                 U16 i;
115
116         for(i = 0; i < Counter; i++)
117             Delay10us(110);
118     }
```

AMC_HallInterface.h

AMC_HallInterface.h 包含命令、参数、AMC 数据的索引。

AMC_HallInterface.h 的内容如下。

其中，

线路 18 ~ 22： 指示邮箱的状态；

线路 22 ~ 46： 指示 AMC 的命令；

线路 59 ~ 68： 指示 AMC 的状态；

```
1     #ifndef  _AMC_HallInterface_h_
2     #define  _AMC_HallInterface_h_
3
4     //for IEC-60730
5     #define LOCK_ROTOR_DELAY_TIME     10  // Resolution：1 sec/step，1 means 1s
6     #define OVER_CURRENT_MUTE_TIME    10  // Resolution：0.5 sec/step，1 means 0.5s
7
8     // MCU mail box status
9     #define MCU_MAILBOX_INTR_STOP     0x01
10    #define MCU_MAILBOX_INTR_START    0xFE
11
12    // MRX0 bit description
13    #define AMC_PROCESSING_CMD        0x02
14    #define AMC_CALCULATING           0x04
15    #define AMC_FAULT                 0x08
16    #define AMC_STARTUP_READY         0x10
17    #define AMC_LOCK_MOT_READY        0x20
18    #define AMC_RESET_READY           0x40
19    #define SAFETY_WARNING            0x80
20
21    // MCU CMD list//
22    // Write data CMD
23    #define CMD_AMC_CONTROL          0x10
24    #define CMD_DUTY                  0x12
25    #define CMD_ANGLE_SHIFT           0x14
26    #define CMD_ALIGNMENT_TIME        0x16
27    #define CMD_ALIGNMENT_WAIT_TIME   0x18
28    #define CMD_ALIGNMENT_DUTY        0x1A
29    #define CMD_ALIGNMENT_ANGLE       0x1C
30    #define CMD_RAMPUP_START_DUTY     0x1E
31    #define CMD_RAMPUP_END_DUTY       0x20
32    #define CMD_RAMPUP_ACC_TIME       0x22
33    #define CMD_FORWARD_RUN_SPEED     0x24
34    #define CMD_KP                    0x26
35    #define CMD_KI                    0x28
36    #define CMD_DIGITAL_FILTER        0x2A
37    #define CMD_LOSS_STEP             0x2C
38    #define CMD_RESOLUTION            0x2E
39    #define CMD_MINIMUM_SPEED         0x30
40    #define CMD_RAMPUP_ID_ERR         0x32
41    #define CMD_LOCK_ROTOR_DELAY_TIME 0x34
42    #define CMD_OC_MUTE_TIME          0x36
43    #define CMD_SHUNT_OFFSET          0x38
44    #define CMD_HALL_PIN              0x80
45    #define CMD_ENABLE_WATCHDOG       0xFA            // Watchdog enable command
46    #define CMD_TRANS_COMPLETE        0xFE
47    //Read data CMD
48    #define CMD_MCU_READ_AMC_DATA     0xFC
49
50    // MCU Read Data Index
51    #define AMC_DATA_CORE_ID          0x00
52    #define AMC_DATA_CORE_VERSION     0x01
```

```
53    #define AMC_DATA_THETA            0x02
54    #define AMC_DATA_KP               0x03
55    #define AMC_DATA_KI               0x04
56    #define AMC_DATA_ID_ERR           0x05
57    #define AMC_DATA_FW_SPEED         0x06
58
59    // AMC status code
60    #define AMC_COMPLETED             0x00
61    #define AMC_BUSY_TIMEOUT          0x01
62    #define AMC_WRITE_CMD_ERROR       0x02
63    #define AMC_RUNCMD_TIMEOUT        0x03
64
65    #define AMC_ERRCODE_PASS          0x80
66    #define AMC_ERRCODE_BUSY_TOUT     0x81
67    #define AMC_ERRCODE_CMD_ERROR     0x82
68    #define AMC_ERRCODE_RUNCMD_TOUT   0x83
69
70    // Motor Contril define
71    #define MOTOR_FREE                0x00
72    #define MOTOR_RUN                 0x01
73    #define MOTOR_CW                  0x02
74    #define USER_DEFINE_SVM_TABLE     0x20
75
76    #define CW                        0x01
77    #define CCW                       0x00
78
79    #define HALL_PINCFG1       0x80  // HA= P24, HB= P25, HC= P26
80    #define HALL_PINCFG0       0x40  // HA= P14, HB= P15, HC= P16
81
82    //---------------------------------------------------------------------------
83    //  Export Function
84    //---------------------------------------------------------------------------
85    extern U8 _bHall_Type;
86    extern SEG_BIT btTransmit_ParameterToAMC();
87
88    extern SEG_CODE U8 AS_Table[];
89
90    extern SEG_BIT btInitial_AMCToHallInterface();
91    extern U8 bWriteCmdToAMC(U8 bCommand, U8 bData_HB, U8 bData_LB);
92    extern void Delay10us(unsigned int Counter);
93    extern void Delay1ms(unsigned int Counter);
94
95    #endif
```

使用霍尔接口

使用霍尔接口库来控制带霍尔传感器的电机，根据 MCDS IDE 中所附的示例程序 *Sample_Hall_Interface* 进行详细介绍。

功能

示例程序中的主函数是：

- 霍尔传感器输入
- 占空比由 VR (0 ~ 4 V) 控制
- 正弦波电流驱动
- FO（每转 3 个脉冲，参见图 9）
- 方向控制
- 电流保护（逐周期限流）
- 短路保护

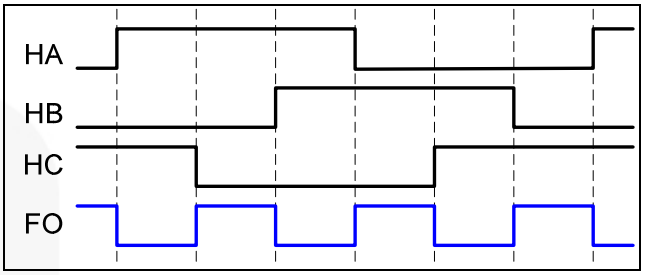

图 9. FO 输出

表 6. 引脚分配

引脚	类型	功能	说明
P14	输入	HA 输入	输入范围 = 0~ 5 V
P15	输入	HB 输入	
P16	输入	HC 输入	
P10	输出	FO	FO 输出，每转 3 个脉冲
P25	输入	方向控制	0：CW 1：CCW
P26	输出	LED 状态	关：系统就绪 开：OCP 保护 闪存：AMC 初始化失败
ADC0	输入	速度控制	输入范围 = 0~4 V < 0.156 V：电机停止 > 0.156 V：电机启动
AOUT	输出	角度输出	0~4 V

硬件电路

除配置部分的设置外；示例程序中的已用 I/O 引脚、定时器和中断也需要设置。这些设置如下所述。

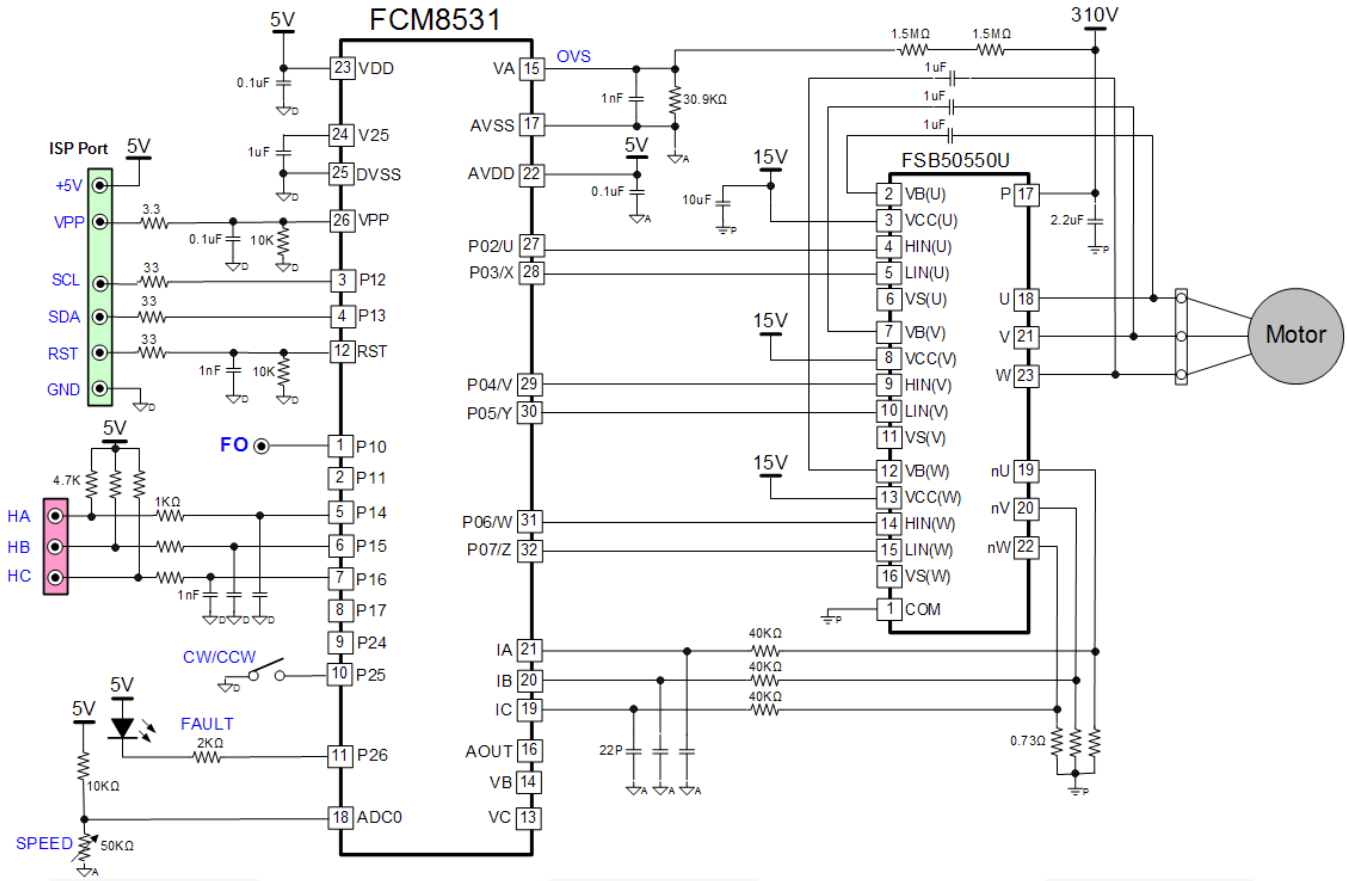

图 10. 电路图

IO 设置

请参考图 10 和表 6。因为专为输出信号而设计，P10 被设置为“输出直接驱动”，引脚名称为 FO。因为专为方向控制信号输入而设计，P25 被设置为“开路漏极”，引脚名称为 DIR。因为专为 LED 控制信号输出而设计，P26 被设置为“开路漏极”，引脚名称为 nLED。

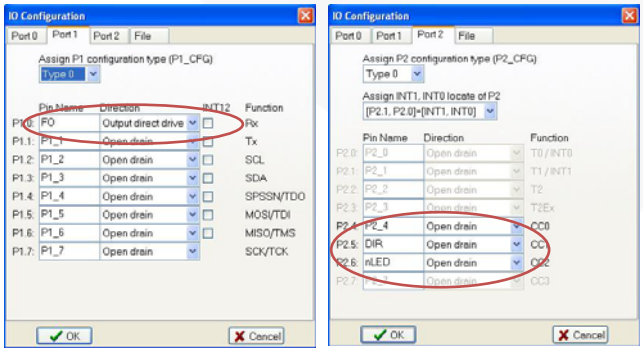

图 11. I/O 配置设置

定时器设置

示例程序中使用 TIMER0 作为提升/降低斜率的基础，设为 26.2 ms。

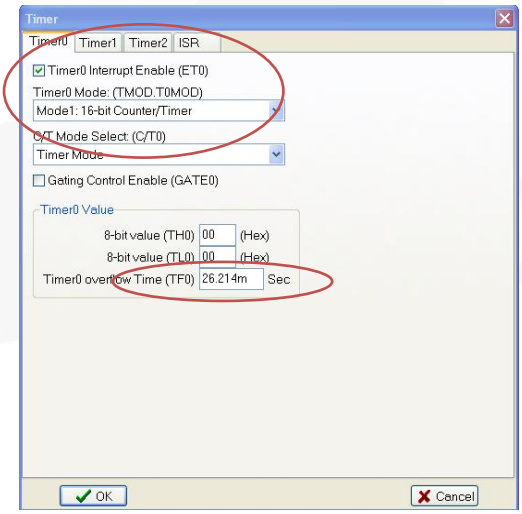

图 12. Timer0 设置

中断设置

示例程序中的霍尔中断函数用于生成 FO 信号，霍尔中断函数 (INT10) 必须使能，触发器类型设为上升/下降沿触发器。

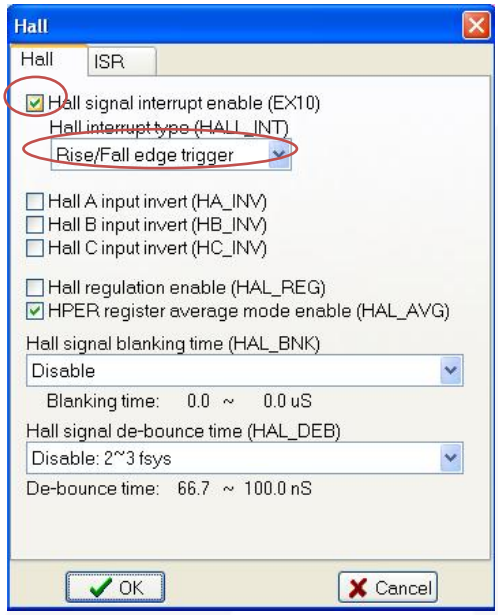

图 13. 霍尔配置设置

ADC设置

示例程序的 ADC0 (VSP) 用于速度控制输入，ADC 中断 (INT9) 必须使能，采样类型设为 SAW 顶部。ADC 转换完成之后，产生 ADC 中断 (INT9) 以读取用于控制速度的 ADC0 (VSP)。

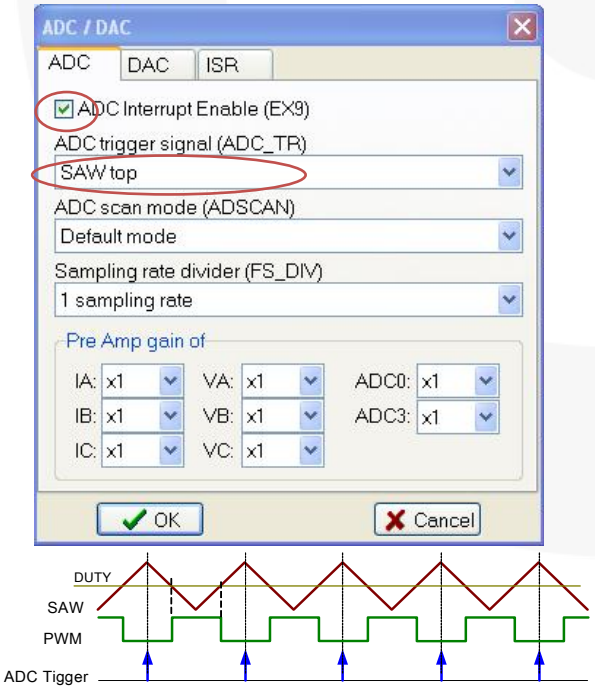

图 14. ADC 配置

固件示例

主程序中的初始子程序用于设置 PWM 频率、保护点、中断参数等的初始值。通过初始子程序可复位 AMC 的启动参数并测试 AMC 与 MCU 之间的通信。初始子程序仅在主程序开始时执行，然后执行无限循环。

ADC 对每个 PWM 周期采样，并在转换器完成后生成中断请求。在中断服务例程中，ADC0 值存储在 **_wTarget_Duty**。主程序判断 **_wTarget_Duty** 是否大于 **MOTOR_RAMPUP_DUTY**。如果大于后者，PWM 将导通以运转电机。电机旋转方向依据 DIR 开关设置来确定，MCNTL (MSFR 00h) 的 RUN 位置位。如果 **_wTarget_Duty** 小于或等于 **MOTOR_RAMPUP_DUTY**；MCTRL、占空比和 AS 清零，电机保持空闲状态，ADC0 值被连续检测。

超时值由 TIMER0 设置为约 26.2 ms。当主程序中产生超时，计数值增加 1。如果计数值大于或等于 **SPEED_LOOP_RATE**，更新占空比和 AS。

完整的源代码位于 IDE 安装盘的下列目录中：*\Fairchildsemi\MCDS\Examples\keilC\Sample_Hall_Interface.*（对于 *Keil C*）

\Fairchildsemi\MCDS\Examples\SDCC\Sample_Hall_Interface.（对于*SDCC*）

Main.c

Main.c 与系统处理控制相关。程序说明如下：

线路 4： 系统时钟设置为 30 MHz；

线路 7 ~12： 定义常量；其中：

MAX_SPEED：ADC0 的最大值；

MIN_SPEED：ADC0 的最小值；

SPEED_LOOP_RATE：更新占空比的时间，单位为 26.2 ms；

MOTOR_RAMPUP_DUTY：当 VSP 大于此值，电机开始旋转；

AS_GAIN：角度位移值 = duty/AS_GAIN + AS_OFFSET（仅限于正弦波 PWM）；

AS_OFFSET：角度位移偏置值；

线路 14 ~ 40： 定义变量

线路 43 ~ 46： 初始化 MCU；

线路 48 ~ 50： 初始化变量；

线路 52 ~ 61： 使能中断并初始化 AMC。如果 AMC 初始化或传输失败，AMC 再次初始化并切换 LED；否则关闭 LED；

线路 64： 发送空闲命令至 AMC。

线路 68 ~ 79： 判断 **_bTarget_Duty** 是否大于 **MOTOR_RAMPUP_DUTY** 和 **btShortciircuit_Protect** 清零，若结果为“是”，MSFR 的 RUN 位和 CW/CCW 位依据 DIR 状态置位，发送运行命令至 AMC；

线路 81： 判断占空比和 AS 是否需要更新；

线路 83 ~ 94： 增大或减小 **bCurrent_Duty**，如果 **bCurrent_Duty** 大于 **MAX_SPEED** 或小于 **MIN_SPEED**，**bCurrent_Duty** 不改变；

线路 95 ~ 97： 更新占空比和 AS 至 AMC，然后改变 PWM 输出；

线路 98 ~ 99： 清零 **bSpeed_Count**；

线路 102 ~ 109： 如果 **_bTarget_Duty** 小于 **MOTOR_RAMPUP_DUTY** 或 **btShortcircuit_Protect** 已激活，PWM 输出关断，然后清零占空比和 AS。

线路 110 ~ 121： 如果 **btShortcircuit_Protect** 已激活，打开 LED。如果未激活，清零变量。

程序代码如下。其中，小字符是由 MCDS 代码生成器生成的程序代码。

```
1     //--------------------------------------------------------------------
2     //  Initial Const
3     //--------------------------------------------------------------------
4     #define INITIAL_SLEEP     0x04
5
6     //User program start here.(09)
7     #define MAX_SPEED          500
8     #define MIN_SPEED          0
9     #define SPEED_LOOP_RATE    2     //1 step = 26.21 ms
10    #define MOTOR_RAMPUP_DUTY  10
11    #define AS_GAIN            128   //AS = DUTY/AS_GAIN + AS_OFFSET
12    #define AS_OFFSET          0
13
14    U8 bSpeed_Count=0;
15    UU16 _wCurrent_Duty;
16
17    #ifndef SDCC
18    extern SEG_BIT _btShortCircuit_Protect;
19    extern U8 _bShortCircuit_Cnt, _bADCCounterForSC;
20    extern U8 _bADCCounterForSC_Temp;
21    extern UU16 _wTarget_Duty;
22    #endif
23
24    //User program end here.(09)
25
26    //--------------------------------------------------------------------
27    //  Main Routine
28    //--------------------------------------------------------------------
29    #if defined SDCC
30        SEG_BIT __at (0x30) btInit_AMCStatus;
31    #else
32        SEG_BIT btInit_AMCStatus;
33    #endif
34    void main(void)
35    {
36        //User variable start here.(0A)
37    #if defined SDCC
38        SEG_BIT __at (0x31) btTrans_Status;
39    #else
40        SEG_BIT btTrans_Status;
41    #endif
42        //User variable end here.(0A)
43        CKCON = 0;
44        WRITE_MSFR(MSFR_SLEEP, INITIAL_SLEEP);
45
46        Initial_Procedure();
```

```
47        //User program start here.(02)
48        _bHall_Type= HALL_PINCFG0;    //HA=P14, HB= P15, HC=P16
49        _wTarget_Duty.U16 =0;
50        _wCurrent_Duty.U16 =0;
51        //User program end here.(02)
52        EA = 1;
53        btInit_AMCStatus = btInitial_AMCToHallInterface();
54
55        //User program start here.(03)
56        if(btInit_AMCStatus)
57        {
58            btInit_AMCStatus = btInitial_AMCToHallInterface();
59            nLED ^= 1;
60        }
61        nLED = 1;
62
63        //User program end here.(03)
64        bWriteCmdToAMC(CMD_AMC_CONTROL, 0, 0);
65        for(;;)
66        {
67          //User program start here.(04)
68          if((_wTarget_Duty.U16>MOTOR_RAMPUP_DUTY) && (!_btShortCircuit_Protect))
69          {
70             if(DIR)
71             {
72                 WRITE_MSFR(MSFR_MCNTL, MOTOR_RUN | MOTOR_CW);
73             }
74             else
75             {
76                 WRITE_MSFR(MSFR_MCNTL,  MOTOR_RUN);
77             }
78
79             bWriteCmdToAMC(CMD_AMC_CONTROL, 0, 1);
80
81             if (bSpeed_Count >= SPEED_LOOP_RATE)
82             {
83                 if(_wCurrent_Duty.U16 < _wTarget_Duty.U16)
84                 {
85                    // If current duty less than target, current duty + 1
86                    if(_wCurrent_Duty.U16 < MAX_SPEED)
87                       _wCurrent_Duty.U16++;
88                 }
89                 else  // If current duty more than target, current duty - 1
90                    if(_wCurrent_Duty.U16 > _wTarget_Duty.U16)
91                    {
92                       if(_wCurrent_Duty.U16 > MIN_SPEED)
93                          _wCurrent_Duty.U16--;
94                     }
95                 btTrans_Status = bWriteCmdToAMC(CMD_ANGLE_SHIFT, 0,
                                        _wCurrent_Duty.U16/AS_GAIN);
96                 if(!btTrans_Status)
97                    btTrans_Status = bWriteCmdToAMC(CMD_DUTY,
                                         _wCurrent_Duty.U8[0], _wCurrent_Duty.U8[1]);
98                 if(!btTrans_Status)
99                    bSpeed_Count = 0;
100            }
101         }
102         else
103         {
104            _wCurrent_Duty.U16 = 0;
105            WRITE_MSFR(MSFR_MCNTL,  MOTOR_FREE);
106
107            bWriteCmdToAMC(CMD_AMC_CONTROL, 0, 0);
```

```
108             btTrans_Status = bWriteCmdToAMC(CMD_ANGLE_SHIFT, 0, 0);
109             btTrans_Status = bWriteCmdToAMC(CMD_DUTY, 0, 0);
110             if(_btShortCircuit_Protect)
111             {
112                nLED = 0;  //Fault LED on, release the flag after reset device
113             }
114             else    //clear protect flag
115             {
116                _bADCCounterForSC = 0;
117                _bShortCircuit_Cnt = 0;
118                _bADCCounterForSC_Temp = 0;
119                _btShortCircuit_Protect = 0;
120                nLED = 1;  //Fault LED off
121             }
122          }
123          //User program end here.(04)
124       }
```

MCS51.c

MCS51.c 与初始 I/O 端口设置相关，用于 MCU 和中断请求处理。程序说明如下：

线路 6 ~ 23： 中断 TIMER0 超时的服务例程；

线路 19： 增加 **bSpeed_Count**。

程序代码如下。其中，小字符是由 MCDS 代码生成器生成的程序代码。

```
1   //-----------------------------------------------------------------------
2   //  Interrupt Service Routine
3   //-----------------------------------------------------------------------
4   // Timer0 ISR
5   INTERRUPT(ISR_T0, VECTOR_ET0)
6   {
7       U8 bBackupADR;
8       UU16 V;
9       //User variable start here.(39)
10
11      //User variable end here.(39)
12
13      bBackupADR = MSFRADR;
14      //Initial Timer0
15      V.U16 = INITIAL_T0_INTERVAL;
16      TH0 = V.U8[MSB];
17      TL0 = V.U8[LSB];
18      //User program start here.(13)
19      bSpeed_Count++;
20
21      //User program end here.(13)
22      MSFRADR = bBackupADR;
23  }
```

MotorCtrl.c

MotorCtrl.c 与初始寄存器设置相关，用于 MSFR 和由电机控制产生的中断请求处理。程序说明如下：

线路 5 ~ 17： 中断霍尔信号触发器的服务例程；

线路 13： 切换 FO 引脚；

线路 20 ~ 48：中断 ADC 触发器的服务例程；

线路 30 ~ 32：将 ADC0 值 (9 位) 存储至 **_wTarget_Dut**；

线路 34 ~ 35： 将 theta 输出至 AOUT 引脚；

线路 40 ~ 52： OCP 分析例程；

线路 58 ~ 87： 故障中断服务例程；

程序代码如下。其中，小字符是由 MCDS 代码生成器生成的程序代码。

```
1    //---------------------------------------------------------------------
2    //  Interrupt Service Routine
3    //---------------------------------------------------------------------
4    // Hall ISR
5    INTERRUPT(ISR_Hall, VECTOR_EX10)
6    { // Ex10
7        U8 bBackupADR;
8        //User variable start here.(22)
9
10       //User variable end here.(22)
11       bBackupADR = MSFRADR;
12       //User program start here.(16)
13       FO ^= 1;
14
15       //User program end here.(16)
16       MSFRADR = bBackupADR;
17   }
18
19   // ADC ISR
20   INTERRUPT(ISR_ADC, VECTOR_EX9)
21   {  //EX9
22      U8 bBackupADR;
23      U8 bV;
24      //User variable start here.(23)
25      U8 bAngle;
26
27      //User variable end here.(23)
28      bBackupADR = MSFRADR;
29      //User program start here.(1A)
30      READ_MSFR(MSFR_ADC0H, _wTarget_Duty.U8[0]);
31      READ_MSFR(MSFR_ADC0L, _wTarget_Duty.U8[1]);
32      _wTarget_Duty.U16 >>= 7;
33      _bADCCounterForSC++;
34      READ_MSFR(MSFR_ECL, bAngle);
35      WRITE_MSFR(MSFR_DAC3, bAngle);
36
37      //User program end here.(1A)
38
39      READ_MSFR(MSFR_OCSTA, bV);
40      if (bV)
41      {
42         if (bV & 0x08)  // OCAH
43            Evt_OCHA();
44         if (bV & 0x10)  // OCBH
45            Evt_OCHB();
46         if (bV & 0x20)  // OCCH
47            Evt_OCHC();
48      }
49
50      //User program start here.(17)
51      else
```

```
52          nLED = 1;
53
54      //User program end here.(17)
55      MSFRADR = bBackupADR;
56  }
57
58  // Fault ISR
59  INTERRUPT(ISR_Fault, VECTOR_EX8)
60  {  //Ex8
61      U8 bBackupADR;
62      U8 bV;
63      //User variable start here.(24)
64
65      //User variable end here.(24)
66
67      bBackupADR = MSFRADR;
68      READ_MSFR(MSFR_MSTAT, bV);
69      if (bV & 0x7F)
70      {
71          //User program start here.(1B)
72
73          //User program end here.(1B)
74          if (bV & 0x20)  // Short A
75              Evt_ShortA();
76          if (bV & 0x10)  // Short B
77              Evt_ShortB();
78          if (bV & 0x08)  // Short C
79              Evt_ShortC();
80          if (bV & 0x04)  // Hall Error
81              Evt_HERR();
82          //User program start here.(18)
83
84          //User program end here.(18)
85      }
86      MSFRADR = bBackupADR;
87  }
```

相关资源

FCM8531 – 嵌入式 MCU 和可配置三相 PMSM / BLDC 电机控制器

AN-8202 – FCM8531 用户手册- 硬件说明

AN-8207 –FCM8531 的 MCDS IDE 用户指南