Buletinul Ştiinţific al Universităţii "Politehnica" din Timişoara

Seria ELECTRONICĂ şi TELECOMUNICAŢII
TRANSACTIONS on ELECTRONICS and COMMUNICATIONS

Tom 49(63), Fascicola 1, 2004

# Standard Sockets Revolution in ASIC Design

Traian Tulbure, Razvan Jipa, Dan Nicula[1], Adam Levinthal[2]

**Abstract – The current approach for designing an integrated circuit as SoC (System-on-Chip) is based on reusing the models for modules with a well-defined functionality. For easier interconnection, these IP cores should have an interface that obeys the rules of a standard socket.**

**Ideally, a SoC socket would enable core designers to concentrate mainly on the core functionality. Similarly, SoC system integrators should be able to concentrate mainly on the SoC timing, core service bandwidth and latency requirements, and the final floor-plan design independent of core functionality. The socket would therefore provide the necessary physical and exchange protocol delineation necessary to achieve this well-defined layering.**

**This paper presents an integrated circuit implementation that uses two standard sockets. It also depicts the problems and the corresponding solutions for socket conversion and interconnection.**

**The presented design has been checked in synthesis and it was proved by an ASIC implementation.**

**Keywords: SoC, OCP, Wishbone, IP core, HDL, Digital Integrated Circuits, ASIC.**

## I. INTRODUCTION

Intellectual-property (IP) cores can offer a tremendous advantage to designers building complex system-on-a-chip devices. By acquiring pre-built functions, designers can shorten design time and focus on the application—providing value-added elements that emphasize the design organization's application expertise and corporate competencies. To work with the variety of cores in the market today, designers need to focus on a specific strategy for managing IP interconnect to ensure efficient integration and system-level verification.

## II. STANDARD SOCKETS IN ASIC DESIGN

For easier IP core interconnection there are used several standardized SoC busses and sockets as: AMBA, ClearConnect, CoreConnect, OCP, SilliconBackplane, Wishbone and many others. A design can use any these standard sockets for IP core interconnection and also more sockets can be used in the same design.

This paper will focus on OCP (Open Core Protocol) and Wishbone sockets that will be used in the presented integrated circuit implementation.

Open Core Protocol (OCP) is a common standard for intellectual property (IP) core interfaces, or sockets, that facilitate "plug and play" design. To make complex SoC design a reality for a broader audience, the industry needs a complete socket standard that everyone can use, no matter what their on-chip architecture is, or whose processor cores they're using. The OCP-IP CoreCreator™ tool automates the tasks of building, simulating, verifying and packaging OCP-compatible cores. IP core products can be fully "componentized" by consolidating core models, timing parameters, synthesis scripts, verification suites and test vectors in accordance with the OCP specification.

The 'Wishbone' Interconnection Architecture for Portable IP Cores is a flexible design methodology for use with semiconductor IP cores. Its purpose is to faster design reuse by alleviating System-on-Chip integration problems. This is accomplished by creating a common interface between IP cores. This improves the portability and reliability of the system, and results in faster time-to-market for the end user. OpenCores recommends the Wishbone Interconnect as the interface to all cores that require interfacing to other cores inside a chip FPGA or ASIC.

## III. SOC ARCHITECTURE

The IP core described in this paper is part of a bigger platform called FlexASIC. The FlexASIC structured-ASIC products are designed as general-purpose configurable logic devices with standard-cell speed, density, and production costs, and FPGA ease of use and prototype costs. Each member of the FlexASIC

[1] Dept. of Electronics and Computers, TRANSILVANIA University of Braşov, Romania
29 Eroilor, 500036 Braşov, Romania, e-mail: tulbure@vega.unitbv.ro, jipa@vega.unitbv.ro, nicula@vega.unitbv.ro
Tel. +40-268-414482, Tel./Fax +40-268-478705

[2] eASIC Inc. 3555 Woodford Drive, SanJose, CA 95124, USA, e-mail: adam@easic.com

product family contains an embedded 8051-based uController.

The eASIC Manager-uProcessor (eMμ is the configuration and control module inside of FlexASIC structured-ASIC devices. eMμ performs built-in self test (Logic, LUT and bRAM BIST, etc), logic bitstream initialization, and clock configuration of the FlexASIC at system reset. After initialization, eMμ can (optionally) implement a number of system-level functions for the FlexASIC user design, including SPI (flash) memory management, system timer functions, and clock/power management functions. eMμ contains a flex8051 uController that operates from 128Kbits (16KBytes) of on-chip ROM. FlexASIC developers may write custom application code for the flex8051 in eMμ to extend the capabilities of their system ([7]).

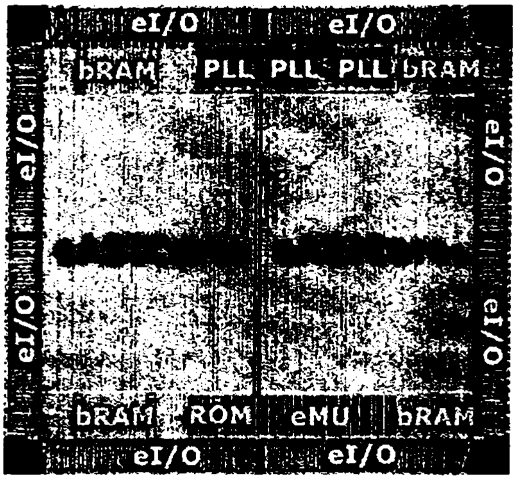

Figure 1: eMμ placement in FlexASIC

The placement of eMμ inside the die plot of the smallest FlexASIC device is illustrated in Figure 1.

eMμ was designed using a modular design approach based on the 'Open Core Protocol' (OCP) interface standard. All eMμ modules interface through OCP sockets and are inter-connected through a switch fabric created with OCP merge and split modules. The socket-based design approach provided numerous benefits during the design and implementation process, allowing the entire design to be created, from architectural specification to tapeout, in 12 weeks. This could be achieved by reusing the flex8051 processor core and a SOC debug interface provided by Flextronics semiconductor and also available in the public domain.

The eMμ module is designed around the flex8051 uController core and nine sub-modules. The uController and sub-modules are connected through an OCP switch fabric. The overall eMμ block diagram is shown in Figure 2.

The flex8051 core had 'Wishbone' ports (maintained at www.opencores.org) as its primary interfaces for instruction and data accesses. These ports were converted to byte-wide OCP socket connections through Wishbone to OCP (wb2ocp) adapters. All of the eMμ sub-modules were connected through an 'OCP-switch' created with combinatorial (unpipelined) OCP merge and split modules.

The OCP switch was implemented as a 5:1 merge (M0 ...M3) followed by a 9:1 split (S0 ...S7). The 4:1 merge limits the overall throughput to one transfer per clock. No OCP pipelining was supported, but read prefetching was implemented to improve the performance of slower modules (such as romPort and spiPort). The merge module implementes fixed-priority arbitration with a priority ordering of:

*jtagPort port (highest) -> flex8051 debug -> flex8051 Inst -> flex8051 data -> OCP user port (lowest)*

The splitter portion of the OCP switch implemented address-based splitting based on the most significant bit (MSB) the incoming master address value. The organization of split modules was chosen to im lement the address map required for eMμ.

The first 4Kbyte page of the OCP base memory region is designated as the 'bootPage' and is assignable to viaRom, SPI memory, or bRAM memory. The bootPage mapping is selected through a control register. This register is initialized at system reset to map the bootPage to viaRom, SPI memory, or bRAM memory. Boot page accesses complete in a variable number of clock cycles, depending on the type of memory that is mapped. This assignation of the bootPage is realized by the bootMux in the OCP switch.

The eMμ sub-modules are listed in Table 1. The sub-modules were designed with byte-wide OCP ports as their native interfaces. The top-level module integration was done using the CoreCreator tool from OCP International Partners (www.ocpip.com). CoreCreator provided generic master and slave modules for initial development use, and was used to add OCP protocol checkers to all the OCP connections in the design.

| Port Name | Port Function |
|---|---|
| jtagPort | JTAG Tap Controller |
| ecPort | eCore Array configuration interface. |
| brPort | bRAM Array interface (through BIST ports) |
| cgPort | ClkGen serial register interface port |
| flexPort | Interrupt, GPIO, and system register interface |
| romPort | OCP controller for FlexASIC viaROM. |
| spiPort | Off-chip SPI memory controller. |
| ipPort | OCP Initiator Port to user design |
| tpPort | OCP Target Port from user design |

Table 1: eMμ sub-modules

All sub-modules are OCP compatible and some of them were redesigned from a previous platform, design that also used the OCP standard socket for IP interconnection.

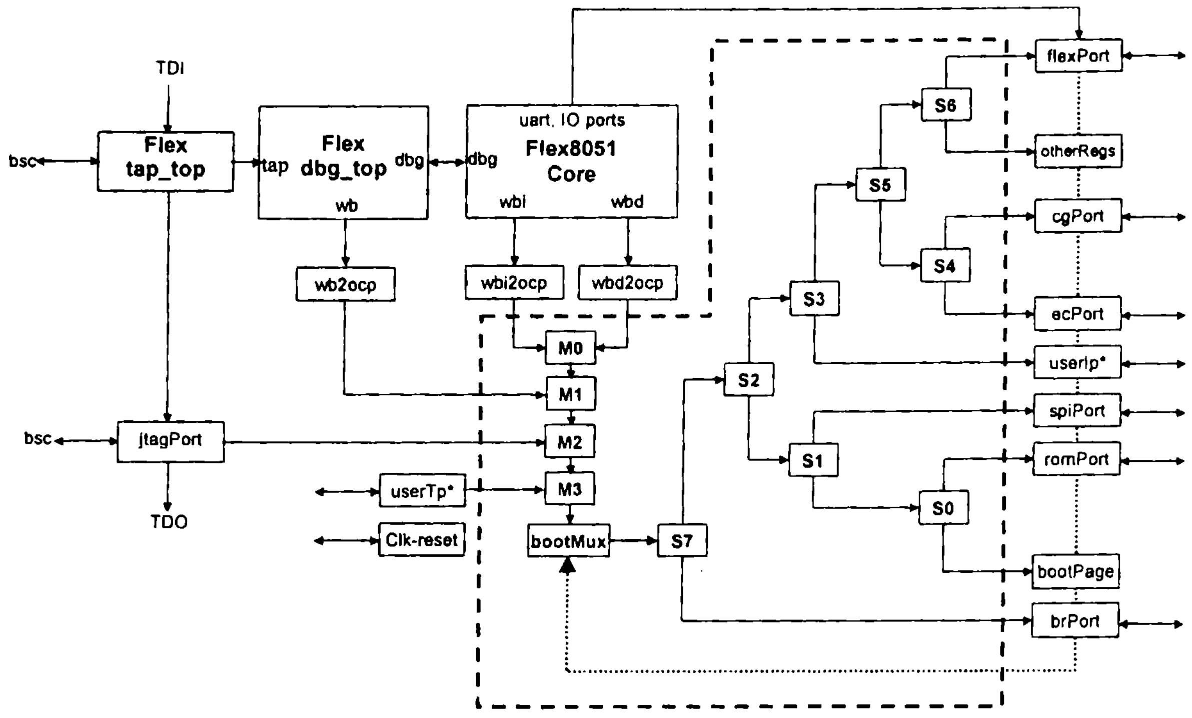

Figure 2: eMμ internal architecture

Having to reuse modules that use different connection sockets lead to design of the socket conversion bridges.

## VI. SOCKET BRIDGES

The chapter depicts the methods of Wishbone-to-OCP conversion (wb2ocp modules), conversion issues and their corresponding solutions.

The Wishbone to OCP (wb2ocp) converters are used to adapt the flex8051 uController to OCP-compliant peripherals. The wbd2ocp module converts the Wishbone data interface to OCP and the wb2ocp module converts the Wishbone instruction to OCP. There is also a Wishbone-to-OCP conversion performed between the uController debug interface Wishbone port and the rest of the SOC.

Any Wishbone compliant module uses either the single read/write or the block (burst) transfers. The flex8051 implements a single read/write transfer – any transfer consists of only one phase of read or write. There can be different types of transfer cycles signaled from the master to slaves through the Wishbone interface ([2]).

To maximize the bandwidth of the transfer the wb2ocp bridges use a speculative-address mechanism based on the burst transfers issued by the master (flex8051). An incremental burst is defined as multiple accesses to consecutive address. The increments can be linear or wrapped. A wrap incremental transfer means that the address increments one but the LSBs of the address are modulo the wrap size. The number of the LSB's used for a wrap transfer are determined by the transfer beats number and the data bus width.

The flex8051 uController use a 4-beat wrap burst to transfer 4 32-bit words from the 16 consecutive addresses of an 8-bit data bus slave. This transfer involves only the 4 LSBs of the address. The burst transfer requires the slave to perform 4 accesses for any given request of the master and then to merge all the data into one response. This is the base for the speculative address since the order of the addresses is known apriori for a given burst transfer.

| | | | | |
|---|---|---|---|---|
| master address 4 LSB's (hex) | 4 | 8 | C | 0 |
| slave generated address 4 LSB's (hex) | 4 5 6 7 | 8 9 A B | C D E F | 0 1 2 3 |

Figure 2: 4-beat wrap burst address generation

The uController Wishbone data interface is converted to an OCP interface in a simple manner because the transfers are single read/write cycles without any wait states inserted by the master (i.e. CPU). This translates in the following correspondence rule between the Wishbone interface control signals (wbdWe and wbdStb the same with wbdCyc) and the OCP command:

ocpMCmd={0, (wbdStb_i & !wbdWe_i), (wbdStb_i & wbdWe_i)}

The Wishbone interface address, output data and input data are copied directly in the corresponding OCP address, output data and input data busses ([3], [5], [6]).

## V. TESTING

The eMµ module design had 272 IO signals at the top-level.

To create a flexible test environment for all eMµ interfaces, two copies of the core design were connected in a 'back-to-back' fashion in the top-level testbench. Figure 3 illustrates the top-level testbench created for eMµ.

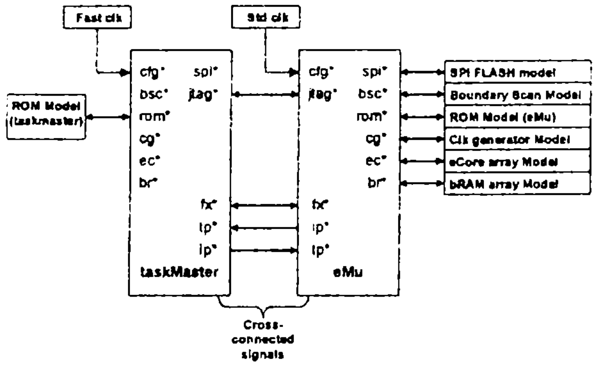

Figure 3: eMµ testbench

The first instance was the main eMµ device under test. The second instance, called taskMaster, was used as a general traffic generator and command co-processor. The signals in the User bundles (OCP ports, GPIO ports, interrupts, counter/timer triggers, and UART signals) were cross-connected between eMµ and taskMaster, allowing these signals to be tested in a flexible way. For example, taskMaster and eMµ could send commands to each other over their UARTs, or create traffic on each other's internal busses over their OCP ports. Protocol checkers instantiated in CoreCreator® ([4]) were used to ensure that each instance produced legal transactions on these OCP ports. A complete 'Wishbone' monitor was instantiated on all wishbone ports. A command protocol was created over the UART serial ports to orchestrate operations between the two devices. For example, eMµ would instruct taskMaster to start random SPI memory access traffic via the OCP ports, and taskMaster would instruct eMµ to reset itself ([10]).

The taskMaster model was driven from an artificially fast clock when high-traffic testing was needed and could issue back-to-back OCP transfers through the eMµ OCP target port while the normal eMµ test programs were running.

All the tests were wrote in embedded C for 8051 and when a test encounters an error in the eMµ, the error handler is called to report the error by writing an error code to the "print module". The error code is used by as index to the error message table by the print module to display the error message string. Depending on the severity of the error, the error handler can either return to the caller if it is a non-fatal error or it might issue a halt "JMP $" instruction for fatal error.

| Test category | Test description |
|---|---|
| basicRdWr test | This test performs a random sequence of pushes and pops to a stack in bRAM memory. Pushes write a variable-length data payload of random values, a 32-bit CRC, and an 8-bit 'length' value. Pops read the length value and pop that number of bytes from the stack, compute the CRC for the payload, and confirm that it matches the CRC fetched from the stack. |
| linkedList test | A linked-list is pre-computed and initialized in memory. A running CRC is computed from the links traversed. The linked list is followed until it a link of 0xFFFF is encountered. At that point, the current CRC is compared to the 32-bit value stored in addr+2. If the values match, a 'test pass' message is posted. If the value does not match, a 'test failed' message is posted. At that point, the test fetches the 16-bit values at addr+6 and continues. The test repeats when a link of 0x0 is encountered. |
| eCoreArray access test | This test consists of getLUT (), setLUT (), and traverseLUT() routines that are used to test various ranges of the LUT and FlipFlop scan-chains. The test issues a random sequence of these operations, verifying that the getLUT() operations return what was set using the setLUT() call at a previous step in the test sequence |
| memory relocation test | A small relocatable program is written to memory. The program copies itself to another memory area, jumps to the new location, and erases the previous program area. This program is scheduled by timer interrupts and run concurrently with other tests |
| soft reset test | This test writes known data to bRAM, saves the boot reason code in flex8051 data memory and issues a soft reset to reboot. After rebooting, the Emu examines the boot reason code and if it indicates a soft reset, the CRC for bRAM is compared against a pre-computed value. |
| traffic generator test | While any of the above tests are running, eMµ instructs taskMaster to start memory read traffic to bRAM memory and viaROM through the OCP target port. Writes are restricted to regions of memory not being used by currently active tests. |
| debug interface test | Taskmaster selects CPU debugging, reads the program counter and accumulator, stalls CPU, flushes CPU instruction cache, write instructions opcode to cache and unstall CPU. |

Table 2: eMµ testing

| Iteration | Critical path (ns) | PrimeTime® analysis (critical path) (ns) | Levels of logic | Area (μm$^2$) |
|---|---|---|---|---|
| Synthesis – 1st pass | 9.4 | 14.15 | 44 | 400000 |
| Re-synthesis with net delay | 12.47 | 13.25 | 51 | 409000 |

Table 3 Implementation results

Another testing method used was stress testing. The overall approach to eMμ stress testing is to write simple tests based on expected functional behavior of each individual module, and create a main 'test scheduler' that calls each tests in an interleaved fashion while introducing maximum variation in the test conditions across all tests ({8},[9]).
The testing phase continued after the real chip manufacturing. The SOC was designed in such way to ensure access point to almost any important register of the each sub-module. This was possible in an easy fashion due to the JTAG port and the OCP compliant socket architecture of the chip.
The JTAG port proved to be a crucial feature that allows testing of every feature of the eMμ sub-modules. The entire suite tests run for RTL and gate netlist simulations were used to prove that the chip works.
Designing a test environment close to the one used for chip testing after manufacturing proved to be very useful due to the easiness of JTAG stimulus generation. The test vectors are obtained from the RTL simulation by observing and dumping into a file the JTAG bundle signal values.

## VI. IMPLEMENTATION

Section presents the implementation results of the SoC in ST 0.13μ standard cell technology. The chosen synthesis strategy was the top-bottom one using the Synopsys Design Compiler® target for 0.13μm ST process. Place and route stage was accomplished using the Cadence Silicon Ensemble®.
Improvements were to be made in order to improve the critical path of the design. To improve the timing the net delay information was supplied to the synthesizer together with the placed&routed netlist. This 2nd synthesis iteration generates a more place&route suited netlist because the synthesizer was aware of the additional time penalty incurred by the net delay and tried to re-structure the netlist accordingly.
The final placed and routed netlist was analyzed with a static time analysis tool - Synopsys PrimeTime. The final analysis included parasitic information about the design and the silicon die.

## VII. FURTHER DEVELOPMENT

The STA analysis shown that the critical paths owe to the signal bundle that connects eMμ to the bRAM banks. This problem can be solved by a redesign of the bRAM interface inside eMμ that will cut the critical path by registering all the outputs to the bRAM banks. These critical paths appeared due to the floor plan of the design that placed the farthest bRAM in the opposite corner of the die than eMμ (see Fig 1).
Another improvement that can be added is the OCP syncronizers for the user ip and tp ports. Since these ports assure communication with the user design placed on the eASICCore® array the data and control signals passed between eMμ and user design must be synchronized and must maintain OCP compliance. Since there can be no assumption regarding the eMμ and user design clock ratio the only valid solution could be an OCP synchronizer using FIFO semantics.
This synchronizer could use 2 paths:

- one for the master command;
- one for the slave response.

The key of synchronization of a bundle is to synchronize only a 1-bit signal preferably a pulse that becomes active after the entire bundle is latched. When the load signal is synchronized on the other clock domain the entire bundle can be latched onto the target clock. It can be looked as a 1 level depth dual clock FIFO whose implementation is simplified because of the particular depth (requires toggle flip-flops as 1 bit Gray counters).

## VIII. CONCLUSIONS

Cores designed for reuse with standard sockets ease the design effort considerably. Although a lot of cores use standard sockets, they use different standards and one needs to connect them using bridges that do not affect the socket's performance. Bridging standard sockets is a key point in designing large SoC with less design effort and very high reliability.
Converting one standard socket to another is driven by a set of decision. The main factor is the number of the IP using a standard socket and the possibility of reusing it. Another factor that favors the choosing of one standard socket over the other is the amount support offered. The software tools offered by Sonics for OCP socked based design developers take the test-bench and protocol integrity burden off the developer's shoulders. It also provides to the user with an of-the-shelf IP library that are OCP compliant and can be hooked together without considerable design effort.
The use of a commercial uController core (the flex8051) and OCP socket connections to all peripherals, as well as uController-based testing written in C, allowed design and testing of individual

modules in eMμ to occur in parallel. The ability to re-use unit tests in the full-module final testbench using a top-level test sequencer eliminated the need for additional full-chip test development. The entire process represents a streamlined approach to uController-based ASIC design based on standard sockets.

This project was developed in the frame of the research program between the eASIC Inc. and the "Transilvania" University of Brasov.

## REFERENCES

[1] *** www.easic.com

[2] *Wishbone System-on-Chip (SoC). Interconnection Architecture for Portable IP Cores.* Revision: B.3, Released: September 7, 2002

[3] *Open Core Protocol Specification.* Version 1.2. Sonics Inc. 2000

[4] *FastForward Development Environment - FastForward Tutorial*, Revision B03, Sonics Inc., 2001

[5] J. Bhasker, *A Verilog Primer*, Second Edition, Star Galaxy Publishing, 1999

[6] D.E Thomas, P.R Moorby, *The Verilog Hardware Description Language*, Fourth Edition, Kluwer Academic Publishers, 1998

[7] M.J S. Smith, *Application-Specific Integrated Circuit*, Addison-Wesley, 1997

[8] Simon Teran, Marko Mlinar, *Flex8051 Design Documents*, Revision 0.2

[9] Igor Mohor, *SOC Debug Interface*, Revision 2.0

[10] Janick Bergeron, *Writing Testbenches:Functional Verification of HDL Models*, Kluwer Academic Publishers, 2000

[11] *** www.opencores.org

[12] *** www.sonics.org

[13] *** *www.solvnet.synopsys.com*