

MIX - Micronas Interconnect Spec expander

Userreference

Author: Wilfried Gäsheimer

\$Date: 2005/04/13 10:20:10 \$

Version: \$Id: MIX_Userreference.lyx,v 1.6 2005/01/26 14:01:59 wig Exp

1 Introduction

MIX is part of the registmaster tools. The registmaster 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

1	Introduction	1
2	Management Summary	1
3	Table of Contents	1
4	Installation	3
5	Getting Started	4
5.1	A simple example	4
5.2	Mix it!	5
5.2.1	You get what you typed	6

6	Details	8
6.1	Initialization with -init	8
6.2	Common worksheet properties	8
6.3	Special Spreadsheet-format properties	9
6.4	HIER sheet properties	9
6.5	HIER columns details	9
6.5.1	::use Add Project Specific Libraries	10
6.6	Special HIER sheet properties	11
6.7	CONN sheet details	11
6.8	typecast	11
6.9	Automatic Generation of Toplevel Ports	11
6.10	CONN columns details	11
6.11	Special CONN sheet signals	11
6.11.1	%OPEN% aka. open	12
6.11.2	Constants	12
6.11.3	Generics and Parameters	14
6.12	Simple text macros	15
6.13	Predefined and user macros	15
7	CONN sheet macros	16
8	Generator statements	17
8.1	Hierachy Generator Operators	17
8.1.1	Constructor Generator statement	17
8.1.2	Match generator	17
8.1.3	Bounded match generator	17
8.1.4	Advanced features of the match generators	18
8.1.5	Additional Information	18
9	IO sheet	18
9.1	IO sheet column header	20
9.2	IO sheet %SEL% rows	20
9.3	IO sheet pad rows	20
9.4	IO cell and Pad connections	21
9.5	IO Parser global configuration	21
10	I²C sheet	21
10.1	I ² C Registerblock	22
10.2	I ² C sheet column header	22
10.3	I ² C Parser global configuration	24
10.4	I ² C Sheet example	24
11	VI2C sheet	24
12	Alarm clock example	24
13	Alarm clock example	24

14 MIX converter man page	24
14.1 Synopsis	24
14.2 Command line switches	24
14.3 Runtime options and configuration	25
14.4 Misc features	26
14.4.1 -delta mode	26
14.4.2 Intermediate Excel Sheet	26
14.4.3 Output Redirection	26
15 Alarm clock example	27
15.1 Core logic	27
15.2 IO logic	27
16 Other examples	27
17 Helper Tools	27
17.1 Converting between MS-Win and UNIX	27
17.2 Convert ExCEL to SXC and CVS	27
18 Known Bugs and limitations	28
19 Issue tracking: CADNET - Issue tracking - CAD Software - MIX	28
20 Links	28
20.1 MIX paper and documentation	28
20.2 Micronas internal	28
20.3 Others	28
20.4 Resources	28
20.5 Used Software	28
20.6 Release Notes	28
20.6.1 20030716	28
20.6.2 20030709	28
20.6.3 20030605	29

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.*.msi. For UNIX (Solaris/Linux) no user installation is required. Everything needed is installed in /tools/mix. Start MIX from the network drive like described in the examples below. That will also make sure, that you are using the most recent release automatically. After the Perl installation, you can run MIX immediately. Open a command shell on your desktop workstation:

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).

5 Getting Started

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**) 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*.

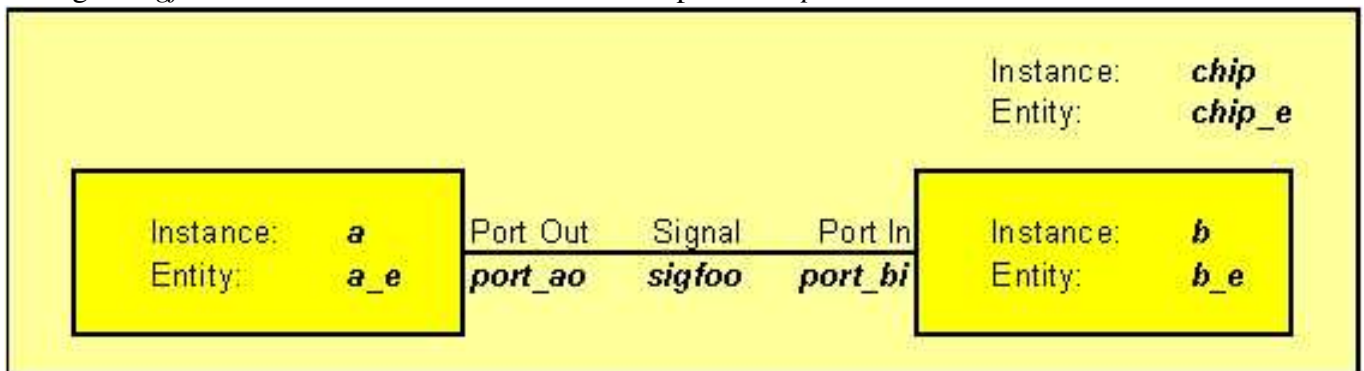


Image 1: mix_simple.xls

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:

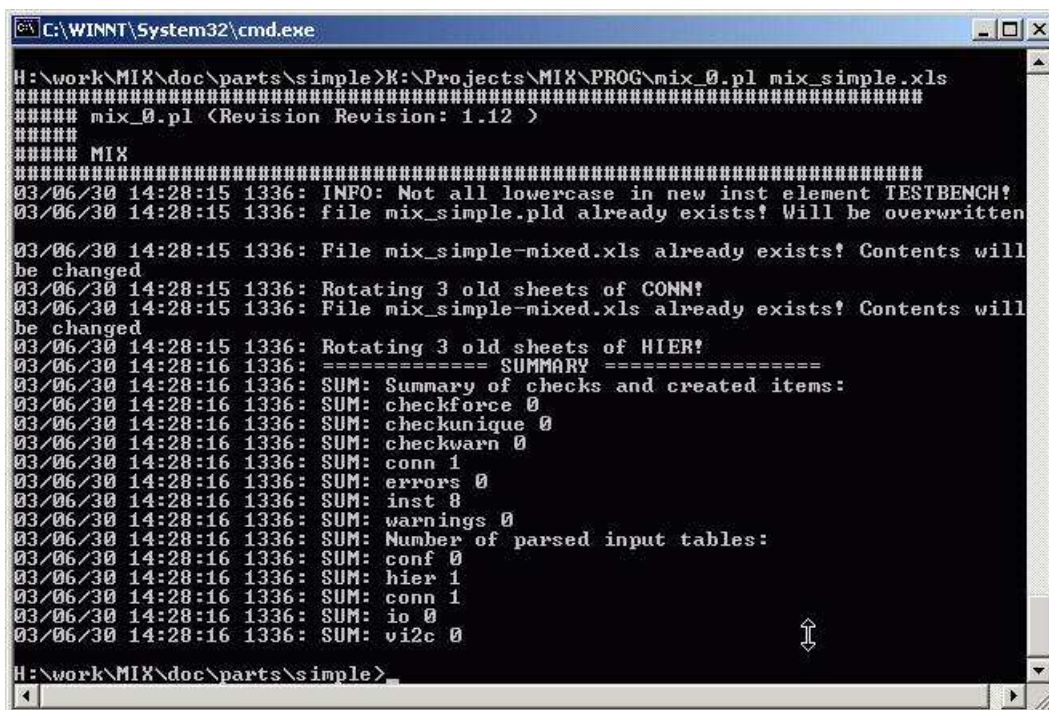
```
$ K:\Projects\MIX\PROG\mix_0.pl foo.xls
```

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_simple.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.



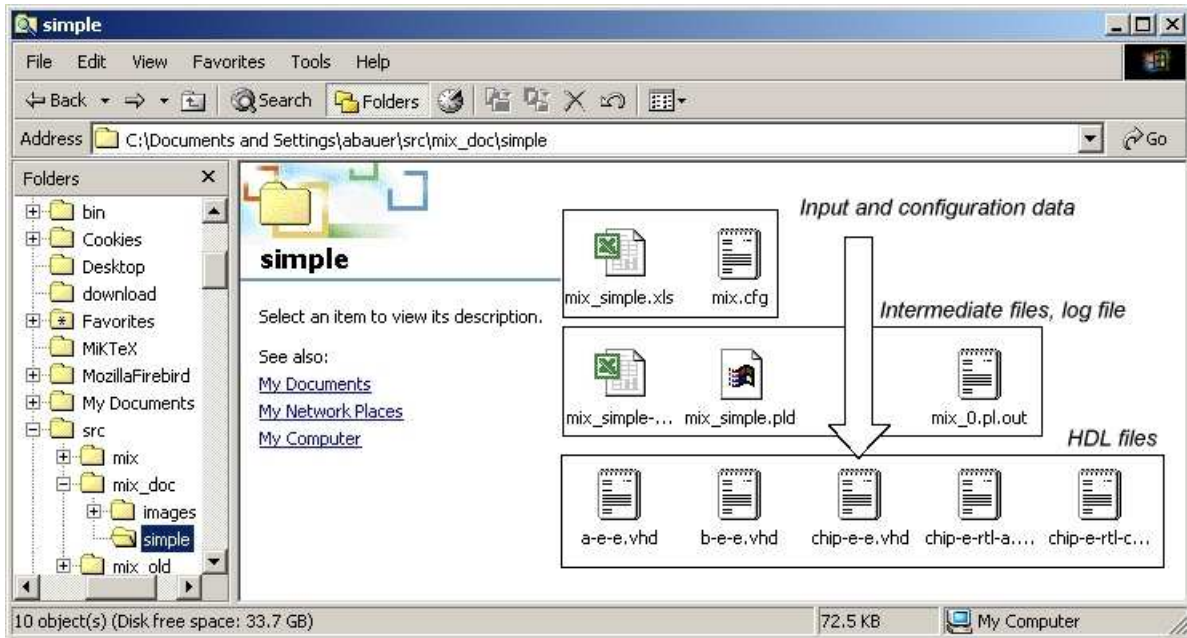
```
C:\WINNT\System32\cmd.exe
H:\work\MIX\doc\parts\simple>K:\Projects\MIX\PROG\mix_0.pl mix_simple.xls
#####
##### mix_0.pl <Revision Revision: 1.12 >
#####
##### MIX
#####
03/06/30 14:28:15 1336: INFO: Not all lowercase in new inst element TESTBENCH!
03/06/30 14:28:15 1336: file mix_simple.pld already 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.

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 “-”



5.2.1 You get what you typed

MIX generates various HDL files defined by the input data. If you select VHDL (also the default language) as output description for a hierarchy element, each element results in an appropriate entity, architecture and configuration description. By default MIX writes one file for all entities, one for all architecture and one for all configuration descriptions. Those file names are derived from the last excel input file name by stripping of the .xls extension and attaching a -e.vhd, -a.vhd and -c.vhd.

Here the working directory of the simple example contains a file mix.cfg, which is the convenient storage for MIX run-time configuration options. The lines "MIXCFG outarch ARCH", "MIXCFG outenty ENTY", and "MIXCFG outconf CONF" switch MIX outputs to separate files for each entity, architecture and configuration. In this case the file names are defined by the element name:

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.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 whether 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.

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

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 keywords `::debug`, `::hierarchy`, `::skip` and `::default` are used. Please do not use them. Apart from that you are free to add columns of your own. User defined columns are usable in `\%::NAME\%` macro expansion and are listed in the intermediate design data output.

Only the `::inst` column has to contain a value in each row (which could be a `\%::NAME\%` macro, though). All other columns will receive more or less reasonable default values in case they are left empty.

6.5 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_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`

6.6 Special HIER sheet properties

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

6.7 CONN sheet details

The design connections are defined in the CONN sheet. The primary key of this sheet is the signal-name 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:

<code>::name</code>	<code>signal_a</code>
<code>::type</code>	<code>type_a</code>
<code>::in</code>	<code>inst_a/port_a'typefunc</code>
<code>::out</code>	<code>inst_b/port_b'typefunc</code>

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\%`

6.11.1 %OPEN% aka. open

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

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

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

XXX write this in spreadsheet XXX

You could also force the ::in pins to high or low, instead, e.g. use:

\%HIGH\%, \%HIGH_BUS\%, \%LOW or \%LOW_BUS\%. Or a constant.

6.11.2 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/hdl_basehex_p
		Formats			integer					2#1010_1010#	inst_aa/hdlbase2_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_p

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

```

...
-- 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 nix_const_10_c : string := "ein string"; -- __I_ConstNoconv
signal nix_const_10 : string;
constant nix_const_14_c : std_ulogic_vector(7 downto 0) := "11111111"; -- __I_ConvConstant: 16#FF#
signal nix_const_14 : std_ulogic_vector(7 downto 0);
constant nix_const_15_c : std_ulogic_vector(7 downto 0) := "00010001"; -- __I_ConvConstant: 16#11#
signal nix_const_15 : std_ulogic_vector(7 downto 0);
constant nix_const_16_c : std_ulogic_vector(10 downto 0) := "000111111111"; -- __I_ConvConstant: 16#FF#
signal nix_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;
nix_const_10 <= nix_const_10_c;
nix_const_14 <= nix_const_14_c;
nix_const_15 <= nix_const_15_c;
nix_const_16 <= nix_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 => nix_const_10,
std_u_11_vport => nix_const_15,
std_u_logic_vport => nix_const_14,
std_u_logic_vport_ext => nix_const_16,
);
-- End of Generated Instance Port Map for inst_aa

```

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:

The screenshot shows a text editor window titled "generic_chip_e-rtl-a.vhd + ...MIX\doc\parts\simple) - GVIM2". The menu bar includes "Datei", "Editieren", "Werkzeuge", "Syntax", "Puffer", "Ansicht", and "Hilfe". The toolbar contains various icons for file operations and editing. The main text area displays the following VHDL code:

```

--
-- 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
--

```

The status bar at the bottom right shows "20,3-17" and "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 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

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 variables 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.).

8 Generator statements

8.1 Hierachy Generator Operators

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

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

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

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

8.1.1 Constructor Generator statement

MIX takes the variable and the range 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:

<code>::ign</code>	<code>::gen</code>	<code>::variants</code>	<code>::parent</code>	<code>::inst</code>	<code>::lang</code>	<code>::entity</code>	<code>::config</code>
	<code>\$i(1..10)</code>		<code>inst_a</code>	<code>inst \$i</code>	VHDL	<code>inst \$i_e</code>	<code>inst \$i_e rtl conf</code>

will give ten instances from `inst_1` to `inst_10`, each being a submodule of `inst_a` with each having a entity `inst_1_e` to `inst_10_e`, dito. for the configuration. Simple arithmetic can be applied to derive values from the run time parameter `$i`.

Constructor's will yield new instances. An instance name has to be given in the `::inst` column.

8.1.2 Match generator

For each instance defined up to now, `match_expression` is evaluated. If this yields true, the line is executed. Parts of the expression in parantheses are used to set the variables `$1`, `$2`, ... accordingly. See the perl regular expression man page `perlre(1)` for more details. This variables can be used in the other cells. Simple arithmetic is possible, e.g. `$i + 1` or `$1 * 2`.

A match generator will yield new instances only if the name in the `::inst` column is set to a value different from the matching instance.

Example:

XXX t.b.d.

8.1.3 Bounded match generator

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

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

Example:

XXX t.b.d.

8.1.4 Advanced features of the match generators

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

E.g. the connection generator

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

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

8.1.5 Additional Information

Match generators work the same way on the CONN sheets. By default the match expression will match against all defined instances, not connections.

To make a match expression iterate over all connection, prefix the whole expression by the `CONN:` keyword.

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

9 IO sheet

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

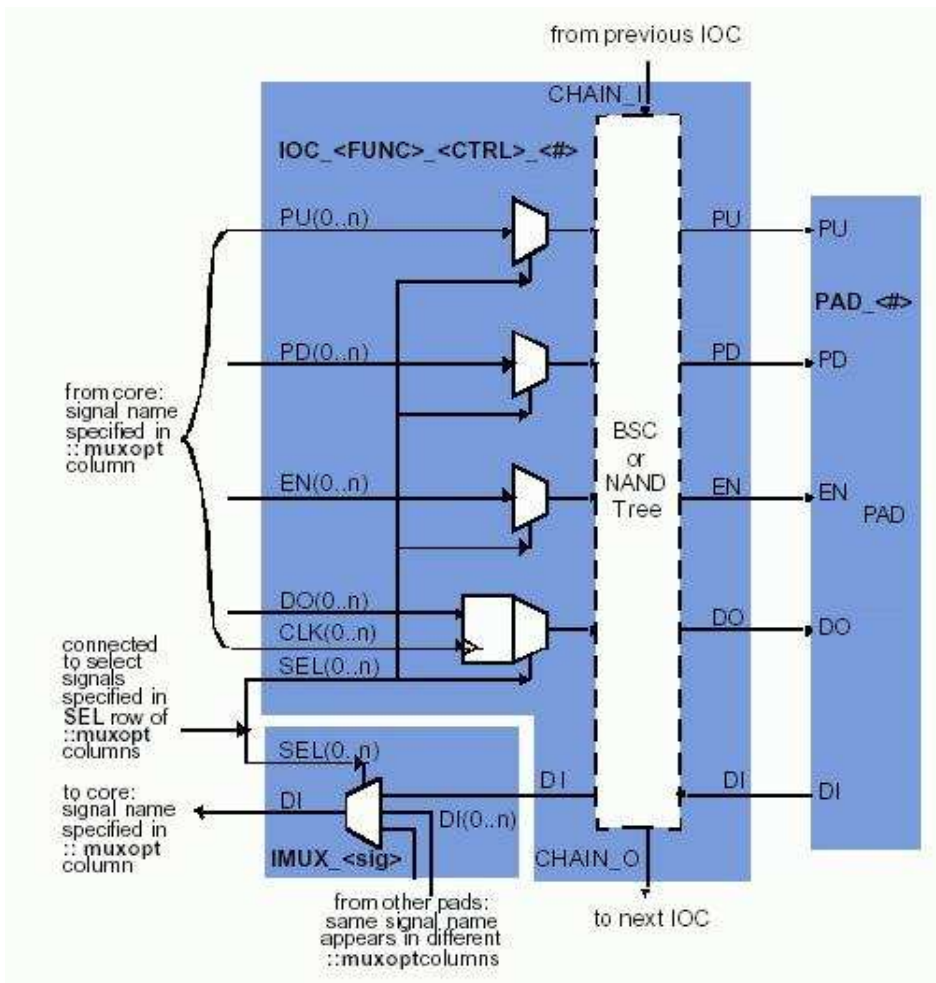


Image 3: IO Cell and Pad Layout

The IO sheet is a simple way to specify the connections of the IO cell to the design core logic. MIX will derive the connections of the DI(0..n), DO(0..n), PU(0..n), PD(0..n), EN(0..n) and the accompanying select lines. MIX will not add special purpose connections and the IO cell to Pad cell connections. This should be done by using the macro and generator statements in the CONN sheets. A simple example illustrates the various input fields and their usage.

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

9.1 IO sheet column header

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

- `::ign` A \# marks a comment line. Has to be first column. Optional.
- `::class` The class name is forwarded to the `::class` field of the CONN sheet. (opt.)
- `::ispin` Set to one if 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 be 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 entity 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.

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 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_sel(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 taken 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 explicitly, usually by means of `::gen` match operators.

Additional 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									
sign	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												
sign	gen	bundle	class	dock	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*olw*}_{d+}/	PAD_CTRL	IO	multiple	std_ulogic				pad_do_\$2	ioc_\$1_\$2/p_do	pad_\$2/do	data out
	/ioc_{w_w*olw*}_{d+}/	PAD_CTRL	IO	multiple	std_ulogic				pad_en_\$2	ioc_\$1_\$2/p_en	pad_\$2/en	pad enable
	/ioc_{w_w*ilw*}_{d+}/	PAD_CTRL	IO	multiple	std_ulogic				pad_di_\$2	pad_\$2/di	ioc_\$1_\$2/p_di	data in

9.5 IO Parser global configuration

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

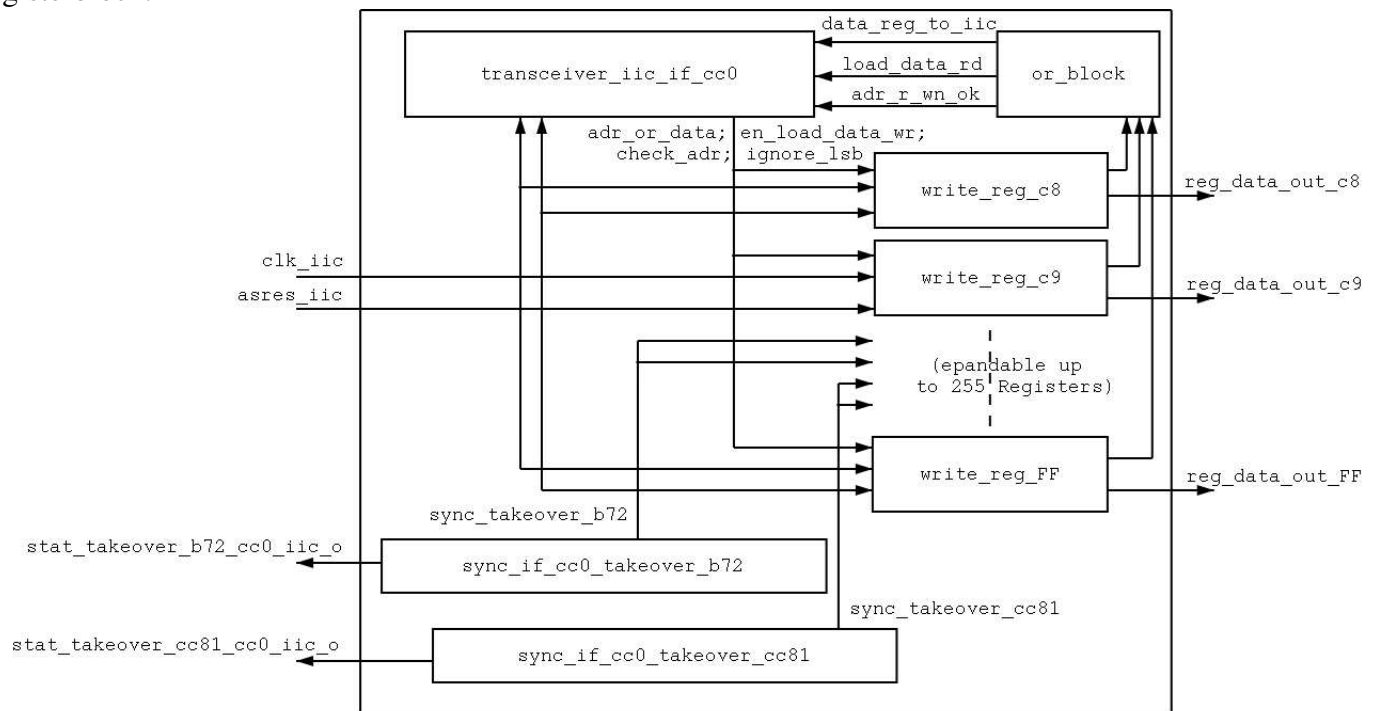
iocell.name	Rule determines naming of the io cell instances. Default "\%::iocell_\%::pad\%"
iocell.auto	If set to "bus", create busses as needed. Otherwise MIX relies on appropriate definition of busses by the CONN sheets.
iocell.bus	If iocell.auto is set to bus, MIX will attach the keyword listed here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

10 I²C sheet

The I²C sheet is one more category of input specification. It's content describes connections from other modules to I²C registerblocks. The hierarchical position of a registerblock can be defined in the HIER-sheet.

10.1 I²C Registerblock

The I²C module is split into two separate untis called I²C-kernel and I²C-registerblock. Settings made in a MIX-I²C-sheet only describe registerblock specific Informations. Here you can see a serial I²C-registerblock.



A I²C-registerblock consists of a transceiver, one or more synchronisation blocks and some registers (max. 255). The registers direction, synchronisation, in-/out-connections... can be specified in the I²C sheet. The **or_block** is a simple script generated block which connects incoming signals to a **or** and outgoing ones to the transceiver. This work has to be done by a MIX external script, because MIX is not thought for generating logics.

10.2 I²C sheet column header

The following section describes the meanings of identifiers used in a MIX I²C sheet. Default settings and mendatory/optional specification can be found in the table bellow.

Column name	Description	Default value	Req.	Example
::ign	Ignore line	<empty>	man.	# comm.
::variants	Variant selector	Default	opt.	Var1
::type	Register type	serial	man.	serial
::dev	Device name	n/a	man.	FRCA0
::sub	Sub address	VHDL	opt.	27
::interface	Domain name	W_NO_CONFIG	man.	cc0
::block	Block Name	<empty>	opt.	mc
::dir	Direction	<empty>	opt.	RW
::spec	Update Signal	NTO	opt.	takeover_b72
::clock	Clock Domain	<empty>	opt.	clkcc81
::reset	Reset Signal	<empty>	opt.	asresc_n
::busy	Busy Signal	<empty>	opt.	b81
::b	Bit n	<empty>	man.	FMSYN.2
::init	Reset Value	<empty>	opt.	0
::rec	Recommended Value	<empty>	opt.	0
::comment	Comment field	<empty>	opt.	This is a comment

::ign	every columns row, containing # or // is ignored by MIX
::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.
::ign	ignore this row if # is detected.
::variants	select a variant.
::type	register type; only serial registers implemented yet.
::dev	specify the device name
::sub	define registerblock subaddress
::interface	domain name
::block	registerblock Name. The registerblock name and its sub-address are building the instance-name.
::dir	defines a registers input/output direction, possible values are: R, W or RW
::spec	update signal
::clock	set a clock domain. Every I ² C register may receive it's own clock.
::reset	specify a Reset signal. This signal resets the I ² C registers.
::busy	busy signal
::b	specifies a signal which is connected to a registers pin. This is a multiple column which can be used to set different signals to the same register
::init	Set initializing value of register connection(s).
::rec	Set recommended value of a register connection(s).
::comment	all entries in these cells are threaten as comments. MIX will keep comments to put them later into VHDL/Verilog output.

10.3 I²C Parser global configuration

Also here some global configuration are available to tailor the I²C sheet parser behaviour:

i2c_cell.type	specifies the default register type (default: ser)
i2c_cell.\%IIC_SER_PREFIX	serPrefix for serial subregister entity (default: iic_ser_reg_)
i2c_cell.\%IIC_PAR_PREFIX	parPrefix for parallel subregister entity (default: iic_par_reg_)
i2c_cell.\%IIC_SYNC_PREFIX	syncPrefix for sync-block (default: sync_iic)

10.4 I²C Sheet example

In this section a example will demonstrate I²C sheet usage, by connecting the a_clk example to a registerblock. <t.b.d>

11 VI2C sheet

12 Alarm clock example

13 Alarm clock example

14 MIX converter man page

14.1 Synopsis

14.2 Command line switches

```
-out OUTPUTFILE.ext
    defines output filename and type
-outenty OUT-e.vhd|ENTY|COMB
    Write all entities into OUT-e.vhd.
    If argument is ENTY, each entity will be written
    into a file called entityname-e.vhd.. (The exact
    naming depends on changeable rules).
    If argument is COMB, entity, architecture and configuration
    will all be written into one file called entityname.vhd
-outarch OUT-rtl-a.vhd|ARCH|COMB
    See description of outenty option.
-outconf OUT-c.vhd|CONF|COMB
    See description of outenty option.
-combine
    write entity, architecture and configuration into one file for
    each entity. Shortcut for setting -out[entylarchlconf] to COMB
    individually.
-dir DIRECTORY
    write intermediate, internal and backend data into the given
    DIRECTORY. By default MIX writes to the current working directory
-top TOPCELL
```

use TOPCELL as top. Default is TESTBENCH or daughter of TESTBENCH.

-adump
dump internal data in ASCII format, too
(debugging, use with small data set).

-variant VAR1 Select VAR1 from the HIER worksheet.

-conf key.key.key=value
Overload \$EHkeykeykey with value or add a
new configuration variable.

-listconf
Print out all available/predefined configurations options

-delta
Output will be compared against previous runs.

-sheet SHEET=MATCH
SHEET can be one of "hier", "conn", "vi2c". MATCH is
a perl regular expression.

-strip
Remove old and diff sheets. These sheets are named with "O_" or "DIFF".

Add your options here

"Standard" options:

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

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

14.3 Runtime options and configuration

Runtime configuration is controlled by (increasing precedence):

- built-in default values
- `mix.cfg` files, if found 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. `$ENV{HOME}`, `$ENV{HOMEDRIVE}`, `$ENV{HOMEPATH}`, `$ENV{USERPROFILE}` or `C:\`
(only from the first matching location)
2. `$ENV{$PROJECT}`
3. `"."` (`cwd()`)

14.4 Misc features

14.4.1 -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. Messages are appended to `mix_0.pl.out`

If a new HDL-file needs to be created by the changes, the `-delta` mode is not be applied for that file, but it is generated as new. If you never have written a `FOO-mixed.xls` intermediate output, 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, then.

14.4.2 Intermediate Excel Sheet

Format of intermediate xls sheet will be kept as is as long as the number and order of columns is unchanged.

MIX saves three old versions of the generated **HIER** and **CONN** sheets. The worksheets names are rotated by a trailing `_` and number, e.g **HIER_0**, **HIER_1**,

14.4.3 Output Redirection

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

`-dir DIRECTORY`

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

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

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

15 Alarm clock example

15.1 Core logic

15.2 IO logic

16 Other examples

17 Helper Tools

17.1 Converting between MS-Win and UNIX

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

```
$ module load freeware; recode "ps..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.
```

17.2 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 separated by "=:==>" followed by the ExCEL sheet name. Even the first sheet will be marked by such a line.

xls2csv.pl options:

- csv create csv formatted output
- sxc create OpenOffice (sxc) formatted output. By default both format will be created.
- sep X take X as field delimiter. Default is ";".
- [no]head do [not] separate sheets with a special marker line (=:==> SHEETNAME). On by default.
- noquote do not quote the csv cells. Quotes are on by default.
- quote X use X as quoting character. Default are double-quotes (").
- sheet RE select only sheets which names match the regular expression RE .

Please remove the created files before updating.

For MS-Win workstations, use

K:\Projects\MIX\PROG\xls2csv.pl

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

18 Known Bugs and limitations

19 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.

20 Links

20.1 MIX paper and documentation

EDP_2003_final_030331.pdf

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

20.2 Micronas internal

Micronas HDL coding guidelines.pdf

20.3 Others

20.4 Resources

Downloads for:

- IO Examples
 - Standard bus examples
 - NAND Tree
 - BS example
 - miscellaneous useful macros
- wig: this section needs to be done!*

20.5 Used Software

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

20.6 Release Notes

20.6.1 20030716

see doc\release_20030716.txt

20.6.2 20030709

see doc\release_20030709.txt

20.6.3 20030605

See doc\release_20030605.txt