Abstract. Due to the complexity of microprocessor, an efficient testing is a crucial point and serious challenge in safety systems. A new instruction opcode for ALU based Built-In-Test (BIT) is proposed in this paper. With this novel method stuck-at-fault in Multiplexer (MUX) for Arithmetic Logic Unit (ALU) can be determined. A model that consists of the command and faulty states is developed. According to the designed state model, an algorithm and pseudo program that tests the stuck-at-fault in MUX is implemented and described in this paper.

Keywords: Built-in self-test, instruction based self-test, stuck-at-fault
2. Overview

Testing microprocessor functionality requires a good developers’ knowledge about design and structure of the hardware they are dealing with. There are different abstraction levels in hierarchical structure of microprocessor like transistor level, gate level and register transfer (RT) level. The RT level is often considered as a high level and contains control units for steering logic modules, functional blocks such as shifter, arithmetic and logic elements and registers.

The proposed methodology in this paper is based on the microprocessors instruction set architecture and tests the correctness of command select MUX in microprocessor by register transfer level. The main advantage of the high level RT test program is to achieve high fault coverage with few test cycles [1].

The testing of microprocessors’ functionality is an active research area and has been intensively studied by researchers for pipelined microprocessors [2] and even for multiprocessors [3]. Despite the significant advantages of functionality test based on data paths, structural test of microprocessor [4] that concentrates on operation of control logic is highly required. In [5] determining and characterising faults and their vulnerability of microprocessor structure are studied and implemented. Based on the functionality and structural test aspects, a new approach to detect stuck-at-fault in MUX block for instruction selection has been implemented here.

Generally, a microprocessor consists of following units such as control unit, ALU, memory unit, periphery unit, address unit and bus system. Complex microprocessors have multiple of the previously mentioned units in addition to several peripheral function blocks. In a microprocessor system, the command execution requires following steps: Instruction Fetch (IF), Opcode Fetch (OF), Execution (Exec), Memory Access (MA) and Write Back (WB). Some modern microprocessors can operate all these different steps at the same time where the term ‘pipeline’ comes. In superscalar microprocessors not only those different steps are executed but the multiple same steps can also be executed at the same time. For superscalar microprocessors the programmers must make allowance for the execution of a specific unit selected, if both instructions access the same logic or arithmetic unit or they access the same data. In this case some faults named “hazards” occur. Therefore test programmer must have good knowledge about the structure, instruction set architecture and data paths of the target microprocessor.

To create an effective BIST the faults have to be modeled and architecture of the ALU has to be more precisely studied. Figure 1 shows the architecture of the ALU of an n-bit microprocessor that contains arithmetical units, (adder, subtraction, multiplier, divider, floating point), logical units (AND, OR, XOR, NOT...) and state flags which are changed according to running operations, such as Carry (C). Moreover processed data will be selected by multiplexers and activated to output lines Z.
The operations executed by the ALU such as division, different shift and bit operations and so on are selected with the help of MUX. The multiplexers, on the right side of Fig1., select operations for ALU, depending on the state of the lines S0 to Sm (0 or 1). In case an error occurs on those lines, the wrong response will be activated into the output lines. During execution following faults on MUX can occur:

- Stuck-at-faults: this fault is the most common fault model and means that the circuit or line is fixed to logical “0” (stuck-at-0) or logical “1” (stuck-at-1). It occurs on the input or output line of logic gates or flip flops.
- Transition faults: logical level transitions between logical “0” and “1” are not executed.
- Coupling faults: when logical transitions on lines (0→1 or 1→0) take place, the state of another line is changed.

3. Implementation

The basic idea of this approach is to generate an effective BIST logic based on instruction set description file. The aim is to detect stuck-at-fault in MUX of ALU in microprocessor core. The generating of RT level code for hard-to-test steering logic, such as MUX, is not enough for testing; furthermore to study the effect of fault-masking it would be helpful to improve test results [6]. To detect stuck-at-fault on MUX, instructions set is classified by command opcode bits in instruction register. The BIST logic is described in Figure2.
Figure 2. Built-In Test (BIT) logic for stuck-at-fault detection of MUX in ALU.

At the beginning the opcode generator selects untested command from the instruction set description file. Due to the selected command all untested commands will be classified into two sections that depend on the most bit of command opcode. The BIT logic generates an instruction set from selected command and pattern. After execution of this instruction set in the ALU, the correctness of response is tested. If any fault is detected, the BIT logic tests the next operation, which differs from previous command opcode in only one bit. This procedure will be repeated until all commands are tested and after all feasible executions a stuck-at-fault on MUX can be determined. For the whole command test, the test program takes lot of time and is therefore not efficient enough. To evaluate the effectiveness of this approach a model that can generate optimized test program has been developed.

The proposed high level software self-test methodology is illustrated here for a 32 bit MCF5206e microprocessor with ColdFire processor core, which can execute 134 different instructions without considering the number of addressing mode and pattern registers.

The instructions are categorized according to their instruction opcode bits as shown Table 1.

| Bits15-12 | Hex | Operation                                      |
|-----------|-----|------------------------------------------------|
| 1         | 0000| 0 Bit Manipulation/immediate                    |
| 2         | 0001| 1 Move Byte                                     |
| 3         | 0010| 2 Move Longword                                 |
| 4         | 0011| 3 Move Word                                     |
| 5         | 0100| 4 Miscellaneous                                 |
| 6         | 0101| 5 ADDQ/SUBQ/Scc/TPF                             |
| 7         | 0110| 6 Bcc/BSR/BRA                                   |
| 8         | 0111| 7 MOVEQ/MVS/MVZ                                 |
| 9         | 1000| 8 OR/DIV                                        |
| 10        | 1001| 9 SUB/SUBX                                      |
| 11        | 1010| A MAC/EMAC Instructions/MOV3Q                   |
| 12        | 1011| B CMP/EOR                                       |
| 13        | 1100| C AND/MUL                                       |
| 14        | 1101| D ADD/ADDX                                      |
| 15        | 1110| E Shift                                         |
| 16        | 1111| F Floating-Point/Debug/Cache Instructions        |

Table 1. Operation code map of Coldfire Microprocessor’s Family [7].
In Figure 3 a new proposed testing approach that has been modeled in Matlab/Simulink Stateflow is shown. This model consists of the five following states: untested, tested, stuck-at-fault, transition fault and finally the block fault state. The first state shows that the command has not been tested yet whereby the second one refers to a tested command and shows that the result is correct. The last three states are faulty states, i.e. that by execution of commands the associated error occurs. In Figure 3, the S1, S2, S3 and S4 are the bits of the MUX dedicated for the instruction select whereby P0 and P1 are the test patterns. The functional units perform the actual execution of instructions and verify MUX operation.

Figure 3. BIST model for detecting stuck-at-fault in instruction selection MUX modeled in Matlab/Simulink Stateflow.

This model gives us the possibility to simulate the command selection bits on MUX in ALU efficiently. With the new approach discussed in this paper, only those states in the built model are enough to detect stuck-at-fault errors on MUX.
Table 2. Stuck-at-fault test program of MUX selecting bits.

| State of instruction select bits of MUX for detecting stuck-at-fault | Opcode for test sequence | Test sequence |
|------------------------|--------------------------|---------------|
| S1 S2 S3 S4 (P0)       | 1101                     | ADD.L %d0,%d1 |
| notS1 S2 S3 S4         | 0101                     | ADDQ.L #0x12345678,%d0 |
| notS1 notS2 S3 S4      | 0001 0001                | MOVE.B #0x33,%d1 |
| notS1 notS2 notS3 S4   | 0011 1101                | MOVE.W #0x1122,%d0 |
| notS1 notS2 notS3 notS4| 0010                     | MOVE.L #0x11223344,%d1 |

Table 2 presents the states of instruction select bits of MUX and the associated program sequence according to the designed state model. The stuck-at-fault on every select bit (S1, S2, S3 and S4) of the MUX can be detected after 5 test sequence, even after 3 test sequences for S3 and S4.

Before executing this test sequence, the data registers must be loaded with any random values. The first command (ADD.L) with the patterns (%d0, %d1) is executed. The associated instruction register format is shown in Table 3. To provide the fastest and efficient test program, arithmetical and logical operations are used, because they directly map onto the hardware components in the microprocessor. It is very important to recognize this MUX blocks, which are responsible over 80 instructions of the total amount of instructions (134) for the used microprocessor. In order to avoid pipelining hazards, the required commands and patterns are here selected.

Table 3. Instruction format for the command ADD.L %d0, %d1.

| 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 | Destination register | Opmode | Source effective Address | Mode | Register |
|----------------------------------------|-----------------------|--------|--------------------------|------|----------|
| 1 1 0 1                                | Register D1           | 001    | 010                      | 000  | 000      |

After first command execution (ADD.L), the result is tested in register d1. If a fault occurs, the reason of the fault may lie to the adder arithmetic block in ALU or to the multiplexer block for ALU function code from the instruction code.

During the execution of the generated test sequence, stuck-at-faults on select bits in MUX as well as block faults in arithmetic units can be detected. In the above treated example, the executed test sequence detected an adder block-fault after 4 commands.

4. Conclusion
In this paper a new instruction based built-in testing of instruction select MUX in ALU has been introduced. The BIST technique showed an attractive and precise solution to the testing of complex units in microprocessor. The design and simulation of a small and effective BIST for the detection of stuck-at fault by instruction execution have been one of the aims of this work. The model that consists of the command and faulty states which shows if the command executed or the result is correct, is...
developed. The states of instruction select bits of MUX and the associated program sequence are implemented according to the designed state model. The designed model and modified and reduced test program leads to a total reduction in the test application time. The developed model will be expanded for transition fault at MUX in future work. Moreover the implementation of this technique for modified processor core on FPGA is planned in the near future.

References
[1] N. Kranits, A.Merentitis, D.Gizopoulos “Hybrid-SBST Methodology for Efficient Testing of Processor Core”, IEEE Design & Test of Computers, Vol.25, 2008, pp.64-75
[2] D. Gizopoulos, M. Psarakis, M. Hatzimihail, M. Maniatakos, A. Paschalis, A. Raghunathan, S. Ravi “Systematic Software-Based Self-Test for Pipelined Processors” IEEE Trans on VLSI Systems, Vol.16, 2008, pp. 1441-1453
[3] A. Apostolakis, D. Gizopoulos, M. Psarakis, A. Paschalis, “Software-Based Self-Testing of Symmetric Shared-Memory Multiprocessors”, IEEE Trans on Computers, Vol. 58, 2009, pp. 1682-1694
[4] Y. Zhang, H. Li, X. Li „Software-Based Self-Testing of Processors Using Expanded Instructions“ 19th IEEE Asian Test Symposium (ATS), 2010, pp.415-420
[5] S. Pan, Y.Hu, X.Li “Characterizing the vulnerability of microprocessor structure to intermittent faults”, DATE’10 Proceedings of the Conference on Design, Automation and Test in Europe, 2010, pp.238-243
[6] C-H Chen, C-K Wei, T-H Lu, H-W Gao “Software-Based Self-Testing With Multiple-Level Abstractions for Soft Processor Cores”, IEEE Trans on VLSI Systems, Vol.15, 2007, pp.505-516
[7] Coldfire Family Programmers Guide Reference Manual, 2005
http://www.freescale.com/files/dsp/doc/ref_manual/CFPRM.pdf