I2C Master Top Modülünün OpenLane ile GDSII Üretim Süreci Raporu

Hazırlayanlar: Emre YAĞCI (360523012), Sabri SEVİNÇLİ (360523033)

Amaç

Bu çalışmada amaç, "I2C Master Top" modülünü OpenLane kullanarak ASIC akışına sokmak, sentetik, floorplan, placement ve routing adımlarını gerçekleştirip final GDS çıktısı almaktır.

Kullanılan Ortam

-WSL

-Ubuntu (Ubuntu 22.04 kullanıldı)

-Docker

-Python 3

-OpenLane

-PDK olarak Sky130A

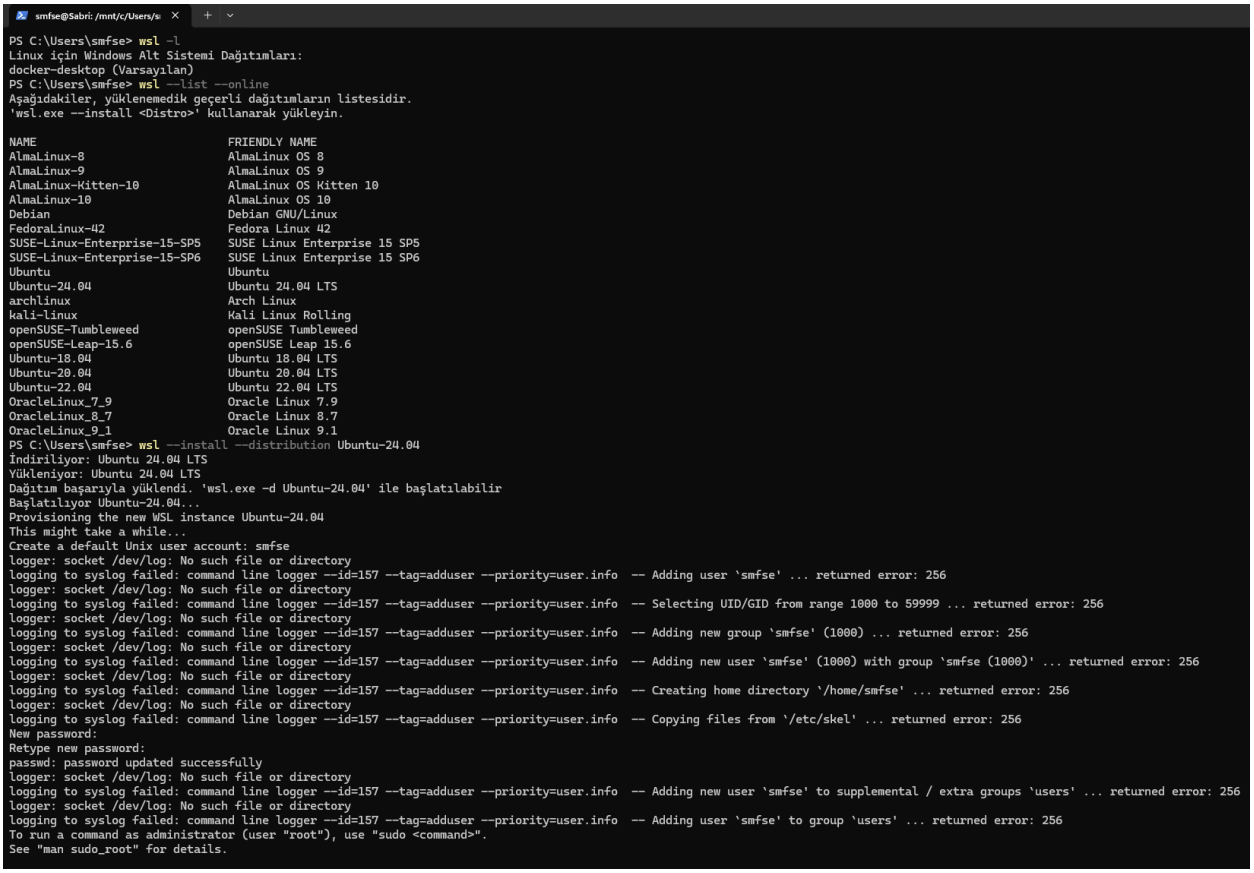
-Klayout (layout görüntüleme için)

-Volare (PDK yönetimi için)

İçindekiler

WSL Kurulumu

İlk olarak WSL'i gösterildiği üzere kurup Ubuntu 24.04 sistemini kuruyoruz. WSL bize linux komutlarını Windows üzerinden erişmemizi sağlayacak.

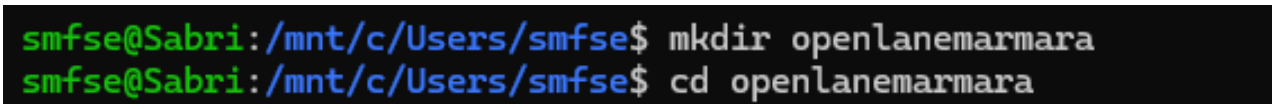


Openlane Ortamının Hazırlanması

Ardından Openlane kurmak için gerekli ortamı hazırlamalayız.

mkdir -p openlanemarmara adında openlane çalıştıracağımız bir dosya oluşturuyoruz.

cd openlanemarmara dosyaımızın içine giriyoruz.



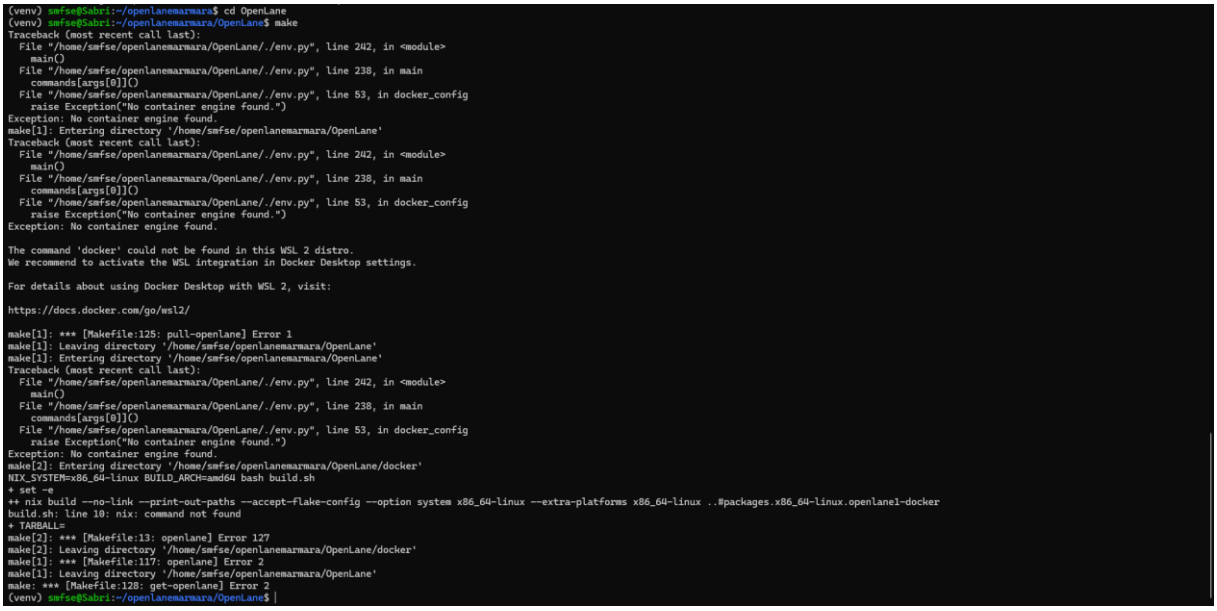
Ardından openlane kurmak için bu repozitoyu klonladık.

Openlanemarmara içine klonladığımız openlane klasörüne giriyoruz.

Make diyerek openlane i çalıştırıyoruz ama önümüze bazı hatalar çıkıyor.

Make, bizim için gerekli her şeyi kuruyor python3 volare dahil, ayrı bir işlemle Python3 ve volare kurulmasına gerek yok.

Volare ve Python, OpenLane için gerekli olan PDK dosyasını oluşturmayı sağlıyor.

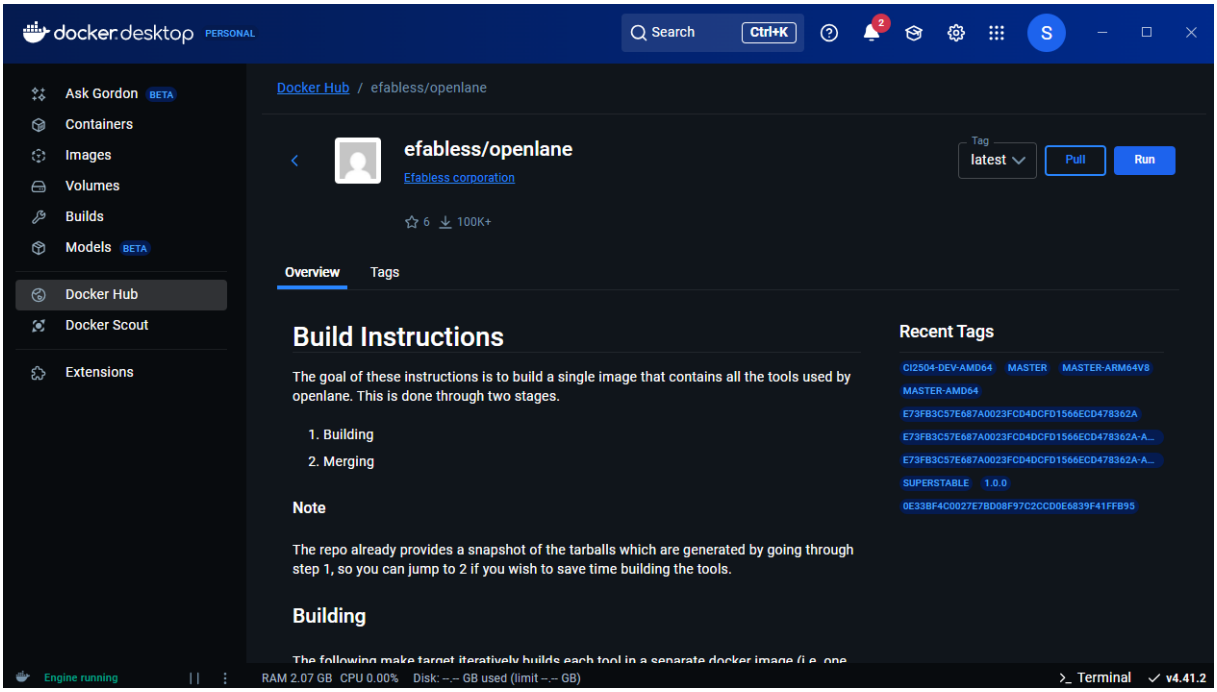


Bu hatalar genellikle docker, volare ve Python3 kurulumundan kaynaklanan hatalar olmakta. Şu anki hatamız docker dan kaynaklanmaktadır.

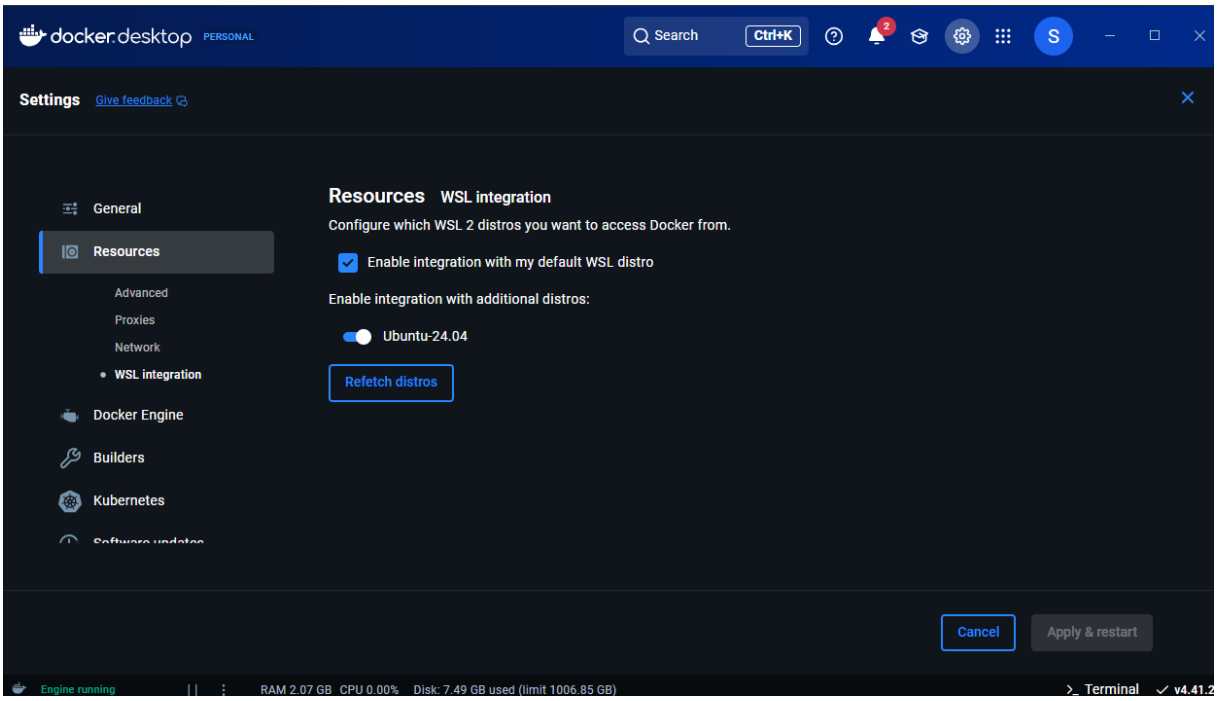
WSL üzerinden OpenLane kurduğumuz için docker ı docker desktop indirerek aktif etmemiz gerekmekte. Docker'ın Windows için olan sürümünü indirdikten sonra docker da bazı ayarlamalar yapmamız gerekti.

Docker Desktop Ubuntu için Enable Etmek

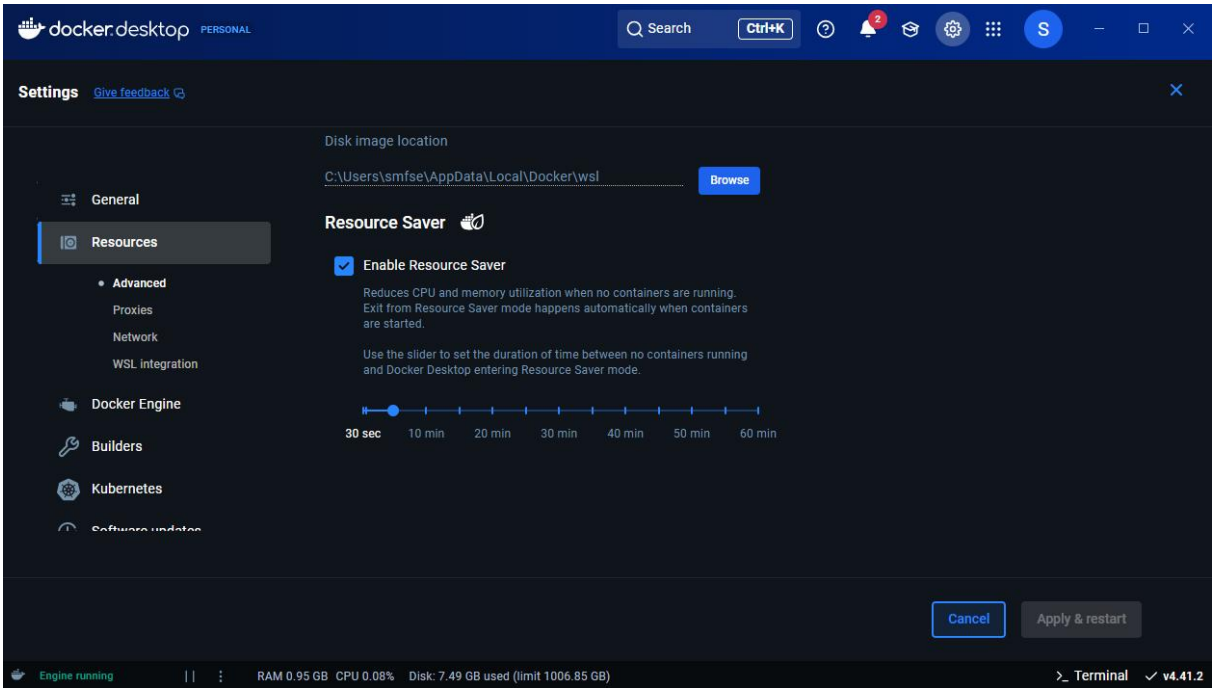
Burada docker hub içerisinde efabless openlane hub ı pull etmeliyiz.



Ardından ayarlar bölümünden resource kısmından WSL entegrasyonu sağlayıp kullandığımız Ubuntu-24.04 dosyasını Windows için entegre etmeliyiz.



Aynı zamanda resource advenced kısmında Recource saver aktif edilmeli.



Openlane kurulumundan GDSII ya kadar her aşamada docker dektop açık olmalıdır.

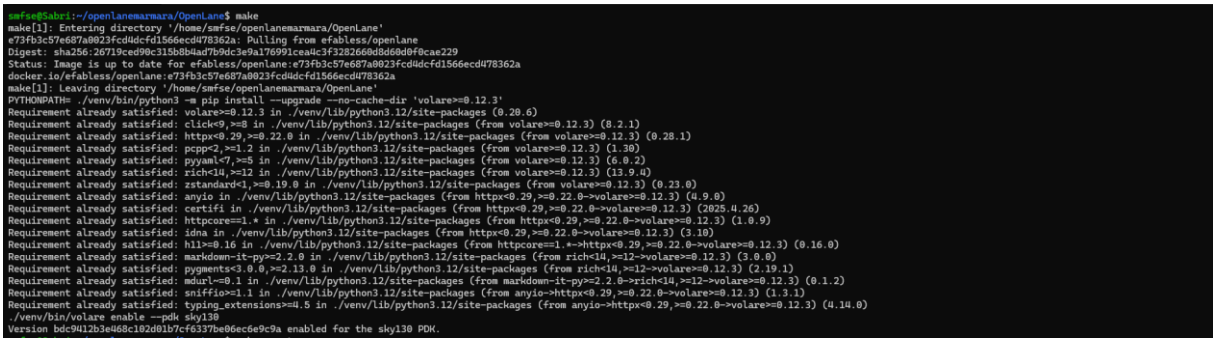
OpenLane Kurulumunun Tamamlanması Volare ile sky 130 pdk Kurulumu.

Şimdi make dediğimizde errorlar ortadan kalkmış olacak.

Burada volare ile pdk “sky 130 pdk” olarak belirlendi.

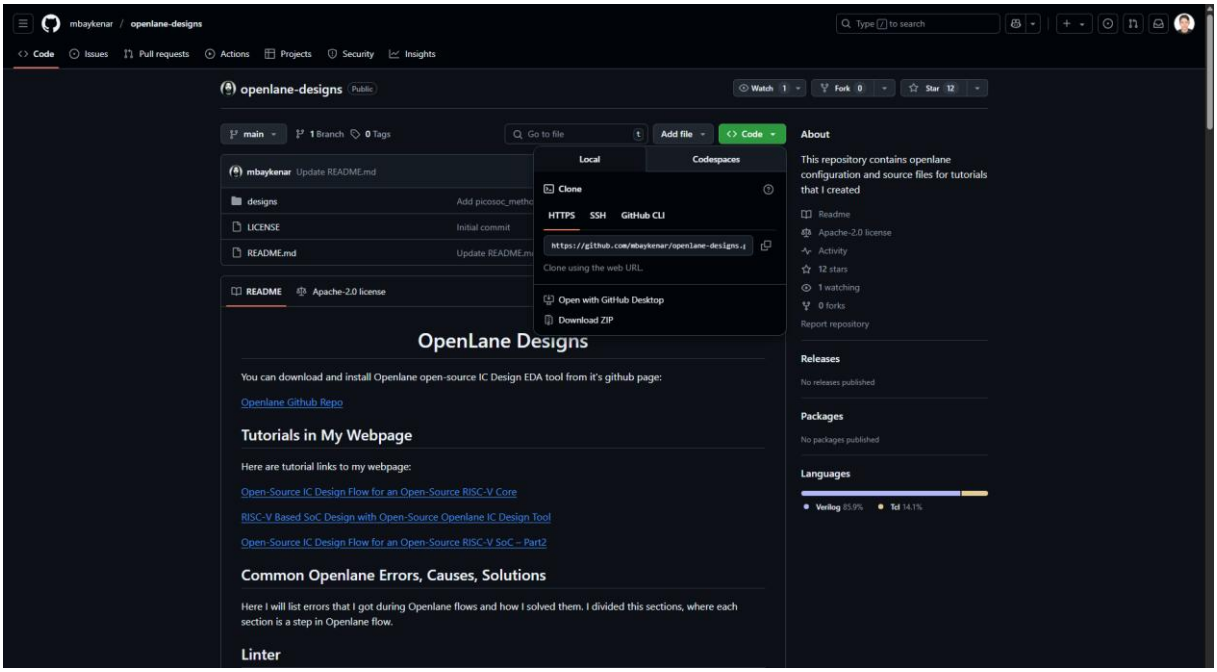
Python3.12 kuruldu

Make bizim için gerekli Python ve volare kurdu.

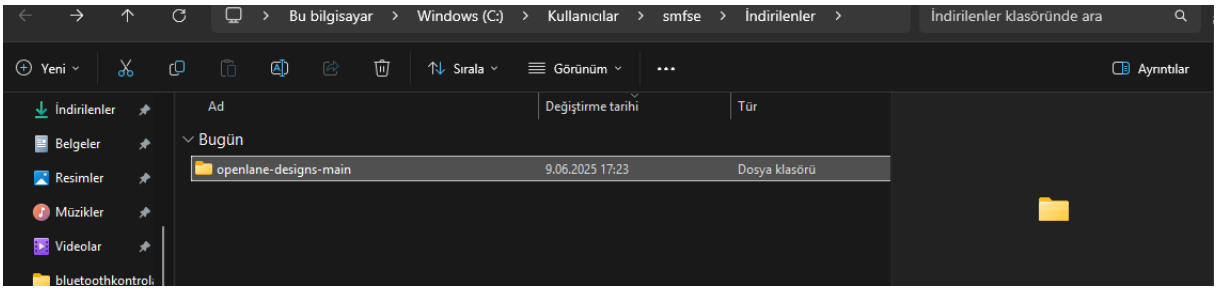


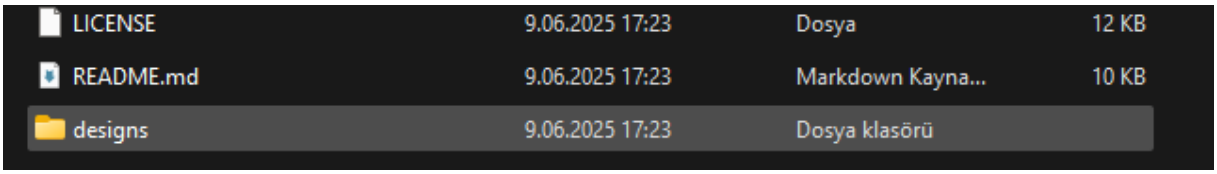
i2c_master_top RTL Dosyasını İndirmek.

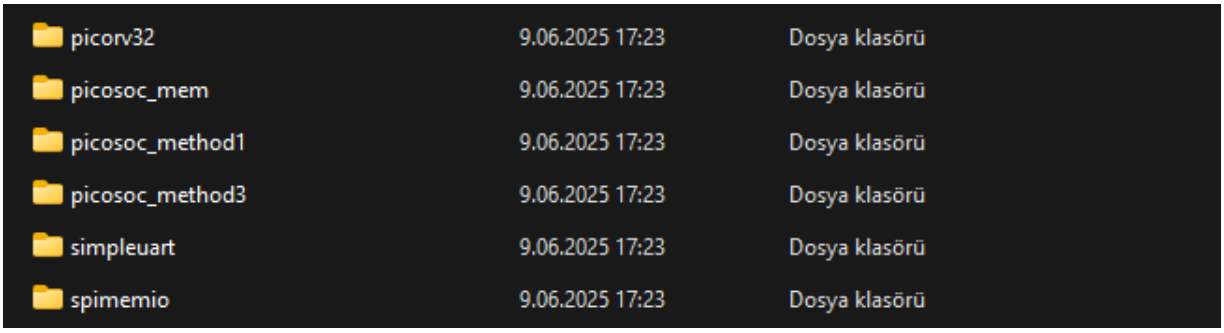
Şimdi de “i2c_master_top” rtl dosyasını Burak Aykenar’ın reposundan edindik.



Zip olarak indirip çıkarttık.







Designs içersindeki bu dosyaları

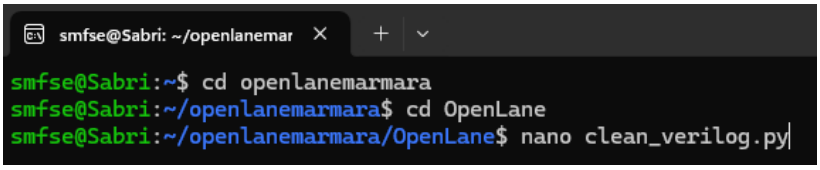
\\wsl.localhost\Ubuntu-24.04\home\smfse\openlanemarmara\OpenLane\designs

Adresine kopyaladık.

“make mount ” demekle desing içerisindeki RTL dosyalarını GDS formatına dönüşümünü başlattık. Ancak edindiğimiz bu dosyalarda verilog dosyaları hatalı yazılmış bu hatalı temizleyecek kodu gpt den istedik.

clean_verilog.py

nano ile clean_verilog.py uzantılı dosya oluşturduk.



İçerisine aşağıdaki kodu yapıştırıp çıktık.

```
import os
import re

# Hangi dizini temizleyecek
INPUT_DIR = "designs/i2c_master_top"
OUTPUT_DIR = "designs/i2c_master_top_clean"
# Çıktı dizinini oluştur
os.makedirs(OUTPUT_DIR, exist_ok=True)

# Verilog dosyalarını bul
verilog_files = []
for root, dirs, files in os.walk(INPUT_DIR):
    for file in files:
        if file.endswith(".v"):
            verilog_files.append(os.path.join(root, file))

# Kalıp: delay pattern
delay_pattern = re.compile(r'#\s*[0-9]+')

# Temizle ve yeni dosyaya yaz
for filepath in verilog_files:
    filename = os.path.basename(filepath)
    output_path = os.path.join(OUTPUT_DIR, filename)
    with open(filepath, "r", encoding="utf-8", errors="ignore") as f:
        lines = f.readlines()
    cleaned_lines = []
    skip_initial_block = False
    initial_block_level = 0
    for line in lines:
        stripped = line.strip()
```

```
        # Initial block başlarsa → skip başlat
        if re.match(r'initial\s+begin', stripped):
            skip_initial_block = True
            initial_block_level = 1
            continue

        # Eğer initial block içindeysek → end arayana kadar atla
        if skip_initial_block:
            if "begin" in stripped:
                initial_block_level += stripped.count("begin")
            if "end" in stripped:
                initial_block_level -= stripped.count("end")
                if initial_block_level <= 0:
                    skip_initial_block = False
            continue  # skip this line

        # Wait satırlarını atla
        if re.search(r'\bwait\b', stripped):
            continue

        # Delay (#) kaldır
        line_no_delay = delay_pattern.sub('', line)
        cleaned_lines.append(line_no_delay)

        # Temiz dosyayı yaz
        with open(output_path, "w", encoding="utf-8") as f:
        f.writelines(cleaned_lines)

print(f"\nTemizleme tamamlandı! → Çıktı dizini: {OUTPUT_DIR}")
```

Yapıştırıken importu düzgün kopyalamıyor onu düzenlemeyi unutmayalım.

Bu kod i2c_master_top içerisindeki verilog dosyalarını isteğimiz doğrultusunda temizleyip i2c_master_top_clean olarak kaydediyor.

Aslında "i2c_master_top" içerisinde bazı yerler farklı yazılmış 1'ler '#1' olarak verilmişti bu .py dosyası bize tek tek bunları temizlemekle uğraştırmadan bizim için temizlemiş oldu.

Çıktısından örnek bir kısım önceki ve sonraki hali:

```
        start_b:
        begin
            c_state <= #1 start_c;
            scl_oen <= #1 1'b1; // set SCL high
            sda_oen <= #1 1'b1; // keep SDA high
            sda_chk <= #1 1'b0; // don't check SDA output
        end
```

```
    start_b:
    begin
        c_state <=  start_c;
        scl_oen <=  1'b1; // set SCL high
        sda_oen <=  1'b1; // keep SDA high
        sda_chk <=  1'b0; // don't check SDA output
    end
```

```
smfse@Sabri:~/openlanemarmara/OpenLane$ nano clean_verilog.py
smfse@Sabri:~/openlanemarmara/OpenLane$ python3 clean_verilog.py

Temizleme tamamlandı! → Çıktı dizini: designs/i2c_master_top_clean
smfse@Sabri:~/openlanemarmara/OpenLane$
```

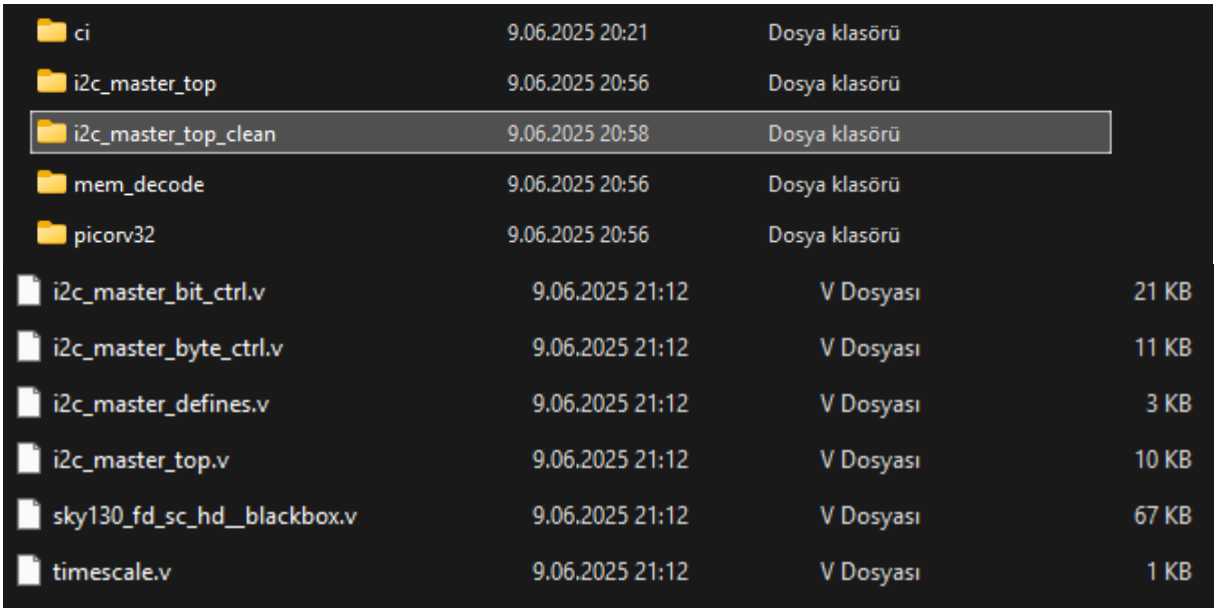
Ardından temizlenmiş dosyalar için belirttiği klasör yoluna gidiyoruz.

\\wsl.localhost\Ubuntu-24.04\home\smfse\openlanemarmara\OpenLane\designs\i2c_master_top_clean adresinde temizlenmiş verilog dosyaları mevcut olacak.

Neden Bu temizliği Yaptık

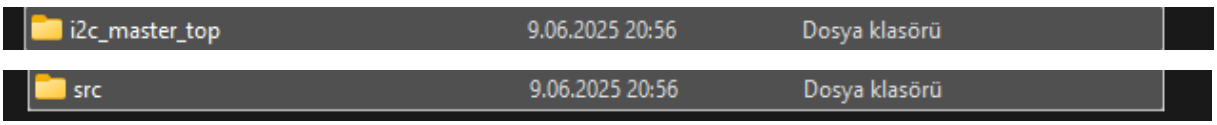
OpenLane içinde Verilator linter çalışıyor, bu aşama Verilog kodunda sentaks ve yapısal sorunları buluyor. Verilator, donanımsal olmayan kodları (yani simülasyon için olan kodları) hata olarak algılıyor.

Bizim amacımız: GDSII üretmek, bu yüzden sadece sentaks ve donanımsal anlamda geçerli olan kodlar bırakmamız gerekiyordu.



i2c_master_top_clean içeresindeki temiz dosyaları i2c_master_top içerisine eski dosyaların üzerine yapıştırıyoruz.

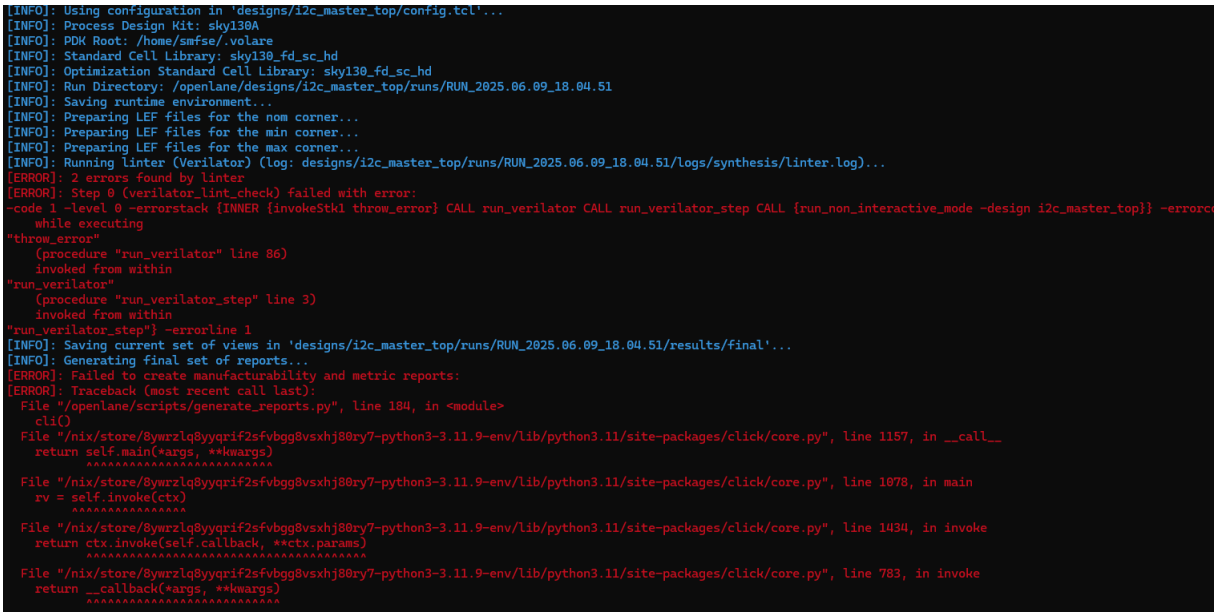
\\wsl.localhost\Ubuntu-24.04\home\smfse\openlanemarmara\OpenLane\designs\i2c_master_top\src bu adresteki klasördeki dosyaları değiştiriyoruz.



Burada Identifier olanları silmemiz gerekiyor.

i2c_master_bit_ctrl.v	9.06.2025 17:23	V Dosyası	21 KB
i2c_master_bit_ctrl.v:Zone.Identifier	9.06.2025 17:23	IDENTIFIER Dosyası	1 KB
i2c_master_byte_ctrl.v	9.06.2025 17:23	V Dosyası	11 KB
i2c_master_byte_ctrl.v:Zone.Identifier	9.06.2025 17:23	IDENTIFIER Dosyası	1 KB
i2c_master_defines.v	9.06.2025 17:23	V Dosyası	3 KB
i2c_master_defines.v:Zone.Identifier	9.06.2025 17:23	IDENTIFIER Dosyası	1 KB
i2c_master_top.sdc	9.06.2025 17:23	SDC Dosyası	1 KB
i2c_master_top.sdc:Zone.Identifier	9.06.2025 17:23	IDENTIFIER Dosyası	1 KB
i2c_master_top.v	9.06.2025 17:23	V Dosyası	11 KB
i2c_master_top.v:Zone.Identifier	9.06.2025 17:23	IDENTIFIER Dosyası	1 KB
timescale.v	9.06.2025 17:23	V Dosyası	1 KB
timescale.v:Zone.Identifier	9.06.2025 17:23	IDENTIFIER Dosyası	1 KB

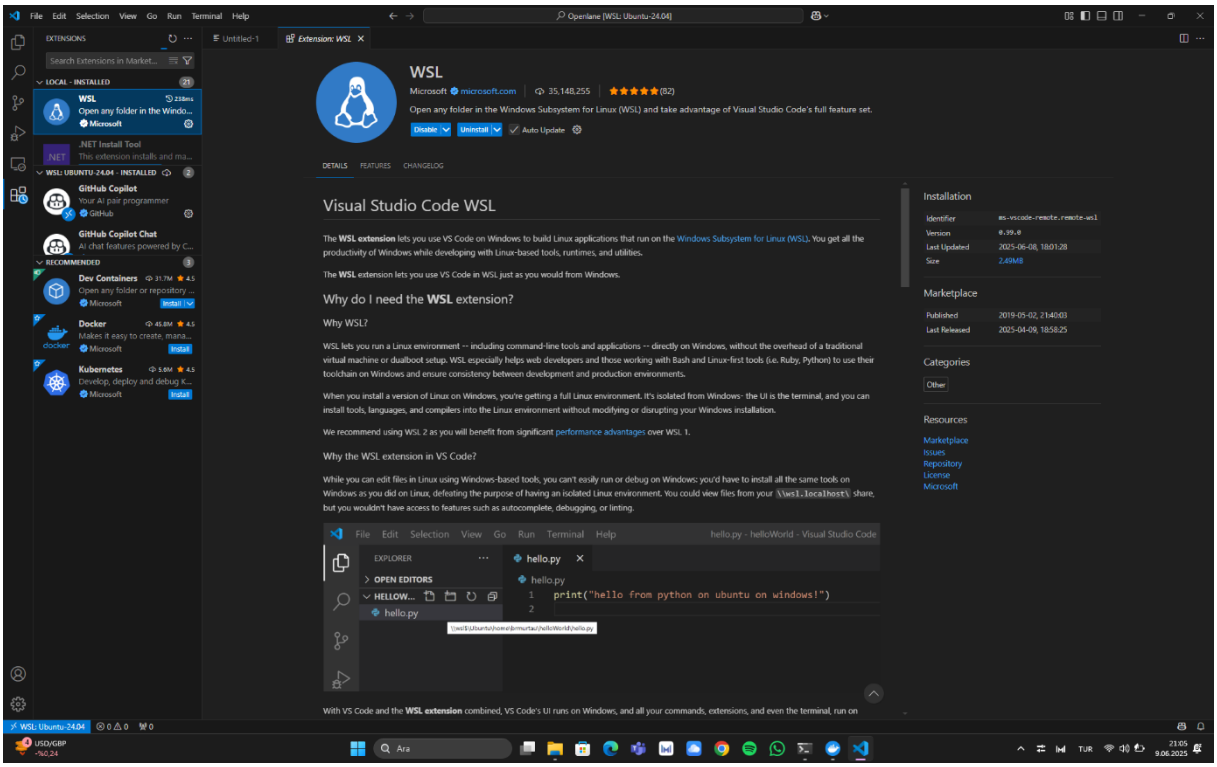
Tekrar make mount dediğimizde gene error aldık.



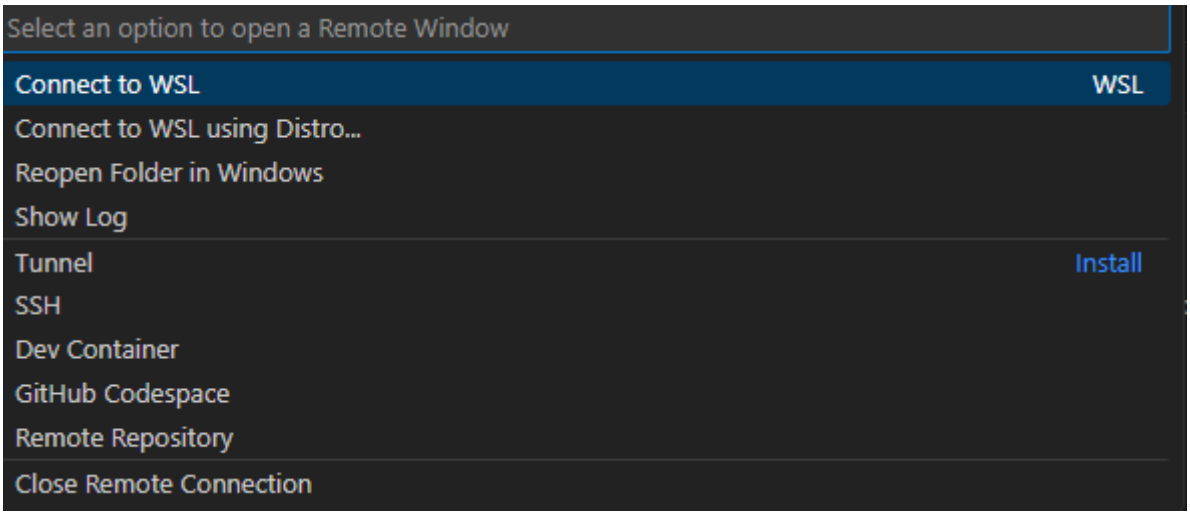
```
[INFO]: Using configuration in 'designs/i2c_master_top/config.tcl'...
[INFO]: Process Design Kit: sky130A
[INFO]: PDK Root: /home/smfse/.volare
[INFO]: Standard Cell Library: sky130_fd_sc_hd
[INFO]: Optimization Standard Cell Library: sky130_fd_sc_hd
[INFO]: Run Directory: /openlane/designs/i2c_master_top/runs/RUN_2025.06.09_18.04.51
[INFO]: Saving runtime environment...
[INFO]: Preparing LEF files for the nom corner...
[INFO]: Preparing LEF files for the min corner...
[INFO]: Preparing LEF files for the max corner...
[INFO]: Running linter (Verilator) (log: designs/i2c_master_top/runs/RUN_2025.06.09_18.04.51/logs/synthesis/linter.log)...
[ERROR]: 2 errors found by linter
[ERROR]: Step 0 (verilator_lint_check) failed with error:
-code 1 -level 0 -errorstack {INNER {invokeStk1 throw_error} CALL run_verilator CALL run_verilator_step CALL {run_non_interactive_mode -design i2c_master_top}} -errorc
    while executing
"throw_error"
    (procedure "run_verilator" line 86)
    invoked from within
"run_verilator"
    (procedure "run_verilator_step" line 3)
    invoked from within
"run_verilator_step"} -errorline 1
[INFO]: Saving current set of views in 'designs/i2c_master_top/runs/RUN_2025.06.09_18.04.51/results/final'...
[INFO]: Generating final set of reports...
[ERROR]: Failed to create manufacturability and metric reports:
[ERROR]: Traceback (most recent call last):
  File "/openlane/scripts/generate_reports.py", line 184, in <module>
    cli()
  File "/nix/store/8ywrzlq8yyqrif2sfvbgg8vsxhj80ry7-python3-3.11.9-env/lib/python3.11/site-packages/click/core.py", line 1157, in __call__
    return self.main(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/nix/store/8ywrzlq8yyqrif2sfvbgg8vsxhj80ry7-python3-3.11.9-env/lib/python3.11/site-packages/click/core.py", line 1078, in main
    rv = self.invoke(ctx)
         ^^^^^^^^^^^^^^^^
  File "/nix/store/8ywrzlq8yyqrif2sfvbgg8vsxhj80ry7-python3-3.11.9-env/lib/python3.11/site-packages/click/core.py", line 1434, in invoke
    return ctx.invoke(self.callback, **ctx.params)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/nix/store/8ywrzlq8yyqrif2sfvbgg8vsxhj80ry7-python3-3.11.9-env/lib/python3.11/site-packages/click/core.py", line 783, in invoke
    return __callback(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

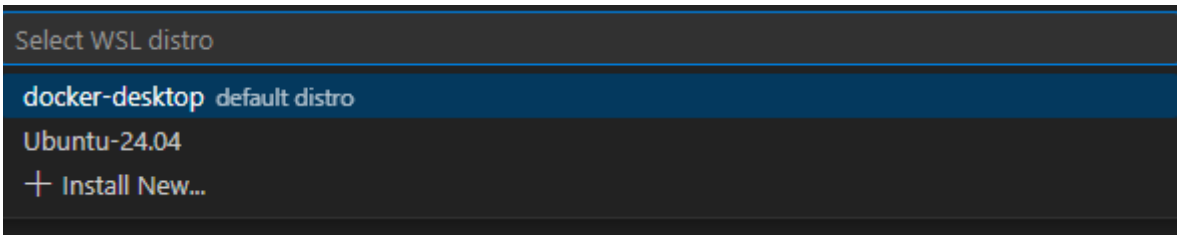
Bu hata verilog dosyasında bir kısmın eksik olmasından kaynaklanıyordu bunu verilog dosyasını elle düzenleyerek çözdük bunun için verolog dosyasının içerisine erişmemiz gerekti bunu da visual studio ile yaptık. Ancak WSL kullandığımız için vs kodun dosyalara erişimi VS code da WSL eklentisini kurarak yaptık.

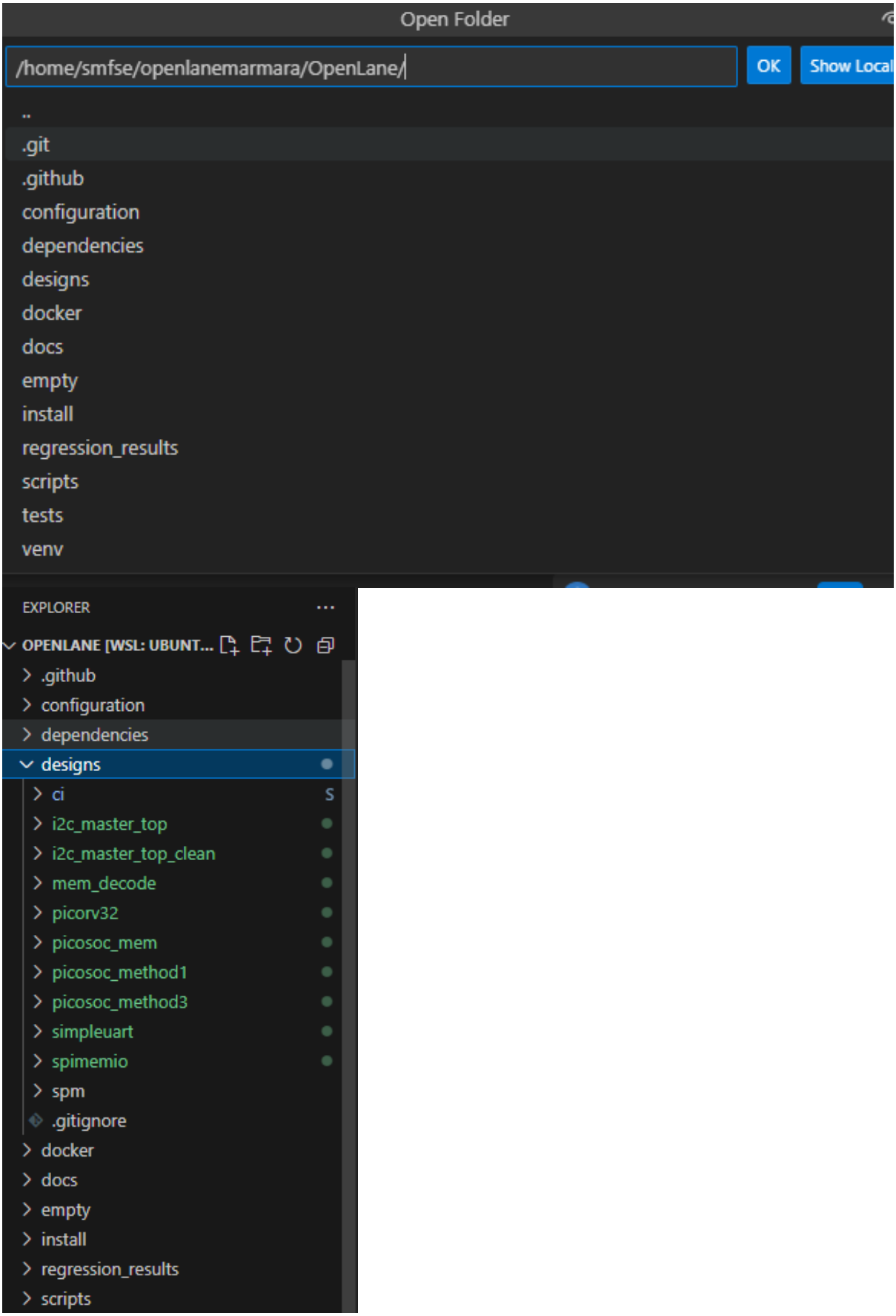
VS Code WSL Entegrasyonu



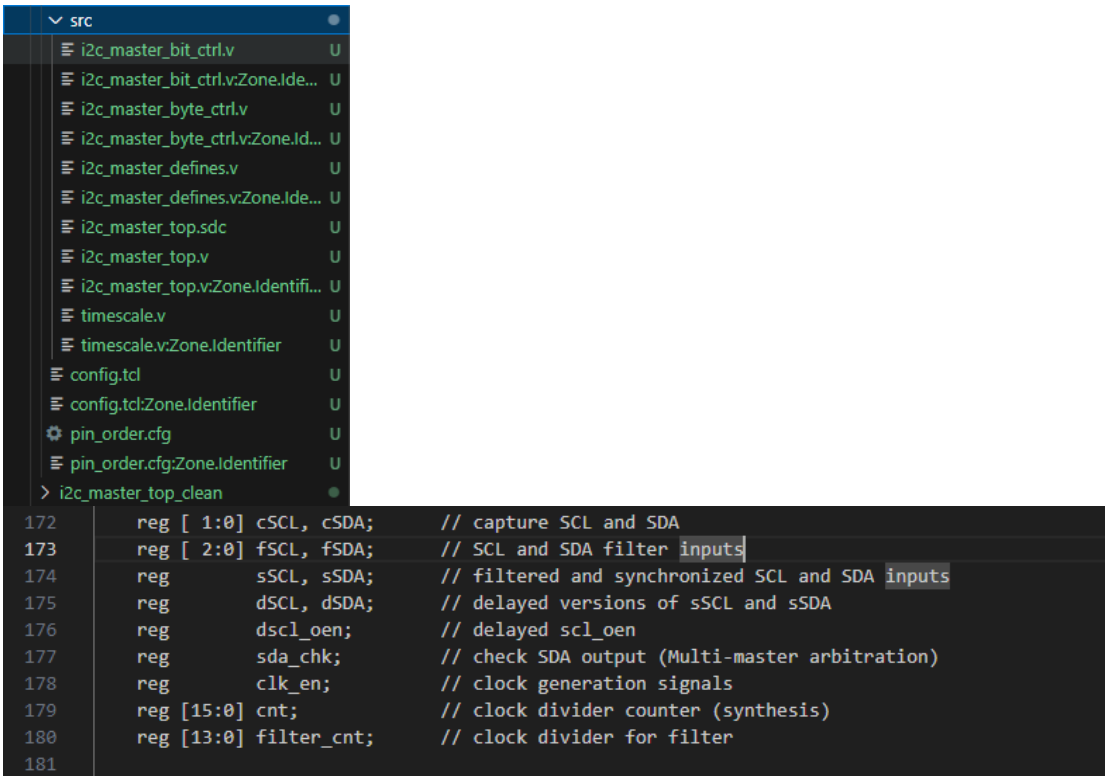
Gerekli konum resimlerdeki gösterildiği üzere gittik.



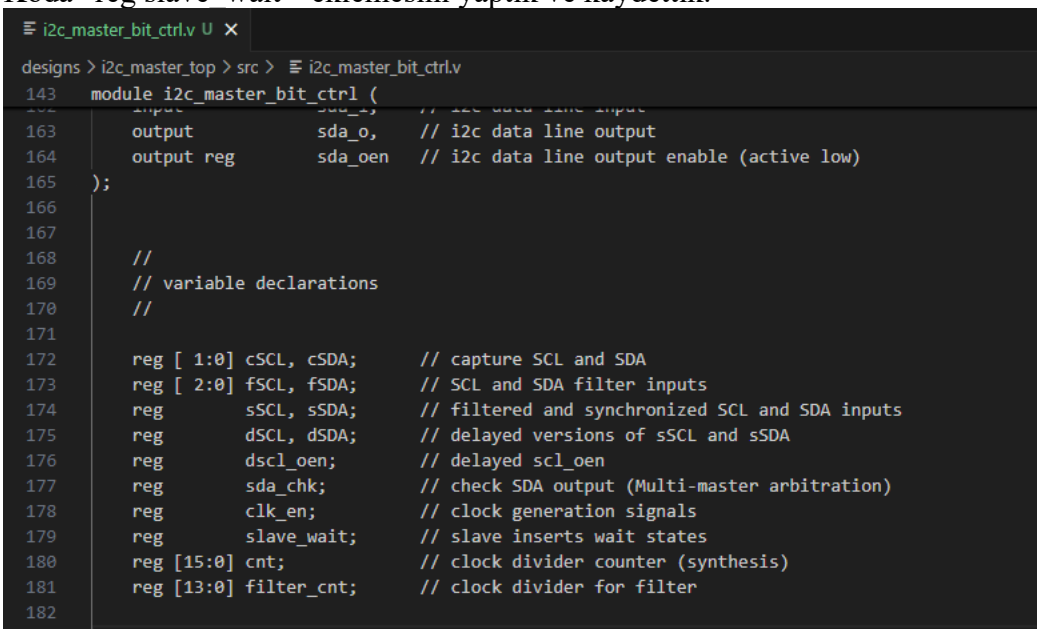



Open Folder
/home/smfse/openlanemarmara/OpenLane/
OK
Show Local
..
.git
.github
configuration
dependencies
designs
docker
docs
empty
install
regression_results
scripts
tests
venv
EXPLORER
OPENLANE [WSL: UBUNT...
.github
configuration
dependencies
designs
ci
i2c_master_top
i2c_master_top_clean
mem_decode
picorv32
picosoc_mem
picosoc_method1
picosoc_method3
simpleuart
spimemio
spm
.gitignore
docker
docs
empty
install
regression_results
scripts

Burada designs içerisindeki src dosyasının altında “i2c_master_bit_ctrl.v” verilog koduna bir ekleme yaptık.



Koda “reg slave wait ” eklemesini yaptık ve kaydettik.



Otemizliklerde delay (#) kaldırırken veya initial blokları silerken bu slave_wait tanımını ya da kullanımını yanlışlıkla kaldırabiliyor veya eksik bırakabiliyor.

Mesela slave_wait sinyali clock enable (clk_en) üretirken kullanılıyor ve bu i2c zamanlamasında çok kritik:

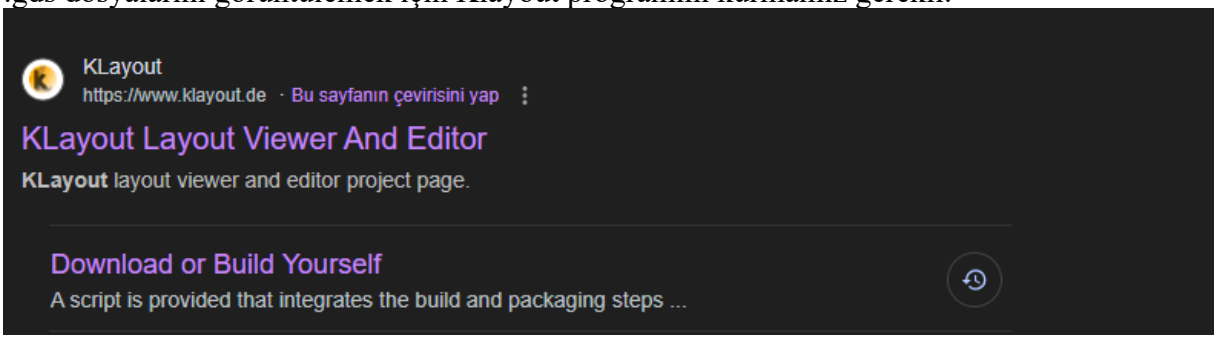
RTL to GDSII ASIC Tasarımının Başlaması

Tekrar make mount dediğimizde asicc tasarım aşamalarının hepsini yapmış olduk.

Burada flow complete diyerek gds dosyasını oluşturdu demek burada 876 tane linter uyarısı aldık. İstersek bu uyarıları tek tek elle düzeltebiliriz. Bunu config dosyasında değerleri değiştirerek yapabiliriz.

Klayout Programında Tasrımı Görme

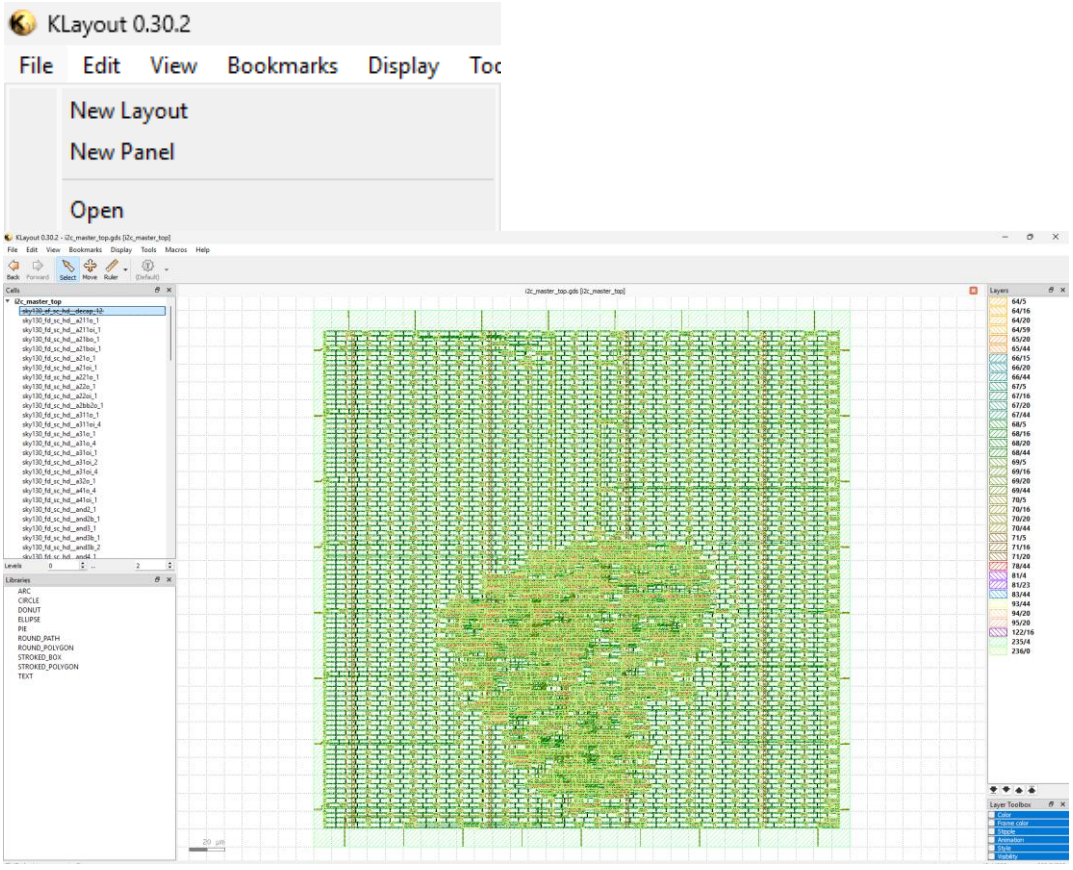
.gds dosyalarını görüntülemek için Klayout programını kurmamız gerekli.

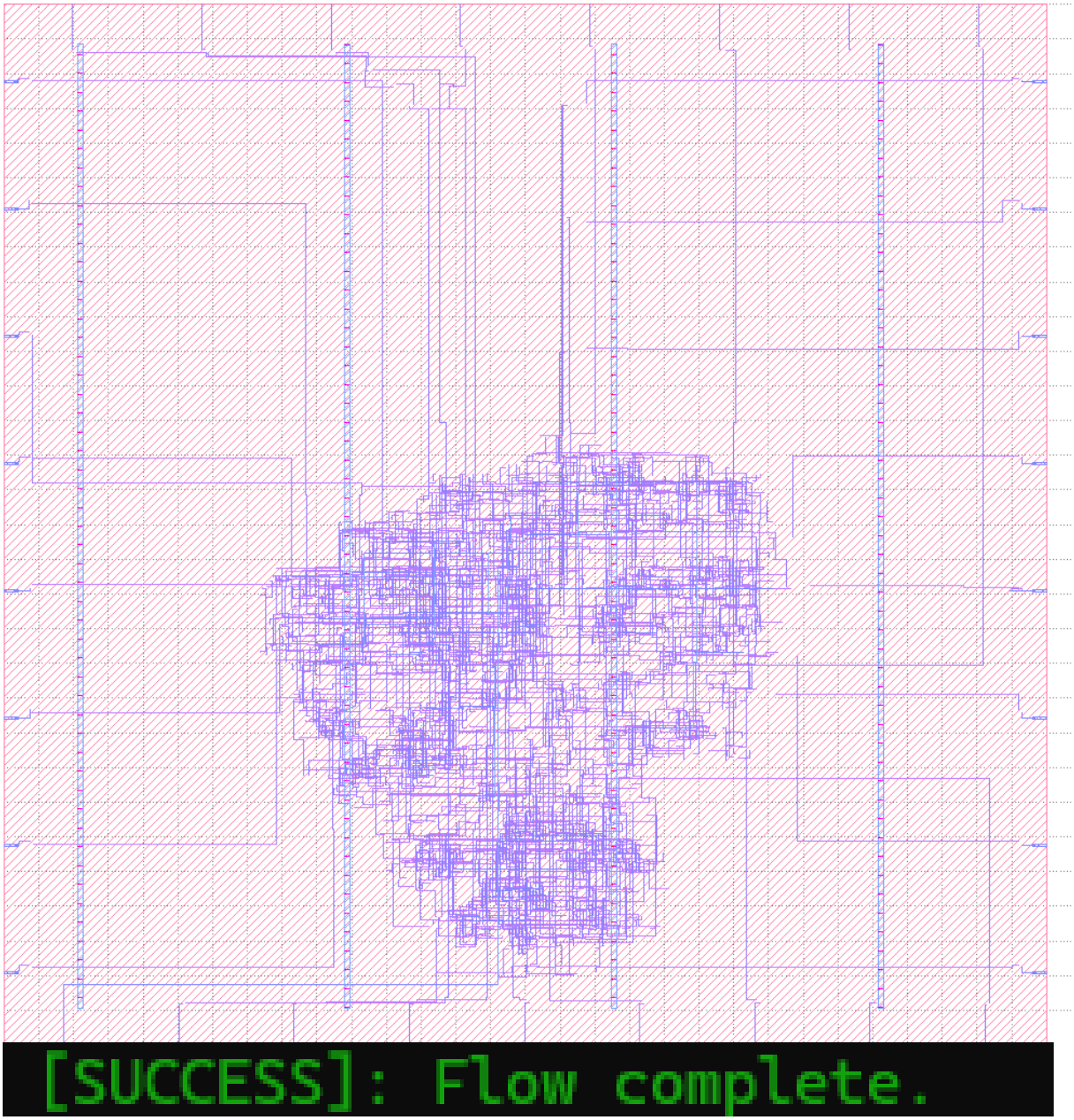


Kurulum sonrasında .gds dosyalrını görüntülememiz gerekli. Klayout programını başlattan açıp içerisinde gds dosyalarını open diyerek belirli adreslerdeki gds dosyalarını açıyoruz.

İstediğimiz gds dosyasını file open tıklayarak görebiliriz.

\\wsl.localhost\Ubuntu-24.04\home\smfse\openlanemarmara\OpenLane\designs\i2c_master_top\runs\RUN_2025.06.09_18.16.34\results\final\gds\ i2c_master_top.gds




[SUCCESS]: Flow complete.

OpenLane Akışı

- **Synthesis (Sentetik)**

Bu aşamada i2c_master_top tasarımımızdaki Verilog kodunu (yani yazdığımız mantıksal devre tanımını) donanıma çevirecek adımı yapıyoruz. Yazdığımız Verilog kodu doğrudan fiziksel devreye çevrilemez. O yüzden önce mantıksal kapı (gate) seviyesine dönüştürülüyor. Burada Yosys aracı kullanılıyor. Sonuç olarak: Mantıksal kapı seviyesinde netlist dediğimiz bağlantı listesi çıkıyor. Böylece fiziksel olarak neyin nereye bağlanacağı belirlenmiş oluyor. Linter kontrolü burada yapılıyor. Linter, Senteze başlamadan önce Verilog kodunun hatasız olduğundan emin olmak için sentezin başında yapılır.

- **Floorplan**

Bu adımda artık mantıksal bağlantılar belli olduğuna göre: Çipin üzerinde bu devreyi nereye yerleştireceğimizi belirliyoruz. Çipin boyutunu ayarlıyoruz. Giriş çıkış pinlerinin kenarlarda nereye konulacağını belirliyoruz. Güç ve toprak hatlarını yerleştiriyoruz.

- **Placement**

Mantıksal hücreleri (AND kapısı, Flip-Flop gibi) fiziksel olarak çip üzerinde uygun yerlere yerleştiriyoruz. Hücrelerin sıkışmaması ve doğru bağlantı yapabilmesi için boşluklar ve düzen sağlıyoruz. Placement sonrası devre artık çip üzerinde nerede duracak tamamen belli olmuş oluyor. Burada STA yapılır.

Static Timing Analysis (STA)

STA çip tasarımında zamanlama (timing) hatalarını bulmak için yapılan bir kontroldür. Saat (Clock) sinyaliyle uyumlu çalışması gereken yolların geç kalmaması gerekir. Buna setup time ve hold time diyoruz. STA aşamasında araç, her yolun geçiş süresini (delay) hesaplıyor. Hiçbir yolun geç kalıp kalmadığını geçersiz bir veri üretmediğini veya bozulma yaşanmadığını kontrol ediyor. Eğer zamanlama problemi (timing violation) çıkarsa, placement veya routing tekrar ayarlanıyor. Kısacası STA aşaması sayesinde çipin tüm sinyallerinin doğru zamanda çalıştığından emin oluyoruz.

- **CTS (Clock Tree Synthesis)**

Çip içinde clock sinyali kullanılıyor ki tüm işlemler anda gerçekleşsin. Clock sinyali her hücreye eşit sürede ulaşacak şekilde özel yollar (clock tree) oluşturuluyor. Böylece tüm hücreler aynı anda çalışabiliyor**.** Clock skew dediğimiz zaman farkları minimize ediliyor. Burada da STA yapılır.

- **Routing**

Artık hücrelerin yeri belli oldu. Şimdi hücreleri metal yollarla bağlanmalı. Çipin hücreleri metal yollarla bağlanıyor. Hem global routing hem de detaylı routing yapılıyor. Routing bitince çipin tüm bağlantıları tamamlanmış oluyor. Burada da STA yapılır.

- **DRC ve LVS Kontrolleri**

Bu aşamada tasarımımızı kontrol ediyoruz:

DRC (Design Rule Check): Çipin üretim kurallarına uyup uymadığını kontrol ediyoruz (minimum çizgi kalınlığı, mesafe vs.).

LVS (Layout vs Schematic): Tasarımın mantıksal olarak doğru olup olmadığını kontrol ediyoruz. Yani Verilog'taki mantıkla fiziksel çizim aynı mı diye bakıyoruz.

- **GDSII Üretimi**

Son olarak tasarımımızı GDSII formatına çeviriyoruz. GDSII, fabrikaya gönderilecek dosyadır. Bu dosya doğrudan çipin üretiminde kullanılır. GDSII dosyasını ayrıca Klayout aracı ile açıp görsel olarak da inceledik. Böylece GDSII dosyamız hazır hale geldi.

Sonuç ve Değerlendirme

Bu projede openlane ile asic design yapmayı öğrendik. Bunu yaparken karşılaştığımız "docker, volare, pyhton3.12" sorunlarını çözmeyi öğrendik. I2c_master_top RTL dosyasında Verilog koldarındaki hataları temizlemeyi öğrendik. OpenLane ilea sic tasarımında ne aşamaların yapıldığını öğrendik. WSL ile Ubuntu kurup linux kullanmayı öğrendik. Klayout ile GDSII dosya formatını görmeyi öğrendik. Yani kısaca: I2C Master Top modülünü başarıyla GDSII seviyesine kadar getirdik. Flow sırasında hem tasarım hem de yazılım tarafında pek çok hata ve problem ile karşılaşıp çözdük. Clean Verilog işlemi sayesinde daha stabil bir OpenLane flow elde ettik. Klayout kullanarak GDSII dosyasının görsel kontrolünü de yaptık.

Kaynakça

Docker dektop download. (2025, Haziran 1). Docker dektop: https://desktop.docker.com/win/main/amd64/Docker%20Desktop%20Installer.exe?utm_source=docker&utm_medium=webreferral&utm_campaign=docs-driven-download-win-amd64 adresinden alındı

efabless github. (2025, Haziran 1). Github: https://github.com/efabless/caravel adresinden alındı

Kaylayout download. (2025, Haziran 1). Klayout: https://www.klayout.org/downloads/Windows/klayout-0.30.2-win64-install.exe adresinden alındı

mbaykenar github. (2025, Haziran 1). github: https://github.com/mbaykenar/openlane-designs/tree/main/designs adresinden alındı

Mehmet burak aykenar chip tasarımı dersleri. (2025, Haziran 1). Youtube: https://youtu.be/haTgQTjgXj8?si=y1Dap7YCeUvpbkBf adresinden alındı

The-OpenROAD-Project. (2025, Haziran 1). GitHub: https://github.com/The-OpenROAD-Project/OpenLane adresinden alındı

Teşekkürler

Dr. Öğr. Üyesi Sezen BAL

Öğr. Emre YAĞCI 360523012

Öğr. Sabri SEVİNÇLİ 360523033