找出Iq

目前的流程 全都在FOC\_CurrControllerM1()中唯一會去真正計算 Iq以及Id的地方

ADC 取樣 => R3\_2\_GetPhaseCurrents()

Clarke 轉換 => MCM\_Clarke()

Park 轉換 => MCM\_Park()

PI 控制 =>PI\_Controller()

反轉換 → αβ =>MCM\_Rev\_Park()

SVPWM => PWMC\_SetPhaseVoltage()

輸出

FOCVars[M1].Iqd = Iqd;

**FOC (解析FOC)**

先宣告變數後

1. **決定目前模式 mode = MCI\_GetControlMode( &Mci[M1] );**

* Mci在 /mc\_type.h/MC\_ControlMode\_t

1. **取得馬達所使用的速度/位置**

* speedHandle =STC\_GetSpeedSensor(pSTC[M1]);

1. **取得目前的 Electrical angle hElAngle = SPD\_GetElAngle(speedHandle);**

### hElAngle = SPD\_GetElAngle(speedHandle);

* 在speed\_pos\_fdbk.h/ SpeednPosFdbk\_Handle\_t /hElAngle
* 而hElAngle是 mc\_tasks\_foc.c 內的 (void)STO\_PLL\_CalcElAngle(&STO\_PLL\_M1, &STO\_Inputs);
* pHandle->\_Super.hElAngle += hRotor\_Speed;
* hRotor\_Speed = STO\_ExecutePLL(pHandle, hAux\_Alfa, -hAux\_Beta);
* hRotor\_Speed = (Kp \* error) / Kp\_div + (IntegralTerm / Ki\_div) 🡺(PI\_Controller()
* hAux\_Alfa = (int16\_t)(hAux\_Alfa \* wDirection); wDirection = 1/-1
* hAux\_Beta = (int16\_t)(hAux\_Beta \* wDirection);
* #ifndef FULL\_MISRA\_C\_COMPLIANCY\_STO\_PLL
* hAux\_Alfa = (int16\_t)(pHandle->wBemf\_alfa\_est >> pHandle->F2LOG);
* hAux\_Beta = (int16\_t)(pHandle->wBemf\_beta\_est >> pHandle->F2LOG);
* #else
* hAux\_Alfa = (int16\_t)(pHandle->wBemf\_alfa\_est / pHandle->hF2);
* hAux\_Beta = (int16\_t)(pHandle->wBemf\_beta\_est / pHandle->hF2);
* #endif 但是後續就找不到wBemf\_alfa\_est 是如何出現的不過公式可能是
* wBemf\_alfa\_est = Vα - R \* Iα - L \* dIα/dt;
* wBemf\_beta\_est = Vβ - R \* Iβ - L \* dIβ/dt;
* ---摺疊結束--------------------------------------------------

1. **角度補償 （因為 FOC 有計算與輸出延遲）**

* hElAngle += SPD\_GetInstElSpeedDpp(speedHandle) \* PARK\_ANGLE\_COMPENSATION\_FACTOR;

(目前為0) SpeednPosFdbk\_Handle\_t 在speed\_pos\_fdbk.h

1. **取得當前電流向量 PWMC\_GetPhaseCurrents(pHandle, &Iab);**

由於程式內部把GetPhaseCurrents指向R3\_2\_GetPhaseCurrents;

所以代表在執行一次R3\_2\_GetPhaseCurrents;

* 在r3\_2\_g4xx\_pwm\_curr\_fdbk.內的R3\_2\_Init()
* pHandle->\_Super.pFctGetPhaseCurrents = &R3\_2\_GetPhaseCurrents;
* R3\_2\_GetPhaseCurrents() => 用ADC pin(類比轉數位)，讀取馬達三相電壓 Ia Ib Ic
* Aux = (int32\_t)(pHandle->PhaseAOffset) - (int32\_t)(ADCDataReg1);
* PhaseAOffset =2048 ， ADCDataReg1=0~4095 🡺 Aux=-2048 ~ +2048
* Stypedef struct {
* int16\_t a; Iab->a = (int16\_t)Aux; // 寫入 Ia 是12bit值
* int16\_t b; Iab->b = (int16\_t)-Aux; // 寫入 Ib -2048 ~+ 2048
* } ab\_t;
* 在r3\_2\_g4xx\_pwm\_curr\_fdbk.中使用的函式
* void R3\_2\_GetPhaseCurrents(PWMC\_Handle\_t \*pHdl, ab\_t \*Iab) 可折疊看
* \_\_weak void R3\_2\_GetPhaseCurrents(PWMC\_Handle\_t \*pHdl, ab\_t \*Iab)
* {
* #ifdef NULL\_PTR\_CHECK\_R3\_2\_PWM\_CURR\_FDB
* if (MC\_NULL == Iab)
* {
* /\* Nothing to do \*/
* }
* else
* {
* #endif
* int32\_t Aux;
* uint32\_t ADCDataReg1;
* uint32\_t ADCDataReg2;
* PWMC\_R3\_2\_Handle\_t \*pHandle = (PWMC\_R3\_2\_Handle\_t \*)pHdl; //cstat !MISRAC2012-Rule-11.3
* TIM\_TypeDef \*TIMx = pHandle->pParams\_str->TIMx;
* uint8\_t Sector;
* Sector = (uint8\_t)pHandle->\_Super.Sector;
* ADCDataReg1 = pHandle->pParams\_str->ADCDataReg1[Sector]->JDR1;
* ADCDataReg2 = pHandle->pParams\_str->ADCDataReg2[Sector]->JDR1;
* /\* Disable ADC trigger source \*/
* /\* LL\_TIM\_CC\_DisableChannel(TIMx, LL\_TIM\_CHANNEL\_CH4) \*/
* LL\_TIM\_SetTriggerOutput(TIMx, LL\_TIM\_TRGO\_RESET);
* switch (Sector)
* {
* case SECTOR\_4:
* case SECTOR\_5:
* {
* /\* Current on Phase C is not accessible \*/
* /\* Ia = PhaseAOffset - ADC converted value) \*/
* Aux = (int32\_t)(pHandle->PhaseAOffset) - (int32\_t)(ADCDataReg1);
* /\* Saturation of Ia \*/
* if (Aux < -INT16\_MAX)
* {
* Iab->a = -INT16\_MAX;
* }
* else if (Aux > INT16\_MAX)
* {
* Iab->a = INT16\_MAX;
* }
* else
* {
* Iab->a = (int16\_t)Aux;
* }
* /\* Ib = PhaseBOffset - ADC converted value) \*/
* Aux = (int32\_t)(pHandle->PhaseBOffset) - (int32\_t)(ADCDataReg2);
* /\* Saturation of Ib \*/
* if (Aux < -INT16\_MAX)
* {
* Iab->b = -INT16\_MAX;
* }
* else if (Aux > INT16\_MAX)
* {
* Iab->b = INT16\_MAX;
* }
* else
* {
* Iab->b = (int16\_t)Aux;
* }
* break;
* }
* case SECTOR\_6:
* case SECTOR\_1:
* {
* /\* Current on Phase A is not accessible \*/
* /\* Ib = PhaseBOffset - ADC converted value) \*/
* Aux = (int32\_t)(pHandle->PhaseBOffset) - (int32\_t)(ADCDataReg1);
* /\* Saturation of Ib \*/
* if (Aux < -INT16\_MAX)
* {
* Iab->b = -INT16\_MAX;
* }
* else if (Aux > INT16\_MAX)
* {
* Iab->b = INT16\_MAX;
* }
* else
* {
* Iab->b = (int16\_t)Aux;
* }
* /\* Ia = -Ic -Ib \*/
* Aux = (int32\_t)(ADCDataReg2) - (int32\_t)(pHandle->PhaseCOffset); /\* -Ic \*/
* Aux -= (int32\_t)Iab->b; /\* Ia \*/
* /\* Saturation of Ia \*/
* if (Aux > INT16\_MAX)
* {
* Iab->a = INT16\_MAX;
* }
* else if (Aux < -INT16\_MAX)
* {
* Iab->a = -INT16\_MAX;
* }
* else
* {
* Iab->a = (int16\_t)Aux;
* }
* break;
* }
* case SECTOR\_2:
* case SECTOR\_3:
* {
* /\* Current on Phase B is not accessible \*/
* /\* Ia = PhaseAOffset - ADC converted value) \*/
* Aux = (int32\_t)(pHandle->PhaseAOffset) - (int32\_t)(ADCDataReg1);
* /\* Saturation of Ia \*/
* if (Aux < -INT16\_MAX)
* {
* Iab->a = -INT16\_MAX;
* }
* else if (Aux > INT16\_MAX)
* {
* Iab->a = INT16\_MAX;
* }
* else
* {
* Iab->a = (int16\_t)Aux;
* }
* /\* Ib = -Ic -Ia \*/
* Aux = (int32\_t)(ADCDataReg2) - (int32\_t)(pHandle->PhaseCOffset); /\* -Ic \*/
* Aux -= (int32\_t)Iab->a; /\* Ib \*/
* /\* Saturation of Ib \*/
* if (Aux > INT16\_MAX)
* {
* Iab->b = INT16\_MAX;
* }
* else if (Aux < -INT16\_MAX)
* {
* Iab->b = -INT16\_MAX;
* }
* else
* {
* Iab->b = (int16\_t)Aux;
* }
* break;
* }
* default:
* break;
* }
* pHandle->\_Super.Ia = Iab->a;
* pHandle->\_Super.Ib = Iab->b;
* pHandle->\_Super.Ic = -Iab->a - Iab->b;
* #ifdef NULL\_PTR\_CHECK\_R3\_2\_PWM\_CURR\_FDB
* }
* #endif
* }
* ---摺疊結束------------------------------------------------------------------

1. **執行pin腳功能 重新使用ADC 通道**

RCM\_ReadOngoingConv();

RCM\_ExecNextConv();

1. **執行Clarke 轉換 Ialphabeta = MCM\_Clarke(Iab);**

* 將Ia,Ib => Iα, Iβ typedef struct 一樣是12bit值

{

int16\_t alpha; Iα

int16\_t beta; Iβ

} alphabeta\_t;

Ialphabeta 也是一樣的結構

在mc\_math.c 中使用的函式

MCM\_Clarke()

\_\_weak alphabeta\_t **MCM\_Clarke**(ab\_t Input)

{

alphabeta\_t Output;

int32\_t a\_divSQRT3\_tmp;

int32\_t b\_divSQRT3\_tmp;

int32\_t wbeta\_tmp;

int16\_t hbeta\_tmp;

/\* qIalpha = qIas\*/

Output.alpha = Input.a;

a\_divSQRT3\_tmp = divSQRT\_3 \* ((int32\_t)Input.a);

b\_divSQRT3\_tmp = divSQRT\_3 \* ((int32\_t)Input.b);

/\* qIbeta = -(2\*qIbs+qIas)/sqrt(3) \*/

**#ifndef** FULL\_MISRA\_C\_COMPLIANCY\_MC\_MATH

/\* WARNING: the below instruction is not MISRA compliant, user should verify

that Cortex-M3 assembly instruction ASR (arithmetic shift right) is used by

the compiler to perform the shift (instead of LSR logical shift right) \*/

//cstat !MISRAC2012-Rule-1.3\_n !ATH-shift-neg !MISRAC2012-Rule-10.1\_R6

wbeta\_tmp = (-(a\_divSQRT3\_tmp) - (b\_divSQRT3\_tmp) - (b\_divSQRT3\_tmp)) >> 15;

**#else**

wbeta\_tmp = (-(a\_divSQRT3\_tmp) - (b\_divSQRT3\_tmp) - (b\_divSQRT3\_tmp)) / 32768;

**#endif**

/\* Check saturation of Ibeta \*/

**if** (wbeta\_tmp > INT16\_MAX)

{

hbeta\_tmp = INT16\_MAX;

}

**else** **if** (wbeta\_tmp < (-32768))

{

hbeta\_tmp = ((int16\_t)-32768);

}

**else**

{

hbeta\_tmp = ((int16\_t)wbeta\_tmp);

}

Output.beta = hbeta\_tmp;

**if** (((int16\_t )-32768) == Output.beta)

{

Output.beta = -32767;

}

**else**

{

/\* Nothing to do \*/

}

**return** (Output);

}

---摺疊結束--------------------------------------------------

1. **執行Park 轉換 Iqd = MCM\_Park(Ialphabeta, hElAngle);**

* (Clarke → dq），將Iα, Iβ => Id , Iq

Id = Iα \* cos(θ) + Iβ \* sin(θ) typedef struct

Iq = -Iα \* sin(θ) + Iβ \* cos(θ) {

int16\_t q; // Iq：轉矩分量

int16\_t d; // Id：磁通分量

} qd\_t;

在mc\_math.c 中使用的函式

### MCM\_Park()

\_\_weak qd\_t MCM\_Park(alphabeta\_t Input, int16\_t Theta)

{

qd\_t Output;

int32\_t d\_tmp\_1;

int32\_t d\_tmp\_2;

int32\_t q\_tmp\_1;

int32\_t q\_tmp\_2;

int32\_t wqd\_tmp;

int16\_t hqd\_tmp;

Trig\_Components Local\_Vector\_Components;

Local\_Vector\_Components = MCM\_Trig\_Functions(Theta);

/\* No overflow guaranteed \*/

q\_tmp\_1 = Input.alpha \* ((int32\_t )Local\_Vector\_Components.hCos);

/\* No overflow guaranteed \*/

q\_tmp\_2 = Input.beta \* ((int32\_t)Local\_Vector\_Components.hSin);

/\* Iq component in Q1.15 Format \*/

#ifndef FULL\_MISRA\_C\_COMPLIANCY\_MC\_MATH

/\* WARNING: the below instruction is not MISRA compliant, user should verify

that Cortex-M3 assembly instruction ASR (arithmetic shift right) is used by

the compiler to perform the shift (instead of LSR logical shift right) \*/

wqd\_tmp = (q\_tmp\_1 - q\_tmp\_2) >> 15; //cstat !MISRAC2012-Rule-1.3\_n !ATH-shift-neg !MISRAC2012-Rule-10.1\_R6

#else

wqd\_tmp = (q\_tmp\_1 - q\_tmp\_2) / 32768;

#endif

/\* Check saturation of Iq \*/

if (wqd\_tmp > INT16\_MAX)

{

hqd\_tmp = INT16\_MAX;

}

else if (wqd\_tmp < (-32768))

{

hqd\_tmp = ((int16\_t)-32768);

}

else

{

hqd\_tmp = ((int16\_t)wqd\_tmp);

}

Output.q = hqd\_tmp;

if (((int16\_t)-32768) == Output.q)

{

Output.q = -32767;

}

else

{

/\* Nothing to do \*/

}

/\* No overflow guaranteed \*/

d\_tmp\_1 = Input.alpha \* ((int32\_t )Local\_Vector\_Components.hSin);

/\* No overflow guaranteed \*/

d\_tmp\_2 = Input.beta \* ((int32\_t )Local\_Vector\_Components.hCos);

/\* Id component in Q1.15 Format \*/

#ifndef FULL\_MISRA\_C\_COMPLIANCY\_MC\_MATH

/\* WARNING: the below instruction is not MISRA compliant, user should verify

that Cortex-M3 assembly instruction ASR (arithmetic shift right) is used by

the compiler to perform the shift (instead of LSR logical shift right) \*/

wqd\_tmp = (d\_tmp\_1 + d\_tmp\_2) >> 15; //cstat !MISRAC2012-Rule-1.3\_n !ATH-shift-neg !MISRAC2012-Rule-10.1\_R6

#else

wqd\_tmp = (d\_tmp\_1 + d\_tmp\_2) / 32768;

#endif

/\* Check saturation of Id \*/

if (wqd\_tmp > INT16\_MAX)

{

hqd\_tmp = INT16\_MAX;

}

else if (wqd\_tmp < (-32768))

{

hqd\_tmp = ((int16\_t)-32768);

}

else

{

hqd\_tmp = ((int16\_t)wqd\_tmp);

}

Output.d = hqd\_tmp;

if (((int16\_t)-32768) == Output.d)

{

Output.d = -32767;

}

else

{

/\* Nothing to do \*/

}

return (Output);

}

---摺疊結束--------------------------------------------------

1. **執行PI\_Controller() 計算目前電流誤差 輸出q 軸轉矩、d 軸磁通**

* Vqd.q = PI\_Controller(pPIDIq[M1], (int32\_t)(FOCVars[M1].Iqdref.q) - Iqd.q);
* Vqd.d = PI\_Controller(pPIDId[M1], (int32\_t)(FOCVars[M1].Iqdref.d) - Iqd.d);
* Vqd.q =>控制轉矩 Vqd.d控制「磁通分量」（常為 0）
* (Kp \* error) / Kp\_div + (IntegralTerm / Ki\_div) 🡺(PI\_Controller()

1. **判斷是否為OPEN-LOOP**

* **if** (mode == MCM\_OPEN\_LOOP\_VOLTAGE\_MODE)
* { Vqd = OL\_VqdConditioning(pOpenLoop[M1]); }

1. **限制Vqd 向量長度 控制在一個** 最大圓形半徑內

* Vqd = Circle\_Limitation(&CircleLimitationM1, Vqd);

1. **更新目前的** 電角角度 hElAngle

* hElAngle += SPD\_GetInstElSpeedDpp(speedHandle) \* REV\_PARK\_ANGLE\_COMPENSATION\_FACTOR;1

1. **執行反 Park 轉換 Valphabeta =最終電壓命令向量**

* Valphabeta = MCM\_Rev\_Park(Vqd, hElAngle);

1. **把電壓命令轉成三相輸出 並回傳是否有錯誤**

* hCodeError = PWMC\_SetPhaseVoltage(pwmcHandle[M1], Valphabeta);
* 將 SVM 計算出的三相 PWM compare 值（CCR）先儲存在 pHandle 結構中等待Timer 下一次 Update 事件觸發 把 CntPhX 寫入 TIMx->CCRn
* pHandle->CntPhA = (uint16\_t)(MAX(wTimePhA, 0));
* pHandle->CntPhB = (uint16\_t)(MAX(wTimePhB, 0));
* pHandle->CntPhC = (uint16\_t)(MAX(wTimePhC, 0));

1. **將儲存在 CntPhA/B/C 的 PWM Compare 值真正寫入 Timer**

* 在r3\_2\_g4xx\_pwm\_curr\_fdbk.中使用的函式 有把值寫入Timer
* **R3\_2\_WriteTIMRegisters**(PWMC\_Handle\_t \*pHdl, uint16\_t SamplingPoint)
* LL\_TIM\_OC\_SetCompareCH1(TIMx, (uint32\_t) pHandle->\_Super.CntPhA);
* LL\_TIM\_OC\_SetCompareCH2(TIMx, (uint32\_t) pHandle->\_Super.CntPhB);
* LL\_TIM\_OC\_SetCompareCH3(TIMx, (uint32\_t) pHandle->\_Super.CntPhC);
* LL\_TIM\_OC\_SetCompareCH4(TIMx, (uint32\_t) SamplingPoint);

假設 PWM Period = 2000 → 20kHz PWM

Half\_PWMPeriod = 1000

CCR1 = 1000 + Valpha \* K;

Valpha = 0 → PWM = 50% Duty  
Valpha > 0 → PWM > 50%  
Valpha < 0 → PWM < 50%

Duty = CCR1 / 2000 \* 100% => 1000/2000\*100% =50%

**由於FOC全部運算都是用電壓比例(-2048~+2048)運算所以我找了轉換成電流的函式，**在mc\_interface.c/MCI\_GetIqd\_F 中有

iqd.d = (float\_t)((float\_t)pHandle->pFOCVars->Iqd.d \* pHandle->pScale->current);

iqd.q = (float\_t)((float\_t)pHandle->pFOCVars->Iqd.q \* pHandle->pScale->current);

這是把 iqd.d = Iqd.d \* ( Vref / ADC\_Resolution ) / ( Rshunt \* Amplification\_gain )

Iqd.d \* (3.3/4096) / (0.003\*9.14) 🡺 Iqd.d \*0.0293

2048\*0.0293=60.0064

#define MAX\_CURRENT (ADC\_REFERENCE\_VOLTAGE / (2 \* RSHUNT \* AMPLIFICATION\_GAIN)) =3.3/(2\*0.003\*9.14) = 60.175

在mc\_tasks\_foc.c 中使用的函式

### FOC\_CurrControllerM1()

inline uint16\_t FOC\_CurrControllerM1(void)

{

qd\_t Iqd, Vqd;

ab\_t Iab;

alphabeta\_t Ialphabeta, Valphabeta;

int16\_t hElAngle;

uint16\_t hCodeError;

SpeednPosFdbk\_Handle\_t \*speedHandle;

MC\_ControlMode\_t mode;

mode = MCI\_GetControlMode( &Mci[M1] );

speedHandle = STC\_GetSpeedSensor(pSTC[M1]);

hElAngle = SPD\_GetElAngle(speedHandle);

hElAngle += SPD\_GetInstElSpeedDpp(speedHandle)\*PARK\_ANGLE\_COMPENSATION\_FACTOR;

PWMC\_GetPhaseCurrents(pwmcHandle[M1], &Iab);

RCM\_ReadOngoingConv();

RCM\_ExecNextConv();

Ialphabeta = MCM\_Clarke(Iab);

Iqd = MCM\_Park(Ialphabeta, hElAngle);

Vqd.q = PI\_Controller(pPIDIq[M1], (int32\_t)(FOCVars[M1].Iqdref.q) - Iqd.q);

Vqd.d = PI\_Controller(pPIDId[M1], (int32\_t)(FOCVars[M1].Iqdref.d) - Iqd.d);

if (mode == MCM\_OPEN\_LOOP\_VOLTAGE\_MODE)

{

Vqd = OL\_VqdConditioning(pOpenLoop[M1]);

}

else

{

/\* Nothing to do \*/

}

Vqd = Circle\_Limitation(&CircleLimitationM1, Vqd);

hElAngle += SPD\_GetInstElSpeedDpp(speedHandle)\*REV\_PARK\_ANGLE\_COMPENSATION\_FACTOR;

Valphabeta = MCM\_Rev\_Park(Vqd, hElAngle);

hCodeError = PWMC\_SetPhaseVoltage(pwmcHandle[M1], Valphabeta);

FOCVars[M1].Vqd = Vqd;

FOCVars[M1].Iab = Iab;

FOCVars[M1].Ialphabeta = Ialphabeta;

FOCVars[M1].Iqd = Iqd;

FOCVars[M1].Valphabeta = Valphabeta;

FOCVars[M1].hElAngle = hElAngle;

return (hCodeError);

}

---摺疊結束--------------------------------------------------