Matty Baba Allos
Alex Olson
Nurettin Can Orbegi
ECE 593 - Spring 2019
Final Project Report

Abstract

The purpose of this project was to verify some of the functionality of a MIPS-16 processor. The processor consists of five stages (instruction fetch, instruction decode, execute, writeback, and memory) as well as a hazard detection unit, instruction/data memory, and ALU to support the five stages. The testbench strategy for verifying the design was to use an object-oriented approach with modules for drivers, checkers, coverage, and generators. Due to limited time and an unexpected surprise for one of the team members, we were only able to focus on what we considered to be the critical parts of the design verification. With that in mind we chose to verify the ALU because it was the core part of the functionality of the design, and we chose to verify the hazard detection unit because it is responsible for preventing the manifestation of many, many bugs in an otherwise working design. We also developed a chip level testbench to verify the functionality of the chip at a higher level.

The Original Design

Original design is a 16-bit MIPS processor. It is designed for educational purposes. It is published on https://opencores.org Our verification plan is made on the design documents which are within the Documents/DUV Overview Documents/ folder.

Introduction

The design chosen for verification was a 16-bit MIPS educational processor from opencores.org that had a minimized instruction set for a simple learning environment. The processor supports the following 13 instructions: NOP, ADD, ADD Immediate, SUB, AND, OR, XOR, Shift Left, Shift Right, Shift Right Logical (SRU), Load, Store, and Branch. Note that the processor does not require a jump instruction because the branch instruction can cover the entire 16-bit address space. The complexity of this project was fairly complex, namely due to the enclosed nature of the top-level design where the only inputs were clk and rst. The input/output of the processor's instructions and data were piped in and out of the memory integrated within the processor.

Stimulus

For verifying the ALU within the EX stage, the design is driven with random 16-bit data and with random 4-bit input for the 13 operations. This also covers the 3 undefined operation in the design. The ALU transactions are performed with the use of an ALU transaction class which is extended from a parent transaction class. This extended class generates a random instruction as well as random data to operate with when it is instantiated. The transactions are generated within the ALU environment class which also holds the driver for this part of the DUT. The driver uses an interface to drive signals into the ALU that were randomly generated and signals a mailbox that is used by the scoreboard. In order to control the transaction with the design, a callback system is used with peeking so that the driver can send another command once the DUT is finished with the previous transaction. The input for the hazard detection unit also uses randomly generated stimulus driven in using the same transaction method.

Coverage

The first coverage group used to monitor the ALU results is one that monitors the data used for the input operands. The specific coverpoints checked that each input experiences all zeros, all ones, and a mixture of the two. The second coverage group monitors the result register of the ALU to also ensure that the result is capable of all zeros, all ones, and a sampling of in between as well. Monitoring the coverage of the hazard detection unit was simplified to checking that the input/output pins all experience a high and low signal.

Checking

The results were examined with a simple reference model inside the scoreboard class. For each transaction observed going into the ALU, the result is predicted by the scoreboard and compared used a method from the transaction class. Once the test has finished running, a message displays the results of the test to the console. The hazard detection unit also uses a reference model with the purpose of predicting the results of different transactions. If a hazard detection transaction fails, then a message is displayed to the console.

Testbench

The ALU testbench in the design simply instantiates the ALU test environment and gives the commands to build and run the test as well as generate all of the transactions to drive into the test.. The environment is where the testbench components (driver, scoreboard, monitor,

coverage, etc.) are instantiated and integrated together. The stopping condition for the testbench is after it drives 19 transactions into the design. The hazard detection testbench essentially uses the same random input parameters, but repeats the random transactions for a total of 1000 times.

Chip Level Verification

The design starts executing instructions one by one in every positive edge of clock signal. The design fetches instructions from the instruction register. Instructions must be saved into the instruction register before execution. So, this type of structure is not appropriate for on the fly generated test cases. Therefore, Chip level test bench generates instructions and saves them into a txt file. Then, It fills the instruction register of DUV with these pre-generated instructions.

Coverage

There are two coverage groups used for monitoring coverage for opcodes and register values. The design supports 2 types of instructions; R-type and I-type. First coverage group tracks each type of instructions individually. Second coverage group tracks values of destination register, source registers and offset values.

Checking

The design has internal instruction registers and data registers. The design fetches and performs the instructions and saves outputs into data registers. The reference design has similar structure. Thus, checker compares the data registers of reference model and the design. If any differences it gives warning.

Reference Model

Reference model is basic 5 - stage MIPS architecture. Designed to behave like the original design. The reference model has internal instruction registers and data registers. It performs operations and saves the results into the data registers. The reference model class has different methods for IF, ID, EX, MEM, WB stages. Unlike the original design, hazard detection is performed during ID stage in the reference model.

- How much effort was required (roughly number of hours)?

  We put approximately 80+ hours of work on the project between three people.
- Challenges you encountered
  - We wanted to base our testbench for unit level verification on the Utopia testbeach shown in class. It took us a bit of time to understand that testbench before we started coding our own testbenches. However, for the chip level verification we created a basic object oriented testbench. Our design is relatively big design so, covering everything in a limited time was challenging.
- Makefile and environment needed to run
  - For the environment we used QuestSim GUI simulator to compile and run our testbench. Next we moved to create a makefile as it's requirement for the project.
- What pre-existing code/ideas did you leverage?
  - We based our ALU and hazard detection testbenches heavily on the Utopia example.
- Description of test bench/tests executed/assessment of coverage
  - Due to the time and team member constraints, we were only able to "verify" the ALU unit, hazard detection unit, and the chip level design. Both ALU unit and hazard detection are very similar in structure.
- Did you find bugs? Did you inject bugs?
  - Our test found that the DUV was working for the purpose of our environment. We injected some ALU operation bugs to miscalculate an ADD operation and the testbench displayed appropriate error messages to the console. However, during the chip level verification we realized some times data registers in duv get X values. It causes a mismatch between the registers in DUV and registers in reference design.
- Next steps: What more might do if you had more time?

- We would have finished verifying the rest of the processor and would have more improvements to the testbench. We would have made the test environment more dynamic and added more test case scenarios.

Cross Reference of Chip Level Verification

- MIPS_top.sv
  - **Classes:**

    MIPS_coverage: Includes coverage groups.

    mips_IF: Interface signals.

    MIPS_generator: Generates tightly constrained random instructions.

    MIPS_reference: Reference model.

  - **Function and Tasks:**

    fill_inst_mem() : Fills instruction memory with generated instructions.

    enable_clock(): Activates clock generator.
- MIPS_coverage.sv
- MIPS_reference
  - **Classes:**

    MIPS_scoreboard_checker: This class includes scoreboard and checking methods. Checking method controls the register values of DUV and reference design. Scoreboard saves instruction number, register values of DUV and reference model.
  - **Functions and Tasks:**

    loadInstructionMemory: This method loads the instruction memory of reference design.

    readRegister: This method returns the register value at given address.

    writeToRegister: This method writes given value into the given address.

    IF: It fetches instructions.

    ID: Decodes instructions and checks any hazards.

    EX: Executes operations.

    MEM: Performs memory operations.

    WB: Write backs values into registers.

    execute: Runs the reference model.

- MIPS_generator.sv
    - **Functions and Tasks:**

      REG_gnrt: Picks random register.

      IMM_gnrt: Generates 6 bit immediate value.

      NOP_gnrt: Generates NOP instruction.

      ADD_gnrt: Generates ADDinstruction.

      SUB_gnrt: Generates SUB instruction.

      AND_gnrt: Generates AND instruction.

      OR_gnrt: Generates OR instruction.

      XOR_gnrt: Generates XOR instruction.

      SL_gnrt: Generates SL instruction.

      SR_gnrt: Generates SR instruction.

      SRU_gnrt: Generates SRU instruction.

      ADDI_gnrt: Generates ADDIinstruction.

      LD_gnrt: Generates LD instruction.

      ST_gnrt: Generates ST instruction.

      BZ_gnrt: Generates BZ instruction.

      generateTestFile: Generates a txt file that includes 255*13 instruction.
- MIPS_scoreboard_checker.sv
    - **Functions and Tasks:**

      checking: This method compares the register values of DUV and reference design.

      scoreboard: This method saves instruction number and register values into a txt file.
- MIPS_if.sv: Interface signals
- MIPS_pkg.sv: Includes definition of opcodes.

Cross Reference of ALU Verification

Alu_coverage.sv - this class is used to collect functional coverage

- class alu_coverage;
    - function new(); - create a new instance
        - alu_txn_cov = new;
        - alu_result_cov = new;

          Called From:
            - env.build
    - function void sample_alu_txn(alu_txn txn); - Sample input stimulus
        - alu_txn_cov.sample();

          Called From:
            - driver_coverage_cbs.post_drive
    - function void sample_alu_result(alu_result_txn txn); - Sample output result
        - alu_result_cov.sample();

          Called From:
            - monitor_coverage_cbs.post_monitor
    - *

Alu_defs.sv - Define constants and macros

Alu_driver.sv -Define classes needed to drive stimulus on the DUT

- class alu_driver_cbs; - Define a set of functions to be called before and after driving stimulus. This class allows for writing a flexible testench.
    - virtual task pre_drive(input alu_driver drv, inout bit drop); - Function to be called before driving stimulus

      Called From:
        - alu_driver.pre_drive
    - virtual task post_drive(input alu_driver drv, alu_txn txn); - Function to be called before after driving stimulus

      Called From:
        - alu_driver.post_drive
- class alu_driver; - Drive stimulus to the DUT
    - function new(mailbox generator_to_driver, event driver_to_generator_event, virtual alu_intf intf); - Create a new class instance

Called From:

- env.build

- task run(); - Forever wait for stimulus to be driven on the DUT
  - pre_drive();
  - drive();
  - post_drive();

    Called From:

    env.run
- task drive (); - Drive signal level stimulus on the DUT.
- protected task pre_drive(); - Prepare the DUT before driving stimulus
  - generator_to_driver.peek(txn);
  - cbs_list[i].pre_drive(this, drop);
- protected task post_drive(); - Clean up after driving simulus.
  - cbs_list[i].post_drive(this, txn);
  - generator_to_driver.get(txn);

Alu_env.sv - Define the testbench environment.

- class driver_scb_cbs extends alu_driver_cbs; - Used to send stimulus transaction to the scoreboard
  - function new(alu_scoreboard scb);
  - virtual task post_drive(input alu_driver drv, alu_txn txn); send stimulus transaction to scoreboard
    - scb.save_current_txn(txn);
- class driver_coverage_cbs extends alu_driver_cbs; Used to send stimulus transaction to the coverage
  - function new(alu_coverage cov);
  - virtual task post_drive(input alu_driver drv, alu_txn txn); send stimulus transaction to coverage
    - cov.sample_alu_txn(txn);
- class monitor_scb_cbs extends alu_monitor_cbs; - Used to send result from monitor to scoreboard
  - function new(alu_scoreboard scb);
  - virtual task post_monitor(alu_result_txn txn); Send result from monitor to scoreboard to perform the check.

-
    - scb.check_result(txn);
- class monitor_coverage_cbs extends alu_monitor_cbs; -Send results from monitor to coverage
    - function new(alu_coverage cov);
    - virtual task post_monitor(alu_result_txn txn); -Send results from monitor to coverage
        - cov.sample_alu_result(txn);
- class environment; - Define the test environment for the testbench
    - function new(virtual alu_intf intf); - Create a new instance of the class.
        - Called From:

            Test
    - virtual function void build(); - Build the environment by creating and connecting all the required object to perform the tests
        - generator_to_driver = new;
        - driver = new(generator_to_driver, driver_to_generator_event, intf);
        - generator = new(generator_to_driver, driver_to_generator_event);
        - monitor = new(intf);
        - scoreboard = new;
        - coverage = new;
        - driver_to_scb_cbs = new (scoreboard);
        - monitor_to_scb_cbs = new (scoreboard);
        - driver_to_coverage_cbs = new (coverage);
        - monitor_to_coverage_cbs = new (coverage);

        Called From:

            Test
    - virtual task run(); - Run the test environment components
        - driver.run();
        - monitor.run();
        - generator.run();

        Called From:

            Test

- virtual function void finish(); -Finish the test and clean up the environment
  - scoreboard.finish();

    Called From:

    Test

Alu_generator.sv

- function new(mailbox generator_to_driver, event driver_to_generator_event);
- task run(); - run the stimulus generator
  - txn = new;
  - txn.randomize()
  - generator_to_driver.put(txn);

  Called From:

  env.run

Alu_intf.sv - Defines the interface that connects to the DUT.

Alu_monitor.sv - Implements a monitor that gather results as they come out of the DUT.

- class alu_monitor_cbs; - defines a callback hook to be used to pass result transactions.
  - virtual task post_monitor(alu_result_txn txn); - Callback hook to pass result transactions.

    Called From:

    alu_monitor.run
- class alu_monitor; - defines the class that will monitor the results from the DUT and send it using the callback class.
  - function new(virtual alu_intf intf); - Create a new instance of the class
  - task run(); - starts the monitor
    - mointor_alu();
    - cbs_list[i].post_monitor(txn);

    Called From:

    env.run
  - task mointor_alu(); - listen to signal level changes to detect a new output result.

Alu_scoreboard.sv - Implements a scoreboard and a reference model

- class alu_scoreboard;
    - function new(); - create a new instance of the scoreboard

        Called From:

            env.run
    - function void save_current_txn(alu_txn txn); - Save in flight transaction for later checking

        Called From:

            driver_scb_cbs.save_current_txn
    - function void check_result(alu_result_txn result_txn); - Check the results that comes out of the DUV and make sure it's the expected result.
        - if(!result_txn.compare(predict_result(current_txn)))

        Called From:

            monitor_scb_cbs.post_monitor
    - function void finish(); - End the scoreboard and make sure there is not a transaction in flight.

        Called From:

            Test
    - protected function alu_result_txn predict_result(alu_txn txn); - Implements a reference model that produce the predicted result that DUV should produce.
        - predicted_txn = new();

Alu_test.sv - Implements a test program and environment.

Alu_top.sv - the top level module, connects the interface and starts the test.

Transaction.sv - Defines a stimulus transaction that goes into the DUT.

- virtual class transaction; - Defines the transaction class
    - function new(); - create a new transaction instance
    - pure virtual function bit compare(transaction to); - Deep compares this transaction to another instance.
    - pure virtual function transaction copy(transaction to=null); - Deep copies this class instance and return its handle
    - pure virtual function void display(string prefix=""); - Display the transaction fields

      Called from:

        transaction

- class alu_txn extends transaction; - Defines input stimulus transaction
    - function new();
        - super.new();

          Called from:

          Used throughout the testbench.

          ---These functions overrides the super class and are similar.
    - virtual function bit compare(transaction to);
    - virtual function transaction copy(transaction to=null);
    - virtual function void display(string prefix="");
- class alu_result_txn extends transaction; - Defines the output transaction
    - function new();
        - super.new();

          Called from:

          Used throughout the testbench.

          ---These functions overrides the super class and are similar.
    - virtual function bit compare(transaction to);
    - virtual function transaction copy(transaction to=null);
    - virtual function void display(string prefix="");

Cross Reference of Hazard Detection Verification

hd_coverage.sv - this class is used to collect functional coverage

- class hd_coverage;
    - function new(); - create a new instance
        - hd_txn_cov = new;
        - hd_result_cov = new;

          Called From:

            - env.build
    - function void sample_hd_txn(hd_txn txn); - Sample input stimulus
        - hd_txn_cov.sample();

          Called From:

            - driver_coverage_cbs.post_drive
    - function void sample_hd_result(hd_result_txn txn); - Sample output result

        - hd_result_cov.sample();

          Called From:

            - monitor_coverage_cbs.post_monitor

hd_defs.sv - Define constants and macros

hd_driver.sv -Define classes needed to drive stimulus on the DUT

- class hd_driver_cbs; - Define a set of functions to be called before and after driving stimulus. This class allows for writing a flexible testench.
    - virtual task pre_drive(input hd_driver drv, inout bit drop); - Function to be called before driving stimulus

      Called From:

        - hd_driver.pre_drive
    - virtual task post_drive(input hd_driver drv, hd_txn txn); - Function to be called before after driving stimulus

      Called From:

        - hd_driver.post_drive
- class hd_driver; - Drive stimulus to the DUT
    - function new(mailbox generator_to_driver, event driver_to_generator_event, virtual hd_intf intf); - Create a new class instance

      Called From:

        - env.build
    - task run(); - Forever wait for stimulus to be driven on the DUT
        - pre_drive();
        - drive();
        - post_drive();

          Called From:

            env.run
    - task drive (); - Drive signal level stimulus on the DUT.
    - protected task pre_drive(); - Prepare the DUT before driving stimulus
        - generator_to_driver.peek(txn);
        - cbs_list[i].pre_drive(this, drop);
    - protected task post_drive(); - Clean up after driving simulus.

        - cbs_list[i].post_drive(this, txn);
        - generator_to_driver.get(txn);

hd_env.sv - Define the testbench environment.

- class driver_scb_cbs extends hd_driver_cbs; - Used to send stimulus transaction to the scoreboard
    - function new(hd_scoreboard scb);
    - virtual task post_drive(input hd_driver drv, hd_txn txn); send stimulus transaction to scoreboard
        - scb.save_current_txn(txn);
- class driver_coverage_cbs extends hd_driver_cbs; Used to send stimulus transaction to the coverage
    - function new(hd_coverage cov);
    - virtual task post_drive(input hd_driver drv, hd_txn txn); send stimulus transaction to coverage
        - cov.sample_hd_txn(txn);
- class monitor_scb_cbs extends hd_monitor_cbs; - Used to send result from monitor to scoreboard
    - function new(hd_scoreboard scb);
    - virtual task post_monitor(hd_result_txn txn); Send result from monitor to scoreboard to perform the check.
    -
        - scb.check_result(txn);
- class monitor_coverage_cbs extends hd_monitor_cbs; -Send results from monitor to coverage
    - function new(hd_coverage cov);
    - virtual task post_monitor(hd_result_txn txn); -Send results from monitor to coverage
        - cov.sample_hd_result(txn);
- class environment; - Define the test environment for the testbench
    - function new(virtual hd_intf intf); - Create a new instance of the class.
        - Called From:

                Test

- virtual function void build(); - Build the environment by creating and connecting all the required object to perform the tests
    - generator_to_driver = new;
    - driver = new(generator_to_driver, driver_to_generator_event, intf);
    - generator = new(generator_to_driver, driver_to_generator_event);
    - monitor = new(intf);
    - scoreboard = new;
    - coverage = new;
    - driver_to_scb_cbs = new (scoreboard);
    - monitor_to_scb_cbs = new (scoreboard);
    - driver_to_coverage_cbs = new (coverage);
    - monitor_to_coverage_cbs = new (coverage);

  Called From:

    Test
- virtual task run(); - Run the test environment components
    - driver.run();
    - monitor.run();
    - generator.run();

  Called From:

    Test
- virtual function void finish(); -Finish the test and clean up the environment
    - scoreboard.finish();

  Called From:

    Test

hd_generator.sv

- function new(mailbox generator_to_driver, event driver_to_generator_event);
- task run(); - run the stimulus generator
    - txn = new;
    - txn.randomize()
    - generator_to_driver.put(txn);

  Called From:

env.run

hd_intf.sv - Defines the interface that connects to the DUT.

hd_monitor.sv - Implements a monitor that gather results as they come out of the DUT.

- class hd_monitor_cbs; - defines a callback hook to be used to pass result transactions.
    - virtual task post_monitor(hd_result_txn txn); - Callback hook to pass result transactions.

      Called From:

        hd_monitor.run
- class hd_monitor; - defines the class that will monitor the results from the DUT and send it using the callback class.
    - function new(virtual hd_intf intf); - Create a new instance of the class
    - task run(); - starts the monitor
        - mointor_hd();
        - cbs_list[i].post_monitor(txn);

      Called From:

        env.run
    - task mointor_hd(); - listen to signal level changes to detect a new output result.

hd_scoreboard.sv - Implements a scoreboard and a reference model

- class hd_scoreboard;
    - function new(); - create a new instance of the scoreboard

        Called From:

          env.run
    - function void save_current_txn(hd_txn txn); - Save in flight transaction for later checking

        Called From:

          driver_scb_cbs.save_current_txn
    - function void check_result(hd_result_txn result_txn); - Check the results that comes out of the DUV and make sure it's the expected result.
        - if(!result_txn.compare(predict_result(current_txn)))

Called From:

monitor_scb_cbs.post_monitor

- function void finish(); - End the scoreboard and make sure there is not a transaction in flight.

  Called From:

  Test

- protected function hd_result_txn predict_result(hd_txn txn); - Implements a reference model that produce the predicted result that DUV should produce.
  - predicted_txn = new();

hd_test.sv - Implements a test program and environment.

hd_top.sv - the top level module, connects the interface and starts the test.

Transaction.sv - Defines a stimulus transaction that goes into the DUT.

- virtual class transaction; - Defines the transaction class
  - function new(); - create a new transaction instance
  - pure virtual function bit compare(transaction to); - Deep compares this transaction to another instance.
  - pure virtual function transaction copy(transaction to=null); - Deep copies this class instance and return its handle
  - pure virtual function void display(string prefix=""); - Display the transaction fields

    Called from:

    transaction

- class hd_txn extends transaction; - Defines input stimulus transaction
  - function new();
    - super.new();

      Called from:

      Used throughout the testbench.

      ---These functions overrides the super class and are similar.
  - virtual function bit compare(transaction to);
  - virtual function transaction copy(transaction to=null);
  - virtual function void display(string prefix="");
- class hd_result_txn extends transaction; - Defines the output transaction
  - function new();

- super.new();

  Called from:

  Used throughout the testbench.

  ---These functions overrides the super class and are similar.

- virtual function bit compare(transaction to);
- virtual function transaction copy(transaction to=null);
- virtual function void display(string prefix="");

Coverage Report

```
COVERGROUP COVERAGE:
----------------------------------------------------------------------------------------------------
Covergroup                                              Metric       Goal    Status

----------------------------------------------------------------------------------------------------
 TYPE /MIPS_top_sv_unit/MIPS_coverage/valid_ops        87.50%        100    Uncovered
    covered/total bins:                                    12         13
    missing/total bins:                                     1         13
    % Hit:                                             92.30%        100
    type_option.weight=1
    type_option.goal=100
    type_option.comment=
    type_option.strobe=0
    type_option.merge_instances=auto(1)
    Coverpoint valid_ops::R_type                      100.00%        100    Covered
        covered/total bins:

        % Hit:                                        100.00%        100
        type_option.weight=1
        type_option.goal=100
        type_option.comment=
        bin NOP                                             1          1    Covered
        bin ADD                                            40          1    Covered
        bin SUB                                            42          1    Covered
        bin AND                                            55          1    Covered
        bin OR                                             81          1    Covered
        bin XOR                                            45          1    Covered
        bin SL                                             45          1    Covered
        bin SR                                             44          1    Covered
        bin SRU                                            34          1    Covered
    Coverpoint valid_ops::I_type                       75.00%        100    Uncovered
```

```
        covered/total bins:                                3          4
        missing/total bins:                                1          4
        % Hit:                                         75.00%        100
        type_option.weight=1
        type_option.goal=100
        type_option.comment=
        bin ADDI                                          45          1    Covered
        bin LD                                            21          1    Covered
        bin ST                                            21          1    Covered
        bin BZ                                             0          1    ZERO
TYPE /MIPS_top_sv_unit/MIPS_coverage/registers         72.48%        100    Uncovered
   covered/total bins:                                    38        100
   missing/total bins:                                    62        100
   % Hit:                                             38.00%        100
   type_option.weight=1
   type_option.goal=100
   type_option.comment=
   type_option.strobe=0
   type_option.merge_instances=auto(1)
   Coverpoint registers::destination                  100.00%        100    Covered
       covered/total bins:                                3          3
       missing/total bins:                                0          3
       % Hit:                                       100.00%        100
       type_option.weight=1
       type_option.goal=100
       type_option.comment=
       bin zeros                                         22          1    Covered
       bin ones                                          34          1    Covered
       bin others                                       439          1    Covered
   Coverpoint registers::source1                      100.00%        100    Covered
       covered/total bins:                                3          3
       missing/total bins:                                0          3
       % Hit:                                       100.00%        100
       type_option.weight=1
       type_option.goal=100
       type_option.comment=
       bin zeros                                         22          1    Covered
       bin ones                                          48          1    Covered
       bin others                                       425          1    Covered
   Coverpoint registers::source2                      100.00%        100    Covered
       covered/total bins:                                3          3
       missing/total bins:                                0          3
       % Hit:                                       100.00%        100
       type_option.weight=1
       type_option.goal=100
```

type_option.comment=			
bin zeros	22	1	Covered
bin ones	57	1	Covered
bin others	416	1	Covered
Coverpoint registers::offset_val	32.81%	100	Uncovered
covered/total bins:	21	64	
missing/total bins:	43	64	
% Hit:	32.81%	100	
type_option.weight=1			
type_option.goal=100			
type_option.comment=			
bin auto[0]	42	1	Covered
bin auto[1]	0	1	ZERO
bin auto[2]	20	1	Covered
bin auto[3]	0	1	ZERO
bin auto[4]	17	1	Covered
bin auto[5]	27	1	Covered
bin auto[6]	26	1	Covered
bin auto[7]	0	1	ZERO
bin auto[8]	0	1	ZERO
bin auto[9]	0	1	ZERO
bin auto[10]	0	1	ZERO
bin auto[11]	0	1	ZERO
bin auto[12]	0	1	ZERO
bin auto[13]	19	1	Covered
bin auto[14]	0	1	ZERO
bin auto[15]	0	1	ZERO
bin auto[16]	0	1	ZERO
bin auto[17]	0	1	ZERO
bin auto[18]	20	1	Covered
bin auto[19]	0	1	ZERO
bin auto[20]	17	1	Covered
bin auto[21]	21	1	Covered
bin auto[22]	0	1	ZERO
bin auto[23]	0	1	ZERO
bin auto[24]	23	1	Covered
bin auto[25]	0	1	ZERO
bin auto[26]	0	1	ZERO
bin auto[27]	21	1	Covered
bin auto[28]	52	1	Covered
bin auto[29]	0	1	ZERO
bin auto[30]	0	1	ZERO
bin auto[31]	0	1	ZERO
bin auto[32]	0	1	ZERO
bin auto[33]	19	1	Covered

bin auto[34]	28	1	Covered
bin auto[35]	22	1	Covered
bin auto[36]	0	1	ZERO
bin auto[37]	0	1	ZERO
bin auto[38]	0	1	ZERO
bin auto[39]	5	1	Covered
bin auto[40]	0	1	ZERO
bin auto[41]	0	1	ZERO
bin auto[42]	21	1	Covered
bin auto[43]	0	1	ZERO
bin auto[44]	0	1	ZERO
bin auto[45]	0	1	ZERO
bin auto[46]	0	1	ZERO
bin auto[47]	30	1	Covered
bin auto[48]	0	1	ZERO
bin auto[49]	24	1	Covered
bin auto[50]	0	1	ZERO
bin auto[51]	19	1	Covered
bin auto[52]	0	1	ZERO
bin auto[53]	0	1	ZERO
bin auto[54]	0	1	ZERO
bin auto[55]	0	1	ZERO
bin auto[56]	0	1	ZERO
bin auto[57]	0	1	ZERO
bin auto[58]	0	1	ZERO
bin auto[59]	22	1	Covered
bin auto[60]	0	1	ZERO
bin auto[61]	0	1	ZERO
bin auto[62]	0	1	ZERO
bin auto[63]	0	1	ZERO
Cross registers::reg_inputs	29.62%	100	Uncovered
covered/total bins:	8	27	
missing/total bins:	19	27	
% Hit:	29.62%	100	
type_option.weight=1			
type_option.goal=100			
type_option.comment=			
bin <zeros,zeros,zeros>	22	1	Covered
bin <others,ones,ones>	4	1	Covered
bin <ones,others,ones>	1	1	Covered
bin <others,others,ones>	52	1	Covered
bin <ones,ones,others>	3	1	Covered
bin <others,ones,others>	41	1	Covered
bin <ones,others,others>	30	1	Covered
bin <others,others,others>	342	1	Covered

```
        bin <ones,zeros,zeros>                          0          1    ZERO
        bin <others,zeros,zeros>                        0          1    ZERO
        bin <zeros,ones,zeros>                          0          1    ZERO
        bin <ones,ones,zeros>                           0          1    ZERO
        bin <others,ones,zeros>                         0          1    ZERO
        bin <zeros,others,zeros>                        0          1    ZERO
        bin <ones,others,zeros>                         0          1    ZERO
        bin <others,others,zeros>                       0          1    ZERO
        bin <zeros,zeros,ones>                          0          1    ZERO
        bin <ones,zeros,ones>                           0          1    ZERO
        bin <others,zeros,ones>                         0          1    ZERO
        bin <zeros,ones,ones>                           0          1    ZERO
        bin <ones,ones,ones>                            0          1    ZERO
        bin <zeros,others,ones>                         0          1    ZERO
        bin <zeros,zeros,others>                        0          1    ZERO
        bin <ones,zeros,others>                         0          1    ZERO
        bin <others,zeros,others>                       0          1    ZERO
        bin <zeros,ones,others>                         0          1    ZERO
        bin <zeros,others,others>                       0          1    ZERO

TOTAL COVERGROUP COVERAGE: 79.99%  COVERGROUP TYPES: 2
```