Openlane2

Nix-based Installation Steps (Windows 10 +) and Smoke Test

Prerequisites

- Windows 10 version 2004 (Build 19041+) or Windows 11
- WSL 2 enabled and up-to-date
- Quad-core CPU (≥ 2.0 GHz) and ≥ 8 GiB RAM (16 GiB recommended)

1. Set up WSL 2 (Optional if you are using Linux)

1. Open PowerShell as Administrator.
2. Install Ubuntu:

   ```
   wsl --install -d Ubuntu
   ```

3. Verify WSL version:

   ```
   wsl --list --verbose
   ```

   Ensure your Ubuntu distribution is running on version 2.
4. Launch Ubuntu from the Start Menu.

2. Install Nix

1. Update Ubuntu's package index and install `curl`:

   ```
   sudo apt-get update
   sudo apt-get install -y curl
   ```

2. Run the official installer with OpenLane's cache settings:

```
curl --proto 'https' --tlsv1.2 -sSf -L https://install.determinate.systems/nix
\
  | sh -s -- install --no-confirm --extra-conf "
     extra-substituters = https://openlane.cachix.org
     extra-trusted-public-keys = openlane.cachix.org-1:qqdwh+QMNGmZA
uyeQJTH9ErW57OWSvdtuwfBKdS254E=
   "
```

3. **Close and reopen the Ubuntu terminal so Nix's environment is loaded.**

3.Enable OpenLane's Binary Cache

If you already have Nix installed, run:

```
nix-env -f "<nixpkgs>" -iA cachix
sudo env PATH="$PATH" cachix use openlane
sudo pkill nix-daemon
```

This avoids rebuilding all dependencies from source.

4. Clone the OpenLane Repository

```
git clone https://github.com/efabless/openlane2
```

5. Enter the Nix Shell

1. Change into the project directory:

   ```
   cd openlane2
   ```

2. Launch the development environment:

   ```
   nix-shell
   ```

   The first run may take ~10 minutes while binaries download from the cache.

6. Run the Smoke Test

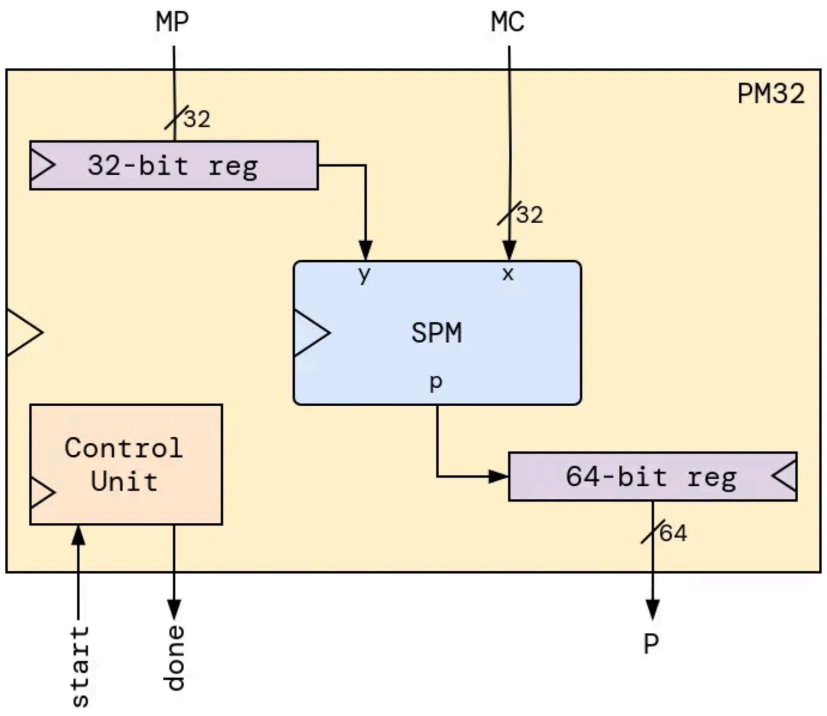
Within the `nix-shell`, execute:

```
openlane --smoke-test
```

A successful run confirms your Nix-based OpenLane setup is operational.

PM32 example:

We are going to use a simple design: a 32-bit parallel multiplier which performs a simple multiplication between MP and MC and outputs the product on a bus P. MP32 uses a serial-by-parallel multiplier, a serializer, a deserializer, and a control unit. The block diagram of the PM32 is shown in the next figure



The serial-by-parallel signed 32-bit multiplier **SPM** block inside the **PM32** performs a simple shift-add algorithm, where the parallel input X is multiplied by each bit of the serial input **y** as it is shifted in. The product is generated and serially output on the wire **p**. Check this paper for more information about the **SPM**.

Steps :

Step 1 :

```
mkdir -p ~/my_designs/pm32
```

Step 2:

Configuration

Designs in OpenLane have configuration files. A configuration file contains values set by the user for various openlane.config.Variable(s). With them, you control the flows. This is the configuration file for the pm32 design:

```
touch ~/my_designs/pm32/config.json
```

Add the following configurations in config.json using any text editor

▼ **config.json**

```
{
  "DESIGN_NAME": "pm32",
  "VERILOG_FILES": ["dir::pm32.v", "dir::spm.v"],
  "CLOCK_PERIOD": 25,
  "CLOCK_PORT": "clk"
}
```

Step 3: RTL

```
touch ~/my_designs/pm32/pm32.v ~/my_designs/pm32/spm.v
```

open **pm32.v** and **spm.v copy the following code into them.**

The source RTL of the design will consist of 2 files, which are **spm.v** and **pm32.v**.

▼ **PM32.v**

```
// A signed 32×32 Multiplier utilizing SPM
//
// Copyright 2016, mshalan@aucegypt.edu

`timescale          1ns/1ps
`default_nettype    none

module pm32 (
    input wire          clk,
    input wire          rst,
    input wire          start,
    input wire  [31:0]  mc,
    input wire  [31:0]  mp,
    output reg  [63:0]  p,
    output wire         done
);
    wire        pw;
    reg [31:0]  Y;
    reg [7:0]   cnt, ncnt;
    reg [1:0]   state, nstate;

    localparam  IDLE=0, RUNNING=1, DONE=2;

    always @(posedge clk or posedge rst)
        if(rst)
            state  <= IDLE;
        else
            state <= nstate;

    always @*
        case(state)
            IDLE    :   if(start) nstate = RUNNING; else nstate = IDLE;
            RUNNING :   if(cnt == 64) nstate = DONE; else nstate = RUNNING;
            DONE    :   if(start) nstate = RUNNING; else nstate = DONE;
            default :   nstate = IDLE;
        endcase

    always @(posedge clk)
        cnt <= ncnt;

    always @*
        case(state)
            IDLE    :   ncnt = 0;
            RUNNING :   ncnt = cnt + 1;
            DONE    :   ncnt = 0;
            default :   ncnt = 0;
        endcase

    always @(posedge clk or posedge rst)
        if(rst)
            Y <= 32'b0;
        else if((start == 1'b1))
            Y <= mp;
        else if(state==RUNNING)
            Y <= (Y >> 1);

    always @(posedge clk or posedge rst)
        if(rst)
            p <= 64'b0;
        else if(start)
            p <= 64'b0;
        else if(state==RUNNING)
            p <= {pw, p[63:1]};

    wire y = (state==RUNNING) ? Y[0] : 1'b0;

    spm #(.SIZE(32)) spm32(
        .clk(clk),
        .rst(rst),
        .x(mc),
        .y(y),
        .p(pw)
    );

    assign done = (state == DONE);

endmodule
```

▼ **SPM.v**

```
/*
    A Serial-Parallel Multiplier (SPM)
    Modeled after the design outlined by ATML for their
    AT6000 FPGA in application notes DOC0716 and DOC0529:
        - https://ww1.microchip.com/downloads/en/AppNotes/DOC0529.PDF
        - https://ww1.microchip.com/downloads/en/AppNotes/DOC0716.PDF

    Implemented by mshalan@aucegypt.edu, 2016
*/

`timescale          1ns/1ps
`default_nettype    none

module spm #(parameter SIZE = 32)(
    input wire              clk,
    input wire              rst,
    input wire              y,
    input wire [SIZE-1:0]   x,
    output wire             p
);
    wire [SIZE-1:1]     pp;
    wire [SIZE-1:0]     xy;
    genvar i;

    CSADD csa0 (.clk(clk), .rst(rst), .x(x[0]&y), .y(pp[1]), .sum(p));

    generate
        for(i=1; i<SIZE-1; i=i+1) begin
            CSADD csa (.clk(clk), .rst(rst), .x(x[i]&y), .y(pp[i+1]), .sum(pp[i]));
        end
    endgenerate

    TCMP tcmp (.clk(clk), .rst(rst), .a(x[SIZE-1]&y), .s(pp[SIZE-1]));

endmodule


// Carry Save Adder
module CSADD(
    input wire  clk,
    input wire  rst,
    input wire  x,
    input wire  y,
    output reg  sum
);

    reg sc;

    // Half Adders logic
    wire hsum1, hco1;
    assign hsum1 = y ^ sc;
    assign hco1 = y & sc;

    wire hsum2, hco2;
    assign hsum2 = x ^ hsum1;
    assign hco2 = x & hsum1;
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            sum <= 1'b0;
            sc <= 1'b0;
        end
        else begin
            sum <= hsum2;
            sc <= hco1 ^ hco2;
        end
    end
endmodule

// 2's Complement
module TCMP (
    input wire  clk,
    input wire  rst,
    input wire  a,
    output reg  s
);

    reg z;

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            s <= 1'b0;
            z <= 1'b0;
        end
        else begin
            z <= a | z;
            s <= a ^ z;
        end
    end

endmodule
```

Step 4:

```
cd openlane2
```

then type this command and enter

```
nix-shell
```

Then you will see

> 💡 [nix-shell:~/openlane2]$

Step 5: Run the following command

```
openlane ~/my_designs/pm32/config.json
```

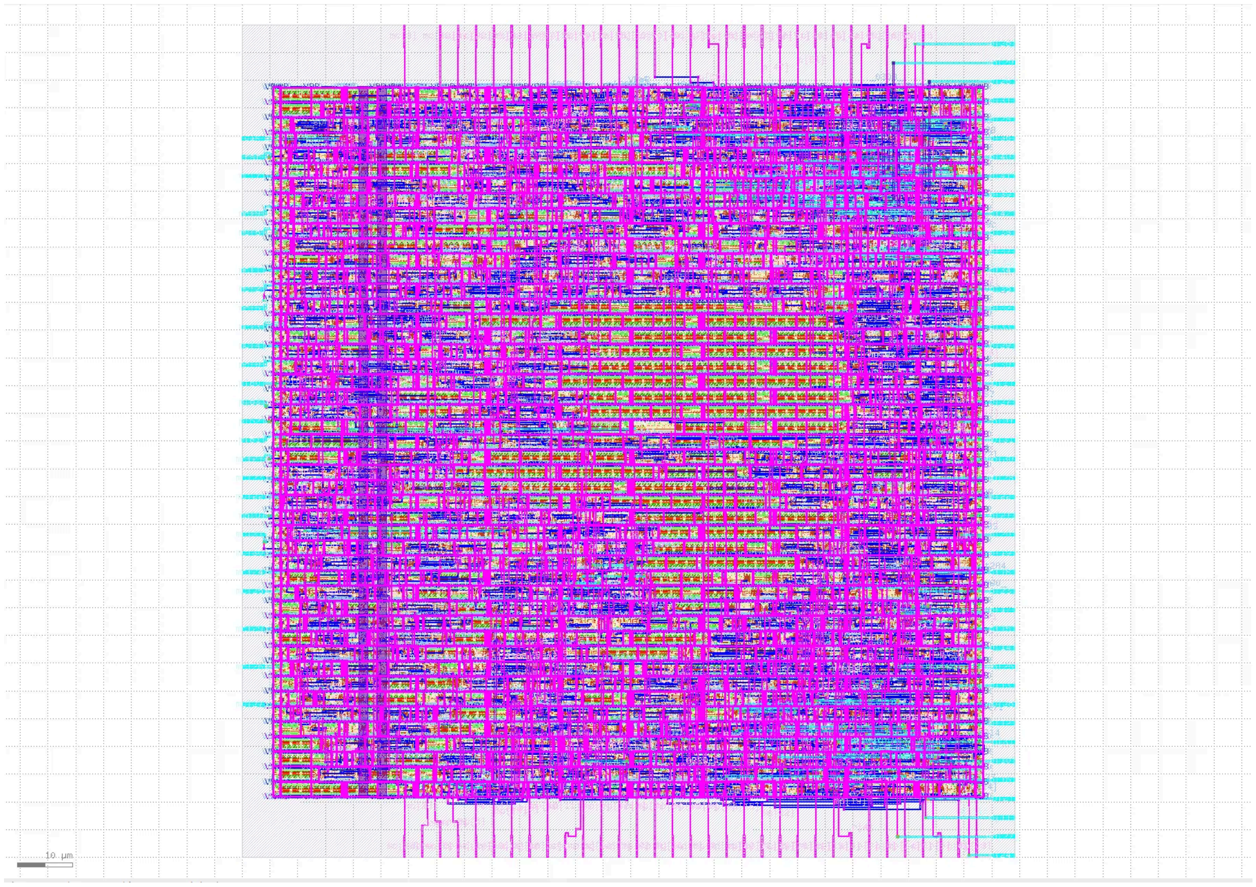
Checking the results

Viewing the Layout

To open the final GDSII layout run this command:

```
openlane --last-run --flow openinklayout ~/my_designs/pm32/config.json
```

This opens KLayout and you should be able to see the following:



If you wish to view the layout in the OpenROAD GUI, try this command instead:

```
openlane --last-run --flow openinopenroad ~/my_designs/pm32/config.json
```

For detailed explanation please refer to the Openlane2 documentation.