Hardware/Software Interface Management with Regtify

Joachim Knaeblein, Timo Schuering
http://www.eda-solutions.de

Abstract

Modern integrated systems are often composed of one or more processors and some associated integrated circuits. A processor communicates with the integrated devices via their hardware/software interface. This interface basically is formed by a number of registers incorporated in the circuit. The management of these registers and their properties in terms of specification, documentation, implementation and verification may be a tedious and error prone task. Regtify supports designers in coping with this situation in a highly customizable, efficient and flexible way.

1. Introduction

Many of today's integrated systems (IS) or system on chips (SoC) communicate with an embedded or external microprocessor. The microprocessor (MP) controls higher level functions of the system. Normally the over all function of an IS can be classified in a high speed part and a high sophisticated part. The high speed task is done by an integrated circuit whereas the high sophisticated function is realized in software on an MP. For the combination of the two participants a hardware/software interface is required.

This interface consists of registers inside the IS. These registers can be set to a value or can be read back via the MP interface. The values of the registers may have an impact on the operation of the IS or they reflect its internal state. Basically the MP to IS communication comprises three possible operations:

- Setting up the IS by writing particular values to registers
- Reading the internal state of the IS
- Reacting to a request of the IS for immediate action (interrupt)

From a high level system point of view registers can be regarded as variables which are set and read by different parts of the IS.

In modern systems the number of registers tends to grow rapidly. Currently several 10.000 registers are state-of-the-art. Of course this number varies from application to application. Like variables as known from software, registers have several characteristics, e.g. read/write capability and type. There are other properties which are based on the fact that we deal with real hardware like addresses, bit position and so on. In the end an IS can easily have around 20.000 register instances with around 10 attributes each. These attributes are partly software or hardware related.
The characteristics of the registers must be specified somewhere and their hardware representation must be realized, documented and verified. One can imagine that implementing 20.000 registers with 10 properties each is a very, very tedious job which everybody would like to avoid.

This is where Regtify comes into the game. It can reduce the effort for the HW/SW interface management tremendously. Regtify comes with a GUI which eases the specification pain by bringing automation and assistance for this task. The specification can be checked against user customizable rules. The tool creates synthesizable VHDL code. The VHDL generation is very flexible and can be tuned to the needs in a very wide range. It is possible to export the register data in hyperlinked PDF for documentation purposes. Finally software or verification drivers for the register map can be generated on a user customizable basis.

2. The Distributed Register Map Approach

It is common practice to decompose the overall function into sub-blocks in order to cope with the large complexity of today's designs. Following the distributed register map approach means to do so with the register set, too. Each functional sub-block gets a register block associated, containing the registers which are required for it.

Such an approach has several advantages:

- Every block designer can verify his/her block together with the registers which are relevant to it, this means reduced top-level register hassle
- The register block synthesis process is split in parts and distributed among the block designers.
- No layout hotspot due to one big central register block everybody on the chip talks to.
- Reduced clock transition problems, since local register blocks can be clocked by locally available block clocks.

See figure 1 for an illustration:

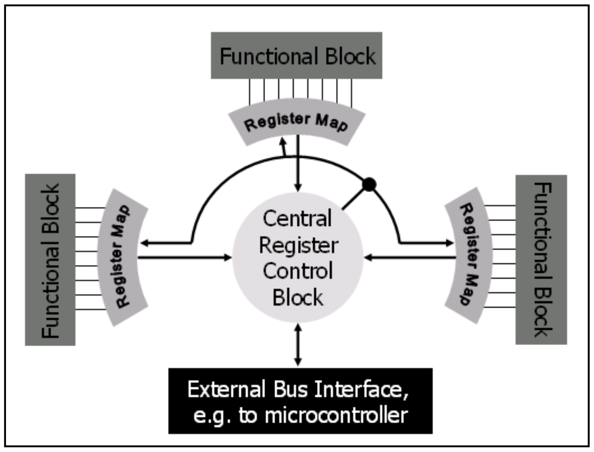


Figure 1: The distributed register map approach

Regtify fully supports this register map structure by providing the capability to assign block names to every object. All objects bearing the same block tag are grouped into one VHDL block. Grouping is *not* based on addressing or other attributes.

3. Regtify Features

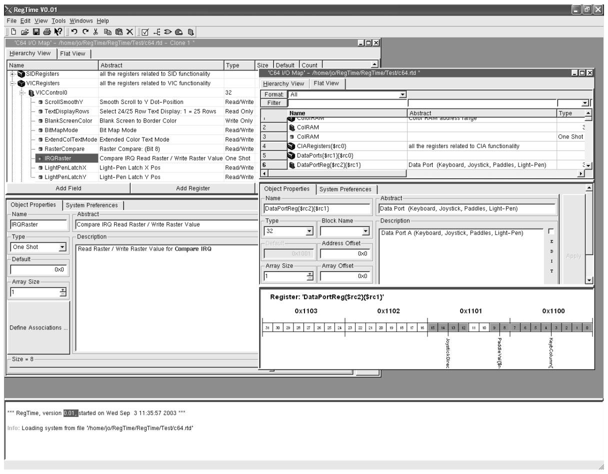
3.1 The Object Hierarchy Browser

Register are managed hierarchically similar to the way like files are organized in a file system. Groups may contain an arbitrary number of registers and sub groups. Registers, again, contain fields, which are the basic information units of the register map. The hierarchy browser displays this structure therefore in the same way like file system browsers do.

Since too much information can be more distracting than helpful, two levels of verbosity are available, the software and the hardware view. In the software view hardware related information like addresses, bit positions, etc. is hidden.

Beside this hierarchical view a tabular information access is possible, too. This table display is suitable to get an overlook over the register data. This is even more helpful since every object can have a repeat count value, i.e. it is instantiated multiple from one definition.

Figure 2 shows the cockpit of Regtify.



Name	Abstract	Type
SIDRegisters	all the registers related to SID functionality	
VICRegisters	all the registers related to VIC functionality	
VICControl0		32
ScrollSmoothY	Smooth Scroll to Y Dot-Position	Read/Write
TextDisplayRows	Select 24/25 Row Text Display: 1 = 25 Rows	Read Only
BlankScreenColor	Blank Screen to Border Color	Write Only
BitMapMode	Bit Map Mode	Read/Write
ExtendColTextMode	Extended Color Text Mode	Read/Write
RasterCompare	Raster Compare: (Bit 8)	Read/Write
IRQRaster	Compare IRQ Read Raster / Write Raster Value	One Shot
LightPenLatchX	Light-Pen Latch X Pos	Read/Write
LightPenLatchY	Light-Pen Latch Y Pos	Read/Write

Name: IRQRaster
Abstract: Compare IRQ Read Raster / Write Raster Value
Type: One Shot
Description: Read Raster / Write Raster Value for **Compare IRQ**
Default: 0x0

```
*** RegTime, version 0.01, started on Wed Sep  3 11:35:57 2003 ***
Info: Loading system from file '/home/jo/RegTime/RegTime/Test/c64.rtd'
```

Figure 2: The cockpit of Regtify

3.3 The Association Manager

Object associations result in internal connections between objects. These internal object connections are required for e.g. interrupt trees or enable signals. The definition of associations is done either by using a graphical aid or by algorithm. Especially the algorithmic approach is very powerful when a large number of connections must be specified. This programmable approach is applied in different contexts inside Regtify and therefore described separately in more detail in paragraph 4. For a glance on the association manager see figure 3.

3.4 The Rule Checker

Since the number of registers with their attributes can be overwhelming, a customizable rule checker is very useful. There may be rules which flag an error, a warning or a note. Rules may check conventions concerning naming, addressing, associations, etc. An arbitrary number of rules may be specified in a similar algorithmic way like associations. For more details on that, refer to paragraph 4.

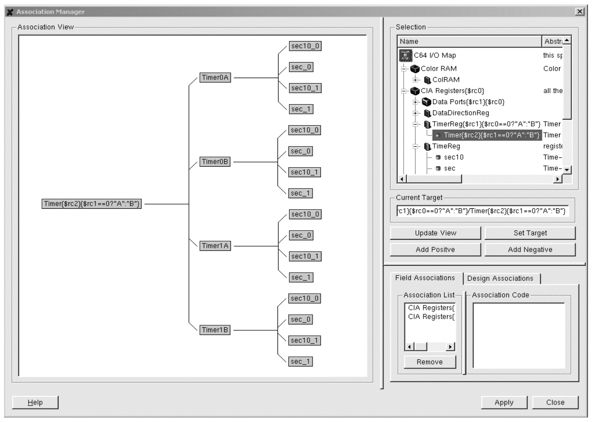


Figure 3: The association manager

3.6 The Documentation Generator

The documentation generator creates a PDF file of the specification in one out of three different formats. There is the …

- hyperlinked PDF format which reflects the hierarchical display
- tabular format which reflects the table view
- memory map to get an overlook over the address allocation

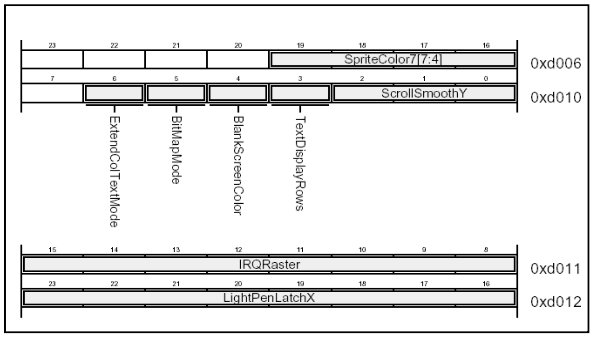


Figure 4: Sample memory map output

3.7 The Driver Generator

The driver generator is a very powerful tool. Although it carries driver in its name it is not only suitable to generate software and verification drivers for the given register specification, but also to write any specification related information to a file. See paragraph 4 for more details.

3.8 The HDL Generator

The HDL generator generates VHDL (87/93) code for register blocks, test benches, etc. according to the register specification.

3.8.1 Details of the VHDL Generation Process

The approach of Regtify for the generation of VHDL code is very flexible. It can easily be customized by the user in several ways. Regtify in sum has four inputs which it processes to create VHDL code in the end. These inputs are:

1. The register specification
2. The preference settings
3. A template library which is a set of VHDL code fragments containing placeholders for code
4. A component library which comprises a number of VHDL entities. These entities match the type entries in the Regtify specification

Regtify takes the four inputs and glues them together. The template is the skeleton where selected entities are integrated. The selection of the entities is based on the Regtify specification itself and the settings. Each entity comes with attributes at its ports. These attributes determine how the entity is connected to signals inside the templates and which ports are to be generated at the entity of the resulting register block.

Attributes are available for:

- Register block ports and generics
- Internal signals like clocks, resets, …
- Data bus connection
- Internal inter-object connections resulting from associations
- RAM block integration

The work flow of Regtify looks like this:

1. Collect all registers which are allocated to the block which is currently processed
2. For each selected register find an appropriate component definition in the user provided library
3. Analyze the attributes. Add ports and generics specific for this register to the register block entity. Connect other component ports to template signals according to the attached attributes.
4. Add associated registers to the block and connect them.
5. The resulting VHDL code is inserted into the template at the location of specific placeholders.
6. Write out the resulting file

Example:

```
ENTITY write_reg IS
 GENERIC (width:integer); --attribute: width
 PORT (
  pNAME  : OUT unsigned(0 TO width-1);-- attr:port
  clock  : IN  std_ulogic; --attr:internal
  dread  : IN unsigned(0 TO width-1); --attr:wrdata
  dwrite : OUT unsigned(0 TO width-1)--attr:rddata
 );
END ENTITY write_reg;
```

Instantiation code for this component for object ‘foo’:

```
u_foo: write_reg
GENERIC MAP ( width =>8)
PORT MAP (      pNAME => pfoo,
                clock => clock,
                dread => dread_foo,
                dwrite => dwrite_foo
);
```

The example has been strongly simplified. It is not supposed to be realistic. To understand it fully some remarks are necessary:

- Trailing comments ‘attr:foobar’ indicate the function of this entity port and how to connect it
- ‘NAME’ in the port name will be replaced by the objects name when being connected to a port/signal
- In the instantiation width of 8 is derived from the width of object ‘foo’. Right-sided ‘pfoo’ in the port map is a port at the entity of the generated register block, signal ‘clock’ on the right side of the assignment is a predefined signal in the template, ‘datread_foo’ and ‘datwrite_foo’ are the lines to the address decoder/multiplexer

3.8.2. RAM Integration

It is possible to integrate RAM blocks in the generated code. From the Regtify point of view a RAM block is a component for a set of objects.

3.9. The Entire Regtify Workflow

When working with Regtify a typical workflow could look like shown in figure 5. The user specification is checked for consistency and completeness with the help of the rule checker. The verified database is fed into the VHDL generator to create VHDL code. Additionally drivers e.g. for the VHDL simulation environment and the software environment can be produced. Documentation in PDF format is available from the document generator. Regtify covers the whole area of HW/SW interface management from specification to hardware validation.

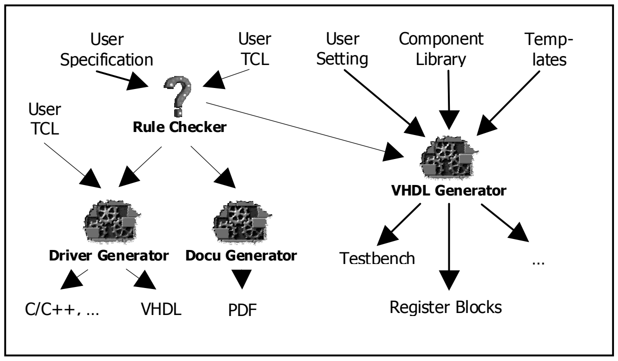


Figure 5: Overview over Regtify Workflow

4. Algorithmic Concept

The association manager, the rule checker and the driver generator make use of an algorithmic approach to provide more flexibility. All theses tools rely on a TCL engine, which is fed by the Regtify information on one side and user provided TCL code on the other.

The register information passes the user code object by object. The users's program fragment does the job …

- For the association manager it checks if the passing register is to be associated with the target register
- For the rule checker it verifies if the passing object complies to the rule
- For the driver generator it writes a piece of text (e.g. code snippet, object information in a certain format, …) to a specific file

Example (rule check for unique field names):

```
set viol 0
for {set i 0} {$i < $field::nr} {incr i} {
  if { [info exists nd($field::name($i))]} {
    RTWarning  "Field name $field::name($i) \
            already defined in $field::path($i)"
    incr viol
  }
  set nd($field::name($i)) $field::path($i)
}

return $viol
```

This example needs some explanation:

For each register field we count the violations and return them to the rule checker. A violation in this case is the fact that a field name is used multiple times. All field names are stored to an associative array. When a new field name is passed it is checked whether this field name has been used before. If so, a warning is issued and the violation counter increased.

5. Conclusion

The Regtify approach provides a very customizable, efficient and flexible way for HW/SW interface management. Especially the code generation of register blocks can be affected in a wide range. There are four degrees of freedom which the user can tune.

- The register specification
- The component library
- The template set
- The TCL coding for the algorithmic concept

Deploying Regtify yields a number of benefits:

- Facilitation of register data capturing, documentation, implementation and verification
- No consistency problem between specification, documentation, implementation and verification due to single source
- Exchange of register map data intra/inter-enterprise-wide, the same register specification can be implemented differently by using other templates/component libraries
- Uniform register map approach enterprise-wide is promoted by the use of Regtify
- Re-use from register specification down to register implementation
- Distributed register map approach is supported bringing advantages like mentioned before
- Integration of RAM blocks in register blocks is supported
- Open data representation using XML, can be easily processed by common (scripting) languages like Perl, TCL, Python or C
- Simplicity of register definition allows generous addition of registers for test/debug purposes
- Possibility of fast architecture modification enables the user to try out different approaches
- Instantiation of special debug register map blocks to support design verification. The register map normally is the information center of a design

Contact

Regtify is marketed under the brand name Recardis and available via

INGENIEURBÜRO SCHÜRING
IC-DESIGN CONSULTING - EDA · ENGINEERING
FROMMANNSTR. 3
D-90419 NÜRNBERG
GERMANY
www.eda-solutions.de
Phone: ++ 49-911-300-1554, Fax.: ++49-911-501659

References

[1] Arm Limited, AMBA Specification (Rev 2.0)
[2] IEEE, IEEE Standard VHDL Reference Manual
[3] John K. Ousterhout, Tcl and the Tk Toolkit
[4] AHO/Sethi/Ullman Compilerbau