ASYNCHRONOUS RESEARCH CENTER

Portland State University

Subject: **Ninth Class Handout – Nine/One GasP**
Date: **November 23, 2010**
From: **Ivan Sutherland**
ARC#: **2010-is56**

References:

ARC# 2010-is43: Class 1 – Ring Oscillators, Ivan Sutherland, 26 September 2010
ARC# 2010-is44: Class 2 – GasP Rings, Ivan Sutherland, 1 October 2010
ARC# 2010-is45: Class 3 – Linear GasP, Ivan Sutherland, 8 October 2010
ARC# 2010-is49: Class 4 – Proper Stopper, Ivan Sutherland, 15 October 2010
ARC# 2010-is52: Class 5 – Broad Branch and Broad Merge, Ivan Sutherland, 22 October 2010
ARC# 2010-is53: Class 6 – Round Robin FIFO, Ivan Sutherland, 25 October 2010
ARC# 2010-is53: Class 7 – Data-Controlled Branch & Demand Merge, Ivan Sutherland, 10 Nov 2010
ARC# 2009-is26: The GasP Tutorial Aids, Ivan Sutherland, 25 August 2009

PURPOSE

This memo considers the 9/1 GasP control circuit that was your homework from last time. The simulation assignment for next time exhibits a 9/1 GasP circuit.

THE 9/1 GasP CIRCUIT – step[1,2,3,4]ofHILO

Last week we used the "paper dolls" from ARC# 2009-is26 to examine various GasP structures. With those aids we could make all kinds of GasP circuits from 8/2 GasP to 2/8 GasP. Because the GasP circuits we considered use loops of five logic gates, their cycle time is ten gate delays per cycle. The sum of the forward and backward latency, which is always the cycle time, is likewise ten gate delays.

If the forward latency is an odd number of gate delays, so must be the reverse latency because their sum is even. Moreover, if the forward latency from state wire to state wire is odd, successive state wires must follow opposite conventions. The successor state wire of any odd-latency stage must follow the LO-is-FULL convention if the predecessor follows the HI-is-FULL convention, and vice versa. We use 6/4 GasP a lot precisely because identical 6/4 GasP stages can function together.

For 23 November your assignment was to design a 9/1 GasP control circuit. I hope that this assignment has made you think about the constraints on the GasP family of circuits.

Careful thought about constraints leads directly to a working 9/1 GasP circuit. Because both 9 and 1 are odd, the sense of the predecessor and successor state wires must differ. Thus there are two types of 9/1 GasP modules: for its predecessor state wire one form uses the HI-is-FULL convention; the other form uses the LO-is-FULL convention on its predecessor state wire. Both forms use the opposite convention for their successor state wire. This memo offers designs for both forms.

Let us go step-by-step to develop a 9/1 GasP circuit. We'll use HI-is-FULL as the convention for the predecessor state wire. We'll use LO-is-FULL as the convention for the successor state wire. The final design is called `gasp9/1hi-lo` and the steps towards it are illustrated in the schematics called `step1ofHILO`, `step2ofHILO`, `step3ofHILO`, and `step4ofHILO`.

First, the conventions we've chosen for the predecessor and successor state wires dictate the kind of AND function we need. Regardless of the convention, the AND always acts when the predecessor state wire is FULL and the successor state wire is EMPTY. For the conventions we've chosen, the AND function must act when both state wires are HI. A NAND gate, with its two series N-type transistors, does this nicely.

Second, there's only a single gate delay in the reverse direction. The single gate delay lacks room for an AND function separate from the transistor that drains the predecessor state wire. We must include an AND function in the transistor that drains the predecessor state wire. Instead of a lonesome N-type transistor, we use a series pair of N-type transistors to drain the predecessor state wire.

Third, we can preserve a lonesome transistor to fill the successor state wire. Because the successor state wire uses the LO-is-FULL convention, the lonesome transistor that fills the successor state wire must be N-type.

The result of these three steps is the partial circuit called **`step1ofHILO`**.

The circuit called `step1ofHILO` has two separate but related AND functions. Because they are closely related, I like to put the twin AND functions close together in the schematic as you see in `step1ofHILO`. Inside the NAND gate there are two series N-type transistors wired exactly in parallel with the series transistor pair just below the NAND gate. When the output of NAND goes LO, the external series pair of N-type transistors drains the predecessor state wire to LO. The twin AND functions act together.

The position of the NAND gate suggests that its left input should be somehow associated with the predecessor state wire. Our next step will be to figure out that connection. We must remember to make loops of five gates.

Fourth, suppose that this stage is waiting only for input on its predecessor state wire because both predecessor and successor state wires are empty. If so, the successor state wire will be HI, meaning EMPTY, and the predecessor state wire will be LO, also meaning EMPTY. Shortly after the predecessor state wire goes HI, meaning FULL, this GasP stage should fire, taking the usual three actions: capture data, drain the predecessor, and fill the successor.

One action must drain the predecessor state wire, but that drain action should happen five gate delays after the predecessor state wire becomes FULL. Thus we need four more logic gates between the predecessor state wire and the pair of AND functions. Four inverters will do as shown in **step2ofHILO**. Notice the loop of five gates associated with the predecessor state wire. Five gate delays after the predecessor state wire goes HI the series N-type transistors will drain it back to LO.

Fifth, let us suppose instead that the stage is waiting for space at its successor state wire. In this case the successor state wire is LO, meaning FULL, and the predecessor state wire is HI, also meaning FULL. When space arrives the successor state wire goes HI, meaning EMPTY, the twin AND functions must act. The series N-type transistors will immediately drain the predecessor state wire, providing the reverse latency of only one gate delay. The NAND gate's output will also go LO which must turn on the lonesome transistor at the top of the figure to fill the successor state wire. We need a successor loop of five logic gates. That loop already involves the NAND gate and the lonesome N-type transistor at the top of the schematic, and so we need to add three inverters as shown in **step3ofHILO**.

Count the number of gates in the forward and reverse directions in step3ofHILO. From state wire to state wire, there are 9 gates in the forward direction and only one in the reverse direction. We have a 9/1 GasP circuit.

Sixth, let us add loads to each inverter to make them all drive loads of about three times their own strength and thus all have about the same delay. The result appears in **step4ofHILO**. This circuit is complete except for missing keepers and its unconnected fire output. The next section reasons about the keepers. A later section deals with the fire signal outputs.

KEEPERS – gasp9/1hi-lo

The figure called step4ofHILO is the raw 9/1 GasP circuit without keepers.

Adding the successor keeper is easy. The keeper for the successor state wire appears at the right of **gasp9/1hi-lo**. Because this stage will fill the successor state wire by driving it LO, the successor keeper's job is to keep the successor state wire HI under certain conditions. The conditions are: (A) the state wire is already HI, and (B) this stage is not about to fill the successor state wire by driving it LO. The series pair of P-type transistors in the keeper meet these conditions. The upper P-type

transistor turns on only if the state wire is already HI, to meet condition (A). To meet condition (B), the lower P-type transistor turns off two gate delays before this stage drains the successor state wire.

I found the predecessor keeper much harder to figure out. Because this stage will drain the predecessor state wire by driving it LO, the predecessor keeper's job is to keep the predecessor state wire HI under certain conditions. The conditions are: (C) master clear is off, (D) the predecessor state wire is already HI, and (E) this stage is not draining the predecessor state wire. The three series P-type transistors in the predecessor keeper meet these conditions. The upper P-type transistor turns off during master clear, to meet condition (C). The middle P-type transistor turns on only if the state wire is already HI, to meet condition (D).

Condition (E) gave me the most grief. In effect condition (E) says to avoid keeping if "this stage is about to drain the predecessor state wire." This stage drains the predecessor state wire only if predecessor is FULL and the successor is EMPTY. Therefore, the keeper should "keep" only when predecessor is FULL, and successor is also FULL. That is when predecessor and successor are HI and LO respectively.

If the predecessor state wire is HI, meaning FULL, this stage will drain the predecessor state wire only if the successor state wire becomes HI, meaning EMPTY. As long as the successor state wire remains LO, meaning FULL, the stage is waiting for successor space and will not drain its predecessor state wire. As soon as the successor state wire goes HI, meaning EMPTY, the keeper should get out of the way. Thus connecting the successor state wire to the third P-type transistor as shown in `gasp9/1hi-lo` meets condition (E).

CHOOSING FIRE SIGNALS - gasp9/1hi-lo and gasp9/1lo-hi

There's considerable flexibility in where one places fire signals. For example, consider three identical 6/4 GasP stages that produce fire signals called `fire[X]`, `fire[Y]`, and `fire[Z]`. The forward latency is six between each pair of state wires and each pair of fire signals. However, suppose we amplify `fire[Y]` with two inverters to produce a much stronger signal that we'll call `firePrime[Y]`. The amplification time increases the forward latency from `fire[X]` to `firePrime[Y]` by two gate delays for a total of eight gate delays. Likewise the amplification time decreases the forward latency between `firePrime[Y]` and `fire[Z]` by two gate delays for a total of only four gate delays. The reverse latencies change in the opposite way. From `firePrime[Y]` to `fire[X]` the reverse latency is only two gate delays. From `fire[Z]` to `firePrime[Y]` the reverse latency is now six gate delays.

When all stages are identical, the forward latency from state wire to state wire is the same as from fire signal to fire signal. Moreover, if all stages are identical all latencies must be even. However, if successive stages differ, we must distinguish two measures of latency. First, the latency from state wire to state wire is what we generally

think of when we describe GasP circuits. Thus the 9/1 GasP circuits considered here have an odd latency from state wire to state wire in both the forward and the reverse direction. As a result successive state wire conventions differ.

However, if GasP circuits drive identical latches, their fire pulses must be similar. Similar pulses can result only from even numbers of gate delays, and thus the fire-to-fire forward and reverse latencies must all be even. Indeed, one is free to adjust the fire-to-fire latency by adding or removing amplifiers. As we saw at the beginning of this section, even in a series of 6/4 GasP stages, one might choose to delay the fire signal for a particular stage to amplify that fire signal.

I used this kind of flexibility to choose where to extract the fire signals from `gasp9/1hi-lo` and `gasp9/1lo-hi`.

Consider the class homework assignment called `twoFIFOs`. It has two identical control paths whose datapaths differ slightly. Each control path uses two stages of 9/1 GasP, `fire[3]` and `fire[4]` to provide extra time for a parity circuit called `par`. Moreover, I chose the fire outputs of the two 9/1 GasP circuits to provide ten gate delays of forward latency between `fire[3]` and `fire[4]`. The latency between these fire pulses had to be even, and I chose to make it ten rather than eight. This choice had ramifications that I didn't anticipate. We'll consider that next.

First, let us count gate delays in `twoFIFOs`. From `sw[2]` to `sw[4]` there are two stages of 9/1 GasP for a total forward latency of 18 gate delays from state wire to state wire. That's the equivalent of three stages of 6/4 GasP between the existing 6/4 GasP modules that produce `fire[2]` and `fire[5]`. Thus the latency from `fire[2]` to `fire[5]` must be the same as for four stages of 6/4 GasP. We know that `fire[5]` must follow `fire[2]` by 24 gate delays.

The fire signals from the 9/1 GasP modules, `fire[3]` and `fire[4]` must fit somewhere inside those 24 gate delays. I chose to separate `fire[3]` and `fire[4]` by ten gate delays, leaving 14 gate delays unassigned. I chose to allocate 8 of those 14 between `fire[2]` and `fire[3]` and the remaining 6 between `fire[4]` and `fire[5]`. Thus the step from `fire[4]` to `fire[5]` is the same as for 6/4 GasP, but the step from `fire[2]` to `fire[3]` is two gate delays longer than usual.

DATAPATH CONSTRAINTS

We have not yet considered datapath timing constraints. Suppose all fire signals are positive pulses of approximately five gate delays duration. Consider two successive stages `fire[A]` and `fire[B]`. If source limited, `fire[A]` occurs before `fire[B]`. For 6/4 GasP their separation in time is six gate delays and in the source-limited case they never overlap. However, if sink limited, `fire[B]` occurs before `fire[A]`. For 6/4 GasP the reverse latency is four gate delays, and the two pulses overlap slightly. A homework question in lesson 3 asked you to measure the overlap.

The overlap gets larger for smaller reverse latency. During the overlap period the latches in the two stages are both transparent. An error may occur if the datapath is fast enough for data to sneak through two latches during the overlap time.

There is a two-sided timing constraint on the delay in the datapath. For the source-limited case, the datapath must be fast enough to keep up with the forward advance of the control. For the sink-limited case, the datapath must have enough delay to prevent data from sneaking through two stages during the time both are transparent. The second case, sink-limited, is harder to think about and thus more prone to design errors.

When designing circuits with different forward latency one must bear these constraints in mind. The 9/1 GasP circuits of the present exercise give us an excellent forum to consider these constraints.

The long forward latency between `fire[2]` and `fire[3]` in `twoFIFOs` worries me. I am concerned that the latches we use are fast enough to cause trouble between these two stages in the sink-limited case.

The eight gate delay forward latency between `fire[2]` and `fire[3]` implies a two gate delay reverse latency. Thus there's a three gate delay interval when both `fire[2]` and `fire[3]` are HI. Could fresh data pass through both sets of latches at once? The datapath between `fire[2]` and `fire[3]` must have at least three gate delays and not more than eight gate delays to ensure proper operation. The delay of the latches attached to `fire[2]` may not be long enough.

That's why there are two FIFOs in the experiment. In the lower FIFO I have added some delay in the datapath between stages 12 and 13. This delay should have no effect on the source-limited operation of the FIFOs, but it may be essential to correct operation in the sink-limited case.

ASSIGNMENT

The exercise circuit for your final homework assignment is called `twoFIFOs`. This experiment puts long latency between `fire[3]` and `fire[4]` to make room for a parity circuit that bears the label `par`. You will see all the important action in this circuit in the first 12 nsec of simulation.

The parity circuit computes parity of four input bits, g[1:4]. It has two ranks of exclusive or (XOR) gates. Because these gates need true and complement inputs, there are also some inverters. A final inverter provides for strong output drive.

The pulse generators at the left of the schematic provide master clear (mc) and some fake data signals. The data signal d[1] is always ground. The "J box" isolating d[1] from ground preserves the name d[1] separate from ground. A single pulse generator drives the two signals called d[2] and d[4]. The J box separating the

d[2,4] cable from the pulse generator preserves the distinction between d[2] and d[4] while driving them both from the same pulse generator. The pulse generators provide signals that sometimes have even and sometimes have odd parity.

The most interesting thing about today's assignment is the relationship between `fire[3]` and `fire[4]`. Because these signals are spaced 10 gate delays apart, they seem almost identical. Although the pulses on these two wires seem almost identical, they are very different. Their difference becomes evident only when the pipeline starts or drains.

J BOXES – a feature of Electric

Electric's J box is a very useful all-purpose connector. Its two inputs are connected together electrically but retain their separate names. Moreover, let us suppose that two cables, say A[1:3] and B[1:9] both attach to the same J box. The J box connects their internal wires together in sequence. The first three are obvious: A[1] to B[1], A[2] to B[2], A[3] to B[3]. But that doesn't exhaust the B wires, and so the A's repeat for the next three B's: A[1] to B[4], A[2] to B[5], and A[3] to B[6]. Repeating again, A[1] to B[7], A[2] to B[8] and A[3] to B[9]. Thus B[1,4,7] are all connected to each other and to A[1]. J boxes are very useful for renaming the signals in a bus.

ANSWER SHEET – due by beginning of class on **30 November 2010** **9**

Name _____

Turned in on Date _____

My simulation uses _____ technology – e.g. 180 MOSIS

Answer from simulation:

My simulation shows groups of pulses. The groups start every _____ nsec. Why?

Within each group my simulation shows pulses at `fire[2]` every _____ psec for a throughput of _____GDI/s.

The forward latencies between fire signals in my simulation are:
Hint: measure between similar places on each pulse in the first group.

`fire[2]` to `fire[3]` _____ gate delays for _____ ns

`fire[3]` to `fire[4]` _____ gate delays for _____ ns

`fire[4]` to `fire[5]` _____ gate delays for _____ ns

Each change in the parity signal, `a[1]`, precedes the start of fire[4] by _____ ns.

Each change in the latch outputs, `f[2,3]`, precedes the start of `fire[3]` by _____ ns.

Each change in the latch outputs, `f[6,7]`, precedes the start of `fire[13]` by ____ ns.

Why are `f[6,7]` later than `f[2,3]`?

Answer from the second group of pulses in the simulation:

The reverse latencies between fire signals in my simulation are:

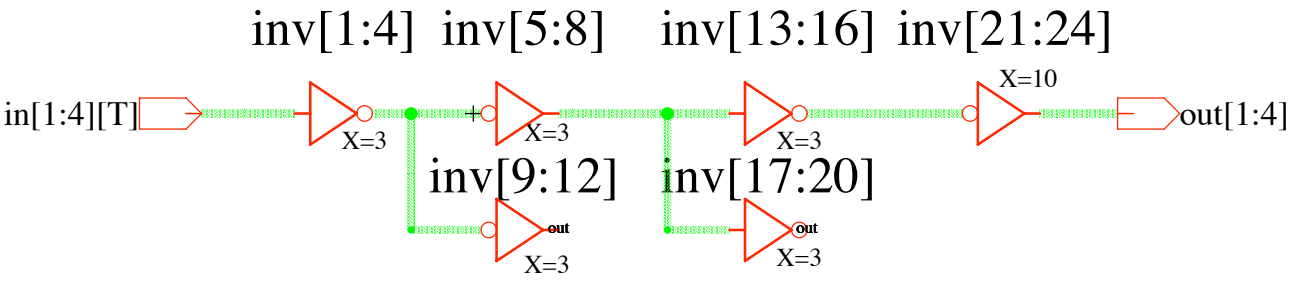
`fire[3]` to `fire[2]` _____ gate delays for _____ ns

`fire[4]` to `fire[3]` _____ gate delays for _____ ns

`fire[5]` to `fire[4]` _____ gate delays for _____ ns

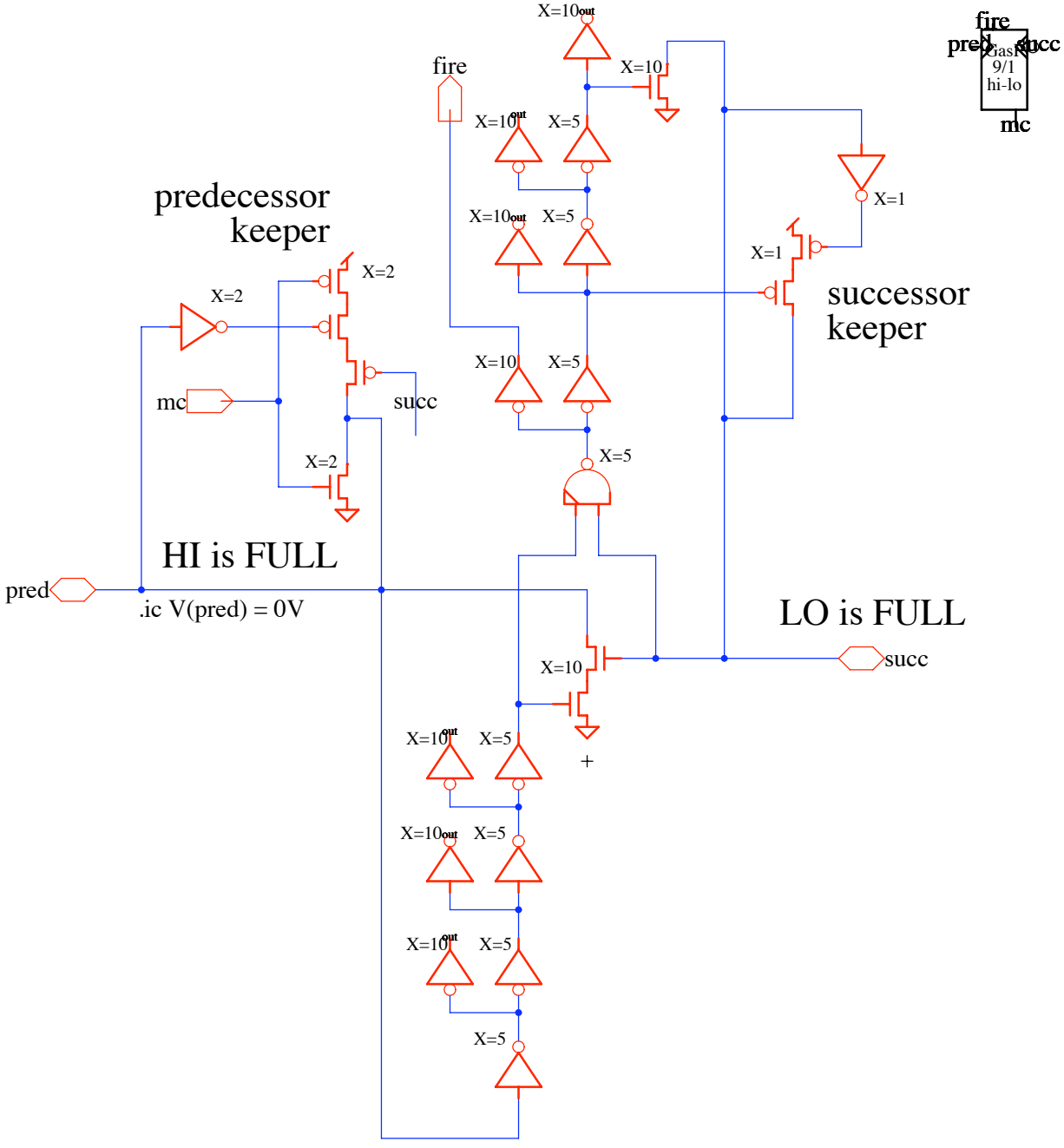
delay

ies 22 November 2010



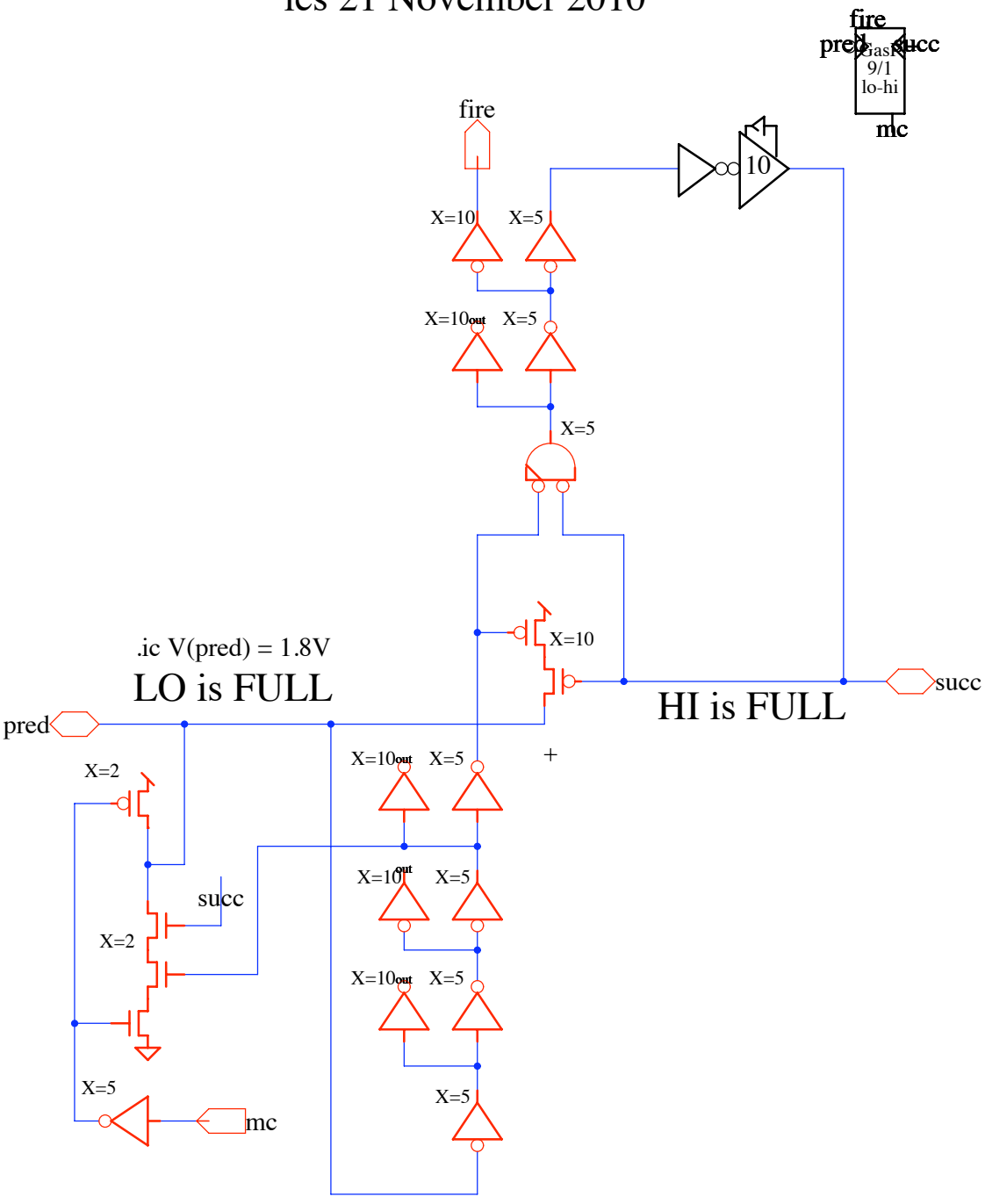
gasp9/1hi-lo

ies 21 November 2010



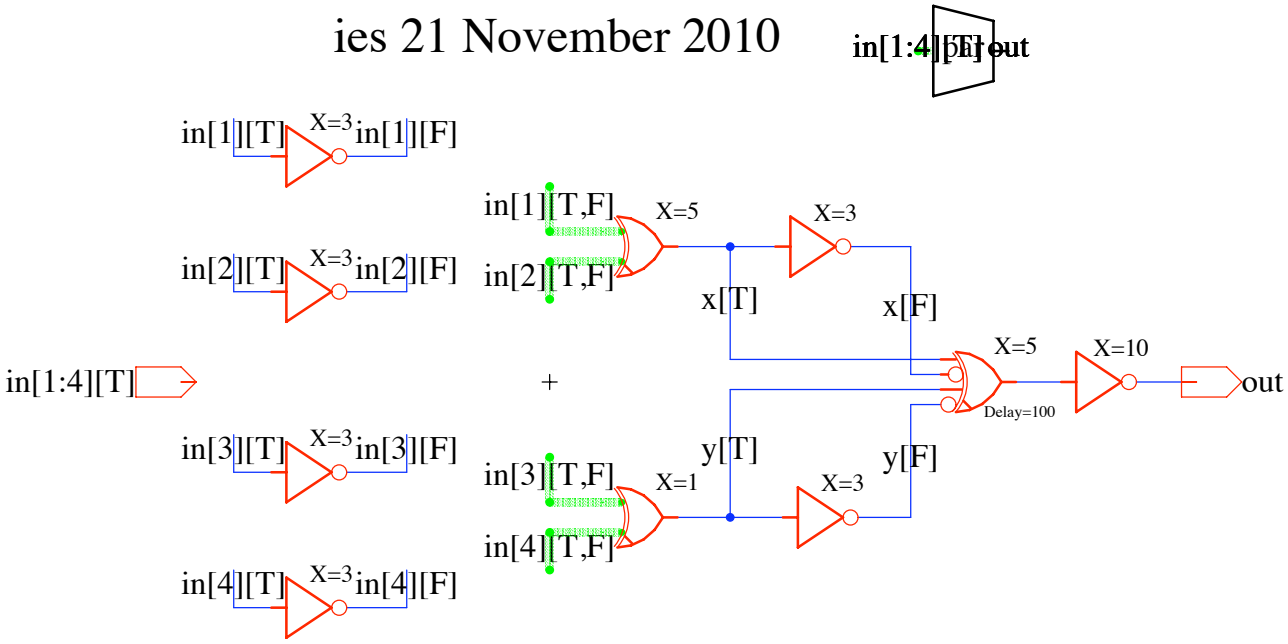
gasp9/1lo-hi

ies 21 November 2010



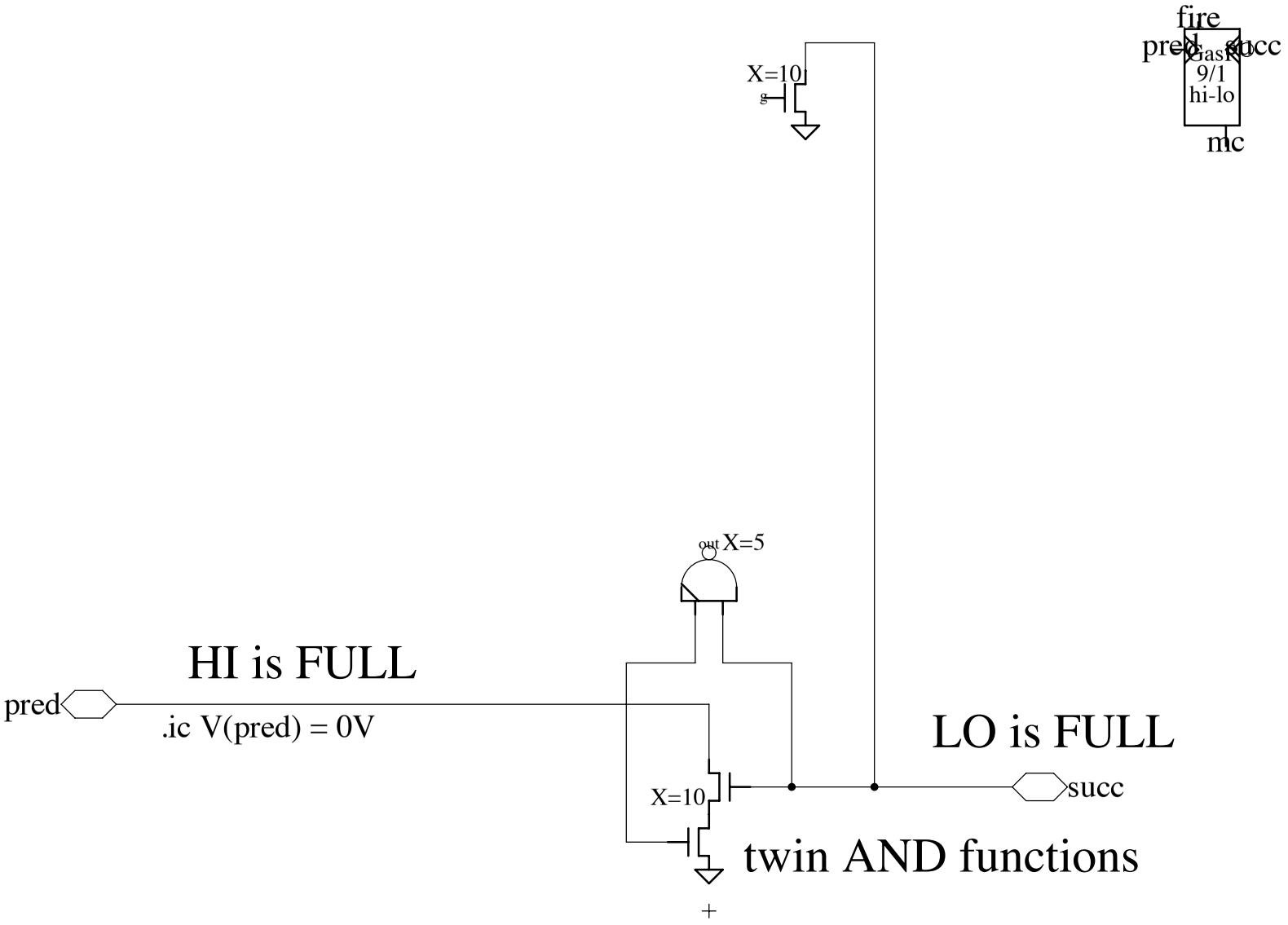
parity

ies 21 November 2010



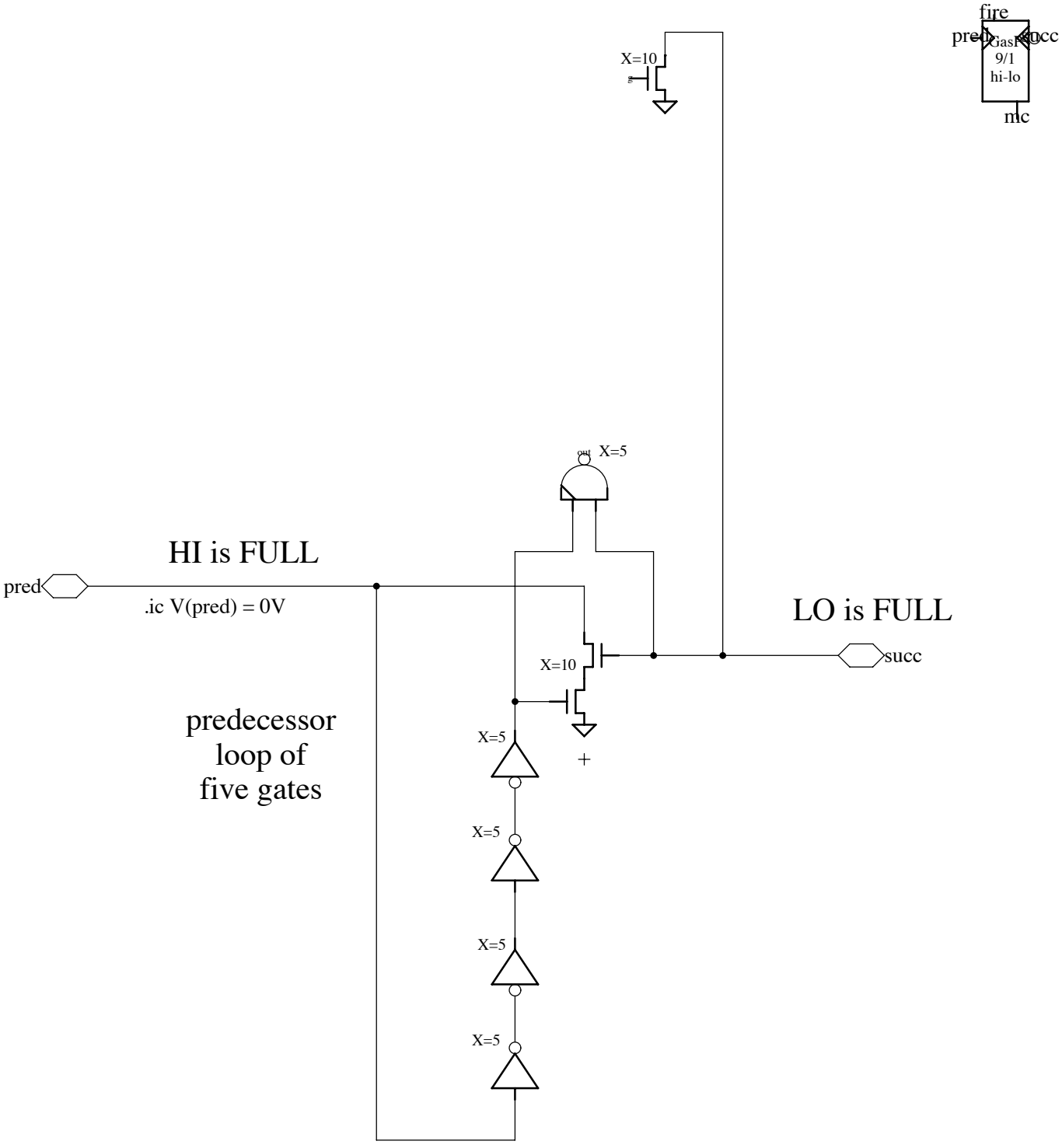
step1ofHILO

ies 22 November 2010



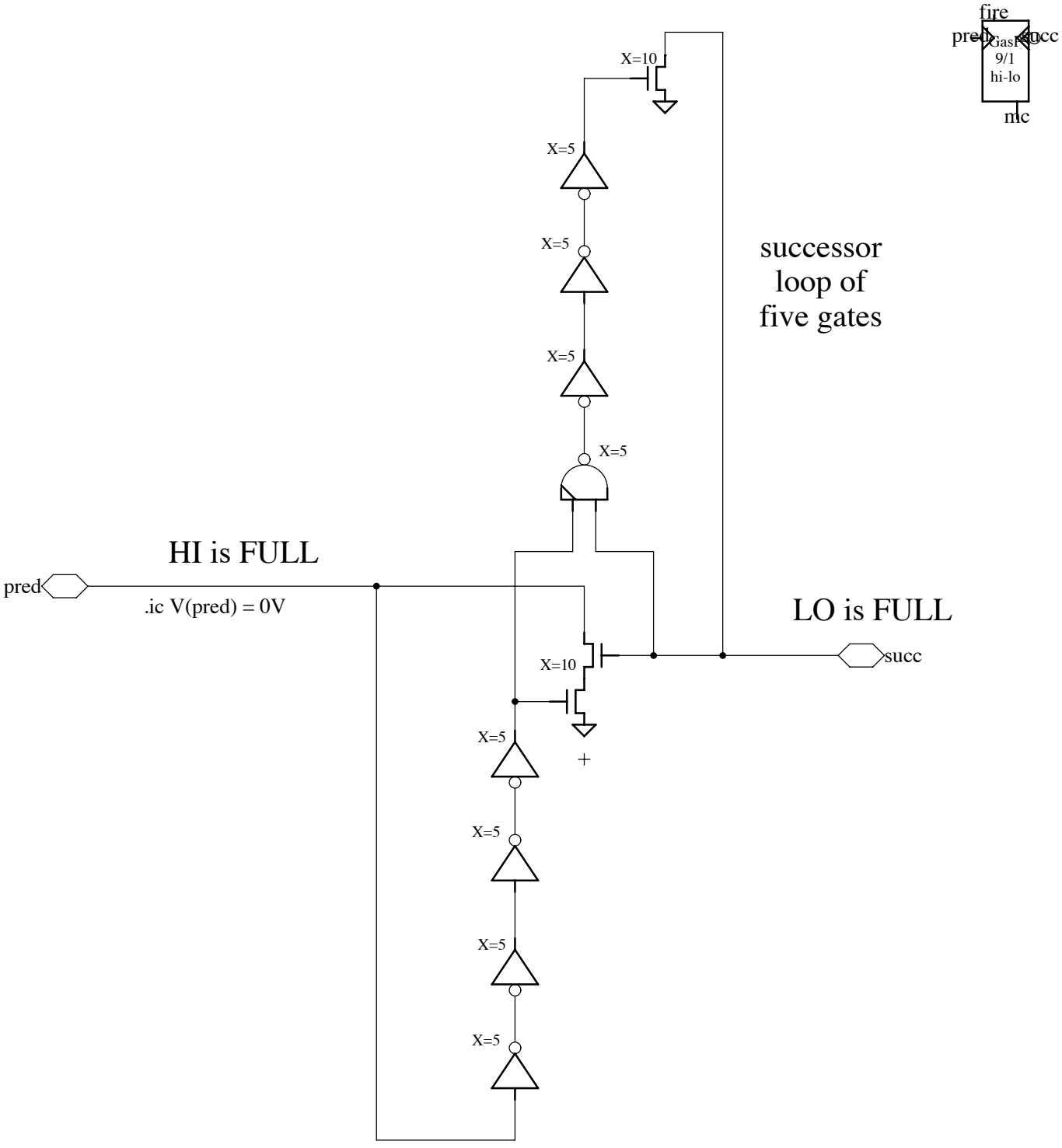
step2ofHILO

ies 22 November 2010



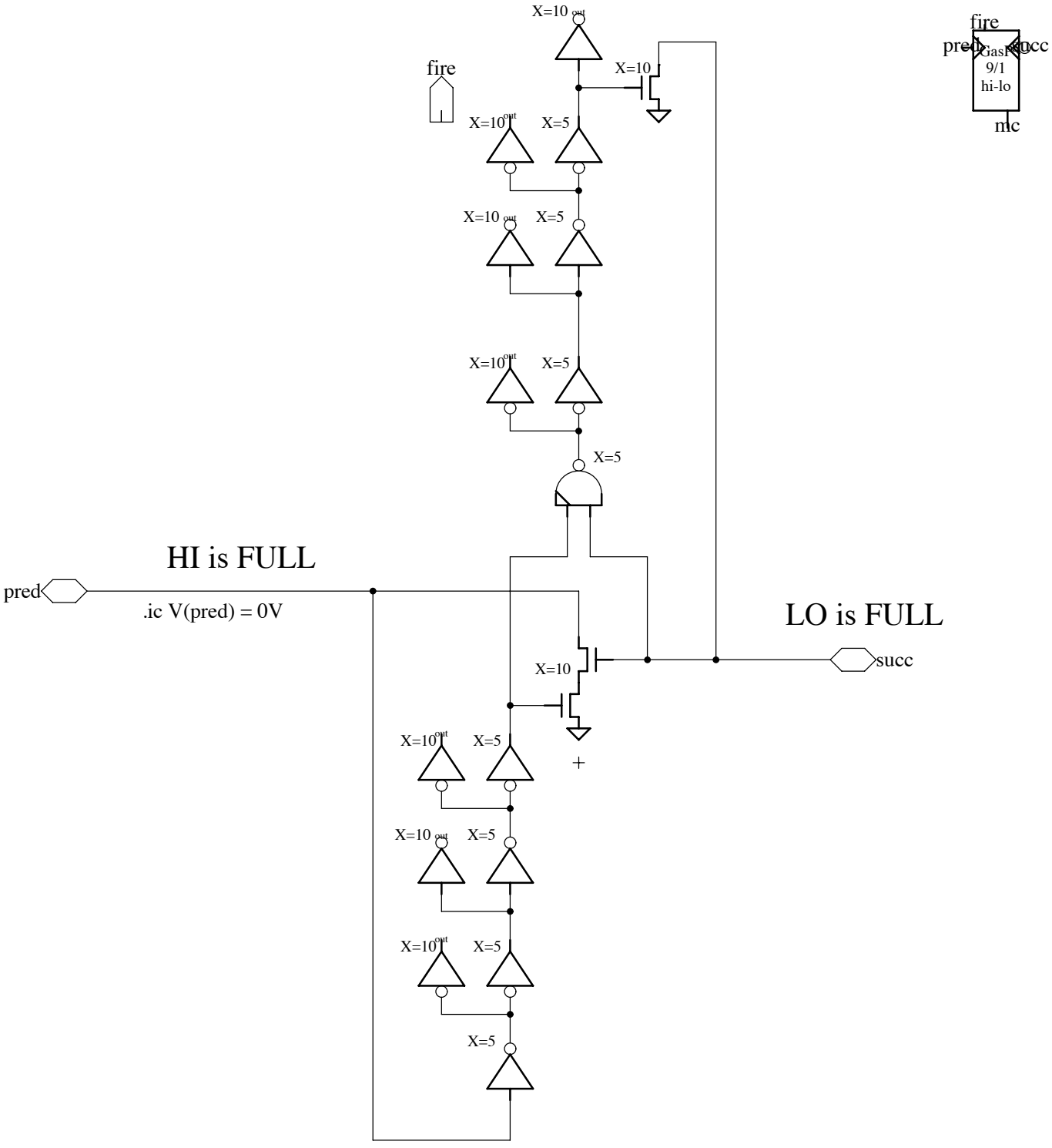
step3ofHILO

ies 22 November 2010



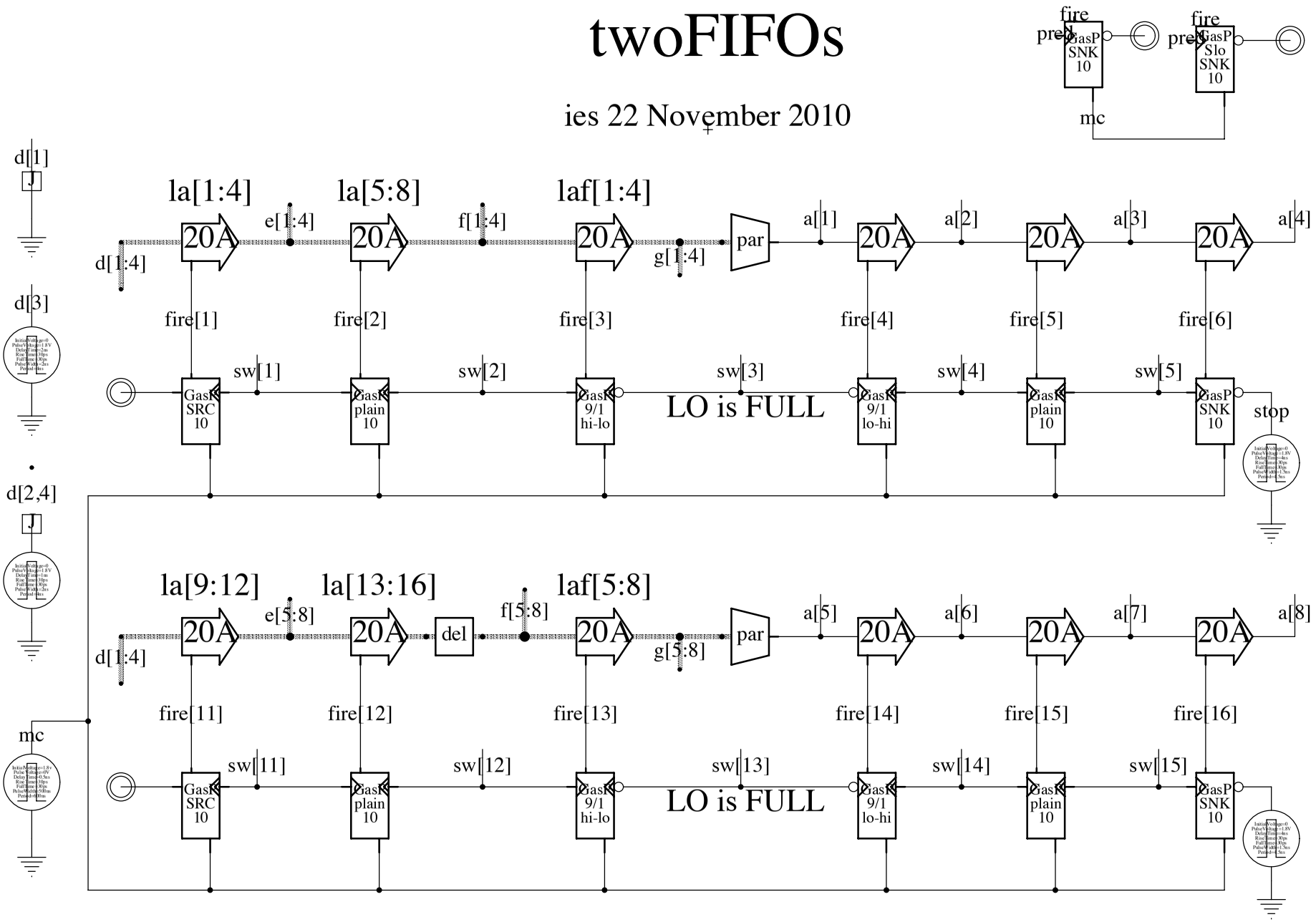
step4ofHILO

ies 22 November 2010



twoFIFOs

ies 22 November 2010



ANSWER SHEET – due by beginning of class on **30 November 2010** **9**

Name _____

Turned in on Date _____

My simulation uses _____ technology – e.g. 180 MOSIS

Answer from simulation:

My simulation shows groups of pulses. The groups start every _____ nsec. Why?

Within each group my simulation shows pulses at `fire[2]` every _____ psec for a throughput of _____GDI/s.

The forward latencies between fire signals in my simulation are:
Hint: measure between similar places on each pulse in the first group.

`fire[2]` to `fire[3]` _____ gate delays for _____ ns

`fire[3]` to `fire[4]` _____ gate delays for _____ ns

`fire[4]` to `fire[5]` _____ gate delays for _____ ns

Each change in the parity signal, `a[1]`, precedes the start of fire[4] by _____ ns.

Each change in the latch outputs, `f[2,3]`, precedes the start of `fire[3]` by _____ ns.

Each change in the latch outputs, `f[6,7]`, precedes the start of `fire[13]` by ____ ns.

Why are `f[6,7]` later than `f[2,3]`?

Answer from the second group of pulses in the simulation:

The reverse latencies between fire signals in my simulation are:

`fire[3]` to `fire[2]` _____ gate delays for _____ ns

`fire[4]` to `fire[3]` _____ gate delays for _____ ns

`fire[5]` to `fire[4]` _____ gate delays for _____ ns