ECE 593: Fundamentals of PreSilicon Validation
Cross Reference

Group 3: Yashodhan Wagle, Ramaa Potnis, Supreet Gulavani

The project has been divided in mainly two directories: `duv`/ and `tb`/ and contains a `Makefile` to run the SV project. The python script should be run before everything else providing an argument in the command line that would be equal to the `NumberOfTests` constraint in the `Makefile`

Makefile:

The makefile contains wildcards for the directories to be run and has the following recipes:
`make clean` : Cleans the build, removes all the generated test files
`make setup`: Sets up the work library
`make compile` : Compiles the project in the specified order with `duv` compiling first and then the `tb`
`make tests` : Creates the necessary .mem files that will be loaded into the memory. It has dynamically constrained stimuli `TXN` and `NumberOfTests` that can be specified before running the recipes. `TXN` specifies the transaction aka operation that would be tested by creating the mem file with the corresponding relevant opcode. `NumberOfTests` would eventually determine the number of files that would be created since the program size has been constrained to 1024
`make release` : Runs the test files generated using `make tests`
`make report` : Generates a text file called `report_func_cov.txt` with all the gathered functional coverage
`make html` : Generates html files of the coverage report, making it easier to view the coverage (tabular format)
`make all` : Runs all the recipes at once
`make info` : Displays all the recipes that can be run and the files in the build

duv/ :

This directory contains all the design files

- `blockram.sv` : A block ram module that loads from a memory file
- `disassembler.sv` : Disassembles the instruction codes to form a text string to be displayed
- `kcpsm_register.sv` : KCPSMX 16-location register module file

- kcpsm_rojo.sv : main file for RojoBlaze. Instantiates Instruction ROM, Instruction decode unit, register file, ALU, stack, scratchpad RAM, initializes pipeline registers, and runs the execute task
- kcpsmx_alu.sv : ALU file that contains operations in a case statement - ADD, ADDCY, COMPARE, SUB, SUBCY, AND, OR, TEST, XOR, RS
- kcpsmx_scratch.sv : A KCPSMX 64-location scratchpad RAM module file
- kcpsmx_stack.sv : KCPSMX call return stack module
- kcpsmx3_idu.sv : An instruction decode unit module
- kcpsmx3_inc.sv: A package file that has parameters, flags, typedef enum for shift operations, opcodes, typedef structures for register/register, register/constant, scratchpad operation, port operation, jump/call/return, interrupts, instruction opcode, and type
- rojoblaze_defs.sv : A package file that contains all the needed typedefs for the disassembler.sv file
- rojo_ref_model.sv : The reference model created by the team implementing the required functionality of the design. It instantiates the scratchpad module and has case statements for all the opcodes performing the necessary operations
- tb_testprogs.sv : A testbench file by the author that instantiates the entire design
- script.py : A python script that accepts an argument that is equal to the NumberOfTests (a dynamically constrained stimulus in the Makefile and creates a test.sv file
- test.sv : Created by running the python script, updates a variable memfile[] that is used to run the .mem files in tb_testprogs.sv

tb/ :

- transaction.sv : The transaction class provides constraints to generate a specific number of test cases and also options to constraint various types of instructions. Do note that since the program code length is limited to 1024 words, for test cases more than 1024, multiple files containing 1024 instructions each are created. Constraints are as follows: control_type, logical_type, interrupt_type, arithmetic_type, storage_type, io_type and shift_type. These create instructions of specific type and can be constrained from the Makefile.
- generator.sv : A generator module with mailbox gen_driv that communicates with the driver about which signals to drive. Contains an event ended that is triggered when all the inputs are driven.
    - task main : The main task that runs the mem files generated using the python script and make tests. Also randomizes the transaction class inputs

- `driver.sv` : This file contains an interface that is declared as a virtual interface in the driver class that's also in the same file
    - `task reset` : a reset task for the driver, resets all the inputs
    - `task main` : main task for the class, sends the inputs from the transaction class to the interface. Also counts the number of transactions that occurred
- `environment.sv` : This is the environment class that includes all the class files- transaction, generator, driver, monitor, scoreboard.
    - `task pre_test` : Calls the `reset` task from `driver`
    - `task test` : Calls the main tasks from the generator and the driver, fork joins both the tasks
    - `task post_test` : waits for the ended event to get triggered, counts the final number of transactions generated
    - `task run` : Calls all the above tasks
- `env_top.sv` : Contains a program test
- `rojo_bfm.sv` : A BFM to communicate between environment and design.
    - `task reset` : A reset task for the design
    - `task send_op_rojo` : A task that sends signals from the environment to the design
    - `task send_op_alu` : A task that sends signals to the ALU specifically and has conditions to trigger the ALU module
- `alu_tester.sv` : A tester module that generates static probability inputs for the BFM signals. Created as a part of unit testing that could not be completed
- `coverage.sv` : Contains all the `covergroups` and `coverpoints` in order to gather the coverage needed. There can be more coverpoints for internal registers, and they haven't been covered here since they would be a part of the unit testing. Also contains a sampling block where all the groups are sampled.
    - `covergroup cg_input_signals` :
        - `rojob_in_port` : Coverpoint for the `in_port`. 3 bins to cover values from 8'h00 to 8'FF
        - `rojob_interrupt` : Coverpoint for the interrupt signal
        - `rojob_reset` : Coverpoint for `reset` signal
        - `rojob_in_portxrojob_interrupt` : A cross between `rojob_in_port` and `rojob_interrupt`
        - `rojob_in_portxrojob_reset` : A cross between `rojob_in_port` and `rojob_reset`
        - `rojob_interruptxrojob_reset` : A cross between `rojob_interrupt` and `rojob_reset`
        - `rojob_in_en` : Coverpoint for `enable` signal in the block ram
        - `rojob_in_we` : Coverpoint for write enable in block ram

        - rojob_in_enxwe : Cross for en and we
        - rojob_in_ad : Coverpoint for ad signal in block ram
        - rojob_in_din : Coverpoint for din signal in block ram
        - rojob_in_write_enable : Coverpoint for write_enable in scratch
        - rojob_in_operation : Coverpoint for the type of operation in ALU
        - rojob_in_shiftop : Coverpoint for shift operation in ALU
        - rojob_in_shiftdir : Coverpoint for shift direction in ALU
        - rojob_in_shiftcons : Coverpoint for shift constant in ALU
        - rojob_in_operanda : Coverpoint for operand_a in ALU. Contains two bins opa0 and opa1
        - rojob_in_operandb : Coverpoint for operand_b in ALU. Contains two bins opb0 and opb1
        - rojob_in_carryin : Coverpoint for carry_in in ALU
        - rojob_in_operationxopa : Cross for operation and operand_a
        - rojob_in_operationxopb : Cross for operation and operand_b
        - rojob_in_operandaxb : Cross for operand_a and operand_b
        - rojob_in_shiftopxshiftdir : Cross for shift_operation and shift_direction
        - rojob_in_shiftopxshiftcons : Cross for shift_operation and shift_constant
        - rojob_in_shiftdirxshiftcons : Cross for shift_direction and shift_constant
        - rojob_in_operandaxcin : Cross for operand_a and carry_in
        - rojob_in_operandbxcin : Cross for operand_b and carry_in

    - covergroup cg_output_signals:
        - rojob_out_portid: Coverpoint for port_id of the main rojo module with three bins portid0, portid1, portid2
        - rojob_out_writestrobe : Coverpoint for write_strobe in rojo module
        - rojob_out_readstrobe: Coverpoint for read_strobe in rojo module
        - rojob_out_interupptack: Coverpoint for interrupt_ack in rojo module
        - rojob_out_outport: Coverpoint for out_port of the main rojo module with three bins outport0, outport1, outport2
        - rojob_out_portidxreadstrobe: Cross between rojob_out_portid and rojob_out_readstrobe
        - rojob_out_portidxwritestrobe: Cross between rojob_out_portid and rojob_out_writestrobe

        - `rojob_out_portidxinterruptack`: Cross between `rojob_out_portid` and `rojob_out_interruptack`
        - `rojob_out_readstrobexinterruptack`: Cross between `rojob_out_readstrobe` and `rojob_out_interruptack`
        - `rojob_out_writestrobexinterruptack`: Cross between `rojob_out_writestrobe` and `rojob_out_interruptack`
        - `rojob_out_dout` : Coverpoint for `dout` with two bins `dout0`, `dout1`
        - `rojob_out_result` : Coverpoint for `result` of the ALU module with two bins `res0` and res1
        - `rojob_out_zout` : Coverpoint for `zero_out` of the ALU module
        - `rojob_out_carryout` : Coverpoint for `carry_out` of the ALU module
        - `rojob_out_resultxzout` : Cross between `rojob_out_result` and `rojob_out_zout`
        - `rojob_out_resultxcout` : Cross between `rojob_out_result` and `rojob_out_carryout`
- `tb_monitor.sv` : A monitor module for sampling signals from the `duv`. Contains a `task main` for that purpose
- `testbench.sv` : Instantiates all the modules- `rojo_bfm`, `alu_tester`, `coverage_ifid`, `rojo_ref_model`. The `top` for the project