ASYNCHRONOUS RESEARCH CENTER

Portland State University

Subject: **Fifth Class Handout – Broad Branch and Broad Merge**
Date: **October 22, 2010**
From: **Ivan Sutherland**
ARC#: **2010-is52**

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-is44: Class 3 – Linear GasP, Ivan Sutherland, 8 October 2010
ARC# 2010-is44: Class 4 – Proper Stopper, Ivan Sutherland, 15 October 2010

PURPOSE

This memo introduces the broad branch and broad merge GasP modules. We will simulate FIFOs with parallel branches to see how parallel branches affect both latency and throughput.

INTRODUCTION

So far we've seen GasP circuits participating only in linear FIFOs. Each module in a linear FIFO has a single predecessor and a single successor. Each GasP module fires only when two conditions are true: 1) its predecessor proffers data and 2) there is space to hold the fresh data. When a GasP module fires three things happen: A) the fire signal itself copies the data values, B) a "predecessor driver" drains the predecessor state wire, and C) a "successor driver" fills the successor state wire.

This memo shows how to deliver signals from a GasP module to two or more successors. A *branch* module delivers data to more than one successor module. Although there are many kinds of branch modules that we'll visit later on, here we will consider only the *broad branch*. The broad branch module sends a duplicate of each incoming data element to all of its successors. The broad branch module can form the basis for broadcast transmission.

This memo also shows how a GasP module can accept inputs from two or more predecessors. A *merge* module takes input from several predecessors and delivers values to a single successor. Although there are many types of merge modules

that we'll visit later on, here we will consider only the *broad merge*. The broad merge module combines incoming data elements from its inputs and sends the combined data values to its successor. A broad merge module is suitable for control of a binary operation such as addition for which pairs of inputs combine for each output. A broad merge might also be used to concatenate data from multiple sources.

BROAD MERGE

The library called gaspL contains a broad merge GasP circuit called `gasp2src10`. It has two predecessor state wires, `predA` and `predB`, and a single successor state wire, `succ`. I don't much like its name or its icon, so I have copied the circuit into this week's library: `Class5-brMr.jelib`. I also changed its name to `gaspBdMr10` and improved its icon. The letters BdMr in the new name remind us that this is a broad merge and not some other type.

In place of the inverter usually connected to the input state wire `gaspBdMr10` has a NAND gate. Before the `fire` pulse can appear, both predecessor state wires must be HI, meaning FULL, and the successor state wire must be LO, meaning EMPTY. Thus three, rather than two, conditions must be true before the `fire` pulse can happen. The fire pulse evokes the usual three actions, but it can accept data from either or both of its predecessors.

Notice that I used a symmetric NAND gate in `gaspBdMr10` for the two predecessor state wire inputs, `predA` and `predB`. I used a symmetric NAND gate to make the two predecessor inputs as similar as possible. An asymmetric NAND gate would also provide the proper logic at the cost of slightly different delays for the two inputs. I like symmetry.

Two predecessor drivers appear in `gaspBdMr10` at the left of the circuit, one for `predA` and one for `predB`. These predecessor drivers act independently. During the `fire` pulse each predecessor driver drives its state wire LO to drain it. One predecessor driver might take longer to drain its state wire than the other. The time each takes to drain its predecessor state wire depends on the load on the state wire.

The broad merge requires only one successor driver. During the `fire` pulse the successor driver fills the successor state wire by driving it HI. Whichever filling or draining process finishes first terminates the `fire` pulse.

The library called gaspL has two broad merge circuits. The broad merge we use here, `gaspBdMr10` is patterned after `gasp2src10`, and has size 10 predecessor and successor drivers and a size 40 central AND column to drive its `fire` pulse. The other broad merge circuit in gaspL is called `gasp2pred1src`. It has stronger drive transistors but the same topology. I don't like the names of either of these circuits in gaspL nor the mismatch between their icons and the "standard" GasP icon that appears in `gaspPlain10`. I like the names and icons I've assigned for this lesson better. Our

libraries reflect the history of accumulating circuits; errors and inconsistencies also accumulate over time.

BROAD BRANCH

The library called gaspL fails to provide any broad branch GasP module. There is no reason for this omission from gaspL. I've put a broad branch module called `gaspBdBr10` into this week's library: `Class5-brMr.jelib`. The broad branch module, `gaspBdBr10`, is the mate to the broad merge module, `gaspBdMr10`. Perhaps both of these will appear in a later edition of the library. For now, however, two broad branch modules called `gaspBdBr10` and `gasp2sucThree` appear as facets in the library for this class assignment.

The broad branch module, `gaspBdBr10`, has two successor state wires called `succA` and `succB` and a single predecessor state wire called `pred`. It can fire only when both successor state wires are LO meaning EMPTY and its predecessor state wire is HI meaning FULL. So like the broad merge, the broad branch requires that three conditions hold before it will fire. It requires a three-input AND function.

The broad merge module, `gaspBdMr10`, had an obvious place to put an extra NAND gate to accommodate the extra predecessor state wire. For the broad branch module it's less obvious how to add an extra successor input to the AND function of the central column. One could just replace the usual two-input NOR with a three-input NOR as shown in `gasp2sucThree`. I prefer the alternative that appears in `gaspBdBr10`. In `gaspBdBr10` I have distributed the AND function over two stages of amplification to avoid using a three-input gate.

Does it matter which of the three bubble inputs in `gaspBdBr10` serves which of the state wires? I have chosen to use the symmetric inputs of a two-input gate for the two successor state wires. I like symmetry. Although my choice preserves symmetry, any connection of state wires to the three bubble inputs would be logically equivalent. Moreover, the two-input gate I chose for the two successors need not be symmetric.

The two successor drivers at the right of `gaspBdBr10` each fill their successor state wires independently. How long it takes for each to fill its state wire depends, of course, on the load on the state wire. The predecessor driver, meanwhile, drains the predecessor state wire. Whichever filling or draining process finishes first terminates the `fire` pulse.

SIMULATION ASSIGNMENT

The simulation assignment for this week involves the three FIFOs that appear in the facet called `bdBrMr`. Each FIFO begins with a source module and ends with a sink module, and so we might expect each of them to run at full speed. Your simulation will show whether or not they meet that expectation.

The top FIFO, with fire signals `fire[1:10]` has two GasP modules in each parallel branch, namely the modules that produce `fire[4:5]` and `fire[6:7]`. Symmetry suggests that the top FIFO will run at full speed. We might expect the latency through the 5 stages from `fire[2]` to `fire[9]` to be six gate delays per stage, or a total of 30 gate delays. At ten gate delays per cycle, we might expect, on average, three data elements and two bubbles to occupy this stretch of FIFO. That count, of course, assumes that the data elements and bubbles in each branch match.

The middle and bottom FIFOs have branches whose lengths differ. The upper branch of the middle FIFO has only two GasP modules, namely the ones that produce `fire[14:15]`. The lower branch has three GasP modules that produce `fire[16,x,17]`. How does this imbalance in length affect its throughput?

The bottom FIFO, `fire[21:29]`, has branches that differ even more in length. Its upper branch has three GasP modules producing `fire[24:26]` and its lower branch has six Gasp modules producing `fire[o,p,q,r,s,t]`. How does this greater imbalance in length affect its throughput?

Your simulation will reveal what throughput the middle and lower FIFOs offer. Your task is to understand why their throughputs differ. You know how to look at the inputs to the AND function to see what's retarding the `fire` pulse from a GasP module. Look at the AND inputs in module 27. Which input is late? Also look at the AND inputs in module 23. Which input is late?

Think about how long it takes for signals to pass through a GasP module. The forward latency is six gate delays per stage; the reverse latency is four gate delays per stage. How long does it take for the `succA` terminal (the upper one) of module 13 to notice that its state wire is LO meaning EMPTY?

ANSWER SHEET – due **2 November 2010** **5**

Name _____

Turned in on Date _____

Answer by examining the circuits – no simulation required for these questions:

The first `fire[9]` pulse must follow the `fire[2]` pulse by about _____ gate delays. (hint – read the text)

The first `fire[27]` pulse must follow the `fire[23]` pulse by about _____ gate delays. (hint – count the longer path)

After the first `fire[22]` pulse the next `fire[27]` pulse can't happen until about _____ gate delays later. (hint – count the reverse path)

Answer from simulation:

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

My simulation shows pulses at `fire[9]` every _____ psec for a throughput of _____GDI/s.

What pattern of pulses do you see at `fire[19]`? `Why?`

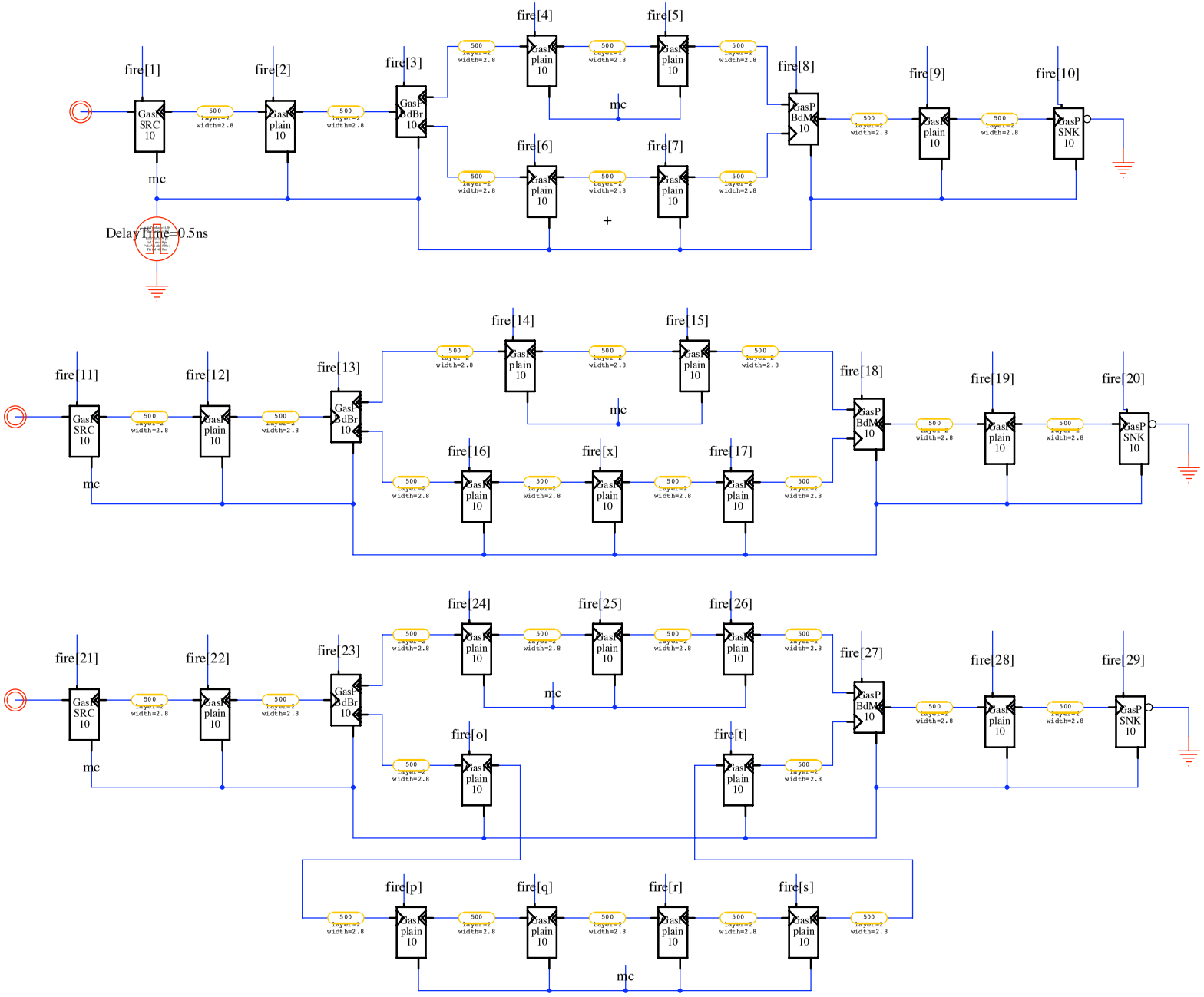
What pattern of pulses do you see at `fire[28]`? `Why?`

My simulation shows an average throughput at `fire[19]` of _____ GDI/s.

My simulation shows an average throughput at `fire[28]` of _____ GDI/s.

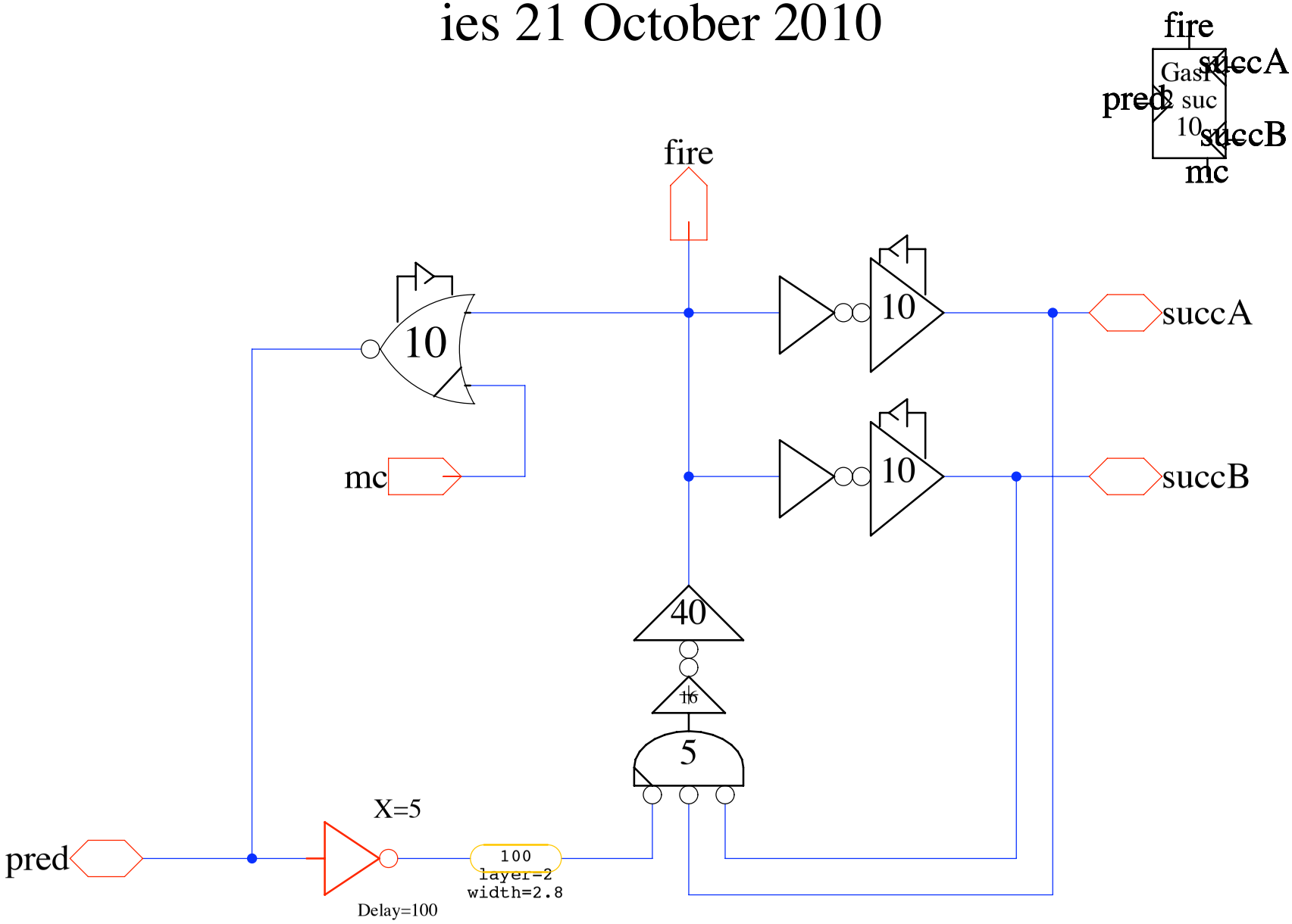
bdBrMr

ies 21 October 2010



gasp2sucThree

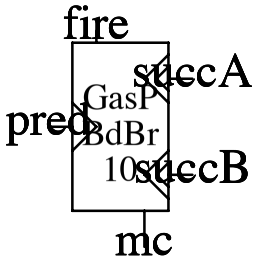
ies 21 October 2010

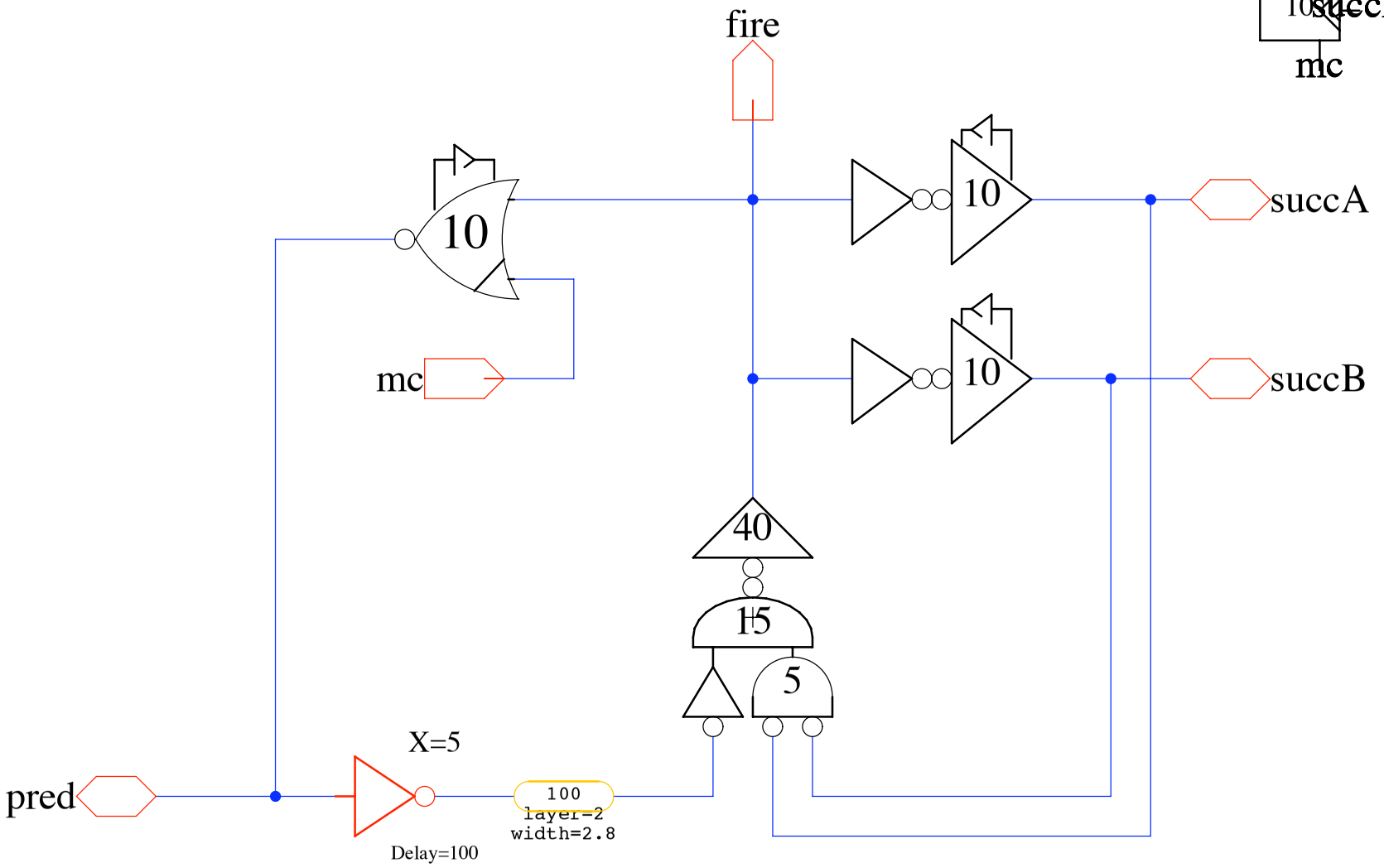


gaspBdBr10

ies 21 October 2010

a broad branch (BdBr) GasP module

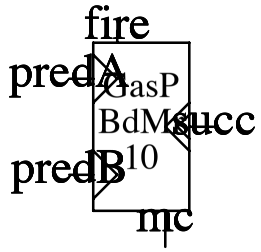


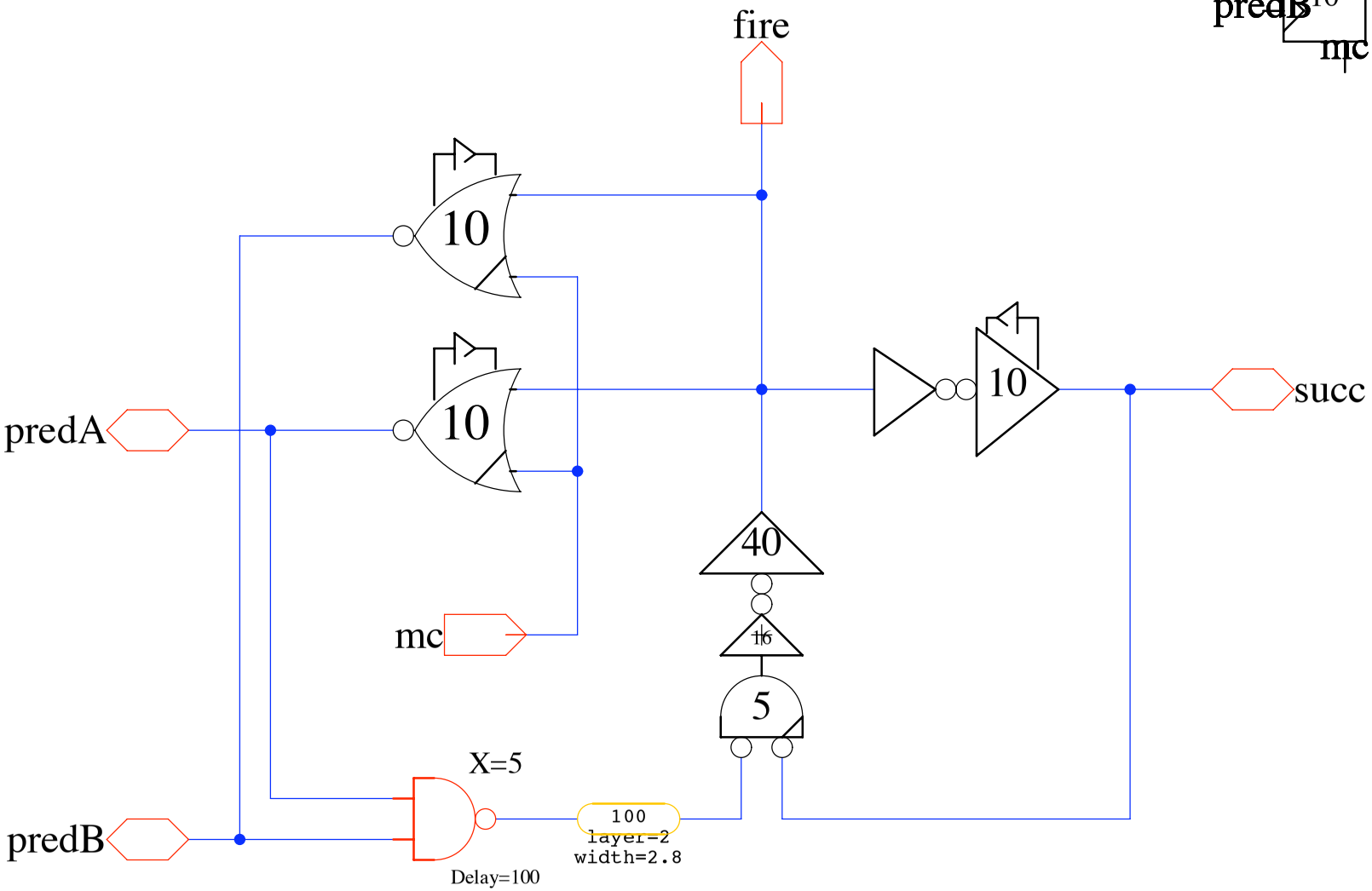


gaspBdMr10

ies 21 June 2010

a broad merge (BdMr) GasP module





ANSWER SHEET – due **2 November 2010** **5**

Name _____

Turned in on Date _____

Answer by examining the circuits – no simulation required for these questions:

The first `fire[9]` pulse must follow the `fire[2]` pulse by about _____ gate delays. (hint – read the text)

The first `fire[27]` pulse must follow the `fire[23]` pulse by about _____ gate delays. (hint – count the longer path)

After the first `fire[22]` pulse the next `fire[27]` pulse can't happen until about _____ gate delays later. (hint – count the reverse path)

Answer from simulation:

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

My simulation shows pulses at `fire[9]` every _____ psec for a throughput of _____GDI/s.

What pattern of pulses do you see at `fire[19]`? `Why?`

What pattern of pulses do you see at `fire[28]`? `Why?`

My simulation shows an average throughput at `fire[19]` of _____ GDI/s.

My simulation shows an average throughput at `fire[28]` of _____ GDI/s.