512-Row SRAM Decoder Layout Design and Verification in TSMC N16ADFP Phase 2: Preliminary Layout

Nandini Kumawat

November 26, 2025

Guided by: Prof. Chris Kim

Circuit Overview

This is a **hierarchical 512-row decoder** for SRAM that takes a 9-bit address input (a0-a8) plus a clock signal and generates 512 unique wordline select signals (wls<0> through wls<511>) and (wln<0> through wln<511>).

Hierarchical Architecture

The design uses a **three-stage predecoding approach** that breaks down the 9-bit decoding problem into manageable blocks:

Stage 1: Subblock1 - 4-input AND Predecoder

- **Inputs:** a0, a1, a2, clk (4 bits total)
- **Outputs:** 8 predecoded signals (pd0_0 through pd0_7)
- Uses 8 instances of and4_schematic (4-input AND gates)
- Generates all $2^3 = 8$ combinations of the 3 address bits, gated by the clock
- Each AND4 gate combines one combination of {a0, a1, a2} with the clock signal

Stage 2: Subblock2 & Subblock3 - 3-input AND Predecoders

Subblock2:

- **Inputs:** a3, a4, a5 (3 bits)
- **Outputs:** 8 predecoded signals (pd1_0 through pd1_7)
- Uses and3_inv8 cells (3-input NAND followed by strong inverter)

Subblock3:

- **Inputs:** a6, a7, a8 (3 bits)
- **Outputs:** 8 predecoded signals (pd2_0 through pd2_7)

- Also uses and3_inv8 cells

Both subblocks generate all $2^3 = 8$ combinations of their respective address bits.

Stage 3: Final AND Matrix

- **512 instances** of and3_schematic (3-input AND gates)
- Each AND3 gate combines:
  - 1 signal from pd0 (subblock1) → 8 choices
  - 1 signal from pd1 (subblock2) → 8 choices
  - 1 signal from pd2 (subblock3) → 8 choices
- Total combinations: 8 × 8 × 8 = **512 unique wordlines**

Cell Library Used

Basic Gates:

1. **inverter_schematic** - Minimal inverter (1 fin NMOS/PMOS)
2. **Inv8** - Strong 8-fin inverter for driving wordlines
3. **inv2** - 2-fin inverter
4. **inv4** - 4-fin inverter for intermediate buffering

Compound Gates:

1. **and3_inv8** - 3-input NAND (4-fin) + Inv8 buffer
2. **and3_schematic** - 3-input NAND (4-fin) + inv2 buffer
3. **and4_schematic** - 4-input NAND (4-fin) + Inv8 buffer

All transistors use **TSMC 16nm FinFET** technology:

- nch_svt_mac (NMOS) and pch_svt_mac (PMOS)
- Channel length: L = 16nm
- Variable fin counts (nfin) for strength scaling

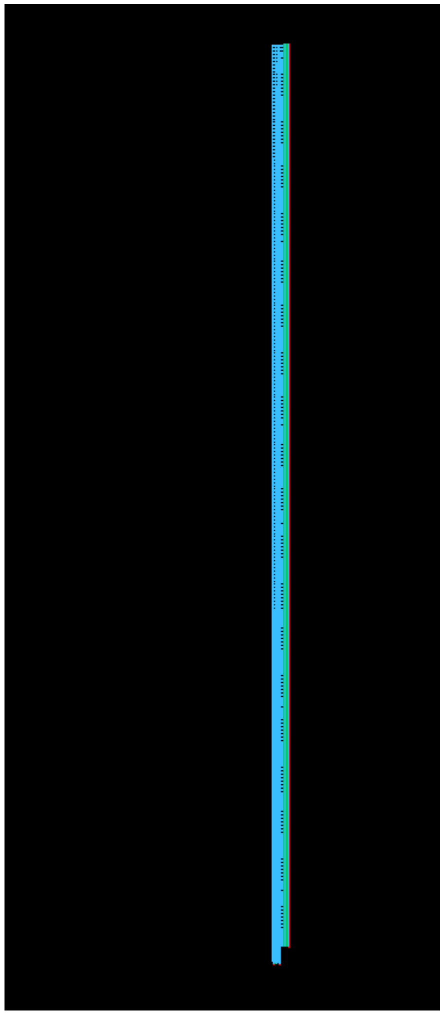
Output Stage Buffering

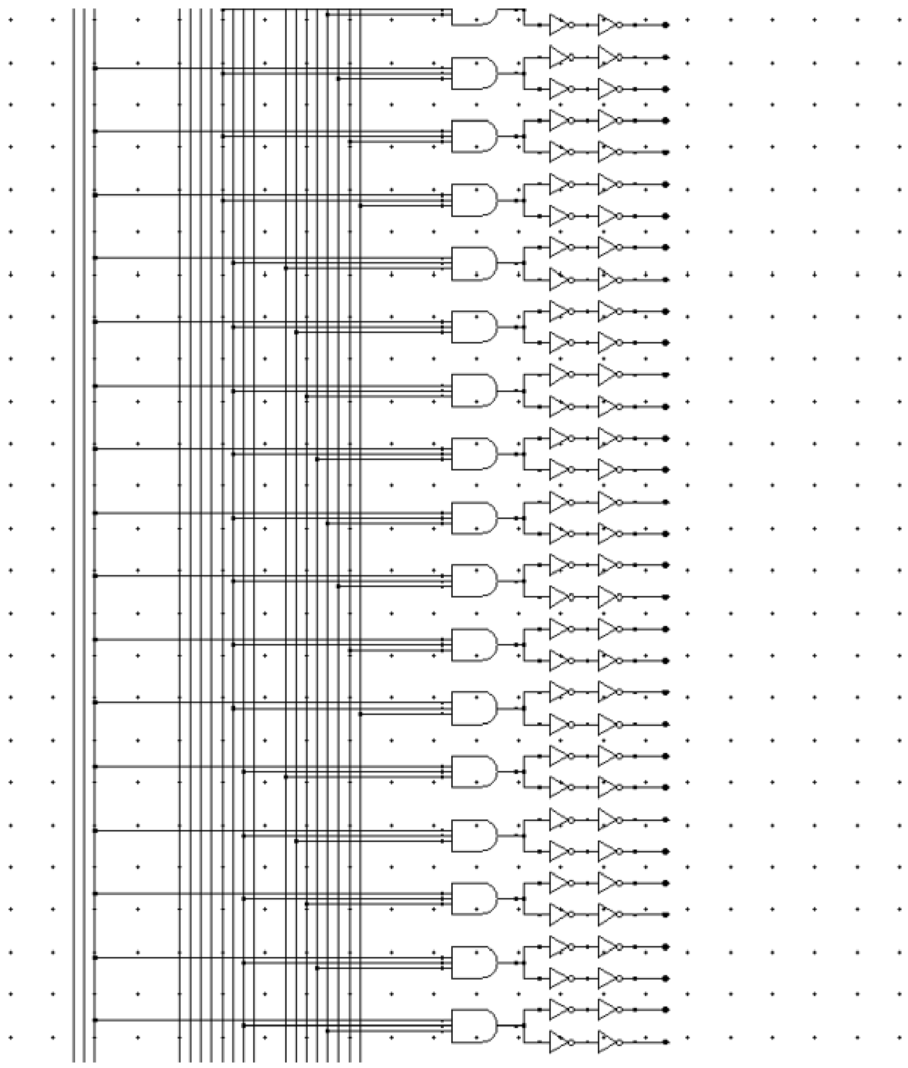
Each of the 512 decoded signals goes through **dual inv4 buffers**:

- One buffer generates **wls** (wordline select going South)
- Other buffer generates **wln** (wordline select going North)

This provides both polarities for SRAM bitcell access with strong drive capability (4-fin inverters).

Schematic Generated using Cadence SKILL





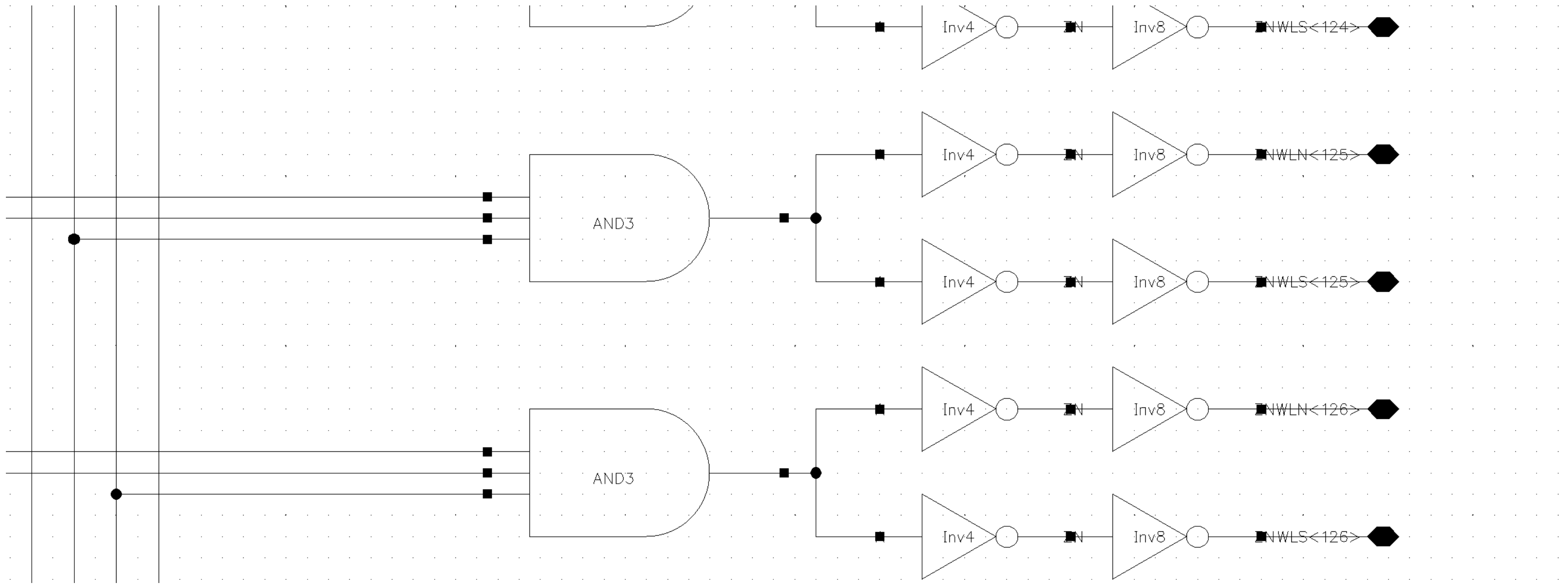


Figure: Wordline Driver Organization

Wordline driver organization

The intermediate AND outputs feed a two stage driver:

- inv4_schematic
    - nfin = 4.
    - Local intermediate buffer that reduces fan out and improves slew.

- Inv8
    - nfin = 8.
    - Strongest driver stage.
    - Feeds the final wordline nets wls<0:511> or wln<0:511>.

The final configuration provides:

- Strong edges at the bitcell wordline pin into the large distributed RC of the row.
- Uniform delay across all 512 rows because each row passes through exactly one inv4 + one Inv8 chain.

SKILL CODE

`block_measurement.il`

```
procedure(reportSymbolLabelsAsPins(libName cellName)

  let((cv bbox ll ur dbu width height)
    cv = dbOpenCellViewByType(libName cellName "symbol" "" "r")
    when(!cv
      error("Cannot open symbol view for %s/%s\n" libName cellName)
    )

    bbox = cv~>bBox
    ll = car(bbox)
    ur = cadr(bbox)

    dbu = cv~>dbuPerUU
    unless(numberp(dbu)
      dbu = 1.0
    )

    width = (car(ur) - car(ll)) / dbu
    height = (cadr(ur) - cadr(ll)) / dbu

    printf("\n========================================\n")
    printf("Symbol: %s/%s\n" libName cellName)
    printf("========================================\n")
    printf("  Width:  %.3f (user units)\n" width)
    printf("  Height: %.3f (user units)\n" height)
    printf("  DBU/UU: %.3f\n" dbu)

    printf("\n  Label-based pins:\n")
    printf("  %-15s %10s %10s\n" "PIN NAME" "X (uu)" "Y (uu)")
    printf("  ---------------------------------------------\n")

    foreach(shape cv~>shapes
      when(shape~>objType == "label"
        let((txt xy x y)
          txt = (shape~>theLabel || shape~>label)
          xy = shape~>xy
          when(txt && xy && length(xy) >= 2
            x = car(xy)/dbu
            y = cadr(xy)/dbu
            printf("  %-15s %10.3f %10.3f\n" txt x y)
          )
        )
      )
    )

    dbClose(cv)
    t
  )
)
```

decoder_512.il

```
;;;#########################################################
;;;#
;;;#  SKILL code for creating 512-row Decoder
;;;#  Correct routing: Vertical buses from PDs,
;;;#  Horizontal buses to AND gates
;;;#
;;;#  Usage:
;;;#    load("decoder_512.il")
;;;#    create_512_decoder()
;;;#
;;;#########################################################

procedure(create_512_decoder()
  let((cv cv_pd0 cv_pd1 cv_pd2 cv_and
       lib cellname
       pd0_x pd0_y pd1_x pd1_y pd2_x pd2_y
       pin_master wire
       i j k idx inst_name
       and_x_start spacing_y
       vdd_rail_y vss_rail_y
       and_vdd_pwr_x and_vss_pwr_x
       pd0_out_y pd1_out_y pd2_out_y
       pd0_bus_x pd1_bus_x pd2_bus_x
       horiz_bus_y wl_y_pos)

    ;-------------------------------------------------------
    ; Parameters
    ;-------------------------------------------------------
    lib      = "ProjectSRAM"
    cellname = "Decoder_512"

    printf("\n=== Creating 512-row Decoder (Correct Routing) ===\n")
```

```
;-------------------------------------------------------
; Open cellViews
;-------------------------------------------------------
cv      = dbOpenCellViewByType(lib cellname "schematic" "schematic" "a")
cv_pd0  = dbOpenCellViewByType(lib "subblock1" "symbol" "" "r")
cv_pd1  = dbOpenCellViewByType(lib "subblock2" "symbol" "" "r")
cv_pd2  = dbOpenCellViewByType(lib "subblock3" "symbol" "" "r")
cv_and  = dbOpenCellViewByType(lib "and3" "symbol" "" "r")

when(!cv error("Failed to open schematic %s/%s\n" lib cellname))
when(!cv_pd0 error("Failed to open subblock1 symbol\n"))
when(!cv_pd1 error("Failed to open subblock2 symbol\n"))
when(!cv_pd2 error("Failed to open subblock3 symbol\n"))
when(!cv_and error("Failed to open and3 symbol\n"))

;===============================================
; LAYOUT PARAMETERS
;===============================================
; Predecoder positions (at bottom)
pd0_x = 3.0
pd0_y = -10.0

pd1_x = 7.0
pd1_y = -10.0

pd2_x = 11.0
pd2_y = -10.0

; AND gates start position
and_x_start = 20.0
spacing_y   = 1.6      ; Vertical spacing between AND gates

; Power rails
```

```
vdd_rail_y = 10.0
vss_rail_y = pd0_y - 3.0

; Vertical power rails for AND array
and_vdd_pwr_x = and_x_start - 1.5
and_vss_pwr_x = and_x_start - 2.0

; Vertical bus X positions (between PDs and ANDs)
pd0_bus_x = makeVector(8 0.0)
pd1_bus_x = makeVector(8 0.0)
pd2_bus_x = makeVector(8 0.0)

;==============================================
; STEP 1: Place PreDecoders
;==============================================
printf("Placing PreDecoders...\n")
dbCreateInst(cv cv_pd0 "I_PD0" list(pd0_x pd0_y) "R0")
dbCreateInst(cv cv_pd1 "I_PD1" list(pd1_x pd1_y) "R0")
dbCreateInst(cv cv_pd2 "I_PD2" list(pd2_x pd2_y) "R0")

;==============================================
; STEP 2: Create Power Rails
;==============================================
printf("Creating power rails...\n")

; Horizontal VDD rail
wire = schCreateWire(cv "draw" "full"
  list(0.0:vdd_rail_y (and_x_start + 10.0):vdd_rail_y)
  0.125 0.125 0)
schCreateWireLabel(cv car(wire) 0.5:vdd_rail_y
  "VDD" "centerLeft" "R0" "fixed" 0.125 nil)

; Horizontal VSS rail
wire = schCreateWire(cv "draw" "full"
  list(0.0:vss_rail_y (and_x_start + 10.0):vss_rail_y)
  0.125 0.125 0)
schCreateWireLabel(cv car(wire) 0.5:vss_rail_y
  "VSS" "centerLeft" "R0" "fixed" 0.125 nil)

; Vertical VDD rail for AND array
wire = schCreateWire(cv "draw" "full"
  list(and_vdd_pwr_x:vdd_rail_y
       and_vdd_pwr_x:(vss_rail_y + 2.0))
  0.0625 0.0625 0)

; Vertical VSS rail for AND array
wire = schCreateWire(cv "draw" "full"
  list(and_vss_pwr_x:vss_rail_y
       and_vss_pwr_x:vdd_rail_y)
  0.0625 0.0625 0)

;==============================================
; PD Power Connections
;==============================================
; PD0 power
let((pd0_vdd_x pd0_vdd_y pd0_vss_x pd0_vss_y pd0_pwr_x)
  pd0_vdd_x = pd0_x + 0.125
  pd0_vdd_y = pd0_y - 0.963
  pd0_vss_x = pd0_x + 0.250
  pd0_vss_y = pd0_y - 0.963
  pd0_pwr_x = pd0_x - 0.2

  wire = schCreateWire(cv "draw" "full"
    list(pd0_vdd_x:pd0_vdd_y pd0_pwr_x:pd0_vdd_y pd0_pwr_x:vdd_rail_y)
    0.0625 0.0625 0)
  wire = schCreateWire(cv "draw" "full"
    list(pd0_vss_x:pd0_vss_y pd0_pwr_x:pd0_vss_y pd0_pwr_x:vss_rail_y)
    0.0625 0.0625 0)
```

```
)

; PD1 power
let((pd1_vdd_x pd1_vdd_y pd1_vss_x pd1_vss_y pd1_pwr_x)
  pd1_vdd_x = pd1_x + 1.062
  pd1_vdd_y = pd1_y - 1.025
  pd1_vss_x = pd1_x + 1.188
  pd1_vss_y = pd1_y - 1.025
  pd1_pwr_x = pd1_x + 0.4

  wire = schCreateWire(cv "draw" "full"
    list(pd1_vdd_x:pd1_vdd_y pd1_pwr_x:pd1_vdd_y pd1_pwr_x:vdd_rail_y)
    0.0625 0.0625 0)
  wire = schCreateWire(cv "draw" "full"
    list(pd1_vss_x:pd1_vss_y pd1_pwr_x:pd1_vss_y pd1_pwr_x:vss_rail_y)
    0.0625 0.0625 0)
)

; PD2 power
let((pd2_vdd_x pd2_vdd_y pd2_vss_x pd2_vss_y pd2_pwr_x)
  pd2_vdd_x = pd2_x + 0.688
  pd2_vdd_y = pd2_y - 1.025
  pd2_vss_x = pd2_x + 0.812
  pd2_vss_y = pd2_y - 1.025
  pd2_pwr_x = pd2_x + 0.3

  wire = schCreateWire(cv "draw" "full"
    list(pd2_vdd_x:pd2_vdd_y pd2_pwr_x:pd2_vdd_y pd2_pwr_x:vdd_rail_y)
    0.0625 0.0625 0)
  wire = schCreateWire(cv "draw" "full"
    list(pd2_vss_x:pd2_vss_y pd2_pwr_x:pd2_vss_y pd2_pwr_x:vss_rail_y)
    0.0625 0.0625 0)
)
```

```
;==============================================
; STEP 3: Create Vertical Buses from PDs (going UP)
;==============================================
printf("Creating vertical buses from predecoders...\n")

; PD output Y positions (at TOP of symbols)
pd0_out_y = pd0_y - 0.225
pd1_out_y = pd1_y - 0.287
pd2_out_y = pd2_y - 0.287

; Create 8 vertical buses from PD0 (going UP)
for(i 0 7
  let((pd0_out_x bus_x)
    pd0_out_x = pd0_x + (0.375 - i*0.125)
    bus_x = 14.0 + i*0.2
    pd0_bus_x[i] = bus_x

    ; Vertical wire from output going UP
    wire = schCreateWire(cv "draw" "full"
      list(pd0_out_x:pd0_out_y
           pd0_out_x:(pd0_out_y + 1.0)
           bus_x:(pd0_out_y + 1.0)
           bus_x:vdd_rail_y)
      0.0625 0.0625 0)
    schCreateWireLabel(cv car(wire) bus_x:(pd0_out_y + 0.5)
      sprintf(nil "PD0_%d" i) "centerLeft" "R90" "fixed" 0.0625 nil)
  )
)

; Create 8 vertical buses from PD1 (going UP)
for(j 0 7
  let((pd1_out_x bus_x)
    pd1_out_x = pd1_x + (1.375 - j*0.125)
    bus_x = 16.0 + j*0.2
    pd1_bus_x[j] = bus_x

    wire = schCreateWire(cv "draw" "full"
      list(pd1_out_x:pd1_out_y
           pd1_out_x:(pd1_out_y + 1.0)
           bus_x:(pd1_out_y + 1.0)
           bus_x:vdd_rail_y)
      0.0625 0.0625 0)
    schCreateWireLabel(cv car(wire) bus_x:(pd1_out_y + 0.5)
      sprintf(nil "PD1_%d" j) "centerLeft" "R90" "fixed" 0.0625 nil)
  )
)

; Create 8 vertical buses from PD2 (going UP)
for(k 0 7
  let((pd2_out_x bus_x)
    pd2_out_x = pd2_x + (1.000 - k*0.125)
    bus_x = 18.0 + k*0.2
    pd2_bus_x[k] = bus_x

    wire = schCreateWire(cv "draw" "full"
      list(pd2_out_x:pd2_out_y
           pd2_out_x:(pd2_out_y + 1.0)
           bus_x:(pd2_out_y + 1.0)
           bus_x:vdd_rail_y)
      0.0625 0.0625 0)
    schCreateWireLabel(cv car(wire) bus_x:(pd2_out_y + 0.5)
      sprintf(nil "PD2_%d" k) "centerLeft" "R90" "fixed" 0.0625 nil)
  )
)

;==============================================
; STEP 4: Place AND gates with horizontal buses
;==============================================
printf("Placing 512 AND gates with horizontal buses...\n")

idx = 0
for(i 0 7
  for(j 0 7
    for(k 0 7
      ; Calculate WL Y position
      wl_y_pos = vdd_rail_y - 2.0 - idx * spacing_y
      horiz_bus_y = wl_y_pos + 0.5
      inst_name = sprintf(nil "I_AND_%d" idx)

      ; Place AND gate
      dbCreateInst(cv cv_and inst_name list(and_x_start wl_y_pos) "R0")

      ; AND pin positions
      let((and_in_x and_in1_y and_in2_y and_in3_y
           and_out_x and_out_y
           and_vdd_x and_vdd_y and_vss_x and_vss_y)

        and_in_x  = and_x_start - 1.525
        and_in1_y = wl_y_pos + 0.375  ; A1
        and_in2_y = wl_y_pos + 0.500  ; A2
        and_in3_y = wl_y_pos + 0.625  ; A3
        and_out_x = and_x_start - 0.100
        and_out_y = wl_y_pos + 0.562
        and_vdd_x = and_x_start - 1.087
        and_vdd_y = wl_y_pos + 1.125
        and_vss_x = and_x_start - 1.087
        and_vss_y = wl_y_pos - 0.125

        ; Create horizontal bus from PD0[i] to A1
        wire = schCreateWire(cv "draw" "full"
          list(pd0_bus_x[i]:horiz_bus_y
               and_in_x:horiz_bus_y
               and_in_x:and_in1_y)
          0.0625 0.0625 0)

        ; Create horizontal bus from PD1[j] to A2
        wire = schCreateWire(cv "draw" "full"
          list(pd1_bus_x[j]:horiz_bus_y
               and_in_x:horiz_bus_y
               and_in_x:and_in2_y)
          0.0625 0.0625 0)

        ; Create horizontal bus from PD2[k] to A3
        wire = schCreateWire(cv "draw" "full"
          list(pd2_bus_x[k]:horiz_bus_y
               and_in_x:horiz_bus_y
               and_in_x:and_in3_y)
          0.0625 0.0625 0)

        ; Wire ZN to WL output
        wire = schCreateWire(cv "draw" "full"
          list(and_out_x:and_out_y
               (and_out_x + 4.0):and_out_y)
          0.0625 0.0625 0)
        schCreateWireLabel(cv car(wire) (and_out_x + 1.0):and_out_y
          sprintf(nil "WL<%d>" idx) "centerLeft" "R0" "fixed" 0.0625 nil)

        ; Power connections
        wire = schCreateWire(cv "draw" "full"
          list(and_vdd_x:and_vdd_y and_vdd_pwr_x:and_vdd_y)
          0.0625 0.0625 0)
        wire = schCreateWire(cv "draw" "full"
          list(and_vss_x:and_vss_y and_vss_pwr_x:and_vss_y)
          0.0625 0.0625 0)
      )
```

```
        idx = idx + 1
        when(mod(idx 64) == 0
          printf("  Connected WL<%d>...\n" idx-1)
        )
      )
    )
  )

;==============================================
; STEP 5: Input Pins
;==============================================
printf("Creating input pins...\n")
pin_master = dbOpenCellView("basic" "ipin" "symbol" nil "r")

let((pin_y)
  pin_y = vss_rail_y - 1.0

  ; PD0 inputs
  schCreatePin(cv pin_master "CLK" "input" nil list(pd0_x - 0.375 pin_y) "R0")
  wire = schCreateWire(cv "draw" "full"
    list((pd0_x - 0.375):(pd0_y - 0.963) (pd0_x - 0.375):pin_y) 0.0625 0.0625 0)

  schCreatePin(cv pin_master "A0" "input" nil list(pd0_x + 0.000 pin_y) "R0")
  wire = schCreateWire(cv "draw" "full"
    list((pd0_x + 0.000):(pd0_y - 0.963) (pd0_x + 0.000):pin_y) 0.0625 0.0625 0)

  schCreatePin(cv pin_master "A1" "input" nil list(pd0_x - 0.125 pin_y) "R0")
  wire = schCreateWire(cv "draw" "full"
    list((pd0_x - 0.125):(pd0_y - 0.963) (pd0_x - 0.125):pin_y) 0.0625 0.0625 0)

  schCreatePin(cv pin_master "A2" "input" nil list(pd0_x - 0.250 pin_y) "R0")
  wire = schCreateWire(cv "draw" "full"
    list((pd0_x - 0.250):(pd0_y - 0.963) (pd0_x - 0.250):pin_y) 0.0625 0.0625 0)
  ; PD1 inputs
  schCreatePin(cv pin_master "A3" "input" nil list(pd1_x + 0.938 pin_y) "R0")
  wire = schCreateWire(cv "draw" "full"
    list((pd1_x + 0.938):(pd1_y - 1.025) (pd1_x + 0.938):pin_y) 0.0625 0.0625 0)

  schCreatePin(cv pin_master "A4" "input" nil list(pd1_x + 0.812 pin_y) "R0")
  wire = schCreateWire(cv "draw" "full"
    list((pd1_x + 0.812):(pd1_y - 1.025) (pd1_x + 0.812):pin_y) 0.0625 0.0625 0)

  schCreatePin(cv pin_master "A5" "input" nil list(pd1_x + 0.688 pin_y) "R0")
  wire = schCreateWire(cv "draw" "full"
    list((pd1_x + 0.688):(pd1_y - 1.025) (pd1_x + 0.688):pin_y) 0.0625 0.0625 0)

  ; PD2 inputs
  schCreatePin(cv pin_master "A6" "input" nil list(pd2_x + 0.562 pin_y) "R0")
  wire = schCreateWire(cv "draw" "full"
    list((pd2_x + 0.562):(pd2_y - 1.025) (pd2_x + 0.562):pin_y) 0.0625 0.0625 0)

  schCreatePin(cv pin_master "A7" "input" nil list(pd2_x + 0.438 pin_y) "R0")
  wire = schCreateWire(cv "draw" "full"
    list((pd2_x + 0.438):(pd2_y - 1.025) (pd2_x + 0.438):pin_y) 0.0625 0.0625 0)

  schCreatePin(cv pin_master "A8" "input" nil list(pd2_x + 0.312 pin_y) "R0")
  wire = schCreateWire(cv "draw" "full"
    list((pd2_x + 0.312):(pd2_y - 1.025) (pd2_x + 0.312):pin_y) 0.0625 0.0625 0)
)

;==============================================
; STEP 6: Output Pins WL<0>..WL<511>
;==============================================
printf("Creating output pins...\n")
pin_master = dbOpenCellView("basic" "opin" "symbol" nil "r")

for(i 0 511
  let((pin_name y_pos wl_pin_x)
    pin_name = sprintf(nil "WL<%d>" i)
    y_pos = vdd_rail_y - 2.0 - i * spacing_y
    wl_pin_x = and_x_start - 0.100 + 4.5

    schCreatePin(cv pin_master pin_name "output" nil
      list(wl_pin_x y_pos + 0.562) "R0")
  )
)

;==============================================
; STEP 7: Power Pins
;==============================================
printf("Creating power pins...\n")
pin_master = dbOpenCellView("basic" "iopin" "symbol" nil "r")

schCreatePin(cv pin_master "VDD" "inputOutput" nil list(-0.5 vdd_rail_y) "R0")
schCreatePin(cv pin_master "VSS" "inputOutput" nil list(-0.5 vss_rail_y) "R0")

wire = schCreateWire(cv "draw" "full"
  list((-0.5):vdd_rail_y 0.0:vdd_rail_y) 0.125 0.125 0)
wire = schCreateWire(cv "draw" "full"
  list((-0.5):vss_rail_y 0.0:vss_rail_y) 0.125 0.125 0)

;==============================================
; Save and Close
;==============================================
dbClose(cv_pd0)
dbClose(cv_pd1)
dbClose(cv_pd2)
dbClose(cv_and)
dbSave(cv)
dbClose(cv)
```

```
        printf("\n=== 512-Row Decoder Complete (Correct Routing)! ===\n")
        printf("✓ 3 PreDecoders placed at bottom\n")
        printf("✓ 24 vertical buses running UP from PD outputs\n")
        printf("✓ 512 horizontal buses feeding AND gates\n")
        printf("✓ 512 AND gates placed\n")
        printf("✓ Power distribution complete\n")
        printf("✓ All pins created\n\n")
    )
)
```

Rationale behind sizing:

- Predecoder inverters use minimum fin count to save area and dynamic power, since they drive only local gates.
- Predecoder AND gates use 4 fin devices in both pull down and pull up. This limits delay through the stacked NMOS path.
- Final wordline drivers use 8 fin devices to drive the long wordline plus bitcell gate capacitances.
- Two driver stages (inv4 then Inv8) were chosen instead of one huge stage to:
    - Reduce input capacitance seen by the AND outputs.
    - Smooth the transition between low capacitance internal nodes and high capacitance wordlines.
    - Achieve better energy delay trade off than a single large inverter.

The HSPICE netlist for rowDecoder512 defines all devices at the transistor level, with the correct TSMC 16 nm models.

Simulation setup:

- Environment:
    - Process: typical corner of the academic 16 nm PDK.

    - Supply: VDD = 0.8 V.
    - Temperature: 80 °C for worst case delay and noise margin.

- Stimulus:
    - Sweep A[8:0] through random access patterns.
    - Toggle CLK to control access timing.
    - Include realistic wordline loading, modeled as capacitors extracted from a representative bitcell row or as simple lumped loads.

```
.TEMP 80
.OPTION POST
.lib
/soft/ece_soft_2/TSMCHOME/Executable_Package/Collaterals/Tech/SPICE/N16ADFP_S
PICE_MODEL/n16adfp_spice_model_v1d0_usage.l TTMacro_MOS_MOSCAP
.INCLUDE rowDecoder.sp
.PARAM VDD=0.8

VDD vdd! 0 VDD
VSS vss! 0 0

*CLK - continuous clock with 2ns period
VCLK CLK 0 VDD

*Address pattern using PULSE sources with fast rise/fall times
*addr 0: A[8:0]=000000000 from 0-8ns
*addr 1: A[8:0]=000000001 from 8-16ns
*addr 8: A[8:0]=000001000 from 16-24ns
*addr 64: A[8:0]=001000000 from 24-32ns
*addr 511: A[8:0]=111111111 from 32-42ns
```

```
*A0: 0,1,0,0,1
VA0 A0 0 PULSE(0 VDD 8n 10p 10p 8n 100n)

*A1-A2: stay 0 until addr 511
VA1 A1 0 PULSE(0 VDD 32n 10p 10p 10n 100n)
VA2 A2 0 PULSE(0 VDD 32n 10p 10p 10n 100n)

*A3: 0,0,1,0,1
VA3 A3 0 PULSE(0 VDD 16n 10p 10p 8n 100n)

*A4-A5: stay 0 until addr 511
VA4 A4 0 PULSE(0 VDD 32n 10p 10p 10n 100n)
VA5 A5 0 PULSE(0 VDD 32n 10p 10p 10n 100n)

*A6: 0,0,0,1,1
VA6 A6 0 PULSE(0 VDD 24n 10p 10p 18n 100n)

*A7-A8: stay 0 until addr 511
VA8 A8 0 PULSE(0 VDD 32n 10p 10p 10n 100n)

*Add load capacitors on outputs to see cleaner signals

.TRAN 0.1n 42n
.END
```

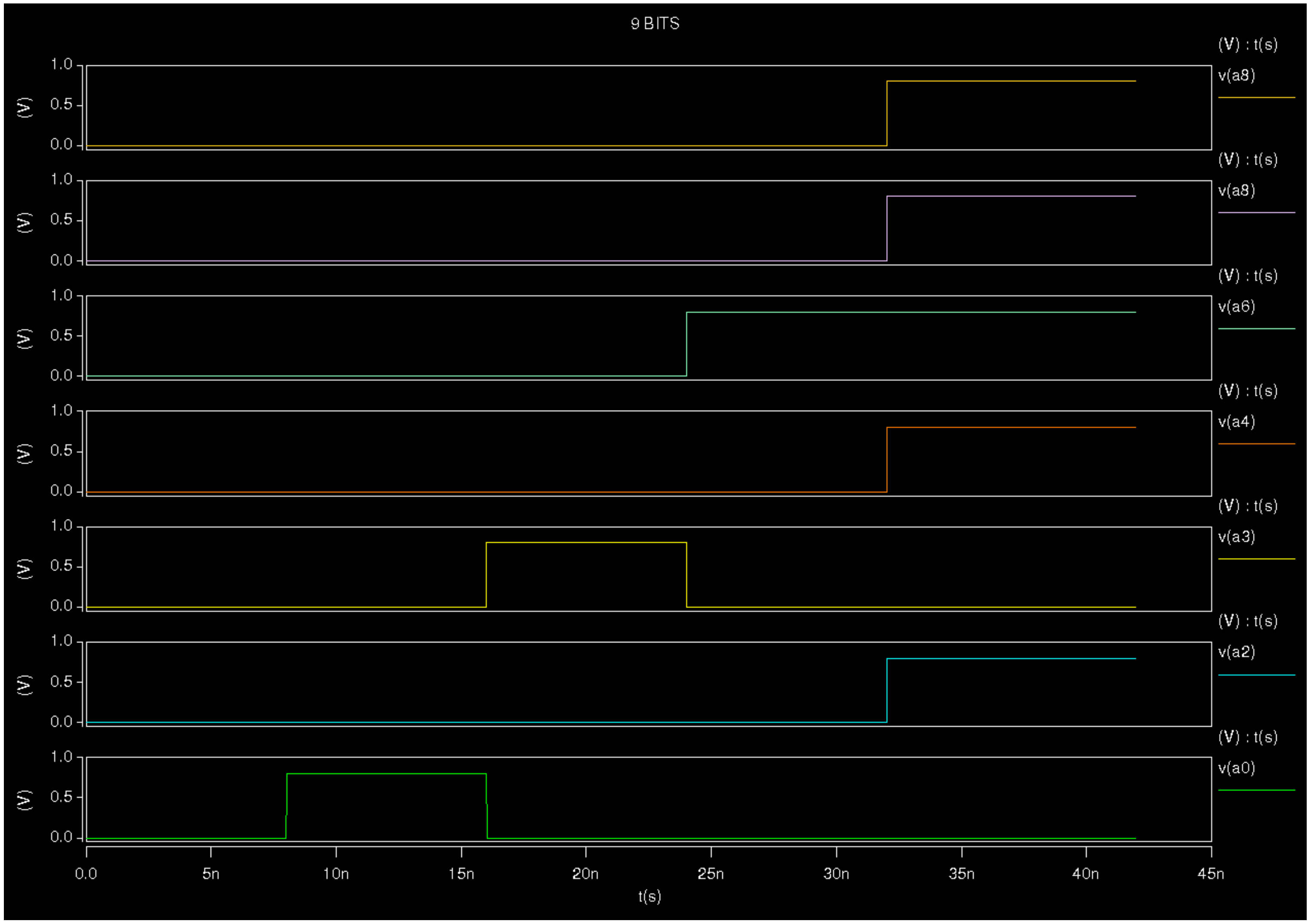


Figure: Input signals transitioning

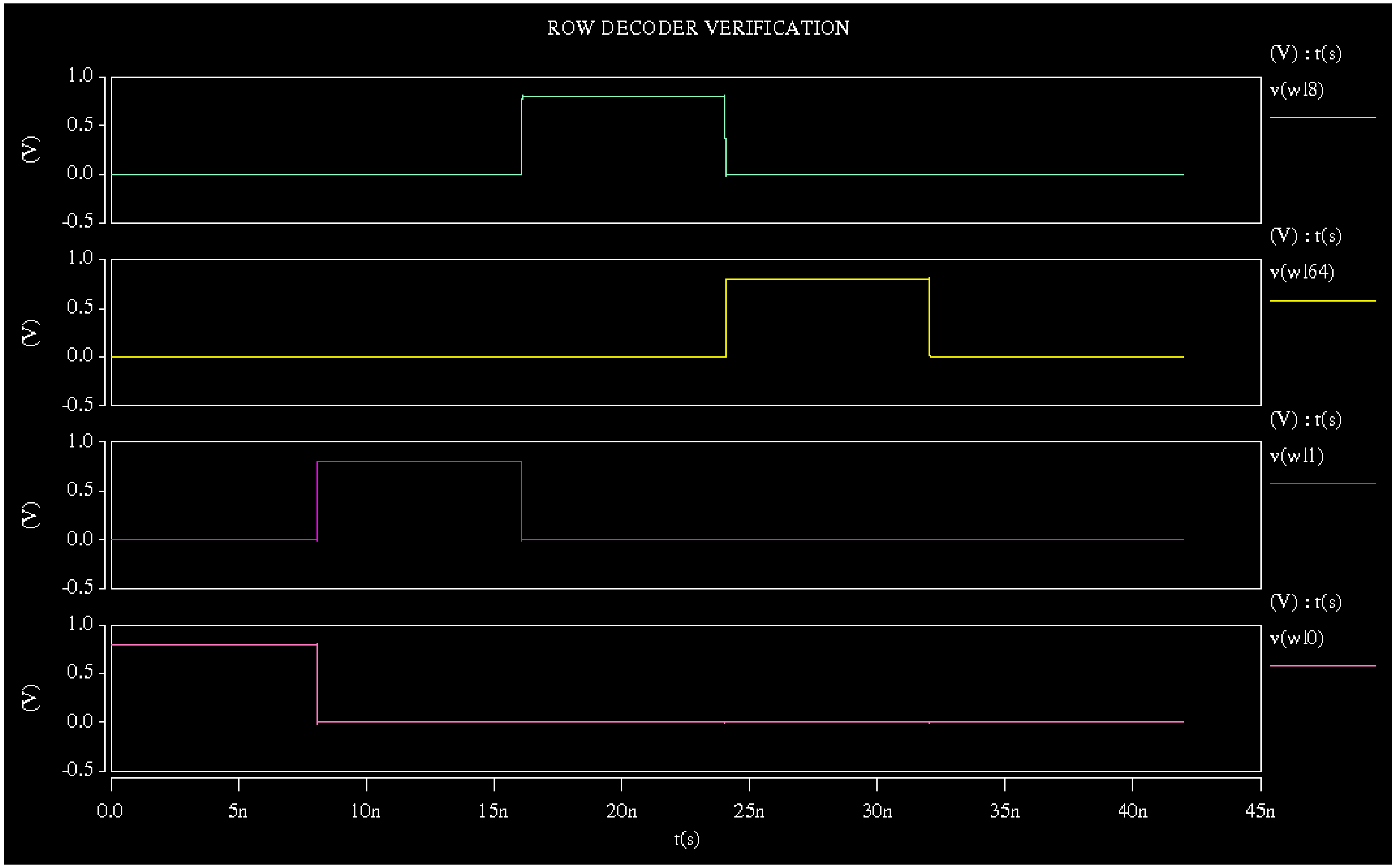


Figure: Activated WL Signals

Preliminary layout strategy

The layout work focuses on a clean, abuttable row decoder block that can drop in next to the SRAM array.

Block partitioning

- Three main tiled subblocks:
    - subblock1 placed closest to the column of address drivers and clock.
    - subblock2 and subblock3 stacked horizontally.
    - The 512 final AND and driver chains arranged in a dense grid behind the predecoders.

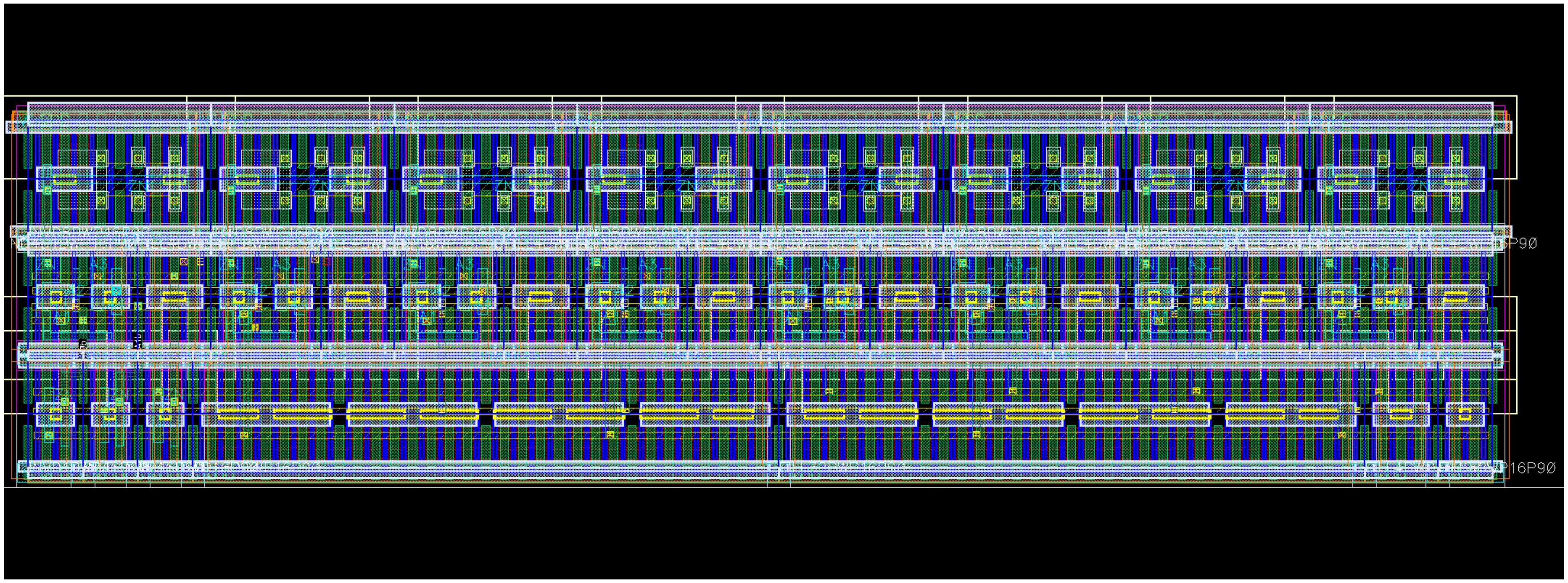


Figure: Predecoder 1 - Layout

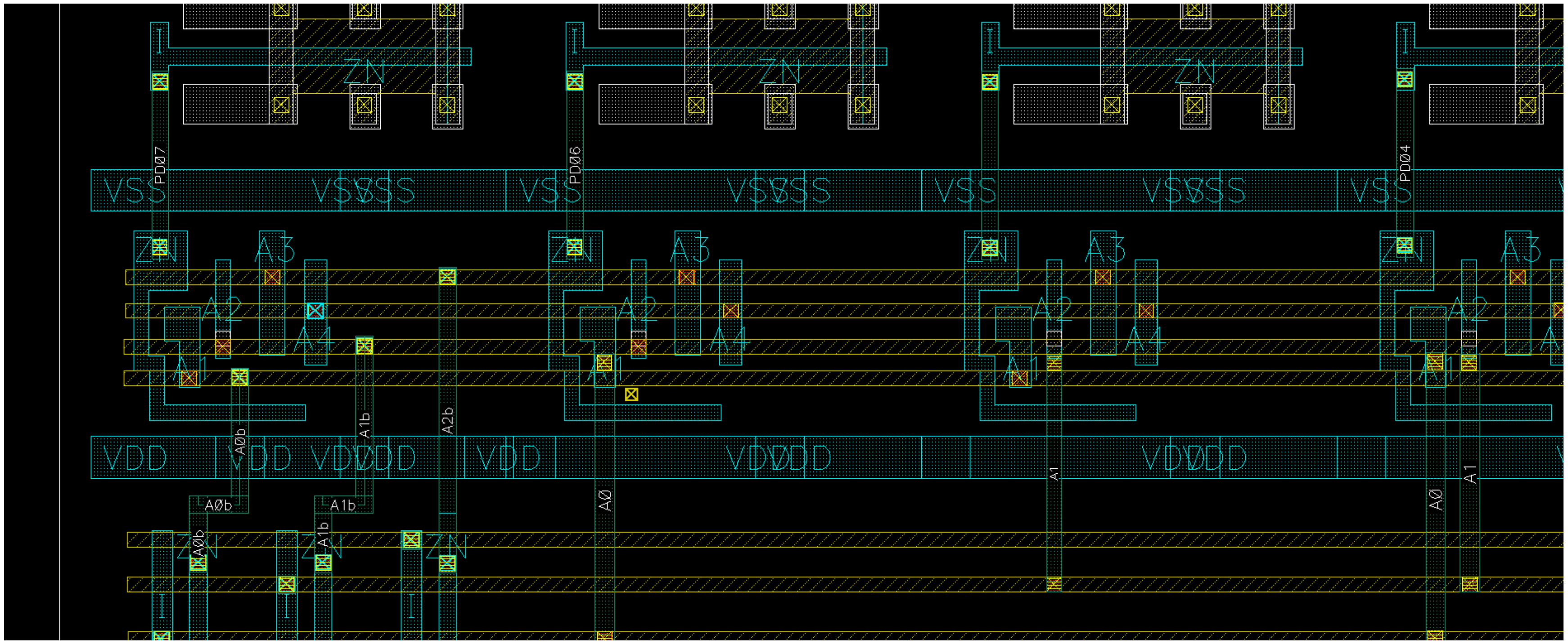


Figure: Predecoder 1 – Metal Connections

DRC and LVS status

- Each primitive cell (inverter_schematic, inv2, inv4, Inv8, and the AND gate layouts) is DRC and LVS clean.
- Predecoder blocks subblock1, subblock2, and subblock3 are being verified at layout level.
- The full rowDecoder512 layout in Phase 2 is a preliminary version

- Global DRC and LVS for the entire 512 row array will be finalized when the bitcell array is fully integrated.

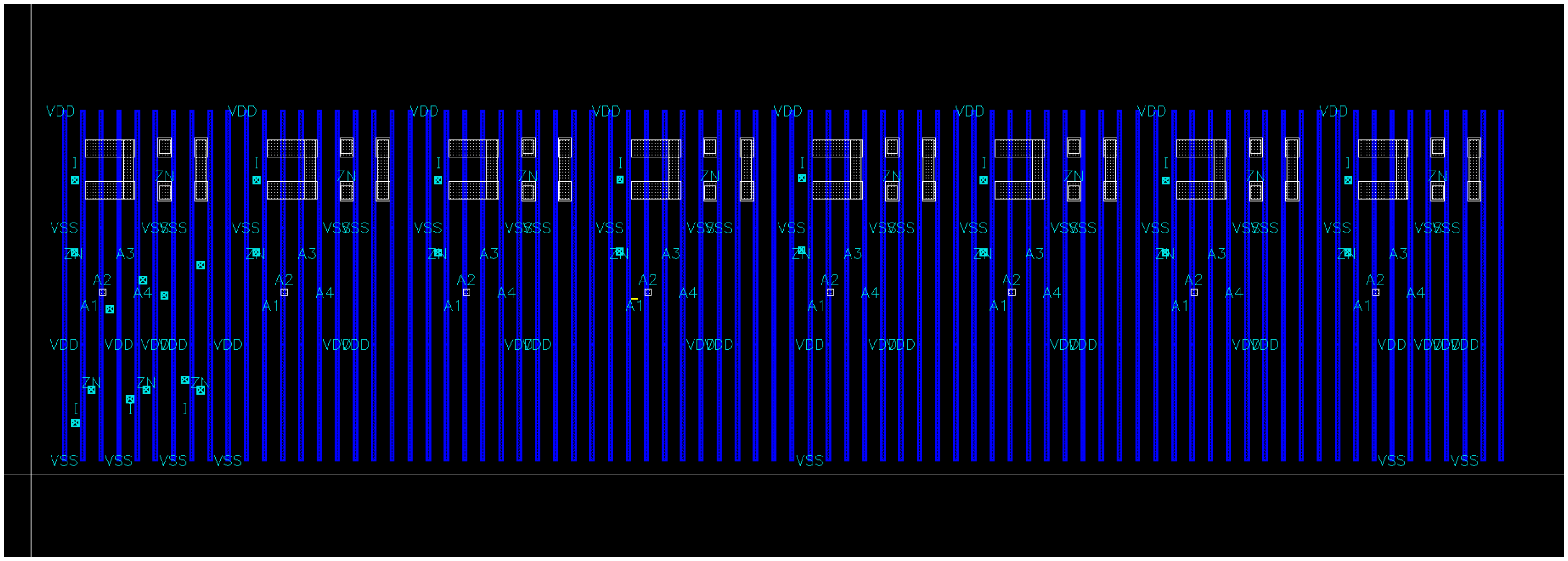


Figure: Filler Cell inserted alternately between each NAND4 gate

Global Net Conflicts and LVS Challenges

My schematic uses global net names VDD! and VSS!. These names force every instance to attach to the same implicit global rails. This simplifies simulation in early stages. It creates problems during layout verification.

The layout uses library cells from the N16ADFP standard cell library. These cells follow the foundry naming rules for power. They use pins named VDD and VSS tied to specific metal rails inside each cell. These rails are not the same as the global schematic rails VDD! and VSS!.

LVS compares the schematic and the extracted layout. If one side uses global power names and the other side uses explicit power pins, the tool cannot match nets. The checker sees the global rail in the schematic, but sees multiple segmented power rails in the layout. It raises unmatched net errors. This blocks LVS closure.

DRC is also affected. Some standard cells expect continuous power metal at fixed heights. When custom devices connect to global rails in irregular ways, the metal spacing and continuity rules break. The tool cannot confirm that the power delivery is safe. This results in DRC spacing and enclosure violations near the custom gates.

Symmetric Filler Use in Predecoder Slices

The predecoder is tiled horizontally. Each slice must keep uniform height and symmetric diffusion edges. Filler cells are added on both sides of each slice. This maintains continuity of diffusion and local power rails. It also prevents jogs in M1 that can violate enclosure rules. These symmetric fillers reduce routing detours and help keep timing similar across all eight predecoder lines.

Integration Strategy for the Full Block

The SKILL code attempted earlier helped place repeated logic, but routing mismatches and coordinate errors created gaps and floating nets. These issues delayed clean LVS. To avoid this, the automated SKILL flow will be used only for integrating major blocks with the final 6T SRAM array.

The core idea is simple. The predecoder macro, the 512 row decoder grid, and the 6T array will be placed as stable anchor blocks. SKILL will generate the repeated vertical wordline routing and connect wls[k] and wln[k] into the array pins. This approach keeps the large repetitive routing consistent. It avoids manual mistakes. It reduces the chance of pin mismatches at the array boundary.

The blocks will share a single power grid built from the N16ADFP library rails. This removes global net conflicts. The final layout will have uniform power delivery and match the schematic power connections exactly. This supports clean LVS and DRC closure for the entire SRAM macro.