Verification Plan Proposal ECE 510   
FINAL PROJECT

Rohit Kulkarni Anuja Vaidya

Contents

[Introduction: 2](#_Toc419875156)

[Design components: 2](#_Toc419875157)

[IFD Unit 2](#_Toc419875158)

[Description: 2](#_Toc419875159)

[Execution Unit 3](#_Toc419875160)

[Description: 3](#_Toc419875161)

[Memory 5](#_Toc419875162)

[Description: 5](#_Toc419875163)

[Clock Gen 5](#_Toc419875164)

[Description: 5](#_Toc419875165)

[Unit Level Testbenches 6](#_Toc419875166)

[Unit Level Testbench for IFD unit 6](#_Toc419875167)

[Stimulus: 7](#_Toc419875168)

[Checks to perform: 7](#_Toc419875169)

[Assertions: 7](#_Toc419875170)

[Coverage: 7](#_Toc419875171)

[Unit Level Testbench for EXEC unit 8](#_Toc419875172)

[Stimulus: 8](#_Toc419875173)

[Checks to perform: 9](#_Toc419875174)

[Assertions: 10](#_Toc419875175)

[Coverage: 10](#_Toc419875176)

[Chip level testbench 11](#_Toc419875177)

[Stimulus: 11](#_Toc419875178)

[Checks to perform: 11](#_Toc419875179)

[Assertions: 11](#_Toc419875180)

[Coverage: 11](#_Toc419875181)

[Figure 1: FSM decode unit 2](#_Toc419875182)

[Figure 2:Fsm: Execution unit 4](#_Toc419875183)

[Figure 3: Unit Level test Bench: IFD unit 6](#_Toc419875184)

[Figure 4: UNIT level test bench: Execution unit 8](#_Toc419875185)

[Figure 5: FULL chip test bench 11](#_Toc419875186)

# Introduction:

# Design components:

## IFD Unit

### Description:



Figure : FSM decode unit

## Execution Unit

### Description:



Figure :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 sends the instruction to execution unit (EXE) which processes the instruction and reads/writes from/to memory.



## Clock Gen

### Description:

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



# Unit Level Testbenches

## Unit Level Testbench for IFD unit



Figure : Unit Level test Bench: IFD unit

### Stimulus:

Stimulus to this block will mainly consist of different 28 different opcodes. Responses to each of the block should be observed. Additionally we will also have some test cases with illegal Opcodes to check how the FSM behaves and if it goes in the NOP state.

### Checks to perform:

* If reset\_n is de-asserted start\_address/base\_addess = o200
* Check timing requirements such that the FSM stays in the particular state only for 1 cycle.
* 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 and flag should be reset and the FSM should be in the IDLE state.
* To check if the decoding part is done correctly, i.e. data read from memory should be correctly decoded into one of the 7 compatible opcodes and the remaining should be treated as NOP.

### Assertions:

Assertions would be based on the following flags:

Reset\_n 🡪 to check if the base address is correctly loaded.

Reset\_n 🡪 to check if the internal registers are to the pre-defined reset values.

STALL 🡪 to decide the appropriate state change in the FSM to DONE.

### Coverage:

To ensure the full coverage we will need State coverage which will update us on which all states are covered and Arc coverage will show us which state transitions. Path coverage which will combine both the state and the arc coverage to give the entire coverage analysis.

## Unit Level Testbench for EXEC unit



Figure : UNIT level test bench: Execution unit

### Stimulus:

Different instructions which fall under this category should be provided as the input stimulus.

Different instructions are processed one at time from the Branch state. All the instructions in this task are to perform read/write operations from the memory.

False instructions will also be provided to check the response to the FSM.

In the section below we determine the relation between the instruction and what check should be performed for each one of them.

### Checks to perform:

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction** | **Instruction response check** | **State transition** | **Number of cycles** |
| CLA\_CLL | Clears Acc and linker 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 location from the opcode. Write data is the PC value | JMS\_UPDT\_PC | 1 |
| 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 |

### Assertions:

* To check if the address is valid for memory instructions.
* If in STALL state and there is a new instruction then STALL should be 1.
* If in BRANCH, PC must increment and the FSM should not be in this state for more than 1 cycle.
* On Reset\_n stay in IDLE or else in IDLE PC === base\_address.

### Coverage:

Since the structure of this module is similar to the decode logic we will use the state coverage to check if all the states and covered and then the Arc coverage to check if it makes the correct transition.

Finally the path coverage to ensure complete coverage analysis.

# Chip level testbench



Figure : FULL chip test bench

### Stimulus:

### Checks to perform:

### Assertions:

### Coverage: