

# AN AMBA AHB-BASED RECONFIGURABLE SOC ARCHITECTURE USING MULTIPLICITY OF DEDICATED FLYBY DMA BLOCKS

Adeoye Olugbon<sup>1</sup>, Sami Khawam<sup>1</sup>, Tughrul Arslan<sup>1,2</sup>, Ioannis Nousias<sup>1</sup>, Iain Lindsay<sup>1</sup>

<sup>1</sup>School of Engineering and Electronics, The University of Edinburgh, Scotland

<sup>2</sup>Institute for System-Level Integration, Livingston, Scotland

[A.Olugbon@ed.ac.uk](mailto:A.Olugbon@ed.ac.uk)

**Abstract** – We propose a System-on-Chip (SoC) architecture for reconfigurable applications based on the AMBA High-Speed Bus (AHB). The architecture features multiple low-area flyby DMA blocks for transferring configuration data. Furthermore, the architecture eliminates the use of energy-consuming instructions used in comparable commercial reconfigurable SoCs. The flyby DMA blocks achieve a reduction of up to 98% in the number of gates found in general-purpose DMA controllers. The DMA blocks also achieve the flyby throughput which halves the number of clock cycles used in conventional DMA for data transfer. We also demonstrate the presence of parallel processing which contributes to improved system performance of the proposed architecture over commercial comparatives.

## I. Introduction

Reconfigurable Systems-on-Chip (SoC) combine microprocessors or microcontrollers as well as embedded FPGAs and other reconfigurable logic on the same chip. Such devices offer developers the opportunity to define the environment around the microprocessor/microcontroller. Custom applications implemented in the FPGA/reconfigurable logic may be altered by reloading a new device configuration represented as bitstream data, usually stored in serial EPROM, flash memory or any other serial programming device and loaded to the FPGA upon power up. SoC devices such as those in the Xilinx Virtex family and the Altera Excalibur can also be partially reconfigured by programs running on the microprocessor by transferring configuration data to a specific section of the FPGA while the rest of the device continues working. A popular technique of doing this as featured in commercial SoCs [1] and [2] involves transferring configuration bits, one word at a time, by the processor over an on-chip bus to the FPGA. This technique is an inefficient use of system resources and consumes a lot of energy.

We present a reconfigurable SoC architecture that uses a novel dedicated flyby DMA technique to address the inefficiency and limitation present in the techniques currently employed in commercial SoCs to reconfigure FPGAs. We demonstrate that the proposed architecture is resource-efficient by enabling the processor and reconfigurable logic perform concurrent activities during reconfiguration. We also show that the proposed architecture eliminates the iterative execution of energy-consuming instructions [3,4,5,6,7]. Finally, compared with

conventional DMA controller cores, our dedicated flyby DMA logic has area advantages that would allow us easily scale the reconfigurable SoC architecture to a multi-layered bus architecture SoC.

The rest of this paper is organized as follows: Section II summarises the key features of the proposed architecture and two commercial comparatives, section III discusses the interfacing of FPGAs or reconfigurable logic with microprocessors, section IV describes reconfiguration of FPGAs under microprocessor control, section V discusses some performance issues related to reconfigurable SoCs, section VI summarises our results, section VII highlights an important scalability advantage and the final section contains a summary and conclusions.

## II. Reconfigurable SoCs

Product documentation [1] and [2] present examples of commercial chips that combine FPGAs and microprocessors on the same chip.

The proposed AMBA AHB based SoC architecture is based on the coarse-grained synthesisable reconfigurable arrays presented in [8]. Like the Excalibur and the Virtex-II, our proposed architecture would support reconfiguration under host processor/microcontroller control. In terms of pipeline depth, operating frequency, on-chip bus and process geometry of the host processor, our proposed architecture based on the LEON-2 processor is comparable to the Xilinx Virtex-II and Altera Excalibur devices.

## III. Reconfigurable Array Interface

The easiest and most popular way to interface reconfigurable logic to a microprocessor in a bus-based SoC is memory-mapped interfacing. Writing data to the array would be achieved using a STORE instruction whose operands are the word to be written and a valid memory-mapped address corresponding to the peripheral. In ARM assembly, this could be written as

```
STR R0, [R1]
```

Which would load the word in R0 to the effective address pointed to by R1. R1 would have been set up to hold the memory-mapped address corresponding to the

reconfigurable array. Virtually all processors/microcontroller support this register indirect addressing scheme. In PowerPC assembly language, the same transaction would be carried out using

```
stw rs, (rA)
```

Where *rs* holds the word to be transferred and *rA* points to the memory-mapped address of the peripheral.

#### IV. On-chip bus interfacing and Reconfiguration under host processor control

The Altera Excalibur and Xilinx Virtex-II devices are reconfigurable by user programs running on the host. Product documentation [1] and [2] contain flowcharts of user programs for configuring the Virtex-II and Excalibur devices. These programs both involve using a loop to iteratively load words from anywhere in memory into the FPGA. One word is written at a time until the FPGA is fully configured. Documentation from Xilinx [9] points out that parallel configuration modes of Xilinx FPGAs using a microprocessor are not faster than the bit-serial mode except in Spartan XL and XC4000XLA devices.

To reinforce our proposal, we implemented two AHB wrappers for interfacing reconfigurable arrays to the AHB. One of the wrappers for interfacing the reconfigurable arrays would permit reconfiguration in the fashion provided by the Xilinx Virtex-II and Altera Excalibur. The other includes the novel dedicated flyby DMA logic. Descriptions follow

##### A AHB slave wrapper

In implementing this wrapper, the reconfigurable array is considered a slave on the AHB. All transactions between the host processor and the reconfigurable array are initiated by the host. The reconfigurable array is programmed through bitstream loading of the bitstream via a shift chain as performed by commercial FPGAs.

##### B AHB Peripheral with flyby DMA logic

This involves the design of a wrapper with AHB master and AHB slave logic. The host transfers two words to the slave logic. The first word is the size of DMA transfer (in words) to be performed by the peripheral and the second is the starting address in memory from which the transfer should begin. The AHB slave logic then signals to the AHB master logic to perform the transfer from the address supplied by the host. This transfer would be a series of READ operations to retrieve configuration bitstream from memory. A single word transfer is completed in 2 clock cycles on the AHB. However, using conventional DMA technique, a READ operation would be completed in 4 clock cycles since the DMA would have to perform a READ (from the source) in two clock cycles and perform a WRITE (to the destination)

in another 2 clock cycles. Our DMA technique achieves the high-speed throughput of flyby DMA.

Both wrappers were designed using Verilog HDL and implemented on UMC 0.18 micron CMOS technology. Table I shows performance results. The power results were obtained during configuration activity in which the wrapper received configuration data (or retrieved them by DMA transfer) on the AHB and configured instances of the reconfigurable array with the configuration data.

TABLE I

Performance comparatives of two implemented AHB wrappers

Interface	Area	Power
AHB Slave wrapper	1	1
AHB wrapper with flyby DMA logic	1.38	1.53

#### V. Performance

##### A. Energy

Tiwari, Malik and Wolfe [10] formulated a hypothesis for evaluating power/energy cost of a program running on a processor.

The ideas proposed in [10] have been verified. Nikolaidis [3,4,5] developed instrumentation that validated the model proposed in [10]. The energy cost of a program running on an ARM processor was evaluated by adding up the estimated energy cost of all the instructions in the program. The actual measured results were within 0.43% of the estimated value.

In [1] and [2] a STORE instruction is executed iteratively in a loop to load configuration bitstream. The results of the work in [6] and [7] showed that LOAD and STORE instructions consume significantly more energy than the other instructions. Results in [6] reported that up to twice as much energy consumed was by LOAD and STORE instructions than the other instructions for a simulation system that is a 5-stage pipeline MIPS-II compatible RISC implemented in 0.25u CMOS technology. Results in [7] reported energy costs of up to 170mA for STORE instructions on a 5-stage pipeline ARM8 prototype.

##### B Parallelism

Parallelism is considered to be a key performance driver in computing and embedded systems. Portions of [11] present discussions on parallelism that covers lookahead, pipelining, vectorization, concurrency and many other features at different processing levels. In our proposed architecture, the host CPU is free to execute instructions in other active execution threads while the flyby DMA block retrieves configuration bitstream from on-chip memory thereby achieving parallelism through concurrency.

## VI. Results

From Table II we see that the additional flyby DMA logic leads to nearly 40% area overhead. In terms of gate count, it is a mere additional 1,268 extra gates. Therefore we propose an architecture in which several instances of reconfigurable arrays on the SoC have AHB wrappers that include flyby DMA logic as shown in Figure I. Each of these arrays would be capable of retrieving configuration bitstream using a fast flyby technique that is twice as fast as conventional DMA.

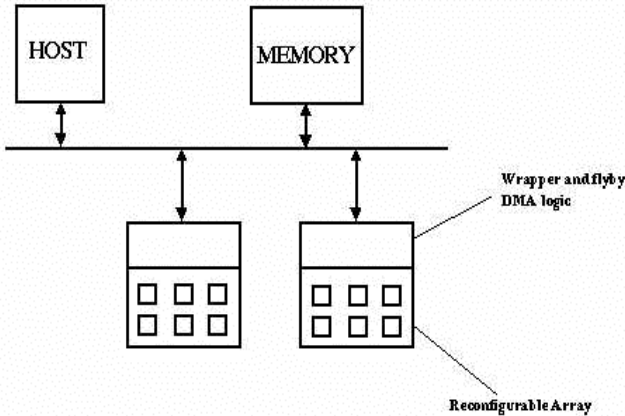


Figure 1: Our proposed architecture in which each instance of a reconfigurable array is interfaced to the bus using a wrapper that includes flyby DMA logic

Table II shows comparative figures of the gate count in some commercial dedicated DMA controller cores. From this table we can easily work out that our proposed architecture would still prove area-competitive in an implementation with up to four instances of reconfigurable arrays each having a wrapper with its own flyby DMA logic.

TABLE II

A comparison of three DMA controller cores and the flyby DMA logic present in our wrapper

	Approx gate count	Supports Flyby	Target technology
Our DMA logic	1268	Yes	UMC 0.18
ARM PL080 DMA controller [17]	82000	No	Avant 0.25
ARM PL081 DMA controller [18]	24900	No	Avant 0.25
Cast C8237 DMA controller [19]	5500	No	UMC 0.18

From an energy perspective, previous work on instruction-level power analysis for embedded systems [3,4,5,6,7] justify our use of DMA for retrieving configuration bitstream. We do not have figures from instruction-level power analysis of the IBM PowerPC 405 and ARM 922T cores. However, according to [12], the average power/instruction for a StrongArm processor running at

170MHz is 200mW. Load/Store instructions consume 260mW. This figure exhibits correlation with previous work demonstrating that LOAD/STORE instructions typically consume more power than other instructions. Unpublished [14] presents some results of instruction power profile for the PowerPC 603, but the work excludes data for PowerPC LDW and STW instructions.

Exact figures are not available, but our architecture eliminates one component of embedded software power consumption by freeing the host processor of the transfer task described in [1] and [2]. The discussion that follows outlines the energy components.

### A Load/Store power component in program

The value of this component would depend on the power consumed by the STORE instruction used to transfer a word as described in [1] and [2]. It would also depend on the number of iterations performed by the loop. If the FPGA/reconfigurable array is used for implementing hardware for computationally-intensive applications, then this power component would be significant. Ref [15] demonstrated, using 8 real-life applications, that there were 2523 code regions that were candidates for speedup on dedicated hardware. Assume, for the sake of analysis, that a particular program contains 8 such regions (very moderate estimate) and each of those regions require an average of 1024 cycles to transfer a moderate 4KB of configuration data (a very moderate estimate), the estimated energy of the code segment required to transfer the configuration would be approximately 8000 times the energy cost of executing a single STORE instruction. If we use figures available for the StrongARM core, the energy cost for the store instruction is  $\text{Energy Cost} = \text{current} * \text{cycles} * \text{voltage/frequency}$

Assuming a CPI of 1, this works out to be 1.53 nJ since instruction level analysis for a 170 MHz StrongARM core reports that the load/store instructions draw 260 mW on average.

1024 cycles/per code region and 8 code regions in our assumption imply 8192 cycles (or 48 microseconds for a 170 MHz processor), and a total energy cost of 12.5 uJ for the code segment driving the WRITE signals on the bus.

### B On-chip bus component

Our proposed architecture and the Altera Excalibur feature the AMBA standard which is very similar to the CoreConnect present in the Xilinx Virtex-II. Ref [16] reported figures of instruction energy analysis on the AMBA bus using a simulation system clocked at 100MHz. READ, WRITE and IDLE activity modes were evaluated for power consumption over a simulation period of 50us the total simulation energy reported was 839.6 uJ of which READ and WRITE activities accounted for 733uJ.

Since flyby DMA halves the number of AMBA AHB read and write cycles required for direct memory access transactions. Our proposed technique leads to significant power consumption reduction on the bus when compared with conventional DMA techniques.

### C Reconfigurable Array/FPGA component

The work in [8] showed that the reconfigurable array in our proposed architectures achieves 75% power reduction in comparison with commercial generic reconfigurable logic (Xilinx Virtex-E device)

## VII. Scalability

One of the attributes of a scalable system is the presence of features that succeeding generations of that system can inherit. Our dedicated DMA controller has competitive area advantages because it is not loaded with functionality to support general-purpose applications. With this small area overhead, the proposed architecture as shown in Fig. 1 can be scaled to a multi-layer bus architecture. In a multi-layer bus architecture, each array as well as the wrapper can be easily instantiated without severe area penalties.

## VIII. Summary and Conclusions

The reconfigurable SoC architecture proposed in this paper contains features that allow more efficient data transfer in comparison with commercial versions. It employs DMA for retrieving configuration data. However, unlike conventional generic DMA cores, we implemented dedicated flyby DMA logic within the bus wrapper for the reconfigurable logic. These dedicated flyby DMA blocks have low area. In comparison with three synthesizable general-purpose DMA controllers, our dedicated flyby logic achieves up to 98% reduction in the number of gates used in implementation. High flyby throughput of twice the data rate of conventional DMA also contributes to better overall system performance. Therefore several instances of reconfigurable arrays in the SoC can each include a dedicated flyby DMA block to throttle configuration data

### References

- [1] Reconfiguring Excalibur Devices Under Processor Control: Application note 298, Feb 2003, ver 1.0. <http://www.altera.com>
- [2] Using a Processor to Configure Xilinx FPGAs via Slave Serial or SelectMap mode: XAPP502 (v1.4), Nov 2003. <http://www.xilinx.com>
- [3] T. Laopoulos, P. Neofotistos, C.A. Kosmatopoulos, S. Nikolaidis, "Measurement of current variations for the estimation of software-related power consumption", *Instrumentation and Measurement, IEEE Transactions on*, Vol. 52, pp. 1206 – 1212, 2003
- [4] S. Nikolaidis, N. Kavvadias, P. Neofotistos, C. A. Kosmatopoulos, T. Laopoulos, L. Bisdounis, "Instrumentation Set-up for Instruction Level Power Modeling", *Energy-Aware System-on-Chip (EASY) Project*: <http://www.easy.intranet.gr>
- [5] S. Nikolaidis, "Instruction-level energy characterization of an ARM processor", *Energy-Aware System-on-Chip (EASY) Project*: <http://www.easy.intranet.gr>
- [6] R. Krashinshky, "Microprocessor Energy Characterization and Optimization through Fast, Accurate and Flexible Simulation", *Master of Science Thesis, Massachusetts Institute of Technology*, 1999.
- [7] P. Laramie, "Instruction Level Power Analysis and Low Power Design Methodology of a Microprocessor", *MS Thesis, University of California at Berkeley*, 1998
- [8] S. Khawam, T. Arslan, F. Westall, "Embedded reconfigurable array targeting motion estimation applications", *Circuits and Systems, Proceedings of the 2003 International Symposium on*, Vol. 2, pp. II-760 - II-763, 2003
- [9] Xilinx FPGAs: A Technical Overview for the First-Time User. APPLICATION NOTE. XAPP 097 December 12, 1998 (Version 1.3). <http://www.xilinx.com>
- [10] V. Tiwari, S. Malik, A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization", *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, Vol. 2, pp. 437 – 445, 1994
- [11] K Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw Hill International Editions, 1993.
- [12] G. De Micheli, "System Level Power Optimization: Algorithms and Software. Stanford University lecture notes", unpublished.
- [13] LEON2: Gaisler Research; <http://www.gaisler.com>
- [14] O. A. Patino, M. Jimenez, "Instruction Level Power Profile for the PowerPC Microprocessor", unpublished
- [15] A. Jantsch, P. Ellervee, J. Oberg, A. Hemani, "A Case Study on Hardware/Software Partitioning", *FPGAs for Custom Computing Machines, 1994. Proceedings. IEEE Workshop on*, pp. 111-118, 1994
- [16] M. Caldari, M. Conti, M. Coppola, P. Crippa, S. Orcioni, L. Peralisi, C. Turchetti, "System-level power analysis methodology applied to the AMBA AHB bus [SoC applications]" *Design, Automation and Test in Europe Conference and Exhibition, 2003* pp. 32 - 37 suppl. ,2003
- [17] PrimeCell DMA Controller (PL080) Technical Reference Manual <http://www.arm.com>
- [18] PrimeCell DMA Controller (PL081) Technical Reference Manual, <http://www.arm.com>
- [19] C8237 Programmable DMA Controller Core, <http://www.cast-inc.com>.