REVISION		
REV	DESCRIPTION	DATE
1.1	Initial Revision	23.04.2006

DigART

DIGITAL CIRCUIT SIMULATOR PROJECT

Test Specification Report

Number of Pages								
Doc	Appendixes							
	A	B	C	D	E	F	G	H
12	–	–	–	–	–	–	–	–

STARSOFT
Middle East Technical University
06531 ODTÜ, Ankara-TÜRKIYE

Controlled Folder: TBD	**A4**				1

PREPARED BY			
NAME	**TITLE/ROLE**	**REV**	**SIGNATURE**
Elif SAYGI	Quality Engineer	1.1	
Halit DEVELIOGLU	Project Manager	1.1	
Serhat KOYULMUS	Software Developer	1.1	
Seniz SOZER	Configuration Manager	1.1	
Volkan ICEL	Software Developer	1.1	

CONTRIBUTIONS			
NAME	**TITLE/ROLE**	**REV**	**SIGNATURE**
Serkan BAL	Advisor	1.1	

CHECKED BY			
NAME	**TITLE/ROLE**	**REV**	**SIGNATURE**
Serkan BAL	Advisor	1.1	
Elif SAYGI	Quality Engineer	1.1	
Volkan ICEL	Software Developer	1.1	

TABLE OF CONTENTS

1. INTRODUCTION

1.1. GOALS AND OBJECTIVES

Our project DIGART is an implementation of a graphical tool which is used for editing digital electronics circuits and executing them.We have two main divisions in the project,

1 - Circuit Engine component where the design and editing of combinatorial and sequential digital circuits are handled, the inputs are taken.

2 - Simulation component which simulates the designed circuit and returns the outputs.

The main concern for such a tool is the validity of the outputs that the system should return correct outputs for every type of circuit in a reasonable time and another concern is to provide a user friendly interface for designing the circuit. Firstly we should verify that each component of the project (Circuit Engine,GUI,File Handling, Simulation etc..) functions correctly and then verify that whole project functions correctly when the components integrate. Testing process will lead us to produce logically correct, user friendly and high performance product.

1.2. SCOPE

Up to now the components of the project were tested by the team members who were owners of that component, changes were done due to feedback we gain in weekly meetings or when the component is tested and failed by the owner of that component. We also did some small tests when we integrated the components of the projects (File ,GUI, Simulation,and Circuit Engine) and made changes accordingly.

As we approach to the completion of the project we establish our testing plan. Below are the testing strategies that we will use to verify the correctness and quality of the product.

- Unit testing
- Integration testing
- Interface testing
- Validation testing
- High-Order testing

The document will give the details of the Testing Plan of DigArt project.

1.3. MAJOR CONSTRAINTS

The major constraints in testing the system are the time, platform independency and group members. As we come closer to the final release we should carefully arrange the time to be spent to development(improvement) and testing the software since each member continues to develop the components. We claim that our product will be platform independent so tests should be done in both Windows and Linux platforms. Also we test the user friendliness of the project, we try to achieve this by the feedback we gain in weekly meetings and also discuss with group members.

2. TESTING STRATEGIES

2.1. UNIT TESTING

By unit testing we will test each component separately. The unit testing will help us to test the low levels of the project. Test Codes will be written to test important paths of the components. Tests will be done by passing valid and invalid data through the components and monitoring it to find the errors. If all the individual tests pass then the unit testing is accomplished. The test will be done by the group member who is/are the owner of the component.We will do unit testing to File, CircuitEngine, Simulation and CustomCircuitElement components due to lack of time and that some components need the other components to be developed. (GUI ,help…) We will use JUnit tool for Unit testing.

Below are some important paths of the components that should be tested and verified by the owner.

2.1.1. File Component

- **Sample test 1**
  -Give null filename to the load file method in the File component

  -See Filereader is not activatede

  -Return value should be null (GUI takes this value and sets the filename on top if it is null does not change the filename)

- **Sample test 2:**
  -Give a valid filename to the load file method

  -See Filereader is activated

  -Read contents of the file ,fill the CircuitElements vector passed as parameter to load file method

  -Return filename (GUI takes this value and sets the filename)

2.1.2. CustomCircuitElement Component

- **Sample test 1:**
  -Give CircuitElements vector (that is in an invalid order to be designed) as parameter to Parse_and_CreateCustomCircuitElement method

  -See the escape from loop when the inappropriateness is detected

  -Exit function

- **Sample test 2:**
  -Give CircuitElements vector (in valid order)
  -Do not provide a name for the CustomCircuitElement
  -See CustomCircuitElement's name is "Untitled.dae"

2.1.3. CircuitEngine Component

- **Sample test 1:**
-

  -Insert “OR”

  -See Item Id is increased

  -See the element is added to the CircuitElements vector

- **Sample test 2**
  -Move the object

  -See if the center coordinates of the object is changed and is correct

- **Sample test 3**
  -Connect two ports

  -See if the connected_item field of the element is changed and is correct

2.1.4. Simulation Component:

- **Sample test 1**
  -Add “And” (call function addAnd)
  -Call Connect method to connect ports
  -Give value 1 to both ports
  -See return value is 1

Simulation component should test if it works properly for circuits no matter how many elements and which types of elements are in the circuit. The circuit elements that are not connected to any other circuit element must be handled. The tester should test what happens in that situation (correction done: The input ports of the element are connected to ground) .Simulation should return the result of every output element in the circuit. Behavior of the circuit elements should be tested.

2.2. INTEGRATION TESTING

We have different components in the program. The maturity of one component alone does not guarantee that it will work correctly when the whole system integrate especially there are some components not tested by Unit test. We must make sure that data sharing occurs properly among the components. We will use bottom up integration technique. Firstly one component will be added to the system and its integration is tested.

After the verification, another module is added and tested. After the test of all independent components, whole system will be tested.

2.3. INTERFACE TESTING

2.3.1. Menubar

.

- **Sample test 1:**
  -Click to Easy Access Tool button in the tool menu
  -See a dialog box opens
  - Choose the circuit elements from left to right.
  -Click Apply
  -See that the buttons in the easy access bar has set to draw the circuit elements selected by the user and the button names are changed.

2.3.2. Easy Access Menu and Toolbar

- **Sample test 1:**
  -Select object or objects by drawing rectangle with the mouse

  -See the objects are highlighted

  -Click “delete” button from toolbar

  -See selected objects are removed

- **Sample test 2:**
  -Click “Or” button from easy access bar see if the button name has changed as Or(1),click once more see if it is changed to Or(2)

  -Click any point in the canvas

  -See “ Or” component drawn

  -Click once more in canvas

  -See second “Or” is also drawn

2.3.3. Keys

- **Sample test 1:**
  -Select object
  -Press “R”
  -Object will rotate clockwise 90 degrees.

2.4. VALIDATION TESTING

By Validation testing we will verify that the product meets the requirements specified in our Software Requirements Analysis. The requirements of DigArt are divided into sub requirements as File Operations, Edit, View, Insert, Simulation, Circuit Display, Tools and help operations. Each requirement will be tested uniquely. Black Box testing will be applied to whole software that is integrated. The inputs will be given according to each requirement under these main subtopics.

We will use the product to test the whole functional requirements and sometimes we may use unit testing to ensure that the communication is proper between the components. We have CiruitElements vector that is filled by the circuit elements drawn in the canvas with their unique properties. This vector is parsed and send to simulation so we always have to verify the correctness of the vector.

2.4.1. File Operations

We will load, save diglog or digart files or test the other requirement stated in the report and we will see its communication with the circuit engine if the file loaded correctly that our CircuitElements vector is filled correctly and the flag variables are set accordingly. File Operation Component and Circuit Engine component's communication will be figured out.

2.4.2. Edit, View, Insert, Circuit Display, Tools

These requirements will be tested by using the GUI .We will test all the functionalities stated by DigArt if they are provided and handled correctly by the Circuit Engine Component .We will both use the keys assigned and the buttons in the GUI of the product.

2.4.3. Simulation

We will use the product to enter the inputs from Canvas where circuit is designed and expect to see the results in the canvas. The communication between the Circuit Engine Component and Simulation Component will be tested by Unit testing since it is the most important path of the product. The verification will be done by entering valid inputs with predetermined results and compare these expected results with the results gained from the Simulation component.

We expect not to have trouble since Unit testing and integration testing is also applied but in case of failure the results will be recorded and traced to requirement number in Software Requirements report. Change will be done to correct the wrong implementation of the requirement.

2.4.4. Design Validation

The design validation is another concern for us. We are consistent with our design specifications, the components are implemented accordingly but we have made some changes, more classes are added to the components to improve the design.

2.5. HIGH ORDER TESTING

We plan to do higher order testing to determine the efficiency of our product in different conditions. Performance Testing will be done. We will observe the time it takes the software to produce output when too many circuit elements are drawn in the canvas. We will check the response time in slow machines. If it is not in our determined bounds we will revise our parsing and simulation algorithms. We will use JMeter to view statistics regarding our programs performance.

3. RESOURCES, STAFF, TOOLS

Testing process is very important for the projects. In big projects there will be specialist testers to detect errors in boundaries or branches. We did not assign a special member to be a tester. All the members will contribute to testing process. For Unit tests each member will be responsible to use JUnit tool to verify the validity of the component. Every developer will write JUnit tests for the classes that he developed also run the testing code at every time that he/she changed some coding on the class. He will also be responsible for changing test cases if necessary.

3.1. TOOLS

JUnit test tool is used because of its friendly interface and that it is easier to learn the usage. It has reporting tools. Since none of us has experienced using such a tool before we have chosen it among other test tools of Java. For the performance test we will use JMeter tool.

3.2. TESTING ENVIRONMENT

We will use PCs for testing our project. We will also use slower machines for high-order testing. Our software is platform independent, so we will test it in Linux platform to observe the performance.

3.3. TEST RECORD KEEPING AND LOG

For JUnit test the input data and the results should be saved Data recording will be in excel since its usage is good and some operations like sorting will be useful. We will also keep the description of the test .A template will be prepared where each tests results and required data will be saved. Test log will be kept to monitor the tests that have been applied. An error log will also be kept to monitor problems during testing.

4. TEST SCHEDULE

Test Plan Delivery	23.04.2006
Unit Test and Integration test	Till 09.05.2006
Validation test	09.05.2006- 16.05.2006
High-Order test	16.05.2006-18.05.2006
Trace and Corrections	Till 22.05.2006