**#include "Subsystem.h"**

**#include "rtwtypes.h"**

**#include <math.h>**

**#include <float.h>**

**#include <stddef.h>**

**#define** NumBitsPerChar 8U

**#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** PORTABLE\_WORDSIZES

**#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 != (0xFFFFU) ) || ( INT\_MAX != (0x7FFF) )

**#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 != (0xFFFFFFFFUL) ) || ( LONG\_MAX != (0x7FFFFFFFL) )

**#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**

/\* Skipping ulong\_long/long\_long check: insufficient preprocessor integer range. \*/

**#endif**  /\* PORTABLE\_WORDSIZES \*/

/\* Block signals and states (default storage) \*/

DW rtDW;

/\* External inputs (root inport signals with default storage) \*/

ExtU rtU;

/\* External outputs (root outports fed by signals with default storage) \*/

ExtY rtY;

/\* Real-time model \*/

static RT\_MODEL rtM\_;

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

extern real\_T rt\_remd\_snf(real\_T u0, real\_T u1);

static real\_T look1\_pbinlx(real\_T u0, const real\_T bp0[], const real\_T table[],

uint32\_T prevIndex[], uint32\_T maxIndex);

static real\_T rtGetNaN(void);

static real32\_T rtGetNaNF(void);

extern real\_T rtInf;

extern real\_T rtMinusInf;

extern real\_T rtNaN;

extern real32\_T rtInfF;

extern real32\_T rtMinusInfF;

extern real32\_T rtNaNF;

static void rt\_InitInfAndNaN(size\_t realSize);

static boolean\_T rtIsInf(real\_T value);

static boolean\_T rtIsInfF(real32\_T value);

static boolean\_T rtIsNaN(real\_T value);

static boolean\_T rtIsNaNF(real32\_T value);

typedef struct **{**

struct **{**

uint32\_T wordH;

uint32\_T wordL;

**}** words;

**}** BigEndianIEEEDouble;

typedef struct **{**

struct **{**

uint32\_T wordL;

uint32\_T wordH;

**}** words;

**}** LittleEndianIEEEDouble;

typedef struct **{**

union **{**

real32\_T wordLreal;

uint32\_T wordLuint;

**}** wordL;

**}** IEEESingle;

real\_T rtInf;

real\_T rtMinusInf;

real\_T rtNaN;

real32\_T rtInfF;

real32\_T rtMinusInfF;

real32\_T rtNaNF;

static real\_T rtGetInf(void);

static real32\_T rtGetInfF(void);

static real\_T rtGetMinusInf(void);

static real32\_T rtGetMinusInfF(void);

/\*

\* Initialize rtNaN needed by the generated code.

\* NaN is initialized as non-signaling. Assumes IEEE.

\*/

static real\_T rtGetNaN(void)

**{**

size\_t bitsPerReal = sizeof(real\_T) \* (NumBitsPerChar);

real\_T nan = 0.0;

if (bitsPerReal == 32U) **{**

nan = rtGetNaNF();

**}** else **{**

uint16\_T one = 1U;

enum **{**

LittleEndian,

BigEndian

**}** machByteOrder = (\*((uint8\_T \*) &one) == 1U) ? LittleEndian : BigEndian;

switch (machByteOrder) **{**

case LittleEndian:

**{**

union **{**

LittleEndianIEEEDouble bitVal;

real\_T fltVal;

**}** tmpVal;

tmpVal.bitVal.words.wordH = 0xFFF80000U;

tmpVal.bitVal.words.wordL = 0x00000000U;

nan = tmpVal.fltVal;

\* RelationalOperator: '<S2>/Compare'

\* RelationalOperator: '<S3>/Compare'

\* Switch: '<S1>/Switch1'

\*/

rtY.Out5 = ((((int16\_T)rtU.y == 6) || ((int16\_T)rtU.y == 2)) &&

rtb\_RelationalOperator4);

/\* Outport: '<Root>/Out6' incorporates:

\* Constant: '<S1>/Constant1'

\* Switch: '<S1>/Switch2'

\*/

rtY.Out6 = 0.0;

/\* Outport: '<Root>/Out3' incorporates:

\* Inport: '<Root>/u1'

\* Logic: '<S1>/Logical Operator14'

\* Logic: '<S1>/Logical Operator2'

\* RelationalOperator: '<S8>/Compare'

\* RelationalOperator: '<S9>/Compare'

\* Switch: '<S1>/Switch3'

\*/

rtY.Out3 = ((((int16\_T)rtU.y == 3) || ((int16\_T)rtU.y == 1)) &&

rtb\_RelationalOperator4);

/\* Outport: '<Root>/Out1' incorporates:

\* Constant: '<S1>/Constant1'

\* Switch: '<S1>/Switch4'

\*/

rtY.Out1 = 0.0;

/\* Outport: '<Root>/Out4' incorporates:

\* Inport: '<Root>/u1'

\* Logic: '<S1>/Logical Operator12'

\* Logic: '<S1>/Logical Operator4'

\* RelationalOperator: '<S12>/Compare'

\* RelationalOperator: '<S13>/Compare'

\* Switch: '<S1>/Switch5'

\*/

rtY.Out4 = ((((int16\_T)rtU.y == 5) || ((int16\_T)rtU.y == 4)) &&

rtb\_RelationalOperator4);

**}** else **{**

/\* Outport: '<Root>/Out2' incorporates:

\* Inport: '<Root>/u1'

\* Logic: '<S1>/Logical Operator'

\* Logic: '<S1>/Logical Operator6'

\* RelationalOperator: '<S2>/Compare'

\* RelationalOperator: '<S3>/Compare'

\* Switch: '<S1>/Switch'

\*/

rtY.Out2 = (real\_T)(rtb\_RelationalOperator && (((int16\_T)rtU.y == 6) ||

((int16\_T)rtU.y == 2)));

/\* Outport: '<Root>/Out5' incorporates:

\* Inport: '<Root>/u1'

\* Logic: '<S1>/Logical Operator1'

\* Logic: '<S1>/Logical Operator7'

\* RelationalOperator: '<S6>/Compare'

\* RelationalOperator: '<S7>/Compare'

\* Switch: '<S1>/Switch1'

\*/

rtY.Out5 = ((((int16\_T)rtU.y == 5) || ((int16\_T)rtU.y == 1)) &&

rtb\_RelationalOperator);

/\* Outport: '<Root>/Out6' incorporates:

\* Inport: '<Root>/u1'

\* Logic: '<S1>/Logical Operator2'

\* Logic: '<S1>/Logical Operator8'

\* RelationalOperator: '<S8>/Compare'

\* RelationalOperator: '<S9>/Compare'

\* Switch: '<S1>/Switch2'

\*/

rtY.Out6 = (real\_T)((((int16\_T)rtU.y == 3) || ((int16\_T)rtU.y == 1)) &&

rtb\_RelationalOperator);

/\* Outport: '<Root>/Out3' incorporates:

\* Inport: '<Root>/u1'

\* Logic: '<S1>/Logical Operator3'

\* Logic: '<S1>/Logical Operator9'

\* RelationalOperator: '<S10>/Compare'

\* RelationalOperator: '<S11>/Compare'

\* Switch: '<S1>/Switch3'

\*/

rtY.Out3 = ((((int16\_T)rtU.y == 4) || ((int16\_T)rtU.y == 6)) &&

rtb\_RelationalOperator);

/\* Outport: '<Root>/Out1' incorporates:

\* Inport: '<Root>/u1'

\* Logic: '<S1>/Logical Operator10'

\* Logic: '<S1>/Logical Operator4'

\* RelationalOperator: '<S12>/Compare'

\* RelationalOperator: '<S13>/Compare'

\* Switch: '<S1>/Switch4'

\*/

rtY.Out1 = (real\_T)(rtb\_RelationalOperator && (((int16\_T)rtU.y == 5) ||

((int16\_T)rtU.y == 4)));

/\* Outport: '<Root>/Out4' incorporates:

\* Inport: '<Root>/u1'

\* Logic: '<S1>/Logical Operator11'

\* Logic: '<S1>/Logical Operator5'

\* RelationalOperator: '<S4>/Compare'

\* RelationalOperator: '<S5>/Compare'

\* Switch: '<S1>/Switch5'

\*/

rtY.Out4 = (rtb\_RelationalOperator && (((int16\_T)rtU.y == 2) || ((int16\_T)

rtU.y == 3)));

**}**

/\* End of Switch: '<S1>/Switch' \*/

/\* Update absolute time for base rate \*/

/\* The "clockTick0" counts the number of times the code of this task has

\* been executed. The absolute time is the multiplication of "clockTick0"

\* and "Timing.stepSize0". Size of "clockTick0" ensures timer will not

\* overflow during the application lifespan selected.

\*/

rtM->Timing.t[0] =

((time\_T)(++rtM->Timing.clockTick0)) \* rtM->Timing.stepSize0;

**{**

/\* Update absolute timer for sample time: [6.25E-5s, 0.0s] \*/

/\* The "clockTick1" counts the number of times the code of this task has

\* been executed. The resolution of this integer timer is 6.25E-5, which is the step size

\* of the task. Size of "clockTick1" ensures timer will not overflow during the

\* application lifespan selected.

\*/

rtM->Timing.clockTick1++;

**}**

**}**

/\* Model initialize function \*/

void Subsystem\_initialize(void)

**{**

/\* Registration code \*/

/\* initialize non-finites \*/

rt\_InitInfAndNaN(sizeof(real\_T));

**{**

/\* Setup solver object \*/

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

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

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

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

rtsiSetRTModelPtr(&rtM->solverInfo, rtM);

**}**

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

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

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

rtM->Timing.stepSize0 = 6.25E-5;

**}**