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

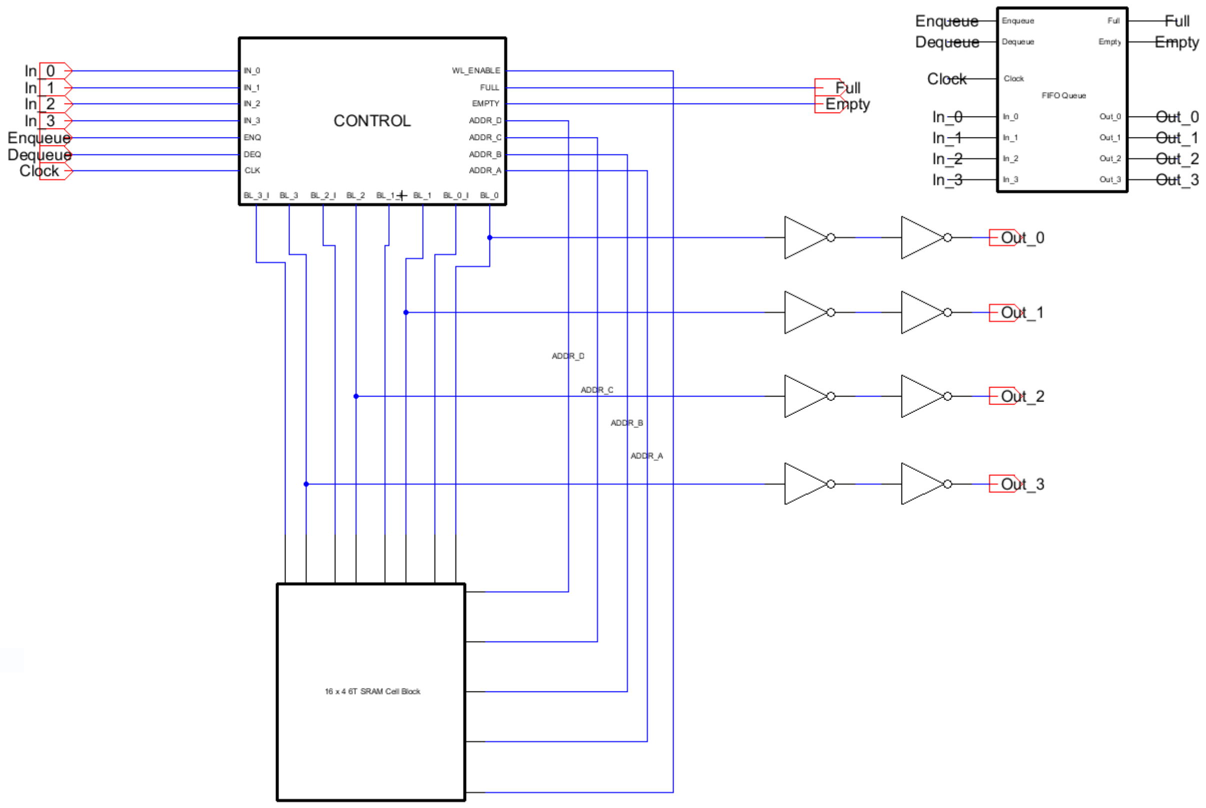


Figure 1: Queue top-level schematic

Memory Cell Block

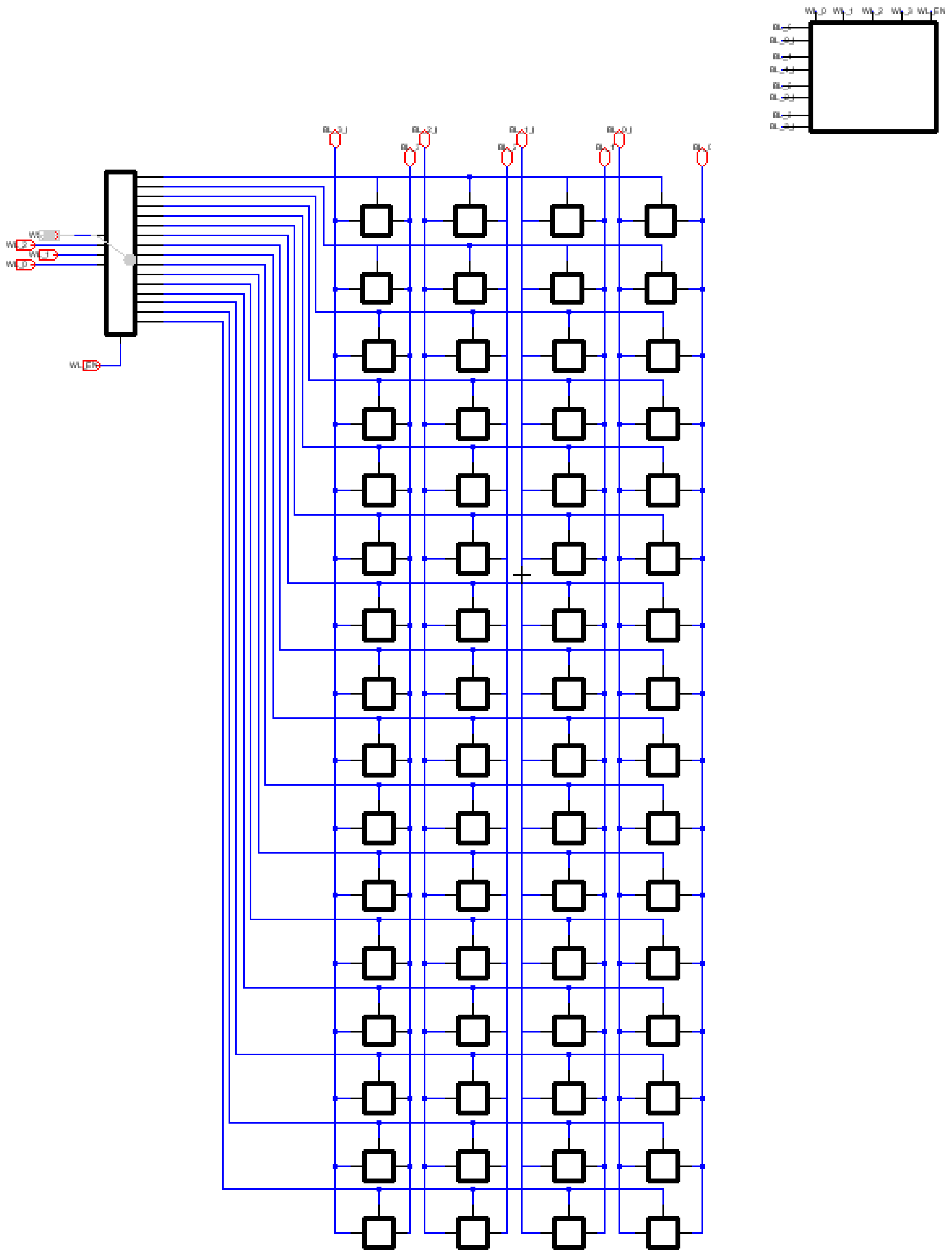


Figure 2: Memory block toplevel schematic

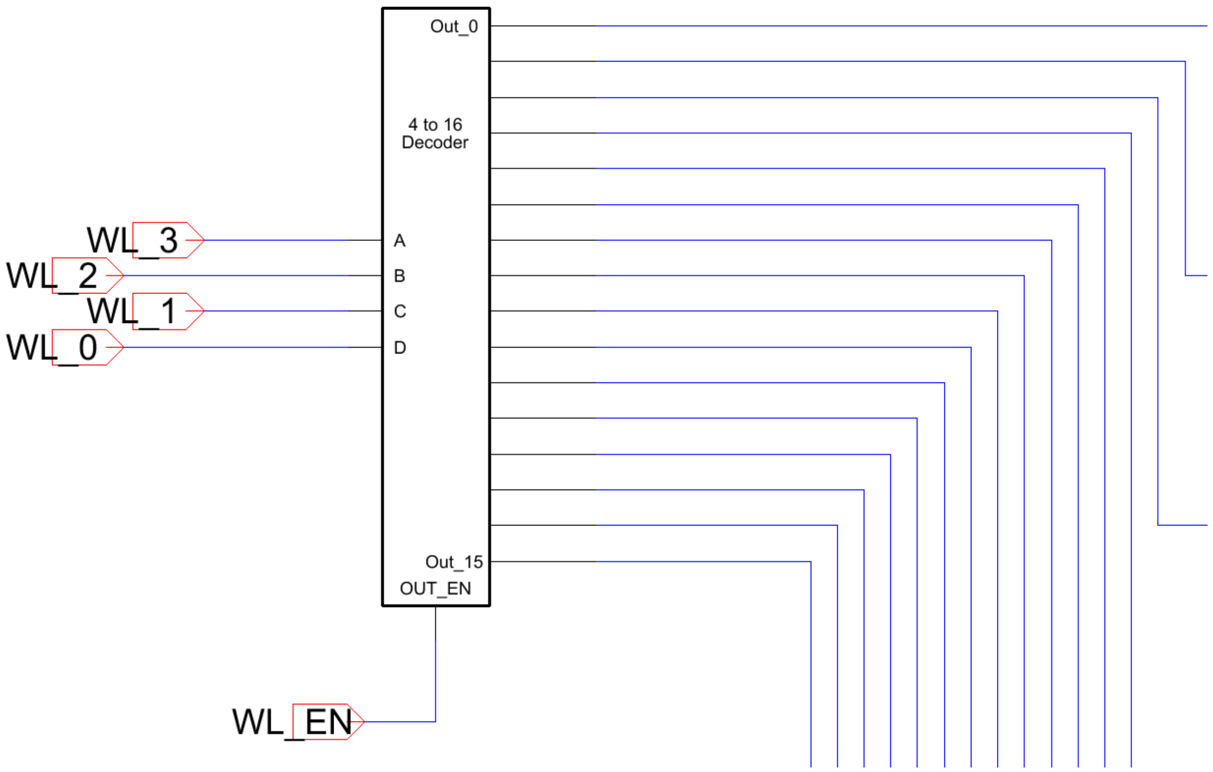


Figure 3: Memory block schematic in detail, decoder

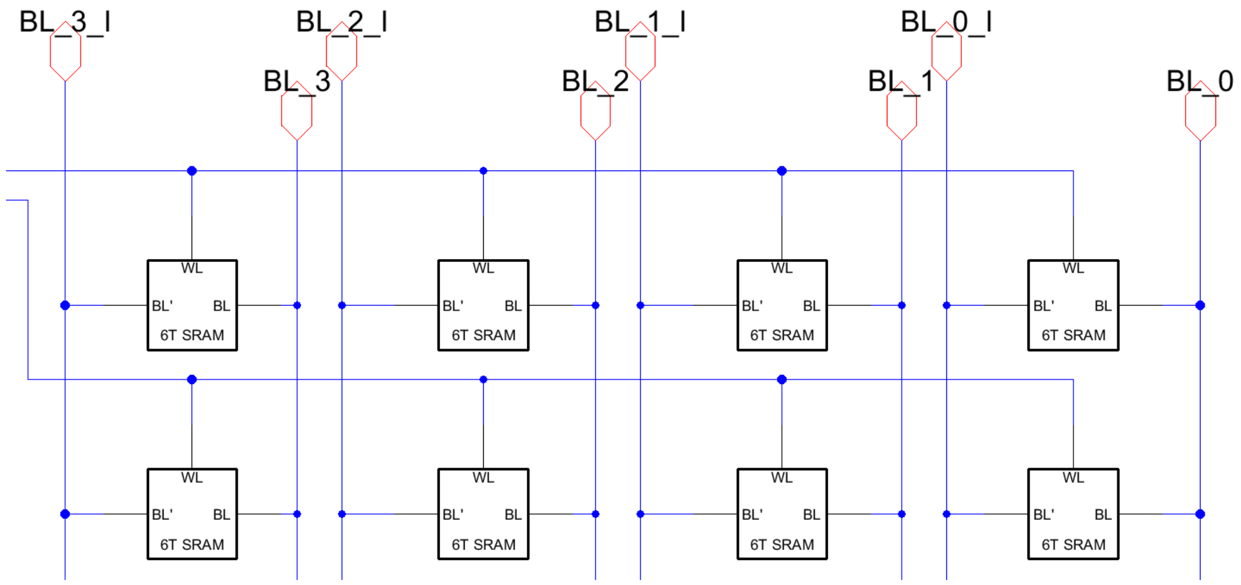


Figure 4: Memory block schematic in detail, word slice

6T SRAM Cell

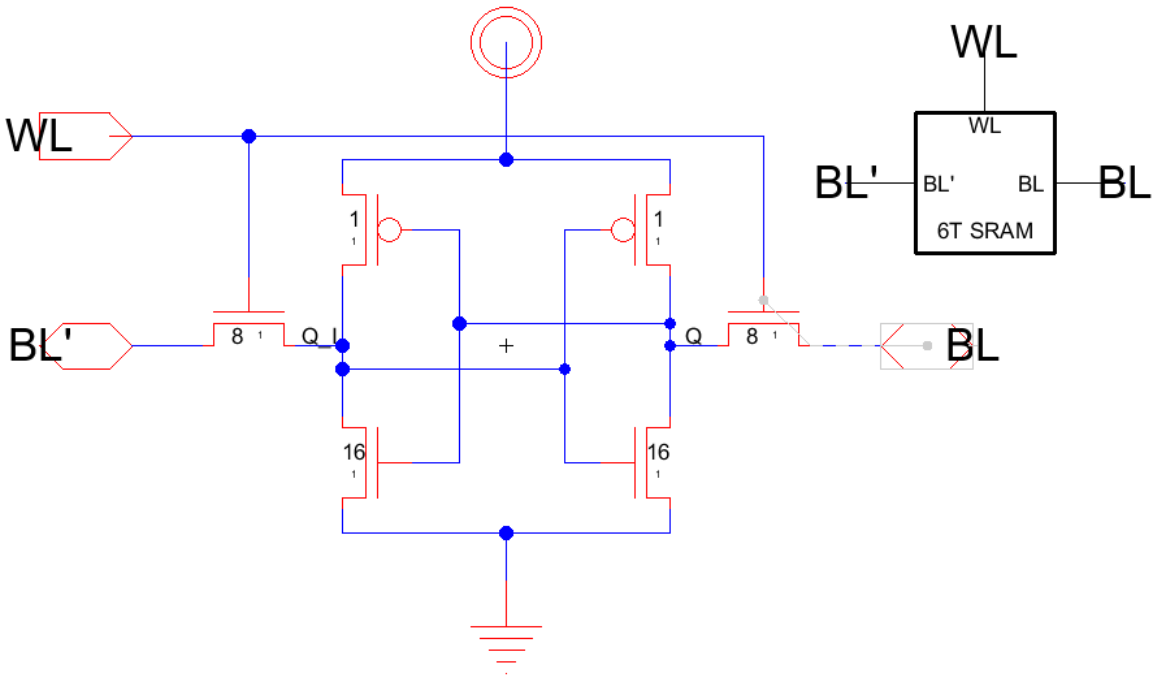


Figure 5: SRAM cell schematic

Control Block

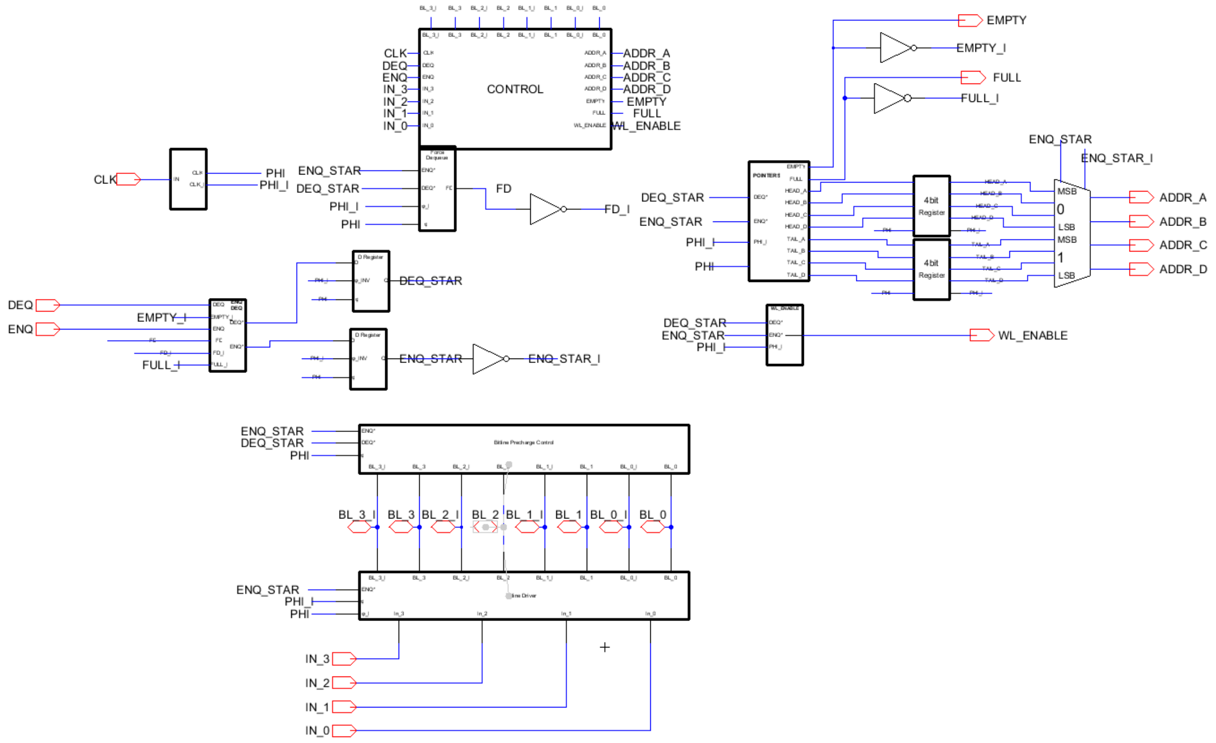


Figure 6: Control block toplevel schematic

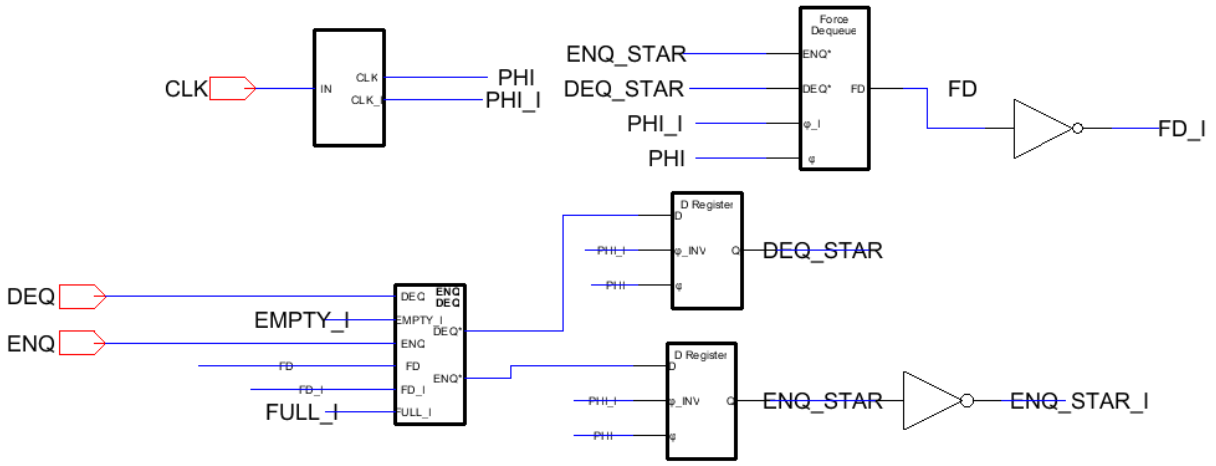


Figure 7: Control block schematic in detail, FD, clock generator, and ENQ*/DEQ* generation

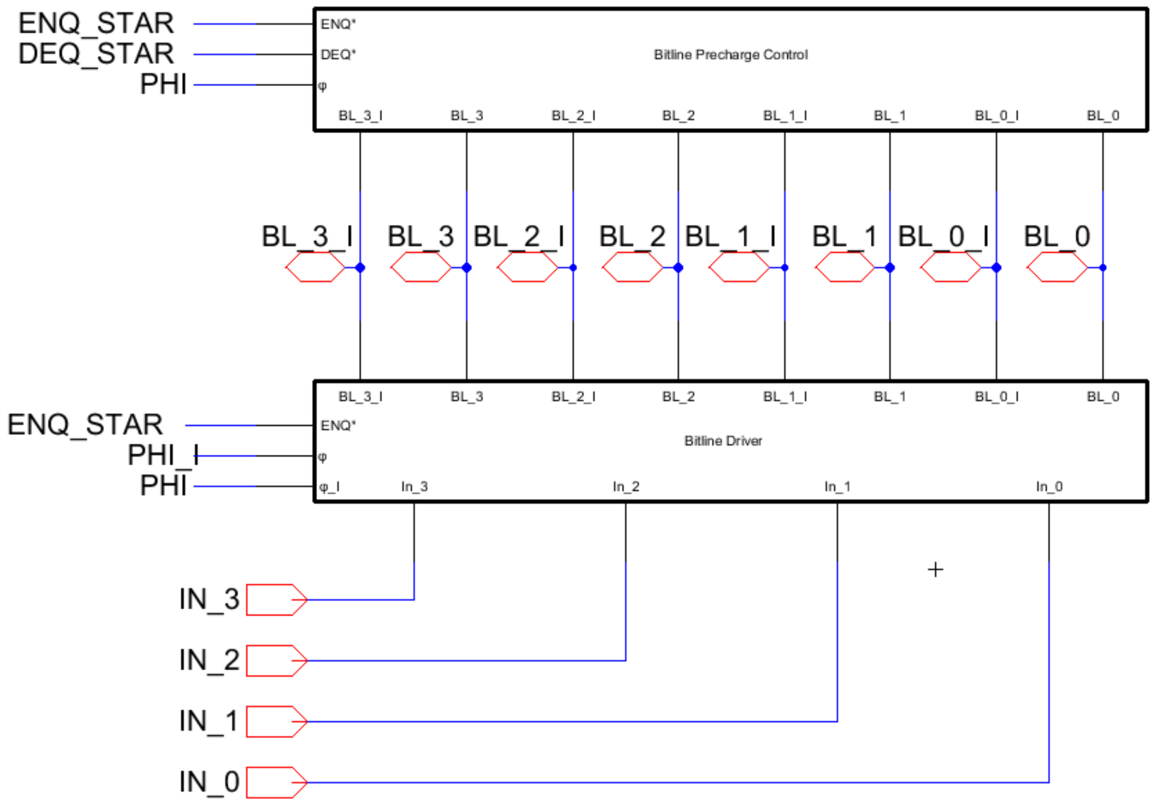


Figure 8: Control block schematic in detail, bitline driver and precharger

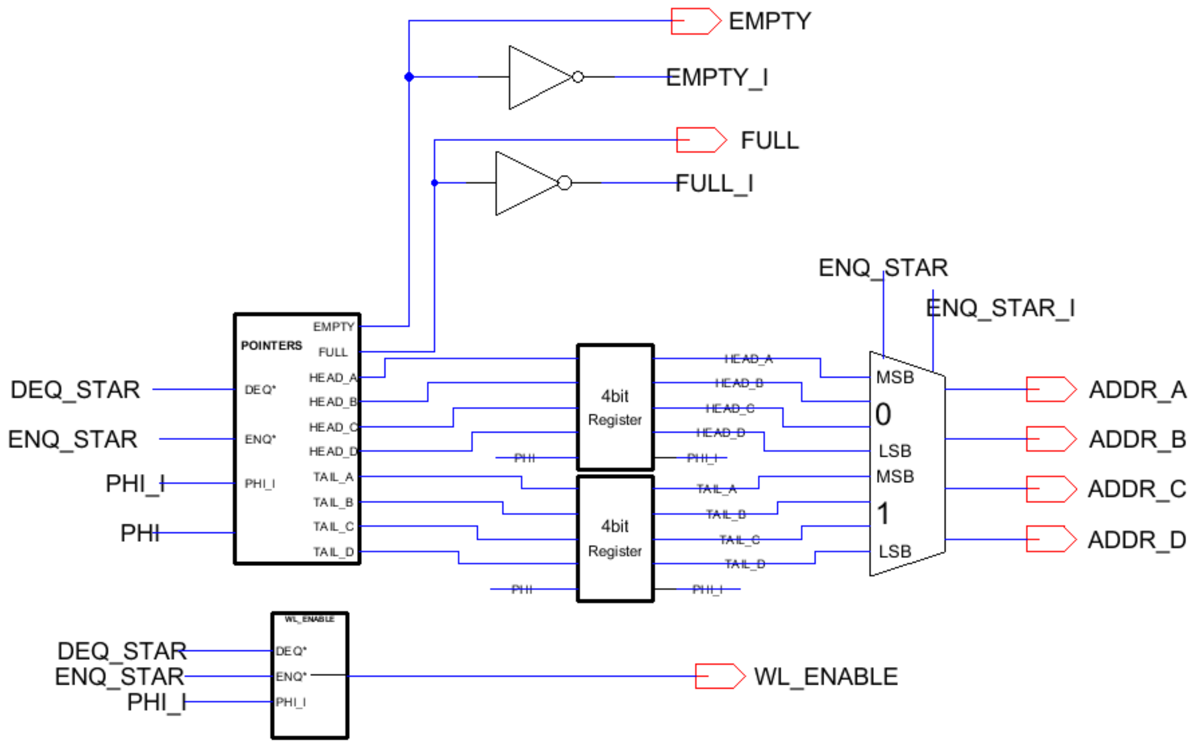


Figure 9: Control block schematic in detail, pointers, WL_EN, and EMPTY/FULL generation

Clock Generator

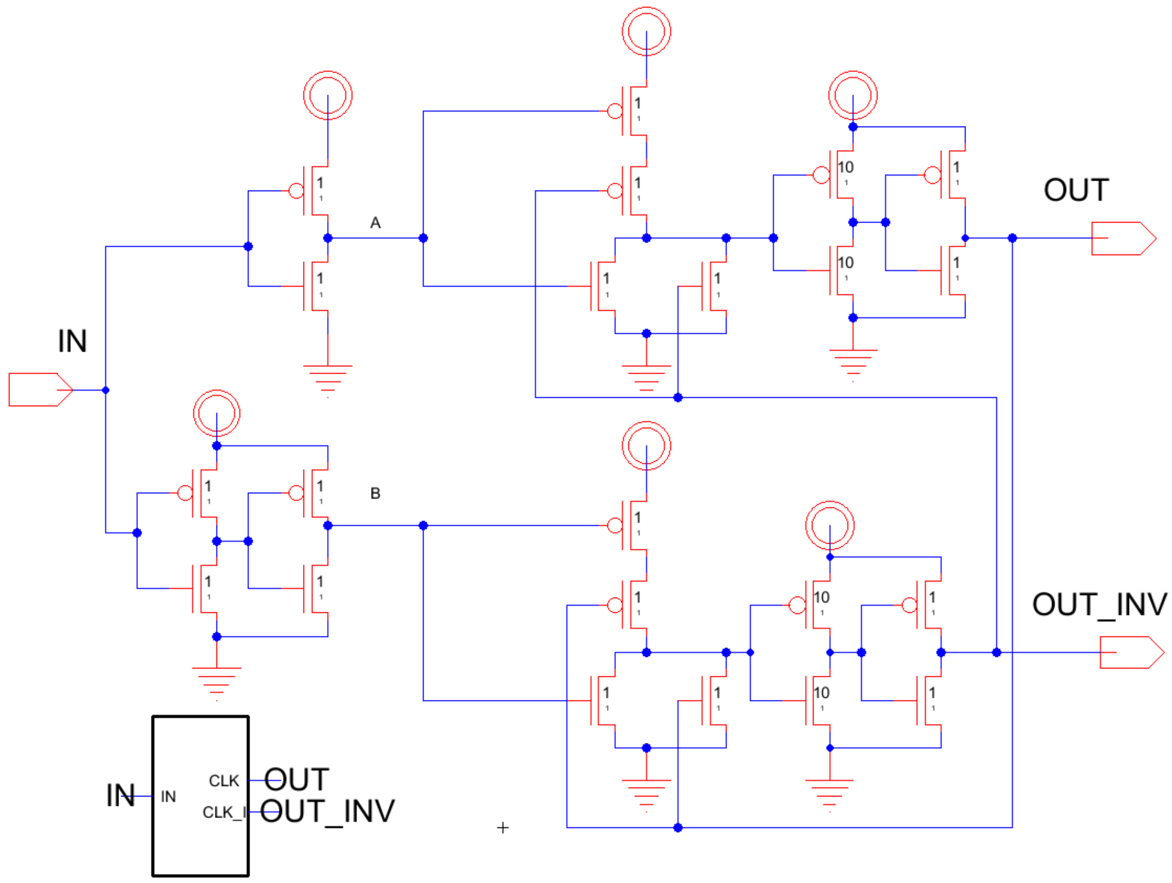


Figure 10: Clock generator schematic

Force Dequeue (FD)

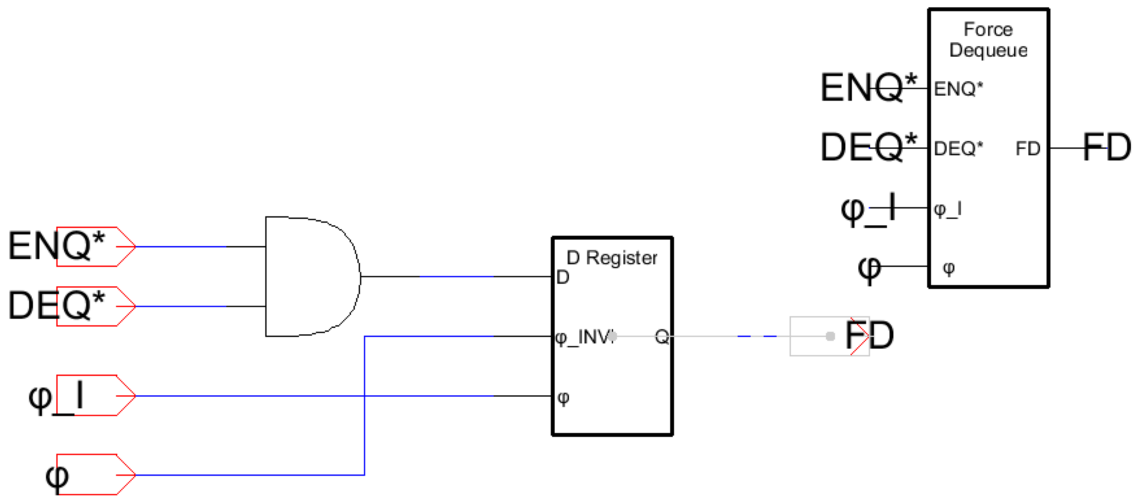


Figure 11: Force dequeue generation schematic

Pointers

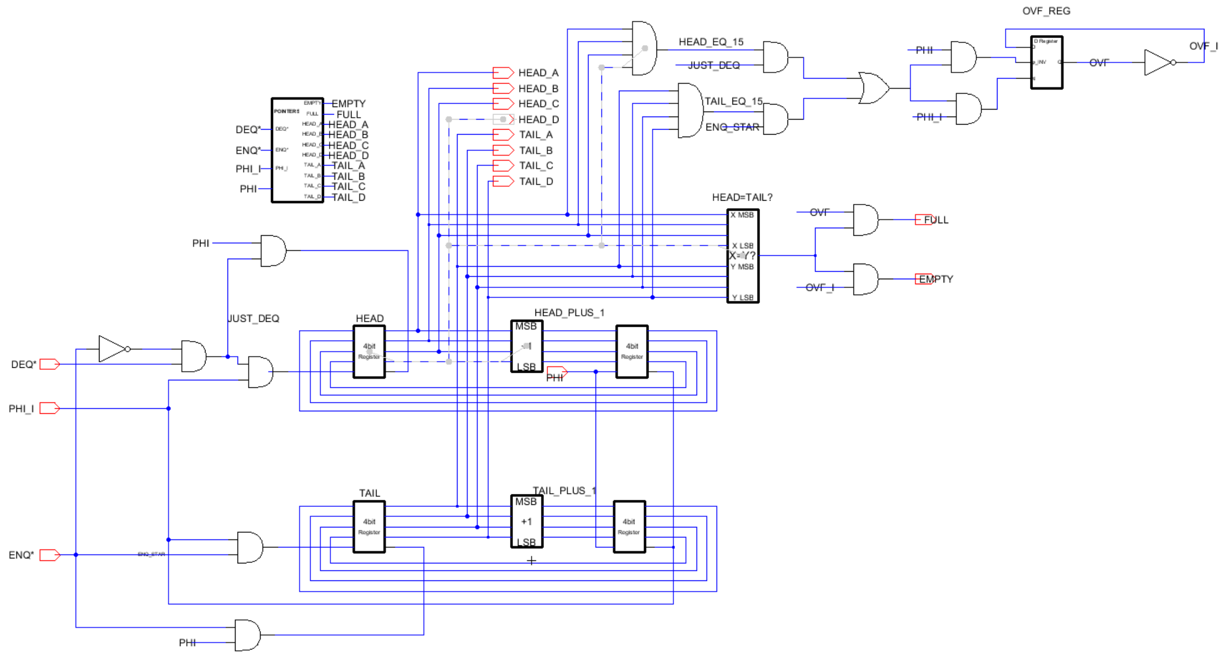


Figure 12: Pointers toplevel schematic

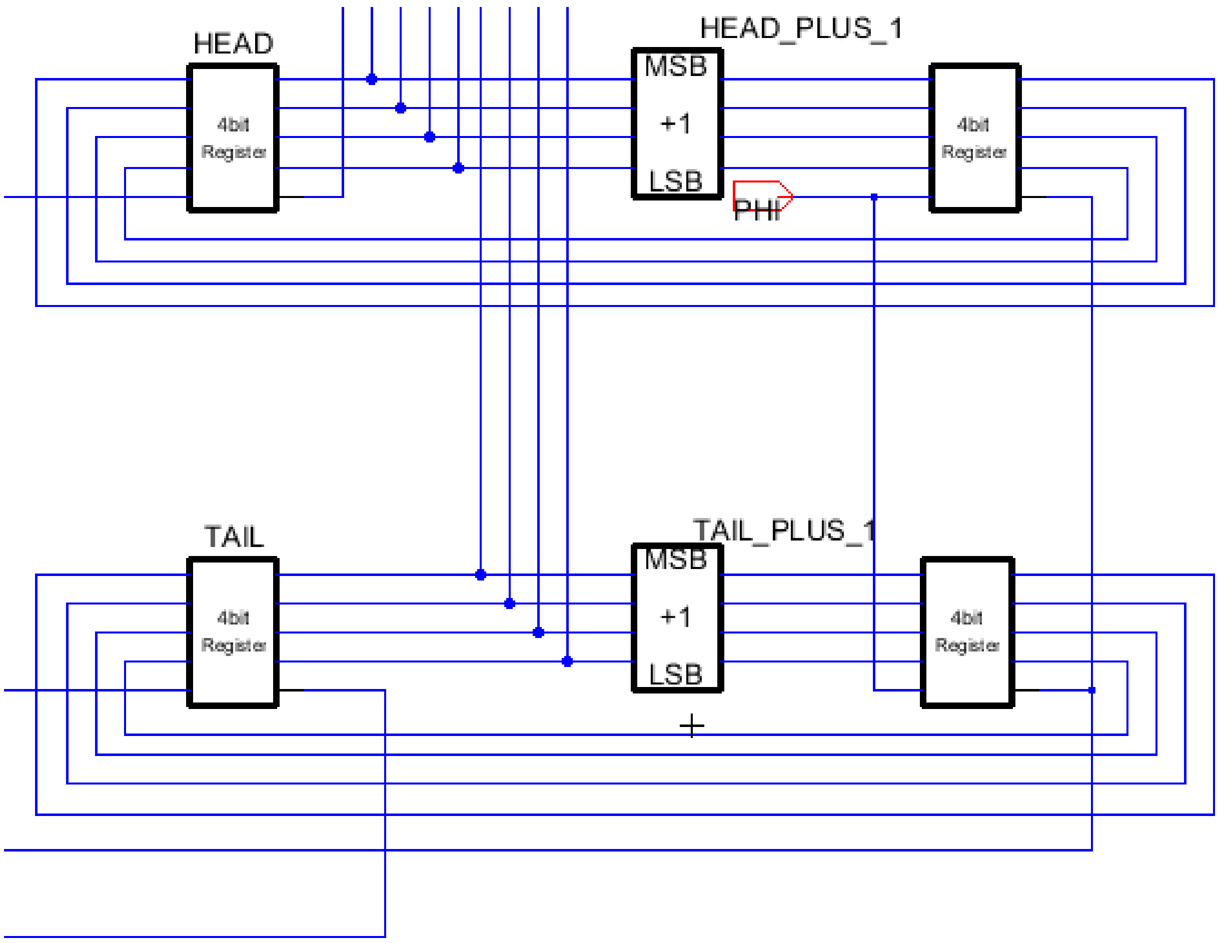


Figure 13: Pointers schematic in detail, head/tail state

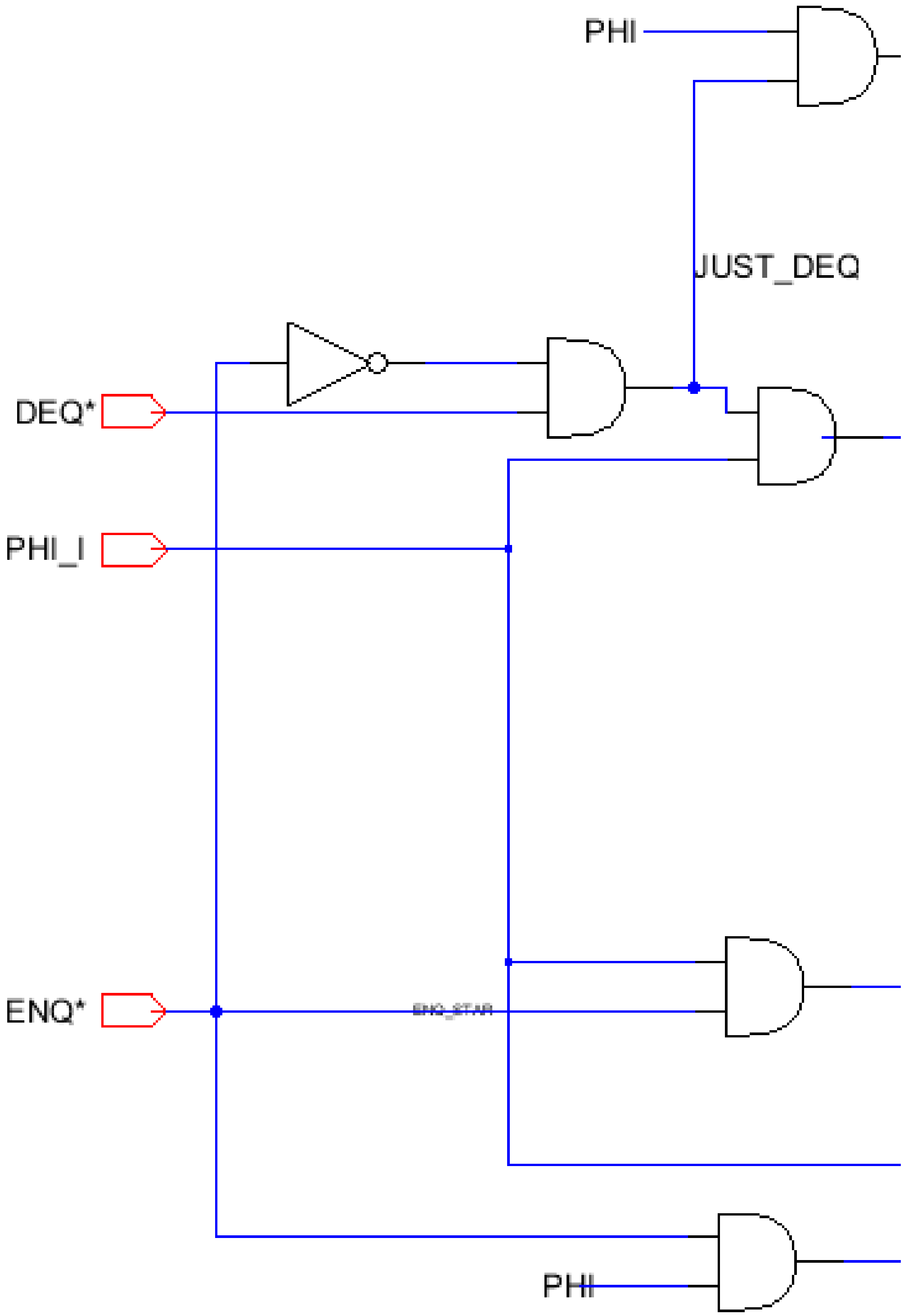


Figure 14: Pointers schematic in detail, inputs

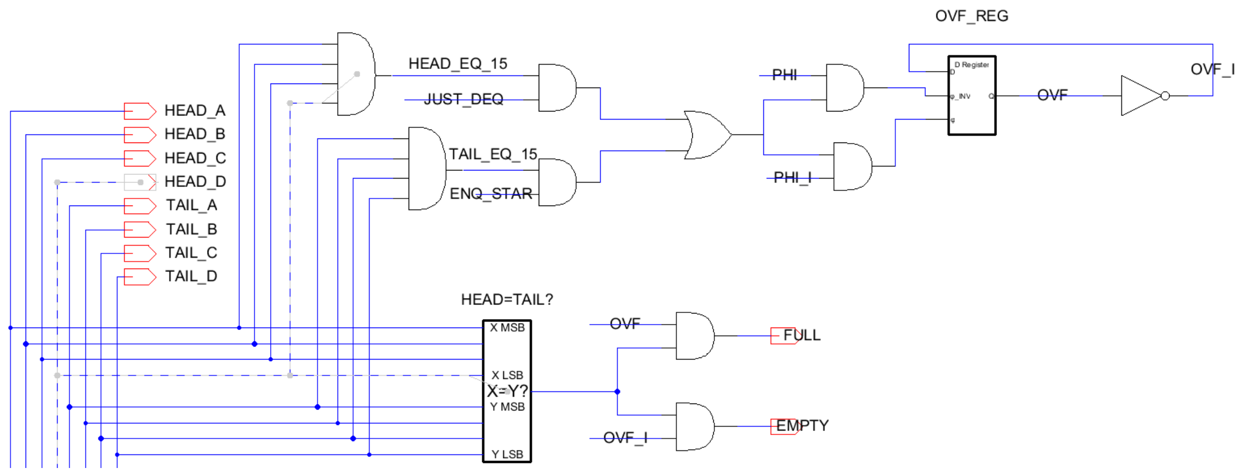


Figure 15: Pointers schematic in detail, overflow handler

D Latch

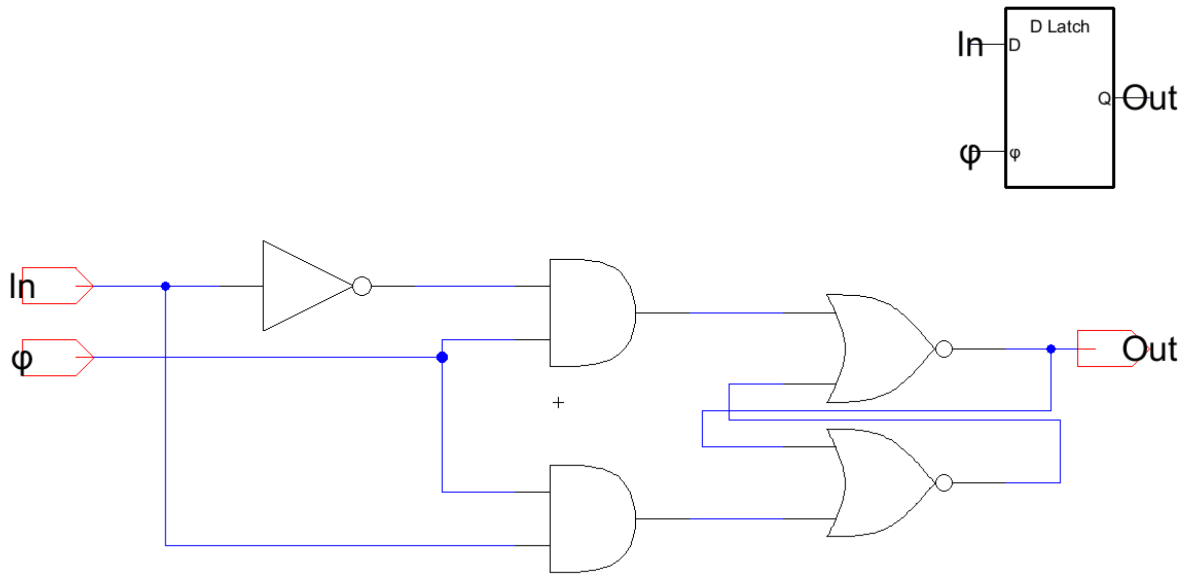


Figure 16: D latch schematic

D Register

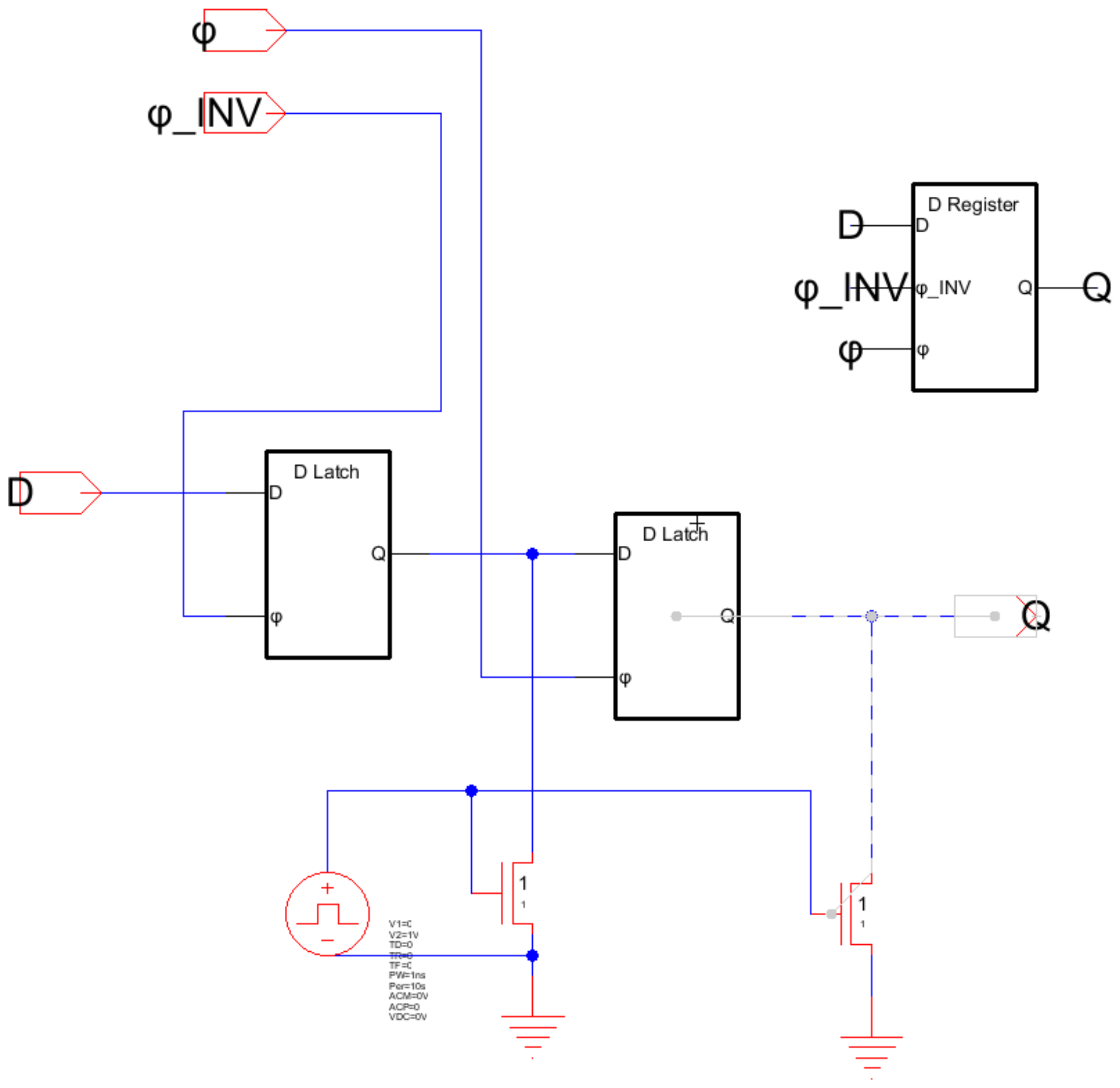


Figure 17: D register schematic

4-bit Register

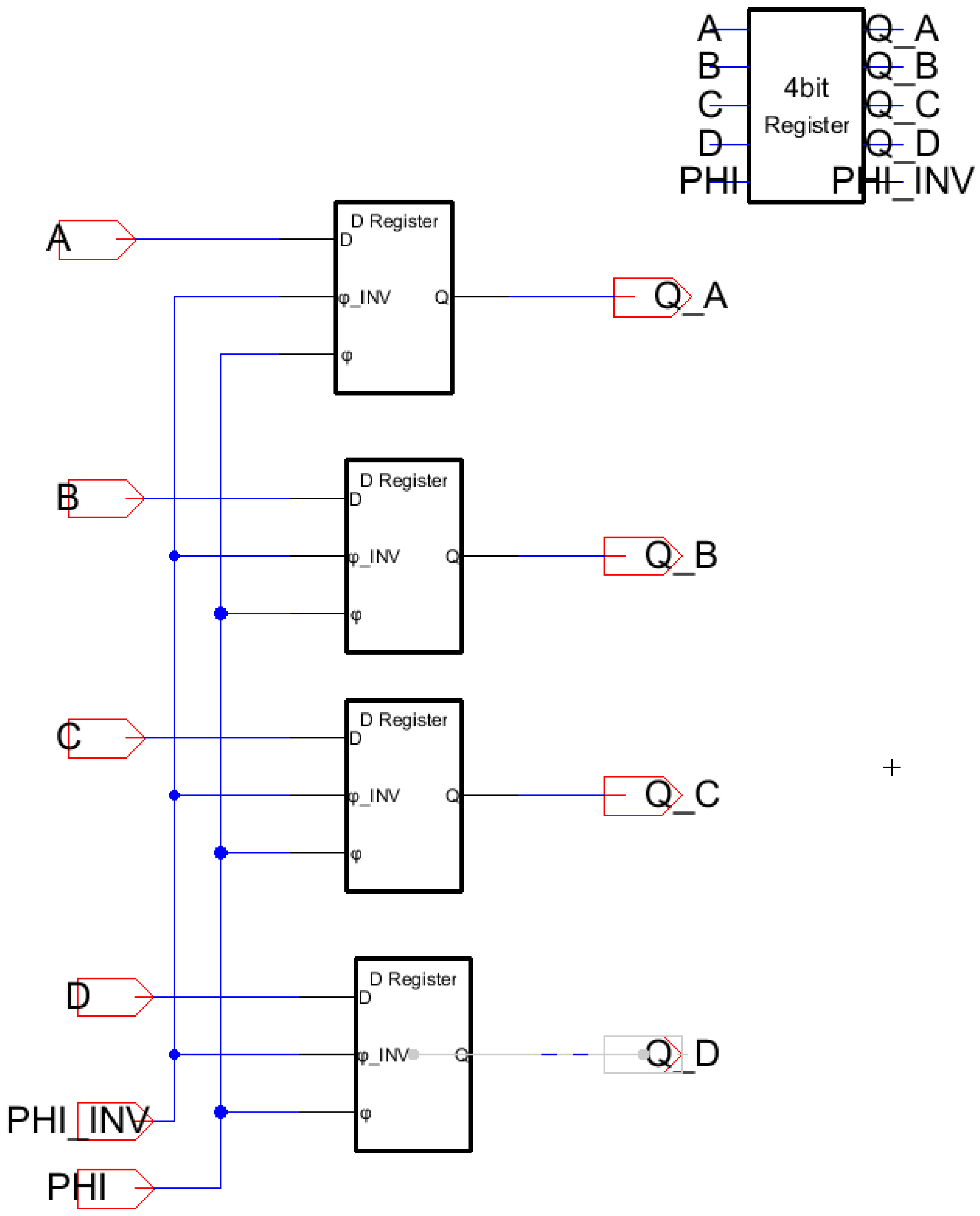


Figure 18: 4-bit register schematic

Incrementer

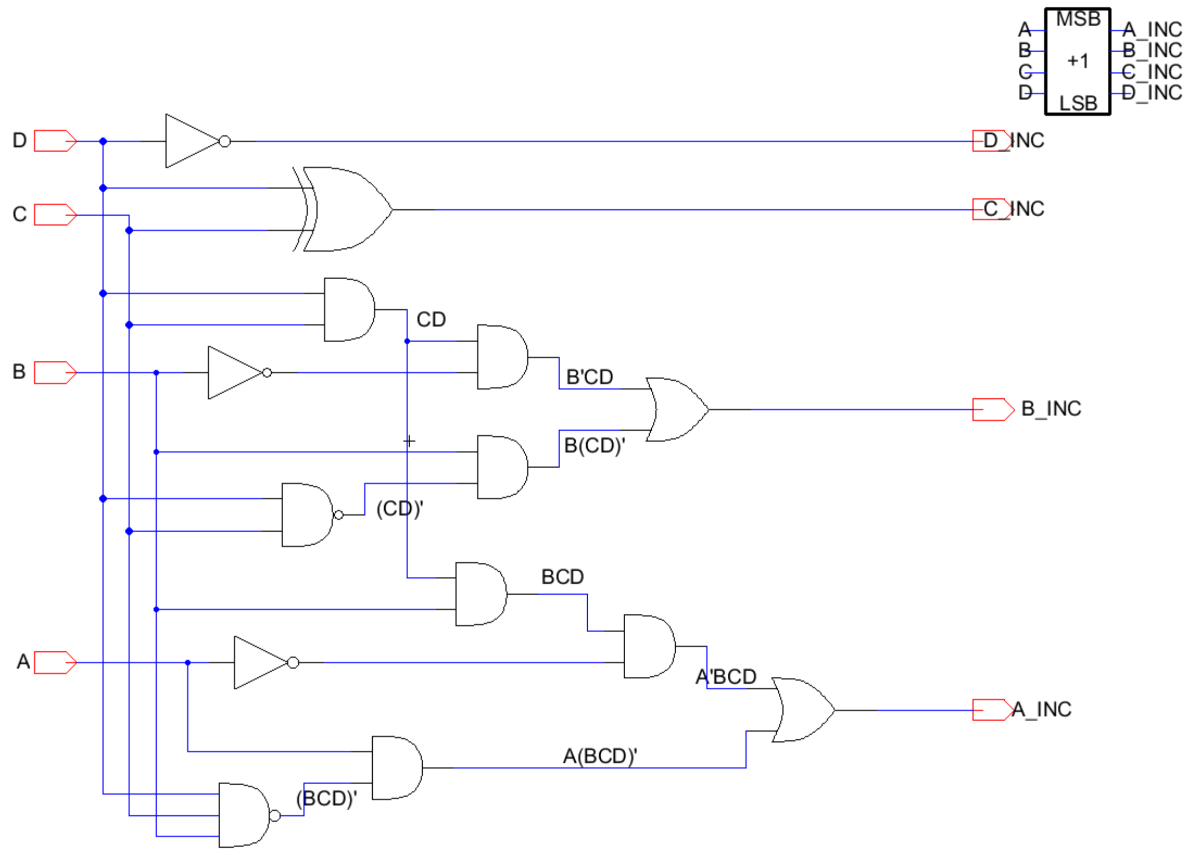
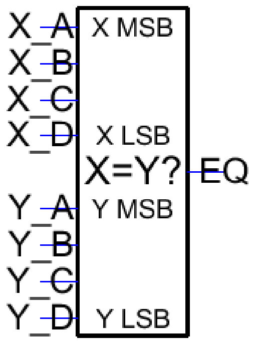


Figure 19: Incrementer schematic

Comparator



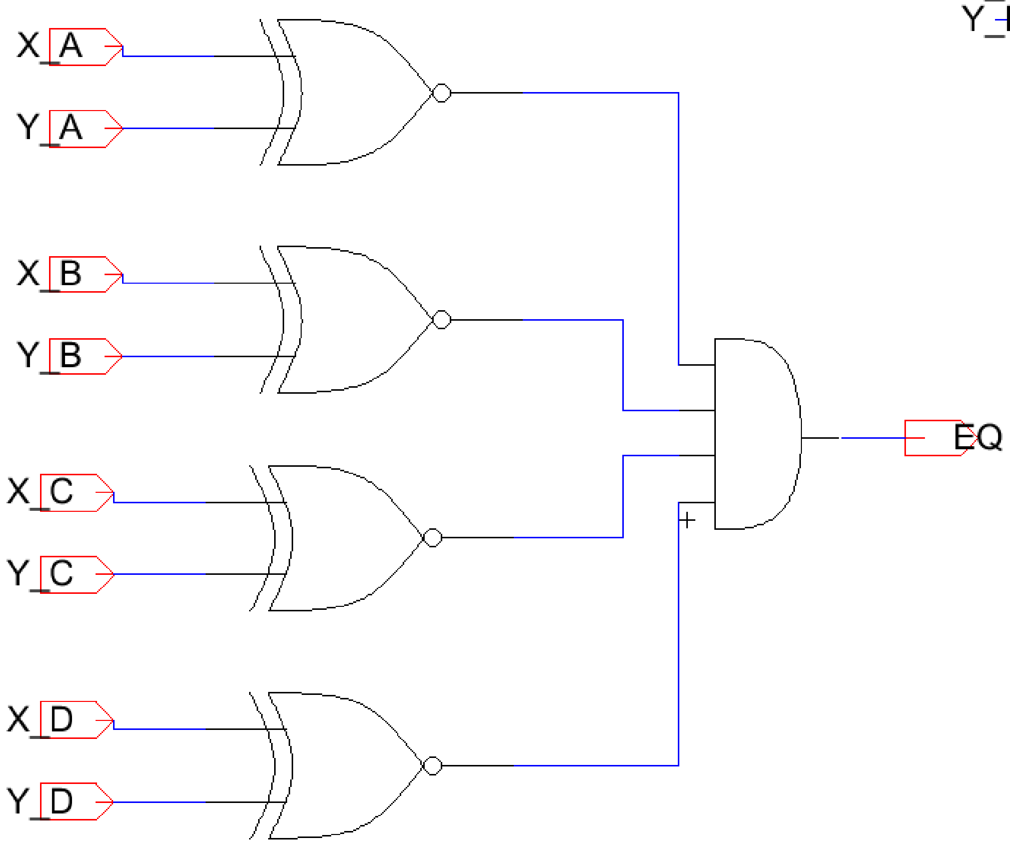


Figure 20: Equality comparator schematic

Mux

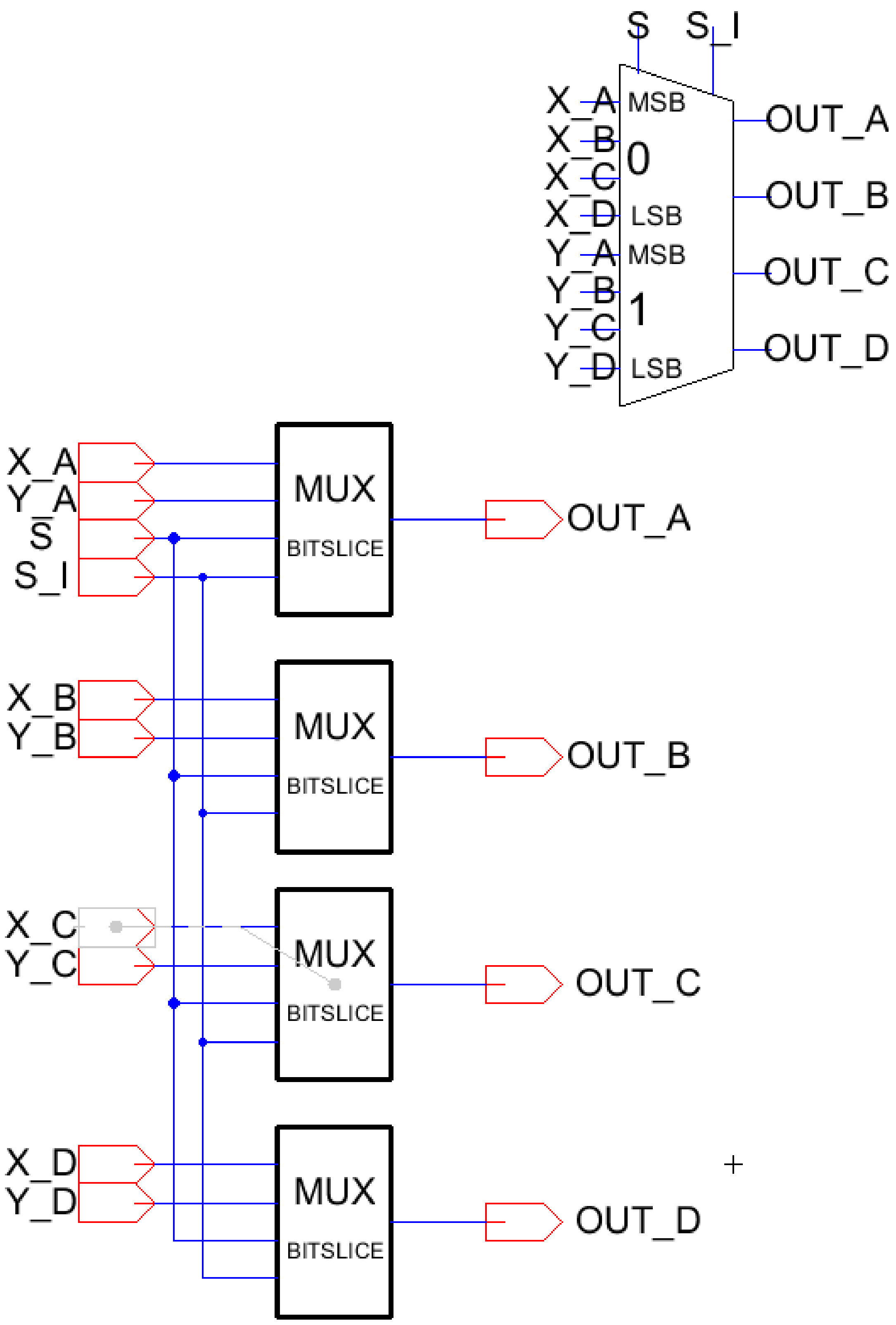


Figure 21: Mux schematic

Mux Bitslice

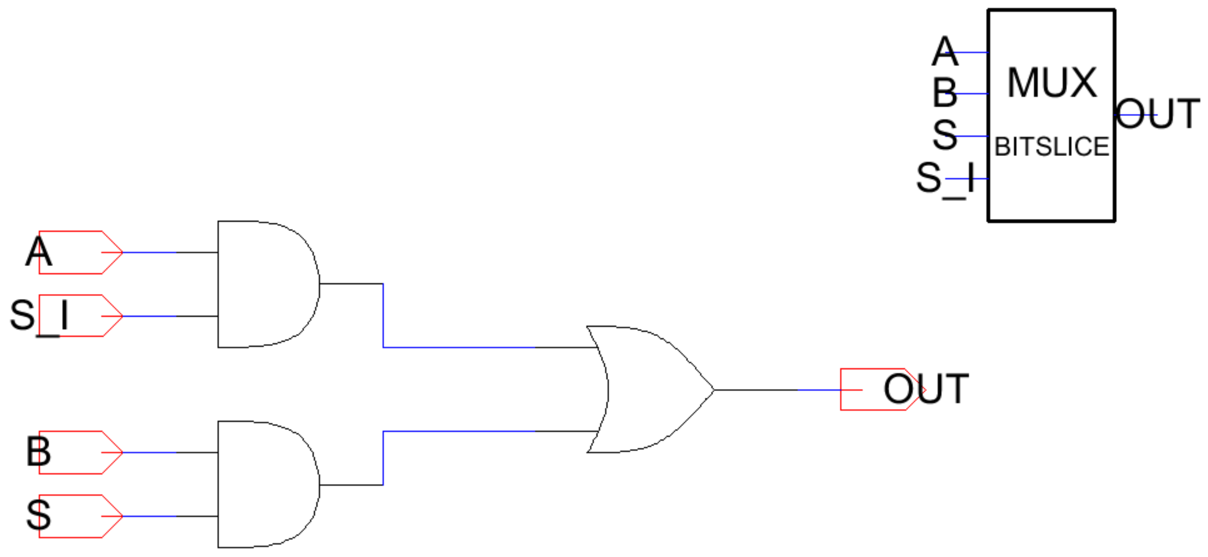


Figure 22: Mux bitslice schematic

Wordline Enable (WL_EN)

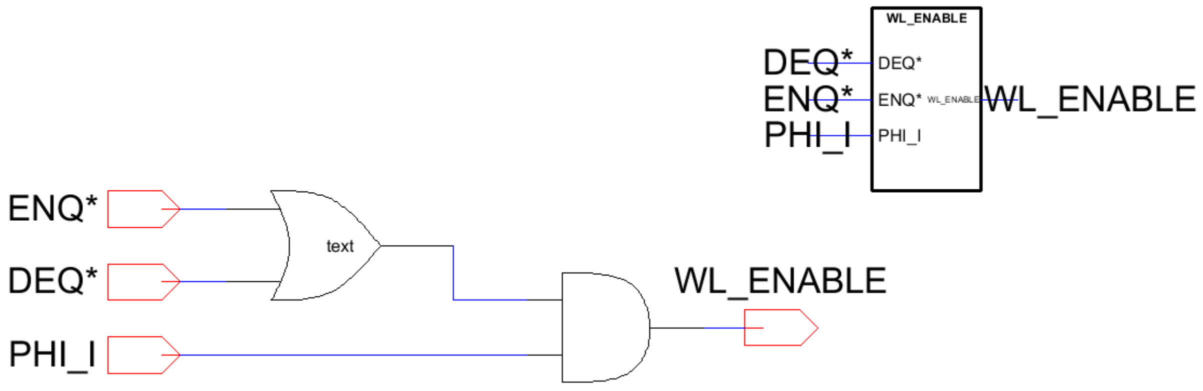


Figure 23: Word line enable generator schematic

ENQ*/DEQ* Generator

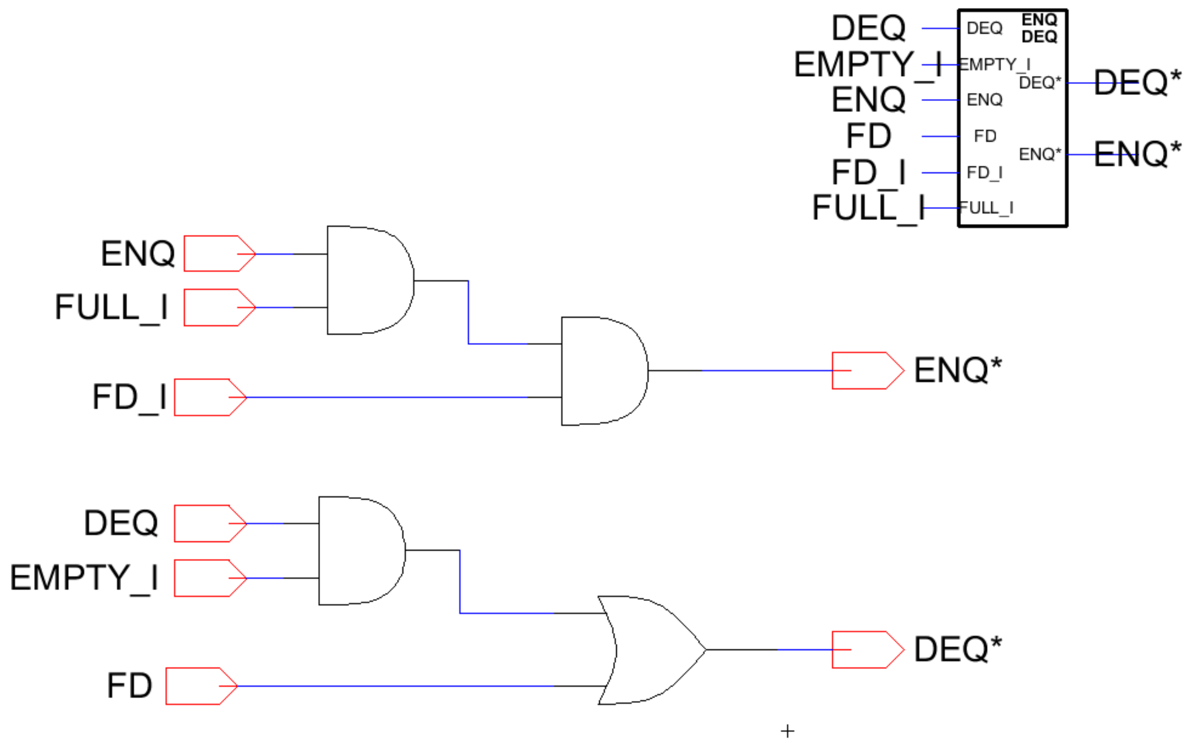


Figure 24: ENQ*/DEQ* generator schematic

4-bit Bitline Precharger

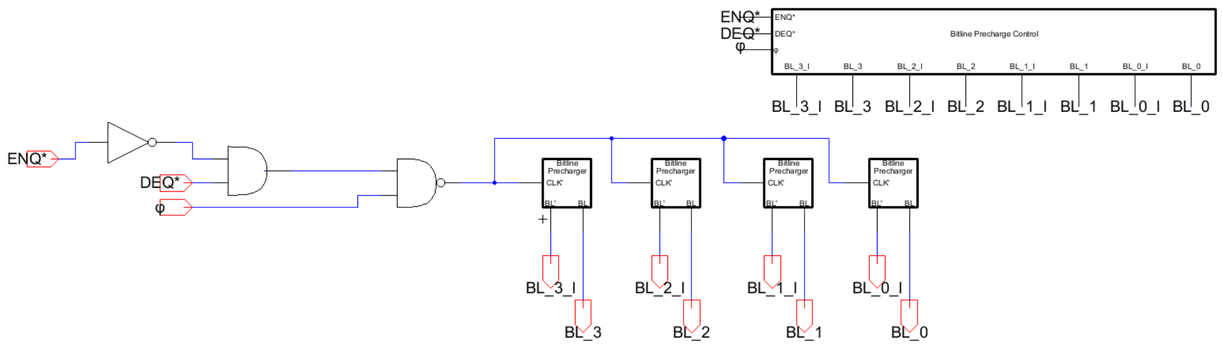


Figure 25: 4-bit bitline precharger schematic

1-bit Bitline Precharger

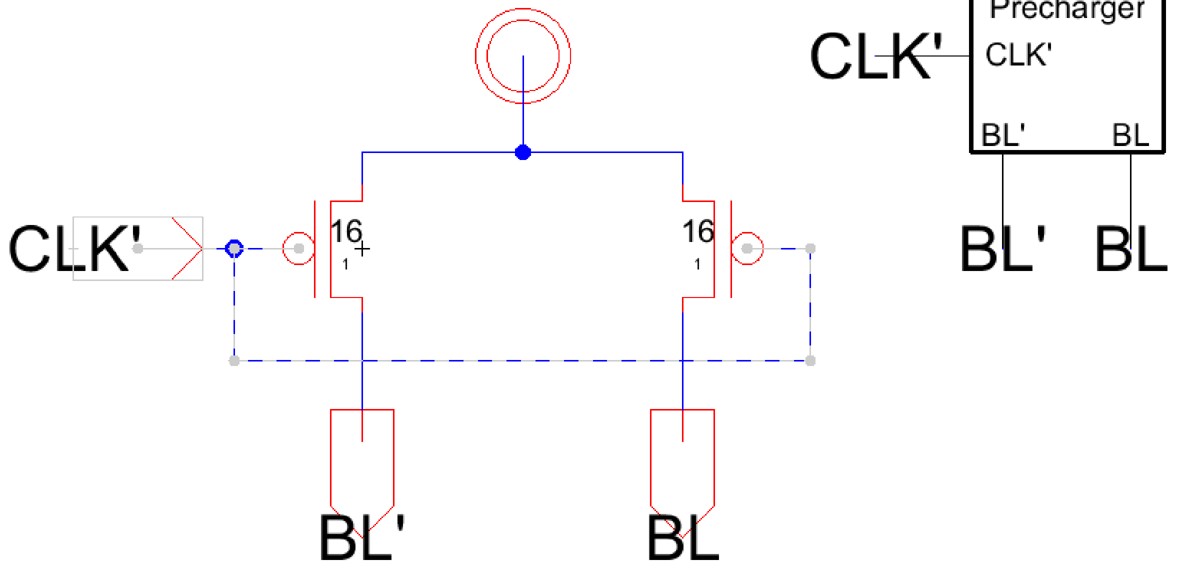


Figure 26: 1-bit bitline precharger schematic

Bitline Driver

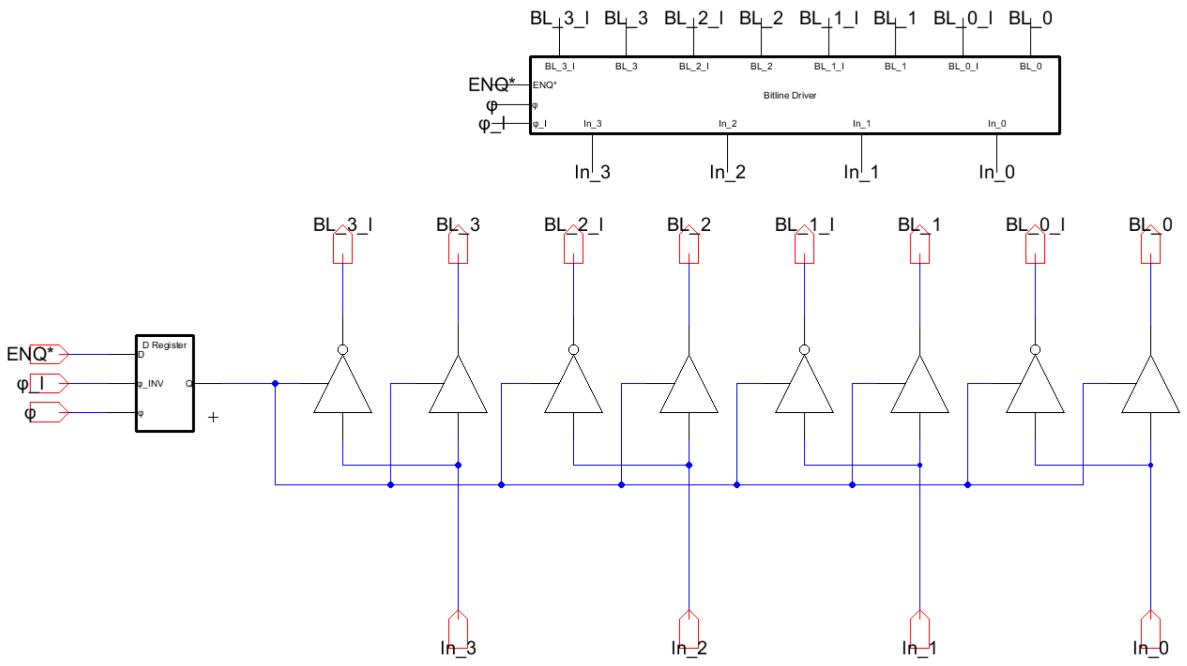


Figure 27: Bitline driver schematic

Inverter

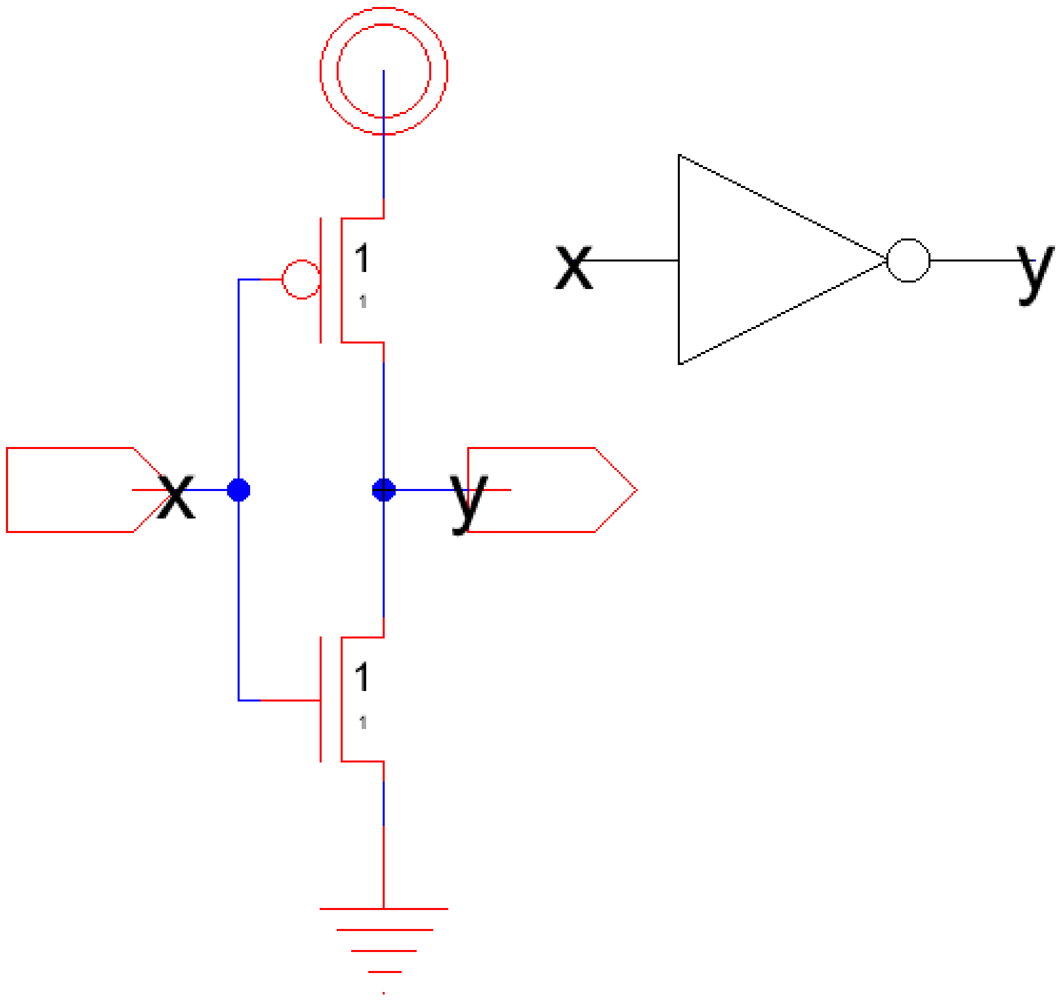


Figure 28: Inverter schematic

NAND2

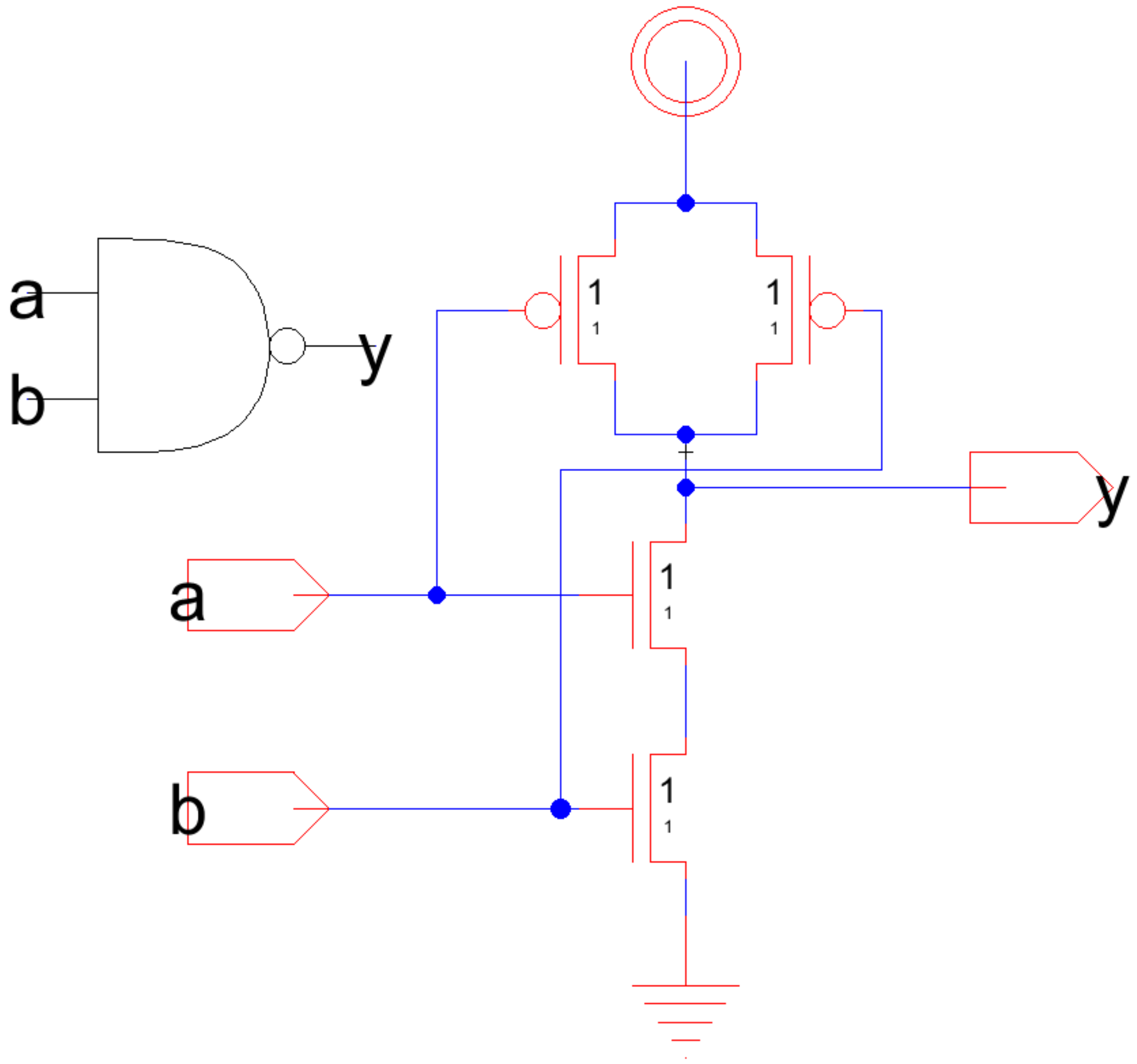


Figure 29: NAND2 gate schematic

AND2

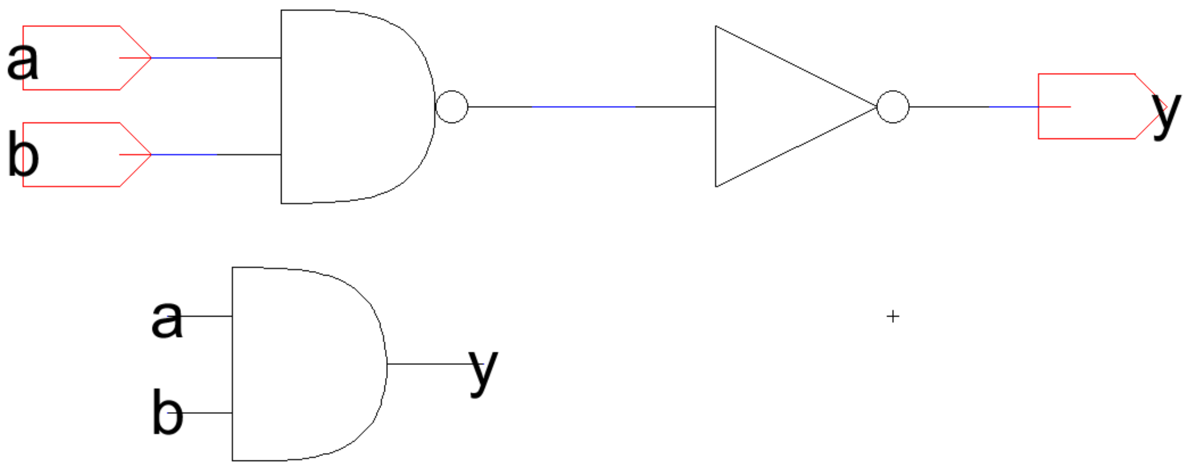


Figure 30: AND2 gate schematic

AND4

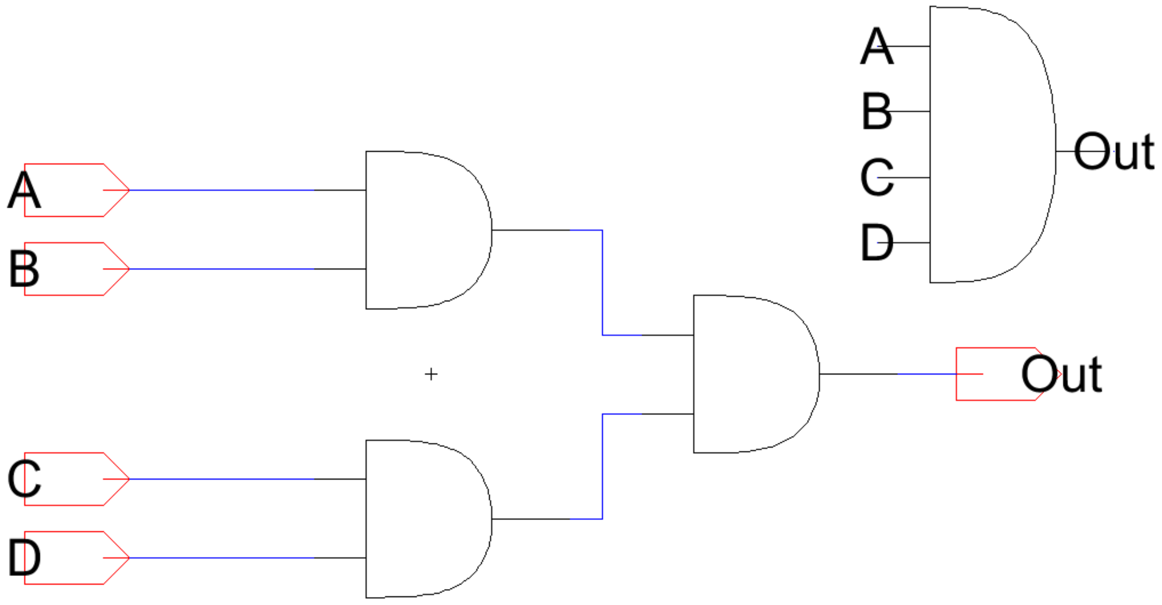


Figure 31: AND4 gate schematic

NOR2

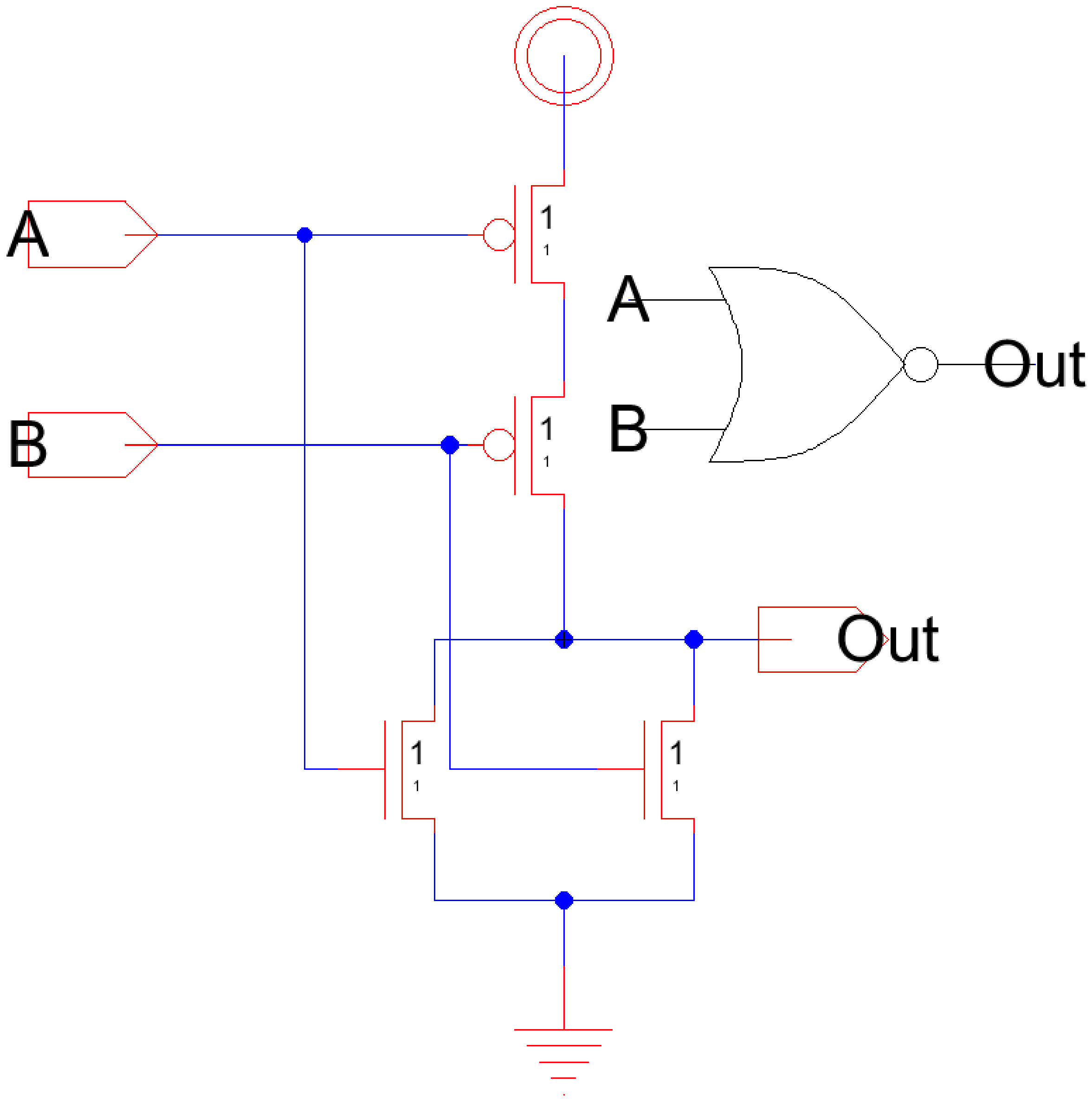


Figure 32: NOR2 gate schematic

Decoder

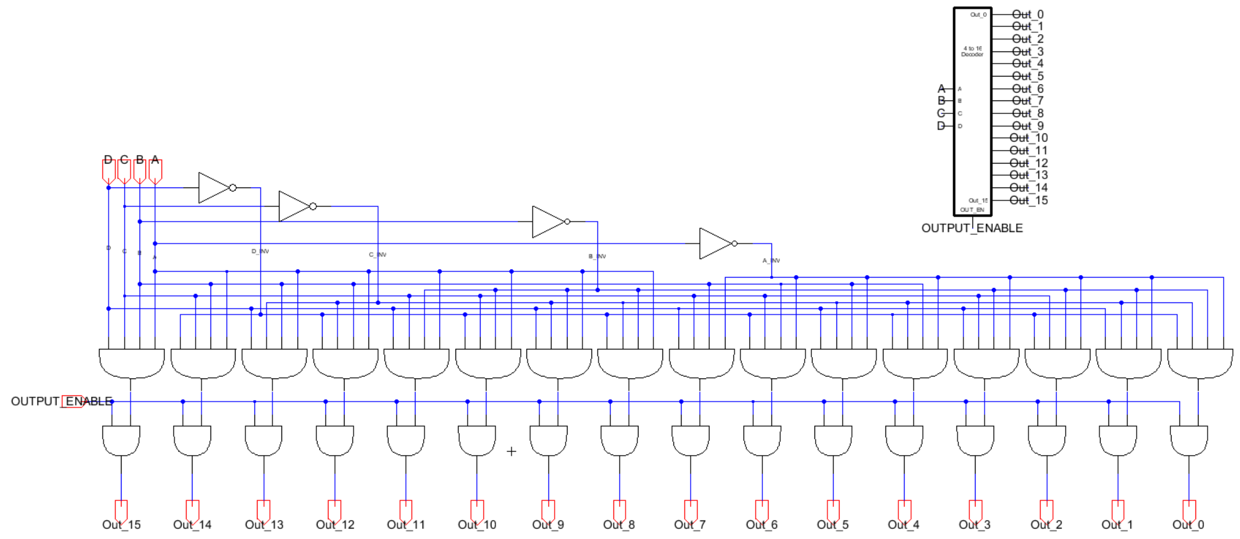


Figure 33: 4-to-16 decoder schematic

Tri-State Buffer

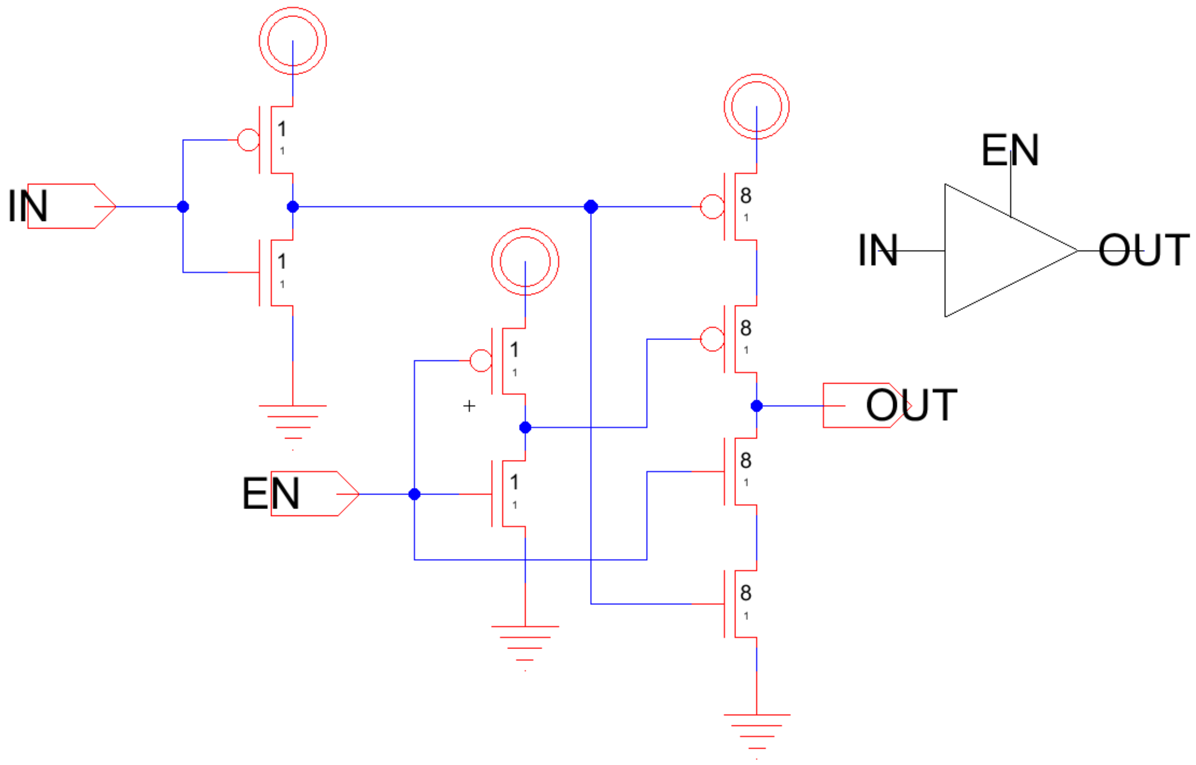


Figure 34: Tri-state buffer schematic

Tri-State Inverter

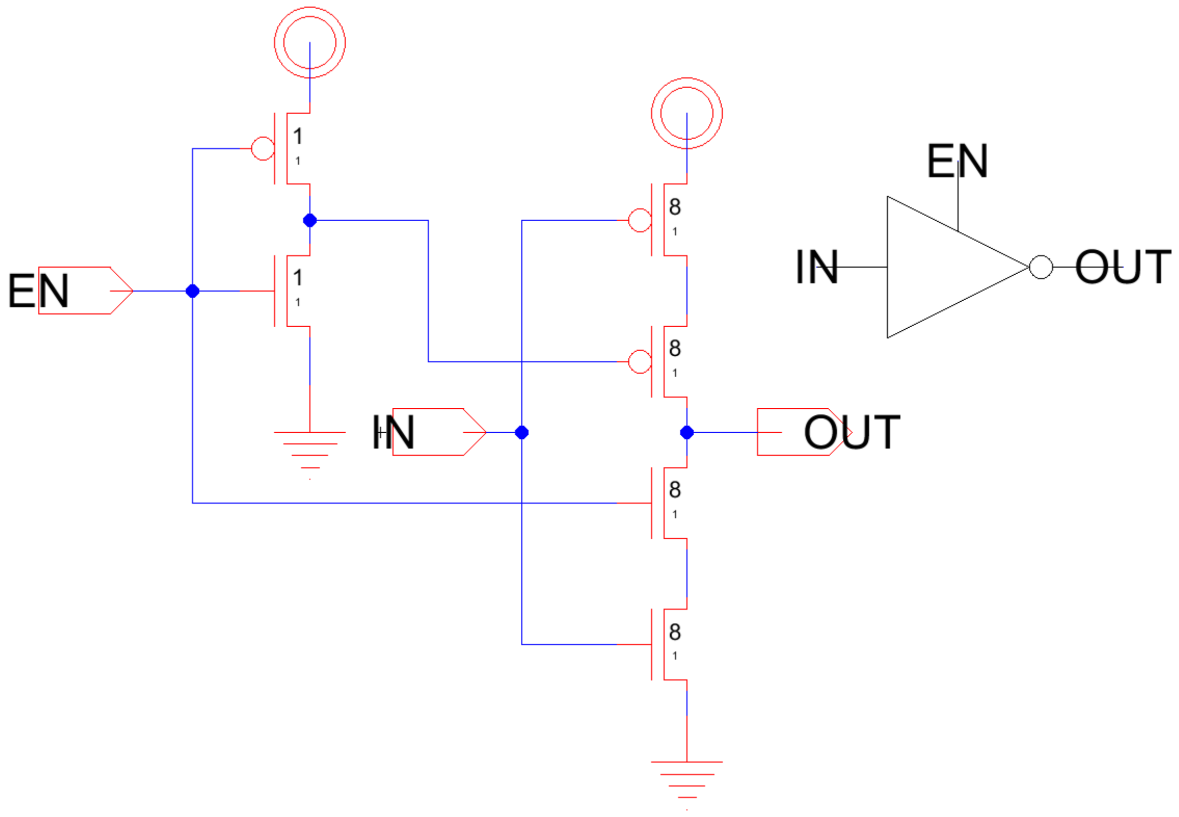


Figure 35: Tri-state inverter schematic

Timing of Key Signals

The critical path that we reasoned was from the user asserting dequeue or enqueue to the bit-line values/memory cell contents settling at the correct final values. (All delays were measured from the rising edge of the clock cycle, when the enqueue/dequeue is latched, to Out_0.)

Dequeue

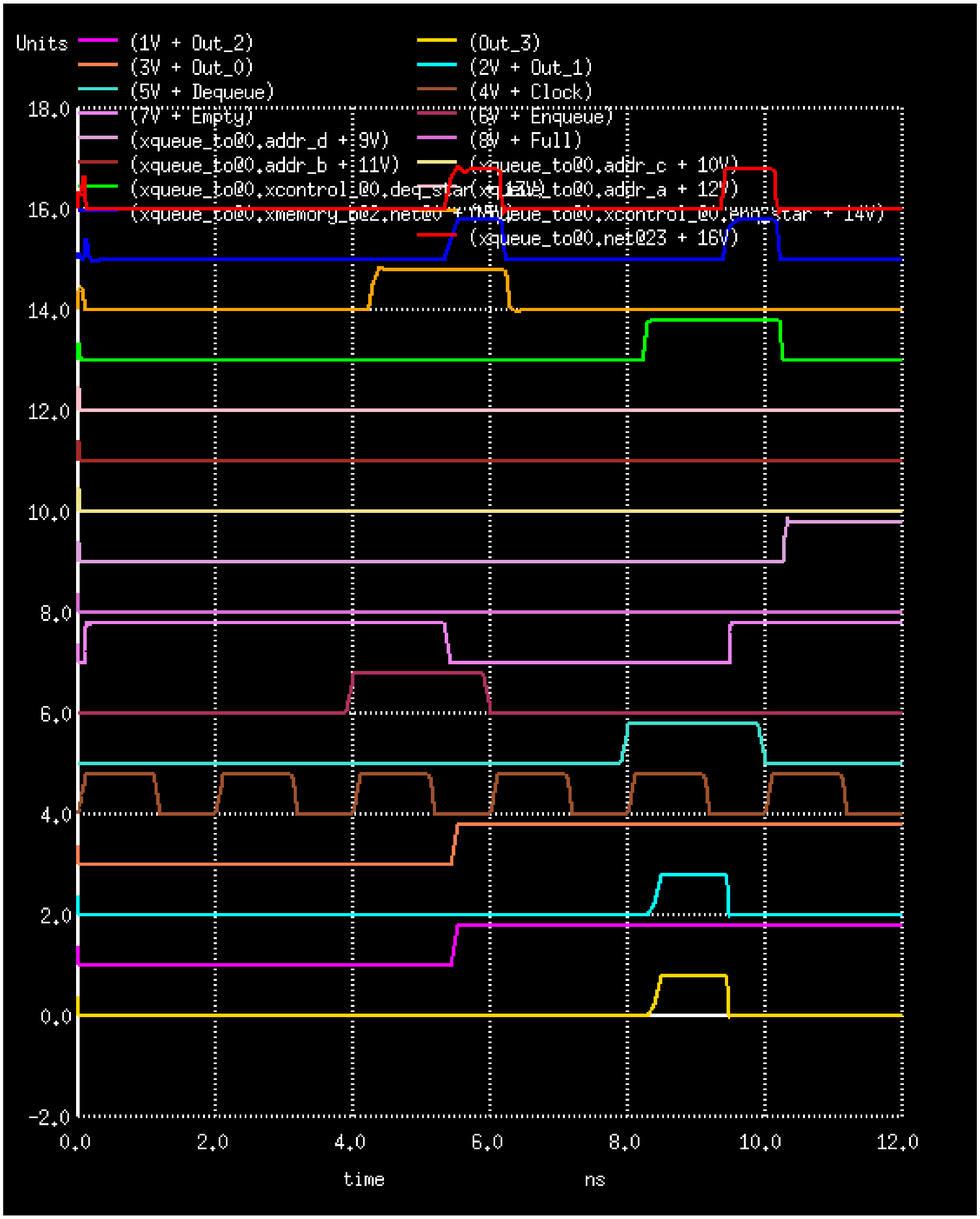


Figure 36: Queue toplevel simple dequeue

Dequeue Delay Results:

```
xqueue_to@0.net@23 is WL_ENABLE (topmost signal, red)
xqueue_to@0.xmemory_b@2.net@0 is WL's Out_0 (call this WL_Out_0)
    (2nd topmost signal, blue)

delay between clock and deq*
2.01613 x 10^-10 s

delay between deq* and WL_Out_0
1.17101 x 10^-9 s

delay between deq* and WL_ENABLE
1.12029 x 10^-9 s

delay between WL_Out_0 and Out_1 (top-level output)
3.44828 x 10^-11 s

delay between WL_ENABLE and Out_1 (top-level output)
7.98851 x 10 ^-11 s
```

Enqueue

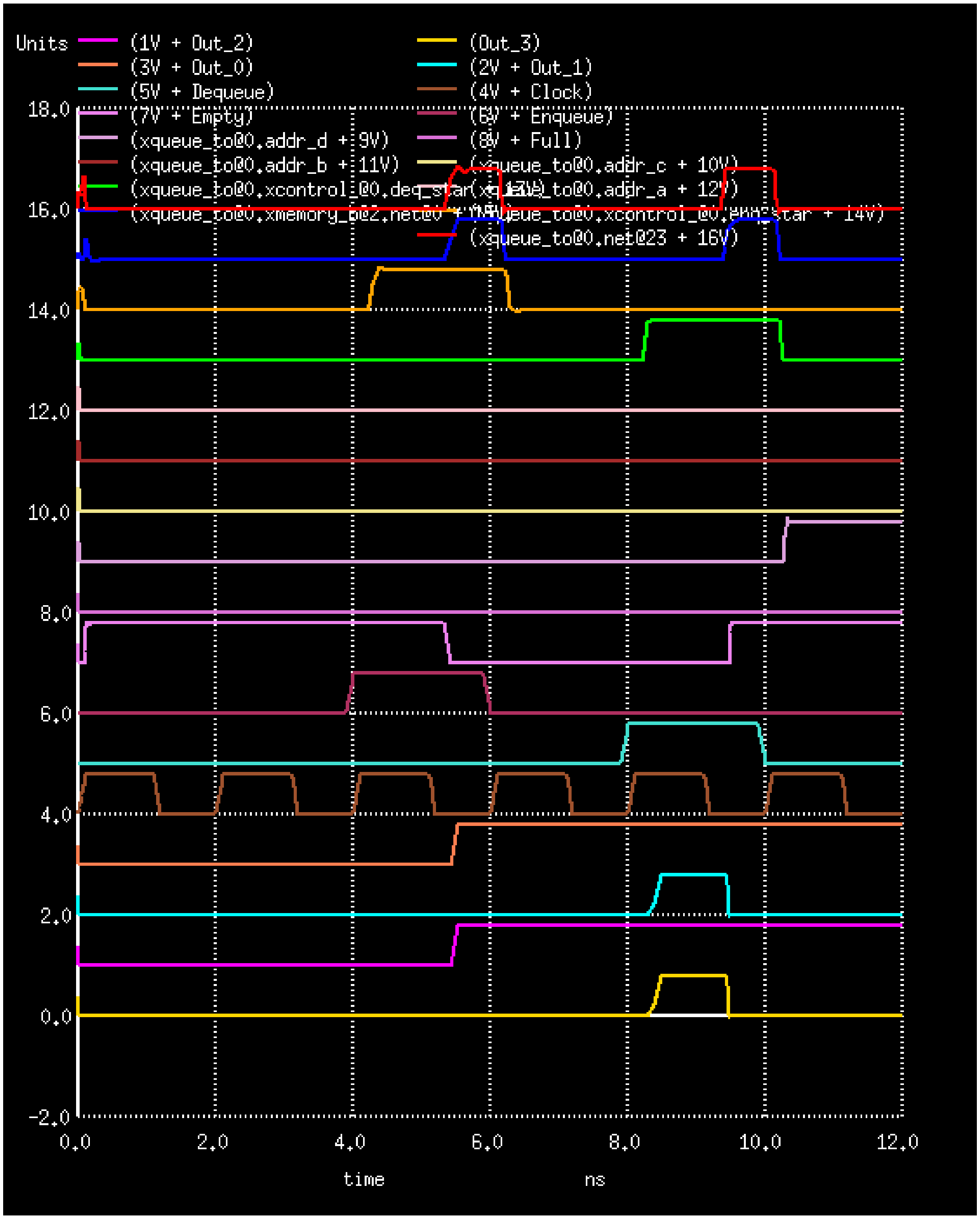


Figure 37: Queue toplevel simple dequeue

```
xqueue_to@0.net@23 is WL_ENABLE (topmost signal, red)
xqueue_to@0.xmemory_b@2.net@0 is WL's Out_0 (call this WL_Out_0)
    (2nd topmost signal, blue)

delay between clock and enq*
3.3587 x 10^-10 s

delay between enq* and WL_Out_0
1.16 x 10^-9 s

delay between enq* and WL_ENABLE
1.12308 x 10^-9 s

delay between WL_Out_0 and Out_0 (top-level output)
5.58559 x 10^-11 s

delay between WL_ENABLE and Out_0 (top-level output)
7.65766 x 10 ^-11 s
```

Note: NGSPICE was not behaving properly when we tried to plot the node names inside cells. It kept changing the node names for no apparent reason, and we became very frustrated at times. The information we plotted above was done to the best of our ability.

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.)

Optimization of Energy

One of the ways in which we optimized the energy was by minimizing the number of transistors throughout the design. For example, in the 4-bit bitline precharge module, the individual bitline precharge modules require CLK to be asserted low in order for that bitline to be precharged, since the precharging is controlled by PMOS transistors. The logic function for precharging is that ENQ* is not asserted, DEQ* is, and the clock is high. We originally had it so that all of these values were being ANDed and then inverted, but we reduced this just to a NAND gate because the inverter was redundant. This reduces static energy usage slightly because there are two fewer transistors without the inverter.

Another way we reduced the energy was by keeping V_{dd} low, at 0.8 V. We thought about increasing V_{dd} to 1 V to reduce the delay, but we reasoned that this was unnecessary because delay was not a specification that we were specifically asked to optimize in this project. Increasing V_{dd} also would lead to more static power dissipation (since V_{dd} is higher), as well as potentially more dynamic power dissipation (since there would be a wider voltage range in which the two transistors in an inverter would be turned on at once).

We also minimized the amount of energy used by keeping transistors sized as reasonably small as possible. For example, in the 6T memory cell, we sized the transistors at the minimum size ($W = 16$) so that read upsets would be avoided and changes on the bit lines would not cause the stored values to be overwritten. (See description of bit cell circuit operation for more detail on why we had to do this.) We also kept all other transistors in the design minimum-sized, except for the bit-line drivers and prechargers (which were needed to overcome the bitline capacitance). Keeping W small reduces the amount of static current drawn and thus reduces the amount of power used.

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

The unit tests were done to ensure that each component operated according to the specifications given in our design above. Most of them do not require a description since many are combinational or simple sequeuntial circuits. If a description is needed, it is provided in the appropriate test.

Clock Generator

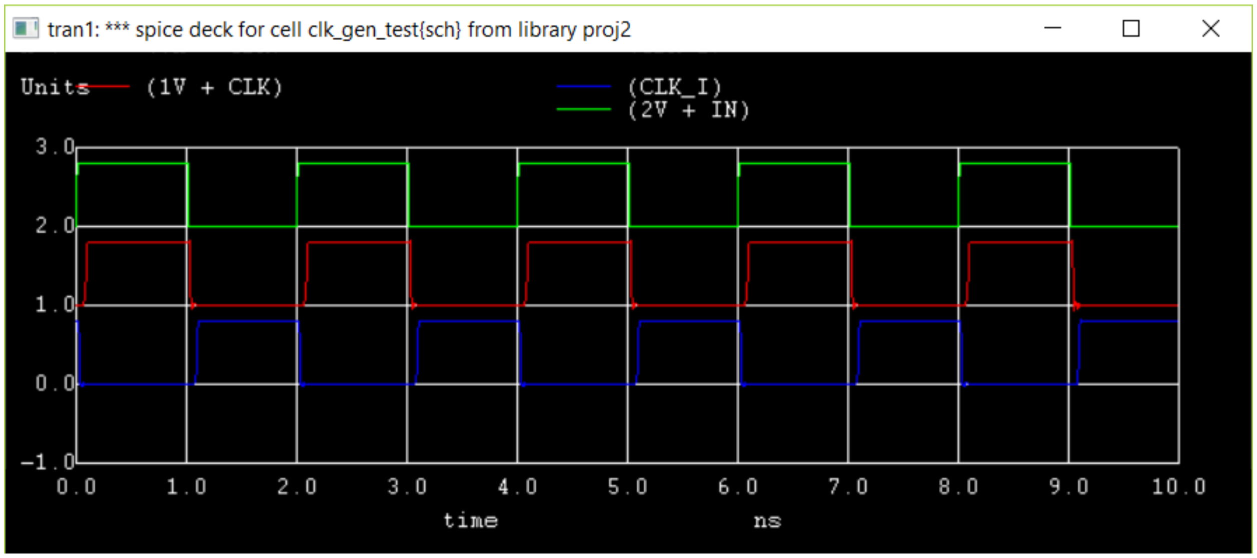


Figure 38: Clock generator test

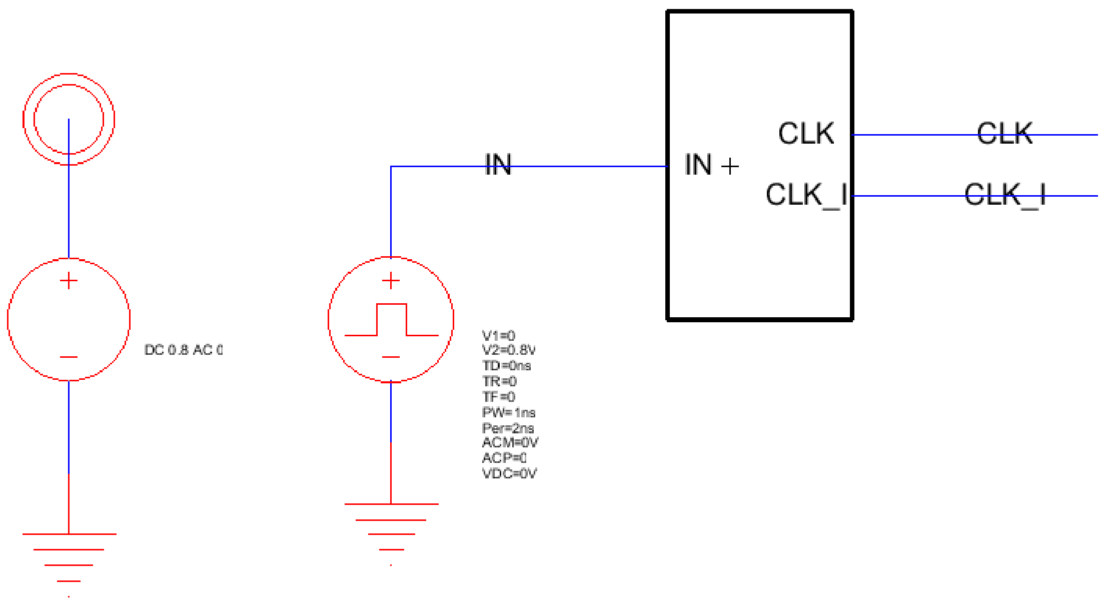


Figure 39: Clock generator test schematic

Comparator

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

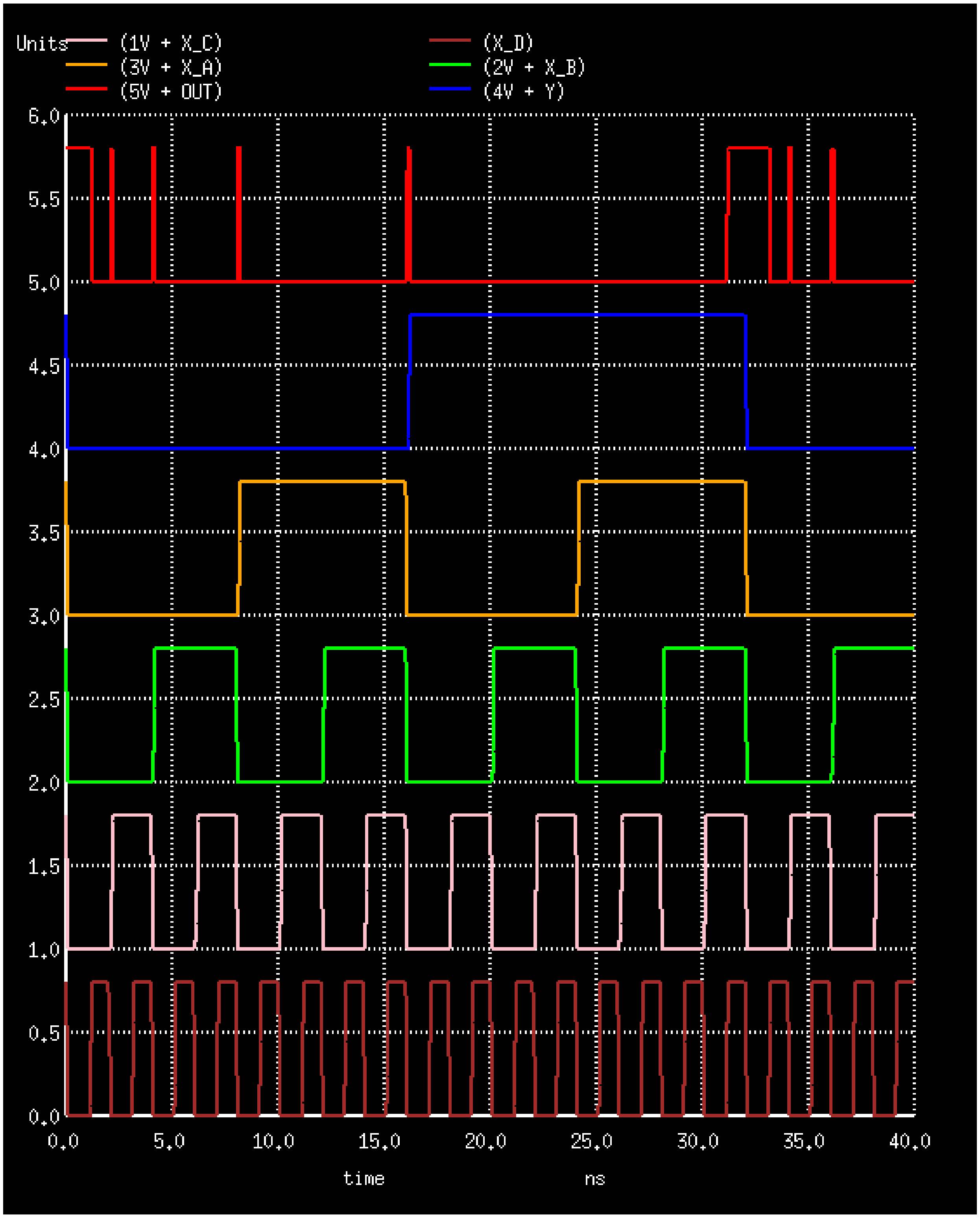


Figure 40: Comparator test, different

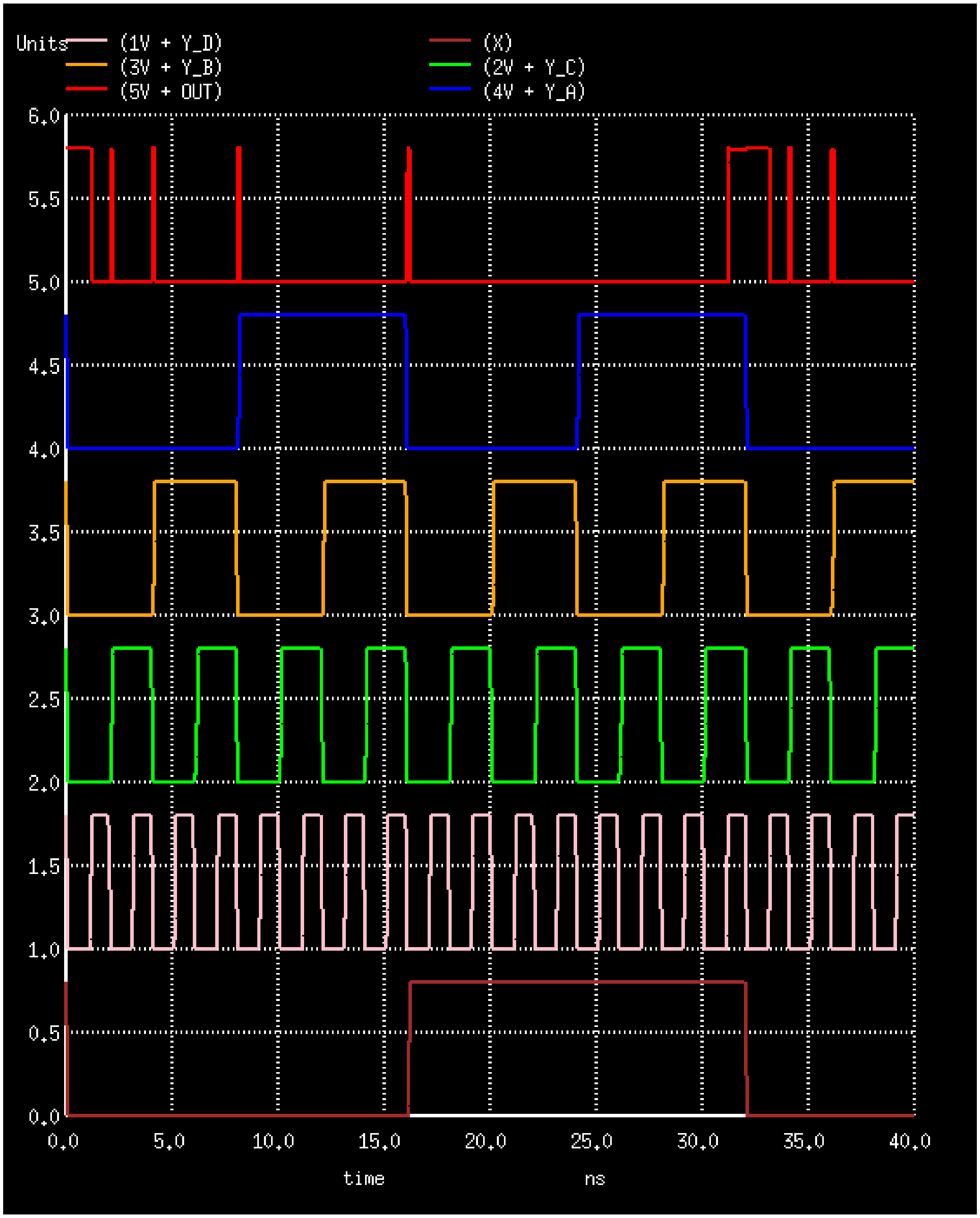


Figure 41: Comparator test, different and swapped

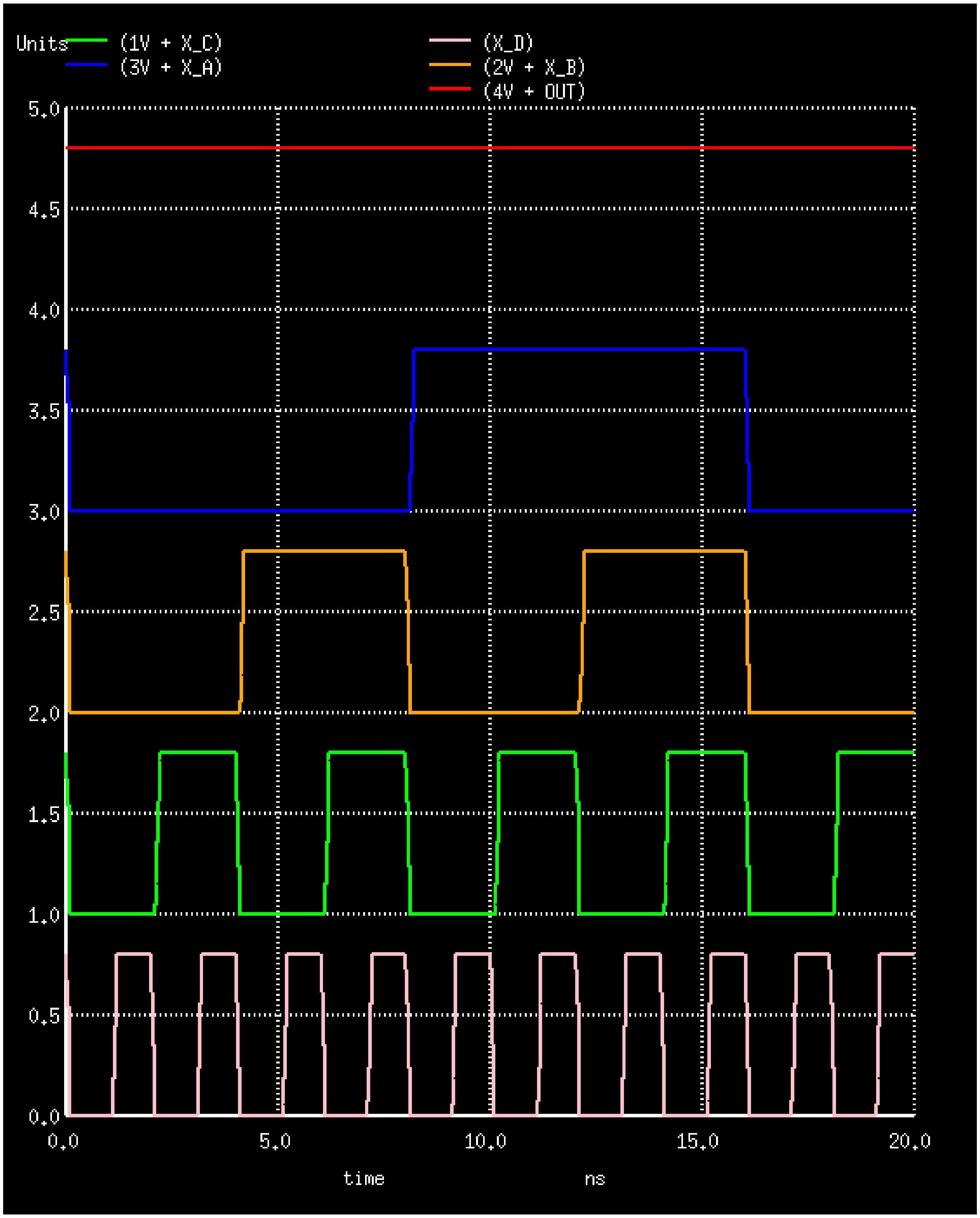


Figure 42: Comparator test, same

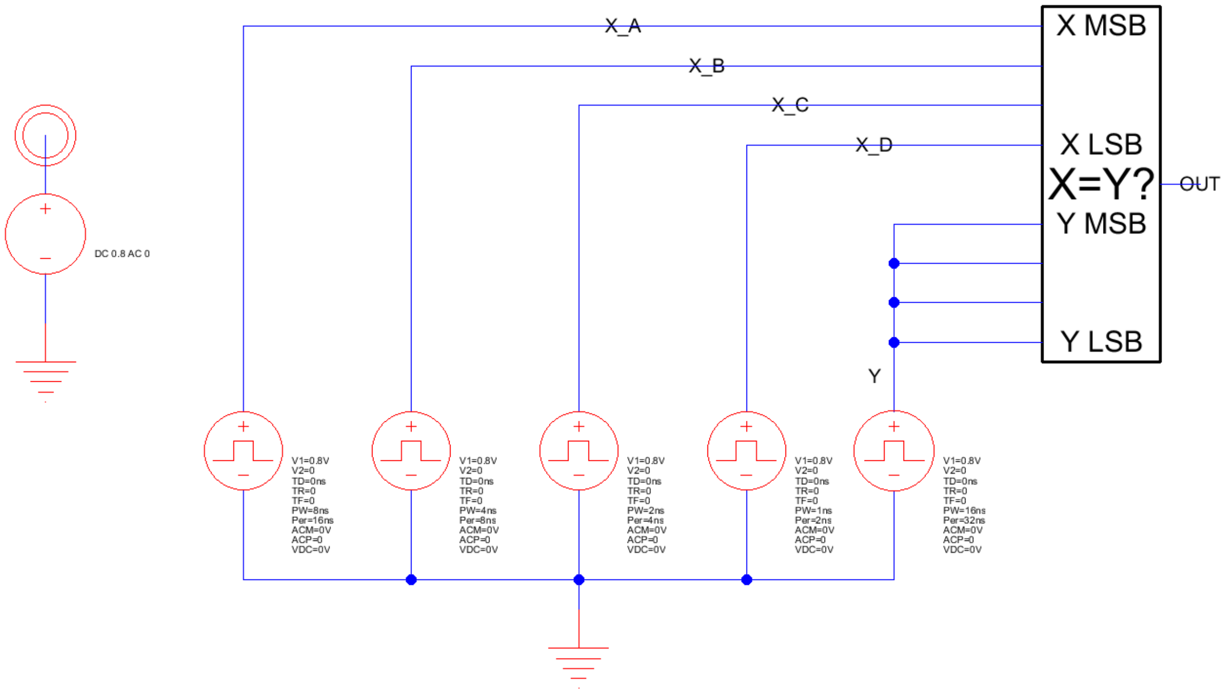

X_A
X_B
X_C
X_D
X MSB
X LSB
X=Y?
OUT
Y MSB
Y LSB
Y
DC 0.8 AC 0
V1=0.8V V2=0 TD=0ns TR=0 TF=0 PW=8ns Per=16ns ACM=0V ACP=0 VDC=0V
V1=0.8V V2=0 TD=0ns TR=0 TF=0 PW=4ns Per=8ns ACM=0V ACP=0 VDC=0V
V1=0.8V V2=0 TD=0ns TR=0 TF=0 PW=2ns Per=4ns ACM=0V ACP=0 VDC=0V
V1=0.8V V2=0 TD=0ns TR=0 TF=0 PW=1ns Per=2ns ACM=0V ACP=0 VDC=0V
V1=0.8V V2=0 TD=0ns TR=0 TF=0 PW=16ns Per=32ns ACM=0V ACP=0 VDC=0V

Figure 43: Comparator test schematic

D Latch

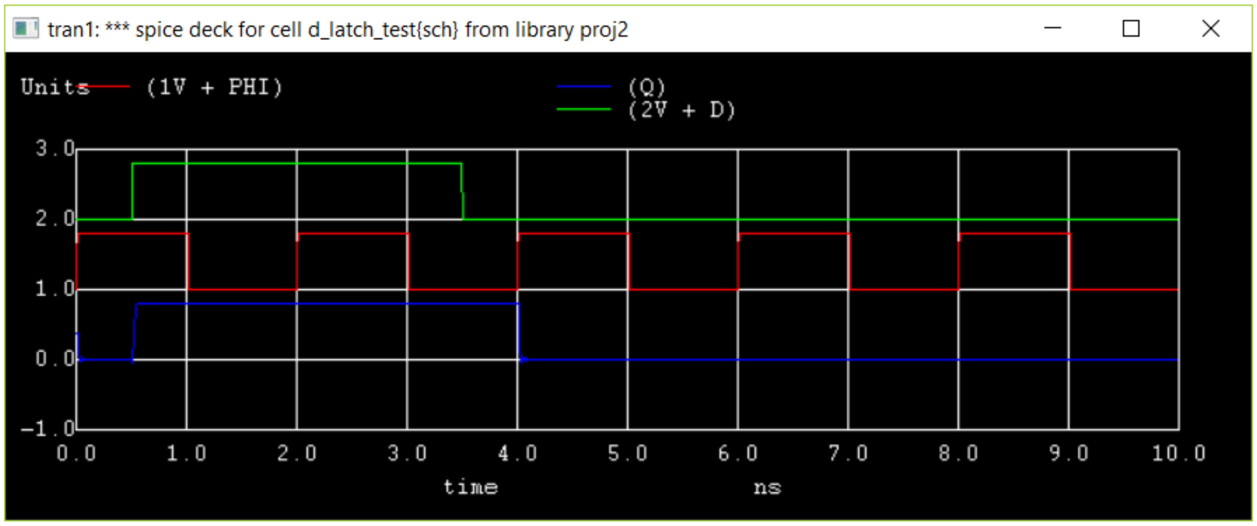


Figure 44: D Latch Test

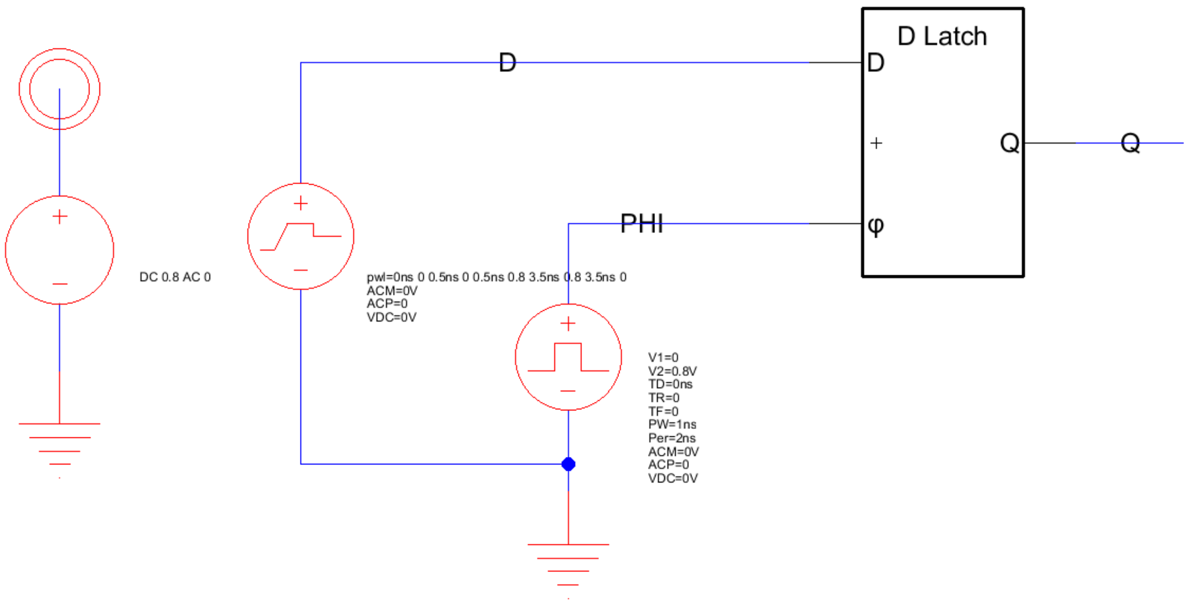


Figure 45: D latch test schematic

D Register

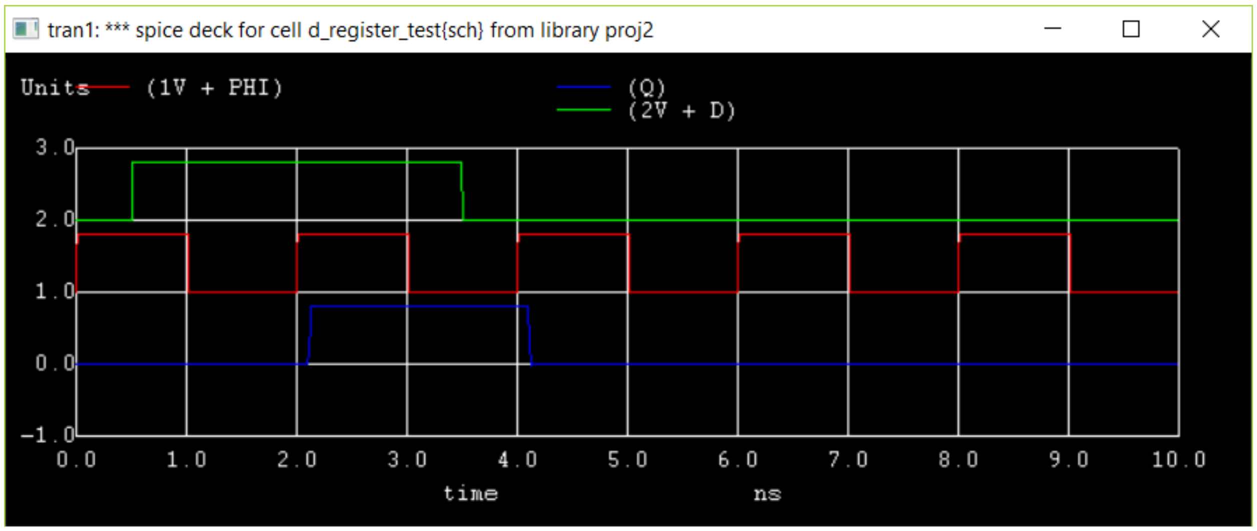


Figure 46: D Register Test

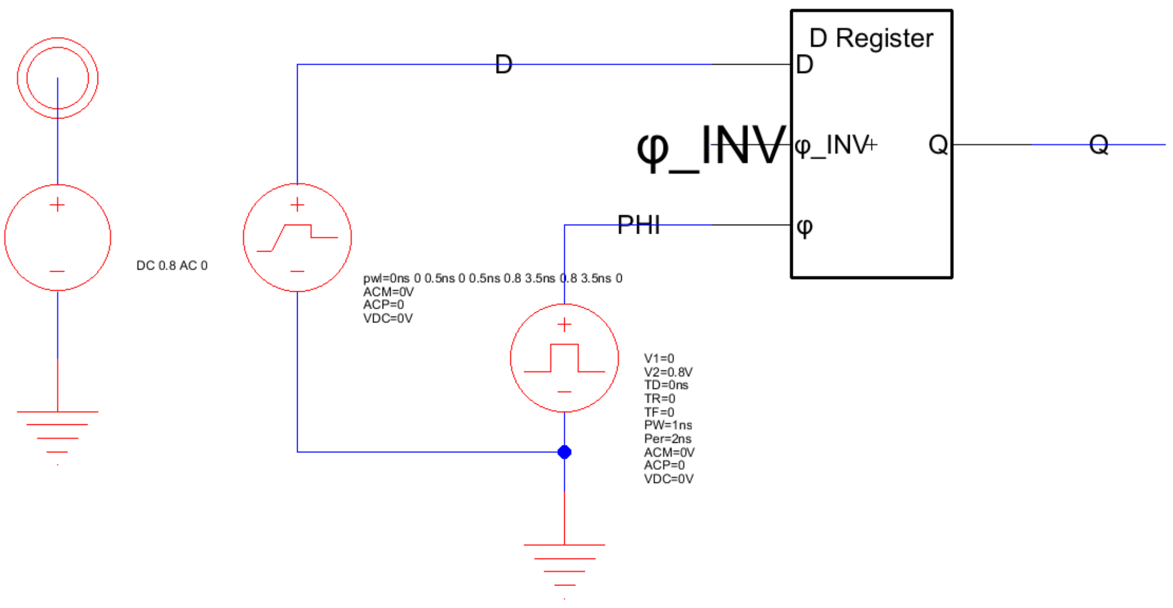


Figure 47: D register test schematic

ENQ*/DEQ* Module

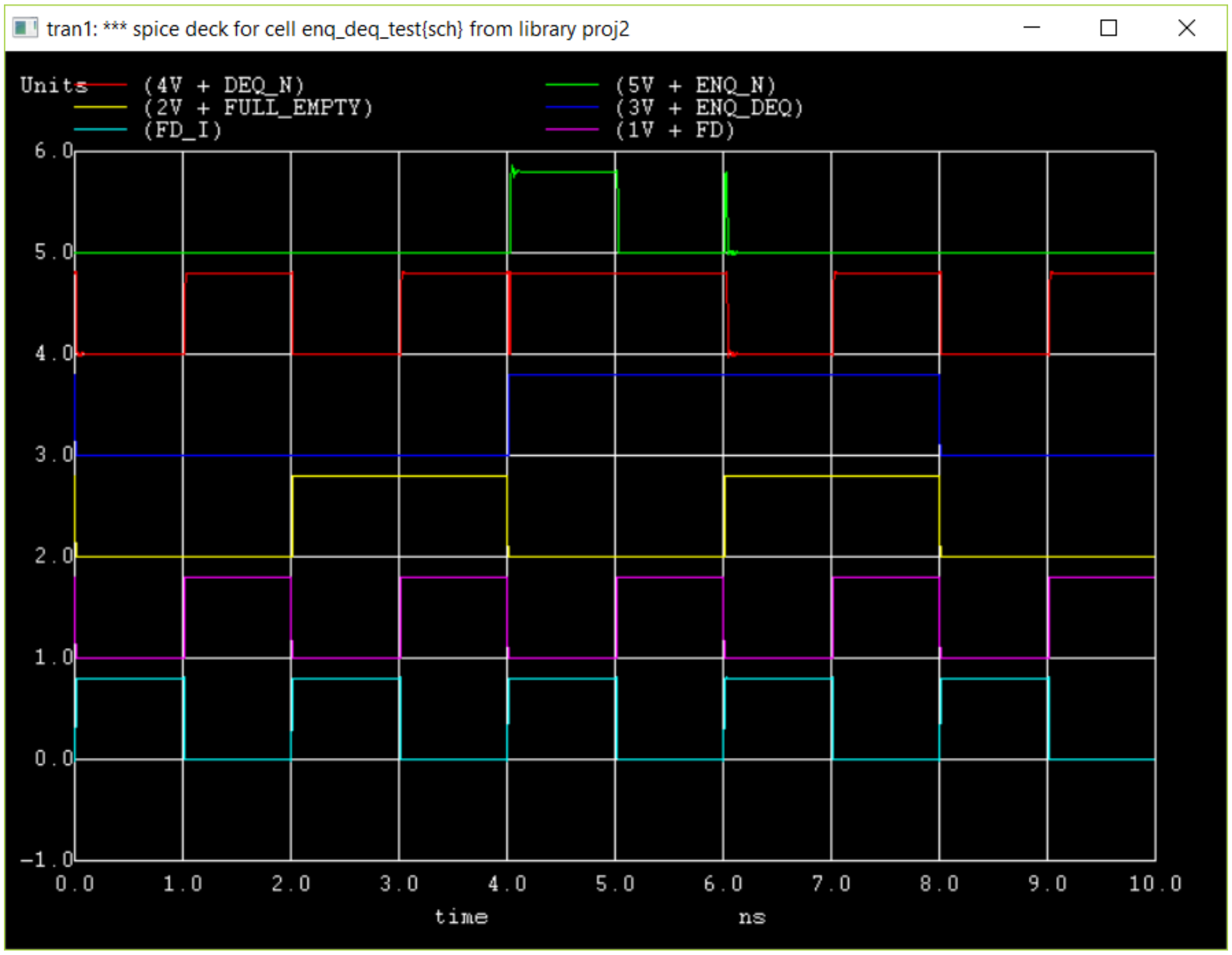


Figure 48: ENQ*/DEQ* Module Test

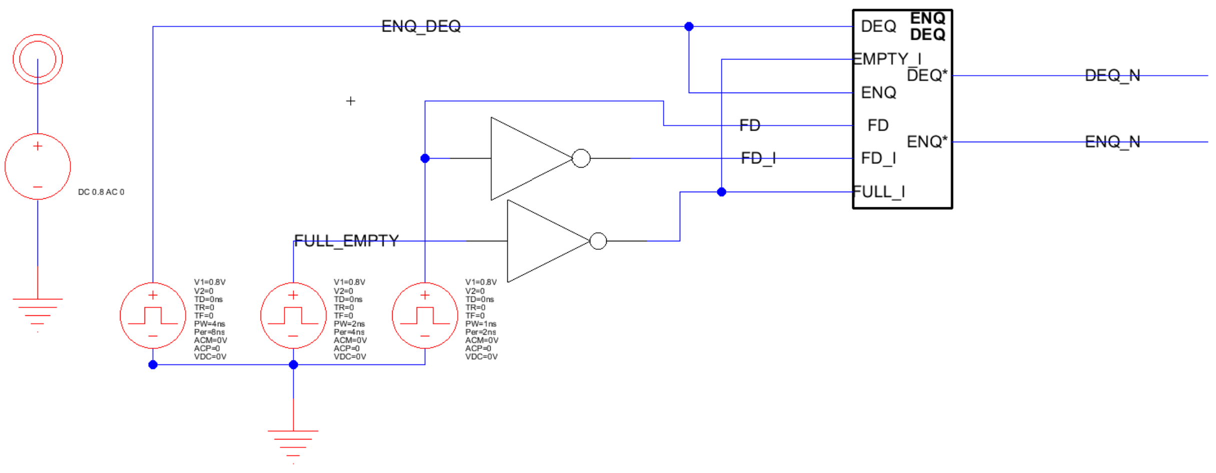


Figure 49: ENQ*/DEQ* test schematic

Incrementer

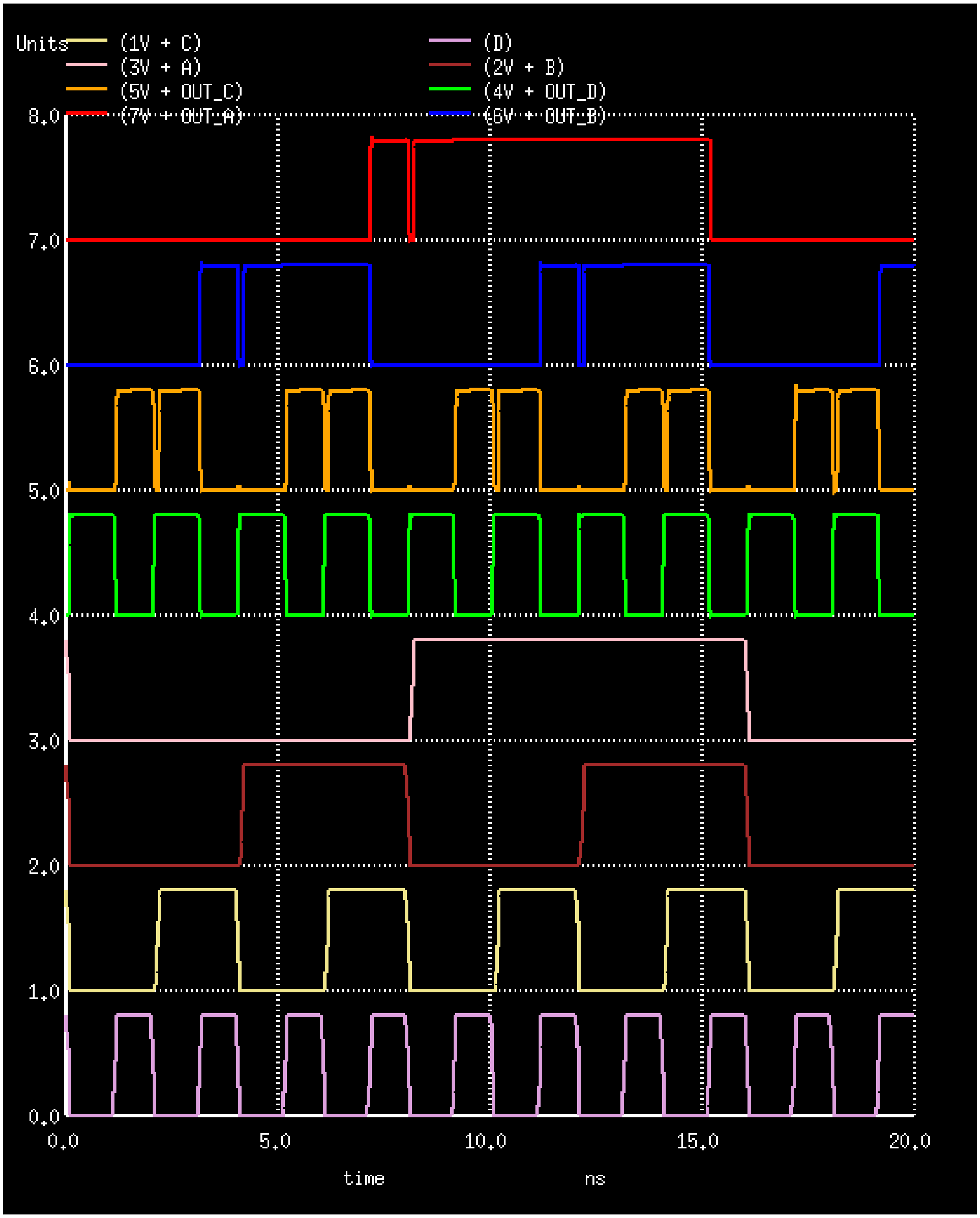


Figure 50: Incrementer Test

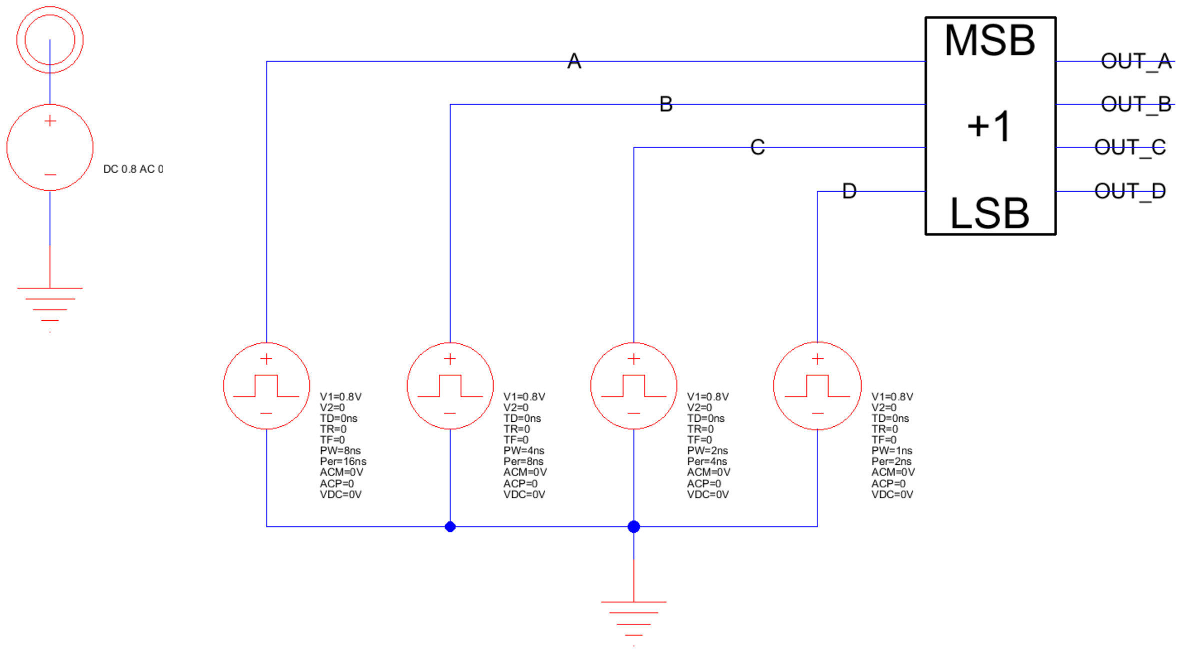

DC 0.8 AC 0
A
B
C
D
MSB
+1
LSB
OUT_A
OUT_B
OUT_C
OUT_D
V1=0.8V V2=0 TD=0ns TR=0 TF=0 PW=8ns Per=16ns ACM=0V ACP=0 VDC=0V
V1=0.8V V2=0 TD=0ns TR=0 TF=0 PW=4ns Per=8ns ACM=0V ACP=0 VDC=0V
V1=0.8V V2=0 TD=0ns TR=0 TF=0 PW=2ns Per=4ns ACM=0V ACP=0 VDC=0V
V1=0.8V V2=0 TD=0ns TR=0 TF=0 PW=1ns Per=2ns ACM=0V ACP=0 VDC=0V

Figure 51: 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)).

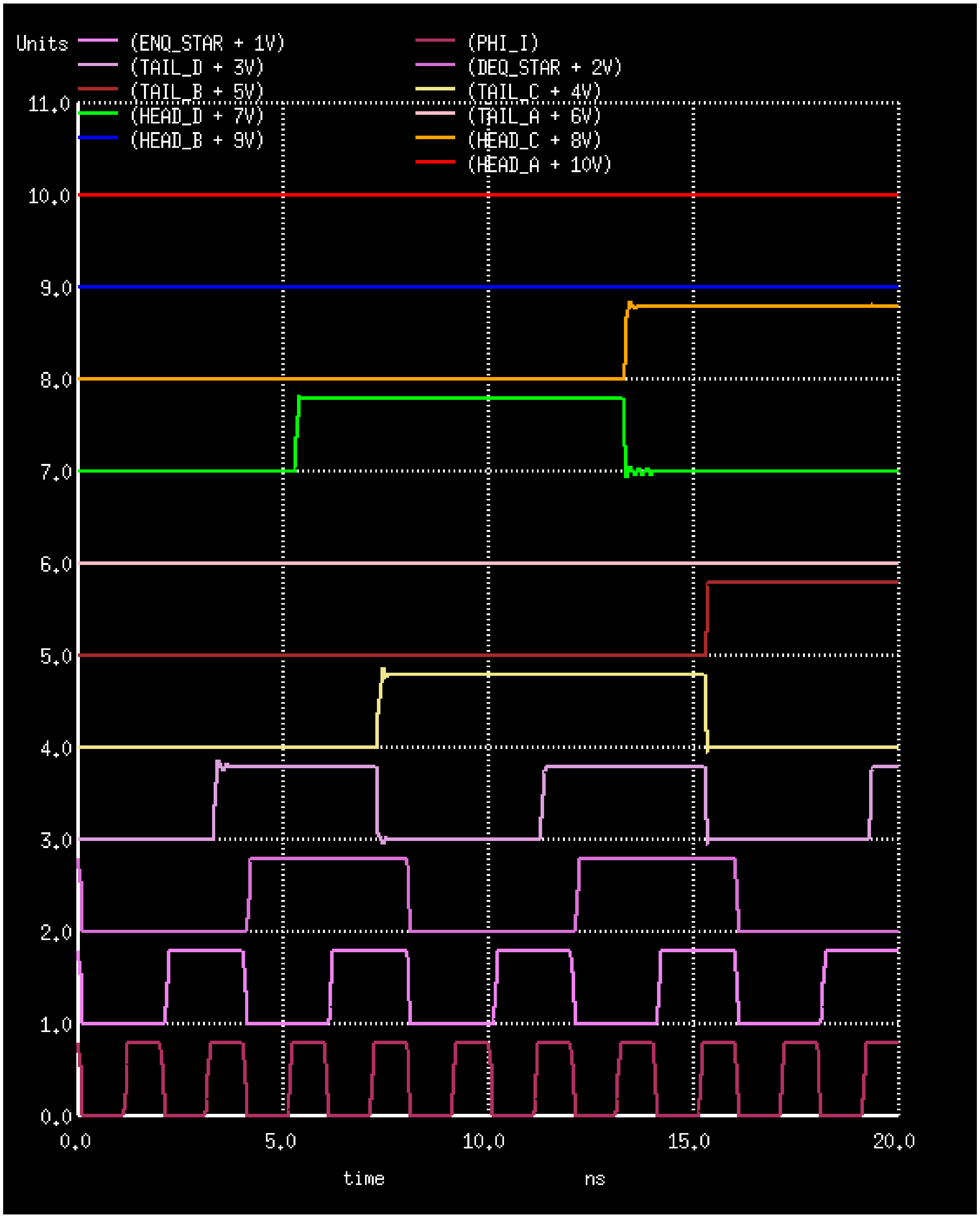


Figure 52: Pointers test, proper increment

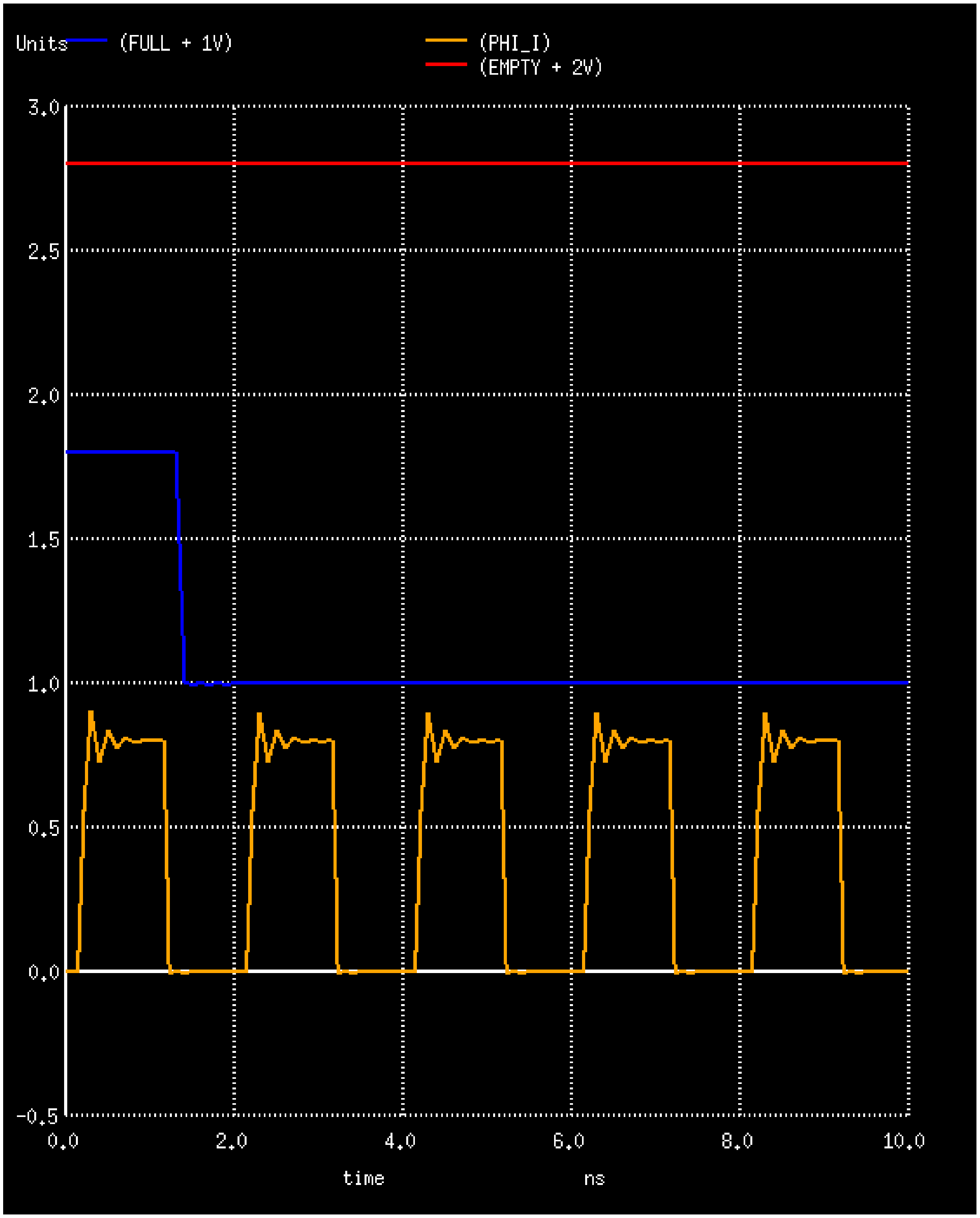


Figure 53: Pointers test, no incrementing

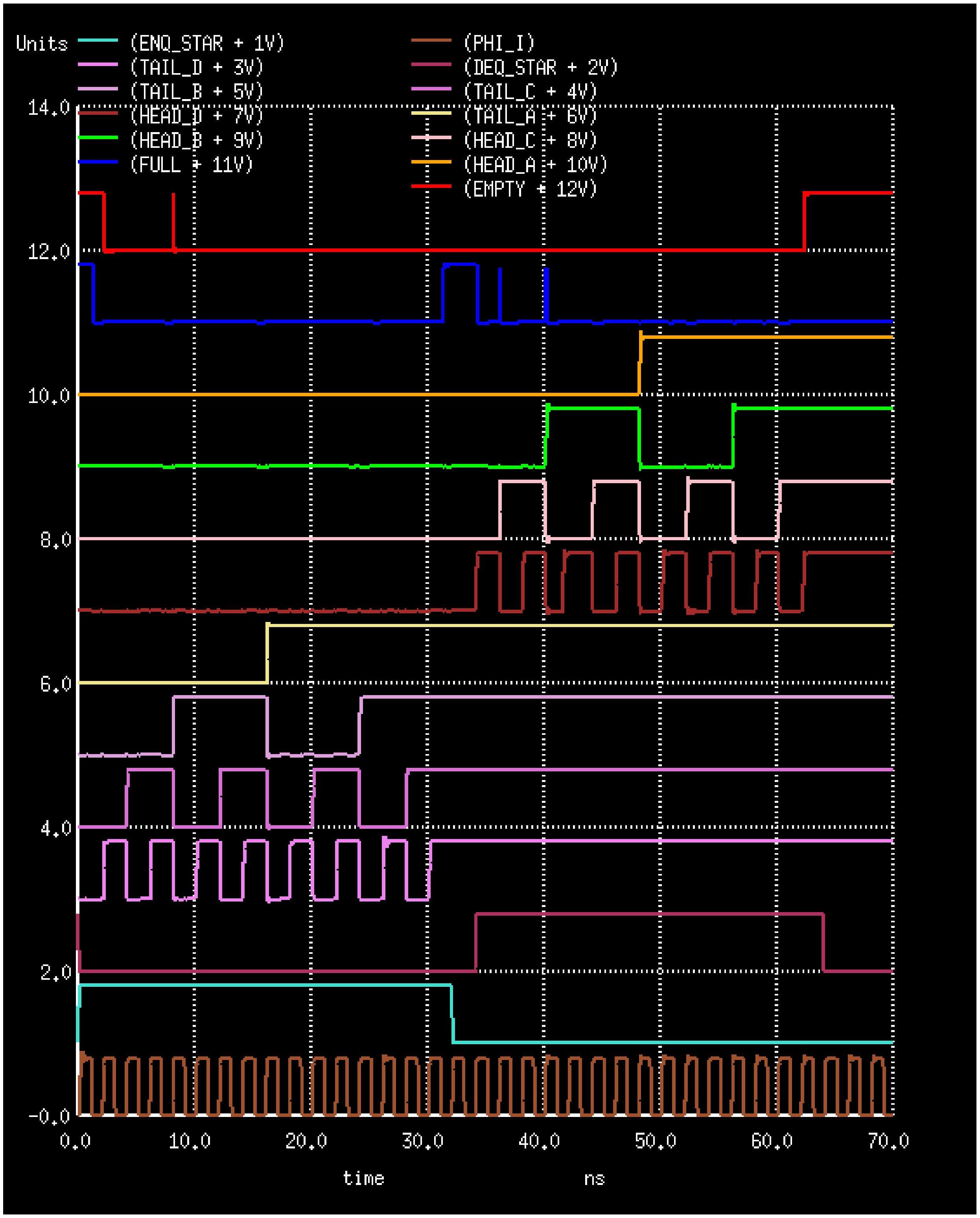


Figure 54: Pointers test, empty and full

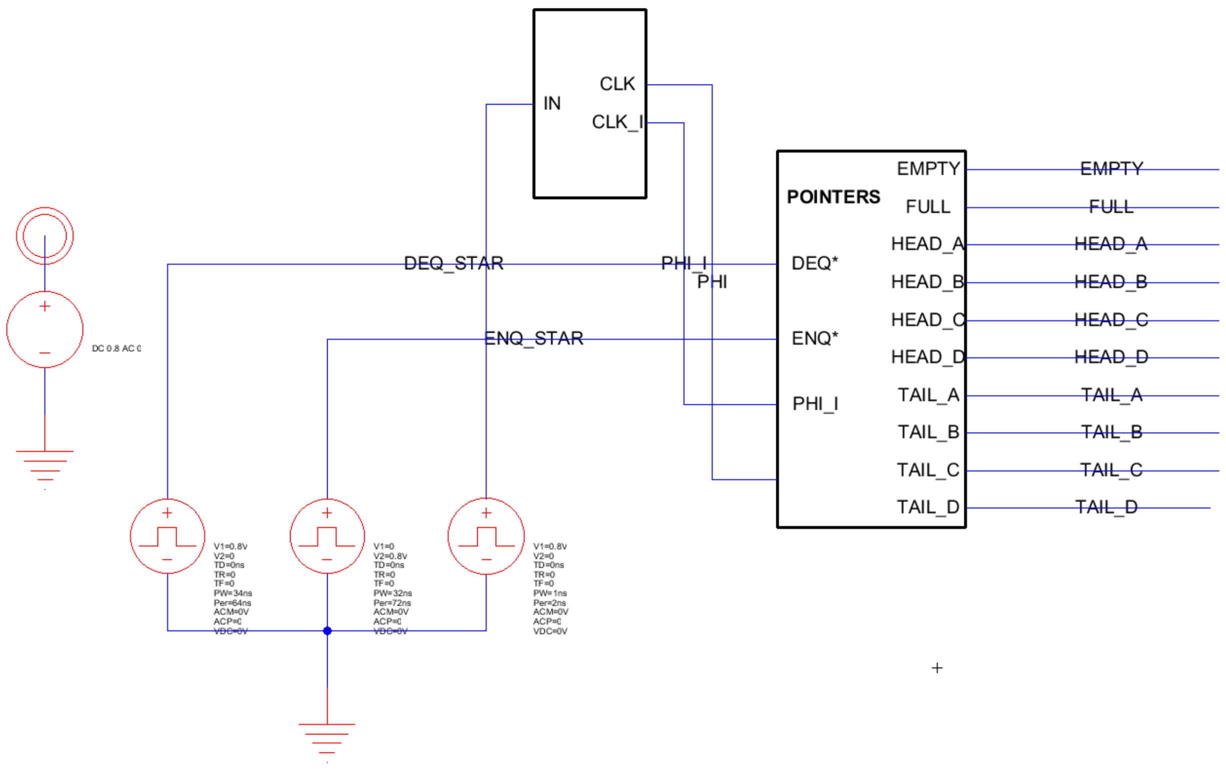

IN
CLK
CLK_I
POINTERS
EMPTY
FULL
HEAD_A
HEAD_B
HEAD_C
HEAD_D
TAIL_A
TAIL_B
TAIL_C
TAIL_D
DEQ*
ENQ*
PHI_I
DEQ_STAR
ENQ_STAR
PHI_I
PHI
EMPTY
FULL
HEAD_A
HEAD_B
HEAD_C
HEAD_D
TAIL_A
TAIL_B
TAIL_C
TAIL_D
DC 0.8 AC 0
V1=0.8V V2=0 TD=0ns TR=0 TF=0 PW=34ns Per=64ns ACM=0V ACP=0 VDC=0V
V1=0 V2=0.8V TD=0ns TR=0 TF=0 PW=32ns Per=72ns ACM=0V ACP=0 VDC=0V
V1=0.8V V2=0 TD=0ns TR=0 TF=0 PW=1ns Per=2ns ACM=0V ACP=0 VDC=0V

Figure 55: Pointers test schematic

Decoder

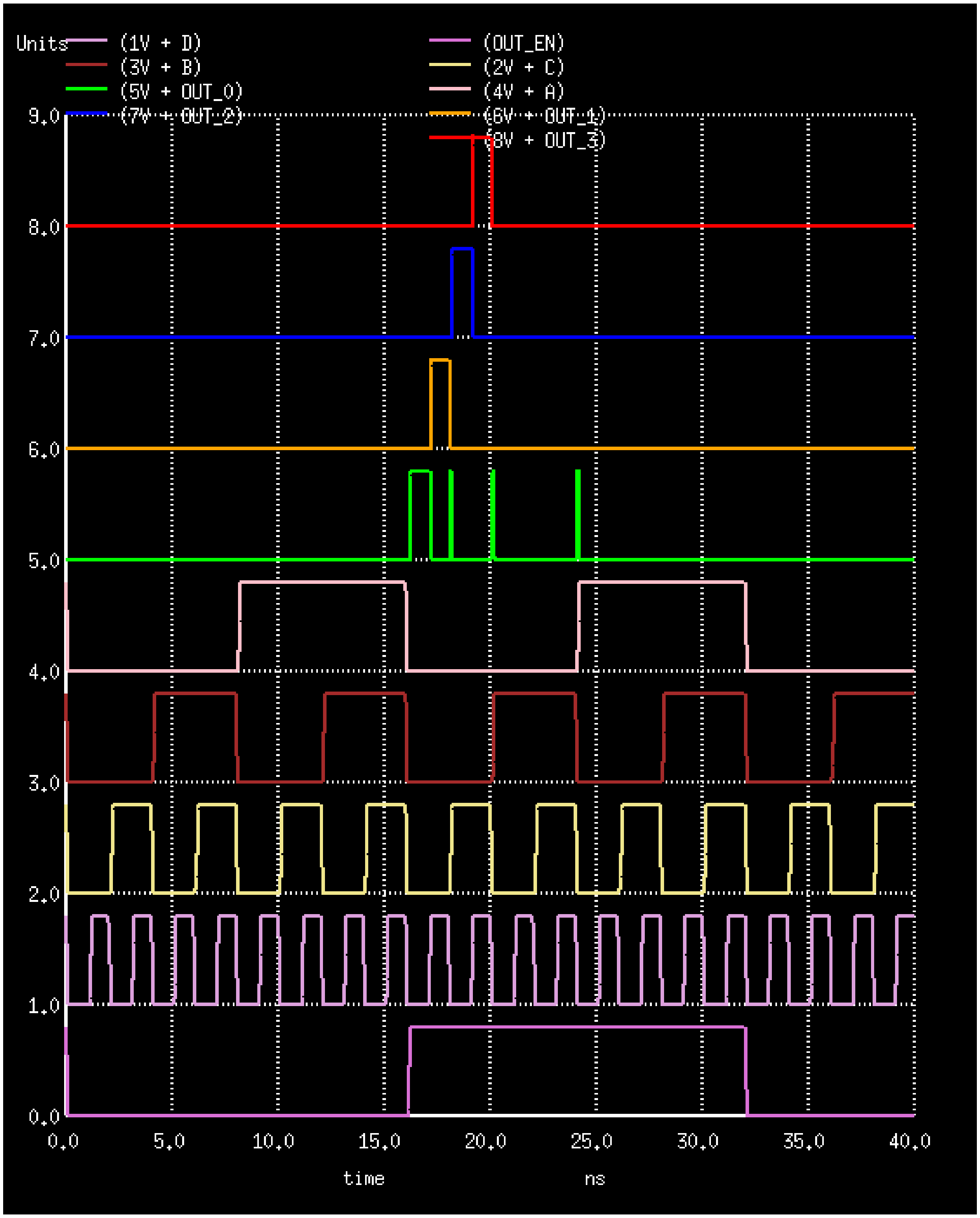


Figure 56: Decoder test, outputs 0 to 3

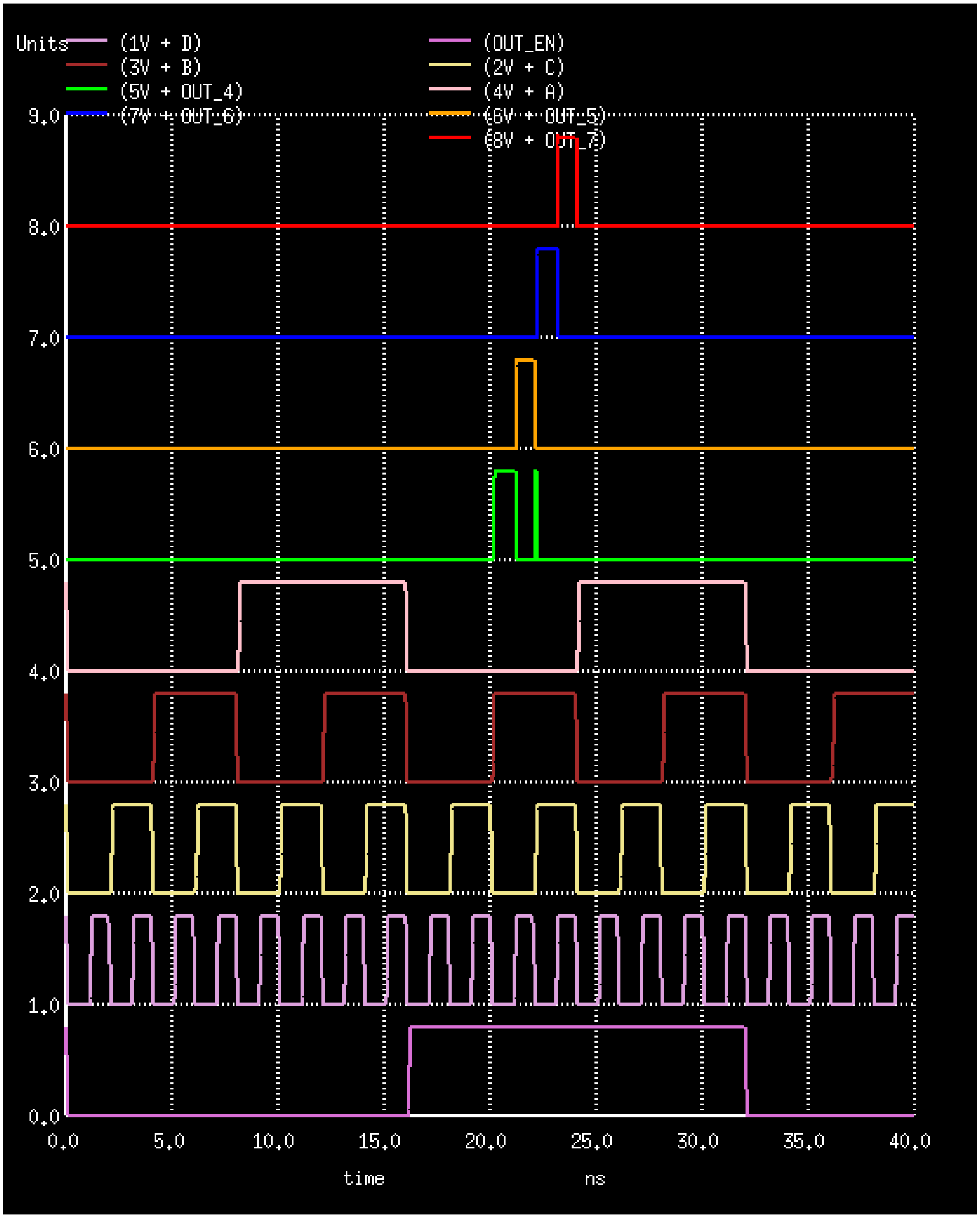


Figure 57: Decoder test, outputs 4 to 7

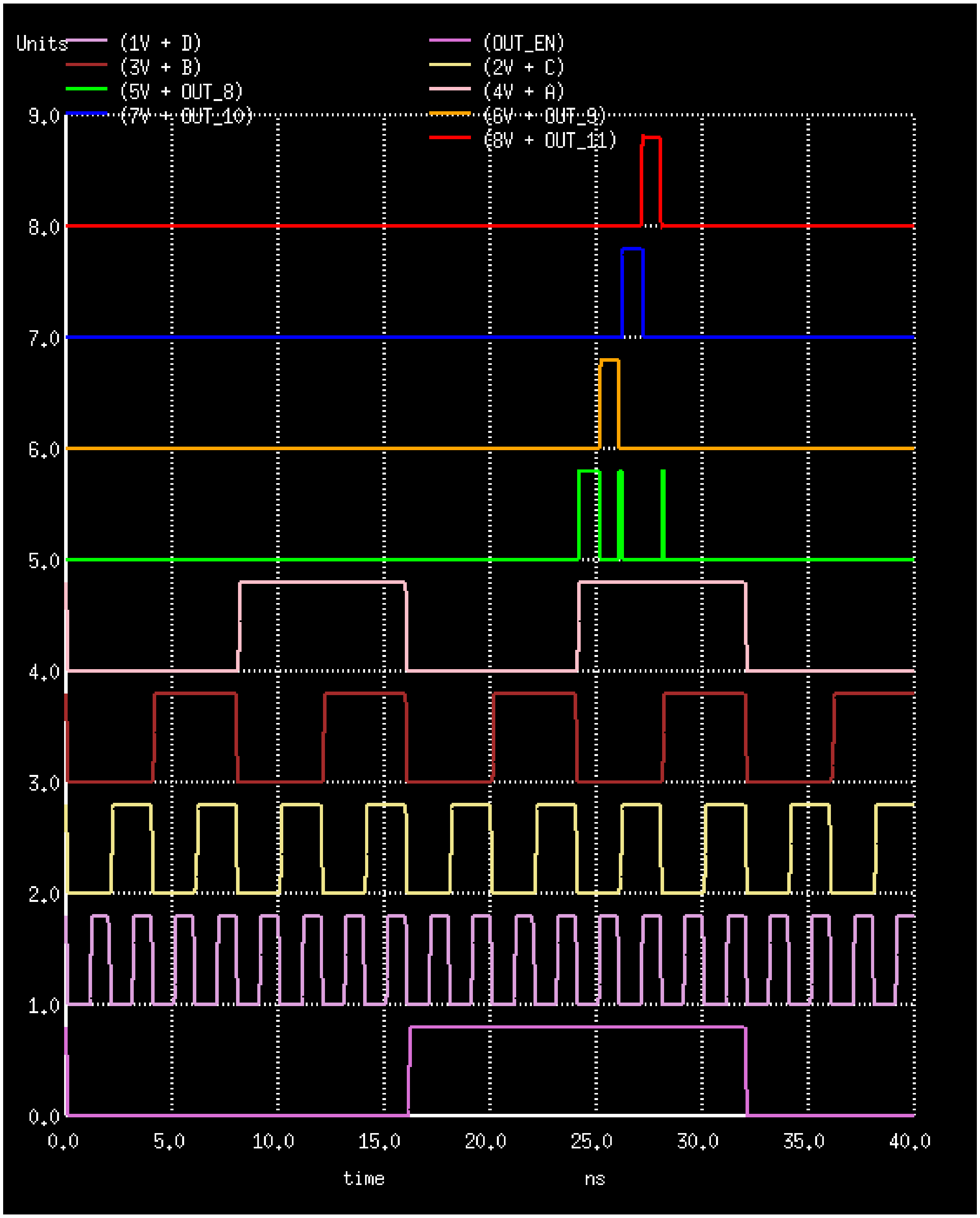


Figure 58: Decoder test, outputs 8 to 11

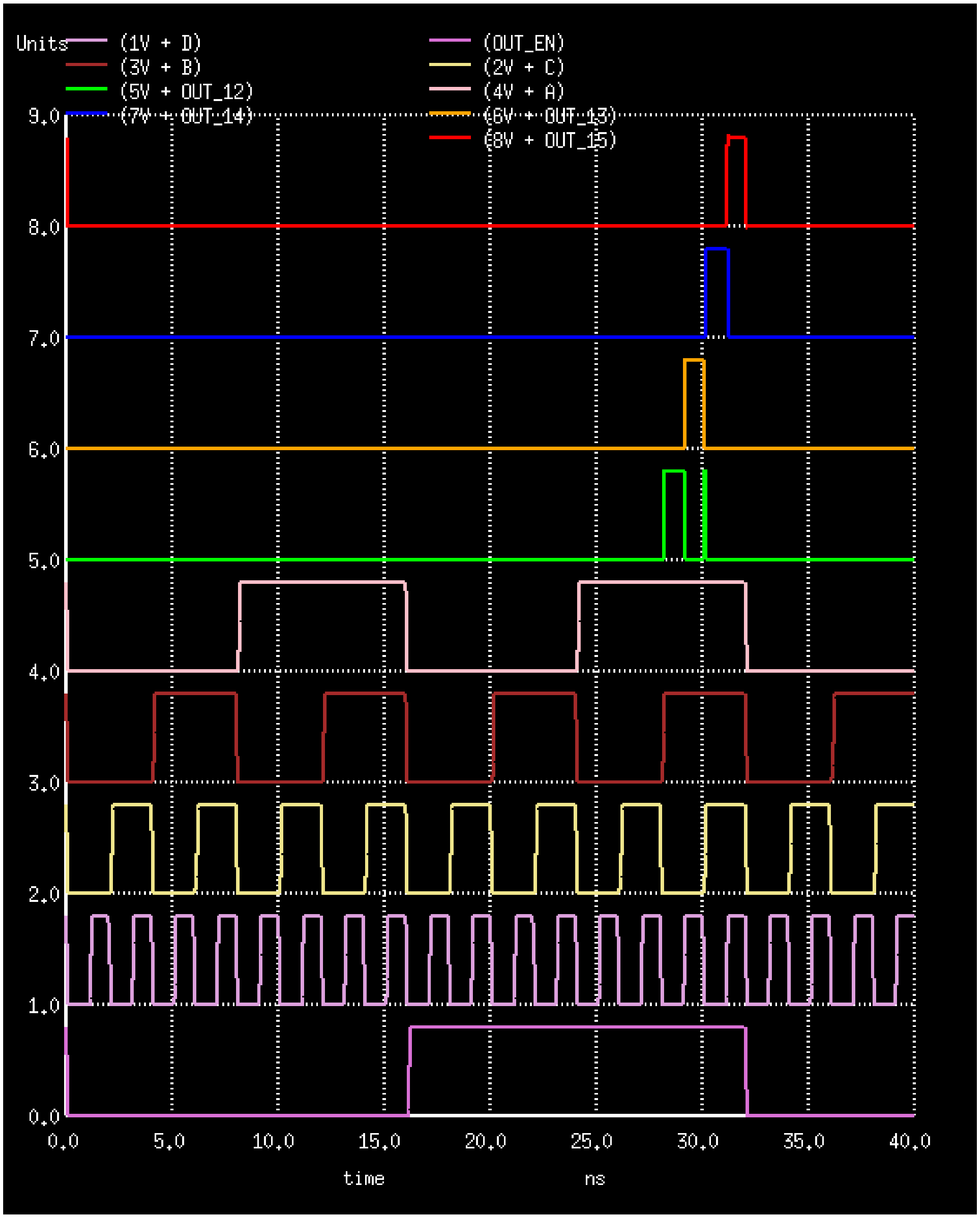


Figure 59: Decoder test, outputs 12 to 15

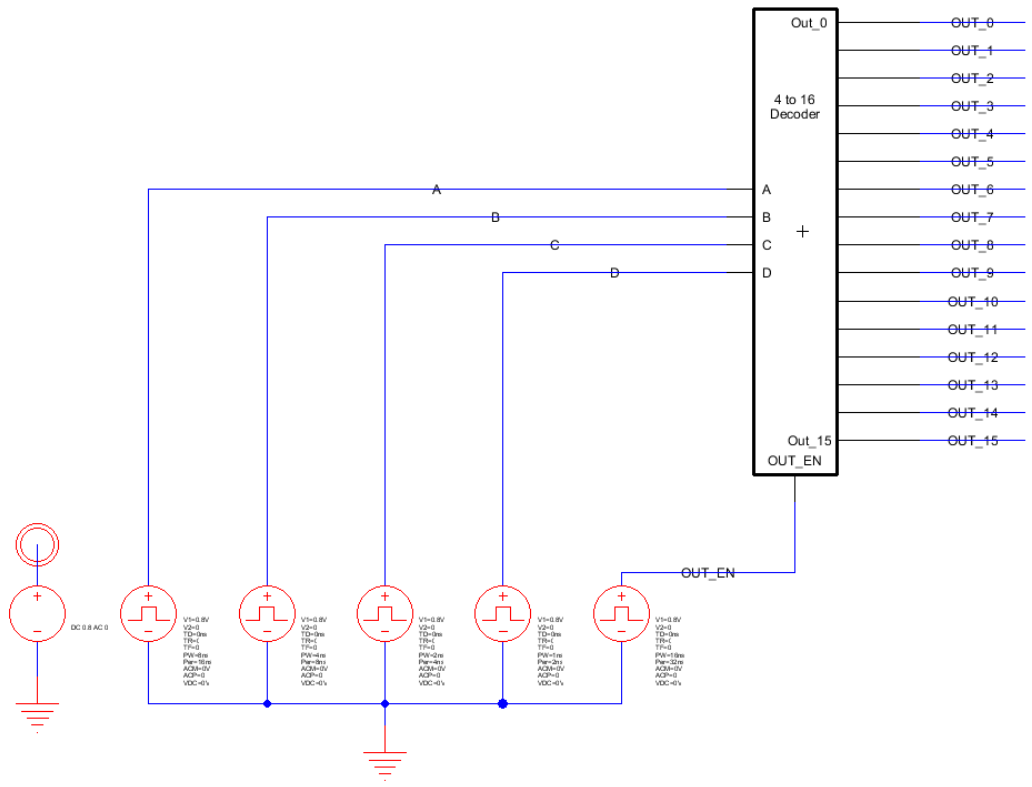

4 to 16 Decoder
Out_0
Out_15
OUT_EN
A
B
C
D
OUT_0
OUT_1
OUT_2
OUT_3
OUT_4
OUT_5
OUT_6
OUT_7
OUT_8
OUT_9
OUT_10
OUT_11
OUT_12
OUT_13
OUT_14
OUT_15
OUT_EN
DC 0.8 AC 0
V1=0.8V V2=0 TD=0ns TR=(TF=0 PW=8ns Per=16ns ACM=0V ACP=0 VDC=0's
V1=0.8V V2=0 TD=0ns TR=(TF=0 PW=4ns Per=8ns ACM=0V ACP=0 VDC=0's
V1=0.8V V2=0 TD=0ns TR=(TF=0 PW=2ns Per=4ns ACM=0V ACP=0 VDC=0's
V1=0.8V V2=0 TD=0ns TR=(TF=0 PW=1ns Per=2ns ACM=0V ACP=0 VDC=0's
V1=0.8V V2=0 TD=0ns TR=(TF=0 PW=16ns Per=32ns ACM=0V ACP=0 VDC=0's

Figure 60: Decoder test schematic

Force Dequeue

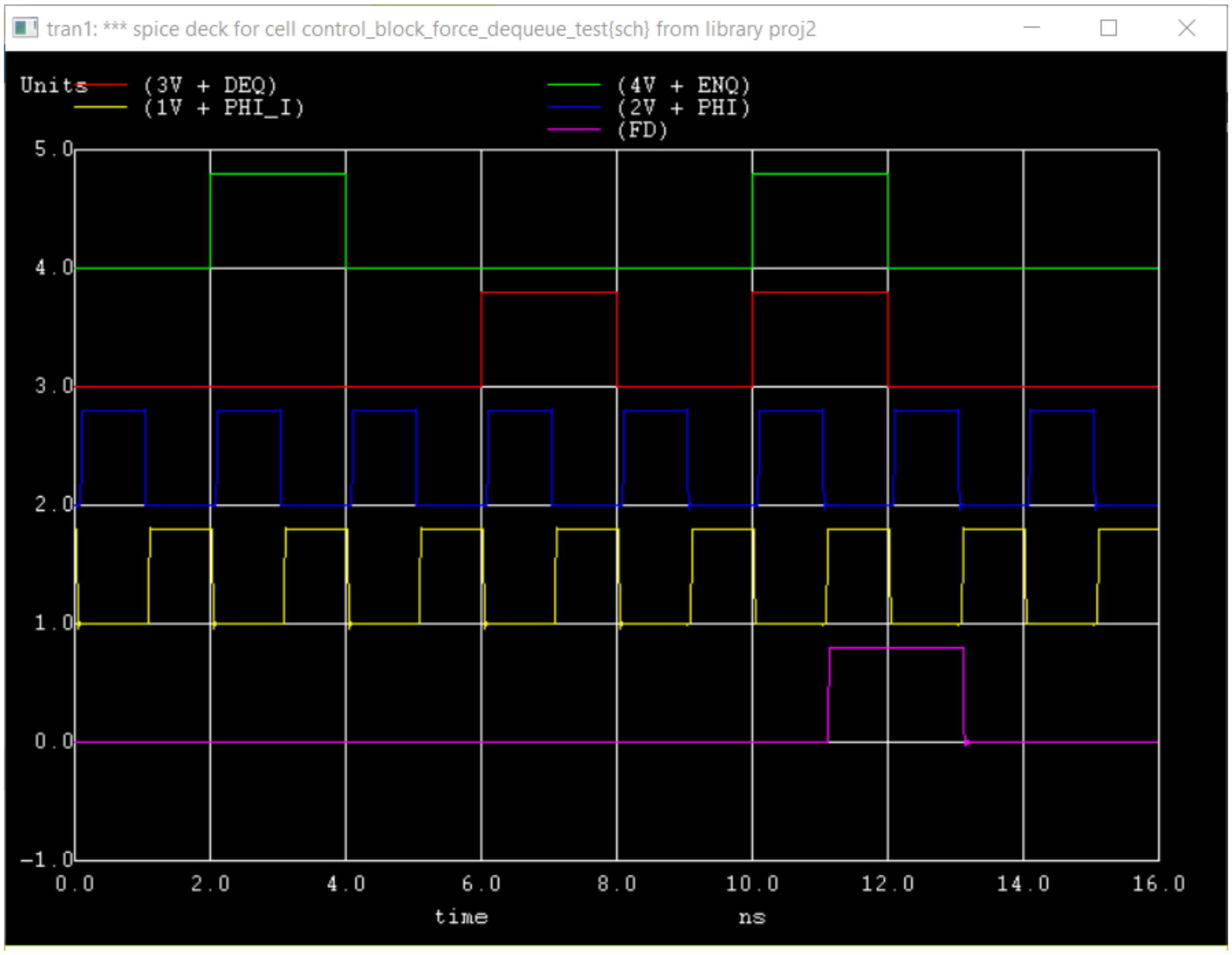


Figure 61: Force dequeue module test

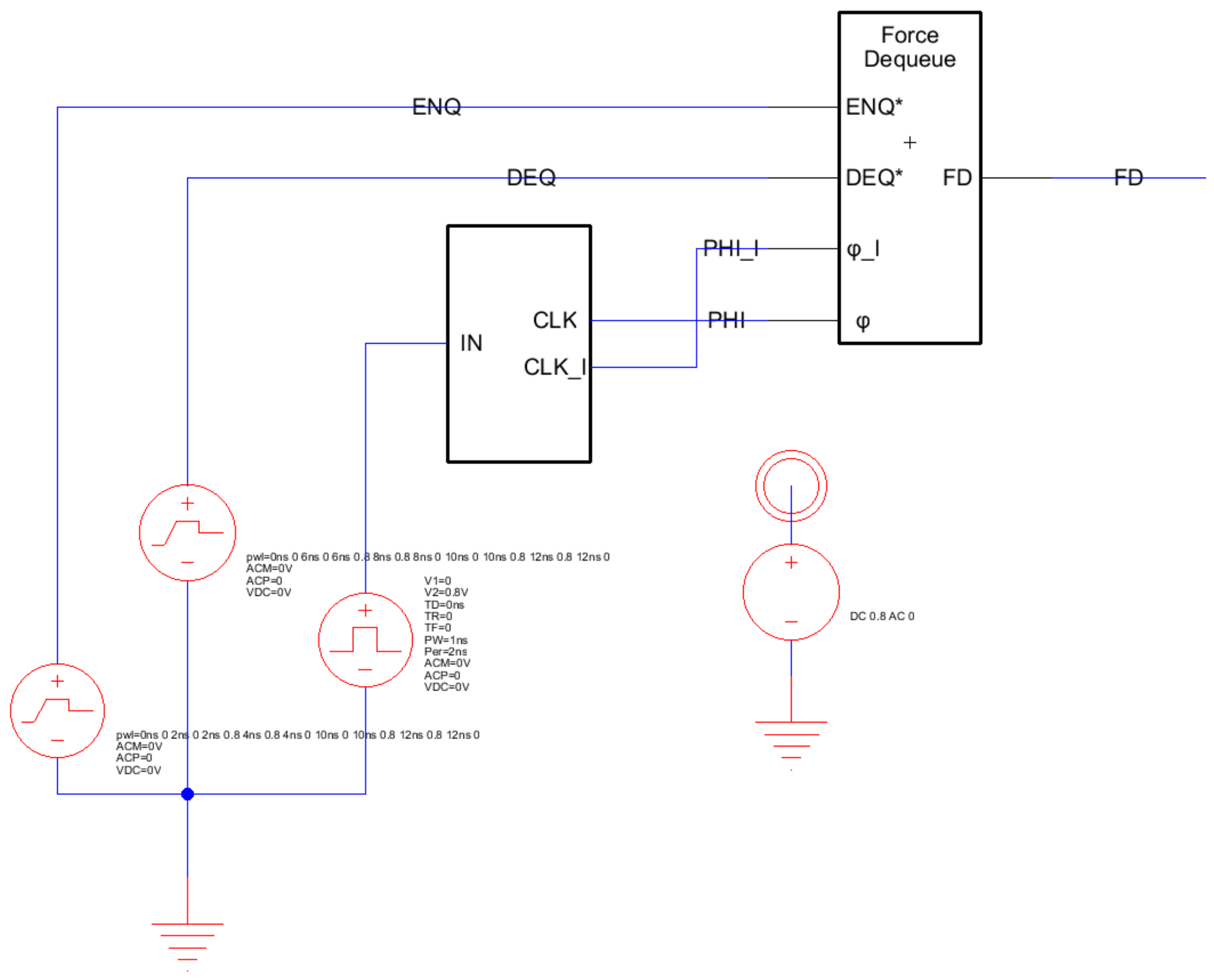


Figure 62: Force dequeue test schematic

4-bit Precharge Module

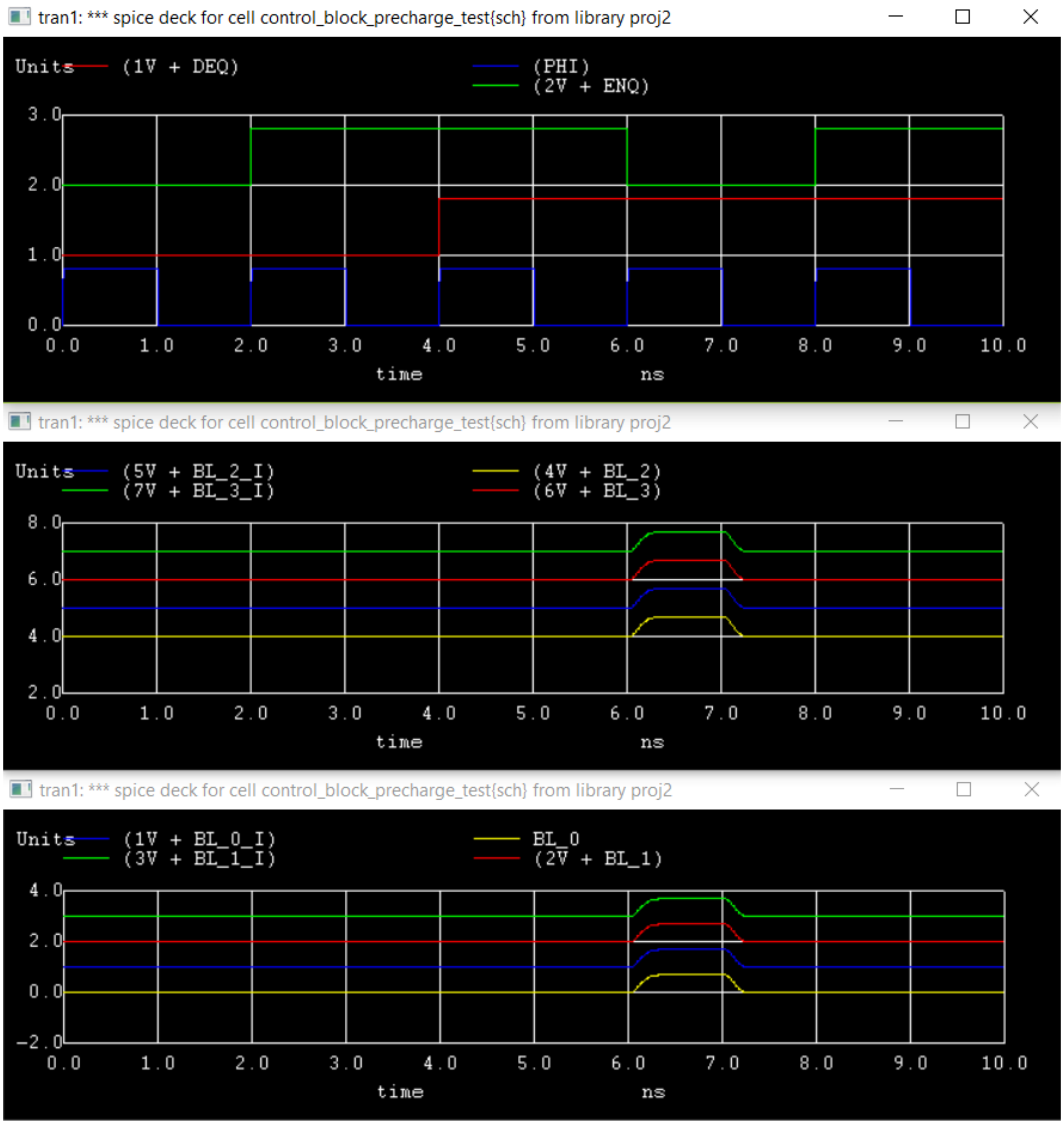


Figure 63: 4-bit precharge module test

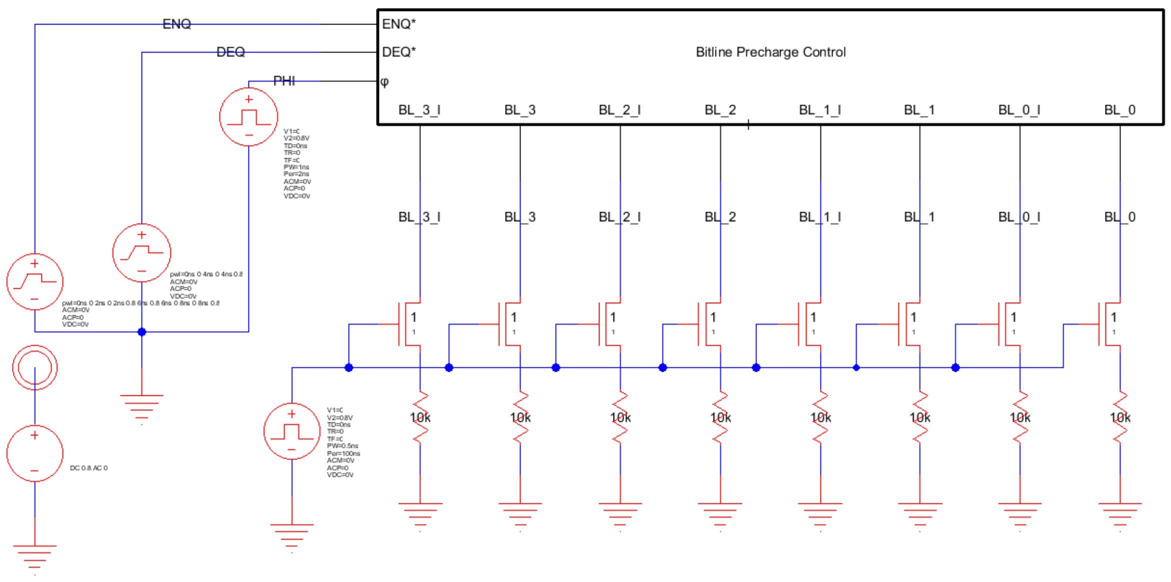


Figure 64: 4-bit precharge test schematic

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.

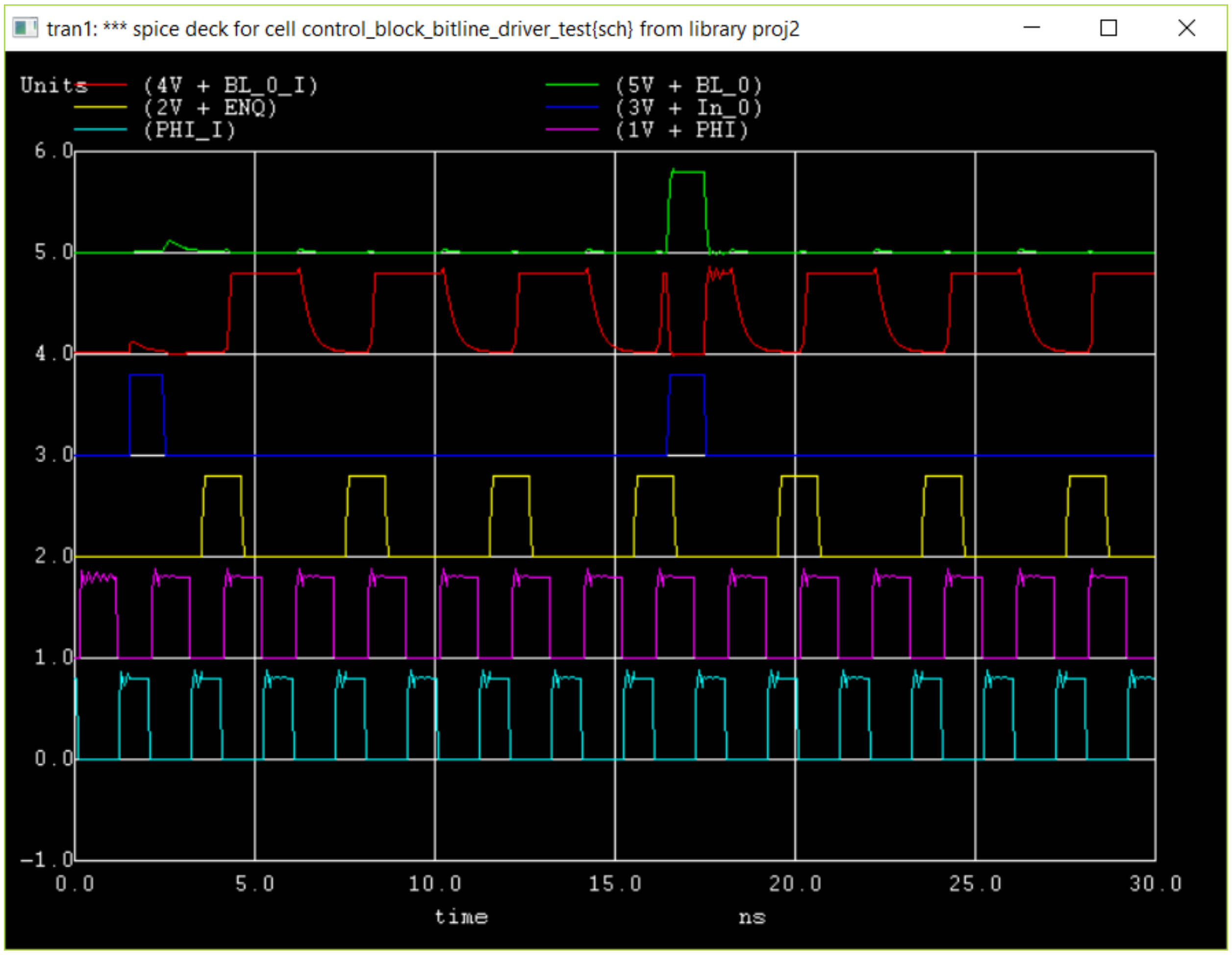


Figure 65: Bitline driver test, bit 0

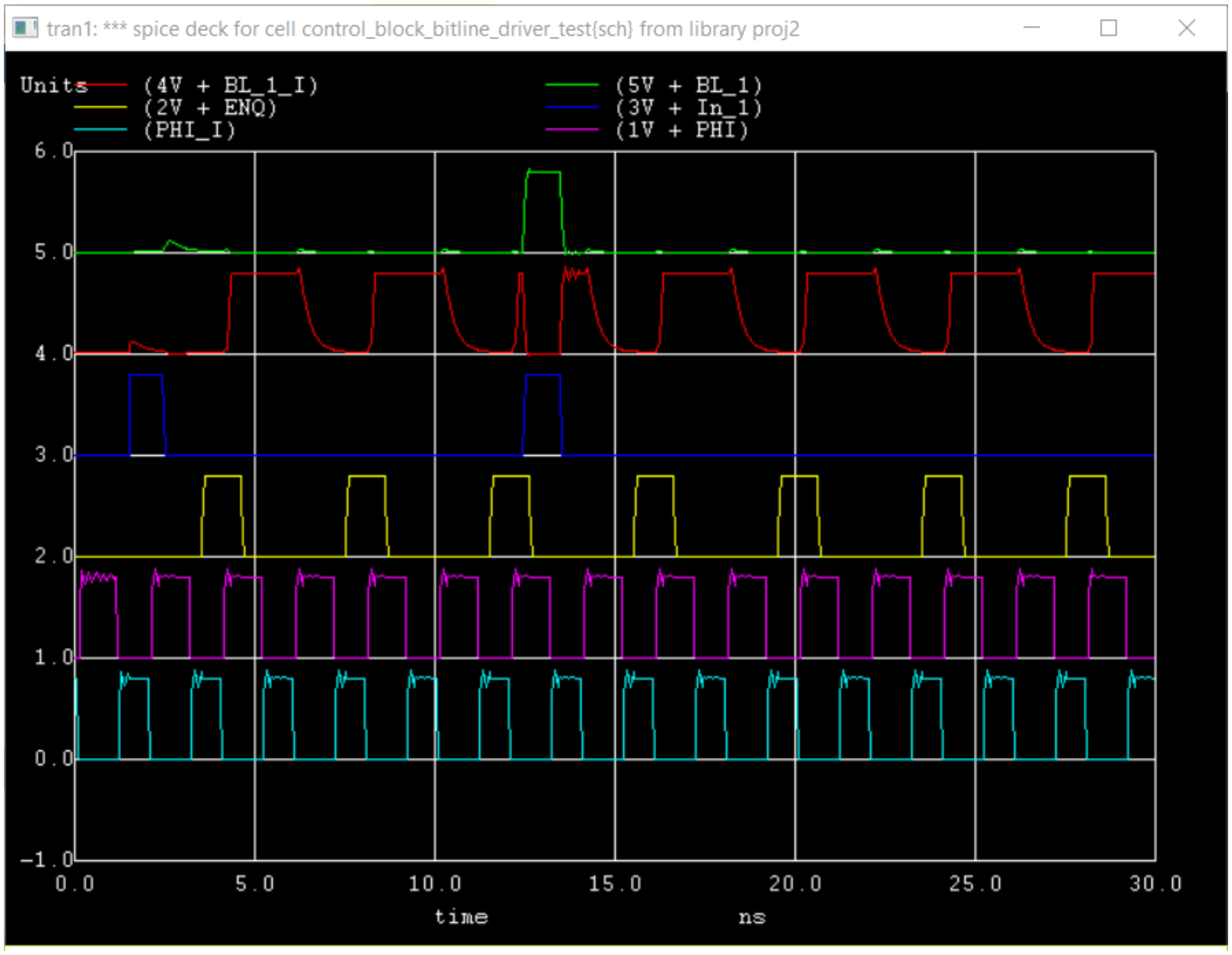


Figure 66: Bitline driver test, bit 1

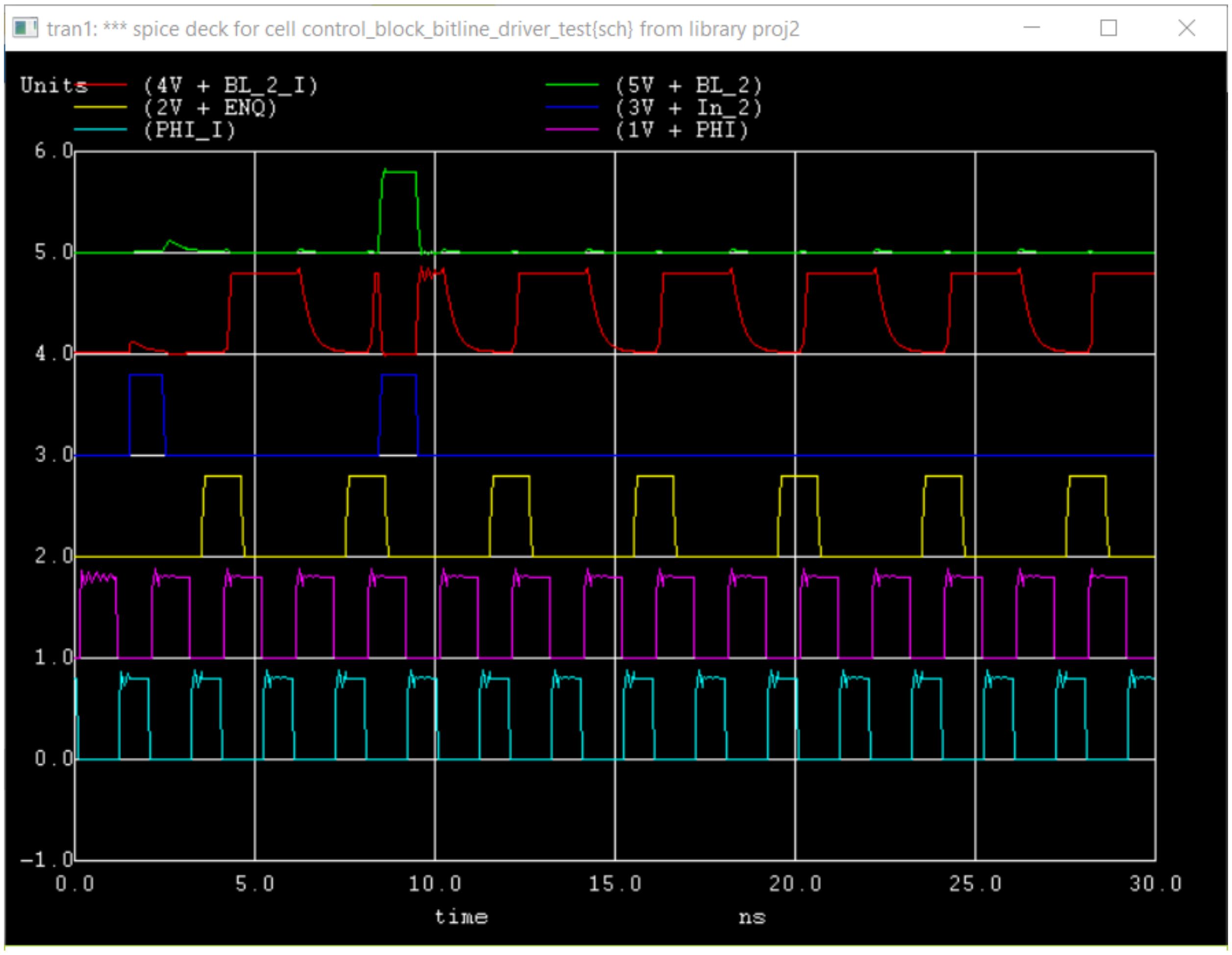


Figure 67: Bitline driver test, bit 2

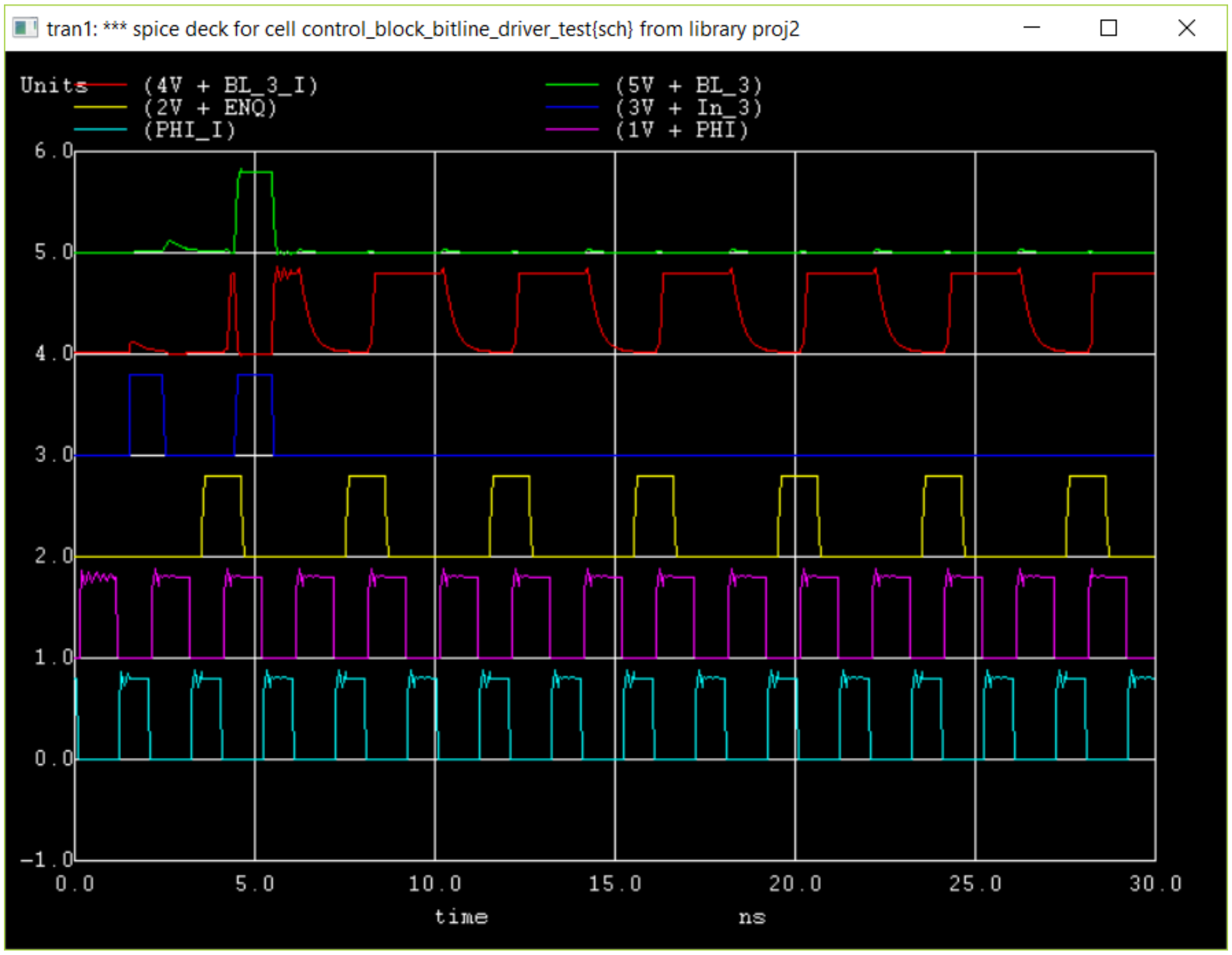


Figure 68: Bitline driver test, bit 3

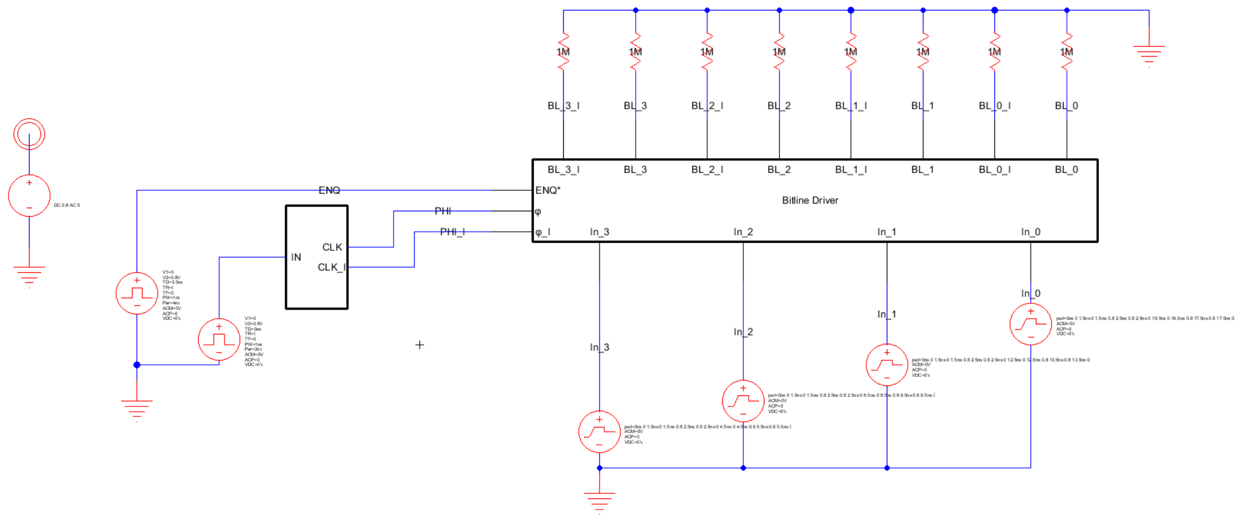


Figure 69: Bitline driver test schematic

Full FIFO Queue

For the full queue, we tested several operations.

The first operation that we tested is enqueueing once, then dequeueing twice. This was done to ensure that dequueing an empty queue does not result in the head or tail pointers changing. It also tests that we are dequeueing the correct value when we issue a valid dequeue request.

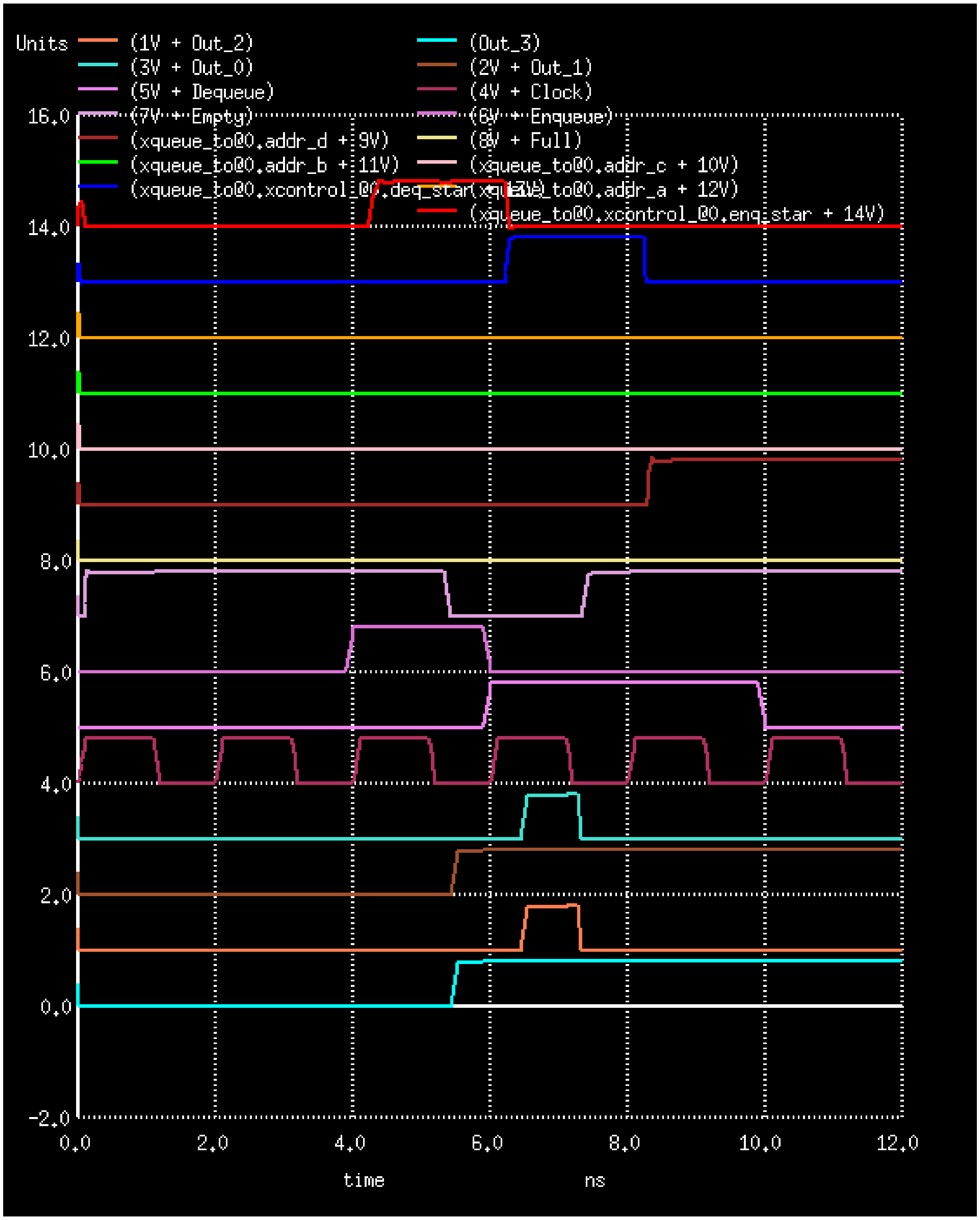


Figure 70: Queue test, enqueue once and then dequeue twice

The second test we did was ensuring that enqueueing on a full queue does not result in the tail pointer changing.

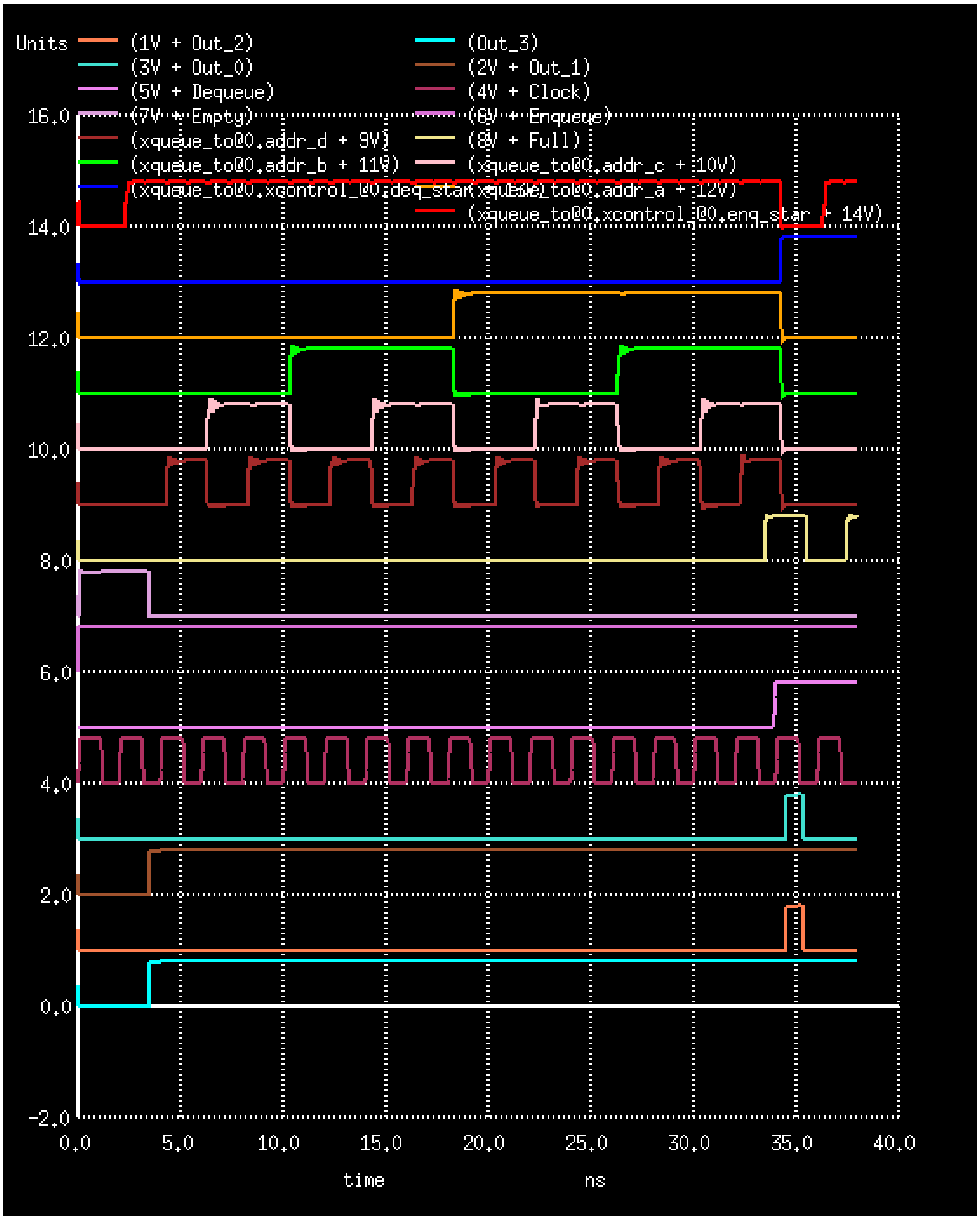


Figure 71: Queue test, enqueue on full queue

We also performed a test in which we enqueued and dequeued a few times. We first enqueue 1 and then enqueue 0 2 times. The first dequeue returns 1, and the next two dequeues return 0, as expected.

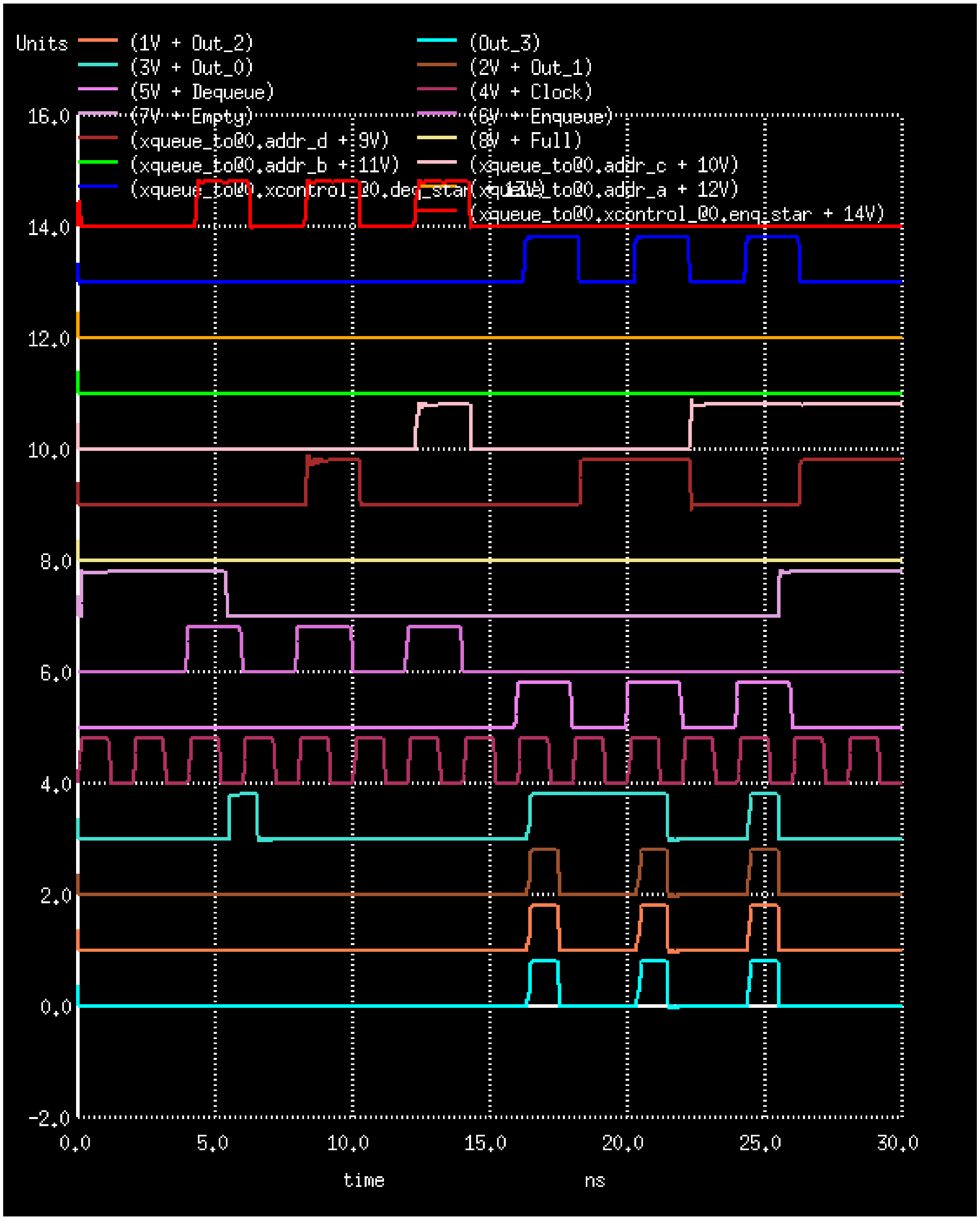


Figure 72: Queue test, enqueue and dequeue a few times

We also tested that enqueueing and dequeuing simultaneously worked according to the specifications. As shown, enq/deq on an empty queue just enqueues, while enq/deq on a non-empty, non-full queue does enqueue and dequeue a value.

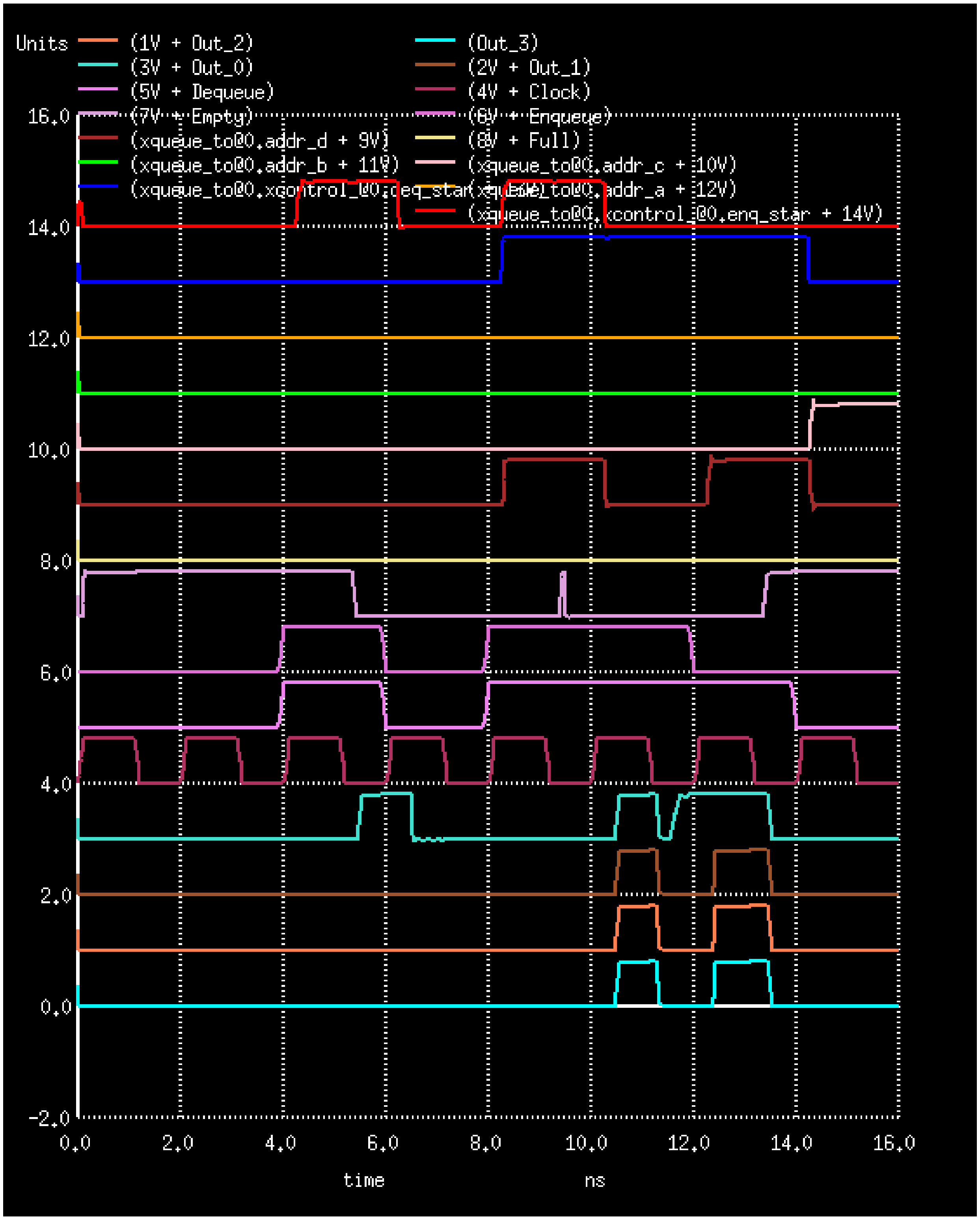


Figure 73: Queue test, simultaneous enqueue and dequeue on empty and nonempty, non-full

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.