Challenge #9

VLM Accelerator

Core task: Module filter (extract only "execution stage" from prompt)

Inputs and Outputs:

Input: [fetch, decode, execute, memory, writeback]

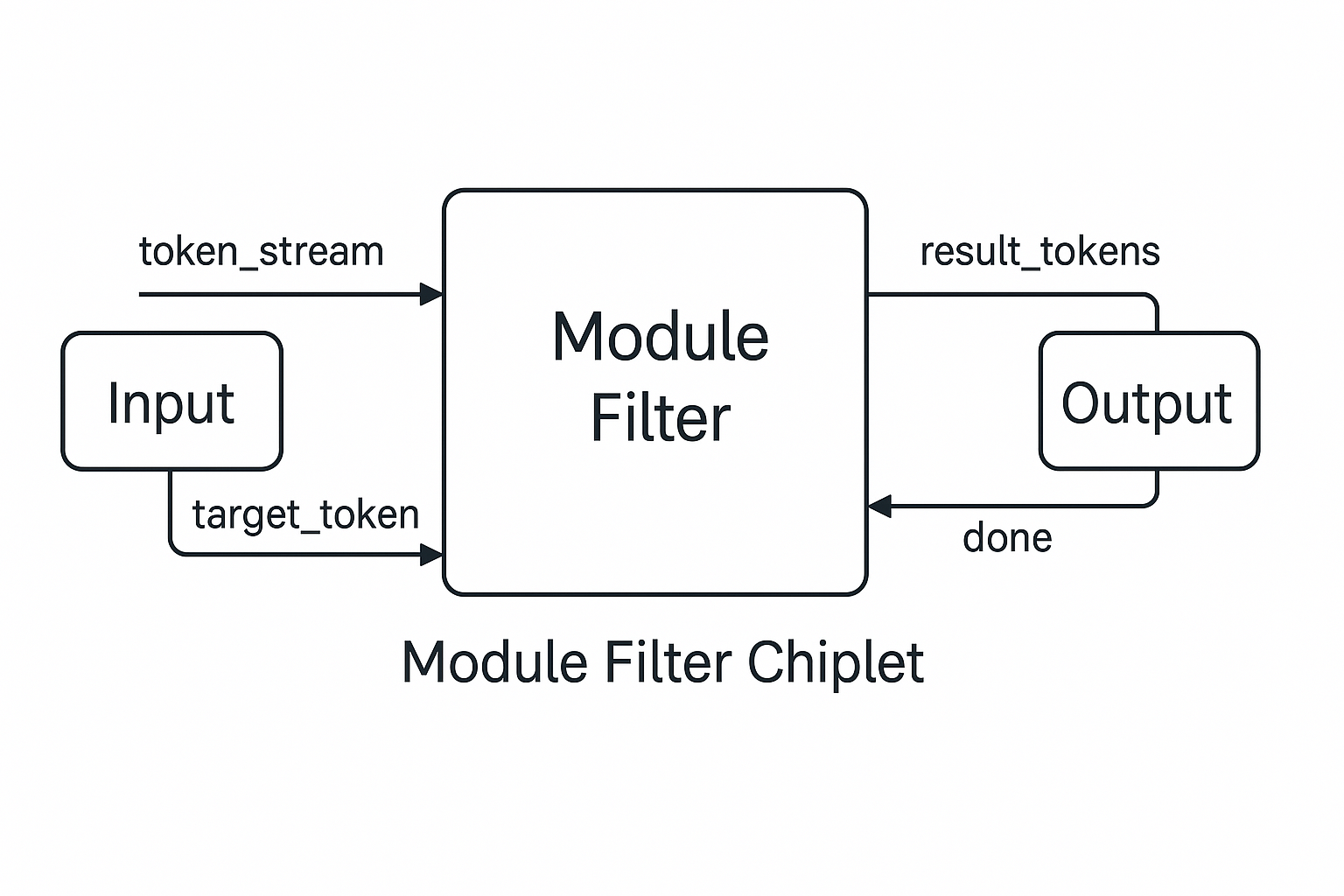
Target: "execute"

Output: execute

input [7:0] token\_stream [0:255]; // incoming prompt tokens

input [7:0] target\_token [0:7]; // "execute"

output logic [7:0] result\_tokens [0:N]; // matching block only

module filter code:

This version: Matches a target word (e.g., "execute") in a stream of incoming ASCII characters.

Extracts that module block until an end marker (e.g., ;, {, or newline).

module module\_filter (

input clk,

input rst,

input [7:0] token\_in,

input token\_valid,

output reg [7:0] token\_out,

output reg token\_out\_valid,

output reg match\_active

);

reg [7:0] buffer [0:6]; // Recent 7 tokens (stream window)

reg [7:0] TARGET [0:6]; // "execute"

integer i;

reg [2:0] state;

reg matched;

localparam S\_IDLE = 0;

localparam S\_OUTPUTTING = 1;

initial begin

TARGET[0] = "e";

TARGET[1] = "x";

TARGET[2] = "e";

TARGET[3] = "c";

TARGET[4] = "u";

TARGET[5] = "t";

TARGET[6] = "e";

end

always @(posedge clk or posedge rst) begin

if (rst) begin

token\_out <= 0;

token\_out\_valid <= 0;

match\_active <= 0;

matched <= 0;

state <= S\_IDLE;

for (i = 0; i < 7; i = i + 1)

buffer[i] <= 0;

end else begin

token\_out\_valid <= 0;

// Shift in tokens

if (token\_valid) begin

for (i = 6; i > 0; i = i - 1)

buffer[i] <= buffer[i - 1];

buffer[0] <= token\_in;

end

case (state)

S\_IDLE: begin

if (token\_valid) begin

matched = 1;

for (i = 0; i < 7; i = i + 1)

if (buffer[i] != TARGET[i])

matched = 0;

if (matched) begin

state <= S\_OUTPUTTING;

match\_active <= 1;

end

end

end

S\_OUTPUTTING: begin

if (token\_valid) begin

token\_out <= token\_in;

token\_out\_valid <= 1;

if (token\_in == "}") begin

state <= S\_IDLE;

match\_active <= 0;

end

end

end

endcase

end

end

endmodule

This testbench:

* Simulates a token stream that includes several module blocks.
* Searches for the module named "execute".
* Verifies that the filtered result matches the expected output.

Code:

`timescale 1ns / 1ps

module tb\_module\_filter;

reg clk;

reg rst;

reg [7:0] token\_in;

reg token\_valid;

wire [7:0] token\_out;

wire token\_out\_valid;

wire match\_active;

module\_filter uut (

.clk(clk),

.rst(rst),

.token\_in(token\_in),

.token\_valid(token\_valid),

.token\_out(token\_out),

.token\_out\_valid(token\_out\_valid),

.match\_active(match\_active)

);

// Clock generator

always #5 clk = ~clk;

// Test input string

string input\_code;

integer i;

reg started\_printing;

initial begin

$dumpfile("dump.vcd");

$dumpvars(0, tb\_module\_filter);

clk = 0;

rst = 1;

token\_valid = 0;

token\_in = 0;

started\_printing = 0;

#10 rst = 0;

// Change this string to test other modules

input\_code = "fetch{a=1;}decode{b=2;}execute{c=3;}memory{d=4;}";

for (i = 0; i < input\_code.len(); i = i + 1) begin

@(posedge clk);

token\_valid <= 1;

token\_in <= input\_code[i];

end

@(posedge clk);

token\_valid <= 0;

token\_in <= 0;

// Let output finish

repeat (20) @(posedge clk);

$display("\n\nDone.");

$finish;

end

// Print the filtered output to console

always @(posedge clk) begin

if (token\_out\_valid) begin

if (!started\_printing) begin

$display("\nFiltered Output:");

started\_printing = 1;

end

$write("%c", token\_out); // instead of %s

end

end

endmodule

OpenLane Code:

set ::env(DESIGN\_NAME) module\_filter

set ::env(VERILOG\_FILES) "$::env(DESIGN\_DIR)/src/module\_filter.v"

# Synthesis settings

set ::env(SYNTH\_STRATEGY) "AREA 0"

set ::env(PL\_TARGET\_DENSITY) 0.50

set ::env(CLOCK\_PERIOD) "10"

set ::env(CLOCK\_PORT) "clk"

# Floorplan

set ::env(DIE\_AREA) "0 0 300 300"

set ::env(CORE\_AREA) "10 10 290 290"

Outputs of OpenLane:

openlane/module\_filter/runs/<run\_id>/

├── results/

│ ├── synthesis/

│ │ └── module\_filter.synthesis.rpt

│ ├── placement/

│ │ └── module\_filter.placement.def

│ ├── routing/

│ │ └── module\_filter.gds

├── logs/

│ └── openlane.log

└── reports/

├── area.rpt

├── power.rpt

├── timing.rpt

OpenLane-Compatible: module\_filter.v (Streaming Style)

This version:

Uses a streaming interface (one token per clock)

Matches a hardcoded target ("execute")

Outputs matching token stream in real-time

Verilog code:

module module\_filter (

input wire clk,

input wire rst,

input wire [7:0] token\_in,

input wire token\_valid,

output reg [7:0] token\_out,

output reg token\_out\_valid,

output reg match\_active

);

// Internal state

reg [2:0] state;

reg [7:0] buffer [0:7]; // "execute" = 7 characters

integer i;

localparam [7:0] TARGET [0:6] = { "e", "x", "e", "c", "u", "t", "e" };

localparam S\_IDLE = 0;

localparam S\_MATCHING = 1;

localparam S\_OUTPUTTING = 2;

always @(posedge clk or posedge rst) begin

if (rst) begin

state <= S\_IDLE;

token\_out <= 0;

token\_out\_valid <= 0;

match\_active <= 0;

for (i = 0; i < 8; i = i + 1) buffer[i] <= 0;

end else begin

token\_out\_valid <= 0;

case (state)

S\_IDLE: begin

if (token\_valid) begin

// Shift in token

for (i = 6; i > 0; i = i - 1)

buffer[i] <= buffer[i-1];

buffer[0] <= token\_in;

// Check match

if (buffer[6:0] == TARGET) begin

match\_active <= 1;

token\_out <= buffer[6]; // start output

token\_out\_valid <= 1;

state <= S\_OUTPUTTING;

end

end

end

S\_OUTPUTTING: begin

if (token\_valid) begin

token\_out <= token\_in;

token\_out\_valid <= 1;

// Simple end-block check: stop at '}'

if (token\_in == "}") begin

match\_active <= 0;

state <= S\_IDLE;

end

end

end

endcase

end

end

endmodule  
  
top.v:

module top (

input clk,

input rst,

input [7:0] token\_in,

input token\_valid,

output [7:0] token\_out,

output token\_out\_valid

);

wire match\_active;

module\_filter mf (

.clk(clk),

.rst(rst),

.token\_in(token\_in),

.token\_valid(token\_valid),

.token\_out(token\_out),

.token\_out\_valid(token\_out\_valid),

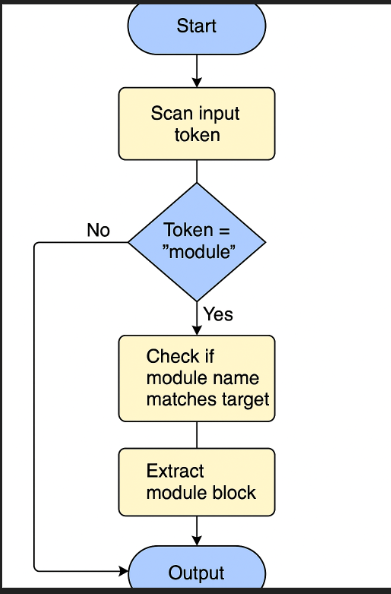
.match\_active(match\_active)

);

endmodule

Algorithm

Flowchart:



Python code for checking the functionality:

import re  
  
  
def filter\_module(input\_text: str, target\_module: str) -> str:  
 *"""  
 Extract the block for the target module from the full input string.  
 Each module follows the format: modulename{...}  
 """* # Build regex dynamically for the given target module  
 pattern = re.compile(rf'({target\_module})\s\*\{{.\*?\}}', re.DOTALL)  
  
 match = pattern.search(input\_text)  
 if match:  
 return match.group(0)  
 else:  
 return f"Module '{target\_module}' not found."  
  
  
def main():  
 print("Module Filter Tool")  
  
 # Get input from the user at runtime  
 input\_string = input("Enter the full VLM module string:\n> ")  
 target\_module = input("Enter the module to extract:\n> ")  
  
 result = filter\_module(input\_string, target\_module)  
 print("\nFiltered Output:\n", result)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Output:

Module Filter Tool

Enter the full VLM module string:

> fetch{a=1;}decode{b=2;}

Enter the module to extract:

> execute

Filtered Output:

Module 'execute' not found.