```
#ifndef __controlSuite_double_sim_hpp__
#define __controlSuite_double_sim_hpp__


// ***** "double" precision CUR_MOD_MACRO and CUR_CONST_MACRO Macros (simulation)
************************************************************

typedef struct  { double  IDs;         // Input: Syn. rotating d-axis current (pu)
                  double  IQs;         // Input: Syn. rotating q-axis current (pu)
                  double  Wr;          // Input: Rotor electrically angular
velocity (pu)
                  double  IMDs;        // Variable: Syn. rotating d-axis
magnetizing current (pu)
                  double  Theta;       // Output: Rotor flux angle (pu)
                  double  Kr;          // Parameter: constant using in magnetizing
current calculation
                  double  Kt;          // Parameter: constant using in slip
calculation
                  double  K;           // Parameter: constant using in rotor flux
angle calculation
                  double  Wslip;       // Variable: Slip
                  double  We;          // Variable: Angular freq of the stator
                } CURMOD;

typedef struct  { double  Rr;          // Input: Rotor resistance (ohm)
                  double  Lr;          // Input: Rotor inductance (H)
                  double  fb;          // Input: Base electrical frequency (Hz)
                  double  Ts;          // Input: Sampling period (sec)
                  double  Kr;          // Output: constant using in magnetizing
current calculation
                  double  Kt;          // Output: constant using in slip
calculation
                  double  K;           // Output: constant using in rotor flux
angle calculation
                  double  Tr;          // Variable: Rotor time constant (sec)
                } CURMOD_CONST;


#define CUR_CONST_MACRO(v)      \
    v.Tr = v.Lr/v.Rr;           \
                                \
    v.Kr = v.Ts/v.Tr;           \
    v.Kt = 1/(v.Tr*2*PI*v.fb);  \
    v.K = v.Ts*v.fb;


double prev_IMDs = 1.0;
double guard__IMDS_zero(double imds_val)
{
    if(imds_val == 0)
    {
        imds_val = prev_IMDs;
    }
    else
    {
        prev_IMDs = imds_val;
    }
    return imds_val;
}
```

```
double saturate(double sum, double max, double min);  //(defined below)

#define CUR_MOD_MACRO(v)                                \
    v.IMDs +=  v.Kr * (v.IDs - v.IMDs);                  \
    v.IMDs = guard__IMDS_zero(v.IMDs);                        \
    v.Wslip =  saturate(v.Kt * v.IQs / v.IMDs, PI/.00005, -PI/.00005);    \
    v.We = v.Wr + v.Wslip;                                \
    v.Theta +=  v.K * v.We;                      \
                                                          \
    if (v.Theta > 2*PI)                                  \
       v.Theta -=  2*PI;                                 \
    else if (v.Theta < 2*PI)                             \
       v.Theta += 2*PI;

#define CURMOD_DEFAULTS { 0,0,0,0,0, \
                          0,0,0,0,0 \
                        }

#define CURMOD_CONST_DEFAULTS { 0,0,0,0, \
                                0,0,0,0  \
                              }

CURMOD cm1 = CURMOD_DEFAULTS;
CURMOD_CONST cm1_const = CURMOD_CONST_DEFAULTS;

    // Setting Lr = 2.71067e-06 or Ts = .0005 gives the effect we want. This makes
the gain 10x from the specified Lr/Rr. Find out why.
    // Setting to .01 also works and seems to provide a much higher gain but with
instability. (NOTE: 100 is the turns ratio between Stator and Rotor conductors?)
#define Lr_Rr_SCALING_CONST  .1

void init_cm1(void)
{

    // Initialize the CUR_MOD constant module     (Start with some values...)
    cm1_const.Rr = (.0001);                                  // RR;
    cm1_const.Lr = (Lr_Rr_SCALING_CONST * 2.71067e-05);   // LR;
    cm1_const.fb = .5 * VO_TRAJ /( 2 * PI);                // BASE_FREQ;
    cm1_const.Ts = .00005;
    CUR_CONST_MACRO(cm1_const)

    // Initialize the CUR_MOD module
    cm1.Kr = cm1_const.Kr;
    cm1.Kt = cm1_const.Kt;
    cm1.K  = cm1_const.K;




}

//
*****************************************************************************
*********************************************************


// ***** "double" precision PI_MACRO  Macro (simulation)
*****************************************************************************
****
```

```
double saturate(double sum, double max, double min)
{
if( sum > max )
    sum = max;
if( sum < min )
    sum = min;

return sum;
}

typedef struct {  double  Ref;            // Input: reference set-point
              double  Fbk;            // Input: feedback
              double  Out;            // Output: controller output
              double  Kp;             // Parameter: proportional loop gain
              double  Ki;             // Parameter: integral gain
              double  Umax;           // Parameter: upper saturation limit
              double  Umin;           // Parameter: lower saturation limit
              double  up;             // Data: proportional term
              double  ui;             // Data: integral term
              double  v1;             // Data: pre-saturated controller output
              double  i1;             // Data: integrator storage: ui(k-1)
              double  w1;             // Data: saturation record: [u(k-1) - v(k-
1)]
            } PI_CONTROLLER;



#define PI_MACRO(v)                                                    \
                                                                       \
/* proportional term */                                                \
v.up = v.Kp * (v.Ref - v.Fbk);                                         \
                                                                       \
/* integral term */                                                    \
v.ui = (v.Out == v.v1)?((v.Ki * v.up)+ v.i1) : v.i1;                   \
v.i1 = v.ui;                                                           \
                                                                       \
/* control output */                                                   \
v.v1 = v.up + v.ui;                                                    \
v.Out= saturate(v.v1, v.Umax, v.Umin);                                 \
//v.w1 = (v.Out == v.v1) ? _IQ(1.0) : _IQ(0.0);                        \

#define PI_CONTROLLER_DEFAULTS {        \
                        0,              \
                        0,              \
                        0,              \
                        1.0,            \
                        0.0,            \
                        1.0,            \
                        -1.0,           \
                        0.0,            \
                        0.0,            \
                        0.0,            \
                        0.0,            \
                        1.0             \
                       }
```

```
PI_CONTROLLER pi_spd = PI_CONTROLLER_DEFAULTS;
PI_CONTROLLER pi_id = PI_CONTROLLER_DEFAULTS;
PI_CONTROLLER pi_iq = PI_CONTROLLER_DEFAULTS;

uint16_t SpeedLoopPrescaler = 10;

void init_pi_spd_pi_id_pi_iq(void)
{
// Initialize the PI module for Id
    pi_spd.Kp=50.0;
    pi_spd.Ki=cm1_const.Ts*SpeedLoopPrescaler/0.5;
    pi_spd.Umax =80;
    pi_spd.Umin =-80;

// Initialize the PI module for Iq
    pi_id.Kp=10.0;
    pi_id.Ki=cm1_const.Ts/0.004;
    pi_id.Umax =DC_BUS_VOLTAGE;
    pi_id.Umin =-DC_BUS_VOLTAGE;

// Initialize the PI module for speed
    pi_iq.Kp=10.0;
    pi_iq.Ki=cm1_const.Ts/0.004;
    pi_iq.Umax =DC_BUS_VOLTAGE;
    pi_iq.Umin =-DC_BUS_VOLTAGE;



}




//
*****************************************************************************
*****************************************************


// *****  "double" precision CLARKE_MACRO Macro (simulation
*****************************************************************************
*

typedef struct {  double  As;          // Input: phase-a stator variable
                  double  Bs;          // Input: phase-b stator variable
                  double  Cs;          // Input: phase-c stator variable
                  double  Alpha;       // Output: stationary d-axis stator variable
                  double  Beta;        // Output: stationary q-axis stator variable
                } CLARKE;


#define  ONEbySQRT3   0.57735026918963    /* 1/sqrt(3) */

#define CLARKE_MACRO(v)                                          \
v.Alpha = v.As;                                                  \
v.Beta = v.As + (v.Bs * v.Bs) * ONEbySQRT3;

#define CLARKE1_MACRO(v)                                         \
        v.Alpha = v.As;                                          \
        v.Beta  = (v.Bs - v.Cs) * ONEbySQRT3;
```

```
#define CLARKE_DEFAULTS { 0, \
                          0, \
                          0, \
                          0, \
                          0, \
                        }

CLARKE clarke1 = CLARKE_DEFAULTS;

//
//---------------------------------------------------------------------------
//-----------------------------------------------------

// ***** "double" precision PARK_MACRO Macro (simulation)
//---------------------------------------------------------------------------
//---

typedef struct {  double  Alpha;       // Input: stationary d-axis stator variable
                  double  Beta;        // Input: stationary q-axis stator variable
                  double  Angle;       // Input: rotating angle (pu)
                  double  Ds;          // Output: rotating d-axis stator variable
                  double  Qs;          // Output: rotating q-axis stator variable
                  double  Sine;
                  double  Cosine;
                } PARK;

#define PARK_MACRO(v)                                                   \
                                                                        \
    v.Ds = v.Alpha * v.Cosine + v.Beta * v.Sine;                        \
    v.Qs = v.Beta * v.Cosine - v.Alpha * v.Sine;



#define PARK_DEFAULTS {   0, \
                          0, \
                          0, \
                          0, \
                          0, \
                          0, \
                          0, \
                          }

PARK park1 = PARK_DEFAULTS;

//
//---------------------------------------------------------------------------
//-----------------------------------------------------



#endif
```