ALDgen: Generate IBM 1401 ALD (Automated Logic Diagrams)

The ALDgen program generates IBM Automated Logic Diagrams, or ALDs.

The ALD page

ALD pages are described in IBM document 223-6875-1, beginning on page 12.

An ALD page contains 198 lines of 120 columns. It was originally printed by a specially-adapted 1403 printer, at twelve lines per inch. Each page is 16.5 inches tall (34 1403 sprocket holes).

The top line contains:

- Columns 1-7: A numeric drawing number.
- Columns 8-94: A centered title.
- Columns 95-98: A machine number, such as 1401.
- Columns 109-118: A logic page identifier, consisting of four numeric fields separated by dots. The first three fields are two digits each, and the fourth is one digit.

The second line is blank.

The next nine regions of 18 lines each, comprising 162 lines altogether, contain the input connections, output connections, logic boxes, and lines that connect them.

Horizontally, each region is divided into seven column groups.

1 Columns 1-15: Input connections.

2-6 Columns 16-31, 32-48, 49-65, 66-82, and 83-105: Logic boxes with space for traces between them.

7 Columns 106-120: Output connections.

There are six row groups, each containing three lines, in each of the nine regions.

In column groups 1 and 7, the first two lines are nomenclature, beginning with a + or - sign, followed by a single letter in MNPRSTUVWXYZ, that specifies the signal level. The first line might be blank. The third line is a logic page identifier. In column group 1, the logic page identifier is followed by a two-column trace, an asterisk, and a tag (letter or digit) for an edge-connector, described below, or a four-column trace. In column group 7, the logic page identifier is preceded by an asterisk, an edge-connector tag, and a two-column trace, or a four-column trace.

Column groups 2-6 can contain logic boxes, in columns 22-31, 39-48, 56-65, 73-82, and 90-99, consisting of 18 lines:

1. Can be used for horizontal traces.

2. Always blank

3. Optional centered 10-character title.

4. Horizontal line.

5. Symbol (SMS card type).

6. Machine function index (e.g. TD=tape drive; two dots or BA for the basic circuit.)

7. Mode MNPRSTUVWXYZ Fig. 13 on page 15.

8. Modular I: FFME F = frame 01-99, M = module A B, E = Engineering Change level A-Z.
   Modular II: FFGE F = frame 01-99, G = gate A-D, E = Engineering Change level A-Z.

9. Modular I: GCRR G = gate 1-8, C = column A-F, R = row 01-26.
   Modular II: HRCC H = Chassis 1-4, R = row A-K, Column = 01-28.

10. SMS Card Code AA-ZZ, assigned in that order, excluding I and O, Cap Connection ZZ-AA, assigned in that order, excluding I and O, or -- if not used.

11. Line with two digits of block configuration centered Fig. 12 on page 14.

12. Box coordinates on the page.

14-18 Can be used for horizontal traces.

If the column group does not contain a logic box, lines 1 and 3-18 can be used for horizontal traces.

Between column groups 1 and 2 or between columns 6 and 7, six columns can be used for horizontal or vertical traces. Between other successive pair of column groups, seven columns can be used for horizontal or vertical traces.

After at least one blank line, up to 18 lines of edge connectors are listed, six per line (166-183). Each edge connector is preceded by a six-character horizontal line if it is not tagged in an input or output connection, or two blanks, an asterisk, a one-character note reference, and a two character horizontal line. Edge connector nomenclature occupies eight columns. Before the second through sixth edge connector listing on each line, there are five blanks. So an edge-connector line occupies at most 106 characters.

After at least one blank line there is a block containing up to 14 lines (185-198) of comments in columns 1-39, and up to four groups of engineering change numbers per line. Each group of engineering change numbers begins with a blank and a vertical line, a tag number, a space, an eight-character date, a space, and an engineering change designator consisting of six digits and a letter.

Input to the ALDgen program

Input to ALDgen consists of *specifications* of the form

[*label:*] *spec-name, field [, field ...]* .

Where the *label:* is optional on some specifications, and required on others. Each *field* consists of

field-name = value.

The *value* can be just one item, or a sequence of items, enclosed between square brackets and separated by commas.

All text is case insensitive.

A specification may be continued onto the following line by an ampersand or dollar sign.

A comment begins with a semicolon, not within a quoted string, and extends to the end of the line. To embed a comment as a separate line between lines of a specification, begin it with an ampersand or dollar sign.

An item can be one of

- an identifier that begins with a letter and consists of letters, digits, and underscores,
- a number,
- a string enclosed in apostrophes or quotation marks,
- a page coordinate consisting of a digit in the range 2-6 followed by a letter in the range A-J except I, or
- one of the above preceded by * (asterisk), ∼ (tilde), - (minus), or + (plus).

ALDgen recognizes ten specifications:

Block Description of an SMS block.

A `Block` specification is required to have a label. The first character must be a digit in 2-6. It specifies the column group, the horizontal position on the page. The second must be a letter in A-H or J. It specifies the vertical position on the page. This appears below the box.

A `Block` specification can have the following fields. Unless otherwise specified, all fields are required.

- `title`: Up to ten characters. This appears above the box. This field is optional.
- `symbol`: Up to four characters, an SMS description. This appears in the first line of the box.
- `func`: Up to four characters, the functional unit description, such as BA for basic unit, TD for tape drive, OVLP for overlap. This appears in the second line of the box.
- `mode`: Either four characters, which must be a string if it contains embedded blanks, or two items of at most two characters each, separated by a dot. This gives the signal levels. The first two characters are for inputs, and the second two are for outputs. If

either of the two parts consists of only one character, it applies to all inputs or outputs. Otherwise, the first character applies to the top three, and the second character applies to the bottom three. the letters must be in MNPRSTUVWXYZ. This appears in the third line of the box.

- `frame`: Two digits. These are the first two characters of the fourth line of the box.
- `module`: A or B. This is the third character of the fourth line of the box. It is used only if the form is Modular I.
- `ec`: One letter, the engineering change level. This is the fourth character of the fourth line of the box.
- `gate`: Modular I: One digit, 1-8. This is the first character of the fifth line of the box.
  Modular II: One letter A-D. This is the third character of the fourth line of the box.
- `column`: Modular I: One letter. This is the second character of the fifth line of the box.
  Modular II: Two digits 01-28. This is the third and fourth characters of the fifth line of the box.
- `chassis`: One digit in 1-4. This is the first column of the fifth line of the box. This is used only if the form is Modular II.
- `row`: Modular I: Two digits. These are the third and fourth characters of the fifth line of the box.
  Modular II: One letter in A-K. This is the third column of the fifth row of the box.
- `card`: Two letters, an SMS card identifier. These are the first two characters of the sixth line of the box.
- `cap`: Two letters, or a string containing two hyphens enclosed in apostrophes or quotation marks, the cap connections. These are the last two characters of the sixth line of the box. This field is optional.
- `block`: Two digits. These appear as the middle two characters in the line that forms the bottom (seventh) line of the box.

`show` : Show an unconnected pin. Two fields separated by a dot. First a number in 1-8 (on the left) or 9-16 (on the right). Then one or two letters in `ABCDEFGHJKLMNPQR`. The first of these is a pin connection; the second is a tie-down pin on the backplane.

Comment Comments to be displayed at the bottom of the page.

A `Comment` specification has only one required field.

- `text` Up to fourteen quoted strings, each containing no more than 39 characters.

Dot A "dot" used to indicate that lines are connected.

A `Dot` specification should have a label so that it can be referenced by `Line` specifications. A `Dot` specification has only one required field.

- `at`: A value of the form `C.H.V` where `C` is a logic box coordinate location, `H` is a horizontal position within the region, and `V` within the region is a vertical position. The logic box coordinate location need not be the label of a `Block` specification. If the first digit of the coordinate is 1, the horizontal position must be in 2-7. If the first digit of the coordinate is 6, the horizontal position must be in 1-6, otherwise 1-7. The horizontal position is the column number after the one where the second character of a logic block connection could appear. The vertical position must be in 1-18.

EC An engineering change to be listed at the bottom of the page, on the same lines as comments.

An `EC` specification has three required fields.

- `date`: An eight-character quoted string consisting of a two-digit month, a two-digit day, and a two-digit year, separated by hyphens.
- `number`: A six-digit number, or a value of the form `N.A` where `N` is a six-digit number and `A` is a letter.
- `tag`: A single letter.

Edge An edge connector to be listed at the bottom of the page, above comments and engineering changes.

An `Edge` specification has only one required field.

- `text`: An eight-character identifier or quoted string – probably usually a quoted string because the nomenclature begins with a digit and contains letters, such as `01B4D04J`.

Input Description of an input signal at the left edge of the page.

An `Input` specification is required to have a label beginning with `I`, followed by a letter in `A-HJ` to indicate the box region, and a digit in 1-6 to indicate the vertical position within that region. An `Input` specification has the following fields. Unless otherwise specified, all are required.

- `edge`: Either an eight character identifier or quoted string as described above for the `text` field of an `EC` specification, or a value of the form `E.T` where `E` is such an identifier and `T` is a one character tag. This field is optional. If it appears, the edge connector and tag will be listed at the bottom of the page and the tag will be shown adjacent to the third line, as described above.
- `level`: + or -, followed by a letter in MNPRSTUVWXYZ.
- `logic`: A logic page identifier consisting of four numeric fields separated by dots. The first three are two digits each and the fourth is one digit.
- `text`: A quoted string, or two quoted strings separated by a dot. If only one appears, it cannot contain more than twelve characters. If two appear, the first cannot contain more than twelve characters and the second cannot contain more than fifteen characters.

Line Description of a line starting from an `Input` specification, `Block` specification, or `Dot` specification, and ending at an `Output` specification, `Block` specification, or `Dot` specification.

A `Line` specification has two required fields, `from` and `to`, of the same form. The `from` field can contain only a single item. The `to` field can contain an arbitrary number of items, enclosed in square brackets and separated by commas, provided the `from` field identifies a `Dot` specification.

An item may be the label of a `Dot`, `Input`, or `Output` specification, `B.#.P`, `B.#.P.R`, or `L.R`. The label of an `Input` specification can appear only in the `from` field. The label of an `Output` specification can appear only in the `to` field.

`B` is the label of a `Block` specification.

`#` is a number in 1-16, indicating a row of the logic block (1-8 on the left, 9-16 on the right). `#` = 1 or 9 corresponds to row 4 of the logic box as described above, with larger values corresponding to succeeding rows.

`L` is the label of a `Dot` or `Output` specification.

`P` consists of one or two letters in `ABCDEFGHJKLMNPQR`. The first of these is a pin connection; the second is a tie-down pin on the backplane. The `P` field is shown at the boundary of the logic box in row `#`.

`R` can only appear in a `to` field. It indicates a route for the line to follow. It consists of alternating letters and numbers, with `D` (for down), `L` (for left), `R` (for right), or `U` (for up). The number is the number of columns (for `R` or `L`) or rows (for `D` or `U` to move. The route needs to be specified only though the direction of the penultimate segment. The direction and length of the ultimate segment are not used. For example, for a two-segment path that goes upward and then left or right, `U` is sufficient. For a three-segment path that goes up 6 rows, then right, then down, `U6R` is sufficient.

The `B.#.P[.R]` item may be preceded by an asterisk to indicate a coaxial cable or twisted-pair connection, in which case the edge of the box for row `#` is printed as a solid box instead of a vertical line.

Multi A specification that vertically-adjacent blocks in the same column are to be joined by two vertical lines, to indicate that they are parts of a compound circuit.

A `Multi` specification has only one required field.

- `block`: two or more block labels, separated by commas and enclosed in square brackets. The horizontal positions of the blocks must be the same, and the vertical positions must be contiguous, but not necessarily in order.

Output Description of an output signal at the right edge of the page.

An `Output` specification is required to have a label beginning with `O`, followed by a letter in `A-HJ` to indicate the box region, and a digit in 1-6 to indicate the vertical position within that region. The fields of an `Output` specification are the same as for an `Input` specification, except that the maximum numbers of characters in the `text` items are one less.

Page Information to display at the top of the page.

A `Page` specification has four required fields.

- `logic` A logic page identifier consisting of four numeric fields, separated by dots. The first three fields are two digits each, and the fourth is one digit.
- `machine` A numeric machine identifier consisting of up to four digits.
- `number` A seven-digit drawing number.
- `title` A quoted string of up to forty characters.

Each ALD page begins with a `page` specification, and continues to the end of the file, or the next `page` specification. Otherwise, there is no restriction on the order of specifications.

How the program works

The program is an interpreter of the "little language" described above.

It uses conventional compiler-construction methods and tools.

Input is read by a *lexer* that groups characters into *tokens*, analagous to parts of speech in natural languages. Some of the tokens are "parts of speech," such as operators or punctuation, that have only one member. These are called "true terminal" symbols. Others, such as numbers and identifiers, have several members. These are called "pseudo terminal" symbols.

Characteristics of each category of tokens are specified by *regular expressions*. These look like algebraic expressions, but the operator symbols have different meanings. Adjacent non-operator terms are concatenated. An infix | operator means "or." A suffix * operator means "zero or more." A suffix + operator means "one or more." A suffix ? operator means "zero or one." Subexpressions may be grouped. For example, the expression for identifiers is

```
letter ( letter | digit | '_' )+
```

where `letter` and `digit` are the names of classes of characters.

A program called a *lexer generator* transforms regular expressions into a table that represents a computational device called a *deterministic finite automaton*. One can think of this as a roadmap, where each place has zero or more roads leading away from it, each labeled by a class of characters, and some places are designated as *final* places, meaning the lexer has recognized a token.

Characters are stored in a `character table`. The end of each token in the character table is given by a number in a `string table`, in which the value of the zero$^{\text{th}}$ element is zero.

As characters are read, they are added to the end of the character table. When the lexer recognizes a pseudo-terminal token, it is looked up in the string table. If it is not found, a new string is created. Otherwise, the found string is used. Thereafter, a token's text is indicated by its position in the string table, called its *string index*; no further searching of the string table is necessary. True terminal tokens do not need a string index because their text is always known.

Tokens are then grouped into statements, analogous to sentences in natural language, by a component called a *parser*. The result of parsing is an *abstract syntax tree*, analogous to a sentence diagram for a natural-language sentence. It is called *abstract* because some of the steps of the parsing process are not represented.

Structural forms of the abstract syntax tree are represented by internal vertices. Pseudo-terminal symbols are leaves of the tree. True terminal symbols are not represented. Different structural forms are represented by different labels of the internal vertices.

We represent a tree (or subtree) in prefix form, that is, the root of the tree, followed by its immediately-adjacent members (herein called *sons*, by analogy with family trees), and the entire tree (or subtree) enclosed in angle brackets.

The result of parsing a labeled specification is a subtree of the form

< `label` *identifier spec-tree* >,

where *spec-tree* is of the form

< `spec` *spec-name field-tree* ... >,

and *field-tree* is of the form

< `field` *field-name field-value* ... >.

The *field-value* is an expressions represented by trees of the form

< `asterisk` *item* >,
< `minus` *item* >, or
< `plus` *item* >

where each *item* is a *simple-item* or

< `dot` *simple-item* ... >.

and each *simple-item* is an identifier, a logic box coordinate, a number, or a character string.

The abstract syntax tree is traversed to check the types and values of all the fields. To aid this process, a *declaration table* is constructed. Elements of the declaration table are referenced by elements of the *symbol table*, which is indexed by the string index.

A declaration indicates the usage of a symbol. There are no reserved words, so a symbol might have several declarations, for example as a label, a specification name, or a field name.

To aid the type-checking process, labels are entered into the declaration table. A field of the declaration table gives the index of the `spec` node for the specification that the label labels. When a label is expected, the string index of the symbol is used to scan the declarations for that symbol. If a label declaration is found, the tree index of its declaration is put into a field of the label-reference tree node, called a *declaration*, so that subsequent processes no longer need to access the declaration table.

After type checking is completed, another process copies some information into data structures that ease subsequent analysis, in particular checking that signal levels at the ends of `Line` specifications are consistent. When a table element is constructed for a specification, the first son of the `spec` node, i.e., the *spec name* node, is decorated with the index of the table element.

After data from the specifications are entered into the tables, in particular signal information for`Block`, `Input`, `Line`, and `Output` specifications, signal information for `Dot` specifications are checked. It is verified that all `from` fields of `Line` specifications that designate a `Dot` spec have the same signal level (but the sign isn't checked if a `from` or `to` field references a `Block` node). This is an iterative process because the `from` and `to` fields of a `Line` spec might both be `Dot` specs. Eventually, all such specifications have signals computed. Then the signal levels at the ends of `from` and `to` fields are checked (but the sign isn't checked if a `from` or `to` field references a `Block` node).

After everything is checked, the ALD page is drawn by traversing the abstract syntax tree.