SIMON FRASER UNIVERSITY

SCHOOL OF ENGINEERING SCIENCE

ENSC 350: Digital System Design
Spring 2009

Lab 3

Due the Week of March 23rd, 2009

For this lab you will update your DES datapath core from Lab 2 into an Avalon Memory-Mapped (Avalon-MM) bus slave on the DE2 board. It will be used to allow an Avalon-MM bus master to write input data to and read output data from your DES decryption circuit. You will need to use a mixture of structural datapath and behavioral state machine techniques in order to build your slave and generate the correct Avalon-MM slave communication protocols. Although you will reuse the DES block you designed from Lab 2 for the datapath portion of the slave, the focus of this lab will be on implementing valid slave responses to traffic on the Avalon System Interconnect Fabric.

As for the previous lab, you will be required to demonstrate that your design functions on the DE2-70 board with a sequence of bus operations and input data we provide to you on the day of your demo. You will also be required to provide a report (the details of which will be provided later in this document). There is significant documentation available on the web for the Avalon specification. Of particular note is the following document from the Altera website:

http://www.altera.com/literature/manual/mnl_avalon_spec.pdf.

You are responsible for reading this document and understanding the protocol to implement this lab. Understanding and implementing a design specification is an extremely important design skill, so it will be good practice for working in industry. The good news is that this is a relatively simple design specification to implement. The next section will detail which portions of the Avalon-MM Slave support you will be required to incorporate into your DES Slave core (henceforth referred to as the DES Slave).

Avalon-MM DES Slave (DES Slave)

Avalon Slave Interface Requirements

Table 1 shows the mapping between the ports used in your Avalon-MM slave interface and those described in the Altera Avalon Interface Specification. All of the DES Slave's ports are Avalon-MM slave interface ports. Input signals use the prefix '**i_slave**' while output signals are denoted by '**o_slave**'. All of the ports in Table 1 are *mandatory* for your slave implementation, although they may not be required for every Avalon-MM slave.

Table 1: DES Slave Ports

Port	Avalon-MM signal	Size (bits)
Clk	Clk	1
reset	Reset	1
i_slave_read	Read	1
i_slave_write	Write	1
i_slave_address	Address	2
i_slave_byteenable	Byteenable	8
i_slave_writedata	Writedata	64
o_slave_readdatavalid	Readdatavalid	1
o_slave_waitrequest	Waitrequest	1
o_slave_readdata	Readdata	64

The slave must support: 1) peripheral-controlled waitrequest reads and writes, and 2) pipelined reads with variable latency.

Section 3.5.1 and Figure 3.3 of the Avalon Interface Specification describe in detail the slave protocol for peripheral controlled waitrequest. Note that the optional begintransfer signal is not used for your DES Slave.

Section 3.5.3.1 of the Avalon Interface Specification describes slave pipelined read transfers with variable latency. Figure 3.5 of the Avalon Interface Specification shows the specific timing protocol for variable latency pipelined reads with n=2 maximum pending read transfers. Your slave interface must support at least 1 pending read transfer. This number is a property of the interface and affects the master/slave system performance but not the design of the bus master accessing the slave. It is the slave's responsibility to prevent the number of pending reads from exceeding this maximum with the use of the **o_slave_waitrequest** signal. The slave state information included in this document refers to a slave interface with a maximum of 1 pending read transfer. ***For a 10% Bonus, you can attempt to implement a slave interface supporting 2 pending transfers. [Note that for 2 pending read transfers the read data must always be returned in the order it was requested.]***

The DES slave will support accesses of both 32 and 64 bits. The size of the access will be controlled by the value on the **i_slave_byteenable** signal. This feature allows 32-bit masters to access the DES Slave.
Your slave only needs to support 4 values for **i_slave_byteenable** shown in Table 2 (other values can be ignored).

Table 2: Byte Enable Values for DES Slave

i_slave_byteenables(7 downto 0)	**Access Size (bits)**	**Valid bytes**
"11111111"	64	0,1,2,3,4,5,6,7
"11110000"	32 (MSW)	4,5,6,7
"00001111"	32 (LSW)	0,1,2,3
"00000000"	0	None

DES Slave Top Level (des_slave.vhd)

The block diagram shown in Figure 1 is a suggested architecture for your design. It shows the flow of data through the DES Slave but does not include control and status signals. The slave shown here consists of a control block, an address decoder, a register bank, DES FIFOs and an output multiplexer. It is ***strongly recommended*** that you base your implementation on this architecture but alternate working solutions will be accepted as long as they meet the requirements of the Avalon interface and the testbed. Note that the responses of the DES Slave to read and write transfer requests as well as the functional behavior of your slave must still be consistent description in this document if you choose not to use the architecture in Figure 1. Your DES decryption circuit will interface with two FIFOs, similar to the testbed design in Lab2, but you are required to use the FIFO components provided for this Lab instead. These FIFOs are slightly different from the ones used in Lab 2 and their functionality will be covered in the next section.

DES block (des.vhd)

The DES block as mentioned earlier will be reused from Lab 2. **No** modifications to this block's datapath or interface are required for this lab.

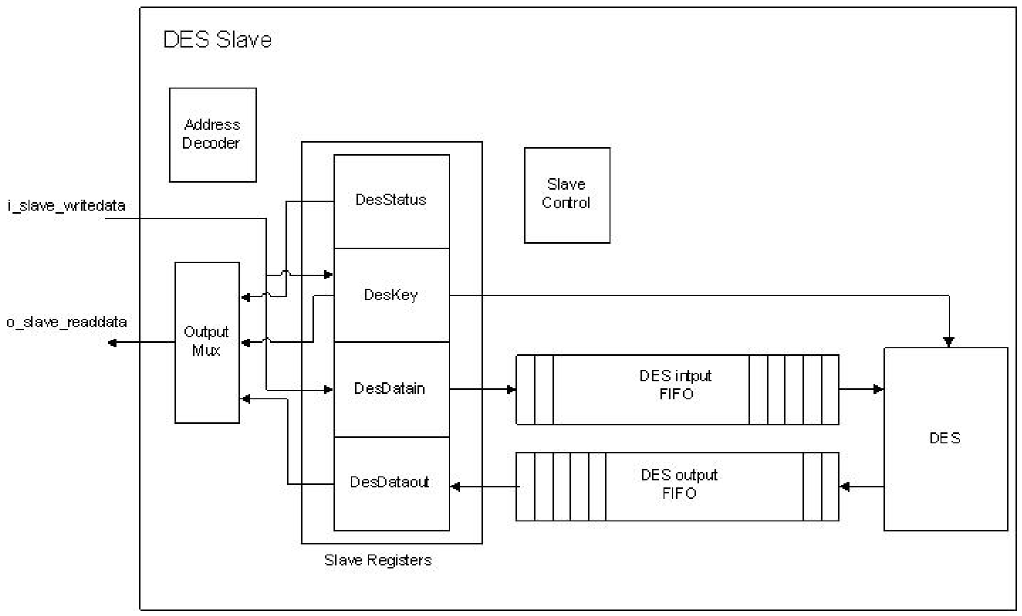

Figure 1: Block Diagram of Avalon-MM DES Slave

DES Input FIFO (des_input_fifo.vhd)

The DES Input FIFO block, as shown in Figure 2, contains two 32-bit wide x 64-word deep FIFOs. On the 'DES side' of the FIFO block, the ports for these internal FIFOs are aggregated to present an identical DES input interface to the one for your Lab 2 testbed. The DES block uses the **i_des_read** port to read data on the 64-bit **o_des_data** bus. The timing for DES FIFO reads is unchanged from the Input FIFO used in Lab 2. The **o_des_empty** signal presents the logical **OR** of the two 32-bit FIFO empty signals to the DES block.

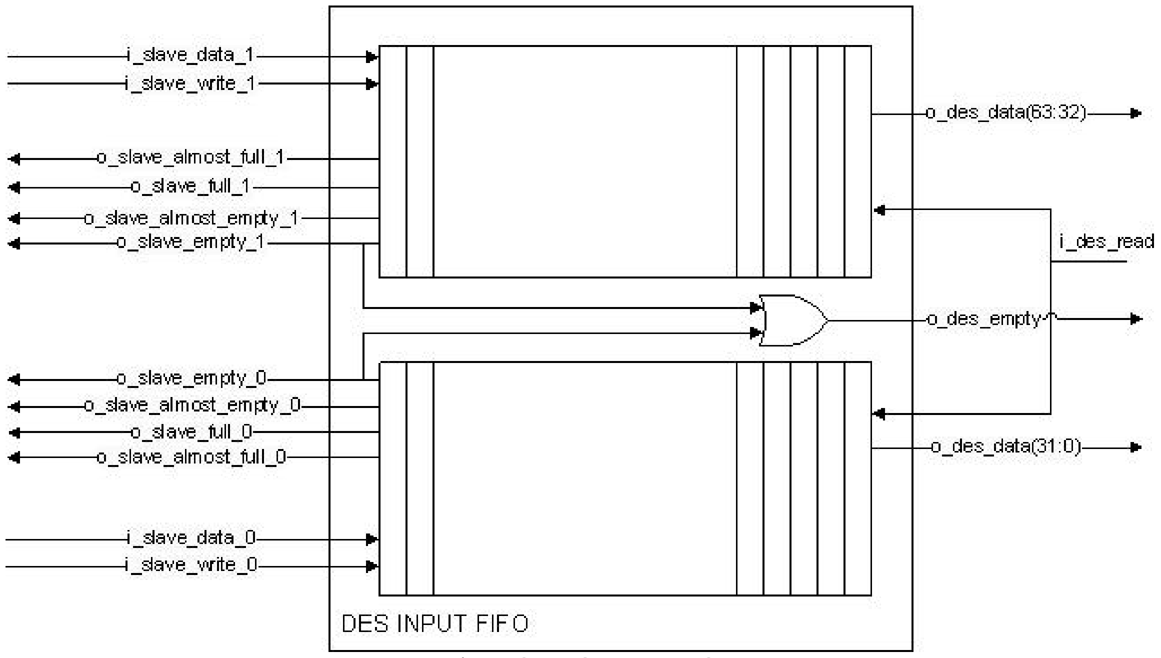

Figure 2: DES Input FIFO

For the 'Avalon-MM side' of the FIFO block, control, data and status ports are available for each 32-bit internal FIFO. The almost empty ports, when high, signal that the corresponding FIFO contains less than 16 words (1/4 max capacity). The almost full ports, when high, signal that the corresponding FIFO contains at least 48 words (3/4 max capacity).

DES Output FIFO (des_output_fifo.vhd)

The DES Output FIFO provides a 'read' interface for decrypted data. Once again this FIFO block is made up of 2 32-bit wide FIFOs. On the 'DES side' of the FIFO block, the ports for these internal FIFOs are aggregated to present an output data interface identical to the one used for your Lab 2 testbed. The DES block uses the **i_des_write** port to write data to the 64-bit **i_des_data** port. The timing for DES FIFO writes is unchanged from the Output FIFO used in Lab 2. The **o_des_full** signal presents the logical **OR** of the two 32-bit FIFO full signals to the DES block.

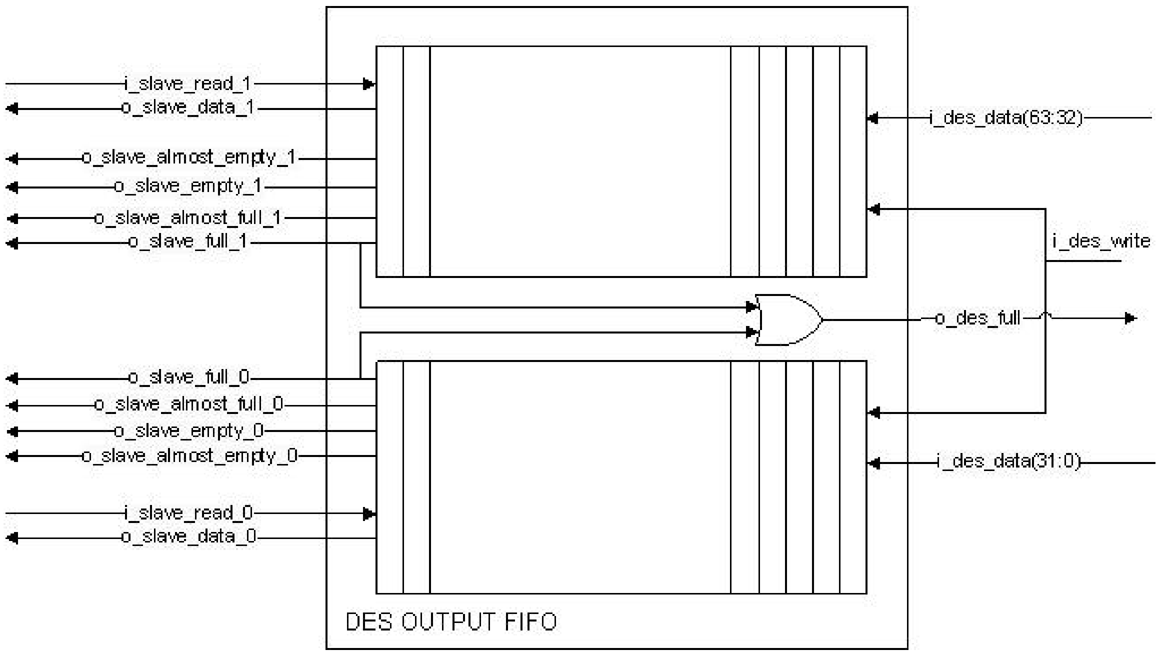

Figure 3: DES Output FIFO

The only difference between the 'Avalon-MM side' of the DES input and DES output FIFOs is the **i_slave_read** and **o_slave_readdata** ports, which replace the DES INPUT FIFO **i_slave_write** and **i_slave_writedata** ports respectively.

Address Decoder

The Address Decoder block is a combinational block that decodes the value on the **i_slave_address** port to provide 'enable' signals for individual slave registers. The truth table for this operation is shown in Table 3.

Table 3: Slave Address Decoding

i_slave_address(1 downto 0)	Slave Select Signals (3 downto 0)
"00"	"0001"
"01"	"0010"
"10"	"0100"
"11"	"1000"

The Address Decoder block also encodes the values on the **i_slave_byteenable** port to provide enable signals for individual slave register words[1]. The truth table for this operation is shown in Table 4.

[1] For this lab the term, word will refer to a 32-bit value.

Table 4: Slave Byte Enable Encoding

i_slave_byteenable(7 downto 0)	Slave Word Enable (1 downto 0)
“11111111”	“11”
“11110000”	“10”
“00001111”	“01”
“00000000”	“00”

DES Slave Address Map

Table 5 shows the mapping of addresses to 64-bit registers inside the DES Slave. The most significant word (MSW), bits 63 downto 32, of a register is accessed when the **slave word enable** signal **= “1x”** (as shown in Table 4). Similarly the least significant word (LSW) is accessed when the **slave word enable** signal = “x1” (again see Table 4).

Table 5: DES Slave Address Map

Address	i_slave_address (1 downto 0)	Access Type	Description
0	“00”	Read-Only	DES Slave Status
1	“01”	Read-Write	DesKey
2	“10”	Write-Only	DesDatain
3	“11”	Read-Only	DesDataout

DES Slave Registers

It is the responsibility of the slave to manage access restrictions. Writes to a Read-Only address should be ignored by the slave and reads from a Write-Only address should always return zero data with no wait states. All register bits should be set to ‘0’ after ‘reset’ is asserted.

a) DES Slave Status

Status for the DES Slave is stored in the DES Slave Status read-only register. Status is captured by the DES slave on a per word basis, as shown in Table 6. All flags and bits are active high. There are two types of status bits stored in this register: FIFO status and Transfer status.

FIFO status bits are related to the DES input and DES output FIFOs while Transfer status bits are related to the successful completion of slave transfers. FIFO status should be constantly updated (i.e. every clock cycle) and Transfer status should be updated upon completion of a transfer on the related word.
There are two types of transfer status bits: timeout status and ‘sticky’ timeout status. Timeout status indicates that the most recent word transfer ‘timed out’. This event originates from within the slave control block and is described in detail later. The ‘sticky’ timeout flag is set when a transfer timeout occurs and remains set until the slave ‘reset’ signal is asserted. The flag is termed ‘sticky’ because once it has been set, it remains set to indicate that an error has occurred during one of the previous transactions as opposed to the timeout flag, which can only indicate if an error occurred during the ***previous*** transaction.

b) DesKey

The DES Key register is accessible through the DesKey address and stores the 64 bit DES key used by the DES block. This address is considered read/write. The slave does not have to concern itself with synchronizing a new key with incoming data. It is the master’s responsibility to ensure a new key is loaded when the slave is not currently processing input data.

c) DesDatain

The DesDatain address is a write-only address that allows an Avalon-MM master to write data to the DES INPUT FIFO for processing by the DES component.

d) DesDataout

The DesDataout address is a read-only address that allows an Avalon-MM master to read decrypted data from the DES OUTPUT FIFO.

Table 6: Des Slave Status Register Definition

Bit	Name	Description
63:32	RESERVED	Always reads 0x"00000000"
31:30	RESERVED	Always reads "00"
29	write_fifo1_almost_full	Almost full status of the MSW FIFO in the DES Input FIFO block
28	write_fifo1_almost_empty	Almost empty status of the MSW FIFO in the DES Input FIFO block
27	write_fifo1_full	Full status of the MSW FIFO in the DES Input FIFO block
26	write_fifo1_empty	Empty status of the MSW FIFO in the DES Input FIFO block
25	write_fifo1_timeout	Transfer timeout status of last transfer on MSW of DesDatain
24	write_fifo1_timeout_sticky	'Sticky' transfer timeout flag for MSW of DesDatain
23:22	RESERVED	Always reads "00"
21	write_fifo0_almost_full	Almost full status of the LSW FIFO in the DES Input FIFO block
20	write_fifo0_almost_empty	Almost empty status of the LSW FIFO in the DES Input FIFO block
19	write_fifo0_full	Full status of the LSW FIFO in the DES Input FIFO block
18	write_fifo0_empty	Empty status of the LSW FIFO in the DES Intput FIFO block
17	write_fifo0_timeout	Transfer timeout status of last transfer on LSW of DesDatain
16	write_fifo0_timeout_sticky	'Sticky' transfer timeout flag for LSW of DesDatain
15:14	RESERVED	Always reads "00"
13	read_fifo1_almost_full	Almost full status of the MSW FIFO in the DES Output FIFO block
12	read_fifo1_almost_empty	Almost empty status of the MSW FIFO in the DES Output FIFO block
11	read_fifo1_full	Full status of the MSW FIFO in the DES Output FIFO block
10	read_fifo1_empty	Empty status of the MSW FIFO in the DES Output FIFO block
9	read_fifo1_timeout	Transfer timeout status of last transfer on MSW of DesDataout
8	read_fifo1_timeout_sticky	'Sticky' transfer timeout flag for MSW of DesDataout
7:6	RESERVED	Always reads "00"
5	read_fifo0_almost_full	Almost full status of the LSW FIFO in the DES Output FIFO block
4	read_fifo0_almost_empty	Almost empty status of the LSW FIFO in the DES Output FIFO block
3	read_fifo0_full	Full status of the LSW FIFO in the DES Output FIFO block
2	read_fifo0_empty	Empty status of the LSW FIFO in the DES Output FIFO block
1	read_fifo0_timeout	Transfer timeout status of last transfer on LSW of DesDataout
0	read_fifo0_timeout_sticky	'Sticky' transfer timeout flag for LSW of DesDataout

Output Multiplexer

The Output Multiplexer uses the decoded slave address and encoded slave byte enable signals to route the appropriate register value to the **o_slave_readdata** port. This block also drives all 'invalid' bytes of the DES Slave read data low.

Control Block

The control block implements the transfer state machine for the DES Slave. It generates: 1) control signals for the 'Avalon-MM side' of the DES INPUT and DES OUTPUT FIFOs, 2) timeout status signals for the DES Status register, 3) the **o_slave_waitrequest** and **o_slave_readdatavalid** signals driven on the Avalon-MM Slave interface.

Examples of possible read and write state-machines are given in the following section. Note for a slave supporting 1 pending pipelined read (i.e. n=1), the read state-machine and write state-machine can be implemented separately as these operations are mutually exclusive. For multiple pending pipelined reads (i.e. n>1), Avalon-MM slaves must be able to handle write requests that are initiated between pending read requests.

Example Read State machine

Table 7: Example of Read States

State	o_readdatavalid	o_waitrequsest	Description
idle	‘0’	‘X’	Waiting for a read request or processing a write transfer.
wait	‘0’	‘1’	Waiting for data to complete transfer.
valid	‘1’	‘0’	Read data is available on Avalon interface.

Table 8: Example Read State Transitions

Current State	Next State	Description
idle	idle	reset = ‘1’
		i_slave_read = ‘0’
	wait	i_slave_read = ‘1’ and data not ready for transfer
	valid	i_slave_read = ‘1’ and data ready transfer
wait	idle	reset = ‘1’
	wait	Data not ready for transfer
	valid	Data ready for transfer
		Transfer timeout.
valid	idle	reset = ‘1’
		i_slave_read = ‘0’
	iait	i_slave_read = ‘1’ and data not ready for transfer
	valid	i_slave_read = ‘1’ and data ready transfer

Example Write Statemachine

Table 9: Example Write States

State	o_readdatavalid	o_waitrequsest	Description
idle	‘X’	‘X’	Waiting for a write request or processing a read transfer.
wait	‘0’	‘1’	Waiting until slave is ready to accept data.
valid	‘0’	‘0’	Write data is accepted by the slave

Table 10: Example Write State Transitions

Current State	Next State	Description
idle	idle	reset = ‘1’
		i_slave_write = ‘0’
	wait	i_slave_write = ‘1’ and slave not ready to accept data.
	valid	i_slave_write = ‘1’ and slave ready to accept data
wait	idle	reset = ‘1’
	wait	Slave not ready to accept data.
	valid	Slave ready to accept data.
		Transfer timeout.
valid	idle	reset = ‘1’
		i_slave_write = ‘0’
	wait	i_slave_write = ‘1’ and slave not ready to accept data.
	valid	i_slave_write = ‘1’ and slave ready to accept data.

Timeout Feature

If a single transfer causes the **o_slave_waitrequest** to be asserted for greater than the defined timeout threshold, the slave control block will release the requesting master by deasserting **o_slave_waitrequest** and report the transfer as 'timed out' in the DES Slave Status register. The timeout threshold is dependent on the maximum latency (in clock cycles) of your design. You must determine this value based on your design and justify the value used. (*Hint: a starting point is the maximum latency of your DES block*). Timed out read transfers must still complete over the Avalon-MM interface. For timed-out read transfers, the slave should drive all zeroes on the **o_slave_readdata** port. It is the master's responsibility to check the DES Status register to see if a transfer was successful.

Avalon Test System

To test your DES Slave, you will be provided the test system shown in . This system, which will use a 32-bit Avalon-MM master, vector ROMs and comparison logic to verify your design, is similar to the Lab 2 testbed. Note that all logic with the exception of the DES Slave will be provided on the course website. The following section briefly describes each test system component provided.

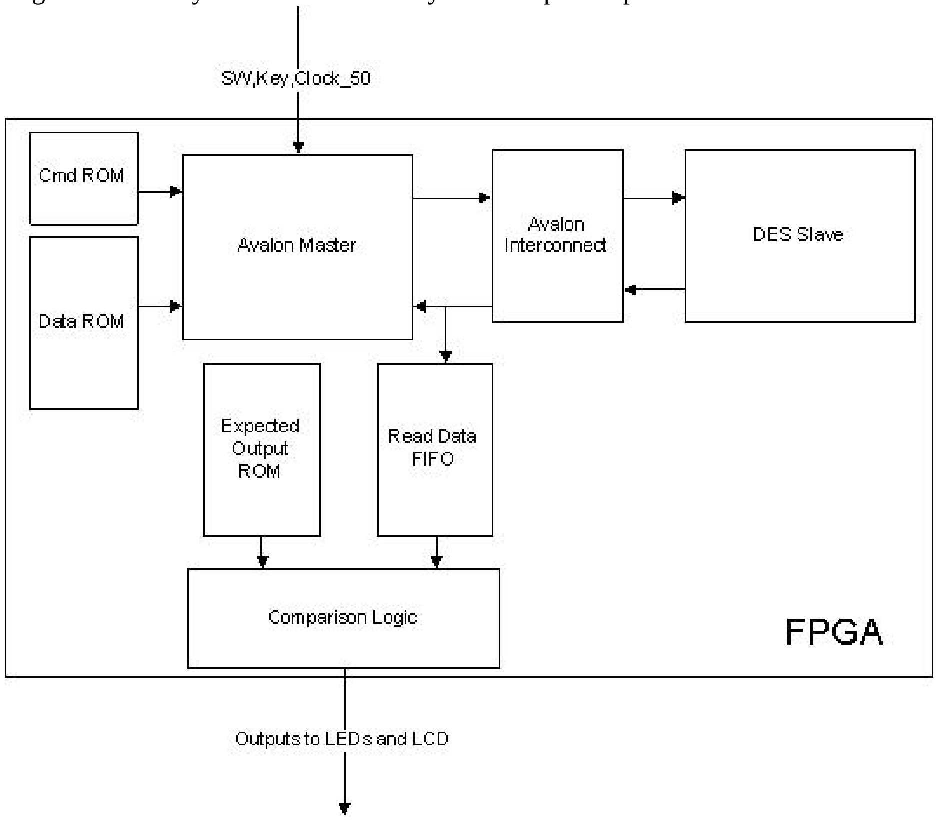

Figure 4: Avalon Slave Test System

Avalon Interconnect (avalon_interconnect_m32_s64.vhd)

The System Interconnect Fabric for an Avalon System provides: 1) master arbitration, 2) slave selection, and 3) address alignment to support communication between multiple Avalon-MM components. This is typically generated by Altera's System on Programmable Chip Builder (SOPCBuilder), a tool used to automate the connection of hardware IP components for use on Altera FPGAs. For this lab, where the focus is on the Avalon-MM slave protocol, we will provide you with a simple component to mimic the behaviour of the Avalon System Interconnect Fabric for a single-master/single-slave Avalon system. This component only implements the address alignment functionality of the interconnect fabric as no master arbitration or slave selection is needed for our simple system. Avalon address alignment is covered in detail in Section 3.6 of the Avalon Interface Specification.

Avalon Master (avalon_MM_master_32.vhd)

The Avalon Master initiates Avalon compliant read and write transfers for your DES Slave. It is a simple Master that fetches commands and operands from an instruction memory (Cmd ROM) and data from a separate data ROM and executes simple read and write transfer requests to the Avalon Interconnect. The master supports peripheral controlled wait states and pipelined read requests and can fully exercise your DES Slave's requirements. The Master executes instructions only, it does not make decisions based on feedback from the slave. For example the Avalon Master will not wait for read data to be returned before moving on to another request unless stalled by the DES Slave waitrequest signal. The master can also request writes to an already full DES INPUT FIFO or reads from an empty DES OUTPUT FIFO. The DES Slave will have to stall the master until these transactions complete or timeout.

Command ROM (master_cmd_rom.vhd)

The Avalon Master Command ROM (Cmd ROM) is a 21-bit wide x 64-word deep instruction ROM used which stores the Master execution sequence. Table 11: Master Command Word DefinitionTable 11 shows the description of the fields in a single Cmd ROM entry or instruction word. After your program starts the master will iterate through the Cmd ROM (starting at address 0) until it has executed all 64 entries. At the end of the program signal the master will go into a 'done' state and only reset can restart the master program.

Table 11: Master Command Word Definition

Bit	Name	Description
20:19	Opcode	"00" – NOP. No operation. Master does nothing. "01" – Read. Initiate an Avalon Read Transfer. "10" – Write, Initiate an Avalon Write Transfer. "11" – Wait, Insert wait cycles.
18:16	Master Address	3 bit master address to be used for read and write transfers. Only valid for Read and Write commands.
15:7	Count	For Read commands this indicates the number of successive reads to be requested from Master Address. For Write commands this indicates the number of successive writes to be requested to Master Address. For Wait commands this indicates the number of cycles to wait before fetching a new command word. This command field is ignored for NOP commands.
6:0	Data ROM Address	7 bit Data ROM address indicates the location of the initial data that should be written to the Slave. Only valid for Write opcodes. For Write commands with Count > 1, the ROM Address increments with the current count. For example if Count = 3 and Data ROM Address = 0x10, then the first data value written to the DES Slave is stored in the Data ROM at address 0x10. The second data value is stored at address 0x11 and so on.

Data ROM(master_data_rom.vhd)

The Data ROM is a 32-bit wide x 128-word deep ROM that stores write data for the Avalon Master. When the Avalon Master fetches a command word with a 'Write' opcode, it then fetches 32 bits of data from the Data ROM and requests to write it to the DES Slave.

Read Data FIFO

Valid (i_master_readdatavalid = ‘1’) data returned to the master should be driven into the Read Data FIFO for comparison. This 32-bit wide x 64-word deep FIFO will be provided. Once again the write and read timing for this FIFO is consistent with the Lab 2 Input and Output FIFOs.

Expected Data ROM

The expected data ROM is a 32-bit wide by 1024-word deep ROM used to verify the master read data. The only difference between this Expected Data ROM and the Expected Data ROM used in Lab 2 is the new width of 32 bits to match the master read data.

Comparison Logic

The comparison logic compares data from the new Expected Data ROM and the Read Data FIFO components. Once again green LED is turned on if the result matches the expected value, otherwise a red LED is turned on and the comparison logic is stalled to let the user view the offending result on the LCD.

<u>The Testbed</u>

A testbed for verifying your design on the DE2-70 board has been provided. This testbed will be used in your demo to evaluate your design. The testbed files are:

- avalon_testbed.vhd – top level testbed file that implements the Avalon Test System.
- avalon_MM_master_32.vhd – 32 bit Avalon MM master.
- avalon_interconnect_m32_s64.vhd – Component simulating Avalon Interconnect Fabric.
- expected_rom.vhd – wrapper vhdl file for 1024x32-bit ROM of expected output values
- expected_rom.cmp – netlist file for 1024x32-bit ROM of expected output values
- expected_rom.mif – initialization file for 1024x32-bit ROM of expected output values
- read_fifo.vhd – wrapper vhdl file for read FIFO the Avalon System interacts with
- read_fifo.cmp– netlist file for the read FIFO the Avalon System interacts with
- master_cmd_rom.vhd – wrapper vhdl file for 64x21-bit Command ROM
- master_cmd_rom.cmp – netlist file for 64x21-bit Command ROM
- master_cmd_rom.mif – initialization file for 64x21-bit Command ROM
- master_data_rom.vhd – wrapper vhdl file for 128x32-bit Command ROM
- master_data_rom.cmp – netlist file for 128x32-bit Command ROM
- master_data_rom.mif – initialization file for 128x32-bit Command ROM

Use the Testbed:

1. KEY(0) is the circuit reset. This MUST be pressed after downloading to the board
2. KEY(1) is the master start signal. This MUST be pressed after reset to run the program in the master Cmd ROM.
3. LEDG(0) will light up as long as the output of your circuit matches that of the expected output ROM. If this stays lit then everything seems to be working fine.
4. LEDR(0) will light up the DES circuit output does not match the expected output.
5. If a mismatch is detected, the comparison circuit in the testbed will stall. No more comparisions will be done until after a system reset.
6. You can view the expected output, circuit output, and ROM address where mismatch occurred on the LCD. To view information on the LCD you must use SW(17:16) to select which value you want to display (00=circuit, 01=expected, 10 and 11 = address) and then push KEY(2) to update the LCD.

What you are going to Do:

Task 1: Draw State Diagrams for n = 1 and n = 2

Although you are only required to implement the slave interface for pending requests such that n=1, you are also required to draw the state diagram for n = 2.

Important note: Those of you who are good with Google will likely find lots of VHDL descriptions of the Avalon interface on the web. Feel free to use these as a ***reference***, but remember, your design has to match our specifications ***exactly***. Even if this were not a form of ***plagiarism***, it is probably more work to modify an existing implementation than to write your own given the information provided here. Also beware that at least a few of the on-line versions have bugs. Finally, remember any direct copying of on-line versions would be plagiarism resulting in the loss of 30% of your final mark (as specified during the first class). *[Hint: This should look familiar from your previous lab. =)]*

Task 2: Implement the VHDL for each of the necessary subcomponents in the design

Write VHDL for each of the new subcomponents for your design and simulate them thoroughly.

Task 3: Integrate your subcomponents to create your final design

This should be the final implementation of your overall design. Again, be sure to simulate your design thoroughly.

Task 4: Implement your design on the DE2-70 board and test it

Implement your design on the DE2-70 board and test it with the command and data input vectors provided on the website. The processing results will be compared with the provided values to be stored in your "expected output" ROM.

Task 5: Demonstrate your Design and Submit your report.

In your demonstration you will be asked:

- Show your source code and answer questions about it
- Show a simulation of your DES circuit converted into an Avalon slave or any of the sub-circuits and explain what is going on in the waveform
- Show your circuit working on the DE2-70 board with the provided testbed
- ***Note: Be sure to let us know if you implemented the bonus (i.e. n=2)***

Your written submission should include:

- A title page with both students names, student numbers, course number, and date.
- A state diagram (what type of state machine did you use) for n=1 and n=2,
- A system block diagram,
- A summary of resource usage,
- The minimum clock period and maximum throughput of your design,
- A hierarchy tree for your VHDL files
- Anything that you think is exceptional about the design of your DES Slave block (e.g. n=2).
- A zipped softcopy of all your source files. This must be messaged to the TA via WebCT. Source files should have comments, consistent indentation, adequate white spaces, descriptive names for signals/variables, one entity/architecture pair per file, and a maximum line length of 80 characters.

Please note that your reports are not marked by weight. Clearly written, short and succinct reports are highly preferable. There is no maximum or minimum page limit, but your target should be between 3-5 pages, not including title page. Also, a picture is worth a 1000 words, so if possible illustrate. Please do not include a background description of the generic DES algorithm or the Avalon interface, it is not necessary.

Your source files will be run through a comparison program to help identify copied segments of source code between groups. Copying any portion of another groups source files is considered cheating.

<u>Marking Scheme</u>

Demo

2 marks for working Simulations and reasonable responses to questions

2 marks for working on the DE2-70 and reasonable responses to questions

Total = 4

Report

2 mark for accurate State Diagrams (where n=1 implementation must match VHDL)

1 mark for accurate System Block Diagram (must match VHDL)

1 mark for resource usage report

1 mark for hierarchy tree and VHDL coding practices

1 mark for readability of report

Total = 6

<u>Suggested Milestones:</u>

The following are some suggested milestones for this project. We aren't going to be checking your progress, but you can use this list to see if you are falling behind.

Start Project: March 7th, 2009

Milestone 1: Draw State Diagrams for n = 1 and n =2 , March 10th, 2009

Milestone 2: Implement all of the VHDL for your new subcomponents: March15th, 2009

Milestone 3: Integrate all of these components: March 20th, 2009

Milestone 4: Finish a thorough testing of your design on the board and demo it: March 25th, 2009

Milestone 5: Submit your report on your design: March 27th, 2009

<u>Bonus ONLY:</u>

Remember that a 10% bonus is available for those of you that choose to support n=2 pending reads and writes.

<u>What to hand in:</u>

1) A copy of your lab report submitted into the ensc350 drop box by **3:30pm on March 27th, 2009**. Note the time, as we won't accept excuses (including jammed printers/no paper in the printers) and we won't be accepting soft copies.

2) A zip file with your VHDL files for both your new DES Slave core and your testbed as well as a README file with hierarchy, group members and student IDs, group number. Please note this should be emailed to the TA's WebCT email account.

<u>REMEMBER: Don't be late, as late submissions will be severely penalized (minimum of 50%) even if it is only by 1 hour.</u>

Then you are done!!!