Circuit Editor

Using Java

Department of Computer Science and Technology

Bengal Engineering and Science University, Shibpur

Under the Guidance of

Prof. Manas Hira

Submitted by

Soham Sinha

Rajdeep Chakraborty

Sukhen Mitra

Acknowledgment

Successful competition of any project is completely dependent on team effort rather than individual effort. We have taken great efforts in this project. However, it would not have been possible without the kind support and help of some individuals and forums. We are highly indebted to Professor Manas Hira for his guidance and constant supervision as well as for providing necessary information regarding the project. We would like to extend our sincere thanks to all others who have helped us.

Finally, yet importantly words are inadequate to express our heartfelt thanks to our beloved parents and teachers for their blessings, our friends for their help and wishes for the successful completion of this project.

_____

Soham Sinha (110508053)

_____

Rajdeep Chakraborty (110508041)

_____

Sukhen Mitra (110508043)

CERTIFICATE

This is to certify that the project work entitled “Circuit Editor Using JAVA” being submitted by Soham Sinha (Exam Roll - 110805053), Rajdeep Chakraborty (Exam Roll - 110805041) and Sukhen Mitra (Exam Roll – 110508043), final year students of Bachelor of Engineering (Department of Computer Science & Technology) of Bengal Engineering & Science University, Shibpur, is a record of the work which has been carried out under my supervision.

(Prof. Manas Hira)
Project Guide
Department of Computer Science & Technology
BESU, Shibpur

(Prof. S. Chakroborty)
Head of the Department
Department of Computer Science & Technology
BESU, Shibpur

Table of Contents

Chapter 1

Introduction

Circuit Editor is software implemented by Object-Oriented Programming approach (using JAVA). As the name suggests, this software is capable of providing an environment to design new circuit with different circuit components. The features and the objectives of the circuit editor will be described and also short description about the entire document will be covered in this chapter.

1.1 Objective:

Main aim in developing circuit editor is to provide a user friendly interface mainly for the novice users as well as expert users to integrate different kinds of circuit components to design a circuit. The best way of developing complex circuit as well as integrating of various components is tried to be given. This system provides the detail structure of all the components, integrating process, dismantling of the components. Core purpose is to manage all functionalities of a circuit editor and facilitates users to design the circuit as per their choice.

1.2 Features:

Some of the important aspects and features of circuit editor is described below:

- Novice users as well as expert user can use this to draw a circuit with ease and his/her own preference.
- One can add existing circuit components as well as new components, created by their own
- Design a circuit and modify them as and when required by the editing options available.
- Smoothly move the structure/components for designing of new circuits using existing circuit components/new components.
- Save and load a circuit for future use.
- Save the circuits in different standard file formats already available.

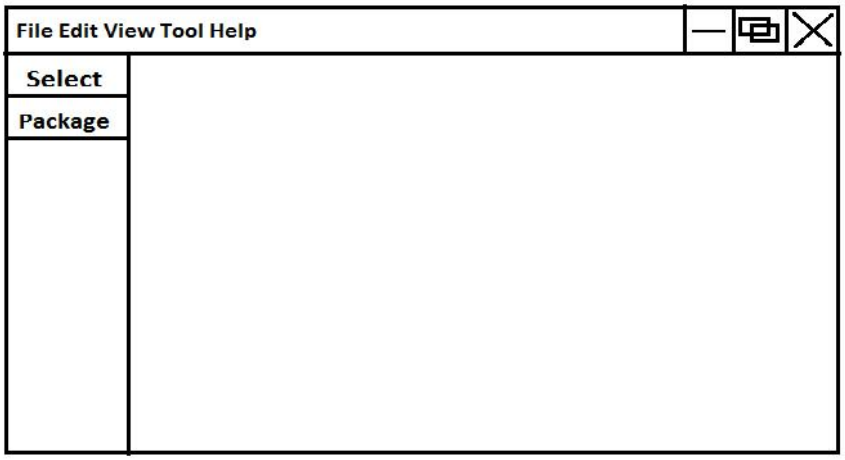
The total document is divided into a number of chapters and each chapter is described briefly later.

Chapter 1 contains the objectives and the features of this circuit editor. Chapter 2 consists of the detailed functional requirements of the circuit editor. It is basically about the functionality that should be implemented in the software. Chapter 3 tells about the tools used during the development stages and the adopted formats to do different things within the circuit editor such as file format etc. Chapter 4 is all about the detailed description of the classes used and the class diagram generated from NetBeans and Umbrello. In the Chapter 5, a session with the editor is shown. Different functionalities available in the circuit editor and how to achieve them, has been described in this chapter. In Chapter 6, the futuristic scope of this work is discussed and how it can be handful for other jobs and various fields of technology has been explored.

Chapter 2

Requirements

The functional requirements are described briefly. A user-friendly Circuit-Editor should have a Main Menu containing the following options:



2.1 File:

Somethings about file. ?? Picture

2.1.1 New*:* This option will give a new drawing pad to design a new circuit. In case a circuit is opened already, then a confirmation to Save/Cancel will be asked.

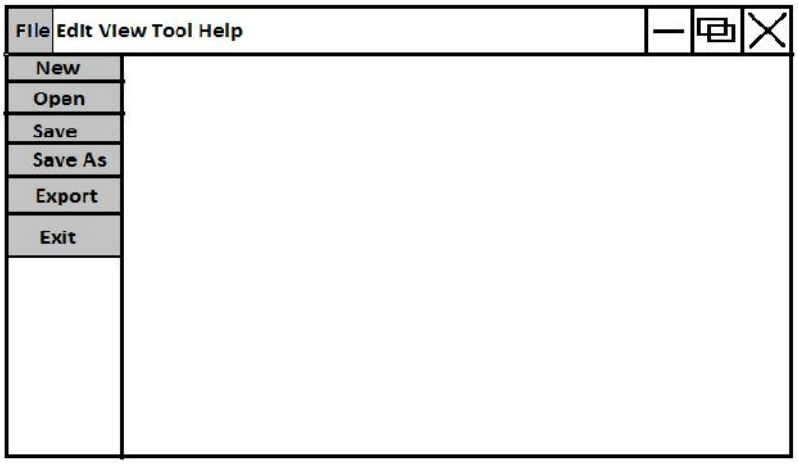
2.1.2 Open*:* Whenever user will click this option, a new file browser window will open and user can select previously saved circuit.

2.1.3 Save*:* By clicking this option, a new file browser window will be opened, where user can save the current drawn circuit in XML format. This certain XML file can be used in future.

2.1.4 Save As*:* This option is for saving an existing circuit with a new name.

2.1.5 Export*:* Here the user can save the drawn circuit in some special formats like JPEG, JPG, PDF etc. These formats are not modifiable. So, these exported files can't be used in future for modifying the circuit. But sometime, these kinds of formats are needed for different purpose. So, this useful feature needs to be incorporated.

2.1.6 Exit*:* The application will be closed. An extra checking option will be there for the unsaved projects after the click the Exit menu item.



2.2 Edit:

2.2.1 Cut*:* This option will remove the selected components from the drawing/designing area and copy them in the clipboard.

2.2.2 Copy*:* This option will copy selected components of the circuit in the clipboard.

2.2.3 Paste*:* The copied or cut portions will be placed in another position in the circuit where the user wants.

2.2.4 Select*:* User can select certain components in the circuit drawn.

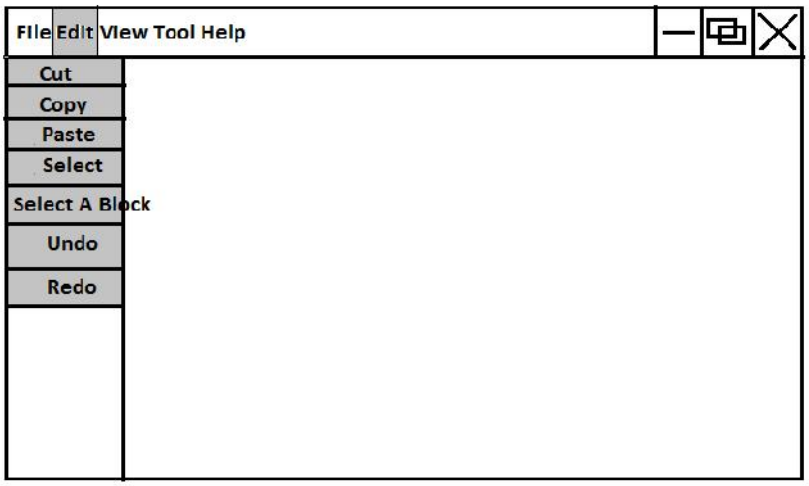
2.2.5 Select All*:* The entire circuit can be selected by this menu item.

2.2.6 Select A Block: User can select a block (block consists of multiple components together) at a time and can perform necessary edit-work like copy, paste, cut, move etc.

2.2.7 Move*:* User can select any component and move it around the whole drawing pad. Along with the circuits, the connections with the certain components will also be updated.

2.2.8 Undo*:* The state, before the last action will be retrieved. This is a standard 'undo' action like all other application.

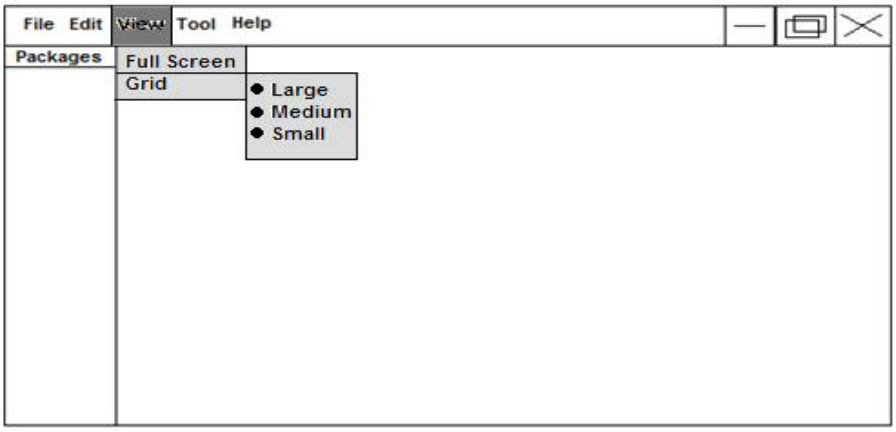
2.2.9 Redo*:* The state from where UNDO action was performed can be retrieved using REDO option. This is also a standard action.



2.3 View:

2.3.1 Full Screen: The drawing pad will occupy the full screen. This is for better viewing experience.

2.3.2 Grid: It will show grid in the drawing pad. User can set the size of the grid as per their choice. Three Types of grids are possible (Small, Medium, Large). User can also go for no grid.

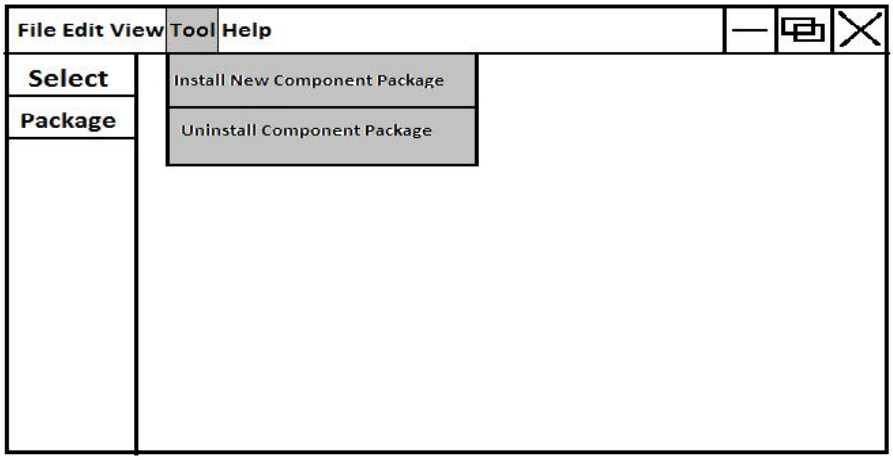


2.4 Tools:

2.4.1 Tool-> Install new component package*:* If the user wants to add some new circuit components then user can select this option. Again a File browser will open to select the package file (zip).

The package will contain the circuit components and their functionality within it. It will provide the necessary information so that the application can add the component into its functionality. More about these packages will be discussed later.

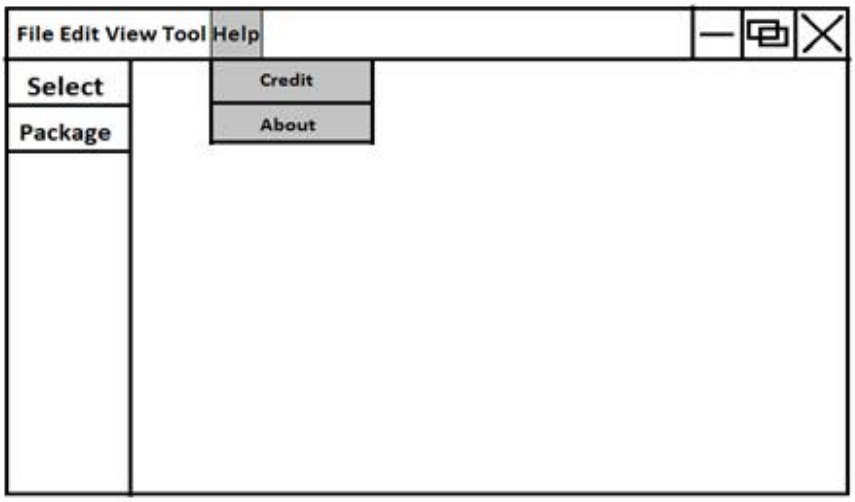
2.4.2 Tool-> Uninstall component package: if the user wants to remove some installed packages then user can select this option. After selecting this, new window will open where user can select among all the installed packages which user wants to remove. All the information, related to the component will also be removed.



2.5 Help: ???? pics

2.5.1 Help->Credit*:* Users can know the working persons behind this application by clicking this menu item.

2.5.2 Help->About*:* Users can find the current version they are working and also can know about online helping option for using the Circuit Editor.



Detailed and proper functional requirement is very necessary to build a certain application. This helps to figure out the working procedure, steps and partition of work among the involved persons. The given functional requirement is well structured and very much modular in nature.

Chapter 3

Development Stages and Environment

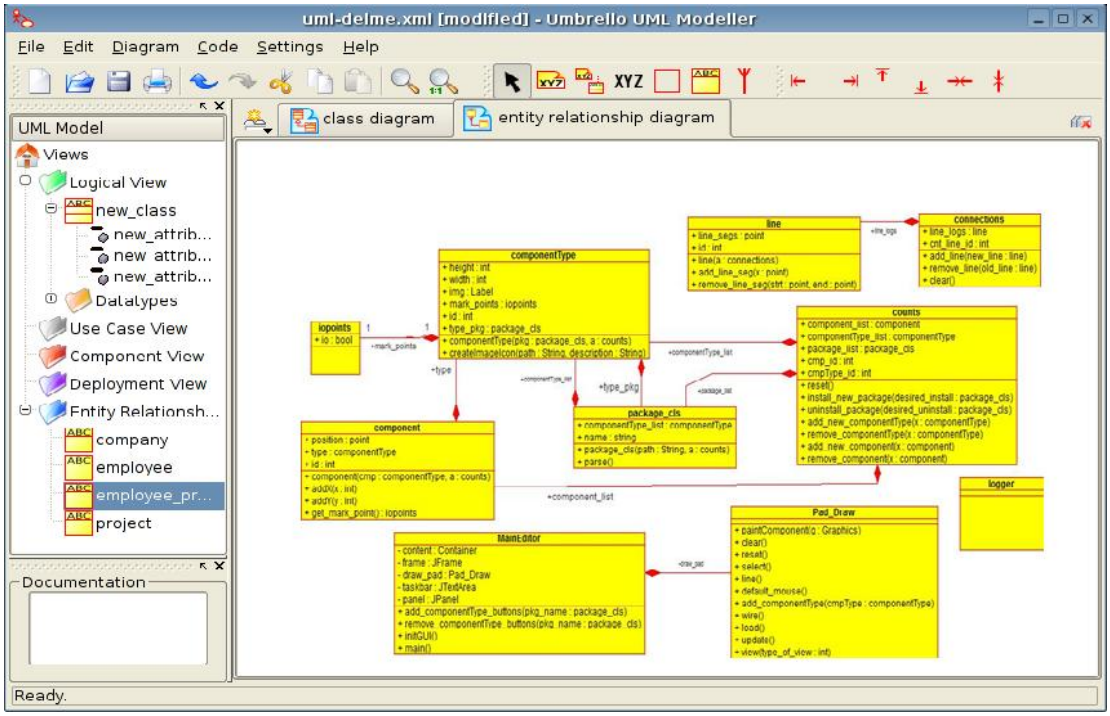
The development of the circuit editor passes through several steps using various tools. Object Oriented Programming approach is taken to develop the circuit editor. This approach requires some certain procedures to be followed like a well-structured class diagram design etc. These processes are done step by step. For successful implementation of the procedures, some standard tools are used. These tools are praised widely for their features. Those tools are described in short here. Also, during this development, some formats are taken as standard exclusively for this circuit editor, like what kind of file formats to be used for internal processing of this circuit editor etc. These custom exclusive formats are being described here also.

3.1 Class Diagram:

The class diagram represents all the classes along with its attributes and methods. It is the most efficient way to express the structure of the project and the interconnectivity among the modules as each class virtually represents a module. The class diagram is generated from “UMBRELLO UML MODELER”.

UMBRELLO UML MODELER: It is a free software UML diagram tool available natively for Unix-like platforms as well as Microsoft Windows (as part of KDE-Windows). Umbrello handles all the standard UML diagram types. Some of the supported export programming languages are C++, Java, PHP, Pascal, JavaScript, Python etc.

At the first step of this project, the classes needed, are conceptualized. Then the classes are added to this UML Modeler. Attributes and Methods are also created. This requires step by step process. As developing this, many new aspects of the problem are discovered and solutions are implemented accordingly.

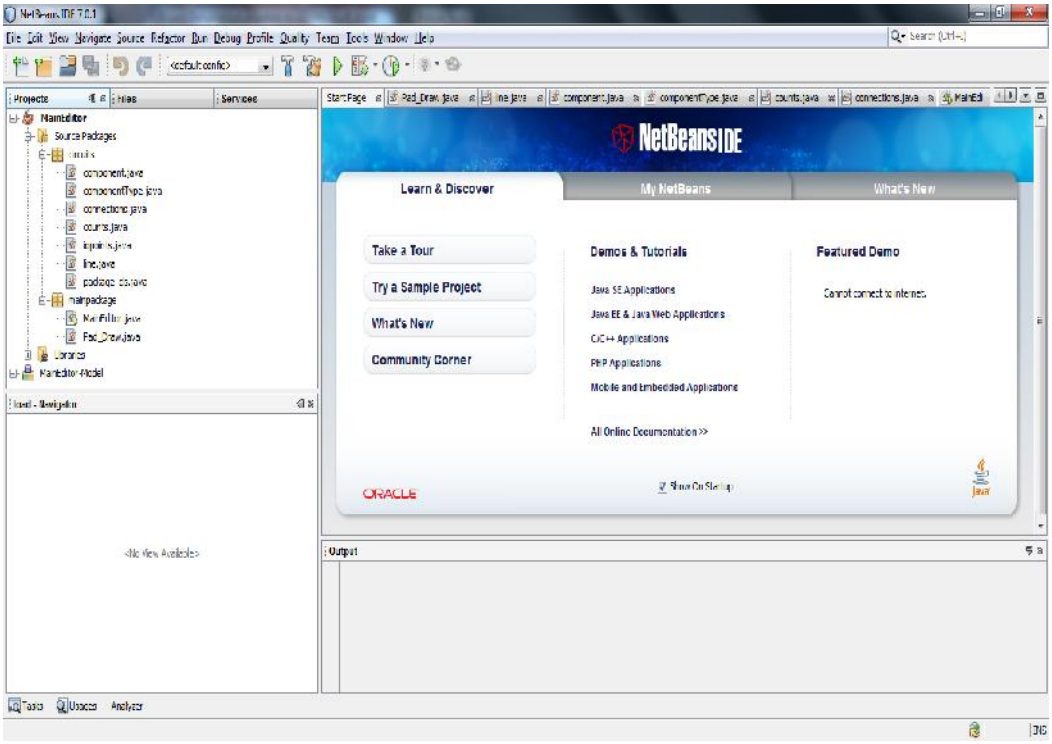


After a satisfactory structure of the class diagram, the codes are generated from the classes. Codes can be generated directly from Umbrello. Java is chosen as the coding language. So, Java codes are generated and imported to the next step. An IDE is used for coding. So, codes are imported to the IDE. The class diagram is given later in this document. Also the description of the classes, their attributes and methods are added.

3.2 NetBeans:

NetBeans refers to both a platform framework for Java desktop applications, and an Integrated Development Environment (IDE) for developing with Java, JavaScript, PHP, Python etc. The NetBeans IDE is written in Java and can run on Windows, Mac OS, Linux, Solaris and other platforms supporting a compatible JVM. A pre-existing JVM or a JDK is not required.

The NetBeans platform allows applications to be developed from a set of modular software components called *modules*. Applications based on the NetBeans platform (including the NetBeans IDE) can be extended by third party developers.



The coding part in Java is done in NetBeans IDE. As Java is object oriented programming language and some pre-defined libraries are there already to be used, this programming language is chosen. Java is also well equipped and built for all kind of programmers. NetBeans provide well documented method, attributes and class descriptions of all Java libraries and it's easy to be used. So, NetBeans is chosen as the IDE. Also, some of the UML modeling parts are drawn using NetBeans plug-ins. Using Reverse engineering; class diagram is exported from the Java codes for the circuit editor.

The codes can be found at the end of this document. The classes are given part by part for better realization.

3.3 Adopted Formats:

Throughout this project, different formats are taken to standardize different things as per the project's requirement. Some specifications will be mentioned so that anyone can know the formats being adopted in this project and work with the project for further development. Mainly, there are specifications about couple of things like how a package, containing different circuit components or modules, have been designed and how the circuit is being stored in an xml file. We are using xml file format many times because here, one can create his own tag and attribute and store necessary information.

3.3.1 Package Format:

A package contains single or multiple numbers of circuit components. So, when a user install any package with correct format (will be defined), he/she can add those circuit components into the editor and work with those. Now, standard package format will be defined.

A package has different files and those files are kept in a single folder. In the folder, there should be an xml file which contains all the information about the package and the files in the folder. Success of installing a package depends on the correct parsing of this xml file. This file is named as "manifest.xml".

As "manifest.xml" is explored, the package format can be known. The xml file is within "package" tag. In the "package", there is "name" tag. This is the name of the package and this name only is stored within the editor as a unique id of the package. Also, after successful installation of the package, the folder name is renamed with the value under "name" tag. After "name" tag, there are multiple numbers of "componentType" tags. Each of these tags consists of all the information about a single circuit component or module (like a 2-input OR-gate or 3-input NAND-gate).

Under each "componentType" tag, there are "id", "img" and "mark_points" tags. The value of "id" tag is the unique id of the circuit component under that certain package. The value of "img" tag has the relative address of the corresponding image file of the circuit component. This image file should be in JPEG format and rectangular in shape. It's advised that images of the circuit components should not be stored within a sub-folder in the main package folder and it should be kept in the main folder instead. Although correct relative address should give the desired output. To know the "mark_points" tag, firstly mark-points of a circuit component should be known. Mark-

points are those points of a circuit-component by which a component is connected with another. These mark points can be any point within the rectangular boundary of the image. Mark-points are given with respect to the top-left corner as (0, 0). Now, “mark_points” tag in the xml file, should contain an attribute named, “no”, which is indicator of how many number of mark-points are there for the circuit component. Under the “mark_points” tag, there are certain numbers of “position” tag, which contain the positions of the mark-points. “position” tag has “x” and “y” tags which are x and y co-ordinate of the mark-points respectively. “position” tag has one more tag under it, named “io”. The value of “io” indicates the nature of the mark-point. It may be input-type or output-type. This tag is used for future purpose.

This is all about how the “manifest.xml” file should look like. A sample “manifest.xml” file is given below:

```
<package>
     <name>DEF_PKG</name>
     <componentType>
          <id>1</id>
          <img>or_gate.jpg</img>
          <mark_points no="3">
               <position>
                    <x>0</x>
                    <y>14</y>
                    <io>0</io>
               </position>
               <position>
                    <x>0</x>
                    <y>54</y>
                    <io>0</io>
               </position>
               <position>
                    <x>123</x>
                    <y>35</y>
                    <io>1</io>
               </position>
          </mark_points>
     </componentType>
     <componentType>
          <id>2</id>
          <img>and_gate.jpg</img>
          <mark_points no="0"></mark_points>
     </componentType>
</package>
```

Other than the “manifest.xml” file, there should be image files in the folder in JPEG format with rectangular shape. After keeping all these files, the folder is zipped with ‘.zip’ extension. Now, the package is standardized with this project and ready to be installed.

3.3.2 Saved File Format of Circuit:

After the circuit have been designed with different installed circuit component and connected with wires, the circuit is saved in xml format for future use. The saved file in xml format can be opened later to re-structure or modify it.

The file is saved in a standard xml format as per the project’s requirement. The whole file is under “circuit” tag to specify it as a circuit. Then there is “components” tag. This tag contains all the circuit components. Under “components” tag, there are certain numbers of “comp” tags. Each of the “comp” tags have all the information about a single circuit component of certain type. Like, a “comp” tag may contain a circuit component at (25, 35) of 2-input OR-gate. Now, the 2-input OR-gate should be defined in some package and that package must be installed previously. In a package, as we have seen that a certain type of circuit component is stored with a unique id and not by the type of the circuit component. Like, a 2-input OR-gate may be stored with “id” as ‘1’ in a certain package, but nowhere in the editor the type of the circuit component is stored. So, the type of the circuit component is only the “id” in the certain package.

Each of the “comp” tag has an attribute, named “id” which defines the unique id of the component in the editor. Then, there is “comp_type_id” tag, which is the id of the type of the circuit component in the certain package. “pkg_name” is the name of the package in which the type of the circuit component belongs. Then, the “position” tag is consisting of the x and y co-ordinate of the component in the circuit editor. There may be multiple numbers of such “comp” tags, each of them defining a single circuit component.

After the circuit components, there are the connections between the components. Connections are kept within the “wires” tag after “comp” tags end. Within “wires” tag, there are multiple numbers of “wire” tags. Each of these “wire” tag is representation of a connection between two circuit components. “wire” tag has attribute, named “id” which is the unique id of that particular wire. Each “wire” tag has three tags within it, “start”, “inter_point” and “end”. “start” is for the starting point of the particular wire. It

has two tags under it, “comp_type_id” and “mark_point”. Here, “comp_type_id” and “mark_point” refers the marking point of the specific component in which the starting point of the wire is connected. “mark_point” is the serial number (array index) of the marking point which has been received from the package and corresponding type of circuit component. Under “inter_point” tag, there is an attribute, named “no”, indicating the number of intermediate points are there before reaching the end of the wire. Then, under “end” tag, there are similar tags like in the “start” tag. This is all about the format of the xml file, describing a circuit for the circuit editor.

There is an example below describing a circuit in xml format:

```
<circuit>
      <components>
            <comp id="1">
                  <comp_type_id> 1 </comp_type_id>
                  <pkg_name> DEF_PKG </pkg_name>
                  <position>
                        <x> 25 </x>
                        <y> 35 </y>
                  </position>
             </comp>
            <comp id="2">
                  <comp_type_id> 1 </comp_type_id>
                  <pkg_name> DEF_PKG </pkg_name>
                  <position>
                        <x> 200 </x>
                        <y> 80 </y>
                  </position>
             </comp>
            <comp id="3">
                  <comp_type_id> 2 </comp_type_id>
                  <pkg_name> DEF_PKG </pkg_name>
                  <position>
                        <x> 300 </x>
                        <y> 150 </y>
                  </position>
             </comp>
      </components>
      <wires>
            <wire id="1">
                  <start>
                        <comp_type_id> 1 </comp_type_id>
                        <mark_point> 2 </mark_point>
                  </start>
                  <inter_point no="2">
                        <position>
                              <x> 20 </x>
                              <y> 58 </y>
                        </position>
                        <position>
                              <x> 39 </x>
```

```
                              <y> 45 </y>
                        </position>
                  </inter_point>
                  <end>
                        <comp_type_id> 2 </comp_type_id>
                        <mark_point> 0 </mark_point>
                  </end>
            </wire>
      </wires>
</circuit>
```

This is an example of sample xml saved file format of a circuit for the circuit editor. By looking at the file, the working procedure of the circuit can be realized if the package and the function of the circuit components are known. So, this xml file can be used for various fields for the realization of the circuit and it has a great future scope. The future scope will be discussed later.

Chapter 4

Development

The success of an object oriented project relies on the efficient implementation of the classes to be used. This is the most important part of this kind of approach. Class diagrams are to be shown and detailed descriptions of the classes are to be given here, so that a full picture of the project can be extracted from this.

4.1 Class Diagram

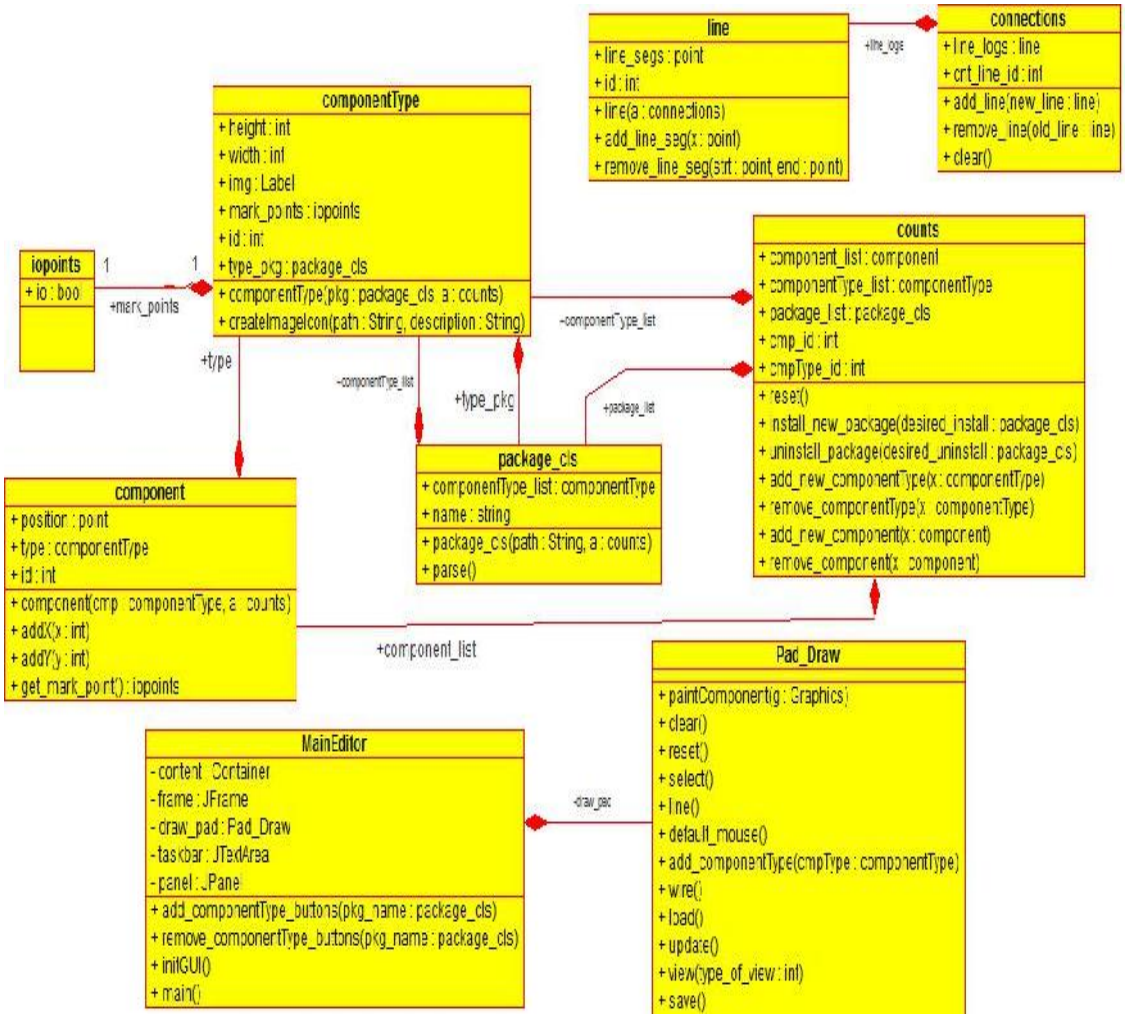
Class diagram is the main building block of object oriented modeling. It is used both for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. The classes in a class diagram represent both the main objects and or interactions in the application and the objects to be programmed. In the class diagram these classes are represented with boxes which contain three parts:

- The upper part holds the name of the class
- The middle part contains the attributes of the class
- The bottom part gives the methods or operations the class can take or undertake

All the Visibility of a Class Members Are:

```
+          Public
-          Private
#          Protected
```

The class diagram is shown in the next page.



4.2 Class Description:

The classes that can be seen in the above diagrams are now going to be described. Each of the class has its own functionality.

4.2.1 List of Classes:

I) MainEditor
II) Pad_Draw
III) counts

IV) connections
V) line
VI) componentType
VII) component
VIII) package_cls
IX) iopoints

4.2.1.1 MainEditor:

Attributes:

*Content: Container::*Create a container of content type, which will contain the circuit.

frame: Jframe:: Create a frame to display the name of the circuit.

*draw_pad: Pad_Draw::*Create the drawing pad at the centre for the circuit to be designed.

*taskbar: JTextArea::*Displays the current status of the job.

*panel: JPanel::*Contains the buttons WIRE, CLEAR, SELECT etc.

Methods:

*add_componentType_buttons(package_clspkg_name,final counts a)::*User want to add a particular component type into the circuit, then creates a button of that particular component type and then add the button in the panel.

remove_componentType_buttons (package_clspkg_name）**::**Removes a componentType Button from the frame. A class to limit the extension in the FileChooser menu.

initGUI () ::initializes the GUI.

Main() :: calls the initGUI method.

4.2.1.2 Pad_Draw ::

Draw the circuits in the drawing pad and performs necessary tasks like, reset, clear, select, add, remove etc.

Methods:

paint_component(Graphics g):: paint the components. Set the 2D pad with white and the set the colour of the component's as black.

Clear (Counts a) :: Clear all the components of the drawing pad.

Reset () :: Reset to default so that we can add other component.

Select(final counts a) :: Selects a portion of the components or entire circuit and then add mouse listener to it.

Change(int i, point p) :: change the location of a component.

Default_mouse() :: Set the mouse as default.

Remove_all_mouse_listener() :: remove all the mouse listeners.

Add_componentType (final componentTypecmpType, final counts a) **::** add a new instance of component in the draw pad. Update the pad by a mouse click.

*Wire() ::*Get a wire for connections among the components.

Pad_draw () :: initializes the drawing pad.

Load(String path, counts a) ::The pictures are saved in XML format so loading the pictures into the pad.

Update(componentTypecmpType, Pointp, counts a) :: create a new label and update the position.

View(int type of view) ::depending on the type of view grids will be displayed.

Save():: Save the current circuit in XML format.

4.2.1.1 counts:

This class tracks all the packages, types of components and components in one instance of the circuit editor. This class is used globally among all other classes when needed. “counts” is also used when someone wants to save or open some circuit. As this class contains all the information about package, components etc, this class is very much important and mainly performs addition or removing options.

Attributes:

compoentType::list: List of the componentType currently available. All the componentType are kept in one ArrayList.

component: list :: List of instances we have in our list.

package_cls:list :: List of packages we have to import components, structures.

cmp_id:int :: Generates ID of the component to identify it specifically.

*cmpType_id:int ::*Generate the identification no of each and every component type for identification and for the ease off parsing.

Methods:

Counts () :: It basically works a default constructor. Initializes all attributes to zero.

Reset () :: this method clears all the data related to a component, type, package list. Set the component id and type id as zero.

install_new_package(package_clsdesired_install) :: installs an existing package from the list of packages we have, if no packages are there, a message will be generated by showing that “ Package list is empty”.

uninstall_package(package_clsdesired_uninstall) ::
Method will remove an existing package from the list of package and along with all its information.

Add_new_component_Type(component Type x) ::
Add a new component type in component type list and assign a component id to it.

Remove_component_Type(componentType x) ::
Remove an existing component type deassign the id from the component type.
Add_new_component(component x) ::
Add a new component in the component list and assign a new component id to the component.

Remove_component(component x) ::
Remove and existing component and de assign the component id from the component.

Boolean_is_empty( ) ::
If no component and component Type is there then returns true.

4.2.1.4 Connections ::

This class keeps the track of no of line/ connections used in the circuit.

Attributes:

Line_logs: list :: an array list used to store all the connections.

Cnt_Line_id: int :: unique identification of present connection and whenever a new line is created increase it respect to the present line_id.

Methods:

Connections() :: default constructor used for initialization.

*add_Line(Line new_line) ::*add a new wire connection.

Remove_Line(Line old_line) :: remove an existing wire connection from the circuit.

Clear( ):: remove all connections from Log and clear their corresponding id.

4.2.1.5 Line::

Defines a wire connector.

Attributes:

Line_segs:points list:: One wire with different segments of lines.
Id:int :: unique id of the wire.

Methods :

*Line(connections a) ::*initializes the line segments and get the value of "Cnt_Line_id" from connections.

Add_line_segments(point X) :: add the points to the line segments to extend the line.

Remove_line_segments(Point strt, Point end) :: Get the starting and end index of the line segment to be removed. The remove the whole segment.

4.2.1.6 ComponentType ::

A Class which will define a specific type component of a circuit.

Attributes:

*Height: int ::*height of the icon representing the component in the drawing pad.

Width: int :: width of the icon representing the component in drawing pad.

*Image: JLabel ::*Label of the component type.

type_img: Image:: Image of the component type.

mark_points: points list:: marking points at the boundary of the structure with respect to top left(0,0).

Id: int :: Unique id of the component type.

*type_pkg: package_cls ::*The package it belongs to.

Methods:

Component_Type(package_clspkg, counts a, String path) ::
initialized with 'pkg' package_cls and updates the counts. Set the component type id.

create_Image_Icon(String path, String description) ::
If the path is found then set the height, width and the label of the image to create the icon in the draw pad.

4.2.1.7 Component ::

Defines the instance of a component.

Attributes:

Position: points :: position of the component denoting the top-left corner of the component in the drawing pad.

type: componentType :: type of the component, which it belongs, stored in componentType class.

Id: int ::represents the unique id of the instance/structure.

Label: Jlabel :: label of the component.

type_pkg: package_cls:: type of the package class it belongs to.

Methods:

Component (componentTypecmp,Point pos, counts a) : :
Each time an instance will be created the position and id type will be set.

*addX(intm) ::*To move the instance in X direction with the offset to m value.

addY(int n):: To move the instance in Y direction with the offset to n value.

*Get_mark_point(): iopoints ::*adding the offset in the X and Y direction fromcomponentType to the component. The marks points of the instance are found and returned.

isHit(Point point):: To check wheither the rectangle area contains the points or not.

4.2.1.8 Package_cls : :

This class defines the prototype of a package and its parsing means the contents of a package and its loading into a circuit and vice versa.

Attributes:

componentTypelist: list :: Total list of the type of components that the package consist of.

name: String:: The name of the package.

Methods:

package_cls(String path, counts a) :: a default constructor used to initialize the parameters.

parse(String path, counts a) :: This method takes the manifest.xml file as input and then parse from the saved location and unzip it to extract the components of the package to import the components in the circuit.

*add_component_type(componentType a) ::*add the component type into the component type list.

unzipFileIntoDirectory(ZipFilezipFile, File jiniHomeParentDir):: unzip the components into a fixed directory.

init_package_data(String pkg_name,intno_componentType,int[] id, String[] img_addr,counts a) :: initializing all the elements of the package. Setting the package name.setting the component type of the package as the current component type.

4.2.1.9 iopoints ::

Attributes:

Io: bool :: input and output.

if io=0 then input

if io=1 then output

Methods:

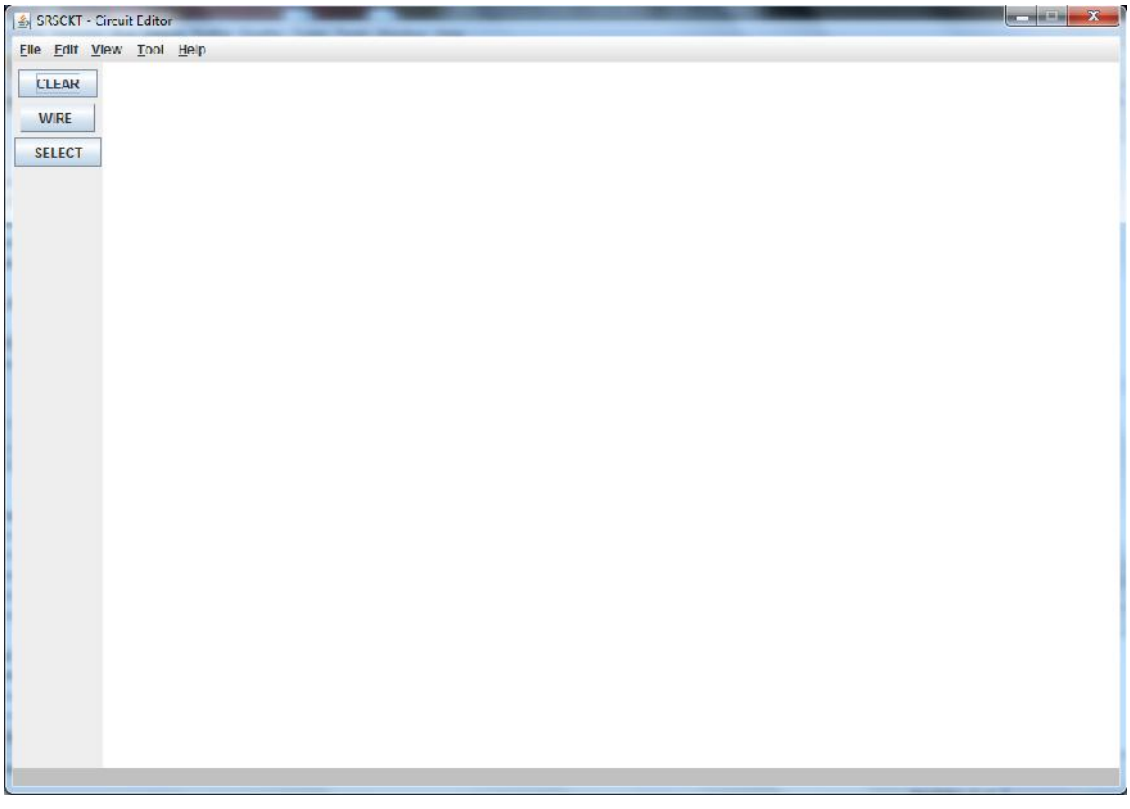
Iopoints() :: by default the points are input points.

Chapter 5:

A Session with the Editor

The functionalities that have been incorporated in the circuit editor will now be explained with the screenshots of the circuit editor. User will be able to use this circuit editor with ease by going through this interactive session.

Java must be installed in the terminal to use the circuit editor. Running the .jar executable file, a window will be opened like below.

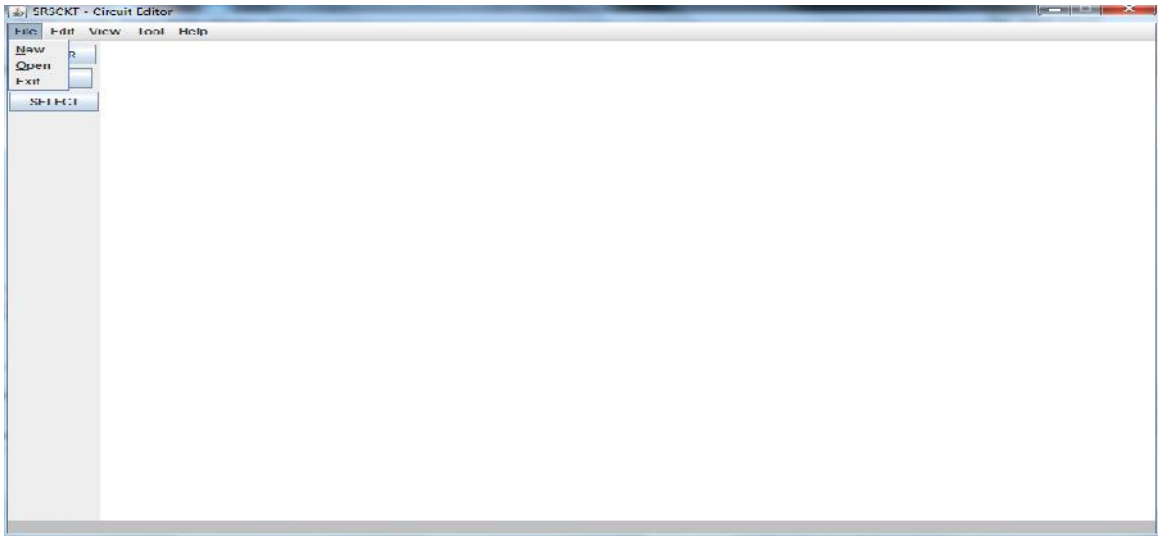


The circuit editor will be opened with some default basic features. There are main menus. Each of the main menu items will be explained in this session. The white space

at the center is working area. All the circuits are to be drawn here. There are 3 buttons at the left side panel. “CLEAR” button indicates that all the drawn design will be removed from the drawing pad or working area. “WIRE” is for adding connections between two circuit components. “SELECT” is the option to select a certain circuit component and move that in the working area anywhere.

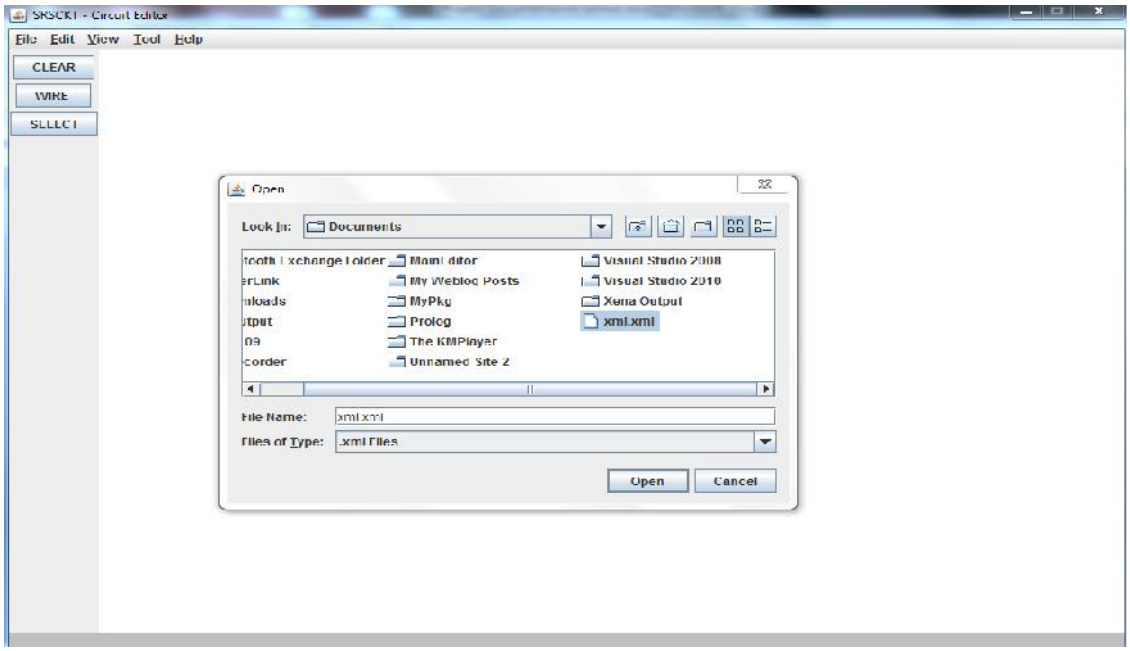
Now main menu items are to be described briefly.

After user clicks ‘File’ (Alt+F) in the menu bar, a sub-menu will be opened. There are two options, still implemented in the sub-menu - 1) New, 2) Open 3) Exit.

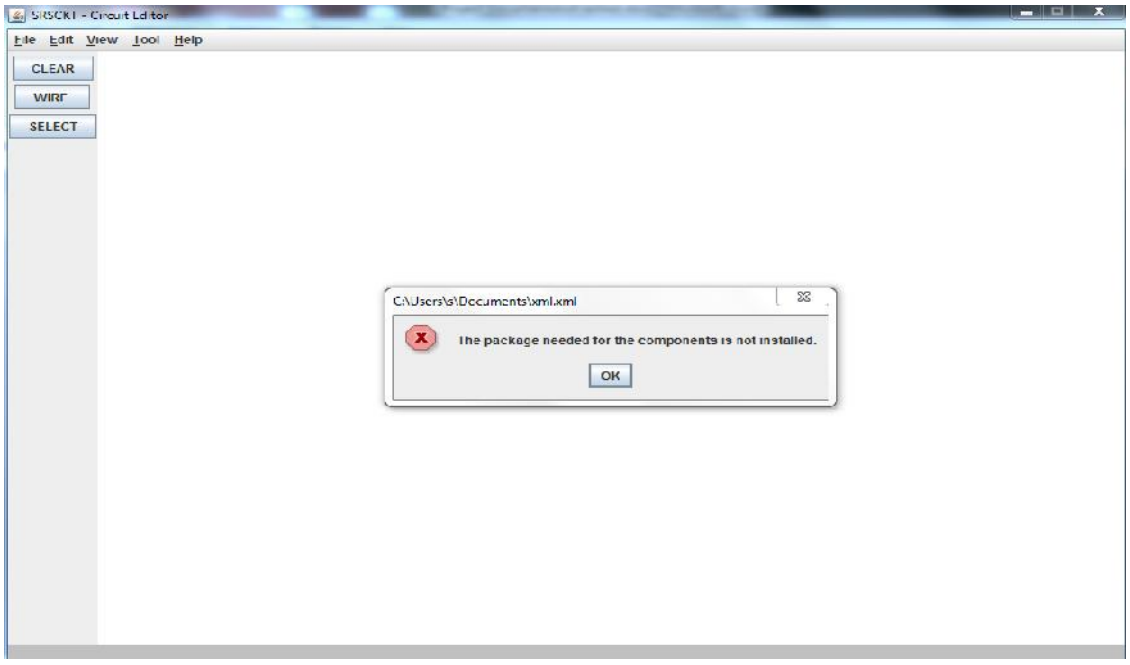


If ‘New’ (Alt+N) menu item is clicked, a new window with the default functionalities will be opened.

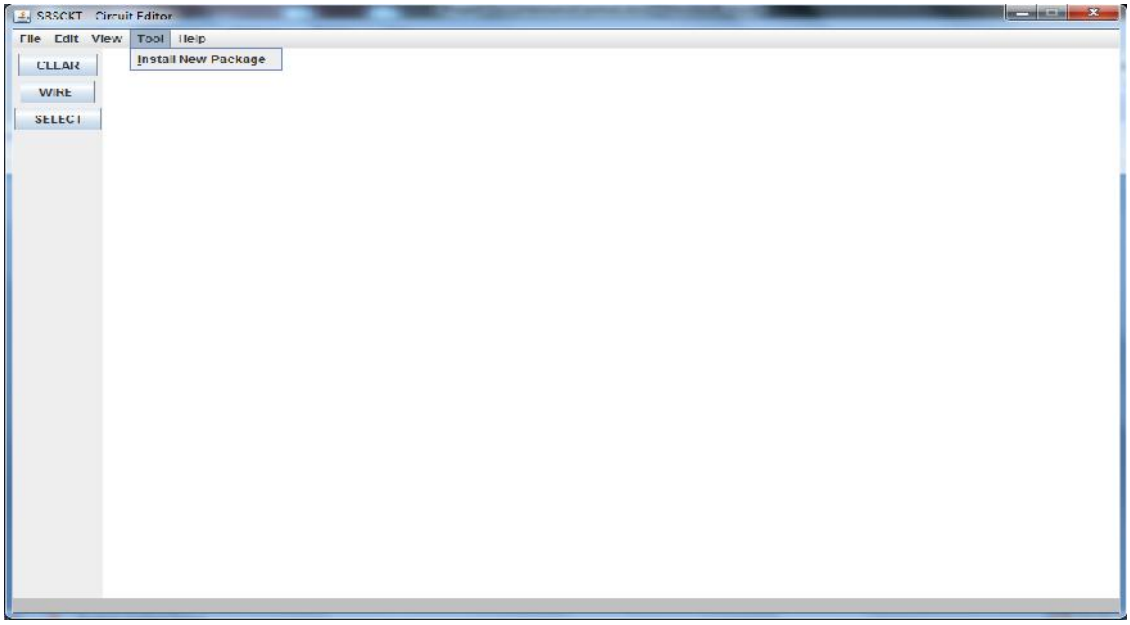
If ‘Open’ (Alt+O) is selected, then a file browser window will be there to open a certain xml file which is previously saved.



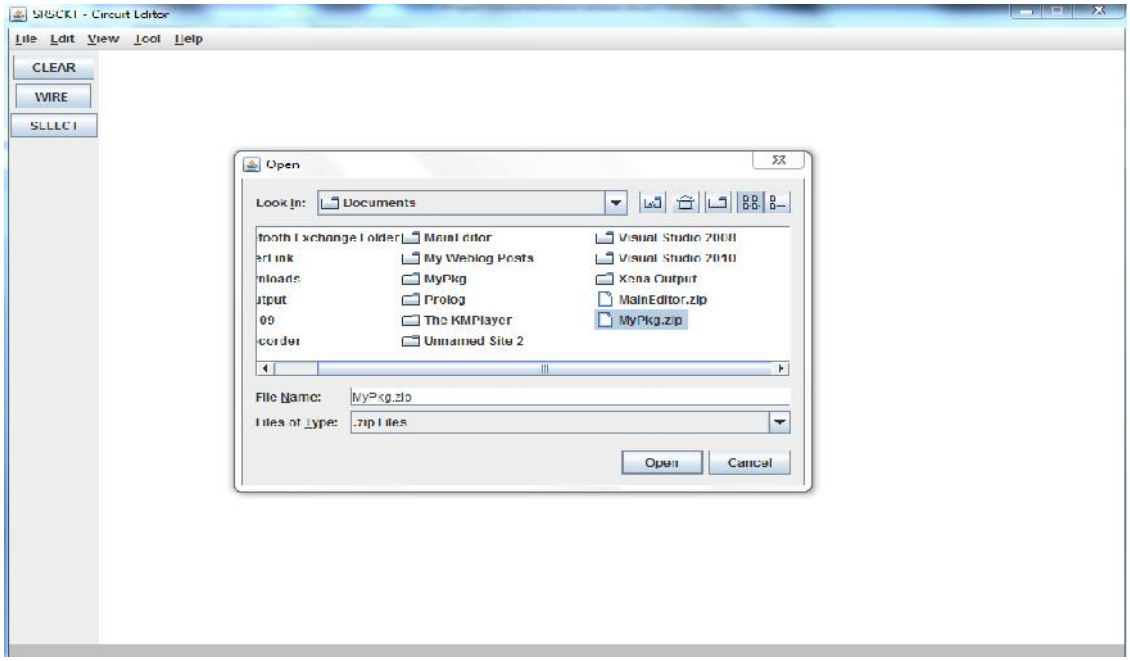
After clicking the open button in the dialogue box, the file will be opened. But this has a pre-requisite. The packages mentioned in the xml file should already be installed in the circuit editor. Otherwise, there will be an error message shown, like below:



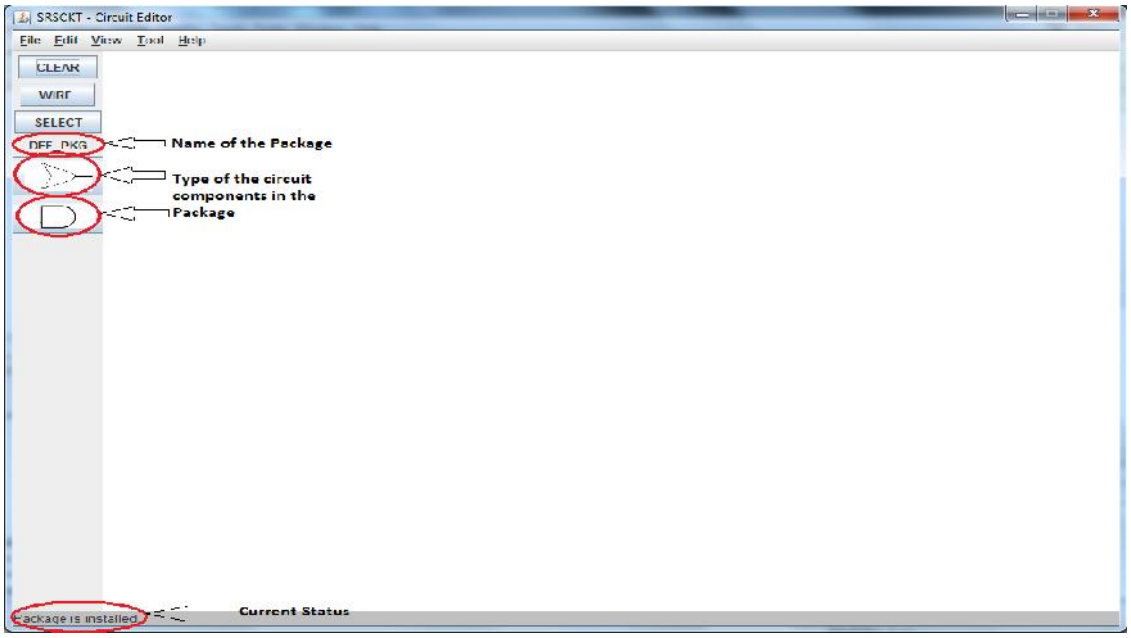
Now, user needs to install some package(s) to add the circuit component in the drawing pad or design pad. To install package user have to click the button ‘Tools’ (Alt+T) in the main menu.



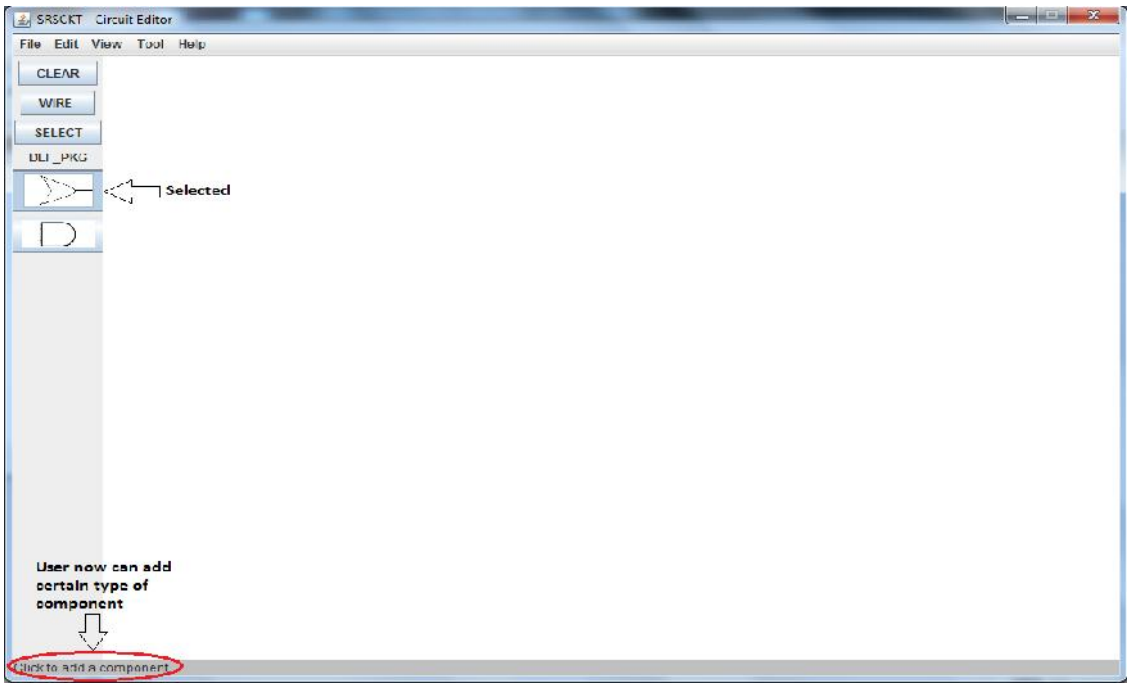
After clicking, ‘Install New Package’ menu item, a dialogue box will be opened. In the file browser window, there the correct package .zip file must be selected to install certain package. In the below screenshot ‘MyPkg.zip’ is such a package.



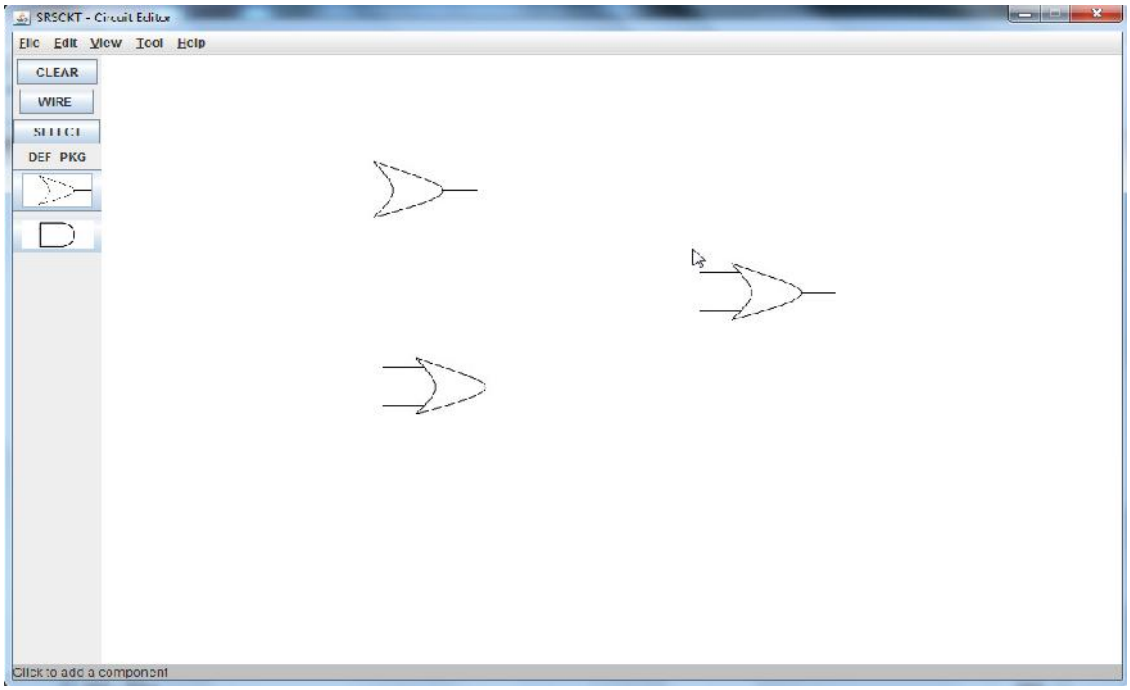
If the package is installed successfully, then the installed packages and the respective types of circuit components will be shown at the left side of the panel, with the Package name as heading, as can be seen below:



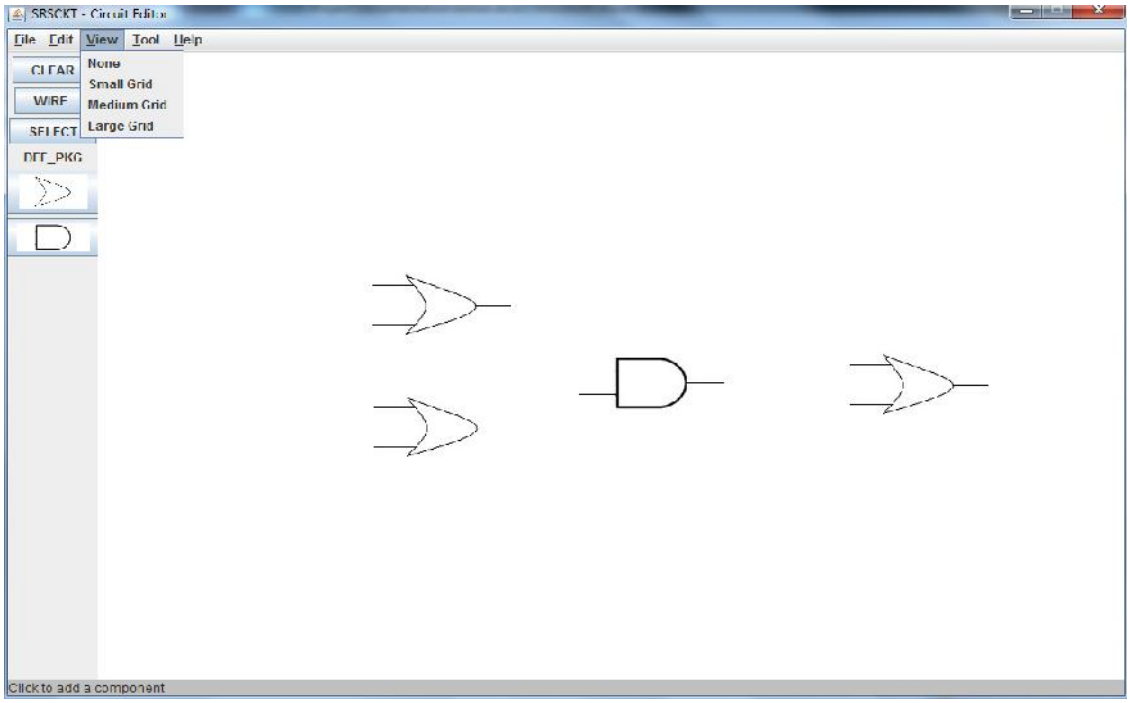
Now the button of certain type of circuit component enables the addition of that component into the working area.



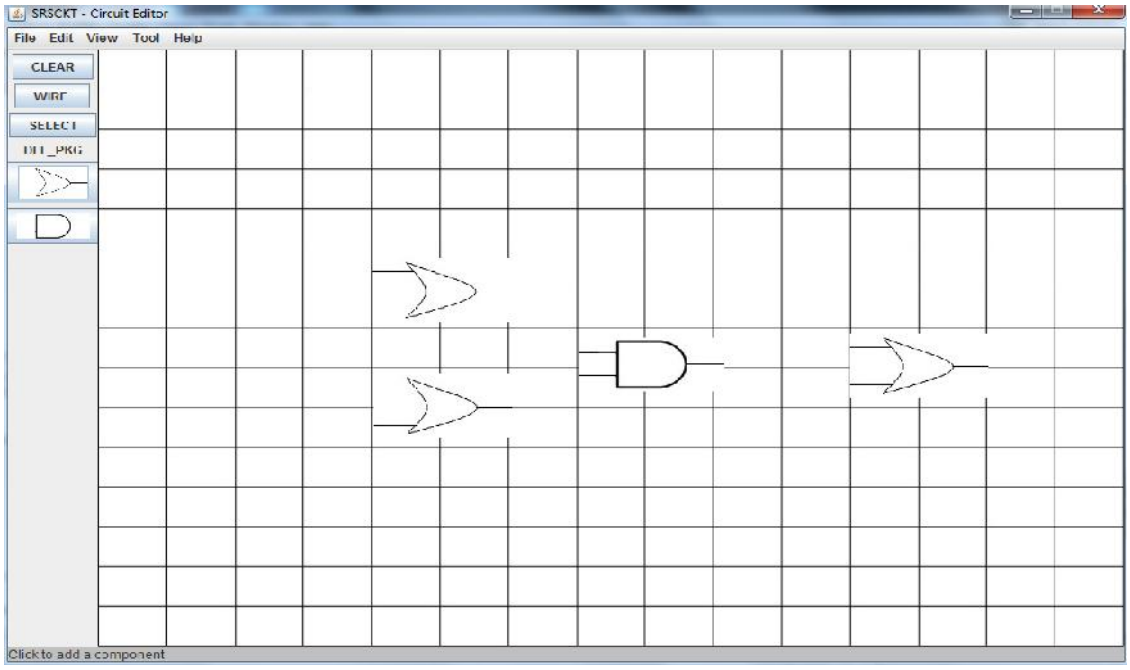
Now, anywhere in the drawing pad, a single mouse click will add a certain type of component there. In the picture given below, 2-input OR-gate is added with a single mouse click.



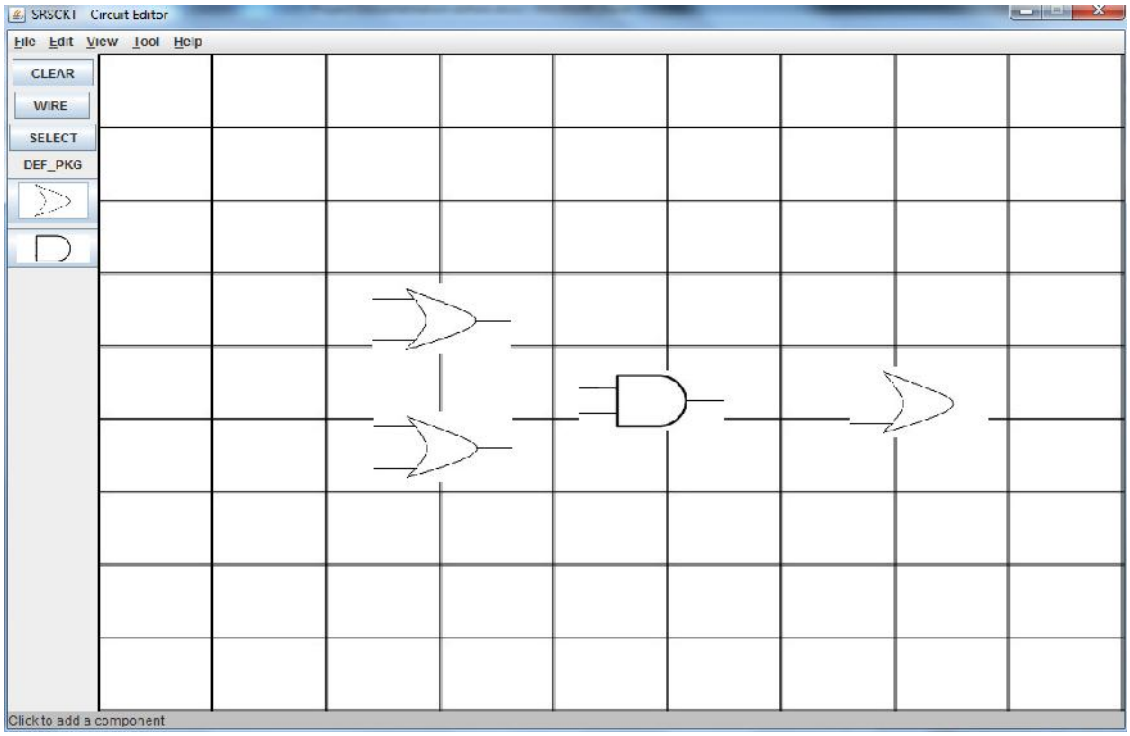
For better positioning of the components, user can enable grid view by using the 'View' (Alt+V) menu item.



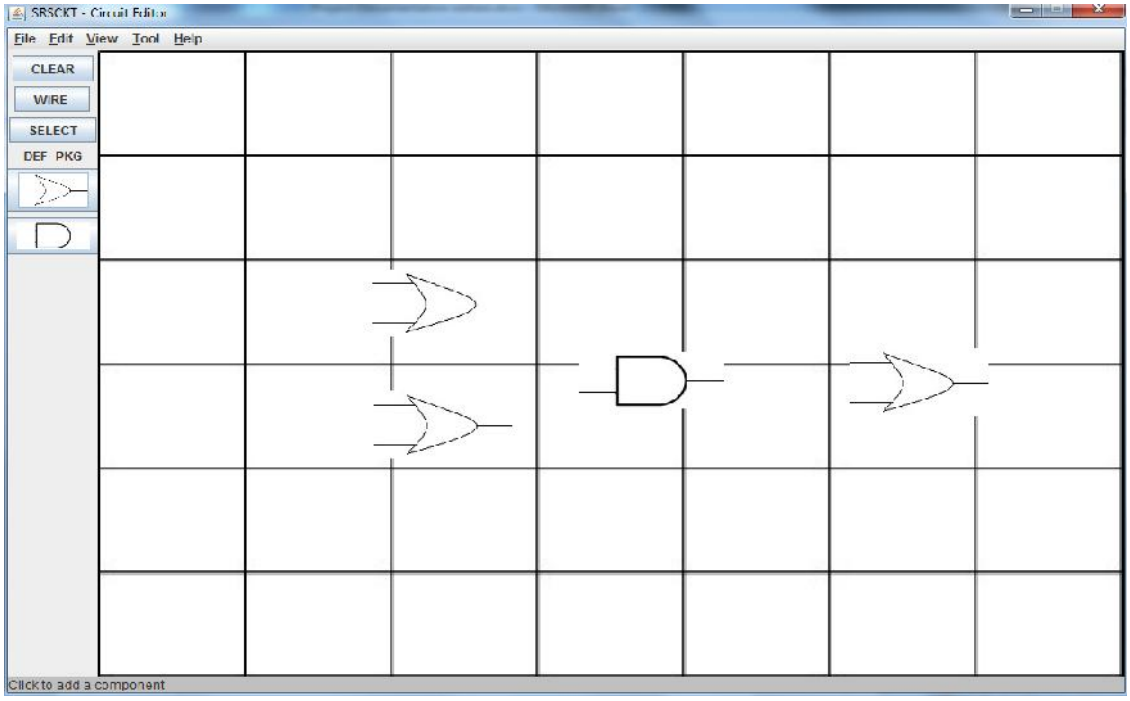
Three types of grid are available: Small, Medium, Large.
Small grid will look like this:



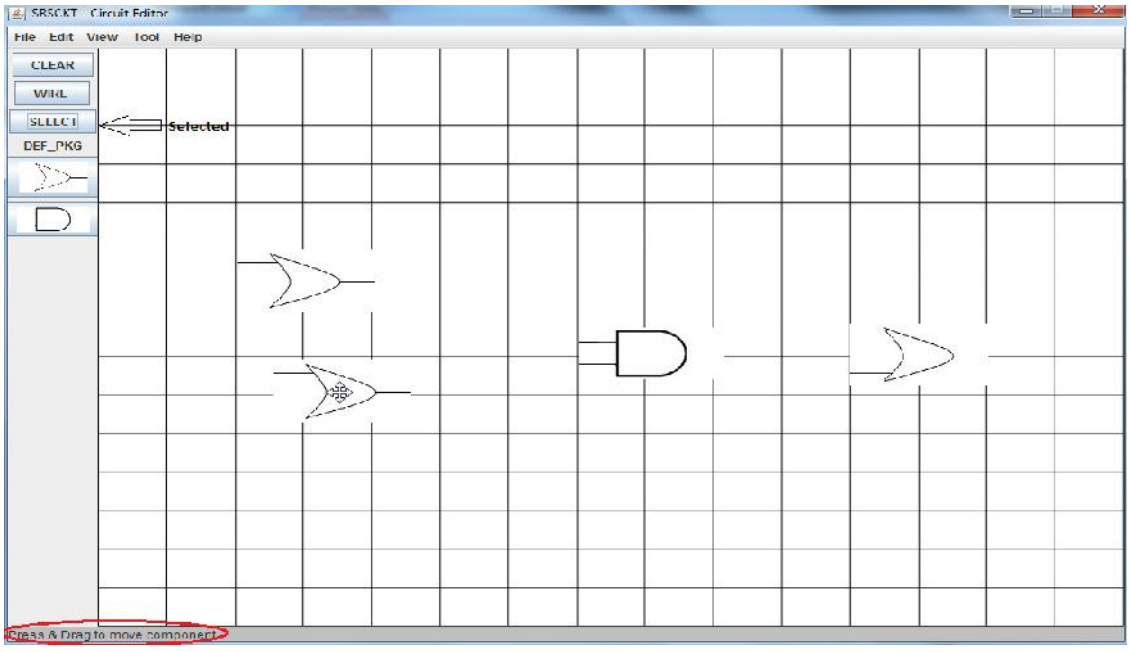
Medium grid will be slightly bigger than the last one:



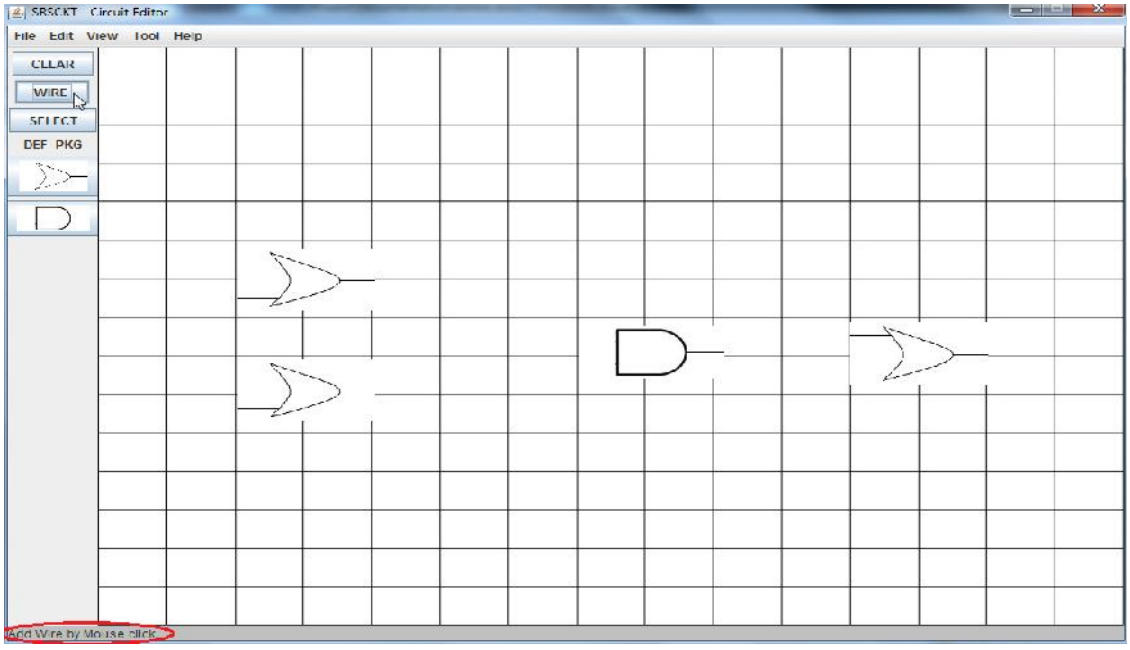
Large grid has the biggest one:



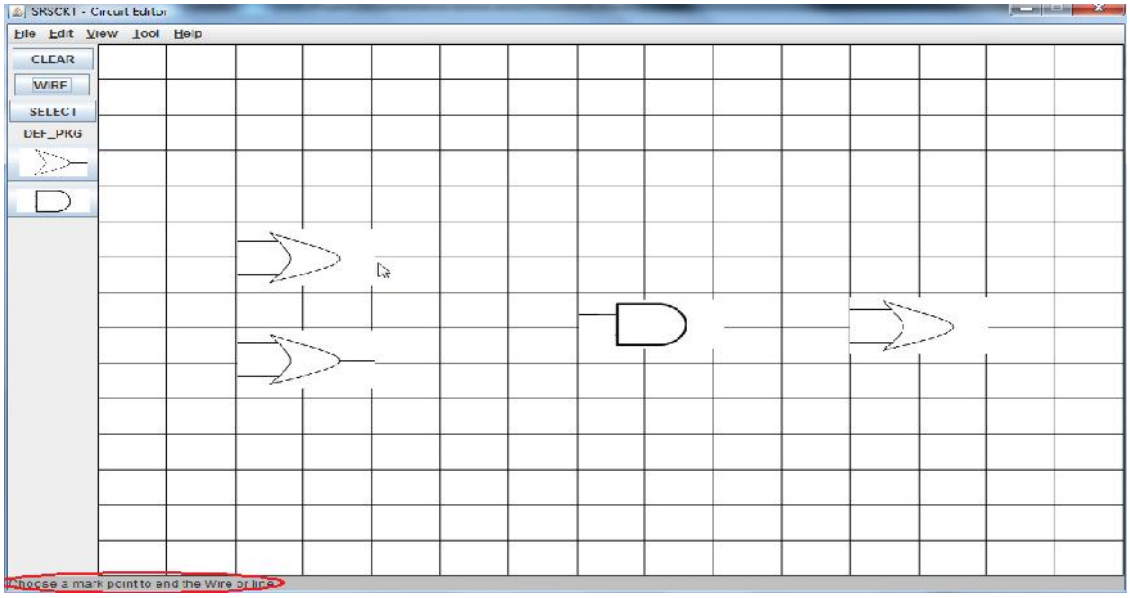
If user wants to move the components in the working area, that can be done by clicking the “SELECT” button in the left panel. Then clicking on any of the component will enable the user to move the component in the design area.



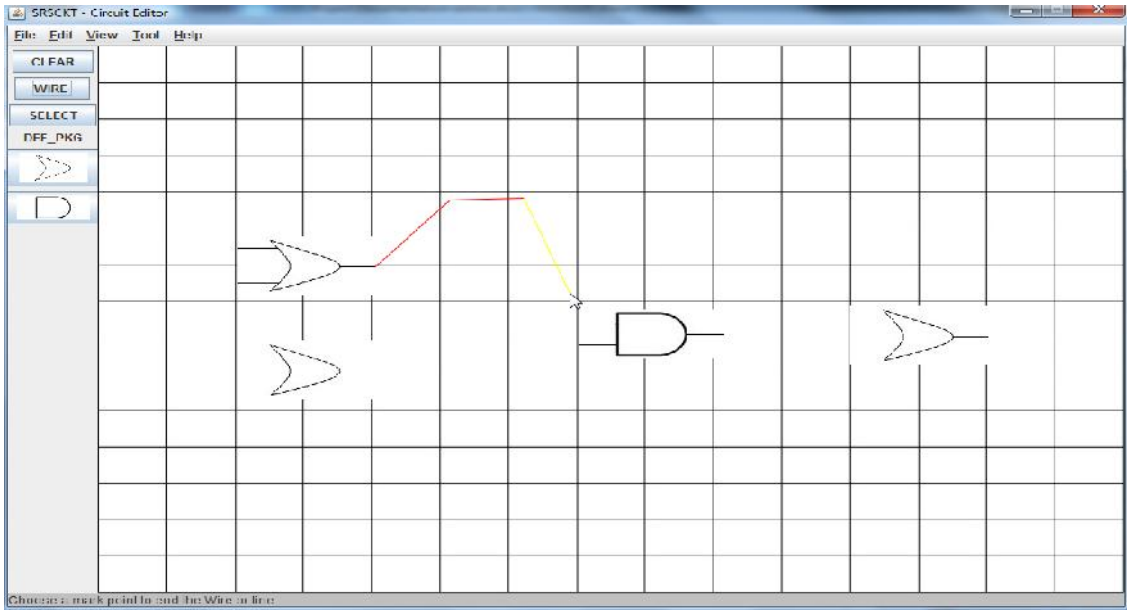
Now, user can connect two components by wires. Wires can be added by clicking “WIRE” button in the left panel. In taskbar, “Add Wire by Mouse click” will be shown on the button-click.



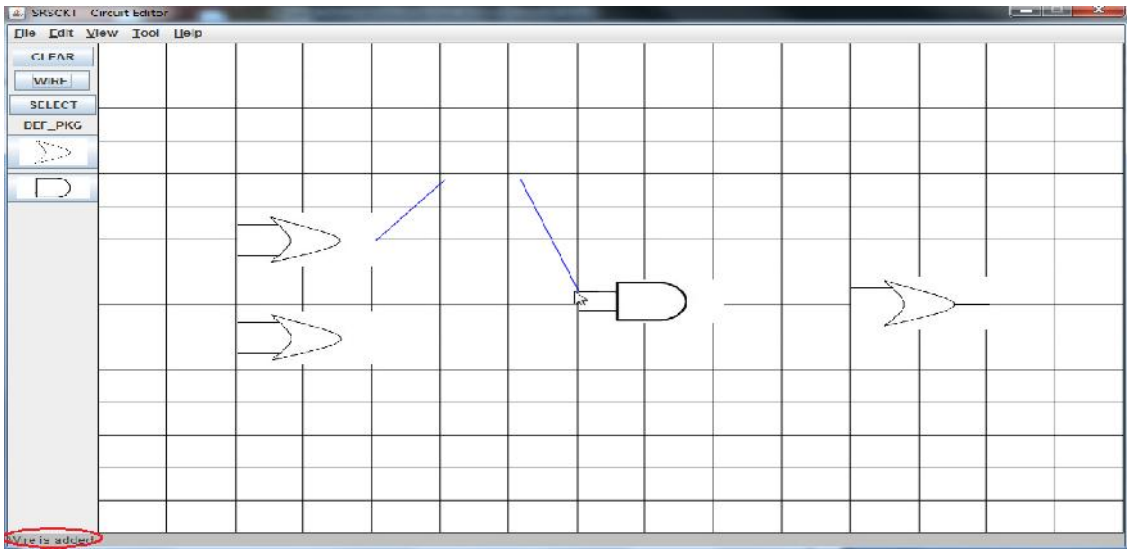
Now, clicking on any of the input output or marking point (or nearby those points) of the certain component will start the process of wire adding.



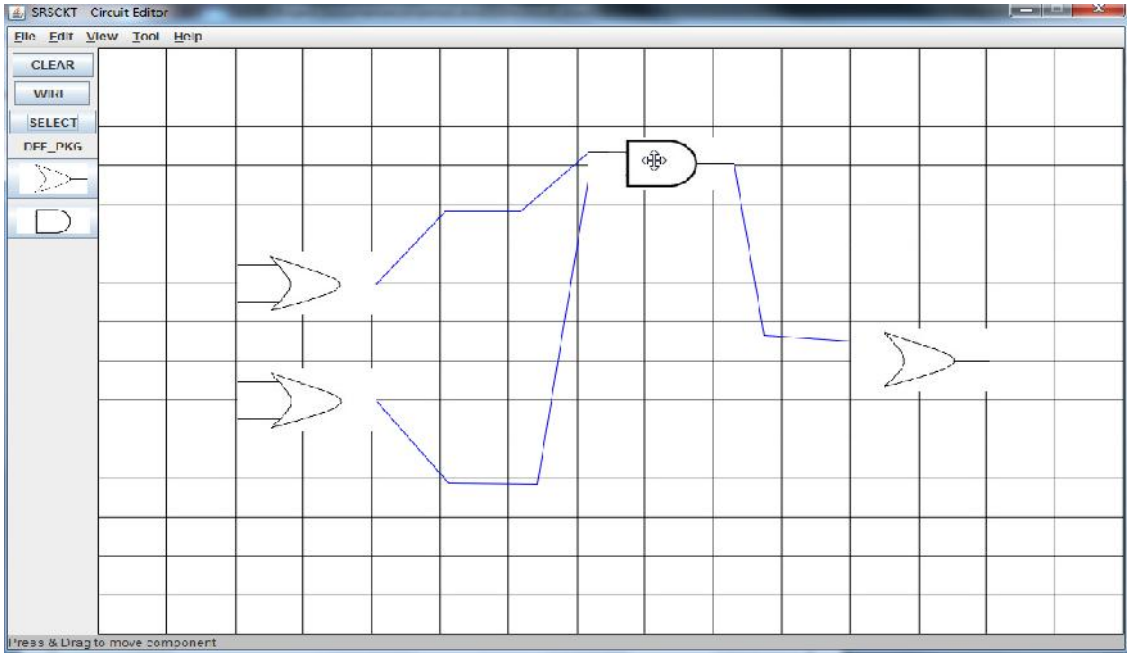
After clicking that, only clicking a marking point will terminate the wire. This is also seen on the status on the taskbar, “Choose a mark point to end the Wire or line”. Though different other point on the drawing area can be clicked as intermediate points. If some mark point is found then the wire is colored with blue. Otherwise, color of the wire will be red. When, Mouse is pressed and dragged, then color will be yellow.



If the wire has found a mark point, then wire will turn blue, with a status “wire is added”:



Now, still user can adjust the position of the components by the “SELECT” button:



‘Exit’ (Alt+E) in the main menu will terminate the application.

These functions are readily available with this circuit editor. More enhanced features will be enabled in future.

Chapter 6

Conclusion & Future Scope

The primary goal of this project is to give an interface for designing a circuit for the expert as well as for the novice users. This is an application built in JAVA following an object oriented approach. Object oriented approach is relatively new in the software designing area. The basic procedures for the approach are tried to be maintained so that the features of the approach can be utilized. All the works should be sub-divided into some parts and distributed among involved persons. That was also tried to be implemented as and when possible. But the capabilities are also kept in mind and actions are taken accordingly.

After successful implementation of the projected steps, the application is almost ready to be used. Some other functionality is to be added. They are easily implementable from the already drawn modules. They require a simple and concentrated study of this project. After that anyone with good programming knowledge can incorporate those features with fewer efforts.

The circuits are saved in XML files. These files are very much useful for future use. These files can be used to realize the circuit and work with circuit extensively such as simulation of a circuit. The XML files can be parsed easily with a XML Parser. After parsing the connections between certain components are available. If the functionality of the components are known (can be known from the installed package), then circuit simulation is not a hard task. An existing hardware language such as Verilog, VHDL is enough to for the simulation work. So, by this circuit editor, user has a graphical view of the circuit. This graphical view can be used for various presentation works like printing in some paper or showing it to some viewing media. Apart from graphical view, user can simulate it and check the circuitry if it is okay or not. So, this serves the dual purpose with minimum efforts and easy interface.

Appendix A:

JAVA Code:

A.1 Class MainEditor:

```
package mainpackage;


import circuits.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.*;
import javax.swing.JFileChooser;
import javax.swing.filechooser.FileFilter;



/**
 * Class MainEditor
 */
public class MainEditor {


  //
  // Fields
```

```
//


private JFrame frame;

private Container content;

private Pad_Draw draw_pad;

private JTextArea taskbar;

private JPanel panel;

private Pad_Draw drawPad;

private String name, path;


//

// Constructors

//

public MainEditor ()

{

    this.initGUI();

};


//

// Accessor methods

//


/**

 */

public static void main(String[] args)

{

    MainEditor theApp = new MainEditor();

}


  /**
```

```
 * Set the value of frame
 * @param newVar the new value of frame
 */
private void setFrame (final counts a, final connections cnc_a) {
  //Creates a frame with a title of "Circuit Editor"
  frame = new JFrame("SRSCKT - Circuit Editor");

  JMenu menu;
  JMenuBar menuBar;
  JMenuItem menuItem;
  /******************Menu Bar*****************/
  menuBar= new JMenuBar();//Creating MenuBar

  //Build the first (Main)-menu.
  menu = new JMenu("File");
  menu.setMnemonic(KeyEvent.VK_F);
  menuBar.add(menu);

  //Sub-menu of "File"
  //a group of JMenuItems
  menuItem = new JMenuItem("New",KeyEvent.VK_N);
  menuItem.getAccessibleContext().setAccessibleDescription("A New Window");
  // File -> New || ActionListener
  ActionListener new_window=new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
          MainEditor new_wndw=new MainEditor();
      };
  };
  menuItem.addActionListener(new_window);
```

```
    menu.add(menuItem);


    menuItem = new JMenuItem("Open",KeyEvent.VK_O);
    menuItem.getAccessibleContext().setAccessibleDescription("Open    previously
saved file");
    // File -> Open || ActionListener
    ActionListener open=new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            if(a.is_empty())
            {
                JFileChooser opn_c = new JFileChooser();
                limit_extnsn xml=new limit_extnsn(".xml");
                opn_c.setFileFilter(xml);
                    int rVal = opn_c.showOpenDialog(panel);
                    if (rVal == JFileChooser.APPROVE_OPTION)
                    {
                        path = opn_c.getSelectedFile().getPath();
                        drawPad.load(path, a, cnc_a);
                    }
            }
            else
                JOptionPane.showMessageDialog(frame, "Your Pad is not empty.
Open a new window and then try.", "Error", JOptionPane.ERROR_MESSAGE);
        };
    };
    menuItem.addActionListener(open);
    menu.add(menuItem);


    menuItem = new JMenuItem("Exit",KeyEvent.VK_E);
```

```
    menuItem.getAccessibleContext().setAccessibleDescription("Open    previously
saved file");

    // File -> Open || ActionListener

    ActionListener exit=new ActionListener() {

        @Override

        public void actionPerformed(ActionEvent e) {

            frame.dispose();

            };

    };

    menuItem.addActionListener(exit);

    menu.add(menuItem);


    menu = new JMenu("Edit");

    menu.setMnemonic(KeyEvent.VK_E);

    menuBar.add(menu);


    menu = new JMenu("View");

    menu.setMnemonic(KeyEvent.VK_V);

    menuBar.add(menu);


    //Sub-menu of "View"

    //a group of JMenuItems

    menuItem = new JMenuItem("None");

    menuItem.getAccessibleContext().setAccessibleDescription("No Grid");

    // File -> New || ActionListener

    ActionListener no_grid=new ActionListener() {

        @Override

        public void actionPerformed(ActionEvent e) {

                try {

                    drawPad.view(0);
```

```
                } catch (IOException ex) {

Logger.getLogger(MainEditor.class.getName()).log(Level.SEVERE, null, ex);

                }

        };

    };

    menuItem.addActionListener(no_grid);

    menu.add(menuItem);


    menuItem = new JMenuItem("Small Grid");

    menuItem.getAccessibleContext().setAccessibleDescription("Small         Grid
View");

    // File -> New || ActionListener

    ActionListener sml_grid=new ActionListener() {

        @Override

        public void actionPerformed(ActionEvent e) {

                try {

                    drawPad.view(1);

                } catch (IOException ex) {

Logger.getLogger(MainEditor.class.getName()).log(Level.SEVERE, null, ex);

                }

        };

    };

    menuItem.addActionListener(sml_grid);

    menu.add(menuItem);


    menuItem = new JMenuItem("Medium Grid");

    menuItem.getAccessibleContext().setAccessibleDescription("Medium        Grid
View");

    // File -> New || ActionListener

    ActionListener mdm_grid=new ActionListener() {
```

```
        @Override
        public void actionPerformed(ActionEvent e) {
                try {
                    drawPad.view(2);
                } catch (IOException ex) {
Logger.getLogger(MainEditor.class.getName()).log(Level.SEVERE, null, ex);
                }
        };
    };
    menuItem.addActionListener(mdm_grid);
    menu.add(menuItem);

    menuItem = new JMenuItem("Large Grid");
    menuItem.getAccessibleContext().setAccessibleDescription("Large         Grid
View");
    // File -> New || ActionListener
    ActionListener lrg_grid=new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
                try {
                    drawPad.view(3);
                } catch (IOException ex) {
Logger.getLogger(MainEditor.class.getName()).log(Level.SEVERE, null, ex);
                }
        };
    };
    menuItem.addActionListener(lrg_grid);
    menu.add(menuItem);
```

```
    menu = new JMenu("Tool");
    menu.setMnemonic(KeyEvent.VK_T);
    menuBar.add(menu);

    //a group of JMenuItems
    menuItem = new JMenuItem("Install New Package",KeyEvent.VK_I);
    menuItem.getAccessibleContext().setAccessibleDescription("Add a new type of
circuit component through a Package");
    ActionListener new_package=new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            JFileChooser i_n_p = new JFileChooser();//i_n_p => install new
package
            limit_extnsn b=new limit_extnsn(".zip");
            i_n_p.setFileFilter(b);
            int rVal = i_n_p.showOpenDialog(panel);
            if (rVal == JFileChooser.APPROVE_OPTION)
            {

frame.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
                path = i_n_p.getSelectedFile().getPath();
                package_cls new_pkg= new package_cls(path,a);
                add_componentType_buttons(new_pkg,a);

frame.setCursor(Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
                taskbar.setText("Package is installed");
            }
        };
    };
    menuItem.addActionListener(new_package);
    menu.add(menuItem);
```

```
    menu = new JMenu("Help");
    menu.setMnemonic(KeyEvent.VK_H);
    menuBar.add(menu);

    frame.setJMenuBar(menuBar);
    /******************End of Menu Bar*****************/
    this.setContent(a,cnc_a);
    //sets the size of the frame
    frame.setSize(1000, 700);
    //makes it so that you can close different instance indiviually(if you've
pressed "File->New")
    frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    //so that u can't change the dafault size
    frame.setResizable(false);
    //makes it so you can see it
    frame.setVisible(true);

  }

  /**
   * Get the value of frame
   * @return the value of frame
   */
  private JFrame getFrame ( ) {
    return frame;
  }

  /**
   * Set the value of content
```

```
 * @param newVar the new value of content
 */
private void setContent (final counts a,final connections cnc_a) {
  //Creates a new container
  content = frame.getContentPane();
  //sets the layout
  content.setLayout(new BorderLayout());
  this.setTaskbar();//sets the taskbar
  //adding the taskbar to the bottom-part
  content.add(taskbar,BorderLayout.SOUTH);
  this.setDraw_pad(a,cnc_a,this.getTaskbar());//sets the drawPad
  //sets the padDraw in the center
  content.add(drawPad, BorderLayout.CENTER);
  this.setPanel(a,cnc_a);//sets the panel
  //sets the panel to the upper portion
  content.add(panel, BorderLayout.WEST);
}

/**
 * Get the value of content
 * @return the value of content
 */
private Container getContent ( ) {
  return content;
}

/**
 * Set the value of draw_pad
 * @param newVar the new value of draw_pad
 */
```

```
private void setDraw_pad (counts a, connections cnc_a, JTextArea task) {
  //creates a new PadDraw
  drawPad = new Pad_Draw(a,cnc_a,task);
}

/**
 * Get the value of draw_pad
 * @return the value of draw_pad
 */
private Pad_Draw getDraw_pad ( ) {
  return draw_pad;
}

/**
 * Set the value of taskbar
 * @param newVar the new value of taskbar
 */
private void setTaskbar ( ) {
  /******************Task Bar*****************/
  taskbar = new JTextArea();
  taskbar.setVisible(true);
  taskbar.setBackground(Color.lightGray);
  taskbar.setEditable(false);
  /******************End of Task Bar*****************/
}

/**
 * Get the value of taskbar
 * @return the value of taskbar
 */
```

```
private JTextArea getTaskbar ( ) {
  return taskbar;
}

/**
 * Set the value of panel
 * @param newVar the new value of panel
 */
private void setPanel (final counts a,final connections cnc_a) {

  //creates a JPanel
  panel = new JPanel();
  //This sets the size of the panel
  panel.setPreferredSize(new Dimension(80, 50));
  panel.setMinimumSize(new Dimension(80, 50));
  panel.setMaximumSize(new Dimension(80, 50));

  JButton clearButton = new JButton("CLEAR");
  clearButton.addActionListener(new ActionListener()
  {
      @Override
      public void actionPerformed(ActionEvent e)
      {
          drawPad.clear(a,cnc_a);
          taskbar.setText(null);
      }
  });

  JButton wireButton = new JButton("WIRE ");
  wireButton.addActionListener(new ActionListener()
```

```
  {
      @Override
      public void actionPerformed(ActionEvent e)
      {
          drawPad.wire(cnc_a, a);
          taskbar.setText("Add Wire by Mouse click.");
      }
  });

  JButton slctButton= new JButton("SELECT");
  slctButton.addActionListener(new ActionListener()
  {
      @Override
      public void actionPerformed(ActionEvent e)
      {
          drawPad.select(a,cnc_a);
          taskbar.setText("Press & Drag to move component");
      }
  });

  //adds the buttons to the panel
  panel.add(clearButton);
  panel.add(wireButton);
  panel.add(slctButton);

}

/**
 * Get the value of panel
 * @return the value of panel
```

```
 */
private JPanel getPanel ( ) {
  return panel;
}

//
// Other methods
//

/**
 */
private void initGUI(  )
{
    counts theApp_new_counts = new counts();
    connections theApp_new_connections = new connections();
    this.setFrame(theApp_new_counts,theApp_new_connections);
}

/**
 *
 */
public void add_componentType_buttons(package_cls pkg_name,final counts a)
{
    int i;
    JLabel pkg_hdng=new JLabel(pkg_name.getName());
    panel.add(pkg_hdng);
    for(i=0;i<pkg_name.getComponentType_list().size();i++)
    {
        final                                                    componentType
cmp_Types=pkg_name.getComponentType_list().get(i);
```

```
            int btnHeight,btnWidth;
            double scale=0.5;
            btnWidth=(int)(scale*(double)cmp_Types.getWidth());
            btnHeight=(int)(scale*(double)cmp_Types.getHeight());
            ImageIcon myIcon=new ImageIcon(cmp_Types.getType_Img());

            BufferedImage bi = new
BufferedImage(btnWidth,btnHeight,BufferedImage.TYPE_INT_ARGB);
            Graphics2D g = bi.createGraphics();
            g.scale(scale,scale);
            myIcon.paintIcon(null,g,0,0);
            g.dispose();

            JButton strctButton = new JButton(new ImageIcon(bi));
            strctButton.addActionListener(new ActionListener(){
                @Override
                public void actionPerformed(ActionEvent e)
                {
                    drawPad.add_componentType(cmp_Types,a);
                    taskbar.setText("Click to add a component");
                }
            });
            panel.add(strctButton);
        }
        panel.validate(); //Updates the Panel
    }

    /**
     * Removes the buttons in 'pkg_name" package.
     * @param        pkg_name
```

```
   */
  public void remove_componentType_buttons( package_cls pkg_name )
  {
  }


    /**
     *a class to limit the extension in the FileChooser menu
     * like if we want to take only .jpg extension
     */
     public class limit_extnsn extends FileFilter
    {
        String type;
        public limit_extnsn(String extnsn)
        {
            type=extnsn;
        }
        @Override
        public boolean accept(File f) {
            String name = f.getName().toLowerCase();
            //if (f.isDirectory()) { return true; }
            return f.isDirectory() || name.endsWith(type);//if folder then show
folder otherwise extension
        }
        @Override
        public String getDescription() {
            return type+" Files";
        };
    }
}
```

A.2 Class Pad_Draw:

```
package mainpackage;


import circuits.*;

import java.awt.*;

import java.awt.Cursor;

import java.awt.event.*;

import java.io.File;

import java.io.IOException;

import java.lang.String;

import java.util.ArrayList;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.imageio.ImageIO;

import javax.swing.*;

import javax.swing.JLabel;

import org.w3c.dom.*;

import org.apache.xerces.parsers.DOMParser;

import org.xml.sax.SAXException;



/**
 * Class Pad_Draw
 */
public final class Pad_Draw extends JComponent{


    /**
     * The Graphics used throughout the class
     */
```

```
private Graphics2D graphics2D;
/**
 * Local counts class instance, used for moving the component
 * copy of the actual counts class-instance(global)
 */
private counts Pad_counts;
/**
 * Local connections class instance, used for drawing the lines
 * copy of the actual connections class-instance(global)
 */
private connections Pad_connections;
/**
 * Background Image
 */
private Image bgImage;
/**
 * updates the current status to task bar
 */
JTextArea taskbr;

//for lines
private int currentX = 0, currentY = 0, oldX = 0, oldY = 0,tmpX,tmpY;
private boolean mousedragActv=false, desn=true,locked=true;//for lines



/**
 * Class to define how the Select & Move works
 */
class MovingAdapter extends MouseAdapter
{
```

```
        private int x=0;

        private int y=0;

        private int mutual=-2;//mutual is to select only one component if co-
ordinate coinsides


        @Override

        public void mousePressed(MouseEvent e)

        {

            x = e.getX();

            y = e.getY();

            int i=0;

            for(component comp : Pad_counts.getComponent_list())

            {

                //System.out.println("As6e..");

                if (comp.isHit(e.getPoint()))

                {

                    //System.out.println("Got the Hit"+i);

                    //System.out.println("AA"+comp.getPosition()+","+x+","+y);

                    set_move_cursor();

                }

                i++;

            }

        }


        @Override

        public void mouseDragged(MouseEvent e)

        {

            int dx = e.getX() - x;

            int dy = e.getY() - y;

            //System.out.println("MYDX"+dx+","+dy+","+x+","+y);
```

```
            component comp = null;
            //System.out.println("It's coming2");
            for(int i=0; i<Pad_counts.getComponent_list().size(); i++)
            {
                comp=Pad_counts.getComponent_list().get(i);
                if (comp.isHit(e.getPoint()) && (mutual==i || mutual==-2))
                {
                    mutual=i;
                    set_move_cursor();
                    comp.addX(dx);
                    comp.addY(dy);
                    change(i,comp.getPosition());
                    //System.out.println("Got                                    the
HitA"+mutual+","+x+","+y+","+e.getX()+","+e.getY());
                    //System.out.println("A"+comp.getPosition());
                    //add_instances();
                    break;
                }
           }
           x+= dx;
           y+= dy;
        }

        @Override
        public void mouseMoved(MouseEvent e)
        {
            //System.out.println("It's coming1");
            set_default_cursor();
            mutual=-2;
        }
```

```
  }
  MovingAdapter slct=new MovingAdapter();

//
// Fields
//

//
// Constructors
//
/**
   * default constructor. initialize the drawing-pad with a and cnc_a
   * @param a
   * @param cnc_a
   */
public Pad_Draw(counts a,connections cnc_a,JTextArea task)
{
    this.setDoubleBuffered(false);
    this.setLayout(null);
    Pad_counts=a;
    Pad_connections=cnc_a;
    locked=true;
    this.bgImage=null;
    this.clear(a,cnc_a);
    this.taskbr=task;
};

//
// Methods
//
```

```
  //
  // Accessor methods
  //


  //
  // Other methods
  //


  /**
   * @param        g
   */
    @Override
  public void paintComponent( Graphics g )
  {
      graphics2D = (Graphics2D)g;
      graphics2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
      graphics2D.setPaint(Color.white);
      graphics2D.fillRect(0, 0, getSize().width, getSize().height);
      if(bgImage!=null)
      {
          graphics2D.drawImage(bgImage, 0, 0, null);
      }
      graphics2D.setColor(Color.BLUE);
      int j=1;
      for(line a:Pad_connections.getLine_logs())
      {
          if(a.getId()==-1)
```

```
            graphics2D.setColor(Color.red);
        for(int i=1;i<a.getLine_segs().size();i++)
        {
            Point p1=a.getLine_segs().get(i-1);
            Point p2=a.getLine_segs().get(i);
            //System.out.println(p1);
            graphics2D.drawLine(p1.x, p1.y, p2.x, p2.y);
        }
        j++;
    }
    graphics2D.setColor(Color.YELLOW);
    if(mousedragActv) g.drawLine(oldX, oldY, tmpX, tmpY);
}



/**
   * Clears all the components from draw-pad and user can start from new
   * @param a
   * @param cnc_a
   */
public final void clear(counts a,connections cnc_a)
{
    a.reset();
    cnc_a.clear();
    this.removeAll();
    this.reset();
    repaint();
}


/**
```

```
     * reset to default so that we can add other component
     */
    public void reset(  )
    {
        oldX=oldY=tmpX=tmpY=currentX=currentY=0;
        desn=true;
        this.default_mouse();
        if(!locked && !Pad_connections.getLine_logs().isEmpty())
        {
            System.out.println("REMOVED");
            Pad_connections.remove_line_i(Pad_connections.getLine_logs().size()-
1);
            //make the previous line as locked i.e blue-line
            locked=true;
        }
        repaint();
    }


    /**
     * selecting
     * @param a
     */
    public void select(final counts a, connections b)
    {

        this.reset();
        Pad_counts=a;
        Pad_connections=b;
        this.addMouseListener(slct);
```

```
    this.addMouseMotionListener(slct);
}



/**
 * Change the location of i-th component to p Point
 * @param i
 * @param p
 */
public void change (int i,Point p)
{
    //System.out.println(p);
    this.getComponent(i).setLocation(p);
    Pad_connections.update(Pad_counts);
    repaint();
}


/**
 * sets mouse to the default condition
 */
public void default_mouse(  )
{
    this.set_default_cursor();
    this.remove_all_mouse_listeners();
}



/**
 * Removes all MouseListeners and MouseMotionListeners
 */
```

```
public void remove_all_mouse_listeners()
{
    for(MouseListener a : this.getMouseListeners())
        this.removeMouseListener(a);
    for(MouseMotionListener a : this.getMouseMotionListeners())
        this.removeMouseMotionListener(a);
}

/**
 * this enables user to add new component of compnentType in the draw-pad
 * @param cmpType
 * @param a the counts of this instance of MainEditor
 */
public void add_componentType( final componentType cmpType, final counts a )
{
    this.reset();
    this.Pad_counts=a;

    MouseListener add_component= new MouseAdapter(){
        @Override
        public void mouseClicked(MouseEvent e)
        {

                update(cmpType,e.getPoint(),a);
        }
    };
    this.addMouseListener(add_component);
    //System.out.println("component");
}
```

```
  /**
   * enables user to add wire or line segments in the draw-pad
   * @param a
   * @param b needed to check the vicinity of component
   */
  public void wire(final connections a, counts b)
  {
      this.reset();
      Pad_connections=a;
      Pad_counts=b;

      MouseListener line_connct = new MouseAdapter(){
          int m=0;//to mark the first entry i.e first line-segment of the
current_line
          line current_line=null;
          iopoints temp;
          boolean last_point=false,frst_point=true;//mark that any of the last
point is reached
          int [] ret = new int [2];
            @Override
            public void mousePressed(MouseEvent e)
            {
                mousedragActv=false;
                if(desn && e.getButton()==MouseEvent.BUTTON1)
                {
                    temp=new iopoints();
                    if(vicinityCheck(e.getPoint(),Pad_counts,temp,ret))
                    {

//System.out.println("SuccessP"+temp+","+ret[0]+","+ret[1]);
```

```
                        oldX = temp.getLocation().x;
                        oldY = temp.getLocation().y;
                        current_line=new line();
                        current_line.add_line_seg(temp.getLocation());
                        //here current_line is added temporarily by accessing
getLine_logs()
                        //so current_line is not getting a valid id, id remains
(-1)
                        current_line.set_start_param(ret[0], ret[1]);
                        Pad_connections.getLine_logs().add(current_line);
                        m++;//mark that it's not the first line-segment of
current_line
                        desn=false;
                        //make the current_line unlocked i.e red-line
                        locked=false;
                        //System.out.println(oldX+","+oldY+"--
"+currentX+","+currentY+"=Press"+','+","+locked);

//System.out.println("P"+Pad_connections.getLine_logs().size());
                        frst_point=false;//fisrt point is covered
                    }
                }
            }

            //At Mouse-Release we draw the line actually
            @Override
            public void mouseReleased (MouseEvent e)
            {
                mousedragActv=false;
                currentX=e.getX();
                currentY=e.getY();
                if(desn==false && e.getButton()==MouseEvent.BUTTON1)
```

```
                {
                    //if   it's   not   the   first   then   remove   the   last-
added(unlocked) line
                    if(m>0/* && !frst_point*/)
                    {
                        //System.out.println("RREMOVED");

Pad_connections.remove_line_i(Pad_connections.getLine_logs().size()-1);
                    }
                    iopoints temp1= new iopoints();
                    if(m!=0)
                    {

                        if(vicinityCheck(e.getPoint(),Pad_counts,temp1,ret)   &&
temp1.equals(temp))
                        {
                            frst_point=true;
                            //here   current_line   is   added   temporarily   by
accessing getLine_logs()
                            //so  current_line  is  not  getting  a  valid  id,  id
remains (-1)
                            Pad_connections.getLine_logs().add(current_line);
                            //System.out.println("R+frst");
                            if(m>1)  taskbr.setText("First  and  last  point  of
Wire or line can't be same.");
                            else taskbr.setText("Choose a mark point to end the
Wire or line");
                        }
                        else
if(vicinityCheck(e.getPoint(),Pad_counts,temp,ret))
                        {
                            frst_point=false;
                            current_line.add_line_seg(temp.getLocation());
```

```
//System.out.println("SuccessR"+temp+","+ret[0]+","+ret[1]);
                            //here only, current_line is added permanently and
gets a valid id
                            current_line.set_end_param(ret[0], ret[1]);
                            Pad_connections.add_line(current_line);
                            //make this line locked i.e blue-line
                            locked=true;

//System.out.println("C"+Pad_connections.getLine_logs().size());
                            //repaint to see the effect
                            repaint();
                            taskbr.setText("Wire is added.");
                            //reset so that to add a new wire or line user have
to click the button again
                            reset();
                        }
                        else
                        {
                            current_line.add_line_seg(e.getPoint());
                            oldX=currentX;
                            oldY=currentY;
                            //replace with the new
                            //here   current_line   is   added   temporarily   by
accessing getLine_logs()
                            //so current_line is not getting a valid id, id
remains (-1)
                            Pad_connections.getLine_logs().add(current_line);
                            //make this current_line unlocked i.e red-line
                            locked=false;
                            frst_point=false;
                            taskbr.setText("Choose a mark point to end the Wire
or line");
```

```
                        }
                        m++;
                    }

//System.out.println("R"+Pad_connections.getLine_logs().size());
                    desn=false;
                }
                //System.out.println(oldX+","+oldY+"--
"+currentX+","+currentY+"=Release,"+locked);
                repaint();
            }
    };

    /**
       * This listener is to show the drawing line or wire when dragged
       */
    MouseMotionListener line_show= new MouseMotionAdapter(){
            @Override
            public void mouseDragged(MouseEvent e)
            {
                if(desn==false)
                {
                    tmpX = e.getX();
                    tmpY = e.getY();
                    mousedragActv=true;
                }
                repaint();
            }
            @Override
            public void mouseMoved(MouseEvent e)
            {
```

```
                mousedragActv=false;
            }
    };
      this.addMouseListener(line_connct);
      this.addMouseMotionListener(line_show);
}

/**
 * search among the component in a
 * @param a search in a
 * @param ret_point just like isVicinity in component
 * @return just like isVicinity in component
 */
public boolean vicinityCheck(Point p,counts a, iopoints ret_point,int [] ret)
{
    //int [] ret = new int[2];
    for(component cntr : a.getComponent_list())
    {
        if(cntr.isVicinity(p, ret_point,ret))
        {
            ret_point.change(ret_point);
            //System.out.println("W"+ret[0]+","+ret[1]);
            return true;
        }
    }
    ret_point=null;
    return false;
}

/**
```

```
    * Loads the file with path
    * @param path the required path of the file to be loaded
    * @param a
    */
   public void load( String path,counts a, connections cnc_a )
   {
       int cmp_id,x_pos = 0,y_pos = 0,current_comp_type_id = 0;
       String pkg_type = null;
       // create a DOMParser
       DOMParser parser=new DOMParser();
         try {
             parser.parse(path);
         } catch (SAXException ex) {
             Logger.getLogger(Pad_Draw.class.getName()).log(Level.SEVERE,   null,
ex);
         } catch (IOException ex) {
             Logger.getLogger(Pad_Draw.class.getName()).log(Level.SEVERE,   null,
ex);
         }
       // get the DOM Document object
      Document doc=parser.getDocument();

      NodeList nodelist = doc.getElementsByTagName("comp");
      for(int i=0;i<nodelist.getLength();i++)
      {
          //getting single component
          Node node = nodelist.item(i);

          //getting the id
          NamedNodeMap id = node.getAttributes();
          cmp_id=Integer.parseInt(id.getNamedItem("id").getNodeValue());
```

```
            System.out.println(cmp_id);

            // get the child nodes of a single node
            NodeList childnodelist = node.getChildNodes();
            //getting the package & componentType
            for(int j=0;j<childnodelist.getLength();j++)
            {
                Node childnode = childnodelist.item(j);
                Node textnode = childnode.getFirstChild();
                String childnodename = childnode.getNodeName();
                //System.out.println(childnodename);
                if(childnodename.equals("comp_type_id"))
                {
current_comp_type_id=Integer.parseInt(textnode.getNodeValue().trim());
                    //System.out.println(current_comp_type_id);
                }
                else if(childnodename.equals("pkg_name"))
                {
                    pkg_type=(String)textnode.getNodeValue().trim();
                    //System.out.println(pkg_type);
                }
                else if(childnodename.equals("position"))
                {
                    NodeList children=childnode.getChildNodes();
                    for(int k=0;k<children.getLength();k++)
                    {
                        Node pos_child = children.item(k);
                        Node text_pos = pos_child.getFirstChild();
                        String pos_name = pos_child.getNodeName();
```

```
                    if(pos_name.equals("x"))
                    {
x_pos=Integer.parseInt(text_pos.getNodeValue().trim());
                        //System.out.println(x_pos);
                    }
                    else if(pos_name.equals("y"))
                    {
y_pos=Integer.parseInt(text_pos.getNodeValue().trim());
                        //System.out.println(y_pos);
                    }
                }
            }
        }
        try
        {
            int j=0;
            circuits.componentType current_comp_type;
            for(j=0;j<a.getPackage_list().size();j++)
                if(a.getPackage_list().get(j).getName().equals(pkg_type))
                    break;
current_comp_type=a.getPackage_list().get(j).getComponentType_list().get(curren
t_comp_type_id-1);
            this.update(current_comp_type,new Point(x_pos,y_pos),a);
        }
        catch(java.lang.IndexOutOfBoundsException ex)
        {
            JOptionPane.showMessageDialog(this, "The package needed for the
components is not installed.", path,JOptionPane.ERROR_MESSAGE);
            break;
```

```
            }
        }
        //System.out.println(nodelist.getLength());
    }



    /**
     * updates the Pad with a new component added on the pad
     */
    private void update(componentType cmpType,Point p,counts a)
    {
        JLabel new_label;
        component current=new component(cmpType,p,a);
        new_label=new JLabel(current.getType().getImg().getIcon());

//System.out.println(current.getPosition().x+","+current.getPosition().y+","+cu
rrent.getType().getWidth()+","+current.getType().getHeight());

        new_label.setBounds(current.getPosition().x,      current.getPosition().y,
current.getType().getWidth(), current.getType().getHeight());
        this.add(new_label);
        //System.out.println("Here");
        repaint();
    }



    /**
     * Sets the background grids
     * @param         type_of_view depending of 'type_of_view', the grids will be
drawn.
     * 0 - no grid, 1 to 3 - small to large
     */
    public void view( int type_of_view ) throws IOException
```

```
  {
      switch(type_of_view)
      {
          case 0:
              bgImage=null;
              break;
          case 1:
              bgImage=ImageIO.read(new File("sml_grid.jpg"));
              bgImage=bgImage.getScaledInstance(this.getWidth(),
this.getHeight(),Image.SCALE_FAST);
              break;
          case 2:
              bgImage=ImageIO.read(new File("med_grid.jpg"));
              bgImage=bgImage.getScaledInstance(this.getWidth(),
this.getHeight(),Image.SCALE_FAST);
              break;
          case 3:
              bgImage=ImageIO.read(new File("lrg_grid.jpg"));
              bgImage=bgImage.getScaledInstance(this.getWidth(),
this.getHeight(),Image.SCALE_FAST);
              break;
          default:
              bgImage=null;
      }
      repaint();
  }

  /**
   * save the current file
   */
  public void save(  )
```

```
    {
    }

    /**
     * sets the move cursor
     */
     public void set_move_cursor()
      {
          this.setCursor(Cursor.getPredefinedCursor(Cursor.MOVE_CURSOR));
      }

     /**
      * sets the default mouse cursor
      */
      public void set_default_cursor()
      {
          this.setCursor(Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
      }

}
```

A.3 Class counts:

```
package circuits;


import circuits.package_cls;
import java.util.ArrayList;



/**
 * Class counts
 * Keeps track of the total component's, componentType's,package_cls's
 */
public class counts {


  //
  // Fields
  //


  /**
   * Total List of the component
   */
  public ArrayList<component> component_list;
  /**
   * Total List of the componentType
   */
  public ArrayList<componentType> componentType_list;
  /**
   * Total List of the Packages
```

```
 */
public ArrayList<package_cls> package_list;
/**
 * generates ID of the cmp
 */
public int cmp_id;
/**
 * generates ID of the cmpType
 */
public int cmpType_id;


//
// Constructors
//
public counts ()
{
    this.setCmpType_id(0);
    this.setCmp_id(0);
    this.componentType_list=new ArrayList<componentType>();
    this.component_list=new ArrayList<component>();
    this.package_list=new ArrayList<package_cls>();
};


//
// Methods
//



//
// Accessor methods
```

```
//


/**
 * Set the value of component_list
 * Total List of the component
 * @param newVar the new value of component_list
 */
/*public void setComponent_list ( component newVar )
{
  component_list = newVar;
}*/


/**
 * Get the value of component_list
 * Total List of the component
 * @return the value of component_list
 */
public ArrayList<component> getComponent_list ( ) {
  return component_list;
}


/**
 * Set the value of componentType_list
 * Total List of the componentType
 * @param newVar the new value of componentType_list
 */
/*public void setComponentType_list ( componentType newVar ) {
  componentType_list = newVar;
}*/
```

```
/**
 * Get the value of componentType_list
 * Total List of the componentType
 * @return the value of componentType_list
 */
public ArrayList<componentType> getComponentType_list ( ) {
  return componentType_list;
}

/**
 * Set the value of package_list
 * Total List of the componentType
 * @param newVar the new value of package_list
 */
/*public void setPackage_list ( package_cls newVar ) {
  package_list = newVar;
}*/

/**
 * Get the value of package_list
 * Total List of the componentType
 * @return the value of package_list
 */
public ArrayList<package_cls> getPackage_list ( ) {
  return package_list;
}

/**
 * Set the value of component_id
 * generates ID of the component
```

```
   * @param newVar the new value of component_id
   */
  public final void setCmp_id ( int newVar ) {
    cmp_id = newVar;
  }

  /**
   * Get the value of component_id
   * generates ID of the component
   * to get the ID, one has to add the component first and the request for the
id
   * @return the value of component_id
   */
  public int getCmp_id ( ) {
    return cmp_id;
  }

  /**
   * Set the value of cmpType_id
   * generates ID of the cmpType
   * @param newVar the new value of cmpType_id
   */
  public final void setCmpType_id ( int newVar ) {
    cmpType_id = newVar;
  }

  /**
   * Get the value of cmpType_id
   * generates ID of the cmpType
   * to get the ID, one has to add the componentType first and the request for
the id
```

```
 * @return the value of cmpType_id
 */
public int getCmpType_id ( ) {
  return cmpType_id;
}

//
// Other methods
//

/**
 * resets all to zero
 */
public void reset(  )
{
   cmp_id=0;
   cmpType_id=0;
   this.componentType_list.clear();
   this.component_list.clear();
   this.package_list.clear();
}



/**
 * adds new package
 * @param        desired_install
 */
public void install_new_package( package_cls desired_install )
{
    this.package_list.add(desired_install);
```

```
    //System.out.println("Hello,");
}



/**
 * removes package
 * @param        desired_uninstall
 */
public void uninstall_package( package_cls desired_uninstall )
{
    this.package_list.remove(desired_uninstall);
}



/**
 * adds new componentType
 * @param        x
 */
public void add_new_componentType( componentType x )
{
    this.componentType_list.add(x);
    this.setCmpType_id(this.getCmpType_id()+1);
}



/**
 * removes componentType
 * @param        x
 */
public void remove_componentType( componentType x )
```

```
{
    this.componentType_list.remove(x);
    //this.setCmpType_id(this.getCmpType_id()-1);
}



/**
 * adds new component
 * @param        x
 */
public void add_new_component( component x )
{
    this.component_list.add(x);
    this.setCmp_id(this.getCmp_id()+1);
}



/**
 * removes component
 * @param        x
 */
public void remove_component( component x )
{
    this.component_list.remove(x);
    //this.setCmp_id(this.getCmp_id()-1);
}


/**
 * if no component and no componentType is there, then returns true i.e empty
 *
```

```
     */
    public boolean is_empty()
    {
        if(this.getCmp_id()==0)
            return true;
        else
            return false;
    }

    /**
     * Returns the component with id x
     * @param x
     * @return the component with x id
     */
    public component get_component_by_id(int x)
    {
        for(component a:this.getComponent_list())
        {
            if(a.getId()==x)
            {
                return a;
            }
        }
        return null;
    }

}
```

A.4 Class connections:

```
package circuits;


import java.util.ArrayList;



/**
 * Class connections
 * logs the connections
 */
public class connections {

  //
  // Fields
  //

  public ArrayList<line> line_logs;
  public int cnt_line_id;

  //
  // Constructors
  //
  public connections () {
      line_logs=new ArrayList<line>();
      this.setCnt_line_id(0);
  };

  //
  // Methods
```

```
//



//
// Accessor methods
//

/**
 * Set the value of line_logs
 * @param newVar the new value of line_logs
 */
/*public void setLine_logs ( line newVar ) {
  line_logs = newVar;
}*/

/**
 * Get the value of line_logs
 * @return the value of line_logs
 */
public ArrayList<line> getLine_logs ( ) {
  return line_logs;
}

/**
 * Set the value of cnt_line_id
 * @param newVar the new value of cnt_line_id
 */
public final void setCnt_line_id ( int newVar ) {
  cnt_line_id = newVar;
}
```

```
/**
 * Get the value of cnt_line_id
 * @return the value of cnt_line_id
 */
public int getCnt_line_id ( ) {
  return cnt_line_id;
}

//
// Other methods
//

/**
 * @return       line
 * @param        new_line
 */
public void add_line( line new_line )
{
    this.setCnt_line_id(this.getCnt_line_id()+1);
    new_line.setId(this.getCnt_line_id());
    this.getLine_logs().add(new_line);
}



/**
 * @param        old_line
 */
public void remove_line( line old_line )
{
```

```
        this.getLine_logs().remove(old_line);
    }


    /**
     * removes the i-th line from line_logs
     * @param i the i-th line to remove
     */
    public void remove_line_i(int i)
    {
        this.getLine_logs().remove(i);
    }



    /**
     * clear all the line_logs and sets to default
     */
    public void clear(  )
    {
        line_logs.clear();
        this.setCnt_line_id(0);
    }


    public void update(counts a)
    {
        for(line cntr:this.getLine_logs())
        {
            cntr.update_end_point_params(a);
        }
    }
}
```

A.5 Class line:

```
package circuits;


import java.util.ArrayList;



/**
 * Class connections
 * logs the connections
 */
public class connections {

  //
  // Fields
  //

  public ArrayList<line> line_logs;
  public int cnt_line_id;

  //
  // Constructors
  //
  public connections () {
      line_logs=new ArrayList<line>();
      this.setCnt_line_id(0);
  };

  //
  // Methods
```

```
//



//
// Accessor methods
//

/**
 * Set the value of line_logs
 * @param newVar the new value of line_logs
 */
/*public void setLine_logs ( line newVar ) {
  line_logs = newVar;
}*/

/**
 * Get the value of line_logs
 * @return the value of line_logs
 */
public ArrayList<line> getLine_logs ( ) {
  return line_logs;
}

/**
 * Set the value of cnt_line_id
 * @param newVar the new value of cnt_line_id
 */
public final void setCnt_line_id ( int newVar ) {
  cnt_line_id = newVar;
}
```

```
/**
 * Get the value of cnt_line_id
 * @return the value of cnt_line_id
 */
public int getCnt_line_id ( ) {
  return cnt_line_id;
}

//
// Other methods
//

/**
 * @return       line
 * @param        new_line
 */
public void add_line( line new_line )
{
    this.setCnt_line_id(this.getCnt_line_id()+1);
    new_line.setId(this.getCnt_line_id());
    this.getLine_logs().add(new_line);
}


/**
 * @param        old_line
 */
public void remove_line( line old_line )
{
```

```
        this.getLine_logs().remove(old_line);
    }


    /**
     * removes the i-th line from line_logs
     * @param i the i-th line to remove
     */
    public void remove_line_i(int i)
    {
        this.getLine_logs().remove(i);
    }



    /**
     * clear all the line_logs and sets to default
     */
    public void clear(  )
    {
        line_logs.clear();
        this.setCnt_line_id(0);
    }


    public void update(counts a)
    {
        for(line cntr:this.getLine_logs())
        {
            cntr.update_end_point_params(a);
        }
    }
}
```

A.6 Class componentType:

```
package circuits;


import java.awt.Image;

import java.awt.Label;

import java.awt.Point;

import java.util.ArrayList;

import java.util.Arrays;

import javax.swing.ImageIcon;

import javax.swing.JLabel;


/**

 * Class componentType

 * This class is to define one componentType of our circuit editor.

 */

public class componentType {


  //

  // Fields

  //


  /**

   * height of the componentType

   */

  public int height;

  /**

   * width of componentType
```

```
 */
public int width;
/**
 * the Label of the componentType
 */
public JLabel img;
/**
 * the Image of the componentType
 */
public Image type_img;
/**
 * mark_points with respect to Top-Left as (0,0)
 */
public ArrayList<iopoints> mark_points;
/**
 * Unique id.
 */
public int id;
/**
 * In which package it belongs
 */
public package_cls type_pkg;

//
// Constructors
//
/**
 * initialized with 'pkg' package_cls and updates the counts.
 * @param        pkg
 * @param        a needed so that counts can be updated.
```

```
   * @param        path needed to create the image
   */
  public componentType( package_cls pkg, counts a, String path, iopoints[]
marks )
  {
      this.setType_pkg(pkg);
      a.add_new_componentType(this);
      this.setId(a.getCmpType_id());
      this.createImageIcon(path, "componentType");
      this.setMark_points(marks);
      this.img.setVisible(true);
      this.img.setOpaque(true);
      this.img.setEnabled(true);
      this.img.setDoubleBuffered(true);
      System.out.println(this.getMark_points());
  }

  //
  // Methods
  //



  //
  // Accessor methods
  //

  /**
   * Set the value of height
   * height of the componentType
   * @param newVar the new value of height
```

```
 */
private void setHeight ( int newVar ) {
  height = newVar;
}

/**
 * Get the value of height
 * height of the componentType
 * @return the value of height
 */
public int getHeight ( ) {
  return height;
}

/**
 * Set the value of width
 * width of componentType
 * @param newVar the new value of width
 */
private void setWidth ( int newVar ) {
  width = newVar;
}

/**
 * Get the value of width
 * width of componentType
 * @return the value of width
 */
public int getWidth ( ) {
  return width;
```

```
}

/**
 * Set the value of img
 * the image of the componentType
 * @param newVar the new value of img
 */
public void setImg ( JLabel newVar ) {
  img = newVar;
}

/**
 * Get the value of img
 * the image of the componentType
 * @return the value of img
 */
public final JLabel getImg ( ) {
  return img;
}

/**
 * Set the value of img
 * the image of the componentType
 * @param newVar the new value of img
 */
public void setType_img ( Image newVar ) {
  type_img = newVar;
}

/**
```

```
 * Get the value of img
 * the image of the componentType
 * @return the value of img
 */
public final Image getType_Img ( ) {
  return type_img;
}

/**
 * Set the value of mark_points
 * mark_points with respect to Top-Left as (0,0)
 * @param newVar the new value of mark_points
 */
private void setMark_points ( iopoints[] newVar ) {
  mark_points = new ArrayList<iopoints>(Arrays.asList(newVar));
}

/**
 * Get the value of mark_points
 * mark_points with respect to Top-Left as (0,0)
 * @return the value of mark_points
 */
public ArrayList<iopoints> getMark_points ( ) {
  return mark_points;
}


/**
 * Set the value of id
 * Unique id.
```

```
 * @param newVar the new value of id
 */
public final void setId ( int newVar ) {
  id = newVar;
}

/**
 * Get the value of id
 * Unique id.
 * @return the value of id
 */
public int getId ( ) {
  return id;
}

/**
 * Set the value of type_pkg
 * In which package it belongs
 * @param newVar the new value of type_pkg
 */
public final void setType_pkg ( package_cls newVar ) {
  type_pkg = newVar;
}

/**
 * Get the value of type_pkg
 * In which package it belongs
 * @return the value of type_pkg
 */
public package_cls getType_pkg ( ) {
```

```
  return type_pkg;
}

/**
 * Get the name of the package
 * @return the value of type_pkg.getName()
 */
public String getType_pkg_name ( ) {
  return type_pkg.getName();
}

//
// Other methods
//

/**
 * creates the image or the Label
 * @param        path
 * @param        description
 */
public final void createImageIcon( String path, String description )
{
    ImageIcon a;
    if (path != null)
    {
        a = new ImageIcon(path, description);
    }
    else
    {
        System.err.println("Couldn't find file: " + path);
```

```
            a = null;
        }
        this.setType_img(a.getImage());
        this.setHeight(a.getIconHeight());
        this.setWidth(a.getIconWidth());
        this.setImg(new JLabel(a));
    }

}
```

A.7 Class component:

```
package circuits;


import java.awt.*;
import java.awt.geom.Rectangle2D;
import java.util.ArrayList;
import javax.swing.JLabel;
// import MainEditor.


/**
 * Class component
 * This class is to define one component of our circuit editor.
 *
 * Like a single Two-input OR gate, a single 3-input NAND gate.
 */
public class component {


  //
  // Fields
  //


  /**
   * the Left-most point of the Label(Image) or the component.
   */
  public Point position;
  /**
   * type of the componentType
   */
  public componentType type;
```

```
/**
 * the unique ID, which will be available from counts.
 */
public int id;


public JLabel label;


//
// Constructors
//
  /**
 * componentType will be initialized and counts will be updated.
 * @param        cmp
 * @param        a to update the counts and to get the id.
 */
public component( componentType cmp,Point pos,counts a )
{
     this.setType(cmp);
     this.setPosition(pos);
     a.add_new_component(this);
     this.setId(a.getCmp_id());
     this.label=new JLabel(this.getType().getImg().getIcon());
}


//
// Methods
//



//
```

```
// Accessor methods
//

/**
 * Set the value of position
 * the Left-most point of the Label(Image) or the component.
 * @param newVar the new value of position
 */
public final void setPosition ( Point newVar ) {
  position = newVar;
}

/**
 * Get the value of position
 * the Left-most point of the Label(Image) or the component.
 * @return the value of position
 */
public Point getPosition ( ) {
  return position;
}

/**
 * Set the value of type
 * type of the componentType
 * @param newVar the new value of type
 */
public final void setType ( componentType newVar ) {
  type = newVar;
}
```

```
/**
 * Get the value of type
 * type of the componentType
 * @return the value of type
 */
public final componentType getType ( ) {
  return type;
}

/**
 * Set the value of id
 * the unique ID, which will be available from counts.
 * @param newVar the new value of id
 */
public final void setId ( int newVar ) {
  id = newVar;
}

/**
 * Get the value of id
 * the unique ID, which will be available from counts.
 * @return the value of id
 */
public int getId ( ) {
  return id;
}

//
// Other methods
//
```

```
/**
 * to move 'x' in X-direction
 * @param        x
 */
public void addX( int x )
{
    this.setPosition(new Point(this.getPosition().x+x,this.getPosition().y));
}



/**
 * to move 'y' in Y-direction
 * @param        y
 */
public void addY( int y )
{
    this.setPosition(new Point(this.getPosition().x,this.getPosition().y+y));
}



/**
 * real-time mark_points of the component.
 * will be available from package and componentType
 * @return  an ArrayList of iopoints with the calculated mark_points
 */
public ArrayList<iopoints> get_mark_point(){
    ArrayList<iopoints> rsltd = new ArrayList<iopoints>();
    for(iopoints p : this.getType().getMark_points())
```

```
      {
          iopoints a = new iopoints();
          //after adding the offset from componentType to the component's X &
Y-Coordinate
          a.setLocation(p.getX()+this.getPosition().getX(),
p.getY()+this.getPosition().getY());
          a.setIo(p.getIo());
          rsltd.add(a);
      }
      return rsltd;
  }

  /**
   * Get the i-th Mark Point
   * @param i desired Mark Point
   * @return Returns the desired Mark Point in iopoints class
   */
  public iopoints get_mark_points (int i) {
      return this.get_mark_point().get(i);
  }

  /**
   * If a Point is within the covered area of component returns true.
   * @param point
   * @return true, if point is within the component area. false, otherwise
   */
    public boolean isHit(Point point)
    {
        Rectangle2D.Float myarea= new Rectangle2D.Float();
        myarea.setRect(this.getPosition().x,              this.getPosition().y,
this.getType().getWidth(), this.getType().getHeight());
```

```
        if (myarea.contains(point))
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    /**
     * returns true if the p Point is within vicinity of any mark_point and
with the matched Mark Point at ret_mark_point
     * vicinity means within -8,+8 of the center point
     * @param p the point to be checked
     * @param ret_mark_point if true, then desired Mark Point is stored. null,
otherwise.
     * @param ret[0] if method returns true, stores the id of the component. if
false, stores -1
     * @param ret[1] if method returns true, stores Mark Point Number(counter).
if false, stores -1
     * @return true, if p is within the vicinity of any mark_points
     */
    public boolean isVicinity(Point p, iopoints ret_mark_point,int [] ret)
    {
        Rectangle2D.Float myarea= new Rectangle2D.Float();
        int i=0;
        for(iopoints cntr : this.get_mark_point())
        {
            myarea.setRect(cntr.getX()-8,cntr.getY()-8,16,16);
            if(myarea.contains(p))
            {
```

```
                ret_mark_point.change(cntr);
                ret[0]=this.getId();
                ret[1]=i;
                //System.out.println("P"+ret[0]+","+ret[1]);
                return true;
            }
            i++;
        }
        ret[0]=ret[1]=-1;
        ret_mark_point=null;
        return false;
    }
}
```

A.8 Class package_cls:

```
package circuits;


import java.awt.Point;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.util.ArrayList;

import java.util.Enumeration;

import java.util.logging.Level;

import java.util.logging.Logger;

import java.util.zip.*;

import javax.swing.JOptionPane;

import org.w3c.dom.*;

import org.apache.xerces.parsers.DOMParser;

import org.xml.sax.SAXException;


/**
 * Class package_cls
 */
public class package_cls {


  //
  // Fields
  //


  public ArrayList<componentType> componentType_list;

  public String name;
```

```
//
// Constructors
//
public package_cls( String path, counts a )
{
    this.componentType_list=new ArrayList<componentType>();
    a.install_new_package(this);
    this.parse(path,a);
}

//
// Methods
//



//
// Accessor methods
//

/**
 * Set the value of componentType_list
 * @param newVar the new value of componentType_list
 */
/*public void setComponentType_list ( componentType newVar ) {
  componentType_list = newVar;
}*/

/**
```

```
 * Get the value of componentType_list
 * @return the value of componentType_list
 */
public ArrayList<componentType> getComponentType_list ( ) {
  return this.componentType_list;
}

/**
 * Set the value of name
 * @param newVar the new value of name
 */
public void setName ( String newVar ) {
  name = newVar;
}

/**
 * Get the value of name
 * @return the value of name
 */
public String getName ( ) {
  return name;
}

//
// Other methods
//



/**
 */
```

```
  public final void parse( String path, counts a )

  {

      String pkg_foldername = null;

      String pkg_name = null;

      int [] id;

      iopoints [][]marks;

      String [] img_addr;

        try {

            String
mypath=System.getProperty("user.home")+File.separator+"SRSCKT"+File.separator+"
pkg";

            //System.out.println(mypath);

            pkg_foldername                                                    =
mypath+File.separator+this.unzipFileIntoDirectory(new     ZipFile(path),     new
File(mypath));



        } catch (IOException ex) {

            Logger.getLogger(package_cls.class.getName()).log(Level.SEVERE,
null, ex);

        }

        File mypkg_dir=new File(pkg_foldername);//getting the foldername

        System.out.println(mypkg_dir.getAbsolutePath());

        File[] fileList=mypkg_dir.listFiles();

        //searching the manifest file

        boolean err=true;

        for(File f:fileList)

        {

            if(f.getName().equals("manifest.xml"))

            {

                err=false;

                break;

            }
```

```
        }
        if(!err)
        {
              // create a DOMParser
              DOMParser parser=new DOMParser();
                try {

parser.parse(mypkg_dir.getAbsolutePath()+File.separator+"manifest.xml");
                } catch (SAXException ex) {

Logger.getLogger(package_cls.class.getName()).log(Level.SEVERE, null, ex);
                } catch (IOException ex) {

Logger.getLogger(package_cls.class.getName()).log(Level.SEVERE, null, ex);
                }
             // get the DOM Document object
             Document doc=parser.getDocument();


             try
             {
                 int i,j;
                 int componentType_size = 0;//counter for the componentType, at
last total number of componentType
                 int
tmp=doc.getElementsByTagName("componentType").getLength();//calculates       the
number of componentType in package
                 //at last, componentType_size=tmp
                 //System.out.println(tmp);
                 id = new int[tmp];
                 img_addr = new String[tmp];
                 marks = new iopoints[tmp][];
                 NodeList nodelist = doc.getElementsByTagName("package");
                 NodeList our_nodes = nodelist.item(0).getChildNodes();
```

```
                for(i=0;i<our_nodes.getLength();i++)
                {
                    Node mynode=our_nodes.item(i);
                    Node txtnode=mynode.getFirstChild();
                    String mynode_name=mynode.getNodeName();
                    if(mynode_name.equals("name"))
                        pkg_name=txtnode.getNodeValue().trim();
                    else if(mynode_name.equals("componentType"))
                    {
                        nodelist=mynode.getChildNodes();//nodelist    can    be
reused as it's not required.
                        //System.out.println(nodelist.getLength());
                        for(j=0;j<nodelist.getLength();j++)
                        {
                            Node child = nodelist.item(j);
                            Node text = child.getFirstChild();
                            String child_name = child.getNodeName();
                            //System.out.println(","+text.getNodeName());
                            if(child_name.equals("id"))
                            {

id[componentType_size]=Integer.parseInt(text.getNodeValue().trim());
                                //System.out.println(id[componentType_size]);
                            }
                            else if(child_name.equals("img"))
                            {

img_addr[componentType_size]=mypkg_dir.getAbsolutePath()+File.separator+text.ge
tNodeValue().trim();

//System.out.println(img_addr[componentType_size]);
                            }
```

```
                            else if(child_name.equals("mark_points"))
                            {
                                NamedNodeMap no=child.getAttributes();
                                //posS - total number of mark_points
                                int
posS=Integer.parseInt(no.getNamedItem("no").getNodeValue());
                                //if(posS==0)
                                    marks[componentType_size]=new
iopoints[posS];

System.out.println("C"+componentType_size+","+posS+","+marks[componentType_size
].length);
                                int  pos_cntr=0;//counter  for  the  number  of
mark_points
                                NodeList mark_children=child.getChildNodes();
                                for(int k=0;k<mark_children.getLength();k++)
                                {
                                    Node pos_node = mark_children.item(k);
                                    Node pos_val = pos_node.getFirstChild();
                                    String pos_name = pos_node.getNodeName();
                                    if(pos_name.equals("position"))
                                    {
                                        marks[componentType_size][pos_cntr]   =
new iopoints();
                                        NodeList
children=pos_node.getChildNodes();
                                        for(int
l=0;l<children.getLength();l++)
                                        {
                                            Node pos_child = children.item(l);
                                            Node        text_child_pos        =
pos_child.getFirstChild();
                                            String       pos_child_name       =
pos_child.getNodeName();
```

```
                                        if(pos_child_name.equals("x"))
                                        {
//System.out.println(Integer.parseInt(text_child_pos.getNodeValue().trim()));
marks[componentType_size][pos_cntr].x=Integer.parseInt(text_child_pos.getNodeVa
lue().trim());
                                        }
                                        else
if(pos_child_name.equals("y"))
                                        {
//System.out.println(Integer.parseInt(text_child_pos.getNodeValue().trim()));
marks[componentType_size][pos_cntr].y=Integer.parseInt(text_child_pos.getNodeVa
lue().trim());
                                        }
                                        else
if(pos_child_name.equals("io"))
                                        {
//System.out.println(Integer.parseInt(text_child_pos.getNodeValue().trim()));
marks[componentType_size][pos_cntr].setIo(Integer.parseInt(text_child_pos.getNo
deValue().trim()));
                                        }
                                    }
//System.out.println("P"+marks[componentType_size][pos_cntr]);
                                    pos_cntr++;
                                }
                            }
                            //System.out.println("Coming Here");
                        }
                    }
```

```
                        //System.out.println("HelloA"+componentType_size);
                        componentType_size++;
                        //System.out.println("HelloB"+componentType_size);
                    }
                }
                //System.out.println("BB"+componentType_size);

init_package_data(pkg_name,componentType_size,marks,img_addr,a);
                //Node txt_node = a_node.getFirstChild();
                //if(a_node.getNodeName().equals("name"))
                  //  pkg_name=txt_node.getNodeValue().trim();

                if(!(new
File(mypkg_dir.getParent()+File.separator+pkg_name)).exists())
                {
                    System.out.println(mypkg_dir.renameTo(new
File(mypkg_dir.getParent()+File.separator+pkg_name)));

//System.out.println(a_node.getNextSibling().getNodeName());
                }
                else
                    System.out.println("Directory already exists!");
            }
            catch(DOMException ex)
            {
                JOptionPane.showMessageDialog(null,  "Error  in  XML  Parsing.
Check your package","Error!",JOptionPane.ERROR_MESSAGE);
            }
        }
        else
            JOptionPane.showMessageDialog(null,        "manifest.xml        not
found!","Error!",JOptionPane.ERROR_MESSAGE);
  }
```

```
  public void add_componentType(componentType a)
  {
      this.componentType_list.add(a);
  }


   public      String      unzipFileIntoDirectory(ZipFile      zipFile,      File
jiniHomeParentDir) {
    Enumeration files = zipFile.entries();
    String foldername=((ZipEntry) files.nextElement()).getName();
    File f = null;
    FileOutputStream fos = null;

    while (files.hasMoreElements()) {
        try {
        ZipEntry entry = (ZipEntry) files.nextElement();
        InputStream eis = zipFile.getInputStream(entry);
        byte[] buffer = new byte[1024];
        int bytesRead = 0;

        f  =  new  File(jiniHomeParentDir.getAbsolutePath()  +  File.separator  +
entry.getName());

        if (entry.isDirectory()) {
          f.mkdirs();
          /*System.out.println("A:"+f.getAbsolutePath());
          System.out.println("B:"+f.getPath());
          System.out.println("C:"+f.getParent());*/
          continue;
        } else {
```

```
            /*System.out.println(f.getAbsolutePath());
            System.out.println(f.getPath());
            System.out.println(f.getParent());*/
            f.getParentFile().mkdirs();
            f.createNewFile();
            /*System.out.println("A1:"+f.getAbsolutePath());
            System.out.println("B1:"+f.getPath());
            System.out.println("C1:"+f.getParent());*/
          }

          fos = new FileOutputStream(f);

          while ((bytesRead = eis.read(buffer)) != -1) {
            fos.write(buffer, 0, bytesRead);
          }
        } catch (IOException e) {
          continue;
        } finally {
          if (fos != null) {
            try {
              fos.close();
            } catch (IOException e) {
              // ignore
            }
          }
        }
      }
      return foldername;
    }
```

```
   /**
    * makes the component List of the package,sets the package_name
    * @param pkg_name the name of the package - Unique
    * @param marks the mark points
    * @param img_addr the image addresses
    * @param a counts for the componentType class updating
    */
    private          void          init_package_data(String          pkg_name,int
no_componentType,iopoints[][] marks, String[] img_addr,counts a)
    {
        this.setName(pkg_name);
        for(int i=0;i<no_componentType;i++)
        {
            componentType                                                current=new
componentType(this,a,img_addr[i],marks[i]);
            this.getComponentType_list().add(current);
            //System.out.println("AAA"+i+","+marks[i]);
        }
    }

}
```

A.9 Class iopoints:

```
package circuits;


import java.awt.Point;



/**
 * Class iopoints
 */
public class iopoints extends Point{


  //
  // Fields
  //


  /**
     * if io=0 then input
     * if io=1 then output
  */
  public int io;


  //
  // Constructors
  //
  public iopoints () {
      io=0;//by default it's an input point
  };


  /**
```

```
 * copy constructor but method-name is change
 * @param newVar
 */
public void change(iopoints newVar)
{
    this.setIo(newVar.getIo());
    this.setLocation(newVar.getLocation());
}

//
// Methods
//



//
// Accessor methods
//

/**
 * Set the value of io
 * @param newVar the new value of io
 */
public void setIo ( int newVar ) {
  io = newVar;
}

/**
 * Get the value of io
 * @return the value of io
 */
```

```
  public int getIo ( ) {
    return io;
  }

  //
  // Other methods
  //
}
```