Implementing the Cortex-M0 DesignStart Processor in a Low-end FPGA

Pedro Ignacio Martos & Fabricio Baglivo
Laboratorio de Sistemas Embebidos
Facultad de Ingeniería – Universidad de Buenos Aires
Buenos Aires, Argentina
pmartos@fi.uba.ar / baglivofabricio@gmail.com

Abstract— **The Cortex™-M processor family from ARM is optimized for cost and power sensitive MCU and mixed signal devices, available as soft cores and hard cells in FPGAs. There are Cortex-M solutions in FPGAs from Altera (the Cortex-M1 is delivered as a soft core) and Actel (the Cortex-M1 is delivered as a soft core and the Cortex-M3 as a hardened block); however, there are currently no Cortex-M solutions from Xilinx. Moreover, ARM has recently launched a low cost, reduced version of the Cortex-M0 processor (Cortex-M0 DesignStart™), which could be synthesized into a FPGA or used for silicon implementations. In this article we show the results of the implementation of the Cortex-M0 DesignStart processor in a low-end FPGA from Xilinx, extending the implementations available for Cortex-M processors in FPGA.**

Keywords- Cortex-M0, FPGA

I. INTRODUCTION

A. ARM Processor Families

In the ARM processor line, the Cortex family, shown in Figure 1, consist of cores ranging from low cost microcontroller solutions to high end processors capable of supporting large, complex operating systems. Older processors include the ARM7, ARM9 and ARM11 families, and the specialized series SecurCore™ for security and cryptographic applications.

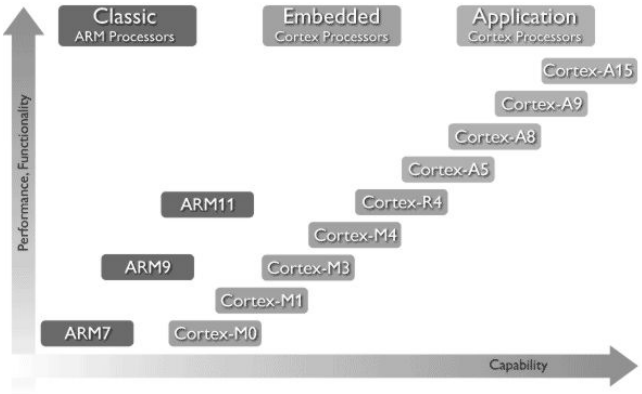

Figure 1. Performance vs capabilities in the ARM processor families

The Cortex-M0 processor is the lowest member of the Cortex-M family. This family allows different tradeoffs between cost, design simplicity, power, performance and computation power in an embedded processor.

The Cortex-M0 processor aims to get low power and a small area to be competitive against high-end 8-bit processors and 16-bit processors, but it maintains code compatibility with the other members of the family (shown in Figure 2), like the Cortex-M3 processor. The smallest Cortex-M0 processor has a power rating of 84µW/MHz and has a gate count of approximately 12,000. As a reference, a classic processor like the i8051 has a gate count around 10,000 gates.

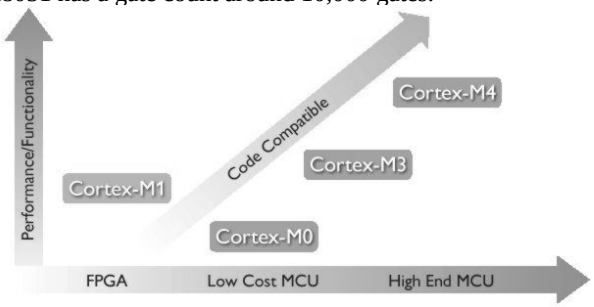

Figure 2. Comparision of different Cortex-M processors

B. Cortex-M0 Architecture

The Cortex-M0 processor, shown in Figure 3 is based on the ARMv6-M architecture (Von Neumann) with a 3-stage pipeline. It archieves a Dhrystone mark of 0.9DMIPS/MHz, It can be implemented with up to 32 maskable interrupts, plus one non-mascable interrupt using a Nested Vectored Interrupt Controller (NVIC) with a fixed latency of 16 machine cycles **[1]**.

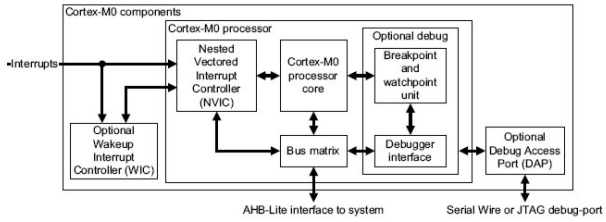

Figure 3. Cortex-M0 block diagram

The Cortex-M0 processor implements a reduced version of the Advanced Microcontroller Bus Architecture (AMBA®), the AMBA-Lite bus (shown in Figures 4 to 6), to connect to different peripherals. This bus was designed as a high performance bus for fast transactions between processors and peripherals needing a wide bandwidth or high throughput. The Cortex-M0 is, in general, the only master device, and all the peripherals are slaves over the bus; however, the bus specification allows for a multimaster system. The most important features of the bus are: single or burst transfers; single cycle transfers; tri-state buffers; and variable bus width configurations (32, 64, 128, 256,512 or 1024 bits) **[2]**.

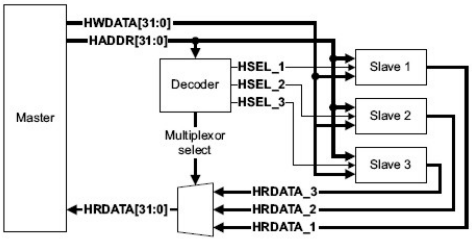

Figure 4. AMBA-Lite system

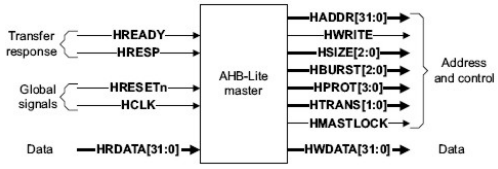

Figure 5. AMBA-Lite master signals

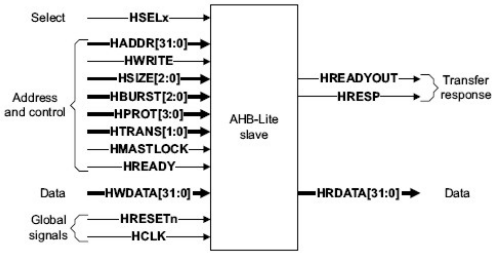

Figure 6. AMBA-Lite slave signals

C. *Cortex-M0 DesignStart Processor*

The Cortex-M0 DesignStart processor (M0DS) is a fixed configuration of the Cortex-M0 Processor (M0S). The M0DS processor was launched on August 4th, 2010. The main differences between the two processors are: the M0S processor has master and slave signals in its AMBA-Lite bus, where the M0DS processor has only master signals. The M0S can be configured with a fast single-cycle multiplier or with a slower 32 cycles multiplier, the M0DS only has the slower one. The M0S processor can be configured with up to 32 interrupt sources, the M0DS has 16 fixed interrupt sources. The M0S can be configured with a wake-up interrupt controller, architectural clock gating, a debug port (up to 4 hardware breakpoints, a serial or JTAG interface) and a 24 bit system timer. The M0DS processor does not have these features **[3][4][5]**.

II. IMPLEMENTATION

A. *Hardware and Software*

We use a mature and low-end FPGA for our implementation: the Xilinx S3E500-4. Its features are: 500K gates, 10500 logic cells (1100 configurable logic blocks–CLB or 4600 slices), 20 hardware multipliers, 360Kbits of block RAM, 73Kbits of distributed RAM, 4 digital clock managers (DCMs) and a maximum operating frequency of 300 MHz **[6]**.

We used a starter board from Digilent, the Nexys2, shown in Figure 7. It has an S3E500 FPGA, a USB-based programming and communications interface, 16Mbytes of PSDRAM, 16Mbytes of flash ROM plus a configuration PROM, a 50 MHz oscillator, 8 LEDs, 4 seven segment displays, 4 pulse buttons and 8 switchs. There are 60 I/O pins available from the FPGA. **[7].**

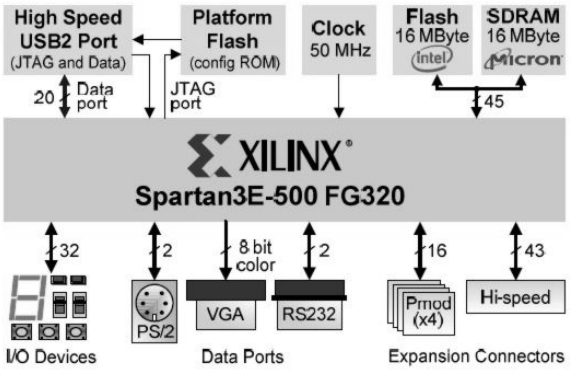

Figure 7. Nexys2 board block diagram

With this board and a plugin (Adept) it is posible to use Xilinx tools to work with the boad (Impact, ChipScope Pro, xmd, etc.), so it is possible to program and to see its internal state using standard tools and scripts.

The FPGA toolchain was ISE 12.2. The software toolchain was the ARM Microcontroller Development Kit from Keil™ .

B. *Cortex-M0 DesignStart Kit*

This kit has two parts: the processor, implemented in two synthesizable Verilog files, and a testbench with a basic program. The documentation is in .pdf format, which includes the release notes. **[3]**.

The testbench, shown in Figure 8, has a non- synthesizable Verilog module that instantiates the Cortex-M0 processor connected to a memory loaded with the basic program. Other items in the testbench include the C source code of the basic program, the binary image of the program, and a makefile to build the image from the C source code.

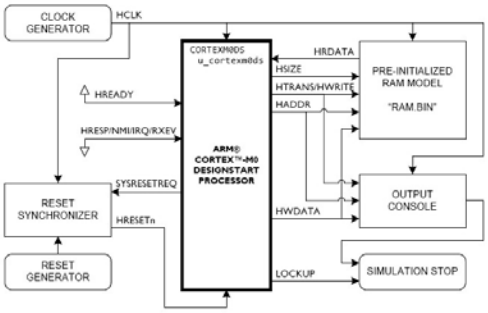

Figure 8. Testbench block diagram

C. *Implementation Working Plan*

We planned to do the following activities: a) verify that the processor could be synthesized in the chosen FPGA; b) create a project in ISE with the testbench and check it by functional simulation with ISIM; c) generate the image file from the C source code using the makefile; d) verify the binary file generated using the testbench project; e) generate a synthesizable testbench project in ISE. f) generate a program that can interact with the hardware on the board and verify it with ARM MDK debugger, g) integrate the image generated in the synthesizable testbench and verify it with ISIM. h) synthesize the project and load it in the board to check the correct implementation, and i) verify the system with ChipScope Pro.

D. *Implementation Results*

Item (a) was done successfully. It showed that the Cortex-M0 implementation used aproximately. 50% of the fabric of the FPGA. During the implementation of item (b), when we simulated the testbench, we found that the processor entered in a blocked state before entering the main routine. We decided to move on to item (c) to have a software simulation of the program.

When we studied the makefile in detail for use with the ARM MDK (as stated in the release notes **[3]**), we realized that it was not consistent. The tools for compiling and linking were from ARM MDK (armcc and armlink), but the calling parameters weren't for those tools. Looking for information in other toolchain providers, we found that the parameters were made for the IAR Embedded Workbench toolchain (iccarm and ilinkarm). So the makefile was made from parts of different makefiles for different toolchains, rendering it useless for generating the image file from the C source code.

With these findings, we decided to generate a new project in ARM MDK using only the C source code, get an image file, and use this new image file to see if the processor locks in the funcional simulation on ISIM. We found that the processor was still blocked before the main routine, but having a software simulation, we could compare the software simulation on ARM MDK against the functional simulation on ISIM. Checking that, we saw that the software simulation worked as expected, but the ISIM simulation showed strange values over the data bus on data fetches from memory. Further inspection of the testbench showed that the problem was in the memory initialization: the binary image was read using Verilog's $fread function, expecting to read four bytes in each read, filling one 32-bit word of the memory. Xilinx's implementation of this function makes one-byte reads in each call to the function. So we modified the testbench to make four reads before filling the 32-bit word of the memory. Doing so solved the problem and the software simulation was the same as the functional simulation.

So, we continued with items (e), the synthesizable system, and (f), a program using hardware resources. We started with the program. Since we did not want to add complexity to the project by generating an AMBA-Lite slave device connected to the processor, we decided to make a program that loads two different constant values fetched from memory into an internal register on the processor. Those values should appear on the data bus, where it could be possible to see and catch them. The program was tested in the ARM MDK simulator, shown in Figure 9, to see the memory fetches to get the values. Doing that, the program was loaded in the testbench and with an ISIM functional simulation we could see the values on the data bus.

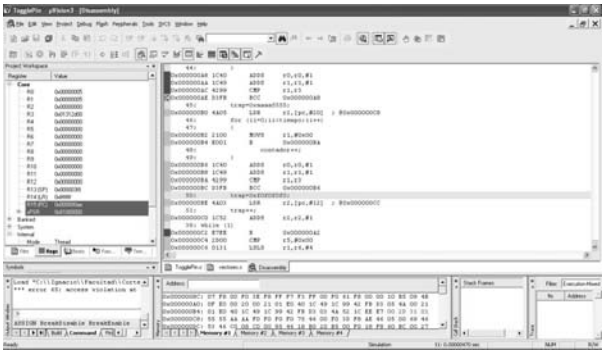

Figure 9. ARM MDK simulator showing the memory fetch

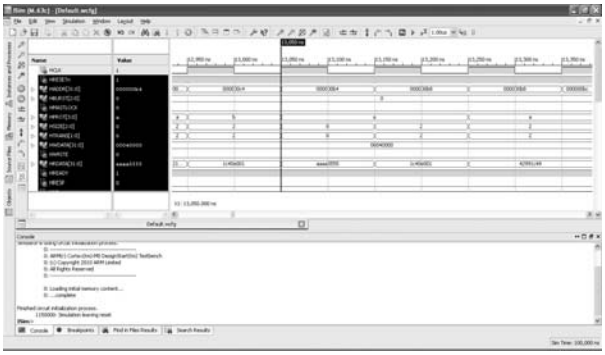

Figure 10. ISIM simulator showing the memory fetch

With a successfull simulation, we synthesized the system. The system has these parts: a) the processor, with the Cortex-M0

DesignStart Verilog code; b) a reset synchronizer, using a counter, where we generated a reset signal synchronized with the system clock; c) memory implemented with block RAM preloaded with the image file; d) a clock, which is the 10 MHz system clock generated with a DCM from the 50 MHz oscillator; and e) a bus signal detector, which detects the constant values fetched from memory on the data read bus. It commands a LED on the board to switch on when one value is present on the bus, and switch off when the other constant value appears on the data read bus. It was necessary to make a program to convert the binary data file from ARM MDK (.bin) to the memory initialization values in CoreGen (.COE). The system was simulated with ISIM, in particular the bus signal detector, to verify that the constant values were detected when they were on the data read bus.

Finally , the system was synthesized and loaded on the board. We then verified that we saw the constant values on the data read bus during the memory fetches with ChipScope Pro; and visually, we saw the LED toggling at programmed intervals, so we considered the implementation validated.

III. RESULTS

The most important result is that it is possible to implement this processor in a low-end FPGA, so it can be used in low-cost/low-resources embedded systems in an FPGA. At the same time, the FPGA technologies where Cortex-M processors can be used was expanded. Now it is possible to implement Cortex-M processors in the three main providers of FPGA technology (Xilinx, Altera and Actel), so Cortex-M processors can be a good choice when FPGA portability is needed. Figure 11 shows our implementation statistics.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	846	9,312	9%	
Number of 4 input LUTs	4,601	9,312	49%	
Number of occupied Slices	2,903	4,656	62%	
Number of Slices containing only related logic	2,903	2,903	100%	
Number of Slices containing unrelated logic	0	2,903	0%	
Total Number of 4 input LUTs	4,721	9,312	50%	
Number used as logic	4,601			
Number used as a route-thru	120			
Number of bonded IOBs	9	232	3%	
Number of RAMB16s	4	20	20%	
Number of BUFGMUXs	2	24	8%	
Number of DCMs	1	4	25%	
Average Fanout of Non-Clock Nets	3.69			

Figure 11. FPGA usage with the implementation

As a note, the high block Ram usage is not because of the system. Memory is used by ChipScope Pro hardware for sample storage. So the block Ram usage in the system is mainly dependent of the size of the code. For temporal results, the synthesizer showed a maximum working frequency of around 40MHz, but this value could be higher because we did not put any time or area constraints in the design.

Another feature of this processor is that it is possible to view the internal registers, so it could be used in educational environments to show the similarities between the software simulation, the hardware simulation and the real implementation at the internal registers level.
Another tool that would be usefull is a full testbench that generates a binary image from the C source code. That was not possible with the elements in the delivered testbench, so it was necessary to do other activities that were not planned.

As a final conclusion, this processor adds to the processors that could be implemented in Xilinx FPGAs, with the added value that can be used in other FPGA architectures, too. For a future work, we will create an implementation of the AMBA-Lite bus and a set of peripherals (UART, I2C, SPI, etc) that could be connected to this bus to expand the processor's capacity. This model could become a good choice to develop embedded systems using Xilinx FPGAs, with the capability to execute embedded Linux.

IV. REFERENCES

[1] ARM Ltd, "ARM DDI 0419C ARMv6-M Architecture Reference Manual", September 2010.

[2] ARM Ltd, "ARM IHI 0033A AMBA 3 AHB-Lite Protocol V.1.0 Specification", June 2006.

[3] ARM Ltd. "AT510-DC-80001-r0p0-00-rel0 ARM Cortex-M0 DesignStart Release Note", August 2010.

[4] ARM Ltd. "ARM DDI 0432C Cortex-M0 Revision r0p0 Technical Reference Manual", November 2009.

[5] ARM Ltd, "ARM DUI 0497A Cortex-M0 Devices Generic User Guide", October 2009.

[6] Xilinx, "DS312 Spartan-3E FPGA Family: Datasheet", August 2009.

[7] Digilent, "Digilent Nexys2 Board Reference Manual", June 2008

V. ACKNOWLEDGEMENTS

To the ARM University Program, including William Hohl and Joe Bungo, as well as Fiona Cole from Digilent, and the people at the Xilinx University Program (XUP) for their support and cooperation.

VI. TRADEMARKS AND COPYRIGHTS