เอกสารอ้างอิง

1. J.R.Struas, J.E.Kleckner, A.R.Newton, SPLICE Version 1A.2 User's Guide, Department of EECS, University of California, Berkeley,Ca.

2. J.D.Crawford, M.Y.Hsueh, A.R.Newton, D.O.Pederson, User's Guide for MOTIS-C Version 1.3A, Department of EECS, University of California, Berkeley, Ca,1978.

3. Spectrum SOFTWARE, Micro-logic logic design simulation system,1978.

4. สุพรรณ กุลพานิชย์,กิตติ ตีรเศรษฐ, สิทธิชัย โภไคยอุดม, "การใช้ไมโครคอมพิวเตอร์ช่วยในการออกแบบวงจรตรรก", การประชุมวิชาการทางวิศวกรรมไฟฟ้า 8 สถาบันอุดมศึกษา ครั้งที่ 9, พ.ศ.2529.

5. สุพรรณ กุลพาณิชย์,กิตติ ตีรเศรษฐ, "การใช้ไมโครคอมพิวเตอร์ช่วยในการออกแบบวงจรตรรก 2", การประชุมวิชาการทางวิศวกรรมไฟฟ้า 9 สถาบันอุดมศึกษา ครั้งที่ 10, พ.ศ.2530.

6. Newton, Richard A., "The simulation of large-scale Integrated circuits", U.C. Berkeley Memorandum No. UCB/ERL M78/52, July 1978.

7. Breuer, Melvin A., Friedman, Arthur D., Diagnosis and Reliable design of digital system, Computer Science Pres INC, 1976.

8. โคทม อารียา, วงจรเชิงเลข, วงจรอิเล็กทรอนิกส์, เล่ม 1, ภาคที่ 2, บริษัทซีเอ็ดยูเคชั่น,พิมพ์ครั้งที่ 1, พ.ศ. 2521.

9. Miczo, Alexander, Digital logic testing and simulation,Harper&Row Publisher Inc., Newyork, 1986.

ภาคผนวก

program listing

```
/*-----------------------------
 *      LAP 1.0
 *-----------------------------*/
#include   <syntax.h>            /* Syntax and constant definition */
/*
 *   Machine name
 */
#define        IBMPC              1
/* #define     UNIX               0 */
/*---------------------*
 *  Software constant  *
 *---------------------*/
#define     MAXTIME               1024     /* Max simulation period  */
#define     IDENLEN               6        /* Identifier length      */
#define     CMDLEN                4        /* User command len       */
/*-------------------------------------------*
    Define of some data type used in program
 *-------------------------------------------*/
typedef     char                  NAMESTR[IDENLEN+1];
#define     LOGICSTATE            unsigned char
#define     LOW                   0
#define     HIGH                  1
#define     UNKNOWN               2      /* Unknown  state */
#define     ZSTATE                3      /* High impedance ,donot
                                            change to any value*/
/* Status return from evaluation function */
#define  BUS_CONTENSION       0x80
#define  SPIKE_DETECT         0x40
#define  SEQN_NO_TRIG         0x20
#define  EVAL_NORMAL          0x00
/*---------------------*
    Data Structure
*---------------------*/
#define     MAXIPIN      8                /* Max  I/O pin of model */
#define     MAXOPIN      2                /* Max  OUTPUT IS 2      */
#define     MAXREFOUT    1000             /* Gnd and Vcc maximum FANOUT */
/* Type definition of data used */
typedef  struct  lmodel          LOGICMODEL;
typedef  struct  lnode           LOGICNODE;
typedef  struct  lgate           LOGICELEMENT;
typedef  struct  lsource         LOGICSOURCE;
/*-----------------*
    LOGIC MODEL
 *-----------------*/

/*
    Definition gate Major type
    SEQN  -  standard sequential model
    COMB  -  standard combination model
    FLPF  -  build in flip-flop model
    BUS   -  Bus interface model
*/
enum  main_model{ SEQN,COMB,FLPF,BUS};
/* Define model Minor type */
enum sub_model{AND,OR,INV,NAND,NOR,XOR,BUFF,TFF,DFF,JKFF,DRIVER};

struct  lmodel {
                NAMESTR name;            /* Model  name                */
                enum sub_model   smodel; /* Submodel type              */
```

```
                enum main_model  mmodel; /* Major model type            */
                BYTE clkpin;             /* Pin number of clock for FF  */
                int  inum,onum;          /* Number of I/O pin           */
                int  td;                 /* Transport delay             */
                LOGICSTATE *op[MAXOPIN]; /* Pointer to True Table       */
                LOGICSTATE *state;       /* Pointer to next state table */
                LOGICMODEL *next;
                  } ;
/*-------------*
    LOGIC NODE
 *-------------*/
#define     MAXFANIN              20
#define     MAXFANOUT             20
/* Type of logic node */
#define     INTERN        1     /* Internal node      */
#define     XTERN         2     /* External node      */

struct lnode{
                NAMESTR name;
                BYTE type;               /* Node type             */
                BYTE onum;               /* Output count          */
                BYTE inum;               /* Input count           */
                BYTE simcount;           /* Flag used to detect spike */
                LOGICSTATE  state;       /* logic state of node   */
                LOGICELEMENT  **in,**out;  /* pointer to fin-fout table */
                LOGICSTATE *outstate;      /* pointer to output table   */
                LOGICNODE  *next;
          } ;

/*-----------------*
   LOGIC ELEMENT
 *-----------------*/
    struct lgate {
                NAMESTR name;
                BYTE type;           /* Type of element */
                BYTE onum;           /* Output count    */
                BYTE inum;           /* Input count     */
                LOGICMODEL *model;   /* Model of this element */
                LOGICSTATE ps;       /* Present state for SEQ Element */
                LOGICSTATE cks;      /* Present state of clock pin    */
                int td;              /* Gate delay                    */
                LOGICNODE *in[MAXIPIN],*out[MAXOPIN];
                LOGICELEMENT  *next;
             } ;
/*---------------*
  LOGIC SOURCE
 *---------------*/
/*    Type of logic source      */
#define     LSRC              1   /* USER DEFINE WAVE FORM */
#define     CLKSRC            2   /* CLOCK INPUT           */
#define     MAXSBREAK             20      /* Max Logic source break point  */
     struct lsource {
                NAMESTR name;
                int  stype;          /* Type of logic source  */
                int  index;          /* INDEX COUNT OF SOURCE */
                int  tc;             /* TIME COUNT            */
                int  pnum;           /* PERIOD OF THIS SOURCE */
                LOGICNODE *outnode;  /* NODE THAT THIS SOURCE DRIVE  */
                LOGICSTATE state[MAXSBREAK];  /*  STATE DATA OF SOURCE */
                int  time[MAXSBREAK];  /* time that source change */
                LOGICSOURCE *next;

           } ;
```

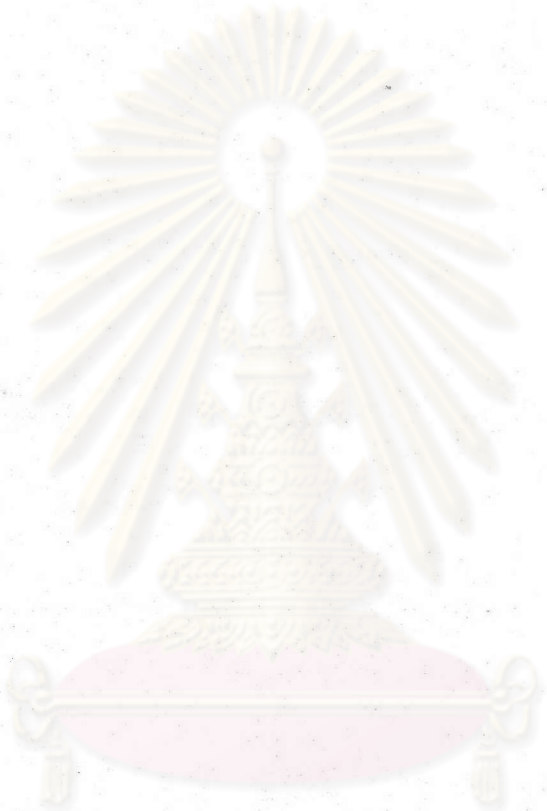
```
/*-------------------------------------------------------------
        COMMAND STRUCTURE FOR MODULE COMMUNICATION
      Communication between each module in program use
      command package.When teminate.each module will send
      back  return status.
 *-------------------------------------------------------------*/

/*    Command sending to DISPLAY MODULE module   */
#define     MAX_DISPLAY_NODE      15 /* Maximum node that can display
                                        simultaneously. */
#define     MAXDPSTEP             500    /* Maximum display step        */
#define     MAXDPSCALE            99     /* Maximum display zoom factor */
#define     MAXGRIDSTEP           200    /* Maximum grid width          */
#define     SETUP                 1
#define     TABLE                 2
#define     TIMING                3
#define     NODESEL               4
#define     INITDP                5
/* Type of display device */
typedef     struct {
                BYTE   cmd;       /* operation perform          */
                int    nodisp;    /* Number of node to display */
                int    timestart; /* Time display index         */
                int    rtime;     /* Real time start            */
                BYTE   scale;     /* Scale factor               */
                BYTE   gridstep;  /* Grid step to display       */
                BYTE   waittime;  /* Waiting time for display   */
                LOGICSTATE *simdata[MAX_DISPLAY_NODE];
                char       *nodename[MAX_DISPLAY_NODE];
                } CMD_DISPLAY;

/* Command sending to simulation module */
#define     SIM_SETUP             1
#define     SIM_NEW               2
#define     SIM_CONT              3
typedef     struct {
                int timelen;        /* Time length of simulation */
                BYTE  cmd;          /* operation perform         */
              }  CMD_SIM;
```

```
/*
 *  MODULE:    LAPDATA
 *  AUTHOR:    P.UTHAYOPAS
 *  DATE:
 *
 *      This module contain routine that handle linklist
 *  of circuit data.
 */
typedef  struct listnode  LIST;
struct listnode {
                 void  *first;    /* first element in linklist  */
                 void  *last;     /* last element in link list  */
                 int   num;       /* number of element in li nk list */
                };

/* Some of macro for link list function */
#define  HEAD(listptr)          listptr->first
#define  TAIL(listptr)          listptr->last
#define  LENGTH(listptr)        listptr->num
#define  LOOPLIST(p,listptr)    for(p=listptr->first; p != NULL ;p=p->next)

/* Some of important external variable */
extern  LOGICNODE       *vccptr,*gndptr; /* Pointer to Ref. node */
extern  LIST            *elmhead ;
extern  LIST            *nodehead;
extern  LIST            *srchead ;

/* Type and argument lint for all module */
#ifdef IBMPC
extern void initdata(void);
extern void makerefnode();
extern LOGICELEMENT *getgateptr(char *name);
extern LOGICELEMENT *makenewgate(char *name);
extern LOGICNODE    *getnodeptr(char *name);
extern LOGICNODE    *makenewnode(char *name);
extern LOGICSOURCE  *getsrcptr( char *name);
extern LOGICELEMENT *oldgate(char *name);
extern LOGICSOURCE  *oldsrc(char *name);
extern LOGICNODE    *oldnode(char *name);
extern int          addelm(LOGICELEMENT *p);
extern int          addnode(LOGICNODE *p);
extern void         setnode(char *nodename,char *st);
#else
   #ifdef UNIX
extern void initdata();
extern void makerefnode();
extern LOGICELEMENT *getgateptr();
extern LOGICELEMENT *makenewgate();
extern LOGICNODE    *getnodeptr();
extern LOGICNODE    *makenewnode();
extern LOGICSOURCE  *getsrcptr();
extern LOGICELEMENT *oldgate();
extern LOGICSOURCE  *oldsrc();
extern LOGICNODE    *oldnode();
extern int          addelm();
extern int          addnode();
extern void         setnode();
  #endif
#endif
```

```
/*
 *  MODULE: LAPEDIT
 *  AUTHOR: P.Uthayopas
 *  DATE:
 *
 *       This module get user circuit script from file
 *  and add to data structure
 */

/* FILE FUNCTION COMMAND CODE */
#define   READ       1
#define   WRITE      2
/* COMMAND STRUCTURE FOR FILE HANDLING */
typedef   struct filecmd  CMDFILE;
struct filecmd {
                char *fname;
               };
#ifdef  IBMPC
extern  void  lapfile(int cmd,CMDFILE *data);
extern  void  edititmain(int argc,char *argv[]);
extern  int   fetchstr(char *st);
#else
#ifdef UNIX
extern  void  lapfile();
extern  void  edititmain();
extern  int   fetchstr();
#endif UNIX
#endif IBMPC
```

```
/* LAPUTIL.H */
/*  Key name for getkey function */
#define BS              8
#define FORMFEED        12
#define CR              13
#define ESC             27
#define HOMEKEY         327
#define ENDKEY          335
#define UPKEY           328
#define DOWNKEY         336
#define PGUPKEY         329
#define PGDNKEY         337
#define LEFTKEY         331
#define INSKEY          338
#define RIGHTKEY        333
#define DELKEY          339
#define CTRLLEFTKEY     371
#define CTRLRIGHTKEY    372
#define F1 315
#define F2 316
#define F3 317
#define F4 318
#define F5 319
#define F6 320
#define F7 321
#define F8 322
#define F9 323
#define F10 324
/*
 *  Define error for laperr
 */
#define  SYNTAXERR              01
#define  INVALIDCMD             02
#define  INVALIDMDL             03
#define  MEMFAULT               04

/* COMMAND CODE TABLE FORMAT */
typedef struct cmdtable {
                 char str[9];    /* Command string */
                 int  code;      /* Command code   */
               } CMDTABLE;

extern int   getkey(void);
extern int   getint(void);
extern void  prmlist(int argc,char *argv[]);
extern int   findcode(char *cmd,CMDTABLE  *cmdtab,int length);
extern char  *getcmdln(char *prompt,char *buff);
extern int   strargv(char *ln,char *argv[]);
extern char  *strtrunc(char *st1,char *st2,int n);
extern char  logictoc(LOGICSTATE s);
extern LOGICSTATE tologic(char c);
/* System interface function */
extern int   dir(char *fname);
extern unsigned long diskfree(int drv);
extern unsigned long memavail(void);
extern void  delfile(char *fname);
extern void  typefile(char *fname);
extern void  copyfile(char *source,char *dest);
```

```
/*-----------
   LAPEVAL.H
 *-----------*/
/*  Function  prototype  */
extern LOGICMODEL *getmodelptr(char *);
extern int listmodel(void);
extern int findlogic(LOGICELEMENT *gate,LOGICSTATE *inpin,LOGICSTATE *outpin);
extern void  add_usr_model(LOGICMODEL *model);
extern void  free_usr_model(void);
extern LOGICMODEL *dupmodel(LOGICMODEL *model);
```



```
/*----------------*
    LAPMACRO.H
 *----------------*/
/* Data type used */
typedef  struct  macro    MACRO;
typedef  struct  ingate   MACROGATE;
typedef  struct  msym     MACROSYM;
struct   msym {
                 NAMESTR  str;
                 LOGICNODE *node;
              };
struct ingate {
               NAMESTR   name;
               LOGICMODEL *model;
               int td;
               int inum,onum;
               MACROSYM   *ipin[MAXIPIN];
               MACROSYM   *opin[MAXOPIN];
               struct ingate *next;
              };

struct macro {
              NAMESTR name;                    /* Macro name */
              MACROGATE *gfirst,*glast;  /* Pointer to macro gate link */
              MACROSYM  *xsym,*isym;     /* Pointer to symbol table    */
              int elmnum;        /* Macro gate element count      */
              int inodenum;      /* Internal node count           */
              int xnodenum;      /* External node count           */
              int expand;        /* Macro expansion count         */
              struct  macro *next;
             };

/* Function */
extern void     expandmacro(int argc,char *argv[],MACRO *m);
extern void     editmacro(int margc,char *margv[]);
extern void     listmacro(void);
extern void     freemacro(void);
extern MACRO    *getmacroptr(char *st);
```

```
/*
 *     MODULE:  main
 *     AUTHOR:  P.Uthayopas  B815754
 *     DATE:
 *              This is main module of LAP 1.0 a logic simulation
 *     program on IBM PC/XT.
 */
#include   <stdio.h>
#include   <signal.h>
#include   "lap.h"
#include   "lapcmd.h"
#include   "laputil.h"
#include   "lapedit.h"

#define  MAXARGUMENT            20     /* Maximum command line argument */

/*
 *   IMPORT FUNCTION
 */
#ifdef   IBMPC
IMPORT   void  simulator(CMD_SIM *cmdptr); /* From lapsim.c  */
IMPORT   void  dspmodule(CMD_DISPLAY *cmdptr);     /* From lapdisp   */
IMPORT   void  creat_tq();
#endif
/*
 *  Intern' used function of this module
 */
#ifdef   IBMPC
LOCAL    void  savemain(int argc,char *argv[]);
LOCAL    void  loadmain(int argc,char *argv[]);
LOCAL    void  simmain();
LOCAL    void  displaymain(int argc,char *argv[]);
LOCAL    void  helpmain(void);
LOCAL    void  initdpmodule(void);
#else
#if      UNIX
LOCAL    void  savemain();
LOCAL    void  loadmain();
LOCAL    void  simmain();
LOCAL    void  displaymain();
LOCAL    void  helpmain();
LOCAL    void  initdpmodule();
#endif
#endif
/*
 *   Title : print title message on screen
 */
LOCAL void title()
{
printf("MicroLAP 1.0 : Microcomputer Logic Analysis Program\n\n");
  puts("Developed By : Putchong Uthayopas,B815754");
  puts("               Department of Electrical Engineering");
  puts("               Faculty of Engineering");
  puts("               Chulalongkorn University");
printf("               September ,1988\n\n");
}
/*
 *   LAP HELP MODULE
 *   help file is lap.hlp
 */
```

```
PUBLIC void helpmain()
{
LOCAL char *hlp="LAP.HLP";   /* Help file name */
LOCAL char  linebuff[150];
   FILE *hf;
   int lc;

   if ((hf=fopen(hlp,"r"))==NULL) then
    {  puts("Help file LAP.HLP not found in current directory"); }
    else
     {
       for(lc=0; ! feof(hf); )
        {
          linebuff[0]=NULL;
          fgets(linebuff,150,hf);
          printf("%s",linebuff);
          lc=( lc < 20 ? ++lc : 0);
          if (lc==0) getch();
        }
        fclose(hf);
     }
   printf("\n");
}

/*
 *   File handling interface module
 */

/*
 *       savemain
 *    This function call file saving function
 */
 LOCAL void savemain(argc,argv)
 int argc;
 char *argv[];
 {
    static CMDFILE  buff,*p=&buff;
    char name[20];

    if (argc !=2 ) then
       {
          printf("SAVE <Filename>\n");
          return;
       }
    strcat(argv[1],".LAP");
    p->fname=argv[1];
    lapfile(WRITE,p);
 }

 LOCAL void loadmain(argc,argv)
 int argc;
 char *argv[];
 {
    static CMDFILE  buff,*p=&buff;
    char name[20];

    if (argc !=2 ) then
       {
          printf("LOAD <Filename>\n");
          return;
```

```
        }
    strcat(argv[1],".LAP");
    p->fname=argv[1];
    lapfile(READ,p);
    printf("\n");
}
/*
 *  Simulation module interface
 */
void  simmain(argc,argv)
int argc;
char *argv[];
{
    /* Command for simulator module */
    static CMD_SIM  simbuff,*simcmd=&simbuff;
    static CMDTABLE simcmdtab[]={
                            {"CONT",1},{"STAR",2},{"SET",3}
                             };
    static int  simcmdlen = sizeof(simcmdtab)/sizeof(CMDTABLE);
    char buff[10];
    int  code;

     if (argc >3) { puts("SYNTAX ERROR!"); return; }
     if (argc==1)
        { simcmd->cmd=SIM_NEW;
          simulator(simcmd);
          return; }
     strtrunc(buff,argv[1],CMDLEN); /* Truncate string  */
     code=findcode(buff,simcmdtab,simcmdlen);
     switch(code)
     {
        case 1 :  simcmd->cmd=SIM_CONT;          break;
        case 2 :  simcmd->cmd=SIM_NEW;           break;
        case 3 :  simcmd->cmd=SIM_SETUP;
                  simcmd->timelen=MAXTIME;          break;
        default :   puts("SYNTAX ERROR!");       return;
     }
     simulator(simcmd);
}
/*
 *   Display module interface
 */

/*
 * Share command buffer for display module
 */
LOCAL CMD_DISPLAY disptag;
LOCAL CMD_DISPLAY *dispcmd = &disptag;

LOCAL void initdpmodule()
{
       dispcmd->cmd=INITDP;
       dspmodule(dispcmd);
}

LOCAL void displaymain(argc,argv)
int argc;
char *argv[];
{
    static CMDTABLE dspcmd[]={
```

```
                                {"SET",1},
                                {"TABLE",2},
                                {"SIGNAL",3},
                                {"NODE",4}
                            };
    static int dspcmdlen = sizeof(dspcmd)/sizeof(CMDTABLE);
    static char buff[IDENLEN+2];
    int code;

    dispcmd->waittime=0;
    if (argc==1) then
      {
        dispcmd->cmd=TIMING;
        dspmodule(dispcmd);
        return;
       }
    strtrunc(buff,argv[1],IDENLEN); /* Truncate string  */
    code=findcode(buff,dspcmd,dspcmdlen);
    switch(code)
    {
       case 1   :  dispcmd->cmd=SETUP;
                   break;
       case 2   :  dispcmd->cmd=TABLE;
                   break;
       case 3   :  dispcmd->cmd=TIMING;
                   dispcmd->waittime=atoi(argv[2]);
                   break;
       case 4   :  dispcmd->cmd=NODESEL;
                   break;
    }
    dspmodule(dispcmd);
} /* end of Displaymain */

/*
 *   SETUP PARAMETER MAIN
 */

LOCAL void setupmain(argc,argv)
int argc;
char *argv[];
{

   static CMDTABLE setcmd[]={
            {"DEVICE",1},{"DEV",1} ,{"NODE",2}
                            };
   static int   setcmdlen = sizeof(setcmd)/sizeof(CMDTABLE);

   static char buff[IDENLEN+2];
   int code;

   if (argc==1) { puts("SET <DEVICE,NODE,SIM>"); return; }
   strtrunc(buff,argv[1],IDENLEN); /* Truncate string  */
   code=findcode(buff,setcmd,setcmdlen);
   switch(code)
   {
     case 1 :
            puts("SET DEVICE PARAMETER - NOT IMPLEMENT");
            break;
     case 2 :
            setnode(argv[2],argv[3]);
```

```
            break;
      default :
            puts("INCORRECT PARAMETER FOR SET COMMAND");
   }
}
/*
 *
 *   Dos command  eg. DIR,TYPE,DEL,COPY
 *
 */

void dirmain(argc,argv)
int argc;
char *argv[];
{
   if (argc==1)
     {
        dir("*.LAP");
        return;
     }
   if (argc==2)
     {
       dir(argv[1]);
       return;
     }
   puts("SYNTAX ERROR!");
}

void pause(argc,argv)
int argc;
char *argv[];
{
   if (argc ==1) { waitkey(); return;}
   if (argc ==2)  sleep(atoi(argv[1]) );
}

LOCAL void showfreemem()
{
    unsigned long memory;
    memory=memavail();
    printf("memory available:%10ld Byte\n",memory );
}
void delmain(argc,argv)
int argc;
char *argv[];
{
   if (argc !=2) return;
   delfile(argv[1]);
}
void typemain(argc,argv)
int argc;
char *argv[];
{
   if (argc !=2) return;
   typefile(argv[1]);
}
void copymain(argc,argv)
int argc;
char *argv[];
{
```

```
   if (argc != 3) return;
   copyfile(argv[1],argv[2]);
}

/*-----------------------*
   New circuit command
 *-----------------------*/
LOCAL void newcircuit()
{
     freegate();
     freenode();
     freesrc();
     freemacro();
     free_usr_model();
     initdata();
     initdpmodule();
}

/*-----------------------------*
    M A I N    P R O G R A M
 *-----------------------------*/

/*    Share Variable in lapmain */

LOCAL  char       *prompt ="LAP>";
LOCAL  char       pathname[40];
LOCAL  char       *argv[MAXARGUMENT];
LOCAL  int        argc;
LOCAL  char       cmdbuffer[20][20];
LOCAL  char       linebuff[256];   /* Command input buffer */


LOCAL void initmain()
{
    int i;

    /* Reserve area for argument string */
    for(i=0;i<20;i++)
      argv[i]  = cmdbuffer[i];
    /*  Make Vcc and GND node */
    makerefnode();
    /* Init routine that handle data structure  */
    initdata();
    initdpmodule();
    create_tq();    /* create time queue */
    showfreemem();
}

/* This routine fetch command string from user */
int  fetch(argv)
char *argv[];
{
     IMPORT char *prompt,linebuff[];
     int argc;

     for(argc=0; (argc=strargv(getcmdln(prompt,linebuff) , argv))==0;);
     return(argc);
}

int cmdexec(argc,argv)
```

```
int argc;
char *argv[];
{
static CMDTABLE  cmdtab[]={
                        {"QUIT",1},{"EXIT",1},
                        {"LIST",2},{"SIM" ,3},
                        {"DISP",4},{"SAVE",5},
                        {"LOAD",6},{"DEFI",7},
                        {"HELP",9},{"ED",8},
                        {"NEW",11},{"MEM",10},
                        {"SET",13} ,{"DIR",14},
                        {"PAUS",15},{"DEL",16},
                        {"TYPE",17},{"COPY",18}
                          };
static int      cmdtablen = sizeof(cmdtab)/sizeof(CMDTABLE);
static char buff[CMDLEN+2];
static int code;

      /* Command in LAP use only first CMDLEN character */
      strtrunc(buff,argv[0],CMDLEN);  /* Truncate string */
      code=findcode(buff,cmdtab,cmdtablen);
      /*  Command switching   */
      switch(code)
       {
        case  1  : break;   /* exit and quit command */
        case  2  :
                   listmain(argc,argv);
                   break;
        case  3  :
                   simmain(argc,argv);
                  break;
       case  4  :
                  displaymain(argc,argv);
                  break;
       case  5  :
                  savemain(argc,argv);
                  break;
       case  6  :
                  loadmain(argc,argv);
                  break;
       case  7  :
                  puts("DEFINE NOT IMPLEMENT NOW");
                  break;
       case  8  :
                  printf("EDIT>"); getch();
                  printf("\n");
                  break;
       case  9  :
                  helpmain();
                  break;
       case 10  :
                  showfreemem();
                  break;
       case 11  :
                  newcircuit();
                  break;
       case 13  :
                  setupmain(argc,argv);
                  break;
       case 14  :
```

```
                dirmain(argc,argv);
                break;
        case 15  : pause(argc,argv);
                break;
        case 16  : delmain(argc,argv);
                break;
        case 17  : typemain(argc,argv);
                break;
        case 18  : copymain(argc,argv);
                break;
        default  :
                puts("Invalid command!\07");
         }
       return(code);   /* Return action code to caller */
}

#define  EXIT_ACTION              1

#ifdef   UNIX
void brk_sig()
{
   char ans;

   printf("Press Y to abort program:");
   ans=getchar();
   if (ans=='Y')
     {
       printf("Program terminate by user!!\n");
       exit(1);
     }
}
#endif

void main()
{
    int  action;

#ifdef  UNIX
    ssignal(SIGINT,brk_sig);
#endif
    title();
    initmain();
    do {
      argc=fetch(argv);
      action=cmdexec(argc,argv);
    } while ( action != EXIT_ACTION);  /* Action code 1 is exit action */
    newcircuit();  /* Clear all circuit */
    printf("** End Logic Analysis Program LAP1.0 **\n");
    exit(0);
}
```

```
/*
 *  MODULE: LAPEDIT
 *  AUTHOR: P.Uthayopas
 *  DATE:
 *
 *       This module get user circuit script from file
 *  and add to data structure
 */
#include  <stdio.h>
#include  <string.h>
#include  <mem.h>
#include "lap.h"
#include "lapdata.h"
#include "laputil.h"
#include "lapedit.h"
#include "lapeval.h"
#include "lapmacro.h"

#define   COMMENTCHAR     '*'
#define   NORMAL_MODEL    1
#define   MACRO_MODEL     2
#define   GARBAGE_MODEL   3
/*--------------------*
   E D I T   M A I N
 *--------------------*/

static  char  line[255];    /* Input buffer */
FILE    *network;           /* Input file   */
/* PUBLIC NAMESTR circuitname; */ /* name of this circuit */

/*
 *   Internal function
 */
#ifdef  IBMPC
LOCAL  void  editsrc(int argc,char *argv[]);
LOCAL  void  editclk(int argc,char *argv[]);
LOCAL  void  editgate(int argc,char *argv[]);
LOCAL  void  editbus(int argc,char *argv[]);
LOCAL  void  editmodel(int argc,char *argv[]);
LOCAL  void  editnode(int argc,char *argv[]);
LOCAL  LOGICMODEL  *chkelm(int argc,char *argv[]);
LOCAL  int   looktype(char *name,void **model);
LOCAL  void  normalgate(int argc,char *argv[],LOGICMODEL *m);
#else
  #ifdef UNIX
LOCAL  void  editsrc ();
LOCAL  void  editclk ();
LOCAL  void  editgate();
LOCAL  void  editbus();
LOCAL  void  editmodel();
LOCAL  void  editnode();
LOCAL  LOGICMODEL  *chkelm();
LOCAL  int   looktype();
LOCAL  void  normalgate();
  #endif
#endif

PUBLIC void editmain(argc,argv)
int argc;
char *argv[];
```

```
{
   static CMDTABLE gatecmd[]={
                {"SIG",1},{"SIGN",1},
                {"CLK",2},{"CLOC",2},
                {"BUS",3},{"NODE",4},
                {"MODE",5},{"MACR",6}
                        };
   static int   gatecmdlen = sizeof(gatecmd)/sizeof(CMDTABLE);

   static char buff[IDENLEN+2];
   int code;

   if (argc==0) then return; /* Do nothing for NULL argument */
   strtrunc(buff,argv[1],CMDLEN); /* Truncate string  */
   code=findcode(buff,gatecmd,gatecmdlen);
   switch(code)
   {
     case 1  :
             editsrc(argc,argv);
             break;
     case 2  :
             editclk(argc,argv);
             break;
     case 3  :
             editbus(argc,argv);
             break;
     case 4  :
             editnode(argc,argv);
             break;
     case 5  :
             editmodel(argc,argv);
             break;
     case 6  :
             editmacro(argc,argv);
             break;
     default :
             editgate(argc,argv);
   }
}
void editsrc(argc,argv)
int argc;
char *argv[];
{
    LOGICSOURCE *p;
    char sname[10];
    int i;

    if (argc != 4) then {
      printf("<NAME> SIGNAL <OUTPUT NODE> <NO. OF BREAK POINT>\n"); return; }
    strtrunc(sname,argv[0],IDENLEN);
    p=getsrcptr(sname);  /* get pointer to source */
    p->stype=LSRC;
    p->index=0; p->tc=0;
    /* Maximum number of break point is MAXSBREAK */
    p->pnum= ( (p->pnum=atoi(argv[3])) > MAXSBREAK ? MAXSBREAK : p->pnum);
    strtrunc(sname,argv[2],IDENLEN);
    p->outnode=getnodeptr(sname);
    /* Read waveform */
    for(i=0;i<p->pnum;i++) {
         fetchstr(line);
```

```
        strargv(line,argv);
        p->time[i]=atoi(argv[0]);
        p->state[i]=tologic(*argv[1]);
        printf("%5d%5s\n",p->time[i],argv[1]);
      }
}

void editclk(argc,argv)
int argc;
char *argv[];
{
    LOGICSOURCE *p;
    char sname[10];
    int ton,toff;

    if (argc != 5) then {
       printf("<NAME> CLOCK <OUTPUT NODE> <Ton ns> <Toff ns>\n"); return; }
    strtrunc(sname,argv[0],IDENLEN);
    p=getsrcptr(sname);  /* get pointer to source */
    p->stype=CLKSRC;
    p->index=0; p->tc=0;
    p->pnum=2;
    p->state[0]=LOW; p->state[1]=HIGH;
    p->time[0]=atoi(argv[4]);  /* T  on */
    p->time[1]=atoi(argv[3]);  /* T off */
    p->outnode=getnodeptr(argv[2]);
}

void editbus(argc,argv)
int argc;
char *argv[];
{
    LOGICELEMENT *p;
    LOGICMODEL   *m;
    char busname[10];
    register i,index;
    int fanin;

    if (argc<3) then
      {
        printf("<BUSNAME> BUS <INPUT 1..n> <OUTPUT>\n ");
        printf("<BUSNAME> BUS <Time delay (nSec) >");
      }
    if (argc > 3) /* ADD NEW BUS ELEMENT */
     {
        m=getmodelptr(argv[1]);
        fanin=argc-3;
        if(fanin > m->inum)
          {   printf("ERROR! TOO MUCH BUS FAN-IN,MAXIMUM=%d\n",m->inum);
              return;
          }
        strtrunc(busname,argv[0],IDENLEN);
        if ((p=oldgate(busname))!=NULL)
          {  printf("BUS %s ALREADY EXIST!\07\n",busname);
             return; }
        p=getgateptr(busname);
        p->inum=fanin;
        p->model=m;
        p->onum=m->onum; /* Bus output is 1 pin      */
        p->td=m->td;     /* Set bus delay to default */
```

```
        index=2;
        /* init all fanin */
        for(i=0;i<p->inum;i++)
           { p->in[i]=getnodeptr(argv[index++] ); }
        /* init 1 fanout */
        p->out[0]=getnodeptr(argv[index]);
        p->ps=0;  /* Present state is 0 */
      }
    else /* SET BUS ELEMENT DELAY */
      {
       /* Force time delay minimum 1 ns */
        strtrunc(busname,argv[0],IDENLEN);
        if ((p=oldgate(busname)) !=NULL) {
           p->td= ( (p->td=atoi(argv[2])) > 0 ? p->td : 1 ); }
         else  printf("BUS %s DOES NOT EXIST!\07\n",busname);
      }
}
void  editmodel(argc,argv)
int argc;
char *argv[];
{

    LOGICMODEL *p,*m;
    char modelname[10];
    if (argc != 4 )
      {
        printf("<NAME> MODEL  <BUILD IN MODEL>  <Td>\n");
        return;
      }
    strtrunc(modelname,argv[0],IDENLEN); /* Truncate model name */
    if ((p=getmodelptr(modelname)) !=NULL)
      {
        printf("Model already exist,redefine fail!\n");
        return;
      }
    if ((p=getmodelptr(argv[2])) ==NULL)
      {
        printf("Please choose correct build in model to copy !\n");
        printf("<NAME> MODEL  <BUILD IN>  <Td>\n");
        return;
      }
    m=dupmodel(p);
    strcpy(m->name,modelname);
    /* Allow timedelay = 0 for testing */
    m->td=( (m->td=atoi(argv[3])) >=0 ? m->td : 1);
    add_usr_model(m);
}

void  editnode(argc,argv)
int argc;
char *argv[];
{
    LOGICNODE *p;

    if (argc !=3)
      {  puts("<NAME>  NODE  <LOGIC STATE>");
         return; }
    p=getnodeptr(argv[0]);
  /* Convert first char of argv[2] to logic state */
```

```
    p->state=tologic(*argv[2]);
}

void  normalgate(argc,argv,m)
int argc;
char *argv[];
LOGICMODEL *m;
{
    LOGICELEMENT *p;

    char gatename[10];
    register i;
    int index,x;

    x=argc - (m->inum + m->onum) ;
    if (! ( (x<2) || (x>3) )) then
     {   /* Syntax checking passed */
         strtrunc(gatename,argv[0],IDENLEN);
         if ( oldgate(gatename) != NULL)
           {  printf(" GATE EXIST! NOT ALLOW TO REDEFINE\07\n");
              return; }
         p=getgateptr(gatename);
         p->model=m;
         p->inum=m->inum;
         p->onum=m->onum;
         if ((argc - m->inum - m->onum)==3) /* User define time delay */
           {  /* Force time delay minimum 1 ns */
              p->td= ( (p->td=atoi(argv[argc-1])) > 0 ? p->td : 1 );
           }
         else p->td=m->td;  /* If not def use default time delay */
         index=2;
         /* init all fanin */
         for(i=0;i<p->inum;i++)
           { p->in[i]=getnodeptr(argv[index++] ); }
         /* init all fanout */
         for(i=0;i<p->onum;i++)
           { p->out[i]=getnodeptr(argv[index++]); }
         p->ps=0; /* Present state is 0 */
     }
     else printf("<GATENAME> <MODELS> <IN 1..n> <OUT 1..n> [delay]\n");
}

int   looktype(name,model)
char *name;
void **model;
{
       void *m;
       if (( m=(void *)getmodelptr(name) )!=NULL)
         {
           *model=m;  return(NORMAL_MODEL);
         }
       if (( m=(void  *)getmacroptr(name)) !=NULL)
         {
            *model=m; return(MACRO_MODEL);
         }
       *model=NULL;
       return(GARBAGE_MODEL);
}

void  editgate(argc,argv)
```

```
int argc;
char *argv[];
{
     void *model;
    char modelname[10];

    /* Truncate name to IDENLEN */
     strtrunc(modelname,argv[1],IDENLEN);
     switch(looktype(modelname,&model) )
      {
        case NORMAL_MODEL   :
                        normalgate(argc,argv,(LOGICMODEL *)model);
                        break;
        case MACRO_MODEL    :
                        expandmacro(argc,argv,(MACRO *) model);
                        break;
      }
}
/*---  END  OF  EDIT  FUNCTION ---*/

/*-------------------*
    File Handling
 *-------------------*/
void  lapread();
void  lapwrite();

void  lapfile(command,buff)
int command;
CMDFILE *buff;
{
       switch(command)
       {
        case  READ :    lapread(buff);
                        break;
        case  WRITE :   lapwrite(buff);
                        break;
        default     :
            lapperr(INVALIDCMD);
        }
}
/*-----------------------*
    File reading function
 *-----------------------*/
PUBLIC int fetchstr(st)
char *st;
/* This function fetch a string from file return NULL if end of file */
{
   IMPORT FILE *network;
    *st=NULL;  /* first set string to NULL string */
   if (feof(network)==0)
     {/* Read file 1 line */
       fgets(st,200,network);
       strupr(st);
       return(1);
     }
    return(0);
}
LOCAL  void  lapread(buff)
CMDFILE *buff;
{
```

```
    CMDFILE *cmd;

    static  char argb[20][20];
    static  char *argv[20];
    static  int  argc;
    int i;

    cmd=(CMDFILE *)buff;
    for(i=0;i<20;i++) argv[i]=argb[i]; /* SET BUFFER */
    if (( network=fopen(cmd->fname,"r"))==NULL)
      {
        printf("FILE NOT FOUND!!\n");
        return;
      }
   printf("READ FILE: %s\n",cmd->fname);
   while( fetchstr(line) )
   {
       printf("%s",line);
       if ( *line !=COMMENTCHAR )      /* Skip comment line */
        {
          argc=strargv(line,argv);
          editmain(argc,argv);
        }
    }
  fclose(network);
}

/*-------------------------*
   F I L E   W R I T I N G
 *-------------------------*/

#ifdef  IBMPC
LOCAL void elmwrite(FILE *fp,char *name);
LOCAL void nodewrite(FILE *fp);
LOCAL void clkwrite(FILE *fp);
#endif

LOCAL  void  lapwrite(cmdptr)
CMDFILE *cmdptr;
{
   CMDFILE *cmd;
   int i;
   FILE *fp;

   cmd=(CMDFILE *)cmdptr;

   if (( fp=fopen(cmd->fname,"w"))==NULL)
     {
       printf("FILE OPEN ERROR!\n");
       return;
     }
   puts("*** CIRCUIT NODE STATE WRITING ***");
/*
   elmwrite(fp,cmd->fname);
   clkwrite(fp);
*/
   nodewrite(fp);

   fclose(fp);
}
```

```
/*-----------------------------------------*
   elmwrite and clkwrite are not used now
 *-----------------------------------------*/
void elmwrite(fp,name)
FILE *fp;
char *name;
{
   LOGICELEMENT *p;
   register i;

   fprintf(fp,"%8s%8s\n",name,"CIRCUIT");
   LOOPLIST(p,elmhead)
   {
    fprintf(fp,"%8s%8s",p->name,p->model->name);
    for(i=0;i <p->inum;i++) fprintf(fp,"%*s",IDENLEN+1,p->in[i]->name);
    for(i=0;i <p->onum;i++) fprintf(fp,"%*s",IDENLEN+1,p->out[i]->name);
    fprintf(fp,"%*d\n",IDENLEN,p->td);
    }
}

void nodewrite(fp)
FILE *fp;
{
  LOGICNODE *node;

  fprintf(fp,"*--------------------*\n");
  fprintf(fp,"*    Logic state     *\n");
  fprintf(fp,"*--------------------*\n");
  LOOPLIST(node,nodehead)
   {
     if (node->type==XTERN) {
     fprintf(fp,"%8s%8s%6c\n",node->name,"NODE",logictoc(node->state)); }
   }
}

void clkwrite(fp)
FILE *fp;
{
  LOGICSOURCE *s;
  register i;

  LOOPLIST(s,srchead)
    {
     fprintf(fp,"%8s",s->name);
     if (s->stype==CLKSRC)
       {
          fprintf(fp,"%8s%8s%4d%4d\n",
                "CLOCK",s->outnode->name,s->time[1],s->time[0]);
        /* writing Ton before Toff */
       }
     else {
     fprintf(fp,"%*s%*s%4d\n",IDENLEN+1,"SIGNAL",IDENLEN+1,
                              s->outnode->name,s->pnum);
     for(i=0;i<s->pnum;i++) {
          fprintf(fp,"%5d%5c\n",s->time[i],logictoc(s->state[i]));
         }
      }
    }  /* END loop source */
}
```

```
/*
 *  MODULE:    LAPDATA
 *  AUTHOR:    P.UTHAYOPAS
 *  DATE:
 *
 *      This module contain routine that handle linklist
 *  of circuit data.
 */
#include <stdio.h>
#include <string.h>
#include <alloc.h>
#include "lap.h"
#include "lapdata.h"
#include "lapeval.h"
#include "laputil.h"
#include "lapmacro.h"

/*
 * Internal use function
 */
#ifdef  IBMPC
LOCAL  void freegate(void);
LOCAL  void freenode(void);
LOCAL  void freesrc(void);
LOCAL  int  addsrc(LOGICSOURCE *p);
#else
  #ifdef  UNIX
LOCAL  void freegate();
LOCAL  void freenode();
LOCAL  void freesrc();
LOCAL  int  addsrc();
  #endif
#endif

PUBLIC  LIST  buf1;
PUBLIC  LIST  buf2;
PUBLIC  LIST  buf3;
PUBLIC  LIST  *elmhead = &buf1;   /* head of element link list */
PUBLIC  LIST  *nodehead= &buf2;   /* head of node    link list */
PUBLIC  LIST  *srchead = &buf3;   /* head of source  link list */

LOGICNODE  *vccptr,*gndptr;

LOCAL  void initelm()
{
/* Init element list */
    LENGTH(elmhead)=0;
    TAIL(elmhead)=NULL;
    HEAD(elmhead)=NULL;
}

LOCAL void initnode()
{
/*   Init node list    */
/*  puts("Init node link list"); */
    LENGTH(nodehead)=0;
    TAIL(nodehead)=nodehead;
    HEAD(nodehead)=NULL;
}
```

```
LOCAL void initsrc()
{
/* Init source list  */
/*    puts("Init source link list");  */
    LENGTH(srchead)=0;
    TAIL(srchead)=srchead;
    HEAD(srchead)=NULL;
}

LOCAL void initrefnode()
{    LOGICNODE *p;
     /* Initial vcc */
/*        puts("Init Vcc and GND"); */
        p=vccptr;
        p->inum=p->onum=0;
        p->state=HIGH;        /* set initial state of node */
        p->outstate[0]=LOW; /* set initial for op-table */
     /* Initial ground */
        p=gndptr;
        p->inum=p->onum=0;
        p->state=LOW;        /* set initial state of node */
        p->outstate[0]=LOW; /* set initial for op-table */
}
PUBLIC void makerefnode() /* CALL ONCE WHEN PROGRAM START */
{
  LOGICNODE *p;

/*      puts("Make reference node");  */
     /* Initialize VCC */
        if ( (p=malloc(sizeof(LOGICNODE))) ==NULL )
          { lapperr(MEMFAULT); exit(MEMFAULT); }
        p->in =(LOGICELEMENT **)calloc(MAXFANIN,sizeof(LOGICELEMENT *) );
        p->out=(LOGICELEMENT **)calloc(MAXREFOUT,sizeof(LOGICELEMENT *) );
        strcpy(p->name,"VCC"); /* set VCC name */
     /* Set out put state table */
        p->outstate=(LOGICSTATE *)calloc(MAXTIME,sizeof(LOGICSTATE) );
        vccptr=p;
     /* Initialize GND */
        p=(LOGICNODE *) malloc(sizeof(LOGICNODE));
        p->in =(LOGICELEMENT **)calloc(MAXFANIN,sizeof(LOGICELEMENT *) );
        p->out=(LOGICELEMENT **)calloc(MAXREFOUT,sizeof(LOGICELEMENT *) );
        strcpy(p->name,"GND"); /* set VCC name */
     /* Set out put state table */
        p->outstate=(LOGICSTATE *)calloc(MAXTIME,sizeof(LOGICSTATE) );
        gndptr=p;
}

PUBLIC void initdata()
{
   initelm();
   initnode();
   initsrc();
   initmacro();
   initrefnode();
}

/*--------------------*
   Clear data storage
 *--------------------*/
void freegate()
```

```
{
  LOGICELEMENT *p;
/*  printf("free gate\n");  */
  LOOPLIST(p,elmhead)
    {
       free(p);  /* free this element */
    }
}
void freenode()
{
  LOGICNODE *p;
/*  printf("free node\n"); */
  LOOPLIST(p,nodehead)
   {
     /* free output table */
     if (p->outstate !=NULL) free(p->outstate);
     free(p->in);  /* Free fanin table */
     free(p->out); /* Free fanout table */
     free(p);
   }
}
void freesrc()
{
  LOGICSOURCE *p;
/*  printf("free source\n");  */
  LOOPLIST(p,srchead)
     free(p);
}
PUBLIC int addelm(p)
LOGICELEMENT *p;
{
    LOGICELEMENT *last;

    last=(LOGICELEMENT *)TAIL(elmhead);
    p->next=NULL;
    if ( HEAD(elmhead) == NULL)  HEAD(elmhead) = p;
       else  last->next=p;
    TAIL(elmhead)=p;
    return(++LENGTH(elmhead));
}
PUBLIC int addnode(p)
LOGICNODE *p;
{
    LOGICNODE *last;

    last=(LOGICNODE *)TAIL(nodehead);
    p->next=NULL;
    if ( HEAD(nodehead)==NULL) HEAD(nodehead)=p;
      else last->next=p;
    TAIL(nodehead)=p;
    return(++LENGTH(nodehead));
}
LOCAL int addsrc(p)
LOGICSOURCE *p;
{
    LOGICSOURCE *last;

    last=(LOGICSOURCE *)TAIL(srchead);
    p->next=NULL;
    if ( HEAD(srchead)==NULL) HEAD(srchead)=p;
```

```
      else last->next=p;
    TAIL(srchead)=p;
    return(++LENGTH(srchead));
}

/*-------------------------------------
   Element existance checking
   for logicelement,node,logic source
   return: pointer to object if exist
           NULL if not found
 *-------------------------------------*/
PUBLIC LOGICELEMENT *oldgate(name)
char *name;
{
   LOGICELEMENT *p;
   for(p=(LOGICELEMENT *)HEAD(elmhead);
       (p != NULL) && (strcmp(p->name,name) !=0 );p=p->next) ;
     return(p);
}

PUBLIC LOGICSOURCE *oldsrc(name)
char *name;
{
   LOGICSOURCE *p;
    for(p=(LOGICSOURCE *) HEAD(srchead);
        (p !=NULL) && (strcmp(p->name,name) != 0);p=p->next);
    return(p);
}
PUBLIC LOGICNODE *oldnode(name)
char *name;
{
   LOGICNODE *p;
    for(p=(LOGICNODE *)HEAD(nodehead);
        (p !=NULL) && (strcmp(p->name,name) != 0);p=p->next);
    return(p);
}
/*-----------------------------------*
    Data allocation and retrieval
 *-----------------------------------*/
PUBLIC LOGICELEMENT *makenewgate(st)
char *st;
{       LOGICELEMENT *p;
        if ((p=malloc(sizeof(LOGICELEMENT)) )==NULL)
         { lapperr(MEMFAULT);  exit(MEMFAULT); }
        strcpy(p->name,st);
        return(p);
}
 /*  FUNC: getgateptr()
    USE:  get pointer to gate if exist,else allocate
          storage for new gate.
    RET:  pointer to LOGICELEMENT
 */
PUBLIC LOGICELEMENT *getgateptr(st)
char *st;
{
    LOGICELEMENT *p;
    int n;

    /* Allocate  new gate */
    p=makenewgate(st);
```

```
    p->type=XTERN;
    addelm(p);
    return(p);
}

PUBLIC LOGICNODE *makenewnode(st)
char *st;
{
        LOGICNODE *p;
        if ((p=malloc(sizeof(LOGICNODE)))==NULL)
          { lapperr(MEMFAULT); exit(MEMFAULT); }
        p->in =(LOGICELEMENT **)calloc(MAXFANIN,sizeof(LOGICELEMENT *) );
        p->out=(LOGICELEMENT **)calloc(MAXFANOUT,sizeof(LOGICELEMENT *) );
        strcpy(p->name,st); /* set node name */
        p->inum=p->onum=0;
        p->state=LOW;       /* set initial state of node */
        return(p);
}
/*
    FUNC: getnodeptr()
    USE:  get pointer to node if exist,else allocate
          storage for new node and return pointer.
    RET:  pointer to LOGICNODE
 */
PUBLIC LOGICNODE *getnodeptr(st)
char *st;
{
    LOGICNODE *p;
    int n;

    /* Reference node */
    if ( strcmp("VCC",st)==0 ) { return(vccptr); }
    if ( strcmp("GND",st)==0 ) { return(gndptr); }
    /* Normal node */
    p=oldnode(st);
    if (p==NULL) then /*  node not exist */
      {
        p=makenewnode(st);
        p->type=XTERN;
        p->outstate=(LOGICSTATE *)calloc(MAXTIME,sizeof(LOGICSTATE) );
        p->outstate[0]=0; /* set initial 0 */
        addnode(p);
      }
     return(p);
}
/*  FUNC: getsrcptr()
    USE:  get pointer to LOGICSOURCE if exist,else allocate
          storage for new source and return pointer.
    RET:  pointer to LOGICSOURCE
 */
PUBLIC LOGICSOURCE *getsrcptr(st)
char *st;
{
    LOGICSOURCE *p;
    int n;

    p=oldsrc(st);
    if (p==NULL) then /*  source not exist */
      {  /*  Create new source */
         if ((p=malloc(sizeof(LOGICSOURCE)))==NULL)
```

```
        { lapperr(MEMFAULT); exit(MEMFAULT); }
        strcpy(p->name,st); /* set source name */
        addsrc(p);
      }
    return(p);
}

PUBLIC void setnode(nodename,s)
char *nodename;
char *s;
{
    LOGICNODE *node;
    LOGICSTATE state;

    state=tologic(*s);
    if ( ! strcmp(nodename,"ALL") ) then
      {
        LOOPLIST(node,nodehead)
         {
            printf("SET NODE:%8s to %5s\n",node->name,s);
            node->state=state;
         }
        return;
      }
    else
      {
         if ( (node=oldnode(nodename)) != NULL)
          then {
             node->state=state;
             printf("SET NODE:%8s to %5s\n",node->name,s);
               }
            else   puts("INVALID NODE NAME !");
         return;
      }
}
```

```
/*-------------------------
   Circuit element listing
 *-------------------------*/
#include <stdio.h>
#include <string.h>
#include <alloc.h>
#include "lap.h"
#include "lapdata.h"
#include "laputil.h"

int     listgate(void);
int     listnode(void);
int     listclk(void);
void    listfio(void);

PUBLIC void  listmain(int argc.char *argv[])
{

static CMDTABLE  listab[]={
                        {"GATE",1},
                        {"SOURCE",2},
                        {"SIGNAL",2},
                        {"FIO" ,3},
                        {"NODE",4},
                        {"MODEL",5},
                        {"MACRO",6}
                          };
static int      listablen = sizeof(listab)/sizeof(CMDTABLE);
  int srcnum,gatenum,code;
  char cmdstr[10];

  if (argc >2 ) { lapperr(SYNTAXERR); return; }
  if (argc==1)
  {
    gatenum=listgate();
    srcnum =listclk();
    printf("\nTOTAL GATE:%4d SIGNAL:%4d\n",gatenum,srcnum);
    return;
  }
  strtrunc(cmdstr,argv[1],IDENLEN);
  code=findcode(cmdstr,listab,listablen);
  switch(code)
   {
     case 1 :   gatenum=listgate();
                printf("\nTOTAL GATE:%d\n",gatenum);
                break;
     case 2 :   srcnum=listclk();
                printf("\nTOTAL SIGNAL:%d\n",srcnum);
                break;
     case 3 :   listfio();
                break;
     case 4 :   listnode();
                break;
     case 5 :   listmodel();
                break;
     case 6 :   listmacro();
                break;
   }
}
```

```
/*-----------------------------------------*
    FIOLIST-Fanin fanout listing for all node
 *-----------------------------------------*/
LOCAL void listfio(void)
{
  LOGICNODE *p;
  int i;

  LOOPLIST(p,nodehead)
   {
      if (p->type==XTERN)
      {
      printf("NODE: %s\n",p->name);
      for(i=0;i < p->inum;i++)  /* set fanin   */
        { printf("\t\tFAN  IN# %2d->GATE:%s\n",i+1,p->in[i]->name); }
      for(i=0;i < p->onum;i++)   /* set fanout */
        { printf("\t\tFAN OUT# %2d->GATE:%s\n",i+1,p->out[i]->name); }
      }
   }
}

LOCAL int listgate(void)
{
  LOGICELEMENT *p;
  int i;

   LOOPLIST(p,elmhead)
   {
/*    if (p->type ==XTERN)      */
     {
    printf("%8s%8s%6d ns ",p->name,p->model->name,p->td);
    for(i=0;i <p->inum;i++) printf("%8s",p->in[i]->name);
    for(i=0;i <p->onum;i++) printf("%8s",p->out[i]->name);
    printf("\n");
      }
    }
  return(elmhead->num);
}

LOCAL int listnode(void)
{
  LOGICNODE *p;
  printf("%9s%9s\n","NODE","STATE");
  LOOPLIST(p,nodehead)
  {
    if (p->type ==XTERN) /* list only external node */
    printf("%9s%9c\n",p->name,logictoc(p->state));
  }
  return(nodehead->num);
}

LOCAL int listclk(void)
{
  LOGICSOURCE *p;
  int i;

  LOOPLIST(p,srchead)
    {
     printf("%8s",p->name);
     if (p->stype==CLKSRC)
```

```
      {
       printf("%8s","CLOCK");
       printf("%8s  ",p->outnode->name);
       printf("Ton:%-4dns Toff:%-4dns\n",p->time[1],p->time[0]);
      }
     else {
     printf("%8s","SIGNAL");
     printf("%8s\n",p->outnode->name);
     for(i=0;i<p->pnum;i++)
        printf("%8d%8c\n",p->time[i],logictoc(p->state[i]));
      }
    }
  return(srchead->num);
}
```

```
/*
 *  MODULE: LAPSIM
 *  AUTHOR: P.Uthayopas
 *
 *      This  module perform simulation function.
 */
#include        <stdio.h>
#include        <alloc.h>
#include        <string.h>
#include        "lap.h"
#include        "lapcmd.h"
#include        "lapdata.h"
#include        "lapeval.h"

/*
 *  Local data structure for time queue
 *  use only in this module.
 */
typedef    struct tqnodetag   TQNODE;
typedef    struct timequeue   TIMEQUEUE;
struct tqnodetag {
                   LOGICNODE   *node;
                   LOGICSTATE  state;  /* For evaluation */
                   BYTE        status; /* For state recording */
                   struct tqnodetag *next;
                 };

struct timequeue{
         TQNODE  *first,*last;
         int     num;
       } ;
TIMEQUEUE   *timeq;
/*
 *   PUBLIC FUNCTION
 */
PUBLIC void simulator(CMD_SIM *simcmd);
/*
 *  INTERNAL FUNCTION
 */

/*
 * 1. time queue data management
 */
LOCAL void  add_to_tq(  LOGICNODE *node,
                        LOGICSTATE newstate,
                        BYTE status,
                        int mrt);

/*
 * 2. simulation routine
 */
LOCAL  void  sch_node(TQNODE *p,int mrt);
LOCAL  void  sch_src(int mrt);
LOCAL  void  dupoldstate(int mrt);
LOCAL  void  simulate(CMD_SIM *cmdptr);
/*
 *  Share variable
 */
PUBLIC  int             mrt;            /* Simulation time step    */
PUBLIC  int             usrtime;        /* Real time  from user    */
PUBLIC  BOOLEAN         isnewsim=TRUE;  /* True if new simulation  */
```

```
/*
 *    Time queue Handler
 */

/*
 *  add TQNODE into link list
 */
LOCAL void  addtqnode(int mrt,TQNODE *p)
{
   ++timeq[mrt].num;  /* Increment event count */
   p->next=NULL;
   if (timeq[mrt].first==NULL) /* Is first TQNODE ? */
      timeq[mrt].first=p;
     else  timeq[mrt].last->next=p;
   timeq[mrt].last=p;
}
/*
 *  get_tqdata  : pop data out of time queue
 *  RETURN:  Ptr to TQNODE
 *           NULL if queue empty
 */
LOCAL TQNODE  *get_tqnode(int time)
{
    TQNODE *p;
    if ( (p=timeq[time].first) !=NULL) {
          --timeq[time].num;
           timeq[time].first=p->next; }
    return(p);
}
LOCAL int get_tqlen(int mrt)
{
     return(timeq[mrt].num);
}
/*
 *  Create TQNODE and add in to time queue
 */
void  add_to_tq(LOGICNODE  *node,LOGICSTATE newstate,BYTE status,int mrt)
{
   TQNODE  *p;
   if ( mrt < MAXTIME)
   {
#if defined(DEBUGSIM)
   printf("TQADD node=%s t=%d st=%d\n",node->name,mrt,newstate);
#endif
   p=(TQNODE *)malloc(sizeof(TQNODE) ); /* MAKE TQNODE */
   p->node=node;                        /* INIT VALUE  */
   p->state=newstate;
   p->status=status;     /* Use for display only */
   addtqnode(mrt,p);
   }
}

void  clear_tq()
{
    register  i;
   /* clear time queue control */
    for(i=0;i< MAXTIME;i++)
       { timeq[i].num=0; timeq[i].first=timeq[i].last=NULL; };
}
```

```
void  create_tq()
{
   /* Create time queue call once when loading program */
   timeq=calloc(MAXTIME,sizeof(TIMEQUEUE));
}
/*--- END TIME QUEUE HANDLER ---*/

/*-------------------------*
    Display data handler
 *-------------------------*/
void  record_state(int mrt)
{
  LOGICNODE *p;

  LOOPLIST(p,nodehead)  /* Scan all node */
    { /* Record state only external node */
      if (p->type==XTERN) p->outstate[mrt]=p->state;
    }
}
/*-----------------------------*
     S I M U L A T I O N
 *-----------------------------*/

/*
 *       kick_node
 *  This routine schedule all node in time queue
 *  when simualtion start.
 *
 */
void kick_node(int time)
{
  LOGICNODE *p;

  /* on simulation start  schdule all node in time queue */
  LOOPLIST(p,nodehead)  /* Scan all node 1 time*/
   {
        add_to_tq(p,p->state,EVAL_NORMAL,time);
   }
}
/*
 *                  sch_node
 *   When accept time queue node,this routine scan
 *   all output node and calling evaluation function
 *   for each gate that it drive.Then,check output
 *   change with algorithm used and schedule output
 *   that will change in the future into timequeue.
 */
void sch_node(TQNODE *p,int mrt)
{
static LOGICSTATE optab[MAXOPIN]; /* Buffer for output table */
static LOGICSTATE iptab[MAXIPIN]; /* Buffer for input table  */

   LOGICNODE *node,*gnode;
   LOGICELEMENT *gate;
   LOGICSTATE   prestate,newstate;
   int fanout,fanin,outpin,newtime;
   int i;
   int err_stat;  /*  error status return from logic evaluation module */
   BYTE  status;  /*  Logic status for display                         */
```

```
    node=p->node;
    /* Update real node in circuit */
    node->state=p->state;
    if (node->simcount>0)  { --(node->simcount); }
    for(fanout=0;fanout< node->onum;fanout++) /* Scan all fanout */
     {
        gate=node->out[fanout];
        /* FIND LOGIC OUTPUT */
        /* Prepare input vector */
        for(fanin=0;fanin < gate->inum;fanin++)
           { iptab[fanin]=gate->in[fanin]->state; }
        err_stat=findlogic(gate,iptab,optab);
        status=0; /* Node evaluation status */
        if (err_stat==BUS_CONTENSION) {
            printf("BUS CONTENSION NODE=%6s,TIME=%5d\n",
                    gate->out[0]->name,usrtime);
            status=BUS_CONTENSION;
          }
        if (err_stat==SEQN_NO_TRIG) /* This element is block */
          return; /* No further processing */
        /* Scan all element output */
        for( outpin=0;outpin < gate->onum;outpin++)
         {
            gnode=gate->out[outpin];
            newtime=mrt+gate->td;
            prestate=gnode->state;
            newstate=optab[outpin];
            if (gnode->simcount >0 ) then
             {
                printf("* SPIKE * TIME:%4d NODE:%s\n",usrtime,gnode->name);
                ++(gnode->simcount);
                status |=SPIKE_DETECT;
                add_to_tq(gnode,newstate,status,newtime);
                return;
             }
            /* case Simcount  is 0 */
            if (prestate !=newstate)  /* if state change */
               {
                 ++(gnode->simcount);
                 add_to_tq(gnode,newstate,status,newtime);
                }
             else {
#if  defined(DEBUGSIM)
                    printf("BLOCKED node=%s\n",gnode->name);
#endif
                }
         }
     }
}
/*
 *              sch_src
 *    Check and process logic source in network.
 *
 */

void  sch_src(int mrt)
{
   LOGICSOURCE *p;
   LOGICSTATE  newstate,prestate;
   LOGICNODE   node;
```

```
   /* for all source */
   LOOPLIST(p,srchead)
      {
      if (p->tc >= p->time[p->index])
         {
           p->tc=0;
           ++(p->index);
           if ( p->index >= p->pnum) p->index=0;
         }
      newstate=p->state[p->index];
      if  ( p->state[p->index] != p->outnode->state )  /* STATE CHANGE */
           add_to_tq(p->outnode,newstate,EVAL_NORMAL,mrt);
      ++(p->tc);
   }
}
/*
 *          simulate
 *   Get command from simulation main and simulate
 *   circuit operation.
 *
 */
void  simulate(CMD_SIM *cmdptr)
{
   int i,tqlen,fullslot;
   TQNODE *p;

   printf("** SIMULATOR **\n\n");
   printf("SIMULATION START TIME:%5d LENGTH:%5d\n",
            usrtime,cmdptr->timelen);
   mrt=0;
   fullslot=0; /* Active time queue slot counter */
   /* Power on add VCC and GND to time queue */
   if ( isnewsim==TRUE)
    {   puts("** POWER ON **");
        add_to_tq(vccptr,HIGH,EVAL_NORMAL,mrt);
        add_to_tq(gndptr,LOW,EVAL_NORMAL,mrt);
    }
    /* schdule node with no fanin into time queue */
    kick_node(mrt);
    do
    {
#if defined(DEBUGSIM)
      printf("MRT:%3d TIME:%d\n",mrt,usrtime);
#endif
      sch_src(mrt);  /* Scheduling all source change */
      if ( (tqlen=get_tqlen(mrt)) !=0) {
         ++fullslot;
#if defined(DEBUGSIM)
         printf("MRT=%03d TIME QUEUE LENGTH=%4d\n",mrt,tqlen);
#endif
         }
      while ( ( p=get_tqnode(mrt)) !=NULL)   /* do while queue not empty */
        {
#if  defined(DEBUGSIM)
          printf("do node:%8s time=%d\n",p->node->name,mrt);
#endif
          sch_node(p,mrt);
          free(p);
        }
      record_state(mrt);     /*  record state of all node */
```

```
        ++mrt;  ++usrtime;     /*  GOTO NEXT TIME          */
     } while (mrt < cmdptr->timelen);
    printf("\n** Simulation complete **\n");
    printf("Active time queue slot=%4d\n",fullslot);
}

/*
 *  set node fanin fanout to correct position in network
 */
 LOCAL void preprocess()
{
    LOGICELEMENT *p;
    LOGICSOURCE  *q;
    LOGICNODE *node;
    int i;

    LOOPLIST(node,nodehead) { /* for all node */
         node->inum=node->onum=0; /* CLEAR NODE FANIN-OUT */
         node->simcount=0;        /* Init simulation counter */
      }
    LOOPLIST(p,elmhead) { /*    for all gate     */
       p->cks=LOW;        /* state of clock now */
       for(i=0;i < p->inum;i++)  /* set fanin    */
          { node=p->in[i];  node->out[(node->onum)++]=p; };
       for(i=0;i < p->onum;i++)   /* set fanout */
          { node=p->out[i];  node->in[(node->inum)++]=p; };
     }
}

LOCAL void  simsetup(CMD_SIM *simcmd);
LOCAL void  simfirst(CMD_SIM *simcmd);
LOCAL void  simcont(CMD_SIM *cmd);

void  init_old_sim(void)
{
     LOGICNODE   *p;
     LOGICSOURCE *s;

     isnewsim=FALSE;
     clear_tq();
}
void  init_new_sim(void)
{
     register   i,j;
     LOGICNODE   *p;
     LOGICSOURCE *s;

   puts("** INITIALIZE NEW SIMULATION **");
   usrtime=0;
   isnewsim=TRUE;
    /* Initial value for all source */
    LOOPLIST(s,srchead)
    {
       s->index=0;
       s->tc=0;
    }
    clear_tq();
}
 /* Simulation control parameter */
LOCAL   CMD_SIM         simparam={MAXTIME,SIM_NEW};
```

```
/*-------------------------------------*
   simsetup  - setup simulation parameter
 *-------------------------------------*/
void  simsetup(CMD_SIM *simcmd)
{
    memcpy(&simparam,simcmd,sizeof(CMD_SIM) );
}
/*-----------------------------------
    SIMCONT continue old simulation
 *-----------------------------------*/
void  simcont(CMD_SIM *cmd)
{
   puts("*** CONTINUE SIMULATION ***");
   init_old_sim();
   simulate(cmd);
}
/*-------------------------------------*
     SIMFIRST   start new simulation
 *-------------------------------------*/
void  simfirst(CMD_SIM *simcmd)
{
   init_new_sim();
   preprocess();
   simulate(simcmd);
}
/*
 *      S I M U L A T O R
 *   This module is entry point module for simulation.
 */
PUBLIC void simulator(CMD_SIM *simcmd)
{
    extern int evalcount;
    evalcount=0;
    switch(simcmd->cmd)
      {
         case SIM_SETUP  :  simsetup(simcmd);      break;
         case SIM_NEW    :  simfirst(&simparam);   break;
         case SIM_CONT   :  simcont(&simparam);    break;
      }
   printf(" TOTAL GATE PROCESSED=%5d\n",evalcount);
}
```

```
/*-------------------------------------------------*
  LOGIC STATE EVALUATION MODULE FOR LAP 1.1

 NOTE: This evaluation function assumed that all
 build-in gate is TTL-Logic gate.All input
 with ZSTATE  (High-IMPEDANCE MODE ) was
 transform to HIGH according to our assumption.
 *-------------------------------------------------*/
#include <stdio.h>
#include <alloc.h>
#include <mem.h>
#include "lap.h"
#include "lapeval.h"
#include "lapdata.h"

PUBLIC int evalcount=0;  /* Number of calling to find logic function */
extern int modelnum;     /* Number of build in model */
extern LOGICMODEL  gatemodel[];  /* Array of build in model */
/*-------------------------*
   Logic models function
 *-------------------------*/
LOCAL struct listnode usrlisbuff={NULL,NULL,0},*usrlist=&usrlisbuff;

/* duplicate model */
PUBLIC LOGICMODEL *dupmodel(LOGICMODEL *p)
{
    LOGICMODEL *m;
    m=(LOGICMODEL *)malloc(sizeof(LOGICMODEL));
    memcpy(m,p,(unsigned)sizeof(LOGICMODEL));
    return(m);
}
PUBLIC void free_usr_model()
{
   LOGICMODEL *p;

   for(p=HEAD(usrlist);p !=NULL;p=p->next)
     free(p);
   HEAD(usrlist)=TAIL(usrlist)=NULL;  /* Set all to empty list */
   LENGTH(usrlist)=0;
}
PUBLIC void add_usr_model(LOGICMODEL *p)
{
    LOGICMODEL *last;

    last=TAIL(usrlist);
    p->next=NULL;
    if (HEAD(usrlist)==NULL) /* if this is first user define model */
        HEAD(usrlist)=p;
     else  last->next=p;
    TAIL(usrlist)=p;
    ++LENGTH(usrlist);
}

LOCAL LOGICMODEL *scan_in_model(char *st)
{
     LOGICMODEL *m;
     int i;
     for(i=0,m=gatemodel;i<modelnum;i++,m++)
       {
          if (!strcmp(st,m->name) ) return(m);
```

```
        }
      return(NULL); /* NOT FOUND */
}
LOCAL LOGICMODEL *scan_usr_model(char *st)
{
       LOGICMODEL *p;

       for(p=HEAD(usrlist);p !=NULL;p=p->next) {
          if (!strcmp(st,p->name) ) return(p); }
      return(NULL); /* NOT FOUND */
}

PUBLIC LOGICMODEL *getmodelptr(char *st)
{
   LOGICMODEL *m;
   if ( (m=scan_in_model(st))==NULL)
     {  /* If not build in,scan for usr define model */
        return( scan_usr_model(st) );
     }
   return(m);
}
PUBLIC int listmodel()
{
     LOGICMODEL *m;
     int i,usrnum;

     puts("\t\tUSER DEFINED MODELS");
     printf("No.  %10s %4s %4s %4s\n","NAME","IN","OUT","DELAY");
     for(i=0,m=HEAD(usrlist);m !=NULL;m=m->next)
       {
         printf("<%2d> %10s %4d %4d %4d\n",++i,m->name,m->inum,m->onum,m->td);
       }
     usrnum=i;
     puts("\t\tBUILD IN MODELS");
     printf("No.  %10s %4s %4s %4s\n","NAME","IN","OUT","DELAY");
     for(i=0,m=gatemodel;i<modelnum;i++,m++)
       {
         printf("<%2d> %10s %4d %4d %4d\n",i+1,m->name,m->inum,m->onum,m->td);
       }
      printf("\t\tTOTAL: %5d Models\n",modelnum+usrnum);
      return(modelnum+usrnum);
}
/*-------------------------*
   Logic element evaluation
 *-------------------------*/

/*-------------------------------------------------
   DOCOMB - Procedure used to evaluate
            Standard Full table combination gate
 *-------------------------------------------------*/
LOCAL void docomb(LOGICELEMENT *gate,LOGICSTATE *inpin,LOGICSTATE *outpin)
{
    unsigned register i,index;
    /* Input[0] is left most bit */
    for(index=i=0;i<gate->inum;i++)
        index = (index << 2 ) | inpin[i];
    for(i=0;i< gate->onum;i++)
        outpin[i]= *( gate->model->op[i] + index);
}
/*---------------------------------------*
```

```
   Tricky Evaluation of build-in model
   AND,OR,NAND,NOR,INVERTER,DRIVER
 *-------------------------------------*/
LOCAL void do_and(LOGICELEMENT *gate,LOGICSTATE *inpin,LOGICSTATE *outpin)
{
/*  AND gate evaluate use INPUT COUNT METHOD */

    unsigned register i;

    /* Scan all input */
    for(i=0;i<gate->inum;i++) {
      switch(inpin[i])
       {
         case LOW     :  outpin[0]=LOW;      return;
         case UNKNOWN :  outpin[0]=UNKNOWN;  return;
       }
      }
    outpin[0]=HIGH;
    return;
}

LOCAL void do_nand(LOGICELEMENT *gate,LOGICSTATE *inpin,LOGICSTATE *outpin)
{
    unsigned register i,xcount;
    /* Scan all input */
    for(i=0;i<gate->inum;i++) {
      switch(inpin[i])
       {
         case LOW     :  outpin[0]=HIGH;      return;
         case UNKNOWN :  outpin[0]=UNKNOWN;  return;
       }
      }
    outpin[0]=LOW;
    return;

}

LOCAL void do_or(LOGICELEMENT *gate,LOGICSTATE *inpin,LOGICSTATE *outpin)
{
    unsigned register i;

    for(i=0;i<gate->inum;i++) {
      switch(inpin[i])
       {
         case HIGH    :  outpin[0]=HIGH;      return;
         case UNKNOWN :  outpin[0]=UNKNOWN;  return;
       }
      }
    outpin[0]=LOW;
    return;
}

LOCAL void do_nor(LOGICELEMENT *gate,LOGICSTATE *inpin,LOGICSTATE *outpin)
{
    unsigned register i,state;

    for(i=0;i < gate->inum;i++) {
      switch(inpin[i])
       {
         case HIGH    :  outpin[0]=LOW;      return;
```

```
        case UNKNOWN  :  outpin[0]=UNKNOWN;   return;
       }
      }
    outpin[0]=HIGH;
    return;
}

/*---------------------------------*
    Tri-state bus checking routine
 *---------------------------------*/
LOCAL int do_bus(LOGICELEMENT *gate,LOGICSTATE *inpin,LOGICSTATE *outpin)
{
    unsigned register i,index;
    LOGICSTATE state;

    state=ZSTATE;
    /* Scan all input */
    for(index=0,i=0;i<gate->inum;i++) {
       if ( inpin[i] != ZSTATE) {
          state = inpin[i];
          ++index; }
     }
    if (index>1) /* Bus contension detect when index >1 */
      {
       outpin[0]=UNKNOWN; /* Force output state to UNKNOWN */
       return(BUS_CONTENSION); /* return error status */
       }
    /* If there are only one input active */
    outpin[0]=state;
    return(EVAL_NORMAL);
}
/*
 *   Flip-flop and sequential circuit
 */
LOCAL int do_flp(LOGICELEMENT *gate,LOGICSTATE *inpin,LOGICSTATE *outpin)
{
        register i,index;
        LOGICSTATE pr,clr;
        int inmax,clkpin;
        BYTE state;
        /*
         *   preset clear alway the 2 rightmost pin
         */
        inmax=gate->inum-2;
        clr=inpin[gate->inum-1];
        pr =inpin[gate->inum-2];

        /* PRESET,CLEAR Checking */
        if ((pr==UNKNOWN)||(clr==UNKNOWN) )
         {
           gate->ps=outpin[0]=outpin[1]=UNKNOWN;
           return(EVAL_NORMAL);
         }
        if (clr== LOW)
          { /* CLEAR ACTIVE */
           gate->ps=outpin[0]=LOW;
           outpin[1]=HIGH;
           return(EVAL_NORMAL);
           }
         if (pr == LOW )
```

```
        { /* If preset */
          gate->ps=outpin[0]=HIGH;
          outpin[1]=LOW;
          return(EVAL_NORMAL);
        }
        /* check CLK trigger */

         clkpin=gate->model->clkpin;
            /* check CLK trigger */
         if (gate->cks == inpin[clkpin]) /* Clock not trig */
          { return(SEQN_NO_TRIG); }
          else
           {
             gate->cks=inpin[clkpin];/* store clock state */
           }
         /*
          *   normal flip-flop  logic finding
          */
        for(index=i=0;i< inmax;i++) {
          index = (index << 2 ) | inpin[i]; }
        index= index | (gate->ps << inmax*2);
        gate->ps=  *( gate->model->state + index );
        outpin[0]= *( (gate->model->op[0]) + index);
        outpin[1]= *( (gate->model->op[1]) + index);
        return(EVAL_NORMAL);
}
LOCAL int do_seqn(LOGICELEMENT *gate,LOGICSTATE *inpin,LOGICSTATE *outpin)
{
        register index,i;
        int clkpin;

        clkpin=gate->model->clkpin;
           /* check CLK trigger */
        if (gate->cks == inpin[clkpin]) /* Clock not trig */
           { return(SEQN_NO_TRIG); }
           else
            {
              gate->cks=inpin[clkpin];/* store clock state */
            }
    /* Full table sequential circuit */
        for(index=i=0;i<gate->inum;i++)
           index = (index <<2 ) | inpin[i];
        /* MASK Present state */
        index= index | (gate->ps << (gate->inum*2) );
        /* Find next state */
           gate->ps = *(gate->model->state + index);
         for(i=0;i< gate->model->onum;i++)
            outpin[i]= *( (gate->model->op[i]) + index);
         return(EVAL_NORMAL);
}

PUBLIC int findlogic(LOGICELEMENT *gate,
                     LOGICSTATE   *inpin,
                     LOGICSTATE   *outpin)
{       int status;
        register i;

        evalcount++;
        status=EVAL_NORMAL;
```

```
        if (gate->model->mmodel !=BUS)
        for(i=0;i<gate->inum;i++)
          { /* Convert input ZSTATE to HIGH */
             if (inpin[i]==ZSTATE ) inpin[i]=HIGH;
          }
/* Tricky type checking in order to apply dirty method to build in model */
        switch(gate->model->mmodel)
        {
           case COMB  : /* first,check build in combinational gate */
                        switch(gate->model->smodel)
                         {
                           case AND  :  do_and(gate,inpin,outpin);
                                        break;
                           case OR   :  do_or(gate,inpin,outpin);
                                        break;
                           case NAND :  do_nand(gate,inpin,outpin);
                                        break;
                           case NOR  :  do_nor(gate,inpin,outpin);
                                        break;
                           default   :  docomb(gate,inpin,outpin);
                         }
                        break;
           case FLPF  :
                        status=do_flp(gate,inpin,outpin);
                        break;
           case SEQN  :
                        status=do_seqn(gate,inpin,outpin);
                        break;
           case BUS   :
                        status=do_bus(gate,inpin,outpin);
                        break;
         }
        return(status);
}
```

```
#include <stdio.h>
#include "lap.h"
/*-----------------------------------------------*
   True table for build in element evaluation
 *-----------------------------------------------*/
/* STANDARD COMBINATIONAL ELEMENT */
  LOGICSTATE xortab[] =    {LOW,HIGH,UNKNOWN,HIGH,
                            HIGH,LOW,UNKNOWN,LOW,
                            UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                            HIGH,LOW,UNKNOWN,LOW};
 LOGICSTATE drvltab[]={ LOW,HIGH,UNKNOWN,ZSTATE,
                            ZSTATE,ZSTATE,ZSTATE,ZSTATE,
                            UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                            ZSTATE,ZSTATE,ZSTATE,ZSTATE};
 LOGICSTATE invtab[] ={HIGH,LOW,UNKNOWN,LOW};
 LOGICSTATE buftab[] ={LOW,HIGH,UNKNOWN,ZSTATE};

/* SEQUENTIAL ELEMENT */

/* Positive Edge trig flip-flop */
/*
 LOGICSTATE jkffq[]  ={0,0,0,0,0,1,0,1,1,1,1,0,1,1,1,0,1,0};
 LOGICSTATE jkffqb[] ={1,1,1,1,1,0,1,0,0,0,0,1,0,0,0,1,0,1};
 */
LOGICSTATE tfnq[]  =  {LOW,LOW,UNKNOWN,LOW,
                       HIGH,LOW,UNKNOWN,LOW,
                       UNKNOWN,LOW,UNKNOWN,LOW,
                       HIGH,LOW,UNKNOWN,LOW,
                       HIGH,HIGH,UNKNOWN,HIGH,
                       LOW,HIGH,UNKNOWN,HIGH,
                       UNKNOWN,HIGH,UNKNOWN,HIGH,
                       LOW,HIGH,UNKNOWN,HIGH,
                       UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                       UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                       UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                       UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN};

LOGICSTATE  tfnqb[]= {HIGH,HIGH,UNKNOWN,HIGH,
                      LOW,HIGH,UNKNOWN,HIGH,
                      UNKNOWN,HIGH,UNKNOWN,HIGH,
                      LOW,HIGH,UNKNOWN,HIGH,
                      LOW,LOW,UNKNOWN,LOW,
                      HIGH,LOW,UNKNOWN,LOW,
                      UNKNOWN,LOW,UNKNOWN,LOW,
                      HIGH,LOW,UNKNOWN,LOW,
                      UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                      UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                      UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                      UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN};
LOGICSTATE dfnq[]={
                          LOW,LOW,UNKNOWN,LOW,
                          HIGH,LOW,UNKNOWN,LOW,
                          UNKNOWN,LOW,UNKNOWN,LOW,
                          HIGH,LOW,UNKNOWN,LOW,
                          LOW,HIGH,UNKNOWN,HIGH,
                          HIGH,HIGH,UNKNOWN,HIGH,
                          UNKNOWN,HIGH,UNKNOWN,HIGH,
                          HIGH,HIGH,UNKNOWN,HIGH,
                          LOW,UNKNOWN,UNKNOWN,UNKNOWN,
                          HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
```

```
                        UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                        HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
                        LOW,HIGH,UNKNOWN,HIGH,
                        HIGH,HIGH,UNKNOWN,HIGH,
                        UNKNOWN,HIGH,UNKNOWN,HIGH,
                        HIGH,HIGH,UNKNOWN,HIGH
                    };
LOGICSTATE dfnqb[]={
                        HIGH,HIGH,UNKNOWN,HIGH,
                        LOW,HIGH,UNKNOWN,HIGH,
                        UNKNOWN,HIGH,UNKNOWN,HIGH,
                        LOW,HIGH,UNKNOWN,HIGH,
                        HIGH,LOW,UNKNOWN,LOW,
                        LOW,LOW,UNKNOWN,LOW,
                        UNKNOWN,LOW,UNKNOWN,LOW,
                        LOW,LOW,UNKNOWN,LOW,
                        HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
                        LOW,UNKNOWN,UNKNOWN,UNKNOWN,
                        UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                        LOW,UNKNOWN,UNKNOWN,UNKNOWN,
                        HIGH,LOW,UNKNOWN,LOW,
                        LOW,LOW,UNKNOWN,LOW,
                        UNKNOWN,LOW,UNKNOWN,LOW,
                        LOW,LOW,UNKNOWN,LOW
                    };
LOGICSTATE dffq[]={
                    LOW,LOW,UNKNOWN,LOW,
                    LOW,HIGH,UNKNOWN,HIGH,
                    LOW,UNKNOWN,UNKNOWN,UNKNOWN,
                    LOW,HIGH,UNKNOWN,HIGH,
                    HIGH,LOW,UNKNOWN,LOW,
                    HIGH,HIGH,UNKNOWN,HIGH,
                    HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
                    HIGH,HIGH,UNKNOWN,HIGH,
                    UNKNOWN,LOW,UNKNOWN,LOW,
                    UNKNOWN,HIGH,UNKNOWN,HIGH,
                    UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                    UNKNOWN,HIGH,UNKNOWN,HIGH,
                    HIGH,LOW,UNKNOWN,LOW,
                    HIGH,HIGH,UNKNOWN,HIGH,
                    HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
                    HIGH,HIGH,UNKNOWN,HIGH
};
LOGICSTATE  dffqb[]={
                    HIGH,HIGH,UNKNOWN,HIGH,
                    HIGH,LOW,UNKNOWN,LOW,
                    HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
                    HIGH,LOW,UNKNOWN,LOW,
                    LOW,HIGH,UNKNOWN,HIGH,
                    LOW,LOW,UNKNOWN,LOW,
                    LOW,UNKNOWN,UNKNOWN,UNKNOWN,
                    LOW,LOW,UNKNOWN,LOW,
                    UNKNOWN,HIGH,UNKNOWN,HIGH,
                    UNKNOWN,LOW,UNKNOWN,LOW,
                    UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                    UNKNOWN,LOW,UNKNOWN,LOW,
                    LOW,HIGH,UNKNOWN,HIGH,
                    LOW,LOW,UNKNOWN,LOW,
                    LOW,UNKNOWN,UNKNOWN,UNKNOWN,
                    LOW,LOW,UNKNOWN,LOW,
```

```
};
LOGICSTATE  tffq[]={
                LOW,LOW,UNKNOWN,LOW,
                LOW,HIGH,UNKNOWN,HIGH,
                LOW,UNKNOWN,UNKNOWN,UNKNOWN,
                LOW,HIGH,UNKNOWN,HIGH,
                HIGH,HIGH,UNKNOWN,HIGH,
                HIGH,LOW,UNKNOWN,LOW,
                HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
                HIGH,LOW,UNKNOWN,LOW,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                HIGH,HIGH,UNKNOWN,HIGH,
                HIGH,LOW,UNKNOWN,LOW,
                HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
                HIGH,LOW,UNKNOWN,LOW,
                };

LOGICSTATE  tffqb[]={
                HIGH,HIGH,UNKNOWN,HIGH,
                HIGH,LOW,UNKNOWN,LOW,
                HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
                HIGH,LOW,UNKNOWN,LOW,
                LOW,LOW,UNKNOWN,LOW,
                LOW,HIGH,UNKNOWN,HIGH,
                LOW,UNKNOWN,UNKNOWN,UNKNOWN,
                LOW,HIGH,UNKNOWN,HIGH,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                LOW,LOW,UNKNOWN,LOW,
                LOW,HIGH,UNKNOWN,HIGH,
                LOW,UNKNOWN,UNKNOWN,UNKNOWN,
                LOW,HIGH,UNKNOWN,HIGH,
                };
LOGICSTATE  jknq[]={
                LOW,LOW,UNKNOWN,LOW,
                LOW,LOW,UNKNOWN,LOW,
                UNKNOWN,LOW,UNKNOWN,LOW,
                LOW,LOW,UNKNOWN,LOW,
                HIGH,LOW,UNKNOWN,LOW,
                HIGH,LOW,UNKNOWN,LOW,
                UNKNOWN,LOW,UNKNOWN,LOW,
                HIGH,LOW,UNKNOWN,LOW,
                UNKNOWN,LOW,UNKNOWN,LOW,
                UNKNOWN,LOW,UNKNOWN,LOW,
                UNKNOWN,LOW,UNKNOWN,UNKNOWN,
                UNKNOWN,LOW,UNKNOWN,LOW,
                HIGH,LOW,UNKNOWN,LOW,
                HIGH,LOW,UNKNOWN,LOW,
                UNKNOWN,LOW,UNKNOWN,LOW,
                HIGH,LOW,UNKNOWN,LOW,
                HIGH,HIGH,UNKNOWN,HIGH,
                LOW,HIGH,UNKNOWN,HIGH,
                UNKNOWN,HIGH,UNKNOWN,HIGH,
                LOW,HIGH,UNKNOWN,HIGH,
                HIGH,HIGH,UNKNOWN,HIGH,
```

```
                LOW,HIGH,UNKNOWN,HIGH,
                UNKNOWN,HIGH,UNKNOWN,HIGH,
                LOW,HIGH,UNKNOWN,UNKNOWN,
                UNKNOWN,HIGH,UNKNOWN,HIGH,
                UNKNOWN,HIGH,UNKNOWN,HIGH,
                UNKNOWN,HIGH,UNKNOWN,HIGH,
                UNKNOWN,HIGH,UNKNOWN,UNKNOWN,
                HIGH,HIGH,UNKNOWN,HIGH,
                LOW,HIGH,UNKNOWN,HIGH,
                UNKNOWN,HIGH,UNKNOWN,HIGH,
                LOW,HIGH,UNKNOWN,HIGH,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                LOW,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                LOW,UNKNOWN,UNKNOWN,UNKNOWN,
                HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                HIGH,HIGH,UNKNOWN,HIGH,
                LOW,HIGH,UNKNOWN,HIGH,
                UNKNOWN,HIGH,UNKNOWN,HIGH,
                LOW,HIGH,UNKNOWN,HIGH,
                HIGH,HIGH,UNKNOWN,HIGH,
                LOW,HIGH,UNKNOWN,HIGH,
                UNKNOWN,HIGH,UNKNOWN,HIGH,
                LOW,HIGH,UNKNOWN,UNKNOWN,
                UNKNOWN,HIGH,UNKNOWN,HIGH,
                UNKNOWN,HIGH,UNKNOWN,HIGH,
                UNKNOWN,HIGH,UNKNOWN,HIGH,
                UNKNOWN,HIGH,UNKNOWN,UNKNOWN,
                HIGH,HIGH,UNKNOWN,HIGH,
                LOW,HIGH,UNKNOWN,HIGH,
                UNKNOWN,HIGH,UNKNOWN,HIGH,
                LOW,HIGH,UNKNOWN,HIGH
                };

LOGICSTATE      jknqb[]={
                HIGH,HIGH,UNKNOWN,HIGH,
                HIGH,HIGH,UNKNOWN,HIGH,
                UNKNOWN,HIGH,UNKNOWN,HIGH,
                HIGH,HIGH,UNKNOWN,HIGH,
                LOW,HIGH,UNKNOWN,HIGH,
                LOW,HIGH,UNKNOWN,HIGH,
                UNKNOWN,HIGH,UNKNOWN,HIGH,
                LOW,HIGH,UNKNOWN,HIGH,
                UNKNOWN,HIGH,UNKNOWN,HIGH,
                UNKNOWN,HIGH,UNKNOWN,HIGH,
                UNKNOWN,HIGH,UNKNOWN,UNKNOWN,
                UNKNOWN,HIGH,UNKNOWN,HIGH,
                LOW,HIGH,UNKNOWN,HIGH,
                LOW,HIGH,UNKNOWN,HIGH,
```

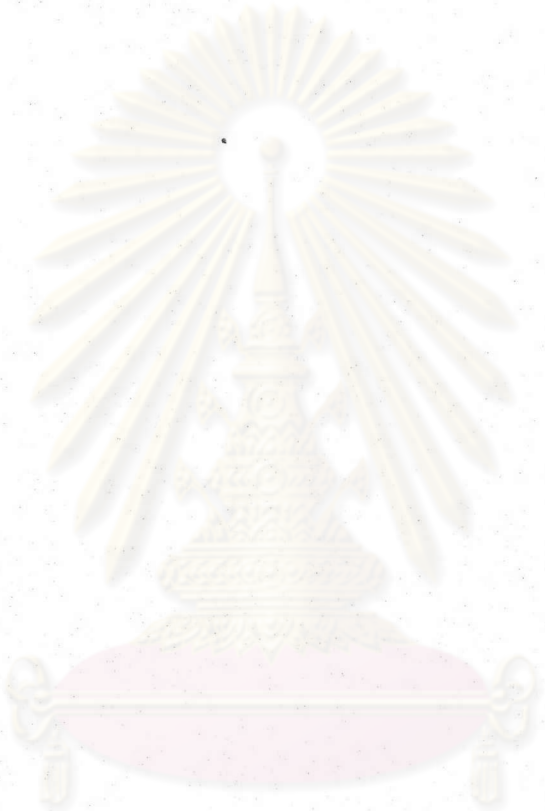
```
                UNKNOWN,HIGH,UNKNOWN,HIGH,
                LOW,HIGH,UNKNOWN,HIGH,
                LOW,LOW,UNKNOWN,LOW,
                HIGH,LOW,UNKNOWN,LOW,
                UNKNOWN,LOW,UNKNOWN,LOW,
                HIGH,LOW,UNKNOWN,LOW,
                LOW,LOW,UNKNOWN,LOW,
                HIGH,LOW,UNKNOWN,LOW,
                UNKNOWN,LOW,UNKNOWN,LOW,
                HIGH,LOW,UNKNOWN,UNKNOWN,
                UNKNOWN,LOW,UNKNOWN,LOW,
                UNKNOWN,LOW,UNKNOWN,LOW,
                UNKNOWN,LOW,UNKNOWN,LOW,
                UNKNOWN,LOW,UNKNOWN,UNKNOWN,
                LOW,LOW,UNKNOWN,LOW,
                HIGH,LOW,UNKNOWN,LOW,
                UNKNOWN,LOW,UNKNOWN,LOW,
                HIGH,LOW,UNKNOWN,LOW,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
                LOW,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                LOW,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
                LOW,LOW,UNKNOWN,LOW,
                HIGH,LOW,UNKNOWN,LOW,
                UNKNOWN,LOW,UNKNOWN,LOW,
                HIGH,LOW,UNKNOWN,LOW,
                LOW,LOW,UNKNOWN,LOW,
                HIGH,LOW,UNKNOWN,LOW,
                UNKNOWN,LOW,UNKNOWN,LOW,
                HIGH,LOW,UNKNOWN,UNKNOWN,
                UNKNOWN,LOW,UNKNOWN,LOW,
                UNKNOWN,LOW,UNKNOWN,LOW,
                UNKNOWN,LOW,UNKNOWN,LOW,
                UNKNOWN,LOW,UNKNOWN,UNKNOWN,
                LOW,LOW,UNKNOWN,LOW,
                HIGH,LOW,UNKNOWN,LOW,
                UNKNOWN,LOW,UNKNOWN,LOW,
                HIGH,LOW,UNKNOWN,LOW
                };

LOGICSTATE jkpq[]={
                LOW,LOW,UNKNOWN,LOW,
                LOW,LOW,UNKNOWN,LOW,
                LOW,UNKNOWN,UNKNOWN,UNKNOWN,
                LOW,LOW,UNKNOWN,LOW,
                LOW,HIGH,UNKNOWN,HIGH,
                LOW,HIGH,UNKNOWN,HIGH,
                LOW,UNKNOWN,UNKNOWN,UNKNOWN,
```

```
LOW,LOW,UNKNOWN,LOW,
LOW,UNKNOWN,UNKNOWN,UNKNOWN,
LOW,UNKNOWN,UNKNOWN,UNKNOWN,
LOW,UNKNOWN,UNKNOWN,UNKNOWN,
LOW,UNKNOWN,UNKNOWN,UNKNOWN,
LOW,HIGH,UNKNOWN,HIGH,
LOW,HIGH,UNKNOWN,HIGH,
LOW,UNKNOWN,UNKNOWN,UNKNOWN,
LOW,HIGH,UNKNOWN,HIGH,
HIGH,LOW,UNKNOWN,LOW,
HIGH,LOW,UNKNOWN,LOW,
HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
HIGH,LOW,UNKNOWN,LOW,
HIGH,HIGH,UNKNOWN,HIGH,
HIGH,LOW,UNKNOWN,LOW,
HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
HIGH,LOW,UNKNOWN,LOW,
HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
HIGH,HIGH,UNKNOWN,HIGH,
HIGH,LOW,UNKNOWN,LOW,
HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
HIGH,LOW,UNKNOWN,LOW,
UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
UNKNOWN,LOW,UNKNOWN,LOW,
UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
UNKNOWN,LOW,UNKNOWN,LOW,
UNKNOWN,HIGH,UNKNOWN,HIGH,
UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
UNKNOWN,HIGH,UNKNOWN,HIGH,
UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
HIGH,LOW,UNKNOWN,LOW,
HIGH,LOW,UNKNOWN,LOW,
HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
HIGH,LOW,UNKNOWN,LOW,
HIGH,HIGH,UNKNOWN,HIGH,
HIGH,LOW,UNKNOWN,LOW,
HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
HIGH,HIGH,UNKNOWN,HIGH,
HIGH,LOW,UNKNOWN,LOW,
HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
HIGH,LOW,UNKNOWN,LOW,
};

LOGICSTATE ikpqb[]={
```

HIGH,HIGH,UNKNOWN,HIGH,
HIGH,HIGH,UNKNOWN,HIGH,
HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
HIGH,HIGH,UNKNOWN,HIGH,
HIGH,LOW,UNKNOWN,LOW,
HIGH,LOW,UNKNOWN,LOW,
HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
HIGH,HIGH,UNKNOWN,HIGH,
HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
HIGH,LOW,UNKNOWN,LOW,
HIGH,LOW,UNKNOWN,LOW,
HIGH,UNKNOWN,UNKNOWN,UNKNOWN,
HIGH,LOW,UNKNOWN,LOW,
LOW,HIGH,UNKNOWN,HIGH,
LOW,HIGH,UNKNOWN,HIGH,
LOW,UNKNOWN,UNKNOWN,UNKNOWN,
LOW,HIGH,UNKNOWN,HIGH,
LOW,LOW,UNKNOWN,LOW,
LOW,HIGH,UNKNOWN,HIGH,
LOW,UNKNOWN,UNKNOWN,UNKNOWN,
LOW,HIGH,UNKNOWN,HIGH,
LOW,UNKNOWN,UNKNOWN,UNKNOWN,
LOW,UNKNOWN,UNKNOWN,UNKNOWN,
LOW,UNKNOWN,UNKNOWN,UNKNOWN,
LOW,UNKNOWN,UNKNOWN,UNKNOWN,
LOW,LOW,UNKNOWN,LOW,
LOW,HIGH,UNKNOWN,HIGH,
LOW,UNKNOWN,UNKNOWN,UNKNOWN,
LOW,HIGH,UNKNOWN,HIGH,
UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
UNKNOWN,HIGH,UNKNOWN,HIGH,
UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
UNKNOWN,HIGH,UNKNOWN,HIGH,
UNKNOWN,LOW,UNKNOWN,LOW,
UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
UNKNOWN,LOW,UNKNOWN,LOW,
UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
UNKNOWN,UNKNOWN,UNKNOWN,UNKNOWN,
LOW,HIGH,UNKNOWN,HIGH,
LOW,HIGH,UNKNOWN,HIGH,
LOW,UNKNOWN,UNKNOWN,UNKNOWN,
LOW,HIGH,UNKNOWN,HIGH,
LOW,LOW,UNKNOWN,LOW,
LOW,HIGH,UNKNOWN,HIGH,
LOW,UNKNOWN,UNKNOWN,UNKNOWN,
LOW,HIGH,UNKNOWN,HIGH,
LOW,UNKNOWN,UNKNOWN,UNKNOWN,
LOW,UNKNOWN,UNKNOWN,UNKNOWN,
LOW,UNKNOWN,UNKNOWN,UNKNOWN,
LOW,UNKNOWN,UNKNOWN,UNKNOWN.

```
                LOW,LOW,UNKNOWN,LOW,
                LOW,HIGH,UNKNOWN,HIGH,
                LOW,UNKNOWN,UNKNOWN,UNKNOWN,
                LOW,HIGH,UNKNOWN,HIGH,
                };

/*--------------------*
   DEFAULT GATE MODEL
 *--------------------*/

/*
 *  NOTE: all flip-flop use active low preset and clear input
 */.
PUBLIC LOGICMODEL gatemodel[]=  {
     { "AND2",AND,COMB,0,2,1,10,NULL ,NULL,NULL,NULL}, /* 74LS08 */
     { "AND3",AND,COMB,0,3,1,10,NULL ,NULL,NULL,NULL},
     { "AND4",AND,COMB,0,4,1,10,NULL ,NULL,NULL,NULL},
     { "AND5",AND,COMB,0,5,1,10,NULL ,NULL,NULL,NULL},
     { "AND6",AND,COMB,0,6,1,10,NULL ,NULL,NULL,NULL},
     { "AND7",AND,COMB,0,7,1,10,NULL ,NULL,NULL,NULL},
     { "AND8",AND,COMB,0,8,1,10,NULL ,NULL,NULL,NULL},

     { "OR2",OR,COMB,0,2,1,14,NULL,NULL,NULL,NULL},  /* 74LS32 */
     { "OR3",OR,COMB,0,3,1,14,NULL,NULL,NULL,NULL},
     { "OR4",OR,COMB,0,4,1,14,NULL,NULL,NULL,NULL},
     { "OR5",OR,COMB,0,5,1,14,NULL,NULL,NULL,NULL},
     { "OR6",OR,COMB,0,6,1,14,NULL,NULL,NULL,NULL},
     { "OR7",OR,COMB,0,7,1,14,NULL,NULL,NULL,NULL},
     { "OR8",OR,COMB,0,8,1,14,NULL,NULL,NULL,NULL},

     { "NAND2",NAND,COMB,0,2,1,10,NULL,NULL,NULL,NULL}, /* 74LS00 */
     { "NAND3",NAND,COMB,0,3,1,10,NULL,NULL,NULL,NULL},
     { "NAND4",NAND,COMB,0,4,1,10,NULL,NULL,NULL,NULL},
     { "NAND5",NAND,COMB,0,5,1,10,NULL,NULL,NULL,NULL},
     { "NAND6",NAND,COMB,0,6,1,10,NULL,NULL,NULL,NULL},
     { "NAND7",NAND,COMB,0,7,1,10,NULL,NULL,NULL,NULL},
     { "NAND8",NAND,COMB,0,8,1,10,NULL,NULL,NULL,NULL},

     { "NOR2",NOR ,COMB,0,2,1,10,NULL ,NULL,NULL,NULL}, /* 74LS02 */
     { "NOR3",NOR ,COMB,0,3,1,10,NULL ,NULL,NULL,NULL},
     { "NOR4",NOR ,COMB,0,4,1,10,NULL ,NULL,NULL,NULL},
     { "NOR5",NOR ,COMB,0,5,1,10,NULL ,NULL,NULL,NULL},
     { "NOR6",NOR ,COMB,0,6,1,10,NULL ,NULL,NULL,NULL},
     { "NOR7",NOR ,COMB,0,7,1,10,NULL ,NULL,NULL,NULL},
     { "NOR8",NOR ,COMB,0,8,1,10,NULL ,NULL,NULL,NULL},
/*  Standard combination evaluation element */
     { "INV" ,INV ,COMB,0,1,1,10,invtab ,NULL,NULL,NULL}, /* 74LS04 */
     { "XOR" ,XOR ,COMB,0,2,1,10,xortab ,NULL,NULL,NULL},
     { "BUFF",BUFF,COMB,0,1,1,10,buftab ,NULL,NULL,NULL},
     { "DRVL",DRIVER,COMB,0,2,1,10,drvltab,NULL,NULL,NULL},
     { "BUS" ,BUS   ,BUS ,0,20,1,10,NULL,NULL,NULL,NULL}, /* TRI-STATE BUS */
     /* Flip-flop with out preset and clear input */
     /* this model use standard sequential method to find logic */
     { "TFF"  ,TFF ,SEQN,1,2,2,15,tffq,tffqb,tffq,NULL},
     { "DFF"  ,DFF ,SEQN,1,2,2,25,dffq,dffqb,dffq,NULL},
     { "DFN"  ,DFF ,SEQN,1,2,2,25,dfnq,dfnqb,dfnq,NULL},
     { "TFN"  ,TFF ,SEQN,1,2,2,15,tfnq,tfnqb,tfnq,NULL},
     { "JKN" ,JKFF ,SEQN,2,3,2,15,jknq,jknqb,jknq,NULL},
     { "JKP" ,JKFF ,SEQN,2,3,2,15,jkpq,jkpqb,jkpq,NULL},
     /* Flip-flop with preset and clear input                    */
```

```
    /* use special subroutine to check preset and clear input */
    { "TFFC" ,TFF ,FLPF,1,4,2,15,tffq,tffqb,tffq,NULL},
    { "DFFC" ,DFF ,FLPF,1,4,2,25,dffq,dffqb,dffq,NULL},
    { "DFNC" ,DFF ,FLPF,1,4,2,25,dfnq,dfnqb,dfnq,NULL},
    { "TFNC" ,TFF ,FLPF,1,4,2,15,tfnq,tfnqb,tfnq,NULL},
    { "JKNC" ,JKFF,FLPF,2,5,2,15,jknq,jknqb,jknq,NULL},
    { "JKPC" ,JKFF,FLPF,2,5,2,15,jkpq ,jkpqb,jkpq,NULL}  /* 74LS76 */
                    };
/* Number of build in model */
PUBLIC int modelnum = (sizeof(gatemodel)/sizeof(LOGICMODEL) );
```



```
 /*--------------------------*
    Macro handling function
 *--------------------------*/
#include <stdio.h>
#include <string.h>
#include <alloc.h>
#include "lap.h"
#include "lapedit.h"
#include "laputil.h"
#include "lapeval.h"
#include "lapdata.h"
#include "lapmacro.h"
/*-----------------*
   PUBLIC FUNCTION
 *-----------------*/
PUBLIC void expandmacro(int argc,char *argv[],MACRO *m);
PUBLIC void editmacro(int margc,char *margv[]);
PUBLIC void listmacro(void);
PUBLIC void freemacro(void);
MACRO *getmacroptr(char *st);
PUBLIC void initmacro(void);
/*--------------------*
   INTERNAL FUNCTION
 *--------------------*/
/* 1.Macro  data structure management */
LOCAL void  addmacrogate(MACRO *m,MACROGATE *p);
LOCAL void  delmgate(MACRO *m);
LOCAL MACRO *newmacro(void);
LOCAL void  rmmacro(MACRO *m);
/* 2.Macro compilation                */
LOCAL void  makesymtable(MACRO *m);
LOCAL MACROSYM   *searchsym(char *st,MACROSYM symtab[],int symlen);
LOCAL int   scan1gate(MACRO *m,char *buff);
LOCAL void  scan_gate(MACRO *m);
LOCAL void  list1macro(MACRO *m);
/*  Share variable */
LOCAL char    linebuff[256];      /* Line input buffer */
LOCAL char    *argv[20];
LOCAL char    argvb[20][20];
LOCAL MACRO   *firstm=NULL,*lastm=NULL;
LOCAL int     macrocount=0;
/*-----------------------*
   MACRO DATA MANAGEMENT
 *-----------------------*/
MACRO *newmacro()
{
    MACRO *p;
    if ((p=malloc(sizeof(MACRO) )) ==NULL)
      { lapperr(MEMFAULT);  exit(MEMFAULT); }
    p->gfirst=p->glast=NULL;
    p->elmnum=p->xnodenum=p->inodenum=p->expand=0;
    return(p);
}
int addmacrolist(MACRO *p)
{
   p->next=NULL;
   if(firstm==NULL) firstm=p;
     else lastm->next=p;
   lastm=p;
   return(++macrocount);
```

```
}
void rm1macro(MACRO *m)
{
        delmgate(m);
        free(m->xsym);
        free(m->isym);
}
PUBLIC void freemacro(void)
{
   MACRO *p;
   int i;

   for(p=firstm;p !=NULL;p=p->next)
     rm1macro(p);
   macrocount=0;
}
PUBLIC MACRO *getmacroptr(char *st)
{
     MACRO *p;
     int i;

     for(p=firstm; p !=NULL;p=p->next)
       {
          if (!strcmp(st,p->name) ) return(p);
       }
      return(NULL); /* NOT FOUND */
}
/*-------------------------------------------------------------
   Init macro must be called once when software start
   or load new network
 *-------------------------------------------------------------*/
PUBLIC void  initmacro(void)
{
    firstm=lastm=NULL;
    macrocount=0;
}
/*-------------------------------*
    Internal macro gate routine
 *-------------------------------*/
void addmacrogate(MACRO *m,MACROGATE *p)
{
     p->next=NULL;
     if (m->gfirst==NULL)  m->gfirst=p;
       else  m->glast->next=p;
     m->glast=p;
}
void delmgate(MACRO *m)
{
    MACROGATE *p;

    for(p=m->gfirst;p !=NULL;p=p->next)
      free(p);
}
MACROGATE *newmgate(void)
{
   MACROGATE *p;
   if ((p=malloc(sizeof(MACROGATE))) ==NULL) {
     puts("Macro gate memory allocation error!");
    exit(MEMFAULT);
     }
```

```
    return(p);
}

/*-----------------*
   MACRO COMPILE
 *-----------------*/
void makesymtable(MACRO *m)
{
    int i;

    fetchstr(linebuff);
    m->xnodenum=strargv(linebuff,argv);
    printf("EXT: %s",linebuff);
    m->xsym=(MACROSYM *)calloc(m->xnodenum,sizeof(MACROSYM));
    for(i=0;i < m->xnodenum;i++)
       strcpy( m->xsym[i].str,argv[i]);

    fetchstr(linebuff);
    m->inodenum=strargv(linebuff,argv);
    printf("INT: %s",linebuff);
    m->isym=(MACROSYM *)calloc(m->inodenum,sizeof(MACROSYM));
    for(i=0;i < m->inodenum;i++)
       strcpy(m->isym[i].str,argv[i]);
#if  defined(DEBUG_MACRO)
  puts("SHOW SYMTABLE");
    for(i=0;i<m->xnodenum;i++)
         printf("[%02d] %s\n",i,m->xsym[i].str);
    for(i=0;i < m->inodenum;i++)
         printf("<%02d> %s\n",i,m->isym[i].str);
#endif
}

MACROSYM *searchsym(char *st,MACROSYM symtab[],int symlen)
{
     register i;

     for(i=0;i<symlen;i++)
       if (strcmp(symtab[i].str,st)==0) return(symtab+i);
     return(NULL);  /* return Null if not found */
}
int scan1gate(MACRO *m,char *buff)
{
    int  pin,i,index;
    MACROGATE  *p;
    MACROSYM   *entry;
    LOGICMODEL *model;
    char tbuff[10];

    strargv(buff,argv);
    if (strcmp(argv[0],"ENDM")==0) return(FALSE);
    /* Perform syntax check here */
    strtrunc(tbuff,argv[1],IDENLEN);
    if ( (model=getmodelptr(tbuff)) !=NULL)
      {  /* Syntax all ok. */
        p=newmgate();
        strtrunc(p->name,argv[0],IDENLEN);
        p->model=model;
        p->inum=model->inum;
        p->onum=model->onum;
        p->td=model->td;      /* Now use only default delay */
```

```
        for(index=2,pin=0;pin<p->inum;pin++)
         {
           if ( (entry=searchsym(argv[pin+index],m->isym,m->inodenum))==NULL)
            if ( (entry=searchsym(argv[pin+index],m->xsym,m->xnodenum))==NULL)
             { printf("INVALID INPUT NODE PIN:%d\n",pin);  entry=m->xsym; }
           p->ipin[pin]=entry;
         }
        for(index=p->inum+2,pin=0;pin<p->onum;pin++)
         {
           if ( (entry=searchsym(argv[pin+index],m->isym,m->inodenum))==NULL)
            if ( (entry=searchsym(argv[pin+index],m->xsym,m->xnodenum))==NULL)
             { printf("INVALID OUTPUT NODE PIN:%d\n",pin);  entry=m->xsym; }
           p->opin[pin]=entry;
         }
         addmacrogate(m,p);
       }
     else puts("INVALID MODEL REFERENCE IN MACRO COMPILATION!");
     return(TRUE);
}
void scan_gate(MACRO *m)
{
    do {
      fetchstr(linebuff);
      printf("%s",linebuff);
    } while (scan1gate(m,linebuff));
}
PUBLIC void editmacro(int margc,char *margv[])
{
      MACRO *mptr;
      register i;

      if( margc !=2) {  puts("MACRO SYNTAX ERROR !");  return;  }
      for(i=0;i<20;i++)  argv[i]=argvb[i];
      mptr=newmacro();
      strtrunc(mptr->name,margv[0],IDENLEN);
      makesymtable(mptr);
      scan_gate(mptr);
      addmacrolist(mptr);  /* Add new macro to MACRO link list */
}
/*------  END MACRO EDITOR FUNCTION ------*/
/*-------------------------*
   MACRO LISTING FUNCTION
 *-------------------------*/
void list1macro(MACRO *m)
{
   int i;
   MACROGATE *p;

   printf("MACRO: %*s\n",IDENLEN,m->name);
   printf("EXT: ");
   for(i=0;i<m->xnodenum;i++)
     printf("%6s",m->xsym[i].str);
   printf("\nINT: ");
   for(i=0;i<m->inodenum;i++)
     printf("%6s",m->isym[i].str);
   printf("\n");
   for(p=m->gfirst;p !=NULL;p=p->next)
    {
       printf("%6s%6s%6d",p->name,p->model->name,p->td);
       for(i=0;i<p->inum;i++) printf("%6s",p->ipin[i]->str);
```

```
        for(i=0;i<p->onum;i++) printf("%6s",p->opin[i]->str);
        printf("\n");
     }
    printf("END MACRO: %*s\n",IDENLEN,m->name);
}
PUBLIC void listmacro(void)
{
    MACRO *p;

    for(p=firstm;p !=NULL;p=p->next) list1macro(p);
    printf("Total macro defined:%5d \n",macrocount);
}
/*-------------------*
    MACRO EXPANSION
 *-------------------*/
PUBLIC void expandmacro(int argc,char *argv[],MACRO *m)
{
     register i,pin;
     LOGICNODE *p;
     LOGICELEMENT *gate;
     MACROGATE *g;

     if (argc != (m->xnodenum+2) ) /* Check for correct arg */
       {  lapperr(SYNTAXERR); return; }
     /* Fill Internal node link */
     for(i=0;i<m->inodenum;i++)
      {
         m->isym[i].node=p=makenewnode("$");
         p->type=INTERN;
         addnode(p);
      }
     /* Fill xtern link table */
     for(i=0;i<m->xnodenum;i++)
       {
           m->xsym[i].node=p=getnodeptr(argv[i+2]);
           p->type=XTERN;
       }
     /* Scan all element */
      for(g=m->gfirst;g !=NULL;g=g->next)
      {
          gate=makenewgate("#");
          gate->type=INTERN;
          gate->model=g->model;
          gate->td=g->td;
          gate->inum=g->inum;
          gate->onum=g->onum;
          gate->ps=LOW;
          for(pin=0;pin<g->inum;pin++)
            gate->in[pin]=g->ipin[pin]->node;
          for(pin=0;pin<g->onum;pin++)
            gate->out[pin]=g->opin[pin]->node;
          addelm(gate);
       }
}
```

```
/*
 *   MODULE: LAPDISPLAY
 *   AUTHOR: P.UTHAYOPAS
 *   DATE:
 *          This is display section for lap software
 */
#include   <stdio.h>
#include   <dos.h>
#include   <ctype.h>
#include   "lap.h"
#include   "lapcmd.h"
#include   "lapdata.h"
#include   "laputil.h"
#include   "hgraphic.h"

IMPORT   const int   usrtime;    /* Real sim time now from lapsim */

#define  XMIN           50        /* Minimum display area       */
#define  XMAX           640       /* Maximum display area       */
#define  XST            140       /* x end of label block       */
#define  XTSTR          140       /* START X OF TIMING DISPLAY  */
#define  XTEND          640       /* END X OF TIMING DISPLAY    */
#define  MAXTSCALE      500       /* maximum display point      */
#define  GRIDHIGH        19
#define  GRIDLOW        257
#define  MIDDLECOL       42
#define  LABELLEFT       8
#define  STATLINMIN      36
#define  STATLINMAX      42
#define  statline(n)     (n+STATLINMIN)  /* ROW OF STAT LINE n */
#define  labelbase(n)    (16*n+16)       /* BASE Y CO-ORDINATE OF LABEL */
/* Level of signal graph */
#define  signal0(n)      (16*n+14)
#define  HLEVEL           8
#define  ZLEVEL           4
#define  baseline(n)     (n*8)           /* base Y of line n */

LOCAL void  heading(st) /* Display heading field */
char *st;
{     int stcol;
      stcol=MIDDLECOL-(strlen(st)/2);
      box(XMIN,baseline(2)-12,XMAX,baseline(2)+2);
      gotogxy(stcol,2); gputs(st);
}
LOCAL void  label(n,st) /* Display label #n <1-15>  */
int n;
char *st;
{
       if ((n<=15)&&(n>0) ) {
        writestr(LABELLEFT,2*n+2,st); }
}
LOCAL void dispframe()
{
  register i;
  int yh,yl;

  yh=18;    yl=258;
  box(XMIN,yh,XMAX,yl);
  line(XST,yh,XST,yl);
  for(i=1;i<=15;i++)
```

```
    {
        yh=labelbase(i)+2;  line(XMIN,yh,XST,yh);
    }
}
LOCAL void dpstatbox()
{
     box(XMIN,baseline(STATLINMIN)-5,XMAX,baseline(STATLINMAX)+5);
}
LOCAL void  dpstatline(n,col,st)
int n;
int col;
char *st;
{
   writestr(LABELLEFT+col-1,statline(n),st);
}
LOCAL void dptimescale(scale)
int scale;
{
    int x,y,inc;

    gotogxy(LABELLEFT,34);  gputs("TIME");
    y=baseline(34);
    inc=scale*10;
    line(XTSTR,y-5,XTEND,y-5);
    for(x=0;x<=MAXTSCALE;x+= inc)
      line(XTSTR+x,y-3,XTSTR+x,y-7);
}
/*---------------------------*
  Time grid control routine
 *---------------------------*/
LOCAL void initmovgrid();
LOCAL void drawgrid();
LOCAL void gridright();
LOCAL void gridleft();
LOCAL void movgrid();

LOCAL BOOLEAN gridon=FALSE;
LOCAL int  gridxst;
LOCAL int  gridx;
LOCAL int  mgridstep;

void initmovgrid()
{
     gridon=FALSE;
     gridx=0;   /* Grid first position on left most */
     mgridstep=1;
}

void drawgrid(step,scale) /* Fix grid line */
int step;
int scale;
{
    int x,inc;

    if ( ! gridon)
    {
    inc=step*scale;
    drawmode(PENXOR);
    for(x=0;x<MAXTSCALE;x+=inc) {
      vdash(XTSTR+x,GRIDHIGH,GRIDLOW,3);  }
```

```
    drawmode(PENDOWN);
    }
}
/* Moving grid routine */
void gridleft(scale)
int scale;
{
 char buff[5];
 int inc;
 if (gridon)
 {
  drawmode(PENXOR);
  inc=mgridstep*scale;
  vdash(XTSTR+gridx,GRIDHIGH,GRIDLOW,1); /* erase old grid */
  gridx=( ( (gridx - inc) < 0 ) ? gridx : gridx-inc);
  vdash(XTSTR+gridx,GRIDHIGH,GRIDLOW,1); /* draw new grid */
  drawmode(PENDOWN);
  sprintf(buff,"%3d",gridxst+ gridx/scale);
  dpstatline(4,8,buff);
 }
}
void gridright(scale)
int scale;
{
  char buff[5];
  int inc;
 if (gridon)
 {
  drawmode(PENXOR);
  vdash(XTSTR+gridx,GRIDHIGH,GRIDLOW,1); /* erase old grid */
  inc=mgridstep*scale;
  gridx=( ((gridx+ inc)>MAXTSCALE) ? gridx : gridx+inc);
  vdash(XTSTR+gridx,GRIDHIGH,GRIDLOW,1); /* draw new grid */
  drawmode(PENDOWN);
  sprintf(buff,"%3d",gridxst+ gridx/scale);
  dpstatline(4,8,buff);
 }
}
void gridfast()
{
  char buff[5];
  mgridstep=( (++mgridstep >40) ? 1 : mgridstep);
  sprintf(buff,"%2d",mgridstep);
  dpstatline(4,22,buff);
}
void gridslow()
{
  char buff[5];
  mgridstep=( (--mgridstep < 1) ? 40 : mgridstep);
  sprintf(buff,"%2d",mgridstep);
  dpstatline(4,22,buff);
}
void movgrid()
{
    int yh,yl;
    yh=18;    yl=258;
    drawmode(PENXOR);
    if (gridon)  gridon=FALSE;
       else gridon=TRUE;
    vdash(XTSTR+gridx,yh,yl,1);
```

```
    drawmode(PENDOWN);
}
/*------------------------*
   Signal display routine
 *------------------------*/
LOCAL void dplow(xs,ys,scale)
int xs;
int ys;
int scale;
{
  line(xs,ys,xs+scale,ys);
}
LOCAL void dphigh(xs,ys,scale)
int xs;
int ys;
int scale;
{
  line(xs,ys-HLEVEL,xs+scale,ys-HLEVEL);
}
LOCAL void dpzstate(xs,ys,scale)
int xs;
int ys;
int scale;
{
  line(xs,ys-ZLEVEL,xs+scale,ys-ZLEVEL);
  line(xs,ys-ZLEVEL-1,xs+scale,ys-ZLEVEL-1);
}
LOCAL void dpunknown(xs,ys,scale)
int xs;
int ys;
int scale;
{
    line(xs,ys-HLEVEL,xs+scale,ys-HLEVEL);
    line(xs,ys,xs+scale,ys);
    line(xs,ys,xs+scale,ys-HLEVEL);
    line(xs,ys-HLEVEL,xs+scale,ys);
}
LOCAL void dpbusfault(xs,ys,scale)
int xs;
int ys;
int scale;
{
   line(xs,ys-HLEVEL,xs+scale,ys-HLEVEL);
   line(xs,ys,xs+scale,ys);
   line(xs,ys,xs+scale,ys-HLEVEL);
}

LOCAL void dp1signal(n,data,scale)
int n;
LOGICSTATE *data;
int scale;
{
       register t;
       LOGICSTATE  logic,prelogic;
       int   xplot,yplot,maxplot;
       static int spike_mask=0xbf;

       prelogic=LOW;
       maxplot=MAXDPSTEP/scale; /* Maximum x plot */
       for(t=0;t < maxplot ;t++)
```

```
        {
          xplot=scale*t + XTSTR;
          yplot=signal0(n);
          logic=( ((data[t] & BUS_CONTENSION) !=0) ?  BUS_CONTENSION : data[t]);
          logic &=spike_mask;  /* suppress spike display  */
          switch( logic )
           {
             case LOW    :
                       dplow(xplot,yplot,scale);
                       break;
             case HIGH   :
                       dphigh(xplot,yplot,scale);
                       break;
             case ZSTATE :
                       dpzstate(xplot,yplot,scale);
                       break;
             case UNKNOWN :
                       dpunknown(xplot,yplot,scale);
                       break;
             case BUS_CONTENSION :
                       dpbusfault(xplot,yplot,scale);
                       break;
           }
          if (logic != prelogic )
             line(xplot,yplot,xplot,yplot-HLEVEL);
          prelogic=logic;
        }
}

LOCAL  void showsig(cmdptr)
CMD_DISPLAY *cmdptr;
{
     int i,t,n,range;
     static char buff[100];
     IMPORT int gridx,gridxst;

     hirespage(1);
     heading("SIMULATION RESULT FROM LAP 1.0");
     dispframe();
     dptimescale(cmdptr->scale);
     dpstatbox();
     range=(MAXDPSTEP/cmdptr->scale)+cmdptr->rtime+cmdptr->timestart;
     gridxst=cmdptr->rtime+cmdptr->timestart;
     sprintf(buff,"TIME:%05d TO %05d SCALE:%02d GRID:%02d MRT",
               gridxst,range,cmdptr->scale,cmdptr->gridstep);
     dpstatline(2,1,buff);
     sprintf(buff,"OFFSET:%3d MRT SPEED:%2d STEP",gridxst,mgridstep);
     dpstatline(4,1,buff);
     sprintf(buff,"F2-HARDCOPY  F3-GRID ON/OFF  F4-MOVING GRID ESC-EXIT");
     dpstatline(6,1,buff);
     for(i=0;i< cmdptr->nodisp;i++) {
       n=i+1;
       sprintf(buff,"%s",cmdptr->nodename[i]);
       label(n,buff);
       dp1signal(n,(cmdptr->simdata[i] + cmdptr->timestart),cmdptr->scale);
      }
}

LOCAL  void showtiming(cmdptr)
CMD_DISPLAY *cmdptr;
```

```
{
     int timelimit;
     int key;

     drawmode(PENDOWN);
     initmovgrid();
    /* Adjust display start time  */
    timelimit=MAXTIME-(MAXDPSTEP/cmdptr->scale);
    cmdptr->timestart=(  cmdptr->timestart >timelimit
                       ? timelimit
                       : cmdptr->timestart);
    cmdptr->rtime=( usrtime >MAXTIME ? usrtime-MAXTIME : 0);
     showsig(cmdptr);
     if (cmdptr->waittime!=0)
        { sleep(cmdptr->waittime);
          text();
          return; }
     while( (key=getkey()) !=ESC ) /* wait for ESC key */
     {
       switch(key)
        {
          case F2    :  hardcopy(1);
                        break;
          case F3    :  drawgrid(cmdptr->gridstep,cmdptr->scale);
                        break;
          case F4    :  movgrid();
                        break;
          case LEFTKEY  : gridleft(cmdptr->scale);
                          break;
          case RIGHTKEY : gridright(cmdptr->scale);
                          break;
          case '+'      : gridfast();
                          break;
          case '-'      : gridslow();
                          break;
        }
      }
     text();
}  /* end of display */

/*-------------------------
   Set display parameter
 *-------------------------*/
LOCAL  void setdpparam(cmdptr)
CMD_DISPLAY *cmdptr;
{
    LOGICNODE *p;
    int num,timelimit;

    puts("** Display parameter setting **");

    /* Reading display scale */
    do {
    printf("Display scale[%02d]>",cmdptr->scale);
    num=getint();
    } while ( (num > MAXDPSCALE) ||(num < 0) );
    if (num ==0) cmdptr->scale=( cmdptr->scale >0 ? cmdptr->scale : 1);
       else cmdptr->scale=num;

    /* Read display starting time */
```

```
    do {
    printf("Display start at step <0-%d>[%05d]>",MAXTIME,cmdptr->timestart);
    num=getint();
    } while ( (num > MAXTIME) ||(num < 0) );
    cmdptr->timestart=num;
    /* reading display grid step width */
    do {
    printf("Display grid every[%03d]>",cmdptr->gridstep);
    num=getint();
    } while ( (num > MAXGRIDSTEP) ||(num < 0) );
    if (num ==0) cmdptr->gridstep=( cmdptr->gridstep>0 ? cmdptr->gridstep : 1);
       else cmdptr->gridstep=num;

}
LOCAL  void setdpnode(cmdptr)
CMD_DISPLAY *cmdptr;
{
    LOGICNODE *p;
    int num,i;
    static char buff[20];

    do {
    printf("NO. of node to display (less than 15):");
    num=getint();
    } while ( (num > MAX_DISPLAY_NODE) ||(num < 0) );
    for(i=0;i < num;i++)
     {
       do {
       printf("<%2d> NAME:",i+1); gets(buff);
       strupr(buff);
       if ((p=oldnode(buff))==NULL) then
         printf("Invalid node name or name not found!\n");
       } while ( p ==NULL);
       cmdptr->simdata[i]=p->outstate;
       cmdptr->nodename[i]=p->name;
     }
     cmdptr->nodisp=num;
}

LOCAL void   showsigtab(p)
CMD_DISPLAY *p;
{
     int i,col,limit;

     printf("\t\t*** DISPLAY SIMULATION RESULT TABLE ***\n");
     limit=( (MAXTIME - p->timestart) < 70 ? MAXTIME - p->timestart : 70 );
     printf("TIME:%5d TO %5d\n\n",p->timestart,p->timestart+limit);
     for(col=0;col < p->nodisp;col++)
       {
         printf("%6s:",p->nodename[col]);
         for(i=0;i< limit ;i ++)
           {
            printf("%c",logictoc(*(p->simdata[col] + p->timestart +i)) );
           }
         printf("\n");
       }
    printf("\n\n");
}

LOCAL void init_display(p)
```

```
CMD_DISPLAY *p;
{
     register i;

     p->cmd=SETUP;
     p->nodisp=0;  /* No data to display */
     p->rtime=0;
     p->scale=1;
     p->gridstep=20;
}
/*
 *   Main section of display module
 */
PUBLIC void dspmodule(p)
CMD_DISPLAY *p;
{
  switch(p->cmd)
  {
     case  INITDP       :   init_display(p);
                            break;
     case  SETUP        :   setdpparam(p);
                            break;
     case  NODESEL      :   setdpnode(p);
                            break;
     case  TABLE        :   showsigtab(p);
                            break;
#ifdef   IBMPC
     /* Timing display can only used in IBM with Hercules card */
     case  TIMING       :   showtiming(p);
                            break;
#endif
  }
}
```

```
#include   <stdio.h>
#include   <fcntl.h>
#include   <ctype.h>
#include   <string.h>
#include   <dir.h>
#include   <stat.h>
#include   <dos.h>
#include   <bios.h>
#include   <io.h>
#include   <errno.h>
#include   <alloc.h>
#include   "lap.h"
#include   "laputil.h"

/* Uses the BIOS to read the next keyboard character */
int getkey(void)
{
 int key, lo, hi;

 key = bioskey(0);
 lo = key & 0X00FF;
 hi = (key & 0XFF00) >> 8;
 return((lo == 0) ? hi + 256 : lo);
} /* getkey */

void waitkey()
{
  printf("\nWAITING...");
  getkey();
}

/*-------------------------------------------------
      NAME: findcmdcode(cmd,cmdtab,length)
      USAGE: convert command string to code
      char     *cmdstr;          Command string
      CMDTABLE *cmdtab;          Command string table
      int  length;       length of code table
 *-------------------------------------------------*/
int  findcode(char *cmd,CMDTABLE  *cmdtab,int length)
{
   register i;
   for(i=0; i< length;i++ )
        if ( !strcmp(cmd,(cmdtab+i)->str) ) return( (cmdtab+i)->code);
   /* not found  */
   return(NULL);
}

/*-----------------------------------
   DISPLAY PROMPT AND GET DATA INPUT
 *-----------------------------------*/
PUBLIC char     *getcmdln(char *prompt,char *buff)
{
   char *p;
   do
    {  printf(prompt);
       gets(buff);
    } while  (strlen(buff)==0);
   /* CONVERT CONTROL CODE TO SPACE */
   for(p=buff;*p;p++) if ( iscntrl(*p) ) *p=0x20;
    strupr(buff);  /* Convert string to uppercase */
```

```
    return(buff);
}

/*-------------------------------------------------------------*
  NAME:  int strarg(char *argv[]);
  USAGE:
        SEPERATE INPUT STRING INTO PART USING WHITE SPACE
        AS SEPERATOR
    Parameter:
        argv =array of pointer to buffer (same as argv in C)
    Return:  parameter count
 *-------------------------------------------------------------*/
PUBLIC int strargv(char *ln,char *argv[])
{
    int argc;
    char *p;

    /* Change all  control code to blank */

    for(argc=0; *ln ;)
       {
         while( isspace( *ln) ) ++ln; /* SKIP WHITE SPACE */
         if (*ln==0) continue;  /* check if space only in input buffer */
         for(p=argv[argc++]; ! (isspace(*ln) || (*ln==0) ) ;)
           *(p++) = *(ln++);
         *p=0;
         }
     return(argc);
}

/*-------------------------------------------------------------*
    NAME:prmlist(argc,argv)
    USAGE: list parameter accept from getcmd()
    PARAMETER:
       unsigned argc=argument count
       char *argv[] =pointer to argument value
    RETURN: none
 *-------------------------------------------------------------*/
PUBLIC void prmlist(int argc,char *argv[])
{
   int i;
   printf("Number of parameter: %d\n",argc);
   for (i=0;i< argc;i++)
     printf("Parameter [%2d] : %s\n",i,argv[i]);
}

PUBLIC int getint(void) /*  return integer value read from user */
{
  static char buffer[20];
  static int x;

  gets(buffer);
  x=atoi(buffer);
  return(x);
}
/* Truncate string to n character */
PUBLIC char *strtrunc(char *st1,char *st2,int n)
{
      strncpy(st1,st2,n);  /* Truncate only first n char */
      st1[n]=NULL;         /* NULL terminatre for  buffer */
```

```
    return(st1);
}

/* Convert input char to logicstate */
LOGICSTATE tologic(char c)
{
    switch(c)
     {
       case  '1' :
       case  'H' :   return(HIGH);
       case  '0' :
       case  'L' :   return(LOW);
       case  'Z' :   return(ZSTATE);
       case  'X' :   return(UNKNOWN);
       default   :   return(UNKNOWN);
     }
}
char  logictoc(LOGICSTATE s)
{
    static char stab[]={'L','H','X','Z'};
    return(stab[s]);
}
/*
 *    Standard error message string
 *    imitate perr\90 function
 */
PUBLIC   int laperrnum;   /* internal error code in lap */
PUBLIC   int lapmaxerr=2;   /* maximum error code         */
PUBLIC   char *laperrstr[]={
           "Syntax error!",
           "Invalid command",
           "Memory allocation fault! abort LAP1.0"
           };

/*-------------------------------------------------*
   LAPERRMSG  - return char * to error message
 *-------------------------------------------------*/
PUBLIC char *laperrmsg(int errnum)
{
    return(laperrstr[errnum]);
}

PUBLIC void lapperr(int num)
{
  printf("%s\n",laperrstr[num]);
}

/*---------------------------------*
   S Y S T E M   I N T E R F A C E
 *---------------------------------*/
/*
      NAME: dir( char *pathname)
      display directory specify "pathname"
*/
int dir(char *st)
{
  struct ffblk ffblk;
  int done,count,colc;
  char *name,*ext;
  unsigned long dfree;
```

```
  static char *blank="    ";
  printf("DIRECTORY of %s\n\n",st);
  count=0;
  colc=0;
  done=findfirst(st,&ffblk,0);
  while(!done) {
   count++;
   if ( colc > 4 ) then { printf("\n"); colc=1; }
     else colc++;
   name=ffblk.ff_name;
   if ( (ext=strchr(name,'.') ) ==NULL)  ext=blank;
     else {  *ext=0; ++ext; }
   printf("  %-8s.%-3s ",name,ext);
   done=findnext(&ffblk);
   }
   dfree=diskfree(0); /* Get current drive free space */
   printf("\n\t%4d FILE(S) %10ld Bytes available on disk\n",count,dfree);
   return(count);
}
unsigned long diskfree(int drv)
{
  unsigned long bperc;  /* byte per cluster */
  struct dfree dfbuff,*p;

  p=&dfbuff;
  getdfree(drv,p);
  bperc=p->df_bsec * p->df_sclus;
  return(p->df_avail * bperc);
}

/*   MEMAVAIL()  - Report memory size available */
unsigned long memavail(void)
{
  return( coreleft() );
}
void delfile(char *st)
{
    int retcode;

    retcode=unlink(st);
    if (retcode==0) return;
    switch(errno)
     {
        case ENOENT  :
                puts("PATH OR FILE NOT FOUND! TRY FULL PATH NAME");
                break;
        case EACCES :
                puts("ACCESS DENIE FOR THIS NAMED FILE!");
                break;
     }
    errno=0;  /* reset errno */
     return;
}
PUBLIC void typefile(char *st)
{
   FILE *fp;
   char c;

   if ((fp=fopen(st,"r"))==NULL) then
```

```
    {  puts("File not found in current directory!"); }
    else
     {
        while( (c=fgetc(fp)) !=EOF)
          putchar(c);
        fclose(fp);
     }
   printf("\n");
}
PUBLIC void copyfile(char *source,char *target)
{
  int src,dest;   /* Source and destination file handle */
  int saccess,daccess,dpermiss; /* Access code of file */
  static char buff[512];
  int len;
  extern int sys_nerr;
  extern char *sys_errlist[];

  /* Construct source file access key */
  saccess=O_RDONLY | O_BINARY;
  if ((src =open(source,saccess) ) ==-1)
    {
       if (errno < sys_nerr)
         printf("%s !\n" ,sys_errlist[errno]);
        errno=0;
       return;
    }
  /* Construct destination file access key */
  daccess=O_CREAT | O_TRUNC | O_BINARY | O_WRONLY;
  dpermiss= S_IREAD | S_IWRITE;
  if ( (dest=open(target,daccess,dpermiss)) == -1)
    {
       if (errno < sys_nerr)
         printf("%s !\n" ,sys_errlist[errno]);
        errno=0;
       return;
    }
  while ( (len=read(src,buff,512)) > 0)
    write(dest,buff,len);
  close(src);
  close(dest);
}
```

ประวัติผู้เขียน

นายภุชงค์ อุทโยภาศ เกิดเมื่อวันที่ 26 มิถุนายน พ.ศ.2507 ที่อำเภอบางกอกน้อย กรุงเทพมหานคร สำเร็จการศึกษาขั้นปริญญาตรีจาก ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2527 ปัจจุบัน ทำงานในตำแหน่งวิศวกรออกแบบ แผนกซอฟแวร์และออกแบบ บริษัทไฟฟ้าฟิลิปส์แห่งประเทศไทย