MIX - Micronas Interconnect Specification Expander

Userreference

6.10.2003

1 MIX - Micronas Interconnect Spec expander

2 Introduction

3 Management Summary

Contents

4 Installation

The MIX toolset uses the Perl scripting language. Please install a recent version of Perl on your workstation, e.g. the freely available ActiveState Perl from K:\PROJECTS\MIX\PROG\ActivePerl-5.6.1.633-MSWin32-x86.msi. Apart from that no installation is required.
Start MIX from the network drive like described in the examples below. That will also make

sure, that you are using the most recent release automatically. After the Perl installation, you can run MIX. immediately. Open a command shell on your desktop workstation:

Start ⟶ *Run* ⟶ *cmd*

and type:

K:\Projects\MIX\PROG\mix_0.pl foo.xls

Caveat: Currently MIX is available on MS-Windows, only!

5 Getting Started

To receive useful results, MIX reads in various input spreadsheet data. At least you will need to prepare a description of your design hierarchy (**HIER**) and the connectivity sheet (**CONN**). Optionally MIX will convert a input/output sheet IO, listing input/output pads and iocells and how these are linked to the core logic into appropriate hierarchy and connection lists.

5.1 A simple example

To understand the usage of the various tables and options, a very simple example is shown and extended step by step. The simple example just has two components *a* and *b*, which are connected by the signal *sigfoo*. The two instances have a common parent *chip*.

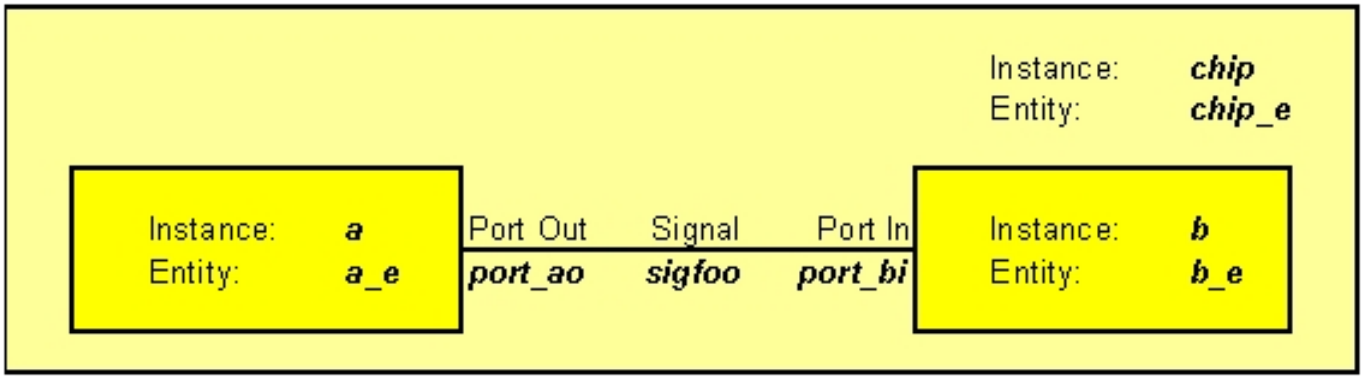


Image 1: mix_simple.xls

The equivalent description of this simple design in MIX is made up from a worksheet **HIER**.

::ign	::gen	::variants	::parent	::inst	::entity	::config	::comment
#Mix SimpleExample, V0.1, 20030630, hierachy							
		Default	TESTBENCH	chip	chip_e	chip_e_rtl_conf	
		Default	chip	a	a_e	a_e_rtl_conf	
		Default	chip	b	b_e	b_e_rtl_conf	

The first row defines the table headers names. The names have to be in the form "::NAME". Several of the columns are required, some are optional and you can define additional columns on your own. For HIER sheets the "::inst" column is the primary key. One design element will be generated for each new name of an ::inst row. If a name is defined several times, these lines will be overloaded or summarized, depending on built-in rules of the MIX converter.
The "::ign" column has to come first. If it starts with a #, the rest of this row will be ignored. All other columns can be added in arbitrary order.

The second required worksheet is **CONN**:

::ign	::gen	::bundle	::class	::clock	::type	::high	::low	::mode	::name	::out	::in	::descr	::comment
# Mix SimpleExample, V0.1, 20030630, hierachy													
		Chip	Data	Clk	std_ulogic			S	sigfoo	a/port_ao	b/port_bi	Simple	

As you see, this worksheet also starts with the table header definition line. The primary field is the "::name" column. The "::in" and "::out" columns are used to define the drivers and loads for the signals.

5.2 Mix it!

Run the MIX converter tool in the directory the excel spread sheet is stored in:

```
$ cd \work\MIX\doc\parts\simple
$ K:\Projects\MIX\PROG\mix_0.pl mix_simple.xls
```

This reads in the design description and evaluates the various sheets. It creates output files with an intermediate excel design description (`mix_simple-mixed.xls`) and the same data in a internal format (`mix_simple.pld`). A log file (`mix_0.pl.log`) and the HDL output files are written in the same run. Image 2 shows a screenshot of the `mix_simpe.xls` conversion. Only the most important errors and warnings are written to the screen, while a lot of information will be written to the log file. Search for the keywords "ERROR" and "WARNING" to verify proper conversion.

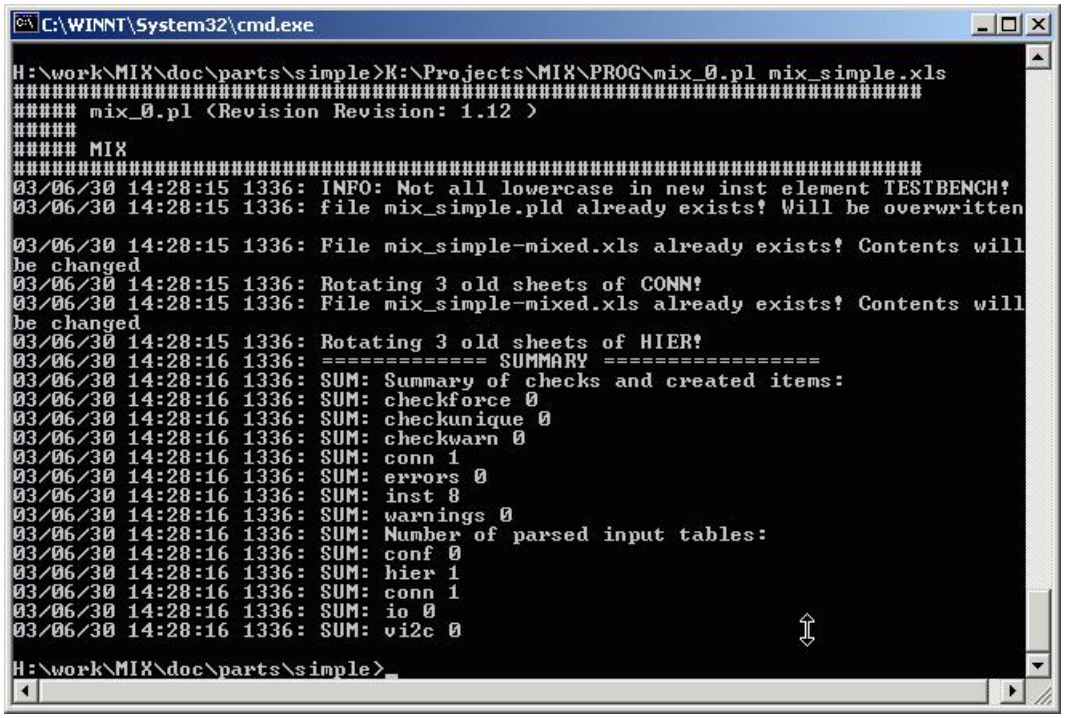
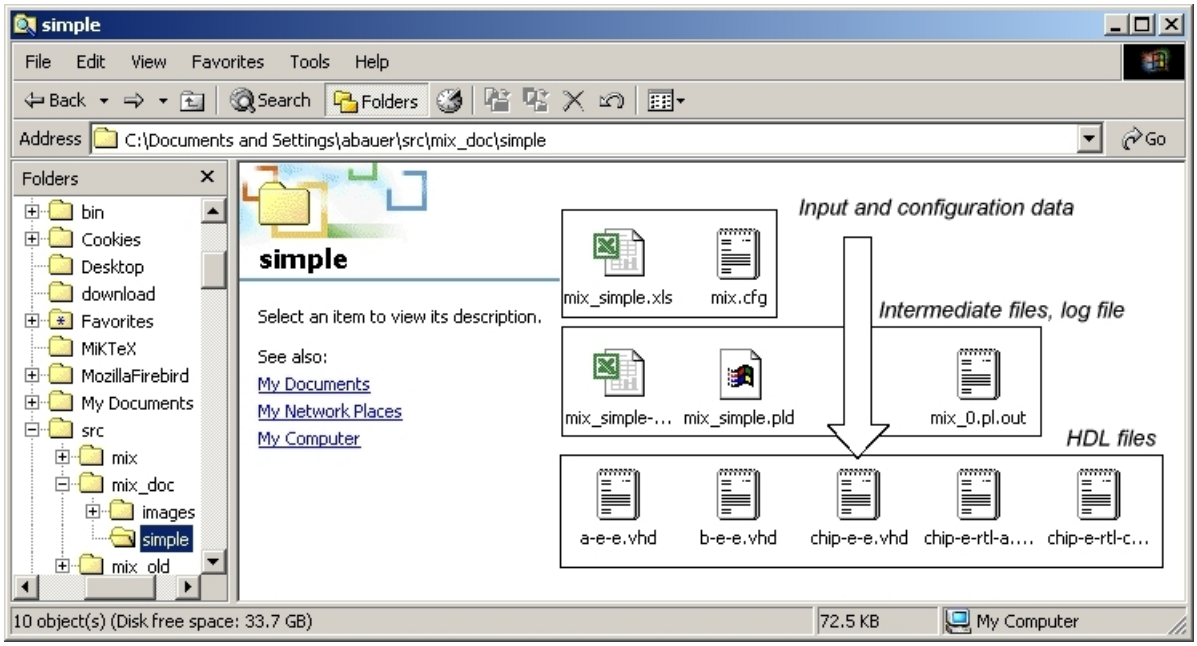


Image 2: mix_simple conversion

All output files are stored in the current working directory. Old versions of the output files are overwritten. Except the log file that is appended by each converter run. The intermediate excel description file keeps a history of previous HIER and CONN sheets by rotating _N extended worksheet names.



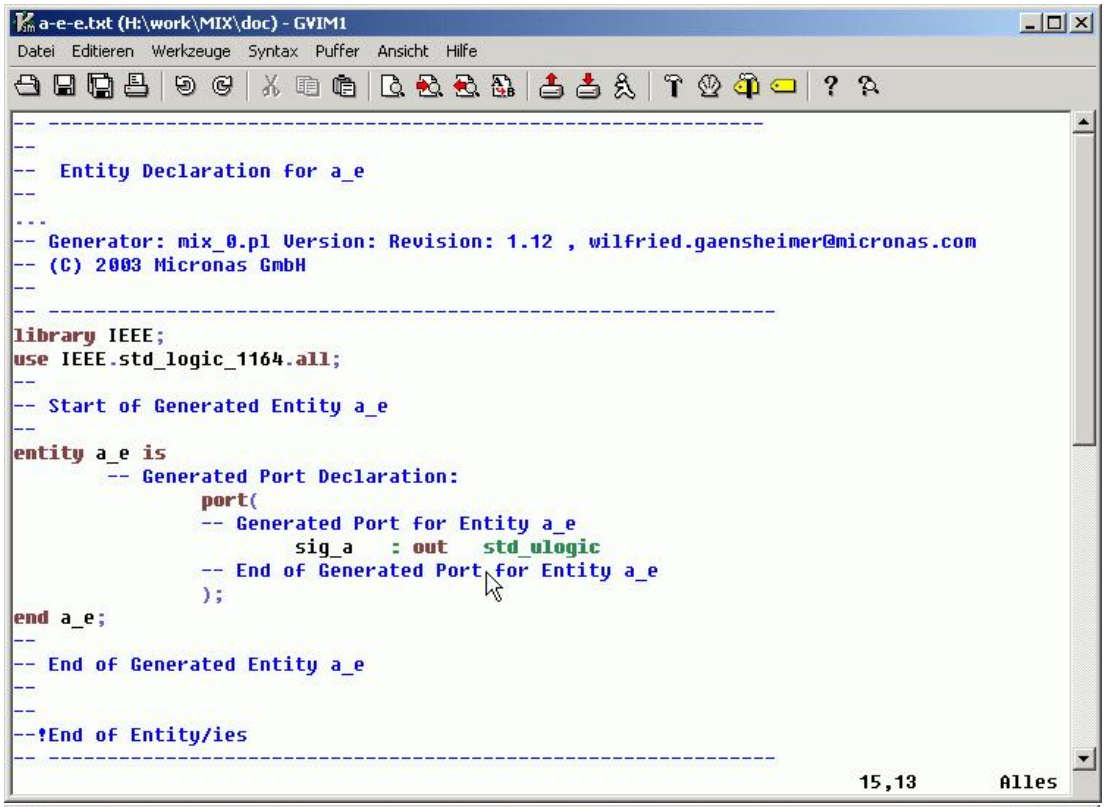
5.2.1 You get what you typed

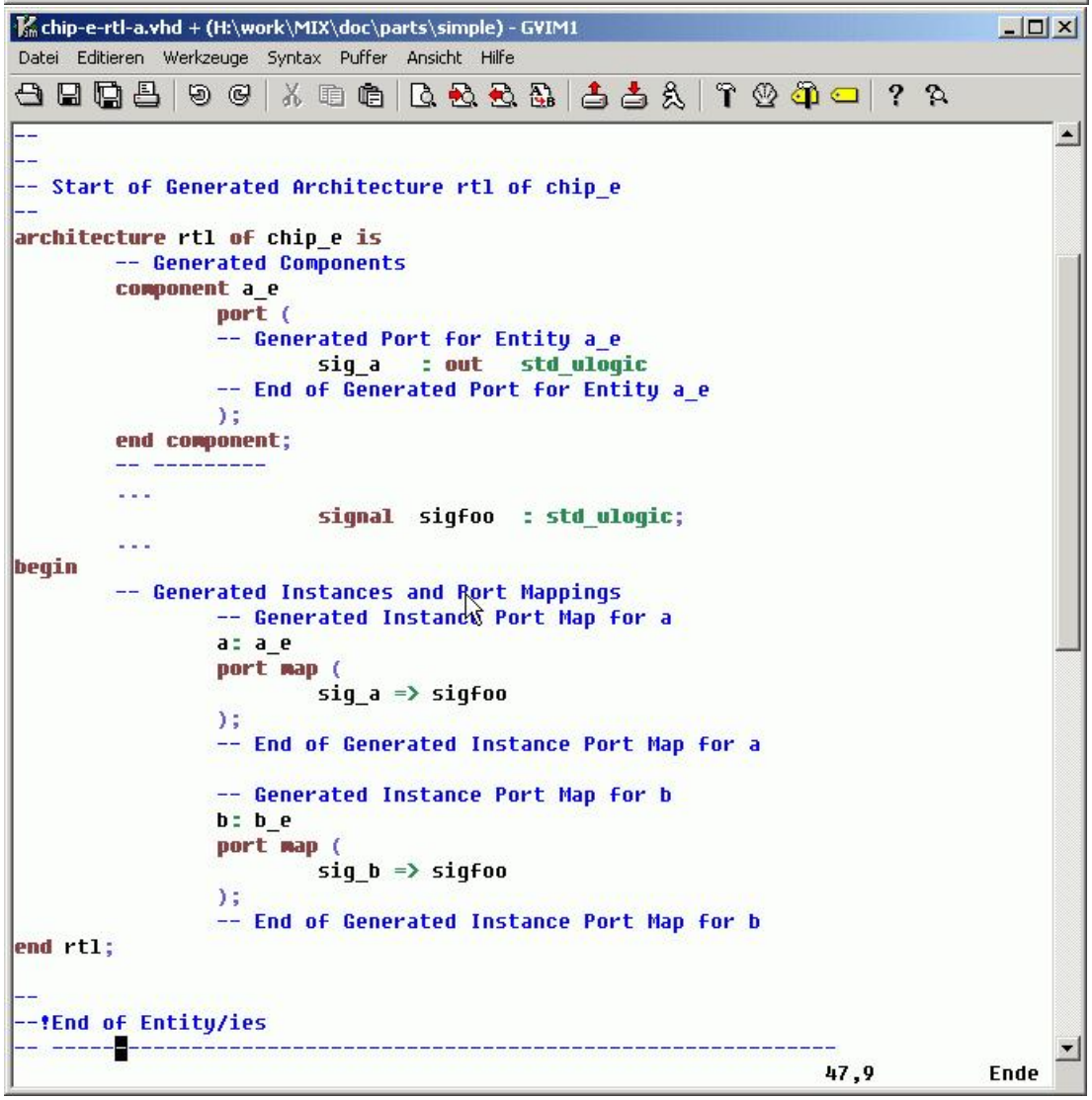
MIX generates various HDL files defined by the input data. If you select VHDL (also the default language) as output description for a hierarchy element, each element results in an appropriate entity, architecture and configuration description. By default MIX writes one file for all entities, one for all architecture and one for all configuration descriptions. Those file names are derived from the last excel input file name by stripping of the .xls extension and attaching a -e.vhd, -a.vhd and -c.vhd.
Here the working directory of the simple example contains a file mix.cfg, which is the convenient storage for MIX run-time configuration options. The lines "MIXCFG outarch ARCH", "MIXCFG outenty ENTY", and "MIXCFG outconf CONF" switch MIX outputs to separate files for each entity, architecture and configuration. In this case the file names are defined by the element name:

Type	**Basename**	**Extension**	**Example**
Entity	::entity-column	-e.vhd	chip_e-e.vhd
Architecture	::entity-column	-rtl-a.vhd	chip_e-rtl-a.vhd
Configuration	::entity-column	-c.vhd	chip_e-rtl-conf-c.vhd

Table 1: "_" in the filenames extension are converted to "-"

By default MIX does not write output data for leaf blocks (instances which are not parent for other instances). Adding a line like "MIXCFG generate.output.arch leaf" into the mix.cfg file changes that.
The generated output files contain head, body and footer sections. See the screenshots of the file a-e-e.vhd and chip-e-rtl-a.vhd for examples of an entity and an architecture definition.


a-e-e.txt (H:\work\MIX\doc) - GVIM1
Datei Editieren Werkzeuge Syntax Puffer Ansicht Hilfe
-- --
--
-- Entity Declaration for a_e
--
...
-- Generator: mix_0.pl Version: Revision: 1.12 , wilfried.gaensheimer@micronas.com
-- (C) 2003 Micronas GmbH
--
-- --
library IEEE;
use IEEE.std_logic_1164.all;
--
-- Start of Generated Entity a_e
--
entity a_e is
        -- Generated Port Declaration:
                port(
                -- Generated Port for Entity a_e
                        sig_a   : out   std_ulogic
                -- End of Generated Port for Entity a_e
                );
end a_e;
--
-- End of Generated Entity a_e
--
--
--!End of Entity/ies
-- --
15,13 Alles


chip-e-rtl-a.vhd + (H:\work\MIX\doc\parts\simple) - GVIM1
Datei Editieren Werkzeuge Syntax Puffer Ansicht Hilfe
--
--
-- Start of Generated Architecture rtl of chip_e
--
architecture rtl of chip_e is
        -- Generated Components
        component a_e
                port (
                -- Generated Port for Entity a_e
                        sig_a   : out   std_ulogic
                -- End of Generated Port for Entity a_e
                );
        end component;
        -- ---------
        ...
                        signal sigfoo : std_ulogic;
        ...
begin
        -- Generated Instances and Port Mappings
                -- Generated Instance Port Map for a
                a: a_e
                port map (
                        sig_a => sigfoo
                );
                -- End of Generated Instance Port Map for a

                -- Generated Instance Port Map for b
                b: b_e
                port map (
                        sig_b => sigfoo
                );
                -- End of Generated Instance Port Map for b
end rtl;

--
--!End of Entity/ies
-- --
47,9 Ende

6 Details

6.1 Initialization with -init

You can use the -init command line option to create the needed files:

```
$ mix_0.pl -init foo.vhd bar.xls
```

will create the file `bar.xls`, which has the following three worksheet categories:

- empty HIER, CONN and IO sheets
- template sheets with numerous examples TMPL_(HIER|CONN|IO)
- import sheets IMP_HIER and IMP_CONN (only of more *.vhd or *.v files are given as command line arguments).

You will also get a `mix.cfg` file. If `bar.xls` or `mix.cfg` already exists, the command will exit without changes. The import of `*.vhd` and `*.v` files is experimental and meant to give a way of getting a start description of your design. In case of VHDL files, only entity descriptions are imported. Take care of getting signal names and instance names properly.

6.2 Common excel worksheet properties

All excel worksheets parsed by MIX share some common properties. There needs to be a header line consisting only of keywords with leading double colon. All data before the header line is ignored. Only the first header line will be evaluated. Data in columns with no header or malformed headers will be ignored.
Commonly understood table headings are `::ign` and `::comment`. The `::ign` column is special, because it needs to be the first column of a sheet. If a cell in the `::ign` column starts with a `#` or a `//`, the complete row is ignored.The `::comment` column can contain user or program generated comments for a given row. It's data will be appended to it's contents as it appears. MIX reads the excel cell values. This is what you see, not the real contents of the cells. Thus all excel formulas can be used to define the cell values.
A lot of predefined text macros are understood and converted by MIX. A text macro is made up by a name surrounded by `%` signs. Retrieve a complete list of macros with the -listconf command line switch and grep all lines starting with `"MIXCFG macro"`. The table `%macro%` gives a list of predefined macros, their default settings and wether this macro can be used by the user. You can use text macros inside of any cell.

::ign	::gen	::variants	::parent	::inst	::lang	::entity	::config	::comment
# Mix SimpleExample, Macro Expansion, V0.1, 20030630, hierachy								
		Default	TESTBENCH	chip	VHDL	%::inst%_e	%::inst%_e_rtl_conf	
		Default	chip	a	VHDL	%::inst%_e	%::inst%_e_rtl_conf	
		Default	chip	b	%LANGUAGE%	%::inst%_e	%::inst%_e_rtl_conf	

A second type of macros are available to reference other columns of a like through `%::NAME%`. This will be replaced by the contents of the `::NAME` column of this row. `::NAME` has to be defined for the current worksheet, obviously. In the above example `%::inst%_e` will evaluate to `chip_e`, `a_e` and `b_e` accordingly. `%LANGUAGE%` becomes vhdl. Macro expansion happens just before the intermediate design data is written out, after evaluation of all input data. Recursive macro expansion is not implemented. Macros in primary keys like signal and instance names are evaluated at signal or instance creation time.

6.3 HIER sheet properties

The hierarchy of a design is defined in the HIER sheet. By default the HIER sheet is named HIER, but you can set the configuration value **hier.xls** to a Perl regular expression. MIX will then consider all worksheets which names match the perl regular expression, to be HIER definitions. In that case you are free to use different header definitions on different sheets.
The HIER sheets require at least the columns marked man. (mandatory) in the following table.

Column name	Descriptionend	Default value	Req.	Example
::ign	Ignore line	<empty>	man.	# comm.
::gen	Generator and match	<empty>	man.	see below
::variants	Variant selector	Default	opt.	Var1
::parent	Instance name of this instances parent cell	W_NO_PARENT	man.	chip
::inst	Instance name, primary key!	n/a	man.	a_i1
::lang	Language definition	VHDL	opt.	vhdl
::entity	Entity name	W_NO_ENTITY	man.	a
::config	Configuration name	W_NO_CONFIG	man.	a_rtl_conf
::shortname	Short name	<empty>	opt.	text
::use	Additional, project specific libraries	<empty>	opt.	padlib.foo
::comment	Comment field	<empty>	opt.	text

Internally the kewords `::debug`, `::hierarchy`, `::skip` and `::default` are used. Please do not use them. Apart from that you are free to add columns of your own. User defined columns are usable in `%::NAME%` macro expansion and are listed in the intermediate design data output. Only the `::inst` column has to contain a value in each row (which could be a `%::NAME%` macro, though). All other columns will receive more or less reasonable default values in case they are left empty.

6.4 HIER columns details

::gen If a cell here is not empty, the line will be considered as generator. See description of generator statements below.

::variants Select a line depending on the -variant command line switch. If -variant VAR is set, only lines whose `::variant` cell contains the keyword VAR, Default or empty, are selected and read in. Several variants may be given in one cell, separated by ",". Without specifying the `-variant` switch, the "Default" and empty `::variant` cells are read in and evaluated.

::inst Defines the instance name. If the same name appears in several rows, the resulting row will be overloaded from all input rows. The exact behaviour depends on the column name. Some are concatenated, some are replaced.

::lang HDL language selection, case insensitive. If this column is omitted or empty, VHDL output is generated. The default value can be changed by means of the macro.%LANGUAGE% macro. Currently only VHDL and Verilog are supported (XXX: Verilog output still t.b.d.).

::config Define the configuration name. It defaults to %DEFAULT_CONFIG%, which evaluates to %::entity%_%::arch%_conf. If the language of this entity is set to Verilog, no configuration will be printed nor will it be added to parent cell

configurations. The keyword %NO_CONFIG% will also suppress output of configuration for this entity.

Additional columns:

::arch If no ::arch column is given, architecture will default to "rtl". This is defined by the configuration hier.field.::arch.[3] = rtl; and cannot be changed globally.

::use Add project specific libraries and work packages to the HDL description files. See the ::use details below.

6.4.1 ::use Add Project Specific Libraries

The optional "`::use`" column allows to add libraries and work packages for a entity. Several libraries can be added, separated by comma or white space. In case of VHDL output, the use statement is added to the entity declaration, only. You can override that by adding a leading `ALL:`, `ARCH:` or `CONF:` keyword for this instance/entity. To change that globally, modify the configuration variable output.generate.use.

For each given library an appropriate text sequence is added:

```
foo.lib, bar.lib.something
```

will be printed as

```
library foo;
use foo.lib.all;
library bar;
use bar.lib.something.all;
```

If different instantiations of a entity have different ::use definitions, MIX adds up all these.

To change the global default value of

```
library IEEE;
use IEEE.std_logic_1164.all;
```

modify macro `%VHDL_USE_DEFAULT%`.

To add a library globally, add the appropriate text to the macro.`%VHDL_NOPROJ%` configuration, which will be used if no library is defined in the `::use` column.

A second usage is to suppress the component declaration output by the `%NDC%` or `%NO_COMPONENT_DECLARATION%` keyword. This is useful for instances, whose entities are taken from libraries.

Examples: see `t/sigport.xls` and `t/sigport/use/*.vhd` for more examples.
Configuration: `output.generate.use = enty`

6.5 Special HIER sheet properties

`%TOP%` is a pseudo instance and used for internal purposes, only.

6.6 CONN sheet details

The design connections are defined in the CONN sheet. The primary key of this sheet is the signalname as given in "`::name`". Signalnames are globally known for the design. All appearances of a name are connected, creating intermediate ports as needed.

6.7 typecast

The type of a signal can be typecasted by appending a function name with a `'` to the `::in` and `::out` definitions.

Example:

```
::name          signal_a
::type          type_a
::in            inst_a/port_a'typefunc
::out           inst_b/port_b'typefunc
```

The signal `signal_a` will be of type `type_a`, while the connecting ports of `inst_a` and `inst_b` are converted by applying the typecast function `typefunc`.
Caveat: typecast support is experimental.

6.8 CONN columns details

6.9 Special CONN sheet signals

Special signal names:
`%LOW%`, `%HIGH%`, `%LOW_BUS%`, `%HIGH_BUS%`

6.9.1 %OPEN% aka. open

Used the `%OPEN%` signal to leave some pins of module bar port a open MIX has no knowledge about ports. Everything is defined in terms of signals and instances. Use the "`open`" pseudo signal to define the extra pins.

E.g.: wire `foo/a` port to `bar/a` port. `bar/a` has extra pins.

```
signal_a ::high 7, ::low 0, ::out foo/a ::in bar/asn
%OPEN%, ::high 1, ::low 0, ::out -, ::out bar/a(9:8)
```

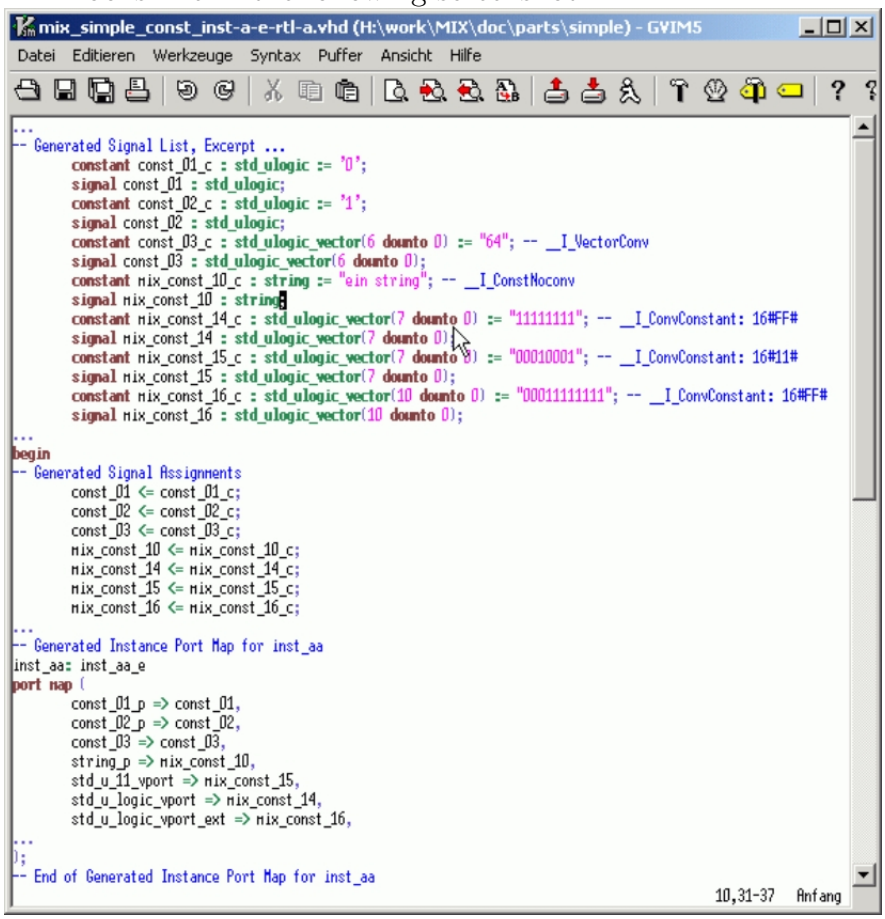
XXX write this in spreadsheet XXX
You could also force the `::in` pins to high or low, instead, e.g. use:
`%HIGH%`, `%HIGH_BUS%`, `%LOW` or `%LOW_BUS%`. Or a constant.

6.9.2 Constants

Constant values can be defined and used in several ways in the CONN worksheet. Constants can be marked with a C in the `::mode` column. Basically anything resembling a number written in the `::out` column will be considered to be a constant value. String and bit vector constants are enclosed in single or double quotes. Remember to type two single quotes in excel to start a single quote string. Excel takes the first quote character to prevent the following string to be interpolated by excel.
If you do not name a constant (leave the `::name` field empty), MIX will generate a name like `mix_const_N`. N starts by one and increments for each new constant value.
Constant values can be assigned to instance ports in the `::in` column of that constant.
Depending on the form of a constant and the output language, MIX tries to convert the constant value into something suitable. See the constant.xls example (XXXLINK).

::ign	::gen	::bundle	::class	::clock	::type	::high	::low	::mode	::name	::out	::in
		CONSTANT			std_ulogic			C	const_01	0	inst_aa/const_01_p
		CONSTANT			std_ulogic			C	const_02	1	inst_aa/const_02_p
		const_signal	c_sig		std_ulogic_vector	6	0		const_03	%CONST%/'64'	inst_aa/const_03
		const_signal	c_sig		std_ulogic_vector	3	0		const_04	%CONST%/'32'	inst_ab/const_04
		const_signal	c_sig		std_ulogic				const_05	1	inst_aa/const_05
		const_signal	c_sig		std_ulogic_vector	6	0		const_06	0xf	inst_ea/const_06_p
# Allowed formats: 0x[0-9a-f], 0[0-7], 10101, 'xxxx', 'zzzz', "abc"											
		Formats			std_ulogic					0	inst_aa/zero_p
		Formats			std_ulogic					1	inst_aa/one_p
		Formats			integer					10	inst_aa/integer_p
		Formats			real					10.2	inst_aa/real_p
		Formats			real					1_000_000.0	inst_aa/under_p
		Formats			integer					16#FF#	inst_aa/vhdl_basehex_p
		Formats			integer					2#1010_1010#	inst_aa/vhdlbase2_p
		Formats			real					2.2E-6	inst_aa/reale_p
		Formats			time					10 ns	inst_aa/int_time_p
		Formats			time					2.27us	inst_aa/real_time_p
		Formats			string					"ein string"	inst_aa/string_p
		Formats			bit_vector	7	0			"11111111"	inst_aa/bit_vector_p
		Formats			bit_vector	7	0			"1010"	inst_ea/bad_width_p
		Formats			std_ulogic_vector	7	0			'01010101'	inst_aa/std_ulogic_vector_p
# Various other constant formats											
		Std_ulogic			std_ulogic_vector	7	0			16#FF#	inst_aa/std_u_logic_vport
		Std_ulogic			std_ulogic_vector	7	0			16#11#	inst_aa/std_u_11_vport
		Std_ulogic			std_ulogic_vector	10	0			16#FF#	inst_aa/std_u_logic_vport_ext
		Std_ulogic			std_ulogic_vector	7	0			0xff	inst_aa/std_u_logic_port_02
		Std_ulogic			std_ulogic_vector	7	0			0b01010101	inst_aa/std_u_logic_bin_p
		Std_ulogic			std_ulogic_vector	7	0			8#07#	inst_aa/std_u_logic_octv_p
		Std_ulogic			std_ulogic_vector	7	0			2#11001100#	inst_aa/std_u_logic_binv_p
		Std_ulogic			std_ulogic_vector	7	0			4#3030#	inst_aa/std_u_logic_quadv_p
		Std_ulogic			std_ulogic_vector	3	0			16#ee#	inst_aa/std_u_logic_hexerr_p

If the target language is VHDL, XXXXX
MIX looks like in the following screenshot:

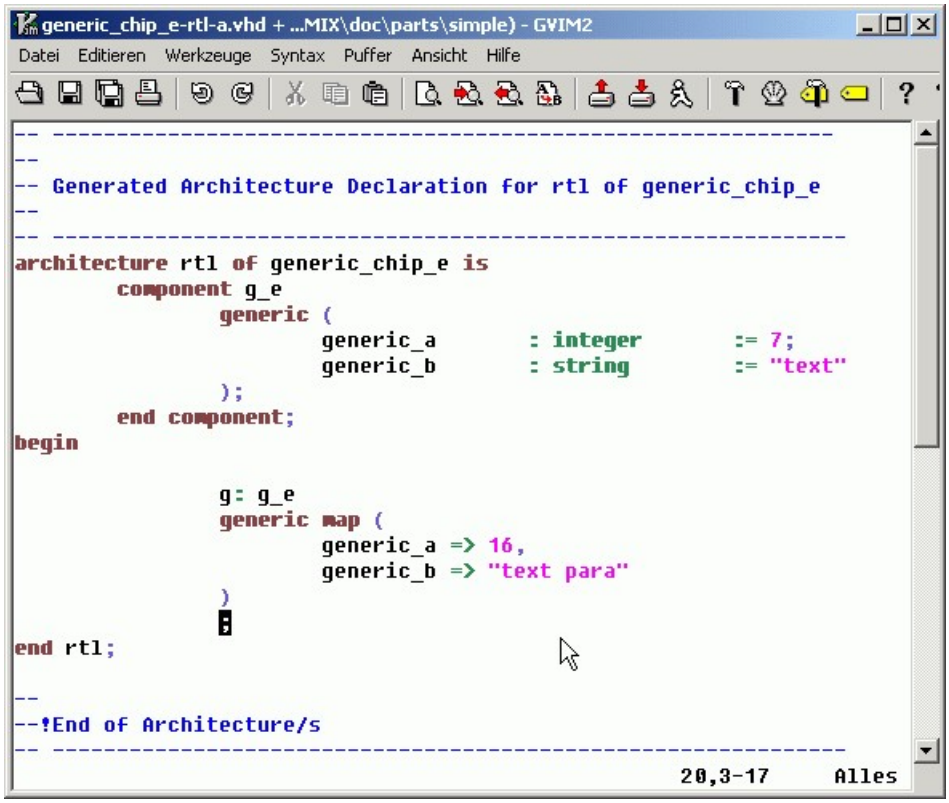


Search for lines with `__E` in the comments to detect constants MIX was not able to translate properly. Verilog constant output is not very mature as of today.

6.9.3 Generics and Parameters

Generics for entities are defined by a **G** in the `::mode` column. The `::name` column defines the generics name. If the `::out` column contains a string or a number, the value will be the default of this generic.
A **P** in the `::mode` column marks parameters. The value given in the `::out` column will be applied to the instances in the `::in` column.
If the `::name` column is empty, MIX will assign a generated name. The HDL generic name is either the "port" name given in the `::in` column or the `::name`.
A example description taken from a CONN sheet will create the following output in VHDL:

::type	::mode	::name	::out	::in
integer	G	generic_a	7	g/generic_a
integer	P	parameter_a	16	g/generic_a
string	G	generic_b	"text"	g/generic_b
string	P	parameter_b	"text"	g/generic_b



6.10 Simple text macros

Text marked with with a % on both sides is subject to be replaced, if a macro or a postfix of that name is defined. See the table of predefined macros. Additionally the contents of the input table can be referenced like %::column_head%. If no suitable text macro is found, the text will be left as is.

The text macro replacement does not operate recursively in general.

Please consider that some text macros are used internally and some have a special meaning, too. E.g. the pseudo signals %OPEN% and %LOW_BUS%.

6.11 Predefined and user macros

The following table contains a list of predefined macros. Some of them are defined at run time (e.g. the current date), some are for internal purposes only. macros marked with a yes in the "User" column can be set freely by the user on the command line or in the `mix.cfg` configuration file.

Macro Name	**Default Value**	**Description**	**User**
%0%	mix_0.pl	Program name	*run time*
%ARGV%	K:\Projects\MIX\PROG \mix_0.pl -listconf	Program arguments	*run time*
%BUFFER%	buffer		yes
%BUS_TYPE%	std_ulogic_vector		yes
%CONST%	__CONST__		yes
%DATE%	Mon Jun 30 16:22:41 2003-06-30	Current date	*run time*
%DEFAULT_MODE%	S	Signal default mode	yes
%EMPTY%		Empty string	yes
%GENERIC%	__GENERIC__		-
%H%	$	internal	-
%HIGH%	mix_logic1	Name of logic high value	yes
%HIGH_BUS%	mix_logic1_bus	Name of logic high value bus	yes
%HOME%	H:\		-
%IOCELL_SELECT_PORT%	select		yes
%IOCELL_TYPE%	__E_DEFAULT_IOCELL__		-
%IOCR%	\n		-
%LANGUAGE%	vhdl		yes
%LOW%	mix_logic0		yes
%LOW_BUS%	mix_logic0_bus		yes
%NULL%	0		-
%OPEN%	open		-
%PAD_CLASS%	PAD		yes
%PAD_TYPE%	__E_DEFAULT_PAD__		-
%PARAMETER%	__PARAMETER__		-
%PROJECT%	NO_PROJECT_SET		yes
%SIGNAL%	std_logic		yes
%SPACE%			yes
%TAB%	\t		-

%TBD%	__W_TO_BE_DEFINED		-
%TOP%	__TOP__		yes
%UNDEF%	ERROR_UNDEF		-
%UNDEF_1%	ERROR_UNDEF_1		-
%UNDEF_2%	ERROR_UNDEF_2		-
%UNDEF_3%	ERROR_UNDEF_3		-
%UNDEF_4%	ERROR_UNDEF_4		-
%USER%	wig		*run time*
%VERILOG_TIMESCALE%	'timescale 1ns / 1ns;		yes
%VERSION%	Revision 1.12		-
%VHDL_NOPROJ%	– No project specific VHDL libraries		-
%VHDL_USE%	– No project specific VHDL libraries		yes
%VHDL_USE_ARCH%	%VHDL_USE_DEFAULT% %VHDL_USE%		yes
%VHDL_USE_CONF%	%VHDL_USE_DEFAULT% %VHDL_USE%		yes
%VHDL_USE_DEFAULT%	library IEEE; use IEEE.std_logic_1164.all		yes
%VHDL_USE_ENTITY%	%VHDL_USE_DEFAULT% %VHDL_USE%		yes

The default values can be changed by using

```
-conf macro.%THIS_MACRO%=my_value
```

command line switch. New macros can be defined the same way.
Alternatively a line like

```
MIXCFG macro.%THIS_MACRO% my_value
```

to the mix.cfg configuration file will achieve the same result.

7 CONN sheet macros

To simplify the wiring of standard interfaces, MIX provides the connection macro facility. The connection macros are entered just like any other connection, but are marked by MH, MD and MX in the ::gen column. MH is the macro header line, which has to be followed by one to many MD macro definition row. The first non-comment line without a MD tag stops the macro definition.
The tag MX marks lines subject to be macro expanded. The macro expansion takes place after the initial tables were parsed.
You can define simple, one-letter variabes in the macro cells. Any text in ::ign, ::gen, ::comment and ::descr cells of a MH and MX row will not be subject to matching and evaluation, but be ignored. Apart from that the column names have no special meaning.

A simple example will illustrate the connection macro usage.

::ign	::gen	::cola	::colb	::colc	::cold	::cole	::colf	::colg
	MH	$n	$1	$h	alarm_time_$4_$5			
	MD	Uddrv_gen_$n	$1	$h	alarm_time_$4_$5		d_$4_$5/alarm_time(3:0)	Display storage buffer $n $4_$5
	MD	Uddrv_gen_$n	$1	$h	current_time_$4_$5		d_$4_$5/current_time(3:0)	Display storage buffer $n $4_$5
	MD	Uddrv_gen_$n	$1	$h	key_buffer_$n		d_$4_$5/key_buffer(3:0)	Display storage buffer $n $4_$5
	MD	Uddrv_gen_$n	$1	6	display_$4_$5	d_$4_$5/display(6:0)		Display storage buffer $n $4_$5
	MD	Uddrv_gen_$n	$1	$h	alarm	d_$4_$5/sound_alarm=($n)	u_and_f/y($n)=($n)	Display storage buffer $n $4_$5
	MX	0	Display	3	alarm_time_ls_min			
	MX	1	Display	3	alarm_time_ms_min			

Here the macro header MH defines the variables $n, $1, $h, $4 and $5. The MIX parser extracts the MH row and accompanying MD rows (five lines here) from the table. In a second run each MX row will trigger a match operation against all MH definitions. MX and MH are a match, if each cell defined in the MH row has a matching counterpart in the MX row. Variables in the MH cells are considered to match any string (.+ in perl regular expression syntax). Thus the first MX line matches the MH lines here. The variables defined in the MH cells are assigned the matching values from the MX line. E.g. $n will be 0 for the first MX, 1 for the second. The variable $4 is assignd ls, $5 is min.

If MX and MH match, the accompanying MD lines are executed. Variables are replaced by the value they are assigned too, currently. Here the first MX will result in the following table to be generated:

::cole	::colf	::colg
	d_ls_min/alarm_time(3:0)	Display storage buffer 0 ls_min
	d_ls_min/current_time(3:0)	Display storage buffer 0 ls_min
	d_ls_min/key_buffer(3:0)	Display storage buffer 0 ls_min
d_ls_min/display(6:0)		Display storage buffer 0 ls_min
d_ls_min/sound_alarm=(0)	u_and_f/y(0)=(0)	Display storage buffer 0 ls_min

The ::gen column is set to G_MX after evaluation. Empty cells in MD lines are filled with the value given by MX, if any.

Please see the contrib directory for commonly available connection macros (t.b.d.).

8 Generator statements

8.1 Hierachy Generator Operators

Another powerfull feature of MIX are the generator statements. The generators are applied after initial tool setup and connection macro evaluation, first the hierachy generators, then the connection generators.

Generators are defined in the ::gen column. Basically three types are available:

- Constructor generators: $i(N..M)
- Match generator: /match_expression/
- Bounded match generator: $i(N..M),/match_expression/

match_expression is like perl(1) regular expressions, but with some extensions and specials.

8.1.1 Constructor Generator statement

MIX takes the variable and the range definied in ::gen and evaluates the rest of the row for each value in the range. Currently only one variable is allowed and the range has to be of form (N..M).

A simple example illustrates the usage:

::ign	::gen	::variants	::parent	::inst		::lang	::entity	::config
	$i(1..10)		inst_a	inst_$i	VHDL		inst_$i_e	inst_$i_e_rtl_conf

will give ten instances from inst_1 to inst_10, each being a submodule of inst_a with each having a entity inst_1_e to inst_10_e, dito. for the configuration. Simple arithmetic can be applied to derive values from the run time parameter $i.
Constructor's will yield new instances. An instance name has to be given in the ::inst column.

8.1.2 Match generator

For each instance defined up to now, match_expression is evaluated. If this yields true, the line is executed. Parts of the expression in parantheses are used to set the variables $1, $2, ... accordingly. See the perl regular expression man page perlre(1) for more details. This variables can be used in the other cells. Simple arithmetic is possible, e.g. $i + 1 or $1 * 2.
A match generator will yield new instances only if the name in the ::inst column is set to a value different from the matching instance.

Example:

XXX t.b.d.

8.1.3 Bounded match generator

By adding a run variable and a range $i(N..M), match generators can be restricted to only apply if the variable $i is within the range. $i has to be defined in match_expression and will match any number.

In opposite to the constructor generator, MIX will not evaluate all possible values for $i, but only make sure $i stays within the bounds of the range.

Example:

XXX t.b.d.

8.1.4 Advanced features of the match generators

The match expression can contain references to all fields of the to match object with the ::NAME_OF_COL=string(match):: reference.

E.g. the connection generator

::ign	::gen	::type	::iname	::out	::in
	$i(270..302),/iom_(.*)::pin=$i/	std_ulogic	i_$i_{$i+1}	iom_$1/serial_o	
	$i(271..303),/iom_(.*)::pin=$i/	std_ulogic	i_{$i-1}_$i		iom_$1/serial_i

will wire all instances iom_.* having a property ::pin in the given range with a newly defined signal i_N_N+1. iom_foo with ::pin = 270 port serial_o will drive the signal i_270_271, the instance iom_bar with ::pin = 271 is connected to the signal i_270_271 to it's port serial_i

by the next line. for instance iom_foo only the first line matches, thus that module will net get a connection to serial_i by these generator lines.

A trainling :: can be omitted. If two propertied are to be matched, this will look like
/::prop1=(.*)::::prop2=string(match?)/

8.1.5 Additional Information

Match generators work the same way on the CONN sheets. By default the match expression will match against all defined instances, not connections.
To make a match expression iterate over all connection, prefix the whole expression by the CONN: keyword.

Please have a look into the distributed howto.xls and the macro.xls in the text case directory t to see more examples.

9 IO sheet

A third category of input specification is the IO sheet. The contents of the IO sheet is parsed and translated into instances of io cell blocks and pad cells and connections of the io logic with the design core logic.

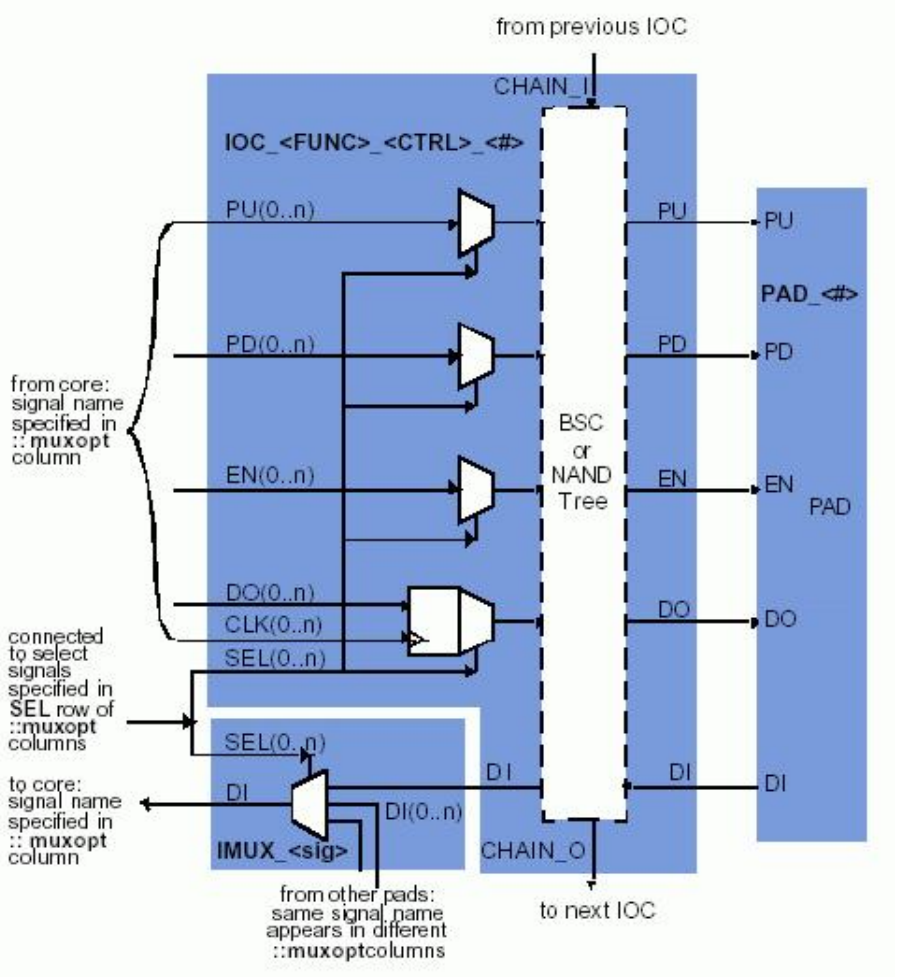


Image 3: IO Cell and Pad Layout
The IO sheet is a simple way to specify the connections of the IO cell to the design core logic. MIX will derive the connections of the DI(0..n), DO(0..n), PU(0..n), PD(0..n), EN(0..n) and

the accompanying select lines. MIX will not add special purpose connections and the IO cell to Pad cell connections. This should be done by using the macro and generator statements in the CONN sheets.
A simple example illustrates the various input fields and their usage.

::ign	::class	::ispin	::pin	::pad	::type	::iocell	::port	::name	::muxopt	::muxopt	::muxopt	::muxopt
	%SEL%						sel	pad	iosel_0	iosel_1	iosel_2	iosel_3
	DATA_I	1	1	1	w_pad_i	ioc_g_i	di	data_i1	data_i1.0	data_i1.1	data_i1.2	data_i1.3
	DATA_O		1	2	w_pad_o	ioc_g_o	do	data_o1	data_o1.0	data_o1.1	data_o1.2	data_o1.3
	%SEL%						sel	pad	iosel_disp	iosel_ls_min		
	DISPLAY	1	12	12	w_disp	ioc_r_io	di, do, en	disp_2	di2.0, disp2.0, disp2_en.0	, dis_ls_min.0, dis_ls_en		
	DISPLAY	1	13	13	w_disp	ioc_r_io	di, do, en	disp_3	di2.1, disp2.1, disp2_en.1	, dis_ls_min.1, dis_ls_en		
	DISPLAY	1	14	14	w_disp	ioc_r_io	di, do, en	disp_4	di2.3, disp2.3, disp2_en.3	, dis_ls_min.2, dis_ls_en		
	DISPLAY	1	15	15	w_disp	ioc_r_io	di, do, en	disp_5	di2.4, disp2.4, disp2_en.4	, dis_ls_min.3, dis_ls_en		

9.1 IO sheet column header

The IO sheet starts with the usual header line, defining mandatory and optional columns:

- `::ign` A # marks a comment line. Has to be first column. Optional.
- `::class` The class name is forwarded to the `::class` field of the CONN sheet. (opt.)
- `::ispin` Set to one it this pad is actually bonded (ignored by MIX).
- `::pin` Number or name of the pin. (ignored by MIX)
- `::pad` The primary key of the IO sheet. Has to an integer value. The pad numbers do not have to be consecutive. Mandatory column.
- `::type` Defines the entity of the generated pad cell.
- `::iocell` Defines name and entitiy of the generated io cell
- `::port` Define the io cell port name towards to the core logic. Port names are separated by , and/or <Alt><CR>.
- `::name` pad name
- `::muxopt` Connection matrix of io cell ports to core logic. Signals are separated by , and/or <Alt>-<CR>. Signals may be single bits or a one bit bus slice. Type `core_sig.N` or `core_sig(N)` for bit N of the core signale `core_sig.::muxopt` is allowed to appear several times. The number of signals and the order matches the port names in the ::port field.

9.2 IO sheet %SEL% rows

The rows with the key `%SEL%` in the `::class` field define the wiring of the IO cells multiplexer select lines. The name in the `::port` field defined the io cell port to connect to. The `::name` field is ignored. The signal names listed in the `::muxopt` columns are connected from the designs core with the appropriate slice of the io cell multiplexer select lines (one hot select lines). The leftmost `::muxopt` connects to bit 0, the next to bit 1 and so forth. `core_sel.N` or

`core_cel(N)`. can be used to wire select buses. The number of non-empty consecutive `::muxopt` fields also define the width of the multiplexer. The actual multiplexer width is stored in the `%::_muxwidth_%` field.
The given values for the select lines are valid until the last row or another `%SEL%` line is found. `%NOSEL%` will stop usage of in/out multiplexer and select lines.
Setting the configuration switch "`iocell.select`" to "`bus`" changes from a one-hot architecture to a select bus architecture. The select bus name is take in the leftmost `::muxopt` column, an appropriate width is calculated by the number of defined `::muxopt` columns.

9.3 IO sheet pad rows

All other rows make up the connection matrix. For each row an io cell is instantiated. By default this io cell's name is composed by the type listed in the `::type` field with the `::pad` number attached, separated by a `_`.
Secondly a pad cell is instantiated. By default this pad cell's name is composed from the prefix `pad_` and the pad number as given in the `::pad` field. The default naming for both io cell and pad is given by the configuration variables

```
pad.name = %PREFIX_PAD_GEN%%::pad%
iocell.name = %::iocell%_%::pad%
```

You can leave unused `::muxopt` columns empty. Then MIX will reduce the number of select lines accordingly.

9.4 IO cell and Pad connections

MIX IO sheet parser connects the IO cell internal interface towards the design core logic, only. Connections between pad and io cell need to be defined explicitely, usually be means of `::gen` match operators.
Additonal wires for NAND tree or boundary scan are specified the same way.Signal busses need to be defined properly, esp. the width and type. MIX will derive these properties from the definition.
The generated pad and iocells need to be linked into the design hierarchy properly, e.g. by adding `::gen` match operators in the `HIER` sheet.

	HIER							
::ign	::gen	::variants	::parent	::inst	::lang	::entity	::config	::comment
#Pads, name is pad_NN, NN is the number from :pad column in IO								
		Default	%TOP%	padframe	vhdl	padframe_e	%::entity%_conf	
	/pad_(\d+)/	Default	padframe	pad_$1	vhdl		%::entity%_conf	
	/ioc_[.*]/	Default	padframe	ioc_$1	vhdl		%::entity%_conf	

		CONN										
::ign	::gen	::bundle	::class	::clock	::type	::high	::low	::mode	::name	::out	::in	::descr
# Standard Pad Cells wiring: do, en, di; triggered by iocell name which is of the form ioc_[rg]_[io...]_N will be derived by the IO ..												
	/ioc_(\w_\w*o\w*)_(\d+)/	PAD_CTRL	IO	multiple	std_ulogic				pad_do_$2	ioc_$1_$2/p_do	pad_$2/do	data out
	/ioc_(\w_\w*o\w*)_(\d+)/	PAD_CTRL	IO	multiple	std_ulogic				pad_en_$2	ioc_$1_$2/p_en	pad_$2/en	pad enable
	/ioc_(\w_\w*i\w*)_(\d+)/	PAD_CTRL	IO	multiple	std_ulogic				pad_di_$2	pad_$2/di	ioc_$1_$2/p_di	data in

9.5 IO Parser global configuration

Some global configuration are available to tailor the IO sheet parser behaviour:

`iocell.name`	Rule determines naming of the io cell instances. Default `"%::iocell_%::pad%"`
`iocell.auto`	If set to `"bus"`, create busses as needed. Otherwise MIX relies on appropriate definition of busses by the CONN sheets.
`iocell.bus`	If `iocell.auto` is set to bus, MIX will attach the keyword listed here (`_vector`) to signal type definitions.
`iocell.defaultdir`	Defines default iocell port direction. Default: `in`
`iocell.in`	comma and/or white-space separated list of in ports Default: `do, en, pu, pd, xout`
`iocell.out`	list of io-cells out ports Default: `di, xin`
`iocell.select`	List of keywords defining the select multiplexer connections: `onehot` -> use one hot architecture for select lines, `bus` -> use select line bus, `auto` -> calculate width of select bus or one-hot based on `%SEL%` width; Default: `onehot,auto`
`pad.name`	Rule determines the naming of the pad instances. Default: `"%PREFIX_PAD_GEN%%::pad%"`

10 I2C sheet

t.b.d.

11 VI2C sheet

12 Alarm clock example

13 Alarm clock example

14 MIX converter man page

14.1 Synopsis

14.2 Command line switches

`-out OUTPUTFILE.ext`
: defines output filename and type

`-outenty OUT-e.vhd|ENTY|COMB`
: Write all entities into OUT-e.vhd.
If argument is ENTY, each entity will be written
into a file called entityname-e.vhd.. (The exact
naming depends on changeable rules).
If argument is COMB, entity, architecture and configuration
will all be written into one file called entityname.vhd

`-outarch OUT-rtl-a.vhd|ARCH|COMB`
See description of outenty option.
`-outconf OUT-c.vhd|CONF|COMB`
See description of outenty option.
`-combine`
write entity, architecture and configuration into one file for each entity. Shortcut for setting -out[enty|arch|conf] to COMB individually.
`-dir DIRECTORY`
write intermediate, internal and backend data into the given DIRECTORY. By default MIX writes to the current working directory
`-top TOPCELL`
use TOPCELL as top. Default is TESTBENCH or daughter of TESTBENCH.
`-adump`
dump internal data in ASCII format, too (debugging, use with small data set).
`-variant VAR1` Select VAR1 from the HIER worksheet.
`-conf key.key.key=value`
Overload $EHkeykeykey with value or add a new configuration variable.
`-listconf`
Print out all available/predefined configurations options
`-delta`
Output will be compared against previous runs.
`-sheet SHEET=MATCH`
SHEET can be one of "hier", "conn", "vi2c". MATCH is a perl regular expression.
`-strip`
Remove old and diff sheets. These sheets are named with "O_" or "DIFF".

Add your options here ....

"Standard" options:

```
my @stdopts = qw(help|h!  verbose|v!  quiet|q!  nobanner!  debug:i
  makeopts=s@ gmakeopts=s@);
```

Caveat: the -h option will not work on MS-Windows

14.3 Runtime options and configuration

Runtime configuration is controlled by (increasing precedence):

- built-in default values
- `mix.cfg` files, if found `$HOME`, `$PROJECT` and/or in current directory. Format is: `MIXCFG name.of.conf` *value*
- CONF sheet found in input xls files
- command line switch: `-conf foo.bar=`*value*
- dedicated command line options

MIX reads in `mix.cfg` configuration files in the following locations:

1. `$ENV{HOME}`
   `"HOMEDRIVE"`, `"HOMEPATH"`, `"USERPROFILE"` or `"C:\"`
   (only from the first matching location)

2. `$ENV{PROJECT}`

3. `"."` (cwd)

Order: `HOME` / `HOME` / `cwd()` / `-conf` (last wins)

14.4 Misc features

14.4.1 -delta mode

Do not change output files, but report number of changes. Adds extra sheets `DIFF_CONN` and `DIFF_HIER` (and old versions of them) to the intermediate output. The `FOO.pld` internal output gets overwritten, though. All messages are appended to `mix_0.pl`
If a new HDL-file needs to be created by the changes, the -delta mode is not be applied for that file, but it is generated as new. If you never have written a `FOO-mixed.xls` intermediate output, there is no `HIER` or `CONN` sheet generated.
By adding the following two lines to your configuration (e.g. `./mix.cfg`), delta mode will be the default:

```
MIXCFG output.generate.delta 1
MIXCFG output.delta sort,remove
```

See the configuration option description for more details.

`-nodelta` switches delta mode off, then.

14.4.2 Imtermediate Excel Sheet

Format of intermediate xls sheet will be kept as is as long as the number and order of columns is unchanged.
MIX saves three old versions of the generated `HIER` and `CONN` sheets. The worksheets names are rotated by a trailing `_` and number.

14.4.3 Output Redirection

MIX writes all output into the current working directory. By using the

```
-dir DIRECTORY
```

option, you can set the output directory for all intermediate, internal and HDL files to `DIRECTORY`. Absolute path names defined by other options (e.g. -out) are not changed, though. If `DIRECTORY` does not exist it will be created. To have separate output directories, use the `mix.cfg` file:

```
MIXCFG output.path HDLDIRS
MIXCFG internal.path MIXINTERNAL
MIXCFG intermediate MIXINTERMEDIATE
```

writes internal, intermediate and HDL files to the given pathes. All directories have to exist and will not be created.

15 Alarm clock example

15.1 Core logic

15.2 IO logic

16 Other examples

17 Known Bugs and limitations

MS-Win/UNIX end-of-line issue: Some EDA tools are not able to cope with the different end-of-line (CR vs LF/CR) of UNIX and MS-Windows. Use:

```
$ module load freeware; recode "pc..lat1" *.vhd
```

or (in MIX's Base directory):

```
$ dos2unix.pl <filename>
```

for Dos to Unix convertion and:

```
$ unix2dos.pl <filename>
```

for Unix to Dos convertion.

18 Issue tracking: CADNET -> Issue tracking -> CAD Software -> MIX

If you find unexpected or buggy behavior, please issue a trouble ticket. Don't forget to add a short description. The test case should contain all source files and also log files, the exact command line switches and configuration files applied.
Please provide a comprehensive description of issues found including all required input data and command line switch to allow fast debugging and fixing.

19 Links

19.1 MIX paper and documentation

EDP_2003_final_030331.pdf
MIX_Specification.xls (see \\Galaxy\Development\PROJECTS\MIX)
MIX_Intro.ppt (see \\Galaxy\Development\PROJECTS\MIX

19.2 Micronas internal

Micronas HDL coding guidelines.pdf

19.3 Others

19.4 Resources

Downloads for:
- IO Examples
- Standard bus examples
- NAND Tree

- BS example
- miscellaneous useful macros

19.5 Used Software

- Perl
- Komodo
- Vim
- WinCVS

19.6 Release Notes

19.6.1 20030716

see doc\release_20030716.txt

19.6.2 20030709

see doc\release_20030709.txt

19.6.3 20030605

See doc\release_20030605.txt