Getting Started With Java Bread Board In Windows

Table of Contents

Introduction... 3
Bread boards 5
Components 5
Wires...10
Simulation...11
Saving and Loading Circuits..13
Custom Integrated Circuits 14
User defined Integrated Circuits 23

Introduction

The aim of this document is to introduce the main functionality of the Java bread board (JBB) simulator. The document's layout has been constructed to support JBB laboratory 1, explaining the required software tools and steps involved to accomplish each task. The Java bread board software is a simulation tool that replicates the hardware development environment you will be using in the laboratory sessions, allowing you to test out ideas and develop designs in the software labs or at home. The software can be downloaded from:

http://www.cs.york.ac.uk/~mjf/CSA/APPS/JavaBreadBoard.zip

This zip file contains all the files you require and some additional documentation on this simulation tool. Using your preferred web browser download the file JavaBreadBoard.zip to c:\temp. Double click on this file to unzip it. Note, you can not run this software from your home directory owing to the expanded directory name used by this remote directory.

To execute this Java program you must have a Java runtime environment and virtual machine installed on your computer. Note, for software and hardware lab machines within the department this has already been pre-installed. For home machines this software may not and can be downloaded from:

http://java.com/en/download/manual.jsp

At the time of writing this document the current Java runtime environment for the Windows operating system is: Jre-6u17-windows-i586-iftw-rv.exe. If required download and install this software onto your home PC. Note, you will require administrator rights to do this.

The simulation software can be started at the command prompt or by double clicking on the supplied batch file. To launch this program at the command line you first need to open a command prompt:

Start -> Programs -> Accessories -> Command Prompt

At the command prompt type:

```
c:
cd c:\temp\JavaBreadBoard
```

This will move you to the directory containing the simulator's class file directory hierarchy. To launch the simulator at the command prompt type:

```
go.bat
```

Alternatively you can double left click on this batch file from the file browser. This will launch the Java bread board main interface as shown in figure 1.

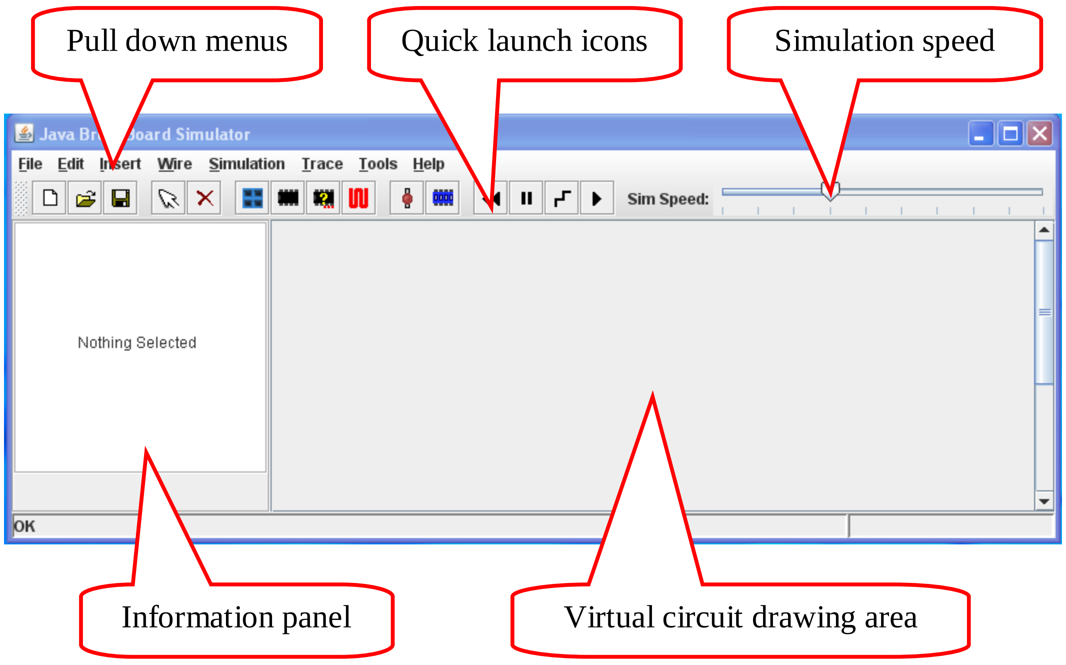
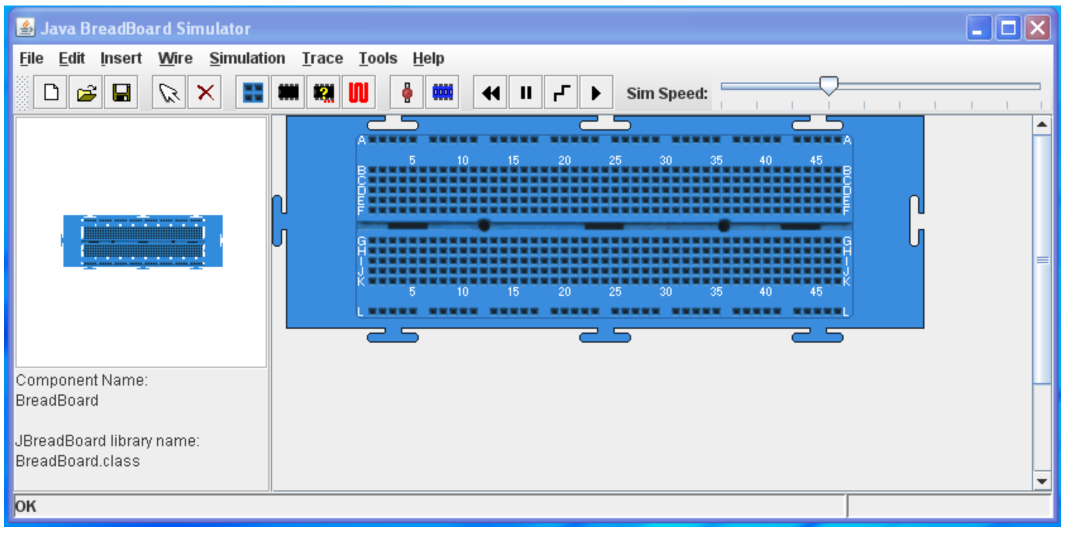

Pull down menus
Quick launch icons
Simulation speed
Java BreadBoard Simulator
File Edit Insert Wire Simulation Trace Tools Help
Sim Speed:
Nothing Selected
OK
Information panel
Virtual circuit drawing area

Figure 1 : Java bread board main interface


Java BreadBoard Simulator
File Edit Insert Wire Simulation Trace Tools Help
Sim Speed:
OK

Component Name:
BreadBoard

JBreadBoard library name:
BreadBoard.class

Figure 2 : Virtual bread board

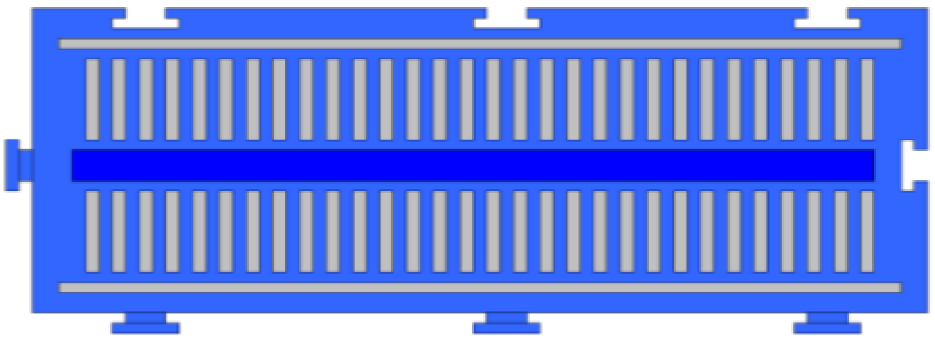


Figure 3 : Virtual bread board internal construction

Bread boards

To assemble your virtual circuit you need at least one bread board. These can be added to the simulator by either clicking the bread board icon or click on the pull down:

```
Insert -> BreadBoard
```

This will add a bread board to the simulator as shown in figure 2 onto which integrated circuits, wires and other electronic components can be added. The bread boards internal construction is illustrated in figure 3. Horizontal rows A and L run the continuous length of the board, whilst rows B – K are divided into 47 vertical columns spanning each half of the board. Following tradition the top horizontal row of holes (row A) is connected to VCC and the bottom horizontal row of holes (row L) is connected to GND.

Note, more bread boards can be added to the simulator as the complexity of your virtual circuit increases. To delete a bread board left click on an empty area of the desired board, this will update the information panel. The selected board can be deleted by pressing the DEL button, clicking on the Delete object icon or click on the pull down:

```
Edit -> Delete
```

Note, if the bread board contains components a warning popup will appear asking to confirm this action. If you press OK all circuit elements on this board will be deleted.

Components

To add an integrated circuit to your virtual circuit either click on the 'Select and Add Chip' icon or click on the pull down:

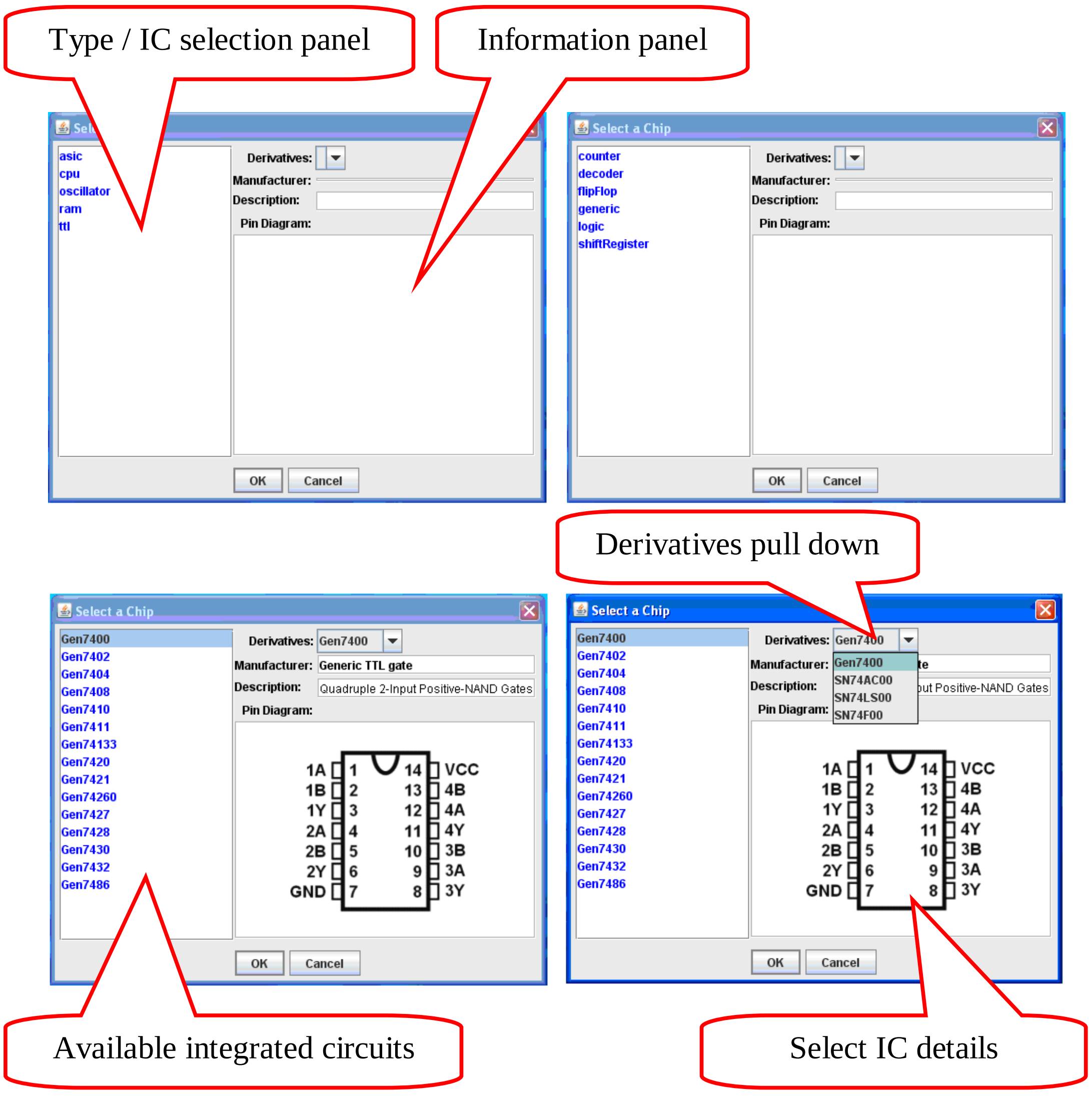
```
Insert -> Chip
```

This will open the 'Select a Chip' window as shown in figure 3, allowing you to select from a range of pre-defined, custom or user defined integrated circuits. To help organise these ICs a common directory structure is used, classifying each IC by its type and then function. To navigate down through this directory hierarchy double left click on the desired IC type within the selection panel. To move back up this directory hierarchy double right click on an empty region within the selection panel. The example shown in figure 3 are the directory levels involved in selecting a simple Boolean logic function IC.

```
ttl -> logic
```

Note, if the required IC can not be found please refer to the sections on custom or user defined integrated circuits. To one of the listed integrated circuits single left click on the required IC. In figure 3 a generic 7400 dual input NAND gate has been selected. Single clicking on one of the listed ICs will update the right hand side information panel of this window, giving you a brief description and the integrated circuit and its pinout diagram.

To minimise the number of different ICs presented to the user at one time only the top level generic IC type is displayed in the selection panel. If available you may select a specific derivative of this generic integrated circuit i.e. a particular manufacturer or silicon implementation, using the ‘Derivatives’ pull down box as shown in the bottom right frame of figure 3. Note, different derivatives will implement the same logical function, but could have different timing characteristics i.e. switching speeds as defined in their datasheets.

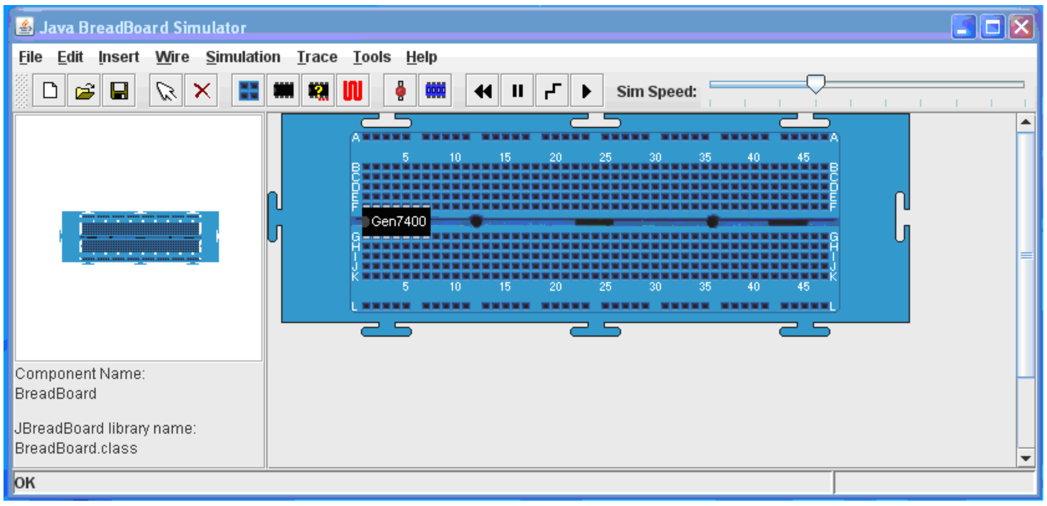


Derivatives: Gen7400
Manufacturer: Generic TTL gate
Description: Quadruple 2-Input Positive-NAND Gates

Figure 3 : Select a Chip (top left) top level chip hierarchy, (top right) TTL subdirectory, (bottom left) logic subdirectory, (bottom right) Gen7400 derivatives list

To confirm your selection left click on the OK button. This will add the selected IC to your bread board as shown in figure 4.

If required the IC can now be moved by performing a left click and hold on the ‘chip’ graphic. This will allow you to drag the IC to the required position, as shown in figure 5. Note, the IC can only be moved to positions where there are enough unused holes which do not invalidate the systems design rules e.g. connecting an output to an


Java BreadBoard Simulator
File Edit Insert Wire Simulation Trace Tools Help
Sim Speed:
Gen7400

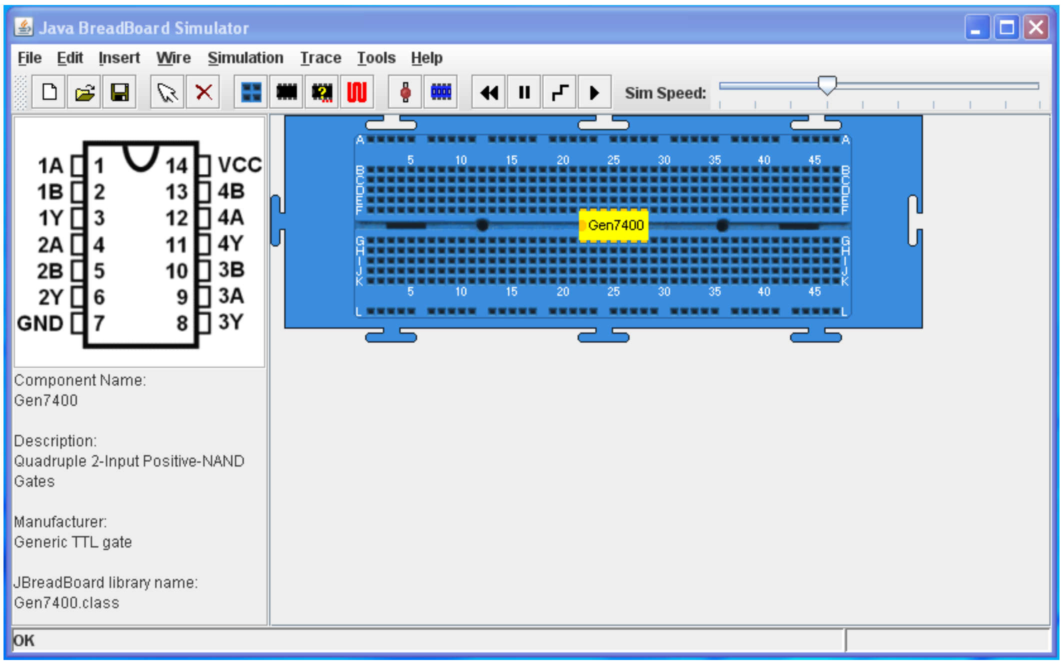
Component Name:
BreadBoard

JBreadBoard library name:
BreadBoard.class

OK

Figure 4 : added 7400 integrated IC

output or a power supply rail etc. This limits the IC's position to the central row of the bread board e.g. rows F and G. Left clicking on an IC will update the IC information panel on the left hand side of the main window.


Java BreadBoard Simulator
File Edit Insert Wire Simulation Trace Tools Help
Sim Speed:
1A 1B 1Y 2A 2B 2Y GND
VCC 4B 4A 4Y 3B 3A 3Y
Gen7400

Component Name:
Gen7400

Description:
Quadruple 2-Input Positive-NAND Gates

Manufacturer:
Generic TTL gate

JBreadBoard library name:
Gen7400.class

OK

Figure 5 : repositioned 7400 IC and information panel

To allow user defined test signals to be applied to a virtual circuit dual in-line package (DIP) switches can be used. To add a DIP switch either click on the 'Add DIP Switches' icon 🟦 or click on the pull down:

```
Insert -> Dip Switches -> Single
                       -> Double
                       -> Triple
                       -> Quad
```

If required the DIP switch can be moved by performing a left click, hold and drag as previously described. Note, the default DIP switch package size used by the quick launch icon is triple, this being updated to the most recent pull down menu selection. Owing to the passive nature and size, DIP switches can be placed on a larger range of rows than ICs, as shown in figure 6. However, design rule checks still apply e.g. the three DIP switch banks starting at column 14 in figure 6 could be configured to short out the power supply. To prevent this the simulator will prevent all three switches in that column being switched on. The default switch position of each switch in the DIP is off i.e. open circuit. To change the state of a switch to on i.e. closed circuit, left click on the dark blue square above the selected switch element. This will move the white switch bar up, turning that switch on e.g. the DIP switch bank starting at column 30 in figure 6 has been set to OFF – ON – OFF. To change the state of a switch to off, again click on the lower blue square below the selected switch element. This will move the white switch bar down, turning that switch off.

To allow a user to view the state of an input or an output light emitting diodes (LED) can be used. To add a LED either click on the 'Add LED' icon or click on the pull down:

```
Insert -> LED -> Red
              -> Yellow
              -> Green
```

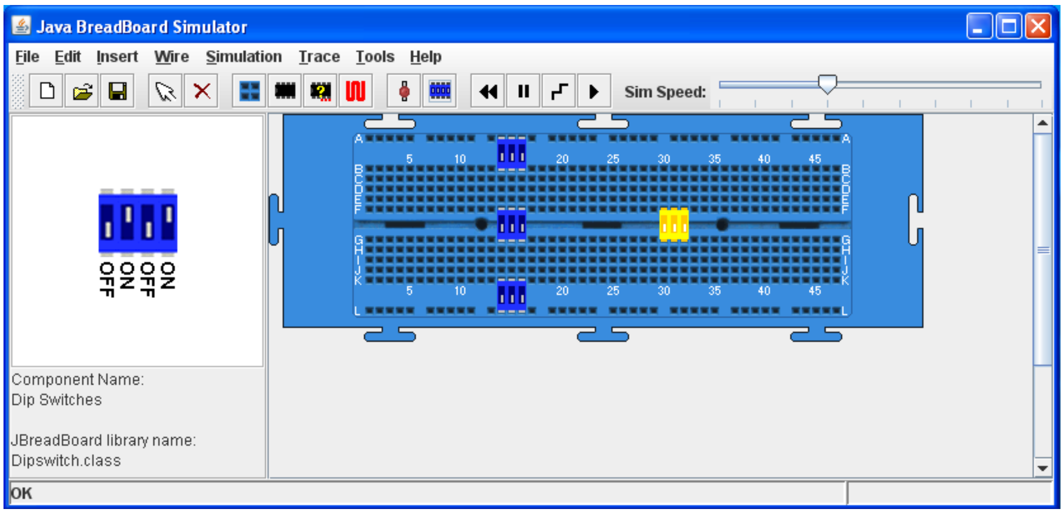
If required the LED can be moved by performing a left click, hold and drag as previously described. Note, the default LED colour used by the quick launch icon is red, this being updated to the most recent pull down menu selection. Owing to the passive nature and size, LEDs can be placed on a larger range of rows than ICs, as shown in figure 7. However, design rule checks still apply e.g. an LED can not be placed between rows G and K as the anode and cathode would be shorted out. A LED is always orientated with the anode top and cathode bottom. To illuminate a LED, a positive voltage (VCC or DIP switch / IC output) must be connected to the anode whilst the cathode is connected to GND. Note, the Java bread board simulator only simulates discrete signal states i.e. logic 1, logic 0 and high impedance. Therefore, the red, yellow and green LED network shown in figure 7 which would be illuminated in a real circuit will not be illuminated during a simulation as the simulator can not simulate the potential divider formed by these three LEDs.

WARNING : when constructing a real implementation of your virtual circuit in the hardware remember to add a current limiting resister in series with each LED. Failure to do this will damage the LED and the IC. The value of the resister used is dependent on the LED, typically start with 1KΩ, if the LED is not very bright the resister can be reduced in value to 470Ω.

To delete a component left click on the component's graphic and press the DEL button, click on the Delete object icon or click on the pull down:

```
Edit -> Delete
```

If you can not select the desired component you may need to switch to 'Select Mode',

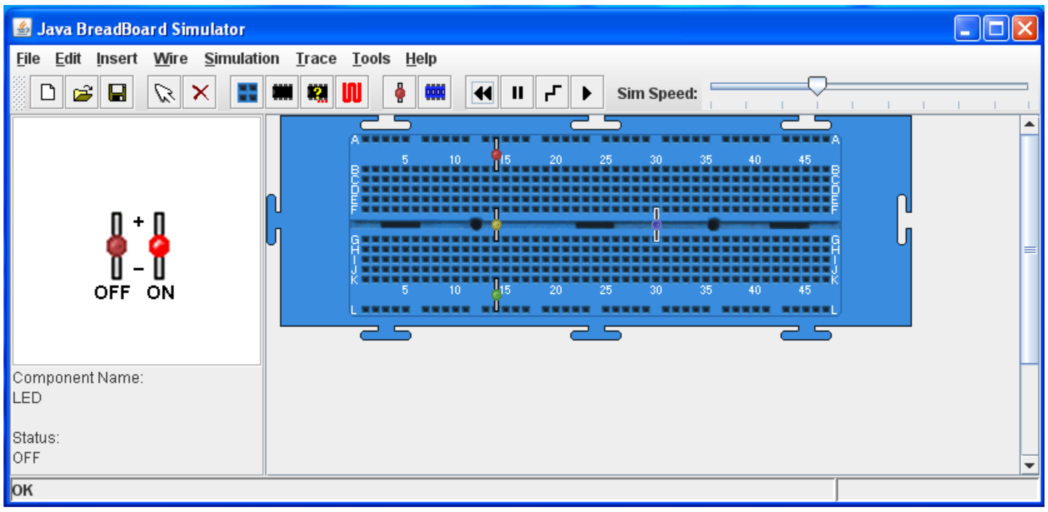

Java BreadBoard Simulator
File Edit Insert Wire Simulation Trace Tools Help
Sim Speed:
OFF ON OFF ON

Component Name:
Dip Switches

JBreadBoard library name:
Dipswitch.class

OK

Figure 6 : four triple DIP switch banks

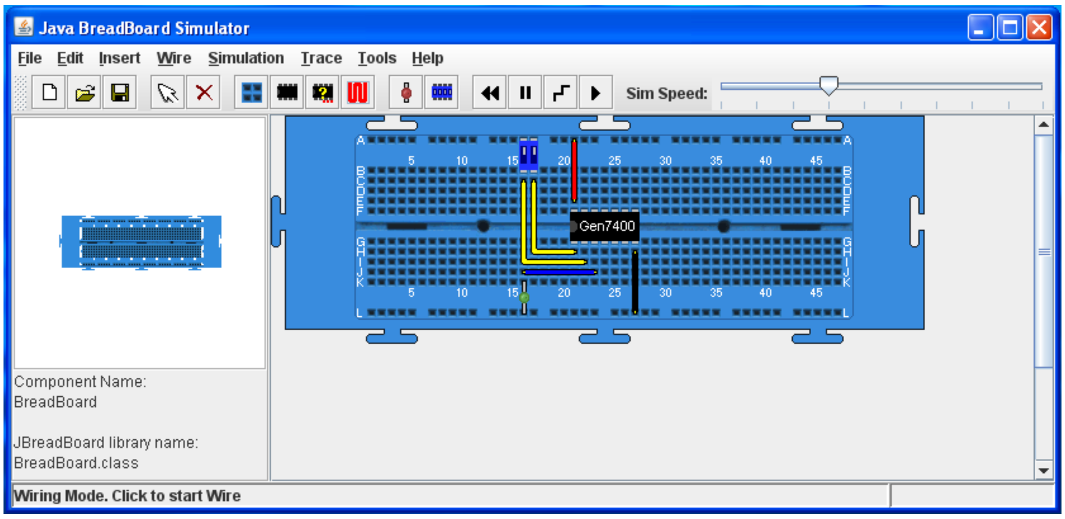

Java BreadBoard Simulator
File Edit Insert Wire Simulation Trace Tools Help
Sim Speed:
OFF ON

Component Name:
LED

Status:
OFF

OK

Figure 7 : four light emitting diodes


Java BreadBoard Simulator
File Edit Insert Wire Simulation Trace Tools Help
Sim Speed:
Gen7400

Component Name:
BreadBoard

JBreadBoard library name:
BreadBoard.class

Wiring Mode. Click to start Wire

Figure 8 : wiring a virtual circuit

this can be achieved by click on the ‘Selector’ icon or click on the pull down:

```
Edit -> Selection Mode
```

Wires

To add wire interconnects to your virtual circuit either click on the ‘Wiring Mode’ icon or click on the pull down:

```
Wire -> Add Wire
```

Note, to allow an IC’s functionality to be simulated its power supply lines must be connected using wires to VCC (top horizontal row A) and GND (bottom horizontal row L). To connect a wire between two points single left click on the starting bread board hole and double left click on the destination hole. To route a wire around components i.e. turn through a 90 degree bend, single left click on the bread at the point you wish to create the bend. If an anytime you wish to un-route a wire segment you have laid down press the ESC key or click on the pull down:

```
Wire -> Cancel Wire Segment
```

This will sequentially remove each segment back to and including the initial starting hole. Note, the default wire colour used by the quick launch icon is red, this being updated to the most recent pull down menu selection.

```
Wire -> White
     -> Black
     -> Red
     -> Orange
     -> Yellow
     -> Green
     -> Blue
```

Typically different colours are used to indicate the different roles within a circuit, as shown in figure 8. This circuit has been constructed to allow the truth table for a 7400 two input NAND gate to be tested. Traditionally red is used to signify VCC and black GND. Input and output colours are user defined, in this example inputs have been colour coded as yellow and outputs in blue.

To delete a wire once laid, left click on the wire and press the DEL key, click on the Delete object icon or click on the pull down:

```
Edit -> Delete
```

To allow you to select a wire you may need to switch from ‘Wiring Mode’ to ‘Select Mode’, this can be achieved by click on the ‘Selector’ icon or click on the pull down:

```
Edit -> Selection Mode
```

Simulation

When a virtual circuit has been designed its functionality can be confirmed through simulation. Test inputs to the circuit are applied using DIP switches. To view the state of specific test points the user can either add LEDs or test probes. Multiple test probes can be added to a design allowing the user to capture the state of the circuit at a specific time. To add a probe click on the pull down:

```
Trace -> Insert Probe
```

This will add a test probe to your circuit P, the default position being row B, column 1. To move a test probe left click, hold and drag the probe graphic to the bread board hole you wish to monitor. If you can not select the desired probe you may need to switch to 'Select Mode', this can be achieved by click on the 'Selector' icon or click on the pull down:

```
Edit -> Selection Mode
```

Due to their size a test probe can be place in any bread board hole, as shown in figure 9 i.e. probe 1 row A VCC, probe 2 row F unconnected and probe 3 row L GND. Note, if you are unsure of a probe's ID single left click on the probe, this will update the information panel.

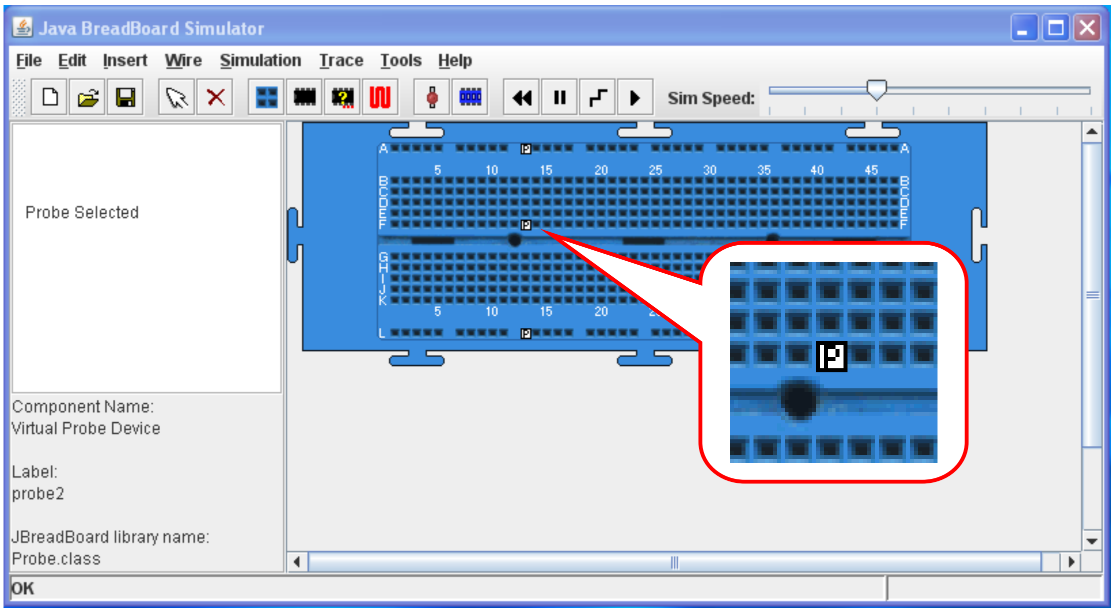

Java BreadBoard Simulator
File Edit Insert Wire Simulation Trace Tools Help
Sim Speed:
Probe Selected
Component Name:
Virtual Probe Device
Label:
probe2
JBreadBoard library name:
Probe.class
OK

Figure 9 : test probe icon

To delete a probe once placed, left click on the probe and press the DEL key, click on the Delete object icon or click on the pull down:

```
Edit -> Delete
```

If you can not select the desired probe you may need to switch to 'Select Mode' as previously described.

To simulate a circuit either click on the 'Run Simulation' icon ▶ or click on the pull down:

```
Simulation -> Run
```

The speed of the simulation will be determined by the processing performance of the computer the Java bread board software is being executed upon on and the complexity of the circuit. To enable the user to view individual gate transitions it is sometimes useful to intentionally slow down the simulation speed. This can be achieved by moving the simulation speed slider in the top right of the main interface window.

At anytime during a simulation DIP switch positions may be changed, however, no additional components or wires may be added to or removed from the circuit.

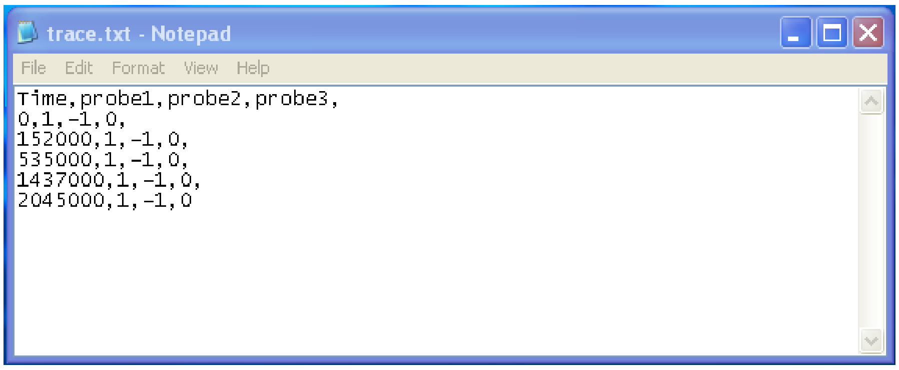
To pause a simulation either click on the 'Pause Simulation' icon ⏸ or click on the pull down:

```
Simulation -> Pause
```

A simulation is also paused when the circuit's probe state is saved. To save the circuits state click on the pull down:

```
Trace -> Save Probe
```

This will open a new window 'Save' allowing the user to specify the text file this data should be written to. To restart the simulation click on the 'Run Simulation' icon as previously described. The state of the circuit can be saved multiple times during a simulation. If this data is written to the same file it will be automatically concatenated onto its end, as shown in figure 10.



```
Time,probe1,probe2,probe3,
0,1,-1,0,
152000,1,-1,0,
535000,1,-1,0,
1437000,1,-1,0,
2045000,1,-1,0
```

Figure 10 : trace text file

This is the simulation trace for the test probes shown in figure 9, for which the circuit's state was recorded four times. Note, the state of the circuit at time zero is always automatically added. Each save operation adds a new line to the specified file, containing the current time and probe values separated by commas. Probe values may be true '1', false '0' or unknown '-1'.

The simulation can be reset i.e. set to simulation time zero, by either pressing the BACK SPACE key, clicking on the 'Reset Simulation' icon or click on the pull down:

```
Simulation -> Reset Simulation
```

An alternative approach to enable the user to view individual gate transitions is to single step through the circuits simulation events. A simulation is event driven. When a components input is updated a timed event is created for its associated outputs based on the logic relationship and its transport delay. These events are stored in a queue and can be stepped through sequentially rather than continuously as previously described.

To step to the next timed event either press the ENTER key, click on the 'Step Simulation' icon or click on the pull down

```
Simulation -> Step Simulation
```

Saving and Loading Circuits

To save a virtual circuit so that the user can continue working on a design at a later time click on the 'Save' icon or click on the pull down:

```
File -> Save
```

This will open a 'Save' window allowing the user to specify an output file (.cir extension). Enter a file name and left click on the Save button.

To load a virtual circuit to continue to develop an existing design click on the 'Open' icon or click on the pull down:

```
File -> Open
```

This will open the 'Open' window allowing the user to specify an input file (.cir extension). Browse to and highlight the desired file and left click on the Open button. Note, if there is a bread board already open an warning window will open information you that all circuits currently present will be deleted.

In situation where a number of identical or very similar bread boards need to be created a previously saved circuit can be inserted into a design i.e. additional bread boards containing the desired circuit can be repeatably added to a design. To insert an existing virtual circuit click on the pull down:

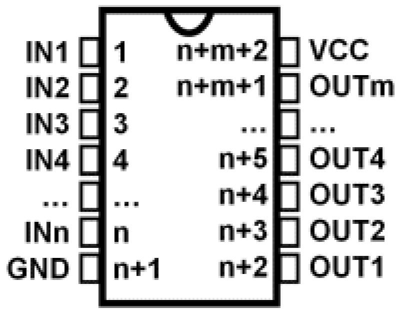
```
File -> Insert Circuit
```

To start a new design i.e. delete all previous circuits currently open click on the 'New' icon or click on the pull down:

```
File -> New
```

Custom Integrated Circuits

If the required integrated circuit is not included in the pre-installed catalogue, or if the desired functionality is not available in a commercial IC the Java bread board software allows you to define a custom IC. There are two types of custom integrated circuit supported, 'logic' and 'state machine'. Purely combinational logic based circuits e.g. SOP networks, should use the 'logic' custom IC and those with an internal state e.g. binary counter, the state machine custom IC. Each custom IC has a fixed package layout with *n* inputs and *m* outputs as shown in figure 11. Note, inputs are always on the left hand side of the IC and outputs on the right hand side.



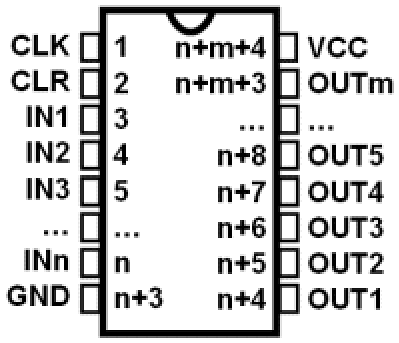
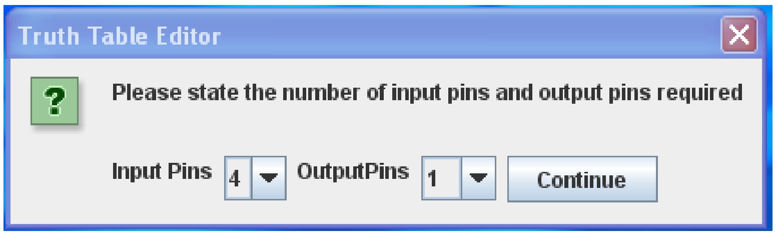


Figure 11 : logic IC (left), state machine IC (right)

To define the truth table for a logic custom IC the user can either use the truth table editor, or the schematic capture editor. To open the truth table editor click on the pull down:

```
Tools -> Truth Table Editor
```

This will open the initial Truth Table Editor window allowing the user to specify the required number of inputs and outputs using the associated pull down boxes, as shown in figure 12. The maximum number of inputs and outputs is currently limited to 8 and 16 respectively, in this example 4 inputs and 1 output have been selected. Note, any combination of inputs and outputs are allowed. Where required 'not connected' (NC) pins will be automatically inserted to pad out unused pin positions within the package foot print.

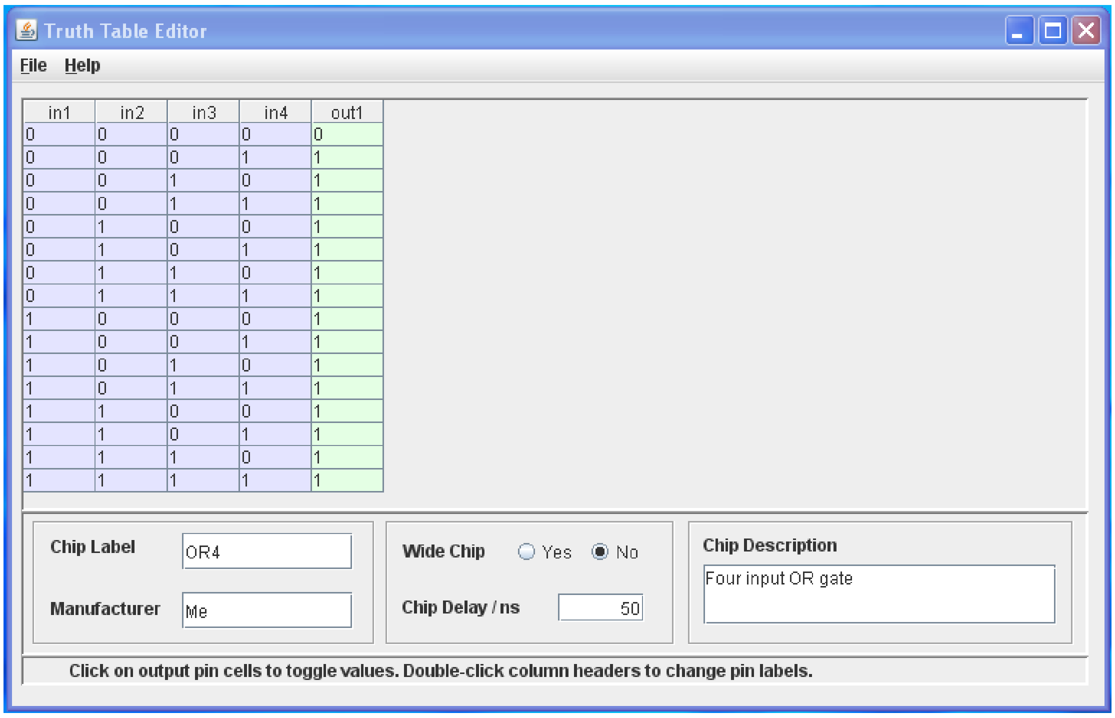

Truth Table Editor

Please state the number of input pins and output pins required

Input Pins 4 OutputPins 1

Figure 12: initial input and output pin selection

Left click the Continue button to proceed. This will launch the main Truth Table Editor window, allowing you to define the IC's input to output relationship, as shown in figure 13. To change an output of a specific input state to the required value single left click on that output's column bit i.e. the light green column. This will toggle the bit value from 0→1, or 1→0. The user may also edit the input and output names by double left clicking on the default names e.g. In1, In2, Out1 etc, at the top of each column.



in1	in2	in3	in4	out1
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Chip Label: OR4
Manufacturer: Me
Wide Chip: Yes [✓ No]
Chip Delay / ns: 50
Chip Description: Four input OR gate

Click on output pin cells to toggle values. Double-click column headers to change pin labels.

Figure 13: truth table editor of a four input OR gate

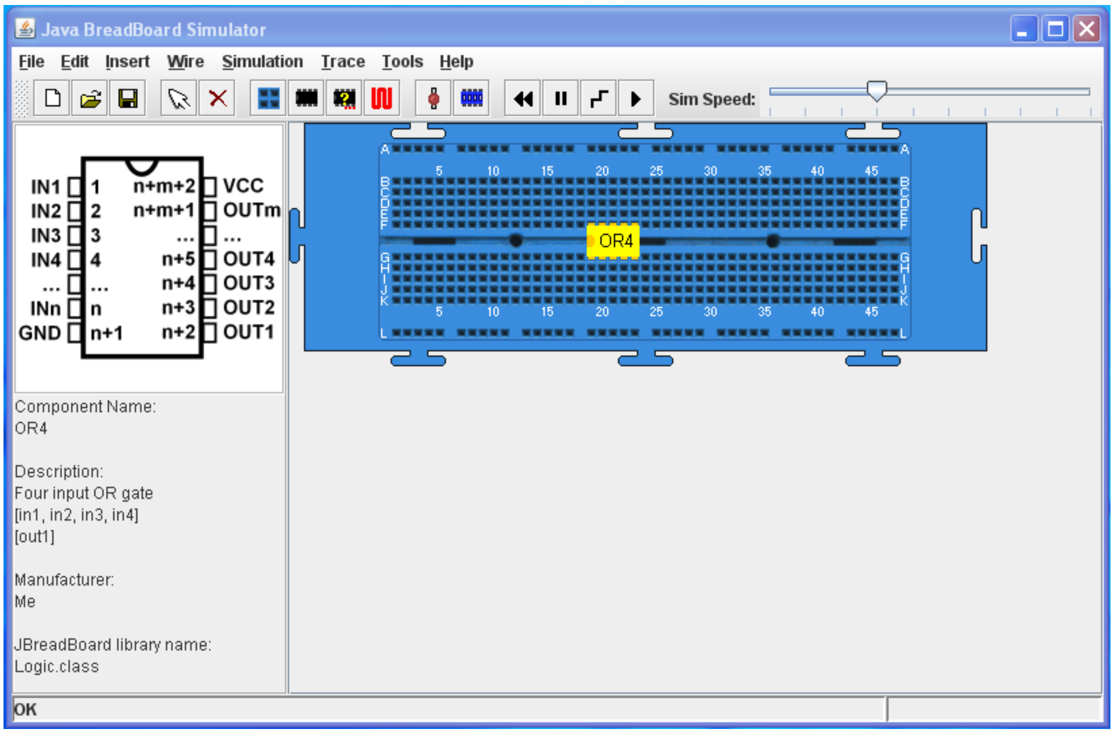
The name and a description of this IC can be entered in the bottom panel of this window. A common propagation delay or Chip Delay i.e. the delay from an input changing to an output being updated, is used for all outputs and can be specified in nano seconds. One final option available to the user is to specify if this IC should use a .600 (Wide) or .300 (Default) package size. This is option is purely to allow the user to match the ICs profile to aid in routing design of real implementations.

Once the truth table and other details have been entered this data can be stored to a file. This can be accessed by clicking on the pull down:

```
File -> Create Chip File
```

This will open a ‘Save’ window allowing the user to specify an output file (.chp extention). Enter a file name and left click on the Save button. This will open the ‘Chip File Created’ window, you may either continue i.e. enter another truth table or exit back to the main interface. If you select continue you may re-edit the current truth table and save it under a different name. To change the number of inputs and output click on the pull down:

```
File -> New
```



Component Name:
OR4

Description:
Four input OR gate
[in1, in2, in3, in4]
[out1]

Manufacturer:
Me

JBreadBoard library name:
Logic.class

Figure 14: inserting a custom logic IC

To instantiate this new custom IC on a bread board click on the ‘Select and Add Chip’ icon or click on the pull down:

```
Insert -> Chip
```

Navigate down the directory hierarchy

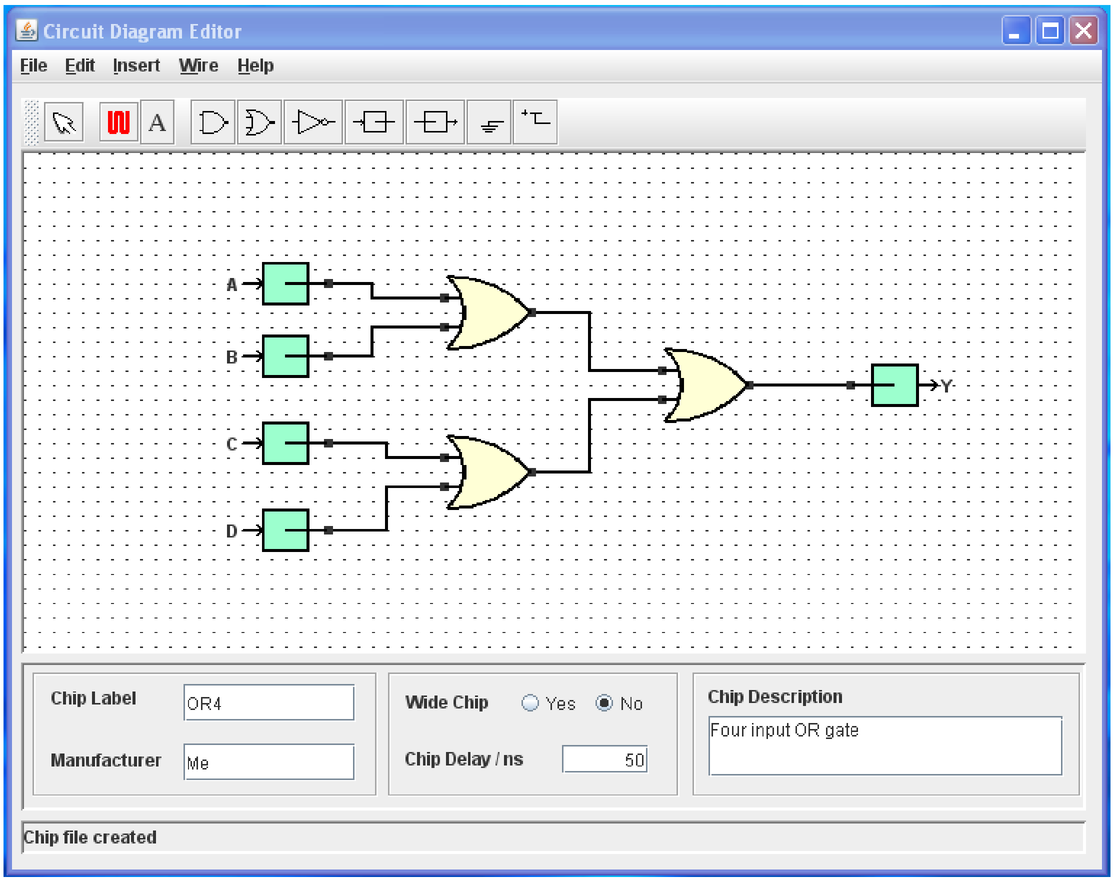
```
ttl -> generic : Logic
```

selecting the generic Logic component as previously described. Click on OK. This will open the ‘Open’ window allowing the user to select a .chp file that will configure this generic logic IC. Using this file an IC of the correct size will be instantiated, as shown in figure 14. This example is of a four input OR as defined in figure 13. Note, as there are more inputs than outputs some of the output pin positions are defined as NC i.e. pins 7, 8 and 9. VCC and GND being assigned pins 5 and 10 respectively.

An alternative method to defining a logic custom IC’s truth table is to use the schematic capture editor. To open the schematic capture editor click on the pull down:

```
Tools -> Schematic Capture Editor
```

This will open the Circuit Diagram Editor window allowing the user to specify the required logic function as a circuit diagram, as shown in figure 15. Note, the schematic capture editor only supports combinational logic circuits implemented from AND, OR and NOT gates.



Chip Label: OR4
Manufacturer: Me
Wide Chip: Yes [✓ No]
Chip Delay / ns: 50
Chip Description: Four input OR gate
Chip file created

Figure 15: schematic capture editor of a four input OR gate

To add an input pin either click on the 'Add Chip Input Pin' icon or click on the pull down:

```
Insert -> Input Pin
```

This will open the 'Pin Label' window, allowing the user to assign a name to this pin. Maximum size 12 characters. Enter a name and click OK. The user can now position this input pin on the schematic by moving the mouse pointer to the desired position and performing a single left click. To move a pin once position left click, hold and drag the pin to the new position. To delete a pin left click the pin and press the DEL key or click on the pull down:

```
Edit -> Delete
```

If you can not select the desired component you may need to switch to 'Select Mode', this can be achieved by click on the 'Selector' icon .

To add an output pin either click on the 'Add Chip Output Pin' icon or click on the pull down:

```
Insert -> Output Pin
```

Operations as for input pin. The combinational logic linking the input and output pins is constructed from a network of AND, OR, NOT gates and constants. To add one of these components to a schematic either click on the pull down:

```
Insert -> AND Gate
       -> OR Gate
       -> NOT Gate
       -> Ground
       -> VCC
```

or its associated icon. Operations as for pins. To add wire interconnects to a circuit either click on the 'Wiring Mode' icon or click on the pull down:

```
Wire -> Entering Wiring Mode
```

To connect a wire between two points single left click on the starting grid position and double left click on the destination grid position. To route a wire around components i.e. turn through a 90 degree bend, single left click on the grid at the point you wish to create the bend. If an anytime you wish to un-route a wire segment you have laid down press the ESC key or click on the pull down:

```
Wire -> Cancel Wire Segment
```

This will sequentially remove each segment back to and including the initial starting point. Note, the default wire colour used by the quick launch icon is black, this being updated to the most recent pull down menu selection.

```
Wire -> Black
     -> Red
     -> Orange
     -> Yellow
     -> Green
     -> Blue
     -> Custom
```

To delete a wire once laid, left click on the wire and press the DEL key, or click on the pull down:

```
Edit -> Delete
```

To allow you to select a wire you may need to switch from 'Wiring Mode' to 'Select Mode', this can be achieved by click on the 'Selector' icon or click on the pull down:

```
Wire -> Exit Wiring Mode
```

Note, moving a component e.g. an AND gate will automatically delete its attached wires. The name and a description of the IC using this schematic can be entered in the bottom panel of this window. Saving and instantiating a logic custom IC based on this circuit is the same as for the truth table editor.

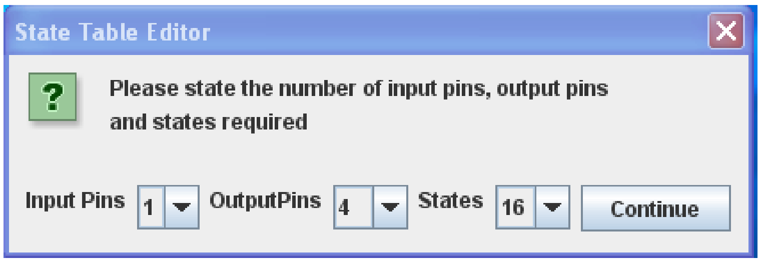
The truth table editor and schematic entry tools allow the user to produce custom combinational logic blocks e.g. address decoders and constant bit patterns e.g. initialisation data, they can not be used to design synchronous logic components. Small synchronous logic ICs can be designed using the state table editor, however, as the user has to manually enter all state information it is not suitable for large circuits. The state machine custom IC package has two additional pins: CLK and CLR, as shown in figure 11. The CLR pin is a synchronous clear resetting the ICs state back to its default initial conditions. The CLK pin is rising edge sensitive clock driving the internal D-type state flip-flops.

To open the state table editor click on the pull down:

```
Tools -> State Table Editor
```

This will open the initial State Table Editor window allowing the user to specify the required number of inputs, outputs and states using the associated pull down boxes, as shown in figure 16. The maximum number of inputs, outputs and states is currently limited to 8, 16 and 16 respectively, in this example 1 input, 4 outputs and 16 states have been selected. Note, any combination of inputs and outputs are allowed. Where required 'not connected' (NC) pins will be automatically inserted to pad out unused pin positions within the package foot print.

Left click the Continue button to proceed. This will launch the main State Table Editor window, allowing you to define the IC's present state, next state relationship, as shown in figure 17. To change the output state of a specific input state to the required value single left click on that output's column bit i.e. the light green column.

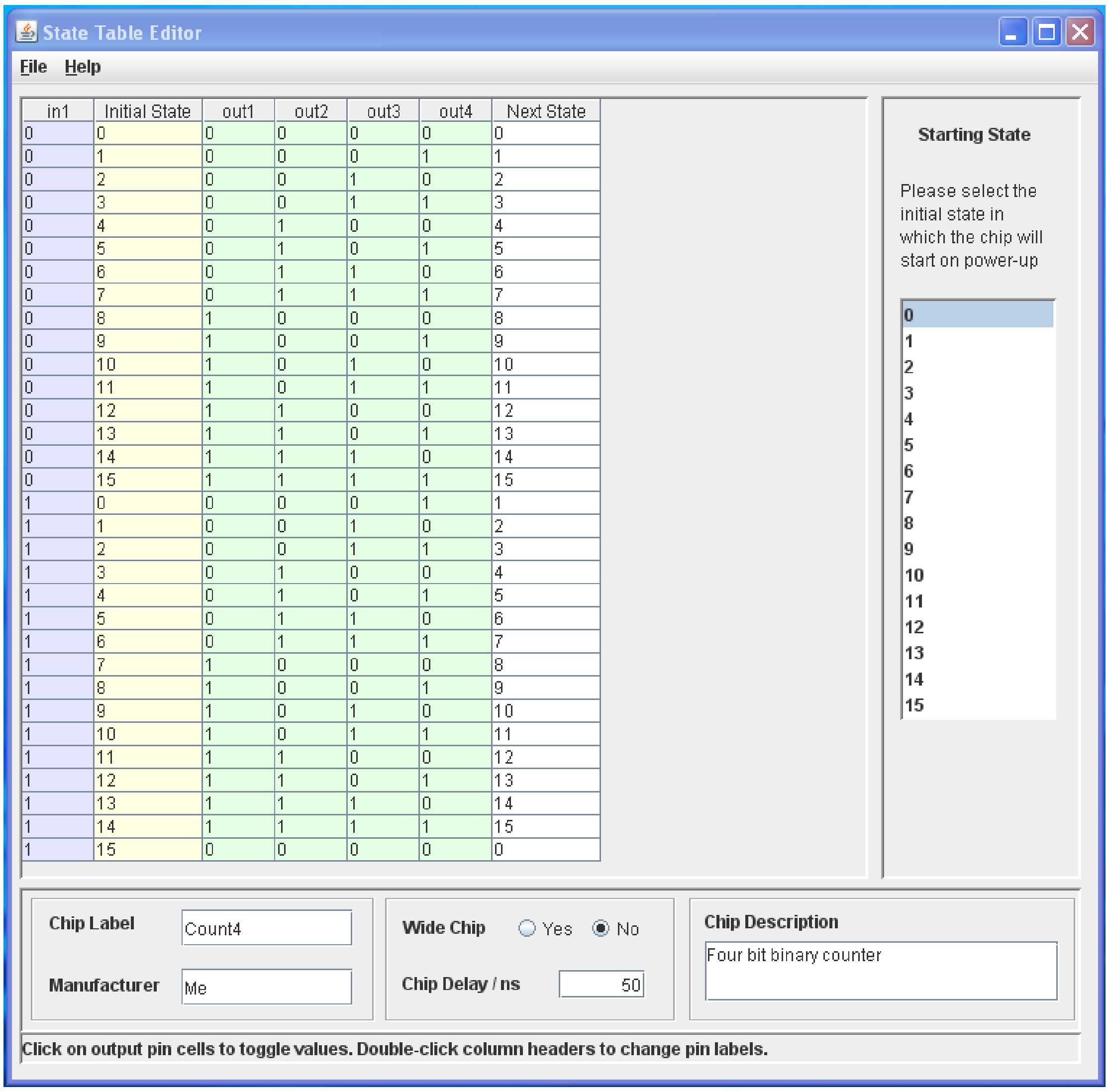

State Table Editor

Please state the number of input pins, output pins and states required

Input Pins 1 OutputPins 4 States 16

Figure 16: initial input pin, output pin and state selection

This will toggle the bit value from 0→1, or 1→0. To change the next state of a specific input state to the required value single left click on that next state's column bit i.e. the white column. This will open a pull down menu from which you can select the desired next state. Note, the initial starting state of this IC can be selected from the right hand side Starting State panel. The user may also edit the input and output names by double left clicking on the default names e.g. In1, In2, Out1 etc, at the top of each column.



in1	Initial State	out1	out2	out3	out4	Next State
0	0	0	0	0	0	0
0	1	0	0	0	1	1
0	2	0	0	1	0	2
0	3	0	0	1	1	3
0	4	0	1	0	0	4
0	5	0	1	0	1	5
0	6	0	1	1	0	6
0	7	0	1	1	1	7
0	8	1	0	0	0	8
0	9	1	0	0	1	9
0	10	1	0	1	0	10
0	11	1	0	1	1	11
0	12	1	1	0	0	12
0	13	1	1	0	1	13
0	14	1	1	1	0	14
0	15	1	1	1	1	15
1	0	0	0	0	1	1
1	1	0	0	1	0	2
1	2	0	0	1	1	3
1	3	0	1	0	0	4
1	4	0	1	0	1	5
1	5	0	1	1	0	6
1	6	0	1	1	1	7
1	7	1	0	0	0	8
1	8	1	0	0	1	9
1	9	1	0	1	0	10
1	10	1	0	1	1	11
1	11	1	1	0	0	12
1	12	1	1	0	1	13
1	13	1	1	1	0	14
1	14	1	1	1	1	15
1	15	0	0	0	0	0

Starting State

Please select the initial state in which the chip will start on power-up

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Chip Label: Count4

Manufacturer: Me

Wide Chip: Yes [✓ No]

Chip Delay / ns: 50

Chip Description: Four bit binary counter

Click on output pin cells to toggle values. Double-click column headers to change pin labels.

Figure 17: state table editor for a four bit binary counter

The design example shown in figure 17 is for a four bit binary counter with an input enable EN (in1). This line is sampled on the rising edge of the CLK. If the CLR is high the counter is reset to its initial state, state 0, otherwise the EN is tested:

1) EN low: counter not incremented. Next state set to present state. Output value maintained.
2) EN high: counter incremented. Present state set to Next state i.e. the next count state. Output value updated to the next binary value i.e. current value + 1.

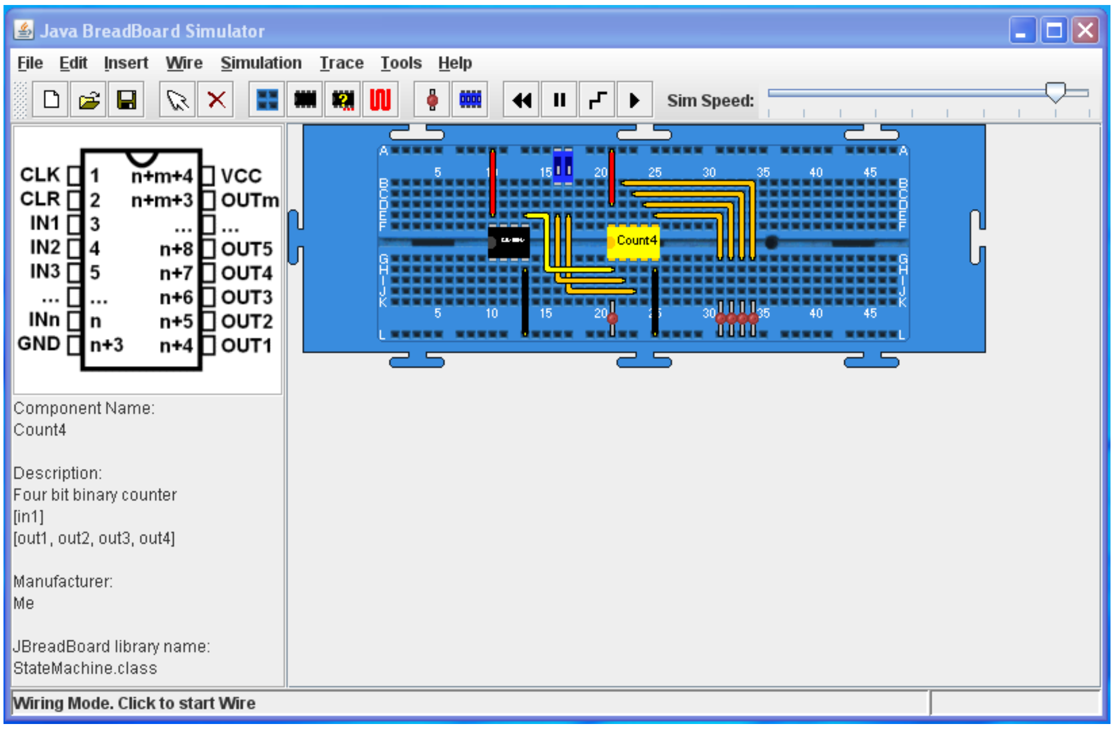
The name and a description of this IC can be entered in the bottom panel of this window. A common propagation delay or Chip Delay i.e. the delay from an input changing to an output being updated, is used for all outputs and can be specified in nano seconds. One final option available to the user is to specify if this IC should use a .600 (Wide) or .300 (Default) package size. This is option is purely to allow the user to match the ICs profile to aid in routing design of real implementations.

Once the truth table and other details have been entered this data can be stored to a file. This can be accessed by clicking on the pull down:

```
File -> Create Chip File
```

This will open a ‘Save’ window allowing the user to specify an output file (.chp extention). Enter a file name and left click on the Save button. This will open the ‘Chip File Created’ window, you may either continue i.e. enter another truth table or exit back to the main interface. If you select continue you may re-edit the current truth table and save it under a different name. To change the number of inputs, outputs or states click on the pull down:

```
File -> New
```



Component Name:
Count4

Description:
Four bit binary counter
[in1]
[out1, out2, out3, out4]

Manufacturer:
Me

JBreadBoard library name:
StateMachine.class

Wiring Mode. Click to start Wire

Figure 18: inserting a custom state machine IC

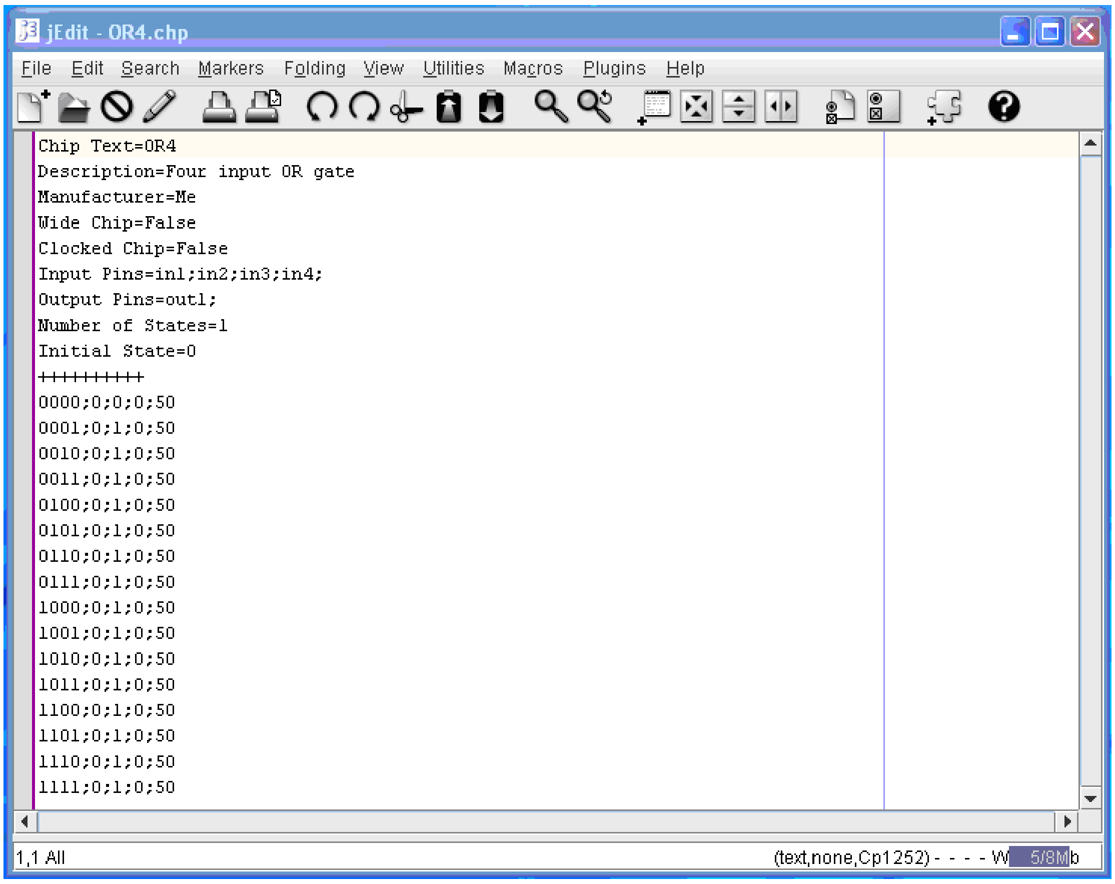
To instantiate this new custom IC on a bread board click on the ‘Select and Add Chip’ icon or click on the pull down:

```
Insert -> Chip
```

Navigate down the directory hierarchy

```
ttl -> generic : StateMachine
```

selecting the generic State Machine component as previously described. Click on OK. This will open the ‘Open’ window allowing the user to select a .chp file that will configure this generic state machine IC. Using this file an IC of the correct size will be instantiated, as shown in figure 18. This example is of a four bit counter as defined in figure 17. Note, as there are more outputs than inputs one of the input pin positions is defined as NC i.e. pins 4. VCC and GND being assigned pins 5 and 10 respectively. This example shows a fully functional test circuit with the CLK line driven by an oscillator IC, inputs CLR and EN controlled by a DIP switch and the outputs displayed on a bank of five LEDS.


jEdit - OR4.chp
File Edit Search Markers Folding View Utilities Macros Plugins Help
1,1 All
(text,none,Cp1252) - - - - W 5/8Mb

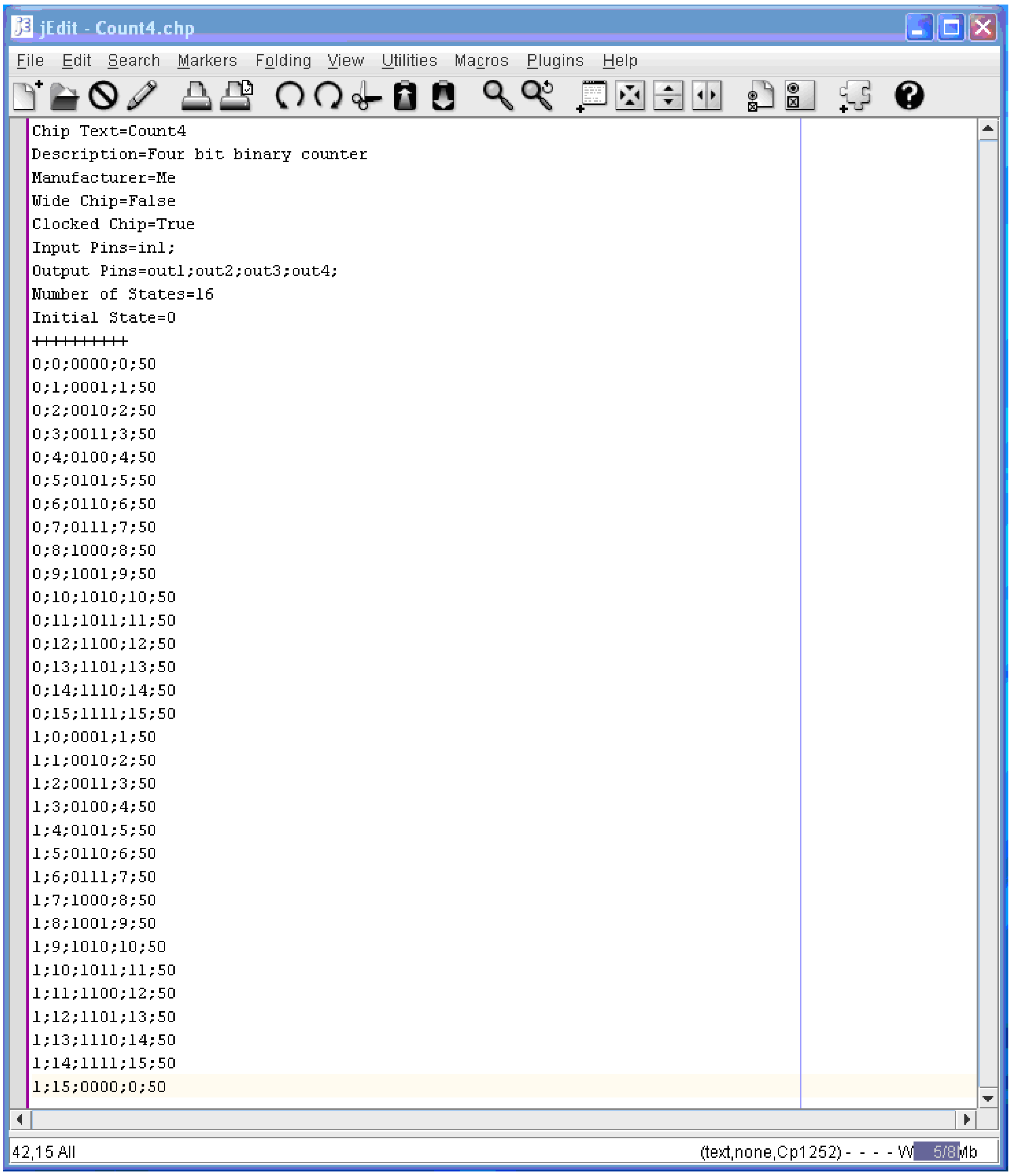
```
Chip Text=OR4
Description=Four input OR gate
Manufacturer=Me
Wide Chip=False
Clocked Chip=False
Input Pins=in1;in2;in3;in4;
Output Pins=out1;
Number of States=1
Initial State=0
++++++++++
0000;0;0;0;50
0001;0;1;0;50
0010;0;1;0;50
0011;0;1;0;50
0100;0;1;0;50
0101;0;1;0;50
0110;0;1;0;50
0111;0;1;0;50
1000;0;1;0;50
1001;0;1;0;50
1010;0;1;0;50
1011;0;1;0;50
1100;0;1;0;50
1101;0;1;0;50
1110;0;1;0;50
1111;0;1;0;50
```

Figure 19: .chp for a four input OR gate

At present there is now software support to allow you to edit .chp files. However, these text files can be manually edited using a standard text editor. Both the custom logic and state machine ICs use the same file format as shown in figure 19 and 20 respectively. Lines 1 – 9 contain data relating to the name and description of the IC. The remaining line contains a semicolon delimited present state / next state table defining the IC's functionality. The row format is:

Input state; Present internal state; Output state; Next internal state; Delay

The internal state information contained in a custom logic IC .chp file is not used as it does not contain any memory elements. Note, to update these manual changes into the simulator the virtual circuit must be reloaded into the Java bread board.


jEdit - Count4.chp
File Edit Search Markers Folding View Utilities Macros Plugins Help

```
Chip Text=Count4
Description=Four bit binary counter
Manufacturer=Me
Wide Chip=False
Clocked Chip=True
Input Pins=in1;
Output Pins=out1;out2;out3;out4;
Number of States=16
Initial State=0
++++++++++
0;0;0000;0;50
0;1;0001;1;50
0;2;0010;2;50
0;3;0011;3;50
0;4;0100;4;50
0;5;0101;5;50
0;6;0110;6;50
0;7;0111;7;50
0;8;1000;8;50
0;9;1001;9;50
0;10;1010;10;50
0;11;1011;11;50
0;12;1100;12;50
0;13;1101;13;50
0;14;1110;14;50
0;15;1111;15;50
1;0;0001;1;50
1;1;0010;2;50
1;2;0011;3;50
1;3;0100;4;50
1;4;0101;5;50
1;5;0110;6;50
1;6;0111;7;50
1;7;1000;8;50
1;8;1001;9;50
1;9;1010;10;50
1;10;1011;11;50
1;11;1100;12;50
1;12;1101;13;50
1;13;1110;14;50
1;14;1111;15;50
1;15;0000;0;50
```

Figure 20: .chp for a four bit Counter

User defined Integrated Circuits

If the required integrated circuit is not included in the pre-installed catalogue and can not be implemented using custom ICs, the user can define their own components. All ICs used in the Java bread board simulator are coded in the Java programming language and are based on a common superclass: IntegratedCircuit.java, as shown in figure 21. An integrated circuit is defined as a component with zero or more pins. Each pin associated with an IC is based on a common superclass: Pin.java, these defining the role of that pin and its functionality as shown in figures 21 and 22. The IntegratedCircuit class has been designed to support the most common functionality found in an integrated circuit e.g. identifying a pins position on a package, or its logical state. To illustrate how the user can develop a new integrated circuit the

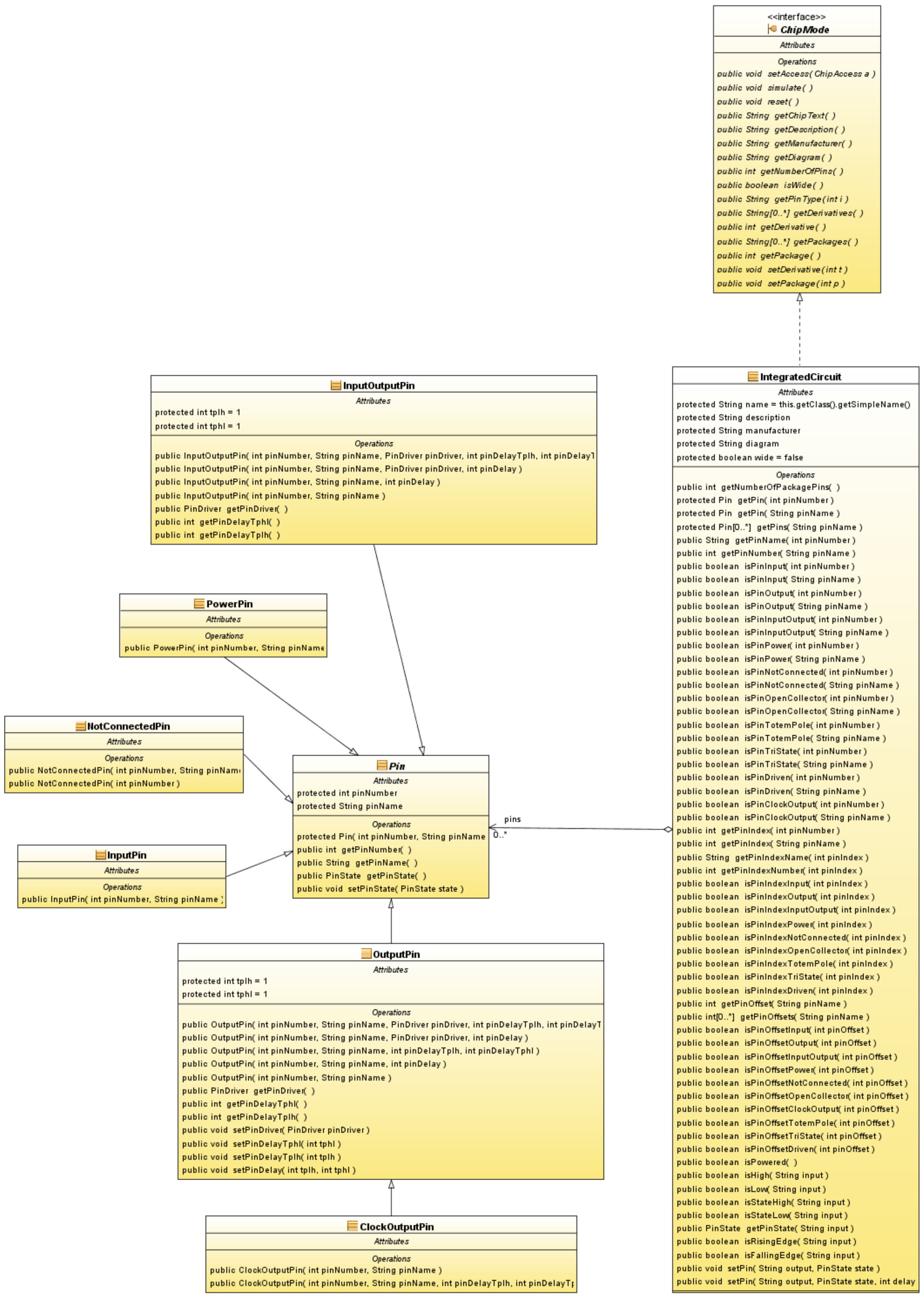

<<interface>>
ChipMode
Attributes
Operations
public void setAccess(ChipAccess a)
public void simulate( )
public void reset( )
public String getChipText( )
public String getDescription( )
public String getManufacturer( )
public String getDiagram( )
public int getNumberOfPins( )
public boolean isWide( )
public String getPinType(int i)
public String[0..*] getDerivatives( )
public int getDerivative( )
public String[0..*] getPackages( )
public int getPackage( )
public void setDerivative(int t)
public void setPackage(int p)
IntegratedCircuit
Attributes
protected String name = this.getClass().getSimpleName()
protected String description
protected String manufacturer
protected String diagram
protected boolean wide = false
Operations
public int getNumberOfPackagePins( )
protected Pin getPin(int pinNumber)
protected Pin getPin(String pinName)
protected Pin[0..*] getPins(String pinName)
public String getPinName(int pinNumber)
public int getPinNumber(String pinName)
public boolean isPinInput(int pinNumber)
public boolean isPinInput(String pinName)
public boolean isPinOutput(int pinNumber)
public boolean isPinOutput(String pinName)
public boolean isPinInputOutput(int pinNumber)
public boolean isPinInputOutput(String pinName)
public boolean isPinPower(int pinNumber)
public boolean isPinPower(String pinName)
public boolean isPinNotConnected(int pinNumber)
public boolean isPinNotConnected(String pinName)
public boolean isPinOpenCollector(int pinNumber)
public boolean isPinOpenCollector(String pinName)
public boolean isPinTotemPole(int pinNumber)
public boolean isPinTotemPole(String pinName)
public boolean isPinTriState(int pinNumber)
public boolean isPinTriState(String pinName)
public boolean isPinDriven(int pinNumber)
public boolean isPinDriven(String pinName)
public boolean isPinClockOutput(int pinNumber)
public boolean isPinClockOutput(String pinName)
public int getPinIndex(int pinNumber)
public int getPinIndex(String pinName)
public String getPinIndexName(int pinIndex)
public int getPinIndexNumber(int pinIndex)
public boolean isPinIndexInput(int pinIndex)
public boolean isPinIndexOutput(int pinIndex)
public boolean isPinIndexInputOutput(int pinIndex)
public boolean isPinIndexPower(int pinIndex)
public boolean isPinIndexNotConnected(int pinIndex)
public boolean isPinIndexOpenCollector(int pinIndex)
public boolean isPinIndexTotemPole(int pinIndex)
public boolean isPinIndexTriState(int pinIndex)
public boolean isPinIndexDriven(int pinIndex)
public int getPinOffset(String pinName)
public int[0..*] getPinOffsets(String pinName)
public boolean isPinOffsetInput(int pinOffset)
public boolean isPinOffsetOutput(int pinOffset)
public boolean isPinOffsetInputOutput(int pinOffset)
public boolean isPinOffsetPower(int pinOffset)
public boolean isPinOffsetNotConnected(int pinOffset)
public boolean isPinOffsetOpenCollector(int pinOffset)
public boolean isPinOffsetClockOutput(int pinOffset)
public boolean isPinOffsetTotemPole(int pinOffset)
public boolean isPinOffsetTriState(int pinOffset)
public boolean isPinOffsetDriven(int pinOffset)
public boolean isPowered( )
public boolean isHigh(String input)
public boolean isLow(String input)
public boolean isStateHigh(String input)
public boolean isStateLow(String input)
public PinState getPinState(String input)
public boolean isRisingEdge(String input)
public boolean isFallingEdge(String input)
public void setPin(String output, PinState state)
public void setPin(String output, PinState state, int delay
InputOutputPin
Attributes
protected int tplh = 1
protected int tphl = 1
Operations
public InputOutputPin(int pinNumber, String pinName, PinDriver pinDriver, int pinDelayTplh, int pinDelayT
public InputOutputPin(int pinNumber, String pinName, PinDriver pinDriver, int pinDelay)
public InputOutputPin(int pinNumber, String pinName, int pinDelay)
public InputOutputPin(int pinNumber, String pinName)
public PinDriver getPinDriver( )
public int getPinDelayTphl( )
public int getPinDelayTplh( )
PowerPin
Attributes
Operations
public PowerPin(int pinNumber, String pinName
NotConnectedPin
Attributes
Operations
public NotConnectedPin(int pinNumber, String pinNam
public NotConnectedPin(int pinNumber)
InputPin
Attributes
Operations
public InputPin(int pinNumber, String pinName)
Pin
Attributes
protected int pinNumber
protected String pinName
Operations
protected Pin(int pinNumber, String pinName
public int getPinNumber( )
public String getPinName( )
public PinState getPinState( )
public void setPinState(PinState state)
pins
0..*
OutputPin
Attributes
protected int tplh = 1
protected int tphl = 1
Operations
public OutputPin(int pinNumber, String pinName, PinDriver pinDriver, int pinDelayTplh, int pinDelayT
public OutputPin(int pinNumber, String pinName, PinDriver pinDriver, int pinDelay)
public OutputPin(int pinNumber, String pinName, int pinDelayTplh, int pinDelayTphl)
public OutputPin(int pinNumber, String pinName, int pinDelay)
public OutputPin(int pinNumber, String pinName)
public PinDriver getPinDriver( )
public int getPinDelayTphl( )
public int getPinDelayTplh( )
public void setPinDriver(PinDriver pinDriver)
public void setPinDelayTphl(int tphl)
public void setPinDelayTplh(int tplh)
public void setPinDelay(int tplh, int tphl)
ClockOutputPin
Attributes
Operations
public ClockOutputPin(int pinNumber, String pinName)
public ClockOutputPin(int pinNumber, String pinName, int pinDelayTplh, int pinDelayTp

Figure 21: class hierarchy

7400 NAND gate will be used as a case study. Most ICs will contain a family of functionally comparable variants e.g. different manufactures, timings etc. A key design goal when implementing a new design is to capture and encapsulate the IC's core functionality as a generic class, refining this model in subclasses to match specific device timings, as shown in figure 23. Therefore, minimising the amount of new code that needs to be written and modifications to existing tested code.

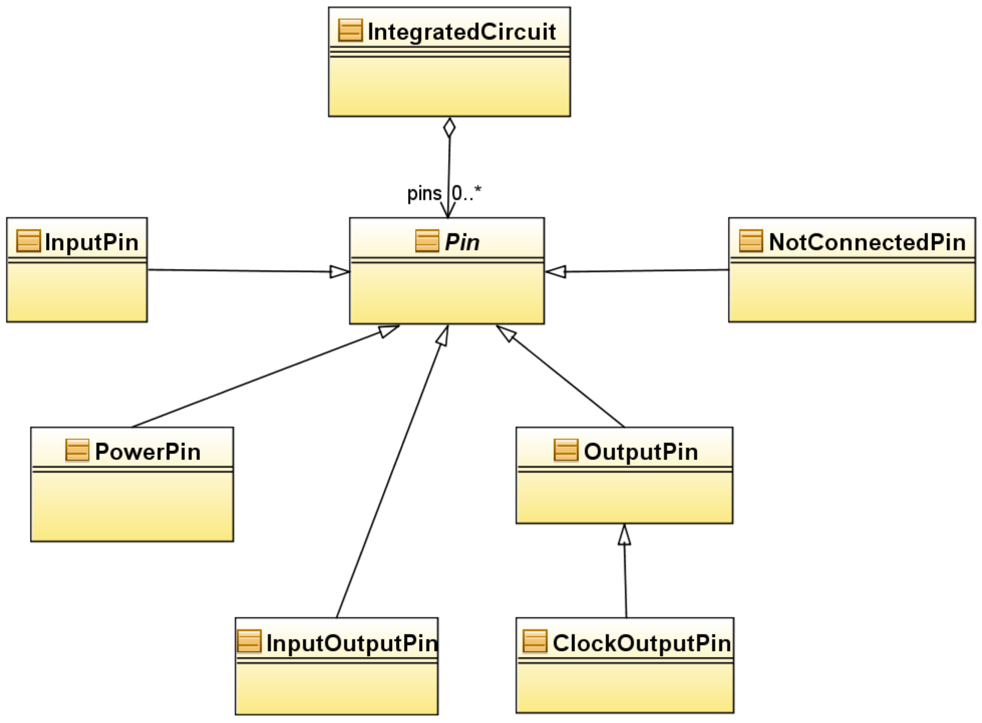


Figure 22: integratedCircuit.java class

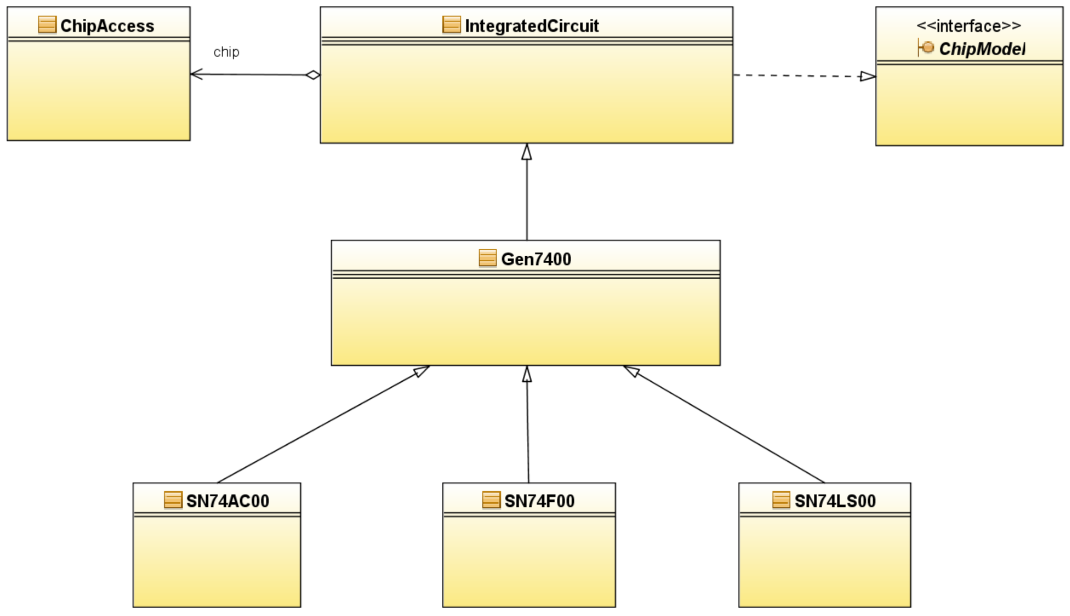
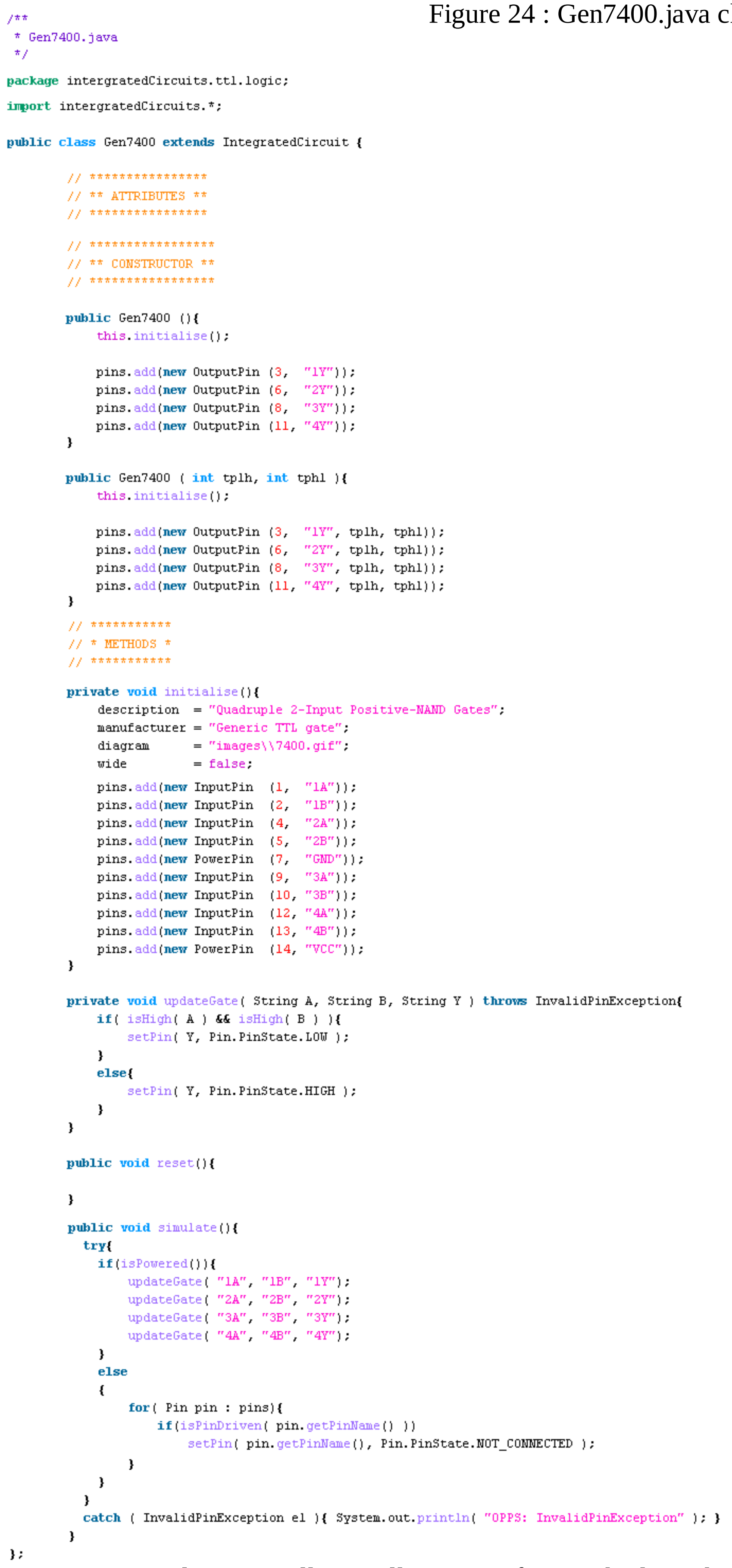


Figure 23: expanding an integrated circuit family

Figure 24 : Gen7400.java class



```
/**
 * Gen7400.java
 */
package intergratedCircuits.ttl.logic;
import intergratedCircuits.*;

public class Gen7400 extends IntegratedCircuit {

        // ****************
        // ** ATTRIBUTES **
        // ****************

        // *****************
        // ** CONSTRUCTOR **
        // *****************

        public Gen7400 (){
            this.initialise();

            pins.add(new OutputPin (3,  "1Y"));
            pins.add(new OutputPin (6,  "2Y"));
            pins.add(new OutputPin (8,  "3Y"));
            pins.add(new OutputPin (11, "4Y"));
        }

        public Gen7400 ( int tplh, int tphl ){
            this.initialise();

            pins.add(new OutputPin (3,  "1Y", tplh, tphl));
            pins.add(new OutputPin (6,  "2Y", tplh, tphl));
            pins.add(new OutputPin (8,  "3Y", tplh, tphl));
            pins.add(new OutputPin (11, "4Y", tplh, tphl));
        }
        // ***********
        // * METHODS *
        // ***********

        private void initialise(){
            description  = "Quadruple 2-Input Positive-NAND Gates";
            manufacturer = "Generic TTL gate";
            diagram      = "images\\7400.gif";
            wide         = false;
            pins.add(new InputPin  (1,  "1A"));
            pins.add(new InputPin  (2,  "1B"));
            pins.add(new InputPin  (4,  "2A"));
            pins.add(new InputPin  (5,  "2B"));
            pins.add(new PowerPin  (7,  "GND"));
            pins.add(new InputPin  (9,  "3A"));
            pins.add(new InputPin  (10, "3B"));
            pins.add(new InputPin  (12, "4A"));
            pins.add(new InputPin  (13, "4B"));
            pins.add(new PowerPin  (14, "VCC"));
        }

        private void updateGate( String A, String B, String Y ) throws InvalidPinException{
            if( isHigh( A ) && isHigh( B ) ){
                setPin( Y, Pin.PinState.LOW );
            }
            else{
                setPin( Y, Pin.PinState.HIGH );
            }
        }

        public void reset(){

        }

        public void simulate(){
          try{
            if(isPowered()){
                updateGate( "1A", "1B", "1Y");
                updateGate( "2A", "2B", "2Y");
                updateGate( "3A", "3B", "3Y");
                updateGate( "4A", "4B", "4Y");
            }
            else
            {
                for( Pin pin : pins){
                    if(isPinDriven( pin.getPinName() ))
                        setPin( pin.getPinName(), Pin.PinState.NOT_CONNECTED );
                }
            }
          }
          catch ( InvalidPinException e1 ){ System.out.println( "OPPS: InvalidPinException" ); }
        }
};
```

A new integrated circuit will typically contain five methods as shown in figure 24:

- Constructor
- Initialise
- UpdateGate
- Reset
- Simulate

Different class constructors allow a user to pass specific timing data to each instantiated object. In this example the Gen7400 has two constructors, the default with no parameters instantiates an object using the default parameters specified in IntegratedCircuit.java. Where as the second constructor allows output rise and fall times to be specified. Common to both of these constructor is an `initialise()` method defining the IC's description and common input, not connected and power pins. To allow a new IC to be integrated into the Java bread board simulator the methods `reset()` and `simulate()` must be supported. As this device is a purely combinational logic design i.e. has no state information, the reset function contains no functionality. For synchronous devices this method would be used to reset all ICs to their default initial conditions at the start of a simulation. During a simulation if the simulator detects that an IC's input state has changed the `simulate()` method is called. This method first determines if this specific instance is powered i.e. VCC pin connected to +5V and GND pin is connected to 0v. If it is not then all pins are set to a not_connected state i.e. effectively removing the IC from the circuit. If it is powered each output is updated using the `updateGate()` method. Using this approach means that the software structure can be used to implement a number of different ICs e.g. Gen7400, Gen7408, Gen7432 etc.

Once an ICs core functionality has been defined specific derivatives can be added. As these devices are functionality equivalent very little additional code needs to be defined. An example of a 74LS00 is shown in figure 26. In this example the package's foot print are identical, however, the IC's timing data is manufacturer dependent. Therefore, the user only needs to extend the Gen7400 class passing this new data. There are three different options in how this can be implemented as shown in figure 26. Once a new IC class has been design the user just needs to copy the .class file to the integrated circuit directory hierarchy, as shown in figure 25, the Java bread board software will automatically detect this IC the next time an IC is added.

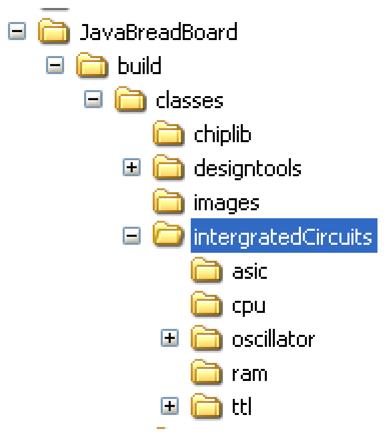
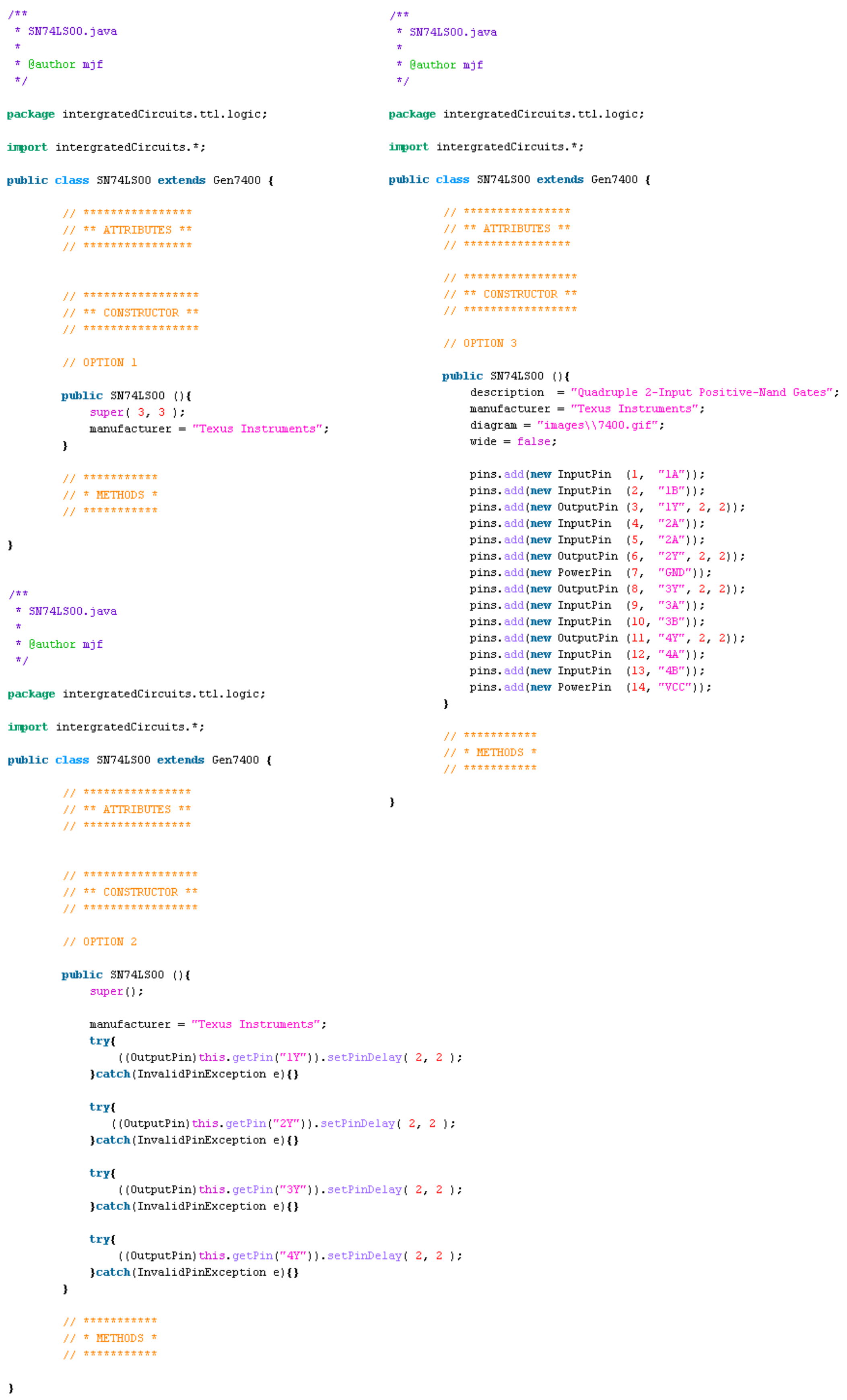


Figure 25: IC directory hierarchy



```
/**
 * SN74LS00.java
 *
 * @author mjf
 */

package intergratedCircuits.ttl.logic;

import intergratedCircuits.*;

public class SN74LS00 extends Gen7400 {

        // ****************
        // ** ATTRIBUTES **
        // ****************


        // *****************
        // ** CONSTRUCTOR **
        // *****************

        // OPTION 1

        public SN74LS00 (){
            super( 3, 3 );
            manufacturer = "Texus Instruments";
        }

        // ***********
        // * METHODS *
        // ***********

}
```

```
/**
 * SN74LS00.java
 *
 * @author mjf
 */

package intergratedCircuits.ttl.logic;

import intergratedCircuits.*;

public class SN74LS00 extends Gen7400 {

        // ****************
        // ** ATTRIBUTES **
        // ****************


        // *****************
        // ** CONSTRUCTOR **
        // *****************

        // OPTION 2

        public SN74LS00 (){
            super();

            manufacturer = "Texus Instruments";
            try{
                ((OutputPin)this.getPin("1Y")).setPinDelay( 2, 2 );
            }catch(InvalidPinException e){}

            try{
               ((OutputPin)this.getPin("2Y")).setPinDelay( 2, 2 );
            }catch(InvalidPinException e){}

            try{
                ((OutputPin)this.getPin("3Y")).setPinDelay( 2, 2 );
            }catch(InvalidPinException e){}

            try{
                ((OutputPin)this.getPin("4Y")).setPinDelay( 2, 2 );
            }catch(InvalidPinException e){}
        }

        // ***********
        // * METHODS *
        // ***********

}
```

```
/**
 * SN74LS00.java
 *
 * @author mjf
 */

package intergratedCircuits.ttl.logic;

import intergratedCircuits.*;

public class SN74LS00 extends Gen7400 {

        // ****************
        // ** ATTRIBUTES **
        // ****************

        // *****************
        // ** CONSTRUCTOR **
        // *****************

        // OPTION 3

        public SN74LS00 (){
            description  = "Quadruple 2-Input Positive-Nand Gates";
            manufacturer = "Texus Instruments";
            diagram = "images\\7400.gif";
            wide = false;

            pins.add(new InputPin  (1,  "1A"));
            pins.add(new InputPin  (2,  "1B"));
            pins.add(new OutputPin (3,  "1Y", 2, 2));
            pins.add(new InputPin  (4,  "2A"));
            pins.add(new InputPin  (5,  "2A"));
            pins.add(new OutputPin (6,  "2Y", 2, 2));
            pins.add(new PowerPin  (7,  "GND"));
            pins.add(new OutputPin (8,  "3Y", 2, 2));
            pins.add(new InputPin  (9,  "3A"));
            pins.add(new InputPin  (10, "3B"));
            pins.add(new OutputPin (11, "4Y", 2, 2));
            pins.add(new InputPin  (12, "4A"));
            pins.add(new InputPin  (13, "4B"));
            pins.add(new PowerPin  (14, "VCC"));
        }

        // ***********
        // * METHODS *
        // ***********

}
```

Figure 25: SN74LS00.java class