ESE 370: CIRCUIT-LEVEL OPTIMIZATION FOR DIGITAL SYSTEMS

Project 2 Milestone: FIFO Queue

Mauricio Mutai, Jack Harkins

Instructor: Dr. Tania Khanna

TA: Martin Deng

Date: 11/26/16

December 12, 2016

Design Schematics

Top-Level Design

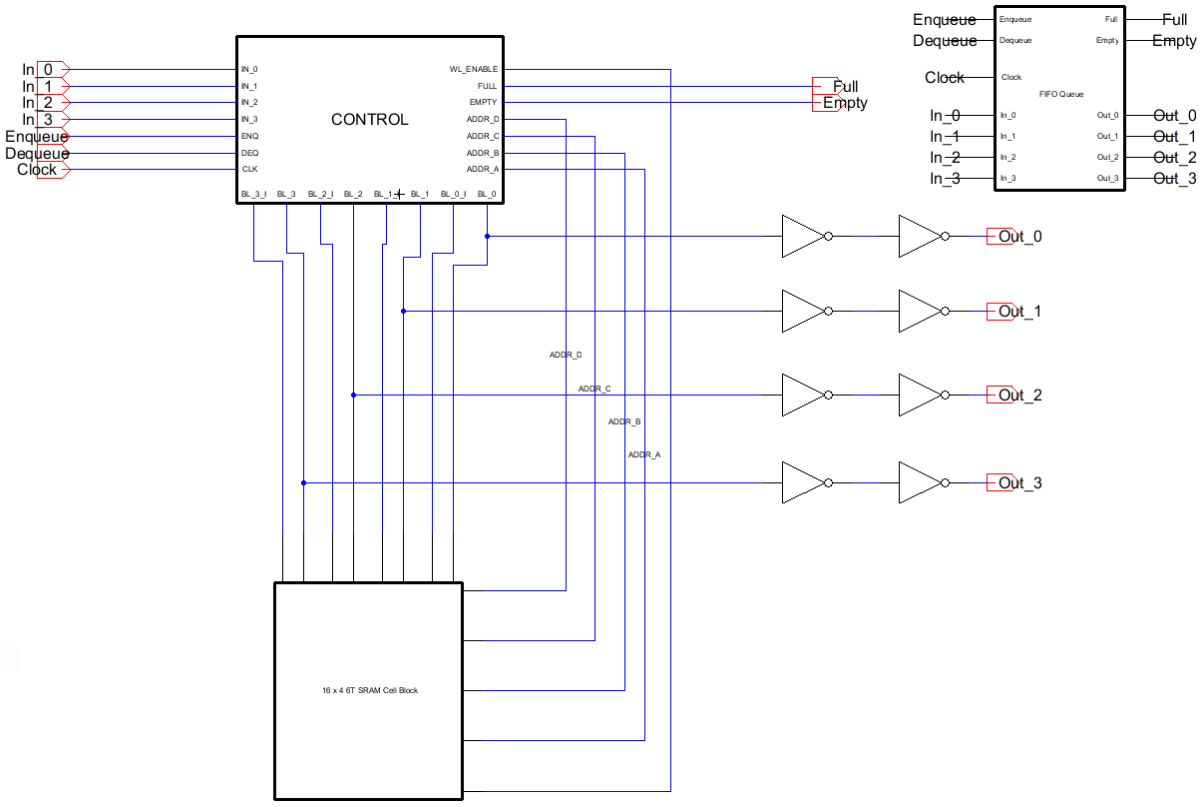

Figure 1: Queue top-level schematic

Memory Cell Block

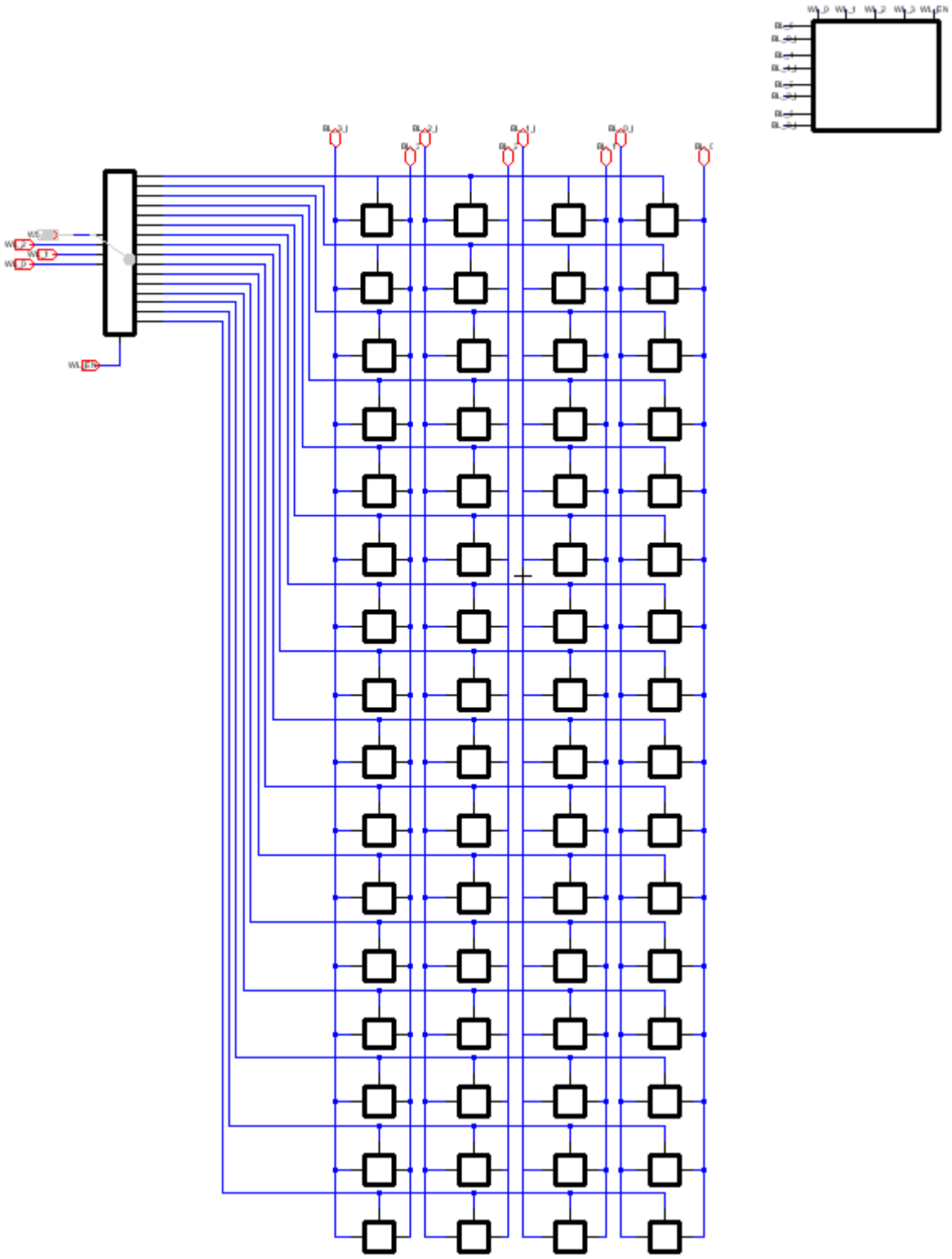

Figure 2: Memory block toplevel schematic

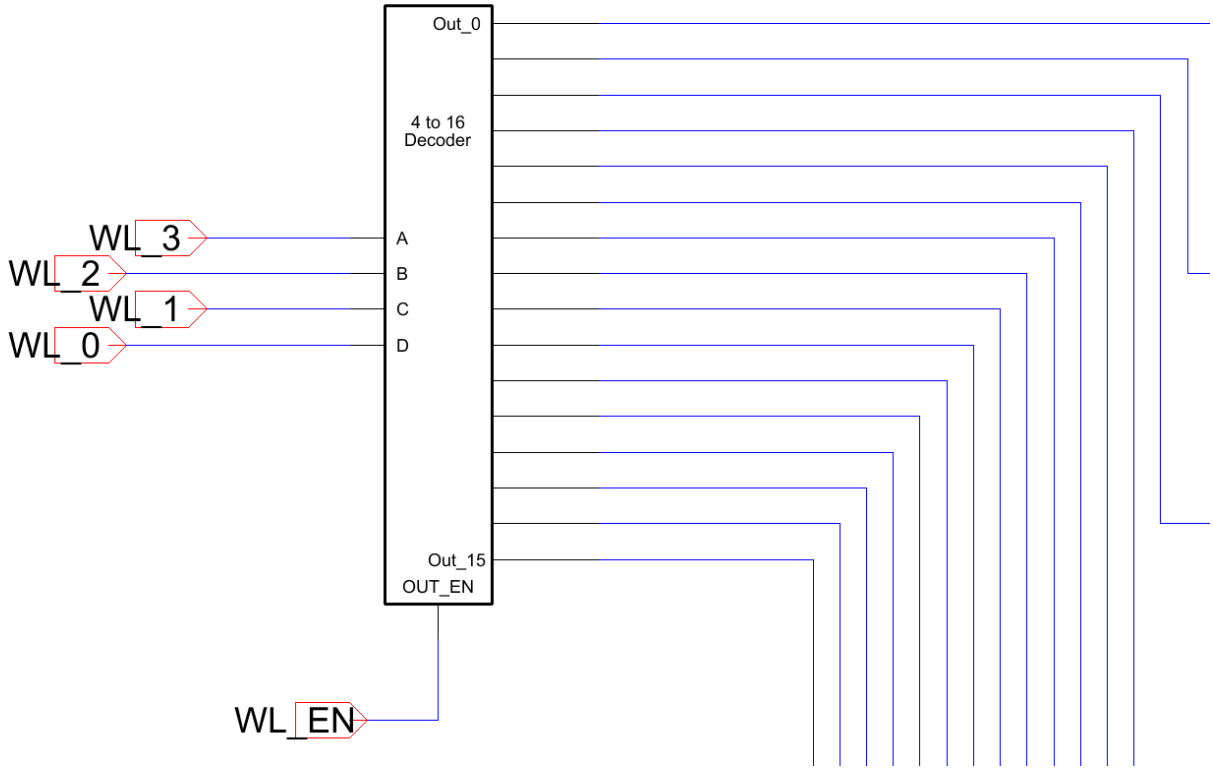

Figure 3: Memory block schematic in detail, decoder

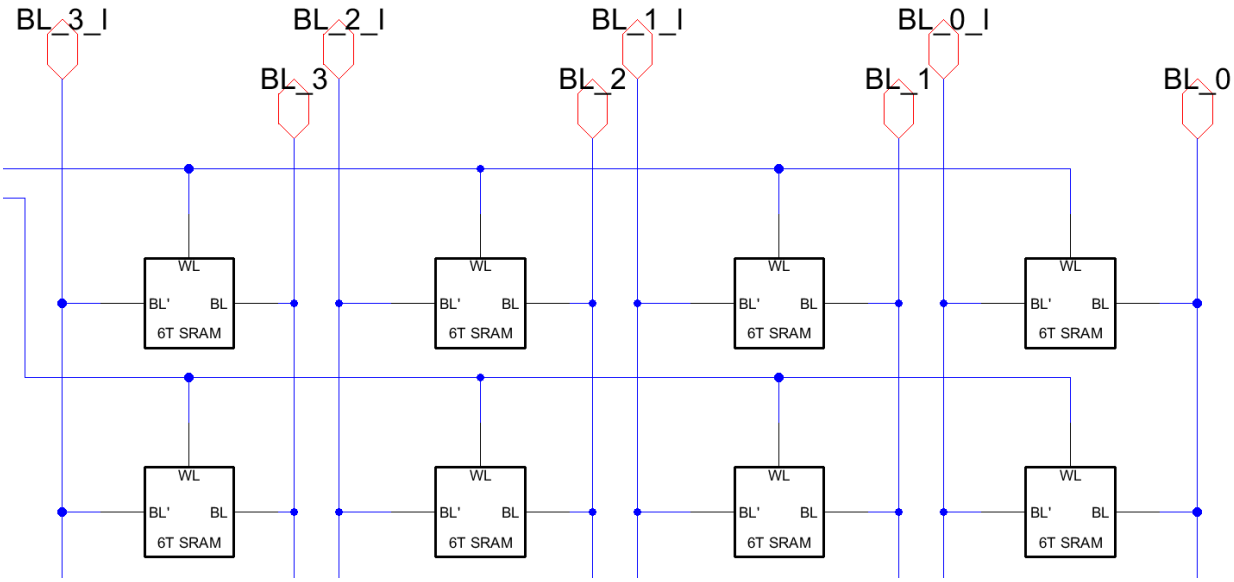

Figure 4: Memory block schematic in detail, word slice

6T SRAM Cell

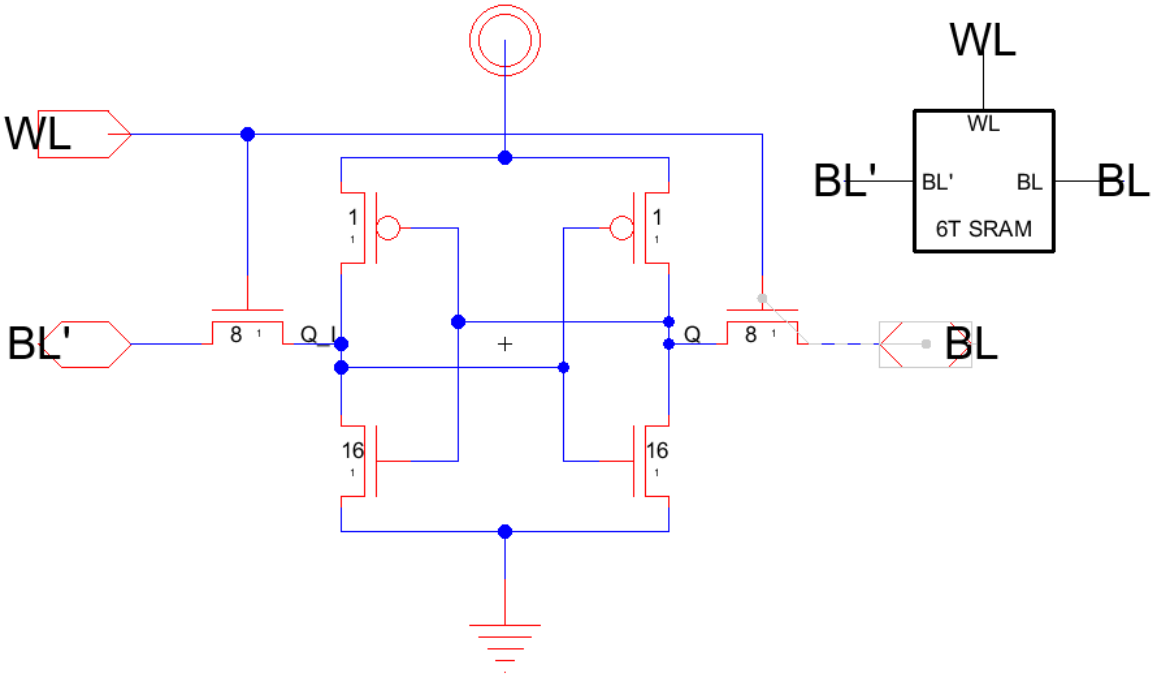

Figure 5: SRAM cell schematic

Control Block

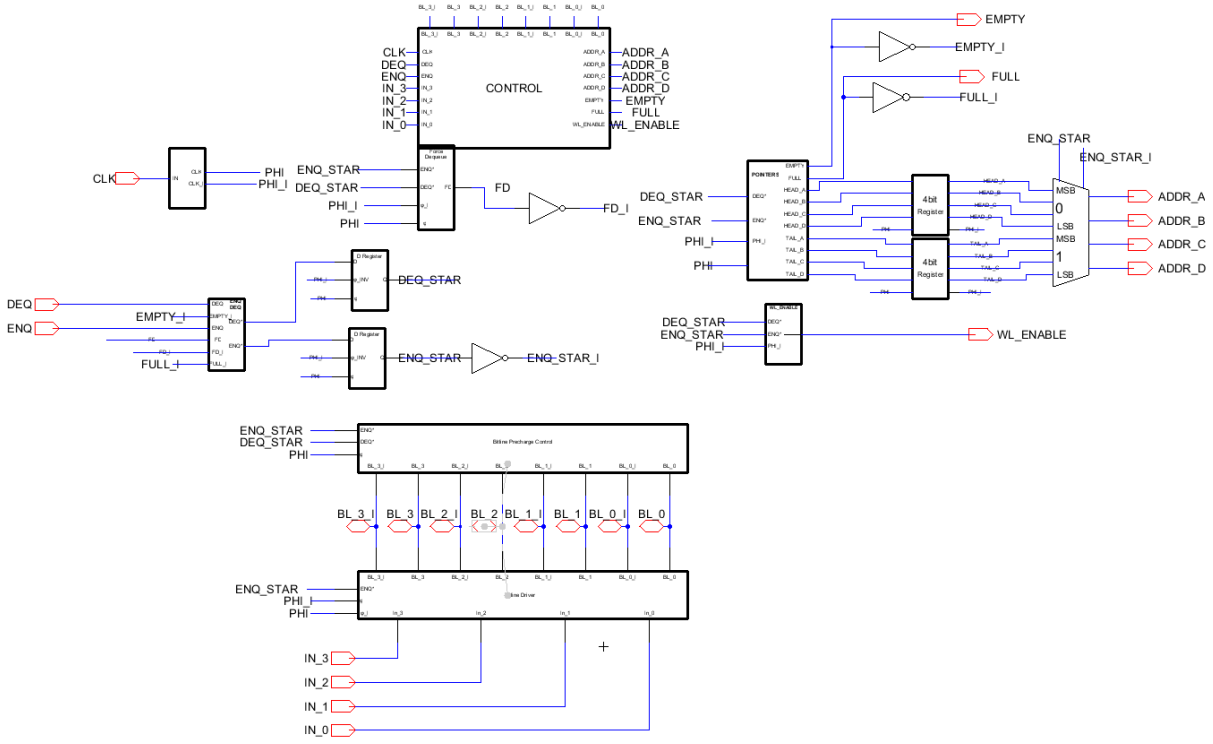

Figure 6: Control block toplevel schematic

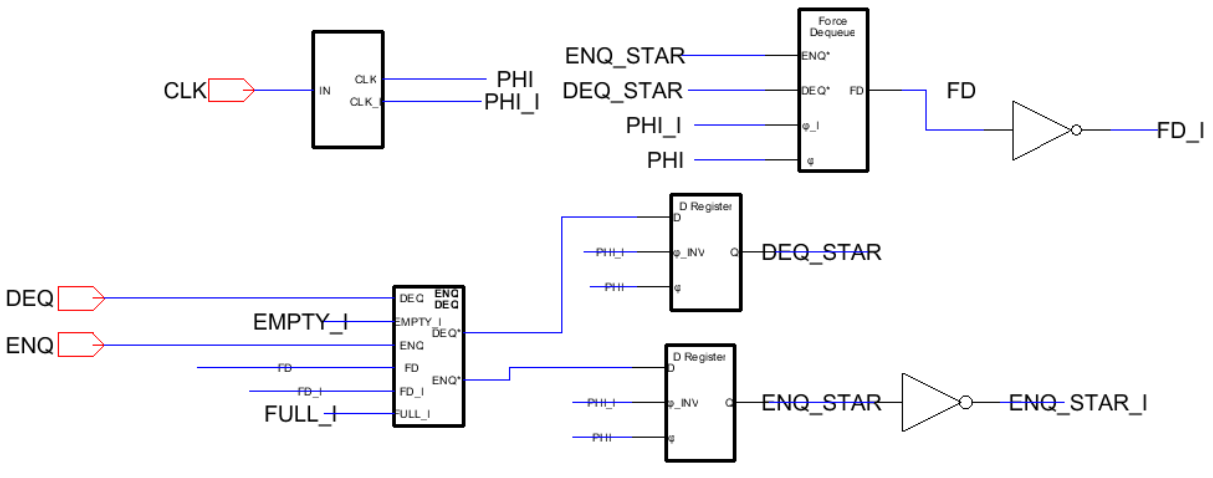

Figure 7: Control block schematic in detail, FD, clock generator, and ENQ*/DEQ* generation

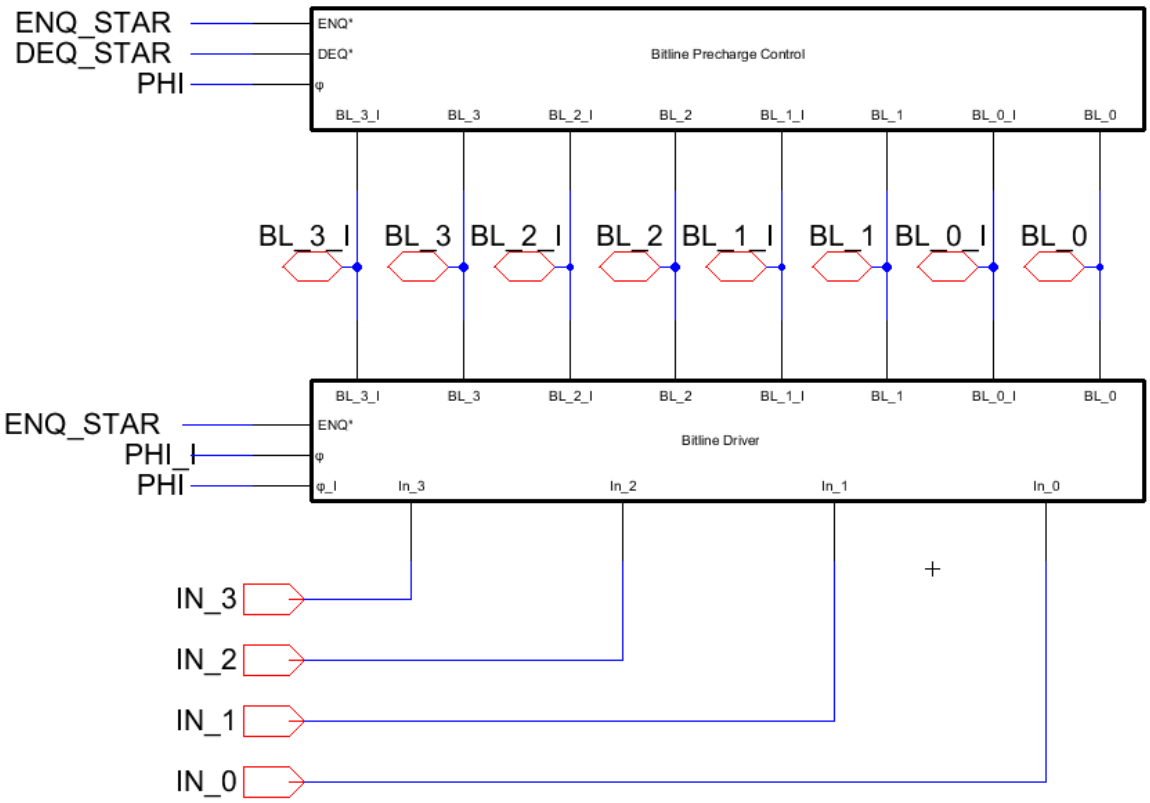

Figure 8: Control block schematic in detail, bitline driver and precharger

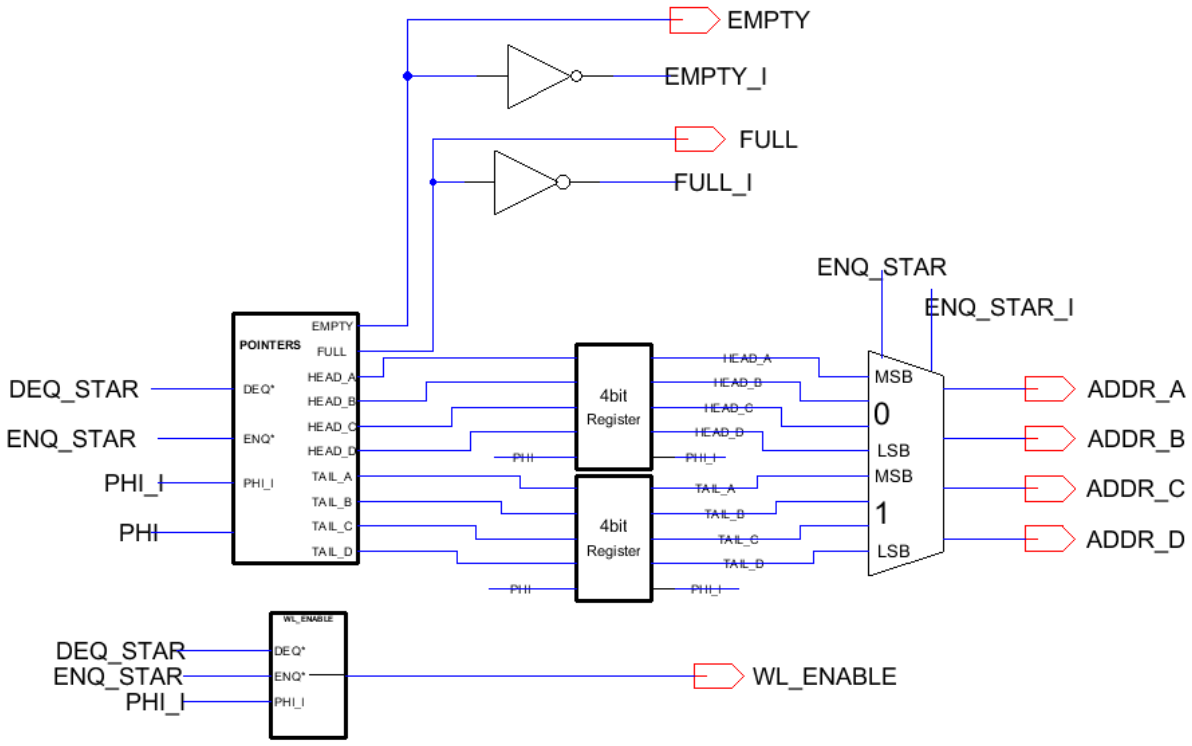

Figure 9: Control block schematic in detail, pointers, WL_EN, and EMPTY/FULL generation

Clock Generator

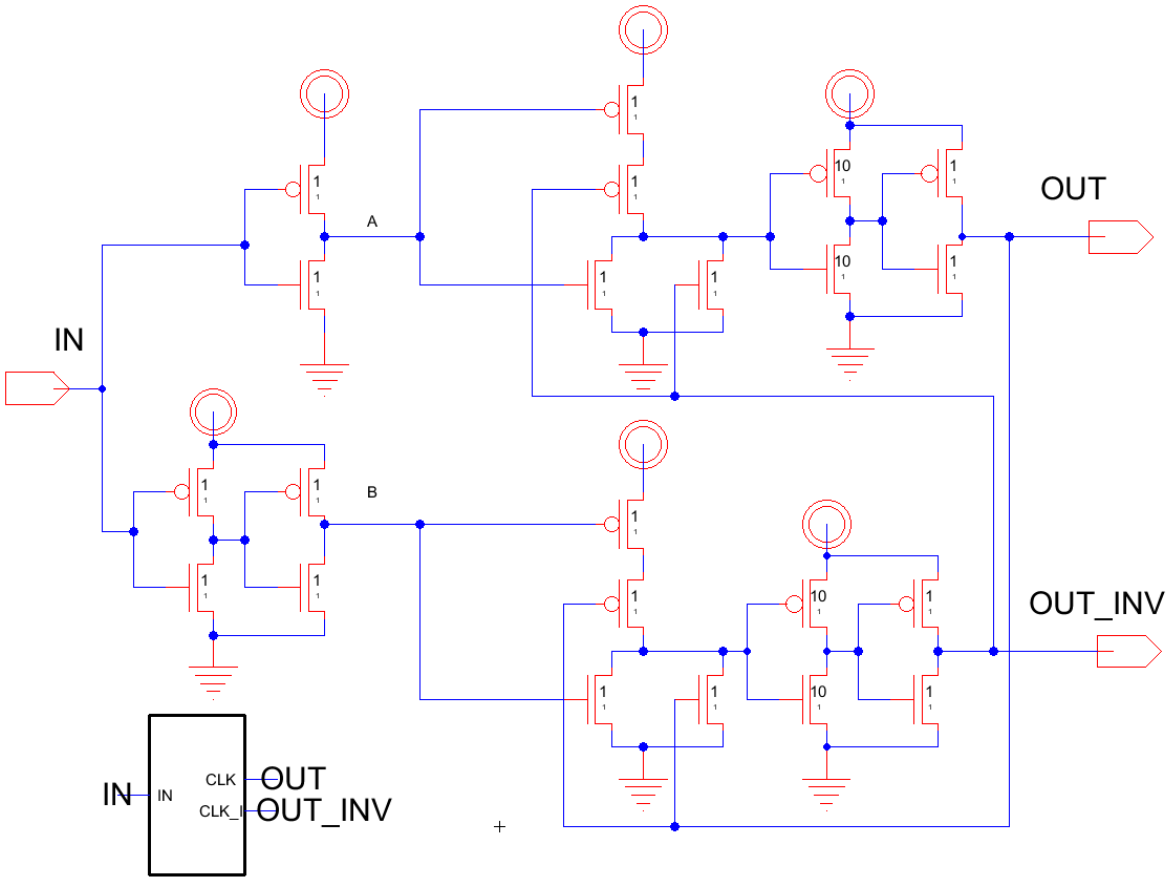

Figure 10: Clock generator schematic

Force Dequeue (FD)

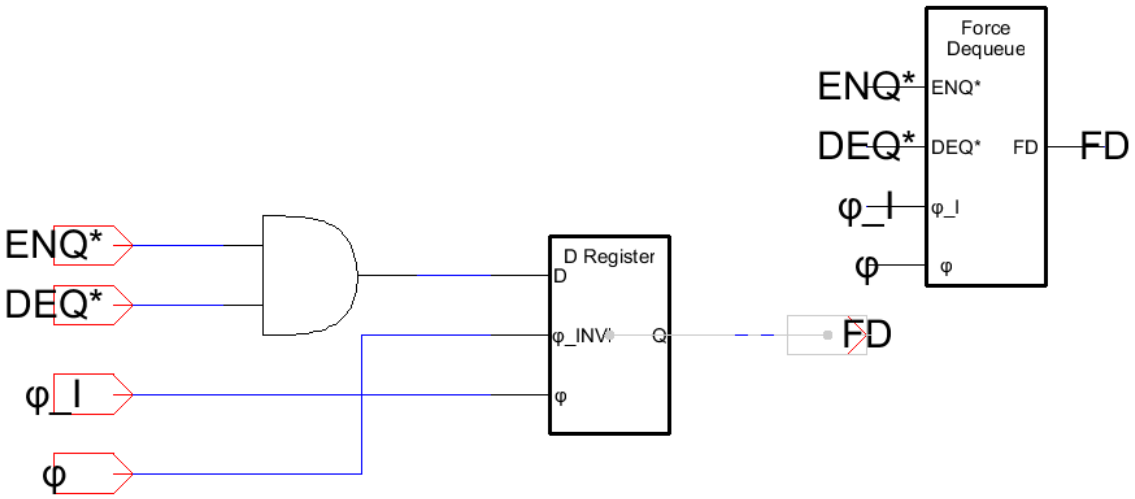

Figure 11: Force dequeue generation schematic

Pointers

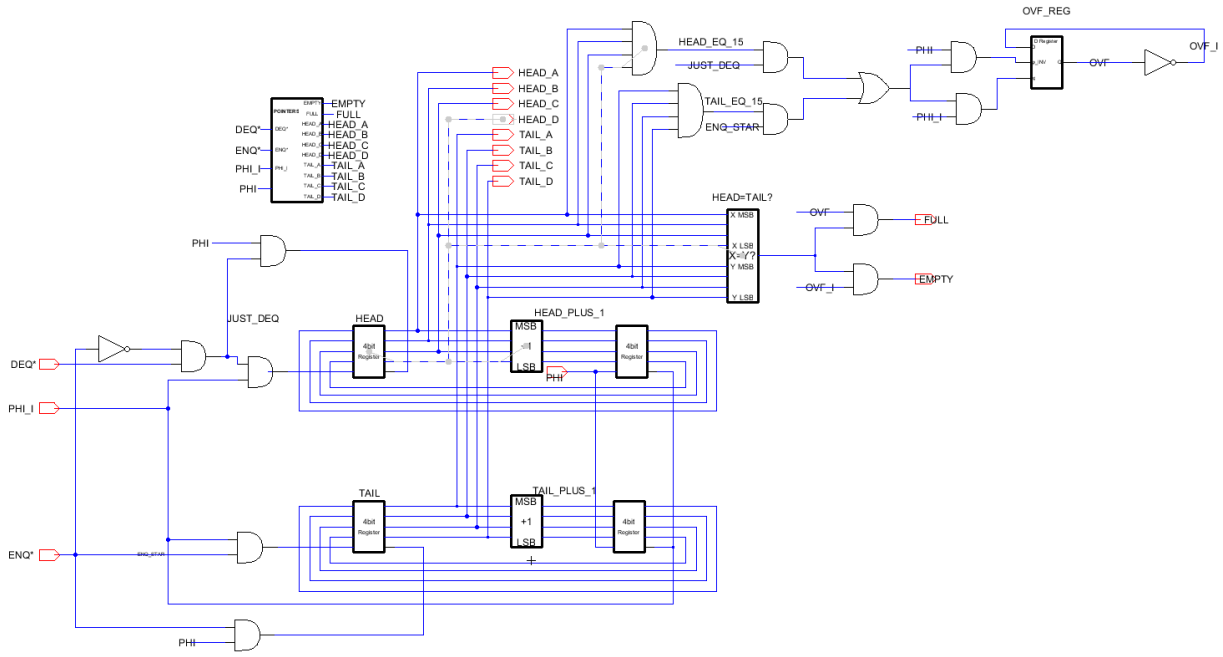

Figure 12: Pointers toplevel schematic

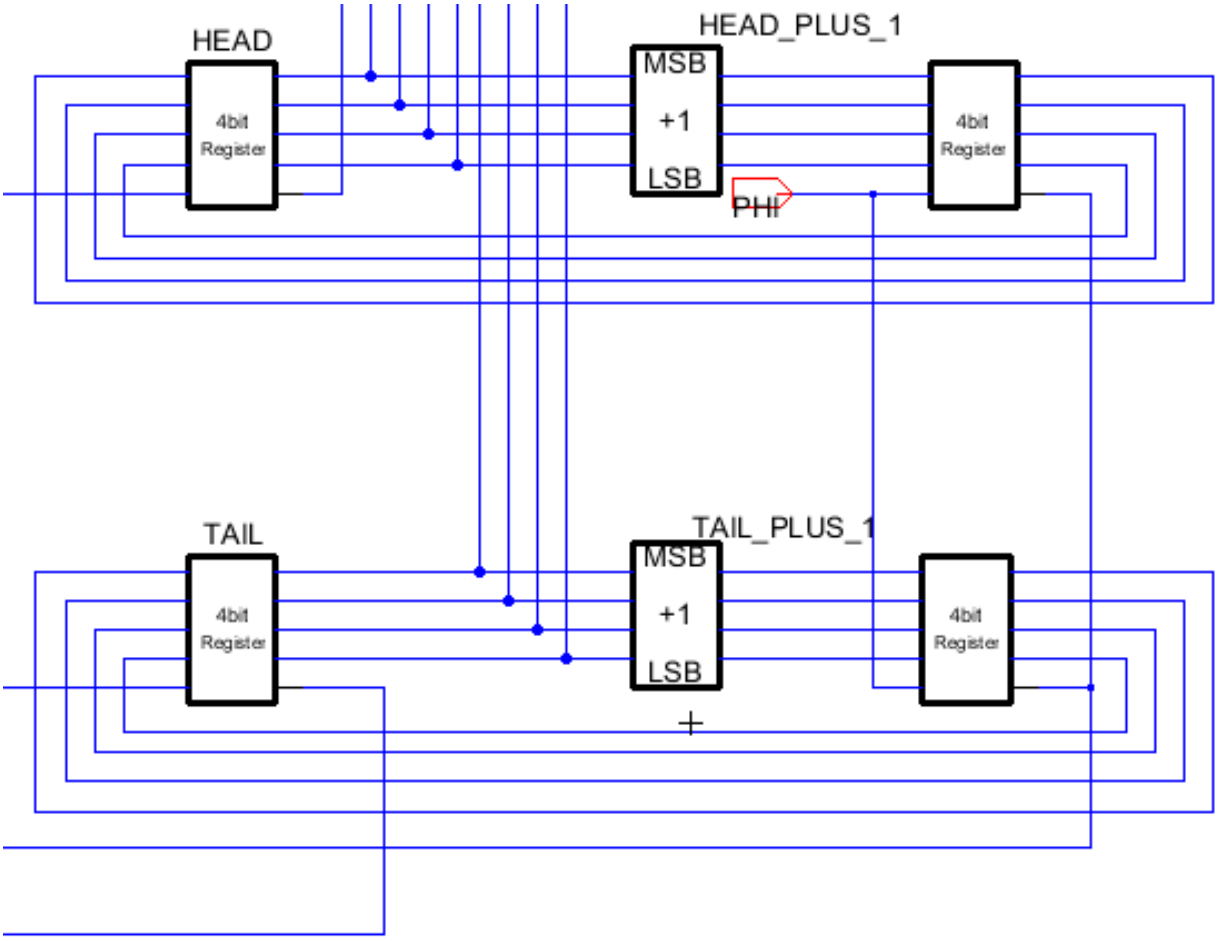

Figure 13: Pointers schematic in detail, head/tail state

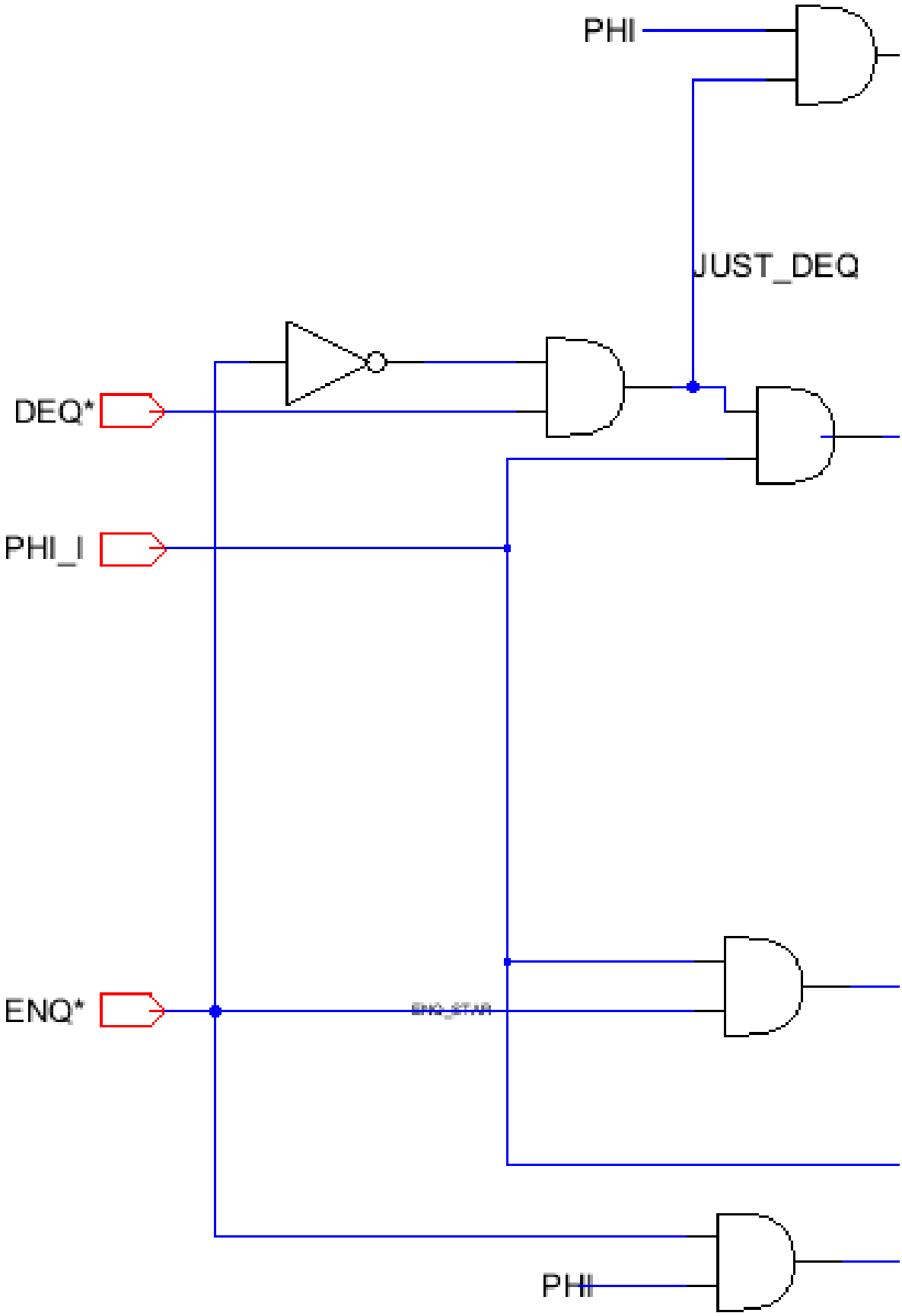

Figure 14: Pointers schematic in detail, inputs

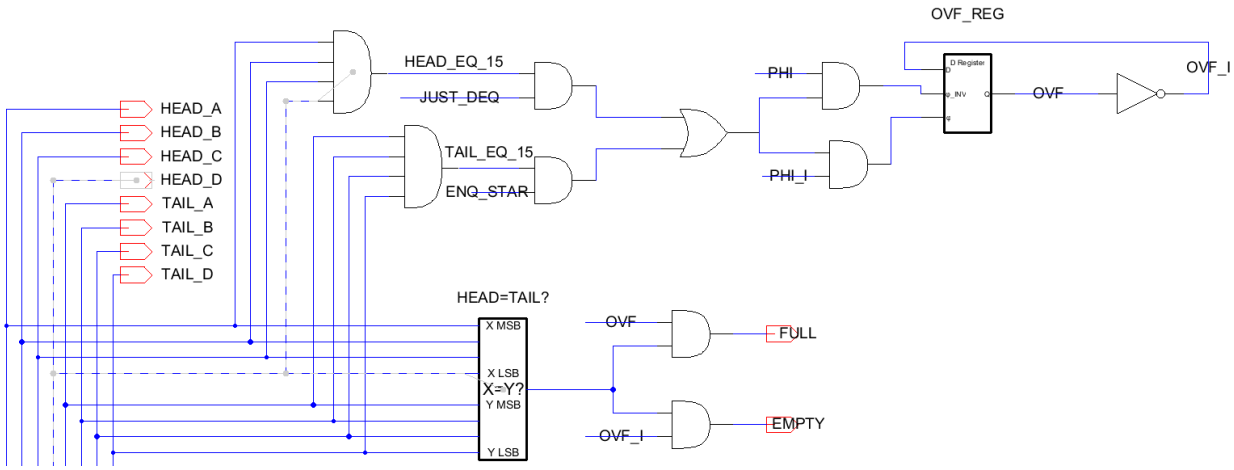

Figure 15: Pointers schematic in detail, overflow handler

D Latch

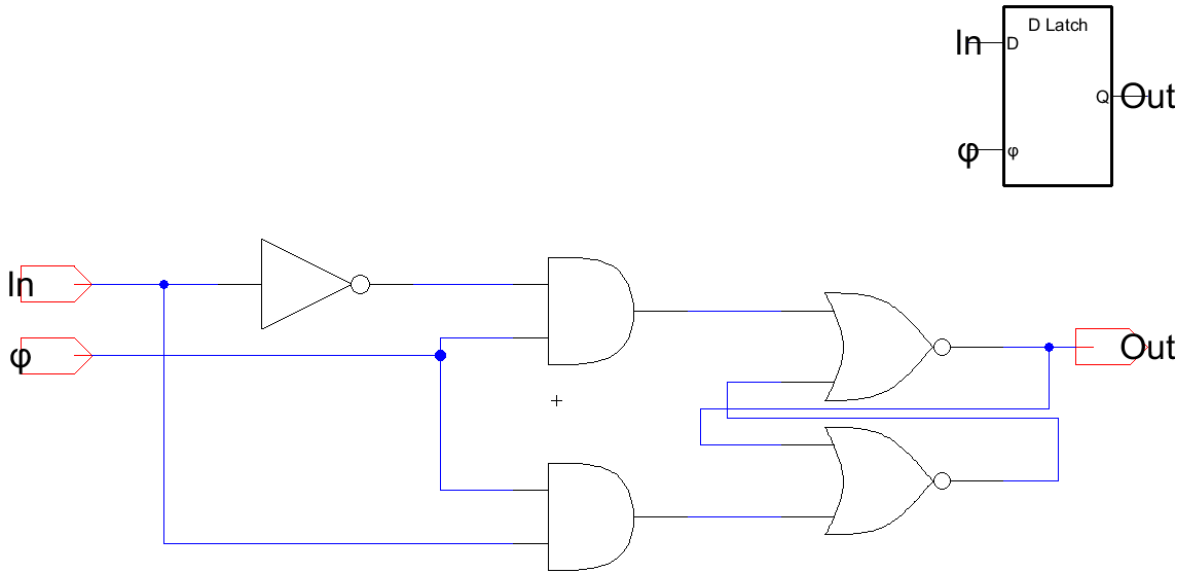

Figure 16: D latch schematic

D Register

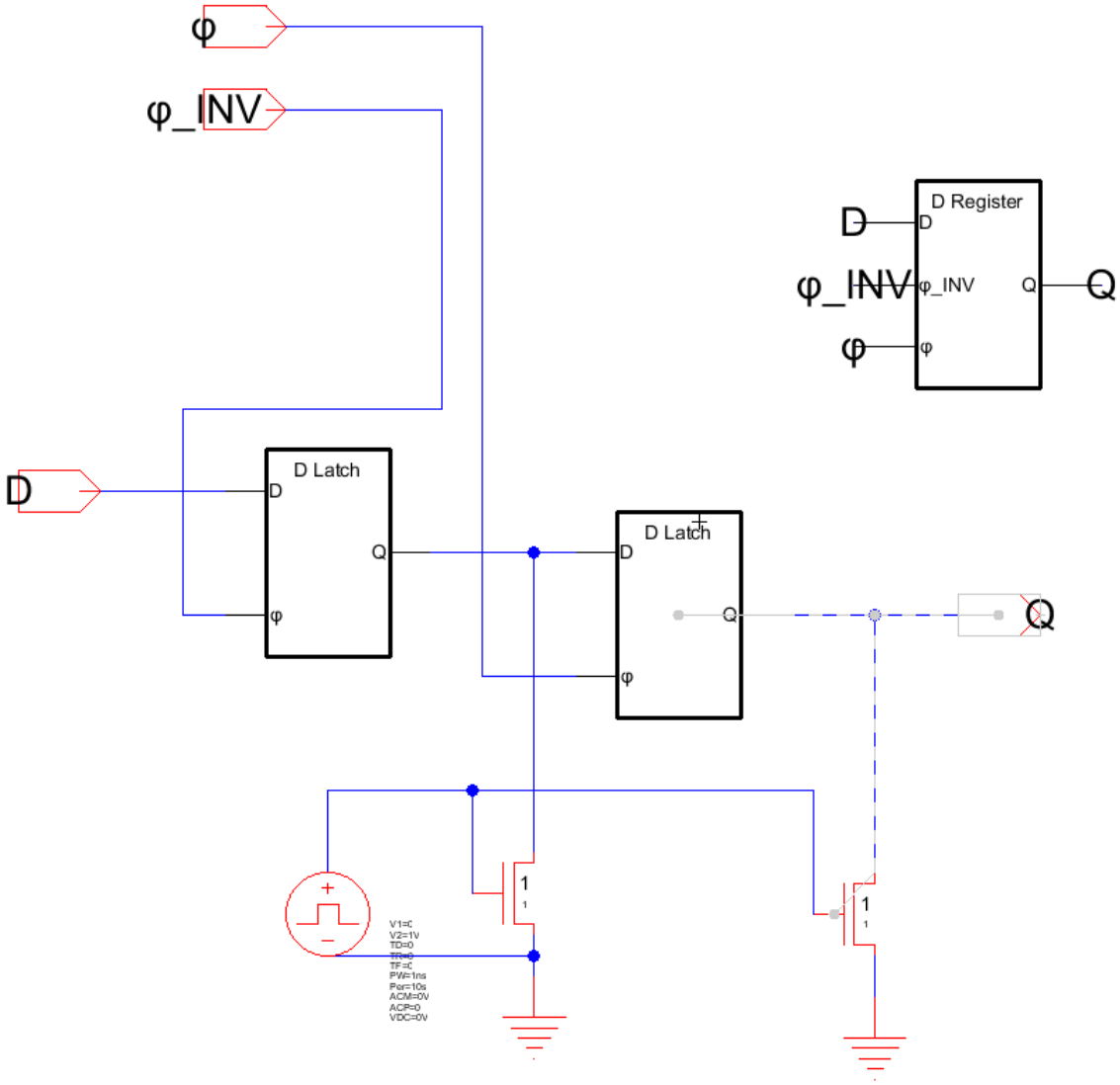

Figure 17: D register schematic

4-bit Register

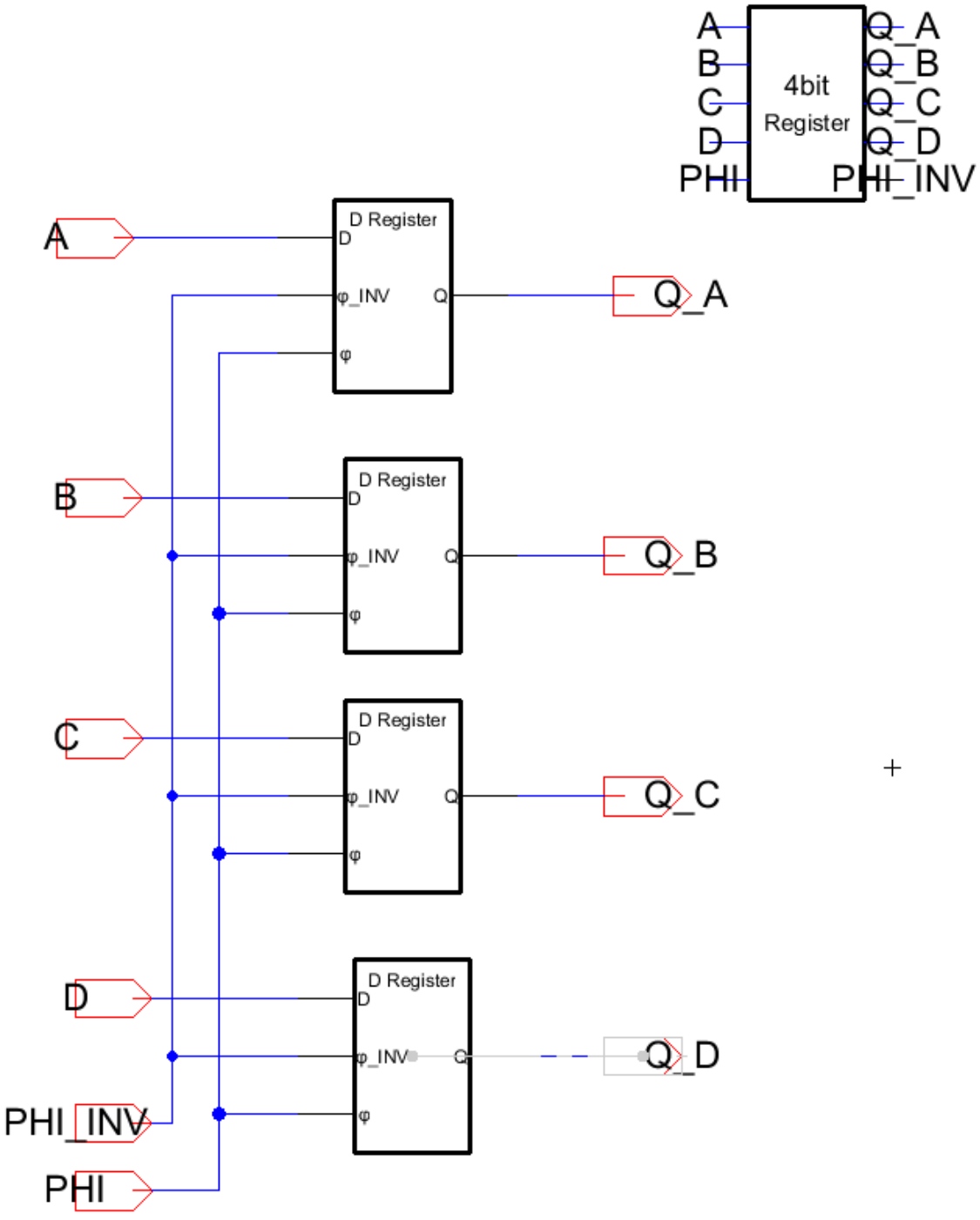

Figure 18: 4-bit register schematic

Incrementer

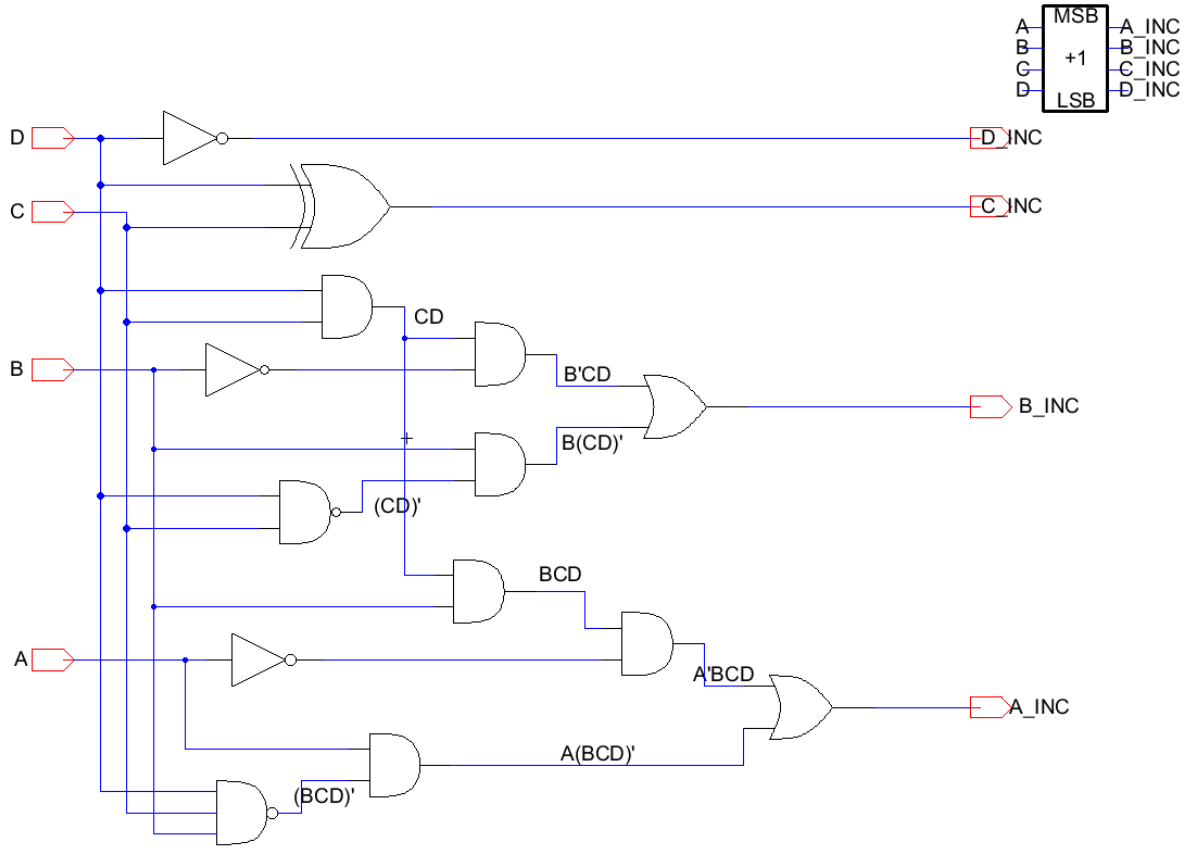

Figure 19: Incrementer schematic

Comparator

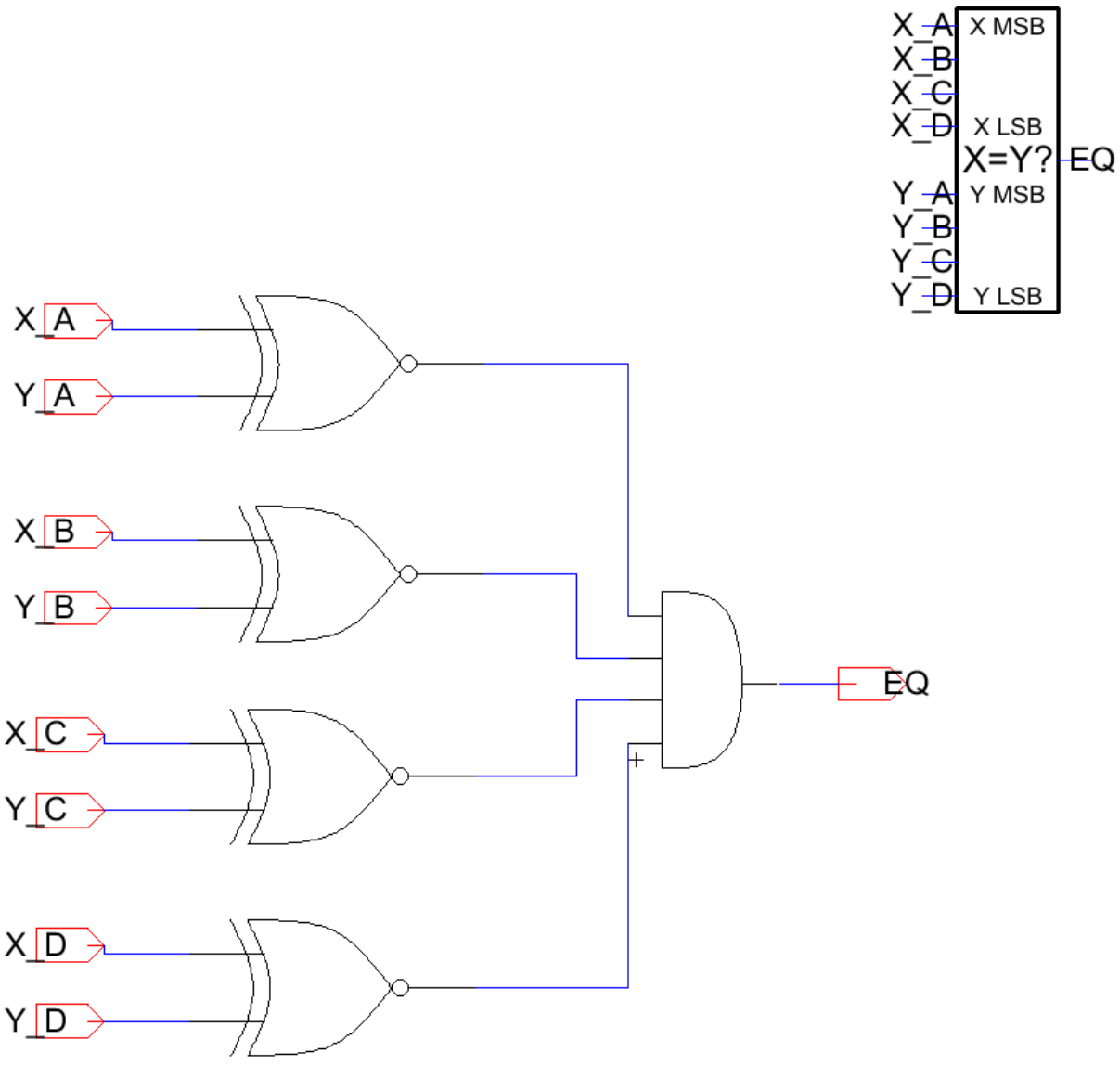

Figure 20: Equality comparator schematic

Mux

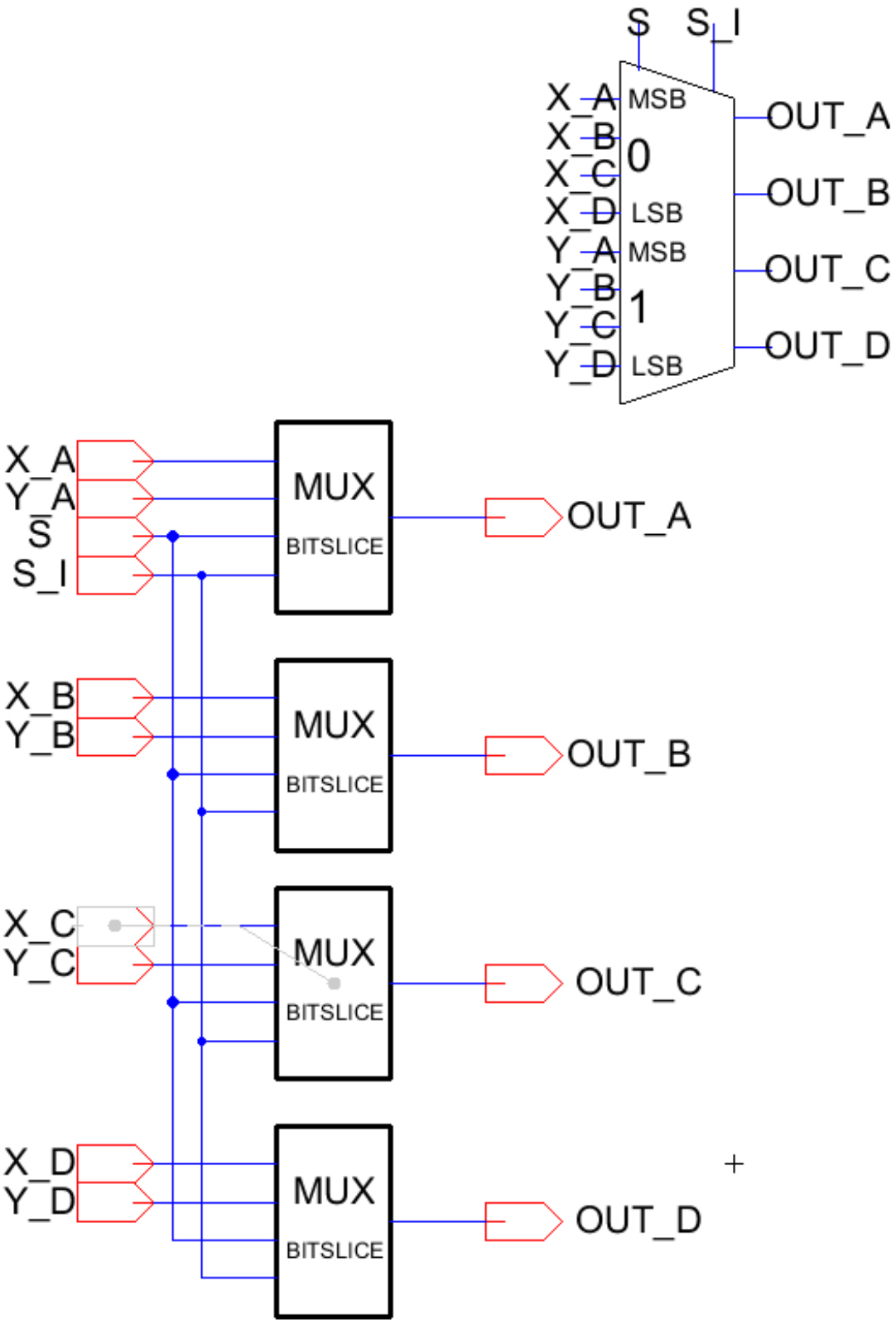

Figure 21: Mux schematic

Mux Bitslice

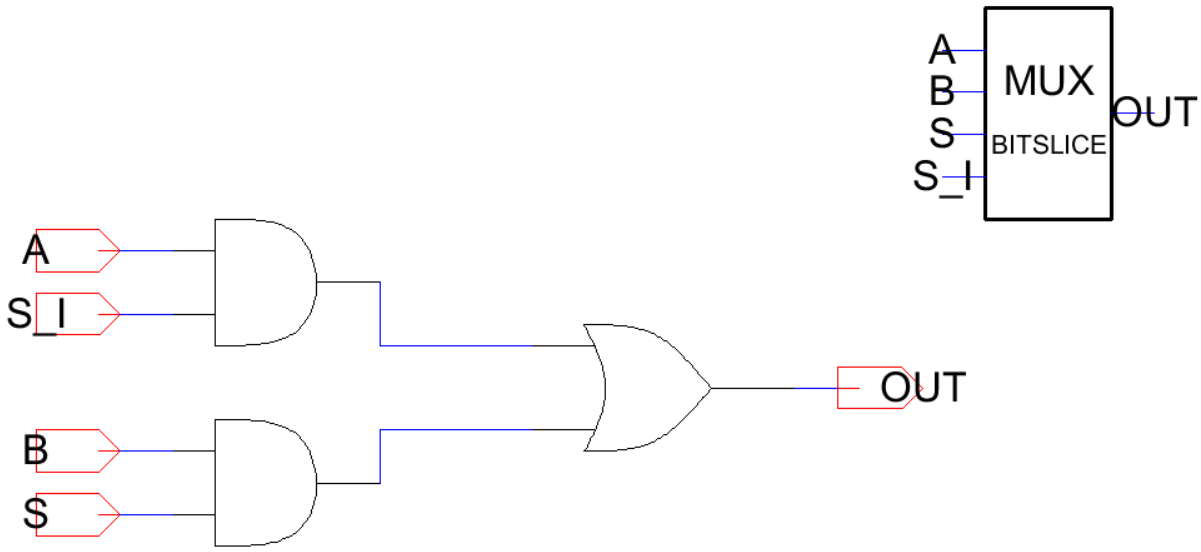

Figure 22: Mux bitslice schematic

Wordline Enable (WL_EN)

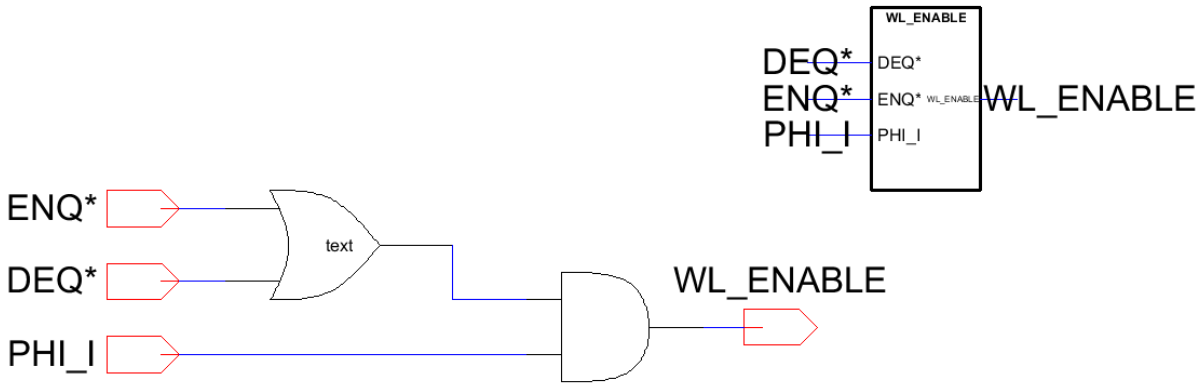

Figure 23: Word line enable generator schematic

ENQ*/DEQ* Generator

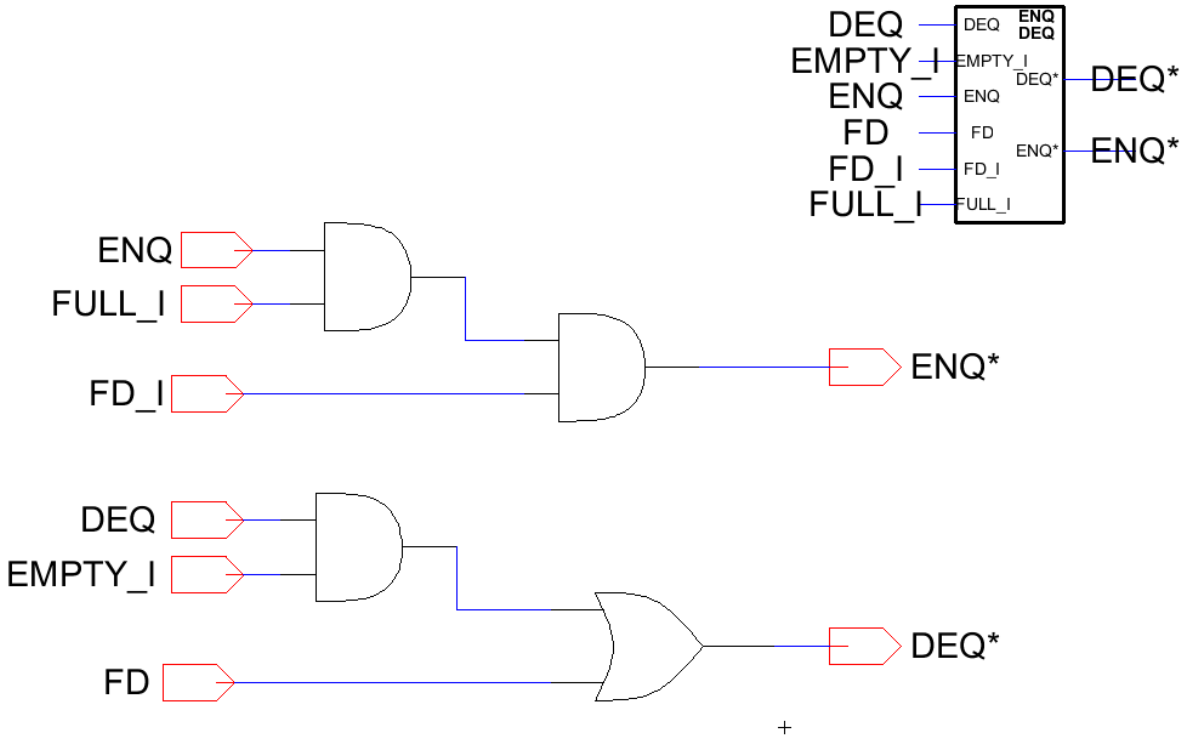

Figure 24: ENQ*/DEQ* generator schematic

4-bit Bitline Precharger

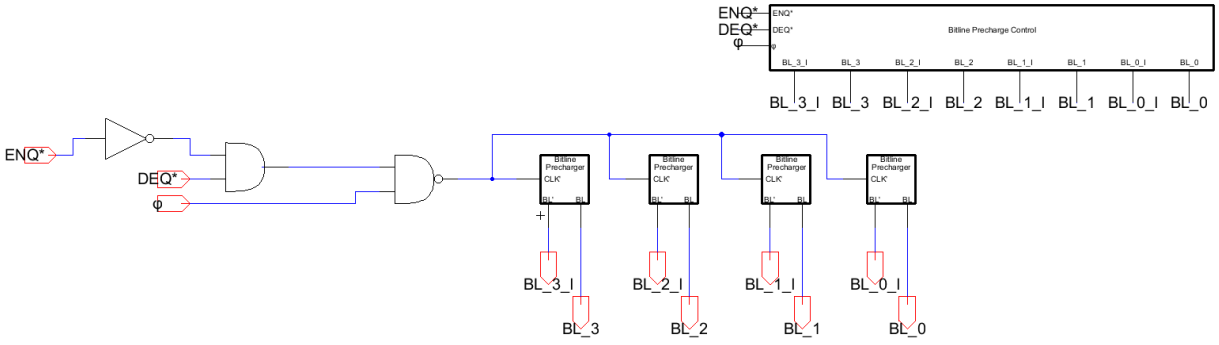

Figure 25: 4-bit bitline precharger schematic

1-bit Bitline Precharger

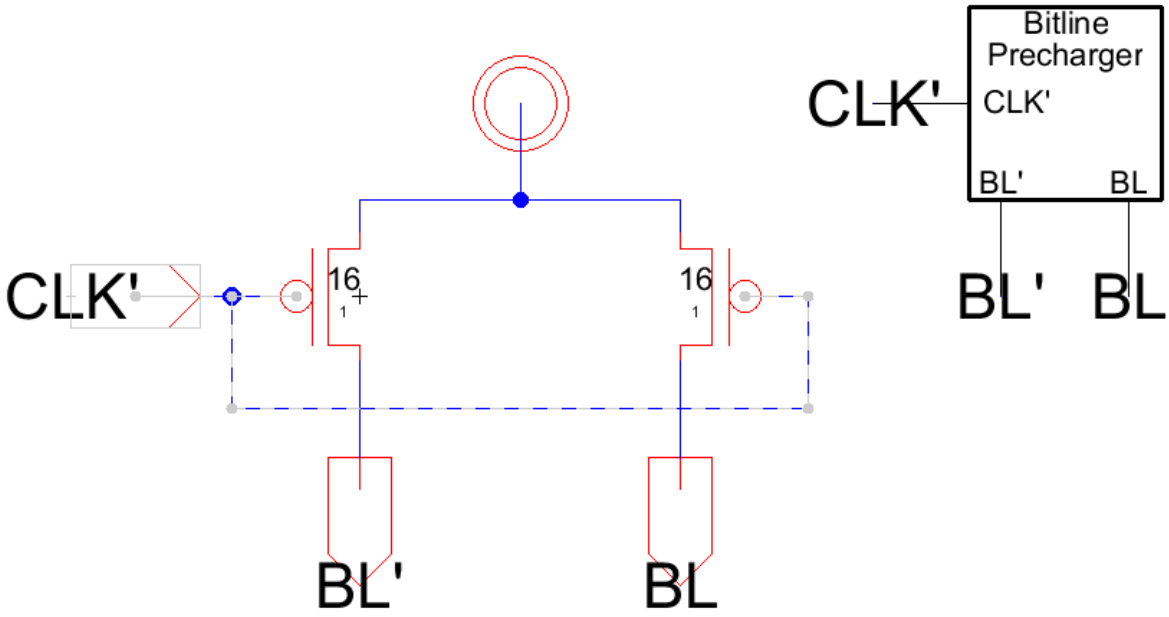

Figure 26: 1-bit bitline precharger schematic

Bitline Driver

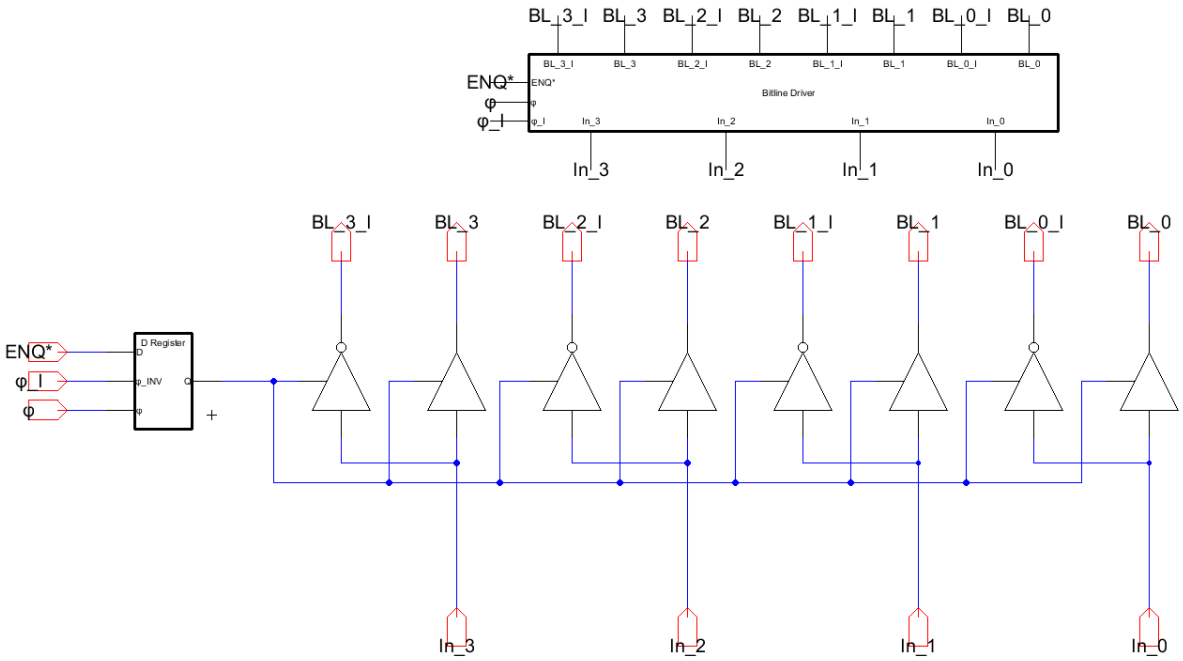

Figure 27: Bitline driver schematic

Inverter

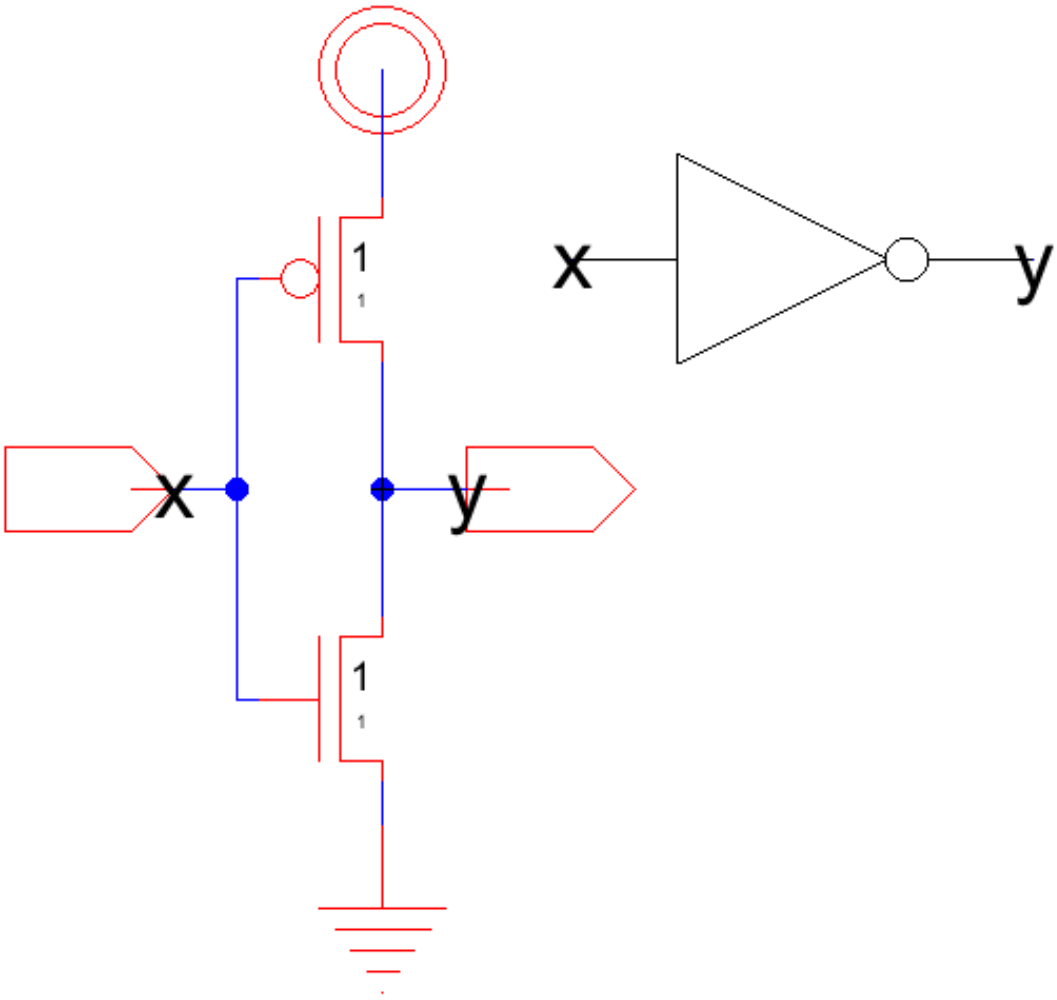

Figure 28: Inverter schematic

NAND2

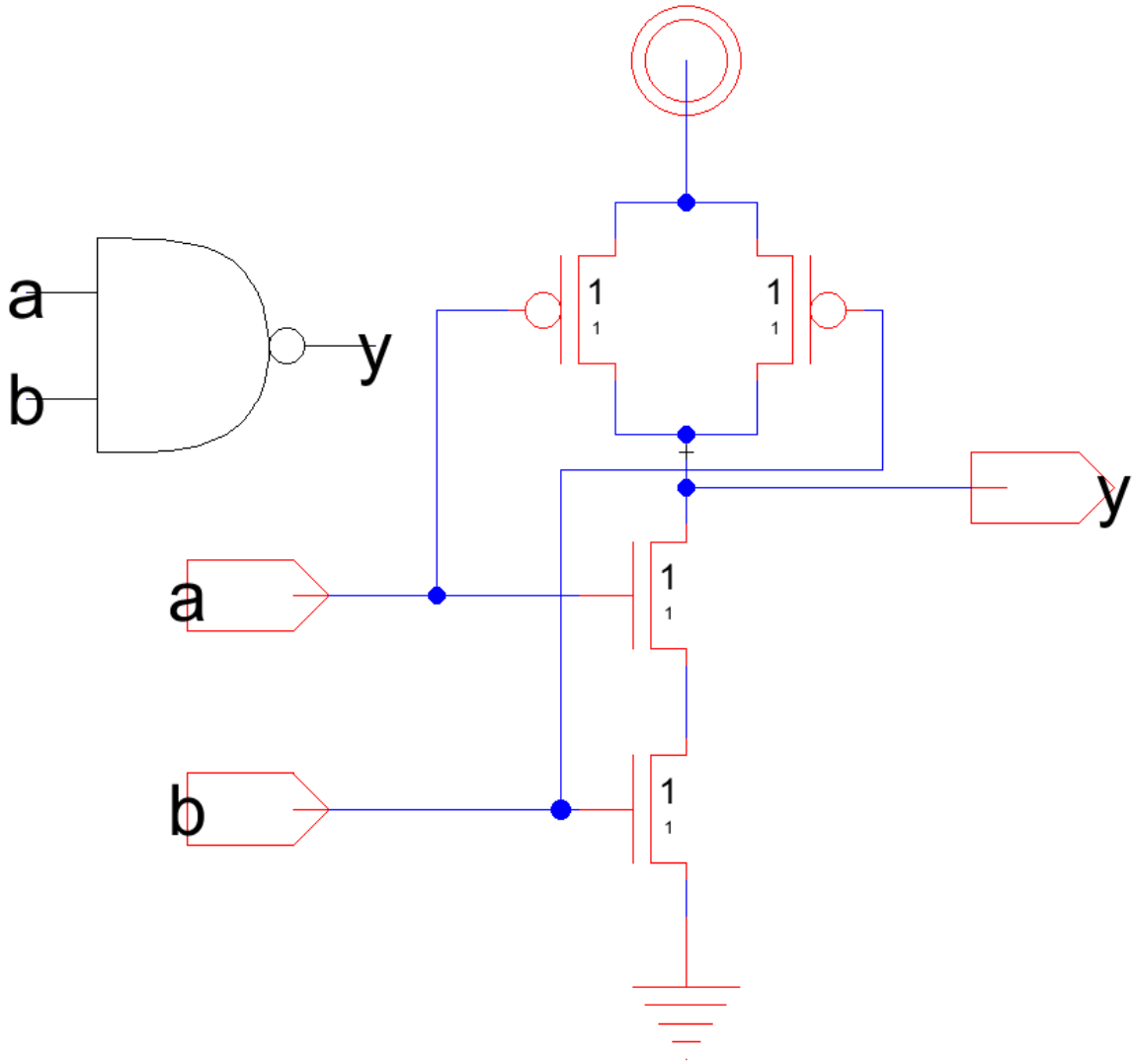

Figure 29: NAND2 gate schematic

AND2

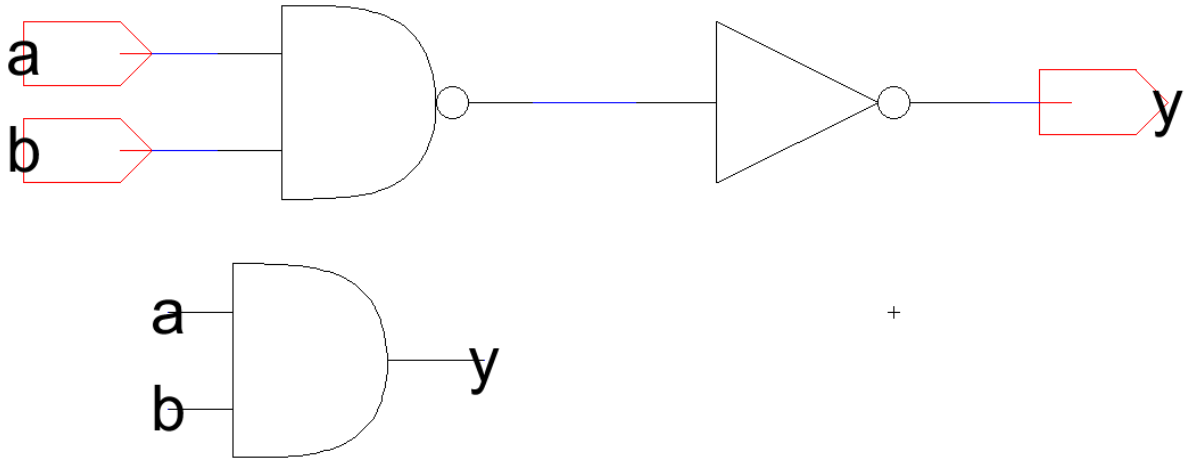

Figure 30: AND2 gate schematic

AND4

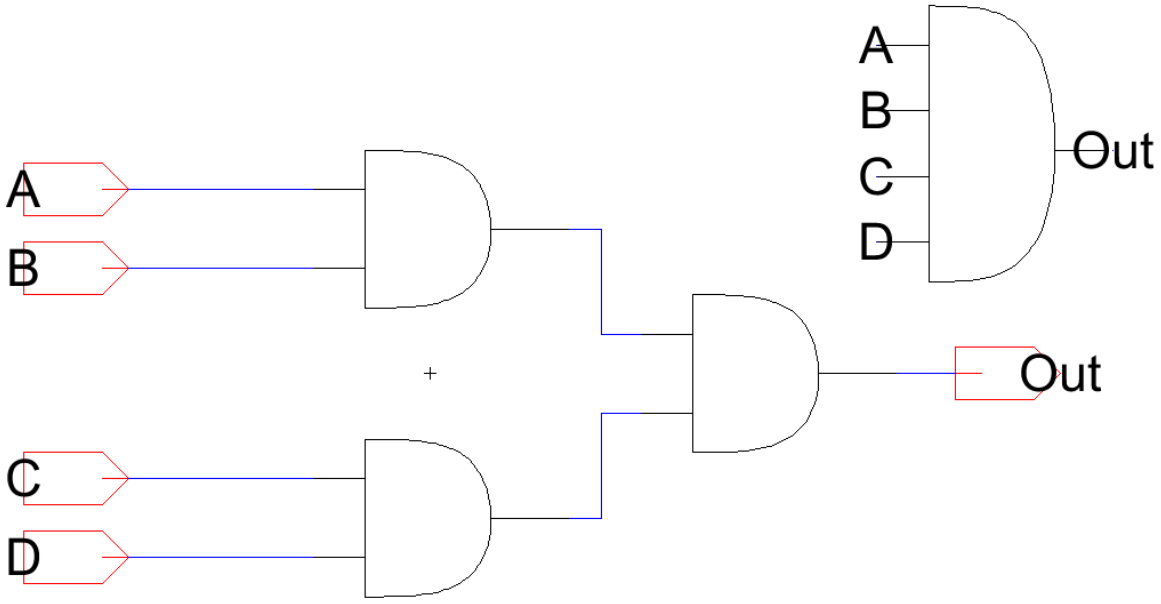

Figure 31: AND4 gate schematic

NOR2

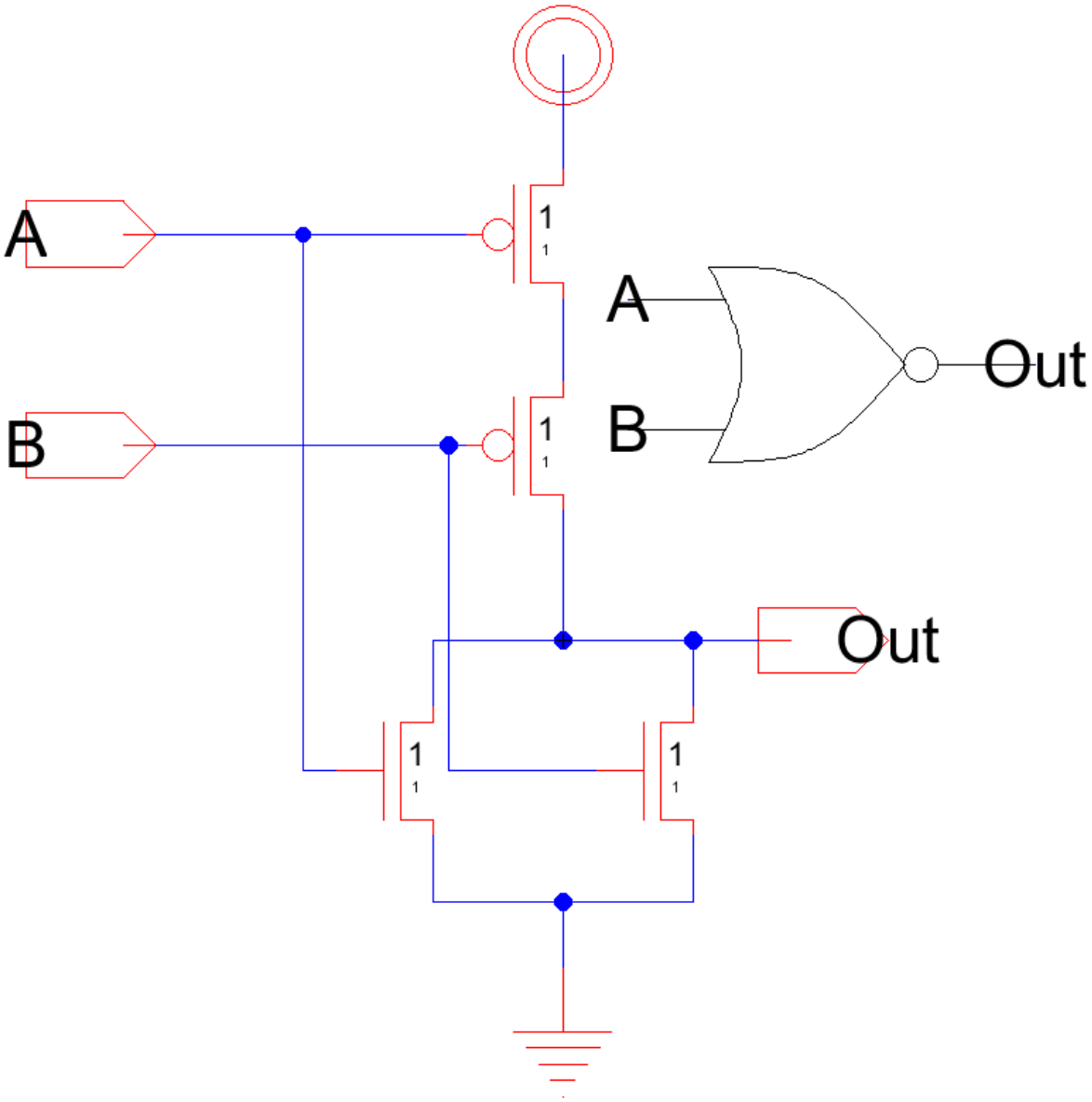

Figure 32: NOR2 gate schematic

Decoder

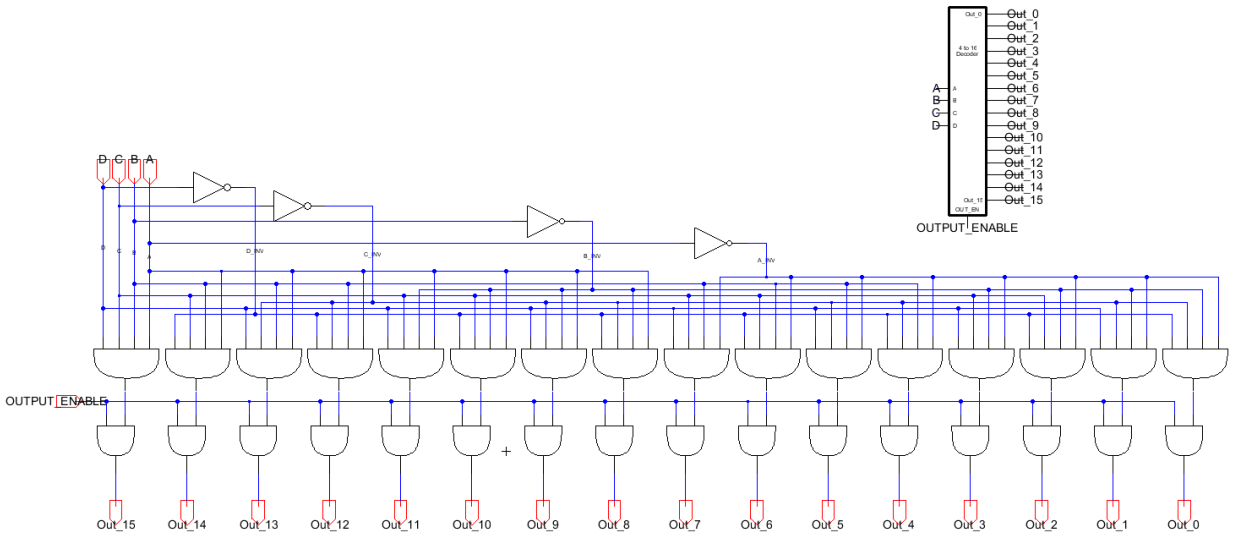

Figure 33: 4-to-16 decoder schematic

Tri-State Buffer

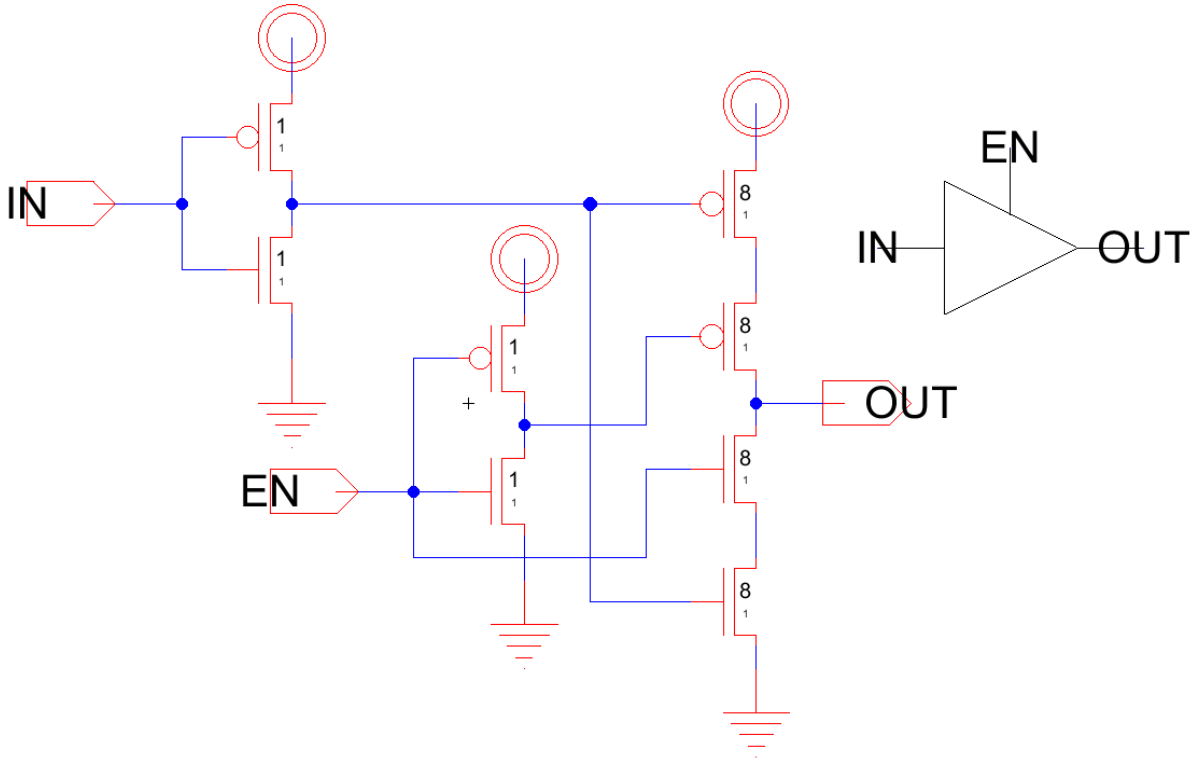

Figure 34: Tri-state buffer schematic

Tri-State Inverter

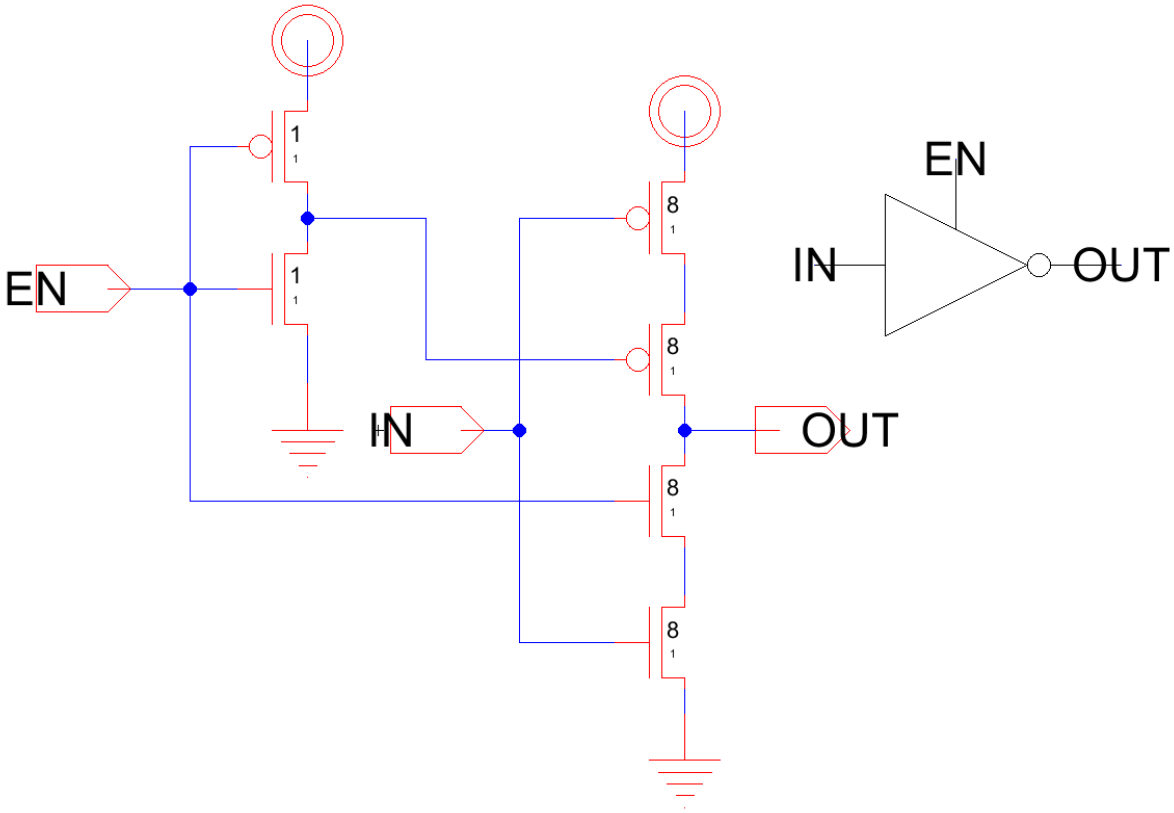

Figure 35: Tri-state inverter schematic

Timing of Key Signals

Memory Operation and Design Choices

Control Block

The control block is broken down as follows;

1. Clock Generator

   The clock generator is used to generate non-overlapping waveforms for the inputs to the D registers. This is needed because these registers consist of two D-latches wired in series. If the two clocks overlapped, the second D latch would sample the input at the same time as the first, meaning that the output woudn't update on the edge of a clock cycle, as it should be.

2. Force Dequeue

   This module is designed to trigger a dequeue (following an enqueue) when both Enqueue and Dequeue are asserted by the user. It consists of a D register that latches the state of (ENQ* and DEQ*) on the falling edge of ϕ, or the rising edge of ϕ'.

3. ENQ*/DEQ* Generator

   This module generates two internal signals, ENQ* and DEQ*, that are used to trigger an enqueue or a dequeue, respectively. ENQ* is set whenever Enqueue is asserted by the user, the queue is not full, and a dequeue is not forced. (It will only trigger when a dequeue is not forced to prevent the user from enqueueing in the clock cycle following a simultaneous enqueue and dequeue, since this clock cycle is when the dequeue will occur.) DEQ* is set whenever the user asserts Dequeue and the queue is not empty, or when a dequeue from a simultaneous enqueue/dequeue event needs to be completed.

   The ENQ* and DEQ* outputs from this module are fed into registers to latch the values until the next clock cycle. The user therefore does not have to assert Enqueue and Dequeue for the entire clock cycle.

4. WL Enable

   Whenever ENQ* or DEQ* are high, the WL Enable module enables and disables the decoder output in the memory cell, which determines which word line is being set high. (See Decoder for more information on why an enable input was necessary.)

5. Pointers

   The Pointers module holds the state for the head and tail pointers and contains the logic for updating these pointers and determining if the queue is empty or full. It is broken down as follows:

   (a) Head (HEAD) and Tail (TAIL) Pointers

These contain the addresses of where the queue will enqueue to/dequeue from the next time Enqueue or Dequeue are triggered. In our queue, enqueues add elements to the tail, and dequeues remove elements from the head. Therefore, the address of tail is always greater than the address of head.

HEAD is incremented whenever DEQ* is high, ENQ* is not high, and a falling edge of the clock occurs. Likewise, TAIL is incremented whenever ENQ* is high, DEQ* is not high, and a falling edge of the clock occurs.

(b) Head and Tail Incrementers (HEAD_PLUS_1 and TAIL_PLUS_1)

These are combinational circuits that increment the current values of HEAD and TAIL. They are used to increment the current values of HEAD and TAIL on enqueues and dequeues, respectively.

The incremented values are followed by registers that toggle on the rising edge of the clock in order to prevent these incremented values from arriving too early at HEAD/TAIL and updating HEAD/TAIL prematurely. Without them, we ran into an issue where HEAD and TAIL would sometimes ”jump” to higher values, even though they were supposed to rise by 1.

(c) Comparator (HEAD = TAIL)

This comparator is a combinational circuit that checks whether the values of HEAD and TAIL are equal to each other. It outputs whether the queue is full (FULL) or empty (EMPTY) depending on whether OVF has been set by OVF_REG (see below).

(d) OVF_REG

This portion of the circuit checks when the current values of HEAD and TAIL are equal to 15, signalling that they are about to wrap-around to address 0. When this happens, OVF_REG is toggled. (Observe that OVF_REG is always set high when TAIL wraps around and is always set low when HEAD wraps around. This is because OVF_REG starts at 0, TAIL wraps around first (it's ahead of HEAD), and TAIL wrapping around is always followed by HEAD wrapping around (and vice-versa).)

Also observe that when OVF is set and HEAD == TAIL, then the queue is full since TAIL has wrapped around and HEAD has not yet. Likewise, when OVF is not set and HEAD == TAIL, the queue is empty since HEAD has wrapped around and TAIL has not yet. (See above for more information on the Comparator.)

The HEAD and TAIL outputs of Pointers are fed into two 4-bit registers that latch on the rising edge of the clock, and the outputs of these registers are muxed to the address bits of the SRAM memory block. The outputs are latched to prevent the address bits from updating before an enqueue or dequeue is performed. They are muxed to update the address bits according to whether an enqueue or dequeue is

performed. (For example, an enqueue will update the address lines to TAIL, while a dequeue will update them to HEAD.)

6. Bitline Precharge Control

   This module triggers a precharge of the bitlines whenever ENQ* is low and DEQ* is high, and the clock is high. (The NAND gate is present instead of an AND because the precharge enable is active-low.)

   Each bit-level precharge circuit consists of two PMOS transistors sized at $W = 16$. We sized them like this to overcome the additional capacitance presented on the bitline by the 16 SRAM memory cells.

7. Bitline Driver

   The bitline driver is used during enqueuing to write to the memory cells. Writes are triggered whenever ENQ* is asserted and on the rising edge of the clock cycle, and last for a full clock cycle.

Memory Block

The memory block is broken down as follows:

1. 6T SRAM Cells

   There are 64 SRAM cells, each holding a single bit (16 words x 4 bits per word). These cells consist of four NMOS and two PMOS transistors arranged in a cross-coupled inverter configuration. (This is the same configuration presented in lecture).

   The two access transistors are sized at $W = 8$ in order to prevent the cell from being overwritten when the bitline changes suddenly. We originally had them at $W = 4$ when we tested the milestone, but when we tested the entire array, we found that when dequeueing (which involves precharging the bitlines), some cells were being overwritten.

   The bottom two NMOS transistors are sized at $W = 16$ to prevent read upsets - the value of Q being overwritten when the cell is being read. Originally, for the milestone, they were set at $W = 8$, but we doubled the width to keep the ratio between this W and $Waccess$ the same after we changed $Waccess$ to 8, as described above.

2. Decoder

   This 4-to-16 decoder is used to reduce the amount of address lines needed from the control block. It is a standard CMOS decoder design that outputs the 16 possible minterms generated from 4 logic inputs (A, B, C, and D). It also includes an output enable bit to turn off the decoder's output when we are not reading or writing, therefore ensuring that the word line of one bit is not high at once. (If one word

line was high at all times, we would be reading or writing at all times, which would interfere with reading or writing at the intended time because the bit lines would need to be precharged or driven.)

Breakdown of Energy Contributions

Optimization of Energy

Validation of Correctness

We validated the correctness of the design by unit-testing each module, and then running integration tests on the whole queue.

Unit Tests

Clock Generator

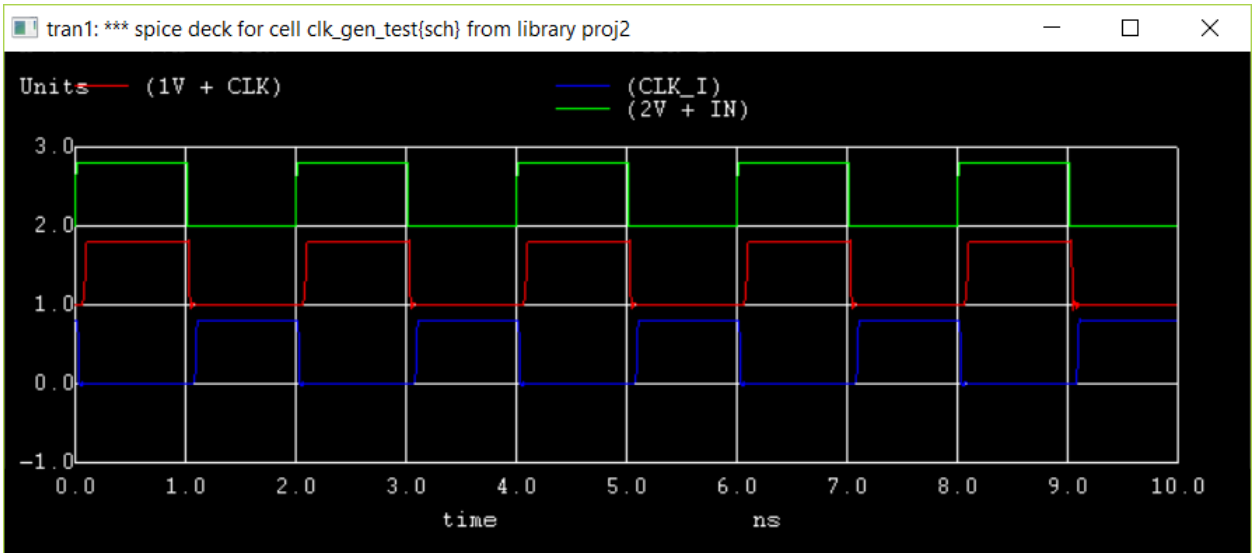

Figure 36: Clock generator test

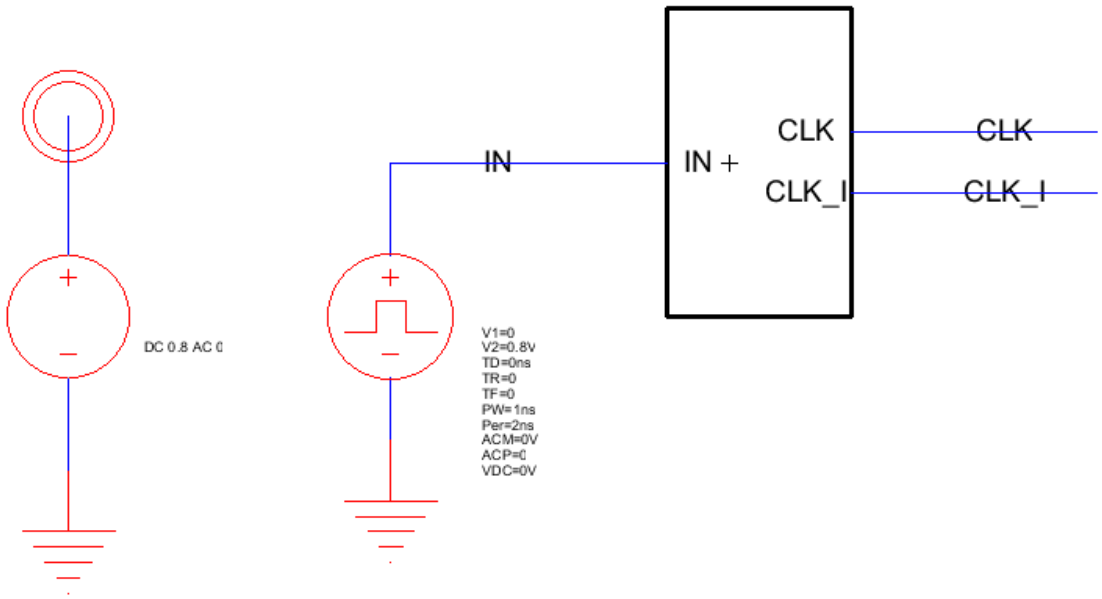

Figure 37: Clock generator test schematic

Comparator

The comparator was broken into 3 tests: numbers the same, numbers were different but swapped, and numbers were different.

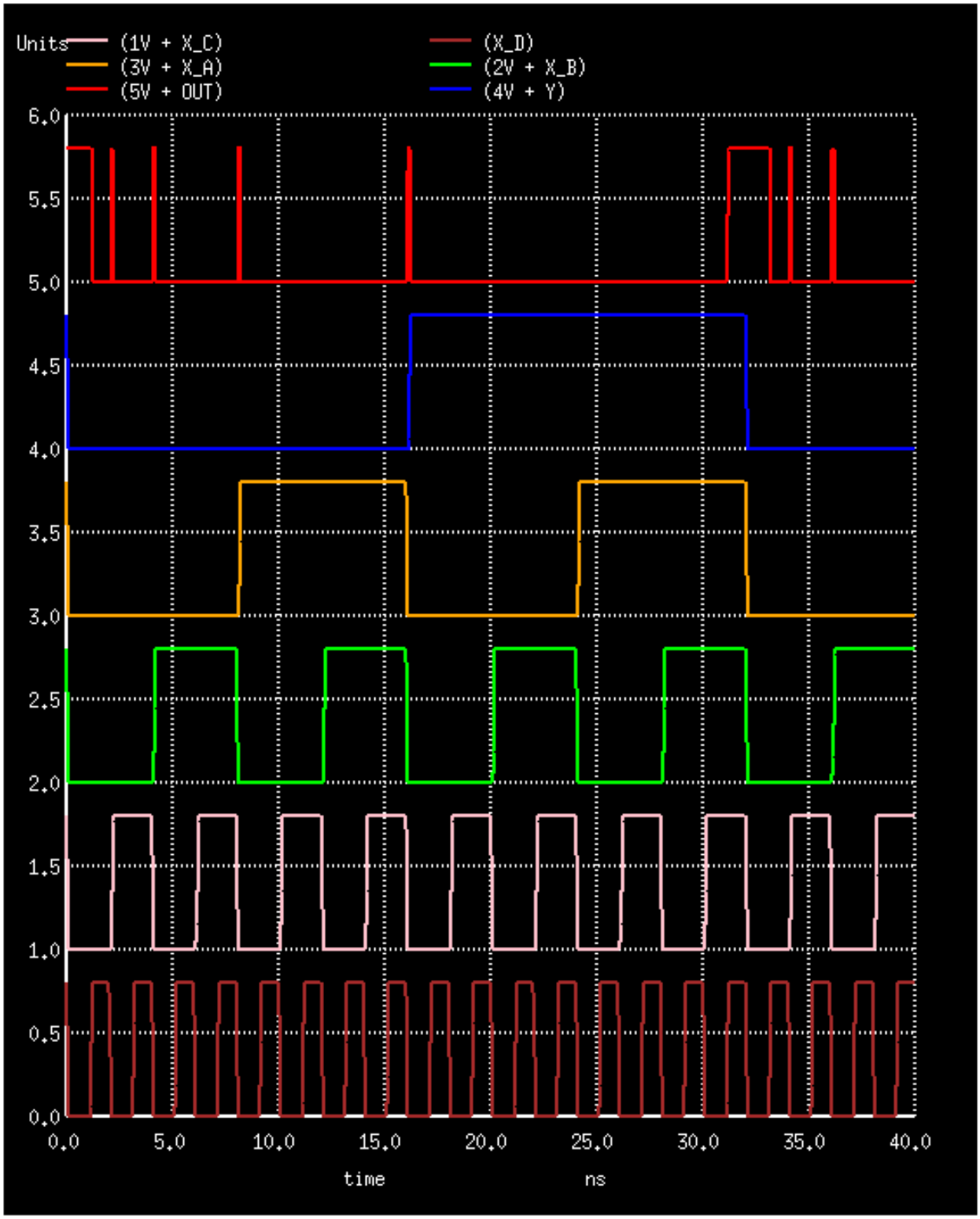

Figure 38: Comparator test, different

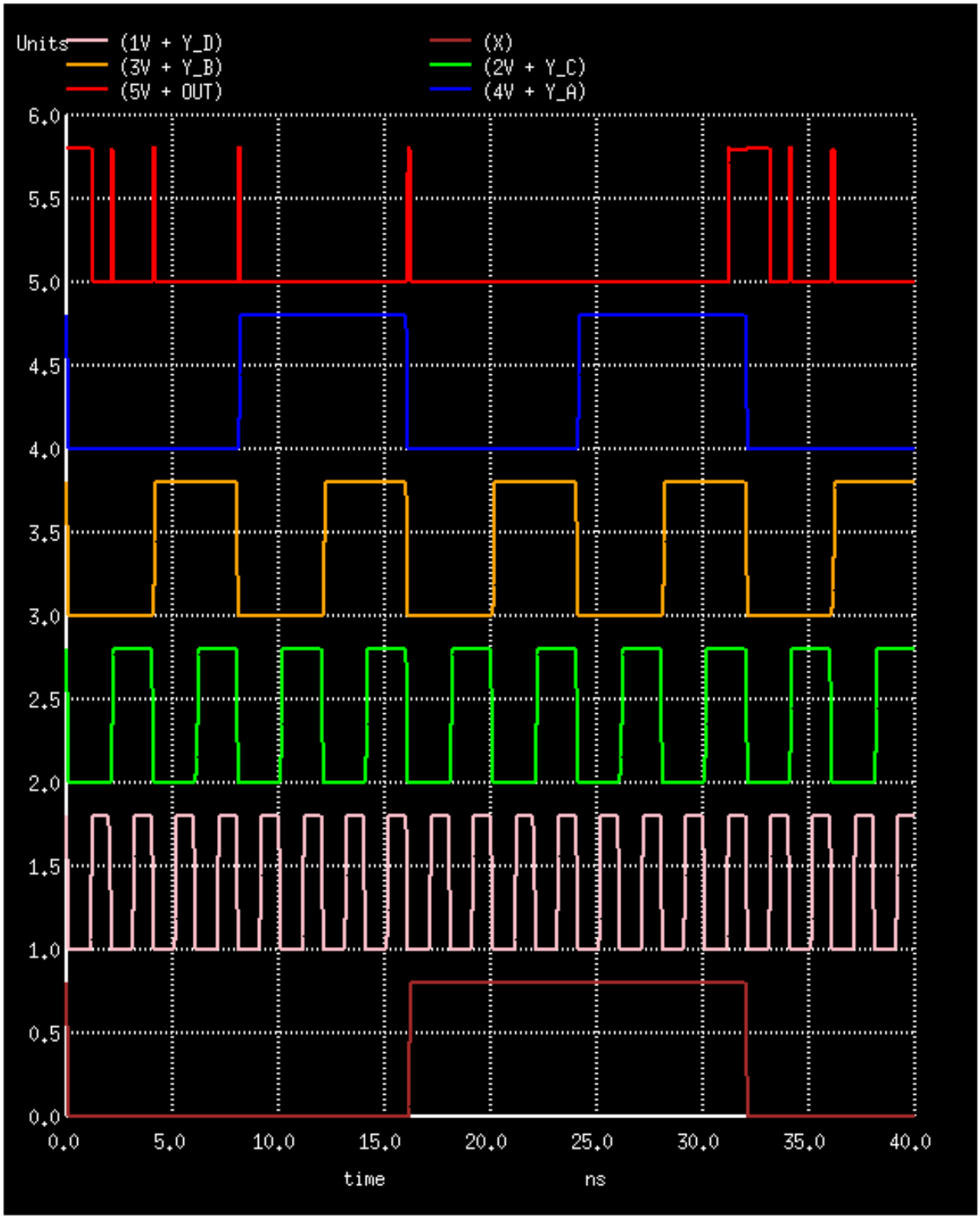

Figure 39: Comparator test, different and swapped

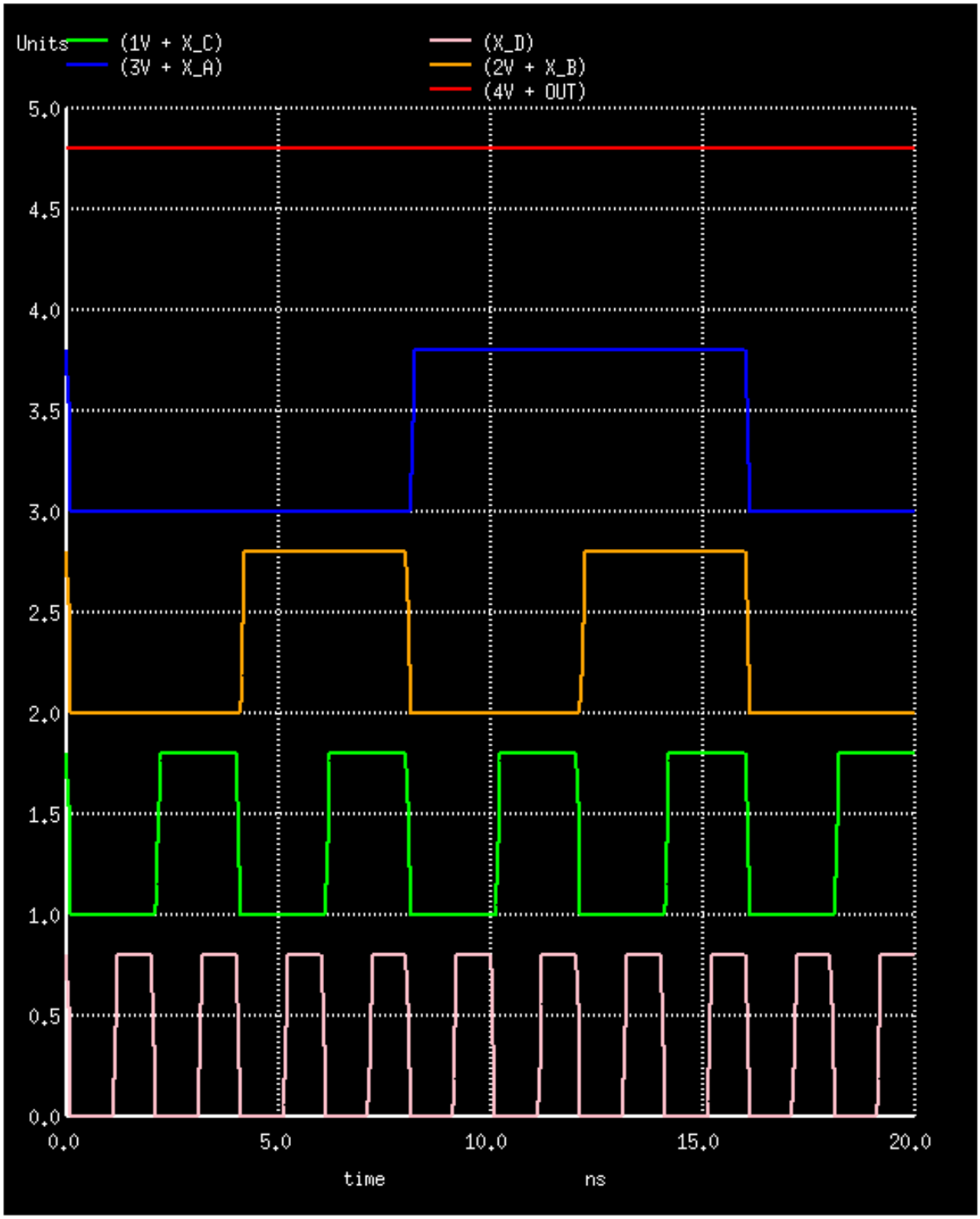

Figure 40: Comparator test, same

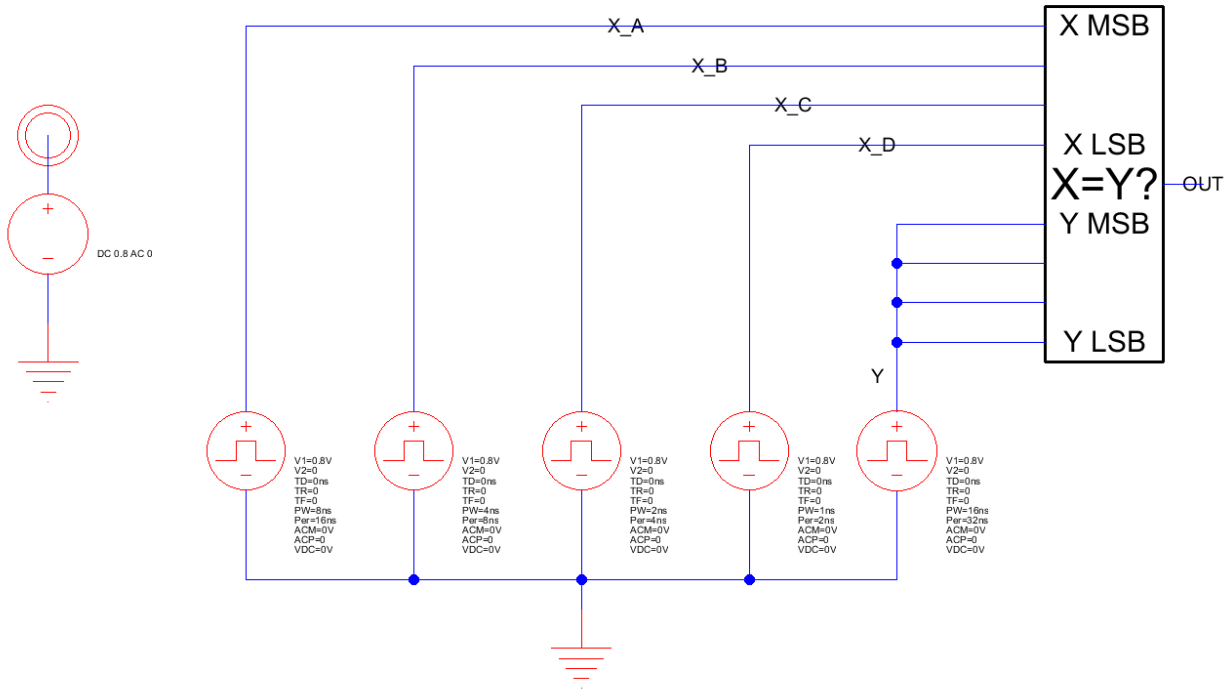

Figure 41: Comparator test schematic

D Latch

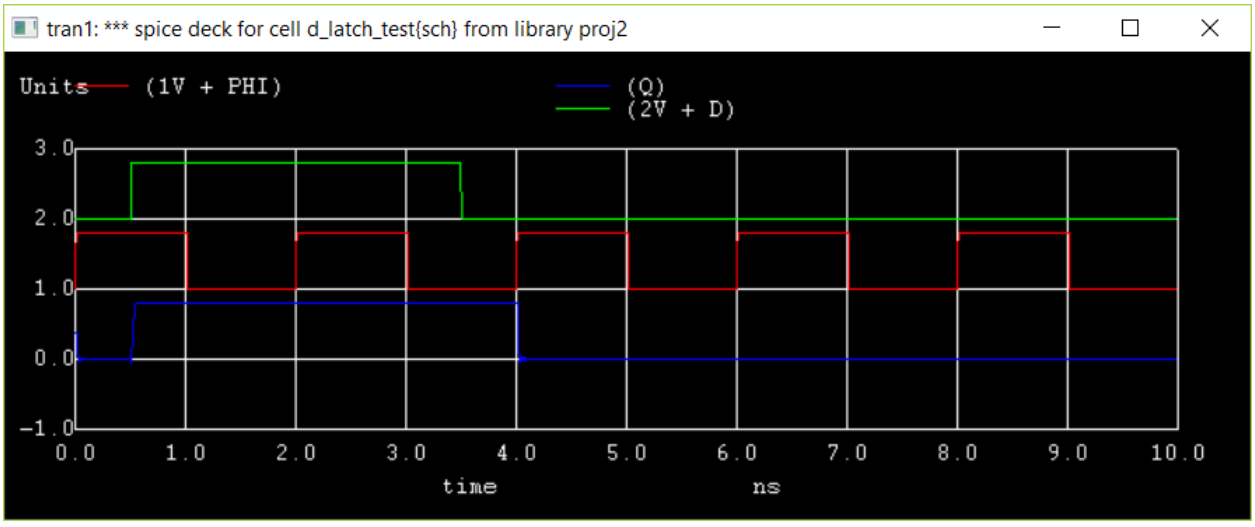

Figure 42: D Latch Test

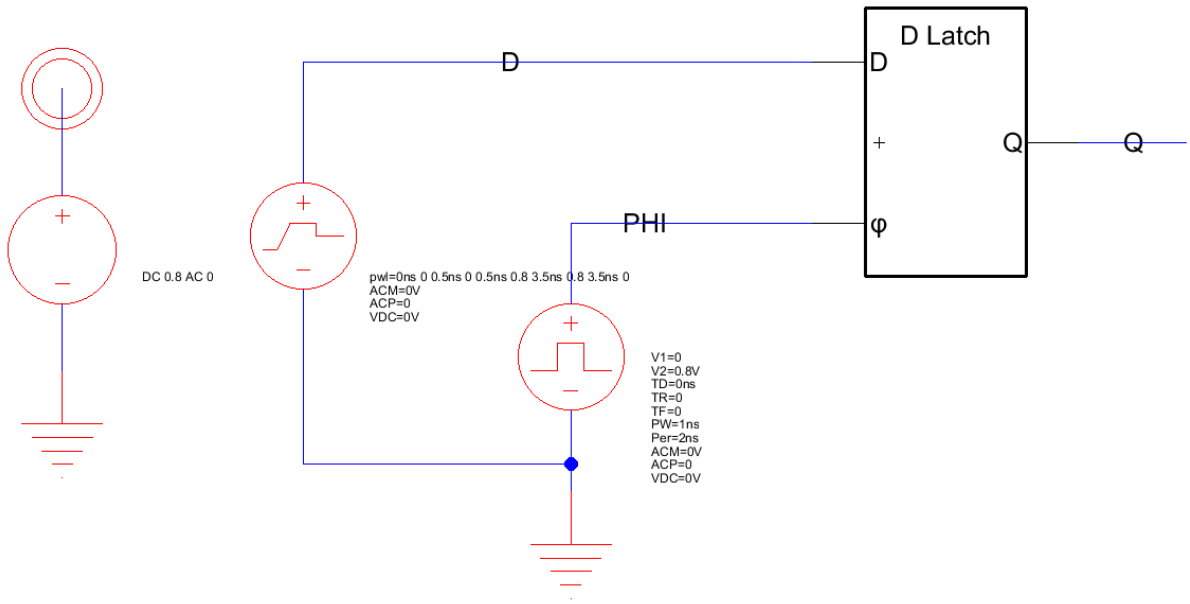

Figure 43: D latch test schematic

D Register

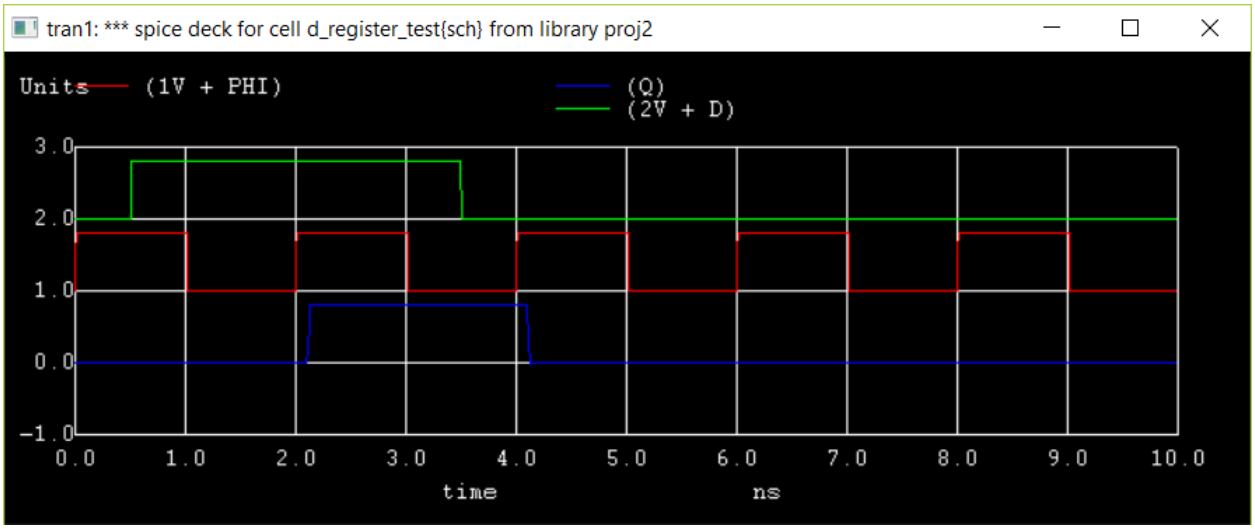

Figure 44: D Register Test

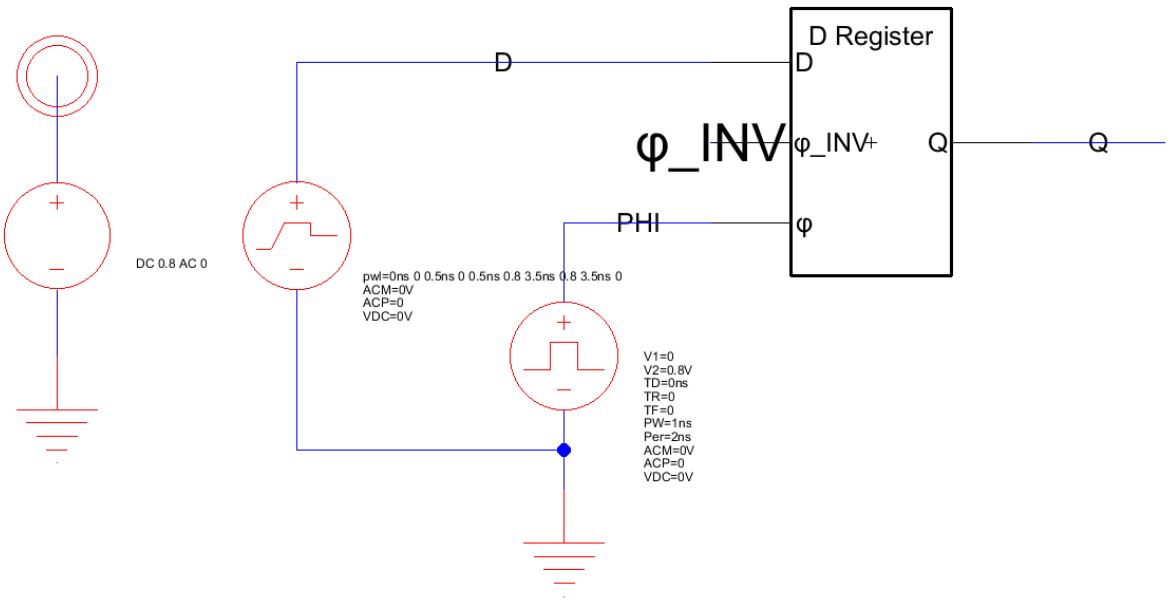

Figure 45: D register test schematic

ENQ*/DEQ* Module

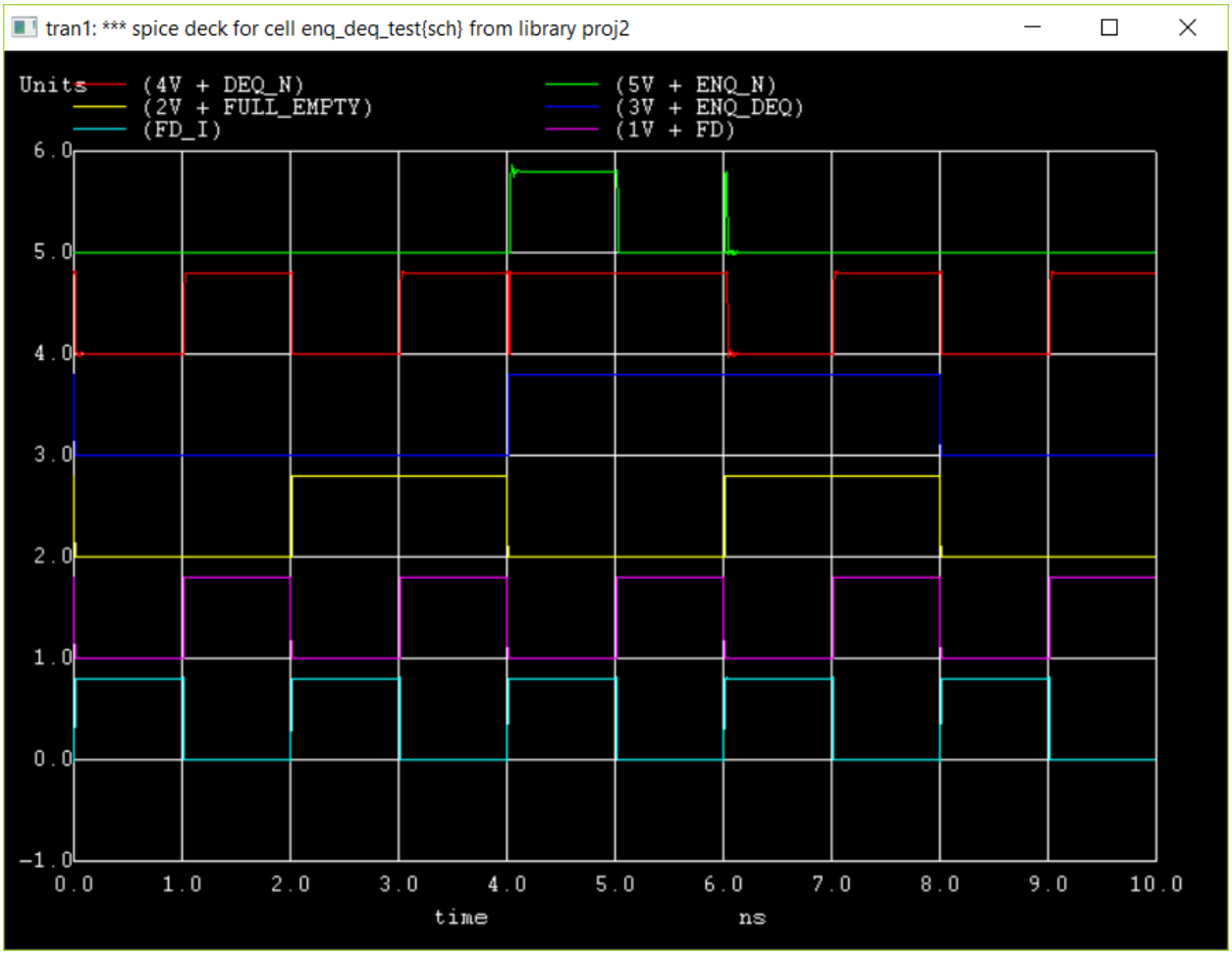

Figure 46: ENQ*/DEQ* Module Test

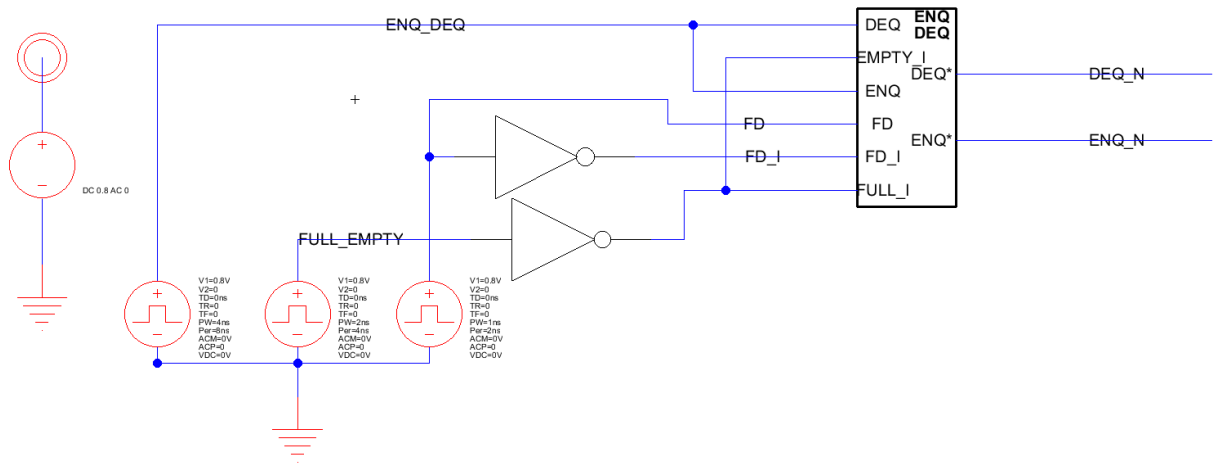

Figure 47: ENQ*/DEQ* test schematic

Incrementer

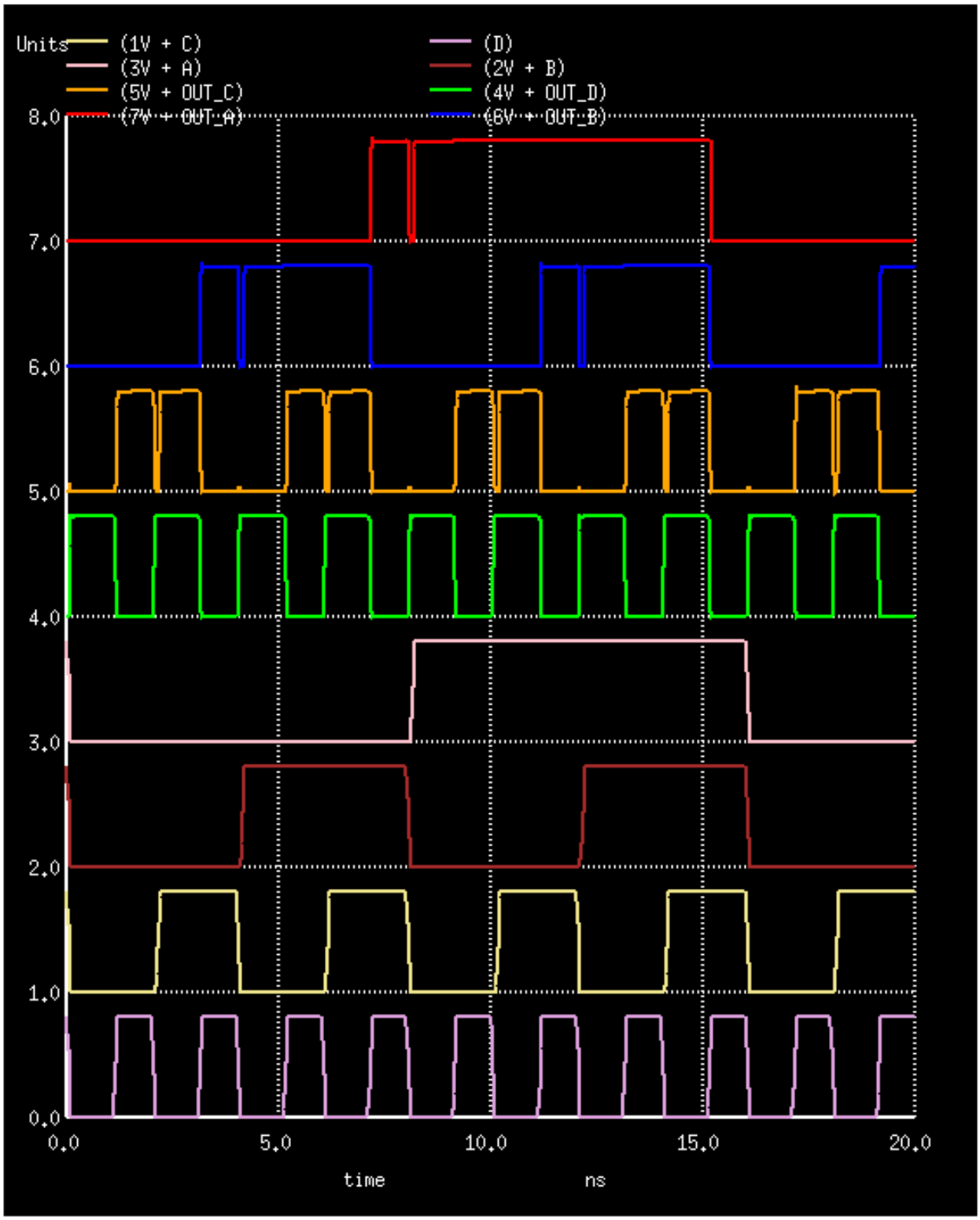

Figure 48: Incrementer Test

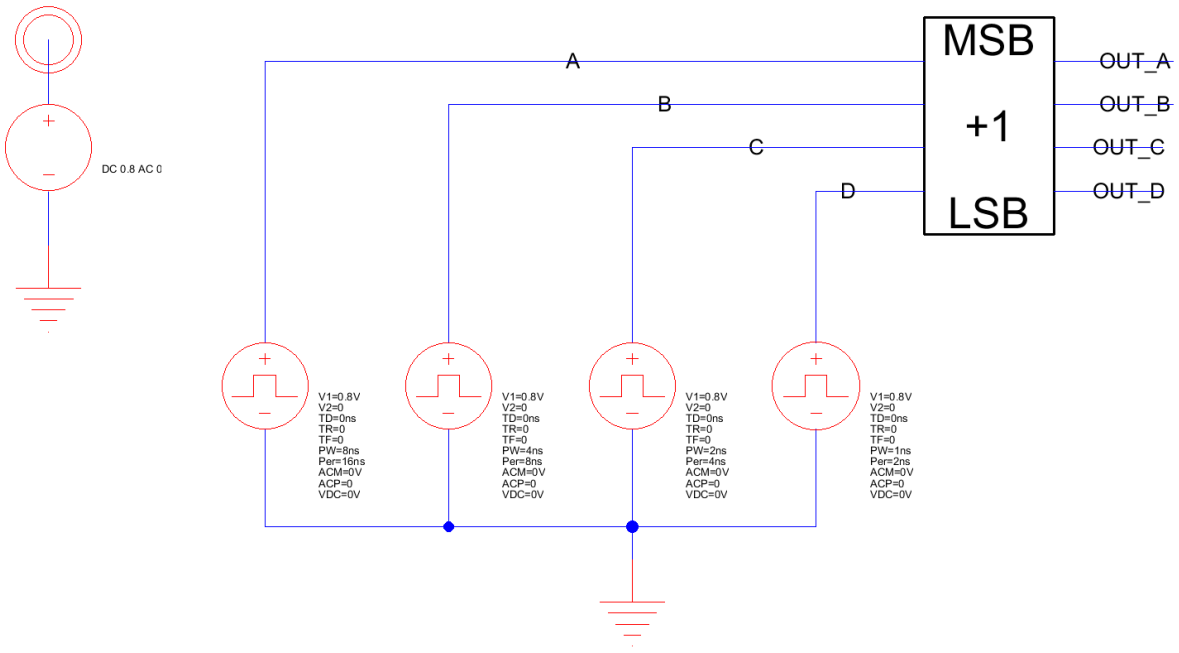

Figure 49: Incrementer test schematic

Pointers

The pointers test is broken down into testing for proper incrementing, testing for no incrementing when not enqueuing or dequeuing, and testing that the empty and full lines are asserted at the appropriate times (and not asserted when the queue is neither empty nor full (head != tail)).

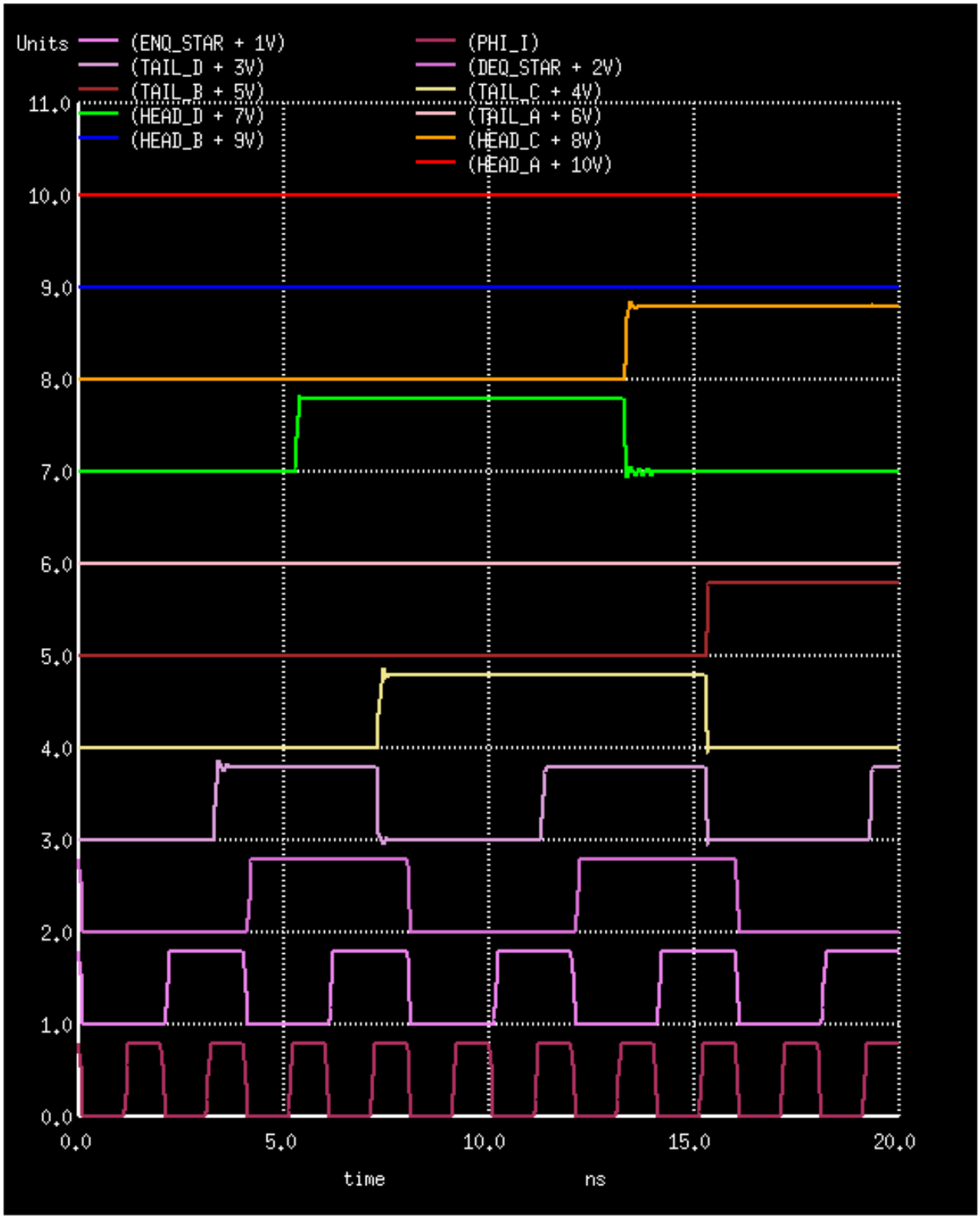

Figure 50: Pointers test, proper increment

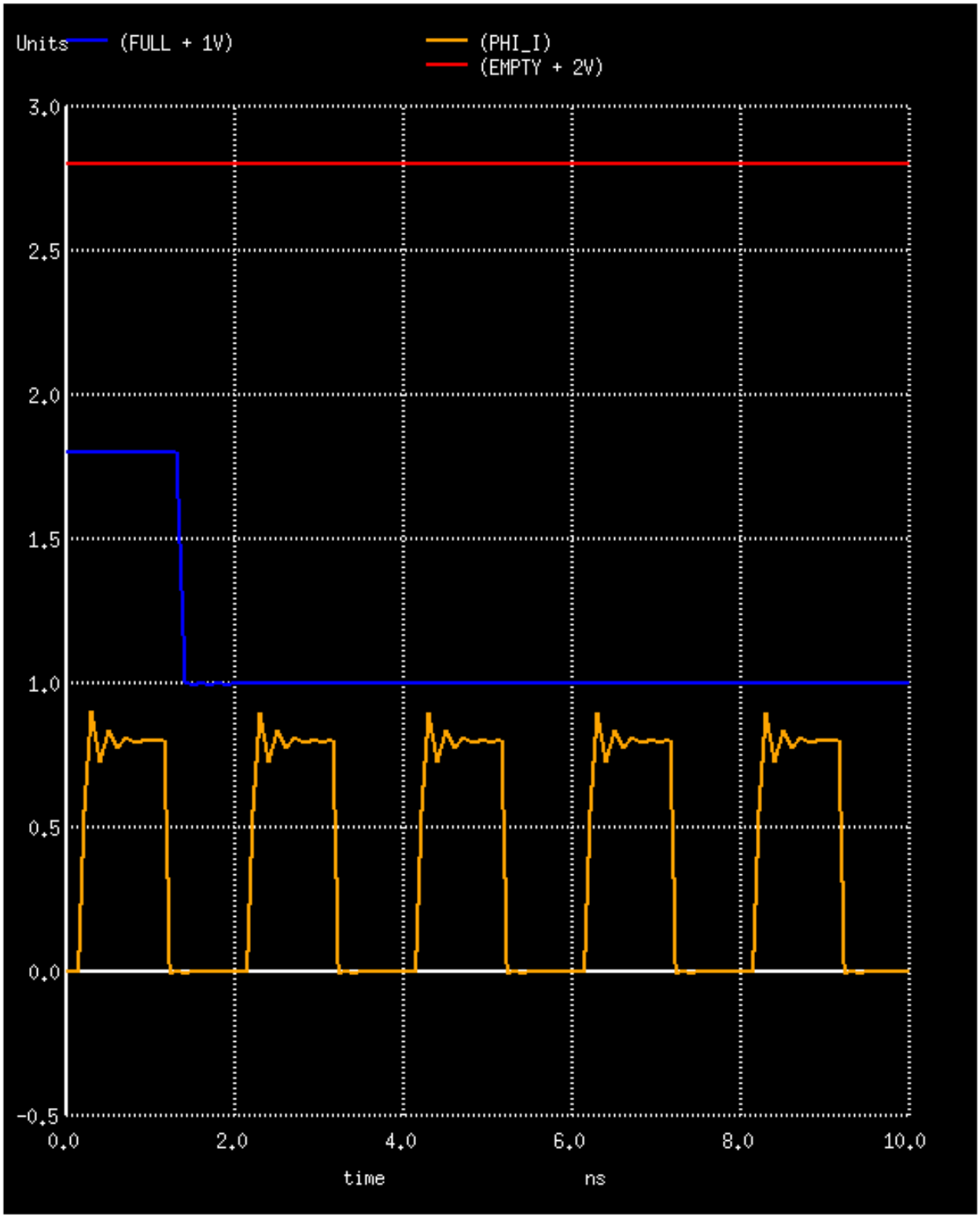

Figure 51: Pointers test, no incrementing

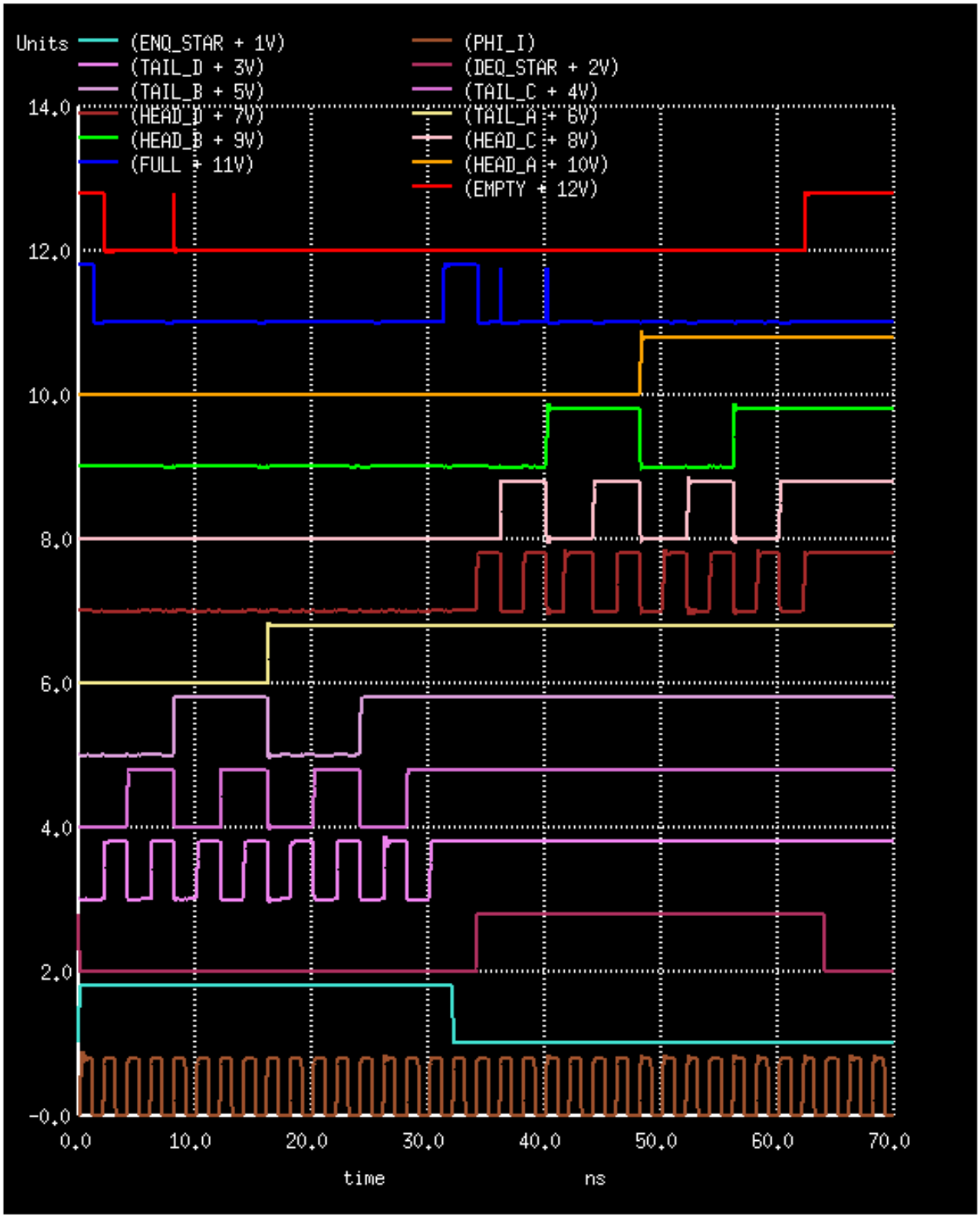

Figure 52: Pointers test, empty and full

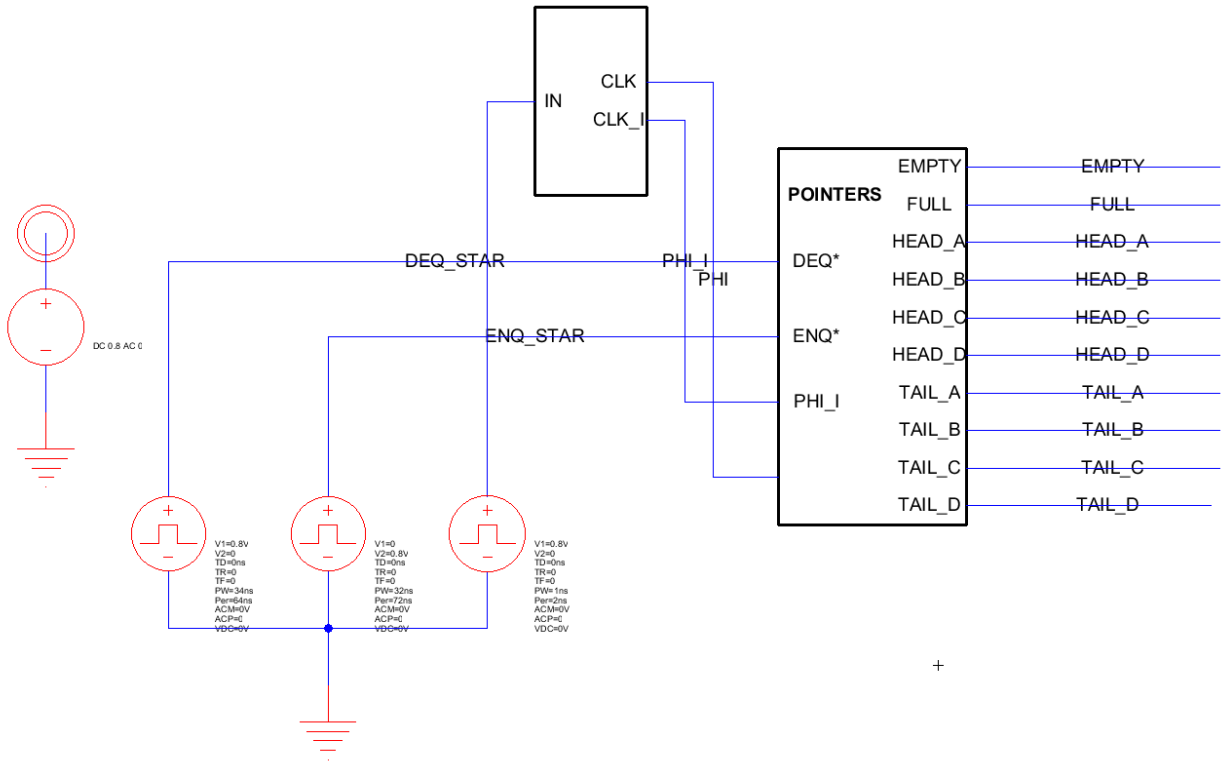

Figure 53: Pointers test schematic

Decoder

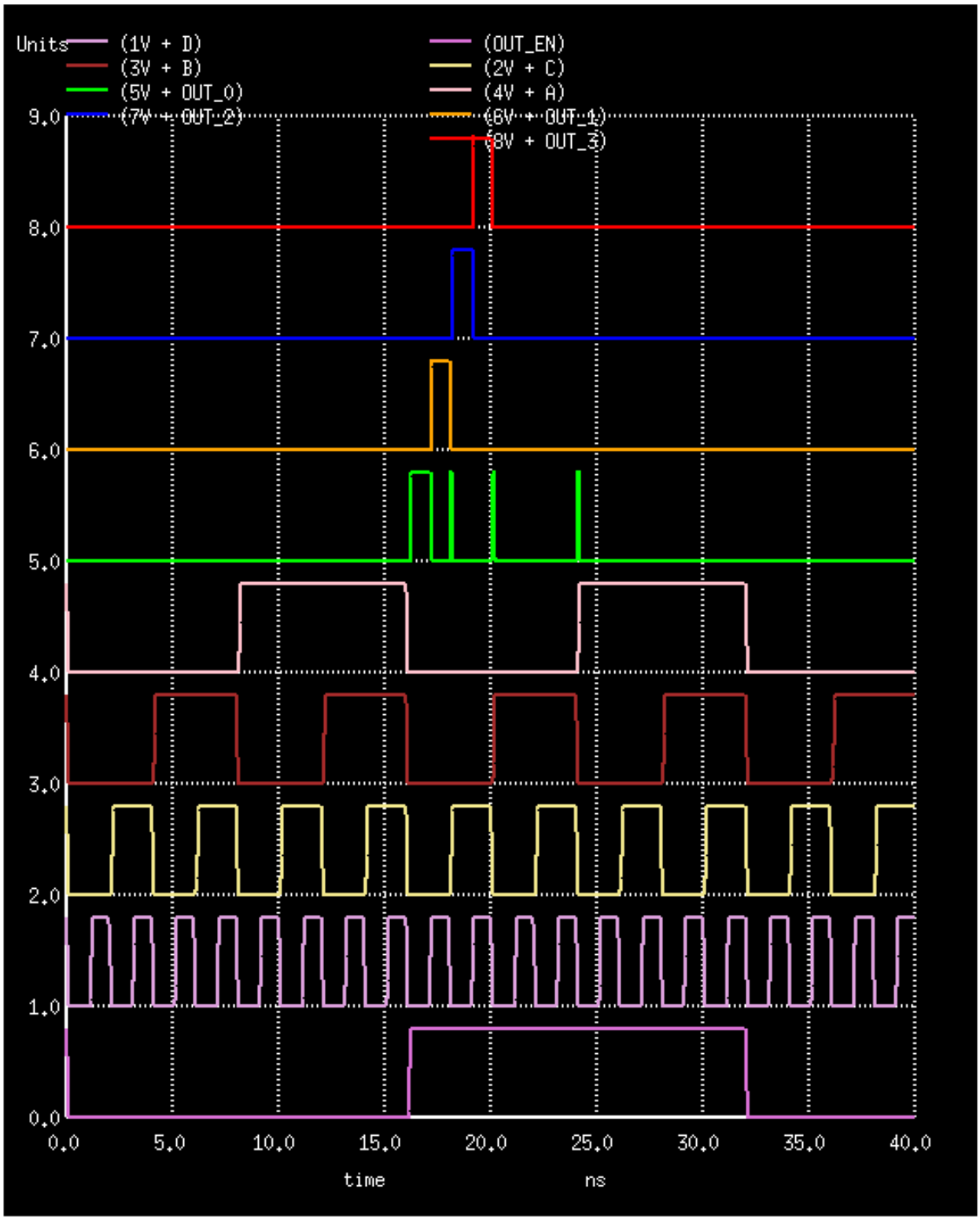

Figure 54: Decoder test, outputs 0 to 3

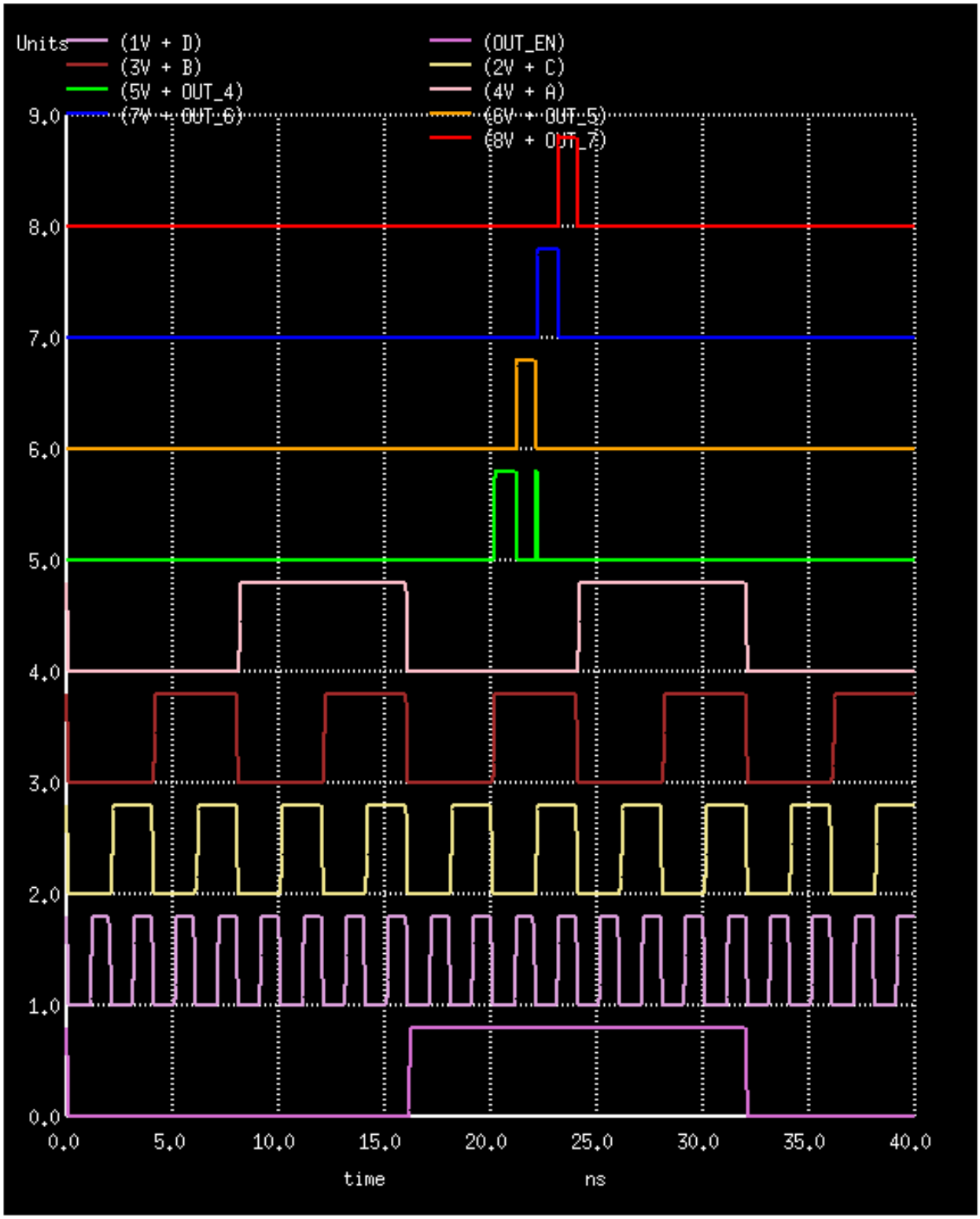

Figure 55: Decoder test, outputs 4 to 7

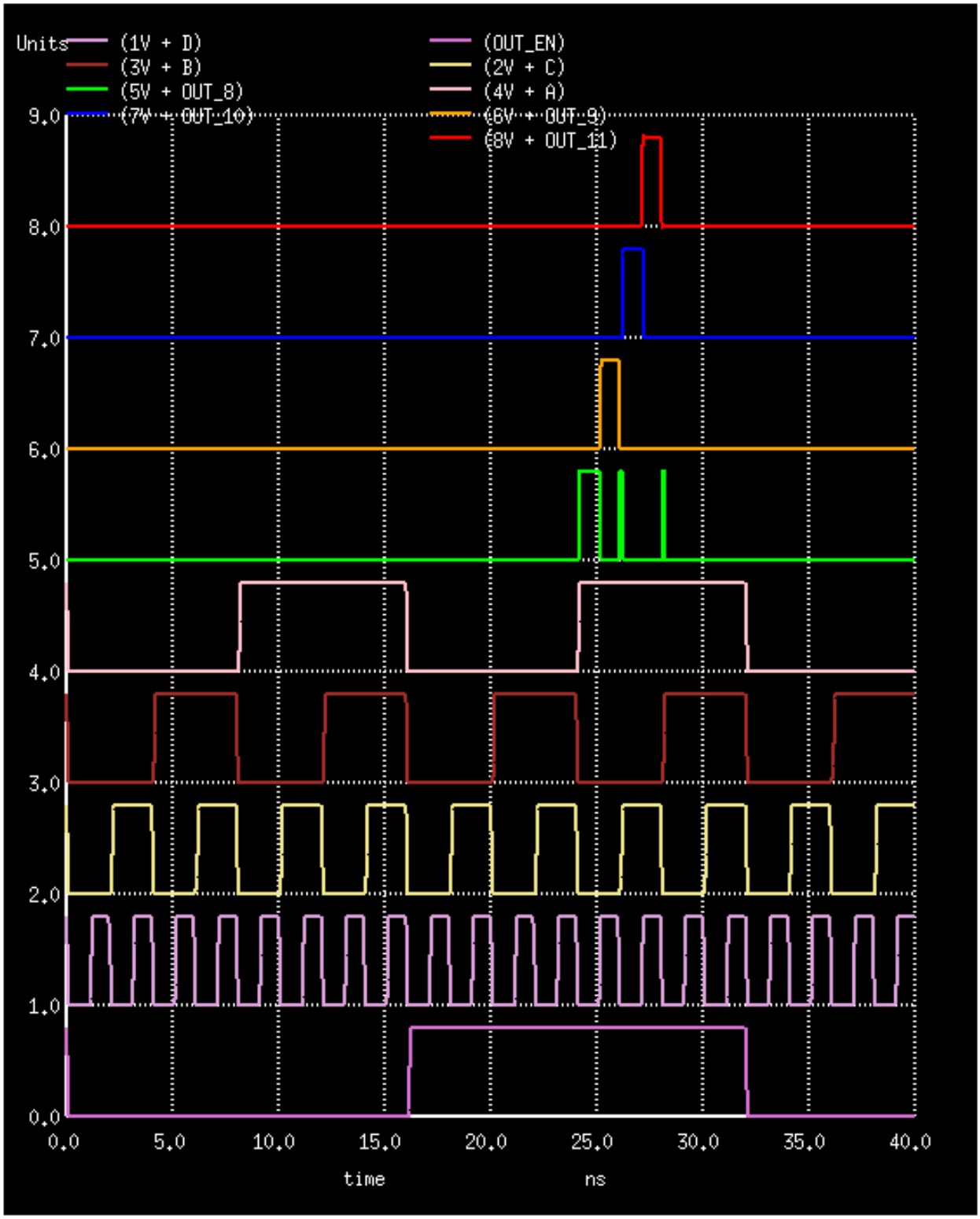

Figure 56: Decoder test, outputs 8 to 11

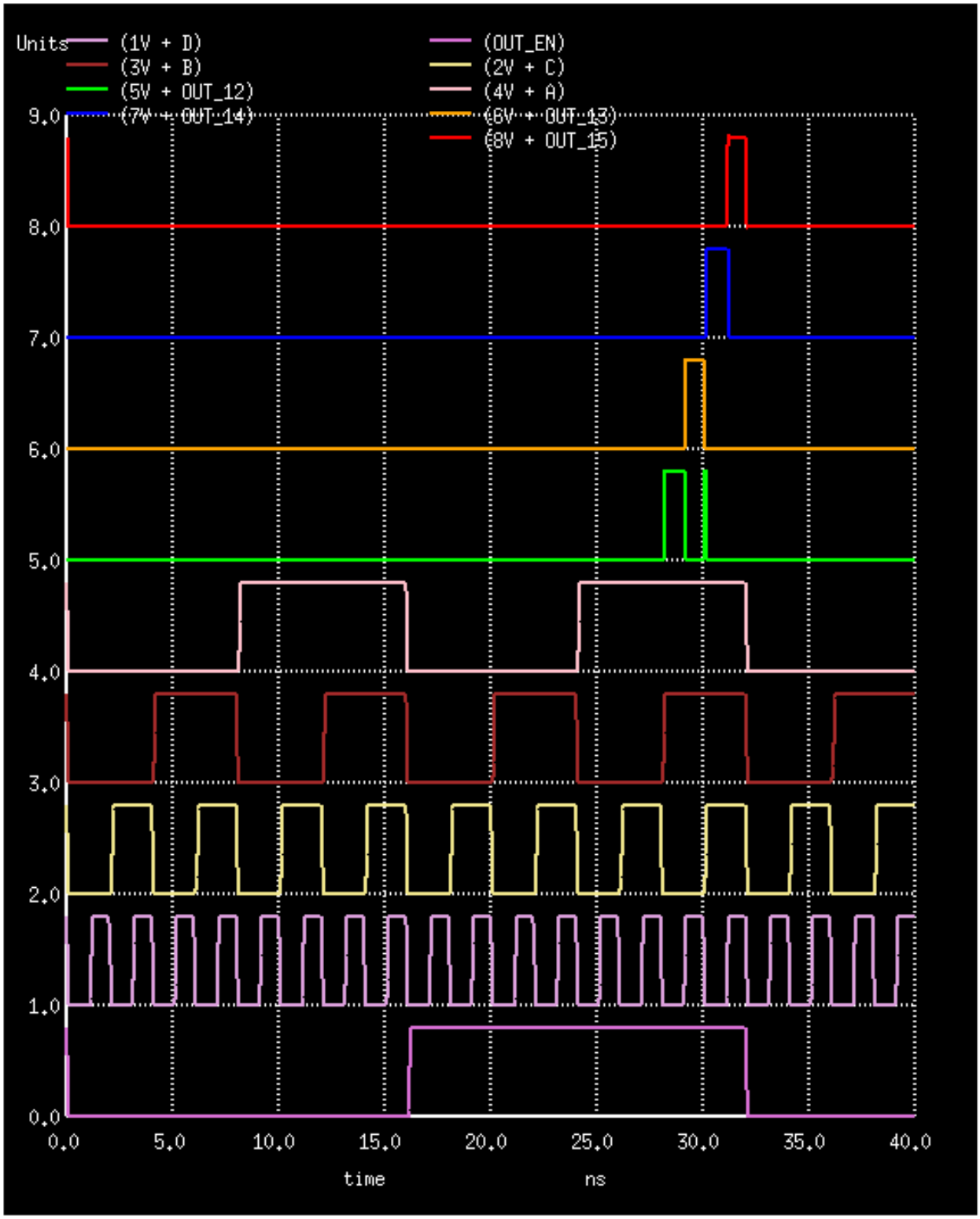

Figure 57: Decoder test, outputs 12 to 15

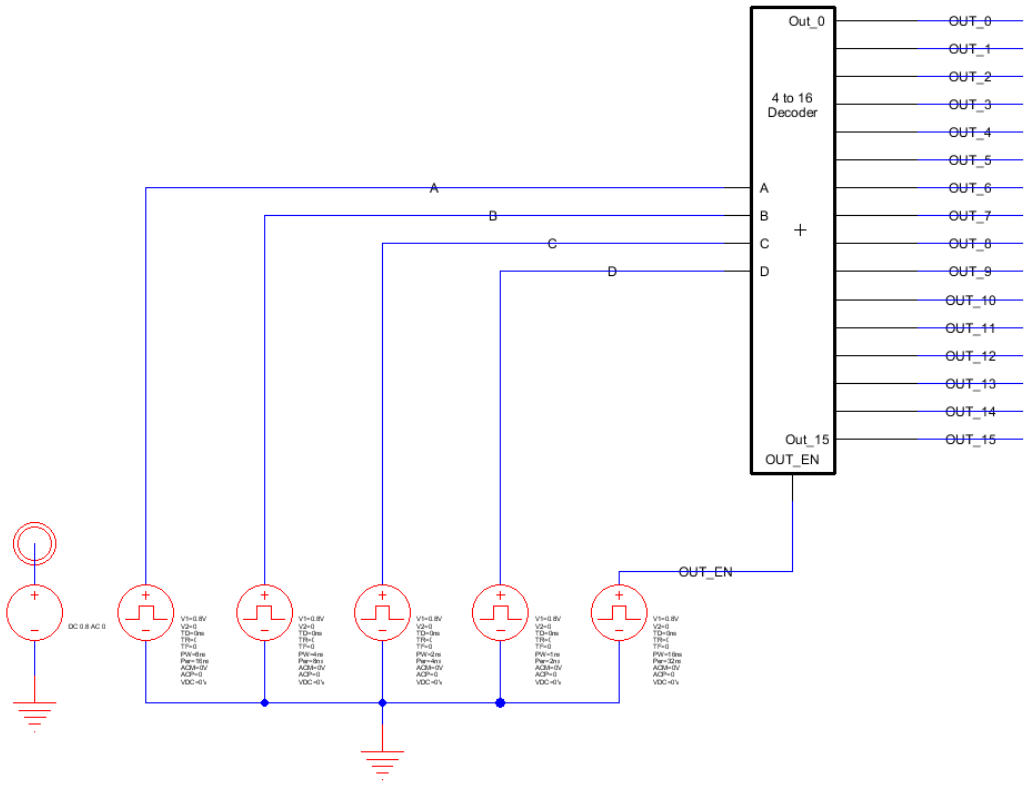

Figure 58: Decoder test schematic

Force Dequeue

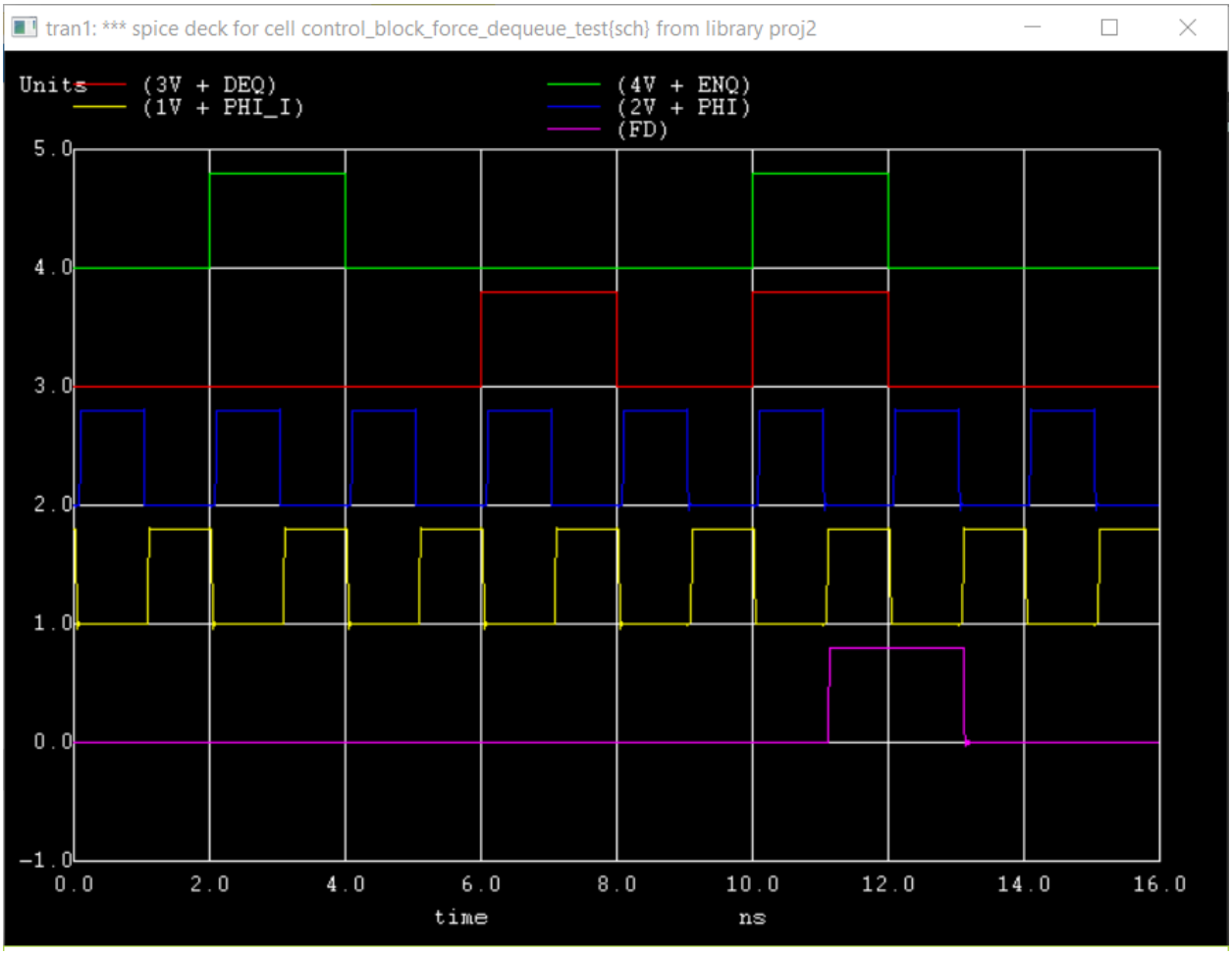

Figure 59: Force dequeue module test

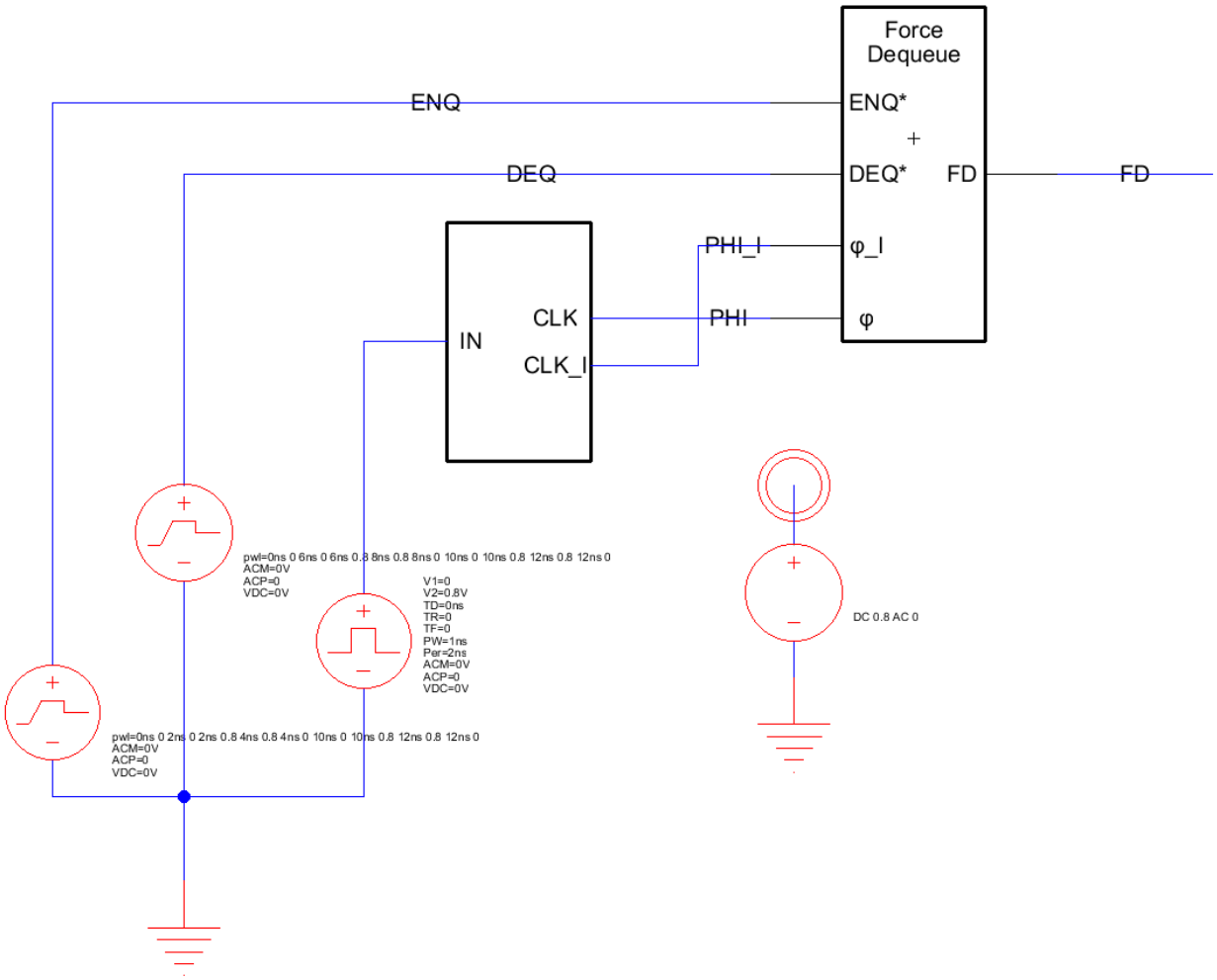

Figure 60: Force dequeue test schematic

4-bit Precharge Module

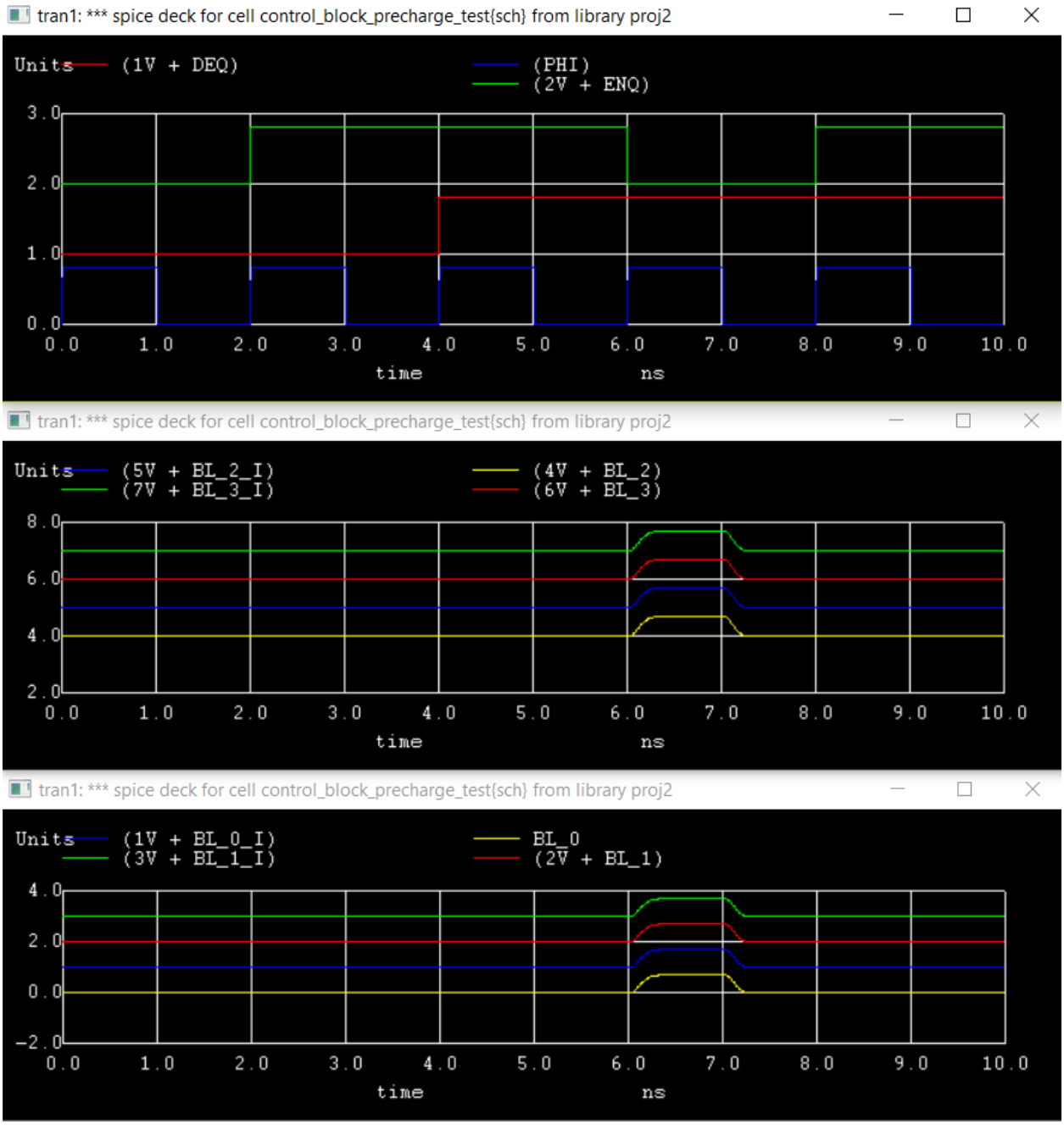

Figure 61: 4-bit precharge module test

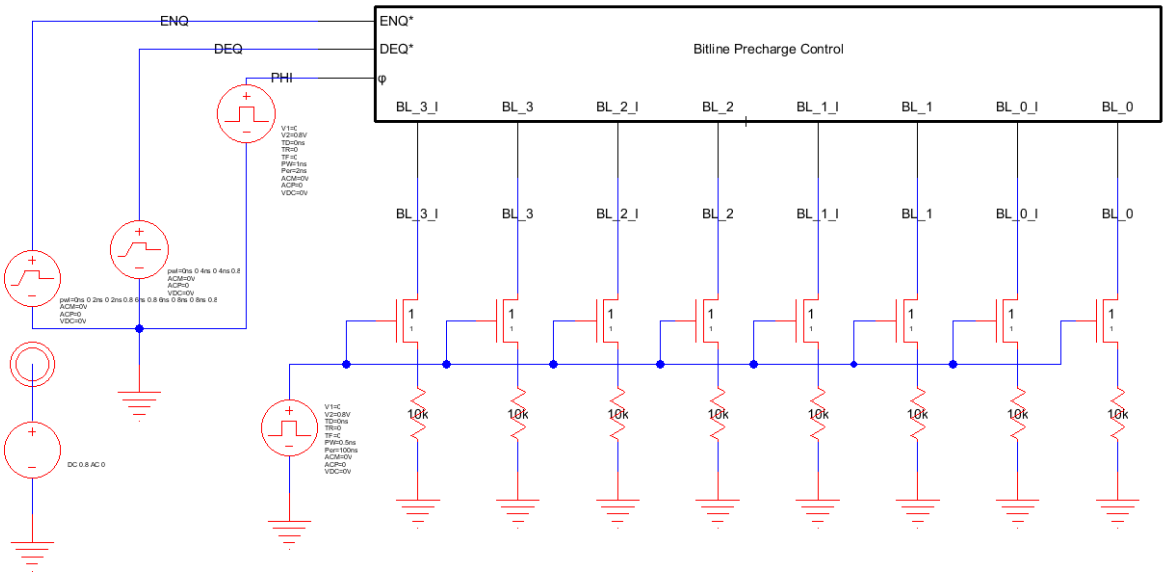

Figure 62: 4-bit precharge test schematic

Bitline Driver

Note that we used resistors in our test circuit to drive the outputs of the tri-state buffers low after write-enable was disabled. We did not use resistors in our actual design.

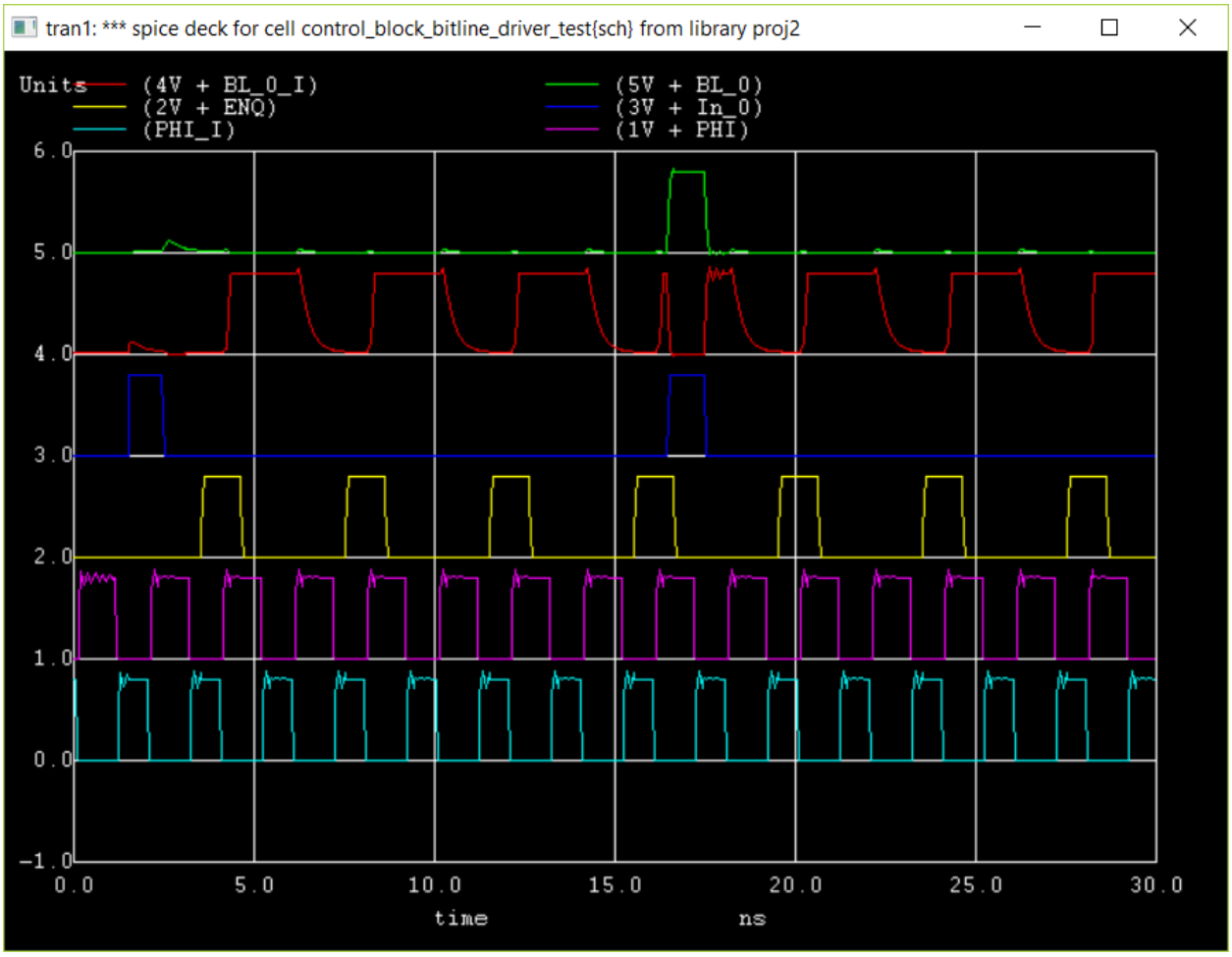

Figure 63: Bitline driver test, bit 0

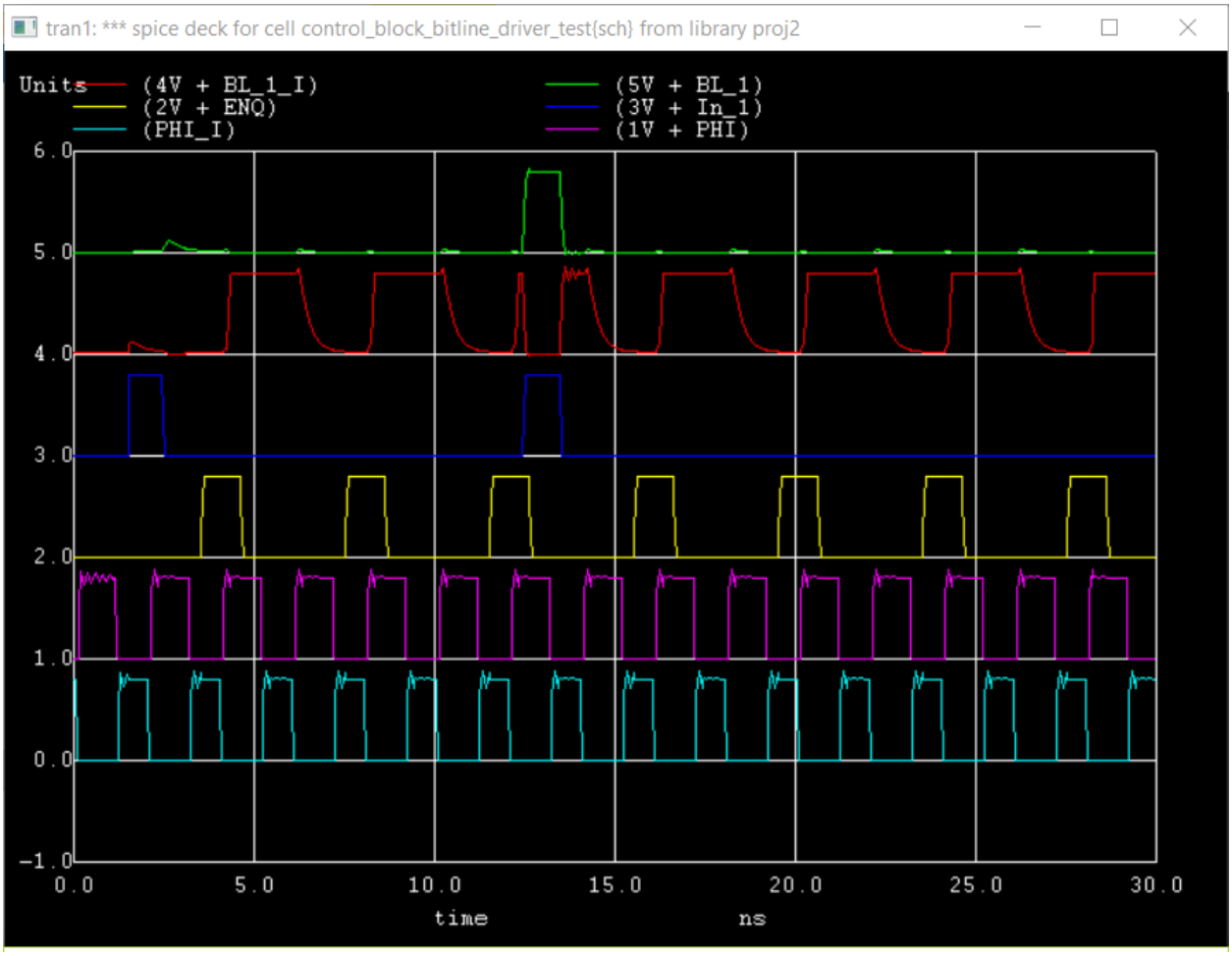

Figure 64: Bitline driver test, bit 1

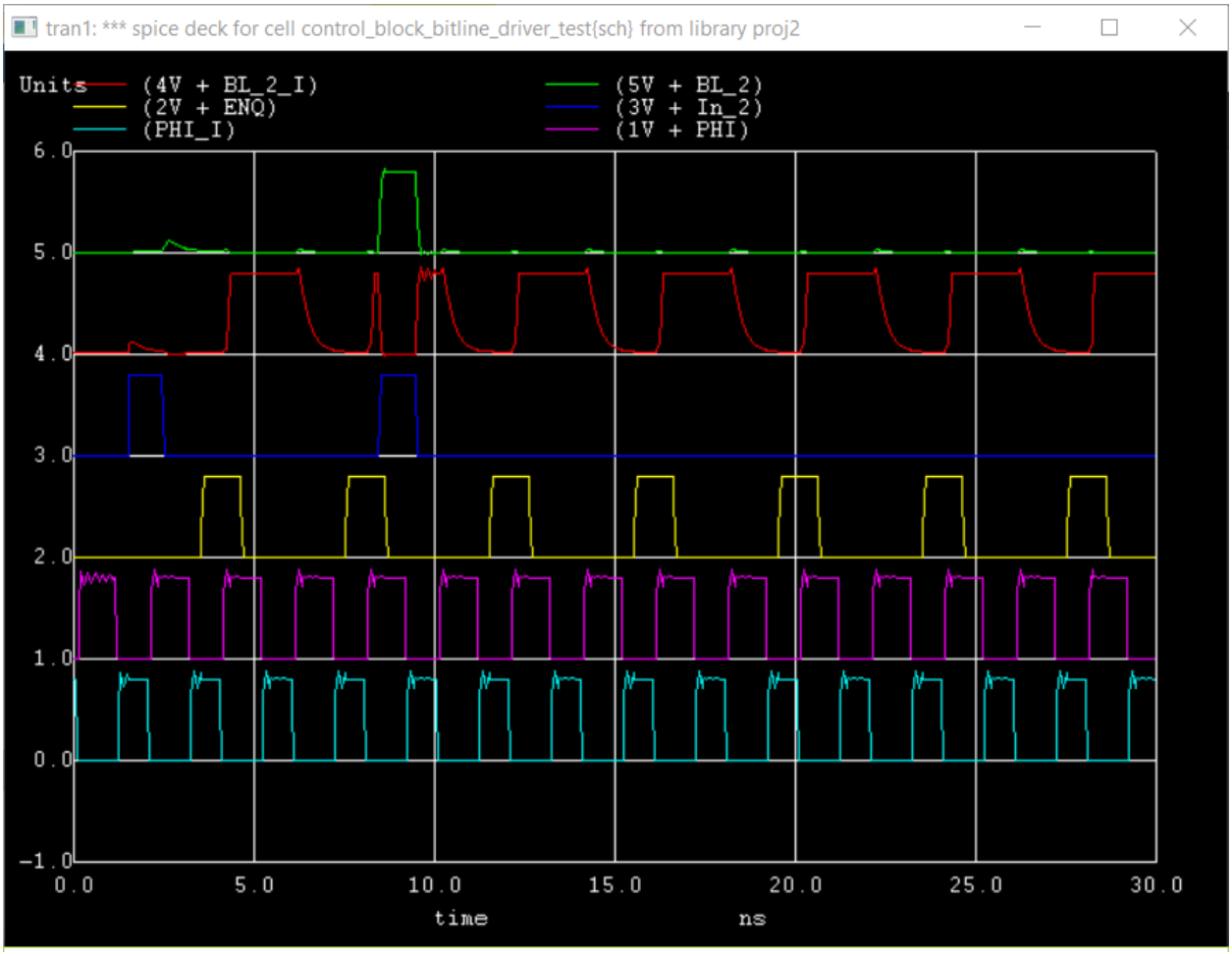

Figure 65: Bitline driver test, bit 2

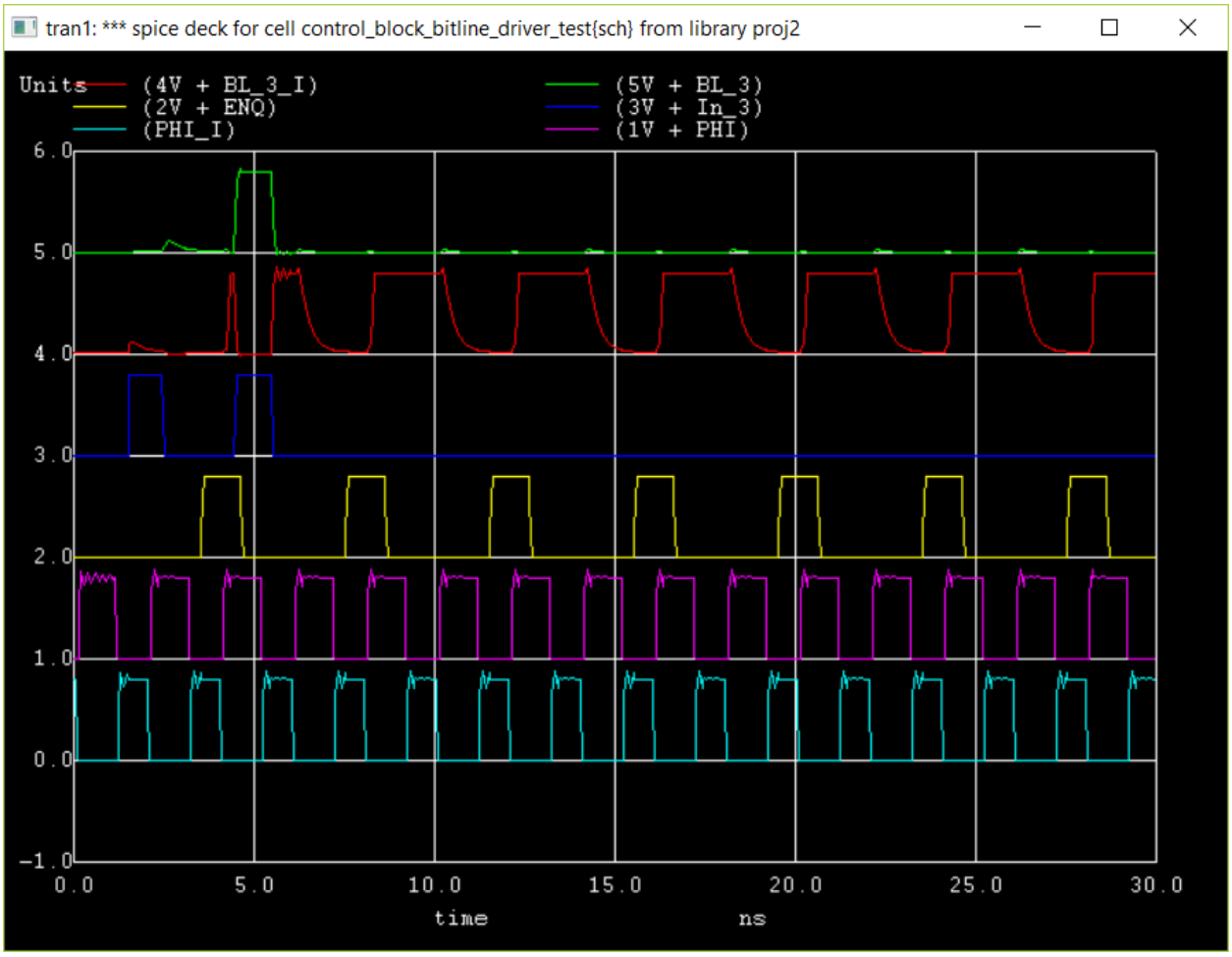

Figure 66: Bitline driver test, bit 3

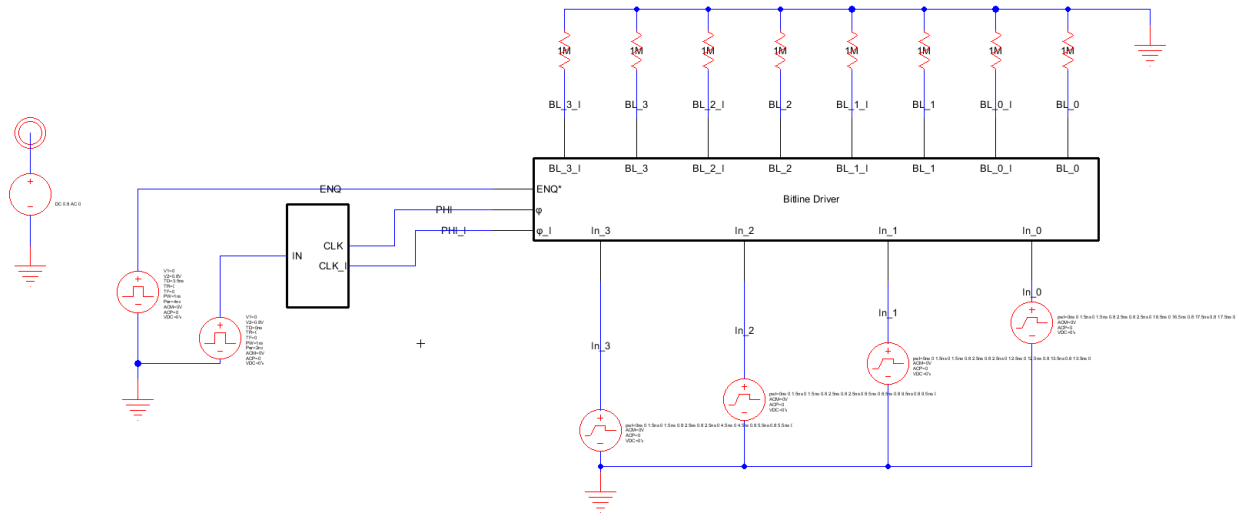

Figure 67: Bitline driver test schematic

Summary of Design Metrics

The design metrics are summarized as follows:

Area

Total area: 5892 (summing transistor widths). See breakdown below.

Memory Cell Area

Area: 50

Areas are broken down as follows:

```
General Components
    Inverter: 2
    NAND Gate: 4
    AND2 Gate: 6
    NOR Gate: 4
    OR Gate: 6
    XOR Gate: 12
    XNOR Gate: 12
    AND4 Gate: 18
      3 AND2 Gates
    Tri-state Buffer: 38
    Tri-state Inverter: 38
    D Latch: 22
      1 inverter: 2
      2 AND2 gates: 12
      2 NOR gates: 8
    D Register: 46
      2 D latches: 44
      2 NMOS gates for reset: 2
    4-bit D Register: 184
      4 D registers
    Incrementer: 74
      6 AND2 Gates: 36
      2 OR Gates: 12
      2 NAND Gates: 8
      1 XOR Gate: 12
      3 inverters: 6
    Comparator: 66
```

```
  4 XNOR Gates: 48
  1 AND4 Gate: 18
Mux Bitslice: 18
  2 AND2 Gates: 12
  1 OR2 Gate: 6
Mux: 72
  4 Mux Bitslices
Enq/Deq Module: 24
  3 AND2 Gates: 18
  1 OR2 Gate: 6
WL Enable Module: 12
  1 OR2 Gate: 6
  1 AND2 Gate: 6
Clock Generator: 58
  (directly in transistors)
Force Dequeue: 52
  AND Gate: 6
  D Register: 46
Pointers Module: 1108
  4 4-bit Registers: 736
  2 Incrementers: 148
  1 Comparator: 66
  1 D Register: 46
  2 AND4 Gates: 36
  11 AND2 Gates: 66
  1 OR2 Gate: 6
  2 Inverters: 4
1-bit Bitline Precharge Module: 32
  2 PMOS Gates (W = 16): 32
4-bit Bitline Precharge Module: 140
  4 1-bit Precharge Modules: 128
  1 NAND2 Gate: 4
  1 AND2 Gate: 6
  1 inverter: 2
Bitline Driver Module: 350
  4 Tri-state Buffers: 152
  4 Tri-state Inverters: 152
  1 D Register: 46
6T SRAM Cell: 50
  2 NMOS Gates (W = 8): 16
  2 NMOS Gates (W = 16): 32
  2 PMOS Gates (W = 1): 2
4-to-16 Decoder: 392
  16 AND4 Gates: 288
  4 inverters: 8
```

```
      16 AND2 Gates: 96


Total Top-Level Design Area: 5892
    Control Block: 2284
      1 Clock Generator: 58
      1 Force Dequeue Module: 52
      1 Pointers Module: 1108
      2 4-bit Registers: 368
      1 Mux: 72
      1 Enq/Deq Module: 24
      1 WL Enable Module: 12
      1 4-bit Bitline Precharge Module: 140
      1 Bitline Driver Module: 350
      2 D Registers: 92
      4 inverters: 8
    Memory Block: 3592
      64 6T SRAM Cells: 3200
      1 4-to-16 Decoder: 392
    8 inverters for buffering: 16
```

Enqueue Energy

Enqueueing 0x1111 into 0x0000: $4.78\text{x}10^{-14}$ J (higher value). This was done using the command

```
meas tran yint integ I(vv_generi@0) from=36ns to=38ns
```

and produced the results

```
yint = 5.97328e-14 from=  3.60000e-08 to=  3.80000e-08
```

(Energy was multiplied by 0.8 since $V_{dd} = 0.8$V.)

Enqueueing 0x0000 into 0x1111: $4.73\text{x}10^{-14}$ J.

```
meas tran yint integ I(vv_generi@0) from=36ns to=38ns
yint = -5.90996e-14 from=  3.60000e-08 to=  3.80000e-08
```

Dequeue Energy

Dequeueing 0x0000: $3.89\text{x}10^{-14}$ J (higher value).

```
yint = -4.86632e-014 from=  8.00000e-009 to=  1.00000e-008
```

Dequeueing 0x1111: 3.818792x10^{-14} J

```
yint = -4.77349e-014 from=  8.00000e-009 to=  1.00000e-008
```

Simultaneous Enqueue/Dequeue Energy

Result: 3.95x10^{-13} J

```
meas tran yint integ I(vv_generi@0) from=36ns to=40ns
yint = -4.93721e-13 from=  3.60000e-08 to=  4.00000e-08
```

Standby Energy

Result: 1.54x10^{-14} J

```
yint = -1.93039e-014 from=  4.00000e-009 to=  6.00000e-009)
```

Average Energy

We calculated the average energy as a weighted average of the energies determined above and using the distribution specified in the report writeup. The average energy was 7.79x10^{-14} J and was calculated as $0.15 * 3.95\text{x}10^{-13}\text{J} + 0.1 * 4.78\text{x}10^{-14} + 0.1 * 3.89\text{x}10^{-14}\text{J} + 0.65 * 1.54\text{x}10^{-14}$ J

Honor Pledge

We, Jack Harkins and Mauricio Mutai, certify that we have complied with the University of
Pennsylvania's Code of Academic Integrity in completing this project.