Appendix A

Diagrams of ATM Modeling

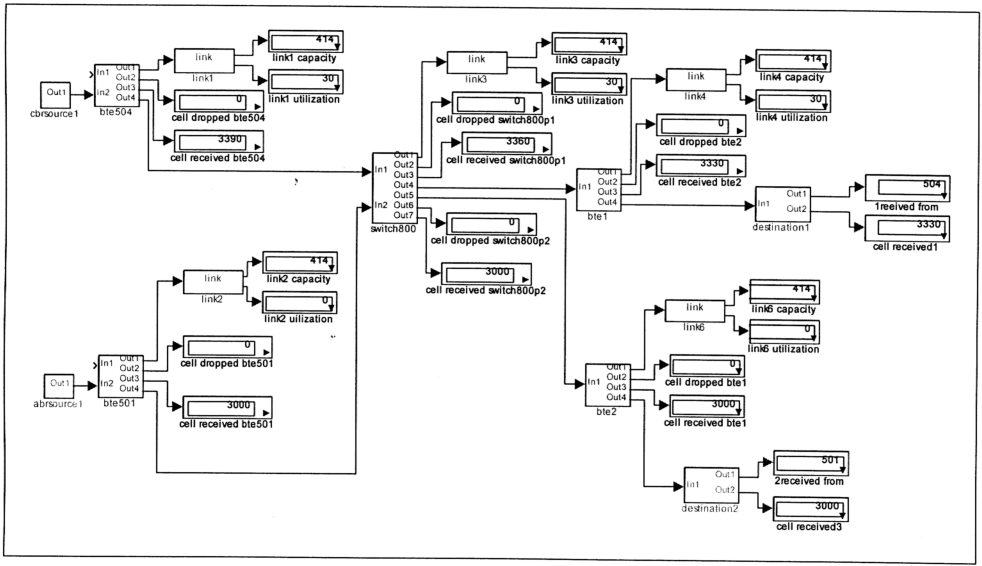

Figure A.1 Single Hop Topology: Port 1 to Port 1, Port 2 to Port 2

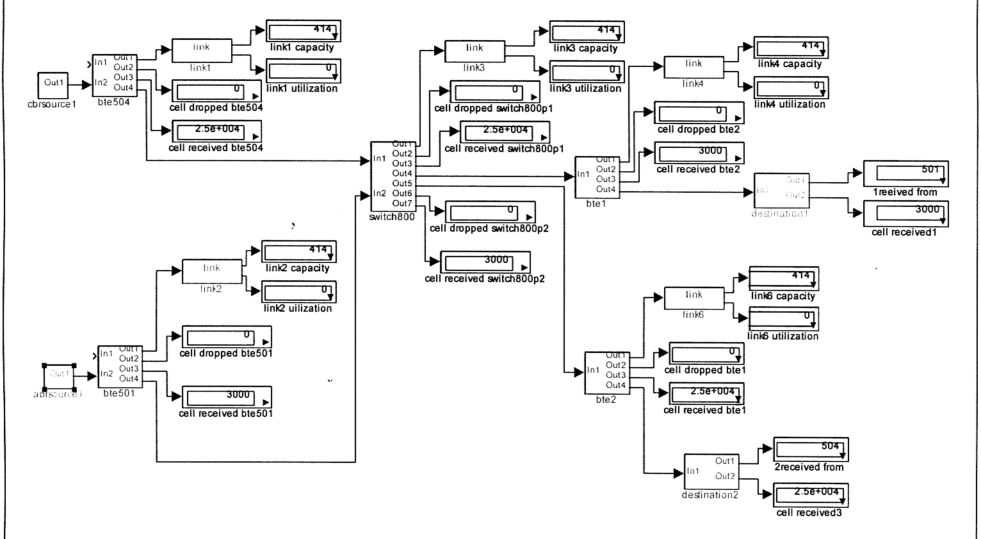

Figure A.2 Single Hop Topology: Port 1 to Port 2, Port 2 to Port 1

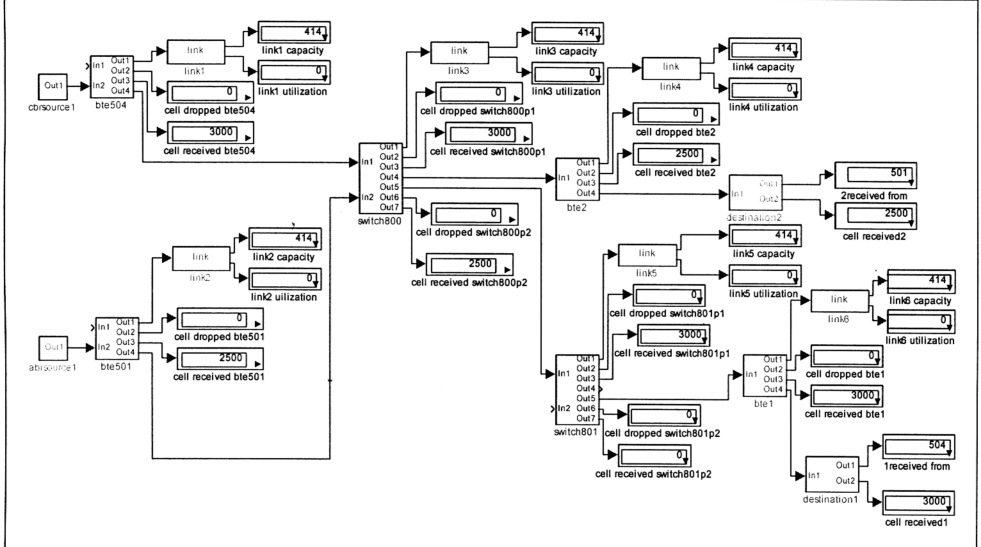

Figure A.3 Multi Hop Topology: 2 switches with one connection

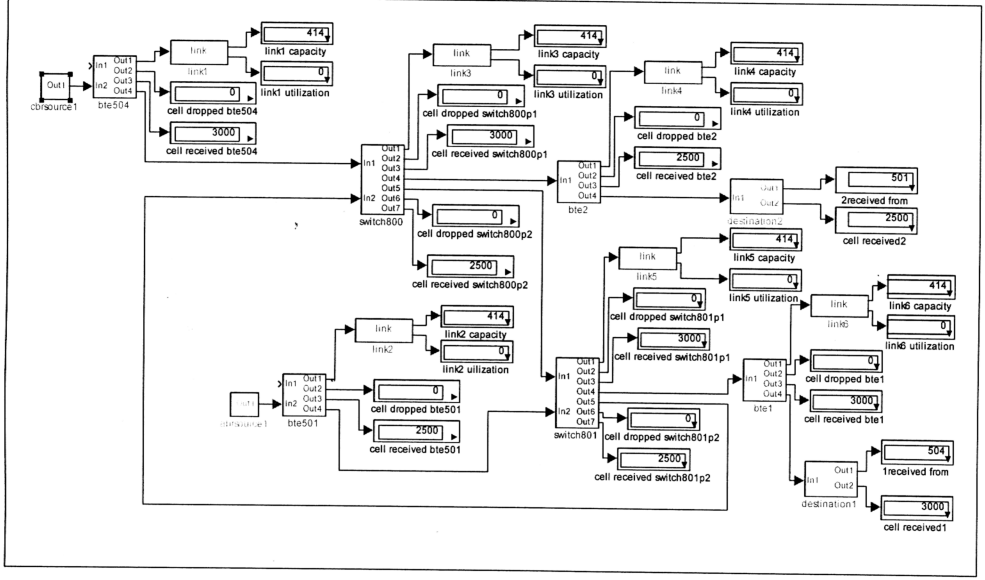

Figure A.4 Multi Hop Topology: 2 switches with connection to each other

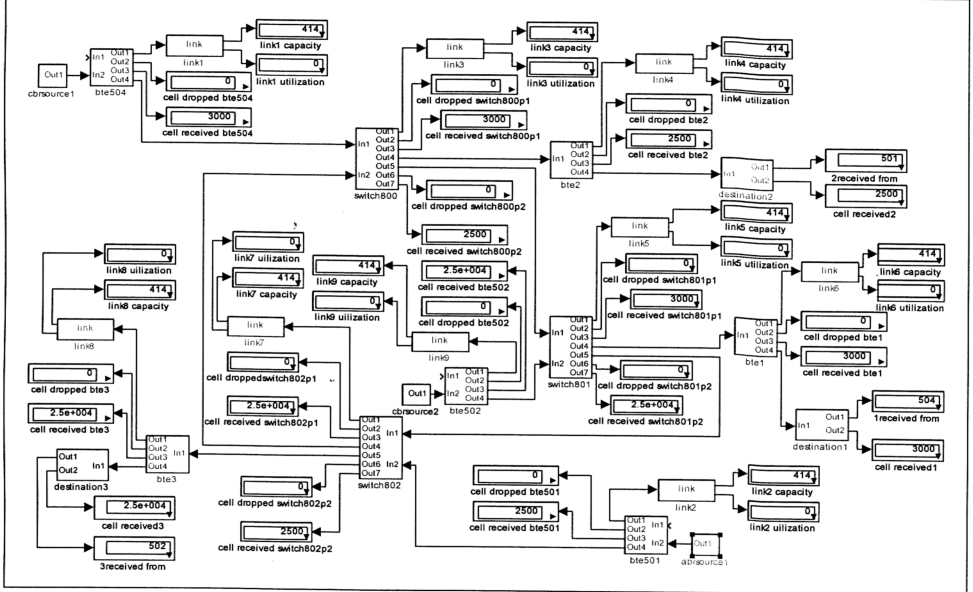

Figure A.5 Ring Topology: 3 switches with large buffer (no cells are dropped)

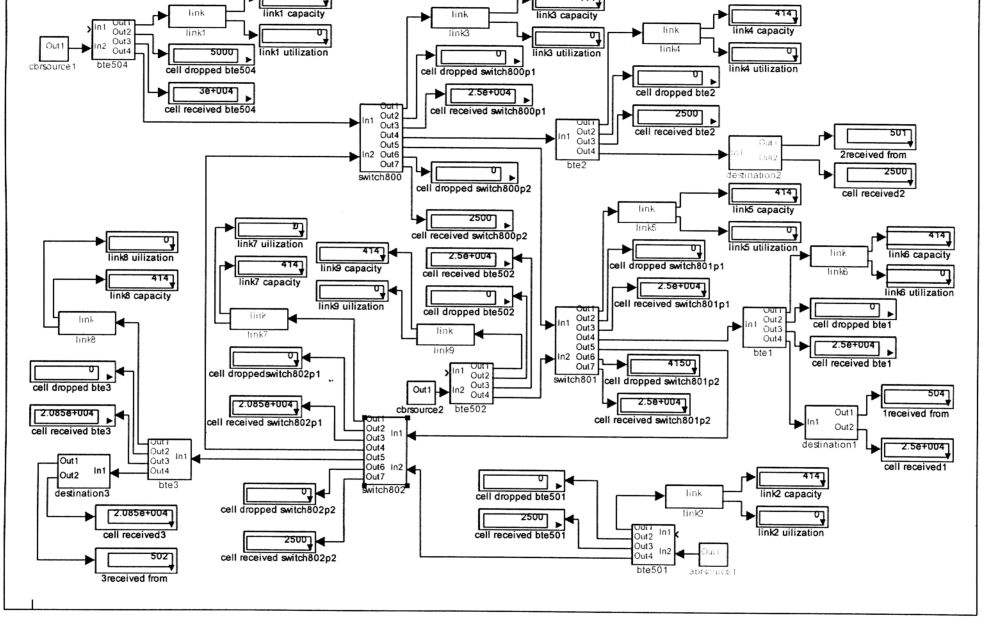

Figure A.6 Ring Topology: 3 switches with small buffer (some cells are dropped)

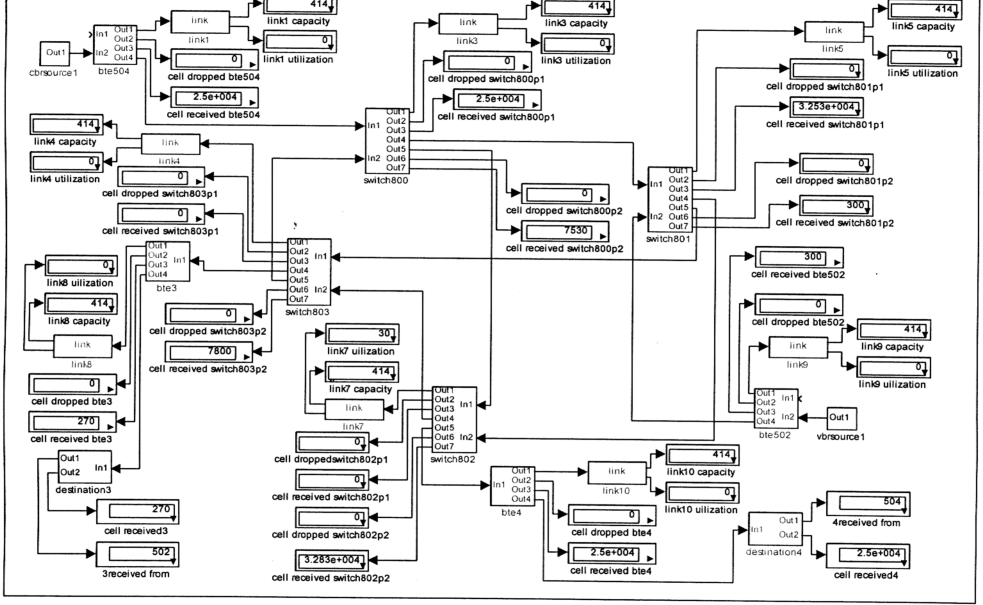

Figure A.7 Mesh Topology: 4 switches connecting each other

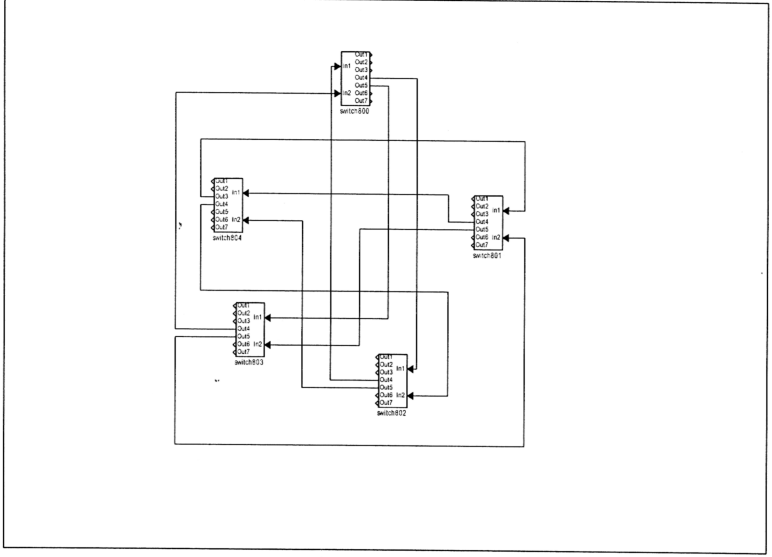

Figure A.8 Star Topology: 4 switches connecting each other

Appendix B

Source Codes of S-Function

```
/*
 * File : abrsource.c
 * Abstract:
 *       An ATM source generate available bit rate traffic
 */


#define S_FUNCTION_NAME  abrsource
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"
#include "tmwtypes.h"


#define MBPS_IDX  0
#define MBPS_PARAM(S) ssGetSFcnParam(S,MBPS_IDX)

#define MBTOSEND_IDX    1
#define MBTOSEND_PARAM(S) ssGetSFcnParam(S,MBTOSEND_IDX)

#define DEST_IDX  2
#define DEST_PARAM(S) ssGetSFcnParam(S,DEST_IDX)

#define NPARAMS   3


#define MDL_CHECK_PARAMETERS
#if defined(MDL_CHECK_PARAMETERS) && defined(MATLAB_MEX_FILE)
 /* Function: mdlCheckParameters ================================================
  * Abstract:
  *    Validate our parameters to verify they are okay.
  */
static void mdlCheckParameters(SimStruct *S)

          /* Check 1st parameter: bit rate in Mbits/sec */
     {
         if (!mxIsNumeric(MBPS_PARAM(S))) {
             ssSetErrorStatus(S,"1st parameter is bit rate in Mbits/sec, must
e a number.");
                 return;
                   }
     }

     /* Check 2nd parameter: total of Mbits to be sent */
     {
         if (!mxIsNumeric(MBTOSEND_PARAM(S))) {
             ssSetErrorStatus(S,"2nd parameter is Mbits to be sent, must be a
umber.");
                 return;
                   }
     }

     /* Check 3rd parameter: destination where bits have to be sent */
     {
         if (!mxIsNumeric(DEST_PARAM(S))) {
             ssSetErrorStatus(S,"3rd parameter is the destination where bits
ave to be sent, must be a number.");
```

```
                return;
                }
        }
}
#endif /* MDL_CHECK_PARAMETERS */


/* Function: mdlInitializeSizes ========================================================
 * Abstract:
 *    Setup sizes of the various vectors.
 */
static void mdlInitializeSizes(SimStruct *S)
{
     ssSetNumSFcnParams(S, NPARAMS);  /* Number of expected parameters */
     #if defined(MATLAB_MEX_FILE)
    if (ssGetNumSFcnParams(S) == ssGetSFcnParamsCount(S)) {
       mdlCheckParameters(S);
       if (ssGetErrorStatus(S) != NULL) {
           return;
       }
    } else {
       return; /* Parameter mismatch will be reported by Simulink */
    }
     #endif

    if (!ssSetNumInputPorts(S, 0)) return;

    if (!ssSetNumOutputPorts(S,2)) return;
    ssSetOutputPortWidth(S, 0, (mxGetPr(MBPS_PARAM(S))[0]*1024)/(48*8));       /*
output per second measured in cell */
    ssSetOutputPortWidth(S, 1, (mxGetPr(MBPS_PARAM(S))[0]*1024)/(48*8));       /*
output per second measured in cell */

     ssSetNumSampleTimes(S, 1);

     ssSetNumIWork(S,5);
     ssSetNumRWork(S,5);

    /* Take care when specifying exception free code */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);


* Function: mdlInitializeSampleTimes ==================================================
* Abstract:
*    Specifiy the sample time.
*/
static void mdlInitializeSampleTimes(SimStruct *S)

    ssSetSampleTime(S, 0, 1.0);
    ssSetOffsetTime(S, 0, 0.0);


define MDL_START
if defined(MDL_START)
* Function: mdlStart ==================================================================
```

```
 * Abstract:
 *    Initialize various value regarding cells to be sent
 */
static void mdlStart(SimStruct *S)
{
       int_T                          *iwork = ssGetIWork(S);
       real_T                         *rwork = ssGetRWork(S);


    rwork[1] = mxGetPr(MBPS_PARAM(S))[0];
    rwork[2] = mxGetPr(MBTOSEND_PARAM(S))[0];
    rwork[3] = mxGetPr(DEST_PARAM(S))[0];

       iwork[0] = (rwork[1]*1024)/(48*8);       /* output per second measured in
cell */
       iwork[1] = (rwork[1]*1024)/(48*8);       /* cell to be sent per second */
       iwork[2] = (rwork[2]*1024)/(48*8);       /* total cell to be sent */
       iwork[3] = (rwork[2]*1024)/(48*8);       /* cell to be sent after period
of time */
}
#endif /* MDL_START */


/* Function: mdlOutputs ========================================================
 * Abstract:
 *    Send cells in one sample time
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
       int_T                          *iwork = ssGetIWork(S);
       real_T                         *rwork = ssGetRWork(S);
       int_T             i;
    int_T             width0 = ssGetOutputPortWidth(S,0);
       real_T                *y0    = ssGetOutputPortRealSignal(S,0);
    int_T             width1 = ssGetOutputPortWidth(S,1);
       real_T                *y1    = ssGetOutputPortRealSignal(S,1);


       /* if still have cells to be sent */
       if (iwork[3] > 0){
            /* get the number of cells to be sent */
            if (iwork[3] < iwork[1]){
                 iwork[0] = iwork[3];
            }/* end if */

         /* sent traffic type 3 = abr */
            for (i=0; i<width0; i++){
                 if (i < iwork[0])
                      *y0++ = 3;
                 else
                      *y0++ = 0;
            }/* end for */

            /* sent destination name */
            for (i=0; i<width1; i++){
                 if (i < iwork[0])
                      *y1++ = rwork[3];
```

```
                else
                        *yl++ = 0;
        }/* end for */
    }/* end if */
    /* if no more cells need to be sent */
    else{
        for (i=0; i<width0; i++){
            *y0++ = 0;
        }
        for (i=0; i<width1; i++){
            *yl++ = 0;
        }
    }/* end else */
}

#define MDL_UPDATE
#if defined(MDL_UPDATE)
/* Function: mdlUpdate ========================================================
 * Abstract:
 *          Update cells yet to be sent
 */
static void mdlUpdate(SimStruct *S, int_T tid)
{
    int_T                       *iwork = ssGetIWork(S);
    real_T                      *rwork = ssGetRWork(S);


    iwork[3] = iwork[3] - iwork[0];
}
#endif /* MDL_UPDATE */


/* Function: mdlTerminate =====================================================
 * Abstract:
 *    No termination needed, but we are required to have this routine.
 */
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
```

```
/*
 * File : atmswitch.c
 * Abstract:
 *       An 2x2 ATM switch from logical view
 */


#define S_FUNCTION_NAME  atmswitch
#define S_FUNCTION_LEVEL 2

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


#define NAME_IDX 0
#define NAME_PARAM(S) ssGetSFcnParam(S,NAME_IDX)

#define BTE1_IDX 1
#define BTE1_PARAM(S) ssGetSFcnParam(S,BTE1_IDX)

#define BTE2_IDX 2
#define BTE2_PARAM(S) ssGetSFcnParam(S,BTE2_IDX)

#define BUFSIZE_IDX  3
#define BUFSIZE_PARAM(S) ssGetSFcnParam(S,BUFSIZE_IDX)

#define OUTLINK_IDX  4
#define OUTLINK_PARAM(S) ssGetSFcnParam(S,OUTLINK_IDX)

#define NPARAMS   5


#define MDL_CHECK_PARAMETERS
#if defined(MDL_CHECK_PARAMETERS) && defined(MATLAB_MEX_FILE)
  /* Function: mdlCheckParameters ===============================================
   * Abstract:
   *    Validate our parameters to verify they are okay.
   */
  static void mdlCheckParameters(SimStruct *S)
  {
      /* Check 1st parameter: Name of the switch */
      {
          if (!mxIsNumeric(NAME_PARAM(S))) {
              ssSetErrorStatus(S,"1st parameter is name of the switch, must be a
number.");
                  return;
                    }
      }

            /* Check 2nd parameter: Name of the first BTE attached */
      {
          if (!mxIsNumeric(BTE1_PARAM(S))) {
              ssSetErrorStatus(S,"2nd parameter is name of the first BTE
attached, must be a number.");
                  return;
                    }
```

```
    }

        /* Check 3rd parameter: Name of the second BTE attached */
    {
        if (!mxIsNumeric(BTE2_PARAM(S))) {
            ssSetErrorStatus(S,"3rd parameter is name of the second BTE
attached, must be a number.");
                return;
                  }
    }

    /* Check 4th parameter: Buffer size where cells can be queue in number of
cells */
    {
        if (!mxIsNumeric(BUFSIZE_PARAM(S))) {
            ssSetErrorStatus(S,"4th parameter is buffer size where cells can
be queue in number of cells, must be a number.");
                return;
                  }
    }

    /* Check 5th parameter: capacity of output link in in Mbits/sec */
    {
        if (!mxIsNumeric(OUTLINK_PARAM(S))) {
            ssSetErrorStatus(S,"5th parameter is capacity of output link in
Mbits/sec, must be a number.");
                return;
                  }
    }

#endif /* MDL_CHECK_PARAMETERS */


/* Function: mdlInitializeSizes ===========================================
 * Abstract:
 *   Setup sizes of the various vectors.
 */
static void mdlInitializeSizes(SimStruct *S)

    ssSetNumSFcnParams(S, NPARAMS);  /* Number of expected parameters */
    #if defined(MATLAB_MEX_FILE)
  if (ssGetNumSFcnParams(S) == ssGetSFcnParamsCount(S)) {
      mdlCheckParameters(S);
      if (ssGetErrorStatus(S) != NULL) {
          return;
      }
  } else {
      return; /* Parameter mismatch will be reported by Simulink */
  }
    #endif

  if (!ssSetNumInputPorts(S, 6)) return;
     ssSetInputPortWidth(S, 0, DYNAMICALLY_SIZED);
  ssSetInputPortWidth(S, 1, DYNAMICALLY_SIZED);
  ssSetInputPortWidth(S, 2, DYNAMICALLY_SIZED);
  ssSetInputPortWidth(S, 3, DYNAMICALLY_SIZED);
  ssSetInputPortWidth(S, 4, DYNAMICALLY_SIZED);
```

```
    ssSetInputPortWidth(S, 5, DYNAMICALLY_SIZED);

    if (!ssSetNumOutputPorts(S,10)) return;
    ssSetOutputPortWidth(S, 0, DYNAMICALLY_SIZED);
    ssSetOutputPortWidth(S, 1, DYNAMICALLY_SIZED);
    ssSetOutputPortWidth(S, 2, (mxGetPr(OUTLINK_PARAM(S))[0]*1024)/(48*8));   /*
output link capacity measured in cell */
    ssSetOutputPortWidth(S, 3, (mxGetPr(OUTLINK_PARAM(S))[0]*1024)/(48*8));   /*
output link capacity measured in cell */
    ssSetOutputPortWidth(S, 4, (mxGetPr(OUTLINK_PARAM(S))[0]*1024)/(48*8));   /*
output link capacity measured in cell */
    ssSetOutputPortWidth(S, 5, (mxGetPr(OUTLINK_PARAM(S))[0]*1024)/(48*8));   /*
output link capacity measured in cell */
    ssSetOutputPortWidth(S, 6, (mxGetPr(OUTLINK_PARAM(S))[0]*1024)/(48*8));   /*
output link capacity measured in cell */
    ssSetOutputPortWidth(S, 7, (mxGetPr(OUTLINK_PARAM(S))[0]*1024)/(48*8));   /*
output link capacity measured in cell */
    ssSetOutputPortWidth(S, 8, DYNAMICALLY_SIZED);
    ssSetOutputPortWidth(S, 9, DYNAMICALLY_SIZED);

       ssSetNumIWork(S,15);
       ssSetNumRWork(S,(12*(int)mxGetPr(BUFSIZE_PARAM(S))[0]));
       ssSetNumPWork(S,2);

       ssSetNumSampleTimes(S, 1);

    /* Take care when specifying exception free code */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}


/* Function: mdlInitializeSampleTimes =========================================
 * Abstract:
 *    Specifiy that sample time inherited from the driving block.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
       ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


#define MDL_START
#if defined(MDL_START)
/* Function: mdlStart =========================================================
 * Abstract:
 *    Initialize current queue size and numbers of cell dropped and received
 */
static void mdlStart(SimStruct *S)
{
       int_T                *iwork;
       real_T              *rwork;
       void                   **pwork = ssGetPWork(S);



       /* memory allocation */
```

```
    iwork = calloc (ssGetNumIWork(S), sizeof(int_T));
    rwork = calloc (ssGetNumRWork(S), sizeof(real_T));

    /* information of first port */
    iwork[0] = 0;           /* current cbr/vbr queue size filled */
    iwork[1] = 0;           /* current abr queue size filled */
    iwork[2] = 0;           /* numbers of cell dropped */
    iwork[3] = 0;           /* numbers of cell received */

    /* information of ATM switch */
    iwork[4] = mxGetPr(NAME_PARAM(S))[0];
    iwork[5] = (int)mxGetPr(BUFSIZE_PARAM(S))[0];
    iwork[6] = mxGetPr(BTE1_PARAM(S))[0];
  iwork[7] = mxGetPr(BTE2_PARAM(S))[0];

  /* information of second port */
    iwork[9] = 0;           /* current cbr/vbr queue size filled */
    iwork[10] = 0;   /* current abr queue size filled */
    iwork[11] = 0;   /* numbers of cell dropped */
    iwork[12] = 0;   /* numbers of cell received */

    /* put the pointers into pointer work vector */
    pwork[0] = iwork;
    pwork[1] = rwork;

}
#endif /* MDL_START */

/* Function: mdlOutputs =======================================================
 * Abstract:
 *    Deliver outgoing cells
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
    int_T                   *iwork;
    real_T                  *rwork;
    void                    **pwork = ssGetPWork(S);
    boolean_T               link_slot_filled;
    int_T           i, j, k , l, m, n, bte_type;
  real_T            *y0    = ssGetOutputPortRealSignal(S,0);
  real_T          *y1    = ssGetOutputPortRealSignal(S,1);
  real_T            *y2    = ssGetOutputPortRealSignal(S,2);
  real_T          *y3    = ssGetOutputPortRealSignal(S,3);
  real_T            *y4    = ssGetOutputPortRealSignal(S,4);
  real_T            *y5    = ssGetOutputPortRealSignal(S,5);
  real_T            *y6    = ssGetOutputPortRealSignal(S,6);
  real_T            *y7    = ssGetOutputPortRealSignal(S,7);
  real_T            *y8    = ssGetOutputPortRealSignal(S,8);
  real_T            *y9    = ssGetOutputPortRealSignal(S,9);
  int_T             width1 = ssGetOutputPortWidth(S,2);
  int_T             width2 = ssGetOutputPortWidth(S,5);


    /* get the pointers from pointer wor0 vector */
    iwork = pwork[0];
    rwork = pwork[1];
```

```
      /* determine the of attached BTE, whether is destination or not */
      if ((iwork[6] < 100) || (iwork[7] < 100)){
            if ((iwork[6] < 100) && (iwork[7] < 100)){
                  bte_type = 1;
            }
            else{
                  if (iwork[6] < 100){
                        bte_type = 2;
                  }
                  else{
                        bte_type = 3;
                  }/* end else */
            }/* end else */
      }/* end if */
      else{
            bte_type = 4;
      }/* end else */


      /* perform switching */
      switch (bte_type)
      {
            case 1:           /* both output ports are attached to destination
btes */


                  j = 0;
         k = 0;
                  l = 0;
                  m = 0;
                  link_slot_filled = 0;

                  /* deliver outgoing cells at first destination port */

                  for (i=0; i<width1; i++){
                        /* from first input port cbr/vbr queue */
                        if ((iwork[0] > 0) && (!link_slot_filled) && (j <
iwork[0])){
                              /* if cbr/vbr queue is not empty */
                              if (rwork[j+2*iwork[5]] == iwork[6]){
                                    /* if cell reaches it destination at first
destination */
                                    *y2++ = rwork[j];
                                    *y3++ = rwork[j+iwork[5]];
                                    *y4++ = rwork[j+2*iwork[5]];
                                    for (n=j; n<iwork[0]-1; n++){
                                          rwork[n] = rwork[n+1];
                                          rwork[n+iwork[5]] =
rwork[n+iwork[5]+1];
                                          rwork[n+(2*iwork[5])] =
rwork[n+(2*iwork[5])+1];
                                    }
                                    iwork[0] = iwork[0] - 1;
                                    link_slot_filled = 1;
                              }/* end if */
                              else{
```

```
                        j = j + 1;
                    }/* end else */
                }/* end if */

                /* from first input port abr queue */
                if ((iwork[1] > 0) && (!link_slot_filled) && (k <
iwork[1])){
                    /* if abr queue is not empty */
                    if (rwork[k+5*iwork[5]] == iwork[6]){
                        /* if cell reaches it destination at first
destination */
                        *y2++ = rwork[k+3*iwork[5]];
                        *y3++ = rwork[k+4*iwork[5]];
                        *y4++ = rwork[k+5*iwork[5]];
                        for (n=k; n<iwork[1]-1; n++){
                            rwork[n+(3*iwork[5])] =
rwork[n+(3*iwork[5])+1];
                            rwork[n+(4*iwork[5])] =
rwork[n+(4*iwork[5])+1];
                            rwork[n+(5*iwork[5])] =
rwork[n+(5*iwork[5])+1];
                        }
                        iwork[1] = iwork[1] - 1;
                        link_slot_filled = 1;
                    }/* end if */
                    else{
                        k = k + 1;
                    }/* end else */
                }/* end if */

                /* from second input port cbr/vbr queue */
                if ((iwork[9] > 0) && (!link_slot_filled) && (l <
iwork[9])){
                    /* if cbr/vbr queue is not empty */
                    if (rwork[l+8*iwork[5]] == iwork[6]){
                        /* if cell reaches it destination at first
destination */
                        *y2++ = rwork[l+6*iwork[5]];
                        *y3++ = rwork[l+7*iwork[5]];
                        *y4++ = rwork[l+8*iwork[5]];
                        for (n=l; n<iwork[9]-1; n++){
                            rwork[n+(6*iwork[5])] =
rwork[n+(6*iwork[5])+1];
                            rwork[n+(7*iwork[5])] =
rwork[n+(7*iwork[5])+1];
                            rwork[n+(8*iwork[5])] =
rwork[n+(8*iwork[5])+1];
                        }
                        iwork[9] = iwork[9] - 1;
                        link_slot_filled = 1;
                    }/* end if */
                    else{
                        l = l + 1;
                    }/* end else */
                }/* end if */

                /* from second input port abr queue */
```

```
                        if ((iwork[10] > 0) && (!link_slot_filled) && (m <
iwork[10])){
                              /* if abr queue is not empty */
                              if (rwork[m+11*iwork[5]] == iwork[6]){
                                    /* if cell reaches it destination at first
destination */
                                    *y2++ = rwork[m+9*iwork[5]];
                                    *y3++ = rwork[m+10*iwork[5]];
                                    *y4++ = rwork[m+11*iwork[5]];
                                    for (n=m; n<iwork[10]-1; n++){
                                          rwork[n+(9*iwork[5])] =
rwork[n+(9*iwork[5])+1];
                                          rwork[n+(10*iwork[5])] =
rwork[n+(10*iwork[5])+1];
                                          rwork[n+(11*iwork[5])] =
rwork[n+(11*iwork[5])+1];
                                    }
                                    iwork[10] = iwork[10] - 1;
                                    link_slot_filled = 1;
                              }/* end if */
                              else{
                                    m = m + 1;
                              }/* end else */
                        }/* end if */

                        /* if both queue is empty */
                        if (!link_slot_filled){
                              *y2++ = 0;
                              *y3++ = 0;
                              *y4++ = 0;
                        }/* end if */

                        link_slot_filled = 0;

                  }/* end for */


                  j = 0;
            k = 0;
                  l = 0;
                  m = 0;
                  link_slot_filled = 0;

                  /* deliver outgoing cells at second destination port */

                  for (i=0; i<width2; i++){
                        /* from second input port cbr/vbr queue */
                        if ((iwork[9] > 0) && (!link_slot_filled) && (l <
iwork[9])){
                              /* if cbr/vbr queue is not empty */
                              if (rwork[l+8*iwork[5]] == iwork[7]){
                                    /* if cell reaches it destination at second
destination */
                                    *y5++ = rwork[l+6*iwork[5]];
                                    *y6++ = rwork[l+7*iwork[5]];
                                    *y7++ = rwork[l+8*iwork[5]];
                                    for (n=l; n<iwork[9]-1; n++){
```

```
                                        rwork[n+(6*iwork[5])] =
rwork[n+(6*iwork[5])+1];
                                        rwork[n+(7*iwork[5])] =
rwork[n+(7*iwork[5])+1];
                                        rwork[n+(8*iwork[5])] =
rwork[n+(8*iwork[5])+1];
                                  }
                                  iwork[9] = iwork[9] - 1;
                                  link_slot_filled = 1;
                            }/* end if */
                            else{
                                  l = l + 1;
                            }/* end else */
                      }/* end if */

                      /* from second input port abr queue */
                      if ((iwork[10] > 0) && (!link_slot_filled) && (m <
iwork[10])){
                            /* if abr queue is not empty */
                            if (rwork[m+11*iwork[5]] == iwork[7]){
                                  /* if cell reaches it destination at second
destination */
                                  *y5++ = rwork[m+9*iwork[5]];
                                  *y6++ = rwork[m+10*iwork[5]];
                                  *y7++ = rwork[m+11*iwork[5]];
                                  for (n=m; n<iwork[10]-1; n++){
                                        rwork[n+(9*iwork[5])] =
rwork[n+(9*iwork[5])+1];
                                        rwork[n+(10*iwork[5])] =
rwork[n+(10*iwork[5])+1];
                                        rwork[n+(11*iwork[5])] =
rwork[n+(11*iwork[5])+1];
                                  }
                                  iwork[10] = iwork[10] - 1;
                                  link_slot_filled = 1;
                            }/* end if */
                            else{
                                  m = m + 1;
                            }/* end else */
                      }/* end if */

                      /* from first input port cbr/vbr queue */
                      if ((iwork[0] > 0) && (!link_slot_filled) && (j <
iwork[0])){
                            /* if cbr/vbr queue is not empty */
                            if (rwork[j+2*iwork[5]] == iwork[7]){
                                  /* if cell reaches it destination at second
destination */
                                  *y5++ = rwork[j];
                                  *y6++ = rwork[j+iwork[5]];
                                  *y7++ = rwork[j+2*iwork[5]];
                                  for (n=j; n<iwork[0]-1; n++){
                                        rwork[n] = rwork[n+1];
                                        rwork[n+iwork[5]] =
rwork[n+iwork[5]+1];
                                        rwork[n+(2*iwork[5])] =
rwork[n+(2*iwork[5])+1];
```

```
                                    }
                                    iwork[0] = iwork[0] - 1;
                                    link_slot_filled = 1;
                              }/* end if */
                              else{
                                    j = j + 1;
                              }/* end else */
                        }/* end if */

                        /* from first input port abr queue */
                        if ((iwork[1] > 0) && (!link_slot_filled) && (k <
iwork[1])){

                              /* if abr queue is not empty */
                              if (rwork[k+5*iwork[5]] == iwork[7]){
                                    /* if cell reaches it destination at second
destination */

                                    *y5++ = rwork[k+3*iwork[5]];
                                    *y6++ = rwork[k+4*iwork[5]];
                                    *y7++ = rwork[k+5*iwork[5]];
                                    for (n=k; n<iwork[1]-1; n++){
                                          rwork[n+(3*iwork[5])] =
rwork[n+(3*iwork[5])+1];

                                          rwork[n+(4*iwork[5])] =
rwork[n+(4*iwork[5])+1];

                                          rwork[n+(5*iwork[5])] =
rwork[n+(5*iwork[5])+1];

                                    }
                                    iwork[1] = iwork[1] - 1;
                                    link_slot_filled = 1;
                              }/* end if */
                              else{
                                    k = k + 1;
                              }/* end else */
                        }/* end if */

                        /* if both queue is empty */
                        if (!link_slot_filled){
                              *y5++ = 0;
                              *y6++ = 0;
                              *y7++ = 0;
                        }/* end if */

                        link_slot_filled = 0;

                  }/* end for */

                  break;

            case 2:           /* first output port is attached to a destination
ote, second to a switch */


                  j = 0;
      k = 0;
                  l = 0;
                  m = 0;
```

```
                link_slot_filled = 0;

                /* deliver outgoing cells at destination port */

                for (i=0; i<width1; i++){
                     /* from first input port cbr/vbr queue */
                     if ((iwork[0] > 0) && (!link_slot_filled) && (j <
iwork[0])){
                          /* if cbr/vbr queue is not empty */
                          if (rwork[j+2*iwork[5]] == iwork[6]){
                               /* if cell reaches it destination */
                               *y2++ = rwork[j];
                               *y3++ = rwork[j+iwork[5]];
                               *y4++ = rwork[j+2*iwork[5]];
                               for (n=j; n<iwork[0]-1; n++){
                                    rwork[n] = rwork[n+1];
                                    rwork[n+iwork[5]] =
rwork[n+iwork[5]+1];
                                    rwork[n+(2*iwork[5])] =
rwork[n+(2*iwork[5])+1];
                               }
                               iwork[0] = iwork[0] - 1;
                               link_slot_filled = 1;
                          }/* end if */
                          else{
                               j = j + 1;
                          }/* end else */
                     }/* end if */

                     /* from first input port abr queue */
                     if ((iwork[1] > 0) && (!link_slot_filled) && (k <
iwork[1])){
                          /* if abr queue is not empty */
                          if (rwork[k+5*iwork[5]] == iwork[6]){
                               /* if cell reaches it destination */
                               *y2++ = rwork[k+3*iwork[5]];
                               *y3++ = rwork[k+4*iwork[5]];
                               *y4++ = rwork[k+5*iwork[5]];
                               for (n=k; n<iwork[1]-1; n++){
                                    rwork[n+(3*iwork[5])] =
rwork[n+(3*iwork[5])+1];
                                    rwork[n+(4*iwork[5])] =
rwork[n+(4*iwork[5])+1];
                                    rwork[n+(5*iwork[5])] =
rwork[n+(5*iwork[5])+1];
                               }
                               iwork[1] = iwork[1] - 1;
                               link_slot_filled = 1;
                          }/* end if */
                          else{
                               k = k + 1;
                          }/* end else */
                     }/* end if */

                     /* from second input port cbr/vbr queue */
                     if ((iwork[9] > 0) && (!link_slot_filled) && (l <
iwork[9])){
```

```
                        /* if cbr/vbr queue is not empty */
                        if (rwork[l+8*iwork[5]] == iwork[6]){
                            /* if cell reaches it destination */
                            *y2++ = rwork[l+6*iwork[5]];
                            *y3++ = rwork[l+7*iwork[5]];
                            *y4++ = rwork[l+8*iwork[5]];
                            for (n=l; n<iwork[9]-1; n++){
                                rwork[n+(6*iwork[5])] =
rwork[n+(6*iwork[5])+1];
                                rwork[n+(7*iwork[5])] =
rwork[n+(7*iwork[5])+1];
                                rwork[n+(8*iwork[5])] =
rwork[n+(8*iwork[5])+1];
                            }
                            iwork[9] = iwork[9] - 1;
                            link_slot_filled = 1;
                        }/* end if */
                        else{
                            l = l + 1;
                        }/* end else */
                    }/* end if */

                    /* from second input port abr queue */
                    if ((iwork[10] > 0) && (!link_slot_filled) && (m <
iwork[10])){
                        /* if abr queue is not empty */
                        if (rwork[m+11*iwork[5]] == iwork[6]){
                            /* if cell reaches it destination */
                            *y2++ = rwork[m+9*iwork[5]];
                            *y3++ = rwork[m+10*iwork[5]];
                            *y4++ = rwork[m+11*iwork[5]];
                            for (n=m; n<iwork[10]-1; n++){
                                rwork[n+(9*iwork[5])] =
rwork[n+(9*iwork[5])+1];
                                rwork[n+(10*iwork[5])] =
rwork[n+(10*iwork[5])+1];
                                rwork[n+(11*iwork[5])] =
rwork[n+(11*iwork[5])+1];
                            }
                            iwork[10] = iwork[10] - 1;
                            link_slot_filled = 1;
                        }/* end if */
                        else{
                            m = m + 1;
                        }/* end else */
                    }/* end if */

                    /* if both queue is empty */
                    if (!link_slot_filled){
                        *y2++ = 0;
                        *y3++ = 0;
                        *y4++ = 0;
                    }/* end if */

                    link_slot_filled = 0;

                }/* end for */
```

```
                j = 0;
        k = 0;
                l = 0;
                m = 0;
                link_slot_filled = 0;


                /* deliver outgoing cells at switch port */

                for (i=0; i<width2; i++){
                      /* from second input port cbr/vbr queue */
                      if ((iwork[9] > 0) && (!link_slot_filled) && (l <
iwork[9])){
                            /* if cbr/vbr queue is not empty */
                            if (rwork[l+8*iwork[5]] != iwork[6]){
                                  /* if cell not reaches it destination */
                                  *y5++ = rwork[l+6*iwork[5]];
                                  *y6++ = rwork[l+7*iwork[5]];
                                  *y7++ = rwork[l+8*iwork[5]];
                                  for (n=l; n<iwork[9]-l; n++){
                                        rwork[n+(6*iwork[5])] =
rwork[n+(6*iwork[5])+1];
                                        rwork[n+(7*iwork[5])] =
rwork[n+(7*iwork[5])+1];
                                        rwork[n+(8*iwork[5])] =
rwork[n+(8*iwork[5])+1];
                                  }
                                  iwork[9] = iwork[9] - 1;
                                  link_slot_filled = 1;
                            }/* end if */
                            else{
                                  l = l + 1;
                            }/* end else */
                      }/* end if */

                      /* from second input port abr queue */
                      if ((iwork[10] > 0) && (!link_slot_filled) && (m <
iwork[10])){
                            /* if abr queue is not empty */
                            if (rwork[m+11*iwork[5]] != iwork[6]){
                                  /* if cell not reaches it destination */
                                  *y5++ = rwork[m+9*iwork[5]];
                                  *y6++ = rwork[m+10*iwork[5]];
                                  *y7++ = rwork[m+11*iwork[5]];
                                  for (n=m; n<iwork[10]-1; n++){
                                        rwork[n+(9*iwork[5])] =
rwork[n+(9*iwork[5])+1];
                                        rwork[n+(10*iwork[5])] =
rwork[n+(10*iwork[5])+1];
                                        rwork[n+(11*iwork[5])] =
rwork[n+(11*iwork[5])+1];
                                  }
                                  iwork[10] = iwork[10] - 1;
                                  link_slot_filled = 1;
                            }/* end if */
```

```
                    else{
                        m = m + 1;
                    }/* end else */
                }/* end if */

                /* from first input port cbr/vbr queue */
                if ((iwork[0] > 0) && (!link_slot_filled) && (j <
iwork[0])){
                    /* if cbr/vbr queue is not empty */
                    if (rwork[j+2*iwork[5]] != iwork[6]){
                        /* if cell not reaches it destination */
                        *y5++ = rwork[j];
                        *y6++ = rwork[j+iwork[5]];
                        *y7++ = rwork[j+2*iwork[5]];
                        for (n=j; n<iwork[0]-1; n++){
                            rwork[n] = rwork[n+1];
                            rwork[n+iwork[5]] =
rwork[n+iwork[5]+1];
                            rwork[n+(2*iwork[5])] =
rwork[n+(2*iwork[5])+1];
                        }
                        iwork[0] = iwork[0] - 1;
                        link_slot_filled = 1;
                    }/* end if */
                    else{
                        j = j + 1;
                    }/* end else */
                }/* end if */

                /* from first input port abr queue */
                if ((iwork[1] > 0) && (!link_slot_filled) && (k <
iwork[1])){
                    /* if abr queue is not empty */
                    if (rwork[k+5*iwork[5]] != iwork[6]){
                        /* if cell not reaches it destination */
                        *y5++ = rwork[k+3*iwork[5]];
                        *y6++ = rwork[k+4*iwork[5]];
                        *y7++ = rwork[k+5*iwork[5]];
                        for (n=k; n<iwork[1]-1; n++){
                            rwork[n+(3*iwork[5])] =
rwork[n+(3*iwork[5])+1];
                            rwork[n+(4*iwork[5])] =
rwork[n+(4*iwork[5])+1];
                            rwork[n+(5*iwork[5])] =
rwork[n+(5*iwork[5])+1];
                        }
                        iwork[1] = iwork[1] - 1;
                        link_slot_filled = 1;
                    }/* end if */
                    else{
                        k = k + 1;
                    }/* end else */
                }/* end if */

                /* if both queue is empty */
                if (!link_slot_filled){
                    *y5++ = 0;
```

```
                        *y6++ = 0;
                        *y7++ = 0;
                    }/* end if */

                    link_slot_filled = 0;

                }/* end for */

                break;


            case 3:             /* first output port is attached to a switch,
second to a destination bte */


                j = 0;
        k = 0;
                l = 0;
                m = 0;
                link_slot_filled = 0;

                /* deliver outgoing cells at destination port */

                for (i=0; i<width1; i++){
                    /* from second input port cbr/vbr queue */
                    if ((iwork[9] > 0) && (!link_slot_filled) && (l <
iwork[9])){
                        /* if cbr/vbr queue is not empty */
                        if (rwork[l+8*iwork[5]] == iwork[7]){
                            /* if cell reaches it destination */
                            *y5++ = rwork[l+6*iwork[5]];
                            *y6++ = rwork[l+7*iwork[5]];
                            *y7++ = rwork[l+8*iwork[5]];
                            for (n=l; n<iwork[9]-1; n++){
                                rwork[n+(6*iwork[5])] =
rwork[n+(6*iwork[5])+1];
                                rwork[n+(7*iwork[5])] =
rwork[n+(7*iwork[5])+1];
                                rwork[n+(8*iwork[5])] =
rwork[n+(8*iwork[5])+1];
                            }
                            iwork[9] = iwork[9] - 1;
                            link_slot_filled = 1;
                        }/* end if */
                        else{
                            l = l + 1;
                        }/* end else */
                    }/* end if */

                    /* from second input port abr queue */
                    if ((iwork[10] > 0) && (!link_slot_filled) && (m <
iwork[10])){
                        /* if abr queue is not empty */
                        if (rwork[m+11*iwork[5]] == iwork[7]){
                            /* if cell reaches it destination */
                            *y5++ = rwork[m+9*iwork[5]];
                            *y6++ = rwork[m+10*iwork[5]];
```

```
                                    *y7++ = rwork[m+11*iwork[5]];
                                    for (n=m; n<iwork[10]-1; n++){
                                          rwork[n+(9*iwork[5])] =
rwork[n+(9*iwork[5])+1];

                                          rwork[n+(10*iwork[5])] =
rwork[n+(10*iwork[5])+1];

                                          rwork[n+(11*iwork[5])] =
rwork[n+(11*iwork[5])+1];

                                    }
                                    iwork[10] = iwork[10] - 1;
                                    link_slot_filled = 1;
                              }/* end if */
                              else{
                                    m = m + 1;
                              }/* end else */
                        }/* end if */

                        /* from first input port cbr/vbr queue */
                        if ((iwork[0] > 0) && (!link_slot_filled) && (j <
iwork[0])){
                              /* if cbr/vbr queue is not empty */
                              if (rwork[j+2*iwork[5]] == iwork[7]){
                                    /* if cell reaches it destination */
                                    *y5++ = rwork[j];
                                    *y6++ = rwork[j+iwork[5]];
                                    *y7++ = rwork[j+2*iwork[5]];
                                    for (n=j; n<iwork[0]-1; n++){
                                          rwork[n] = rwork[n+1];
                                          rwork[n+iwork[5]] =
rwork[n+iwork[5]+1];

                                          rwork[n+(2*iwork[5])] =
rwork[n+(2*iwork[5])+1];

                                    }
                                    iwork[0] = iwork[0] - 1;
                                    link_slot_filled = 1;
                              }/* end if */
                              else{
                                    j = j + 1;
                              }/* end else */
                        }/* end if */

                        /* from first input port abr queue */
                        if ((iwork[1] > 0) && (!link_slot_filled) && (k <
iwork[1])){
                              /* if abr queue is not empty */
                              if (rwork[k+5*iwork[5]] == iwork[7]){
                                    /* if cell reaches it destination */
                                    *y5++ = rwork[k+3*iwork[5]];
                                    *y6++ = rwork[k+4*iwork[5]];
                                    *y7++ = rwork[k+5*iwork[5]];
                                    for (n=k; n<iwork[1]-1; n++){
                                          rwork[n+(3*iwork[5])] =
rwork[n+(3*iwork[5])+1];

                                          rwork[n+(4*iwork[5])] =
rwork[n+(4*iwork[5])+1];

                                          rwork[n+(5*iwork[5])] =
rwork[n+(5*iwork[5])+1];
```

```
                                    }
                                    iwork[1] = iwork[1] - 1;
                                    link_slot_filled = 1;
                              }/* end if */
                              else{
                                    k = k + 1;
                              }/* end else */
                        }/* end if */

                        /* if both queue is empty */
                        if (!link_slot_filled){
                              *y5++ = 0;
                              *y6++ = 0;
                              *y7++ = 0;
                        }/* end if */

                        link_slot_filled = 0;

                  }/* end for */


                  j = 0;
            k = 0;
                  l = 0;
                  m = 0;
                  link_slot_filled = 0;


                  /* deliver outgoing cells at switch port */

                  for (i=0; i<width2; i++){
                        /* from first input port cbr/vbr queue */
                        if ((iwork[0] > 0) && (!link_slot_filled) && (j <
iwork[0])){
                              /* if cbr/vbr queue is not empty */
                              if (rwork[j+2*iwork[5]] != iwork[7]){
                                    /* if cell not reaches it destination */
                                    *y2++ = rwork[j];
                                    *y3++ = rwork[j+iwork[5]];
                                    *y4++ = rwork[j+2*iwork[5]];
                                    for (n=j; n<iwork[0]-1; n++){
                                          rwork[n] = rwork[n+1];
                                          rwork[n+iwork[5]] =
rwork[n+iwork[5]+1];
                                          rwork[n+(2*iwork[5])] =
rwork[n+(2*iwork[5])+1];
                                    }
                                    iwork[0] = iwork[0] - 1;
                                    link_slot_filled = 1;
                              }/* end if */
                              else{
                                    j = j + 1;
                              }/* end else */
                        }/* end if */

                        /* from first input port abr queue */
```

```
                    if ((iwork[1] > 0) && (!link_slot_filled) && (k <
iwork[1])){
                        /* if abr queue is not empty */
                        if (rwork[k+5*iwork[5]] != iwork[7]){
                            /* if cell not reaches it destination */
                            *y2++ = rwork[k+3*iwork[5]];
                            *y3++ = rwork[k+4*iwork[5]];
                            *y4++ = rwork[k+5*iwork[5]];
                            for (n=k; n<iwork[1]-1; n++){
                                rwork[n+(3*iwork[5])] =
rwork[n+(3*iwork[5])+1];
                                rwork[n+(4*iwork[5])] =
rwork[n+(4*iwork[5])+1];
                                rwork[n+(5*iwork[5])] =
rwork[n+(5*iwork[5])+1];
                            }
                            iwork[1] = iwork[1] - 1;
                            link_slot_filled = 1;
                        }/* end if */
                        else{
                            k = k + 1;
                        }/* end else */
                    }/* end if */

                    /* from second input port cbr/vbr queue */
                    if ((iwork[9] > 0) && (!link_slot_filled) && (l <
iwork[9])){
                        /* if cbr/vbr queue is not empty */
                        if (rwork[l+8*iwork[5]] != iwork[7]){
                            /* if cell not reaches it destination */
                            *y2++ = rwork[l+6*iwork[5]];
                            *y3++ = rwork[l+7*iwork[5]];
                            *y4++ = rwork[l+8*iwork[5]];
                            for (n=l; n<iwork[9]-1; n++){
                                rwork[n+(6*iwork[5])] =
rwork[n+(6*iwork[5])+1];
                                rwork[n+(7*iwork[5])] =
rwork[n+(7*iwork[5])+1];
                                rwork[n+(8*iwork[5])] =
rwork[n+(8*iwork[5])+1];
                            }
                            iwork[9] = iwork[9] - 1;
                            link_slot_filled = 1;
                        }/* end if */
                        else{
                            l = l + 1;
                        }/* end else */
                    }/* end if */

                    /* from second input port abr queue */
                    if ((iwork[10] > 0) && (!link_slot_filled) && (m <
iwork[10])){
                        /* if abr queue is not empty */
                        if (rwork[m+11*iwork[5]] != iwork[7]){
                            /* if cell not reaches it destination */
                            *y2++ = rwork[m+9*iwork[5]];
                            *y3++ = rwork[m+10*iwork[5]];
```

```
                                    *y4++ = rwork[m+11*iwork[5]];
                                    for (n=m; n<iwork[10]-1; n++){
                                          rwork[n+(9*iwork[5])] =
rwork[n+(9*iwork[5])+1];
                                          rwork[n+(10*iwork[5])] =
rwork[n+(10*iwork[5])+1];
                                          rwork[n+(11*iwork[5])] =
rwork[n+(11*iwork[5])+1];
                                    }
                                    iwork[10] = iwork[10] - 1;
                                    link_slot_filled = 1;
                              }/* end if */
                              else{
                                    m = m + 1;
                              }/* end else */
                        }/* end if */

                        /* if both queue is empty */
                        if (!link_slot_filled){
                              *y2++ = 0;
                              *y3++ = 0;
                              *y4++ = 0;
                        }/* end if */

                        link_slot_filled = 0;

                  }/* end for */

                  break;


            case 4:           /* both output ports are attached to switches */


                  link_slot_filled = 0;

                  /* deliver outgoing cells at first switch port */

                  for (i=0; i<width1; i++){
                        /* from first input port cbr/vbr queue */
                        if ((iwork[0] > 0) && (!link_slot_filled)){
                            /* if cbr/vbr queue is not empty */
                              *y2++ = rwork[0];
                              *y3++ = rwork[iwork[5]];
                              *y4++ = rwork[2*iwork[5]];
                              for (n=0; n<iwork[0]-1; n++){
                                    rwork[n] = rwork[n+1];
                                    rwork[n+iwork[5]] = rwork[n+iwork[5]+1];
                                    rwork[n+(2*iwork[5])] =
rwork[n+(2*iwork[5])+1];
                              }
                              iwork[0] = iwork[0] - 1;
                              link_slot_filled = 1;
                        }/* end if */

                        /* from first input port abr queue */
                        if ((iwork[1] > 0) && (!link_slot_filled)){
```

```
                        /* if abr queue is not empty */
                        *y2++ = rwork[3*iwork[5]];
                        *y3++ = rwork[4*iwork[5]];
                        *y4++ = rwork[5*iwork[5]];
                        for (n=0; n<iwork[1]-1; n++){
                              rwork[n+(3*iwork[5])] =
rwork[n+(3*iwork[5])+1];
                              rwork[n+(4*iwork[5])] =
rwork[n+(4*iwork[5])+1];
                              rwork[n+(5*iwork[5])] =
rwork[n+(5*iwork[5])+1];
                        }
                        iwork[1] = iwork[1] - 1;
                        link_slot_filled = 1;
                  }/* end if */

                  /* from second input port cbr/vbr queue */
                  if ((iwork[9] > 0) && (!link_slot_filled)){
                        /* if cbr/vbr queue is not empty */
                        *y2++ = rwork[6*iwork[5]];
                        *y3++ = rwork[7*iwork[5]];
                        *y4++ = rwork[8*iwork[5]];
                        for (n=0; n<iwork[9]-1; n++){
                              rwork[n+(6*iwork[5])] =
rwork[n+(6*iwork[5])+1];
                              rwork[n+(7*iwork[5])] =
rwork[n+(7*iwork[5])+1];
                              rwork[n+(8*iwork[5])] =
rwork[n+(8*iwork[5])+1];
                        }
                        iwork[9] = iwork[9] - 1;
                        link_slot_filled = 1;
                  }/* end if */

                  /* from second input port abr queue */
                  if ((iwork[10] > 0) && (!link_slot_filled)){
                        /* if abr queue is not empty */
                        *y2++ = rwork[9*iwork[5]];
                        *y3++ = rwork[10*iwork[5]];
                        *y4++ = rwork[11*iwork[5]];
                        for (n=0; n<iwork[10]-1; n++){
                              rwork[n+(9*iwork[5])] =
rwork[n+(9*iwork[5])+1];
                              rwork[n+(10*iwork[5])] =
rwork[n+(10*iwork[5])+1];
                              rwork[n+(11*iwork[5])] =
rwork[n+(11*iwork[5])+1];
                        }
                        iwork[10] = iwork[10] - 1;
                        link_slot_filled = 1;
                  }/* end if */

                  /* if both queue is empty */
                  if (!link_slot_filled){
                        *y2++ = 0;
                        *y3++ = 0;
                        *y4++ = 0;
```

```
                }/* end if */

                link_slot_filled = 0;

          }/* end for */


          link_slot_filled = 0;


          /* deliver outgoing cells at second switch port */

          for (i=0; i<width2; i++){
                /* from second input port cbr/vbr queue */
                if ((iwork[9] > 0) && (!link_slot_filled)){
                      /* if cbr/vbr queue is not empty */
                      *y5++ = rwork[6*iwork[5]];
                      *y6++ = rwork[7*iwork[5]];
                      *y7++ = rwork[8*iwork[5]];
                      for (n=0; n<iwork[9]-1; n++){
                            rwork[n+(6*iwork[5])] =
rwork[n+(6*iwork[5])+1];
                            rwork[n+(7*iwork[5])] =
rwork[n+(7*iwork[5])+1];
                            rwork[n+(8*iwork[5])] =
rwork[n+(8*iwork[5])+1];
                      }
                      iwork[9] = iwork[9] - 1;
                      link_slot_filled = 1;
                }/* end if */

                /* from second input port abr queue */
                if ((iwork[10] > 0) && (!link_slot_filled)){
                      /* if abr queue is not empty */
                      *y5++ = rwork[9*iwork[5]];
                      *y6++ = rwork[10*iwork[5]];
                      *y7++ = rwork[11*iwork[5]];
                      for (n=0; n<iwork[10]-1; n++){
                            rwork[n+(9*iwork[5])] =
rwork[n+(9*iwork[5])+1];
                            rwork[n+(10*iwork[5])] =
rwork[n+(10*iwork[5])+1];
                            rwork[n+(11*iwork[5])] =
rwork[n+(11*iwork[5])+1];
                      }
                      iwork[10] = iwork[10] - 1;
                      link_slot_filled = 1;
                }/* end if */

                /* from first input port cbr/vbr queue */
                if ((iwork[0] > 0) && (!link_slot_filled)){
                      /* if cbr/vbr queue is not empty */
                      *y5++ = rwork[0];
                      *y6++ = rwork[iwork[5]];
                      *y7++ = rwork[2*iwork[5]];
                      for (n=0; n<iwork[0]-1; n++){
                            rwork[n] = rwork[n+1];
```

```
                                    rwork[n+iwork[5]] = rwork[n+iwork[5]+1];
                                    rwork[n+(2*iwork[5])] =
rwork[n+(2*iwork[5])+1];
                              }
                              iwork[0] = iwork[0] - 1;
                              link_slot_filled = 1;
                        }/* end if */

                        /* from first input port abr queue */
                        if ((iwork[1] > 0) && (!link_slot_filled)){
                              /* if abr queue is not empty */
                              *y5++ = rwork[3*iwork[5]];
                              *y6++ = rwork[4*iwork[5]];
                              *y7++ = rwork[5*iwork[5]];
                              for (n=0; n<iwork[1]-1; n++){
                                    rwork[n+(3*iwork[5])] =
rwork[n+(3*iwork[5])+1];
                                    rwork[n+(4*iwork[5])] =
rwork[n+(4*iwork[5])+1];
                                    rwork[n+(5*iwork[5])] =
rwork[n+(5*iwork[5])+1];
                              }
                              iwork[1] = iwork[1] - 1;
                              link_slot_filled = 1;
                        }/* end if */

                        /* if both queue is empty */
                        if (!link_slot_filled){
                              *y5++ = 0;
                              *y6++ = 0;
                              *y7++ = 0;
                        }/* end if */

                        link_slot_filled = 0;

                  }/* end for */

                  break;

      }/* end switch */

      /* numbers of cell dropped at first port */
      *y0 = iwork[2];
   /* numbers of cell received at first port */
      *y1 = iwork[3];

      /* numbers of cell dropped at second port */
      *y8 = iwork[11];
   /* numbers of cell received at second port */
      *y9 = iwork[12];

      /* put the pointers into pointer work vector */
      pwork[0] = iwork;
      pwork[1] = rwork;
}
```

```
#define MDL_UPDATE
#if defined(MDL_UPDATE)
/* Function: mdlUpdate ========================================================
 * Abstract:
 *          Update buffer/queue
 */
static void mdlUpdate(SimStruct *S, int_T tid)
{
      int_T                              *iwork;
      real_T                             *rwork;
      void                               **pwork = ssGetPWork(S);
      int_T                        i;
   InputRealPtrsType    uPtrs0 = ssGetInputPortRealSignalPtrs(S,0);
   InputRealPtrsType    uPtrs1 = ssGetInputPortRealSignalPtrs(S,1);
   InputRealPtrsType    uPtrs2 = ssGetInputPortRealSignalPtrs(S,2);
   InputRealPtrsType    uPtrs3 = ssGetInputPortRealSignalPtrs(S,3);
   InputRealPtrsType    uPtrs4 = ssGetInputPortRealSignalPtrs(S,4);
   InputRealPtrsType    uPtrs5 = ssGetInputPortRealSignalPtrs(S,5);
   int_T                width0 = ssGetInputPortWidth(S,0);
   int_T                width1 = ssGetInputPortWidth(S,1);
   int_T                width2 = ssGetInputPortWidth(S,2);
   int_T                width3 = ssGetInputPortWidth(S,3);
   int_T                width4 = ssGetInputPortWidth(S,4);
   int_T                width5 = ssGetInputPortWidth(S,5);


      /* get the pointers from pointer work vector */
      iwork = pwork[0];
      rwork = pwork[1];

   /* buffer cells from first input port */
      for (i=0; i<width1; i++) {
            /* buffer cbr/vbr type of traffic */
            if ((*uPtrs1[i] == 1) || (*uPtrs1[i] == 2)){
                  if (iwork[0] < iwork[5]){
                        /* if buffer is not full */
                  rwork[iwork[0]] = *uPtrs0[i];
                        rwork[(iwork[0]+iwork[5])] = *uPtrs1[i];
                        rwork[(iwork[0]+(2*iwork[5]))] = *uPtrs2[i];
                        iwork[0] = iwork[0] + 1;
                  }
                  else{
                        /* if buffer is full, drop the cell */
                        iwork[2] = iwork[2] + 1;
                  }
                  /* cell received */
            iwork[3] = iwork[3] + 1;
            }/* end if */
            else{
                  /* buffer abr type of traffic */
                  if ((*uPtrs1[i] == 3)){
                        /* if buffer is not full */
                        if (iwork[1] < iwork[5]){
                        rwork[iwork[1]+(3*iwork[5])] = *uPtrs0[i];
                              rwork[iwork[1]+(4*iwork[5])] = *uPtrs1[i];
                              rwork[iwork[1]+(5*iwork[5])] = *uPtrs2[i];
```

```
                        iwork[1] = iwork[1] + 1;
                  }
                  else{
                        /* if buffer is full, drop the cell */
                        iwork[2] = iwork[2] + 1;
                  }
                  /* cell received */
               iwork[3] = iwork[3] + 1;
               }/* end if */
         }/* end else */
   }/* end for */


      /* buffer cells from second input port */

      for (i=0; i<width4; i++) {
            /* buffer cbr/vbr type of traffic */
            if ((*uPtrs4[i] == 1) || (*uPtrs4[i] == 2)){
                  if (iwork[9] < iwork[5]){
                        /* if buffer is not full */
                  rwork[(iwork[9]+(6*iwork[5]))] = *uPtrs3[i];
                        rwork[(iwork[9]+(7*iwork[5]))] = *uPtrs4[i];
                        rwork[(iwork[9]+(8*iwork[5]))] = *uPtrs5[i];
                        iwork[9] = iwork[9] + 1;
                  }
                  else{
                        /* if buffer is full, drop the cell */
                        iwork[11] = iwork[11] + 1;
                  }
                  /* cell received */
            iwork[12] = iwork[12] + 1;
            }/* end if */
            else{
                  /* buffer abr type of traffic */
                  if ((*uPtrs4[i] == 3)){
                        /* if buffer is not full */
                        if (iwork[10] < iwork[5]){
                        rwork[iwork[10]+(9*iwork[5])] = *uPtrs3[i];
                              rwork[iwork[10]+(10*iwork[5])] = *uPtrs4[i];
                              rwork[iwork[10]+(11*iwork[5])] = *uPtrs5[i];
                              iwork[10] = iwork[10] + 1;
                        }
                        else{
                              /* if buffer is full, drop the cell */
                              iwork[11] = iwork[11] + 1;
                        }
                        /* cell received */
                  iwork[12] = iwork[12] + 1;
                  }/* end if */
            }/* end else */
   }/* end for */

      /* put the pointers into pointer work vector */
      pwork[0] = iwork;
      pwork[1] = rwork;

}
```

```
#endif /* MDL_UPDATE */


/* Function: mdlTerminate =====================================================
 * Abstract:
 *    Free the memory that was allocated in mdlStart
 */
static void mdlTerminate(SimStruct *S)
{
      int_T                       *iwork;
      real_T                      *rwork;
      void                        **pwork = ssGetPWork(S);


      /* get the pointers from pointer work vector */
      iwork = pwork[0];
      rwork = pwork[1];

      if (iwork != NULL){
           free(iwork);
      }

      if (rwork != NULL){
           free(rwork);
      }
}


#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
```

```
/*
 * File : bte.c
 * Abstract:
 *       An Broad-band Terminal Equipment (BTE) from logical view
 */


#define S_FUNCTION_NAME  bte
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"
#include "tmwtypes.h"

#define NAME_IDX 0
#define NAME_PARAM(S) ssGetSFcnParam(S,NAME_IDX)

#define BUFSIZE_IDX  1
#define BUFSIZE_PARAM(S) ssGetSFcnParam(S,BUFSIZE_IDX)

#define OUTLINK_IDX  2
#define OUTLINK_PARAM(S) ssGetSFcnParam(S,OUTLINK_IDX)

#define NPARAMS   3


#define MDL_CHECK_PARAMETERS
#if defined(MDL_CHECK_PARAMETERS) && defined(MATLAB_MEX_FILE)
  /* Function: mdlCheckParameters ===============================================
   * Abstract:
   *    Validate our parameters to verify they are okay.
   */
  static void mdlCheckParameters(SimStruct *S)
  {
      /* Check 1st parameter: Name of the BTE */
      {
          if (!mxIsNumeric(NAME_PARAM(S))) {
              ssSetErrorStatus(S,"1st parameter is name of the BTE, must be a
number.");
                return;
                  }
      }

      /* Check 2nd parameter: Buffer size where cells can be queue in number of
cells */
      {
          if (!mxIsNumeric(BUFSIZE_PARAM(S))) {
              ssSetErrorStatus(S,"2nd parameter is buffer size where cells can
be queue in number of cells, must be a number.");
                return;
                  }
      }

      /* Check 3rd parameter: capacity of output link in in Mbits/sec */
      {
          if (!mxIsNumeric(OUTLINK_PARAM(S))) {
              ssSetErrorStatus(S,"3rd parameter is capacity of output link in
Mbits/sec, must be a number.");
```

```
                return;
                }
        }
}
#endif /* MDL_CHECK_PARAMETERS */


/* Function: mdlInitializeSizes ===========================================
 * Abstract:
 *    Setup sizes of the various vectors.
 */
static void mdlInitializeSizes(SimStruct *S)
{
      ssSetNumSFcnParams(S, NPARAMS);  /* Number of expected parameters */
      #if defined(MATLAB_MEX_FILE)
    if (ssGetNumSFcnParams(S) == ssGetSFcnParamsCount(S)) {
        mdlCheckParameters(S);
        if (ssGetErrorStatus(S) != NULL) {
            return;
        }
    } else {
        return; /* Parameter mismatch will be reported by Simulink */
    }
      #endif

    if (!ssSetNumInputPorts(S, 3)) return;
       ssSetInputPortWidth(S, 0, DYNAMICALLY_SIZED);
    ssSetInputPortWidth(S, 1, DYNAMICALLY_SIZED);
    ssSetInputPortWidth(S, 2, DYNAMICALLY_SIZED);

    if (!ssSetNumOutputPorts(S,5)) return;
    ssSetOutputPortWidth(S, 0, DYNAMICALLY_SIZED);
    ssSetOutputPortWidth(S, 1, DYNAMICALLY_SIZED);
    ssSetOutputPortWidth(S, 2, (mxGetPr(OUTLINK_PARAM(S))[0]*1024)/(48*8));   /*
output link capacity measured in cell */
    ssSetOutputPortWidth(S, 3, (mxGetPr(OUTLINK_PARAM(S))[0]*1024)/(48*8));   /*
output link capacity measured in cell */
    ssSetOutputPortWidth(S, 4, (mxGetPr(OUTLINK_PARAM(S))[0]*1024)/(48*8));   /*
output link capacity measured in cell */

      ssSetNumIWork(S,10);
      ssSetNumRWork(S,(3*(int)mxGetPr(BUFSIZE_PARAM(S))[0]));

      ssSetNumSampleTimes(S, 1);

    /* Take care when specifying exception free code */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}


/* Function: mdlInitializeSampleTimes =====================================
 * Abstract:
 *    Specifiy that sample time inherited from the driving block.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
      ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
```

```
    ssSetOffsetTime(S, 0, 0.0);
}


#define MDL_START
#if defined(MDL_START)
/* Function: mdlStart =======================================================
 * Abstract:
 *     Initialize current queue size and numbers of cell dropped and received
 */
static void mdlStart(SimStruct *S)
{
       int_T                          *iwork = ssGetIWork(S);

       iwork[1] = 0;           /* current queue size filled */
       iwork[2] = 0;           /* numbers of cell dropped */
       iwork[3] = 0;           /* numbers of cell received */

       iwork[4] = mxGetPr(NAME_PARAM(S))[0];
    iwork[5] = (int)mxGetPr(BUFSIZE_PARAM(S))[0];
}
#endif /* MDL_START */


/* Function: mdlOutputs =====================================================
 * Abstract:
 *     Deliver outgoing cells
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
      int_T                          *iwork = ssGetIWork(S);
      real_T                         *rwork = ssGetRWork(S);
      int_T             i,n;
   real_T               *y0    = ssGetOutputPortRealSignal(S,0);
   real_T             *y1    = ssGetOutputPortRealSignal(S,1);
   real_T               *y2    = ssGetOutputPortRealSignal(S,2);
   real_T             *y3    = ssGetOutputPortRealSignal(S,3);
   real_T               *y4    = ssGetOutputPortRealSignal(S,4);
   int_T                width  = ssGetOutputPortWidth(S,2);


     /* deliver outgoing cells to output ports */

          for (i=0; i<width; i++){
                    /* from queue */
                    if (iwork[1] > 0){
                          /* if queue is not empty */
                          *y2++ = rwork[0];
                          *y3++ = rwork[iwork[5]];
                          *y4++ = rwork[2*iwork[5]];
                          for (n=0; n<iwork[1]-1; n++){
                                rwork[n] = rwork[n+1];
                                rwork[n+iwork[5]] = rwork[n+iwork[5]+1];
                                rwork[n+(2*iwork[5])] =
rwork[n+(2*iwork[5])+1];
                          }
                          iwork[1] = iwork[1] - 1;
```

```
                        }/* end if */
                        else{
                              /* if queue is empty*/
                              *y2++ = 0;
                              *y3++ = 0;
                              *y4++ = 0;
                        }/* end else */
                  }/* end for */

      /* numbers of cell dropped */
      *y0 = iwork[2];
      /* numbers of cell received */
      *y1 = iwork[3];
}


#define MDL_UPDATE
#if defined(MDL_UPDATE)
/* Function: mdlUpdate ============================================================
 * Abstract:
 *          Update buffer/queue
 */
static void mdlUpdate(SimStruct *S, int_T tid)
{
      int_T                                   *iwork = ssGetIWork(S);
      real_T                                  *rwork = ssGetRWork(S);
      int_T                         i;
   InputRealPtrsType    uPtrs0 = ssGetInputPortRealSignalPtrs(S,0);
   InputRealPtrsType    uPtrs1 = ssGetInputPortRealSignalPtrs(S,1);
   InputRealPtrsType    uPtrs2 = ssGetInputPortRealSignalPtrs(S,2);
   int_T                width0 = ssGetInputPortWidth(S,0);
   int_T                width1 = ssGetInputPortWidth(S,1);
   int_T                width2 = ssGetInputPortWidth(S,2);

      /* buffer cells from first input port */
      for (i=0; i<width1; i++) {
           if (*uPtrs1[i] != 0){
                 /* buffer cell */
                 if (iwork[1] < iwork[5]){
                       /* if buffer is not full */
                       if (*uPtrs0[i] == 0){
                       rwork[iwork[1]] = iwork[4];
                       }
                       else{
                             rwork[iwork[1]] = *uPtrs0[i];
                       }
                       rwork[(iwork[1]+iwork[5])] = *uPtrs1[i];
                       rwork[(iwork[1]+(2*iwork[5]))] = *uPtrs2[i];
                       iwork[1] = iwork[1] + 1;
                 }/* end if */
                 else{
                       /* if buffer is full, drop the cell */
                       iwork[2] = iwork[2] + 1;
                 }/* end else */
                 /* cell received */
           iwork[3] = iwork[3] + 1;
           }/* end if */
```

```
    }/* end for */
}
#endif /* MDL_UPDATE */


/* Function: mdlTerminate =====================================================
 * Abstract:
 *    No termination needed, but we are required to have this routine.
 */
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
```

```
/*
 * File : cbrsource.c
 * Abstract:
 *       An ATM source generate constant bit rate traffic
 */


#define S_FUNCTION_NAME  cbrsource
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"
#include "tmwtypes.h"


#define MBPS_IDX  0
#define MBPS_PARAM(S) ssGetSFcnParam(S,MBPS_IDX)

#define MBTOSEND_IDX    1
#define MBTOSEND_PARAM(S) ssGetSFcnParam(S,MBTOSEND_IDX)

#define DEST_IDX  2
#define DEST_PARAM(S) ssGetSFcnParam(S,DEST_IDX)

#define NPARAMS   3


#define MDL_CHECK_PARAMETERS
#if defined(MDL_CHECK_PARAMETERS) && defined(MATLAB_MEX_FILE)
  /* Function: mdlCheckParameters ===============================================
   * Abstract:
   *    Validate our parameters to verify they are okay.
   */
static void mdlCheckParameters(SimStruct *S)
{
          /* Check 1st parameter: bit rate in Mbits/sec */
      {
          if (!mxIsNumeric(MBPS_PARAM(S))) {
              ssSetErrorStatus(S,"1st parameter is bit rate in Mbits/sec, must
be a number.");
                return;
                  }
      }

      /* Check 2nd parameter: total of Mbits to be sent */
      {
          if (!mxIsNumeric(MBTOSEND_PARAM(S))) {
              ssSetErrorStatus(S,"2nd parameter is Mbits to be sent, must be a
number.");
                return;
                  }
      }

      /* Check 3rd parameter: destination where bits have to be sent */
      {
          if (!mxIsNumeric(DEST_PARAM(S))) {
              ssSetErrorStatus(S,"3rd parameter is the destination where bits
have to be sent, must be a number.");
```

```
                return;
                }
        }
}
#endif /* MDL_CHECK_PARAMETERS */


/* Function: mdlInitializeSizes ===========================================
 * Abstract:
 *    Setup sizes of the various vectors.
 */
static void mdlInitializeSizes(SimStruct *S)
{
      ssSetNumSFcnParams(S, NPARAMS);  /* Number of expected parameters */
      #if defined(MATLAB_MEX_FILE)
    if (ssGetNumSFcnParams(S) == ssGetSFcnParamsCount(S)) {
        mdlCheckParameters(S);
        if (ssGetErrorStatus(S) != NULL) {
            return;
        }
    } else {
        return; /* Parameter mismatch will be reported by Simulink */
    }
      #endif

    if (!ssSetNumInputPorts(S, 0)) return;

    if (!ssSetNumOutputPorts(S,2)) return;
    ssSetOutputPortWidth(S, 0, (mxGetPr(MBPS_PARAM(S))[0]*1024)/(48*8));       /*
output per second measured in cell */
    ssSetOutputPortWidth(S, 1, (mxGetPr(MBPS_PARAM(S))[0]*1024)/(48*8));       /*
output per second measured in cell */

       ssSetNumSampleTimes(S, 1);

       ssSetNumIWork(S,5);
       ssSetNumRWork(S,5);

    /* Take care when specifying exception free code */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}


/* Function: mdlInitializeSampleTimes =====================================
 * Abstract:
 *    Specifiy the sample time.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, 1.0);
    ssSetOffsetTime(S, 0, 0.0);
}


#define MDL_START
#if defined(MDL_START)
/* Function: mdlStart =====================================================
```

```
 * Abstract:
 *    Initialize various value regarding cells to be sent
 */
static void mdlStart(SimStruct *S)
{
       int_T                          *iwork = ssGetIWork(S);
       real_T                         *rwork = ssGetRWork(S);


    rwork[1] = mxGetPr(MBPS_PARAM(S))[0];
    rwork[2] = mxGetPr(MBTOSEND_PARAM(S))[0];
    rwork[3] = mxGetPr(DEST_PARAM(S))[0];

       iwork[0] = (rwork[1]*1024)/(48*8);        /* output per second measured in
cell */
       iwork[1] = (rwork[1]*1024)/(48*8);        /* cell to be sent per second */
       iwork[2] = (rwork[2]*1024)/(48*8);        /* total cell to be sent */
       iwork[3] = (rwork[2]*1024)/(48*8);        /* cell to be sent after period
of time */
}
#endif /* MDL_START */


/* Function: mdlOutputs ========================================================
 * Abstract:
 *    Send cells in one sample time
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
       int_T                          *iwork = ssGetIWork(S);
       real_T                         *rwork = ssGetRWork(S);
       int_T             i;
    int_T             width0 = ssGetOutputPortWidth(S,0);
       real_T                *y0    = ssGetOutputPortRealSignal(S,0);
    int_T             width1 = ssGetOutputPortWidth(S,1);
       real_T                *y1    = ssGetOutputPortRealSignal(S,1);


       /* if still have cells to be sent */
       if (iwork[3] > 0){
            /* get the number of cells to be sent */
            if (iwork[3] < &iwork[1]){
                 iwork[0] = iwork[3];
            }/* end if */

         /* sent traffic type 1 = cbr */
            for (i=0; i<width0; i++){
                  if (i < iwork[0])
                        *y0++ = 1;
                  else
                        *y0++ = 0;
            }/* end for */

            /* sent destination name */
            for (i=0; i<width1; i++){
                  if (i < iwork[0])
                        *y1++ = rwork[3];
```

```
                else
                        *y1++ = 0;
            }/* end for */
        }/* end if */
        /* if no more cells need to be sent */
        else{
            for (i=0; i<width0; i++){
                *y0++ = 0;
            }
            for (i=0; i<width1; i++){
                *y1++ = 0;
            }
        }/* end else */
}


#define MDL_UPDATE
#if defined(MDL_UPDATE)
/* Function: mdlUpdate ========================================================
 * Abstract:
 *          Update cells yet to be sent
 */
static void mdlUpdate(SimStruct *S, int_T tid)
{
        int_T                           *iwork = ssGetIWork(S);
        real_T                          *rwork = ssGetRWork(S);


        iwork[3] = iwork[3] - iwork[0];
}
#endif /* MDL_UPDATE */


/* Function: mdlTerminate =====================================================
 * Abstract:
 *    No termination needed, but we are required to have this routine.
 */
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
```

```
/*
 * File : destination.c
 * Abstract:
 *       A destination of traffic source
 */


#define S_FUNCTION_NAME  destination
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"


/* Function: mdlInitializeSizes ========================================================
 * Abstract:
 *    Setup sizes of the various vectors.
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, 0);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return; /* Parameter mismatch will be reported by Simulink */
    }

    if (!ssSetNumInputPorts(S, 3)) return;
    ssSetInputPortWidth(S, 0, DYNAMICALLY_SIZED);
    ssSetInputPortWidth(S, 1, DYNAMICALLY_SIZED);
    ssSetInputPortWidth(S, 2, DYNAMICALLY_SIZED);
    ssSetInputPortDirectFeedThrough(S, 0, 1);

    if (!ssSetNumOutputPorts(S,2)) return;
    ssSetOutputPortWidth(S, 0, DYNAMICALLY_SIZED);
    ssSetOutputPortWidth(S, 1, DYNAMICALLY_SIZED);

       ssSetNumIWork(S,2);

    ssSetNumSampleTimes(S, 1);

    /* Take care when specifying exception free code */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}

/* Function: mdlInitializeSampleTimes ==================================================
 * Abstract:
 *    Specifiy that sample time inherited from the driving block.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


#define MDL_START
#if defined(MDL_START)
/* Function: mdlStart ==================================================================
```

```
 * Abstract:
 *    Initialize numbers of cell received
 */
static void mdlStart(SimStruct *S)
{
      int_T       *iwork = ssGetIWork(S);

   iwork[0] = 0;  /* number of cells received */
}
#endif /* MDL_START */


/* Function: mdlOutputs ========================================================
 * Abstract:
 * Accumulate incoming cells
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
       int_T                        *iwork = ssGetIWork(S);
       int                          i, j;
       InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    real_T              *y0   = ssGetOutputPortRealSignal(S,0);
    real_T              *y1   = ssGetOutputPortRealSignal(S,1);
    int_T               width = ssGetInputPortWidth(S,0);

       j = 0;

       for (i=0; i<width; i++) {
                /* if have incoming cells */
                if ((*uPtrs[i]) != 0){
                     iwork[1] = (*uPtrs[i]);
                     j = j + 1;
                }
    }

       /* source name */
       *y0 = iwork[1];

       /* total numbers of cell received */
       *y1 = iwork[0] + j;

       iwork[0] = iwork[0] + j;
}


/* Function: mdlTerminate =====================================================
 * Abstract:
 *    No termination needed, but we are required to have this routine.
 */
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
```

```
/*
 * File : link.c
 * Abstract:
 *       A link from logical point of view in ATM network
 */


#define S_FUNCTION_NAME  link
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"


/* Function: mdlInitializeSizes ===============================================
 * Abstract:
 *   Setup sizes of the various vectors.
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, 0);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return; /* Parameter mismatch will be reported by Simulink */
    }

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

    if (!ssSetNumOutputPorts(S,2)) return;
    ssSetOutputPortWidth(S, 0, DYNAMICALLY_SIZED);
       ssSetOutputPortWidth(S, 1, DYNAMICALLY_SIZED);

    ssSetNumSampleTimes(S, 1);

    /* Take care when specifying exception free code */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}


/* Function: mdlInitializeSampleTimes =========================================
 * Abstract:
 *    Specifiy that sample time inherited from the driving block.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/* Function: mdlOutputs =======================================================
 * Abstract:
 * Calculate link utilization
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
       int                             i, j;
```

```
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    real_T            *y0   = ssGetOutputPortRealSignal(S,0);
    real_T            *y1   = ssGetOutputPortRealSignal(S,1);
    int_T             width = ssGetInputPortWidth(S,0);

    j = 0;

    for (i=0; i<width; i++) {
            /* if link accupied by cells */
            if ((*uPtrs[i]) != 0){
                j = j + 1;
            }
    }

    /* link capacity measured in numbers of cell can be transmitted per
second */
    *y0 = width;

    /* link utilization measured in numbers of cell have been transmitted per
second */
    *y1 = j;
}


/* Function: mdlTerminate =====================================================
 * Abstract:
 *    No termination needed, but we are required to have this routine.
 */
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
```

```
/*
 * File : vbrsource.c
 * Abstract:
 *       An ATM source generate variable bit rate traffic
 */


#define S_FUNCTION_NAME  vbrsource
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"
#include "tmwtypes.h"


#define MBPS_IDX 0
#define MBPS_PARAM(S) ssGetSFcnParam(S,MBPS_IDX)

#define MBTOSEND_IDX  1
#define MBTOSEND_PARAM(S) ssGetSFcnParam(S,MBTOSEND_IDX)

#define DEST_IDX  2
#define DEST_PARAM(S) ssGetSFcnParam(S,DEST_IDX)

#define NPARAMS   3


#define MDL_CHECK_PARAMETERS
#if defined(MDL_CHECK_PARAMETERS) && defined(MATLAB_MEX_FILE)
  /* Function: mdlCheckParameters ===========================================
   * Abstract:
   *    Validate our parameters to verify they are okay.
   */
static void mdlCheckParameters(SimStruct *S)
{
            /* Check 1st parameter: bit rate in Mbits/sec */
      {
          if (!mxIsNumeric(MBPS_PARAM(S))) {
              ssSetErrorStatus(S,"1st parameter is bit rate in Mbits/sec, must
be a number.");
                  return;
                }
      }

      /* Check 2nd parameter: total of Mbits to be sent */
      {
          if (!mxIsNumeric(MBTOSEND_PARAM(S))) {
              ssSetErrorStatus(S,"2nd parameter is Mbits to be sent, must be a
number.");
                  return;
                }
      }

      /* Check 3rd parameter: destination where bits have to be sent */
      {
          if (!mxIsNumeric(DEST_PARAM(S))) {
              ssSetErrorStatus(S,"3rd parameter is the destination where bits
have to be sent, must be a number.");
```

```
                return;
                }
        }
}
#endif /* MDL_CHECK_PARAMETERS */


/* Function: mdlInitializeSizes ===========================================
 * Abstract:
 *   Setup sizes of the various vectors.
 */
static void mdlInitializeSizes(SimStruct *S)
{
      ssSetNumSFcnParams(S, NPARAMS);  /* Number of expected parameters */
      #if defined(MATLAB_MEX_FILE)
    if (ssGetNumSFcnParams(S) == ssGetSFcnParamsCount(S)) {
        mdlCheckParameters(S);
        if (ssGetErrorStatus(S) != NULL) {
            return;
        }
    } else {
        return; /* Parameter mismatch will be reported by Simulink */
    }
      #endif

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

    if (!ssSetNumOutputPorts(S,2)) return;
    ssSetOutputPortWidth(S, 0, (mxGetPr(MBPS_PARAM(S))[0]*1024)/(48*8));       /*
output per second measured in cell */
    ssSetOutputPortWidth(S, 1, (mxGetPr(MBPS_PARAM(S))[0]*1024)/(48*8));       /*
output per second measured in cell */

       ssSetNumSampleTimes(S, 1);

       ssSetNumIWork(S,5);
       ssSetNumRWork(S,5);

    /* Take care when specifying exception free code */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}


/* Function: mdlInitializeSampleTimes =====================================
 * Abstract:
 *    Specifiy the sample time.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, 1.0);
    ssSetOffsetTime(S, 0, 0.0);
}


#define MDL_START
```

```
#if defined(MDL_START)
/* Function: mdlStart ========================================================
 * Abstract:
 *    Initialize various value regarding cells to be sent
 */
static void mdlStart(SimStruct *S)
{
       int_T                          *iwork = ssGetIWork(S);
       real_T                         *rwork = ssGetRWork(S);


    rwork[1] = mxGetPr(MBPS_PARAM(S))[0];
    rwork[2] = mxGetPr(MBTOSEND_PARAM(S))[0];
    rwork[3] = mxGetPr(DEST_PARAM(S))[0];

       iwork[0] = (rwork[1]*1024)/(48*8);       /* output per second measured in
cell */
       iwork[1] = (rwork[1]*1024)/(48*8);       /* cell to be sent per second */
       iwork[2] = (rwork[2]*1024)/(48*8);       /* total cell to be sent */
       iwork[3] = (rwork[2]*1024)/(48*8);       /* cell to be sent after period
of time */
}
#endif /* MDL_START */


/* Function: mdlOutputs ======================================================
 * Abstract:
 *    Send cells in one sample time
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
       int_T                          *iwork = ssGetIWork(S);
       real_T                         *rwork = ssGetRWork(S);
       int_T            i;
    int_T            width0 = ssGetOutputPortWidth(S,0);
       real_T                *y0    = ssGetOutputPortRealSignal(S,0);
    int_T            width1 = ssGetOutputPortWidth(S,1);
       real_T                *y1    = ssGetOutputPortRealSignal(S,1);


       /* if still have cells to be sent */
       if (iwork[3] > 0){
            /* get the number of cells to be sent */
            if (iwork[0] > iwork[3]){
                 if (iwork[3] < width0){
                      iwork[0] = iwork[3];
                 }
                 else{
                      iwork[0] = width0;
                 }
            }/* end if */
            else{
                 if (iwork[0] > width0){
                      if (iwork[3] < width0){
                           iwork[0] = iwork[3];
                      }
                      else{
```

```
                        iwork[0] = width0;
                    }
                }/* end if */
            }/* end else */

        /* sent traffic type 2 = vbr */
            for (i=0; i<width0; i++){
                if (i < iwork[0])
                    *y0++ = 2;
                else
                    *y0++ = 0;
            }/* end for */

            /* sent destination name */
            for (i=0; i<width1; i++){
                if (i < iwork[0])
                    *y0++ = rwork[3];
                else
                    *y1++ = 0;
            }/* end for */
        }/* end if */
        /* if no more cells need to be sent */
        else{
            for (i=0; i<width0; i++){
                *y0++ = 0;
            }
            for (i=0; i<width1; i++){
                *y1++ = 0;
            }
      }/* end else */
}


#define MDL_UPDATE
#if defined(MDL_UPDATE)
/* Function: mdlUpdate ========================================================
 * Abstract:
 *          Update cells yet to be sent
 */
static void mdlUpdate(SimStruct *S, int_T tid)
{
       int_T                    *iwork = ssGetIWork(S);
       InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    int_T              width = ssGetInputPortWidth(S,0);


       iwork[3] = iwork[3] - iwork[0];
       iwork[0] = (int)(*uPtrs[0]);

}
#endif /* MDL_UPDATE */


/* Function: mdlTerminate =====================================================
 * Abstract:
 *    No termination needed, but we are required to have this routine.
 */
```

```
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
```