8051 ALU UVM Verification Report

Almomenbelah Allam: V23009973 Mahmoud Mansour: V23010559

1. Introduction

1.1. Design Description

The 8051 ALU module is the arithmetic logic unit of the 8051 processor family. It is responsible for efficiently performing any arithmetic operations that are needed by the processor. It is able to perform the any one of the following operations:

- OC8051_ALU_NOP – no operation
- OC8051_ALU_ADD - adding
- OC8051_ALU_SUB - substracting
- OC8051_ALU_MUL - multiplying
- OC8051_ALU_DIV - dividing
- OC8051_ALU_DA – decimal adjust
- OC8051_ALU_NOT – negation, bit negation
- OC8051_ALU_AND – and, bit and
- OC8051_ALU_XOR – exclusive or
- OC8051_ALU_OR - or
- OC8051_ALU_RL – rotacion left
- OC8051_ALU_RLC – rotacion left with carry (operation swap nibbles)
- OC8051_ALU_RR – rotacion right
- OC8051_ALU_RRC – rotacion right with carry
- OC8051_ALU_PCS – adding 16 bit unsigned number with 8 bit signed number **(**two's-complement**)**
- OC8051_ALU_XCH – excange, first input is transfered to second output and *vice versa*. If carry is set only lowest halfs of bytes are changed

Most of these operations are combinational and produce the output in zero time, except for the multiplication and division operations, which take multiple cycles to compute their output. That is why the ALU needs to by synchronized using a clock and a reset signal.

2. Signals

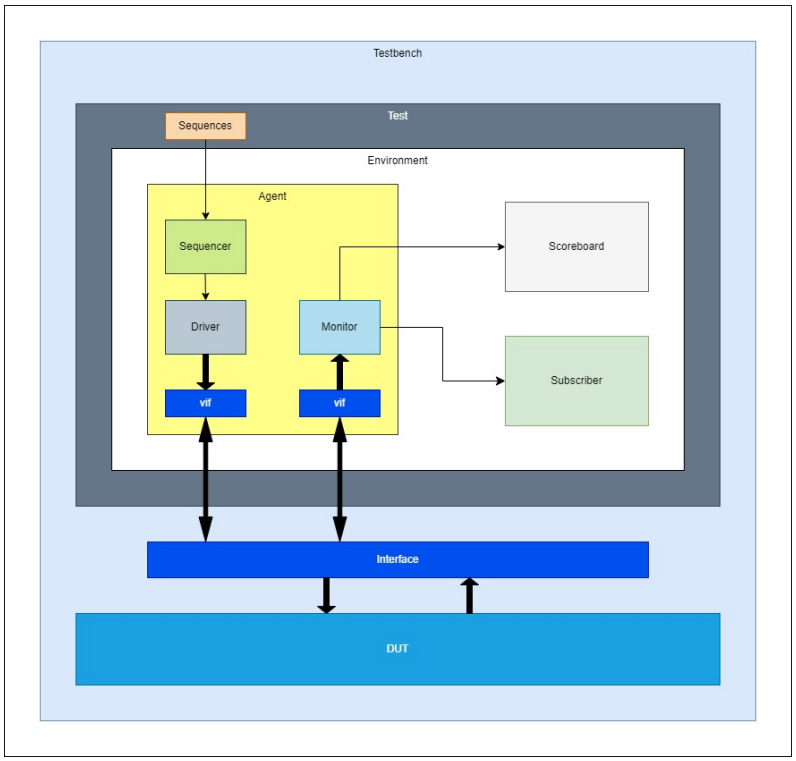
The ALU module interfaces with the following signals:

Signal Name:	Number of bits:	Description
clk	1	Clock used to synchronize the multiplier and divider modules
rst	1	Active high reset signal for the multiplier and the divider
op_code	4	Signal used to select the type of operation to perform on the inputs
src1	8	1st arithmetic input
src2	8	2nd arithmetic input
src3	8	3rd arithmetic input
bit_in	1	Signal used in some bitwise operations
srcCy	1	Carry in
srcAc	1	Auxiliary carry in
des1	8	1st arithmetic output
des2	8	2nd arithmetic output
des_acc	8	Auxiliary arithmetic output
desCy	1	Carry out
desAc	1	Auxiliary carry out
desOv	1	Overflow flag
sub_result	8	Result of src1-(src2+srcCy)

3. Verification Approach

3.1. Testbench Architecture and Hierarchy

The testbench will follow standard UVM architecture. It will contain one agent that will contain a Sequencer for managing the incoming sequence, a Driver for driving the interface, and monitor to sample the interface. The monitor sends its data to the scoreboard for error checking and to the subscriber for coverage collection. The schematic for the UVM environment is shown below.



3.2. Exit Criteria

To ensure that the module operation and functionalities are mostly covered, The following goals were set forward for the simulation:

- 100% code coverage of all lines in the RTL.
- 100% toggle coverage of all signals in the interface.
- 100% coverage of the created covergroups.

4. Coverage Collection

Functional coverage collection will be performed using the subscriber class. The table below shows the coverage bins needed for coverage collection.

Signal	Coverage Bins Description
reset_cp	1 bin for rst = 0, and 1 bin for rst = 1
op_code_cp	1 bin for each of the 16 op_codes
srcCy_cp	1 bin for srcCy= 0, and 1 bin for srcCy= 1
srcAc_cp	1 bin for srcAc= 0, and 1 bin for srcAc= 1
bit_in_cp	1 bin for bit_in= 0, and 1 bin for bit_in= 1
src1_cp	1 bin for src1= 0, 1 bin for src1= 255, and 1 bin for any value in between.
Src2_cp	1 bin for src2= 0, 1 bin for src2= 255, and 1 bin for any value in between.
Src3_cp	1 bin for src3= 0, 1 bin for src3= 255, and 1 bin for any value in between.
src1Xop_code	Cross bins between src1_cp and op_code_cp
Src2Xop_code	Cross bins between src2_cp and op_code_cp
Src3Xop_code	Cross bins between src3_cp and op_code_cp

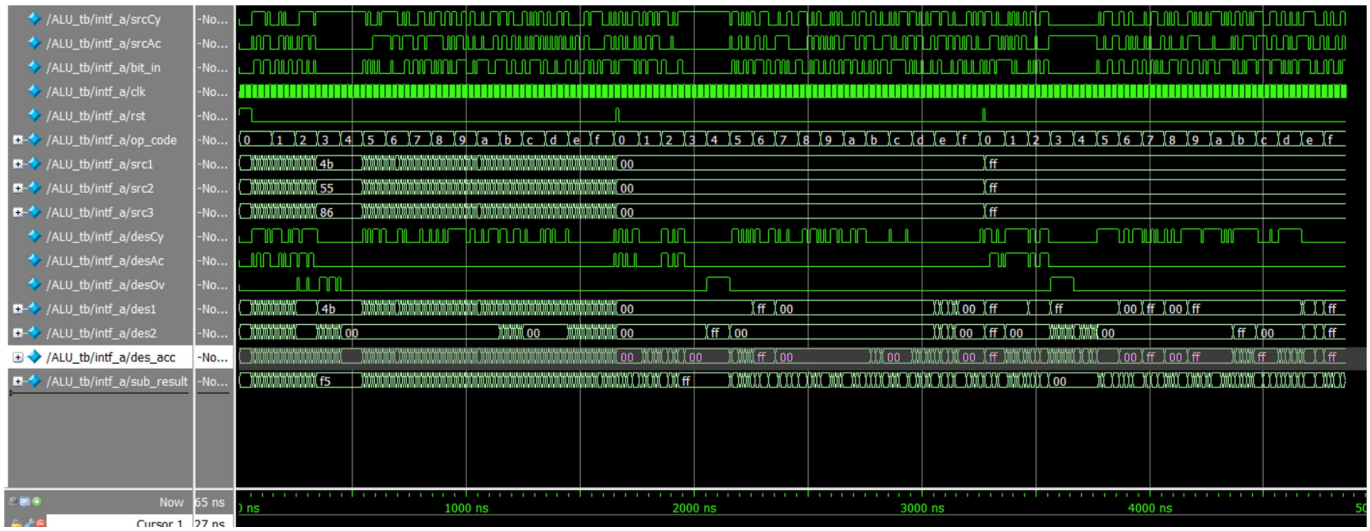
The bins created for the src1, src2, and src3 inputs prioritize their minimum and maximum values of 0 and 255 respectively. This is because these extreme values are more likely to cause unexpected errors in the ALU.

5. Test Implementation.

The DUT was subjected to 480 operations, consisting of 3 sets of a reset operation followed by 160 random operations. The Op_code input to the DUT changes every 10 operations. The first 160 operations (which cover all 16 operation types) have randomized src1, src2, and src3 inputs. The next 160 operations have the source inputs set to 0. The final 160operations have the source input set to 255. This way, we can ensure that we test the operations of the ALU at all extreme values. This could have also been implemented by adding constraints to the source inputs to make the values of 0 and 255 more probable during randomization. But this directed way is better to ensure 100% coverage of the created coverbins.

6. Evaluation.

The figure below shows waveforms of the test procedure.



The DUT output was found to be equivalent to the reference output at every clock cycle, which gives confidence that there are no bugs in the RTL.

The below figures show the coverage reports which were generated automatically by the simulation tool. The figure below shows the functional coverage of the RTL.

Coverage Summary By Instance:

Scope	TOTAL	Statement	Branch	FEC Expression	FEC Condition	Toggle
TOTAL	96.10	100.00	100.00	90.47	100.00	90.04
ALU_DUT	95.02	100.00	100.00	87.50	100.00	87.61
oc8051_mul1	98.03	100.00	100.00	--	100.00	92.15
oc8051_div1	99.44	100.00	100.00	100.00	100.00	97.22

We can see here that all statements and branches in the DUT were fully covered, with only minor toggle coverages for internal signals remaining missing. This confirms that all the logical parts of the DUT have been fully covered

The next figure shows the toggle coverage of all the signals in the interface.

Signal / Value	Hits						ExtMode	Status
	0L->1H	1H->0L	0L->Z	Z->1H	1H->Z	Z->0L		
bit_in	1	1	--	--	--	--	--	100.00%
clk	1	1	--	--	--	--	--	100.00%
des1[7-0]	1	1	--	--	--	--	--	100.00%
des2[7-0]	1	1	--	--	--	--	--	100.00%
desAc	1	1	--	--	--	--	--	100.00%
desCy	1	1	--	--	--	--	--	100.00%
desOv	1	1	--	--	--	--	--	100.00%
des_acc[7-0]	1	1	--	--	--	--	--	100.00%
op_code[3-0]	1	1	--	--	--	--	--	100.00%
rst	1	1	--	--	--	--	--	100.00%
src1[7-0]	1	1	--	--	--	--	--	100.00%
src2[7-0]	1	1	--	--	--	--	--	100.00%
src3[7-0]	1	1	--	--	--	--	--	100.00%
srcAc	1	1	--	--	--	--	--	100.00%
srcCy	1	1	--	--	--	--	--	100.00%
sub_result[7-0]	1	1	--	--	--	--	--	100.00%

The figure shows that all pins in the interface have performed rise toggles and fall toggles. This is useful in post-fabrication testing to check if any fabrication errors caused some the chip pins to be stuck at a specific value.

The final figure below shows the coverage of our implemented coverpoints.

cvr_grp

Summary	Total Bins	Hits	Hit %
Coverpoints	33	33	**100.00%**
Crosses	144	144	**100.00%**

Search:

CoverPoints	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
bit_in_cp	2	2	0	100.00%	**100.00%**	**100.00%**
op_code_cp	16	16	0	100.00%	**100.00%**	**100.00%**
reset_cp	2	2	0	100.00%	**100.00%**	**100.00%**
src1_cp	3	3	0	100.00%	**100.00%**	**100.00%**
src2_cp	3	3	0	100.00%	**100.00%**	**100.00%**
src3_cp	3	3	0	100.00%	**100.00%**	**100.00%**
srcAc_cp	2	2	0	100.00%	**100.00%**	**100.00%**
srcCy_cp	2	2	0	100.00%	**100.00%**	**100.00%**

Search:

Crosses	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
src1Xop_code	48	48	0	100.00%	**100.00%**	**100.00%**
src2Xop_code	48	48	0	100.00%	**100.00%**	**100.00%**
src3Xop_code	48	48	0	100.00%	**100.00%**	**100.00%**

We can see that our randomized test successfully achieved 100% coverage of our created cover groups. Even the very large cross bins between the op_code and the source inputs.