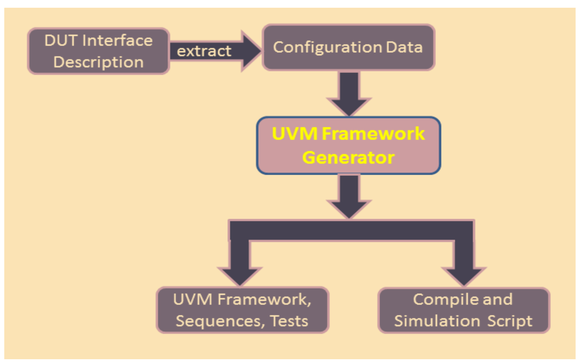
Community Based Verification of MIPS Microprocessor using UVM

Prof. Morris Jones, Saurabh Purohit, Udit Jaisalmeria
Department of Electrical Engineering, San Jose State University, San Jose, California 95192

Introduction

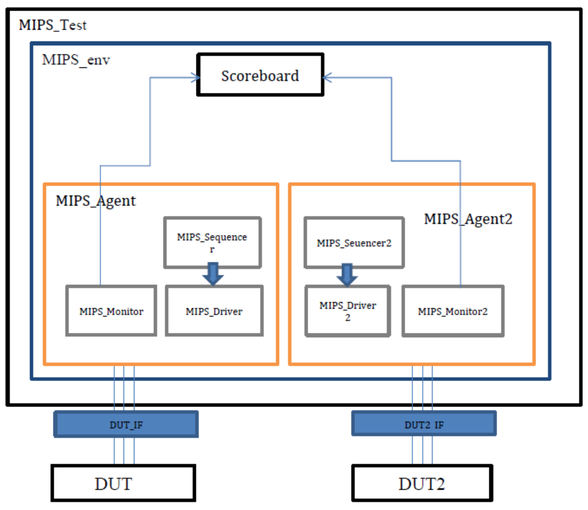
Our project focuses mainly on the verification methodology to test a particular design. We picked out 'MIPS Microprocessor Design' as our targeted design. We aim to create a UVM verification environment that is completely scalable to accommodate large number of identical designs and then perform a clean verification that compares the responses from all the designs. We call this verification as Community Based Verification[1].



Key Features of UVM:

- It has SystemVerilog Base Class Library.
- It is based on constrained Random Verification approach
- Powerful stimulus generation mechanism using sequences
- It supports reuse
- UVM Factory helps in easy to update or modify component without changing original code[2]
- UVM configuration helps in customizing the environment from Top or anywhere[2].

Verification Approach



Following are the basic building blocks of our UVM framework:

- Transaction Item (mips_seqitem) is a packet full of information which we pass through the framework.

Verification Approach

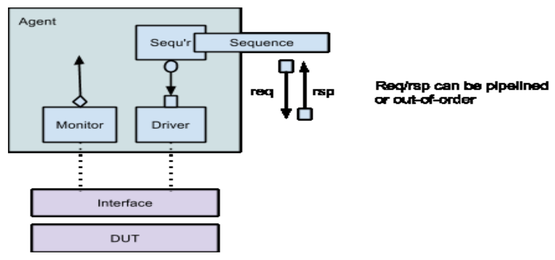
- Sequences(mips_sequence_r,mips_sequence_j,mips_sequence_i) R-type, I-type, J-type

operation	opcode	rs	rt	rd	sa	funct
ADD	000000	NO(1, 31)	NO(1, 31)	NO(1, 31)	00000	100000
SUB	000000	NO(1, 31)	NO(1, 31)	NO(1, 31)	00000	100010
AND	000000	NO(1, 31)	NO(1, 31)	NO(1, 31)	00000	100100
OR	000000	NO(1, 31)	NO(1, 31)	NO(1, 31)	00000	100101
XOR	000000	NO(1, 31)	NO(1, 31)	NO(1, 31)	00000	100110

operation	opcode	rs	Rt	immediate
ADDI	001000	NO(1, 31)	NO(1, 31)	-
ANDI	001100	NO(1, 31)	NO(1, 31)	-
ORI	001101	NO(1, 31)	NO(1, 31)	-
XORI	001110	NO(1, 31)	NO(1, 31)	-

operation	opcode	jmp
J	000010	jmp % 4 = 0
JAL	000011	jmp % 4 = 0

- Sequencers (mips_sequencer,mips_sequencer2) are two different sequencers each meant for running sequences for respective DUTs.
- Drivers (mips_driver,mips_driver2) is to convert the transaction level modelling into signal level behaviour.
- Monitor (mips_monitor,mips_monitors) are meant to receive the responses from the respective DUTs.
- Agents (mips_agent,mips_agent2) engulfs the sequencer and driver. These agents connect the driver with the respective sequencer inside a particular environment.



The most challenging and interesting part of UVM based verification is to create a way to built three different types of sequences (R-type, I-type, J-type). We adopted an approach in which we use a single transaction item names 'mips_seqitem' which has all the fields required by the three types of instruction formats. Table shows the contents of our transaction item

enable	inst_wr	inst_addr_top	opcode	rs	rt	rd	sa	funct	imm	jmp
1bit	1bit	12bit	6bit	5bit	5bit	5bit	5bit	6bit	16bit	26bit

The similarity in the responses from the two DUTs is measured from these parameters:

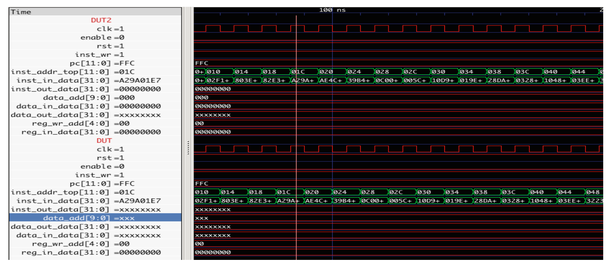
pc	inst_out_data	data_add	data_in_data	data_out_data	reg_wr_add	reg_in_data
12bit	32 bit	10 bit	32 bit	32 bit	5 bit	5bit

Results & Analysis

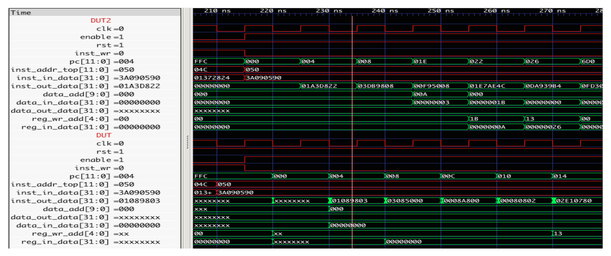
Design Inconsistency: We found that the output responses for the random tests were not identical. Results shown in table below:

Design name	pc	enable	inst_out_data	data_add	data_in_data	data_out_data	reg_wr_add	reg_in_data	Match (pc)
DUT	'h0	1	'h01089803	'h0	'h0	'hx	'h13	'h0	√
DUT2	'h0	1	'h01a3d822	'h0a	'h03	'hx	'h1b	'ha	
DUT	'h4	1	'h03085000	'h0	'h0	'hx	'h0a	'h20	√
DUT2	'h4	1	'h03db9808	'h0	'h1b	'hx	'h13	'h26	
DUT	'h8	1	'h8a800	'h0	'h0	'hx	'h15	'h8	√
DUT2	'h8	1	'h00f95008	'h0	'h0	'hx	'h0	'h2a	
DUT	'hc	1	'h80802	'h0	'h0	'hx	'h01	'h8	×
DUT2	'h1e	1	'h01e7ae4c	'h0	'h7	'hx	'h15	'h0	
DUT	'h10	1	'h2e10780	'h0	'h0	'hx	'h00	'h1f	×
DUT2	'h22	1	'h00a939b4	'h0	'h9	'hx	'h1f	'h0	
DUT	'h14	1	'h80b80382	'h387	'h0	'hx	'h18	'hx	×
DUT2	'h26	1	'h0fd30c00	'h0	'h13	'hx	'h1f	'h0	

Stimulus generation (green signals)



Output response waveform



Conclusions

The two designs were compared by injecting same random transactions multiple number of times. The results shows that there are some loopholes in one or probably both the designs. But as per the objective of our project, the result of our observation of responses is that we were able to find the test cases for which the two designs disagreed. As a result, , it becomes important for the designers of the DUTs to review their designs again to sort out the differences. This project can be extended to include any number of designs and rather than building a UVM checker and a predictor, we can simply use the results from the cloud of designs and know if the deign under consideration is faulty.

Key References

{1] Chai, L. (2014). A verification methodology for reusable test case and coverage based on system Verilog. *Electronic device and solid state circuits (EDSSC),2014 IEEE International Conference* (pp. 1-2). IEEE

[2] Drechsler, R., Chevallaz , C., Fummi , F., Hu, A., Morad, R., Schrimeister, F., & Goraychev, A. (2014). Panel: Future SoC Verification Mehodology: UVM evolution or revolution . *Design, Automation & Test in Europe Conference and Exhibition* (pp. 1-5). Europe: IEEE Conference Publications.

[3] Spear, C., & Tumbush, G. (2012). *System Verrilog For Verification.* Springer.

Acknowledgments

We would like to thank Prof. Morris Jones and Dr. Thuy Le for giving us the opportunity. Prof. Morris Jones's reviews, comments and discussions on this project work were really valuable.

For further information

Please contact udit.jaisalmeria@sjsu.edu or saurabh.purohit@sjsu.edu