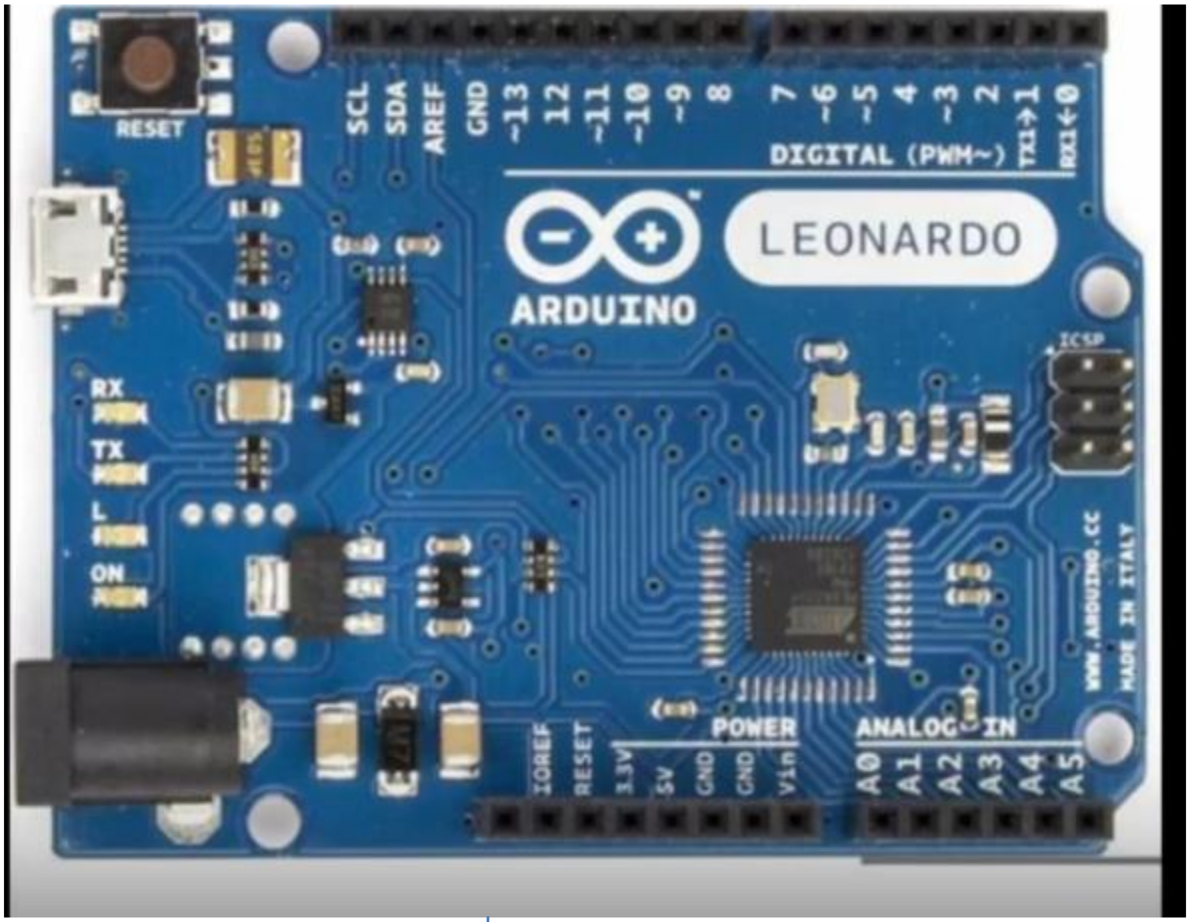
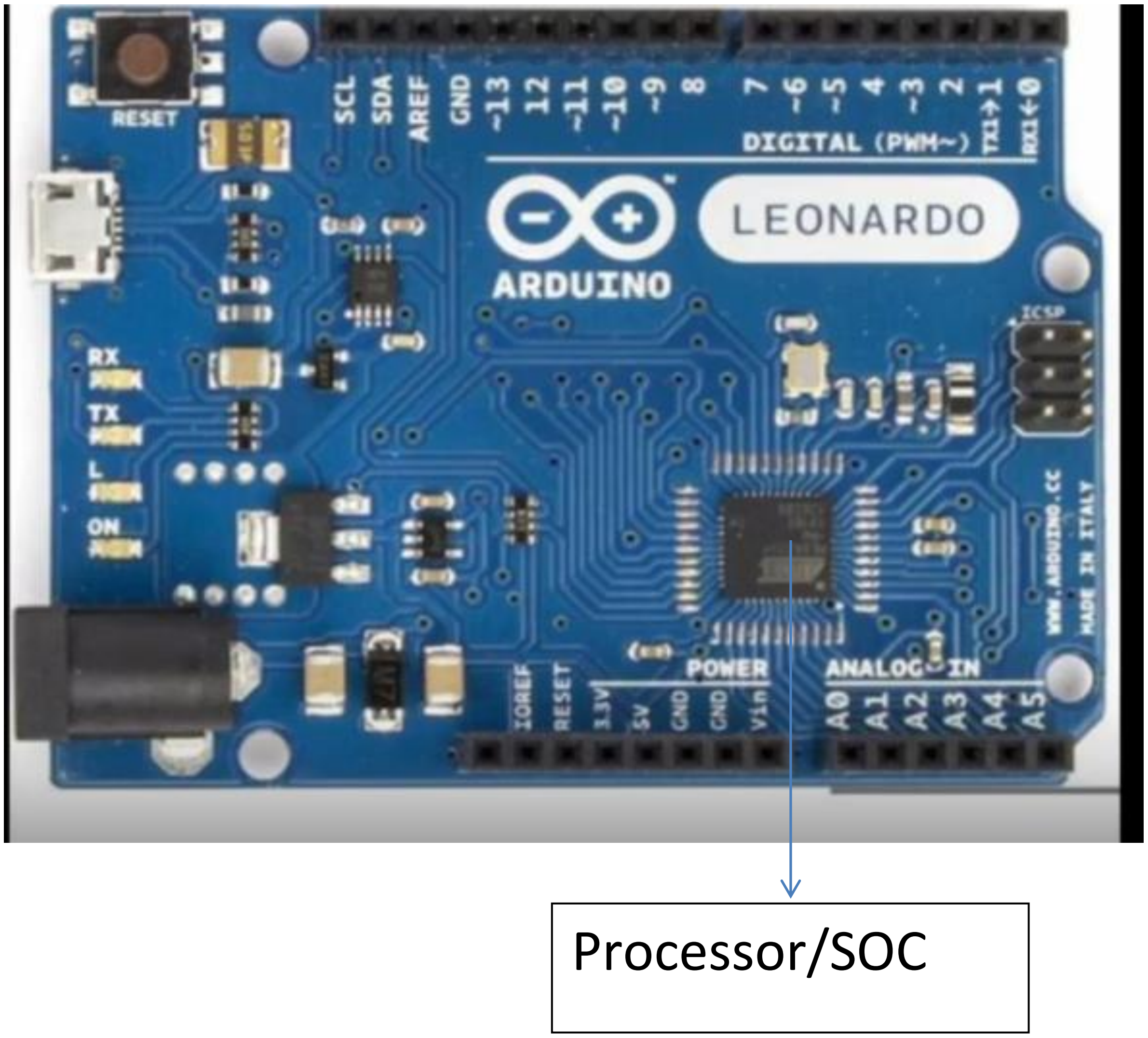
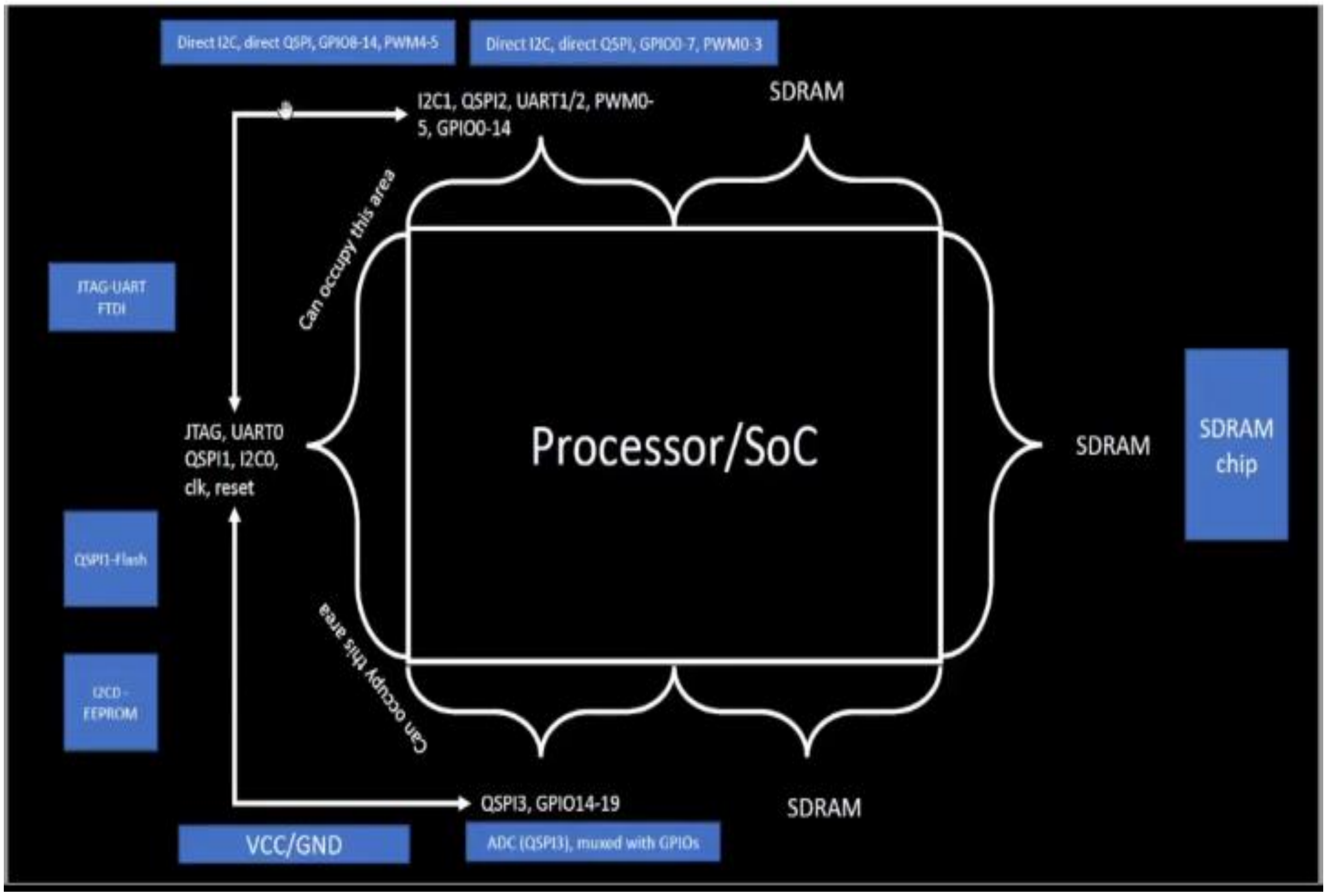
Day 1

Arduino board



This is a typical electric board.


RESET
SCL
SDA
AREF
GND
~13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX1→1
RX1←0
DIGITAL (PWM~)
ARDUINO
LEONARDO
ICSP
RX
TX
L
ON
WWW.ARDUINO.CC
MADE IN ITALY
IOREF
RESET
3.3V
5V
GND
GND
Vin
POWER
ANALOG IN
A0
A1
A2
A3
A4
A5
Processor/SOC

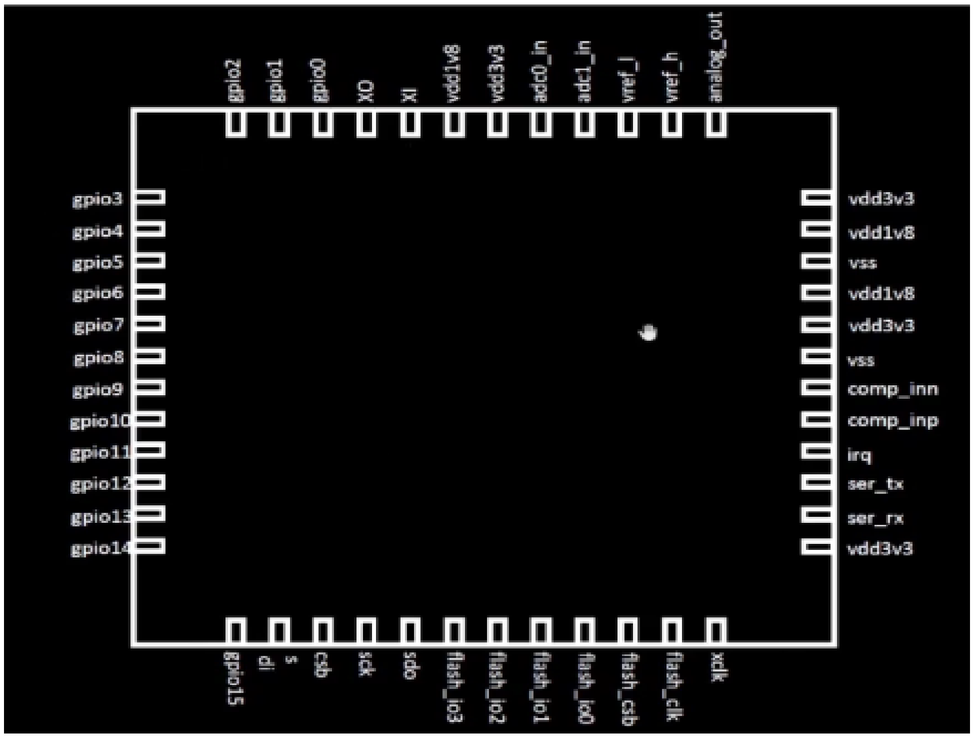

Direct I2C, direct QSPI, GPIO8-14, PWM4-5
Direct I2C, direct QSPI, GPIO0-7, PWM0-3
I2C1, QSPI2, UART1/2, PWM0-5, GPIO0-14
SDRAM
Can occupy this area
JTAG-UART FTDI
JTAG, UART0 QSPI1, I2C0, clk, reset
Processor/SoC
SDRAM
SDRAM chip
QSPI1-Flash
I2C0 - EEPROM
Can occupy this area
QSPI3, GPIO14-19
SDRAM
VCC/GND
ADC (QSPI3), muxed with GPIOs

What is a package?

A package is a collection of files, including code and configuration data that allows to use a specific type of Arduino board within the IDE by providing the necessary information for compiling and uploading code to that board.

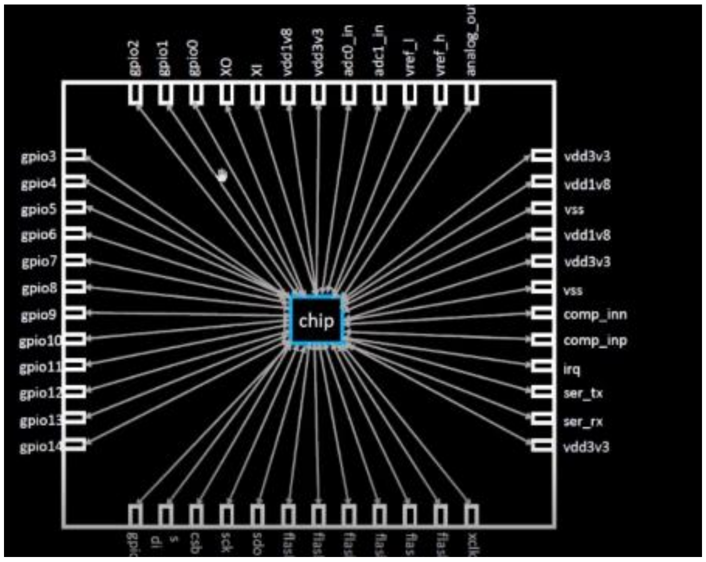
Examples:QFN-48

Quad flat – No leads



The chip will be somewhere in the center of the package.

And the package is connected to the chip like below.



.

We are connecting all the pins to the boundaries of the chip.IN this way we are able to transfer all the signals from the external world to the interiors of the chip.

When we open the chip. It will have many components such as:

1) Pads

2) Die

3) Core

4) IP’s

PADS:

Pads are something used for sending signals inside the chip.

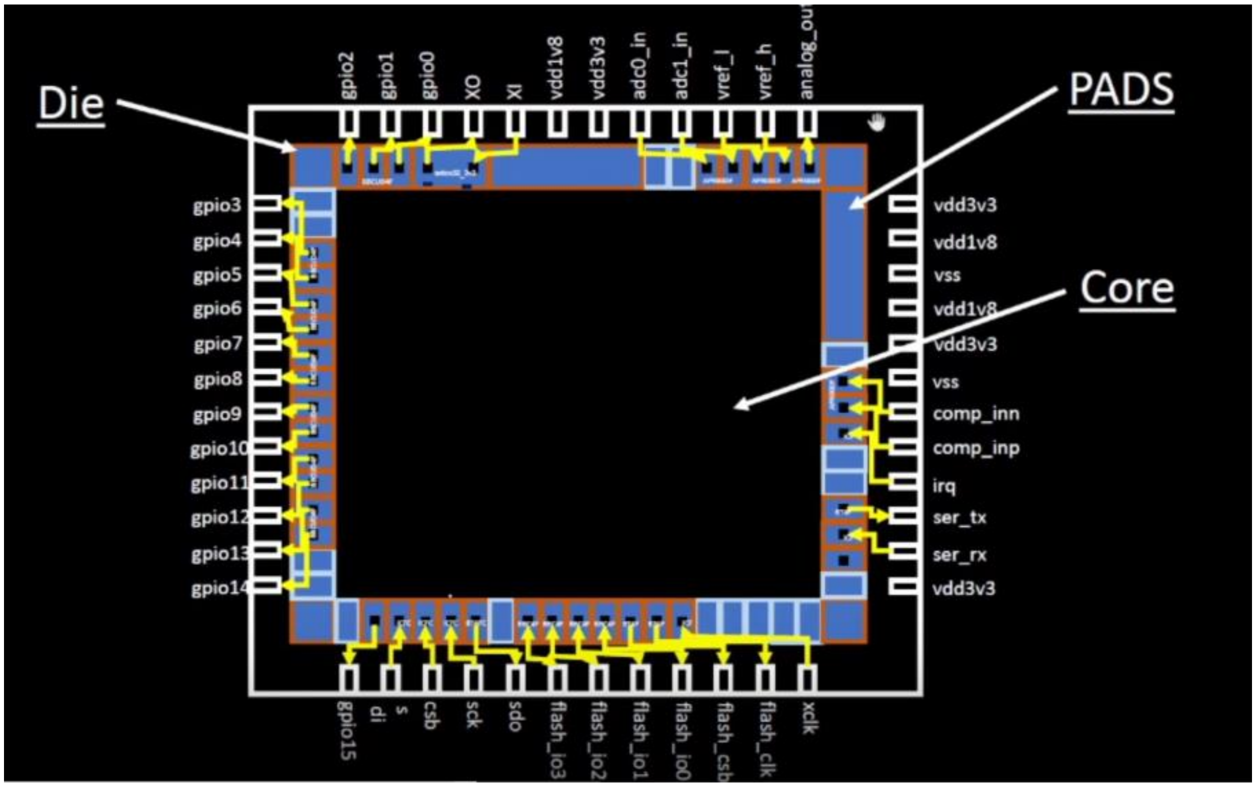
Any signal which can go inside the chip can go outside the chip through pads.

Core:

This is called the core of the chip. This is the place where it contains all the digital logic such as “and” gates, “or” gates etc.

Die:

Die is the basically the size of the entire chip.



The core consists of:

1) RISC V SoC

- Gpio bank

2) SRAM

3) PLL

4) Adc0

5) Adc1

6) Dac

7)SPI

Foundry IP’s

PLL, SRAM, Adc1, Adc0 and Dac are called the foundry ip’s.

The word Foundry is a typical word used in chip design. Mobiles and laptop are very much dependent on Foundry IP’s.

Foundry is a big factory which has got some machines.

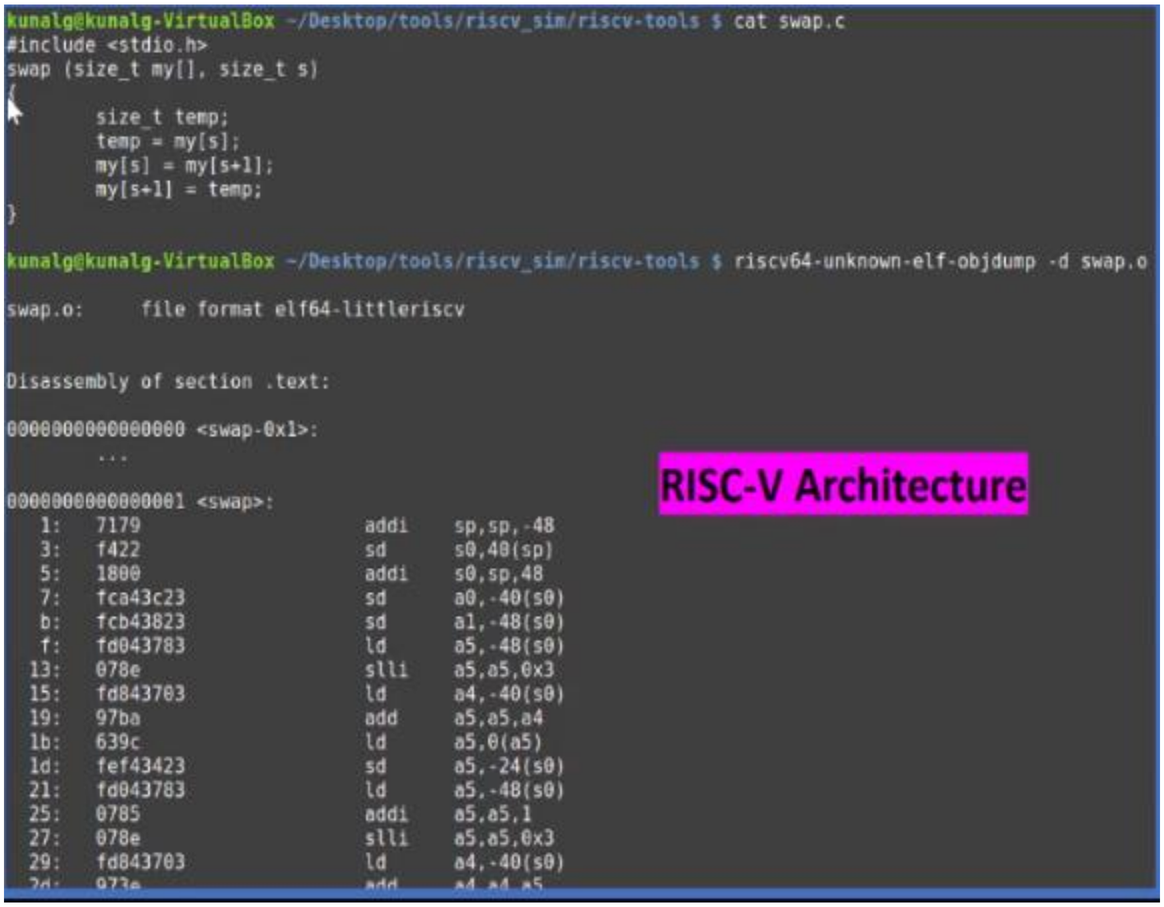
Macros

RISC-V and SPI are called the Macros. Macros are something which is pure digital logic.

What is RISC-V?

RISC-V is also called instruction Set Architecture (ISA).This is language of a computer. This is the way we talk to computer.

For example:



```
kunalg@kunalg-VirtualBox ~/Desktop/tools/riscv_sim/riscv-tools $ cat swap.c
#include <stdio.h>
swap (size_t my[], size_t s)
{
        size_t temp;
        temp = my[s];
        my[s] = my[s+1];
        my[s+1] = temp;
}

kunalg@kunalg-VirtualBox ~/Desktop/tools/riscv_sim/riscv-tools $ riscv64-unknown-elf-objdump -d swap.o

swap.o:     file format elf64-littleriscv


Disassembly of section .text:

0000000000000000 <swap-0x1>:
        ...

0000000000000001 <swap>:
   1:   7179                    addi    sp,sp,-48
   3:   f422                    sd      s0,40(sp)
   5:   1800                    addi    s0,sp,48
   7:   fca43c23                sd      a0,-40(s0)
   b:   fcb43823                sd      a1,-48(s0)
   f:   fd043783                ld      a5,-48(s0)
  13:   078e                    slli    a5,a5,0x3
  15:   fd843703                ld      a4,-40(s0)
  19:   97ba                    add     a5,a5,a4
  1b:   639c                    ld      a5,0(a5)
  1d:   fef43423                sd      a5,-24(s0)
  21:   fd043783                ld      a5,-48(s0)
  25:   0785                    addi    a5,a5,1
  27:   078e                    slli    a5,a5,0x3
  29:   fd843703                ld      a4,-40(s0)
  2d:   973e                    add     a4,a4,a5
```

Now we have a C program. This needs to be done on a hardware which has got a layout q(interior of the chip in the laptop).

So first the C program is compiled into the assembly language which is nothing but the RISC-V assembly language program.

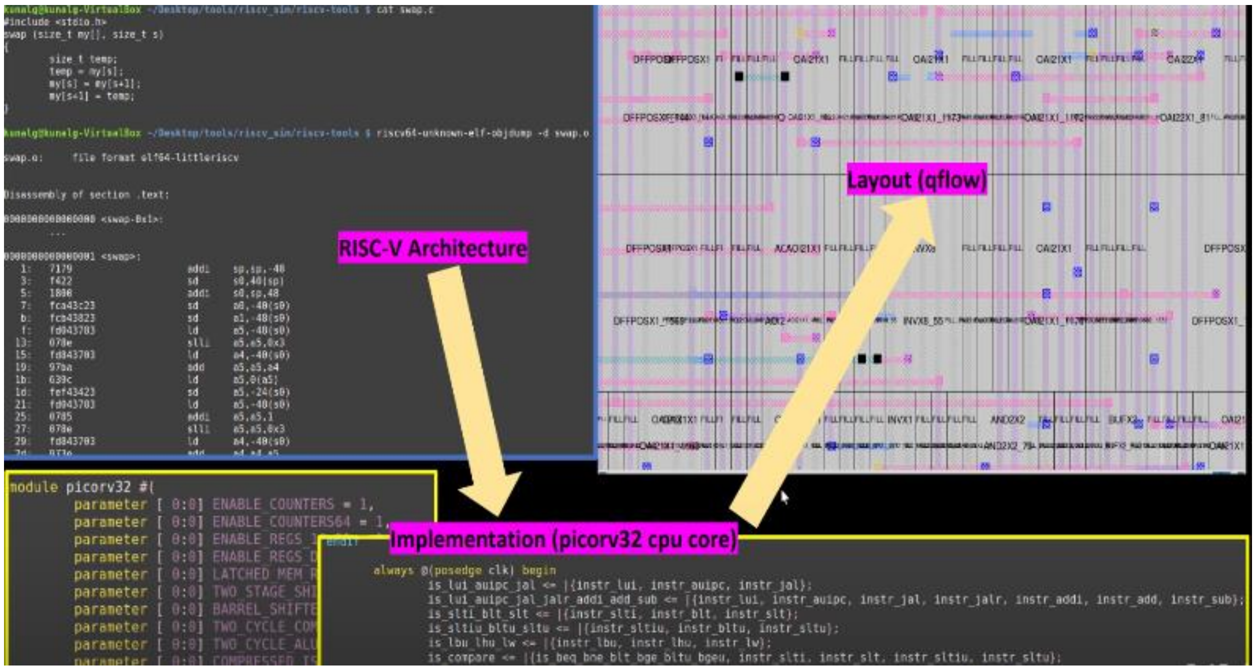
Then the assembly language is converted into the machine language program(binary language program which is nothing but the electronic signals logic 1’s and 0’s which are understood by the hardware of the computer.

It’s currently present in the hexadecimal format. This needs to be converted to a binary format and then finally this bits executed in the particular layout and we get the required output.

There is other interface that needs to be present between RISCV architecture and layout. That is nothing but a hardware description language. We need to implement

The RISC-V specifications using some RTL.For example we will take the RTL as picorv-32 CPU core and from RTL it will be implemented to Layout.

RISC-V Architecture $\rightarrow$ implementation (picorv-32 CPU core) $\rightarrow$ Layout



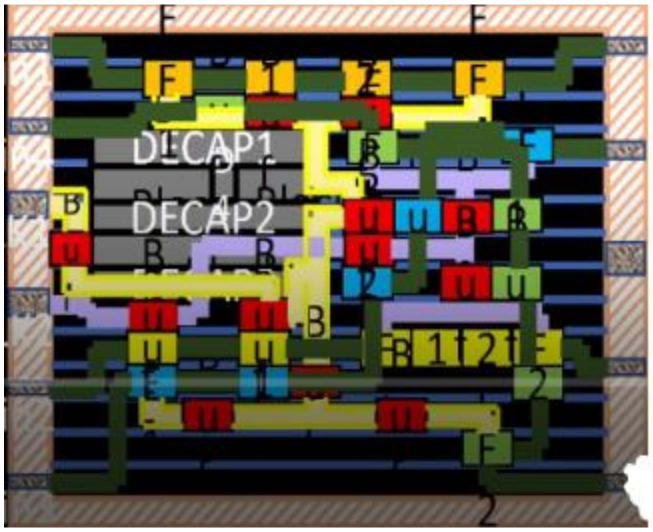
```
kunalg@kunalg-VirtualBox ~/Desktop/tools/riscv_sim/riscv-tools $ cat swap.c
#include <stdio.h>
swap (size_t my[], size_t s)
{
        size_t temp;
        temp = my[s];
        my[s] = my[s+1];
        my[s+1] = temp;
}
kunalg@kunalg-VirtualBox ~/Desktop/tools/riscv_sim/riscv-tools $ riscv64-unknown-elf-objdump -d swap.o

swap.o:     file format elf64-littleriscv


Disassembly of section .text:

0000000000000000 <swap-0x1>:
        ...

0000000000000001 <swap>:
   1:   7179                    addi    sp,sp,-48
   3:   f422                    sd      s0,40(sp)
   5:   1800                    addi    s0,sp,48
   7:   fca43c23                sd      a0,-40(s0)
   b:   fcb43823                sd      a1,-48(s0)
   f:   fd043703                ld      a5,-40(s0)
  13:   078e                    slli    a5,a5,0x3
  15:   fd843703                ld      a4,-40(s0)
  19:   97ba                    add     a5,a5,a4
  1b:   639c                    ld      a5,0(a5)
  1d:   fef43423                sd      a5,-24(s0)
  21:   fd043703                ld      a5,-40(s0)
  25:   0785                    addi    a5,a5,1
  27:   078e                    slli    a5,a5,0x3
  29:   fd843703                ld      a4,-40(s0)
```

```
module picorv32 #(
        parameter [ 0:0] ENABLE_COUNTERS = 1,
        parameter [ 0:0] ENABLE_COUNTERS64 = 1,
        parameter [ 0:0] ENABLE_REGS_
        parameter [ 0:0] ENABLE_REGS_D
        parameter [ 0:0] LATCHED_MEM_R
        parameter [ 0:0] TWO_STAGE_SHI
        parameter [ 0:0] BARREL_SHIFTE
        parameter [ 0:0] TWO_CYCLE_COM
        parameter [ 0:0] TWO_CYCLE_ALU
```

```
endif

        always @(posedge clk) begin
                is_lui_auipc_jal <= |{instr_lui, instr_auipc, instr_jal};
                is_lui_auipc_jal_jalr_addi_add_sub <= |{instr_lui, instr_auipc, instr_jal, instr_jalr, instr_addi, instr_add, instr_sub};
                is_slti_blt_slt <= |{instr_slti, instr_blt, instr_slt};
                is_sltiu_bltu_sltu <= |{instr_sltiu, instr_bltu, instr_sltu};
                is_lbu_lhu_lw <= |{instr_lbu, instr_lhu, instr_lw};
                is_compare <= |{is_beq_bne_blt_bge_bltu_bgeu, instr_slti, instr_slt, instr_sltiu, instr_sltu};
```

These are some apps we use in our day to day life.

These apps actually run on the laptop or this is a representation of chip present in out laptop.



The apps work on this particular hardware

How does this actually work?

First the application software enters into a block called system software and the system software converts the application program into a binary language.

The major components of system software:

- OS(Operating system)
- Compiler
- Assembler

The next job is of the OS. Its functions are:

- Handle IO operations
- Allocate memory
- Low level system functions

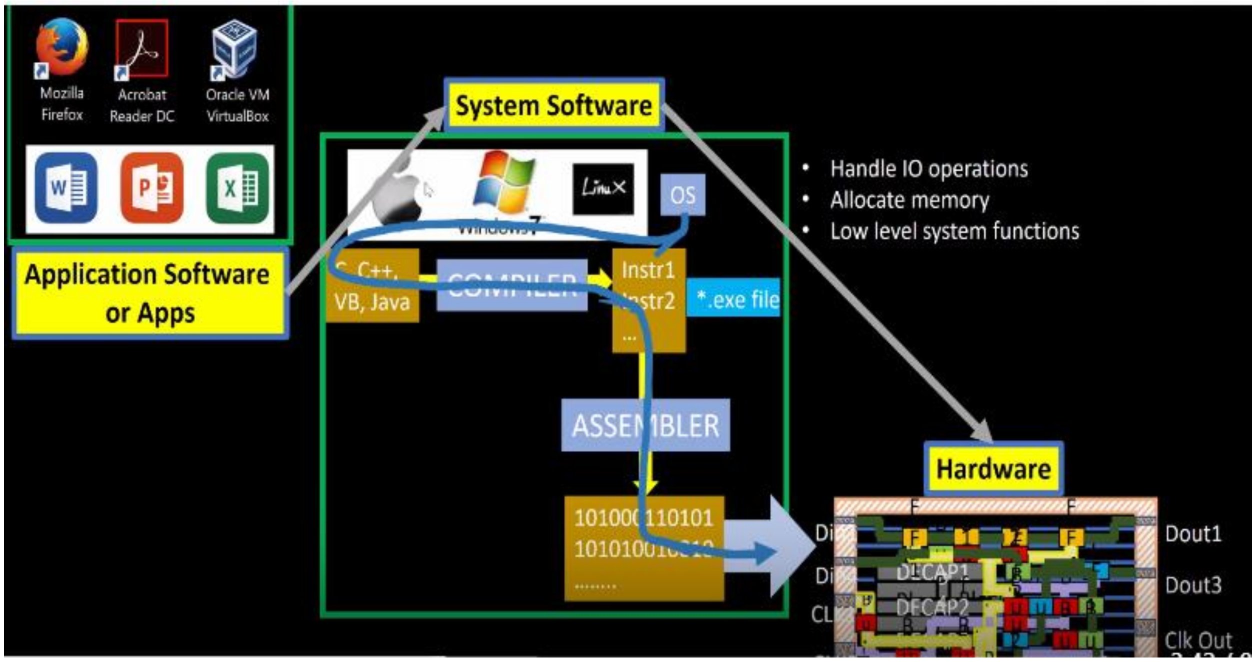
The other function of OS is to take the particular app and convert it into assembly language program and then finally into binary language program.

The output of OS is nothing but small functions in the C, C++ or Java. These are taken by the respected compiler and the compiler converts it into instructions. The syntax of the instruction depends on what type of hardware it is.The output of compiler depends on type of hardware.

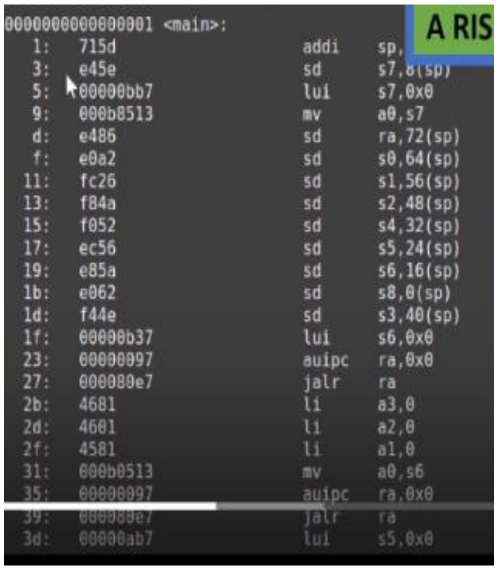
The job of the assembler is to take the output of the instructions and convert it into binary number (machine language program).

The binary language is then feed into the hardware. Accordingly it will generate the output

If the hardware is RISC-V then the output of compiler will be RISC-V instructions .

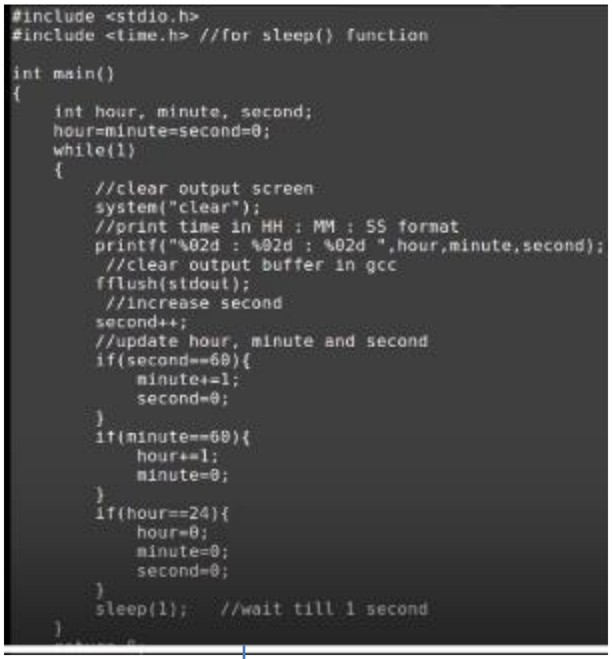


RISC-V instructions



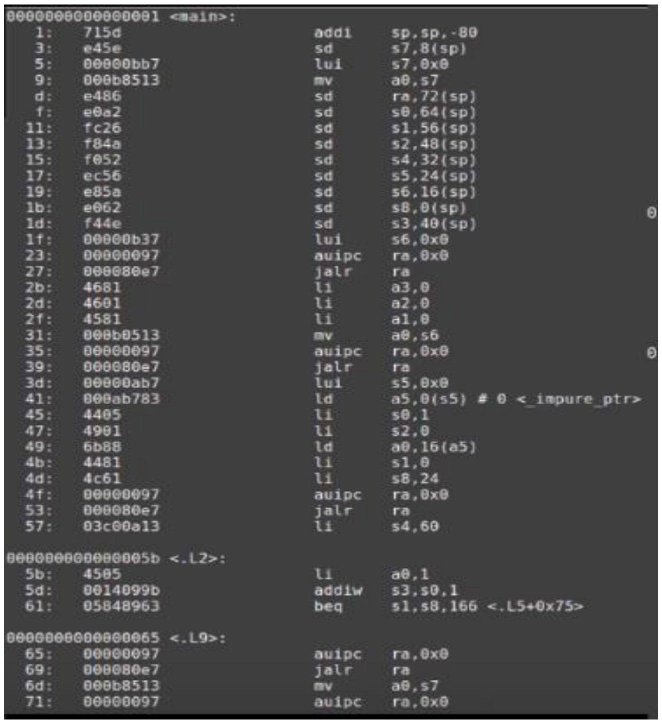
```
0000000000000001 <main>:
   1:   715d                    addi    sp,
   3:   e45e                    sd      s7,8(sp)
   5:   00000bb7                lui     s7,0x0
   9:   000b8513                mv      a0,s7
   d:   e486                    sd      ra,72(sp)
   f:   e0a2                    sd      s0,64(sp)
  11:   fc26                    sd      s1,56(sp)
  13:   f84a                    sd      s2,48(sp)
  15:   f052                    sd      s4,32(sp)
  17:   ec56                    sd      s5,24(sp)
  19:   e85a                    sd      s6,16(sp)
  1b:   e062                    sd      s8,0(sp)
  1d:   f44e                    sd      s3,40(sp)
  1f:   00000b37                lui     s6,0x0
  23:   00000097                auipc   ra,0x0
  27:   000080e7                jalr    ra
  2b:   4681                    li      a3,0
  2d:   4601                    li      a2,0
  2f:   4581                    li      a1,0
  31:   000b0513                mv      a0,s6
  35:   00000097                auipc   ra,0x0
  39:   000080e7                jalr    ra
  3d:   00000ab7                lui     s5,0x0
```

Expandation of C program:



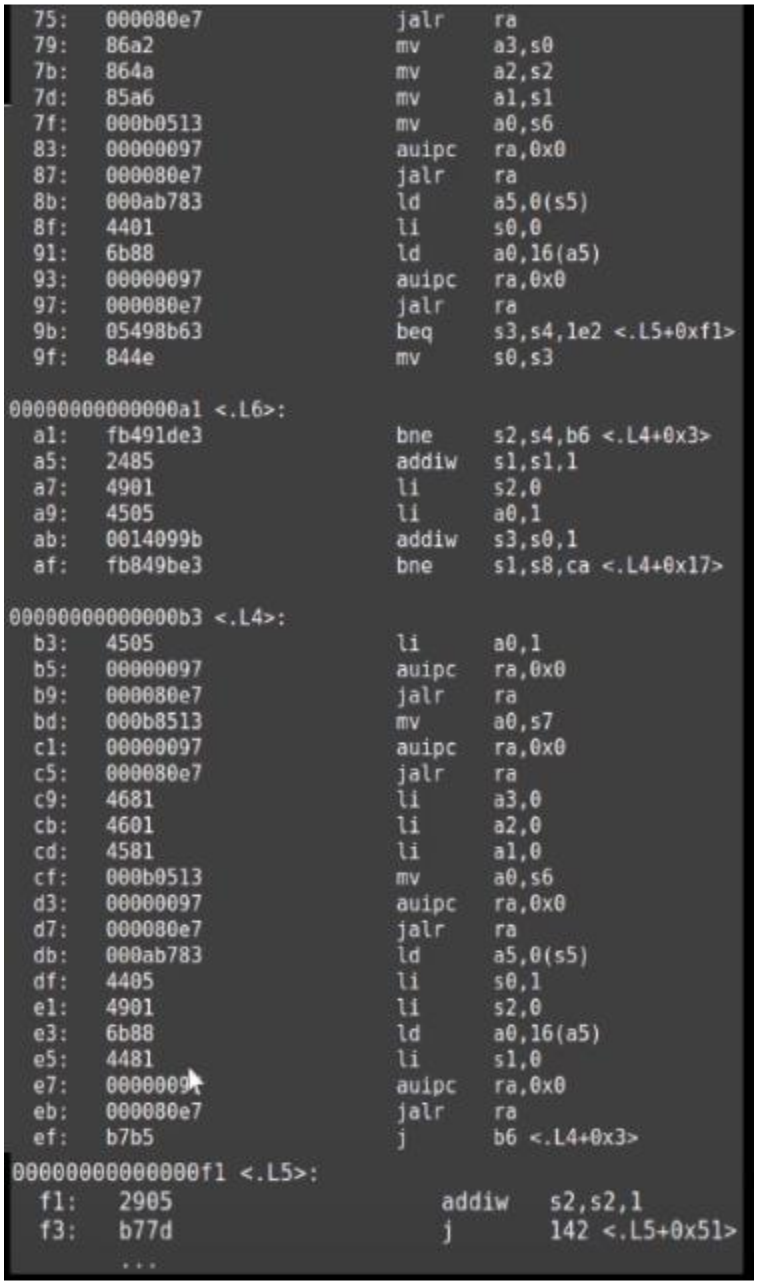
```
#include <stdio.h>
#include <time.h> //for sleep() function

int main()
{
    int hour, minute, second;
    hour=minute=second=0;
    while(1)
    {
        //clear output screen
        system("clear");
        //print time in HH : MM : SS format
        printf("%02d : %02d : %02d ",hour,minute,second);
         //clear output buffer in gcc
        fflush(stdout);
         //increase second
        second++;
        //update hour, minute and second
        if(second==60){
            minute+=1;
            second=0;
        }
        if(minute==60){
            hour+=1;
            minute=0;
        }
        if(hour==24){
            hour=0;
            minute=0;
            second=0;
        }
        sleep(1);   //wait till 1 second
    }
```

Input of the compiler



```
0000000000000001 <main>:
   1:   715d                    addi    sp,sp,-80
   3:   e45e                    sd      s7,8(sp)
   5:   00000bb7                lui     s7,0x0
   9:   000b8513                mv      a0,s7
   d:   e486                    sd      ra,72(sp)
   f:   e0a2                    sd      s0,64(sp)
  11:   fc26                    sd      s1,56(sp)
  13:   f84a                    sd      s2,48(sp)
  15:   f052                    sd      s4,32(sp)
  17:   ec56                    sd      s5,24(sp)
  19:   e85a                    sd      s6,16(sp)
  1b:   e062                    sd      s8,0(sp)
  1d:   f44e                    sd      s3,40(sp)
  1f:   00000b37                lui     s6,0x0
  23:   00000097                auipc   ra,0x0
  27:   000080e7                jalr    ra
  2b:   4681                    li      a3,0
  2d:   4601                    li      a2,0
  2f:   4581                    li      a1,0
  31:   000b0513                mv      a0,s6
  35:   00000097                auipc   ra,0x0
  39:   000080e7                jalr    ra
  3d:   00000ab7                lui     s5,0x0
  41:   000ab783                ld      a5,0(s5) # 0 <_impure_ptr>
  45:   4405                    li      s0,1
  47:   4901                    li      s2,0
  49:   6b88                    ld      a0,16(a5)
  4b:   4481                    li      s1,0
  4d:   4c61                    li      s8,24
  4f:   00000097                auipc   ra,0x0
  53:   000080e7                jalr    ra
  57:   03c00a13                li      s4,60

000000000000005b <.L2>:
  5b:   4505                    li      a0,1
  5d:   0014099b                addiw   s3,s0,1
  61:   05848963                beq     s1,s8,166 <.L5+0x75>

0000000000000065 <.L9>:
  65:   00000097                auipc   ra,0x0
  69:   000080e7                jalr    ra
  6d:   000b8513                mv      a0,s7
  71:   00000097                auipc   ra,0x0
```

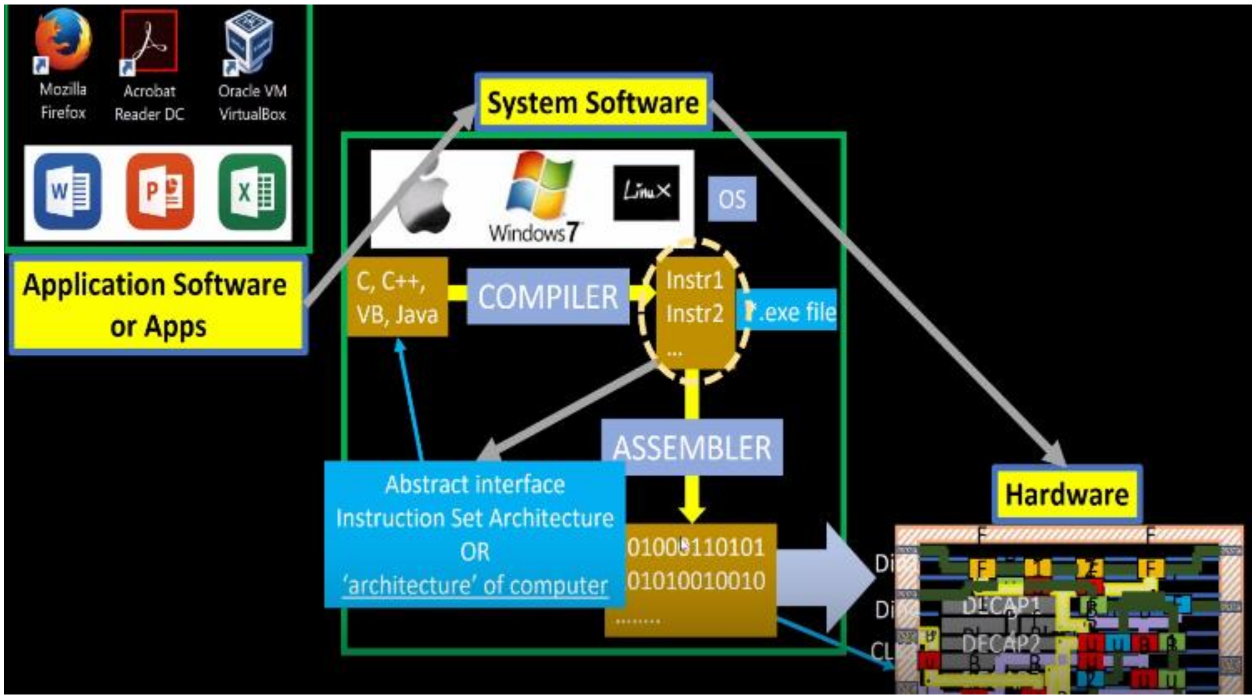
Output of assembler



```
  75:   000080e7                jalr    ra
  79:   86a2                    mv      a3,s0
  7b:   864a                    mv      a2,s2
  7d:   85a6                    mv      a1,s1
  7f:   000b0513                mv      a0,s6
  83:   00000097                auipc   ra,0x0
  87:   000080e7                jalr    ra
  8b:   000ab783                ld      a5,0(s5)
  8f:   4401                    li      s0,0
  91:   6b88                    ld      a0,16(a5)
  93:   00000097                auipc   ra,0x0
  97:   000080e7                jalr    ra
  9b:   05498b63                beq     s3,s4,1e2 <.L5+0xf1>
  9f:   844e                    mv      s0,s3

00000000000000a1 <.L6>:
  a1:   fb491de3                bne     s2,s4,b6 <.L4+0x3>
  a5:   2485                    addiw   s1,s1,1
  a7:   4901                    li      s2,0
  a9:   4505                    li      a0,1
  ab:   0014099b                addiw   s3,s0,1
  af:   fb849be3                bne     s1,s8,ca <.L4+0x17>

00000000000000b3 <.L4>:
  b3:   4505                    li      a0,1
  b5:   00000097                auipc   ra,0x0
  b9:   000080e7                jalr    ra
  bd:   000b8513                mv      a0,s7
  c1:   00000097                auipc   ra,0x0
  c5:   000080e7                jalr    ra
  c9:   4681                    li      a3,0
  cb:   4601                    li      a2,0
  cd:   4581                    li      a1,0
  cf:   000b0513                mv      a0,s6
  d3:   00000097                auipc   ra,0x0
  d7:   000080e7                jalr    ra
  db:   000ab783                ld      a5,0(s5)
  df:   4405                    li      s0,1
  e1:   4901                    li      s2,0
  e3:   6b88                    ld      a0,16(a5)
  e5:   4481                    li      s1,0
  e7:   0000009                 auipc   ra,0x0
  eb:   000080e7                jalr    ra
  ef:   b7b5                    j       b6 <.L4+0x3>
00000000000000f1 <.L5>:
  f1:   2905                    addiw   s2,s2,1
  f3:   b77d                    j       142 <.L5+0x51>
        ...
```

Output of the compiler

The instructions acts as an abstract interface (Instruction set architecture or the architecture of the computer) between the C language and the hardware. The instructions actually represent the hardware. This is the language with which a human or a user speaks to the computer.



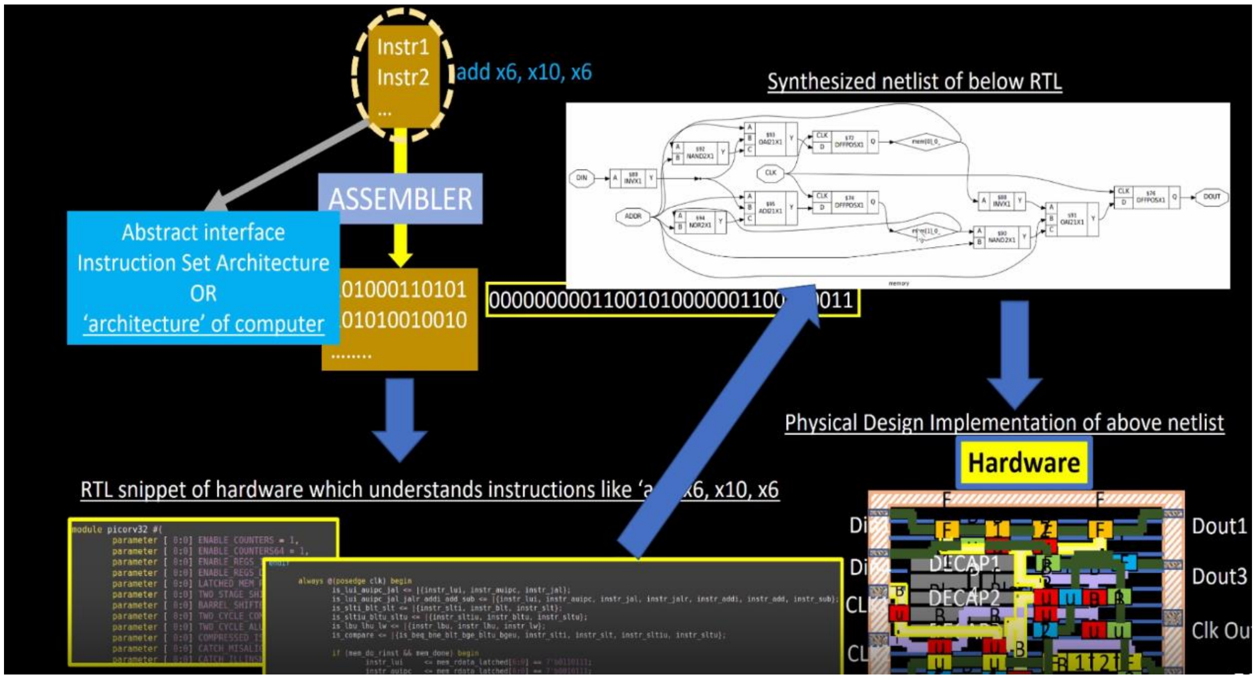
Further Expandation:

There is one more interface that will reach from instructions to hardware and that interface is the hardware description language. Hardware understands only 0's and 1's.

For example : If 00000000011001010000001100110011 is the input the hardware understands that it should add x6, x10, x6.

"x6, x10, x6" is called as the output of the compiler. The output of the assembler is the binary pattern.

Now we need a RTL which understands these specifications (Example: “x6, x10, x6”) and that is called as the implementation or the RTL implementations of the instructions set. Then the RTL is getting synthesized into the netlist.This is in form of gates. From this point to the hardware it is the physical design implementation of the net list.

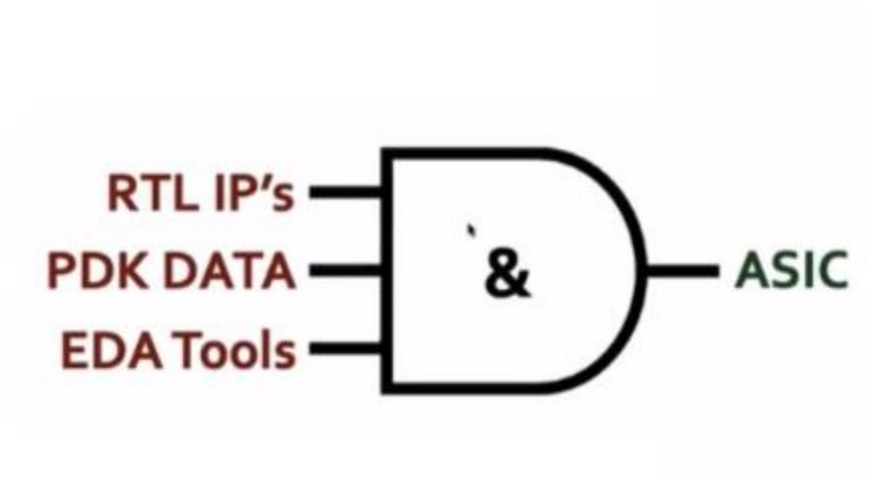

Instr1
Instr2
add x6, x10, x6
Synthesized netlist of below RTL
ASSEMBLER
Abstract interface
Instruction Set Architecture
OR
'architecture' of computer
01000110101
01010010010
Physical Design Implementation of above netlist
Hardware
RTL snippet of hardware which understands instructions like 'a x6, x10, x6
DECAP1
DECAP2
Dout1
Dout3
Clk Ou

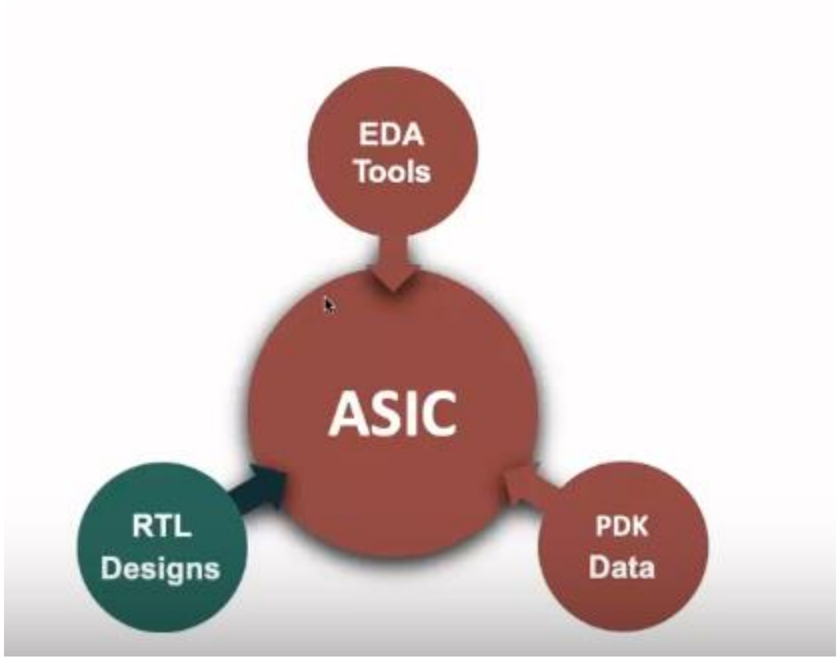
Open Lane

ASIC requires mainly three components for design

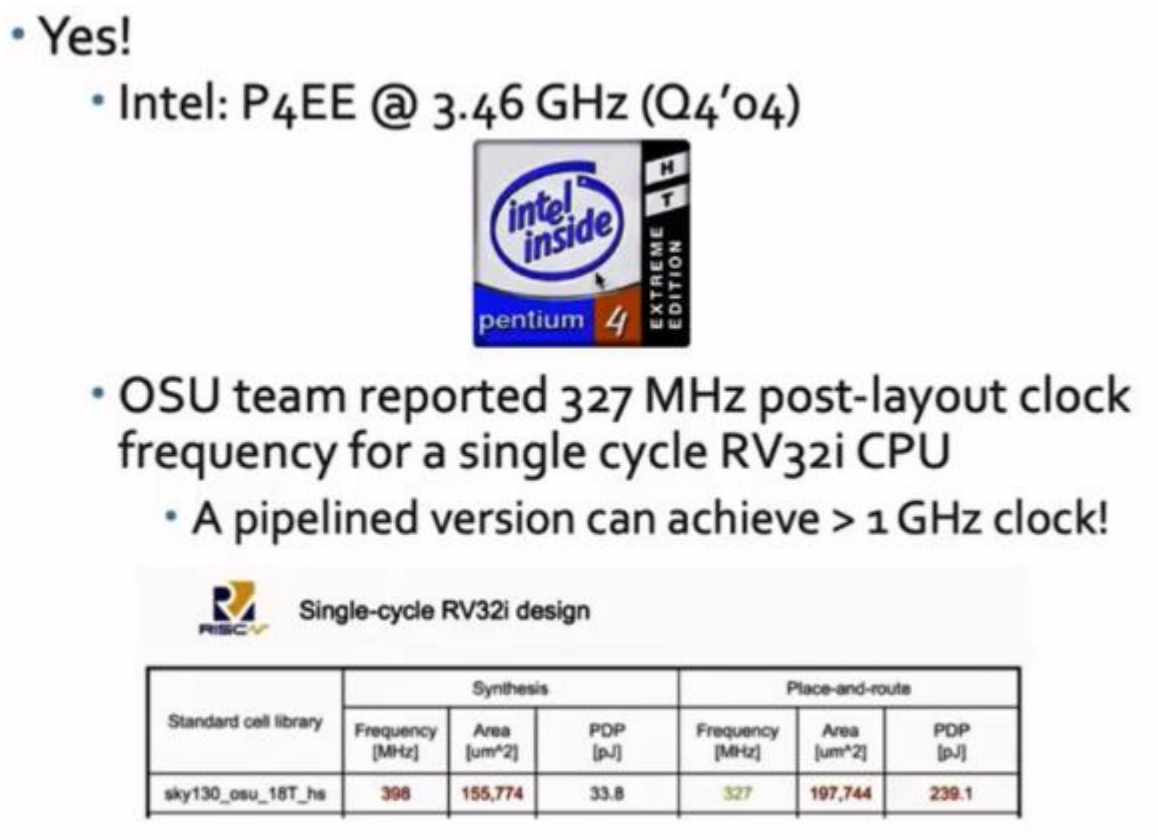
- RTL IPs - they are building blocks written in a hardware description language. These blocks describe the functioning of the circuit at a basic level and are pre designed and verified.

- EDA Tools - EDA, which expands to Electronic Design Automation tools are used for design, simulation and verification and the analyzing of circuit designs.
- PDK data - Process Design Kit data is a collection of files used to model the fabrication process for EDA tools used to design an IC. It is the interface between FAB and the designers
    - Process design Rules
    - Device models
    - Digital Standard Cell Libraries
    - I/O libraries





Is 130nm fast?



- Yes!
  - Intel: P4EE @ 3.46 GHz (Q4'04)
  - OSU team reported 327 MHz post-layout clock frequency for a single cycle RV32i CPU
    - A pipelined version can achieve > 1 GHz clock!

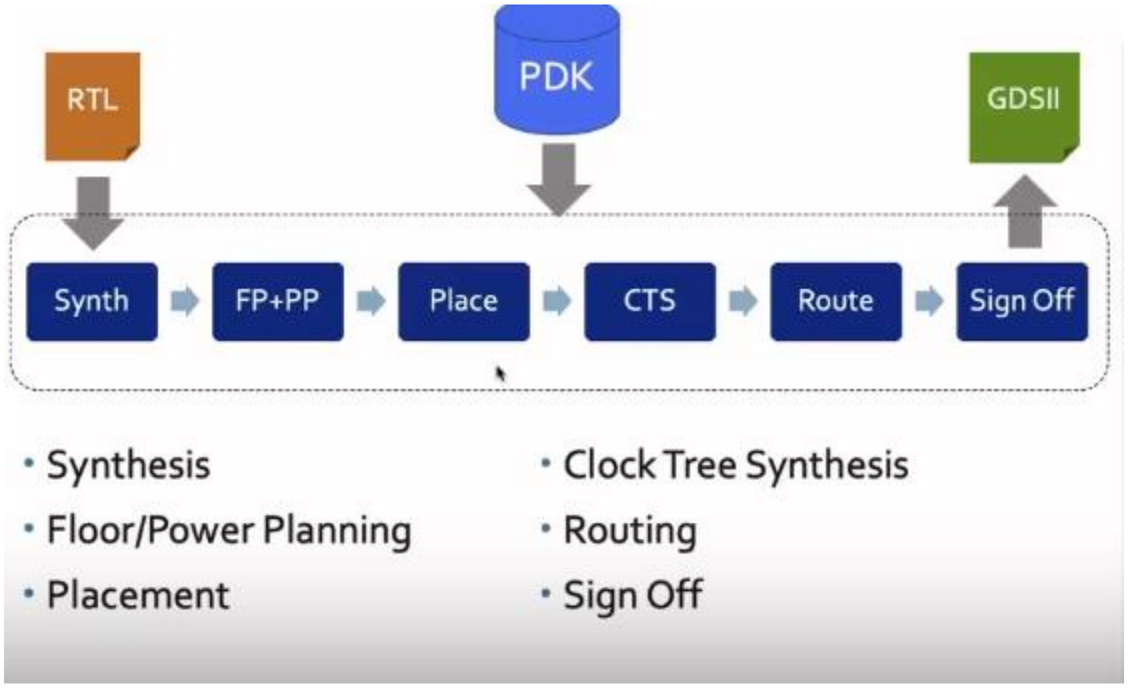
Single-cycle RV32i design

Standard cell library	Synthesis			Place-and-route		
	Frequency [MHz]	Area [um^2]	PDP [pJ]	Frequency [MHz]	Area [um^2]	PDP [pJ]
sky130_osu_18T_hs	398	155,774	33.8	327	197,744	239.1

ASIC flow objective: RTL to GSDll

- Also called the Automated PnR and/or Physical Implementation

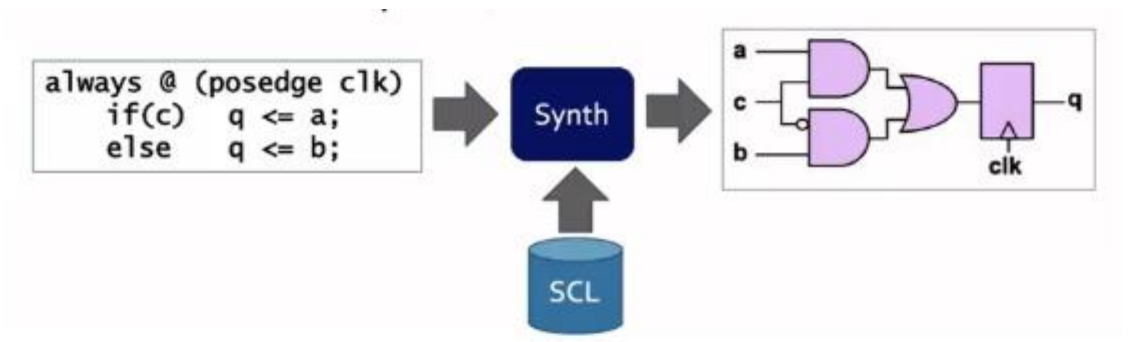
Simplified RTL to GSDll Flow:



<u>Synthesis:</u>

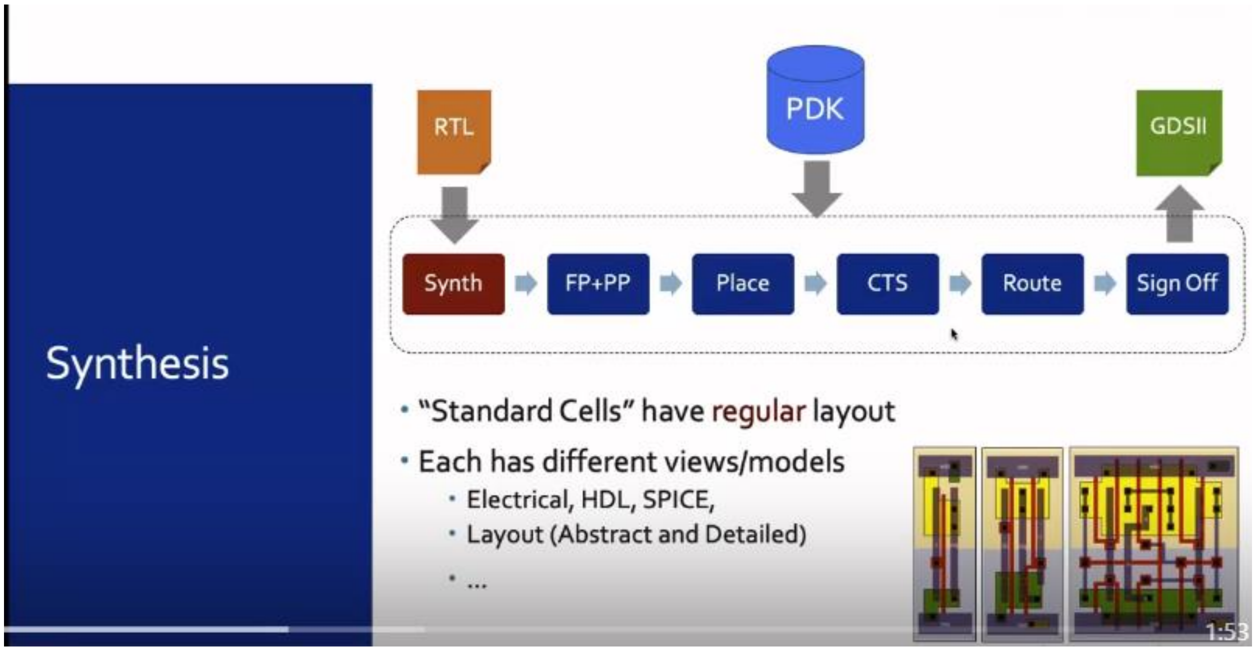
The RTL to GDSII (Register Transfer Level to Graphic Design System II) design process takes many steps, that are -:

- Synthesis - it converts hardware description languages such as VERILOG into gate-level representations part of a standard cell library. The cells part of this library have a regular layout. Each of these have different views/models such as Electrical, HDL, Spice etc.



```
always @ (posedge clk)
   if(c)   q <= a;
   else    q <= b;
```

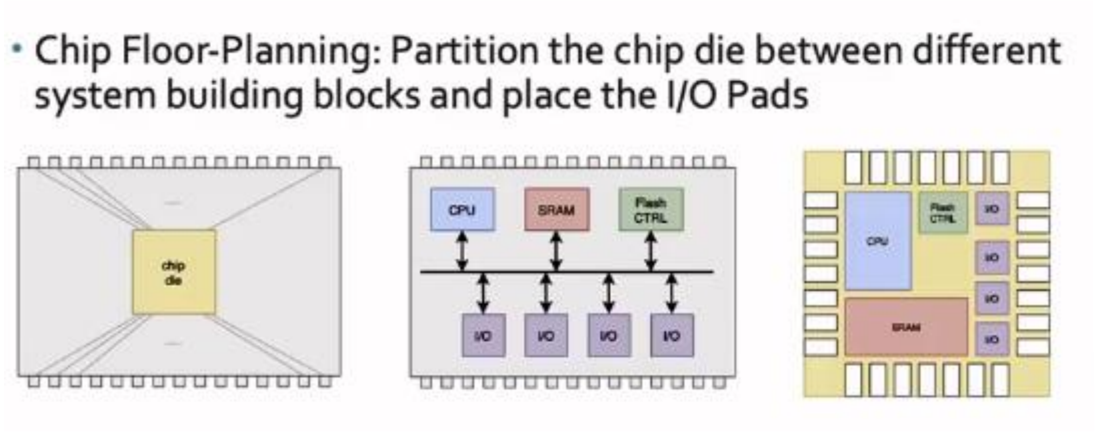
- Floor Planning involves optimizing chip performance, area utilization, and connectivity through spatial arrangements (i.e. the layout and placement of various components)The three main purposes of floor planning are firstly, minimizing wire lengths, secondly, reducing signal delays, thirdly, optimizing power distribution, and fourthly, ensuring efficient chip utilization. Power Planning aims to ensure stable and reliable power delivery to all components by effective distribution and design of power supplies and power distribution networks (PDN). The main purposes of this are minimizing voltage drop and noise, reducing power distribution network (PDN) resistances and capacitances, and ensuring uniform power distribution throughout. Usually the chip is powered through VDD pads which are connected to various components through parallel rectangular strips causing lesser resistance.

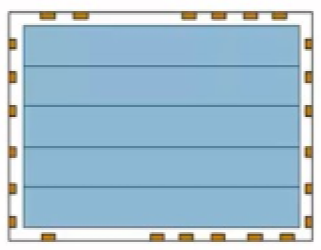

Synthesis

- "Standard Cells" have regular layout
- Each has different views/models
  - Electrical, HDL, SPICE,
  - Layout (Abstract and Detailed)
  - ...

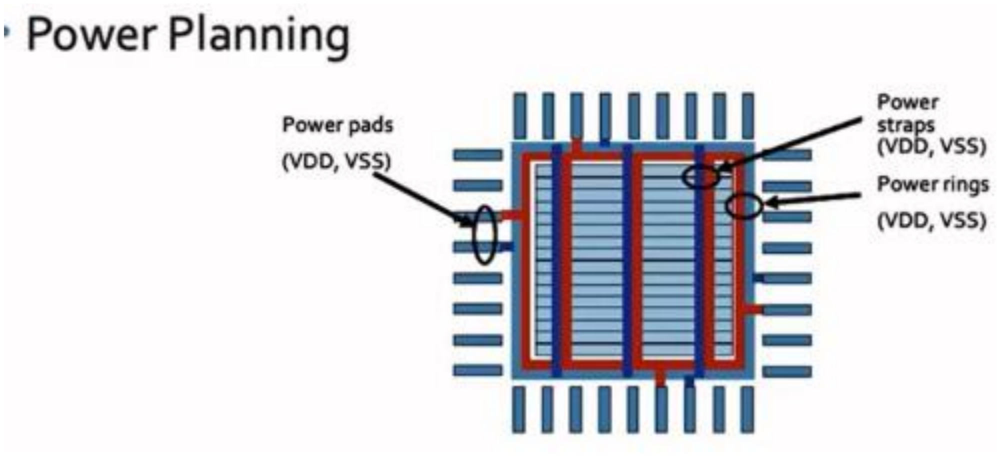
Floor and Power planning:

- Floor


- Chip Floor-Planning: Partition the chip die between different system building blocks and place the I/O Pads



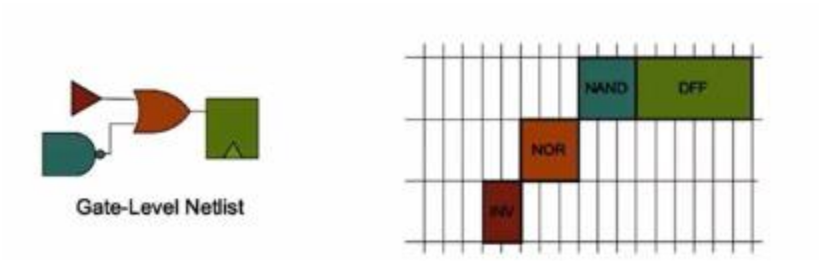
- <u>Power planning</u>



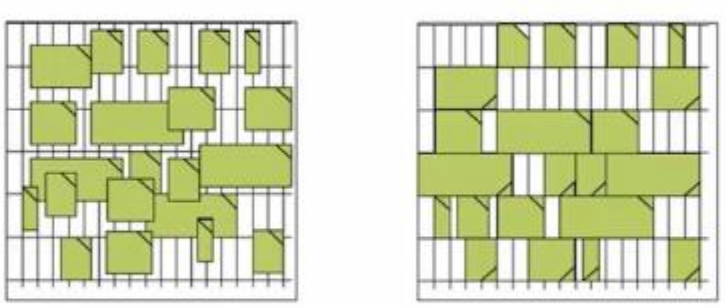
What is placement?

- Placement - it is the process of determing where a component will be placed on the chip. The components can include standard cells, macros, and I/O pads. The cells are usually placed on floorplan rows, and are aligned with the sites. There are majorly two steps - global and detailed.

<u>Place the cells on the floorplan rows, aligned with the sites.</u>

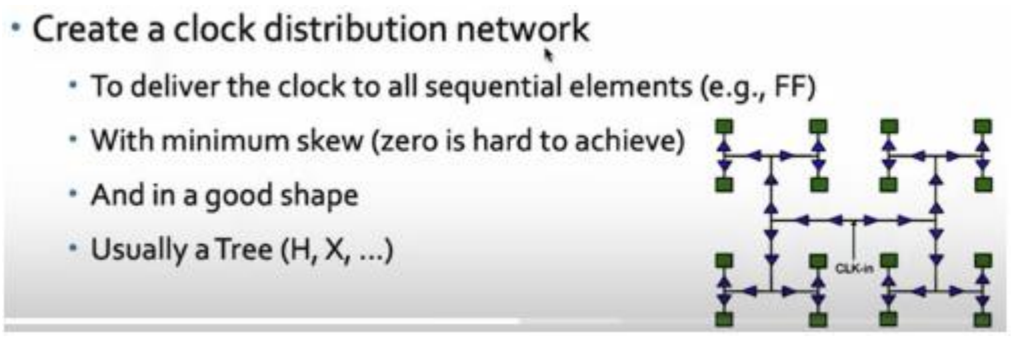


Usually done in 2 steps:



Clock tree synthesis:

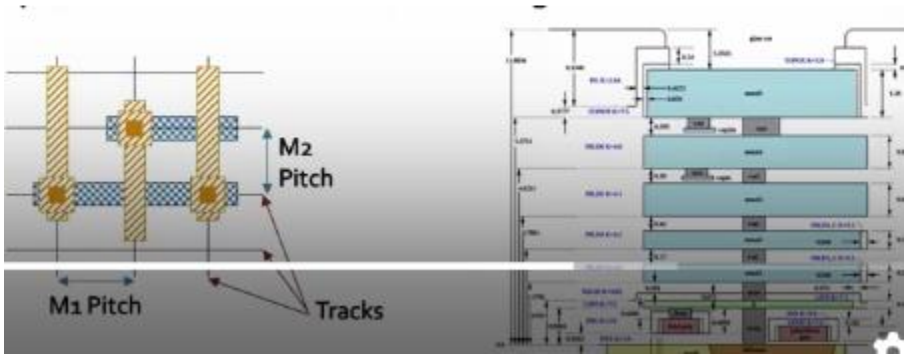
This step is done before routing, because the clock needs to be routed by delivering the clock to all sequential elements.



- Create a clock distribution network
  - To deliver the clock to all sequential elements (e.g., FF)
  - With minimum skew (zero is hard to achieve)
  - And in a good shape
  - Usually a Tree (H, X, ...)

Routing:

The determination of the interconnections of the components through the various metal layers, whose thickness, pitch etc is detailed by the PDK. The SKY130 has 6 layers.



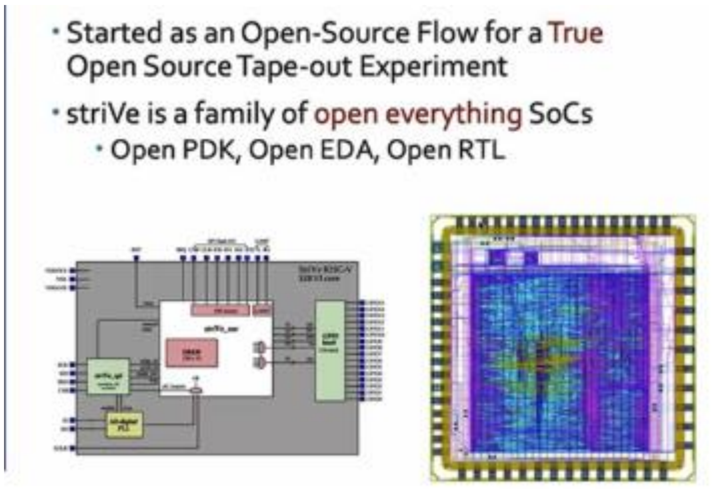
- Metal tracks from a routing grid.
- Routing grid is huge
- Divide and conquer
    - <u>Global routing: </u>Generates routing guides
    - <u>Detailed routing</u>: Uses the routing guides to implement the actual wiring.

<u>Sign off:</u>

- Physical verifications
    - Design Rules checking (DRC)

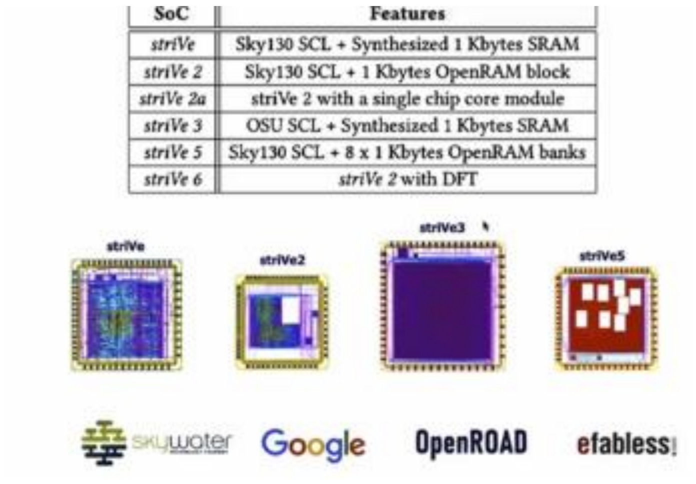
        - Layout vs. schematic (LVS)
    - Timing verification
        - Static timing analysis

<u>Open Lane:</u>



- Started as an Open-Source Flow for a True Open Source Tape-out Experiment
- striVe is a family of open everything SoCs
    - Open PDK, Open EDA, Open RTL

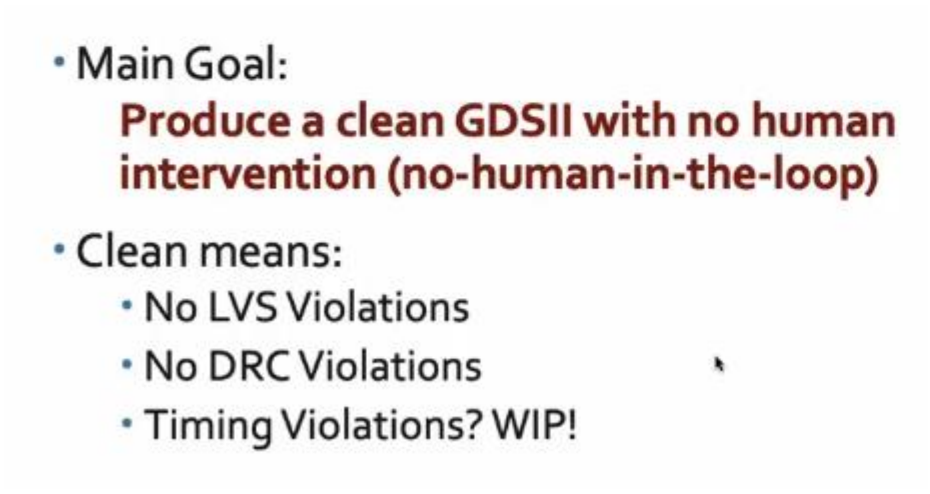
<u>Strive SoC family:</u>



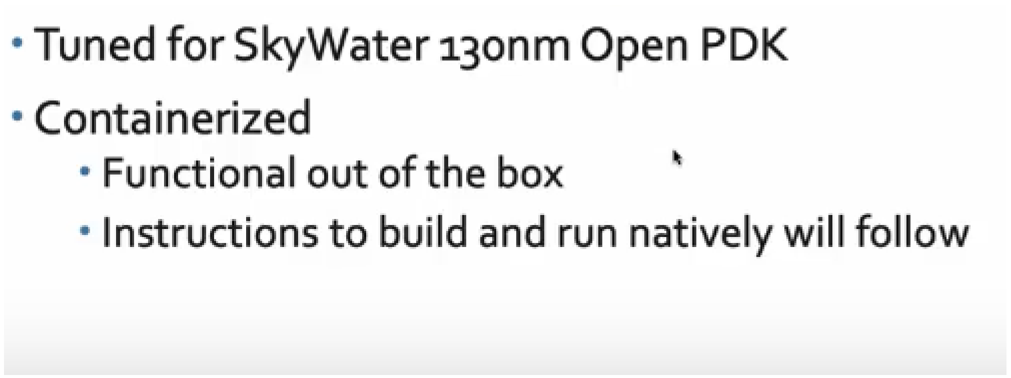
SoC	Features
striVe	Sky130 SCL + Synthesized 1 Kbytes SRAM
striVe 2	Sky130 SCL + 1 Kbytes OpenRAM block
striVe 2a	striVe 2 with a single chip core module
striVe 3	OSU SCL + Synthesized 1 Kbytes SRAM
striVe 5	Sky130 SCL + 8 x 1 Kbytes OpenRAM banks
striVe 6	*striVe 2* with DFT

striVe
striVe2
striVe3
striVe5

<u>Open Lane ASIC Flow:</u>



- Main Goal:
  Produce a clean GDSII with no human intervention (no-human-in-the-loop)
- Clean means:
  - No LVS Violations
  - No DRC Violations
  - Timing Violations? WIP!



- Tuned for SkyWater 130nm Open PDK
- Containerized
  - Functional out of the box
  - Instructions to build and run natively will follow

Can be used to harden Macro chips

Two modes of operation:

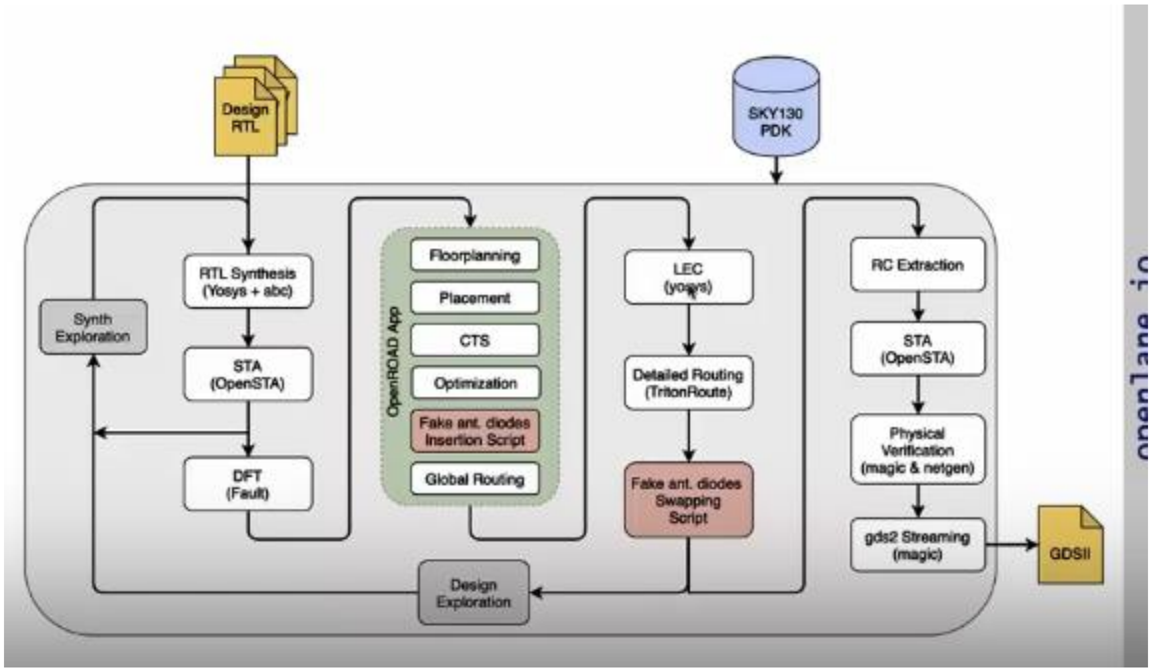
- Autonomous or Interactive

Design space exploration:

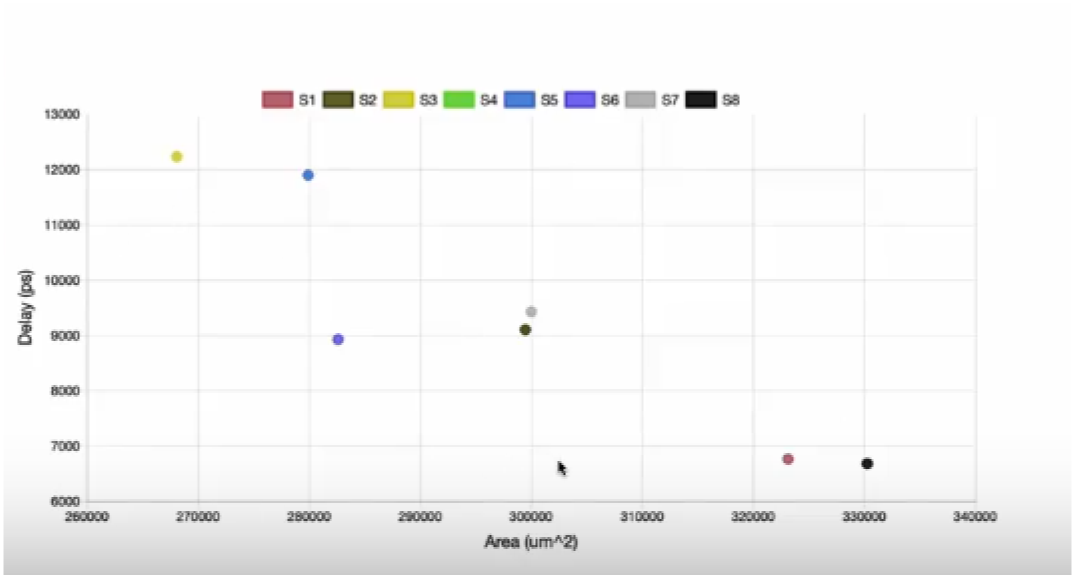
- Find the best set of flow configurations.

Large number of design samples.

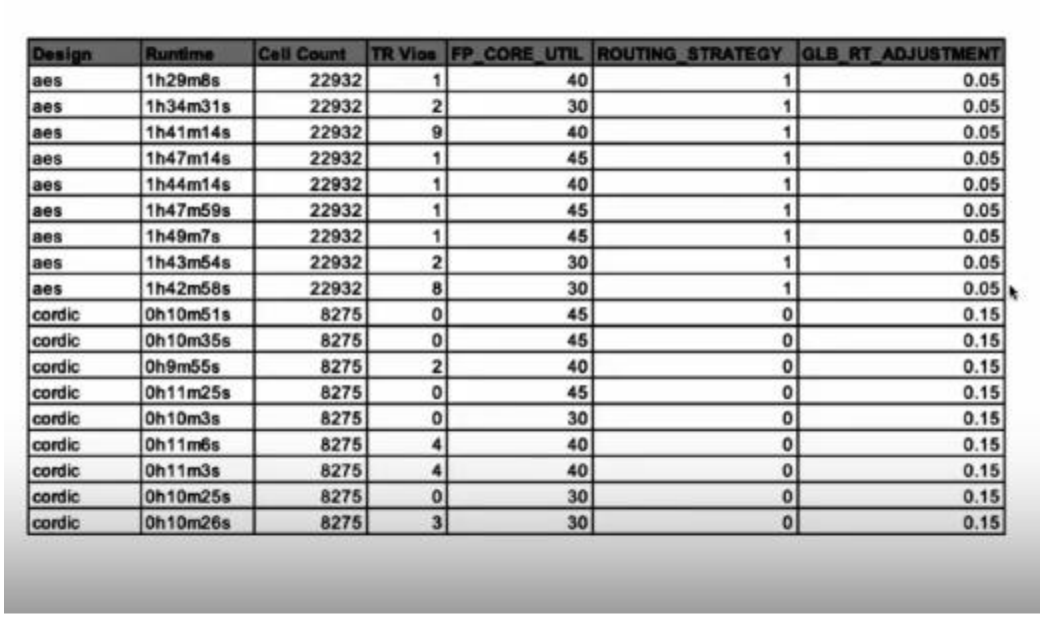
- 43 designs with their best configurations
- More will be added soon.



Synthesis Exploration:

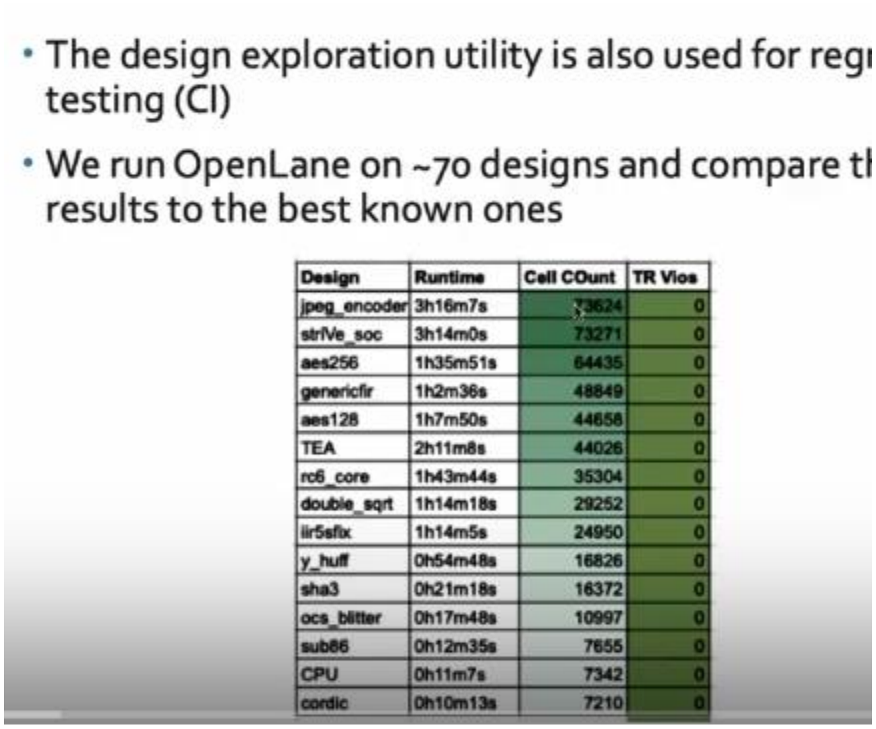


Design exploration:



Design	Runtime	Cell Count	TR Vios	FP_CORE_UTIL	ROUTING_STRATEGY	GLB_RT_ADJUSTMENT
aes	1h29m8s	22932	1	40	1	0.05
aes	1h34m31s	22932	2	30	1	0.05
aes	1h41m14s	22932	9	40	1	0.05
aes	1h47m14s	22932	1	45	1	0.05
aes	1h44m14s	22932	1	40	1	0.05
aes	1h47m59s	22932	1	45	1	0.05
aes	1h49m7s	22932	1	45	1	0.05
aes	1h43m54s	22932	2	30	1	0.05
aes	1h42m58s	22932	8	30	1	0.05
cordic	0h10m51s	8275	0	45	0	0.15
cordic	0h10m35s	8275	0	45	0	0.15
cordic	0h9m55s	8275	2	40	0	0.15
cordic	0h11m25s	8275	0	45	0	0.15
cordic	0h10m3s	8275	0	30	0	0.15
cordic	0h11m6s	8275	4	40	0	0.15
cordic	0h11m3s	8275	4	40	0	0.15
cordic	0h10m25s	8275	0	30	0	0.15
cordic	0h10m26s	8275	3	30	0	0.15

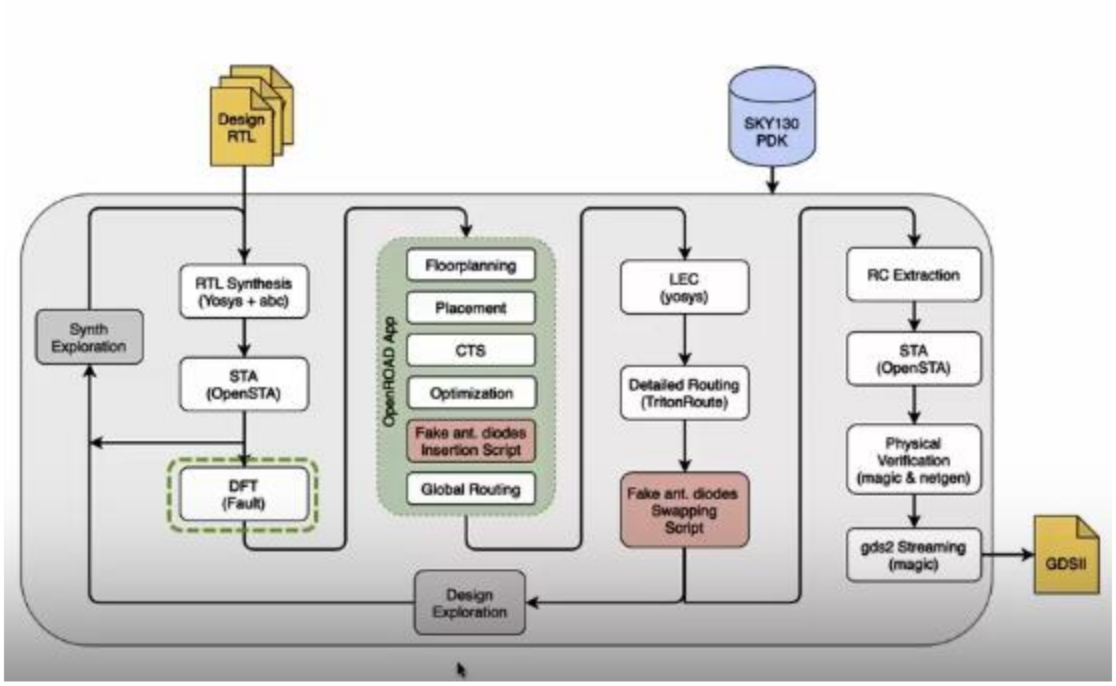
Open Lane Regression Testing:



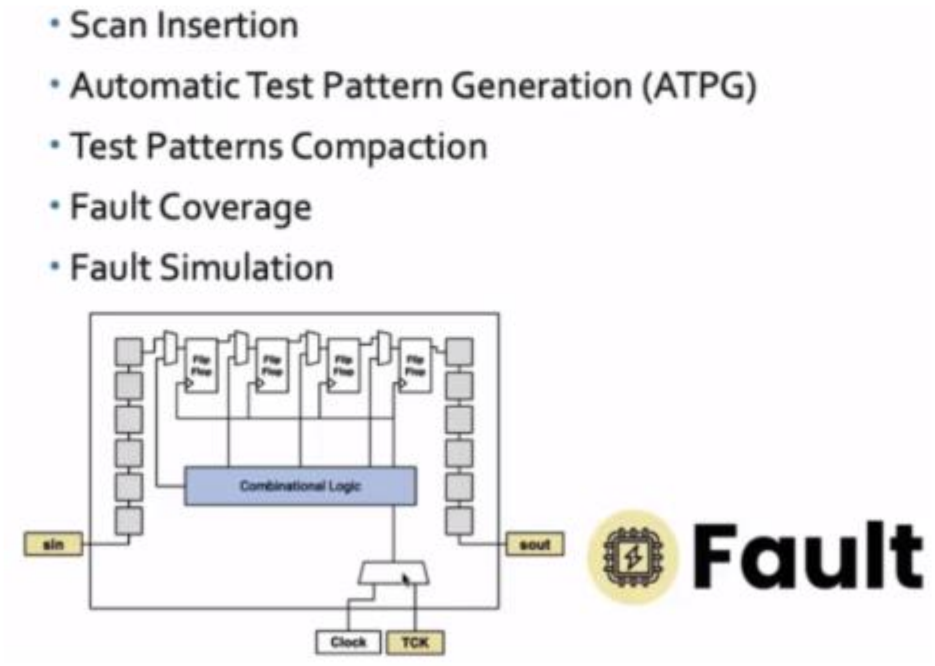
- The design exploration utility is also used for regı testing (CI)
- We run OpenLane on ~70 designs and compare tl results to the best known ones

Design	Runtime	Cell COunt	TR Vios
jpeg_encoder	3h16m7s	73624	0
striVe_soc	3h14m0s	73271	0
aes256	1h35m51s	64435	0
genericfir	1h2m36s	48849	0
aes128	1h7m50s	44658	0
TEA	2h11m8s	44026	0
rc6_core	1h43m44s	35304	0
double_sqrt	1h14m18s	29252	0
iir5sfix	1h14m5s	24950	0
y_huff	0h54m48s	16826	0
sha3	0h21m18s	16372	0
ocs_blitter	0h17m48s	10997	0
sub86	0h12m35s	7655	0
CPU	0h11m7s	7342	0
cordic	0h10m13s	7210	0

Open Lane ASIC Flow:

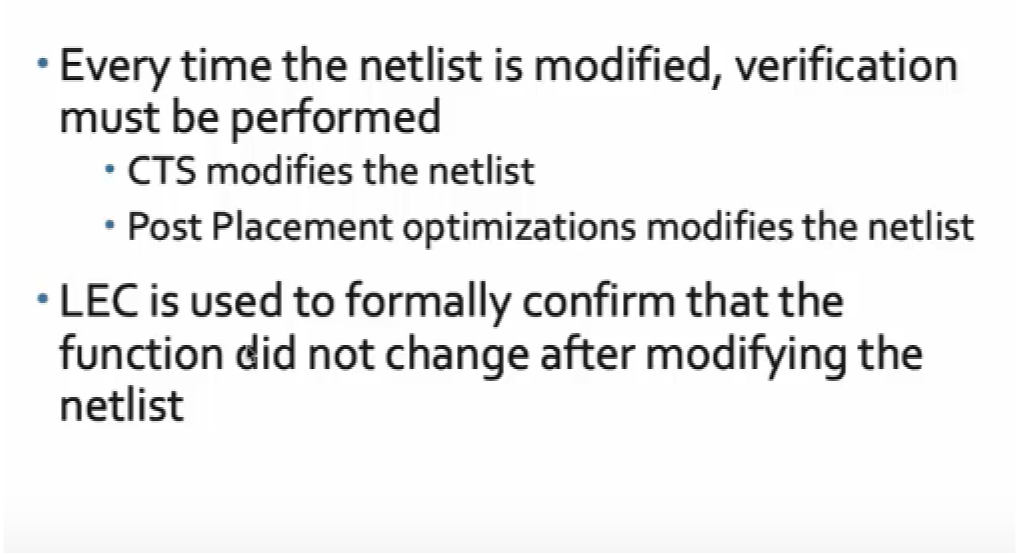


Design for test:



- Scan Insertion
- Automatic Test Pattern Generation (ATPG)
- Test Patterns Compaction
- Fault Coverage
- Fault Simulation

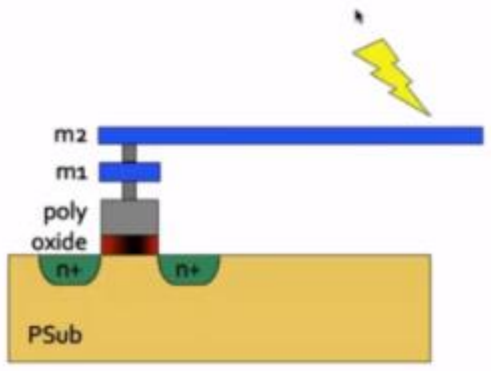
Logic Equivalence Check (LEC):



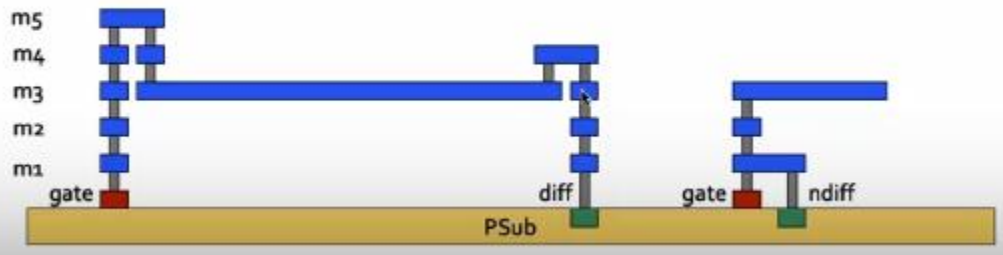
- Every time the netlist is modified, verification must be performed
  - CTS modifies the netlist
  - Post Placement optimizations modifies the netlist
- LEC is used to formally confirm that the function did not change after modifying the netlist

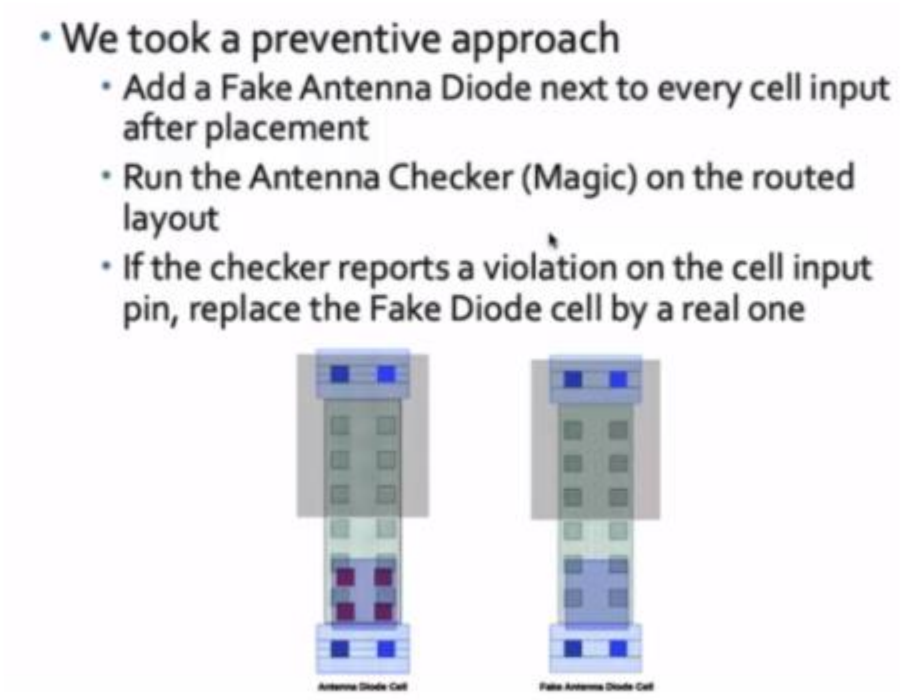
Dealing with Antenna Rules violations:

- When a metal wire segment is fabricated, it can act as an antenna.
  - Reactive ion etching causes charge to accumulate on the wire.
  - Transistor gates can be damaged during fabrication



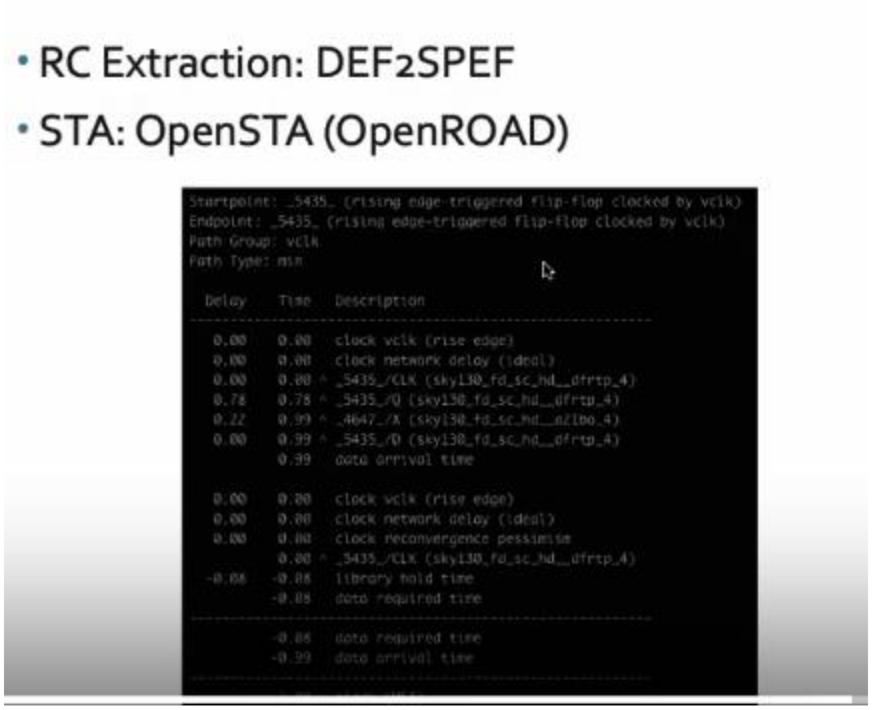
- Two solutions:
  - Bridging attaches a higher layer intermediary
    - Requires Router awareness (not there yet!)
  - Add antenna diode cell to leak away charges
    - Antenna diodes are provided by the SCL





- We took a preventive approach
  - Add a Fake Antenna Diode next to every cell input after placement
  - Run the Antenna Checker (Magic) on the routed layout
  - If the checker reports a violation on the cell input pin, replace the Fake Diode cell by a real one

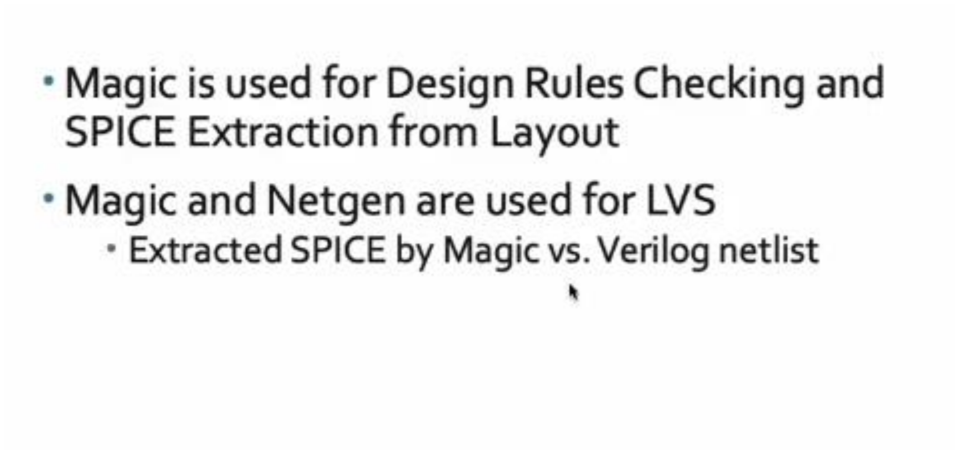
Static Time analysis:



- RC Extraction: DEF2SPEF
- STA: OpenSTA (OpenROAD)

```
Startpoint: _5435_ (rising edge-triggered flip-flop clocked by vclk)
Endpoint: _5435_ (rising edge-triggered flip-flop clocked by vclk)
Path Group: vclk
Path Type: min

  Delay    Time   Description
---------------------------------------------------------
   0.00    0.00   clock vclk (rise edge)
   0.00    0.00   clock network delay (ideal)
   0.00    0.00 ^ _5435_/CLK (sky130_fd_sc_hd__dfrtp_4)
   0.78    0.78 ^ _5435_/Q (sky130_fd_sc_hd__dfrtp_4)
   0.22    0.99 ^ _4647_/X (sky130_fd_sc_hd__a21bo_4)
   0.00    0.99 ^ _5435_/D (sky130_fd_sc_hd__dfrtp_4)
           0.99   data arrival time

   0.00    0.00   clock vclk (rise edge)
   0.00    0.00   clock network delay (ideal)
   0.00    0.00   clock reconvergence pessimism
           0.00 ^ _5435_/CLK (sky130_fd_sc_hd__dfrtp_4)
  -0.08   -0.08   library hold time
          -0.08   data required time
---------------------------------------------------------
          -0.08   data required time
          -0.99   data arrival time
---------------------------------------------------------
```

Physical verification DRC and LVS:

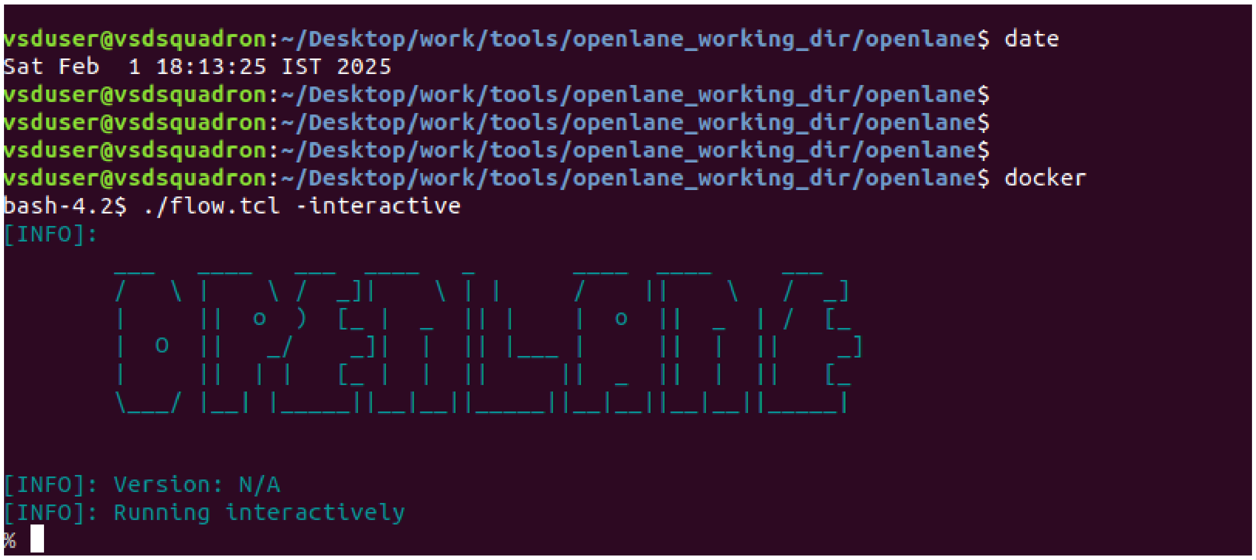


- Magic is used for Design Rules Checking and SPICE Extraction from Layout
- Magic and Netgen are used for LVS
  - Extracted SPICE by Magic vs. Verilog netlist

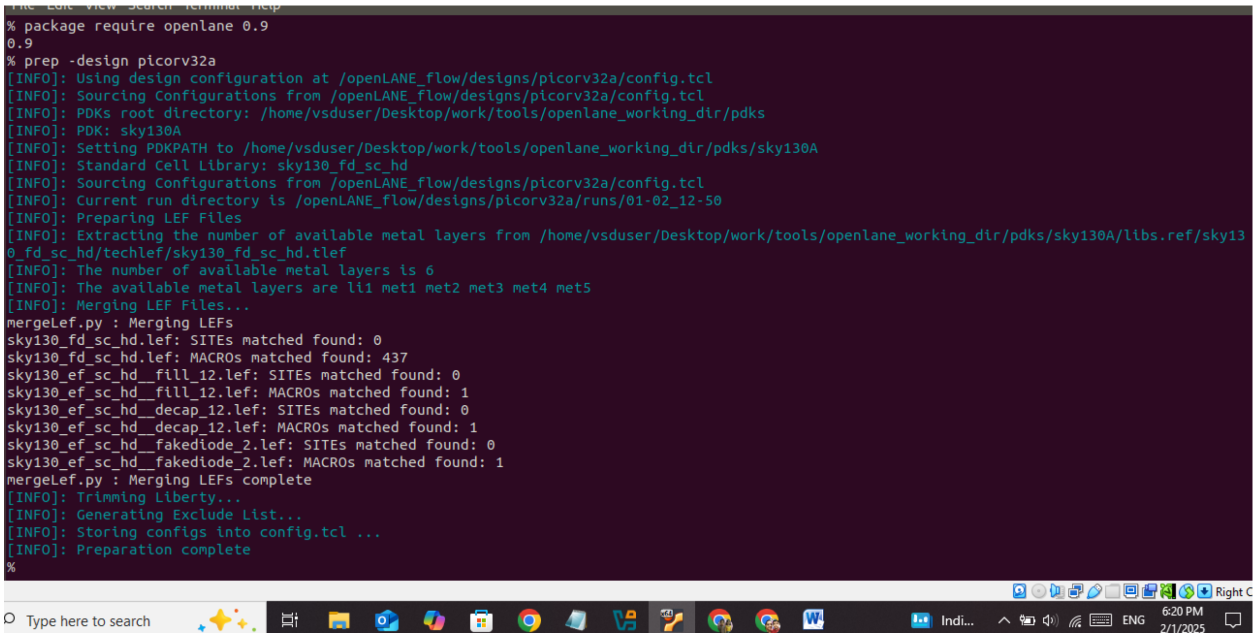
What is GITHUB?

GITHUB is a remote repository collection.

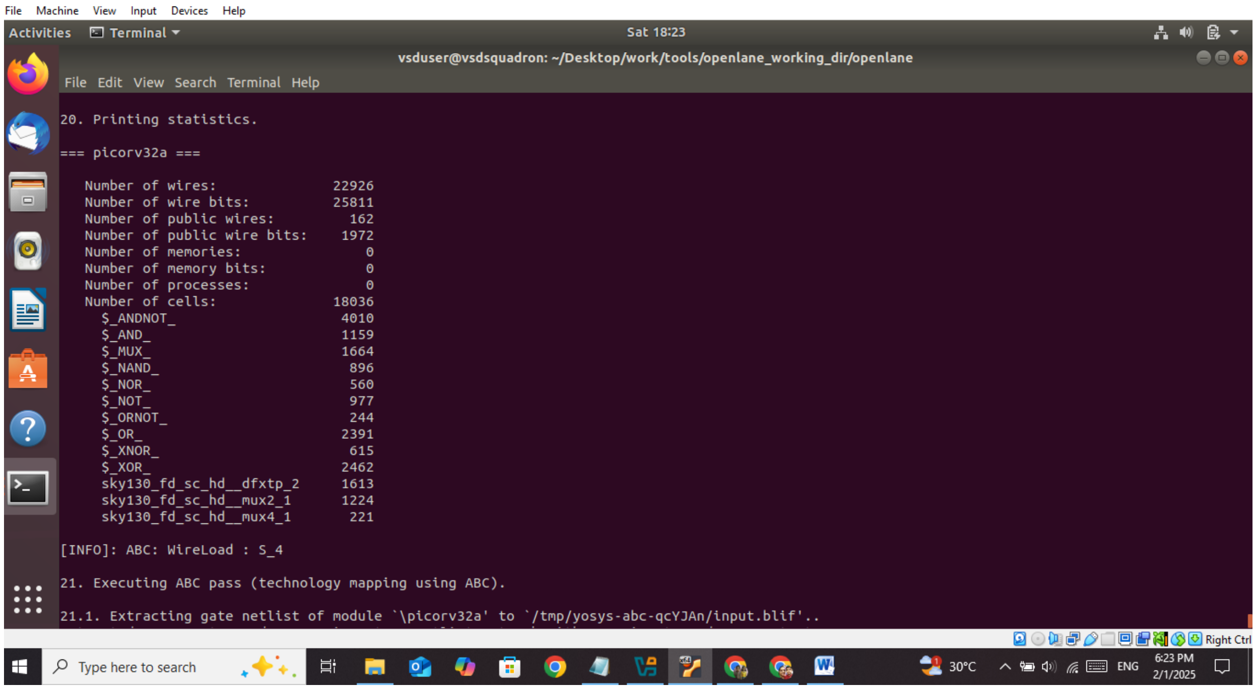
DAY 1 : Lab 1



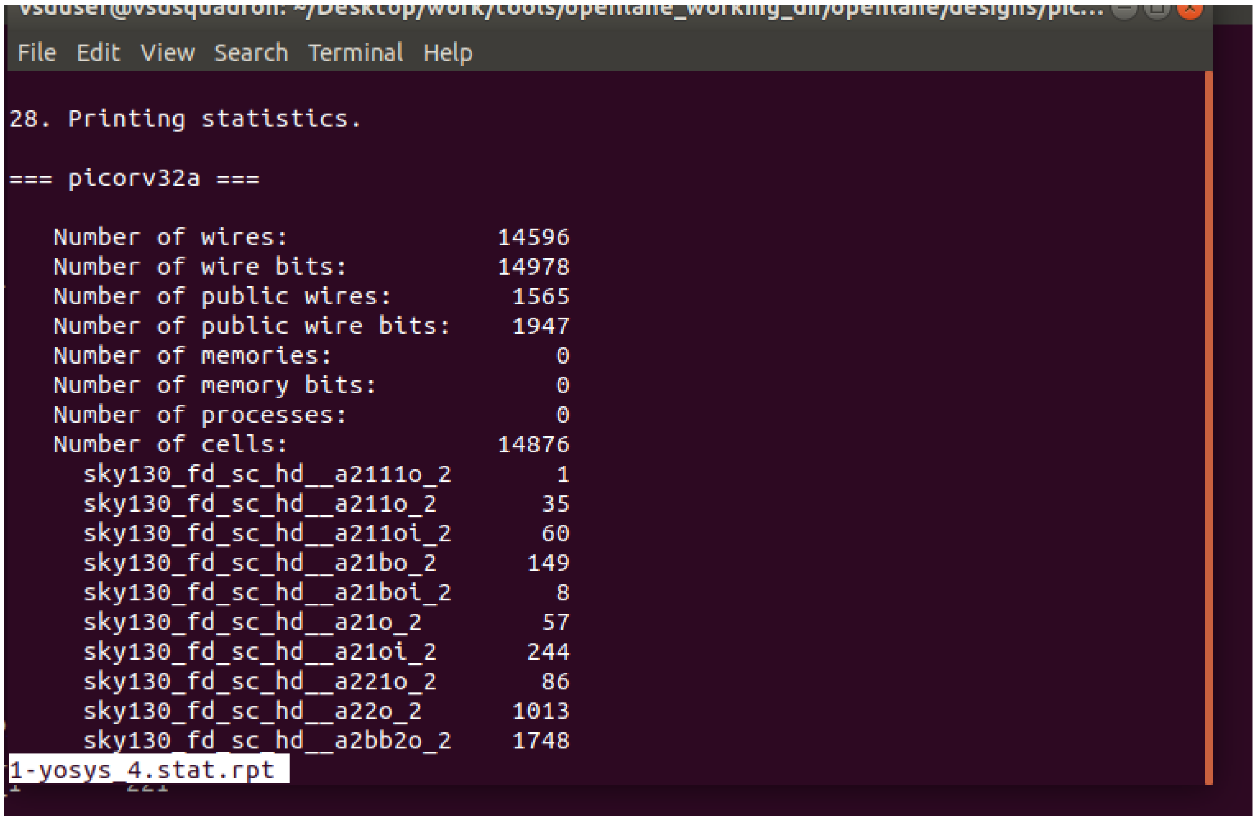
```
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/openlane$ date
Sat Feb  1 18:13:25 IST 2025
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/openlane$
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/openlane$
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/openlane$
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/openlane$ docker
bash-4.2$ ./flow.tcl -interactive
[INFO]:
         ___   ____   ___  ____   _       ____  ____     ___
        /   \ |    \ /  _]|    \ | |     /    ||    \   /  _]
        |     ||  o  )  [_ |  _  || |    |  o  ||  _  | /  [_
        |  O  ||   _/    _]|  |  || |___ |     ||  |  ||    _]
        |     ||  |  |   [_ |  |  ||     ||  _  ||  |  ||   [_
        \___/ |__|  |_____||__|__||_____||__|__||__|__||_____|


[INFO]: Version: N/A
[INFO]: Running interactively
%
```



```
% package require openlane 0.9
0.9
% prep -design picorv32a
[INFO]: Using design configuration at /openLANE_flow/designs/picorv32a/config.tcl
[INFO]: Sourcing Configurations from /openLANE_flow/designs/picorv32a/config.tcl
[INFO]: PDKs root directory: /home/vsduser/Desktop/work/tools/openlane_working_dir/pdks
[INFO]: PDK: sky130A
[INFO]: Setting PDKPATH to /home/vsduser/Desktop/work/tools/openlane_working_dir/pdks/sky130A
[INFO]: Standard Cell Library: sky130_fd_sc_hd
[INFO]: Sourcing Configurations from /openLANE_flow/designs/picorv32a/config.tcl
[INFO]: Current run directory is /openLANE_flow/designs/picorv32a/runs/01-02_12-50
[INFO]: Preparing LEF Files
[INFO]: Extracting the number of available metal layers from /home/vsduser/Desktop/work/tools/openlane_working_dir/pdks/sky130A/libs.ref/sky13
0_fd_sc_hd/techlef/sky130_fd_sc_hd.tlef
[INFO]: The number of available metal layers is 6
[INFO]: The available metal layers are li1 met1 met2 met3 met4 met5
[INFO]: Merging LEF Files...
mergeLef.py : Merging LEFs
sky130_fd_sc_hd.lef: SITEs matched found: 0
sky130_fd_sc_hd.lef: MACROs matched found: 437
sky130_ef_sc_hd__fill_12.lef: SITEs matched found: 0
sky130_ef_sc_hd__fill_12.lef: MACROs matched found: 1
sky130_ef_sc_hd__decap_12.lef: SITEs matched found: 0
sky130_ef_sc_hd__decap_12.lef: MACROs matched found: 1
sky130_ef_sc_hd__fakediode_2.lef: SITEs matched found: 0
sky130_ef_sc_hd__fakediode_2.lef: MACROs matched found: 1
mergeLef.py : Merging LEFs complete
[INFO]: Trimming Liberty...
[INFO]: Generating Exclude List...
[INFO]: Storing configs into config.tcl ...
[INFO]: Preparation complete
%
```

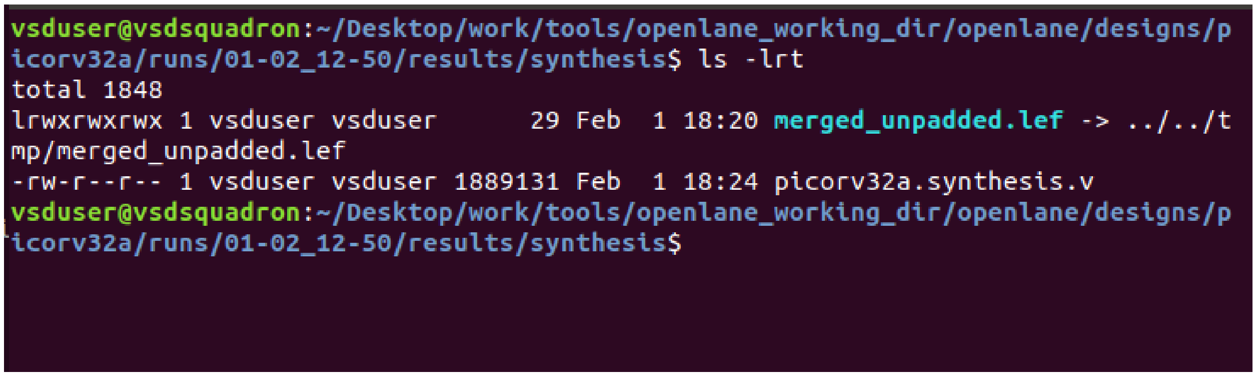

File Machine View Input Devices Help
Activities
Terminal
Sat 18:23
vsduser@vsdsquadron: ~/Desktop/work/tools/openlane_working_dir/openlane
File Edit View Search Terminal Help

```
20. Printing statistics.

=== picorv32a ===

   Number of wires:              22926
   Number of wire bits:          25811
   Number of public wires:         162
   Number of public wire bits:    1972
   Number of memories:               0
   Number of memory bits:            0
   Number of processes:              0
   Number of cells:              18036
     $_ANDNOT_                    4010
     $_AND_                       1159
     $_MUX_                       1664
     $_NAND_                       896
     $_NOR_                        560
     $_NOT_                        977
     $_ORNOT_                      244
     $_OR_                        2391
     $_XNOR_                       615
     $_XOR_                       2462
     sky130_fd_sc_hd__dfxtp_2     1613
     sky130_fd_sc_hd__mux2_1      1224
     sky130_fd_sc_hd__mux4_1       221

[INFO]: ABC: WireLoad : S_4

21. Executing ABC pass (technology mapping using ABC).

21.1. Extracting gate netlist of module `\picorv32a' to `/tmp/yosys-abc-qcYJAn/input.blif'..
```



vsduser@vsdsquadron: ~/Desktop/work/tools/openlane_working_dir/openlane/designs/pic...
File Edit View Search Terminal Help

```
28. Printing statistics.

=== picorv32a ===

   Number of wires:              14596
   Number of wire bits:          14978
   Number of public wires:        1565
   Number of public wire bits:    1947
   Number of memories:               0
   Number of memory bits:            0
   Number of processes:              0
   Number of cells:              14876
     sky130_fd_sc_hd__a2111o_2       1
     sky130_fd_sc_hd__a211o_2       35
     sky130_fd_sc_hd__a211oi_2      60
     sky130_fd_sc_hd__a21bo_2      149
     sky130_fd_sc_hd__a21boi_2       8
     sky130_fd_sc_hd__a21o_2        57
     sky130_fd_sc_hd__a21oi_2      244
     sky130_fd_sc_hd__a221o_2       86
     sky130_fd_sc_hd__a22o_2      1013
     sky130_fd_sc_hd__a2bb2o_2    1748
1-yosys_4.stat.rpt
```



```
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/openlane/designs/p
icorv32a/runs/01-02_12-50/results/synthesis$ ls -lrt
total 1848
lrwxrwxrwx 1 vsduser vsduser      29 Feb  1 18:20 merged_unpadded.lef -> ../../t
mp/merged_unpadded.lef
-rw-r--r-- 1 vsduser vsduser 1889131 Feb  1 18:24 picorv32a.synthesis.v
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/openlane/designs/p
icorv32a/runs/01-02_12-50/results/synthesis$
```