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

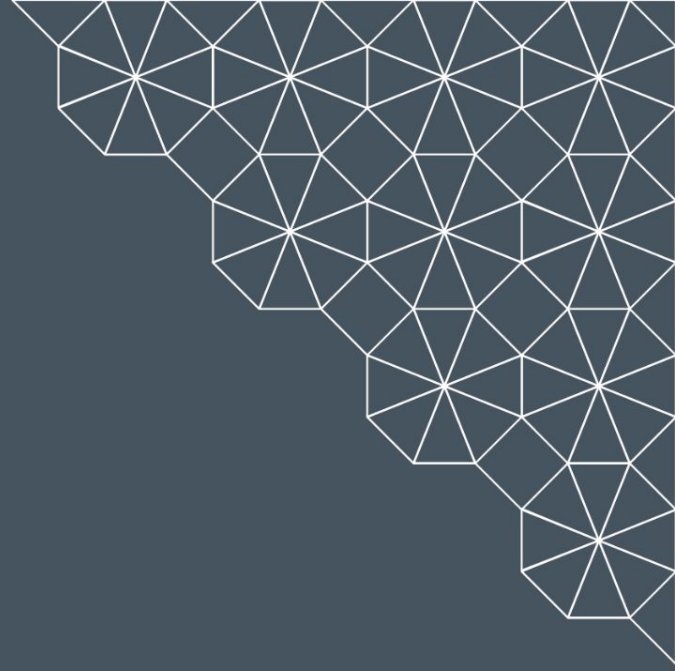
Dan Fritchman

ISPD 2023

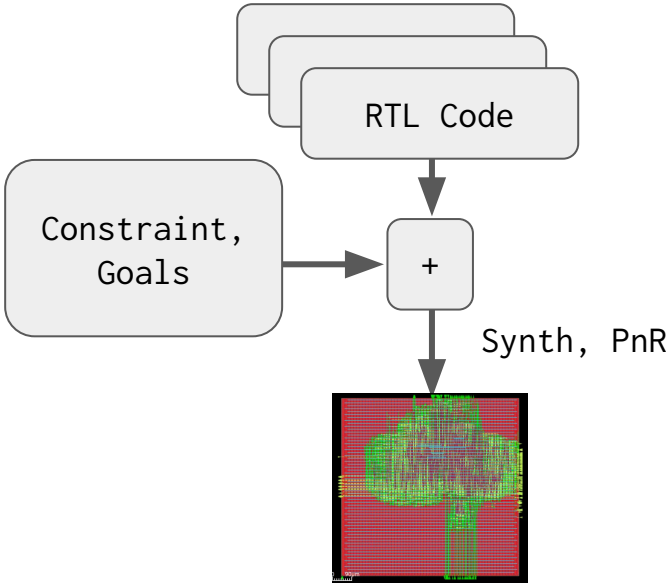
Berkeley
UNIVERSITY OF CALIFORNIA

About Me

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

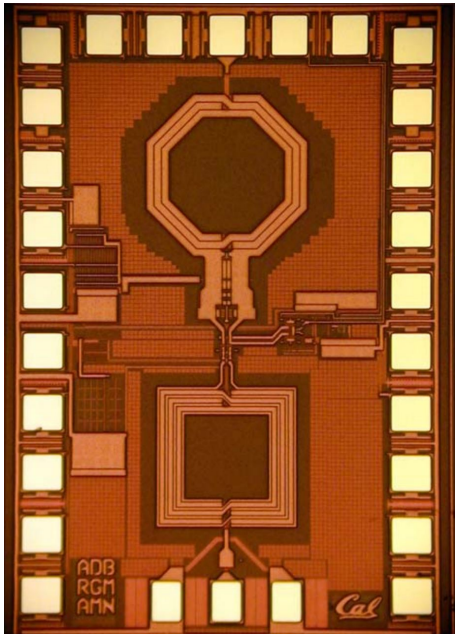


The Two Ways 99.999% of IC Layout is Made



by compilation & constraints

AKA digital PnR

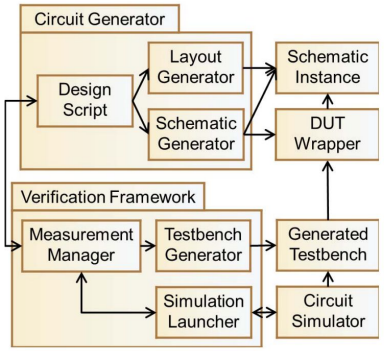


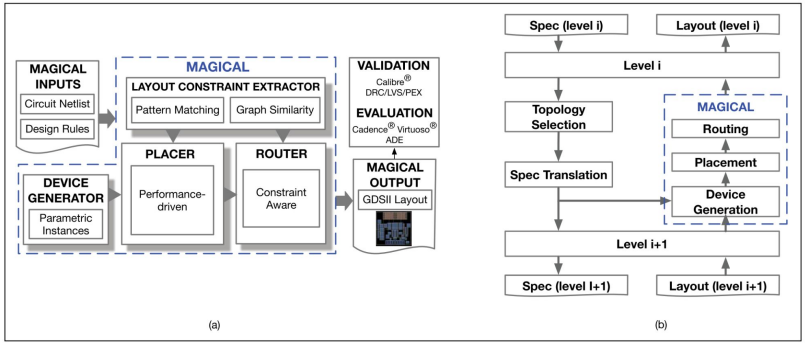
Full Custom

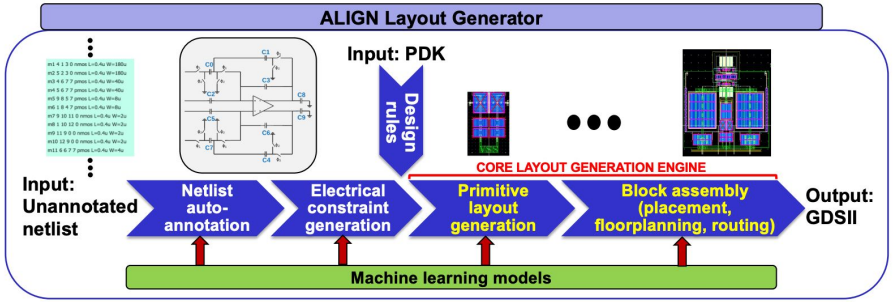
AKA by GUI Clicks

Recent Prior Art

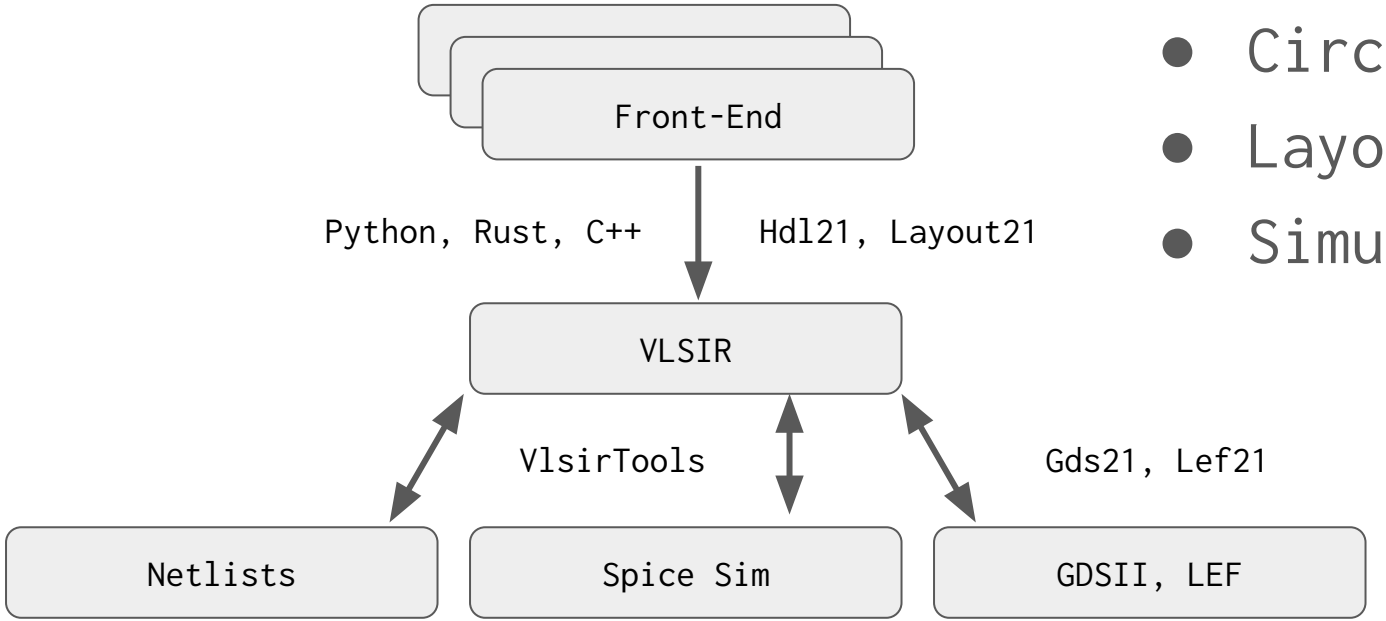
- ALIGN == analog layout intelligently generated from netlists
- MAGICAL == machine generated analog IC layout”
- BAG == Berkeley Analog Generator







VLSIR == Protobuf
Design Database



- Circuits
- Layout
- Simulation

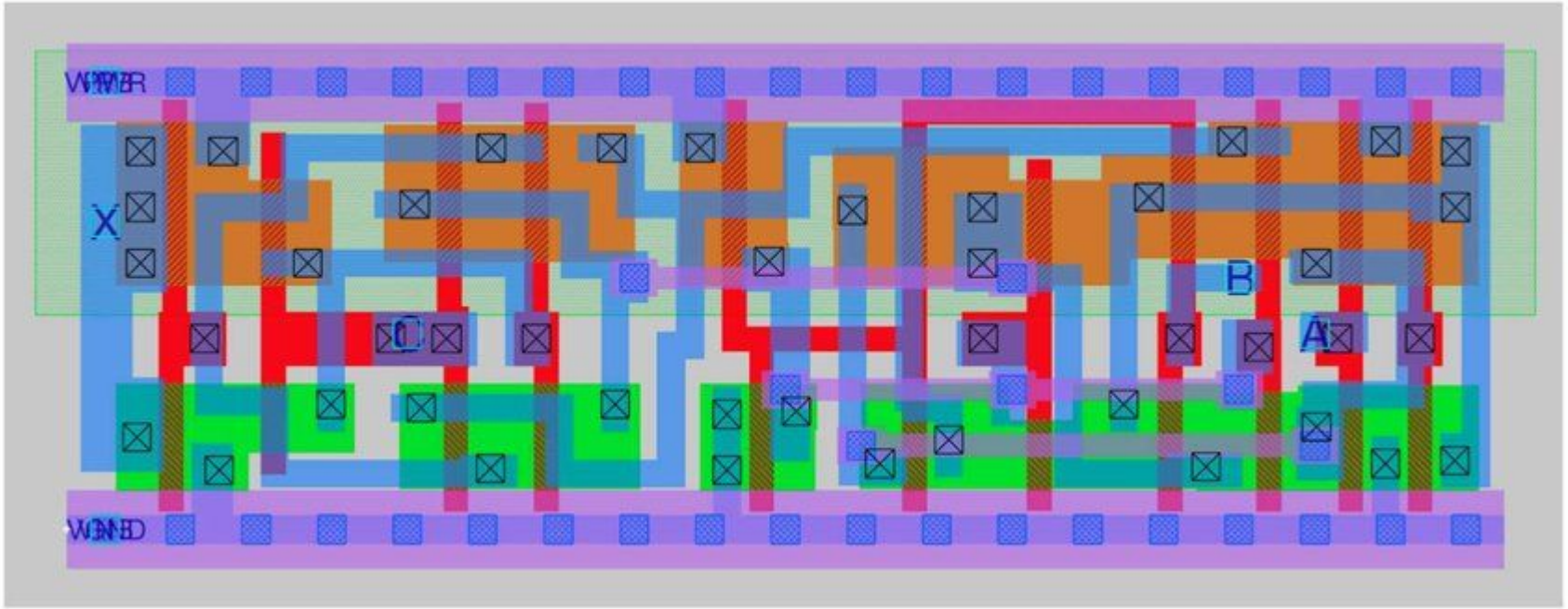
Dinner Party Tests

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

Dinner Party Test #1:

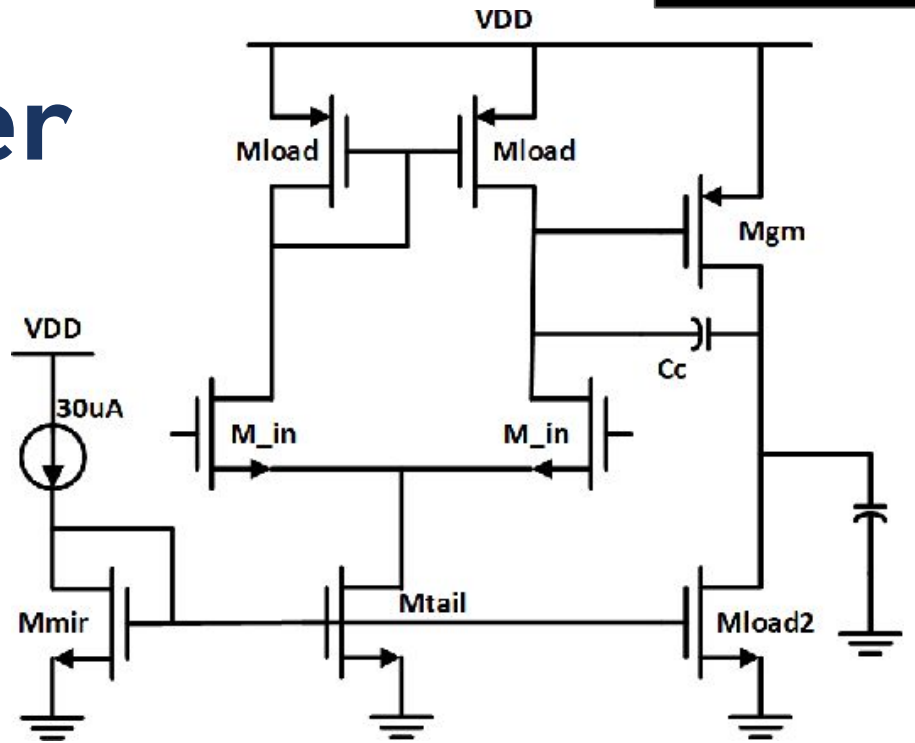
```
print("Hello World!")
print("Hello Again")

if something:
    print("Something is true")

a_number = 5
while a_number > 3:
    print(a_number)
    a_number = get_a_random_number()
```

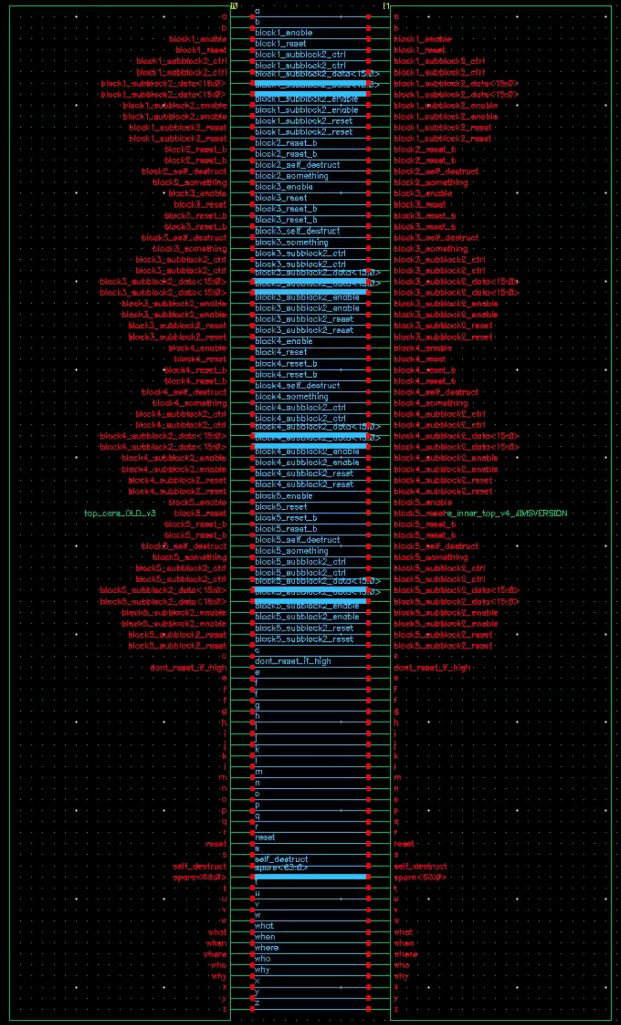
Dinner Party Test #2:



Dinner Party Test #3: (MUCH) Harder

Zoom Presenter





Most Real (Bad) Schematics

Zoom Presenter

Hdl21 ==
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 hdl21 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)
```

Zoom
Presenter

```
@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
```

Hdl21 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

Zoom Presenter

```
@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:

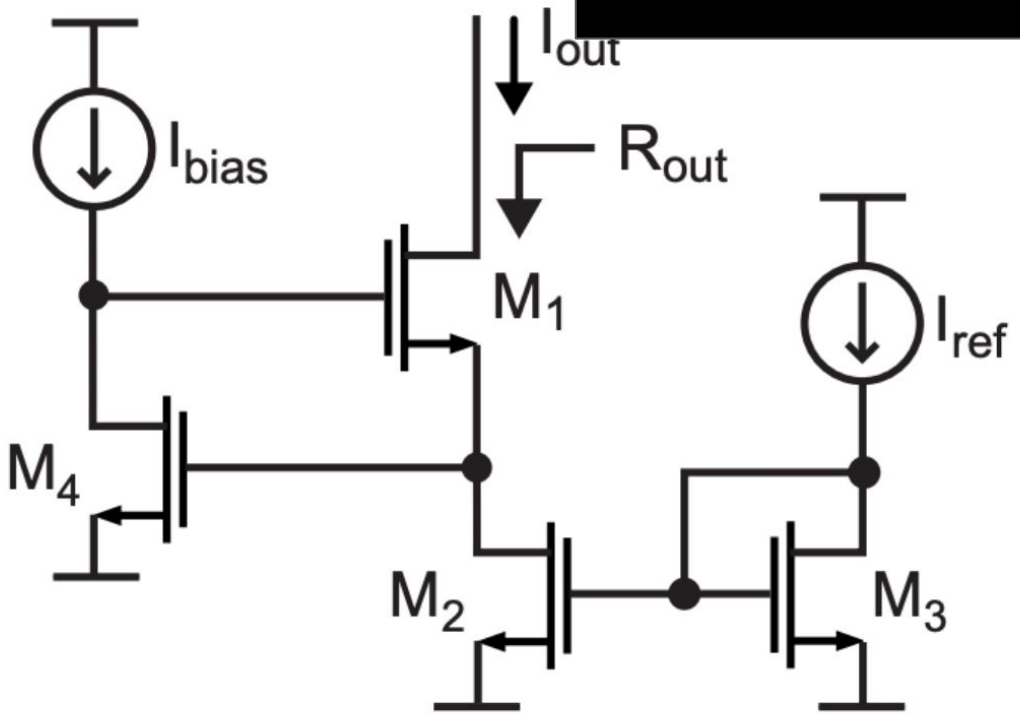
Zoom Presenter

MOST

Schematics should be code

But There Are Some *Awesome* Schematics

Zoom Presenter



Hdl21 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 hdl21 Generators

Search or jump to... / Pull requests Issues Codespaces Marketplace Explore

Vlsir / Hdl21Schematics Public

Edit Pins

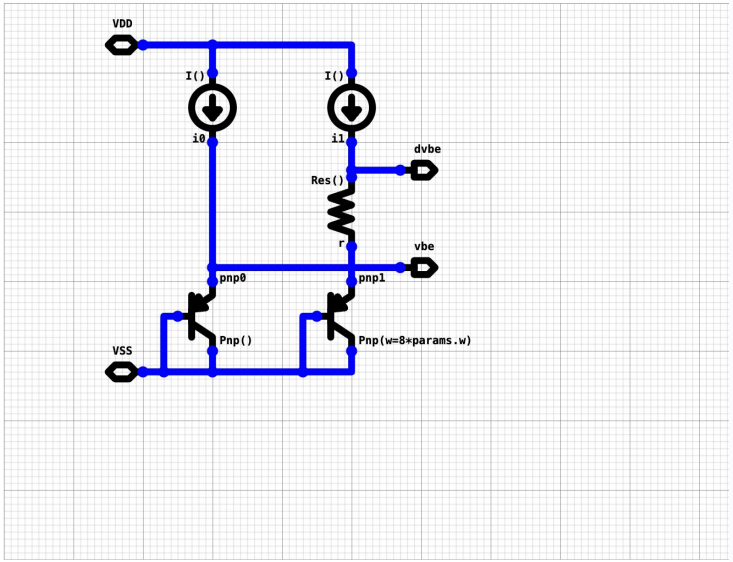
Code Issues 19 Pull requests Discussions Actions Projects Wiki

main Hdl21Schematics / docs / readme.md

dan-fritchman Add docs/readme ✓

1 contributor

21 lines (11 sloc) | 152 Bytes



Search GitLab /

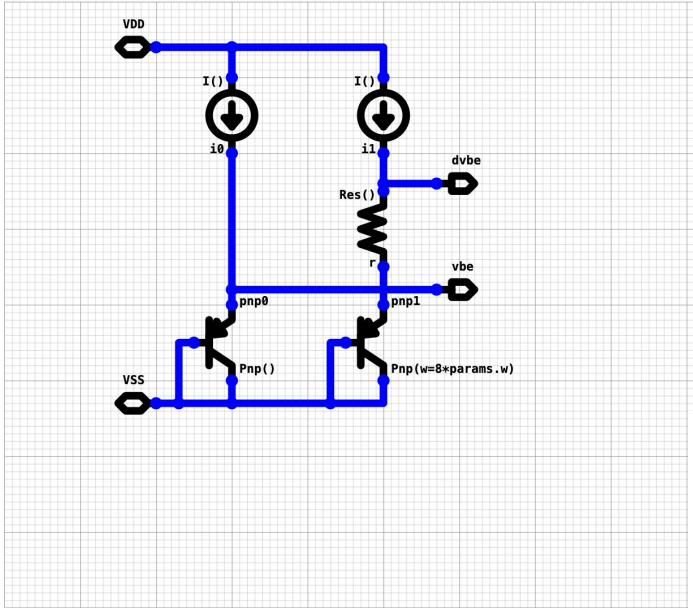
H

df › Hdl21Schematics

Add docs/readme
Dan Fritchman authored 22 hours ago

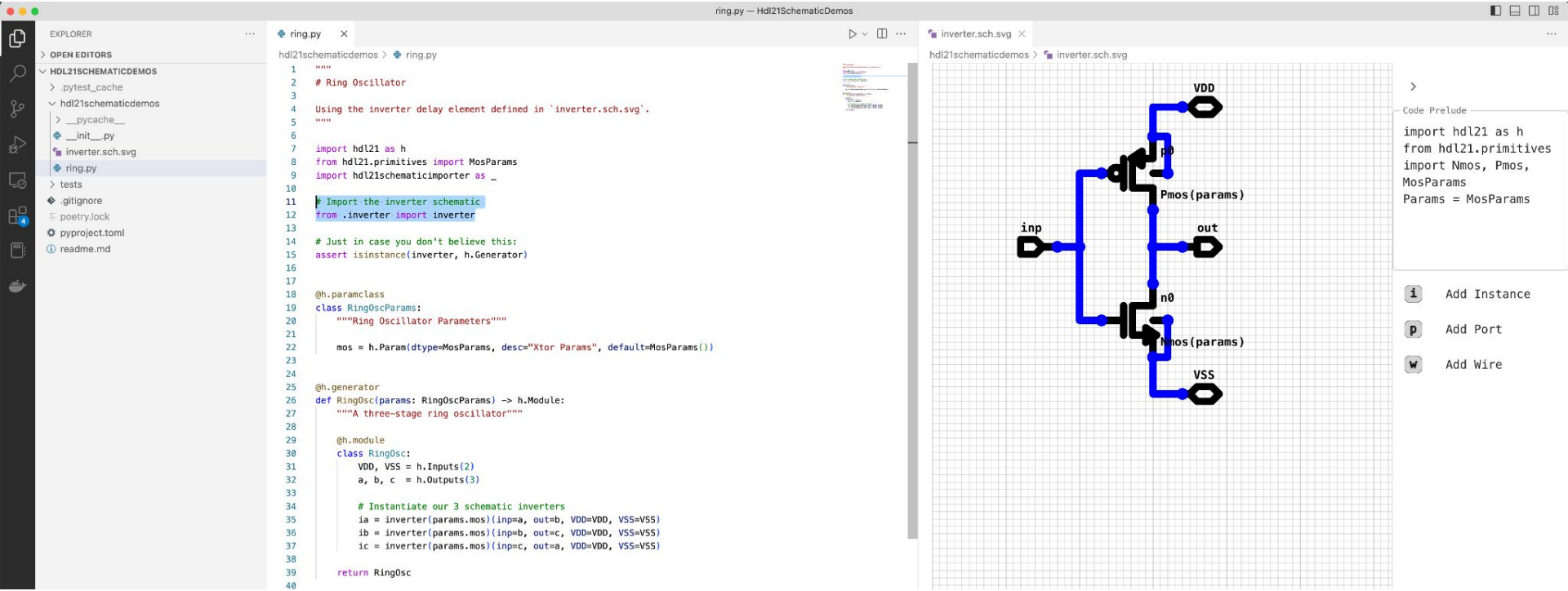
main hdl21schematics / docs / readme.md Find file

readme.md 152 bytes Open in Web IDE

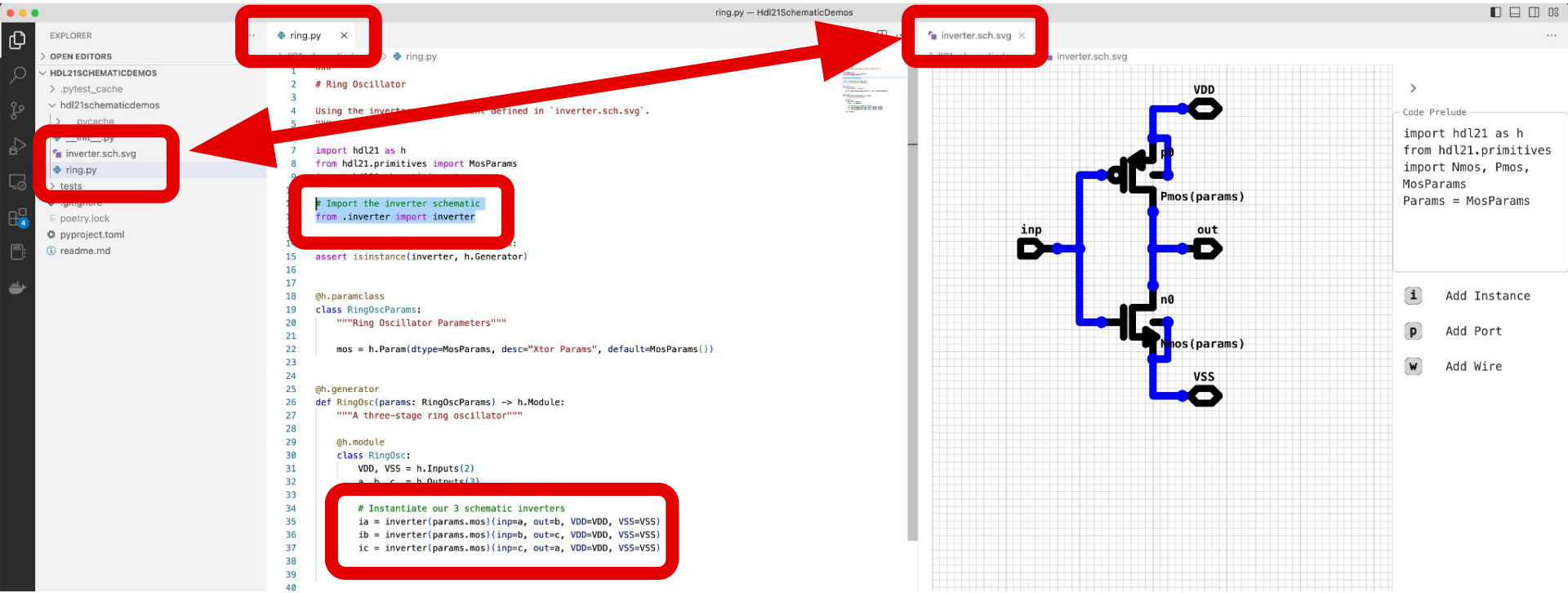


Zoom Presenter

Zoom Presenter

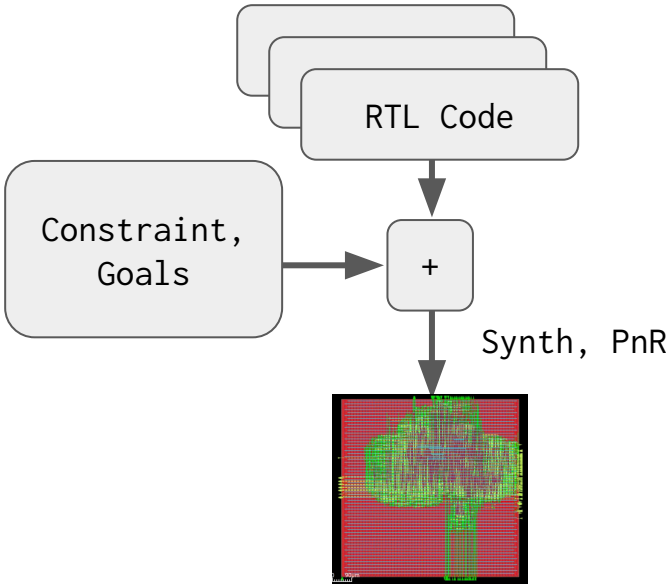


Zoom
Presenter



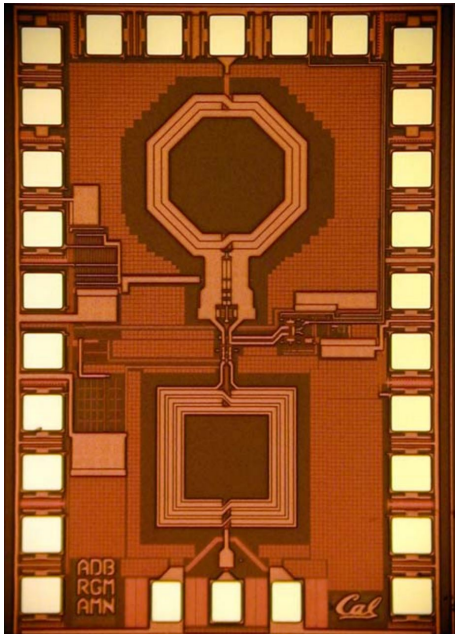
The Two Ways 99.999% of IC Layout is Made

Zoom Presenter



by compilation & constraints

AKA digital PnR

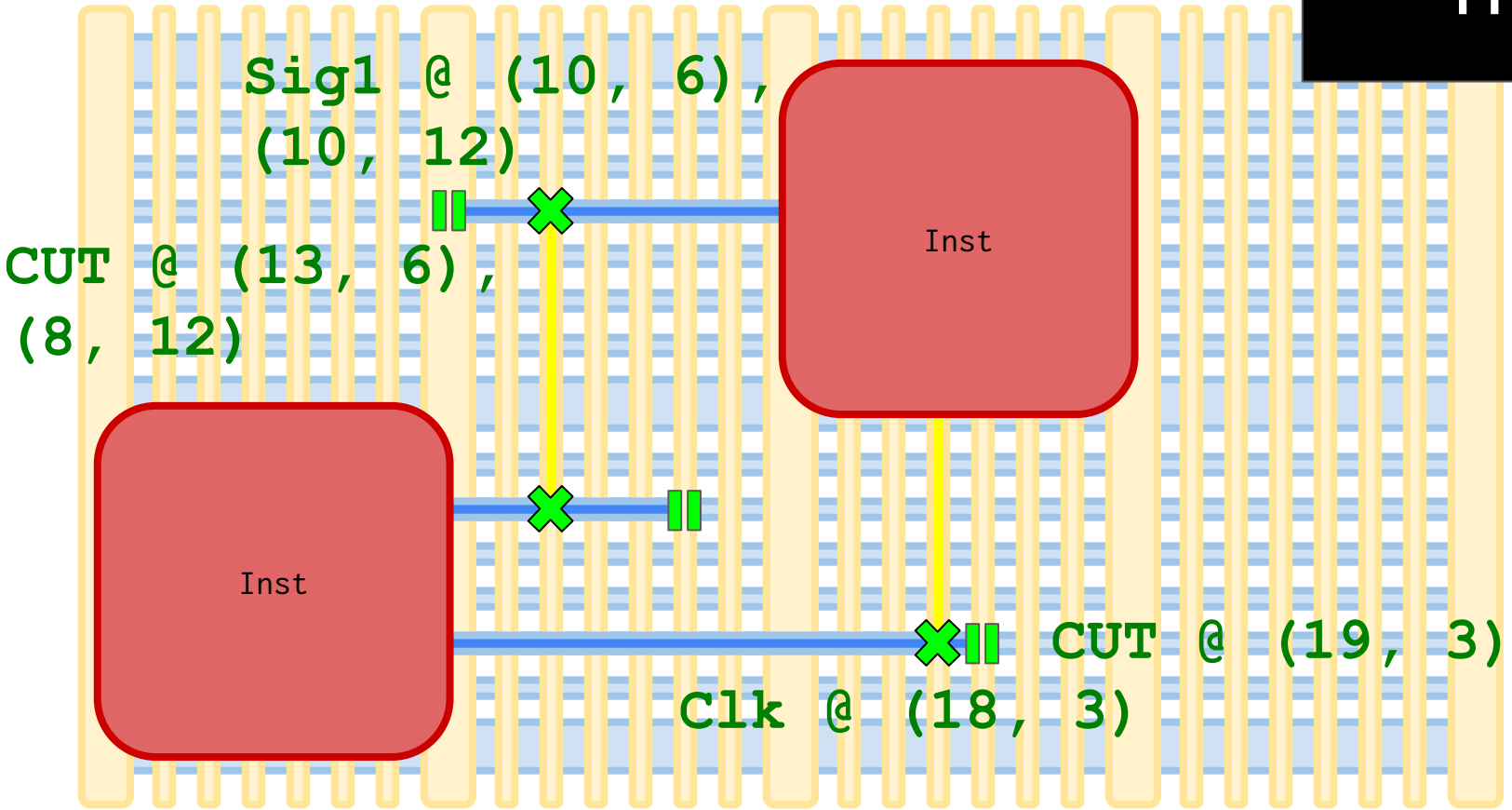


Full Custom

AKA by GUI Clicks

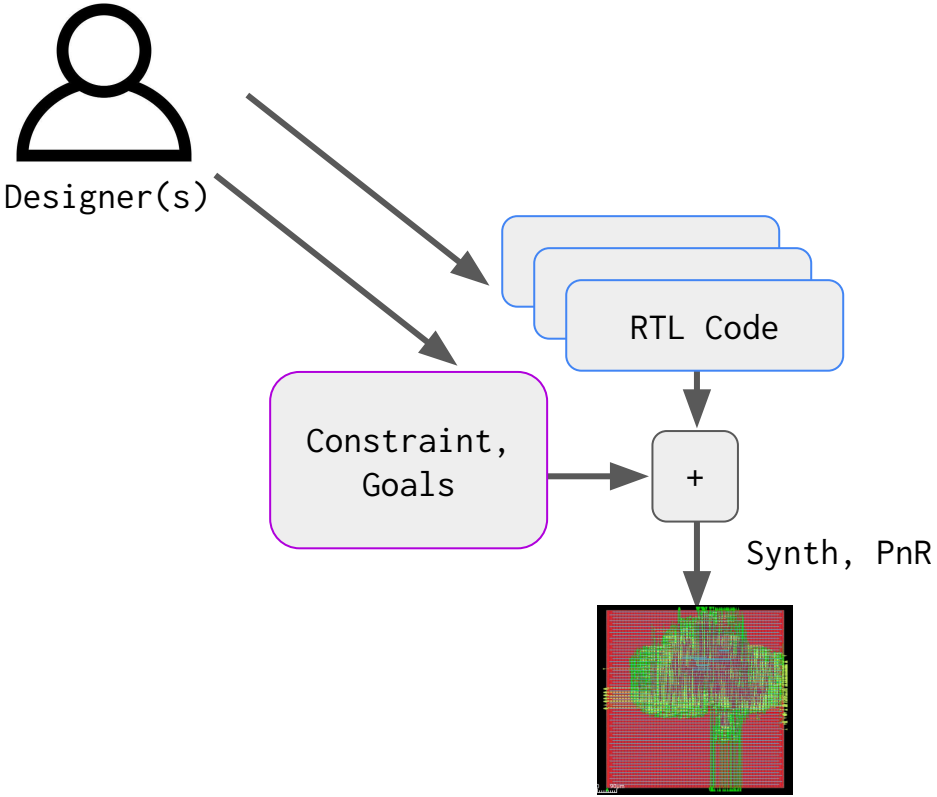
Tetris Routing

Zoom
Presenter



Typical PnR Compiler

Zoom Presenter



Zoom
Presenter

Designer

Hdl21 Modules

Everything that goes into a netlist

What's differential

Which signals are important

What needs to be matched

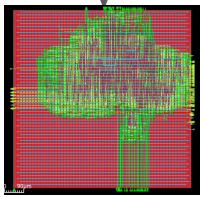
etc

Netlist

Constraint, Goals

+

PnR Program



Berkeley
UNIVERSITY OF CALIFORNIA