LAMPIRAN

```
#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME  IM2
#include <math.h>
//#include "rtwintgt.h"
#include "simstruc.h"

#define U(element) (*uPtrs[element])  /* Pointer to Input Port0 */

static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(S, 7);

    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 4);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    ssSetInputPortOverWritable(S, 0, 1);

    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 10);

    ssSetNumSampleTimes(S, 1);

    /* Take care when specifying exception free code - see
sfuntmpl.doc */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);

}

static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}

#define MDL_INITIALIZE_CONDITIONS
static void mdlInitializeConditions(SimStruct *S)
  {
    real_T  *X0     = ssGetContStates(S);
    int_T   nStates = ssGetNumContStates(S);
    int_T   i;

/* initialize the states to 0.0 */
    for (i=0; i < nStates; i++) {
    X0[i] = 0.0;
    }
  }

static void mdlOutputs(SimStruct *S, int_T tid)
{
    real_T            *Y    = ssGetOutputPortRealSignal(S,0);
    real_T            *X    = ssGetContStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);


real_T   K =0.816497 , L = 0.866025, Ls = 0.355;//
real_T   Lm = 0.347, Lr = 0.355, Np = 4.0, pi2 = 6.28318530718;
real_T   ia,ib;
real_T G   = (1-(Lm*Lm/(Ls*Lr)));

    ia = X[0]*cos(X[6]) - X[1]*sin(X[6]);
    ib = X[0]*sin(X[6]) + X[1]*cos(X[6]);

    Y[0] = K*ia;
    Y[1] = K*( -0.5*ia + L*ib );
    Y[2] = K*( -0.5*ia - L*ib );

    Y[3] = Np*Lm*Lm*(X[1]*X[2]-X[0]*X[3])/Lr;
    Y[4] = X[6];

    Y[5] = X[5];

    Y[6]    = X[4];
    Y[7]    = X[2];
    Y[8]    = Lm*X[2];
    Y[9]    = Lm*X[3];

}

#define MDL_DERIVATIVES
static void mdlDerivatives(SimStruct *S)
  {
    real_T            *dX   = ssGetdX(S);
    real_T            *X    = ssGetContStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);



    real_T      Lm  = 0.347,    Ls  = 0.355,pi2 = 6.28318530718;
    real_T      Lr  = 0.355;
    real_T      Rs  = 0.087;
    real_T      Rr  = 0.228;
    real_T      G   = (1-(Lm*Lm/(Ls*Lr)));
    real_T      K   = 0.816497;
    real_T      L   = 0.866025;
    real_T      B   = 0.1;
    real_T      J   = 0.5;
    real_T      Np  = 4.0;

      real_T       idsn_dot, iqsn_dot;
      real_T       imrd_dot, imrq_dot, wr_dot, theta_r_dot,
      theta_e_dot, vds, vqs,vsd, vsq,we, imra;



   vds  = K*( U(0) - 0.5*U(1) - 0.5*U(2) );
   vqs  = K*L*( U(1) - U(2) );
   vsd = vds*cos(X[6]) + vqs*sin(X[6]);
   vsq = -vds*sin(X[6]) + vqs*cos(X[6]);

   if (fabs(X[2]) <= 0.001 ) { imra = 0.001;}

   else{ imra=X[2];}
   we = (Np*X[4]) + ((Rr*X[1])/(Lr*imra));


   idsn_dot=(-Rs/(G*Ls)-Rr*(1-
G)/(G*Lr))*X[0]+we*X[1]+(Lm*Lm*Rr/(G*Ls*Lr*Lr))*X[2]+(Np*Lm*Lm*X[4
]/(G*Ls*Lr))*X[3]+vsd/(G*Ls);
   iqsn_dot=-we*X[0]+(-Rs/(G*Ls)-Rr*(1-G)/(G*Lr))*X[1]-
(Np*Lm*Lm*X[4]/(G*Ls*Lr))*X[2]+(Lm*Lm*Rr/(G*Ls*Lr*Lr))*X[3]+vsq/(G
*Ls);
   imrd_dot=(Rr/Lr)*(X[0]-X[2])+(we-Np*X[4])*X[3];
   imrq_dot=(Rr/Lr)*(X[1]-X[3])-(we-Np*X[4])*X[2];
   wr_dot=(Np*Lm*Lm*(X[1]*X[2]-X[0]*X[3])/Lr-U(3)-B*X[4])/J;
   theta_r_dot  = X[4];
   theta_e_dot = we;

   while(X[6] > pi2){ X[6] -= pi2;}
   while(X[6] < 0.0) { X[6] += pi2;}

    while(X[5] > pi2){ X[5] -= pi2;}
    while(X[5] < 0.0) { X[5] += pi2;}



    dX[0]   = idsn_dot;
    dX[1]   = iqsn_dot;
    dX[2]   = imrd_dot;
    dX[3]   = imrq_dot;
    dX[4]   = wr_dot;
    dX[5]   = theta_r_dot;
    dX[6]   = theta_e_dot;


  }


static void mdlTerminate(SimStruct *S)  /* Keep this function
empty */
{                   /* since no memory is allocated */
}
```

```
#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a
MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration
function */
#endif
#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME  RFOC2  /* dt = 1e-4 */

#include "tmwtypes.h"
#include "simstruc.h"
#include <math.h>

#define U(element) (*uPtrs[element])  /* Pointer to Input Port0 */


static void mdlInitializeSizes(SimStruct *S)
{

    ssSetNumDiscStates(S, 25);

   if (!ssSetNumInputPorts(S, 1)) return;
   ssSetInputPortWidth(S, 0, 6);
   ssSetInputPortDirectFeedThrough(S, 0, 1);
   ssSetInputPortOverWritable(S, 0, 1);

   if (!ssSetNumOutputPorts(S, 1)) return;
   ssSetOutputPortWidth(S, 0, 15);

    ssSetNumSampleTimes(S, 1);

    /* Take care when specifying exception free code - see
sfuntmpl.doc */
    ssSetOptions(S, (SS_OPTION_EXCEPTION_FREE_CODE |
SS_OPTION_DISCRETE_VALUED_OUTPUT));
}


static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, 1e-4);   /* Sample time in second */
    ssSetOffsetTime(S, 0, 0.0);    /* Starting time in second */
}


#define MDL_INITIALIZE_CONDITIONS
static void mdlInitializeConditions(SimStruct *S)
{
    real_T  *xD0 = ssGetRealDiscStates(S);
    int_T   nDStates = ssGetNumDiscStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    int_T i;

    /* initialize the states to 0.0 */
    for (i=0; i < nDStates; i++) {
    xD0[i] = 0.0;
    }
}

static void mdlOutputs(SimStruct *S, int_T tid)
{
    real_T            *y    = ssGetOutputPortRealSignal(S,0);
    real_T            *xD   = ssGetRealDiscStates(S);

   y[0] = xD[13];//va
   y[1] = xD[14];//vb
   y[2] = xD[15];//vc
   y[3] = xD[7];//imr7
   y[4] = xD[9];//Te
   y[5] = xD[0];//theta_e
   y[6] = xD[2];//idref
   y[7] = xD[1];//id
   y[8] = xD[5];//iqref
   y[9] = xD[4];//iq
   y[10]= xD[16];//vd
   y[11]= xD[17];//vq
   y[12]= xD[18];//we
   y[13]= xD[23];//vs
   y[14]= xD[24];//vdm
}


#define MDL_UPDATE
static void mdlUpdate(SimStruct *S, int_T tid)
{
   real_T   *xD = ssGetRealDiscStates(S);
   InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);

/* constant */
   real_T       pi2 = 6.28318530718;
   real_T       dt  = 1e-4;
   real_T       K   = 0.816497;
   real_T       L   = 0.866025;

/* limiter constant */
   real_T       Imrmax  = 3.00;
   real_T       Imrmin  = 0.1;
   real_T       MMAX    = 0.9;
   real_T       Ismax2  = MMAX*MMAX*43.2964;
   real_T       VDC     = 311.0;
   real_T       vinv    = 0.5*VDC*MMAX;

/* motor parameters */
   real_T       p   = 4.0;
   real_T       Lm  = 0.347;
   real_T       Lr  = 0.355;
   real_T       Ls  = 0.355;
   real_T       Rs  = 0.0087;
   real_T       Rr  = 0.228;
   real_T       invTr   = (Rr/Lr);
   real_T       sigma   = (1-(Lm * Lm)/(Ls*Lr));
```

```
/* current controller gains */
   real_T       Ti    = 0.005;//0.01;//0.0025;//
   real_T       Kidp  = Ls/Ti;
   real_T       Kidi  = Rs/Ti;
   real_T       Kiqp  = Ls/Ti;
   real_T       Kiqi  = Rs/Ti;
   real_T       Kimrp = 0.00099;//0.4;//0.00125;//0.001125;//
   real_T       Kimri = 0.001;
   real_T       Kvqp  = 0.0003;
   real_T       Kvqi  = 0.05;

/* variables */
      real_T       ialfa, ibeta, id, iq;
      real_T       theta_e, imr, imr_prv, imra, Te;
      real_T       we, eid, idref, xid, xid_prv, ud, xiqrepnew,
      xiqrepprv, xevabnew, xevabprv;
      real_T       eiq, iqref, xiq, xiq_prv, uq, Iqmax;
      real_T       vflt, valfa, vbeta, evab, vd, vq, va, vb, vc,
      vdm, evd, Iqrep, temp, idrep, vs;
      real_T       vasat, vbsat, vcsat, verror;

/*  S T A R T   A L G O R I T H M  */
/*======read states data=======*/
theta_e = xD[0];
id  = xD[1];
idref   = xD[2];
xid_prv = xD[3];
iq  = xD[4];
iqref   = xD[5];
xiq_prv = xD[6];
imr = xD[7];
imr_prv = xD[8];
Te  = xD[9];
vflt    = xD[10];
valfa   = xD[11];
vbeta   = xD[12];
va  = xD[13];
vb  = xD[14];
vc  = xD[15];
vd  = xD[16];
vq  = xD[17];
we  = xD[18];
xevabprv = xD[20];
xiqrepprv = xD[19];
xevabprv = xD[20];
Iqmax = xD[21];
Iqrep = xD[22];
vs = xD[23];
vdm = xD[24];

/* UVW to d-q axis convert */
   ialfa = (U(0)-0.5*U(1)-0.5*U(2))*K;
   ibeta = (L*(U(1)-U(2)))*K;

   id = ialfa*cos(theta_e) + ibeta*sin(theta_e);
   iq = -ialfa*sin(theta_e) + ibeta*cos(theta_e);
```

```
/* rotor flux equation */
   imr = imr_prv + dt*(Rr*(id - imr_prv))/Lr;

/* torque equation */
   Te = p*(1-sigma)*Ls*iq*imr;

/* Synchronous position equation */
   if (fabs(imr) <= 0.001) { imra = 0.001;}
   else { imra = imr; }
   we = p*U(5) + (Rr*iq)/(imra*Lr);
   theta_e = theta_e + dt*we;
    {while(theta_e >= pi2) { theta_e -= pi2;}}
    {while(theta_e < 0.0) { theta_e += pi2;}}

/* d-axis current controller */
   eid= idref-id;
   xid= xid_prv+dt*eid;
   ud = Kidp * eid + Kidi * xid;//

/* q-axis current controller */
   eiq= iqref-iq;
   xiq=xiq_prv+dt*eiq;
   uq = Kiqp * eiq + Kiqi * xiq;

/*====== 1/Wr Method for idref====================*/
//      if(fabs(U(5)) > 77.2136){
//      idref=Imrmax*77.2136/fabs(U(5));}
//      else if(fabs(U(5)) > 350.0){
//      idref=Imrmax*77.2136/(U(5)*U(5));}
//      else {idref = Imrmax;}


/******************************************************/
//field weakening generator
/******************************************************/

/*** voltage controller ***/

   vflt=(1-dt*100.0)* vflt+dt*100.0*(vd*vd+vq*vq);
   evab = U(3)*U(3) - (vflt);
   xevabnew = xevabprv + dt * (evab);
   idref = Kimrp * evab + Kimri * xevabnew;

/*====== Idref limiter ====================*/
   if(idref >= Imrmax ) { idref = Imrmax;xevabnew = xevabprv;}
   if(idref <= Imrmin ) { idref = Imrmin;xevabnew = xevabprv;}

Iqmax = sqrt(Ismax2 - idref*idref);
if (Iqmax < Iqrep) {Iqrep = Iqmax;}
if (Iqmax > -Iqrep) {Iqrep = -Iqmax;}


/* torque reference */
   iqref = U(4);

   if(iqref > Iqmax) {iqref = Iqmax;}
   if(iqref < -Iqmax) {iqref = -Iqmax;}




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

/* decoupling equations */
   vd=ud-we*Ls*sigma*iq;//+Ls*(1-sigma)*(imr-imr_prv);
   vq=uq+we*Ls*sigma*id+we*Ls*(1-sigma)*imr;

   valfa =vd*cos(theta_e)-vq*sin(theta_e);
   vbeta =vd*sin(theta_e)+vq*cos(theta_e);

   va = valfa*K;
   vb = (-0.5*valfa+L*vbeta)*K;
   vc = (-0.5*valfa-L*vbeta)*K;

   vasat = va;
   vbsat = vb;
   vcsat = vc;
/*=========voltage limiter=========*/
if(va >= vinv){ vasat = vinv;}// xid=xid_prv; xiq=xiq_prv;}
if(va <= -vinv){ vasat = -vinv;}// xid=xid_prv; xiq=xiq_prv;}
if(vb >= vinv){ vbsat = vinv;}// xid=xid_prv; xiq=xiq_prv;}
if(vb <= -vinv){ vbsat = -vinv;}// xid=xid_prv; xiq=xiq_prv;}
if(vc >= vinv){ vcsat = vinv;}// xid=xid_prv; xiq=xiq_prv;}
if(vc <= -vinv){ vcsat = -vinv;}// xid=xid_prv; xiq=xiq_prv;}


verror=vasat+vbsat+vcsat;
va=vasat;
vb=vbsat;
vc=vcsat;

   valfa = (va-0.5*vb-0.5*vc)*K;
   vbeta = (vb-vc)*K*L;

   vd = valfa*cos(theta_e)+vbeta*sin(theta_e);
   vq = -valfa*sin(theta_e)+vbeta*cos(theta_e);

   vs = sqrt((vd*vd) + (vq*vq));
   vdm = sqrt(vd*vd);


/*======update states data=======*/
xD[0] = theta_e;
xD[1] = id;
xD[2] = idref;
xD[3] = xid;
xD[4] = iq;
xD[5] = iqref;
xD[6] = xiq;
xD[7] = imr;
xD[8] = imr;
```

```
xD[9] = Te;
xD[10] = vflt;
xD[11] = valfa;
xD[12] = vbeta;
xD[13] = va;
xD[14] = vb;
xD[15] = vc;
xD[16] = vd;
xD[17] = vq;
xD[18] = we;
xD[20] = xevabnew;

xD[19] = xiqrepnew;
xD[21] = Iqmax;
xD[22] = Iqrep;
xD[23] = vs;
xD[24] = vdm;
}

static void mdlTerminate(SimStruct *S)
{}

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a
MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration
function */
#endif
```

```
#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME  SPD_CTL
#include <math.h>
/*#include "rtwintgt.h"*/
#include "simstruc.h"

#define U(element) (*uPtrs[element])

static void mdlInitializeSizes(SimStruct *S){
  ssSetNumDiscStates(S, 3);
  if (!ssSetNumInputPorts(S, 1)) return;
  ssSetInputPortWidth(S, 0, 4);
  ssSetInputPortDirectFeedThrough(S, 0, 1);
  ssSetInputPortOverWritable(S, 0, 1);
  if (!ssSetNumOutputPorts(S, 1)) return;
  ssSetOutputPortWidth(S, 0, 1);
  ssSetNumSampleTimes(S, 1);

  ssSetOptions(S, (SS_OPTION_EXCEPTION_FREE_CODE
          | SS_OPTION_DISCRETE_VALUED_OUTPUT));}

static void mdlInitializeSampleTimes(SimStruct *S){
  ssSetSampleTime(S, 0, 1e-3);
  ssSetOffsetTime(S, 0, 0.0);}

#define MDL_INITIALIZE_CONDITIONS
static void mdlInitializeConditions(SimStruct *S){
  real_T   *X0 = ssGetRealDiscStates(S);
  int_T    nXStates = ssGetNumDiscStates(S);
  InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
  int_T i;

/* initialize the states to 0.0 */
  for (i=0; i < nXStates; i++) {
    X0[i] = 0.0; } }

static void mdlOutputs(SimStruct *S, int_T tid){
  real_T   *Y = ssGetOutputPortRealSignal(S,0);
  real_T   *X = ssGetRealDiscStates(S);
  real_T   Tref;

  Tref = X[1];
  Y[0] = Tref; }

#define MDL_UPDATE
static void mdlUpdate(SimStruct *S, int_T tid) {
  real_T    *X = ssGetRealDiscStates(S);
  InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);

  real_T    dt = 1e-3;
  real_T    Tmax = 7.0;
  real_T    Tref, Wref, Wr, xWr_new, xWr_prv, Kspi, Kspp;

      Kspp = U(0); Kspi = U(1); Wref = U(2); Wr = U(3); xWr_prv =
X[0];
```

```
  xWr_new = xWr_prv + dt * ( Wref - Wr ) ;
  Tref =   Kspi * xWr_new + Kspp * ( Wref - Wr );
  if( Tref >= Tmax ){Tref = Tmax; xWr_new = xWr_prv;}
  if( Tref <= -Tmax ){Tref = -Tmax; xWr_new = xWr_prv;}

  X[0] = xWr_new;  X[1] = Tref;}


static void mdlTerminate(SimStruct *S)  /* Keep this function
empty */
{ }                 /* since no memory is allocated */

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a
MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration
function */
#endif
```

```
#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME  MRAS1
#include "tmwtypes.h"
#include "simstruc.h"
#include <math.h>
#define U(element) (*uPtrs[element])  /* Pointer to Input Port0 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumDiscStates(S, 10);
   if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 5);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    ssSetInputPortOverWritable(S, 0, 1);
    if (!ssSetNumOutputPorts(S, 1)) return;
    //ssSetOutputPortWidth(S, 0, 5);
    ssSetOutputPortWidth(S, 0, 3);
    ssSetNumSampleTimes(S, 1);
    /* Take care when specifying exception free code - see
sfuntmpl.doc */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, 1e-4);   /* Sample time in second */
    ssSetOffsetTime(S, 0, 0.0);    /* Starting time in second */
}
static void mdlInitializeConditions(SimStruct *S)
{
    real_T  *xD0 = ssGetRealDiscStates(S);
    int_T   nDStates = ssGetNumDiscStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    int_T i;
    /* initialize the states to 0.0 */
    for (i=0; i < nDStates; i++)
    {
    xD0[i] = 0.0; }
}
static void mdlOutputs(SimStruct *S, int_T tid)
{
    real_T            *y    = ssGetOutputPortRealSignal(S,0);
    real_T            *xD   = ssGetDiscStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    y[0] = xD[0];//id_est
   y[1] = xD[1];//iq_est
   y[2] = xD[2];//fluks_d_est
   y[3] = xD[3];//fluks_q_est
   y[4] = xD[9];//wr_est


       }
#define MDL_UPDATE
static void mdlUpdate(SimStruct *S, int_T tid)
{
   real_T   *xD = ssGetRealDiscStates(S);
   InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);

/* motor parameters */
   real_T       p   = 2.0;
   real_T       Lm  = 0.2279;
   real_T       Lr  = 0.2349;
   real_T       Ls  = 0.2349;
   real_T       Rs  = 2.76;
   real_T       Rr  = 2.90;
   real_T       Tr  = (Lr/Rr);
   real_T       pi2 = 6.28318530718;
   real_T       sigma   = (1-(Lm * Lm)/(Ls*Lr));
   real_T       a0 = Lr/Lm;
   real_T       a1 = Lm/Tr;

/*constant*/
   real_T       dt  = 1e-4;
   real_T       Kp  = 10;
   real_T       Ki  = 300;

/* variables */

   real_T   fluks_d_ref,fluks_q_ref, fluks_d_adj,fluks_q_adj;
   real_T   E ,Eintgrl,Eintgrl_prv;
   real_T   wr_est;
   real_T   fluks_d_ref_prv,fluks_q_ref_prv;
   real_T   fluks_d_adj_prv,fluks_q_adj_prv;
   real_T   id,iq,vd,vq,we;
   real_T   ed,eq;
/* input */

   id = U(0);
   iq = U(1);
   vd = U(2);
   vq = U(3);
   we = U(4);

  /*  S T A R T   A L G O R I T H M  */
fluks_d_ref       = xD[0];
fluks_q_ref       = xD[1];
fluks_d_adj       = xD[2];
fluks_q_adj       = xD[3];
fluks_d_ref_prv   = xD[4];
fluks_q_ref_prv   = xD[5];
fluks_d_adj_prv   = xD[6];
fluks_q_adj_prv   = xD[7];
Eintgrl_prv       = xD[8];
wr_est            = xD[9];

fluks_d_ref   = (fluks_d_ref_prv + dt*(a0*vd -
(Rs+(sigma*Ls*p))*id));           /* reference sumbu d */
fluks_q_ref   = (fluks_q_ref_prv + dt*(a0*vq -
(Rs+(sigma*Ls*p))*iq));          /* reference sumbu q */
fluks_d_adj   = (fluks_d_adj_prv + dt*(((-1/Tr)*fluks_d_ref_prv) -
(U(4)*fluks_q_ref_prv) + a1*id));   /* adaptif d */
fluks_q_adj   = (fluks_q_adj_prv + dt*((U(4)*fluks_d_ref_prv) -
((-1/Tr)*fluks_q_ref_prv) + a1*iq));   /* adaptif q */

/* wr estimasi */
ed = fluks_d_ref - fluks_d_adj;
eq = fluks_q_ref - fluks_q_adj;
E = ((eq*fluks_d_adj)-(ed*fluks_q_adj));
```

```
Eintgrl = Eintgrl_prv + dt*E;
wr_est= Kp*E + Ki*Eintgrl;





       xD[0]= fluks_d_ref;
       xD[1]= fluks_q_ref;
       xD[2]= fluks_d_adj;
       xD[3]= fluks_q_adj;
       xD[4]= fluks_d_ref;
       xD[5]= fluks_q_ref;
       xD[6]= fluks_d_adj;
       xD[7]= fluks_q_adj ;
       xD[8]= Eintgrl;
       xD[9]= wr_est;

}
static void mdlTerminate(SimStruct *S)
{}
#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a
MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration
function */
#endif
```