Summer Fellowship Report

On

PSpice to KiCad Converter

Submitted by

Sumanto Kar

Under the guidance of

Prof.Kannan M. Moudgalya
Chemical Engineering Department
IIT Bombay

June 15, 2020

Acknowledgment

I would like to express my very gratefulness to Prof. Kannan M. Moudgalya for his valuable and constructive suggestions during the planning and development of this research work. His willingness to give his time so generously and encouraging fellows has been very much appreciated.

I would also like to thank the eSim team for enabling me to for their help in offering me the resources and guiding me all throughout the project.

A special thanks to my mentor, Gloria Madam for helping me, sharing a lot of knowledge with me and giving me a wonderful fellowship experience.

Finally, I wish to thank my parents for their support and encouragement throughout my study.

Contents

1 Introduction 4

2 Problem Statement 5

3 Implementation 6

3.1 Conversion of Schematic Files . . . 7
3.1.1 parser.py . . . 7
3.1.2 componentInstances.py . . . 9
3.1.3 attribute.py . . . 10
3.1.4 Associate Files and Programs . . . 11
3.2 Conversion of Library Files . . . 15
3.2.1 libParser.py . . . 15
3.2.2 component.py . . . 16

4 Results 18

4.1 Example 1 . . . 18
4.1.1 PSpice Schematic and Results . . . 18
4.1.2 KiCad Schematic . . . 19
4.1.3 KiCad Results . . . 19
4.2 Example 2 . . . 20
4.2.1 PSpice Schematic and Results . . . 20
4.2.2 KiCad Schematic . . . 20
4.2.3 KiCad Results . . . 21
4.3 Example 3 . . . 21
4.3.1 PSpice Schematic and Results . . . 21
4.3.2 KiCad Schematic . . . 22
4.3.3 KiCad Results . . . 22
4.4 Example 4 . . . 23
4.4.1 PSpice Schematic and Results . . . 23
4.4.2 KiCad Schematic . . . 23
4.4.3 KiCad Results . . . 24
4.5 Example 5 . . . 24
4.5.1 PSpice Schematic and Results . . . 24
4.5.2 KiCad Schematic . . . 25
4.5.3 KiCad Results . . . 25
4.6 Example 6 . . . 26
4.6.1 PSpice Schematic . . . 26

4.6.2 KiCad Schematic 26

5 Workflow **27**

6 Conclusion and Future Scope **28**
6.1 Conclusion 28
6.2 Future Work 28

Bibliography **29**

Chapter 1

Introduction

PSpice simulation technology combines industry-leading, native analog and mixed-signal engines to deliver a complete circuit simulation and verification solution. It meets the changing simulation needs of designers as they progress through the design cycle, from circuit exploration to design development and verification. Designed for use in conjunction with PSpice A/D, PSpice Advanced Analysis helps designers improve yield, and reliability.[1]

KiCad is a free software suite for electronic design automation (EDA). It facilitates the design of schematics for electronic circuits and their conversion to PCB designs. KiCad was originally developed by Jean-Pierre Charras. It features an integrated environment for schematic capture and PCB layout design. Tools exist within the package to create a bill of materials, artwork, Gerber files, and 3D views of the PCB and its components.[3]

eSim is a free/libre and open source EDA tool for circuit design, simulation, analysis and PCB design. It is an integrated tool built using free/libre and open source software such as KiCad, Ngspice, NGHDL and GHDL. eSim is released under GPL. Because of this, it has the necessary packages and tools to integrate micro-controller into it.[4]

Earlier version of PSpice supported simulation and didn't have support for PCB. Later they joined with OrCAD for PCB generation. The PSpice is a paid software and due to which the users can have limited access to the libraries and software development. So we worked on how to convert the PSpice schematic files into KiCad which can be further used for PCB generation and simulation in Ngspice. Since it is a open source, user can add his own library and contribute in development of the software.

The tool maps the symbol schematic of PSpice to KiCad symbol schematic. The tool also converts the PSpice slb files to library files of KiCad .

Chapter 2

Problem Statement

As discussed earlier, the earlier version of PSpice was used for spice simulation but didn't have the feature of PCB making whereas the KiCad was used for PCB designing didn't have the feature of spice simuation.

Hence, in order to meet the disadvantages of both the software we thought to design an application that will convert the PSpice Schematic and Library files to KiCad Schematic and Library files respectively with proper mapping of the components and the wiring. By this way one will be able to simulate their schematics in PSpice and get the PCB layout in KiCad.

The application should -

- Be as user-friendly as possible
- Easily implemented
- Error free
- Independent of the versions of PSpice
- Should be able to map maximum number of components

Chapter 3

Implementation

The design of the PSpice to KiCad Converter is achieved by creating a Parser file in Python Programming Language. Two Parser files are created named parser.py and libParser.py. The parser.py parses the PSpice schematic file(.sch) into the components file(componentinstance.py), attributes file(attribute.py) and associate files(wire.py, design.py and misc.py) and maps it to the KiCad schematic file(.sch). The libParser.py parses the PSpice library file(.slb) into the components file(component.py) and maps it to the KiCad library file(.lib).

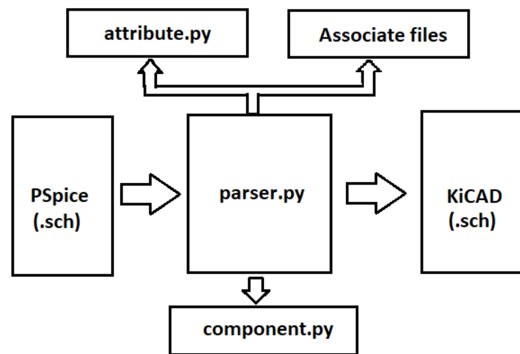

Figure 3.1: Conversion of .sch file

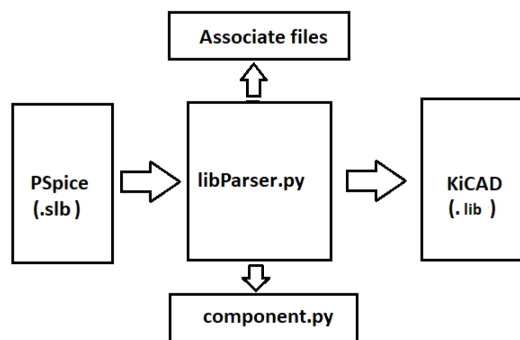

Figure 3.2: Conversion of .slb file

3.1 Conversion of Schematic Files

3.1.1 parser.py

To parse, in computer science, is where a string of commands usually a program is separated into more easily processed components, which are analyzed for correct syntax and then attached to tags that define each component. The computer can then process each program chunk and transform it into machine language. [5]

Here as well the parser.py does the process of parsing. It is the most important part of our framework as it breaks the work in modules and acts like a kind of interface between the PSpice schematic and the KiCad schematic. The process of parsing is achieved in the following ways:

- Imports all the necessary associate files.
- The parser.py creates .proj, .pro and the .sch files for KiCad in the directory specified by the user.
- It creates a directory.
- It creates the string of libraries and library names to be added in the eeschema and project file respectively
- Reading the components from the PSpice schematic file and mapping them to the KiCad schematic file using the componentInstances.py file.
- Mapping the wires connecting the components from PSpice schematic file to KiCad schematic file using wire.py file.

Creating .proj and .sch files:

```
#print('opening .proj file')
fproj  = open(fprojname,'w+')
fproj.write('schematicFile '+ base_name + '.sch.sch' + '\n')
fproj.close()
#print('closing .proj file')

#print('opening .pro file')
fpro = open(fproname, 'w+')
fpro.write(proDescr + '\n')
fpro.close()
#print('closing .pro file')
textline = file.readline().strip()
while('@status' not in textline):
    textline = file.readline().strip()
    #print(textline)
```

Calling functions from componentInstances.py and wires.py in parser.py :

```
componentInstances = []
g = file.tell()
textline = file.readline().strip()
while(textline[:4] == 'port' and textline != ''):
    #print('decoding ports')
    #print(textline)
    file.seek(g)
    ci = ComponentInstance(file)
    if ci.type_ == 'AGND' or ci.type_ == 'GND_ANALOG' or ci.type_ == 'GND_EARTH'
     ↪  or ci.type_ == 'EGND' or ci.type_ == '+5V' or ci.type_ == '-5V' :
        #print(ci.type_)
        fixInst(ci)
        componentInstances.append(ci)
        componentInstances[-1].attrs[0].value =
         ↪  '#PWR'+str(len(componentInstances))
    g = file.tell()
textline = file.readline().strip()
'''file.seek(g)
g = file.tell()
print(file.readline().strip())
file.seek(g)
print('**', textline, ('@parts' in textline))
'''
while('@parts' not in textline and textline!=''):
^^I#print('parts')
^^Itextline = file.readline().strip()
^^I#print(textline)
g = file.tell()

#textline = file.readline().strip()
textline = file.readline().strip()
#print('part->', textline)
while(textline[:4] == 'part' and textline != ''):
    file.seek(g)
    #print('part',file.readline().strip())
    file.seek(g)
    ci = ComponentInstance(file)
    fixInst(ci)
    ci.type_ = ci.type_ + nameAppend
    componentInstances.append(ci)
    g = file.tell()
    textline = file.readline().strip()

file.seek(g)
```

```
#print('len of componentInstances = ', len(componentInstances))
for i in range(0, len(componentInstances)):
    componentInstances[i].print(fsch)

while('@conn' not in textline and textline != ''):
    textline = file.readline().strip()

wires = []
file.readline().strip()
parseWire(file, wires)

for i in range(0, len(wires)):
    wires[i].print(fsch)

conns = []
parseConn(file, conns)

for i in range(0, len(conns)):
    conns[i].print(fsch)

fsch.write('$EndSCHEMATC\n')
fsch.close()
```

3.1.2 componentInstances.py

The purpose of the this program is to take inputs of components of PSpice Schematic from the parser.py and map the components to the KiCad schematic and return the mapped components to the parser.py file.

Its implementation is as follows:

- Import the required associate programs and files
- It maps the coordinates of PSpice to that of KiCad.
- It also maps the orientation and directions of the pins from Pspice to KiCad using the if else structure
- It also maps the pins of the components in PSpice schematic to that of the KiCad
- It calls functions from attribute.py
- All the mapping done and the data obtained is used by the parser.py to write to the KiCad schematic file.

The following is the block of the code componentInstance.py which shows mapping of components:

```
def print(self, output_stream):
    output_stream.write('$Comp\n'+'L '+self.type_+' '+self.attrs[0].value+'\n')
    output_stream.write('U 1 1 '+str(randint(0,
     ↪  sys.maxsize+1)%90000000+10000000)+'\n')
    output_stream.write('P '+str(self.x)+' '+str(self.y)+'\n') #printing the
     ↪  postion of component
    output_stream.write('F 0')
    self.attrs[0].print(output_stream)
    output_stream.write('F 1')
    self.attrs[1].print(output_stream)
    output_stream.write('\t1    '+str(self.x)+' '+str(self.y)+'\n')
    if self.orient == 'v':
        output_stream.write('\t0    -1    -1    0\n')
    if self.orient == 'V':
        output_stream.write('\t0    1    -1    0\n')
    if self.orient == 'h':
        output_stream.write('\t1    0    0    -1\n')
    if self.orient == 'H':
        output_stream.write('\t-1    0    0    -1\n')
    if self.orient == 'u':
        output_stream.write('\t-1    0    0    1\n')
    if self.orient == 'U':
        output_stream.write('\t1    0    0    1\n')
    if self.orient == 'd':
        output_stream.write('\t0    1    1    0\n')
    if self.orient == 'D':
        output_stream.write('\t0    -1    1    0\n')
    output_stream.write('$EndComp\n')
```

3.1.3 attribute.py

The purpose of the attribute.py is to create the necessary scaling and transforming the strings into coordinates which is later used by the componentInstance.py files.

The implementation of attribute.py is as follows:

- Import the necessary files
- Splitting and mapping the input string into coordinates given by the component files
- Scaling the coordinates as the coordinates of PSpice are very small than that of KiCad
- It also checks the type of the attribute
- Gives the data back to the components file

The following is the block of the code attribute.py:

```
if len(line) != 0:
            input_line = line.strip().split()
            #print(input_line)
            a,t,temp,vis,x0,y0,temp = input_line[:7]
            self.orient,self.hjust,self.vjust = list(input_line[7])
            t= input_line[8]
            temp = ' '.join(map(str,input_line[9:]))
            temp = temp.split()[0]
            x0 = int(x0)
            y0 = int(y0)
            self.x = x0 * MULT
            self.y = y0 * MULT
            t = temp.find('=')
            #everything in temp occuring before the '=' is the "name", and
             ↪  everything after it is the "value".
            self.name = temp[0:t]
            self.value = temp[t+1:]
            if vis.find('13') == -1:
                self.isHidden = True
            else:
                self.isHidden  = False
```

3.1.4 Associate Files and Programs

There are some other associate files and programs implemented which are listed below:

- header.py- It includes all the necessary header files and constants. It also has the value to be multiplied with the components. This is mainly because PSpice components are very small in size. To scale the components, the dimensions of the components are multiplied by 10 and then converted to KiCad symbols.

```
MULT = 10
nameAppend = ''
REMOVEDCOMPONENTS = ''
```

- include.py- It includes all the necessary associate files.

- design.py- It creates the block and shape of the component like line, circle, rectangle, etc using the coordinates and passes it to component files. It also writes the text of the component(For Example:'LM741' for IC, R for resistor, etc). Basically, it creates the block diagram of the component. Shapes of

Rectangle and Circle components in design.py:

```
class Rectangle:
    x1 = 0
    y1 = 0
    x2 = 0
    y2 = 0
    def __init__(self, input_stream, shiftx, shifty):
        input_line = input_stream.readline().strip()
        #print('Rect->',input_line)
        self.x1, self.y1,self.x2,self.y2 = input_line.split()[:4]
        self.x1 = (int(self.x1) * MULT) - shiftx
        self.x2 = (int(self.x2) * MULT) - shiftx
        self.y1 = (int(self.y1) * -1 * MULT) - (-1*shifty)
        self.y2 = (int(self.y2) * -1 * MULT) - (-1*shifty)
def print(self, output_stream):
        output_stream.write('S '+str(self.x1)+' '+str(self.y1)+'
         ↪  '+str(self.x2)+' '+str(self.y2)+' 0 1 0 N\n')
class Circle:
    x = 0
    y = 0
    r = 0
    def __init__(self, input_stream, shiftx, shifty):
        self.x, self.y, self.r =
         ↪  map(int,input_stream.readline().strip().split())
        #tmp = input_stream.readline().strip()
        #print('Circle->','x=',self.x,'y=',self.y,'r=',self.r)
        self.x*= MULT
        self.x-= shiftx
        self.y*=-1*MULT
        self.y-=-1*shifty
        self.r*= MULT
    def print(self, output_stream):
        output_stream.write('C '+str(self.x)+' '+str(self.y)+'
         ↪  '+str(self.r)+' 0 1 0 N\n')
```

- wire.py- It creates the wires and connectors in order to connect the components. It takes inputs from the parser.py file and splits the wire configuration from the strings and creates new wires and connection required by the parser.py. Parsing wires and connections in wire.py:

```
def parseWire(input_stream, wires):
    line = input_stream.readline().strip()
```

```
    while(line[0]!='@'):
    #print('parsing wire', line)
        if line[0] == 's':
            #print('Yes')
            string = line
            t,x1,y1,x2,y2 = string.split()[:-1]
            x1 = int(x1)
            y1 = int(y1)
            x2 = int(x2)
            y2 = int(y2)
            w = Wire(x1*MULT, y1*MULT, x2*MULT, y2*MULT)
            wires.append(w)
    line = input_stream.readline().strip()
    return input_stream
def parseConn(input_stream, conns):
    line = input_stream.readline().strip()
    while(line[0]!= '@'):
        if line[0] == 'j':
            string = line
            t,x1,y1 = string.split()
            x1 = int(x1)
            y1 = int(y1)
            c = Connector(x1*MULT, y1*MULT)
            conns.append(c)
        line = input_stream.readline().strip()
    return input_stream
```

- misc.py- While component files create the orientation and direction of the pins, misc.py maps the numbers, names, values, etc to the pins. It also maps the names of the plots and prints. Mapping of components in misc.py:

```
#print('last3')
if ref == 'F':
    for i in range(0,len(c.pins)):
        if c.pins[i].n == '1':
            c.pins[i].n  = '3'
            continue
        if c.pins[i].n == '2':
            c.pins[i].n = '4'
            continue
        if c.pins[i].n == '3':
            c.pins[i].n = '1'
            continue
        if c.pins[i].n == '4':
            c.pins[i].n = '2'
            continue
```

```
        return
    #print('last2')
    if ref == 'G':
        for i in range(0,len(c.pins)):
            if c.pins[i].n == '1':
                c.pins[i].n  = '3'
                continue
            if c.pins[i].n == '2':
                c.pins[i].n = '4'
                continue
            if c.pins[i].n == '3':
                c.pins[i].n = '1'
                continue
            if c.pins[i].n == '4':
                c.pins[i].n = '2'
                continue
    return
```

3.2 Conversion of Library Files

Except the libParser.py and the component.py, all other programs and files used in conversion of PSpice Library(slb) files to KiCad library(lib) files are the same. So here we have shown the implementation of libParser.py and component.py.

3.2.1 libParser.py

The purpose of libParser.py is same as that of parser.py. The libParser.py maps the components of PSpice library to that of the KiCad library. It also uses the same component.py file as discussed above. The implementation is done as shown below:

- Imports all the necessary associate files.
- The parser.py creates .lib files for KiCad in the directory specified by the user.
- It creates a directory.
- Reading the components from the PSpice library file and mapping them to the KiCad library file using the component.py file.

The block of creating library files for KiCad in libParser.py is shown below:

```
libDescr  = 'EESchema-LIBRARY Version 2.3  Date: \n#encoding utf-8\n'
nameAppend  = '_PSPICE'
REMOVEDCOMPONENTS = ['TITLEBLK', 'PARAM', 'readme', 'VIEWPOINT', 'LIB',
↪  'copyright', 'WATCH1', 'VECTOR', 'NODESET1']
for fcounter in range(1, len(sys.argv[1:])+1):
    input_file = open(sys.argv[fcounter], 'r+')
    fbasename = os.path.basename(sys.argv[fcounter])
    flname = fbasename[:fbasename.find('.')] + '.lib'
    flib = open(flname, 'w+')
    print('Library file name: ',flname)
    flib.write(libDescr)
    line = skipTo(input_file,'*symbol')
    print('Parser',line)
    '''
    while(line != '' and '*symbol' not in line):
        line = input_file.readline().strip()
        print(line)
    '''
    while(line != '__ERROR__'):
        #print(input_file.tell())
        #print('Compo line',line)
        d = line.find(' ')
        cnametmp = line[d+1:]
        #print('cnametmp',cnametmp)
        d = cnametmp.find(' ')
```

```
        if d == -1:
            cname = cnametmp
        else:
            cname = cnametmp[0:d]
        #print('cname->',cname)
        fileTMP^^I= open(sys.argv[fcounter])
        c = Component(fileTMP, cname)
        #print(c.ref)
        fixComp(c)
        #print('After fixComp',cname, 'ref=', c.ref)
        write = True
        for i in range(len(REMOVEDCOMPONENTS)):
            if cname == REMOVEDCOMPONENTS[i]:
                write = False
                break
        #print('write->', write)
        #print('line->', line)
        if write:
            c.type_ = c.type_ + nameAppend
        c.print(flib)
        '''line = input_file.readline().strip()
        while(line != '' and '*symbol' not in line):
            line = input_file.readline().strip()
            print(line)
        '''
        line = skipTo(input_file, '*symbol')
    flib.write('#\n#End Library\n')
    flib.close()
```

3.2.2 component.py

The purpose of the this program is to take inputs of components of PSpice Library from the libParser.py and map the components to the KiCad library and return the mapped components to the libParser.py file.

Its implementation is as follows:

- Import the required associate programs and files
- It maps the coordinates and type of components of PSpice .slb to that of KiCad .lib.
- It scalescaling the coordinates as the coordinates of PSpice are very small than that of KiCad.
- It also maps the orientation and directions of the pins from Pspice to KiCad using the if else structure

- It also maps the pins of the components in PSpice library to that of the KiCad library
- It calls functions from different classes like attribute.py
- All the mapping done and the data obtained is used by the libParser.py to write to the KiCad library file.

```
def print(self, output_stream, shiftx, shifty):
    output_stream.write("X"+" "+"~ "+str(self.n)+" "+str(self.x-shiftx)+"
     ↪  "+str(self.y-((-1)*shifty))+" ,! "+str(self.length)+" ")
        if self.orientation == 'h':
            output_stream.write('R')
        elif self.orientation == 'u':
            output_stream.write('L')
        elif self.orientation == 'v':
            output_stream.write('U')
        elif self.orientation == 'd':
            output_stream.write('D')
        output_stream.write(' 30 30 0 1 ')
        if self.elec_type == 'i':
            output_stream.write('I\n')
        elif self.elec_type == 'o':
            output_stream.write('O\n')
        elif self.elec_type == 'p':
            output_stream.write('W\n')
        elif self.elec_type == 'x':
            output_stream.write('P\n')
        elif self.elec_type == 'b':
            output_stream.write('B\n')
        else:
            output_stream.write('P\n')
```

Chapter 4

Results

4.1 Example 1

4.1.1 PSpice Schematic and Results

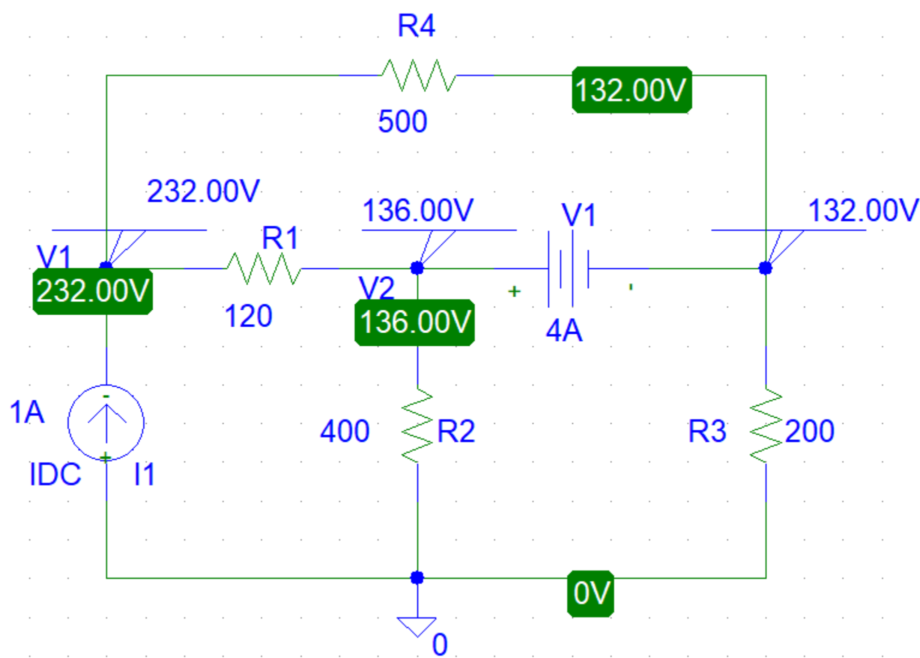

Figure 4.1: PSpice Schematic Example 1

4.1.2 KiCad Schematic

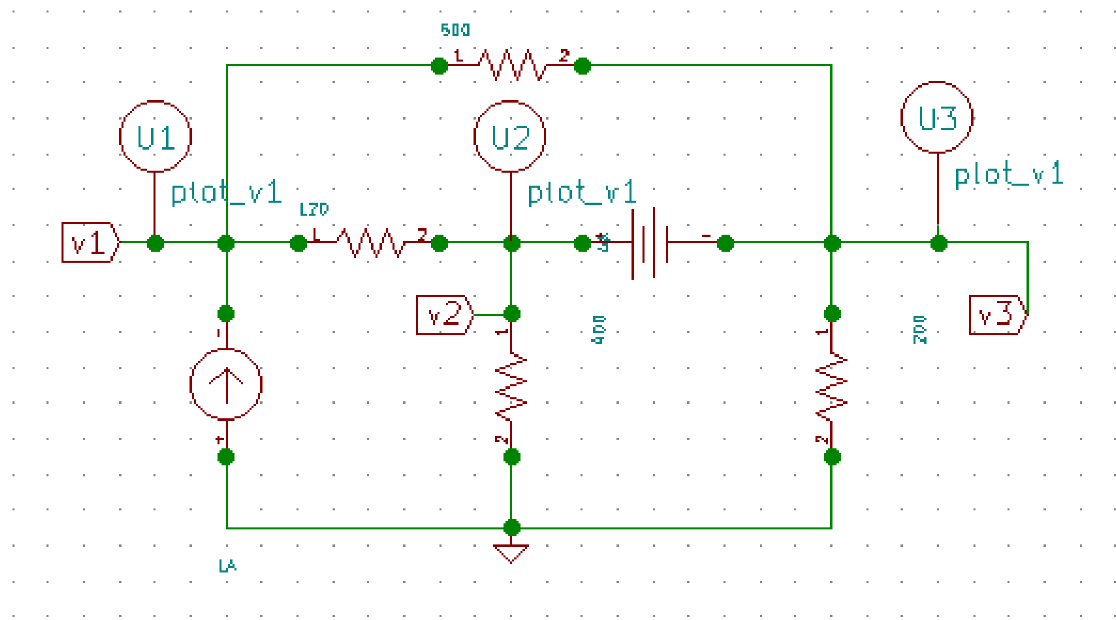

Figure 4.2: KiCad Schematic Example 1

4.1.3 KiCad Results

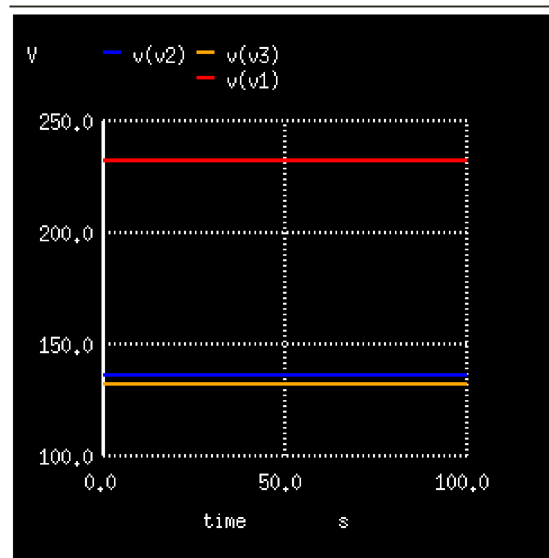

Figure 4.3: KiCad Results Example 1

4.2 Example 2

4.2.1 PSpice Schematic and Results

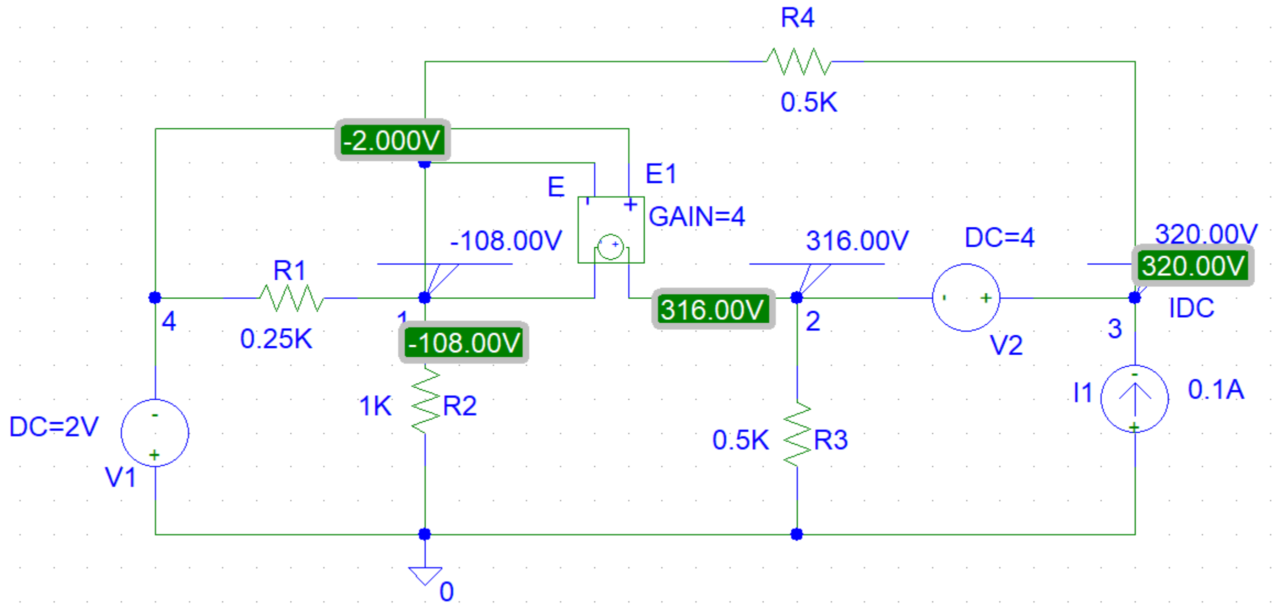

Figure 4.4: PSpice Schematic Example 2

4.2.2 KiCad Schematic

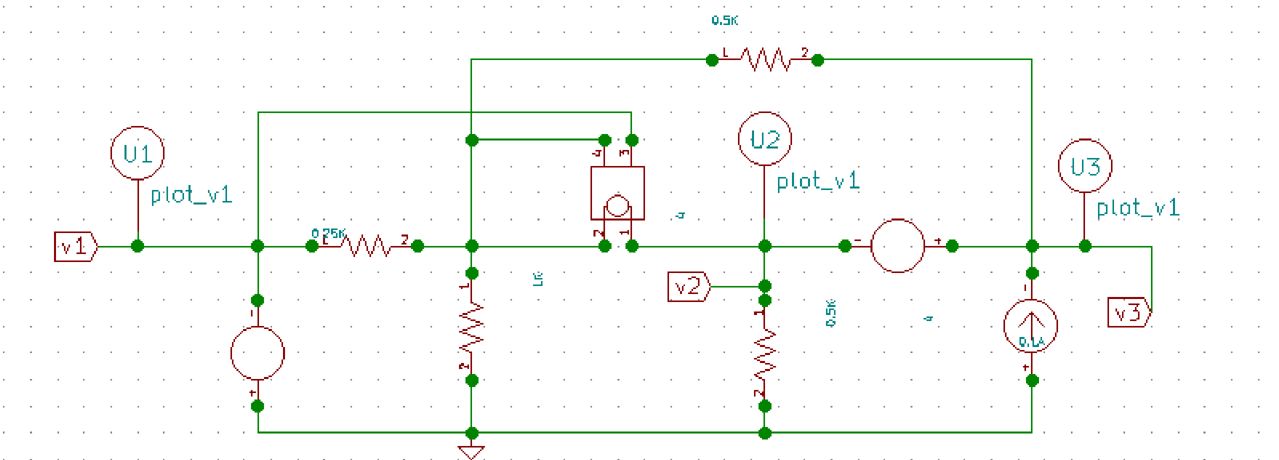

Figure 4.5: KiCad Schematic Example 2

4.2.3 KiCad Results

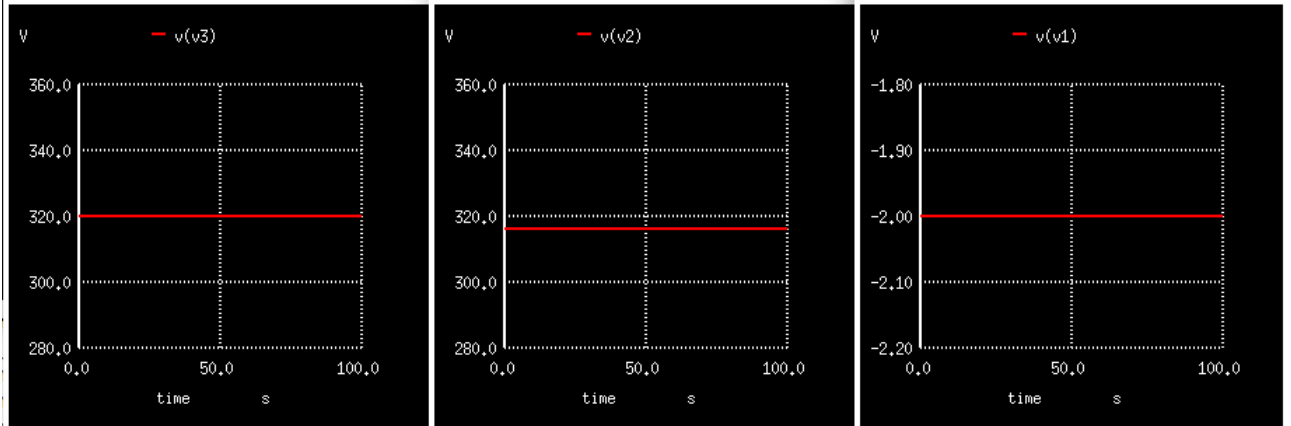

Figure 4.6: KiCad Results Example 2

4.3 Example 3

4.3.1 PSpice Schematic and Results

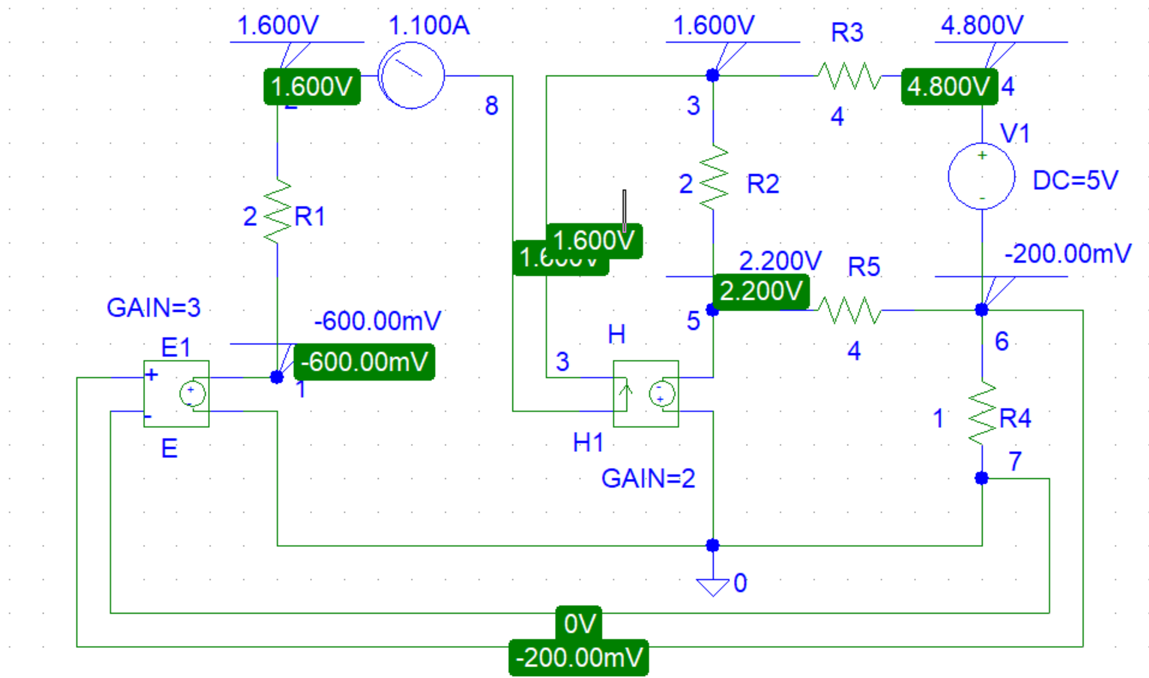

Figure 4.7: PSpice Schematic Example 3

4.3.2 KiCad Schematic

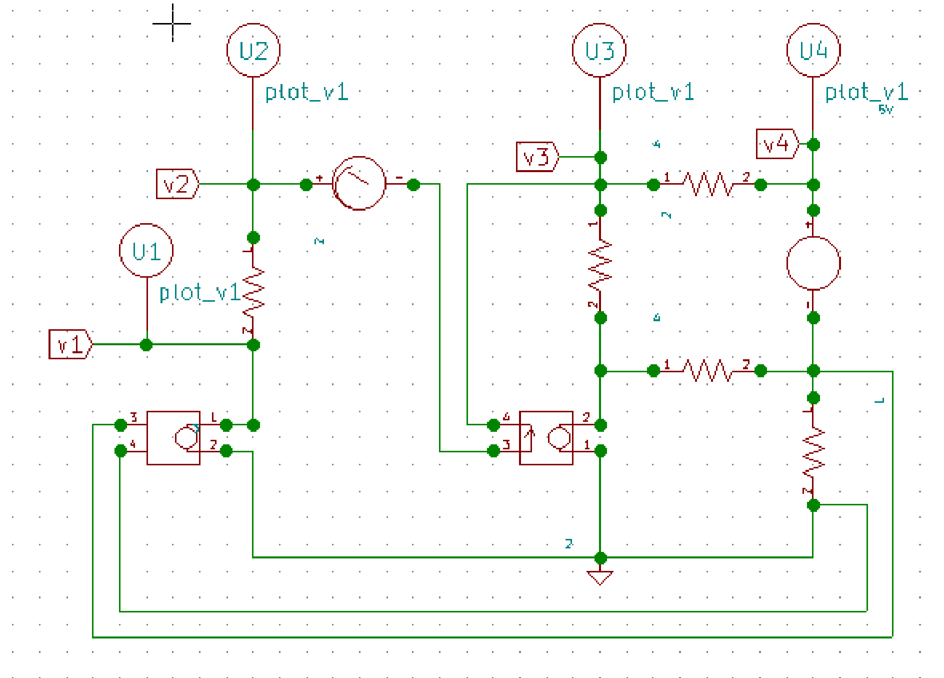

Figure 4.8: KiCad Schematic Example 3

4.3.3 KiCad Results

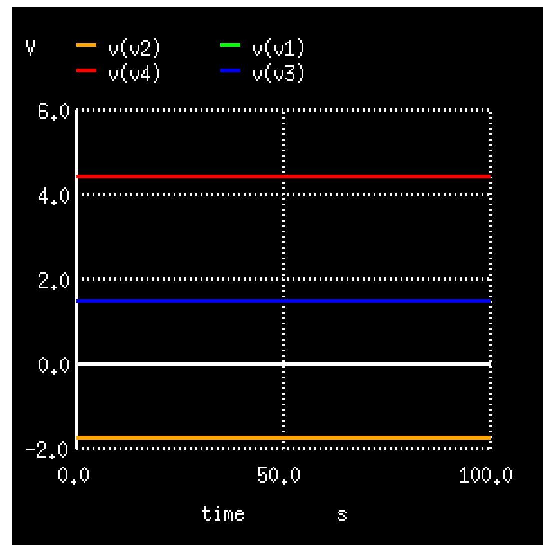

Figure 4.9: KiCad Results Example 3

4.4 Example 4

4.4.1 PSpice Schematic and Results

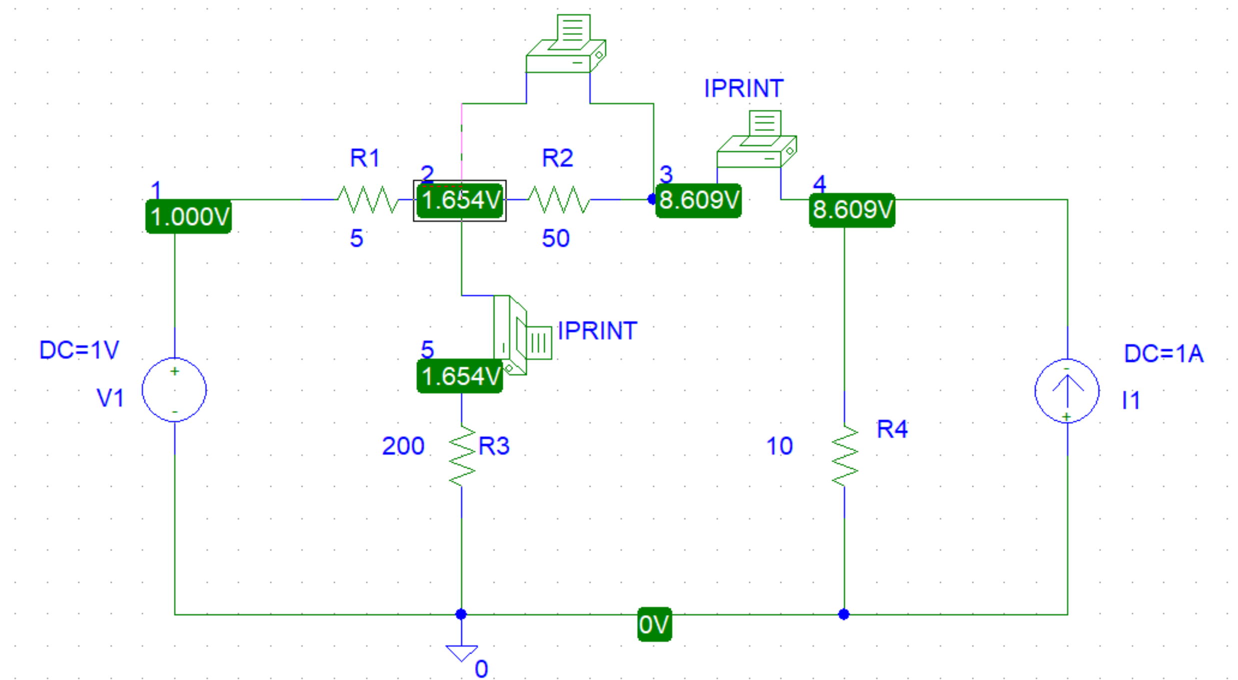

Figure 4.10: PSpice Schematic Example 4

4.4.2 KiCad Schematic

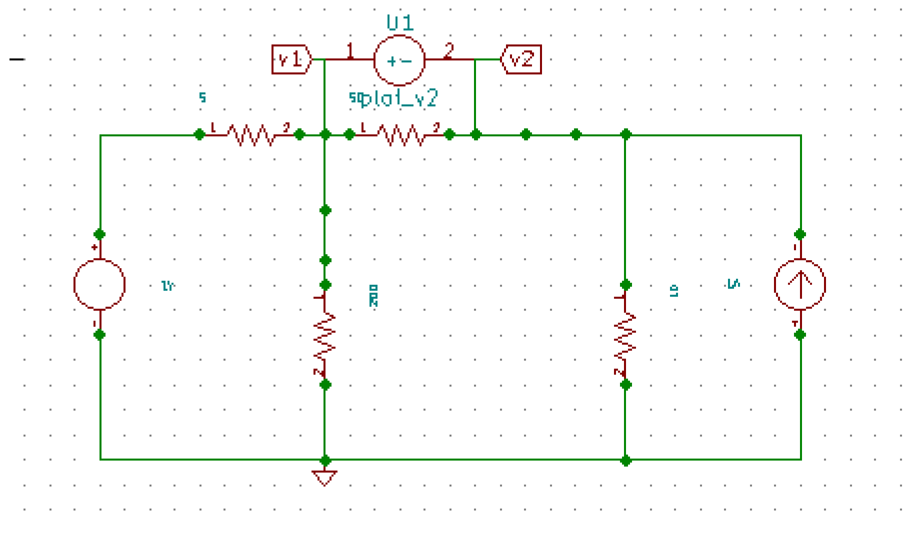

Figure 4.11: KiCad Schematic Example 4

4.4.3 KiCad Results

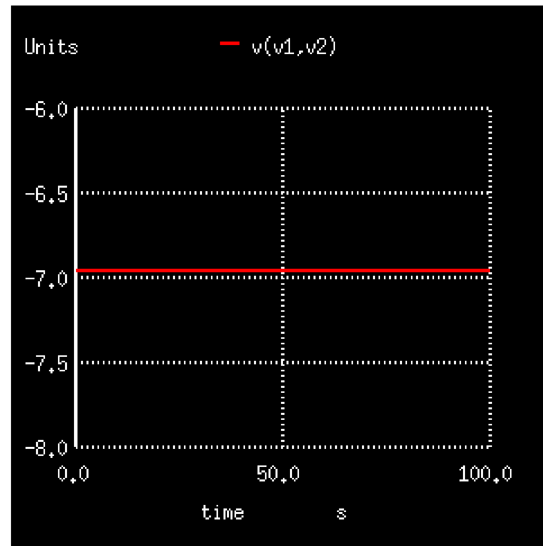

Figure 4.12: KiCad Results Example 4

4.5 Example 5

4.5.1 PSpice Schematic and Results

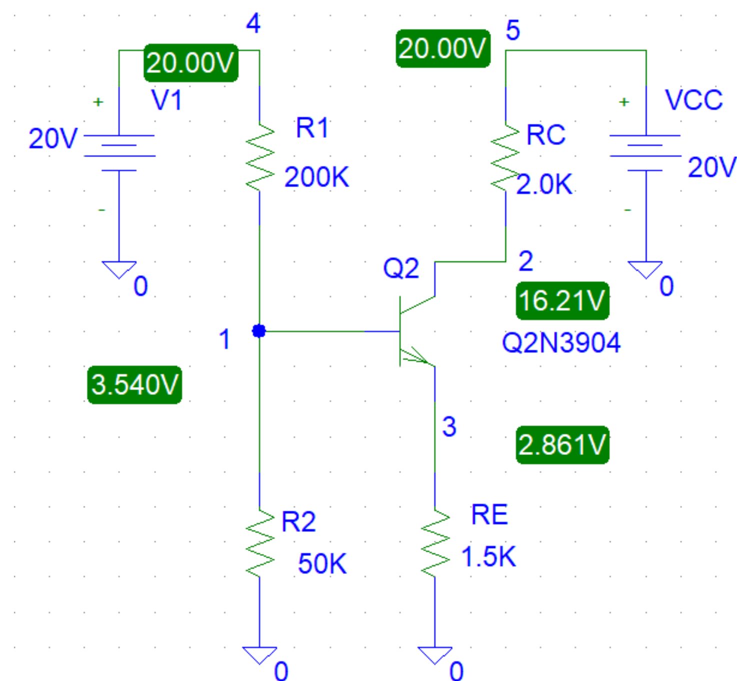

Figure 4.13: PSpice Schematic Example 5

4.5.2 KiCad Schematic

Figure 4.14: KiCad Schematic Example 5

4.5.3 KiCad Results

Figure 4.15: KiCad Results Example 5

4.6 Example 6

4.6.1 PSpice Schematic

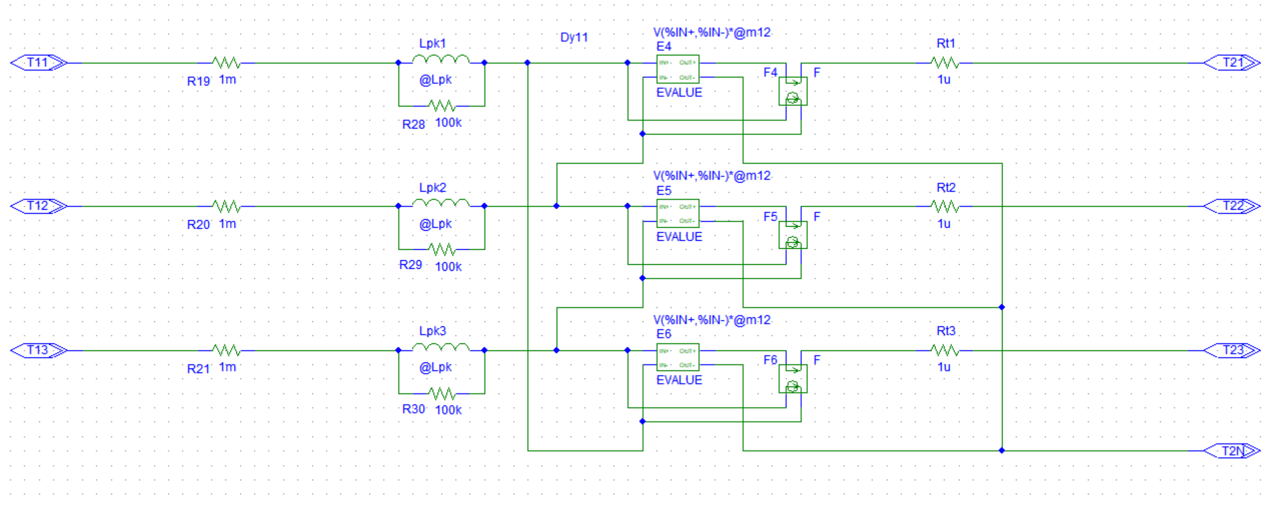

Figure 4.16: PSpice Schematic Example 6

4.6.2 KiCad Schematic

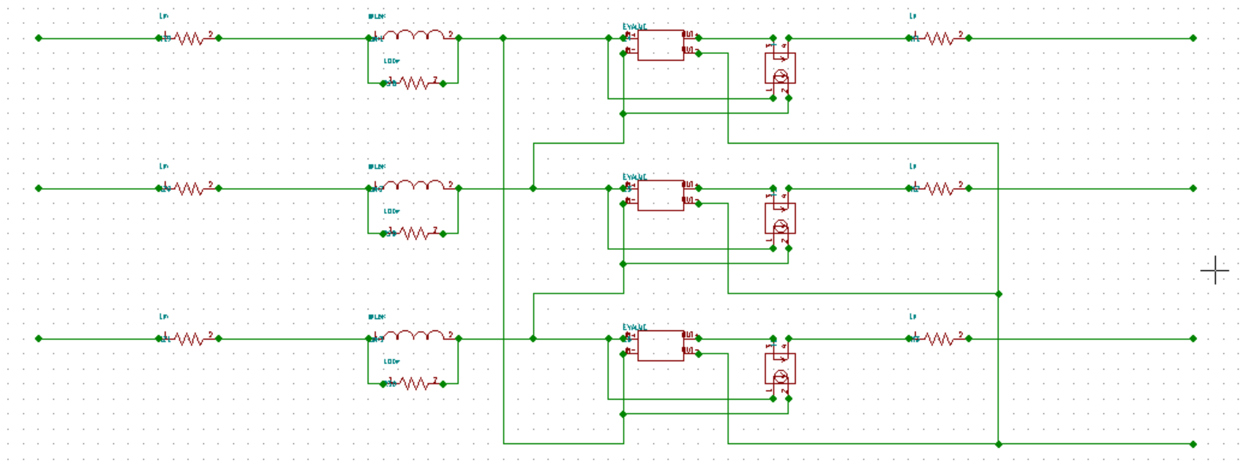

Figure 4.17: KiCad Schematic Example 6

For more examples, please refer to the following GitHub Repositories:

1)https://github.com/Eyantra698Sumanto/PSpice-to-KiCad-Converted-Examples.git
2)https://github.com/FOSSEE/eSim_PSpice_to_KiCad_Python_Parser.git

Chapter 5

Workflow

The steps to be carried out for executing the files are:

- Make sure the python compiler is installed in the PC otherwise install it.
- Clone or Download the *eSim_PSpice_to_KiCad_Python_Parser* from the Git in the Home folder(or any other folder) of the local computer.
- Change the directory where the downloaded folder is located
- To convert the PSpice library(slb) files to KiCad library(lib) files enter the command in the terminal:
  $ sudo python3 parser.py <path/to/pspice-schematic.sch> <path/to/output-project-name-without-extension>
- To convert the PSpice schematic files to KiCad schematic files enter the command in the terminal:
  $ sudo python3 parser.py <path/to/pspice-schematic.sch> <path/to/output-project-name-without-extension>
- The PSpice library and schematic files get converted

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

We were able to successfully convert the PSpice schematic and library files to KiCad schematic and library files. The user can open the schematic and generate PCB design. If the user wants to simulate the circuit then the user has to change the reference according to the Ngspice reference. The parser automatically changes the reference for device component and linear controlled devices.

6.2 Future Work

Most of the DIP packages are referred as U (user) reference in PSpice and KiCad libraries. To make it Ngspice compatible the reference has to be changed to x(subcircuit). As of now this is done manually. In future this task will be automated.

Bibliography

[1] OrCAD Official website. 2020.
URL: https://www.orcad.com/pspice-free-trial/

[2] Github Official Website
URL: https://github.com/FOSSEE/Pspice-Kicad-Converter.git

[3] Wikipedia Official Website. 2020.
URL: https://en.wikipedia.org/wiki/KiCad/

[4] eSim Official website. 2020.
URL: https://esim.fossee.in/

[5] Technopedia Official Website. 2020.
URL: https://www.techopedia.com/definition/3853/parse

[6] Python Official Webite. 2020.
URL: https://www.python.org/

[7] Linux Official Website. 2020.
URL: https://www.linux.org/

[8] KiCad Official Website. 2020.
URL: https://KiCad-pcb.org/

[9] PSpice Official website. 2020.
URL: https://www.pspice.com/