Register Management with CSRCompiler

An Introduction

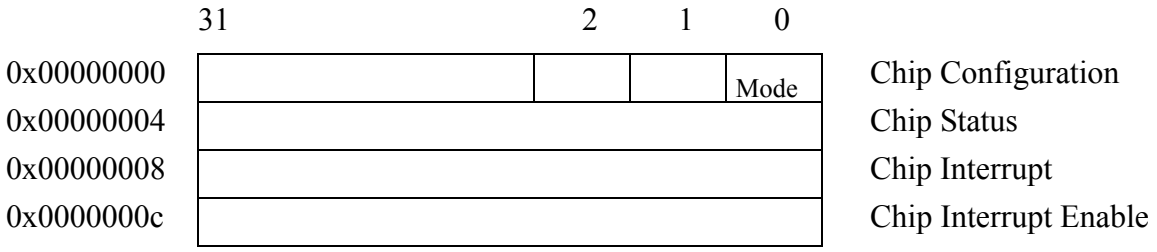
145 Eldora Drive
Mountain View, CA 94041
(650) 960-0925
http://www.semifore.com

Version: 1.7
July 2009

1 An Introduction

The CSRSpec language and the CSRCompiler together form an information processing system to manage the register specification of a design. The high level architectural view of the register information is the address map. The address map specifies the address of software accessible registers in a design. The address map also includes information about the position of fields within the registers. The complete register specification also includes implementation information about the function of the fields and registers and any side-effects due to software accesses. A diagram of an address map is given in Figure 1. “Address Map” page 1

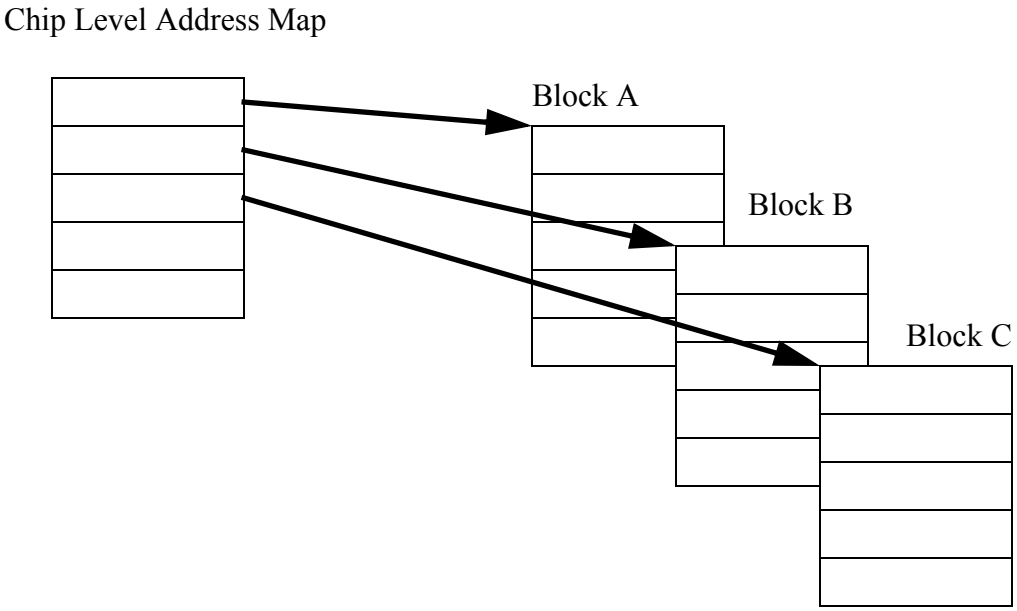
Figure 1. Address Map



The address map can also be hierarchical. One addressmap can include several other address maps. In an implementation with several blocks, each block can have its own address map. These block level address maps can be included in a higher level hierarchical address map. An example of a hierarchical address map is given in Figure 2. “Hierarchical Address Map” page 2

The implementation of each address map includes an address decoder. The top level address map includes an address decoder that decodes the transaction to determine to which lower level address map will receive the transaction. The implementation of the lower level address map will decode the transaction to the specific register that is addressed by the transaction.

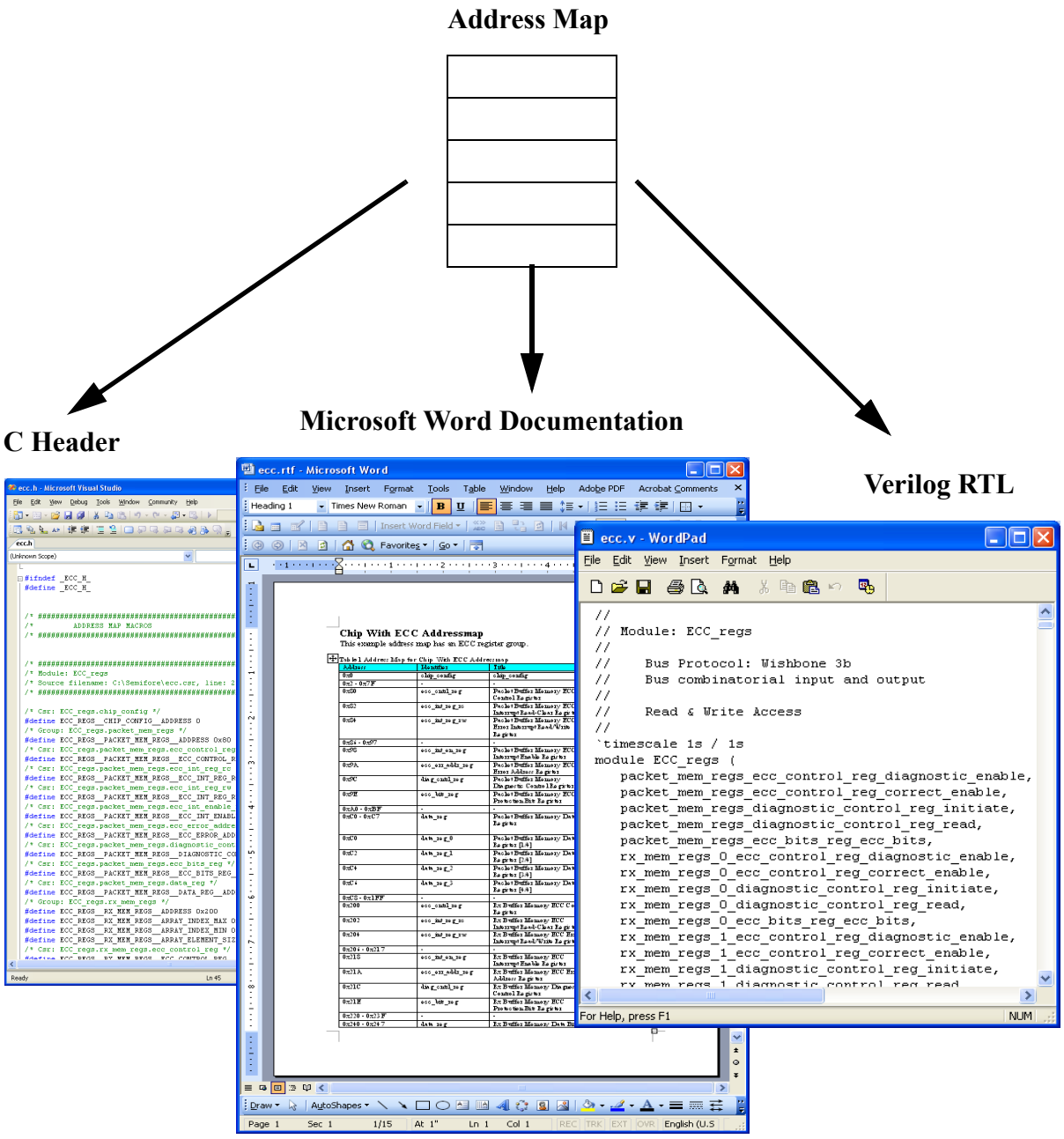
Figure 2. Hierarchical Address Map



1.1 Whole team uses the address map information

From a single source the entire design team has access to the address map information. The information is provided in many views. Each team member can access the information in the format required. The RTL designer is provided with the synthesizable RTL implementation of the addressmap including the bus interface with address decode, the registers, and the fields. The design verification engineer is provided with header files or classes used to access the fields in a verification environment. The software engineer is provided with the header files and hardware abstraction layer to provide driver access to the registers. The technical writing team is provided with the detailed reference document for internal and external distribution with tables for the addresses, register layout, and field function. The whole team benefits from automatically generated dynamic Web pages with the address map information. An example of the views provided is given in Figure 3. “Single Source For Multiple Views” page 3

Figure 3. Single Source For Multiple Views

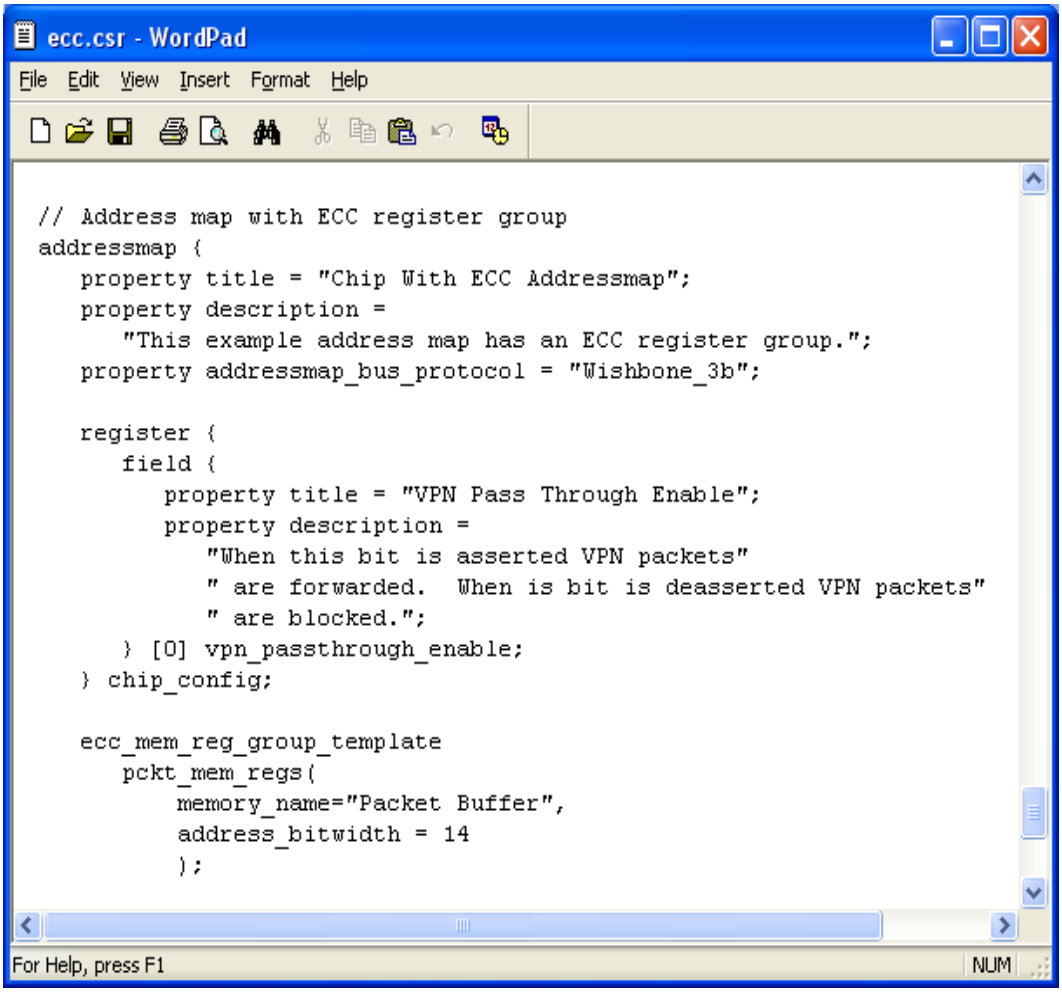


1.2 CSRCompiler Inputs

The CSRCompiler can accept several single source inputs to provide the required views. These inputs include CSRSpec, Spirit SystemRDL, Spirit IP-XACT XML, and others. This section will give a brief introduction to CSRSpec.

An example of the CSRSpec language is given in Figure 4. “Example CSRSpec File” page 4. The CSRSpec language allows a designer to specify the architecture in a text file. This input method has the power to handle very large and complex designs. The language allows the designer to create architectural templates to support design reuse with the project and among many other projects.

Figure 4. Example CSRSpec File



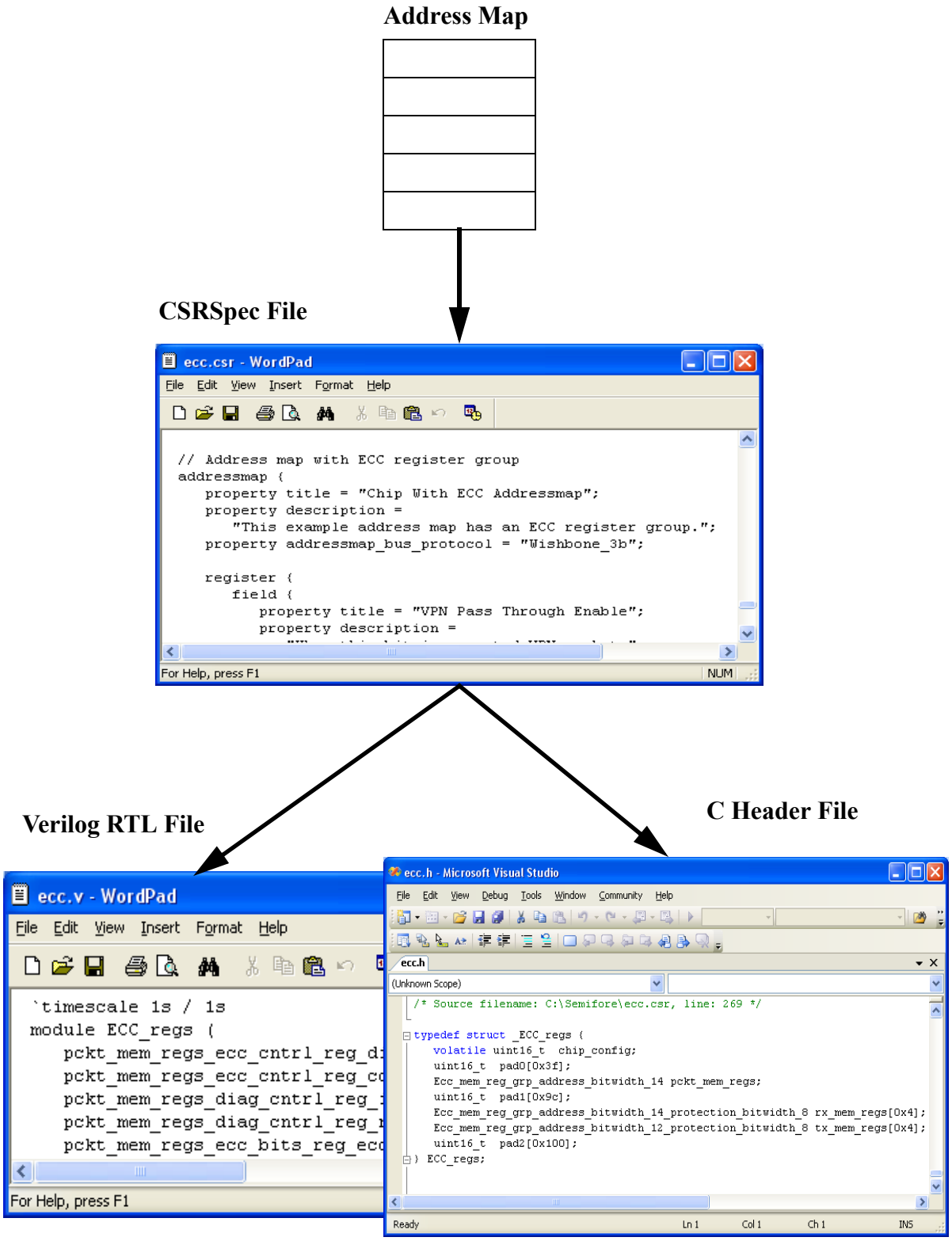
2 CSRSpec Introduction

The CSRSpec language provides a single source for specifying the address map architecture of a design. The language supports a data model that includes hierarchical address maps, registers, and fields. Each object can also have a set of properties that document or specify the behavior of the object. CSRSpec has a provision for object templates to provide consistent, reusable, and parameterizable architectural definitions.

2.1 CSRSpec Addressmap Object

The CSRSpec language provides a container object for address maps. This container is specified with the keyword addressmap. The implementation of an address map contains the bus interface, address decode, registers, and fields. The RTL implementation of an address map is a Verilog module or a VHDL entity. An address map is represented in a C header file as text macros for the address and offsets for its members and as a struct. An example of a CSRSpec addressmap is given in Figure 5. “CSRSpec Addressmap Object” page 6

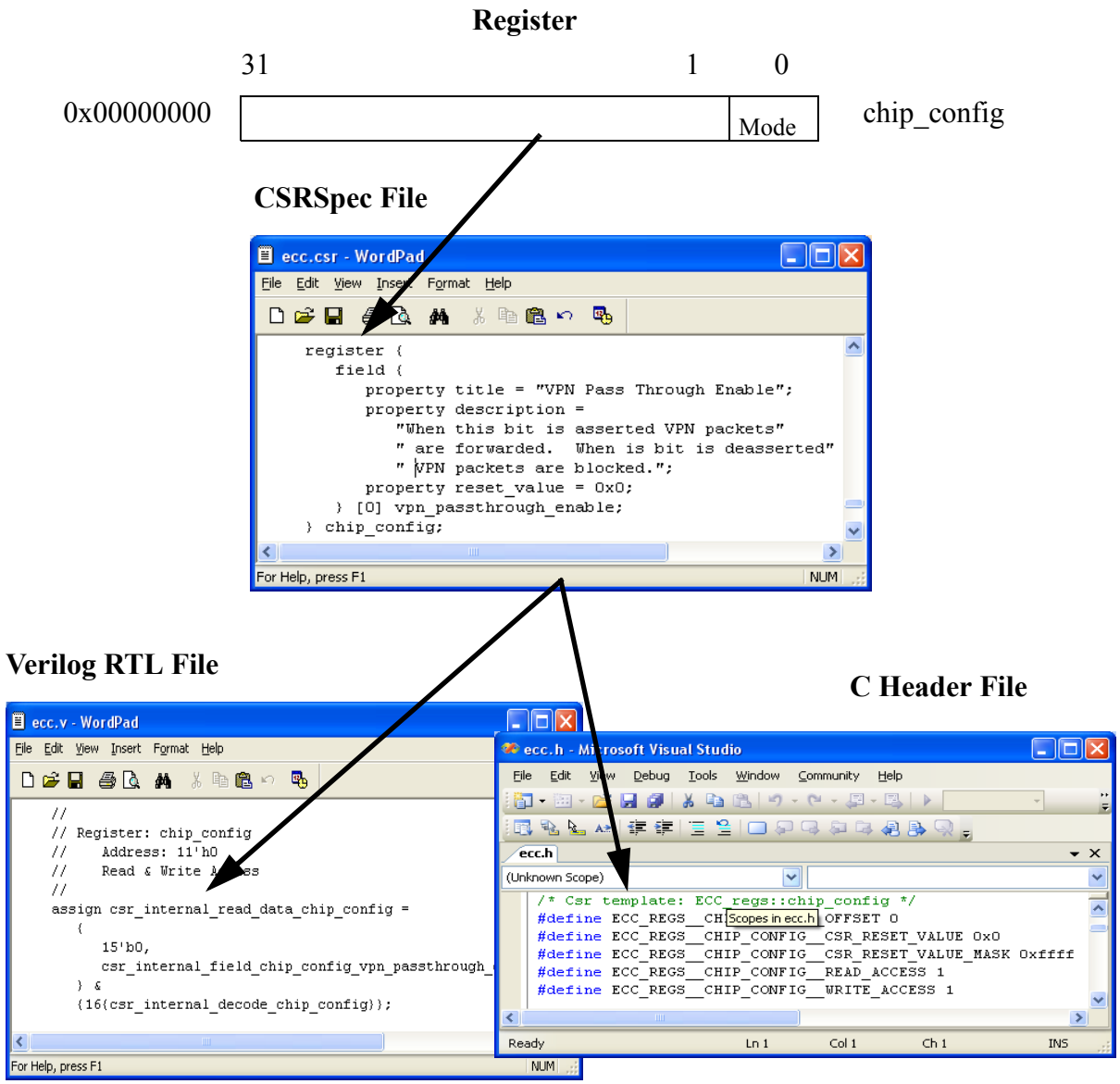
Figure 5. CSRSpec Addressmap Object



2.2 CSRSpec Register Object

The CSRSpec language provides the register container object for registers. A register has an address and contains fields. A register can be accessed in one transaction by software. The RTL implementation of a register is a concatenation of the fields with zeros provided for undefined positions. A register is represented in a C Header file as a set of text macros providing the reset values and as a member of a struct. An example of a register is given in Figure 6. “CSRSpec Register Object” page 7.

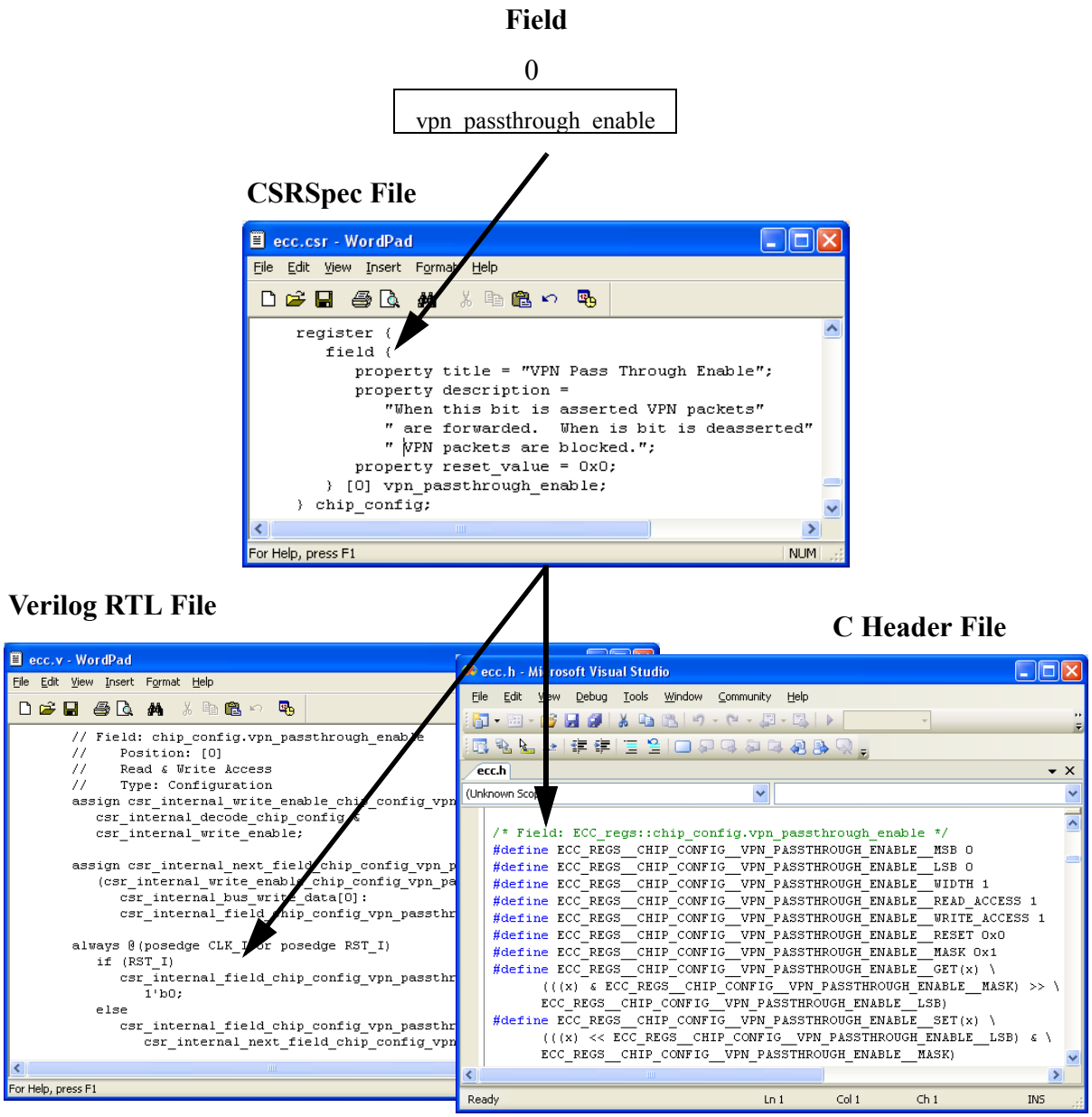
Figure 6. CSRSpec Register Object



2.3 CSRSpec Field Object

The CSRSpec language provides the field object to represent a register field. A field is a single bit or a consecutive group of bits that provide a single function. A field has a position within a register. The RTL implementation of a field depends on the specific architecture of the field and any optional properties that may affect the function of the field. A field is represented in a C header file as a group of text macros that provides the position, reset value, masks, and access macros. An optional representation is a bitfield declaration in the struct for the register. An example of a CSRSpec field is given in Figure 7. "CSRSpec Field Object" page 8.

Figure 7. CSRSpec Field Object



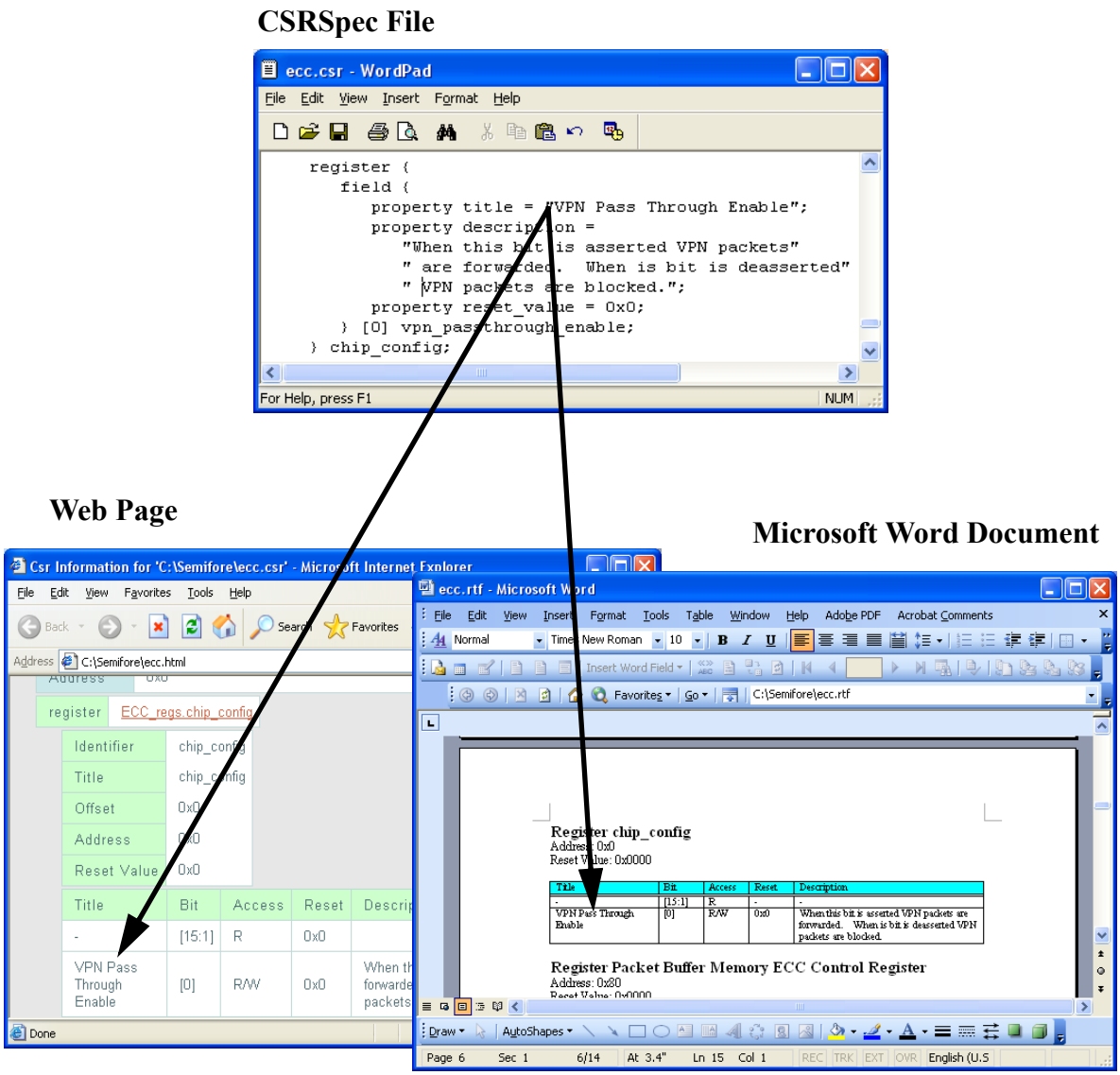
2.4 CSRSpec Properties

The CSRSpec language provides properties for the objects. Some basic properties are documentation title and description and field or register reset value.

2.4.1 CSRSpec title Property

In the CSRSpec language, each object can have a title for documentation. This property is specified in the CSRSpec input file and propagates to the automatically generated web pages and documentation files. An example of a title property is given in Figure 8. “CSRSpec title Property” page 9.

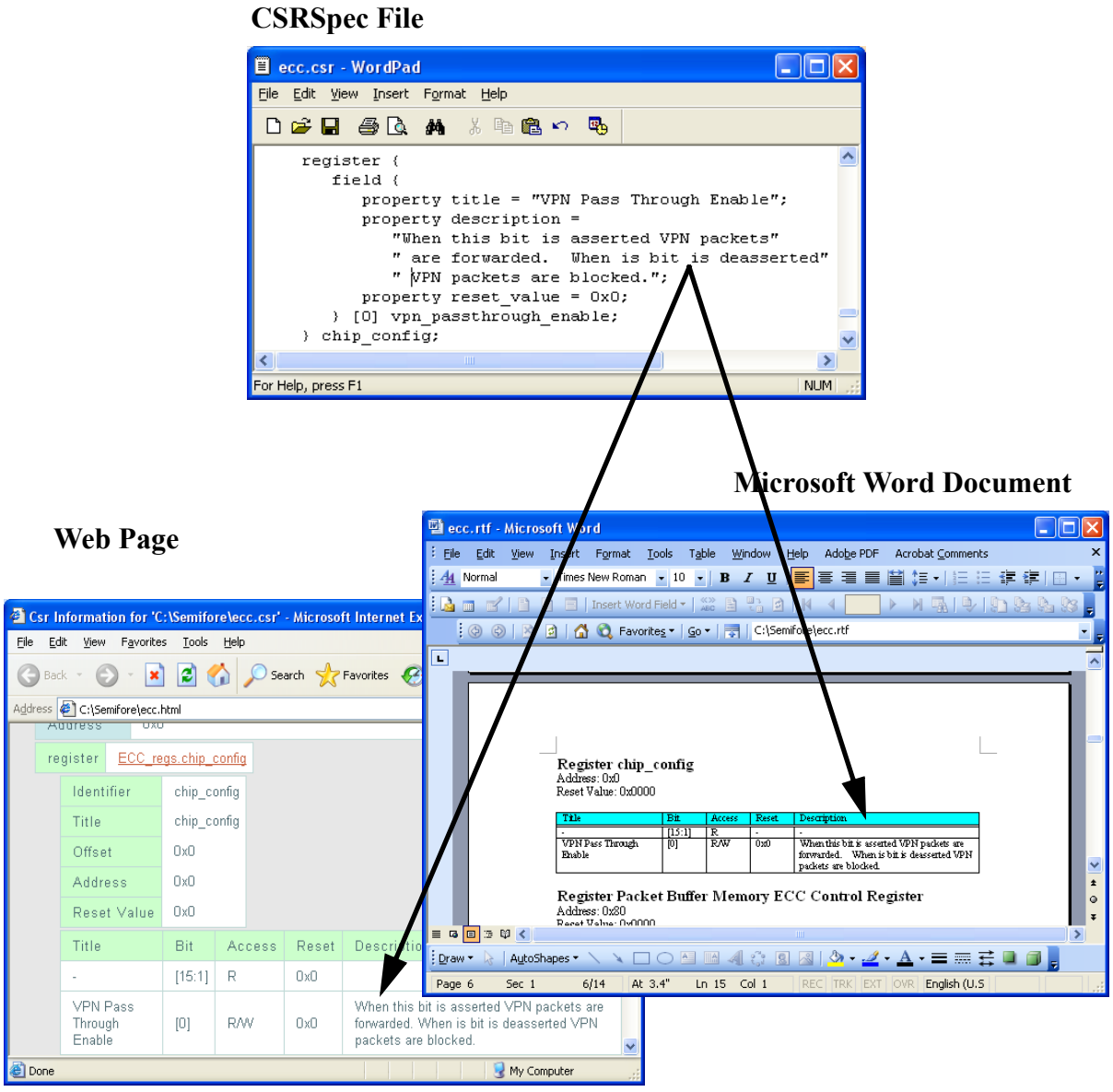
Figure 8. CSRSpec title Property



2.4.2 CSRSpec description Property

The CSRSpec language provides a description property for all objects. The description propagates out to the automatically generated web pages and documentation files. Two characters have special meaning in a description. A newline character is converted into a paragraph break. A tab character remains tab and follows the tab settings of the output file. An example of the description property is given in Figure 9. “CSRSpec description Property” page 10.

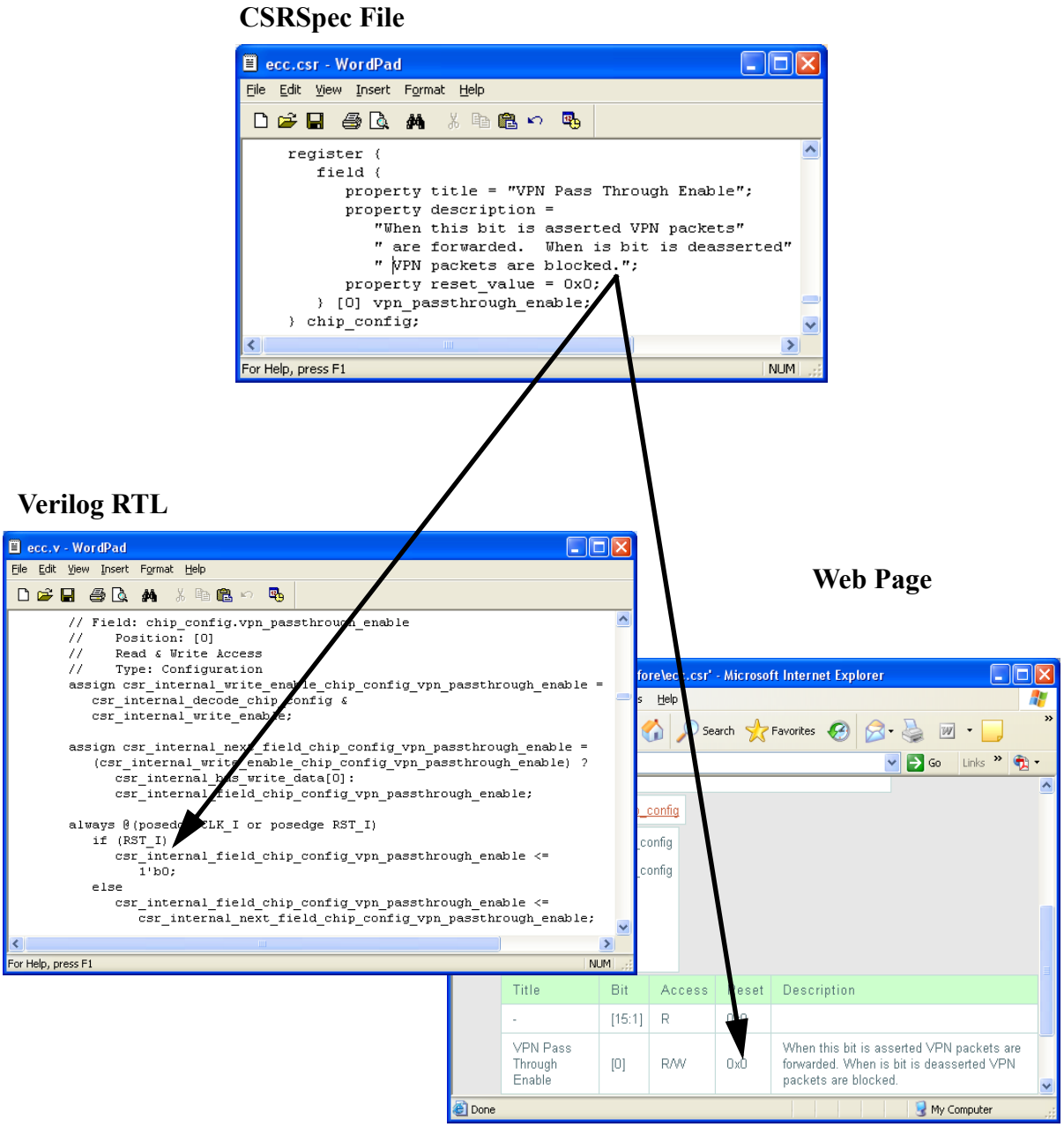
Figure 9. CSRSpec description Property



2.4.3 CSRSpec reset_value Property

The CSRSpec language provides field and register object with a reset_value property. In the implementation the reset value becomes the value forced into the field when the reset is asserted. The reset value is also propagated to the header files, automatically generated web pages, and documentation files. When a reset_value is specified for a register object, the fields of the register inherit the appropriate bits of the reset_value. An example of the reset_value property is given in Figure 10. “CSRSpec reset_value Property” page 11.

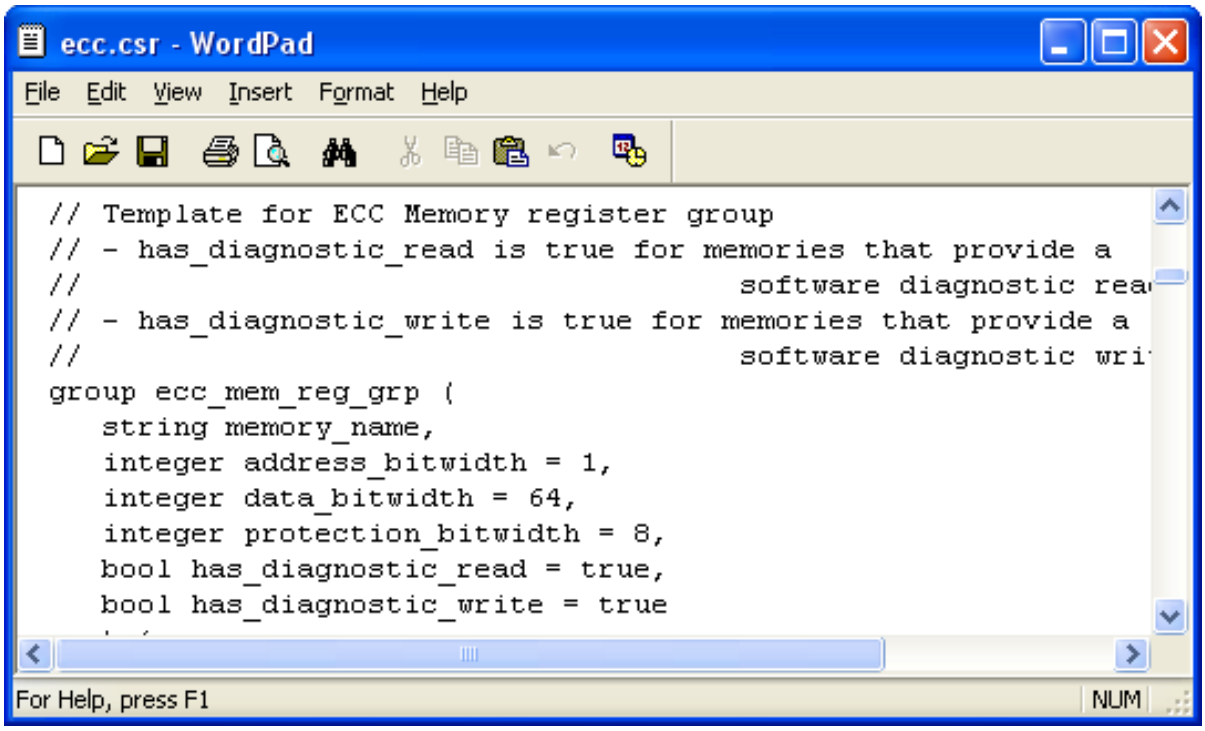
Figure 10. CSRSpec reset_value Property



2.5 CSRSpec Templates

The CSRSpec languages provides object templates. These templates provide a way to reuse an architectural definition in a parameterizable way. There can be templates for addressmaps, registers, and fields. Templates provide a way to have a consistent set of architectures throughout the design. When templates are used, the software team will see a more consistent view of the firmware interface to the hardware, and, they will be able to leverage the software across the design. An example of a template for a group object is given in Figure 11. "CSRSpec Template Example" page 12. A group is a collection of related registers or other smaller groups.

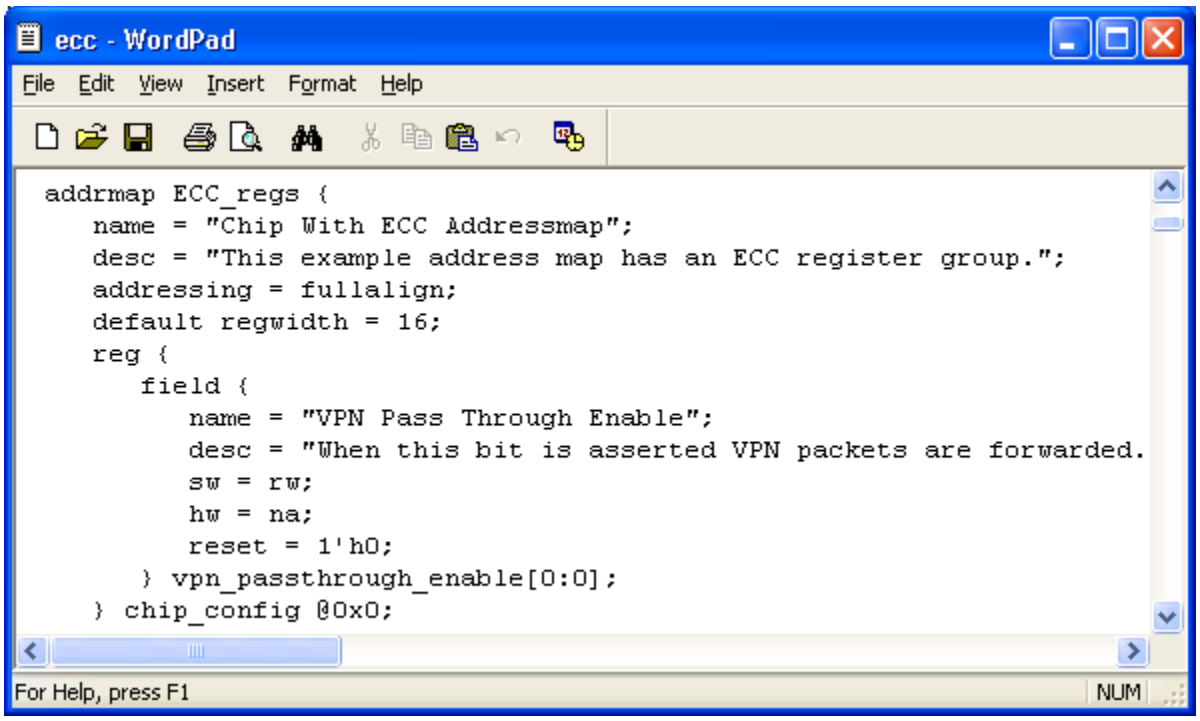
Figure 11. CSRSpec Template Example



2.6 Spirit SystemRDL Input

The CSRCompiler can read and write Spirit SystemRDL files. The set of objects represented by SystemRDL is very similar to CSRSpec. The SystemRDL addrmap, regfile, reg, and field objects map to CSRSpec addressmaps, groups, registers, and fields. An example of a SystemRDL file is given in Figure 12. "Spirit SystemRDL Example" page 13.

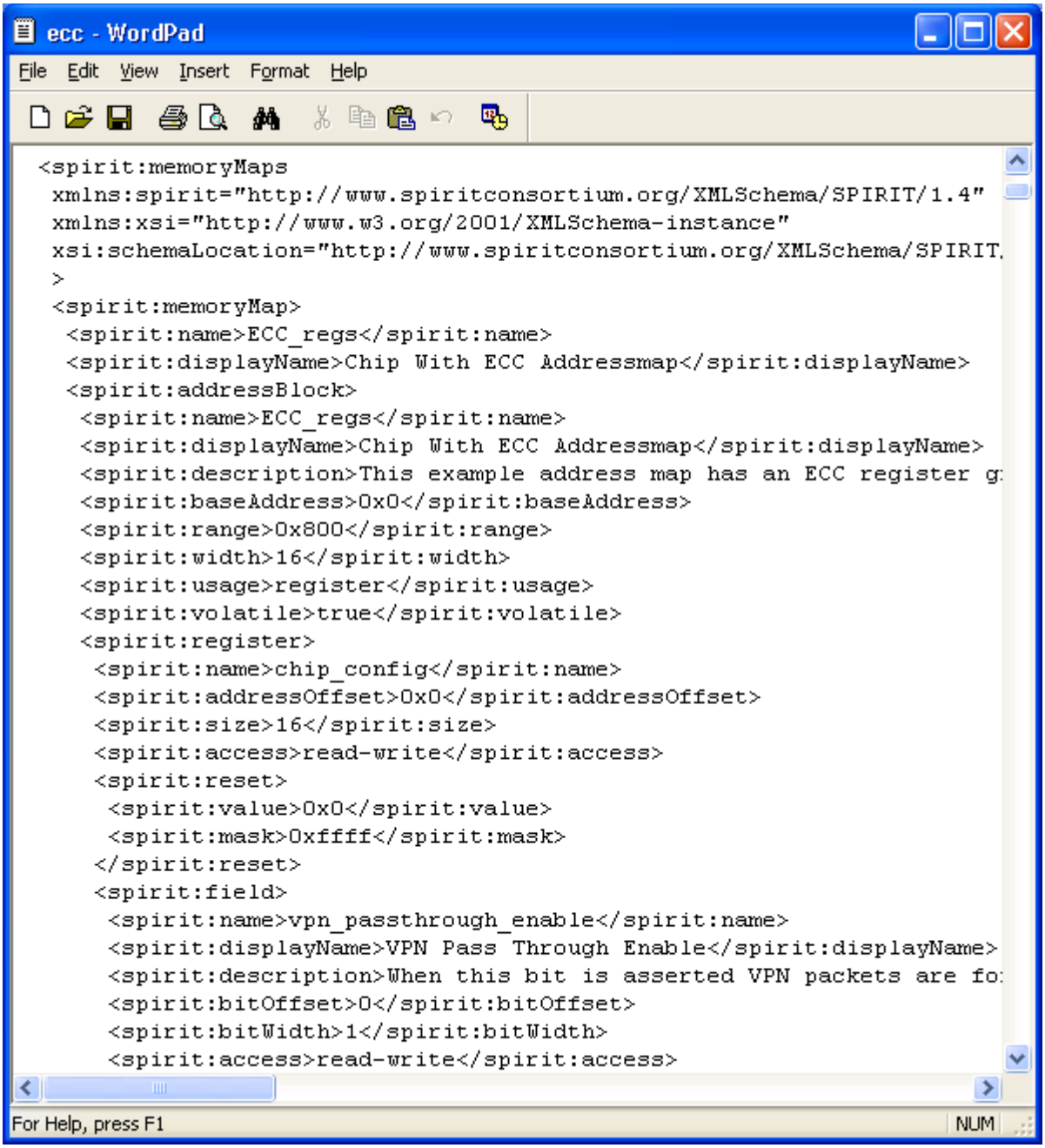
Figure 12. Spirit SystemRDL Example



2.7 Spirit IP-XACT XML Input

The CSRCompiler can read and write Spirit IP-XACT XML files. IP-XACT provides a wealth of information for IP cores. Included in the IP-XACT file are tags for register information. IP-XACT addressblock, register, and field map to CSRSpec addressmap, register, and field. An example of a IP-XACT file is given in Figure 12. “Spirit SystemRDL Example” page 13.

Figure 13. Spirit IP-XACT Example



2.8 Speadsheet Input

The CSRCompiler can read a Comma Separated Value, CSV, format file from common spreadsheet applications like Microsoft Excel. Each row of the spreadsheet represents one object in the register architecture, and, each column represents a property of the object. The first row of the spreadsheet contains the titles of the columns. The column title specifies which property is con-

tained in the column. The columns may be in any order. An example of a spreadsheet to define the register architecture is given in Figure 14. “Spreadsheet Input example” page 15

Figure 14. Spreadsheet Input example

Microsoft Excel - ecclegacy5.csv

File Edit View Insert Format Tools Data Window Help Adobe PDF

A1 Address

	A	B	C	D	E	F	G	
1	Address	Position	Title	Identifier	Type	Access	Reset Value	Description
2			ECC_legacy	ecc_legacy	addressmap			
3	0x0		chip_config	chip_config	register			
4		[0]		vpn_passthrough	configuration	R/W	0x0	When this
5	0x80		Packet Buffer	ecc_cntrl_reg	register			This registe
6		[1]		diag_en	configuration	R/W	0x0	This bit ena
7		[0]		correct_en	configuration	R/W	0x0	This bit ena
8	0x82		Packet Buffer	ecc_int_reg	register			This registe
9		[1]		sbe_int	interrupt	RC/W		Set when a
10		[0]		mbe_int	interrupt	RC/W		Set when a
11	0x98		Packet Buffer	ecc_int_en_reg	register			This registe
12		[1]		sbe_int_en	configuration	R/W		Enables sir
13		[0]		mbe_int_en	configuration	R/W		Enables mu
14	0x9a		Packet Buffer	ecc_err_addr_reg	register			This registe
15		[13:0]		address	configuration	R/W		This field c
16	0x9c		Packet Buffer	diag_cntrl_reg	register			This registe
17		[15]		initiate	configuration	R/W		When this
18		[14]		read	configuration	R/W		If this bit is
19		[13:0]		diagnostic_addre	configuration	R/W		
20	0x9e		Packet Buffer	ecc_bits_reg	register			This registe
21		[7:0]		ecc_bits	configuration	R/W	0x00	This registe

ecclegacy5

Ready NUM

2.9 RTL Implementation

The CSRSpec language provides enough detail to create a synthesizable RTL implementation of the addressmap. The CSRCompiler can generate both Verilog and VHDL versions of the RTL implementation. The RTL implementation for an addressmap includes the software transaction bus slave interface, one-hot address decode, field read and write controls, field storage elements, field ports, field hardware function, field software semantics, read multiplexer, and error detection. The details of the implementation are controlled by the property values in the CSRSpec file. A diagram of the RTL implementation is given in Figure 15. “CSRSpec RTL Implementation Diagram” page 16.

Figure 15. CSRSpec RTL Implementation Diagram