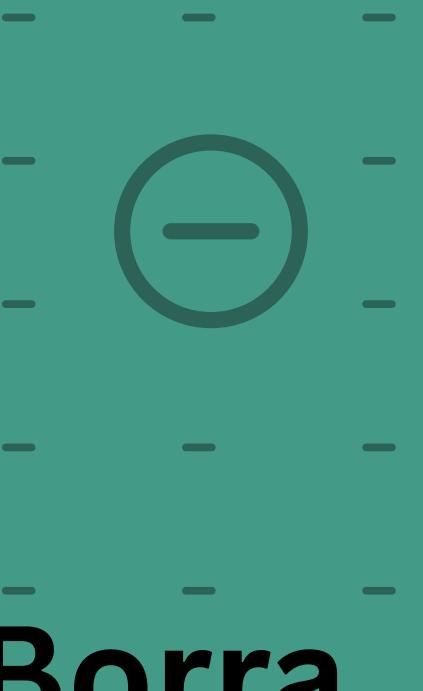
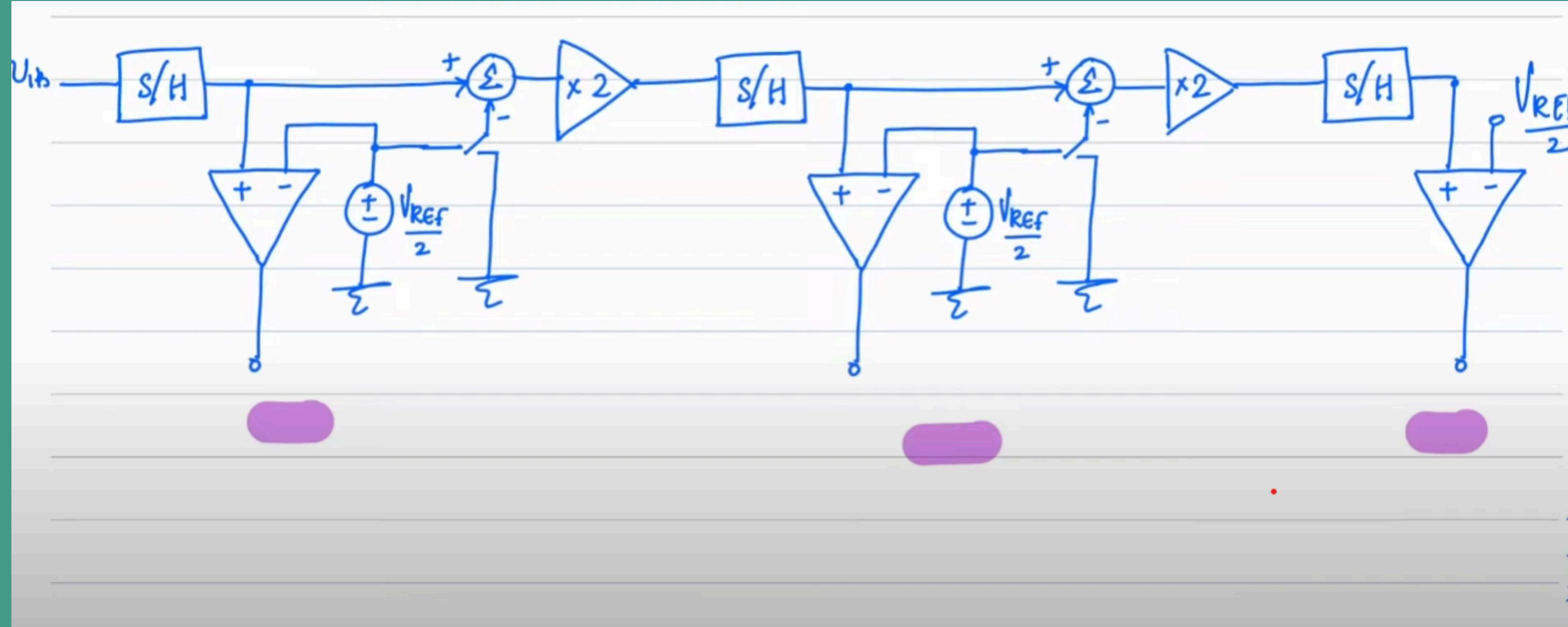


8 Bit adc



Safal gupta Borra

Adc Basic Stru



Sample and Hold

```
1 'include "constants.vams"
2 'include "disciplines.vams"
3
4 module Sample_Hold(vdd, vss, in, out, clk);
5   inout vdd, vss;
6   input in, clk;
7   output out;
8   electrical in, out, clk;
9
10 parameter real clk_vth = 0.5, delay = 0, ttime = 1p
11
12 real v;
13
14 analog begin
15   if ((V(clk) > clk_vth))
16     v = V(in); // passing phase
17
18   @(cross(V(clk) - clk_vth, -1))
19     v = V(in); // sampling phase
20
21   // @(cross(V(clk) - clk_vth, +1))
22   //   v = V(in);
23
24   V(out) <+ transition(v, delay, ttime);
25 end
26
27 endmodule
```

if ((V(Clk) > clk_vth))

v = V(in); // passing phase

This part checks if the clock signal (Clk) is greater than a threshold voltage (clk_vth), which is defined earlier as 0.5.

If the clock voltage exceeds this threshold, the input voltage (V(in)) is assigned to the variable v.

@(cross(V(Clk) - clk_vth, -1))

v = V(in); // sampling phase

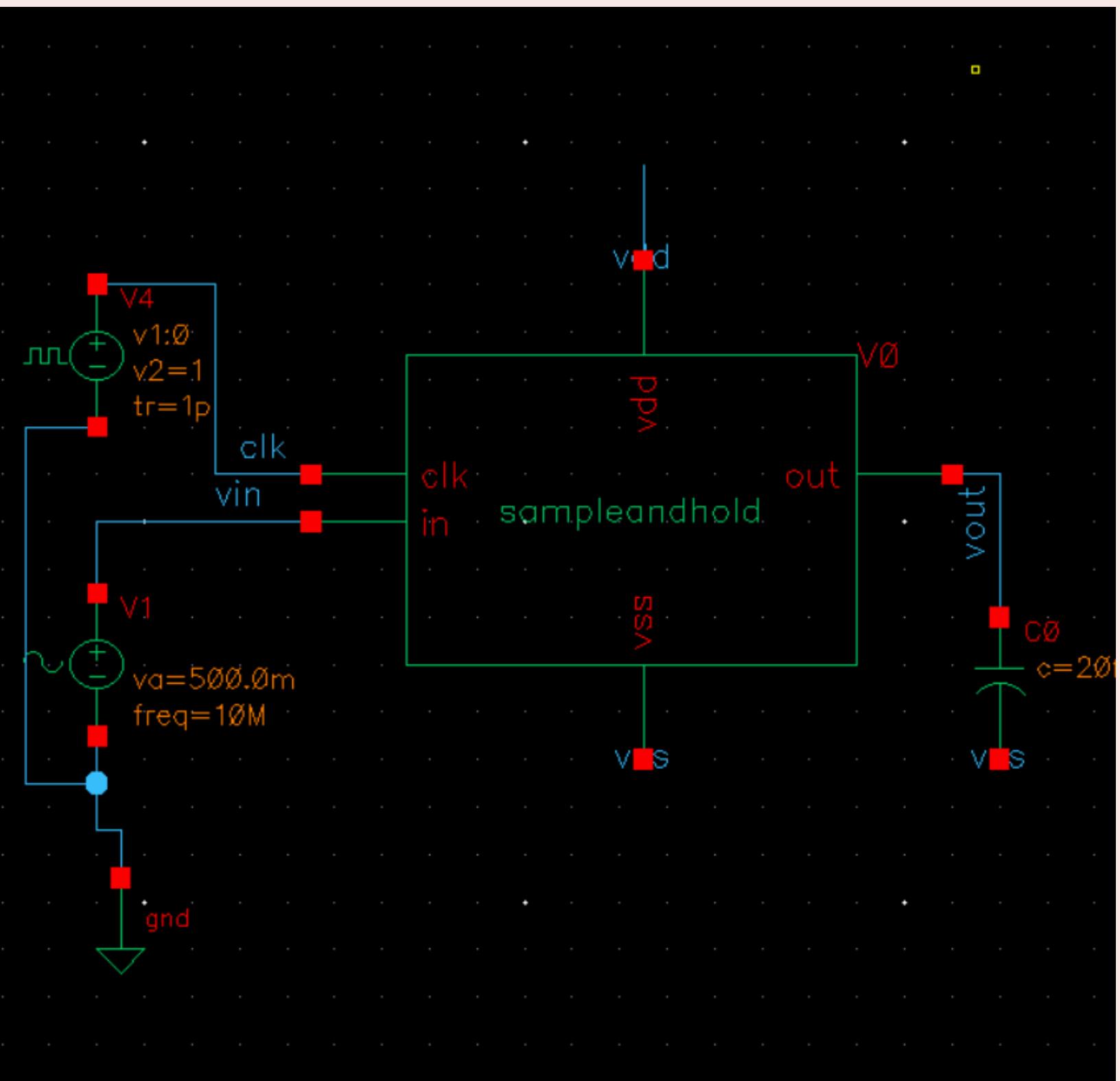
- This uses the @cross function, which detects when the clock signal (V(Clk)) crosses the threshold (clk_vth) from high to low (indicated by the -1 argument). The cross function detects transitions of the clock signal.

- When the clock crosses this point, the value of the input signal (V(in)) is sampled and stored in v.

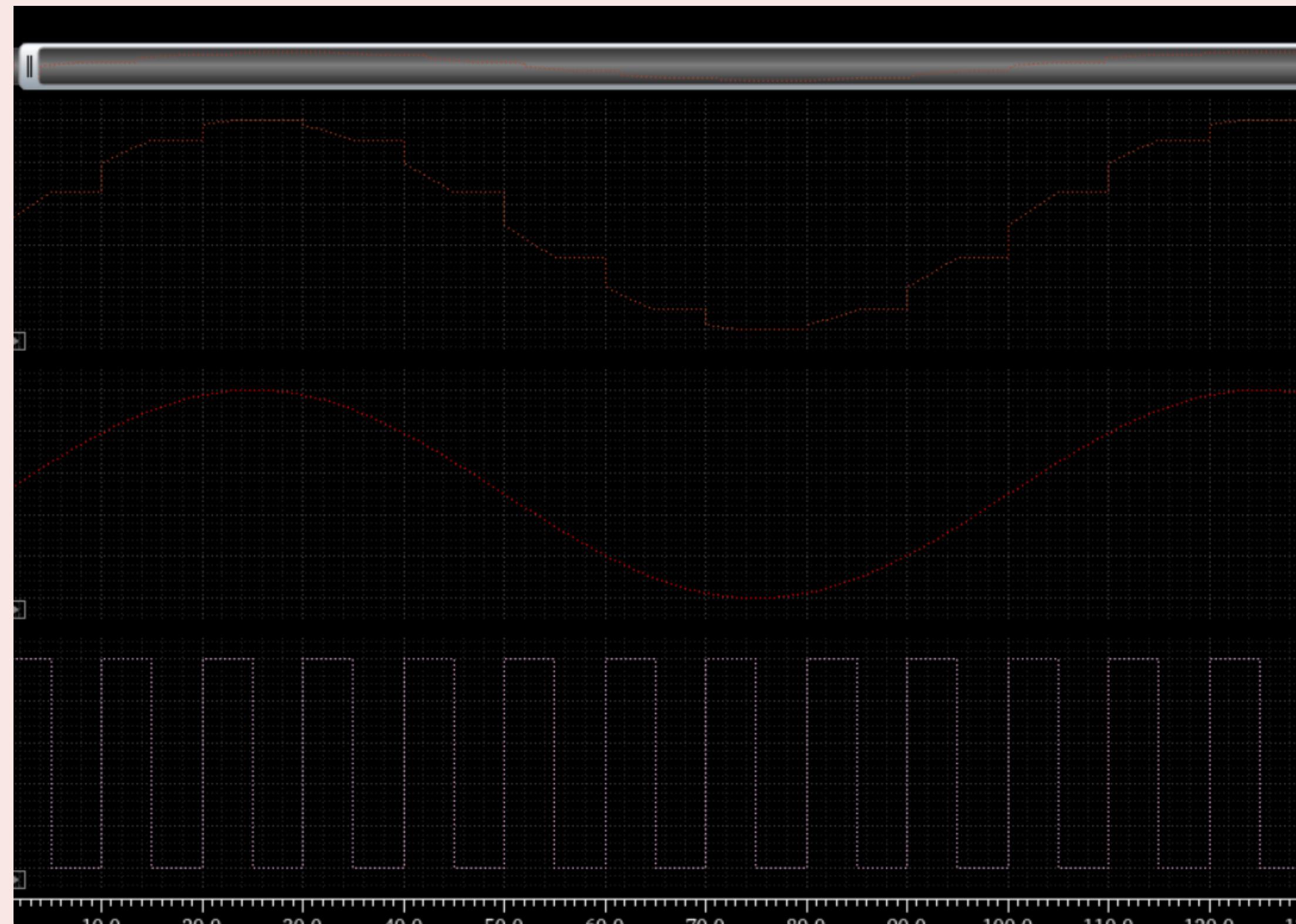
V(vout) <+ transition(result, delay, ttime);

The output voltage V(vout) is set to the value of result using the transition() function. transition(result, delay, ttime) ensures that the output does not change instantaneously but transitions smoothly with a specified delay (delay = 0 in this case) and transition time (ttime = 1 picosecond by default).

Sample and Hold TB



Sample and Hold Output



Comparator

```
*include "constants.vams"
*include "disciplines.vams"

module Comparator(vin, vout, vdd, vss, ref);
    input vin, ref;
    input vdd, vss;
    output vout;
    parameter delay = 0, ttime = 1p;
    electrical vin, ref, vdd, vout;

    real result;

    analog begin
        @(cross(V(vin) - V(ref), 0)or initial_step)
            if (V(vin) >= V(ref))
                result = V(vdd);
            else
                result = 0.0;
        V(vout) <+ transition(result, delay, ttime);
    end
endmodule
```

@(cross(V(vin) - V(ref), 0) or initial_step)

The `@cross(V(vin) - V(ref), 0)` function triggers whenever the difference between the input voltage `vin` and the reference voltage `ref` crosses zero. This indicates the moment when `vin` equals `ref`.

```
if (V(vin) >= V(ref))
    result = V(vdd);
else
    result = 0.0
```

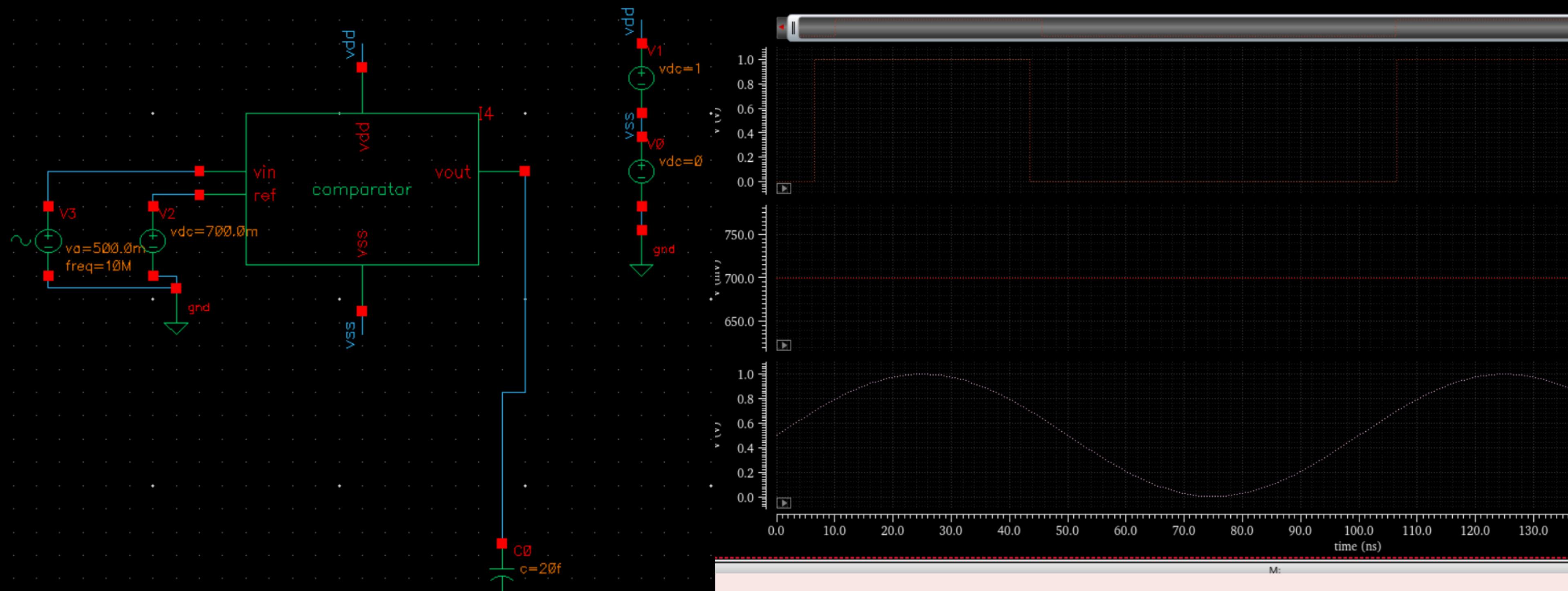
If the input voltage `V(vin)` is greater than or equal to the reference voltage `V(ref)`, the output result is set to `V(vdd)`. In most cases, `V(vdd)` is the supply voltage, representing a logic high (or positive output)

`V(vout) <+ transition(result, delay, ttime);`

The output voltage `V(vout)` is set to the value of `result` using the `transition()` function. `transition(result, delay, ttime)` ensures that the output does not change instantaneously but transitions smoothly with a specified delay (`delay = 0` in this case) and transition time (`ttime = 1 picosecond by default`).

Comparator Tb

comparator output



Dac

```
module MDAC(din, vin, vout, vdd, vss);
  parameter real gain = 2;
  parameter real delay = 0.0;
  parameter real ttime = 1p;

  inout vdd, vss;
  input vin, din;
  output vout;

  electrical vin, din, vout, vdd, vss;
  real result;

  analog begin
    if (V(din) == 1)
      result = (V(vin) - ((V(vdd) - V(vss)) / 2)) * gain;
    else
      result = V(vin) * gain;
    // Limiting the output to rail to rail
    case (1)
      result > V(vdd): result = V(vdd);
      result < V(vss): result = V(vss);
    endcase
    V(vout) <+ transition(result, delay, ttime);
  end
endmodule
```

if (V(din) == 1)
result = (V(vin) - ((V(vdd) - V(vss)) / 2)) * gain;

else

result = V(vin) * gain;

The output calculation depends on the value of the digital input din.

If din is equal to 1, it calculates the result using the formula:

result = (V(vin) - ((V(vdd) - V(vss)) / 2)) * gain;

Here, the output is derived from the input voltage vin, adjusted by the midpoint of the supply voltages $((V(vdd) - V(vss)) / 2)$ and then multiplied by the gain.

If din is not equal to 1, it simply scales the input voltage vin by the gain:

result = V(vin) * gain;

case (1)

result > V(vdd): result = V(vdd);

result < V(vss): result = V(vss);

Endcase

This part ensures that the output voltage does not exceed the specified supply voltage range.

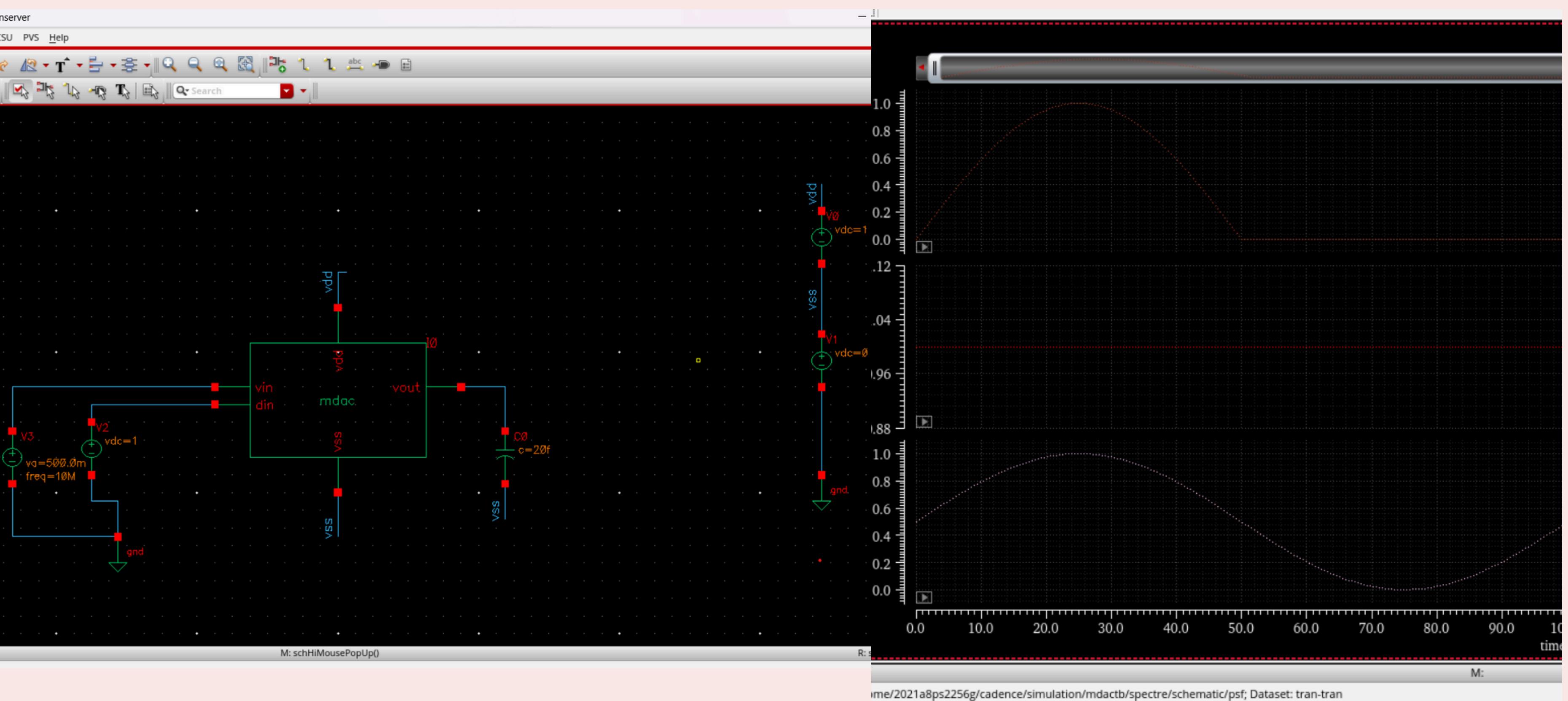
If result exceeds V(vdd), it is clipped to V(vdd).

If result falls below V(vss), it is set to V(vss).

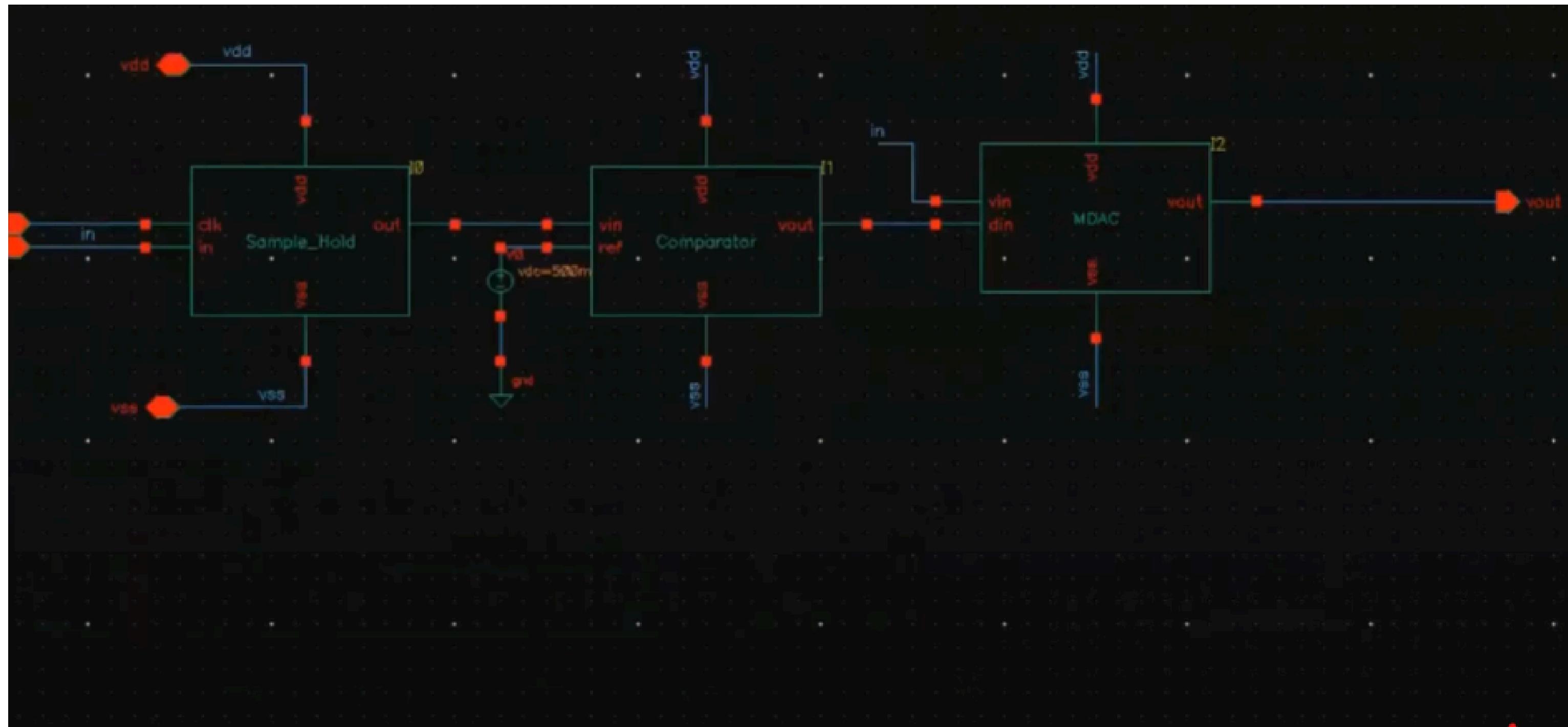
Now lets see the schematic for it

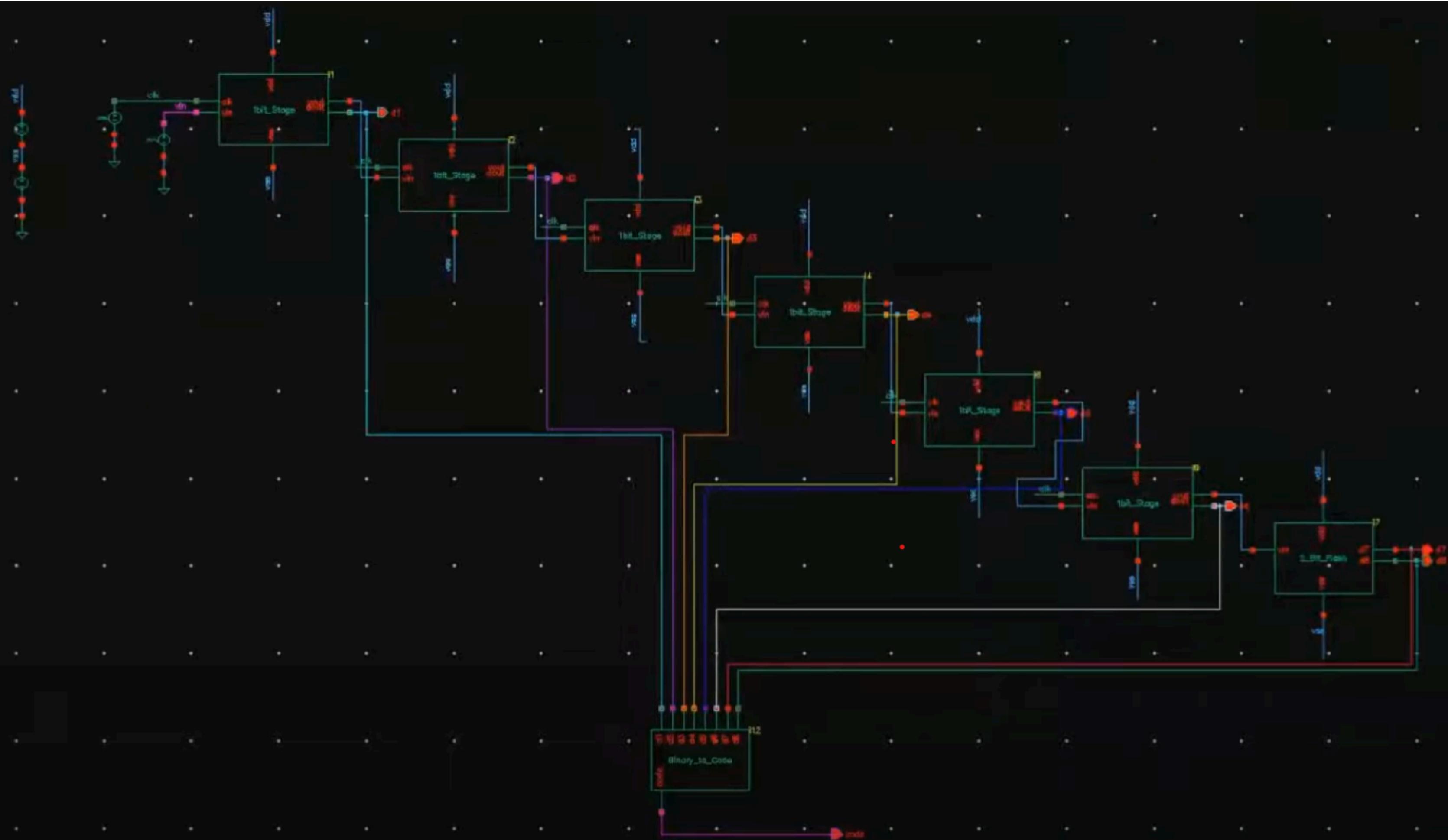
Dac TB

Dac output

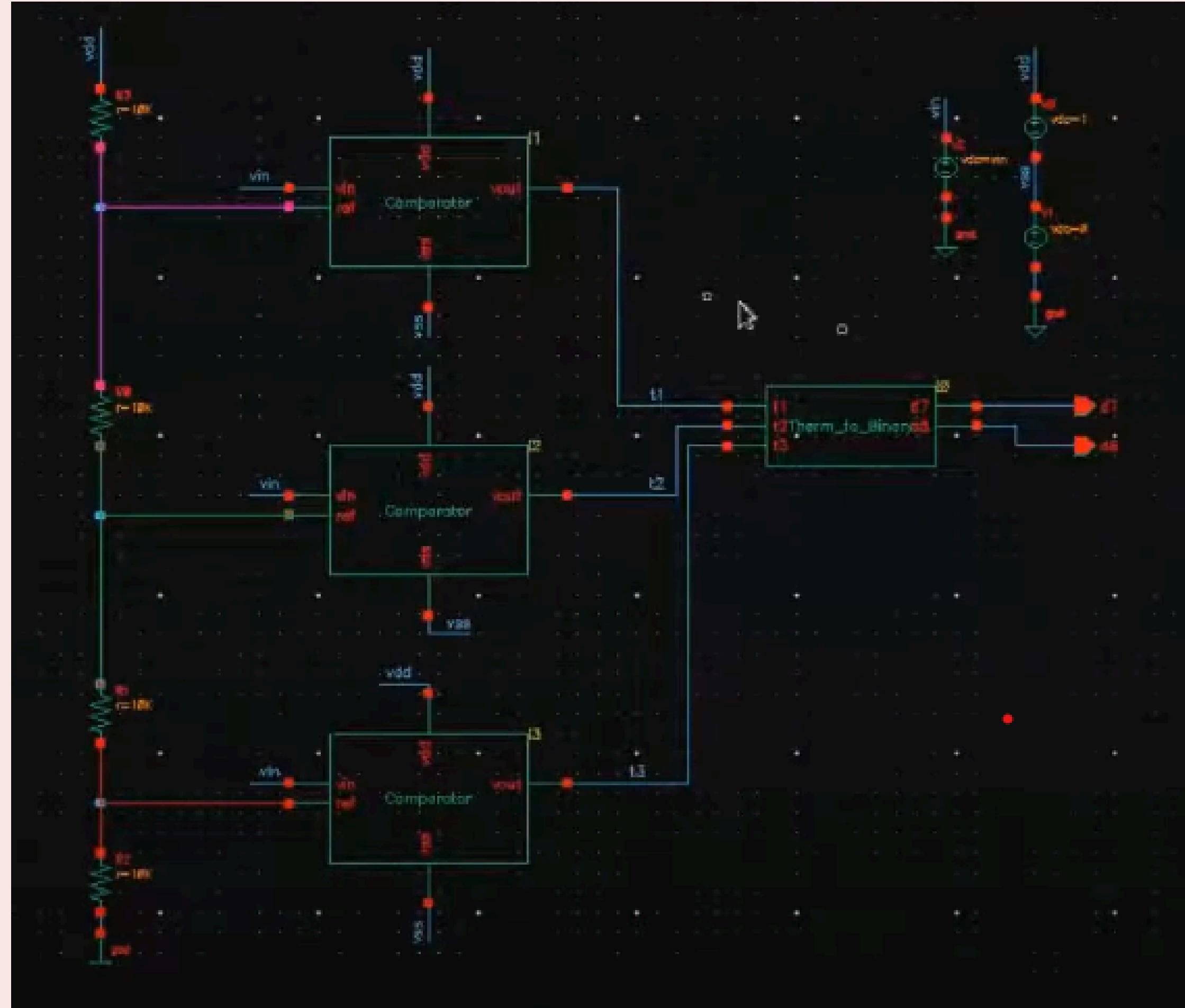


1 Bit Adc pipelining

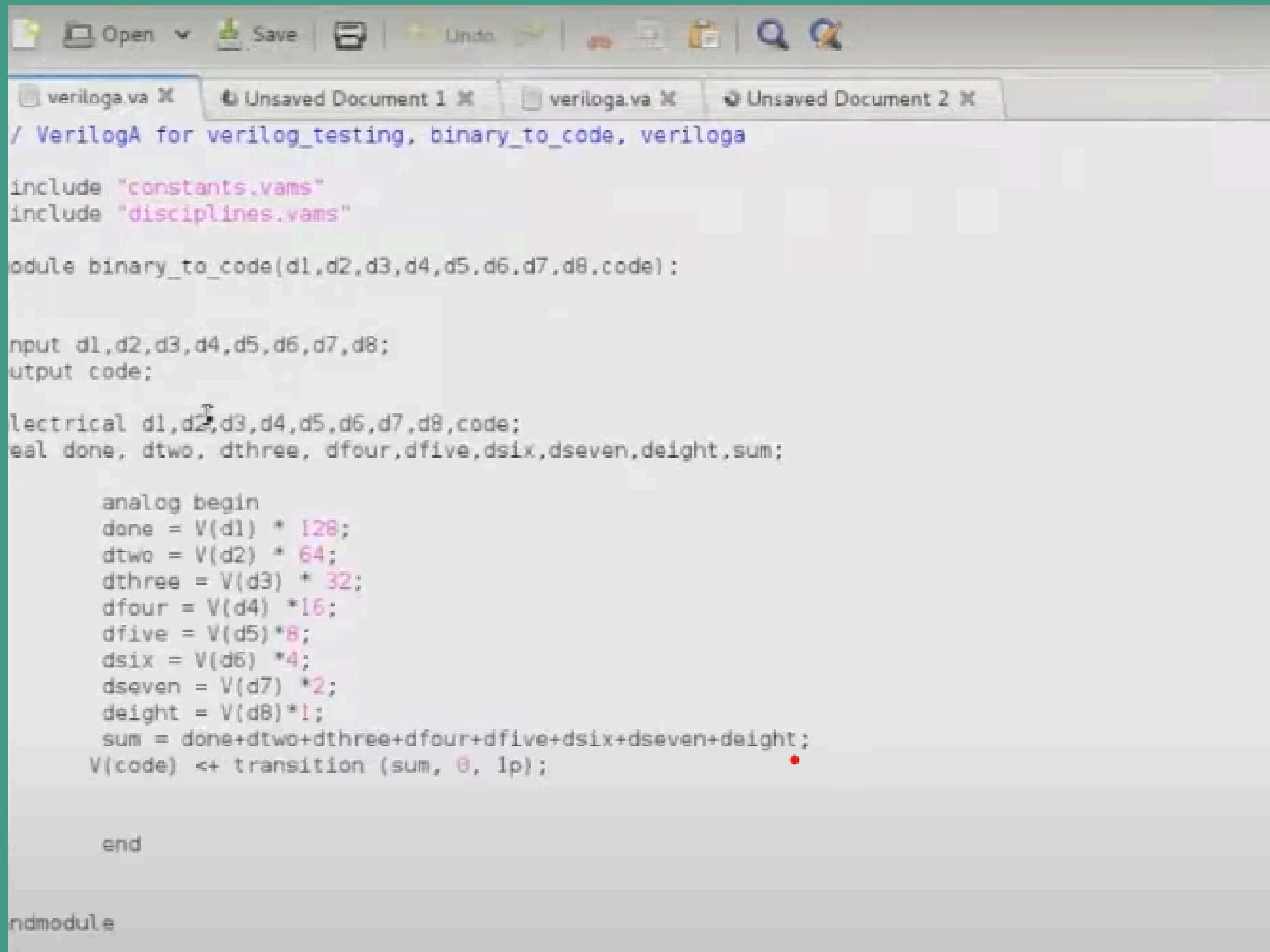




2 bit flash Adc



Binary to Code



The screenshot shows a Verilog editor interface with a toolbar at the top and a menu bar below it. The menu bar includes 'File', 'Edit', 'Search', 'Tools', 'Help', and 'About'. The main window displays a Verilog module named 'binary_to_code'. The code implements a binary-to-code conversion algorithm using analog assignment statements to calculate weights for each bit position and sum them up.

```
verilogA.vams verilogA.vams verilogA.vams verilogA.vams
/ VerilogA for verilog_testing, binary_to_code, verilogA

include "constants.vams"
include "disciplines.vams"

odule binary_to_code(d1,d2,d3,d4,d5,d6,d7,d8,code);

nput d1,d2,d3,d4,d5,d6,d7,d8;
utput code;

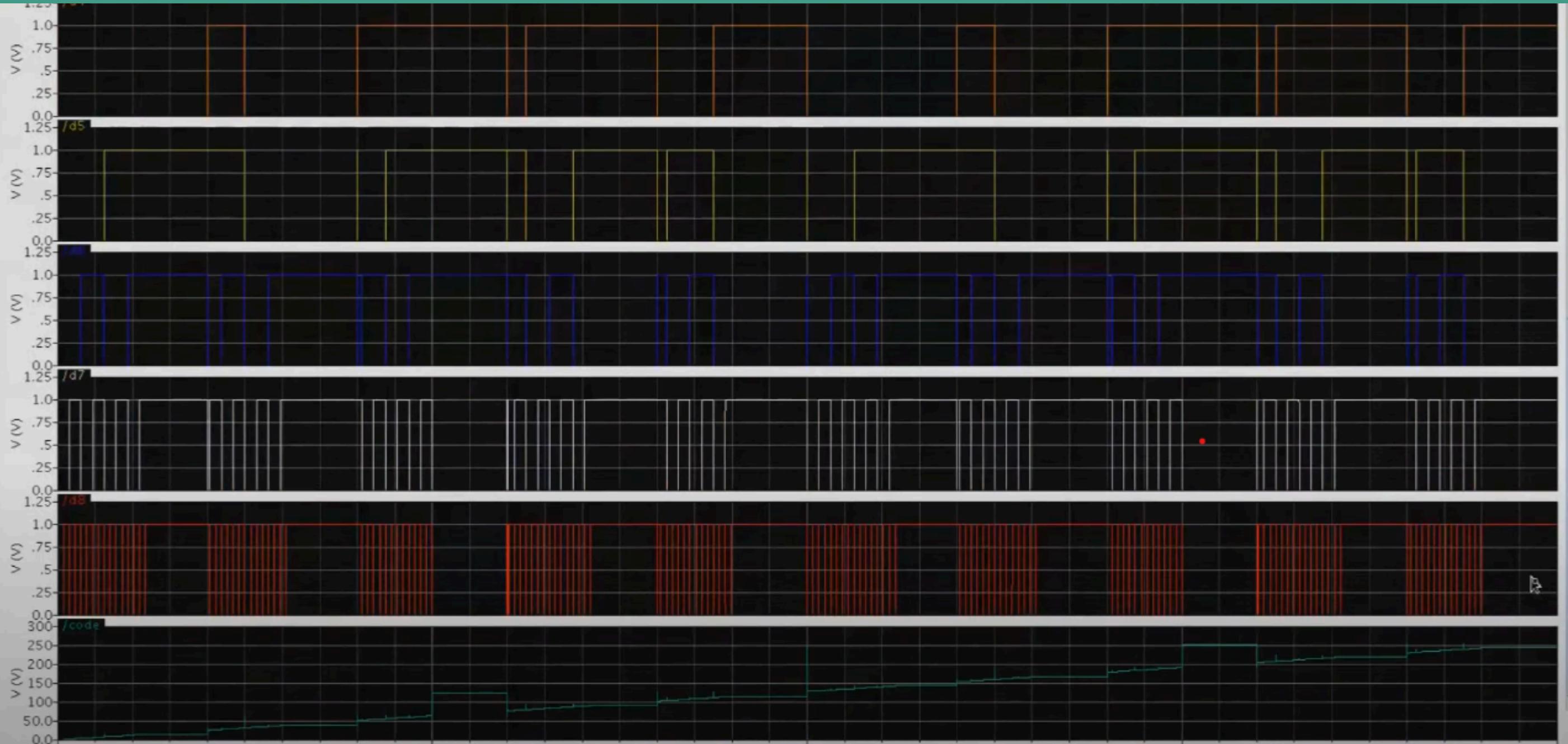
lectrical d1,d2,d3,d4,d5,d6,d7,d8,code;
real done, dtwo, dthree, dfour, dfive, dsix, dseven, deight, sum;

analog begin
done = V(d1) * 128;
dtwo = V(d2) * 64;
dthree = V(d3) * 32;
dfour = V(d4) *16;
dfive = V(d5)*8;
dsix = V(d6) *4;
dseven = V(d7) *2;
deight = V(d8)*1;
sum = done+dtwo+dthree+dfour+dfive+dsix+dseven+deight;
V(code) <+ transition (sum, 0, 1p);
end

ndmodule
```

Output

d4
d5
d6
d7
d8
code



Output

clk

input

D1

D2

D3

D4

