
Politecnico di Torino

# Testing and fault tolerance

# Assignment 2016/2017

## Report

Master degree in Computer Engineering

Referents: Prof. Matteo Sonza Reorda, Riccardo Cantoro

Authors: Luca Mocerino, 229006
Ngongang Tchamou Leonel, 234740

January 22, 2017

# Summary

The main purpose of this assignment is developing a functional test program for manufacturing test of an OpenMSP 430 processor by Texas Instruments. Given the RTL version of processor, its synthesized netlist and the fault list, we have developed a test program in order to achieve the highest possible fault coverage considered as fault model the stuck-at.

## OpenMSP

The units under test for this assignment are a 16-bit RISC CPU and the hardware multiplier peripheral.

- **CPU**: The CPU has a single address space for instructions and data. Memory is byte-addressed, and pairs of bytes are combined little-endian to make 16-bit words. The processor has 16 registers, 13 of which are general purpose and the others are program counter (R0) , stack pointer (R1), status register (R2). The instruction set includes 27 instructions that belong to three main categories (single-operand arithmetic, two-operand arithmetic and conditional jump) and, for the majority are available in byte or in word version. There also 24 emulated operations and 7 addressing modes availables.
- **Hardware multiplier**: a memory-mapped hardware multiplier peripheral which performs various $16x16 + 32 \longrightarrow 33$-bit multiply-accumulate operations. This peripheral does not interfere with CPU activities.

# Test program

This section describes the different parts in which our the test program is divided.

### ALU

For testing the ALU we have used the combinational ATPG in order to have the relevant inputs, in that case the two 16-bit operands . Then we parse the file containing the input patterns, convert them in decimal and store then in two registers. Moreover, we implement a function that tests all ALU operations in word mode (ADD, ADDC, MOV, SUB, SUBC, CMP, DADD, BIT, BIC, BIS, XOR, AND, RRA, RRC, SWPB, SXT) with those inputs. The alu operations in byte are tested for one couple of input values. After those operations, the untested parts of the ALU are the jumps for which we write an additional section of code in order to test them and the flags in the status register. You can find all the scripts used for that part in the folder *".../scripts/alu"*.

### HW Multiplier

For testing the hardware multiplier the same flow described in the previous lines has been adopted. In this case, the two values stored are two operands of 16-bit and 8-bit. We implement a routine in order to test all possible operations available (unsigned and signed multiplication, unsigned and signed multiply and accumulate). Moreover, we implement another code section that reads and writes all multiplier accessible registers (MPY, MPYS, MAC, MACS, OP2, SUMEXT, RESHI, RESLO).

### Register File

For register file testing we implement a "sort" of MARCH algorithm plus some special additional instructions. The algorithm is composed by the following steps:

1. Write all '0' in all the registers (except R0 that is the program counter)
2. Apply move operations among registers
3. Write all '1' in all the registers (except R0 that is the program counter)
4. Write all '0' in all the registers (except R0 that is the program counter)
5. Subtruct '1' in all the registers using DEC instruction (except R0 that is the program counter)
6. Add '1' in all the registers using INC instruction(except R0 that is the program counter)

### Others

After the operations described above the testing program mainly covers ALU , register file, multiplier. Moreover we tested status register (**SFR**) reading and writing some flags (CPU_ID, CPU_NR, IE1,

IFG1). For the remaing parts we covered the remaing instructions with all addressing mode. In particular we tested:

- CALL
- PUSH/POP
- Pseudo (or Emulated) instruction as ADC, CLRC, CLRN, CLRZ, SETC, SETN, SETZ, DECD, INCD, SBC

## Results

In order to execute the test program we have used the provided scripts for simulation first and fault testing then. The results achieved are summarized in the following table:

| Number of faults | Test Coverage(%) | Instance name (type) |
|---|---|---|
| 29424 | 91.11 | top_module |
| 122 | 86.24 | omsp_clock_module |
| 5328 | 79.96 | omsp_frontend |
| 14544 | 94.47 | omsp_execution_unit |
| 9472 | 94.47 | omsp_register_file |
| 2568 | 96.11 | omsp_alu |
| 162 | 100.00 | omsp_alu_DW01_add_9 |
| 2054 | 93.09 | omsp_mem_backbone |
| 210 | 87.02 | omsp_sfr |
| 6362 | 92.44 | omsp_multiplier |
| 322 | 91.30 | omsp_multiplier_DW01_add_0 |
| 2752 | 93.31 | omsp_multiplier_DW02_mult_0 |
| 340 | 81.18 | omsp_multiplier_DW01_add_1 |

Table 1: Detailed fault coverage

Other relevant data are :

- **Number of input patterns**: 18992
- **Size of code**: 22Kb
- **Test duration**: 354.51 CPU time