M A S S A C H U S E T T S I N S T I T U T E O F T E C H N O L O G Y
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.884 Complex Digital Systems
Spring 2005 - Quiz - March 18, 2005
80 Minutes

NAME: _____ SCORE: _____

Please write your name on every page of the quiz.

Not all questions are of equal difficulty, so look over the entire quiz and budget your time carefully.

Please carefully state any assumptions you make.

Enter your answers in the spaces provided below. If you need extra room for an answer or for scratch work, you may use the back of each page but please *clearly indicate where your answer is located.*

A list of useful equations is printed at the end of this quiz. You can detach this sheet for reference and do not have to hand this in. *We will not grade anything written on the equation sheet.*

You will also receive a separate handout containing a copy of the relevant Bluespec lecture slides. *We will not grade anything written on the Bluespec slides.*

You must not discuss the quiz's contents with other students who have not yet taken the quiz. If, prior to taking it, you are inadvertently exposed to material in a quiz — by whatever means — you must immediately inform the instructor or a TA.

	Points	**Score**
Problem 1	22	
Problem 2	23	
Problem 3	15	
Problem 4	20	
Problem 5	20	

Problem 1 : Optimizing delay of a sign-extension circuit (22 total points)

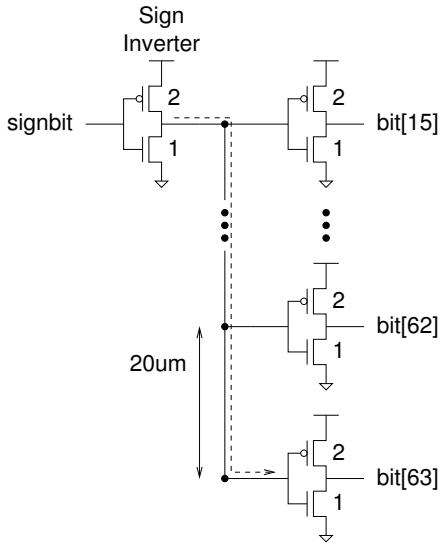
Sign-extension is a common operation in arithmetic circuits, where a narrower binary integer is converted into a wider binary integer by replicating the sign bit in the higher order bits of the destination. In this question, we examine the delay penalty for extending a 16-bit number to a 64-bit value. For this problem, assume that the sign bit is generated by a minimum-sized inverter, and that the sign-extension circuit must eventually drive 49 other minimum-sized inverters. All bits in the datapath are arranged linearly 20 μm apart. The following table lists various parameters which you may find useful when solving this problem. Remember that there is a list of useful equations at the end of this quiz.

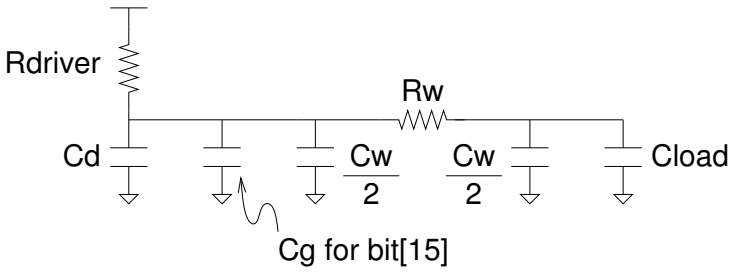
Parameters for Minimum-Sized Inverter	**Symbol**	**Value**
Ratio of PMOS to NMOS transistor widths for equal rise/fall times	ρ	2
Gate capacitance for PMOS pull-up transistor	$C_{p,g}$	2 fF
Gate capacitance for NMOS pull-down transistor	$C_{n,g}$	1 fF
Total parasitic drain capacitance	C_d	3 fF
Effective on resistance for PMOS pull-up transistor	$R_{p,on}$	2 kΩ
Effective on resistance for NMOS pull-down transistor	$R_{n,on}$	2 kΩ

Parameters for Metal 1 Wire	**Symbol**	**Value**
Wire resistance per unit length	R_{m1}	1 Ω/μm
Wire capacitance per unit length	C_{m1}	0.2 fF/μm

Part 1.A : Unoptimized delay of sign-extension circuit (9 points)

To begin, we naively use Metal 1 to wire the sign inverter directly to the 49 output inverters as shown in the diagram. Use a simple RC delay model to estimate the delay from the output of the sign inverter to the input of the inverter in the most-significant bit (the corresponding path is indicated with a dashed line). The numbers beside each transistor denote the width of that transistor normalized to the width of the NMOS in a minimum sized inverter. Report the delay as an RC time constant in picoseconds.

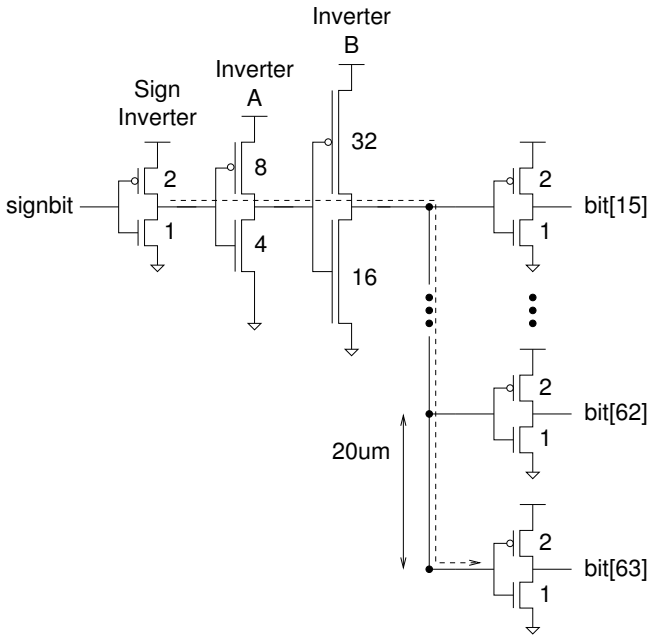




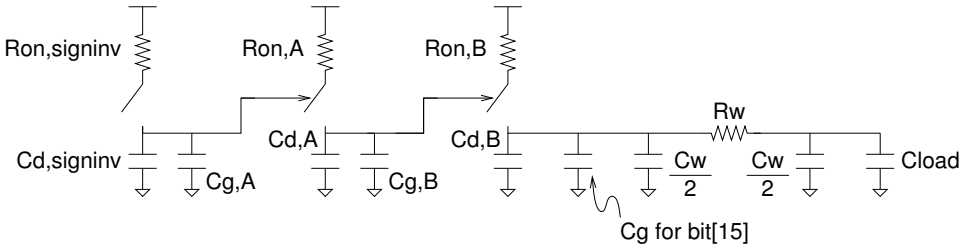
$$
\begin{aligned}
R_{wire} &= L_{wire} \times R_{m1} = (48 \times 20\mu\text{m}) \times 1\Omega/\mu\text{m} = 960\Omega \\
C_{wire} &= L_{wire} \times C_{m1} + 47 \times (C_{p,g} + C_{n,g}) = (48 \times 20\mu\text{m}) \times 0.2\text{fF}/\mu\text{m} + 47 \times 3\text{fF} = 333\text{fF} \\
C_{load} &= (C_{p,g} + C_{n,g}) = 3\text{fF} \\
\text{Delay} &= \left(R_{driver} \times \left(\frac{C_{wire}}{2} + C_d + C_{g,bit15}\right)\right) + (R_{driver} + R_{wire}) \times \left(\frac{C_{wire}}{2} + C_{load}\right) \\
&= 2\text{k}\Omega \times 172.5\text{fF} + (2\text{k}\Omega + 960\Omega) \times (166.5\text{fF} + 3\text{fF}) \\
&= 847\text{ps}
\end{aligned}
$$

Part 1.B : Reducing delay using a multi-stage driver (10 points)

To improve performance you decide to use the multi-stage driver shown in the diagram. Notice that this driver design adheres to the rule-of-thumb mentioned in class - each driver stage is scaled up by a factor of four. Again, use a simple RC delay model to estimate the delay from the output of the sign inverter, through inverter A, through inverter B, and to the input of the inverter in the most-significant bit (the corresponding path is indicated with a dashed line). Report the delay as an RC time constant in picoseconds.



We first determine the RC time constant of the sign inverter and inverter A. We then determine the RC time constant of the output of inverter B to the input of the inverter in the most-significant bit. Finally, we add all three time constants together. We are assuming that after the time constant associated with the sign inverter, inverter A turns on and then after the time constant associated with inverter A, inverter B turns on, and so on. Note that R_{wire}, C_{wire}, C_{load} remain the same as in Part 1.A.



$$
\begin{aligned}
T_{signinv} &= R_{on,signinv} \times (C_{d,signinv} + C_{g,A}) = 2\text{k}\Omega \times (3\text{fF} + 12\text{fF}) = 30\text{ps} \\
T_A &= R_{on,A} \times (C_{d,A} + C_{g,B}) = 0.5\text{k}\Omega \times (12\text{fF} + 48\text{fF}) = 30\text{ps} \\
T_{B+wire} &= \left(R_{on,B} \times \left(\frac{C_{wire}}{2} + C_{d,B} + C_{g,bit15}\right)\right) + (R_{on,B} + R_{wire}) \times \left(\frac{C_{wire}}{2} + C_{load}\right) \\
&= 125\Omega \times 172.5\text{fF} + (125\Omega + 960\Omega) \times 169.5\text{fF} = 205\text{ps} \\
\text{Delay} &= T_{signinv} + T_A + T_{B+wire} = 265\text{ps}
\end{aligned}
$$

Part 1.C : Further improvements to the multi-stage driver (3 points)

Qualitatively describe another approach which might further decrease the delay of our sign-extension circuit. Limit your answer to less than three sentences.

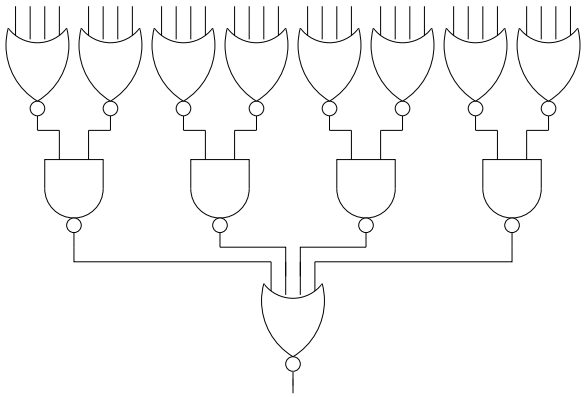
The wire delay is dominating the total delay of this path, so we should consider a distributed driver where the stages are spread along the wire. We could use logical effort or standard repeater techniques to determine where to place the stages and how to size them.

Problem 2 : Optimizing delay of branch comparator (23 total points)

The branch comparator in the SMIPS processor requires a comparator that can check whether 32 bits are all equal to zero. The output of the comparator is one if all inputs are zero. In this problem we will use the logical effort methodology to compare the delay of various branch comparator implementations. Remember that there is a list of useful equations at the end of this quiz.

Part 2.A : Optimal delay of an initial implementation (9 points)

The following circuit uses a tree of NAND and NOR gates to implement the branch comparator. Verify to yourself that the output is one if and only if all 32 inputs are zero. Use the method of logical effort to estimate the optimal delay (in picoseconds) for this circuit. Assume that the input capacitance of a 4-input NOR gate is 3 fF and that the branch comparator must drive a load capacitance of 3 fF. Also assume that the delay unit (τ) for this process is 20 ps and that the parasitic delay of a minimum-sized inverter is 1.



$$
\begin{aligned}
F &= GBH = (9/3)(4/3)(9/3) \times 1 \times (3/3) = 12 \\
P &= 4p_{inv} + 2p_{inv} + 4p_{inv} = 10 \\
\hat{D} &= NF^{1/N} + P = 3(12)^{1/3} + 10 = 16.9 \\
\hat{D}_{abs} &= \hat{D}\tau = 337\text{ps}
\end{aligned}
$$

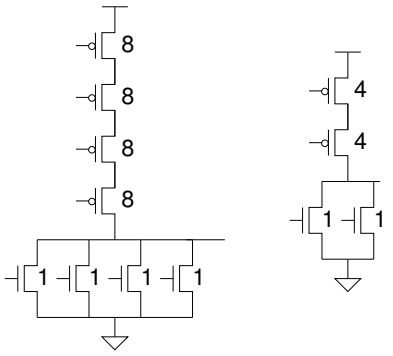
Part 2.B : Optimal delay of various NOR/NAND trees (10 points)

We are given a gate library which contains the following four gates: 4-input NAND, 4-input NOR, 2-input NAND, and 2-input NOR. We will now use logical effort to evaluate all of the ways we can construct the branch comparator from this library. First, fill in the following table with all of the possible NOR/NAND trees which implement the correct logic function. Remember that the output of the tree must be one if all of the inputs are zero. You cannot use inverters. To denote a given NOR/NAND tree simply list the type of gate used in each stage of the tree. For example, the tree corresponding to Part 2.A is { `nor4, nand2, nor4` }, and it is already filled in on the table. Use the logical effort methodology to fill in the path logical effort (G), the total path effort (F), the path parasitic delay (P), and the optimal path delay ($\hat{D}_{abs}$) in picoseconds for each NOR/NAND tree. You should be able to fill in the first row of the table based on your answer from Part 2.A. *Hint: The number of possible NOR/NAND trees is equal to the number of rows in the table.* Which is the fastest implementation?

We did not tell you the input capacitance of a 2-input NOR gate. So if you assumed that both 2-input and 4-input NOR gates had the same input capacitance you would get the following result.

NOR/NAND Tree	G	C_{in}	H	F	$F^{1/N}$	P	N	$\hat{D}$	$\hat{D}_{abs}$
{ *nor4,nand2,nor4* }	*12*	*3*	*1*	*12*	*2.29*	*10*	*3*	*16.9*	*337ps*
{ *nor4,nand4,nor2* }	*10*	*3*	*1*	*10*	*2.15*	*10*	*3*	*16.5*	*329ps*
{ *nor2,nand4,nor4* }	*10*	*3*	*1*	*10*	*2.15*	*10*	*3*	*16.5*	*329ps*
{ *nor2,nand2,nor2,nand2,nor2* }	*8.2*	*3*	*1*	*8.2*	*1.52*	*10*	*5*	*17.6*	*352ps*

A 2-input NOR gate, however, does not *have the same input capacitance as a 4-input NOR gate. The mosfet diagrams to the right show the two NOR gates sized assuming: (a) that PMOS transistors are twice as slow as NMOS, and (b) that we want equal rise/fall times. The 4-input NOR gate has an input capacitance of 3fF so the capacitance per unit of transistor width is 3/9fF. Thus a reasonable assumption is that the 2-input NOR gate has an input capacitance of* $3/9 \times 5$ *or 1.67fF.*



NOR/NAND Tree	G	C_{in}	H	F	$F^{1/N}$	P	N	$\hat{D}$	$\hat{D}_{abs}$
{ *nor4,nand2,nor4* }	*12*	*3*	*1.0*	*12*	*2.29*	*10*	*3*	*16.9*	*337ps*
{ *nor4,nand4,nor2* }	*10*	*3*	*1.0*	*10*	*2.15*	*10*	*3*	*16.5*	*329ps*
{ *nor2,nand4,nor4* }	*10*	*1.7*	*1.8*	*18*	*2.62*	*10*	*3*	*17.8*	*357ps*
{ *nor2,nand2,nor2,nand2,nor2* }	*8.2*	*1.7*	*1.8*	*14.8*	*1.71*	*10*	*5*	*18.5*	*371ps*

Part 2.C : Optimal delay for alternative capacitive load (4 points)

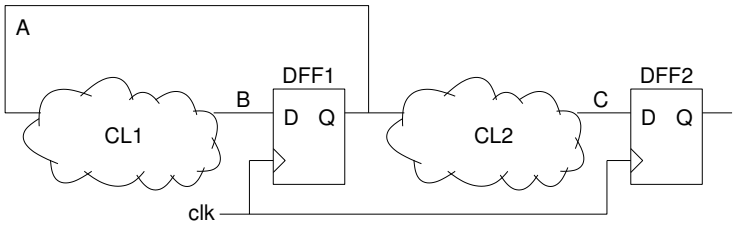
Now assume that the branch comparator must drive an output load which is a thousand times larger (3,000 fF). Which NOR/NAND tree is the fastest implementation? You may answer this question numerically or with a brief qualitative argument.

Using the implementation with the most stages will help mitigate the increased output load. More stages increase N and thus the effort portion of the optimal delay decreases. Adding too many stages, though, will increase the parasitic term of the optimal delay and result in too little effort per stage. For this problem the five stage tree should perform best with the larger output load. We show this numerically below.

NOR/NAND Tree	G	C_{in}	H	F	$F^{1/N}$	P	N	$\hat{D}$	$\hat{D}_{abs}$
{ *nor4,nand2,nor4* }	*12*	*3*	*1k*	*12k*	*22.9*	*10*	*3*	*78.6*	*1.6ns*
{ *nor4,nand4,nor2* }	*10*	*3*	*1k*	*10k*	*21.5*	*10*	*3*	*74.6*	*1.5ns*
{ *nor2,nand4,nor4* }	*10*	*1.7*	*1.8k*	*18k*	*26.2*	*10*	*3*	*88.6*	*1.8ns*
{ *nor2,nand2,nor2,nand2,nor2* }	*8.2*	*1.7*	*1.8k*	*14.8k*	*6.83*	*10*	*5*	*44.1*	*882ps*

Problem 3 : Calculating minimum clock period (15 total points)

The following diagram shows a finite state machine built from combinational logic (CL) and D-flip-flops (DFFs). The table lists the various timing parameters. The initial clock period is 9.



Parameters for DFFs	**Symbol**	**Value**
Clock to Q min delay	T_{CQMIN}	2
Clock to Q max delay	T_{CQMAX}	3
Setup time	T_{setup}	1
Hold time	T_{hold}	5

Parameters for CLs	**Symbol**	**Value**
CL1 min propagation delay	$T_{CL1,PDMIN}$	2
CL1 max propagation delay	$T_{CL1,PDMAX}$	3
CL2 min propagation delay	$T_{CL2,PDMIN}$	4
CL2 max propagation delay	$T_{CL2,PDMAX}$	5

Part 3.A : Identifying timing violation (5 points)

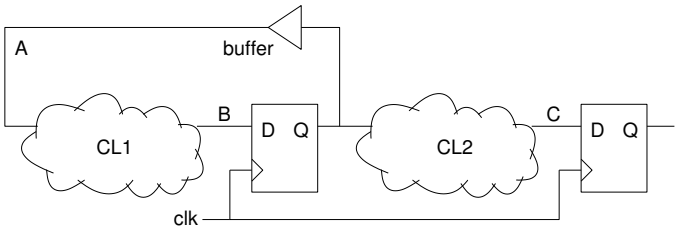
There is a timing violation in this circuit. What is the violation and on what path does it occur?

Hold time violation on `DFF1.Q->A->CL1->DFF1.D` *since* $T_{CQMIN} + T_{CL1,PDMIN} < T_{hold}$

Part 3.B : Fixing timing violation (5 points)

Assume you have a non-inverting buffer for which $T_{BUF,PDMIN}$ is 2 and $T_{BUF,PDMAX}$ is 3. Draw a new circuit diagram showing how these buffers can be added to the circuit to resolve the timing violation.

Add one buffer at node A so that $T_{CQMIN} + T_{BUF,PDMIN} + T_{CL1,PDMIN} > T_{hold}$



Part 3.C : Final clock period (5 points)

What is the final clock period? How did fixing the timing violation affect the clock period?

$T_{period} = T_{CQMAX} + T_{BUF,PDMAX} + T_{CL1,PDMAX} + T_{setup} = 10$

Problem 4 : Rule firing in Bluespec (20 total points)

In this problem we will explore the behavior of the two stage pipeline presented in slides L13-3 to L13-6 (these slides are included at the end of the quiz). You should assume that `bu` has a maximum capacity of two instruction templates. You should also assume the following starting state:

- `bu` holds `Tuple2(99, EBz {cond: 0, addr:200})`
- `pc` is 100
- Instruction at address 100 is `Add {dst: R3, src1: R1, src2: R2}`
- Instruction at address 101 is `Add {dst: R6, src1: R4, src2: R5}`
- Instruction at address 200 is `Add {dst: R9, src1: R4, src2: R7}`

Part 4.A : (4 points)

Describe the contents of `pc` and `bu` after applying the `Fetch&Decode` rule.

```
pc = 101

Instruction templates in bu:
  Tuple2(100, EAdd {dst: R3, op1:rf[R1], op2:rf[R2]}) and
  Tuple2( 99, EBz {cond: 0, addr: 200})
```

Part 4.B : (4 points)

Describe the contents of `pc` and `bu` after applying the `Fetch&Decode` rule followed by the `Execute` rule.

```
pc = 200

Instruction templates in bu:
  Empty
```

Part 4.C : (4 points)

Describe the contents of `pc` and `bu` after applying the `Execute` rule followed by the `Fetch&Decode` rule.

```
pc = 201

Instruction templates in bu:
  Tuple2(200, EAdd {dst: R9, op1:rf[R4], op2:rf[R7]})
```

Part 4.D : (8 points)

We can write a single rule that achieves the effect described in Part 4.C. Fill in the following rule so that it has the same effect as applying the `Execute` rule followed by the `Fetch&Decode` rule.

Notice that we have used a new method for the `bu` FIFO which clears the fifo and then enques the given value in the same cycle. An acceptable solution might also call `clear()` and `enq()` directly but in this case, the solution must make some kind of note indicating that there might be an issue with using both of these methods in the same rule on the same queue. There are some subtle issues with this part which we are currently working out - we will post an updated set of solutions in the next few days.

```
rule compoundBzFetchAdd ( instr matches Add {dst:.rd, src1:.ra,src2:.rb}
                          &&& it matches EBz {cond:.cv,addr:.av} );

  if ( cv == 0 ) then begin

    bu.clearThenEnq(tuple2(av, EAdd {dst:rd, op1:rf[ra], op2:rf[rb]}));
    pc <= av+1;

  end
  else if ( !stall ) then begin

    bu.deq();
    bu.enq(tuple2(pc, EAdd {dst:rd, op1:rf[ra], op2:rf[rb]}));
    pc <= pc+1;

  end
  else begin

    bu.deq();

  end
endrule
```

Problem 5 : Bluespec synthesis (20 total points)

In this problem we will explore the circuit that is generated for the example taken from the lecture slide L08-20. You may find slide L10-26 helpful. These slides are included at the end of the quiz.

```
(* descending_urgency = "r1, r2" *)

// Moving packets from input FIFO i1
rule r1;
  Tin x = i1.first();
  if ( dest(x) == 1 ) o1.enq(x);
  else                o2.enq(x);
  i1.deq();
  if (interesting(x)) c <= c + 1;
endrule

// Moving packets from input FIFO i2
rule r2;
  Tin x = i2.first();
  if ( dest(x) == 1 ) o1.enq(x);
  else                o2.enq(x);
  i2.deq();
  if (interesting(x)) c <= c + 1;
endrule
```

Naming convention: The Data, Ready and Enable wires of the method `g` of module `m` are named `m.gData`, `m.gRdy`, and `m.gEn`, respectively. We may attach rule names to these names for further clarification if necessary. The boolean equations for the circuits that are generated for rule `r1` may be expressed as follows where `can_fire_r1` gives the conditions under which rule `r1` can fire.

Guard Logic

```
x1 = i1.firstData;
p1 = (dest(x1) == 1);
q1 = interesting(x1);
can_fire_r1 =      i1.firstRdy
              && ((p1 && o1.enqRdy) || (!p1 && o2.enqRdy));
```

Action logic (just for rule 1)

```
o1.enqEn_r1 =  p1; o1.enqData_r1 = x1;
o2.enqEn_r1 = !p1; o2.enqData_r1 = x1;
i1.deqEn_r1 =   1;
cEn_r1      =  q1; cWriteData_r1 = (cReadData+1);
```

Part 5.A : (4 points)

Write down the equation for `can_fire_r2` (i.e. the conditions under which rule `r2` can fire).

Assuming `p2 = (dest(i2.firstData) == 1)` then

```
can_fire_r2 = i2.firstRdy
              && ((p2 && o1.enqRdy) || (!p2 && o2.enqRdy))
```

Part 5.B : (8 points)

Write down the equations for the conditions under which rules `r1` and `r2` will fire. Do not forget the effect of urgency annotations.

```
will_fire_r1 = can_fire_r1
will_fire_r2 = !can_fire_r1 && can_fire_r2
```

Part 5.C : (8 points)

Write down the logic equations for the following signals obtained by combining the logic for the two rules. Let `MUX((x1, c1), (x2, c2))` represent the MUX that produces `x1` when `c1` is true and `x2` when `c2` is true, assuming `c1` and `c2` can never be true simultaneously.

```
o1.enqEn =   MUX((o1.enqEn_r1, will_fire_r1),
                 (o1.enqEn_r2, will_fire_r2))

o1.enqData = MUX((o1.enqData_r1, will_fire_r1),
                 (o1.enqData_r2, will_fire_r2))

i1.deqEn =   will_fire_r1
```

Equation Sheet

Equation or Symbol	Description
g	Gate logical effort
$h = C_{out}/C_{in}$	Gate electrical effort
$f = gh$	Gate effort
p	Gate parasitic delay
p_{inv}	Parasitic delay of minimum-sized inverter
$d = f + p$	Delay in units of τ
τ	Delay unit
$d_{abs} = d\tau$	Absolute delay in seconds
$G = \prod g_i$	Path logical effort
$H = C_{out}/C_{in}$	Path electrical effort
$F = GH$	Path effort
$D = \sum d_i = \sum g_i h_i + \sum p_i$	Path delay
$\hat{f} = g_i h_i = F^{1/N}$	Optimal stage effort
$\hat{D} = NF^{1/N} + P$	Optimal path delay
$\hat{h}_i = 1/g_i \times F^{1/N}$	Optimal stage electrical effort
Delay $= \sum_{i=0}^{n} \left(\sum_{j=0}^{j=i} R_j \right) C_i$	Penfield-Rubenstein wire-delay model
R_d	Resistance of driver
R_w	Total resistance of wire
C_w	Total capacitance of wire
Delay $= R_d \times C_w/2 + (R_d + R_w) \times (C_w/2 + C_{load})$	Simple lumped π model

	Number of inputs					
Gate Type	**1**	**2**	**3**	**4**	**5**	**n**
Inverter Logical Effort	1					
NAND Logical Effort		4/3	5/3	6/3	7/3	$(n+2)/3$
NOR Logical Effort		5/3	7/3	9/3	11/3	$(2n+1)/3$
Inverter Parasitic Delay	p_{inv}					
NAND Parasitic Delay		$2p_{inv}$	$3p_{inv}$	$4p_{inv}$	$5p_{inv}$	np_{inv}
NOR Parasitic Delay		$2p_{inv}$	$3p_{inv}$	$4p_{inv}$	$5p_{inv}$	np_{inv}