Memory-mapped I/O over Dual Port BRAM on FPGA

Rodrigo A. Melo, David M. Caruso, Salvador E. Tropea
Laboratorio de Desarrollo Electrónico con Software Libre
Centro de Electrónica e Informática
Instituto Nacional de Tecnología Industrial
Buenos Aires, Argentina
Email: {rmelo,david,salvador}@inti.gob.ar

***Abstract*—Nowadays, *Direct Memory Access* (DMA) is one of the most used mechanisms for data transfer between a processor and its peripherals. Another possibility is to map peripherals directly in the memory space, which has the disadvantage of requiring dual port memories when the device handles large quantities of data. It typically is the case of video and network applications. In this work we propose the use of dual port BRAM often available in modern FPGAs to implement a core using *Memory mapped I/O* (MMIO). As a case study, we present the development of an AVR microcontroller core with the *Media Access Controller* (MAC) Ethernet built in. It is capable of running the uIP TCP/IP stack, with a Web Server as example application. Additionally, we discuss the advantages of moving the program code to an external memory that use the *Common Flash Interface* (CFI) standard. This design was simulated with Free Software tools and it was verified in hardware using a Xilinx Virtex 4 FPGA.**

***Index Terms*—MMIO, DMA, BRAM, FPGA, AVR, uIP, CFI, MAC, Ethernet, Web server.**

I. Introduction

Direct Memory Access is a well known and widely used mechanism to transfer data between memory, processors and their peripherals. It allows a higher data rate and speed of operation, freeing the processors, at the expense of requiring a controller that is responsible for arbitrating the access to the bus.

Another useful mechanism is called Memory-mapped I/O (MMIO). It maps the inputs and outputs of a peripheral directly to the memory space of the processor. Dual port memories are very useful to implement it. This option was often used for video cards, but it was replaced by regular SDRAMs, mainly because of its high cost.

In this paper we propose to take advantage of the dual port BRAMs, often available as internal memory on modern FPGAs, to implement MMIO instead of implementing a DMA system. As test case of the proposed methodology, we connected two cores previously developed by our laboratory: an AVR microcontroller [1] and a MAC Ethernet controller [2]. We implemented a web server to test the architecture. For the TCP/IP stack we selected uIP [3], which is Free Software. Additionally, optimizations were performed both in hardware and firmware to reduce the BRAMs usage.

This paper is structured as follows: in section II we do a brief discussion about MMIO and the proposed architecture. Section III presents the developed core as test case while section IV contains information about the used TCP/IP stack, the adaptations made to work on the AVR microcontroller and some useful free software tools. Optimizations in the use of BRAMs are detailed in Section V. Sections VI and VII describes the tests performed and the results obtained respectively. Finally the conclusions are exposed in section VIII and section IX proposes future work.

II. Proposed methodology

A. Background

Memory-mapped I/O is a method of performing input/output between a processor and their peripheral devices. The same address bus is used to address both memory and I/O devices, thus the processor instructions used to access the memory are also used for accessing devices. This simplifies the system design and leads to simpler and faster hardware; a particular advantage in embedded systems. To access the peripherals the processor simply performs read or write operations to addresses in the given peripherals address space.

To implement MMIO each device needs a dual port memory. In a printed circuit board (PCB) it is not a viable solution: it is very expensive and it occupies a lot of board space. However, dual port BRAM are often available on modern FPGAs, so it is an easier and zero-cost alternative when there are enough resources. It should be noted that an implementation as distributed memory is not recommended due to the high consumption of FPGA area.

Another question to remark is that this methodology should be apply over devices that drive at least hundreds of data bytes. Such devices often need a memory to store data, so the only special modification is the type of memory which change to dual port BRAM.

B. System architecture

Fig. 1 shows a simplified scheme of the proposed architecture. Here, the processor is only connected to the memory space in which the system RAM and one port of each BRAM

978-1-4673-0186-2/12/$31.00

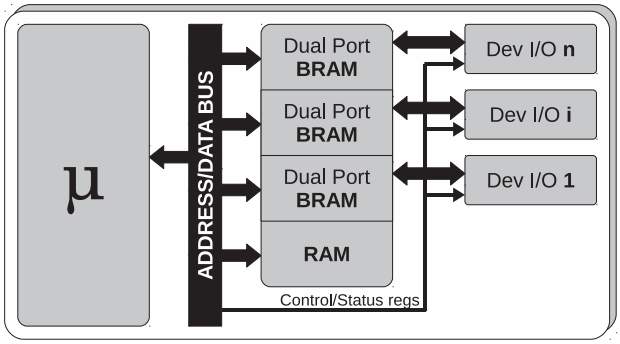

Figure 1. Proposed architecture

are mapped. The second port of each BRAM is driven by a peripheral.

In this architecture the processor can access the system memory, or a peripheral memory, just specifying its address on the bus. The control and status are implemented using registers, mapped in the same memory space, or using a separated I/O space.

III. Case study: the developed IP core

A. Implementation

An AVR microcontroller core was implemented which includes a MAC Ethernet. Both cores were interconnected using the MMIO methodology.

The core was implemented using standard VHDL 93 language, and it was developed with the tools and under the guidelines recommended by the FPGALibre project [4] [5]. A block diagram of the obtained microcontroller, which is called ATeth, is depicted in Fig. 2.

The AVR core was instantiated with a configuration equivalent to an ATmega32. All their peripherals are enabled by generics and their characteristics are similar to the CPU presented in the previous work [1].

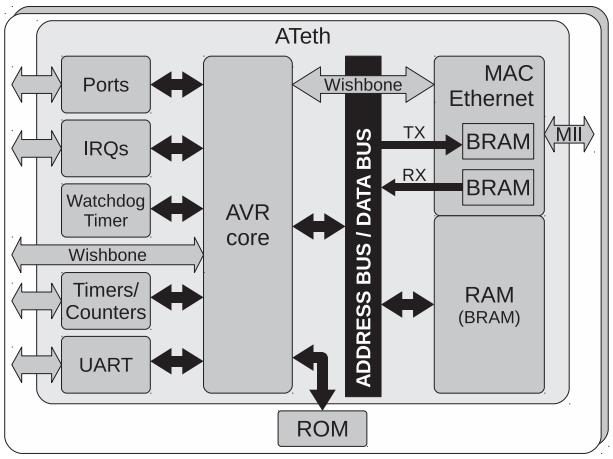

Figure 2. Block diagram of ATeth

Table I
Memory mapping

(a) Option 1

15	14	Device
0	0	RAM
1	0	RX
1	1	TX

(b) Option 2

15	14	13	Device
0	0	0	RAM
1	1	0	RX
1	1	1	TX

The MAC Ethernet core was modified from a previous work [2] where we used FIFOs and control/status registers. The reception and transmission buffers are mapped in the memory space and the registers were mapped in a WISHBONE [6] slave interface. The used WISHBONE master was implemented as internal bus instead of using the already available extension bus, to avoid the limitation of the external addresses and keep compatibility with previous developments. Each WISHBONE master were mapped in the *I/O Registers*. Both share the address register, but there are independent registers for data.

In order to decode the system memory and the mapped peripherals, we proposed the use of the most significant bits of the address bus, to make it simpler. We considered two possible decoding cases, one using the two MSBs (Table I(a)), and the other using the three MSBs (Table I(b)). The first option has a problem. In the AVR architecture the first 96 data memory bytes are used by *General Purpose Registers File* and *I/O Registers*, and next is the internal RAM. The stack pointer is initialized pointing to the last memory position, which as example in an ATmega32 is 2K+96. If the bit 15 is used to select the RAM, 32KB are available, but, it could produce an overlap between the stack pointer (32K+96) and the reception channel which start at 32K. To solve this problem, we use the second alternative.

B. MAC Ethernet usage mode

The WISHBONE registers of the MAC Ethernet core are depicted in Fig. 3. Register 0 is used for configuration and control. If bits TXerr or RXerr are set to '1' after a transfer, it means that an error has been detected and more information can be obtained from register 1. Registers 2 and 3 are used together to indicate the number of bytes received (read) or to

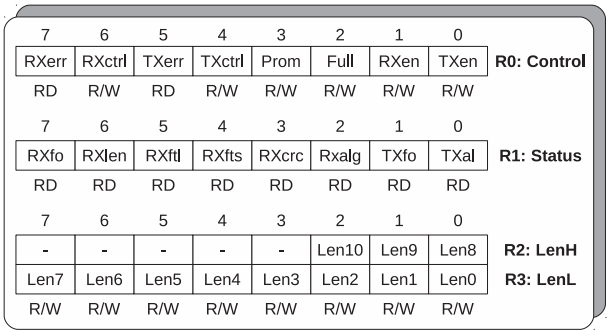

Figure 3. Registers of the MAC Ethernet core

transmit (write). Only 11 bits (2KB) are implemented because the maximum size of an Ethernet Frame is 1514 bytes.

An Ethernet frame contains 6 bytes for the destination MAC address, 6 bytes for the source MAC address, 2 bytes for the length/type field, the information and finally the CRC, which is automatically added in transmission and is discarded in reception.

- Configuration: bits TXen, RXen, Full and Prom of the Control register are respectively used to enable/disable the transmission channel, the reception channel, the full duplex mode and the promiscuous mode.
- Transmission: in order to determine if the channel is ready for a new transmission the user should ensure that the TXctrl bit is 0. The data to be transmitted have to be copied to the transmission memory and then the number of bytes have to be specified on registers 2 and 3. Transmission starts when the bit TXctrl is set. If an error has occurred, bit TXerr is set and details could be obtained from bits 0 and 1 of the Status register.
- Reception: there are available data when bit RXctrl is 0. The amount of bytes received can be obtained from registers 2 and 3. If there are errors the bit RXerr is set and details could be obtained from bits 2 to 7 of the Status register. Finally, is needed to indicate that the data have been processed setting the bit RXctrl.

IV. Firmware

A. TCP/IP stack

We conducted a search on available TCP/IP stacks which have low hardware requirements and a free license. We selected the uIP TCP/IP stack, version 1.0, which nowadays is part of the Contiki Operating System [7]. The uIP stack is written in C language, it provides TCP/IP connectivity to 8 bit microcontrollers, and is RFC compliant. As test application we selected a simple web server, used to present four different web pages.

The following features made uIP suitable for our project:

- It has very low hardware requirements.
- It is vastly used by embedded systems.
- There is a version that was ported to AVR, so, we knew that probably it would run with ATeth.
- Is free software and so, we can modify, use and distribute without the need of pay royalties.
- Source code structured and modular.
- Well documented.
- Available examples, including the selected web server.

B. Adaptations

To use uIP with ATeth, we wrote the driver for the MAC core. Basically, it implements functions to initialize, send and receive data, following the steps of section III-B.

The example used as base for the web server was designed as a simulation test using virtual interfaces, under GNU[8]/Linux. It was necessary to modify the example *main* function to use a real Ethernet interface.

Furthermore, the ATmega32's 2KB of data memory was not enough for the embedded web pages. Our FPGA implementation, supports the configuration of the memory size. The only special care is to move the start address of the stack pointer. In our case, it was modified to 0x405F (16K+96 *bytes*).

C. Software tools

The implemented core includes the entire instructions set of the original processor and an equivalent configuration to one existent microcontroller, so, it is possible to use available software tools, such as the avr-gcc, a version of the GNU C compiler [9].

Fig. 4 shows the tool-chain used to obtain the hardware description of the memory program from the source code of the web server. Starting from C sources (and their headers .h) an executable (elf) is obtained using avr-gcc. Next, the program memory (bin) is extracted with avr-objcopy. The tools bin2hex and vhdlspp were developed by our laboratory as part of the FPGALibre project: bin2hex converts a bin file into hexadecimal values for VHDL assignments (dat) and then vhdlspp (VHDL Simple Pre Processor) combines it with a ROM skeleton and the resulting file is ready to use in ATeth.

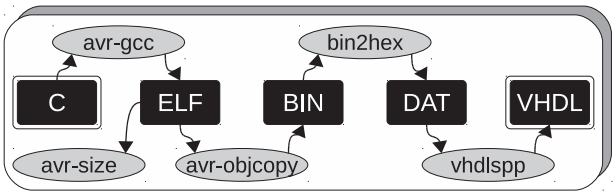

Figure 4. Getting the program memory

We used the avr-size tool to determine the amount of memory needed for the resulting application. In this way we can determine if the memories are properly dimensioned. The tool shows the amount of bytes used for sections *text*, *data* and *bss*. The sum of *text* (executable code) and *data* (values of initialized variables) sections is equal to the number of bytes used by the program memory, while the sum of *data* and *bss* (uninitialized variables) is the quantity of bytes of the data memory.

V. Reducing the usage of BRAMs

BRAMs are a resource frequent but finite in FPGAs, so, it should be used with moderation. To optimize its use, different alternatives were analyzed, changing both the hardware and the firmware.

A. Moving the ROM to a parallel flash memory

First of all, it was observed that most of BRAM were used as ROM memory by the AVR, because the TCP/IP stack code was about 20KB. We chose to move the ROM of the microcontroller to an external parallel flash memory that previously existed in the kit. It has a CFI [10] interface, an open standard developed by AMD, Fujitsu, Intel and Sharp, which can provide information through a predefined

command sequence. This information includes details such as the memory size, erasing times, size of the record buffer, voltages used, etc. Moreover, most memories use standardized mechanisms to save and delete data, defined as *AMD/Fujitsu Command Set* and *Intel/Sharp Command Set*. The second one is the used by our CFI memory.

To work with this kind of memory, our laboratory developed a CFI controller core. In short, it is a hardware Finite State Machine (FSM) that allows to read and write to the flash memory, with commands sended by a PC through USB. Steps are: first, the FPGA have to be configured with the CFI controller. After that, the flash memory is recorded via USB. Finally, the FPGA is configured again, but this time with ATeth.

The used flash memory has a page read time longer than the data access time. AThet can not insert wait states by itself, then has to read the memory at a compatible speed with page changes. This is a lower working frequency that the previous case. To improve the read performance, the *Flash ROM Interface* core [11] was used. This core insert wait states to microcontroller when a page change is encountered, otherwise allows the data is extracted at maximum speed. Fig. 5 shows a connection scheme between the microcontroller, the *Flash ROM Interface* core and the flash memory.

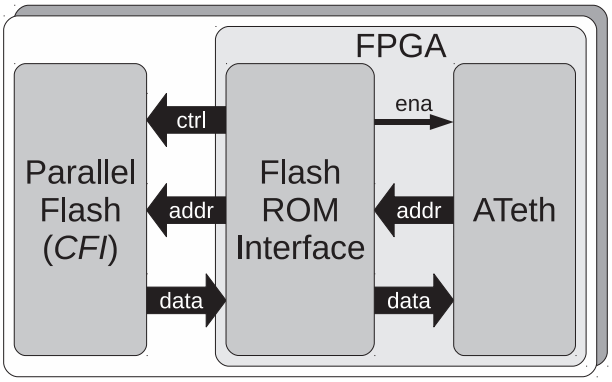

Figure 5. Interconnection diagram of ATeth, *Flash ROM Interface* and CFI

B. Firmware optimization

The uIP firmware was developed to be device independent. Because of this, it has not specific hardware considerations.

The AVR is a modified Harvard architecture where program and data are stored in separated physical memory, which use a different address space. By default, the instructions takes data from the RAM.

String constants are stored in ROM, and copied to the RAM before passing the control to the main() function. This data redundancy requires more RAM than what is actually needed.

The gcc compiler provides a special mechanism to avoid this duplication. This is achieved using attributes, a gcc extension to ANSI C. As a side effect, the code must copy the constant to RAM before calling the consumer function. A scratch buffer is used to maintain only one of the needed string constants in RAM.

We applied these attributes mainly over the constants that represent the web pages, which are a big amount of data, and we made some modifications to the uIP code, based on an AVR adaptation [12].

VI. Tests and Validations

A. Testbench

For the simulation of the core GHDL [13] 0.29 was used. We made a very simple testbench to test the interconnection between the AVR and the MAC Ethernet:

- The program responds packets of type ARP (*Address Resolution Protocol*) and ICMP (*Internet Control Message Protocol*) through MAC Ethernet interface. We called this program loopback_eth.
- To generate stimulus, we used scripts originally developed for the MAC Ethernet project. Two files were generated using random bytes: the first with reception data and the second with the correspondent transmission data.
- The testbench simply injects data from the reception file and compare the output with data from the transmission file, aborting when a difference is detected.

B. Hardware validation

This development was validated using a Xilinx Virtex 4 FPGA (XC4VLX25) and the ISE WebPack 11.3 - L.57 tool. The external CFI memory is an Intel Strata Flash TE28F640. We use a personal computer with Debian [14] GNU/Linux Operating System.

The first test was performed using the loopback_eth program on the AVR. We tested the system by sending and receiving *ping* packets, and then, we analyzed the interchanged packets between the PC and the core with the *wireshark* program [15].

The following tests include the uIP stack program. We made a script that download the served web pages and after that, the downloaded pages and the developed local versions are compared. Moreover, we browsed the four web pages provided by the server and finally, we tested again by sending and receiving *ping* packets.

VII. Results

A. Synthesis

Table II(a) shows the results of ATeth synthesis for each modification and improvement which were explained in section V. Additionally, the memory usage is shown in the same Table. In case 1 the ROM is implemented on BRAMs, while in case 2 and the followings is in an external flash memory (CFI). In cases 1, 3 and 4, the working frequency was 50 MHz, which is one of the available clock sources on the used Virtex-4 Evaluation board. In case 2 the system run at 8 MHz using a Digital Clock Manager (DCM), because the flash memory has a 120 ns delay when a page change is performed.

Table II
RESULTS OF THE SYNTHESIS

(a) The developed IP core

Case	FFs	LUTs	Slices	Fmax (MHz)	BRAMs	PING (ms)	Text	Data	bss	ROM	RAM
1	739	2558	1486	62	27	0.25	13640	6774	7045	20414	13819
2	736	2411	1417	56	11	0.74					
3	754	2567	1499	61	11	0.32					
4	727	2612	1488	61	5	0.32	23946	30	1224	23976	1254

(b) Compared to other cores

Case	Processor	Type	Slices	GCLKs	BRAMs
xapp433	MicroBlaze	Softcore	3737	5	8
xapp434	PowerPC	Hardcore	4383	5	12
ATeth	AVR	Softcore	1488	3	5

Comparing cases 1 and 2, the BRAM consumption has dropped 60% and the slices has slightly decreased by 6% when the ROM is moved to the external CFI. On the other hand, the time of ping response is triplicated. So, in case 2 the hardware consumption is less but the time response is worst.

In case 3 the *Flash ROM Interface* is used to improve the CFI access time. In comparison to case 1, the results are a slight increase of the ping time response, where the slices consumption is similar but the BRAM consumption remain stable respect to the case 2.

Finally, the case 4 corresponds to the firmware optimizations, which has the aim of reduce the RAM redundancies. 5.5K bytes of web pages were moved to program memory. In such case the RAM is reduced by a 90% and consequently the BRAMs are reduced by 55% respect to the case 3 and 80% respect to the case 1. The use of ROM has slightly increased by 15% due to the use of special functions to read constants from this memory.

B. Other implementations

Two application notes [16] [17] from Xilinx were found, which may be used to contrast results. The designed systems in these application notes have the following features:

- Use a Xilinx Virtex 4 FPGA (XC4VFX12).
- A processor (softcore or hardcore) is connected to 16 Kb of BRAM.
- An external SRAM memory is used to host 3.8K bytes of web pages.
- An external DDR SDRAM memory is used for the executable code.
- An UARTLite, an Ethernet 10/100 MAC and a General Purpose I/O are connected.
- An On-chip Peripheral Bus (OPB) is used for interconnection.
- The lwIP [18] stack is used.

The extracted results of the notes and also the ATeth results are depicted in Table II(b). It shows that our implementation takes at least 2.5 times less slices, occupies less BRAMs and uses 3 global clocks instead of 5.

VIII. CONCLUSIONS

The MMIO methodology proved to be very simple to implement. When the MAC Ethernet core was adapted and its registers were mapped in a WISHBONE slave interface, the connection with the AVR core and the control from the firmware was trivial. The arbitrator and the cache memory, that are needed in a similar DMA implementation, have not been used here, which means it uses less area of the FPGA. In short, MMIO is a recommended methodology for the fast and agile development over FPGA that have dual port BRAM, which nowadays are the most common.

Even though it is not possible an exact comparison with the implementations of the application notes, we can conclude that our core use less than half of the area. Moreover, the use of BRAM in our best case is lower.

Moving the program memory to an external flash significantly reduced the number of BRAMs. As a side effect, it reduced the operation frequency, slowing down the ping response time. Using the *Flash ROM Interface* core we achieved a response time similar to the BRAM only version. Moving most constants to the ROM, the data section is reduced as well as the quantity of BRAM.

The uIP TCP/IP stack and the Web Server example were adapted and used in a very simple way. This denotes the versatility of uIP to be used in embedded systems.

The use of the standard VHDL 93 language allows the core to be synthesized using FPGAs from most manufacturers, which is a desired characteristics. Tools and methodologies proposed by the FPGALibre project proved to be suitable for this work.

IX. FUTURE WORK

As future work, we will to implement a system with multiple processors and devices, using both DMA and MMIO, in order to obtain an accurate comparison of resources involved, ease of implementation, performance, maximum operating frequency, etc.

REFERENCES

[1] S. E. Tropea and D. M. Caruso, “Microcontrolador compatible con AVR, interfaz de depuración y bus wishbone,” in *Proceedings of the FPGA Designer Forum 2010*, Ipojuca, Brazil, 2010, pp. 1–6.

[2] R. A. Melo and S. E. Tropea, “IP core MAC Ethernet,” in *Proceedings of the FPGA Designer Forum 2011*, Córdoba, Argentina, 2011, pp. 1–4.

[3] A. Dunkels. (2011, Oct.) uIP TCP/IP stack. Networked Embedded Systems group / Swedish Institute of Computer Science. [Online]. Available: "http://www.sics.se/~ adam/old-uip/"

[4] S. E. Tropea, D. J. Brengi, and J. P. D. Borgna, “FPGAlibre: Herramientas de software libre para diseño con FPGAs,” in *FPGA Based Systems*. Mar del Plata: Surlabs Project, II SPL, 2006, pp. 173–180.
[5] INTI Electrónica e Informática *et al.*, “Proyecto FPGA Libre,” http://fpgalibre.sourceforge.net/.
[6] Silicore and OpenCores.Org. (2011, Oct.) WISHBONE System-on-Chip (SoC) interconnection architecture for portable IP cores. [Online]. Available: http://prdownloads.sf.net/fpgalibre/wbspec_b3-2.pdf?download
[7] (2011, Oct.) The operating system for the internet of things. Cisco, Redwire LLC, SAP, SICS, and others. [Online]. Available: http://www.contiki-os.org/
[8] “GNU project,” http://www.gnu.org/, Jun. 2010.
[9] (2009, Nov.) GCC, the GNU compiler collection. [Online]. Available: http://gcc.gnu.org/
[10] JEDEC Solid State Technology Association. Common Flash Interface (CFI). [Online]. Available: http://www.jedec.org/download/search-/jesd68-01.pdf
[11] D. M. Caruso and S. E. Tropea, “Comparación del desempeño de microcontroladores AVR de 4ta generación,” in *Congreso Argentino de Sistemas Embebidos - Libro de Trabajos*, Buenos Aires, Argentina, 2011, p. 179.
[12] T. Harbaum. (2012, Feb.) Wlan for avr. [Online]. Available: "http://www.harbaum.org/till/spi2cf/index.shtml"
[13] T. Gingold. (2010, Jun.) A complete VHDL simulator. [Online]. Available: http://ghdl.free.fr/
[14] I. Murdock *et al.* (2010, Jun.) Debian GNU/Linux operating system. [Online]. Available: http://www.debian.org/
[15] G. Combs and contributors. (2010, Jun.) Network protocol analyzer. [Online]. Available: http://www.wireshark.org/
[16] (2011, Oct.) Embedded system example: Web server design using microblaze soft processor. Xilinx. [Online]. Available: http://-www.xilinx.com/support/documentation/application_notes/xapp433.pdf
[17] (2011, Oct.) Web server reference design using a powerpc-based embedded system. Xilinx. [Online]. Available: http://www.xilinx.com/-support/documentation/application_notes/xapp434.pdf
[18] L. W. Adam Dunkels *et al.* (2011, Oct.) The lwIP TCP/IP Stack. Swedish Institute of Computer Science. [Online]. Available: "http://www.sics.se/ adam/lwip/"