마이크로프로세서 응용 보고서 (PI 제어기 튜닝하기)

자동차IT융합학과 20183376

박선재

```
/*전류제어만 있는 코드*/
#include "MPC5604P_M26V.h"
#include "freemaster.h"
#include "init_base.h"

/********************* Macro **********************/
#define LPF(out, in, fct)  out = out-in>0 ? fct*(out-in)+ in :-fct*(in-out)+ in
#define FILFCT(wc, fct, ts)  fct =  ts/(wc+ts)
#define bound(in,lim)  ((in > (lim)) ? (lim) : ((in < -(lim)) ? -(lim) : in));
#define abs(x) ((x > 0) ? x : -x)
#define rad2rpm 9.54929
#define TWO_PI6.28316

/********************* Variables **********************/
volatile int i = 0, temp10ms=0;
volatile int j = 0;
int Reset = 1, DSPRUN = 0, SpdCon = 0;
float IdcPre=0.0f, IdcErr=0.0f, IdcRef=0.0f, IdcFdb=0.0f, IdcOffset=0.0f;
float VdcRef=0.0f, VdcPterm=0.0f, VdcPiterm=0.0f, VdcIterm=0.0f;
float VdcFil=0.0f, VdcPre=0.0f, vRefUlim=0.0f, PWM_Scale=0.0f;
float Kpc=0.0f, Kic=0.0f;
int PWM_B=0, PWM_Peak=0;
long int delta_m=0, mold=0, mnew = 0;
float rpm = 0., rpmFil = 0., wrTemp = 0., wr = 0., wrFil = 0., rpmRef = 0., rpmErr
=0.;
float spdPterm = 0., spdPiterm = 0., spdIterm = 0., spdPiUlim = 0., spdPiOut = 0.;
float kpSpd = 0., kiSpd = 0.;
float iScale=0.0f, VdcScale=0.0f, FadcScale=0.0f, FvdcHwScale=0.0f, FiHwScale=0.0f,
EncoderScale=0.0f, SpeedScale=0.0f, MotorEncPulse=4.0f;
unsigned int offsetTimer=0, OFF_SW=1, ErrCode=0;
char Err1 = 0, Err2 = 0;
float OverCurrent = 0.0f, OverSpeed = 0.0f;
float VdcFdbLpfFct=0.0f, IdcOffsetLpfFct=0.0f, rpmFilLpfFct=0.0f;
uint8_t ErrReset = 0;
float IadcDATA = 0;
float Motor_R = 0.0f;
float Motor_L = 0.0f;
```

```
unsigned int Wcc = 0;
float absIdcFdb=0.0;
float VdcCmd = 0.0f;
int encodercnt = 0;

/********************* Funtion **********************/
void init_Variable(void);
void init_PIN(void);
void init_ADC(void);
void init_FlexPWM0_sub1(void);
void MainISR(void);
void ADC_Read(void);
void Errdetect(void);
void H_BridgeRun(void);
void adc_offset_cal(void);
void PWMISR(void);
void init_PIT(void);
void init_eTimer0(void);
void ISR_PIT_100ms(void);
void EncoderFdb(void);

int main(void)
{
        initModesAndClock();
        disableWatchdog();
        enableIrq();
        initOutputClock();
        FMSTR_Init();
        init_INTC();
        init_Linflex0();
        init_PIN();
        init_ADC();
        init_FlexPWM0_sub1();
        init_Variable();
        init_PIT();
        init_eTimer0();

        INTC_InstallINTCInterruptHandler(PWMISR,183,6);
        INTC_InstallINTCInterruptHandler(ISR_PIT_100ms,59,5);

        /* Loop forever */
```

```
        for (;;)
        {
                FMSTR_Recorder();
                FMSTR_Poll();

                i++;
        }
}

void init_Variable(void)
{
        FadcScale = 5.0f/1023.0f;
        FvdcHwScale = 10.0f/40.0f;
        FiHwScale = 1.0f / 20.0f;
        iScale = (float)(FadcScale/FiHwScale);
        VdcScale = (float)(FadcScale/FvdcHwScale);
        SIU.GPDO[40].B.PDO = 1; // RESET에 1 (active low임)
        SIU.GPDO[61].B.PDO = 1; // SR
        SIU.GPDO[58].B.PDO = 1; //Gate High 1을 이 핀에 내려보냄
        SIU.GPDO[59].B.PDO = 1; //Gate Low
        PWM_Peak = 1600; //duty 중간값
        PWM_Scale = PWM_Peak/12;
        vRefUlim = 12.0f;
        Motor_R = 0.002;
        Motor_L = 0.02;
        Wcc=150;
        Kpc = Motor_L * Wcc;
        Kic = Motor_R * Wcc;
        kpSpd = 0.0f;
        kiSpd = 0.0f;
        OverCurrent = 10.0f;
        kpSpd = 0.00001f;
        kiSpd = 0.00001f;
        FILFCT(10. , IdcOffsetLpfFct , 100.);
        FILFCT(100. , IdcOffsetLpfFct , 10000.);
        EncoderScale = (float)(6.28316/4.0);
}
//-----------------------------------------
//      MPC5604P Device Configuration
//-----------------------------------------
void init_PIN(void)
```

```
{
        SIU.PCR[62].R = 0x0600; // FlexPWM_0 B[1]->PHASE
        SIU.PCR[58].R = 0x0300;// GateIC PWMH
        SIU.PCR[59].R = 0x0300;// GateIC PWML
        SIU.PCR[61].R = 0x0300; // GateIC SR
        SIU.PCR[40].R = 0x0300; // GateIC Reset
        SIU.PCR[29].R = 0x0100; // GateIC Err2
        SIU.PCR[30].R = 0x0100; // GateOC Err1
        SIU.PCR[23].R = 0x2000; // ADC0 AN[0] VBAT
        SIU.PCR[34].R = 0x2000; // ADC0 AN[3] IDC
        SIU.PCR[0].R = 0x0500; // ADC0 AN[3] IDC
        SIU.PCR[1].R = 0x0500; // ADC0 AN[3] IDC


}

void init_PIT(void)
{
        PIT.PITMCR.R = 0;
        PIT.CH[0].LDVAL.R = 6400000;
        PIT.CH[0].TCTRL.B.TIE = 0x1;
        PIT.CH[0].TCTRL.B.TEN = 0x1;
}

void init_eTimer0(void)
{
        ETIMER_0.ENBL.R = 0x0000;
        ETIMER_0.CHANNEL[0].CNTR.R = 0x0000;
        ETIMER_0.CHANNEL[0].CTRL.B.CNTMODE = 0b100;
        ETIMER_0.CHANNEL[0].CTRL.B.PRISRC = 0x0000;
        ETIMER_0.CHANNEL[0].CTRL.B.SECSRC = 0x0001;
        ETIMER_0.ENBL.R = 0x0001;

}

int temp100ms = 0;
void ISR_PIT_100ms(void)
{
        temp100ms++;

        EncoderFdb();
```

```
        //SpeedCon();
        PIT.CH[0].TFLG.B.TIF = 1;

}

void EncoderFdb(void)
{
        encodercnt++;
        mold = mnew;
        mnew = ETIMER_0.CHANNEL[0].CNTR.R;
        delta_m = ((mold - mnew));
        wr = (float)(delta_m)*(EncoderScale/0.1);
        LPF(wrFil, wr, rpmFilLpfFct);
        rpmFil = (float)(wrFil*9.54929);
}

void init_ADC(void) // 0번 모듈 , 0번 채널 , 3번 채널
{
   ADC_0.MCR.B.ABORT = 1; //진행중인 변환 중단
   ADC_0.MCR.B.PWDN = 0; //POWER DOWN MODE 꺼
   ADC_0.CTR[0].R = 0x00008208;
   ADC_0.NCMR[0].R = 0x00000009; //0번채널, 3번채널 ENABLE
   ADC_0.CDR[1].R = 0x00000000; //데이터 받아오는 레지스터 초기화
   ADC_0.CDR[3].R = 0x00000000;  //데이터 받아오는 레지스터 초기화
}

void init_FlexPWM0_sub1(void)
{
        FLEXPWM_0.OUTEN.B.PWMB_EN = 0b0010;
        FLEXPWM_0.MCTRL.B.LDOK  = 0b0010;
        FLEXPWM_0.MCTRL.B.RUN   = 0b0010;

        // complementary PWM pair 안함. 독립적으로 쓸 거임
        FLEXPWM_0.SUB[1].CTRL2.B.INDEP = 1;
        FLEXPWM_0.SUB[1].INIT.R = -3200;
        FLEXPWM_0.SUB[1].VAL[0].R = 0;
        FLEXPWM_0.SUB[1].VAL[1].R = 3200;

        //compare interrupt val0, val1, val2, val3 enable CMPIE REG
        FLEXPWM_0.SUB[1].INTEN.R = 0x0001;
```

```
        // PWM B Fault Mask : Turn on
        FLEXPWM_0.SUB[1].DISMAP.B.DISB = 0;
        FLEXPWM_0.MCTRL.B.LDOK = 0b0010; //1번모듈 ldok
        FLEXPWM_0.OUTEN.B.PWMB_EN = 0b0010;
}

void PWMISR(void)
{

        j++;
        ADC_Read();
        //Errdetect();
        H_BridgeRun();
        //if(OFF_SW) adc_offset_cal();
        //PWM 1번 모듈에서 발생한 CMPI FLAG 전부 내려
        FLEXPWM_0.SUB[1].STS.R = 0x0001;
}

void ADC_Read(void)// 0번 모듈 사용, 3번채널은 전류 , 0번채널은 전압
{
        ADC_0.MCR.B.NSTART = 1; // Module 0 Conversion Start
        while(ADC_0.MCR.B.NSTART) asm("nop");
        if(ADC_0.CDR[0].B.VALID == 1)
        {

                IadcDATA = (float)ADC_0.CDR[3].B.CDATA;
                IdcPre = -((float) ADC_0.CDR[3].B.CDATA - 512.0f) * (5.0f/1023.0f) *
10.0f;
                VdcPre = (float) ADC_0.CDR[0].B.CDATA * VdcScale; //VBAT값을 읽
어온거 보정
        }

        IdcFdb = IdcPre - IdcOffset;
}

void Errdetect(void)
{
        Err1 = SIU.GPDI[30].B.PDI; //PB 14 값을 읽어오겠다
        Err2 = SIU.GPDI[29].B.PDI; //PB 13 값을 읽어오겠다
        if(Err1 && !Err2)       ErrCode = 100;                  //Error    :    Over
Temperature or Voltage
```

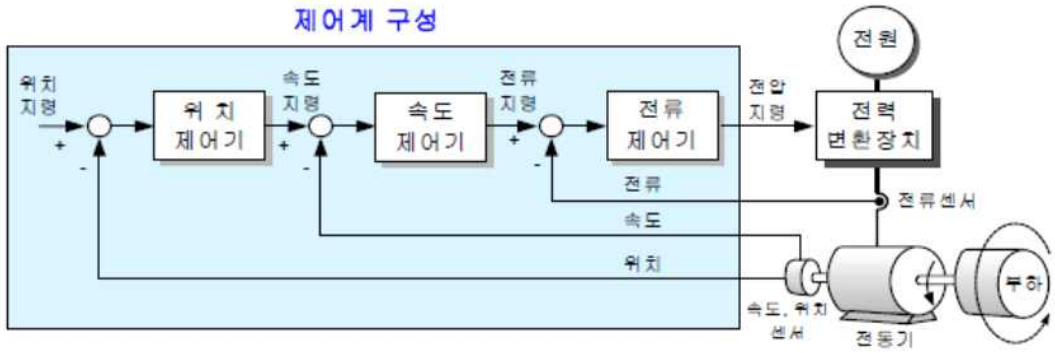
```
        else if(!Err1 && Err2)   ErrCode = 101;          //Error   :   Short   Circuit
detection
        else if(Err1 && Err2)    ErrCode = 111;          //Error : Under Voltage
        absIdcFdb = abs(IdcFdb);

        if((!OFF_SW) && ((abs(IdcFdb) > OverCurrent)))
        {
                ErrCode = 1;
//Error : Over Current
        }
        if(!OFF_SW && ((rpmFil > OverSpeed) || (rpmFil < -OverSpeed)))
        {
                ErrCode = 10;
//Error : Over Speed
        }

        ////////////////////

        if(ErrCode) DSPRUN = 0;          //숫자를 논리값으로 볼 때는 0이아니면 무조건
참, 0이어야 flase
        //아무튼 이 코드에서 뭔가 오류가 생겨가지구 errcode가 0이 아니게 되면 dsprun
= 0이 되면서 제어 멈춤
}

/*void SpeedCon(void)
{
        if((SpdCon==1)&&(DSPRUN==1))
        {
                rpmErr = rpmRef - rpmFil;
                spdPterm = kpSpd * rpmErr;
                spdPiterm = spdPterm + kiSpd * rpmErr + spdIterm;
                spdPiOut = bound(spdPiterm, spdPiUlim);
                spdIterm += kiSpd * rpmErr;
                spdIterm = bound(spdIterm, spdPiUlim);
                IdcRef = spdPiOut;
                FLEXPWM_0.SUB[1].VAL[4].R = (unsigned short)-PWM_B;
                FLEXPWM_0.SUB[1].VAL[5].R = (unsigned short) PWM_B;
                FLEXPWM_0.MCTRL.B.LDOK = 0b0010; //1번모듈 ldok
                FLEXPWM_0.OUTEN.B.PWMA_EN = 0b0010;

                SIU.GPDO[40].B.PDO = 1;
```

```
                SIU.GPDO[58].B.PDO = 1; //Gate High
                SIU.GPDO[59].B.PDO = 1; //Gate Low
                SIU.GPDO[61].B.PDO = 1;
        }
        else
        {
                SIU.GPDO[40].B.PDO = 0;
                SIU.GPDO[58].B.PDO = 0;
                SIU.GPDO[59].B.PDO = 0;
                SIU.GPDO[61].B.PDO = 0;
                FLEXPWM_0.OUTEN.B.PWMA_EN = 0x0;
                IdcRef = 0.0;
                VdcRef = 0.0;
                VdcPterm = 0.;
                VdcIterm = 0.0;
                VdcPiterm = 0.0;
        }
}*/

void H_BridgeRun(void)
{
        if(DSPRUN)
        {
                IdcErr = IdcRef - IdcFdb;
                VdcPterm = Kpc * IdcErr;
                VdcPiterm = VdcPterm + Kic * IdcErr + VdcIterm;
                VdcRef = bound(VdcPiterm, vRefUlim);
                VdcIterm += Kic * IdcErr;
                VdcIterm = bound(VdcIterm, vRefUlim);
                PWM_B = (int16_t)(VdcRef*PWM_Scale) + PWM_Peak; // 1600/12
//1~3ms 안에 전류가 도달해야함
                FLEXPWM_0.SUB[1].VAL[4].R = (unsigned short)-PWM_B;
                FLEXPWM_0.SUB[1].VAL[5].R = (unsigned short) PWM_B;
                FLEXPWM_0.MCTRL.B.LDOK = 0b0010; //1번모듈 ldok
                FLEXPWM_0.OUTEN.B.PWMA_EN = 0b0010;
                SIU.GPDO[40].B.PDO = 1;
                SIU.GPDO[58].B.PDO = 1; //Gate High
                SIU.GPDO[59].B.PDO = 1; //Gate Low
                SIU.GPDO[61].B.PDO = 1;
        }
        else
```

```
        {
                SIU.GPDO[40].B.PDO = 0;
                SIU.GPDO[58].B.PDO = 0;
                SIU.GPDO[59].B.PDO = 0;
                SIU.GPDO[61].B.PDO = 0;
                FLEXPWM_0.OUTEN.B.PWMA_EN = 0x0;
                IdcRef = 0.0;
                VdcRef = 0.0;
                VdcPterm = 0.;
                VdcIterm = 0.0;
                VdcPiterm = 0.0;
        }
}

void adc_offset_cal(void)
{
        offsetTimer++;
        if(offsetTimer > 10000)
        {
                OFF_SW = 0;
                offsetTimer = 10001;
        }
        else
        {
                DSPRUN = 0;
                OFF_SW = 1;
                ErrCode=0;
                LPF(IdcOffset, IdcPre, IdcOffsetLpfFct);
        }
}
```

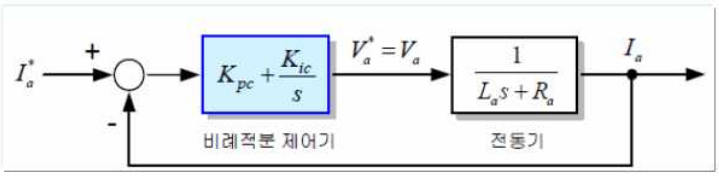
=> PIT를 100ms 초기설정 후 PIT ISR을 활성화한다. 이후 e-Timer 모듈을 초기설정하고 PIT ISR 함수를 설정한다. 마지막으로 EncoderFdb 함수를 세팅한다.

1.1 직류전동기의 제어계 구성



=> 일반적으로 직류전동기는 전류지령을 통해 속도를 제어한다. 전류에 따라 속도가 변화하며 부하가 커도 일정 전류가 흐르게 만들어주어 괴대한 전류가 흐르지 않도록 한다.

1.2 전류제어기 전달함수



< 전류제어기 블록다이어그램 & 전달함수 >

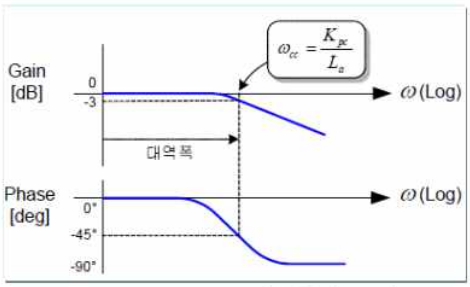
$$K_p = L\omega_c$$

$$K_i = R\omega_c$$

$$I(s) = \frac{\alpha\frac{K_p}{L}s + \frac{K_i}{L}}{s^2 + \frac{K_p + R}{L}s + \frac{K_i}{L}} I^*(s) \Rightarrow \frac{I(s)}{I^*(s)} = \frac{\omega_c}{s + \omega_c}$$

*Pole-Zero Cancellation 기법 이용하여 전류제어기를 1차 전달함수로 만들 수 있다

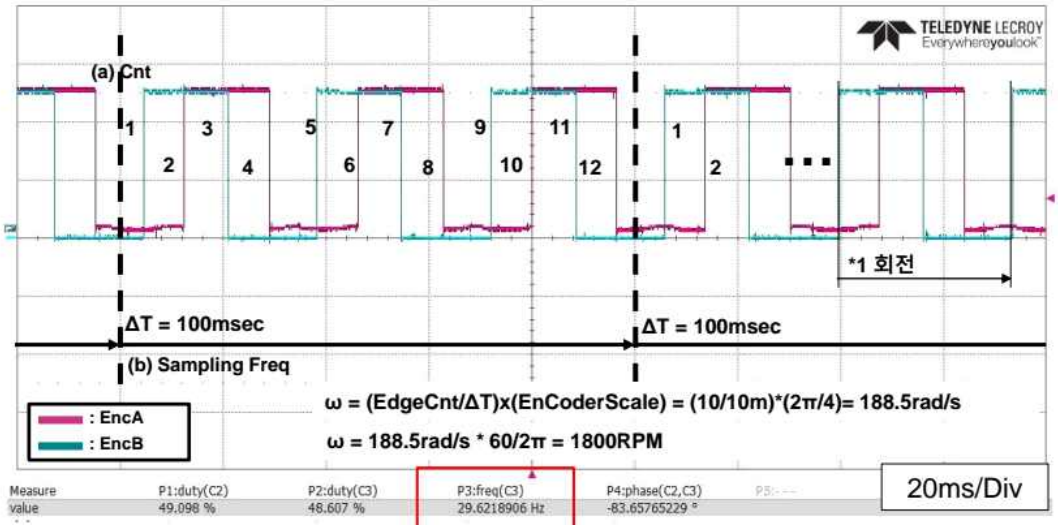
- ω_c: 주파수대역폭(Bandwidth)
- Overshoot 없는 전류제어기
- 주파수대역폭 설정으로 이득선정



< Pole-Zero Cancellation 전류제어기 보드선도 >

=> 전류제어기 전달함수에서는 비례적분 제어기를 활용한다. LPF를 사용하여 오버슛이 발생하지 않게 하며 차단주파수에 따라 값이 변경된다. P는 비례제어기이며, I는 적분제어기이다. S변환된 값을 Z변환 하여 Backward, Forward, Trapezoidal rule을 사용하여 적분을 한다. PI제어로 제어 정밀도가 향상되며 응답성이 개선되며 신뢰성이 대폭 향상하고 보수성, 유연성이 증가한다. 또한 파라미터의 설정이 용이해진다.

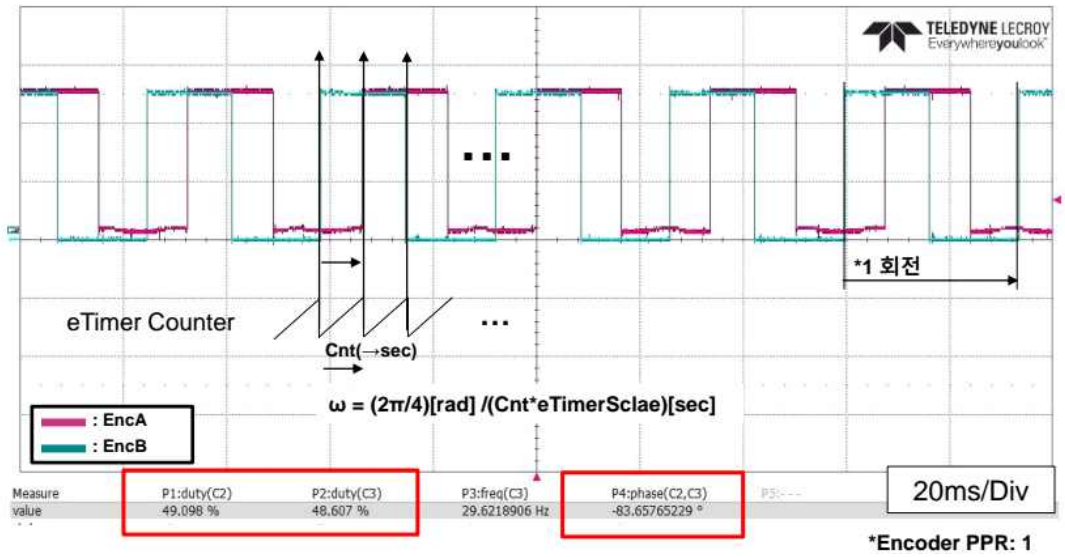
- Enc신호 속도환산 - M방식 (시간당 펄스수)



=> M방식은 pulse가 많을 때 유리하다. 시간당 pulse 수를 측정하는 방식이다.

=> rps는 1초당 pulse이며 1초가 매우 긴 시간이기 때문에 100ms당 pluse로 측정한다. (시간으로 scaling)

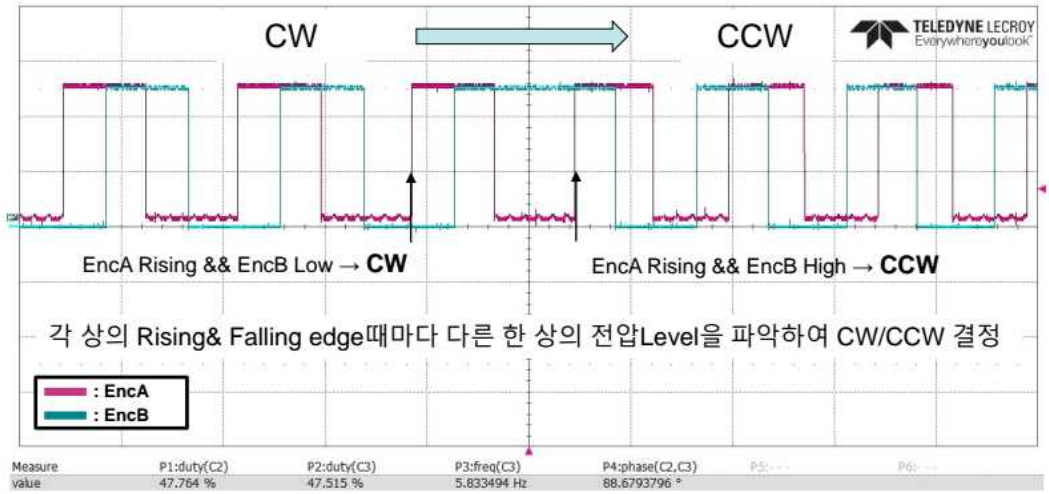
- Enc신호 속도환산 - T방식 (일정 회전량당 걸리는 시간) *EdgeToEdge = $\frac{2\pi}{4}$ [rad]



=> T방식은 pulse가 적을 때 유리하다. 1개의 pulse가 생성되는 시간, 즉 일정 회전량 당 걸리는 시간이다.

=> 쿼드러쳐 엔코더를 사용하며 타이머를 특정한다. 쿼드러쳐 엔코더란 2 개의 출력 채널을 가지고 있으며 구형 전자파를 반복하며 위상이 90도이다.

- Enc신호 방향결정로직



=> Rising&Falling edge에서 다른 한 상의 전압 Level을 파악하여 시계방향과 반시계방향의 모터 방향을 결정한다.

27.7.2.5 QUADRATURE-COUNT mode

When the CNTMODE field is set to 100, the counter will decode the primary and secondary external inputs as quadrature encoded signals. Quadrature signals are usually generated by rotary or linear sensors used to monitor movement of motor shafts or mechanical equipment. The quadrature signals are square waves that are 90 degrees out of phase. The decoding of quadrature signal provides both count and direction information.

Figure 27-24 shows a timing diagram illustrating the basic operation of a quadrature incremental position encoder.

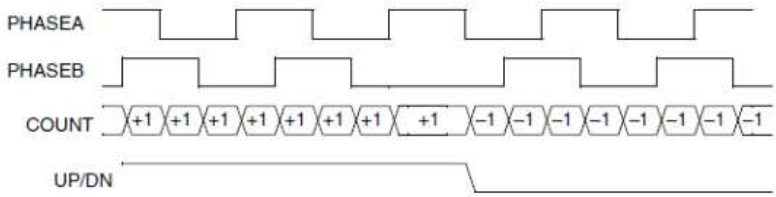
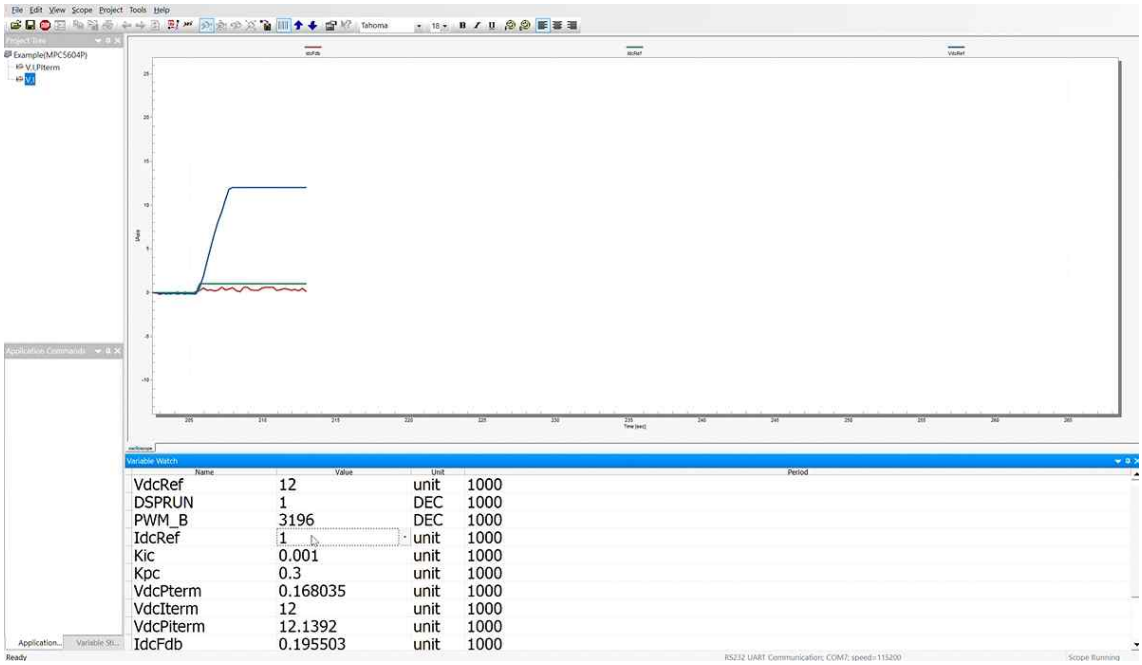


Figure 27-24. Quadrature Incremental position encoder

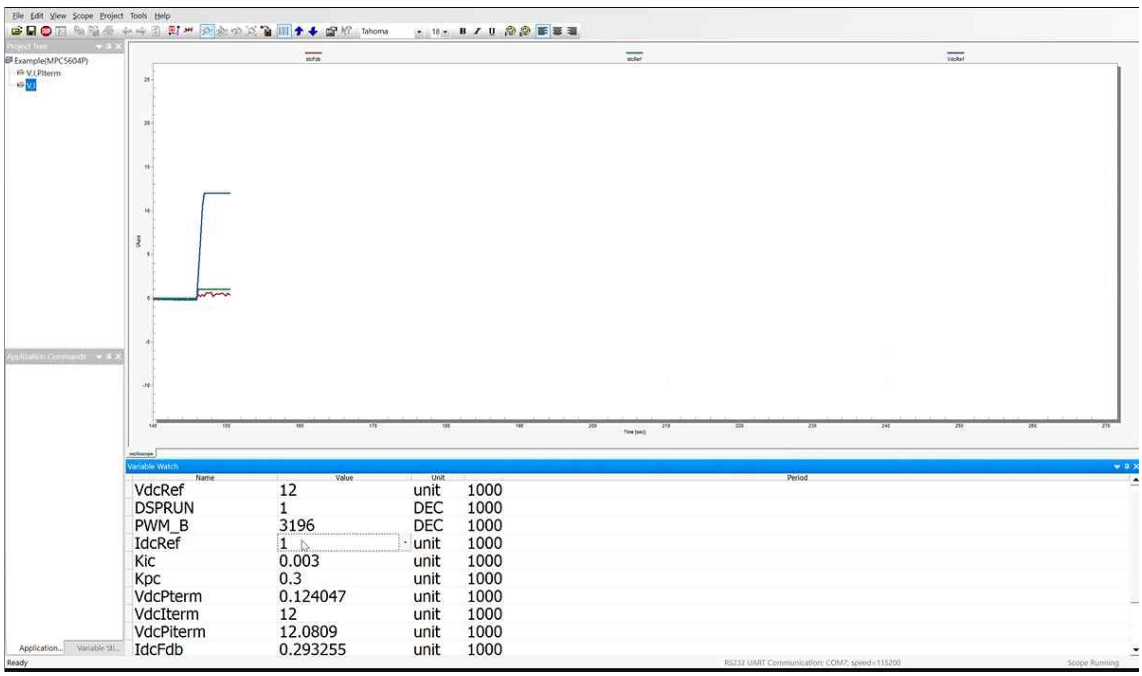
=> 4상한 카운터를 사용하여 간편한 엔코더 신호처리가 가능하다. Phase A가 상한일 때 Phase B가 하한, 상한인 경우와 Phase A가 하한일 때 Phase B가 하한, 상한인 경우가 존재한다. 또한 count가 증감하며 방향이 바뀌게 된다.

1. eTimer Count값을 Load한다.
2. Counter의 변화량을 계산한다. (오버플로우 경우를 대비하여 shift 연산을 처리한다. -> 부호 비트를 처리하기 위함(음수 방지)) : 100ms마다 pulse가 변화한다.
3. eTimer Count 수를 Encoder scale에 맞게 rad로 변환한다.
4. 샘플링 주기로 나누어 각속도를 변환한다.
5. LPF 후 RPM으로 환산하여 속도제어기 Fdb 변수로 사용한다.



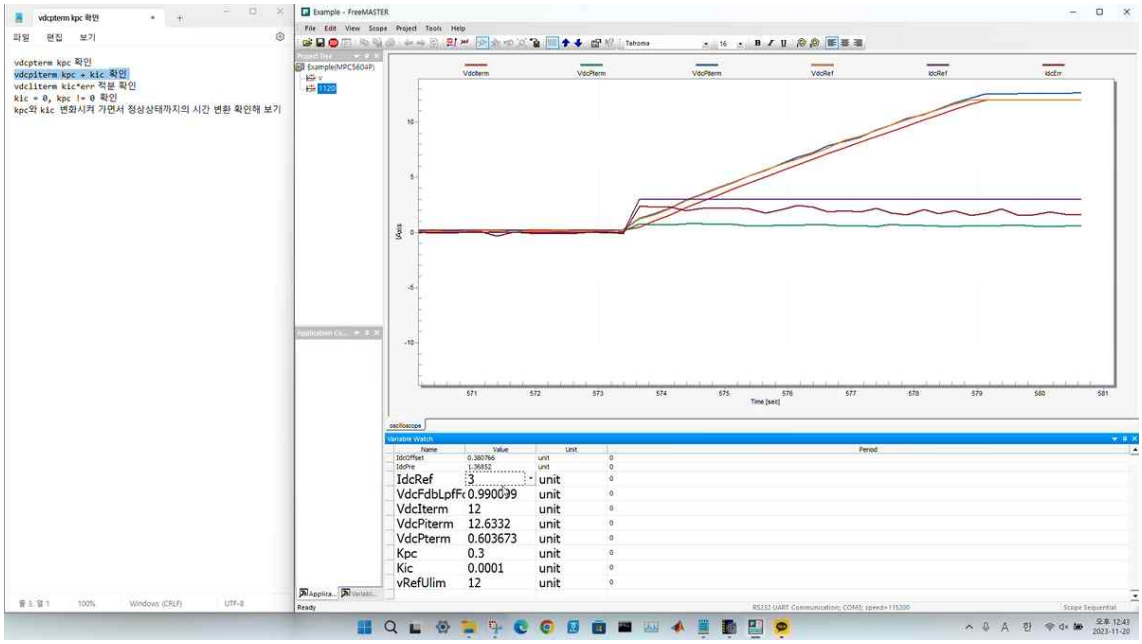
=> P계수 0.3, I계수 0.001일때 전류 1A 입력 시 전압의 변화

=> 적분값을 조금씩 반영하여 가속하는 속도가 비교적 느림



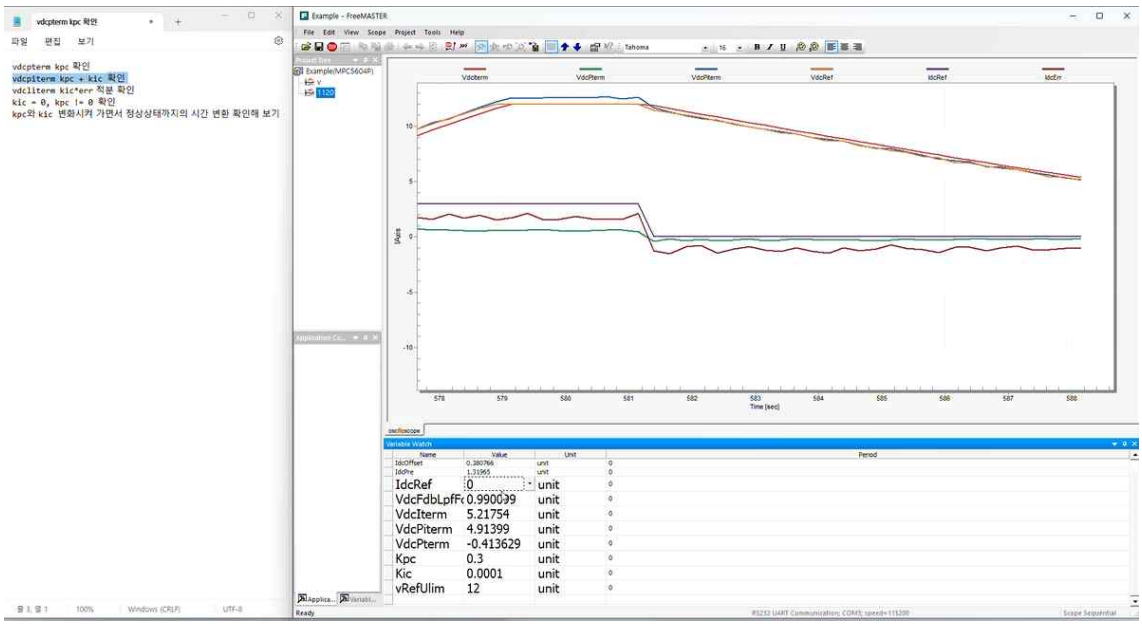
=> P계수 0.3, I계수 0.003일때 전류 1A 입력 시 전압의 변화

=> 적분값을 비교적 많이 반영하여 가속하는 속도가 비교적 빠름

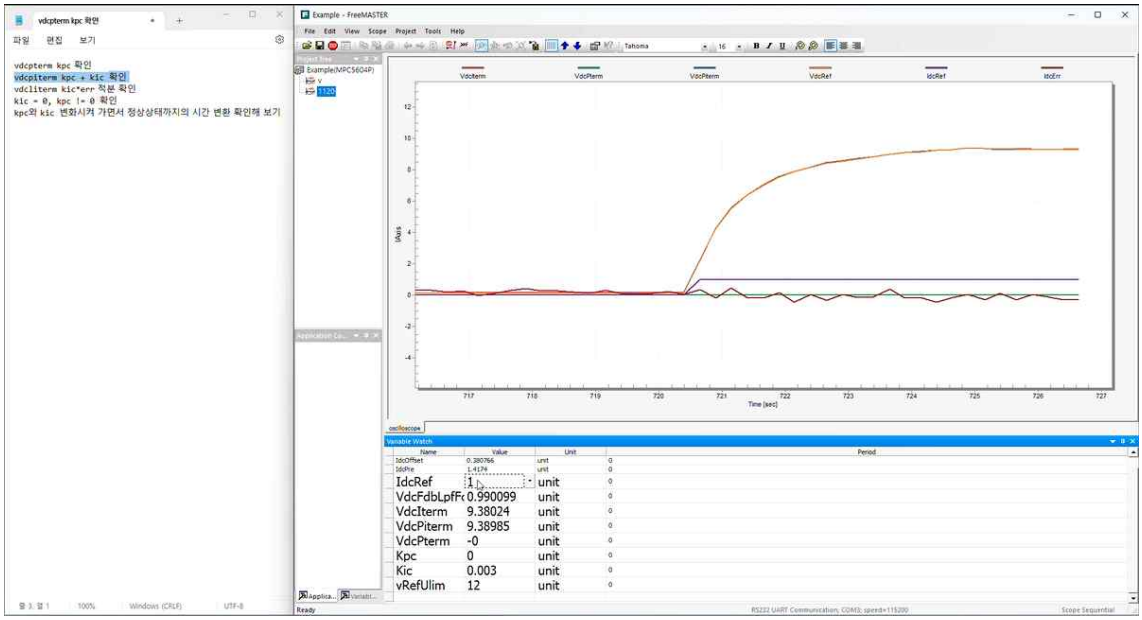


=> P계수 0.3, I계수 0.0001일때 전류 3A 입력 시 전압의 변화

=> 적분값을 매우 조금씩 반영하여 가속하는 속도가 비교적 느림

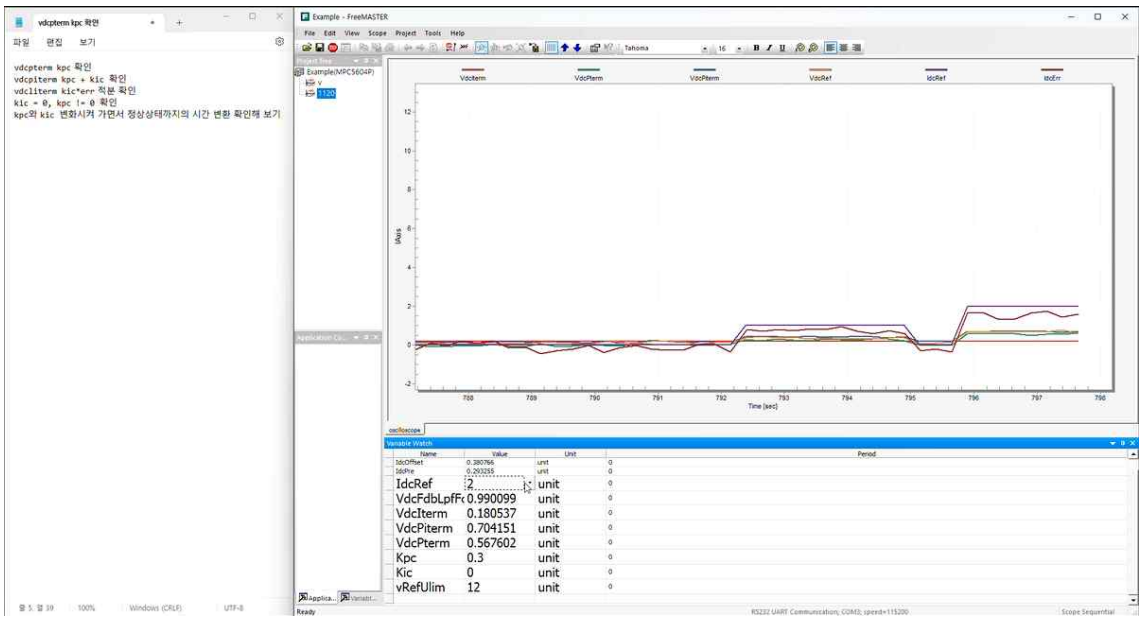


=> 적분값을 매우 조금씩 반영하여 감속하는 속도가 비교적 느림



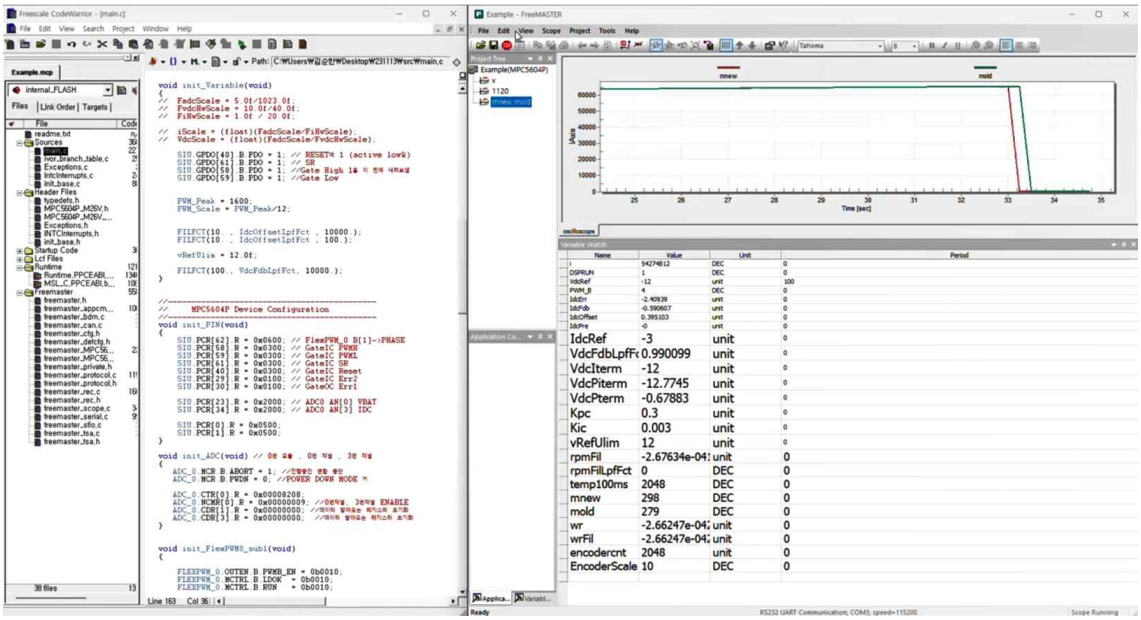
=> P계수 0, I계수 0.003일때 전류 1A 입력 시 전압의 변화

=> P가 반영이 안되기 때문에 상승시간은 길어져 목표값에 느리게 도달하고, 대신 오버슈트는 감소한다.

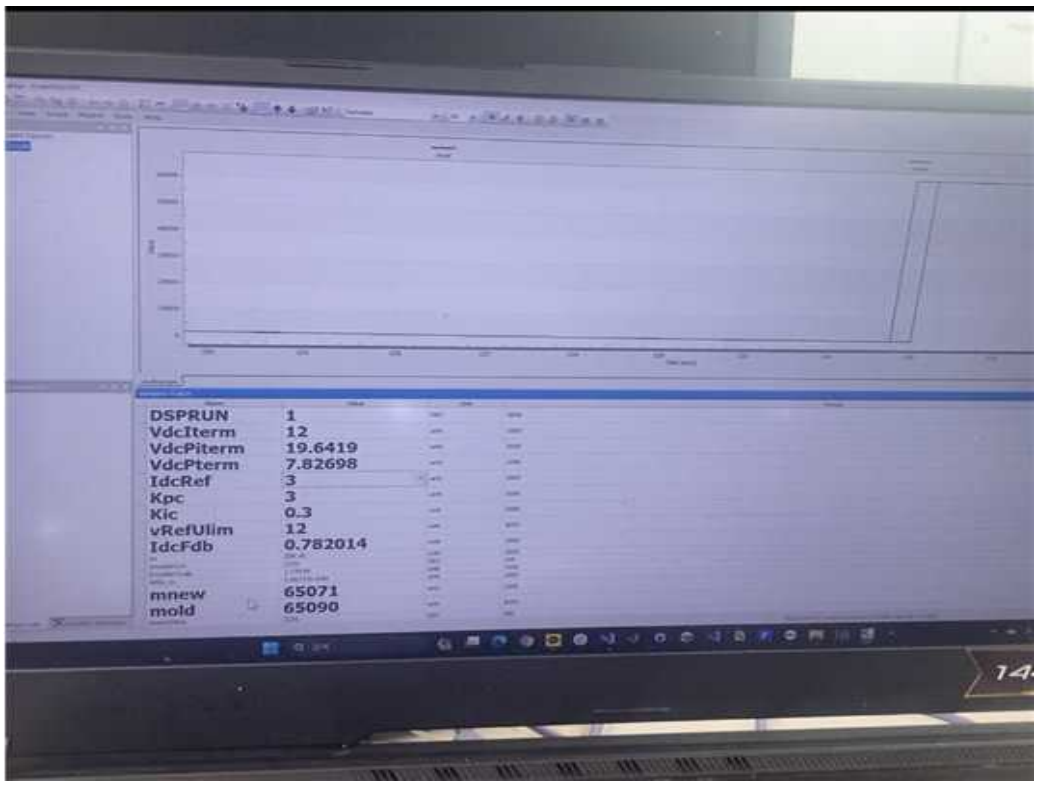


=> P계수 0.3, I계수 0일때 전류 2A 입력 시 전압의 변화

=> I가 반영이 안되기 때문에 시스템이 오차를 계속해서 누적하지 않고, 정상 상태에서의 오차를 제거하는 데 어려움이 있을 수 있다.



=> M방식으로 E-timer을 측정하여 전류가 변수의 최소 표현값까지 감소하고 최소 표현값 이후에 다시 최대 표현값부터 역으로 측정함. (2의 16승 = 65,536)



=> M방식으로 음수의 전류를 입력하였을 때 E-timer을 측정하여 변수의 최대 표현값까지 증가하고 최대 표현값 이후에 다시 0부터 측정함. (2의 16승 = 65,536)