A FPGA based Forth microprocessor

P. H. W. Leong, P. K. Tsang and T. K. Lee
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, N.T. Hong Kong

Systems which employ a microprocessor together with an application specific FPGA based coprocessor are common today. These applications can reduce power consumption and system costs by incorporating the microprocessor in the FPGA. For such applications, a microprocessor which has good performance, occupies a minimal amount of FPGA resources, has a good high level language software development environment and good code density is desirable. In this paper a 16 bit FPGA based microprocessor, called MSL16, optimised for such applications is described. MSL16 utilises a stack architecture with each instruction occupying only 4 bits, leading to a small instruction set, simple datapath and control, and high code density. MSL16 was specifically designed to efficiently execute the programming language "Forth". The Forth language has the desirable features of portability and high code density, and it is well suited to control, DSP, real–time and embedded applications.

The architecture for MSL16 is similar to that of the MuP21 [2]. The MuP21 is a 20 bit CPU which has 25 5–bit instructions and implemented in 1.2 micron CMOS process, uses 7000 CMOS transistors and has a peak execution rate of 100 MIPS. Compared with the MuP21, the MSL16 architecture has 16 4–bit instructions, and when implemented using a Xilinx Inc, 4000 series FPGA, occupies 175 configurable logic blocks (CLBs) at a peak clock frequency of 33 MHz on a 4006E–1 device (i.e. a peak execution rate of 33 MIPS).

The datapath of MSL16 is shown in Figure 1. MSL16 is a 2 stack machine with 16 bit data and memory buses. The data stack is used for temporary variable storage and subroutine parameter passing, and the return stack is used mainly to hold subroutine return addresses. The data and return stack are implemented internally on the FPGA which allows them to be accessed in parallel with instruction fetches on the memory bus. A two stage FETCH/EXECUTE pipeline is employed. Instructions involving a memory reference (@ and !), change the flow of execution, (CALL and GOTO) and SWAP take two cycles and the remaining instructions are single cycle. The main components in the datapath of MSL16 are a 16 deep data stack (DS) for temporary variables and subroutine parameters; the T register which holds the very top element of the stack so that the top two stack elements are available to the ALU simultaneously; a 16 deep return stack (RS) to store subroutine return addresses; an instruction register (IR) which holds the four 4-bit instructions to be executed; a PC (Program Counter) which stores the address of the next instruction; an IR (Instruction Register) which stores the address of the next instruction; and an ALU which takes operands from T and the top element of either DS or RS and returns the result to T.

Excepting the CALL instruction, MSL16 instructions (see Table 1) are encoded with 4 bits, allowing four instructions to be packed into a 16 bit word. A CALL instruction is recognised by the most significant bit of the IR being set, the remaining 15 bits forming the address. A consequence of this implementation of the CALL instruction is that the first slot is restricted to those instructions which has the high bit low (see top part of Figure 1), namely those with opcodes 0 to 7. The addition of a "SPECIAL" instruction which retrieves its opcode from the stack would enable the instruction set to be extended to any size, although obviously a performance penalty would be experienced for such instructions. A 16–bit instruction fetch obtains 4 instructions, reducing the required memory bandwidth and overall system power consumption. All of the instructions with the exception of CALL and LIT expect their operands to be on the stack. LIT expects its operands to be the remaining part of the IR. Thus if the LIT appears in the first slot of the IR, the remaining 12 bit constant of the IR is sign extended and pushed onto the stack; if the LIT is in the third slot, the remaining 4 bit constant is used. This feature improves code density since pushing small constants such as 0, -1 and 1 is reasonably common.

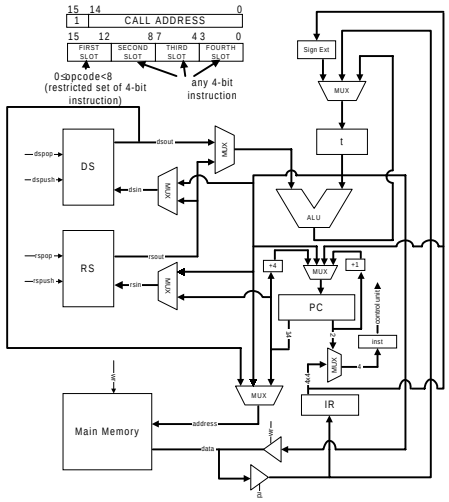

Figure 1: The datapath and instruction format of MSL16.

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 load a literal value to T
6	2/	T ⇐ T / 2
7	-	T ⇐ DS − T, pop DS
8	DUP	push T to DS
9	DROP	pop DS
10	GOTO	Jump to T if $T \neq 0$
11	R>	pop T to DS, pop RS to T
12	>R	pop 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 T

Table 1: The MSL16 instruction set

In the current implementation the memory speed is assumed to be the same as the processor's cycle time (33 MHz). A version in which the memory is 4 times slower than the processor's clock rate has also been developed, allowing either higher clock speeds or slower memory devices. A disadvantage of running the processor 4 times faster than memory is that when a memory operation such as a load, store or jump instruction is encountered, the pipeline must be stalled and 4 cycles used for the access hence greatly reduce the performance of the processor.

MSL16 has a simple instruction set and hence a short critical delay path. The two stage pipeline has low latency so the effect of stalling the pipeline during memory operations and branches is reduced. Operands to the ALU are normally the two top elements of the stack and the result is usually stored in the T register. This scheme virtually eliminates the instruction decoding and register fetch process normally required in a RISC machine.

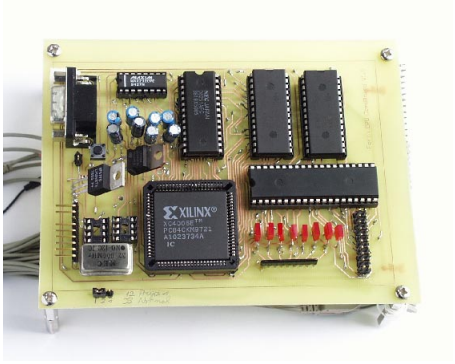

Figure 2: MSL16 prototype board.

The MSL16 design was synthesised from a VHDL description using the Synopsys Inc. *FPGA compiler*. The use of a high level language for the implementation of MSL16 enables easy customisation of the design. For example, the stack, sizes datapath widths, memory interface width and instruction set could be changed in the design. The resulting design occupied 175 configurable logic blocks (CLBs) and the VHDL description is generic except that the Xilinx RAM feature was used to implement the two stacks and program memory. A prototype printed circuit board containing MSL16 (implemented in a Xilinx 4006E–1 FPGA), static RAM, an Intel 8255 PPI chip, display LEDs and a RS232 interface was developed (see Figure 2). The system was tested with a program which utilises every MSL16 instruction as well as exercises the critical paths of the chip. It was found to be operational up to 33 MHz.

Forth machines are eminently suited for embedding in FPGA applications because they offer good code density, easy customisation, easily developed software development tools, high performance and small area. A 16 bit Forth processor was described which enjoys the above characteristics. This design, when synthesised from a VHDL description, occupies 175 Xilinx 4000 series configurable logic blocks (CLBs) and can operate at 33 MHz.

References

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

[2] C. H. Ting and C. H. Moore. Mup21–a high performance misc processor. *Forth Dimensions (also available at http://www.dnai.com/~jfox/mup21.html)*, January 1995.