

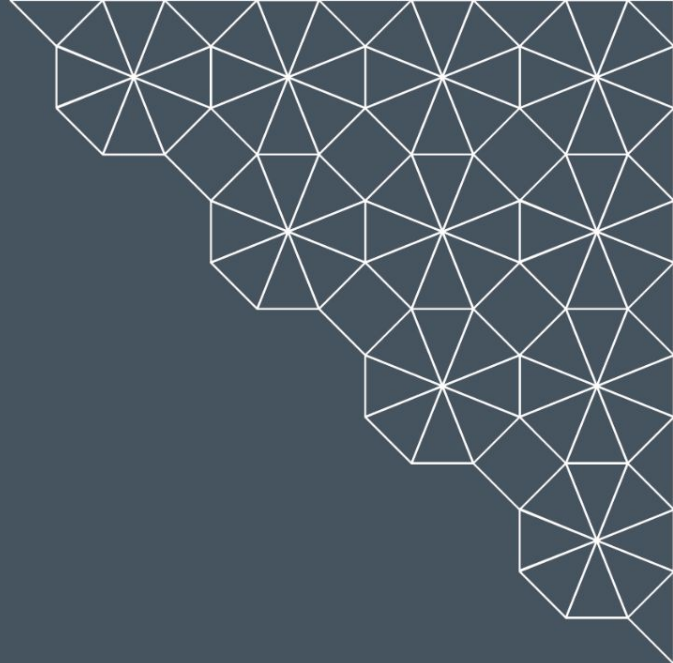
VLSIR - A Modular Framework for Programming Analog & Custom Circuits & Layouts

Berkeley
UNIVERSITY OF CALIFORNIA

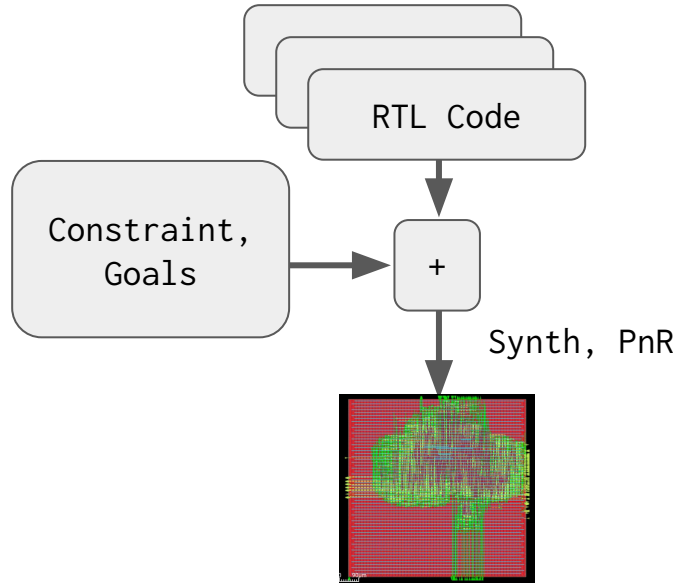
Dan Fritchman
ISPD 2023

About Me

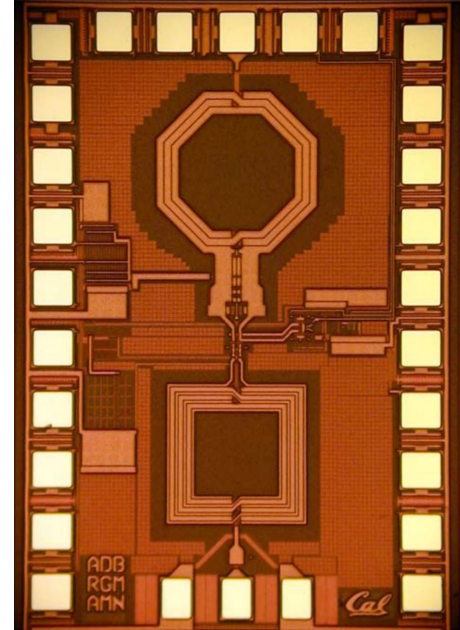
- Duke 2006
- Apple 2009-19
 - Embedded systems
 - SoC analog & mixed-signal IP
 - Software therefore
- Berkeley 2020-present
 - Software for AMS circuits



The Two Ways 99.999% of IC Layout is Made



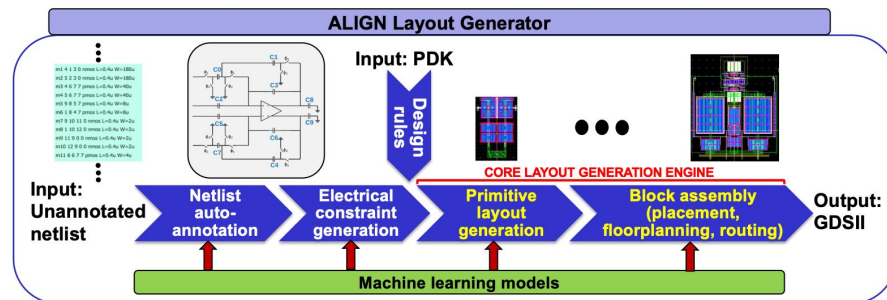
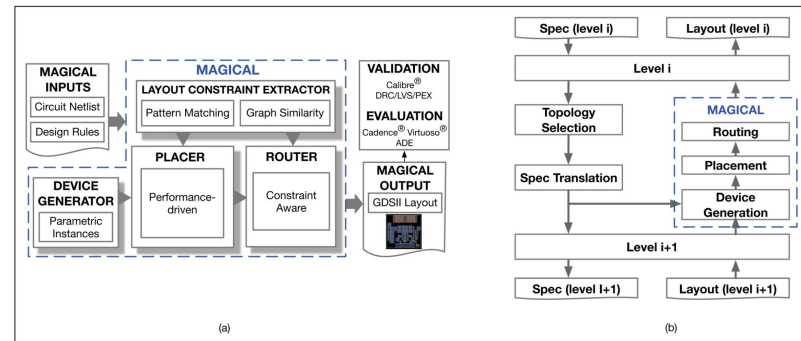
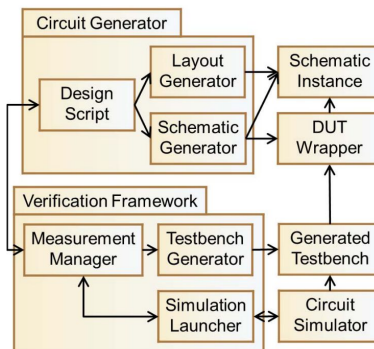
by compilation & constraints
AKA digital PnR



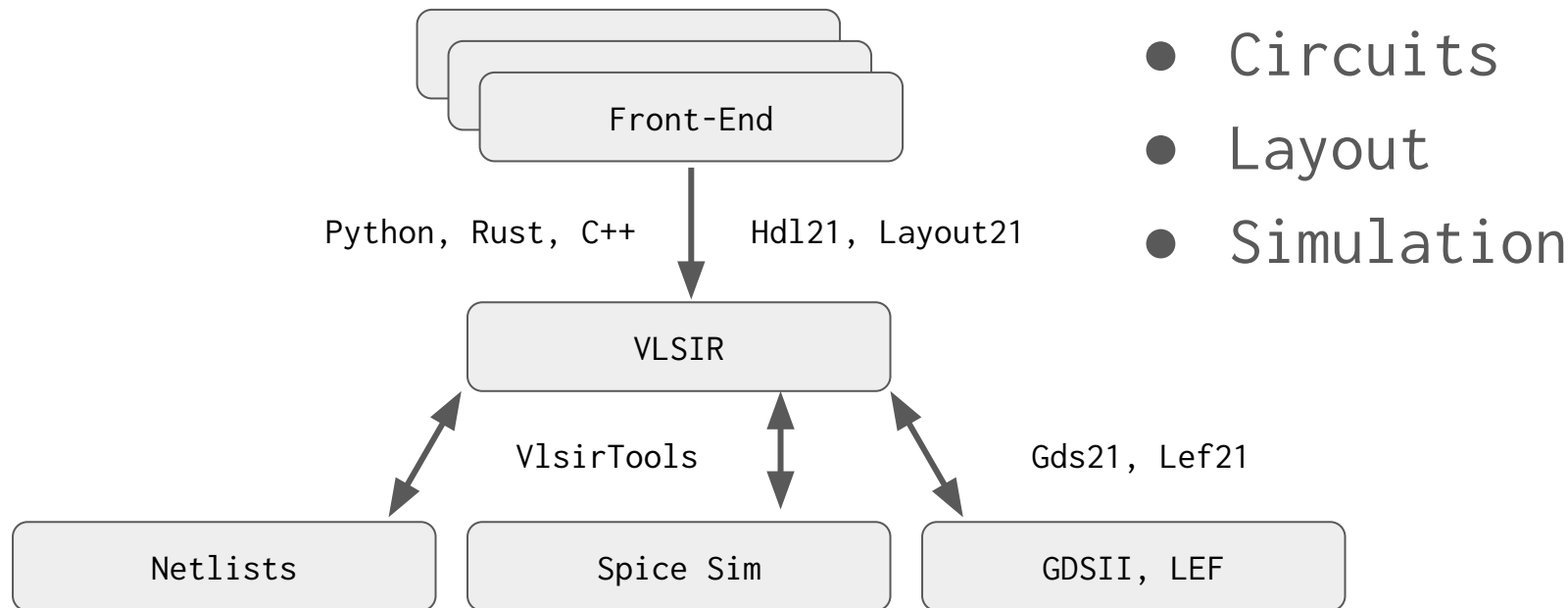
Full Custom
AKA by GUI Clicks

Recent Prior Art

- ALIGN == analog layout intelligently generated from netlists
- MAGICAL == machine generated analog IC layout”
- BAG == Berkeley Analog Generator



VLSIR == Protobuf Design Database



Dinner Party Tests

- Explain this to someone smart
- But not in the field

Dinner Party Test #1:

```
print("Hello World!")
```

```
print("Hello Again")
```

```
if something:
```

```
    print("Something is true")
```

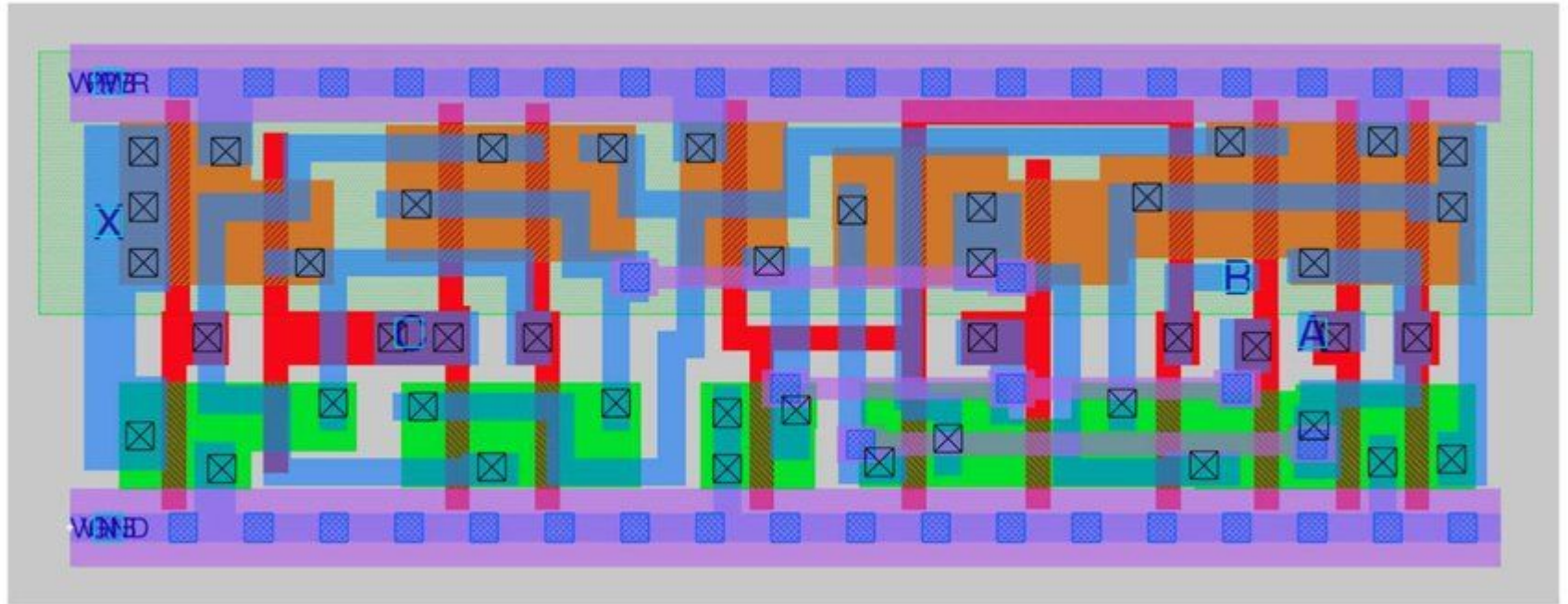
```
a_number = 5
```

```
while a_number > 3:
```

```
    print(a_number)
```

```
    a_number = get_a_random_number()
```

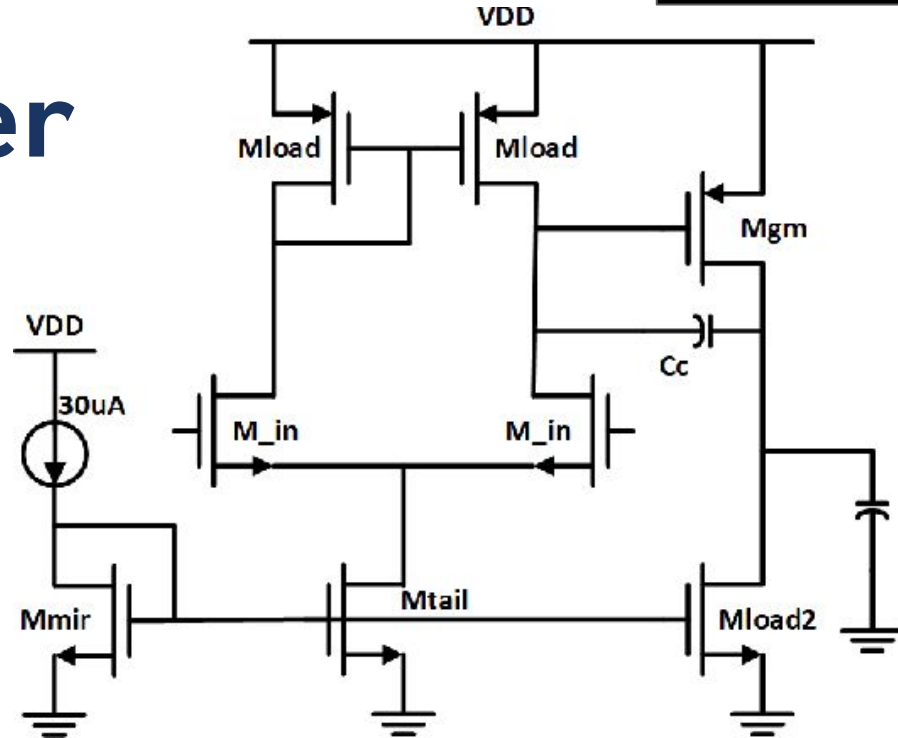
Dinner Party Test #2:



Dinner Party

Test #3: (MUCH) Harder

Zoom
Presenter



Hd121 ==

Analog HDL + Python

Zoom
Presenter

- For circuit designers
- Expose the concepts they know, in the most approachable language available (Python)
- Max productivity, min fancy-programming skill

```
import hd121 as h
```

```
@h.module
```

```
class MyModule:
```

```
    i = h.Input()
```

```
    o = h.Output(width=8)
```

```
    s = h.Signal()
```

```
    a = AnotherModule(a=i, b=s, c=o)
```



```
@h.module  
  
class Inner:  
    inp = h.Input()  
    out = h.Output()
```

```
@h.module  
  
class BackToBack:  
    t1 = Inner()  
    t2 = Inner(  
        inp=t1.out,  
        out=t1.inp  
    )
```

```
b2 = h.Module(name="BackToBack")  
b2.t1 = Inner()  
b2.t2 = Inner()  
b2.t2.inp = b2.t1.out  
b2.t2.out = b2.t1.inp
```



Hd121 Generator == f(Params) -> Module

Zoom
Presenter

```
@h.generator
def Gen(params: MyParams) -> h.Module:
    m = h.Module()                # Initialize our result Module
    m.i = h.Input(width=params.w) # Add a parametric input
    if params.has_reset:          # Based on another parameter,
        m.reset = h.Input()       # maybe add another input
    return m                      # And return our Module
```



The Stuff Only an Analog-er Could Love

Zoom
Presenter

- PDKs
- Generic primitives
- Compiling generics into PDKs
- PVT Corners
- Spice sims
- Diff pairs
- Matched arrays
- etc

```
@h.module

class Tb:

    # All it takes to be a testbench
    VSS = h.Port()



# @h.sim # Using `mypdk.install`


class SimMyPdk:

    tb = MyTestBench
    models = Include(
        path=mypdk.install.models
    )

SimMyPdk.run() # Run it!
```

```
@h.module

class Rlc:

    p, n = h.Ports(2)

    res = h.Res(r=1*K) (p=p, n=n)
    cap = h.Cap(c=1*μ) (p=p, n=n)
    ind = h.Ind(l=1*n) (p=p, n=n)

    # Write a spice-format netlist to stdout
    h.netlist(Rlc, sys.stdout, fmt="spice")
```

```
@h.module

class Inv:

    i, o, VDD, VSS = h.Ports(4) # Create some IO
    # And now create some generic transistors!

    ps = Pmos(w=1*μ, l=1*μ, vth=MosVth.STD) (d=o, g=i, s=VDD, b=VDD)
    ns = Nmos(w=1*μ, l=1*μ, vth=MosVth.STD) (d=o, g=i, s=VSS, b=VSS)

import sky130
sky130.compile(Inv)
```

Bundle == struct conn

Diff == a special Bundle

Pair == two identical Instances

```
@h.bundle
```

```
class Diff:
```

```
    p, n = h.Signals(2)
```

```
@h.bundle
```

```
class QuadratureClock:
```

```
    i = Diff() # In-phase component
```

```
    q = Diff() # Quadrature component
```

```
@h.module
```

```
class DiffAmp:
```

```
    i = h.Diff()
```

```
    o = h.Diff()
```

```
    bias, VDD, VSS = h.Inputs(3)
```

```
    ni = Nbias(g=bias, s=VSS)
```

```
    ns = h.Pair(Ninp)(s=ni.d, g=i, d=o)
```

```
    rl = h.Pair(Rl)(p=o, n=VDD)
```

My General Outlook:

Zoom
Presenter

MOST

Schematics are BAD

My General Outlook:

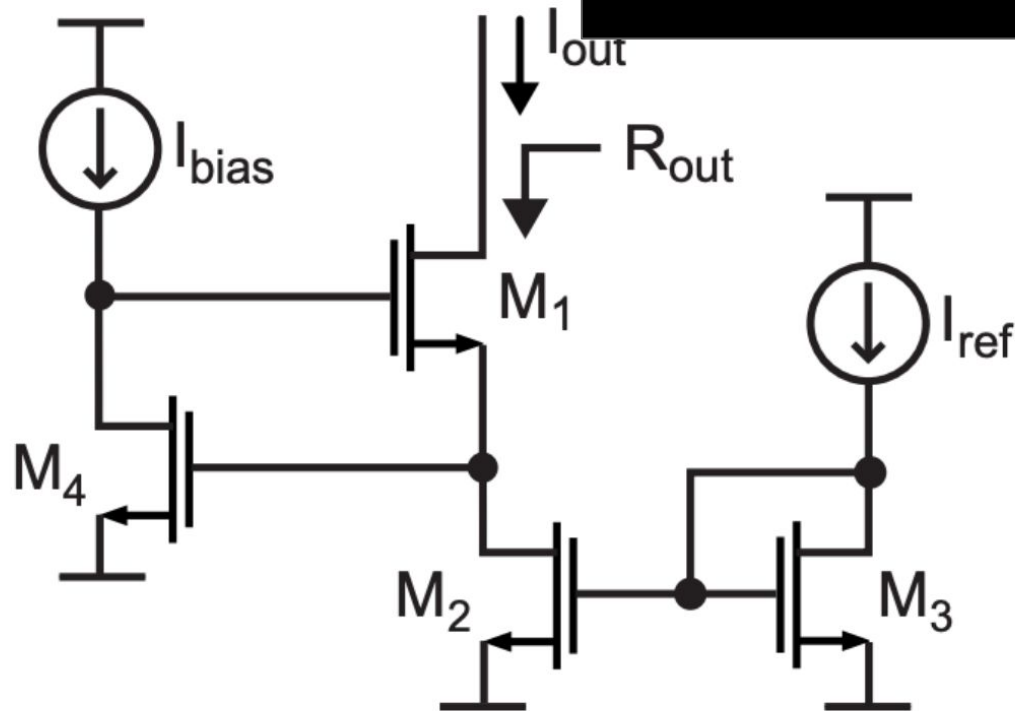
Zoom
Presenter

MOST

**Schematics should
be code**

But There Are
Some
Awesome
Schematics

Zoom
Presenter



Hd121 Schematics

Zoom
Presenter

- Schematics are **SVG images**
 - Single file, no dependencies, no “database”
 - Every modern browser & OS can render
- Schematics are “**graphical Python modules**”
 - Import as hd121 Generators



Vlsir / Hdl21Schematics Public

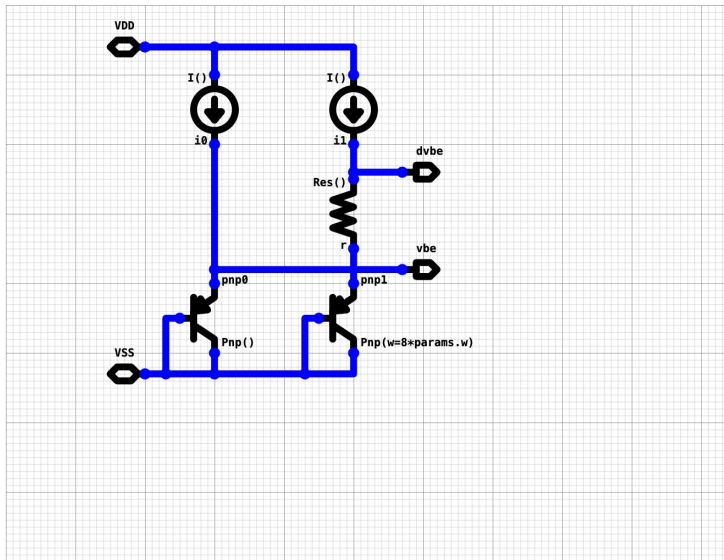
<> Code Issues 19 Pull requests Discussions Actions Projects Wiki

main Hdl21Schematics / docs / readme.md

dan-fritchman Add docs/readme ✓

1 contributor

21 lines (11 sloc) 152 Bytes



df > Hdl21Schematics



Add docs/readme

Dan Fritchman authored 22 hours ago

main hdl21schematics / docs / readme.md

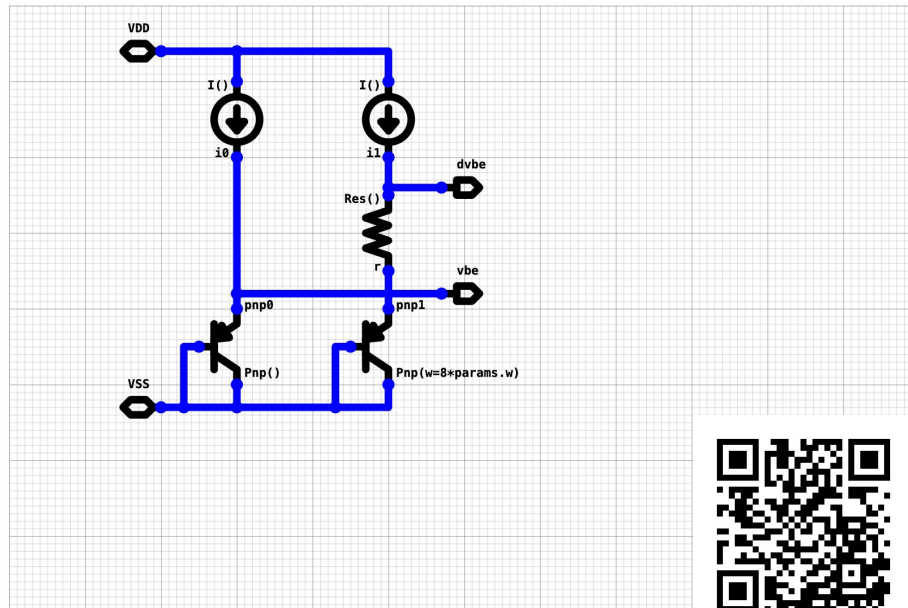
Find file

readme.md 152 bytes

</>

Open in Web IDE

Open in Web IDE



Zoom Presenter

EXPLORER

OPEN EDITORS

HDL21SCHEMATICDEMOS

> .pytest_cache

> hdl21schematicdemos

> __pycache__

__init__.py

inverter.sch.svg

ring.py

> tests

.gitignore

poetry.lock

pyproject.toml

readme.md

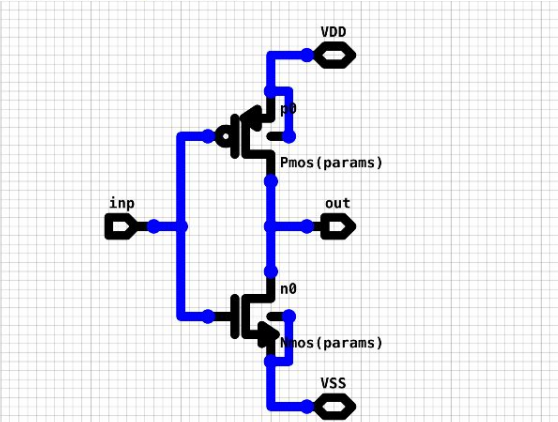
ring.py

hdl21schematicdemos > ring.py

```
1  """
2  # Ring Oscillator
3
4  Using the inverter delay element defined in 'inverter.sch.svg'.
5  """
6
7  import hdl21 as h
8  from hdl21.primitives import MosParams
9  import hdl21schematicimporter as _
10
11  Import the inverter schematic
12  from .inverter import inverter
13
14  # Just in case you don't believe this:
15  assert isinstance(inverter, h.Generator)
16
17
18  @h.paramclass
19  class RingOscParams:
20      """Ring Oscillator Parameters"""
21
22      mos = h.Param(dtype=MosParams, desc="Xtor Params", default=MosParams())
23
24
25  @h.generator
26  def RingOsc(params: RingOscParams) -> h.Module:
27      """A three-stage ring oscillator"""
28
29      @h.module
30      class RingOsc:
31          VDD, VSS = h.Inputs(2)
32          a, b, c = h.Outputs(3)
33
34          # Instantiate our 3 schematic inverters
35          ia = inverter(params.mos)(inp=a, out=b, VDD=VDD, VSS=VSS)
36          ib = inverter(params.mos)(inp=b, out=c, VDD=VDD, VSS=VSS)
37          ic = inverter(params.mos)(inp=c, out=a, VDD=VDD, VSS=VSS)
38
39      return RingOsc
```

inverter.sch.svg

hdl21schematicdemos > inverter.sch.svg



>

Code Prelude

```
import hdl21 as h
from hdl21.primitives
import Nmos, Pmos,
MosParams
Params = MosParams
```

i

Add Instance

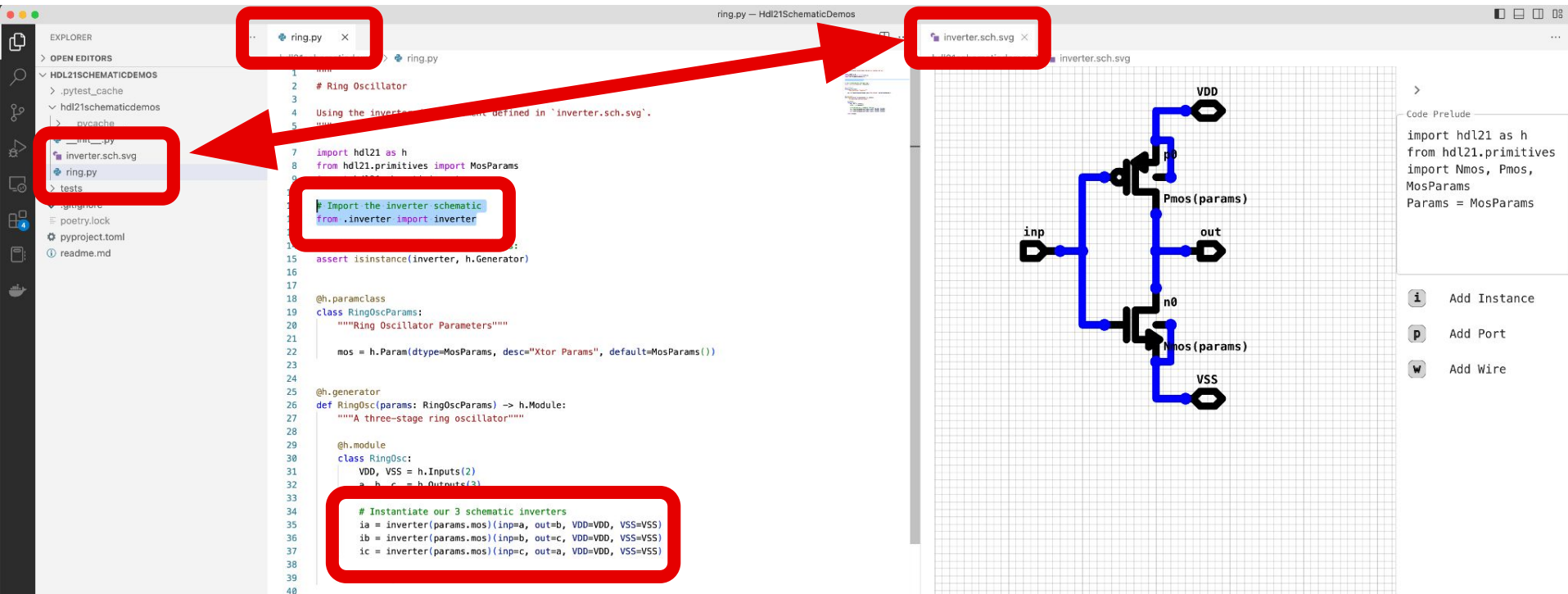
p

Add Port

w

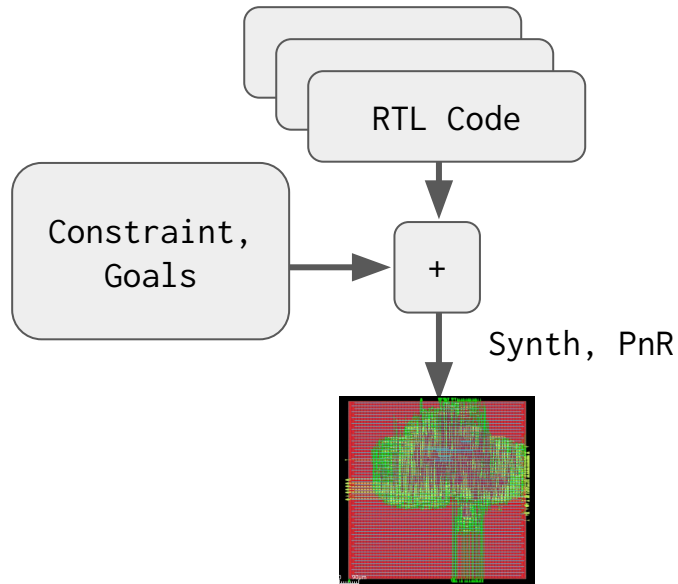
Add Wire

Zoom Presenter

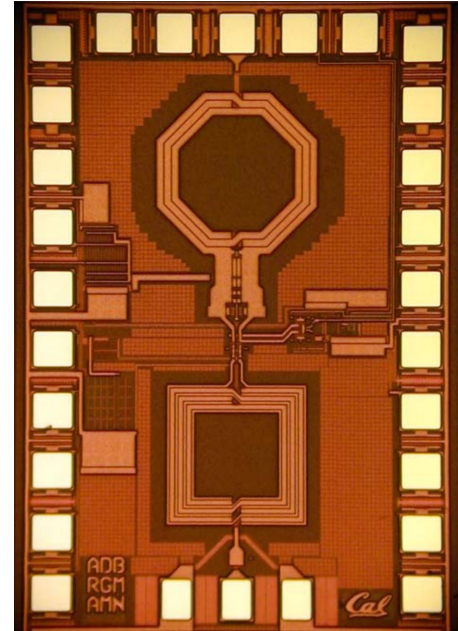


The Two Ways 99.999% of IC Layout is Made

Zoom
Presenter



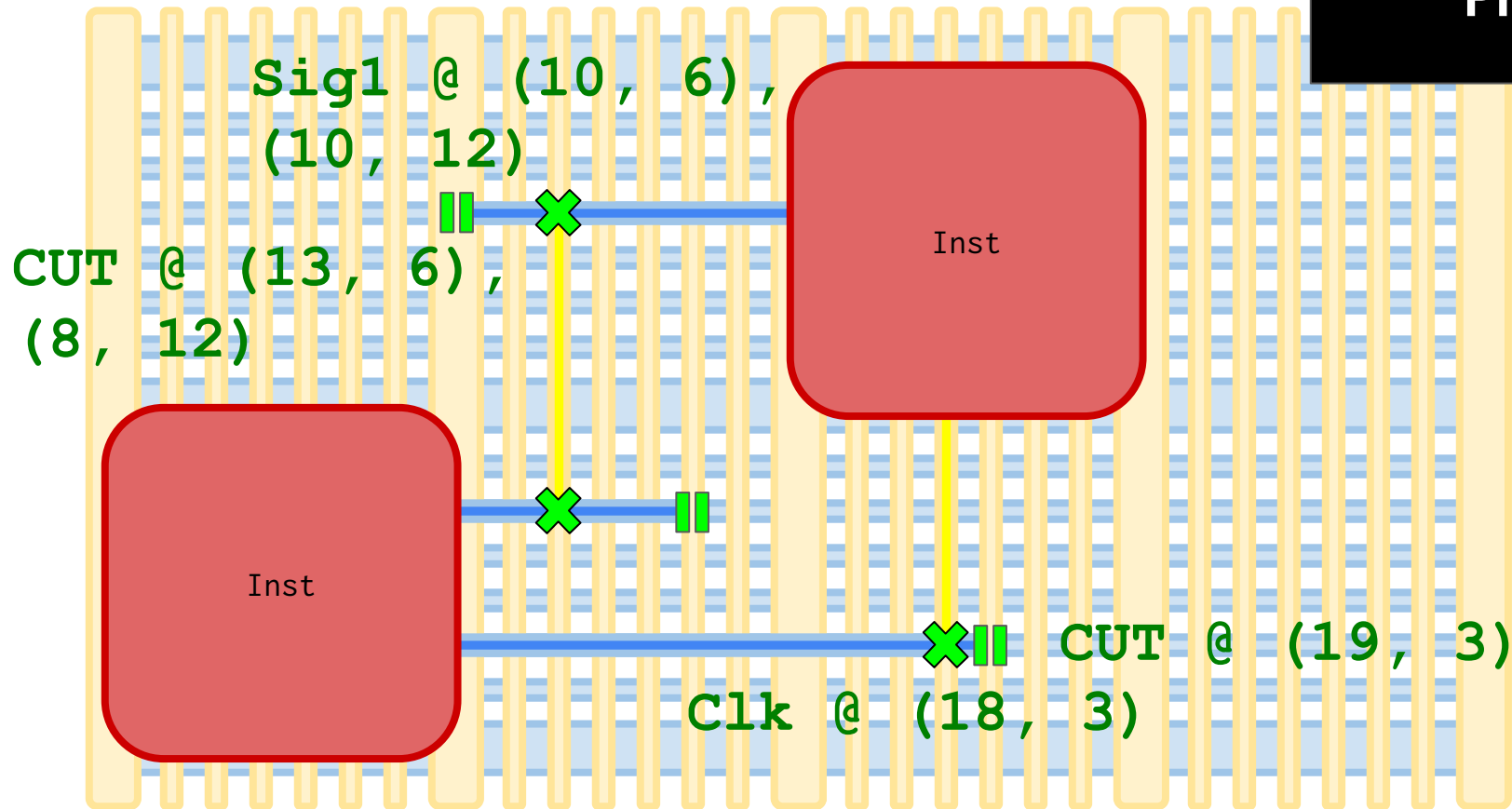
by compilation & constraints
AKA digital PnR



Full Custom
AKA by GUI Clicks

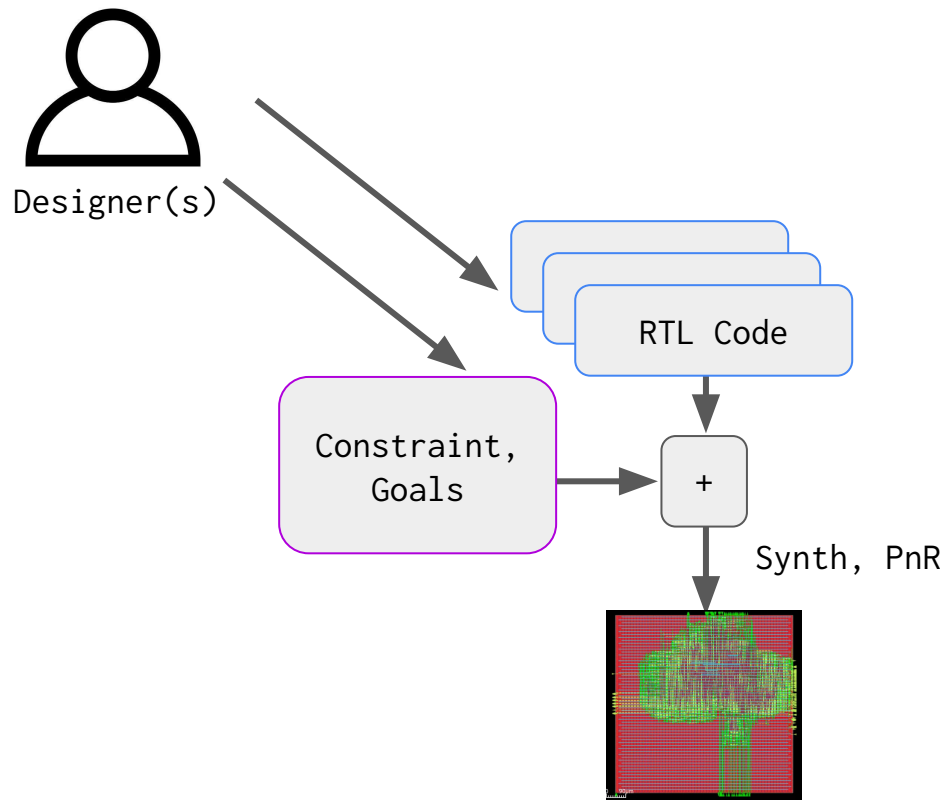
Tetris Routing

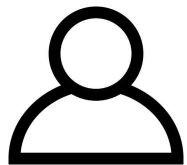
Zoom
Presenter



Typical PnR Compiler

Zoom
Presenter





Designer



Hd121 Modules

Everything that goes into a netlist

What's differential

Which signals are
important

What needs to be
matched

etc



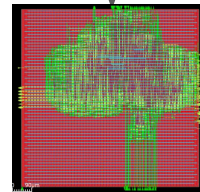
Netlist



PnR
Program



Constraint,
Goals



Zoom
Presenter

