Register Wizard 1.0.0

User Manual

Rev. A

Document Change History

Revision	Date	Change	Author(s)
A	05-02-2016	Document created	Andreas Tornes

Acronyms and Abbreviations

Acronym / Abbreviation	Definition
BFM	Bus Functional Model
CPU	Central Processing Unit
HDL	Hardware Description Language
IRQC	Interrupt Request Controller
JSON	JavaScript Object Notation
lsb	least significat bit
MDF	Model Description File
msb	most significat bit
PIF	Processor InterFace
RTL	Register Transfer Language
SBI	Simple Bus Interface, see section 6.1 for details
TB	Test Bench
UART	Universal Asynchronous Receiver/Transmitter
VHDL	VHSIC Hardware Description Language
VHSIC	Very-High-Speed Integrated Circuits

About Register Wizard

With Register Wizard you can auto generate a number of files from a single source, used in both FPGA and software development, as shown in Figure 1. The single source is a description of a module and its interfaces, registers and individual fields. The description is written as a Model Description File (MDF) described in section 3.2.

Register Wizard can generate HDL files, software header files, test benches, ModelSim scripts and documentation.

Register Wizard allows you to customize file generation and documentation layout and content through a range of parameters and templates.

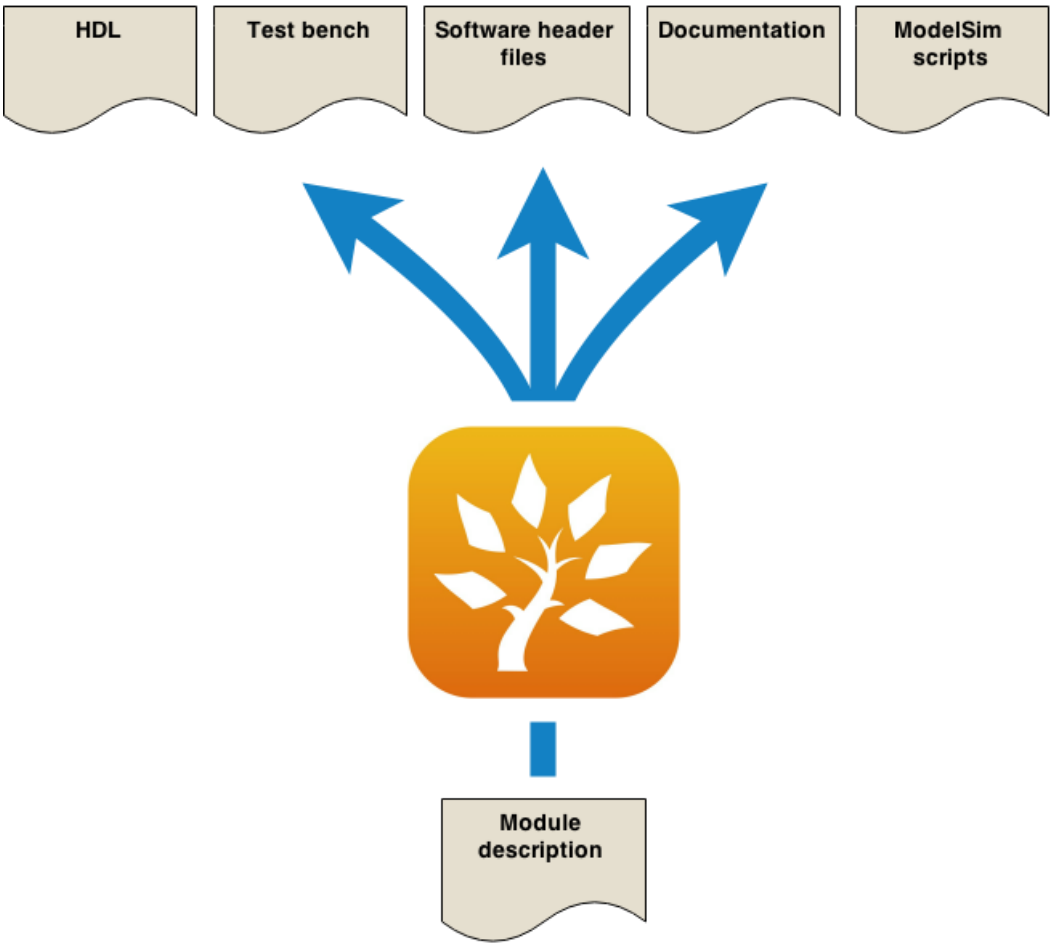

Figure 1 Files generated by Register Wizard

About This User Manual

This User Manual applies to Register Wizard 1.0.0 and consists of the parts shown in Table 1. Text deserving special attention is preceded with symbols as listed in Table 2.

For latest version of this User Manual visit http://www.bitvis.no/products.

Table 1 Main parts of the User Manual

Part	Content
Quick Start Guide	Installation and basic usage
	Troubleshooting
Tutorial and Examples	Step by step tutorial
Technical Reference	Command line options
	Model Description File
	Configuration
	Templates
	Generated output
License	Register Wizard licensing terms
Change Log	Register Wizard change history

Table 2 Symbols for special attention

Symbol	Meaning
	Information worth attention
	A warning or other important message
	Information for Linux users
	Information for Windows users

Register Wizard License

The following is a summary, not a substitute for, of the Register Wizard v1.0.0 license. For complete license text see section 4.

Register Wizard is released under the **Creative Commons Attribution-NoDerivatives 4.0 International Public License.**

You may copy and redistribute the material in any medium or format, for any purpose, even commercially. The licensor cannot revoke these freedoms as long as you follow the license terms.

Attribution

You must give **appropriate credit**, provide a link to the license, and **indicate if changes were made**. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

No Derivatives

If you **remix, transform, or build upon** the material, you may not distribute the modified material.

Table of Contents

1 Quick Start Guide

This Quick Start Guide will briefly introduce you to Register Wizard and guide you through the installation process. For detailed technical reference see section 3.

1.1 Installation

1.1.1 System Requirements

You can run Register Wizard on Windows 7 and later or any Linux operating system.

Register Wizard requires Java Runtime Environment 1.7 or higher.

Optional: If you want to run a test bench or compile the HDL source using the generated ModelSim scripts, you will have to install ModelSim.

1.1.2 Bundled Libraries

Register Wizard is bundled with libraries listed in Table 3.

Table 3 Libraries bundled with Register Wizard.

Library	Description	Location
UVVM Utility Library	UVVM Utility Library is an open source VHDL test bench (TB) infrastructure library for verification of FPGA and ASIC. Used by Register Wizard when running generated test benches. For more information on UVVM Utility Library and latest release please visit http://www.bitvis.no/products/	<install_dir>/uvvm_util
Bitvis Simple Bus Interface	Bitvis Simple Bus Interface (SBI) Bus Functional Model (BFM). Used by Register Wizard when running generated test benches. For details se section 6.1.	<install_dir>/bitvis_vip_sbi

1.1.3 Installing Register Wizard

1. Install the required Java Runtime Environment as mentioned in section 1.1.1.
2. Download the Register Wizard installer from http://www.bitvis.no/products
3. Start the installer from file explorer or a terminal window. The installer will modify the PATH variable.

```
$ java -jar registerwizard-1.0.0-installer.jar
```

1.1.4 Uninstalling Register Wizard

To uninstall execute:

```
$ <install_dir>/Uninstaller/uninstall.bat
```

```
$ <install_dir>/Uninstaller/uninstall.sh
```

1.2 Running Register Wizard

To run Register Wizard open a terminal window and execute:

```
$ registerwizard.bat
```

```
$ registerwizard.sh
```

The above command will print basic usage with explanation of all available options.

For an overview of command line options supported by Register Wizard see section 3.1.

1.3 Support

For support or bug reports please send email to **support@bitvis.no**.

1.4 Troubleshooting

Table 4 lists known issues and the corresponding solutions.

Table 4 List of known issues and solutions.

	Problem	Reason and Solution
	Unable to run examples from installation directory.	**Reason** Register Wizard might have been installed by the Administrator to a directory where you do not have write access. If you have installed Register Wizard to Program Files under Windows 7 or later, the User Access Control mechanism of Windows will force these folders to be write protected. **Solution** Reinstall Register Wizard to another location or copy the example directory to another location where you have write permissions.
	Register Wizard not in path after installation	**Reason** The Installer adds the path to Register Wizard to the PATH environmental variable. The updated PATH is not visible for already open terminals. **Solution** Open new terminal. Restart the computer if the above does not help.
	Register Wizard not in path after installation	**Reason** The Installer adds the path to Register Wizard to the PATH environmental variable. The updated PATH is not visible for already open terminals. **Solution** Open new terminal or execute in an already open terminal: $source ~/.bashrc

2 Tutorial and Example

This tutorial will help you to get familiar with Register Wizard through step-by-step example. The example follow a workflow as described in section 0. We will take advantage of the full generation capabilities of Register Wizard to produce:

- VHDL source files
- VHDL test benches
- ModelSim scripts
- Documentation
- Software header files

2.1 Requirements

To be able to run the example you have to install Register Wizard as described in section 1.1. In addition, you have to install ModelSim to be able to execute test benches and ModelSim scripts generated by Register Wizard.

All files used in the examples are part of the Register Wizard installation. We advise you to create a project directory, in any desirable location, where you copy directories <install_dir>/regwiz/templates and <install_dir>/regwiz/examples.

2.2 Tutorial Workflow

Examples in this tutorial are based on a workflow consisting of four steps as show in Figure 2. Each step in the workflow is described below.

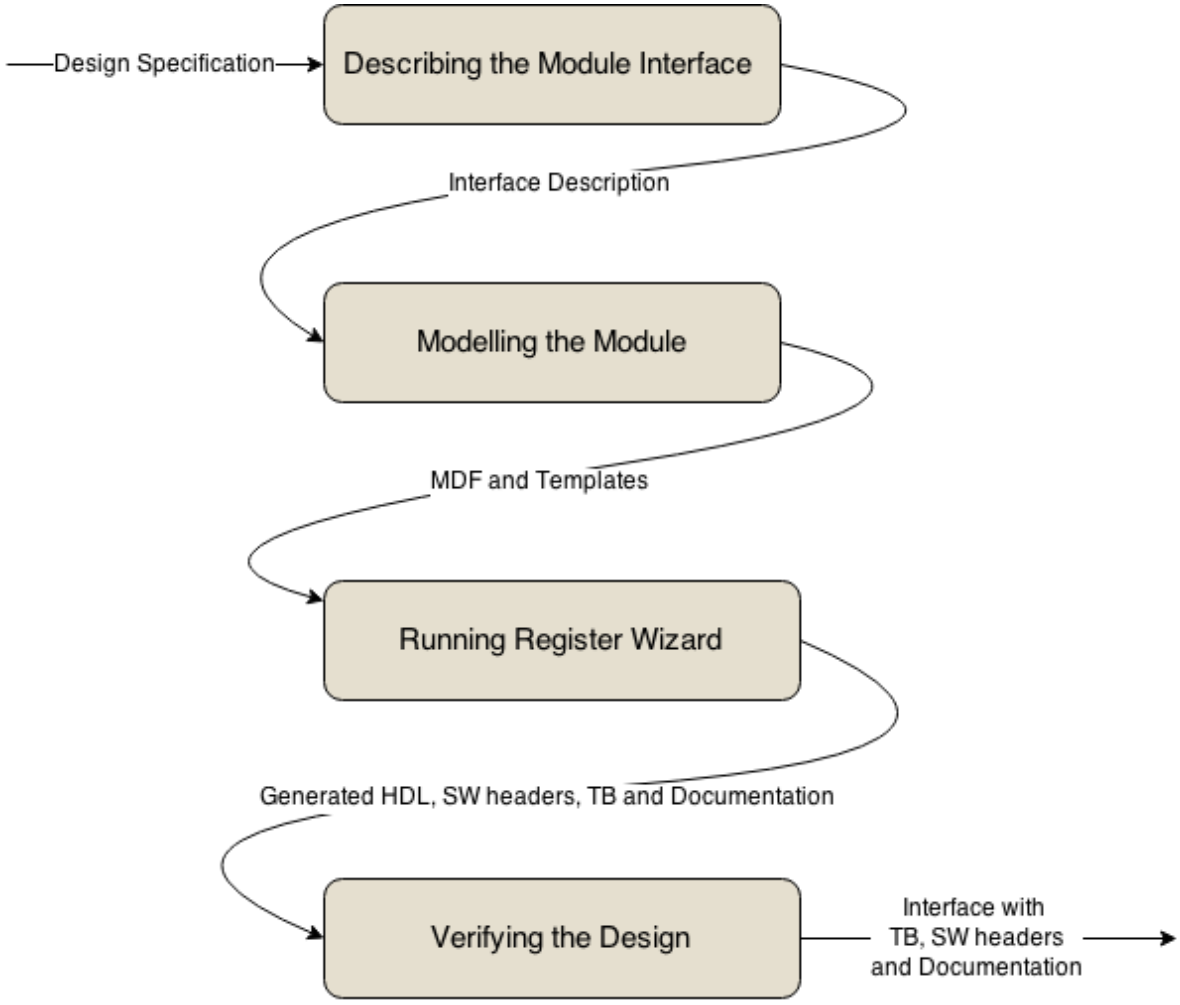

Figure 2 Workflow used in examples

2.3 Step 1: Describing the module interface

In this step we prepare information needed in order to model a module interface in a Model Description File (MDF). This can be done by listing all signals which should be accessible through a given interface. Signals which are not part of the module interface might not be defined at this stage, but if they are, they can be included and explicitly marked as not accessible. Table 5 is an example of such a list, while Table 6 defines all properties required for module modelling.

Table 5 Example of a list of signals for a given module

Signal name	Description	Accessible by SBI
IER	Interrupt Enable Register. Enables interrupts from that address to the CPU (given that IRQ2CPU_allowed is active).	Yes
ICR	Interrupt Clear Register. Clears the interrupt register at the given address.	Yes
irq_source	An interrupt source at that address. Not accessible by the PIF.	No

Table 6 Example of a list of module registers with properties

Register Name	Width	Access Type	Signal Type	Reset Value	Location	Interface
IER	32	Read/Write	std_logic_vector	0x0	Pif	SBI
ICR	32	Write-to-Trigger	std_logic_vector	0x0	Core	SBI

2.4 Step 2: Modelling the module

In this step we create an interface model of a given module, but before we start writing a module MDF we have to decide what we want Register Wizard to generate and where to put the results. We can specify output directories in a separate configuration file as described in section 3.3 or in a configuration object included in the module MDF. If no configuration is provided, Register Wizard will use default values as described in section 3.3 . We can specify custom templates for generated documentation and source file headers. Customization of documentation templates is described in section 3.4.2, while section 3.4.1 describes the file header templates.

Modelling of the module interface is done by populating a MDF with objects corresponding to the external interface type, registers, fields and the relevant properties for all objects. Section 3.2 describes a complete list of all objects and their properties.

2.5 Step 3: Running Register Wizard

In this step we run Register Wizard to process our MDF, with optional configuration and templates. We can control Register Wizard through a number of command line options described in section 3.1.

When Register Wizard finishes processing, we review and explain all generated files.

2.6 Step 4: Verifying the Design

In this step we compile the generated HDL files using ModelSim and run simulation to verify the design.

2.7 Example 1: IRQC

This example will show you how to model a simple Interrupt Request Controller (IRQC) module attached to a Simple Bus Interface (SBI). The following list is a summary of what is covered in this example:

- Modelling of module interface registers without fields
- Auto register addressing
- Modelling of registers in core with write enable
- Modelling of read/write registers
- Modelling of read-only registers
- Modelling of write-only registers
- Modelling of write-trigger registers
- Running Register Wizard
- Running simulation
- Review of the results

2.7.1 IRQC Module Design

The design for the IRQC module we will model in this example is shown in Figure 3. The IRQC module has two main parts, a Processor Interface (PIF) and a Core. The PIF connects the module to the SBI, while the Core holds module specific implementation.

The IRQC handles interrupts to a CPU from multiple sources, in our case 32 (n=32). The IRQC allows enabling and disabling interrupts for individual sources, in addition to enabling and disabling interrupts to the CPU altogether. An interrupt can be triggered by an external source through the *irq_source* signal, or a write to the *ITR* register.

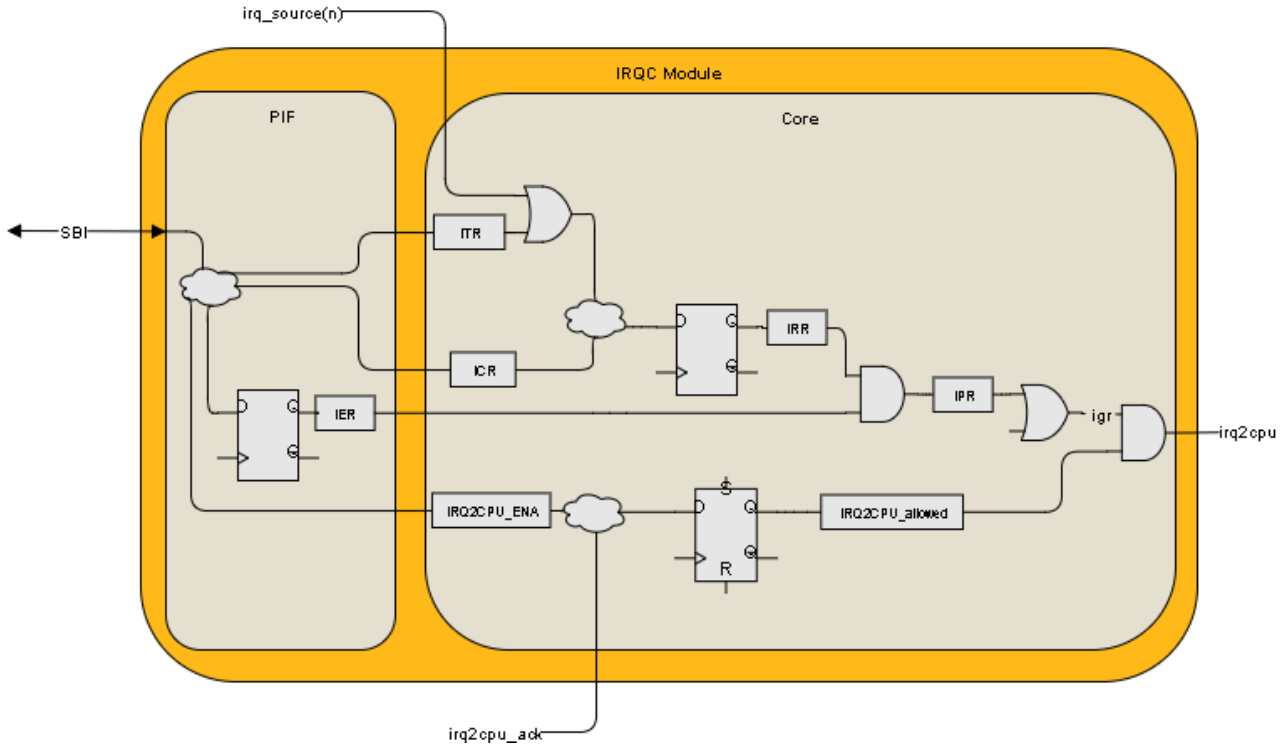

Figure 3 The IRQC module.

2.7.1.1 IRQC signals

Table 7 lists all signals for the IRQC module. Note that only signals accessible by the SBI are relevant for the IRQC model processed by Register Wizard. The remaining signals are part of the IRQC core implementation and are consequently not accessible through the SBI.

We want to manipulate the signals *ICR*, *IER*, *ITR* and *IRQ2CPU_ENA*. These signals need to be written to in order to trigger and clear interrupts through SBI, and for the interrupts to be registered at the CPU.

We want to read the signals *IPR*, IRR and *IRQ2CPU_allowed* to be able to observe the status of the IRQC.

The signals *igr*, *irq_source*, *irq2cpu* and *irq2cpu_ack* will not be accessed by the PIF. The reason is that they are part of the core functionality of the IRQC, and can be inferred by reading all other status registers.

For the signals accessible through the *SBI*, only the *IER* signal is not located in the PIF. Signals implemented in the IRQC core can be access through the SBI. A core signal can also be a dummy signal.

Table 7 Both external and internal signals for the IRQC module.

Signal name	Description	Remark	Accessible by SBI
IER	Interrupt Enable Register. Enables individual interrupts to the CPU (given that *IRQ2CPU_allowed* is active).	Resides in the PIF	Yes
ICR	Interrupt Clear Register. Clears the interrupt register for a given source.	Core to PIF	Yes
IPR	Interrupt Pending Register. Indicates that interrupts are enabled for a given source, and that and interrupt has been triggered.	Core to PIF	Yes
IRR	Interrupt Request Register. Holds an interrupt request for a given source until cleared.	Core to PIF	Yes
ITR	Interrupt Trigger Register. Trigger an interrupt for a given source through the SBI, as opposed to through the *irq_source*.	Core to PIF	Yes
IRQ2CPU_ENA	Enable signal for the IRQC. Activates *IRQ2CPU_allowed*.	Core to PIF	Yes
IRQ2CPU_allowed	Enables interrupts to the CPU for a given source where IER is set.	Core to PIF	Yes
igr	Interrupt Global Register. Indicates that an interrupt has been triggered for one of the sources.	Not accessible by the PIF	No
irq_source	An interrupt source at that address.	Not accessible by the PIF	No
irq2cpu	An interrupt signal from the IRQC to the CPU.	Not accessible by the PIF	No
irq2cpu_ack	Reset for *IRQ2CPU_allowed*. Acknowledge from the CPU that an interrupt has been detected.	Not accessible by the PIF	No

2.7.2 Running IRQC Example

2.7.2.1 Step 1: Describing the module interface

In this step we will map the properties of the registers that shall be accessed through the SBI. The only register that resides in the PIF is the *IER*. All of the other registers are in the IRQC core.

Table 8 shows the module interface register map for the IRQC. The width of 32 refers to the number of interrupt sources.

Write-to-Trigger means that a single-cycle pulse is generated from the PIF to the IRQC core when written to. Since these registers don't have any actual registers implemented, the location setting does not have any effect other than a generated dummy register.

Table 8 Register map for the IRQC interface

Register Name	Width	Access Type	Signal Type	Reset Value	Location	Interface
IER	32	Read/Write	std_logic_vector	0x0	PIF	SBI
ICR	32	Write-to-Trigger	std_logic_vector	0x0	Core	SBI
IPR	32	Read-Only	std_logic_vector	0x0	Core	SBI
IRR	32	Read-Only	std_logic_vector	0x0	Core	SBI
ITR	32	Write-to-Trigger	std_logic_vector	0x0	Core	SBI
IRQ2CPU_ENA	1	Write-Only	std_logic	0x0	Core w/ enable	SBI
IRQ2CPU_allowed	1	Read-Only	std_logic	0x0	Core	SBI

2.7.2.2 Step 2: Modelling the module

Figure 4 shows the structure of the MDF we are going to create in this step, and the relationship between the MDF and the block diagram of the design. The *irq_source*, *igr*, *irq2cpu* and *irq2cpu_ack* signals are not interfaced by the SBI, as already mentioned in section 2.7.1.1.

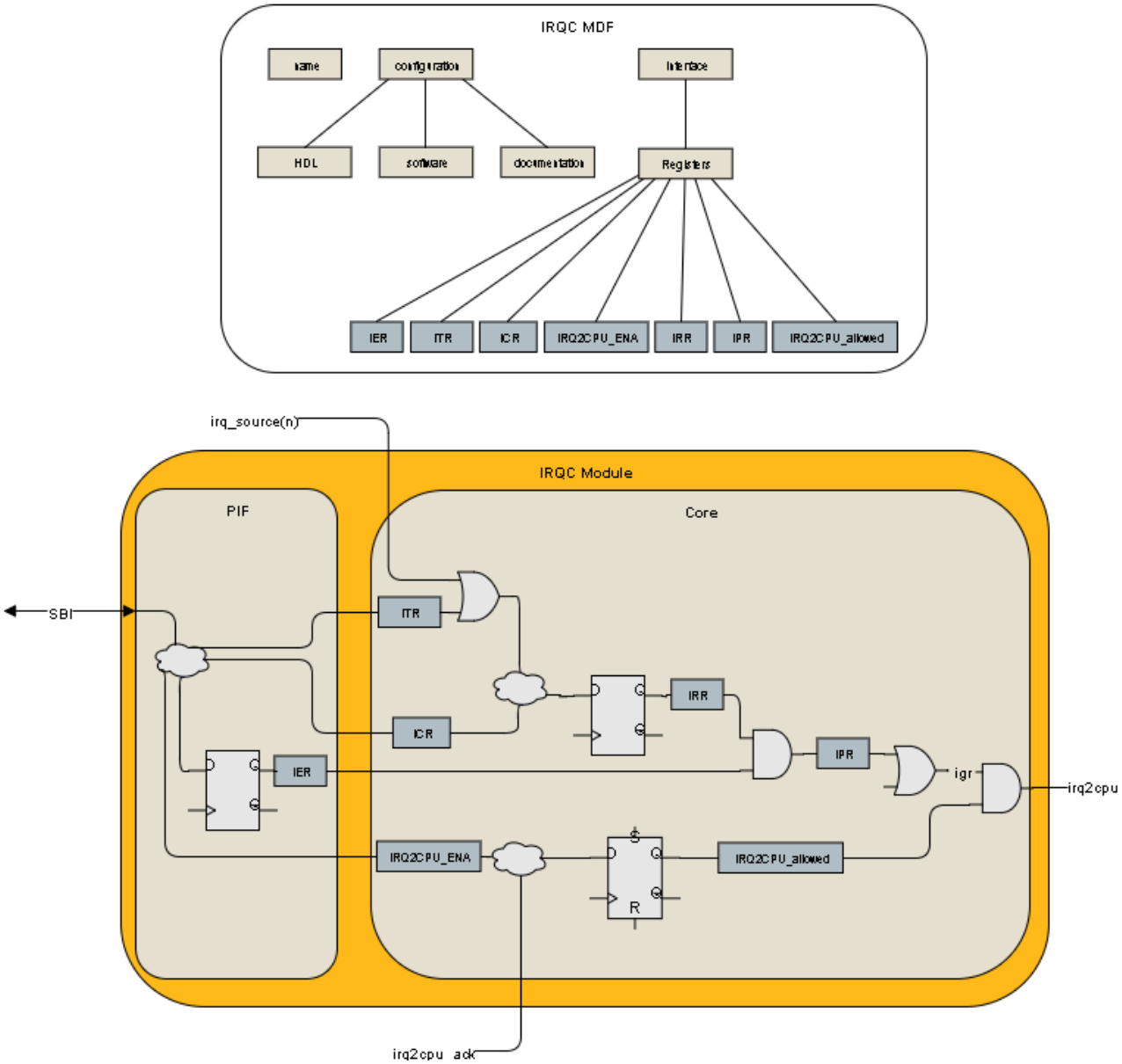

Figure 4 Relation between the model and the design for the IRQC module.

Creating IRQC MDF

Now we are ready to create the IRQC MDF.

> **The IRQC MDF example file can be found here:**
> **<install_dir>/regwiz/examples/irqc.regwiz.**

We start with the module object. The **module** object consists of three parameters: **name**, **configuration** and **interface** as shown in Listing 1.

name: this parameter sets the design name, *irqc* in our case.

configuration: this optional set of parameters is used for setting the paths to generated files, and for setting the paths to templates used for documentation and software header file generation.

interfaces: This required array of parameters defines the PIFs.

Listing 1 Module object

```
{
   "name": "irqc",
   "configuration": {},
   "interfaces": []
}
```

We continue with the **configuration** object shown in Listing 2. Each parameter of the **configuration** object is optional. This applies also to any sub-parameter. Default values are used for parameters not provided here.

Listing 2 Configuration object

```
"configuration": {
      "hdl": {}
      "software": {}
      "documentation": {}
}
```

hdl: this set of parameters sets the paths for all HDL-related files, i.e., the paths for the HDL source code, the HDL test bench and the ModelSim scripts. The names of the design and the test benches libraries are defined here, and finally the path to the file header template. The path prefix *<model_file>* means that the path is relative to the location of the module MDF. For more details on path prefix see section 3.3.2.1. The **hdl** object is shown in Listing 3. For more details see section 0.

Listing 3 Hdl object

```
"hdl" : {
      "rtlPath" : "<model_file>/irqc/src",
      "simPath" : "<model_file>/irqc/sim",
      "tbPath"  : "<model_file>/irqc/tb",
      "rtlLibrary" : "bitvis_irqc",
      "tbLibrary"  : "bitvis_irqc_tb",
      "headerTemplate" : "<model_file>/../templates/headers/header.txt"
      }
```

software: this parameter sets the path to the generated software header files and is shown in Listing 4.

Listing 4 Software object

```
"software" : {
      "path" : "<model_file>/irqc/sw"
}
```

documentation: this set of parameters sets the path to the generated documentation, and the path to documentation templates. The **documentation** object is shown in Listing 5. All paths are relative to the MDF file (*model_file*).

Listing 5 Documentation object

```
"documentation" : {
      "path" : "<model_file>/irqc/doc",
      "moduleTemplate" : [
            "<model_file>/../templates/docx/moduleregistermap-template.docx",
            "<model_file>/../templates/docx/registerdescription-template.docx"
      ]
}
```

interfaces: for the IRQC we only need one PIF, the SBI. The interface object is shown in Listing 6. The **name** parameter is set to *sbi*, but can be anything. The **type** parameter must be set to *SBI*. The **registers** parameter is an array of all of the registers we want to implement, and will be explained below.

Listing 6 Interface object

```
"interfaces": [{
      "name": "sbi",
      "type": "SBI",
      "description": [
      "Control/Status register interface."
      ],
      "registers": []
}]
```

registers: each register consist of a set of parameters. We have specified most of these parameters earlier in Table 8, so we just fill these in. However, there are a few parameters we have not considered yet. The **address** parameter is set to *auto* for all registers in the IRQC. This will automatically increment address of every register added based on the address of the previous register. First register will be at address 0. The **summary** and **description** parameters are set for documentation purposes. For detailed description of each parameter see section 3.2.1.2.

Listing 7 shows the *IER* register, while the *ICR* register is shown in Listing 8.

Listing 7 Register object for the IER register

```
"registers": [{
      "name": "IER",
      "address": "auto",
      "access": "RW",
      "signal": "std_logic_vector",
      "reset": "0x0",
      "width": 32,
      "location": "pif",
      "summary": [
            "Interrupt Enable Register."
      ],
      "description": [
            "Interrupt Enable Register"
      ]}
      ...
]
```

Listing 8 Register object for the ICR register

```
"registers": [
      ...
      {
      "name": "ICR",
      "address": "auto",
      "access": "WO",
      "signal": "std_logic_vector",
      "reset": "0x0",
      "width": 32,
      "location": "core",
      "summary": [
            "Interrupt Clear Register."
      ],
      "description": [
            "Interrupt Clear Register"
      ]},
      ...
]
```

2.7.2.3 Step 3: Running Register Wizard

We have now described the IRQC interface in step1, and modelled it as a MDF in step 2. Now it's time to fire up Register Wizard.

We start Register Wizard with this command:

```
$ registerwizard.sh -module irqc.regwiz -initial
```

```
$ registerwizard.bat -module irqc.regwiz -initial
```

The *-module* option followed by a file name specifies the MDF file to process. The *-initial* option tells Register Wizard to generate module structural code with a template for the module core and ModelSim scripts.

Register Wizard will output messages during execution as shown in Listing 9.

Note that the additional files generated with the -initial option can be modified by the user. Register Wizard will not overwrite these files if run over again with the -initial option, but will output an error message for every file this applies to. For a complete list of all generated files and which file can be modified see section 0.

Listing 9 Register Wizard messages during execution

```
[INFO]: loading <path>/irqc.regwiz
[INFO]: generating RTL for module 'irqc'
[INFO]: generating compiler script 'comp_irqc.tcl' for module 'irqc'
[INFO]: generating simulation script 'sim_irqc_sbi_tb.tcl' for module 'irqc',
interface 'sbi'
[INFO]: generating compiler script 'comp_irqc_sbi_tb.tcl' for module 'irqc',
interface 'sbi'
[INFO]: Generating SW...
[INFO]: Writing Module SW header file: irqc_sbi.h
[INFO]: Generating test benches...
[INFO]: Generating documentation for module 'irqc'
```

Always review the messages for warnings and errors.

Reviewing Generated Files

Register Wizard has now created a directory **irqc** in the directory where the IRQC MDF is located. There are five directories in the **irqc** directory: **src**, **tb**, **sw**, **doc** and **sim**. We will now go through each of these directories and describe all present files.

src: This directory contains the HDL files for the IRQC module:

- **irqc.vhd**
  this file contains the top level of the design, and must be further maintained by the user. In case of the IRQC, three signals from Table 7 must be added: *irq2cpu*, *irq_source*, *irq2cpu_ack*. These signals must also be port mapped to the *i_core* entity in the *irqc_core.vhd* below.

  These files can be modified by the user. Register Wizard will not overwrite these files.

- **irqc_core.vhd**
  this file contains the IRQC core itself, and must be further maintained by the user. In case of the IRQC, three signals *irq2cpu*, *irq_source*, *irq2cpu_ack* must be added as for the *irqc.vhd*. Additionally, the functionality of the IRQC core must be implemented.

  These files can be modified by the user. Register Wizard will not overwrite these files.

- **irqc_pif_pkg.vhd**
  this is the IRQC PIF package file which defines signals between the core and the PIF. Verify that the names and widths of the signals are equal to what was defined in the MDF. Also note that the prefixes of the signals are equal to the parameters in the MDF, e.g., *awo* for core-located write-only signal, and *rw* for a read/write register.

  Do not modify this file! Register Wizard can overwrite this file.

- **irqc_sbi.vhd**
  this file implements the IRQC register interface, and contains the registers with address decoding logic.

  Do not modify this file! Register Wizard can overwrite this file.

- **irqc_sbi_pkg.vhd:**
  this file contains constants for all register addresses. The register addresses have been set automatically as a result of the address parameters set to auto in the MDF.

  Do not modify this file! Register Wizard can overwrite this file.

tb: this directory contains the test bench files for the IRQC module interface.

- **irqc_sbi_tb.vhd:**
  this is the VHDL test bench for the PIF. The IRQC core is not verified by this test bench. The test bench checks behavior of the generated registers. Listing 10 shows a part of the test bench.

  ⚠ **Do not modify this file! Register Wizard can overwrite this file.**

***Listing 10 Part of the test bench file* irqc_sbi_tb.vhd.**

```
...—Check RW field IER.
    -- Field bit 0, register bit 0
    sbi write(C ADDR IER, "00000000000000000000000000000001", "Invert bit from
default");
    check_value(sbi_p2c.rw_ier, "00000000000000000000000000000001", ERROR,
"Check signal");
    sbi_check(C_ADDR_IER, "00000000000000000000000000000001", "Check
register");
...
```

sw: this directory contains the software header files for the IRQC module interface.

- **irqc_sbi.h:**
  this file contains constants for all register addresses. Listing 11 shows part of the file.

  ⚠ **Do not modify this file! Register Wizard can overwrite this file.**

***Listing 11 Part of the software header file* irqc_sbi.h.**

```
#ifndef _IRQC_PERIPHERALS_H_
#define _IRQC_PERIPHERALS_H_
#define IRQC_SBI_IER                0x0
#define IRQC_SBI_IRR                0x4
#define IRQC_SBI_IPR                0x8
 ...
#endif
```

doc: this directory contains the documentation for the IRQC module interface.

- **irqc_moduleregistermap.docx:**
  this file contains the register map as shown in Figure 5.

  ⚠ **Do not modify this file! Register Wizard can overwrite this file.**

- **irqc_registerdescription.docx:**
  this file contains description of each register listed in **irqc_moduleregistermap.docx** as shown **in** Figure 6**.**

  ⚠ **Do not modify this file! Register Wizard can overwrite this file.**

Name	Address	Access	Width	Default	Description
IRQC_SBI_IER	0x00	RW	32	0x0	Interrupt Enable Register.
IRQC_SBI_IRR	0x04	RO	32	0x0	Interrupt Request Register.
IRQC_SBI_IPR	0x08	RO	32	0x0	Interrupt Pending Register.
IRQC_SBI_ICR	0x0C	WT	32	0x0	Interrupt Clear Register.
IRQC_SBI_ITR	0x10	WT	32	0x0	Interrupt Clear Register.
IRQC_SBI_IRQ2CPU_ENA	0x14	WT	1	0x0	Interrupt Request to CPU Register.
IRQC_SBI_IRQ2CPU_ALLOWED	0x18	RO	1	0x0	Interrupt Request to CPU Allowed Register.

Figure 5 Register map as in the* irqc_moduleregistermap.docx *file.

0x0 - IRQC_SBI_IER

Interrupt Enable Register

Bit	Name	Access	Default	Description
31:0	IER	RW	0x0	Interrupt Enable Register.

0x4 - IRQC_SBI_IRR

Interrupt Request Register

Bit	Name	Access	Default	Description
31:0	IRR	RO	0x0	Interrupt Request Register.

***Figure 6 Part of the Register description file* irqc_registerdescription.docx.**

sim: this directory contains the simulation files for the IRQC module interface.

- **comp_irqc.tcl:**
  this file is a ModelSim compile script which compiles all generated HDL files listed in the **rtlSources** into the library **rtlLibrary**.
- **comp_irqc_sbi_tb.tcl:**
  this file is a ModelSim compile script which compiles all test bench files listed in the **tbSources** into the library **tbLibrary**.
- **sim_irqc_sbi_tb.tcl:**
  this file is a ModelSim simulation script which elaborates the test bench and the design, sets up waveforms (GUI only) and runs simulation.

Do not modify these files! Register Wizard can overwrite these files.

2.7.2.4 Step 4: Verifying the Design

We will now verify the generated HDL files by running the ModelSim simulation scripts. First we have to execute the ModelSim compile scripts.

Running ModelSim compiler scripts

To compile, execute the following from the command line to compile the PIF and the IRQC core:

```
$ vsim -do "comp_irqc.tcl"
```

Execute the following command, either in the command line, or in ModelSim GUI command line if Modelsim GUI started after the previous command.

```
$ vsim -do "comp_irqc_sbi_tb.tcl"
```

The test bench for the SBI is now compiled by Modelsim.

Now execute the simulation of the SBI controller by executing this command:

```
$ vsim -do "sim_irqc_sbi_tb.tcl"
```

The simulation log shown in Listing 12 lists verification actions. Listing 13 shows the verification summary at the end of the simulation.

Listing 12 Simulation log

```
# Bitvis: ID_LOG_HDR              3917.5 ns  TB seq.          Checking
IRQ2CPU_ENA
# Bitvis: ----------------------------------------------------------------------
-----------------------------------------------------
# Bitvis: ID_POS_ACK              3917.5 ns  TB seq.          check_value() =>
OK, for std_logic 0. awo_irq2cpu_ena default value
# Bitvis: ID_BFM                  3927.5 ns  SBI BFM          SBI
check(A:x"14", x"0")=> OK, read data = x"00000000". IRQ2CPU_ENA
# Bitvis:                                                     default value

...
```

Listing 13 Verification summary

```
# Bitvis:
================================================================================
=====================================
# Bitvis:      *** FINAL SUMMARY OF ALL ALERTS  ***
# Bitvis:
================================================================================
=====================================
# Bitvis:                                REGARDED   EXPECTED  IGNORED
Comment?
# Bitvis:                NOTE         :      0         0         0         ok
# Bitvis:                TB_NOTE      :      0         0         0         ok
# Bitvis:                WARNING      :      0         0         0         ok
# Bitvis:                TB_WARNING   :      0         0         0         ok
# Bitvis:                MANUAL_CHECK :      0         0         0         ok
# Bitvis:                ERROR        :      0         0         0         ok
# Bitvis:                TB_ERROR     :      0         0         0         ok
# Bitvis:                FAILURE      :      0         0         0         ok
# Bitvis:                TB_FAILURE   :      0         0         0         ok
# Bitvis:
================================================================================
=====================================
# Bitvis:      >> Simulation SUCCESS: No mismatch between counted and expected
serious alerts
# Bitvis:
================================================================================
=====================================
```

3 Technical Reference

This part of the User Manual contains technical details about Register Wizard. You should at least browse through the following sections to get familiar with the content. This will help you to quickly look up the information you might need when using Register Wizard.

3.1 Command Line Options

Table 9 lists all register Wizard options.

Table 9 Command line options

Option	Required operand	Description
-help (-h, -?)		Prints usage and options and exits.
-debug (-d)		Print debug information.
-module	*<file>*	Specifies a MDF to process.
-config	*<file>*	Specified a configuration file.
-initial		Generates additional files with a template for the module core and ModelSim scripts.
-no-doc		Suppress generation of documentation. By default, documentation is generated if documentation templates are listed in MDF.
-no-rtl		Suppress generation of HDL files. By default, HDL files are generated for the processor interface sub-module.
-no-sw		Suppress generation of software files. By default, software header files are generated if defined in MDF.
-no-tb		Suppress generation of test bench. By default, a simple test bench is generated for the processor interface sub-module.

3.1.1 Option dependencies

Table 10 shows option dependencies.

Table 10 Command line option dependencies

Option	Description
-help (-h, -?)	Cannot be used with **-v** or **-module**
-version (-v)	Cannot be used with **-h** or **-module**
-debug (-d)	Can be combined with all arguments.
-config	Can only be used with **-module**
-initial	Can only be used with **-module**
-no-doc	Can only be used with **-module**
-no-rtl	Can only be used with **-module**
-no-sw	Can only be used with **-module**
-no-tb	Can only be used with **-module**

3.2 Model Description File

In Register Wizard a model is referring to a description of a module. The description is contained in a Model Description File (MDF). MDF files comply with the JSON file format. The following sections specify the structure of the module MDF. The module object is the top object in a module MDF. Other objects, such as interfaces, registers and fields are organized in a hierarchy as shown in Figure 7.

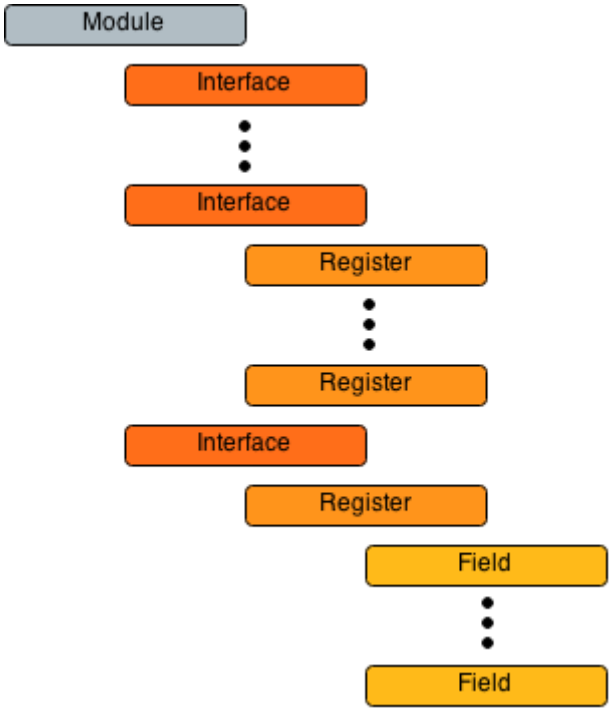

Figure 7 Module MDF structure

3.2.1 Module Object

A module object is parent to one or more Interface objects. Listing 14 shows the JSON format, while Table 11 lists the parameters for the module object.

Listing 14 Module object in JSON format

```
{
        "name": "",
        "configuration": {},
        "interfaces": []
}
```

Table 11 Module object parameters

Parameter	Description	Necessity	Type
name	Name of the module	required	string
configuration	Configuration for this module	optional	configuration object
interfaces	Interface list	required	array of interface objects

3.2.1.1 Interfaces Object

An interfaces object is parent to one or several register objects and is child to a module object. Listing 15 shows the JSON format, while

Table 12 lists the parameters for the interfaces object.

Listing 15 Interfaces object in JSON format

```
"interfaces": [
        {
        "name": "",
        "configuration" : {},
        "description": [],
        "type": "",
        "addressWidth": "",
        "dataWidth": "",
        "registers": []
        }
]
```

Table 12 Interfaces object parameters

Parameter	Description	Necessity	Type
name	Interface name	required	string
configuration	Configuration for this Interface	optional	configuration object
description	Interface description	optional	string
type	Interface type. Valid types: *SBI*	required	string
registers	Register list	required	array of register objects

3.2.1.2 Register Object

A register object is parent to one or more field objects and child to an interfaces object. Listing 16 shows the JSON format, while lists the parameters for the register object.

The necessity of some parameters depends on whether there are field objects linked to the register object. These parameter values are automatically calculated based on field values, thus the corresponding value assignment in the register object is not allowed. The relevant parameters are marked as *not allowed* in

Table 13.

Listing 16 Register object in JSON format.

```
"registers": [
      {
      "name": "",
      "configuration": {},
      "address": "",
      "summary": [],
      "description": [],
      "width": ,
      "access": "",
      "signal": ""
      "reset": "",
      "location": "",
      "coreSignalProperties": {},
      "fields": []
      }
```

Table 13 Register object parameters

Parameter	Description	Type	Necessity without fields	Necessity with fields
name	Register name.	string	required	required
configuration	Configuration for this register	configuration object	optional	optional
address	Register address. *auto|<value>[:stride:<count>[:<increment>]]* Register addresses may be hard-coded, or automatically computed by setting value to *auto*. A hard-coded value always overrides an automatically computed address. An address set to *auto* in a proceeding register or field is based on the hard-coded address and incremented accordingly. The auto increment assume byte addressing, and is dependent on the data width. Address is auto-incremented according to the following formula: *increment = data width / 8* *address* value follows the format specified in Table 15. A register address may be specified as a stride, allowing a register to be replicated specified number of times. When using strides, a stride count must be specified with an optional stride increment. The stride count specifies number of replicated registers, and the stride increment specifies address increment between stride registers. The increment should be given in bytes, so the data width must be kept in mind. Default stride increment is the same as the auto-increment for non-stride registers.	string	required	required
summary	Register summary	array of string	optional	optional
description	Register description	array of string	optional	optional
width	Register data width	number	required	not allowed
access	Register access type. Valid types: *RW*, *RO*, *WO* *RW: Read/Write*. The register is written and read through the register interface. The value read from the register is the last value written. *RO: Read-Only*. The register cannot be written - writes are discarded. The value read from the register is supplied from the core. *WO: Write-Only*. The register can only be written and not read back (always reads back as 0). See the below table for relation of this parameter to the register location.	string	required	not allowed
signal	Signal type. Valid types: *std_logic*, *std_logic_vector*, *unsigned*, *signed*, *boolean*	string	required	not allowed
reset	Reset value *reset* value follows the format specified in Table 15.	number	required	not allowed

location	Register location. Valid values: *pif, core* See Table 14 for the relation of the register location to the *access* parameter.	string	required	required *
coreSignalProperties	Core signals are defined with the following boolean parameters: *useReadEnable*: Adds a read enable signal from the PIF to the core, triggering a read. Necessary if, for example, your register is implemented in RAM or as a FIFO. *useWriteEnable*: Adds a write enable signal from the PIF to the core, triggering a write. Probably necessary for any writable register located in the core. All core signal parameters defaults to 'false'	object	optional	not allowed
fields	List of fields objects	array of field objects	not allowed	required

**Location is optional for a register if location is defined for all fields within it.*

Table 14 summarizes what is generated for different combinations of the register location and the *access* parameters.

Table 14 Results based on the combination of the access and the register location.

Access	Location	Description
RW	pif	Normal, Read/Write register in PIF. Core gets signal from PIF with current value.
RW	core	Read/Write register in core. Core gets data value from register access and optionally read/write strobes.
RO	pif	Read-Only register in PIF. Can only have a fixed value (given by *reset* value). Useful for version registers, but not much else.
RO	core	Read-Only register in core. Core sends data to PIF. Used for status registers, etc.
WO	pif	Write-Only register in PIF.
WO	core	Write-Only register in core. Useful for functionality where the written data is no longer available, e.g. it was pushed out on an interface or into a FIFO.
WO*	core (*useWriteEnable = false*)	Write-to-Trigger. Generates a single-cycle pulse from PIF to core when written.

Table 15 Address and reset value format

Format	Value prefix	Quotes	Example
Binary	0b	yes	"0b0010"
Decimal	0d (optional)	optional	23 or ("0d23")
Hexadecimal	0x	yes	"0xFF"

3.2.1.3 Field object

A field object is child to a register object. Listing 17 shows the JSON format, while Table 16 lists the parameters for the field object.

Listing 17 Field object in JSON format

```
"fields": [
      {
      "name": "",
      "position": "",
      "description": [],
      "access": "",
      "signal": "",
      "reset": "",
      "location": "",
      "coreSignalProperties": {}
      }
]
```

Table 16 Field object parameters

Parameter	Description	Necessity	Type
name	Field name	required	string
position	Field position Valid format for position is bit number and msb:lsb For example: A single bit can be defined as 0, 1, 2, etc… A vector can be defined as *15:0*.	required	string
description	Field description	required	array of string
access	Field access type. Valid types: *RW*, *RO*, *WO* See Table 14 for details.	required	string
signal	Field signal type. Valid types: *std_logic*, *std_logic_vector*, *unsigned*, signed, *boolean*	required	string
reset	Field reset value *reset* value follows the format specified in Table 15.	required	number
location	Register location. Valid values: *pif, core*	required*	string
coreSignalProperties	This parameter has the same properties as in the Register.	Optional	object

**If field-location is not specified, the register location (if specified) will be used instead. Location must be specified either for the register, or for all fields.*

3.3 Configuration

You can configure Register Wizard through a configuration file and from within a MDF. Within a MDF, configuration can be provided for the whole model and/or individually for each object supporting configuration. A configuration file or a configuration provided in a MDF consists of a configuration object defined in section 3.3.2. Partial configuration is allowed, meaning that you can choose which configuration parameters to provide. Default built-in values are used for any parameters not provided.

User configuration is optional.

3.3.1 Configuration Prioritization

Configuration can be provided both as a configuration file and in the MDF simultaneously, a prioritized configuration hierarchy is defined as shown in Figure 8. The following rules apply:

- Any user provided configuration parameter will override the corresponding built in default configuration parameter.
- Any configuration parameter in a MDF will override the corresponding parameter from a configuration file.
- Within a MDF, provided configuration parameters for an object applies to given object and all objects below it in the object hierarchy, regardless of present configuration parameters for any object above in the object hierarchy.

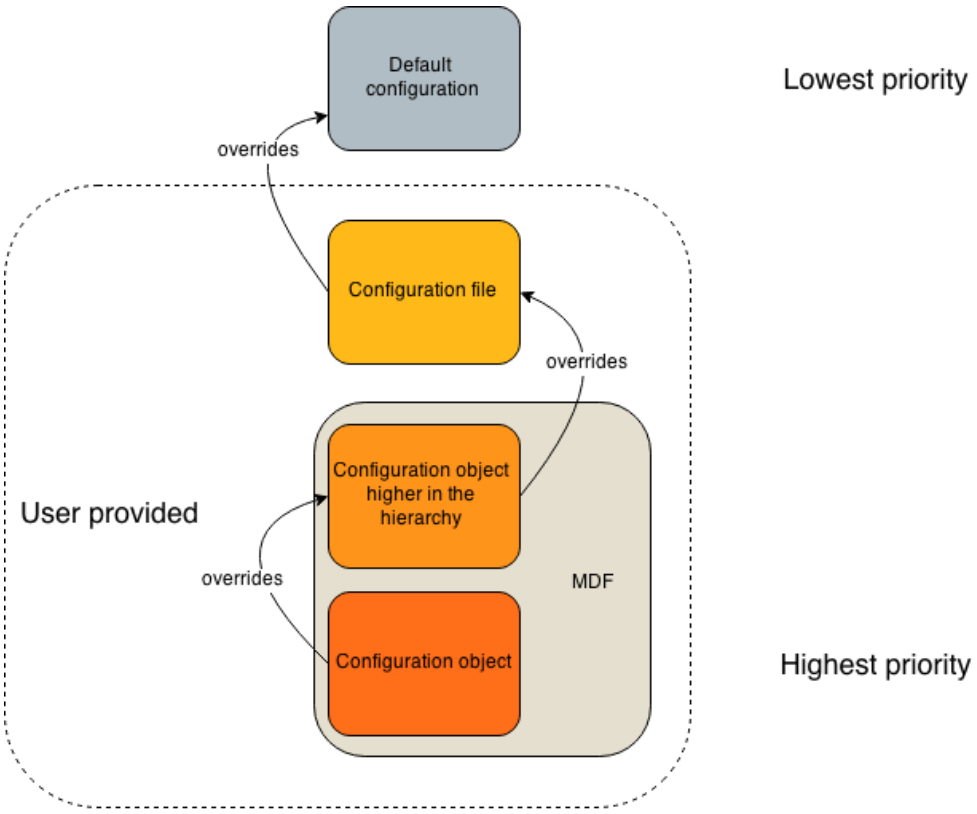

Figure 8 Configuration prioritization.

3.3.1.1 Configuration Prioritization Example

Figure 9 visualizes a few configuration parameters provided simultaneously, both through a configuration file and within a MDF. Register Wizard assigns default values to all parameters shown in grey in Figure 9. The default value for parameter A is here overridden only in the configuration file so it becomes effective for all objects in the MDF. Parameters B and C are overridden in the configuration for the module object. Parameter C is effective for the module object and any object below in the hierarchy. Parameter B, however, is overridden in the configuration of the interface object. This means that the value assigned to parameter B in the module object does not apply to the interface object or any object further below in the hierarchy.

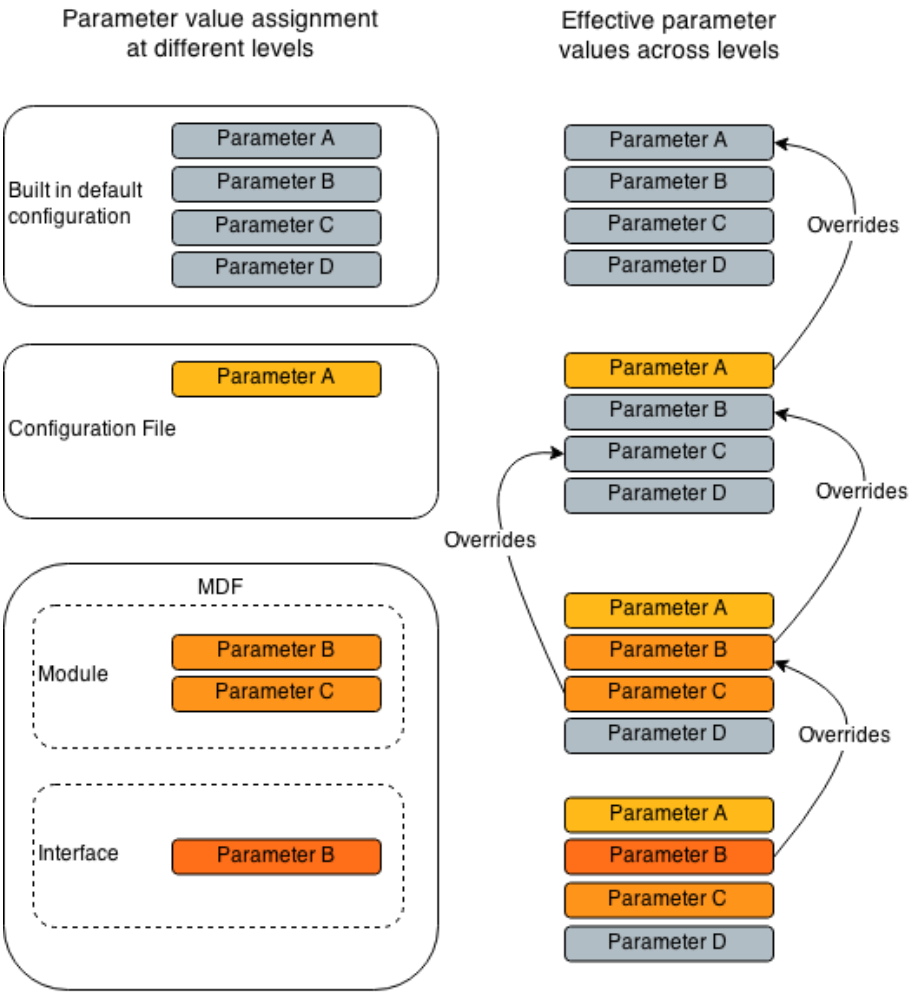

Figure 9 Configuration prioritization example.

3.3.2 Configuration Object

A common configuration object is defined for all model objects supporting configuration, however, not all configuration parameters are relevant to every object. Which parameter applies to which object is specified in the tables below in column Applies to. Listing 18 shows the JSON format, while

Table 17 lists the main parameter groups for the configuration object. The proceeding sections describe each parameter group in details.

Listing 18 Configuration object in JSON format.

```
"hdl":
      {
      "rtlPath" : "",
      "simPath" : "",
      "tbPath"  : "",
      "rtlLibrary" : "",
      "tbLibrary"  : "",
      "headerTemplate" : "",
      "indentation" :
      },
"software" :
      {
      "path" : "",
      "headerTemplate" : "",
      "indentation" :
      },
"documentation" :
      {
      "path" : "",
      "moduleTemplate" : [
      "",
      ...,
      ]
      },
"interface" :
      {
      "sbi" : {
                "slaveAddressing" : "",
                "addressWidth" : "",
                "dataWidth" : ""
      }
}
```

Table 17 Configuration object parameters.

Parameters	Description	Type	Necessity
hdl	Configuration for HDL generation	object	optional
software	Configuration for software header file generation	object	optional
documentation	Configuration for documentation generation	object	optional
interface	Configuration for given interface	object	optional

3.3.2.1 Path parameters

Several of the configuration parameters sets a path to an input or an output file or a directory. Path related parameters are always prefixed with one of the following modifiers:

- *<model_file>* : given path is relative to the MDF path
- *<config_file>* : given path is relative to the configuration file path
- *<absolute>* : given path is absolute
- *<parent>* : the modifier is inherited if you have provided this path parameter in a configuration object of higher priority or a default value is used.

Listing 19 is an example of how path can be specified. The given *rtlPath* is relative to the default location if the *rtlPath* parameter is not specified with other modifier than *<parent>* in a configuration file. The path given by *simPath* is relative to the MDF path, while the *tbPath* is relative to the configuration file. The path to the header template file given by the *headerTemplate* is absolute.

Listing 19 Example of path specification.

```
"hdl":
      {
      "rtlPath" : "<parent>/rtl",
      "simPath" : "<model_file>/sim",
      "tbPath"  : "<config_file>/testbench",
      "headerTemplate" : "<absolute>/headers/header.txt"
      }
```

3.3.2.2 Hdl Object

This object defines the output paths for all generated HDL files and the path to the file header template. Table 18 lists all parameters for the hdl object, while the JSON format is shown in Listing 18.

Table 18 Hdl objects parameters.

Parameters	Default value	Description	Necessity	Type	Applies to object
rtlPath	*<model_file>./rtl*	The directory where generated HDL files are placed	optional	string	module
tbPath	*<model_file>./tb*	The directory where generated test bench is placed	optional	string	module
simPath	*<model_file>./sim*	The directory where scripts are generated	optional	string	module
rtlLibrary	*lib_<module name>*	Name of the VHDL library into which HDL is compiled	optional	string	module
tbLibrary	*lib_<module name>*	Name of the VHDL library into which test bench is compiled	optional	string	module
headerTemplate	*<model_file>* Path modifier only	The texts file which will be used as header in the generated code. If this parameter is not given, no custom header is created. See section 3.4.1 for more details.	optional	string	module
indentation	*2*	The indentation levels for all generated VHDL code	optional	number	module

3.3.2.3 Software Object

This object defines the output path for all generated software files and the path to the file header template. Table 19 lists all parameters for the software object, while the JSON format is shown in Listing 18.

Table 19 Software object parameters

Parameters	Default value	Description	Necessity	Type	Applies to object
path	*<model_file>./sw*	The directory where generated software header file is placed	optional	string	module
headerTemplate	*<model_file>* Path modifier only	The text file which will be used as header in the generated code. If this parameter is not given, no custom header is created. See section 3.4.1 for more details.	optional	string	module
indentation	*4*	The indentation levels for generated SW header file	optional	number	module

3.3.2.4 Documentation Object

This object defines the output path for all generated documentation and the path to the documentation templates. Table 20 lists all parameters for the documentation object, while the JSON format for the documentation object is shown in Listing 18.

Table 20 Documentation object parameters

Parameter	Default value	Description	Necessity	Type	Applies to object
path	*<model_file>./doc*	The directory where generated documentation is placed	optional	string	module
moduleTemplate	*[<absolute><install_dir>/regwiz/templates/docx/moduleregistermap-template.docx,* *<absolute><install_dir>/regwiz/templates/docx/registerdescription-template.docx]*	Path to documentation templates. Can be a single file, or a list of several templates.	optional	string	module

3.3.2.5 Interface Object

The Interface configuration object consists of sub-objects specifying behavior for each interface type. Only the SBI is supported in the current version of Register Wizard. Table 21 lists all parameters for the interface object, while the JSON format is shown in Listing 18.

Table 21 Interface objects parameters

Parameters	Default value	Description	Necessity	Type	Applies to object
sbi	*N/A*	object for SBI interfaces	optional	object	module, interfaces

Sbi Object

The Sbi configuration object consists of parameters that specify the SBI bus behavior. Table 22 lists all parameters for the interface object, while the JSON format is shown in Listing 18. The generated interface output signal *ready* is set constant high by Register Wizard. For more information about the SBI protocol, see section 6.1.

Table 22 Sbi objects parameters

Parameters	Default value	Description	Necessity	Type	Applies to object
slaveAddressing	*chipselect*	*chipselect, baseaddress*. Sets whether the slaves shall be accessed by using chip select signals or base addresses.	optional	string	module, interfaces
addressWidth	*none*	Address width in bits. Address width must be equal or larger than the largest register.	optional	number	module, interfaces
dataWidth	*none*	Data width in bits.	optional	number	module, interfaces

3.4 Templates

You can customize files generated by Register Wizard through use of templates. Two types of templates are supported; file header templates and documentation templates. These are described in the following sections.

Templates are assigned to a module through the configuration object in a MDF or in a configuration file as described in section 3.3.

3.4.1 File Header Templates

You can provide your own file header templates to be inserted in all generated HDL, test bench, ModelSim scripts and software header files. The file header template is a plain text file. The content of this file is inserted below the fixed header. Each line of the header gets commented out in the appropriate way for the file it is inserted into. The same template can be used for all files, or separately for the HDL related files and the software header files as described in Table 18 and Table 19.

3.4.1.1 Default File Header Templates

Register Wizard is installed with a default file header template. The default file header template can be found in **<install_dir>/regwiz/templates/headers/header.txt**.

You should always copy the default template to another location before modifying it.

3.4.2 Documentation Templates

You can customize layout and content for documentation generated by Register Wizard by providing documentation templates written in Office Open XML format. Section 3.4.3 describes how to customize the documentation templates.

3.4.2.1 Default Documentation Templates

Register Wizard provides default documentation templates for both the register map and the register description documentation. These template files are listed in Table 23 and can be found in the **<install_dir>/regwiz /templates/docx/** directory.

Table 23 Documentation template files

Template file	Description
moduleregistermap-template.docx	Template for module register map documentation.
registerdescription-template.docx	Template for register description documentation.

You should always copy the default template to another location before modifying it.

3.4.3 Working with documentation templates

The Office Open XML format allows you to customize the layout for the generated documentation files in the same way as for regular documents. In addition, Register Wizard supports merge fields. Merge fields allows text to be inserted into a document at specific locations without any knowledge of the document layout and style. The following sections lists supported merge fields and shows you how to use them

3.4.3.1 Register Wizard Merge Fields

Register Wizard support three merge field collections, each corresponding to a MDF object.

Interface Object Merge Field

The interface merge field collection corresponds to the interface object described in section 3.2.1.1. Table 24 lists all merge fields in the interface collection.

Table 24 Interface fileds available in the template

Interface object	Description
$interface.name	Name of interface
$interface.moduleName	Name of module which the interface is part of
$interface.registers	Array of register objects

Register Object Merge Field

The register merge field collection corresponds to the register object described in section 3.2.1.2. Table 25 lists all merge fields in the register collection.

Table 25 Register merge fileds available in the template

Register object	Description
$register.name	Name of register
$register.moduleName	Name of module which the register is part of
$register.interfaceName	Name of interface which the register is part of
$register.reset	Register Reset value
$register.access	Register access type
$register.address	Register address
$register.globalAddress	Global address
$register.width	Register width
$register.description	Register description
$register.summary	Register summary
$register.fields	Array of field objects

Field Object Merge Field

The field merge field collection corresponds to the field object described in section 3.2.1.3. Table 26 lists all merge fields in the field collection.

Table 26 Field merge fileds available in the template

Field object	Description
$field.name	Name of field
$field.position	Field position
$field.access	Field access type
$field.reset	Field reset value
$field.description	Field description

Inserting Merge Fields

The following sections apply to Microsoft Word.

To insert a merge field into a template do the following:

- Set the curser where you want to insert the merge field
- Press **CTRL+F9**, Word generates an empty merge field **{ }** as shown in Figure 10.

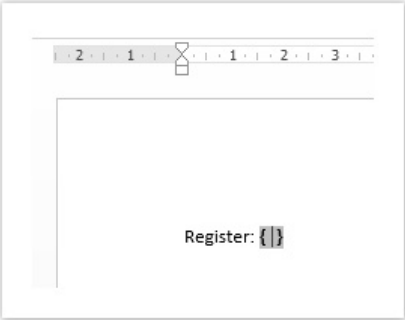

Figure 10 Insertion of a merge field.

- Select it and right-click to select **Edit Field...**.
- The Field dialog opens. Select **MergeField** as shown in Figure 11.

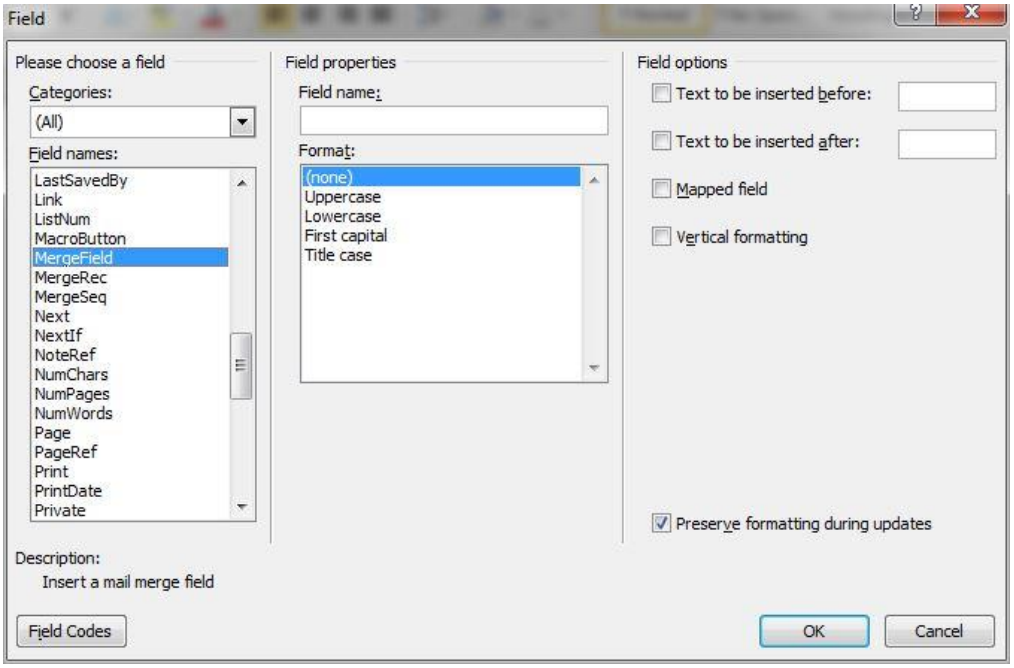

Figure 11 Field dialog.

- Enter a field name, e.g. **$Register.name**, as shown in Figure 12, and click on OK.

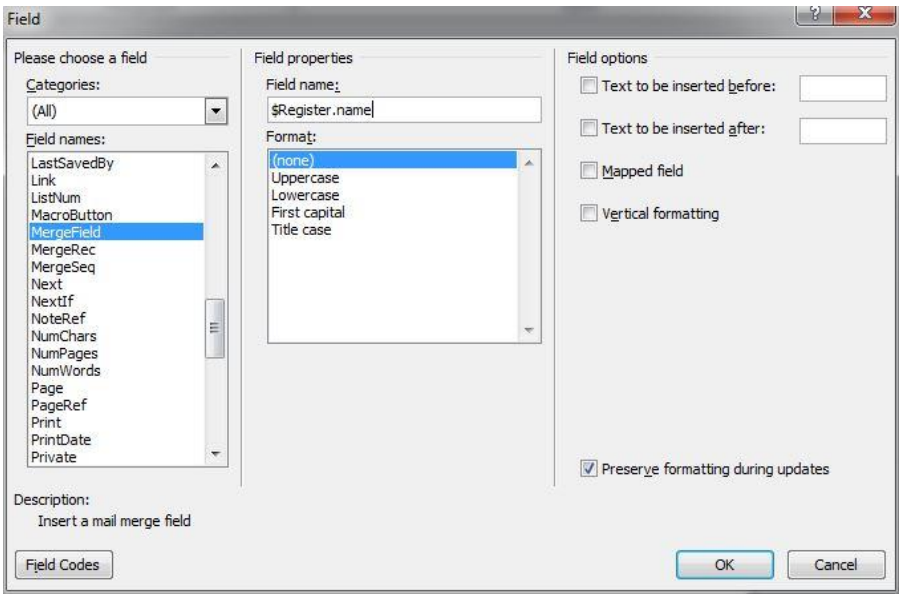

Figure 12 Entering field name.

- Now the merge field should be created as shown in Figure 13.

Figure 13 Newly created merge field

Working with field arrays

Array fields are inserted using a loop syntax as shown in Listing 20.

Listing 20 Syntax for inserting fields from an array

```
foreach(<array index> in <array>)
      <array index>.<array member>
#end
```

3.5 Generated Output

You can use Register Wizard to generate a number of files from a single source as shown in Figure 14. All generated files are described in the following sections.

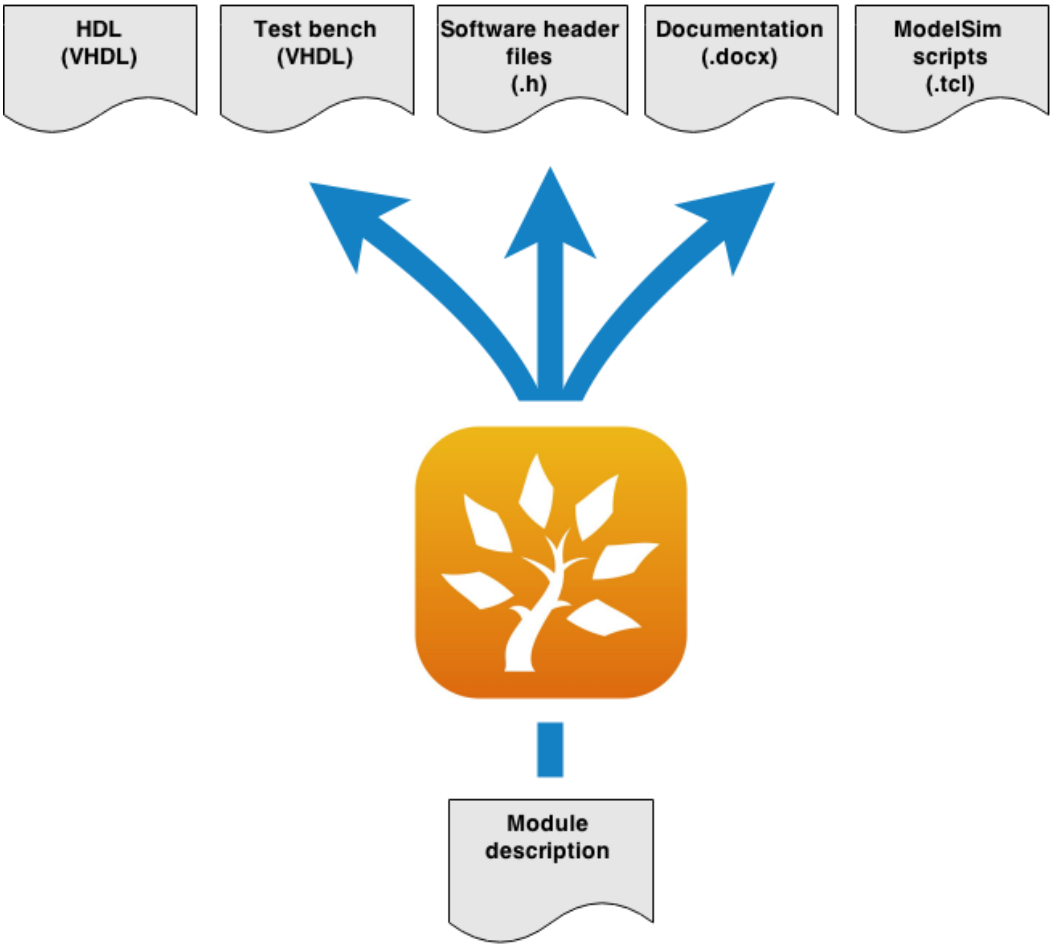

Figure 14 Overview of files generated by Register Wizard

3.5.1 HDL Files

Register Wizard splits the module into code maintained by Register Wizard and code maintained by the user as shown in Figure 15.

Register Wizard maintains the PIF, which implements the protocol of the processor bus and contains all the physical registers. It does address decoding and selection of read data. Register Wizard also maintains a package with all the address information, and a package describing the interface between the PIF and the Core.

The user must maintain the Core, where the user functionality of the module resides. This can be generated as an initial template by Register Wizard, when run with the *-initial* option. No functional logic is generated – only an entity and an empty architecture. The *-initial* option also generates the Module top level, connecting the PIF and the Core. You can add other interfaces to the Module and Core. The signal package ensures that the Core entity and the Module architecture don't need to know about the details of the interface between the PIF and the Core.

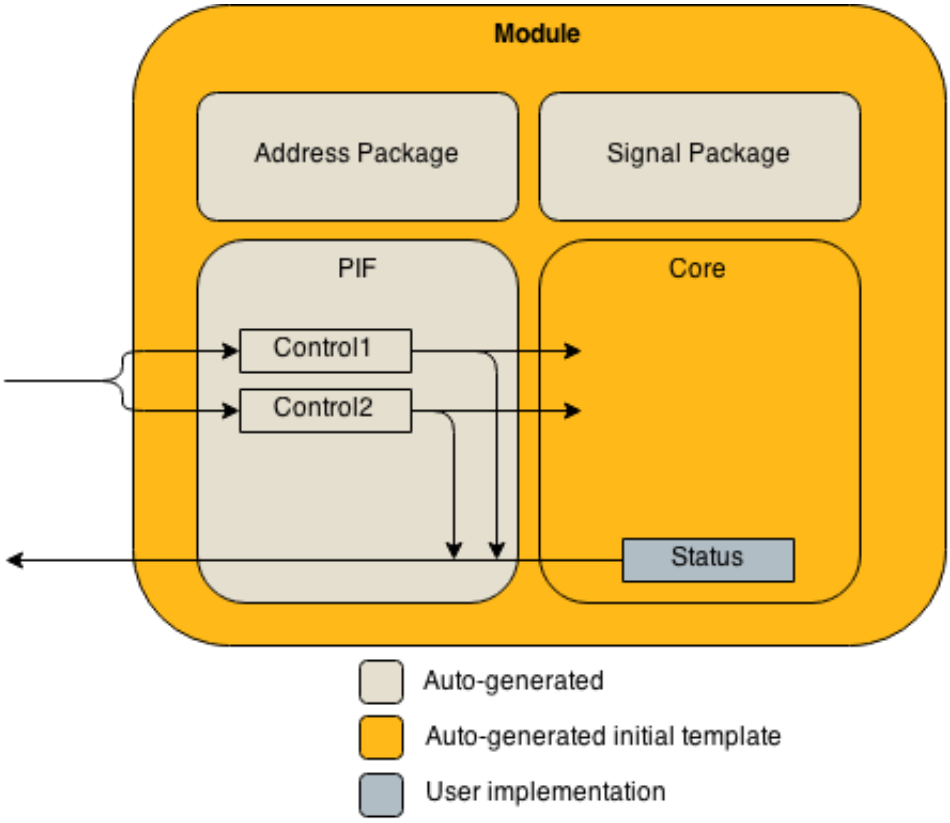

Figure 15 Module hierarchy

Occasionally, a register can have special requirements that are not easily generated. For example, you may want to place a FIFO at a register address, so the processor is able to write directly to the FIFO. In such cases, you set the register location to core, as shown in Figure 16. Register Wizard will not generate such registers, only the address decoding necessary to write to or read from them (in the PIF module). This functionality enables you to place any kind of custom logic behind a register address on the processor bus.

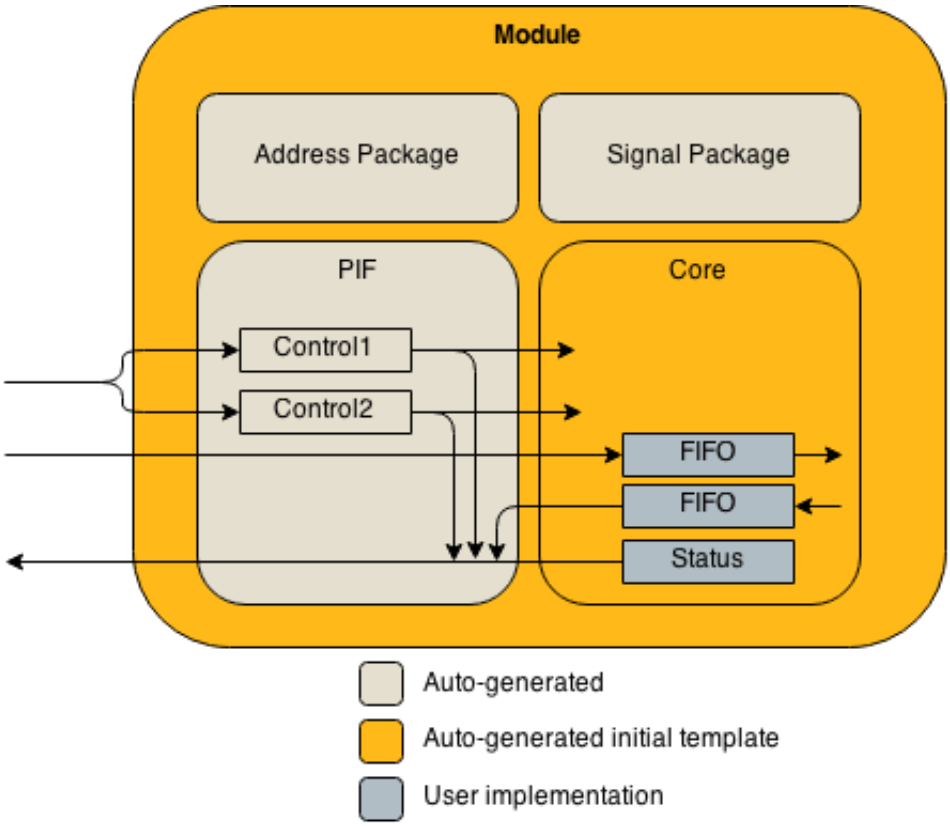

Figure 16 Core-located registers

Register Wizard generates HDL files in VHDL. By default, the HDL files listed in Table 27 are generated.

Execute Register Wizard with option -initial to generate files listed in

Table 28.

Table 27 Generated HDL files

File	Description
<module_name>_<interface_name>_pkg.vhd	Package defining the internal addresses of all registers belonging to one of the interfaces in a module. Each interface (distinguished by the interface name) has its own package.
<module_name>_<interface_name>.vhd	The processor interface sub-module entity and architecture. Each interface (distinguished by the interface name) has its own file.
<module_name>_pif_pkg.vhd	Package that defines the interface between module core and processor interface sub-module(s).

Do not modify these files! Register Wizard can overwrite these files.

Table 28 Generated additional HDL files for option -initial

File	Description
<module_name>.vhd	Instantiates the module core and processor interface sub-module(s).
<module_name>_core.vhd	Empty architecture except for some aliases for the bus interface (clock, reset, p2c, c2p). The core logic should be implemented in this file.

These files can be modified by user. Register Wizard will not overwrite these files.

3.5.2 Test Bench Files

By default, Register Wizard generates a simple test bench for the generated code. This test bench verifies that the registers in the PIF are generated correctly, and that they behave as expected. The test bench can be incorporated into a regression suite, or it can be used as a single "handover test" from Register Wizard to the user.

Note that the test bench covers the generated code in the PIF only, as shown in Figure 17. It has no knowledge of the functionality implemented in the Core.

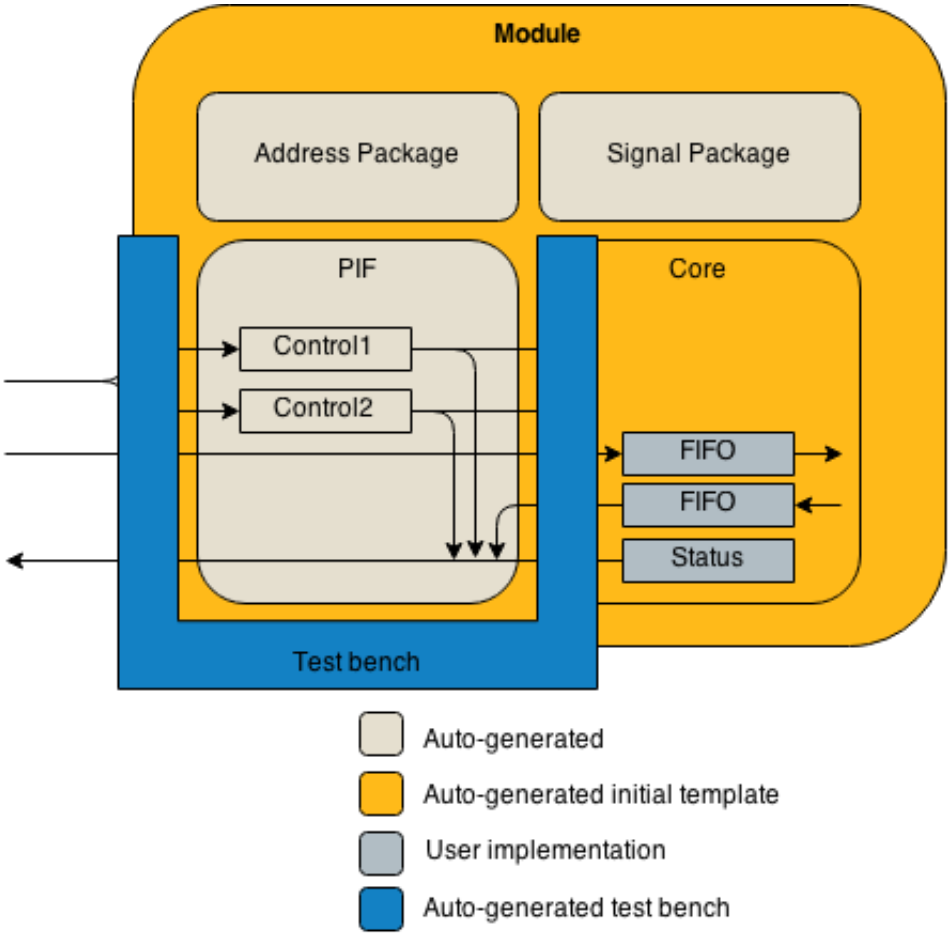

Figure 17 Generated test bench

Register Wizard generates test bench files in VHDL. By default, the files in Table 30 are generated.

Table 29 Generated test bench files

File	Description
<module>_<interface>_tb.vhd	Each interface (distinguished by the interface name) has its own test bench.

⚠ **Do not modify these files! Register Wizard can overwrite these files.**

3.5.3 ModelSim Scripts

Run Register Wizard with option *-initial* to generate files listed in Table 30. The ModelSim compile scripts perform compilation of all design and verification sources.

To run the scripts, execute them from the ModelSim GUI or use the following command:

```
$ vsim -c -do <script>
```

Table 30 Generated ModelSim scripts

File	Description
comp_<module>.tcl	Compiles all the module files in the "rtlPath" directory.
comp_<module>_<interface>_tb.tcl	Compiles a test bench. One script per test bench (distinguished by the interface name).
sim_<module>_<interface>_tb.tcl	Simulates a test bench. One script per test bench (distinguished by the interface name).

⚠ **Do not modify these files! Register Wizard can overwrite these files.**

3.5.4 Software Files

Register Wizard generates a software header files, containing register addresses, as shown in Table 31.

Table 31 Generated Software header files

File	Description
<module_name>_<interface_name>.h	A header file in the C language containing constants for all register addresses. Each interface (distinguished by the interface name) has its own file.

⚠ **Do not modify these files! Register Wizard can overwrite these files.**

3.5.5 Documentation Files

Register Wizard generates documentation files in the Office Open XML format. Table 32 lists the documentation files. The layout and content of the documentation files is specified by templates described in section 3.4.2. Section 3.3.2.4 describe how to assign documentation templates.

Table 32 Generated documentation files

File	Description
<module_name>_<template_file_name>.docx	A documentation file for given module based on a template. The template file name is included in the document file name.

4 License

Register Wizard v1.0.0 is licensed under the Creative Commons Attribution-NoDerivatives 4.0 International Public License.

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution-NoDerivatives 4.0 International Public License ("Public License"). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions.

4.1 Definitions

a. **Adapted Material** means material subject to Copyright and Similar Rights that is derived from or based upon the Licensed Material and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound recording, Adapted Material is always produced where the Licensed Material is synched in timed relation with a moving image.

b. **Copyright and Similar Rights** means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section 4.2(b)(1)-(2) are not Copyright and Similar Rights.

c. **Effective Technological Measures** means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.

d. **Exceptions and Limitations** means fair use, fair dealing, and/or any other exception or limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.

e. **Licensed Material** means the artistic or literary work, database, or other material to which the Licensor applied this Public License.

f. **Licensed Rights** means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.

g. **Licensor** means the individual(s) or entity(ies) granting rights under this Public License.

h. **Share** means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.

i. **Sui Generis Database Rights** means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.

j. **You** means the individual or entity exercising the Licensed Rights under this Public License. **Your** has a corresponding meaning.

4.2 Scope

a. **License grant**.

1. Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:

   A. reproduce and Share the Licensed Material, in whole or in part; and

   B. produce and reproduce, but not Share, Adapted Material.

2. *Exceptions and Limitations.* For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.

3. *Term.* The term of this Public License is specified in Section 4.6(a).

4. *Media and formats; technical modifications allowed.* The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures. For purposes of this Public License, simply making modifications authorized by this Section 4.2(a)(4) http://creativecommons.org/licenses/by-nd/4.0/legalcode - s2a4never produces Adapted Material.

5. *Downstream recipients.*

6. *Offer from the Licensor – Licensed Material.* Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.

7. *No downstream restrictions.* You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed Material.

8. *No endorsement.* Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 4.3(a)(1)(A)(i).

b. **Other rights**.

1. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.

2. Patent and trademark rights are not licensed under this Public License.

3. To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties.

4.3 License Conditions

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

a. **Attribution**.

1. If You Share the Licensed Material, You must:

   A. retain the following if it is supplied by the Licensor with the Licensed Material:

      i. identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);

      ii. a copyright notice;

      iii. a notice that refers to this Public License;

      iv. a notice that refers to the disclaimer of warranties;

      v. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;

   B. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and

   C. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.

   For the avoidance of doubt, You do not have permission under this Public License to Share Adapted Material.

2. You may satisfy the conditions in Section 4.3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.

3. If requested by the Licensor, You must remove any of the information required by Section 4.3(a)(1)(A) to the extent reasonably practicable.

4.4 Sui Generis Database Rights

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

a. for the avoidance of doubt, Section 4.2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database, provided You do not Share Adapted Material;

b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material; and

c. You must comply with the conditions in Section 4.3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4.4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

4.5 Disclaimer of Warranties and Limitation of Liability

a. **Unless otherwise separately undertaken by the Licensor, to the extent possible, the Licensor offers the Licensed Material as-is and as-available, and makes no representations or warranties of any kind concerning the Licensed Material, whether express, implied, statutory, or other. This includes, without limitation, warranties of title, merchantability, fitness for a particular purpose, non-infringement, absence of latent or other defects, accuracy, or the presence or absence of errors, whether or not known or discoverable. Where disclaimers of warranties are not allowed in full or in part, this disclaimer may not apply to You.**

b. **To the extent possible, in no event will the Licensor be liable to You on any legal theory (including, without limitation, negligence) or otherwise for any direct, special, indirect, incidental, consequential, punitive, exemplary, or other losses, costs, expenses, or damages arising out of this Public License or use of the Licensed Material, even if the Licensor has been advised of the possibility of such losses, costs, expenses, or damages. Where a limitation of liability is not allowed in full or in part, this limitation may not apply to You.**

c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

4.6 Term and Termination

a. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.

b. Where Your right to use the Licensed Material has terminated under Section 4.6(a), it reinstates:

   1. automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or

   2. upon express reinstatement by the Licensor.

For the avoidance of doubt, this Section 4.6(b) does not affect any right the Licensor may have to seek remedies for Your violations of this Public License.

c. For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License.

d. Sections 4.1, 0, 4.6, 4.7, and 4.8 survive termination of this Public License.

4.7 Other Terms and Conditions

a. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.

b. Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License.

4.8 Interpretation

a. For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose conditions on any use of the Licensed Material that could lawfully be made without permission under this Public License.

b. To the extent possible, if any provision of this Public License is deemed unenforceable, it shall be automatically reformed to the minimum extent necessary to make it enforceable. If the provision cannot be reformed, it shall be severed from this Public License without affecting the enforceability of the remaining terms and conditions.

c. No term or condition of this Public License will be waived and no failure to comply consented to unless expressly agreed to by the Licensor.

d. Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal processes of any jurisdiction or authority.

5 Register Wizard Change Log

Table 33 lists all Register Wizard releases with the corresponding changes.

Table 33 Register Wizard change log.

Release	Changes
1.0.0	First release

6 Appendix

6.1 Simple Bus Interface

The Simple Bus Interface is a simplified bus interface, yet advanced enough for most peripherals.

The bus provides single-cycle access with completely flexible address and data widths. Since the bus provides single-cycle access, there are no multi-word or pipelining issues. Handshaking by use of a **slave** ready signal is optional.

SBI is an OR-bus. That means that the read data input to the **master** is an OR-ed signal of all the read-data output signals from each **slave**. That in turn requires that the data output signal from each **slave** is only active when the chip select input signal to that **slave** is active. Otherwise, the data output signal must be set to zero.

SBI is not a defined standard.

6.1.1 Timing Diagrams

The following sections describe timing diagrams for the SBI.

6.1.1.1 Read data without a slave ready signal

Figure 18 shows the timing diagram for the read data without a **slave** ready signal. The SBI **master** asserts the *rd_ena* and *addr* signals at period 1. Because of this, the **slave** will present data from address *addr* on *rd_data*. This happens combinatorically. At period 2 the **master** samples the data on *rd_data* and de-asserts *rd_ena* and *addr*, which causes the **slave** to set *rd_data* to 0x00 (shown as a negative edge here for simplicity).

At period 5 the **master** again asserts the *rd_ena* signal along with *addr*. The **slave** presents data on *rd_data* immediately afterwards. The **master** samples *rd_data* at period 6 and increments *addr*. The **master** samples data from address 0x02 on period 7 and the **master** de-asserts *rd_ena* and *addr*, causing the **slave** to set the *rd_data* signal back to 0x00.

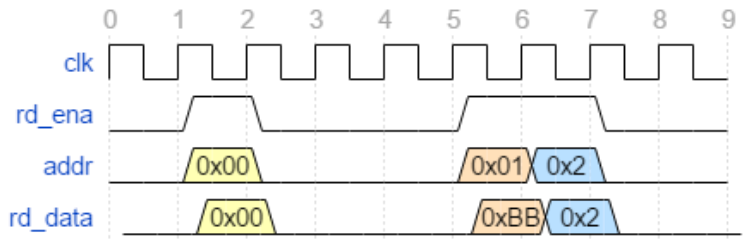

Figure 18 Read data without a slave ready signal

6.1.1.2 Read data with a slave ready signal

With handshaking enabled via a **slave** ready signal the **master** waits until the *ready* signal has been asserted by the **slave** before sampling *rd_data and* de-asserting *rd_ena* and *addr,* as shown in Figure 19. This may take several clock cycles.

At period 1 the **master** asserts *rd_ena* and *addr*. At period 4 the **master** detects that the *ready* signal is high, samples *rd_data*, and de-asserts *rd_ena* and *addr*.

At period 6 the **master** again wants to read data from the **slave**. The **master** reads two data words before the **slave** de-asserts *ready* at period 9. The **master** detects that *ready* is active again at period 11 and samples the final data word before de-asserting *rd_ena*.

Not shown in this waveform is the case where the **slave** asserts *ready* before the **master** asserts *rd_ena*. That case will result in a single-cycle access equal to the case in 6.1.1.1.

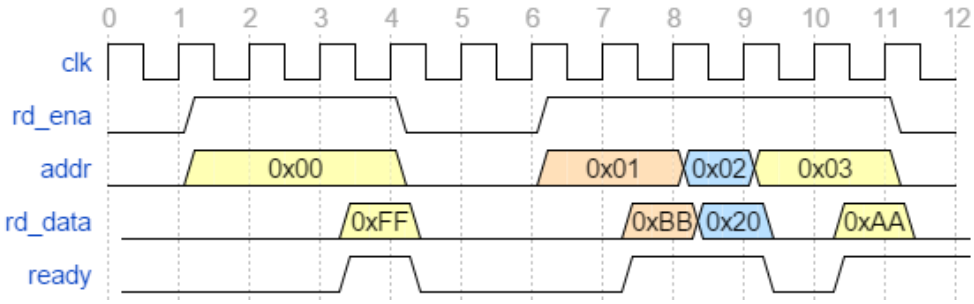

Figure 19 Read data with a slave ready signal

6.1.1.3 Write data without a slave ready signal

Single-cycle write of each data word. To write several data words in a row the **master** simply has to enable *wr_ena* for several clock cycles as shown in see Figure 20.

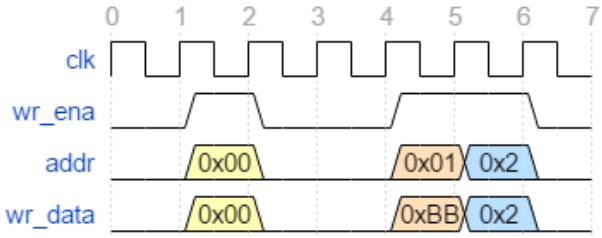

Figure 20 Write data without a slave ready signal

6.1.1.4 Write data with a slave ready signal

The **master** asserts *wr_ena*, *addr* and *wr_data* at period 1 and waits for the **slave** to assert *ready*, as shown in Figure 21. The **slave** asserts *ready* and samples *wr_data* at period 3. At period 4 the **master** detects that *ready* is active and disables *wr_ena*, *addr* and *wr_data*.

At period 6 the **master** again wants to write to the **slave**. The **slave** asserts *ready* at period 7. The **master** is able to write two data words before the **slave** de-asserts *ready* again at period 9. The **master** wants to write a final data word and must therefore wait until *ready* goes high.

At period 10, the **slave** re-asserts *ready*. The master detects that *ready* is high at period 11, which means that the final data word has been written to the **slave**.

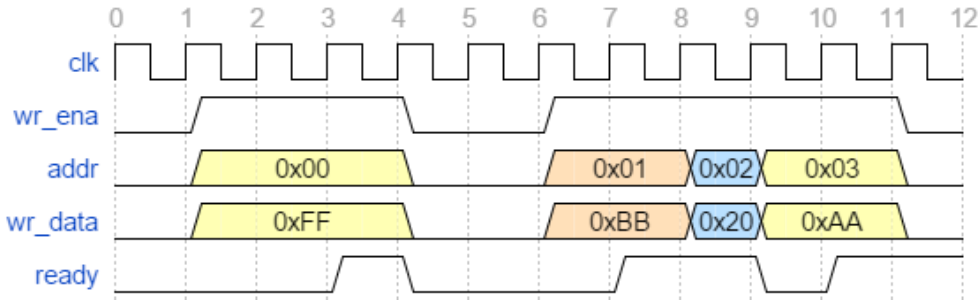

Figure 21 Write data with a slave ready signal

6.1.1.5 OR-ing of read data from multiple slaves

SBI is an OR-bus. That means that output data from each **slave** is OR-ed onto the same line. Each **slave** must always set the *rd_data* output low when its chip-select input signal is inactive.

In Figure 22 there are two **slaves** connected to one **master** via SBI. At period 1, **master** activates *cs_slave0* along with *rd_ena* and an address value. Note that only **slave 0** sets data on the *rd_data* output. At period 2 the data is sampled by the **master**, and *cs_slave0* is de-asserted, causing **slave 0** to set its *rd_data* output low.

At period 3 the **master** asserts *cs_slave1*, which means that only **slave 1** will output data on *rd_data* for the duration of the read operation.

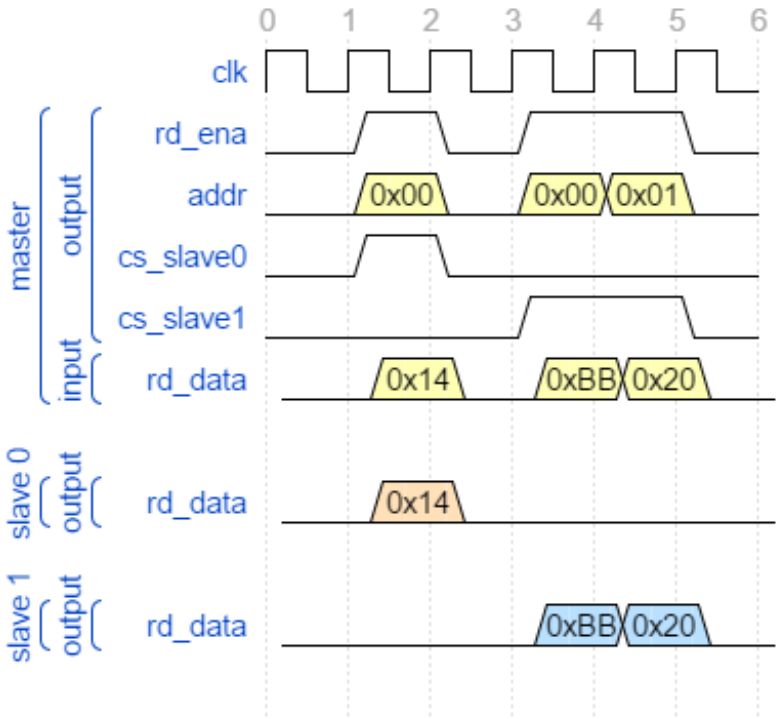

Figure 22 OR-ing of read data from multiple slaves