Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

DETAILED DESIGN SPECIFICATION: A VERY LARGE SCALE INTEGRATION DESIGN OF AN ONLINE ALGORITHM FOR THE COMPUTATION OF ROTATION FACTORS

Stephen George Faris

April 1989
CSD-890016

TABLE OF CONTENTS

7. Simulation Results Summary

    7.1 Standard Cell Esim Results: Summary Sheet

    7.2 Standard Cell Spice Results: Summary Sheet

    7.3 Final (Chip-wide) Esim Results: Summary Sheet

    7.4 Final (Chip-wide) Spice Results: Summary Sheet

1.1 Input/Output Formats (SS)

• I/O Format

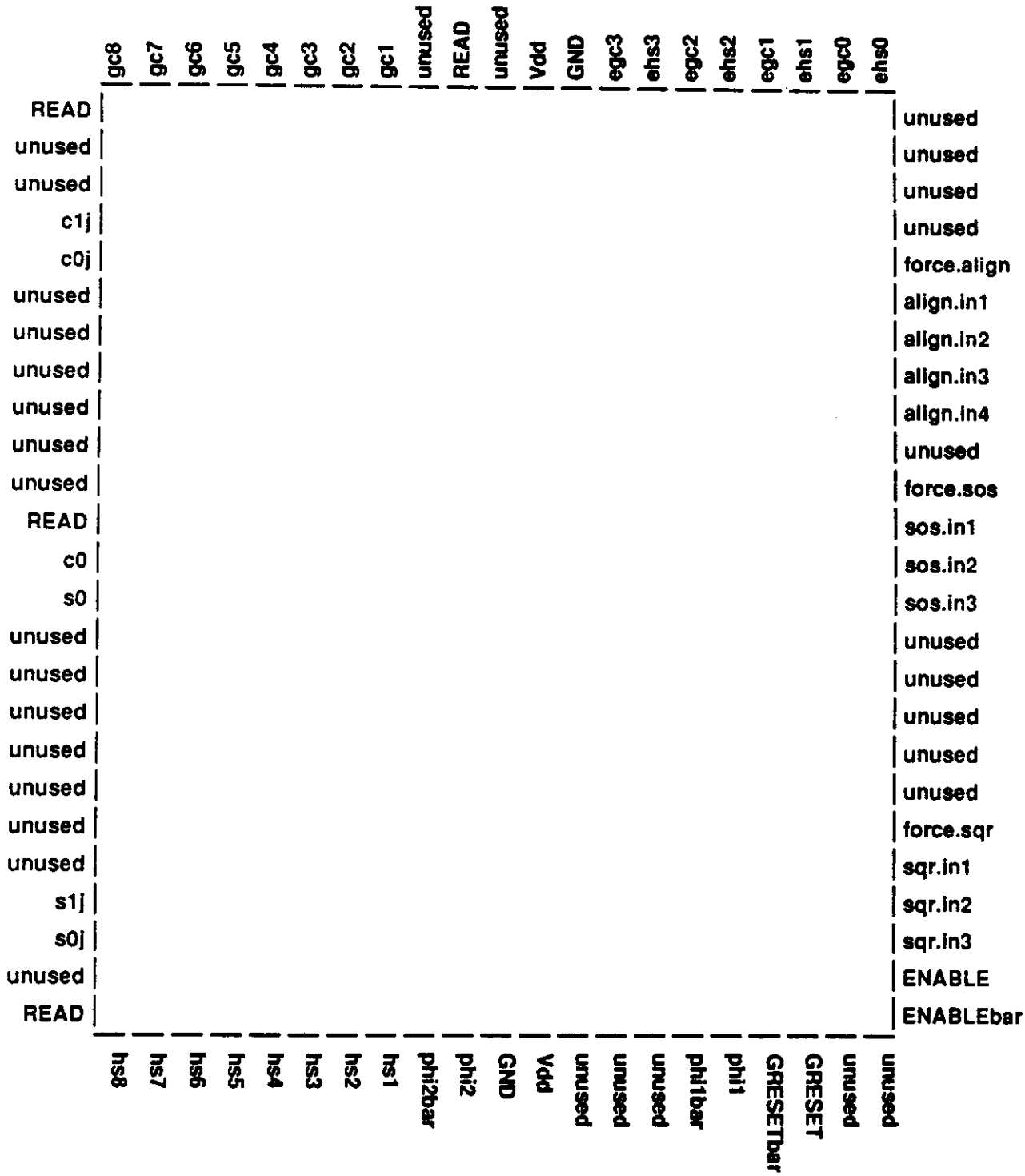
$$C = \frac{G}{(G^2 + H^2)^{(1/2)}} \qquad S = \frac{H}{(G^2 + H^2)^{(1/2)}}$$

where:	G = g * 2^eg	g = . g1 g2 g3 g4 g5 g6 g7 g8	(normalized sign-and-mag.)
		eg = eg3 eg2 eg1 eg0	(2's complement)
	H = h * 2^eh	h = . h1 h2 h3 h4 h5 h6 h7 h8	(g≠0/h≠0 guaranteed)
		eh = eh3 eh2 eh1 eh0	
	C = c * 2^ec	c = c0 . c1 c2 c3 c4 c5 c6 c7 c8	
		ec = ec3 ec2 ec1 ec0	
	S = s * 2^es	s = s0 . s1 s2 s3 s4 s5 s6 s7 s8	
		es = es3 es2 es1 es0	

• Pinouts

		Pin(s)	Direction(s)	Function	Total
INPUTS/OUTPUTS		egc3 . . . egc0	bi-directional	eg (in)/ec (out)	4
		ehs3 . . . ehs0	bi-directional	eh (in)/es (out)	4
		c0 . gc1 . . . gc8	bi-directional	g (in)/c (out) [c0 is uni-directional]	9
		s0 . hs1 . . . hs8	bi-directional	h (in)/s (out) [s0 is uni-directional]	9
		s1j, s0j, c1j, c0j	uni-directional	serial **s1(j)**, **s0(j)**, **c1(j)**, **c0(j)** outputs	4
SYSTEM		Vdd, GND	uni-directional	power rails	2
		Ø1, Ø1bar, Ø2, Ø2bar	uni-directional	two-phase clocking	4
		ENABLE, ENABLEbar	uni-directional	chip enables (**ENABLEbar** generated externally for convenience only)	2
		GRESET, GRESETbar	uni-directional	global reset (**BOTH** generated externally for convenience only)	2
		READ	uni-directional	used to sample chip outputs **ec3..ec0**, **es3..es0**, **c0..c8**, **s0..s8** at <u>any</u> time	1
TEST		force.align	uni-directional	used to override outputs of the alignment unit	1
		align.in1 .. 4	bi-directional	overrides for **x(j)**, **y(j)**, **x(j)bar.hold**, and **y(j)bar.hold**, respectively	4
		force.sos	uni-directional	used to override outputs of the sos unit	1
		sos.in1 .. 3	bi-directional	overrides for **z2(j)bar**, **z1(j)bar**, and **z0(j)bar**, respectively	3
		force.sqr	uni-directional	used to override outputs of the sqr unit	1
		sqr.in1 .. 3	bi-directional	overrides for **d1(j)bar.hold**, **d1(j).hold**, and **d0(j).hold**, respectively	3
					Total = 54

• Pad Connection Diagram



1.2 Timing Issues (SS)

· Computation Phases

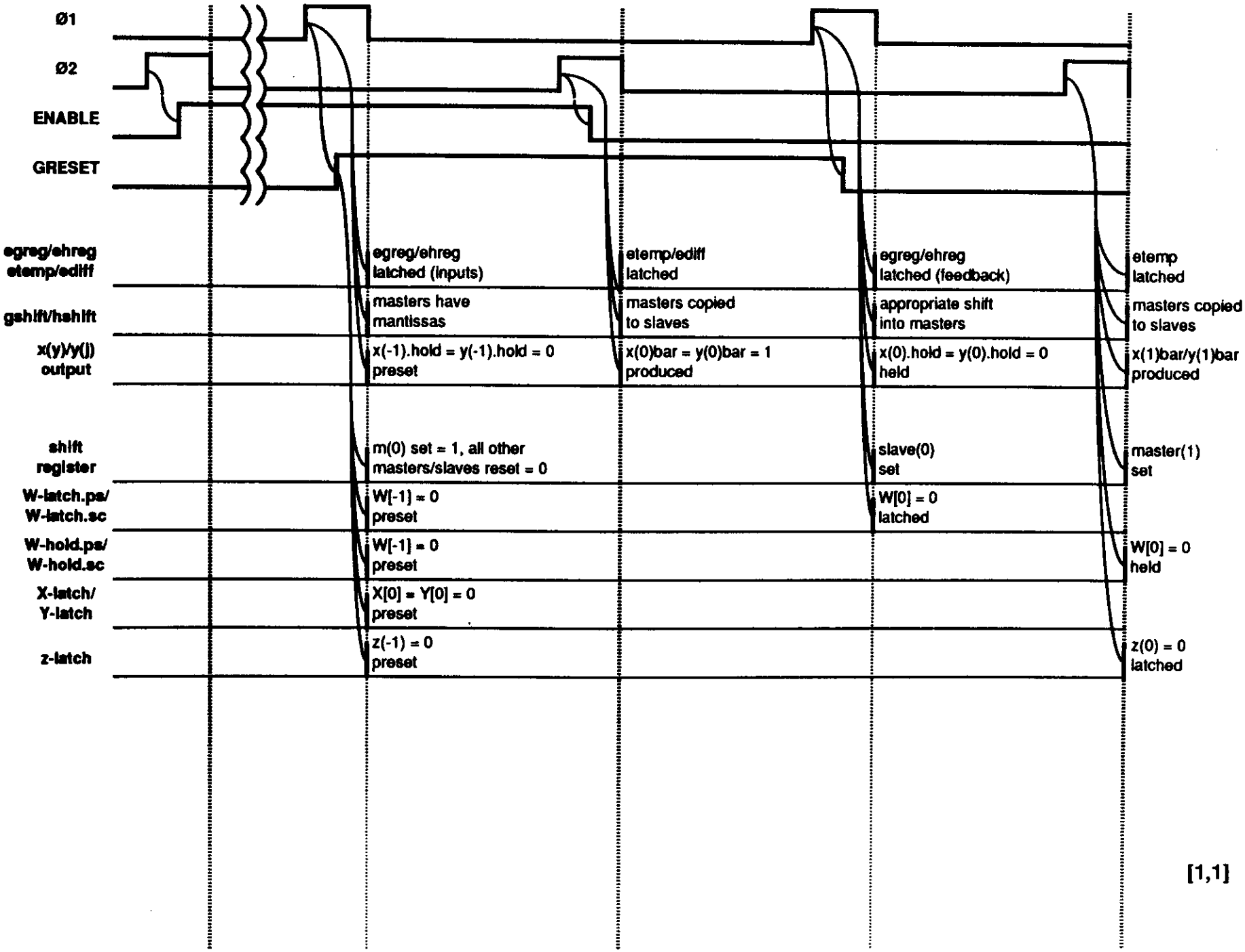
Because of the dual-phase clocking scheme used, two computation strokes are available during each beat of the "system clock", namely, the Ø2-Ø1 and Ø1-Ø2 strokes. Both phases occur from rise-to-rise of the clocks since all flip-flops in the standard library are level-triggered.

• Timing Anomaly

One small anomaly exists in the operation of the flip-flops of the standard library, namely, the Qbar outputs of preset-able flip-flops will stray and follow the input D (as Dbar) rather than assume their correct preset values **during the period in which the GRESET signal is high and the clock of that flip-flop is high**. The only time this is potentially dangerous, of course, is during the initial reset phase of the chip's operation when GRESET is active (see the accompanying Drawing Sheet which describes the anomaly and its point of occurence in detail). To avoid problems, **one should not use the Qbar output of Ø2-clocked flip-flops unless it can be assured that adequate time exists from the fall of Ø2 to the rise of Ø1 for any computation to take place** (at that one particular moment in the chip's operation).

1.3 Global Timing Diagram (DS)

GLOBAL ALIGNMENT SUM of SQUARES

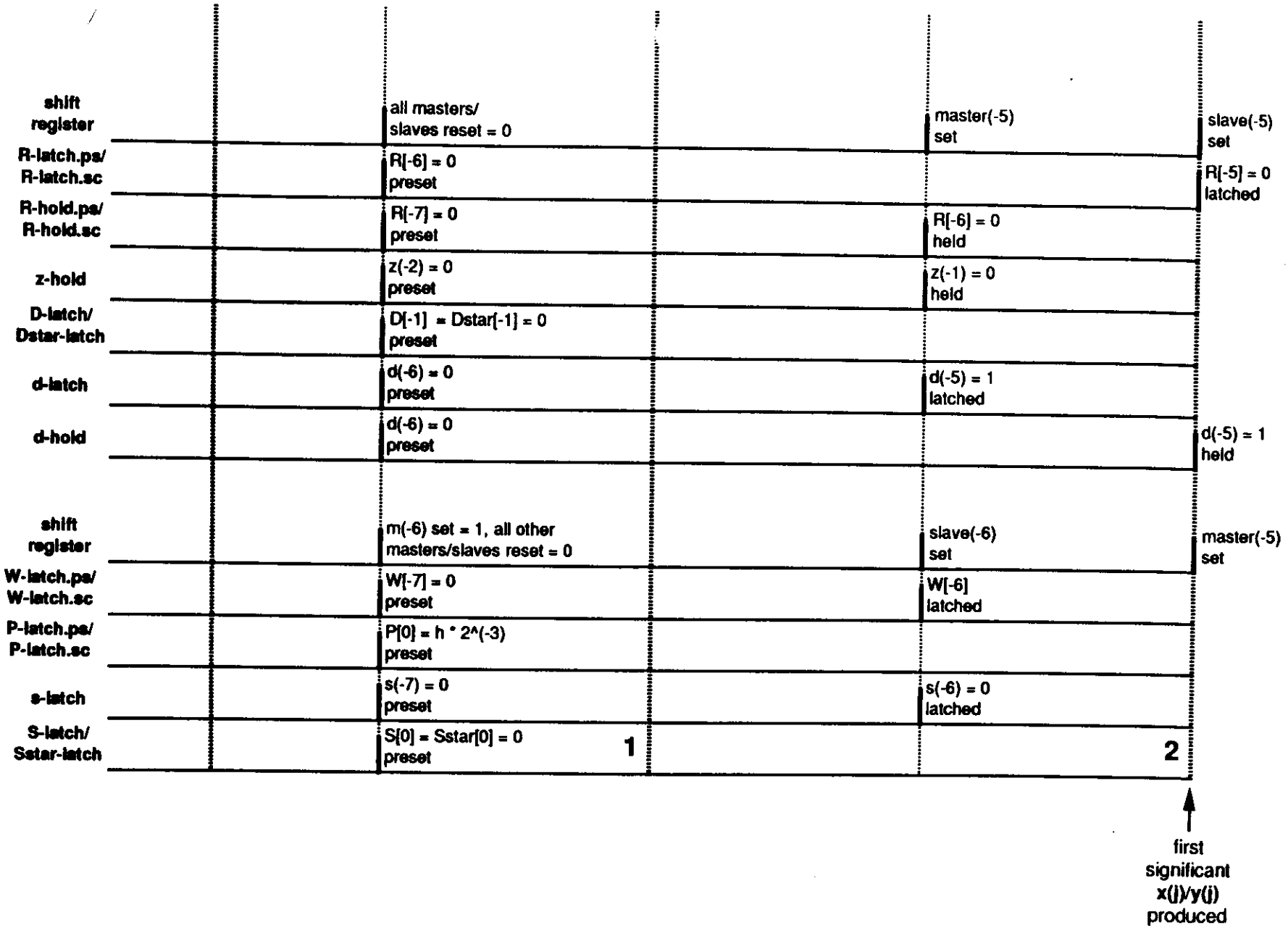


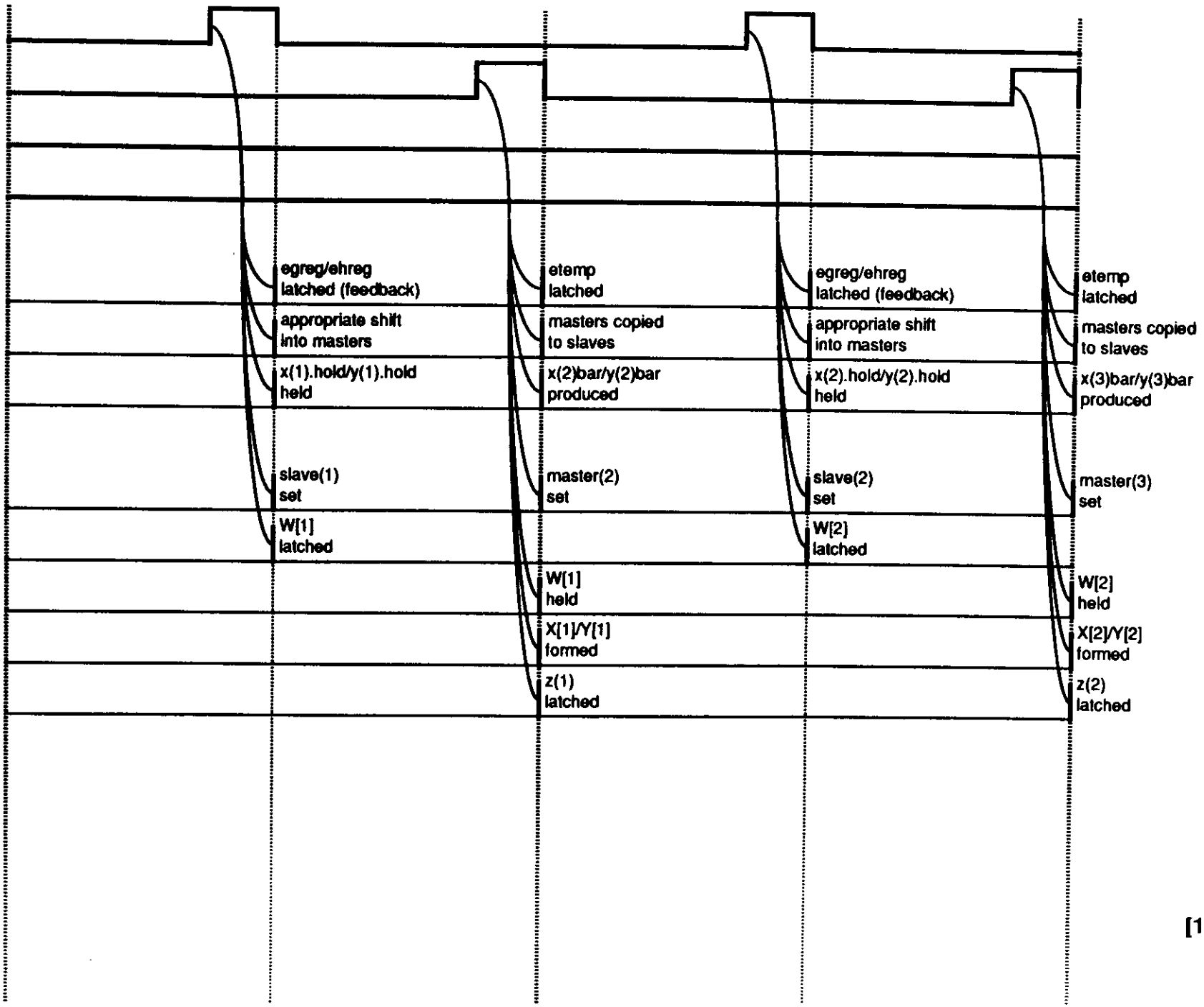
[1,1]

SQUARE ROOT

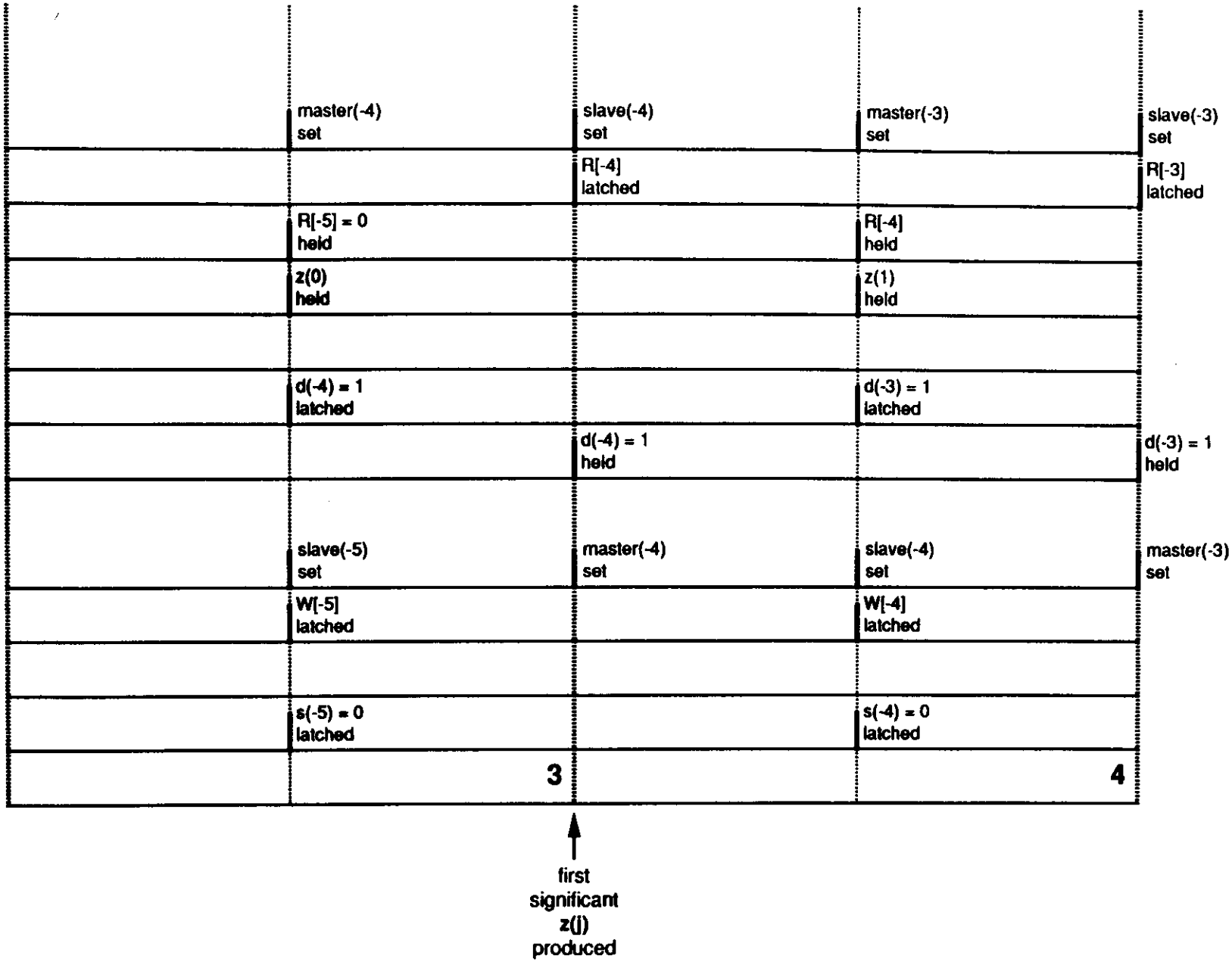
DIVISION

NOTES

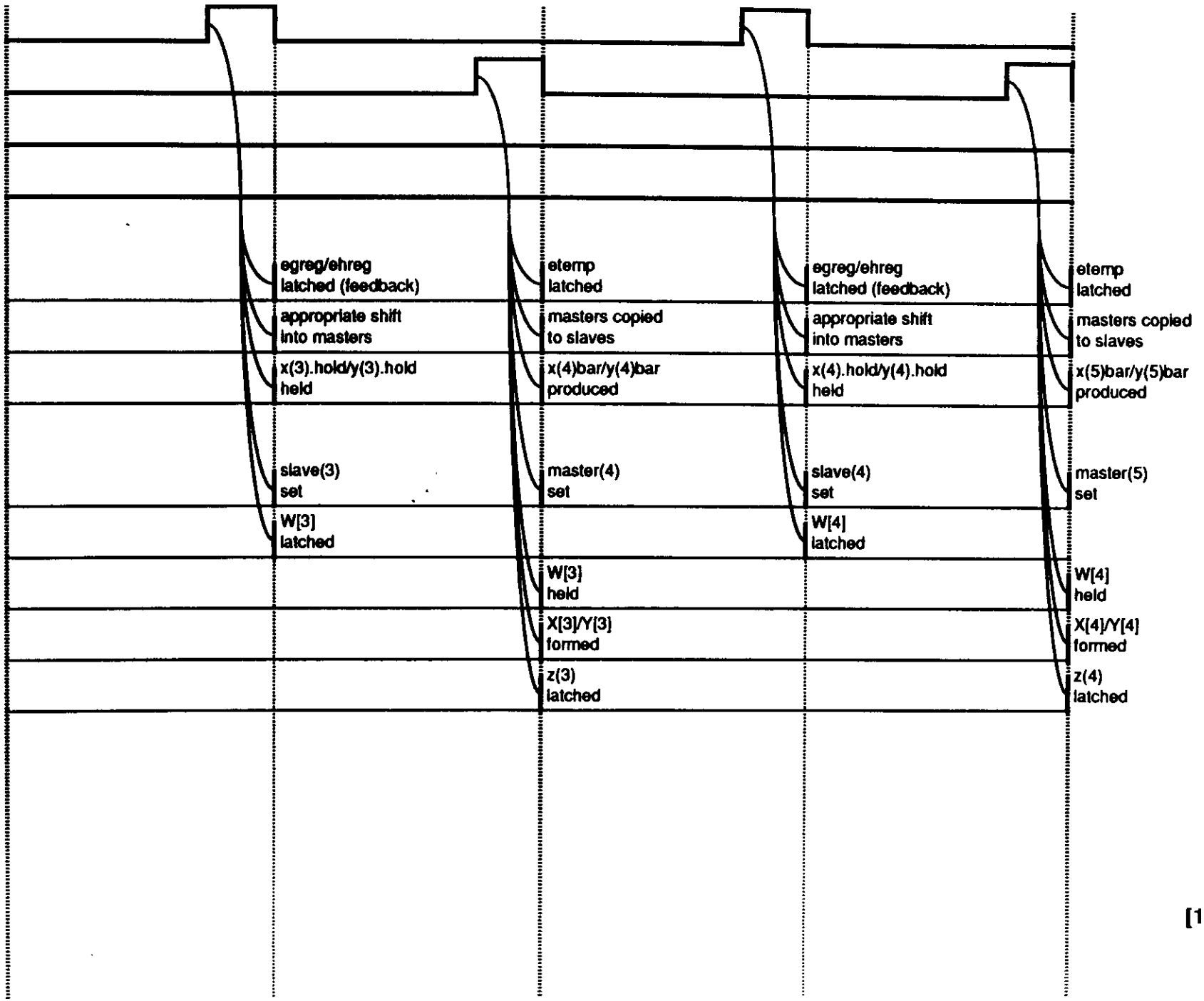



egreg/ehreg
latched (feedback)
appropriate shift
into masters
x(1).hold/y(1).hold
held
slave(1)
set
W[1]
latched
etemp
latched
masters copied
to slaves
x(2)bar/y(2)bar
produced
master(2)
set
W[1]
held
X[1]/Y[1]
formed
z(1)
latched
egreg/ehreg
latched (feedback)
appropriate shift
into masters
x(2).hold/y(2).hold
held
slave(2)
set
W[2]
latched
etemp
latched
masters copied
to slaves
x(3)bar/y(3)bar
produced
master(3)
set
W[2]
held
X[2]/Y[2]
formed
z(2)
latched

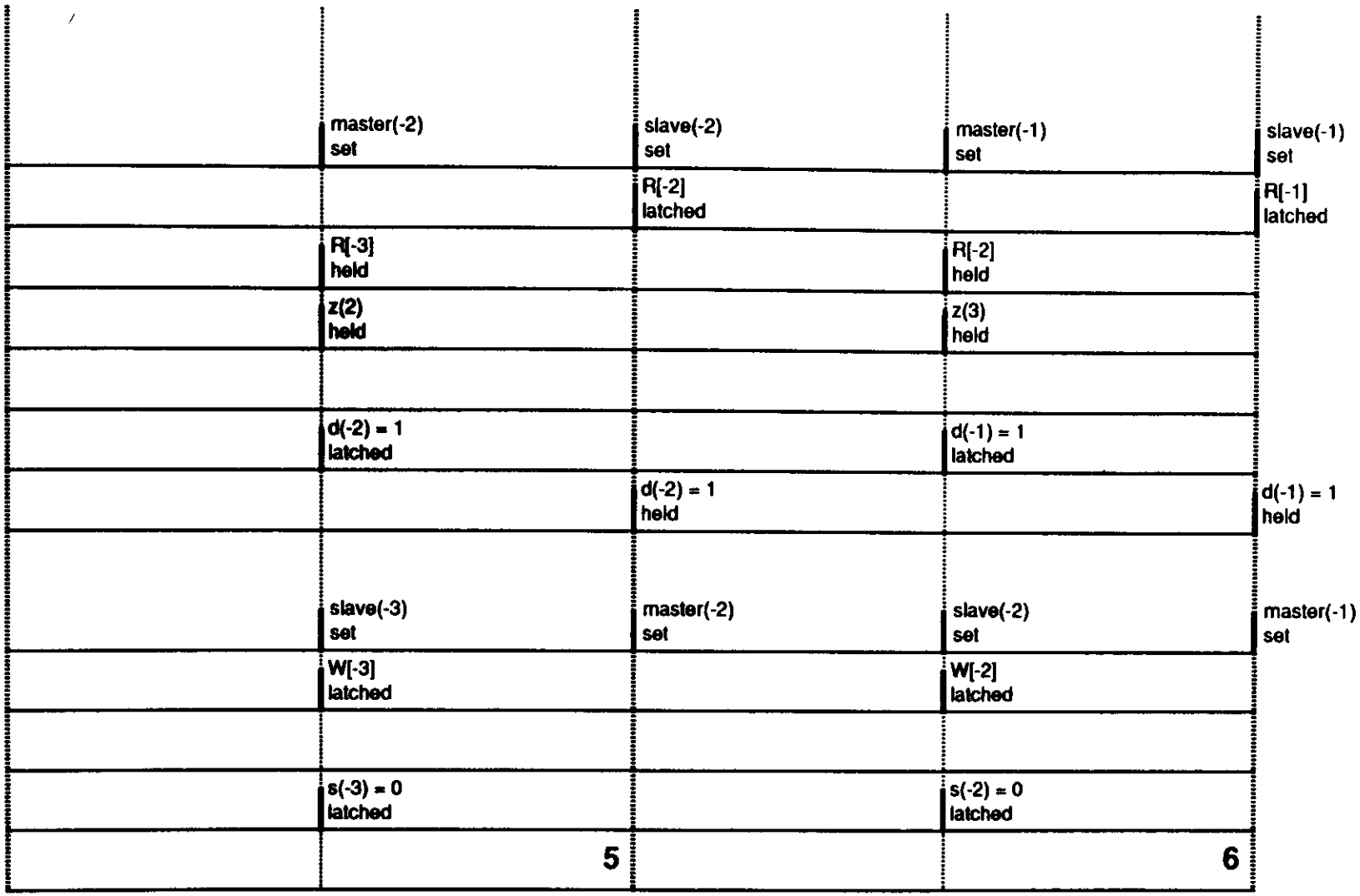
[1,2]


master(-4)
set
slave(-4)
set
master(-3)
set
slave(-3)
set
R[-4]
latched
R[-3]
latched
R[-5] = 0
held
R[-4]
held
z(0)
held
z(1)
held
d(-4) = 1
latched
d(-3) = 1
latched
d(-4) = 1
held
d(-3) = 1
held
slave(-5)
set
master(-4)
set
slave(-4)
set
master(-3)
set
W[-5]
latched
W[-4]
latched
s(-5) = 0
latched
s(-4) = 0
latched
3
4
first
significant
z(j)
produced

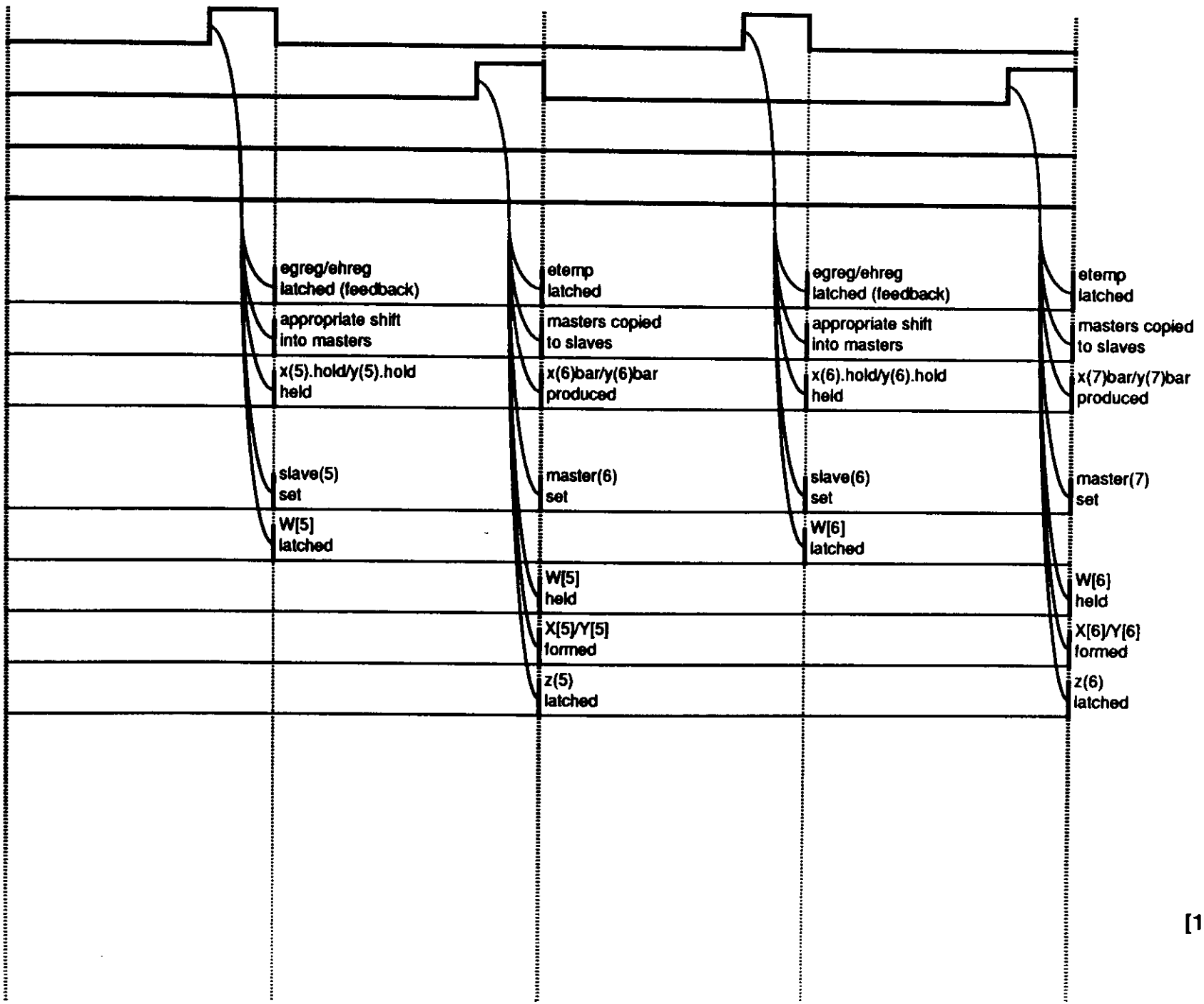
[2,2]


egreg/ehreg
latched (feedback)
appropriate shift
into masters
x(3).hold/y(3).hold
held
slave(3)
set
W[3]
latched
etemp
latched
masters copied
to slaves
x(4)bar/y(4)bar
produced
master(4)
set
W[3]
held
X[3]/Y[3]
formed
z(3)
latched
egreg/ehreg
latched (feedback)
appropriate shift
into masters
x(4).hold/y(4).hold
held
slave(4)
set
W[4]
latched
etemp
latched
masters copied
to slaves
x(5)bar/y(5)bar
produced
master(5)
set
W[4]
held
X[4]/Y[4]
formed
z(4)
latched

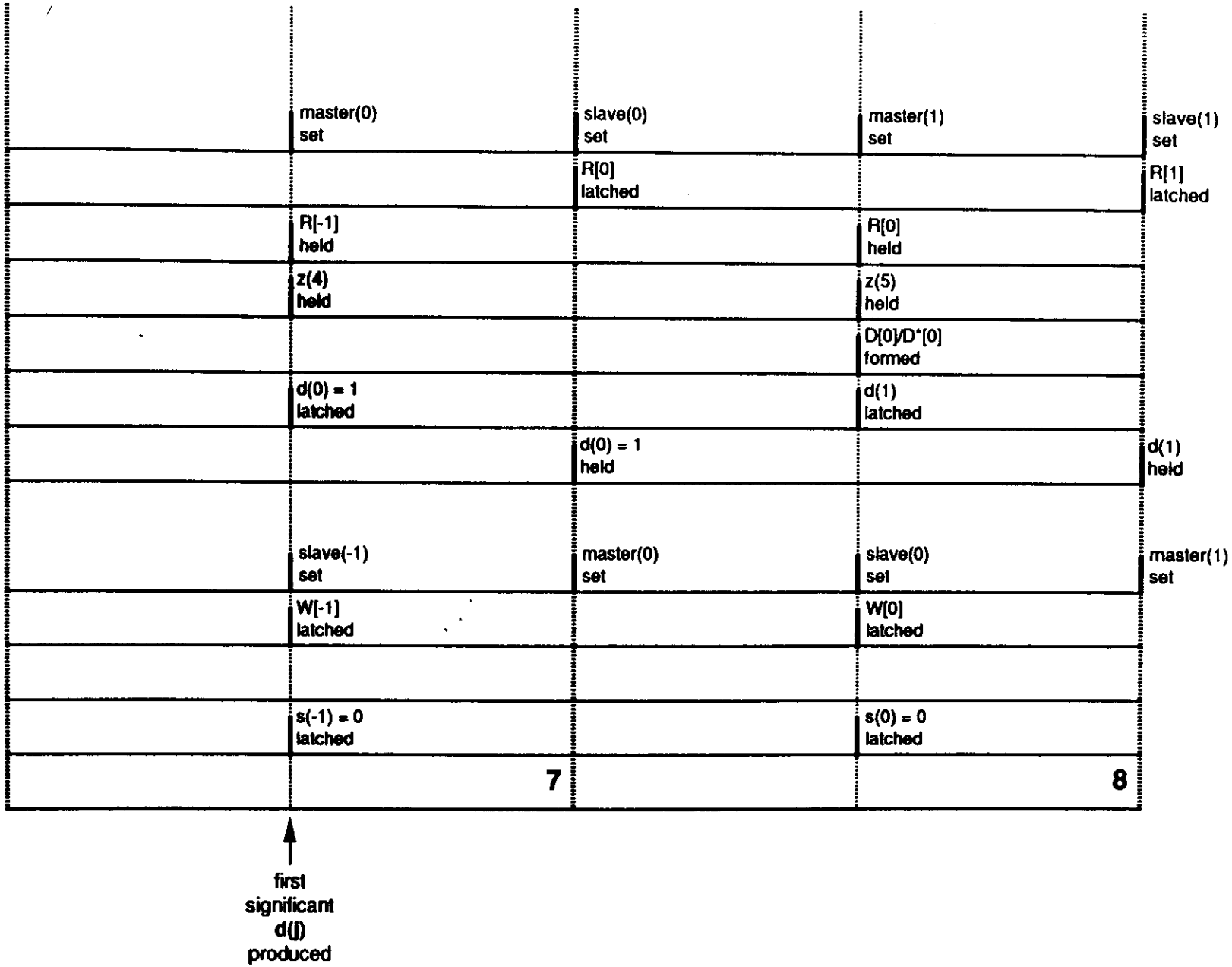
[1,3]


master(-2)
set
slave(-2)
set
master(-1)
set
slave(-1)
set
R[-2]
latched
R[-1]
latched
R[-3]
held
R[-2]
held
z(2)
held
z(3)
held
d(-2) = 1
latched
d(-1) = 1
latched
d(-2) = 1
held
d(-1) = 1
held
slave(-3)
set
master(-2)
set
slave(-2)
set
master(-1)
set
W[-3]
latched
W[-2]
latched
s(-3) = 0
latched
s(-2) = 0
latched
5
6

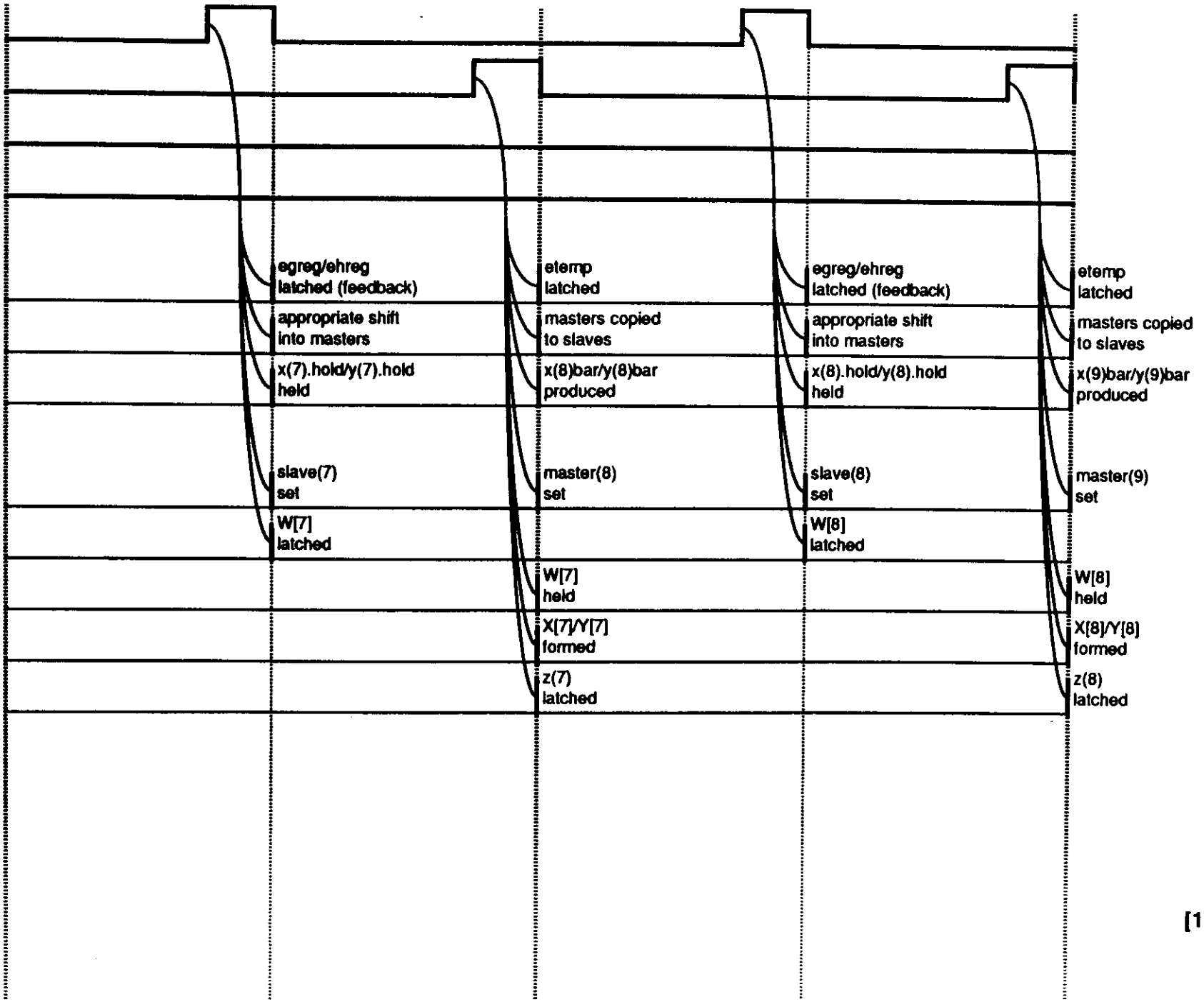
[2,3]


egreg/ehreg
latched (feedback)
appropriate shift
into masters
x(5).hold/y(5).hold
held
slave(5)
set
W[5]
latched
etemp
latched
masters copied
to slaves
x(6)bar/y(6)bar
produced
master(6)
set
W[5]
held
X[5]/Y[5]
formed
z(5)
latched
egreg/ehreg
latched (feedback)
appropriate shift
into masters
x(6).hold/y(6).hold
held
slave(6)
set
W[6]
latched
etemp
latched
masters copied
to slaves
x(7)bar/y(7)bar
produced
master(7)
set
W[6]
held
X[6]/Y[6]
formed
z(6)
latched

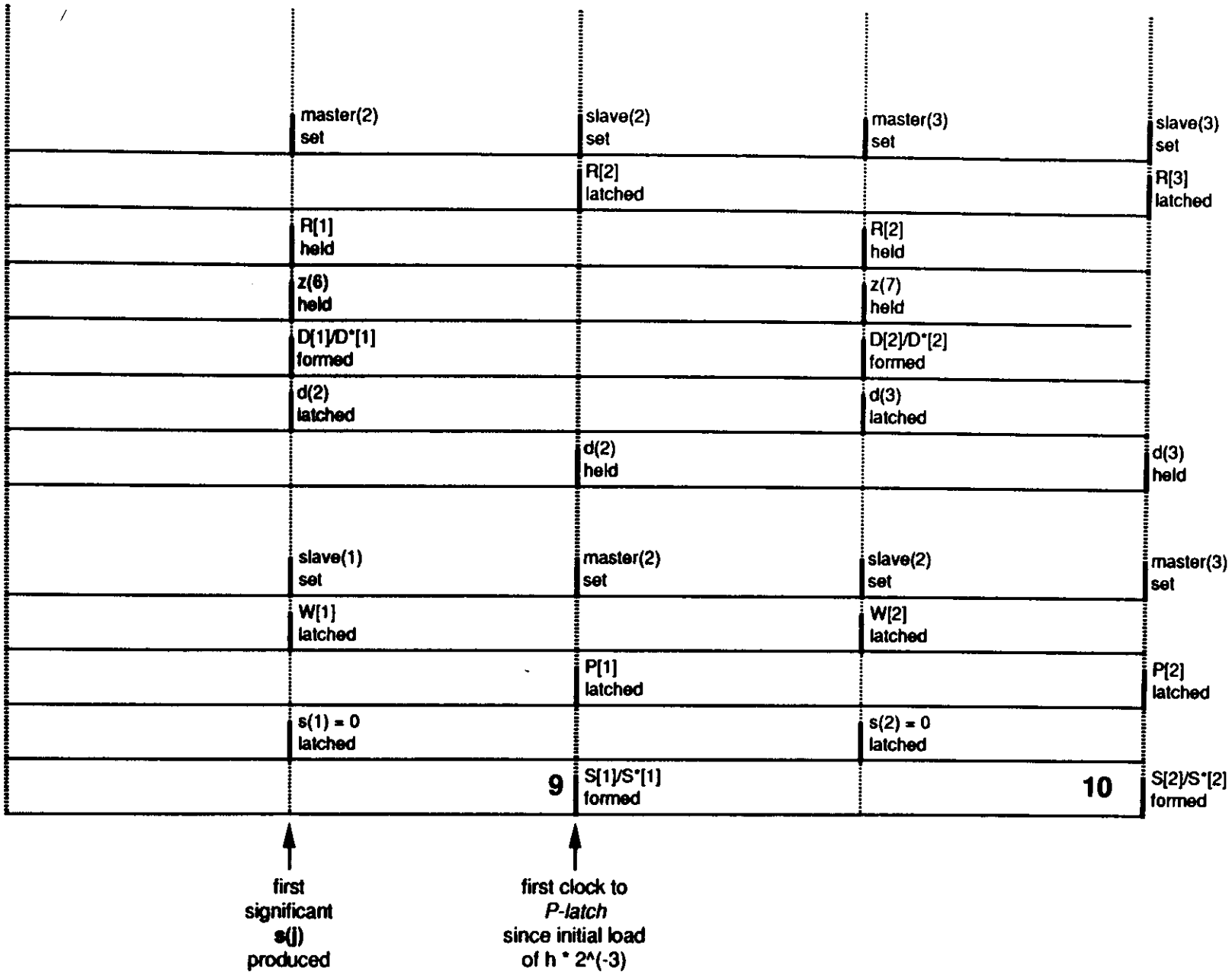
[1,4]


master(0)
set
slave(0)
set
master(1)
set
slave(1)
set
R[0]
latched
R[1]
latched
R[-1]
held
R[0]
held
z(4)
held
z(5)
held
D[0]/D*[0]
formed
d(0) = 1
latched
d(1)
latched
d(0) = 1
held
d(1)
held
slave(-1)
set
master(0)
set
slave(0)
set
master(1)
set
W[-1]
latched
W[0]
latched
s(-1) = 0
latched
s(0) = 0
latched
7
8
first
significant
d(j)
produced

[2,4]

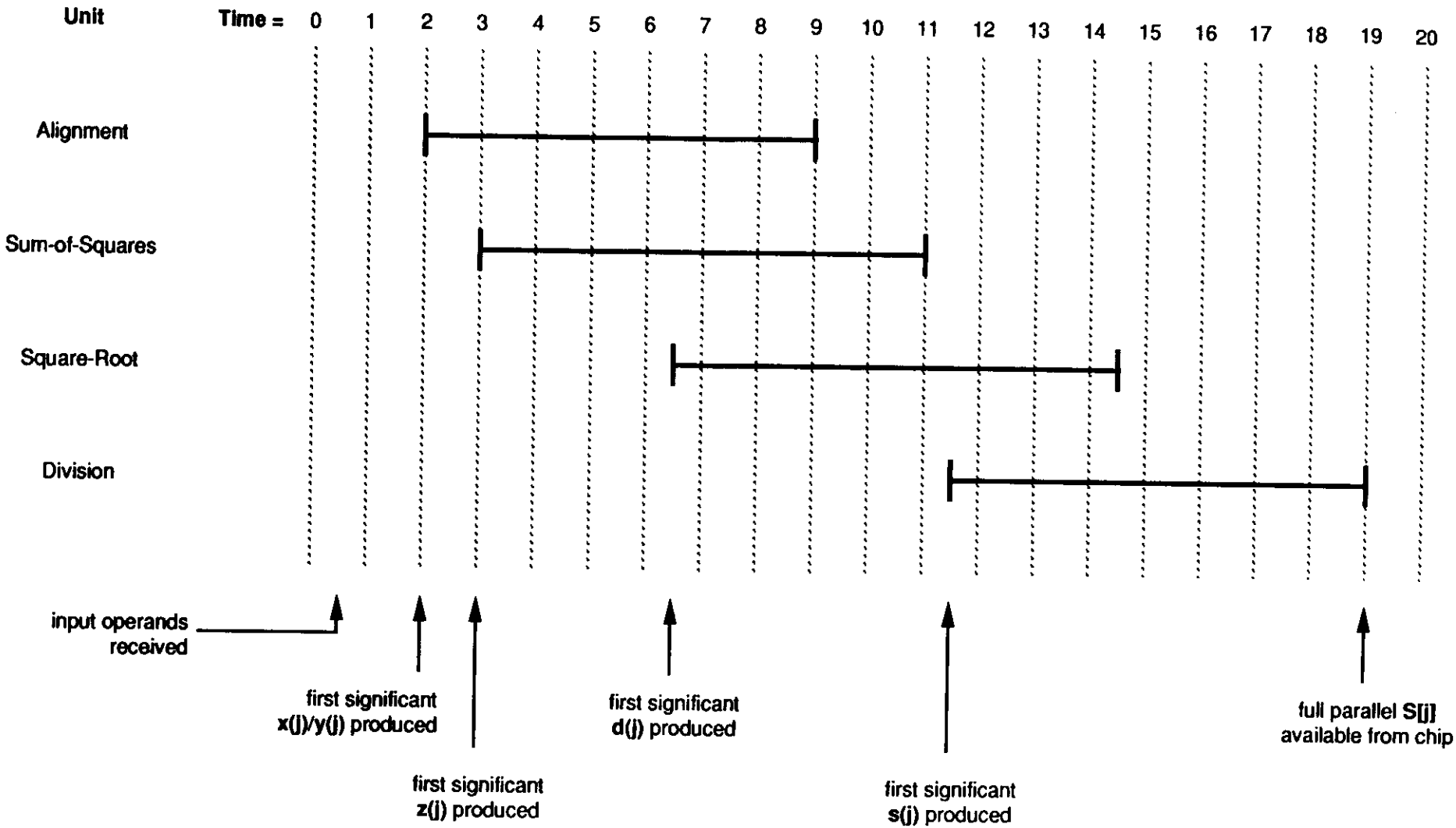

egreg/ehreg
latched (feedback)
appropriate shift
into masters
x(7).hold/y(7).hold
held
slave(7)
set
W[7]
latched
etemp
latched
masters copied
to slaves
x(8)bar/y(8)bar
produced
master(8)
set
W[7]
held
X[7]/Y[7]
formed
z(7)
latched
egreg/ehreg
latched (feedback)
appropriate shift
into masters
x(8).hold/y(8).hold
held
slave(8)
set
W[8]
latched
etemp
latched
masters copied
to slaves
x(9)bar/y(9)bar
produced
master(9)
set
W[8]
held
X[8]/Y[8]
formed
z(8)
latched

[1,5]


master(2) set
slave(2) set
master(3) set
slave(3) set
R[2] latched
R[3] latched
R[1] held
R[2] held
z(6) held
z(7) held
D[1]/D*[1] formed
D[2]/D*[2] formed
d(2) latched
d(3) latched
d(2) held
d(3) held
slave(1) set
master(2) set
slave(2) set
master(3) set
W[1] latched
W[2] latched
P[1] latched
P[2] latched
s(1) = 0 latched
s(2) = 0 latched
9
S[1]/S*[1] formed
10
S[2]/S*[2] formed
first significant s(j) produced
first clock to P-latch since initial load of h * 2^(-3)

[2,5]

• **Timing Summary: Online Delays, Overlapped Operation, and Output Availability**



1.4 Layout Dimensions (SS)

• Unit Dimensions

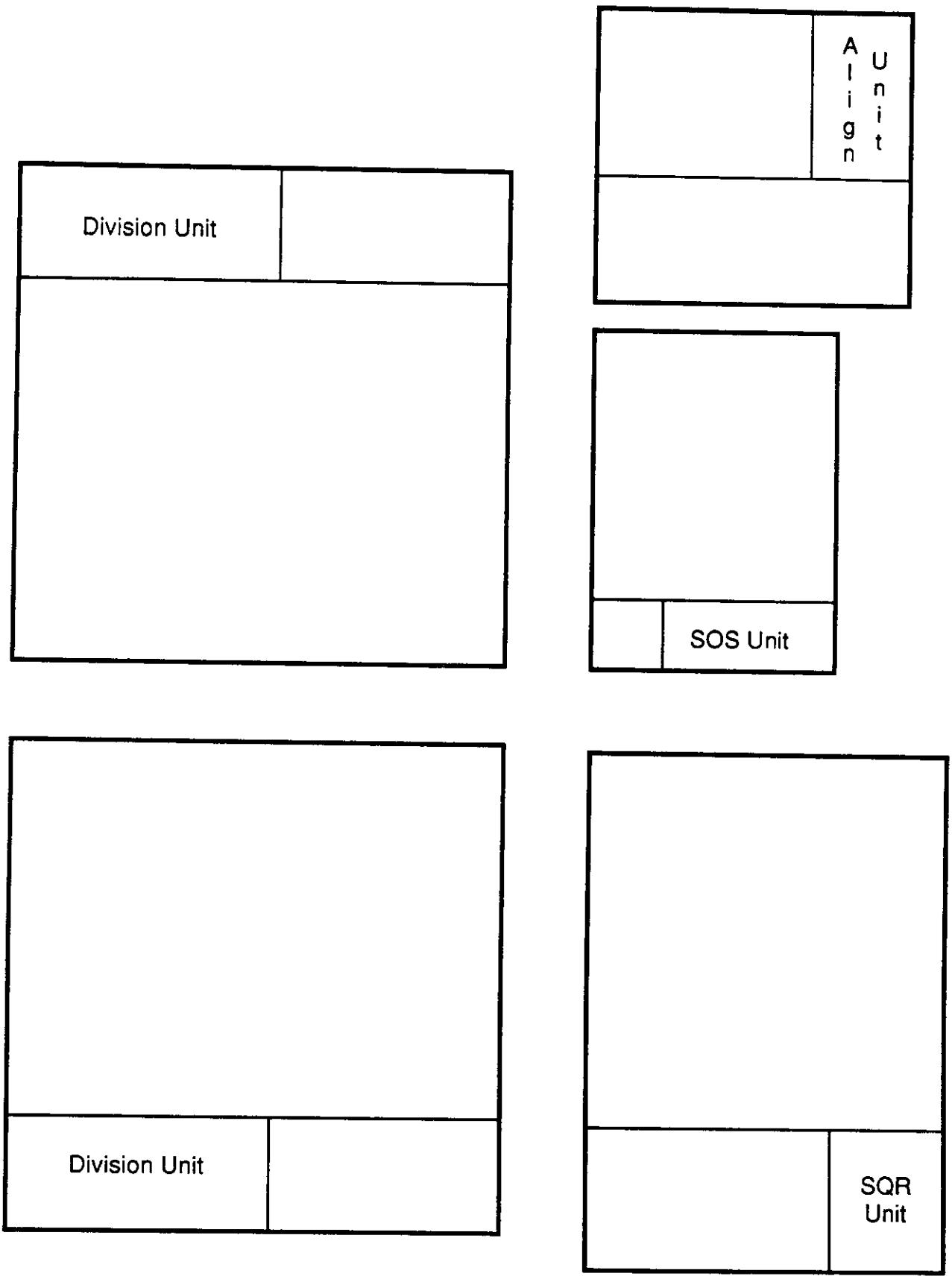
Unit	Subunit	Dimensions (H x W in λ)	
Alignment	align.top	621 x 754	Unit = 1066 x 1112 λ
	align.bot	437 x 1108	1599 x 1668 μ
Sum-of-Squares	sos.left	963 x 193	Unit = 1185 x 855 λ
	sos.mid	963 x 82	1778 x 1283 μ
	sos.right	963 x 84	
	sos.cpa	218 x 227	
Square-Root	sqr.left	1352 x 398	Unit = 1827 x 1230 λ
	sqr.mid	1352 x 98	2741 x 1845 μ
	sqr.mid3	1352 x 98	
	sqr.mid4	1352 x 98	
	sqr.mid5	1352 x 98	
	sqr.right	1352 x 142	
	sqr.cpa	467 x 832	
Division	div.left	1349 x 482	Unit = 1744 x 1689 λ
	div.mid0	1349 x 111	2616 x 2534 μ
	div.mid	1349 x 119	
	div.right	1349 x 148	
	div.cpa	387 x 758	

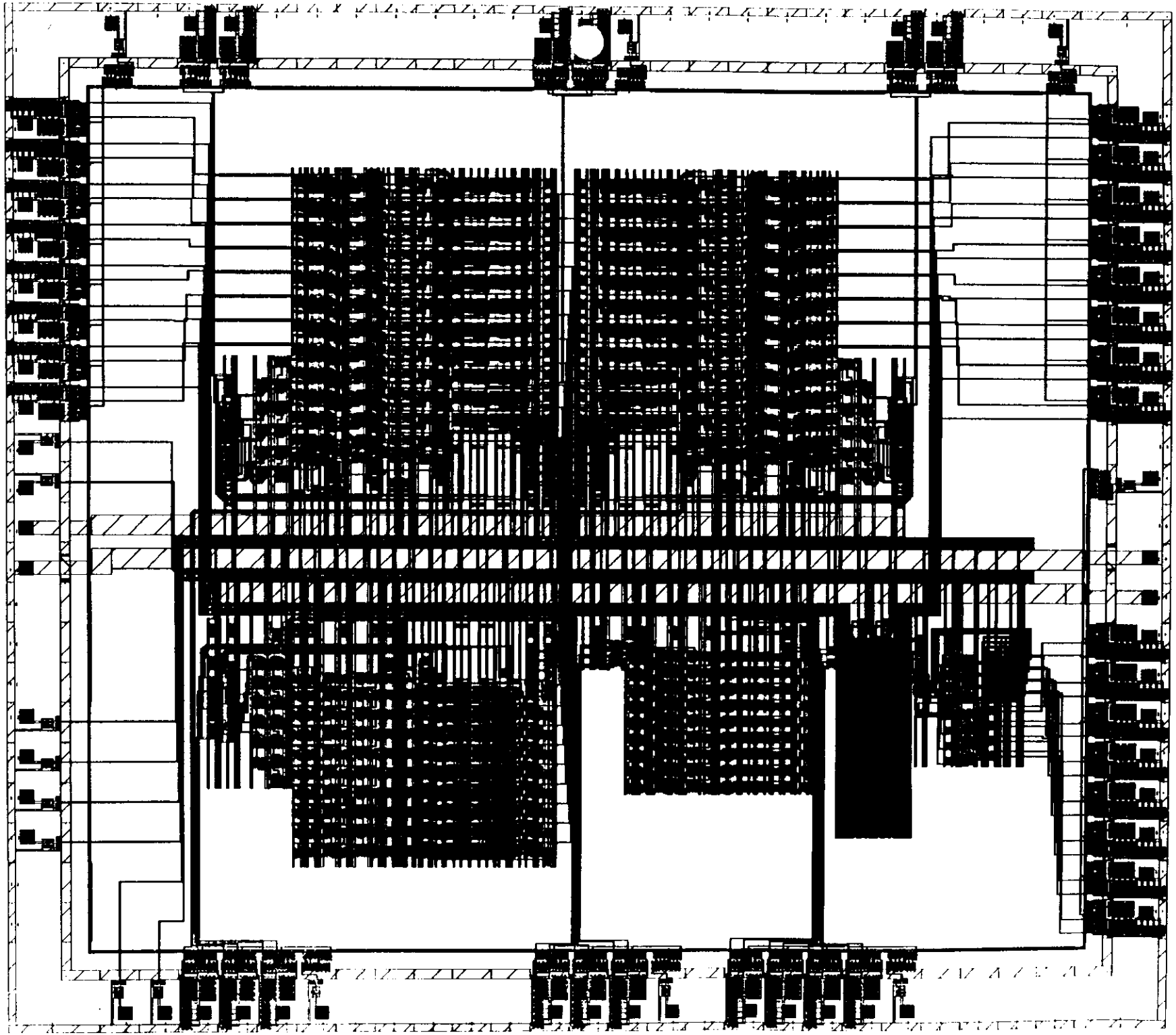
• Chip Dimensions

Area	Dimensions (H x W in λ)	Dimensions (H x W in μ)
Internal Circuitry	4310 x 3523	6465 x 5285
Internal Circuitry + Wire Channel Frame	5080 x 4280	7620 x 6420
Internal Circuitry + Wire Channel Frame + Pads	5934 x 5134	8901 x 7701

• Floorplan

Scale: 1 inch = 500 lambda. All Dimensions scaled to nearest 1/10 inch.


Align Unit
Division Unit
SOS Unit
Division Unit
SQR Unit

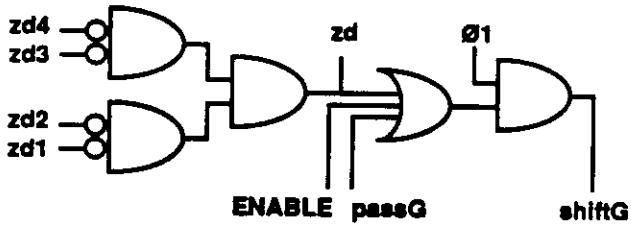
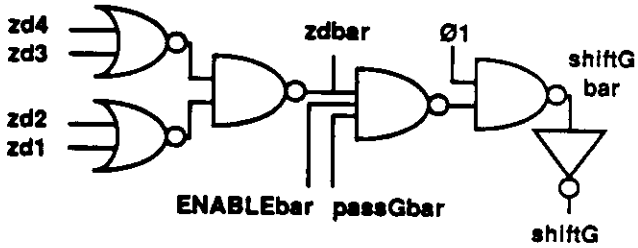
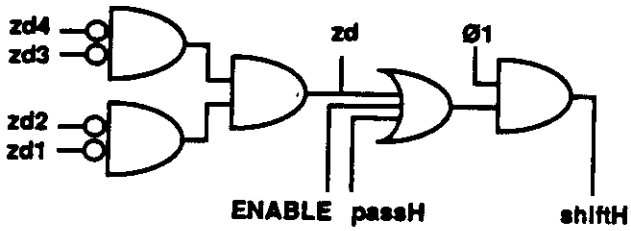
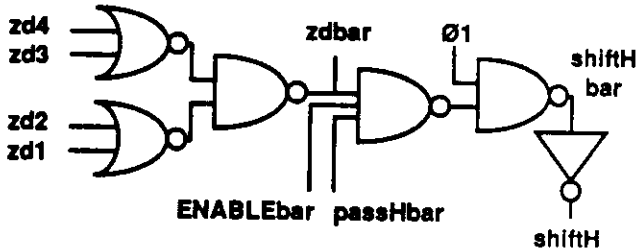


2.1 Alignment Unit (SS)

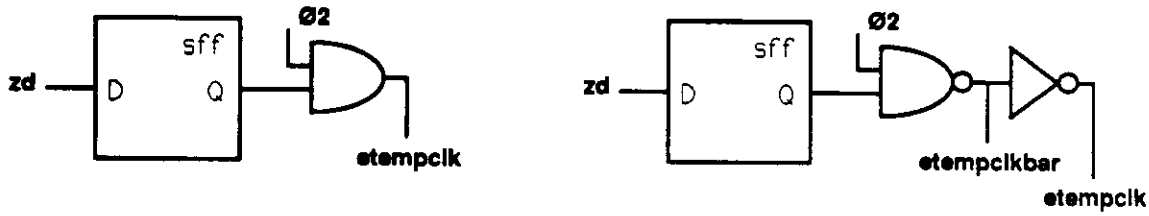
• Registers/Levels

		Name	Type	Function
ARITHMETIC	SECTION	*egreg*	muxff	initially loads **eg**; thereafter acts as an intermediary register
		ehreg	muxff	initially loads **eh**; thereafter loads either 0 or -2
		cpa-l	fad	initially computes **etemp** = **ediff** = (**eg** - **eh**) = **eg** + (**ehbar** + 1); thereafter used to count **etemp** either up or down to 0. A constant input of 1 is forced at right.
		eclock-gen	logic/sff	generates clock enables which are ANDed with Ø2 to clock *etemp/ediff*
		etemp	dff	stores the original exponent difference **etemp** = **ediff** = (**eg** - **eh**); thereafter used as an intermediary register during countup/countdown for 0 detection.
		shiftX-gen	logic	detects 0-difference case, generates clock enable which is ANDed with Ø1 to clock **gshift/hshift**
		ediff	dff	stores the original exponent difference, ie. **etemp** = **ediff** = (**eg** - **eh**) for all time; used in determining **es** & **ec**
		cpa-r	fad	calculates - **ediff** = **ediffbar** + 0 + 1
		es-gen/ ec-gen	nor2	selects **es** = - **ediff** (**ediff** ≥ 0) or **es** = 0 (**ediff** < 0); selects **ec** = 0 (**ediff** ≥ 0) or **ec** = **ediff** (**ediff** < 0)
SHIFTING	SECTION			
		gshift	muxshift	initially loads mantissa(**g**) = . **g1** . . . **g8**; thereafter used to shift in **g** (as **x**)
		hshift	muxshift	initially loads mantissa(**h**) = . **h1** . . . **h8**; thereafter used to shift in **h** (as **y**)

• Logic Block Reductions

Name	Function	Implementation
shiftX-gen		
		

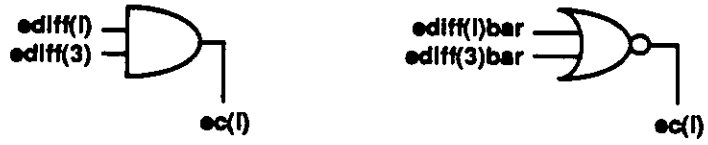
eclock-gen



eclock-gen

es-gen

ec-gen

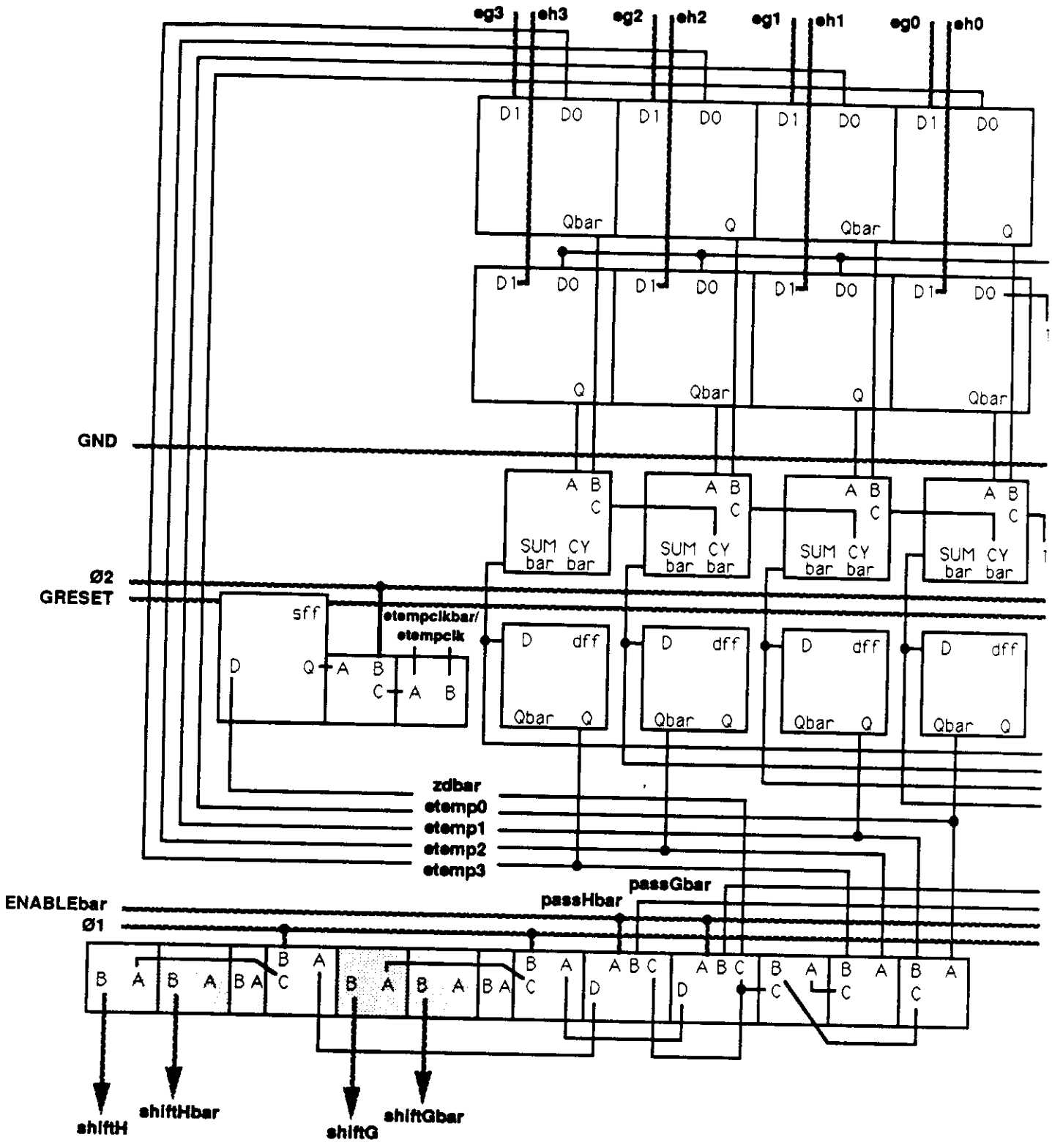


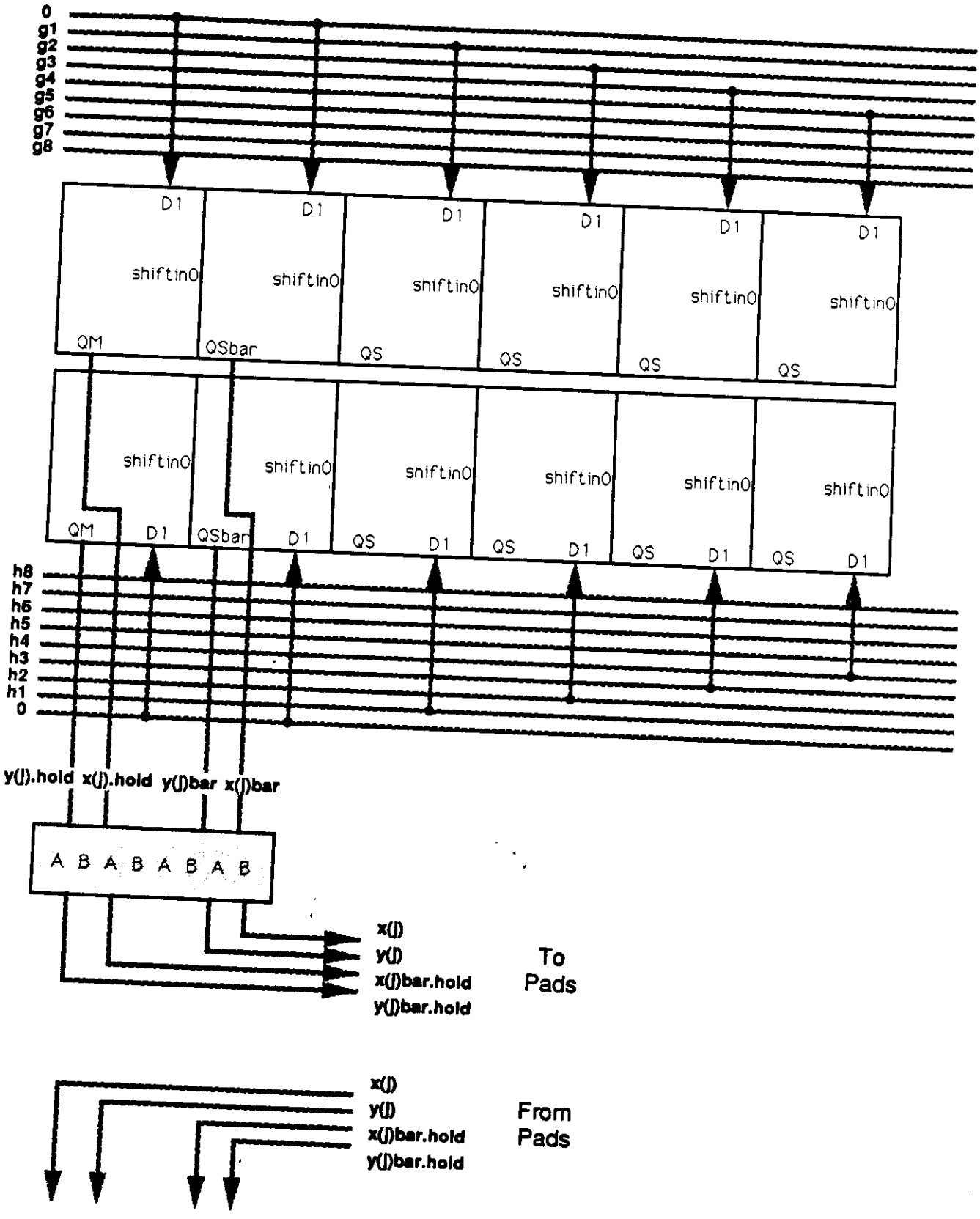
Operation

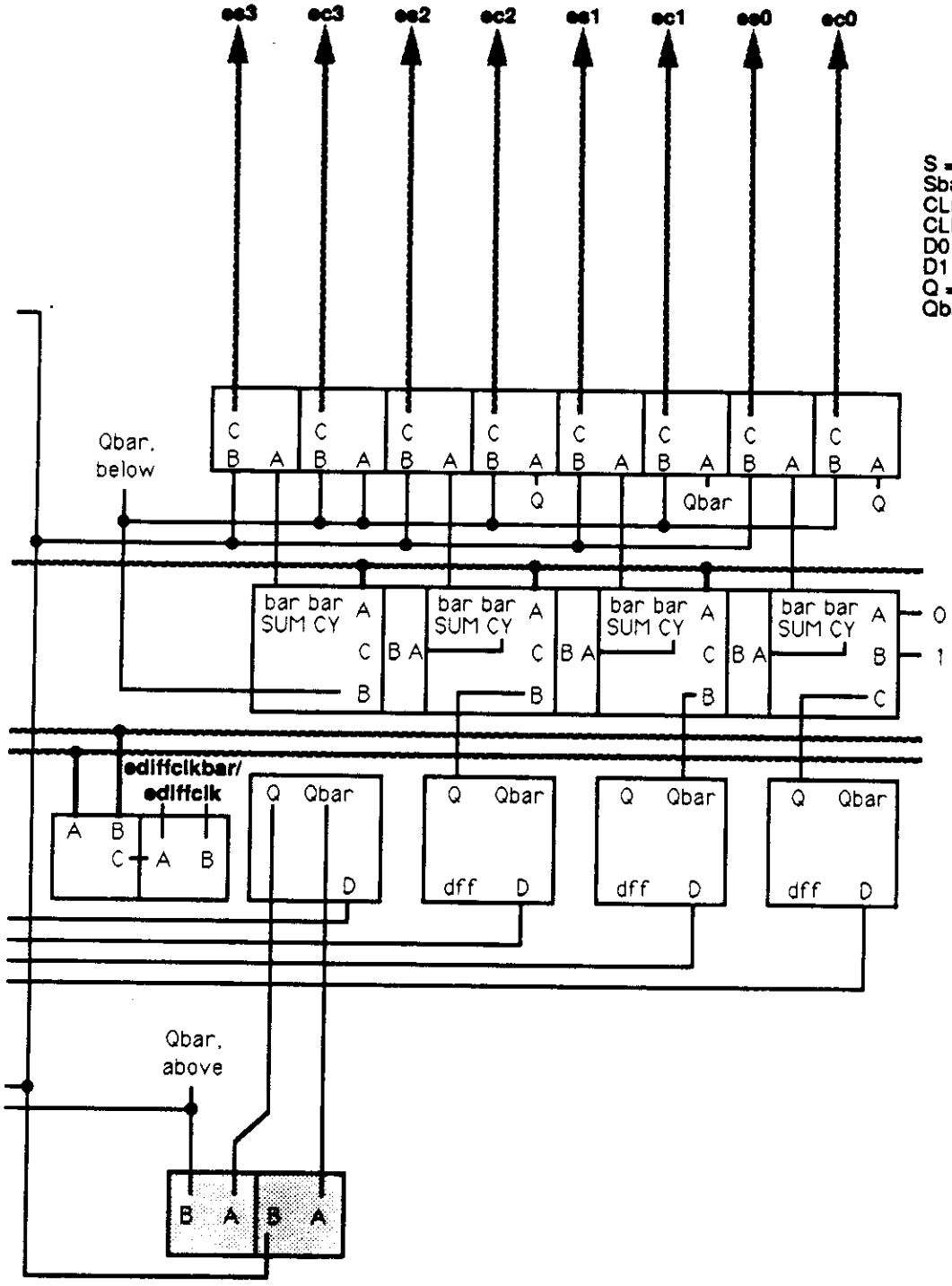
At the beginning of the chip's operation, the **ENABLE** signal is used to re-direct the inputs of *egreg* and *ehreg* to accept the initial inputs **eg** and **eh** (a constant 1 is forced at the right end of *cpa-l* to form - **eh** = **ehbar** + 1). Thereafter, these registers are utilized as a temporary intermediary register and as a source of either a -2 or 0, respectively. When added to the aforementioned constant 1 fed into the right end of *cpa-l*, either a -1 or 1 respectively is formed, and any exponent difference is counted either up or down to 0 using *etemp*. Clocking of *etemp* ceases when 0 is reached. *ediff* receives only a single clock; it's sole function is to store the original exponent difference for computation of **es** and **ec**. *cpa-r* generates - **ediff** from **ediffbar**, 0, and 1. Immediately thereafter, *es-gen* selects either **es** = - **ediff** (**ediff** ≥ 0) or **es** = 0 (**ediff** < 0), while *ec-gen* selects **ec** = 0 (ediff ≥ 0) or **ec** = **ediff** (**ediff** < 0).

shiftG/shiftH are formed by ORing the **ENABLE** signal (which brings in the original mantissas), **zd** (valid when *etemp* = 0 is reached), and either **passG/passH**. **passG** is valid immediately iff the original exponent difference (**ediff**) was positive, while **passH** is valid iff **ediff** was negative. The end result is that valid **x(j)**'s and **y(j)**'s are shifted out of *gshift* and *hshift*, respectively, at their proper times.

2.2 Alignment Unit (DS)




0
g1
g2
g3
g4
g5
g6
g7
g8
D1
shiftin0
QM
QSbar
QS
h8
h7
h6
h5
h4
h3
h2
h1
0
y(J).hold
x(J).hold
y(J)bar
x(J)bar
A B A B A B A B
x(J)
y(J)
x(J)bar.hold
y(J)bar.hold
To Pads
From Pads



ARITHMETIC SECTION

muxff *(egreg/ ehreg)*

S = ENABLE
Sbar = ENABLEbar
CLK = Ø1
CLKbar = Ø1bar
D0 = shown
D1 = shown
Q = shown
Qbar = shown

nor2 *(es-gen/ ec-gen)*

all shown

fad *(cpa-l cpa-r)*

all shown

logic/sff *(eclock-gen)*

PRESET = GRESET
D = shown
CLK = Ø1
CLK = Ø1bar
Q = shown
Qbar = unused

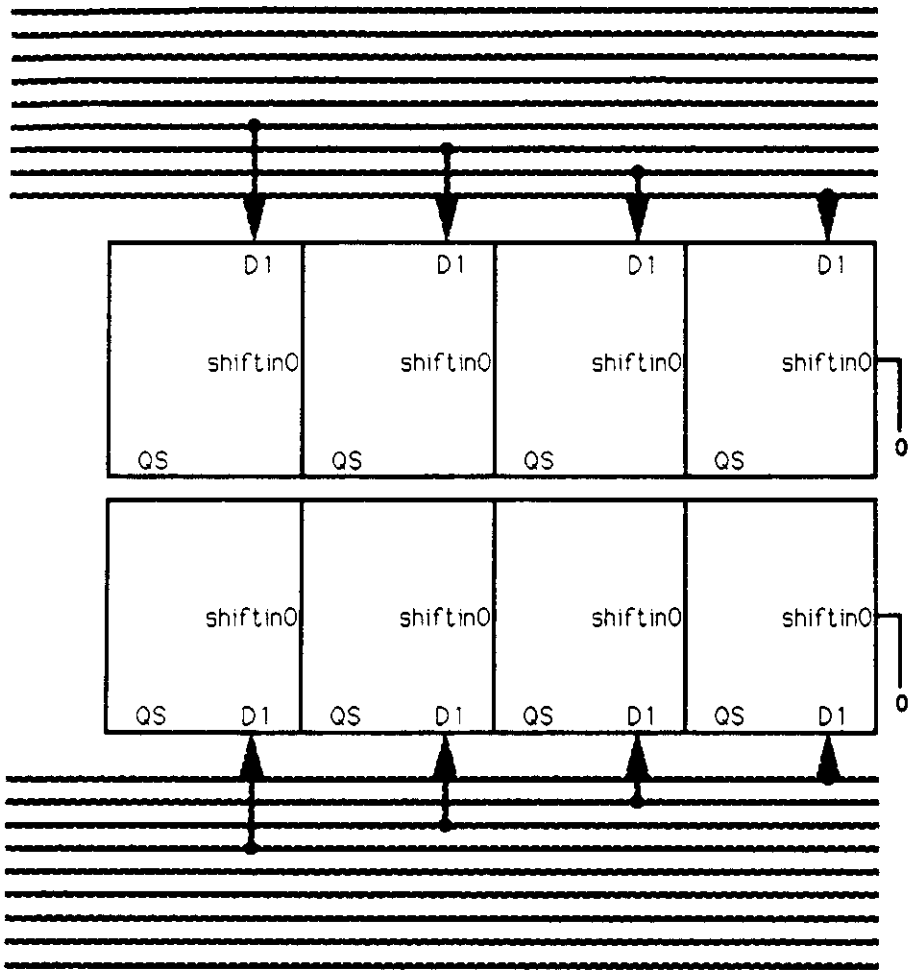
dff *(etemp/ ediff)*

D = shown
CLK = eXXXXclk, left
CLK = eXXXXclkbar, left
Q = shown
Qbar = shown

logic *(shiftX-gen)*

all shown

[1,2]



SHIFTING SECTION

muxshift

S = ENABLE *(gshift)*
Sbar = ENABLEbar
shiftin0 = auto
D1 = shown
CLKM = shiftG/shiftH
CLKMbar = shiftGbar/shiftHbar
CLKS = Ø2
CLKSbar = Ø2bar
QM = shown
QMbar = unused *(hshift)*
QS = unused
QSbar = shown
shiftout = auto

3.1 Sum-of-Squares Unit (SS)

• Recurrence Operands

W[j] = 2*W[j-1] — previous value of W[j] held in *W-hold.ps / W-hold.sc*

+ 2*{X[j-1] + x(j)*2^-(j+1)}*x(j) — 2*{Qx}*x(j), produced by *Qx-low* of operand section

+ 2*{Y[j-1] + y(j)*2^-(j+1)}*y(j) — 2*{Qy}*y(j), produced by *Qy-low* of operand section

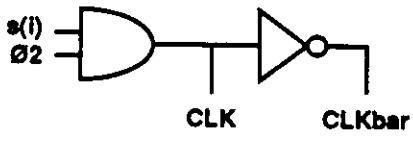
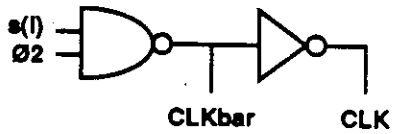
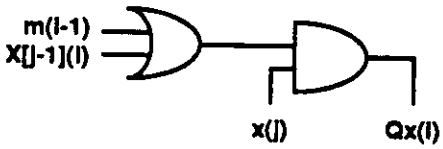
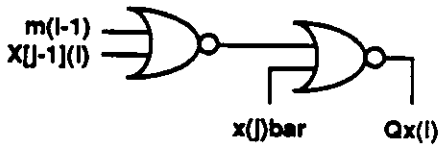
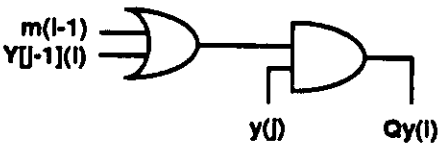
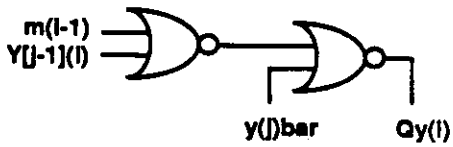
Bit Patterns: Qx = [X , 0 , 1(j+1) , 0 , 0 , ... , 0]
Qy = [Y , 0 , 1(j+1) , 0 , 0 , ... , 0]

Qx/Qy are gated as {Qx}*x(j)/{Qy}*y(j) and the result is fed into the arithmetic section with elements correctly positioned above csa/latch structures to accomplish the multiplication by 2

• Registers/Levels

		Name	Type	Function
OPERAND	SECTION	*shift-register*	rsff [sff in m(0)]	Traveling bit pair: **m(j)** is used to force the low bit of **Qx/Qy** high; **s(j)** marks the location where appending occurs to produce **X[j]/Y[j]**
		append-clock	nand2/inv	syncs the bit marking the location where appending is to occur [**s(j)**] with the **Ø2** clock
		X-latch	rsff	stores the appended value of **X[j]**
		Y-latch	rsff	stores the appended value of **Y[j]**
		Qx-low	nor2	ORing logic used to force **Qx(j+1)** high, followed by kill logic which forces **Qx=0** for **x(j)=0**
		Qy-low	nor2	ORing logic used to force **Qy(j+1)** high, followed by kill logic which forces **Qy=0** for **y(j)=0**
ARITHMETIC	SECTION	*W-hold.ps*	rsff	holds over the partial-sum component of **W[j-1]** during the 4-2 reduction forming **W[j]**
		csa-level1	fad	performs 3-2 reduction of 2*Qx*x(j), 2*Qy*y(j), and *W-hold.ps*
		W-hold.sc	rsff	holds over the saved-carry component of **W[j-1]** during the 4-2 reduction forming **W[j]**
		csa-level2	fad	performs 3-2 reduction of the outputs of *csa-level1* plus *W-hold.sc*
		W-latch.ps	rsff	contains the partial-sum component of **W[j]**
		W-latch.sc	rsff	contains the saved-carry component of **W[j]**
SELECTION	SECTION	*cpa*	fad	performs 2-1 reduction of *W-latch.ps*, *W-latch.sc* forming **z(j)**
		z-latch	rsff	stores the **z(j)** formed

• Logic Block Reductions

Name	Function	Implementation
append-clock		
Qx-low		
Qy-low		

• Operation

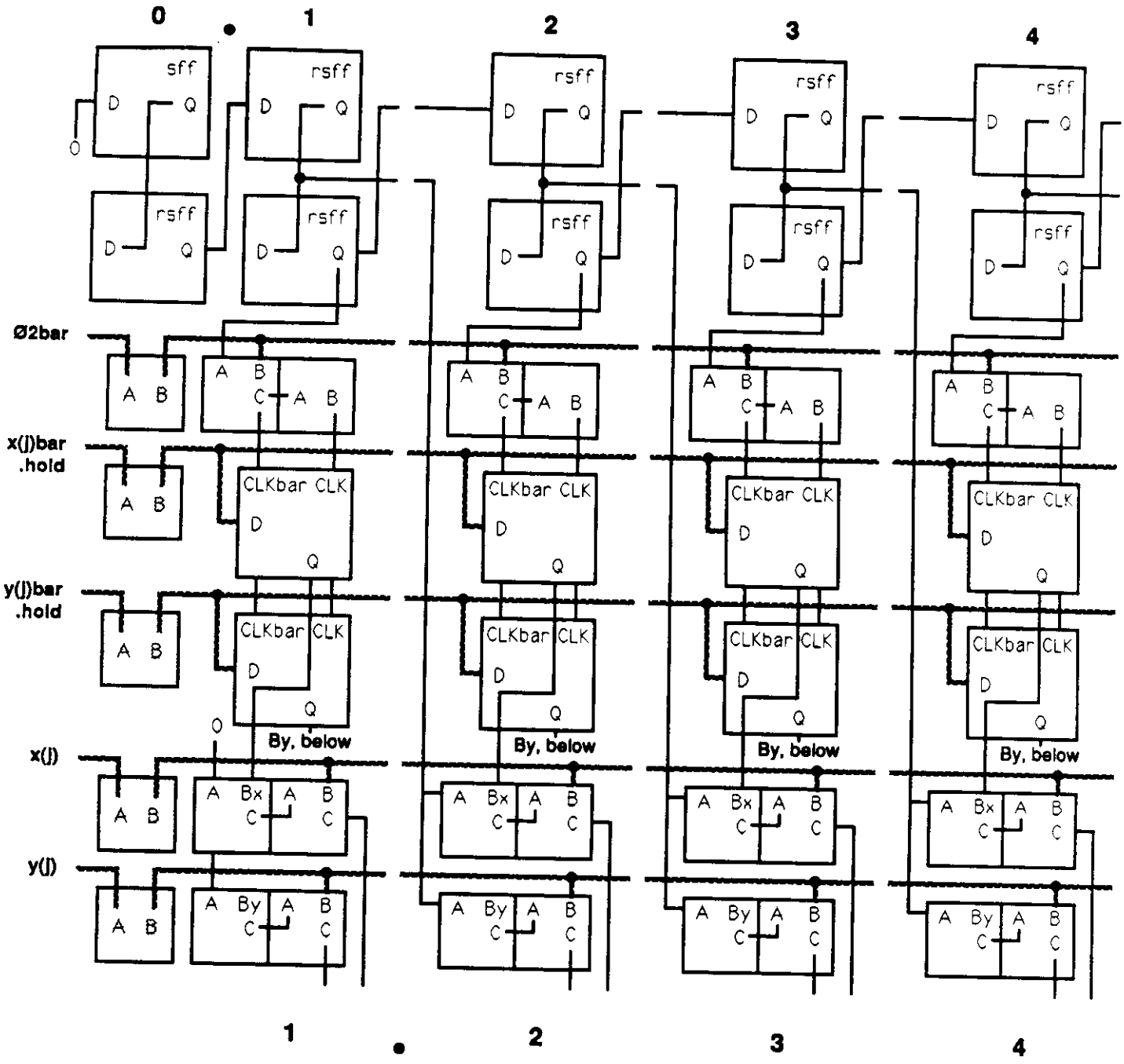
At the beginning of the chip's operation, the **GRESET** signal is used to preset all latches to 0 [except for **m(0)** of *shift-register*, which is set to 1]. It is proper then to examine the operation of each section separately.

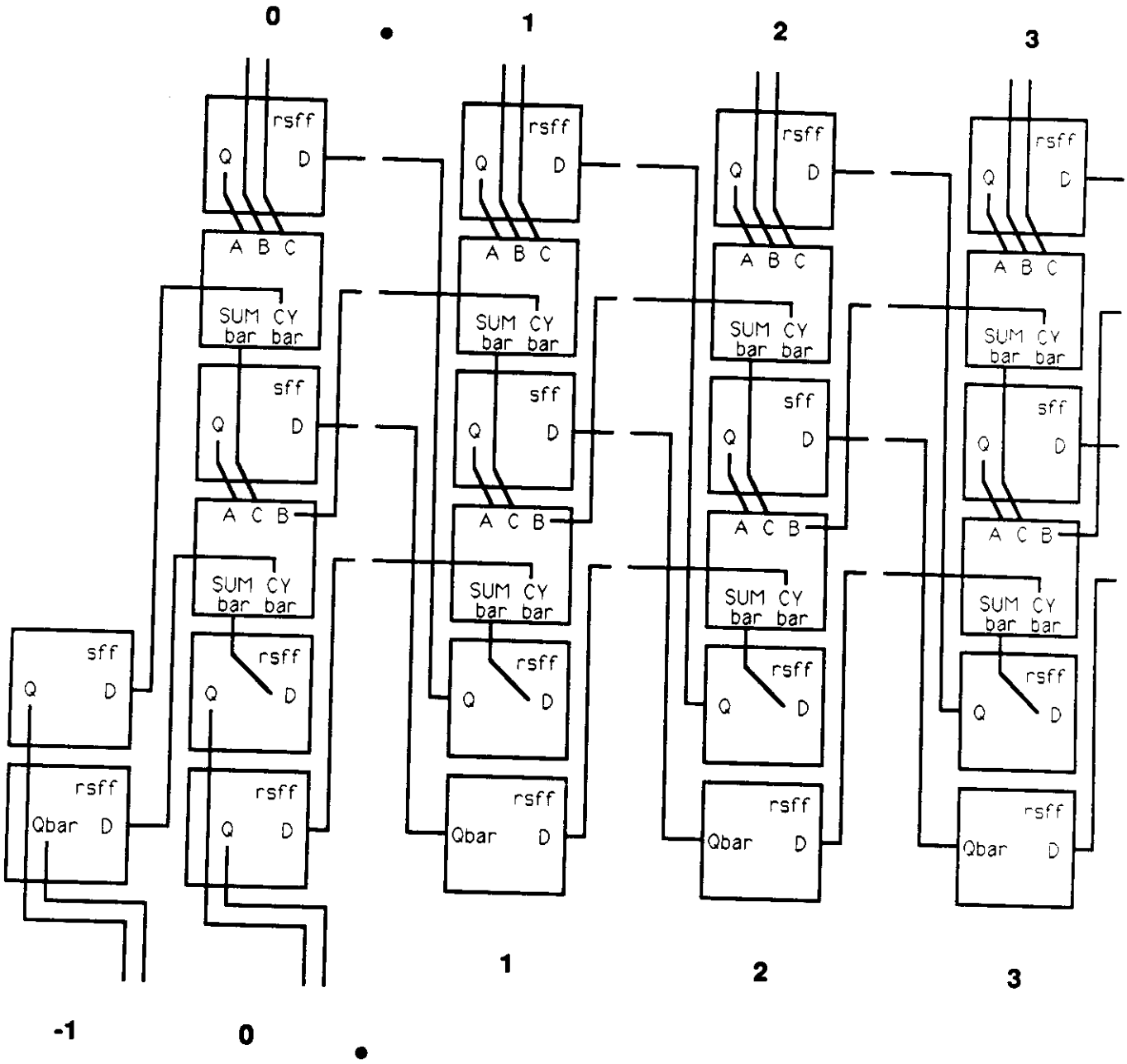
In the operand section, a bit pair travels across the shift register such that position j can be determined. When **m(j)** is set, it is used to force bit j+1 of **Qx/Qy** high. This bit lives logically in slice j+1. The entire operand section sits above the arithmetic section so that **Qx/Qy**, after gating as **Qx*x(j)/Qy*y(j)**, are fed into the arithmetic section with elements correctly positioned above csa/latch structures. In the next half cycle, **s(j)** is used to mark the position where appending to form **X[j]/Y[j]** (in *X-latch/Y-latch*) should take place, and *append-clock* logic syncs this bit with the **Ø2** clock.

In the arithmetic section, **2*Qx*x(j)**, **2*Qy*y(j)**, and the partial-sum and carry-save components of **W[j-1]** repeatedly undergo a 4-2 reduction to form **W[j]**. The fractional portion of **W[j]** is routed back up to registers *W-hold.ps/W-hold.sc* for computation of the next **W[j]**, while the integer portion is routed to the selection section for **z(j)** determination.

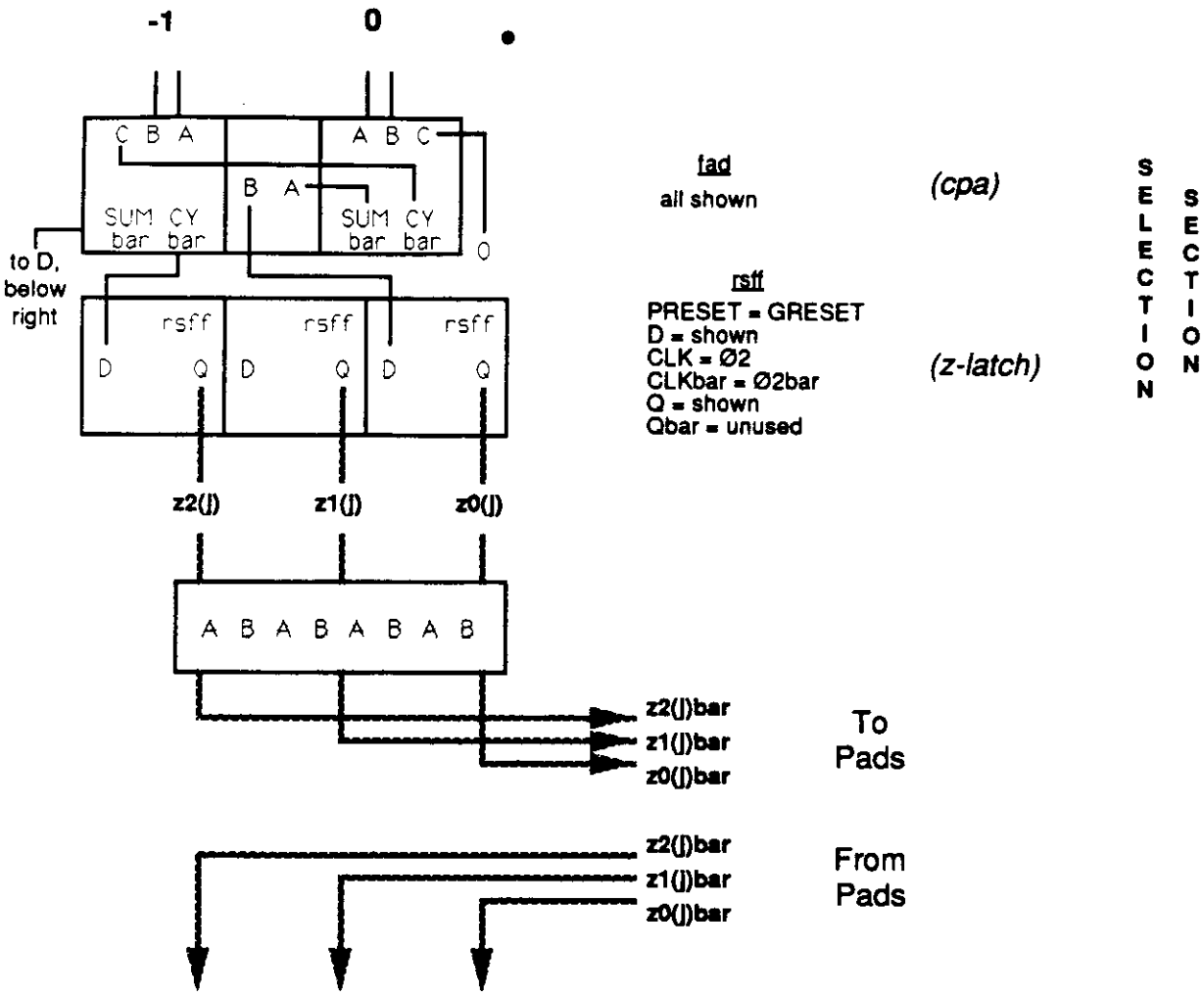
In the selection section, the integer portion of **W[j]**, now in carry-save form, undergoes a 2-1 reduction in *cpa*, and the resulting **z(j)** is held in the *z-latch*.

3.2 Sum-of-Squares Unit (DS)

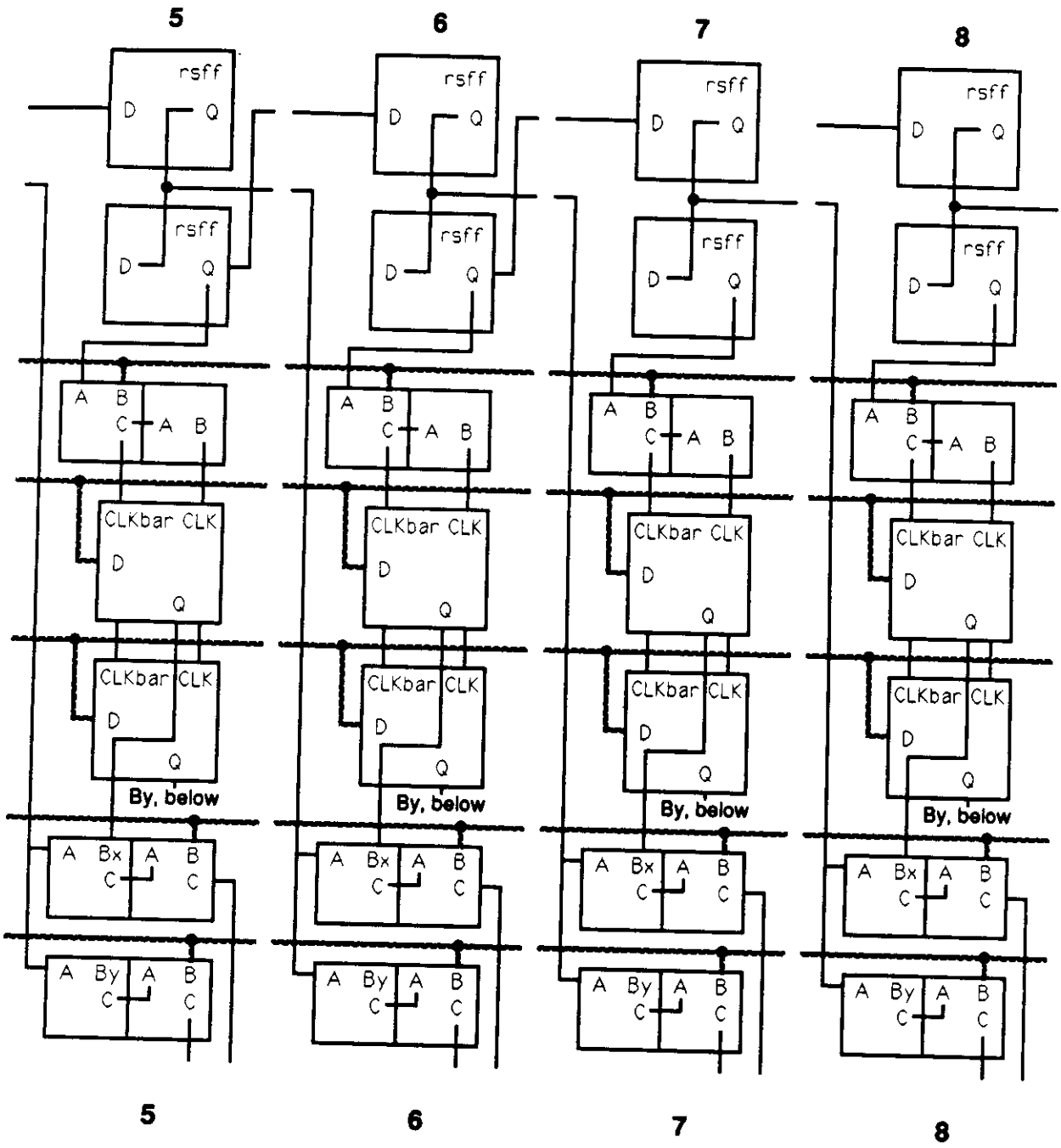




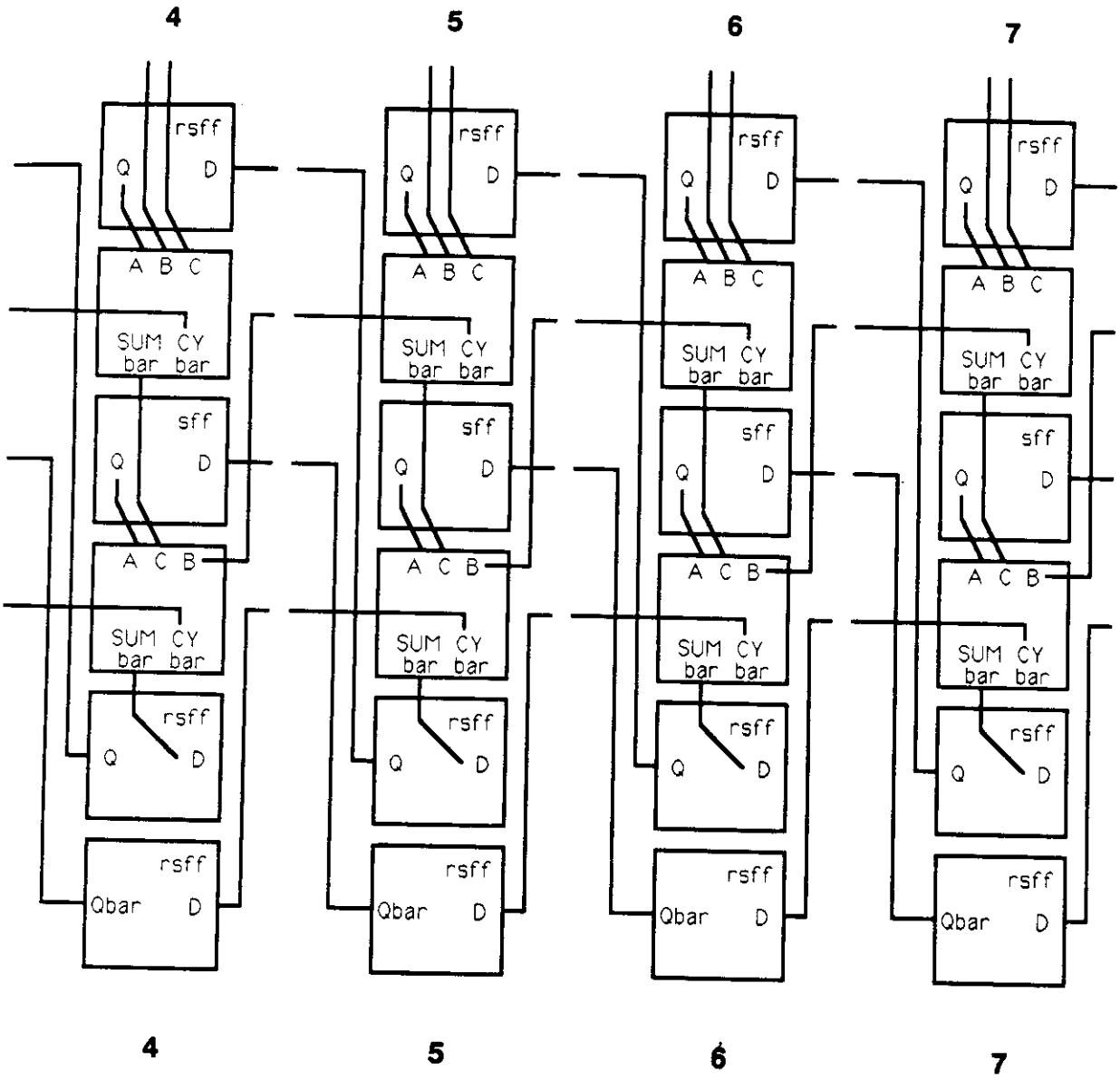
[2,1]


-1
0
C B A
A B C
B A
SUM CY
bar bar
SUM CY
bar bar
0
to D,
below
right
rsff
rsff
rsff
D
Q
D
Q
D
Q
z2(j)
z1(j)
z0(j)
A B A B A B A B
z2(j)bar
z1(j)bar
z0(j)bar
To
Pads
z2(j)bar
z1(j)bar
z0(j)bar
From
Pads
fad
all shown
(cpa)
rsff
PRESET = GRESET
D = shown
CLK = Ø2
CLKbar = Ø2bar
Q = shown
Qbar = unused
(z-latch)
SELECTION
SECTION

[3,1]

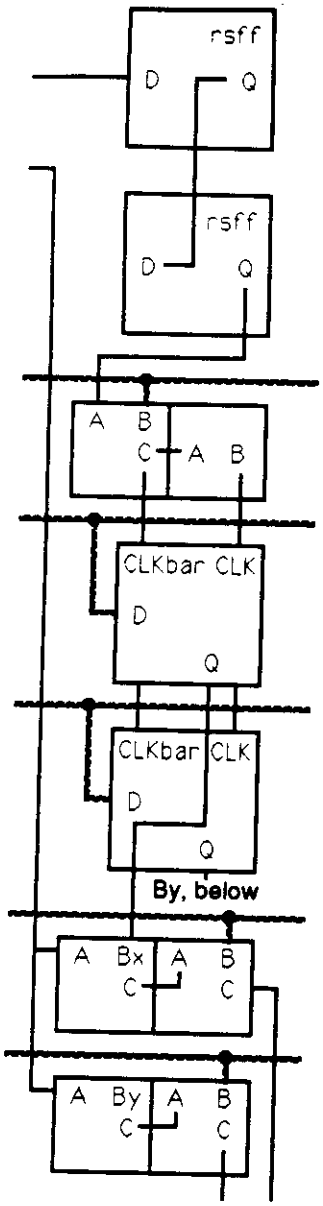


[1,2]



[2,2]

9



rsff

[Except sff in m(0)]

PRESET = GRESET
D = shown
CLK = Ø2
CLKbar = Ø2bar
Q = shown
Qbar = unused

(shift-register)

PRESET = GRESET
D = shown
CLK = Ø1
CLKbar = Ø1bar
Q = shown
Qbar = unused

nand2/inv

all shown

(append-clock)

rsff

PRESET = GRESET
D = shown
CLK = B, above
CLKbar = C, above
Q = shown
Qbar = unused

(X-latch)

PRESET = GRESET
D = shown
CLK = B, above
CLKbar = C, above
Q = shown
Qbar = unused

(Y-latch)

nor2/nor2

all shown

(Qx-low)

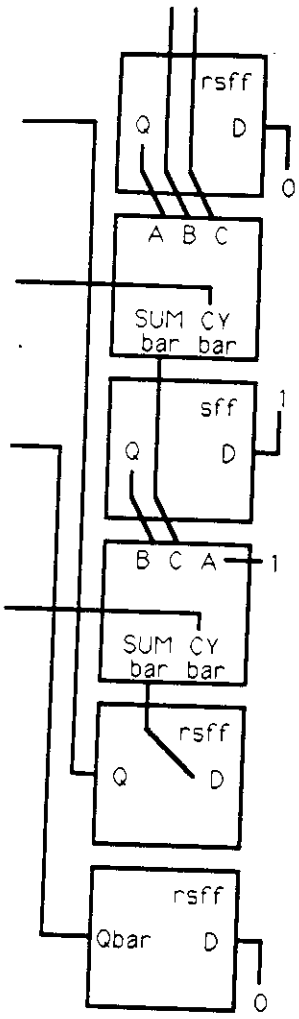
(Qy-low)

OPERAND SECTION

9

[1,3]

8



rsff

PRESET = GRESET
D = shown
CLK = Ø2
CLKbar = Ø2bar
Q = shown
Qbar = unused

(W-hold.ps)

fad

all shown

(csa-level1)

sff

PRESET = GRESET
D = shown
CLK = Ø2
CLKbar = Ø2bar
Q = shown
Qbar = unused

(W-hold.sc)

fad

all shown

(csa-level2)

rsff

PRESET = GRESET
D = shown
CLK = Ø1
CLKbar = Ø1bar
Q = shown
Qbar = unused

(W-latch.ps)

PRESET = GRESET
D = shown
CLK = Ø1
CLKbar = Ø1bar
Q = unused
Qbar = shown

(W-latch.sc)

ARITHMETIC SECTION

8

[2,3]

4.1 Square-Root Unit (SS)

• Recurrence Operands

R[j] = 2*R[j-1] — previous value of **R[j]** held in *R-hold.ps / R-hold.sc*

+ z(4+j)*2^(-4) — **{Qz}** value held in *Qz-hold*

+ 2*{-d(j)*D[j-1] - 2^(-(j+1))*d(j)^2} — 2*{**Qd**}, where **Qd** is produced in the operand section and fed into the arithmetic section with elements correctly positioned above csa/latch structures to accomplish mult. by 2

Bit Patterns:

Qd = [Dbar , 1 , 1(j+1) , 0 , 0 , ... , 0]

Qd = [0 , ... , 0 , 0(j+1) , 0 , 0 , ... , 0]

Qd = [Dstar , 1 , 1(j+1) , 0 , 0 , ... , 0]

d(j) = 1 = 1 | 0 1

d(j) = 0 = 0 | 0 0

d(j) = -1 = 0 | 1 1

d0(j)
d1(j)
d2(j) [non-binary, spec. defined]

• Registers/Levels

	Name	Type	Function
OPERAND SECTION	*shift-register*	rsff	Traveling bit train (requiring logic to detect the end, unlike a bit pair): **m(i)bar/m(i+1)** produce **force(i)** to force the low-order 2 bits of **Qd** high **s(i)bar/s(i+1)** produce **end(i)** to mark where appending occurs by: - re-directing the inputs to *D-latch/Dstar-latch* in position j, via their select lines, from the parallel load to the append input - always enabling the clocks to *D-latch/Dstar-latch* in position j **m(i+1)bar** is used in *Qd-multiplex* to convert [D,0,...,0] to [Dbar,0,...,0]
	force-bit/ end-bit	inv/nor2 nor2	**force(i)** detects the end of the train to force the low-order 2 bits of **Qd** high; **end(i)** detects the end of the train to enable the clocks to *D-latch/Dstar-latch* in position j as well as re-direct their inputs for appending **d0(j).hold** to *D-latch* and **d0(j)bar.hold** to *Dstar-latch*
	D-clock	nor2/nor2 inv	produces clock enable if either **d1(j).hold=1** [load *D-latch* from *Dstar-latch*] or **end(i)=1**, then syncs to the **Ø1** clock
	D-latch	rsmuxff	stores the on-the-fly converted value of **D[j]**
	Dstar-latch	rsmuxff	stores the on-the-fly converted value of **Dstar[j]**
	Dstar-clock	nor2/nor2 inv	produces clock enable if either **d2(j).hold=1** [load *Dstar-latch* from *D-latch*] or **end(i)=1**, then syncs to the **Ø1** clock
	Qd-multiplex	nor2/mux	**m(i+1)bar** converts [D,0,...,0] to [Dbar,0,...,0], then **Qdhat=**[Dbar,0,...,0] or **Qdhat=**[Dstar,0,...,0] is selected according to **d1(j).latch**
	Qd-low	nor3/nor2	ORing logic using **force(i-1)/force(i)** to force the low-order 2 bits of **Qd** high, producing either **Qd=**[Dbar,1,1j+1,0,0,...,0] or **Qd=**[Dstar,1,1j+1,0,0,...,0], followed by kill logic which forces **Qd=0** for **d0(j).latch=0**
	Qz-hold	rsff	holds over **z(j)** value produced by sum-of-squares unit for 1/2 cycle

ARITHMETIC SECTION	*R-hold.ps*	rsff	holds over the partial-sum component of **R[j-1]** during the 4-2 reduction forming **R[j]**
	csa-level1	fad	performs 3-2 reduction of **Qz**, 2*{**Qd**}, and *R-hold.ps*
	R-hold.sc	rsff	holds over the saved-carry component of **R[j-1]** during the 4-2 reduction forming **R[j]**
	csa-level2	fad	performs 3-2 reduction of the outputs of csa-level1 plus *R-hold.sc*
	R-latch.ps	rsff	contains the partial-sum component of **R[j]**
	R-latch.sc	rsff	contains the saved-carry component of **R[j]**
SELECTION SECTION	*3-2 reduce*	fad/ inv.fad	performs 3-2 reduction of *R-latch.ps*, *R-latch.sc*, **z(4+j)*2^(-5)**
	cpa	fad	performs 2-1 reduction of outputs of *3-2 reduce*
	d-latch	rsff/ logic	decodes **d(j)** as **d2(j).latch/d1(j).latch/d0(j).latch**, then stores the result
	d-hold	rsff	holds over the **d(j)** value produced above for 1/2 cycle.

• Converter Action Table

Cases:	d(j)	d2(j)	d1(j)	d0(j)	Parallel Load	Appended Bit	
	d(j) = 1 = 1	0	1		**Dstar[j](0..j-1) <- D[j](0..j-1)**	**D[j](j) <- 1**	**Dstar[j](j) <- 0**
	d(j) = 0 = 0	0	0		no parallel load	**D[j](j) <- 0**	**Dstar[j](j) <- 1**
	d(j) = -1 = 0	1	1		**D[j](0..j-1) <- Dstar[j](0..j-1)**	**D[j](j) <- 1**	**Dstar[j](j) <- 0**

• Decoding Logic - d(j) selection

Range: Range {**Qsel = Rhatstar[j]**} = [-2.5,3.5], therefore 3 integer bits needed for selection

Precision: to 1/4's, therefore 2 fractional bits needed for selection

Qsel Selection: Table

					Qsel	**d(j)**		**d2(j)**	**d1(j)**	**d0(j)**
0	1	1	1	0						
.	.	.	.	.						
0	0	0	0	1	≥ 0	1	=	1	0	1
0	0	0	0	0						
1	1	1	1	1	= -1/4	0	=	0	0	0
1	1	1	1	0						
1	1	1	0	1	≤ -1/2	-1	=	0	1	1
.	.	.	.	.						
1	0	1	1	0						

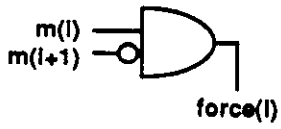
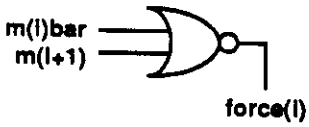
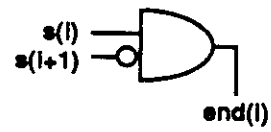
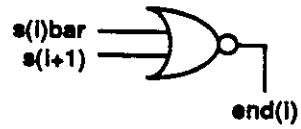
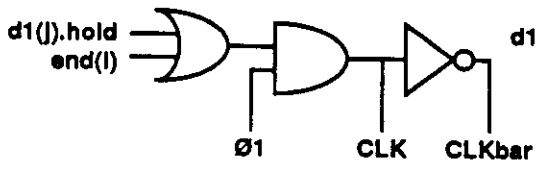
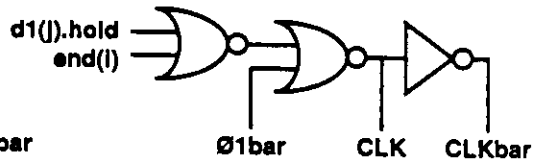
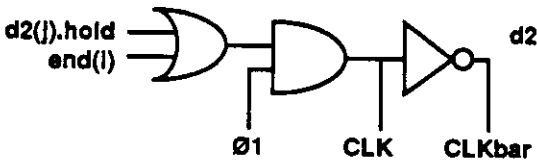
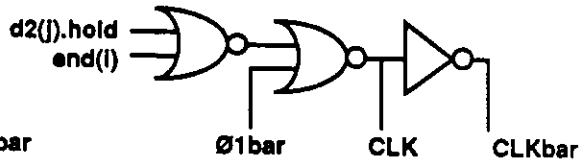
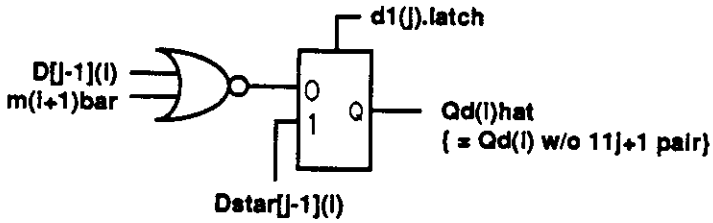
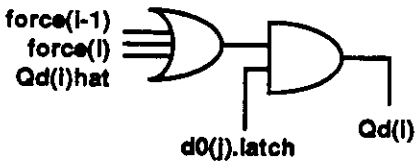
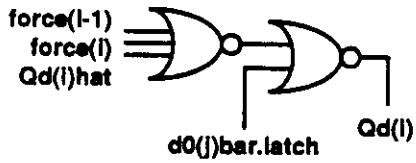
Logic: Expressions

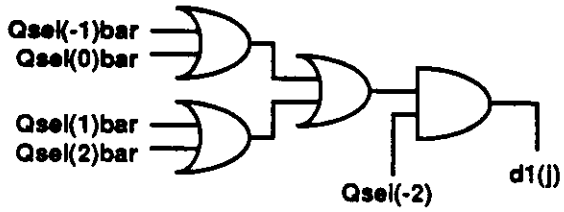
d2(j) = Qsel(-2)bar

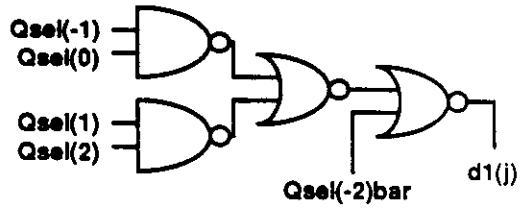
d1(j) = Qsel(-2) * {Qsel(-1)bar + Qsel(0)bar + Qsel(1)bar + Qsel(2)bar}

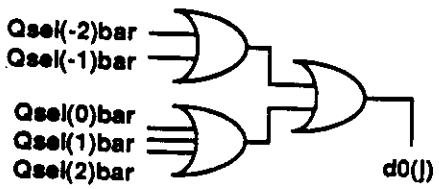
d0(j) = Qsel(-2)bar + Qsel(-1)bar + Qsel(0)bar + Qsel(1)bar + Qsel(2)bar

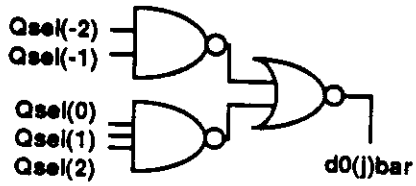
• Logic Block Reductions

Name	Function	Implementation
force-bit		
end-bit		
D-clock		
Dstar-clock		
Qd-multiplex		
Qd-low		
d-latch	**d2(j) = Qsel(-2)bar**	









• Operation

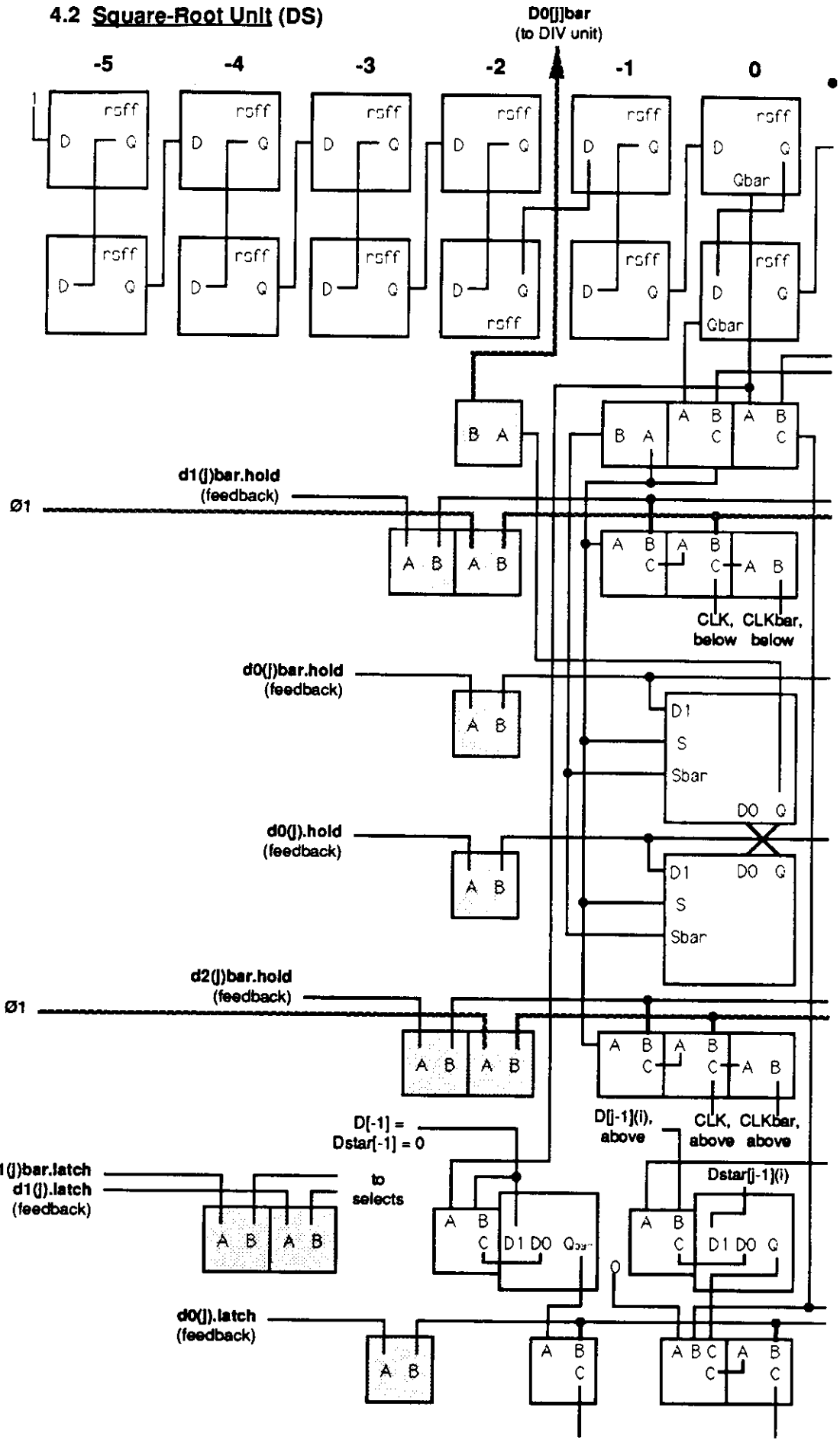
At the beginning of the chip's operation, the **GRESET** signal is used to preset all latches to 0. It is proper then to examine the operation of each section separately.

In the operand section, a bit train travels across the shift register such that position j can be determined while leaving a marker on those positions already crossed. In *force-bit*, **m(i)bar/m(i+1)** are NORed {producing **force(i)**} to determine the position of the end of the train, and thus of the low-order 2 bits of **Qd**, in order to force them to 1. In *end-bit*, **s(i)bar/s(i+1)** are NORed {producing **end(i)**}, again to determine the position of the end of the train, and thus of where appending, rather than loading, should occur in *D-latch* and *Dstar-latch*. Here, the inputs to *D-latch*/*Dstar-latch* are re-directed via their select lines, from the parallel load to the append input, and a clock is forced for the append operation. The clock logic for both *D-latch*/*Dstar-latch* simply ORs the proper bit of **d(j)** {**d1(j).hold** for *D-latch*, **d2(j).hold** for *Dstar-latch*} with the **end(i)** signal, and syncs this with the **Ø1** clock. Next, in *Qd-multiplex*, we form **Qdhat** by NORing **D[j-1](i)** with **m(i+1)bar** (hence the need for a bit train), and select either **Qdhat=**[Dbar,0,...,0] or **Qdhat=**[Dstar,0,...,0] according to **d1(j).latch**. Finally, in *Qd-low*, we use the **force(i)** signal produced in every slice to force the low-order 2 bits of **Qdhat** high, producing either **Qd=**[Dbar,1,1$_{j+1}$,0,0,...,0] or **Qd=**[Dstar,1,1$_{j+1}$,0,0,...,0], and add kill logic which forces **Qd=**0 for **d0(j).latch=**0.

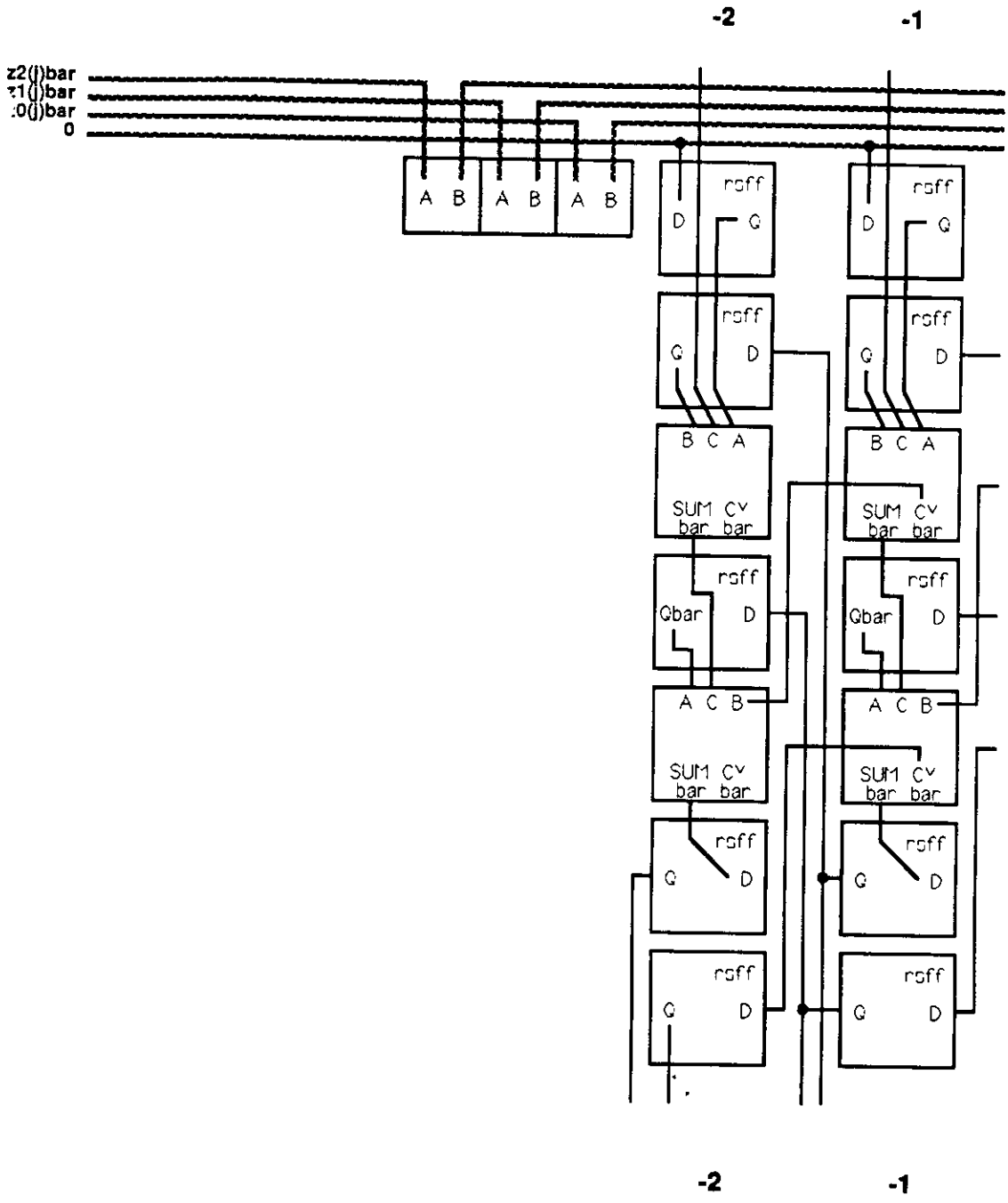
In the arithmetic section, 2*{**Qd**}, **Qz**, and the partial-sum and carry-save components of **R[j-1]** repeatedly undergo a 4-2 reduction to form **R[j]**. Since the range of **Rhatstar[j]** is [-2.5,3.5], three integer bits are kept (in carry-save form) for the **d(j)** selection process. **R[j]** (minus its largest digit) is routed back up to registers *R-hold.ps*/*R-hold.sc* after selection for computation of the next **R[j]**.

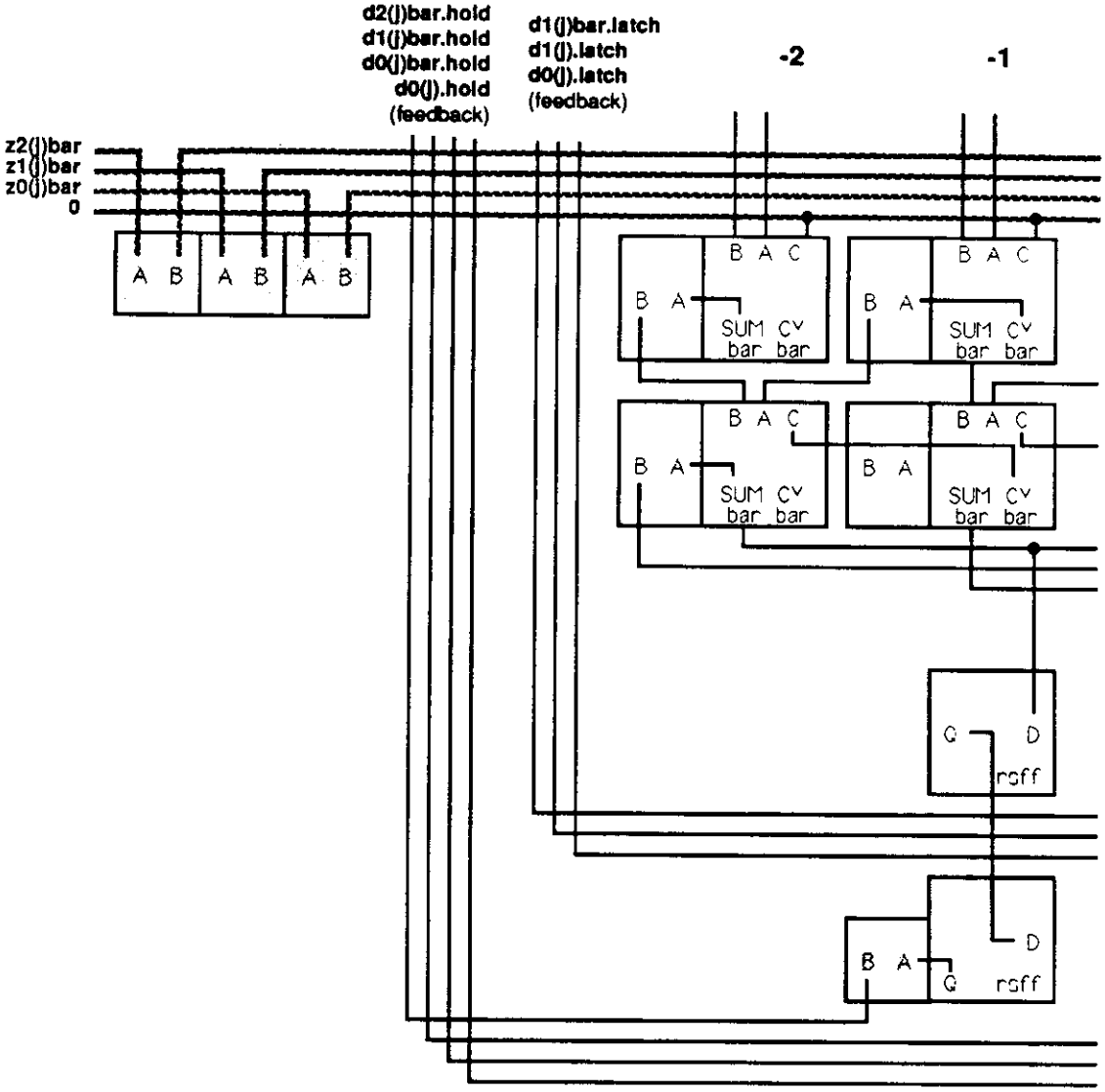
In the selection section, *3-2 reduce* performs a 3-2 reduction of *R-latch.ps*, *R-latch.sc*, and **z(4+j)*2^(-5)**, *cpa* performs 2-1 reduction of outputs of *3-2 reduce*, *d-latch* decodes **d(j)** as **d2(j)/d1(j)/d0(j)**, then stores the result, and finally *d-hold* holds over the **d(j)** value produced above for 1/2 cycle.

4.2 Square-Root Unit (DS)

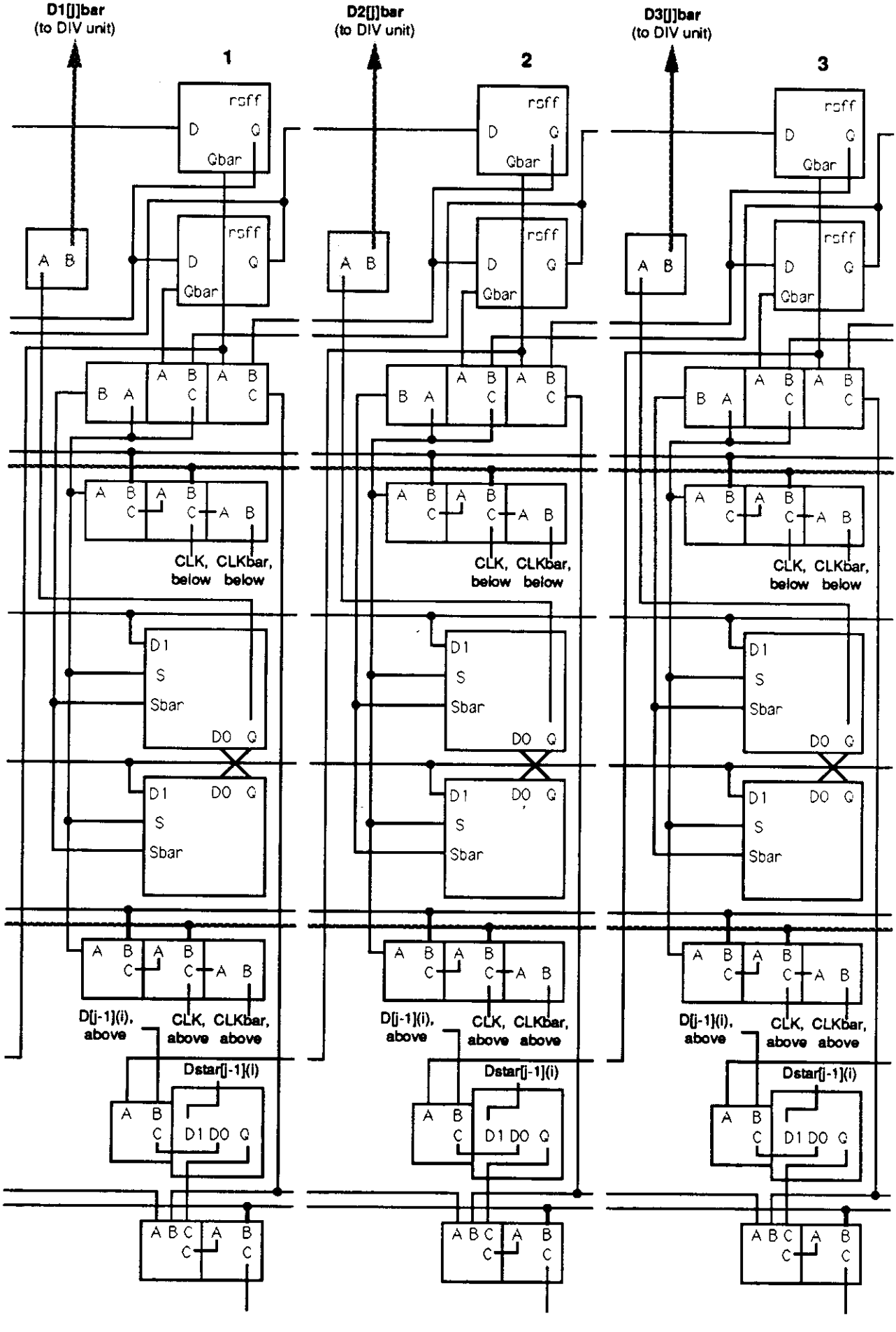


[1,1]

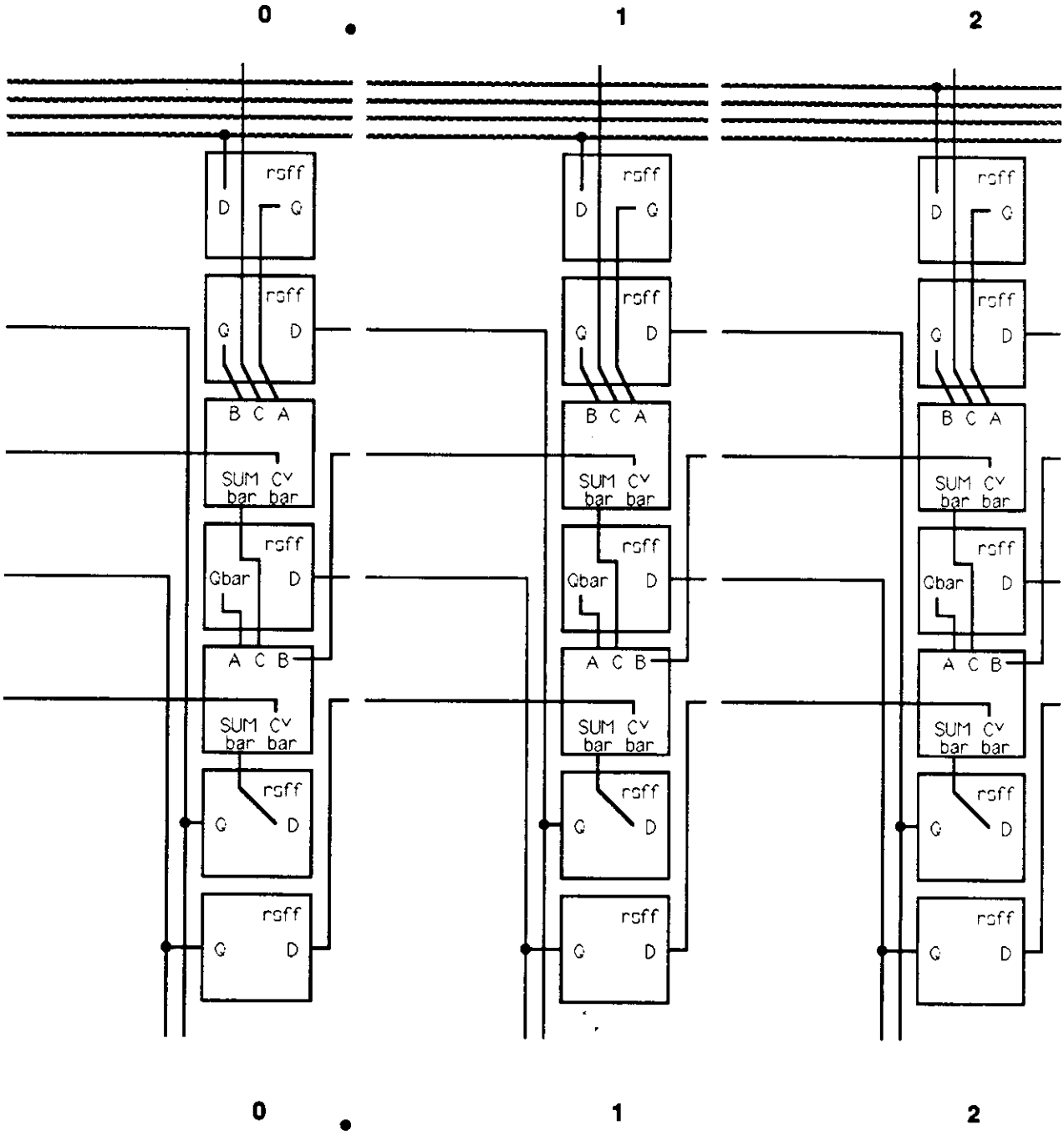

-2
-1
z2(j)bar
z1(j)bar
z0(j)bar
0
A B
A B
A B
rsff
D
Q
Q
D
B C A
SUM Cv
bar bar
Qbar
A C B
-2
-1

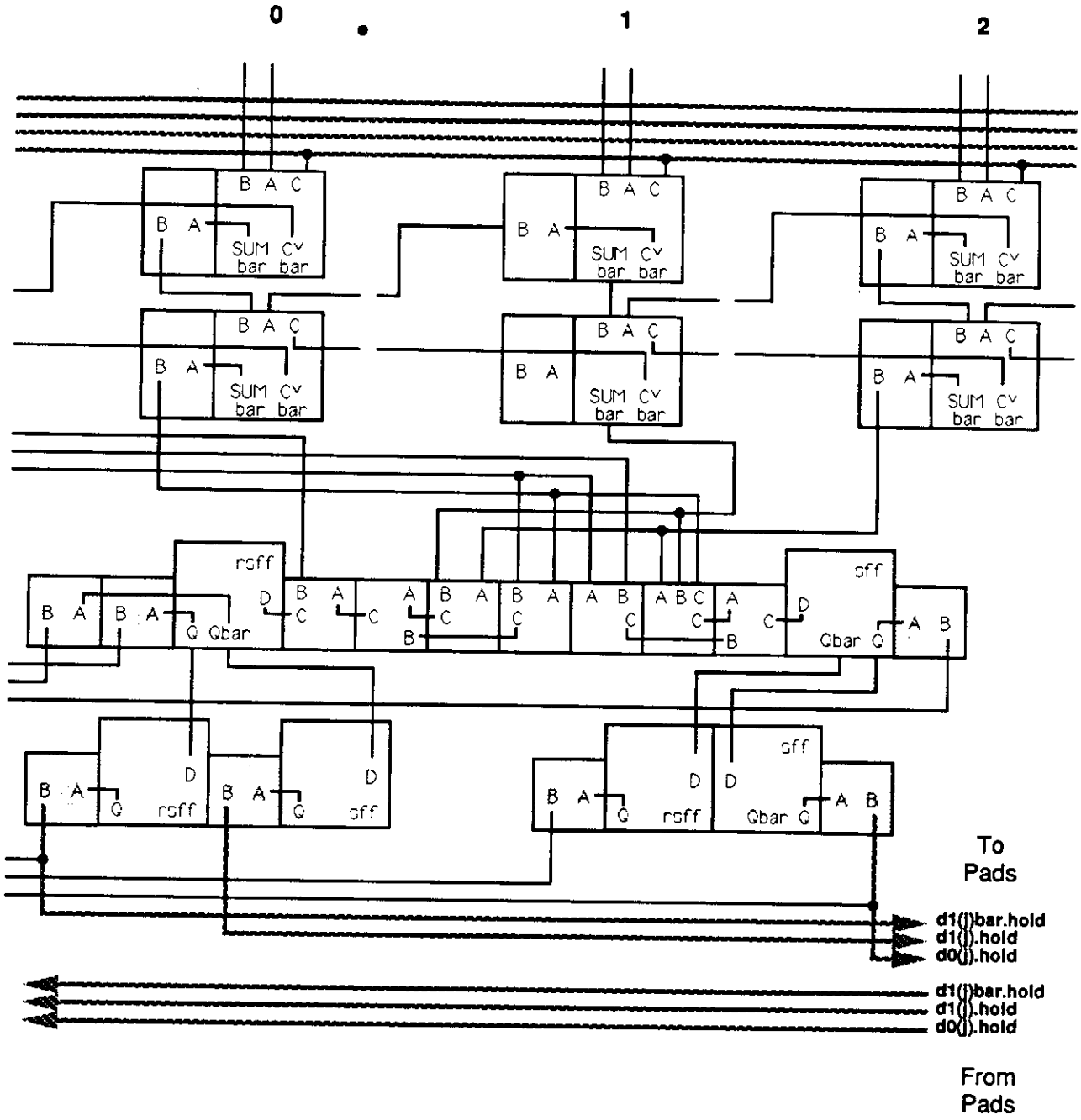

d2(j)bar.hold
d1(j)bar.hold
d0(j)bar.hold
d0(j).hold
(feedback)
d1(j)bar.latch
d1(j).latch
d0(j).latch
(feedback)
-2
-1
z2(j)bar
z1(j)bar
z0(j)bar
0
A B A B A B
B A C
B A
SUM C^v
bar bar
G D
rsff

[3,1]

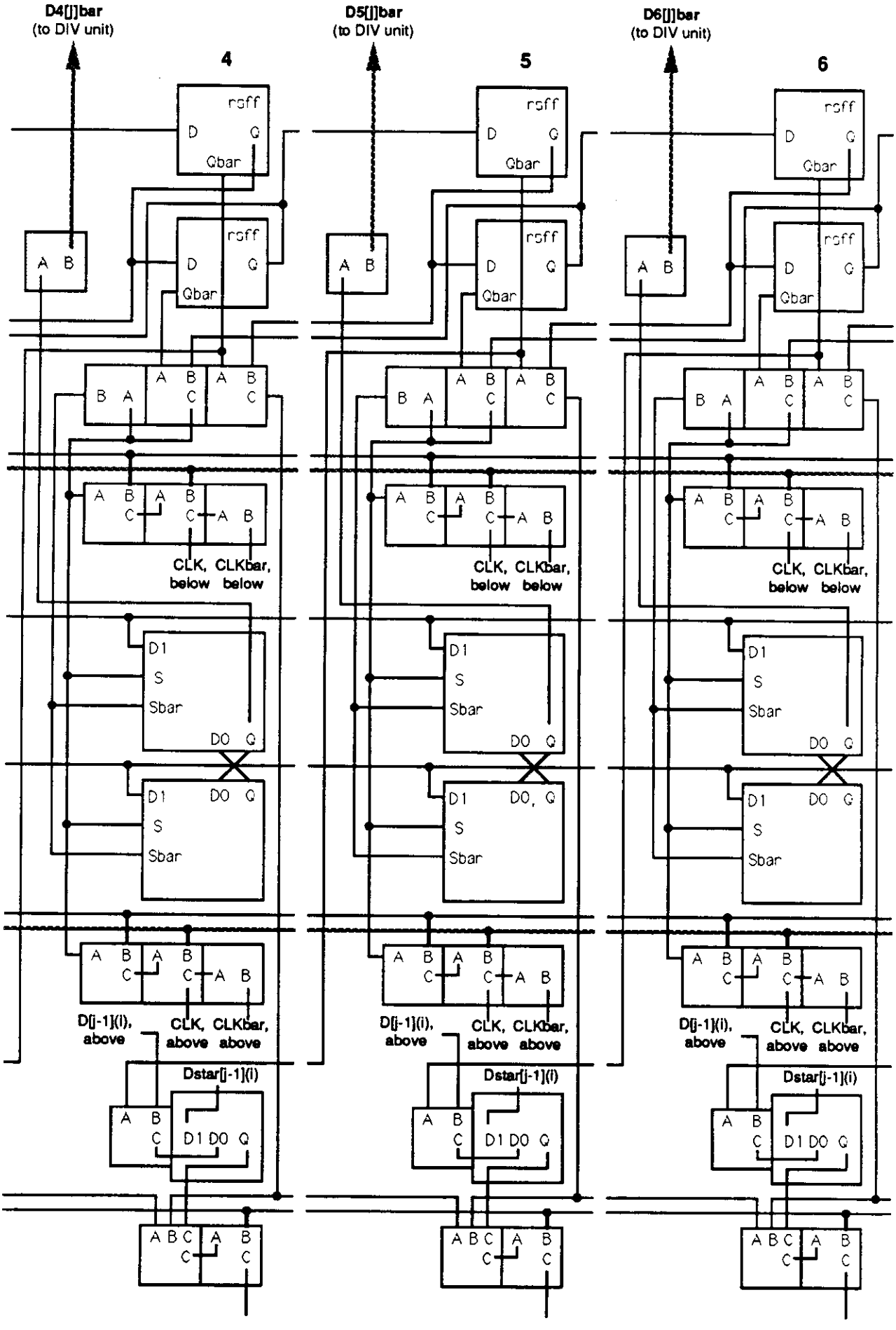

D1[j]bar
(to DIV unit)
D2[j]bar
(to DIV unit)
D3[j]bar
(to DIV unit)
1
2
3
rsff
D
Q
Gbar
A B
B A
A B C
CLK, CLKbar, below below
D1
S
Sbar
D0 Q
D[j-1](i), above
CLK, CLKbar, above above
Dstar[j-1](i)
A B C D1 D0 Q
A B C C A B C
1
2
3

[1,2]

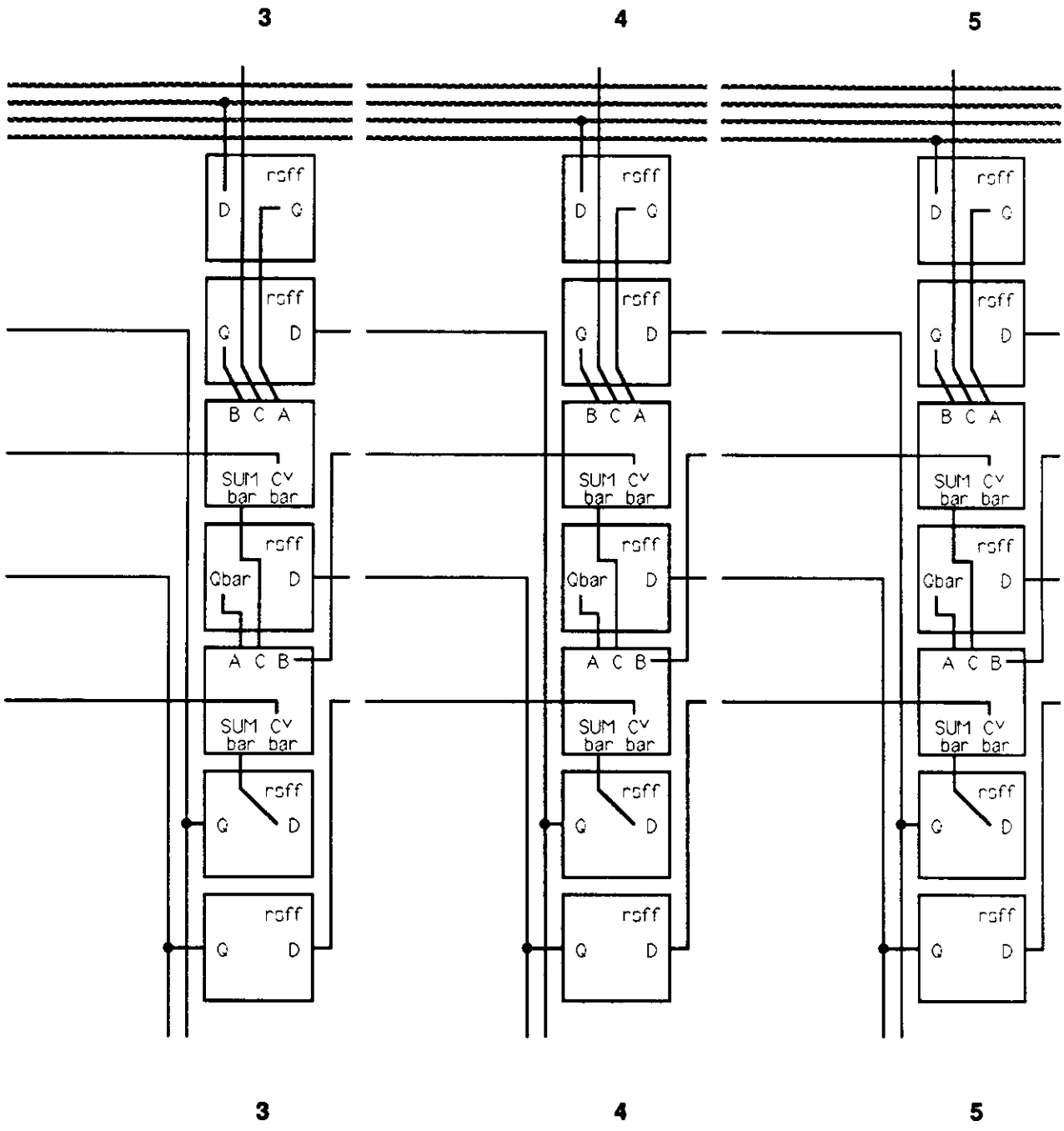

0
1
2
rsff
D
G
Q
B C A
SUM Cv
bar bar
Qbar
A C B
0
1
2

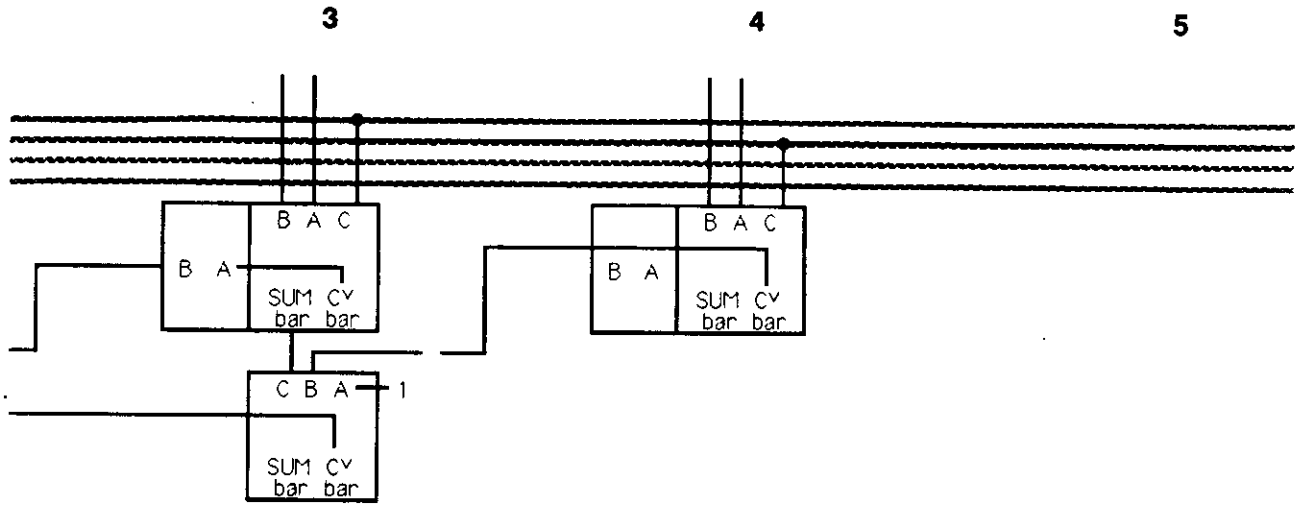

0
1
2
B A C
B A
SUM Cv
bar bar
rsff
sff
D
C
A
B
Q Qbar
Qbar Q
To
Pads
d1(j)bar.hold
d1(j).hold
d0(j).hold
From
Pads

[3,2]

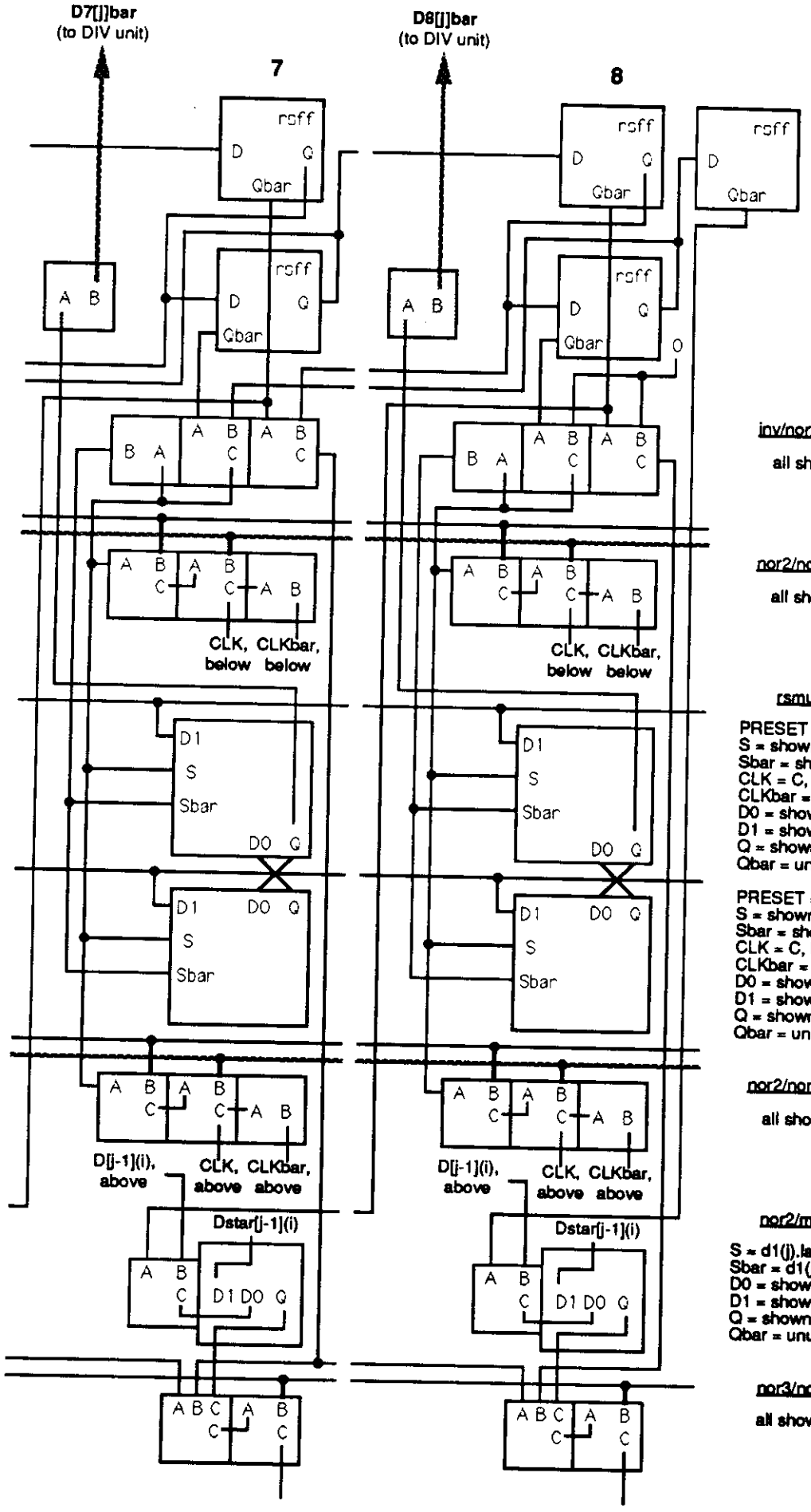

D4[j]bar
(to DIV unit)
D5[j]bar
(to DIV unit)
D6[j]bar
(to DIV unit)
4
5
6
rsff
D
Q
Qbar
A B
B A
A B C
CLK, CLKbar,
below below
D1
S
Sbar
DO Q
D1 DO Q
D[j-1](i),
above
CLK, CLKbar,
above above
Dstar[j-1](i)
A B C D1 DO Q
A B C C A B C
4
5
6

[1,3]


3
4
5
rsff
D
Q
Qbar
B C A
A C B
SUM
bar
CV
3
4
5



[3,3]



rsff

PRESET = GRESET
D = shown
CLK = Ø1
CLKbar = Ø1bar
Q = shown
Qbar = unused

(shift-register)

PRESET = GRESET
D = shown
CLK = Ø2
CLKbar = Ø2bar
Q = shown
Qbar = unused

inv/nor2/nor2

all shown

(end-bit/ force-bit)

nor2/nor2/inv

all shown

(D-clock)

OPERAND SECTION

rsmuxff

PRESET = GRESET
S = shown
Sbar = shown
CLK = C, above
CLKbar = B, above
D0 = shown
D1 = shown
Q = shown
Qbar = unused

(D-latch)

PRESET = GRESET
S = shown
Sbar = shown
CLK = C, below
CLKbar = B, below
D0 = shown
D1 = shown
Q = shown
Qbar = unused

(Dstar-latch)

nor2/nor2/inv

all shown

(Dstar-clock)

nor2/mux

S = d1(j).latch (fb)
Sbar = d1(j)bar.latch (fb)
D0 = shown
D1 = shown
Q = shown
Qbar = unused

(Qd-multiplex)

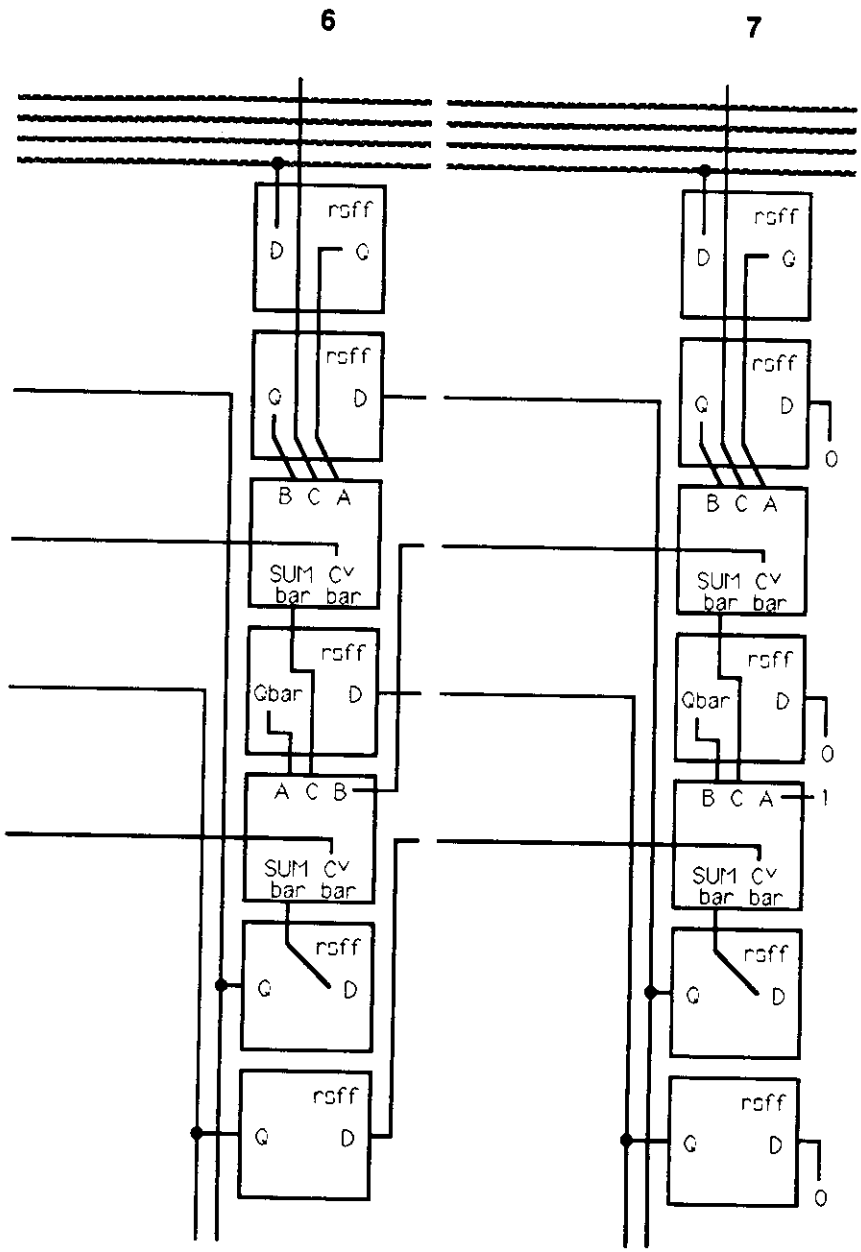
nor3/nor2

all shown

(Qd-low)

7 8

[1,4]



rsff

PRESET = GRESET
D = shown
CLK = Ø1
CLKbar = Ø1bar
Q = shown
Qbar = unused

(Qz-hold)

rsff

PRESET = GRESET
D = shown
CLK = Ø1
CLKbar = Ø1bar
Q = shown
Qbar = unused

(R-hold.ps)

fad
all shown

(csa-level1)

rsff

PRESET = GRESET
D = shown
CLK = Ø1
CLKbar = Ø1bar
Q = unused
Qbar = shown

(R-hold.sc)

fad
all shown

(csa-level2)

rsff

PRESET = GRESET
D = shown
CLK = Ø2
CLKbar = Ø2bar
Q = shown
Qbar = unused

(R-latch.ps)

PRESET = GRESET
D = shown
CLK = Ø2
CLKbar = Ø2bar
Q = shown
Qbar = unused

(R-latch.sc)

ARITHMETIC SECTION

inv.fad/fad

all connections shown *(3-2 reduce)*

fad

all connections shown *(cpa)*

rsff/logic

PRESET = GRESET
D = shown
CLK = Ø1
CLKbar = Ø1bar
Q = shown
Qbar = shown
logic = all shown *(d-latch)*

PRESET = GRESET
D = shown
CLK = Ø2
CLKbar = Ø2bar
Q = shown
Qbar = unused *(d-hold)*

SELECTION SECTION

5.1 Division Unit (SS)

• Recurrence Operands

W[j] = 2*P[j-1]
+ {-d(j)*S[j-1]}

previous value of **P[j]** held in *P-latch.ps / P-latch.sc*

P[j] = W[j]
+ {-s(j)*D[j]}

{Qs} value produced directly above the arithmetic section; no shifting of the resulting **Qs** is required

W[j] value held in *W-latch.ps / W-latch.sc*

{Qd} value produced in arithmetic section directly above *P-latch.ps/P-latch.sc*

Bit Patterns:

Qs = [Sbar , 1 , ... , 1 , ... , 1]
Qs = [0 , 0 , ... , 0 , ... , 0]
Qs = [S , 0 , ... , 0 , ... , 0]

d(j) = 1 = 1 | 0 1 (1 inserted at far right of *csa-level1*)
d(j) = 0 = 0 | 0 0
d(j) = -1 = 0 | 1 1

d0(j)
d1(j)
d2(j) [non-binary, spec. defined]

Qd = [Dbar , 1 , ... , 1 , ... , 1]
Qd = [0 , 0 , ... , 0 , ... , 0]
Qd = [D , 0 , ... , 0 , ... , 0]

s(j) = 1 = 1 | 0 1 (1 inserted at far right of *csa-level2*)
s(j) = 0 = 0 | 0 0
s(j) = -1 = 0 | 1 1

s0(j)
s1(j)
s2(j) [non-binary, spec. defined]

• Registers/Levels

	Name	Type	Function
OPERAND SECTION	*shift-register*	rsff [sff in m(-6)]	Traveling bit pair: **m(i)** is used to set a flip-flop in slice #1 (controlling clocking of *P-latch.ps/P-latch.sc*; **s(i)** is used for two purposes: - re-directing the inputs to *S-latch/Sstar-latch* in position j, via their select lines, from the parallel load to the append input - always enabling the clocks to *S-latch/Sstar-latch* in position j
	S-clock	nor2/nor2 inv	produces clock enable if either **s1(j)=1** [load *S-latch* from *Sstar-latch*] or **s(i)=1**, then syncs to the **Ø2** clock
	S-latch	rsmuxff	stores the on-the-fly converted value of **S[j]**
	Sstar-latch	rsmuxff	stores the on-the-fly converted value of **Sstar[j]**
	Sstar-clock	nor2/nor2 inv	produces clock enable if either **s2(j)=1** [load *Sstar-latch* from *S-latch*] or **s(i)=1**, then syncs to the **Ø2** clock
	Qs-multiplex	inv/mux nor2	**Qs=**[Sbar,1,...,1] or **Qs=**[S,0,...,0] is selected according to **d1(j).hold**, followed by kill logic which forces **Qs=0** for **d0(j).hold=0**
ARITHMETIC SECTION	*csa-level1*	fad	performs 3-2 reduction of *P-latch.ps*, *P-latch.sc*, and **Qs**
	W-latch.ps	rsff	contains the partial-sum component of **W[j]** for the 3-2 reduction forming P[j]
	W-latch.sc	rsff	contains the saved-carry component of **W[j]** for the 3-2 reduction forming **P[j]**
	Qd-multiplex	inv/mux nand2	**Qd=**[Dbar,1,...,1] or **Qd=**[D,0,...,0] is selected according to **s1(j)**, followed by kill logic which forces **Qd=0** for **s0(j)=0**
	csa-level2	fad	performs 3-2 reduction of *W-latch.ps*, *W-latch.sc*, and **Qd**

	P-enable	nor2/inv/rsff	produces clock enable for the initial load of *P-latch.ps* (**ENABLE** synced to **Ø1**); thereafter produces the clock enable for the entire *P-latch* (synced to Ø2) beginning at the appropriate cycle of the unit's operation
	P-clock	nand2/nand2 nand2/inv	
	P-latch.ps	muxff	initially loads **Qh=h*2^(-3)**; thereafter latches the partial-sum component of **P[j]**
	P-latch.sc	rsff	contains the saved-carry component of **P[j]**
SELECTION SECTION			
	3-2 reduce	fad/ inv.fad	performs 3-2 reduction of *P-latch.ps*, *P-latch.sc*, & **Qsign=-(1-sign(d(j)))*2^(-3)**
	cpa	fad	performs 2-1 reduction of outputs of *3-2 reduce*
	s-latch	rsff/ logic	decodes **s(j)** as **s2(j)/s1(j)/s0(j)**, then stores the result

• Converter Action Table

Cases:	s(j)	s2(j)	s1(j)	s0(j)	Parallel Load	Appended Bit	
	1	1	0	1	Sstar[j](0..j-1) <- S[j](0..j-1)	S[j](j) <- 1	Sstar[j](j) <- 0
	0	0	0	0	no parallel load	S[j](j) <- 0	Sstar[j](j) <- 1
	-1	0	1	1	S[j](0..j-1) <- Sstar[j](0..j-1)	S[j](j) <- 1	Sstar[j](j) <- 0

Decoding Logic - s(j) selection

Range: Range {**Qsel = What[j]**} = [-23/8,+23/8), therefore 3 integer bits needed for selection

Precision: to 1/8's, therefore 3 fractional bits needed for selection

Qsel Selection: Table

						Qsel	**s(j)**		**s2(j)**	**s1(j)**	**s0(j)**
0	1	0	1	1	1						
.	.	.	.	.	.						
0	0	0	0	1	1						
0	0	0	0	1	0	≥ 1/4	1	=	1	0	1
0	0	0	0	0	1						
0	0	0	0	0	0						
1	1	1	1	1	1	[-1/4,1/8]	0	=	0	0	0
1	1	1	1	1	0						
1	1	1	1	0	1						
1	1	1	1	0	0	≤ -3/8	-1	=	0	1	1
.	.	.	.	.	.						
1	0	1	0	0	1						

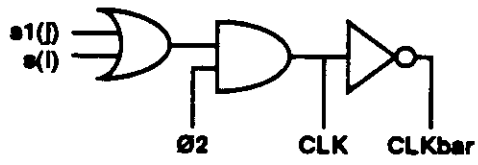
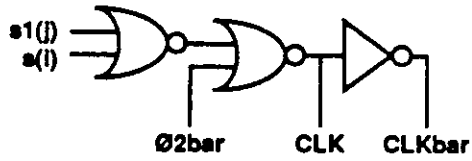
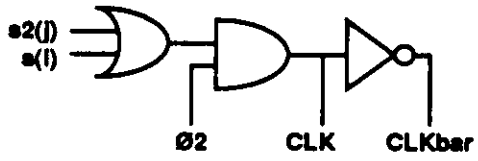
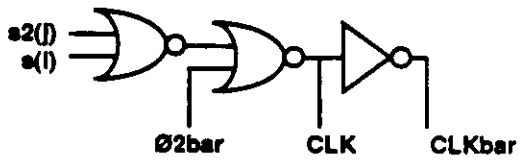
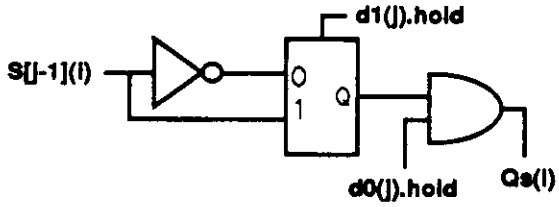
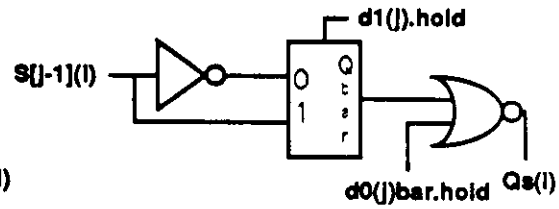
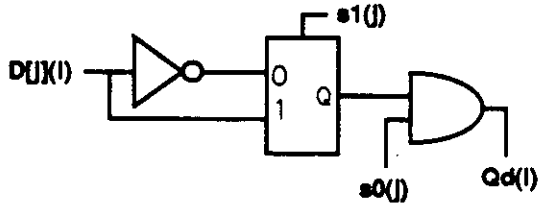
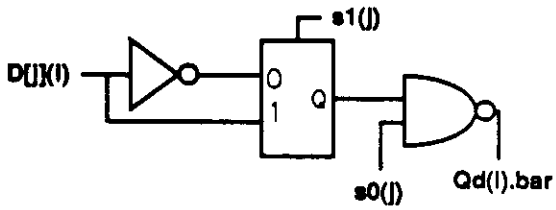
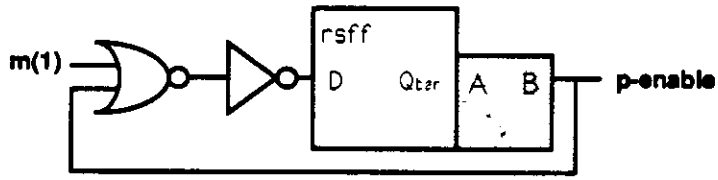
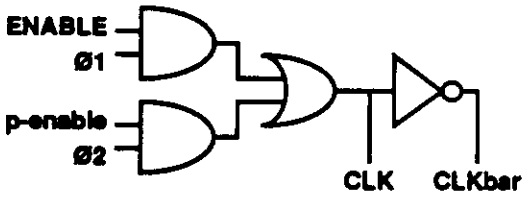
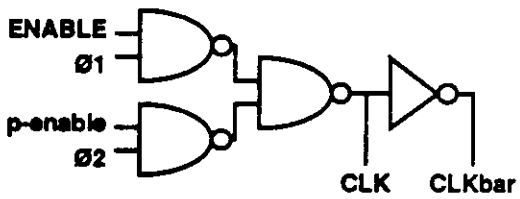
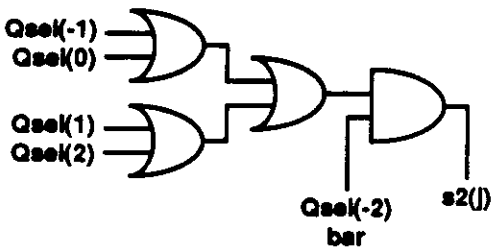
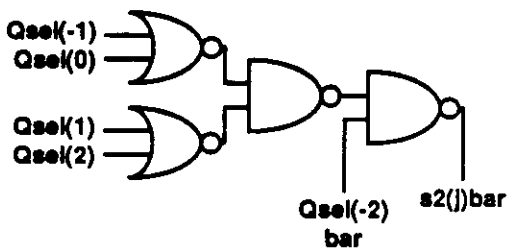
Logic: Expressions

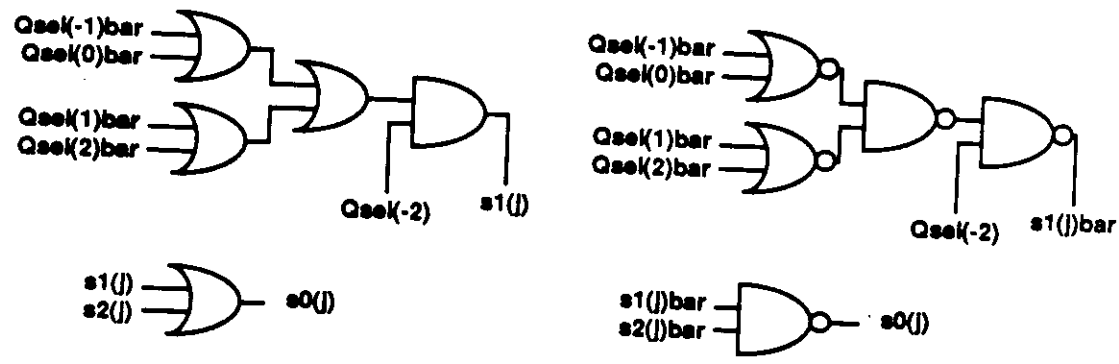
s2(j) = Qsel(-2)bar * bar{Qsel(-1)bar * Qsel(0)bar * Qsel(1)bar * Qsel(2)bar}
= Qsel(-2)bar * {Qsel(-1) + Qsel(0) + Qsel(1) + Qsel(2)}

s1(j) = Qsel(-2) * bar{Qsel(-1) * Qsel(0) * Qsel(1) * Qsel(2)}
= Qsel(-2) * {Qsel(-1)bar + Qsel(0)bar + Qsel(1)bar + Qsel(2)bar}

s0(j) = s1(j) + s2(j) = bar {s1(j)bar * s2(j)bar}

• Logic Block Reductions

Name	Function	Implementation
S-clock		
Sstar-clock		
Qs-multiplex		
Qd-multiplex		
P-enable		
P-clock		
s-latch		



• Operation

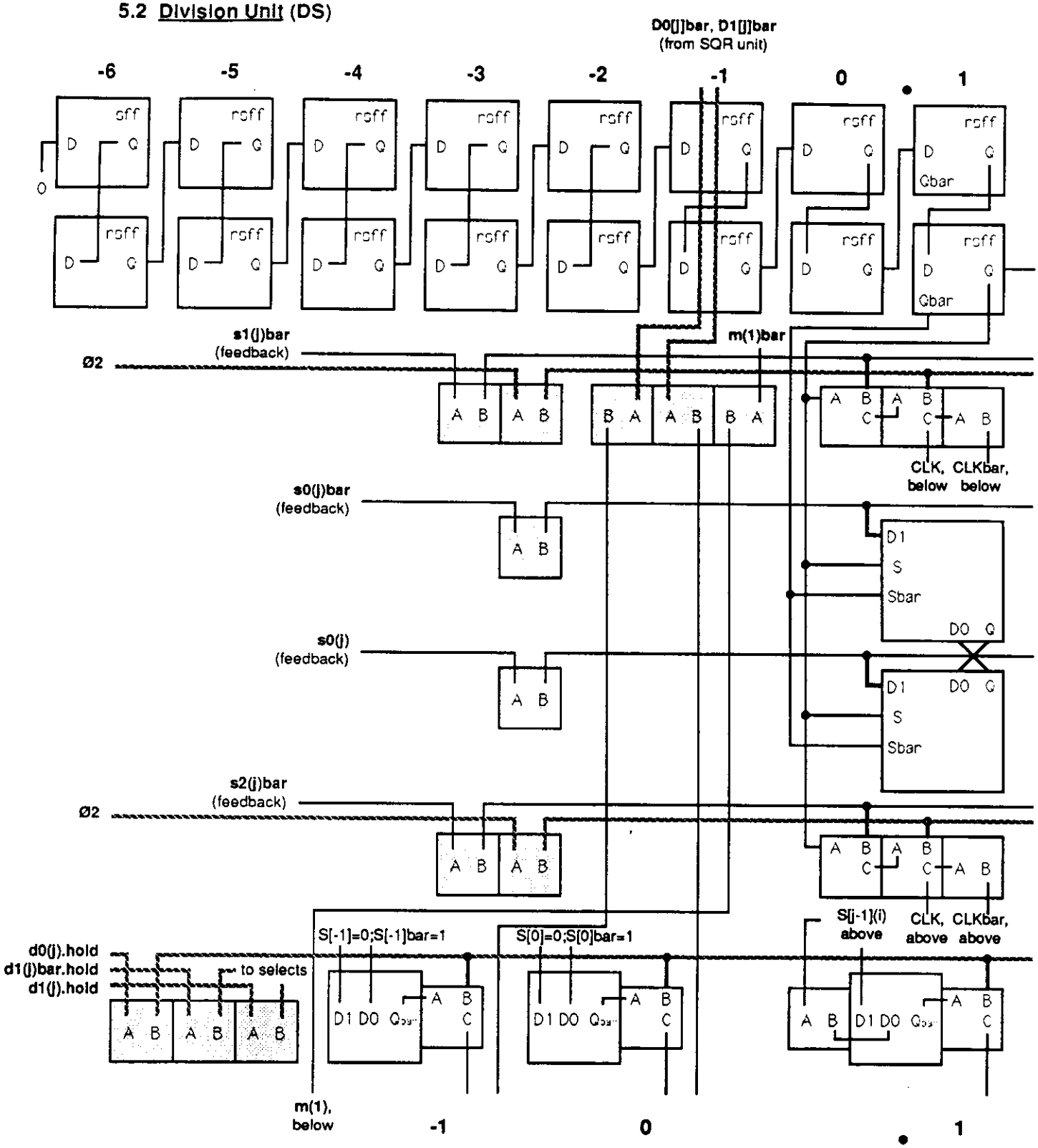
At the beginning of the chip's operation, the **GRESET** signal is used to preset all latches to 0 [except for **m(0)** of *shift-register*, which is set to 1]. It is proper then to examine the operation of each section separately.

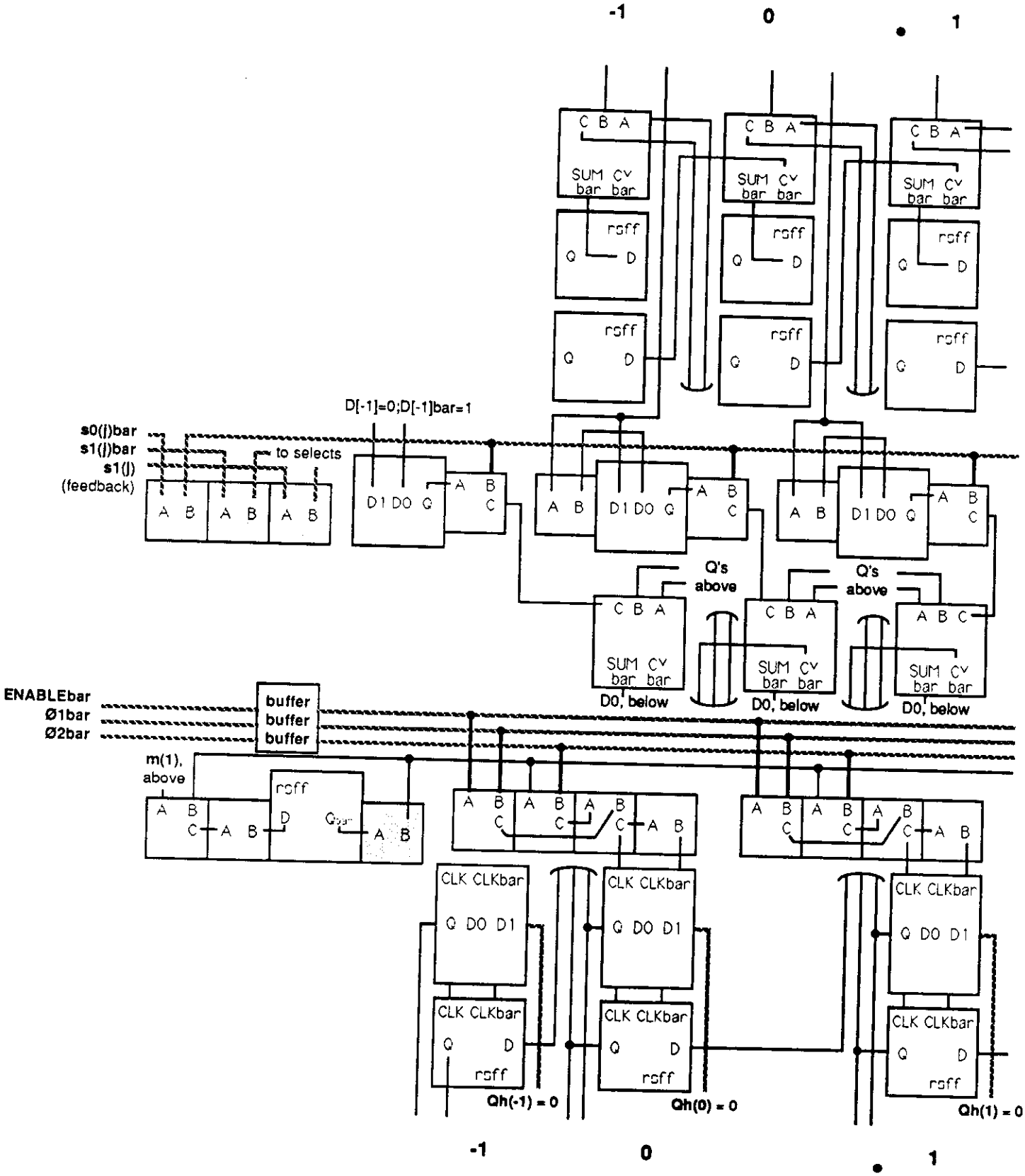
In the operand section, a bit pair travels across the shift register such that position j can be determined. **m(1)** is the only master output used in the unit - its function is to set a flip-flop in *P-enable* (which thereafter latches on its own output) to enable clocking to the latches of *P-latch*. **s(I)** is used to mark the position where appending, rather than parallel loading, should occur in *S-latch* and *Sstar-latch*. Their inputs are re-directed via their select lines, from the parallel load to the append input, and a clock is forced for the append operation. The clock logic for both *S-latch*/*Sstar-latch* simply ORs the proper bit of **s(j)** {**s1(j)** for *S-latch*, **s2(j)** for *Sstar-latch*} with **s(i)**, and syncs this with the **Ø2** clock. Next, in *Qs-multiplex*, we form **Qs** by selecting either **Qs=**[Sbar,1,...,1] or **Qs=**[S,0,...,0] using **d1(j).hold**, and finish with kill logic which forces **Qs=**0 for **d0(j).hold=**0.

In the arithmetic section, **Qs** and the partial-sum and carry-save components of **P[j-1]** undergo a 3-2 reduction to form **W[j]**. Since **s(j)** is produced at the same time, it is available for use in *Qd-multiplex* to form **Qd**. This is done by selecting either **Qd=**[Dbar,1,...,1] or **Qd=**[D,0,...,0] using **s1(j)**, following with kill logic which forces **Qd=**0 for **s0(j)=**0. Finally, **Qd** and the partial-sum and carry-save components of **W[j]** undergo a 3-2 reduction to form **P[j]**.

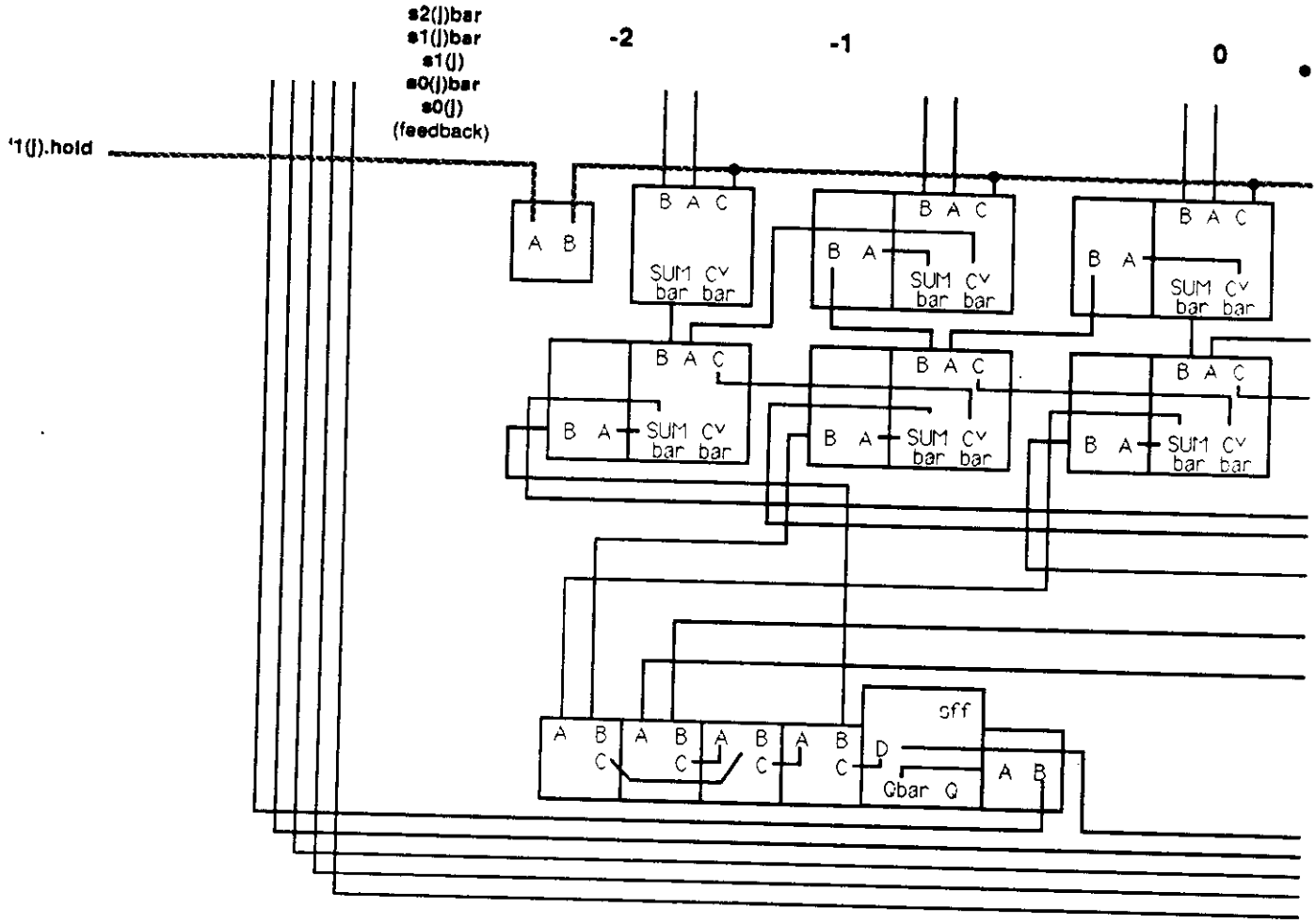
In the selection section, *3-2 reduce* performs a 3-2 reduction of *P-latch.ps*, *P-latch.sc*, and **Qsign=-{1-sign(d(j))}*2^(-3)**, *cpa* performs 2-1 reduction of outputs of *3-2 reduce*, and *s-latch* decodes **s(j)** as **s2(j)**/**s1(j)**/**s0(j)**, storing the result.

5.2 Division Unit (DS)

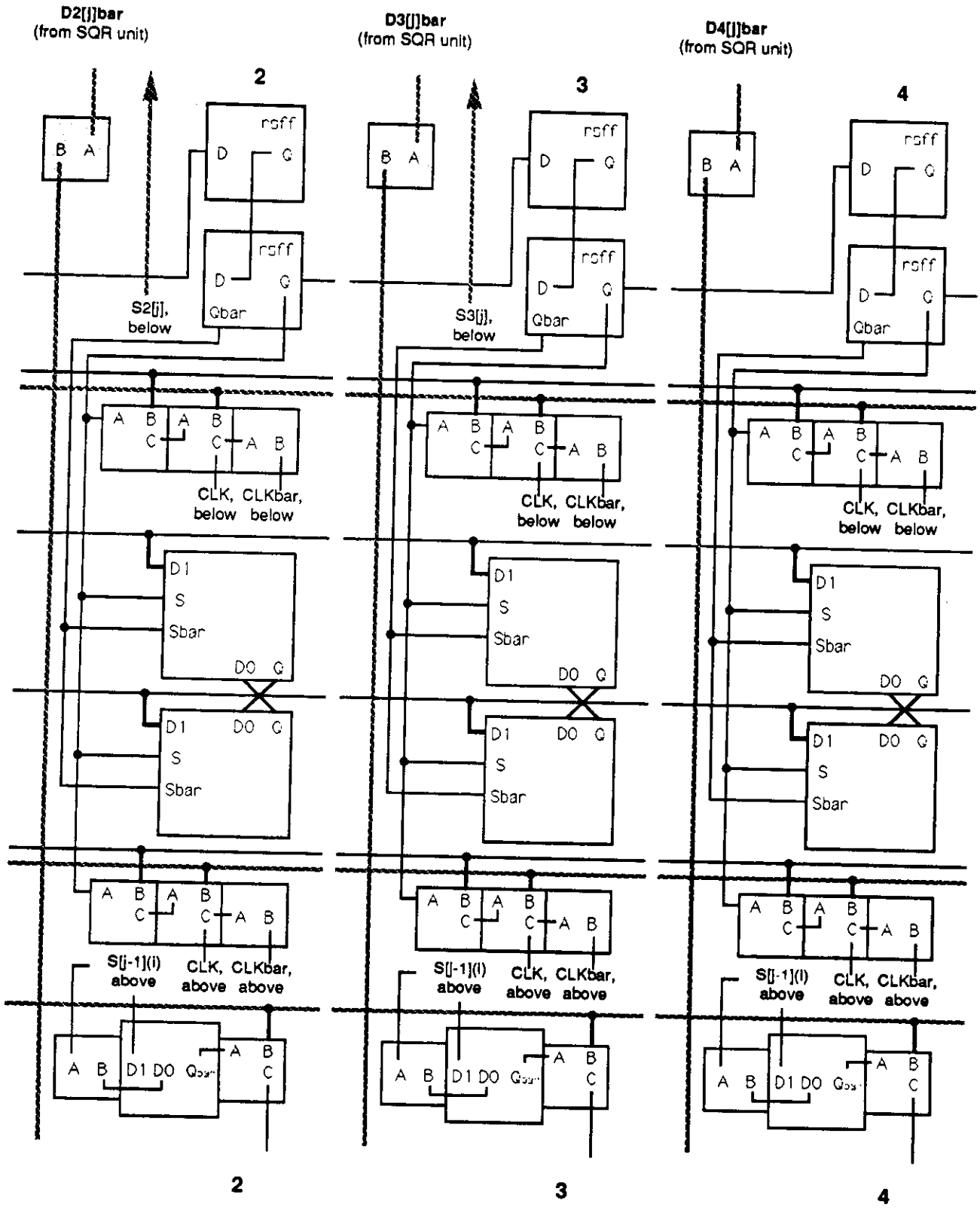



-1
0
1
C B A
SUM Cv
bar bar
rsff
Q D
s0(j)bar
s1(j)bar
s1(j)
(feedback)
to selects
D[-1]=0;D[-1]bar=1
A B
D1 DO Q
A B C
Q's
above
D0, below
ENABLEbar
Ø1bar
Ø2bar
buffer
buffer
buffer
m(1),
above
CLK CLKbar
Q DO D1
Qh(-1) = 0
Qh(0) = 0
Qh(1) = 0
-1
0
1

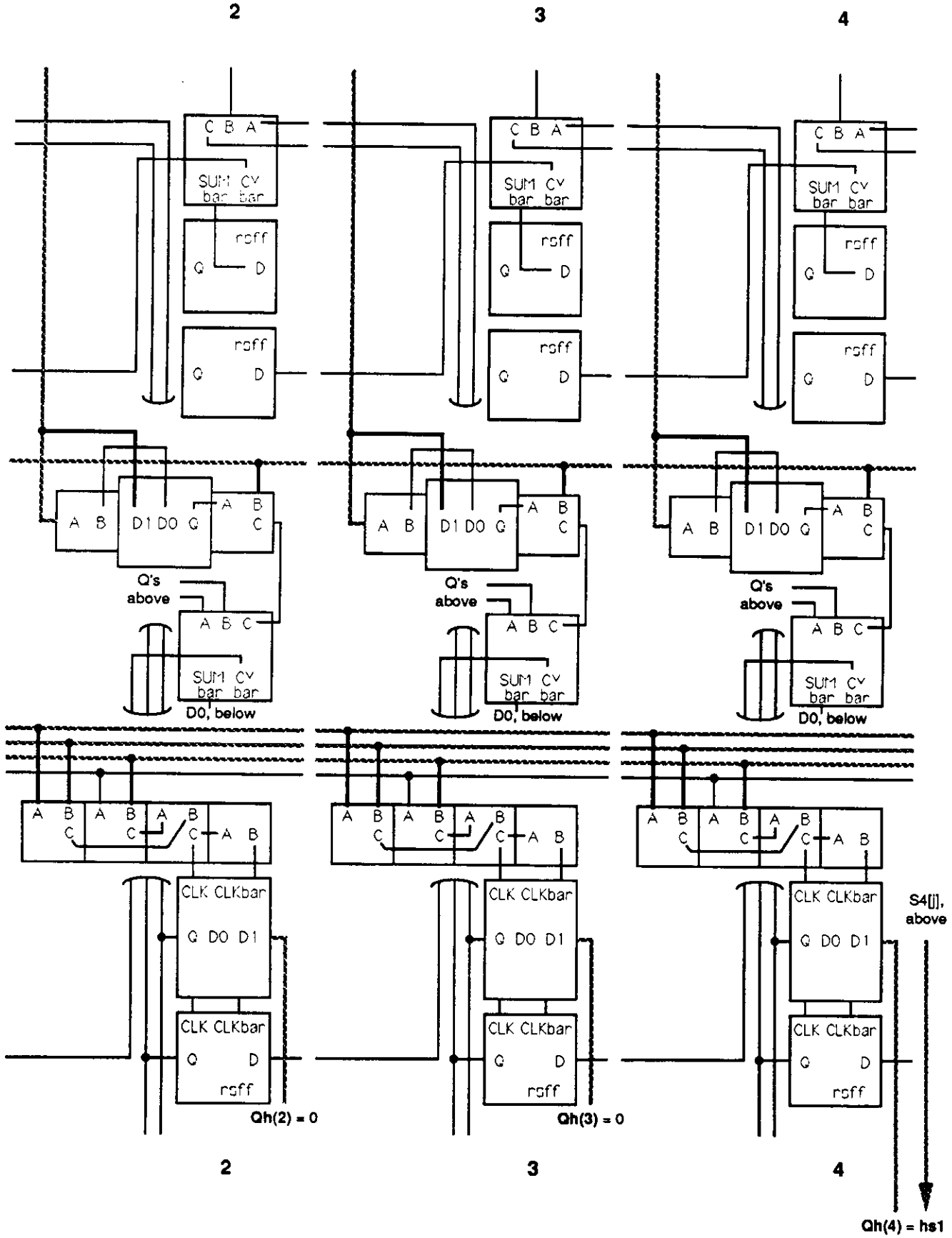
[2,1]


s2(j)bar
s1(j)bar
s1(j)
s0(j)bar
s0(j)
(feedback)
'1(j).hold
-2
-1
0
A B
B A C
SUM Cv
bar bar
B A
off
D
Qbar Q

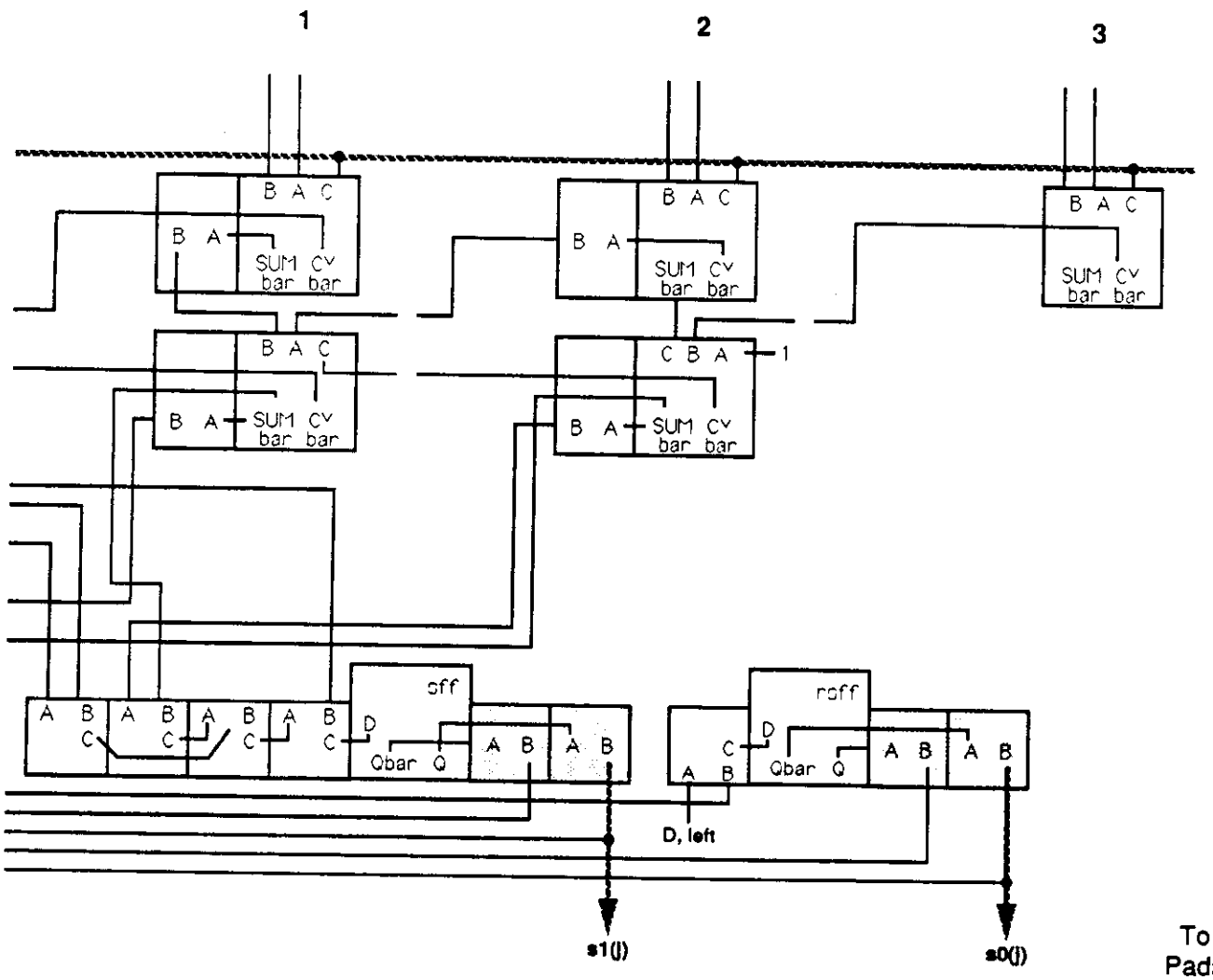
[3,1]


D2[j]bar
(from SQR unit)
D3[j]bar
(from SQR unit)
D4[j]bar
(from SQR unit)
2
3
4
rsff
D
Q
Qbar
S2[j],
below
S3[j],
below
A B
C
CLK, CLKbar,
below below
D1
S
Sbar
DO Q
S[j-1](i)
above
CLK, CLKbar,
above above
D1 DO Qbar
2
3
4

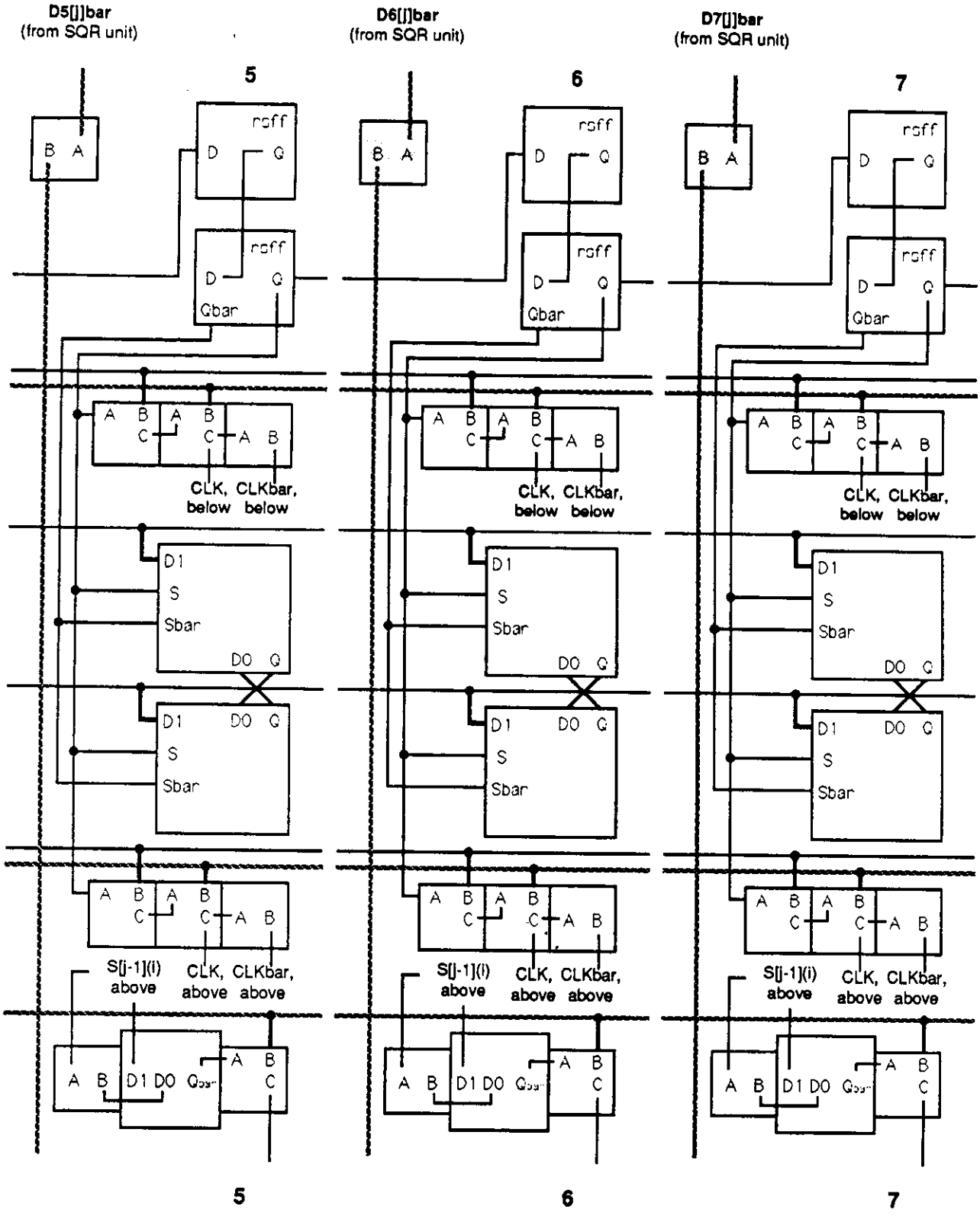
[1,2]


2
3
4
C B A
SUM CY
bar bar
rsff
Q D
A B
D1 DO Q
A B C
Q's above
D0, below
CLK CLKbar
Q DO D1
S4[j], above
Qh(2) = 0
Qh(3) = 0
Qh(4) = hs1

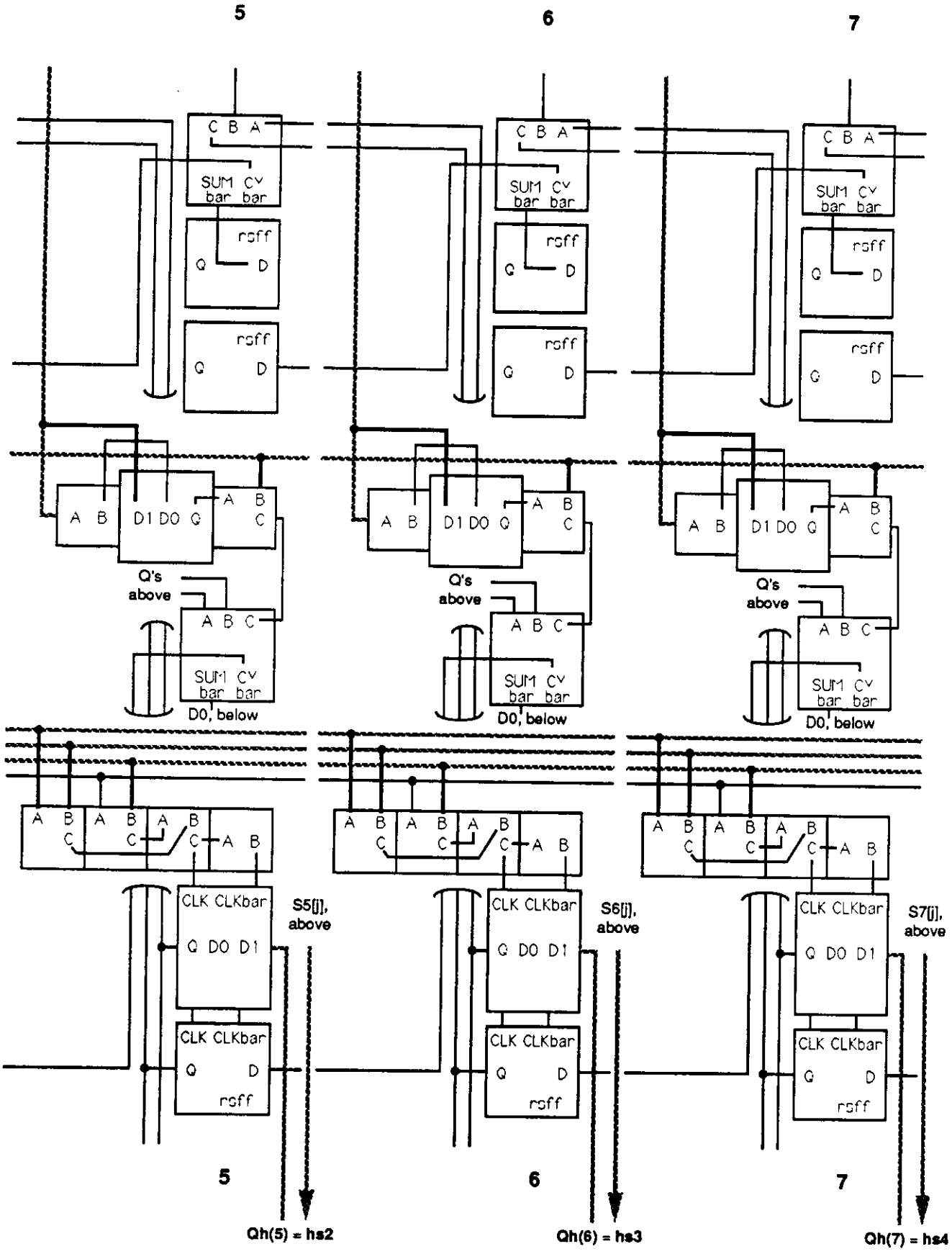
[2,2]


1
2
3
B A C
B A
SUM Cv
bar bar
C B A
1
sff
rsff
D
Qbar Q
A B
C
D, left
s1(j)
s0(j)
To
Pads

[3,2]


D5[j]bar
(from SQR unit)
D6[j]bar
(from SQR unit)
D7[j]bar
(from SQR unit)
5
6
7
rsff
D
Q
Qbar
A
B
C
CLK, CLKbar,
below below
D1
S
Sbar
DO
S[j-1](i)
above
CLK, CLKbar,
above above
5
6
7

[1,3]


5
6
7
C B A
SUM CV
bar bar
rsff
Q D
A B
D1 DO Q
A B C
Q's above
D0, below
CLK CLKbar
Q DO D1
S5[j], above
S6[j], above
S7[j], above
Qh(5) = hs2
Qh(6) = hs3
Qh(7) = hs4

[2,3]

inv.fad/fad

all connections shown *(3-2 reduce)*

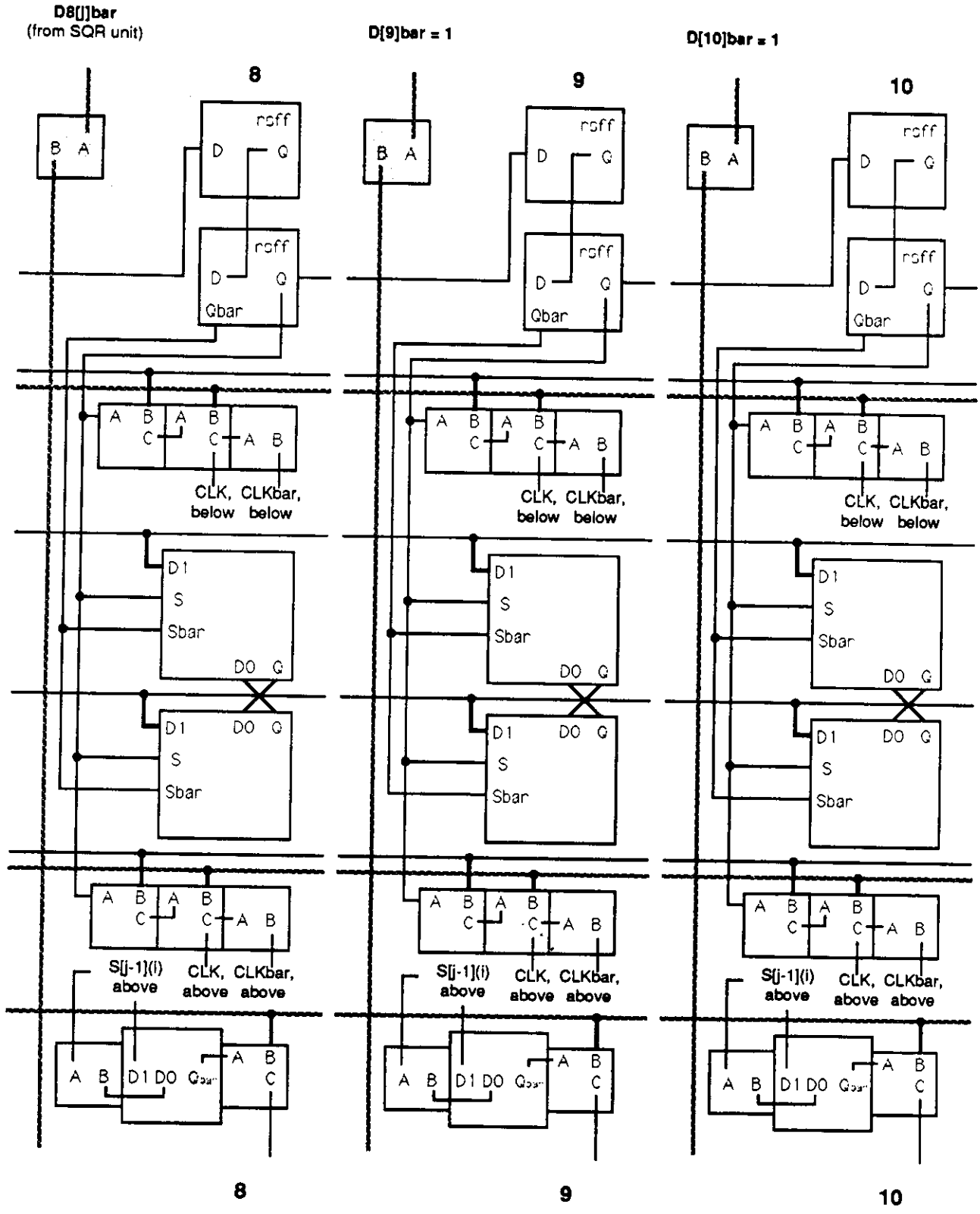
fad

all connections shown *(cpa)*

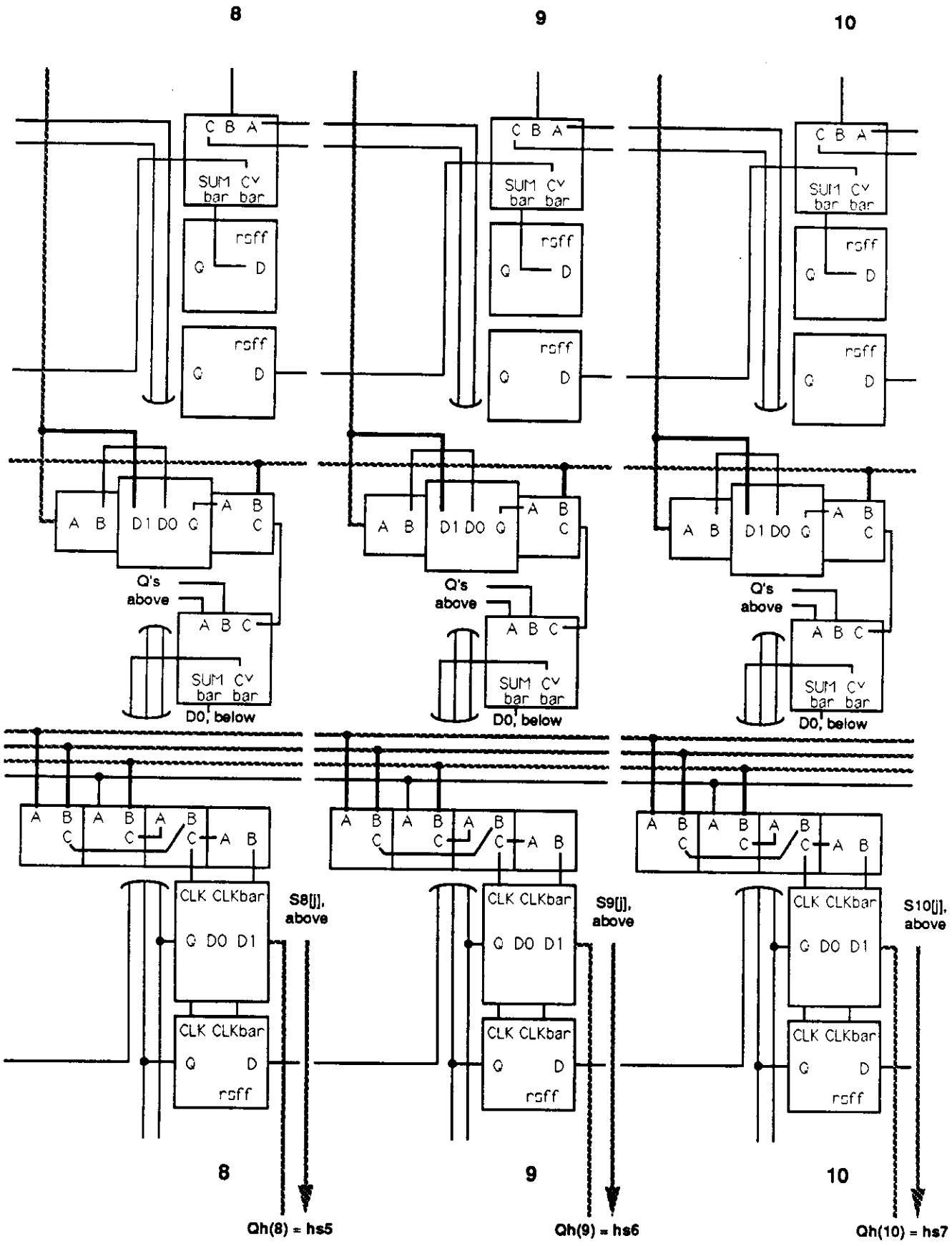
SELECTION SECTION

rsff/logic

PRESET = GRESET
D = shown
CLK = Ø1
CLKbar = Ø1bar
Q = shown
Qbar = shown
logic = all shown *(s-latch)*

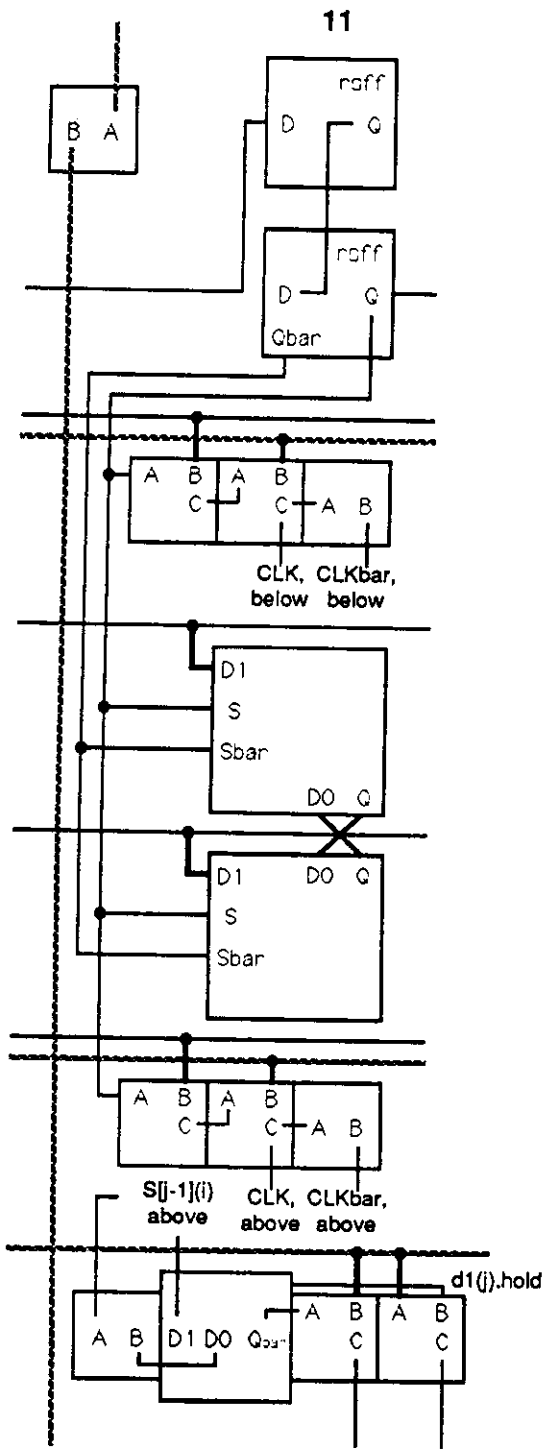

D8[j]bar
(from SQR unit)
D[9]bar = 1
D[10]bar = 1
8
9
10
rsff
D
Q
Qbar
A B
C
CLK, CLKbar,
below below
D1
S
Sbar
DO
S[j-1](i)
above
CLK, CLKbar,
above above
8
9
10

[1,4]


8
9
10
C B A
SUM CV
bar bar
rsff
G D
A B D1 DO Q A B C
Q's above
A B C
SUM CV
bar bar
DO' below
A B C A B C A B C A B
CLK CLKbar
G DO D1
S8[j], above
S9[j], above
S10[j], above
Qh(8) = hs5
Qh(9) = hs6
Qh(10) = hs7

[2,4]

D[11]bar = 1



11

rsff
[Except sff in m(-6)]

PRESET = GRESET
D = shown
CLK = Ø2
CLKbar = Ø2bar
Q = shown
Qbar = unused

(shift-register)

PRESET = GRESET
D = shown
CLK = Ø1
CLKbar = Ø1bar
Q = shown
Qbar = shown

nor2/nor2/inv

all shown

(S-clock)

rsmuxff

PRESET = GRESET
S = shown
Sbar = shown
CLK = C, above
CLKbar = B, above
D0 = shown
D1 = shown
Q = shown
Qbar = unused

(S-latch)

PRESET = GRESET
S = shown
Sbar = shown
CLK = C, below
CLKbar = B, below
D0 = shown
D1 = shown
Q = shown
Qbar = unused

(Sstar-latch)

nor2/nor2/inv

all shown

(Sstar-clock)

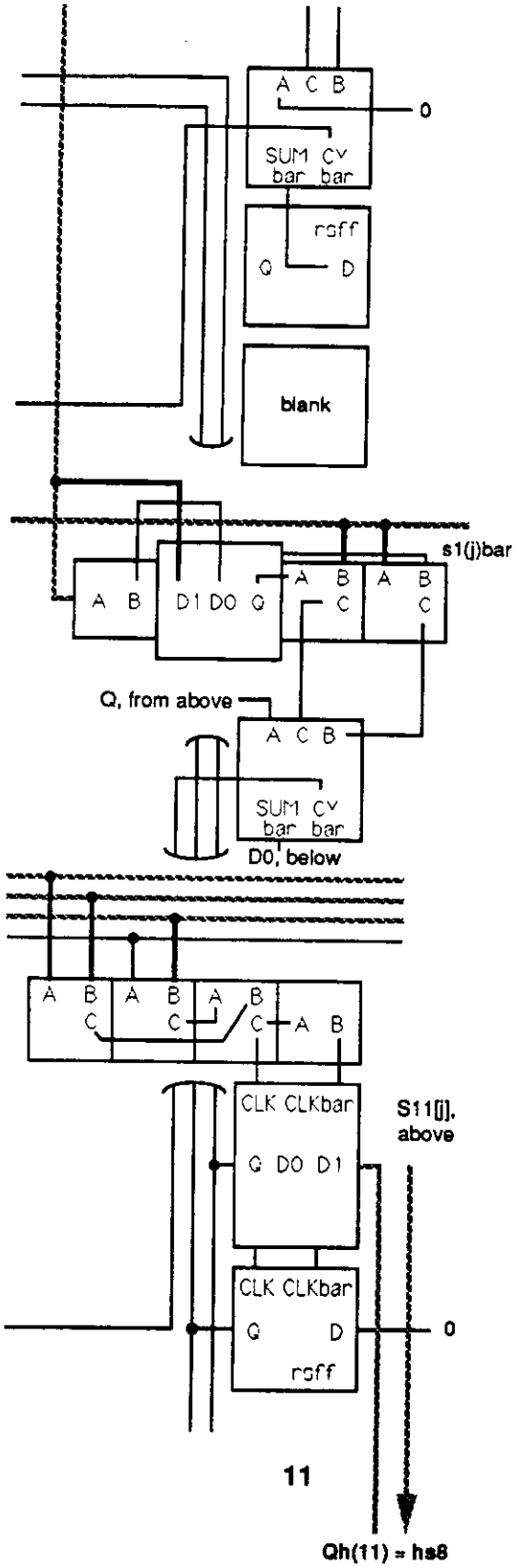
inv/mux/nor2/(nor2)

S = d1(j).hold
Sbar = d1(j)bar.hold
D0 = shown
D1 = shown
Q = unused
Qbar = shown

(Qs-multiplex)

OPERAND SECTION

[1,5]



fad
all shown *(csa-level1)*

rsff

PRESET = GRESET
D = shown
CLK = Ø1
CLKbar = Ø1bar
Q = shown
Qbar = unused *(W-latch.ps)*

PRESET = GRESET
D = shown
CLK = Ø1
CLKbar = Ø1bar
Q = shown
Qbar = unused *(W-latch.sc)*

inv/mux/nand2

S = s1(j)
Sbar = s1(j)bar
D0 = shown
D1 = shown
Q = shown
Qbar = unused *(Qd-multiplex)*

fad
all shown *(csa-level2)*

ARITHMETIC SECTION

rsff [i=1]/logic

PRESET = GRESET
D = shown
CLK = Ø1
CLKbar = Ø1bar
Q = shown
Qbar = unused
logic = all shown *(P-enable/P-clock)*

muxff

S = ENABLE
Sbar = ENABLEbar
CLK = C, above
CLKbar = B, above
D0 = shown
D1 = shown
Q = shown
Qbar = unused *(P-latch.ps)*

rsff

PRESET = GRESET
D = shown
CLK = C, above
CLKbar = B, above
Q = shown
Qbar = unused *(P-latch.sc)*

[2,5]

6.1 Standard Cells (SS)

- **Cell Summary**

Cell	Type	Size (HxW)	Nodes	Description
inv.mag	logic	47x16=752	A B	input output
nand2.mag		47x25=1175	A,B C	inputs output
nand3.mag		47x33=1551	A,B,C D	inputs output
nor2.mag		47x25=1175	A,B C	inputs output
nor3.mag		47x33=1551	A,B,C D	inputs output
mux.mag		75x53=3975	S,Sbar D0,D1 Q,Qbar	selects inputs outputs
fad.mag	math	87x56=4872	A,B,C CYbar SUMbar	inputs output output
inv.fad.mag		87x20=1740	A B	input output
dff.mag	flip-flop	61x40=2440	D CLK,CLKbar Q,Qbar	data input clocks outputs
rsff.mag		68x48=3264	PRESET D CLK,CLKbar Q,Qbar	reset data input clocks outputs
sff.mag		68x58=3944	PRESET D CLK,CLKbar Q,Qbar	preset data input clocks outputs
muxff.mag		103x61=6283	S,Sbar D0,D1 CLK,CLKbar Q,Qbar	selects data inputs clocks outputs
rsmuxff.mag		110x69=7590	PRESET S,Sbar D0,D1 CLK,CLKbar Q,Qbar	reset selects data inputs clocks outputs

muxshift.mag	shift	117x100=11700	S,Sbar	selects
			shiftin0,D1	inputs
			CLKM,CLKMbar	master clocks
			CLKS,CLKSbar	slave clocks
			QM,QMbar	outputs
			QS,QSbar	outputs
			shiftout	(=QS)
rshift.mag		82x88=7216	PRESET	reset master
			shiftin	shift input
			CLKM,CLKMbar	master clocks
			CLKS,CLKSbar	slave clocks
			QM,QMbar	outputs
			QS,QSbar	outputs
			shiftout	(=QS)
sshift.mag		82x98=8036	PRESET	preset master
			shiftin	shift input
			CLKM,CLKMbar	master clocks
			CLKS,CLKSbar	slave clocks
			QM,QMbar	outputs
			QS,QSbar	outputs
			shiftout	(=QS)
buff.mag	buffer	47x25=1175	A	input
			B	output

• Assumptions

1. The mobility of a p-transistor is given approximately by:

   ```
   mu(p) = (1/2 to 1/3) * mu(n)
   ```

   The current drive capability of a transistor is directly proportional to the mobility and the W/L ratio in the channel (under the gate), given by:

   ```
   i(p) => mu(p) * W/L(p)
   i(n) => mu(n) * W/L(n)
   ```

   Therefore, the W/L ratio of p-transistor tree (the "p-tree") in any cell must be adjusted with respect to the W/L ratio of the n-transistor tree (the "n-tree") in order for both to provide equivalent current sourcing and sinking capability, and hence equivalent rise and fall times. In most cases, the determination of a tree's equivalent W/L must be made on a cell-by-cell basis, because the series/parallel connection of each type of transistor has an effect on the equivalent W/L for the entire tree.

2. An approximation to equivalent drive capabilities is achieved by maintaining a ratio of 2:1 between the equivalent W/L's of the p-type and n-type trees:

   $$\frac{\text{equivalent W/L of p-tree}}{\text{equivalent W/L of n-tree}} = 2$$

   The purpose of this is to compensate for the lower mobility of the p-transistors, making the p-tree as capable of sourcing current (pulling the output high) as the n-tree is capable of sinking current (pulling the output low). Rise/Fall times are thus made equal.

3. W/L equivalences are computed similarly to resistance equivelances. For instance, for two transistors in series, the equivalent W/L ratio is HALF that of a single transistor. Equivalent W/L's for transistors in series and parallel are given by the following formulas:

$$\text{Series:} \quad W/L(\text{equiv}) = \frac{1}{L/W\ (1st) + L/W\ (2nd) + \ldots}$$

$$\text{Parallel:} \quad W/L(\text{equiv}) = W/L\ (1st) + W/L\ (2nd) + \ldots$$

Quick Rule: For series transistors, if the numerators are the same, simply add denominators and place under the numerator (a hack)

For parallel transistors, if the denominators are the same, simply add numerators and place over the denominator (add as fractions)

Special Case: See note (4) below !!!!!!!

4. In the case where 1 or more transistors (of either type) are in PARALLEL, we should consider the W/L of a SINGLE transistor, rather than the equivalent W/L of all transistors in parallel, in calculating the 2:1 ratio. This is because, in the worst case, only one of the transistors may be on, and thus have to source/sink all the output node current itself.

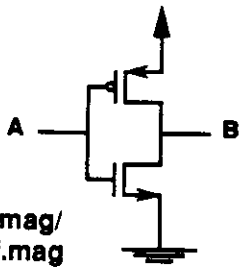
5. Finally, one must consider the effects of the increased gate capacitance of any cell whose p-transistors are scaled to produce a 2:1 W/L ratio between the trees. Since gate capacitance comprises the bulk of the load on any p-transistor trying to raise an input to a scaled gate, a compromise must be reached between scaling to 2:1 to increase p-transistor drive versus the speed loss due to the increased gate capacitance. Simulation shows that a rather flat optimum exists at a ratio of about **1.5:1**. This was the tree-to-tree ratio selected for our cells.

• Limitations

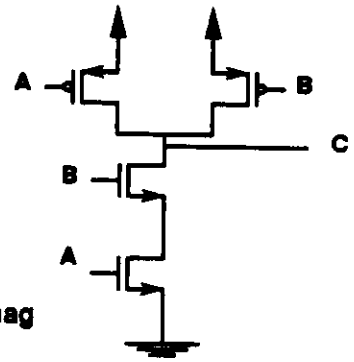
Any flip-flop which is preset-able, including rsff, sff, rsmuxff, rshift, and sshift is subject to certain TIMING ANAMOLIES. These anamolies involve the behavior of the Qbar output during the moment in which BOTH the GRESET signal and the clock to that flip-flop are high. See the separate design sheet for details.

6.2 Standard Cells (DS)

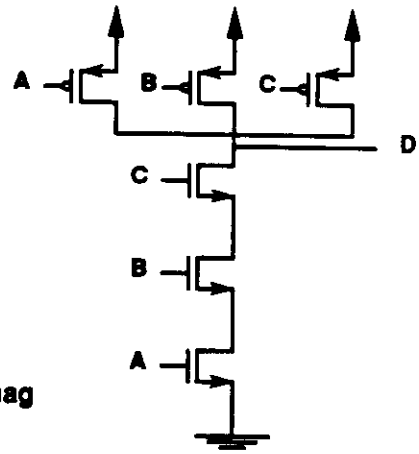
- Logic Cells



inv.mag/
buff.mag

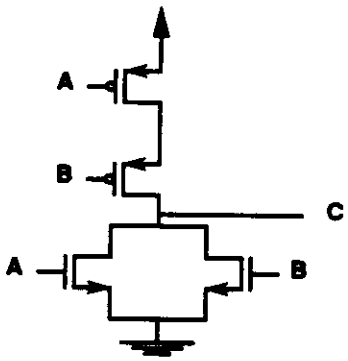


nand2.mag

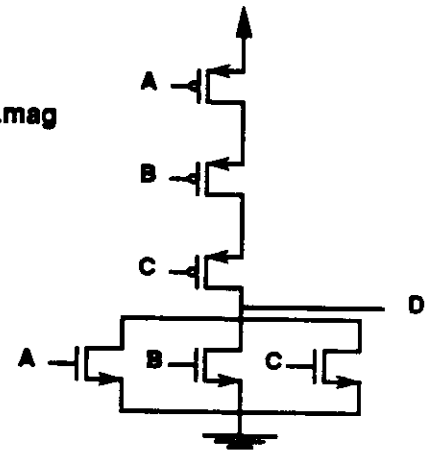


nand3.mag

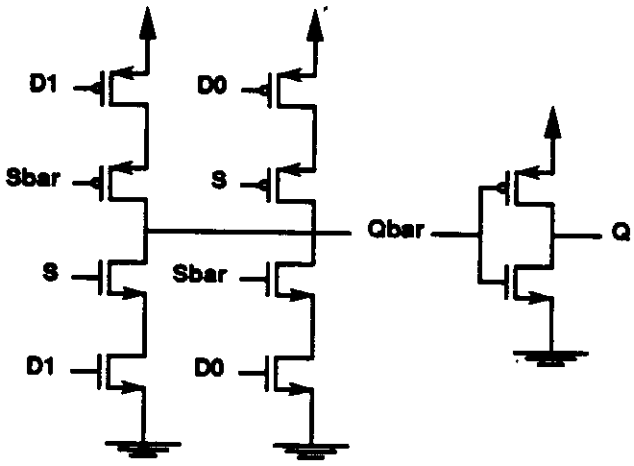
nor2.mag



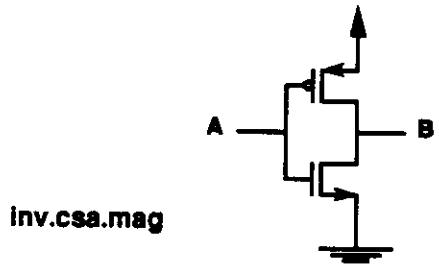
nor3.mag



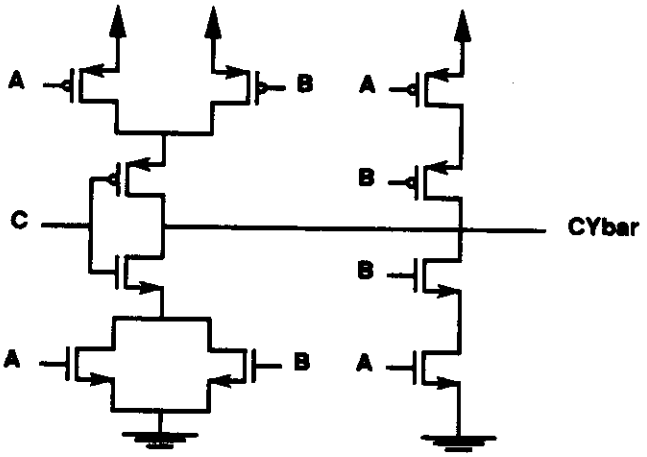
mux.mag



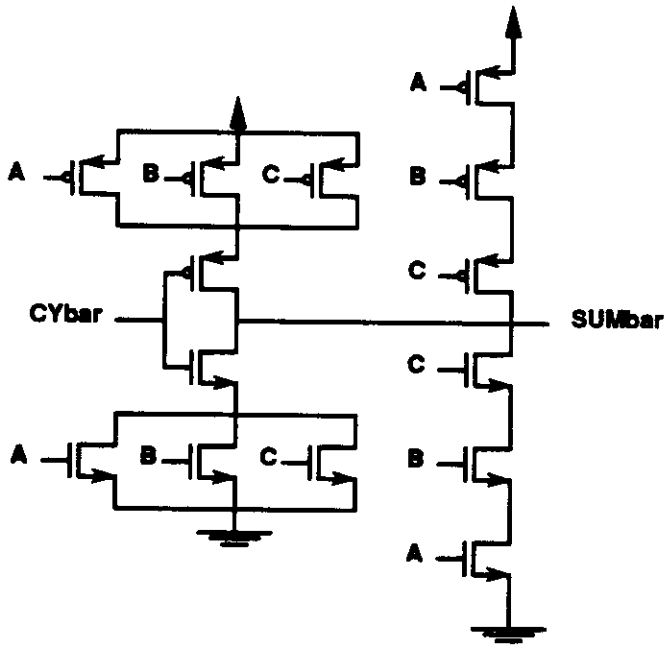
• **Math Cells**



inv.csa.mag



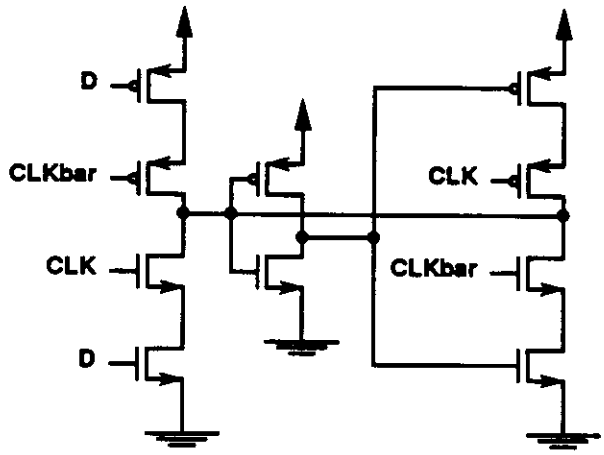
fad.mag
(carry stage)



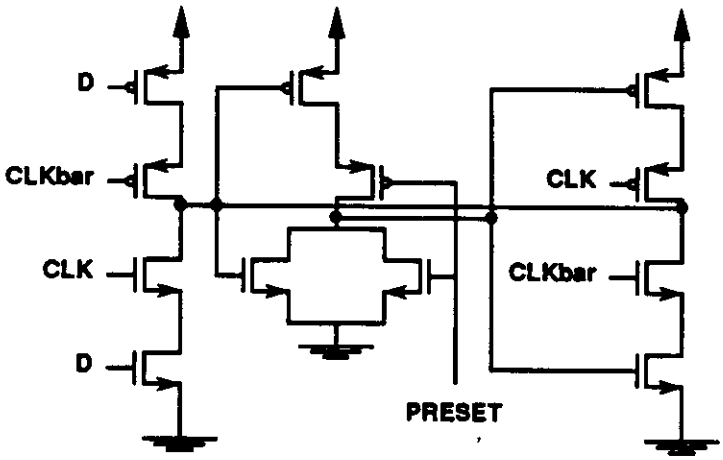
fad.mag
(sum stage)

• Flip-Flop Cells

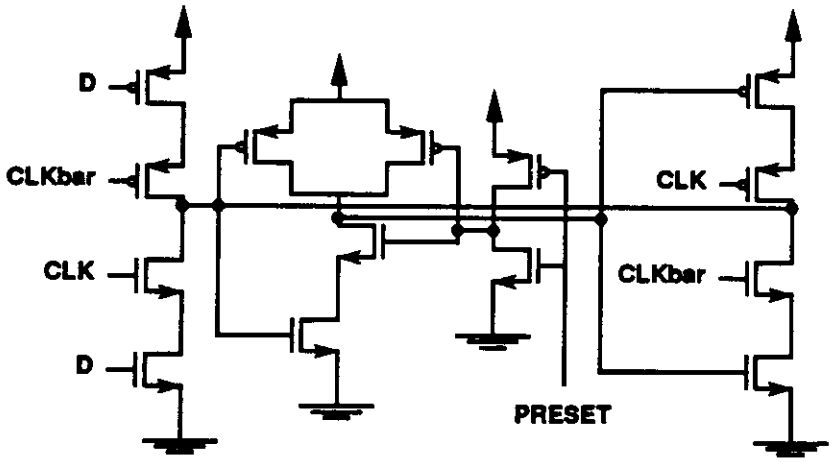
dff.mag



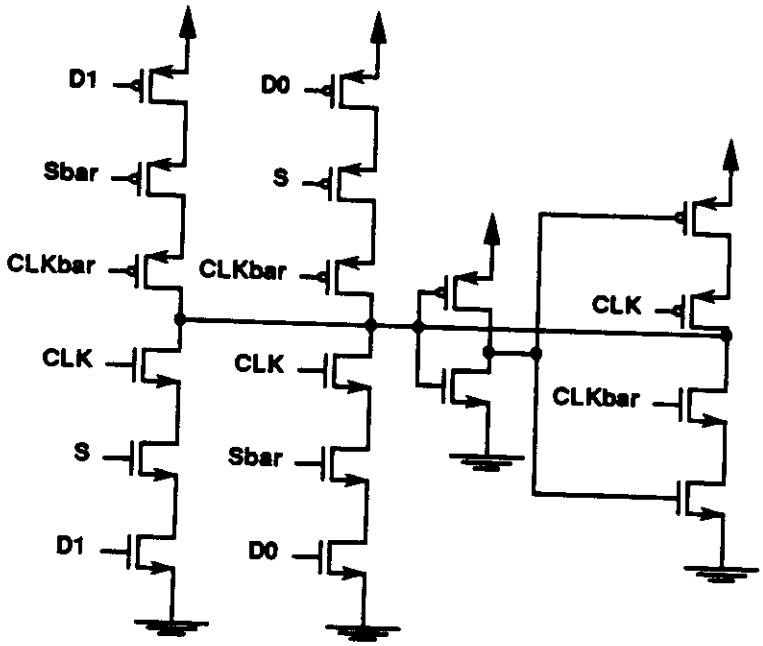
rsff.mag



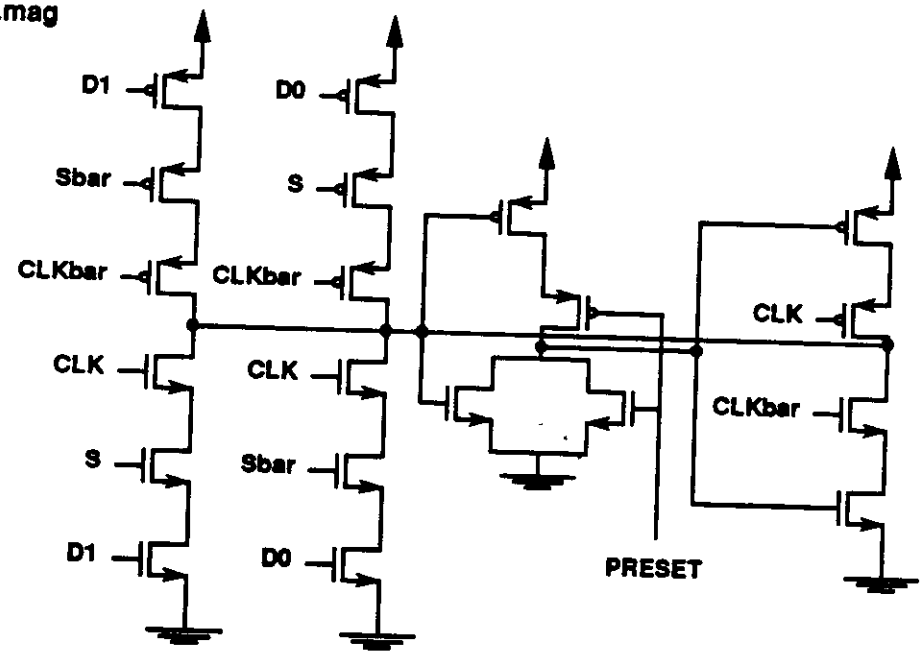
sff.mag



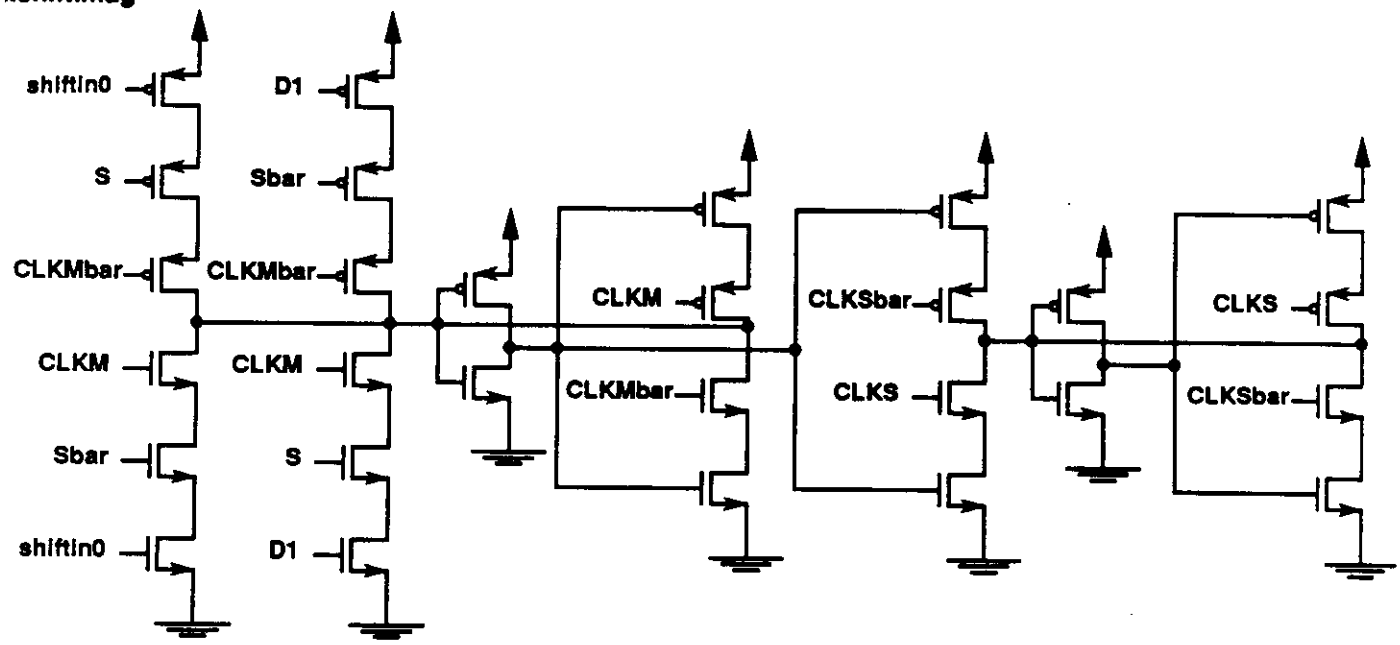
muxff.mag



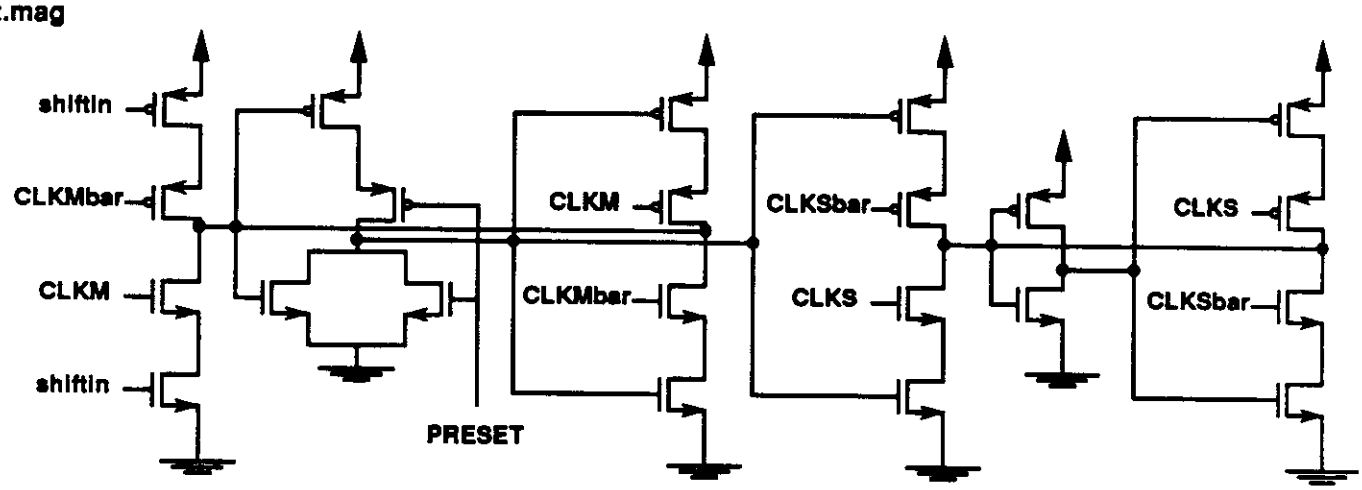
rsmuxff.mag



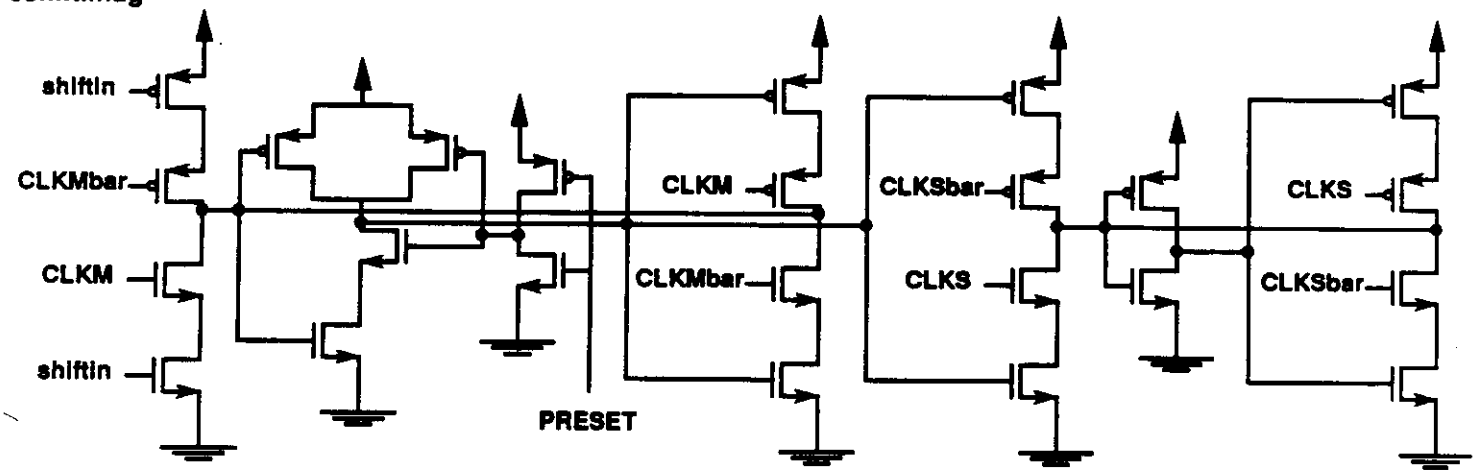
xshift.mag



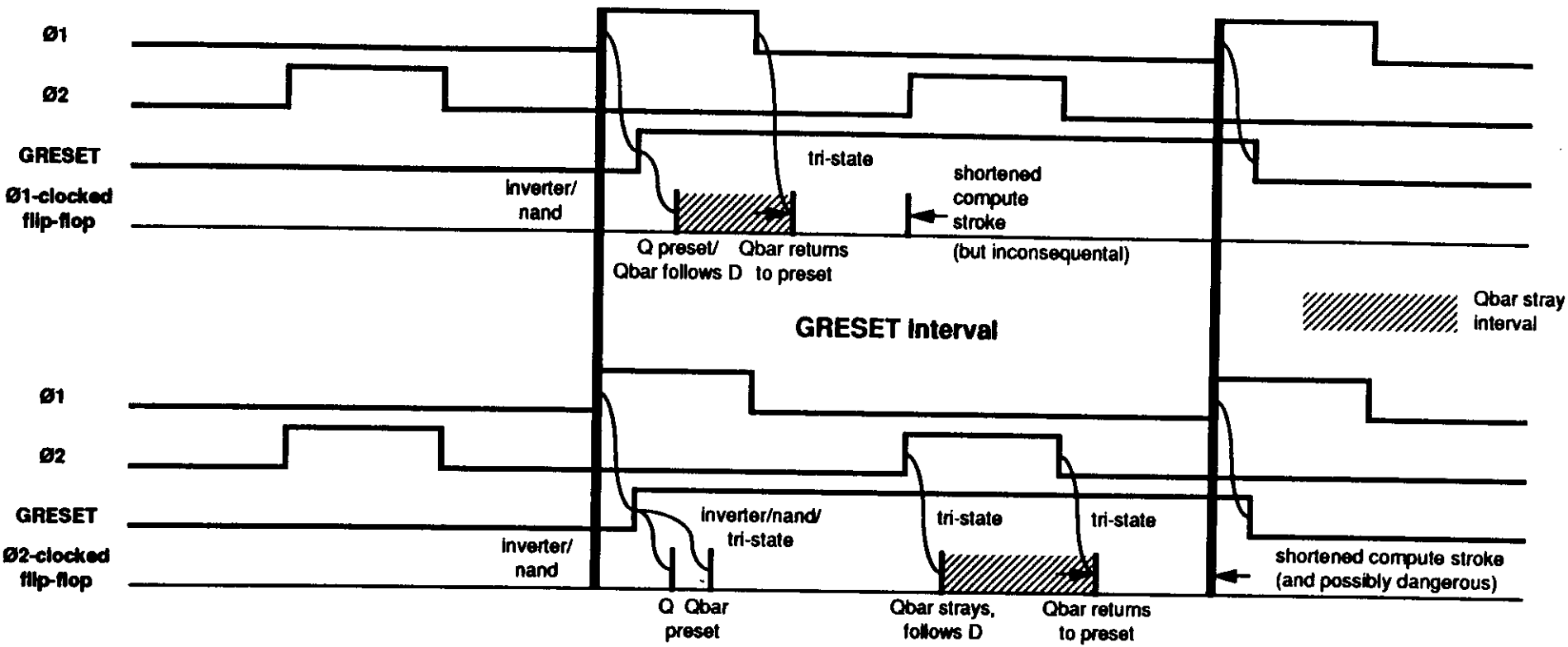
rshift.mag



sshift.mag



6.3 Flip-flop Timing Anamoly (DS)



NOTES: The diagrams above illustrate the problem encountered in using the Qbar output of any flip-flop, namely, that the value of Qbar will stray (following D as Dbar) **during the period in which the GRESET signal is high and the clock of that flip-flop is high**, resulting in a shortened computation time for the following stroke. The resulting stroke lasts from approximately fall-to-rise instead of rise-to-rise.

This impacts the design as follows. If a flip-flop is Ø1-clocked, it is OK to use its Qbar output, because the following Ø2 stroke, which would clock in the results of gating a Ø1-clocked flip-flop's output thru some logic, is run over by the GRESET signal. However, using the Qbar of a Ø2-clocked flip-flop may be dangerous, since the following Ø1 stroke is in a valid non-GRESET interval, and will latch in whatever was computed during the shortened interval.

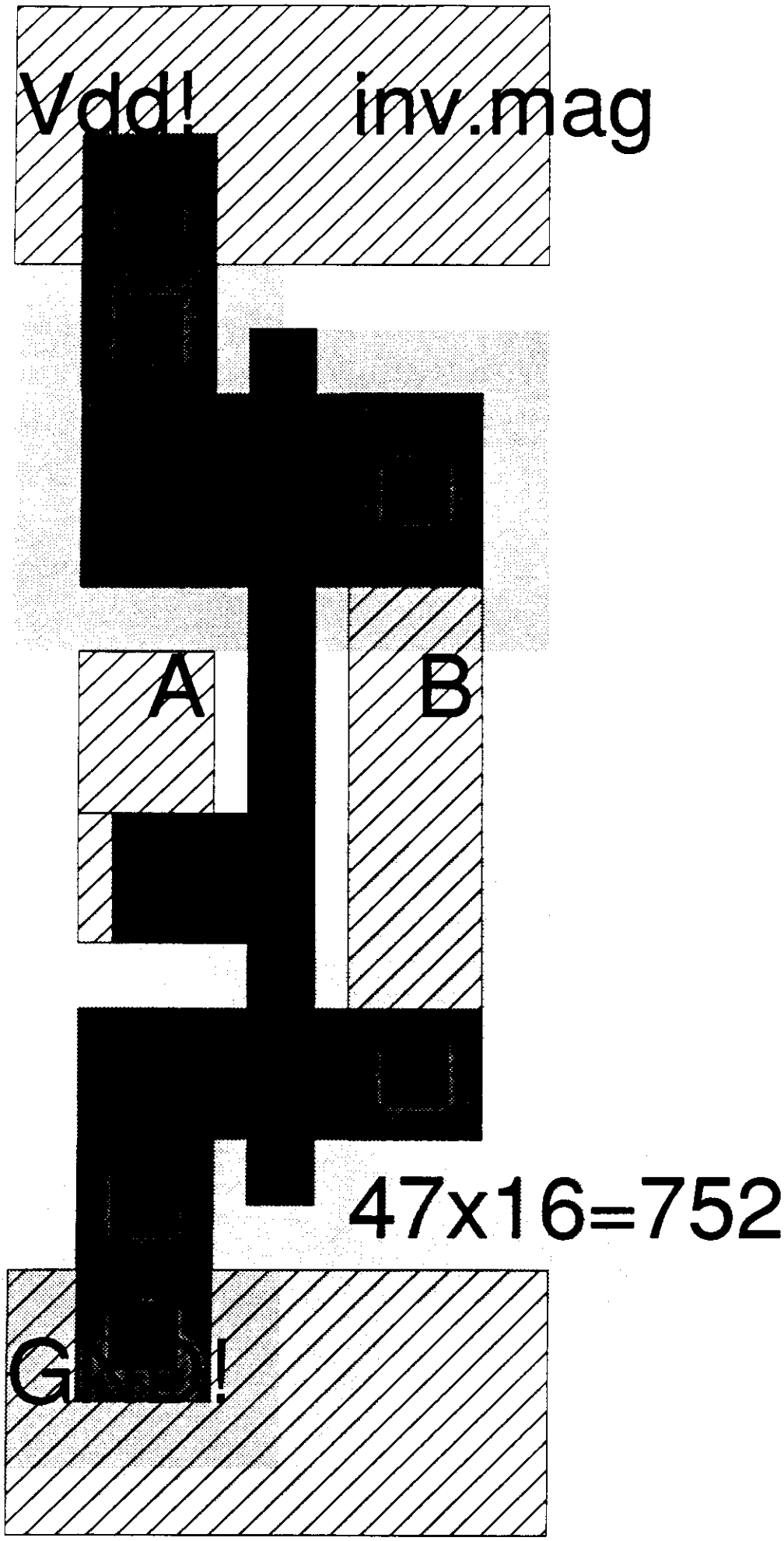
Summarizing, it is OK to use the Qbar output of a Ø1-clocked flip-flop since the following Ø2 stroke is inconsequental. However, when using the Qbar output of a Ø2-clocked flip-flop, you must ensure that adequate computation time exists between the **fall** of that Ø2 stroke and the **rise** of the following Ø1 stroke.

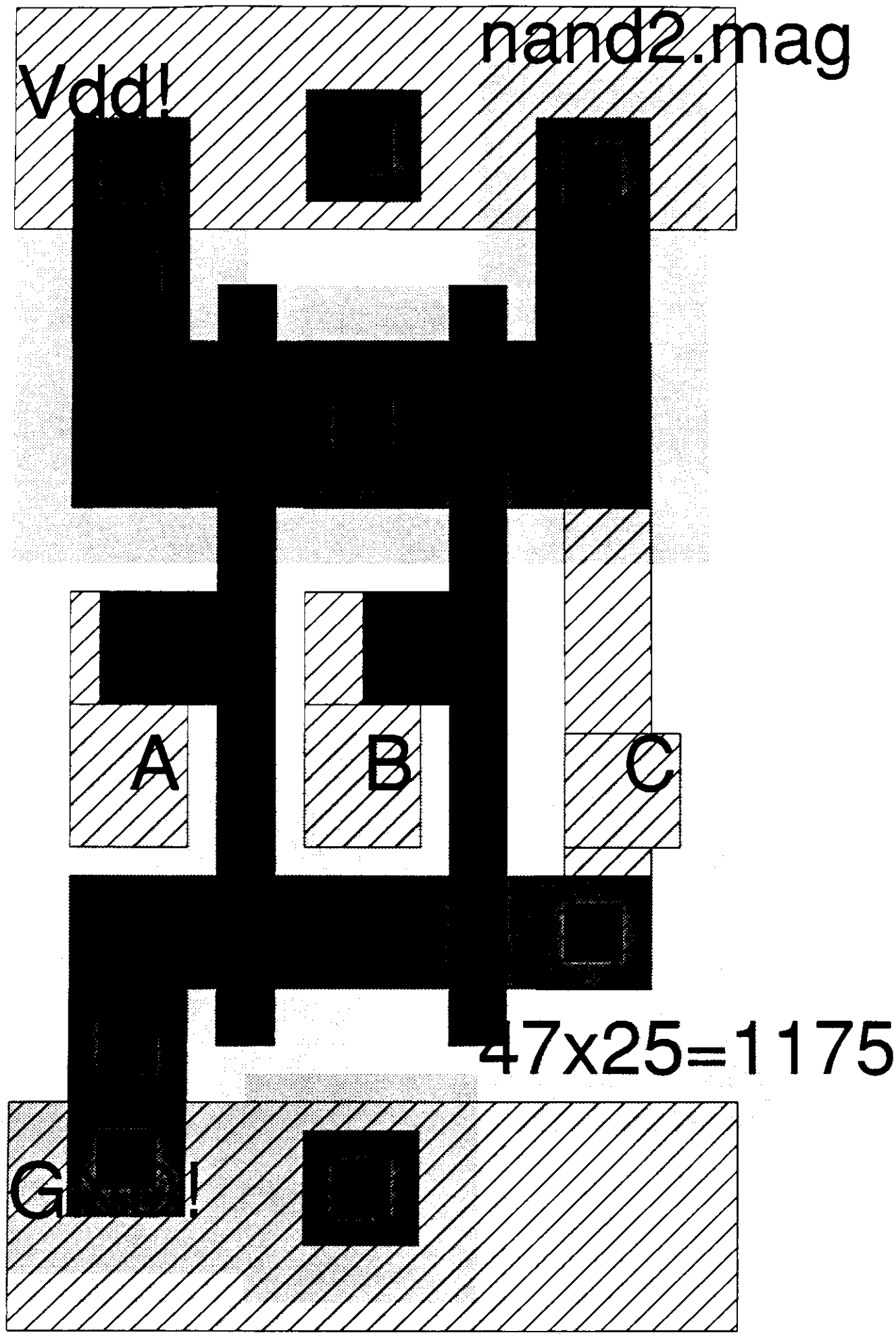
SUMMARY

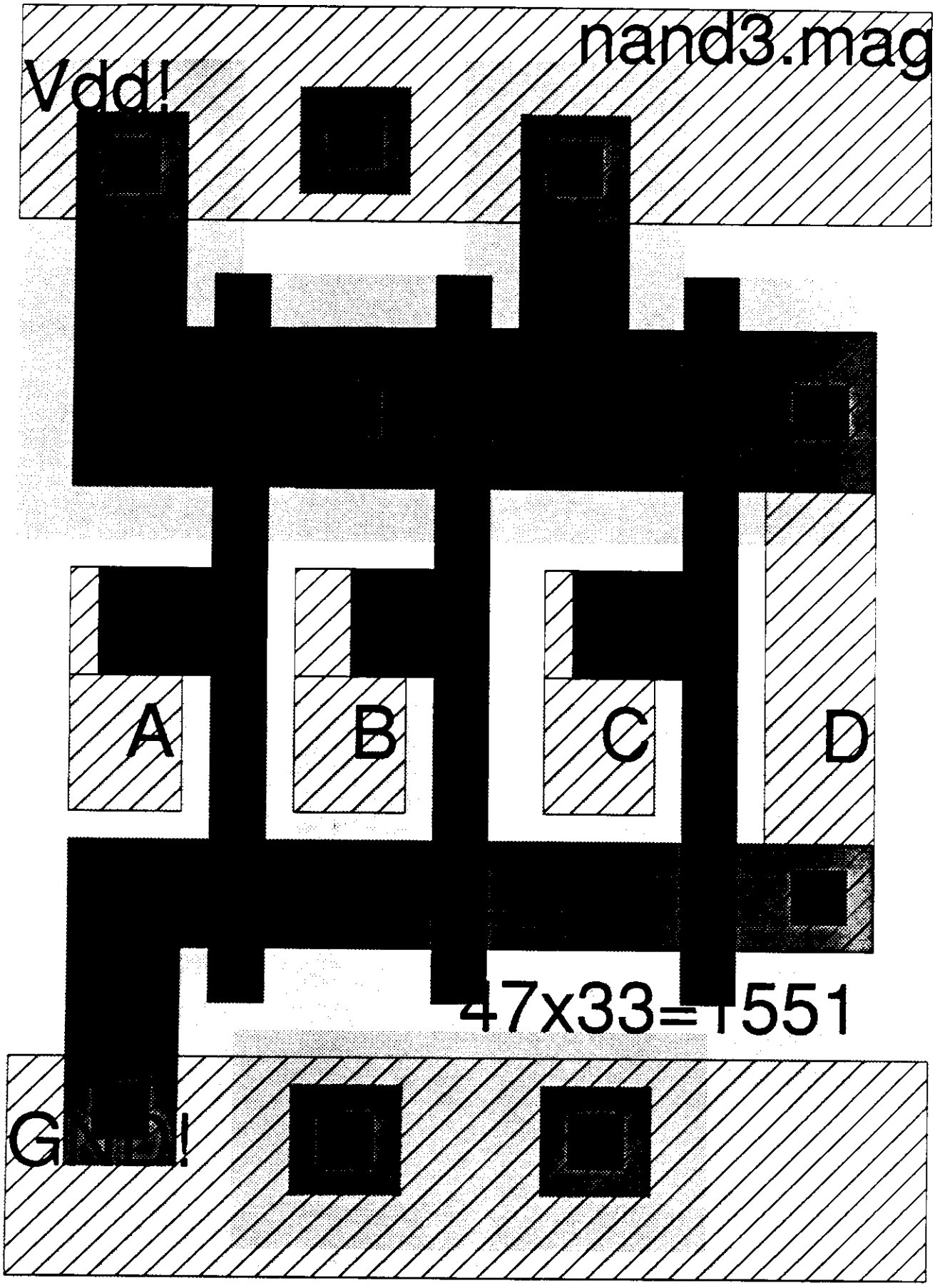
OK to use Qbar of **Ø1-clocked** flip-flop

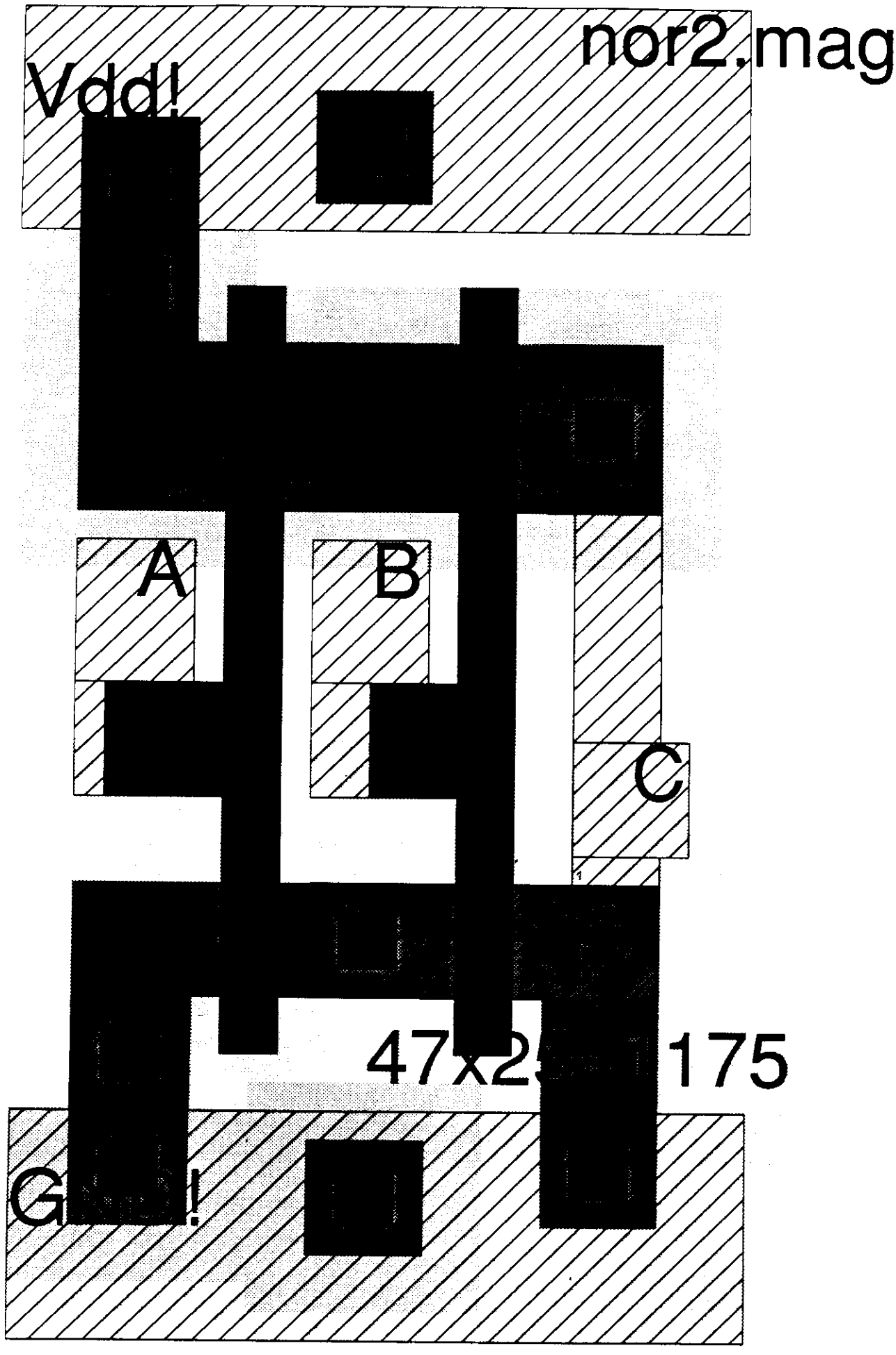
NOT OK to use Qbar of **Ø2-clocked** flip-flop, except if adequate computation time exists from fall-to-rise.

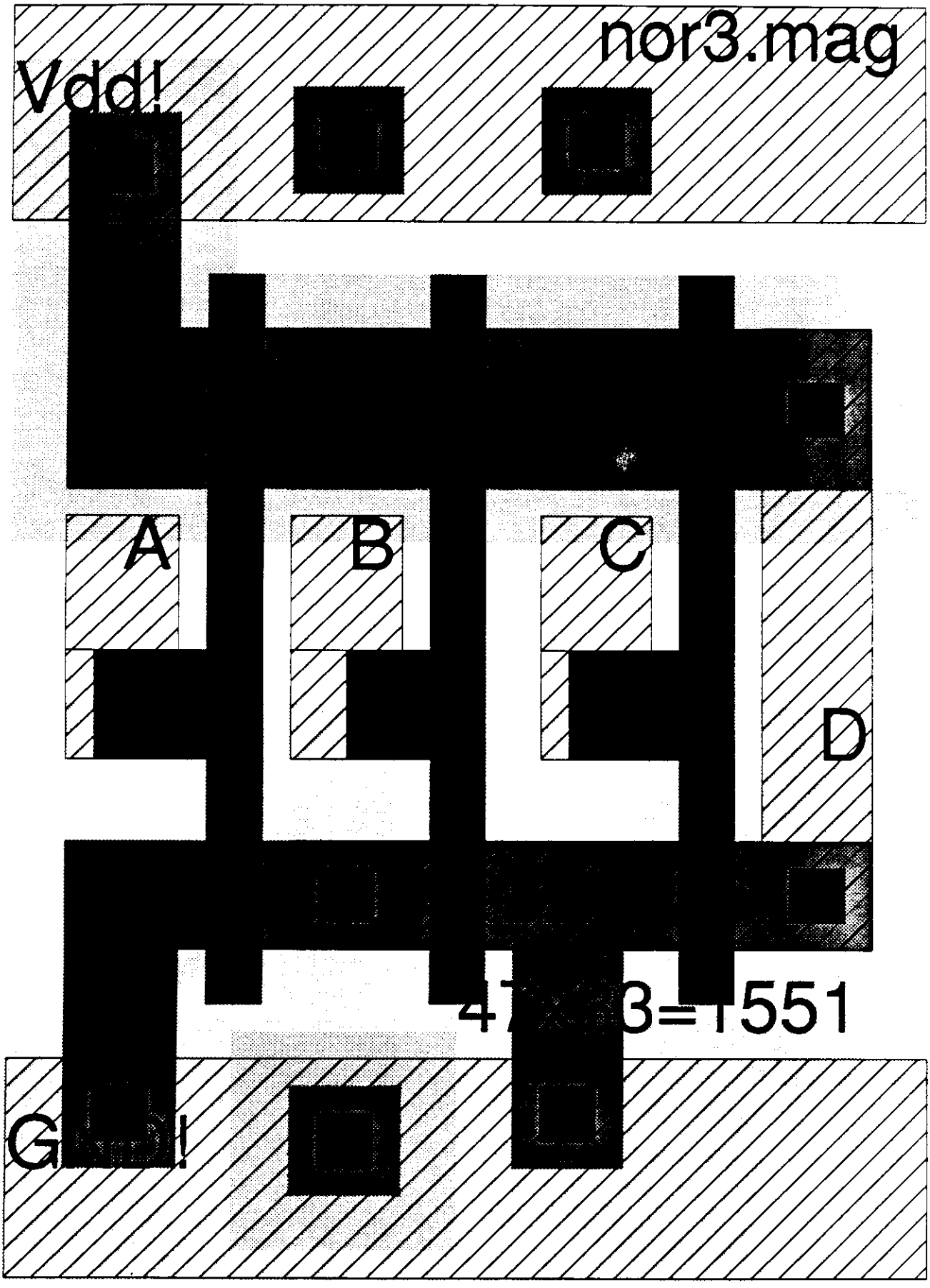
6.4 Standard Cell Layout Plots (DS)

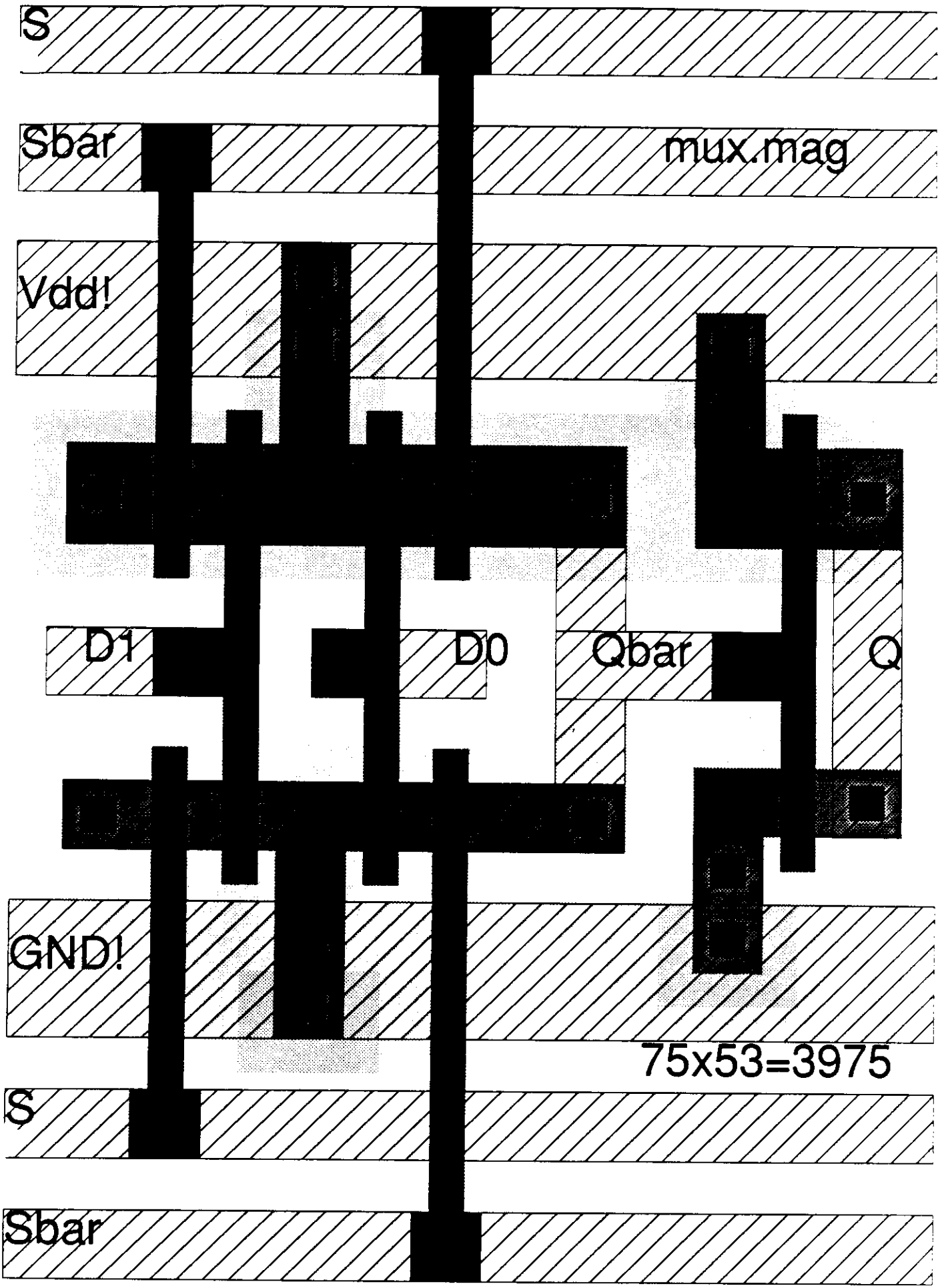

Vdd!
inv.mag
A
B
47x16=752
G
!

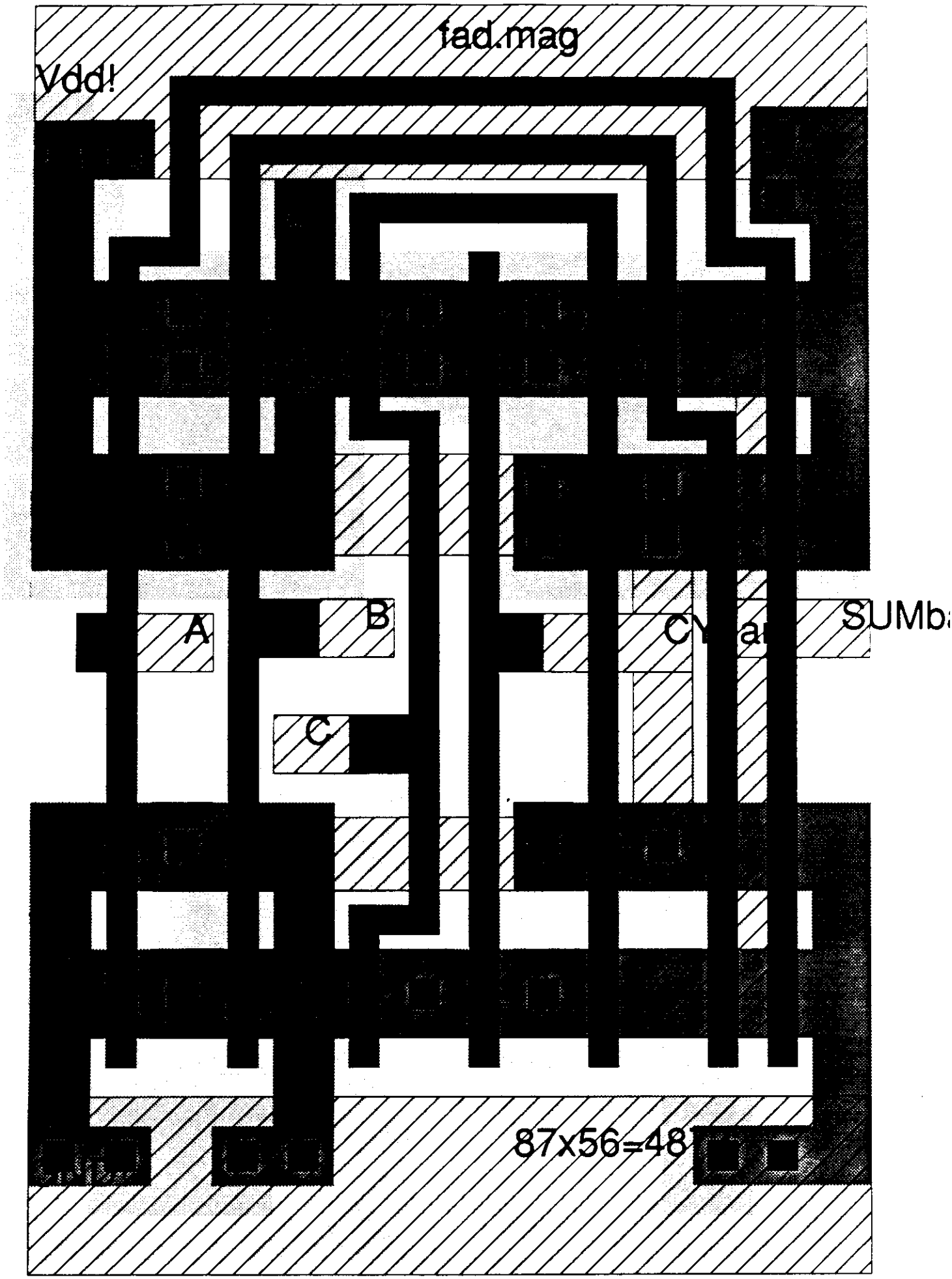

nand2.mag
Vdd!
A
B
C
47x25=1175
G
!

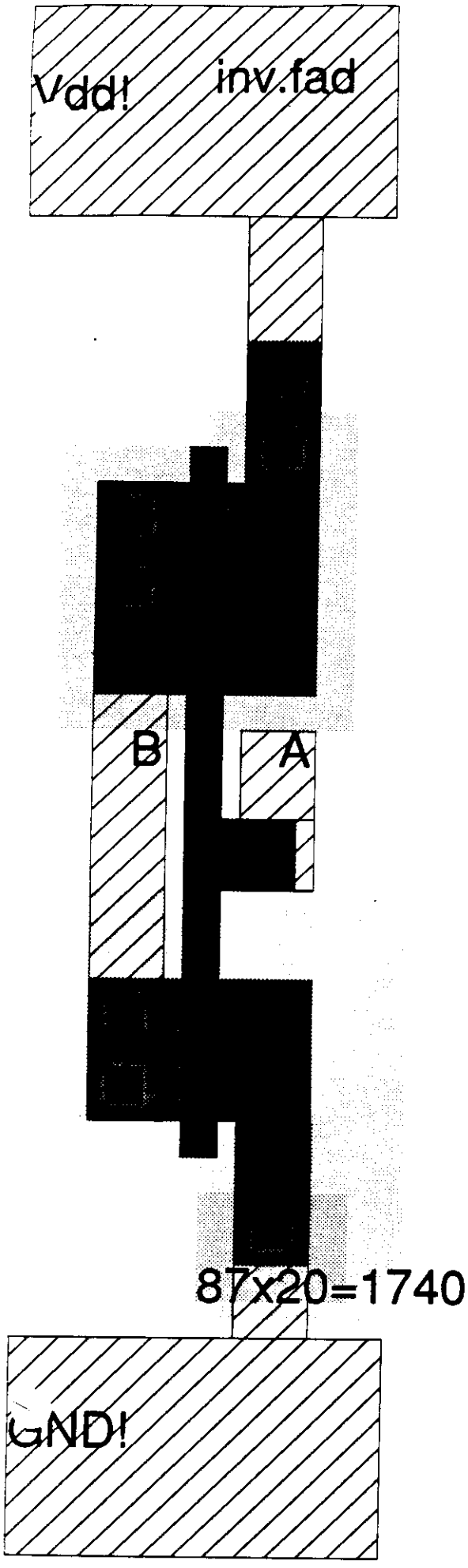

nand3.mag
Vdd!
A
B
C
D
47x33=1551
Gnd!

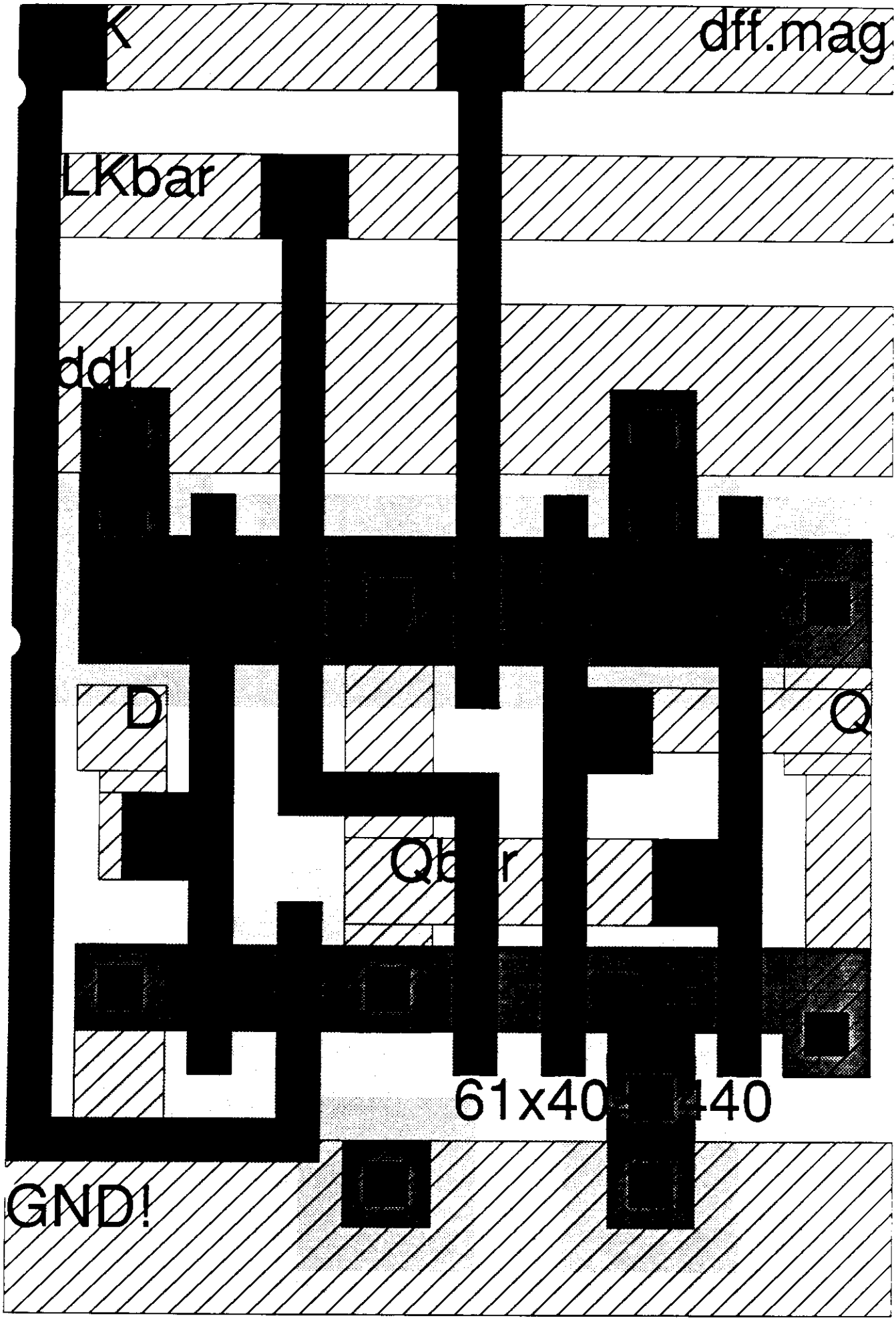

nor2.mag
Vdd!
A
B
C
47x25
175
G
!

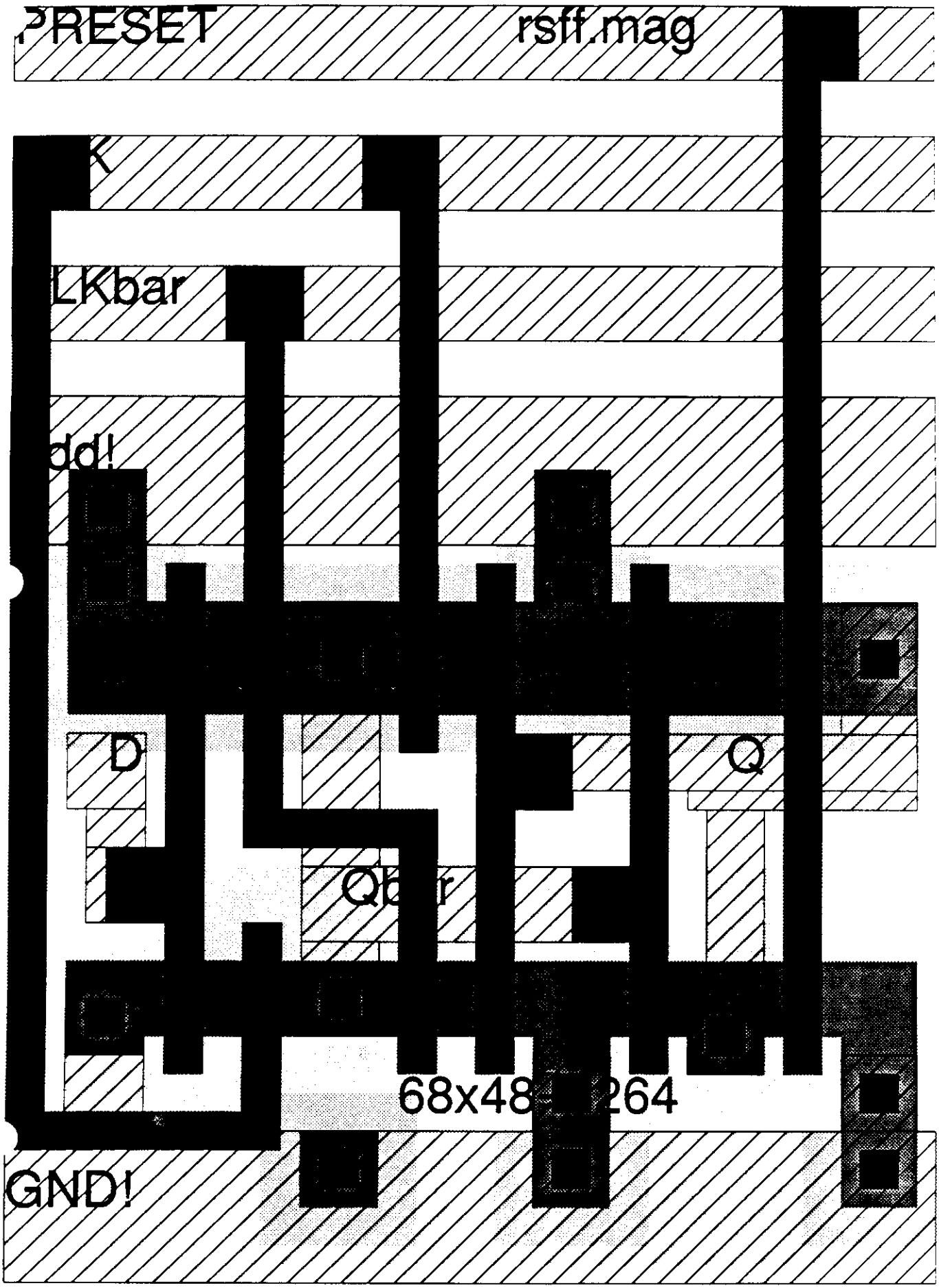

nor3.mag
Vdd!
A
B
C
D
Gnd!

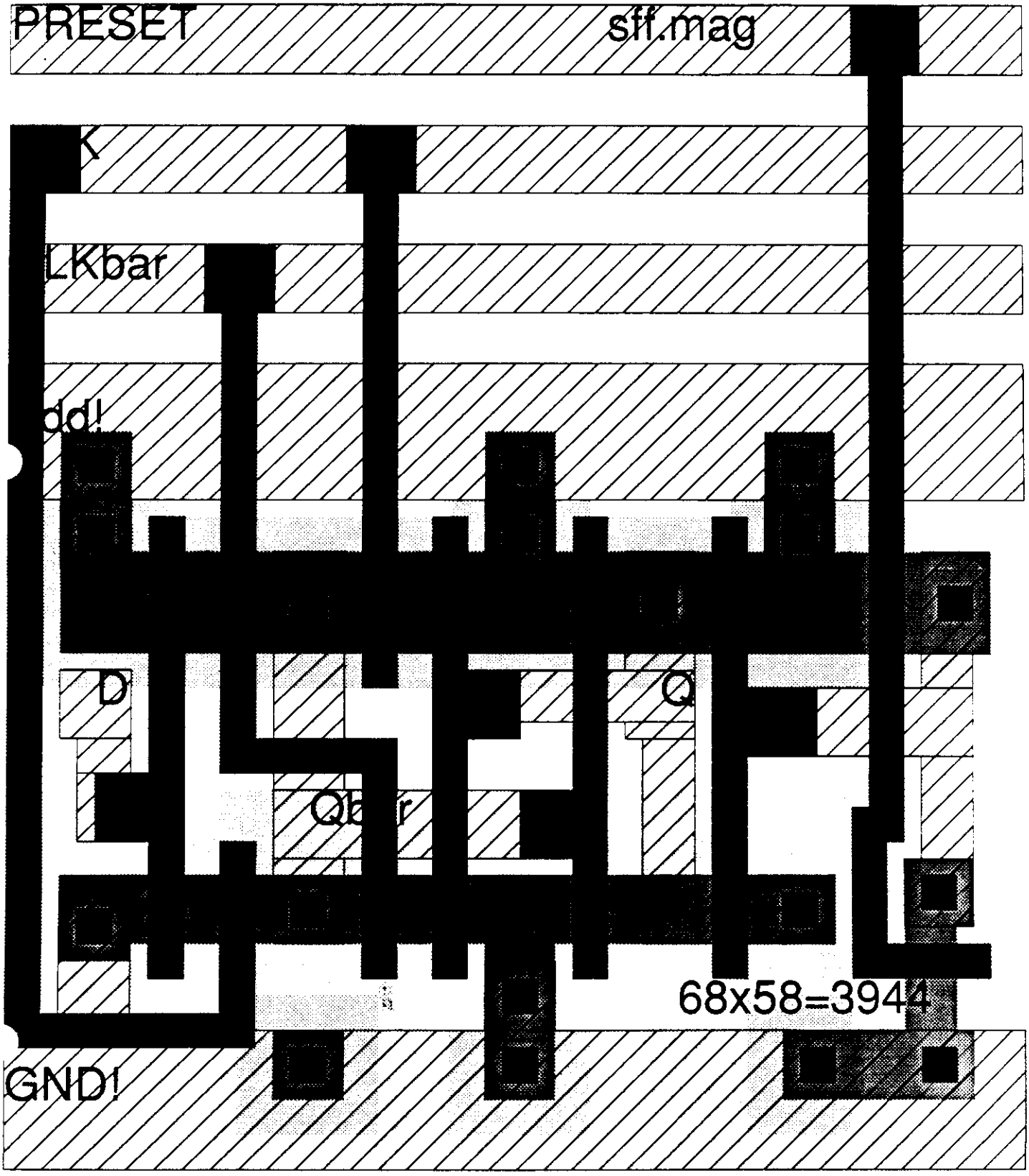

S
Sbar
mux.mag
Vdd!
D1
D0
Qbar
Q
GND!
75x53=3975
S
Sbar

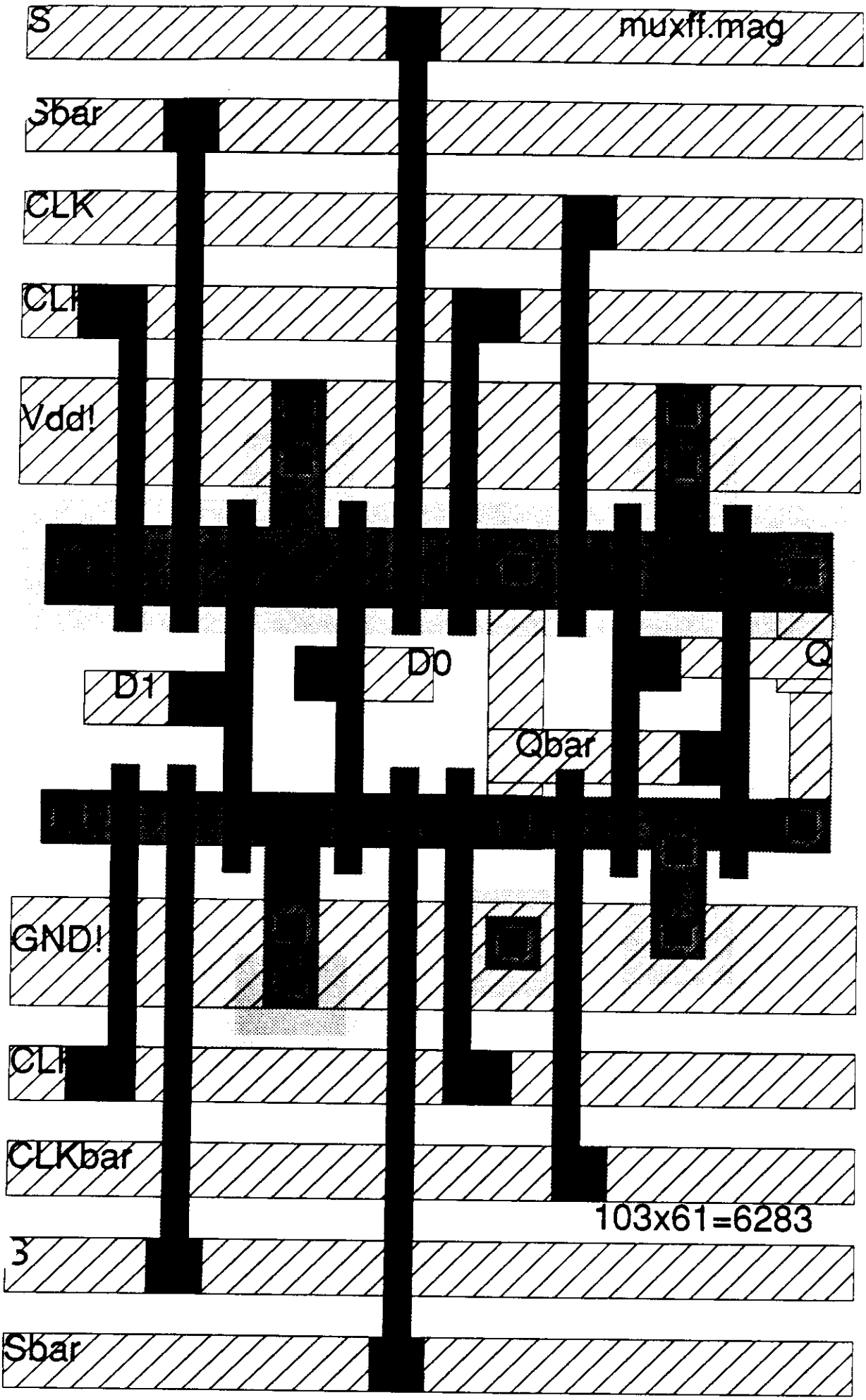

fad.mag
Vdd!
A
B
C
SUMb
87x56=48

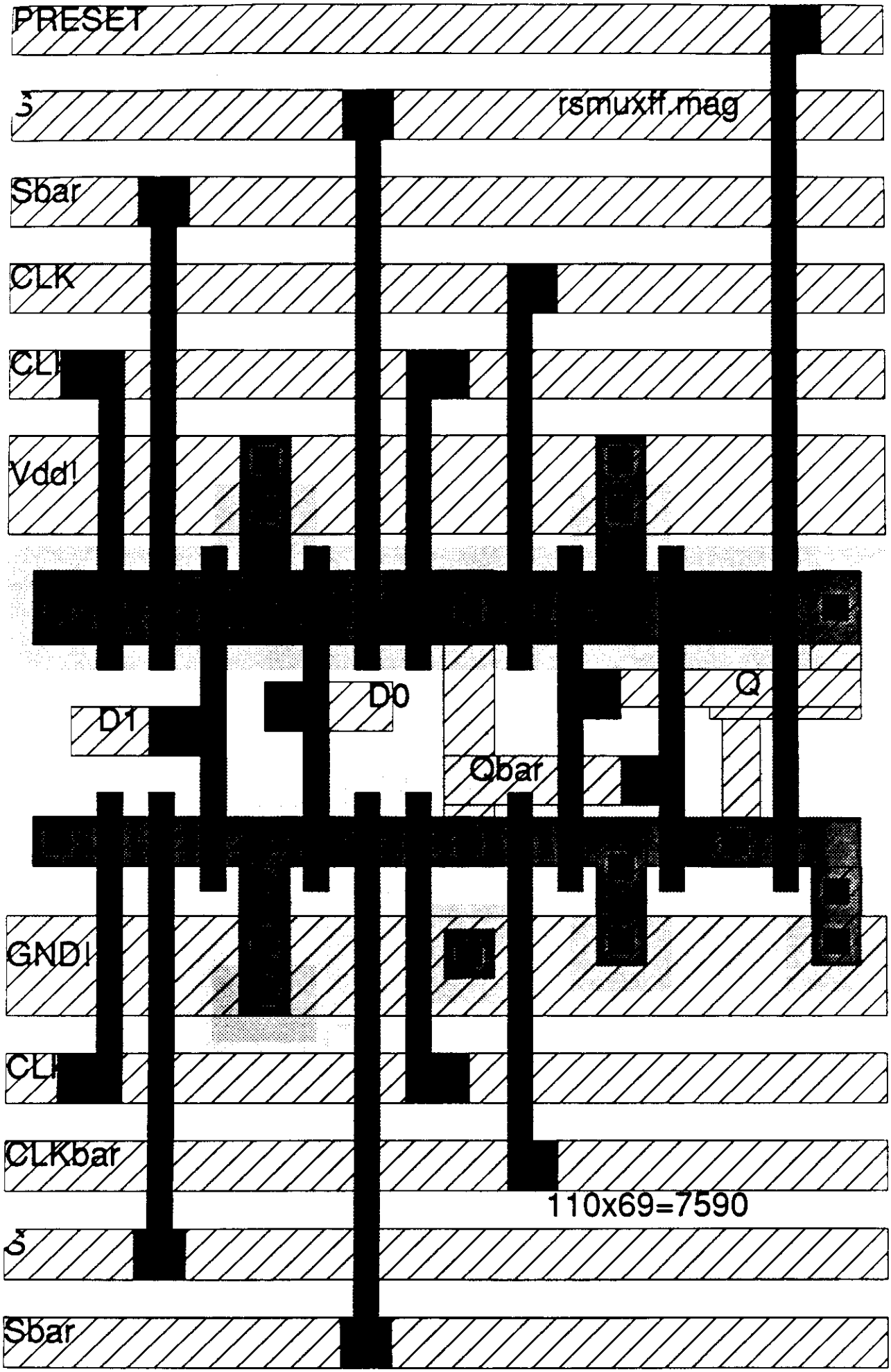

Vdd!
inv.fad
B
A
87x20=1740
GND!

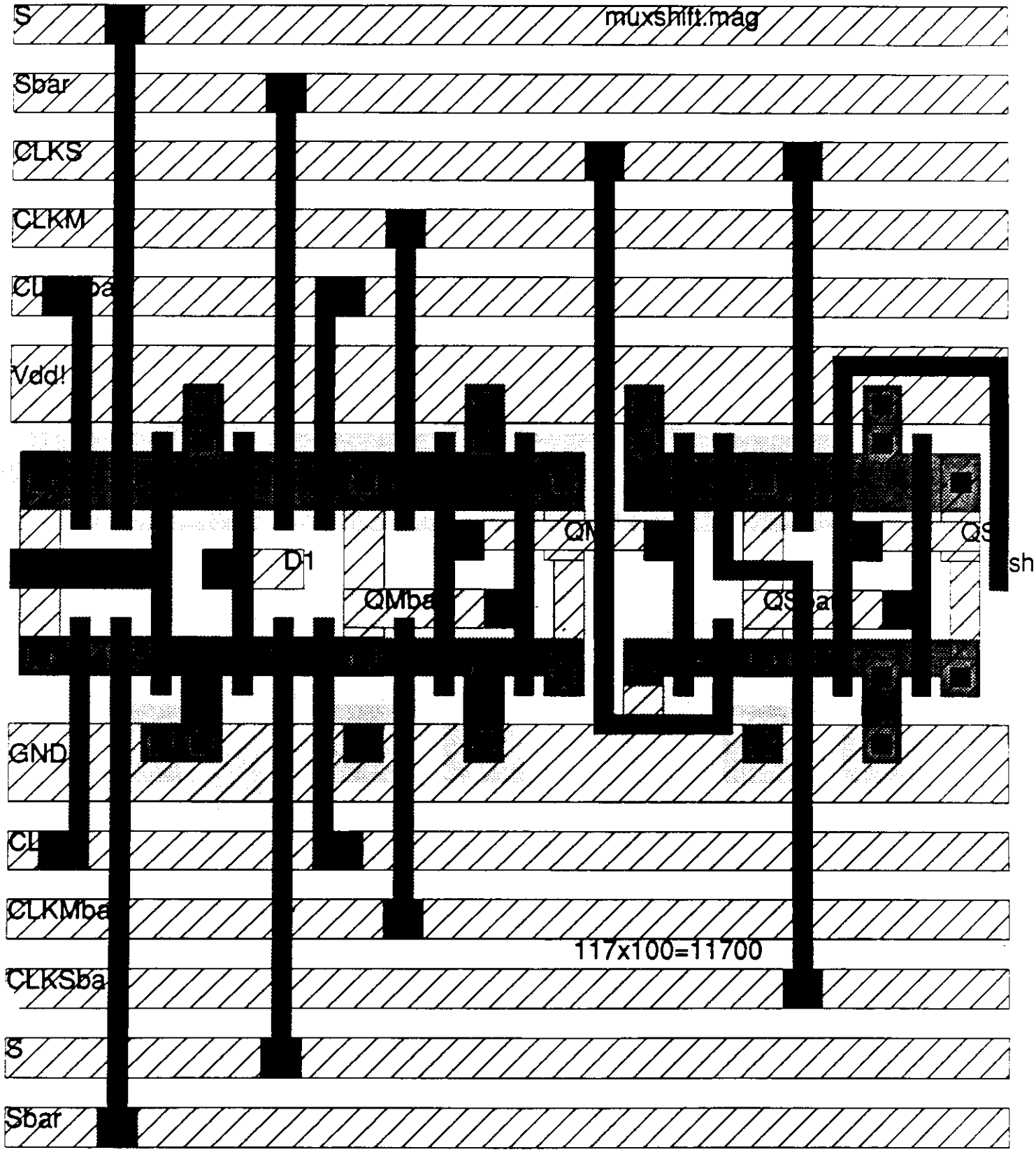

K
dff.mag
LKbar
dd!
D
Q
Qbar
61x40
GND!

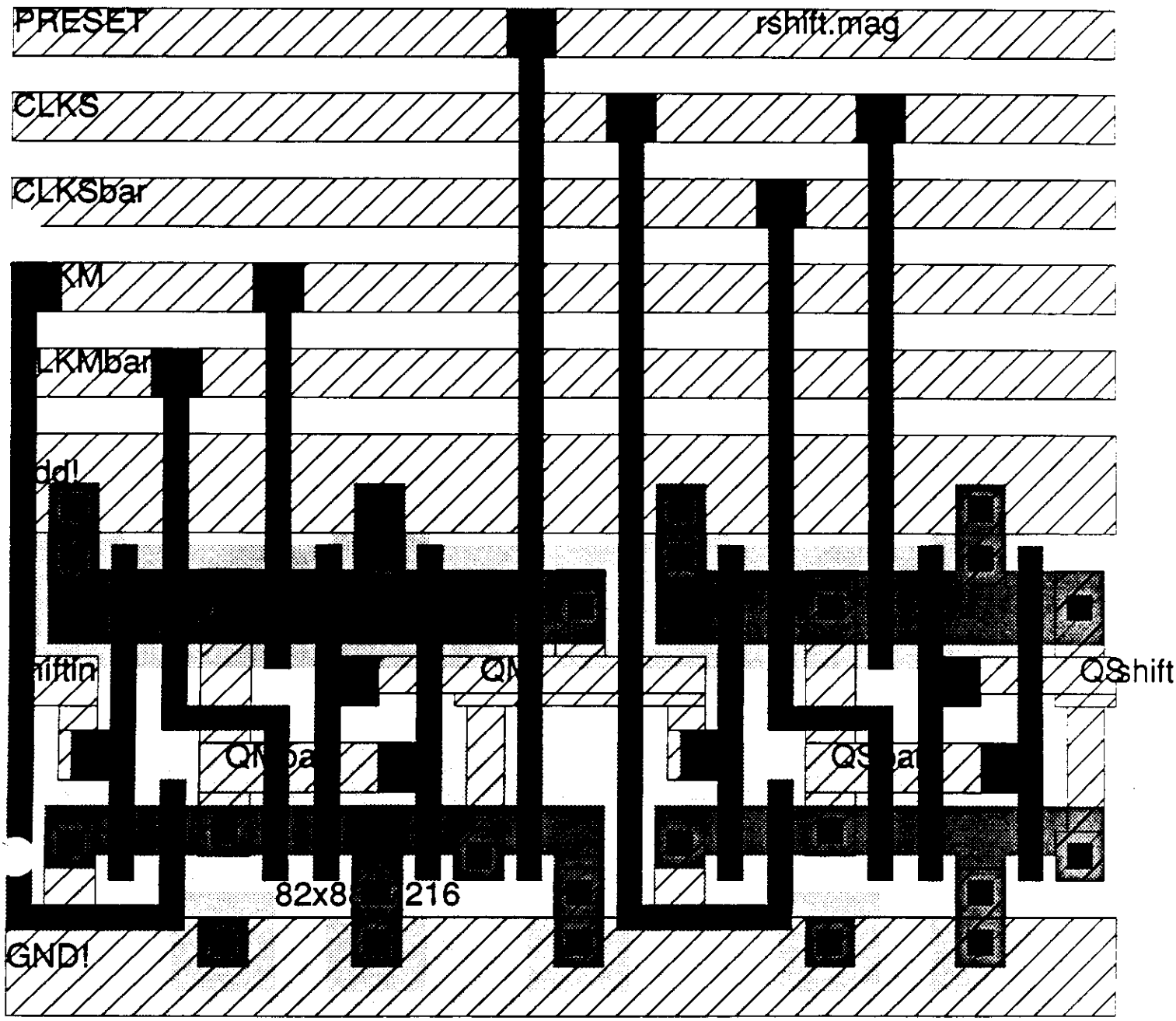

PRESET
rsff.mag
K
LKbar
dd!
D
Q
Qbar
68x48
264
GND!

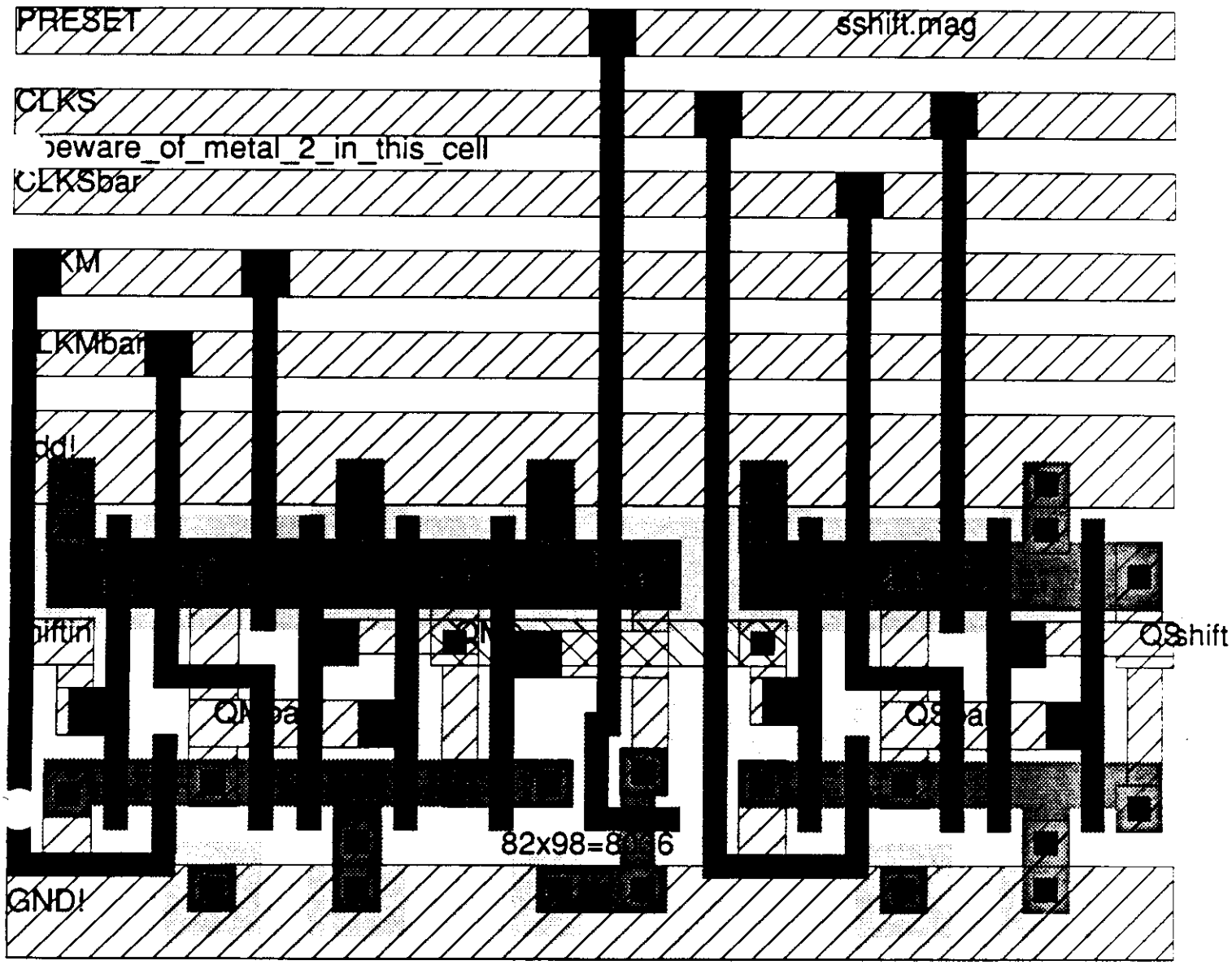

PRESET
sff.mag
K
LKbar
dd!
D
Q
Qbar
68x58=3944
GND!

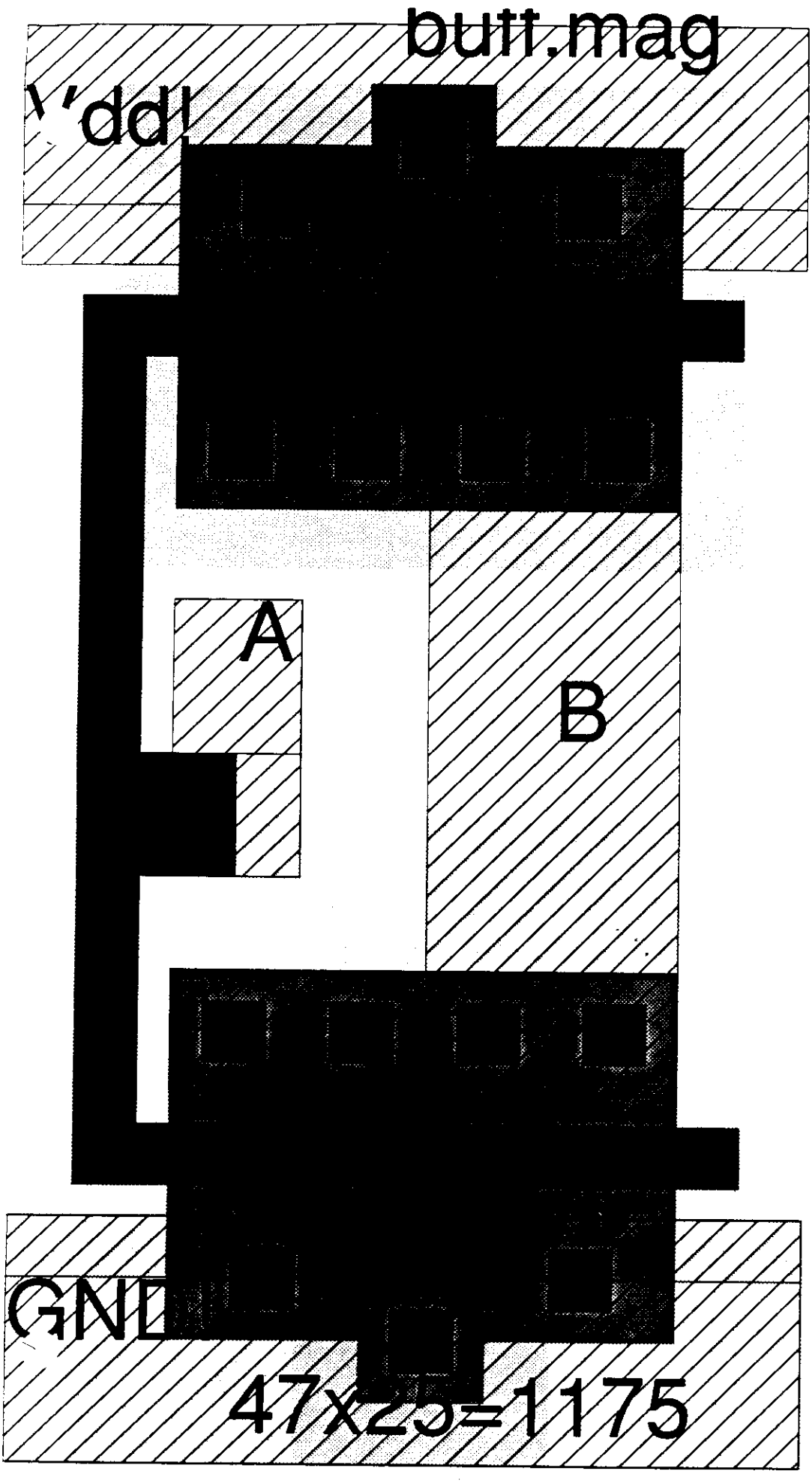

muxff.mag
S
Sbar
CLK
CLK
Vdd!
D0
D1
Q
Qbar
GND!
CLK
CLKbar
103x61=6283
S
Sbar


PRESET
S
rsmuxff.mag
Sbar
CLK
CLI
Vdd!
D1
D0
Q
Qbar
GND!
CLI
CLKbar
110x69=7590
S
Sbar


muxshift.mag
S
Sbar
CLKS
CLKM
Vdd!
D1
GND
117x100=11700
S
Sbar


PRESET
rshift.mag
CLKS
CLKSbar
KM
LKMbar
dd!
QSshift
82x8 216
GND!


PRESET
sshift.mag
CLKS
CLKSbar
Q3shift
82x98=8
GND!


butt.mag
Vdd
A
B
GND
47x25=1175

7.1 Standard Cell Esim Results (SS)

- **Cell-by-Cell Output**

Logic:

inv.mag
>01:A
>10:B

nand2.mag
>0011:A
>0101:B
>1110:C

nand3.mag
>00001111:A
>00110011:B
>01010101:C
>11111110:D

nor2.mag
>0011:A
>0101:B
>1000:C

nor3.mag
>00001111:A
>00110011:B
>01010101:C
>10000000:D

mux.mag
>0000000011111111:S
>0000111100001111:D0
>0011001100110011:D1
>0000111100110011:Q
>1111000011001100:Qbar

Math:

fad.mag
>00001111:A
>00110011:B
>01010101:C
>11101000:CYbar
>10010110:SUMbar

inv.fad.mag
>01:A
>10:B

Flip-flop:

dff.mag
>0011000000110000001100000001100:CLK
>1111111000000001111111111111111:D
>XX1111111100000000111111111111:Q
>XX0000000011111111000000000000:Qbar

rsff.mag
>0000001100000011000000000000000:PRESET
>0011000000110000001100000001100:CLK
>0000001111111111111111100000000:D
>XX0000000011110000111111110000:Q
>XX1111111100001111000000001111:Qbar

sff.mag
>0000001100000011000000000000000:PRESET
>0011000000110000001100000001100:CLK
>1111110000000000000000011111111:D
>XX1111111100001111000000001111:Q
>XX0000000011110000111111110000:Qbar

sff.a1 (anomoly for phi1)
>0011111111000000001111111110000:PRESET
>0011000000110000001100000001100:CLK
>0000000000000011111111111111111:D
>XX1111111100000000111111111111:Q
>XX1100000011111111000000000000:Qbar

sff.a2 (anomoly for phi2)
>0000001111111100000000111111110000:PRESET
>0011000000110000001100000011000000:CLK
>0000000000000011111111111111111111:D
>XX0000111111111111111111111111111111:Q
>XX1111000011000000000000000000000:Qbar

muxff.mag
>0110000001100000011000000110000000110:CLK
>0000000000000111111111111111110000000:S
>1111100000000111111111000000001111111:D0
>0000011111111000000001111111110000000:D1
>X1111111100000000000000001111111111:Q
>X0000000011111111111111110000000000:Qbar

rsmuxff.mag
>0011000000000000000000000000000000000000000:PRESET
>0000001100000011000000110000001100000110:CLK
>0000000000000000001111111111111111110000000:S
>1111111111000000001111111100000000111111:D0
>0000000000111111111000000001111111110000000:D1
>XX0000111111111000000000000000001111111111:Q
>XX1111000000001111111111111111100000000000:Qbar

Shift: muxshift.mag
>0110000001100000011000000110000000110:CLKM
>0000011000000110000001100000001100000:CLKS
>0000000000000111111111111111110000000:S
>1111100000000111111111000000001111111:shiftin0
>0000011111111000000001111111110000000:D1
>X1111111100000000000000001111111111:QM
>X0000000011111111111111110000000000:QMbar
>XXXXX1111111100000000000000001111111:QS
>XXXXX0000000011111111111111110000000:QSbar

rshift.mag
>0110000001100000011000000110000000110:CLKM
>0000011000000110000001100000011000000:CLKS
>0000000000000110000000000000011000000:PRESET
>1111111111111111111110000000011111111:shiftin
>X1111111111110000111111110000000111:QM
>X0000000000001111000000001111111000:QMbar
>XXXXX1111111100000000111111110000000:QS
>XXXXX0000000011111111000000001111111:QSbar

sshift.mag
>0110000001100000011000000110000000110:CLKM
>0000011000000110000001100000011000000:CLKS
>0000000000000110000000000000011000000:PRESET
>0000000000000000000001111111110000000:shiftin
>X0000000000001111000000001111111000:QM
>X1111111111110000111111110000000111:QMbar
>XXXXX0000000011111111000000001111111:QS
>XXXXX1111111100000000111111110000000:QSbar

Buffer:

buff.mag
>01:A
>10:B

7.2 Standard Cell Spice Results (SS)

- Input Vectors

Cell	Case	Output	Rise	Fall	Load
inv.mag	A: 0 -> 1	B		1.25ns	2.50ns
	A: 1 -> 0	B	1.50ns		2.75ns
nand2.mag	AB:11 -> 01	C	2.00ns		3.50ns
	AB:01 -> 11	C		2.50ns	4.00ns
	AB:11 -> 00	C	1.00ns		2.00ns
	AB:00 -> 11	C		2.25ns	4.00ns
nand3.mag	ABC:111 -> 011	D	2.75ns		4.25ns
	ABC:011 -> 111	D		4.75ns	6.75ns
	ABC:111 -> 000	D	1.00ns		2.00ns
	ABC:000 -> 111	D		4.00ns	6.00ns
nor2.mag	AB:10 -> 00	C	3.50ns		5.25ns
	AB:00 -> 10	C		1.75ns	3.00ns
	AB:11 -> 00	C	3.25ns		4.75ns
	AB:00 -> 11	C		1.00ns	2.00ns
nor3.mag	ABC:100 -> 000	D	7.00ns		9.00ns
	ABC:000 -> 100	D		1.50ns	3.00ns
	ABC:111 -> 000	D	5.5ns		7.50ns
	ABC:000 -> 111	D		0.75ns	1.75ns
mux.mag	S: 0 -> 1	Q	6.75ns		8.00ns
		Qbar		4.00ns	6.00ns
	S: 1 -> 0	Q		7.50ns	9.00ns
		Qbar	5.00ns		6.75ns

fad.mag (SUMbar)	ABC:111 -> 011	CYbar			
		SUMbar	5.50ns		7.25ns
	ABC:011 -> 111	CYbar			
		SUMbar		4.50ns	6.75ns
	ABC:100 -> 000	CYbar			
		SUMbar	8.25ns		10.00ns
	ABC:000 -> 100	CYbar			
		SUMbar		3.00ns	4.50ns
	ABC:111 -> 000	CYbar	1.75ns		3.00ns
		SUMbar	6.00ns		7.00ns
	ABC:000 -> 111	CYbar		1.25ns	2.50ns
		SUMbar		2.50ns	5.00ns
(CYbar)	ABC:111 -> 001	CYbar	4.75ns		6.50ns
		SUMbar			
	ABC:001 -> 111	CYbar		1.25ns	2.50ns
		SUMbar			
	ABC:101 -> 001	CYbar	5.00ns		6.75ns
		SUMbar		8.25ns	10.00ns
	ABC:001 -> 101	CYbar		2.75ns	4.50ns
		SUMbar	7.50ns		9.00ns
	ABC:110 -> 010	CYbar	4.75ns		6.25ns
		SUMbar		8.25ns	10.00ns
	ABC:010 -> 110	CYbar		3.00ns	4.75ns
		SUMbar	7.00ns		8.75ns
(CPA)	ABC:000 -> 001	CYbar			
		SUMbar		2.75ns	4.25ns
	ABC:001 -> 000	CYbar			
		SUMbar	7.00ns		9.00ns
	ABC:010 -> 011	CYbar		2.25ns	3.75ns
		SUMbar	7.00ns		8.75ns
	ABC:011 -> 010	CYbar	3.50ns		5.25ns
		SUMbar		7.00ns	8.50ns
	ABC:100 -> 101	CYbar		2.50ns	4.00ns
		SUMbar	7.00ns		8.50ns
	ABC:101 -> 100	CYbar	3.50ns		5.00ns
		SUMbar		6.50ns	8.25ns
	ABC:110 -> 111	CYbar			
		SUMbar		4.00ns	6.25ns
	ABC:111 -> 110	CYbar			
		SUMbar	3.75ns		5.50ns
inv.fad.mag	A: 0 -> 1	B		1.00ns	2.00ns
	A: 1 -> 0	B	1.25ns		2.25ns

dff.mag	D: 1 CLK: 010	Q Qbar	6.00ns		7.25ns 4.75ns
				3.00ns	
	D: 0 CLK: 010	Q Qbar	4.00ns	7.00ns	8.5ns 5.75ns
rsff.mag	D: 1 CLK: 010	Q Qbar	8.25ns		9.75ns 4.75ns
				3.00ns	
	D: 0 CLK: 010	Q Qbar	4.00ns	7.75ns	9.25ns 5.75ns
sff.mag	D: 1 CLK: 010	Q Qbar	6.5ns		8.00ns 4.75ns
				3.00ns	
	D: 0 CLK: 010	Q Qbar	4.00ns	7.75ns	9.75ns 5.75ns
buff.mag	Vtbuff: 1 -> 0	Vbbuff Vend	3.25ns 5.75ns		
	Vtbuff: 0 -> 1	Vbbuff Vend		2.25ns 5.50ns	

7.3 Final Esim Results (SS)

- **Case #1** Verifies the smallest result which can be produced

g1 .. g8 = .10000000, eg = 1110
h1 .. h8 = .10000000, eh = 0110

Unit	Value	Outputs	True	Delta
align	ec	1000	1000	none
	es	0000	0000	none
	xj	0.00000000	0.00000000	none
	yj	0.10000000	0.10000000	none
sos	Z	0.01000000	0.01000000	none
sqr	D	0.10000001	0.10000000	.00000001
div	C	$0.11111111 * 2^{1000}$	0.00000000	none
	S	$0.11111111 * 2^{0000}$	1.00000000	-.00000001

- **Case #2** Verifies the largest result which can be produced

g1 .. g8 = .11111111, eg = 0000
h1 .. h8 = .11111111, eh = 0000

Unit	Value	Outputs	True	Delta
align	ec	0000	0000	none
	es	0000	0000	none
	xj	0.11111111	0.11111111	none
	yj	0.11111111	0.11111111	none
sos	Z	1.11111011	1.11111100	-.00000001
sqr	D	1.01101001	1.01101000	.00000001
div	C	$0.10110101 * 2^{0000.}$	0.10110101	none
	S	$0.10110101 * 2^{0000}$	0.10110101	none

- **Case #3**

g1 .. g8 = .10101010, eg = 1110
h1 .. h8 = .10101010, eh = 0110

Unit	Value	Outputs	True	Delta
align	ec	1000	1000	none
	es	0000	0000	none
	xj	0.00000000	0.00000000	none
	yj	0.10101010	0.10101010	none
sos	Z	0.01110000	0.01110000	none
sqr	D	0.10101010	0.10101010	none
div	C	$0.11111111 * 2^{1000}$	0.00000000	none
	S	$0.11111111 * 2^{0000}$	1.00000000	-.00000001

- **Case #4**

g1 .. g8 = .11111111, eg = 1110
h1 .. h8 = .11111111, eh = 0110

Unit	Value	Outputs	True	Delta
align	ec	1000	1000	none
	es	0000	0000	none
	xj	0.00000000	0.00000000	none
	yj	0.11111111	0.11111111	none
sos	Z	0.11111101	0.11111110	-.00000001
sqr	D	0.11111111	0.11111111	none
div	C	1.00000000 * 21000	0.00000000	.00000001
	S	1.00000000 * 20000	1.00000000	none

- **Case #5**

g1 .. g8 = .10101010, eg = 0000
h1 .. h8 = .10101010, eh = 0000

Unit	Value	Outputs	True	Delta
align	ec	0000	0000	none
	es	0000	0000	none
	xj	0.10101010	0.10101010	none
	yj	0.10101010	0.10101010	none
sos	Z	0.11100001	0.11100001	none
sqr	D	0.11110001	0.11110000	.00000001
div	C	0.10110101 * 20000	0.10110101	none
	S	0.10110101 * 20000	0.10110101	none

- **Case #6**

g1 .. g8 = .10101010, eg = 0110
h1 .. h8 = .10101010, eh = 0010

Unit	Value	Outputs	True	Delta
align	ec	0000	0000	none
	es	1100	1100	none
	xj	0.10101010	0.10101010	none
	yj	0.00001010	0.00001010	none
sos	Z	0.01110001	0.01110001	none
sqr	D	0.10101011	0.10101010	.00000001
div	C	0.11111111 * 20000	0.11111111	none
	S	0.11111111 * 21100	0.00001111	none

- **Case #7**

g1 .. g8 = .11111111, eg = 0110
h1 .. h8 = .11111111, eh = 0010

Unit	Value	Outputs	True	Delta
align	ec	0000	0000	none
	es	1100	1100	none
	xj	0.11111111	0.11111111	none
	yj	0.00001111	0.00001111	none
sos	Z	0.11111110	0.11111110	none
sqr	D	0.11111111	0.11111111	none
div	C	$0.11111111 * 2^{0000}$	0.11111111	none
	S	$0.11111111 * 2^{1100}$	0.00001111	none

- **Case #8**

g1 .. g8 = .11111111, eg = 0000
h1 .. h8 = .10101010, eh = 0000

Unit	Value	Outputs	True	Delta
align	ec	0000	0000	none
	es	0000	0000	none
	xj	0.11111111	0.11111111	none
	yj	0.10101010	0.10101010	none
sos	Z	1.01101110	1.01101110	none
sqr	D	1.00110011	1.00110010	.00000001
div	C	$0.11010101 * 2^{0000}$	0.11010101	none
	S	$0.10001110 * 2^{0000}$	0.10001110	none

- **Case #9**

g1 .. g8 = .11001100, eg = 1110
h1 .. h8 = .10101010, eh = 1101

Unit	Value	Outputs	True	Delta
align	ec	0000	0000	none
	es	1111	1111	none
div	C	$0.11101100 * 2^{0000}$	0.11101100	none
	S	$0.11000100 * 2^{1111}$	0.01100010	none

- **Case #10**

g1 .. g8 = .11101010, eg = 1111
h1 .. h8 = .10111111, eh = 0000

Unit	Value	Outputs	True	Delta
align	ec	1111	1111	none
	es	0000	0000	none
div	C	1.00001011 * 21111	0.10000101	none
	S	0.11011010 * 20000	0.11011010	none

- **Case #11**

g1 .. g8 = .11011001, eg = 0010
h1 .. h8 = .10101010, eh = 0011

Unit	Value	Outputs	True	Delta
align	ec	1111	1111	none
	es	0000	0000	none
div	C	1.00010011 * 21111	0.10001001	none
	S	0.11010111 * 20000	0.11011000	-.00000001

- **Case #12**

g1 .. g8 = .10001111, eg = 1000
h1 .. h8 = .11101010, eh = 1001

Unit	Value	Outputs	True	Delta
align	ec	1111	1111	none
	es	0000	0000	none
div	C	0.10010101 * 21111	0.01001010	none
	S	0.11110100 * 20000	0.11110100	none

7.4 Final Spice Results (SS)

Critical Paths

Unit	Phase	Cells	Description
Alignment	Ø1 -> Ø2	1 buff 1 muxff - Q output 4-bit cpa - 3 CYbar (A/B simultaneous from Q/Qbar of muxff, C fixed at right end), 1 SUMbar (A/B simultaneous from same, C = carry in from previous)	initial computation of **ediff** = (**eg** - **eh**); all subsequent computations during countup/countdown to 0
	Ø2 -> Ø1	1 buff 1 dff 1 nor2 1 nand2 1 nand3 1 nand2 1 inv	0-detection and generation of clock enables which are ANDed with Ø2 to produce **ediffclk/etempclk**
Sum-of-Squares	Ø1 -> Ø2	1 buff 1 rsff - Q output 2-bit cpa - 1 CYbar (A/B simultaneous from Q/Qbar of rsff, C fixed at right end), 1 SUMbar (A/B simultaneous from same, C = carry in from previous)	2-1 reduction of *W-latch.ps*, *W-latch.sc* forming **z(j)**
	Ø2 -> Ø1	1 buff 1 nand2 1 inv 1 rsff - Q output 2 nor2 2 fad - 1 SUMbar (A fixed, B/C simultaneous from nor2), 1 SUMbar (A fixed, B from CYbar, C from SUMbar of above fad)	generation of **X[j]** and **Y[j]**, formation of **Qx(j)** and **Qy(j)**, and 4-2 reduction to form **W[j]** above *W-latch.ps/ W-latch.sc*
Square-Root	Ø1 -> Ø2	1 buff 1 nor2 1 inv 1 rsmuxff - Q output 1 nor2 1 mux 1 nor3 1 nor2 2 fad - 1 SUMbar (A/B fixed, C from nor2), 1 SUMbar (A fixed, B from CYbar, C from SUMbar of above fad)	generation of **D[j]** and **Dstar[j]**, formation of **Qd**, and 4-2 reduction to form **R[j]** in *R-latch.ps/R-latch.sc*

Unit	Phase	Cells	Description
Square-Root (continued)	Ø2 -> Ø1	1 buff 1 rsff - Q output 2 buff 7-bit cpa - 1 CYbar (A/B from rsff, C from sos), 1 CYbar (A fixed, B from CYbar previous/above fad, C from SUMbar of above fad), 4 CYbar (A/B fixed, C = carry in at right end), 1 SUMbar (A/B fixed, C = carry in) 1 inv.fad 1 nand2 1 nor2	3-2 reduction of *R-latch.ps*, *R-latch.sc*, and *z-latch*, followed by 2-1 reduction and selection to determine **d(j)**
Division	Ø1 -> Ø2	1 buff 1 rsff - Qbar output 2 buff 1 mux 1 nand2 1 fad (A/B fixed, C from nand2)	Latching of *s-latch*, feedback routing to *Qd-multiplex* and its circuitry, followed by 3-2 reduction of *W-latch.ps*, *W-latch.sc*, and **Qd** to form **P[j]** in *P-latch.ps*/ *P-latch.sc*
	Ø2 -> Ø1	1 buff 1 rsff - Q output 2 buff 6-bit cpa - 1 CYbar (A/B from FF's, C = carry in), 3 CYbar (A fixed, B = CYbar from previous/above fad, C = SUMbar from above fad), 1 SUMbar (same). Note that the leftmost bit position is not in the critical path! 1 inv.fad 1 nor2 1 nand2 1 nand2	3-2 reduction of *P-latch.ps*, *P-latch.sc*, and **Qsign**, followed by 2-1 reduction and selection to determine **s(j)**

• Cycle Time Estimation (Derived from Spice Simulation of Critical Paths)

Cycle Time = Time required by the critical path in the Ø1 -> Ø2 half-cycle, +
Time required by the critical path in the Ø2 -> Ø1 half-cycle

= Time required by the Square-Root Unit in the Ø1 -> Ø2 half-cycle, +
Time required by the Square-Root Unit in the Ø2 -> Ø1 half-cycle

= 50 nanoseconds + 55 nanoseconds

= 105 nanoseconds, which corresponds to 1000/105 or 9.5 megahertz