zer|o|

Zero ASIC

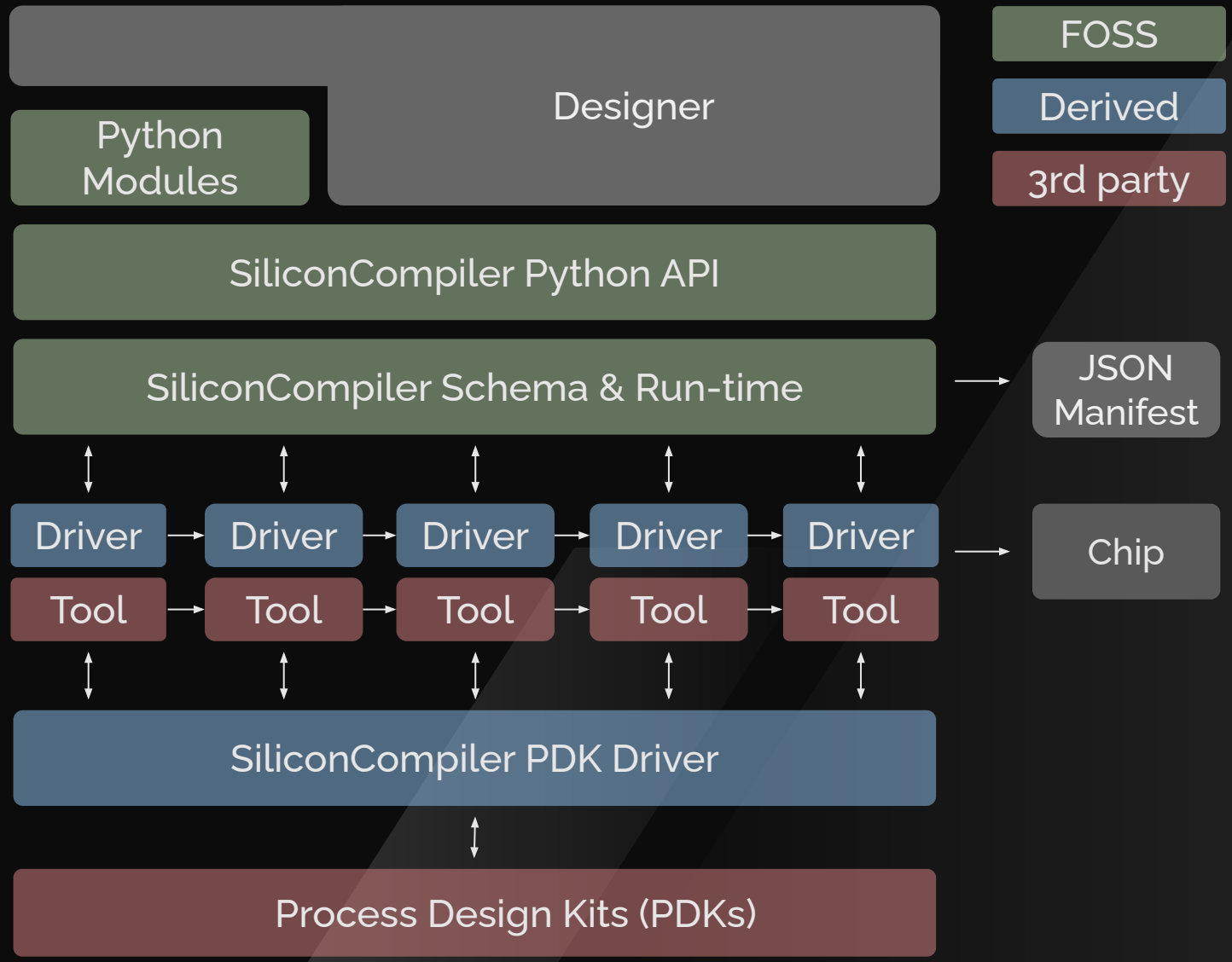
Removing the barrier to custom silicon

LLVM/CIRCT Meeting, Jan 19 2022
Presented by Andreas Olofsson

SiliconCompiler is an open source build system that automates translation from source code to silicon.

“MAKE for silicon”

SiliconCompiler Overview



- Dynamic build schema (json) schema
- Python user API
- Flowgraph based execution model
- File based IPC
- Drivers for all flow tools (executables)
- Run-time tracking of all actions/metrics
- Tested with ASIC and FPGA flows

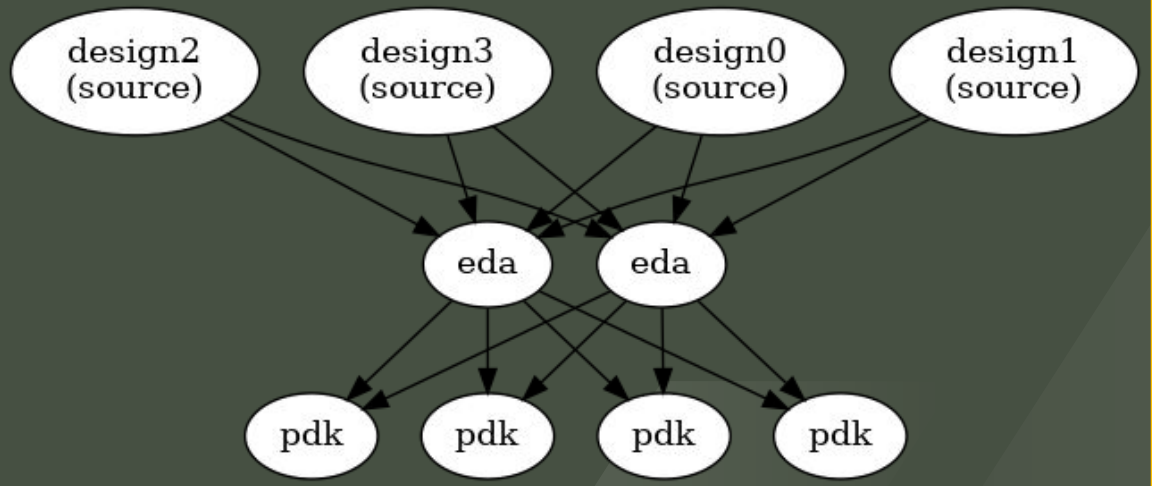
Designer view of SiliconCompiler

```
1  import siliconcompiler                          # import python package
2  chip = siliconcompiler.Chip()                   # create chip object
3  chip.target("asicflow_freepdk45")               # load predefined target
4  chip.add('source','counter.v')                  # define list of sources
5  chip.add('constraint','counter.sdc')            # define constraints
6  chip.add('def','counter.def')                   # define floorplan
7  chip.run()                                      # run compilation
8  chip.summary()                                  # print results summary
```

- Based on nested dictionary that can describe any deterministic process
- Permissive open source: (https://github.com/siliconcompiler)
- Extensively documented (https://docs.siliconcompiler.com/en/latest/)

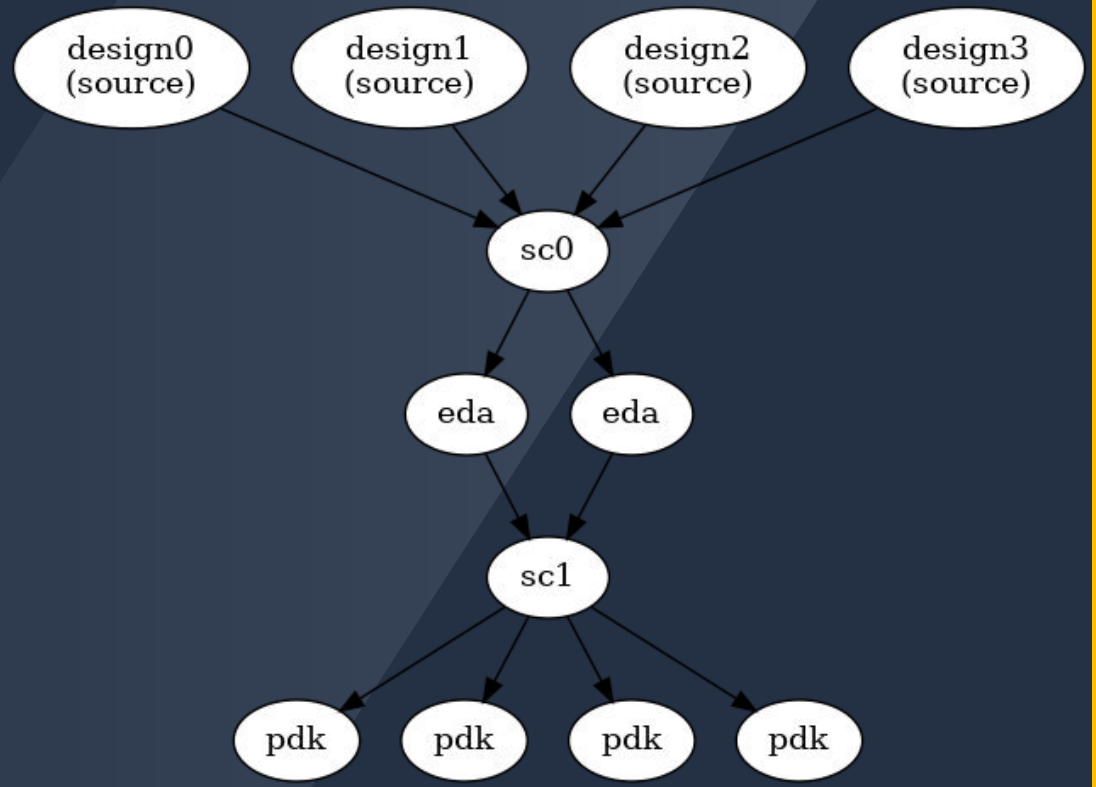
IDEA #1: Solving the "n^2" translation problem with ("LLVM IR for CAD")

Status Quo



Total Flows = D x E x P
D = Designs, E = EDA vendors, P = PDKs
Total Flows (D=100,E = 4,P = 10) = **4,000**

Our Approach



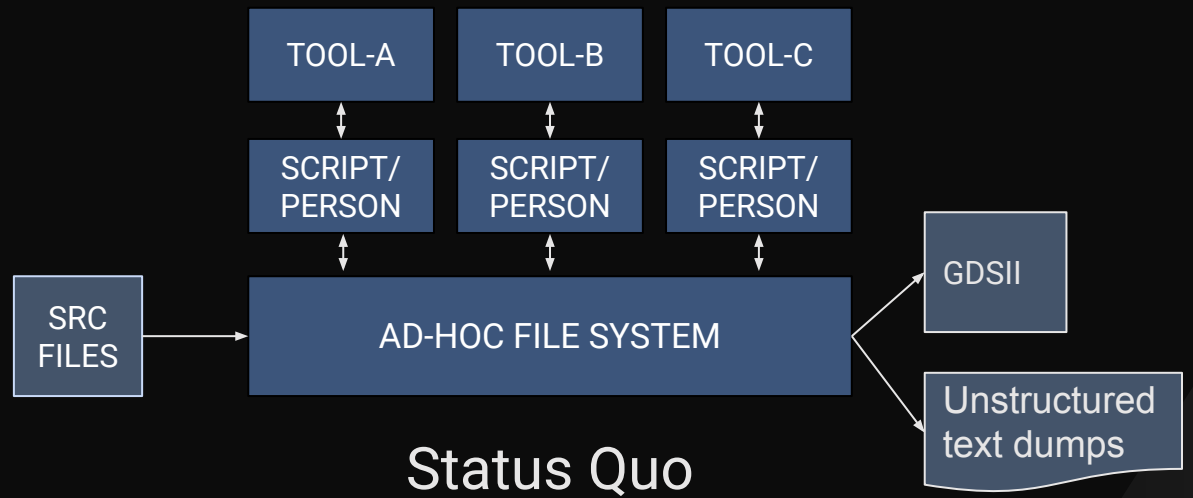
Total Flows (D=100,E = 4,P = 10) = **114**

IDEA #2: Executable compilation schema

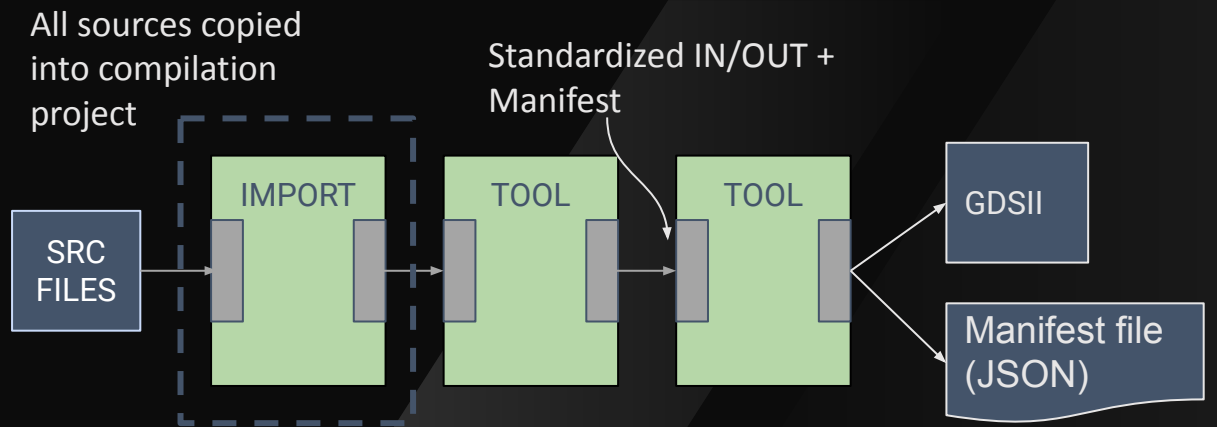
```
cfg['source'] = {
    'switch': None,
    'type': '[file]',
    'lock': 'false',
    'copy': 'true',
    'requirement': None,
    'defvalue': [],
    'filehash': [],
    'hashalgo': 'sha256',
    'date': [],
    'author': [],
    'signature': [],
    'shorthelp': 'Primary source files',
    'example': ["cli: hello_world.v",
                "api: chip.set('source', 'hello_world.v')"],
    'help': """
    A list of source files to read in for elaboration. The files are read
    in order from first to last entered. File type is inferred from the
    file suffix.
    (\\*.v, \\*.vh) = Verilog
    (\\*.vhd)       = VHDL
    (\\*.sv)        = SystemVerilog
    (\\*.c)         = C
    (\\*.cpp, .cc) = C++
    (\\*.py)        = Python
    """
}
```

- **Atomic parameter definitions**
  - Extensions trivial
- **Auto-generated documentation**
  - Always up to date
- **Built-in examples**
  - Use for auto-docs and testing
- **Types**:
  - Object oriented security/abstraction
- **Hashing**:
  - Built into every file parameter
- **Policy control**:
  - Locking, copying, requirement
- **Tracking**:
  - Author signature

IDEA #3: Design manifests



Status Quo



Silicon Compiler

Record feature	SC	Question
Single file manifest	Yes	How as the mfg data created?
PDK, EDA, SC, design version control	Yes	What data was used?
File location	Yes	Where did the file come from?
Origin/author/userid	Yes	Who worked on the design?
File hashing	Yes	What version of file was used?
Data stamps	Yes	When were data files produced?
Tool version checking	Yes	What SW was used?
System info	Yes	What system was it run on?

IDEA #4: Standardizing provenance records

Feature	Description
author	Task author name
userid	Task useride
publickey	Public key (in case of data encryption)
exitstatus	Exit status (success/failure)
org	Name of organization
location	Location of user/organization
toolversion	Version of task tool
starttime, endtime	Start and end times
machine	Name of compute node
region	Cloud region
macaddr	Mac addr
ipaddr	IP addr
platform	OS platform (Windows, macos, linux)
distro	Distribution name
osversion, kernelversion	OS and kernel version
arch	HW arch ((x86_64, armv8)

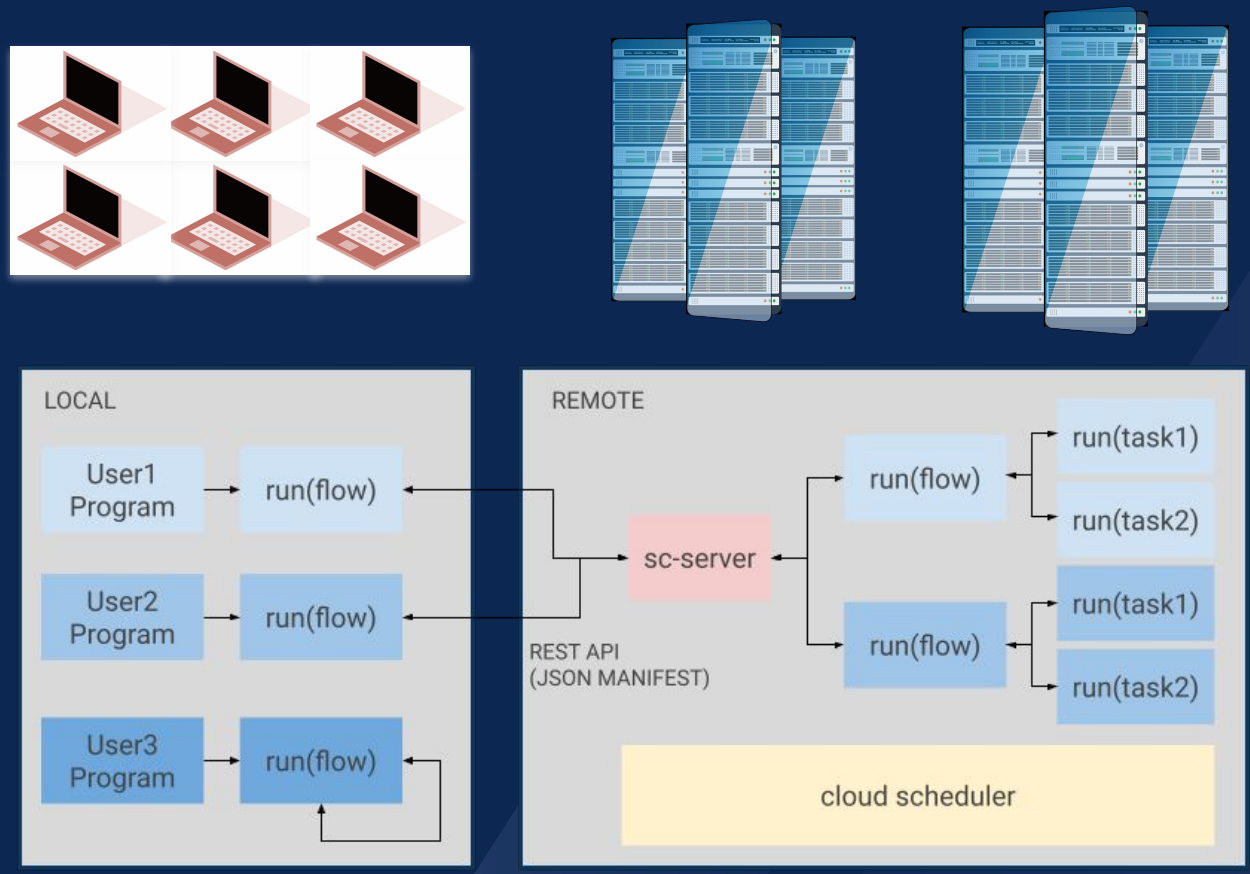
IDEA #5: Standardized compiler metrics to feed into Python ML

```
SUMMARY:

design : gcd
params : None
jobdir : /home/aolofsson/work/zeroasic/siliconcompiler/build/gcd/job0
foundry : virtual
process : freepdk45
targetlibs : NangateOpenCellLibrary
```

	import0	syn0	floorplan0	physyn0	place0	cts0	route0	dfm0	export0
errors	0	0	0	0	0	0	0	0	0
warnings	0	72	1	0	2	3	3	0	0
drvs	0	0	0	0	0	0	0	0	0
unconstrained	0	0	0	0	0	0	0	0	0
luts	0	0	0	0	0	0	0	0	0
dsps	0	0	0	0	0	0	0	0	0
brams	0	0	0	0	0	0	0	0	0
cellarea	0.0	413.63	414.0	414.0	490.0	499.0	499.0	499.0	0.0
totalarea	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
utilization	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
peakpower	0.0	0.0	0.000188	0.000188	0.000207	0.000279	0.000291	0.000292	0.0
standbypower	0.0	0.0	8.62e-06	8.62e-06	1.13e-05	1.17e-05	1.17e-05	1.17e-05	0.0
irdrop	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
holdslack	0.0	0.0	0.11	0.11	0.11	0.11	0.11	0.11	0.0
holdwns	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
holdtns	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
holdpaths	0	0	0	0	0	0	0	0	0
setupslack	0.0	0.0	0.95	0.95	1.11	1.11	1.09	1.09	0.0
setupwns	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
setuptns	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
setuppaths	0	0	0	0	0	0	0	0	0
cells	0	249	363	363	416	421	1869	1869	0
registers	0	0	0	0	0	0	0	0	0
buffers	0	0	0	0	0	0	0	0	0
transistors	0	0	0	0	0	0	0	0	0
nets	0	0	320	320	373	378	378	378	0
pins	0	0	54	54	54	54	54	54	0
vias	0	0	0	0	0	0	2194	0	0
wirelength	0.0	0.0	0.0	0.0	0.0	0.0	6291.0	0.0	0.0
overflow	0	0	0	0	0	0	0	0	0
runtime	0.67	0.99	1.2	1.24	1.45	2.65	2.92	1.31	1.42
memory	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

- Recorded for each task
- Extracted from tool log
- Part of design JSON manifest
- Accessible through Python API at run-time
- Enabler for ML platforms

IDEA #6: Native cloud support



Clients (millions)
- Manual deployment
- Multi-platform front-end
- Human centric metrics

Compute nodes (millions)
- Automated deployment
- Compute back-end
- Machine centric metrics

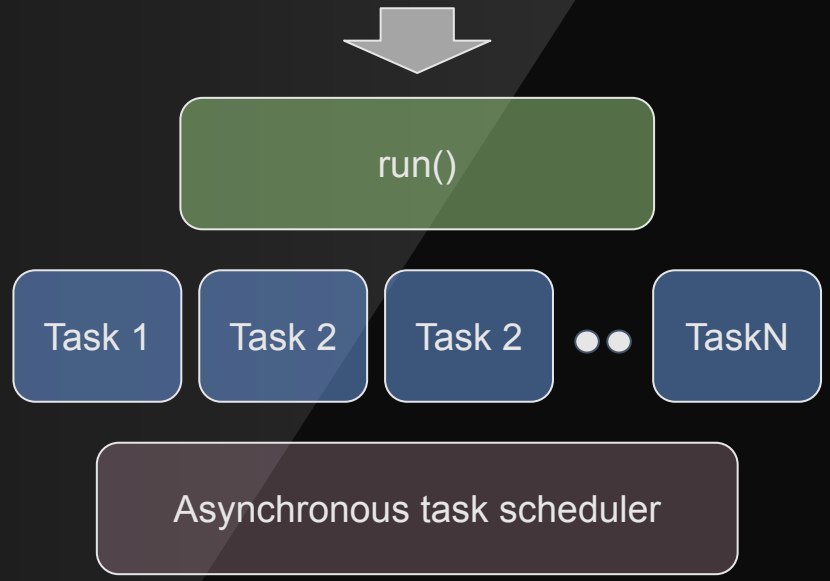
Observations:
- Linux has 1% client market share
- Installation is a high barrier
- NDAs are high barriers
- Cloud compute is elastic
- Cloud compute is almost limitless

Architecture Decisions:
- Make client thin + multiplatform
- Local/remote has common import step
- File based IPC
- Granular async execution model
- Make user interface platform agnostic
- But you have to pick something, so we picked Python…

IDEA #7: A parallel programming model for CAD

```
  export ADDITIONAL_LEFS += $(BLOCK_LEFS)
  export ADDITIONAL_GDS += $(BLOCK_GDS)
  export GDS_FILES += $(BLOCK_GDS)
endif

#-------------------------------------------------------------------------
ifeq (, $(strip $(NPROC)))
  # Linux (utility program)
  NPROC := $(shell nproc 2>/dev/null)

  ifeq (, $(strip $(NPROC)))
    # Linux (generic)
    NPROC := $(shell grep -c ^processor /proc/cpuinfo 2>/dev/null)
  endif
  ifeq (, $(strip $(NPROC)))
    # BSD (at least FreeBSD and Mac OSX)
    NPROC := $(shell sysctl -n hw.ncpu 2>/dev/null)
  endif
  ifeq (, $(strip $(NPROC)))
    # Fallback
    NPROC := 1
  endif
endif
export NUM_CORES := $(NPROC)

export LSORACLE_CMD ?= $(shell command -v lsoracle)
ifeq ($(LSORACLE_CMD),)
  LSORACLE_CMD = $(abspath $(FLOW_HOME)/../tools/install/LSOracle/bin/lsc
endif

LSORACLE_PLUGIN ?= $(abspath $(FLOW_HOME)/../tools/install/yosys/share/yc
export LSORACLE_KAHYPAR_CONFIG ?= $(abspath $(FLOW_HOME)/../tools/install
ifneq ($(USE_LSORACLE),)
  YOSYS_FLAGS ?= -m $(LSORACLE_PLUGIN)
endif
```

```
import siliconcompiler
syn_np = 4
chip = siliconcompiler.Chip()
chip.node('import', 'surelog')
chip.node('synmin', 'minimum')
for i in range(syn_np):
    chip.node('syn', 'yosys', index=i)
    chip.edge('import', 'syn', head_index=i)
    chip.edge('syn', 'synmin', tail_index=i)
```



Current

- Make files (1976) + LSF/Grid
- A low level DSL paradigm
- Low productivity and agility

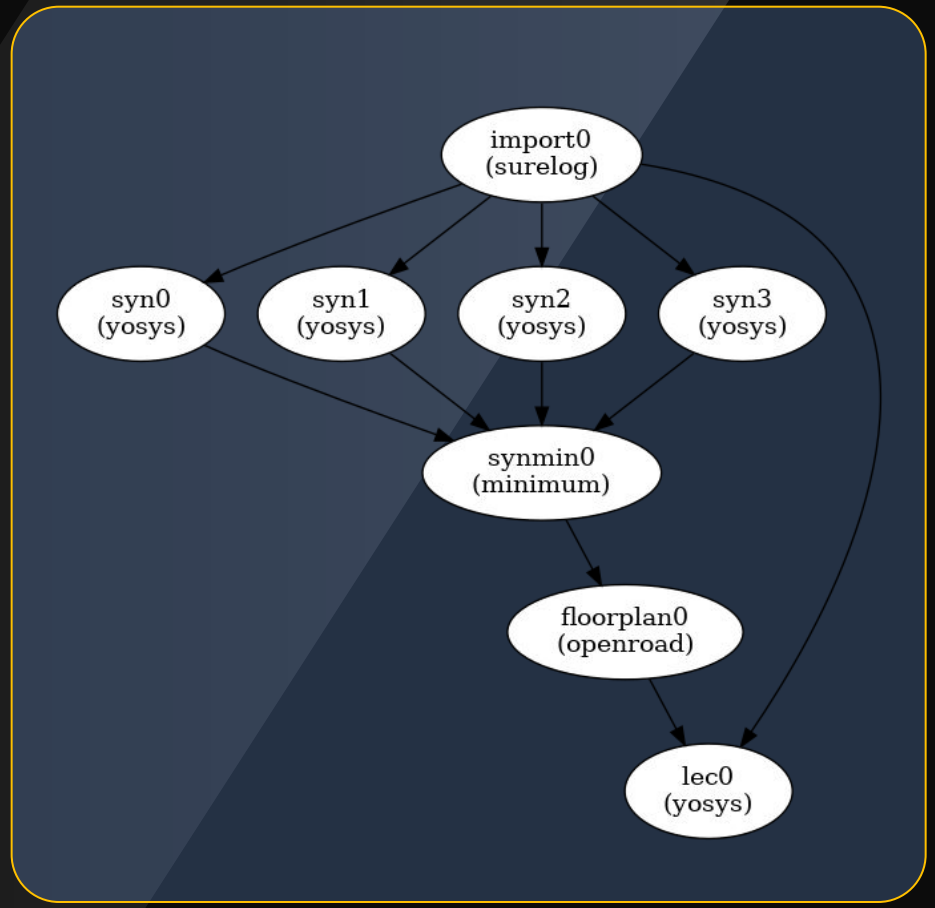
SiliconCompiler

- Task graph based programming model
- Serial graph task launch
- Asynchronous task execution

zero

General DAG Pattern

```python
import siliconcompiler
syn_np = 4
chip = siliconcompiler.Chip()

# nodes
chip.node('import', 'surelog')
for i in range(syn_np):
    chip.node('syn', 'yosys', index=i)
chip.node('synmin', 'minimum')
chip.node('floorplan', 'openroad')
chip.node('lec', 'yosys')

# edges
for i in range(syn_np):
    chip.edge('import', 'syn', head_index=i)
    chip.edge('syn', 'synmin', tail_index=i)
chip.edge('synmin', 'floorplan')
chip.edge('floorplan', 'lec')
chip.edge('import', 'lec')
chip.write_flowgraph("pattern_general.png")
```



IDEA #8: Auto-generated documentation

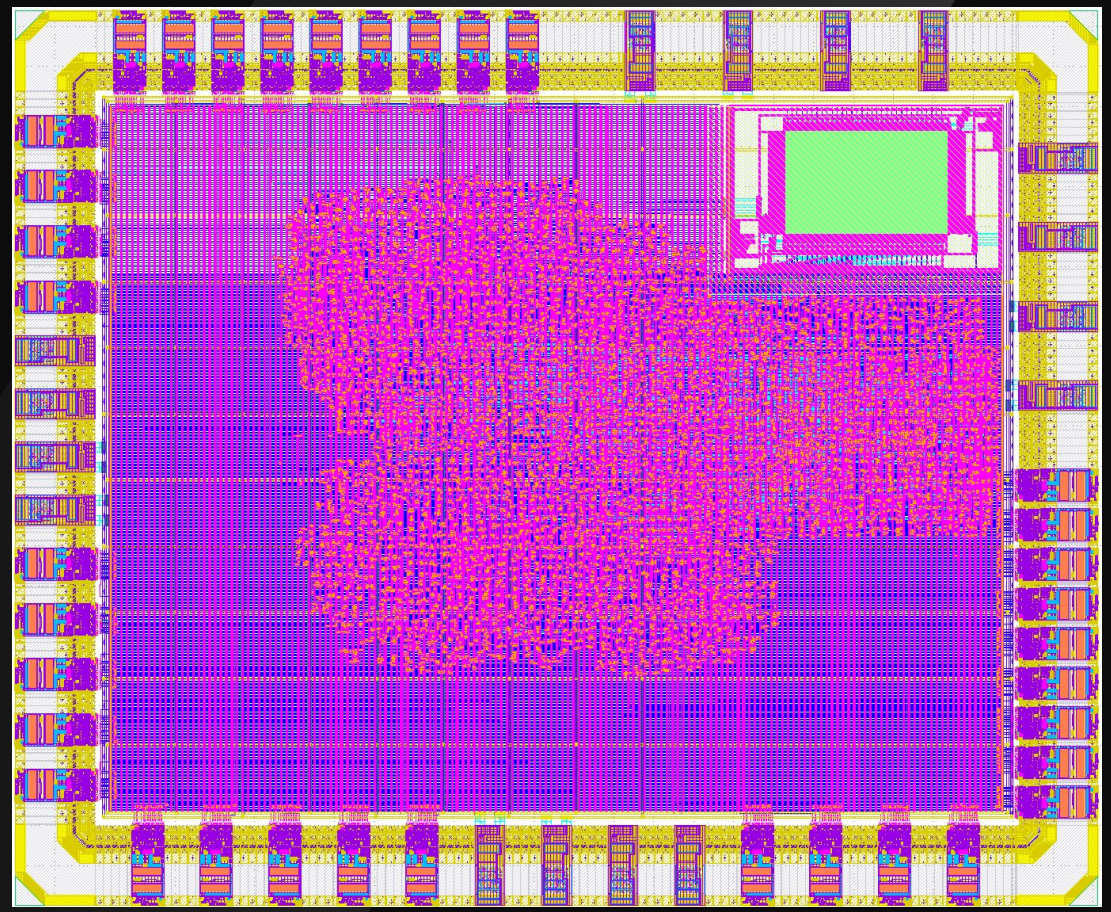
- The vast majority of open source software projects have AWFUL documentation!

- Solve problem by architecting project for auto-generated documentation

https://docs.siliconcompiler.com

ZeroSOC proof of concept demonstration

- OpenTitan RISC-V based SoC
- Full chip hierarchical design
- Fully automated SiliconCompiler flow
- Leverages OpenRoad, Yosys, Surelog, Magic
- Fully open source tools
- Fully open source design
- Fully open source PDK (skywater 130nm)
- LVS/DRC clean chip at skywater130
- https://github.com/siliconcompiler/zerosoc



Status (Jan 2021)

Feature	Metric
Popularity	15K downloads, 276 github stars
LOC	12,000
Parameters	306
User API methods	50+
Documentation	300 pages (docs.siliconcompiler.com)
PDKs tested	5
Tools tested	20 open source / 25 proprietary
Platforms supported	Redhat, Centos, Ubuntu, Windows, MacOS
Python version supported	3.6 - 3.10

SiliconCompiler Value Proposition

Feature	SiliconCompiler	Current CAD	Metric	Value
User interface	Python	TCL/Make	#developers	1000x larger workpool
Open API/schema	Yes	No	1/0	Speed/agility
Single file provenance manifest	Yes	No	1/0	QA
Cloud scale automation	Python, slurm,..	No, Make, proprietary	10x improvement in runtime	Speed/agility
Cloud ready client/server architecture	Yes	No	Zero install ramp up	Speed/agility
Tapeout archiving	Automated	Manual, ad/hoc	1/0	QA
PDK/EDA agnostic setup	Yes	Not usually	1/0	Speed/agility
Unified FPGA/ASIC schema	Yes	No	1/0	Speed/agility