Appendix 1
Alecsis installation and use

A1.1. Alecsis installation

Alecsis is program for UNIX operating system. Up to now, it was installed on following workstations:

- IBM RISC (AIX operating system),
- HP 9000s300/400 (HPUX operating system),
- HP9000s700/800 (HPUX operating system),
- Silicon Graphics (IRIX operating system),
- SUN Sparc (SUNOS operating system).

Installation is performed also for IBM PC (LINUX operating system) but is still considered shaky.

It is delivered on a single floppy disk, as a single file `alecsis.tar.Z`. File is compressed using UNIX `compress` command, and archived using UNIX `tar` command. When the archive is opened, directory `alecsis` is created, with following subdirectories:

- `src2.3` - source code;
- `include` - standard header files;
- `sys` - standard libraries (in AleC++ code);
- `lib` - standard libraries (compiled into Alecsis object code);

- `bin` - executables;
- `agnu1.1` - waveform display program (explained in separate Appendix);
- `alm` - Alecsis Library Manager (explained in separate Appendix);
- `nrl` - programs for postprocessing of digital simulation results (explained in separate Appendix);
- `p2a` - PSpice2Alecsis converter (explained in separate Appendix).

A1.1.1. Paths for UNIX shell

It is necessary to make Alecsis executable, Alecsis include files and Alecsis libraries visible from your working directory. The most convenient way is to define these paths in your UNIX shell. If you are using C-shell, you can modify the `.cshrc` file in your home directory. This is to be done before compiling Alecsis source code, as Alecsis `Makefile` uses some shell variables described here.

If Alecsis executable file `alec` is stored in directory `alecsis/bin` (which is default), you should add in your `.cshrc` file the following line:

```
set path = ($path $HOME/alecsis/bin)
```

assuming that directory `alecsis` is unpacked below your home directory `$HOME`. If you have stored executable file `alec` in some other directory, the appropriate path has to be defined.

It is also necessary to put command:

```
setenv ALEC_HOME $HOME/alecsis
```

in your `.cshrc` file. This makes standard header files located in `$HOME/alecsis/include`, and standard libraries located in `$HOME/alecsis/lib` visible. Header files, containing declarations, should have extension '`.h`' (like in C/C++). Header files, included using '<' and '>' as parentheses, would be searched for in directory `$HOME/alecsis/include`, e.g.

```
# include <alec.h>
```

List of directories where Alecsis search for included files can be expanded from the command line, using option '`-I`', explained later in this Chapter. If the file name is included using quotation marks, e.g.:

```
# include "header.h"
```

it will be searched for in current working directory.

Libraries contain definitions of modules and functions that are declared in header files. They are compiled into Alecsis object code, and have file extension '`.ao`'. Libraries included using `library` command (explained later in this Chapter) or '`-l`' command option (also explained later in this Chapter) are searched for in directory `$HOME/alecsis/lib`. Users often define their own libraries, and it is often necessary to search some other directories, too. These directories can be added using command option '-L' and shell variable ALEC_LIB_PATH. For instance:

```
setenv ALEC_LIB_PATH .:./lib:
```

specifies that definitions are searched for in: current directory '`.`'; directory '`./lib`'; and directory '`/usr/cad/alecsis/ttl`'. Colon '`:`' is used as a separator. In this example, you can see that the library

directory can be specified using relative or absolute path. You should at least specify current directory '.' using ALEC_LIB_PATH. So, you have to put into your .cshrc file:

```
set path = ($path $HOME/alecsis/bin)
setenv ALEC_HOME $HOME/alecsis
setenv ALEC_LIB_PATH .
```

A1.1.2. Compiling Alecsis source code

To compile Alecsis source code, go to directory src2.3 and type make. The Makefile will give you info about compilation on different hardware workstations. The Makefile can be modified for installations on workstations that are not on the list given above, too. Different flags are explained in the Makefile. However, we it can happen that some interventions in the source code are necessary for installation on different hardware platforms.

There is one part of the Makefile that should be edited in any case. It regards paths to the C libraries, location where executable files are stored, etc.

A1.1.3. Compiling Alecsis standard libraries

Alecsis standard libraries are given in directory alecsis/lib as compiled file -- Alecsis object code binaries (extension '.ao'). They are also given in AleC++ source code in directory alecsis/sys (extension '.ac' or '.hi'). AleC++ compiler works in the same manner on any workstation, but there might be some differences in the way data are stored on different workstations. For that reason, after Alecsis installation, libraries ought to be recompiled. Libraries are compiled using options '-c' and '-O':

```
alec -c -O file_name
```

Option '-c' means that files compiled (not interpreted), so that the file with object code is created. Option '-O' turns the optimizer on. For instance, file alloc.hi is compiled using:

```
alec -c -O alloc
```

which creates library alloc.ao. Libraries (files with extension '.ao') must be then moved to alecsis/lib directory.

The most important Alecsis standard libraries are explained in separate Appendix.

Note: For library management, special program alm (Alecsis Library Manager) is created. It is explained in separate Appendix.

A1.2. Alecsis use

Many aspect of Alecsis usage are already explained in this Manual. We will give here overview of Alecsis command line options, list of file name extensions, and some options for including precompiled libraies.

A1.2.1. Program call from the command line -- command options

The name of Alecsis executable file is **`alec`**. The program is invoked from UNIX command line by listing the name (`alec`), one or more input files, and desired options. There can be more input files on the command lines, but only one of them can be in the source (AleC++) code. Other must be object files, that are already compiled into Alecsis object code (file extension '`.ao`').

Alecsis normally creates file with extension '`.ar`' (Alecsis results), that contain results of the simulation. If it is invoked with '-c' option, it creates object files instead (extension '`.ao`'). That means, only compilation of AleC++ code is executed, not the interpretation of the compiled code.

Object files are similar to Alecsis libraries (extension '`.aa`'), which also contain compiled Alecsis object code. The only difference is in processing of their content. **See Appendix on Alecsis Library Manager (`alm`) on how to create and manage libraries.** Libraries have a symbol table of contents at the beginning enabling fast search. If the library is appended using option '`-l`', the desired entity from it (module, function, etc.) will be loaded to memory only if referenced as a global signal (in the linking phase). However, object files listed as arguments on the command line appear **in full** in the memory (not selectively). If the object file contains the **`main`** function (C/C++ -like), Alecsis can execute it. That means Alecsis can interpret previously compiled files - no source code in AleC++ (not compiled) is necessary. However, `root module` cannot be compiled and placed in libraries.

Alecsis 2.3 supports the following options (in alphabetical order):

`-a` listing of library content (no compilation or execution).

`-c` compilation of the original file (without execution). The compilation of file with name *`name`*`.ac` (or *`name`*`.hi`) will produce a file named *`name`*`.ao`.

`-cl` multiple searching of libraries (cycle library). The order of libraries listed is not important with this options since linker would search again in case of an undesired outcome.

`-D`*`symbol<=token>`* equivalent to the command in the code "`#define symbol <token>`".

`-E` Alecsis executes the preprocessing phase only and prints the result in `stdout`.

`-g` appending of object library containing information for location of errors (in compiling), and giving the information on names and number of lines of the user code with the fatal error during the simulation, instead of the system announcement of type "bus error" or "segmentation fault". If able, simulator will give the content of the working stack at the time of fatal error (the order of function calls). Useful for debugging.

Note: Option '`-g`' helps in debugging AleC++ code used in processes. Errors that appear as a result of inconsistent system of equations, for instance, cannot be debugged in this way.

`-i` no development of `inline` function. All inline functions are compiled as non-inline and `static` (this allows the existence of functions with the same name and the same parametric profile in another library).

`-I`*`dir`* expansion of the list of directories where the library appended using `include` command can be found.

`-l`*`lib`* appending the library *`lib`*`.ao` to the linker list. During the resolution of unresolved external symbols, linker will successively search the libraries for these symbols. You can list desired number of libraries using this option more times. The order in the list is important since linker does not return to an already-searched library (this is important if

one external symbol refers to another unresolved external symbols). Option '`-cl`' (cyclic search of libraries) solves this problem.
Option '`-l`' is equivalent to command `library` in the AleC++ code.

`-L`*`dir`* expansion of the list of directories where the library can be found. This option is equivalent to setting environment variable `ALEC_LIB_PATH`.

`-o` *`file`*`.ao` the result of compilation is placed in the file listed after the option. If the file contains the function `main`, the command "`alec` *`file`*`.ao`" can execute it.

`-O` code optimization. Beside reducing the number of instructions, this instruction causes a number of useful warnings of variable masking, absence of function prototypes, etc.

`-over` ban on operator overload.

`-r` no recursion. It is recommended for non-recursive functions because it gives faster code.

`-S` creation of an assembler file *`name`*`.as` from the given source file *`name`*`.ac`.

`-stat` printing of the separate output library (*`name`*`.stat`) containing data on the activity of digital circuit part during the simulation (the number of events and the number of processes in every moment).

`-t` printing of the duration in CPU seconds for particular phases of the program (compilation, linking, preparation and execution of the simulation).

`-v`*verbose_level* Gives more information about the simulation run. There are following options:

- `-v1` tracks symbol table activity
- `-v2` tracks intermediate code generation (operand types etc.)
- `-v3` all LEX tokens are printed out as they arrive
- `-v4` follows voltage generator/inductor loops detection
- `-v5` prints instructions as they are flushed
- `-v6` tracks overloading and prototype mangling
- `-v7` prints list of nodes
- `-v8` follows the use of weights if option `dcon` is used
- `-v9` follows the process of static/global initialization
- `-v10` follows library management
- `-v11` follows function declaration
- `-v12` clear global symbol table before simulation
- `-v13` tracks function prototype existence
- `-v15` tracks class member access control
- `-v16` follows function inline expansion
- `-v31` prints system matrix and right-hand side vector in every iteration, as without reordering
- `-v32` prints system matrix and rhs vector in every iteration as reordered.
- `-v33` prints **both** non-reordered and reordered system matrix and rhs vector, respectively, in every iteration
- `-v55` turns on full logic initialization
- `-v99` changes all calls to exit() with abort() to dump core file

Note: Verbose level `55` (full logic initialization) is rather a simulation option than a verbose level, and it will be organized as such in following versions of Alecsis.

Most of these verbose levels are of interest only for us that created Alecsis, for our debugging purposes. However, there are some of them that can be very useful for Alecsis users. For instance, **verbose level `8` follows use of weight when option `dcon` for difficult convergence problems is used**. This can be very useful for setting correct values for options `max_weight`, `min_weight`, `p`, `q`, and `maxdcon`, if you are not satisfied with their default values (see Chapter 5, section on simulation `options` for details).

Verbose level `31` prints out system matrix, which can sometimes be helpful if you have problems with zero pivot (singular matrix). This is, however, useful only for small matrices, as it is very difficult to analyze large matrices.

Note: If more than one *verbose_level* is given, only the last one will take effect. For example:

```
alec -v1 -v2 file_name
```

has the same effect as:

```
alec -v2 file_name
```

A1.2.2. File name extensions

List of file name extensions is given in Chapter 2, but is given here again for completeness. The names of Alecsis input files are arbitrary (the maximal name length is determined by the specific operating system,) but they have to have the extension `.ac`. Extension `.hi` is also allowed for compatibility with version 1.0. File name extensions are the following:

`ac` - Alecsis input file

`hi` - Alecsis 1.0 input file (accepted by newer versions, too)

`h` - Alecsis header file (as in C/C++)

`ar` - Alecsis results (Alecsis output file, Agnu input file)

`ao` - Alecsis object-code file (compiled input file)

`as` - Alecsis assembly language file (created by compiler using option `-S`)

`aa` - Alecsis library

`stat` - Alecsis statistics file (creted when command option `-stat` is used)

A1.2.3. Listing of libraries in the source file -- command library

Beside from the command line (option '`-l`'), you can list the appended libraries in the source file, as well. The keyword used for that is **`library`**. You can list an unlimited number of libraries separated by a coma, ending with semicolon:

```
library lib1, lib2, lib3, "lib4";
```

The command can appear many times in the text, while the result is a union of all separate lists. The command can be anywhere in the text if on global level (outside functions, modules, etc.). The order of listing of libraries is important unless you use the option '`-cl`'. You can use this command in conjunction with option '`-1`'. Libraries are used in the linking phase, so command `library` should be used in the file where your `main` function or the `root module` is.

The library name can be in quotation marks (`"lib4"` in example above) if the name of the library is masked by some other name in the present context.

A1.3. Overview of Alecsis versions

We use notation of Alecsis versions with tree numbers. First number denotes crucial change of Alecsis/AleC++ functionality. The second one denotes change of functionality (new feature) from the user point of view. The last number is denotes improvement (usually debugging) of existing functions.

Alecsis 1.x	- input language based on C, no object-orientation.
Alecsis 2.1.1. - 2.1.50	- object-oriented input language AleC++ is introduced
Alecsis 2.2.1. - 2.2.33.	- operator `d2dt2` is introduced
Alecsis 2.3.1. - 2.3.x	- through and across `eqn` statements are introduced