MIX - Micronas Interconnect Spec expander

Userreference

Author: Wilfried Gänsheimer

$Date: 2007/03/05 09:00:37 $
Version: $Id: MIX_Userreference.lyx,v 1.6 2005/01/26 14:01:59 wig Exp

1 Introduction

MIX is part of the registermaster tools. The registermaster set is comprised of a method and a tool to automatically derive top-level hardware description, low-level driver software and documentation out of a single source structural specification is presented. In contrast to system-level design tools this approach focuses on structural properties like interconnect, hierarchy and control/status registers instead of refining a functional specification. A table-based specification format was chosen to enable efficient data entry and review. Due to automatic code generation consistency between specification and design is guaranteed and error-prone manual transformations can be avoided. This paper focuses on specification and hardware generation.

2 Management Summary

MIX is a simple to use script, which creates all of a chip top level source files from a single input file. This input file is the single source description of hierachy, connectivity, IO wiring and internal control busses like I2C.

3 Table of Contents

Contents

4 Installation

The MIX toolset uses the Perl scripting language. Please install a recent version of Perl on your MS-Windows workstation, e.g. the freely available ActiveState Perl from `S:\Perl\ActivePerl-5.8.4.*.ms`
Alternatively you can use the cygwin environment. Check the software installation hints at the end of this document.
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:\Projects\MIX\PROG\mix_0.pl foo.xls`
- `TODO: Describe MUI (MIX graphical User Interface).`

For UNIX (Solaris/Linux) no user installation is required. Everything needed is installed in `/tools/mix`. Use “`module load mix`” to prepare your environment. You can select a special version, check the output of “`module avail mix`“ to see what is available.

5 Getting Started

To receive useful results, MIX reads in various input date from spreadsheets, typically stored in ExCEL workbooks. At least you will need to prepare a description of your design hierarchy (**HIER**) and the connectivity sheet (**CONN**). If found, 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. The **`I2C`** sheet can be used to describe an on-chip I2C bus and it’s configuration.

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*.

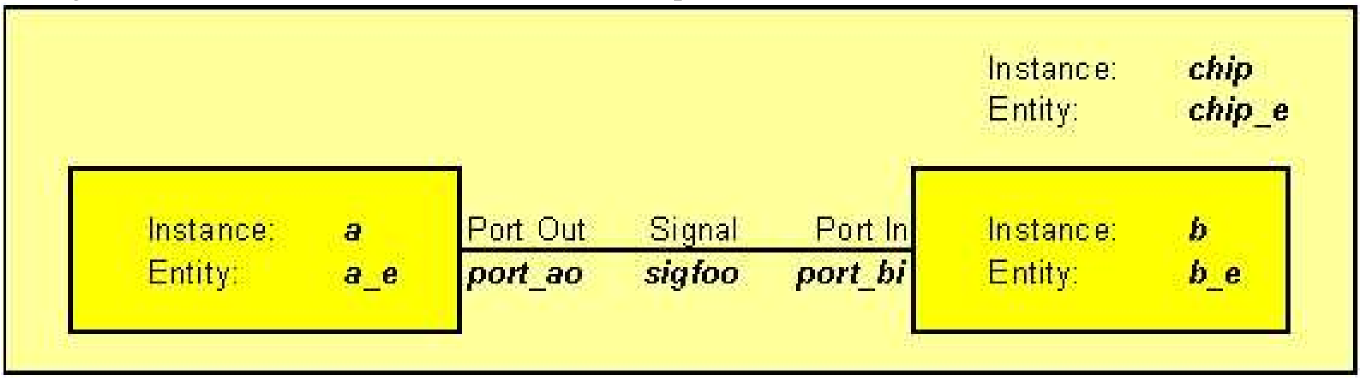
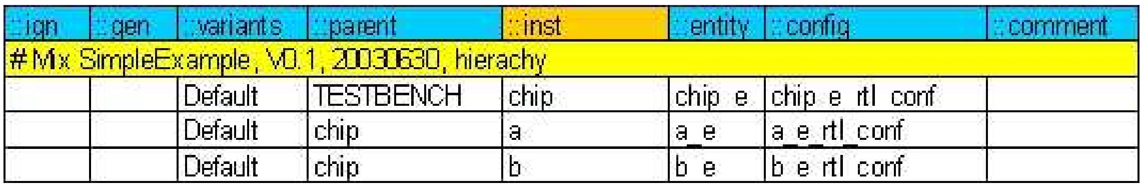


Image 1: mix_simple.xls

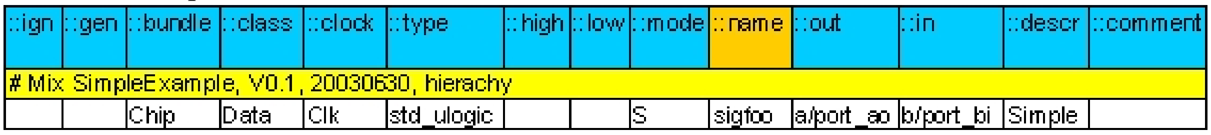
The equivalent description of this simple design in MIX is made up from a worksheet **HIER**.



::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 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**:

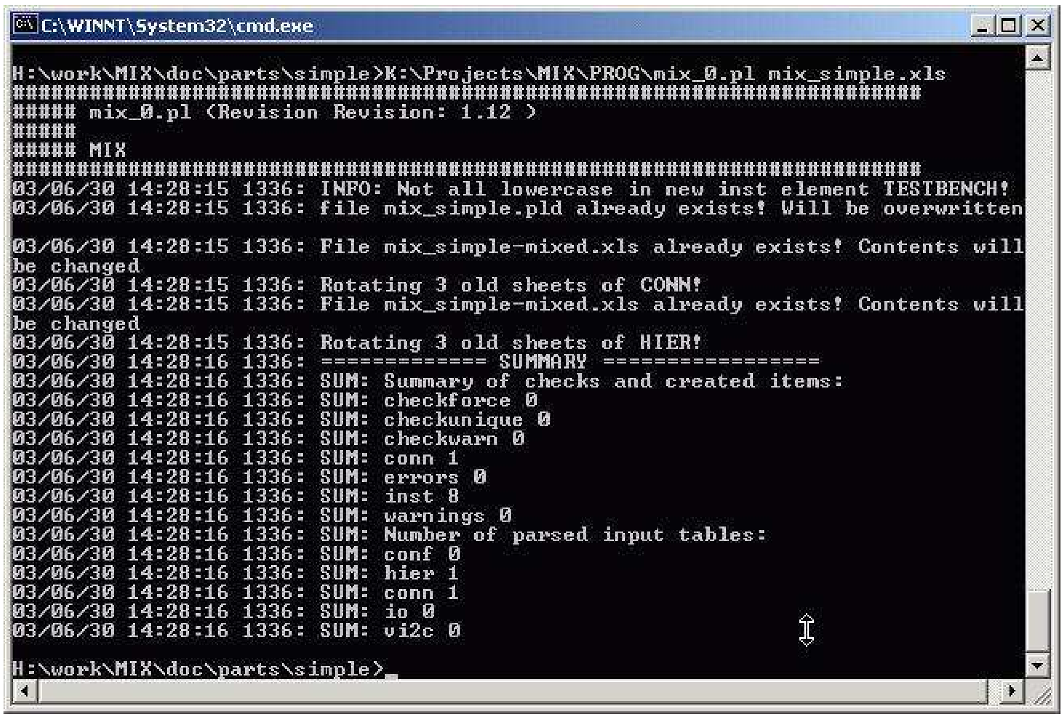


::ign	::gen	::bundle	::class	::clock	::type	::high	::low	::mode	::name	::out	::in	::descr	::comment
# Mix SimpleExample, V0.1, 20030630, hierachy													
		Chip	Data	Clk	std_ulogic			S	sigfoo	a/port_ao	b/port_bi	Simple	

As you see, this worksheet also starts with the table header 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 spreadsheet is stored in:
MS-Windows:
```
   $ K:\Projects\MIX\PROG\mix_0.pl foo.xls
```
UNIX:
```
   $ module load perl mix
   $ mix foo.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.

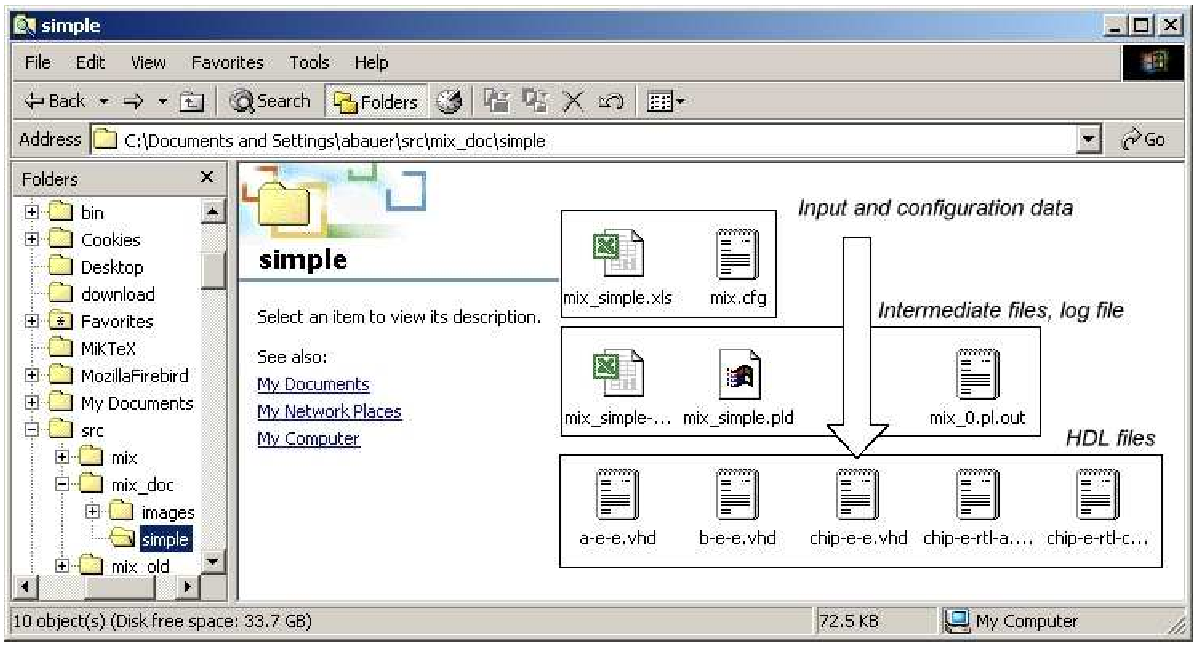


```
H:\work\MIX\doc\parts\simple>K:\Projects\MIX\PROG\mix_0.pl mix_simple.xls
##########################################################################
##### mix_0.pl (Revision Revision: 1.12 )
#####
##### MIX
##########################################################################
03/06/30 14:28:15 1336: INFO: Not all lowercase in new inst element TESTBENCH!
03/06/30 14:28:15 1336: file mix_simple.pld already exists! Will be overwritten

03/06/30 14:28:15 1336: File mix_simple-mixed.xls already exists! Contents will
be changed
03/06/30 14:28:15 1336: Rotating 3 old sheets of CONN!
03/06/30 14:28:15 1336: File mix_simple-mixed.xls already exists! Contents will
be changed
03/06/30 14:28:15 1336: Rotating 3 old sheets of HIER!
03/06/30 14:28:16 1336: ============ SUMMARY ================
03/06/30 14:28:16 1336: SUM: Summary of checks and created items:
03/06/30 14:28:16 1336: SUM: checkforce 0
03/06/30 14:28:16 1336: SUM: checkunique 0
03/06/30 14:28:16 1336: SUM: checkwarn 0
03/06/30 14:28:16 1336: SUM: conn 1
03/06/30 14:28:16 1336: SUM: errors 0
03/06/30 14:28:16 1336: SUM: inst 8
03/06/30 14:28:16 1336: SUM: warnings 0
03/06/30 14:28:16 1336: SUM: Number of parsed input tables:
03/06/30 14:28:16 1336: SUM: conf 0
03/06/30 14:28:16 1336: SUM: hier 1
03/06/30 14:28:16 1336: SUM: conn 1
03/06/30 14:28:16 1336: SUM: io 0
03/06/30 14:28:16 1336: SUM: vi2c 0

H:\work\MIX\doc\parts\simple>_
```

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

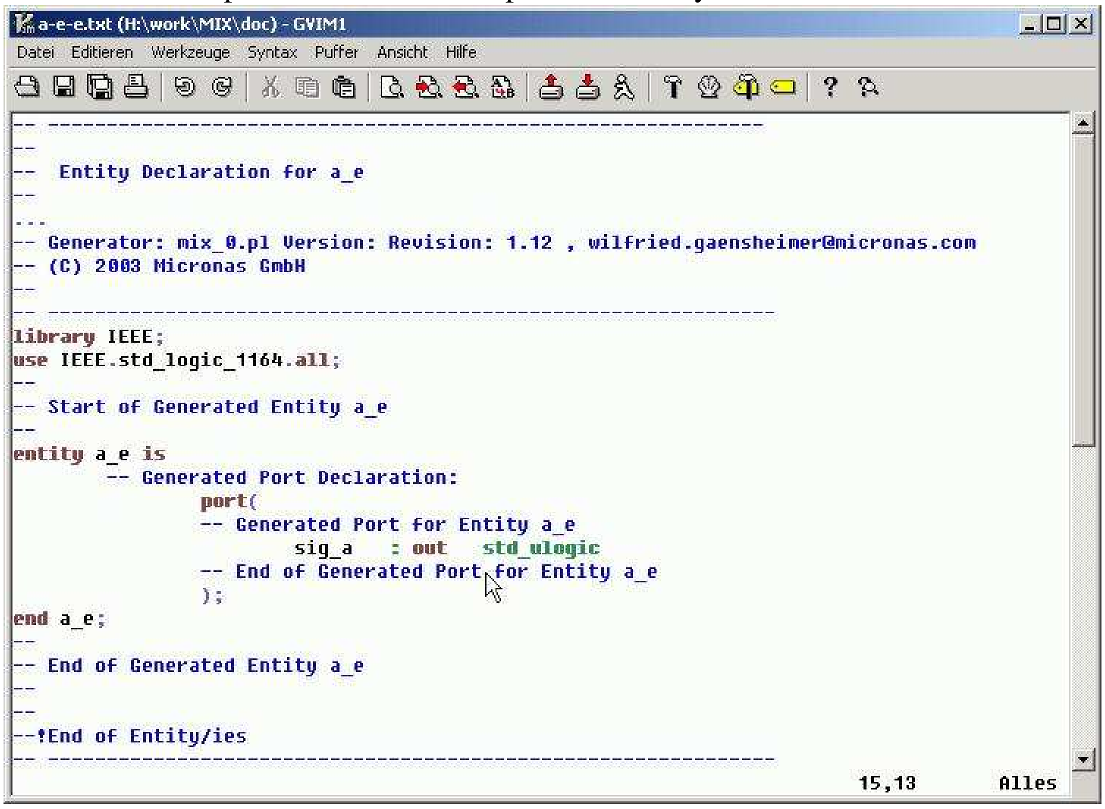
Type	Basename	Extension	Example
Entity	::entity-column	-e.vhd	chip_e-e.vhd
Architecture	::entity-column	-rtl-a.vhd	chip_e-rtl-a.vhd
Configuration	::entity-column	-c.vhd	chip_e-rtl-conf-c.vhd

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

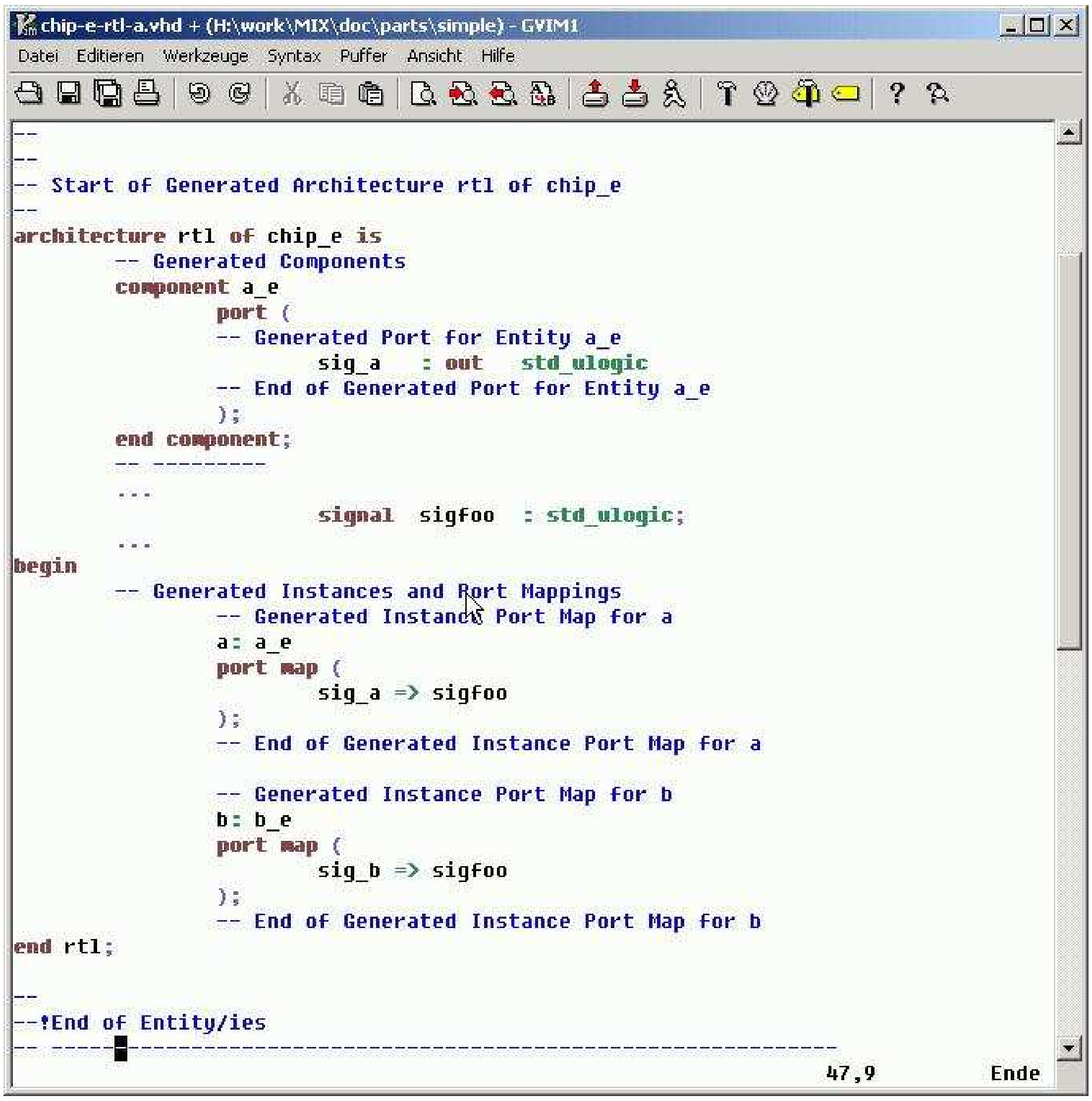
storage for MIX run-time configuration options. The lines "`MIXCFG outarch ARCH`", "`MIXCFG outenty ENTY`", and "`MIXCFG outconf CONF`" switch MIX outputs to separate files for each entity, architecture and configuration. In this case the file names are defined by the element name:

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.


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
```

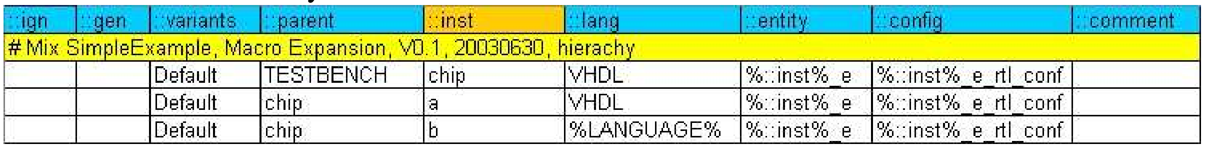
will create the file `bar.xls`, which has the following three worksheet categories:

- empty HIER, CONN and IO sheets
- template sheets with numerous examples TMPL_(HIER|CONN|IO)
- import sheets IMP_HIER and IMP_CONN (only 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.

6.2 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 Star(/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 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 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 full 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.

6.4 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.
Internally the kewords `::debug`, `::hierarchy`, `::skip` and `::default` are used. Please

Column name	Description	Default	Req.	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

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.

6.5 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.

6.5.1 ::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_DECLARAT`
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`

6.5.2 ::udc: Adding 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_LOCAT*
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.

6.6 Special HIER sheet properties

6.6.1 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%` .

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

6.7 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.

6.8 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.

6.9 Automatic Generation of Toplevel 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 usefull if you need to use internal inout signals, but do not want to have them wired to the top-level.

6.10 CONN columns details

6.11 Special CONN sheet signals

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

6.11.1 %OPEN% aka. 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.

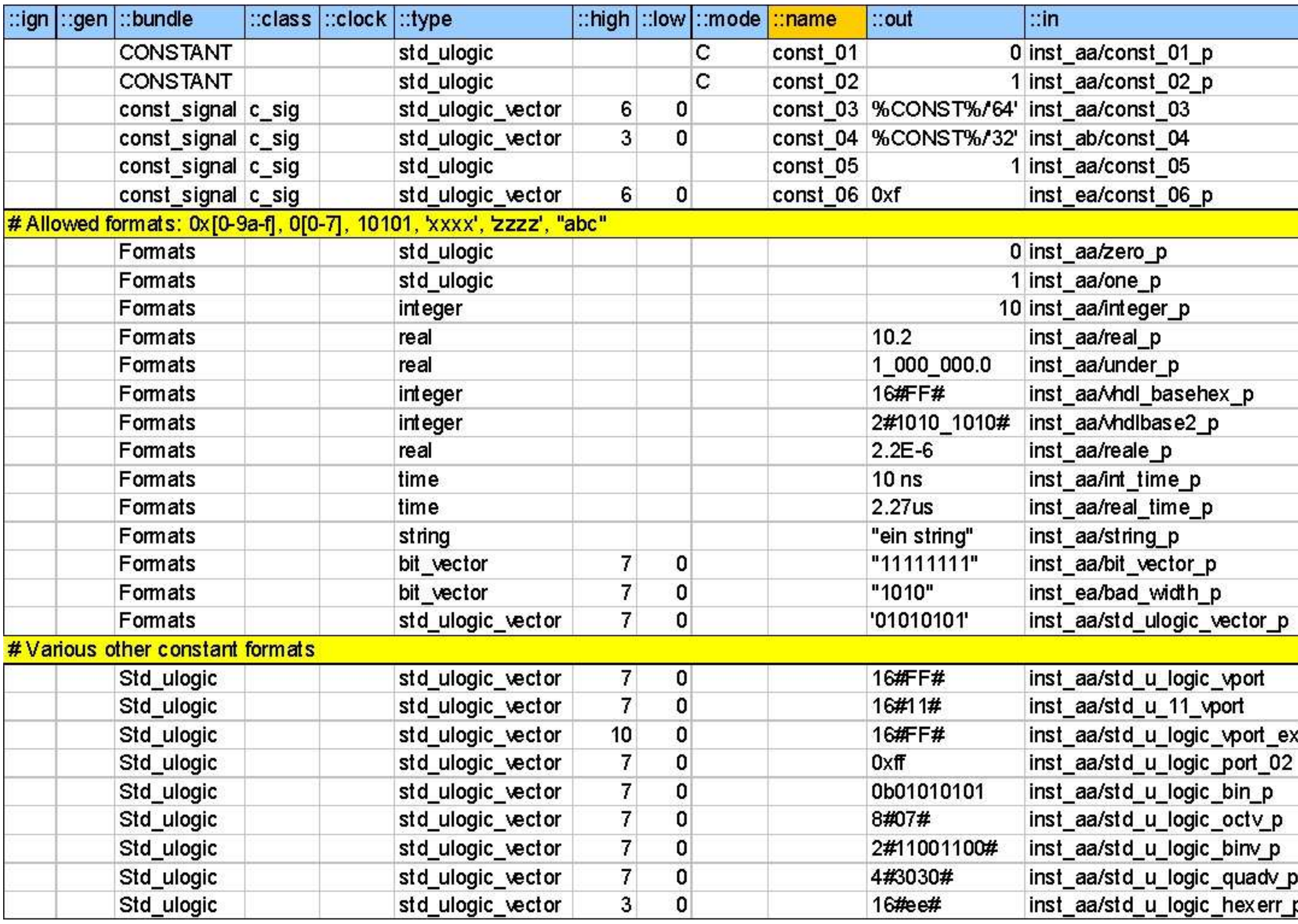
```
::name signal_a, ::high 7, ::low 0, ::out foo/a ::in bar/a
::name %OPEN%, ::high 1, ::low 0, ::in -, ::out bar/a(9:8)
```

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.

6.11.2 Constants

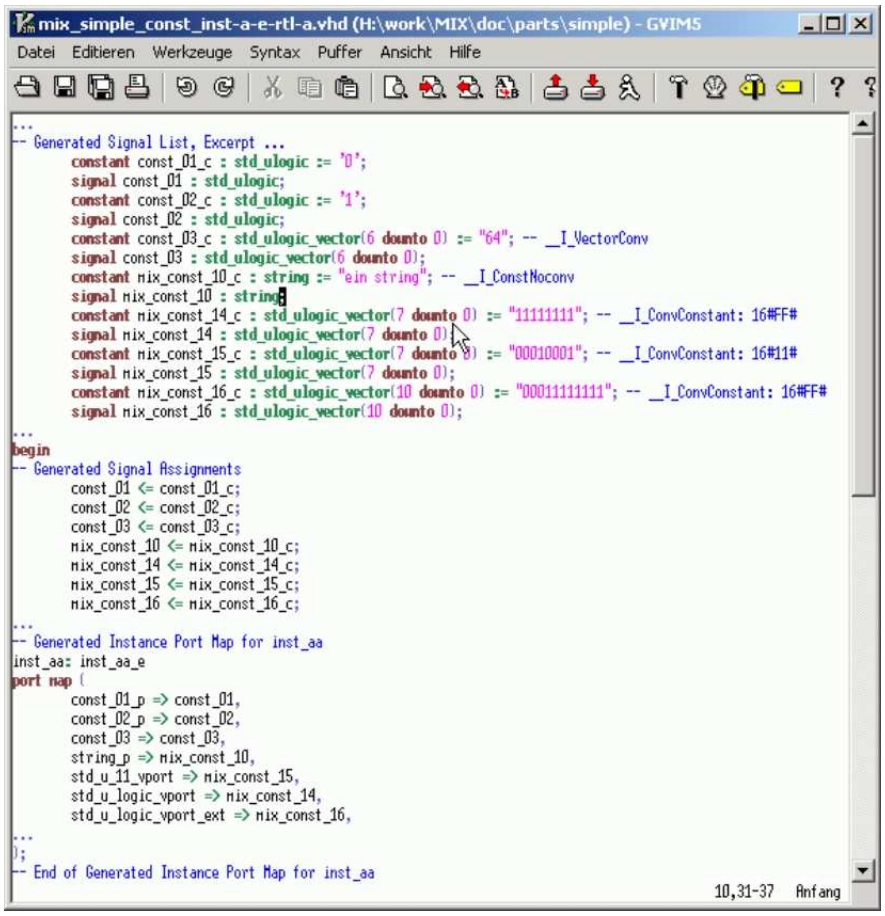
Constant values can be defined and used in several ways in the CONN worksheet. Constants can be marked with a C in the `::mode` column. Basically anything resembling a number written in the `::out` column will be considered to be a constant value. String and bit vector constants are enclosed in single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

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

::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

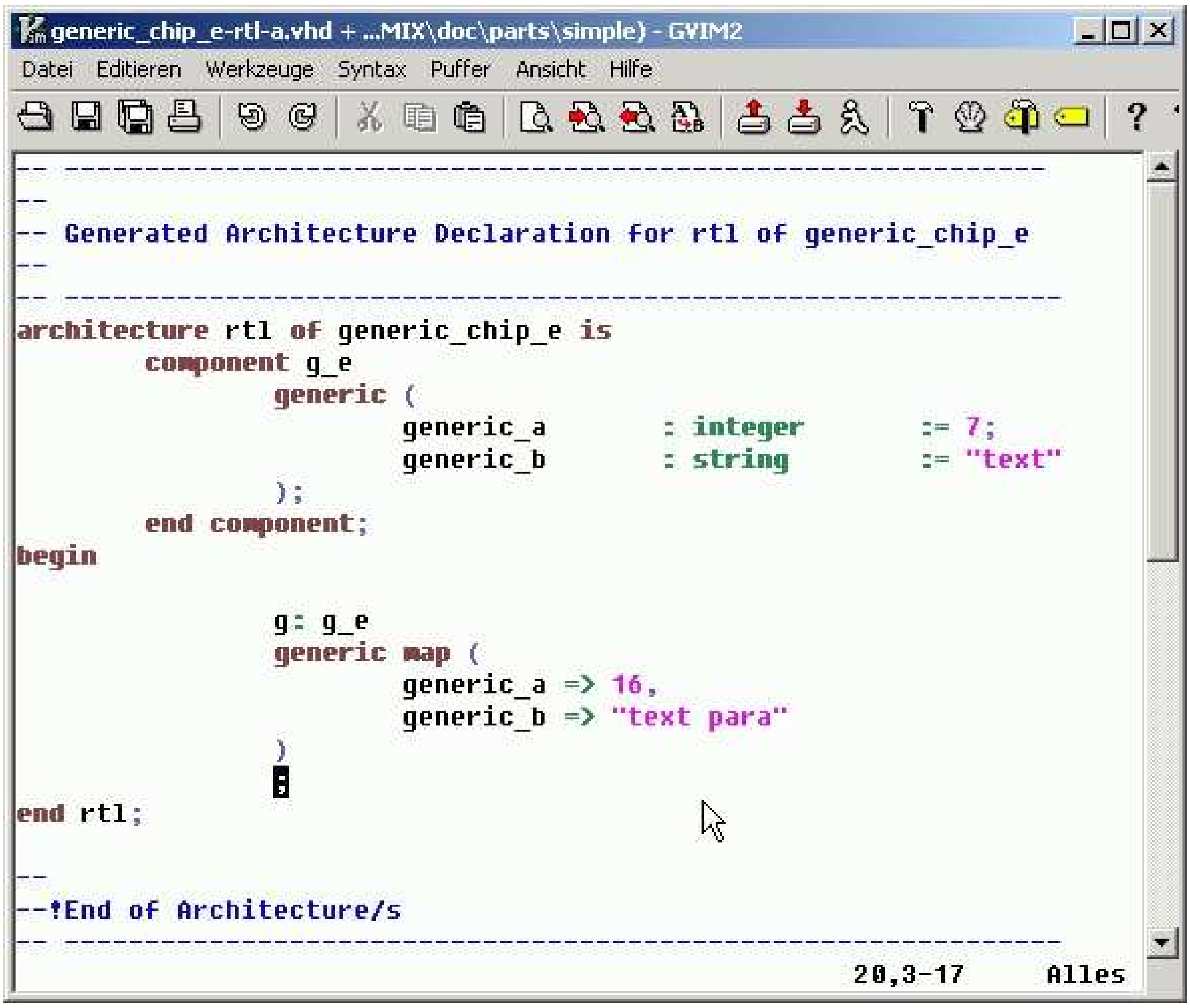

mix_simple_const_inst-a-e-rtl-a.vhd (H:\work\MIX\doc\parts\simple) - GVIM5
Datei Editieren Werkzeuge Syntax Puffer Ansicht Hilfe

```
...
-- Generated Signal List, Excerpt ...
        constant const_01_c : std_ulogic := '0';
        signal const_01 : std_ulogic;
        constant const_02_c : std_ulogic := '1';
        signal const_02 : std_ulogic;
        constant const_03_c : std_ulogic_vector(6 downto 0) := "64"; -- __I_VectorConv
        signal const_03 : std_ulogic_vector(6 downto 0);
        constant mix_const_10_c : string := "ein string"; -- __I_ConstNoconv
        signal mix_const_10 : string;
        constant mix_const_14_c : std_ulogic_vector(7 downto 0) := "11111111"; -- __I_ConvConstant: 16#FF#
        signal mix_const_14 : std_ulogic_vector(7 downto 0);
        constant mix_const_15_c : std_ulogic_vector(7 downto 0) := "00010001"; -- __I_ConvConstant: 16#11#
        signal mix_const_15 : std_ulogic_vector(7 downto 0);
        constant mix_const_16_c : std_ulogic_vector(10 downto 0) := "00011111111"; -- __I_ConvConstant: 16#FF#
        signal mix_const_16 : std_ulogic_vector(10 downto 0);
...
begin
-- Generated Signal Assignments
        const_01 <= const_01_c;
        const_02 <= const_02_c;
        const_03 <= const_03_c;
        mix_const_10 <= mix_const_10_c;
        mix_const_14 <= mix_const_14_c;
        mix_const_15 <= mix_const_15_c;
        mix_const_16 <= mix_const_16_c;
...
-- Generated Instance Port Map for inst_aa
inst_aa: inst_aa_e
port map (
        const_01_p => const_01,
        const_02_p => const_02,
        const_03 => const_03,
        string_p => mix_const_10,
        std_u_11_vport => mix_const_15,
        std_u_logic_vport => mix_const_14,
        std_u_logic_vport_ext => mix_const_16,
...
);
-- End of Generated Instance Port Map for inst_aa
                                                            10,31-37    Anfang
```

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.

6.11.3 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:


generic_chip_e-rtl-a.vhd + ...MIX\doc\parts\simple) - GVIM2
Datei Editieren Werkzeuge Syntax Puffer Ansicht Hilfe

```
-- ---------------------------------------------------------------
--
-- Generated Architecture Declaration for rtl of generic_chip_e
--
-- ----------------------------------------------------------------
architecture rtl of generic_chip_e is
        component g_e
                generic (
                        generic_a       : integer       := 7;
                        generic_b       : string        := "text"
                );
        end component;
begin

                g: g_e
                generic map (
                        generic_a => 16,
                        generic_b => "text para"
                )
                ;
end rtl;

--
--!End of Architecture/s
-- ----------------------------------------------------------------
                                                20,3-17     Alles
```

6.12 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%.

6.13 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
```

6.14 Global configuration parameters

6.14.1 output.generate.portmapsort

The output.generate.portmapsort configuation 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 seperate in/inout/out ports and print them in the order of apperal or generation.
Caveat: ::COL is experimental!

6.14.2 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 multipledrivers
check	used internally
top_open	automatically connect to open, if the signal is not used in it’s top level

By default all the keys are set. Specify a comma seperated list to modify this.

6.14.3 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)

6.14.4 output.filter.file: Filter file to be written

To suppress the generation of certain files, list the instances in the configuration parameter `output.filter.fi`
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.

6.14.5 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_va
```

ALL is an other way of saying ".*".

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 another 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).

6.14.6 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 seperated 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.

6.15 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:

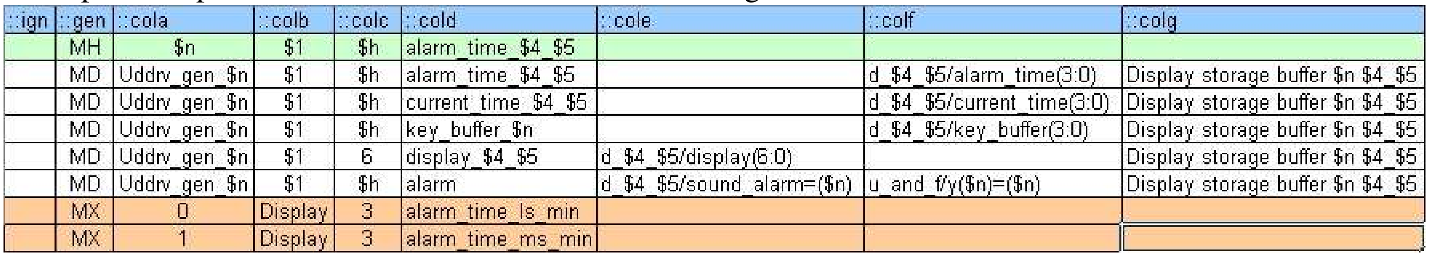
-delta -> auch fuer mif wird ein delta zum vorherigen Mif geschrieben (MIXCFG report.delta 1)
MIXCFG report.portlist.name 'name' => '',
# Define report file name, if empty take name of main module
# INST := name of last instance + _portlist.mif
# ENTY := name of last entity + _portlist.mif –> die Ausgabedatei 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

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.

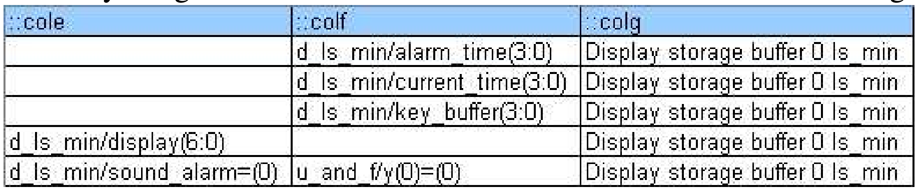
A simple example will illustrate the connection macro usage.



::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 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 assignd *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.).

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 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:

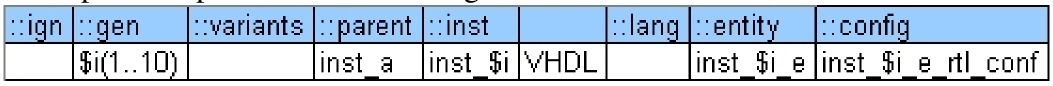
- Constructor generators: `$i(N..M)`
- Match generator: `/match_expression/`
- Bounded match generator: `$i(N..M),/match_expression/`

`match_expression` is a perl(1) regular expressions, 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.

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

`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 parantheses are used to set the variables `$1`, `$2`, ... as you would expect this in perl regular expressions. See the perl regular expression man page perlre(1) for more details. This variables can be used in the other cells. Simple arithmetic is possible, e.g. `$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, otherwise the current instance definition gets overloaded.

Example:

see "macro.xls"

8.1.3 Bounded match generator

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

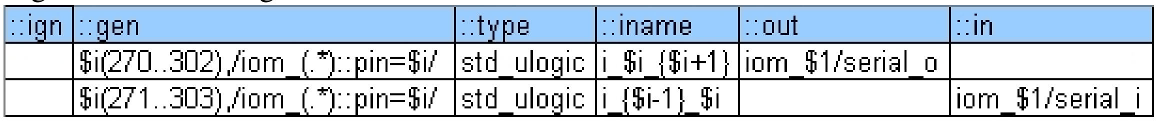
Example:

see "macro.xls"

8.1.4 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 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 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 generator will be executed for each signal bit, the variable `$s` will provide the bit number.

8.1.6 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.

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.

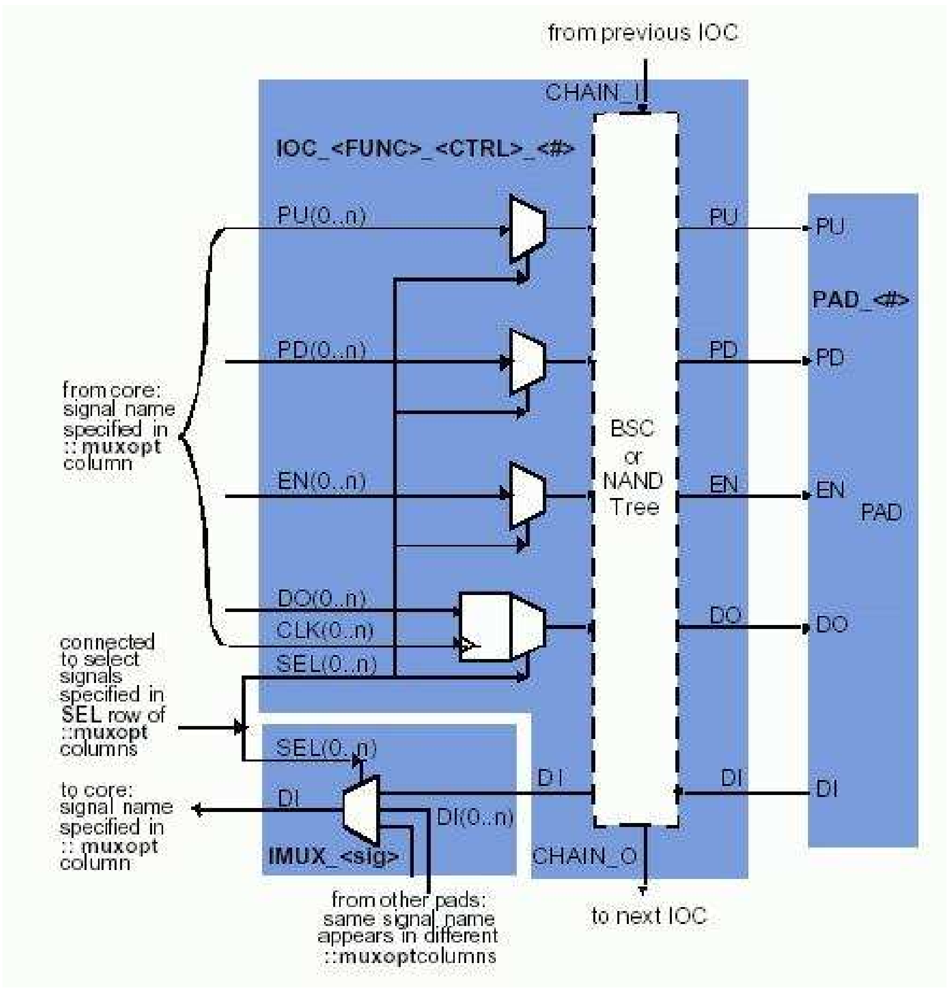


Image 3: IO Cell and Pad Layout

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

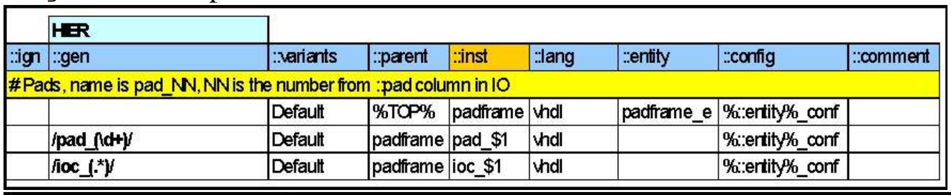
All other rows make up the connection matrix. For each row an io cell is instantiated. By default this io cell's name is composed by the type listed in the `::type` field with the `::pad` number attached, separated by a _.
Secondly a pad cell is instantiated. By default this pad cell's name is composed 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%
```

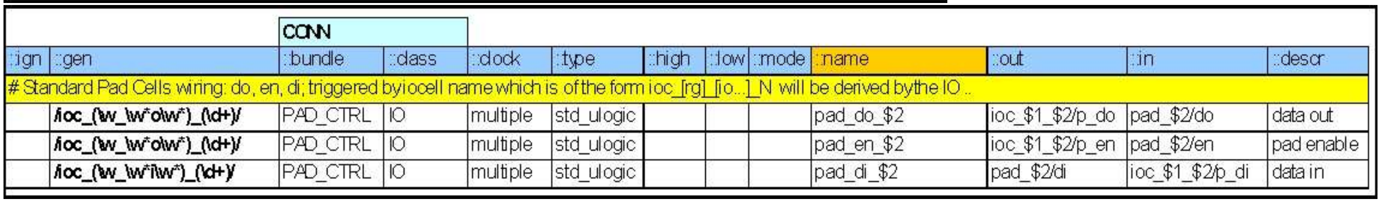
You can leave unused `::muxopt` columns empty. Then MIX will reduce the number of select lines accordingly.

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 Register-Master Sheet

The Register-master sheet is one more category of input specification. It describes configuration register domains of devices. 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.

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

10.1 Register-master Column Header

Column name	Description	Default value	Req.	Example
::ign	Ignore line		opt.	# comm.
::type	Register type; not used by MIX	I2C	man.	I2C
::dev	Device name for board-level purposes		opt.	FRCA0
::width	Width of register in bits	32	man.	32
::sub	Address offset of register in domain	0x0	man.	0x20
::addr	Not used by MIX	0x0	opt.	0x1234
::interface	Domain name		man.	sci_fe1
::block	Function block name; not used by MIX		opt.	fe_ycdet
::inst	Register name		man.	IP_CTRL1
::dir	Direction; access type of field	RW	opt.	RW
::auto	Not used by MIX		opt.	
::spec	Additional field access attribute		opt.	W1C
::update	Not used by MIX	N	opt.	N
::sync	Name of update signal for shadowed fields		opt.	upd_p12
::clock	Clock domain of field		opt.	clk_f20
::reset	Reset signal of field		opt.	asresc_n
::b	Bit n of field (as <field_name>.<n>)		man.	FMSYN.2
::init	Reset value of field	0	opt.	0x0
::rec	Recommended value of field	0	opt.	0
::range	Value range of field	0..1	opt.	0..15
::view	Include in documentation or not	Y	opt.	Y
::vi2c	Register type; not used by MIX		opt.	GI2C_32Bit-Registger
::name	Name of field (like in ::b)		opt.	ctrl1
::definition	Definition of field instance		opt.	ctrl1
::fgroup				
::comment	Comment field for reports		opt.	This is a comment

10.2 Register Views Global Configuration

Key	Value	Default	Description
i2c.xls	<name>		Sheet name in register-master xls/sxc file
i2c.regmas_type	<VGCA|FRCH>	VGCA	to accommodate different styles of register-master xls/sxc sheets
reg_shell.type	<view>	HDL-vgch-rs	selects view to be generated
reg_shell.bus_clock	<name>		name of config bus clock for domain
reg_shell.bus_reset	<name>		name of config bus reset for domain
reg_shell.addrwidth	<value>		width in bits of config bus address signal
reg_shell.datawidth	<value>		width in bits of config bus data signal (same as register width)
reg_shell.use_reg_name_as_prefix	<0|1>	0	If 1, prefix field names with register names

10.3 HDL Register-Shells

View "HDL-vgch-rs": This generates a Verilog OCP bus register-shell for the register domain. The specification for this can be found in `galaxy\\PROJECTS\MIC-32\MIC32C\ProjectDocumenta`

10.3.1 Configuration

Key	**Value**	**Default**	**Description**
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	**0..<n>**	**0**	**Parameter that controls the read-pipelining; If 0, no read-pipelining will be inserted; n<addrwidth**
reg_shell.read_multicycle	**0..<n>**	**0**	**can be one of [0,1,2,..] to insert delays for read-acknowledge**
reg_shell.exclude_regs	**<name>[, <name>]***		**Comma-seperated list of register names to be excluded from generation**
reg_shell.exclude_fields	**<name>[, <name>]***		**Comma-seperated list of field names to be excluded from generation**

10.4 VR_AD Macro

View "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.

10.5 STL Stimuli Files

View "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.

10.5.1 Configuration

Key	Value	Default	Description
reg_shell.stl.initial_idle	<value>	100	number of wait cycles before first transaction
reg_shell.stl.exclude_regs	<name>[, <name>]*		Comma-seperated list of register names to be excluded from generation
reg_shell.stl.use_base_addr	<0|1>	0	not yet implemented

10.6 MIF Report Files

The report can be created with the -report reglist command line option.

10.7 Register-Master Sheet Example

An example sheet can be found in `galaxy\\PROJECTS\VGCH\ProjectDocumentation\Registerma`

11 Alarm clock example

12 Alarm clock example

13 MIX converter man page

13.1 Synopsis

13.2 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".

`- domain DOMAIN`
Restrict register-master view to domain <DOMAIN>

-report <portlist|reglist>
Write portlist and/or register master reports in MIF format.
# 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

13.3 Runtime options and configuration

Runtime configuration is controlled by (increasing precedence):

- built-in default values

- `mix.cfg` files, if found in `$HOME`, `$PROJECT` and/or in the current directory. Format is: `MIXCFG` *`name.of.conf value`*
- CONF sheet found in input xls files
- command line switch: `-conf name.of.conf=value`
- 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()`)

13.4 Misc features

13.4.1 -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

13.4.2 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)` uses old sheet format, auto applies auto-format. Others do nothing special. Default: `previous`
'`intermediate.strip`' if set, all old and diff sheets will get removed
'`intermediate.ext`' (unused)
'`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.

13.4.3 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.

14 Alarm clock example

14.1 Core logic

14.2 IO logic

15 Other examples

16 Helper Tools

16.1 Converting between MS-Win and UNIX

Some EDA tools react stange, if they detect MS-Win style end-of-line. To convert to UNIX end-of-lines use:

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

or (in MIX's Base directory):

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

for Dos to Unix conversion and

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

for Unix to Dos conversion.

16.2 Convert ExCEL to SXC and CVS

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 cvs 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 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 sinlge 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\" 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 contentin 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.

17 Known Bugs and limitations

17.1 Missing perl modules Win32:

05/10/21 09:45:05 FATAL: eval use Win32 failed: Can't locate Win32/OLE/Const.pm
That means, you do not have all required modules installed.
Fix:

For cygwin:
S:\Cygwin\setup.exe
-> install from internet
-> C:\cygwin als Root
-> C:\cygwin als Local
-> use proxy (mucwebproxy:8082)
-> choose the german ftp server
now the cygwin installer will check the installed and available packages.

View by categories, search for perl and select
: perl-libwin32
: perl-WIN32-GUI
-> choose next.
Feel free to ignore messages about having a non current cygwin version installed.
Alternatively you can choose to write out intermediate data in csv or sxc format.

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)

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
wig: this section needs to be done!

19.5 Used Software

- Perl (from www.activestate.com)
- Vim
- WinCVS

19.6 Release Notes

19.6.1 20060411

released mix/1.5, see doc/release20060411.txt

19.6.2 20060314

development release, updated logging, see doc/release20060314.txt

19.6.3 20060119

released mix/1.4, see doc/release_20060119.txt

19.6.4 20051019

see doc/release_20051019.txt

19.6.5 20030716

see doc\release_20030716.txt

19.6.6 20030709

see doc\release_20030709.txt

19.6.7 20030605

See doc\release_20030605.txt