MB-LITE: A robust, light-weight soft-core implementation of the MicroBlaze architecture

Abstract—**Due to the ever increasing number of microprocessors which can be integrated in very large systems on chip the need for robust, highly configurable processors has emerged. Within this paper a light-weight cycle compatible implementation of the MicroBlaze architecture called MB-LITE is presented, in an attempt to fill the quality gap between commercially licensed and opensource processors. Experimental results show that MB-LITE is able to obtain very high clock frequencies compared with other opensource processor designs, while using very few hardware resources. The processor can be easily extended with existing IP components thanks to a modular, highly configurable multiplexed memory bus and wishbone bus adapter. All components are developed using a two-process design methodology to improve performance and improve simulation and synthetization speed. Furthermore attention is paid to reduce code size and optimize readability. The processor and all memory bus components have been thoroughly verified using behavioral and RTL netlist simulations. Continued work on the MB-LITE will focus on including the design in a reconfigurable fabric as well as fabrication in a 90 nm process technology.**

I. Introduction

The increase in performance and capacity of Field Programmable Gate Arrays (FPGAs) has made it possible to include an increasing number of embedded processors within a System On Chip (SOC). Several different processors have been successfully used to simulate novel concepts in the field of Parallel Computing [1], [2] or On-Chip Networks [3], [4].

Many commercially licensed embedded processors exist (e.g. MicroBlaze from Xilinx or NIOS2 from Altera) and can be used in research projects. However, using these commercial processors in complex designs has several drawbacks. MicroBlaze and NIOS are exclusively available as firm-core implementations which makes it impossible to modify the core or implement it on an other platform than FPGA. All of the commercial designs aim to be the fastest while offering as much as possible features, like bus protocols. However, these can not be replaced by a custom bus.

Several opensource processors are available for free which can be used instead of the commercial ones (e.g. LEON3 from Aeroflex Gaisler, aeMB from Aeste Engineering and OpenRISC developed on OpenCores to name a few). All of these designs have moderate performance while using a lot of resources. More importantly these designs are of disputable quality especially when taking metrics like usability, modularity, configurability and portability into account [5]. None of the evaluated designs are properly documented nor follow a design methodology, and all of these processors are written for specifically for certain FPGAs (mostly Xilinx).

In this paper the design and implementation of the MB-LITE embedded microprocessor is presented to fill the gap between commercially licensed processor designs and the drawbacks mentioned of the available opensource processors. MB-LITE is a light-weight highly configurable implementation of the MicroBlaze architecture [6]. It is a cycle accurate model so all instructions have the same latency as the MicroBlaze architecture specification. The processor is also architecturally compatible with MicroBlaze, so the toolchain and libraries from Xilinx can be used without modifications. The processor can be configured to implement a multiplier, barrel shifter or an interrupt. A highly configurable address decoder as well as a modular wishbone bus adapter is provided to simplify the integration in various research projects. It is developed in accordance with a well-defined structured VHDL framework and a well known design methodology.

The remainder of this paper is organized as follows. In Section II some related work and several existing processor models are discussed. Section III will focus on the design of the MB-LITE while in Section IV the performance with respect to several other platforms is discussed. Finally some conclusions will be presented in Section V.

II. Background

A. Related work

In [7] a clear overview is given of several synthesizable CPU cores (i.e. LEON3, MicroBlaze and OpenRISC1200). Many design aspects are taken into account in their research like speed, resource utilization, synthesization and documentation. This work does not answer the question to what extent these designs can be reduced. For research projects it is often important to have a reliable and small implementation instead of a very elaborate and large design.

In [8] a clone of the Altera NIOS processor is presented (UT NIOS) and compared with the original design. This work has been carried out to gain experience with design processes and methodologies and uses the design of this processor as a benchmark rather than a final result.

In [9] a microprocessor is proposed for research on configurable arrays. The authors noticed that a reliable opensource microprocessor implementation did not exist and that the available designs would often require many logic elements or did not offer enough performance. A MicroBlaze compatible core was developed named OpenFire. Unfortunately almost no documentation is available and it does not have interrupts nor a wishbone bus. Furthermore it is targeted specifically to Xilinx FPGAs.

B. Structured VHDL design

Traditional VHDL design approaches contain many small processes and are as a result extremely hard to read and maintain. The absence of a unified signal naming convention makes it even harder to understand the algorithm. Due to many processes which need to be executed concurrently, simulation speeds will degrade. Additionally, no benefit is taken from the increased level of abstraction offered by behavioral VHDL models.

A robust design methodology to overcome the aforementioned problems is presented in [10]. Within this design methodology a clear separation between combinational and sequential elements is employed. The algorithm is completely determined by the combinational logic, while intermediate results are stored in the sequential part. Due to this separation the synthesizer is able to take the most advantage of its optimizing capabilities and the behavior of the algorithm will be much clearer. As a result the models become easier to maintain and less error-prone. Since only two processes need to be executed concurrently, simulation speeds will be much higher as compared with the traditional design method.

In addition to this clear separation between combinational and sequential logic, an even higher abstraction level of abstraction can be obtained by using records and types. Records can be efficiently used to group related signals, while types can be defined to indicate the purpose of a signal. All modern tools have very good support for these constructs and do not introduce discrepancies between simulation and synthetization. Using these high level constructs drastically reduces code size so development can take place faster and code is even easier to maintain.

C. The MicroBlaze architecture

The MicroBlaze architecture is almost identical to the famous MIPS architecture. It is a Harvard architecture with 32-bit instruction and data words. Both architectures are designed to be implemented with five pipeline stages. Most instructions - except for branches - have single-cycle latency. The architecture features a single interrupt which can be connected to an interrupt controller.

Due to the similarity of MIPS and MicroBlaze only a few differences will be discussed. A conditional jump (branch) is taken depending on a register value and zero, while a branch in the MIPS architecture depends on two register values and zero. Therefore an additional adder is necessary to compute the branch target as well as the register values.

The store word (SW) in MicroBlaze uses all register values at the same time as can be seen from the instruction definition $\text{MEM}[\text{Ra} + \text{Rb}] = \text{Rd}$. Therefore all values need to be up to date which implies that in a pipelined implementation all three registers need forwarding logic instead of only the two operands.

Listing 1. Example of the two process design methodology. Signals r and rin as well as variable v are all of type fetch_out_type

```
fetch_o <= r;

PROCESS(fetch_i, r, rst_i)
  VARIABLE v : fetch_out_type;
BEGIN
  IF rst_i = '1' THEN
    v.program_counter := (OTHERS => '0');
  ELSIF fetch_i.hazard = '1' THEN
    v.program_counter := r.program_counter;
  ELSIF fetch_i.branch = '1' THEN
    v.program_counter := fetch_i.branch_target;
  ELSE
    v.program_counter :=
      increment(r.program_counter);
  END IF;
  rin <= v;
END PROCESS;

PROCESS(clk_i)
BEGIN
  IF rising_edge(clk_i) THEN
    IF ena_i = '1' THEN
      r <= rin;
    END IF;
  END IF;
END PROCESS;
```

III. Design

A. Organization

The MB-LITE microprocessor implementation is based on the classic RISC pipeline and includes the stages Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory (MEM) and Writeback (WB). The processor design is based on the MIPS processor implementation as presented in [11] and [12]. All components are designed using a two-process design methodology [10] in order to obtain a reliable and completely synchronous design and to improve both simulation and synthetization speeds. A fragment of this methodology applied to the instruction fetch component is shown in Listing 1. Several modifications to the reference design are applied in order to obtain a MicroBlaze compatible implementation.

The instruction fetch stage handles the control signals of the instruction memory, but does not include the memory itself in order to maintain design modularity. The interface of the instruction memory is defined such that it can be connected to standard ROM or RAM components. Due to the modular approach several extensions (e.g. a bootloader, JTAG) can be easily added.

The decode stage evaluates the instruction and decomposes it into control signals which are used by subsequent stages. Furthermore, the register values corresponding to the instruction operands are fetched from the register file. The reference design assumes that the register file can deal with concurrent reads and writes to the same address, but FPGAs in general do not support this. Therefore optional bypass logic was added to avoid this issue. In order to keep the design synchronous the bypass logic is located in the execution stage.

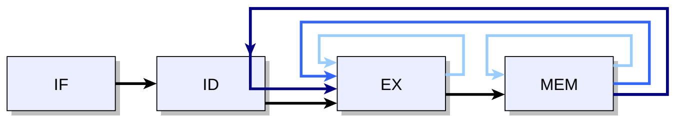

Fig. 1. MB-LITE architecture with focus on forwarding. Lighter arrows represent more recent register values and precede darker values.

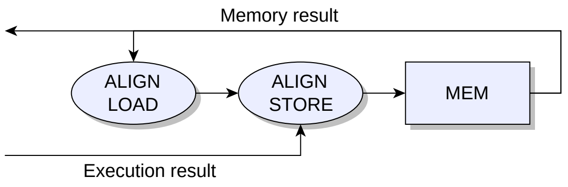

Fig. 2. Forwarding and aligning within the MEM stage

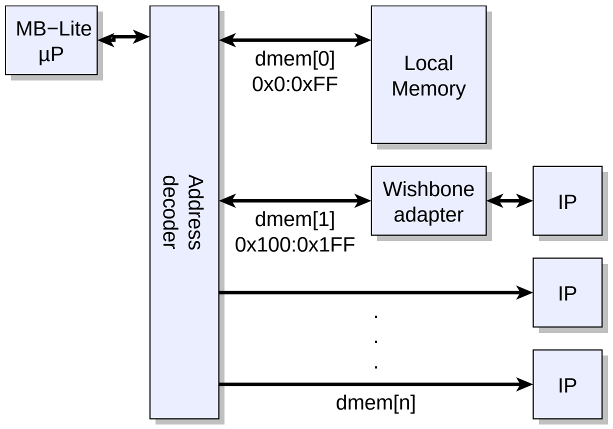

Fig. 3. Example of a memory topology build using the address decoder. Fast local memory is connected to the lower part of the memory map, while multi cycle transactions can be connected using the wishbone bus adapter.

Almost all forwarding logic is postponed to the execution stage. The correct operands are selected first which can come from the output of the execution stage (1-level forward), from memory (2-level forward), bypassed from the register file (3-level forward) or straight from the register (no forward). This forwarding organization is schematically shown in Figure 1. The third level forward is optional since a memory might be designed which is capable of accepting a read and write to the same address during a single clock cycle.

The data memory interface consists of basic control signals and can be connected to any type of synchronous memory. For halfword and byte memory transfers a memory with four independent WRITE ENABLE ports is needed. Alternatively, four smaller components of a single byte each can be used as well. It is up to the designer to select the appropriate memory configuration which is best suited for his target platform. The data memory interface uses a 4-bit select signal which can be used for reading as well as writing.

A data hazard can also occur if the result of a memory read operation is used immediately. Forwarding can be applied to prevent the introduction of unnecessary stalls as shown in Figure 2. Connecting a complex co-processor directly to the memory bus could potentially lead to an increase in logic depth and resource utilization. A parameter has been added to give the designer control over this implementation detail.

B. Data memory bus

To simplify connecting components like memories, co-processors, bridges and adapters a highly configurable address decoder is included to implement a multiplexed bus. The number of output ports and the address map can be completely configured using VHDL generics. The bus is responsible for decoding the selected address and steering the control, address and data signals to the appropriate output ports. It can be connected transparently to the MB-LITE data bus and hence does not introduce additional delay.

Since many peripherals are available for the wishbone bus protocol an adapter is developed to convert the MB-LITE data memory side of the processor to a wishbone compatible master interface. The adapter can be connected directly to the MB-LITE core, but this will introduce at least one delay cycle. The adapter can also be connected to the outputs of the address decoder so the designer has full control over the memory organization. If desired other bus protocols could be designed for the data memory bus interface. An example of a memory topology including the address decoder and bus adapter is shown in Figure 3. The wishbone bus is modeled in accordance with revision 3B of the wishbone specification. Every wishbone transaction takes at least one additional clock cycle with respect to a regular memory transaction in order to complete the synchronous handshake protocol. During a wishbone cycle the execution of the processor is interrupted until the transaction is finished.

C. Framework

To improve design reuse it is important to keep general and specific building blocks strictly separated from the actual design. Just by updating the component library one can take advantage of contributions made by others. The components used by MB-LITE have been distributed over two packages, one containing highly generalized and maybe technology-dependent standard components (e.g. memories and arithmetic components) and the other containing components which are specific for the MB-LITE design. These specific components exist of multiplexors and registers which can be easily synthesized to other platforms. Each design is accompanied by a makefile to quickly generate a ModelSim simulation model.

IV. Results

A. Verification

The behavioral description of the MB-LITE embedded processor has been extensively tested by sending data to a standard input/output interface. This standard interface, which uses the VHDL library TXTIO, is capable of printing characters to the console of a simulator. A wide range of programs was built and successfully tested together with several standard

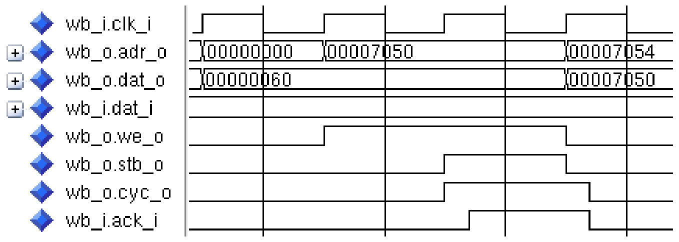

Fig. 4. Plot of a wishbone write cycle.

libraries like STDIO.H, MATH.H and FLOAT.H and the memory operations STRCPY and MALLOC. The Dhrystone benchmark has been ported to MicroBlaze by removing all references to the TIME.H library since these functions are currently not supported. Because there is also no support for exceptions yet it is highly recommended that all software will be compiled in accordance with the ANSI C standard.

After these individual libraries and programs were tested, a large testbench was carefully designed. In order to gain confidence in the quality of the testbench it has been modeled to obtain complete statement coverage which was measured using Mentor Graphics ModelSim 6.5. All statements in the behavioral model are being excercised at least once during this simulation so the execution of individual instructions is very reliable. An estimated total of 1.5 million instructions are being executed during this simulation, and includes integer arithmetic and floating point libraries, iterative function calls, string and memory operations and the complete Dhrystone benchmark.

The data memory bus components have been used within the same verification procedure. A memory topology was set up using the address decoder and the wishbone bus. The lower part of the memory was connected to a standard block RAM while the upper part of the memory connected through the wishbone bus adapter to a wishbone compatible IO device. During the tests of the wishbone bus the acknowledge delay of the peripherals was changed to see if all rules and exceptions of the wishbone bus specification were met. An example of a wishbone write cycle is shown in Figure 4.

A VHDL netlist was generated from the behavioral model using ISE 10.1. The functionality of the synthesized design was successfully verified by comparing the simulation of the behavioral model with the simulation of the synthesized model.

B. Performance

The performance of the MB-LITE processor has been compared to several other designs. All designs were synthesized using a Virtex 5 development board (XC5VLX110-3FF1760) using Xilinx XST 10.1.03. Since it is often interesting to obtain a small and fast implementation in favor of a large and complex one, the cores of all processors have been configured with as much options disabled as possible using default synthetization options. Therefore the focus is primarily on the functionality of the core.

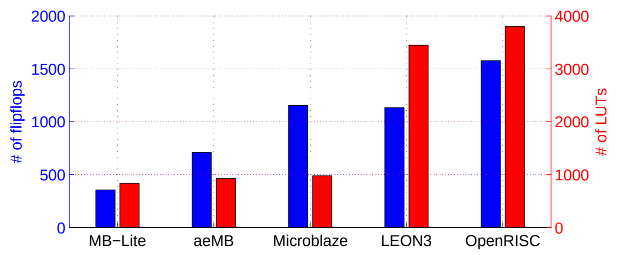

Fig. 5. Resource utilization of the cores of several processors, in LUTs and flipflops

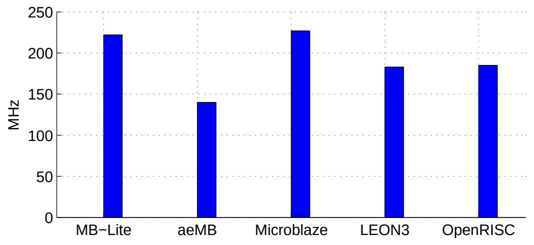

Fig. 6. Performance of the cores of several processors, in MHz

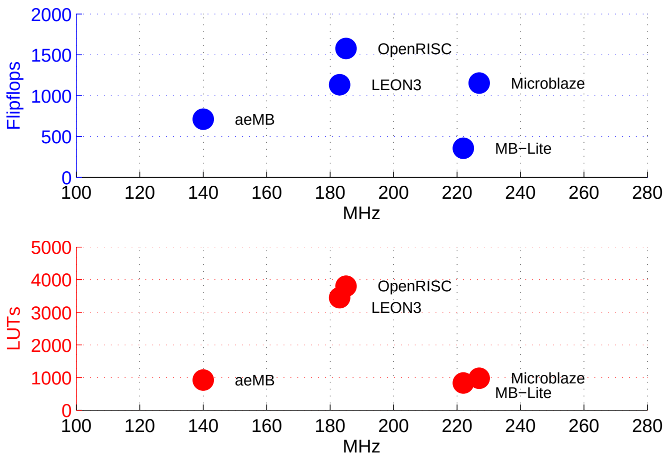

Fig. 7. Cost-performance trade-off for various processors

Performance has been measured in terms of clock frequency and resource requirements and compared with three other processors. The results of these measurements are shown in Figures 5 and 6[1]. From these figures it can be seen that the performance of MB-LITE in terms of clock frequency is largely comparable with MicroBlaze, while much fewer hardware resources are required. In Figure 7 the tradeoff between performance and resource utilization of these processors is shown. Obviously the bottom right corner is the most favorable position since a lot of performance can be expected using only few resources.

The performance of the design might be improved even more in the future. The multiplier and barrel shifter can

[1]The clock frequency of aeMB was measured at 279 MHz. AeMB is a two-threaded barrel processor but since we are interested in single-threaded performance the final result is devided by two to obtain a better comparison.

be spread over two clock cycles to reduce their impact on the clock frequency. The decode and execute stages can be pipelined deeper which would result in an implementation with seven pipeline stages. Another approach to improve the clock frequency could be to replace the single clocked register file with a dual clock register file which support concurrent reads and writes to the same address. This will make a significant part of the forwarding logic obsolete. It has to be noted that the introduction of a dual clocked RAM might introduce difficulties with respect to clock skew or other timing issues.

C. *Configurability*

Several important features can be changed to obtain a faster or smaller device, or to make the design suitable for different memories. For this purpose constants are defined which will act as default values for the parameters, while VHDL generics can be used to override default behavior upon instantiation. To save hardware resources the processor can be configured to stall upon hazards caused by subsequent reads and writes. Alternatively, forwarding can be applied to transparently resolve this issue. Support for interrupts can be disabled although it was found that this hardly influences the performance or resource utilization of the processor.

A hardware multiplier and barrel shifter for integer numbers can be included in the design using configurable parameters. By enabling these components a significant reduction in clock frequency should be expected since these components are modeled as part of the ALU. It might therefore be worthwile to substitute these operations with software libraries.

V. Conclusion

In this paper the design and implementation of a highly configurable opensource embedded processor was presented[2]. MB-LITE is a light-weight implementation of the MicroBlaze architecture, and is designed to obtain high performance using few logic elements. This has been achieved by applying a synchronous two-process design methodology in which combinational and sequential logic is strictly separated. Comparisons with other processors show that MB-LITE can achieve equivalent performance as the highly optimized MicroBlaze while using far less resources.

Conformance with the MicroBlaze architectural specification was thoroughly verified using both behavioral as well as netlist simulations with anotated time information. The design is cycle as well as architecturally compatible with MicroBlaze. MB-LITE can thus be used to replace MicroBlaze in most designs.

The two-process design methodology is not only used to improve design speed, but also to increase readability, facilitate code maintanance, reduce the code-size and improve simulation and synthetization speeds. Using modularity as basic rule, a highly configurable multiplexed bus and wishbone bus adapter can be attached easily.

All components use the technique of inferring logic instead of explicitly instantiating technology dependent components. Therefore this design can be implemented easily on different platforms. The separation between standard and specific components makes that the design can be relatively easy implemented using IC process technology without having to change the structure of the design.

The small size of the MB-LITE processor, the modularity of the bus as well as the incorporation of the two-process design methodology makes this processor very well suited for research and development of Very Large Scale Integrated Systems On Chips and On-Chip Networks. Future research will focus on embedding the MB-LITE in a reconfigurable fabric as well as implementing this processor in a UMC 90 nm process technology.

[2]All project files are published on http://www.opencores.org/project/mblite

References

[1] D. Sheldon, R. Kumar, F. Vahid, D. Tullsen, and R. Lysecky, “Conjoining soft-core FPGA processors,” *Computer-Aided Design, International Conference on*, pp. 694–701, 2006.

[2] F. Plavec, B. Fort, Z. G. Vranesic, and S. D. Brown, “Experiences with soft-core processor design,” in *IPDPS ’05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium.* IEEE Computer Society, 2005, p. 167.2.

[3] R. Holsmark, A. Johansson, and S. Kumar, “On connecting cores to packet switched on-chip networks: A case study with MicroBlaze processor cores,” 7th IEEE Workshop DDECS 04, april 2004.

[4] K. Goossens, J. Dielissen, and A. Radulescu, “Æthereal network on chip: Concepts, architectures, and implementations,” *IEEE Design and Test of Computers*, vol. 22, no. 5, pp. 414–421, 2005.

[5] T. Kranenburg, “Reference design of a portable and customizable microprocessor for rapid system prototyping,” Master’s thesis, Delft University of Technology, 2009.

[6] *MicroBlaze Processor Reference Guide*, Xilinx, January 2008.

[7] D. Mattsson and M. Christensson, “Evaluation of synthesizable CPU cores,” Master’s thesis, Chalmers University of Technology, 2004.

[8] F. Plavec, “Soft-core processor design,” Master’s thesis, University of Toronto, 2004.

[9] S. Craven, C. Patterson, and P. Athanas, “Configurable soft processor arrays using the Openfire processor,” in *Proceedings of 2005 MAPLD International Conference*, 2005, p. 250256.

[10] J. Gaisler, “Fault-tolerant microprocessors for space applications,” pp. 41–50, http://www.gaisler.com/doc/vhdl2proc.pdf.

[11] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, 3rd ed. Morgan Kaufmann Publishers, 2003.

[12] ——, *Computer Organization and Design: The Hardware/Software Approach*, 3rd ed. Morgan Kaufmann Publishers, 2005.