BinPy Documentation

Release 0.3.1

BinPy Development Team

Sep 27, 2017

Contents

BinPy is a Python library for simulation of circuit elements. It was started as a project to facilitate learning electronics. Now we've added few more things in the roadmap. For more information see our wiki on GitHub.

Contents:

CHAPTER 1

Installation

This is a guide on how to install BinPy on your computer.

Linux

Note: We support python 2.7, 3.2, 3.3 and 3.4. If you have any of these versions of python you are good to go.

Dependencies

We have two dependencies *setuptools* and *ipython*. setuptools is used to facilitate installation and ipython is used to provide a separate shell for BinPy.

To install dependencies

```
sudo apt-get install setuptools ipython-notebook  # Debian OS

sudo yum install python-ipython-notebook # Fedora or CenOS
```

Install with pip

To install pip on debian os

```
sudo apt-get install python-pip
```

To install pip on fedora or cent os

```
sudo yum -y install python-pip
```

To get the latest stable version of BinPy

```
pip install BinPy
```

To get the development version of BinPy

```
pip install https://github.com/BinPy/BinPy/zipball/master
```

Install with git

To install git on your system

```
sudo apt-get install git  # Debian OS

sudo yum install git-core  # Fedora or CentOS
```

To get the latest stable version of BinPy

```
git clone https://github.com/BinPy/BinPy
cd BinPy/
sudo python setup.py install
```

To get the development version of BinPy

```
git clone -b develop https://github.com/BinPy/BinPy
cd BinPy/
sudo python setup.py install
```

Windows

To install dependencies(setuptools and ipython) you can check the following links.

- https://pypi.python.org/pypi/setuptools#windows-8-powershell
- http://ipython.org/install.html

To install BinPy on windows you can use our windows installer.

Note: We generate windows installer only for stable release. To get the development version, you'll have to use git.

Hint: If you have the *git cygwin* or you have the *github for windows* , you can clone the development branch of our repository from https://github.com/BinPy/BinPy.

To install it you will have to run the following commands

```
cd BinPy/
python setup.py install
```

CHAPTER 2

Examples

Algorithms

An example to demostrate functionality of ExpressionConvert.py

```
from __future__ import print_function
from BinPy.algorithms.ExpressionConvert import *
```

```
# Given Expression:
expr = '~(((A^B)|(~a^b^C))) ~^ c'
```

```
# Obtained Expression
converted = convertExpression(expr)

print(converted)
```

```
OR(XNOR(A, B), XNOR(a, b, C, c))
```

```
# Given Expression:
expr = '((A AND B)xor(NOT(B) and C) xor(C and NOT(D)))or   E or NOT(F)'
```

```
# Obtained Expression
converted = convertExpression(expr)

print(converted)
```

```
OR(XOR(AND(A, B), AND(NOT(B), C), AND(C, NOT(D))), E, NOT(F))
```

```
# Obtained Expression with two input gate contraint
converted2 = convertExpression(expr, two_input = 1)

print(converted2)
```

```
OR(XOR(XOR(AND(A, B), AND(NOT(B), C)), AND(C, NOT(D))), OR(E, NOT(F)))
```

```
# Given Expression:
expr = '(A XOR B XOR C)'
```

```
# Obtained Expression
converted = convertExpression(expr)

print(converted)
```

```
XOR(A, B, C)
```

```
# Obtained Expression with two input gate contraint
converted2 = convertExpression(expr, two_input = 1)

print(converted2)
```

```
XOR(A, XOR(B, C))
```

```
# Equivalent Expression with only AND, OR & NOT gates
converted3 = convertExpression(expr, only_and_or_not=1)

print(converted3)
```

```
OR(AND(A, NOR(AND(B, NOT(C)), AND(NOT(B), C))), AND(NOT(A), OR(AND(B, NOT(C)),␣
↪AND(NOT(B), C))))
```

```
# Given Expression
expr = 'A XOR B'
```

```
# Equivalent Expression with only NAND gates
converted = convertExpression(expr, only_nand=1)

print(converted)
```

```
NAND(NAND(A, NAND(A, B)), NAND(B, NAND(A, B)))
```

```
# Equivalent Expression with only NOR gates
converted2 = convertExpression(expr, only_nor=1)

print(converted2)
```

```
NOR(NOR(NOR(A, NOR(A, B)), NOR(B, NOR(A, B))), NOR(NOR(A, NOR(A, B)), NOR(B, NOR(A,
↪ B))))
```

An example to demostrate the usage of make boolean function.

```
from __future__ import print_function
from BinPy.algorithms.makebooleanfunction import *
```

```
# Usage of make_boolean() function
logical_expression, gate_form = make_boolean(['A', 'B', 'C'], [1, 4, 7],␣
↪minterms=True)
```

```
# Print the logical function
print(logical_expression)
```

```
((A AND (NOT B) AND (NOT C)) OR (A AND B AND C) OR ((NOT A) AND (NOT B) AND C))
```

```
# Print the gate form
print(gate_form)
```

```
OR(AND(A, NOT(B), NOT(C)), AND(A, B, C), AND(NOT(A), NOT(B), C))
```

```
# Another example
logical_expression, gate_form = make_boolean(['A', 'B', 'C', 'D'], [1, 4, 7, 0],␣
↪maxterms=True)
```

```
# Print the logical function
print(logical_expression)
```

```
(D OR ((NOT A) AND B) OR (A AND (NOT B) AND C) OR (B AND (NOT C)))
```

```
# Print the gate form
print(gate_form)
```

```
OR(D, AND(NOT(A), B), AND(A, NOT(B), C), AND(B, NOT(C)))
```

Combinational

Decoder

A decoder is a device which does the reverse operation of an encoder, undoing the encoding so that the original information can be retrieved. The same method used to encode is usually just reversed in order to decode. It is a combinational circuit that converts binary information from n input lines to a maximum of 2n unique output lines.

This example shows you the working of decoder in BinPy.

```
from __future__ import print_function
from BinPy.Combinational.combinational import *

decoder = Decoder(0, 1)  # Initializing a decoder object
print(decoder.output())
```

```
[0, 1, 0, 0]
```

```
decoder.setInput(1, 0)  # Input at index 1 is changed to 0
print (decoder.output())
```

```
[1, 0, 0, 0]
```

```
# Changing the number of inputs
# No need to set the number, just change the inputs
# Input must be power of 2
decoder.setInputs(1, 0, 0)
```

```
print (decoder.getInputStates())  # Get input states
```

```
[1, 0, 0]
```

```
print (decoder.output())
```

```
[0, 0, 0, 0, 1, 0, 0, 0]
```

```
# Using decoders with Connectors
conn = Connector()
decoder.setOutput(1, conn)  # Set Output of decoder to Connector conn
gate1 = AND(conn, 1)  # Put this connector as the input to gate1
print (gate1.output())
```

```
0
```

```
print (decoder) # To get information about the decoder instance
```

```
Decoder Gate; Output: [0, 0, 0, 0, 1, 0, 0, 0]; Inputs: [1, 0, 0];
```

Example for Decoder class

```
# Imports
from __future__ import print_function
from BinPy.combinational.combinational import *
```

```
# Initializing the Decoder class

decoder = Decoder(0, 1)

# Output of decoder

print (decoder.output())
```

```
[0, 1, 0, 0]
```

```
# Input changes

# Input at index 1 is changed to 0

decoder.set_input(1, 0)

# New Output of the decoder

print (decoder.output())
```

```
[1, 0, 0, 0]
```

```
# Changing the number of inputs
# No need to set the number, just change the inputs
# Input must be power of 2

decoder.set_inputs(1, 0, 0)

# To get the input states

print (decoder.get_input_states())
```

```
[1, 0, 0]
```

```
# New output of decoder

print (decoder.output())
```

```
[0, 0, 0, 0, 1, 0, 0, 0]
```

```
# Using Connectors as the input lines

conn = Connector()

# Set Output of decoder to Connector conn

decoder.set_output(1, conn)

# Put this connector as the input to gate1

gate1 = AND(conn, 1)

# Output of the gate1

print (gate1.output())
```

```
0
```

```
# Information about decoder instance can be found by

print (decoder)
```

```
Decoder Gate; Output: [0, 0, 0, 0, 1, 0, 0, 0]; Inputs: [1, 0, 0];
```

Demultiplexer

A demultiplexer (or demux) is a device taking a single input signal and selecting one of many data-output-lines, which is connected to the single input. A demultiplexer is often used with a complementary multiplexer on the receiving end.

This example shows you working procedure of a demultiplexer in BinPy.

```
from __future__ import print_function
from BinPy.Combinational.combinational import *
```

```
demux = DEMUX(1) # Must be a single input
```

```
demux.selectLines(0) #Select Lines must be power of 2
```

```
print (demux.output())
```

```
[1, 0]
```

```
demux.setInput(0, 0) #Input at index 1 is changed to 0"
```

```
print (demux.output()) # New Output of the demux
```

```
[0, 0]
```

```
print (demux.getInputStates()) # Get Input States
```

```
[0]
```

```
# Using Connectors as the input lines
conn = Connector()
demux.setOutput(0, conn)  # Sets conn as the output at index 0
gate1 = AND(conn, 0)  # Put this connector as the input to gate1
```

```
print (gate1.output())  # Output of the gate1
```

```
0
```

```
demux.selectLine(0, 1) # Changing select lines
```

```
print (demux.output()) # New output of demux
```

```
[0, 0]
```

```
# Information about demux instance can be found by using
print(demux)
```

```
DEMUX Gate; Output: [0, 0]; Inputs: [0];
```

Example for DEMUX class.

```
from __future__ import print_function
from BinPy.combinational.combinational import *
```

```
# Initializing the DEMUX class

# Must be a single input

demux = DEMUX(1)

# Put select lines

# Select Lines must be power of 2

demux.select_lines(0)

# Output of demux

print (demux.output())
```

```
[1, 0]
```

```
# Input changes

# Input at index 1 is changed to 0

demux.set_input(0, 0)

# New Output of the demux

print (demux.output())
```

```
[0, 0]
```

```
# Get Input States

print (demux.get_input_states())
```

```
[0]
```

```
# Using Connectors as the input lines

# Take a Connector

conn = Connector()

# Set Output of demux to Connector conn

# sets conn as the output at index 0

demux.set_output(0, conn)

# Put this connector as the input to gate1

gate1 = AND(conn, 0)

# Output of the gate1

print (gate1.output())
```

```
0
```

```
# Changing select lines

#selects input line 2

demux.select_line(0, 1)

# New output of demux

print (demux.output())
```

```
[0, 0]
```

```
# Information about demux instance can be found by

print (demux)
```

```
DEMUX Gate; Output: [0, 0]; Inputs: [0];
```

Encoder

An encoder is a device, circuit, transducer, software program, algorithm or person that converts information from one format or code to another, for the purposes of standardization, speed, secrecy, security or compressions. A simple encoder assigns a binary code to an active input line.

This example shows you how to use encoder in BinPy.

```
from __future__ import print_function
from BinPy.Combinational.combinational import *
```

```
encoder = Encoder(0, 1)  # Exacly 1 input must be 1
print (encoder.output())
```

```
[1]
```

```
# Changing the number of inputs
# No need to set the number, just change the inputs
# Input must be power of 2
encoder.setInputs(0, 0, 0, 1) # Inputs must be power of 2

# To get the input states
print (encoder.getInputStates())
```

```
[0, 0, 0, 1]
```

```
print (encoder.output())
```

```
[1, 1]
```

```
# Using Connectors as the input lines
conn = Connector()
encoder.setOutput(1, conn)  # Set Output of decoder to Connector conn
gate1 = AND(conn, 1)  #Put this connector as the input to gate1
print (gate1.output())
```

```
1
```

```
# Information about encoder instance can be found by
print(encoder)
```

```
Encoder Gate; Output: [1, 1]; Inputs: [0, 0, 0, 1];
```

Example for Encoder class

```
from __future__ import print_function
from BinPy.combinational.combinational import *
```

```
# Initializing the Encoder class

# Exacly 1 input must be 1

encoder = Encoder(0, 1)

# Output of encoder

print (encoder.output())
```

```
[1]
```

```
# Changing the number of inputs

# No need to set the number, just change the inputs
# Input must be power of 2
#Inputs must be power of 2
```

```
encoder.set_inputs(0, 0, 0, 1)

# To get the input states

print (encoder.get_input_states())
```

```
[0, 0, 0, 1]
```

```
# New output of encoder

print (encoder.output())
```

```
[1, 1]
```

```
# Using Connectors as the input lines
# Take a Connector

conn = Connector()

# Set Output of decoder to Connector conn

encoder.set_output(1, conn)

# Put this connector as the input to gate1

gate1 = AND(conn, 1)

# Output of the gate1

print (gate1.output())
```

```
1
```

```
# Information about encoder instance can be found by

print (encoder)
```

```
Encoder Gate; Output: [1, 1]; Inputs: [0, 0, 0, 1];
```

Example for Half Adder class.

```
# Imports

from __future__ import print_function
from BinPy.combinational.combinational import *
```

```
# Initializing the HalfAdder class

ha = HalfAdder(0, 1)

# Output of HalfAdder

print (ha.output())
```

```
[0, 1]
```

```
# The output is of the form [SUM, CARRY]"

# Input changes

# Input at index 1 is changed to 0

ha.set_input(1, 0)

# New Output of the HalfAdder

print (ha.output())
```

```
[0, 0]
```

```
# Changing the number of inputs

# No need to set the number, just change the inputs

# Input length must be two

ha.set_inputs(1, 1)
```

```
# New output of HalfAdder

print (ha.output())
```

```
[1, 0]
```

```
# Using Connectors as the input lines

# Take a Connector

conn = Connector()

# Set Output at index to Connector conn

ha.set_output(0, conn)

# Put this connector as the input to gate1

gate1 = AND(conn, 0)

# Output of the gate1

print (gate1.output())
```

```
0
```

Example for Half Subtractor class

```
# Imports
from __future__ import print_function
from BinPy.combinational.combinational import *
```

```
# Initializing the HalfSubtractor class

hs = HalfSubtractor(0, 1)
```

```
# Output of HalfSubtractor

print (hs.output())
```

```
[1, 1]
```

```
# The output is of the form [DIFFERENCE, BORROW]

# Input changes

# Input at index 1 is changed to 0

hs.set_input(1, 0)

# New Output of the HalfSubtractor

print (hs.output())
```

```
[0, 0]
```

```
# Changing the number of inputs

# No need to set the number, just change the inputs

# Input length must be two

hs.set_inputs(1, 1)
```

```
# New output of HalfSubtractor

print (hs.output())
```

```
[0, 0]
```

```
# Using Connectors as the input lines

# Take a Connector

conn = Connector()

# Set Output at index to Connector conn

hs.set_output(0, conn)

# Put this connector as the input to gate1

gate1 = AND(conn, 0)

# Output of the gate1

print (gate1.output())
```

```
0
```

Example for MUX class.

```
# Imports
from __future__ import print_function
from BinPy.combinational.combinational import *
```

```
# Initializing the MUX class

mux = MUX(0, 1)

# Put select lines

mux.select_lines(0)

# Output of mux

print (mux.output())
```

```
0
```

```
# Input changes

# Input at index 1 is changed to 0

mux.set_input(1, 0)

# New Output of the mux

print (mux.output())
```

```
0
```

```
# Changing the number of inputs

# No need to set the number, just change the inputs

# Input must be power of 2

mux.set_inputs(1, 0, 0, 1)
```

```
# New output of mux

print (mux.output())
```

```
1
```

```
# Using Connectors as the input lines

# Take a Connector

conn = Connector()

# Set Output of mux to Connector conn

mux.set_output(conn)

# Put this connector as the input to gate1

gate1 = AND(conn, 0)
```

```
# Output of the gate1

print (gate1.output())
```

```
0
```

```
# Changing select lines

# Selects input line 2

mux.select_line(0, 1)

# New output of mux

print (mux.output())
```

```
0
```

```
# Information about mux instance can be found by

print (mux)
```

```
MUX Gate; Output: 0; Inputs: [1, 0, 0, 1];
```

ic - Integrated Circuits

Series 4000 ICs

Usage of IC 4000

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 4000:

ic = IC_4000()

print(ic.__doc__)
```

```
Dual 3 Input NOR gate + one NOT gate IC.
Pin_6 = NOR(Pin_3, Pin_4, Pin_5)
Pin_10 = NOR(Pin_11, Pin_12, Pin_13)
Pin_9 = NOT(Pin_8)
```

```
# The Pin configuration is:

inp = {3: 1, 4: 1, 5: 1, 7: 0, 8: 1, 11: 0, 12: 0, 13: 0, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})
```

```
# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
              ------------
              |                    |
    [Z]     -  1             14  -     [1] VCC
              |                    |
              |                    |
    [Z]     -  2       4     13  -     [0] A3:
              |                    |
              |                    |
A1: [1]     -  3       0     12  -     [0] B3:
              |                    |
              |                    |
B1: [1]     -  4       0     11  -     [0] C3:
              |                    |
              |                    |
C1: [1]     -  5       0     10  -     [0] Q3:
              |                    |
              |                    |
Q1: [0]     -  6              9  -     [0] Q2:
              |                    |
              |                    |
GND [0]     -  7              8  -     [1] B2:
              |                    |
              --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{9: 0, 10: 1, 6: 0}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
              ------------
              |                    |
    [Z]     -  1             14  -     [1] VCC
              |                    |
              |                    |
    [Z]     -  2       4     13  -     [0] A3:
              |                    |
              |                    |
A1: [1]     -  3       0     12  -     [0] B3:
              |                    |
              |                    |
B1: [1]     -  4       0     11  -     [0] C3:
```

```
              |                     |
              |                     |
C1: [1]     -  5       0      10  -     [1] Q3:
              |                     |
              |                     |
Q1: [0]     -  6               9  -     [0] Q2:
              |                     |
              |                     |
GND [0]     -  7               8  -     [1] B2:
              |                     |
              --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                     |
     [Z]     -  1             14  -     [1] VCC
               |                     |
               |                     |
     [Z]     -  2       4     13  -     [0] A3:
               |                     |
               |                     |
 A1: [1]     -  3       0     12  -     [0] B3:
               |                     |
               |                     |
 B1: [1]     -  4       0     11  -     [0] C3:
               |                     |
               |                     |
 C1: [1]     -  5       0     10  -     [1] Q3:
               |                     |
               |                     |
 Q1: [0]     -  6               9  -     [0] Q2:
               |                     |
               |                     |
 GND [0]     -  7               8  -     [1] B2:
               |                     |
               --------------
{9: 0, 10: 1, 6: 0}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(9, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 4001

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 4001:

ic = IC_4001()

print(ic.__doc__)
```

```
Dual 4 input AND gate
Pin_1 = AND(Pin_2, Pin_3, Pin_4, Pin_5)
Pin_13 = AND(Pin_9, Pin_10, Pin_11, Pin_12)
```

```
# The Pin configuration is:

inp = {1: 0, 2: 0, 5: 0, 6: 1, 7: 0, 8: 1, 9: 0, 12: 1, 13: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
              ------------
              |                      |
Q1: [0]    -  1             14   -    [1] VCC
              |                      |
              |                      |
A1: [0]    -  2       4     13   -    [0] Q2:
              |                      |
              |                      |
B1: [1]    -  3       0     12   -    [1] A2:
              |                      |
              |                      |
C1: [0]    -  4       8     11   -    [1] B2:
              |                      |
              |                      |
D1: [1]    -  5       2     10   -    [1] C2:
              |                      |
              |                      |
    [Z]    -  6              9   -    [1] D2:
              |                      |
              |                      |
GND [0]    -  7              8   -    [Z]
              |                      |
              -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to
```

```
print (ic.run())
```

```
{1: 0, 13: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
              ------------
              |                     |
Q1: [0]    -  1             14  -    [1] VCC
              |                     |
              |                     |
A1: [0]    -  2       4     13  -    [1] Q2:
              |                     |
              |                     |
B1: [1]    -  3       0     12  -    [1] A2:
              |                     |
              |                     |
C1: [0]    -  4       8     11  -    [1] B2:
              |                     |
              |                     |
D1: [1]    -  5       2     10  -    [1] C2:
              |                     |
              |                     |
    [Z]    -  6              9  -    [1] D2:
              |                     |
              |                     |
GND [0]    -  7              8  -    [Z]
              |                     |
              -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
              ------------
              |                     |
 Q1: [0]    -  1             14  -    [1] VCC
              |                     |
              |                     |
 A1: [0]    -  2       4     13  -    [1] Q2:
              |                     |
              |                     |
 B1: [1]    -  3       0     12  -    [1] A2:
              |                     |
              |                     |
 C1: [0]    -  4       8     11  -    [1] B2:
```

```
                |                     |
                |                     |
 D1: [1]     -  5       2      10   -     [1] C2:
                |                     |
                |                     |
     [Z]     -  6               9   -     [1] D2:
                |                     |
                |                     |
 GND [0]     -  7               8   -     [Z]
                |                     |
                --------------
{1: 0, 13: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(13, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 4002

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 4002:

ic = IC_4002()

print(ic.__doc__)
```

```
Dual 4 input NOR gate
Pin_1 = NOR(Pin_2, Pin_3, Pin_4, Pin_5)
Pin_13 = NOR(Pin_9, Pin_10, Pin_11, Pin_12)
```

```
# The Pin configuration is:

inp = {2: 0, 3: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 11: 1, 12: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
              ------------
              |                   |
```

```
Q1: [0]    -  1            14  -    [1] VCC
              |                 |
              |                 |
A1: [0]    -  2      4     13  -    [0] Q2:
              |                 |
              |                 |
B1: [0]    -  3      0     12  -    [1] A2:
              |                 |
              |                 |
C1: [0]    -  4      0     11  -    [1] B2:
              |                 |
              |                 |
D1: [0]    -  5      2     10  -    [1] C2:
              |                 |
              |                 |
    [Z]    -  6             9  -    [1] D2:
              |                 |
              |                 |
GND [0]    -  7             8  -    [Z]
              |                 |
              -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{1: 1, 13: 0}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
              ------------
              |                 |
Q1: [1]    -  1            14  -    [1] VCC
              |                 |
              |                 |
A1: [0]    -  2      4     13  -    [0] Q2:
              |                 |
              |                 |
B1: [0]    -  3      0     12  -    [1] A2:
              |                 |
              |                 |
C1: [0]    -  4      0     11  -    [1] B2:
              |                 |
              |                 |
D1: [0]    -  5      2     10  -    [1] C2:
              |                 |
              |                 |
    [Z]    -  6             9  -    [1] D2:
              |                 |
              |                 |
GND [0]    -  7             8  -    [Z]
              |                 |
```

```
             -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                    |
 Q1: [1]    -  1             14  -     [1] VCC
               |                    |
               |                    |
 A1: [0]    -  2       4     13  -     [0] Q2:
               |                    |
               |                    |
 B1: [0]    -  3       0     12  -     [1] A2:
               |                    |
               |                    |
 C1: [0]    -  4       0     11  -     [1] B2:
               |                    |
               |                    |
 D1: [0]    -  5       2     10  -     [1] C2:
               |                    |
               |                    |
     [Z]    -  6              9  -     [1] D2:
               |                    |
               |                    |
 GND [0]    -  7              8  -     [Z]
               |                    |
               -------------
{1: 1, 13: 0}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(13, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 4011

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 4011:

ic = IC_4011()

print(ic.__doc__)
```

```
Quad 2 input NAND gate
Pin_3 = NAND(Pin_1, Pin_2)
Pin_4 = NAND(Pin_5, Pin_6)
Pin_10 = NAND(Pin_8, Pin_9)
Pin_11 = NAND(Pin_12, Pin_13)
```

```
# The Pin configuration is:

inp = {1: 0, 2: 0, 5: 0, 6: 1, 7: 0, 8: 1, 9: 0, 12: 1, 13: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
              ------------
              |                      |
A1: [0]    -  1            14   -    [1] VCC
              |                      |
              |                      |
B1: [0]    -  2      4     13   -    [1] A4:
              |                      |
              |                      |
Q1: [0]    -  3      0     12   -    [1] B4:
              |                      |
              |                      |
Q2: [0]    -  4      1     11   -    [0] Q4:
              |                      |
              |                      |
B2: [0]    -  5      1     10   -    [0] Q3:
              |                      |
              |                      |
A2: [1]    -  6             9   -    [0] B3:
              |                      |
              |                      |
GND [0]    -  7             8   -    [1] A3:
              |                      |
              -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{11: 0, 10: 1, 3: 1, 4: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())
```

```
# Draw the final configuration

ic.drawIC()
```

```
              ------------
              |                    |
A1: [0]    -  1             14  -     [1] VCC
              |                    |
              |                    |
B1: [0]    -  2       4     13  -     [1] A4:
              |                    |
              |                    |
Q1: [1]    -  3       0     12  -     [1] B4:
              |                    |
              |                    |
Q2: [1]    -  4       1     11  -     [0] Q4:
              |                    |
              |                    |
B2: [0]    -  5       1     10  -     [1] Q3:
              |                    |
              |                    |
A2: [1]    -  6              9  -     [0] B3:
              |                    |
              |                    |
GND [0]    -  7              8  -     [1] A3:
              |                    |
              -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
              ------------
              |                    |
 A1: [0]    -  1             14  -     [1] VCC
              |                    |
              |                    |
 B1: [0]    -  2       4     13  -     [1] A4:
              |                    |
              |                    |
 Q1: [1]    -  3       0     12  -     [1] B4:
              |                    |
              |                    |
 Q2: [1]    -  4       1     11  -     [0] Q4:
              |                    |
              |                    |
 B2: [0]    -  5       1     10  -     [1] Q3:
              |                    |
              |                    |
 A2: [1]    -  6              9  -     [0] B3:
              |                    |
              |                    |
 GND [0]    -  7              8  -     [1] A3:
```

```
              |                   |
              -------------
{11: 0, 10: 1, 3: 1, 4: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(11, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 4012

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 4012:

ic = IC_4012()

print(ic.__doc__)
```

```
Dual 4 input NAND gate
Pin_1 = NAND(Pin_2, Pin_3, Pin_4, Pin_5)
Pin_13 = NAND(Pin_9, Pin_10, Pin_11, Pin_12)
```

```
# The Pin configuration is:

inp = {2: 0, 3: 1, 4: 0, 5: 1, 7: 0, 9: 1, 10: 1, 11: 1, 12: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
              ------------
              |                   |
Q1: [0]     -  1             14  -     [1] VCC
              |                   |
              |                   |
A1: [0]     -  2       4     13  -     [0] Q2:
              |                   |
              |                   |
B1: [1]     -  3       0     12  -     [1] A2:
              |                   |
              |                   |
```

```
C1: [0]     -  4       1      11  -     [1] B2:
               |                     |
               |                     |
D1: [1]     -  5       2      10  -     [1] C2:
               |                     |
               |                     |
    [Z]     -  6               9  -     [1] D2:
               |                     |
               |                     |
GND [0]     -  7               8  -     [Z]
               |                     |
               --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{1: 1, 13: 0}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
               ------------
               |                     |
Q1: [1]     -  1              14  -     [1] VCC
               |                     |
               |                     |
A1: [0]     -  2       4      13  -     [0] Q2:
               |                     |
               |                     |
B1: [1]     -  3       0      12  -     [1] A2:
               |                     |
               |                     |
C1: [0]     -  4       1      11  -     [1] B2:
               |                     |
               |                     |
D1: [1]     -  5       2      10  -     [1] C2:
               |                     |
               |                     |
    [Z]     -  6               9  -     [1] D2:
               |                     |
               |                     |
GND [0]     -  7               8  -     [Z]
               |                     |
               --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
# Run the IC

print (ic.run())
```

```
              ------------
              |                    |
 Q1: [1]    -  1             14  -     [1] VCC
              |                    |
              |                    |
 A1: [0]    -  2       4     13  -     [0] Q2:
              |                    |
              |                    |
 B1: [1]    -  3       0     12  -     [1] A2:
              |                    |
              |                    |
 C1: [0]    -  4       1     11  -     [1] B2:
              |                    |
              |                    |
 D1: [1]    -  5       2     10  -     [1] C2:
              |                    |
              |                    |
     [Z]    -  6              9  -     [1] D2:
              |                    |
              |                    |
 GND [0]    -  7              8  -     [Z]
              |                    |
              --------------
{1: 1, 13: 0}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(13, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 4023

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 4023:

ic = IC_4023()

print(ic.__doc__)
```

```
Triple 3 input NAND gate
Pin_6 = NAND(Pin_3, Pin_4, Pin_5)
Pin_9 = NAND(Pin_1, Pin_2, Pin_8)
Pin_10 = NAND(Pin_11, Pin_12, Pin_13)
```

```
# The Pin configuration is:
```

```
inp = {1: 1, 2: 1, 3: 0, 4: 0, 5: 0, 7: 0, 8: 1, 11: 0, 12: 1, 13: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
              ------------
              |                      |
C2: [1]    -  1             14   -    [1] VCC
              |                      |
              |                      |
B2: [1]    -  2       4     13   -    [1] C3:
              |                      |
              |                      |
C1: [0]    -  3       0     12   -    [1] B3:
              |                      |
              |                      |
B1: [0]    -  4       2     11   -    [0] A3:
              |                      |
              |                      |
A1: [0]    -  5       3     10   -    [0] Q3:
              |                      |
              |                      |
Q1: [0]    -  6              9   -    [0] Q2:
              |                      |
              |                      |
GND [0]    -  7              8   -    [1] A2:
              |                      |
              -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{9: 0, 10: 1, 6: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
              ------------
              |                      |
C2: [1]    -  1             14   -    [1] VCC
              |                      |
```

```
              |                       |
B2: [1]     -  2       4      13   -     [1] C3:
              |                       |
              |                       |
C1: [0]     -  3       0      12   -     [1] B3:
              |                       |
              |                       |
B1: [0]     -  4       2      11   -     [0] A3:
              |                       |
              |                       |
A1: [0]     -  5       3      10   -     [1] Q3:
              |                       |
              |                       |
Q1: [1]     -  6              9   -     [0] Q2:
              |                       |
              |                       |
GND [0]     -  7              8   -     [1] A2:
              |                       |
              --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                       |
 C2: [1]     -  1             14   -     [1] VCC
               |                       |
               |                       |
 B2: [1]     -  2       4     13   -     [1] C3:
               |                       |
               |                       |
 C1: [0]     -  3       0     12   -     [1] B3:
               |                       |
               |                       |
 B1: [0]     -  4       2     11   -     [0] A3:
               |                       |
               |                       |
 A1: [0]     -  5       3     10   -     [1] Q3:
               |                       |
               |                       |
 Q1: [1]     -  6              9   -     [0] Q2:
               |                       |
               |                       |
 GND [0]     -  7              8   -     [1] A2:
               |                       |
               --------------
{9: 0, 10: 1, 6: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
```

```
ic.setOutput(9, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 4025

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 4025:

ic = IC_4025()

print(ic.__doc__)
```

```
Triple 3 input NOR gate
Pin_6 = NOR(Pin_3, Pin_4, Pin_5)
Pin_9 = NOR(Pin_1, Pin_2, Pin_8)
Pin_10 = NOR(Pin_11, Pin_12, Pin_13)
```

```
# The Pin configuration is:

inp = {1: 1, 2: 1, 3: 0, 4: 0, 5: 0, 7: 0, 8: 1, 11: 0, 12: 1, 13: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
              ------------
              |                  |
C2: [1]    -  1            14  -    [1] VCC
              |                  |
              |                  |
B2: [1]    -  2      4     13  -    [1] C3:
              |                  |
              |                  |
C1: [0]    -  3      0     12  -    [1] B3:
              |                  |
              |                  |
B1: [0]    -  4      2     11  -    [0] A3:
              |                  |
              |                  |
A1: [0]    -  5      5     10  -    [0] Q3:
              |                  |
              |                  |
Q1: [0]    -  6             9  -    [0] Q2:
              |                  |
```

```
              |                    |
GND [0]     -  7              8   -     [1] A2:
              |                    |
              --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{9: 0, 10: 0, 6: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
              ------------
              |                    |
C2: [1]     -  1             14   -     [1] VCC
              |                    |
              |                    |
B2: [1]     -  2       4     13   -     [1] C3:
              |                    |
              |                    |
C1: [0]     -  3       0     12   -     [1] B3:
              |                    |
              |                    |
B1: [0]     -  4       2     11   -     [0] A3:
              |                    |
              |                    |
A1: [0]     -  5       5     10   -     [0] Q3:
              |                    |
              |                    |
Q1: [1]     -  6              9   -     [0] Q2:
              |                    |
              |                    |
GND [0]     -  7              8   -     [1] A2:
              |                    |
              --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
              ------------
              |                    |
```

```
 C2: [1]    -  1            14  -    [1] VCC
               |                 |
               |                 |
 B2: [1]    -  2      4     13  -    [1] C3:
               |                 |
               |                 |
 C1: [0]    -  3      0     12  -    [1] B3:
               |                 |
               |                 |
 B1: [0]    -  4      2     11  -    [0] A3:
               |                 |
               |                 |
 A1: [0]    -  5      5     10  -    [0] Q3:
               |                 |
               |                 |
 Q1: [1]    -  6       9    9  -    [0] Q2:
               |                 |
               |                 |
 GND [0]    -  7       8    8  -    [1] A2:
               |                 |
              -------------
{9: 0, 10: 0, 6: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(9, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 4069

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 4069:

ic = IC_4069()

print(ic.__doc__)
```

```
Hex NOT gate
Pin_2 = NOT(Pin_1)
Pin_4 = NOT(Pin_3)
Pin_6 = NOT(Pin_5)
Pin_8 = NOT(Pin_9)
Pin_10 = NOT(Pin_11)
Pin_12 = NOT(Pin_13)
```

```
# The Pin configuration is:

inp = {2: 0, 3: 1, 4: 0, 5: 1, 7: 0, 9: 1, 10: 1, 11: 1, 12: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0}) -- \n
```

```
ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
              ------------
              |                     |
Inp [0]    -  1             14  -    [1] VCC
              |                     |
              |                     |
Out [0]    -  2       4     13  -    [0] Inp
              |                     |
              |                     |
Inp [1]    -  3       0     12  -    [1] Out
              |                     |
              |                     |
Out [0]    -  4       6     11  -    [1] Inp
              |                     |
              |                     |
Inp [1]    -  5       9     10  -    [1] Out
              |                     |
              |                     |
Out [0]    -  6              9  -    [1] Inp
              |                     |
              |                     |
GND [0]    -  7              8  -    [0] Out
              |                     |
              -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{2: 1, 4: 0, 6: 0, 8: 0, 10: 0, 12: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
              ------------
              |                     |
Inp [0]    -  1             14  -    [1] VCC
              |                     |
              |                     |
Out [1]    -  2       4     13  -    [0] Inp
              |                     |
              |                     |
Inp [1]    -  3       0     12  -    [1] Out
```

```
              |                     |
              |                     |
Out [0]     -  4       6      11  -     [1] Inp
              |                     |
              |                     |
Inp [1]     -  5       9      10  -     [0] Out
              |                     |
              |                     |
Out [0]     -  6              9   -     [1] Inp
              |                     |
              |                     |
GND [0]     -  7              8   -     [0] Out
              |                     |
              --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                     |
 Inp [0]     -  1             14  -     [1] VCC
               |                     |
               |                     |
 Out [1]     -  2       4     13  -     [0] Inp
               |                     |
               |                     |
 Inp [1]     -  3       0     12  -     [1] Out
               |                     |
               |                     |
 Out [0]     -  4       6     11  -     [1] Inp
               |                     |
               |                     |
 Inp [1]     -  5       9     10  -     [0] Out
               |                     |
               |                     |
 Out [0]     -  6              9   -     [1] Inp
               |                     |
               |                     |
 GND [0]     -  7              8   -     [0] Out
               |                     |
               --------------
{2: 1, 4: 0, 6: 0, 8: 0, 10: 0, 12: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(2, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 4070

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 4070:

ic = IC_4070()

print(ic.__doc__)
```

```
Quad 2 input XOR gate
Pin_3 = XOR(Pin_1, Pin_2)
Pin_4 = XOR(Pin_5, Pin_6)
Pin_10 = XOR(Pin_8, Pin_9)
Pin_11 = XOR(Pin_12, Pin_13)
```

```
# The Pin configuration is:

inp = {2: 0, 3: 1, 4: 0, 5: 1, 7: 0, 9: 1, 10: 1, 11: 1, 12: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
              ------------
              |                    |
A1: [0]    -  1             14  -     [1] VCC
              |                    |
              |                    |
B1: [0]    -  2       4     13  -     [0] A4:
              |                    |
              |                    |
Q1: [1]    -  3       0     12  -     [1] B4:
              |                    |
              |                    |
Q2: [0]    -  4       7     11  -     [1] Q4:
              |                    |
              |                    |
B2: [1]    -  5       0     10  -     [1] Q3:
              |                    |
              |                    |
A2: [0]    -  6              9  -     [1] B3:
              |                    |
              |                    |
GND [0]    -  7              8  -     [0] A3:
              |                    |
```

```
             ------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{11: 1, 10: 1, 3: 0, 4: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
              ------------
              |                    |
A1: [0]    -  1             14   -     [1] VCC
              |                    |
              |                    |
B1: [0]    -  2       4     13   -     [0] A4:
              |                    |
              |                    |
Q1: [0]    -  3       0     12   -     [1] B4:
              |                    |
              |                    |
Q2: [1]    -  4       7     11   -     [1] Q4:
              |                    |
              |                    |
B2: [1]    -  5       0     10   -     [1] Q3:
              |                    |
              |                    |
A2: [0]    -  6              9   -     [1] B3:
              |                    |
              |                    |
GND [0]    -  7              8   -     [0] A3:
              |                    |
              -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
              ------------
              |                    |
 A1: [0]    -  1             14   -     [1] VCC
              |                    |
              |                    |
```

```
 B1: [0]     -  2       4      13  -     [0] A4:
                |                      |
                |                      |
 Q1: [0]     -  3       0      12  -     [1] B4:
                |                      |
                |                      |
 Q2: [1]     -  4       7      11  -     [1] Q4:
                |                      |
                |                      |
 B2: [1]     -  5       0      10  -     [1] Q3:
                |                      |
                |                      |
 A2: [0]     -  6              9  -     [1] B3:
                |                      |
                |                      |
 GND [0]     -  7              8  -     [0] A3:
                |                      |
               --------------
{11: 1, 10: 1, 3: 0, 4: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(3, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 4071

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 4071:

ic = IC_4071()

print(ic.__doc__)
```

```
Quad 2 input OR gate
Pin_3 = OR(Pin_1, Pin_2)
Pin_4 = OR(Pin_5, Pin_6)
Pin_10 = OR(Pin_8, Pin_9)
Pin_11 = OR(Pin_12, Pin_13)
```

```
# The Pin configuration is:

inp = {2: 0, 3: 1, 4: 0, 5: 1, 7: 0, 9: 1, 10: 1, 11: 1, 12: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0}) -- \n

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic
```

```
ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
              ------------
              |                    |
A1: [0]    -  1             14  -     [1] VCC
              |                    |
              |                    |
B1: [0]    -  2       4     13  -     [0] A4:
              |                    |
              |                    |
Q1: [1]    -  3       0     12  -     [1] B4:
              |                    |
              |                    |
Q2: [0]    -  4       7     11  -     [1] Q4:
              |                    |
              |                    |
B2: [1]    -  5       1     10  -     [1] Q3:
              |                    |
              |                    |
A2: [0]    -  6              9  -     [1] B3:
              |                    |
              |                    |
GND [0]    -  7              8  -     [0] A3:
              |                    |
              -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{11: 1, 10: 1, 3: 0, 4: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
              ------------
              |                    |
A1: [0]    -  1             14  -     [1] VCC
              |                    |
              |                    |
B1: [0]    -  2       4     13  -     [0] A4:
              |                    |
              |                    |
Q1: [0]    -  3       0     12  -     [1] B4:
              |                    |
              |                    |
Q2: [1]    -  4       7     11  -     [1] Q4:
              |                    |
              |                    |
```

```
B2: [1]     -  5       1      10  -     [1] Q3:
               |                     |
               |                     |
A2: [0]     -  6               9   -     [1] B3:
               |                     |
               |                     |
GND [0]     -  7               8   -     [0] A3:
               |                     |
               -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                     |
 A1: [0]     -  1             14  -     [1] VCC
               |                     |
               |                     |
 B1: [0]     -  2       4     13  -     [0] A4:
               |                     |
               |                     |
 Q1: [0]     -  3       0     12  -     [1] B4:
               |                     |
               |                     |
 Q2: [1]     -  4       7     11  -     [1] Q4:
               |                     |
               |                     |
 B2: [1]     -  5       1     10  -     [1] Q3:
               |                     |
               |                     |
 A2: [0]     -  6               9   -     [1] B3:
               |                     |
               |                     |
 GND [0]     -  7               8   -     [0] A3:
               |                     |
               -------------
{11: 1, 10: 1, 3: 0, 4: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(3, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 4072

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 4072:

ic = IC_4072()

print(ic.__doc__)
```

```
Dual 4 input OR gate
Pin_1 = OR(Pin_2, Pin_3, Pin_4, Pin_5)
Pin_13 = OR(Pin_9, Pin_10, Pin_11, Pin_12)
```

```
# The Pin configuration is:

inp = {2: 0, 3: 1, 4: 0, 5: 1, 7: 0, 9: 1, 10: 1, 11: 1, 12: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0}) -- \n

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
              ------------
              |          |
Q1: [0]    -  1       14  -    [1] VCC
              |          |
              |          |
A1: [0]    -  2     4 13  -    [0] Q2:
              |          |
              |          |
B1: [1]    -  3     0 12  -    [1] A2:
              |          |
              |          |
C1: [0]    -  4     7 11  -    [1] B2:
              |          |
              |          |
D1: [1]    -  5     2 10  -    [1] C2:
              |          |
              |          |
    [Z]    -  6       9  -    [1] D2:
              |          |
              |          |
GND [0]    -  7       8  -    [Z]
              |          |
              ------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to
```

```
print (ic.run())
```

```
{1: 1, 13: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
              ------------
              |                     |
Q1: [1]    -  1             14  -    [1] VCC
              |                     |
              |                     |
A1: [0]    -  2       4     13  -    [1] Q2:
              |                     |
              |                     |
B1: [1]    -  3       0     12  -    [1] A2:
              |                     |
              |                     |
C1: [0]    -  4       7     11  -    [1] B2:
              |                     |
              |                     |
D1: [1]    -  5       2     10  -    [1] C2:
              |                     |
              |                     |
    [Z]    -  6              9  -    [1] D2:
              |                     |
              |                     |
GND [0]    -  7              8  -    [Z]
              |                     |
              -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
              ------------
              |                     |
 Q1: [1]    -  1             14  -    [1] VCC
              |                     |
              |                     |
 A1: [0]    -  2       4     13  -    [1] Q2:
              |                     |
              |                     |
 B1: [1]    -  3       0     12  -    [1] A2:
              |                     |
              |                     |
 C1: [0]    -  4       7     11  -    [1] B2:
```

```
               |                     |
               |                     |
 D1: [1]    -  5      2      10   -     [1] C2:
               |                     |
               |                     |
     [Z]    -  6              9   -     [1] D2:
               |                     |
               |                     |
 GND [0]    -  7              8   -     [Z]
               |                     |
              -------------
{1: 1, 13: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(13, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 4073

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 4073:

ic = IC_4073()

print(ic.__doc__)
```

```
Triple 3 input AND gate
Pin_6 = AND(Pin_3, Pin_4, Pin_5)
Pin_9 = AND(Pin_1, Pin_2, Pin_8)
Pin_10 = AND(Pin_11, Pin_12, Pin_13)
```

```
# The Pin configuration is:

inp = {2: 0, 3: 1, 4: 0, 5: 1, 7: 0, 9: 1, 10: 1, 11: 1, 12: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0}) -- \n

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
              ------------
              |                       |
C2: [0]    -  1             14   -    [1] VCC
              |                       |
              |                       |
B2: [0]    -  2       4     13   -    [0] C3:
              |                       |
              |                       |
C1: [1]    -  3       0     12   -    [1] B3:
              |                       |
              |                       |
B1: [0]    -  4       7     11   -    [1] A3:
              |                       |
              |                       |
A1: [1]    -  5       3     10   -    [1] Q3:
              |                       |
              |                       |
Q1: [0]    -  6              9   -    [1] Q2:
              |                       |
              |                       |
GND [0]    -  7              8   -    [0] A2:
              |                       |
              -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{9: 0, 10: 0, 6: 0}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
              ------------
              |                       |
C2: [0]    -  1             14   -    [1] VCC
              |                       |
              |                       |
B2: [0]    -  2       4     13   -    [0] C3:
              |                       |
              |                       |
C1: [1]    -  3       0     12   -    [1] B3:
              |                       |
              |                       |
B1: [0]    -  4       7     11   -    [1] A3:
              |                       |
              |                       |
A1: [1]    -  5       3     10   -    [0] Q3:
              |                       |
              |                       |
Q1: [0]    -  6              9   -    [0] Q2:
              |                       |
              |                       |
```

```
GND [0]     -  7              8  -     [0] A2:
               |                   |
               -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
                ------------
                |                   |
 C2: [0]     -  1             14  -     [1] VCC
                |                   |
                |                   |
 B2: [0]     -  2       4     13  -     [0] C3:
                |                   |
                |                   |
 C1: [1]     -  3       0     12  -     [1] B3:
                |                   |
                |                   |
 B1: [0]     -  4       7     11  -     [1] A3:
                |                   |
                |                   |
 A1: [1]     -  5       3     10  -     [0] Q3:
                |                   |
                |                   |
 Q1: [0]     -  6              9  -     [0] Q2:
                |                   |
                |                   |
 GND [0]     -  7              8  -     [0] A2:
                |                   |
                -------------
{9: 0, 10: 0, 6: 0}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(9, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 4075

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 4075:

ic = IC_4075()
```

```
print(ic.__doc__)
```

```
Triple 3 input OR gate
Pin_6 = OR(Pin_3, Pin_4, Pin_5)
Pin_9 = OR(Pin_1, Pin_2, Pin_8)
Pin_10 = OR(Pin_11, Pin_12, Pin_13)
```

```
# The Pin configuration is:

inp = {1: 1, 2: 1, 3: 0, 4: 0, 5: 0, 7: 0, 8: 1, 11: 0, 12: 1, 13: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0}) -- \n

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
              ------------
              |                    |
C2: [1]    -  1             14  -     [1] VCC
              |                    |
              |                    |
B2: [1]    -  2       4     13  -     [1] C3:
              |                    |
              |                    |
C1: [0]    -  3       0     12  -     [1] B3:
              |                    |
              |                    |
B1: [0]    -  4       7     11  -     [0] A3:
              |                    |
              |                    |
A1: [0]    -  5       5     10  -     [0] Q3:
              |                    |
              |                    |
Q1: [0]    -  6              9  -     [0] Q2:
              |                    |
              |                    |
GND [0]    -  7              8  -     [1] A2:
              |                    |
              -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{9: 1, 10: 1, 6: 0}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n
```

```
ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
              ------------
              |                    |
C2: [1]    -  1             14  -     [1] VCC
              |                    |
              |                    |
B2: [1]    -  2       4     13  -     [1] C3:
              |                    |
              |                    |
C1: [0]    -  3       0     12  -     [1] B3:
              |                    |
              |                    |
B1: [0]    -  4       7     11  -     [0] A3:
              |                    |
              |                    |
A1: [0]    -  5       5     10  -     [1] Q3:
              |                    |
              |                    |
Q1: [0]    -  6              9  -     [1] Q2:
              |                    |
              |                    |
GND [0]    -  7              8  -     [1] A2:
              |                    |
              -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
              ------------
              |                    |
 C2: [1]    -  1             14  -     [1] VCC
              |                    |
              |                    |
 B2: [1]    -  2       4     13  -     [1] C3:
              |                    |
              |                    |
 C1: [0]    -  3       0     12  -     [1] B3:
              |                    |
              |                    |
 B1: [0]    -  4       7     11  -     [0] A3:
              |                    |
              |                    |
 A1: [0]    -  5       5     10  -     [1] Q3:
              |                    |
              |                    |
 Q1: [0]    -  6              9  -     [1] Q2:
```

```
              |                    |
              |                    |
 GND [0]    -  7             8   -     [1] A2:
              |                    |
              -------------
{9: 1, 10: 1, 6: 0}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(9, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 4077

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 4077:

ic = IC_4077()

print(ic.__doc__)
```

```
Quad 2 input XNOR gate
Pin_3 = XNOR(Pin_1, Pin_2)
Pin_4 = XNOR(Pin_5, Pin_6)
Pin_10 = XNOR(Pin_8, Pin_9)
Pin_11 = XNOR(Pin_12, Pin_13)
```

```
# The Pin configuration is:

inp = {1: 0, 2: 0, 5: 0, 6: 1, 7: 0, 8: 1, 9: 0, 12: 1, 13: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0}) -- \n

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
              ------------
              |                    |
A1: [0]    -  1             14  -     [1] VCC
              |                    |
              |                    |
B1: [0]    -  2       4     13  -     [1] A4:
```

```
              |                     |
              |                     |
Q1: [0]     -  3       0      12  -     [1] B4:
              |                     |
              |                     |
Q2: [0]     -  4       7      11  -     [0] Q4:
              |                     |
              |                     |
B2: [0]     -  5       7      10  -     [0] Q3:
              |                     |
              |                     |
A2: [1]     -  6               9  -     [0] B3:
              |                     |
              |                     |
GND [0]     -  7               8  -     [1] A3:
              |                     |
              --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{11: 1, 10: 0, 3: 1, 4: 0}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
              ------------
              |                     |
A1: [0]     -  1             14  -     [1] VCC
              |                     |
              |                     |
B1: [0]     -  2       4     13  -     [1] A4:
              |                     |
              |                     |
Q1: [1]     -  3       0     12  -     [1] B4:
              |                     |
              |                     |
Q2: [0]     -  4       7     11  -     [1] Q4:
              |                     |
              |                     |
B2: [0]     -  5       7     10  -     [0] Q3:
              |                     |
              |                     |
A2: [1]     -  6               9  -     [0] B3:
              |                     |
              |                     |
GND [0]     -  7               8  -     [1] A3:
              |                     |
              --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                    |
 A1: [0]    -  1             14  -     [1] VCC
               |                    |
               |                    |
 B1: [0]    -  2       4     13  -     [1] A4:
               |                    |
               |                    |
 Q1: [1]    -  3       0     12  -     [1] B4:
               |                    |
               |                    |
 Q2: [0]    -  4       7     11  -     [1] Q4:
               |                    |
               |                    |
 B2: [0]    -  5       7     10  -     [0] Q3:
               |                    |
               |                    |
 A2: [1]    -  6              9  -     [0] B3:
               |                    |
               |                    |
 GND [0]    -  7              8  -     [1] A3:
               |                    |
               -------------
{11: 1, 10: 0, 3: 1, 4: 0}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(11, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 4078

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 4078:

ic = IC_4078()

print(ic.__doc__)
```

```
8 input NOR gate
Pin_13 = NOR(Pin_2, Pin_3, Pin_4, Pin_5, Pin_9, Pin_10, Pin_11, Pin_12)
```

```
# The Pin configuration is:

#inp = {2: 1, 3: 1, 4: 0, 5: 1, 7: 0, 9: 1, 10: 0, 11: 1, 12: 1, 14: 1}
inp = {2: 0, 3: 0, 4: 0, 5: 0, 7: 0, 9: 0, 10: 0, 11: 0, 12: 0, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0}) -- \n

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
              ------------
              |                     |
    [Z]     -  1             14   -     [1] VCC
              |                     |
              |                     |
Inp [0]     -  2       4     13   -     [0] Out
              |                     |
              |                     |
Inp [0]     -  3       0     12   -     [0] Inp
              |                     |
              |                     |
Inp [0]     -  4       7     11   -     [0] Inp
              |                     |
              |                     |
Inp [0]     -  5       8     10   -     [0] Inp
              |                     |
              |                     |
    [Z]     -  6              9   -     [0] Inp
              |                     |
              |                     |
GND [0]     -  7              8   -     [Z]
              |                     |
              -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{13: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration
```

```
ic.drawIC()
```

```
              ------------
              |                    |
    [Z]     -  1             14  -     [1] VCC
              |                    |
              |                    |
Inp [0]     -  2       4     13  -     [1] Out
              |                    |
              |                    |
Inp [0]     -  3       0     12  -     [0] Inp
              |                    |
              |                    |
Inp [0]     -  4       7     11  -     [0] Inp
              |                    |
              |                    |
Inp [0]     -  5       8     10  -     [0] Inp
              |                    |
              |                    |
    [Z]     -  6              9  -     [0] Inp
              |                    |
              |                    |
GND [0]     -  7              8  -     [Z]
              |                    |
              -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
              ------------
              |                    |
     [Z]     -  1             14  -     [1] VCC
              |                    |
              |                    |
 Inp [0]     -  2       4     13  -     [1] Out
              |                    |
              |                    |
 Inp [0]     -  3       0     12  -     [0] Inp
              |                    |
              |                    |
 Inp [0]     -  4       7     11  -     [0] Inp
              |                    |
              |                    |
 Inp [0]     -  5       8     10  -     [0] Inp
              |                    |
              |                    |
     [Z]     -  6              9  -     [0] Inp
              |                    |
              |                    |
 GND [0]     -  7              8  -     [Z]
              |                    |
              -------------
```

```
{13: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output -- ic.setOutput(8, c)
ic.setOutput(13, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 4081

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 4081:

ic = IC_4081()

print(ic.__doc__)
```

```
Quad 2 input AND gate
Pin_3 = AND(Pin_1, Pin_2)
Pin_4 = AND(Pin_5, Pin_6)
Pin_10 = AND(Pin_8, Pin_9)
Pin_11 = AND(Pin_12, Pin_13)
```

```
# The Pin configuration is:

inp = {1: 0, 2: 0, 5: 0, 6: 1, 7: 0, 8: 1, 9: 0, 12: 1, 13: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0}) -- \n

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
              ------------
              |                     |
A1: [0]     -  1             14   -     [1] VCC
              |                     |
              |                     |
B1: [0]     -  2       4     13   -     [1] A4:
              |                     |
              |                     |
Q1: [0]     -  3       0     12   -     [1] B4:
              |                     |
              |                     |
```

```
Q2: [0]    -  4       8      11  -    [0] Q4:
              |                      |
              |                      |
B2: [0]    -  5       1      10  -    [0] Q3:
              |                      |
              |                      |
A2: [1]    -  6              9  -    [0] B3:
              |                      |
              |                      |
GND [0]    -  7              8  -    [1] A3:
              |                      |
              -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{11: 1, 10: 0, 3: 0, 4: 0}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
              ------------
              |                      |
A1: [0]    -  1             14  -    [1] VCC
              |                      |
              |                      |
B1: [0]    -  2       4     13  -    [1] A4:
              |                      |
              |                      |
Q1: [0]    -  3       0     12  -    [1] B4:
              |                      |
              |                      |
Q2: [0]    -  4       8     11  -    [1] Q4:
              |                      |
              |                      |
B2: [0]    -  5       1     10  -    [0] Q3:
              |                      |
              |                      |
A2: [1]    -  6              9  -    [0] B3:
              |                      |
              |                      |
GND [0]    -  7              8  -    [1] A3:
              |                      |
              -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
# Run the IC

print (ic.run())
```

```
              ------------
              |                    |
 A1: [0]    -  1             14  -     [1] VCC
              |                    |
              |                    |
 B1: [0]    -  2       4     13  -     [1] A4:
              |                    |
              |                    |
 Q1: [0]    -  3       0     12  -     [1] B4:
              |                    |
              |                    |
 Q2: [0]    -  4       8     11  -     [1] Q4:
              |                    |
              |                    |
 B2: [0]    -  5       1     10  -     [0] Q3:
              |                    |
              |                    |
 A2: [1]    -  6              9  -     [0] B3:
              |                    |
              |                    |
 GND [0]    -  7              8  -     [1] A3:
              |                    |
              -------------
{11: 1, 10: 0, 3: 0, 4: 0}
```

```
# Connector Outputs
c = Connector()

# Set the output -- ic.setOutput(8, c)
ic.setOutput(11, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 4082

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 4082:

ic = IC_4082()

print(ic.__doc__)
```

```
Dual 4 input AND gate
Pin_1 = AND(Pin_2, Pin_3, Pin_4, Pin_5)
Pin_13 = AND(Pin_9, Pin_10, Pin_11, Pin_12)
```

```
# The Pin configuration is:

inp = {2: 0, 3: 1, 4: 0, 5: 1, 7: 0, 9: 1, 10: 1, 11: 1, 12: 1, 14: 1}
```

```
# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0}) -- \n

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
              ------------
              |                      |
Q1: [0]    -  1             14   -    [1] VCC
              |                      |
              |                      |
A1: [0]    -  2       4     13   -    [0] Q2:
              |                      |
              |                      |
B1: [1]    -  3       0     12   -    [1] A2:
              |                      |
              |                      |
C1: [0]    -  4       8     11   -    [1] B2:
              |                      |
              |                      |
D1: [1]    -  5       2     10   -    [1] C2:
              |                      |
              |                      |
    [Z]    -  6              9   -    [1] D2:
              |                      |
              |                      |
GND [0]    -  7              8   -    [Z]
              |                      |
              -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{1: 0, 13: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
              ------------
              |                      |
Q1: [0]    -  1             14   -    [1] VCC
              |                      |
              |                      |
```

```
A1: [0]    -  2      4     13  -    [1] Q2:
              |                  |
              |                  |
B1: [1]    -  3      0     12  -    [1] A2:
              |                  |
              |                  |
C1: [0]    -  4      8     11  -    [1] B2:
              |                  |
              |                  |
D1: [1]    -  5      2     10  -    [1] C2:
              |                  |
              |                  |
    [Z]    -  6            9   -    [1] D2:
              |                  |
              |                  |
GND [0]    -  7            8   -    [Z]
              |                  |
              -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
              ------------
              |                  |
 Q1: [0]    -  1            14  -    [1] VCC
              |                  |
              |                  |
 A1: [0]    -  2      4     13  -    [1] Q2:
              |                  |
              |                  |
 B1: [1]    -  3      0     12  -    [1] A2:
              |                  |
              |                  |
 C1: [0]    -  4      8     11  -    [1] B2:
              |                  |
              |                  |
 D1: [1]    -  5      2     10  -    [1] C2:
              |                  |
              |                  |
     [Z]    -  6            9   -    [1] D2:
              |                  |
              |                  |
 GND [0]    -  7            8   -    [Z]
              |                  |
              -------------
{1: 0, 13: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(13, c)
```

```
print(c)
```

```
Connector; State: 1
```

Series 7400 ICs

Usage of IC 7400

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7400:

ic = IC_7400()

print(ic.__doc__)
```

```
This is a QUAD 2 INPUT NAND gate IC
Pin Configuration:

Pin Number      Description
    1   A Input Gate 1
    2   B Input Gate 1
    3   Y Output Gate 1
    4   A Input Gate 2
    5   B Input Gate 2
    6   Y Output Gate 2
    7   Ground
    8   Y Output Gate 3
    9   B Input Gate 3
    10  A Input Gate 3
    11  Y Output Gate 4
    12  B Input Gate 4
    13  A Input Gate 4
    14  Positive Supply

This class needs 14 parameters. Each parameter being the pin value. The input has␣
↪to be defined as a dictionary
with pin number as the key and its value being either 1 or 0

To initialise the ic 7400:
    1. set pin 7:0
    2. set pin 14:1

How to use:

    >>> ic = IC_7400()
    >>> pin_config = {1: 1, 2: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 12: 0, 13: 0, 14:␣
↪1}
    >>> ic.setIC(pin_cofig)
    >>> ic.drawIC()
    >>> ic.run()
    >>> ic.setIC(ic.run())
    >>> ic.drawIC()

Methods:
    pins = [None,0,0,None,0,0,None,0,None,0,0,None,0,0,0]
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 12: 0, 13: 0, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14   -     [1]
          |                     |
          |                     |
[0]     -  2       7     13   -     [0]
          |                     |
          |                     |
[Z]     -  3       4     12   -     [0]
          |                     |
          |                     |
[0]     -  4       0     11   -     [Z]
          |                     |
          |                     |
[0]     -  5       0     10   -     [1]
          |                     |
          |                     |
[Z]     -  6              9   -     [1]
          |                     |
          |                     |
[0]     -  7              8   -     [Z]
          |                     |
          -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 0, 11: 1, 3: 1, 6: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                      |
[1]     -  1             14   -     [1]
          |                      |
          |                      |
[0]     -  2       7     13   -     [0]
          |                      |
          |                      |
[1]     -  3       4     12   -     [0]
          |                      |
          |                      |
[0]     -  4       0     11   -     [1]
          |                      |
          |                      |
[0]     -  5       0     10   -     [1]
          |                      |
          |                      |
[1]     -  6              9   -     [1]
          |                      |
          |                      |
[0]     -  7              8   -     [0]
          |                      |
          -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                      |
     [1]     -  1             14   -     [1]
               |                      |
               |                      |
     [0]     -  2       7     13   -     [0]
               |                      |
               |                      |
     [1]     -  3       4     12   -     [0]
               |                      |
               |                      |
     [0]     -  4       0     11   -     [1]
               |                      |
               |                      |
     [0]     -  5       0     10   -     [1]
               |                      |
               |                      |
     [1]     -  6              9   -     [1]
               |                      |
               |                      |
     [0]     -  7              8   -     [0]
               |                      |
               -------------
{8: 0, 11: 1, 3: 1, 6: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 7401

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7401:

ic = IC_7401()

print(ic.__doc__)
```

```
This is a Quad 2-input open-collector NAND gate IC
```

```
# The Pin configuration is:

inp = {2: 0, 3: 0, 5: 0, 6: 1, 7: 0, 8: 1, 9: 1, 11: 1, 12: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                    |
[Z]     -  1             14  -     [1]
          |                    |
          |                    |
[0]     -  2       7     13  -     [Z]
          |                    |
          |                    |
[0]     -  3       4     12  -     [1]
          |                    |
          |                    |
[Z]     -  4       0     11  -     [1]
          |                    |
          |                    |
[0]     -  5       1     10  -     [Z]
          |                    |
          |                    |
[1]     -  6              9  -     [1]
```

```
          |                    |
          |                    |
[0]     -  7              8   -     [1]
          |                    |
          --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{1: 1, 10: 0, 4: 1, 13: 0}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                    |
[1]     -  1              14  -     [1]
          |                    |
          |                    |
[0]     -  2       7      13  -     [0]
          |                    |
          |                    |
[0]     -  3       4      12  -     [1]
          |                    |
          |                    |
[1]     -  4       0      11  -     [1]
          |                    |
          |                    |
[0]     -  5       1      10  -     [0]
          |                    |
          |                    |
[1]     -  6               9   -     [1]
          |                    |
          |                    |
[0]     -  7               8   -     [1]
          |                    |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
                ------------
                |                    |
     [1]    -   1             14   -     [1]
                |                    |
                |                    |
     [0]    -   2       7     13   -     [0]
                |                    |
                |                    |
     [0]    -   3       4     12   -     [1]
                |                    |
                |                    |
     [1]    -   4       0     11   -     [1]
                |                    |
                |                    |
     [0]    -   5       1     10   -     [0]
                |                    |
                |                    |
     [1]    -   6              9   -     [1]
                |                    |
                |                    |
     [0]    -   7              8   -     [1]
                |                    |
                -------------
{1: 1, 10: 0, 4: 1, 13: 0}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(1, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 7402

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7402:

ic = IC_7402()

print(ic.__doc__)
```

```
This is a Quad 2-input NOR gate IC

Pin Configuration:

Pin Number      Description
    1   Y Output Gate 1
    2   A Input Gate 1
    3   B Input Gate 1
    4   Y Output Gate 2
    5   A Input Gate 2
    6   B Input Gate 2
    7   Ground
    8   A Input Gate 3
```

```
    9   B Input Gate 3
    10  Y Output Gate 3
    11  A Input Gate 4
    12  B Input Gate 4
    13  Y Output Gate 4
    14  Positive Supply

This class needs 14 parameters. Each parameter being the pin value. The input has␣
↪to be defined as a dictionary
with pin number as the key and its value being either 1 or 0

To initialise the ic 7402:
    1. set pin 7:0
    2. set pin 14:1

How to use:

    >>> ic = IC_7402()
    >>> pin_config = {2: 0, 3: 0, 5: 0, 6: 1, 7: 0, 8: 1, 9: 1, 11: 1, 12: 1, 14:␣
↪1}
    >>> ic.setIC(pin_cofig)
    >>> ic.drawIC()
    >>> ic.run()
    >>> ic.setIC(ic.run())
    >>> ic.drawIC()

Default pins:
    pins = [None,None,0,0,None,0,0,0,0,0,None,0,0,None,0]
```

```
# The Pin configuration is:

inp = {2: 0, 3: 0, 5: 0, 6: 1, 7: 0, 8: 1, 9: 1, 11: 1, 12: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                    |
[Z]     -  1             14  -     [1]
          |                    |
          |                    |
[0]     -  2       7     13  -     [Z]
          |                    |
          |                    |
[0]     -  3       4     12  -     [1]
          |                    |
          |                    |
[Z]     -  4       0     11  -     [1]
          |                    |
          |                    |
[0]     -  5       2     10  -     [Z]
```

```
          |                     |
          |                     |
[1]     -  6              9   -     [1]
          |                     |
          |                     |
[0]     -  7              8   -     [1]
          |                     |
          --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{1: 1, 10: 0, 4: 0, 13: 0}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          -------------
          |                     |
[1]     -  1             14   -     [1]
          |                     |
          |                     |
[0]     -  2       7     13   -     [0]
          |                     |
          |                     |
[0]     -  3       4     12   -     [1]
          |                     |
          |                     |
[0]     -  4       0     11   -     [1]
          |                     |
          |                     |
[0]     -  5       2     10   -     [0]
          |                     |
          |                     |
[1]     -  6              9   -     [1]
          |                     |
          |                     |
[0]     -  7              8   -     [1]
          |                     |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
                ------------
                |                   |
     [1]    -   1             14   -    [1]
                |                   |
                |                   |
     [0]    -   2       7     13   -    [0]
                |                   |
                |                   |
     [0]    -   3       4     12   -    [1]
                |                   |
                |                   |
     [0]    -   4       0     11   -    [1]
                |                   |
                |                   |
     [0]    -   5       2     10   -    [0]
                |                   |
                |                   |
     [1]    -   6              9   -    [1]
                |                   |
                |                   |
     [0]    -   7              8   -    [1]
                |                   |
                --------------
{1: 1, 10: 0, 4: 0, 13: 0}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(1, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 7403

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7403:

ic = IC_7403()

print(ic.__doc__)
```

```
This is a Quad 2-input open-collector NAND gate IC

Pin Number      Description
    1   A Input Gate 1
    2   B Input Gate 1
    3   Y Output Gate 1
    4   A Input Gate 2
    5   B Input Gate 2
    6   Y Output Gate 2
    7   Ground
    8   Y Output Gate 3
    9   B Input Gate 3
```

```
    10  A Input Gate 3
    11  Y Output Gate 4
    12  B Input Gate 4
    13  A Input Gate 4
    14  Positive Supply


This class needs 14 parameters. Each parameter being the pin value. The input has␣
↪to be defined as a dictionary
with pin number as the key and its value being either 1 or 0

To initialise the ic 7403:
    1. set pin 7:0
    2. set pin 14:1

How to use:

    >>> ic = IC_7403()
    >>> pin_config = {1: 1, 2: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 12: 0, 13: 0, 14:␣
↪1}
    >>> ic.setIC(pin_cofig)
    >>> ic.drawIC()
    >>> ic.run()
    >>> ic.setIC(ic.run())
    >>> ic.drawIC()

Default pins:
    pins = [None,0,0,None,0,0,None,0,None,0,0,None,0,0,0]
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 12: 0, 13: 0, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                    |
[1]     -  1             14  -     [1]
          |                    |
          |                    |
[0]     -  2       7     13  -     [0]
          |                    |
          |                    |
[Z]     -  3       4     12  -     [0]
          |                    |
          |                    |
[0]     -  4       0     11  -     [Z]
          |                    |
          |                    |
[0]     -  5       3     10  -     [1]
```

```
          |                    |
          |                    |
[Z]     -  6              9  -     [1]
          |                    |
          |                    |
[0]     -  7              8  -     [Z]
          |                    |
          --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 0, 11: 1, 3: 1, 6: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          -------------
          |                    |
[1]     -  1             14  -     [1]
          |                    |
          |                    |
[0]     -  2       7     13  -     [0]
          |                    |
          |                    |
[1]     -  3       4     12  -     [0]
          |                    |
          |                    |
[0]     -  4       0     11  -     [1]
          |                    |
          |                    |
[0]     -  5       3     10  -     [1]
          |                    |
          |                    |
[1]     -  6              9  -     [1]
          |                    |
          |                    |
[0]     -  7              8  -     [0]
          |                    |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
                ------------
                |                   |
     [1]    -   1             14   -    [1]
                |                   |
                |                   |
     [0]    -   2       7     13   -    [0]
                |                   |
                |                   |
     [1]    -   3       4     12   -    [0]
                |                   |
                |                   |
     [0]    -   4       0     11   -    [1]
                |                   |
                |                   |
     [0]    -   5       3     10   -    [1]
                |                   |
                |                   |
     [1]    -   6              9   -    [1]
                |                   |
                |                   |
     [0]    -   7              8   -    [0]
                |                   |
                --------------
{8: 0, 11: 1, 3: 1, 6: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 7404

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7404:

ic = IC_7404()

print(ic.__doc__)
```

```
This is a hex inverter IC

Pin Number      Description
    1   A Input Gate 1
    2   Y Output Gate 1
    3   A Input Gate 2
    4   Y Output Gate 2
    5   A Input Gate 3
    6   Y Output Gate 3
    7   Ground
    8   Y Output Gate 4
    9   A Input Gate 4
```

```
    10  Y Output Gate 5
    11  A Input Gate 5
    12  Y Output Gate 6
    13  A Input Gate 6
    14  Positive Supply

This class needs 14 parameters. Each parameter being the pin value. The input has␣
↪to be defined as a dictionary
with pin number as the key and its value being either 1 or 0

To initialise the ic 7404:
    1. set pin 7:0
    2. set pin 14:1

How to use:

    >>> ic = IC_7404()
    >>> pin_config = {1: 1, 3: 0, 5: 0, 7: 0, 9: 0, 11: 0, 13: 1, 14: 1}
    >>> ic.setIC(pin_cofig)
    >>> ic.drawIC()
    >>> ic.run()
    >>> ic.setIC(ic.run())
    >>> ic.drawIC()

Default pins:
    pins = [None,0,0,None,0,0,None,0,None,0,0,None,0,0,0]
```

```
# The Pin configuration is:

inp = {1: 1, 3: 0, 5: 0, 7: 0, 9: 0, 11: 0, 13: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                    |
[1]     -  1             14  -     [1]
          |                    |
          |                    |
[Z]     -  2       7     13  -     [1]
          |                    |
          |                    |
[0]     -  3       4     12  -     [Z]
          |                    |
          |                    |
[Z]     -  4       0     11  -     [0]
          |                    |
          |                    |
[0]     -  5       4     10  -     [Z]
          |                    |
          |                    |
```

```
[Z]     -  6              9  -     [0]
           |                    |
           |                    |
[0]     -  7              8  -     [Z]
           |                    |
           --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{2: 0, 4: 1, 6: 1, 8: 1, 10: 1, 12: 0}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
           ------------
           |                    |
[1]     -  1             14 -     [1]
           |                    |
           |                    |
[0]     -  2       7     13 -     [1]
           |                    |
           |                    |
[0]     -  3       4     12 -     [0]
           |                    |
           |                    |
[1]     -  4       0     11 -     [0]
           |                    |
           |                    |
[0]     -  5       4     10 -     [1]
           |                    |
           |                    |
[1]     -  6              9  -     [0]
           |                    |
           |                    |
[0]     -  7              8  -     [1]
           |                    |
           --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
                ------------
                |                       |
     [1]     -  1              14   -    [1]
                |                       |
                |                       |
     [0]     -  2       7      13   -    [1]
                |                       |
                |                       |
     [0]     -  3       4      12   -    [0]
                |                       |
                |                       |
     [1]     -  4       0      11   -    [0]
                |                       |
                |                       |
     [0]     -  5       4      10   -    [1]
                |                       |
                |                       |
     [1]     -  6               9   -    [0]
                |                       |
                |                       |
     [0]     -  7               8   -    [1]
                |                       |
                -------------
{2: 0, 4: 1, 6: 1, 8: 1, 10: 1, 12: 0}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 7405

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7405:

ic = IC_7405()

print(ic.__doc__)
```

```
This is hex open-collector inverter IC
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 12: 0, 13: 0, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic
```

```
ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14  -     [1]
          |                     |
          |                     |
[0]     -  2       7     13  -     [0]
          |                     |
          |                     |
[0]     -  3       4     12  -     [0]
          |                     |
          |                     |
[0]     -  4       0     11  -     [0]
          |                     |
          |                     |
[0]     -  5       5     10  -     [1]
          |                     |
          |                     |
[Z]     -  6              9  -     [1]
          |                     |
          |                     |
[0]     -  7              8  -     [Z]
          |                     |
          --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{2: 0, 4: 1, 6: 1, 8: 0, 10: 1, 12: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14  -     [1]
          |                     |
          |                     |
[0]     -  2       7     13  -     [0]
          |                     |
          |                     |
[0]     -  3       4     12  -     [1]
          |                     |
          |                     |
[1]     -  4       0     11  -     [0]
          |                     |
```

```
          |                     |
[0]     -  5       5     10   -     [1]
          |                     |
          |                     |
[1]     -  6               9   -     [1]
          |                     |
          |                     |
[0]     -  7               8   -     [0]
          |                     |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                     |
     [1]     -  1             14   -     [1]
               |                     |
               |                     |
     [0]     -  2       7     13   -     [0]
               |                     |
               |                     |
     [0]     -  3       4     12   -     [1]
               |                     |
               |                     |
     [1]     -  4       0     11   -     [0]
               |                     |
               |                     |
     [0]     -  5       5     10   -     [1]
               |                     |
               |                     |
     [1]     -  6               9   -     [1]
               |                     |
               |                     |
     [0]     -  7               8   -     [0]
               |                     |
               --------------
{2: 0, 4: 1, 6: 1, 8: 0, 10: 1, 12: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 7408

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7408:

ic = IC_7408()

print(ic.__doc__)
```

```
This is a Quad 2 input AND gate IC

Pin Number      Description
    1   A Input Gate 1
    2   B Input Gate 1
    3   Y Output Gate 1
    4   A Input Gate 2
    5   B Input Gate 2
    6   Y Output Gate 2
    7   Ground
    8   Y Output Gate 3
    9   B Input Gate 3
    10  A Input Gate 3
    11  Y Output Gate 4
    12  B Input Gate 4
    13  A Input Gate 4
    14  Positive Supply

This class needs 14 parameters. Each parameter being the pin value. The input has␣
↪to be defined as a dictionary
with pin number as the key and its value being either 1 or 0

To initialise the ic 7408:
    1. set pin 7:0
    2. set pin 14:1

How to use:

    >>> ic = IC_7408()
    >>> pin_config = {1: 1, 2: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 12: 0, 13: 0, 14:␣
↪1}
    >>> ic.setIC(pin_cofig)
    >>> ic.drawIC()
    >>> ic.run()
    >>> ic.setIC(ic.run())
    >>> ic.drawIC()

Default pins:
    pins = [None,0,0,None,0,0,None,0,None,0,0,None,0,0,0]
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 12: 0, 13: 0, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic
```

```
ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14  -     [1]
          |                     |
          |                     |
[0]     -  2       7     13  -     [0]
          |                     |
          |                     |
[Z]     -  3       4     12  -     [0]
          |                     |
          |                     |
[0]     -  4       0     11  -     [Z]
          |                     |
          |                     |
[0]     -  5       8     10  -     [1]
          |                     |
          |                     |
[Z]     -  6              9  -     [1]
          |                     |
          |                     |
[0]     -  7              8  -     [Z]
          |                     |
          --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 1, 11: 0, 3: 0, 6: 0}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14  -     [1]
          |                     |
          |                     |
[0]     -  2       7     13  -     [0]
          |                     |
          |                     |
[0]     -  3       4     12  -     [0]
          |                     |
          |                     |
[0]     -  4       0     11  -     [0]
          |                     |
```

```
          |                     |
[0]    -  5      8     10   -    [1]
          |                     |
          |                     |
[0]    -  6             9   -    [1]
          |                     |
          |                     |
[0]    -  7             8   -    [1]
          |                     |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                     |
     [1]    -  1             14  -    [1]
               |                     |
               |                     |
     [0]    -  2      7      13  -    [0]
               |                     |
               |                     |
     [0]    -  3      4      12  -    [0]
               |                     |
               |                     |
     [0]    -  4      0      11  -    [0]
               |                     |
               |                     |
     [0]    -  5      8      10  -    [1]
               |                     |
               |                     |
     [0]    -  6             9   -    [1]
               |                     |
               |                     |
     [0]    -  7             8   -    [1]
               |                     |
               --------------
{8: 1, 11: 0, 3: 0, 6: 0}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 7410

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7410:

ic = IC_7410()

print(ic.__doc__)
```

```
This is a Triple 3 input NAND gate IC

Pin Number      Description
    1   A Input Gate 1
    2   B Input Gate 1
    3   A Input Gate 2
    4   B Input Gate 2
    5   C Input gate 2
    6   Y Output Gate 2
    7   Ground
    8   Y Output Gate 3
    9   A Input Case 3
    10  B Input Case 3
    11  C Input Case 3
    12  Y Output Gate 1
    13  C Input Gate 1
    14  Positive Supply


This class needs 14 parameters. Each parameter being the pin value. The input has␣
↪to be defined as a dictionary
with pin number as the key and its value being either 1 or 0

To initialise the ic 7410:
    1. set pin 7:0
    2. set pin 14:1

How to use:

    >>> ic = IC_7410()
    >>> pin_config = {1: 1, 2: 0, 3: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 11: 1, 13:␣
↪0, 14: 1}
    >>> ic.setIC(pin_cofig)
    >>> ic.drawIC()
    >>> ic.run()
    >>> ic.setIC(ic.run())
    >>> ic.drawIC()

Default pins:
    pins = [None,0,0,0,0,0,None,0,None,0,0,0,None,0,0]
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 3: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 11: 1, 13: 0, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})
```

```
# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14  -     [1]
          |                     |
          |                     |
[0]     -  2       7     13  -     [0]
          |                     |
          |                     |
[0]     -  3       4     12  -     [Z]
          |                     |
          |                     |
[0]     -  4       1     11  -     [1]
          |                     |
          |                     |
[0]     -  5       0     10  -     [1]
          |                     |
          |                     |
[Z]     -  6              9  -     [1]
          |                     |
          |                     |
[0]     -  7              8  -     [Z]
          |                     |
          -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 0, 12: 1, 6: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14  -     [1]
          |                     |
          |                     |
[0]     -  2       7     13  -     [0]
          |                     |
          |                     |
[0]     -  3       4     12  -     [1]
          |                     |
          |                     |
[0]     -  4       1     11  -     [1]
```

```
          |                     |
          |                     |
[0]    -  5       0     10   -     [1]
          |                     |
          |                     |
[1]    -  6              9   -     [1]
          |                     |
          |                     |
[0]    -  7              8   -     [0]
          |                     |
          -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                     |
     [1]    -  1             14   -     [1]
               |                     |
               |                     |
     [0]    -  2       7     13   -     [0]
               |                     |
               |                     |
     [0]    -  3       4     12   -     [1]
               |                     |
               |                     |
     [0]    -  4       1     11   -     [1]
               |                     |
               |                     |
     [0]    -  5       0     10   -     [1]
               |                     |
               |                     |
     [1]    -  6              9   -     [1]
               |                     |
               |                     |
     [0]    -  7              8   -     [0]
               |                     |
               -------------
{8: 0, 12: 1, 6: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 7411

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7411:

ic = IC_7411()

print(ic.__doc__)
```

```
This is a Triple 3 input AND gate IC

Pin Number      Description
    1   A Input Gate 1
    2   B Input Gate 1
    3   A Input Gate 2
    4   B Input Gate 2
    5   C Input gate 2
    6   Y Output Gate 2
    7   Ground
    8   Y Output Gate 3
    9   A Input Case 3
    10  B Input Case 3
    11  C Input Case 3
    12  Y Output Gate 1
    13  C Input Gate 1
    14  Positive Supply


This class needs 14 parameters. Each parameter being the pin value. The input has␣
↪to be defined as a dictionary
with pin number as the key and its value being either 1 or 0

To initialise the ic 7411:
    1. set pin 7:0
    2. set pin 14:1

How to use:

    >>> ic = IC_7411()
    >>> pin_config = {1: 1, 2: 0, 3: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 11: 1, 13:␣
↪0, 14: 1}
    >>> ic.setIC(pin_cofig)
    >>> ic.drawIC()
    >>> ic.run()
    >>> ic.setIC(ic.run())
    >>> ic.drawIC()

Default pins:
    pins = [None,0,0,0,0,0,None,0,None,0,0,0,None,0,0]
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 3: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 11: 1, 13: 0, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})
```

```
# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14  -     [1]
          |                     |
          |                     |
[0]     -  2       7     13  -     [0]
          |                     |
          |                     |
[0]     -  3       4     12  -     [Z]
          |                     |
          |                     |
[0]     -  4       1     11  -     [1]
          |                     |
          |                     |
[0]     -  5       1     10  -     [1]
          |                     |
          |                     |
[Z]     -  6              9  -     [1]
          |                     |
          |                     |
[0]     -  7              8  -     [Z]
          |                     |
          -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 1, 12: 0, 6: 0}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14  -     [1]
          |                     |
          |                     |
[0]     -  2       7     13  -     [0]
          |                     |
          |                     |
[0]     -  3       4     12  -     [0]
          |                     |
          |                     |
[0]     -  4       1     11  -     [1]
```

```
          |                     |
          |                     |
[0]    -  5       1      10  -     [1]
          |                     |
          |                     |
[0]    -  6               9  -     [1]
          |                     |
          |                     |
[0]    -  7               8  -     [1]
          |                     |
          -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                     |
     [1]    -  1             14  -     [1]
               |                     |
               |                     |
     [0]    -  2       7     13  -     [0]
               |                     |
               |                     |
     [0]    -  3       4     12  -     [0]
               |                     |
               |                     |
     [0]    -  4       1     11  -     [1]
               |                     |
               |                     |
     [0]    -  5       1     10  -     [1]
               |                     |
               |                     |
     [0]    -  6               9  -     [1]
               |                     |
               |                     |
     [0]    -  7               8  -     [1]
               |                     |
               -------------
{8: 1, 12: 0, 6: 0}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 7412

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7412:

ic = IC_7412()

print(ic.__doc__)
```

```
This is a Triple 3 input NAND gate IC with open collector outputs

Pin Number      Description
    1   A Input Gate 1
    2   B Input Gate 1
    3   A Input Gate 2
    4   B Input Gate 2
    5   C Input gate 2
    6   Y Output Gate 2
    7   Ground
    8   Y Output Gate 3
    9   A Input Case 3
    10  B Input Case 3
    11  C Input Case 3
    12  Y Output Gate 1
    13  C Input Gate 1
    14  Positive Supply


This class needs 14 parameters. Each parameter being the pin value. The input has␣
↪to be defined as a dictionary
with pin number as the key and its value being either 1 or 0

To initialise the ic 7412:
    1. set pin 7:0
    2. set pin 14:1

How to use:

    >>> ic = IC_7412()
    >>> pin_config = {1: 1, 2: 0, 3: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 11: 1, 13:␣
↪0, 14: 1}
    >>> ic.setIC(pin_cofig)
    >>> ic.drawIC()
    >>> ic.run()
    >>> ic.setIC(ic.run())
    >>> ic.drawIC()

Default pins:
    pins = [None,0,0,0,0,0,None,0,None,0,0,0,None,0,0]
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 3: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 11: 1, 13: 0, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})
```

```
# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14  -     [1]
          |                     |
          |                     |
[0]     -  2       7     13  -     [0]
          |                     |
          |                     |
[0]     -  3       4     12  -     [Z]
          |                     |
          |                     |
[0]     -  4       1     11  -     [1]
          |                     |
          |                     |
[0]     -  5       2     10  -     [1]
          |                     |
          |                     |
[Z]     -  6              9  -     [1]
          |                     |
          |                     |
[0]     -  7              8  -     [Z]
          |                     |
          -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 0, 12: 1, 6: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14  -     [1]
          |                     |
          |                     |
[0]     -  2       7     13  -     [0]
          |                     |
          |                     |
[0]     -  3       4     12  -     [1]
          |                     |
          |                     |
[0]     -  4       1     11  -     [1]
```

```
          |                     |
          |                     |
[0]    -  5       2      10   -     [1]
          |                     |
          |                     |
[1]    -  6               9   -     [1]
          |                     |
          |                     |
[0]    -  7               8   -     [0]
          |                     |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                     |
     [1]    -  1             14   -     [1]
               |                     |
               |                     |
     [0]    -  2       7     13   -     [0]
               |                     |
               |                     |
     [0]    -  3       4     12   -     [1]
               |                     |
               |                     |
     [0]    -  4       1     11   -     [1]
               |                     |
               |                     |
     [0]    -  5       2     10   -     [1]
               |                     |
               |                     |
     [1]    -  6               9   -     [1]
               |                     |
               |                     |
     [0]    -  7               8   -     [0]
               |                     |
               --------------
{8: 0, 12: 1, 6: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 74138

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 74138:

ic = IC_74138()

print(ic.__doc__)
```

```
This is a 1:8 demultiplexer(3:8 decoder) with output being inverted input
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 3: 1, 4: 0, 5: 0, 6: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({16: 1, 8: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          -------------
          |                     |
[1]     -  1             16  -     [1]
          |                     |
          |                     |
[0]     -  2       7     15  -     [Z]
          |                     |
          |                     |
[1]     -  3       4     14  -     [Z]
          |                     |
          |                     |
[0]     -  4       1     13  -     [Z]
          |                     |
          |                     |
[0]     -  5       3     12  -     [Z]
          |                     |
          |                     |
[1]     -  6       8     11  -     [Z]
          |                     |
          |                     |
[Z]     -  7             10  -     [Z]
          |                     |
          |                     |
[0]     -  8              9  -     [Z]
          |                     |
          --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to
```

```
print (ic.run())
```

```
{7: 1, 9: 1, 10: 0, 11: 1, 12: 1, 13: 1, 14: 1, 15: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                    |
[1]    -  1             16  -     [1]
          |                    |
          |                    |
[0]    -  2       7     15  -     [1]
          |                    |
          |                    |
[1]    -  3       4     14  -     [1]
          |                    |
          |                    |
[0]    -  4       1     13  -     [1]
          |                    |
          |                    |
[0]    -  5       3     12  -     [1]
          |                    |
          |                    |
[1]    -  6       8     11  -     [1]
          |                    |
          |                    |
[1]    -  7             10  -     [0]
          |                    |
          |                    |
[0]    -  8              9  -     [1]
          |                    |
          -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                    |
     [1]    -  1             16  -     [1]
               |                    |
               |                    |
     [0]    -  2       7     15  -     [1]
               |                    |
               |                    |
```

```
     [1]    -  3       4      14  -    [1]
               |                     |
               |                     |
     [0]    -  4       1      13  -    [1]
               |                     |
               |                     |
     [0]    -  5       3      12  -    [1]
               |                     |
               |                     |
     [1]    -  6       8      11  -    [1]
               |                     |
               |                     |
     [1]    -  7              10  -    [0]
               |                     |
               |                     |
     [0]    -  8               9  -    [1]
               |                     |
              --------------
{7: 1, 9: 1, 10: 0, 11: 1, 12: 1, 13: 1, 14: 1, 15: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(7, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 74139

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 74139:

ic = IC_74139()

print(ic.__doc__)
```

```
This is a dual 1:4 demultiplexer(2:4 decoder) with output being inverted input
```

```
# The Pin configuration is:

inp = {1: 0, 2: 0, 3: 0, 14: 0, 13: 1, 15: 0}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({16: 1, 8: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n
```

```
ic.drawIC()
```

```
          ------------
          |                     |
[0]     -  1             16  -     [1]
          |                     |
          |                     |
[0]     -  2       7     15  -     [0]
          |                     |
          |                     |
[0]     -  3       4     14  -     [0]
          |                     |
          |                     |
[Z]     -  4       1     13  -     [1]
          |                     |
          |                     |
[Z]     -  5       3     12  -     [Z]
          |                     |
          |                     |
[Z]     -  6       9     11  -     [Z]
          |                     |
          |                     |
[Z]     -  7             10  -     [Z]
          |                     |
          |                     |
[0]     -  8              9  -     [Z]
          |                     |
          -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{4: 0, 5: 1, 6: 1, 7: 1, 9: 1, 10: 0, 11: 1, 12: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                     |
[0]     -  1             16  -     [1]
          |                     |
          |                     |
[0]     -  2       7     15  -     [0]
          |                     |
          |                     |
[0]     -  3       4     14  -     [0]
          |                     |
          |                     |
[0]     -  4       1     13  -     [1]
          |                     |
          |                     |
```

```
[1]    -  5       3      12  -    [1]
          |                    |
          |                    |
[1]    -  6       9      11  -    [1]
          |                    |
          |                    |
[1]    -  7             10  -    [0]
          |                    |
          |                    |
[0]    -  8              9  -    [1]
          |                    |
          -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                    |
     [0]    -  1             16  -    [1]
               |                    |
               |                    |
     [0]    -  2       7     15  -    [0]
               |                    |
               |                    |
     [0]    -  3       4     14  -    [0]
               |                    |
               |                    |
     [0]    -  4       1     13  -    [1]
               |                    |
               |                    |
     [1]    -  5       3     12  -    [1]
               |                    |
               |                    |
     [1]    -  6       9     11  -    [1]
               |                    |
               |                    |
     [1]    -  7             10  -    [0]
               |                    |
               |                    |
     [0]    -  8              9  -    [1]
               |                    |
               -------------
{4: 0, 5: 1, 6: 1, 7: 1, 9: 1, 10: 0, 11: 1, 12: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(9, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 7413

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7413:

ic = IC_7413()

print(ic.__doc__)
```

```
This is a dual 4 input NAND gate IC

Pin Number      Description
    1   A Input Gate 1
    2   B Input Gate 1
    3   Not Connected
    4   C Input Gate 1
    5   D Input Gate 1
    6   Y Output Gate 1
    7   Ground
    8   Y Output Gate 2
    9   A Input Gate 2
    10  B Input Gate 2
    11  Not Connected
    12  C Input Gate 2
    13  D Input Gate 2
    14  Positive Supply


This class needs 14 parameters. Each parameter being the pin value. The input has␣
↪to be defined as a dictionary
with pin number as the key and its value being either 1 or 0

To initialise the ic 7413:
    1. set pin 7:0
    2. set pin 14:1

How to use:

    >>> ic = IC_7413()
    >>> pin_config = {1: 1, 2: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 12: 1, 13: 1, 14:␣
↪1}
    >>> ic.setIC(pin_cofig)
    >>> ic.drawIC()
    >>> ic.run()
    >>> ic.setIC(ic.run())
    >>> ic.drawIC()

Default pins:
    pins = [None,0,0,0,0,0,None,0,None,0,0,0,0,0,0]
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 12: 1, 13: 1, 14: 1}

# Pin initinalization
```

```
# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14  -     [1]
          |                     |
          |                     |
[0]     -  2       7     13  -     [1]
          |                     |
          |                     |
[0]     -  3       4     12  -     [1]
          |                     |
          |                     |
[0]     -  4       1     11  -     [0]
          |                     |
          |                     |
[0]     -  5       3     10  -     [1]
          |                     |
          |                     |
[Z]     -  6              9  -     [1]
          |                     |
          |                     |
[0]     -  7              8  -     [Z]
          |                     |
          -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 0, 6: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14  -     [1]
          |                     |
          |                     |
[0]     -  2       7     13  -     [1]
          |                     |
          |                     |
```

```
[0]    -  3       4      12  -     [1]
          |                    |
          |                    |
[0]    -  4       1      11  -     [0]
          |                    |
          |                    |
[0]    -  5       3      10  -     [1]
          |                    |
          |                    |
[1]    -  6              9   -     [1]
          |                    |
          |                    |
[0]    -  7              8   -     [0]
          |                    |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                    |
     [1]    -  1             14  -     [1]
               |                    |
               |                    |
     [0]    -  2       7     13  -     [1]
               |                    |
               |                    |
     [0]    -  3       4     12  -     [1]
               |                    |
               |                    |
     [0]    -  4       1     11  -     [0]
               |                    |
               |                    |
     [0]    -  5       3     10  -     [1]
               |                    |
               |                    |
     [1]    -  6              9   -     [1]
               |                    |
               |                    |
     [0]    -  7              8   -     [0]
               |                    |
               --------------
{8: 0, 6: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 74151A

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 74151A:

ic = IC_74151A()

print(ic.__doc__)
```

```
This is 16-pin 8:1 multiplexer featuring complementary W and Y outputs
```

```
# The Pin configuration is:

inp = {
    1: 1,
    2: 0,
    4: 1,
    3: 1,
    7: 0,
    9: 0,
    10: 0,
    11: 0,
    12: 0,
    13: 0,
    14: 1,
    15: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({16: 1, 8: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                  |
[1]     -  1             16  -    [1]
          |                  |
          |                  |
[0]     -  2       7     15  -    [1]
          |                  |
          |                  |
[1]     -  3       4     14  -    [1]
          |                  |
          |                  |
[1]     -  4       1     13  -    [0]
          |                  |
          |                  |
```

```
[Z]     -  5       5      12  -     [0]
           |                     |
           |                     |
[Z]     -  6       1      11  -     [0]
           |                     |
           |                     |
[0]     -  7       A      10  -     [0]
           |                     |
           |                     |
[0]     -  8                9  -     [0]
           |                     |
           --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{5: 1, 6: 0}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
           -------------
           |                     |
[1]     -  1             16  -     [1]
           |                     |
           |                     |
[0]     -  2       7     15  -     [1]
           |                     |
           |                     |
[1]     -  3       4     14  -     [1]
           |                     |
           |                     |
[1]     -  4       1     13  -     [0]
           |                     |
           |                     |
[1]     -  5       5     12  -     [0]
           |                     |
           |                     |
[0]     -  6       1     11  -     [0]
           |                     |
           |                     |
[0]     -  7       A     10  -     [0]
           |                     |
           |                     |
[0]     -  8                9  -     [0]
           |                     |
           --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())
```

```
# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                    |
     [1]    -  1             16  -     [1]
               |                    |
               |                    |
     [0]    -  2       7     15  -     [1]
               |                    |
               |                    |
     [1]    -  3       4     14  -     [1]
               |                    |
               |                    |
     [1]    -  4       1     13  -     [0]
               |                    |
               |                    |
     [1]    -  5       5     12  -     [0]
               |                    |
               |                    |
     [0]    -  6       1     11  -     [0]
               |                    |
               |                    |
     [0]    -  7       A     10  -     [0]
               |                    |
               |                    |
     [0]    -  8              9  -     [0]
               |                    |
               --------------
{5: 1, 6: 0}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(5, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 74152

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 74152:

ic = IC_74152()

print(ic.__doc__)
```

```
This is 14-pin 8:1 multiplexer with inverted input.

```

```
Pin Number      Description
    1   D4
    2   D3
    3   D2
    4   D1
    5   D0
    6   Output W
    7   Ground
    8   select line C
    9   select line B
    10  select line A
    11  D7
    12  D6
    13     D5
    14  Positive Supply

    Selectlines = CBA and Inputlines = D0 D1 D2 D3 D4 D5 D6 D7
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 3: 1, 4: 0, 5: 1, 8: 0, 9: 0, 10: 1, 11: 1, 12: 0, 13: 0}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                    |
[1]     -  1             14  -     [1]
          |                    |
          |                    |
[0]     -  2       7     13  -     [0]
          |                    |
          |                    |
[1]     -  3       4     12  -     [0]
          |                    |
          |                    |
[0]     -  4       1     11  -     [1]
          |                    |
          |                    |
[1]     -  5       5     10  -     [1]
          |                    |
          |                    |
[Z]     -  6       2      9  -     [0]
          |                    |
          |                    |
[0]     -  7              8  -     [0]
          |                    |
          --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{6: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                      |
[1]     -  1             14   -     [1]
          |                      |
          |                      |
[0]     -  2       7     13   -     [0]
          |                      |
          |                      |
[1]     -  3       4     12   -     [0]
          |                      |
          |                      |
[0]     -  4       1     11   -     [1]
          |                      |
          |                      |
[1]     -  5       5     10   -     [1]
          |                      |
          |                      |
[1]     -  6       2      9   -     [0]
          |                      |
          |                      |
[0]     -  7              8   -     [0]
          |                      |
          -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                      |
     [1]     -  1             14   -     [1]
               |                      |
               |                      |
     [0]     -  2       7     13   -     [0]
               |                      |
               |                      |
```

```
     [1]    -  3       4      12  -    [0]
               |                   |
               |                   |
     [0]    -  4       1      11  -    [1]
               |                   |
               |                   |
     [1]    -  5       5      10  -    [1]
               |                   |
               |                   |
     [1]    -  6       2       9  -    [0]
               |                   |
               |                   |
     [0]    -  7              8  -    [0]
               |                   |
               --------------
{6: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(6, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 74153

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 74153:

ic = IC_74153()

print(ic.__doc__)
```

```
This is 16-pin dual 4:1 multiplexer with output same as the input.

    Pin Number  Description
    1   Strobe1
    2   Select line B
    3   1C3
    4   1C2
    5   1C1
    6   1C0
    7   1Y - OUTPUT1
    8   Ground
    9   2Y - OUTPUT2
    10  2C0
    11  2C1
    12  2C2
    13  2C3
    14     Select line A
    15     Strobe2
    16  Positive Supply

    Selectlines = BA ; Inputlines1 = 1C0 1C1 1C2 1C3 ; Inputlines2 = 2C0 2C1 2C2␣
↪2C3
```

```
# The Pin configuration is:

inp = {
    1: 1,
    2: 1,
    3: 1,
    4: 0,
    5: 0,
    6: 0,
    10: 0,
    11: 1,
    12: 0,
    13: 0,
    14: 0,
    15: 0}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({16: 1, 8: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             16  -     [1]
          |                     |
          |                     |
[1]     -  2       7     15  -     [0]
          |                     |
          |                     |
[1]     -  3       4     14  -     [0]
          |                     |
          |                     |
[0]     -  4       1     13  -     [0]
          |                     |
          |                     |
[0]     -  5       5     12  -     [0]
          |                     |
          |                     |
[0]     -  6       3     11  -     [1]
          |                     |
          |                     |
[Z]     -  7             10  -     [0]
          |                     |
          |                     |
[0]     -  8              9  -     [Z]
          |                     |
          --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to
```

```
print (ic.run())
```

```
{9: 0, 7: 0}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             16  -     [1]
          |                     |
          |                     |
[1]     -  2       7     15  -     [0]
          |                     |
          |                     |
[1]     -  3       4     14  -     [0]
          |                     |
          |                     |
[0]     -  4       1     13  -     [0]
          |                     |
          |                     |
[0]     -  5       5     12  -     [0]
          |                     |
          |                     |
[0]     -  6       3     11  -     [1]
          |                     |
          |                     |
[0]     -  7             10  -     [0]
          |                     |
          |                     |
[0]     -  8              9  -     [0]
          |                     |
          -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                     |
     [1]     -  1             16  -     [1]
               |                     |
               |                     |
     [1]     -  2       7     15  -     [0]
               |                     |
               |                     |
```

```
     [1]    -  3       4      14  -    [0]
               |                     |
               |                     |
     [0]    -  4       1      13  -    [0]
               |                     |
               |                     |
     [0]    -  5       5      12  -    [0]
               |                     |
               |                     |
     [0]    -  6       3      11  -    [1]
               |                     |
               |                     |
     [0]    -  7              10  -    [0]
               |                     |
               |                     |
     [0]    -  8               9  -    [0]
               |                     |
              --------------
{9: 0, 7: 0}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(9, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 74156

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 74156:

ic = IC_74156()

print(ic.__doc__)
```

```
This is a dual 1:4 demultiplexer(2:4 decoder) with one output being inverted input
while the other same as the input with open collector
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 3: 0, 13: 1, 8: 0, 16: 1, 15: 1, 14: 0}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n
```

```
ic.drawIC()
```

```
          ------------
          |                    |
[1]     -  1             16  -     [1]
          |                    |
          |                    |
[0]     -  2       7     15  -     [1]
          |                    |
          |                    |
[0]     -  3       4     14  -     [0]
          |                    |
          |                    |
[Z]     -  4       1     13  -     [1]
          |                    |
          |                    |
[Z]     -  5       5     12  -     [Z]
          |                    |
          |                    |
[Z]     -  6       6     11  -     [Z]
          |                    |
          |                    |
[0]     -  7             10  -     [Z]
          |                    |
          |                    |
[0]     -  8              9  -     [Z]
          |                    |
          --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{4: 1, 5: 1, 6: 0, 7: 1, 9: 1, 10: 1, 11: 1, 12: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                    |
[1]     -  1             16  -     [1]
          |                    |
          |                    |
[0]     -  2       7     15  -     [1]
          |                    |
          |                    |
[0]     -  3       4     14  -     [0]
          |                    |
          |                    |
[1]     -  4       1     13  -     [1]
          |                    |
          |                    |
```

```
[1]    -  5       5     12  -    [1]
          |                  |
          |                  |
[0]    -  6       6     11  -    [1]
          |                  |
          |                  |
[1]    -  7             10  -    [1]
          |                  |
          |                  |
[0]    -  8              9  -    [1]
          |                  |
          -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                  |
     [1]    -  1             16  -    [1]
               |                  |
               |                  |
     [0]    -  2       7     15  -    [1]
               |                  |
               |                  |
     [0]    -  3       4     14  -    [0]
               |                  |
               |                  |
     [1]    -  4       1     13  -    [1]
               |                  |
               |                  |
     [1]    -  5       5     12  -    [1]
               |                  |
               |                  |
     [0]    -  6       6     11  -    [1]
               |                  |
               |                  |
     [1]    -  7             10  -    [1]
               |                  |
               |                  |
     [0]    -  8              9  -    [1]
               |                  |
               -------------
{4: 1, 5: 1, 6: 0, 7: 1, 9: 1, 10: 1, 11: 1, 12: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(6, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 7415

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7415:

ic = IC_7415()

print(ic.__doc__)
```

```
This is a Triple 3 input AND gate IC with open collector outputs

Pin Number      Description
    1   A Input Gate 1
    2   B Input Gate 1
    3   A Input Gate 2
    4   B Input Gate 2
    5   C Input Gate 2
    6   Y Output Gate 2
    7   Ground
    8   Y Output Gate 3
    9   A Input Gate 3
    10  B Input Gate 3
    11  C Input Gate 3
    12  Y Output Gate 1
    13  C Input Gate 1
    14  Positive Supply


This class needs 14 parameters. Each parameter being the pin value. The input has␣
↪to be defined as a dictionary
with pin number as the key and its value being either 1 or 0

To initialise the ic 7415:
    1. set pin 7:0
    2. set pin 14:1

How to use:

    >>> ic = IC_7415()
    >>> pin_config = {1:1, 2:0, 3:0, 4:0, 5:0, 7:0, 9:1, 10:1, 11:1, 13:0, 14:1}
    >>> ic.setIC(pin_cofig)
    >>> ic.drawIC()
    >>> ic.run()
    >>> ic.setIC(ic.run())
    >>> ic.drawIC()

Default pins:
    pins = [None,0,0,0,0,0,None,0,None,0,0,0,None,0,0]
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 3: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 11: 1, 13: 0, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})
```

```
ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14  -     [1]
          |                     |
          |                     |
[0]     -  2       7     13  -     [0]
          |                     |
          |                     |
[0]     -  3       4     12  -     [Z]
          |                     |
          |                     |
[0]     -  4       1     11  -     [1]
          |                     |
          |                     |
[0]     -  5       5     10  -     [1]
          |                     |
          |                     |
[Z]     -  6              9  -     [1]
          |                     |
          |                     |
[0]     -  7              8  -     [Z]
          |                     |
          -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 1, 12: 0, 6: 0}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14  -     [1]
          |                     |
          |                     |
[0]     -  2       7     13  -     [0]
          |                     |
          |                     |
[0]     -  3       4     12  -     [0]
```

```
          |                     |
          |                     |
[0]    -  4       1      11   -     [1]
          |                     |
          |                     |
[0]    -  5       5      10   -     [1]
          |                     |
          |                     |
[0]    -  6              9   -     [1]
          |                     |
          |                     |
[0]    -  7              8   -     [1]
          |                     |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                     |
     [1]    -  1             14   -     [1]
               |                     |
               |                     |
     [0]    -  2       7     13   -     [0]
               |                     |
               |                     |
     [0]    -  3       4     12   -     [0]
               |                     |
               |                     |
     [0]    -  4       1     11   -     [1]
               |                     |
               |                     |
     [0]    -  5       5     10   -     [1]
               |                     |
               |                     |
     [0]    -  6              9   -     [1]
               |                     |
               |                     |
     [0]    -  7              8   -     [1]
               |                     |
               --------------
{8: 1, 12: 0, 6: 0}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 7416

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7416:

ic = IC_7416()

print(ic.__doc__)
```

```
This is a Hex open-collector high-voltage inverter
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 3: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 11: 1, 13: 0, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                    |
[1]     -  1             14  -     [1]
          |                    |
          |                    |
[0]     -  2       7     13  -     [0]
          |                    |
          |                    |
[0]     -  3       4     12  -     [Z]
          |                    |
          |                    |
[0]     -  4       1     11  -     [1]
          |                    |
          |                    |
[0]     -  5       6     10  -     [1]
          |                    |
          |                    |
[Z]     -  6              9  -     [1]
          |                    |
          |                    |
[0]     -  7              8  -     [Z]
          |                    |
          --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --
```

```
# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{2: 0, 4: 1, 6: 1, 8: 0, 10: 0, 12: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                    |
[1]     -  1             14  -     [1]
          |                    |
          |                    |
[0]     -  2       7     13  -     [0]
          |                    |
          |                    |
[0]     -  3       4     12  -     [1]
          |                    |
          |                    |
[1]     -  4       1     11  -     [1]
          |                    |
          |                    |
[0]     -  5       6     10  -     [0]
          |                    |
          |                    |
[1]     -  6              9  -     [1]
          |                    |
          |                    |
[0]     -  7              8  -     [0]
          |                    |
          -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                    |
     [1]     -  1             14  -     [1]
               |                    |
               |                    |
     [0]     -  2       7     13  -     [0]
               |                    |
               |                    |
     [0]     -  3       4     12  -     [1]
               |                    |
```

```
                |                       |
     [1]    -   4       1      11   -    [1]
                |                       |
                |                       |
     [0]    -   5       6      10   -    [0]
                |                       |
                |                       |
     [1]    -   6              9    -    [1]
                |                       |
                |                       |
     [0]    -   7              8    -    [0]
                |                       |
                -------------
{2: 0, 4: 1, 6: 1, 8: 0, 10: 0, 12: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 7418

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7418:

ic = IC_7418()

print(ic.__doc__)
```

```
This is a Dual 4-input NAND gates with schmitt-trigger inputs.
```

```
# The Pin configuration is:

inp = {    1: 1,    2: 0,    3: 0,    4: 0,    5: 0,    7: 0,    9: 1,    10: 1,
↪ 11: 1,    12: 1,    13: 1,    14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                     |
[1]    -  1             14  -    [1]
          |                     |
          |                     |
[0]    -  2       7     13  -    [1]
          |                     |
          |                     |
[0]    -  3       4     12  -    [1]
          |                     |
          |                     |
[0]    -  4       1     11  -    [1]
          |                     |
          |                     |
[0]    -  5       8     10  -    [1]
          |                     |
          |                     |
[Z]    -  6              9  -    [1]
          |                     |
          |                     |
[0]    -  7              8  -    [Z]
          |                     |
          -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 0, 6: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                     |
[1]    -  1             14  -    [1]
          |                     |
          |                     |
[0]    -  2       7     13  -    [1]
          |                     |
          |                     |
[0]    -  3       4     12  -    [1]
          |                     |
          |                     |
[0]    -  4       1     11  -    [1]
          |                     |
          |                     |
[0]    -  5       8     10  -    [1]
          |                     |
          |                     |
[1]    -  6              9  -    [1]
          |                     |
          |                     |
```

```
[0]    -  7             8  -    [0]
          |                |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                    |
     [1]    -  1             14  -    [1]
               |                    |
               |                    |
     [0]    -  2       7     13  -    [1]
               |                    |
               |                    |
     [0]    -  3       4     12  -    [1]
               |                    |
               |                    |
     [0]    -  4       1     11  -    [1]
               |                    |
               |                    |
     [0]    -  5       8     10  -    [1]
               |                    |
               |                    |
     [1]    -  6              9  -    [1]
               |                    |
               |                    |
     [0]    -  7              8  -    [0]
               |                    |
               --------------
{8: 0, 6: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 7419

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7419:

ic = IC_7419()
```

```
print(ic.__doc__)
```

```
This is a Hex inverters with schmitt-trigger line-receiver inputs.
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 3: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 11: 1, 13: 0, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                    |
[1]     -  1             14  -     [1]
          |                    |
          |                    |
[0]     -  2       7     13  -     [0]
          |                    |
          |                    |
[0]     -  3       4     12  -     [Z]
          |                    |
          |                    |
[0]     -  4       1     11  -     [1]
          |                    |
          |                    |
[0]     -  5       9     10  -     [1]
          |                    |
          |                    |
[Z]     -  6              9  -     [1]
          |                    |
          |                    |
[0]     -  7              8  -     [Z]
          |                    |
          --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{2: 0, 4: 1, 6: 1, 8: 0, 10: 0, 12: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())
```

```
# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14  -     [1]
          |                     |
          |                     |
[0]     -  2       7     13  -     [0]
          |                     |
          |                     |
[0]     -  3       4     12  -     [1]
          |                     |
          |                     |
[1]     -  4       1     11  -     [1]
          |                     |
          |                     |
[0]     -  5       9     10  -     [0]
          |                     |
          |                     |
[1]     -  6              9  -     [1]
          |                     |
          |                     |
[0]     -  7              8  -     [0]
          |                     |
          -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                     |
     [1]     -  1             14  -     [1]
               |                     |
               |                     |
     [0]     -  2       7     13  -     [0]
               |                     |
               |                     |
     [0]     -  3       4     12  -     [1]
               |                     |
               |                     |
     [1]     -  4       1     11  -     [1]
               |                     |
               |                     |
     [0]     -  5       9     10  -     [0]
               |                     |
               |                     |
     [1]     -  6              9  -     [1]
               |                     |
               |                     |
     [0]     -  7              8  -     [0]
```

```
              |                   |
              -------------
{2: 0, 4: 1, 6: 1, 8: 0, 10: 0, 12: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 7420

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7420:

ic = IC_7420()

print(ic.__doc__)
```

```
This is a dual 4-input NAND gate
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 12: 1, 13: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                  |
[1]     -  1             14  -    [1]
          |                  |
          |                  |
[0]     -  2       7     13  -    [1]
          |                  |
          |                  |
[0]     -  3       4     12  -    [1]
          |                  |
          |                  |
[0]     -  4       2     11  -    [0]
          |                  |
```

```
          |                     |
[0]     -  5       0      10  -     [1]
          |                     |
          |                     |
[Z]     -  6               9  -     [1]
          |                     |
          |                     |
[0]     -  7               8  -     [Z]
          |                     |
          --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 0, 6: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1              14  -     [1]
          |                     |
          |                     |
[0]     -  2       7      13  -     [1]
          |                     |
          |                     |
[0]     -  3       4      12  -     [1]
          |                     |
          |                     |
[0]     -  4       2      11  -     [0]
          |                     |
          |                     |
[0]     -  5       0      10  -     [1]
          |                     |
          |                     |
[1]     -  6               9  -     [1]
          |                     |
          |                     |
[0]     -  7               8  -     [0]
          |                     |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC
```

```
print (ic.run())
```

```
               ------------
               |                    |
     [1]    -  1             14  -     [1]
               |                    |
               |                    |
     [0]    -  2       7     13  -     [1]
               |                    |
               |                    |
     [0]    -  3       4     12  -     [1]
               |                    |
               |                    |
     [0]    -  4       2     11  -     [0]
               |                    |
               |                    |
     [0]    -  5       0     10  -     [1]
               |                    |
               |                    |
     [1]    -  6              9  -     [1]
               |                    |
               |                    |
     [0]    -  7              8  -     [0]
               |                    |
               --------------
{8: 0, 6: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 7421

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7421:

ic = IC_7421()

print(ic.__doc__)
```

```
This is a dual 4-input AND gate
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 12: 1, 13: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})
```

```
ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14  -     [1]
          |                     |
          |                     |
[0]     -  2       7     13  -     [1]
          |                     |
          |                     |
[0]     -  3       4     12  -     [1]
          |                     |
          |                     |
[0]     -  4       2     11  -     [0]
          |                     |
          |                     |
[0]     -  5       1     10  -     [1]
          |                     |
          |                     |
[Z]     -  6              9  -     [1]
          |                     |
          |                     |
[0]     -  7              8  -     [Z]
          |                     |
          -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 1, 6: 0}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14  -     [1]
          |                     |
          |                     |
[0]     -  2       7     13  -     [1]
          |                     |
          |                     |
[0]     -  3       4     12  -     [1]
```

```
          |                     |
          |                     |
[0]    -  4       2     11  -    [0]
          |                     |
          |                     |
[0]    -  5       1     10  -    [1]
          |                     |
          |                     |
[0]    -  6              9  -    [1]
          |                     |
          |                     |
[0]    -  7              8  -    [1]
          |                     |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                     |
     [1]    -  1             14  -    [1]
               |                     |
               |                     |
     [0]    -  2       7     13  -    [1]
               |                     |
               |                     |
     [0]    -  3       4     12  -    [1]
               |                     |
               |                     |
     [0]    -  4       2     11  -    [0]
               |                     |
               |                     |
     [0]    -  5       1     10  -    [1]
               |                     |
               |                     |
     [0]    -  6              9  -    [1]
               |                     |
               |                     |
     [0]    -  7              8  -    [1]
               |                     |
               -------------
{8: 1, 6: 0}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 7422

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7422:

ic = IC_7422()

print(ic.__doc__)
```

```
This is a dual 4-input NAND gate with open collector outputs
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 12: 1, 13: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14   -     [1]
          |                     |
          |                     |
[0]     -  2       7     13   -     [1]
          |                     |
          |                     |
[0]     -  3       4     12   -     [1]
          |                     |
          |                     |
[0]     -  4       2     11   -     [0]
          |                     |
          |                     |
[0]     -  5       2     10   -     [1]
          |                     |
          |                     |
[Z]     -  6              9   -     [1]
          |                     |
          |                     |
[0]     -  7              8   -     [Z]
          |                     |
          -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --
```

```
# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 0, 6: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                      |
[1]     -  1             14   -     [1]
          |                      |
          |                      |
[0]     -  2       7     13   -     [1]
          |                      |
          |                      |
[0]     -  3       4     12   -     [1]
          |                      |
          |                      |
[0]     -  4       2     11   -     [0]
          |                      |
          |                      |
[0]     -  5       2     10   -     [1]
          |                      |
          |                      |
[1]     -  6              9   -     [1]
          |                      |
          |                      |
[0]     -  7              8   -     [0]
          |                      |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                      |
     [1]     -  1             14   -     [1]
               |                      |
               |                      |
     [0]     -  2       7     13   -     [1]
               |                      |
               |                      |
     [0]     -  3       4     12   -     [1]
               |                      |
```

```
                |                       |
     [0]     -  4       2      11   -     [0]
                |                       |
                |                       |
     [0]     -  5       2      10   -     [1]
                |                       |
                |                       |
     [1]     -  6              9   -     [1]
                |                       |
                |                       |
     [0]     -  7              8   -     [0]
                |                       |
               -------------
{8: 0, 6: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 7424

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7424:

ic = IC_7424()

print(ic.__doc__)
```

```
This is a Quad 2-input NAND gates with schmitt-trigger line-receiver inputs
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 12: 1, 13: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                  |
```

```
[1]    -  1             14  -    [1]
          |                 |
          |                 |
[0]    -  2       7     13  -    [1]
          |                 |
          |                 |
[Z]    -  3       4     12  -    [1]
          |                 |
          |                 |
[0]    -  4       2     11  -    [Z]
          |                 |
          |                 |
[0]    -  5       4     10  -    [1]
          |                 |
          |                 |
[Z]    -  6              9  -    [1]
          |                 |
          |                 |
[0]    -  7              8  -    [Z]
          |                 |
          -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 0, 11: 0, 3: 1, 6: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                 |
[1]    -  1             14  -    [1]
          |                 |
          |                 |
[0]    -  2       7     13  -    [1]
          |                 |
          |                 |
[1]    -  3       4     12  -    [1]
          |                 |
          |                 |
[0]    -  4       2     11  -    [0]
          |                 |
          |                 |
[0]    -  5       4     10  -    [1]
          |                 |
          |                 |
[1]    -  6              9  -    [1]
          |                 |
          |                 |
[0]    -  7              8  -    [0]
          |                 |
```

```
         -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                    |
     [1]    -  1             14  -     [1]
               |                    |
               |                    |
     [0]    -  2       7     13  -     [1]
               |                    |
               |                    |
     [1]    -  3       4     12  -     [1]
               |                    |
               |                    |
     [0]    -  4       2     11  -     [0]
               |                    |
               |                    |
     [0]    -  5       4     10  -     [1]
               |                    |
               |                    |
     [1]    -  6              9  -     [1]
               |                    |
               |                    |
     [0]    -  7              8  -     [0]
               |                    |
               --------------
{8: 0, 11: 0, 3: 1, 6: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 7425

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7425:

ic = IC_7425()

print(ic.__doc__)
```

```
This is a Dual 5-Input NOR Gate with Strobe
```

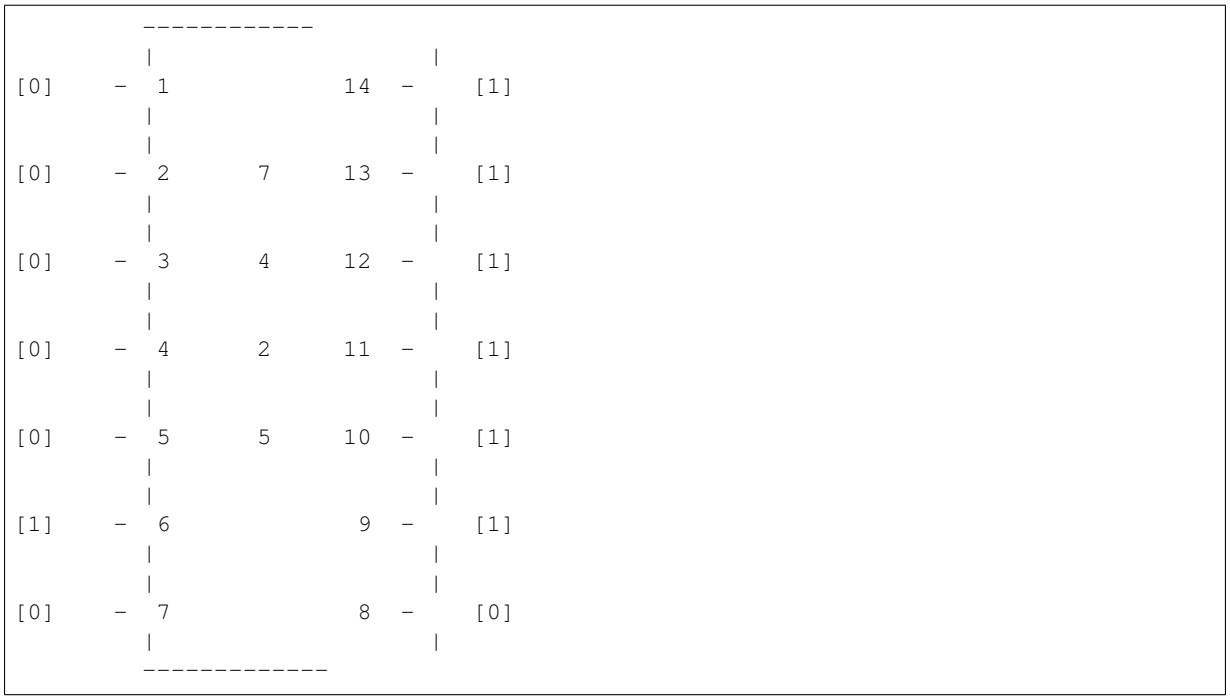
```
# The Pin configuration is:

inp = { 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 11: 1, 12: 1, 13: 1, 14:␣
↪1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                     |
[0]     -  1             14   -     [1]
          |                     |
          |                     |
[0]     -  2       7     13   -     [1]
          |                     |
          |                     |
[0]     -  3       4     12   -     [1]
          |                     |
          |                     |
[0]     -  4       2     11   -     [1]
          |                     |
          |                     |
[0]     -  5       5     10   -     [1]
          |                     |
          |                     |
[Z]     -  6              9   -     [1]
          |                     |
          |                     |
[0]     -  7              8   -     [Z]
          |                     |
          -------------
```

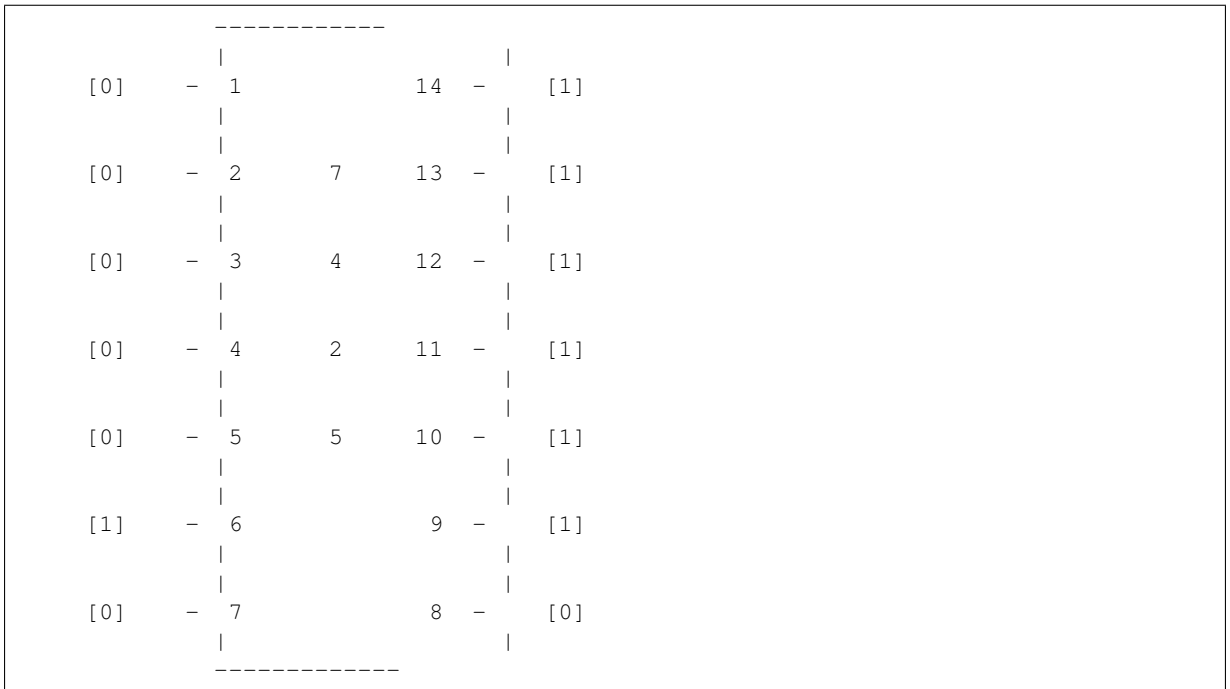
```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 0, 6: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```




```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```




```
{8: 0, 6: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 7426

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7426:

ic = IC_7426()

print(ic.__doc__)
```

```
This is a Quad 2-input open-collector high-voltage NAND gates.
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 12: 1, 13: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                    |
[1]     -  1             14  -     [1]
          |                    |
          |                    |
[0]     -  2       7     13  -     [1]
          |                    |
          |                    |
[Z]     -  3       4     12  -     [1]
          |                    |
          |                    |
[0]     -  4       2     11  -     [Z]
          |                    |
          |                    |
[0]     -  5       6     10  -     [1]
          |                    |
          |                    |
[Z]     -  6              9  -     [1]
```

```
          |                    |
          |                    |
[0]    -  7              8   -     [Z]
          |                    |
          --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 0, 11: 0, 3: 1, 6: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                    |
[1]    -  1             14  -     [1]
          |                    |
          |                    |
[0]    -  2       7     13  -     [1]
          |                    |
          |                    |
[1]    -  3       4     12  -     [1]
          |                    |
          |                    |
[0]    -  4       2     11  -     [0]
          |                    |
          |                    |
[0]    -  5       6     10  -     [1]
          |                    |
          |                    |
[1]    -  6              9   -     [1]
          |                    |
          |                    |
[0]    -  7              8   -     [0]
          |                    |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
                ------------
                |                    |
     [1]    -   1             14   -     [1]
                |                    |
                |                    |
     [0]    -   2       7     13   -     [1]
                |                    |
                |                    |
     [1]    -   3       4     12   -     [1]
                |                    |
                |                    |
     [0]    -   4       2     11   -     [0]
                |                    |
                |                    |
     [0]    -   5       6     10   -     [1]
                |                    |
                |                    |
     [1]    -   6              9   -     [1]
                |                    |
                |                    |
     [0]    -   7              8   -     [0]
                |                    |
                -------------
{8: 0, 11: 0, 3: 1, 6: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 7427

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7427:

ic = IC_7427()

print(ic.__doc__)
```

```
This is a Triple 3-Input NOR Gate
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 3: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 11: 1, 13: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic
```

```
ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14  -     [1]
          |                     |
          |                     |
[0]     -  2       7     13  -     [1]
          |                     |
          |                     |
[0]     -  3       4     12  -     [Z]
          |                     |
          |                     |
[0]     -  4       2     11  -     [1]
          |                     |
          |                     |
[0]     -  5       7     10  -     [1]
          |                     |
          |                     |
[Z]     -  6              9  -     [1]
          |                     |
          |                     |
[0]     -  7              8  -     [Z]
          |                     |
          --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 0, 12: 0, 6: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14  -     [1]
          |                     |
          |                     |
[0]     -  2       7     13  -     [1]
          |                     |
          |                     |
[0]     -  3       4     12  -     [0]
          |                     |
          |                     |
[0]     -  4       2     11  -     [1]
          |                     |
```

```
          |                     |
[0]    -  5      7     10   -    [1]
          |                     |
          |                     |
[1]    -  6             9   -    [1]
          |                     |
          |                     |
[0]    -  7             8   -    [0]
          |                     |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                     |
     [1]    -  1             14  -    [1]
               |                     |
               |                     |
     [0]    -  2      7      13  -    [1]
               |                     |
               |                     |
     [0]    -  3      4      12  -    [0]
               |                     |
               |                     |
     [0]    -  4      2      11  -    [1]
               |                     |
               |                     |
     [0]    -  5      7      10  -    [1]
               |                     |
               |                     |
     [1]    -  6             9   -    [1]
               |                     |
               |                     |
     [0]    -  7             8   -    [0]
               |                     |
               -------------
{8: 0, 12: 0, 6: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 7428

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7428:

ic = IC_7428()

print(ic.__doc__)
```

```
This is a Quad 2-input NOR gates with buffered outputs.
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 12: 0, 13: 0, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14   -     [1]
          |                     |
          |                     |
[0]     -  2       7     13   -     [0]
          |                     |
          |                     |
[0]     -  3       4     12   -     [0]
          |                     |
          |                     |
[0]     -  4       2     11   -     [0]
          |                     |
          |                     |
[0]     -  5       8     10   -     [1]
          |                     |
          |                     |
[0]     -  6              9   -     [1]
          |                     |
          |                     |
[0]     -  7              8   -     [0]
          |                     |
          -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{1: 1, 10: 0, 4: 1, 13: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                    |
[1]     -  1             14  -     [1]
          |                    |
          |                    |
[0]     -  2       7     13  -     [1]
          |                    |
          |                    |
[0]     -  3       4     12  -     [0]
          |                    |
          |                    |
[1]     -  4       2     11  -     [0]
          |                    |
          |                    |
[0]     -  5       8     10  -     [0]
          |                    |
          |                    |
[0]     -  6              9  -     [1]
          |                    |
          |                    |
[0]     -  7              8  -     [0]
          |                    |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                    |
     [1]     -  1             14  -     [1]
               |                    |
               |                    |
     [0]     -  2       7     13  -     [1]
               |                    |
               |                    |
     [0]     -  3       4     12  -     [0]
               |                    |
               |                    |
     [1]     -  4       2     11  -     [0]
               |                    |
               |                    |
```

```
     [0]    -  5      8     10  -    [0]
               |                 |
               |                 |
     [0]    -  6             9  -    [1]
               |                 |
               |                 |
     [0]    -  7             8  -    [0]
               |                 |
               -------------
{1: 1, 10: 0, 4: 1, 13: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(1, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 7430

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7430:

ic = IC_7430()

print(ic.__doc__)
```

```
This is a 8-Input NAND Gate
```

```
# The Pin configuration is:

inp = {1: 0, 2: 1, 3: 1, 4: 1, 5: 1, 6: 1, 7: 0, 11: 1, 12: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                  |
[0]    -  1            14  -    [1]
          |                  |
          |                  |
[1]    -  2      7     13  -    [0]
```

```
          |                     |
          |                     |
[1]    -  3       4      12   -     [1]
          |                     |
          |                     |
[1]    -  4       3      11   -     [1]
          |                     |
          |                     |
[1]    -  5       0      10   -     [0]
          |                     |
          |                     |
[1]    -  6              9   -     [0]
          |                     |
          |                     |
[0]    -  7              8   -     [Z]
          |                     |
          --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          -------------
          |                     |
[0]    -  1             14   -     [1]
          |                     |
          |                     |
[1]    -  2       7     13   -     [0]
          |                     |
          |                     |
[1]    -  3       4     12   -     [1]
          |                     |
          |                     |
[1]    -  4       3     11   -     [1]
          |                     |
          |                     |
[1]    -  5       0     10   -     [0]
          |                     |
          |                     |
[1]    -  6              9   -     [0]
          |                     |
          |                     |
[0]    -  7              8   -     [1]
          |                     |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
                ------------
                |                      |
     [0]     -  1             14   -     [1]
                |                      |
                |                      |
     [1]     -  2       7     13   -     [0]
                |                      |
                |                      |
     [1]     -  3       4     12   -     [1]
                |                      |
                |                      |
     [1]     -  4       3     11   -     [1]
                |                      |
                |                      |
     [1]     -  5       0     10   -     [0]
                |                      |
                |                      |
     [1]     -  6              9   -     [0]
                |                      |
                |                      |
     [0]     -  7              8   -     [1]
                |                      |
                --------------
{8: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 7431

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7431:

ic = IC_7431()

print(ic.__doc__)
```

```
This is a Hex delay element.
```

```
# The Pin configuration is:

inp = {1: 1, 3: 1, 5: 0, 6: 0, 8: 0, 10: 1, 11: 1, 13: 0, 15: 1, 16: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                    |
[1]     -  1             16  -     [1]
          |                    |
          |                    |
[Z]     -  2       7     15  -     [1]
          |                    |
          |                    |
[1]     -  3       4     14  -     [1]
          |                    |
          |                    |
[Z]     -  4       3     13  -     [0]
          |                    |
          |                    |
[0]     -  5       1     12  -     [Z]
          |                    |
          |                    |
[0]     -  6             11  -     [1]
          |                    |
          |                    |
[0]     -  7             10  -     [1]
          |                    |
          |                    |
[0]     -  8              9  -     [Z]
          |                    |
          --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{2: 0, 4: 1, 7: 1, 9: 0, 12: 0, 14: 0}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration
```

```
ic.drawIC()
```

```
          ------------
          |                    |
[1]    -  1             16  -     [1]
          |                    |
          |                    |
[0]    -  2       7     15  -     [1]
          |                    |
          |                    |
[1]    -  3       4     14  -     [0]
          |                    |
          |                    |
[1]    -  4       3     13  -     [0]
          |                    |
          |                    |
[0]    -  5       1     12  -     [0]
          |                    |
          |                    |
[0]    -  6             11  -     [1]
          |                    |
          |                    |
[1]    -  7             10  -     [1]
          |                    |
          |                    |
[0]    -  8              9  -     [0]
          |                    |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                    |
     [1]    -  1             16  -     [1]
               |                    |
               |                    |
     [0]    -  2       7     15  -     [1]
               |                    |
               |                    |
     [1]    -  3       4     14  -     [0]
               |                    |
               |                    |
     [1]    -  4       3     13  -     [0]
               |                    |
               |                    |
     [0]    -  5       1     12  -     [0]
               |                    |
               |                    |
     [0]    -  6             11  -     [1]
               |                    |
```

```
                |                     |
     [1]    -   7             10   -    [1]
                |                     |
                |                     |
     [0]    -   8              9   -    [0]
                |                     |
                -------------
{2: 0, 4: 1, 7: 1, 9: 0, 12: 0, 14: 0}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(9, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 7432

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7432:

ic = IC_7432()

print(ic.__doc__)
```

```
This is a Quad 2-Input OR Gate
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 12: 1, 13: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                   |
[1]    -   1             14   -    [1]
          |                   |
          |                   |
[0]    -   2       7     13   -    [1]
          |                   |
          |                   |
```

```
[Z]     -  3       4      12  -     [1]
           |                      |
           |                      |
[0]     -  4       3      11  -     [Z]
           |                      |
           |                      |
[0]     -  5       2      10  -     [1]
           |                      |
           |                      |
[Z]     -  6              9  -     [1]
           |                      |
           |                      |
[0]     -  7              8  -     [Z]
           |                      |
           --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 1, 11: 1, 3: 1, 6: 0}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
           ------------
           |                      |
[1]     -  1              14  -     [1]
           |                      |
           |                      |
[0]     -  2       7      13  -     [1]
           |                      |
           |                      |
[1]     -  3       4      12  -     [1]
           |                      |
           |                      |
[0]     -  4       3      11  -     [1]
           |                      |
           |                      |
[0]     -  5       2      10  -     [1]
           |                      |
           |                      |
[0]     -  6              9  -     [1]
           |                      |
           |                      |
[0]     -  7              8  -     [1]
           |                      |
           --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())
```

```
# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                      |
     [1]    -  1             14   -    [1]
               |                      |
               |                      |
     [0]    -  2       7     13   -    [1]
               |                      |
               |                      |
     [1]    -  3       4     12   -    [1]
               |                      |
               |                      |
     [0]    -  4       3     11   -    [1]
               |                      |
               |                      |
     [0]    -  5       2     10   -    [1]
               |                      |
               |                      |
     [0]    -  6              9   -    [1]
               |                      |
               |                      |
     [0]    -  7              8   -    [1]
               |                      |
               --------------
{8: 1, 11: 1, 3: 1, 6: 0}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 7433

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7433:

ic = IC_7433()

print(ic.__doc__)
```

```
This is a Quad 2-input open-collector NOR gate
```

```
# The Pin configuration is:

```

```
inp = {2: 0, 3: 0, 5: 0, 6: 0, 7: 0, 8: 1, 9: 1, 11: 1, 12: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                      |
[Z]     -  1             14   -     [1]
          |                      |
          |                      |
[0]     -  2       7     13   -     [Z]
          |                      |
          |                      |
[0]     -  3       4     12   -     [1]
          |                      |
          |                      |
[Z]     -  4       3     11   -     [1]
          |                      |
          |                      |
[0]     -  5       3     10   -     [Z]
          |                      |
          |                      |
[0]     -  6              9   -     [1]
          |                      |
          |                      |
[0]     -  7              8   -     [1]
          |                      |
          --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{1: 1, 10: 0, 4: 1, 13: 0}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                      |
[1]     -  1             14   -     [1]
          |                      |
```

```
          |                       |
[0]     -  2       7      13   -     [0]
          |                       |
          |                       |
[0]     -  3       4      12   -     [1]
          |                       |
          |                       |
[1]     -  4       3      11   -     [1]
          |                       |
          |                       |
[0]     -  5       3      10   -     [0]
          |                       |
          |                       |
[0]     -  6              9   -     [1]
          |                       |
          |                       |
[0]     -  7              8   -     [1]
          |                       |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                       |
     [1]     -  1             14   -     [1]
               |                       |
               |                       |
     [0]     -  2       7      13   -     [0]
               |                       |
               |                       |
     [0]     -  3       4      12   -     [1]
               |                       |
               |                       |
     [1]     -  4       3      11   -     [1]
               |                       |
               |                       |
     [0]     -  5       3      10   -     [0]
               |                       |
               |                       |
     [0]     -  6              9   -     [1]
               |                       |
               |                       |
     [0]     -  7              8   -     [1]
               |                       |
               --------------
{1: 1, 10: 0, 4: 1, 13: 0}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
```

```
ic.setOutput(1, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 7437

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7437:

ic = IC_7437()

print(ic.__doc__)
```

```
This is a Quad 2-input NAND gates with buffered output
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 12: 1, 13: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14  -     [1]
          |                     |
          |                     |
[0]     -  2       7     13  -     [1]
          |                     |
          |                     |
[Z]     -  3       4     12  -     [1]
          |                     |
          |                     |
[0]     -  4       3     11  -     [Z]
          |                     |
          |                     |
[0]     -  5       7     10  -     [1]
          |                     |
          |                     |
[Z]     -  6              9  -     [1]
          |                     |
          |                     |
[0]     -  7              8  -     [Z]
```

```
         |                    |
         -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 0, 11: 0, 3: 1, 6: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                    |
[1]     -  1             14  -     [1]
          |                    |
          |                    |
[0]     -  2       7     13  -     [1]
          |                    |
          |                    |
[1]     -  3       4     12  -     [1]
          |                    |
          |                    |
[0]     -  4       3     11  -     [0]
          |                    |
          |                    |
[0]     -  5       7     10  -     [1]
          |                    |
          |                    |
[1]     -  6              9  -     [1]
          |                    |
          |                    |
[0]     -  7              8  -     [0]
          |                    |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                    |
     [1]     -  1             14  -     [1]
               |                    |
```

```
                |                       |
     [0]     -  2       7      13   -     [1]
                |                       |
                |                       |
     [1]     -  3       4      12   -     [1]
                |                       |
                |                       |
     [0]     -  4       3      11   -     [0]
                |                       |
                |                       |
     [0]     -  5       7      10   -     [1]
                |                       |
                |                       |
     [1]     -  6              9   -     [1]
                |                       |
                |                       |
     [0]     -  7              8   -     [0]
                |                       |
                --------------
{8: 0, 11: 0, 3: 1, 6: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 7400

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7440:

ic = IC_7440()

print(ic.__doc__)
```

```
This is a Dual 4-Input NAND Buffer
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 12: 1, 13: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n
```

```
ic.drawIC()
```

```
          ------------
          |                    |
[1]     -  1             14  -     [1]
          |                    |
          |                    |
[0]     -  2       7     13  -     [1]
          |                    |
          |                    |
[0]     -  3       4     12  -     [1]
          |                    |
          |                    |
[0]     -  4       4     11  -     [0]
          |                    |
          |                    |
[0]     -  5       0     10  -     [1]
          |                    |
          |                    |
[Z]     -  6              9  -     [1]
          |                    |
          |                    |
[0]     -  7              8  -     [Z]
          |                    |
          --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 0, 6: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                    |
[1]     -  1             14  -     [1]
          |                    |
          |                    |
[0]     -  2       7     13  -     [1]
          |                    |
          |                    |
[0]     -  3       4     12  -     [1]
          |                    |
          |                    |
[0]     -  4       4     11  -     [0]
          |                    |
          |                    |
[0]     -  5       0     10  -     [1]
          |                    |
          |                    |
```

```
[1]    -  6              9  -    [1]
          |                 |
          |                 |
[0]    -  7              8  -    [0]
          |                 |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                    |
     [1]    -  1             14  -    [1]
               |                    |
               |                    |
     [0]    -  2       7     13  -    [1]
               |                    |
               |                    |
     [0]    -  3       4     12  -    [1]
               |                    |
               |                    |
     [0]    -  4       4     11  -    [0]
               |                    |
               |                    |
     [0]    -  5       0     10  -    [1]
               |                    |
               |                    |
     [1]    -  6              9  -    [1]
               |                    |
               |                    |
     [0]    -  7              8  -    [0]
               |                    |
               --------------
{8: 0, 6: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 7442

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7442:

ic = IC_7442()

print(ic.__doc__)
```

```
This is a BCD to Decimal decoder
BCD Digits are in order of A B C D where pin 15 = A, pin 12 = D
```

```
# The Pin configuration is:

inp = {8: 0, 12: 0, 13: 0, 14: 0, 15: 1, 16: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                    |
[Z]     -  1             16  -     [1]
          |                    |
          |                    |
[Z]     -  2       7     15  -     [1]
          |                    |
          |                    |
[Z]     -  3       4     14  -     [0]
          |                    |
          |                    |
[Z]     -  4       4     13  -     [0]
          |                    |
          |                    |
[Z]     -  5       2     12  -     [0]
          |                    |
          |                    |
[Z]     -  6             11  -     [Z]
          |                    |
          |                    |
[0]     -  7             10  -     [Z]
          |                    |
          |                    |
[0]     -  8              9  -     [Z]
          |                    |
          --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{1: 1, 2: 0, 3: 1, 4: 1, 5: 1, 6: 1, 7: 1, 9: 1, 10: 1, 11: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                    |
[1]     -  1             16  -     [1]
          |                    |
          |                    |
[0]     -  2       7     15  -     [1]
          |                    |
          |                    |
[1]     -  3       4     14  -     [0]
          |                    |
          |                    |
[1]     -  4       4     13  -     [0]
          |                    |
          |                    |
[1]     -  5       2     12  -     [0]
          |                    |
          |                    |
[1]     -  6             11  -     [1]
          |                    |
          |                    |
[1]     -  7             10  -     [1]
          |                    |
          |                    |
[0]     -  8              9  -     [1]
          |                    |
          -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                    |
     [1]     -  1             16  -     [1]
               |                    |
               |                    |
     [0]     -  2       7     15  -     [1]
               |                    |
               |                    |
     [1]     -  3       4     14  -     [0]
               |                    |
               |                    |
```

```
     [1]    -  4       4      13  -    [0]
               |                   |
               |                   |
     [1]    -  5       2      12  -    [0]
               |                   |
               |                   |
     [1]    -  6              11  -    [1]
               |                   |
               |                   |
     [1]    -  7              10  -    [1]
               |                   |
               |                   |
     [0]    -  8               9  -    [1]
               |                   |
               -------------
{1: 1, 2: 0, 3: 1, 4: 1, 5: 1, 6: 1, 7: 1, 9: 1, 10: 1, 11: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(1, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 7443

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7443:

ic = IC_7443()

print(ic.__doc__)
```

```
This is an excess-3 to Decimal decoder
Excess-3 binary digits are in order of A B C D, where pin 15 = A and pin 12 = D
```

```
# The Pin configuration is:

inp = {8: 0, 12: 0, 13: 1, 14: 0, 15: 1, 16: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                    |
[Z]     -  1             16  -     [1]
          |                    |
          |                    |
[Z]     -  2       7     15  -     [1]
          |                    |
          |                    |
[Z]     -  3       4     14  -     [0]
          |                    |
          |                    |
[Z]     -  4       4     13  -     [1]
          |                    |
          |                    |
[Z]     -  5       3     12  -     [0]
          |                    |
          |                    |
[Z]     -  6             11  -     [Z]
          |                    |
          |                    |
[0]     -  7             10  -     [Z]
          |                    |
          |                    |
[0]     -  8              9  -     [Z]
          |                    |
          --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{1: 1, 2: 1, 3: 0, 4: 1, 5: 1, 6: 1, 7: 1, 9: 1, 10: 1, 11: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                    |
[1]     -  1             16  -     [1]
          |                    |
          |                    |
[1]     -  2       7     15  -     [1]
          |                    |
          |                    |
[0]     -  3       4     14  -     [0]
          |                    |
          |                    |
[1]     -  4       4     13  -     [1]
          |                    |
          |                    |
[1]     -  5       3     12  -     [0]
          |                    |
          |                    |
```

```
[1]    -  6            11  -    [1]
          |                 |
          |                 |
[1]    -  7            10  -    [1]
          |                 |
          |                 |
[0]    -  8             9  -    [1]
          |                 |
          -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                    |
     [1]    -  1            16  -    [1]
               |                    |
               |                    |
     [1]    -  2      7     15  -    [1]
               |                    |
               |                    |
     [0]    -  3      4     14  -    [0]
               |                    |
               |                    |
     [1]    -  4      4     13  -    [1]
               |                    |
               |                    |
     [1]    -  5      3     12  -    [0]
               |                    |
               |                    |
     [1]    -  6            11  -    [1]
               |                    |
               |                    |
     [1]    -  7            10  -    [1]
               |                    |
               |                    |
     [0]    -  8             9  -    [1]
               |                    |
               -------------
{1: 1, 2: 1, 3: 0, 4: 1, 5: 1, 6: 1, 7: 1, 9: 1, 10: 1, 11: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(1, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 7444

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7444:

ic = IC_7444()

print(ic.__doc__)
```

```
This is an excess-3 gray code to Decimal decoder
Excess-3 gray code digits are in order of A B C D, where pin 15 = A and pin 12 = D
```

```
# The Pin configuration is:

inp = {8: 0, 12: 0, 13: 1, 14: 0, 15: 1, 16: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                     |
[Z]     -  1             16  -     [1]
          |                     |
          |                     |
[Z]     -  2       7     15  -     [1]
          |                     |
          |                     |
[Z]     -  3       4     14  -     [0]
          |                     |
          |                     |
[Z]     -  4       4     13  -     [1]
          |                     |
          |                     |
[Z]     -  5       4     12  -     [0]
          |                     |
          |                     |
[Z]     -  6             11  -     [Z]
          |                     |
          |                     |
[0]     -  7             10  -     [Z]
          |                     |
          |                     |
[0]     -  8              9  -     [Z]
          |                     |
```

```
         -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{1: 1, 2: 1, 3: 1, 4: 0, 5: 1, 6: 1, 7: 1, 9: 1, 10: 1, 11: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          -------------
          |                     |
[1]     -  1              16  -     [1]
          |                     |
          |                     |
[1]     -  2       7      15  -     [1]
          |                     |
          |                     |
[1]     -  3       4      14  -     [0]
          |                     |
          |                     |
[0]     -  4       4      13  -     [1]
          |                     |
          |                     |
[1]     -  5       4      12  -     [0]
          |                     |
          |                     |
[1]     -  6              11  -     [1]
          |                     |
          |                     |
[1]     -  7              10  -     [1]
          |                     |
          |                     |
[0]     -  8               9  -     [1]
          |                     |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
              -------------
              |                     |
```

```
     [1]    -  1             16  -    [1]
               |                   |
               |                   |
     [1]    -  2       7     15  -    [1]
               |                   |
               |                   |
     [1]    -  3       4     14  -    [0]
               |                   |
               |                   |
     [0]    -  4       4     13  -    [1]
               |                   |
               |                   |
     [1]    -  5       4     12  -    [0]
               |                   |
               |                   |
     [1]    -  6             11  -    [1]
               |                   |
               |                   |
     [1]    -  7             10  -    [1]
               |                   |
               |                   |
     [0]    -  8              9  -    [1]
               |                   |
               -------------
{1: 1, 2: 1, 3: 1, 4: 0, 5: 1, 6: 1, 7: 1, 9: 1, 10: 1, 11: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(1, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 7445

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7445:

ic = IC_7445()

print(ic.__doc__)
```

```
This is a Four-to-Ten (BCD to Decimal) DECODER
datasheet at http://www.skot9000.com/ttl/datasheets/45.pdf
```

```
# The Pin configuration is:

inp = {8: 0, 12: 0, 13: 1, 14: 0, 15: 0, 16: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})
```

```
# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                     |
[Z]     -  1             16  -     [1]
          |                     |
          |                     |
[Z]     -  2       7     15  -     [0]
          |                     |
          |                     |
[Z]     -  3       4     14  -     [0]
          |                     |
          |                     |
[Z]     -  4       4     13  -     [1]
          |                     |
          |                     |
[Z]     -  5       5     12  -     [0]
          |                     |
          |                     |
[Z]     -  6             11  -     [Z]
          |                     |
          |                     |
[0]     -  7             10  -     [Z]
          |                     |
          |                     |
[0]     -  8              9  -     [Z]
          |                     |
          -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{1: 1, 2: 1, 3: 1, 4: 1, 5: 0, 6: 1, 7: 1, 9: 1, 10: 1, 11: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             16  -     [1]
          |                     |
          |                     |
[1]     -  2       7     15  -     [0]
          |                     |
          |                     |
```

```
[1]    -  3       4      14  -    [0]
          |                     |
          |                     |
[1]    -  4       4      13  -    [1]
          |                     |
          |                     |
[0]    -  5       5      12  -    [0]
          |                     |
          |                     |
[1]    -  6              11  -    [1]
          |                     |
          |                     |
[1]    -  7              10  -    [1]
          |                     |
          |                     |
[0]    -  8               9  -    [1]
          |                     |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                     |
     [1]    -  1             16  -    [1]
               |                     |
               |                     |
     [1]    -  2       7     15  -    [0]
               |                     |
               |                     |
     [1]    -  3       4     14  -    [0]
               |                     |
               |                     |
     [1]    -  4       4     13  -    [1]
               |                     |
               |                     |
     [0]    -  5       5     12  -    [0]
               |                     |
               |                     |
     [1]    -  6             11  -    [1]
               |                     |
               |                     |
     [1]    -  7             10  -    [1]
               |                     |
               |                     |
     [0]    -  8              9  -    [1]
               |                     |
               --------------
{1: 1, 2: 1, 3: 1, 4: 1, 5: 0, 6: 1, 7: 1, 9: 1, 10: 1, 11: 1}
```

```
# Connector Outputs
c = Connector()
```

```
# Set the output connector to a particular pin of the ic
ic.setOutput(1, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 7451

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7451:

ic = IC_7451()

print(ic.__doc__)
```

```
This is a dual 2-wide 2-input AND-OR Invert gate
```

```
# The Pin configuration is:

inp = {
    1: 1,
    2: 1,
    3: 0,
    4: 0,
    5: 0,
    7: 0,
    9: 0,
    10: 0,
    11: 0,
    12: 1,
    13: 1,
    14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -   1             14  -     [1]
          |                     |
          |                     |
[1]     -   2       7     13  -     [1]
          |                     |
          |                     |
```

```
[0]    -  3       4      12  -    [1]
          |                      |
          |                      |
[0]    -  4       5      11  -    [0]
          |                      |
          |                      |
[0]    -  5       1      10  -    [0]
          |                      |
          |                      |
[0]    -  6              9  -    [0]
          |                      |
          |                      |
[0]    -  7              8  -    [0]
          |                      |
          --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 0, 6: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                      |
[1]    -  1              14  -    [1]
          |                      |
          |                      |
[1]    -  2       7      13  -    [1]
          |                      |
          |                      |
[0]    -  3       4      12  -    [1]
          |                      |
          |                      |
[0]    -  4       5      11  -    [0]
          |                      |
          |                      |
[0]    -  5       1      10  -    [0]
          |                      |
          |                      |
[1]    -  6              9  -    [0]
          |                      |
          |                      |
[0]    -  7              8  -    [0]
          |                      |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())
```

```
# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                      |
     [1]     -  1             14   -     [1]
               |                      |
               |                      |
     [1]     -  2       7     13   -     [1]
               |                      |
               |                      |
     [0]     -  3       4     12   -     [1]
               |                      |
               |                      |
     [0]     -  4       5     11   -     [0]
               |                      |
               |                      |
     [0]     -  5       1     10   -     [0]
               |                      |
               |                      |
     [1]     -  6              9   -     [0]
               |                      |
               |                      |
     [0]     -  7              8   -     [0]
               |                      |
               --------------
{8: 0, 6: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 7454

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7454:

ic = IC_7454()

print(ic.__doc__)
```

```
This is a 4-wide 2-input AND-OR Invert gate
```

```
# The Pin configuration is:

```

```
inp =  {1: 1, 2: 0, 3: 0, 4: 0, 7: 0, 9: 1, 10: 1, 11: 0, 12: 0, 13: 0, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14  -     [1]
          |                     |
          |                     |
[0]     -  2       7     13  -     [0]
          |                     |
          |                     |
[0]     -  3       4     12  -     [0]
          |                     |
          |                     |
[0]     -  4       5     11  -     [0]
          |                     |
          |                     |
[Z]     -  5       4     10  -     [1]
          |                     |
          |                     |
[0]     -  6              9  -     [1]
          |                     |
          |                     |
[0]     -  7              8  -     [0]
          |                     |
          -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{6: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14  -     [1]
          |                     |
```

```
          |                       |
[0]     -  2       7      13   -     [0]
          |                       |
          |                       |
[0]     -  3       4      12   -     [0]
          |                       |
          |                       |
[0]     -  4       5      11   -     [0]
          |                       |
          |                       |
[Z]     -  5       4      10   -     [1]
          |                       |
          |                       |
[1]     -  6              9   -     [1]
          |                       |
          |                       |
[0]     -  7              8   -     [0]
          |                       |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                       |
     [1]     -  1             14   -     [1]
               |                       |
               |                       |
     [0]     -  2       7      13   -     [0]
               |                       |
               |                       |
     [0]     -  3       4      12   -     [0]
               |                       |
               |                       |
     [0]     -  4       5      11   -     [0]
               |                       |
               |                       |
     [Z]     -  5       4      10   -     [1]
               |                       |
               |                       |
     [1]     -  6              9   -     [1]
               |                       |
               |                       |
     [0]     -  7              8   -     [0]
               |                       |
               --------------
{6: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
```

```
ic.setOutput(6, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 7455

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7455:

ic = IC_7455()

print(ic.__doc__)
```

```
This is a 4-wide 2-input AND-OR Invert gate
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 3: 0, 4: 0, 7: 0, 9: 1, 10: 1, 11: 0, 12: 0, 13: 0, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

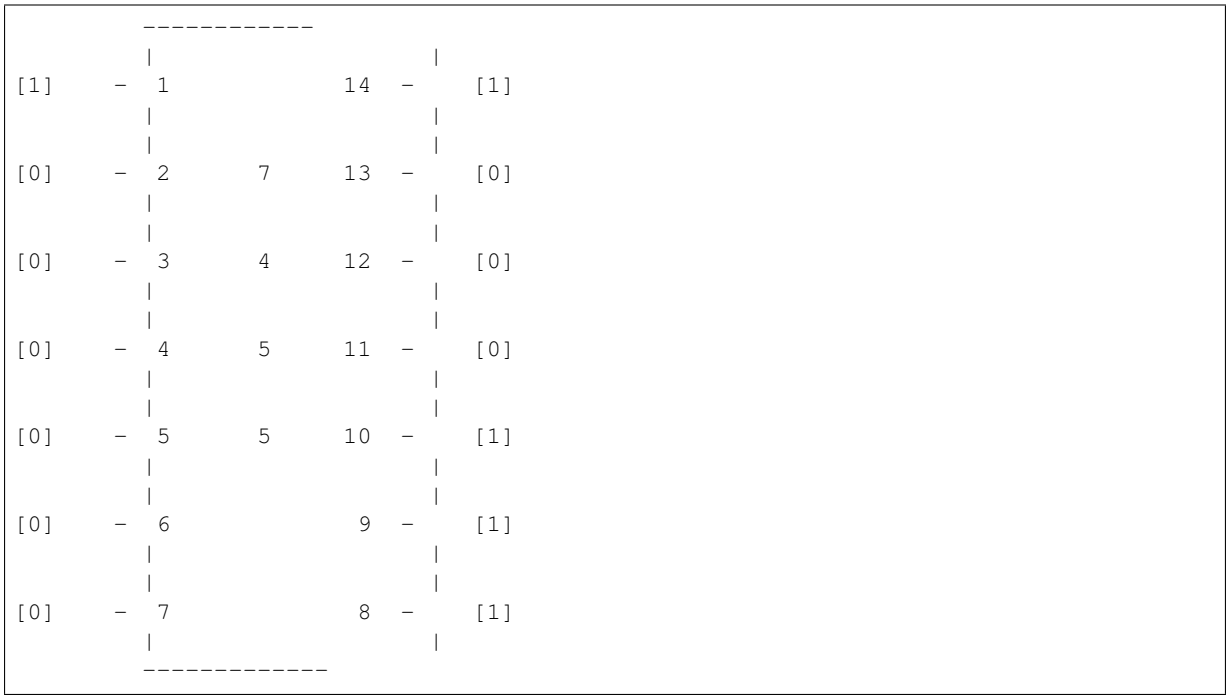
```
          ------------
          |                      |
[1]     -  1             14   -     [1]
          |                      |
          |                      |
[0]     -  2       7     13   -     [0]
          |                      |
          |                      |
[0]     -  3       4     12   -     [0]
          |                      |
          |                      |
[0]     -  4       5     11   -     [0]
          |                      |
          |                      |
[0]     -  5       5     10   -     [1]
          |                      |
          |                      |
[0]     -  6              9   -     [1]
          |                      |
          |                      |
[0]     -  7              8   -     [Z]
```

```
         |                    |
         -------------
```

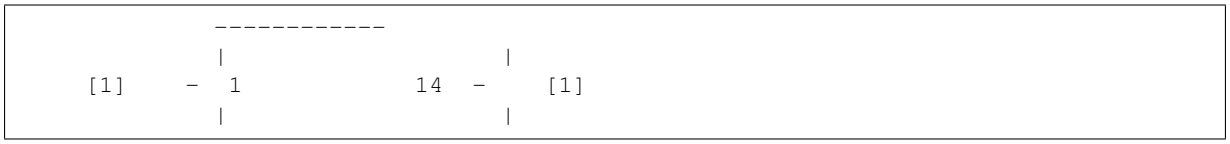
```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 1}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```




```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```



```
                |                       |
     [0]     -  2       7      13   -     [0]
                |                       |
                |                       |
     [0]     -  3       4      12   -     [0]
                |                       |
                |                       |
     [0]     -  4       5      11   -     [0]
                |                       |
                |                       |
     [0]     -  5       5      10   -     [1]
                |                       |
                |                       |
     [0]     -  6              9   -     [1]
                |                       |
                |                       |
     [0]     -  7              8   -     [1]
                |                       |
                --------------
{8: 1}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 1
```

Usage of IC 7458

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7458:

ic = IC_7458()

print(ic.__doc__)
```

```
This is a 2-input and 3-input AND-OR gate
```

```
# The Pin configuration is:

inp = {
    1: 1,
    2: 0,
    3: 0,
    4: 0,
    5: 0,
    7: 0,
    9: 0,
    10: 0,
    11: 0,
    12: 1,
    13: 1,
    14: 1}
```

```
# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                      |
[1]    -  1             14   -    [1]
          |                      |
          |                      |
[0]    -  2       7     13   -    [1]
          |                      |
          |                      |
[0]    -  3       4     12   -    [1]
          |                      |
          |                      |
[0]    -  4       5     11   -    [0]
          |                      |
          |                      |
[0]    -  5       8     10   -    [0]
          |                      |
          |                      |
[0]    -  6              9   -    [0]
          |                      |
          |                      |
[0]    -  7              8   -    [0]
          |                      |
          --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 1, 6: 0}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                      |
[1]    -  1             14   -    [1]
          |                      |
          |                      |
```

```
[0]    -  2       7      13  -    [1]
          |                    |
          |                    |
[0]    -  3       4      12  -    [1]
          |                    |
          |                    |
[0]    -  4       5      11  -    [0]
          |                    |
          |                    |
[0]    -  5       8      10  -    [0]
          |                    |
          |                    |
[0]    -  6              9   -    [0]
          |                    |
          |                    |
[0]    -  7              8   -    [1]
          |                    |
          --------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                    |
     [1]    -  1             14  -    [1]
               |                    |
               |                    |
     [0]    -  2       7     13  -    [1]
               |                    |
               |                    |
     [0]    -  3       4     12  -    [1]
               |                    |
               |                    |
     [0]    -  4       5     11  -    [0]
               |                    |
               |                    |
     [0]    -  5       8     10  -    [0]
               |                    |
               |                    |
     [0]    -  6              9   -    [0]
               |                    |
               |                    |
     [0]    -  7              8   -    [1]
               |                    |
               --------------
{8: 1, 6: 0}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)
```

```
print(c)
```

```
Connector; State: 1
```

Usage of IC 7464

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7464:

ic = IC_7464()

print(ic.__doc__)
```

```
This is a 4-2-3-2 input AND-OR-invert gate
```

```
# The Pin configuration is:

inp = {1: 1, 7: 0, 11: 1, 12: 1, 13: 1, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                  |
[1]     -  1             14  -    [1]
          |                  |
          |                  |
[0]     -  2       7     13  -    [1]
          |                  |
          |                  |
[0]     -  3       4     12  -    [1]
          |                  |
          |                  |
[0]     -  4       6     11  -    [1]
          |                  |
          |                  |
[0]     -  5       4     10  -    [0]
          |                  |
          |                  |
[0]     -  6              9  -    [0]
          |                  |
          |                  |
[0]     -  7              8  -    [Z]
          |                  |
          --------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 0}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                    |
[1]     -  1             14  -     [1]
          |                    |
          |                    |
[0]     -  2       7     13  -     [1]
          |                    |
          |                    |
[0]     -  3       4     12  -     [1]
          |                    |
          |                    |
[0]     -  4       6     11  -     [1]
          |                    |
          |                    |
[0]     -  5       4     10  -     [0]
          |                    |
          |                    |
[0]     -  6              9  -     [0]
          |                    |
          |                    |
[0]     -  7              8  -     [0]
          |                    |
          -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                    |
     [1]     -  1             14  -     [1]
               |                    |
               |                    |
     [0]     -  2       7     13  -     [1]
               |                    |
               |                    |
```

```
     [0]    -  3       4      12  -    [1]
               |                  |
               |                  |
     [0]    -  4       6      11  -    [1]
               |                  |
               |                  |
     [0]    -  5       4      10  -    [0]
               |                  |
               |                  |
     [0]    -  6              9   -    [0]
               |                  |
               |                  |
     [0]    -  7              8   -    [0]
               |                  |
              -------------
{8: 0}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 0
```

Usage of IC 7486

```
from __future__ import print_function
from BinPy import *
```

```
# Usage of IC 7486:

ic = IC_7486()

print(ic.__doc__)
```

```
This is a quad 2-input exclusive OR gate
```

```
# The Pin configuration is:

inp = {1: 1, 2: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 12: 0, 13: 0, 14: 1}

# Pin initinalization

# Powering up the IC - using -- ic.setIC({14: 1, 7: 0})

ic.setIC({14: 1, 7: 0})

# Setting the inputs of the ic

ic.setIC(inp)

# Draw the IC with the current configuration\n

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14  -     [1]
          |                     |
          |                     |
[0]     -  2       7     13  -     [0]
          |                     |
          |                     |
[Z]     -  3       4     12  -     [0]
          |                     |
          |                     |
[0]     -  4       8     11  -     [Z]
          |                     |
          |                     |
[0]     -  5       6     10  -     [1]
          |                     |
          |                     |
[Z]     -  6              9  -     [1]
          |                     |
          |                     |
[0]     -  7              8  -     [Z]
          |                     |
          -------------
```

```
# Run the IC with the current configuration using -- print ic.run() --

# Note that the ic.run() returns a dict of pin configuration similar to

print (ic.run())
```

```
{8: 0, 11: 0, 3: 1, 6: 0}
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --
↪\n

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()
```

```
          ------------
          |                     |
[1]     -  1             14  -     [1]
          |                     |
          |                     |
[0]     -  2       7     13  -     [0]
          |                     |
          |                     |
[1]     -  3       4     12  -     [0]
          |                     |
          |                     |
[0]     -  4       8     11  -     [0]
          |                     |
          |                     |
[0]     -  5       6     10  -     [1]
          |                     |
          |                     |
[0]     -  6              9  -     [1]
          |                     |
          |                     |
```

```
[0]    -  7             8  -    [0]
          |                 |
          -------------
```

```
# Seting the outputs to the current IC configuration using -- ic.setIC(ic.run()) --

ic.setIC(ic.run())

# Draw the final configuration

ic.drawIC()

# Run the IC

print (ic.run())
```

```
               ------------
               |                   |
     [1]    -  1             14  -    [1]
               |                   |
               |                   |
     [0]    -  2       7     13  -    [0]
               |                   |
               |                   |
     [1]    -  3       4     12  -    [0]
               |                   |
               |                   |
     [0]    -  4       8     11  -    [0]
               |                   |
               |                   |
     [0]    -  5       6     10  -    [1]
               |                   |
               |                   |
     [0]    -  6              9  -    [1]
               |                   |
               |                   |
     [0]    -  7              8  -    [0]
               |                   |
               -------------
{8: 0, 11: 0, 3: 1, 6: 0}
```

```
# Connector Outputs
c = Connector()

# Set the output connector to a particular pin of the ic
ic.setOutput(8, c)

print(c)
```

```
Connector; State: 0
```

Tools

Tools

Example to illustrate the usage of bittools

`BinPyBits` is a class inheriting from the `bitstring.BitArray` class. It will be used for efficient manipulation / handling of Bit vectors.

```
from BinPy import *

# Initializing a BinPyBits object

bit_vector = BinPyBits(5)
```

```
# By default all BinPyBits objects are not signed

bit_vector.signed
```

```
False
```

```
# Getting the decimal value

bit_vector.uint
```

```
5
```

```
# Getting the binary string

bit_vector.bin
```

```
'101'
```

```
# Do not use int with unsigned BinPyBits objects

bit_vector.int
```

```
-3
```

```
# This returns -3 since '101' ==> -3 ( 2's Complement representation )

# You could use :

int_value = bit_vector.int if bit_vector.signed else bit_vector.uint

print int_value
```

```
5
```

```
# Creating a BinPyBits object using binary string

bit_vector = BinPyBits('1111', signed=False)

# Converting to decimal

```

```
int_value = bit_vector.int if bit_vector.signed else bit_vector.uint

print int_value
```

```
15
```

```
# Creating a signed BinPyBits

bit_vector = BinPyBits('1111', signed=True)

# Converting to decimal

int_value = bit_vector.int if bit_vector.signed else bit_vector.uint

print int_value
```

```
-1
```

```
# Converting to hex

bit_vector.hex
```

```
'f'
```

Refer the documentation of bittstring to discover additional functionality.

```
# The speciality of BinPyBits lies in the fact that it can be initialized from␣
↪various types of inputs
# Except for the initialization, the rest of the functionalities remain similar to␣
↪that of the bitstring.BitArray

# Initializing a signed value using - sign

bit_vector = BinPyBits('-1111', signed=True)

print bit_vector.int
```

```
-15
```

Tools

Monostable Multivibrator - Multivibrator in Mode 1

```
from __future__ import print_function
from BinPy.tools.clock import Clock
from BinPy.Gates import Connector
from BinPy.tools.multivibrator import Multivibrator
from BinPy.tools.oscilloscope import Oscilloscope
import time
```

```
# MODE selects the mode of operation of the multivibrator.

# Mode No. :  Description
#    1          Monostable
#    2          Astable
#    3          Bistable
```

```
out = Connector()
```

```
# Initialize mutivibrator in MODE 1

m = Multivibrator(0, mode=1, time_period=1)
m.start()
m.setOutput(out)
```

```
# Initialize the oscilloscope
o = Oscilloscope((out, 'OUT'))
o.start()
o.setScale(0.05)  # Set scale by trial and error.
o.setWidth(75)
o.unhold()
time.sleep(0.1)
m.trigger()  # Also works with m()
time.sleep(0.1)
```

```
[0m
[0m
```

```
# Display the oscilloscope
o.display()
```

```
[0m==========================================================================================
BinPy - Oscilloscope
==========================================================================================
                                                      SCALE - X-AXIS : 1 UNIT WIDTH = 0.05
==========================================================================================
          |
          |
          |  -----
     OUT  |  |        |
          - -        -------------------------------------------
          |
          |
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
------------------------------------------------------------
[0m
```

```
# Kill the multivibrator and the oscilloscope threads
m.kill()
o.kill()
```

Astable Multivibrator - Multivibrator in Mode 2

```
from __future__ import print_function
from BinPy.tools.clock import Clock
from BinPy.Gates import Connector
from BinPy.tools.multivibrator import Multivibrator
from BinPy.tools.oscilloscope import Oscilloscope
import time
```

```
# MODE selects the mode of operation of the multivibrator.

# Mode No. :  Description
```

```
#   1           Monostable
#   2           Astable
#   3           Bistable

out = Connector()
```

```
# Initialize mutivibrator in MODE 2 with the adequate on_time and off_time

m = Multivibrator(0, mode = 2, on_time=0.2, off_time=0.8)
m.start()
m.setOutput(out)
```

```
# Initialize the oscilloscope
o = Oscilloscope((out, 'OUT'))
o.start()
o.setScale(0.07)  # Set scale by trial and error.
o.setWidth(75)
o.unhold()
time.sleep(0.2)
m.trigger()  # Also works with m()
time.sleep(5)
```

```
[0m
[0m
```

```
# Display the oscilloscope
o.display()
```

```
[0m==========================================================================================
BinPy - Oscilloscope
==========================================================================================
                                                      SCALE - X-AXIS : 1 UNIT WIDTH = 0.07
==========================================================================================
          |
          |
          |   _           _           _          _          _
     OUT  |   |  |           | |           |  |          |  |          |  |
          - -  ------- -------  -------  -------  --------
          |
          |
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
------------------------------------------------------------
[0m
```

```
# Kill the multivibrator and the oscilloscope threads
m.kill()
o.kill()
```

Bistable Multivibrator - Multivibrator in Mode 3

```
from __future__ import print_function
from BinPy.tools.clock import Clock
from BinPy.Gates import Connector
from BinPy.tools.multivibrator import Multivibrator
from BinPy.tools.oscilloscope import Oscilloscope
import time
```

```
# MODE selects the mode of operation of the multivibrator.

# Mode No. :  Description
#   1          Monostable
#   2          Astable
#   3          Bistable

out = Connector(0)
```

```
# Initialize mutivibrator in MODE 3

m = Multivibrator(0, mode = 3)
m.start()
m.setOutput(out)
```

```
# Initialize the oscilloscope
o = Oscilloscope((out, 'OUT'))
o.start()
o.setScale(0.1)
o.setWidth(75)
o.unhold()
time.sleep(0.001) # This is done to let the oscilloscope thread to synchronize␣
↪with the main thread...
```

```
[0m
[0m
```

```
# Trigger the multivibrator to change the state
print(out())
time.sleep(0.1)
m.trigger()

time.sleep(0.001) # This is done to synchronize the multivibrator thread ...

print(out())
time.sleep(0.5)
m.trigger()

time.sleep(0.001) # This is done to synchronize the multivibrator thread ...

print(out())
time.sleep(1)
m.trigger()

time.sleep(0.001) # This is done to synchronize the multivibrator thread ...

print(out())
time.sleep(2)
m.trigger()

time.sleep(0.001) # This is done to synchronize the multivibrator thread ...

print (out())
```

```
0
1
0
1
0
```

```
# Display the oscilloscope
o.display()
```

```
[0m================================================================================================
BinPy - Oscilloscope
================================================================================================
                                                          SCALE - X-AXIS : 1 UNIT␣
↪WIDTH = 0.1
================================================================================================
           |
           |
           |   ---          -------------
     OUT   |   |    |          |                    |
           - -    ------                  --------------------------
           |
           |
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| |||||
------------------------------------------------------------
[0m
```

```
# Kill the multivibrator and the oscilloscope threads
m.kill()
o.kill()
```

CHAPTER 3

BinPy

Subpackages

Algorithms

Submodules

- AnalogFormulas
- ExpressionConvert
- ImplementBooleanFunction
- MooreOptimizer
- QuineMcCluskey

AnalogFormulas

ExpressionConvert

ImplementBooleanFunction

MooreOptimizer

QuineMcCluskey

Module contents

Analog

Submodules

- base
- source

base

source

Module contents

Combinational

Submodules

- combinational

combinational

Module contents

Gates

Submodules

- connector
- gates
- tree

connector

gates

tree

Module contents

Operations

Submodules

- operations

operations

Module contents

Sequential

Submodules

- counters
- registers
- sequential

counters

registers

sequential

Module contents

dev

Submodules

- parseEquation

parseEquation

class BinPy.dev.parseEquation.**Expr**(*equation*, **var*)

This class is used to parse any expression which contain boolean variables. Input String can be in the form of logical operators which can be parsed to Gates by this class. This is also used to obtain the truth tables"

Logical Operator form: Function takes only equation as an input. Gates Form: Needs The variable inputs also as an argument. .. rubric:: Examples

```
>>> from BinPy import *
>>> expr = Expr('A & B | C')
>>> expr.parse()
'AND(OR(C,B),A)'
>>> expr.truthTable()
A B C O
0 0 0 0
0 0 1 0
0 1 0 0
0 1 1 0
1 0 0 0
1 0 1 1
1 1 0 1
1 1 1 1
>>> expr = Expr('AND(NOT(A), B)', 'A', 'B')
>>> expr.parse()
'AND(NOT(A),B)'
>>> expr.truthTable()
A B O
0 0 0
0 1 1
1 0 0
1 1 0
```

eqnParse(*eqn*, *isOperandtype=<method 'isalpha' of 'str' objects>*)

findMatchingBrace(*position*, *string*)

Returns the index of the opposite matching brace for the brace at string[position]

parse()

removeBraces(*position*, *equation*)

Removes braces due to clubbing of the gates position indicates the index of the clubbed gate

truthTable()

Module contents

ic

Submodules

- base
- series_4000
- series_7000

base

This module includes all the base classes for different ICs.

class BinPy.ic.base.**Base_14pin**
Bases: *BinPy.ic.base.IC*

This method takes base class for IC's having 14 pins

set_pin(*pin_no*, *pin_value*)

set_pin_param(*pin_no*, *parm_dict*)

total_pins = 14

uses_pincls = False

class BinPy.ic.base.**Base_16pin**
Bases: *BinPy.ic.base.IC*

This method takes base class for IC's having 16 pins

set_pin(*pin_no*, *pin_value*)

set_pin_param(*pin_no*, *parm_dict*)

total_pins = 16

uses_pincls = False

class BinPy.ic.base.**Base_24pin**
Bases: *BinPy.ic.base.IC*

This method takes base class for IC's having 24 pins

set_pin(*pin_no*, *pin_value*)

set_pin_param(*pin_no*, *parm_dict*)

total_pins = 24

uses_pincls = False

class BinPy.ic.base.**Base_5pin**
Bases: *BinPy.ic.base.IC*

This method takes base class for IC's having 5 pins

set_pin(*pin_no*, *pin_value*)

total_pins = 5

uses_pincls = False

class BinPy.ic.base.**IC**
This is a base class for IC

draw_IC()

output_connector = {}

set_IC(*param_dict*)
If pin class is not used this method then it takes a dictionary with the format { PINNO:PINVALUE, ... } Else it takes a dictionary of dictionaries with the format -> { PINNO:{PARAM1:VAL1, PARAM2:VAL2, ... }, PINNO2:{PARAM1:VAL1, PARAM2:VAL2, ... } , ... }

set_output(*index*, *value*)

truth_table(*pin_config*)

class BinPy.ic.base.**Logic**(*value=0*)
Implements methods of AND OR and EXOR using BinPy library Gate modules Remaps all basic python implementation of gates on variable of type bool to BinPy's implementation of the same

class BinPy.ic.base.**Pin**(*pin_no*, *param_dict={}*)
Pin class for defining a particular pin of an IC

Sample param_dict for a pin : { 'value':0, 'desc':'IN1: Input 1 of Mux', 'can_vary':True }

First 3 characters of desc will be used as pin_tag

set_pin_param(*param_dict*)

BinPy.ic.base.**pinlist_quick**(*first_arg*)
Defines a method to quickly convert a list of Logic states to pin instances

series_4000

This module has all the classes of ICs belonging to 4000 series.

Please note that the length of list self.pins is 1 more than the number of actual pins. This is so because pin0 is not used as a general term referring to the first pin of the IC. Zeroth index of the self.pins is not being used.

ICs in this module: [4000, 4001, 4002, 4008, 4009, 4010, 4011, 4012, 4013, 4015, 4017, 4019, 4020, 4023, 4025, 4068, 4069, 4070, 4071, 4072, 4073

4075, 4077, 4078, 4081, 4082]

class BinPy.ic.series_4000.**IC_4000**
Bases: *BinPy.ic.base.Base_14pin*

Dual 3 Input NOR gate + one NOT gate IC. Pin_6 = NOR(Pin_3, Pin_4, Pin_5) Pin_10 = NOR(Pin_11, Pin_12, Pin_13) Pin_9 = NOT(Pin_8)

run()

class BinPy.ic.series_4000.**IC_4001**
Bases: *BinPy.ic.base.Base_14pin*

Quad 2 input NOR gate Pin_3 = NOR(Pin_1, Pin_2) Pin_4 = NOR(Pin_5, Pin_6) Pin_10 = NOR(Pin_8, Pin_9) Pin_11 = NOR(Pin_12, Pin_13)

run()

class BinPy.ic.series_4000.**IC_4002**
Bases: *BinPy.ic.base.Base_14pin*

Dual 4 input NOR gate Pin_1 = NOR(Pin_2, Pin_3, Pin_4, Pin_5) Pin_13 = NOR(Pin_9, Pin_10, Pin_11, Pin_12)

run()

class BinPy.ic.series_4000.**IC_4008**
Bases: *BinPy.ic.base.Base_16pin*

4 Bit Binary Full Adder

run()

class BinPy.ic.series_4000.**IC_4009**
Bases: *BinPy.ic.base.Base_16pin*

Hex Inverter with Level Shifted output

run()

class BinPy.ic.series_4000.**IC_4010**
Bases: *BinPy.ic.base.Base_16pin*

Hex Buffer with Level Shifted output

run()

class BinPy.ic.series_4000.**IC_4011**
Bases: *BinPy.ic.base.Base_14pin*

Quad 2 input NAND gate Pin_3 = NAND(Pin_1, Pin_2) Pin_4 = NAND(Pin_5, Pin_6) Pin_10 = NAND(Pin_8, Pin_9) Pin_11 = NAND(Pin_12, Pin_13)

run()

class BinPy.ic.series_4000.**IC_4012**
Bases: *BinPy.ic.base.Base_14pin*

Dual 4 input NAND gate Pin_1 = NAND(Pin_2, Pin_3, Pin_4, Pin_5) Pin_13 = NAND(Pin_9, Pin_10, Pin_11, Pin_12)

run()

class BinPy.ic.series_4000.**IC_4013**
Bases: *BinPy.ic.base.Base_14pin*

CMOS Dual D type Flip Flop

run()

class BinPy.ic.series_4000.**IC_4015**
Bases: *BinPy.ic.base.Base_16pin*

Dual 4 Stage static shift Register

run()

class BinPy.ic.series_4000.**IC_4017**
Bases: *BinPy.ic.base.Base_16pin*

CMOS Counters

run()

class BinPy.ic.series_4000.**IC_4019**
Bases: *BinPy.ic.base.Base_16pin*

8-to-4 line non-inverting data selector/multiplexer with OR function

run()

class BinPy.ic.series_4000.**IC_4020**
Bases: *BinPy.ic.base.Base_16pin*

CMOS 14 BIT asynchornous binary counter with reset

run()

class BinPy.ic.series_4000.**IC_4022**
Bases: *BinPy.ic.base.Base_16pin*

CMOS Octal Counter

run()

class BinPy.ic.series_4000.**IC_4023**
Bases: *BinPy.ic.base.Base_14pin*

Triple 3 input NAND gate Pin_6 = NAND(Pin_3, Pin_4, Pin_5) Pin_9 = NAND(Pin_1, Pin_2, Pin_8) Pin_10 = NAND(Pin_11, Pin_12, Pin_13)

run()

class BinPy.ic.series_4000.**IC_4025**
Bases: *BinPy.ic.base.Base_14pin*

Triple 3 input NOR gate Pin_6 = NOR(Pin_3, Pin_4, Pin_5) Pin_9 = NOR(Pin_1, Pin_2, Pin_8) Pin_10 = NOR(Pin_11, Pin_12, Pin_13)

run()

class BinPy.ic.series_4000.**IC_4027**
Bases: *BinPy.ic.base.Base_16pin*

Dual JK flip flops with set and reset

run()

class BinPy.ic.series_4000.**IC_4028**
Bases: *BinPy.ic.base.Base_16pin*

1-of-10 no-inverting decoder/demultiplexer

run()

class BinPy.ic.series_4000.**IC_4029**
Bases: *BinPy.ic.base.Base_16pin*

4-bit synchronous binary/decade up/down counter

arraytoint(*inputs*)

run()

class BinPy.ic.series_4000.**IC_4030**
Bases: *BinPy.ic.base.Base_14pin*

Quad 2-input XOR gate

run()

class BinPy.ic.series_4000.**IC_4068**
Bases: *BinPy.ic.base.Base_14pin*

8 input NAND gate Pin_13 = NAND(Pin_2, Pin_3, Pin_4, Pin_5, Pin_9, Pin_10, Pin_11, Pin_12)

run()

class BinPy.ic.series_4000.**IC_4069**
Bases: *BinPy.ic.base.Base_14pin*

Hex NOT gate Pin_2 = NOT(Pin_1) Pin_4 = NOT(Pin_3) Pin_6 = NOT(Pin_5) Pin_8 = NOT(Pin_9) Pin_10 = NOT(Pin_11) Pin_12 = NOT(Pin_13)

run()

class BinPy.ic.series_4000.**IC_4070**
Bases: *BinPy.ic.base.Base_14pin*

Quad 2 input XOR gate Pin_3 = XOR(Pin_1, Pin_2) Pin_4 = XOR(Pin_5, Pin_6) Pin_10 = XOR(Pin_8, Pin_9) Pin_11 = XOR(Pin_12, Pin_13)

run()

class BinPy.ic.series_4000.**IC_4071**
Bases: *BinPy.ic.base.Base_14pin*

Quad 2 input OR gate Pin_3 = OR(Pin_1, Pin_2) Pin_4 = OR(Pin_5, Pin_6) Pin_10 = OR(Pin_8, Pin_9) Pin_11 = OR(Pin_12, Pin_13)

run()

class BinPy.ic.series_4000.**IC_4072**
Bases: *BinPy.ic.base.Base_14pin*

Dual 4 input OR gate Pin_1 = OR(Pin_2, Pin_3, Pin_4, Pin_5) Pin_13 = OR(Pin_9, Pin_10, Pin_11, Pin_12)

run()

class BinPy.ic.series_4000.**IC_4073**
Bases: *BinPy.ic.base.Base_14pin*

Triple 3 input AND gate Pin_6 = AND(Pin_3, Pin_4, Pin_5) Pin_9 = AND(Pin_1, Pin_2, Pin_8) Pin_10 = AND(Pin_11, Pin_12, Pin_13)

run()

class BinPy.ic.series_4000.**IC_4075**
Bases: *BinPy.ic.base.Base_14pin*

Triple 3 input OR gate Pin_6 = OR(Pin_3, Pin_4, Pin_5) Pin_9 = OR(Pin_1, Pin_2, Pin_8) Pin_10 = OR(Pin_11, Pin_12, Pin_13)

run()

class BinPy.ic.series_4000.**IC_4077**
Bases: *BinPy.ic.base.Base_14pin*

Quad 2 input XNOR gate Pin_3 = XNOR(Pin_1, Pin_2) Pin_4 = XNOR(Pin_5, Pin_6) Pin_10 = XNOR(Pin_8, Pin_9) Pin_11 = XNOR(Pin_12, Pin_13)

run()

class BinPy.ic.series_4000.**IC_4078**
Bases: *BinPy.ic.base.Base_14pin*

8 input NOR gate Pin_13 = NOR(Pin_2, Pin_3, Pin_4, Pin_5, Pin_9, Pin_10, Pin_11, Pin_12)

run()

class BinPy.ic.series_4000.**IC_4081**
Bases: *BinPy.ic.base.Base_14pin*

Quad 2 input AND gate Pin_3 = AND(Pin_1, Pin_2) Pin_4 = AND(Pin_5, Pin_6) Pin_10 = AND(Pin_8, Pin_9) Pin_11 = AND(Pin_12, Pin_13)

run()

class BinPy.ic.series_4000.**IC_4082**
Bases: *BinPy.ic.base.Base_14pin*

Dual 4 input AND gate Pin_1 = AND(Pin_2, Pin_3, Pin_4, Pin_5) Pin_13 = AND(Pin_9, Pin_10, Pin_11, Pin_12)

run()

series_7400

This module has all the classes of ICs belonging to 7400 series.

Please note that the length of list self.pins is 1 more than the number of actual pins. This is so because pin0 is not used as a general term referring to the first pin of the IC. Zeroth index of the self.pins is not being used.

class BinPy.ic.series_7400.**IC_7400**
Bases: *BinPy.ic.base.Base_14pin*

This is a QUAD 2 INPUT NAND gate IC Pin Configuration:

Pin Number Description 1 A Input Gate 1 2 B Input Gate 1 3 Y Output Gate 1 4 A Input Gate 2 5 B Input Gate 2 6 Y Output Gate 2 7 Ground 8 Y Output Gate 3 9 B Input Gate 3 10 A Input Gate 3 11 Y Output Gate 4 12 B Input Gate 4 13 A Input Gate 4 14 Positive Supply

This class needs 14 parameters. Each parameter being the pin value. The input has to be defined as a dictionary with pin number as the key and its value being either 1 or 0

To initialise the ic 7400:

1. set pin 7:0
2. set pin 14:1

How to use:

```
>>> ic = IC_7400()
>>> pin_config = {1: 1, 2: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 12: 0, 13: 0, 14:␣
↪1}
>>> ic.set_IC(pin_cofig)
>>> ic.drawIC()
>>> ic.run()
>>> ic.set_IC(ic.run())
>>> ic.drawIC()
```

pins = [None,0,0,None,0,0,None,0,None,0,0,None,0,0,0]

run()

class BinPy.ic.series_7400.**IC_7401**
Bases: *BinPy.ic.base.Base_14pin*

This is a Quad 2-input open-collector NAND gate IC

run()

class BinPy.ic.series_7400.**IC_7402**
Bases: *BinPy.ic.base.Base_14pin*

This is a Quad 2-input NOR gate IC

Pin Configuration:

Pin Number Description 1 Y Output Gate 1 2 A Input Gate 1 3 B Input Gate 1 4 Y Output Gate 2 5 A Input Gate 2 6 B Input Gate 2 7 Ground 8 A Input Gate 3 9 B Input Gate 3 10 Y Output Gate 3 11 A Input Gate 4 12 B Input Gate 4 13 Y Output Gate 4 14 Positive Supply

This class needs 14 parameters. Each parameter being the pin value. The input has to be defined as a dictionary with pin number as the key and its value being either 1 or 0

To initialise the ic 7402:

1. set pin 7:0
2. set pin 14:1

How to use:

```
>>> ic = IC_7402()
>>> pin_config = {2: 0, 3: 0, 5: 0, 6: 1, 7: 0, 8: 1, 9: 1, 11: 1, 12: 1, 14:␣
↪1}
>>> ic.set_IC(pin_cofig)
>>> ic.drawIC()
>>> ic.run()
>>> ic.set_IC(ic.run())
>>> ic.drawIC()
```

Default pins: pins = [None,None,0,0,None,0,0,0,0,0,None,0,0,None,0]

run()

class BinPy.ic.series_7400.**IC_7403**
Bases: *BinPy.ic.base.Base_14pin*

This is a Quad 2-input open-collector NAND gate IC

Pin Number Description 1 A Input Gate 1 2 B Input Gate 1 3 Y Output Gate 1 4 A Input Gate 2 5 B Input Gate 2 6 Y Output Gate 2 7 Ground 8 Y Output Gate 3 9 B Input Gate 3 10 A Input Gate 3 11 Y Output Gate 4 12 B Input Gate 4 13 A Input Gate 4 14 Positive Supply

This class needs 14 parameters. Each parameter being the pin value. The input has to be defined as a dictionary with pin number as the key and its value being either 1 or 0

To initialise the ic 7403:

1. set pin 7:0
2. set pin 14:1

How to use:

```
>>> ic = IC_7403()
>>> pin_config = {1: 1, 2: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 12: 0, 13: 0, 14:␣
↪1}
>>> ic.set_IC(pin_cofig)
>>> ic.drawIC()
>>> ic.run()
>>> ic.set_IC(ic.run())
>>> ic.drawIC()
```

Default pins: pins = [None,0,0,None,0,0,None,0,None,0,0,None,0,0,0]

run()

class BinPy.ic.series_7400.**IC_7404**
Bases: *BinPy.ic.base.Base_14pin*

This is a hex inverter IC

Pin Number Description 1 A Input Gate 1 2 Y Output Gate 1 3 A Input Gate 2 4 Y Output Gate 2 5 A Input Gate 3 6 Y Output Gate 3 7 Ground 8 Y Output Gate 4 9 A Input Gate 4 10 Y Output Gate 5 11 A Input Gate 5 12 Y Output Gate 6 13 A Input Gate 6 14 Positive Supply

This class needs 14 parameters. Each parameter being the pin value. The input has to be defined as a dictionary with pin number as the key and its value being either 1 or 0

To initialise the ic 7404:

1. set pin 7:0
2. set pin 14:1

How to use:

```
>>> ic = IC_7404()
>>> pin_config = {1: 1, 3: 0, 5: 0, 7: 0, 9: 0, 11: 0, 13: 1, 14: 1}
>>> ic.set_IC(pin_cofig)
>>> ic.drawIC()
>>> ic.run()
>>> ic.set_IC(ic.run())
>>> ic.drawIC()
```

Default pins: pins = [None,0,0,None,0,0,None,0,None,0,0,None,0,0,0]

run()

class BinPy.ic.series_7400.**IC_7405**
Bases: *BinPy.ic.base.Base_14pin*

This is hex open-collector inverter IC

run()

class BinPy.ic.series_7400.**IC_7406**
Bases: *BinPy.ic.base.Base_14pin*

This is Hex Inverter/Buffer with Hi-Volt Open Collector Output

run()

class BinPy.ic.series_7400.**IC_7408**
Bases: *BinPy.ic.base.Base_14pin*

This is a Quad 2 input AND gate IC

Pin Number Description 1 A Input Gate 1 2 B Input Gate 1 3 Y Output Gate 1 4 A Input Gate 2 5 B Input Gate 2 6 Y Output Gate 2 7 Ground 8 Y Output Gate 3 9 B Input Gate 3 10 A Input Gate 3 11 Y Output Gate 4 12 B Input Gate 4 13 A Input Gate 4 14 Positive Supply

This class needs 14 parameters. Each parameter being the pin value. The input has to be defined as a dictionary with pin number as the key and its value being either 1 or 0

To initialise the ic 7408:

1. set pin 7:0
2. set pin 14:1

How to use:

```
>>> ic = IC_7408()
>>> pin_config = {1: 1, 2: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 12: 0, 13: 0, 14:␣
↪1}
>>> ic.set_IC(pin_cofig)
>>> ic.drawIC()
>>> ic.run()
>>> ic.set_IC(ic.run())
>>> ic.drawIC()
```

Default pins: pins = [None,0,0,None,0,0,None,0,None,0,0,None,0,0,0]

run()

class BinPy.ic.series_7400.**IC_7410**
Bases: *BinPy.ic.base.Base_14pin*

This is a Triple 3 input NAND gate IC

Pin Number Description 1 A Input Gate 1 2 B Input Gate 1 3 A Input Gate 2 4 B Input Gate 2 5 C Input gate 2 6 Y Output Gate 2 7 Ground 8 Y Output Gate 3 9 A Input Case 3 10 B Input Case 3 11 C Input Case 3 12 Y Output Gate 1 13 C Input Gate 1 14 Positive Supply

This class needs 14 parameters. Each parameter being the pin value. The input has to be defined as a dictionary with pin number as the key and its value being either 1 or 0

To initialise the ic 7410:

1. set pin 7:0
2. set pin 14:1

How to use:

```
>>> ic = IC_7410()
>>> pin_config = {1: 1, 2: 0, 3: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 11: 1, 13:␣
↪0, 14: 1}
>>> ic.set_IC(pin_cofig)
>>> ic.drawIC()
>>> ic.run()
>>> ic.set_IC(ic.run())
>>> ic.drawIC()
```

Default pins: pins = [None,0,0,0,0,0,None,0,None,0,0,0,None,0,0]

run()

class BinPy.ic.series_7400.**IC_7411**
Bases: *BinPy.ic.base.Base_14pin*

This is a Triple 3 input AND gate IC

Pin Number Description 1 A Input Gate 1 2 B Input Gate 1 3 A Input Gate 2 4 B Input Gate 2 5 C Input gate 2 6 Y Output Gate 2 7 Ground 8 Y Output Gate 3 9 A Input Case 3 10 B Input Case 3 11 C Input Case 3 12 Y Output Gate 1 13 C Input Gate 1 14 Positive Supply

This class needs 14 parameters. Each parameter being the pin value. The input has to be defined as a dictionary with pin number as the key and its value being either 1 or 0

To initialise the ic 7411:

1. set pin 7:0
2. set pin 14:1

How to use:

```
>>> ic = IC_7411()
>>> pin_config = {1: 1, 2: 0, 3: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 11: 1, 13:␣
↪0, 14: 1}
>>> ic.set_IC(pin_cofig)
>>> ic.drawIC()
>>> ic.run()
>>> ic.set_IC(ic.run())
>>> ic.drawIC()
```

Default pins: pins = [None,0,0,0,0,0,None,0,None,0,0,0,None,0,0]

run()

class BinPy.ic.series_7400.**IC_7412**
Bases: *BinPy.ic.base.Base_14pin*

This is a Triple 3 input NAND gate IC with open collector outputs

Pin Number Description 1 A Input Gate 1 2 B Input Gate 1 3 A Input Gate 2 4 B Input Gate 2 5 C Input gate 2 6 Y Output Gate 2 7 Ground 8 Y Output Gate 3 9 A Input Case 3 10 B Input Case 3 11 C Input Case 3 12 Y Output Gate 1 13 C Input Gate 1 14 Positive Supply

This class needs 14 parameters. Each parameter being the pin value. The input has to be defined as a dictionary with pin number as the key and its value being either 1 or 0

To initialise the ic 7412:

1. set pin 7:0
2. set pin 14:1

How to use:

```
>>> ic = IC_7412()
>>> pin_config = {1: 1, 2: 0, 3: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 11: 1, 13:␣
↪0, 14: 1}
>>> ic.set_IC(pin_cofig)
>>> ic.drawIC()
>>> ic.run()
>>> ic.set_IC(ic.run())
>>> ic.drawIC()
```

Default pins: pins = [None,0,0,0,0,0,None,0,None,0,0,0,None,0,0]

run()

class BinPy.ic.series_7400.**IC_7413**
Bases: *BinPy.ic.base.Base_14pin*

This is a dual 4 input NAND gate IC

Pin Number Description 1 A Input Gate 1 2 B Input Gate 1 3 Not Connected 4 C Input Gate 1 5 D Input Gate 1 6 Y Output Gate 1 7 Ground 8 Y Output Gate 2 9 A Input Gate 2 10 B Input Gate 2 11 Not Connected 12 C Input Gate 2 13 D Input Gate 2 14 Positive Supply

This class needs 14 parameters. Each parameter being the pin value. The input has to be defined as a dictionary with pin number as the key and its value being either 1 or 0

To initialise the ic 7413:

1. set pin 7:0
2. set pin 14:1

How to use:

```
>>> ic = IC_7413()
>>> pin_config = {1: 1, 2: 0, 4: 0, 5: 0, 7: 0, 9: 1, 10: 1, 12: 1, 13: 1, 14:␣
↪1}
>>> ic.set_IC(pin_cofig)
>>> ic.drawIC()
>>> ic.run()
>>> ic.set_IC(ic.run())
>>> ic.drawIC()
```

Default pins: pins = [None,0,0,0,0,0,None,0,None,0,0,0,0,0,0]

run()

class BinPy.ic.series_7400.**IC_74133**
Bases: *BinPy.ic.base.Base_16pin*

This is a 13-input NAND gate

run()

class BinPy.ic.series_7400.**IC_74138**
Bases: *BinPy.ic.base.Base_16pin*

This is a 1:8 demultiplexer(3:8 decoder) with output being inverted input

run()

class BinPy.ic.series_7400.**IC_74139**
Bases: *BinPy.ic.base.Base_16pin*

This is a dual 1:4 demultiplexer(2:4 decoder) with output being inverted input

run()

class BinPy.ic.series_7400.**IC_7415**
Bases: *BinPy.ic.base.Base_14pin*

This is a Triple 3 input AND gate IC with open collector outputs

Pin Number Description 1 A Input Gate 1 2 B Input Gate 1 3 A Input Gate 2 4 B Input Gate 2 5 C Input Gate 2 6 Y Output Gate 2 7 Ground 8 Y Output Gate 3 9 A Input Gate 3 10 B Input Gate 3 11 C Input Gate 3 12 Y Output Gate 1 13 C Input Gate 1 14 Positive Supply

This class needs 14 parameters. Each parameter being the pin value. The input has to be defined as a dictionary with pin number as the key and its value being either 1 or 0

To initialise the ic 7415:

1. set pin 7:0
2. set pin 14:1

How to use:

```
>>> ic = IC_7415()
>>> pin_config = {1:1, 2:0, 3:0, 4:0, 5:0, 7:0, 9:1, 10:1, 11:1, 13:0, 14:1}
>>> ic.set_IC(pin_cofig)
>>> ic.drawIC()
>>> ic.run()
>>> ic.set_IC(ic.run())
>>> ic.drawIC()
```

Default pins: pins = [None,0,0,0,0,0,None,0,None,0,0,0,None,0,0]

run()

class BinPy.ic.series_7400.**IC_74151A**
Bases: *BinPy.ic.base.Base_16pin*

This is 16-pin 8:1 multiplexer featuring complementary W and Y outputs

run()

class BinPy.ic.series_7400.**IC_74152**
Bases: *BinPy.ic.base.Base_14pin*

This is 14-pin 8:1 multiplexer with inverted input.

Pin Number Description 1 D4 2 D3 3 D2 4 D1 5 D0 6 Output W 7 Ground 8 select line C 9 select line B 10 select line A 11 D7 12 D6 13 D5 14 Positive Supply

select_lines = CBA and Inputlines = D0 D1 D2 D3 D4 D5 D6 D7

run()

class BinPy.ic.series_7400.**IC_74153**
Bases: *BinPy.ic.base.Base_16pin*

This is 16-pin dual 4:1 multiplexer with output same as the input.

Pin Number Description 1 Strobe1 2 Select line B 3 1C3 4 1C2 5 1C1 6 1C0 7 1Y - OUTPUT1 8 Ground 9 2Y - OUTPUT2 10 2C0 11 2C1 12 2C2 13 2C3 14 Select line A 15 Strobe2 16 Positive Supply

select_lines = BA ; Inputlines1 = 1C0 1C1 1C2 1C3 ; Inputlines2 = 2C0 2C1 2C2 2C3

run()

class BinPy.ic.series_7400.**IC_74155**
Bases: *BinPy.ic.base.Base_16pin*

This is a dual 1:4 demultiplexer(2:4 decoder) with one output being inverted input while the other same as the input

run()

class BinPy.ic.series_7400.**IC_74156**
Bases: *BinPy.ic.base.Base_16pin*

This is a dual 1:4 demultiplexer(2:4 decoder) with one output being inverted input while the other same as the input with open collector

run()

class BinPy.ic.series_7400.**IC_7416**
Bases: *BinPy.ic.base.Base_14pin*

This is a Hex open-collector high-voltage inverter

run()

class BinPy.ic.series_7400.**IC_7417**
Bases: *BinPy.ic.base.Base_14pin*

This is a Hex open-collector high-voltage buffer

run()

class BinPy.ic.series_7400.**IC_7418**
Bases: *BinPy.ic.base.Base_14pin*

This is a Dual 4-input NAND gates with schmitt-trigger inputs.

run()

class BinPy.ic.series_7400.**IC_74181**
Bases: *BinPy.ic.base.Base_24pin*

This is a 4-bit Arithmetic Logic Unit which performs 16 diff functions. It has two modes active high input mode and active low input mode(Active high mode is used here)

Pin Number Description 1 Input - B0 2 Input - A0 3 Input - Select Line - S3 4 Input - Select Line - S2 5 Input - Select Line - S1 6 Input - Select Line - S0 7 Input - Carry 8 Input - Mode Input(M) 9 Output- F0 10 Output- F1 11 Output- F2 12 Ground 13 Output- F3 14 Output- A=B 15 Output- P 16 Output- NOT(C(n+4)) 17 Output- G 18 Input - B3 19 Input - A3 20 Input - B2 21 Input - A2 22 Input - B1 23 Input - A1 24 VCC

Mode and Select Lines are used to select the function to be performed by the ALU on the two 4-bit input data A3 A2 A1 A0 & B3 B2 B1 B0(Inputs A0-A3 and B0-B3 have to be complemented and given).

run()

class BinPy.ic.series_7400.**IC_7419**
Bases: *BinPy.ic.base.Base_14pin*

This is a Hex inverters with schmitt-trigger line-receiver inputs.

run()

class BinPy.ic.series_7400.**IC_741G00**
Bases: *BinPy.ic.base.Base_5pin*

This is a single 2 input NAND gate IC

run()

class BinPy.ic.series_7400.**IC_741G02**
Bases: *BinPy.ic.base.Base_5pin*

This is a single 2 input NOR gate IC

run()

class BinPy.ic.series_7400.**IC_741G03**
Bases: *BinPy.ic.base.Base_5pin*

This is a single 2 input NAND gate IC

run()

class BinPy.ic.series_7400.**IC_741G04**
Bases: *BinPy.ic.base.Base_5pin*

This is a single inverter IC

run()

class BinPy.ic.series_7400.**IC_741G05**
Bases: *BinPy.ic.base.Base_5pin*

This is a single input NOT gate IC

run()

class BinPy.ic.series_7400.**IC_741G08**
Bases: *BinPy.ic.base.Base_5pin*

This is a single 2 input AND gate IC

run()

class BinPy.ic.series_7400.**IC_7420**
Bases: *BinPy.ic.base.Base_14pin*

This is a dual 4-input NAND gate

run()

class BinPy.ic.series_7400.**IC_7421**
Bases: *BinPy.ic.base.Base_14pin*

This is a dual 4-input AND gate

run()

class BinPy.ic.series_7400.**IC_7422**
Bases: *BinPy.ic.base.Base_14pin*

This is a dual 4-input NAND gate with open collector outputs

run()

class BinPy.ic.series_7400.**IC_7424**
Bases: *BinPy.ic.base.Base_14pin*

This is a Quad 2-input NAND gates with schmitt-trigger line-receiver inputs

run()

class BinPy.ic.series_7400.**IC_7425**
Bases: *BinPy.ic.base.Base_14pin*

This is a Dual 5-Input NOR Gate with Strobe

run()

class BinPy.ic.series_7400.**IC_7426**
Bases: *BinPy.ic.base.Base_14pin*

This is a Quad 2-input open-collector high-voltage NAND gates.

run()

class BinPy.ic.series_7400.**IC_74260**
Bases: *BinPy.ic.base.Base_14pin*

This is a dual 5-input NOR gate

run()

class BinPy.ic.series_7400.**IC_7427**
Bases: *BinPy.ic.base.Base_14pin*

This is a Triple 3-Input NOR Gate

run()

class BinPy.ic.series_7400.**IC_7428**
Bases: *BinPy.ic.base.Base_14pin*

This is a Quad 2-input NOR gates with buffered outputs.

run()

class BinPy.ic.series_7400.**IC_7430**
Bases: *BinPy.ic.base.Base_14pin*

This is a 8-Input NAND Gate

run()

class BinPy.ic.series_7400.**IC_7431**
Bases: *BinPy.ic.base.Base_16pin*

This is a Hex delay element.

run()

class BinPy.ic.series_7400.**IC_7432**
Bases: *BinPy.ic.base.Base_14pin*

This is a Quad 2-Input OR Gate

run()

class BinPy.ic.series_7400.**IC_7433**
Bases: *BinPy.ic.base.Base_14pin*

This is a Quad 2-input open-collector NOR gate

run()

class BinPy.ic.series_7400.**IC_7437**
Bases: *BinPy.ic.base.Base_14pin*

This is a Quad 2-input NAND gates with buffered output

run()

class BinPy.ic.series_7400.**IC_7438**
Bases: *BinPy.ic.base.Base_14pin*

This is a Quad 2-Input NAND Buffer with Open Collector Output

run()

class BinPy.ic.series_7400.**IC_7440**
Bases: *BinPy.ic.base.Base_14pin*

This is a Dual 4-Input NAND Buffer

run()

class BinPy.ic.series_7400.**IC_7442**
Bases: *BinPy.ic.base.Base_16pin*

This is a BCD to Decimal decoder BCD Digits are in order of A B C D where pin 15 = A, pin 12 = D

run()

class BinPy.ic.series_7400.**IC_7443**
Bases: *BinPy.ic.base.Base_16pin*

This is an excess-3 to Decimal decoder Excess-3 binary digits are in order of A B C D, where pin 15 = A and pin 12 = D

run()

class BinPy.ic.series_7400.**IC_7444**
Bases: *BinPy.ic.base.Base_16pin*

This is an excess-3 gray code to Decimal decoder Excess-3 gray code digits are in order of A B C D, where pin 15 = A and pin 12 = D

run()

class BinPy.ic.series_7400.**IC_7445**
Bases: *BinPy.ic.base.Base_16pin*

This is a Four-to-Ten (BCD to Decimal) DECODER using the DEMUX functionality from combinational.py datasheet at http://www.skot9000.com/ttl/datasheets/45.pdf

run()

class BinPy.ic.series_7400.**IC_7447**
Bases: *BinPy.ic.base.Base_16pin*

BCD to 7-segment decoder

run()

class BinPy.ic.series_7400.**IC_7451**
Bases: *BinPy.ic.base.Base_14pin*

This is a dual 2-wide 2-input AND-OR Invert gate

run()

class BinPy.ic.series_7400.**IC_7454**
Bases: *BinPy.ic.base.Base_14pin*

This is a 4-wide 2-input AND-OR Invert gate

run()

class BinPy.ic.series_7400.**IC_7455**
Bases: *BinPy.ic.base.Base_14pin*

This is a 4-wide 2-input AND-OR Invert gate

run()

class BinPy.ic.series_7400.**IC_7458**
Bases: *BinPy.ic.base.Base_14pin*

This is a 2-input and 3-input AND-OR gate

run()

class BinPy.ic.series_7400.**IC_7459**
Bases: *BinPy.ic.base.Base_14pin*

This is a 2-input and 3-input AND-OR inverter gate

run()

class BinPy.ic.series_7400.**IC_7464**
Bases: *BinPy.ic.base.Base_14pin*

This is a 4-2-3-2 input AND-OR-invert gate

run()

class BinPy.ic.series_7400.**IC_7470**
Bases: *BinPy.ic.base.Base_14pin*

AND gated JK Positive Edge triggered Flip Flop with preset and clear

run()

class BinPy.ic.series_7400.**IC_7472**
Bases: *BinPy.ic.base.Base_14pin*

AND gated JK Master-Slave Flip Flop with preset and clear

run()

class BinPy.ic.series_7400.**IC_7473**
Bases: *BinPy.ic.base.Base_14pin*

DUAL JK Flip Flops with clear

run()

class BinPy.ic.series_7400.**IC_7474**
Bases: *BinPy.ic.base.Base_14pin*

Dual D-Type Positive-Edge-Triggered Flip-Flops with preset and clear

run()

class BinPy.ic.series_7400.**IC_7475**
Bases: *BinPy.ic.base.Base_16pin*

4-Bit Bistable Latches

run()

class BinPy.ic.series_7400.**IC_7476**
Bases: *BinPy.ic.base.Base_16pin*

Dual JK Flip Flop with preset and clear

run()

class BinPy.ic.series_7400.**IC_7483**
Bases: *BinPy.ic.base.Base_16pin*

This is a 4-bit full adder with fast carry

run()

class BinPy.ic.series_7400.**IC_7485**
Bases: *BinPy.ic.base.Base_16pin*

4 bit magnitude comparator Comparing two 4-bit binary numbers A3A2A1A0 & B3B2B1B0

Pin Number Description 1 B3(MSB) 2 Cascade Input - A<B 3 Cascade Input - A=B 4 Cascade Input - A>B 5 Output A>B 6 Output A=B 7 Output A<B 8 Ground 9 B0 10 A0 11 B1 12 A1 13 A2 14 B2 15 A3(MSB) 16 VCC

We can compare 8,12,16... by cascading more of these ICs

run()

class BinPy.ic.series_7400.**IC_7486**
Bases: *BinPy.ic.base.Base_14pin*

This is a quad 2-input exclusive OR gate

run()

Module contents

tools

Submodules

- clock
- digital
- ground
- multivibrator
- oscilloscope
- powersource
- steppermotor

clock

class BinPy.tools.clock.**Clock**(*init_state=1*, *frequency=None*, *time_period=None*, *name=None*)
Bases: threading.Thread

This class uses threading technique to create a clock with a certain time period.

```
>>> my_clock = Clock(0,time_period=2,name="My First Clock")
>>> my_clock.get_state()
0
```

Parameters

- **frequency** (*It will decide time interval of the clock, use SI unit i.e. Hertz*) –
- **time_period** (*It will also decide time interval of the clock, use SI unit i.e. second*) –
- **init_state** (*It is the initial state of the clock(1 by default)*) –
- **name** (*It is the name of the clock.(optional)*) –
- **time_period and frequency both have been provided, then time_period** (*If*) –
- **override frequency** (*will*) –
- **nothing is provided, then it will set time_period = 1s by default** (*If*) –

start() [[Depricated] To start the clock thread.] Clock starts at __init__ itself. This need not be used.

get_state() : Get the current state of the clock. set_state() : To set the current state of the clock. kill() : To kill the clock thread.

state : [Property] Return the state of the clock. name : [Property] Return the name of the clock. time_period : [Property] Return the time period of the clock. frequency : [Property] Return the current frequency of the clock.

Note: Once you are done with the clock, use my_clock.kill() to kill the clock. Running too many clocks will unnecessarily overload the CPU.

All operations are thread safe and synchronized between inter / intra thread calls.

---

frequency

get_state()
Returns the current state of the clock

kill()
Kills the clock(Thread)

name
Returns the name of the clock

run()

set_state(*value*)
Resets the state of the clock to the passed value

start()
Do not use this method

state
Returns the currentd state of the clock as a property.

time_period

digital

class BinPy.tools.digital.**DigitDisplay**(*name=None*)
This class emulates a 7 segmented display(Common Cathode)

Parameters **name** – A name given to an object(Optional)

evaluate()

getName()

How to use:

```
>>> myDisplay = DigitDisplay("Display1")
>>> print myDisplay.evaluate([1,1,1,1,1,1,1])
8
```

Note: You can either pass complete list of 10 pins [pin1, pin2, pin3, pin4, pin5, pin6, pin7, pin8, pin9, pin10] in standard order or you can directly pass the list of values corresponding to a, b, c, d, e, f and g in lexicographical order.

---

http://tronixstuff.files.wordpress.com/2010/05/7segpinout.jpg

evaluate(*pin_conf*)
This method evaluates the values passed according to the display and returns an integer varying from 0 to 9

ground

class BinPy.tools.ground.**Ground**
Models a Ground from which various connectors can tap by connecting to it. taps: The list of all connectors connected to this ground. connect(): Takes in one or more connectors as input and connects them to the ground. disconnect(): Takes in one or more connectors as input and disconnects them from the ground.

connect(**connectors*)
Takes in one or more connectors as an input and taps to the ground.

disconnect(**connectors*)
Takes in one or more connectors as an input and disconnects them from the ground. A floating connector has a value of None. A message is printed if a specified connector is not already tapping from this ground.

multivibrator

class BinPy.tools.multivibrator.**Multivibrator**(*init_state=1*, *mode=1*, *frequency=None*, *time_period=None*, *on_time=None*, *off_time=None*)

Bases: threading.Thread

This class uses threading technique to create a multivibrator with a certain time period. USAGE:

```
>>> m1 = Multivibrator()
>>> m1.trigger()    # or m1()
>>> m1.get_state()  # or m1.A.state
0
>>> m1.set_mode(2)
>>> m1.trigger()
>>> m1.get_state()
>>> conn = Connector()
>>> m1.set_output(conn) # To set the output to connector conn
>>> conn()             # Retrieves the current state
```

Note: Once you are done with the multivibrator, use m1.kill() to kill the Multivibrators.

```
>>> m1.kill()
```

Following are the parameters of the class

frequency: It will decide time interval of the Multivibrator, use SI unit i.e. Hertz time_period: It will also decide time interval of the Multivibrator, use SI unit i.e. second

If time_period and frequency both have been provided, then time_period will override frequency If nothing is provided, then it will set time_period = 1s by default

init_state: It is the initial state of the multivibrator(1 by default)

mode: It is the mode of operation. 1 –> Monostable 2 –> Astable 3 –> Bistable

Methods : trigger(),setMode(), getState(), setState(value), getTimePeriod(), kill(), stop(), setOutput()

get_state()
Returns the current state

kill()
Kills the Thread

run()

set_mode(*mode*)
Sets the mode of the Multivibrator

set_output(*conn*)

set_state(*value*)
Resets the state of the clock to the passed value

start()
Do not use this method

stop()

time_period()
Returns the time period of the clock

trigger()

oscilloscope

class BinPy.tools.oscilloscope.**Oscilloscope**(**inputs*)
Bases: threading.Thread

Oscilloscope is helpful in visualizing simulations.

USAGE: # A clock of 1 hertz frequency clock = Clock(1, 1) clk_conn = clock.A

bc = BinaryCounter() os1 = Oscilloscope((bc.out[1],'lsb') , (bc.out[0],'msb')) #Triggering the counter: for i in range(5):

b.trigger() print (b.state())

os1.stop() os1.display()

clear(*keepInputs=False*)

disconnect(*conn*)
Disconnects conn from the inputDict

display()

hold()

kill()

run()

sampler(*trigPoint*)

set_colour(*foreground=None*, *background=None*)
Acceptable values are: 1 –> RED 2 –> GREEN 4 –> BLUE 7 –> WHITE

To RESET call without parameters.

Please note that serColor is not supported by all operating systems. This will run without problems on most Linux systems.

set_inputs(**inputs*)
Set inputs using a list of tuples.

For example: osc1.setInputs((conn1,"label") , (conn2,"label") ...)

set_scale(*scale=0.05*)
This decides the time per unit xWidth. To avoid waveform distortion, follow NYQUIST sampling theorem. That is if the least time period of the waveform is T; Set the scale to be greater than T/2 [preferably T/5 - To avoid edge sampling effects]

There is a lower bound on the scale value [use trial and error to identify this for your particular PC] This limitation is set by the processing time taken to set a plot etc.

set_width(*w=150*)
Set the maximum width of the oscilloscope. This is dependent on your current monitor configuration.

start()
Do not use this method

unhold()

powersource

class BinPy.tools.powersource.**PowerSource**
Models a Power Source from which various connectors can tap by connecting to it.

taps: The list of all connectors connected to this power source.

connect(): Takes in one or more connectors as input and connects them to the power source.

disconnect(): Takes in one or more connectors as input and disconnects them from the power source.

connect(**connectors*)
Takes in one or more connectors as an input and taps to the power source.

disconnect(**connectors*)
Takes in one or more connectors as an input and disconnects them from the power source. A floating connector has a value of None. A message is printed if a specified connector is not already tapping from this source.

steppermotor

Module contents

BinPy.base module

BinPy.base.**get_logging_level**()
This function prints the current logging level of the main logger.

BinPy.base.**init_logging**(*log_level*)

BinPy.base.**ipython_exception_handler**(*shell*, *excType*, *excValue*, *traceback*, *tb_offset=0*)

BinPy.base.**read_logging_level**(*log_level*)

BinPy.base.**set_logging**(*log_level*, *myfilename=None*)
This function sets the threshold for the logging system and, if desired, directs the messages to a logfile. Level options:

'DEBUG' or 1 'INFO' or 2 'WARNING' or 3 'ERROR' or 4 'CRITICAL' or 5

If the user is on the interactive shell and wants to log to file, a custom excepthook is set. By default, if logging to file is not enabled, the way errors are displayed on the interactive shell is not changed.

Module contents

CHAPTER 4

Development workflow

Contents

Introduction

BinPy encourages everyone to join and implement new features, fix some bugs, review pull requests and suggest ideas, take part in general discussions etc.

General discussion takes place on binpy@googlegroups.com mailing list and in the issues.

Discussions also take place on IRC (our channel is **'#binpy-soc at freenode'_**).

Note: Our IRC channel is not logged, so when you don't get your answer from there, please post on the google group.

How to submit a patch

The best way to submit a patch is to send a GitHub pull request against the BinPy repository. We'll review it and merge it with our code base.

Though patches can also be submitted as .patch file, it is highly recommended that you follow the GitHub Pull Request method.

The basic work-flow is as follows:

1. Create your environment, if it was not created earlier.
2. Create a new branch.
3. Modify code and/or create tests for it.
4. Be sure that all tests of BinPy pass. Make sure you repeat the tests with both python 2 and 3.
5. Only then commit changes. Follow commit message guidelines as given here.
6. After you commit make sure the code is properly indented using the *pep8* tool. If not use the *autopep8* tool to make the necessary formatting and recheck for *pep8* conformity. In rare cases you might need to manually make the changes.
7. Send the Pull Request on GitHub.

All those are described in the details below *Workflow process*, but before you read that, it would be useful to acquaint yourself with **'Coding conventions followed in BinPy'_**.

Hint: If you have any questions you can ask them on the mailinglist.

Coding conventions in BinPy

Standard Python coding conventions

We value clean code a lot and hence ‘ pep8 <http://www.python.org/dev/peps/pep-0008>‘_ consistency check has been made mandatory. Please follow the standard Style Guide for Python Code when writing code for BinPy.

In particular,

- Use 4 spaces for indentation levels.
- Use all lowercase function names with words separated by underscores. For example, you are encouraged to write Python functions using the naming convention

```
def print_output()

def update_previous_bits()
```

  instead of the CamelCase convention.

- Use CamelCase for class names and major functions that create objects, e.g.

```
class StateMachineSolver(object)

class NBitRippleCounter()
```

Note: Do not use tabs for indentation.

Note, however, that some functions do have uppercase letters where it makes sense. For example, *enabled_S*.

Documentation strings

An example of a well formatted docstring:

```
"""
This class is the primary medium for data transfer. Objects of this
class can be connected to any digital object.

Example
=======

>>> from BinPy import *
>>> conn = Connector(1)  #Initializing connector with initial state = 1
>>> conn.state
1
>>> gate = OR(0, 1)
>>> conn.tap(gate, 'output')  #Tapping the connector

Methods
=======

* tap
* untap
* isInputof
* isOutputof
* trigger
```

For more information see [[Writing documentation]] article on wiki.

Python 3

BinPy uses a single codebase for Python 2 and Python 3 (the current supported versions are Python 2.7, 3.2, 3.3 and 3.4). This means that your code needs to run in both Python 2 and Python 3.

You may refer this. as a ready reference to implement code that is both python 2 and 3 compatible.

You should also make sure that you have:

```
from __future__ import print_function, division
```

at the top of each file. This will make sure that `print` is a function, and that `1/2` does floating point division and not integer division. You should also be aware that all imports are absolute, so `import module` will not work if `module` is a module in the same directory as your file. You will need to use `import .module`.

Workflow process

Create your environment

Creating of environment is once-only.

Install git

To install *git* in Linux-like systems you can do it via your native package management system:

```
$ yum install git
```

or:

```
$ sudo apt-get install git
```

In Windows systems, first of all, install Python from:

```
http://python.org/download/
```

by downloading the "Python 2.7 Windows installer" and running it. Then do not forget to add Python to the $PATH environment.

On Windows and Mac OS X, the easiest way to get git is to download GitHub's software, which will install git, and also provide a nice GUI (this tutorial will be based on the command line interface). Note, you may need to go into the GitHub preferences and choose the "Install Command Line Tools" option to get git installed into the terminal.

If you do decide to use the GitHub GUI, you should make sure that any "sync does rebase" option is disabled in the settings.

Install other software

Basic git settings

Git tracks who makes each commit by checking the user's name and email. In addition, we use this info to associate your commits with your GitHub account.

To set these, enter the code below, replacing the name and email with your own (*–global* is optional).:

```
$ git config --global user.name "Firstname Lastname"
$ git config --global user.email "your_email@youremail.com"
```

The name should be your actual name, not your GitHub username.

These global options (i.e. applying to all repositories) are placed in *~/.gitconfig*. You can edit this file to add setup colors and some handy shortcuts:

```
[user]
    name = Firstname Lastname
    email = your_email@youremail.com

[color]
    diff  = auto
    status= auto
    branch= auto
    interactive = true

[alias]
    ci = commit
    di = diff --color-words
    st = status
    co = checkout
    log1 = log --pretty=oneline --abbrev-commit
    logs = log --stat
```

Advanced tuning

It can be convenient in future to tune the bash prompt to display the current git branch.

The easiest way to do it, is to add the snippet below to your .bashrc or .bash_profile:

```
PS1="[\u@\h \W\$(git branch 2> /dev/null | grep -e '\* ' | sed 's/^..\(.*\)/{\1}/
↪')]\$ "
```

But better is to use *git-completion* from the *git* source. This also has the advantage of adding tab completion to just about every git command. It also includes many other useful features, for example, promptings. To use *git-completion*, first download the *git* source code (about 27 MiB), then copy the file to your profile directory:

```
$ git clone git://git.kernel.org/pub/scm/git/git.git
$ cp git/contrib/completion/git-completion.bash ~/.git-completion.sh
```

Read instructions in '~/.git-completion.sh'

Note that if you install git from the package manager in many Linux distros, this file is already installed for you. You can check if it is installed by seeing if tab completion works on git commands (try, e.g., *git commi<TAB>*, or *git log –st<TAB>*). You can also check if the PS1 commands work by doing something like:

```
$ PS1='\W $(__git_ps1 "%s")\$ '
```

And your command prompt should change to something like:

```
BinPy master$
```

Note: It is important to define your PS1 using single quotes (‘), not double quotes (”), or else bash will not update the branch name.

Create GitHub account

As you are going to use GitHub you should have a GitHub account. If you have not one yet then sign up at:

- https://github.com/signup/free

Then create your own *fork* of the BinPy project (if you have not yet). Go to the BinPy GitHub repository:

- https://github.com/BinPy/BinPy

and click the "Fork" button.

Now you have your own repository for the BinPy project. If your username in GitHub is *mynick* then the address of the forked project will look something like:

- https://github.com/mynick/BinPy

Cloning BinPy

On your machine browse to where you would like to clone BinPy, and clone (download) the latest code from BinPy's original repository:

```
$ git clone git://github.com/BinPy/BinPy.git
$ cd BinPy
```

Then assign your read-and-write repo to a remote called "github":

```
$ git remote add github git@github.com:mynick/BinPy.git
```

Create separated branch

Typically, you will create a new branch to begin work on a new issue. Also pull request related with them.

A branch name should briefly describe the topic of the patch or pull request. If you know the issue number, then the branch name could be, for example, *1234_sequences*. To create and checkout (that is, make it the working branch) a new branch

```
$ git branch 123_sequential
$ git checkout 123_sequential
```

or in one command using

```
$ git checkout -b 123_sequential
```

To view all branches, with your current branch highlighted, type:

```
$ git branch
```

And remember, **never type the following commands in master**: *git merge*, *git commit*, *git rebase*.

Code modification

Do not forget that all new functionality should be tested, and all new methods, functions, and classes should have doctests showing how to use them.

Be sure that all tests of BinPy_ pass

To see if all tests pass:

```
$ cd BinPy/BinPy/tests
$ nosetests && nosetests3
```

Remember that all tests should pass before committing.

Note that all tests will be run when you make your pull request automatically by Travis CI.

Commit the changes

You can check what files are changed:

```
$ git status
```

Add new files to the index if necessary:

```
$ git add new_file.py
```

Check total changes:

```
$ git diff
```

You are ready to commit changes locally. A commit also contains a *commit message* which describes it. See the next section for guidelines on writing good commit messages. Type:

```
$ git commit
```

An editor window will appear automatically in this case. In Linux, this is vim by default. You can change what editor pops up by changing the *$EDITOR* shell variable.

Also with the help of option *-a* you can tell the command *commit* to automatically stage files that have been modified and deleted, but new files you have not told git about will not be affected, e.g.,:

```
$ git commit -a
```

If you want to stage only part of your changes, you can use the interactive commit feature. Just type:

```
$ git commit --interactive
```

and choose the changes you want in the resulting interface.

Writing commit messages

There are only two formatting rules for commit messages

- All lines should be 78 characters or less. This is so they can be easily read in terminals, which don't automatically line wrap things.
- There should be a single line with a summary, then an empty line, followed by (optional) additional details. A common convention is to not end the first line with a `.`, but all additional lines should (this convention probably exists to save an extra character to make it easier to fit the first line summary in 78 characters).

Some things to note about An ideal commit message:

- The first line gives a brief description of what the commit does. Tools like `git shortlog` or even GitHub only show the first line of the commit by default, so it is important to convey the most important aspects of the commit in the first line.
- The first line starts with a key word which gives context to the commit. A commit won't always be seen in the context of your branch, so it is often helpful to give each commit some context. This is not required, though, as it is not hard to look at the commit metadata to see what files were modified or at the commit history to see the nearby related commits.
- After the first line, there is a paragraph describing the commit in more detail. This is important, as it describes what the commit does, which might be hard to figure out just from looking at the diff. It also gives information that might not be in the diff at all, such as known issues. Such paragraphs should be written in plain English. Commit messages are intended for human readers, both for people who will be reviewing your code right now, and for people who might come across your commit in the future while researching some change in the code. Sometimes, bulleted lists are a good format to convey the changes of a commit.

- Last, there is an example. It is nice to give a concrete example in commits that add new features. This particular example is about improving the speed of something, so the example is a benchmark result.

Other things to do in commit messages:

- **If the bug fixes an issue, reference that issue in the message (with a string like ‘‘closes #123‘‘, see ‘this <https://help.github.com/articles/closing-issues-via-commit-messages>‘_ for exact syntax reference)**. Also reference any pull requests or mailing list messages with links. This will make it easier to find related discussions about the commit in the future. You do not need to add a reference to the pull request that contains the commit. That can be found from the git log.

Try to avoid short commit messages, like “Fix”, and commit messages that give no context, like “Found the bug”. When in doubt, a longer commit message is probably better than a short one.

Create a patch file or pull request for GitHub

Be sure that you are in your own branch, and run:

```
$ git push github 123_sequential
```

This will send your local changes to *your* fork of the BinPy repository. Then navigate to your repository with the changes you want someone else to pull:

https://github.com/mynick/BinPy

Select branch, and press the *Pull Request* button.

After pressing the *Pull Request* button, you are presented with a preview page where you can enter a title and optional description, see exactly what commits will be included when the pull request is sent, and also see who the pull request will be sent to

If you’re sending from a topic branch, the title is pre-filled based on the name of the branch. Markdown is supported in the description, so you can embed images or use preformatted text blocks.

You can switch to the *Commits* tab to ensure that the correct set of changes is being sent. And review the diff of all changes by switching to the *Files Changed*.

Once you’ve entered the title and description, made any necessary customizations to the commit range, and reviewed the commits and file changes to be sent, press the *Send pull request* button.

The pull request is sent immediately. You’re taken to the main pull request discussion and review page. Additionally, all repository collaborators and followers will see an event in their dashboard.

That’s all.

See also *Updating your pull request*

Updating your pull request

If after a time you need to make changes in pull request then the best way is to add a new commit in you local repository and simply repeat push command:

```
$ git commit
$ git push github 123_sequential
```

Note that if you do any rebasing or in any way edit your commit history, you will have to add the *-f* (force) option to the push command for it to work:

```
$ git push -f github
```

You don’t need to do this if you merge, which is the recommended way.

Synchronization with master *BinPy/BinPy*.

Sometimes, you may need to merge your branch with the upstream master. Usually you don't need to do this, but you may need to if

- Someone tells you that your branch needs to be merged because there are merge conflicts.
- While raising a Pull Request, you get a message from github telling you that your branch could not be merged.
- You need some change from master that was made after you started your branch.

Note, that after cloning a repository, it has a default remote called *origin* that points to the *BinPy/BinPy* repository. And your fork remote named as *github*. You can observe the remotes names with the help of this command:

```
$ git remote -v
github  git@github.com:mynick/BinPy.git (fetch)
github  git@github.com:mynick/BinPy.git (push)
origin  git://github.com/BinPy/BinPy.git (fetch)
origin  git://github.com/BinPy/BinPy.git (push)
```

As an example, consider that we have these commits in the master branch of local git repository:

```
A---B---C        master
```

Then we have divergent branch *123_sequential*:

```
A---B---C           master
         \
          a---b     123_sequential
```

In the meantime the remote *BinPy/BinPy* master repository was updated too:

```
A---B---C---D       origin/master
A---B---C           master
         \
          a---b     123_sequential
```

There are basically two ways to get up to date with a changed master: rebasing and merging. Merging is recommended.

Merging creates a special commit, called a "merge commit", that joins your branch and master together:

```
A---B---C------D       origin/master
         \      \
          \      M     merge
           \    /
            a--b       123_sequential
```

Note that the commits `A`, `B`, `C`, and `D` from master and the commits `a` and `b` from `123_sequential` remain unchanged. Only the new commit, `M`, is added to `123_sequential`, which merges in the new commit branch from master.

Rebasing essentially takes the commits from `123_sequential` and reapplies them on the latest master, so that it is as if you had made them from the latest version of that branch instead. Since these commits have a different history, they are different (they will have different SHA1 hashes, and will often have different content):

```
A---B---C---D---a'---b' origin/master
```

Rebasing is required if you want to edit your commit history (e.g., squash commits, edit commit messages, remove unnecessary commits). But note that since this rewrites history, it is possible to lose data this way, and it makes it harder for people reviewing your code, because they can no longer just look at the "new commits"; they have to look at everything again, because all the commits are effectively new.

There are several advantages to merging instead of rebasing. Rebasing reapplies each commit iteratively over master, and if the state of the files changed by that commit is different from when it was originally made, the commit will change. This means what you can end up getting commits that are broken, or commits that do not do what they say they do (because the changes have been "rebased out"). This can lead to confusion if someone in the future tries to test something by checking out commits from the history. Finally, merge conflict resolutions can be more difficult with rebasing, because you have to resolve the conflicts for each individual commit. With merging, you only have to resolve the conflicts between the branches, not the commits. It is quite common for a merge to not have any conflicts but for a rebase to have several, because the conflicts are "already resolved" by later commits.

Merging keeps everything intact. The commits you make are exactly the same, down to the SHA1 hash, which means that if you checkout a commit from a merged branch, it is exactly the same as checking it out from a non-merged branch. What it does instead is create a single commit, the merge commit, that makes it so that the history is both master and your branch. This commit contains all merge conflict resolution information, which is another advantage over rebasing (all merge conflict resolutions when rebasing are "sifted" into the commits that caused them, making them invisible).

Since this guide is aimed at new git users, you should be learning how to merge.

Merging

First merge your local repository with the remote:

```
$ git checkout master
$ git pull
```

This results in:

```
A---B---C---D       master
         \
          a---b     123_sequential
```

Then merge your *123_sequential* branch from *123_sequential*:

```
$ git checkout 123_sequential
$ git merge master
```

If the last command tells you that conflicts must be solved for a few indicated files.

If that's the case then the marks **>>>** and **<<<** will appear at those files. Fix the code with **>>>** and **<<<** around it to what it should be. You must manually remove useless pieces, and leave only new changes from your branch.

Then be sure that all tests pass:

```
$ cd BinPy/BinPy/tests
$ nosetests && nosetests3
```

and commit:

```
$ git commit
```

So the result will be like that (automatic merging *c*):

```
A---B---C-------D     master
         \       \
          a---b---M   123_sequential
```

Rebasing

***Note*: merging is recommended over rebasing.**

The final aim, that we want to obtain is:

```
A---B---C---D           master
             \
              a---b     123_sequential
```

The way to do it is first of all to merge local repository with the remote *BinPy/BinPy*:

```
$ git checkout master
$ git pull
```

So we obtain:

```
A---B---C---D       master
         \
          a---b     123_sequential
```

Then:

```
$ git checkout 123_sequential
$ git rebase master
```

Note that this last one will require you to fix some merge conflicts if there are changes to the same file in `master` and `123_sequential`. Open the file that it tells you is wrong, fix the code with **>>>** and **<<<** around it to what it should be.

Then be sure that all tests pass:

```
$ cd BinPy/BinPy/tests
$ nosetests && nosetests3
```

Then do:

```
$ git add BinPy/BinPy/Sequential/your_conflict_file
$ git rebase --continue
```

(git rebase will also guide you in this).

Changing of commit messages

The only time when it is recommended to rebase instead of merge is when you need to edit your commit messages, or remove unnecessary commits.

Note, it is much better to get your commit messages right the first time. See the section on writing good commit messages above.

Consider these commit messages:

```
$ git log --oneline
7bbbc06 bugs fixing
4d6137b some additional corrections.
925d88fx sequences base implementation.
```

Then run *rebase* command in interactive mode:

```
$ git rebase --interactive 925d88fx
```

Or you can use other ways to point to commits, e.g. *'git rebase –interactive HEAD^^'* or *'git rebase –interactive HEAD~2'*.

A new editor window will appear (note that order is reversed with respect to the *git log* command):

```
pick 4d6137b some additional corrections.
pick 7bbbc06 bugs fixing

# Rebase 925d88f..7bbbc06 onto 925d88f
#
# Commands:
#  p, pick = use commit
#  r, reword = use commit, but edit the commit message
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#  f, fixup = like "squash", but discard this commit's log message
```

To edit a commit message, change *pick* to *reword* (or on old versions of git, to *edit*) for those that you want to edit and save that file.

To squash two commits together, change *pick* to *squash*. To remove a commit, just delete the line with the commit.

To edit a commit, change *pick* to *edit*.

After that, git will drop you back into your editor for every commit you want to reword, and into the shell for every commit you wanted to edit:

```
$ (Change the commit in any way you like.)
$ git commit --amend -m "your new message"
$ git rebase --continue
```

For commits that you want to edit, it will stop. You can then do:

```
$ git reset --mixed HEAD^
```

This will "uncommit" all the changes from the commit. You can then recommit them however you want. When you are done, remember to do:

```
$ git rebase --continue
```

Most of this sequence will be explained to you by the output of the various commands of git. Continue until it says:

```
Successfully rebased and updated refs/heads/master.
```

If at any point you want to abort the rebase, do:

```
$ git rebase --abort
```

Warning: This will run `git reset --hard`, deleting any uncommitted changes you have. If you want to save your uncommitted changes, run `git stash` first, and then run `git stash pop` when you are done.

Reviewing patches

Coding's only half the battle in software development: our code also has to be thoroughly reviewed before release. Reviewers thus are an integral part of the development process. Note that you do *not* have to have any special pull or other privileges to review patches: anyone with Python on his/her computer can review.

Pull requests (the preferred avenue for patches) for BinPy are located here. Feel free to view any open pull request. Each contains a Discussion section for comments, Commits section for viewing the author's commit files and documentation, and Diff section for viewing all the changes in code. To browse the raw code files for a commit, select a commit in the Commits section and click on the "View file" link to view a file.

Based on your level of expertise, there are two ways to participate in the review process: manually running tests. Whichever option you choose, you should also make sure that the committed code complies with the [[Writing documentation]] guidelines.

Manual testing

If you prefer to test code manually, you will first have to set up your environment as described in the Workflow process section. Then, you need to obtain the patched files. If you're reviewing a pull request, you should get the requested branch into your BinPy folder. Go into your folder and execute (<username> being the username of the pull requester and <branchname> being the git branch of the pull request):

```
$ git remote add <username> git://github.com/<username>/BinPy.git
$ git fetch <username>
$ git checkout -b <branchname> <username>/<branchname>
```

After obtaining the pull request or patch, go to your BinPy root directory and execute:

```
$ cd BinPy/BinPy/tests
$ nosetests && nosetests3
```

If there are any problems, notify the author in the pull request by commenting.

Requirements for inclusion

A pull request or patch must meet the following requirements during review before being considered as ready for release.

- **All tests must pass.**
  - Rationale: We need to make sure we're not releasing buggy code.
  - If new features are being implemented and/or bug fixes are added, tests should be added for them as well.
- **The reviews (at least 1) must all be positive.**
  - Rationale: We'd like everyone to agree on the merits of the patch.
  - If there are conflicting opinions, the reviewers should reach a consensus.
- **The patch must have been posted for at least 24 hours.**
  - Rationale: This gives a chance for everyone to look at the patch.

References

CHAPTER 5

About

BinPy Development Team

It was started by two undergraduate college students. After that many people have contributed to this project. Here is a list of few of them.

Note: All people who contributed to BinPy by sending at least a patch or more. (List is in the order of the date of their first contribution)

1. Sudhanshu Mishra <mrsud94@gmail.com>
2. Sarwar Chahal <chahal.sarwar98@gmail.com>
3. Abhinav Gupta <aag999in@gmail.com>
4. Kaushik Kalyan <kaushik.kalyan@gmail.com>
5. Sachin Joglekar <srjoglekar246@gmail.com>
6. Jay Rambhia <jayrambhia777@gmail.com>
7. Salil Kapur <salilkapur93@gmail.com>
8. M S Suraj <mssurajkaiga@gmail.com>
9. Amit <bitsjamadagni@gmail.com>
10. Jaspreet Singh <jaspreetsingh112@gmail.com>
11. Rajat Aggarwal <rajataggarwal1975@gmail.com>
12. Raghav R V <rvraghav93@gmail.com>
13. Aliya Hameer <adhameer@gmail.com>
14. Kunal Arora <kunalarora.135@gmail.com>
15. Karan <karan>
16. Mayuresh <mayuresh2212@gmail.com>
17. mohitgujarathi14 <mohitontherocks14@gmail.com>
18. Joaquín Bermúdez Castanheira <jbcsound@gmail.com>

19. S Sandeep Reddy <sandeepreddys09@gmail.com>
20. Pedro Melgueira <pedromelgueira@gmail.com>
21. Shashank Garg <garg.shashank.5@gmail.com>
22. Ahmed Hemdan <a.hemdan.alatif@gmail.com>
23. Vikas Mishra <vikasmishra95@gmail.com>
24. shrutig <shruti.999.999@gmail.com>
25. Akshay <akshaynukala95@gmail.com>
26. Vandan-V-Phadke <one.donerocks@gmail.com>

Prakhar Bharadwaj has contributed to BinPy by designing its logo.

Support

DigitalOcean is supporting us by providing infrastructural support to host our website on their servers. We are very thankful to them.

CHAPTER 6

License

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

Index

K

L

M

N

O

P

R

S

T

U