リストA bldc_test.cのメイン関数check_bldc_start

```
void check_bldc_start(void)
{
    static int sBldcStartCnt = 0;
    switch(gBLDCState){
        case RESET: // 初期状態(変数初期化してREDYへ)
            sBldcStartCnt = 0;
            gBLDCStartReq = BLDC_STOP_REQ;
            set_bldc_drive_state(STANBY);
            gBLDCState = READY;
            break;
        case READY: // READY状態(SW1_ON待ち)
            if(get_start_sw() == 1){
                sBldcStartCnt++;
            }
            else{
                sBldcStartCnt = 0;
            }
            if(sBldcStartCnt > LATCHCNT){
                sBldcStartCnt = 0;
                gBLDCStartReq = BLDC_START_REQ;
                reset_bldc_error();
                bldc_start();
                gBLDCState = START;
            }
            break;
        case START: // 駆動開始と速度制御
            if(get_start_sw() == 0){
                sBldcStartCnt++;
            }
            else{
                sBldcStartCnt = 0;
            }
            if(sBldcStartCnt > LATCHCNT){
                sBldcStartCnt = 0;
                gBLDCStartReq = BLDC_STOP_REQ;
                gBLDCState = STOP;
            }
            break;
        case STOP: // 減速制御と再スタート監視
            if(get_start_sw() == 1){
                sBldcStartCnt++;
            }
            else{
                sBldcStartCnt = 0;
            }
            if(sBldcStartCnt > LATCHCNT){
                sBldcStartCnt = 0;
                gBLDCStartReq = BLDC_START_REQ;
                gBLDCState = START;
            }
            break;
        default: // デフォルトはリセットと同等処理
            sBldcStartCnt = 0;
            gBLDCStartReq = BLDC_STOP_REQ;
            gBLDCState = READY;
            break;
    }

}
```

リストB cmt.cのbldc_speed_interrupt

```
void bldc_speed_interrupt(void)
{
    float   bldc_speed = 0;
    static int sBLDCAdustCnt = 0;
    static float sBLDCTargetSpeed = 0;
    int cur_adjust_cnt = 0;
    static int sStartWaitCnt = 0;

    switch (gBLDCDriveST){
        case STANBY:
            reset_mtu3();
            reset_cmt();
            gBLDCDriveST = CURADJUST;
            break;
        case CURADJUST:
            cur_adjust_cnt = get_current_adjust_cnt();
            if( (cur_adjust_cnt >= CURRENT_ADJ_CNT) && (get_bldc_start() == BLDC_START_REQ) ){
                gBLDCDriveST = START1;
                bldc_start();
            }
            break;
        case START1: // 1回目d軸電流制御
            if(gId_ref > START_CURRENT){
                sBLDCAdustCnt = 0;
                gBLDCDriveST = ADJUST1;
            }
            else{
                gId_ref = gId_ref + DELTA_CURRENT;
            }
            break;
        case ADJUST1: // 1回目位相(磁石位置)調整
            if(sBLDCAdustCnt > ADJUST_CNT){
                set_encd_pos(0);
                gId_ref = 0;
                gBLDCDriveST = START2;
            }
            else{
                sBLDCAdustCnt++;
            }
            break;
        case START2: // 2回目d軸電流制御
            if(gId_ref > START_CURRENT){
                sBLDCAdustCnt = 0;
                gBLDCDriveST = ADJUST2;
            }
            else{
                gId_ref = gId_ref + DELTA_CURRENT;
            }
            break;
        case ADJUST2: // 2回目位相(磁石位置)調整
            if(sBLDCAdustCnt > ADJUST_CNT){
                gId_ref = 0;
                clrpsw_i();
                sBLDCTargetSpeed = get_vr_speed();
                setpsw_i();
                gBLDCRefSpeed = 0;
                gBLDCDriveST = START_WAIT;
                sStartWaitCnt = 0;
            }
            else{
                sBLDCAdustCnt++;
            }
            break;
```

```
case START_WAIT: // 位相調整完了後安定待ち(念のため)
    if(sStartWaitCnt > 1){
        gBLDCDriveST = DRIVE;
    }
    sStartWaitCnt++;
    break;
case DRIVE: // モータ速度制御(速度PI制御)
    clrpsw_i();
    sBLDCTargetSpeed = get_vr_speed();
    setpsw_i();
    if(sBLDCTargetSpeed > gBLDCRefSpeed){
        gBLDCRefSpeed = gBLDCRefSpeed + BLDC_ACC;
        if(gBLDCRefSpeed > sBLDCTargetSpeed){
            gBLDCRefSpeed = sBLDCTargetSpeed;
        }
    }
    else if(sBLDCTargetSpeed < gBLDCRefSpeed){
        gBLDCRefSpeed = gBLDCRefSpeed - BLDC_DCC;
        if(gBLDCRefSpeed < sBLDCTargetSpeed){
            gBLDCRefSpeed = sBLDCTargetSpeed;
        }
    }
    bldc_speed = get_bldc_speed();
    gIq_ref = speed_pi_ctrl(gBLDCRefSpeed, bldc_speed, KP_SPD, KI_SPD, BLDC_IQ_LIMIT, &gSpdsum);
    if(get_bldc_start() == BLDC_STOP_REQ){
        gBLDCDriveST = BRAKE;
    }
    break;
case BRAKE: // モータ減速制御(速度PI制御)
    gBLDCRefSpeed = gBLDCRefSpeed - BLDC_DCC;
    if(gBLDCRefSpeed < 0){
        gBLDCRefSpeed = 0;
    }
    bldc_speed = get_bldc_speed();
    gIq_ref = speed_pi_ctrl(gBLDCRefSpeed, bldc_speed, KP_SPD, KI_SPD, BLDC_IQ_LIMIT, &gSpdsum);
    if(get_bldc_start() == BLDC_START_REQ){
        clrpsw_i();
        sBLDCTargetSpeed = get_vr_speed();
        setpsw_i();
        gBLDCRefSpeed = 0;
        gBLDCDriveST = DRIVE;
    }
    break;
default:
    bldc_stop();
        break;
    }
}
```

リストC cmt.cのspeed_pi_ctrl

```
float speed_pi_ctrl(float speed_ref, float speed, float kp, float ki, float limit, float *psum)
{
    /* ------------- vd出力---------------- */
    float iref, iref_p, iref_i;
    float spd_diff;

    spd_diff = speed_ref - speed;

    /* P項 */
    iref_p =kp * spd_diff;

    /* I項 */
    iref_i = *psum;
```

```
    iref_i = iref_i + ki * spd_diff;
    /* 積分項上下限チェック*/
    if(iref_i > limit){
        iref_i = limit;
    }
    else if(iref_i < (-limit) ){
        iref_i = -limit;
    }
    *psum = iref_i;
    /* PI出力 */
    iref = iref_p + iref_i;
    /* iq上下限チェック*/
    if(iref > limit){
        iref = limit;
    }
    else if(iref < (-limit) ){
        iref = -limit;
    }
    return iref;
```

リストD mtu3.cのbldc interrupt

```
  void bldc_interrupt(void)
{
    bldc_current_interrupt();
    gSpdPICnt++;
    if( gSpdPICnt >= SPEED_PI_CNT){
        bldc_speed_interrupt();
        gSpdPICnt = 0;
    }
    clear_mot_interrupt();
}
```

リストE mtu3.cのbldc_current_interrupt

```
void bldc_current_interrupt(void)
{
    float vdc;
    float id, iq, id_ref, iq_ref;
    float vd, vq, decp_vd, decp_vq;
    float vu, vv, vw;
    float bldc_refspeed;

    /*1:UVW 相電流検出(V 相は計算により算出) および2:インバータ母線電圧検出*/
    clrpsw_i();
    get_mot_current(&gIu, &gIw, &vdc);
    setpsw_i();
    gVdc = vdc;

    /* 始動時電流調整 */
    if(gAdjust_Current_Cnt <= CURRENT_ADJ_CNT){
        gAdjust_Iu += LPF_K_CURRENT * (gIu - gAdjust_Iu);
        gIu = -gIu + gAdjust_Iu;
        gAdjust_Iw += LPF_K_CURRENT * (gIw - gAdjust_Iw);
        gIw = -gIw + gAdjust_Iw;

             gAdjust_Current_Cnt++;
    }
    else{
        /* 駆動時電流検出 */
        gIu = -gIu + gAdjust_Iu;
        gIw = -gIw + gAdjust_Iw;

        gIu = gIu_Pre + LPF_K_CURRENT * (gIu - gIu_Pre);
```

```
    gIu_Pre = gIu;
    gIw = gIw_Pre + LPF_K_CURRENT * (gIw - gIw_Pre);
    gIw_Pre = gIw;
}
gIv = - (gIu + gIw);
/* 3:モーター速度、位置検出*/
get_mot_speed(&gBldc_Angle, &gBldc_Speed);
/* 4:エラー判定*/
check_bldc_error(gIu, gIv, gIw, vdc, gBldc_Speed);

gRpm = gBldc_Speed/RPM_RAD_COF/7;
/* 5:dq変換 */
dq_conversion(gIu, gIv, gIw, gBldc_Angle);
get_dq_current(&id, &iq);
get_dq_ref(&id_ref, &iq_ref);
gId = id;
gIq = iq;
/* 6:電流PI制御 */
vd = currenr_pi_ctrl(id_ref, id, KP_ID, KI_ID, BLDC_VD_LIMIT, &gVdIsum);
vq = currenr_pi_ctrl(iq_ref, iq, KP_IQ, KI_IQ, BLDC_VQ_LIMIT, &gVqIsum);

/* 非干渉制御 */
    bldc_refspeed = get_bldc_refspeed();
    decp_vd = bldc_refspeed * BLDC_LQ * iq;
    decp_vq = bldc_refspeed * (BLDC_LD * id + BLDC_QSI);
    vd = vd - decp_vd;
    vq = vq + decp_vq;

    /* 7:dq逆変換 */
    dq_reconversion(vd, vq, gBldc_Angle, &vu, &vv, &vw);

    /* 8:pwm_duty設定 */
    set_motpwm_uvw(vu, vv, vw, vdc);
}
```

リストF mtu3.cのdq_conversion

```
void dq_conversion(float iu, float iv, float iw, float angle)
{
    float id_u, id_v, id_w;
    float iq_u, iq_v, iq_w;

    id_u = cosf(angle) * iu;
    id_v = cosf(angle - (BLDC_2PI / 3)) * iv;
    id_w = cosf(angle + (BLDC_2PI / 3)) * iw;
    gId = BLDC_SQRT_2_3 * (id_u + id_v + id_w);

    iq_u = sinf(angle) * iu;
    iq_v = sinf(angle - (BLDC_2PI / 3)) * iv;
    iq_w = sinf(angle + (BLDC_2PI / 3)) * iw;
    gIq = (-1) * BLDC_SQRT_2_3 * (iq_u + iq_v + iq_w);
}
```

リストG mtu3.cのdq_reconversion

```
void dq_reconversion(float vd_ref, float vq_ref, float angle, float *pvu, float *pvv, float *pvw)
{
    float vref_u, vref_v, vref_w;
    float vref_limit = VDC/2;
    /*---------------------------------------------------- dq逆変
換---------------------------------------------------- */
    vref_u = BLDC_SQRT_2_3 * (vd_ref * cosf(angle) - vq_ref * sinf(angle) );
    vref_v = BLDC_SQRT_2_3 * (vd_ref * cosf(angle - BLDC_2PI/3 ) - vq_ref * sinf(angle - BLDC_2PI/3 ) );
    vref_w = BLDC_SQRT_2_3 * (vd_ref * cosf(angle + BLDC_2PI/3 ) - vq_ref * sinf(angle + BLDC_2PI/3 ) );
```

```
/* --------------------変換後各相電圧上下限制限 ---------------*/
    /*---------- U相--------- */
if ( vref_u > vref_limit){
            vref_u = vref_limit;
    }
 else if (vref_u < -vref_limit){
        vref_u = -vref_limit;
 }
 else{
 }
 /*---------- V相----------- */
if ( vref_v > vref_limit){
         vref_v = vref_limit;
 }
 else if (vref_v < -vref_limit){
        vref_v = -vref_limit;
 }
 else{
    }
 /* -----------W相----------- */
if ( vref_w > vref_limit){
         vref_w = vref_limit;
     }
     else if (vref_w < -vref_limit){
            vref_w = -vref_limit;
     }
        else{
        }
/*----3 相PWMDuty へ割当て----*/
    *pvu = vref_u;
    *pvv = vref_v;
    *pvw = vref_w;
}
```

リストH mtu3.cのcurrenr_pi_ctrl

```
float currenr_pi_ctrl(float current_ref, float current, float kp, float ki, float limit, float *psum)
{
    /* ------------- vd出力---------------- */
    float vref, vref_p, vref_i;
    float idiff;
    /* Δ i 算出*/
    idiff = current_ref - current;
    /* P項 */
    vref_p = kp * idiff;
    /* I項 */
    vref_i = *psum;
    vref_i = vref_i + ki * idiff;
    /* 積分項上下限チェック*/
    if(vref_i > limit){
        vref_i = limit;
    }
    else if(vref_i < (-limit) ){
        vref_i = -limit;
    }
    *psum = vref_i;
    /* PI出力 */
    vref = vref_p + vref_i;
    /* 上下限チェック*/
    if(vref > limit){
        vref = limit;
    }
    else if(vref < (-limit) ){
        vref = -limit;
```

```
    }
    return vref;
}
```

リストI set_motpwm_uvw

```
void set_motpwm_uvw(float pwm_u, float pwm_v, float pwm_w, float vdc)
{
    float carrier_half_period, vdc_n;

    carrier_half_period = (float)((SOHO_PWM_PERIOD + DEAD_TIME)/2);
    vdc_n = (vdc / 2);

    pwm_u = -pwm_u;
    pwm_v = -pwm_v;
    pwm_w = -pwm_w;

    MTU3.TGRD = (unsigned short)(((pwm_u * carrier_half_period) / vdc_n) + carrier_half_period);
    MTU4.TGRC = (unsigned short)(((pwm_v * carrier_half_period) / vdc_n) + carrier_half_period);
    MTU4.TGRD = (unsigned short)(((pwm_w * carrier_half_period) / vdc_n) + carrier_half_period);
}
```

リストJ sns.cのget_mot_current

```
void get_mot_current(float *iu, float *iw, float *vdc)
{
    float ad_iu;
    float ad_iw;
    float ad_vdc;

    S12AD0.ADCSR.BIT.ADST = 1;
    while (S12AD0.ADCSR.BIT.ADST == 1); /* 変換待ち */
     ad_iu = (float)S12AD0.ADDR0A;   /* センサ値取得 */
        ad_iw = (float)S12AD0.ADDR1;
      ad_vdc = (float)S12AD0.ADDR2;

     /* A-D→電流値変換 */
     ad_iu = ad_iu - NT_CURRENT_OFFSET;
     ad_iw = ad_iw - NT_CURRENT_OFFSET;
     ad_iu = ad_iu * AD_CURRENT_COF;
        ad_iw = ad_iw * AD_CURRENT_COF;
/*---iv 電流は別途iu, iw から算出iv = -(iu + iw)---*/
    /* A-D→電圧変換 */
     ad_vdc = ad_vdc * AD_VDC_COF;
     *iu = ad_iu;
    *iw = ad_iw;
    *vdc = ad_vdc;
}
```

リストK sns.cのget_mot_vd

```
float get_mot_vdc(void)
{
    float ad_vdc;

    S12AD0.ADCSR.BIT.ADST = 1;
     while (S12AD0.ADCSR.BIT.ADST == 1); /* 変換待ち */
     ad_vdc = (float)S12AD0.ADDR2;
        /* A-D→電圧変換 */
     ad_vdc = ad_vdc * AD_VDC_COF;
      return ad_vdc;
}
```

リストL sns.cのget_vr_speed

```
float get_vr_speed(void)
{
    unsigned short temp_vr;
    float trgt_speed;

    S12AD1.ADCSR.BIT.ADST = 1;
    while (S12AD1.ADCSR.BIT.ADST == 1); /* 変換待ち */
    temp_vr = S12AD1.ADDR2;             /* 抵抗値取得*/
/* AD →速度変換*/
    trgt_speed = (float)temp_vr;
    trgt_speed = trgt_speed * VR_SPD_COF + BLDC_MARGIN_MIN_RPM;
    gTgSpd = trgt_speed;
    /* rad/sに変換 */
    trgt_speed = trgt_speed * RPM_RAD_COF;
    /* 速度検査 */
    if(trgt_speed > BLDC_MAX_RAD){
        trgt_speed = BLDC_MAX_RAD;
    }
    else if(trgt_speed < BLDC_MIN_RAD){
        trgt_speed = BLDC MIN RAD;
    }
    trgt_speed = trgt_speed * BLDC_MGPOLE_PAIR;
    return (trgt_speed);
}
```

リストM bldc_test.cのcheck_bldc_error

```
 void check_bldc_error(float iu, float iv, float iw,
                       float vdc, float speed)
{
     /* 電流検査 */
    if ((iu > BLDC_CURRENT_LIMIT) ||
        (iu < -BLDC_CURRENT_LIMIT)) {
     /* U相電流超過 */
    gBldc_Error = BLDC_ERROR_OVERCURRENT_U;
    }
    if ((iv > BLDC_CURRENT_LIMIT) ||
        (iv < -BLDC_CURRENT_LIMIT)) {
         /* V相電流超過 */
        gBldc_Error = BLDC_ERROR_OVERCURRENT_V;
    }
    if ((iw > BLDC_CURRENT_LIMIT) ||
        (iw < -BLDC_CURRENT_LIMIT)) {
         /* W相電流超過 */
        gBldc_Error = BLDC_ERROR_OVERCURRENT_W;
    }
    /* 電圧確認 */
    if (vdc > BLDC_HIGHVOLT_LIMIT) {
         /* 端子電圧超過 */
        gBldc_Error = BLDC_ERROR_OVERVOLTAGE;
    }
    if (vdc < BLDC_LOWVOLT_LIMIT) {
         /* 端子電圧超過 */
        gBldc_Error = BLDC_ERROR_UNDERVOLTAGE;
    }
     /* 速度検査 */
    if (speed > BLDC_SPEED_LIMIT) {
         /* 速度超過 */
        gBldc_Error = BLDC_ERROR_OVERSPEED;
    }
    if (gBldc_Error != BLDC_NO_ERROR) {
        bldc_stop();
```

```
    }
}
```

リストN hwsetup.cのHardwareSetup

```
void HardwareSetup(void)
{
    /* システム・クロック初期化 */
    /* メイン = 振動子12MHz×8倍(96MHz) */
    SYSTEM.SCKCR.BIT.ICK = 0x00;
    /* 周辺 = 振動子12MHz×4倍(48MHz) */
    SYSTEM.SCKCR.BIT.PCK = 0x01;

    MSTP_MTU = 0; /* MTU用モジュール停止解除 */
        MSTP_S12AD0 = 0;
        MSTP_S12AD1 = 0;

    /* モータ・タイマ初期化(電流制御) */
     initialize_mtu34();
     initialize_pwm();
     initialize_mot_interrupt();
     initialize_encoder();
    /* モータ・タイマ初期化(速度制御) */
    initialize_cmt();
    /* I/Oポート初期化 */
    initialize_port();
    /* センサ類初期化 */
    initialize_sns();
    /* 安全機能初期化 */
    initialize_poe();
    initialize_wdt();
    /* SW,LED初期化*/
    initialize_ui();
}
```