ECE 510
FINAL PROJECT REPORT
VERIFICATION OF PDP8 BASED DUT

Rohit Kulkarni Anuja Vaidya

Contents

Introduction:

Four principal modules in design:

- Clkgen_driver
- Instr_decode
- Instr_exec
- Memory_pdp

Goal: Build a verification environment around the design

Verification Strategy:

- Unit Level Testbenches for
  - IFD Unit
  - EXEC Unit
- Full chip validation for entire design

Design components

IFD Unit

Description:

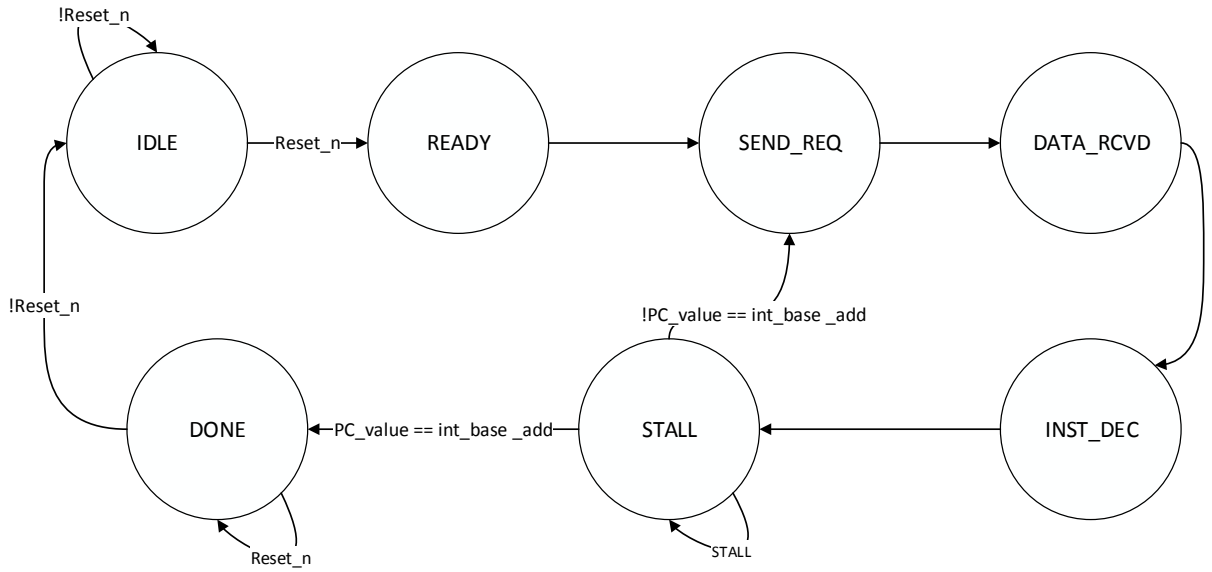


FIGURE 1: FSM DECODE UNIT

- Fetches instruction
- Decodes fetched instruction
- Supports decoding of
  - 6 memory reference instructions
    - Opcode 0 to 5
  - 22 microinstructions
    - Opcode 7 : Group 1 and Group 2

Execution Unit

Description:

- Executes instruction decoded by IFD unit.
- Accesses memory if instruction among opcodes 0 to 5.
- Reads operand from memory
- Writes result into memory
- Stalls IFD unit until current instruction completes execution.

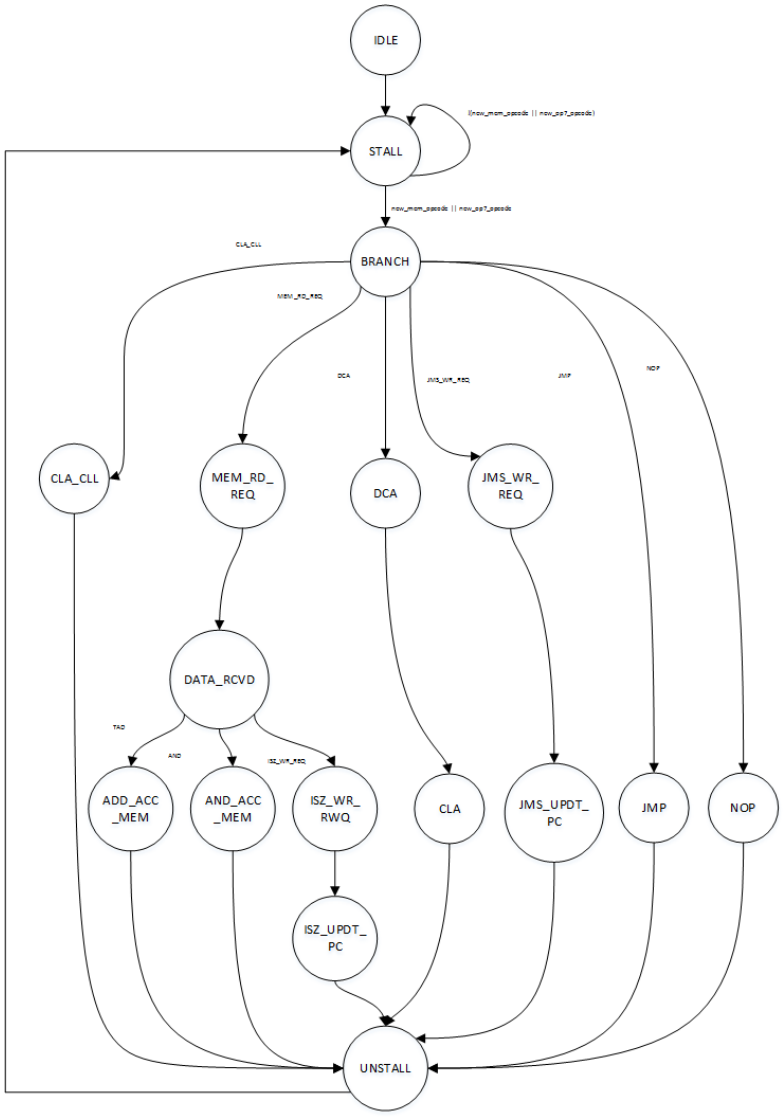
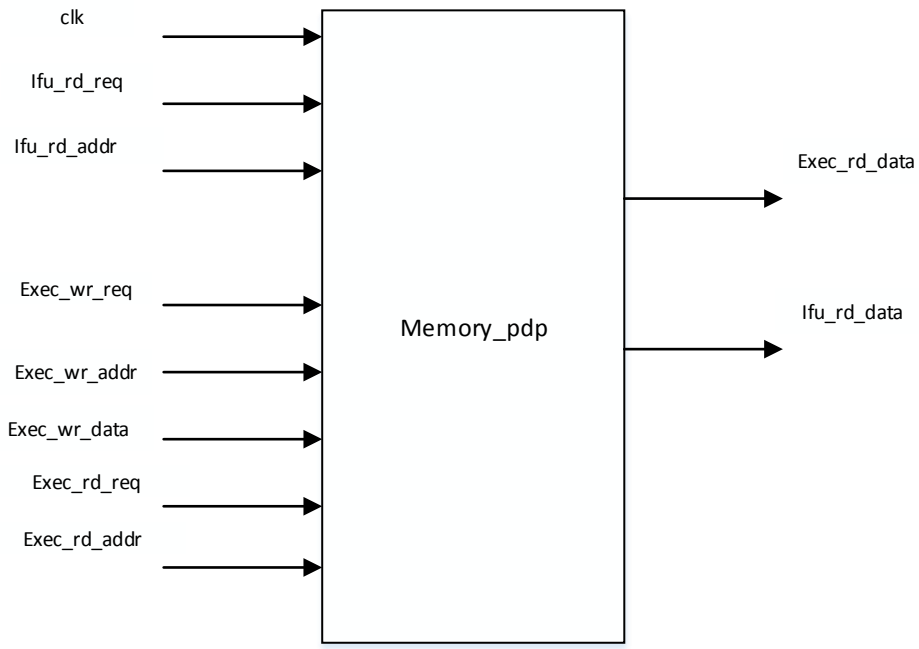


FIGURE 2:FSM: EXECUTION UNIT

Memory

Description:

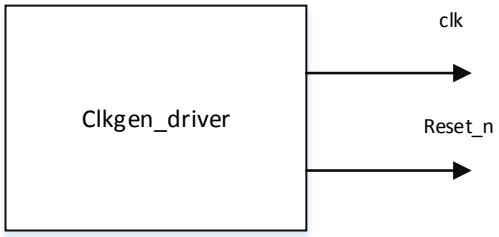
- The memory unit (4K words of 12bit each) is pre-loaded with data derived from a compiled PDP8 assembly language test. IFD unit fetches instruction from memory. EXEC unit fetches operand from memory and writes back the result.



Clock Gen

Description:

- This module provides clock and reset to all the modules in this design.



Unit Level Testbenches

Unit Level Testbench for IFD unit

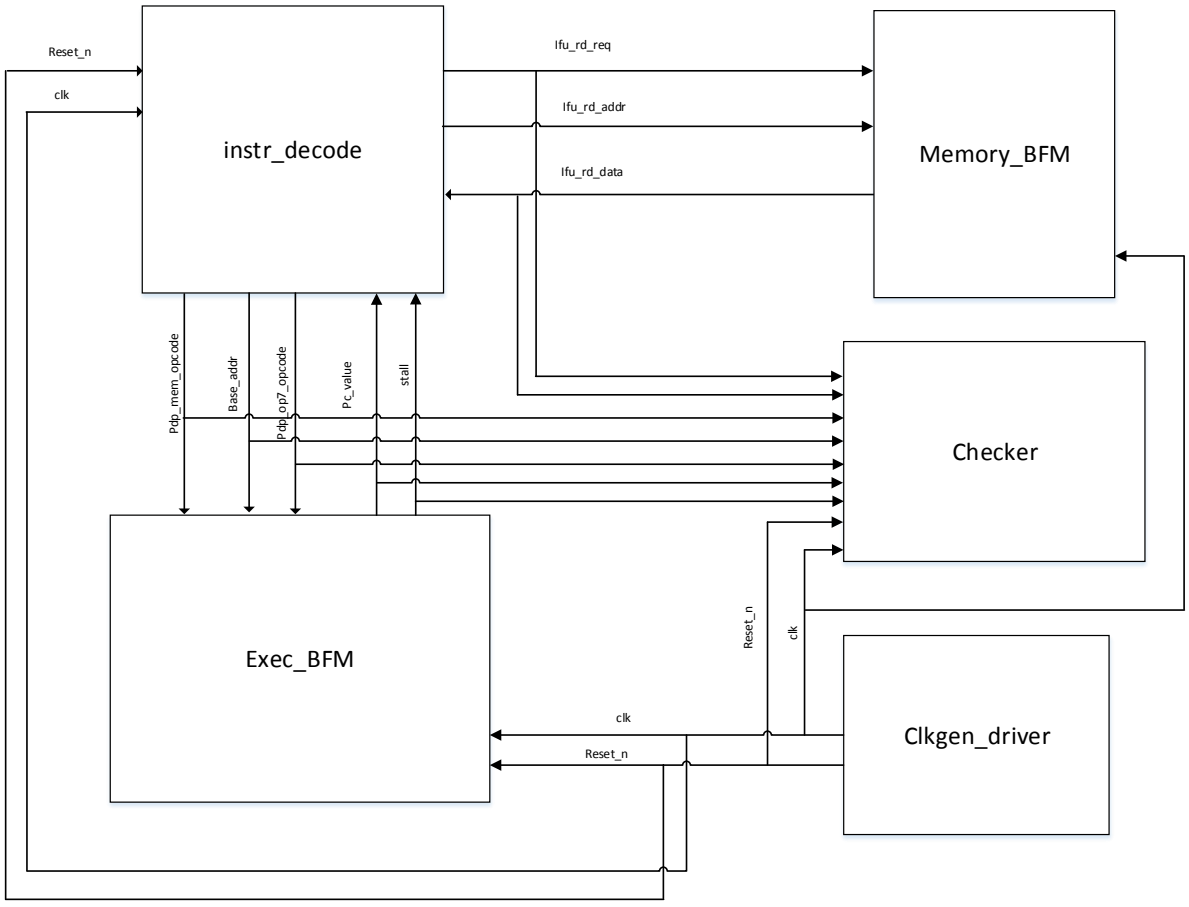


FIGURE 3: UNIT LEVEL TEST BENCH: IFD UNIT

Stimulus:

- **Stimulus:**
  - IFU_RD_DATA from MEMORY_BFM
  - STALL from EXEC_BFM
  - PC_VALUE from EXEC_BFM
  - RESET_N from CLK_GEN

- **Type of stimulus:**
  - **Purely random stimulus**
    - Stall:
      Driven by: EXEC_BFM
      Signal to stall instruction decoder.
      Stall is randomly generated to be asserted or deasserted.

    - PC value:
      Driven by: EXEC_BFM
      Current value of Program Counter.
      PC value is randomly generated 12 bit number

    - IFU_RD_DATA:
      Driven by: MEMORY_BFM
      Data (Instruction) read by the IFU unit from the memory.
      IFU_RD_DATA value is randomly generated 12 bit number by memory_bfm on a ifu_rd_req.

Note: Some deterministic stimulus is added in the stimulus generator to obtain hard to achieve cases which could not be generated randomly even after several million simulation cycles in order to gain maximum coverage.

Checks to perform

- **Type of checking:**
  On-the-fly : Primarily, after decoding every opcode.

- **Checks:**
  - Check if opcode sent to EXEC is correctly decoded
  - Check if address sent to EXEC is correctly decoded
  - When STALL is asserted, IFD should not fetch the next instruction and no new requests sent to EXEC unit
  - If reset_n asserted at any point, then all outputs are cleared within 1 clock cycle
  - On asserting reset, next instruction should be fetched from o200
  - If current state = STALL and PC = 0200, then next state = DONE
  - No instructions are fetched after going into DONE state
  - Check timing requirements such that the FSM stays in the particular state only for 1 cycle.
  - Check whether illegal opcodes are decoded as NOPs
  - If the STALL = 0 and base address is not reached then the FSM should jump into send_req state or if STALL = 1 and base address has been reached then the FSM should jump to DONE state.
  - On reset internal registers should be reset and the FSM should be in the IDLE state.

Coverage

We ran code coverage mechanism on the IFU unit in this unit level testbench.

Following report summarizes the coverage:

Notes:

1. Toggle coverage is slightly low, but the principal reason being `base_address` is constant equal to 12'o200 and hence no toggles will be seen on it.
2. If we exclude this, we get 92% toggle coverage.
3. Coverage is calculated by using the code coverage utility tool supported by Questa Sim.
4. A README document will summarize the commands required for running a simulation with coverage in case one has to reproduce results.
5. As promised, we have 100% FSM state coverage.

IFU UNIT LEVEL TESTBENCH COVERAGE

Coverage Summary By Instance:

Scope	TOTAL	Statement	Branch	Toggle	FSM State	FSM Trans	Assertion
TOTAL	82.60%	97.05%	95.34%	76.87%	100.00%	62.50%	62.50%
instr_decode	87.25%	96.77%	94.11%	76.87%	100.00%	62.50%	--
IFD_CHECKER	83.50%	97.50%	97.14%	76.87%	--	--	62.50%

Local Instance Coverage Details:

Total Coverage:					81.35%	87.25%
Coverage Type	Bins	Hits	Misses	Weight	% Hit	Coverage
Statements	62	60	2	1	96.77%	96.77%
Branches	51	48	3	1	94.11%	94.11%
Toggles	320	246	74	1	76.87%	76.87%
FSMs	23	17	6	1	73.91%	81.25%
States	7	7	0	1	100.00%	100.00%
Transitions	16	10	6	1	62.50%	62.50%

Recursive Hierarchical Coverage Details:

Total Coverage:					81.03%	82.60%
Coverage Type	Bins	Hits	Misses	Weight	% Hit	Coverage
Statements	102	99	3	1	97.05%	97.05%
Branches	86	82	4	1	95.34%	95.34%
Toggles	614	472	142	1	76.87%	76.87%
FSMs	23	17	6	1	73.91%	81.25%
States	7	7	0	1	100.00%	100.00%
Transitions	16	10	6	1	62.50%	62.50%
Assertions	8	5	3	1	62.50%	62.50%

Unit Level Testbench for EXEC unit

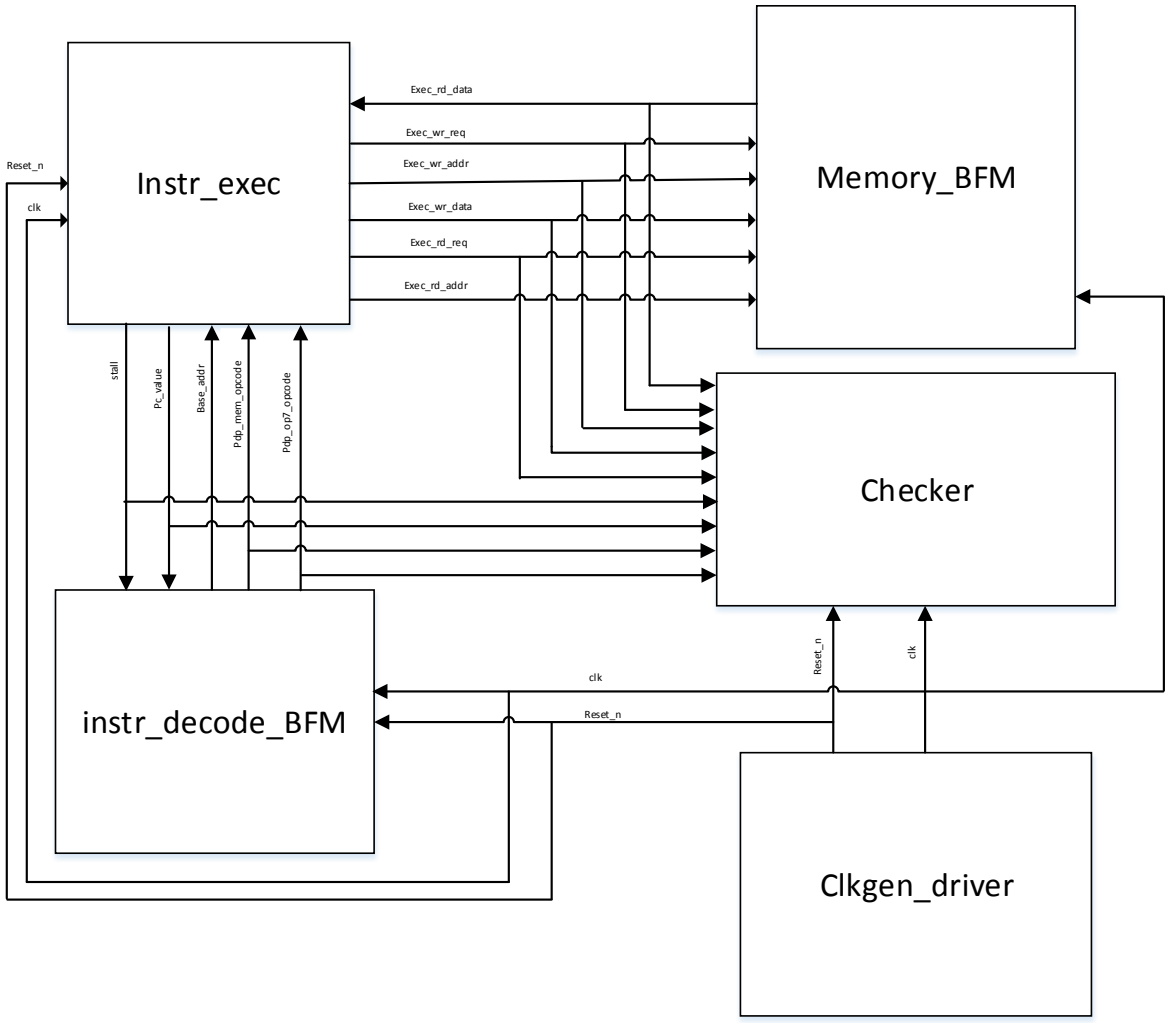


FIGURE 4: UNIT LEVEL TEST BENCH: EXECUTION UNIT

Stimulus

- **Stimulus:**

– PDP_MEM_OPCODE from INSTR_DECODE_BFM

– PDP_OP7_OPCODE from INSTR_DECODE_BFM

– EXEC_RD_DATA from MEMORY_BFM

– RESET_N from CLK_GEN

- **Type of stimulus:**
  - **Purely random stimulus**
    - PDP_MEM_OPCODE:
      Driven by: INSTR_DECODE_BFM
      Random pdp_mem_opcode is driven by the INSTR_DECODE_BFM.
      A random number is generated for pdp_mem_opcode which is 18 bit packed struct.

    - PDP_OP7_OPCODE:
      Driven by: INSTR_DECODE_BFM
      Random pdp_op7_opcode is driven by the INSTR_DECODE_BFM.
      A random number is generated for pdp_op7_opcode which is a 22 bit packed struct.

    - EXEC_RD_DATA:
      Driven by: MEMORY_BFM
      Data (operand) read by the EXEC unit from the memory.
      EXEC_RD_DATA value is randomly generated 12 bit number by memory_bfm on a exec_rd_req.

Note: Some deterministic stimulus is added in the stimulus generator to obtain hard to achieve cases which could not be generated randomly even after several million simulation cycles in order to gain maximum coverage.

Checks to perform

- **Type of checking:**
  On-the-fly : Primarily, after every instruction retire

- **Method of checking:**
  - Computes the golden result to be checked after every instruction retire.
  - By retire, we mean the point when instruction has written its result and a new instruction is fetched.
  - We use the State [UNSTALL] to detect instruction retire.
  - Following golden results are generated to be compared with:
    - 1: **Golden accumulator** contents
    - 2: **Golden link** bit contents
    - 3: **Golden PC** contents
    - 4: **Golden result** written to memory

  - Check if Accumulator contents are correct:

    For all instructions that require writing the result into Accumulator

    1. AND : **C(AC)** <- C(AC) AND C(EAddr)
    2. TAD : **C(AC)** <- C(AC) + C(EAddr)
    3. DCA : **C(AC)** <- 0
    4. CLA_CLL: **C(AC)** <- 0
    5. Other instructions should leave **C(AC)** as they are

  - Check if link bit contents are correct:

    1. TAD : If carry out then complement **Link**
    2. CLA_CLL: **Link** <- 0
    3. No other instruction should modify **link** bit

  - Check if PC is computed correctly

    1. ISZ : **C(PC)** <- C(PC) + 2
    2. JMS : **C(PC)** <- EAddr + 1
    3. JMP : **C(PC)** <- EAddr
    4. All other instructions: **C(PC)** <- C(PC) + 1

- Check if result is computed correctly and written to memory
    1. ISZ : **C(EAddr)** <- C(EAddr) + 1
    2. DCA : **C(EAddr)** <- C(AC)
    3. JMS : **C(EAddr)** <- C(PC)
    4. No other instruction should modify **memory**

Checks to be performed for each state transition:

State	Instruction response check	State transition	Number of cycles
CLA_CLL	Clears Acc and link register	Unstall	1
Mem_RD_REQ	Gets the address from the instruction and read request is sent to that ADDRESS	Data_rcvd	1
Data_rcvd	Data received from the memory and latched. Present value of ACC and linker are stored in a temp reg	Jumps to different states based on the instructions/opcodes	1
ADD_Acc_mem	Data from the memory location from the instruction is added with the accumulator. Carry is obtained in the link which is complemented. TAD is done here	Unstall	1
AND_Acc_mem	Data from the memory is ANDed with the data in the accumulator, AND operation is done here.	Unstall	1
ISZ_Wr_Req	Data from memory is incremented. Write add is calculated here	ISZ_UPDT_PC	1
ISZ_UPDT_PC	PC is incremented here conditionally ,ISZ is done here	Unstall	1
DCA	Write request is asserted here and it gets the address location from the opcode. Write data is the data in the Acc	CLA	1
CLA	ACC is cleared here	Unstall	1
JMS_WR_REQ	Write request is asserted here and it gets the address	JMS_UPDT_PC	1

	location from the opcode. Write data is the PC value		
JMS_UPDT_PC	PC is updated with the value from the PC and JMS is done here	Unstall	1
JMP	PC is updated with the value from the PC and JMP is done here	Unstall	1
NOP	-	Unstall	1
Unstall	Stall is cleared Write request is cleared and PC is incremented internally	Stall	1

Coverage:

We ran code coverage mechanism on the EXEC unit in this unit level testbench.

Following report summarizes the coverage:

Notes:

1. Coverage is calculated by using the code coverage utility tool supported by Questa Sim.
2. A README document will summarize the commands required for running a simulation with coverage in case one has to reproduce results.
3. As promised, we have 100% FSM state coverage.

EXEC UNIT LEVEL TESTBENCH COVERAGE

Scope	TOTAL	Statement	Branch	FEC Expression	FEC Condition	Toggle	FSM State	FSM Trans	Assertion
TOTAL	77.36%	100.00%	96.29%	28.57%	100.00%	60.76%	100.00%	61.90%	75.00%
instr_exec	78.96%	100.00%	94.44%	28.57%	100.00%	69.80%	100.00%	61.90%	--
CHKR_INST	81.17%	100.00%	100.00%	--	--	49.69%	--	--	75.00%

Local Instance Coverage Details:

Total Coverage:					73.24%	78.96%
Coverage Type	Bins	Hits	Misses	Weight	% Hit	Coverage
Statements	66	66	0	1	100.00%	100.00%
Branches	36	34	2	1	94.44%	94.44%
Expressions	28	8	20	1	28.57%	28.57%
UDP	--	--	--	--	--	Excluded
FEC	28	8	20	1	28.57%	28.57%
Conditions	5	5	0	1	100.00%	100.00%
UDP	--	--	--	--	--	Excluded
FEC	5	5	0	1	100.00%	100.00%
Toggles	404	282	122	1	69.80%	69.80%
FSMs	59	43	16	1	72.88%	80.95%
States	17	17	0	1	100.00%	100.00%
Transitions	42	26	16	1	61.90%	61.90%

Recursive Hierarchical Coverage Details:

Total Coverage:					66.70%	77.36%
Coverage Type	Bins	Hits	Misses	Weight	% Hit	Coverage
Statements	98	98	0	1	100.00%	100.00%
Branches	54	52	2	1	96.29%	96.29%
Expressions	28	8	20	1	28.57%	28.57%
UDP	--	--	--	--	--	Excluded
FEC	28	8	20	1	28.57%	28.57%
Conditions	5	5	0	1	100.00%	100.00%
UDP	--	--	--	--	--	Excluded
FEC	5	5	0	1	100.00%	100.00%
Toggles	734	446	288	1	60.76%	60.76%
FSMs	59	43	16	1	72.88%	80.95%
States	17	17	0	1	100.00%	100.00%
Transitions	42	26	16	1	61.90%	61.90%
Assertions	4	3	1	1	75.00%	75.00%

Chip level testbench

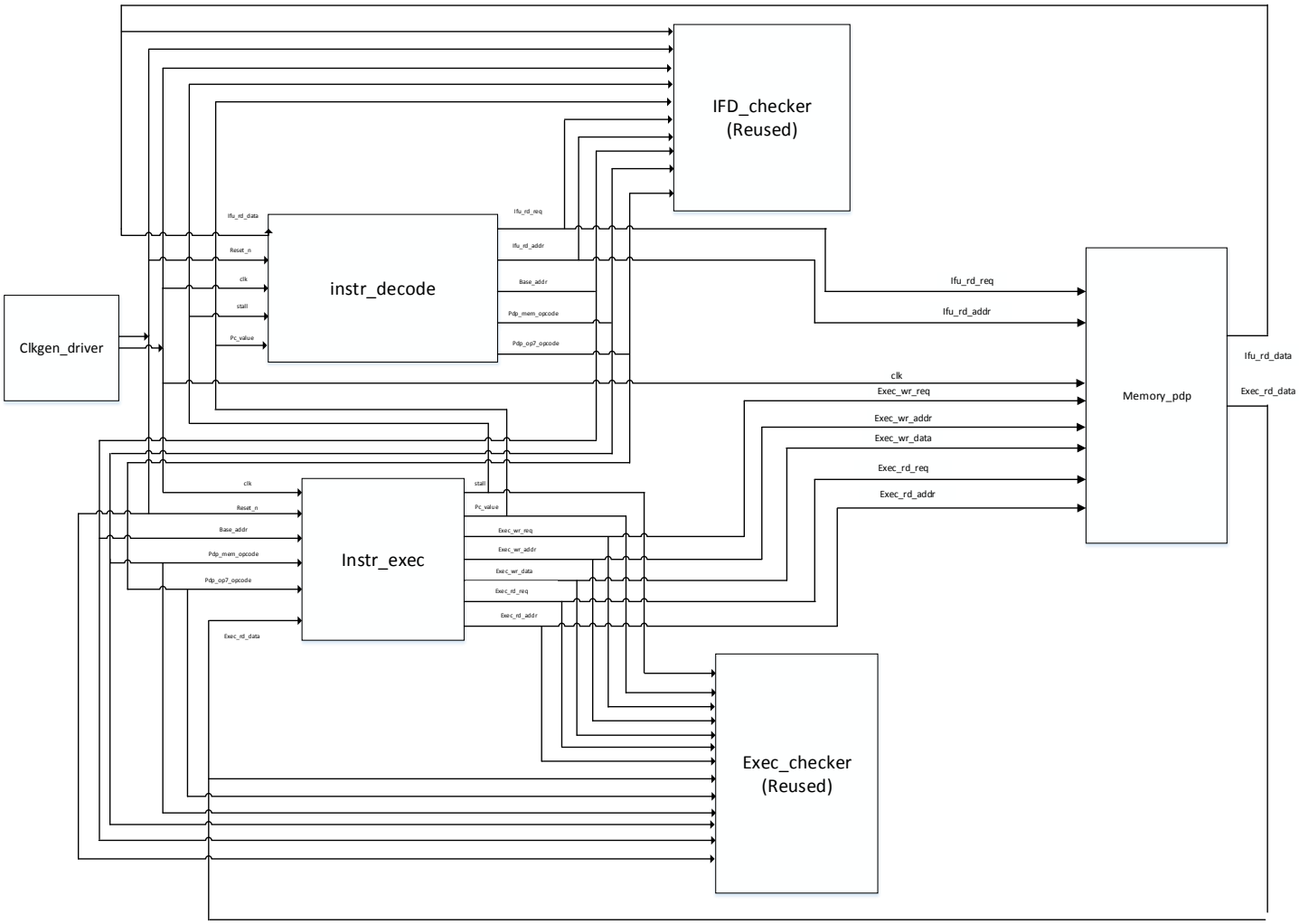


FIGURE 5: FULL CHIP TEST BENCH

Stimulus:

- **Stimulus:**
- Stimulus: Assembly code loaded in memory
- Stimulus should cover all possible opcodes
- We cannot have illegal opcodes assembled and hence we dropped this possibility
- Stimulus covered corner cases like:
  - TAD : Producing carry
  - TAD : Not Producing carry
  - ISZ : Skipping on zero
  - ISZ : Not Skipping on zero
  - AND : Toggling all 12 bits

- **Type of stimulus:**
- Deterministic:
  Assembly code loaded in memory

- Random:
  PERL script to generate random test.mem files
  A seed is provided to generate reproducible random memory files to be loaded in the PDP memory.

Checks to perform:

- Both the unit level checkers are reused :
  1. IFD checker
  2. EXEC checker
- All checks performed by the unit level checkers are performed at the full chip.

Coverage:

We ran code coverage mechanism on the IFU unit in this full chip level testbench.

Following report summarizes the coverage:

Notes:

1. Coverage is calculated by using the code coverage utility tool supported by Questa Sim.
2. A README document will summarize the commands required for running a simulation with coverage in case one has to reproduce results.
3. Some of the ASM files could not be assembled and resulted in getting lower coverage at the full chip level testbench.
4. It is equally difficult to generate stimulus at full chip level which can give us better or comparable coverage to unit level testbench.

IFU FULL CHIP TESTBENCH COVERAGE

Scope	TOTAL	Statement	Branch	Toggle	FSM State	FSM Trans	Assertion
TOTAL	68.69%	74.28%	70.45%	47.49%	100.00%	62.50%	70.00%
instr_decode	69.11%	77.41%	72.54%	45.25%	100.00%	62.50%	--
ifd_checker	64.40%	69.76%	67.56%	50.26%	--	--	70.00%

Local Instance Coverage Details:

Total Coverage:					52.00%	69.11%
Coverage Type	Bins	Hits	Misses	Weight	% Hit	Coverage
Statements	62	48	14	1	77.41%	77.41%
Branches	51	37	14	1	72.54%	72.54%
Toggles	464	210	254	1	45.25%	45.25%
FSMs	23	17	6	1	73.91%	81.25%
States	7	7	0	1	100.00%	100.00%
Transitions	16	10	6	1	62.50%	62.50%

Recursive Hierarchical Coverage Details:

Total Coverage:					52.81%	68.69%
Coverage Type	Bins	Hits	Misses	Weight	% Hit	Coverage
Statements	105	78	27	1	74.28%	74.28%
Branches	88	62	26	1	70.45%	70.45%
Toggles	838	398	440	1	47.49%	47.49%
FSMs	23	17	6	1	73.91%	81.25%
States	7	7	0	1	100.00%	100.00%
Transitions	16	10	6	1	62.50%	62.50%
Assertions	10	7	3	1	70.00%	70.00%

We ran code coverage mechanism on the EXEC unit in this full chip level testbench.

Following report summarizes the coverage:

Notes:

1. Coverage is calculated by using the code coverage utility tool supported by Questa Sim.
2. A README document will summarize the commands required for running a simulation with coverage in case one has to reproduce results.
3. Some of the ASM files could not be assembled and resulted in getting lower coverage at the full chip level testbench.
4. It is equally difficult to generate stimulus at full chip level which can give us better or comparable coverage to unit level testbench.

EXEC FULL CHIP TESTBENCH COVERAGE

Scope	TOTAL	Statement	Branch	FEC Expression	FEC Condition	Toggle	FSM State	FSM Trans	Assertion
TOTAL	84.33%	100.00%	96.42%	67.85%	100.00%	65.11%	100.00%	61.90%	80.00%
instr_exec	85.60%	100.00%	94.44%	67.85%	100.00%	70.37%	100.00%	61.90%	--
exec_checker	84.61%	100.00%	100.00%	--	--	58.47%	--	--	80.00%

Local Instance Coverage Details:

Total Coverage:					75.23%	85.60%
Coverage Type	Bins	Hits	Misses	Weight	% Hit	Coverage
Statements	66	66	0	1	100.00%	100.00%
Branches	36	34	2	1	94.44%	94.44%
Expressions	28	19	9	1	67.85%	67.85%
UDP	--	--	--	--	--	Excluded
FEC	28	19	9	1	67.85%	67.85%
Conditions	5	5	0	1	100.00%	100.00%
UDP	--	--	--	--	--	Excluded
FEC	5	5	0	1	100.00%	100.00%
Toggles	432	304	128	1	70.37%	70.37%
FSMs	59	43	16	1	72.88%	80.95%
States	17	17	0	1	100.00%	100.00%
Transitions	42	26	16	1	61.90%	61.90%

Recursive Hierarchical Coverage Details:

Total Coverage:					71.01%	84.33%
Coverage Type	Bins	Hits	Misses	Weight	% Hit	Coverage
Statements	101	101	0	1	100.00%	100.00%
Branches	56	54	2	1	96.42%	96.42%
Expressions	28	19	9	1	67.85%	67.85%
UDP	--	--	--	--	--	Excluded
FEC	28	19	9	1	67.85%	67.85%
Conditions	5	5	0	1	100.00%	100.00%
UDP	--	--	--	--	--	Excluded
FEC	5	5	0	1	100.00%	100.00%
Toggles	774	504	270	1	65.11%	65.11%
FSMs	59	43	16	1	72.88%	80.95%
States	17	17	0	1	100.00%	100.00%
Transitions	42	26	16	1	61.90%	61.90%
Assertions	5	4	1	1	80.00%	80.00%

Bug Report

With the robust verification environment we built around the DUT, we were in a good state to catch any bugs that crept in the design.

We also did a code review to spot any obvious bugs lurking in the design.

1. **Bug in computing Link bit**
   Type of bug: Implementation bug by designer
   Description:
   According to the specification, for a TAD operation,

   C(AC) <- C(AC) + C(EAddr) and if carry out then complement Link.

   Our checker found out that link bit does not complement when the carry is 1. Instead, the link bit is a function of the previous carry and not the current carry obtained by the instruction retired.

2. **Initial state of Accumulator and Link bit on reset are not defined**
   Type of bug: Ambiguity in architectural specification
   Description:
   The specification does not define the initial state of accumulator and link bit.

   On starting the simulation, both (Acc and Link) are initialized to 'X.

   We feel this is an ambiguous specification and should be resolved by talking to the architects.

   Remedy:

   Start all tests by clearing accumulator and link using the CLA_CLL instruction to get Accumulator and Link bit to a predictable state.

3. **Inherent priority in decoding opcodes not mentioned in the specification**
   Type of bug: Ambiguity in architectural specification
   Description:
   We see that there is an inherent priority in decoding opcodes. The case statement in IFU is where it is implemented. This is not mentioned anywhere in the design and should be reviewed with the architects.

4. **Ambiguity in specification for ISZ**

   Type of bug: Ambiguity in architectural specification
   Description:
   Unsure whether to take the old data or the new data for checking.

5. **INSTR_DEC does not check for stall on coming out of reset**

   Type of bug: Ambiguity in architectural specification / implementation
   Description:
   Should there be support for a case in which instruction fetch and decode comes out of reset, but exec unit asserts STALL preventing IFU to not fetch the first instruction? Need to clarify with architects

6. **I and M bits in the instruction don't have any significance:**
   Type of bug: Incorrect implementation
   Description:

   I and M bits in the instruction are just a part of the address instead of their special significance. The address is 9 bits [3: 11] but the address lines are 12 bits and the upper 3 bits are always 0.

   An instruction having the format

   TAD I Z <LABEL>

   would not execute as expected since we do not support decoding of I and M bits in the instruction.

7. **Code review :**
   Type of bug: Incorrect use of constructs
   Description:
   In the FSM implementation of IFD and EXEC, we found out that non-blocking assignments are used inside `always_comb` procedural blocks.
   Since code inside `always_comb` is meant to be purely combinational, it is highly recommended to use blocking assignments to describe combinational behaviour. Had this guideline been followed, the link bit bug (bug no.1) would not have crept in the design.

Individual Contribution

Anuja	1: Owning Full Chip testbench 2: Understanding and writing assembly test case stimulus for full chip environment 3: Monitoring coverage for full chip environment 4: Adding some unit level checks 5: Documenting results
Rohit	1: Owning unit level testbenches 2: Developing reusable unit level checkers and stimuli 3: Script to generate random memory files to be used as random stimulus for full chip testbench 4: Monitoring unit level coverage 5: Documenting results and report

Retrospect

- We learnt a lot while working on this project.
- We feel we did our best, but we would have liked to do better given more time. Specifically:
    - Add a golden memory model to the verification environment
    - Do end-of-test memory check with the golden model
    - Develop more robust checks
    - Get to 100% coverage on all 3 testbenches.
    - Functional coverage for some corner cases