# MSL16A: An Asynchronous Forth Microprocessor

P.K. Tsang, C.C. Cheung, K.H. Leung, T.K Lee, P.H.W. Leong
Department of Computer Science and Engineering
The Chinese University of Hong Kong
{pktsang, ccheung2, khleung, tonylee, phwl}@cse.cuhk.edu.hk

**Abstract**

*A 16-bit Forth microprocessor, MSL16 [7], was developed for embedded applications as the design offers good code density, easily developed software development tools, high performance, and small area. A delay-insensitive re-implementation of the processor has been developed to explore the potentials of asynchronous logic for low-power applications and to demonstrate the feasibility and practicability of using asynchronous circuits in embedded applications. This paper describes the asynchronous clone of MSL16. The implementation and performance evaluation of it is also presented. The design will be fabricated using AMI 1.2$\mu$ CMOS double layer metal process in 3Q99.*

## 1 Introduction

Many embedded applications employ a coprocessor today and a microprocessor which has desirable features like good performance, high code density, small area, and good development tools is eminently suited for such applications. We have developed an architecture to directly execute the Forth language called MSL16 [7] (which stands for minimal instruction, small and low power 16-bit microprocessor) which was optimized for embedded applications. It utilizes a stack architecture with each instruction occupying only 4 bits, leading to a small instruction set, simple datapath and control, and high code density.

Forth is a (stack based) portable integrated programming environment, operating system and programming language having code density typically higher than that of C or assembly language, and is well suited to DSP, real-time and embedded applications. A Forth system is typically built upon a small number of primitives, and the higher level routines call the lower level primitives to implement the rest of the system. The system (which bundles the operating system and compiler) is very simple and can be ported in a matter of several weeks, compared to man-years of development effort for a reasonable C compiler.

Recent research has demonstrated that asynchronous circuits techniques have matured and implementations of asynchronous processors have been reported [9, 15, 10, 12, 1, 11, 2, 14]. The asynchronous re-implementation of the microprocessor, called MSL16A , has been developed to investigate the potential advantages that asynchronous designs may enjoy, namely average-case performance instead of worst-case performance and low power consumption. MSL16A was also developed to demonstrate the feasibility and practicability of using asynchronous circuits in embedded applications.

The paper is organized as follows. Section 2 gives a brief description of the architecture of MSL16A. The design methodology and circuit style used in designing MSL16A, which were an original application of Alain Martin's method, and its implemenatation are described in Section 3. Evaluation results including the chip layout are presented in Section 4. Finally, section 5 concludes the paper.

## 2 Architecture

MSL16 is a dual stack machine with 16 bit data and memory buses. The data stack is used for variable storage and parameter passing while the return stack is used to hold return addresses. The data and return stacks are implemented internally which allows them to be accessed in parallel with instruction fetches on the memory bus. A two stage FETCH/EXECUTE pipeline is employed. The execution speed of MSL16A is high because of its simple instruction set and a short critical delay path.

All instructions are encoded with 4 bits except the CALL instruction. A single instruction fetch generally obtains 4 instructions which will reduce the effect of pipeline starvation on system performance. Hence, the use of a slower memory will not have a significant impact on performance. The data stack and return stack are both $32 \times 16$ bits in size. A study has shown that a stack depth of 32 is sufficient for most reasonable programs [5]. The instruction format used for

MSL16 is shown in Figure 1.

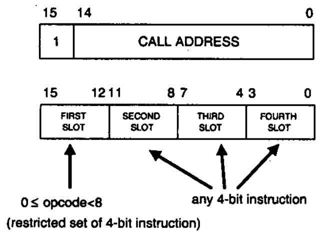

Figure 1: Instruction format for MSL16

The instruction set of MSL16 is given in Table 1. If the most significant bit of the instruction register(IR) is set, it is a CALL instruction and the remaining 15 bits form the subroutine address. Moreover, if LIT appears in the first or second slot of IR, its operand will be the least significant byte of IR. Hence, to load a 16-bit literal into the T register, two successive LITs followed by an XOR are required.

The datapath of MSL16A, which is modified from its synchronous counterpart, is shown in Figure 2.

| Opcode | Instruction | Action |
|---|---|---|
| 0 | NOP | no operation |
| 1 | AND | T ⇐ T AND DS, pop DS |
| 2 | XOR | T ⇐ T XOR DS, pop DS |
| 3 | + | T ⇐ T + DS, pop DS |
| 4 | 0= | T ⇐ -1 if (T=0) else T ⇐ 0 |
| 5 | LIT | push T to DS and, if LPC = 0, T⇐LSB(IR)&"00000000" if LPC = 1, T⇐"00000000"&LSB(IR) if LPC = 3, T⇐processor status word |
| 6 | 2/ | T ⇐ T / 2 |
| 7 | - | T ⇐ DS − T, pop DS |
| 8 | DUP | push T to DS |
| 9 | DROP | pop DS to T |
| 10 | GOTO | Jump to T if $T \neq 0$, pop DS to T |
| 11 | R> | push T to DS, pop RS to T |
| 12 | >R | push T to RS, pop DS to T |
| 13 | @ | LOAD mem[T] to T |
| 14 | ! | STORE T to mem[ds] |
| 15 | SWAP | Swap T with DS |
| MSB=1 | CALL | PUSH PC to RS, jump to IR |

Table 1: The MSL16 instruction set

## 3 Design Methodology and Implementation

The asynchronous circuits we use are called *quasi delay-insensitive*(QDI) circuits which do not use any assumption on delays in operators and wires [8]. The asynchronous control logic was designed by using CAST[1] while the datapath components were realized

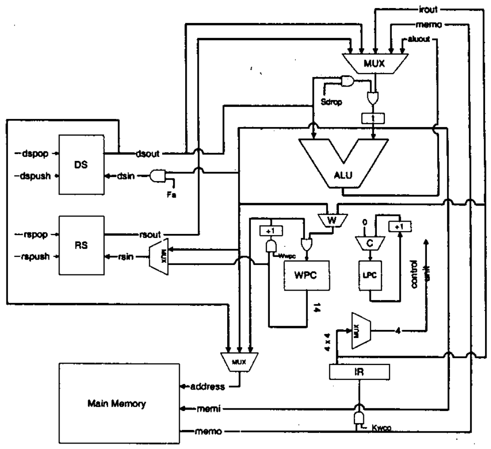

Figure 2: The datapath of MSL16A

by hand.

The processor is implemented based on Martin's Synthesis Method [8]. It is first described by a high level sequential program which is a non-terminating loop containing the FETCH and EXECUTE stages of the pipeline. The sequential program is then decomposed into a set of concurrent processes, which communicate and synchronize with each other, based on Hoare's CSP (Communicating Sequential Processes) model [4]. This method relies on the time-honored "divide-and-conquer" principle. All datapath elements are accompanied by a small control circuit, obeying the four-phase (return-to-zero) protocol, for synchronization of *request* and *acknowledge* signals.

Data are all dual-rail encoded within the processor core. Only one out of two rails is raised at each active phase of the four-phase protocol. All control circuits are generated and verified with CAST, a set of in-house tools for synthesis of asynchronous circuits by Caltech, while all other elements are created by hand with Magic and logically verified by IRSIM.

As data are dual-rail encoded within the processor core, NAND gates (as shown in Figure 2) can be used , instead of using tri-state buffers, to save area along all 16-bit data buses. Moreover, Dual-rail to single-rail and single-rail to dual-rail converters [12] are used to interface with the outside world with bundled data encoding to allocate extra pins for testing. Figure 3 shows the gate-level descriptions of the converters.

### 3.1 ALU

A simple delay-insensitive ALU which delivers comparable performance to more sophisiticated syn-

[1]Caltech Asynchronous Synthesis Tools

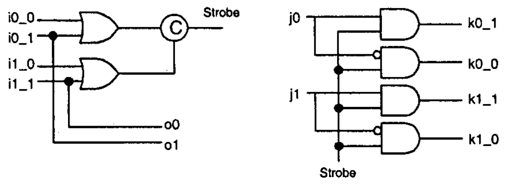

Figure 3: Dual-rail to single-rail and single-rail to dual-rail converters

chronous counterparts was designed. The asynchronous nature of the unit takes advantages of best- and average-case performance while allowing worst case operations to take longer time to complete, thus giving a higher average throughput. The set of functions provided by the ALU consists of basic logic operations, arithmetic shift, addition and subtraction. A Quick-Decision Zero-Checker based on [6] was implemented for the condition test of a conditional branch.

Addition, or subtraction, is the most time consuming function as all logical operations are performed in a bitwise fashion while worst case addition may require communications across all 16 full adders. However, a study by Manchester University [3] suggests that the mean carry propagation length is about 4.4 bits for 32-bit operands. As long carry chains are relatively rare, our adder has no special fast logic and performs addition with a chain of 16 full adders to deliver "typical" performance at a smaller size.

### 3.2 Stack

The stack is pointer based and the pointer is implemented as an internal bit variable within the stack control circuit. Only the top element will be active while others just idly waiting for their neighbours to pass the pointer to them. The top element will send a *request* signal (pointer passing) to the next element, asking it to become active (top of stack) and wait for its *acknowledge* signal, after a stack operation. All other stack elements are not involved in any communication and stay completely idle, conserving power.

## 4 Results

Each sub-system has been extensively simulated with HSPICE and performance estimates for the ALU and the stack are shown in Table 2 and Table 3. In all cases, these results come from a 5V power supply. All PFETs are $8\lambda \times 2\lambda$ and NFETs are $4\lambda \times 2\lambda$ except some of them are sized for better performance. All measurements are based on HSPICE(98.2) on a AMI $1.2\mu$ CMOS double layer metal process, using MOSIS parametric test results of run N81Y. The chip will be fabricated in 3Q99. The longest carry chain in a "typical" addition is assumed to be 4 bits. Similar work on asynchronous ALU can be found in [3].

| operation | Delay(ns) | Power(mW) | Power-Delay-Product(pJ) |
|---|---|---|---|
| XOR | 2.596 | 45.741 | 118.741 |
| AND | 2.605 | 43.341 | 112.903 |
| 2/ | 3.130 | 50.358 | 157.621 |
| 0= | 7.019 | 48.078 | 337.459 |
| +(typ.) | 17.746 | 24.543 | 435.540 |
| +(worst) | 42.343 | 14.995 | 634.888 |

Table 2: Performance of the ALU

The performance of the stack is critical as almost all instructions push/pop data to/from the stack. The speed, power consumption and Power-Delay product of the stack are listed in Table 3. We expect the average processing rate to be 33 MIPS, because on average the EXECUTE stage delay is less than 30ns (assuming the memory is fast enough). As stack and ALU operations are the most data-intensive operations in the processor, the estimated power consumption is below 200mW at full speed.

| | Push | Pop |
|---|---|---|
| Delay(ns) | 11.8377 | 6.205 |
| Power(mW) | 19.24 | 12.38 |
| Power-Delay Product(pJ) | 227.747 | 76.793 |

Table 3: Performance of the stack

The chip layout is shown in Figure 4. All processor components (including bonding pads) are integrated in 4.335mm × 4.671mm (20.249mm$^2$) which is much smaller than both the TITAC-2 chip [14] and the ASPRO-216 [13] if both of them are scaled to the same technology. The TITAC-2 chip was a 32-bit microprocessor which was fabricated using $0.5\mu$ CMOS standard cell technology and it occupied 12.15mm×12.15mm. Similarly, ASPRO-216 was a QDI 16-bit RISC microprocessor targeted on a $0.25\mu$ five layer metal CMOS technology and it occupied about 4mm$^2$.

## 5 Conclusion

MSL16 was targeted for embedded applications as it offers good code density, high performance at a small area. This paper presented an asynchronous re-implementation of MSL16, called MSL16A which is a QDI Forth microprocessor developed based on Martin's Synthesis Method [8]. The estimated performance, power consumption and chip area show that

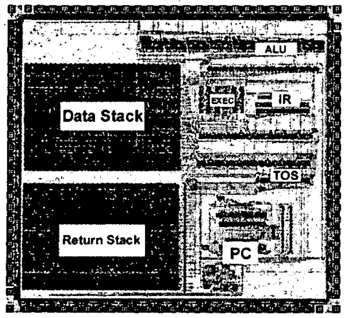

Figure 4: MSL16A chip image

this re-implementation is promising. Furthermore, the synchronous implementation (on silicon) will also be realized to compare the benefits of synchronous and asynchronous design methodology in detail.

## Acknowledgements

The authors are greatful to Alain Martin and his group for proving use of the Caltech Asynchronous Synthesis Tools.

## References

[1] S. B. Furber, P. Day, J. D. Garside, N. C. Paver, and J. V. Woods. AMULET1: A micropipelined ARM. In *Proceedings IEEE Computer Conference (COMPCON)*, pages 476–485, March 1994.

[2] S. B. Furber, J. D. Garside, S. Temple, J. Liu, P. Day, and N. C. Paver. AMULET2e: An asynchronous embedded controller. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 290–299. IEEE Computer Society Press, April 1997.

[3] Jim D. Garside. A CMOS VLSI implementation of an asynchronous ALU. In S. Furber and M. Edwards, editors, *Asynchronous Design Methodologies*, volume A-28 of *IFIP Transactions*, pages 181–207. Elsevier Science Publishers, 1993.

[4] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, August 1978.

[5] Philip Koopman. Why stack machines? *Computer Architecture News*, 21(1), March 1993.

[6] Tak Kwan Lee. *A General Approach to Performance Analysis and Optimization of Asynchronous Circuits.* PhD thesis, California Institute of Technology, 1995. Technical report CS-TR-95-07.

[7] P. H. W. Leong, P. K. Tsang, and T. K. Lee. A FPGA based Forth microprocessor. In Kenneth L. Pocek and Jeffrey Arnold, editors, *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 254–255, Los Alamitos, CA, April 1998. IEEE Computer Society Press.

[8] Alain J. Martin. Programming in VLSI: From communicating processes to delay-insensitive circuits. In C. A. R. Hoare, editor, *Developments in Concurrency and Communication*, UT Year of Programming Series, pages 1–64. Addison-Wesley, 1990.

[9] Alain J. Martin, Steven M. Burns, T. K. Lee, Drazen Borkovic, and Pieter J. Hazewindus. The design of an asynchronous microprocessor. In Charles L. Seitz, editor, *Advanced Research in VLSI*, pages 351–373. MIT Press, 1989.

[10] Alain J. Martin, Andrew Lines, Rajit Manohar, Mika Nystroem, Paul Penzes, Robert Southworth, and Uri Cummings. The design of an asynchronous MIPS R3000 microprocessor. In *Advanced Research in VLSI*, pages 164–181, September 1997.

[11] Shannon V. Morton, Sam S. Appleton, and Michael J. Liebelt. ECSTAC: A fast asynchronous microprocessor. In *Asynchronous Design Methodologies*, pages 180–189. IEEE Computer Society Press, May 1995.

[12] Takashi Nanya, Yoichiro Ueno, Hiroto Kagotani, Masashi Kuwako, and Akihiro Takamura. TITAC: Design of a quasi-delay-insensitive microprocessor. *IEEE Design & Test of Computers*, 11(2):50–63, 1994.

[13] M. Renaudin, P. Vivet, and F. Robin. ASPRO-216: A standard-cell QDI 16-bit RISC asynchronous microprocessor. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 22–31, 1998.

[14] Akihiro Takamura, Masashi Kuwako, Masashi Imai, Taro Fujii, Motokazu Ozawa, Izumi Fukasaku, Yoichiro Ueno, and Takashi Nanya. TITAC-2: An asynchronous 32-bit microprocessor based on scalable-delay-insensitive model. In *Proc. International Conf. Computer Design (ICCD)*, pages 288–294, October 1997.

[15] José A. Tierno, Alain J. Martin, Drazen Borkovic, and Tak Kwan Lee. A 100-MIPS GaAs asynchronous microprocessor. *IEEE Design & Test of Computers*, 11(2):43–49, 1994.