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`

2. 取得馬達所使用的速度/位置

➢ `speedHandle =STC_GetSpeedSensor(pSTC[M1]);`

3. 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---

4. 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

5. 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)

6. 執行 pin 腳功能 重新使用 ADC 通道

RCM_ReadOngoingConv();
RCM_ExecNextConv();

7. 執行 Clarke 轉換 Ialphabeta = MCM_Clarke(Iab);

- 將 Ia,Ib => Iα, Iβ

```
Iα = Ia
Iβ = (Ia + 2*Ib) / √3
```

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()

8. 執行 Park 轉換 Iqd = MCM_Park(Ialphabeta, hElAngle);

- (Clarke → dq），將 Iα, Iβ => Id , Iq

Id = Iα * cos(θ) + Iβ * sin(θ)

Iq = -Iα * sin(θ) + Iβ * cos(θ)

```
typedef struct
{
  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()

9. 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()

10. 判斷是否為 OPEN-LOOP

- `if (mode == MCM_OPEN_LOOP_VOLTAGE_MODE)`
- `{ Vqd = OL_VqdConditioning(pOpenLoop[M1]); }`

11. 限制 Vqd 向量長度 控制在一個 最大圓形半徑內

- `Vqd = Circle_Limitation(&CircleLimitationM1, Vqd);`

12. 更新目前的 電角角度 hElAngle

- hElAngle += SPD_GetInstElSpeedDpp(speedHandle) * REV_PARK_ANGLE_COMPENSATION_FACTOR;1

13. 執行反 Park 轉換 Valphabeta =最終電壓命令向量

- Valphabeta = MCM_Rev_Park(Vqd, hElAngle);

14. 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));`

15. 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);
```

```c
    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---