```
#ifndef __controlSuite_fixed_sim_hpp__
#define __controlSuite_fixed_sim_hpp__




#define FPT_WBITS 8              //It looks like in IQmathLib.h, HVACI_Sensored.c
is using  GLOBAL_Q == 24
                                 //NOTE: Here FPT_WBITS means size of integer not
fraction! (Reversed meaning to GLOBAL_Q in TI IQ MATH)
                                  //According to various links including to this
one (https://s2.smu.edu/~mitch/class/5385/DSPArithmeticTutorial.pdf)
                                  //IQMathLib.h is where GLOBAL_Q is set when
using the C2000 development environment.
#include "./fptc-lib/src/fptc.h"



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


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




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


#define CUR_CONST_MACRO_fix(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;

fpt prev_IMDs_fix = FPT_ONE;
fpt guard__IMDS_zero_fix(fpt imds_val)
{
    if(imds_val == 0)
    {
        imds_val = prev_IMDs_fix;
    }
    else
    {
        prev_IMDs_fix = imds_val;
    }
    return imds_val;
}

#define CUR_MOD_MACRO_fix(v)                                \
    v.IMDs +=  fpt_mul(v.Kr,(v.IDs - v.IMDs));       \
    v.IMDs = guard__IMDS_zero_fix(v.IMDs);            \
    v.Wslip = fpt_div(fpt_mul(v.Kt,v.IQs),v.IMDs);    \
    v.We = fl2fpt(v.Wr) + v.Wslip;                          \
    v.Theta +=  fpt_mul(v.K,v.We);                   \
                                                    \
    if (v.Theta > FPT_ONE)                           \
       v.Theta -= FPT_ONE;                          \
    else if (v.Theta < fl2fpt(0))                       \
       v.Theta += FPT_ONE;



#define CURMOD_DEFAULTS_fix { 0,0,0,0,0, \
                          0,0,0,0,0 \
                        }

#define CURMOD_CONST_DEFAULTS_fix { 0,0,0,0, \
                                0,0,0,0  \
                               }

CURMOD_fix cm1_fix = CURMOD_DEFAULTS_fix;
CURMOD_CONST_fix cm1_const_fix = CURMOD_CONST_DEFAULTS_fix;

    // 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_fix(void)
{

    // Initialize the CUR_MOD constant module     (Start with some values...)
    cm1_const_fix.Rr = (.0001);                                 // RR;
    cm1_const_fix.Lr = (Lr_Rr_SCALING_CONST * 2.71067e-05);   // LR;
    cm1_const_fix.fb = .5 * VO_TRAJ /( 2 * PI);               // BASE_FREQ;
    cm1_const_fix.Ts = .00005;
```

```
    CUR_CONST_MACRO_fix(cm1_const_fix)

    // Initialize the CUR_MOD module
    cm1_fix.Kr = fl2fpt(cm1_const_fix.Kr);
    cm1_fix.Kt = fl2fpt(cm1_const_fix.Kt);
    cm1_fix.K  = fl2fpt(cm1_const_fix.K);

}


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


// ***** "fixed" precision PI_MACRO  Macro (simulation)
*****************************************************************************
*****

fpt saturate_fix(fpt sum, fpt max, fpt min)
{

       // This check may not be necessary.
    if((sum > 0) && (sum > max))
       return max;
    else if((sum < 0) && (sum < min))
       return min;
    else
       return sum;

}

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


#define PI_MACRO_fix(v)                                                 \
                                                                        \
    /* proportional term */                                             \
    v.up = fpt_mul(v.Kp, (v.Ref - v.Fbk));                              \
                                                                        \
    /* integral term */                                                 \
    v.ui = (v.Out == v.v1)?(fpt_mul(v.Ki, v.up)+ v.i1) : v.i1;          \
    v.i1 = v.ui;                                                        \
                                                                        \
    /* control output */                                                \
    v.v1 = v.up + v.ui;                                                 \
```

```
    v.Out= saturate_fix(v.v1, v.Umax, v.Umin);                  \
    //v.w1 = (v.Out == v.v1) ? _IQ(1.0) : _IQ(0.0);             \


#define PI_CONTROLLER_DEFAULTS_fix {     \
                    fl2fpt(0),           \
                    fl2fpt(0),           \
                    fl2fpt(0),           \
                    fl2fpt(1.0),         \
                    fl2fpt(0.0),         \
                    fl2fpt(1.0),         \
                    fl2fpt(-1.0),        \
                    fl2fpt(0.0),         \
                    fl2fpt(0.0),         \
                    fl2fpt(0.0),         \
                    fl2fpt(0.0),         \
                    fl2fpt(1.0)          \
                                }

PI_CONTROLLER_fix pi_spd_fix = PI_CONTROLLER_DEFAULTS_fix;
PI_CONTROLLER_fix pi_id_fix = PI_CONTROLLER_DEFAULTS_fix;
PI_CONTROLLER_fix pi_iq_fix = PI_CONTROLLER_DEFAULTS_fix;

uint16_t SpeedLoopPrescaler_fix = 10;

void init_pi_spd_pi_id_pi_iq_fix(void)
{
    // Initialize the PI module for Id
        pi_spd_fix.Kp=fl2fpt(50.0);
        pi_spd_fix.Ki=fl2fpt(cm1_const.Ts*SpeedLoopPrescaler_fix/0.5);
        pi_spd_fix.Umax =fl2fpt(80);
        pi_spd_fix.Umin =fl2fpt(-80);

    // Initialize the PI module for Iq
        pi_id_fix.Kp=fl2fpt(10.0);
        pi_id_fix.Ki=fl2fpt(cm1_const.Ts/0.004);
        pi_id_fix.Umax =fl2fpt(DC_BUS_VOLTAGE);
        pi_id_fix.Umin =fl2fpt(-DC_BUS_VOLTAGE);

    // Initialize the PI module for speed
        pi_iq_fix.Kp=fl2fpt(10.0);
        pi_iq_fix.Ki=fl2fpt(cm1_const.Ts/0.004);
        pi_iq_fix.Umax =fl2fpt(DC_BUS_VOLTAGE);
        pi_iq_fix.Umin =fl2fpt(-DC_BUS_VOLTAGE);



}


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


// *****  "fixed" precision CLARKE_MACRO Macro (simulation
*****************************************************************************
**
```

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


#define  ONEbySQRT3_fix   fl2fpt(0.57735026918963)    /* 1/sqrt(3) */

#define CLARKE_MACRO_fix(v)
\
v.Alpha = fl2fpt(v.As);
\
v.Beta = fpt_mul(fl2fpt(v.As) + fpt_mul(fl2fpt(v.Bs), fl2fpt(v.Bs)),
ONEbySQRT3_fix);

#define CLARKE1_MACRO_fix(v)                                          \
        v.Alpha = fl2fpt(v.As);                                       \
        v.Beta  = fpt_mul(fl2fpt(v.Bs - v.Cs), ONEbySQRT3_fix);


#define CLARKE_DEFAULTS_fix { 0, \
                          0, \
                          0, \
                          fl2fpt(0), \
                          fl2fpt(0), \
                        }


CLARKE_fix clarke1_fix = CLARKE_DEFAULTS_fix;







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


// ***** "fixed" precision PARK_MACRO Macro (simulation)
*****************************************************************************
****

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

#define PARK_MACRO_fix(v)
\
```

```
\
    v.Ds = fpt_mul(v.Alpha, fl2fpt(v.Cosine)) + fpt_mul(v.Beta, fl2fpt(v.Sine));
\
    v.Qs = fpt_mul(v.Beta, fl2fpt(v.Cosine)) - fpt_mul(v.Alpha, fl2fpt(v.Sine));


#define PARK_DEFAULTS_fix {   fl2fpt(0), \
                              fl2fpt(0), \
                              fl2fpt(0), \
                              fl2fpt(0), \
                              fl2fpt(0), \
                              0, \
                              0, \
                              }

PARK_fix park1_fix = PARK_DEFAULTS_fix;


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


#endif
```