CUSPARC IP Processor: Design, Characterization and Applications

Ezz El-Din O. Hussein, Shoukry I. Shams, Mohamed I. Ali, Amr A. Z. Suleiman, Khalid ElWazeer,
Ehab A. Sobhy, Ahmad A. I. Ibrahim, Ahmed M. G. Ibrahim, Mohamed S. Khairy, Mohamed F. Fouda,
Al-Hussein A. El-Shafie, Ahmed H. M. Hareedy, ElSayed A. Ahmed, Ahmed R. Zakaria,
Khalid M. El-Galaind, Amr A. El Sherief, and S. E.-D. Habib[1]
Department of Electronics and Communications
Cairo University
Cairo, Egypt
[1]Corresponding author: seraged@ieee.org

***Abstract*— In this paper, we introduce the design of an IP processor core code-named CUSPARC for Cairo university SPARC processor. This core is a 32 bit pipelined processor that conforms to SPARC v8 ISA. It is complete with 4 register windows, I and D caches, SRAM and flash memory controller, resolution hardware for the data and branch hazards, interrupts and exception handling, instructions to support I/O transfers, and two standard WISHBONE buses to support high speed and low speed IO transfers. The design was downloaded and tested on different FPGA platforms, in addition to 0.35µm and 0.13µm ASIC technologies. CUSPARC has a promising metric of 0.9663 DMIPS/MHz. A novel debugger tool was developed for validating CUSPARC. This tool facilitates the testing of the processor running complex software loads by invoking Mentor's MODELSIM simulator in the background while maintaining a "simulator-like" GUI in the foreground.**

***Index Terms* — IP processor, processor design, SPARC, CUSPARC,**

I. INTRODUCTION

IP processors are proliferating across embedded systems at an amazing rate. These systems impose demanding constraints on power and cost while seeking to fulfill the throughput requirements of the application considered. Conventional designs focusing mainly on the throughput metric and centered on high performance – albeit power hungry - processors are no longer appropriate for these embedded systems. Modern embedded systems will most likely be based on light-weight, power efficient IP processors mushrooming across a single or multiple ICs to form an energy efficient solution that meets the throughput demands of the application. This paper introduces the Cairo University SPARC (CUSPARC) processor as an IP processors keyed to these application domains.

SPARC is a CPU instruction set architecture (ISA), derived from a (RISC) lineage, and published by Sun Microsystems in 1986 [1]. The SPARC architecture has the following attractive features:

- *Open standard*: Various SPARC assemblers and compilers are available.
- *RISC architecture*
- *Windowed register file*: the architecture contains variable number (implementation dependent) of overlapped registers windows. This register file organization speeds up context switching and reduces memory traffic leading to performance improvement.

The Cairo University SPARC (CUSPARC) processor conforms to SPARC ISA V8 standard [2]. CUSPARC is the first fully-operational Egyptian processor. It was fully designed at the Electronics and Communications department, Cairo University, Egypt during the past five years.

The rest of the paper is organized as follow: in section II the architecture of our design is presented. Section III shows the software tools developed for CUSPARC. Tests & applications are presented in section IV. Finally, performance metrics are given in section V.

II. PROCESSOR ARCHITECTURE

The main components of the CUSPARC processor are the Integer Unit (IU), caching system, cache controller, memory controller, a standard WISHBONE bus, Boot loader, and the peripherals. Figure 1 shows the bock diagram of the processor.

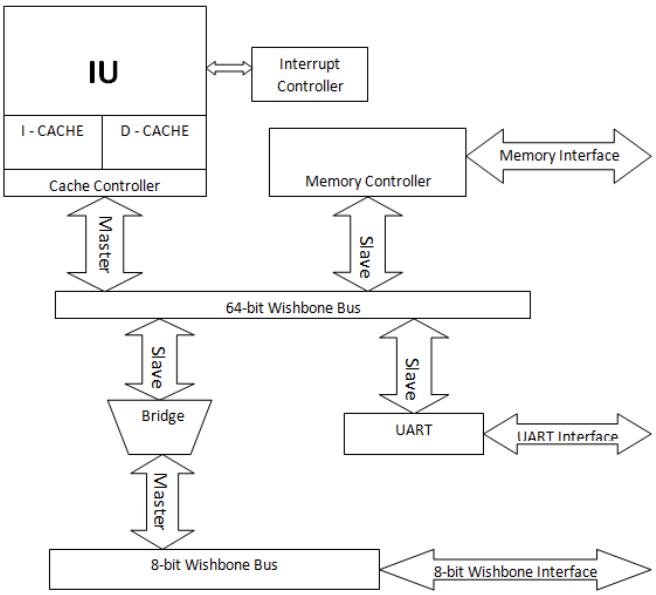

Fig. 1: CUSPARC Architecture

A. *The Integer Unit (IU)*

The IU is a 32 bit, 4-stage pipelined unit. The 4 stages are Fetch, Decode, Execute, and Write back. The memory read/write stage is specially handled so as not to disrupt the pipeline. The instruction set includes about 80 different instructions including Load/Store, Arithmetic/Logic/Shift, Control transfer and Read/Write Control Register Instructions. The CUSPARC IU includes four register file windows. At any one instant, a program sees a large register file made of 8 global registers plus a 24-register window (8 inputs, 8 local and 8 output registers). The windows are overlapped as shown in Figure 2. This overlapping reduces the memory traffic when going up and down the procedure call. It also enhances the CPI (Clock per Instruction) of the processor as it alleviates the need for the stack memory.

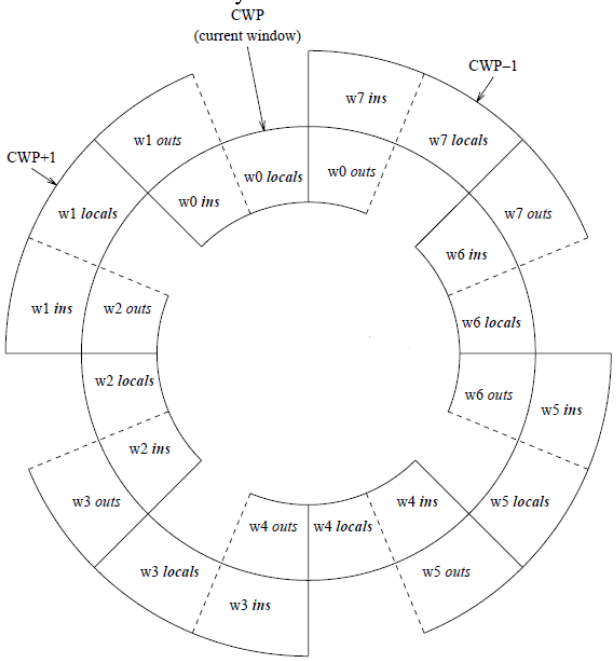

Fig. 2: Register File Windows

B. *Caching System*

CUSPARC has a Harvard Architecture, with separate instruction cache (4 KBytes) and data cache (4 KBytes). Both caches are direct mapped. The I-Cache is just a simple finite state machine that reads the instructions from the main memory in case of a cache miss. On the other hand, the D-Cache is more complex as, in our design, it is required to perform read and write operations, and also deal with data transfers from/to I/O devices. In the later case, no data is stored in the static ram of the D-cache. Rather, the D-cache simply sends an I/O (read/write) request to the cache controller to read or write the data from or to the required I/O device respectively.

C. *Cache Controller*

The cache controller works as a server to respond to the cache miss requests from the two caches or to I/O access requests. The cache controller is the only master on the Wishbone Bus. This makes the bus implementation simpler as there is no need for a bus arbiter. The major tasks attached to the cache controller are as follows:

- Scheduling the incoming requests from I and D caches.
- Interfacing to the WISHBONE bus [2].

D. *Memory controller*

This controller interfaces CUSPARC to the main memory (RAM) or the boot memory (Flash). It maps the address from caches to the external memory. This mapping is based on the memory organization. The design of the controller handshaking timing diagram with external memory is generic and is controlled by software. The software should include the number of clock cycles needed by main memory and boot memory to respond. This provides flexibility to use different RAM or FLASH memories with different speeds.

E. *WISHBONE Bus*

The design includes two structured Wishbone Busses:

- An internal 64-bit bus for interfacing the cache controller as a master to the memory controller and peripherals as slaves.
- An external 8-bit bus, to provide slow I/O interface for CUSPARC.

A bridge (FIFO) is used to connect the two buses.

F. *Boot loader*

The Boot Loader is implemented as a simple Direct Memory Access (DMA) device that is used only to move the Text and Data sections from the boot memory to the main memory. This device is totally guided by software to determine when to start, and give it the exact location of the source and destination of the Text and Data that will be moved.

G. *Peripherals*

CUSPARC has 2 UARTs, 3 timers and a watch dog timer. There's also an interrupt controller, supporting 3 different interrupts.

III. SOFTWARE TOOLS

A. *CUSPARC Compiler*

The open-source LEON GCC cross compiler (BCC – Bare C Cross Compiler) [3] was adapted for our processor. A small software parser was developed to remove LEON processor initializations from the compiler output and insert instead the CUSPARC initializations. These initializations include initializing the stack pointer, setting memories timing (RAM & FLASH), filling trap table and trap handlers. A function library was developed to facilitate testing CUSPARC both in simulation and real time environments. This library includes basic assembly functions for watching and validating variables during simulation such as (watch) and (assert) functions. These functions make it easy to trace where an error happened by stopping simulation immediately after the error. Many other functions are coded inside the library to write to the IO ports and the UARTs available on the processor.

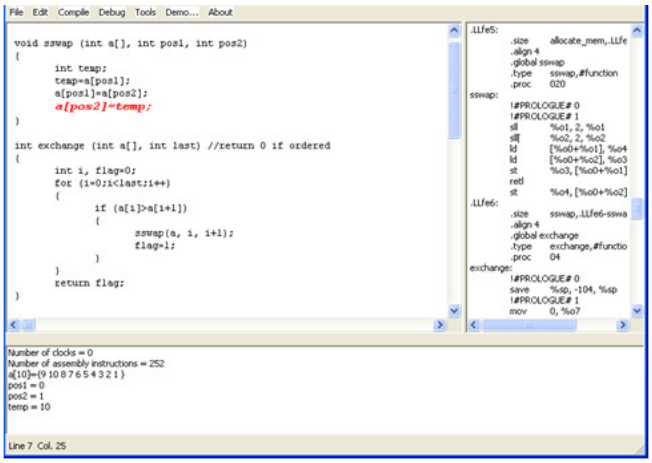

Fig. 3: CUSPARC IDE

B. *CUSPARC Debugger IDE*

CUSPARC debugger is different from the traditional debugger as it is developed to test the hardware design itself, in addition to the software. There are two versions of the debugger; one for simulation and the other for real-time debugging. These two debuggers proved invaluable for uncovering hardware bugs in odd processor "states".

1) *Simulation debugger:*

CUSPARC supports over 80 instructions and over 20 traps (exceptions or interrupts). At any instant of time, 4 instructions are in progress inside its pipeline. This creates a huge number of processor states that are very difficult to trace using dynamic timing simulators like Mentor's ModelSim. The basic idea behind the CUSPARC simulation debugger is to invoke the MODELSIM simulator in the background while maintaining a "simulator-like" GUI in the foreground.

The above-mentioned GUI makes it easy for users to select the variables to watch and to track the output step by step. When debugging starts, the program scans the C file to find the variable names and their type. After that, the user selects the variables to be watched in addition to any processor registers. The debugger then modifies the C code by inserting watches after each line for each variable elected. The new C file is compiled again and fed to ModelSim to simulate it on the processor. After the simulation is completed, our GUI parses the output dump file to extract the variables for each C line. The user finally can scan step by step the code and see how these variables are changing. A snap shot of the GUI is shown in Figure (3). It was written using Microsoft C# .NET.

The important note here is that during simulation, we verify both the software code and also the hardware VHDL code since the simulation is performed for the VHDL model of the processor. Using the functions inside the CUSPARC library, one can judge whether the bug is in the C code itself or in the processor design.

2) *Real time debugger:*

A real time debugger was also developed for validating the CUSPARC processor. This debugger verifies the code while it is running on the processor after burning it on the FPGA.

TABLE I
CUSPARC REPORT ON STRATIX-III FPGA

FPGA device used	Stratix-III (EP3SL150H1152C2)
Combinational ALUTs*	6,763/113,600 (6%)
Dedicated logic registers	5,901/113,600 (5%)
Total memory block bits	79,872/5,630,976 (1%)
Max. Frequency	135MHz

*ALUT: Adaptive Look-Up Table

The UARTs available on the processor are used for debugging in this case. Simply the user selects the variables to watch from within the IDE. The IDE modifies the C code so as to make the processor transmit the values of these variables via the UART and then hang up waiting for a command from the user to step one line or run to a certain breakpoint in the code. This operation continues until the program is finished. Obviously, the main advantage of the real time debugger over the simulation debugger is speed.

IV. TESTING AND APPLICATIONS

We have gone through several phases during the verification and testing of the processor. These phases included test benching the individual building blocks of the processor, simulating the processor while running short sequences of instructions and finally testing the processor while running complex real-life programs. The afore-mentioned simulation and real time debuggers were extensively used during these validation phases.

CUSPARC processor was implemented on different Altera FPGA families (Flex10KE, Apex20KC, Cyclone, Cyclone II, Stratix-II, and Stratix-III.

The CUSPARC processor was used to implement several embedded applications on FPGA platforms. These applications include the PHY layer of a Wi-Fi transceiver and a Bluetooth transceiver. Custom Hardware accelerators are occasionally attached to CUSPARC processor to speed up computationally intensive functions like FFT and Viterbi decoding. The basic data rates of 1Mb/s and 6Mb/s for the Bluetooth and Wi-Fi transceivers respectively were achieved on the STRATIX III FPGA platform. These applications culminated in a Software Defined Radio (SDR) application. This SDR was implemented using several CUSPARCs connected as a processor pipeline on Altera's STRATIX III FPGA DSP kit. This generic hardware platform can be software configured to run the Bluetooth 2 or Wi-Fi protocols. Three CUSPARC processors were used to implement the transmitter and another three processors were used to implement the receiver.

V. PERFORMANCE & METRICS

A. *FPGA design*

As mentioned in section IV, CUSPARC was implemented on many Altera FPGA kits. Table (I) shows the result of CUSPARC on Altera Stratix-III EP3SL150H1152C2 FPGA. Combinational logic occupies around 6% of ALUTs (Adaptive Look-Up Table) of this chip. More than 5% of available registers is used, mainly in the register file. Instruction and

data caches are implemented using the memory blocks in the FPGA, consuming around 1% of the available memory. CUSPARC operates at maximum frequency of 135MHz on this FPGA.

B. *Dhrystone v.2 benchmark*

The Dhrystone metric [4] measures the performance of processors more meaningfully than MIPS metric, because exact instruction count comparisons between different instruction sets (e.g. RISC vs. CISC) are not meaningful. The Dhrystone metric is widely used to measure the performance of embedded processors. The common representation of the Dhrystone benchmark is the DMIPS (Dhrystone MIPS), obtained by dividing the Dhrystone score by 1757 (the number of Dhrystones per second obtained on the VAX 11/780, defined as the unity MIPS machine). CUSPARC DMIPS is shown in Table (II).

TABLE II
CUSPARC DHRYSTONE SCORES, 135 MHz ON STRATIX-III FPGA

Dhrystones per second	229202.04
DMIPS / MHz	0.9663

CUSPARC performs about 230,000 Dhrystone iterations per second at 135MHz on Stratix III EP3SL150H1152C2 FPGA platform. Thus, CUSPARC scores 0.9663 DMIPS/MHz. A comparable ARM processor (ARM7TDMI), scores 0.9 DMIPS/MHz [5].

TABLE III
DHRYSTONE BENCHMARK RESULTS COMPARISON BETWEEN CUSPARC, LEON2, & MICROBLAZE PROCESSORS OPERATING ON 30MHz [5]

	CUSPARC	**LEON2**	**MicroBlaze**
Time for a Dhrystone iteration (us)	19.63	22.5	32.7
Dhrystone iterations/second	50933.8	44444.4	30611.4
Dhrystone iterations/second/MHz	1697.79	1481.48	1020.38
DMIPS/MHz	0.9663	0.84319	0.58075

Another comparison is shown in Table (III) between CUSPARC, LEON2 and MicroBlaze processors [6]. All processors operate on the same 30MHz frequency. The results show that CUSPARC is the best, followed by LEON2 then MicroBlaze.

C. *ASIC design*

CUSPARC was modified to target ASIC design to probe its potential for embedded SoC applications. These minor modifications are mainly in the cache memory blocks. Two different technologies were used, AMS (Austrian Micro Systems) 0.35µm, and IBM 0.13µm technologies. Table (IV) shows a comparison between the two designs.

TABLE IV
ASIC DESIGNS COMPARISON

	AMS 0.35µm	**IBM 0.13µm**
Chip Area	9 mm^2	1.96 mm^2
I & D Caches size	1 kB each	4 kB each
Number of Routing layers	4	8
Frequency	180 MHz	260 MHz

Areas indicated includes cache memory blocks area. Also, sizing down caches in 0.35µm design was due to limited design area. Typical post-layout simulations indicate that the current ASIC implementation of CUSPARC at the 0.13µm node achieves a maximum operating frequency of 260 MHz.

VI. CONCLUSION

A working IP processor core conforming to the SPARC v8 ISA is built, with the ability to interface custom hardware. We code named this processor CUSPARC for Cairo University SPARC. CUSPARC is a 32 bit processor with 4-stage pipelining. Data and control hazard prevention is implemented, traps and interrupts are supported. It has 2 UARTs, 3 timers, watch dog timer and a standard WISHBONE bus interface. The design is ported to several FPGA families.

A complete software suite supporting this processor was developed including a novel real time and off-line Debugger. Three SoC applications were implemented and verified using CUSPARC processor. Also, ASIC designs were targeted on 0.35µm & 0.13µm technologies. A maximum operating frequency of 135 MHz was achieved on Stratix-III FPGA kit and 260MHz for the ASIC design on 0.13µm technology. CUSPARC has scored 0.9663 DMIPS/MHz.

ACKNOWLEDGEMENT

Many people contributed to four earlier versions of CUSPARC. The following list acknowledges these contributors: Mohamed A. Khairy, Hesham W. H. Sabry, Karim A. Hosny, Khalid H. M. Khallaf, Ramy R. A. S. Eissa, Ahmed A. M. Bakr, Amgad K. AbdElSalam, Mahmoud M. AbdAllah, Ahmed H. Mostafa, Ahmed H. Fathy, Ahmed A. Ayoub, Ahmed A. Abdelwahab, Ahmed M. Abdelgaber, Hesham M. Mahfouz, Mohamed K. AbdElFattah, Hesham M. El-Sayed, Mohamed M. Farag, Amr S. Abu-Bakr and Karim A. Tarek.

The authors also acknowledge a MOSIS MEP grant # 4960 granting access to the technology files and standard cell library of IBM 0.13µm CMOS 8RF-DM technology. They also acknowledge AMS foundry for granting access to HitKit package of their AMS 0.35 µm technology.

REFERENCES

[1] The SPARC International Inc. *The SPARC Architecture Manual*, version 8. Available at http://www.sparc.com/standards/V8.pdf
[2] Wishbone bus specification, "WISHBONE system-on-Chip (SoC) Interconnection Architecture for Portable IP Cores". Available at http://opencores.org/opencores,wishbone.
[3] GCC Web site, http://gcc.gnu.org/
[4] Alan R. Weiss, "Dhrystone Benchmark White Paper", Available at http://www.eembc.org/techlit/whitepaper.php#dhrystone, November 2002.
[5] ARM processors, http://www.arm.com/products/processors/classic/
[6] Daniel Mattson & Marcus Christensson, "Evaluation of synthesizable CPU cores", Chalmers University of Technology, Gothenburg 2004.