# CSSE1001

23/10/2009

Assignment 3 – Final Design Document

**Name: Justin Mancinelli**

**Student Number: 42094353**

**Project Title: Antioch**

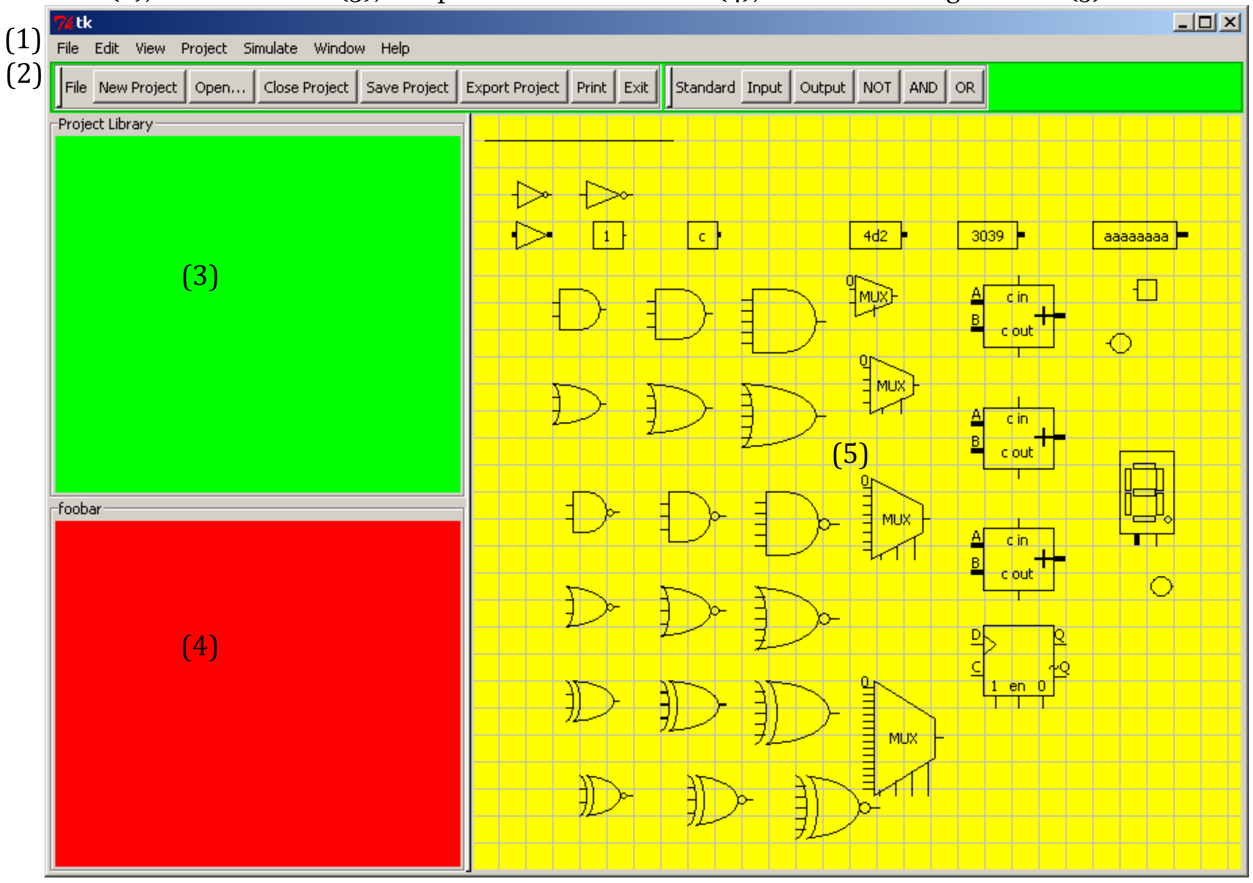
## 1. Description

Logical circuit design is a fundamental aspect of creating computer hardware which can perform different tasks. In the class CSSE1000 at the University of Queensland we use a program called Logisim which is a GUI for logical circuit design and simulation written in Java. While the aspects of logical circuit design are implemented very well within Logisim, I believe the user interaction leaves some to be desired. The purpose of Antioch is to rewrite many of the important aspects of Logisim in Python (since that is the language of this course) and enhance the user interaction behavior.

## 2. User Interface

Antioch will use Tkinter for GUI programming. Tkinter comes bundled with Python and offers a simple API for all of the user interaction needed as specified below.

Antioch's main window is partitioned into 5 visually distinct sections: the menu bar (1), the toolbar frame(2), libraries frame(3), component attributes frame(4), and circuit design canvas (5).

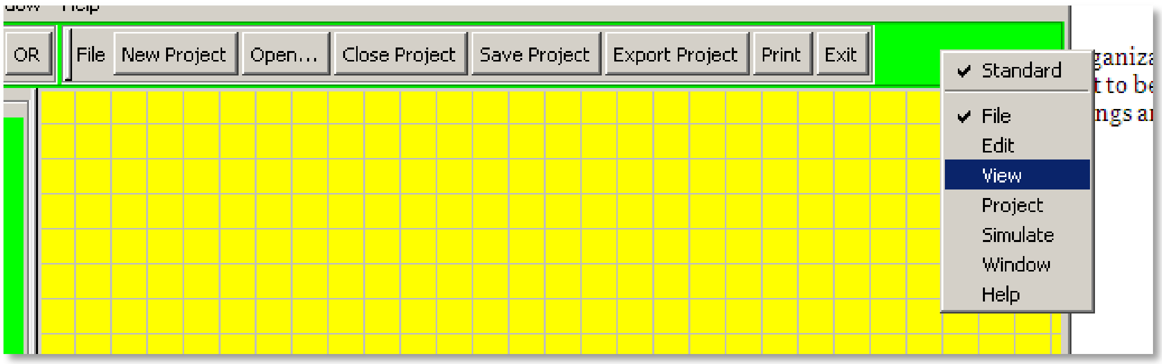

(1)
(2)
tk
File Edit View Project Simulate Window Help
File New Project Open... Close Project Save Project Export Project Print Exit
Standard Input Output NOT AND OR
Project Library
(3)
foobar
(4)
(5)
4d2
3039
aaaaaaaa
MUX
c in
c out
D Q C 1 en 0

**The menu bar:** (Only File→Exit has been implemented at this time)

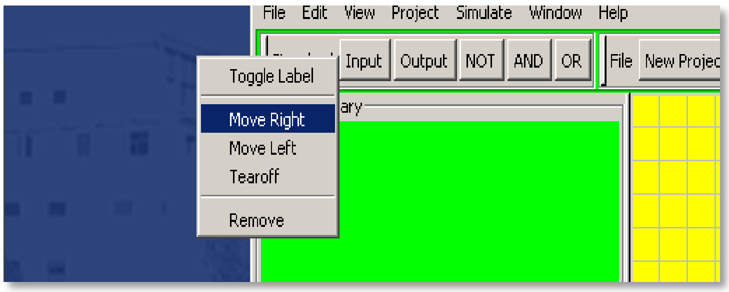
| | |
|---|---|
| 1. File | |
| a. New Project | Create  a new project |
| b. Open… | Open any file Antioch understands |
| c. Close Project | Close the current project |
| d. Save Project | Save the current project |
| e. Export Project… | Export the current project to another format |
| f. Print… | Print as desired |
| g. Exit | Exit the program |
| 2. Edit | |
| a. Undo | Undo last action |
| b. Redo | Redo what was undone |
| c. Cut | Cut selection |
| d. Copy | Copy selection |
| e. Paste | Paste onto canvas |
| f. Delete | Delete selection |
| g. Duplicate | Duplicate selection |
| h. Select All | Select everything on the canvas |
| i. Deselect All | Deselect everything on the canvas |
| 3. View | |
| a. Toolbars-> | Select which toolbars are viewable |
| i. Standard | Set of standard tools |
| 1. Selection | Allow selection of canvas objects and the drawing of wires |
| 2. Text | Write text onto canvas |
| 3. Rectangle | Draw rectangle onto canvas |
| 4. Ellipse | Draw ellipse onto canvas |
| ii. Menus-> | Make toolbars out of menus |
| 1. … | |
| iii. Component Libraries-> | Make toolbars out of component libraries |
| 1. … | |
| b. Libraries Frame | Toggle whether this is seen |
| c. Menu Bar | Toggle whether this is seen |
| d. Component Attributes Frame | Toggle whether this is seen |
| 4. Project | |
| a. Rename Project | Rename current project |
| b. New Circuit | Create a new circuit in the current project |
| c. Rename Circuit | Rename current circuit |
| d. Save Circuit as Component | Save current circuit with component info |
| e. Load Library… | Load a library of circuits or components |
| f. Unload Libraries… | Unload libraries from current project |
| 5. Simulate | |
| a. Not yet implemented | Deactivated until available |
| 6. Window | |
| a. Tabify | Tabify all open circuits |
| b. Cascade | View all open circuits as cascading windows |
| c. Tile Horizontally | Tile all open circuits horizontally |
| d. Tile Vertically | Tile all open circuits vertically |
| e. Minimize all | Minimize all open circuits |
| f. [currently opened circuits in the project] | |
| 7. Help | |
| a. Antioch Help | View main help |
| b. Release Notes | View current and past release notes |
| c. About Antioch | View version/credit information |

**The toolbar frame:**

1. Can be right clicked to list available toolbars which can then be left clicked to be added or removed from the frame.
    a. Currently visible toolbars are ticked
    b. If there is no space available for the toolbar, it is not added



2. Individual toolbars have a handle which can be right clicked:
    a. To be reordered
    b. To be brought into an external window
    c. To be removed from the toolbar frame
    d. To toggle whether the toolbar's label is displayed or hidden



3. Individual button in toolbars act as they would if clicked as a menu item or from the library frame

**The libraries frame:** (Only superficial aspects have been implemented at this time)

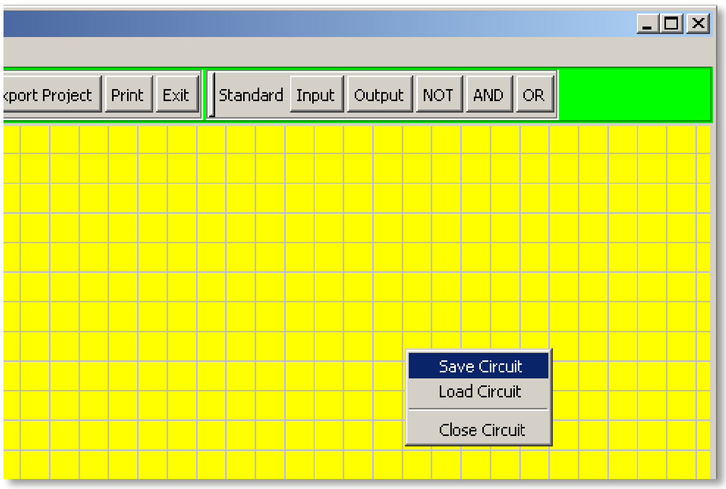
1. The current project is represented as a root folder
2. Circuits contained in this project are represented as children files of the root (at the top of the list)
3. Libraries loaded with this project are represented as children folders of the root (below all circuits)
4. Components in libraries can be selected
    a. They will become anchored to the mouse cursor when within the circuit design canvas
    b. Editable attributes will be shown in the component attributes frame.
5. Components and circuits in libraries can be double clicked which will open their circuit design representation in a new circuit design canvas

**The component attributes frame:** (Only superficial aspects have been implemented at this time)

1. A table of editable attributes of the selected component and their values.
2. Values can be modified on left click in a manner consistent with the domain (colour, orientation, size, etc)

**The circuit design canvas:** (Deviations from current version are noted)

1. Instantiated with a yellow background and faded grid.
2. Components defined in libraries (including the current project) can be drawn onto the canvas
    a. Components can have specific attributes altered through the component attributes frame. These attributes are defined in the component description file (may be a library with multiple component descriptions).
    b. (Note: Most Logism libraries have been implemented and many components from within those libraries can be drawn on the canvas but only from loading from a file)
3. Components can be connected by wires
    a. Component objects have nodes defined which wires can anchor to.
    b. If a node has no wire anchored to it then left-clicking will mark it as the end of a wire segment. Dragging will draw the wire with the other end of the wire segment following the cursor. If the user does not want to drag the wire, he can also simply left-click on another node or the same or a different component. The body of the wire will automatically find its way around obstacles (components which it cannot intersect) but this path can later be modified by the user.
    c. If a node already has a wire anchored to it and is left-clicked then the wire will change its anchor from the component to the cursor. The other end will stay anchored at its current location.
    d. If a component is moved by left-click dragging, wires anchored to its nodes will move along with it.
    e. (Note: Wires are not yet intelligent; Wires are unable to anchor to nodes.)
4. Components have restricted movement
    a. Components can be dragged anywhere on the canvas
    b. If a component is dropped on a non-occupied space, that will be its new position
    c. If a component is dropped on an occupied space, it will jump back to the position it was at before dragging occurred.
    d. (Note: Whether a space is occupied is determined by the position of the mouse cursor, it is still possible to overlap objects. This is not ideal.)
5. Non-circuit objects (text, rectangles, and ellipses) can also be drawn onto the canvas. They are ignored by components, wires and other non-circuit objects.
    a. These objects behave like components but can overlap each other, components, and wires without affecting behavior. They are the graphical equivalent of textual comments in code.
    b. (Note: These objects have yet to be implemented)
6. Can be right clicked to display a popup-menu
    a. Save the current circuit (restricted to Logisim files)
    b. Load a circuit (restricted to Logisim files)
    c. Close the current circuit



## 3. Design

- The application (at this stage) has been implemented using 8 main files.
    1. Antioch.py – the main application
    2. AntiochCanvas.py – Canvas drawing and events
    3. AntiochCircuits.py – Loading, saving, and parsing circuit files
    4. AntiochComponent.py – Defining general component objects
    5. AntiochMenus.py – Defining menu structure and methods
    6. AntiochSidebar.py – Creating Sidebar widgets for placement in the main application
    7. AntiochToolbars.py – Defines the toolbar widgets with necessary methods
    8. AntiochWire.py – Defining wire objects
- Libraries are implemented as plugins written in python. A library is a separate file with classes which inherit from some general component class and contain methods to define the shape of the component and placement of input/output nodes. (These can be extended for use with simulation).
  Below are the libraries which I have completed along with the type of component defined in each. (Note that the ability for libraries to be used as plugins has not been implemented, they must manually be imported within the source code)
    1. AntiochPlexerLibrary.py
        - Multiplexer
    2. AntiochMemoryLibrary.py
        - D Flip-Flop
    3. AntiochGatesLibrary.py
        - NOT
        - Buffer
        - AND
        - OR
        - NAND
        - NOR
        - XOR
        - XNOR
        - Constant
    4. AntiochArithmeticLibrary.py
        - Adder
    5. AntiochIOLibrary.py
        - Pin
        - Button
        - Hex Digit Display
        - LED
- Details – Filenames are **bold**, classes are <u>underlined</u>, methods are *italic*
    1. **Antioch.py**
        - <u>Antioch</u> – This is the main application class. All elements needed for running Antioch are instantiated by this class.
            - *__init__(self, master)* – initializes the master window given by the 'master' attribute and saves an instance of the <u>AntiochMenus</u>, <u>TBContainer</u>, <u>SideBar</u>, <u>Circuit</u>, and <u>WorkSpace</u> classes and packs them into the master window.
    2. **AntiochCanvas.py**
        - <u>WorkSpace</u> – Defines the WorkSpace widget. Inherits from the Tkinter Canvas class
            - *__init__(self, master, circuit = None)* – initializes the workspace by placing it within the 'master' frame, rendering a grid, and setting up event bindings. A default circuit may be sent into the constructor.

- *pressButton1(self, e)* – Event handler for <Button-1>. This allows for the selection of a component. This works with releaseButton1 to determine whether an object should be deselected or selected.
- *clickedComponent(self, x,y)* – Searches for the object which the cursor is inside. It will send out tracers starting at (x,y) to find the object. If more than one tracers finds the object then the cursor is inside its bounding box.
- *search(self, x, y, direction)* – Sends out soldiers (rectangles) to search for components and return the first one they find.
- *b1Motion(self, e)* – Event handler for <B1-Motion>. If an object is selected, it will be dragged.
- *releaseButton1(self, e)* – Event handler for <ButtonRelease-1>. This works with *pressButton1* to determine whether an object should be deselected or selected.
- *save_new_coords(self, obj)* – Saves new coordinates if the component given by 'obj' has been moved.
- *do_popup(self, event)* – Event handler for <Button-3>. Constructs and displays a popup menu. The code for this method has been modified from http://effbot.org/zone/tkinter-popup-menu.htm
- *saveCircuit(self)* – Saves the currently displayed circuit. The code for this method has been modified from http://tkinter.unpythonic.net/wiki/tkFileDialog
- *loadCircuit(self)* – Loads a new circuit from a file and draws it to the workspace after closing the currently displayed circuit. The code for this method has been modified from http://tkinter.unpythonic.net/wiki/tkFileDialog
- *closeCircuit(self)* – Clears the workspace.
- *resize(self, e)* – Event handler for <Configure>. When the workspace canvas is resized, the grid must be updated to fill the frame.
- *renderGrid(self, spacing)* – Renders the grid to the workspace with a given 'spacing'.
- *currenDimensions(self)* – Updates the class's private variables with the current dimensions of its frame.

3. **AntiochCircuits.py**
    - Circuit – Allows for loading and saving of circuit files in Logisim format. Uses the 'expat' XML parser which comes standard with Python.
        - *__init__(self)* – Initializes the Circuit instance with a list of attributes and a list of components which have been programmed so far.
        - *start_element(self, name, attrs)* – Event handler which is called when an XML element is started. Uses expat parsing information to store data about wires, components and their attributes.
        - *load(self, file)* – Loads a Logisim .circ file and parses it for circuit information. It saves a list of every component which is compatible with Antioch into a private variable.
        - *render(self, workspace)* – Renders stored components onto workspace.
        - *save(self, workspace, filename)* – Saves the Antioch 'workspace' to a Logisim .circ file as indicated by 'filename'
        - *preamble(self)* – A convenient place to write down the beginning of a .circ file. This assumes only the default Logisim libraries will be used and that they will have the correct enumeration between different version of logisim. This has been tested with Logisim version 2.3.0
        - *nearest10(self, n)* -- Round a number  'n' to nearest 10. Logisim uses a default grid spacing of 10 and components are snapped to the grid. This method ensures proper conversion.

4. **AntiochComponent.py**
   - InputPin -- Defines a pin for a component which only allows input. Inherits from the Wire class.
     - *__init__(self, canvas, start, end, width = 1, color = "black")* – A Wire instance is created and the TkInter canvas tag "input" is added to the instance.
   - OutputPin -- Defines a pin for a component which only allows output. Inherits from the Wire class.
     - *__init__(self, canvas, start, end, width = 1, color = "black")* – A Wire instance is created and the TkInter canvas tag "output" is added to the instance.
   - Component – General class for components
     - *__init__ (self, canvas, location, widthx, height, direction, width, size = None)* – canvas specifies where to draw the component
       location places the upper-left corner of the bounding box on the canvas
       widthx and height define the dimensions of the bounding box
       direction is used in a child class for orientation of internal objects
       width specifies how many bits can pass through on input or output (of course one can do math on this for various purposes)
       size exists for compatibility with Logisim
     - *location(self, location = None)* – A getter and setter for the location of a component. If the location value is provided as an argument, the component's location is set to the new value and the value is returned. Otherwise, the current value will be returned.
     - *widthx(self, widthx = None)* – A getter and setter for the widthx of a component. If the widthx value is provided as an argument, the component's widthx is set to the new value and the value is returned. Otherwise, the current value will be returned.
     - *height(self, height = None)* – A getter and setter for the height of a component. If the height value is provided as an argument, the component's height is set to the new value and the value is returned. Otherwise, the current value will be returned.
     - *direction(self, direction = None)* – A getter and setter for the direction of a component. If a valid direction value is provided as an argument, the component's direction is set to the new value and the value is returned. Otherwise, the current value will be returned.
     - *width(self, width = None)* – A getter and setter for the width of a component If a width value is provided as an argument, the component's width is set to the new value and the new value is returned. Otherwise, the current value will be returned.
   - MultiInSingleOut – A standard class for components with multiple inputs on one side and a single output on the other. It is assumed that any such object will have at least 2 inputs and has initial dimensions based on the number of inputs. Inherits from the Component class.
     - *__init__(self, canvas, location, direction = E, width = 1, inputs = 2, size = 1)* – Initializes a Component instance with a certain number of inputs and dimensions which are dependent on the number of inputs.
     - *initDimensions(self, inputs)* – Initialize the dimensions of the component based on the number of inputs.
     - *inputPins(self)* – Creates input pins which are placed equidistant from each other and centered on the component edge. They are then saved to private lists to be worked with in a simulation environment.

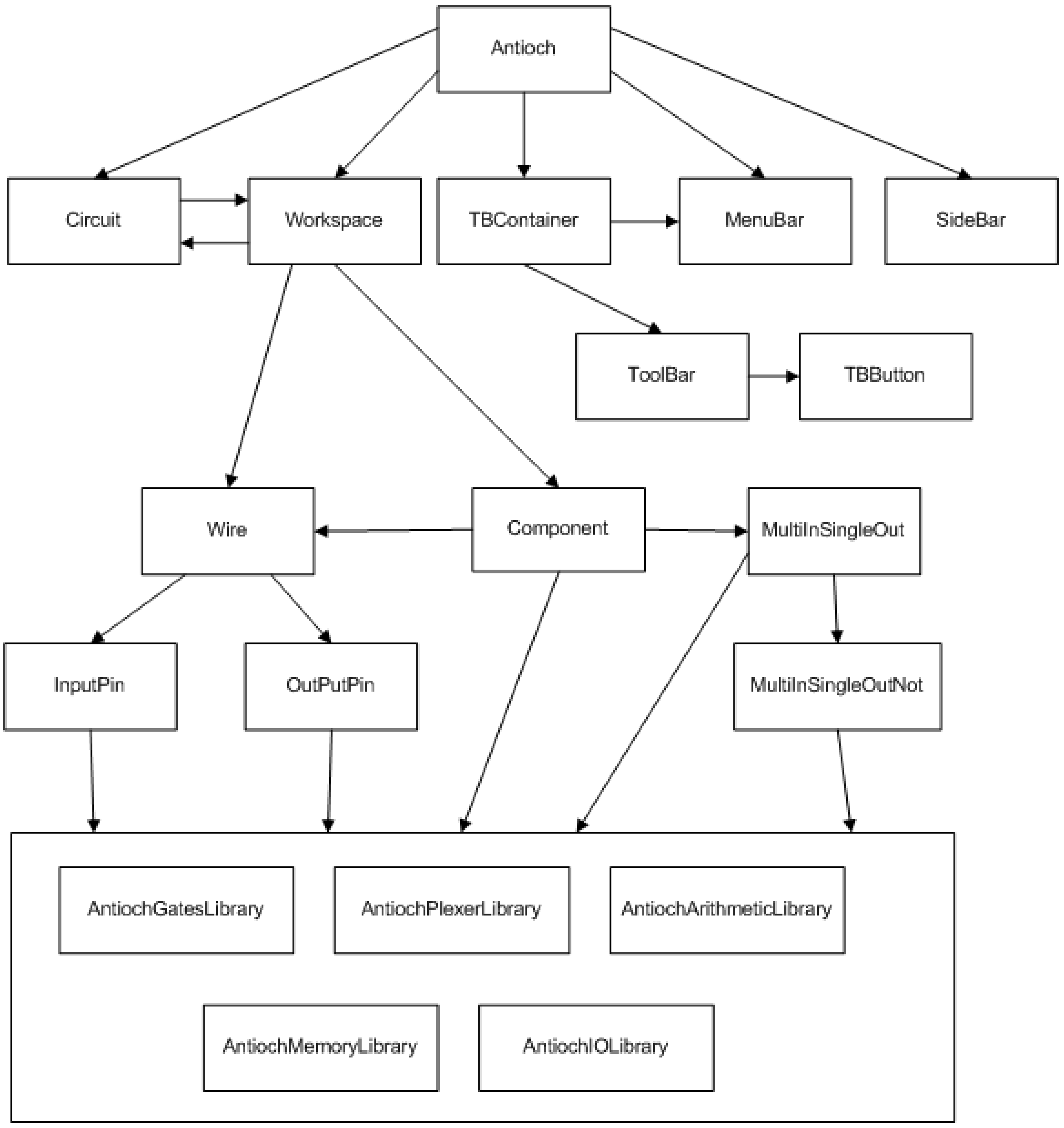
        - *outputPin(self)* – Creates an output pin which is centered on the component edge. It is then saved into a private variable for use in a simulation environment.
    - MultiInSingleOutNot – Extends the MultiInSingleOut class to add a bubble to the output.
        - *bubble(self)* – Draws a bubble before the output pin

5. **AntiochMenus.py**
    - MenuBar – Defines methods and data for the complete static menu for Antioch.
        - *__init__(self, master)* – Initializes a TkInter menu widget on 'master'. The data structure for the menu is hard coded here as a tuple since it will not be changed by the user or the program.
        - *_parse(self, menuItems)* – A private method for parsing the menu datastructure into a Tkinter menu widget.
        - *getList(self)* – Returns a list of all top-level menu items
        - *quit(self)* – Quits the main program (called by a menu item)
        - (Every menu item has an associated callback and they are defined in the file but do nothing except for *quit* as noted above)
6. **AntiochSidebar.py**
    - SideBar – The sidebar contains the Library and Component Attribute frames. Inherits from the TkInter Frame class.
        - *__init__(self, master)* – Initializes a TkInter frame within 'master'. This frame is given a handle which is bound to dragging.
        - *_resize(self, e)* – A private method to resize the sidebar frame and its children. This is an event handler for <B1-Motion>.
    - LibraryFrame – A place to put loaded library information. Inherits from the TkInter LabelFrame class.
        - *__init__(self, master, **options)* – Defines the look of the frame and draws it to 'master'
        - *resize(self, newWidth)* – Resizes the width of the frame to 'newWidth'.
    - CompAttrFrame – A place to put loaded component attribute information. Inherits from the TkInter LabelFrame class.
        - *__init__(self, master, **options)* – Defines the look of the frame and draws it to 'master'
        - *resize(self, newWidth)* – Resizes the width of the frame to 'newWidth'.
7. **AntiochToolbars.py**
    - Toolbar – Defines a toolbar widget. Inherits from TkInter Frame class.
        - *__init__(self, master)* – Initializes the toolbar within the 'master' frame and adds a handle with certain bindings. They are drawn with the grid geometry manager.
        - *_populate(self)* – Private method to populate the toolbar with buttons
        - *setLabel(self, label)* – Sets the label of the toolbar to 'label'
        - *setLabel(self)* – Gets the label of the toolbar
        - *toggleLabel(self)* – Toggles whether the toolbar's label is shown
        - *setButtons(self)* – Save a list of buttons for use in the toolbar
        - *do_popup(self, event)* – Event handler for <Button-3>. Constructs and displays a popup menu. The code for this method has been modified from http://effbot.org/zone/tkinter-popup-menu.htm
        - *_moveRight(self)* – Called when you want to move the toolbar to the right.
        - *_moveLeft(self)* – Called when you want to move the toolbar to the left.

        - *_tearOff(self)* – Remove the toolbar from the main application window and put it in its own toplevel window.
        - *_remove(self)* – Removes the toolbar from its container
        - *Render(self, col = 0)* – Render the toolbar at the specified column.
        - *_checkSpace(self)* – Returns true if there is space to draw the toolbar.
    - TBButton – Defines a toolbar button widget. Inherits from the TkInter Button class.
        - *__init__(self, master, **options)* – Places the button into the toolbar indicated by 'master' and gives it a standard appearance.
    - TBContainer – Sets up a container for toolbar widgets
        - *__init__(self, master, topMenu)* – Creates a list in which to save all created toolbars, instantiates the top-level menu (the toolbars and menus are very closely related), and initializes a standard toolbar.
        - *setTBList(self, tblist)* – Clear the toolbar list and then populate it appropriately with the list given by 'tblist'.
        - *removeTB(self, tb)* – Remove the toolbar given by 'tb' from the geometry manager and from the toolbar list.
        - *addTB(self, tb)* – Add the toolbar given by 'tb' to the toolbar list.
        - *moveTB(self, tb, direction)* – Moves a toolbar given by 'tb' to the left or the right as indicated by 'direction'
        - *getTBList(self)* – Gets the toolbar list.
        - *_initPopup(self)* – Initializes data for the popup menu
        - *do_popup(self, event)* – Event handler for <Button-3>. Constructs and displays a popup menu. The code for this method has been modified from http://effbot.org/zone/tkinter-popup-menu.htm
        - *_toggleTB(self, e)* – Used by the popup menu to remove or add a toolbar to the container.
        - *render(self)* – Renders every toolbar in the list.

8. **AntiochWire.py**
    - Wire – Defines a wire object
        - *__init__(self, canvas, start, end, width = 1, color = "black")* – The arguments 'start' and 'end' are complex number indicated the position and length of the line, 'width' is an integer describing how many bits can travel the wire, 'color' is a TkInter defined color or valid hex string.
        - *start(self, start = None)* – A getter and setter for the starting point of a wire object. If a start value is provided as an argument, the wire's starting point is set to the new value and the new value is returned. Otherwise, the current value will be returned.
        - *end(self, end = None)* – A getter and setter for the ending point of a wire object. If an end value is provided as an argument, the wire's ending point is set to the new value and the new value is returned. Otherwise, the current value will be returned.
        - *width(self,  width = None)* – A getter and setter for the width of a wire object. If a width value is provided as an argument, the wire's width is set to the new value and the new value is returned. Otherwise, the current value will be returned.
        - *color(self,  color = None)* – A getter and setter for the color of a wire object. If a color value is provided as an argument, the wire's color is set to the new value and the new value is returned. Otherwise, the current value will be returned.

- Class Interaction



1. Antioch is the main application class
2. Circuit and Workspace work together to load and save circuit files
3. TBContainer gets information from MenuBar to populate the right-click menu
4. Sidebar isn't used at this time
5. TBContainer holds instances of Toolbar which holds instances of TBButton
6. The Wire and Component classes must be able to draw to the Workspace
7. The Component class has instances of the Wire class
8. InputPin and OutPutPin are extensions of the Wire class
9. MultiInSingleOut is an extension of the Component class
10. MultiInSingleOutNot is an extension of the MultiInSingleOut class
11. All libraries contain classes which reference one or more of InputPin, OutputPin, Component, MultiInSingleOut and MultiInSingleOutNot.

## 4. Notes

Antioch requires several modules which come with Python and TkInter:

- tkFileDialog
- xml.parsers.expat
- math

Some code was modified from external sources. They have been attributed in comments and in the above design details. They are:

- http://effbot.org/zone/tkinter-popup-menu.htm
- http://tkinter.unpythonic.net/wiki/tkFileDialog
- http://effbot.org/zone/tkinter-popup-menu.htm
- Logisim .circ files

Antioch is supposed to be a Python port of Logisim. Logisim is written in Java and is open source. There may be similarities between Antioch code and Logisim code but the only time I looked at the Logisim source was to see if it also hard-coded its libraries. I had already started writing the libraries in Python instead of a markup language to be later parsed and I wanted to see if that was an acceptable way to do it. Logisim did it this way, so I didn't change my way.

Approx. 2500 lines (including comments, not including noops)