ipPROCESS

**VERIFICATION PLAN**

32-bit $\mu$DLX Core Processor

Universidade Federal da Bahia

**Version: 1.0**

# GNU LGPL License

# Revision History

| Date | Description | Author(s) |
|---|---|---|
| 05/23/2014 | First Verification Plan version. | Victor Valente |
| 06/07/2014 | Adding information to Overview section. | Lauê Rami and Igo Amauri |

CONTENTS


1 **Introduction** **4**

1.1 Document Purpose . . . . . 4

1.2 Stakeholders . . . . . 4

1.3 Document Outline Description . . . . . 4

1.4 Acronyms and Abbreviations . . . . . 4

2 **DUT Overview** **5**

3 **Verification Environment** **9**

3.1 Design Under Test Interface (DUT_IF) . . . . . 9

3.1.1 Forwarding . . . . . 9

3.1.2 Branch Taken . . . . . 10

3.1.3 Load Hazard . . . . . 10

3.2 Monitor . . . . . 10

3.3 Golden Model . . . . . 10

3.4 Verification Environment Design Specification . . . . . 10

4 **Features List** **12**

5 **Test List** **13**

6 **Assertions** **18**

7 **Resource Requirements** **19**

8 **Schedule** **20**

# 1. Introduction

## 1.1. Document Purpose

The purpose of this document is define the verification plan of the uDLX Implementation. This document includes the verification environment used to perform the verification of the processor, beside the main characteristics of the design, the list of tests, list of assertions, and others.

## 1.2. Stakeholders

| Name | Roles/Responsibilities |
| --- | --- |
| Igo Amauri Luz | Verification Engineer |
| Lauê Rami Costa | Verification Engineer |

## 1.3. Document Outline Description

## 1.4. Acronyms and Abbreviations

| Acronym | Description |
| --- | --- |
| ASIC | Application Specific Integrated Circuit |
| DUT | Design Under Test |

## 2. DUT Overview

The uDLX is a simplified architecture of the DLX processor. The uDLX correspond to a RISC general propose microprocessor, memory access, arithmetic and logical, and branch instructions. The processor has 5 stages in a pipeline architecture that are : instruction fetch, instruction decode, execute, memory access, and write back.

The uDLX architecture interface is composed by the following components:

- uDLX 32-bit Core: The core four-deep pipeline processor.
- Memory Interface: Provides a middle layer between the core processor and the external memories. This interface also controls the bootloader process.
- SDRAM Controller: Provides the interface for controlling the external SDRAM.
- SRAM Controller: Provides the interface for controlling the external SRAM.

The Figure 1 shows the architecture interface.

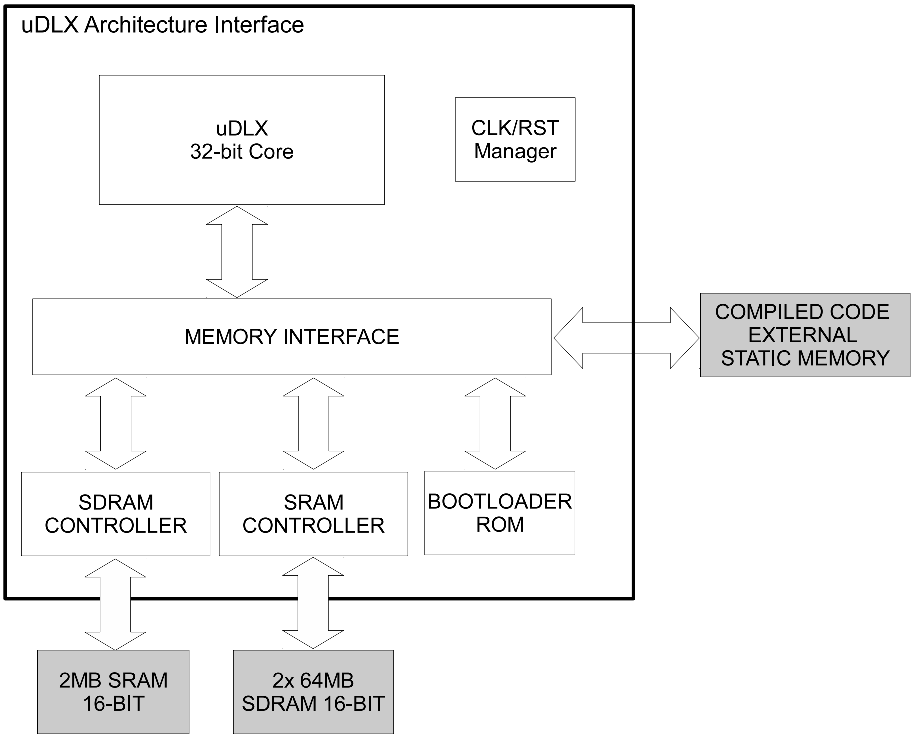


**Figure 1: $\mu$DLX Architecture Interface.**

A Harvard architecture is implemented using a SRAM with 16 bits of data width to access the instructions and a SDRAM to access data. In order to avoid stall due to time need to access the memories the processor and memories controllers work in different clock rate.

The main features of the uDLX are:

- uDLX with multiple stages
- Hazard handling

- SRAM memory access
- SDRAM memory access

The Figure 2 shows the Top Architecture of the $\mu$DLX Processor.

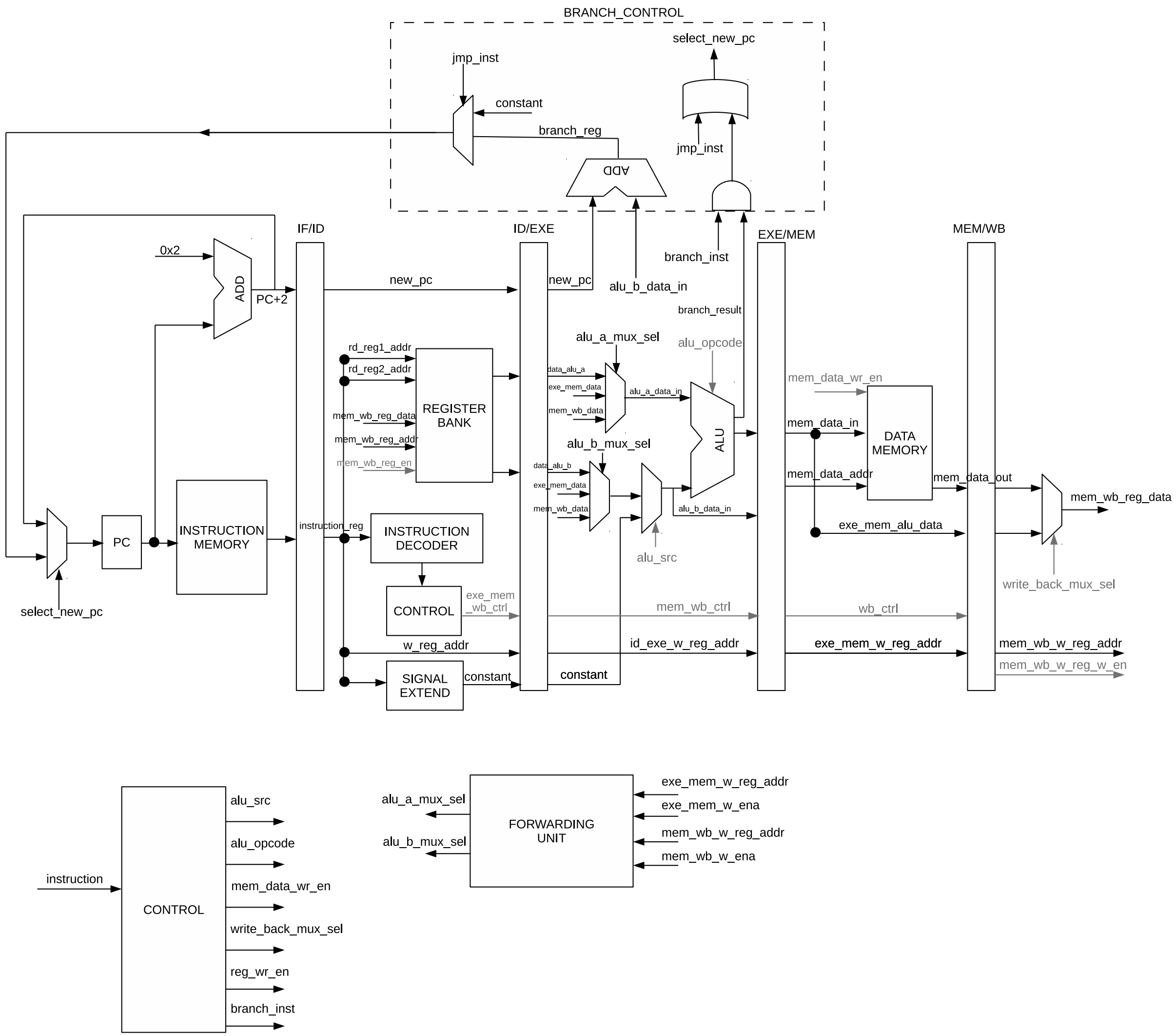


**Figure 2: $\mu$DLX top level architecture overview.**

In order to be the most close to the DLX and also be as compatible as possible to MIPS R2000 and R3000 architecture few instructions were added to the instruction set of this processor. The instruction set supported in the project are: I-type, R-type and J-type. The Figure 3 shows the types of instructions.

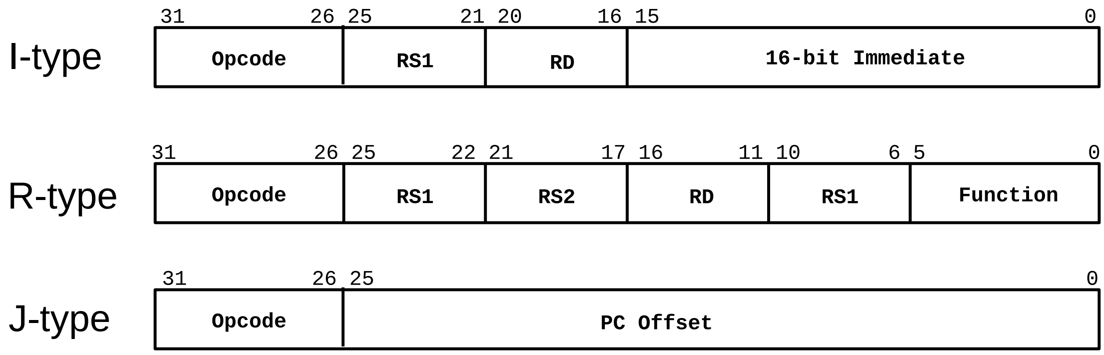


Figure 3: $\mu$DLX Instruction Set types definitions.

# 3. Verification Environment

The verification methodology is based in a simple testbench. Part of the verification will be done using waveform based verification. Some special situations will be verified using assertion based verification. The design under test (DUT) interface will be responsible to catch data from the uDLX and send to the monitor that will have all assertions. The following topics will describe the items that compose the Verification Plan. The Figure 4 shows the Verification Environment.

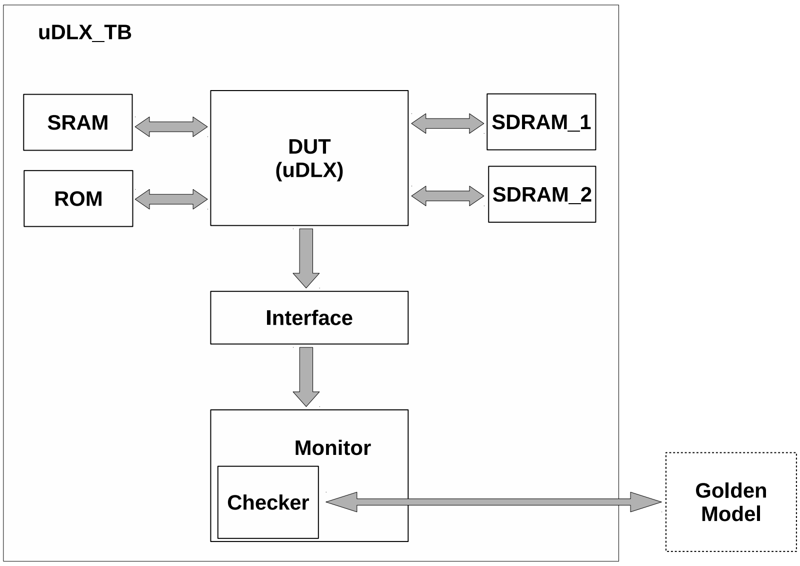


Figure 4: $\mu$DLX Verification Environment.

In the directory sim/tb/scripts have a script that can be used to execute the verification. This script have the name: run_udlx_verification_env.run

## 3.1. Design Under Test Interface (DUT_IF)

The Design Under Test Interface (DUT_IF) makes the interface between the monitor and the DUT. The DUT_IF is responsible to control the data that are transmitted between the Verification Environment and DUT, so it contains the instance of all DUT signals that will be used to perform the verification.

The DUT Interface, also, have all assertions that is implemented to ensure if the behavior of the DUT signals are correct. This interface is instantiated at the Top Level of Verification Environment and the signals are connected with the signals of the DUT.

Through the assertions, the verification ensure, mainly, that the Forwarding, Branch Taken and Load Hazard instructions are working as expected.

### 3.1.1. *Forwarding*

The forwarding operation can occur between the execution and different blocks. This assertion will be used to ensure the forwarding unit is activating the correct signal and doing what is expected.

#### 3.1.2. *Branch Taken*

As the uDLX developed work in the branch not taken method, the monitor needs to ensure that when a branch is taken all flush and new address will work correctly.

#### 3.1.3. *Load Hazard*

During a load hazard the uDLX stall and flush specific stages. The assertions will monitor this condition and give warnings for any unpredicted behavior.

### 3.2. Monitor

The monitor is responsible to observe the behavior of the DUT and, also, collect the output to verify if the instructions are working as expected.

The monitor observe the behavior of the control signals and, when necessary, capture the data that is stored in data memory and in registers.

When the software ends, the monitor identify it and call the Checker. The monitor identify as the end of software a consecutive sequence of six nop instruction. The Checker is responsible to execute the Golden Model with the same software that was used by DUT and compare the data stored in data memory and in registers. If any mismatch are found, the errors are reported.

The test that will be executed in the Golden Model, needs to be defined in sim/tb/defines.sv file. To run the test in DUT, the test needs to be changed in 'read_memh' in rtl/commom/rom.v file.

### 3.3. Golden Model

To ensure that the DLX Processor execute the instructions correctly, was created a Golden Model that simulate the same behavior of the DLX Processor.

The Golden Model was developed using C Language. It is able to execute all the instructions supported by DLX Processor, so the same instruction set that is used to simulate the DUT can be applied to Golden Model. This software stores the data that will be store in SDRAM Memories and in the Register Bank by DUT, so, these data are compared with the same data collected by Verification Environment.

### 3.4. Verification Environment Design Specification

| Component | Description |
|---|---|
| Document Name and Version | $\mu$DLX Verification Plan |
| Document Version and Date | Version 1.0, June 23, 2014 |
| Author(s) / Owner(s) | Lauê Rami / Igo Amauri |
| Verification Methodology | Top-Down |

continued on next page

| continued from previous page | |
|---|---|
| **Component** | **Description** |
| Verification Methods | Simulation and Formal Verification |
| Simulation Components | |
| Application | Cadence Incisive |
| Languages | System Verilog |
| Verification Environmentv | Custom testbench |
| Testbench files | At directory: sim/tb |
| Formal Verification Components | Cadence Encounter Conformal |
| Application | |
| Technologies | FPGA DE2-115 |

## 4. Features List

The features list, describes the features that are planned to be implemented.

| Feature Number | Feature Description | Priority |
|---|---|---|
| DLX_F1 | Instruction must activate the signals corresponding to the functionality | 10 |
| DLX_F2 | Forwarding Unit should identify and forward data correctly | 8 |
| DLX_F3 | Communication with SRAM must work properly | 9 |
| DLX_F4 | Read and write operation to the SDRAM | 9 |
| DLX_F5 | The registers of the Register Bank must be reading and writing | 10 |
| DLX_F6 | All interfaces protocols must work properly | 9 |

## 5. Test List

The test list define the tests that will be performed with the DUT to ensure that the behavior is correct. In addition, this tests list ensures achieving coverage levels.

| Test Number | Description | Method | Level | Features Verified | Priority | Owner | Completion |
|---|---|---|---|---|---|---|---|
| DLX_T1 | Execution of all instructions of the Arithmetic category. | Sim | Unit | DLX_F1, DLX_F5 | 5 | Igo, Lauê | 100% |
| DLX_T1_1 | Execution of addi instruction. All the registers are used to store the results of the operation | Sim | Unit | DLX_F1, DLX_F5 | 5 | Lauê | 100% |
| DLX_T1_2 | Execution of subi instruction. All the registers are used to store the results of the operation | Sim | Unit | DLX_F1, DLX_F5 | 5 | Lauê | 100% |
| DLX_T1_3 | Execution of add instruction. All the registers are used to store the results of the operation | Sim | Unit | DLX_F1, DLX_F5 | 5 | Igo | 100% |
| DLX_T1_4 | Execution of sub instruction. All the registers are used to store the results of the operation | Sim | Unit | DLX_F1, DLX_F5 | 5 | Igo | 100% |

continued on next page

continued from previous page

| Test Number | Description | Method | Level | Features Verified | Priority | Owner | Completion |
|---|---|---|---|---|---|---|---|
| DLX_T1_5 | Execution of not instruction. All the registers are used to store the results and a lot of combinations of sources registers are used to perform the operation | Sim | Unit | DLX_F1, DLX_F5 | 5 | Lauê | 100% |
| DLX_T2 | Execution of all instructions of the Data Transfer category. | Sim | Unit | DLX_F1, DLX_F5 | 5 | Igo, Lauê | 100% |
| DLX_T2_1 | Execution of sw instruction. Store determined value in all positions of memory | Sim | Unit | DLX_F1, DLX_F5 | 5 | Lauê | 100% |
| DLX_T2_2 | Execution of lw instruction. Load the values previously stored in all positions of memory | Sim | Unit | DLX_F1, DLX_F5 | 5 | Igo | 100% |
| DLX_T3 | Execution of all instructions of the Logical category. | Sim | Unit | DLX_F1, DLX_F5 | 5 | Igo, Lauê | 100% |
| DLX_T3_1 | Execution of andi instruction. All the registers are used to store the results of the operation | Sim | Unit | DLX_F1, DLX_F5 | 5 | Lauê | 100% |

continued on next page

continued from previous page

| Test Number | Description | Method | Level | Features Verified | Priority | Owner | Completion |
|---|---|---|---|---|---|---|---|
| DLX_T3_2 | Execution of ori instruction. All the registers are used to store the results of the operation | Sim | Unit | DLX_F1, DLX_F5 | 5 | Lauê | 100% |
| DLX_T3_3 | Execution of and instruction. All the registers are used to store the results of the operations | Sim | Unit | DLX_F1, DLX_F5 | 5 | Igo | 100% |
| DLX_T3_4 | Execution of or instruction. All the registers are used to store the results of the operations | Sim | Unit | DLX_F1, DLX_F5 | 5 | Igo | 100% |
| DLX_T4 | Execution of all instructions of the Conditional Branch category. | Sim | Unit | DLX_F1, DLX_F5 | 5 | Igo, Lauê | 100% |
| DLX_T4_1 | Execution of beqz instruction. Execute a software that have beqz instruction, and, at least, the condition is satisfied and not. | Sim | Unit | DLX_F1, DLX_F5 | 5 | Lauê | 100% |

continued on next page

**continued from previous page**

| Test Number | Description | Method | Level | Features Verified | Priority | Owner | Completion |
|---|---|---|---|---|---|---|---|
| DLX_T4_2 | Execution of bnez instruction. Execute a software that have bnez instruction, and, at least, the condition is satisfied and not. | Sim | Unit | DLX_F1, DLX_F5 | 5 | Igo | 100% |
| DLX_T5 | Execution of all instructions of the Unconditional Jump category. | Sim | Unit | DLX_F1 | 5 | Lauê | 100% |
| DLX_T5_1 | Execution of jr instruction. Execute a software that have jr instructions and use more than one register as source. | Sim | Unit | DLX_F1 | 5 | Lauê | 100% |
| DLX_T5_2 | Execution of jpc instruction. | Sim | Unit | DLX_F1 | 5 | Lauê | 100% |
| DLX_T6 | Test forwarding unit in all possible situations. Including the possibility of dealing with 3 consecutive RAW hazards. | Sim | Unit | DLX_F1, DLX_F2 | 5 | Igo | 100% |
| DLX_T7 | Test the SDRAM access | Assertion | Unit | DLX_F4 | 7 | Lauê | 100% |
| DLX_T8 | Test the SRAM access | Assertion | Unit | DLX_F3 | 9 | Lauê | 100% |
| DLX_T9 | Run programs with the complete architecture | Sim | Unit | DLX_F3, DLX_F4 | 8 | Lauê | 100% |

continued on next page

continued from previous page

| Test Number | Description | Method | Level | Features Verified | Priority | Owner | Completion |
|---|---|---|---|---|---|---|---|
| DLX_T10 | Test All interfaces protocols | Assertion | Unit | DLX_F6 | 8 | Igo | 100% |

## 6. Assertions

Assertion will be used to verify the instructions, specially the instructions that have stall, forwarding and flush.

| Number | Criteria | Status |
| --- | --- | --- |
| DLX_A1 | Assertion to the correct fecth_top operation. | Completed |
| DLX_A2 | Assertion to verify the decode operation. | Completed |
| DLX_A3 | Assertion for the execution block operation. | Completed |
| DLX_A4 | Assertion for the memory read and write back. | Completed |
| DLX_A5 | Assertion to forwarding operation. | Completed |
| DLX_A6 | Assertion to branches and jumps operations. | Completed |
| DLX_A7 | Assertion to verify interfaces protocols. | Completed |
| DLX_A8 | Assertion to verify interfaces with memories. | Completed |

## 7. Resource Requirements

| Resource | Quantity | Description | Start | Duration |
|---|---|---|---|---|
| **Engineering Resources** | | | | |
| Verification Engineer | 2 | 1 year experience | 06/25/14 | 10 days |
| **Compute Resources** | | | | |
| Computer | 1 | | 06/25/14 | 10 days |
| **Software Resources** | | | | |
| Candence Incisive | 1 | | 06/25/14 | 10 days |
| ALTERA Quartus | 1 | WEB Edition | 06/25/14 | 10 days |
| ALTERA ModelSIM | 1 | WEB Edition | 06/25/14 | 10 days |

# 8. Schedule

| Resource | Start | Duration | Action | Compute Resource |
|---|---|---|---|---|
| Igo, Lauê | 06/20/14 | 2 Days | Develop Verification Test Plan | N/A |
| Igo, Lauê | 06/22/14 | 1 Day | Review & Approve Test Plan | N/A |
| Igo, Lauê | 06/25/14 | 5 Days | Create Verification Environment | Server 1 |
| Igo, Lauê | 06/28/14 | 10 Days | Features F1,F2,F3, and F4 Verification | Server 1 |
| Igo, Lauê | 07/08/14 | 7 Days | Add assertion and code coverage | Server 1 |
| Igo, Lauê | 07/16/14 | 11 Days | Complete System simulation and functional verification | Server 1 |