PC to SRAM Interface

School of Computer Science and Electrical Engineering
University of Queensland, Brisbane, Australia.
http://www.csee.uq.edu.au/

THE UNIVERSITY OF QUEENSLAND

Last Modified: 23 February 2001

Contents

1.0 About this design

This design provides an interface for a PC to access the SRAM on the XSV board through the board’s parallel port connection. When this interface is programmed into the Virtex FPGA, a connected PC can read from and write to any location in one bank of the SRAM. This document describes two designs that implement this PC-to-SRAM interface – a stand-alone design and a VHDL module.

The author would like to acknowledge a similar design for the XS40 board (also from XESS Corp.) developed by Universidad de Sevilla. Their design was a source of ideas for the design presented in this document for the XSV board. See http://www.xess.com/projects/xs_pc_intfc.pdf and http://www.xess.com/projects/xs_pc_intfc.zip for the Universidad de Sevilla design.

2.0 The stand-alone design vs. the VHDL module

2.1 The stand-alone PC-to-SRAM interface

This design contains a top-level entity and can be directly implemented on the Virtex FPGA. When the FPGA is configured with this design, a PC has complete read and write access (through a parallel cable) to the *left* bank of SRAM on the XSV board.

Note that the PC-to-SRAM interface can be used to give access to either bank of SRAM. However, the bank used must be specified in the top-level entity and UCF. Currently the top-level and UCF together specify that PC is given access to the *left* bank of SRAM.

Once the stand-alone design has been implemented and its bit file generated, this bit file can be used at any point as a debugging tool. Suppose that you are developing a design for the XSV board and at some point wish to view the contents of one of the SRAM banks. Simply configure the FPGA with the bit file of this stand-alone PC-to-SRAM interface. The content of the SRAM is unaffected by the configuration process. Once the interface is programmed into the FPGA, a utility programme on the PC can be used to view (and even change) the SRAM contents. A Visual Basic programme that provides these features, "XSVSRAMUtility.exe" is included with this resource collection. See the document called "XSV SRAM PC Utility", which describes the programme.

2.2 The PC-to-SRAM interface VHDL module

In addition to the stand-alone version of the PC-to-SRAM interface, there is a reusable VHDL module that provides the same functionality. This module is called "sraminterfacewithpport" and is defined in the file "sraminterfacewithpport-sv01.vhd". Note that this module itself contains other internal modules. As a consequence, several VHDL files need to be added to a user design when the "sraminterfacewithpport" module is going to be used. Note that "sraminterfacewithpport" is referred to in this document as the "top-level" of the module, but it is *not* a top-level entity in the true sense of the word. This VHDL entity is simply the highest-level entity *of the module*.

To understand this module, you first need to be familiar with the SRAM interface module (the VHDL entity called "sraminterface") in this resource collection. The SRAM interface provides a user design with a simple interface to a bank of SRAM. The module is described in the document called "SRAM Interface".

The *PC-to-SRAM* interface VHDL module is designed as a *replacement* for the (plain) SRAM interface VHDL module. Wherever a user design uses the SRAM interface module, it can instead use the PC-to-SRAM interface module. The PC-to-SRAM interface module provides the same direct access capabilities to SRAM that the SRAM interface module provides. Additionally, however, the PC-to-SRAM interface module also allows a PC to access the SRAM.

Using the PC-to-SRAM interface *module*, rather than the stand-alone design, makes viewing (and changing) the SRAM contents from a PC a much simpler process. Suppose a user design has correctly embedded the PC-to-SRAM interface module. The user design's bit file has been generated and used to configure the Virtex FPGA. When the time comes to view the contents of an SRAM bank, no new bit file needs to be downloaded to the FPGA, because the user-design's bit file already includes the circuitry necessary to communicate with the PC. A utility programme on the PC can be used straight away to view and change the contents of SRAM.

Thus the advantage of the PC-to-SRAM interface module over the stand-alone design is that no new bit files need to be downloaded once the module is embedded in the user design.

3.0 Files needed for the stand-alone design

3.1 List of Files

- pctoleftsramtoplevel-sv02.vhd (Contains the top-level entity “pctosramtoplevel”)
- pctosraminterface-sv06.vhd
- sram512kleft16bit50mhzreadreq-sv05.vhd (or sram512kleft16bit50mhzreadreq-sv05c.vhd)
- UCF is :"Y pport, Y debug, Y timing, Y IOB outputs, N input delay, Y fast slew.ucf"

The “svXX” suffix on the filenames stands for “source version XX”. Previous versions of these files are obsolete and have not been released.

3.2 File Descriptions

pctoleftsramtoplevel-sv02.vhd

This file contains the top-level entity for the design, which is called “pctosramtoplevel”. This entity connects the PC-to-SRAM interface to the left bank of XSV SRAM.

pctosraminterface-sv06.vhd

This VHDL module is responsible for implementing the communication protocol that is used between the PC and the XSV board. It translates requests from the PC into signals for the SRAM interface module and vice-versa.

sram512kleft16bit50mhzreadreq-sv05.vhd

This is an SRAM interface module, which provides the actual access to the SRAM bank. For more information on this module, see the document describing the “SRAM Interface” design.

The SRAM interface in this file uses a 2 clock cycle read. If you wish to use a 1 clock cycle read then the file “sram512kleft16bit50mhzreadreq-sv05c.vhd” can be used in place of this file.

4.0 Files needed for the VHDL module

4.1 List of Files

- sraminterfacewithpport-sv01.vhd (This is the top-level (or highest-level) of the VHDL module).
- sram512kleft16bit50mhzreadreq-sv05.vhd (or sram512kleft16bit50mhzreadreq-sv05c.vhd)
- memorymultiplexor-sv01.vhd
- pctosraminterface-sv06.vhd

4.2 File Descriptions

sraminterfacewithpport-sv01.vhd

This file contains the actual PC-to-SRAM interface VHDL module called “sraminterfacewithpport”. Although this entity is the highest-level entity of the module, it internally uses other VHDL modules. That is why the other files below must also be added to any user design that uses the PC-to-SRAM interface module.

sram512kleft16bit50mhzreadreq-sv05.vhd

The PC-to-SRAM interface module internally uses the SRAM interface module in this file to provide the direct SRAM access capabilities. See section 2.0, “The stand-alone design vs. the VHDL module” above.

The SRAM interface in this file uses a 2 clock cycle read. If you wish to use a 1 clock cycle read then the file "sram512kleft16bit50mhzreadreq-sv05c.vhd" can be used in place of this file.

memorymultiplexor-sv01.vhd

The VHDL entity "memorymultiplexor" in this file multiplexes the connections to the SRAM interface module. At any point in time, the SRAM interface is either to be connected to the user design (allowing the design normal SRAM access) or to the "pctosraminterface" entity (allowing a PC access to the SRAM). This VHDL file takes care of switching these connections as necessary.

pctosraminterface-sv06.vhd

This VHDL module is responsible for implementing the communication protocol that is used between the PC and the XSV board. It translates requests from the PC into signals for the SRAM interface module and vice-versa.

5.0 CPLD requirements for the PC-to-SRAM interface

The PC-to-SRAM interface design requires the Virtex FPGA to connect to the XSV board's parallel port connector. The XSV board is such that the FPGA is able to connect to the parallel port connector only through the CPLD. That is, the CPLD must be configured with a design that connects some of the FPGA pins, through the CPLD, to the pins of the parallel port connector.

For the stand-alone PC-to-SRAM interface FPGA design, the CLPD must be configured so that the following connections are made between the Virtex FPGA pins and the pins of the parallel port connector:

This Virtex FPGA pin…	**…must connect (through the CPLD) to this parallel port connector pin:**
P177	D0
P167	D1
P163	D2
P156	D3
P145	D4
P138	D5
P134	D6
P124	D7
P132	S3
P133	S4
P139	S5
P141	S6

For the PC-to-SRAM interface module, technically different Virtex FPGA pins could be specified, provided that a corresponding change was made to the CPLD design. However, for simplicity, it is recommended that the same Virtex FPGA pins listed in the above table be used.

The following are examples of CPLD designs that connect the pins in the manner listed in the above table:

- The default SVF file for the CPLD that comes with the XSV board, called "dwnldpar.svf". Configuring the CPLD with this SVF file allows the PC-to-SRAM interface to be used.
- The CPLD design associated with the IP Stack FPGA design in this collection. The source file for this design is called "cpldnet.vhd". The UCF for the CPLD for this design is called "cpldpins.vhd". Configuring the CPLD with this design will also allow the PC-to-SRAM interface to be used.

Any other CPLD design can be used, providing it makes the pin connections listed in the table above.

6.0 Using the PC-to-SRAM interface module in a user design

While using the PC-to-SRAM interface stand-alone design is relatively straight forward, using the module version of the design is a little more involved.

The PC-to-SRAM interface module is a VHDL entity called “sraminterfacewithpport”.

6.1 Module ports

The “sraminterfacewithpport” entity includes all the ports found in the “sraminterface” entity. These ports are :

Inputs:	**Outputs:**	**Birdirectional:**
CLK	readData(15:0)	SRAMLeftData(15:0)
Resetn	canRead	
doRead	canWrite	
doWrite	CELeftn	
writeAddr(18:0)	OELeftn	
writeData(15:0)	WELeftn	
readAddr(18:0)	SRAMLeftAddr(18:0)	

These ports are used by “sraminterfacewithpport” in exactly the same way as for the “sraminterface” entity, as described in the “SRAM Interface” document.

The “sraminterfacewithpport” entity contains **two extra ports in addition to the above ports**. These two extra ports are as follows:

Port name:	**Direction:**	**Description:**
ppdata(7:0)	Input	Must be connected to the data pins of the parallel port connector.
ppstatus(6:3)	Output	Must be connected to the status pins of the parallel port connector.

6.2 Requirements for using the module

Using the PC-to-SRAM interface module (i.e. the “sraminterfacewithpport” entity) in a user design simply involves meeting the following requirements in regard to the module’s ports.

The ports that are identical to the SRAM interface ports (i.e. the ports of the “sraminterface” entity) are to be connected in exactly the same way as they are for that interface. These connections are described in the “SRAM Interface” document.

The two ports “ppdata” and “ppstatus” must have their connections directly propagated all the way up to the top-level of the user design, as follows:

- Bits 7 to 0 of the “ppdata” input must be connected to the FPGA pins that correspond to pins D7 to D0 respectively of the parallel port connector on the XSV board.
- Bits 6 to 3 of the “ppstatus” output must be connected to the FPGA pins that correspond to pins S6 to S3 respectively of the parallel port connector on the XSV board.

For an explanation of how to ensure that the correct connections are made from the FPGA, through the CPLD, to the parallel port connector, see section 5.0, “CPLD requirements for the PC-to-SRAM interface”.

6.3 Converting a design to use "sraminterfacewithpport" instead of "sraminterface"

Any user design that uses the "sraminterface" module to provide access to a bank of SRAM can be easily modified to use instead the "sraminterfacewithpport" module. Making this modification will allow the user design normal access to an SRAM bank, while also allowing a PC to access the same SRAM bank.

The following changes need to be made to an existing user design to change from the "sraminterface" module to the "sraminterfacewithpport" module:

- Add the extra necessary files to your design, as listed in section 4.0 "Files needed for the VHDL module".
- Change all declarations of an "sraminterface" component to declarations of an "sraminterfacewithpport" component. Make the same name changes to the corresponding component instantiations.
- Add the "ppdata" and "ppstatus" signals to the port listings of the above component declarations. Add the signals to the port mappings of the corresponding component instantiations.
- Add the "ppdata" and "ppstatus" signals to the ports of the user design's top-level entity.
- Add pin constraints to the design's UCF ensuring that the "ppdata" and "ppstatus" signals are connected to the correct FPGA pins, as listed in the table in section 5.0, "CPLD requirements for the PC-to-SRAM interface".

7.0 Communication protocol overview

This section provides an overview (from a PC programme's perspective) of the protocol used by the PC-to-SRAM interface. The information in this section and section 8.0 can be used to write a PC utility programme that communicates with the PC-to-SRAM interface to access the SRAM on the XSV board. (Note that a Visual Basic programme that does this, "XSVSRAMUtility.exe" is included with this resource collection. See the document called "XSV SRAM PC Utility", which describes the programme).

The PC-to-SRAM interface gives complete access to one bank of SRAM. The address and data size details are therefore as follows:

No. of addressable locations:	512 * 1024 = 524 288 = 512K (Addresses are 19 bits wide).
Width of data at each location:	16 bits (2 bytes)
Total capacity of one SRAM bank:	512K * 2 = 1024 KB = 1MB

7.1 Operations

The PC-to-SRAM interface allows a PC programme to carry out the following operations:

- **Set Address**
- **Write**
- **Read**
- **Apply Settings**

Set Address operation

This operation tells the PC-to-SRAM interface the address at which the next write or read will take place. The PC-to-SRAM stores a value for the "current" address. When the PC requests a write or a read, the

write or read is made at the location indicated by the "current" address. This operation is therefore used to set the current address.

The address is sent to the PC-to-SRAM interface in four pieces – known as slices. The first three slices sent are 5 bits wide and the final slice is 4 bits wide – for a total of 19 bits.

Additionally, after every write or read operation, the current address is incremented by 1. This means that writes or reads from consecutive locations only require the "set address" operation to be carried out at the start of the block.

Write operation

This operation tells the PC-to-SRAM interface to write a 16-bit value to the current address. The data to be written is sent in four slices. Each slice is 4-bits wide, for a total of 16 bits of data.

After the write occurs, the current address is incremented by 1.

Read operation

This operation tells the PC-to-SRAM interface to read a 16-bit value from the current address. The data read is sent from the PC-to-SRAM interface to the PC in four slices. Each slice is 4-bits wide, for a total of 16-bits of data.

After the read occurs, the current address is incremented by 1.

Apply Settings operation

The PC-to-SRAM interface has a useful "settings" feature. The interface is capable of storing several flags (also called settings) internally, whose value can be set by the PC to be either 0 or 1. The idea is that these flags affect the behaviour of the PC-to-SRAM interface in some way. In this manner, the PC can dynamically control aspects of the PC-to-SRAM interface's behaviour. The communication protocol currently allows for up to 5 of these flags.

Currently, only one such flag is implemented. This flag is called the "PC Control-Enabled" flag (or setting). The value of this flag makes no difference to the stand-alone version of the PC-to-SRAM interface, but it has an important effect on the VHDL module version of the interface, as explained in the following:

> When PC Control-Enabled is set to 1, the PC has exclusive access to the XSV SRAM via the PC-to-SRAM interface module. If the user design (in which the module is embedded) tries to access the SRAM at this time, its read and write requests will be ignored.
>
> When PC Control-Enabled is set to 0, the user design has exclusive access to the XSV SRAM via the PC-to-SRAM interface module. If the PC tries to access the SRAM at this time, the write and read operations will not be successfully carried out.

The "apply settings" operation is used to update the value of the flags. Currently the only flag that exists (and can therefore be updated) is the PC Control-Enabled flag.

7.2 Signals used

The PC-to-SRAM interface uses 8 signal lines from the PC to the XSV board, and 4 signal lines from the XSV board to the PC.

The names given to these signals and their pins on the parallel port connectors are listed below. In reading the table, note the following:

- A signal Dx (where x is a number) is a data pin on the parallel port connectors.
- A signal Sx (where x is a number) is a status pin on the parallel port connectors.
- The notation signalname(x:y) is used to indicate a bus named "signalname", with a most significant bit number of x and a least significant bit number of y.
- The notation signalname(z) is used to indicate bit number z of a bus named "signalname".
- The notation Dx:Dy or Sx:Sy (where x and y are numbers) is used to indicate a range of data or status pins on the parallel port connectors.

Signal name:	Parallel port pin:	Description:
NextSlice	D7	Transitions in the value of NextSlice are used to indicate when the next slice of an address or data is being transmitted. NextSlice is also used for the "apply settings" operation.
Read	D6	Setting Read = 1 (with Write = 0) instructs the PC-to-SRAM interface to perform the read operation. Setting Read = 1 and Write = 1 instructs the PC-to-SRAM interface to perform the "set address" operation.
Write	D5	Setting Write = 1 (with Read = 0) instructs the PC-to-SRAM interface to perform the write operation. Setting Write = 1 and Read = 1 instructs the PC-to-SRAM interface to perform the "set address" operation.
DataIn(4:0)	D4:D0	These 5 signals from the PC to the PC-to-SRAM interface are used to transmit each slice of an address or each slice of data to be written.
DataOut(3:0)	S6:S3	These 4 signals from the PC-to-SRAM interface to the PC are used to transmit each slice of data that has been read.

8.0 Communication protocol details

This section describes the actions that a PC programme should take in order to carry out the four operations the PC-to-SRAM interface can handle, namely: set address, write, read, and apply settings.

All of the operations require the PC programme to write to the data register of the parallel port and read from the status register of the parallel port. Note the following about the parallel port of a PC:

- The data register's I/O address is the same as the Base Address of the parallel port.
- The status register's I/O address is equal to (Base Address of parallel port + 1).
- The notation Dx (where x is a number) refers to bit number x of the parallel port data register. It also refers to the pin on the parallel port connector on the PC connected to this bit of the data register.
- The notation Sx (where x is a number) refers to bit number x of the parallel port status register. It also refers to the pin on the parallel port connector on the PC connected to this bit of the status register.

Note: The XSV board contains optional inverters on lines D0 and D1 of the parallel port. Jumpers J29 and J30 control whether or not the values on lines D0 and D1 respectively are inverted before reaching the FPGA.

For simplicity, the values in the tables in this section have been specified assuming these inverters are NOT in the signal path. However, most of the time these inverters WILL be in the signal path on the XSV board.

When these inverters are in the signal path, the PC programme must write (to lines D0 and D1 of the data register) the INVERSE of the bits listed in the tables in this section. This only applies to D0 and D1 of the data register. It does not apply to any other bits/pins/lines.

Carrying out each of the operations listed below involves writing to the parallel port data register and reading from the parallel port status register. In the tables below, note the following:

- When writing to the data register, the tables indicate what values are to be written which bits of the data register.
- When reading from the status register, the tables indicate which data will appear on which bits of the status register. (In other words, the tables show which data the PC-to-SRAM interface writes to which bits of the status register).

8.1 Initialising the PC-to-SRAM interface

To initialise the PC-to-SRAM interface, a PC programme should carry out the actions in the following table. Initialising the PC-to-SRAM interface ensures that the interface is ready to commence receiving instructions.

Action taken by PC:	Data Register Bits					Status Register Bits
	D7 [NextSlice]	D6 [Read]	D5 [Write]	D4 [DataIn(4)]	D3:D0 [DataIn(3:0)]	S6:S3 [DataOut(3:0)]
Write to data reg:	0	0	0	0	0	-

8.2 Performing the set address operation

To set the current address being stored by the PC-to-SRAM interface, a PC programme should carry out the actions in the following table.

The address that is being set is referred to as “address” in the table.

Action taken by PC:	Data Register Bits					Status Register Bits
	D7 [NextSlice]	D6 [Read]	D5 [Write]	D4 [DataIn(4)]	D3:D0 [DataIn(3:0)]	S6:S3 [DataOut(3:0)]
Write to data reg:	1	1	1	address(4)	address(3:0)	-
Write to data reg:	0	1	1	address(9)	address(8:5)	-
Write to data reg:	1	1	1	address(14)	address(13:10)	-
Write to data reg:	0	0	0	0	address(18:15)	-

8.3 Performing the write operation

To perform a write to SRAM, a PC programme should carry out the actions in the following table.

The data value that is to be written is referred to as “write_data” in the table.

Action taken by PC:	Data Register Bits					Status Register Bits
	D7 [NextSlice]	D6 [Read]	D5 [Write]	D4 [DataIn(4)]	D3:D0 [DataIn(3:0)]	S6:S3 [DataOut(3:0)]
Write to data reg:	1	0	1	0	write_data(3:0)	-
Write to data reg:	0	0	1	0	write_data(7:4)	-
Write to data reg:	1	0	1	0	write_data (11:8)	-
Write to data reg:	0	0	0	0	write_data (15:12)	-

8.4 Performing the read operation

To perform a read from SRAM, a PC programme should carry out the actions in the following table.

The data value that will be read is referred to as “read_data”” in the table.

Action taken by PC:	Data Register Bits					Status Register Bits
	D7 [NextSlice]	D6 [Read]	D5 [Write]	D4 [DataIn(4)]	D3:D0 [DataIn(3:0)]	S6:S3 [DataOut(3:0)]
Write to data reg:	1	1	0	0	0	-
Read status reg:	-	-	-	-	-	read_data(3:0)
Write to data reg:	0	1	0	0	0	
Read status reg:	-	-	-	-	-	read_data(7:4)
Write to data reg:	1	1	0	0	0	-
Read status reg:	-	-	-	-	-	read_data(11:8)
Write to data reg:	0	0	0	0	0	-
Read status reg:	-	-	-	-	-	read_data(15:12)

8.5 Performing the apply settings operation

To apply settings to the PC-to-SRAM interface, a PC programme should carry out the following actions.

Action taken by PC:	Data Register Bits					Status Register Bits
	D7 [NextSlice]	D6 [Read]	D5 [Write]	D4 [DataIn(4)]	D3:D0 [DataIn(3:0)]	S6:S3 [DataOut(3:0)]
Write to data reg:	1	0	0	0	0	-
Write to data reg:	0	0	0	0	0	-
Write to data reg:	0	0	1	0	0	-
Write to data reg:	0	0	0	0*	Set D3:D1 to 0*. Place value of PC Control-Enabled flag on D0.	-

* If in future designs more flags were added to the PC-to-SRAM interface, the values to set these flags to would be written to D4:D1 at this point, along with the value for PC Control-Enabled on D0.

Remember, if the inverters on the XSV board are in the signal path on lines D0 and D1, the PC programme must write to bits D0 and D1 the inverse of the value desired.