A New Methodology and CAD Programs to Detect Two Serious Faults in VLSI Design: HV/LV Connection Faults and Floating Gate Faults.

QING K. ZHU

Electrical Engineering Department

International Technological University

355 W. San Fernando St.,
San Jose, CA 95113, USA

USA

qkzhu@yahoo.com

Abstract: - This paper presents the new methodology and CAD programs to detect two serious faults in VLSI design: HV/LV connection faults and floating gate faults. A hierarchical circuit netlist is flattened in order to trace the connectivity of MOS devices in hierarchically designed circuits. Programs were coded in Python and table-look up techniques were used to speed up the program run. Program flows and specification files are discussed. Specification file commands allowed designers to waive non-critical faults after reviewing them in the design. We developed GUI capability for highlighting nets in the schematic window. GUI helps designers review and fix faults in Cadence design environment. Programs and GUI capability have been applied in one industry project.

Note: This paper is the extended version of a Draft Paper that was presented in the conference: "2017 MIXDES - 24th International Conference on Mixed Design of Integrated Circuits and Systems".

Key-Words: - VLSI, Design, Multiple voltages, HV/LV, Floating gate, CAD, Tapeout.

1 Introduction

Low power and mixed signal requirements drive the application of dual or multiple supply voltages in VLSI designs. Dual-voltage circuits need special level shifter buffers between HV devices and LV devices [1]. Direct connections between HV and LV devices are not allowed, which can cause significant circuit errors. Floating gate faults occur when one gate terminal of CMOS devices is floating or not connected to other source/drain terminals of CMOS devices. Note that I/O ports can be waived for floating gate faults since they will be connected in the higher-level design. Floating gates may cause the instability of the circuit due to noise coupling from neighboring lines [2,3]. In general, floating gates in the circuit (except I/O ports) have to be connected either to power net or ground net that is called tie-high/tie-low techniques [4].

This paper describes our methodology and developed programs to detect the above two design faults: HV/LV connection faults and floating gate faults in hierarchically designed VLSI circuits. Methods to deal with multiple voltage domains have been investigated in [6,7,8]. We have not been aware of academic publications on methods how to detect HV/LV connection and floating gate faults in VLSI design. Commercial tools such as Synopsys CustomSim are available to detect HV/LV connection faults. Assura tool from Cadence Design System Inc. can detect floating gates using the command ercCheckFloatingDevices [11]. Commercial simulation tools are expensive to purchase and take CAD efforts to setup. EDA licenses are limited in industry design environment. Meanwhile designers would like to check the above two faults frequently in order to fix circuit errors in the early design stage.

This paper is organized in the following sections. Section 2 defines two types of faults: HV/LV connection faults and floating gate faults. Section 3 shows our methodology and GUI capability. Program flows and specification file commands are explained. Section 4 describes hierarchical netlist flattening process and look-up tables for speeding up programs. Section 5 shows experimental results of test circuits from one industry project. Section 6 gives conclusions.

2 Problem Formulation

Semiconductor devices in dual supply voltages environment can be classified in the following two types:

- *HV device*: the device is connected to high VDD supply network.
- *LV device*: the device is connected to low VDD support network.

HV/LV connection fault is defined as follows:

- HV device is directly connected to a LV device or vice versa in the circuit.
- HV device is connected to low VDD supply network.
- LV device is connected to high VDD supply network.

Figure 1 illustrates the definition of HV/LV connection faults. “Net 1” is marked as a HV/LV fault since high-voltage devices “a” or “b” are directly connected to low-voltage devices “c” or “d”. Notice that “Net 1” crosses two circuit blocks in the design. “Net 3” is not an HV/LV connection fault since a voltage level shifter between low-voltage devices “e” or “f” and high-voltage devices “g” or “h” in the circuit.

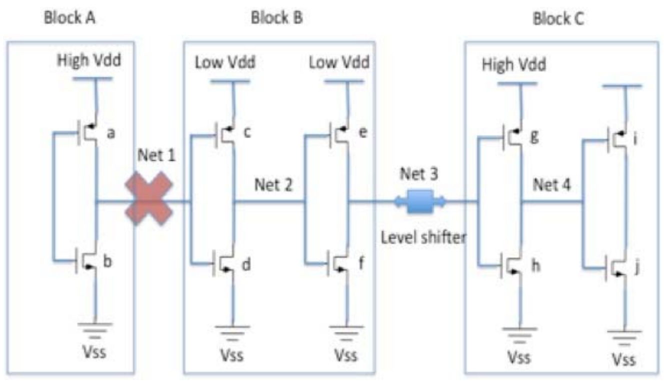

Fig. 1 HV/LV Connection Faults.

Floating gate fault is defined as follows:

- Gate terminal of a device is not connected to any source/drain terminals from other devices in the circuit.
- I/O ports are waived from floating gate faults.

Figure 2 illustrates the definition of floating gate fault. “Net 5” is a floating gate fault since it is not connected to an I/O port and also not connected to source/drain terminals of other devices. “Net 1” is not flagged as a floating gate since it is an input port.

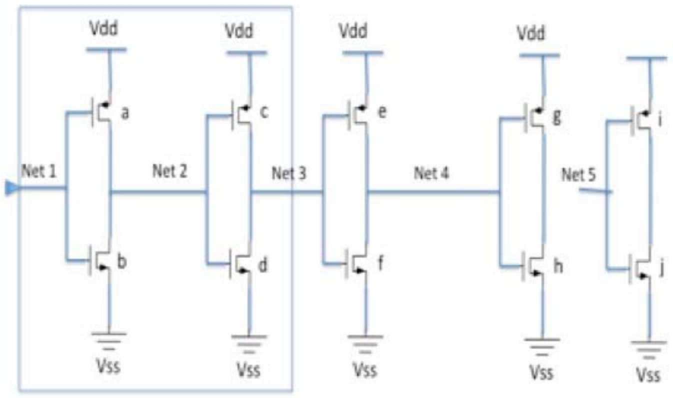

Fig. 2 Floating Gate Faults.

3 Flows and GUI

HV/LV connection checking program reads in a specification file containing the following commands:

- *HV* – a list of high voltage device models.
- *LV* – a list of low voltage device models.

- *SKIPCELL* – a list of master cells or sub-circuit names which will be skipped in the checking.
- *SKIPINST* – a list of flattened instance names, which will be skipped in the checking.
- *SKIPNET* – a list of flattened net names, which will be skipped in the checking.

Specification File of HV/LV Connection Fault Checking:

```
# Spec file for checkHV.py

HV pch_hvt nch_hvt

LV pch_lvt nch_lvt

SKIPCELL inv*d3

SKIPNET */gnd

SKIPINST xxx/xxx/*_digital*

SKIPDEVICE  top/*/IO_buffers/*

# SKIPNET top/XI10/sa/*
```

The above HV command lists high-supply voltage devices. LV command lists low-supply voltage devices. The specification file accepts hierarchy specification (/) and "*" notation for pattern matching of cell names or net names. Comments (#) are accepted in the specification file. Some faults reported by the program are non-critical and they can be waived in the design after careful reviewing from design team. The waived cells or waived nets can be defined in commands SKIPCELL or SKIPNET. For example, HV devices connected to the lower supply voltage could be waived in some designs, since HV devices have the impact on the circuit delay but not the circuit reliability.

The general methodology to fix and waive HV/LV connection faults is as follows. Designers run the program to report HV/LV faults. Designers fix real faults in the circuit. Designer can waive some faults after reviewing results from the program run. A list of cells or nets are then added in the specification file using SKIPCELL or SKIPNET commands. Then rerun the program using the new specification file and modified circuits. The above process is iterated until all the faults are either fixed or waived for the specific circuit. GUI capability has been developed and provided to designers to speed up the above reviewing and fixing process.

Figure 3 shows the flow to detect HV/LV connection faults. Floating gate fault program reads in another specification file. It only has one command SKIPNET that specifies a list of nets to be waved during the checking of floating gates. For example, I/O nets on the top block can be waived, since I/O ports can be connected in the higher-level design. Appendix II shows the Python code of floating gate checking program. A net, belonging to GATENET[net] but not in SOURCENET[net] or DRAINNET[net] or IO[net] in the program, is identified as floating gate fault. Designers run the program to review all the floating gate faults. Some non-critical floating gate faults can be waived by using the SKIPNET command in the specification file. The process is iterated until all the floating gate faults are reviewed, fixed or waived in the current design.

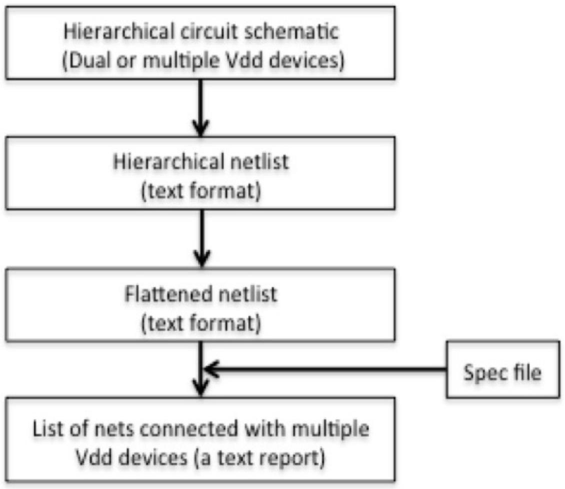
Hierarchical circuit schematic
(Dual or multiple Vdd devices)
Hierarchical netlist
(text format)
Flattened netlist
(text format)
Spec file
List of nets connected with multiple
Vdd devices (a text report)

Fig. 3 Flow to Detect HV/LV Connection Faults.

Figure 4 shows the flow to detect floating gate faults.

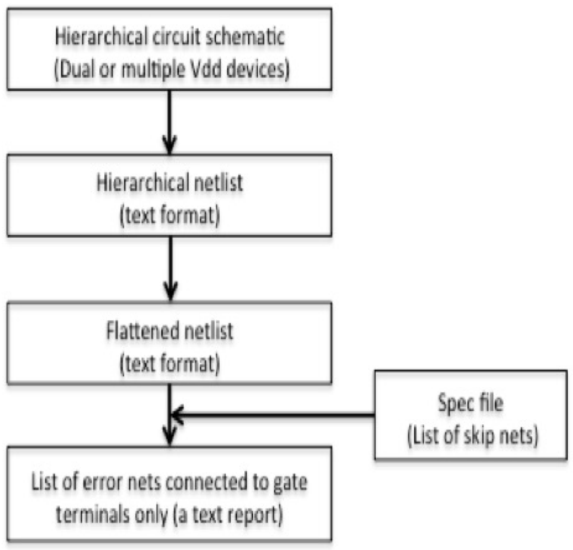
Hierarchical circuit schematic (Dual or multiple Vdd devices)
Hierarchical netlist (text format)
Flattened netlist (text format)
Spec file (List of skip nets)
List of error nets connected to gate terminals only (a text report)

Fig. 4 Flow to Detect Floating Gate Faults.

It is hard to identify and locate HV/LV or floating gate faults in a complex circuit based on textual report files. GUI capability was developed to review reported faults in Cadence design environment. GUI was written in Cadence Skill Language [14]. It reads in errors in report files from HV/LV or floating gate programs in a window panel. When the designer clicks on one specific error in GUI panel, it automatically descends into the corresponding schematic level and highlights the fault net. GUI panel is connected to Cadence schematic window through Skill-based inter-process message communications [15]. GUI also provides the zoom-in capability to narrow down the location of fault nets in Cadence schematic window. GUI improves the productivity significantly for the purpose of locating and reviewing HV/LV and floating gate faults especially in hierarchically designed circuits.

Figure 5 shows the snapshot of GUI panel communicated with Cadence circuit design window. Figure 6 shows the entire flow to use GUI for reviewing and fixing HV/LV connection faults.

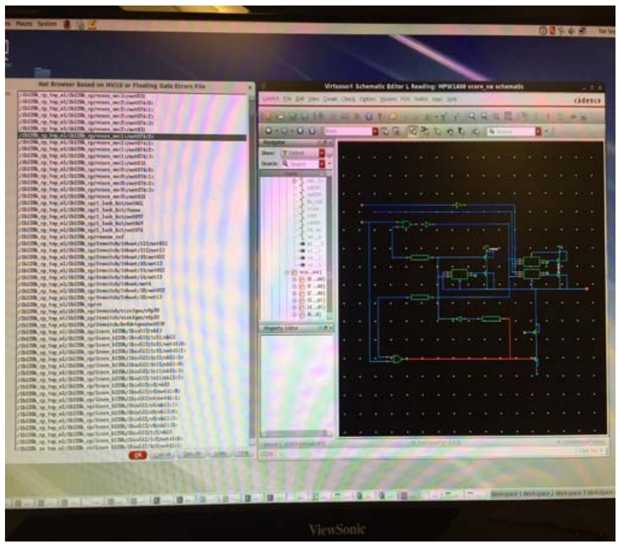

Fig. 5 Snapshot of GUI Panel and Circuit Design Environment.

4 Circuit Netlist Flattening and Look Up Tables

Hierarchical design methodology is commonly used in VLSI projects. Cadence Virtuoso design platform supports hierarchical design methodology based on design cells and views [12-13]. Circuit netlist is a textual file generated from the schematic view of the design in Cadence Virtuoso platform. The netlist defines the hierarchy of sub-circuits and connectivity of sub-circuits and devices in the design. The most popular formats of circuit netlists in the semiconductor industry are CDL netlist and SPICE netlist. We take CDL (Circuit Description Language) netlist format as the input to programs. To detect HV/LV connection faults, we may flatten the hierarchical netlist so we can trace the connectivity of devices in hierarchically designed circuit. The netlist flattening algorithm is critical to the accuracy to detect faults.

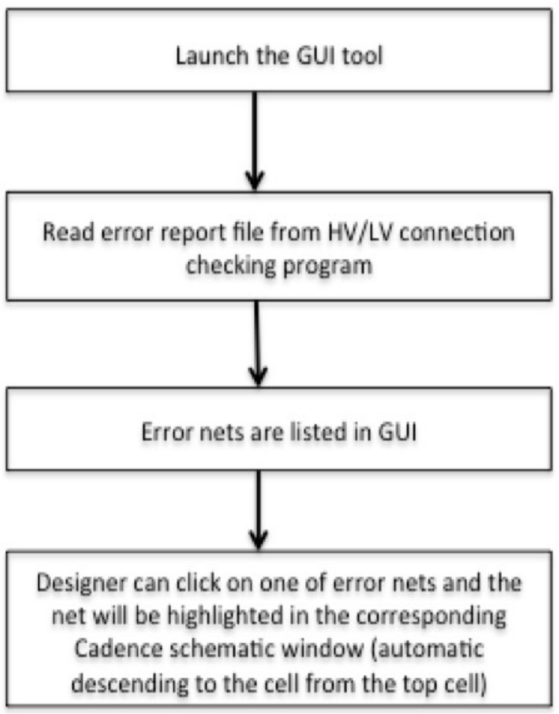
Launch the GUI tool
Read error report file from HV/LV connection checking program
Error nets are listed in GUI
Designer can click on one of error nets and the net will be highlighted in the corresponding Cadence schematic window (automatic descending to the cell from the top cell)

Fig. 6 Flow to Launch and Execute in GUI.

Two subroutines PP() and flattenInstCDL() implemented in Python code are shown below that are recursively called that starts from the top-level cell down to leave cells in the hierarchy of the netlist as shown in Figure 7. PP() subroutine passes port labels from parent cell to the child cell. As shown in Figure 7, annotations P1-P8 indicate the mapping of port names from the parent cell to child cells. For example, P1 mapping function transfers net names in cell “Top” to port names in subcell “Top/XI0” as follows: {net1->in1, gnd->in2, gnd->out1, net2->out2, gnd->in_gnd, vdd->in_vdd}. The flattened device name is named as <top_cell>/<child_cell>/…/<device> in the flattened netlist. The flattened netlist is then used by programs to acquire the connectivity of devices in the circuit. The top-level circuit may contain thousands or millions of MOS devices in modern VLSI circuits, the size of flattened netlist could be extremely large. Lookup tables are used in problems to store critical data that will be reused multiple times in the program run. We expect look-up tables can speed up the problem for large-size circuits. Experimental results in Section 5 shows one test circuit with about 0.2M devices can be finished in about 5 second including both steps of netlist flattening and HV/LV connection faults detection.

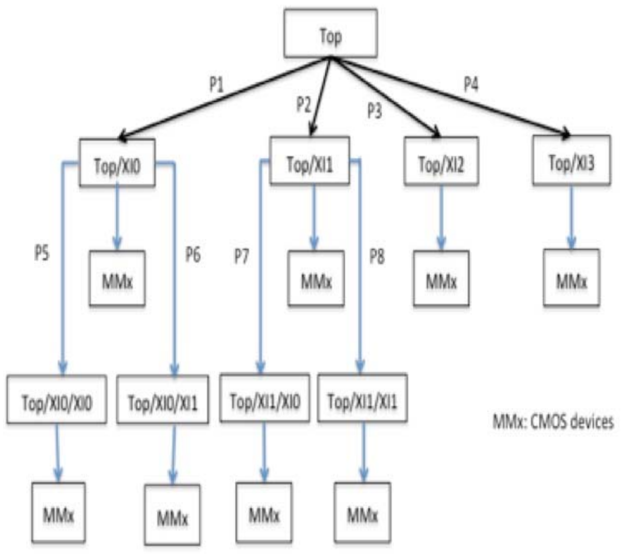

Fig. 7 Hierarchy Tree of Circuit Netlist.

Two Subroutines for Recursive Port Labels Passing and Instance Names Flattening from Parent Cell to Child Cells in a Hierarchical Netlist:

```
#
# PP() —
#
#   Find the root net in hierarchical passing of port
#   labels in CDL netlist.
#
def PP(label, PORTMAP):
    label1 = label
    while label1 in PORTMAP.keys():
        label1 = PORTMAP[label1]
    return(label1)


#
# flattenInstCDL() —
#   Flatten out the current instance recursively in a
flattened netlist format
#
def flattenInstCDL(curInst, curCell, PORTMAP,
                         SKIPCELL, skipInsts,
                  instmaster, INST, PORT):
for line in INST[curCell]:
  if line[0] == 'M':
    words = line.split()
    inst = curInst + "/" + words[0]
    label1 = curInst + "/" + words[1]
    label2 = curInst + "/" + words[2]
    label3 = curInst + "/" + words[3]
    label4 = curInst + "/" + words[4]
    label1 = PP(label1, PORTMAP)
    label2 = PP(label2, PORTMAP)
    label3 = PP(label3, PORTMAP)
    label4 = PP(label4, PORTMAP)
    f1w.write("%s %s %s %s %s" %(inst,
              label1, label2, label3, label4))
    for i in range(5, len(words)):
       f1w.write(" %s" %(words[i]))
       f1w.write("\n")
  elif line[0] == 'X':
    words = line.split()
    childCell = getChild(words)
    if childCell != "" and childCell not in
       SKIPCELL:
       childPortList = PORT[childCell]
       if instmaster == 1:
          childInst = curInst + "/" + words[0] +
                    "/" + childCell
       else:
          childInst = curInst + "/" + words[0]
       for i in range(0, len(childPortList)):
          childLabel = childInst + "/" +
                       childPortList[i]
          parentLabel = curInst + "/" +
                        words[i+1]
          PORTMAP[childLabel] = parentLabel
          childInst2 = curInst + "/" + words[0]
          if checkSI(childInst, skipInsts) == 0
             and
             checkSI(childInst2, skipInsts) == 0:
               flattenInstCDL(childInst,
                              childCell,
                              PORTMAP,
                              SKIPCELL,
                              skipInsts,
                              instmaster, INST,
                              PORT)
```

Programs detecting HV/LV connection or floating gate faults have to be efficient to process large-size circuits. We adopt the table look-up technique in our programs. Look-up tables are built up only one time during the execution and they are reused during the program run. Table 1 lists look-up tables employed in programs. Modern Linux machines contain about 10G-100G bytes memory sizes. HVDEV and LVDEV tables in Table 1 need 10M bytes in the memory size for one circuit with 10M devices. Other tables in Table 1 need much smaller sizes compared to HVDEV and LVDEV tables.

Table 1. Look-up Tables in Programs.

HV/LV connection faults	Floating gate faults
HVDEV[device]: a high-voltage device. *LVDEV[device]*: a low-voltage device. *SKIPCELL[cell]*: a master cell to skip the checking. *PORT[cell]*: list of port	*PORT[cell]*: list of port names associated with the master cell in the original netlist. *INST[cell]*: list of instances inside the master cell in the original netlist. *INSTDEVICE[inst]*:

names associated with the cell in the original netlist. *INST[cell]*: list of instances in the cell in the original netlist. *INSTDEVICE[inst]*: device model corresponding to the instance in the flattened netlist. *HVNET[net]*: list of HV devices connected to this net in the flattened netlist. *LVNET[net]*: list of LV devices connected to this net in the flattened netlist.	device model corresponding to the instance in flattened netlist. *SOURCENET[net]*: list of source terminals of devices connected to the net. *GATENET[net]*: list of gate terminals of devices connected to the net. *DRAINNET[net]*: list of drain terminals of devices connected to the net. *IO[net]*: list of IO nets. FLATNET[net]: list of flattened nets.

5 Experimental Results

The described programs have been applied in one industry project. Programs and specification files are enhanced during the design process. Specification file commands are continually refined based on the feedback from design team in order to review and waive errors in an efficient way. GUI debug tool was requested by design team for identifying and locating faults in Cadence design environment to improve the productivity.

Table 2 shows experimental results and run time of HV/LV connection fault program in circuits from the project. Table 3 shows experimental results and run time of floating gate program. CPU time was measured in Intel Xeon CPU E5-3630 @ 2.4GHZ machines and Redhat 6.8 Linux operating system. Running speeds of programs are reasonably fast even for large-size circuits including both steps of netlist flattening and detection of faults. One circuit containing ~0.2M devices is finished in 5 seconds to check HV/LV connection faults. One circuit containing ~2M devices is finished in 100 seconds to check and report floating gate faults.

Table 2. Experimental Results of HV/LV Connection Fault Program.

	Circuit A	Circuit B	Circuit C
Total devices	204536	1786	3143
HV devices	114208	1784	589
LV devices	90328	2	2554
Total nets	71207	1151	540
HV nets	35871	1150	375
LV nets	36494	4	228
HV/LV faults	1069	2	58
CPU time	5.45s	0.04s	0.15s

Table 3. Experimental Results of Floating Gate Fault Program.

	Circuit A	Circuit B	Circuit C
Total devices	1979170	218428	2260
Total nets	1805322	173228	1160
Floating faults	50	70	55
CPU time	99.46s	5.79s	0.05s

6 Conclusion

This paper describes the new methodology and CAD programs to detect two types of faults in VLSI design: HV/LV connection faults and floating gate faults. Our programs can improve the accuracy for identifying and fixing these two types of faults in the early design stage. Program details and Python procedures for flattening hierarchical circuit netlists are included. Look-up tables are extensively used in programs to improve the speed for large-size netlists.

Reviewing textual report files from program runs can be tedious and easy to miss real faults. We develop GUI capability to identify and highlight errors in Cadence design environment. Programs and GUI have been demonstrated and acknowledged by the design team.

References:

1. I. H. Lysfjord, "Multiple Power Domains", *Department of Electronics and Telecommunications, Norwegian University of Science and Technology*, 2008.
2. H. J. M. Veendrick, "Short-circuit dissipation on static CMOS circuitry and its impact on the design of buffer circuits", *IEEE Journal of Solid-State Circuits*, Vol SC-19, No 4, Aug 1984, pp. 468-473.
3. P. Lee, "Introduction to physical integration and tapeout in VLSIs", *lulu.com*, 2010.
4. V. Kumarreddy, "Dummy transistors connection", *edaboard.com*, 2006.
5. M. Lapedus, "10nm Versus 7nm", *semiengineering.com*, 2016.
6. Efram Rotem, Avi Mendelson, Ran Ginosar, Uri Weiser, "Multiple clock and Voltage Domains for chip multi processors", *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009.
7. M. Igarashi, K. Usami, K. Nogami, F. Minami, Y. Kawasaki, T. Aoki, M. Takano, S. Sonoda, M. Ichida, N. Hatanaka, "A low-power design method using multiple supply voltages", *International Symposium on Low Power Electronics and Design*, 1997.
8. Martin D.F. Wong, "Low power design with multi-Vdd and voltage islands", *7th International Conference on ASIC*, 2007.
9. A. B. Khang, S. Muddu and D. Vidhani, "Noise and delay uncertainty studies for coupled RC interconnects", *Proc. of International ASIC/SOC Conference*, 1999.
10. A. Vittal and M. Marek-Sadowska, "Crosstalk reduction for VLSI", *IEEE Transactions on Computer-aided Design of integrated circuits and systems*. Vol. 16. No. 3, March 1997, pp. 290-298.
11. "Find a list of floating gates in schematic", https://community.cadence.com
12. A. J. L. Martin "Tutorial: Cadence design environment", *http://www2.ece.ohio-state.edu*
13. "Virtuoso® analog design environment user guide", *http://home.engineering.iastate.edu*
14. T. J. Barnes, "SKILL: a CAD system extension language", *27th ACM/IEEE Design Automation Conference*, 1990.
15. "Inter-process communication SKILL functions reference", *http://read.pudn.com*

Appendix I - Python Code of Hierarchical Netlist Flattening and HV/LV Connection Check

```
#!/usr/bin/env python

import sys

import re

import os.path

import fnmatch import fnmatch, fnmatchcase

import time


#

# getChild() —

#

# Get master cell name based on words of instance line

#

def getChild(words):

master = “”

for i in range(0, len(words)):

   if words[i] == “/“:

      master = words[i+1]

return(master)
```

```
#
# XL() —
#
# If ass the XLine in the instance list
# Filter out XR case
#
def XL(line)
status = 0
if line != “”:
    if (line[0] == “X”) and (line[1] == “R”):
        status = 0
   else:
       status = 1
return(status)

#
# PP() —
#
#   Find the root net in the hierarchical passing of port labels in CDL net list
#
def PP(label, PORTMAP):
label1 = label
while label1 in PORTMAP.keys():
     label1 = PORTMAP[label1]
return(label1)

#
# checkSI() —
#   check if it is a skip instance
#
def checkSI(inst, skipInsts):
     skip = 0
     if len(skipInsts) > 0:
          for skipInst in skipInsts:
               if fnmatch(inst, skipInst):
                    skip = 1
    return(skip)

#
# checkSN() —
#   check if it is a skip net
#
def checkSN(net, skipNets):
      skip = 0
      if len(skipNets) > 0:
           for skipNet in skipNets:
                if fnmatch(net, skipNet):
                     skip = 1
      return(skip)

#
# checkSD() —
#   check if it is a skip device
#
#
def checkSD(device, skipDevices):
     skip = 0
     if len(skipDevices) > 0:
          for skipDevice in skipDevices:
               if fnmatch(device, skipDevice):
                    skip = 1
     return(skip)
```

```
#
# Main program
#
start = time.time()
if len(sys, argv) != 3:
    print “ Command: checkHV.py <topCell>.cdl
            checkHV.spec”
    exit()
cdlFile = sys.argv[1]

#
# Merge the continuous lines with beginning “+” in CDL netlist
#
netlist = cdlFile + “.tmp”
if os.path.exists(cdlFile):
    fc1 = open(cdlFile, “r”)
else:
    print(“ERROR: %s does not exist !” %cdlFile)

fc2 = open(netlist, “w”)
lastLine = “”
for line in fc2:
    words = line.split()
    if len(words) > 0:
        if words[0][0:1] == “+”:
            lastLine2 = lastLine.rstrip(‘\n’)
            line2 = line.lstrip(‘+’)
            lastLine = lastLine2 + line2
        else:
      if lastLine != “”:
         fc2.write(“%s” %lastLine)
      lastLine = line
if lastLine != “”:
   fc2.write(“%s” %lastLine)
fc1.close()
fc2.close()

#
# Read the spec file
#
specFile = sys.argv[2]
if os.path.exists(specFile):
   fs = open(specFile, “r”)
else:
   print(“%s does not exists !” %specFile)
   exit()
print(“Spec file: %s” %specFile)
#
# Build map tables from the spec file
#
HVDEV = dict()
LVDEV = dict()
SKIPCELL = dict()
……
fs.close()

# Build map tables for each subckt in CDL net list
#    Skip the subckt in the SKIPCELL table
#    PORT is the list of ports to a subckt
#    INST is the list of instances to a subckt
#
PORT = dict()
INST = dict()
if os.path.exists(netlist):
   fn = open(netlist, “r”)
else:
   print(“%s does not exist !” %netlist)
   exit()
print(“CDL net list: %s” %netlist)
subckt = 0
for line in fn:
words = line.split()
if words[0] = “.SUBCKT”:
   if “.SUBCKT” in line:
      subCkt = 1
      master = words[1]
      portList = list()
      instList = list()
      if master not in SKIPCELL:
         PORT[master] = portList
         INST[master] = instList
         for i in range(2, len(words)):
            portList.append(words[i])
elif words[0] == “.ENDS”:
     subCkt = 0
elif subCkt == 1:
     # Add all instance lines in the instList for INST
     mapping table
     instList.append(line)

if topCell == “”:
   # The last master becomes the top cell
   if master not in SKIPCELL:
      topCell = master
if topCell == “”:
print(“ERROR: top cell name is not defined \n”)
exit
fn.close()

#
# Flatten the CDL netlist in the recursive way
#
flatCDL = topCell + “.cdl.flat”
f1w = open(flatCDL, “w”)
PORTMAP = dict()
flattenInstCDL(topCell, topCell, PORTMAP,
                         SKIPCELL, skipInsts, instmaster,
                         INST, PORT)
print(“Flattened CDL netlist: %s” %flatCDL)
f1w.close()

#
# Report HV/LV connection errors
#
errFile = topCell + “.err.flat”
f2w = open(errFile, “w”)
HVNET = dict()
LVNET = dict()
INSTDEVICE = dict()
FLATNET = dict()
flatNets = list()
f1r = open(flatCDL, “r”)
totalHVDevices = 0
totalLVDevices = 0
totalDevices = 0
for line in f1r:
   words = line.split()
   inst = words[0]
```

```
device = words[5]
INSTDEVICE[inst] = device
totalDevices = totalDevices + 1
if device in HVDEV:
   totalHVDevices = totalHVDevices + 1
elif device in LVDEV:
   totalLVDevices = totalLVDevices + 1
for i in range(1, 5):
   net = words[i]
   if net not in FLATNET:
      FLATNET[net] = 1
       flatNets.append(net)
  if device in HVDEV:
      if net in HVNET:
           hvInstList = HVNET[net]
           hvInstList.append(words[0])
     else:
          hvInstList = list()
          hvInstList.append(words[0])
          HVNET[net] = hvInstList
  elif device in LVDEV:
     if net in LVNET:
          lvInstList = LVNET[net]
          lvInstList.append(words[0])
    else:
         lvInstList = list()
         lvInstList.append(words[0])
         LVNET[net] = lvInstList
  else:
     print("WARNING: Instance %s %s is not
              specified as HV or LV devices \n"
              %(words[0], device))
f1r.close()


#
# Go through HVNET and LVNET to report error nets
#
for net in flatNets:
   if (net in LVNET) and (net in HVNET):
       if checkSN(net, skipNets) == 0:
          f2w.write("ERROR: net %s is connected to
                       both HV and LV devices in the
                       following instances - \n" %net)
          for inst in LVNET[net]:
              if checkSD(inst, skipDevices) == 0:
                 device = INSTDEVICE[inst]
                 vddNet = topCell + "/" + "VDD"
                 if net != vddNet:
                    f2w.write("\tLV device: %s %s\n" %
                                (inst, device))
          for inst in HVNET[net]:
             if checkSD(inst, skipDevices) == 0:
                device = INSTDEVICE[inst]
                vddioNet = topCell + "/" + "VDDIO"
                if net != vddioNet:
                   f2w.write("\tHV device: %s %s\n"
                               %(inst, device))
  else:
        print("WARNING (skip net): net %s is
                 connected to both HV and LV devices
                 but waived in the spec file \n" %net)
f2w.close()
print("HV/LV checking results: %s" %errFile)
print("Statistics: total devices = %d HV devices =
        %d LV devices = %d" %(totalDevices,
totalHVDevices, totalLVDevices))
totalNets = len(flatNets)
totalHVNets = len(HVNET.keys())
totalLVNets = len(LVNET.keys())
print("Statistics: total nets = %d HV nets = %d LV
        nets = %d" %(totalNets, totalHVNets,
        totalLVNets))
print("Checking is finished in %s seconds !"
        %(time.time() - start))
```

Appendix II – Python Code of Floating Gate Program

```
#!/usr/bin/env python
# This python program reads in a CDL netlist and a
specification file.
# It first flattens the CDL netlist. Then detects
floating gate faults based on the flattened netlist
#
import sys
import re
import os.path
import fnmatch import fnmatch, fnmatchcase
import time

#
# Main program
#
start = time.time()
if len(sys, argv) != 2 and len(sys, argv) != 3:
   print " Command: floatGate.py <topCell>.cdl"
   print " Command: floatGate.py <topCell>.cdl
           skipFile"
   exit()

cdlFile = sys.argv[1]

#
# Merge the continuous lines with beginning "+" in
  CDL netlist
#
netlist = cdlFile + ".tmp"
if os.path.exists(cdlFile):
   fc1 = open(cdlFile, "r")
else:
   print("ERROR: %s does not exist !" %cdlFile)

fc2 = open(netlist, "w")
lastLine = ""
for line in fc2:
   words = line.split()
   if len(words) > 0:
      if words[0][0:1] == "+":
         lastLine2 = lastLine.rstrip('\n')
         line2 = line.lstrip('+')
         lastLine = lastLine2 + line2
     else:
         if lastLine != "":
            fc2.write("%s" %lastLine)
         lastLine = line
if lastLine != "":
     fc2.write("%s" %lastLine)
fc1.close()
fc2.close()
```

```
#
# Read spec file
#
skipNets = list()
instmaster = 0
if len(sys.argv) == 3:
    specFile = sys.argv[2]
    if os.path.exists(specFile):
        fs = open(specFile, “r”)
else:
   print(“%s does not exists !” %specFile)
   exit()
   print(“Spec file: %s” %specFile)
   for line in fs:
       if len(line) > 1:
          words = line.split()
          if words[0].lower() == “skipnet”:
              for i in range(1, len(words)):
                  skipNets.append(words[i])
          elif words[0].lower() == “instmaster”:
              if words[1].lower() == “t”:
                  instmaster = 1
fs.close()

# Build map tables for each subckt in CDL net list
#    Skip the subckt in the SKIPCELL table
#    PORT is the list of ports to a subckt
#    INST is the list of instances to a subckt
#
PORT = dict()
INST = dict()
if os.path.exists(netlist):
   fn = open(netlist, “r”)
else:
   print(“%s does not exist !” %netlist)
   exit()
print(“CDL net list: %s” %netlist)
subckt = 0
for line in fn:
    words = line.split()
    if words[0] = “.SUBCKT”:
        if “.SUBCKT” in line:
            subCkt = 1
      master = words[1]
      portList = list()
      instList = list()
      if master not in SKIPCELL:
         PORT[master] = portList
         INST[master] = instList
         for i in range(2, len(words)):
            portList.append(words[i])
elif words[0] == “.ENDS”:
  subCkt = 0
elif subCkt == 1:
  # Add all instance lines in the instList for INST mapping table
      instList.append(line)

if topCell == “”:
   # The last master becomes the top cell
   if master not in SKIPCELL:
      topCell = master
if topCell == “”:
print(“ERROR: top cell name is not defined \n”)
exit
```

```
fn.close()


#
# Flatten the CDL netlist in the recursive way
#
flatCDL = topCell + “.cdl.flat”
f1w = open(flatCDL, “w”)
PORTMAP = dict()
flattenInstCDL(topCell, topCell, PORTMAP, SKIPCELL, skipInsts, instmaster, INST, PORT)
print(“Flattened CDL netlist: %s” %flatCDL)
f1w.close()


#
# Flatten the CDL netlist in the recursive way
#
flatCDL = topCell + “.cdl.flat”
f1w = open(flatCDL, “w”)
PORTMAP = dict()
flattenInstCDL(topCell, topCell, PORTMAP, SKIPCELL, skipInsts, instmaster, INST, PORT)
print(“Flattened CDL netlist: %s” %flatCDL)
f1w.close()


#
# Report floating gate errors
#
errFile = topCell + “.cdl.err”
f2w = open(errFile, “w”)
SOURCENET = dict()
GATENET = dict()
DRAINNET = dict()
BULKNET = dict()
flatNets = list()
FLATNET = dict()
NETINST = dict()
INSTDEVICE = dict()
totalDevices = 0
f1r = open(flatCDL, “r”)
for line in f1r:
  words = line.split()
inst = words[0]
  sourceNet = words[1]
  gateNet = words[2]
  drainNet = words[3]
  bulkNet = words[4]
  if sourceNet not in SOURCENET:
  SOURCENET[sourceNet] =  [inst]
  else:
  SOURCENET[sourceNet].append(inst)
  if gateNet not in GATENET:
  GATENET[gateNet] =  [inst]
  else:
  GATENET[gateNet].append(inst)
if drainNet not in DRAINNET:
  DRAINNET[gateNet] =  [inst]
else:
  DRAINNET[gateNet].append(inst)
device = words[5]
INSTDEVICE[inst] = device
totalDevices = totalDevices + 1
f1r.close()


# Go through flatNets to report error nets which are only connected to GATENET
errorNets = 0
```

```
for net in flatNets:
if (net in GATENET) and ((net not in
SOURCENET) and (net not in DRAINNET)):
  if checkSN(net, skipNets) == 0:
    errorNets = errorNets + 1
    f2w.write(“ERROR: net %s is connected to gate
only (No connections to source/drain) -  \n” %net)
    for inst in GATENET[net]:
      device = INSTDEVICE[inst]
      f2w.write(“\tInstance: %s %s \n” % (inst,
device))
f2w.close()
print(“Floating gate checking results: %s” %errFile)
totalNets = len(flatNets)
totalHVNets = len(HVNET.keys())
totalLVNets = len(LVNET.keys())
print(“Statistics: total nets = %d  Total devices =
%d” %(totalNets, totalDevices))
print(“Number of error nets: %d” %errorNets)
print(“Checking is finished in %s seconds !”
%(time.time() - start))
```