International Journal of IT & Knowledge Management (ISSN: 0973-4414)
Volume 14 • Issue 1 pp. 26-30 Jan 2021 - June 2021 www.csjournals.com

Selective Bitwise Immediate Logic Operation in ALU and Data Memory of Microcontroller

Gulzar Singh, Anil Vohra
Department of Electronic Science, Kurukshetra University, Kurukshetra, Haryana, India

Abstract: In this paper, the authors present a new feature and its implementation, namely, Selective Bitwise Immediate Logic Operation (SBILO) in the ALU and Data Memory of Microcontroller. The designed model was coded in Verilog HDL, simulated using Xilinx ISE, and implemented in Field Programmable Gate Array (FPGA). Two designs have been implemented (i) SBILO in ALU with new SBILOA instruction (ii) SBILO in Data Memory with new SBILOD instruction. The SBILO circuit enhances the capability of ALU and Data Memory to concurrently manipulate multiple bits of a register (with operations like Set, Clear, Invert, No change) in a single machine cycle using Bit Operation Code. Authors also introduce new instructions SBILOA and SBILOD. The use of SBILO architecture is shown to reduce the program space and execution time requirement for bit manipulation up to 50% of the space consumed when using the traditional Bit Level or Byte Instructions. The SBILO circuit has been successfully tested in a microcontroller core in an FPGA.
Keywords: SBILO, ALU, CPU, ILA, Selective, Bitwise, Immediate.

1. INTRODUCTION

In the last several decades, Microprocessors & Microcontrollers have passed through many changes in Architectural Design, Fabrication Technology, Computational Speed, Data Size, etc. Researchers have worked in the past to enhance the capability and performance of Microcontrollers. Bernhard Schools[1] developed an algorithm to optimize the number of bank selection instructions for partitioned memory for a given block of instructions, in which the variables have already been allocated the bank location. The algorithm was tested to optimize for speed and size. To reduce no. of bank selection instructions, hardware-based assistance is provided by implementing shared data memory locations. Frequently used variables are placed in this shared memory. So, instructions are inserted in the application program to allocate addresses in shared memory. ChunyangGou[2] developed an algorithm to optimize the number of bank selection instructions for shared memory. Many microcontrollers do not have banked data memory. To access non-banked data memory, linear addressing is required and the full address is stored in the instruction which increases the requirement of Program Memory. Nash[4] developed a pseudo–linear scheme to avoid the wrap-around problem in segmented & banked memory. To reduce the consumption of Program Memory, instruction width is reduced by dividing the program address range into multiple pages. Page selection is provided through page selection bits in a register. Page selection instructions are then required to be inserted into the application program. Quing[3] developed an algorithm to optimize instructions for page selection. Jinpyo[5] used a control-flow directed acyclic graph (DAG) to optimize memory for embedded systems and XU Chao[6] developed an algorithm to optimize variable allocation based on block architecture. Robert[7] reduced the no. of instructions required to implement Atomic Read-Modify-Write instruction. Peter[8] implemented a method to speed up the read / write operation to/from memory so that maximum bus bandwidth is used. Mathew [9] described memory read – modify – write speed up by using separate banks for each type of read or write. Ray Brown[10] explained the implementation of Read-Modify-Write (RMW) functionality without using individual RMW circuit for each register & also registers to set/clear multiple bits in the register without affecting other bits.

Despite so many advancements in architecture & fabrication technology, there are several unresolved issues. Some of these issues are related to Bit Level Operations, Memory Access Instructions, and Bit Operations at Ports. In banked/paged architecture, bank/ page selection of data/program memory is done by inserting multiple bit level instructions in the application program. Similarly, in non-banked/non-paged architecture, complete instruction is stored in Code memory. Thus, both methods lead to enhancement in program memory utilization in terms of width or locations. In real applications, user-defined Boolean (1-Bit) flags are allocated in the byte-sized registers. The PORTS of the microcontroller also are Bit-addressable. The microcontrollers have code and data memory fabricated on the chip and are small in size, in many cases in KBytes only. The application developers, who aim to embed multiple features in the program, have to be content with this small-sized memory. Therefore, the application developer is required to use all the possible code and data memory optimization techniques in the programs.

Therefore, it will be advantageous to design a circuit that can reduce the number of Bit-level instructions. The authors did not find any source in the literature, to know the average percentage of Bit-Level Instructions in microcontroller application programs. In order to get a better estimate of the percentage of Bit-Level instructions, programs for several different applications were written. These programs have variations in the number of data variables and firmware functions. From these sample programs, it was inferred that the bank/page selection instructions form a considerable percentage in the application program.

To reduce the no of Bit-level instructions in application programs, an algorithm and a circuit based on it, is required for selective concurrent manipulation of bits. However, in the literature, no such work has been found. In this research paper, the authors report the design of a circuit to manipulate multiple bits selectively in a single machine cycle. The circuit designed by the authors has been termed as the ***Selective Bitwise Immediate Logic Operation (SBILO)*** circuit throughout the paper. As discussed later, the SBILO circuit designed by authors reduces the code space consumed by Bit-Level Instructions by 50%, which is beneficial for the application developer.

2. SBILO Circuit Design

The SBILO algorithm and the corresponding circuit perform a logic operation on a bit of a register as shown in Table 1. The architectural details of the SBILO circuit have been presented by the authors in one of their earlier communication[11]. The code for Bit Operation to be performed is passed through the instruction. To test the working of the SBILO circuit, a simple microcontroller architecture was designed and used by the authors. This research-oriented microcontroller architecture is based on generic architecture for microcontrollers. It was kept as simple as possible so that the designed circuit could be tested on a minimized hardware. It has four states, viz, Fetch, Decode, Execute, and Wait and each state completes its operation in one clock cycle, thus each machine cycle is of four clock cycles. In fig 1, the block marked 'Normal ALU' is the ALU implemented in the simple research microcontroller that is enhanced by using the SBILO circuit in ALU.

3. SBILO Circuit in ALU

Fig.1 shows the architecture block diagram of ALU with SBILO circuit, used by authors. As seen in fig 1, the normal ALU forms a part of the SBILO ALU which includes additional functionality of the SBILO circuit. To denote the instruction using mnemonics, the following notation has been used, 1 =>Set, 0 => Clear, ~ =>Invert, N=> No Change. With 0b01110000 loaded in the Accumulator, SBILO instruction 10~N1N~1 is performed and result 0b10011011 is visualized on the PORT LEDs. Similarly, SBILO 10~N1N~1 and SBILO N~01N~~1 on 0b01010010 results in 0b10111001 and 0b00011101 respectively. The whole architecture was coded as a Verilog model and was implemented in FPGA through Xilinx ISE software. The activity and PORT data were recorded using a logic analyzer setup in Chipscope Software. The recorded signals and PORT data are shown in fig. 2. From these results, it has been verified experimentally that the microcontroller worked as per the design requirements.

Table 1: SBILO logic

Operation	Operation Code	Previous Output Q_{n-1}	Output Q_n
Set	11	X	1
Clear	00	X	0
Invert	01	1	0
		0	1
No Change	10	1	1
		0	0

4. SBILO feature in DATA MEMORY

In this section, the authors present the implementation of SBILO capability in Data Memory so that the Data Memory can manipulate the contents of the addressed memory location by itself and, meanwhile, CPU and ALU remain free. The architecture block diagram of SBILO based Data Memory is shown in Fig 3. The SBILO instruction is passed as such to Data Memory. To keep things simple, SBILO features are added to the initial two locations of the memory, however, these can be implemented throughout the memory. The contents to be manipulated by SBILO instruction are loaded to either of these two locations. The data to be stored in data memory is selected by Data Mux and the address of the location is selected using Address Mux. During SBILO instruction execution, the address is selected from SBILO instruction. The data from the addressed location is fed to SBILO circuit. The SBILO circuit performs the Logic Operation on the data according to SBILO Instruction and the result is stored back on the same address location. When normal instruction is executed, the data mux selects data from the data bus and

address from the usual address bus of the microcontroller. The SBILO data memory architecture was practically tested in FPGA. Initially, memory locations 0 to 3 are loaded with 21H, 39H, 65H, 92H. Then, SBILO instructions passed to the SBILO circuit. The operation~N0~01~N on a memory location (0b0001), 1NN0~1N~ on location 0b0000 is performed. As expected, the first SBILO instruction output is 87H and the second is ACH. The recorded timing diagram in fig.4 shows that the SBILO circuit can perform all the four Logic Operations in a single machine cycle on the addressed memory location.

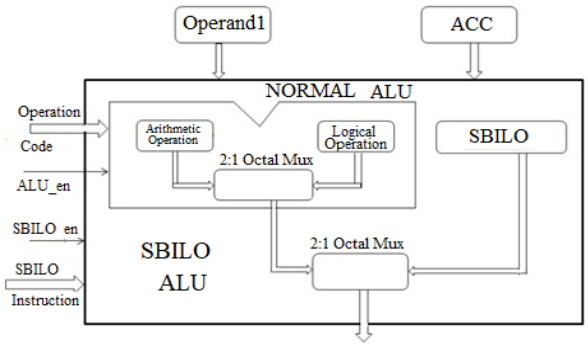

Fig 1: Block diagram of SBILO based ALU

5. Design validation and Testing

To verify the performance, several MCU Benchmark Tools were explored, like Dhrystone™, CoreMark™, IoTMark™, SecureMark™, ULPMark™, FPMark™, MultiBench™, TeleBench™ etc. It was found that these tools can evaluate the performance in terms of Mega Instructions Per Second (MIPS) or Iterations Per Second (IPS) by executing the application-specific standardized algorithms and compare the time or iterations to complete the algorithm. However, it was found that these Benchmarking Tools could not be applied to the work taken up by the authors because of the following limitations.

a. These tools do not support the new SBILO instructions introduced by the authors.

b. These Benchmark tools don't compare the MCUs in terms of code space requirement for a program.

c. These tools don't count and compare the number of instructions used for bank/page selection in the program generated by compilers/ assemblers for any algorithm.

d. These tools don't compare the memory space required for concurrent logic operation on bits in a register.

Since the existing tools could not be used to evaluate the performance of the algorithm and circuit developed by the authors, an alternate strategy was adopted. The authors compared the results obtained from the research-oriented microcontroller with those obtained from two well-known microcontrollers - 8051 and Microchip, in terms of code space and execution time taken by the bit-manipulation instructions. To test the performance of the SBILO operations and the designed circuit, an application program in machine code format was made a part of the microcontroller architecture and the complete Verilog code was then implemented in FPGA. After loading the machine code in the memory of FPGA, a Prg_loaded signal was activated in the design code, thereafter the machine code was run by the microcontroller core and the performance results were obtained. This did not require the use of standard compilers/assemblers and benchmark tools.

To get a comparison, the authors, also run the same code in two well-known microcontrollers – 8051 and MicroChip. All microcontrollers, including the research-microcontroller, were run at the same clock frequency which was chosen so that the machine cycle time was equal to 1 uS. The authors compared the code space required from the build results and execution time from the simulations in respective tools. However, similar results are expected at the rated frequencies of the respective microcontrollers. For comparison, the following operation was performed on the bits of a register:

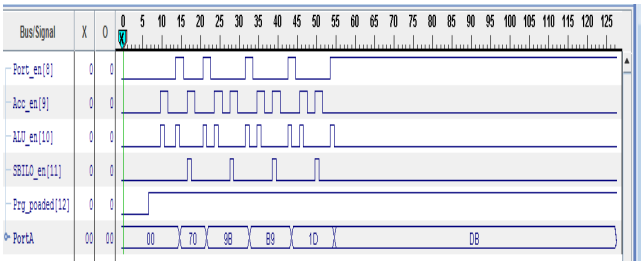

Fig 2: Logic Timing Diagram of SBILO based ALU in FPGA

Set bit nos: (*0, 3, 7*), Clear bit nos: *6*, Invert bit nos: (*1, 5*), and **No Change** bit nos: (*2,4*).

The above operations can be done using either Bit-level instructions or Byte Level Instructions as shown in Table 2 for the case of 8051 microcontroller and the similar instructions were written for Microchip as well. The authors used both of these methods in their studies. The results of the memory space required and the execution time taken by 8051 microcontroller, Microchip (PIC) microcontroller and SBILO based research-oriented microcontroller for the operations are shown in Table 3. It is evident from Table 3 that memory space requirement and the execution time taken by SBILO based instructions/architecture was much smaller, 1/6th of that required for bit-level and byte-level instructions of the 8051 ALU, 1/14th of the PIC. These results are also shown in Fig.5.

IJITKM 0973-4414
International Journal of IT & Knowledge Management (ISSN: 0973-4414)
Volume 14 • Issue 1 pp. 26-30 Jan 2021 - June 2021 www.csjournals.com

5.1 Impact Analysis: Authors wrote programs in embedded C language for the following four popular applications:

a. Stepper Motor Speed/Direction control
b. Metronome control
c. Traction Machine control
d. Data Logger

The programs for these were compiled for Microchip microcontroller using XC Compiler in MPLABX IDE and using KEIL compiler IDE for 8051. These microcontrollers have Bank/Page selection bits fabricated in the Status register. The statistics of bit-level instructions in the assembly code are shown in Table 4. If the SBILO circuit is added to the same microcontroller, then the memory space requirement for the Bit-Level manipulation instructions will reduce to nearly 50% of the normal. Table 5 shows that, for implementing SBILO, there is an enhanced requirement in FPGA resources in terms of Registers and LUTs by 2-3 times and Flip-Flops by 6 times of that used in basic research-oriented Microcontroller.

6. Conclusion

The SBILO circuit is most useful for microcontrollers having bank/page selection bits fabricated along with other bits in a register e.g. Microchip Mid-Range and 8051 architecture.

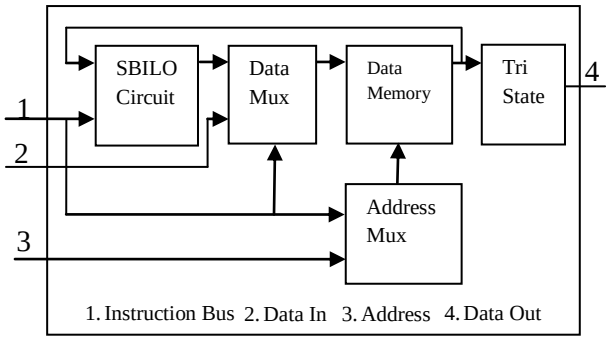

Fig 3: Block Diagram of Data Memory with SBILO capability

SBILO reduces the number of bank and page instructions to half. In the microcontrollers having a special bank/page selection register, the mirror address of the bank/page register is reserved in each bank, which results in wastage of address space. SBILO also resolves the issue of address wastage for Bank/Page register. Further, for the microcontroller having Bank and Page selection bits fabricated in a single register, the SBILO circuit ensures Bank and Page selection concurrently by using a single instruction. Also, the SBILO circuit reduces the execution time and memory space required for concurrent selective manipulation of multiple bits of any register to $1/6^{th}$ of that required for Bit and Byte Level. The memory code space requirement for bank/page selection instructions reduces by 50%. The overall code space-saving depends on the firmware. In the firmware having a large code size and a large number of variables, the code space-saving probability is more due to an increase in bank/page instructions. Since all the SBILO instructions take a single machine cycle to execute, therefore, the time for Bit-Level manipulation also decreases accordingly. In both of these designs, the data path remains the same but there can be an increase in complexity as the Instruction bus may need to be extended up to ALU and DATA Memory which will not add much overhead. It is expected that the overall critical path will not change.

Bit Instructions		Byte Instructions		SBILO
8051	PIC	8051	PIC	
Setb b^7 Clr b^6 CPL b^5 Setb b^3 CPL b^1 Setb b^0	BSF F,7 BCF F,6 BTFSC F,5 Goto j BSF F,5 Goto k J: BCF F,5 k: BSF F,3 BTFSC F,1 Goto L BSF F,1 Goto M L: BCF F,1 M:BSF F,0	MOV A , direct ORL A, #10001001 AND A, #10111111 MOV direct, A CPL b^5 CPL b^1	COmff,d movlw, 0x89 andwf f,1 movlw, 0xBF andwf f,1	SBILO 10~N1N~1

Table 3: Memory Space and Execution Time taken

	Bit Instructions		Byte Instructions		SBILO Instructions
	8051	PIC	8051	PIC	
Memory Space	6	14	6	5	1
Time (Machine Cycles)	6	16	6	5	1

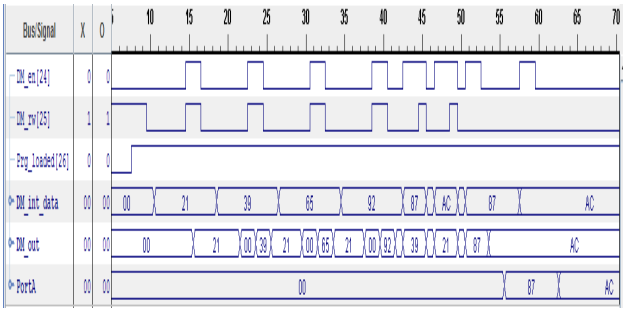

Fig 4: Logic Timing Diagram of SBILO Data memory in FPGA

International Journal of IT & Knowledge Management (ISSN: 0973-4414)
Volume 14 • Issue 1 pp. 26-30 Jan 2021 - June 2021 www.csjournals.com

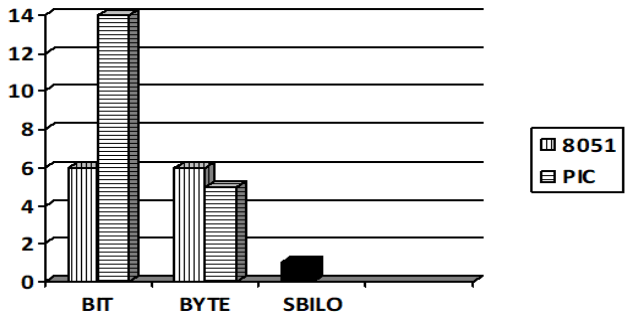

Fig 5: Space and Time Consumption in Traditional and SBILO Microcontrollers

Table 4 : Comparison of Code Space required in Normal Microcontroller and SBILO based Microcontroller

Application→		Stepper Motor Control	Traction Machine Control	Data Logger	Metronome
Bank Instructions	Normal	2	42	170	26
	SBILO	1	21	85	13
Page Instructions	Normal	244	166	680	84
	SBILO	122	83	340	42
Flag Instructions	Normal	3	3	50	0
	SBILO	1	1	6	0
Total Instructions in Application Program	Normal	2325	1434	8192	1539
	SBILO	2200	1326	7723	1484
Bit Level Instructions	Normal	249	214	900	110
	SBILO	124	106	431	55
Space Reduction for BIT Level Instructions		**125 (50.2%)**	**108 (50.4%)**	**469 (52.2%)**	**55 (50%)**

Table 5: Comparison of FPGA Resources used

Microcontroller Features	Slice Registers	Slice LUTs	LUT-FFs
Research Oriented Architecture (ROA)	153	231	90
SBILO ALU addition to ROA	378	409	545
SBILO Data Memory addition to ROA	424	435	574

References

1. Bernhard Schools, *Minimizing Bank Selection Instructions for Partitioned Memory Architectures*, The University of Sydney, October 23-25, 2006, Seoul Korea, Copyright 2006 ACM 1-59593 -543-6/06/0010.
2. Gou, Chunyang&Gaydadjiev, Georgi. (2011). *Addressing GPU On-Chip Shared Memory Bank Conflicts Using Elastic Pipeline*. International Journal of Parallel Programming. 41. 3. 10.1145/2016604.2016608.
3. Quing'An Li, *A Heuristic Algorithm for Optimizing Page Selection Instructions*, Computer School, Wuhan University, Wuhan, China,*2010 2nd International Conference on Software Technology and Engineering, 3-5Oct. 2010, SAN JUAN PR USA* .
4. Nash, James Carl, *A Microcontroller Having a Pseudo-Linear Bank Switching Memory Expansion Scheme*, MOTOROLA INC 1303 East Algonquin Road, Schaumburg, IL (US), European Patent.
5. Jinpyo Hong, *Memory Optimization Techniques for Embedded Systems*, a dissertation, Louisiana State University.
6. XU Chao, *An Optimization Algorithm of Variable Allocation Based on Block Architecture*, Applied Mathematics & Information Services, 7, No. 2, 691-699 (2013).
7. Robert B. Love, *Method & Apparatus for Executing an Atomic Read-Modify-Write Instruction*, Wang Laboratories Inc., Billerica Mass, Patent No. US005542084A.
8. Peter Bruce Gillingham, *Read/Write Timing for Maximum Utilization of Bidirectional Read/Write Bus*, Advanced Memory International Inc, San Jose, Calif., Patent: US006088774A.
9. Mathew Eldridge, *Read – Modify – Write using Read or Write Banks*, T-Ram Inc, Patent: US006804162B1.
10. Ray Brown, *Read/Modify/Write Registers*, LSI Logic Corporation, US006901490B2.
11. Gulzar Singh, Anil Vohra, '*Novel Hardware Accelerator Circuit for Bit-Level Operations in a Microcontroller*", Patent Application No. 202011024696, Patent Publication Date: 12/03/2021, Oficial Journal of The Patent Office, India, Issue No. 11/2021.