MIX User Manual Micronas Interconnect Specification Expander, tool version 2.0

MainMenu

Lu, 24 June 2009 (created 4 March 2009)

no tags

This TiddlyWiki as PDF

Introduction

LU, 11 March 2009 (created 4 March 2009)

no tags

MIX is a methodology and tool to generate hardware descriptions, low-level driver software, verification code and documentation out of a single source specification. In contrast to most *ESL* (electronic system level design) tools, this approach focuses on **structural specification**, like hierarchy, interconnect, **I/Os** and **configuration registers** instead of synthesizing a functional specification (though the user can always insert arbitrary HDL code). Due to automatic code generation, consistency between specification and design is guaranteed and error-prone manual transformations can be avoided.

A table-based specification format was chosen to enable efficient data entry and review. Lately, the Spirit consortiums *IP-XACT* input/output format for register specifications has been added. It is also possible to access the code generation functions through a Perl API which is useful where a table-based entry format contains too much redundancy due to its flat data representation.

Installation

LU, 4 March 2009 (created 4 March 2009)

The MIX toolset uses the Perl scripting language and runs on both Linux and Windows platforms. Currently Perl 5.8.8 is required.

no tags

For Windows: Please install a recent version of Perl on your MS Windows workstation, e.g. the freely available ActiveState Perl.
Map the mix directory (\\samba\tools\mix) to a network drive. 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 immediately start MIX:
- Open a command shell on your desktop workstation:
- Press Start -> Run ; type `cmd`
- Now change to your working directory and start the generator like
- `K:\1.9\mix foo.xls`

For UNIX (Solaris/Linux) no user installation is required. Everything needed is installed in /tools/mix/<version>.
Use `module load mix/<version>` to prepare your environment. Check the output of `module avail mix` to see what versions are available. In case of problems with Perl packages, use `module unload msd` prior to loading the mix module.

GettingStarted

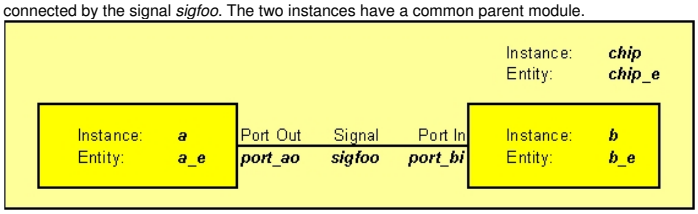
LU, 9 March 2009 (created 4 March 2009)

To receive useful results, MIX reads in various input data from spreadsheets, typically stored in Excel workbooks. At least you will need to prepare a description of your design hierarchy (*HIER* sheet) and the connectivity (*CONN* sheet), or a Register Specification. If found, MIX will convert a *IO Specification* input/output sheet, listing input/output pads and iocells and how these are linked to the core logic into appropriate hierarchy and connection lists. The *Register Specification* sheet can be used to describe device configuration registers.

no tags

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 has just two components *a* and *b*, which are connected by the signal *sigfoo*. The two instances have a common parent module.



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 (unique instance identifier). 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 MIX.

The first column of all sheets has to be `::ign`. 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 defnition line. The primary field is the `::name` column (unique signal name identifier). The `::in` and `::out` columns are used to define the drivers and loads for the signals.

Mix it!

Run the MIX tool in the directory the excel spreadsheet is stored in (workbook contains both HIER and CONN sheet in this example):

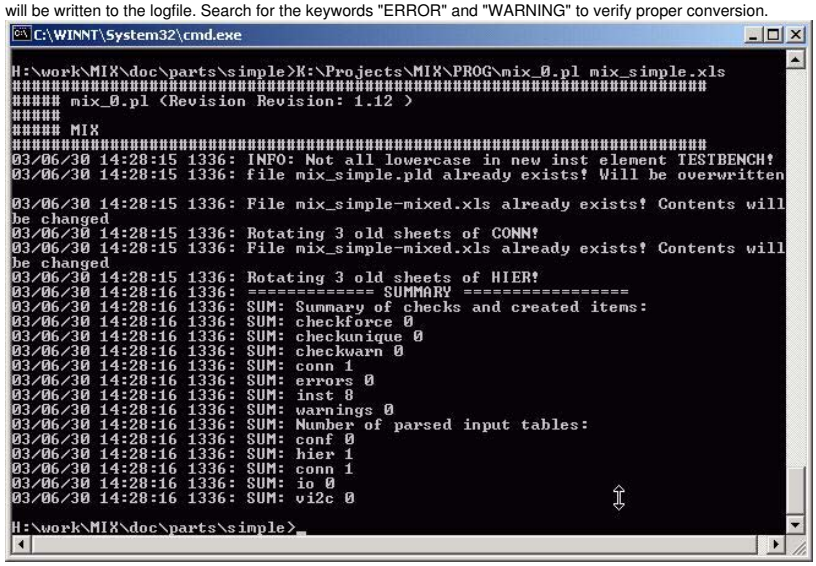
MS-Windows:

```
$ K:\1.9\mix mix_simple.xls
```

UNIX:

```
$ module load perl mix/1.9
$ mix 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*). A logfile (*mix.log*) and the HDL output files are written in the same run. The next image shows a screenshot of the .xls conversion. Only the most important errors and warnings are written to the screen, while a lot of information will be written to the logfile. Search for the keywords "ERROR" and "WARNING" to verify proper 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 MIX run. The intermediate excel description file keeps a history of previous HIER and CONN sheets by rotating _N extended worksheet names.

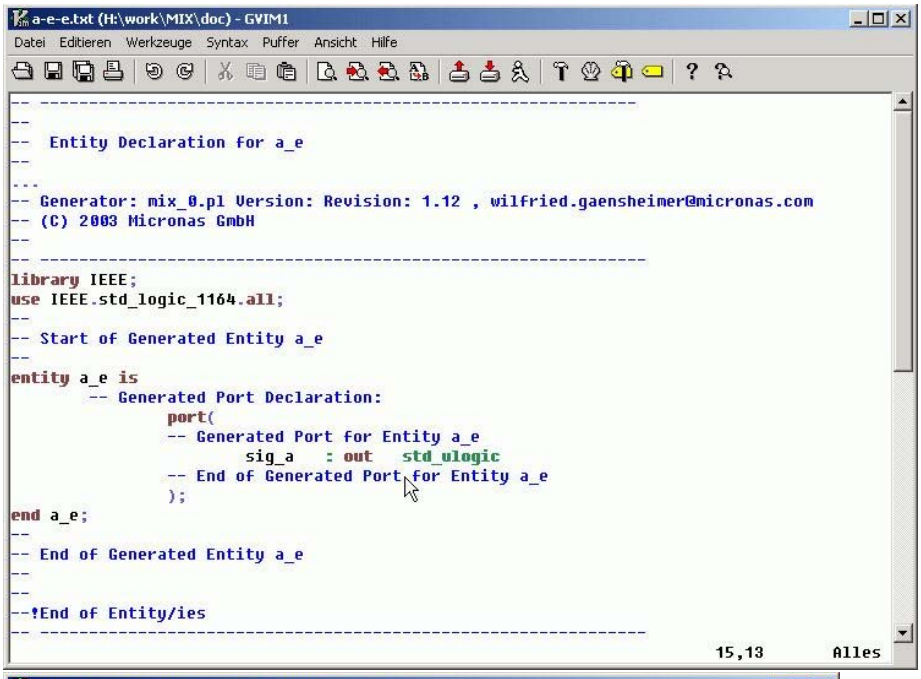
You Get What You Typed

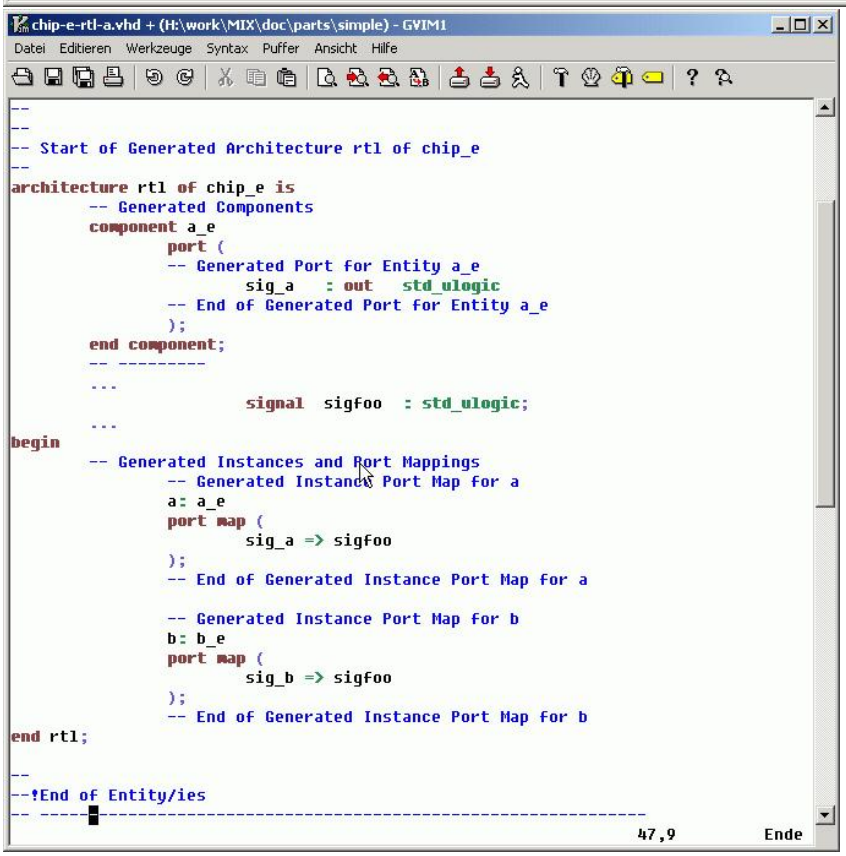
MIX generates various HDL files defined by the input data. If you select VHDL (also the default language) as output description for a hierarchy element, each element results in an appropriate entity, architecture and configuration description. By default MIX writes one file for all entities, one for 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.

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.





StructuralSpecification

LU, 11 March 2009 (created 9 March 2009)

This section explains how hierarchical and connectivity information is captured in the table-based format.

no tags

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 if **.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.

Common Worksheet Properties

All 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 cell values. This is, for Excel and Open-Office, what you see, not the real contents of the cells. Thus all formulas can be used to define the cell values (note: csv is not able to do that).
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 macro `%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.

Special Spreadsheet-format Properties

MIX is able to read three different table formats: csv (Comma-Seperated-Values), sxc (Star-Office-Spreadsheet) and xls (Excel-Spreadsheet). While Excel and StarOffice spreadsheets are fully featured, the csv Format does not understand formulas.This is because the real content and it's values are same in csv. The only colored output can (at this moment) be achieved by writing Excel spreadsheets (for this an already installed Excel is needed). While writing Excel-sheets implies Windows, reading input can be done on every platform (if it got a Perl port). Trying to read a Excel-Sheet on non-Windows grounds will cause MIX to produce csv per default. The output format can be forced by setting format.out to csv, sxc or xls. In this case MIX will read the given input file and produce the forced output format.

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	Description	Default	Required	Example
::ign	Ignore line, usually the first column.	<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 for VHDL modules. includes and defines for Verilog.	<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.

HIER Columns Details

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

::udc	See details below

::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.

For verilog modules (as defined in the ::lang column) the ::use column contents will be split by newline or comma and added to the verilog module top. A typical usage is for `defines and `include. The text is taken literally. Use the text macros %DEFINE% and %INCLUDE% if you want to avoid the backtick.

Examples: see *t/sigport.xls* and *t/sigport/use/*.vhd* for more examples.

Configuration parameter: output.generate.use = enty

::udc - Add User-defined Code

Since the release from July 2005, the MIX template provides several new hooks to add code into the header, body or footer of the created HDL files. The globally available macros %VHDL_HOOK_MODE_LOCATION% and %VERILOG_HOOK_BODY% can be set to an arbitrary string. They will be overloaded by the instance specific ::udc text, if that is defined in the HIER sheet. The ::udc cell contents will be split into several parts by markers like /%LOCATION%/, by default the %BODY% tag will be used. LOCATION can be one of HEAD, BODY or FOOT, in case of VDHL architecure also DECL. Currently the ::udc text will be evaluated for the MODE arch, only. Hooks are available for MODE arch, enty and conf, though.

If the ::udc contains the tags %PINS% and/or %AINS%, the text following is pre- or appended to this components instantiation.

Special HIER Sheet Properties

Pseudo Instances %TOP% et al.

%TOP% is a pseudo instance and used for internal purposes, only. Please use testbench as the top level of your description. Another internally used instance is %TYPECAST% .

Simple Logic Creation

MIX will support simple logic creation when you use one of the following keywords as the entity of a hierachical element:

```
%AND%, %OR%, %NAND%, %NOR%, %XOR%, %NOT%, %WIRE%
```

%WIRE% will be handled as connection (assignment), while the logic will be created with HDL logic description. Currently no bit-splicing is supported and appropriate bit width of busses will not be checked. Make sure only one output signal gets connected. See the testcase *logic* for examples.

Note that you can also use the ::udc column to add logic to the generated architectures.

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.

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.

Automatic Generation of Top-level Ports

If the signal mode as defined in the ::mode column matches one of I, O or IO, MIX will create top level ports. This feature can be disabled by the configuration variable output.generate.inout.

Possible values are `mode` (enable top level port generation) and `noxfix` (do not prefix and postfix generated port names for top level ports), which are set by default. To exclude some signals from automatic top-level port generation, use `output.generate.xinout sig_foo,sig_bar`. This feature is useful if you need to use internal inout signals, but do not want to have them wired to the top-level.

Special CONN Sheet Signals

Special signal names:

```
%LOW%, %HIGH%, %LOW_BUS%, %HIGH_BUS%, %OPEN%
```

%OPEN% vs. open

Use 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` should have some extra pins.

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

This will yield a port a at module bar with a width of 10 bit. The two higher bits will be left open.
You could also force the `::in` pins to high or low, instead, e.g. use:

```
%HIGH%, %HIGH_BUS%, %LOW% or %LOW_BUS%.
```

Or any other constant.

Constants

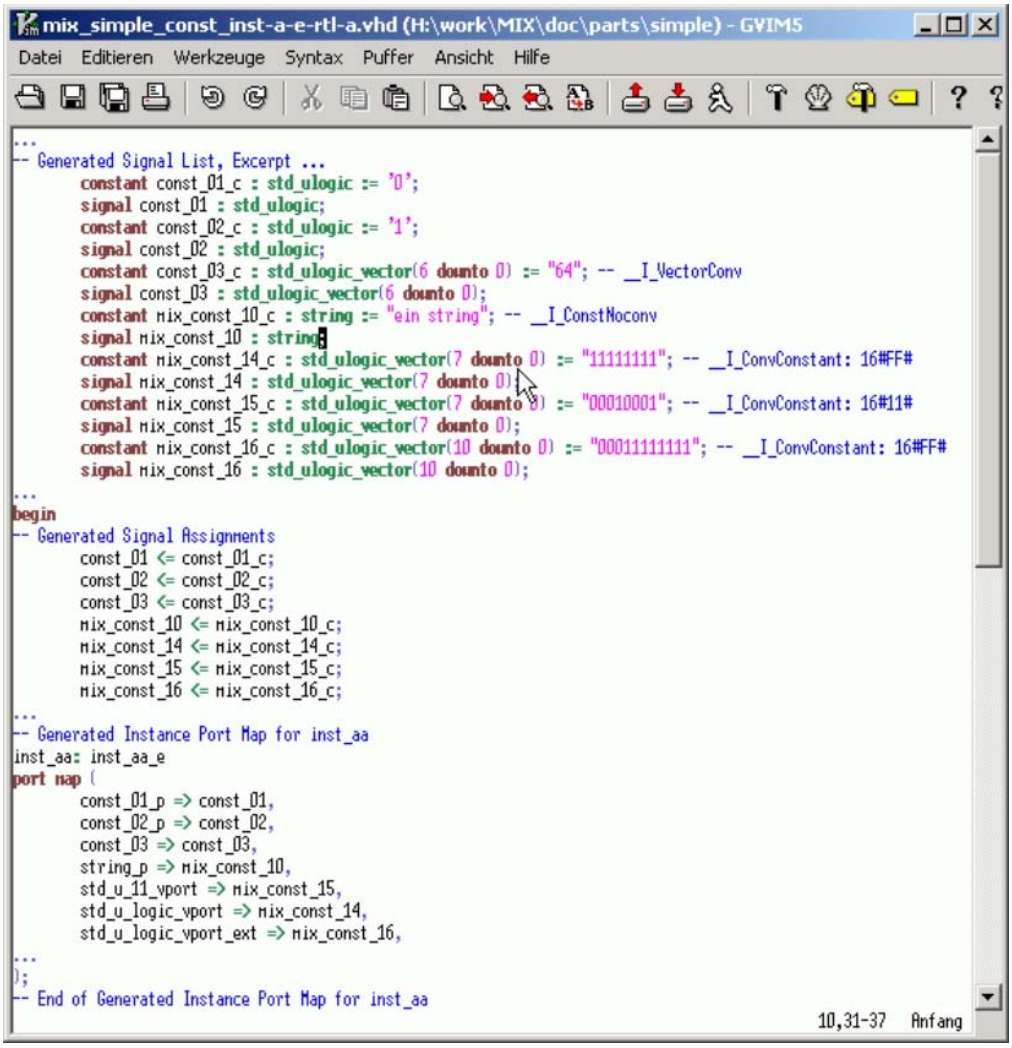
Constant values can be defined and used in several ways in the CONN worksheet. Constants can be marked with a `C` in the `::mode` column. Basically anything resembling a number written in the `::out` column will be considered to be a constant value. String and bit vector constants are enclosed in single 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.

::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, MIX will produce something like:



Search for lines with `__E` in the comments to detect constants MIX was not able to translate properly.

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.

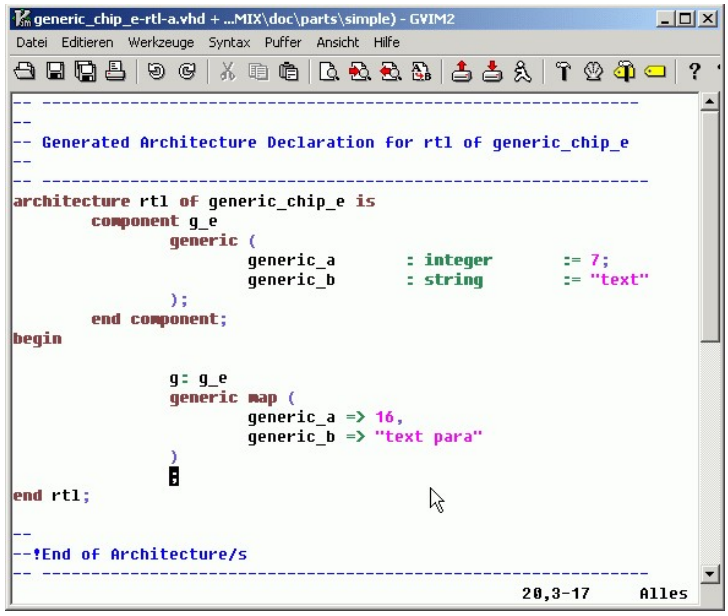
A `P` in the `::mode` column marks parameters. The value given in the `::out` column will be applied to the instances in the `::in` column.

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...

::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

...will create the following output in VHDL:



Global Configuration File

output.generate.portmapsort

The output.generate.portmapsort parameter defines the sort order for portmaps in instantiations and component declarations. You can set any combination of the keys in the following list, seperated by whitespace or comma:

key	description
alpha	strict alphabetical sort order (default)
input	sorted as defined in the input sheets
inout	"in" before "inout" before "out"
outin	"out" before "inout" before "out"
genpost	print generated port at the end
genpre	print generated ports at the beginning
::COL	sort alphabetical by using the values defined in the ::COL column

The sort criteria are applied from left to right. Thus "inout,input" will separate in/inout/out ports and print them in the order of apperance or generation.
Caveat: ::COL is experimental!

check.signal: Controlling Open Port Handling and Misc. Signal Checks

key	description
load	run the signal load check and report all signals without an appropriate load
driver	run the signal driver check and report all signals with missing or multiple drivers
check	used internally
top_open	automatically connect to open, if the signal is not used in it's top level

Per default, all keys are set. Specify a comma separated list to modify this.

check.hdlout: Verify created HDL files with respect to references

Use the -verifyentity PATH option or set check.hdlout.path to start verification of the generated HDL files against existing files. There are several ways to achieve a match of the MIX generated files with the existing files, please change the configuration parameters below accordingly.
Differences are reported in files with the */.ediff* extension. A summary will be printed to the log file and on the screen.
Recommended usage: To check the MIX generated leaf modules, create a verification directory (e.g.*my_entities*) and link all leaf entities to that directory. Apply the following set of parameters:

```
# Adding path for entity verifications:
MIXCFG check.hdlout.path my_entities
MIXCFG check.hdlout.mode entity,leaf,generated,ignorecase
```

Available keys and values for check.hdlout.KEY are:

```
'mode' => "entity,leaf,generated,ignorecase",
ignorecase|ic : ignore case of filename
entity|module|arch[itecture]|conf[iguration]|all: which objects to check
generated : compare generated objects, only (ignores other files in path)
inpath : report if there are extra modules found in path
leaf : only for leaf cells
nonleaf : only non-leaf cells
dcleaf : dont care if leaf (all modules)
```

```
'path' => "",
```

if set to PATH[:PATH:...], MIX will check generated entities against entities found there.

`'delta' => "",`

define how the diffs are made, see output.delta for allowed keys. If it's empty, take `output.delta` contents.

`'filter' =>` (experimental, do not use)

`'extmask' =>` (experimental, do not use)

`output.filter.file`: Filter Files to be Written

To suppress the generation of certain files, list the instances in the configuration parameter `output.filter.file`. A trailing modifier like `arch:`, `enty:`, `conf:` or `module:` makes the filter apply only to that variant of the instance.

Example:

```
output.filter.file inst_a,conf:inst_b
```

Remember that the parameters `output.generate.(arch|enty|conf)` can be used to generate hdl output for leaf cells or not. Default is `noleaf`. Setting to `leaf` will yield HDL description for leaf cells, too.

`output.generate.verimap.modules`: Mapping Verilog Modules for Testing Purposes

When the `output.generate.verimap.modules` paramter is set to a perl regular expression, MIX will create an `ifdef` wrapper around verilog modules instantiations matching the expression.

Example:

```
`ifdef exclude_module_1
// instantiate dummy module
assign sig1_s = 1'b0;
assign sign_s = 1'b0;
`else
module_1 ( .sig1_o (sig1_s); .sign_o (sign_s) );
`endif
```

The module mapping is controlled by the config keys `output.generate.verilog.verimap.*`.

```
# Enable the default wrapper: fixed values:
MIXCFG output.generate.verimap.modules ALL
MIXCFG output.generate.verimap.sigvalue sig_08=1,inst_a[ab]/sig_0.?=some_val
```

`modules` and `sigvalue` take a comma or whitespace seperated list of perl regular expressions.

`modules`: If a module name matches one of the re's from `modules`, this module will be mapped.`ALL` is an other way of saying ".*".

`sigvalue`: Specify the value to be assigned to in the from of `inst_re/signal_re=value` or `signal_re=value` or `value`. Default is "0". In case the value is "1" or "0", it will be automatically extended to `W'b1(xW)` to match the width of busses. All other values will be used literally.

This function is very limited currently, please report problems. E.g. it will not cope well with partial bus assignments and signals connected more then once. VHDL output ist not supported. This parameter was added with the availabity of mix/1.5 (20060411).

`output.generate.emumux.modules`: Insertion of Emulation Multiplexers

When the `output.generate.emumux.modules` parameter is set to a perl regular expression and/or a list of comma seperated modulenames, MIX will create an `` `ifdef `` wrapper around the verilog modules' portmaps to provide configurable input insertion points. By default, all input signals of applicable modules will be routed through multiplexers.

Verilog output example:

```
...
`define insert_emu_mux_inst_aa
// Emulator Data Injection Path, generated by MIX
wire emu_mux_inst_aa = 1'b0;
wire sig_04_emux_s;
wire sig_04_vc_s;
assign sig_04_emux_s = emu_mux_inst_aa ? sig_04_vc_s : sig_04;
`endif
// Generated Instance Port Map for inst_aa
ent_aa inst_aa (
.port_aa_1(sig_01),
.port_aa_2(sig_02[0]),
.port_aa_3(sig_03),
`ifdef insert_emu_mux_inst_aa
.port_aa_4(sig_04_emux_s), // Input for inst_aa
`else
.port_aa_4(sig_04), // Input for inst_aa
`endif
.port_aa_5(sig_05),
...
```

The following configuration parameters in the `output.generate.emumux` namespace are available:

- `modules` Modules matching this regular expression are subject to emumux insertion. This parameter takes a list of comma separated module names and/or perl regular expressions.
- `sigselect` Only signals matching this list or regex are muxed, if left empty it defaults to '.*' for input signals. See also the `options` parameter.
- `select` preset to `emu_mux_%::inst%`, MIX uses this value as mux select signal name. The name will be macro expanded. `%::inst%` is set to the current module name.
- `define` preset to `insert_emu_mux_%::inst%`, MIX uses this value as `` `define `` selector. The name will be macro expandend.
- `muxsnameo` preset to `%::name%_emux_s`, naming for generated signal from mux to port. `%::name%` gets expanded to the current signal name by the macro expansion.
- `muxsnamei` preset to `%::name%_vc_s`, naming for generated signal, insertion signal. Subject to macro expansion.
- `options` defaults to `in`, defines extra (global) options like in | out | inout | leaf | nonleaf. MIX internal usage, only.

An alternative way of inserting emulation multiplexer is by adding a column `::emumux` in the hierachy sheet. Each instance there with a non empty `::emumux` cell will be subject to emulation multiplexer insertion, too. You can set all of the above listed options there in the format `key=value ,key2=value`. They will have precedence over the global options for this instance.

The emumux expansion is available for verilog modules, only. Please be aware, that special cases of port and signal splicing might be wired incorrectly.

Writing Report for Module Connectivity

Use the option `-report portlist` to write a report about the module connectivity in MIF format.

The following config keys are available to influence the output:

TODO ich habe -report portlist entsprechend erweitert (Plan b, war einfacher). Ausserdem gibt es jetzt zusätzlich folgende Optionen/Konfigurationen

```
-delta -> auch fuer mif wird ein delta zum vorherigen Mif geschrieben ( MIXCFG report.delta 1)
MIXCFG report.portlist.name 'name' => "",
# Define report file name, if empty take name of main module
# INST := name of last instance + _portlist.mif
# ENTY := name of last entity + _portlist.mif -> die Ausgabedatei heisst nicht mehr mix_portlist.mif,
sondern entsprechend dem Top-Modul bzw. der Instanz oder Entity
MIXCFG report.portlist.data 'data' => 'port', # Print out port names, not signal names!
MIXCFG report.portlist.split 'split' => 'external::extc,instance',
# Generate seperate portlist for
# external : if column ::external has content
# external::foo : use column ::foo as trigger
# instance : generate a table for each instance
# file[::(INST|ENTY)]
# : generate a file for each instance(*)/entity
# combine with 'name' = INST or ENTY!! -> z.b. mit file::INST, würde man bei einer Schaltung
mit mehreren Blöcken mehrere Dateien bekommen mit je einem Block. Per default gibt es mehrere
Tabellen in einer Datei.
MIXCFG report.portlist.sort 'sort' => 'input',
# or alpha or ::col ....
# pinlist sort order. See portmapsort 'comments' => "0,striphash",
# Limit the number of comment lines to N; 0 -> unlimited
# To switch off all, set report.portlist.comments=""
# striphash := remove leading signs from the comments
```

Macros

LU, 11 March 2009 (created 11 March 2009)

no tags

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. E.g. the pseudo signals `%OPEN%` and `%LOW_BUS%`.

Predefined and User Macros

The following table (XXXwhere is the listXXX) 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.

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.
Examples:

```
MIXCFG macro.%VERILOG_DEFINES% `include bar.vh
```

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 parser 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 assigned `ls`, $5 becomes `min`.

If MX and MH match, the accompanying MD lines are executed. Variables are replaced by their currently assigned value. 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.).

Generator Statements

LU, 11 March 2009 (created 11 March 2009)

no tags

Generator Statements

Hierarchy Generator Operators

Another powerfull feature of MIX are the generator statements. The generators are applied after initial tool setup and after the connection macro evaluation was done. First all generators from hierachy get applied, then those from connectivity sheets.

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 a perl(1) regular expression, but with some extensions and specials. If not defined otherwise, the generator match expression is evaluated against all instance names known at the time of evaluation. Special keywords allow to change to the signal namespace.

Constructor Generator Statement

MIX takes the variable and the range defined 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.

Match Generator

`match_expression` is evaluated against each instance name defined at that time. If this yields true, the line gets executed. Parts of the expression in parentheses are used to set the variables `$1,$2, ...` as you would expect this in perl regular expressions. See the perl regular expression man page perlre(1) for more details. These 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, otherwise the current instance definition gets overloaded.

Example:
see *macro.xls*

Bounded Match Generator

By adding a run variable and a range `$i(N..M)`, a match generator can be restricted to only apply if the variable `$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. Only already existing instance names can successfully match this expression.

Example:
see *macro.xls*

Advanced Features of the Match Generators

The match expression can contain references to all fields of the matched 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 not 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?)/
```

Additional Information

Match generators work for signals and instances 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 known signal names, prefix the whole expression by the `CONN` keyword.
To access single bits from signals, mark the generator with the `CONN:SPLICE` keyword. These generators will be executed for each signal bit, the variable `$s` will provide the bit number.

IF/ELSIF/ELSE

Since the release from 2007-03-06 you can apply the keywords

```
IF, ELSIF and ELSE
```

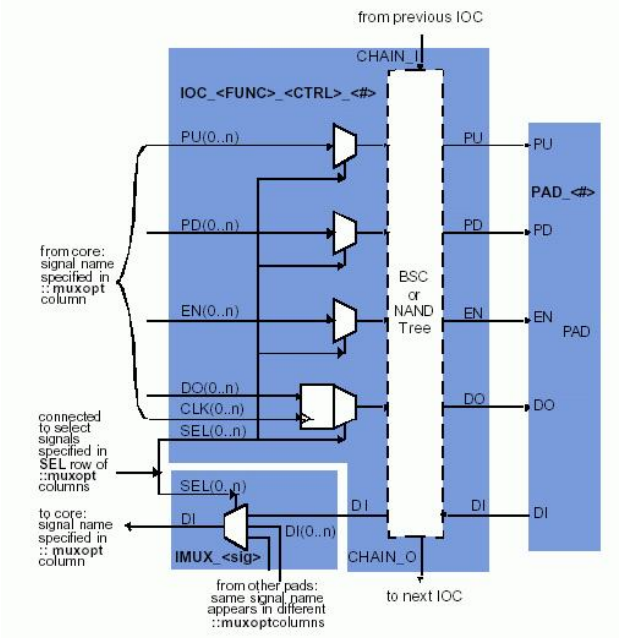
to bind consecutive generator lines together. IF is optional. The keyword ELSIF and ELSE at the beginning of the `::gen` cell will make sure the generator line gets executed only in case the predecessor line did not match.
Please have a look into the distributed *howto.xls* and the *macro.xls* in the test-case directory to see more examples.

IO-Specification

LU, 11 March 2009 (created 11 March 2009)

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.

no tags



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.
	%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		

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 signal `core_sig`. `::muxopt` is allowed to appear several times. The number of signals and the order matches the port names in the `::port` field.

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 design's 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.

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.

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

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%`

RegisterSpecification

Lu, 3 July 2009 (created 11 March 2009)

The Register-Master sheet is one more category of input specification. It describes configuration register domains of devices. A domain comprises all registers that have a common HDL implementation (*register-shell*). A register domain contains registers which in turn contain fields. The MIX tool creates views of a register domain, which is an implementation. Additionally, it can create reports which are used for documentation.

no tags

Templates for the .xls sheet (*registermaster_template*.xls*) can be found in the tool installation directory.

The input can also be in the Spirit Consortiums IP-XACT format.

There are also other in-house tools reading this format, which results in some legacy column data that is not used by MIX.

Column Name	Type	Description	Default Value	Required	Example
::ign	string	Ignore line		opt.	# comment
::type	string	Register type; not used by MIX	*I2C*	man.	I2C
::dev	string	Device name for boardlevel purposes		opt.	VGCH
::width	int	Width of register in bits	32	man.	32
::sub	int	Address offset of register in domain in bytes	0x0	man.	0x20
::addr	int	Not used by MIX	0x0	opt.	0x1234
::interface	string	Domain name		man.	sci_fe1
::block	string	Function block name		opt.	fe_ycdet
::inst	string	Register name		man.	IP_CTRL1
::dir	string	Direction; access type of field	RW	opt.	W
::auto	string	Not used by MIX		opt.	N
::spec	string	Additional field access attribute		opt.	W1C
::update	string	Not used by MIX	N	opt.	N
::sync	string	Name of update signal for shadowed fields		opt.	upd_p12
::clock	string	Clock domain of field		opt.	clk_sci
::reset	string	Reset signal of field		opt.	res_sci_n
::b	string	Bit <n> of field (as <field_name>.<n>)		man.	status.2

::init	int	Reset value of field	0	opt.	0x0
::rec	int	Recommended value of field	0	opt.	0
::range	string	Value range of field	0..1	opt.	0..15
::view	Y|N	Include in documentation or not	Y	opt.	N
::vi2c	string	Register type; not used by MIX		opt.	GI2C_32Bit-Register
::name	string	Name of field (same as in ::b); must be unique per domain		opt.	status
::definition	string	Definition of field instance		opt.	status
::fgroup	string	Functional group of field		opt.	fe_control
::clone	0|1	Field is clonable (1) or not (0)	0	opt.	0
::comment	string	Comment/Description for field		opt.	This is a comment

Register Views Global Configuration Parameters

The configuration is done in the *mix.cfg* file like for the other MIX features. Note: not all parameters are documented here, refer to the *Globals.pm* file in the installation.

Key	Value	Default	Description
i2c.xls	string	I2C	Sheet name in Workbook
i2c.regmas_type	<VGCA|FRCH>	VGCA	to accommodate different styles of register-master *.xls/.sxc* sheets
intermediate.xls_dump	0|1	0	if 1, dumps register-master after parsing in .xls format
intermediate.yaml_dump	0|1	0	if 1, dumps register-master after parsing in .yaml format
reg_shell.type	string	HDL-vgch-rs	Name of the view to generate, can be comma-separated list

Available Register-Views

The following views are currently implemented. They are selected via the `reg_shell.type` parameter. The most important ones are further documented below.

View	Description
hdl-ihb-rs	Verilog IHB (internal host bus) register-shell
hdl-vgch-rs	Verilog OCP register-shell
e_vr_ad	*e* language macros for the *VR_AD* eVC
stl	Register test file in Socket Transaction Language (STL) format
rdl	Denali RDL representation (experimental)
ip-xact	IP-XACT compliant XML output
reglist	Documents all registers in *.mif* file
header	Generates c-header files
vctyheader	the same but takes top-level addresses from *device.in* file
per	Creates Lauterbach *.per* file
vctyper	the same but takes top-level addresses from *device.in* file
perl	creates Perl package (?)
vctyperl	the same but takes top-level addresses from *device.in* file
bd-cfg	view for creating command-files for chip backdoor configuration (FRC project)
none	generate nothing (useful to bypass the view generation dispatcher)

Cloning

In case a design requires multiple instantiations of a register-shell it is not necessary to duplicate the table-based input. MIX can instead *clone* a register domain into a new domain containing <n> uniquified copies of the registers/fields or into <n> new domains.

Cloning Configuration Parameters

Key	Value	Default	Description
reg_shell.clone.number	int	0	Number of clones; if 0, no cloning is applied. if 1, only one clone will be generated, i.e. the naming scheme will be applied
reg_shell.clone.addr_spacing	int	10	number of address bits reserved for every clone
reg_shell.clone.field_naming	string	%F_%N	naming scheme for fields, see **Naming Schemes**
reg_shell.clone.reg_naming	string	%R_%N	naming scheme for registers, see **Naming Schemes**
reg_shell.clone.domain_naming	string	%D_%N	naming scheme for domains, see **Naming Schemes**
reg_shell.clone.unique_clocks	<0|1>	1	If 1, uniquify clock-names of cloned fields

reg_shell.clone.unique_domains	<0|1>	0	If 1, will create new domains for clones; if 0, cloned registers are fitted into old domain

Packing

In case the Register-Master contains registers of a different width than what is required by the HDL implementation, they can be packed/unpacked to the new width. Currently only 64-bit to 32-bit and 32-bit to 16-bit are implemented. The output is captured in the intermediate files (configuration parameter e.g. `intermediate.xls_dump`).

Packing Configuration Parameters

Key	Value	Default	Description
reg_shell.packing.mode	<none|64to32|32to16>	none	packing mode
reg_shell.packing.endianness	<big|little>	big	endianness for registers after packing
reg_shell.packing.postfix_reg_lo	string	_lo	postfix for name of lower portion of splitted register
reg_shell.packing.postfix_reg_hi	string	_hi	postfix for name of higher portion of splitted register
reg_shell.packing.addr_offset	int	0	add an offset to each register address of the packed register-space
reg_shell.packing.addr_factor	int	1	address factor for transforming register addresses into the packed register-space
reg_shell.packing.addr_domain_reset	<0|1>	0	specifies whether the transformed address is reset at domain start

HDL Register-Shells

`reg_shell.type HDL-vgch-rs` or `reg_shell.type HDL-ihb-rs`

This generates a Verilog OCP/IHB bus register-shell for the register domain. The specification for this can be found here: OCP Config Register Specification Unix Windows

Note that this also uses an RTL library (from projects ip_sync and ip_ocp). MIX dumps the file *rtl_libs.xml* containing information about the RTL library versions to be used with the generated code.

Register-Master Configuration Parameters

Key	Value	Default	Description
reg_shell.bus_clock	string		name of config bus clock for domain
reg_shell.bus_reset	string		name of config bus reset for domain
reg_shell.addrwidth	int	14	width in bits of config bus address signal
reg_shell.datawidth	int	32	width in bits of config bus data signal (same as register-width)
reg_shell.multi_clock_domains	<0|1>	1	If 1, generate separate register blocks for all clock domains
reg_shell.infer_clock_gating	<0|1>	1	If 1, insert extra logic for power-saving
reg_shell.infer_sva	<0|1>	1	If 1, insert SystemVerilog assertions into HDL-code
reg_shell.read_pipeline_lvl	int	0	Parameter that controls the read-pipelining; If 0, no read-pipelining will be inserted
reg_shell.read_multicycle	int	0	can be one of [0,1,2,..] to insert cycle delays for read acknowledge
reg_shell.exclude_regs	string[,string]*		Comma-seperated list of register names to be excluded from generation
reg_shell.exclude_fields	string[,string]*		Comma-seperated list of field names to be excluded from generation
reg_shell.exclude_domains	string[,string]*		Comma seperated list/pattern of domain names to exclude, currently only used for view bd-cfg
reg_shell.prefix	string	rs	file/module prefix for register-shell
reg_shell.enforce_unique_addr	<0|1>	1	if 1, allow only one register per address
reg_shell.device	string	%EMPTY%	identifier used for device member of Reg object; if empty, uses the `::dev` column from Register-Master sheet
reg_shell.field_naming	string	%F	naming scheme for fields, see **Naming Schemes**
reg_shell.reg_naming	string	%R	naming scheme for registers, see **Naming Schemes**
reg_shell.domain_naming	string	%D	naming scheme for domains, see **Naming Schemes**

VR_AD Macros

`reg_shell.type e_vr_ad`

This generates e-code for the VR_AD eVC used for configuring/monitoring DUT registers in a Specman test environment. It generates the macro calls to define the registers and fields in the specified domain.

VR_AD Configuration Parameters

Key	Value	Default	Description
reg_shell.e_vr_ad.path	string	e	output dir for e-code
reg_shell.e_vr_ad.regfile_prefix	string	MIC	prefix for the generated vr_ad_regfile type
reg_shell.e_vr_ad.regfile_name	string	(default is concatenated domain names)	name of the generated vr_ad_regfile type

reg_shell.e_vr_ad.file_prefix	string	regdef	prefix of generated file
reg_shell.e_vr_ad.vplan_ref	string	%EMPTY%	VPlan reference for register coverage
reg_shell.e_vr_ad.field_naming	string	%1F	naming scheme for fields, see **Naming Schemes**
reg_shell.e_vr_ad.reg_naming	string	%uR	naming scheme for registers, see **Naming Schemes**
reg_shell.e_vr_ad.cover_ign_read_to_write_only	<0|1>	0	for coverage
reg_shell.e_vr_ad.cover_ign_write_to_read_only	<0|1>	1	for coverage

STL Stimuli Files

reg_shell.type STL

This generates a Socket Transaction Language file for testing DUT registers. The generated file has two sections, *head* and *body* which are enveloped by source-code pragmas (//pragma MIX_STL...). If the output file is generated for the first time, it will be created from a template. If the output file already exists, it is read and only the head and body sections are replaced. This way, the user can make additions to the stimuli file.

STL Configuration Parameters

Key	Value	Default	Description
reg_shell.stl.initial_idle	int	100	value for STL initial_idle statement
reg_shell.stl.exclude_regs	string[,string*]		comma separated list of registers to exclude from STL generation
reg_shell.stl.use_base_addr	<0|1>	0	STL feature base-addr

IP-XACT view

reg_shell.type ip-xact

This dumps the register database in *.xml* format. The configuration parameters have the key xml (see *Globals.pm* from the installation dir for details). The parameters are also valid for XML input.

Naming Schemes

LU, 12 March 2009 (created 11 March 2009)

no tags

Register, Field and Domain names are used as-is in the generated views, for cloned objects a number is attached. This naming scheme can be changed (separately for view generation and cloning). This is useful e.g. if the source specification does not contain unique field names (required for HDL register-shell generation).

The keys of the configuration parameters are

```
reg_shell[.clone|e_vr_ad.]<field_naming|reg_naming|domain_naming>
```

where the value consists of a string pattern that can contain normal characters or placeholders of the form

```
%[<u|l>]<D|R|F|B[<d>]N>
```

with the substitutes

- D for domain name
- R for original name of register (only available in reg_naming and field_naming)
- F for original name of field (only available in field_naming)
- B for block name (only available in field_naming)
- N for a decimal number; can be preceded by a number to fix the number of digits used in representation

The %N pattern can be followed by a simple arithmetic expression like +32 which be evaluated for the number.

The u or l in front of the substitute force upper/lowercase (optional).

Example 1: Register name reg_x of the 7th clone of a clonable register

```
scc_%2N_%uR
```

creates name scc_06_REG_X from original name.

Example 2: Field name CLEAR_INT in the domain si2ocp

```
%lD_%lF_config
```

creates name si2ocp_clear_int_config.

Config+Commandline Options

Lu, 24 June 2009 (created 12 March 2009)

no tags

Command Line Switches

The command line options take precedence over the *mix.cfg* settings.

Note: list is not complete.

- `-cfg <config>` load additional mix configuration file *<config>*
- `-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 <VAR>` Select variant `<VAR>` from the HIER or Register-Master worksheet.
- `-conf <key>=<value>` Overload configuration parameter with value or add a new configuration variable. Example: `-conf reg_shell.type=e_vr_ad`
- `-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".
- `-domain <DOMAIN>` Restrict register-master view to domain `<DOMAIN>`
- `-report portlist` Write portlist and/or register master reports in MIF format.
- `-vinc <verilog-file>` pass a file with Verilog (define) macros that can be used for bit-slices in CONN sheet
- Standard options: `help|h verbose|v quiet|q nobanner debug`

Runtime Options and Configuration

Runtime configuration is controlled by (increasing precedence):
1. built-in default values
2. *mix.cfg* files, if found in *$HOME*, *$PROJECT* and/or in the current directory. Format is: `MIXCFG <key> <value>`
3. *CONF* sheet found in input xls files
4. command line switch: `-conf <key>=<value>`
5. dedicated command line options

MIX reads in mix.cfg configuration files in the following locations:
1. `-cfg <config file>` command line option
2. *$ENV{HOME}*, *$ENV{HOMEDRIVE}*, *$ENV{HOMEPATH}*, *$ENV{USERPROFILE}* or *C:* (only from the first matching location)
3. *$ENV{$PROJECT}*
4. cwd()

Miscellaneous

LU, 13 March 2009 (created 12 March 2009)

no tags

-delta mode

In delta mode, MIX does not change output files, but reports the number of changes in the output files compared to a previous run. Extra sheets DIFF_CONN and DIFF_HIER (and old versions of them) are added to the intermediate output file. The *FOO.pld* internal output gets overwritten, though (if created). Messages are appended to *mix_0.pl.out*

If a new HDL-File needs to be created by the changes, delta mode is not be applied for that file, but it is generated as a new file. If you never have written a *FOO-mixed.xls* intermediate output, no HIER or CONN sheets are 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. Possible values for `output.delta` are (comma seperated list):

- `space`: do not consider whitespace
- `sort`: sort lines before compare (default)
- `comment`: do not remove all comments before compare
- `remove`: remove empyt diff files
- `ignorecase|ic`: ignore case if set

Intermediate Excel Sheet *foo-mixed.xls*

MIX creates the *foo-mixed.xls* intermediate file with the fully evaluated and expanded data from all input sources. The actual output format depends on the platform and kind of input data. An Excel file is created on MS-Windows hosts, while in other cases a csv format will be used.

The file name name will be derived from the last input file name given on the command line with an inserted *-mixed* before the file extension. Format of an 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, e.g HIER_0, HIER_1, ...

Three sheets are created by default: HIER, CONN and CONF. By using various configuration parameters you can change that, however:

- `intermediate.path` defines the storage location. Default is *cwd()*. Will be set by using `-dir <PATH>`, too.
- `intermediate.order` defines the output sort order. Default is `input` (not implemented)

- `intermediate.keep` defines the number of old versions of a sheet to keep, default: 3
- `intermediate.format` is one of: `prev(ious)`, `auto` or `n(o|ew)` ; if set to `prev(ious)` (default) uses old sheet format, `auto` applies auto-format. Others do nothing special.
- `intermediate.strip` if set, all old and diff sheets will get removed
- `intermediate.ext` (unused)
- `intermediate.<xls_dump|yaml_dump>` will also dump intermediate register-master sheets in specified format
- `intermediate.intra` if set create seperate conn sheets for instances, Allowed values are

```
INTRA -> CONN and INTRA
TOP -> CONN_<TOP> and CONN
TOP,INTRA -> CONN_<TOP> and CONN_INTRA
INST[ANCE] -> create one conn sheet for each instance, named: CONN_<instance>
```

- `intermediate.instpre` is prepend to CONN sheet names if `intra` is `inst` . Default: `CONN_`
- `intermediate.topmap` defines mapping of signal modes on sheets keping data of top instances. Default value is `ALL`.Values are
  - `ALL`
  - list of signals (comma seperated list of regular expressions)

If a signal is attached to a top cell and the name matches, the signal mode (I,O,IO,C,P,G, ...) is mapped to `%TM_(I|O|IO|...)%`. These macros are predefined to map back to the original mode, if you run the created CONN sheet through MIX again.

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 parameters in the *mix.cfg* file:

Key	Value
output.path	<hdl-dir>
intermediate.path	<intermediate-dir>
internal.path	<internal-dir>
report.path	<report-dir>
reg_shell.e_vr_ad.path	<e-code-dir>
xml.path	<xml-dir>

writes internal, intermediate and other files to the given pathes. Directory creation is controlled by `output.mkdir` which can be set to `<yes|no>`.

Helper Tools

LU, 13 March 2009 (created 13 March 2009)

no tags

Convert Excel to *sxc* and *csv*

MIX comes with a command line Excel to CSV converter tool. Usage for UNIX is like:

```
$ module load perl mix
$ xls2csv -csv foo[.xls] bar[.xls] [bar2.xls ...]
```

This will convert all sheets in *foo.xls* to *foo.csv*, *bar.xls* to *bar.csv* and so on. As csv only has one sheet, the different input sheets will be seperated by `:=:=:=>` followed by the Excel sheet name. Even the first and maybe only sheet will be marked by such a line.

Now supports the selection of rows and/or columns (see `-column` / `-row` option)

Options:

- `-[no]csv` Don't generate CSV formatter file (on by default)
- `-[no]sxc` Don't generate sxc formatted output file (on by default)
- `-autoq[uote]` Do quoting the Excel style
- `-q[uote] X` Use `X` as quoting char (default: `"`)
- `-sep X` Use `X` as column seperator (default: `;`)
- `-noquote` Do not quote the output
- `-sheet RE` Select only sheets matching the regexp `RE`
- `-[no]single` Write each sheet into a seperate output csv; extend name by sheet name. Default is to write a single csv and/or sxc file per input excel file with seperating the sheets by the sheet header (see `-[no]head`). With `-single` no sheet seperator head will be printed.
- `-[no]accu[mulate]` Combine all excel files into a single output csv or sxc file. Default is to convert a single csv and/or sxc file per input excel file. Basename is taken from the first Excel file name.
- `-[no]head` Do not print sheet seperator head (`=:=:=:=> SHEETNAME`), on by default
- `-[no]verbose` Print more messages
- `-column|(c)range A:B,C` select columns numbered like in Excel or as digit. Ranges start from 1.
- `-row|rrange N:M` select rows
- `-matchc REG_matching` header
- `-matchr RE_matching` first cell contents in row. Possible ranges are: `A..C,EE-FA,G:I,14,16-70,II-` (alpha only for column headers) Repeated options are logically ored (tried one after the other). If column or row selector options are set, only matching

data will be printed.

Example:

```
$ xls2csv -csv -single -sheet my_data -autoq -nohead my_excel.xls
```

will retrieve the sheet "my_data" from the excel document my_excel.xls and store it's content in the file *my_excel*, *my_data.csv* in a format very, very near the one you would get when using "Save As" in Excel and store the worksheet in CSV format.

```
$ xls2csv -csv -row 1-10,1-10 -single my_excel.xls
```

will print the first ten rows two times into the csv file(s) *my_excel-<sheet names>.csv* of each sheet in the excel document. Please consider, that overlapping selections will print the data selected multiple time. *xls2csv* will sort the selectors in numeric start order unless you set `format.csv.sortrange` to 0.

Please remove the created files before updating, otherwise new sheets are added to the existing files, while older version stay with renamed sheet names.

For MS-Win workstations, use

```
K:\Projects\MIX\<version>\xls2csv.pl ....
```

Options and syntax are the same as with UNIX/Linux.

Known Bugs & Limitations

LU, 13 March 2009 (created 13 March 2009)

no tags

1. Parts of this documentation are outdated. See also the *Globals.pm* in the installation for additional features/documentation
2. The HIER/CONN sheet processor does not support multiple instantiations except they are in the same hierarchy. The reason is the flat namespace for *signals* (column `::name` in CONN sheets).
3. If there are problems with loading Perl Modules, do a

```
module unload msd
```

beforehand

Issue Tracking

LU, 13 March 2009 (created 13 March 2009)

no tags

For reporting issues, please the Bugzilla Issue Tracking System (component *MIX*) .

Links

Lu, 3 July 2009 (created 13 March 2009)

no tags

- MIX/IP-XACT presentation
- MIX API presentation
- MIX update presentation
- Register-Master Development site