#include "State.h"

#ifndef rtmIsMajorTimeStep

# define rtmIsMajorTimeStep(rtm) (((rtm)->Timing.simTimeStep) == MAJOR\_TIME\_STEP)

#endif

#ifndef rtmIsMinorTimeStep

# define rtmIsMinorTimeStep(rtm) (((rtm)->Timing.simTimeStep) == MINOR\_TIME\_STEP)

#endif

#ifndef rtmSetTPtr

# define rtmSetTPtr(rtm, val) ((rtm)->Timing.t = (val))

#endif

#ifndef UCHAR\_MAX

#include <limits.h>

#endif

#if ( UCHAR\_MAX != (0xFFU) ) || ( SCHAR\_MAX != (0x7F) )

#error Code was generated for compiler with different sized uchar/char. \

Consider adjusting Test hardware word size settings on the \

Hardware Implementation pane to match your compiler word sizes as \

defined in limits.h of the compiler. Alternatively, you can \

select the Test hardware is the same as production hardware option and \

select the Enable portable word sizes option on the Code Generation > \

Verification pane for ERT based targets, which will disable the \

preprocessor word size checks.

#endif

#if ( USHRT\_MAX != (0xFFFFU) ) || ( SHRT\_MAX != (0x7FFF) )

#error Code was generated for compiler with different sized ushort/short. \

Consider adjusting Test hardware word size settings on the \

Hardware Implementation pane to match your compiler word sizes as \

defined in limits.h of the compiler. Alternatively, you can \

select the Test hardware is the same as production hardware option and \

select the Enable portable word sizes option on the Code Generation > \

Verification pane for ERT based targets, which will disable the \

preprocessor word size checks.

#endif

#if ( UINT\_MAX != (0xFFFFFFFFU) ) || ( INT\_MAX != (0x7FFFFFFF) )

#error Code was generated for compiler with different sized uint/int. \

Consider adjusting Test hardware word size settings on the \

Hardware Implementation pane to match your compiler word sizes as \

defined in limits.h of the compiler. Alternatively, you can \

select the Test hardware is the same as production hardware option and \

select the Enable portable word sizes option on the Code Generation > \

Verification pane for ERT based targets, which will disable the \

preprocessor word size checks.

#endif

#if ( ULONG\_MAX != (0xFFFFFFFFU) ) || ( LONG\_MAX != (0x7FFFFFFF) )

#error Code was generated for compiler with different sized ulong/long. \

Consider adjusting Test hardware word size settings on the \

Hardware Implementation pane to match your compiler word sizes as \

defined in limits.h of the compiler. Alternatively, you can \

select the Test hardware is the same as production hardware option and \

select the Enable portable word sizes option on the Code Generation > \

Verification pane for ERT based targets, which will disable the \

preprocessor word size checks.

#endif

X rtX;

DW rtDW;

ExtU rtU;

ExtY rtY;

RT\_MODEL rtM\_;

RT\_MODEL \*const rtM = &rtM\_;

extern void State\_derivatives(void);

static void rt\_ertODEUpdateContinuousStates(RTWSolverInfo \*si )

{

static const real\_T rt\_ODE3\_A[3] = {

1.0/2.0, 3.0/4.0, 1.0

};

static const real\_T rt\_ODE3\_B[3][3] = {

{ 1.0/2.0, 0.0, 0.0 },

{ 0.0, 3.0/4.0, 0.0 },

{ 2.0/9.0, 1.0/3.0, 4.0/9.0 }

};

time\_T t = rtsiGetT(si);

time\_T tnew = rtsiGetSolverStopTime(si);

time\_T h = rtsiGetStepSize(si);

real\_T \*x = rtsiGetContStates(si);

ODE3\_IntgData \*id = (ODE3\_IntgData \*)rtsiGetSolverData(si);

real\_T \*y = id->y;

real\_T \*f0 = id->f[0];

real\_T \*f1 = id->f[1];

real\_T \*f2 = id->f[2];

real\_T hB[3];

int\_T i;

int\_T nXc = 2;

rtsiSetSimTimeStep(si,MINOR\_TIME\_STEP);

(void) memcpy(y, x,

(uint\_T)nXc\*sizeof(real\_T));

rtsiSetdX(si, f0);

State\_derivatives();

hB[0] = h \* rt\_ODE3\_B[0][0];

for (i = 0; i < nXc; i++) {

x[i] = y[i] + (f0[i]\*hB[0]);

}

rtsiSetT(si, t + h\*rt\_ODE3\_A[0]);

rtsiSetdX(si, f1);

State\_step();

State\_derivatives();

for (i = 0; i <= 1; i++) {

hB[i] = h \* rt\_ODE3\_B[1][i];

}

for (i = 0; i < nXc; i++) {

x[i] = y[i] + (f0[i]\*hB[0] + f1[i]\*hB[1]);

}

rtsiSetT(si, t + h\*rt\_ODE3\_A[1]);

rtsiSetdX(si, f2);

State\_step();

State\_derivatives();

for (i = 0; i <= 2; i++) {

hB[i] = h \* rt\_ODE3\_B[2][i];

}

for (i = 0; i < nXc; i++) {

x[i] = y[i] + (f0[i]\*hB[0] + f1[i]\*hB[1] + f2[i]\*hB[2]);

}

rtsiSetT(si, tnew);

rtsiSetSimTimeStep(si,MAJOR\_TIME\_STEP);

}

void State\_step(void)

{

if (rtmIsMajorTimeStep(rtM)) {

rtsiSetSolverStopTime(&rtM->solverInfo,((rtM->Timing.clockTick0+1)\*

rtM->Timing.stepSize0));

}

if (rtmIsMinorTimeStep(rtM)) {

rtM->Timing.t[0] = rtsiGetT(&rtM->solverInfo);

}

rtY.Out1[0] = (int8\_T)floor(rtX.ReplicaOfSource\_CSTATE[0] \* 64.0);

rtY.Out1[1] = (int8\_T)floor(rtX.ReplicaOfSource\_CSTATE[1] \* 64.0);

rtDW.DTC\_input\_1 = (real\_T)rtU.In1 \* 0.00048828125;

if (rtmIsMajorTimeStep(rtM)) {

rt\_ertODEUpdateContinuousStates(&rtM->solverInfo);

++rtM->Timing.clockTick0;

rtM->Timing.t[0] = rtsiGetSolverStopTime(&rtM->solverInfo);

}

}

void State\_derivatives(void)

{

XDot \*\_rtXdot;

\_rtXdot = ((XDot \*) rtM->derivs);

\_rtXdot->ReplicaOfSource\_CSTATE[0] = 0.0;

\_rtXdot->ReplicaOfSource\_CSTATE[1] = 0.0;

\_rtXdot->ReplicaOfSource\_CSTATE[0] += rtX.ReplicaOfSource\_CSTATE[1];

\_rtXdot->ReplicaOfSource\_CSTATE[0] += 0.4 \* rtDW.DTC\_input\_1;

\_rtXdot->ReplicaOfSource\_CSTATE[1] += rtDW.DTC\_input\_1;

}

void State\_initialize(void)

{

{

rtsiSetSimTimeStepPtr(&rtM->solverInfo, &rtM->Timing.simTimeStep);

rtsiSetTPtr(&rtM->solverInfo, &rtmGetTPtr(rtM));

rtsiSetStepSizePtr(&rtM->solverInfo, &rtM->Timing.stepSize0);

rtsiSetdXPtr(&rtM->solverInfo, &rtM->derivs);

rtsiSetContStatesPtr(&rtM->solverInfo, (real\_T \*\*) &rtM->contStates);

rtsiSetNumContStatesPtr(&rtM->solverInfo, &rtM->Sizes.numContStates);

rtsiSetNumPeriodicContStatesPtr(&rtM->solverInfo,

&rtM->Sizes.numPeriodicContStates);

rtsiSetPeriodicContStateIndicesPtr(&rtM->solverInfo,

&rtM->periodicContStateIndices);

rtsiSetPeriodicContStateRangesPtr(&rtM->solverInfo,

&rtM->periodicContStateRanges);

rtsiSetErrorStatusPtr(&rtM->solverInfo, (&rtmGetErrorStatus(rtM)));

rtsiSetRTModelPtr(&rtM->solverInfo, rtM);

}

rtsiSetSimTimeStep(&rtM->solverInfo, MAJOR\_TIME\_STEP);

rtM->intgData.y = rtM->odeY;

rtM->intgData.f[0] = rtM->odeF[0];

rtM->intgData.f[1] = rtM->odeF[1];

rtM->intgData.f[2] = rtM->odeF[2];

rtM->contStates = ((X \*) &rtX);

rtsiSetSolverData(&rtM->solverInfo, (void \*)&rtM->intgData);

rtsiSetSolverName(&rtM->solverInfo,"ode3");

rtmSetTPtr(rtM, &rtM->Timing.tArray[0]);

rtM->Timing.stepSize0 = 2.0;

rtX.ReplicaOfSource\_CSTATE[0] = 0.0;

rtX.ReplicaOfSource\_CSTATE[1] = 0.0;

}