

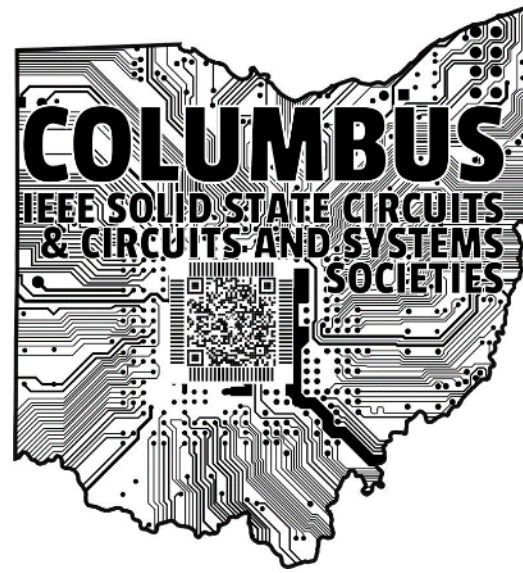
Columbus Joint Chapter  
SSC37 / CAS04

Presentation will  
begin at 6:15 EST



# Columbus Section Joint Chapter, SSC37/CAS04

- Founded on June 30, 2020
- Technical talks and workshops hosted year-round
- Previous talks covering a myriad of topics posted on our [website](#)



IEEE  
**SOLID-STATE  
CIRCUITS SOCIETY™**  
IC Innovation



**CAS**  
IEEE CIRCUITS AND SYSTEMS SOCIETY

[TinyTapeout](#) workshop series  
and shuttle-run sponsorship  
Chapter Application: 9/1/24  
Tapeout: 11/8/24



[SSCS sponsored Arduino  
hardware development  
contest!](#)

Completion Deadline: 8/1/24



# CASS Outreach Program 2023

## 1: Dr. Suat Ay

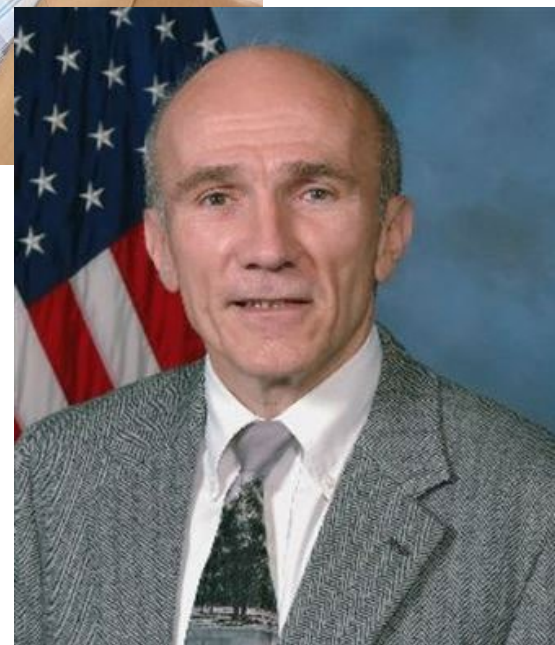
- Professor at the University of Idaho since 2007
- 10+ years of industry experience at Photobit/Micron imaging



Dr. Suat Ay (left)

## 2: Dr. Marvin White

- Professor at the Ohio State University
- Inventor of Correlated Double Sampling (CDS)



Dr. Marvin White (right)

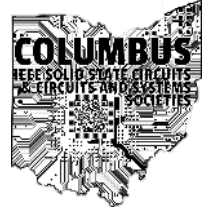
## 3: Dr. Paul McManamon

- President and CTO at Exciting Technology
- Former Chief Scientist at AFRL



Dr. Paul McManamon

# SSCS Educational Chapter of the Year Award



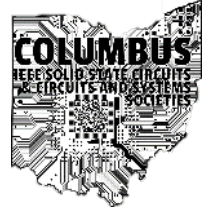
- Our chapter was lucky enough to win the Best Educational Program Award
- Our chapter chair Ramy Tantawy accepted the award at the SSCS flagship conference (ISSCC)
- The chapter was awarded \$2000 in total for continued workshops and outreach



Have an idea for a talk or workshop?

- Contact us!
- [columbus.sscs.cas@gmail.com](mailto:columbus.sscs.cas@gmail.com)

# New Officer Announcement!

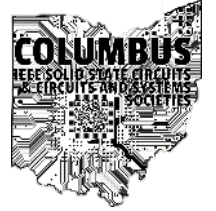


- Dr. Shane Smith has joined the officer team as the Vice Chair!
  - IC design, test, packaging, and integration **subject matter expert**
- Vice chair responsibilities
  - Outreach engagement
  - Sponsorship coordination
- Stay tuned for an exciting workshop over the summer from Dr. Smith on PCB design!

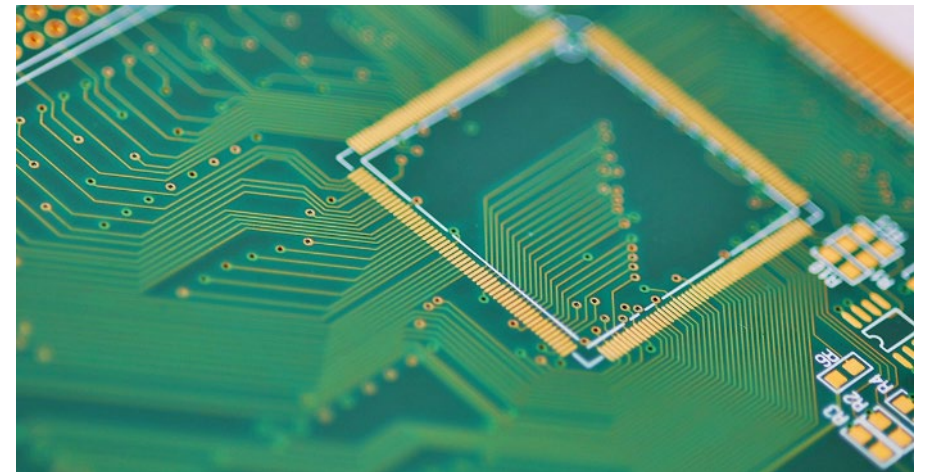
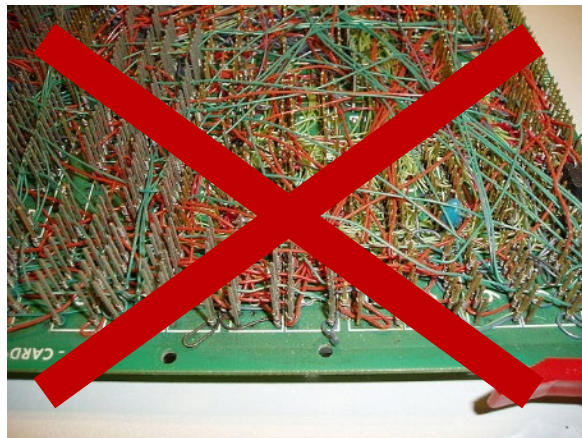


<https://r2.ieee.org/columbus-ssccas/officers/>

# CASS Outreach Program 2024



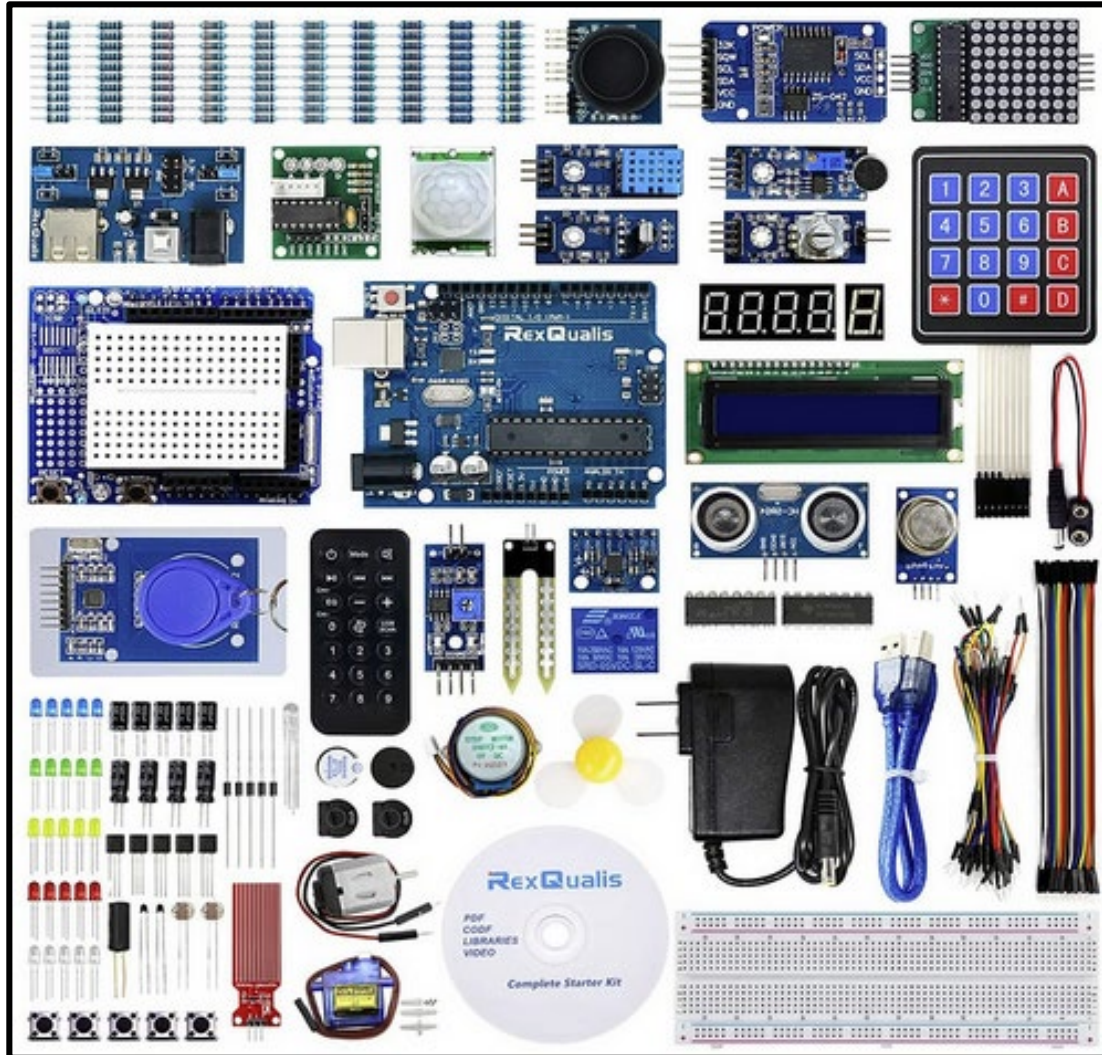
- Foundations of Mixed-Signal IC Design:  
A Practical Approach to Lab-to-Fab” series
  - Fundamental building blocks in Analog/Mixed Signal SoC’s lectures
  - Tiny Tapeout workshops
  - Printed circuit board design and Arduino workshop



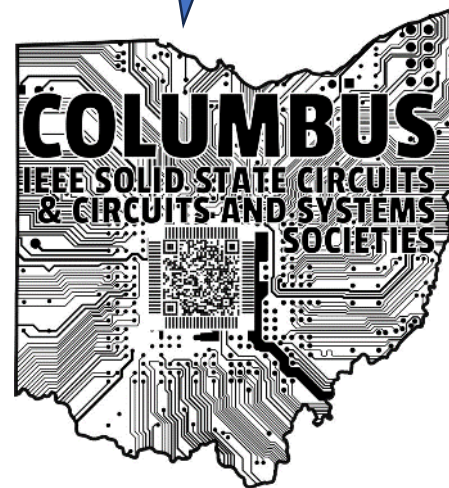


# Opportunities for Engagement

Starter Kits for <\$100!



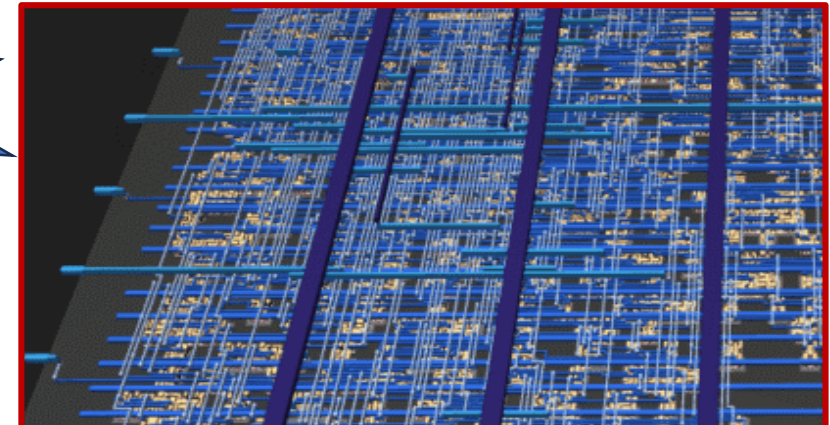
WE CAN SPONSOR YOU!



SenseICs

\$150\*-300 for PCB & ASIC!

\* Early-bird special



# Scan this QR code to fill out the Form to participate!



Sponsorship Application

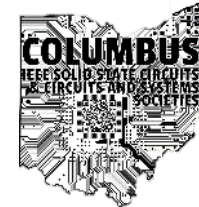
<https://forms.gle/ypWKDA4zrj8zKAR9A>



Event Feedback Form

<https://forms.gle/g5SsnPgFxGNzazYM8>





# Looking For Sponsors

- Our chapter is always looking for sponsorship to increase our educational and outreach footprint
- If you are interested in sponsoring our chapter at any contribution level, please reach out to the leadership directly or at the below email

[columbus.sscs.cas@gmail.com](mailto:columbus.sscs.cas@gmail.com)



**SenseICs**

# CASS New Member Initiative

- Free membership until 08/2024!
- IEEE membership required in good standing
- CASS membership add-on accessible from IEEE account
- CASS membership only \$11 for students after 08/2024

## Have an idea for a talk or workshop?

- Contact us!
- [columbus.sscs.cas@gmail.com](mailto:columbus.sscs.cas@gmail.com)

JOIN IEEE CASS
https://iee-cas.org/



## Celebrate our 75th Anniversary with a Complimentary IEEE Circuit and Systems Society Membership for 2024

### Who We Are

The IEEE Circuits and Systems Society is the leading organization that promotes the advancement of the theory, analysis, design, tools, and implementation of circuits and systems. The field spans their theoretical foundations, applications, and architectures, as well as circuits and systems implementation of algorithms for signal and information processing.

### Offer Details

To take advantage of this offer, simply sign in with your IEEE Account. The membership will present in the Cart at US\$0 (This offer is available for a limited time only, and does not apply to the Preferred Package membership).

**\*This offer is valid until the 2024 IEEE membership year, ending in August 2024.**

**Eligibility Criteria:**

- Must have an IEEE membership.
- New IEEE CASS members, who have never been members before

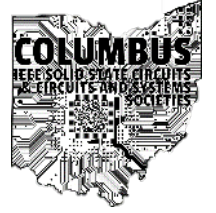
Connect with CASS on social media through Twitter, LinkedIn, and Facebook

### Membership Benefits

This special offer is limited and provides you with all CASS member benefits at the Essential package, including free online access to our financially sponsored journals and conference proceedings in the IEEE Xplore Digital Library, as well as:

- **Discounted registration fees** for all nine CASS flagship and premier conferences
- **Access a wide variety of educational resources** including the CASS Resource Center, CASS Microlearning Program (CASS MiLe), and the CASS-Wide Webinar Series
- **Networking** with over 100 Chapters worldwide
- **Access to 17 Technical Committees and Special Interest Groups as well as 5 CASS Standards Activities Sub-**

# More Opportunities for Open-Source Hardware Development: PICO



- “Platform for IC Design Outreach”
- Deadline is next **Friday, May 10<sup>th</sup>**!
- Similar structure w/ SKY130/GF180 open-source PDKs
- Analog layouts designed and generated in the OpenFASoC [1] environment
- Chaired by Boris Murmann, former Stanford professor and mixed-signal design expert

<https://sscs.ieee.org/about/tc-ose/sscs-pico-design-contest>

<https://www.youtube.com/watch?v=O0J7E198udQ>

[1] OpenFASoC: Fully Open-Source Autonomous SoC Synthesis using Customizable Cell-Based Synthesizable Analog Circuits, <https://github.com/idea-fasoc/OpenFASOC/>

/ Home / About / TC-OSE / SSCS “PICO” Open-Source Chipathon

GOVERNANCE SSCS ADCOM >
IC HISTORY >
SSCS EXECUTIVE OFFICE STAFF
SOLID-STATE CIRCUITS DIRECTIONS >
IEEE TECHNICAL COUNCILS, COMMUNITIES AND INITIATIVES
<b>TC-OSE</b> ✓
SSCS “PICO” Open-Source Chipathon
Committee Members
SSCS PICO Program

## SSCS “PICO” Open-Source Chipathon

### Automating Analog Layout

– Sign-Up Deadline: May 10, 2024 –

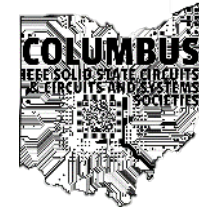
The IEEE Solid-State Circuits Society is pleased to announce its fourth open-source integrated circuit (IC) design contest under the umbrella of its [PICO](#) Program (Platform for IC Design Outreach). While this contest is open to anyone (no restrictions), we encourage the participation of pre-college students, undergraduates, and geographical regions that are underrepresented within the IC design community.

The goal of this year’s event is to advance the automatic generation and open sharing of analog circuit layout cells to increase our community’s design productivity and to catch up with other fields where sharing and automation is a key enabler of progress (e.g., in machine learning).



#### Contest Outline

1. Interested individuals sign up using [this form](#) by May 10, 2024.
2. Phase 1 (~June): Through a series of weekly meet-ups and training sessions, the participants learn to create basic one- or two-transistor layout generators using Python and open-source CMOS PDKs. Using Jupyter Notebooks hosted on Google Colab allows anyone with an internet connection to participate - no downloads or installations required! Relevant circuit examples can be found in [1], [2]. We will leverage code modules available with the OpenFASoC [3] environment.
3. Phase 2 (~July): Interested participants define larger layout building blocks that they wish to automate (examples: comparator, bandgap, phase interpolator, OTA). Teaming among participants is encouraged to maximize collaboration and learning).



# SSC37/CAS04

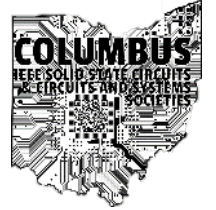
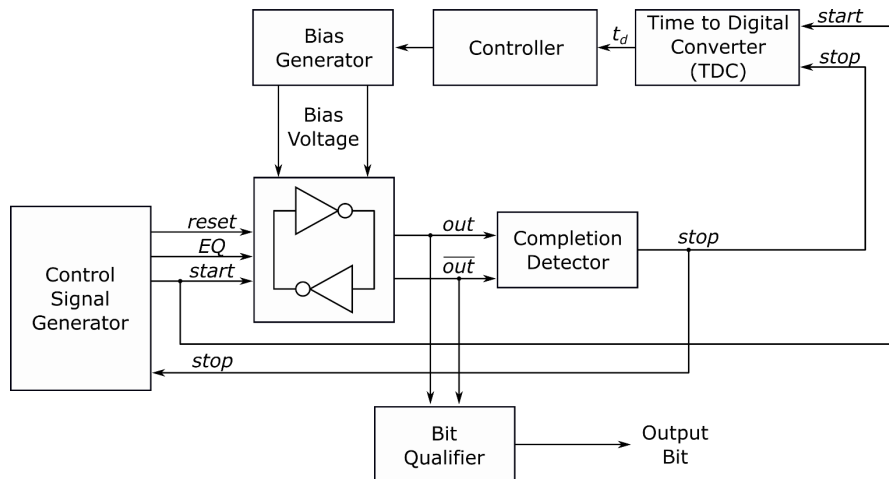
## Columbus Joint-Chapter Seminar

### A Practical Approach to Lab-to-Fab series: Tiny Tapeout Workshop (1)

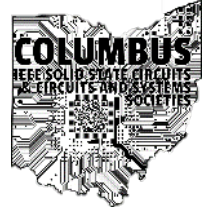
Webinar Location: SenselCs Corporation @ Rev1 Ventures Main  
Conference Room  
Speaker: Sam Ellicott

# About Me

- IEEE Columbus SSCS/CAS Vtools coordinator
- PhD Student at The Ohio State University
  - Circuits Laboratory for Advanced Sensors and Systems
  - RF and Mixed-Signal Integrated Circuit (IC) Design
  - True Random Number Generators
- BSEE at Cedarville University (2019)
- Intern at Analog Devices

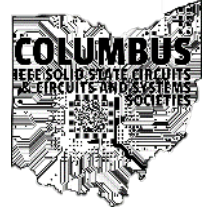


# Workshop Series Goals



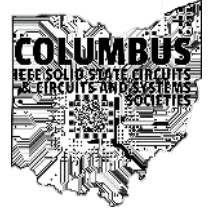
- Understand the workflow for open-source tools
  - Demystify the steps required to generate digital designs
- Hands-on introduction to digital design
  - Crash course to Verilog
  - Ability to design/test simple modules
  - Make a simple project
- Have fun!

# Workshop Series Outline



- Workshop 1: Introduction
  - Today
- Workshop 2: Tooling and Series Project
  - September/October
- Workshop 3: Design Review
  - October/November

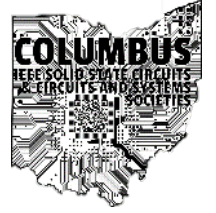
# Workshop Series Outline



- Workshop 1: Introduction (tonight)
  - What are Integrated Circuits (ICs)
  - Brief History of ICs
  - Introduction to Digital Design
  - Introduction to Tiny Tapeout
- Workshop 2: Tooling and Series Project
- Workshop 3: Design Review



# Formatting Note

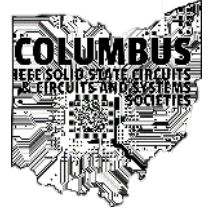


- Some slides are to aid understanding
  - Provide background knowledge that I think is interesting
  - Not required for day-to-day design work
- Background information slides
  - Title in red
- Terminal commands
  - Commands are in *bold Italic*



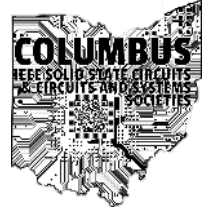
# Introduction to Integrated Circuits

# Hands up if

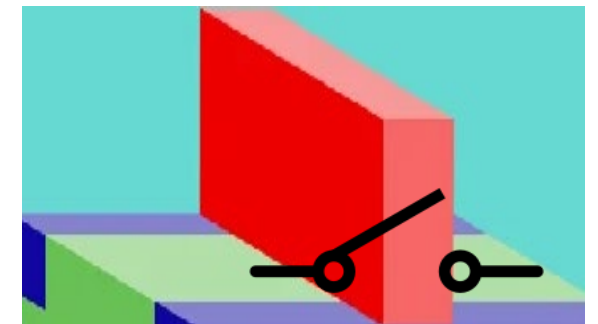
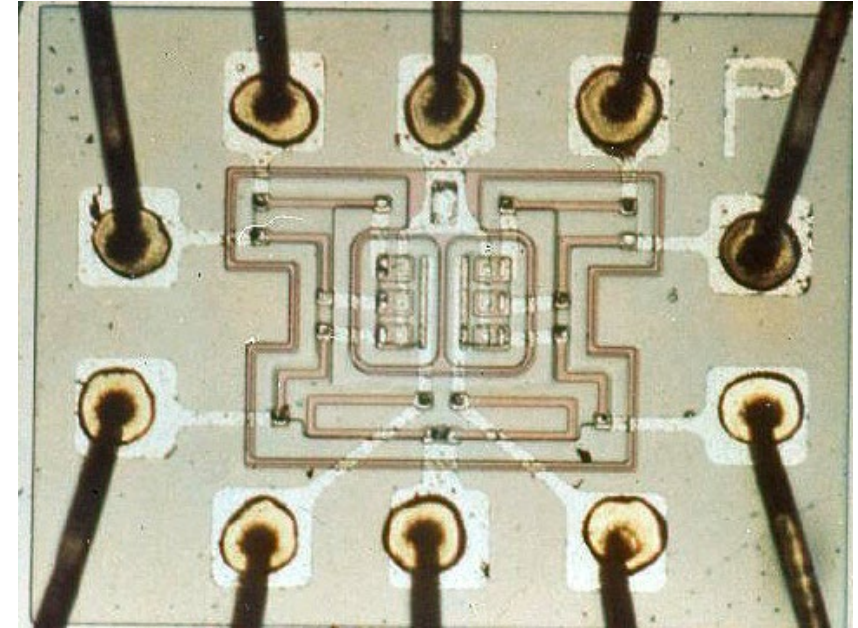


- You've used an app
- You've written a computer program
- Used an Arduino (or similar)
- Designed a chip
- Had your own chip manufactured
- Manufactured your own chip

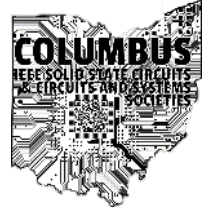
# Some Definitions



- What is an integrated circuit?
  - Also called IC or chip
  - Multiple transistors all in the same substrate
  - Connections between transistors
- What is a transistor?
  - For digital circuits: a electronic controllable switch
  - Things get more complicated for analog circuits

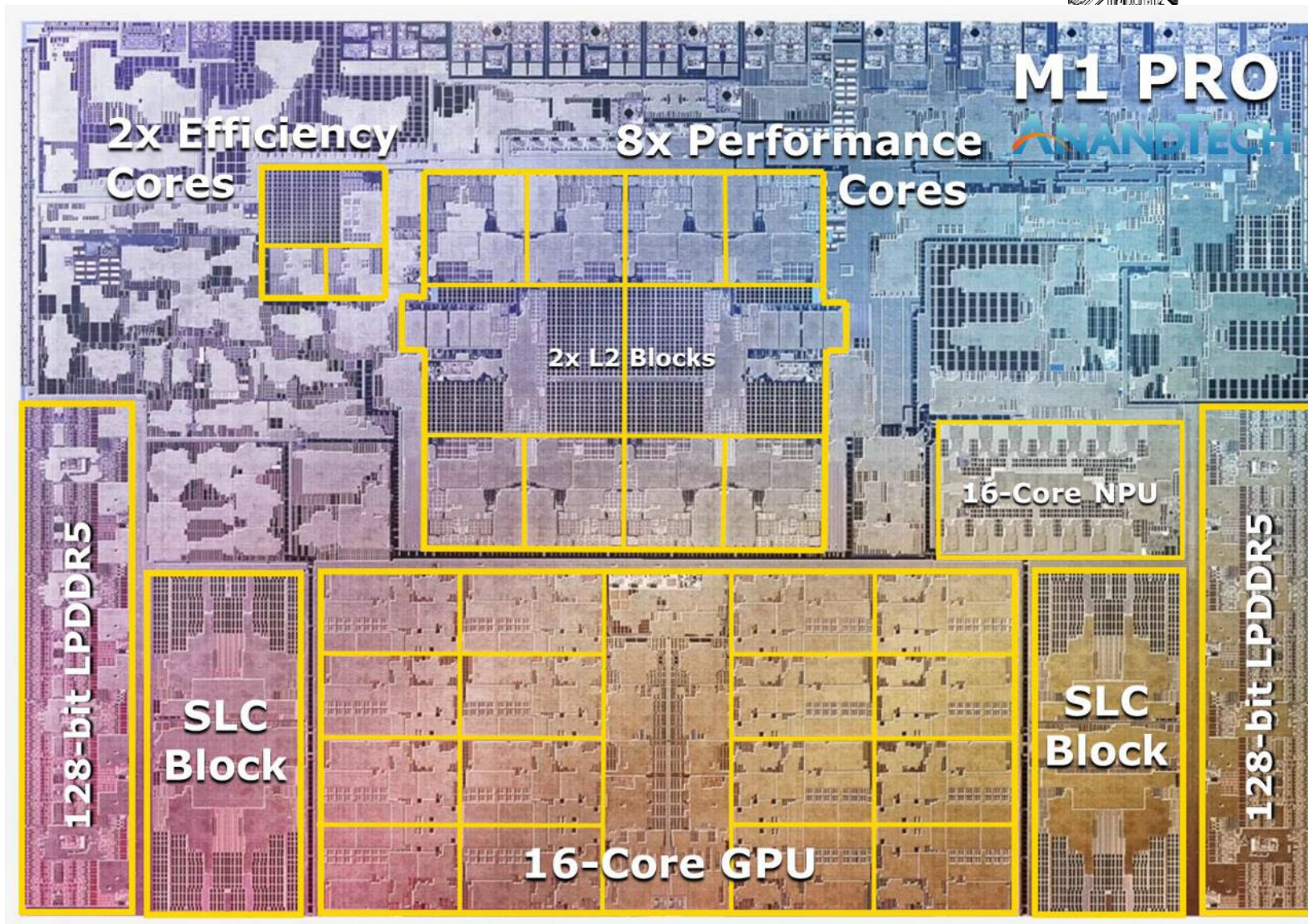


# Why do we care?



# A modern IC (Apple M1 Pro)

- 33.7 Billion Transistors
- $245\text{mm}^2 \rightarrow 16\text{mm}$  square
- $5\text{nm}$  process node



<https://www.anandtech.com/show/17019/apple-announced-m1-pro-m1-max-giant-new-socs-with-allout-performance>

# A Sense of Scale

$10^{+0}$  meters The sleeping man at the picnic



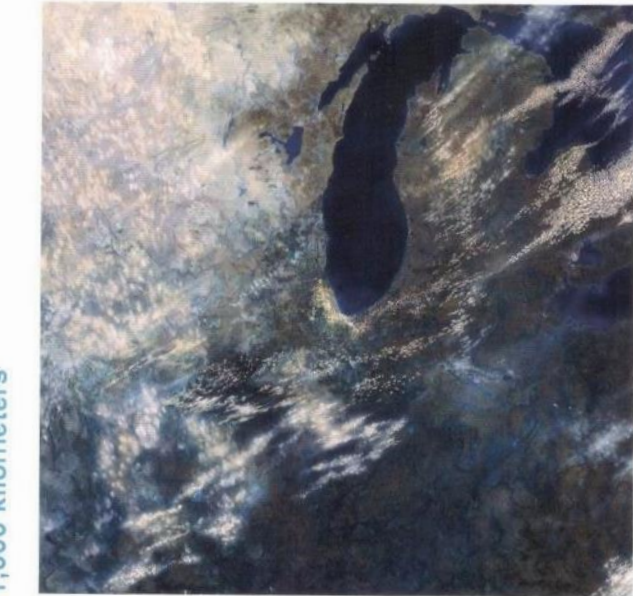
1 meter

$10^{+3}$  meters Soldier Field and ...



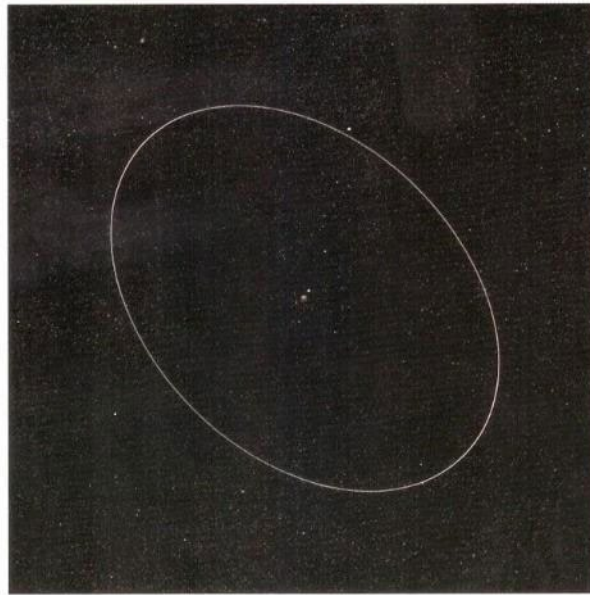
1 kilometer

$10^{+6}$  meters Lake Michigan is fully visible



1,000 kilometers

$10^{+9}$  meters The orbit of the Moon ...

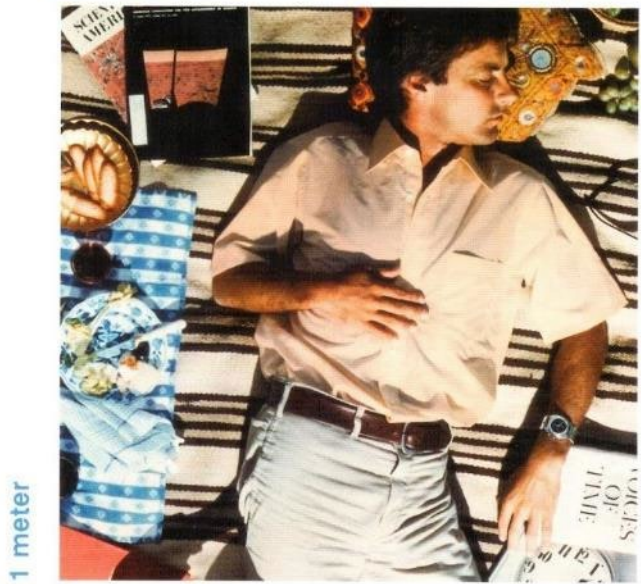


1 million kilometers

Powers of ten  
Charles and  
Ray Eames.

# A Sense of Scale

$10^0$  meters The sleeping man at the picnic



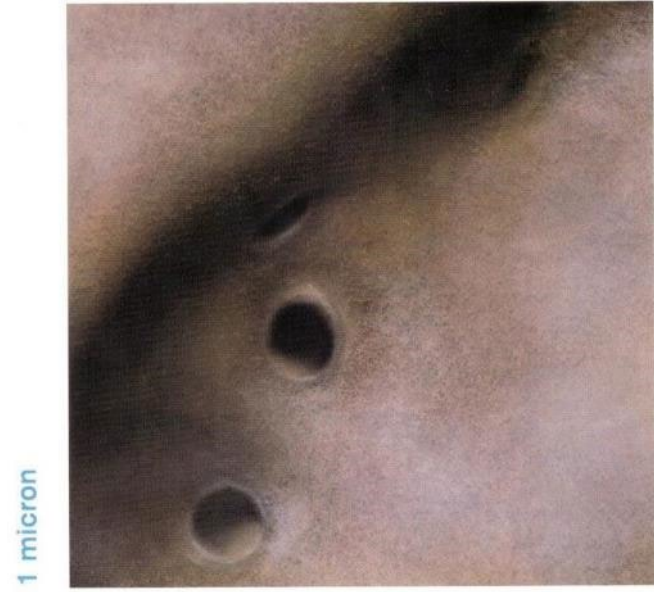
1 meter

$10^{-3}$  meters Within the surface of the skin



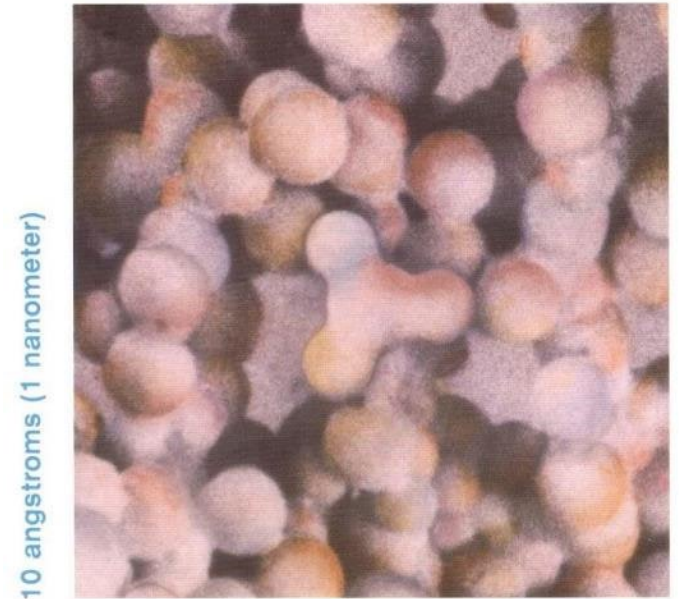
1 millimeter

$10^{-6}$  meters The wall of that cell's nucleus



1 micron

$10^{-9}$  meters Building blocks of DNA . . .

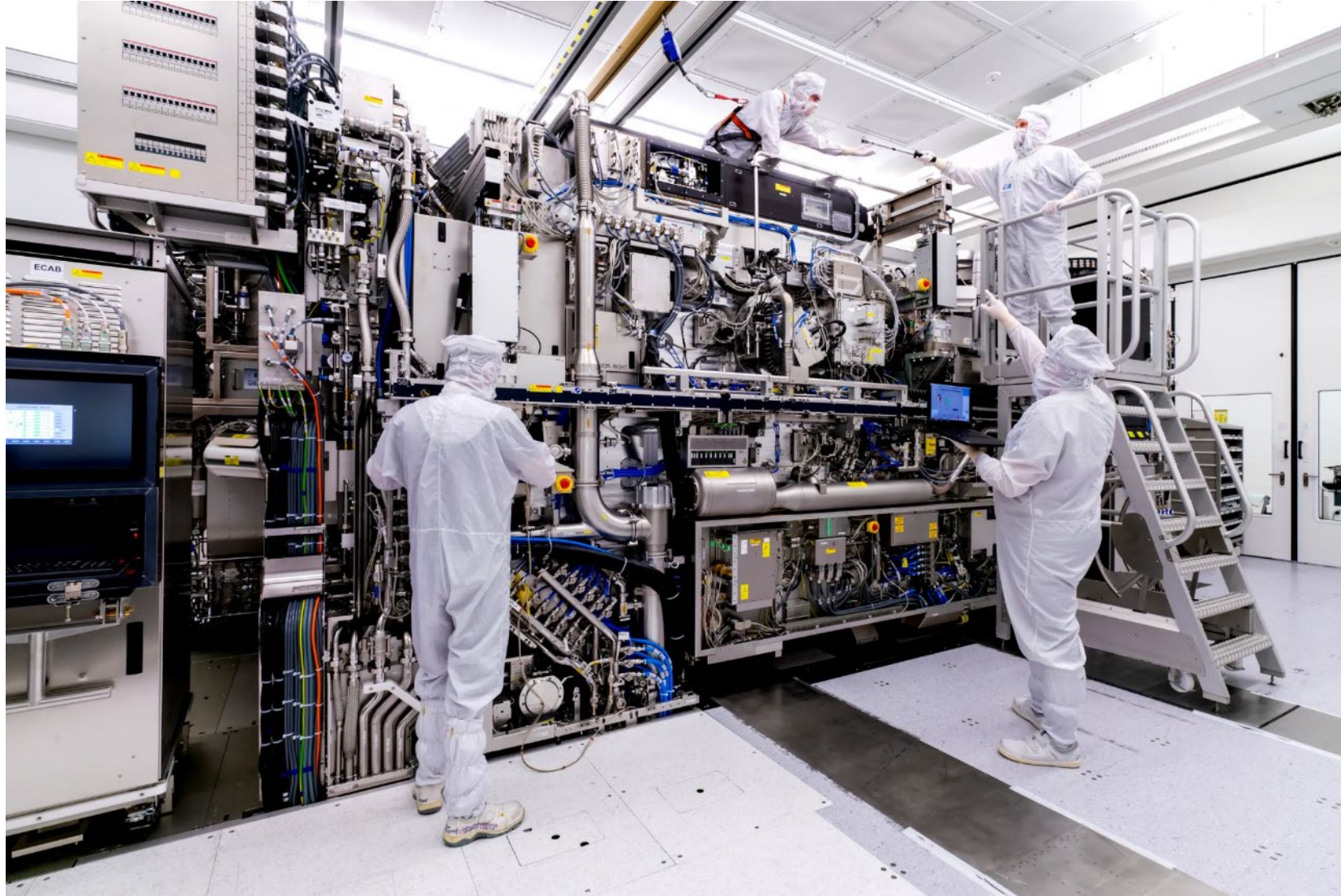
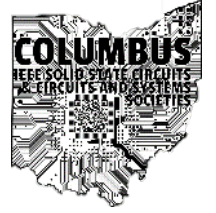


10 angstroms (1 nanometer)

Powers of ten  
Charles and  
Ray Eames.



# Extreme Ultraviolet Lithography



[IHP Factory Tour](#)

[Multi Patterning](#)

[ASML EUV Lithography](#)

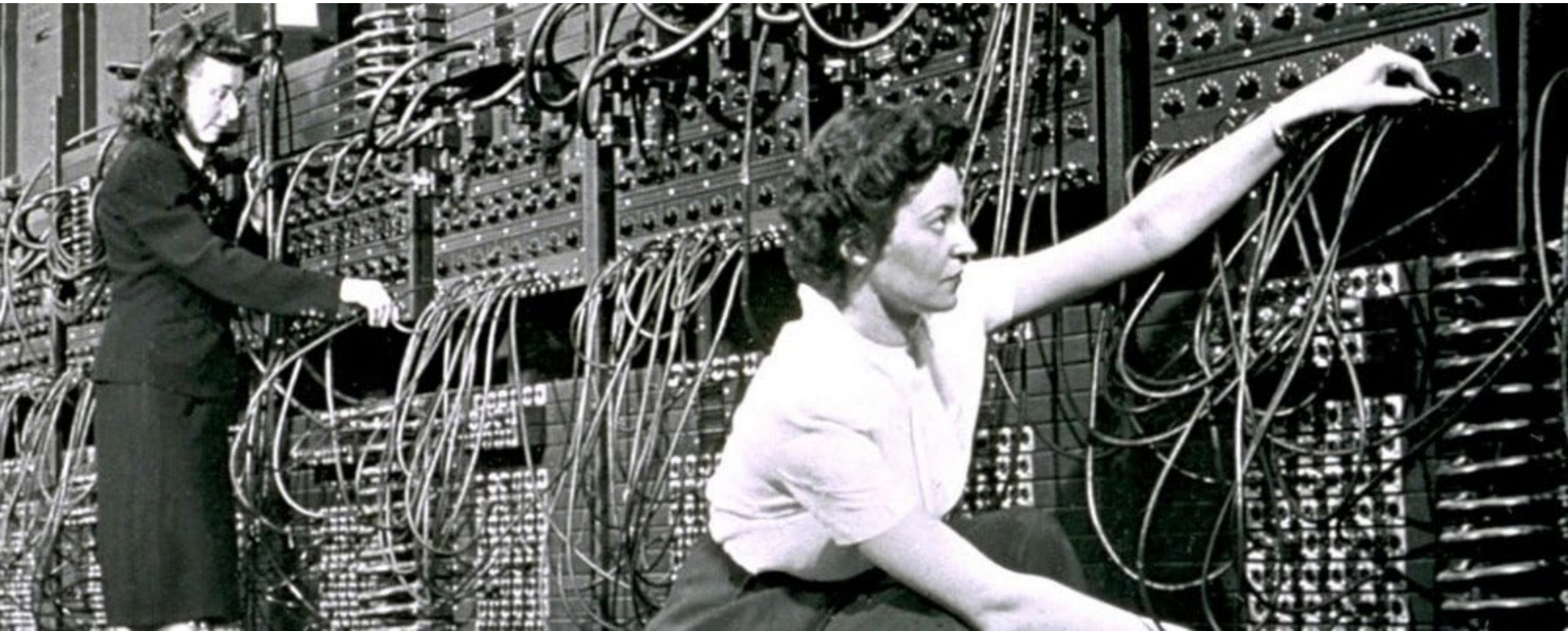
[History of EUV Lithography](#)

[History of IC Lithography](#)



# History of Integrated Circuits

# Before Transistors: 1944

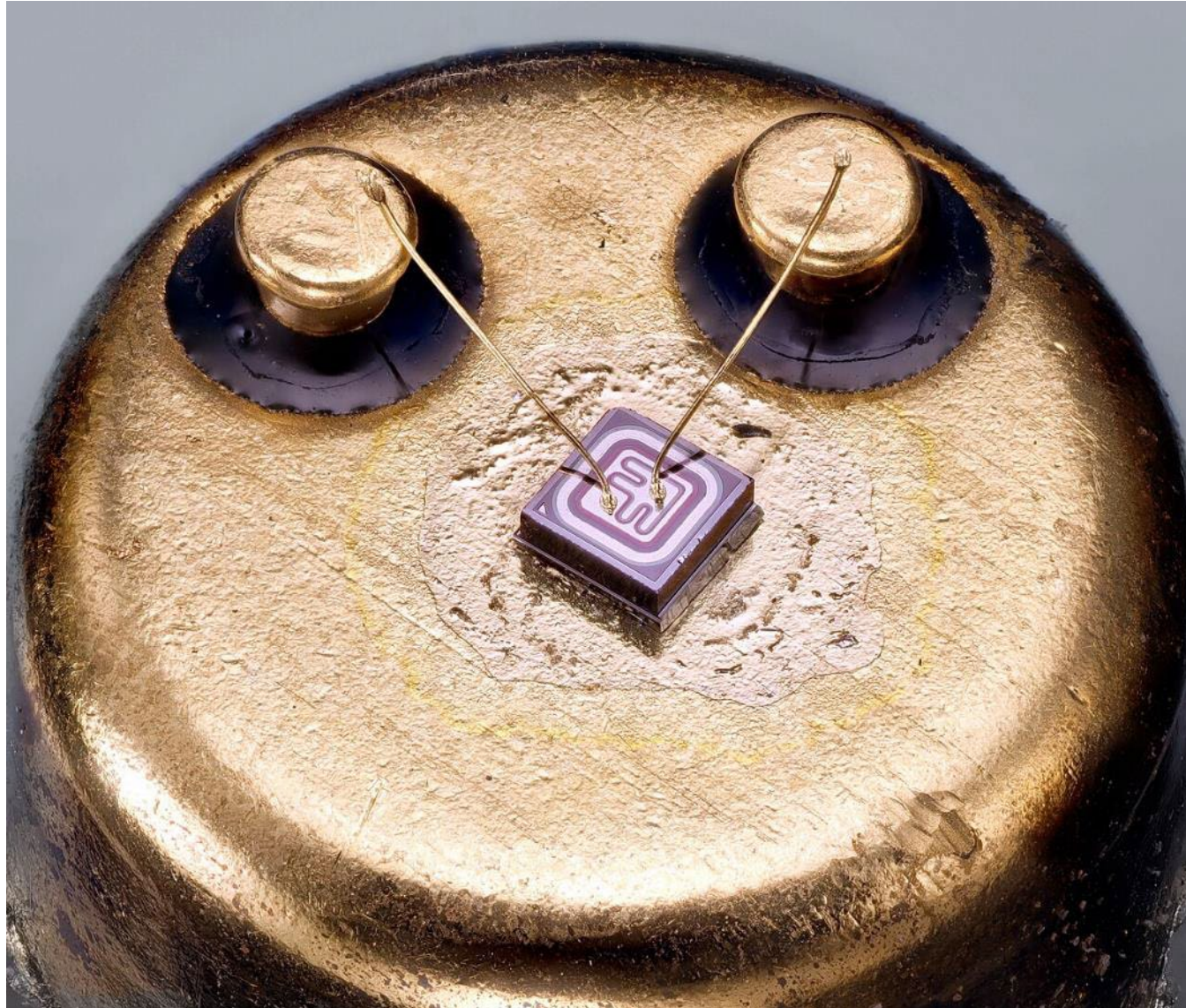


Electronic Numerical Integrator and Computer: programmable for artillery calculations

# The First Transistor: 1947

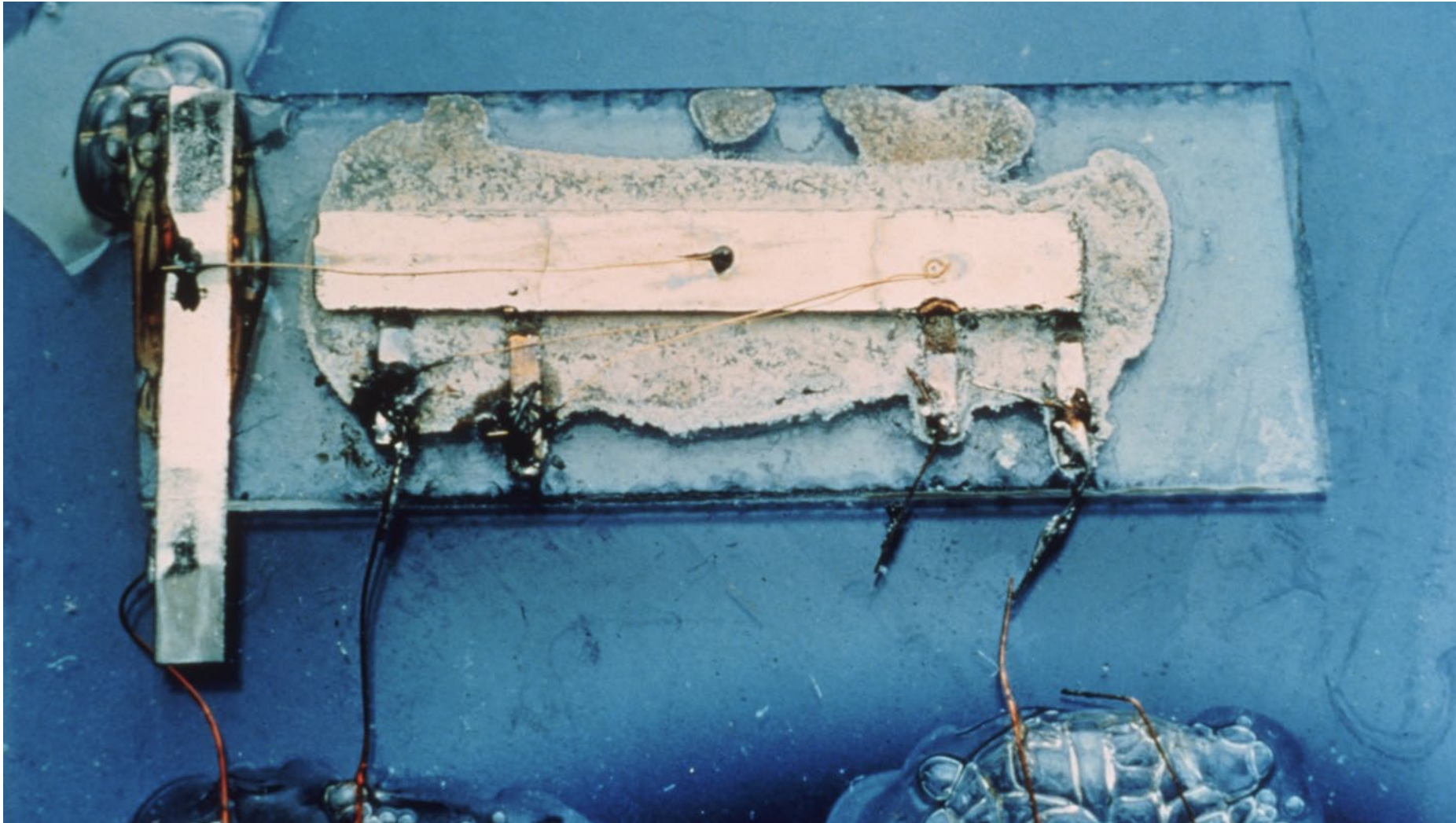


# Discrete Transistor



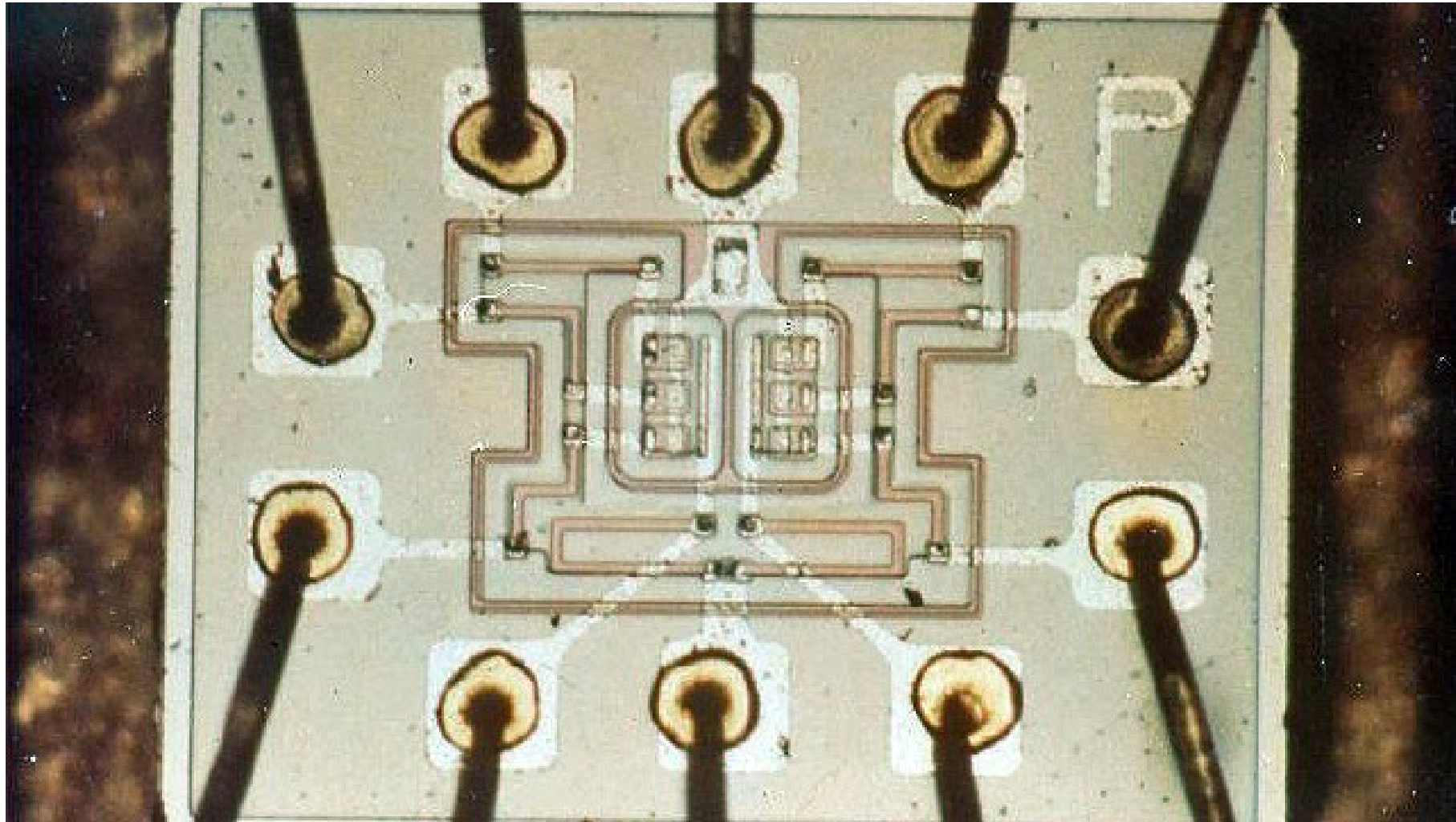
[https://en.wikipedia.org/wiki/Bipolar\\_junction\\_transistor#/media/File:IPRS\\_BANEASA\\_2N2222.jpg](https://en.wikipedia.org/wiki/Bipolar_junction_transistor#/media/File:IPRS_BANEASA_2N2222.jpg)

# Integrated Circuits: 1958



Jack Kilby 1958 at Texas Instruments

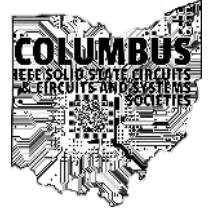
# Integrated Circuits: 1959



Robert Noyce at Fairchild Semiconductor

<https://www.rit.edu/imagine/exhibit-extras/Apollo-Guidance-Computer-ImagineRIT-SKurinec.pdf>

# Shrinking Transistors and Technologies



**1950s**

Silicon Transistor



**1**  
Transistor

100  $\mu\text{m}$

**1960s**

TTL Quad Gate



**16**  
Transistors

10  $\mu\text{m}$

**1970s**

8-bit Microprocessor

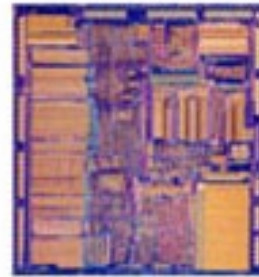


**4500**  
Transistors

1  $\mu\text{m}$

**1980s**

32-bit Microprocessor



**275,000**  
Transistors

**1990s**

32-bit Microprocessor

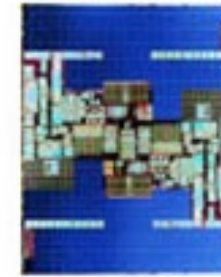


**3,100,000**  
Transistors

100 nm

**2000s**

64-bit Microprocessor

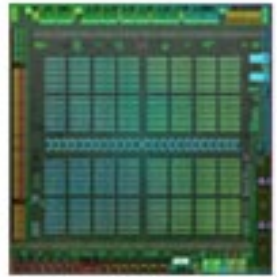


**592,000,000**  
Transistors

10 nm

**2010s**

3072-Core GPU



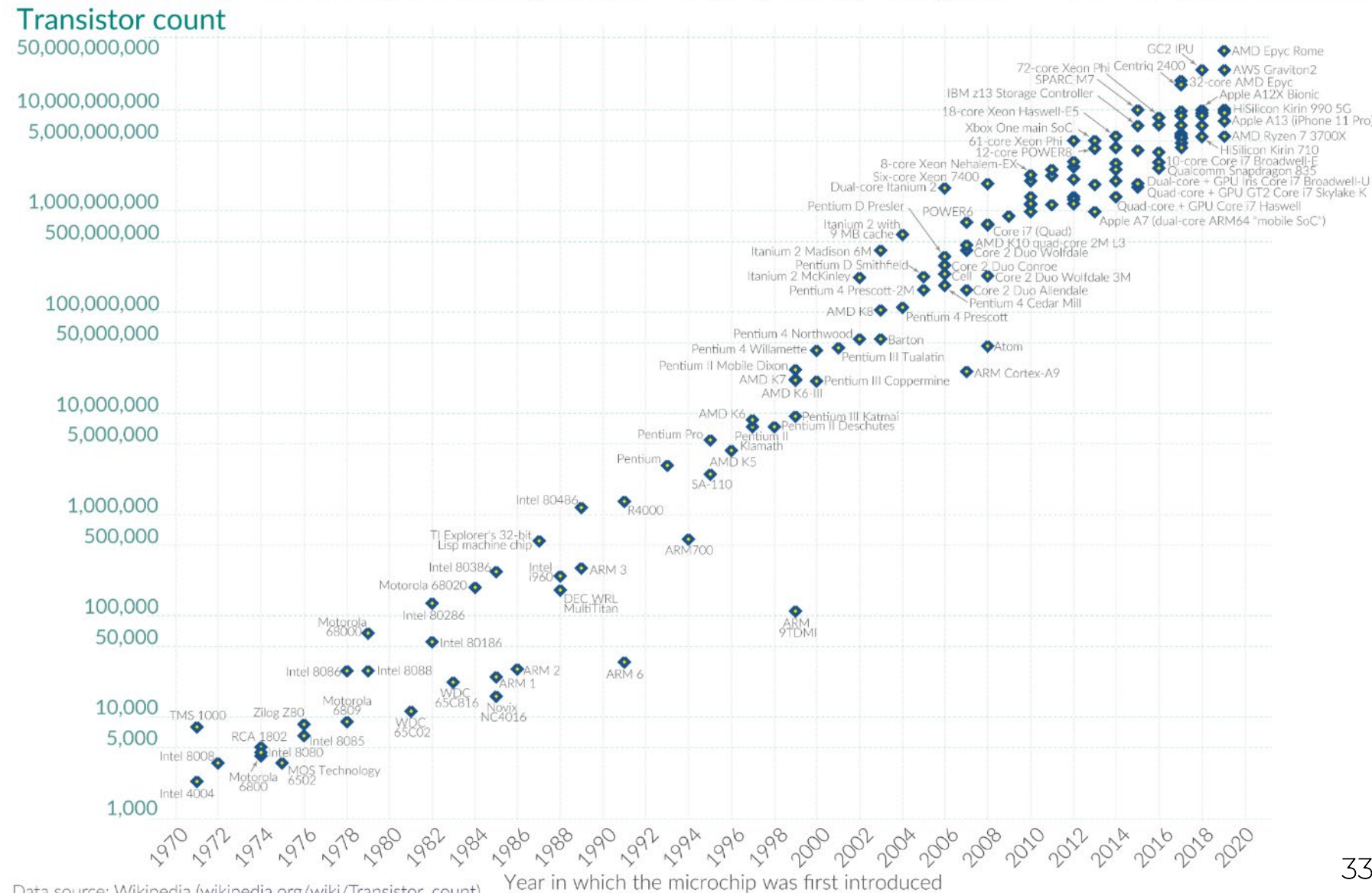
**8,000,000,000**  
Transistors



# Moore's Law

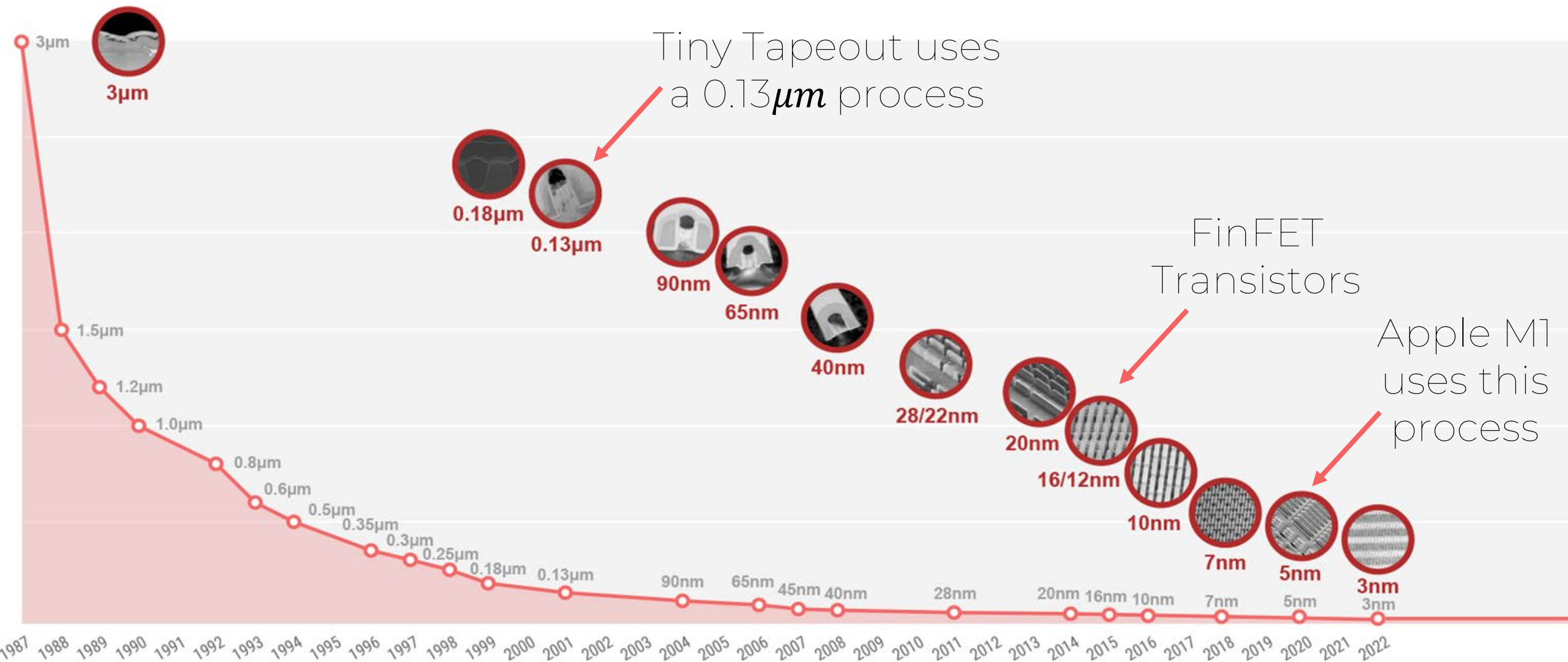
## Moore's Law: The number of transistors on microchips doubles every two years

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.



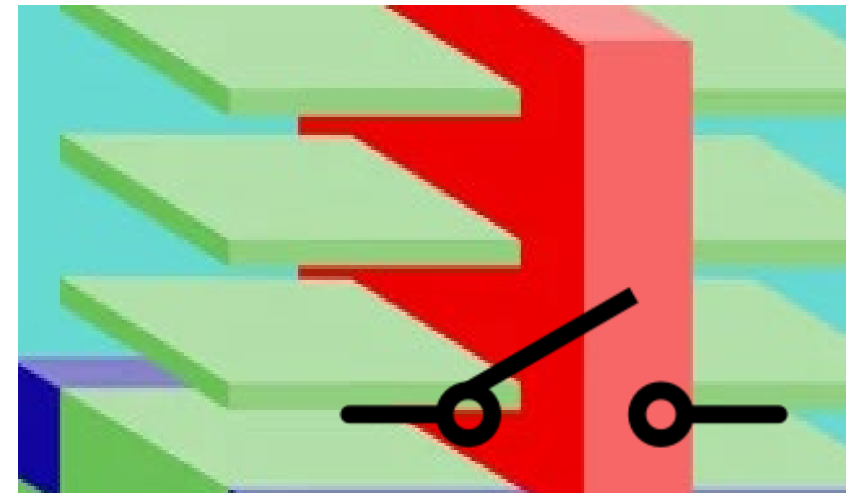
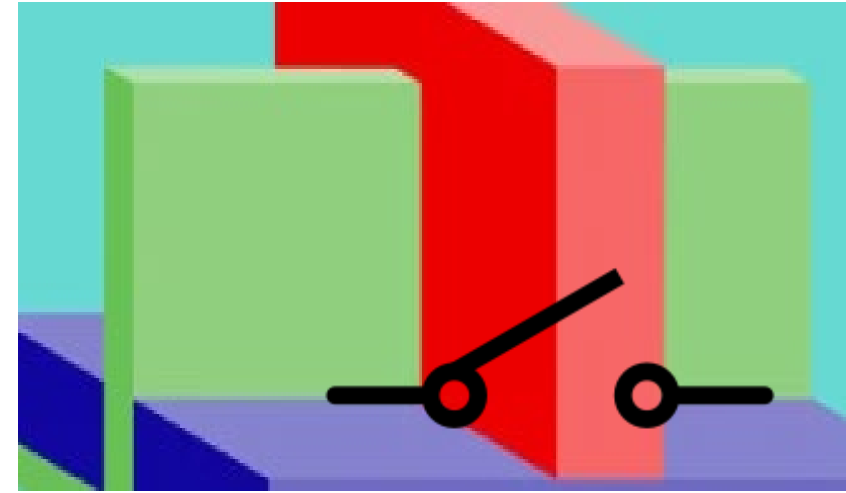
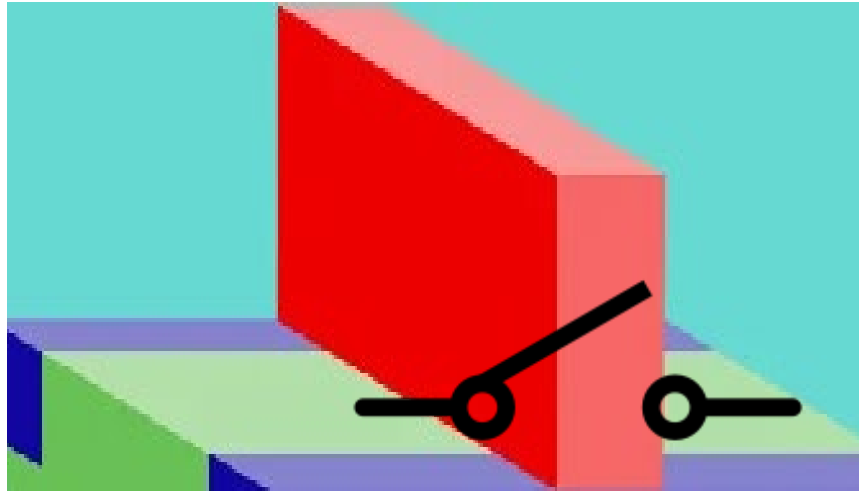
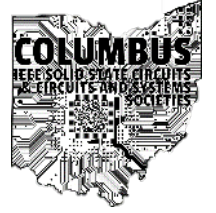
- Number of transistors doubles every 2 years
- Exponential growth!

# Moore's Law: TSMC Processes



[https://www.tsmc.com/english/dedicatedFoundry/technology/logic/l\\_5nm](https://www.tsmc.com/english/dedicatedFoundry/technology/logic/l_5nm)

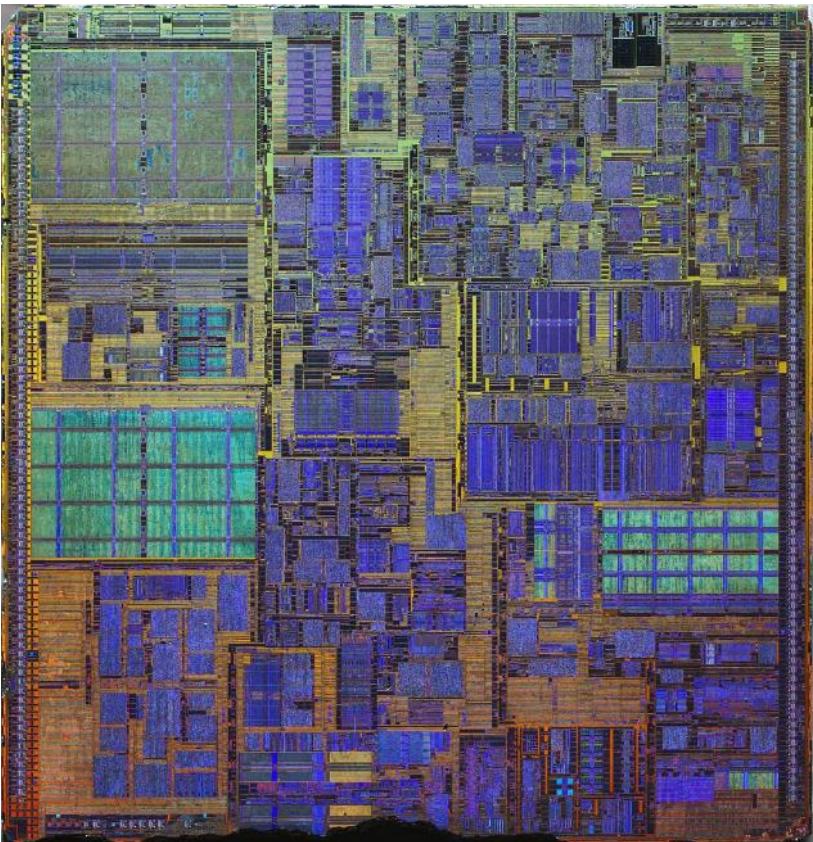
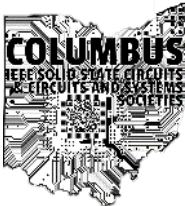
# Transistor Evolution



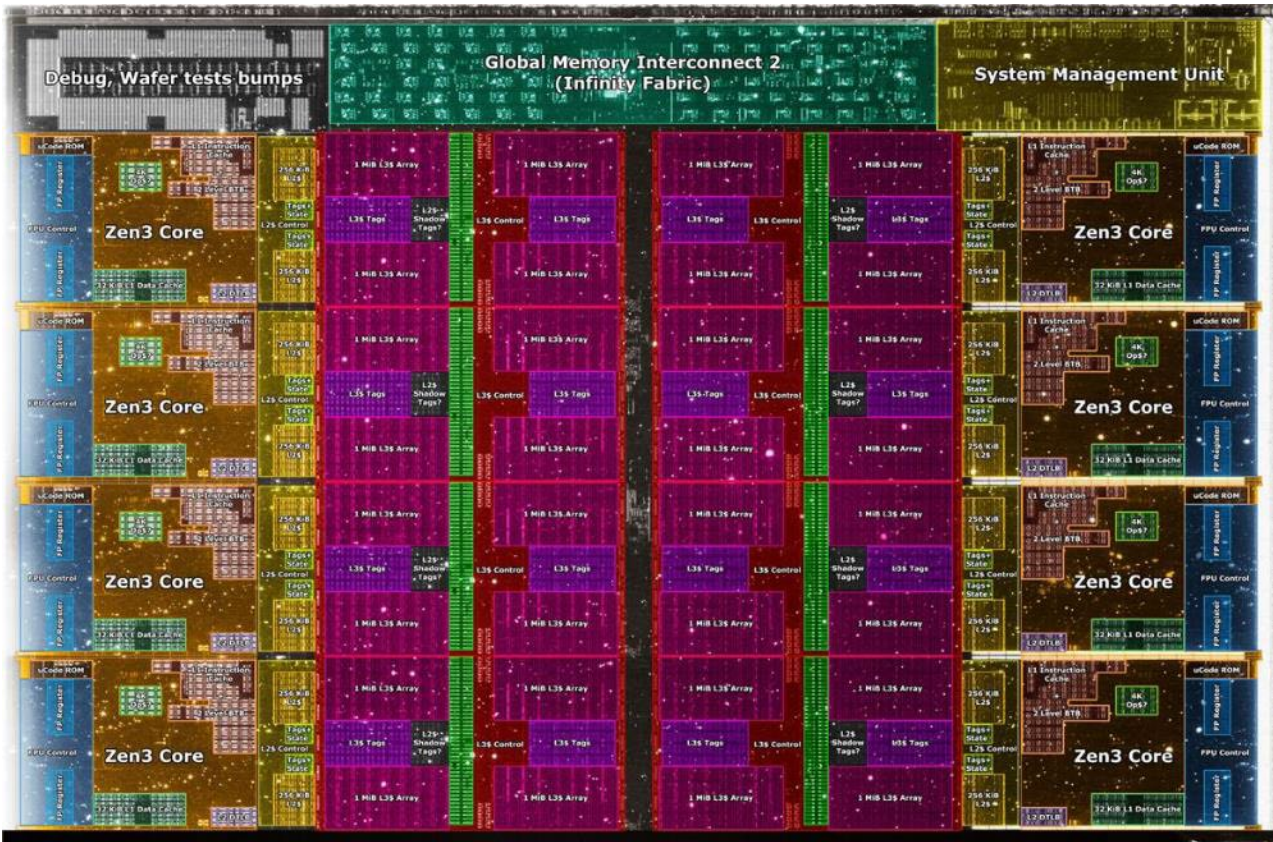
- Planar Transistors:  $> \sim 20\text{nm}$
- FinFET Transistors:  $< \sim 20\text{nm}$
- Gate-All-Around Transistors: Future
- [Video about FinFET Transistors](#)
- [Video about GAA Transistors](#)

<https://www.asml.com/en/news/stories/2022/what-is-a-gate-all-around-transistor>

# Power Density



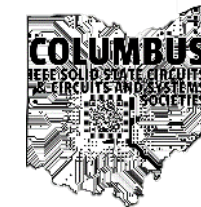
Pentium 4 (Northwood)  
55M 130nm Transistors  
~50W TDP



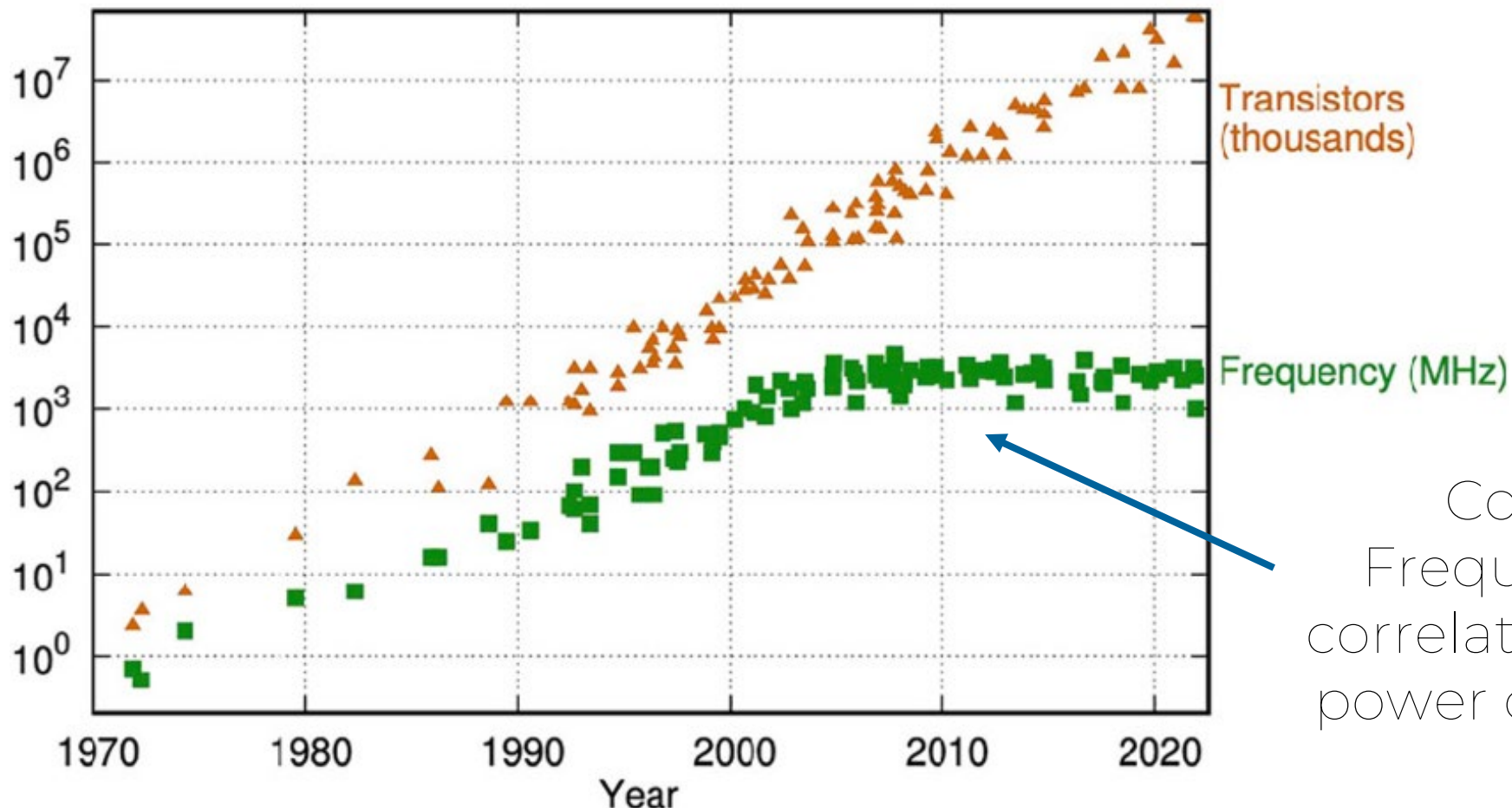
Ryzen 5800X  
4.15B 7nm Transistors  
~65W TDP

[https://hothardware.com/photo-gallery/newsitem/53364?image=big\\_amd\\_die\\_map.jpg&tag=popup](https://hothardware.com/photo-gallery/newsitem/53364?image=big_amd_die_map.jpg&tag=popup)

# Dennard Scaling



50 Years of Microprocessor Trend Data



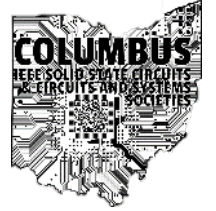
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten  
New plot and data collected for 2010-2021 by K. Rupp

Larus, James. (2023). Evolution of Computing. [10.1007/978-3-031-45304-5\\_3](https://doi.org/10.1007/978-3-031-45304-5_3).

# What if Moore's law/Dennard Scaling worked for cars?

1960	2020	Moore's car law
170 mph (75 m/s)	300 mph (134 m/s)	80,000 million m/s
14.3 mpg	38.8 mpg	15,000 million mpg

# Further Information



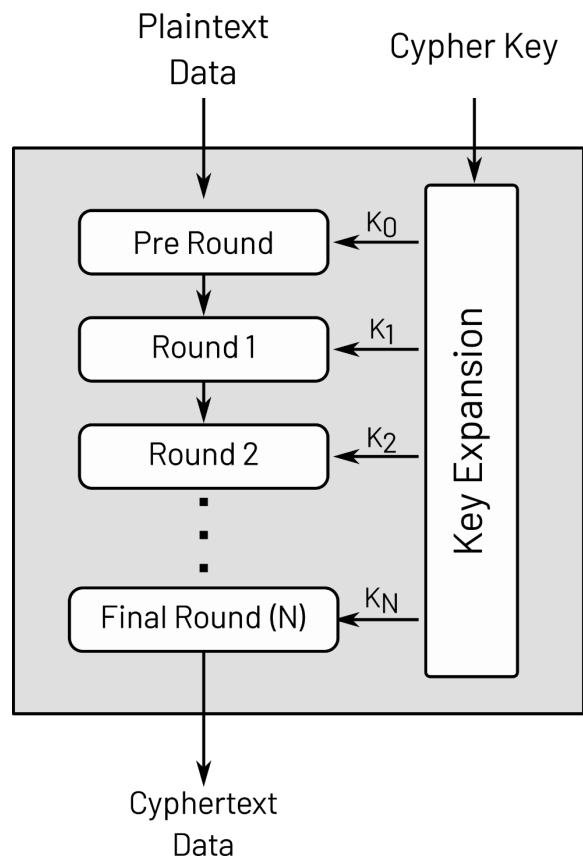
- This is just scratching the surface of the history here
- [Professor Marvin White's CASS Talk](#)
- [PBS Documentary on Silicon Valley](#)
- [Asianometry Youtube Channel](#)

# Introduction to Digital Design

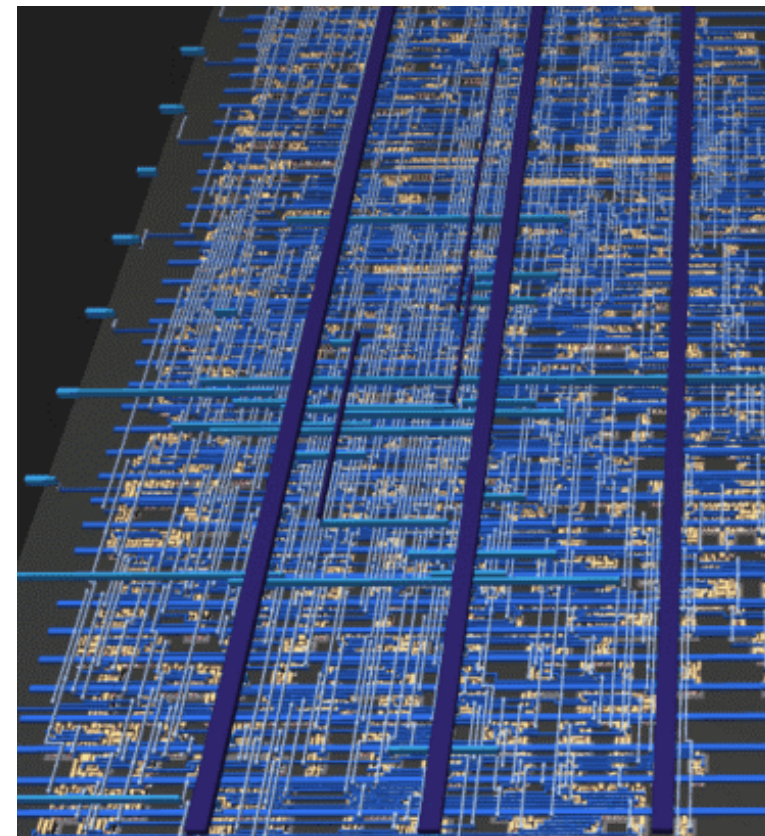


# What are we trying to do?

- Concept (algorithm, behavior, etc) → Physical design (gates)



Digital Design Flow  
(OpenLane)



# How does this work?

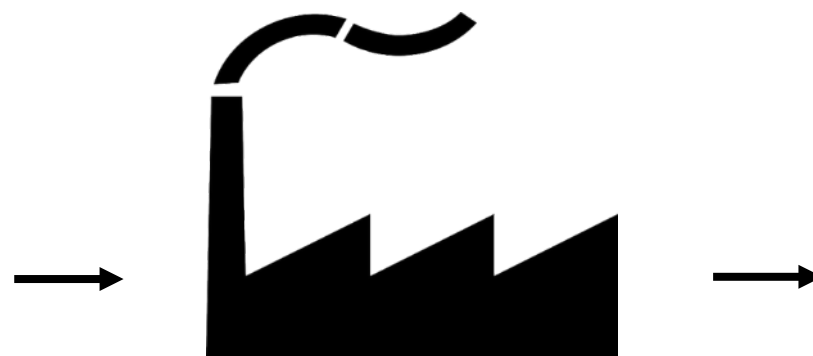
- Calculate Nth Fibonacci number
- Software (C program)

```

1  int fib(int n)
2  {
3      int a = 0, b = 1, c = 0, i;
4      if (n == 0) {
5          return a;
6      }
7      for (i = 2; i <= n; i++) {
8          c = a + b;
9          a = b;
10         b = c;
11     }
12     return b;
13 }

```

<https://godbolt.org/z/dPKEE7qE3>



↑  
Compiler (GCC)

```

1  fib:
2      beq    a0,zero,.L4
3      li    a5,1
4      ble    a0,a5,.L5
5      addi   a3,a0,1
6      li    a5,2
7      li    a0,1
8      li    a4,0
9
10     .L3:
11     mv     a2,a0
12     addi   a5,a5,1
13     add    a0,a0,a4
14     mv     a4,a2
15     bne    a5,a3,.L3
16     ret
17
18     .L4:
19     li    a0,0
20     ret
21
22     .L5:
23     li    a0,1
24     ret

```

# How does this work?

- Calculate Nth Fibonacci number
- Hardware (Digital Logic)

```

reg [WIDTH-1:0] iteration;
reg [WIDTH-1:0] prev;
reg [WIDTH-1:0] current;

assign o_busy = (iteration != RESET);
assign o_fib = current;

always @(posedge i_clk) begin
    if (!o_busy && i_stb) begin
        iteration <= i_n;
        prev [WIDTH-1:0] <= 1;
        current [WIDTH-1:0] <= 0;
    end
    else if (o_busy) begin
        iteration <= iteration - ONE;
        current <= prev + current;
        prev <= current;
    end
end

if (i_n == 0)
    iteration <= 0;
end

endmodule

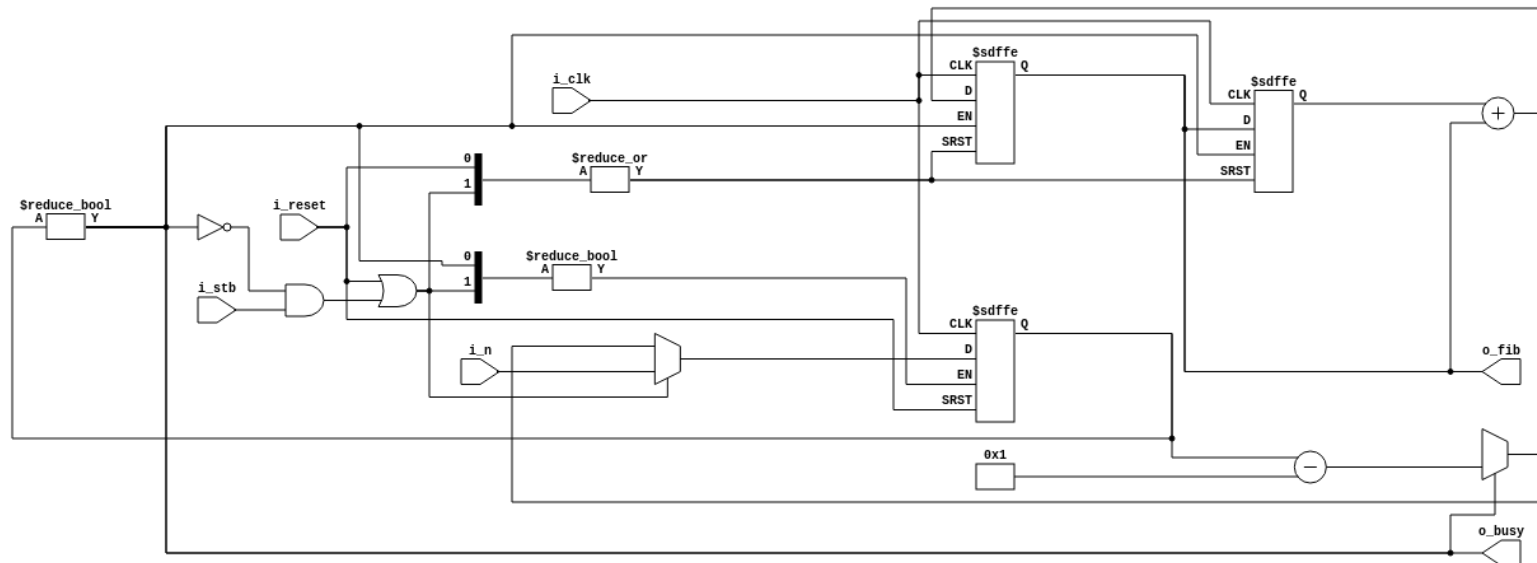
```

Hardware Description Language (Verilog)



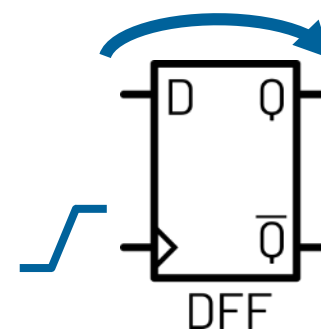
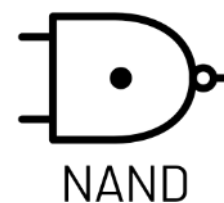
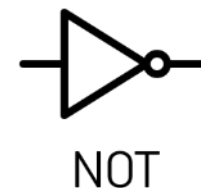
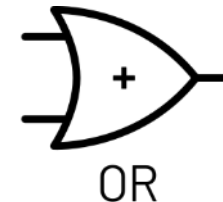
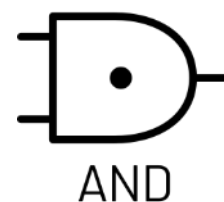
Synthesis Tool (Yosys)

Generic Logic Gates



# What is a "gate"?

- "Legos" of digital design
- Combinational Logic Gate
  - Performs digital logic function
  - AND, OR, NOT, ...
- Sequential Logic
  - Memory Element
  - Flip-Flops
  - Stores data for a "clock cycle"



# What is Verilog? (Wires and Registers)

- Hardware Description Language
  - Behaviorally\* describes the gates we want to generate
  - Combinational and sequential gates generated from code

```

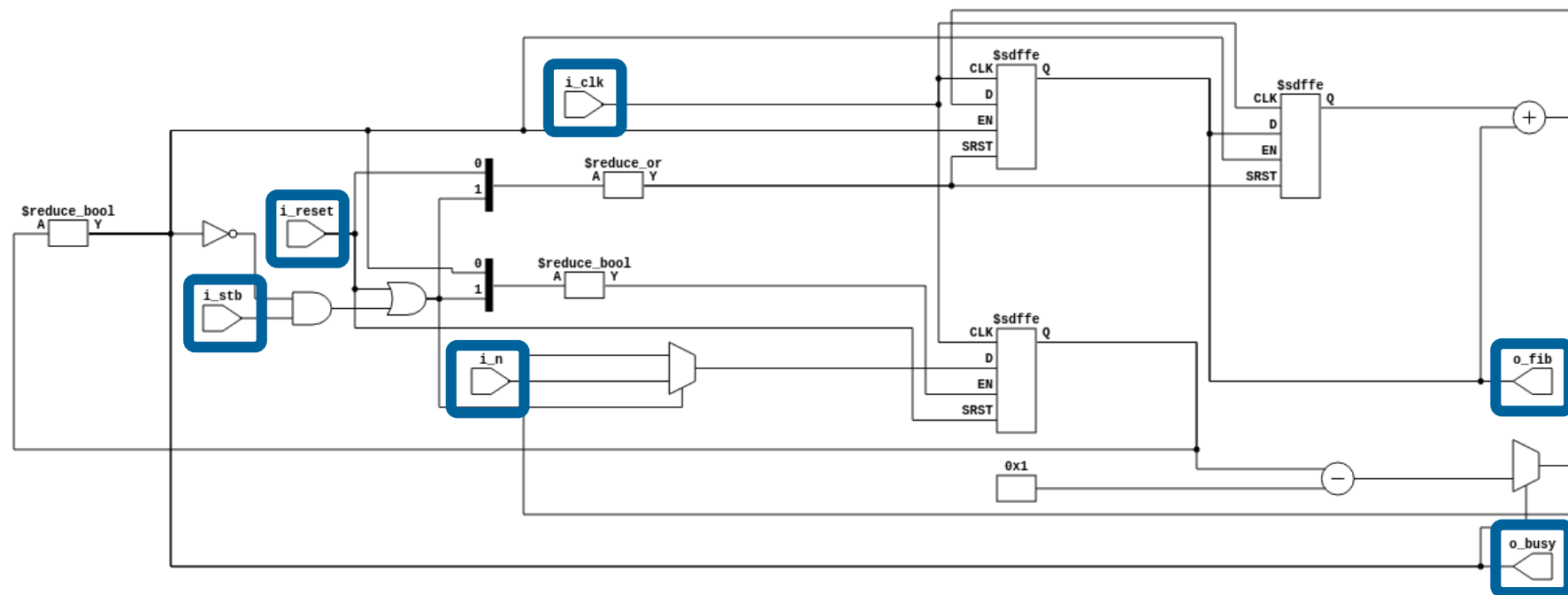
// global control signals
input wire i_reset;
input wire i_clk;

// control signals
input wire i_stb;
output wire o_busy;

// module io
input wire [WIDTH-1:0] i_n;
output wire [WIDTH-1:0] o_fib;

reg [WIDTH-1:0] iteration;
reg [WIDTH-1:0] prev;
reg [WIDTH-1:0] current;
    
```

Define Signals



\*There is also gate level verilog

# What is Verilog? (Wires and Registers)

- Hardware Description Language
  - Behaviorally\* describes the gates we want to generate
  - Combinational and sequential gates generated from code

```

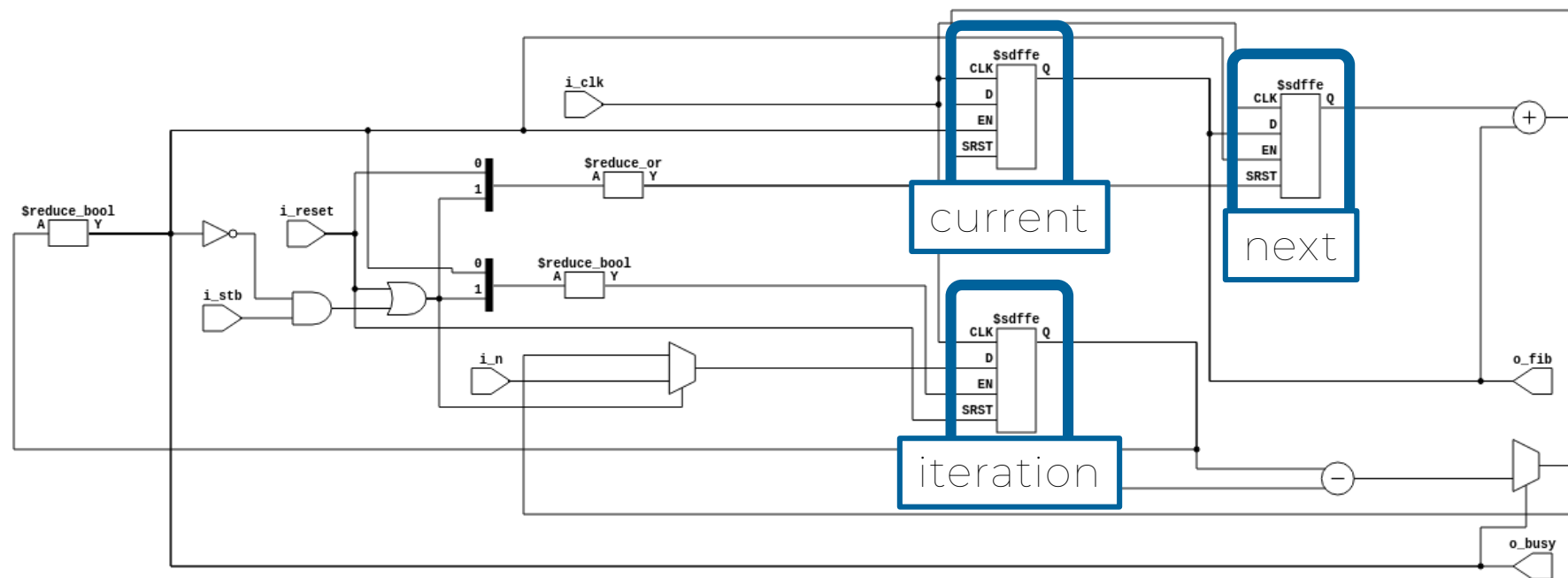
// global control signals
input wire i_reset;
input wire i_clk;

// control signals
input wire i_stb;
output wire o_busy;

// module io
input wire [WIDTH-1:0] i_n;
output wire [WIDTH-1:0] o_fib;

reg [WIDTH-1:0] iteration;
reg [WIDTH-1:0] prev;
reg [WIDTH-1:0] current;
    
```

Define Signals



\*There is also gate level verilog

# What is Verilog? (Assign Statements)

- Hardware Description Language
  - Behaviorally\* describes the gates we want to generate
  - Combinational and sequential gates generated from code

```
// global control signals
input wire i_reset;
input wire i_clk;

// control signals
input wire i_stb;
output wire o_busy;

// module io
input wire [WIDTH-1:0] i_n;
output wire [WIDTH-1:0] o_fib;

reg [WIDTH-1:0] iteration;
reg [WIDTH-1:0] prev;
reg [WIDTH-1:0] current;
```

Define Signals

- Generate combinational logic from operations
- Every assign statement evaluated “in parallel”
- Can only assign to wires



```
assign o_busy = (iteration != RESET);
assign o_fib = current;
```

Purely Combinational Logic

# What is Verilog? (Always Blocks)

- Hardware Description Language
  - Behaviorally\* describes the gates we want to generate
  - Combinational and sequential gates generated from code

```
// global control signals
input wire i_reset;
input wire i_clk;

// control signals
input wire i_stb;
output wire o_busy;

// module io
input wire [WIDTH-1:0] i_n;
output wire [WIDTH-1:0] o_fib;

reg [WIDTH-1:0] iteration;
reg [WIDTH-1:0] prev;
reg [WIDTH-1:0] current;
```

Define Signals

```
assign o_busy = (iteration != RESET);
assign o_fib = current;
```

Purely Combinational Logic

```
always @(posedge i_clk) begin
    if (!o_busy && i_stb) begin
        iteration <= i_n;
        prev [WIDTH-1:0] <= 1;
        current [WIDTH-1:0] <= 0;
    end
    else if (o_busy) begin
        iteration <= iteration - ONE;
        current <= prev + current;
        prev <= current;
    end

    if (i_reset) begin
        iteration <= RESET;
        prev [WIDTH-1:0] <= 1;
        current [WIDTH-1:0] <= 0;
    end
end
```

Sequential Logic Behavior

- Define register behavior
  - Write to registers in “always” block
- Defines what the “next state” of the register should be
- Only evaluated on rising edges
- Multiple Always blocks are executed in parallel



# What is Verilog? (Always Blocks)

- Hardware Description Language
  - Behaviorally\* describes the gates we want to generate
  - Combinational and sequential gates generated from code

```
// global control signals
input wire i_reset;
input wire i_clk;

// control signals
input wire i_stb;
output wire o_busy;

// module io
input wire [WIDTH-1:0] i_n;
output wire [WIDTH-1:0] o_fib;

reg [WIDTH-1:0] iteration;
reg [WIDTH-1:0] prev;
reg [WIDTH-1:0] current;
```

Define Signals

```
assign o_busy = (iteration != RESET);
assign o_fib = current;
```

Purely Combinational Logic

```
always @(posedge i_clk) begin
    if (!o_busy && i_stb) begin
        current [WIDTH-1:0] <= 0;
    end
    else if (o_busy) begin
        iteration = iteration - ONE;
        current = prev + current;
        prev = current;
    end

    if (i_reset) begin
        iteration = RESET;
        current [WIDTH-1:0] <= 0;
    end
end
```

Sequential Logic Behavior

- Always blocks evaluation
  - Top to bottom
  - $\leq$  assignments happen in parallel
- Statements lower in block have higher priority
- Can feel like “normal” programming
  - *Don't get complacent*

# What is Verilog? (Always Blocks)

- Hardware Description Language
  - Behaviorally\* describes the gates we want to generate
  - Combinational and sequential gates generated from code

```
// global control signals
input wire i_reset;
input wire i_clk;

// control signals
input wire i_stb;
output wire o_busy;

// module io
input wire [WIDTH-1:0] i_n;
output wire [WIDTH-1:0] o_fib;

reg [WIDTH-1:0] iteration;
reg [WIDTH-1:0] prev;
reg [WIDTH-1:0] current;
```

Define Signals

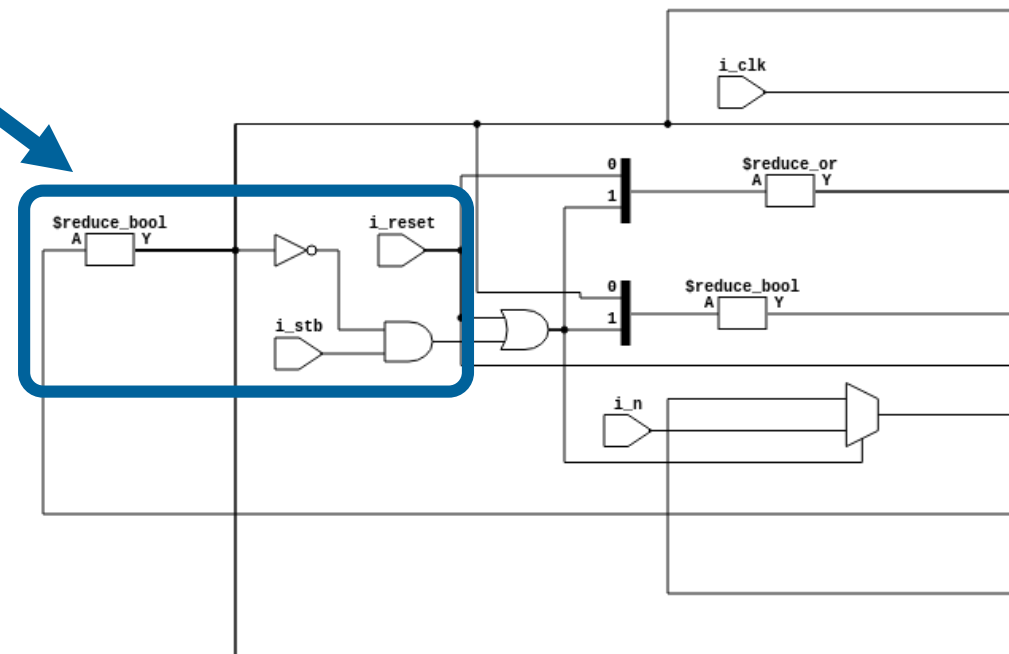
```
assign o_busy = (iteration != RESET);
assign o_fib = current;
```

Purely Combinational Logic

```
always @(posedge i_clk) begin
    if (!o_busy && i_stb) begin
        iteration <= i_n;
        prev [WIDTH-1:0] <= ...;
        current [WIDTH-1:0] <= 0;
    end
    else if (o_busy) begin
        iteration <= iteration - ONE;
        current <= prev + current;
        prev <= current;
    end
    if (i_reset) begin
        iteration <= RESET;
        prev [WIDTH-1:0] <= 1;
        current [WIDTH-1:0] <= 0;
    end
end
```

Sequential Logic Behavior

- Always blocks may also contain logic



# What is Verilog? (Always Blocks)

- Hardware Description Language
  - Behaviorally\* describes the gates we want to generate
  - Combinational and sequential gates generated from code

```
// global control signals
input wire i_reset;
input wire i_clk;

// control signals
input wire i_stb;
output wire o_busy;

// module io
input wire [WIDTH-1:0] i_n;
output wire [WIDTH-1:0] o_fib;

reg [WIDTH-1:0] iteration;
reg [WIDTH-1:0] prev;
reg [WIDTH-1:0] current;
```

Define Signals

```
assign o_busy = (iteration != RESET);
assign o_fib = current;
```

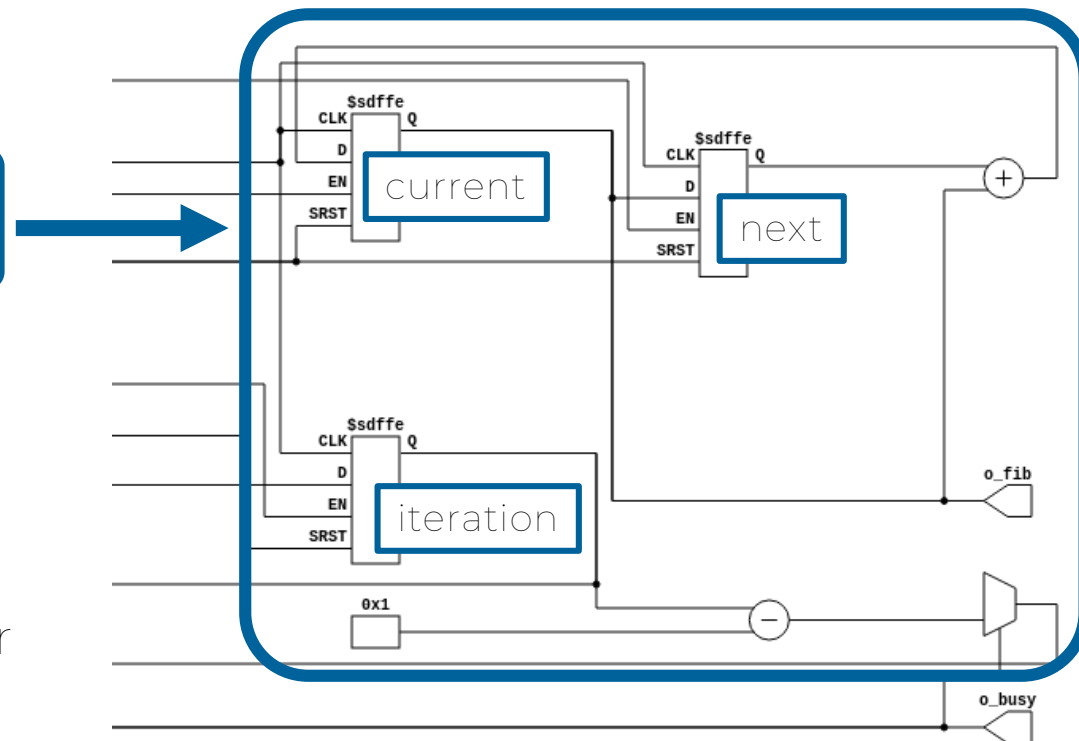
Purely Combinational Logic

```
always @(posedge i_clk) begin
    if (!o_busy && i_stb) begin
        iteration <= i_n;
        prev [WIDTH-1:0] <= 1;
        current [WIDTH-1:0] <= 0;
    end
    else if (o_busy) begin
        iteration <= iteration - ONE;
        current <= prev + current;
        prev <= current;
    end
end

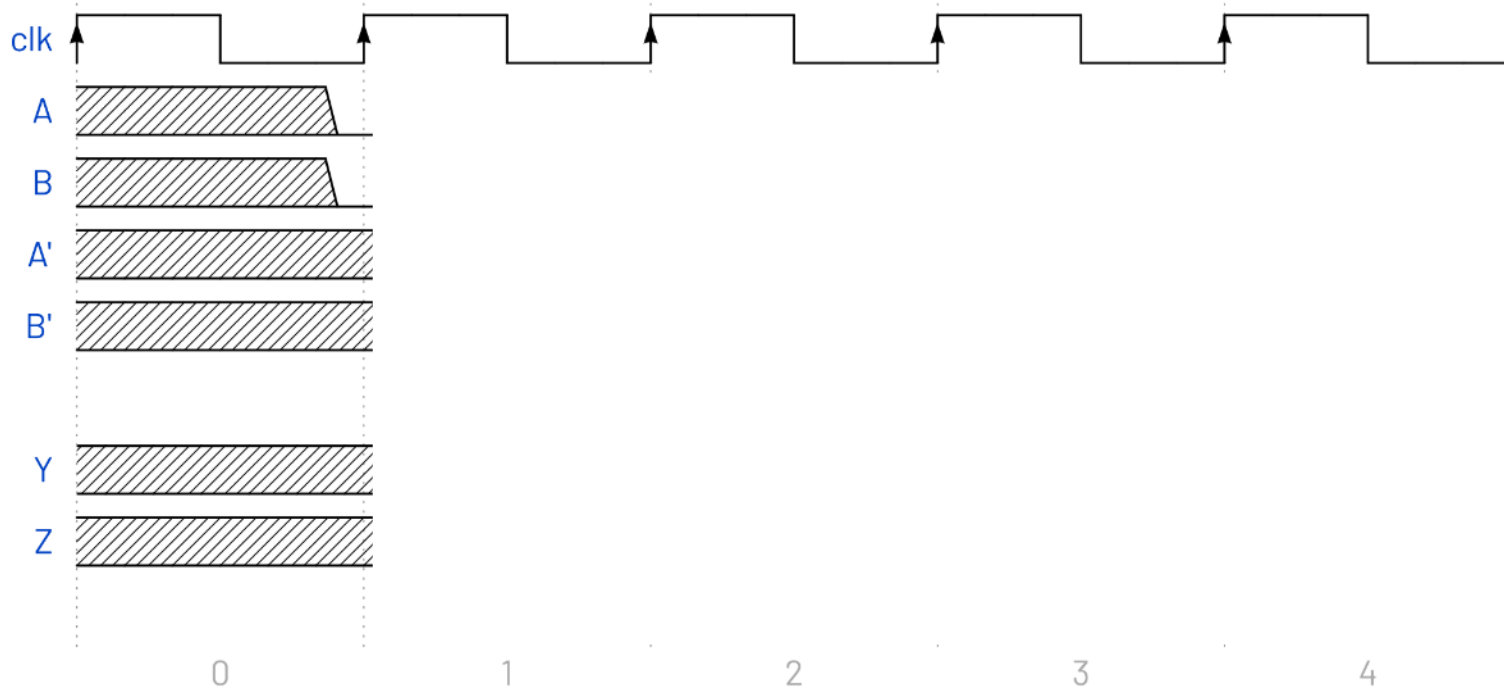
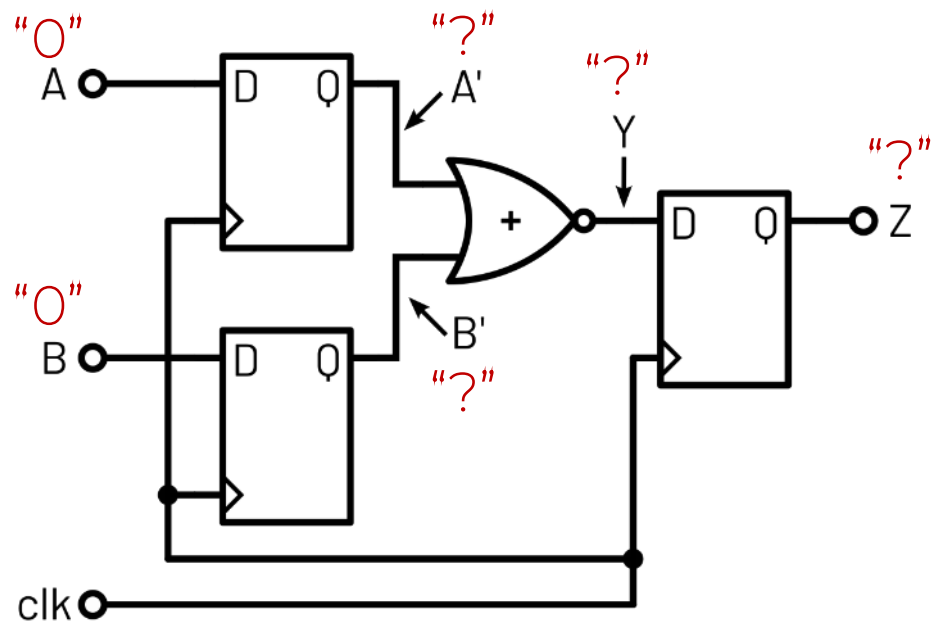
if (i_reset) begin
    iteration <= RESET;
    prev [WIDTH-1:0] <= 1;
    current [WIDTH-1:0] <= 0;
end
end
```

Sequential Logic Behavior

- Always blocks may also contain logic



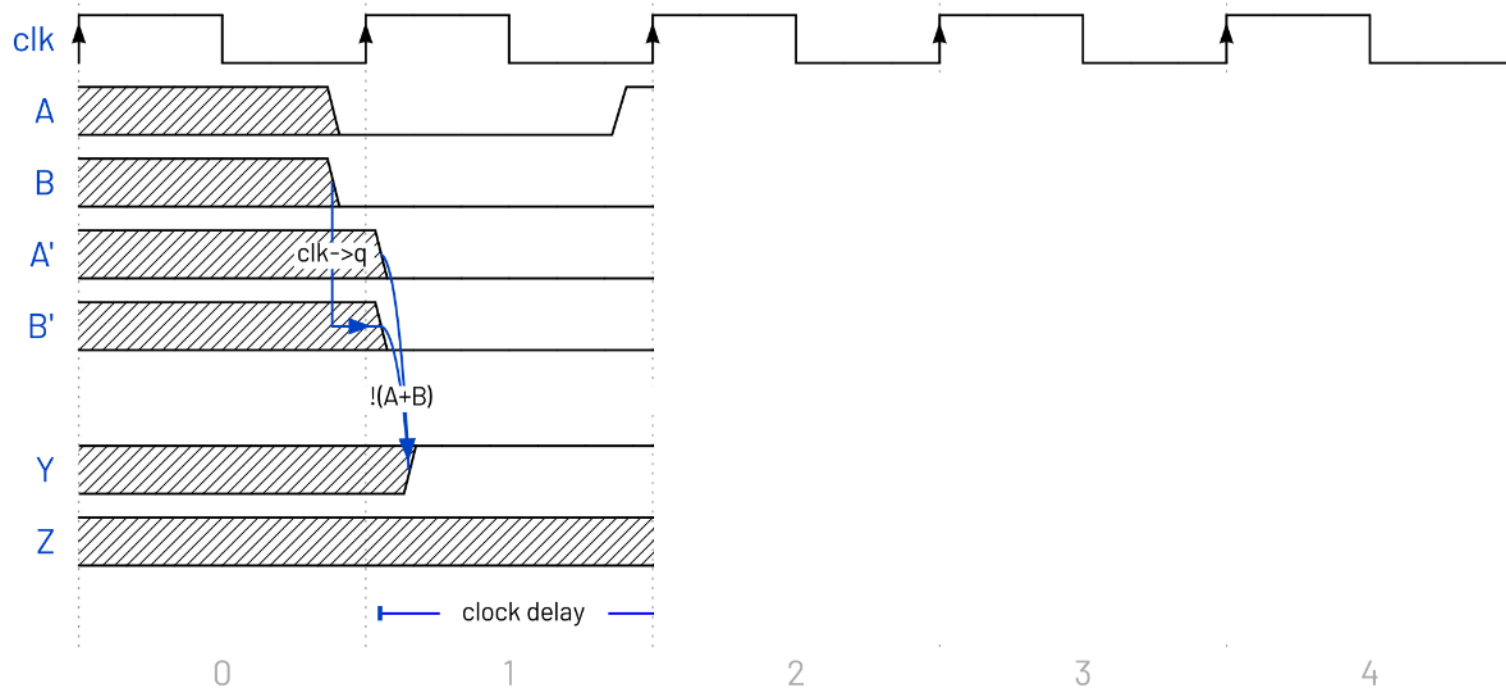
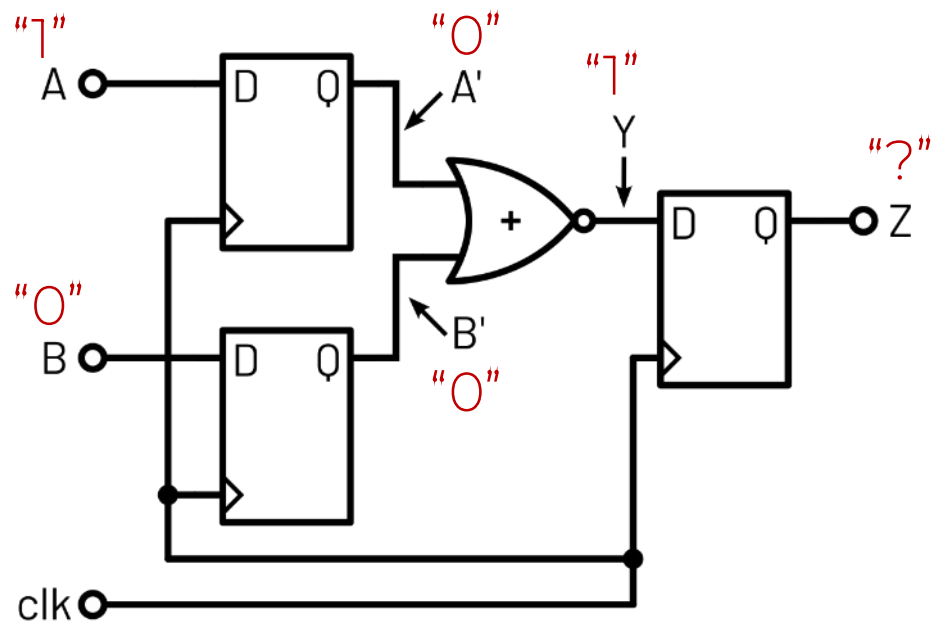
# A simple example



NOR Example Waveforms

- How does the system evaluate over time?
  - What is the state of A', B', Y, and Z?

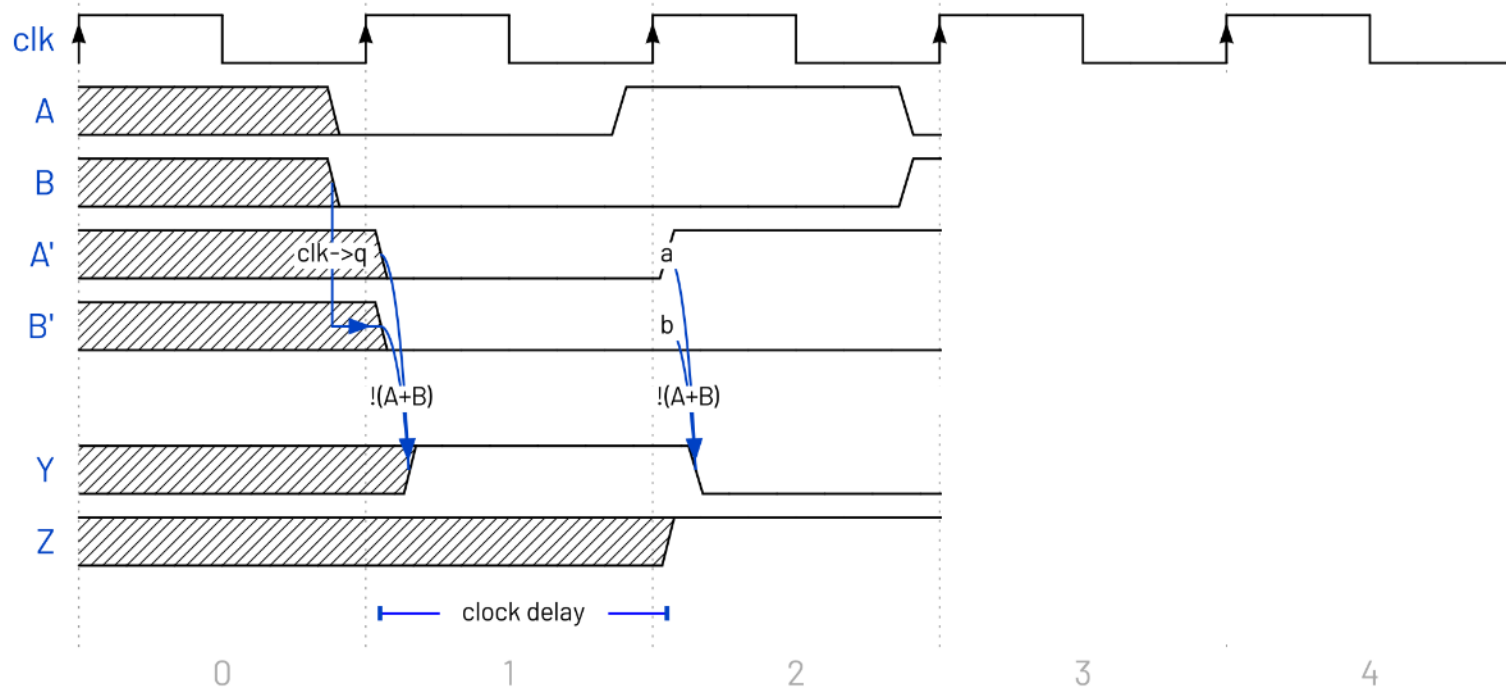
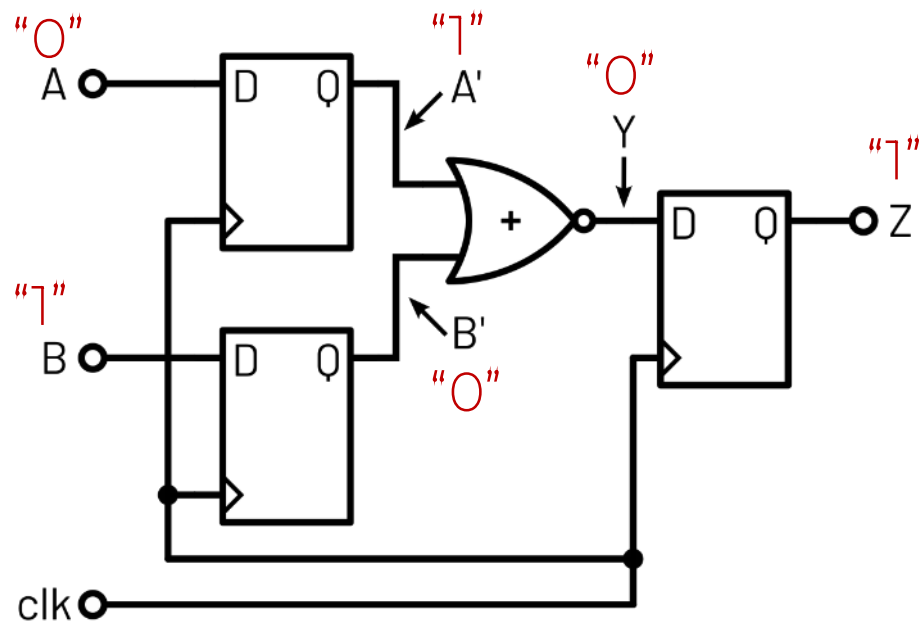
# A simple example



NOR Example Waveforms

- How does the system evaluate over time?
  - What is the next state of A', B', Y, and Z?

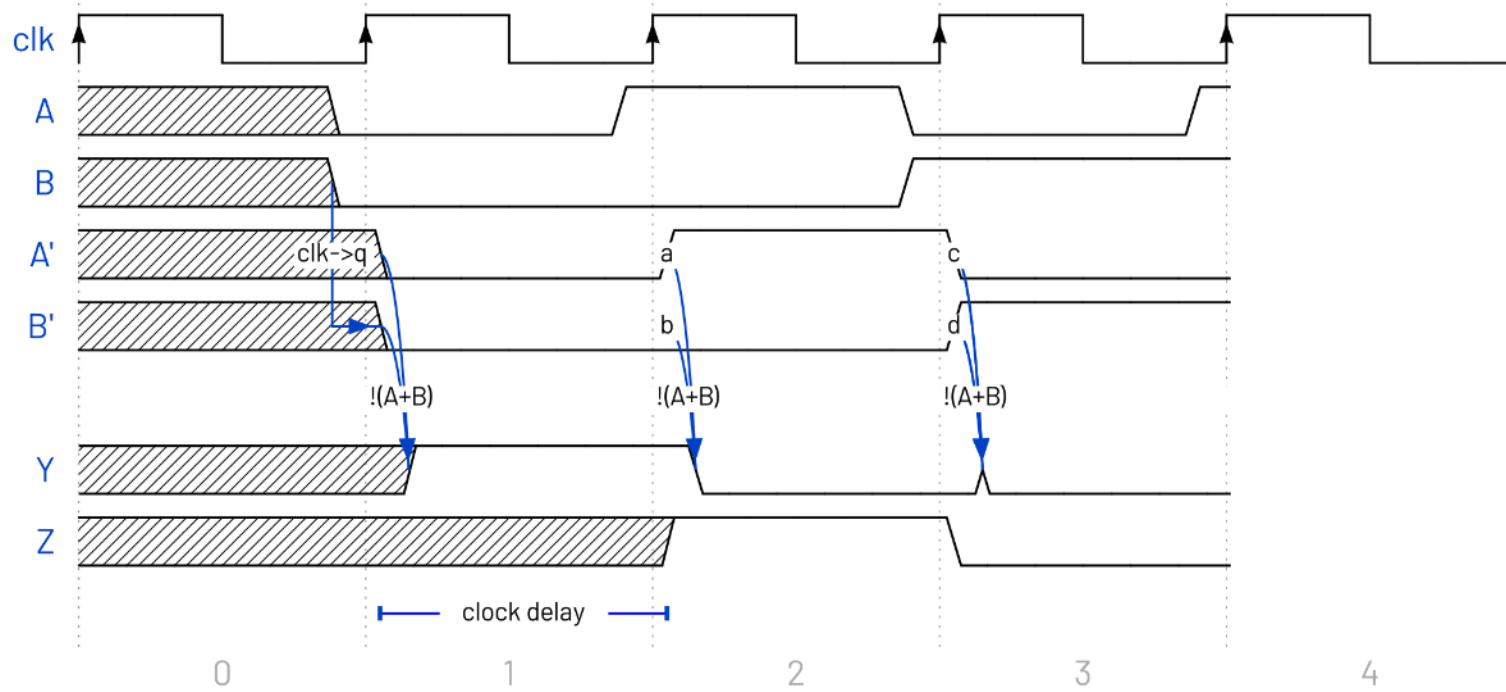
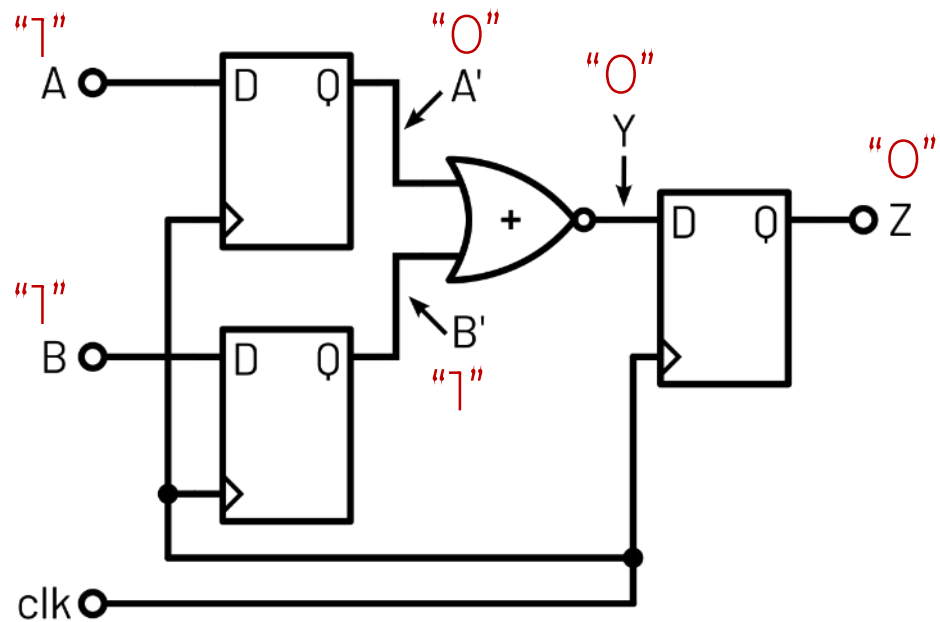
# A simple example



NOR Example Waveforms

- How does the system evaluate over time?
  - What is the next state of A', B', Y, and Z?

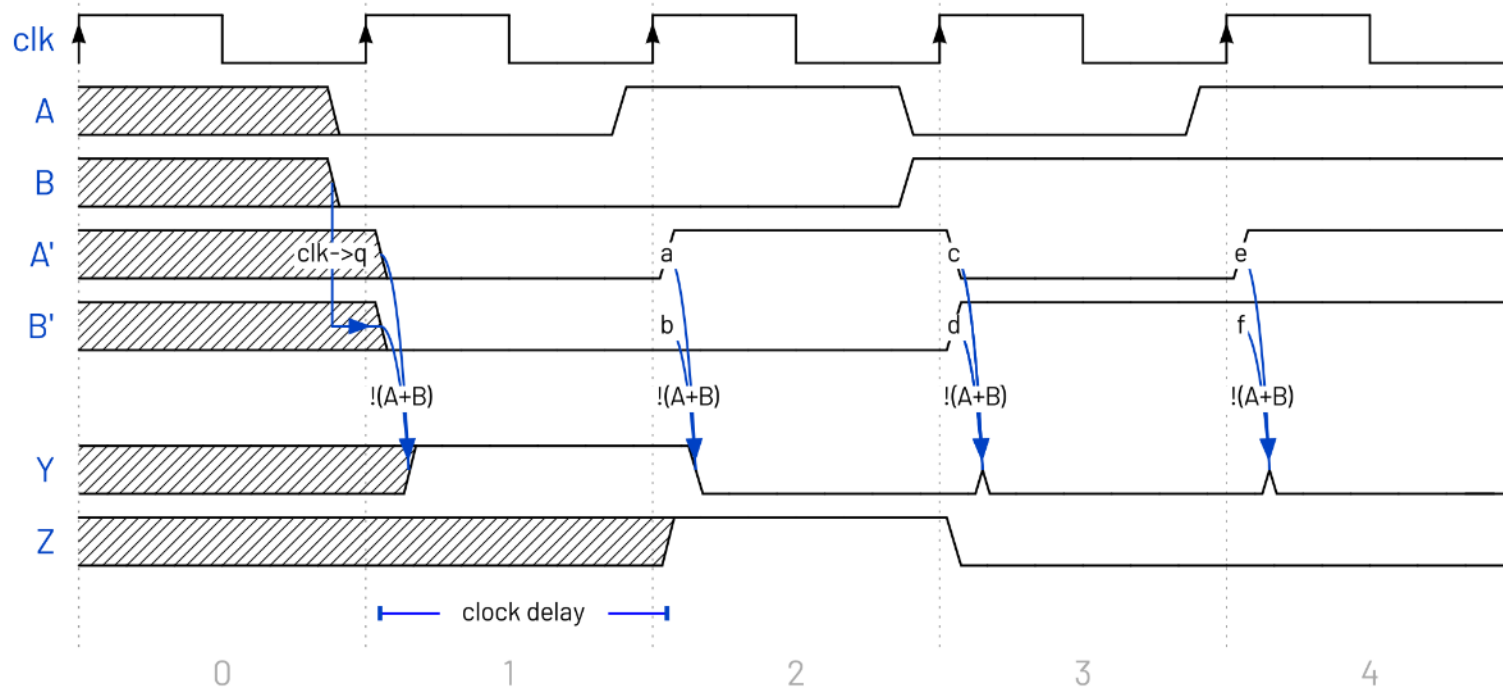
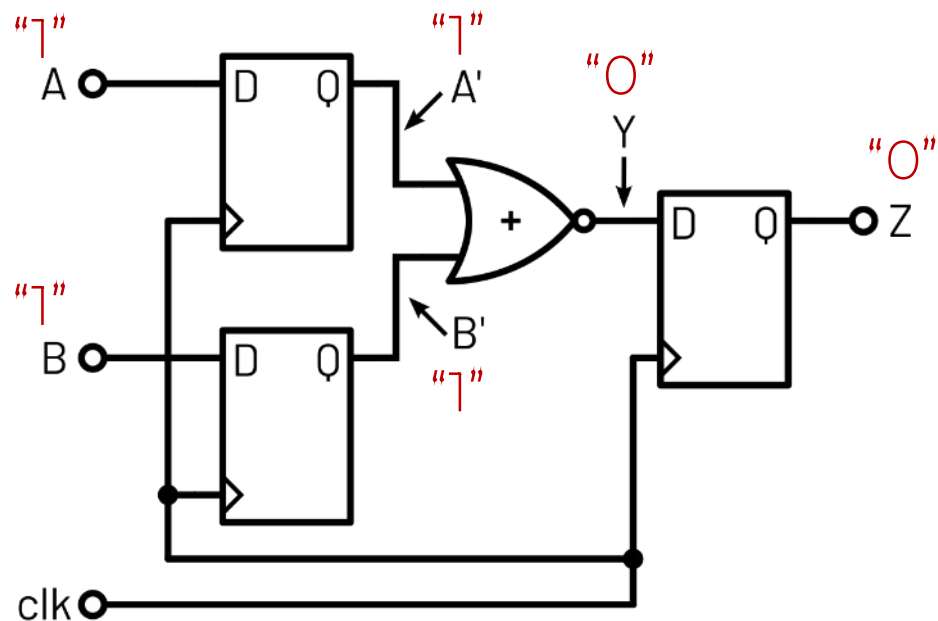
# A simple example



NOR Example Waveforms

- How does the system evaluate over time?
  - What is the next state of A', B', Y, and Z?

# A simple example



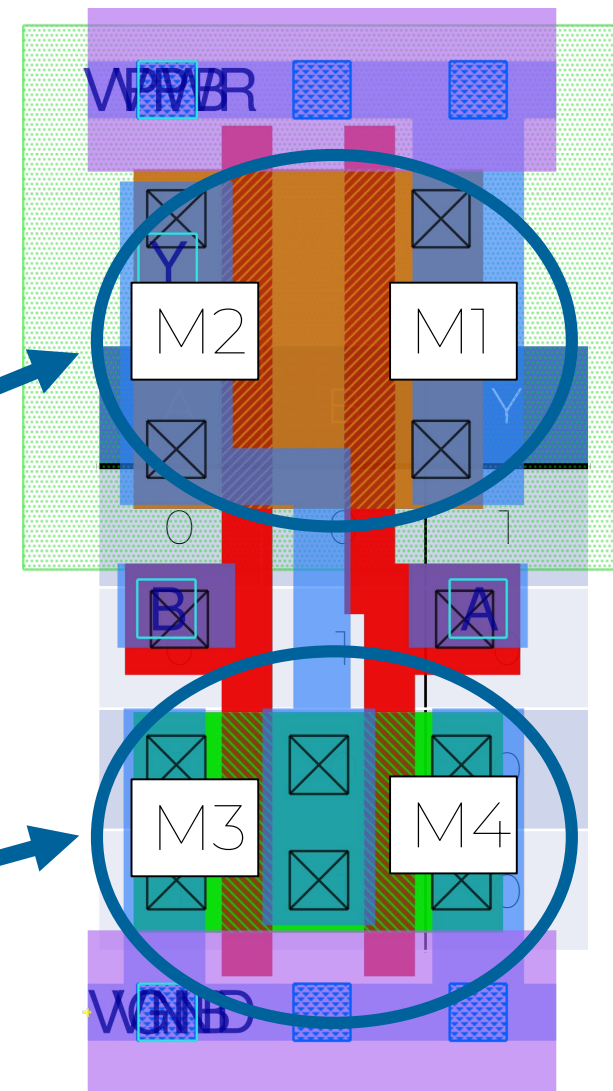
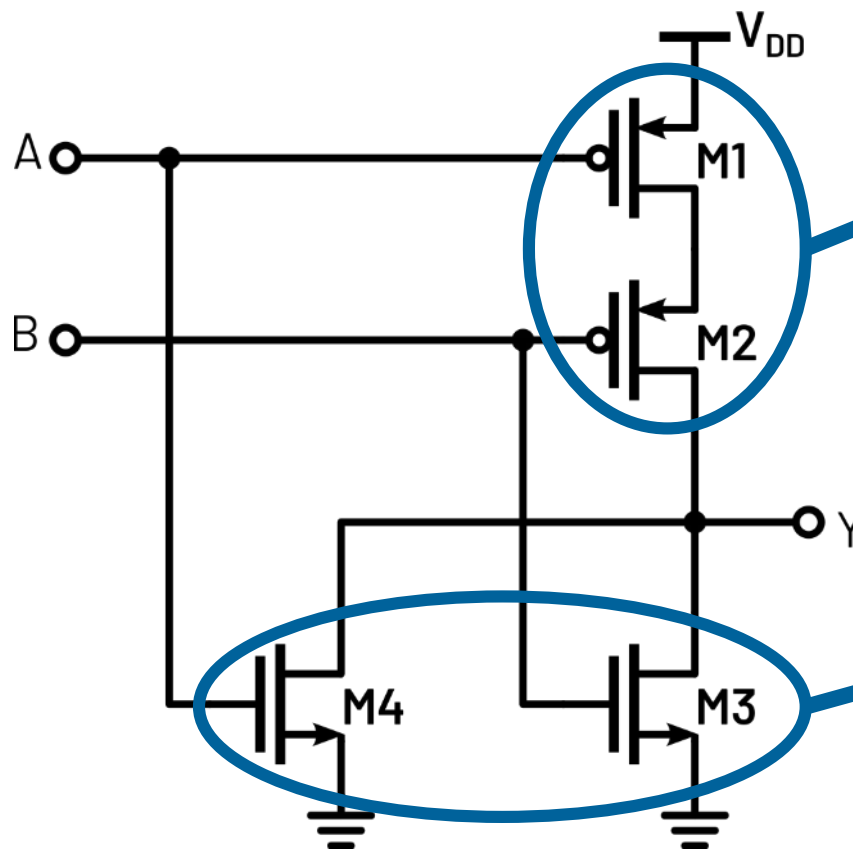
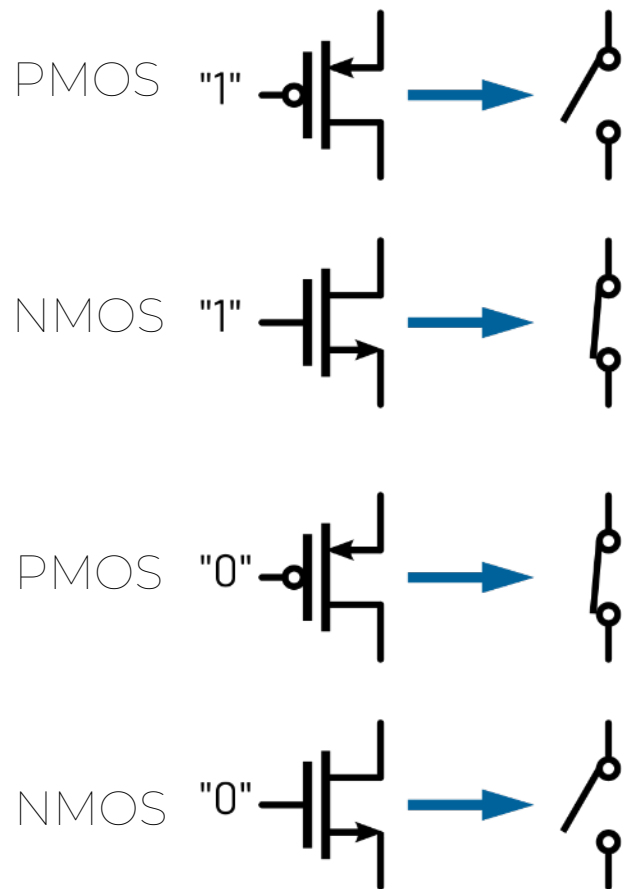
NOR Example Waveforms

- How does the system evaluate over time?
  - What is the next state of A', B', Y, and Z?



# Real Logic Gates

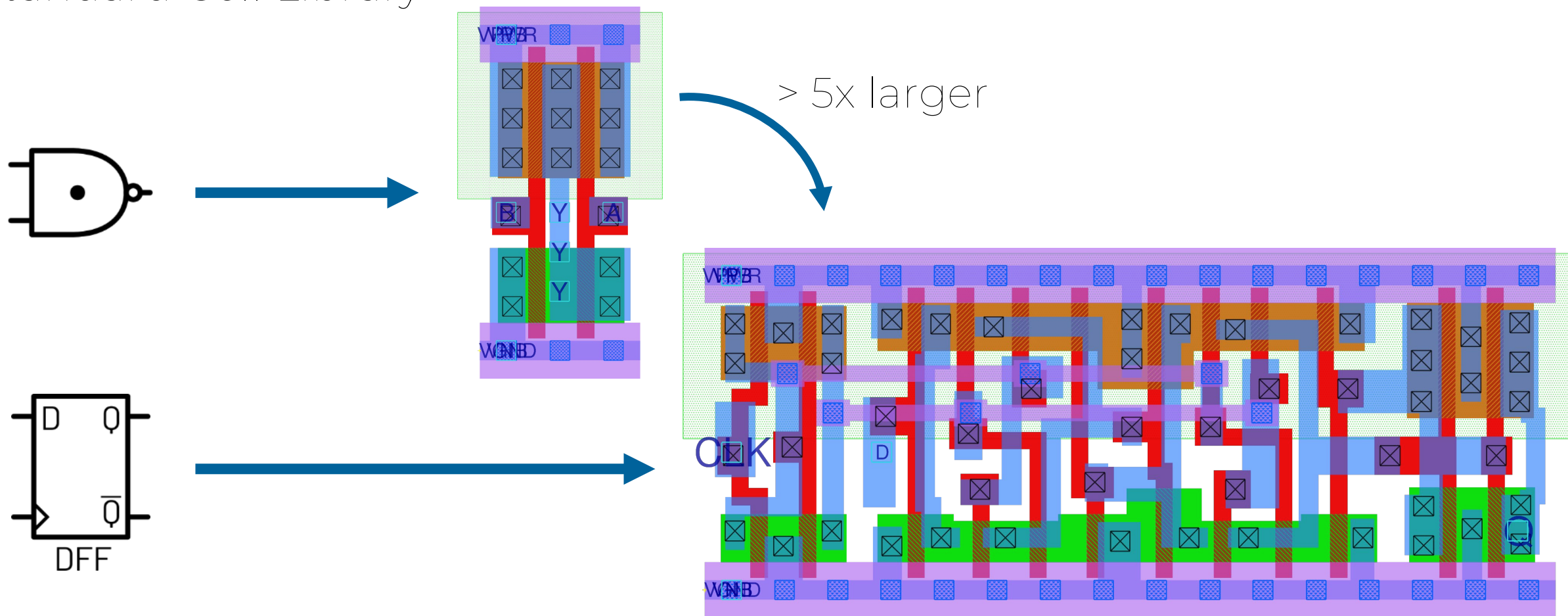
- How is a NOR gate built?
- Transistors → Switches



[https://skywater-pdk.readthedocs.io/en/main/contents/libraries/sky130\\_fd\\_sc\\_hd/cells/nor2/README.html#sky130-fd-sc-hd-nor2-gdsii-layouts](https://skywater-pdk.readthedocs.io/en/main/contents/libraries/sky130_fd_sc_hd/cells/nor2/README.html#sky130-fd-sc-hd-nor2-gdsii-layouts)

# Standard Cell Library

- Skywater provides designs for a large number of logic gates
- Standard Cell Library

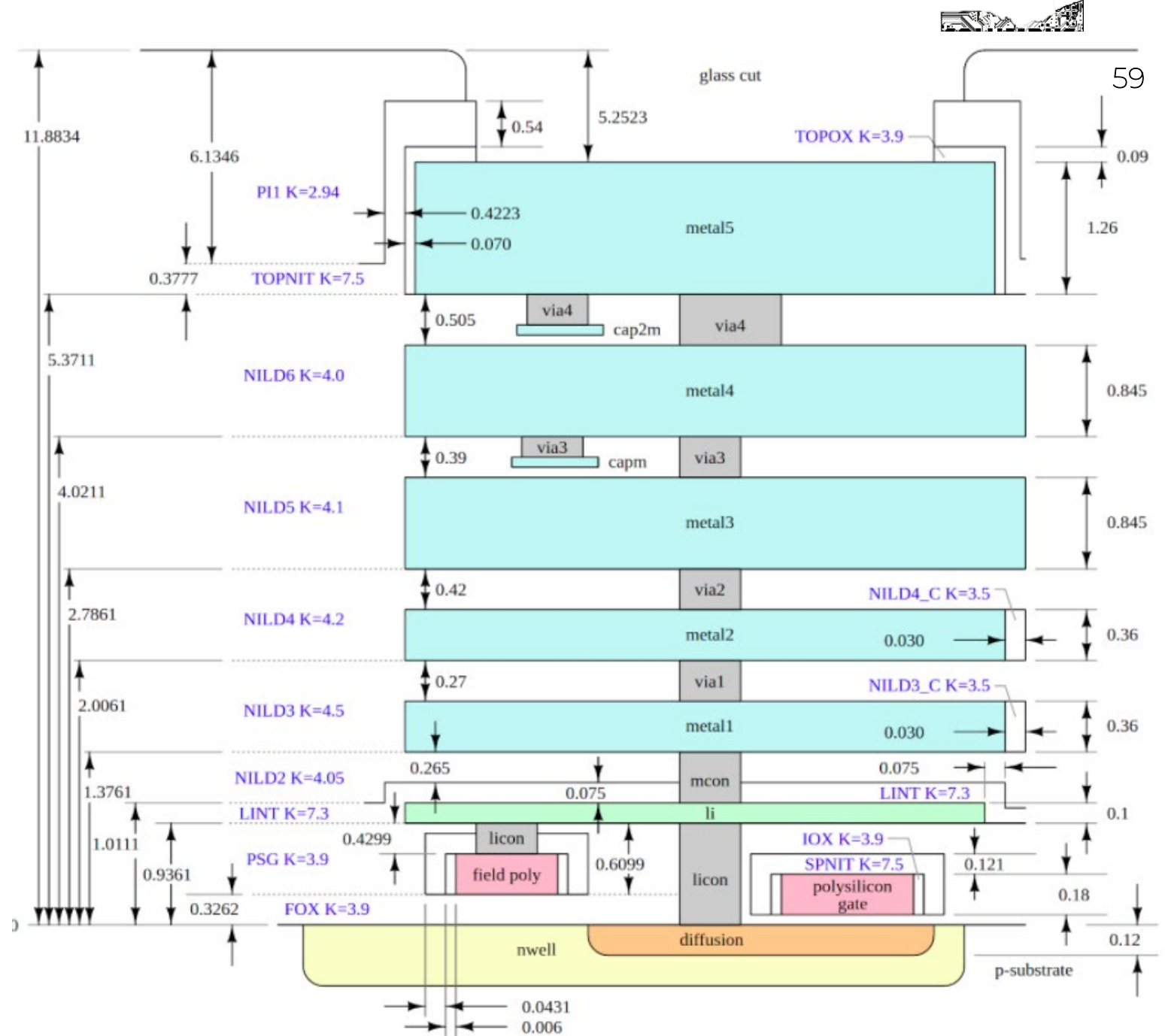


[https://skywater-pdk.readthedocs.io/en/main/contents/libraries/sky130\\_fd\\_sc\\_hd/cells/nand2/README.html#sky130-fd-sc-hd-nand2-gdsii-layouts](https://skywater-pdk.readthedocs.io/en/main/contents/libraries/sky130_fd_sc_hd/cells/nand2/README.html#sky130-fd-sc-hd-nand2-gdsii-layouts)

[https://skywater-pdk.readthedocs.io/en/main/contents/libraries/sky130\\_fd\\_sc\\_hd/cells/dfxtp/README.html](https://skywater-pdk.readthedocs.io/en/main/contents/libraries/sky130_fd_sc_hd/cells/dfxtp/README.html)

# Sky130 Stackup

- World's first open source manufacturable PDK
- Profile of layers in design
- Gates are connected on metal2 – metal5



# What's the difference between Software and Hardware?

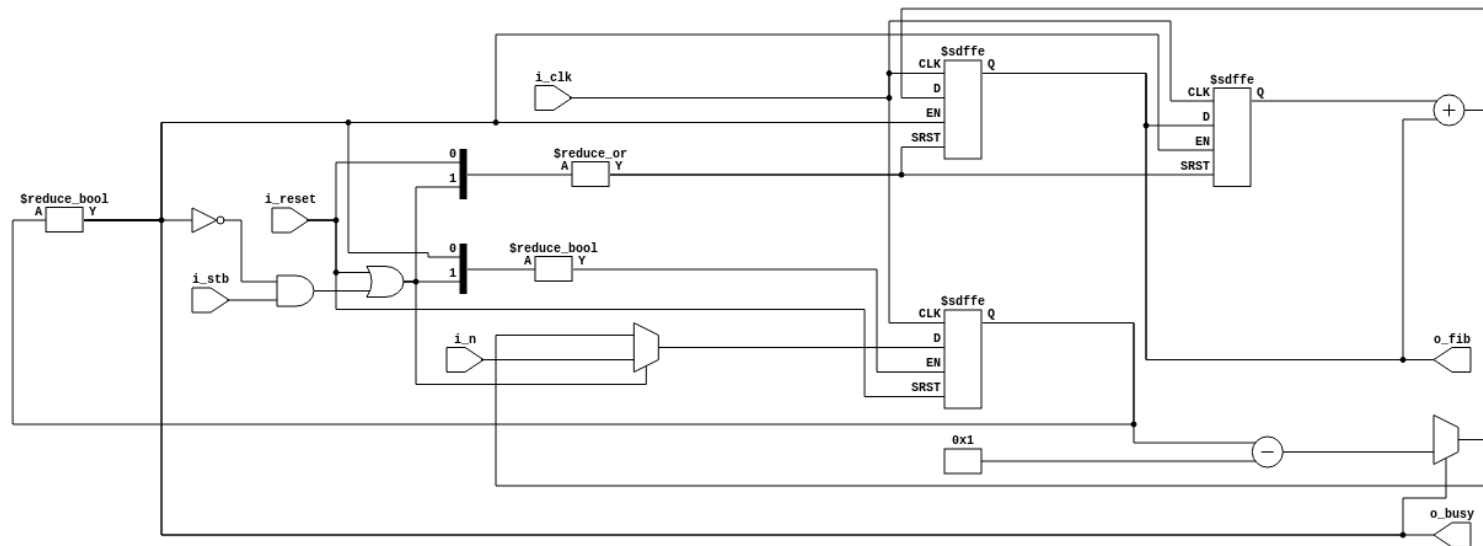
Software

```

1 fib:
2     beq    a0,zero,.L4
3     li     a5,1
4     ble   a0,a5,.L5
5     addi  a3,a0,1
6     li     a5,2
7     li     a0,1
8     li     a4,0
9 .L3:
10    mv     a2,a0
11    addi  a5,a5,1
12    add   a0,a0,a4
13    mv     a4,a2
14    bne   a5,a3,.L3
15    ret
16 .L4:
17    li     a0,0
18    ret
19 .L5:
20    li     a0,1
21    ret
    
```



Hardware



- Software executes sequentially
- Hardware is “always running”
  - Copies run in parallel

# Recap: Digital Design

- Concept (algorithm, behavior, etc) → Physical design (gates)

```

reg [WIDTH-1:0] iteration;
reg [WIDTH-1:0] prev;
reg [WIDTH-1:0] current;

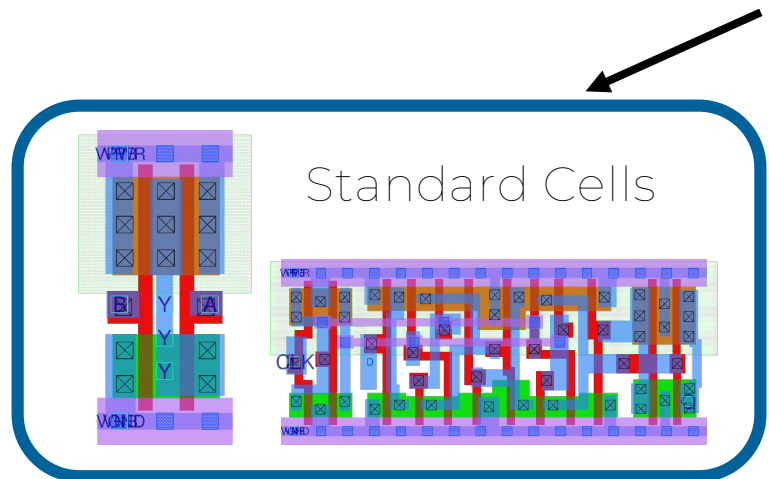
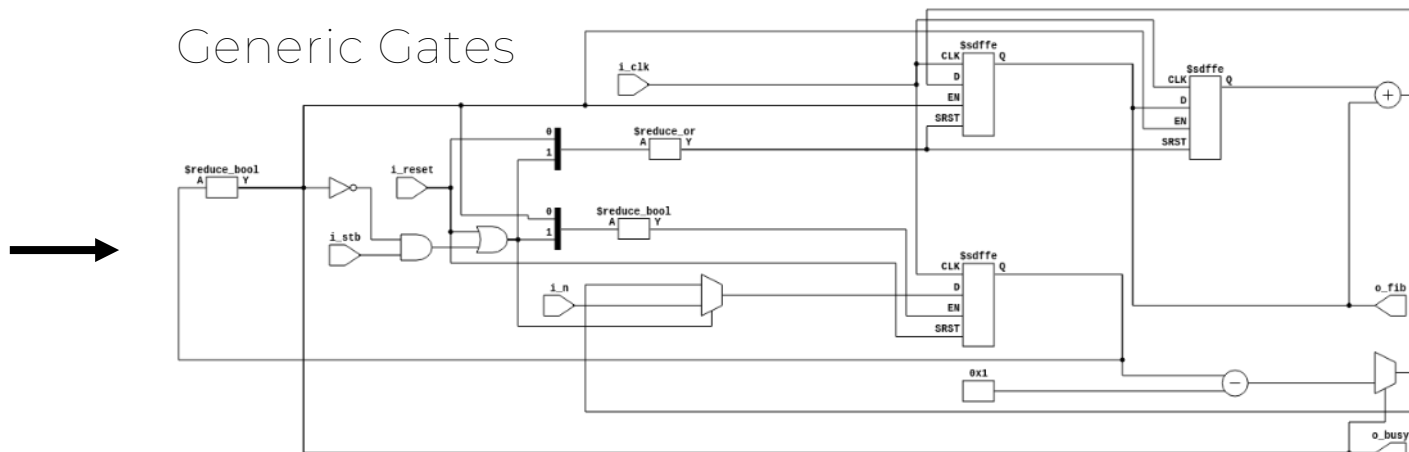
assign o_busy = (iteration != RESET);
assign o_fib = current;

always @(posedge i_clk) begin
  if (i_reset || (!o_busy && i_stb)) begin
    iteration <= i_n;
    prev [WIDTH-1:0] <= 1;
    current [WIDTH-1:0] <= 0;
  end
  else if (o_busy) begin
    iteration <= iteration - ONE;
    current <= prev + current;
    prev <= current;
  end

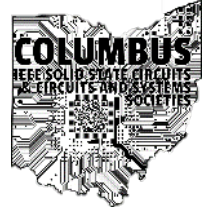
  if (i_reset) begin
    iteration <= RESET;
    prev [WIDTH-1:0] <= 1;
    current [WIDTH-1:0] <= 0;
  end
end
endmodule

```

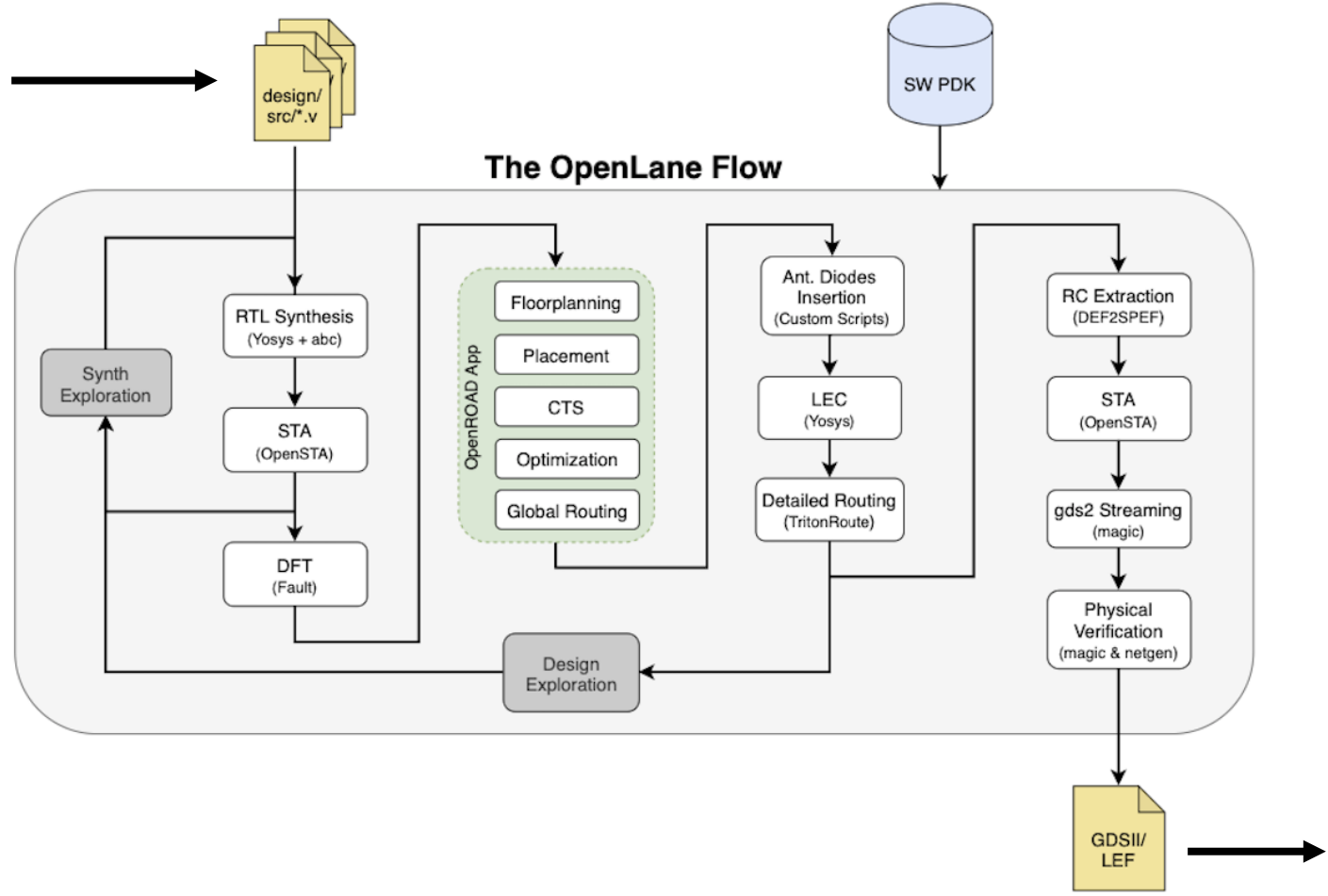
NORMAL fib.v



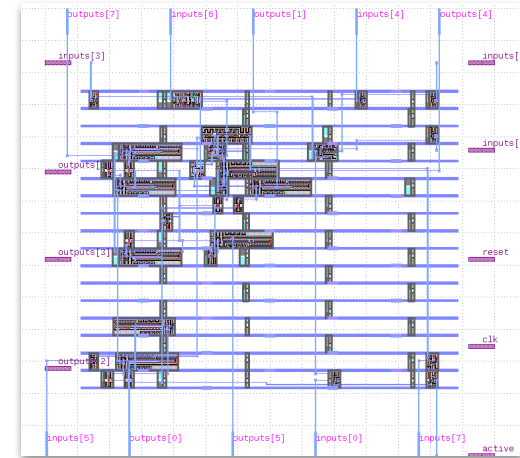
# OpenLane Flow



WOKwi  
or  
Verilog  
Files



Output Design  
(GDS File)

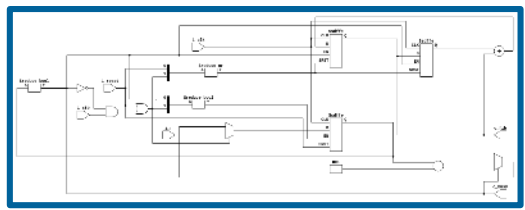
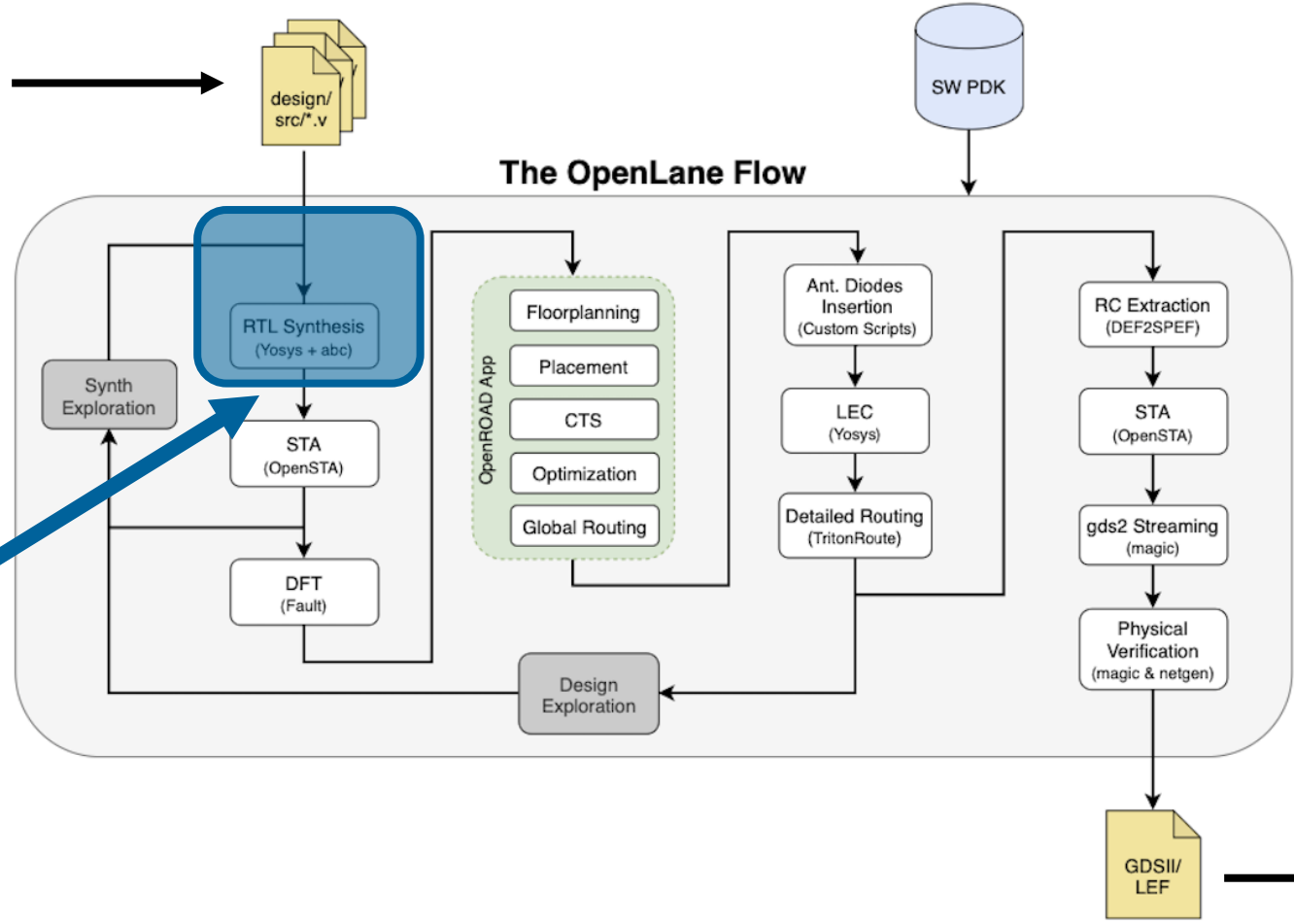


# OpenLane Flow

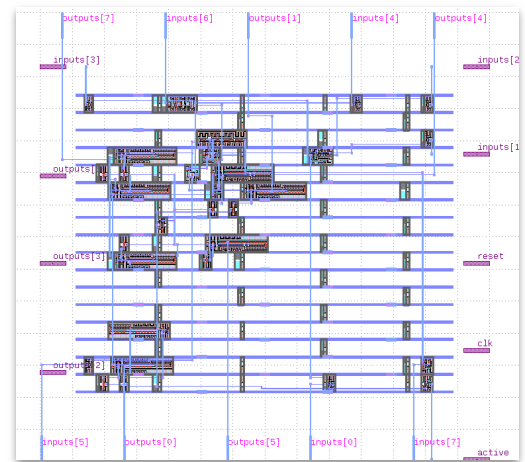
WOKwi

or

Verilog Files



Elaboration:  
Convert to generic gates

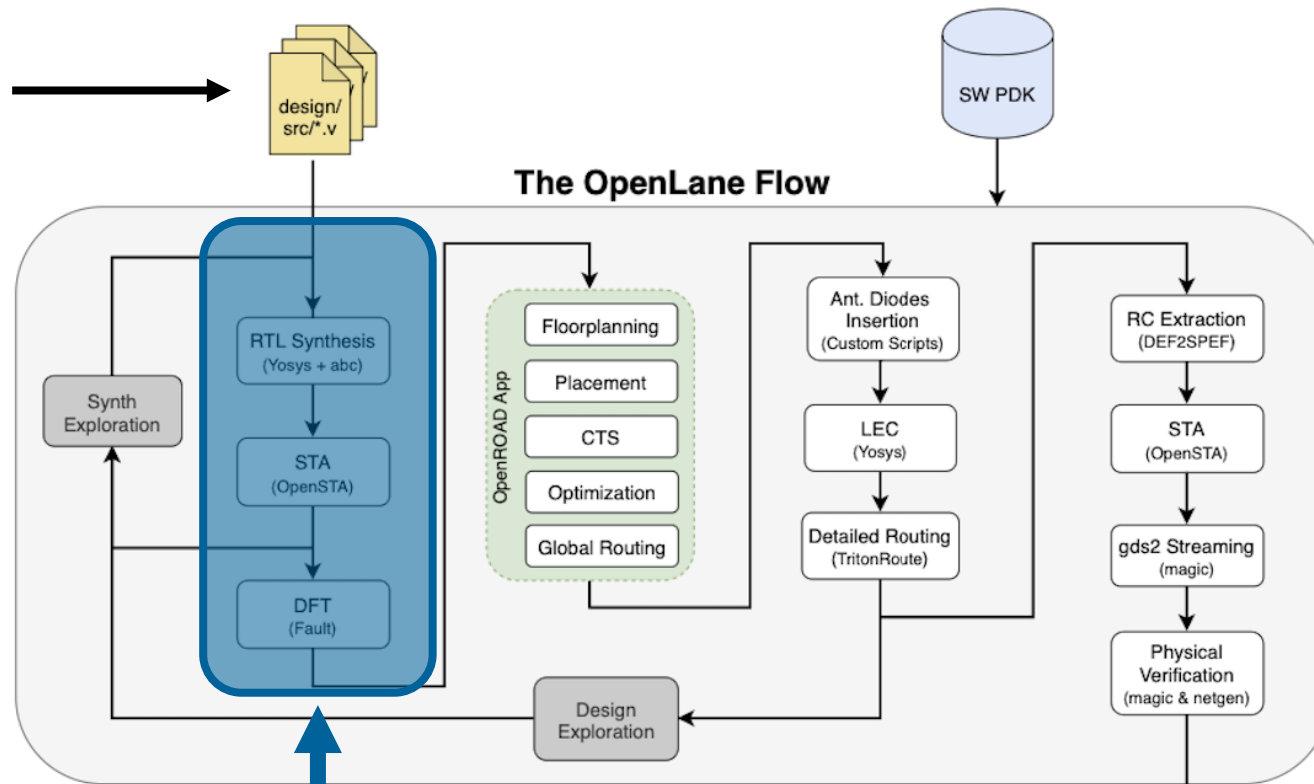
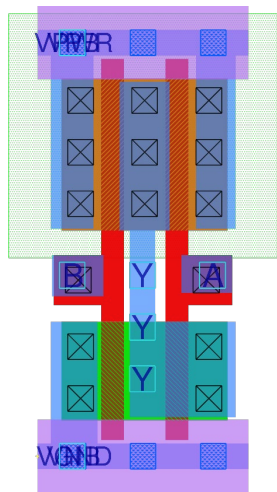


# OpenLane Flow

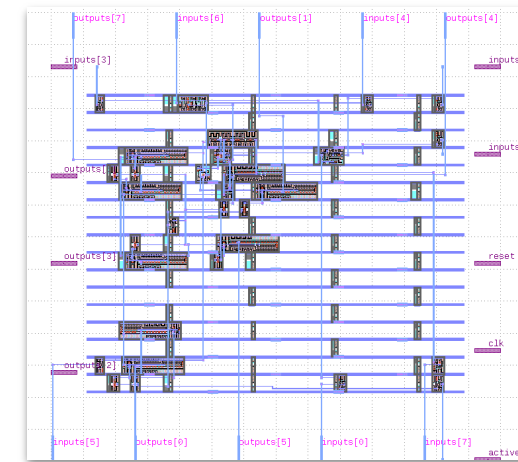
WOKwi

or

Verilog Files



Synthesis:  
Convert to “real” logic gates



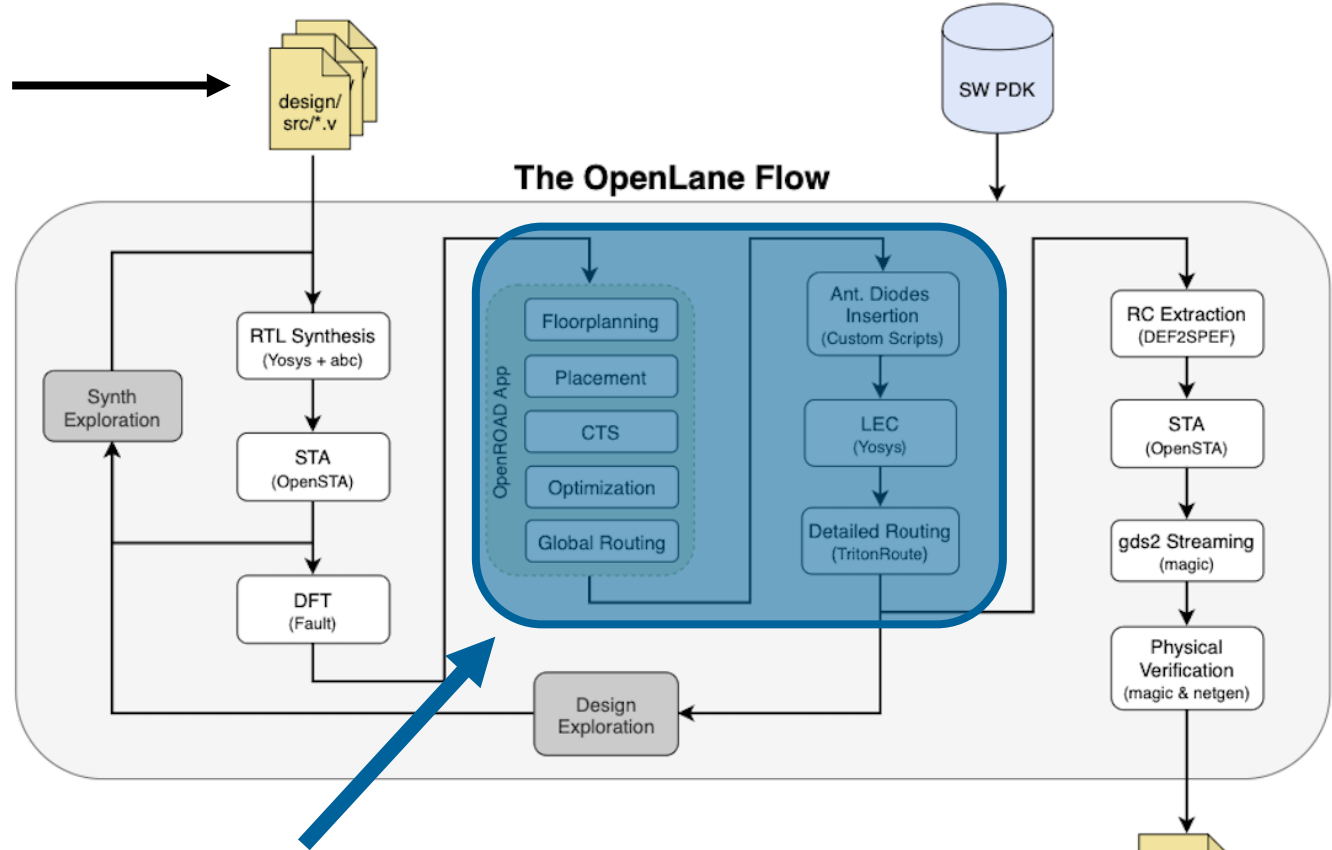


# OpenLane Flow

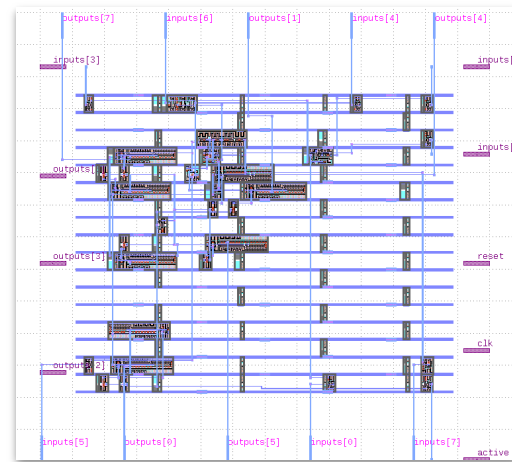
WOKwi

or

Verilog Files

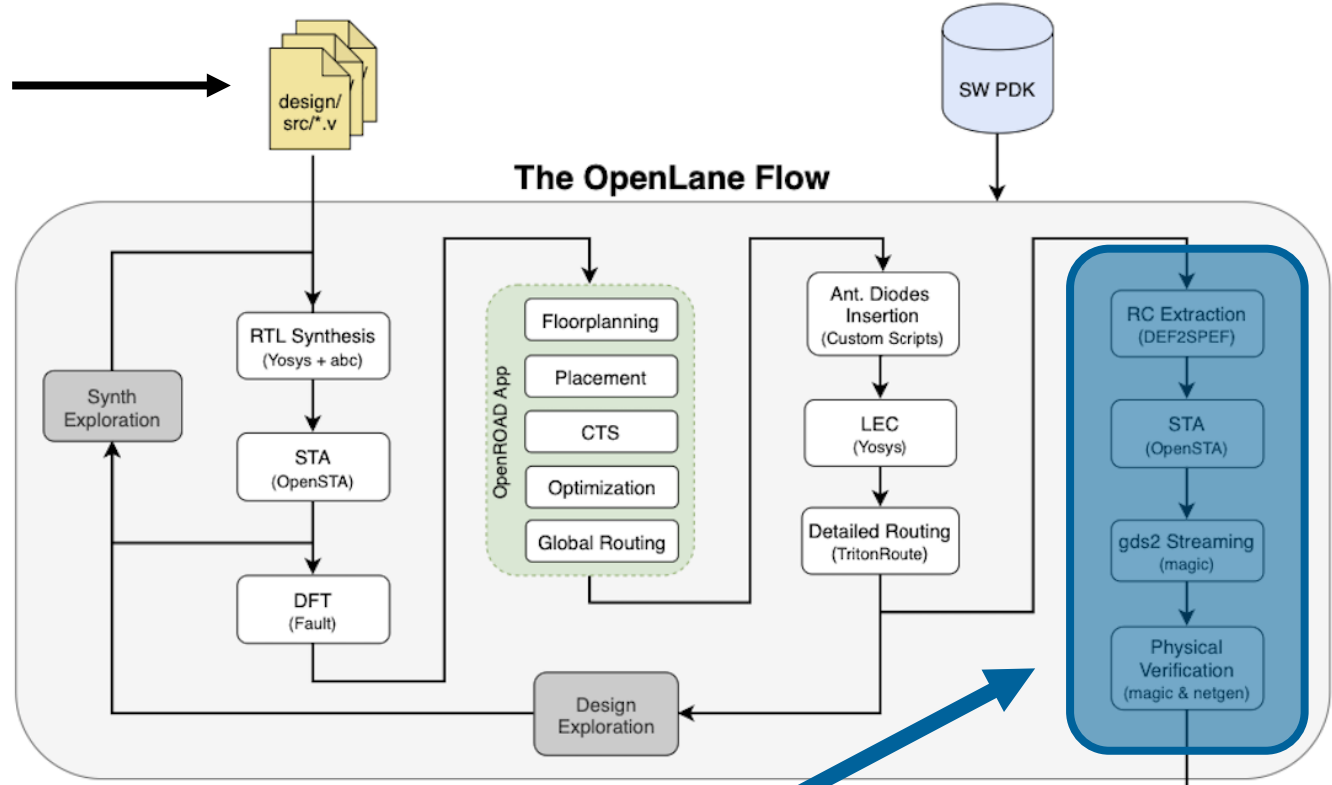


Place and Route (PnR):  
Place gates and route wires  
between them

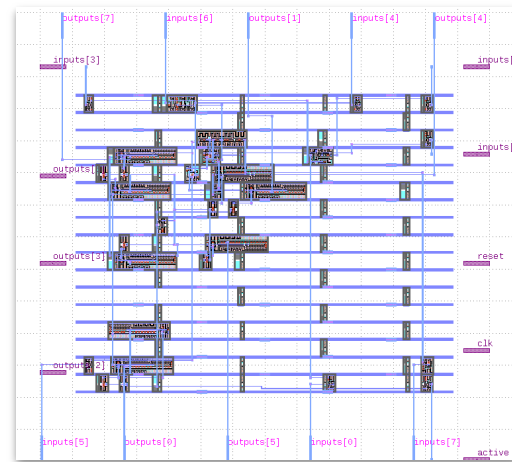


# OpenLane Flow

WOKwi  
or  
Verilog Files



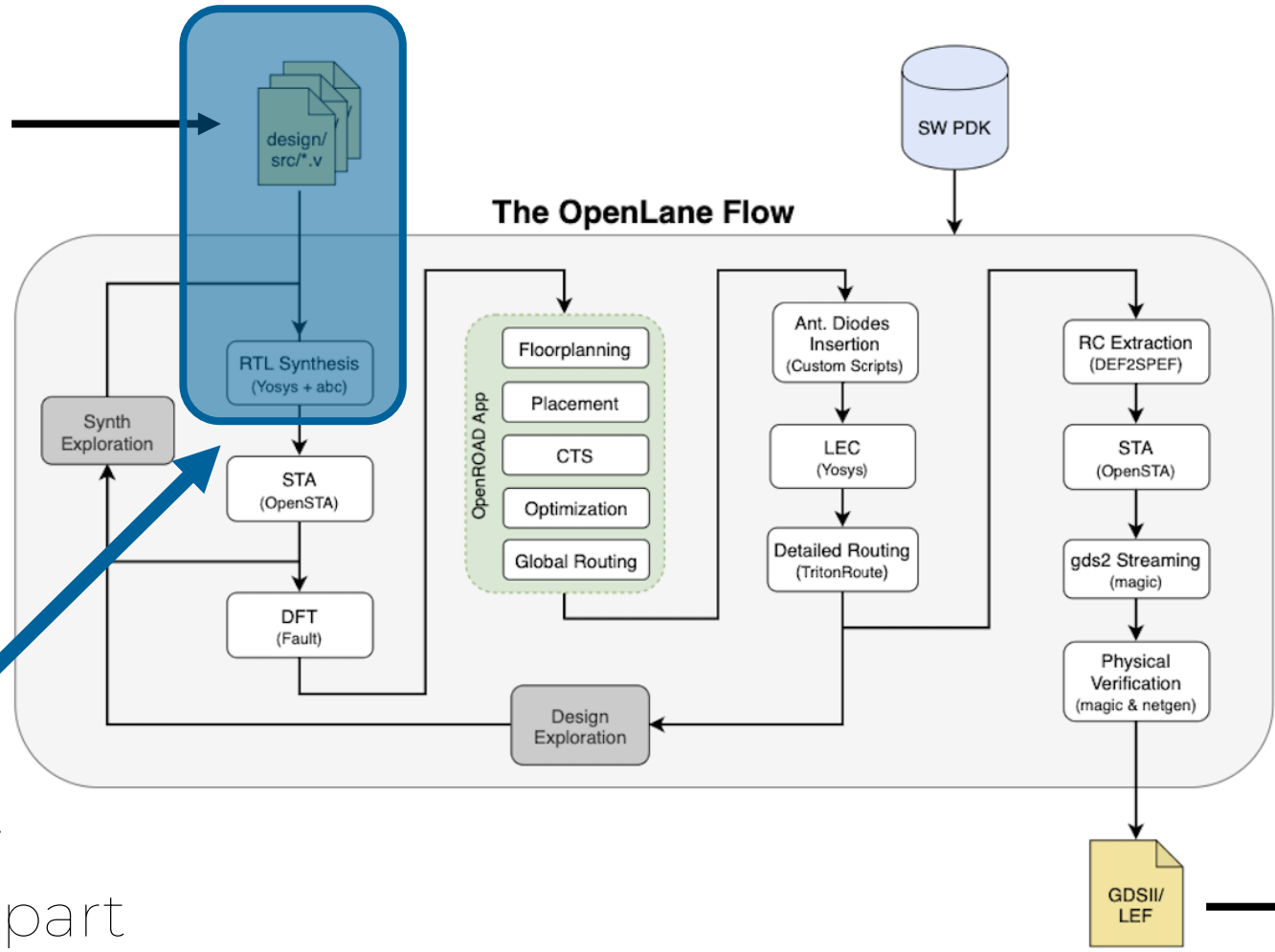
GDSII/LEF



Signoff:  
Make sure everything matches  
No rules are violated

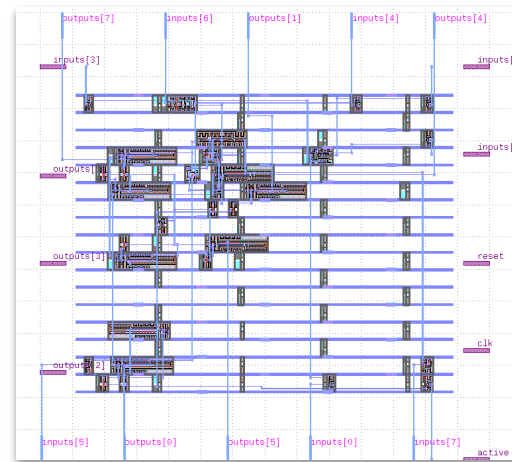
# OpenLane Flow

WOKwi  
or  
Verilog  
Files

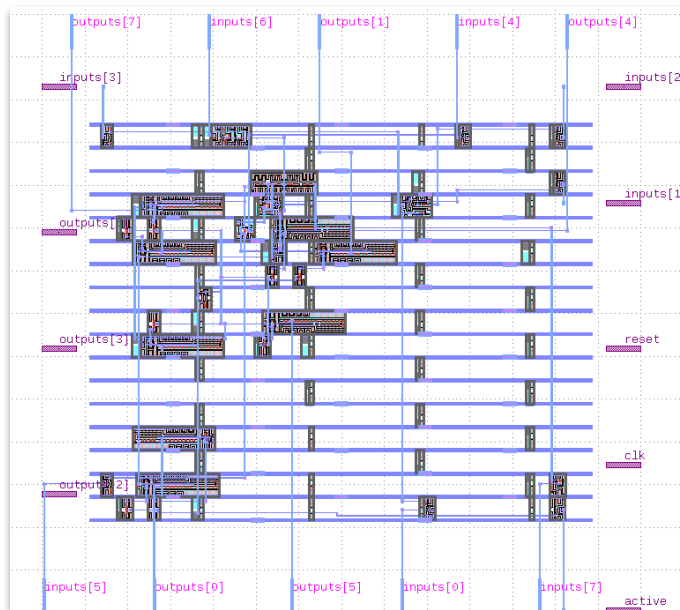


We're mostly dealing with this part

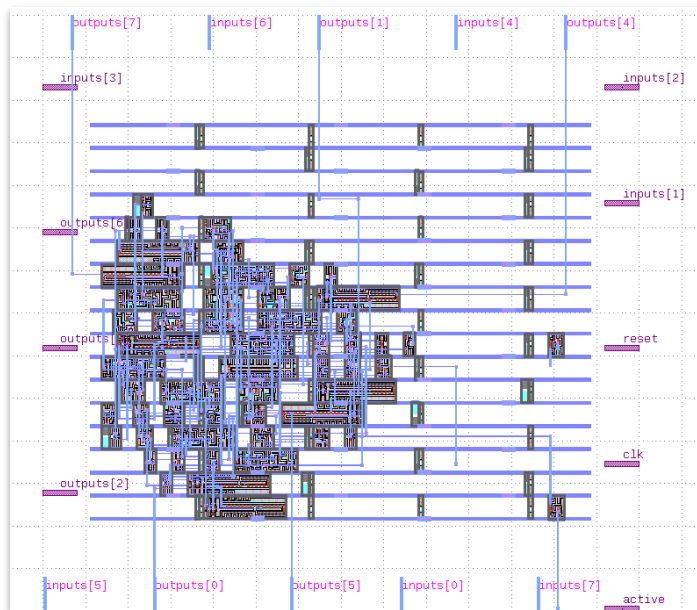
Tiny Tapeout (via Openlane) handles most of this for us!



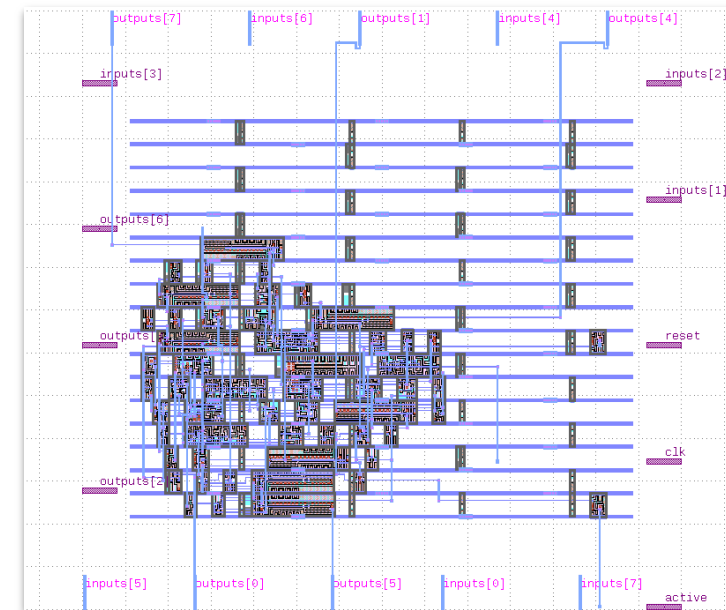
# GDS examples (all 70um x 70um)



binary to decimal  
converter  
25 cells

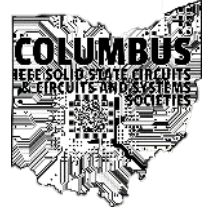


8 bit counter  
49 cells

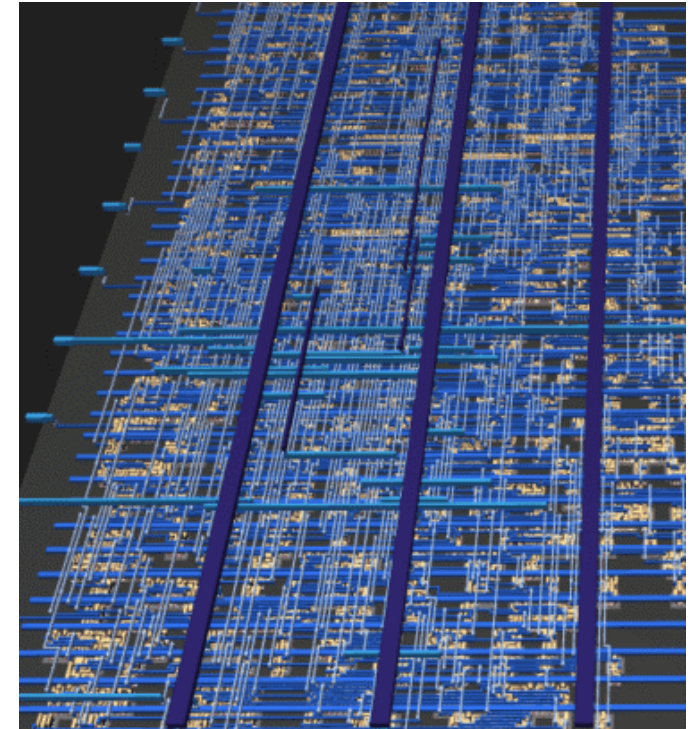


4 bit counter & bcd  
50 cells

# What did we learn?



- Design → Digital Flow → Physical Design
- Digital Design
  - Gates are building blocks of the design
  - Combinational Logic/Sequential Logic
  - Verilog lets us describe gates with code
- Digital Flow (OpenLane)
  - Elaboration: Hardware Description Language → Ideal Gates
  - Synthesis: Generating real gates from the design
  - Place and Route: Layout and route physical gates
  - Signoff: Check the design for layout/timing errors

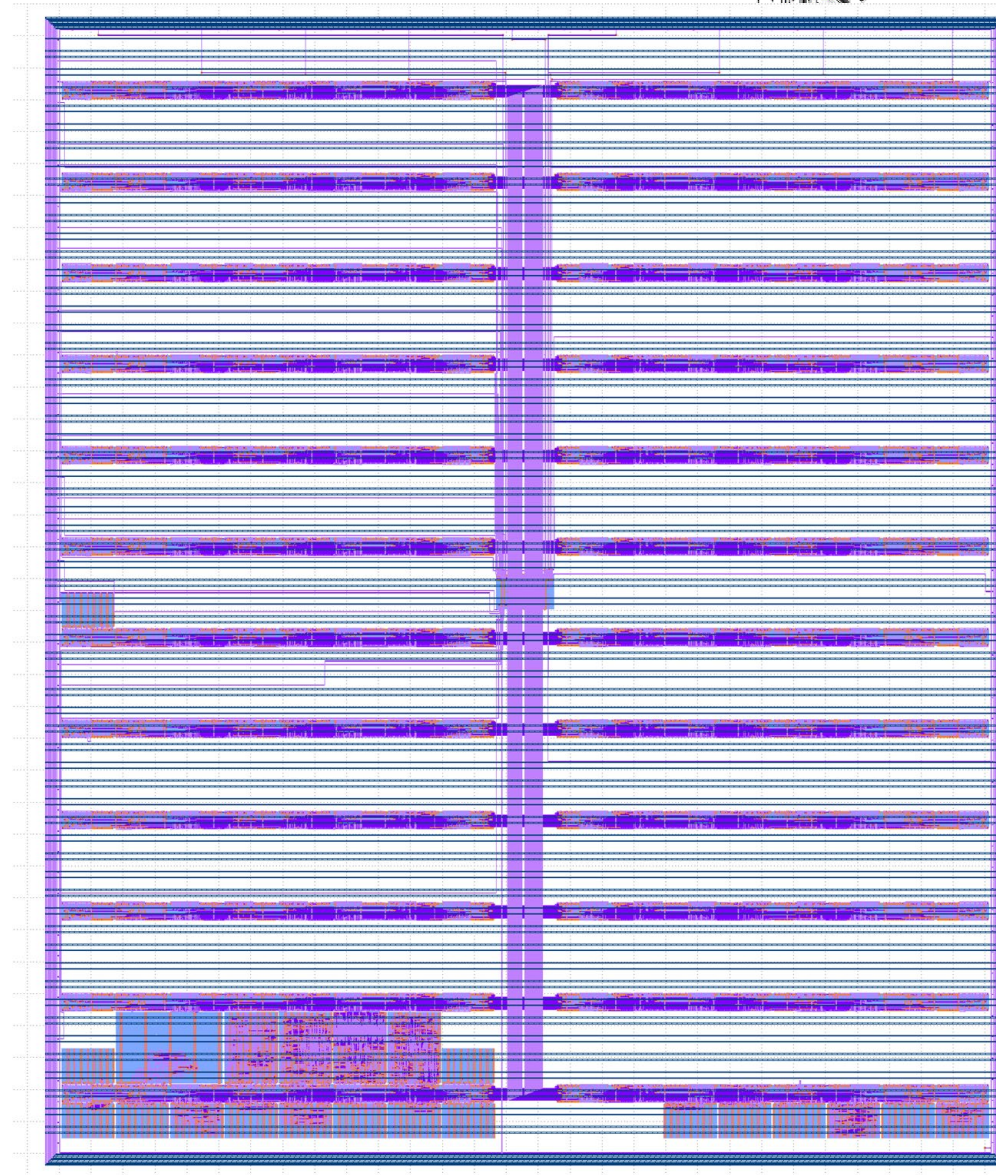




# Tiny Tapeout

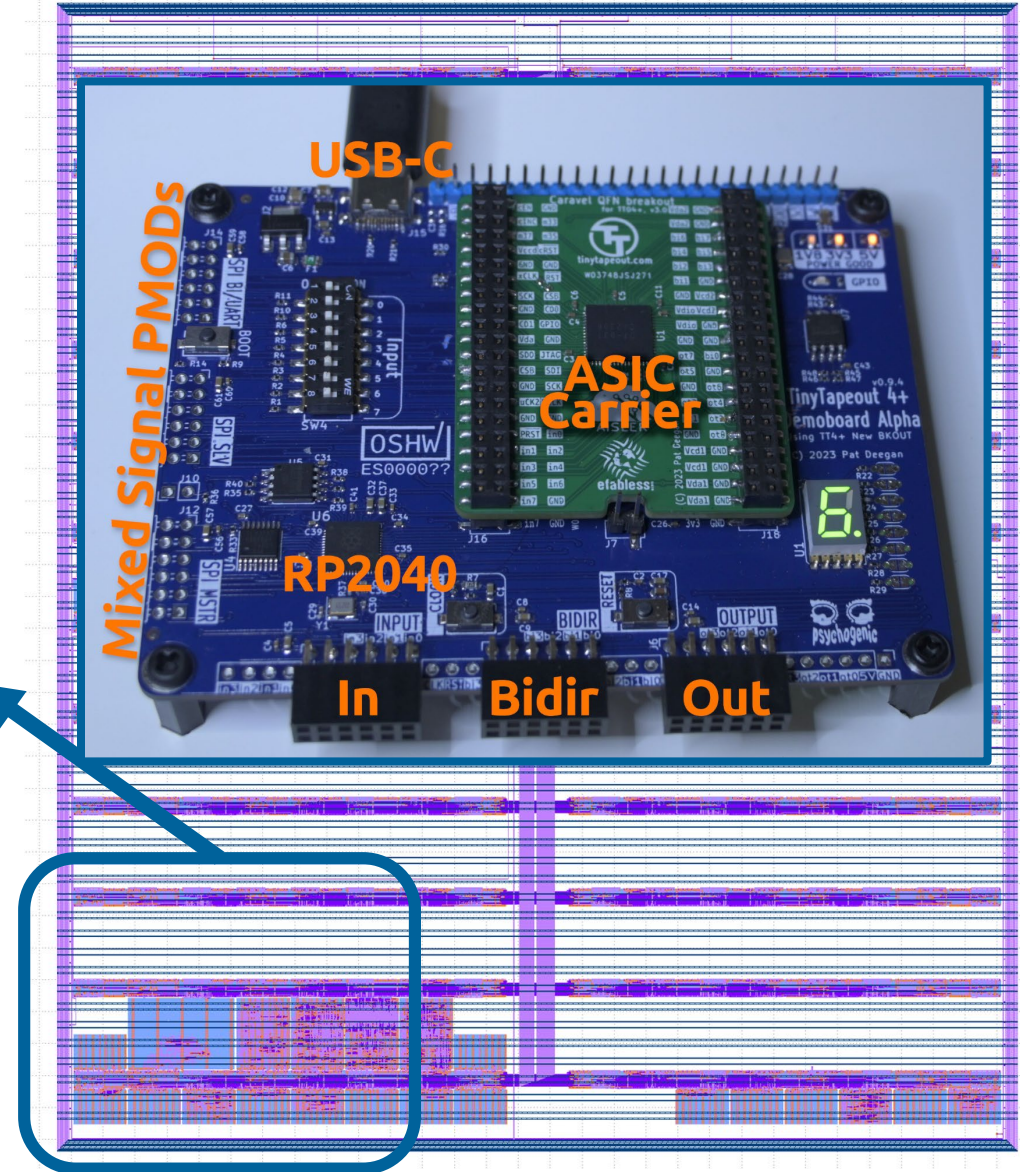
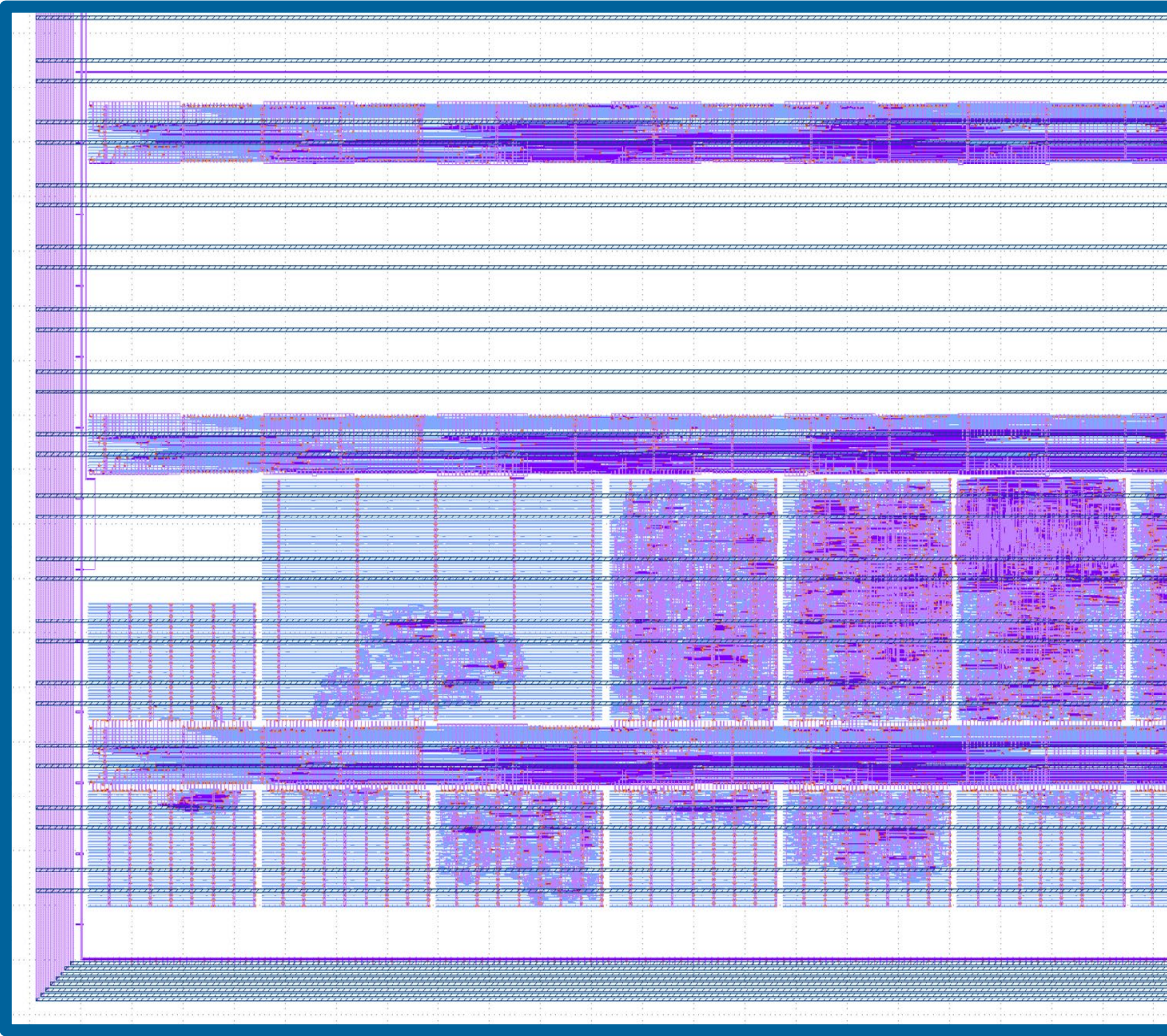
# How Tiny Tapeout Works

- Cloud based design
  - Runs OpenLane in Github actions
  - No tool install or download
  - 3D viewer / explorer
  - [Example Design](#)
- ~500 Projects merged into one IC
  - Reduced cost
  - Try other peoples designs
  - [Test PCB](#)



# How

- Clock
- Power
- Input
- Output
- Bidirectional
- ~50%
- Test PCB







# Tiny Tapeout 02 Datasheet

Project Repository

<https://github.com/TinyTapeout/tinytapeout-02>

December 7, 2022

## Contents

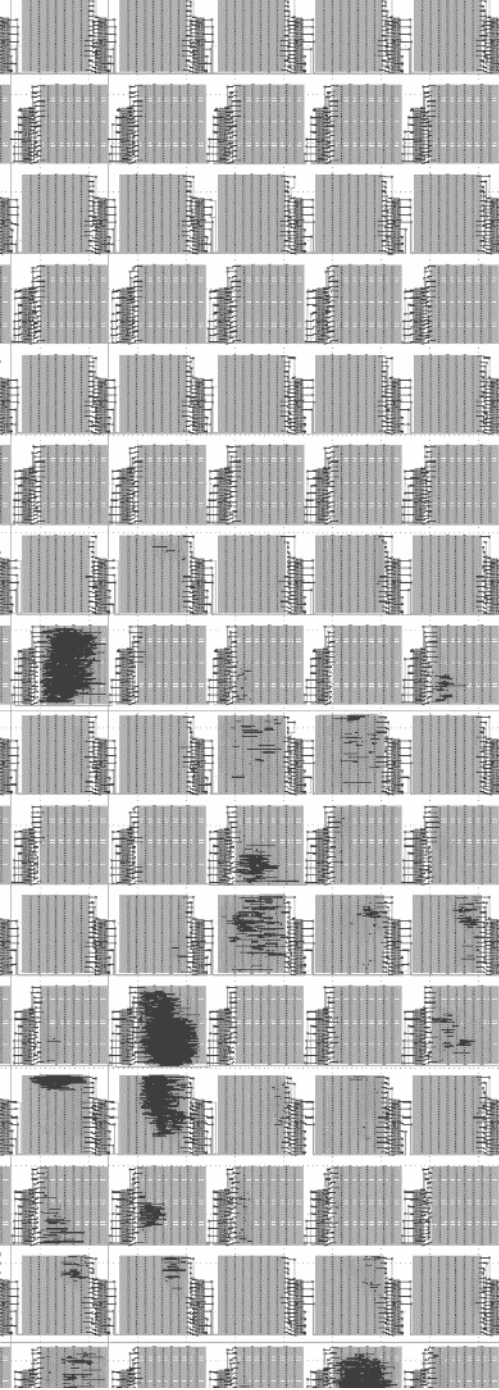
Render of whole chip

### Projects

- 0 : Test Inverter Project
- 1 : SIMON Cipher
- 2 : HD74480 Clock
- 3 : Scrolling Binary Matrix display
- 4 : Power supply sequencer
- 5 : Duty Controller
- 6 : S4GA: Super Slow Serial SRAM FPGA

7 8 9 10 11 12 13 14 15 16 17 18 19

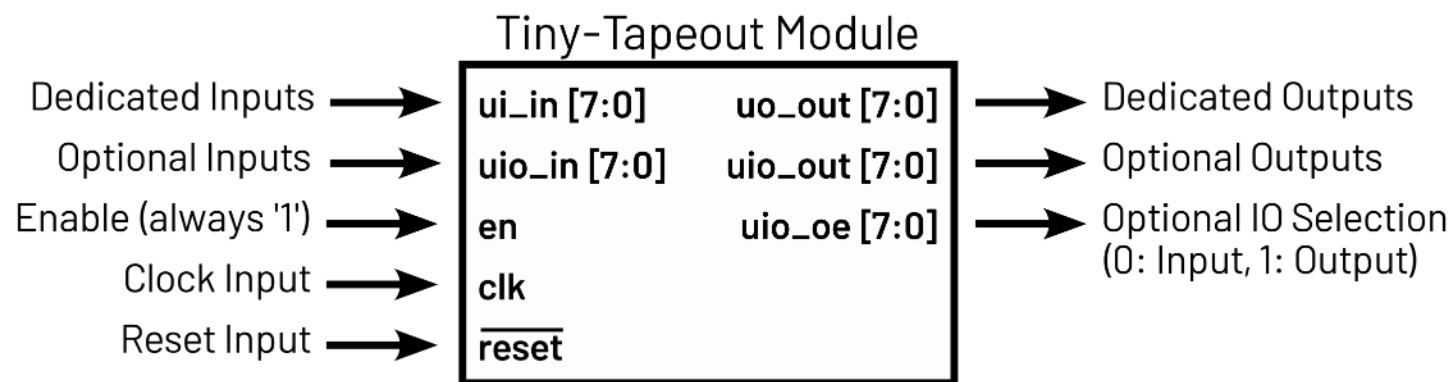
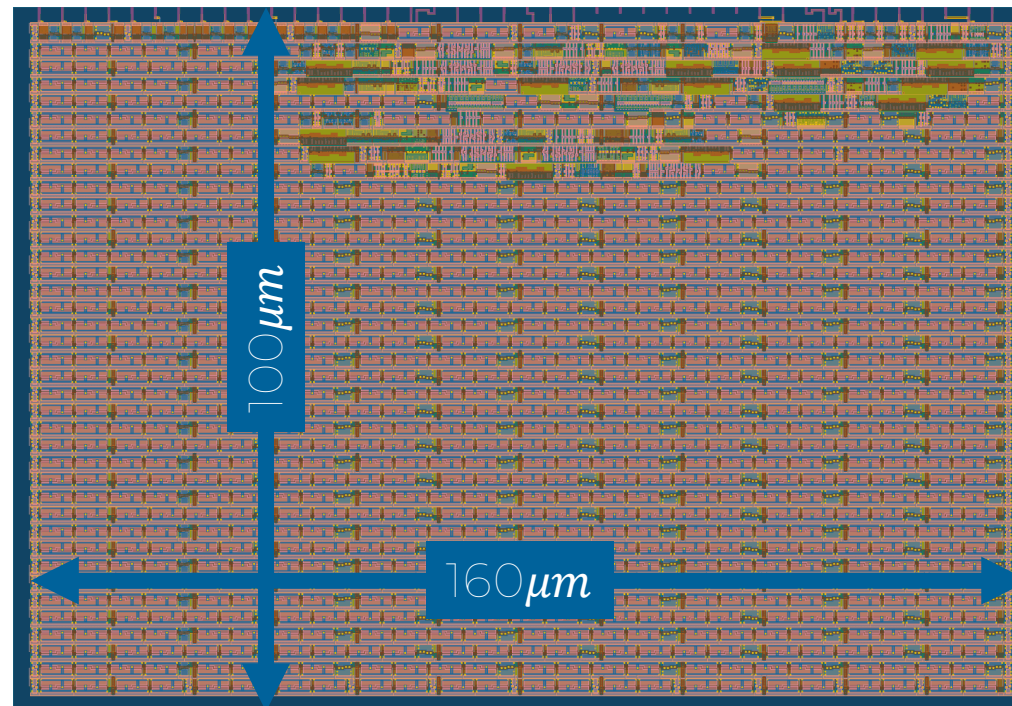
1



EFABLES  
 PN#: F1  
 LOT#: 4205819  
 WAFER#: 01-12  
 QTY: 350  
 DATE: 2333  
 JOB#: 31356  
 Syagrus  
 CAUTION: ELECTROSTATIC SENSITIVE DEVICE

# Physical Interface

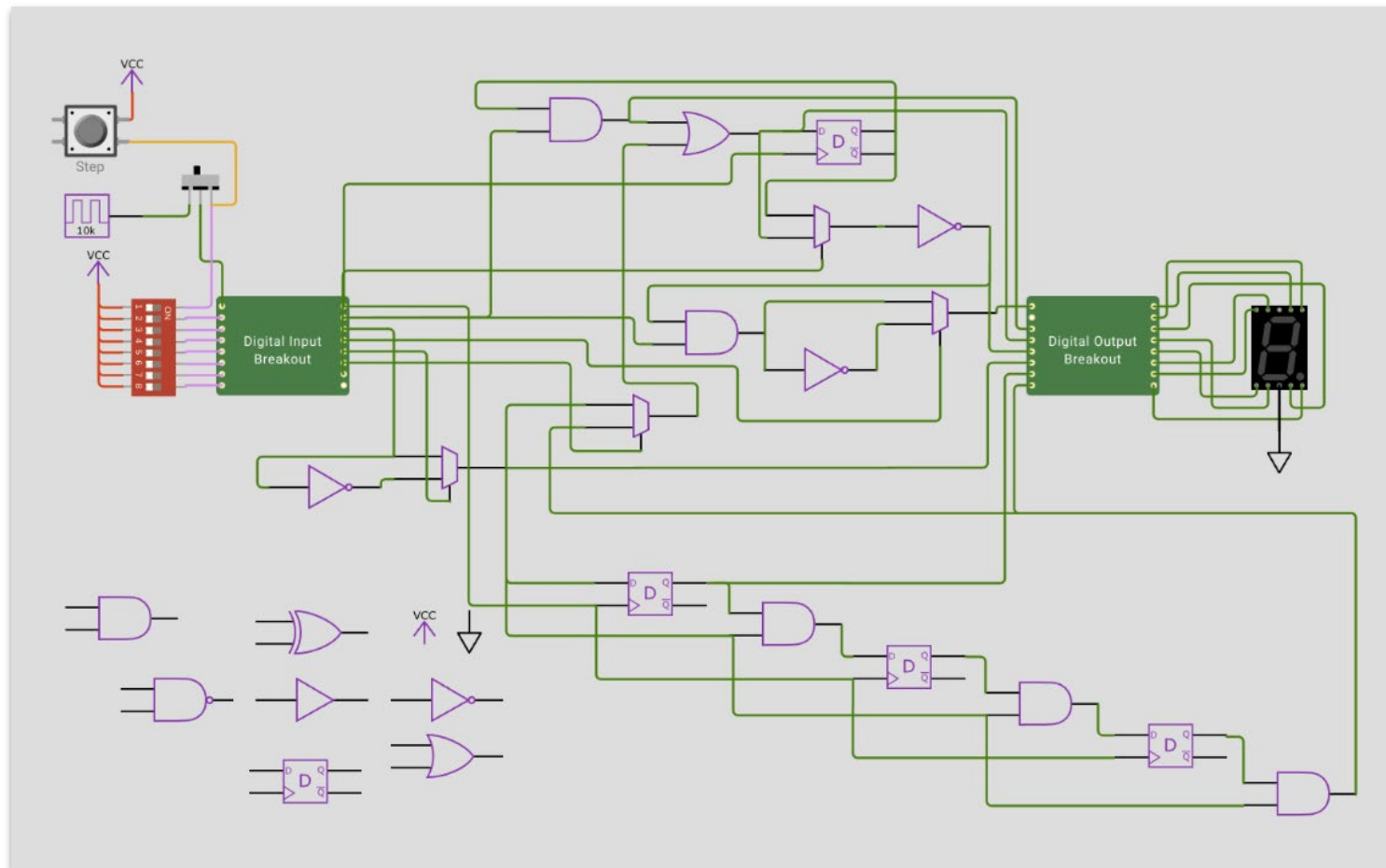
- 1x Tile
  - 160 x 100  $\mu\text{m}$  size
  - ~1000 gates
  - Can buy more than 1 tile
- Pins
  - Clock (~50MHz)
  - Reset
  - 8x Inputs
  - 8x Outputs
  - 8x Bidirectional



# Wokwi Design Flow

- [Wokwi](#) by Uri Shaked
  - Online Simulator
  - Exports Verilog Netlist
- [Examples](#)
  - Padlock
  - UART
  - 7-Segment Display

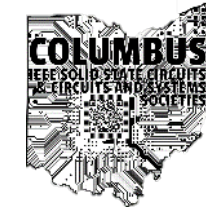
- dtype flop
- inverter
- 2 input and
- 2 input or
- 2 input xor
- 2 input mux
- 2 input nand



# Using GitHub Flow for Verilog

- [Tutorial Video](#)
- [Example Project](#) (Fibonacci Sequence Generator)
- Create new GitHub project by using [the template](#)
  - The project should be public!
  - [Enable GitHub Pages](#) (set to GitHub Actions)
- Update *info.yaml*
  - Top module
  - Source files
  - Area
  - Documentation
- Update *docs/info.md*
  - Add detailed documentation for your project

# Live Demo



[https://github.com/sellicott/sellicott\\_fib\\_seq](https://github.com/sellicott/sellicott_fib_seq)

If live demo isn't working



Demo Slides

Skip Demo

# Demo Slides (Home)



sellicott / sellicott\_fib\_seq

Code Issues Pull requests **Actions** Projects Security Insights **Settings**

sellicott\_fib\_seq Public  
generated from [TinyTapeout/tt07-verilog-template](#)

main 1 Branch 0 Tags

Go to file Add file Code

Commit	Message	Time
Samuel Ellicott	Remove extra quotes from Makefile ✓	9b78bf7 · 17 minutes ago 18 Commits
	.github/workflows	Initial commit 19 hours ago
	docs	Update documentation 17 hours ago
	src	Update project module to use fib module 19 hours ago
	test	Remove extra quotes from Makefile 17 minutes ago
	.gitignore	Initial commit 19 hours ago
	LICENSE	Initial commit 19 hours ago
	README.md	Initial commit 19 hours ago
	info.yaml	update info.yaml with project information 19 hours ago

gds passing docs passing test passing

### Tiny Tapeout Verilog Project Template

**About**  
Tiny Tapeout Fibonacci sequence generator example project

Readme  
Apache-2.0 license  
Activity  
0 stars  
1 watching  
0 forks

**Releases**  
No releases published  
[Create a new release](#)

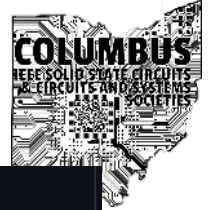
**Packages**  
No packages published  
[Publish your first package](#)

**Languages**

Language	Percentage
Verilog	46.2%
Tcl	27.6%
Python	14.9%
Makefile	11.3%

Exit Demo

# Demo Slides (wrapper source)



The image shows a screenshot of a code editor interface. On the left is a file explorer with a search bar and a list of files and folders. The files `fib.v` and `tt_um_sellicott_fib_seq.v` are highlighted with blue boxes. At the bottom left, there is a blue button labeled "Home". On the right is the code editor showing the Verilog source code for the `tt_um_sellicott_fib_seq` module. The code includes comments, port declarations, and module instantiation.

```
1  /*
2  * Copyright (c) 2024 Sam Ellicott
3  * SPDX-License-Identifier: Apache-2.0
4  */
5
6  `default_nettype none
7
8  module tt_um_sellicott_fib_seq (
9      input  wire [7:0] ui_in,    // Dedicated inputs
10     output wire [7:0] uo_out,   // Dedicated outputs
11     input  wire [7:0] uio_in,   // IOs: Input path
12     output wire [7:0] uio_out,  // IOs: Output path
13     output wire [7:0] uio_oe,   // IOs: Enable path (active high: 0=input, 1=output)
14     input  wire      ena,       // always 1 when the design is powered, so you can ignore it
15     input  wire      clk,       // clock
16     input  wire      rst_n      // reset_n - low to reset
17 );
18
19     fib #(
20         .WIDTH(8)
21     ) fib_inst (
22         // global control signals
23         .i_reset (~rst_n),
24         .i_clk   (clk),
25
26         // control signals
27         .i_stb  (uio_in [0]),
28         .o_busy (uio_out[1]),
29
30         // module inputs/outputs
31         .i_n   (ui_in [7:0]),
32         .o_fib (uo_out[7:0])
33     );
34
35     // All output pins must be assigned. If not used, assign to 0.
36     assign uio_oe [7:0] = 8'h2;
37     assign uio_out[7:2] = 6'h0;
38     assign uio_out[0]  = 1'h0;
39
40 endmodule
```

# Demo Slides (module source)



```
Code Blame 61 lines (51 loc) · 1.17 KB Code 55% faster with GitHub Copilot
1  /* fib.v
2  * Author: Samuel Ellicott
3  * Date: Thu Apr 11 18:43:53 EDT 2024
4  * Test of calculating the nth value of the fibonacci sequence
5  */
6  module fib (
7      // global control signals
8      i_reset,
9      i_clk,
10
11     // control signals
12     i_stb,
13     o_busy,
14
15     // module inputs/outputs
16     i_n,
17     o_fib
18 );
19 parameter WIDTH = 32;
20 localparam [WIDTH-1:0] RESET = 0;
21 localparam [WIDTH-1:0] ONE = 1;
22
23 // global control signals
24 input wire i_reset;
25 input wire i_clk;
26
27 // control signals
28 input wire i_stb;
29 output wire o_busy;
30
31 // module io
32 input wire [WIDTH-1:0] i_n;
33 output wire [WIDTH-1:0] o_fib;
34
35 reg [WIDTH-1:0] iteration;
36 reg [WIDTH-1:0] prev;
37 reg [WIDTH-1:0] current;
38
39 assign o_busy = (iteration != RESET);
40 assign o_fib = current;
```

Home



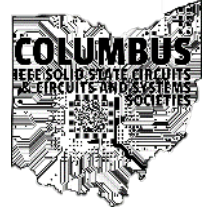
# Demo Slides (info.yaml)



The screenshot shows a code editor interface with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like .github, docs, src, and test, and files like confia.td, fib.v, tt\_um\_sellicott\_fib\_seq.v, .gitignore, LICENSE, README.md, and info.yaml. The code editor shows the content of info.yaml, which is a YAML file for project information. The code is as follows:

```
1 # Tiny Tapeout project information
2 project:
3   title: "Fibonacci Sequence Generator" # Project title
4   author: "Samuel Ellicott" # Your name
5   discord: "sellicott" # Your discord username, for communication and automatically assigning you a Tapeout role (optional)
6   description: "generate the Nth fibonacci number" # One line description of what your project does
7   language: "Verilog" # other examples include SystemVerilog, Amaranth, VHDL, etc
8   clock_hz: 0 # Clock frequency in Hz (or 0 if not applicable)
9
10 # How many tiles your design occupies? A single tile is about 167x108 uM.
11 tiles: "1x1" # Valid values: 1x1, 1x2, 2x2, 3x2, 4x2, 6x2 or 8x2
12
13 # Your top module name must start with "tt_um_". Make it unique by including your github username:
14 top_module: "tt_um_sellicott_fib_seq"
15
16 # List your project's source files here. Source files must be in ./src and you must list each source file separately, one per line:
17 source_files:
18   - "tt_um_sellicott_fib_seq.v"
19   - "fib.v"
20
21 # The pinout of your project. Leave unused pins blank. DO NOT delete or add any pins.
22 pinout:
23   # Inputs
24   ui[0]: "n[0]"
25   ui[1]: "n[1]"
26   ui[2]: "n[2]"
27   ui[3]: "n[3]"
28   ui[4]: "n[4]"
29   ui[5]: "n[5]"
30   ui[6]: "n[6]"
31   ui[7]: "n[7]"
32
33   # Outputs
34   uo[0]: "fib[0]"
35   uo[1]: "fib[1]"
```

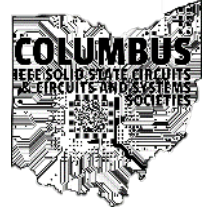
# Demo Slides (Test)



```
10 @cocotb.test()
11 async def test_project(dut):
12     dut._log.info("Start")
13
14     # Set the clock period to 10 us (100 KHz)
15     clock = Clock(dut.clk, 10, units="us")
16     cocotb.start_soon(clock.start())
17
18     # Reset
19     dut._log.info("Reset")
20     dut.ena.value = 1
21     dut.ui_in.value = 0
22     dut.uio_in.value = 0
23     dut.rst_n.value = 0
24     await ClockCycles(dut.clk, 10)
25     dut.rst_n.value = 1
26
27     dut._log.info("Test project behavior")
28
29     for i in range(0, 10):
30         dut._log.info(f"Test n={i}")
31         fib_n = int(await get_fib_n(dut, i))
32         calc_fib = calc_fib_n(i)
33         dut._log.info(f"hw fib: {fib_n}, sw fib: {calc_fib}")
34         assert fib_n == calc_fib
35
36 async def get_fib_n(dut, n):
37     # Set the input values you want to test
38     dut.ui_in.value = n
39     dut.uio_in.value = 1
40
41     # Wait for one clock cycle, then clear the strobe pin
42     await ClockCycles(dut.clk, 1)
43     dut.uio_in.value = 0
44
45     busy_val = True
46
47     while busy_val:
48         # Wait for one clock cycle, then check the output
49         await ClockCycles(dut.clk, 1)
50         busy_val = (dut.uio_out.value & 0x02) != 0
```

Home

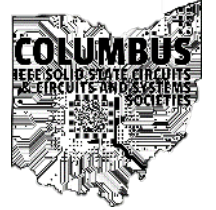
# Demo Slides (Testbench)



The image shows a code editor interface. On the left is a file explorer with a search bar 'Go to file' and a list of files and folders. The 'test' folder is expanded, showing files: Makefile, README.md, fib\_tb.v, requirements.txt, tb.gtkw, tb.v, test.py, .gitignore, LICENSE, README.md, and info.yaml. The 'tb.v' file is selected. At the bottom left is a blue button labeled 'Home'. On the right is the code editor showing Verilog testbench code for a module named 'tb'. The code includes comments, signal declarations, and an instantiation of the 'tt\_um\_sellicott\_fib\_seq' module.

```
4  /* This testbench just instantiates the module and makes some convenient wires
5     that can be driven / tested by the cocotb test.py.
6  */
7  module tb ();
8
9     // Dump the signals to a VCD file. You can view it with gtkwave.
10    initial begin
11        $dumpfile("tb.vcd");
12        $dumpvars(0, tb);
13    #1;
14    end
15
16    // Wire up the inputs and outputs:
17    reg clk;
18    reg rst_n;
19    reg ena;
20    reg [7:0] ui_in;
21    reg [7:0] uio_in;
22    wire [7:0] uo_out;
23    wire [7:0] uio_out;
24    wire [7:0] uio_oe;
25
26    // Replace tt_um_example with your module name:
27    tt_um_sellicott_fib_seq user_project (
28
29        // Include power ports for the Gate Level test:
30    `ifdef GL_TEST
31        .VPWR(1'b1),
32        .VGND(1'b0),
33    `endif
34
35        .ui_in (ui_in),    // Dedicated inputs
36        .uo_out (uo_out), // Dedicated outputs
37        .uio_in (uio_in), // IOs: Input path
38        .uio_out(uio_out), // IOs: Output path
39        .uio_oe (uio_oe), // IOs: Enable path (active high: 0=input, 1=output)
40        .ena   (ena),    // enable - goes high when design is selected
41        .clk   (clk),    // clock
42        .rst_n (rst_n)   // not reset
43    );
44
45    endmodule
```

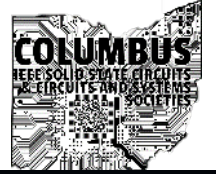
# Demo Slides (Makefile)



```
3
4 # defaults
5 SIM ?= icarus
6 TOPLEVEL_LANG ?= verilog
7 SRC_DIR = $(PWD)/../src
8 PROJECT_SOURCES = tt_um_sellicott_fib_seq.v fib.v
9
10 ifneq ($(GATES),yes)
11
12 # RTL simulation:
13 SIM_BUILD = sim_build/rtl
14 VERILOG_SOURCES += $(addprefix $(SRC_DIR)/,$(PROJECT_SOURCES))
15 COMPILE_ARGS += -I"$(SRC_DIR)"
16
17 else
18
19 # Gate level simulation:
20 SIM_BUILD = sim_build/gl
21 COMPILE_ARGS += -DGL_TEST
22 COMPILE_ARGS += -DFUNCTIONAL
23 COMPILE_ARGS += -DUSE_POWER_PINS
24 COMPILE_ARGS += -DSIM
25 COMPILE_ARGS += -DUNIT_DELAY=#1
26 VERILOG_SOURCES += $(PDK_ROOT)/sky130A/libs.ref/sky130_fd_sc_hd/verilog/primitives.v
27 VERILOG_SOURCES += $(PDK_ROOT)/sky130A/libs.ref/sky130_fd_sc_hd/verilog/sky130_fd_sc_hd.v
28
29 # this gets copied in by the GDS action workflow
30 VERILOG_SOURCES += $(PWD)/gate_level_netlist.v
31
32 endif
33
34 # Include the testbench sources:
35 VERILOG_SOURCES += $(PWD)/tb.v
36 TOPLEVEL = tb
37
38 # MODULE is the basename of the Python test file
39 MODULE = test
40
41 # include cocotb's make rules to take care of the simulator setup
42 include $(shell cocotb-config --makefiles)/Makefile.sim
```

Home

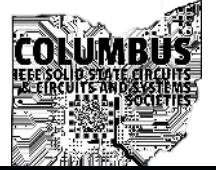
# Demo Slides (GitHub Actions)



The screenshot shows the GitHub Actions interface. The top navigation bar includes links for Code, Issues, Pull requests, Actions (highlighted), Projects, Security, Insights, and Settings. The left sidebar contains 'Actions' with a 'New workflow' button, 'All workflows', 'Workflows' (with sub-items docs, gds, test), and 'Management' (with sub-items Caches, Runners). The main area is titled 'All workflows' and shows a list of workflow runs. Three runs for 'Remove extra quotes from Makefile' are highlighted with blue boxes. Below them are runs for 'try and fix software fibonacci generator' (one failed, one succeeded) and 'Flip check on busy signal' (one failed, one succeeded). A blue button labeled 'Home' is in the bottom left corner.

Workflow Name	Event	Status	Branch	Actor	Time
Remove extra quotes from Makefile	docs #15	Success	main	sellicott	2 hours ago, 1m 20s
Remove extra quotes from Makefile	gds #15	Success	main	sellicott	2 hours ago, 3m 42s
Remove extra quotes from Makefile	test #15	Success	main	sellicott	2 hours ago, 33s
try and fix software fibonacci generator	docs #14	Success	main	sellicott	18 hours ago, 1m 8s
try and fix software fibonacci generator	test #14	Success	main	sellicott	18 hours ago, 36s
try and fix software fibonacci generator	gds #14	Failure	main	sellicott	18 hours ago, 3m 45s
Flip check on busy signal	gds #13	Failure	main	sellicott	19 hours ago, 4m 20s
Flip check on busy signal	docs #12	Success	main	sellicott	19 hours ago, 1m 6s

# Demo Slides (Test Action)



The screenshot displays the GitHub Actions interface for a workflow named 'test'. The 'Actions' tab is selected in the top navigation bar. The workflow run is titled 'Remove extra quotes from Makefile #15' and is in a 'Success' state. The run was triggered by a push to the 'main' branch by user 'sellicott' 2 hours ago. The total duration of the run is 33 seconds, and it produced one artifact. The 'test.yaml' file is shown with the 'test' job, which completed successfully in 23 seconds. The 'Artifacts' section lists a single artifact named 'test-vcd' with a size of 1.25 KB. A 'test summary' section at the bottom indicates that all tests passed (1 test passed). A blue 'Home' button is visible in the bottom left corner of the interface.

Code Issues Pull requests **Actions** Projects Security Insights Settings

← test

✓ Remove extra quotes from Makefile #15

Summary

Jobs

- ✓ test

Run details

- Usage
- Workflow file

Triggered via push 2 hours ago

Triggered via	Status	Total duration	Artifacts
sellicott pushed -o 9b78bf7 main	Success	33s	1

test.yaml  
on: push

✓ test	23s
--------	-----

Artifacts  
Produced during runtime

Name	Size
test-vcd	1.25 KB

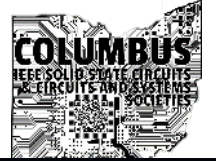
test summary

✓ All tests passed  
1 tests passed

Job summary generated at run-time

Home

# Demo Slides (Test Results)



The screenshot shows a software interface with a top navigation bar containing 'Summary', 'Actions', 'Projects', 'Security', 'Insights', and 'Settings'. A search bar for logs is on the right. On the left, a sidebar lists 'Jobs' with a 'test' job highlighted, and 'Run details' with 'Usage' and 'Workflow file' options. The main area displays a terminal log for 'Run tests'.

```
21 rm -f results.xml
22 MODULE=test TESTCASE= TOPLEVEL=tb TOPLEVEL_LANG=verilog \
23     /usr/bin/vvp -M /opt/hostedtoolcache/Python/3.11.9/x64/lib/python3.11/site-packages/cocotb/libs -m libcocotbvpi_icarus sim_build/rtl/sim.vvp
24     --ns INFO     gpi             .mbed/gpi_embed.cpp:76 in set_program_name_in_venv Did not detect Python virtual environment. Using system-wide Python interpreter
25     --ns INFO     gpi             ../gpi/GpiCommon.cpp:101 in gpi_print_registered_impl VPI registered
26     0.00ns INFO     cocotb          Running on Icarus Verilog version 11.0 (stable)
27     0.00ns INFO     cocotb          Running tests with cocotb v1.8.1 from /opt/hostedtoolcache/Python/3.11.9/x64/lib/python3.11/site-packages/cocotb
28     0.00ns INFO     cocotb          Seeding Python random module with 1714067530
29     0.00ns INFO     cocotb.regression Found test test.test_project
30     0.00ns INFO     cocotb.regression running test_project (1/1)
31     0.00ns INFO     cocotb.tb       Start
32     0.00ns INFO     cocotb.tb       Reset
33     90000.00ns INFO    cocotb.tb       Test project behavior
34     90000.00ns INFO    cocotb.tb       Test n=0
35     110000.00ns INFO   cocotb.tb       hw fib: 0, sw fib: 0
36     110000.00ns INFO   cocotb.tb       Test n=1
37     140000.00ns INFO   cocotb.tb       hw fib: 1, sw fib: 1
38     140000.00ns INFO   cocotb.tb       Test n=2
39     180000.00ns INFO   cocotb.tb       hw fib: 1, sw fib: 1
40     180000.00ns INFO   cocotb.tb       Test n=3
41     230000.00ns INFO   cocotb.tb       hw fib: 2, sw fib: 2
42     230000.00ns INFO   cocotb.tb       Test n=4
43     290000.00ns INFO   cocotb.tb       hw fib: 3, sw fib: 3
44     290000.00ns INFO   cocotb.tb       Test n=5
45     360000.00ns INFO   cocotb.tb       hw fib: 5, sw fib: 5
46     360000.00ns INFO   cocotb.tb       Test n=6
47     440000.00ns INFO   cocotb.tb       hw fib: 8, sw fib: 8
48     440000.00ns INFO   cocotb.tb       Test n=7
49     530000.00ns INFO   cocotb.tb       hw fib: 13, sw fib: 13
50     530000.00ns INFO   cocotb.tb       Test n=8
51     630000.00ns INFO   cocotb.tb       hw fib: 21, sw fib: 21
52     630000.00ns INFO   cocotb.tb       Test n=9
53     740000.00ns INFO   cocotb.tb       hw fib: 34, sw fib: 34
54     740000.00ns INFO   cocotb.regression test_project passed
55     740000.00ns INFO   cocotb.regression
56
57     ** TEST STATUS SIM TIME (ns) REAL TIME (s) RATIO (ns/s) **
58     ** test.test_project PASS 740000.00 0.01 87055366.01 **
59
60     ** TESTS=1 PASS=1 FAIL=0 SKIP=0 740000.00 0.03 21172371.44 **
61
```

Home

# Demo Slides (Test GTKWave)

The screenshot displays the GTKWave interface for a Verilog testbench. The left sidebar shows the project structure with 'fib\_inst' selected. The center pane lists signals, with 'Output Pins' highlighted. The right pane shows a timing diagram with a 100 ns scale. Below the main window, a file explorer lists files like 'tb.gtkw', 'tb.v', 'tb.vcd', 'test.py', and 'test-vcd.zip'. A command prompt window shows the execution of 'gtkwave tb.vcd tb.gtkw' and the resulting simulation output.

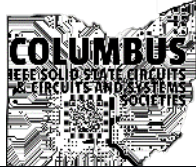
GitHub Actions

Home

Test Results



# Demo Slides (Docs Action)



<> Code Issues Pull requests **Actions** Projects Security Insights Settings

← docs

✓ Remove extra quotes from Makefile #15

Summary

Jobs

- ✓ docs

Run details

- Usage
- Workflow file

Triggered via push 2 hours ago

Triggered via	Status	Total duration	Artifacts
sellicott pushed -o 9b78bf7 main	Success	1m 20s	1

docs.yaml

on: push

- ✓ docs 1m 10s

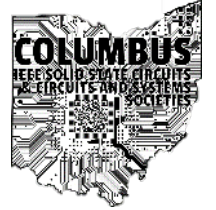
Artifacts

Produced during runtime

Name	Size
PDF	205 KB

Home

# Demo Slides (Datasheet)



## Fibonacci Sequence Generator

- Author: Samuel Ellicott
- Description: generate the Nth fibonacci number
- Language: Verilog

### How it works

The project takes in the index of Fibonacci number to generate ( $n=0 \rightarrow 1$ ,  $n=1 \rightarrow 1, \dots$ ). Where  $n$  is an 8-bit unsigned integer on the  $n[7:0]$  pins. To start generating the sequence  $start\_stb$  should be asserted for one clock cycle. While the module is working, the busy signal will be asserted. After the busy signal falls to 0, the Nth Fibonacci number is available on  $fib[7:0]$  pins

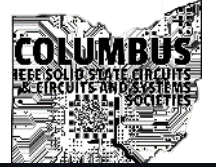
### Pinout

#	Input	Output	Bidirectional
0	$n[0]$	$fib[0]$	$start\_stb$
1	$n[1]$	$fib[1]$	$busy$
2	$n[2]$	$fib[2]$	
3	$n[3]$	$fib[3]$	
4	$n[4]$	$fib[4]$	
5	$n[5]$	$fib[5]$	
6	$n[6]$	$fib[6]$	
7	$n[7]$	$fib[7]$	

GitHub  
Actions

Home

# Demo Slides (GDS Action)



**Jobs**

- gds
- precheck
- gl\_test
- viewer

**Run details**

- Usage
- Workflow file

**Actions** Projects Security Insights Settings duration Artifacts

sellicott pushed → 9b78bf7 main **Success** 3m 42s 6

**gds.yaml**  
on: push

gds (2m 28s) → precheck (57s), gl\_test (46s), viewer (21s)

**Artifacts**  
Produced during runtime

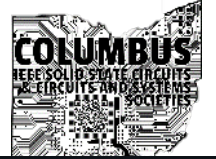
Name	Size
GDS_logs	32 MB
gatelevel_test_vcd	22.4 KB
gds_render	194 KB
github-pages	1.03 MB
precheck_reports	5.92 KB
tt_submission	415 KB

Home

Synthesis/PnR Log Files

Gate Level Test Results

# Demo Slides (Cell Usage)



### Routing stats

Utilisation (%)	Wire length (um)
9.5	2888

### Cell usage by Category

Category	Cells	Count
Fill	<a href="#">decap_fill</a>	1407
Tap	<a href="#">tapvpwrvrgnd</a>	225
Combo Logic	<a href="#">a21o</a> <a href="#">o211a</a> <a href="#">a31o</a> <a href="#">a21oi</a> <a href="#">a311o</a> <a href="#">a211o</a> <a href="#">a32o</a> <a href="#">o21ai</a> <a href="#">a31oi</a> <a href="#">o31a</a> <a href="#">o21a</a> <a href="#">o221a</a> <a href="#">or3b</a> <a href="#">a221o</a>	35
Buffer	<a href="#">buf</a> <a href="#">clkbuf</a>	25
Flip Flops	<a href="#">dfxtp</a>	24
OR	<a href="#">or4</a> <a href="#">xor2</a> <a href="#">or2</a> <a href="#">or3</a>	18
Misc	<a href="#">dlymetal6s2s</a> <a href="#">dlygate4sd3</a> <a href="#">conb</a>	18
AND	<a href="#">and2</a> <a href="#">and3</a>	17
NOR	<a href="#">nor2</a> <a href="#">xnor2</a>	14
NAND	<a href="#">nand2</a>	8
Multiplexer	<a href="#">mux2</a>	8
Inverter	<a href="#">inv</a>	2

**169 total cells (excluding fill and tap cells)**

Job summary generated at run-time

Summary

Jobs

- gds
- precheck
- gl\_test
- viewer



Run details

- Usage
- Workflow file

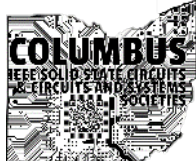
GDS Action

Home

Actions Projects Security Insights Settings



# Demo Slides (2D Render)



The screenshot displays a software interface for circuit design. On the left is a sidebar with a 'Summary' tab and a list of jobs: 'gds', 'precheck', 'gl\_test', and 'viewer', all with green checkmarks. Below the jobs are 'Run details' including 'Usage' and 'Workflow file'. The main workspace is divided into two sections: '3D Viewer' at the top and '2D Preview' below it. The '2D Preview' shows a detailed, colorful 2D layout of a circuit board with various components and traces. A top navigation bar contains 'Actions', 'Projects', 'Security', 'Insights', and 'Settings'. A blue arrow points to the 'Actions' menu. Two blue buttons, 'GDS Action' and 'Home', are overlaid on the bottom left of the interface.

# Demo Slides (3D Viewer)

**KEYS**  
1: Hide Fill, Decap, Tap cells  
2: Hide top cell geometry  
3: Isolate mouse over cell  
Mouse over: FILLER\_0\_19\_205 (sky130\_ef\_sc\_hd\_decap\_12)

**Controls**

**View Settings**

- toggleFillerCells
- toggleTopCellGeometry
- nwell
- diff
- poly
- licon
- li1
- mcon
- met1
- substrate
- via
- met2
- via2
- met3
- via3
- met4

**> Stats**

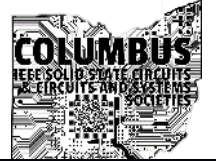
GitHub Actions

GDS Action

Home

2D Render

# Demo Slides (Settings)



Code Issues Pull requests **Actions** Projects Security Insights **Settings**

General

Access

Collaborators

Moderation options

Code and automation

Branches

Tags

Rules

Actions

Webhooks

Environments

Codespaces

**Pages**

Security

Code security and analysis

Deploy keys

Secrets and variables

Integrations

GitHub Apps

Email notifications

## GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

Your site is live at [https://sellicott.github.io/sellicott\\_fib\\_seq/](https://sellicott.github.io/sellicott_fib_seq/)  
Last deployed by sellicott 3 hours ago [Visit site](#)

### Build and deployment

Source

GitHub Actions

Your site was last deployed to the `github-pages` environment by the `gds` workflow.  
[Learn more about deploying to GitHub Pages using custom workflows](#)

### Custom domain

Custom domain

Custom domains allow you to serve your site from a domain other than `sellicott.github.io`. [Learn more about configuring custom domains.](#)

[Save](#) [Remove](#)

**Enforce HTTPS**  
— Required for your site because you are using the default domain (`sellicott.github.io`)

HTTPS provides a layer of encryption that prevents others from snooping on or tampering with traffic to your site. When HTTPS is enforced, your site will only be served over HTTPS. [Learn more about securing your GitHub Pages site with HTTPS.](#)

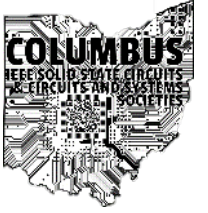
### Visibility

GITHUB ENTERPRISE

With a GitHub Enterprise account, you can restrict access to your GitHub Pages site by publishing it privately. A privately published site can only be accessed by people with read access to the repository the site is published from. You can use

Pages Tab

Home



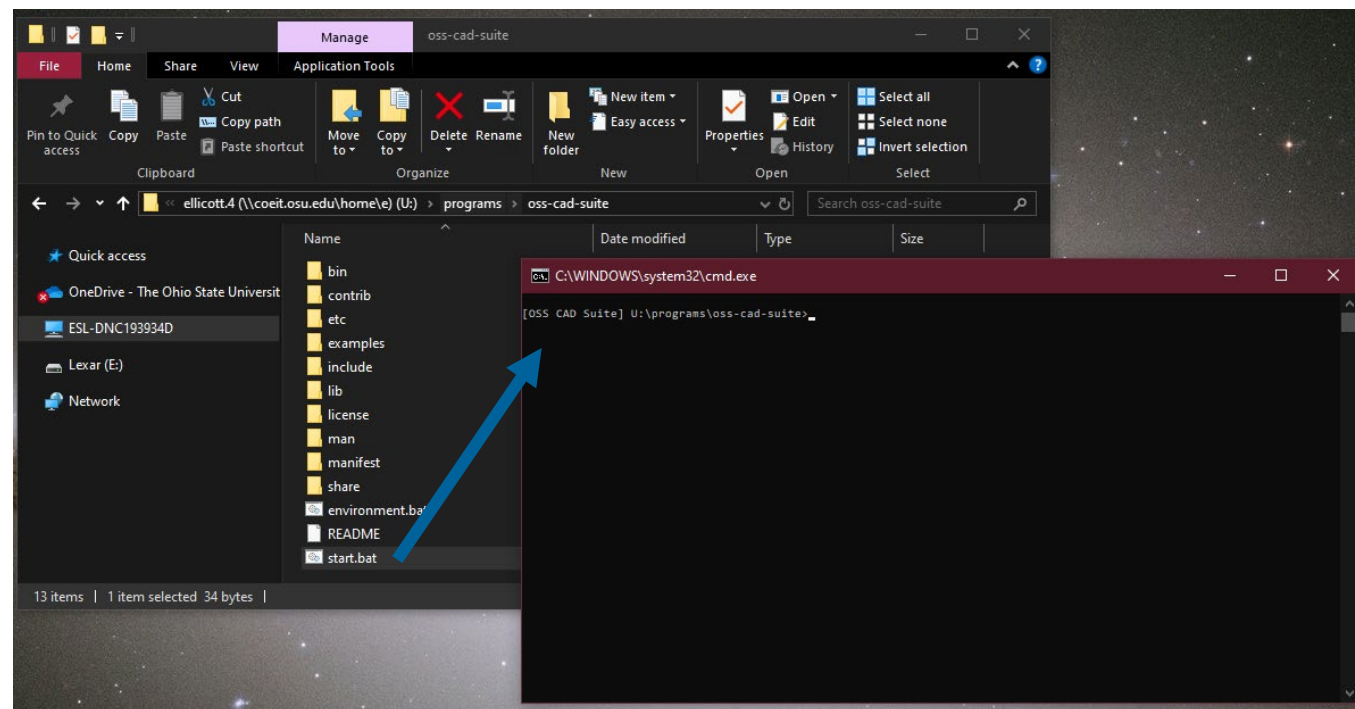
# Running Locally

- Tiny Tapeout allows for cloud based development
  - Full flow in GitHub actions
  - Only need GTKWave (to view simulation results)
  - Slow iteration loop
- We *can* run the whole OpenLane flow locally...
  - Large installation size (~10Gb)
  - Slightly involved installation process (on Windows)
- Run simulations locally
  - Develop Verilog locally
  - Smaller (~1.5Gb)
  - Simulate behavioral HDL → Basic functionality
  - Harden with Tiny Tapeout flow (GitHub actions)



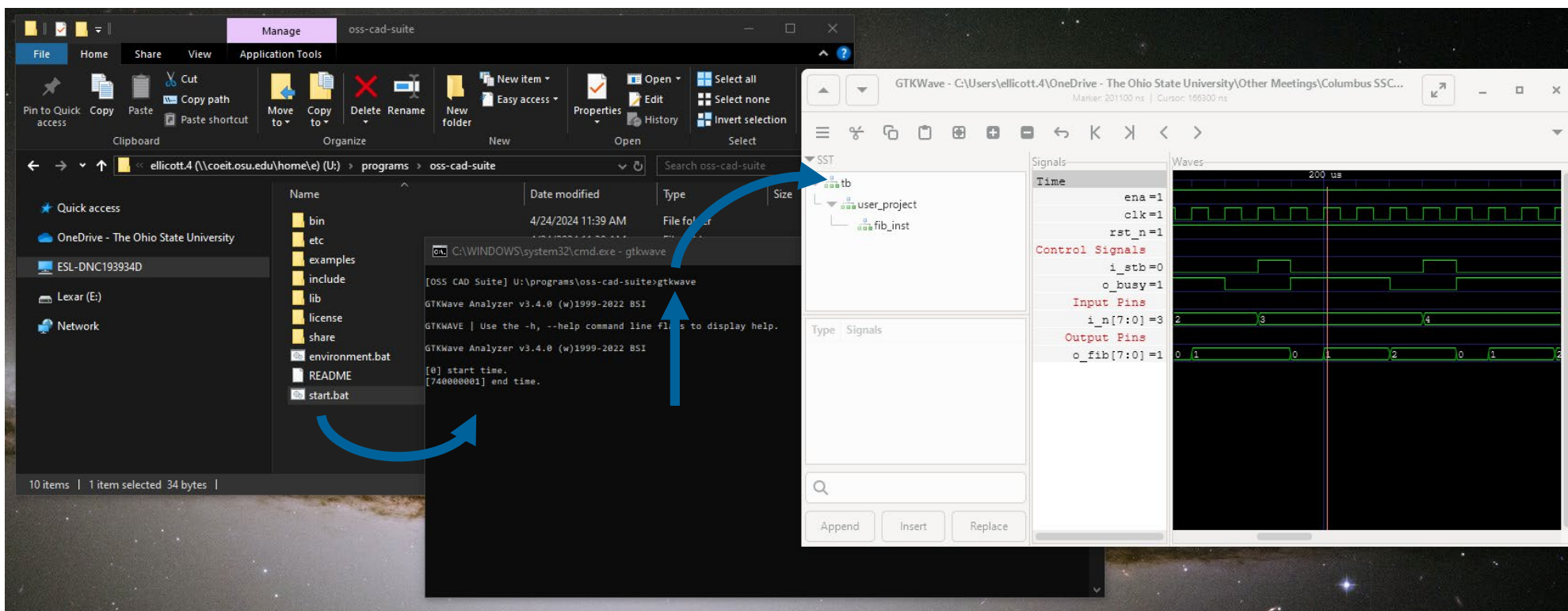
# Installing Tools

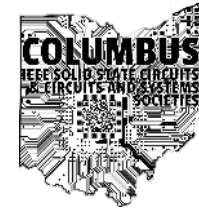
- Simulation Tools
  - <https://github.com/YosysHQ/oss-cad-suite-build>
  - Extract file to a path without spaces
  - GTKWave, Icarus Verilog, Yosis, Etc
- Start by running *start.bat*
  - Should see '[OSS Cad Suite]'
  - Move around with 'cd'
  - Drag and drop folders into terminal
    - Puts folder path into terminal
- Other Tools
  - Git: <https://git-scm.com/downloads>
  - VS Code: <https://code.visualstudio.com/>



# Local Simulation: Running Tools

- In [OSS CAD Suite] terminal!
  - Type the tool you want to use
  - Use 'gtkwave' for GTKWave, etc





# Local Simulation: Running Simulator

- Using Icarus Verilog for Simulation
- In [OSS CAD Suite] terminal move to folder with Verilog code
- Elaborate → Simulate design
  - `iverilog -o tb.vvp -DICARUS <Verilog file 1> <Verilog file 2> ...`
  - `vvp tb.vvp`
- Generates `tb.vcd` for GTKWave
- More Information:
  - [https://steveicarus.github.io/iverilog/usage/getting\\_started.html](https://steveicarus.github.io/iverilog/usage/getting_started.html)

```
Command Prompt
[OSS CAD Suite] C:\Users\ellicott.4\OneDrive - The Ohio State University\Other Meetings\Columbus SSCS\Tiny Tapeout\sellicott_fib_seq\test>
[OSS CAD Suite] C:\Users\ellicott.4\OneDrive - The Ohio State University\Other Meetings\Columbus SSCS\Tiny Tapeout\sellicott_fib_seq\test>iverilog -o tb.vvp -DICARUS tb.v ../src/tt_um_sellicott_fib_seq.v ../src/fib.v
[OSS CAD Suite] C:\Users\ellicott.4\OneDrive - The Ohio State University\Other Meetings\Columbus SSCS\Tiny Tapeout\sellicott_fib_seq\test>vvp tb.vvp
VCD info: dumpfile tb.vcd opened for output.
Start
Reset
Test project behavior
Test n=
1
hw_fib: 0, sw fib: 1
Test n=
2
hw_fib: 0, sw fib: 1
Test n=
3
hw_fib: 0, sw fib: 2
Test n=
4
hw_fib: 0, sw fib: 3
Test n=
5
hw_fib: 0, sw fib: 5
Test n=
6
hw_fib: 0, sw fib: 8
Test n=
7
hw_fib: 0, sw fib: 13
Test n=
8
hw_fib: 0, sw fib: 21
Test n=
9
hw_fib: 0, sw fib: 34
Closing
Test n=
10
hw_fib: 34, sw fib: 34
tb.v:92: $finish called at 1890 (1ns)
[OSS CAD Suite] C:\Users\ellicott.4\OneDrive - The Ohio State University\Other Meetings\Columbus SSCS\Tiny Tapeout\sellicott_fib_seq\test>
```

# Local Simulation: Testbench

*tb.v* Testbench

```

module tb ();

// Dump the signals to a VCD file. You can view it with gtkwave.
initial begin
    $dumpfile("tb.vcd");
    $dumpvars(0, tb);
end
end
    
```

- Define what file to write simulation to
  - \$dumpfile sets the filename
- Select what data to save
  - \$dumpvars selects what signals to save
- See [here](#)

- Define variables that we can interact with
  - Registers for module inputs
  - Wires for module outputs

```

// Wire up the inputs and outputs:
reg clk;
reg rst_n;
reg ena;
reg [7:0] ui_in;
reg [7:0] uio_in;
wire [7:0] uo_out;
wire [7:0] uio_out;
wire [7:0] uio_oe;
    
```

- Instantiate our Tiny Tapeout module
  - Hook up power and gnd if doing a gate level test (Run after synthesis)
  - Other ports hooked up to variables

```

// Replace tt_um_example with your module name:
tt_um_sellicott_fib_seq user_project (
    // Include power ports for the Gate Level test:
    `ifdef GL_TEST
        .VPWR(1'b1),
        .VGND(1'b0),
    `endif

    .ui_in (ui_in), // Dedicated inputs
    .uo_out (uo_out), // Dedicated outputs
    .uio_in (uio_in), // IOs: Input path
    .uio_out (uio_out), // IOs: Output path
    .uio_oe (uio_oe), // IOs: Enable path (active high: 0=input, 1=output)
    .ena (ena), // enable - goes high when design is selected
    .clk (clk), // clock
    .rst_n (rst_n) // not reset
);
    
```

# Local Simulation: Testbench

*tb.v* Testbench

```

// Testbench code for simulating with Icarus Verilog instead of ModelSim
`ifdef ICARUS

localparam CLK_PERIOD = 20; // 20ns -> run at 50MHz
localparam CLK_HALF_PERIOD = CLK_PERIOD / 2;

// toggle clock every half period
always #(CLK_HALF_PERIOD) begin
    clk = ~clk;
end

initial begin: fib_test
    // reset the system
    initialize();

    $display("Test project behavior");
    // test the first 10 fibonacci sequence numbers
    test_n_fib(10);

    // finish the simulation
    close();
end
    
```

- Only run this code if ICARUS flag is set
  - Avoid breaking GitHub Action sims

- Define clock to run at 50MHz

- Run tests
- Three segments
  - Initialization
  - Tests (first 10 fibonacci numbers)
  - Closing simulation

# Local Simulation: Testbench

```

// task to initialize all of the top level registers and reset the system
task initilize();
begin: initilize
    clk = 1'h0;
    $display("Start");
    #2;
    $display("Reset");
    ena = 1'h1;
    ui_in = 8'h0;
    uio_in = 8'h0;
    rst_n = 1'h0;

    // wait 10 clock cycles
    repeat(10) @(posedge clk);
    rst_n = 1'h1;
end
endtask

```

- Puts the system into a defined state
- Sets the values for all the input registers

- Reset the system

```

task close();
begin: close
    $display("Closing");
    // wait 10 clock cycles to give us time to read last value
    repeat(10) @(posedge clk);
    $finish;
end
endtask

```

- Delay a bit before ending the simulation
- Makes it easier to read waveforms

\$finish closes the simulator

# Local Simulation: Testbench

```

task test_n_fib(
  input integer n
);
begin: test_n_fib_block
  integer fib_idx;
  fib_idx = 0;
  for (fib_idx = 0; fib_idx < n; fib_idx++)
  begin: fib_test_iter
    integer fib_calc;
    integer fib_hw;
    fib_hw = 0;
    // calculate the current fibonacci sequence value
    fib_calc = calc_fib(fib_idx);
    run_fib_seq(fib_idx);
    fib_hw = u0_out[7:0];
    $strobe("Test n=%d", fib_idx);
    $strobe("hw fib: %d, sw fib: %d", fib_hw, fib_calc);
    // exit if the values don't match
    if(fib_hw != fib_calc) close();
  end
end
endtask

```

- For loop to test the nth Fibonacci sequence number
- Weird variable definition to keep Verilog happy

- Calculate the Fibonacci number using a "software" method
- Calculate again using module

- Exit simulation if values don't match

# Local Simulation: Testbench

```

// make a task to read the fibonacci value
task run_fib_seq(
  input integer n
);
begin
  // wait a clock cycle and set the strobe signal
  @(posedge clk);
  uio_in[0] = 1'h1;
  ui_in[7:0] = n[7:0];
  // wait a clock cycle and clear the strobe signal
  @(posedge clk);
  uio_in[0] = 1'h0;
  @(posedge clk);
  // wait until the busy signal is low, each time wait another clock cycle
  for(; uio_out[1]; ) @(posedge clk);
end
endtask

```

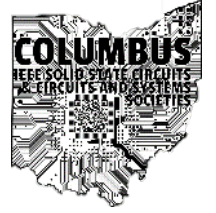
- Assert the i\_stb pin and set what n to test

- Wait a clock cycle and clear the i\_stb pin

- Wait at least one clock cycle
- Wait until o\_busy is cleared



# Local Simulation: Testbench



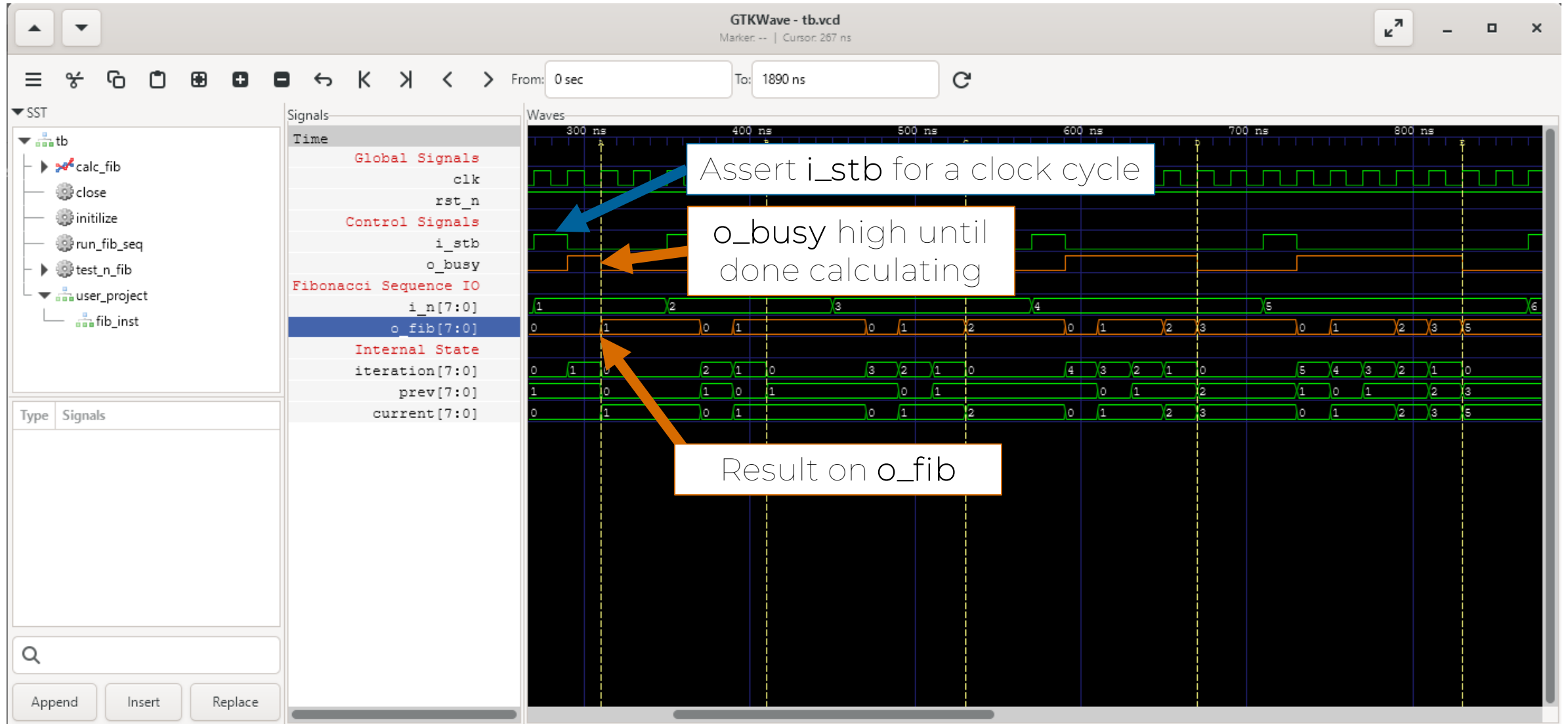
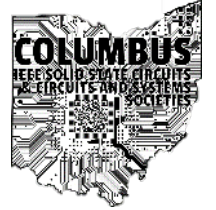
```
// iteratively calculate the nth fibonacci sequence number
function integer calc_fib(
    input integer n
);
begin: calc_fib_block
    integer a;
    integer b;
    integer c;
    integer i;

    a = 0;
    b = 1;
    c = 0;

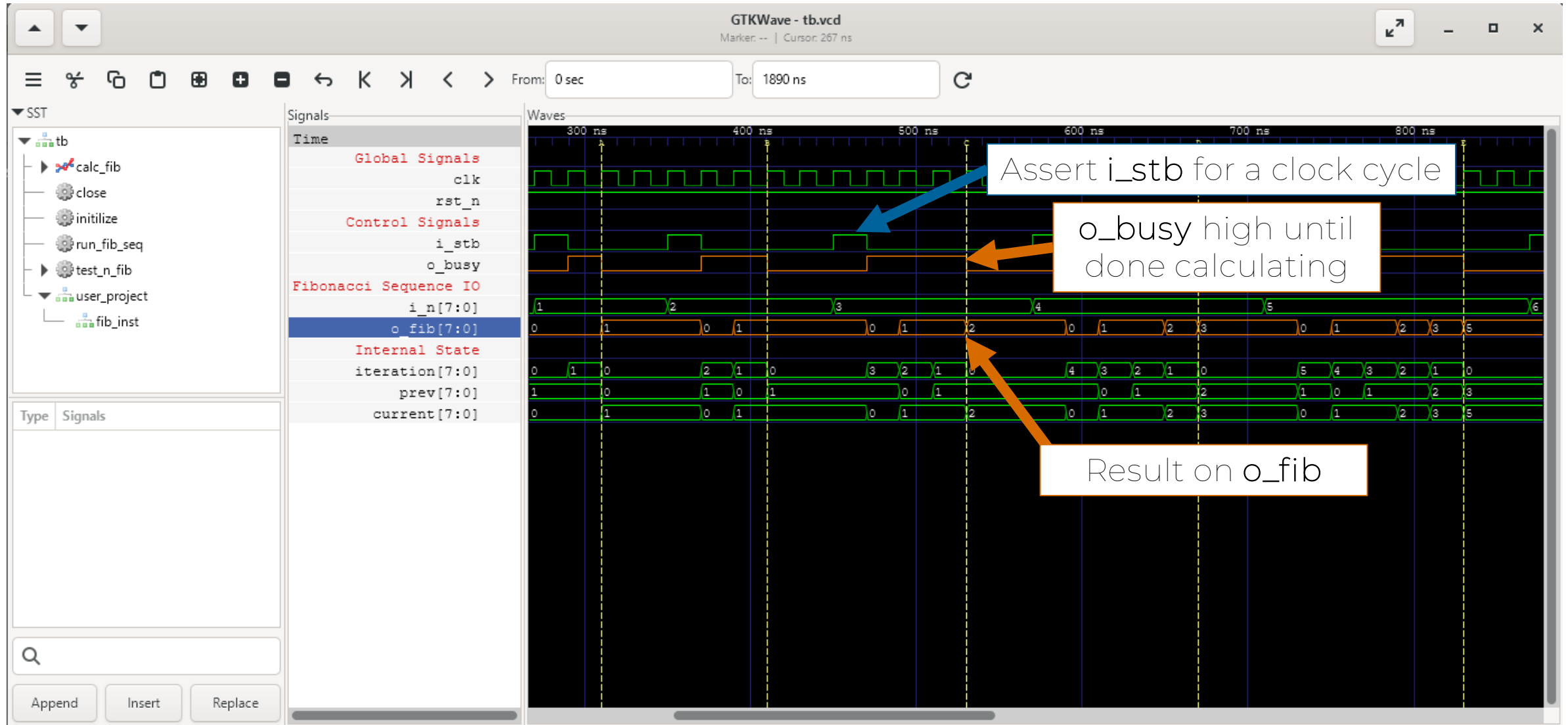
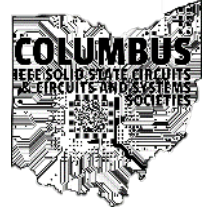
    if (n == 0) begin
        calc_fib = a;
    end
    else begin
        for (i = 1; i < n; i++) begin
            c = a + b;
            a = b;
            b = c;
        end
        calc_fib = b;
    end
end
endfunction
```

- Calculate the  $n$ th Fibonacci number
- Do the whole calculation in Verilog
- Can write basically like software

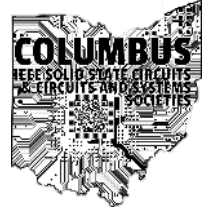
# Expected Results



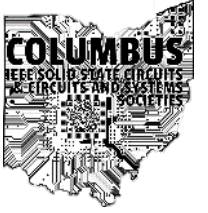
# Expected Results



# Next Steps



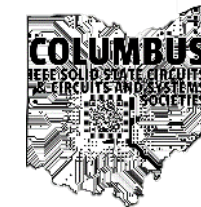
- Install the simulation tools locally
- Follow some [digital design tutorials](#)
- Follow some Verilog tutorials
  - [NAND Land Verilog tutorial](#)
  - [Reference Designer Verilog tutorial](#)
- Test your own designs



# Step by Step Instructions

- Goal: Recreate Fibonacci Sequence Generator
- Install Tools
  - Git: <https://git-scm.com/downloads>
  - VS Code: <https://code.visualstudio.com/>
  - OSS-CAD-Suite: <https://github.com/YosysHQ/oss-cad-suite-build>
- Setup Tiny Tapeout Project
  - Make your own project based on [the template](#)
  - [Enable GitHub Pages](#) (set to GitHub Actions)
  - Update *info.yaml* file
- Write Verilog Code
  - Implement Fibonacci Sequence state machine
  - Write testbench
  - To learn Verilog syntax: [Verilog tutorials](#)
- Simulate Design
  - Use [Icarus Verilog](#) to simulate testbench
  - Use [GTKWave](#) or [VaporView](#) to view output waveforms
- Push Code to GitHub

# Where To Get Help



Here!

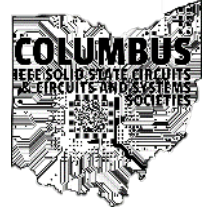


- Tiny Tapeout Discord
  - Lots of people with deeper knowledge than me
  - I'm there too (@sellicott)
- Learning Resources
  - [https://tinytapeout.com/digital\\_design/](https://tinytapeout.com/digital_design/)
  - <https://tinytapeout.com/hdl/>
  - <https://r2.ieee.org/columbus-ssccas/resources/>
- Questions about this workshop
  - Chapter: columbus.sscs.cas@gmail.com



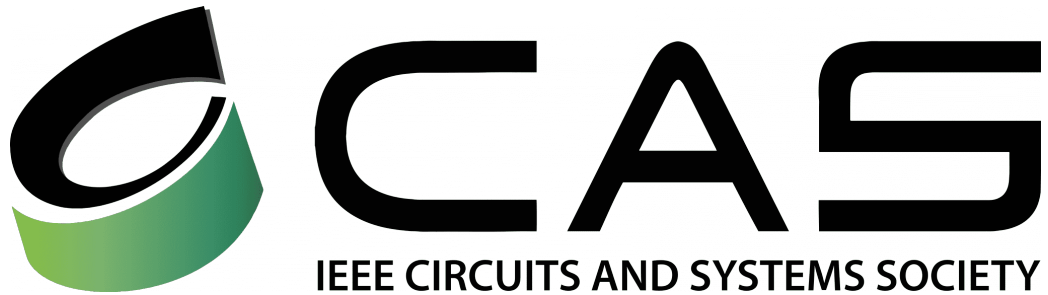
Tiny Tapeout Discord  
<https://discord.gg/BspcX9SB2h>

# Acknowledgements



Matt Venn

- [Original workshop slides](#)
- Advice on running a workshop



**SenseICs**

Everyone who worked on Tiny Tapeout!

**efabless**.com

<https://tinytapeout.com/credits/>

# Scan this QR code to fill out the Form to participate!



Sponsorship Application

<https://forms.gle/ypWKDA4zrj8zKAR9A>

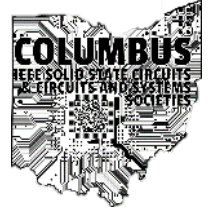


Event Feedback Form

<https://forms.gle/g5SsnPgFxGNzazYM8>



# Extra Slide



This page unintentionally left blank