ASYNCHRONOUS RESEARCH CENTER

Portland State University

Subject: Sixth Class Handout – Round Robin FIFO
Date: November 1, 2010
From: Ivan Sutherland
ARC#: 2010-is53

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
Jo Ebergen, Squaring the FIFO in GasP, Proc IEEE Conf on Asynchronous Circuits and Systems, (ASYNC 2001), Salt Lake City, UT. ISBN: 0-7695-1034-5

PURPOSE

This memo considers a round robin pipeline. Last week we used broad branch and broad merge modules to split a pipeline into two parallel paths. All data elements flowed through both paths in parallel. In contrast, a round robin pipeline uses its parallel paths in turn, sending each successive data element down the next path in round robin fashion. It repeats only when it has served all paths. We will see how GasP elements can select branches in sequence.

BACKGROUND

The broad branch and broad merge GasP modules let us divide a pipeline into two parallel parts. Such a division is useful to separate the bits of each data element into two parts for separate processes and later on to reassemble them.

One place where such a division is useful is as part of a memory system. A read request to a computer memory is really a transfer request. It says, in effect, "copy the content of memory register X into fast register Y." Such a request must carry two addresses: an address in memory, X, and a fast register name, Y. The address in memory, X, generally has many more bits than the register name, Y.

It takes many steps to fetch data from memory. The memory address is usually a virtual address used by the software. First, the hardware must confirm that the

This document contains information developed at the Asynchronous Research Center at Portland State University. You may disclose this information to whomever you please. You may reproduce this document for any not-for-profit purpose. Reproduction for sale is strictly forbidden without written consent of the author. Copies of the material must contain this notice.

software has permission to access that memory location. If there is access permission, the hardware must next map the virtual address into a real hardware address in memory. Next, to fetch the value stored at that memory location requires sending the request to the proper memory bank. Finally, the chosen value emerges from memory.

In the mean time, what has happened to the register name, Y? The memory system must hold the name Y until the selected value destined for Y emerges from memory. One register to hold Y is inadequate for a pipelined memory system that can accept new memory requests before finishing an earlier request. Pipelined memory systems must store several register names to guide distribution of several values as they emerge from memory.

The parallel pipeline that we saw last week is ideal for this service. One branch goes through the memory system. The other branch is just a First In First Out (FIFO) memory with enough capacity to store the register names. The register name FIFO delivers the names in sequence. The broad merge module combines the memory data with the register names for proper delivery.

ROUND ROBIN BRANCH

This memo considers an entirely different kind of branch. Instead of dividing each data element into two parts, a round robin branch sends successive data elements into different paths. A two-way alternating branch of this kind serves two paths, each at half the data rate of the input. Thus it can offer twice as much time for each branch to process data. A three-way round robin branch offers three times as much time for each of its branches to process data. A round robin branch is useful for improving throughput by dividing the total work among several similar processing paths.

A two-way round robin branch is called an “alternate branch” or a “wig-wag.” It has a single predecessor for input and two successors to take its output alternately. A flip-flop in an alternate branch can remember which output to service next. The library called gaspL offers GasP modules for alternate branch and alternate merge. You can examine them at your leisure.

Rather than examining the alternate branch, however, I want here to consider instead a more general form: an N-way branch. We will simulate a composite N-way branch made from the broad branch and broad merge modules we used last week. We’ll use N=3 for illustration, but the circuit generalizes to any N. A three-way branch circuit appears under the name `roundRobin` in the schematic diagrams for this memo.

I have set up the `roundRobin` circuit for you to simulate. Only the center six gasp modules that produce `fire[3:8]` form the round robin circuit itself. To their left are a source module, `fire[1]`, and a plain GasP module, `fire[2]`, providing input to the round robin circuit. To its right are three sink modules, `fire[9:11]`, to absorb the three outputs. Before you run the simulation, let’s examine some details.

In the left column, generating fire[3:5], are three gaspBdBr broad branch modules. In the right column, generating fire[6:8], are three gaspBdMr broad merge modules. Between the columns for fire[3:5] and fire[6:8] are a total of six state wires. The three horizontal state wires, sw[3:5], are the familiar state wires that couple horizontally to the next pipeline stage. The three vertical state wires, r[1:3], remember whose turn is next. Notice that r[1] appears both at the top and the bottom of the column to make a circular connection. In a two-way branch a flip-flop serves for this memory. The more general case considered here is more interesting.

Master clear sets r[1] to LO and both r[2] and r[3] to HI. Master clear does this because the gaspBrMr10icHI modules for fire[6:7] initialize their lower input HI rather than LO. Each data element that passes through the N-way branch advances the LO-HI-HI pattern in r[1:3] one position around the ring.

Here's how that advance happens. When fire[3] happens, its predecessor logic drains sw[2] and it's successor logic fills r[1] and sw[3]. When sw[6] is LO, fire[6] will follow six gate delays after fire[3], draining sw[3] and r[2]. Notice that fire[3] filled r[1] but fire[6] drained r[2]. The combination of fire[3] and fire[6] advances the LO-HI-HI pattern on r[1:3].

Moreover, the combined action takes 10 gate delays. The forward action on sw[3] takes six gate delays and the reverse action on r[2] takes four gate delays. And so fire[4] can happen just 6+4=10 gate delays after fire[3]. The round robin circuit works fast enough to keep up with the 10 gate-delay cycle of the input pipeline.

Simulate roundRobin and see it in operation. On fire[2] you should see a uniform series of pulses. One third of these will produce pulses on fire[3], one third will produce pulses on fire[4] and one third will produce pulses on fire[5]. I find the pattern of pulses on fire[3:5] satisfying.

In the upper left of roundRobin you will some extra GasP modules tied off with ground symbols to keep them from consuming simulation time. In addition to the ordinary source and sink modules, I've provided a slow source module, gaspSlowSource, and a slow sink module, gaspSlowSink. Each of these has an extra loop of inverters to delay its response.

The astute observer will notice that a gaspSlowSink module produces fire[9]. Does the extra delay of this slow sink retard the pulses on fire[6:8]?

There's a problem with roundRobin that deserves comment. Why is the wave form on sw[2] ugly? The state wire called sw[2] is filled when fire[2] happens and drained whenever any of the fire[3:5] pulses happens. The wired OR on sw[2] puts the pull-down transistors that drain sw[2] in parallel, albeit they are in different GasP modules. Because each of those GasP modules also has a keeper, there are three keepers wired in parallel all of which oppose draining sw[2]. The

roundRobin circuit works only because the pull down transistors on sw[2] are strong enough to overwhelm the opposing keepers. The result is the ugly waveform on sw[2]. The next circuit fixes this deficiency.

ROUND ROBIN MERGE – wigWagWog

The same circuit with only minor modification can form an N-way merge. In the circuit diagram called wigWagWog you will recognize two round robin configurations, one with vertical state wires x[1:3] and the other with vertical state wires y[1:3]. You will also see a three-way predecessor driver near the left of the schematic and a three-way successor driver near the right of the schematic. The wigWagWog circuit takes a flow from the source at fire[1], splits it into three separate paths: (3 to 18), (4 to 19), and (5 to 20), and recombines the three paths to reach the sink at fire[22].

The astute observer will notice that a slow sink produces fire[22].

To eliminate the parallel OR on sw[2], I used gaspBdBr10-pred modules to produce fire[3:5]. As the name "-pred" suggests, these modules are minus the predecessor driver. They sense the state of sw[2] but refrain from driving it. Instead, the three-way predecessor driver called pred3OrDri10wMC combines fire[3:5] to drain sw[2]. Similarly, the stages that produce fire[18:20] are of types that omit successor drivers. These stages rely on the three-way successor driver at the right, suc3OrDri10, to combine fire[18:20] to fill sw[18]. These circuits avoid the keeper fight on sw[18].

You must run your simulation of wigWagWog for at least 25 nsec to find the effects I want you to observe. For my 25 nsec of simulated time SPICE took a little over a minute and produced about 15 Mega-bytes of output.

The ordinary source at the left of wigWagWog can deliver data slightly faster than the slow sink at the right will accept it. The difference in data rate gradually fills the FIFO. I found it instructive to observe the filling process by looking at the state wires inside the three separate paths. Remember that a HI state wire means FULL and a LO state wire means EMPTY. If a state wire is mostly LO its stage is usually EMPTY. If a state wire is mostly HI its stage is usually FULL. I think you'll find the wave forms on sw[15, 12, 9, 6, 3] interesting and instructive. Note that I've listed them in reverse order.

After you've answered the questions about wigWagWog with the slow sink, replace the slow sink with an ordinary sink. You can easily change the type of sink by copy (C) and paste (V). Electric will let you substitute modules that have matching exports by copy and paste.

WHAT'S IT GOOD FOR?

A round robin pipeline has two important uses. First, as a FIFO it can provide storage capacity with low latency. The latency of wigWagWog from fire[2] to

`fire[2]` is seven stages or about 7*6 = 42 gate delays. The capacity, including stages 2 and 21, is 20 data items. Adding parallel paths adds capacity but little or no latency.

Lower latency also improves energy efficiency. Because each data element moves along only one of the three parallel paths, it passes through fewer GasP stages and therefore consumes less energy than it would in a linear FIFO of equal capacity.

There are many other ways to split a FIFO into parallel paths to get more capacity with reduced latency. A tree of alternating branch and alternating merge modules can divide a FIFO into many parallel paths. Three levels of binary tree provide an eight-way split. Jo Ebergen's paper called "Squaring the FIFO" describes a way to pass data serially along one path, in parallel down many paths, and recover it into a serial stream. Ebergen's arrangement is much like the serial-parallel-serial (SPS) shift registers commonly used in charge-coupled device cameras.

The second important use of round robin branching is for parallel processing to increase speed. Suppose we have a stream of data to process, but the processing equipment we have is too slow to keep up. We can use multiple serving stations to increase throughput.

Here's a common example. Consider a buffet line for serving food. If the buffet table is against a wall, there is a single serving opportunity because people can pass along only the exposed side to collect food. A buffet table standing away from the wall, however, provides two serving opportunities as diners can pass down both sides.

Round robin service is most useful when the parallel pipelines can proceed at about the same pace. Because it reassembles the data stream into its original sequence, a round robin merge is sensitive to delay in any of its branches. If some data values require much longer processing times than others, a demand-driven scheme is preferable. A later lesson will consider how to build a first-come-first-served system.

ANSWER SHEET – due **9 November 2010** **6**

Name ______________________________Turned in on Date _______________________

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

Answer from simulation of `roundRobin`:

Immediately after master clear r[1] = <HI> <LO> , r[2] = <HI> <LO> , r[3] = <HI> <LO> .
Why? Hint: examine the circuits carefully.

My simulation shows pulses at `fire[2]` every ______________ psec for a throughput of ______________GDI/s.

My simulation shows pulses at `fire[3]` every ______________ psec for a throughput of ______________GDI/s.

The pulses on `fire[3:5]` interdigitate (smoothly) (irregularly).

I (can) (cannot) see any hesitation from the slow sink at fire[9]. Explain why.

Answer from simulation of `wigWagWog (with slow sink)`:

My simulation shows pulses at `fire[2]` every ______________ psec for a throughput of ______________GDI/s.

My simulation shows pulses at `fire[21]` every ______________ psec for a throughput of ______________GDI/s.

State wire `sw[16]` changes from mostly empty to mostly full after _______ of its cycles.

State wire `sw[13]` changes from mostly empty to mostly full after _______ of its cycles.

State wire `sw[10]` changes from mostly empty to mostly full after _______ of its cycles.

I detect a change in the pattern on `fire[2]` after ______________ nsec. Why?

Answer from simulation of `wigWagWog (with ordinary sink)`:

State wire `sw[13]` is now mostly (full) (empty).

State wire `sw[16]` is now mostly (full) (empty).

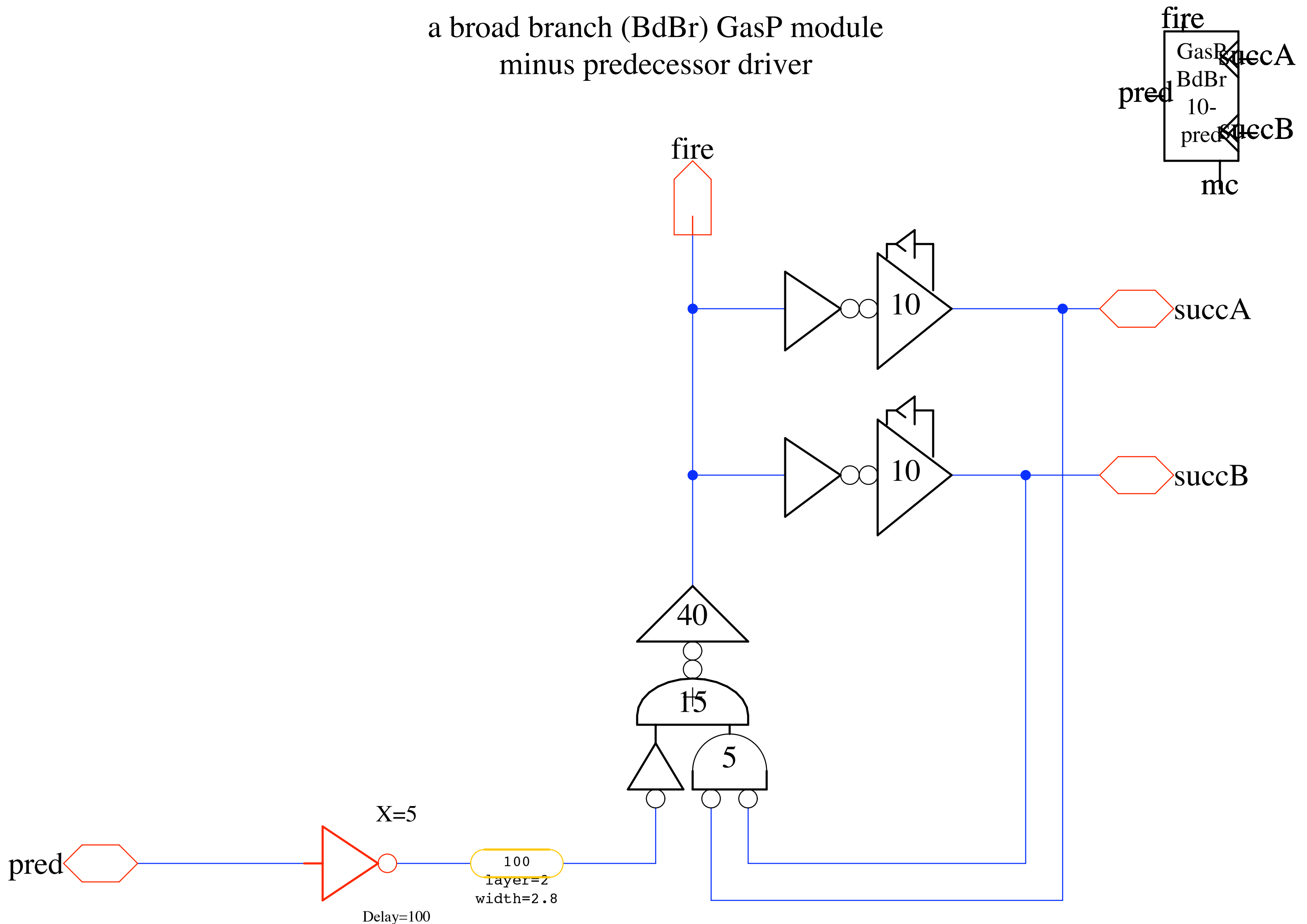
The slowest stage is now stage __________ or __________ or __________.

The holdup is state wire __________.

gaspBdBr10-pred

ies 23 October 2010

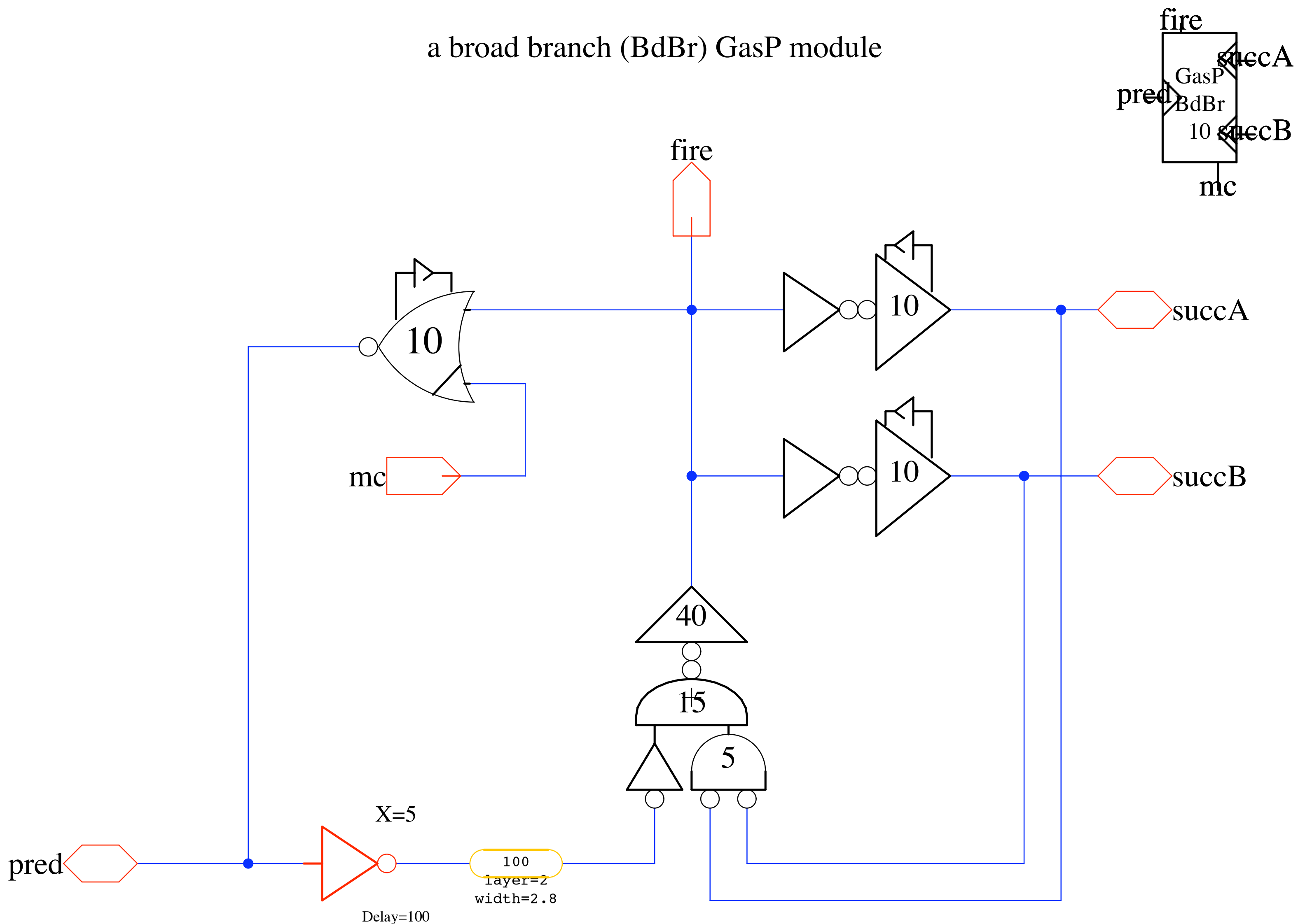
a broad branch (BdBr) GasP module
minus predecessor driver



gaspBdBr10

ies 21 October 2010

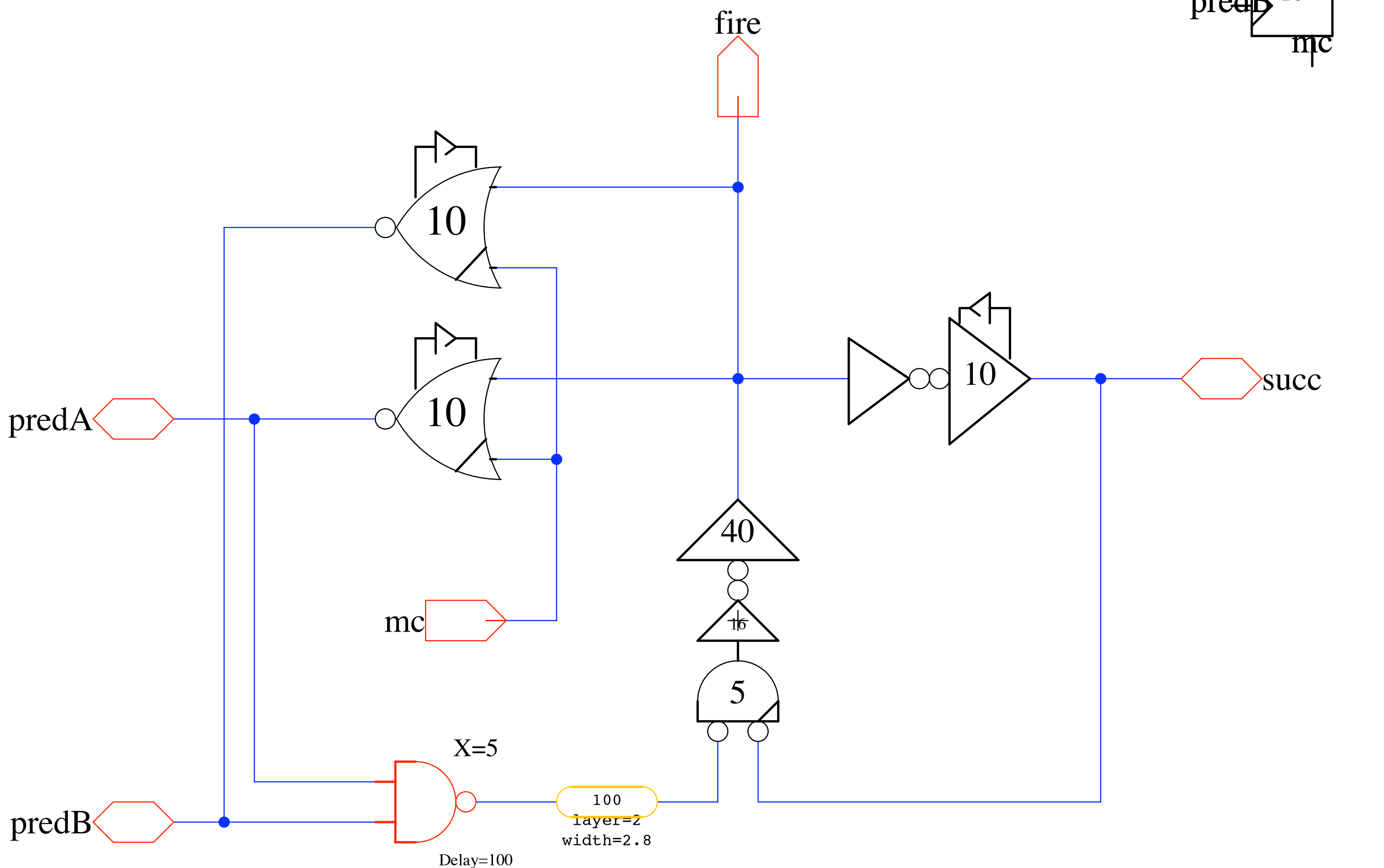
a broad branch (BdBr) GasP module



gaspBdMr10

ies 21 June 2010

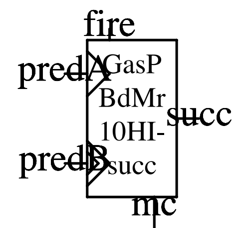
a broad merge (BdMr) GasP module



gaspBdMr10icHI-succ

ies 23 October 2010

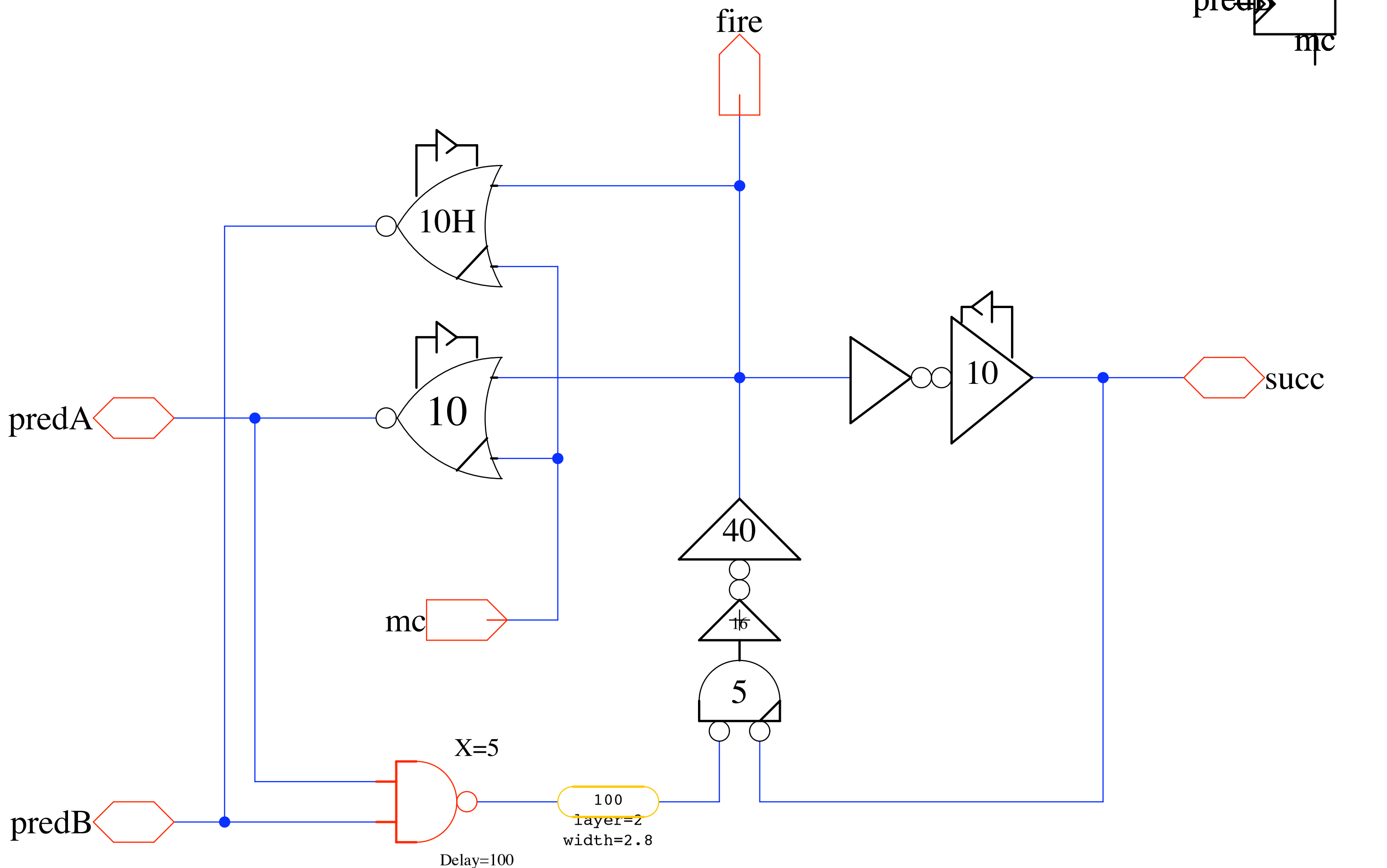
a broad merge (BdMr) GasP module



gaspBdMr10icHI

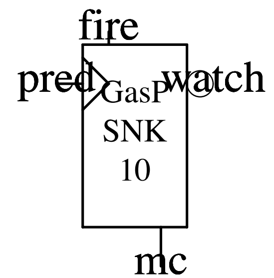
ies 23 October 2010

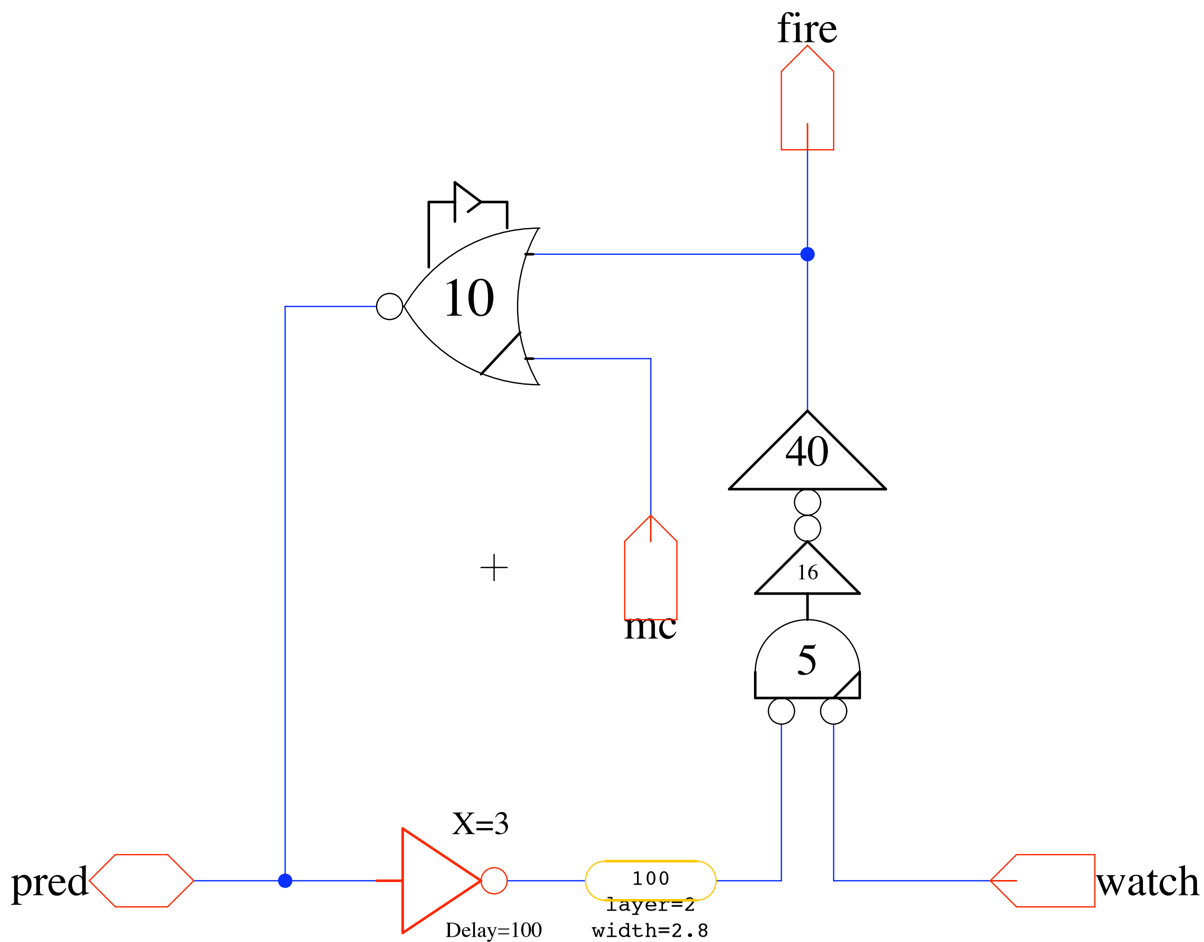
a broad merge (BdMr) GasP module



gaspSink10

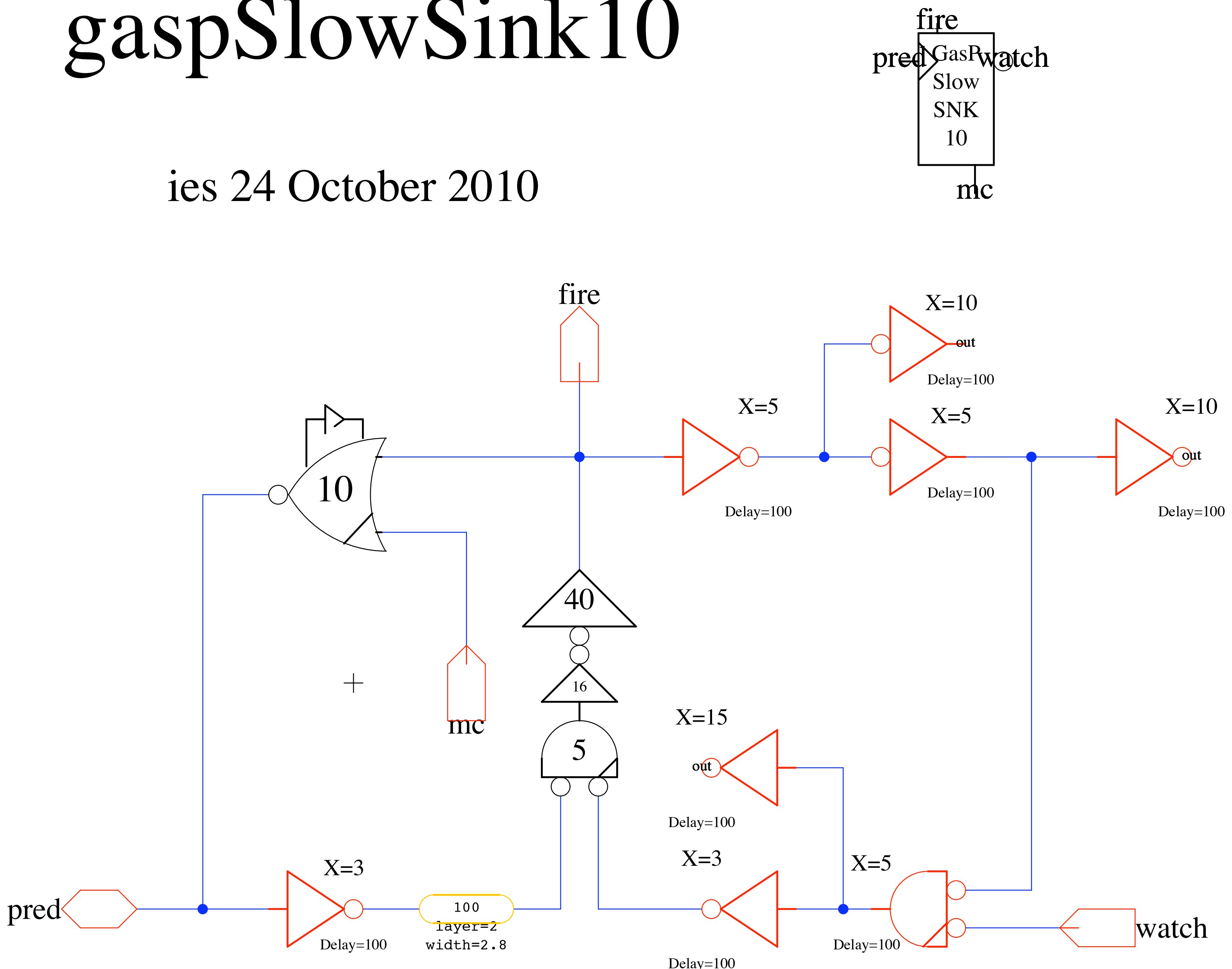
ies 17 June 2010





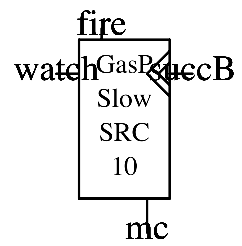
gaspSlowSink10

ies 24 October 2010



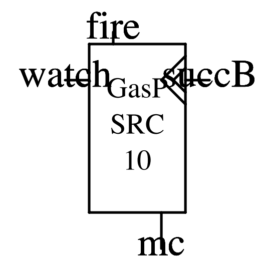
gaspSlowSource10

ies 24 October 2010



gaspSource10

ies 19 June 2010



fire

40

16

10

5

watch

X=3

Delay=100

100
layer=2
width=2.8

succB

X=2

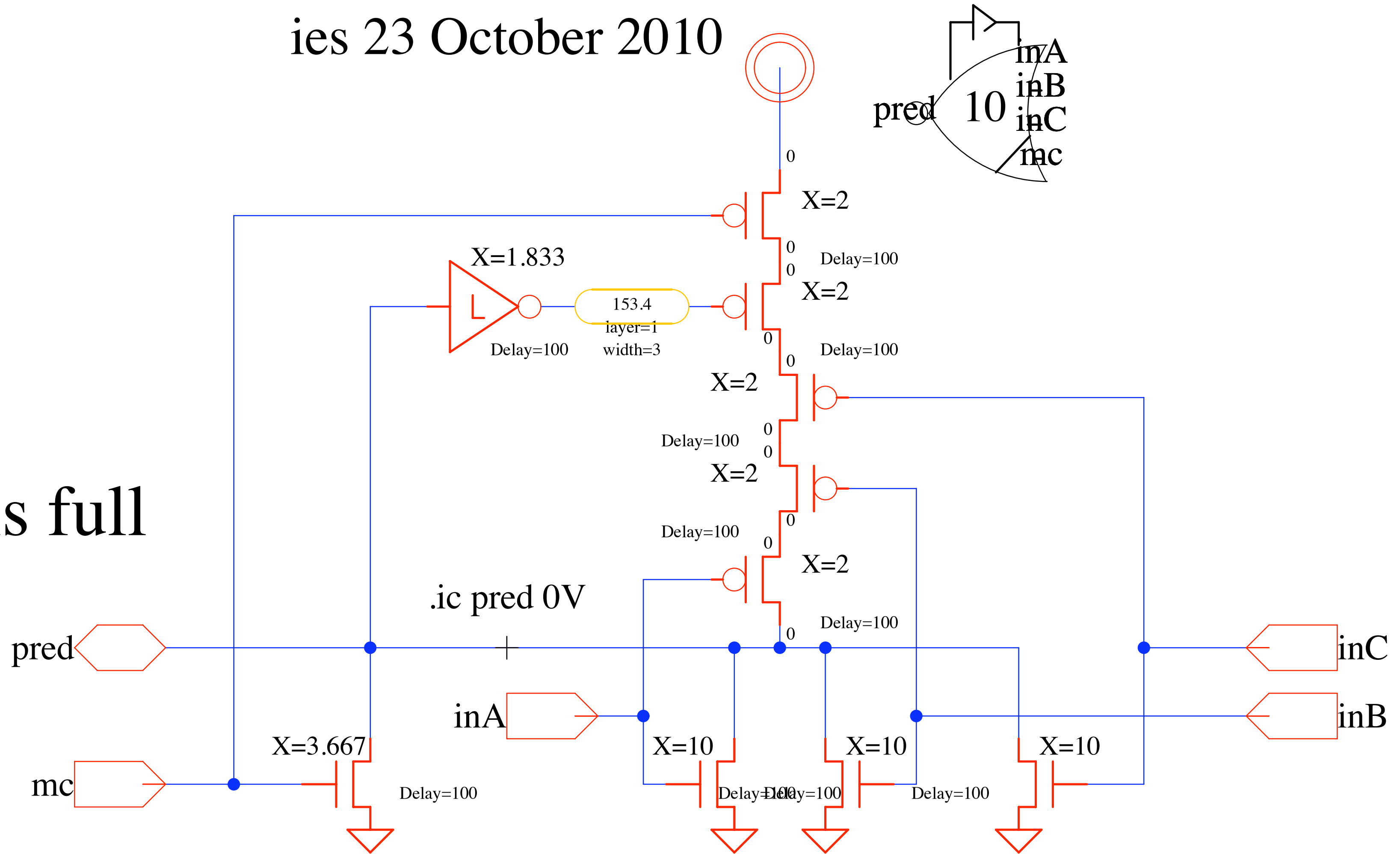
Delay=100

mc

pred3OrDri10wMC

ies 23 October 2010

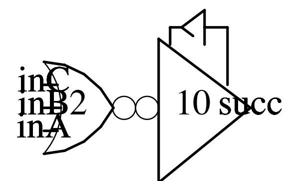
HI is full

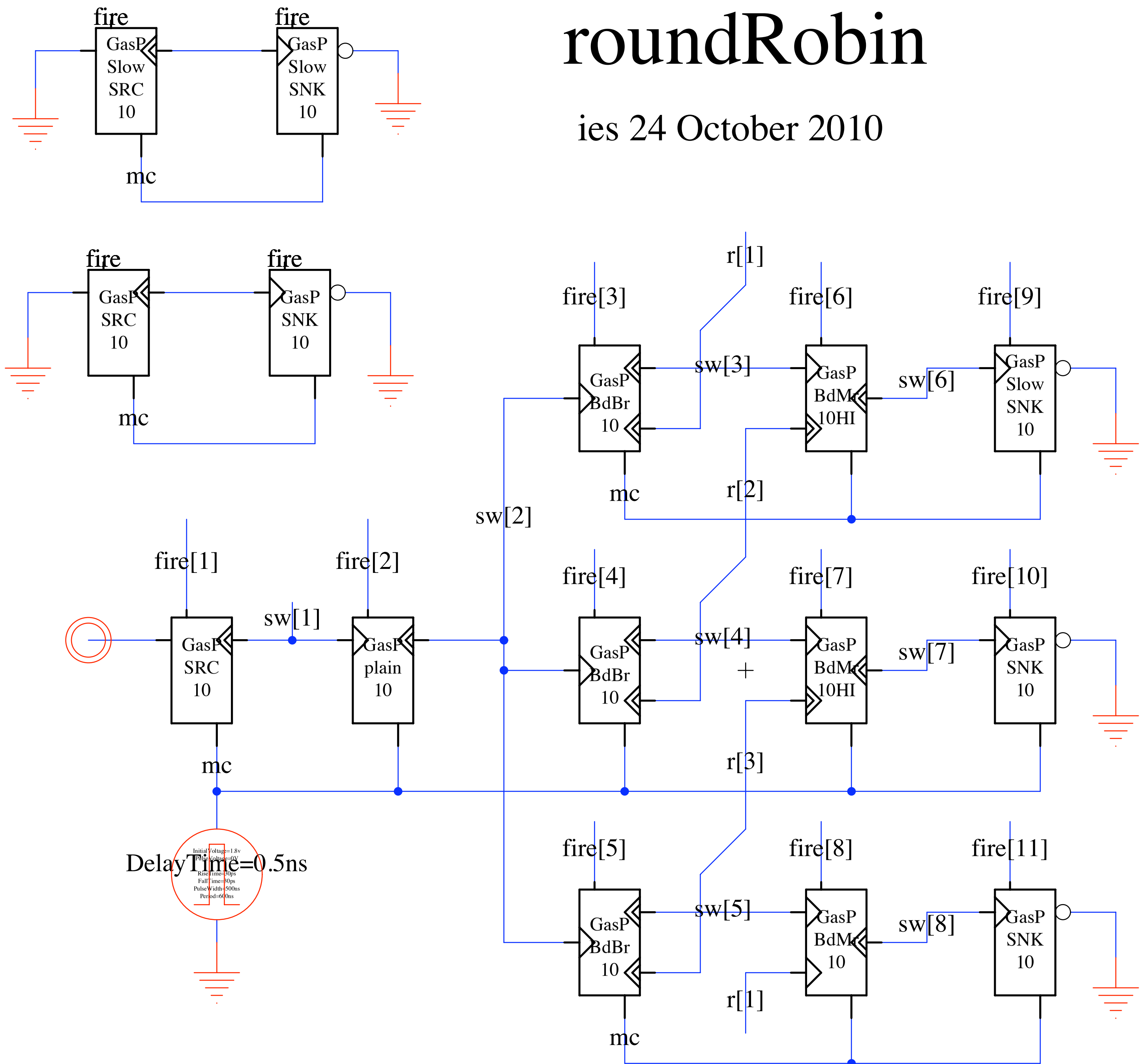


suc3OrDri10

HI is full successor driver

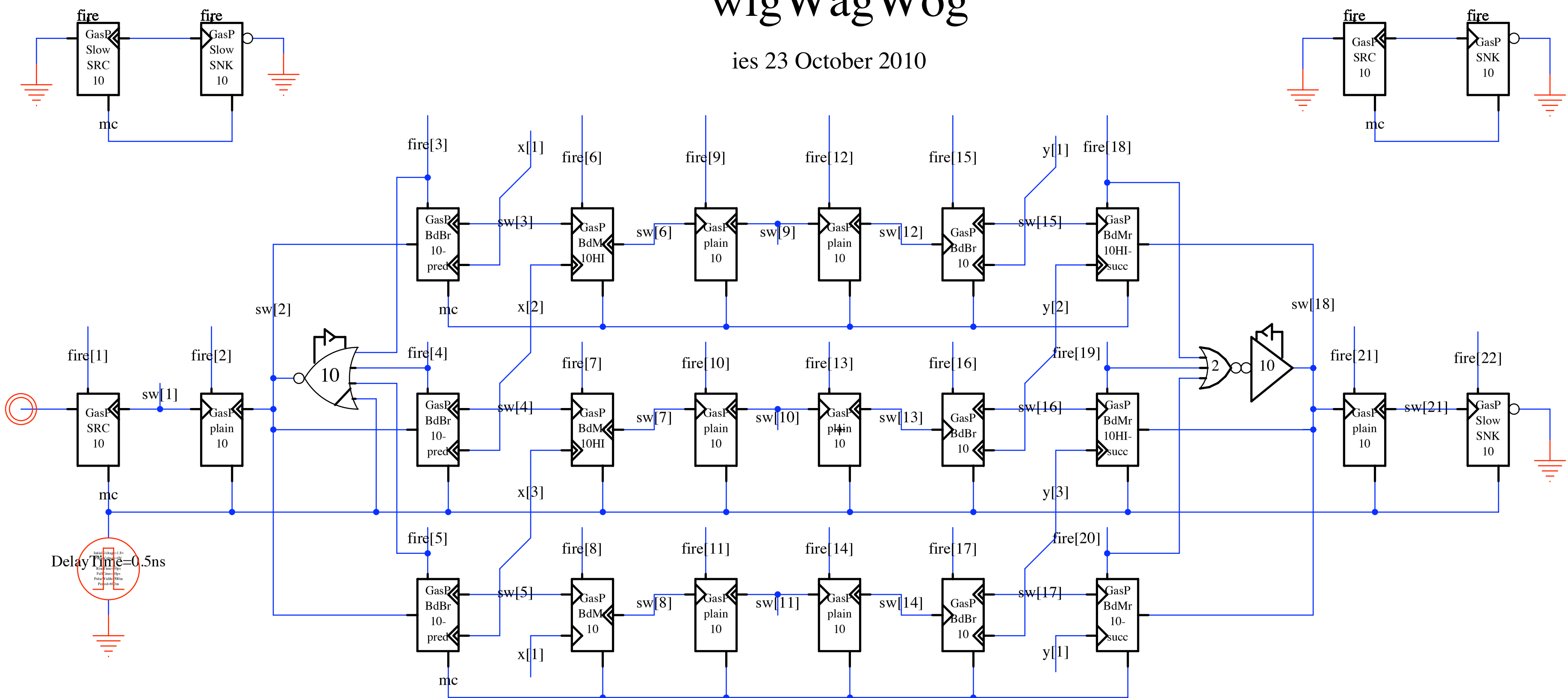
ies 23 October 2010




roundRobin
ies 24 October 2010
fire
GasP
Slow
SRC
10
fire
GasP
Slow
SNK
10
mc
fire
GasP
SRC
10
fire
GasP
SNK
10
mc
fire[1]
GasP
SRC
10
sw[1]
fire[2]
GasP
plain
10
mc
DelayTime=0.5ns
sw[2]
fire[3]
GasP
BdBr
10
r[1]
sw[3]
fire[6]
GasP
BdMe
10HI
sw[6]
fire[9]
GasP
Slow
SNK
10
mc
r[2]
fire[4]
GasP
BdBr
10
sw[4]
fire[7]
GasP
BdMe
10HI
sw[7]
fire[10]
GasP
SNK
10
r[3]
fire[5]
GasP
BdBr
10
sw[5]
fire[8]
GasP
BdMe
10
sw[8]
fire[11]
GasP
SNK
10
r[1]
mc

wigWagWog

ies 23 October 2010



ANSWER SHEET – due **9 November 2010** **6**

Name ______________________________Turned in on Date ________________________

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

Answer from simulation of `roundRobin`:

Immediately after master clear r[1] = <HI> <LO> , r[2] = <HI> <LO> , r[3] = <HI> <LO> .
Why? Hint: examine the circuits carefully.

My simulation shows pulses at `fire[2]` every _______________ psec for a throughput of _______________GDI/s.

My simulation shows pulses at `fire[3]` every _______________ psec for a throughput of _______________GDI/s.

The pulses on `fire[3:5]` interdigitate (smoothly) (irregularly).

I (can) (cannot) see any hesitation from the slow sink at fire[9]. Explain why.

Answer from simulation of `wigWagWog (with slow sink):`

My simulation shows pulses at `fire[2]` every _______________ psec for a throughput of _______________GDI/s.

My simulation shows pulses at `fire[21]` every _______________ psec for a throughput of _______________GDI/s.

State wire `sw[16]` changes from mostly empty to mostly full after ________ of its cycles.

State wire `sw[13]` changes from mostly empty to mostly full after ________ of its cycles.

State wire `sw[10]` changes from mostly empty to mostly full after ________ of its cycles.

I detect a change in the pattern on `fire[2]` after _______________ nsec. Why?

Answer from simulation of `wigWagWog (with ordinary sink):`

State wire `sw[13]` is now mostly (full) (empty).

State wire `sw[16]` is now mostly (full) (empty).

The slowest stage is now stage __________ or __________ or __________.

The holdup is state wire __________.