Implementation of a VME bus to internal bus bridge FPGA core

Xavier Revés
Universitat Politècnica de Catalunya
Departament de Teoria del Senyal i Comunicacions
Jordi Girona 1-3 08034
Barcelona (Spain)
+34 93 401 71 95
xreves@xaloc.upc.es

Antoni Gelonch
Universitat Politècnica de Catalunya
Departament de Teoria del Senyal i Comunicacions
Jordi Girona 1-3 08034
Barcelona (Spain)
+34 93 401 71 97
antoni@xaloc.upc.es

J. L. García
Universitat Politècnica de Catalunya
Departament de Teoria del Senyal i Comunicacions
Jordi Girona 1-3 08034
Barcelona (Spain)
+34 93 401 71 97
garciam@teleline.es

Ferran Casadevall
Universitat Politècnica de Catalunya
Departament de Teoria del Senyal i Comunicacions
Jordi Girona 1-3 08034
Barcelona (Spain)
+34 93 401 64 25
ferranc@tsc.upc.es

ABSTRACT

Since several years ago a wide variety of FPGA-based cores have been appearing all round. This is fruit of the versatility of these devices due basically to the amount of system gates that can implement. Many of the cores are in the market as IP (Intellectual Property) but many others implemented by an engineering team when developing a platform never come out. In this paper we present an FPGA core implementing the required bridging between the standard VME bus and a defined internal bus. This core, while using relatively few resources, has a high level of performance and supports most of functions defined in the standard. This core, when inserted into a system will significantly reduce the complexity of interfacing a complete VME backplane because it can map the elemental behavior of the internal bus to the multiple VME accesses.

Keywords

FPGA, VME bus, microprocessor, interface.

1. INTRODUCTION

The VME bus [1][2] was first introduced in 1981 coming from the architectural concepts of the VERSAbus developed by Motorola in the late 1970s. It established a framework for different computer architectures that can implement single and multiprocessor systems. The VMEbus specification defines an interfacing system used to interconnect microprocessors, data storage, and peripheral control devices in a closely coupled hardware configuration.

The standard soon became an interesting interfacing system because of its robustness, versatility and possibility of linking platforms built with different state-of-the-art. The asynchronous nature of the VME signals give the possibility of interfacing two *fast* boards and also the possibility of interfacing one board with another much slower. This capability translates to a longer life of the platforms available. For instance, an A/D and D/A board useful for a low/medium speed application requiring at most 1 Msamples/second will do with a 5 Mbytes/second interface, while another platform requiring 10 Msamples/second may need a 50 Mbytes/second interface. Both platforms, even having quite different transfer rates, can be integrated under the same VME backplane. Following with the example, the first board may be at this time 10 years old while the other, together with the main system processor, may be only 1 year old. This gives us an idea of the persistence of the platforms over a VME bus.

Obviously the terms commented above are important when choosing the VME bus to build the system but many others can be found depending of the interest of every one. It is not our aim to describe the features and drawbacks of an VME system. In the next sections a high performance master and slave VME interface bridging to a simple internal bus will be described. This interface can be completely implemented into an FPGA [3] and provides most of functions established in the standard.

2. MOTIVATION OF THE WORKING

In our research laboratory a reconfigurable hardware system working under VME bus was built. This system is based on a set of FPGAs ready to perform different kind of tasks (signal processing, I/O communication, system management, etc.). Some of the FPGAs on the board require access to a high performance data transfer bus for the correct execution of the tasks generally assigned to those FPGAs. This bus should be viewed by the VME processors and should be able address other cards on the VME backplane. Moreover, this bus must be simple enough to use only few resources into the FPGAs interfacing to it leaving as much resources as possible for the application-related task assigned to the FPGA.

Several ASIC bridges to VME bus are present in the market. Some give a quite simple interface but sometimes also reduced possibilities (slave only) and others allow more complex tasks with the drawback of more complex interfaces (e.g. PCI bus). Moreover of the bridging interface, more logic was required to accomplish with the requirements of the platform. So it was mandatory adding some glue logic (maybe an FPGA) to the ASIC. So the question is easy, why do not using a single FPGA implementing both the bridging and the glue logic?

For those and other minor reasons, a customised bridge between the VME bus and the internal bus was implemented. Into the same FPGA could be placed all the logic required for the platform accessed, as it will be seen later on, through the register set defined in the interface. This register set is in part definable and represents the main customisable portion of the bridge as it will depend on the requirements of every platform. In general this set of variable registers will not be required if only bridging is wished but they have a permanent address range reserved for an eventual future use.

3. THE INTERNAL BUS

The internal bus has been called IBUS (**S**ynchronous **B**urst **U**ser **I**nterface). As its name indicates, it is a synchronous bus, that is, data are sent and/or received at every validated clock edge. Address lines and data lines are multiplexed in time to reduce the number of connections required among the different elements connected to the bus. Although the use of IBUS is mainly for fast data transfers, some capabilities (as interrupt request and service) have been added to get a more complete set of features.

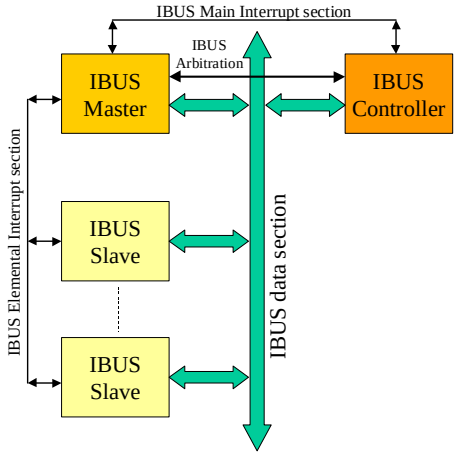

Figure 1. Internal Bus Hierarchy

A total of 40 to 44 signals are enough to completely interface the bus where 32 are for address or data. Only 4 lines are used for control during a data transfer process. The rest are destined to initialization and interrupts and its use depend on the location into the IBUS hierarchy. The way this hierarchy is defined can be observed in figure 1. There exist one bus controller, which can be a bus master, one bus master, which must be also slave, and then a set of slave-only interfaces. No electrical characteristics have been defined for the IBUS, then the designer must take into account bus delays when defining the operating frequency. Bus controller has the task of deciding which master owns the bus, avoiding any kind of contention.

The address size represents an addressing range of 4 Gwords being each word 4 bytes wide. A typical data transfer is divided into three sections: addressing phase, data phase and release phase as it is shown in the figure 2. The first phase consists of a minimum of 4 clock cycles where the master addresses the slave and waits for a recognition. Initial placement is framed with a control signal After these minimum 4 cycles the data flow starts with the first possible word to transmit of the first burst. Each clock cycle that has been validated by the sender of data (the master in write cycles and the slave in read cycles) represents a word of the burst. The first burst ends when 16 words (64 bytes) have been sent/received. After this burst, if more data have to be sent/received, a new data-only handshake of at least 2 cycles follows. The number of bursts that follow the first one has no limit if the bus is not requested by another possible master. To finish the transfer, a minimum of 2 cycles must be left before starting a new one. This is necessary specially when the master decides not completing the current burst because it has no more data or it does not wish more data and the 16 words of the burst have not yet been reached.

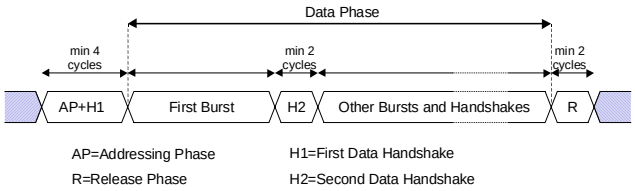

Figure 2. Data transfer sequence defined for IBUS

The addressing phase will determine where data is sent to or where data is read from. When dividing the complete 4 Gword addressing range, there are several rules to follow to make easier and simplify master and slave operation as well as address handling. Consider these main rules. First of all, 2 different slaves cannot have their spaces mapped into the same 256 kwords region (regions are considered starting from address 0h). One slave can have as many regions of 256 kwords as required. Moreover, accesses to slaves are circular, that is, when the data range of the slave has been all read/written and more bursts are initiated data are read/written from/to the beginning of that slave range.

The burst size is strongly related to the family of FPGAs used. The core presented has been implemented over the Xilinx XC4000 family. This family can implement small quantities of RAM internally. One possible configuration is specially suitable to build FIFOs and the minimum depth of the RAM is of 16 different addresses. More deep configurations of RAM can be achieved through logic multiplexing but a maximum of 64 bytes per burst has been considered a very interesting quantity as it is a balance between large amounts of RAM into the FPGAs and data transfer reduction due to burst handshakes.

The quite simple bus structure described requires only about 60 CLBs (that slightly depends on the address range width to handle because wide edge decoders in family XC4000 are used) including the 64 bytes FIFO (16 words of 32 bits) for slave operation. For master/slave operation, the part corresponding to IBUS can be implemented using about 80-90 CLBs. This amount of CLBs has been extracted from the core presented in this paper and can be different depending on the master and slave interfaces to a CPU, memory, etc. Anyway, in most of cases where the bus will be used (for instance, the platform mentioned before) only slave functions are required to sink data. That amount of CLBs for slave is about 10% of a XC4013 but only 5% for XC4036 or going to the new Xilinx Virtex family only about 15-20 CLBs (Virtex family CLBs have twice as much logic elements than XC4000 family CLBs but has no wide edge decoders) and some Bloc SelectRAM+ will be required. Then it can be concluded that IBUS meets the area requirements requested.

What concerns to data transfer speed, as IBUS is a synchronous bus, it is easy to calculate the peak performance. Measured in bytes per second, maximum transfer rate will be of 4 times the

clock frequency. Taking into account the additional 2 cycles per burst for handshaking, the maximum transfer speed will be of about 89% of $4*f_{clk}$. But the clock frequency can be very high because of the protocol used. Due to the nature of transactions, where no flow control exist in a burst except for one bit used by the sender to validate data, delays to be accounted in the bus are those in between the flip-flops of the sender and the flip-flops of the receiver. It's not difficult to achieve frequencies of 40-60MHz if the printed circuit board is properly designed and using an FPGA with a moderated speed grade. This system speed can be exceeded when using fastest FPGAs. Having clock speeds relatively high in a large area (10 to 20 centimetres) requires a good clock distribution. For IBUS an star clock distribution with one driver per branch has been chosen with all branches of the same length to minimise skew.

4. VME TO IBUS BRIDGE

4.1 Functional Diagram

The functional diagram of the bridge is as it is shown in the figure 3. The main blocks are the master and slave blocks for VME and IBUS respectively, the bi-directional FIFO to send data from one bus to the other and the register area. In this register area there is information about how the bridge has to operate. This information not only concerns to data transfer interfaces but also to the aspects of automatic mapping of cycles (both directions), error handling, interrupt management, identification, addressing and DMA (Direct Memory Access) transfers. Also some registers that depend on the environment where the bridge is placed can be located in this block. Its behaviour should be adjusted to the general register behavior.

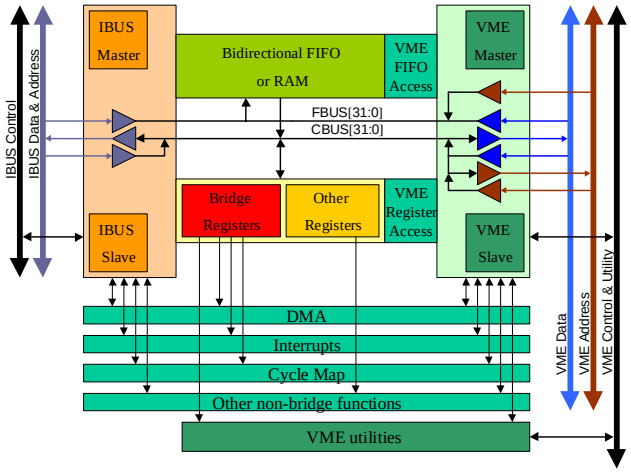

Figure 3. VME to IBUS block diagram

All the functions, registers, memory and buses present in figure 3 can be placed into a XC4013 FPGA from Xilinx. Into the same FPGA some additional logic concerning the specific platform could be placed. The total amount of CLBs used is of about 520 that represents an occupancy of 90% of resources available. Without the particular registers and associated functions it can decrease to about 450 CLBs. Note that the design has been properly mapped to the underlying structure of X4000 family to get the maximum density without compromising the routing task. To get an optimum performance in terms of area and speed, as most of resources were used to store configuration bits and data related to them, a combination of registers and RAM was done. Also all the internal resources that could be shared among different operation modes were carefully designed to avoid having redundant logic and save space and even sometimes increase speed. For instance, FIFO block is used as FIFO but also is used as cache RAM to allow the correct mapping of several types of VME cycles to the data transfer structure of the IBUS. The less expensive blocks in terms of area are the control state machines for every function.

One important aspect to be highlighted is the clock used by each one of the blocks present in figure 3. Registers, FIFO and IBUS control blocks run at IBUS clock frequency while control machines related to VME interface run at twice that speed. This is because VME signals are asynchronous with IBUS ones. To provide a good time resolution and performance of the interface, that has to move data from/to registers or internal RAM synchronously, is interesting running this part with such a clock. To have a completely synchronous system exist 2 possibilities. One is providing only the fastest clock to the FPGA and activating IBUS related areas only half of the rising edges of the clock. Another possibility is to provide 2 input clocks with aligned edges. One of them will have twice the frequency of the other. This last has been the option chosen although it requires an external device to generate those clocks. The main reason of that is the distribution of clock along the IBUS described before. This problem will be overcome when translating the bridge to Virtex family because it incorporates DLLs that will allow doubling de clock frequency of the IBUS for internal use with virtual 0ns delay.

4.2 Some Special Blocks

In the figure 3 appear several features of the bridge to highlight. First note the presence of DMA (Direct Memory Access) related functions and their corresponding registers. The DMA block will be able to start read/write cycles in both masters and transfer data from one side to the other. The rules that govern the DMA block behaviour are the same used to translate cycles from one side to the other and addressing is managed accordingly to IBUS addresses. Start block address for IBUS and VME sides and block length are provided into registers area.

An interrupt manager can also be seen. It will do the tasks of translation of interrupt requests from IBUS and eventually activate the VME master to give service to interrupts. VME bus has 7 different levels of interrupts but only 2 can be activated at a time by IBUS requesting structure. These 2 levels can be selected among the 7 valid ones and modified when required writing to the corresponding register. Also internal registers are used to store the interrupt status word to be supplied to the corresponding interrupt handler over the VME bus. In the reverse sense, the VME master has the capability to handle non-masked interrupts and store into internal registers the status word. Additional interrupt capabilities are associated with extra logic added to the bridge.

One important feature is the automatic mapping of the different cycles present in the VME bus. As IBUS has only one transfer mode and VME has several ones, some kind of translation has to be done. To go from IBUS to VME, the address and cycle translation is performed in conjunction with the known capabilities of VME slave addressed by the bridge VME master. These capabilities are stored into registers and selected as a function of the address generated by the IBUS master. Up to a maximum of 16 different slaves with different capabilities can be

addressed. Capabilities include data and address width supported by the slave but also other parameters can be determined, as block transfer capabilities. These 16 translated addresses can point to the same slave or grouped in any way. The maximum addressing range supplied is of 16 Mbytes over VME bus (1 Mbyte each address) but the 16 Mbytes window can be modified writing to the associated register. In the figure 4 the translation mechanism and a possible mapping used to go from IBUS to VME is represented. Note that an internal IBUS window is used to access VME resources. This window can move along the whole IBUS address range.

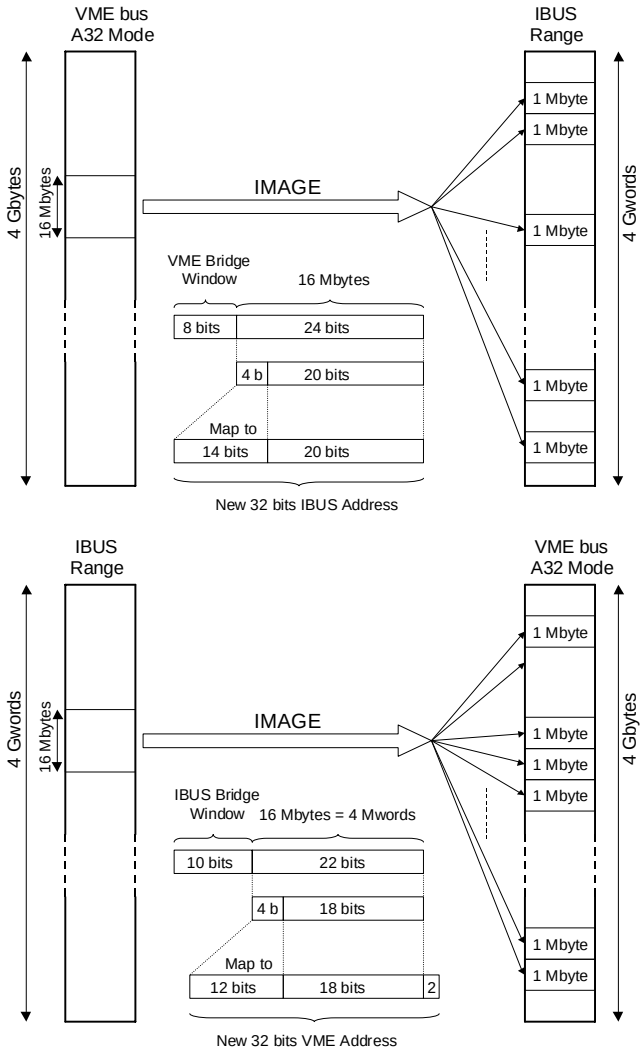

Figure 4. Address translation from IBUS to VME and from VME to IBUS

When going from VME to IBUS the translation is done in a similar way (see figure 4) but without requiring a capabilities register (like the one mentioned before) as IBUS has a single operation mode. Some kind of VME cycles started by the current master cannot be directly translated to an IBUS cycle as data width resolution into IBUS is 32 bits and VME resolution is 8 bits. For these special cycles, specially in write modes, data on IBUS are internally stored into the bridge and resent to the addressee to grant data integrity. The VME window used to access to IBUS can be chosen writing to the dedicated register but the bridge has the possibility to read a hardware address (selected through switches or jumpers) before connecting to the VME backplane.

Another important block to highlight is the *Utilities* one. It will not be entirely required in many cases as it implements some special tasks assigned to a determined position into VME bus. This feature will allow managing VME bus arbitration, system reset and interrupt chains. But it will be generally present as these functions of special location are mixed with the related ones that must implement every system connected to the backplane, although they will be deactivated.

Registers required for this bridge version occupy 170 bytes of the total 1 kbyte of internal addressing organised in 32 bits words but accessible with one byte resolution. Most of them are stored into RAM to minimise area. The remaining 830 bytes can be used for any other thing while there is room enough into the FPGA. For instance, in the platform used as example 10 additional bytes stored in flip-flops are required to perform several actions.

4.3 VME Features

Part of the complexity of the bridge relies on the VME complexity to be hidden to IBUS. The standard over which the bridge has been built is VITA 1-1994 (VME64) that specifies a framework for 8, 16, 32 and 64 bits bus computer architectures.

The bridge being presented has internally implemented most of the features described in the standard mentioned before. What concerns data transfer, VME bus protocol is very straight forward. The master puts addresses onto the bus, delays a minimum of 35 ns and then asserts address strobe (AS*). For a write operation, the master puts data onto the bus, delays a minimum of 35 ns, and then asserts one or both of its data strobes (DS0* and/or DS1*). All the slave cards on the bus monitor the addresses and know that the address is valid after AS* assertion. In a write cycle the selected slave must read data of the bus after DS0* and/or DS1* and assert data acknowledge (DTACK*) to signal that the data has been captured. Read processes are similar changing dada direction. Although this process is quite simple, there exist a lot of variants concerning addressing modes, data modes, transfer types, etc. That is what makes VME bus more complex, and more if you consider that a master or slave must support all transfer types classified in the VME standard under the one wished. For instance, if you wish to transfer data using 32 bits words you must be able to use also transfers using 16 and 8 bits.

The bridge being presented supports a large set of transfer types as the minimum desired level had some under it. Address width handled by the VME slave is only 24 (A24 or standard mode) or 32 (A32 or extended mode) bits modes, but as a master can generate 16 (A16), 24 and 32 bits addresses. It must be taken into account that VME bus uses 6 additional lines to specify different addressing modes.

Data transfer capabilities are wider. As slave, all modes of 8 bits (D8(OE), D8(O), D8BLT), 16 bits (D16, D16BLT), 32 bits (D32, D32BLT) and 64 bits (D64 or MBLT) are supported together with special cycles as unaligned transfers (UAT) or read-modify-write cycles (RMW). This gives enough flexibility to interact with any kind of master. Observe that 32 and 64 bits modes can be directly mapped to IBUS but 8 and 16 bits modes will require some additional translation mechanisms. Working as a master the set is slightly reduced as IBUS cannot generate unaligned transfers or

read-modify-write. Then UAT and RMW cycles are not implemented in the bridge VME master.

Access to internal registers from VME can be done through an A24 addressing mode as A32 addressing mode is reserved to bridge to IBUS. But the set of registers has also been mapped to the Configuration ROM / Control & Status Register (CR/CSR) area accessed by means of an special addressing mode (using 24 bits address). The Configuration ROM / Control & Status Registers provide a mechanism for manufacturer identification, board identification, automatic board initialisation, board test and board configuration. All the mandatory fields required to correctly incorporate CR/CSR feature are included into the bridge.

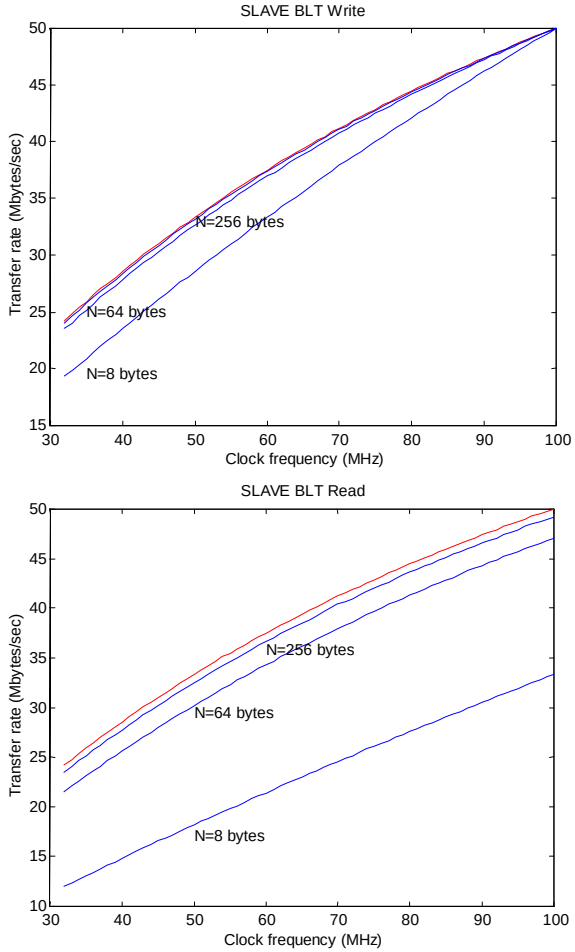

Figure 5. VME Slave BLT performance

One important point is the data transfer performance that can be achieved by the bridge. Before has been presented the IBUS interface where determining the data transfer speed is easy due to its synchronous behaviour. But VME is asynchronous and measurements are slightly more difficult. As the bridge has been built into an FPGA and designs into this kind of devices should be done (although obviously it is not mandatory) synchronous there exist the needing of synchronise signals to move data from/to VME bus. In the previous section has been mentioned that VME interface has a clock 2 times faster than IBUS to improve time resolution and then also performance.

In the figures 5,6,7 and 8 can be observed the performance of the VME master and slave interacting with an ideal slave or master respectively. These figures show the peak transfer rate in the fastest 32 (BLT) and 64 (MBLT) bits transfer modes (block transfer modes) as well as the mean transfer rate for different block lengths (8, 64 and 256 bytes for BLT and 32, 256 and 1024 for MBLT) that takes account of the whole cycle including addressing. It can be expected that for large block transfers the mean transfer rate will be close to peak transfer rate, as it is shown in the figures except for master BLT read transfer where all the cycles take the same time from the beginning to the end of transfer. Note that for not very large transfer block size mean rate approximates to peak rate. When viewing the figures, it should be taken into account that in every case it has been considered that the corresponding VME signals are present on the bus a minimum of one cycle before being captured. This is generally more setup time than required. Anyhow, plots are represented only to show that the interface is able to work at high VME transfer rates.

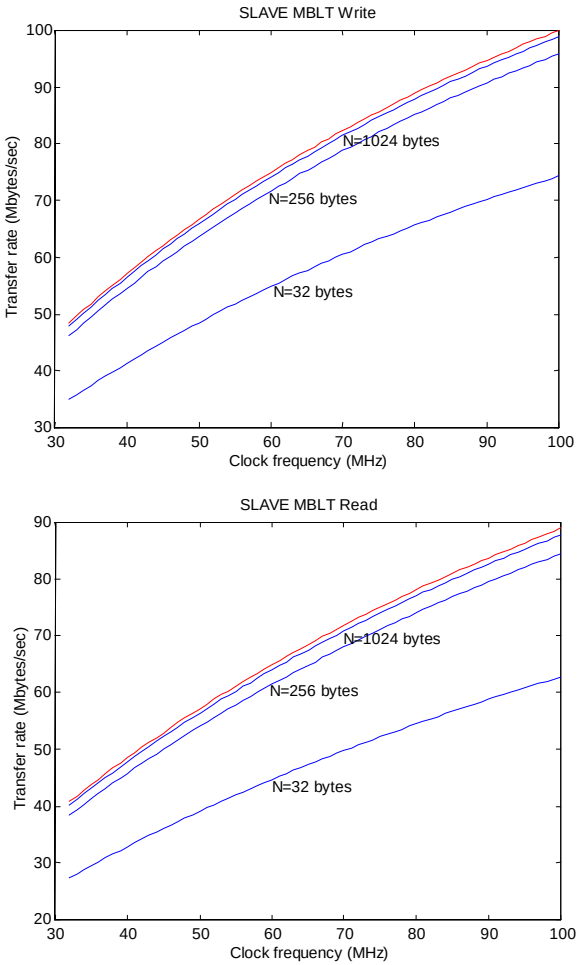

Figure 6. VME Slave MBLT performance

The slave performance could reach 90-100 Mbytes in front of an ideal master although this speed can increase if clock speed is increased. No ideal masters can be found but this demonstrates that masters able to work at high speed rates will take advantage of the bridge presented. Master performance is about 20% inferior (comparing those processes that move data in, master read and

slave write, and those that move data out, master write and slave read) but still very high. This is mainly because of mandatory timings in VME standard.

The clock frequency represented in the abscise axis of the figures 5, 6, 7 and 8 corresponds to twice the IBUS clock frequency. They cover a wide range of possible frequencies but not all the possible in the FPGA family targeted. Only for reference, the clock frequency for an XC4013E-3 (relatively slow 5V device) can reach 64MHz (32MHz IBUS clock), for an XC4013E-1 (fastest 5V speed grade) the clock frequency can go beyond 80MHz and for an XC4013XLA-09 (3V family) it is possible to implement the bridge at speeds reaching 120MHz.

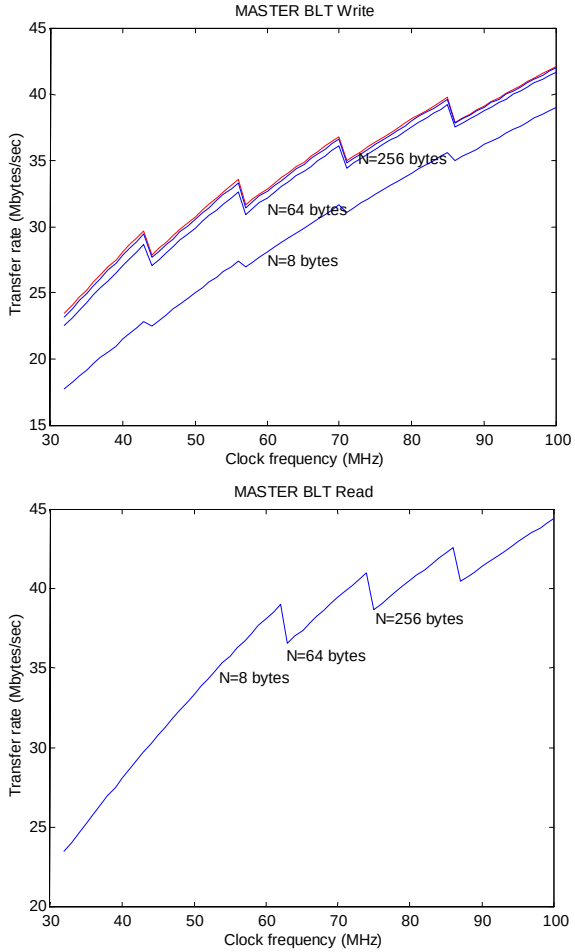

Figure 7. VME master BLT performance

It can be observed that the master curves of performance are jagged because of adjustments in those mandatory timings mentioned before. Timing adjustment is done by means of clock cycle delays. VME specifies some minimum setup and cycle-to-cycle times that the master must respect and depending on the frequency more or less cycles have to be introduced from one event to the next. This produces a drop in the performance around the frequency where one cycle has to be added. In our case the delay can be increased by half a period each time what reduces impact in the lost of transfer speed.

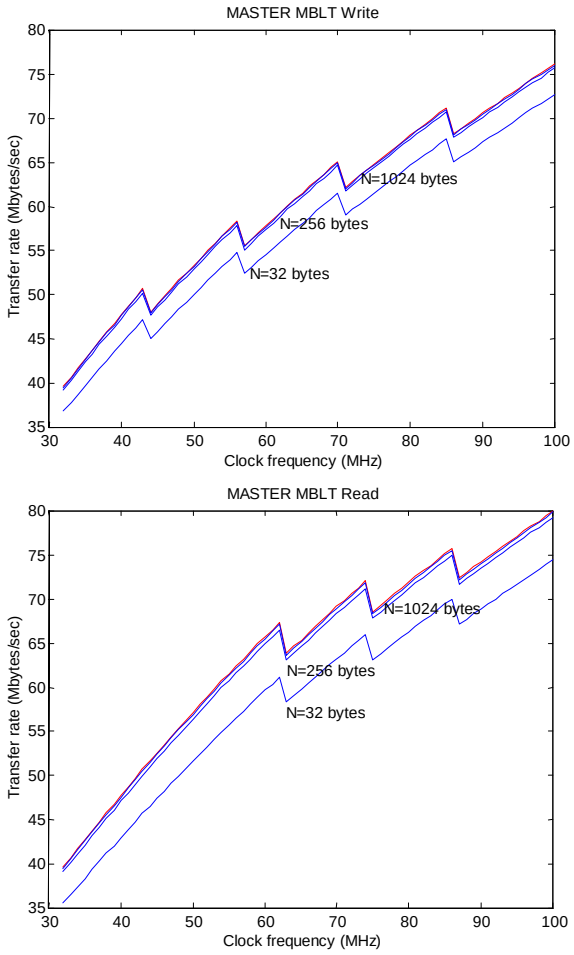

Figure 7. VME master MBLT performance

5. FUTURE WORK AND CONCLUSIONS

The bridge presented in this paper has some characteristics that can make it interesting in many applications. It fits into a relatively small FPGA and offers a large set of features and high performance supporting a recent revision of a nearly twenty years old bus interface. During this time VME bus has evolved from the initial 32 bits interface (IEEE 1014-1987) to the 64 bits interface in 1994. But today's platforms increase performance and require faster interfaces to give the desired throughput, so VME evolution has not stopped there and other mechanisms to transfer data have appeared. With this bridge the possibility of evolving towards these new approaches has been left open.

6. ACKNOWLEDGEMENT

This work has been supported by CYCIT (Spanish National Science Council) under grant TIC98-0684

7. REFERENCES

[1] VITA. VME bus Specification, ANSI/IEEE STD1014-1987.

[2] VITA. VME bus Specification, ANSI/VITA 1-1994.

[3] XILINX. XILINX XC4000E XC4000X Series Field Programmable Gate Arrays. 1999.