

*****To be replaced by***
standard cover**

DMA Controller for AMBA Bus IP Core

Final Report

Version 1.00
Gareth Morris

Project number: 12

Executive Summary

This report documents the complete development process of a DMA controller for an on board computer of a small satellite.

Objectives of the project were to implement a complete pool of source code for the controller, document testing results, and produce a detailed specification.

Briefly, work proceeded roughly as planned - with small changes due to various minor problems which were solved satisfactorily. Once development was complete the end results were working source code for the controller, and a specification which was related to the functionality of the final design. The code was tested as fully as possible and worked correctly to the specifications. Synthesis to a semiconductor device was also tested. Finally this documentation was produced.

All the aims set out initially have been achieved. Some extra work was also completed with regards to improving the design. Therefore the author judges the project to be a success and also a rich learning experience.

Material in this report covers an introductory overview of the project, the work carried out to finish the project, documentation produced in the development of the controller, detailed suggestions for improvement and a chapter of final conclusions.

Table of Contents

Executive Summary.....	i
Table of Contents	ii
Index of Figures.....	iii
1 Detailed Introduction.....	1
1.1 Project Context.....	1
1.2 Potential Applications	2
1.3 Project Brief.....	2
2 Project Progress Review.....	3
2.1 Time Plan.....	3
2.2 Research.....	3
2.2.1 AMBA Bus DMA Controller Specification	3
2.2.2 Final Year Projects 2000/2001	3
2.2.3 LEON-1 VHDL Model Description.....	3
2.2.4 AMBA Bus Specification.....	3
2.2.5 The Student's Guide to VHDL.....	3
2.2.6 8237a High Performance DMA Controller Data Sheet.....	3
2.3 Specification.....	4
2.4 VHDL Coding.....	5
2.4.1 Registers.....	6
2.4.2 AMBA bus Buffering	8
2.4.3 Timing and Control.....	9
2.4.4 Top Level.....	12
2.4.5 Arbiter.....	12
2.5 Mid Project Report	13
2.6 Testing.....	13
2.6.1 Back Annotated Timing	14
2.7 Final Report.....	14
3 Final Results.....	15
3.1 Basic Test.....	15
3.2 Multi-Word Test.....	16
3.3 Address Bus Overrun	17
3.4 Delayed Transfers	18
3.5 Errors During a Transfer.....	19
3.6 Multiple Consecutive Transfers.....	20
4 Improvements	21
4.1 Burst Mode	21
4.2 Transfer Queuing.....	21
4.3 Multiple Transfer Configurations	21
4.4 Maximum Transfer Retries.....	22
4.5 Use AMBA APB for Control.....	22
5 Conclusion	23
5.1 Achievements.....	23
5.2 Learnings	23
5.3 Personal.....	23
6 Signal Name Key	24
7 References	25
8 Glossary.....	26
Appendices	A
Specification.....	A
Time Plan.....	A
Source Code (on diskette).....	A

Index of Figures

Figure 1: Block diagram of DMA controller showing the individual VHDL modules pictorially	5
Figure 2: Diagram to illustrate operation of the 32 bit counter	6
Figure 3: Intermediate test results for a 32 bit wide data register	7
Figure 4: Intermediate test results for a 32 bit wide counter.....	7
Figure 5: Intermediate test results for the AMBA bus output buffering module	8
Figure 6: Intermediate test results for the first timing and control module implementation.....	10
Figure 7: Intermediate test results for the second timing and control module implementation showing significant improvement compared to the original implementation	10
Figure 8: Original(left) and final(right) state diagrams of timing and control module	11
Figure 9: Intermediate test results for AMBA bus arbiter.....	12
Figure 10: Results of DMA controller basic functionality test	15
Figure 11: Results of controller multi-word transfer capability test	16
Figure 12: Results of AMBA bus overrun test.....	17
Figure 13: Results of AMBA bus overrun fix test.....	17
Figure 14: Results of AMBA bus delayed transfer test.....	18
Figure 15: Results of AMBA bus delayed transfer test.....	19
Figure 16: Results of multiple the consecutive transfers test.....	20

1 Introduction

The aim of the project is to develop the documentation, and source code for a DMA controller for the onboard computer of a small satellite. This section is to provide high-level, yet detailed, information regarding this project.

In order to understand this project's paradigm, some contextual information may be of benefit.

1.1 Project Context

Thanks to advances in integrated circuit technology, it is possible to fit increasingly more functionality into an ever decreasing physical size. This may have advantages of reduced cost, improved performance, and reduced power consumption, as well as the obvious size and weight reduction. These factors are important in many applications, not least in small commercial satellites, where it is important to reduce weight and maximise payload, using largely solar power.

At the moment the concept of having a complete satellite on a single IC is still a dream, however current technology allows for an entire satellite onboard computer to be designed onto one silicon device, and at comparatively low cost and development time. This project is being worked on by the University of Surrey's Space Centre.

The single chip system is to be prototyped onto an state of the art FPGA (Field Programmable Gate Array), which is a volatile RAM based device – allowing digital logic circuits to be realised and modified infinitely. The actual IC to be used is manufactured by the Xilinx corporation, and is known as 'Virtex'. As a point of interest these devices are produced using a 0.18 micron process, the same as Intel's Pentium III microprocessors.

One way to develop logic for FPGAs is to use a hardware description language(or HDL). In the original project description (reference 2) it was stipulated that the project was to be developed in VHDL (Very high speed integrated circuit Hardware Description Language). Computing facilities available in the department labs have all the software tools needed for VHDL development.

The satellite on board computer is to use 'LEON' as it's microprocessor. LEON is a reduced instruction set processor, which is based on Sun's SPARC V8 standard (but currently without floating point maths). LEON has been programmed entirely in VHDL (the main reason for the choice of this language for the project) by Jiri Gaisler of the European Space Agency. Public versions of the source code, documentation (reference 3), and software tools (simulator, operating system and compilers) are released from time to time. The University of Surrey has been observing these developments closely.

The latest versions of the LEON microprocessor use AMBA as the system bus. AMBA is a bus standard developed by the ARM corporation for low power embedded systems – such as a satellite on board computer. Full technical documentation for this standard has been released by the company, and is freely available (reference 4). The use of this standard by LEON makes it easy to add external peripherals to the processor in order build up a complete system, all of which could be entirely contained in the same semiconductor device.

One device on the AMBA bus is LEON's system memory, and to complete the satellite onboard computer system other peripherals will have to be added. The application of the microprocessor is such that it is operating in a real time environment, so it is important that the processor operates with the highest possible efficiency. Peripherals of the processor may need to communicate information to and from system memory. In order to do this without interfering with the operation of the processor direct access between the peripherals to memory is required. This maybe achieved with a Direct Memory Access (DMA) Controller.

1.2 Potential Applications

In space radiation causes random inversion of bits stored in RAM devices. Over time, this gradually corrupts the contents of the memory. In order to prevent this, horizontal and vertical parity bits are stored in the memory so it is possible to find where the corrupted bits are and re-invert them. To do this special logic is required. This logic eventually goes through each memory address and ‘washes’ it. To perform this task a DMA controller is crucial – and this ‘memory washing’ is conjectured to be the main use for the controller.

The LEON microprocessor can and will have many peripheral modules interfaced to it, for instance Ethernet and telemetry controller are under development. In order for these modules to write data into the memory of the microprocessor use of a DMA system is desirable in order to avoid interfering with real time processes being executed.

1.3 Project Brief

The project brief was initially set to the following (reference 2):

“ChipSat related projects:

Project “ChipSat”: A system-on-a-chip(SOC) implementation of a small satellite on-board computer. The SOC will be based public domain VHDL Intellectual Property (IP) cores developed by the European Space Agency (ESA) – a SPARC V8 microprocessor “Leon” and a CAN core “Hurricane” and will be implemented on a Xilinx Virtex XCV800 FPGA. The operating system for the LEON core is going to be RTEMS.

.....

4. Title: “Development of a DMA controller for an AMBA interface IP core”

The project will involve VHDL programming and will require knowledge of computer architecture and interfaces. The aim is to develop a DMA controller that can be used as a part of an asynchronous interface AMBA.”

This is quite an open brief, and as the project specification was written more decisions were made as to how the DMA controller would actually function. Eventually the method detailed in the specification was the one approved by the project supervisors, and therefore the final project brief was to implement this specification.

For reference the final version of the project specification is attached to this report as an appendix.

2 Project Progress Review

In this section each of the different activities, outlined in the original time plan, needed to complete the DMA controller are discussed.

2.1 Time Plan

A project time plan was worked out during the second week of the project. This was based on conjecture, with no real understanding of the project requirements. About halfway through the autumn semester the project plan was completely revised based on a more complete assessment of the project's needs and it is this time plan that was included in the mid project report. The time plan was revised again half way through the spring semester to take into account variations in the plan due to coding time changes. This final time plan is included as an appendix to this report.

2.2 Research

The main sources of research for this report are listed in the reference section. Sufficient research to begin real work on the project was achieved after 2 to 3 weeks, however research was continued throughout the whole course of the project on an ad-hoc basis. All the reference sources used during the project, and their contribution is described below:

2.2.1 AMBA Bus DMA Controller Specification

This document, written by the author during the first part of the project was an invaluable source of reference during implementation. The final version of this specification is attached as an appendix.

2.2.2 Final Year Projects 2000/2001

A short document by the project supervisor written to give a brief description of what was required of the DMA controller. An extract of this brief is shown in section 1.3. This was of assistance as a starting point during the first few weeks of the project and was refined later.

2.2.3 LEON-1 VHDL Model Description

This is a very detailed text by Jiri Gaisler of the European Space Agency describing the operation of the LEON microprocessor. This went through two revisions during the course of the project, and appropriate changes had to be made to reflect the later version(with AMBA bus support).

2.2.4 AMBA Bus Specification

Produced by the ARM corporation this specification is available electronically on the company's web site. This was the most used source of reference, containing timing diagrams and signal descriptions for each variation of the AMBA bus.

2.2.5 The Student's Guide to VHDL

One of the recommend texts of the L3 ASIC and VLSI Design course. This book was an incomplete, yet extremely good value reference for the VHDL language used for the project.

2.2.6 8237a High Performance DMA Controller Data Sheet

The 8237 device is a DMA controller made by Intel for 80x86 microprocessor applications. It is often considered the be the archetypal DMA controller. For this reason it was looked at in some detail as a basis for the AMBA bus DMA controller. Eventually, however, the specification was considerably developed away from this device and the final version is only similar in some cursory details.

2.3 Specification

The largest part of the project during the autumn semester was the specification. This involved research into LEON, AMBA bus technology, real time embedded systems, and existing DMA controllers as described previously. Eventually enough information was obtained to work out the operating principals of the controller. To begin with rough state diagrams were drawn in a log book with additional notes. These approximate plans were eventually formalised into a first draft which was submitted as part of the mid term report. In addition this draft was examined by the project supervisors to ensure its suitability and to suggest additional features.

During the implementation phase some small problems and impracticalities were found with this first draft. Details of these may be found in the following sections. The changes made to the design were added back to the documentation so that the specification remained an accurate representation of the exact design performance.

2.4 VHDL Coding

The time required to code the project was slightly misjudged in the time plan. In fact the overall time was roughly as estimated – but the break down for the individual modules was rather awry. Final versions of the project’s VHDL source code may be found on a disk attached to this report. Below the block diagram for the DMA controller is reproduced for reference, for other information please consult the specification attached to this report.

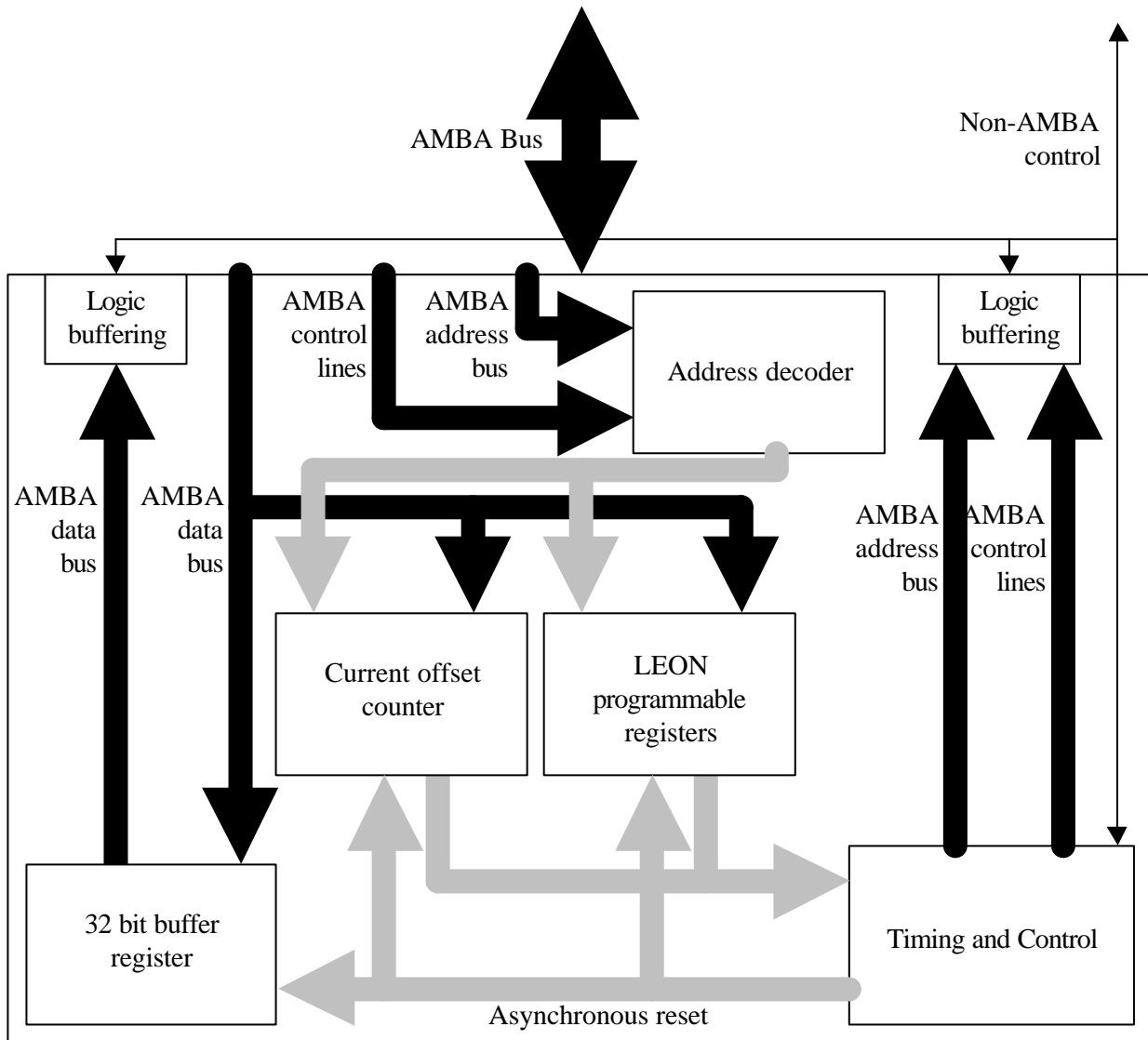


Figure 1: Block diagram of DMA controller showing the individual VHDL modules pictorially

2.4.1 Registers

The DMA controller uses two types of registers – straightforward 32bit d-type latches and a 32 bit counter. These components are similar to each other in design. The VHDL for these modules is very easy to understand, and there was no problems in writing the 32bit register. For the 32 bit wide counter some interesting issues were encountered, and these were easily fixed.

The first problem was caused by the addressing method of the AMBA bus. Data in memory is byte addressed, however the DMA controller works with the full bus word width (32 bit). Initially the counter was incremented by 1 for each word transferred. However this obviously only incremented by 1 byte, and not word. This was simply solved by increasing the size of the increment to 4 bytes.

The second problem was concerned with a feature of VHDL called a ‘sensitivity list’. In VHDL one has a set of concurrently executing processes. When a process has finished it is re-executed when a variable in its ‘sensitivity list’ changes in some way. For the system to work the counter increments on falling clock edges – however processes using the counter data should only execute on positive clock edges (to synchronise with the AMBA bus) to solve this a 32bit register was placed on the output of the counter and was constantly clocked. To see why the counter was incremented on a falling clock edge consult the diagram below:

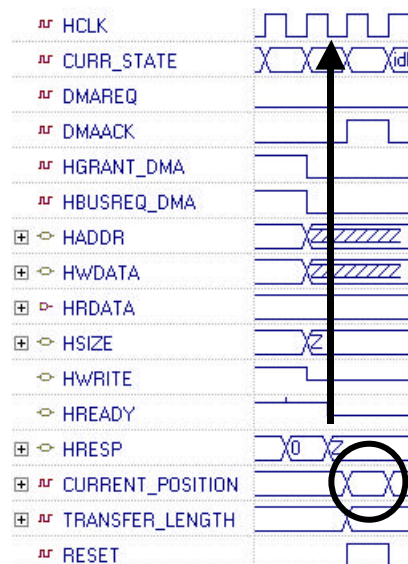


Figure 2: Diagram to illustrate operation of the 32 bit counter

The above diagram shows the end of a DMA transfer. The current position signal shows the output of the counter. Due to the tight timing constraints of the transfer (to minimise bus usage) the counter is actually incremented on the falling clock edge shown by the arrow. This is the earliest opportunity after the transfer is successfully finished and means the output is ready for the next state in the DMA controller, which is to see if the transfer is finished. By clocking on the falling clock edge, one bus cycle is saved for use in another transfer or by the microprocessor.

The intermediate test results of the 32 bit register and counter are shown the next two diagrams:

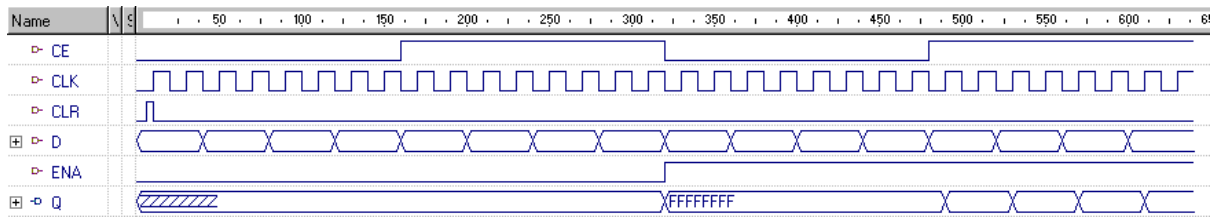


Figure 3: Intermediate test results for a 32 bit wide data register

In this test all functions of the 32 bit data register are involved. The output of the controller (Q) is at first high impedance because the output enable input (ENA) is low. After this is active the clock and clock-enable inputs are tested using a constantly varying input. D-type behaviour is expected and obtained.

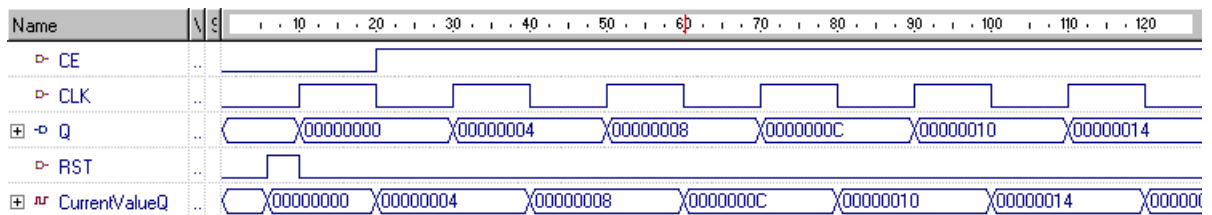


Figure 4: Intermediate test results for a 32 bit wide counter

Again all the features are tested during this vector. The CurrentValueQ signal is an internal signal to the counter, prior to the d-type latch on the output. The counter value continued to increase until \$FFFFFFFF and then the results wrapped back to \$00000000.

2.4.2 AMBA bus Buffering

The interpretation of the AMBA bus buffer changed during the project. Initially the buffering was intended to be simple logic buffering of all signals between the DMA controller and the bus. However this level of buffering turned out to be unnecessary and so the module was removed from the design. When a feature was added which allows the DMA controller to be completely removed from the AMBA bus (using DMA_DISABLE line – reference 1) the buffering module was reinstated into the design. It now tri-states all outputs from the controller, and also splits bi-directional signals into two unidirectional ones. This was very easy to design, the module test results are show below:

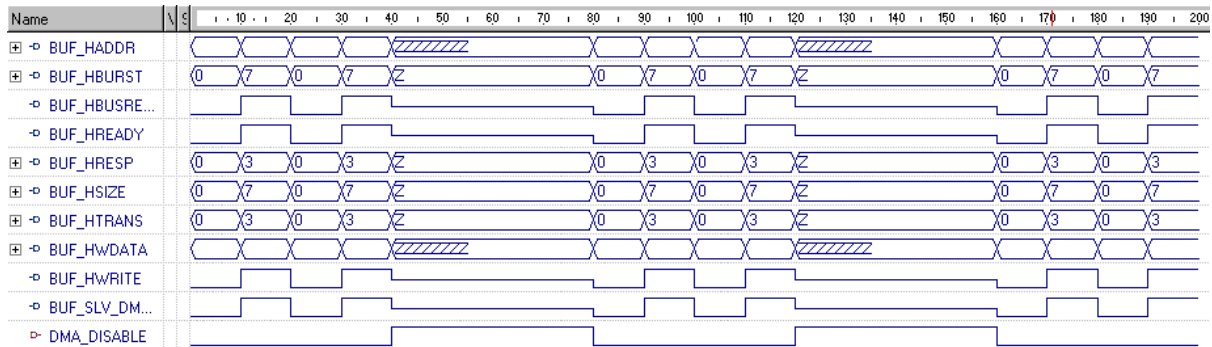


Figure 5: Intermediate test results for the AMBA bus output buffering module

For this test the inputs to the module were all toggled between high and low every 10ns. The DMA_DISABLE was toggled at a much slower rate to ensure the outputs go tri-state when this signal is active. As can be seen from the results, this works with no apparent problems.

2.4.3 Timing and Control

Creating the timing and control VHDL module was by far the largest part in developing the project, and it underwent the most change from its original specification.

Originally the timing and control section was to include the AMBA bus master state machine and also the AMBA slave registers allowing transfer configuration. As features were added it seemed sensible to split these two distinct functional blocks into separate top level modules. This improved the code's coherency and also made the internal interfaces of the DMA controller more elegant. This was due to the fact that previously bi-directional connections to the module could be changed to uni-directional ones. This solved some problems with synthesising the design when the Xilinx synthesis tools were unable to resolve the slightly complex bi-directional interface – including the many tri-state buffers needed to implement combined AMBA bus slave and masters in the same device. Separating the two components also made the routing of internal nets easier as all the AMBA bus signals no longer required links to both the modules.

The address decoder generates clock enable signals for the three 32 bit wide registers holding details of the next DMA transfer. This module was easier to write than the timing and control state machine but was still relatively complex in comparison to the other modules as it was higher level and needed to adhere to the AMBA specifications (reference 1). The behaviour of the address decoder was designed based on the AMBA documentation, it did not work quite correctly the first time as some signals changed value on the wrong clock edges. This was corrected with some minor changes to the code.

Once it was possible to set up values into the control registers, work began on implementing the main part of the project – the state machine performing all aspects of the transfer. The original version of the specification contained a diagrammatic representation is shown in Figure 8.

However, during the implementation phase of the project it became obvious that a state machine implemented in this way was unlikely to work. The reason for this is the way state machines are written in synthesise-able VHDL. State machines are written using two or three concurrently running processes. One which selects the state every machine clock cycle, the others handle state transitions and output signals. For an example of this, please refer to this modules source code(attached as an appendix). This means all transitions occur on a clock edge and a change in state implies waiting for the next edge. Previously it had been assumed that state changes would occur instantly – in VHDL this is possible, but not synthesise-able. To solve this problem additional states were added to the design.

Once the different modules were working as a together correctly possible optimisations for the timing and control module were made to increase the system's performance. These optimisations were possible due to the states added to the design, described above. The diagram below shows the timing of the initial implementation of the DMA controller. The second diagram shows the same test performed on the optimised design.

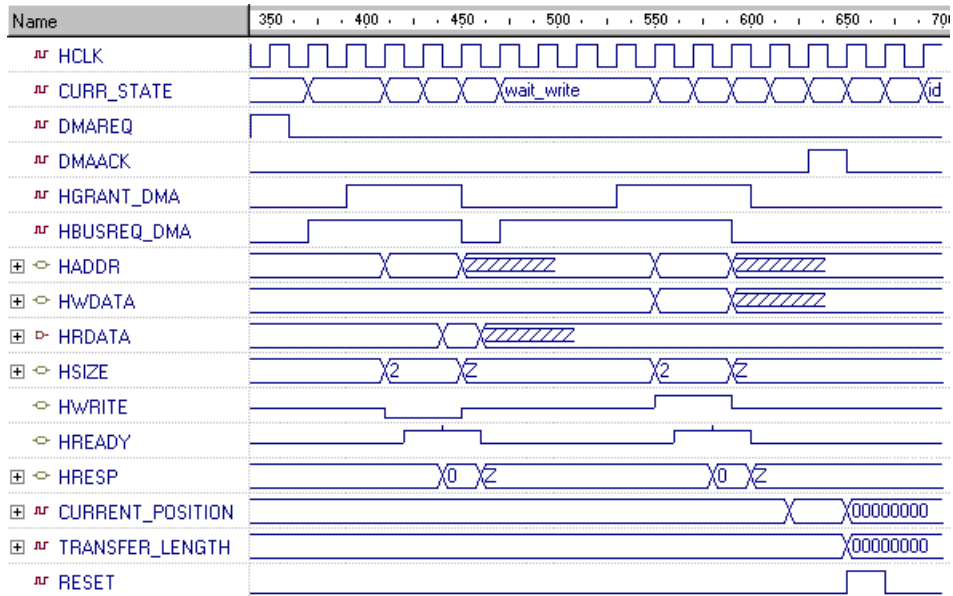


Figure 6: Intermediate test results for the first timing and control module implementation

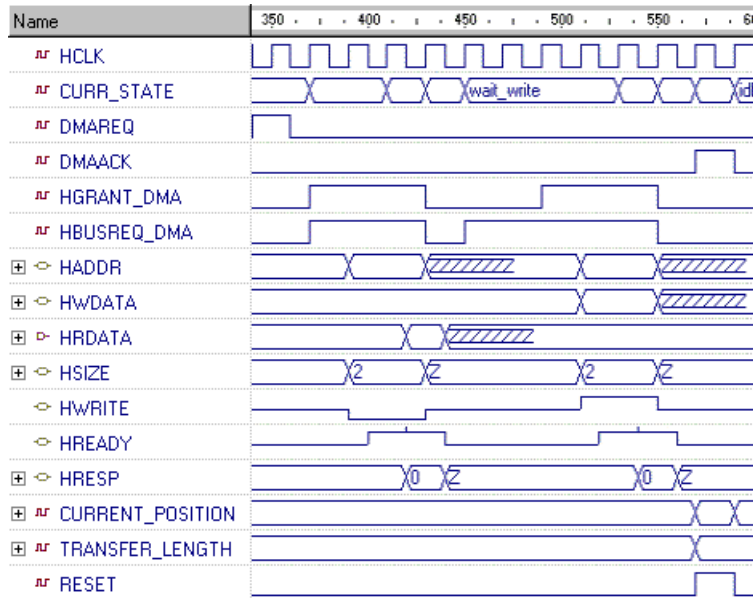


Figure 7: Intermediate test results for the second timing and control module implementation showing significant improvement compared to the original implementation

As can be seen from the diagrams the optimised design has a saving of almost 40% per word transmitted. It should be noted that although the first implementation has different timing compared to the final design both will still work correctly when attached to an AMBA bus as the savings occur mainly before and after the AMBA bus part of the transfer occurs.

The final state machine diagram is reproduced below, along with the first scheme, for comparison:

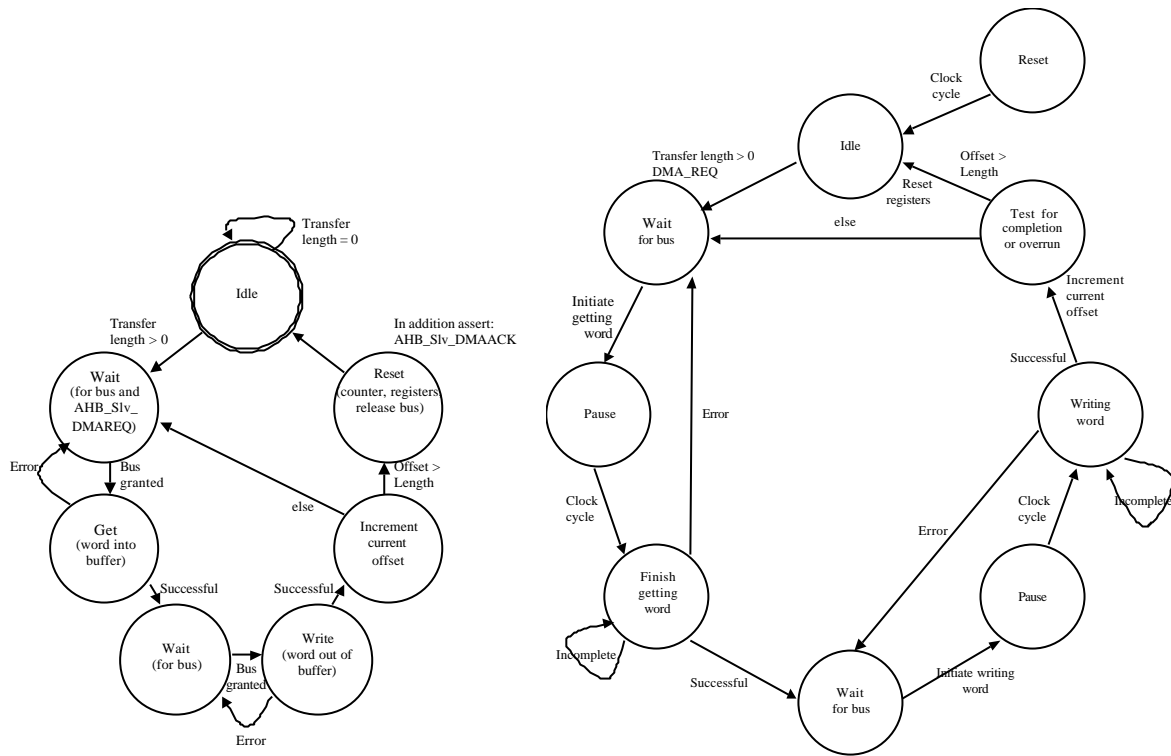


Figure 8: Original(left) and final(right) state diagrams of timing and control module

Although the final state machine has more states the overall performance is actually much improved. The reason for this is that the original design needs additional states adding to make it practical – increasing the number of states in the loop 10. The final design includes these additional states, but others such as the reset and completion testing state can be removed by moving their action to the state transition (for an example see section 2.4.1 –the register implementation). The original scheme had no provision for delaying the AMBA bus read/write cycle which is an important part of the AMBA specification, this has been included in the final design with additional state transitions.

The lone state on the top right of the final scheme is only entered when the DMA controller is physically reset by the AMBA bus global reset signal HRESET.

2.4.4 Top Level

The top level of the DMA controller has no VHDL processes, it is simply composed of all the modules needed for the controller and the connections between them - internally and externally. Hence this module was very easy to write. However constant changes were required whenever the interface to a module was modified. To begin with some AMBA bus buffering functions were present in this module – however for improved coherency they were moved into the logical buffering module.

2.4.5 Arbiter

This was the last module to be coded. The logic is a simple combinatorial scheme, which is sufficient for this project. In reality the arbiter may have to be improved by adding synchronisation to the AMBA bus system clock. For this reason the system clock is connected to the module as an unused input.

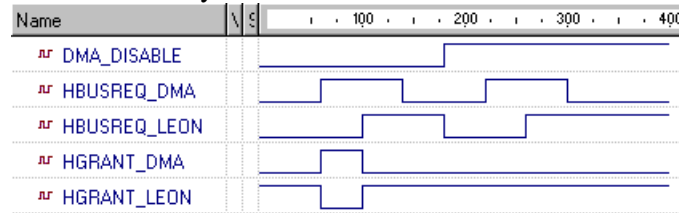


Figure 9: Intermediate test results for AMBA bus arbiter

The arbiter should only grant bus access to the DMA controller when LEON is not requesting the bus, when the controller is not disabled, and when the controller is actually requesting access. All possible input combinations are tested and module performs in this manner.

2.5 Mid Project Report

The mid project report was submitted as per instructions and appeared to get a good response. As additions to the mid project report a then current version of the project specification was attached. At the informal presentation an addendum to the report was submitted with details of the work carried out between the writing of the report and the presentation. During the meeting some improvements to the project were suggested, these were incorporated into design as described.

2.6 Testing

The following overall system test cases were tried:

- Basic single-word DMA transfer
- Basic multi-word DMA transfer
- Multi-word DMA transfer exceeding address space limits
- Multi-word DMA transfer writing to address zero
- Transfer errors
- Transfer delays
- Consecutive transfers

A full description and analysis of the results obtained from these tests can be found in Section 3(Results). VHDL test programs which generate these test vectors may be found on the disc attached to this report.

In addition intermediate testing of each system module was performed. An analysis of these tests, and their results is shown the relevant sections.

2.6.1 Back Annotated Timing

As a final assessment of the DMA controller system it was planned to retest by using 'back annotated timing'. Simply put this means testing the design incorporating simulated logic delays as in a real FPGA device. This is done by synthesising (or compiling) the design by using the specialist tools supplied by the FPGA manufacturer (in this case Xilinx Foundation) and having these tools also generate simulated timing responses and logic behaviour in the form of a special VHDL file. These can then be tested using the same testbench as the original design. If all is correct the results should be near identical.

Unfortunately this final proof of the DMA controller proved impossible to perform. The tools produced the simulation output as expected, but with the following error message generated in the log file:

```
WARNING:Anno:26 - NGDANNO found physical components for which 100 percent
back-annotation is not possible. (These components are listed below.) Some
reasons these components may not be fully back-annotatable include:

1. The logic was replicated during physical mapping.

2. MAP was directed to optimize the logic through use of the -oe or -os
option, or the OPTIMIZE or OPT_EFFORT design attribute.

3. The component's configuration implies a more complex delay model than can
be accurately represented in the original design logic. An example of such a
configuration is an XC4000-family CLB containing both carry logic and
multiple flip-flops.

Simulation models for the following components will be constructed from the
NCD netlist. Signal names buried within these components will be lost.
N756
Timing_and_Control/AMBA_HADDR_tri_enable<1>
Timing_and_Control/C1/N9
Timing_and_Control/CURR_STATE<2>
Timing_and_Control/CURR_STATE<5>
Timing_and_Control/CURR_STATE<7>
```

Which seems to indicate that the Xilinx tools had problems with back-annotating the designs state machine (the signals mentioned in the last part of this extract being part of this).

When test vectors were applied to the back-annotated VHDL the outputs produced were either undefined or unknown. This was investigated to remedy the situation. Firstly a simple combinatorial design was back annotated (the AMBA bus arbiter) and tested using relevant test vectors. This back-annotated combinatorial design worked correctly. The complexity was then increased by attempting to back annotate a simple design containing d-type storage elements, again this worked with no issues. As a final attempt at back annotated testing of the complete design, the VHDL generated by the Xilinx tools was modified to bring the global set/reset and tri-state nets (GSR/GTS) out as external signals. These were toggled directly by the test bench. This direct reset of the design's storage elements caused some signals to become defined, however proceeding with tests was still impossible.

It could be conjectured the issues with back-annotated testing relate to the synthesis of state-machines from VHDL. Simple storage and logic was correctly back-annotated. It is possible the circular nature of state machines means some internal nets cannot be reset – remaining undefined during a simulation. By separating state transitions and signal assignments into separate processes, and by directly implementing tri-state buffers (rather than implying them) it is likely these problems could be solved. This is due to the inability of current synthesis tools to understand these implied features.

2.7 Final Report

As soon as the test results were obtained this final report was compiled.

3 Final Results

Once the project development seemed to be complete the following major tests of the system functionality were run. For a key to the signal names, please refer to section 6.

3.1 Basic Test

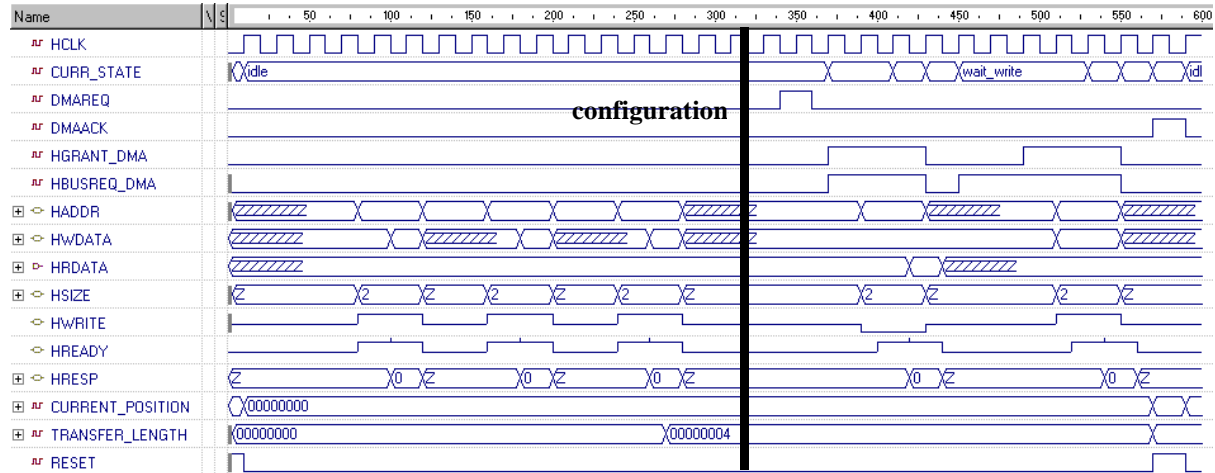


Figure 10: Results of DMA controller basic functionality test

In this test a single word DMA transfer is setup between two addresses in the middle of the possible address space of the AMBA bus. The signals up to around 300ns show the transfer configuration registers being successfully written to by the test bench. For instance the value of TRANSFER_LENGTH is increased from 0 to 4 after this register is set up at around 250ns.

At 340ns the transfer is triggered by pulsing the DMAREQ line. This causes the state machine to begin it's series of state changes until the transfer terminates successfully. When use of the bus is first requested it is instantly granted. The bus read lasts until 440ns. A straightforward read is tested, with the slave device responding instantly and with no errors.

The DMA controller correctly releases use of the bus, and at 450ns it re-requests the bus for the writing to the destination memory address. This time the bus is not instantly granted by the test bench, but is delayed until 490ns, the DMA controller machine correctly waits for this to take place. The DMA controller then writes the buffered value out (very similar to the procedure of reading data).

After the write is completed (at 550ns) the DMA controller accurately assesses that the DMA transfer has completed, and resets its internal registers ready for a new transfer.

3.3 Address Bus Overrun

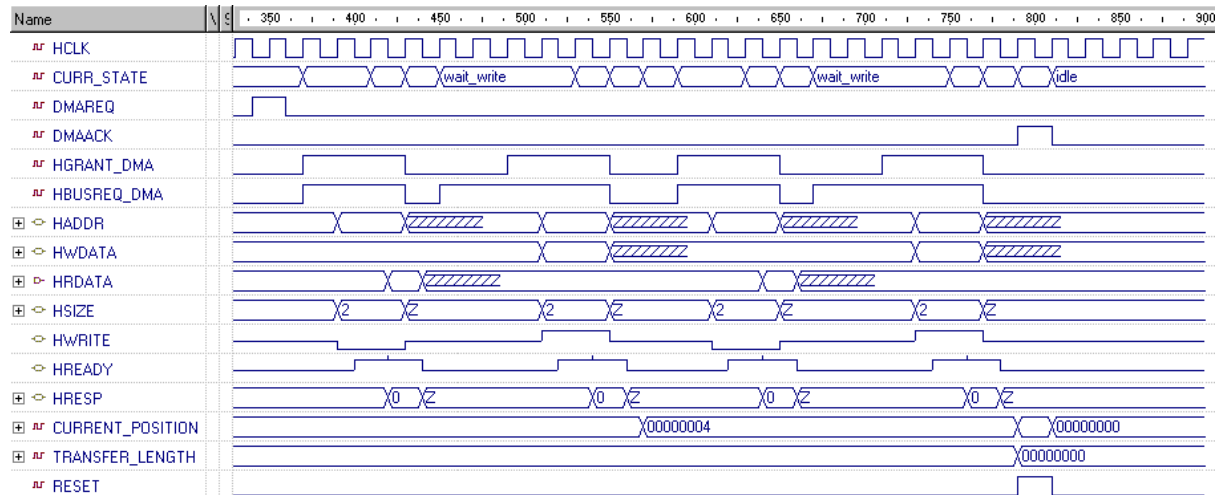


Figure 12: Results of AMBA bus overrun test

The next two vectors test the behaviour of the DMA controllers address over-run detection. The first 340ns of the test vector again looks the same as for the basic transfer – so it is not shown in the timing diagram. The differences are in the way the registers are set up - the transfer is a 3 word transfer, the last word of which will just overrun the xFFFFFFFF address on the AMBA bus. The first two words are transferred (as in the previous test). The DMA controller detects that last word will be written to an address exceeding the maximum allowed by the AMBA bus and terminates the transfer before time, as per its specification.

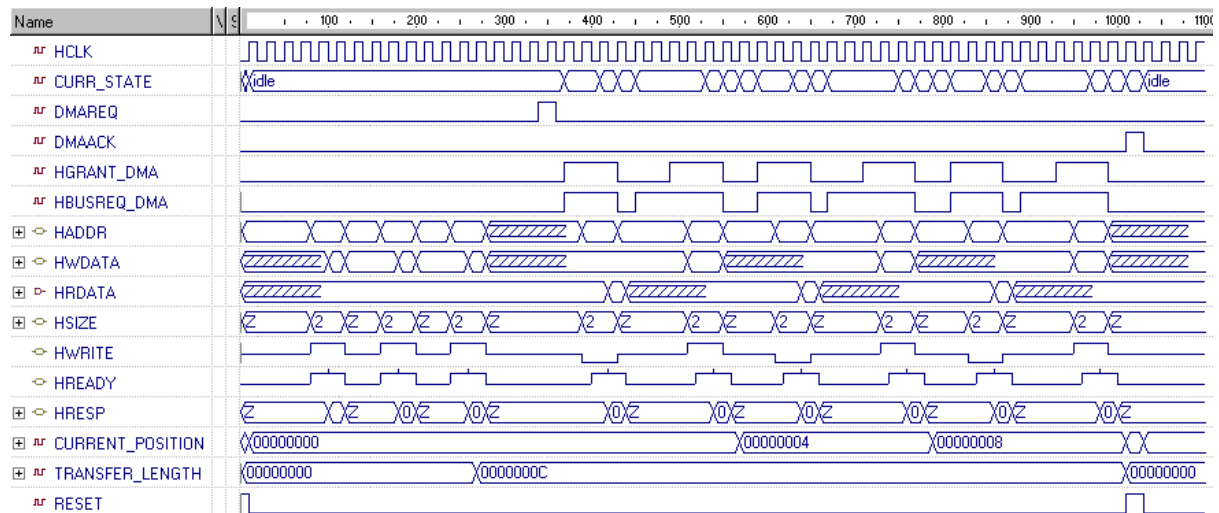


Figure 13: Results of AMBA bus overrun fix test

To ensure that the code used to detect a bus overrun does not interfere with transfers to low addresses, this test writes to the first 3 AMBA bus addresses. Again the first 350ns of the vector sets up the transfer. The rest of the test proceeds identically to the multi-word test, showing that legitimate transfers are unaffected by the overrun detection mechanism. One interesting point of note is that the base address of the configuration registers is x00000000, and so they are overwritten by this test. The first word written changes the source base address, so for the next word the DMA controller reads from somewhere different. The next word written changes the destination address so that the final word goes to another address, not affecting the transfer length register. Although this is clearly a bug in the test vector it verifies that the registers (AMBA slave) and state machine (AMBA master) operate to the same interpretation of the AMBA bus specification.

3.4 Delayed Transfers

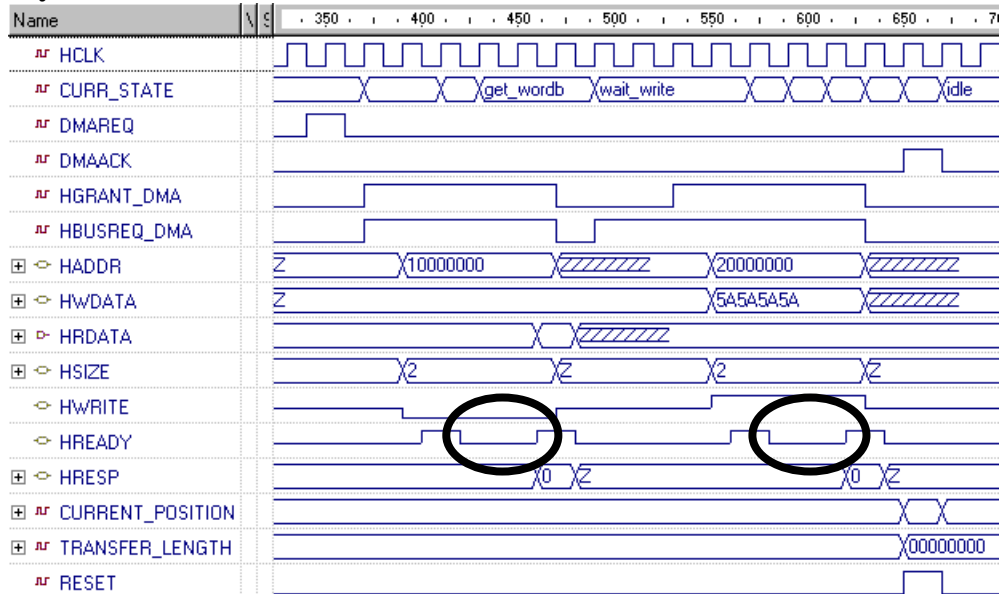


Figure 14: Results of AMBA bus delayed transfer test

This test tries the capability of AMBA bus slaves to delay or extend a transfer by not activating the AMBA bus HREADY line. The timing diagram above shows what happens when the test bench delays setting HREADY by 40ns. This delay is highlighted by the black circles. The result is that the DMA Controller state machine remains in the same state – waiting for the source/destination to put/acknowledge the word on the bus. The behaviour is as per the AMBA specification (ref: 4). The first 350ns of the test are the same as in previous tests – the only change is the delayed HREADY.

3.5 Errors During a Transfer

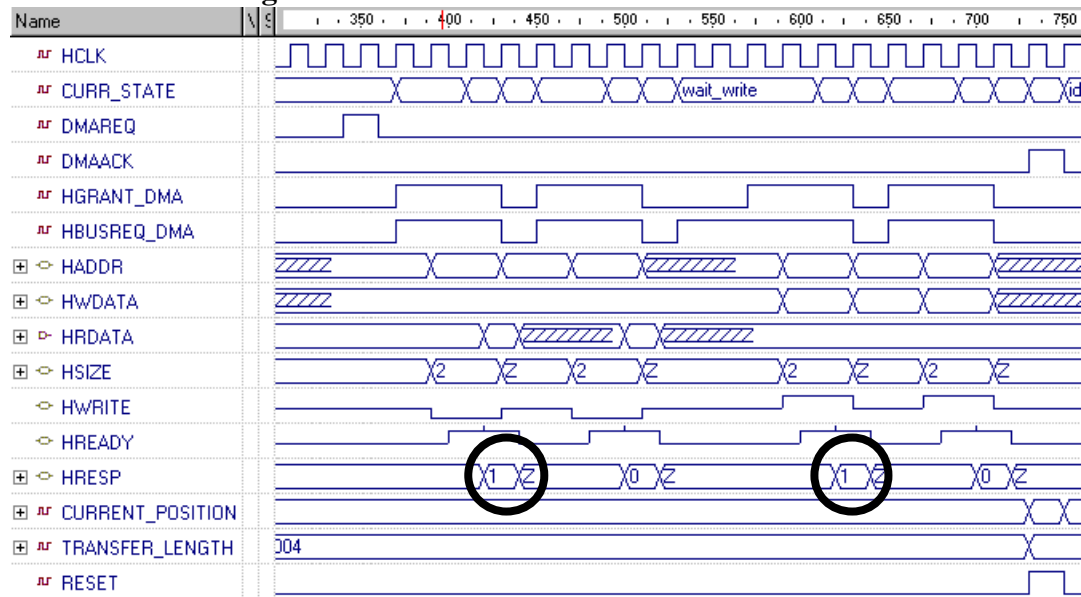


Figure 15: Results of AMBA bus delayed transfer test

In a real system it is likely that some transfers will occasionally generate errors. The AMBA bus specification (ref 4) has a method of indicating when such a transfer has taken place using the two line HRESP bus. If the transfer has been successful the slave device sets this to '00', any other state indicates an error of some kind. For these controlled tests genuine errors are not possible, however the test bench used for this test indicates an error on the first read and write operation (highlighted on the diagram above). As may be seen from the results this causes the DMA controller to redo that stage of the transfer, as in its specification (ref 1).

3.6 Multiple Consecutive Transfers

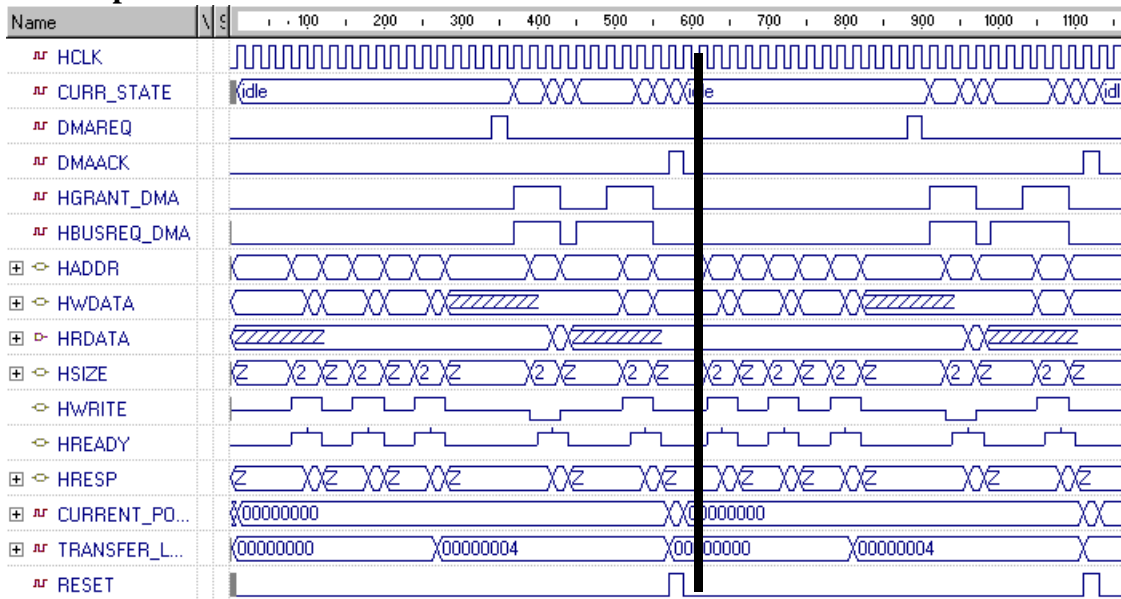


Figure 16: Results of multiple the consecutive transfers test

Here the basic test bench is run twice, except for the initialisations. The test shows that the DMA Controller resets itself properly between transfers and can go on to perform other transfers successfully. The vertical line on the diagram shows the point where the previous DMA transfer has completed, and a new transfer is set up.

4 Improvements

4.1 Burst Mode

The DMA controller produced as part of this project does not use any RAM type memory devices for intermediate storage/buffering. This has the advantage that less FPGA resources are used, meaning more are available for other modules (probably cache for the LEON processor). Design complexity is not in reality reduced by taking this approach. However, by using large intermediate buffering for the data being transferred module performance could be at most doubled – by making use of the burst mode transfer capability of the AMBA bus.

The operational principals of this technique are similar to that implemented, in that data is read from the source into the buffer, and then out to the destination. The difference is that larger amounts of data than just one word are transferred each iteration. Despite these similarities, there would need to be significant changes to the design to add this functionality, probably using additional states for a burst transfer.

4.2 Transfer Queuing

In many modern DMA controllers there is the option to configure multiple transfers into the controller and execute them consecutively. It is usually possible to add to the list of whilst DMA transfers are in progress. This feature should not be too difficult to include in the design, probably by use of FIFO buffering for the 3 different registers holding the transfer details. Obviously these registers are required to be write only, but this is currently the case – so no functionality will be lost. There will also need to be some checking to ensure these FIFOs do not over flow.

4.3 Multiple Transfer Configurations

It is quite likely the DMA controller that has been developed will need to perform many different, yet repetitive transfers. This idea for extra functionality should help improve the efficiency with which these transfers can be realised.

Essentially multiple transfers are configured into different sets of registers in the DMA controller. Transfers are then triggered as previously by use of a DMAREQ line. However there will be multiple DMAREQ lines – each different line triggers a different pre-configured transfer. Once the transfer is completed the details are retained (rather than lost), and it is possible to re-trigger it later on. If other transfers are requested whilst one is still being processed they are queued for processing later. It may even be possible to implement a multi-level priority scheme for the different transfers.

Efficiency is improved as details of each peripherals transfer do not have to be loaded each time and the DMA transfers may be executed instantly, essential in a real time system.

4.4 Maximum Transfer Retries

This is a simple feature to add. Currently the DMA controller continually retries reading and writing to system memory until it is successful. The only way to stop this is by resetting the DMA controller. This strategy is appropriate for its simplicity and does not require a method of indicating to the microprocessor the different possible errors which could occur. It is simply assumed that eventually all read and writes will be successful, which is valid in a closed system. In order to improve this the maximum number of times a read or write transfer is extended or generates an error could be limited. If these limits were exceeded then error messages would be generated in some scheme to be defined.

4.5 Use AMBA APB for Control

As maybe seen from the AMBA bus specification (ref 4) there are three different variants of the bus. The AHB variant is used by the LEON microprocessor and memory controller, and was therefore a natural choice for the project. However for some simple peripheral devices the APB derivative of the bus is used. One characteristic of APB is that all devices may write to each other. If the control registers for the DMA controller were to be converted to APB then slave devices would be able to configure their own transfers, without requiring use of LEON at all. This would mean transfers would be possible without impacting LEON processor time at all. By combining this with 4.3 a very powerful DMA system could be instigated.

5 Conclusion

5.1 Achievements

Over the course of the project the DMA controller has been coded, specified, documented and tested. In the mid term report it was these same objectives that were set out as being required for a successful completion of the project. Furthermore, as can be seen from this report, these objectives have also been met with a degree of proficiency and success. The DMA controller designed fully meets the specifications and will work for its intended application. Furthermore the documentation produced will assist with future developments of the project. The design has sufficient functionality to make useful and usable, however unspecified advanced features have not been added to the design as it was felt that properly tested problem free operation within a given specification, was more important than incomplete functionality which does not quite work correctly.

5.2 Learnings

In order to complete this undertaking many new skills had to be gained. These included learning more advanced VHDL programming techniques, and becoming proficient at translating a digital logic specification into VHDL. Of course not all VHDL may be synthesised onto an FPGA and lots of knowledge was achieved in this regard as the project proceeded. Testing digital logic using a VHDL test bench was also something completely new learnt during development. The chief aspect of VHDL that knowledge was gained over the project was the implementation of state machines in a style that may be synthesised by FPGA tools.

From the non technical perspective competence has been developed in the areas of self motivation and learning. In addition problem solving and documentation have been practised. These are some of the most important things to come out of the task and will be useful in the future.

5.3 Personal

A strong sense of achievement was felt when the project was finally completed. Despite the hard work put into the DMA controller the quality of the finished output makes it worth while. The DMA controller need not be another dead-end student project, but could be made use of in a real world system. The various problems encountered were interesting challenges to solve in themselves and it was good that they were all resolved. If this project were to be recreated greater emphasis would be put on testing the design, possibly with a view to performing back annotated testing. But as it stands the author feels a good overall balance was struck.

6 Signal Name Key

<i>CLK</i>	- Clock
<i>CE</i>	- Clock enable
<i>CURRENT_POSITION</i>	- Counter output holding the current position in the DMA transfer
<i>CURR_STATE</i>	- Internal signal to timing & control module indicating the current state of the DMA controller state machine
<i>DMAREQ</i>	- Requests a DMA transfer
<i>DMAACK</i>	- Strobed by DMA controller to indicate the finish of the DMA transfer
<i>H?????</i>	- AMBA bus signal – see project specification for full details
<i>HADDR</i>	- AMBA bus address bus
<i>HCLK</i>	- AMBA bus synchronisation clock
<i>TRANSFER_LENGTH</i>	- Register output holding the length of the current transfer
<i>RESET</i>	- Signal generated by the DMA controller state machine resetting the internal registers

7 References

- 1 AMBA Bus DMA Controller Specification, Draft 1.03, Gareth Morris 2000
- 2 Final Year Projects 2000/2001, Chipsat Related Projects, Tanya Vladimirova 2000
- 3 LEON-1 VHDL Model Description, Version 2.2a, ESA
- 4 AMBA Bus Specification, Revision 2.0, ARM Ltd.
- 5 The Student's Guide to VHDL, 1st Edition, Peter J. Ashenden 1998

8 Glossary

<i>AMBA</i>	- Advanced Microcontroller Bus Architecture. A system bus standard devised by the ARM corporation for embedded applications.
<i>ARM</i>	- Manufacturer of RISC processors for embedded applications.
<i>DMA</i>	- Direct Memory Access. A method whereby peripherals can access main system memory without using processor resources.
<i>ESA</i>	- European Space Agency. Organises Europe's space affairs
<i>FPGA</i>	- Field Programmable Gate Array. RAM based Semiconductor device which is programmable for different digital logic circuits
<i>LEON</i>	- Microprocessor adhering to the SPARC V8 standard written in VHDL by Jiri Gaisler of the ESA.
<i>Logic analyser</i>	- Instrument for monitoring multiple input and output digital logic circuits
<i>RISC</i>	- Reduced Instruction Set Computer.
<i>SPARC V8</i>	- Scalable Processor ARChitecture Version 8. A standard for RISC based microprocessors devised by Sun Microsystems.
<i>Synthesis</i>	- The automatic process where by VHDL code is transferred into an FPGA design. Requires the VHDL to be written a certain way.
<i>VHDL</i>	- VHSIC Hardware Description Language. A software language for describing digital logic behaviour
<i>VHSIC</i>	- Very High Speed Integrated Circuit
<i>Virtex</i>	- State of the art FPGA made by Xilinx
<i>Xilinx</i>	- Semiconductor corporation manufacturing FPGAs

9 Special Thanks

Hans Tigglar - project supervisor
Jiri Gaisler - formerly of ESA
Dr Tanya Vladimirova - project supervisor

Appendices

Specification

Time Plan

Source Code (diskette)