# *Appendix 2*
# Alecsis standard libraries

In this Appendix, only headers of standard libraries. These headers are included using included command. In headers, all declarations are given. All definitions (modules, functions, etc.) mentioned in the headers are in one file. For instance, header `alec.h` corresponds to library `alec.ao`, referenced as library `alec`.

Only some of the files that are stored in directory `alecsis/include` are given here. In future issues of this Manual, we will give the most important headers specific for digital simulation, too.

Files are given in alphabetical order:

## A2.1 alec.h

This is a standard library containing all main definitions and declarations needed. The header is in the standard place (directory alecsis/include) under the name `alec.h`, and the library body is in the file `alec.ao` (directory `alecsis/lib`). Header file heeds to be included using `include` command. Library `alec` need not to be given explicitely using '`-l`' option or `library` command, as it is standard library.

The content of the header file is following:

```
/* Faculty of Electronic Engineering Nis
 * Alecsis 2 hybrid simulator library header file
 * Library: alec
 *
 * These declarations are not a must, but compiler will complain
```

```
 * about parameters for functions listed below.
 *
 */
#ifndef _ALEC_INCLUDED
#  define _ALEC_INCLUDED

#  ifndef NULL
#    define NULL  (0)
#  endif /* NULL */

#  ifndef EOF
#    define EOF   (-1)
#  endif /* EOF */

# ifndef MAX
#  define MAX(_x, _y) ((_x)>(_y) ? (_x) : (_y))
# endif /* MAX */

# ifndef MIN
#  define MIN(_x, _y) ((_x)<(_y) ? (_x) : (_y))
# endif /* MIN */

# define cout stdout
# define cin  stdin
# define cerr stderr

   typedef struct {
 int   __cnt;
 char  *__ptr;
 char  *__base;
 int   __flag;
 char  __file;
   } FILE;

   /* standard built-in function prototypes */

   extern  int printf(const char *, ... );
   extern  int fprintf(FILE *, const char *, ...);
   extern  int sprintf(char *, const char *, ...);
   extern  int fputc(char, FILE *);
   extern  int putc(char, FILE *);
   extern  int putchar(char);
   extern  char fgetc(FILE *);
   extern  char getc(FILE *);
   extern  char *gets(char *);
   extern  char *fgets(char *, int, FILE*);
   extern  char getchar(void);
   extern  int exit(int=0);
   extern  FILE *fopen(const char *, const char *);
   extern  int fclose(FILE *);
   extern  int fflush(FILE *);
   extern  int feof(FILE *);
   extern  int fseek(FILE *, int, int);
   extern  int ftell(FILE *);
   extern  int rewind(FILE *);
   extern  int fwrite(const void *, int, int, FILE *);
   extern  int fread(void *, int, int, FILE *);
   extern  void *calloc(int, int);
   extern  void *malloc(int);
   extern  void free(void *);
   extern  double node_value(int, int);
   extern  double drand(void);
   extern  void srand(int=1);
```

```
   extern  double time_now(void);
   extern  void warning (const char *, int=0);
   extern  char *strcpy(char *, const char *);
   extern  int strcmp(const char *, const char *);
   extern  int strlen(const char *);
   extern  int get_info (int);
   extern  int get_info (int, char *);
   extern  int atoi(const char*);
   extern  double atof(const char*);
   extern  int system(const char*);

   extern int set_bpoint(double);

 /* standard Boolean choices */
# define True  1
# define False 0

 /* alternative solution - Bool type */
typedef enum { false, true } Bool;

 /* switch-like choices */
# define On  1
# define Off 0

 /* Integration method choices */
# define None           0
# define EulerBackward  1
# define Gear2          2

        /* Matrix renumeration options */
// # define None  0
# define Fast    1
# define Best    2
# define Frontal 3

 /* dcon choices */
# define Initial 1
# define Always  2

 /* get_info selection indx */
# define CurrentPath   0
# define ParentPath    1
# define CurrentLevel  2

 /* the most common implicit aliases */
implicit {
 resistor   r;
 capacitor  c;
 inductor   l;
 vgen       v;
 cgen       i;
 mosfet     m;
 bjt        q;
 jfet       j;
 diode      d;
 switch     s;
};

 /* pulse generator defined as a module */

module pulse (n1, n2) action (    double vlo,     // low level
                  double vhi,     // high level
                  double tr,      // rise delay
```

```
                    double twl,      // high level pulse width
                    double tf=0,     // fall delay
                    double twh=0,    // low level pulse width
                    double td=0      // start delay time
                   );

# ifndef _BIT_INCLUDED
#   include <bit.h>
# endif /* _BIT_INCLUDED */

 /* overloaded << and >> - C++ style I/O */

FILE *operator<< (FILE *fp, int i);
FILE *operator<< (FILE *fp, double d);
FILE *operator<< (FILE *fp, const char *s);
FILE *operator<< (FILE *fp, char c);

FILE *operator>> (FILE *fp, char &c);
#endif /* _ALEC_INCLUDED */
```

This library contains prototypes of all intrinsic functions of general usage. The rest of C functions (except the mathematical) are **not implemented** and cannot be called.

Command `implicit`, used in this file, gives possibility to use Alecsis similarly as SPICE.

We realized trapezoidal voltage generator as a `module` - its declaration is given above.

Beside standard C functions, `alec.h` contains a prototype of function **`warning`**, that is specific for simulaiton, and is therefore not a standard C-function. You can use this function to print information of place, time and context (the process, current component, etc.) during the execution. If the argument differs from 0, the simulator stops the execution, otherwise the simulation continues.

## A2.2. ctype.h

This is not a real library since it has only the header file - there is no file named `ctype.ao`. This library contains the definitions of macros for work with characters:

```
#ifndef _CTYPE_INCLUDED
# define _CTYPE_INCLUDED

#  define _U 01
#  define _L 02
#  define _N 04
#  define _S 010
#  define _P 020
#  define _C 040
#  define _B 0100

     extern const char __ctype[];
     extern const char __upshift[];
     extern const char __downshift[];

#  define isalpha(__c)  (__ctype[__c]&(_U|_L))
#  define isupper(__c)  (__ctype[__c]&_U)
#  define islower(__c)  (__ctype[__c]&_L)
#  define isdigit(__c)  (__ctype[__c]&_N)
#  define isalnum(__c)  (__ctype[__c]&(_U|_L|_N))
#  define isspace(__c)  (__ctype[__c]&_S)
```

```
#  define ispunct(__c)  (__ctype[__c]&_P)
#  define isprint(__c)  (__ctype[__c]&(_P|_U|_L|_N|_B))
#  define isgraph(__c)  (__ctype[__c]&(_P|_U|_L|_N))
#  define iscntrl(__c)  (__ctype[__c]&_C)

#  define isascii(__c)  ((__c) <= 0177)
#  define toupper(__c)  ((__upshift)[__c])
#  define tolower(__c)  ((__downshift)[__c])
#  define toascii(__c)  ((__c))

#endif
```

## A2.3. bit.h

This file contains only declarations, **while the definitions are in the library `alec`**. This is the elementary system of state `bit` containing `'0'` and `'1'` - logic zero and one, as well as the functions for overload of elementary operators:

```
/*
 * Faculty of Electronic Engineering Nis
 * Alecsis 2.0 hybrid simulator library header file
 * Library: alec
 * Content: predefined two-valued logic system
 */

#ifndef _BIT_INCLUDED
# define _BIT_INCLUDED

 typedef enum { '0', '1', ' '=void, '_' = void } bit;

 /* overloaded logical operators for type "bit" */
 extern bit operator~ (bit);
 extern bit operator& (bit, bit);
 extern bit operator| (bit, bit);
 extern bit operator^ (bit, bit);
 extern bit operator~& (bit, bit);
 extern bit operator~| (bit, bit);
#endif
```

## A2.4. gnulib.h

Functions declared here are used for on-line viewing of simulation results. They enable inter-process communication with program `gnuplot`, that is used for viewing of simuation results. Function bodies are in library `gnulib`, that is to be appended using library command or '`-l`' command option.

```
#ifndef _GNULIB_INCLUDED
# define _GNULIB_INCLUDED
#include <alec.h>
#include <unistd.h>

typedef enum { Lines, Points, LinesPoints, Impulses } LineStyle;

struct TraceList {
 char *name;
 int channel;
 LineStyle lst;
```

```
 struct TraceList *next;
};
typedef enum { CallGnu, Close } ExitStyle;


class gnu {
   protected:
 char *gnufile;
 char *title;
 FILE *fp;
 int ntraces, nchannels;
 LineStyle line_style;
 TraceList *traceh, *tracet;
    public:
 gnu (const char* ="gnu.dat", const char* = "Alecsis results");
 ~gnu();
 set_line_style(LineStyle);
 add_trace(const char*, LineStyle = Lines, int = 1);
 add_traces (const char*, ...);
 add_data(double, Bool);
 update  (double, ...);
 close_data ( ExitStyle, const char *geom="" );
 report();
};

struct ResultBuffer {
 double time;
 double *values;
 struct ResultBuffer *next;
};

class GnuOnLine: public gnu {
   protected:
 double tstop, update_period;
 double tprint, last_time;
 double vmin, vmax;
 double cmin, cmax;
 Bool channel_open, first_update;
 int channel[2];
 ResultBuffer *rbh, *rbt;
 char *plot_mess;
 create_plot_mess();
 initialize();
 flush_results();
 tell_gnu(const char*);
 set_value_margins();
   public:
 GnuOnLine (const char* ="gnu.dat",
                 const char* = "Alecsis results");
 ~GnuOnLine ();
 time_frame(double, double, double=0.0);
 value_frame (double, double);
 open_channel (const char* ="");
 open_channel (double, double, double, double,
                       double = 0.0, const char* = "");
 send_data (double, ...);
 main_loop();
};
#endif
```

The most appropriate way to explain these functions is to give an example. Here is an ring oscillator description, where CMOS inverter is used as ring element. This inverter is cloned using `clone` command in an ring of inverters (5 inverters in our example).

```
/*************************************************************/
spice {
# include "omos.mod"
}
# include <alec.h>
#include <gnulib.h>
library gnulib;
/*************************************************************/
    /* define analog inverter as a subcircuit */

module inverter ( In, Out, vdd, vss ) {
  mup (Out, In, vdd, vdd){ model=pomos;l=5u; w=35u;ad=as=170p;
                           pd=ps=50u;};
  mdwn(Out, In, vss, vss){ model=nomos;l=5u; w=10u;ad=as=50p;
                           pd=ps=35u;};
}
/*************************************************************/
module ring (node vdd, vss; node nodes[]) {
 inverter inv;

 action structural (int nring) {
    int i;
    for (i=0; i<nring;  i++) {
       if (i==nring-1)
        clone inv [i] (nodes[0], nodes[i], vdd, vss);
       else
        clone inv [i] (nodes[i+1], nodes[i], vdd, vss);
    }
 }
}
/*************************************************************/
#define Period 50ns

root module ring_oscilator () {
 ring rn;
 vgen vdd;
 node tmp[5] <- { 0.5v, 4.5v, 0.5v, 4.5v, 0.5v };

 vdd (Vdd, 0) 5v;
 rn (Vdd, 0, tmp) action (5);

 timing { tstop = Period; a_step = a_stepmin = Period/200; }

 out { node tmp[0];
       node tmp[1];
       node tmp[2];
       node tmp[3];
       node tmp[4]; }

#ifdef GNU
 action {
    static GnuOnLine gp ("ring", "Ring oscillator");
    process initial {
       gp.add_traces("tmp0", "tmp1", "tmp2", "tmp3", "tmp4", 0);
       gp.time_frame(Period, Period/100);
       gp.value_frame(0v, 5v);
       gp.open_channel();
    }
    process post_moment {
       gp.send_data(tmp[0], tmp[1], tmp[2], tmp[3], tmp[4]);
    }
    process final {
       gp.main_loop();
```

```
    }
  }
#endif
}
/*************************************************************/
```

Usage of gnulib functions are under preprocessor option

```
# ifdef GNU
```

so they will be activated if Alecsis is invoked using:

```
alec -DGNU ring
```

where `ring.ac` is the name of the file listed above.

Firstly, an instance of class `GnuOnLine` with name `gp` is declared. Constructor receives two parameters (both have default values). First is the file name, used to store waveform data, in this example `ring.dat` (extension '`.dat`' is added to the given name). The second parameter is waveform title, to appear above the graphics.

Class method `add_traces` is invoked in `process initial`. It gives the number of waveforms to be traced, and their names. This method is realized as the function with variable name of arguments, and the list of arguments has to be finished with `0`. All waveforms are drawn on one graphics, i.e. with one y-axis.

Method `time_frame` gives the last time point for x-axis (the first time point is assumed to be `0`), and the time step for waveform updating. Method `value_frame` gives **initial** range for y-axis scale (min and max). It is to be noted that y-axis range will be automatically updated during the simulation, as we usually do not know range of our simulation results in advance. However, x-axis cannot be updated, so arguments to `time_frame` must be correct.

Method `open_channel` opens the communication channel with `gnuplot`.

In the `process post_moment`, method `send_data` is used. It is invoked as `post_moment`, since simulation results can be sent **after** given time instant is solved. The number of arguments has to agree with number of non-zero arguments of `add_traces`.

In the process final, function `main_loop` is invoked. Without this function, `gnuplot` will close the drawing when the simulation is finished. Function `main_loop` leave program `gnuplot` active and gives you normal `gnuplot` prompt, so you can analyse the waveforms, create different output formats, etc., when the simulation is finished.

Class `GnuOnLine` can be used for all standard simulation problems. However, we have used Alecsis to solve partial differential equations (simulation of micromechanical sensors), and have also created on-line viewing for spatial 3D drawings. Class `GnuOnLine` was used there as a *base class* to create *derived classes* for specific problems.

## A2.5. math.h

We implemented all mathematics functions as instructions of virtual processor of simulation, so their execution does not require including this header nor appending of a library body. The following declarations are given so you can see what was implemented, and what was not:

```
/*************************************************************
 * This file contains prototype declarations for ALECSIS2.0
 * built-in math functions. Since they really work as
 * instructions, the declarations are necessary just that compiler
```

```
 * does not complain about arguments when invoked with -O
 * (optimizer) option. The simulator works O.K. without this file.
 ************************************************************/
#ifndef MATH_INCLUDED
#  define MATH_INCLUDED

     extern double acos(double);
     extern double asin(double);
     extern double atan(double);
     extern double atan2(double, double);
     extern double cos(double);
     extern double sin(double);
     extern double tan(double);
     extern double cosh(double);
     extern double sinh(double);
     extern double tanh(double);
     extern double exp(double);
     extern double log(double);
     extern double log10(double);
     extern double pow(double, double);
     extern double sqrt(double);
     extern double ceil(double);
     extern double fabs(double);
     extern double floor(double);

     extern int abs(int);

// These constants may be of some help in modelling

#  define M_E          2.7182818284590452354
#  define M_LOG2E      1.4426950408889634074
#  define M_LOG10E     0.43429448190325182765
#  define M_LN2        0.69314718055994530942
#  define M_LN10       2.3025850929940456802
#  define M_PI         3.14159265358979323846
#  define M_PI_2       1.57079632679489661923
#  define M_PI_4       0.78539816339744830962
#  define M_1_PI       0.31830988618379067154
#  define M_2_PI       0.63661977236758134308
#  define M_2_SQRTPI   1.12837916709551257390
#  define M_SQRT2      1.41421356237309504880
#  define M_SQRT1_2    0.70710678118654752440
#endif
```

## A2.6. unistd.h

An equivalent of UNIX `unistd.h` file. There is no appropriate library. Here are declarations of functions for forking and pipelining, used to enable simultaneous simulation and viewing of results (see section on `gnulib.h`).

```
#ifndef _UNISTD_INCLUDED
# define _UNISTD_INCLUDED

/*
 * unistd.h
 * symbolic constants and structures which are used
 * for support of the /usr/group standard.
 *
 */
```

```
#  ifndef NULL
#    define NULL  0
#  endif

#  ifndef R_OK
/* Symbolic constants for the "access" function: */
#    define R_OK  4
#    define W_OK  2
#    define X_OK  1
#    define F_OK  0
#  endif

/* Symbolic constants for the "lseek" function: */
#  ifndef SEEK_SET
#    define SEEK_SET   0
#    define SEEK_CUR   1
#    define SEEK_END   2
#  endif
#  include <time.h>

       extern int close(int);
       extern int dup(int);
       extern int execl(const char *, const char *, ...);
       extern int execv(const char *, const char **);
       extern int pipe(int *);
       extern int read(int, char *, int);
       extern int write(int, const char *, int);
       extern int fork();
       extern int _wait();
       extern int select(int, int*, int*, int*, struct timeval*);

#endif
```

## A2.7. varargs.h

Two macros for work with functions and `action` blocks with variable number of arguments (no library body) are in the file `varargs.h`. Explanations of how to use them are in Chapter 4 for functions, and in Chapter 5 for `action` parameters. Meaning of definition of `DWORD_ALIGNMENT` is explained there, too.

```
#ifndef VARARGS_INCLUDED
#  define VARARGS_INCLUDED
//     ALECSIS2.0 Header file
//     (from standard C "varargs.h")

// Use __mode int for char and short types.

# define TYPE_DOUBLE 8
# define DWORD_SIZE  8

# define va_start(__list, __par) (__list = ((char *) (&__par) +
sizeof(__par)))
# ifndef DWORD_ALIGNMENT
#   define va_arg(__list,__mode) ((__mode *)(__list += \
                                      sizeof(__mode)))[-1]
# else  /* DWORD_ALIGNMENT */
#  define va_arg(__list,__mode) ((sizeof(__mode)==TYPE_DOUBLE && \
                                    ((int)__list%DWORD_SIZE) ) ? \
                                ((__mode *)(__list+=sizeof(__mode)+ \
                              (DWORD_SIZE-((int)__list%DWORD_SIZE)))) \
                : ( (__mode *)(__list += sizeof(__mode)) )  )[-1]
```

```
# endif /* DWORD_ALIGNMENT */

# endif /* VARARGS_INCLUDED */
```