

▪ Digital Circuits ▪

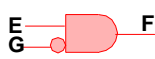
Digital Circuits

People don't understand double negatives

Asserted-Low (Low-True) Signals

Low-true signals implement their name when low.
Their names often end in "(L)" or "_N"

High-true signals implement their name when high.
Their names end in "(H)" or nothing.



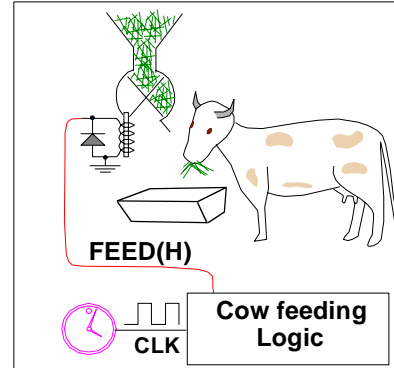
E,G and F have no physical meaning. They are not asserted-high or asserted-low

Example: Cow feeding circuit

FEED(H) is asserted high (high-true).

It implements its name when high.

CLK is not asserted either high or low.



Common Digital Circuits

Asserted-Low Signals

"The electrons have no knowledge of what meanings we attach to these signals." (Richard Hamming)

Asserted-Low (Low-True) Signals

Definition

These are signals which carry out the function of their name, when low.

Examples

FLASH_LED(L), ERROR(L) used to be preferred for schematic design.

CLEAR_LATCH_N, CHIP_SELECT_N for Verilog code which cannot have brackets.

b_N(7), b_N(6), . . . b_N(0) as bits of a binary number.

Warning

The notation "(L)" or "_N" can get very cumbersome.







It is very useful for work involving block diagram interconnections or maintenance.

It is very messy to do Boolean algebra on such names. Instead convert the signals to normal high-true signals to do thinking.

It is very easy to use DeMorgan's theorem to transform them back at the end.

▪ Digital Circuits ▪

Asserted-Low Signals (cont.)

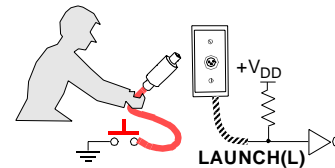
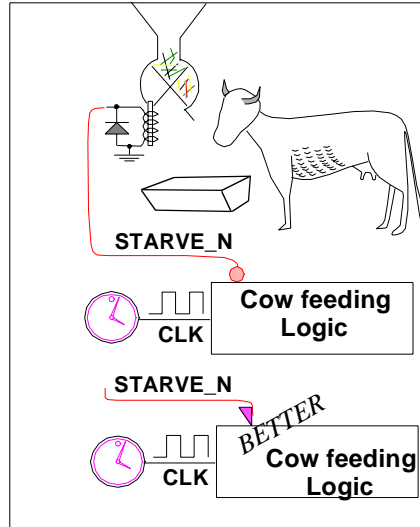
- Change the signal name to STARVE(L)
- Convention places an \circ where a low-true signal enters or leaves a circuit block.
- This only reinforces the low-true naming. **The physical circuit is unchanged!**
- IEEE convention, used by TI and Signetics, uses a  rather than   rather than   
- Avoids temptation to insert a real inverter for a mental one.

Most Board Inputs are Asserted-Low

In TTL logic a disconnected input went high.
Pulling an input plug would assert high-true inputs.
This was usually not wanted.
(suppose the input fired missiles)
Thus TTL inputs were made asserted low

CMOS board designers often still use pull-ups to V_{DD} ,
They then design for asserted-low
to avoid the plug pulling problem..
Such is the strength of tradition

Convert After Thinking is Done






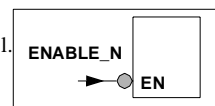
Digital Circuits ▪

Asserted Low Conventions

Digital Circuits

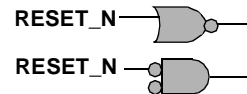
Asserted Low Conventions

- Any asserted low signal entering a block (not a gate) type symbol has an inverting symbol  or better  where it enters the block.
- Then mentally, all signals are asserted high inside the block as a symbol. When $ENABLE_N=0$, EN inside the box symbol is true, and the chip is enabled.
- Since the conversion $ENABLE_N=0$, to an asserted-high EN inside the block is only mental, the  is a better symbol, but it is so seldom used it may cause more confusion than it saves.
- Asserted-low output signals are analogous.



The is not used for gate-level schematics

On a schematic, a low-true signal may have to enter a gate without a \circ or preferably one may be able to DeMorgan the gate to get an input \circ .



▪ Common Circuit Blocks ▪

Common Circuit Blocks

Combinational Circuits

1. Multiplexer (MUX)
 - a. Digital
 - b. Analog
2. DeMUX, Decoder
3. Priority Encoder
4. Barrel Rotators and Shifters
5. Adders
6. Incrementers
7. Subtracters and Two's Complement

Descriptions Given For the Above Circuits

1. Verilog
2. Circuit



Common Circuit Blocks ▪

Common Circuit Blocks

Common Circuit Blocks

These blocks will be discussed in reasonable detail.

▪ Multiplexers ▪

Multiplexers

Digital 2-Input MUX Application

Symbol

IEEE symbol:
 $S=0$ connects $b \rightarrow y$
 $S=1$ connects $a \rightarrow y$

Circuit

Complex Gate Circuit

2-input

Select Inputs S	Virtual signal	Connects
0	1	$b \rightarrow y$
1	1	$a \rightarrow y$

Often shortened to G1

Complex Gate Details

Dig Cir p. 53

© John Knight
Revised; October 9, 2003

Slide 27

Multiplexers ▪

Common Circuit Blocks

IEEE Symbols for a MUX

2-Input MUX

The numbers represent internal “virtual” signals. One could have used say 7 instead of 1.

A G1 stands for “Gate” (AND) input present signal S with a virtual signal 1 from some other point in the circuit.

A $G\bar{1}$ means Gate with signal $\bar{1}$ when this line is low.

The $G1\bar{1}$ means Gate with signal 1 when this line is high and with $\bar{1}$ when it is low.

It is usual to assume the $\bar{1}$ for a MUX and just write G1.

The word MUX at the top shows the circuit combines these gated inputs like a MUX.

The IEEE standard does not retain the shaped gates. They put everything in rectangles.

Carleton University
Digital Circuits p. 54

© John Knight
Revised; October 9, 2003

Comment on Slide 27

▪ Multiplexers ▪

Digital 2-Input MUX (cont.)

Verilog

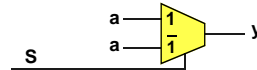
Behavioural Description

Uses *Conditional operator*

If *Cond ? True: False;*

```
//Verilog MUX
//Using Structural If
m
wire y;           //output
wire a, b, S;    //inputs: control

assign y = (S) ? a: b;
```



Multiplexers ▪

2-Input Mux

2-Input Mux

A 2-input MUX is perhaps easiest to write this way.

▪ Multiplexers ▪

Digital 2-Input MUX

Verilog

Structural Description

Built from gates

- Wires are for internal connections.
- Some simple wires are defaults so a declaration is optional
- Gates do not need individual names. We won't refer to them later.
- **S_** is a common way to write S.

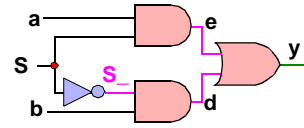
Verilog

Behavioural Description

- Easy to follow.
- Synthesizer will build easily

```
//Verilog MUX
//Built from "primitive" gates
```

```
wire a, b, S, d, e, S_;
wire y;
not (S_, S);
and (e, a, S);
and (d, b, S_);
or (y, e, d);
```



```
//Verilog MUX
//Built from case
```

```
wire a, b, S;
reg y;
always @(a or b)
begin: mux_defn
    case(S)
        0: y=b;
        1: y=a;
    endcase
end
```

Multiplexers ▪

Verilog

Verilog

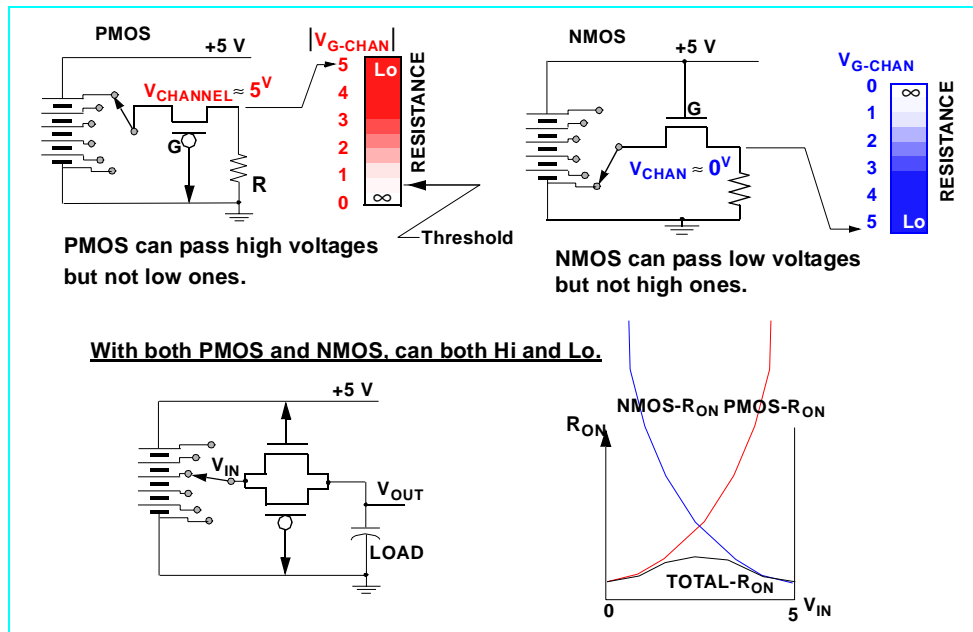
Wires names and gate names

- One-bit wires connecting component instances together are considered implicit wires. They do not need to be declared. However implicit wires are only one bit wide. Bus connection between two components must be declared as a wire.
- Note that wires connecting components in *continuous assignments* **must** be declared. Only those connected directly to ports (the module input/output signals) are defaults. Thus `assign y = (S) ? a : b;` (Slide 31) used only defaults. If one had said `assign f = (S) ? a : b;`
`assign y = ~f;` we would have had to declare `f` a wire. Note `~f;` means `f`.
- Before, we named each gate instance. `nand NANDY(OUT, IN1, IN2);` Gates do not need individual names if one will not refer to them individually later. Thus we can say `and (e, a, S);`

▪ Multiplexers ▪

Transmission Gates; a Switch of Transistors

For Low Channel Resistance, One Needs a Large Gate-to-Channel Voltage

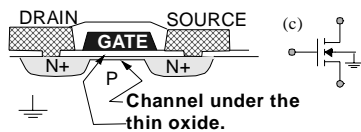


Multiplexers ▪

Verilog

Conducting Transistors

In order for a transistor channel to conduct well, the gate voltage must be much higher than the channel voltage

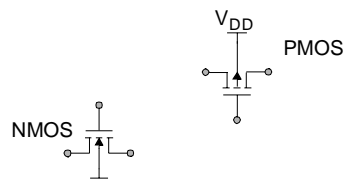


The gate voltage attracts carriers to form a thin layer under the oxide. They make a channel between N⁺ and N⁺. A high gate voltage attracts more carriers and gives lower on resistance.

Substrate Connection For Transmission Gates

The substrate must be connected to the lowest voltage, probably ground, for NMOS and the highest voltage, probably V_{DD} for PMOS.

Otherwise the channel-substrate diode becomes forward biased.

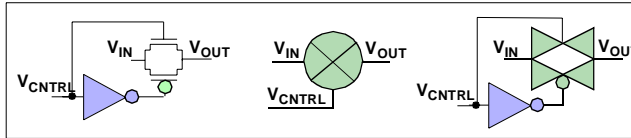


▪ Multiplexers ▪

Analog MUX

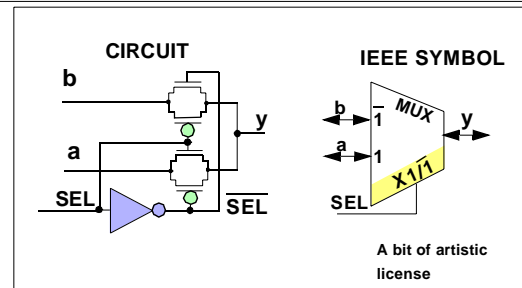
Transmission Gate

- The gate and two symbols



Transmission Gate Mux

- Very small 6 transistor circuit.
- Does not invert.
- Transmission gates add extra resistance and hence delay.
- Parallel transistors are hard to test. The circuit still partially works if one is open.

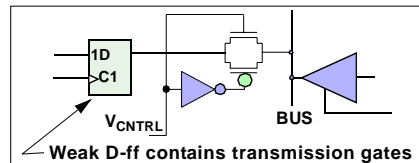


Analog Signals

- The circuit will switch analog signals, but - - -
If *a* and *b* go negative, your channel-substrate junction will become forward biased.

MUX is Bidirectional

- Do not drive buses through a transmission gate. A high powered bus may feed backwards through the gate and overpower weaker logic.



Multiplexers ▪

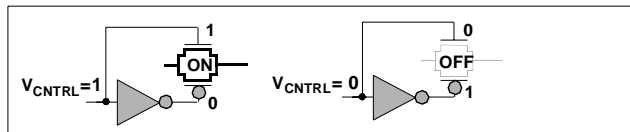
Verilog

Analog MUX

The Transmission Gate

On and Off

The Analog MUX



The IEEE Symbol

- The letter G on the digital MUX is replaced by X. G stood for gated, indicated an ANDing of two leads. X is a bidirectional gate. The analog MUX can work in either direction.
- The double arrows \longleftrightarrow show the signals can travel either way.

Advantages

- This MUX is very simple, replacing 10 or 12 transistors in the gate built MUX, with 6 or 8. The larger number in each case is for an inverter at the output.
- The MUX does not invert. However most MUX standard cells will add an inverter an output buffer..

Disadvantages

- This MUX adds an extra resistance in the signal path. Remember the delay has a component that increases quadratically with the number of series transistors (Comment on Slide 14). Check if the gates driving *a* or *b* have three or more series gates already.
- Suppose one of the parallel transistors in a transmission gate is no good. The other one will still conduct albeit with a higher resistance and a possible threshold drop. Capacitances may be half charged and give erratic results.¹ Many circuit will work but more slowly. Testing for slow response, *delay testing*, is much harder than normal static testing. Testing for static power supply current, *I_{DDQ} testing*, can sometimes find partial charges on capacitors, since these may allow a direct path from V_{DD} to ground. This is changing the present attitude toward transmission gates.

¹ Thomas DeMassa and Zack Ciccone, *Digital Integrated Circuits*, John Wiley, 1996, p.432.

▪ Multiplexers ▪

4-Input MUX

The IEEE symbol

decodes the 2-bit number S_1S_0 into 4 virtual control signals.

The Circuit

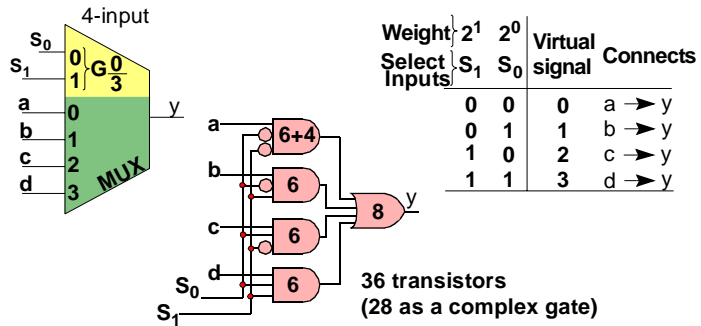
36 transistor CMOS circuit.

Verilog

| = bitwise OR
& = bitwise AND
~ = bitwise NOT

Consider:

```
assign y =
    S1 ? (S0 ? d : c)
    : (S0 ? b : a);
```



```
// Verilog 4-Input MUX
//
wire a, b, c, d;
wire S1, S0, y;

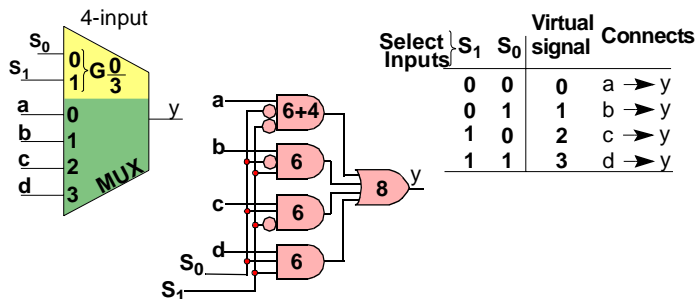
assign y = (~S1 & ~S0 & a) | // virtual sig 0
           (~S1 & S0 & b) | // virtual sig 1
           ( S1 & ~S0 & c) | // virtual sig 2
           ( S1 & S0 & d); // virtual sig 3

// Good for mental exercise
```

Multiplexers ▪

Verilog 4-Input Mux

Verilog 4-Input Mux



Verilog

The integer 2 is translated into 10
{S1,S0} says concatenate bits together.
%d says display {S1,S0} as a decimal.

```
// Verilog 4-Input MUX
wire a, b, c, d;
wire S1, S0;
reg y;

always @ (a or b or c or d or S1 or S0);
case ({S1,S0})
0: y=a;
1: y=b;
2: y=c;
3: y=d;
default: $display
("Bad input %d to 4-input mux", {S1,S0})
endcase
```

Demultiplexers, Decoders

Demultiplexers, Decoders

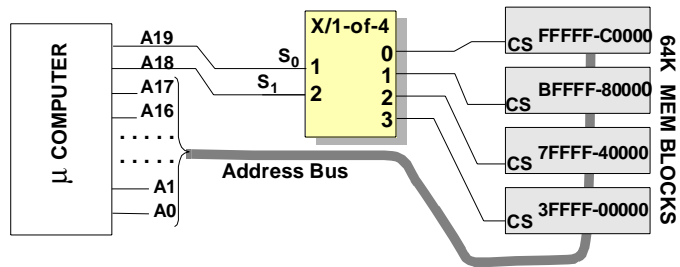
Decoder

Application

Decode 2-bit binary into 1-out-of-4.

Select one of the 64K memory blocks.

CS = chip select



Symbol

IEEE symbols:

The DeMUX and Decode are exactly the same.

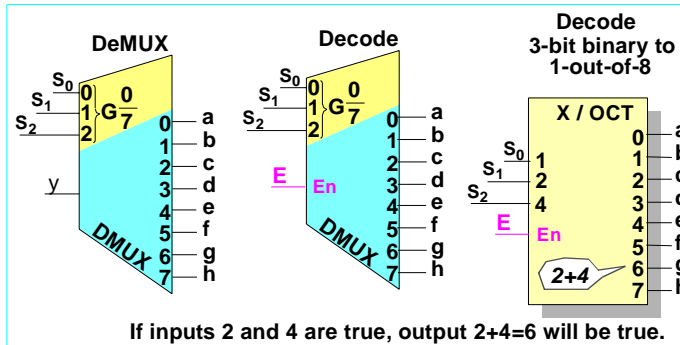
The decode thinks of the y input as an enable.

if $y=0$ all outputs=0

Other decoder symbol

One code in
Another code out.

Σ (nonzero inputs) tells which outputs are true.



Demultiplexers, Decoders

The DeMUX/Decoder

The DeMUX/Decoder

Decoding the memory bank

Only one of the 4 blocks is selected at any given time.

The 18 low-order address lines go to all blocks, but only one is enabled.

IEEE symbols

The official convention does not shape the gates.

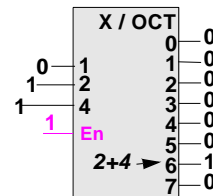
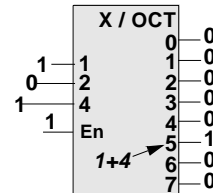
X/ means some code usually binary.

The output lines that are true are calculated from the sum of the numbers of the input lines.

If input "1"=1, "2"=0, "4"=1, Then output line "1"+"4"= 5 would be 1.

If input "1"=0, "2"=1, "4"=1, Then output line "2"+"4"= 6 would be 1.

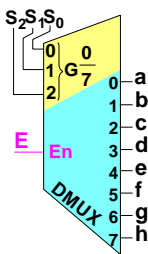
The Enable line (En) must be 1 or all outputs are 0.



▪ Demultiplexers, Decoders ▪

Verilog Decoder

Procedural Verilog for a 3-to-8 Decoder



```

wire S2, S1, S0, E;
reg a, b, c, d, e, f, g, h; //they hold their values like variables in C

always @(S2 or S1 or S0 or E) //execute code below if any of S2,S1,S0 or E change
begin: 3to8decode //Start of procedure code
    a=0; b=0; c=0; d=0; e=0; f=0; g=0; h=0;
    case ({S2,S1,S0}) //Concatenate the 3 bits into a number used below.
        3'd0 : a=E; // When S2S1S0 = 0 decimal (000 binary), make a = E.
        3'd1 : b=E; // 3'd1; a 3-bit number, value in decimal is 1.
        3'd2 : c=E; // When S2S1S0 = 2 decimal (010 binary), make c = E.
        3'd3 : d=E;
        3'd4 : e=E;
        3'd5 : f=E;
        3'd6 : g=E;
        3'd7 : h=E;
        default: $display("invalid control code");
    endcase
end //End of procedure code
endmodule

```

Priority Encoders

Priority Encoders

Priority Encoder

Application

Receive one-bit interrupt signal.
Encode into 3-bit binary.

The computer receives the number of the highest priority interrupt.

Symbol

The HPRI/BIN at the top tells what the block does.

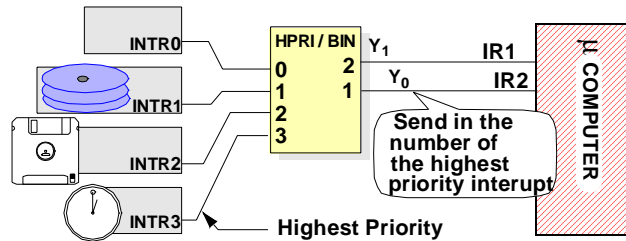
The numbers help one order the pins.

The table shows what really happens.

Gotcha

The output is the same for:
0000001 (a=1) and
0000000 (no input).

To tell the difference, add
"Any Input" signal



		INPUT								OUTPUT			
		h	g	f	e	d	c	b	a	Y ₂	Y ₁	Y ₀	A _y
a	0	1	X	X	X	X	X	X	X	1	1	1	1
b	1	0	1	X	X	X	X	X	X	1	1	0	1
c	2	0	0	1	X	X	X	X	X	1	0	1	1
d	3	0	0	0	1	X	X	X	X	1	0	0	1
e	4	0	0	0	0	1	X	X	X	0	1	0	1
f	5	0	0	0	0	0	1	X	X	0	1	1	1
g	6	0	0	0	0	0	0	1	X	0	0	1	1
h	7	0	0	0	0	0	0	0	1	0	0	0	1
	0/1/2/3/4/5/6/7	0	0	0	0	0	0	0	0	0	0	0	0

Priority Encoders

Priority Encoder

Priority Encoder

To Respond to All Four Interrupts

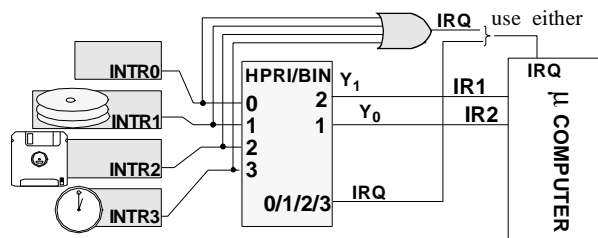
INTR0 makes IR1=0=IR0. It does not give out a signal. This is the same as no interrupt!

To respond to all interrupts, generate IRQ (interrupt request) for any interrupt signal.

Then use IR1 and IR2 to show which one.

One can do this with an OR gate.

Priority encoder macros usually include it. Here it is the 0/1/2 output.



Dependency notation

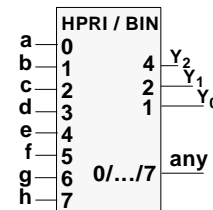
Above, $Y_1 * 2 + Y_0 * 1 = 2 =$ number of the highest priority input.

Thus with interrupts pending on 0, 1 and 2, Y_1, Y_0 would send 10 to the computer.

The 0/1/2/3 means any of the inputs 0, 1, 2, 3 can generate the IRQ signal. It is equivalent to the OR gate shown.

The 8-input priority encoder.

On the 8-input encoder, the 'any' output is written 0/1/2/3/4/5/6/7/ abbreviated¹ here as 0/.../7



¹The standard abbreviation is 0...7

▪ Priority Encoders ▪

Verilog Priority Encoder/ Use of Casex

Procedural Verilog for an 8-line to 3-Bit-Binary Encoder

```

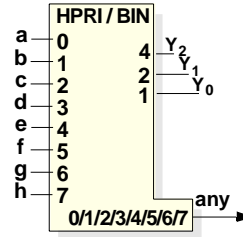
reg    any
reg    [2:0] Y;    //reg hold their values like variables in C++

always @(h or g or f or e or d or c or b or a) //trigger variables for code.
    begin          //Start of procedure code
        casez ({h,g,f,e,d,c,b,a}) //Casez preferred over casex.
            8'b1zzzzzzz : Y= 3'b111; // z will match 0 or 1 or x ( undefined).
            8'b01zzzzzzz : Y= 3'b110; // But x will not match 0 or 1 with casez
            8'b001zzzzzz : Y= 3'b101;
            8'b0001zzzzz : Y= 3'b100;
            8'b00001zzzz : Y= 3'b011;
            8'b000001zzz : Y= 3'b010;
            8'b0000001zz : Y= 3'b001;
            8'b00000001z : Y= 3'b000;
            default      : Y= 3'b000;
        endcase

        any = a | b | c | d | e | f | g | h;

    end          //End of procedure code

```



Priority Encoders ▪

Verilog Priority Encoder

Verilog Priority Encoder

Case, Casex and Casez

Case

The values stored in the `control_var` and those in each statement below it, must agree “1 for 1,” “0 for 0,” “x for x,” and “z for z.”

Casez¹

If `casez`, is used instead of `case`, “z” or “?” will match anything, i.e. “1, 0, x, or z,” in both directions. The important point is a “1” or a “0” WILL NOT match an “x”.

Thus `control_var = 3'bz01` will match the case `3'b101` on line 2 above.

Also `control_var = 3'bx01` will NOT match the case `3'b101`.

Casex

If `casex`, is used instead of

`case`, “x,” “z” will match anything, i.e. “1, 0, x, or z,” in both directions.

Several cases, one action

Place several cases on one line `3'b101, 3'b010: . . .` Separate the cases by commas.

Using “?”

The “?” is equivalent to Z in list of cases inside case blocks. It improves readability.

1. PROBLEM

Add an “Any Input” output line to the Verilog priority encoder description that uses `casez`.

```

1 casez(control_varb)
2 3'b101, 3'b010 : . . .
3 3'b1z1 : . . .
4 3'b0zx : . . .
5 3'b1?? : . . .
6 default : . . .
7 end case

```

¹ Using `casez` instead of `casex` for the priority encoder, and using statements like `3'b1zz` for matching, makes no difference for synthesis, but an accidental X during simulation will never give an unintended for match for a 1 or a 0.

▪ Programmable Shifters ▪

Programmable Shifters

Barrel Rotators (Shifters)

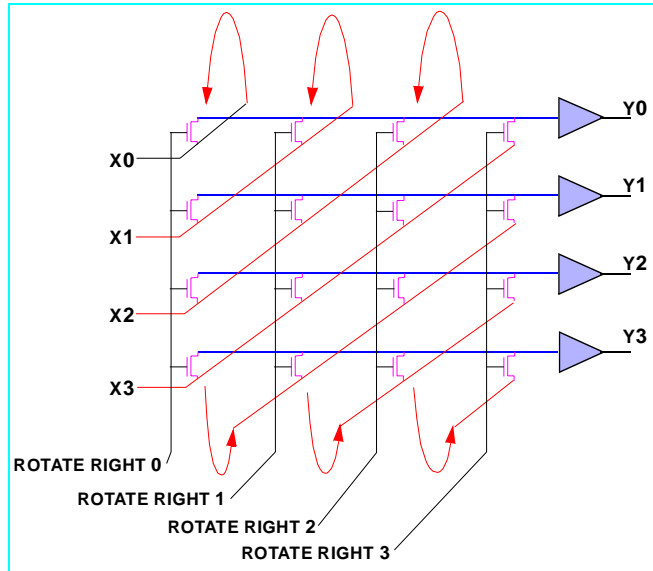
Applications

Multiply/Divide by 2, 2^2 , 2^3 ...

Normalizing numbers especially integer to float pt. conversion.

Not CMOS

- First thought: transmission gates.
- Big shifter takes a lot of area. Speed dominated by wiring capacitance.
- To reduce area have only one well.
⇒ no PMOS.
- NMOS can only pull up to $V_{DD} - V_{THRESH}$
- Slow pull up
Faster pull down



Programmable Shifters ▪

The Barrel Rotator

The Barrel Rotator

NMOS or CMOS

A 32 bit barrel has 32^2 cells. There is little room for the extra wells (tubs) required for PMOS and NMOS¹. Usually only NMOS is used.

The NMOS gates can only pull up $V_{DD} - V_{THRESH}$ and some speed or leakage power is sacrificed.

Speed

There is only one series transistor in any path; this helps speed. Each data line will see n drain capacitances which will reduce speed. However in modern processes the wire capacitance is likely larger than the drain capacitances. This is an important reason for using only NMOS to reduce the area.

Also zeros will go through NMOS transistors better than ones.

Verilog Synthesis

Synthesizers will not produce a very good barrel shifter, since they would try to synthesize gate level CMOS. Try to find a shifter already laid out.

Sign Extension

A negative 2's complement number will have one as its most significant bit. If one shifts to divide by 2, 4 ..., one must move a 1 in from the right for negative numbers and a 0 in for positive ones. This is called *sign extension*.

Thus 100 right shifted 1 position is 110.

A bigger number; 100100100 right shifted 3 positions is 111100100

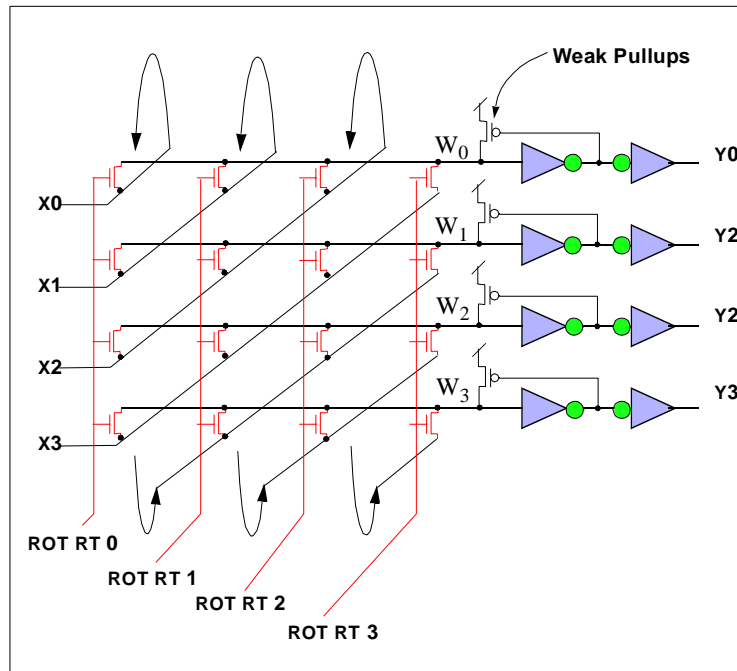
2's complement	
+3	011
+2	010
+1	001
0	000
-1	111
-2	110
-3	101
-4	100

¹ Wayne Wolf, *Modern VLSI Design, A Systems Approach*, Prentice Hall, 1994, pp. 221-223.

▪ Programmable Shifters ▪

Barrel Rotator With Weak Pullups

- Weak pullups overcome NMOS poor pullup properties.
- Shifter is still a reasonable size.



Programmable Shifters ▪

The Barrel Rotator

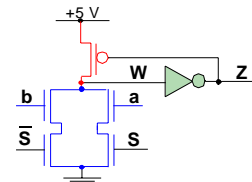
Pull-Ups for the Barrel Shifter.

- The PMOS transistors will pull-up the barrel shifter NMOS transistors. When a W_k lines rise above $V_{DD}/2$ the PMOS transistors will turn on. This raises the W_k line, which lowers the PMOS gate, which raises the W_k line, which ...
- When an X input was high, and then goes low, the PMOS will still be initially pulling up. The PMOS transistor must be made weak enough that the NMOS transistor can pull point W below half way when both transistors are on.
- The complete shifter is much smaller than a complete CMOS implementation would be.

Why does not one make all CMOS that way?

The W_k lines in the barrel shifter are never floating. They are fed by pass transistors which either pull them low, or pull them high. Of course the high is always a threshold or more below V_{DD} .

If CMOS gates had only a single transistor PMOS section, it could not pull up unless the W line was partially high already. If the W line was pulled low, when the NMOS transistors turned off, the W lines would stay low, held by the charge on the stray capacitance, and the Z line would stay high.



Go high and stay high circuit

▪ Programmable Shifters ▪

Verilog rotator/shifters

Synthesis

Synthesis will not give the NMOS circuits described.
You must have a laid-out circuit which can be treated as a special module.

If R is a variable, the synthesizer may not like
 $X \gg R$ or $\{X[R:0], X[N:R-1]\}$.

Code For Simulation

```

wire [1:0] R\\Rotate R bits;
wire [3:0] X,Z;
assign Z =
    {X[R:0],X[3:R-1]};
    
```

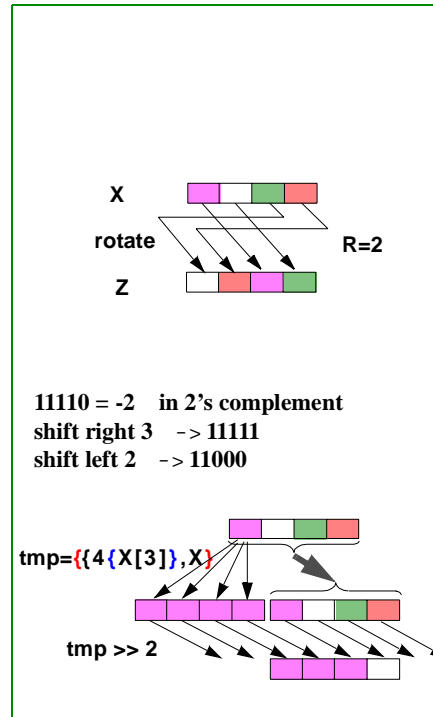
Shifting 2's complement numbers

If the leading bit is "1", the number is negative.
The leading bit must be maintained during shifts.

Use *replication* for sign extension.

```

// Sign extension.
assign tmp={ {4{X[3]},X }
//{4{X[3]}} = {X[3],X[3],X[3]X[3]}
assign Z = tmp >> Sh;
    
```



11110 = -2 in 2's complement
shift right 3 -> 1111
shift left 2 -> 11000

Programmable Shifters ▪

Barrel Rotator Shifter (cont.)

Barrel Rotator Shifter (cont.)

Verilog

Because of the size problem, one does not synthesize a large barrel shifter.

One would lay out the NMOS circuit rather than synthesize it from gates.

The Shift Operators

$V = W \ll S$ means shift W right Y places. V will be zero filled on the right.

$Z = X \gg S$ means shift X right Y places. Z will be zero filled on the left unless it is an integer or real.

Sign Extension

A negative 2's complement number will have one as its most significant bit. If one shifts to divide by 2, 4, . . . , one must move a 1 in from the left for negative numbers and a 0 in for positive ones. This is called *sign extension*.

Thus (-4) 100 right shifted 1 position, with sign extension, is 110 (-2).

A bigger number; (-220) 100100100 right shifted 3 positions is 111100100 (-28).

- Verilog right shifts do a zero fill for variables with a user defined bit width, like 7'd33.
- Right shifts do sign extension for integers or real, but not reg or wire. An integer is the default 32 or 64 bit word of the computer running Verilog.

three-bit 2's complement
+3 011
+2 010
+1 001
0 000
-1 111
-2 110
-3 101
-4 100

Concatenation and Replication

{A,B} concatenates A and B. Thus {2'b11, 3'b001} \Rightarrow {5'b11001}

A Concatenation of many copies of the same thing can be written, {5{X}}, instead of {X,X,X.X.X} the replication number, 5 above, must be a constant.

▪ Programmable Shifters ▪

The Logarithmic Barrel Shifter

cumulative rotate 2	cumulative rotate 1	Rotate
0	0	0
0	1	1
1	0	2
1	1	3

COMPARISON	Barrel	Log Barrel
Numb transistors	N^2	$2N \cdot \log_2(N)$
Numb series trans	1	$\log_2(N)$
fanout data lines	N	$3 \log_2(N)$
fanout control lines	N	N+1

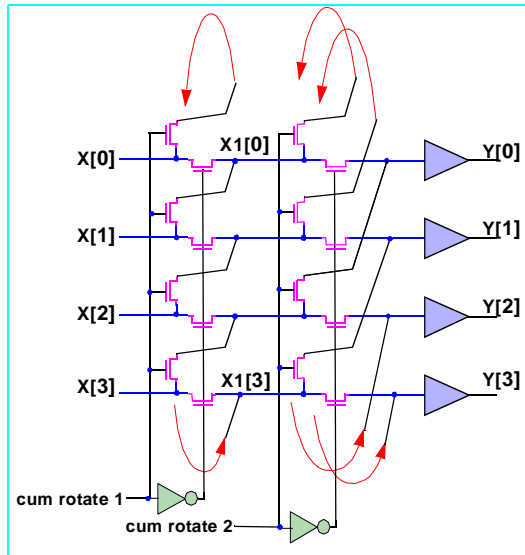
Summary

Log shifter has better area esp. for large N

Speed depends on:
fanout

number of series trans.

Log shifter wins because of lower fanout.



Programmable Shifters ▪

Barrel Rotator Shifter (cont.)

Size Comparison for Rotators

inputs, n	4	8	16	32	64	128
transistors, n^2	16	64	256	1024	4096	16384
transistors, $2n \log_2(n)$	16	48	128	320	768	1792

Speed Comparison

For m transistors in series, the delay is roughly $R(3C)(m(m+1)/2)$

$$m = \log_2(n)$$

R is the channel resistance.

C is the drain capacitance of each transistor. See Comment on Slide 14.

For the logarithmic rotator there are an average of 3 transistors at each junction.

For the standard barrel rotator the major delay is $R(nC) + 2R(nC)$.

The driver transistors for X_i see a capacitance of nC and the pass transistors see a load of nC .

They are summed according to Elmore's formula.¹

For the same values of R and C, this analysis gives relative delays of:

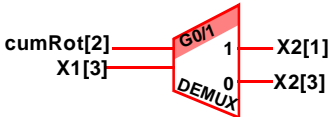
inputs, n	4	8	16	32	64	128
linear delay, $(n+2n)$	12	24	48	96	192	384
logarithmic delay, $3m(m+1)/2$	9	18	30	45	63	84

This does not include control line delay and makes some rather crude assumptions. However they indicate the logarithmic rotator saves time as well as area.

¹ See M.J.S Smith, Application - Specific Integrated Circuits, 1997, Addison Wesley, p. 280.

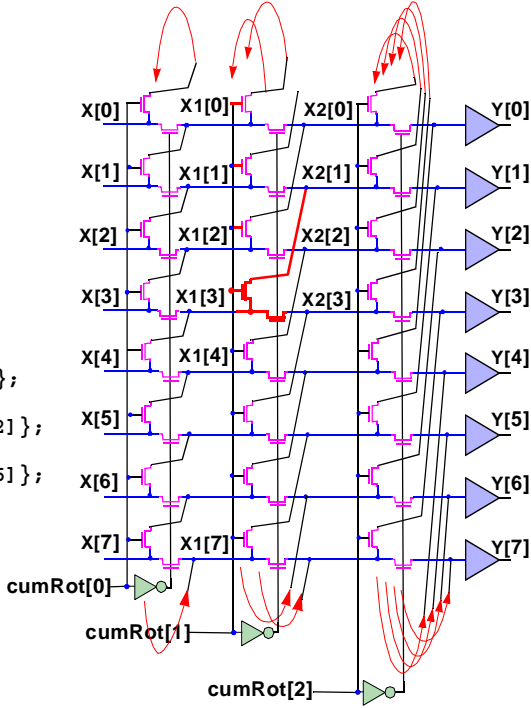
▪ Programmable Shifters ▪

Verilog; 8-bit logarithmic shifter
 This code will synthesize into 24 DEMUXs



```

wire [7:0] X, X1, X2, Y;
wire [3:0] cumRot;
always @(X or X1 or X2 or cumRot);
begin
  X1 = (cumRot[0])? X:{X[0],X[7:1]};
  X2 = (cumRot[1])? X1:
      {X1[1:0],X1[7:2]};
  Y = (cumRot[2])? X2:
      {X2[3:0],X2[7:5]};
end
    
```



Programmable Shifters ▪

Barrel Rotator Shifter (cont.)

Verilog Code

The code shows an 8-bit logarithmic rotator.

The shift operations are fixed, 1, 2, or 4. Not variable amounts, thus the synthesizer will not have trouble.

The synthesizer will use 24 CMOS MUXs. Not the NMOS circuit shown.

However the logarithmic circuit keeps the size reasonable.

The mux shown would replace the two transistors shown in red and bold.

▪ Programmable Shifters ▪

Logarithmic Circuits in General

Examples of some circuits that can be made logarithmic

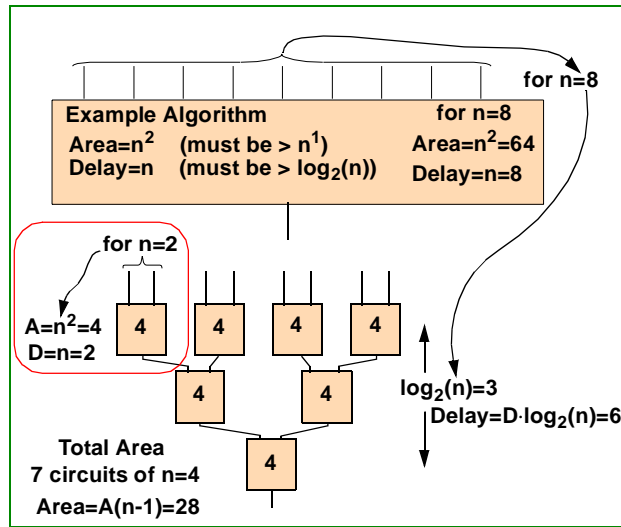
- Barrel Shifter
- Comparator
- Counter
- Brent-Krung Adder (carry-chain)

General Example

n inputs

Algorithm must obey

- Area (or power) = $K \cdot n^b$ $b > 1$
or
- Delay = $f(n)$ $f(n) > C \cdot \log_2(n)$
(K, C and b some constants)
- Must be able to divide circuit into smaller blocks



- If the interconnections are not too large, one can save area or time.
- The comparator behaves like the “Example Algorithm” in the figure.

Programmable Shifters ▪

Logarithmic Circuits

Logarithmic Circuits

General Properties

Some circuits that perform certain functions, can be created to increase logarithmically as the number of inputs “n” grows. The logarithmic barrel shifter has an area that increases as $n \cdot \log_2(n)$. The logarithmic comparator (next slide) has a delay that increases as $\log_2(n)$. The Brent Krung adder (described in the adder section) has a carry chain in which the delay increases as $\log_2(n)$.

Different growth rates as n increases

For large n, the dominate term is the only one that is important. Thus if the area of a circuit increases like

$$\text{Area} = A \cdot \log(n) + B \cdot n + Cn \cdot \log(n) + Dn^2$$

The important term for n large enough is Dn^2 . However if $D=0$, the important term becomes $Cn \cdot \log(n)$, etc.

For small n, one of the lower order terms may dominate if A or B have a large value, but as n increases the higher order terms will always win out.

Interconnections

In the example, the interconnections between the small blocks are shown as single wires. Sometimes they are more complicated, requiring many wires and logic. This interconnect cost may negate the gains from dividing the circuit into small blocks.

Barrel Rotator

Linear Area is proportional to n^2

Logarithmic Area is proportional to $2n \log_2(n)$

Linear Delay is proportional to $(3n)$

Logarithmic Delay is proportional to $3 \log_2(n)(\log_2(n)+1)/2$

▪ The Logarithmic Comparator ▪

The Logarithmic Comparator

Basic Comparator Block

Description of the yellow box

X_b	Y_b	X_a	Y_a	$X > Y$	$X < Y$
1	0	-	-	1	0
0	1	-	-	0	1
$X_b = Y_b$	1	0	1	0	0
$X_b = Y_b$	0	1	0	0	1
$X_b = Y_b$	0	0	0	0	0

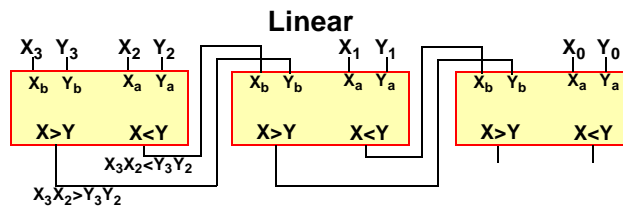
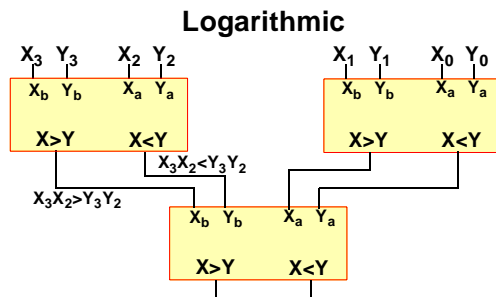
Compare two binary numbers

Each block compares:

- two 2-bit numbers

or

- two 1-bit numbers and the result of a previous compare.



The Logarithmic Comparator ▪

Logarithmic Circuits

The Comparator

The output of the comparator is a bit which acts like a new x and y

Thus $X > Y$ means the number composed of the bits $x_i x_{i-1}$ is larger than the number composed of bits $y_i y_{i-1}$.

The two bits ($X > Y$) and ($X < Y$) can be used in the next comparison.

Linear vs Logarithmic

Delay

For the linear comparator, the delay increases

$$\text{Delay} = B \cdot n - B$$

For the logarithmic comparator the delay increases

$$\text{Delay} = A \cdot \log(n)$$

Thus one can see the logarithmic comparator will be much faster as n gets large.

Area

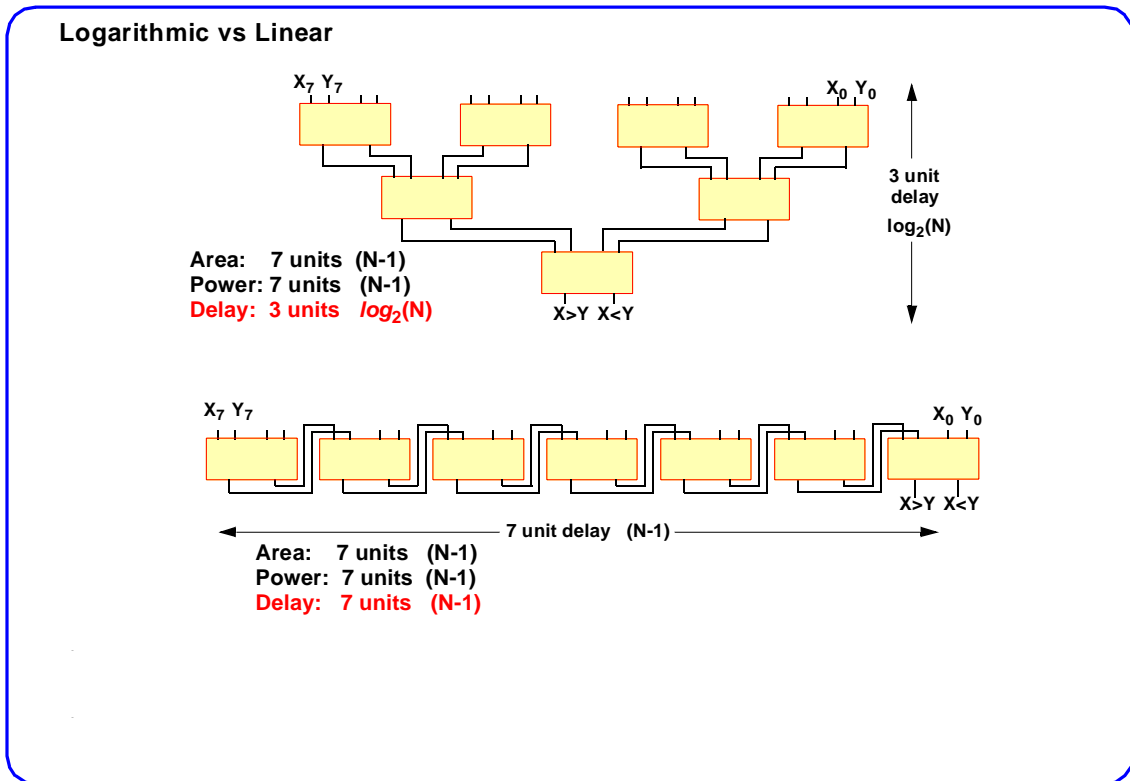
There is no difference in area.

Serial inputs give different results

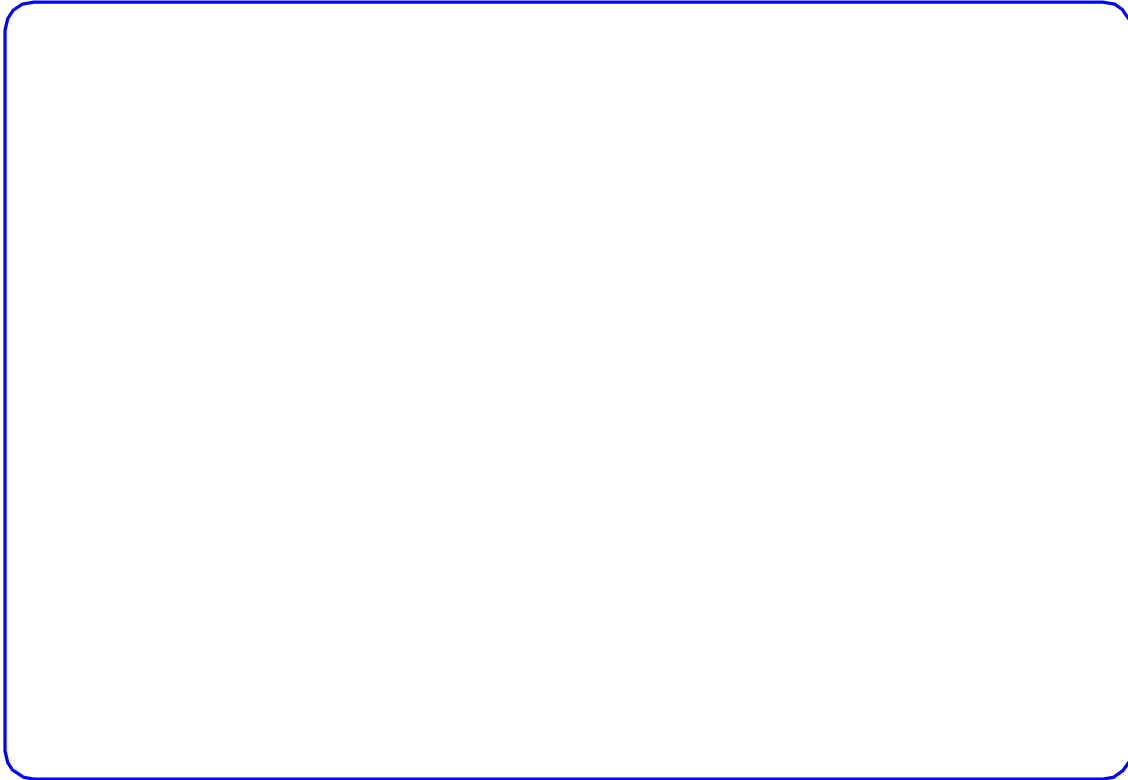
However do not think this spells the death of the serial (linear) comparator. Suppose the data came in serially with X_7, Y_7 coming in first and X_0, Y_0 coming at the end. Then the linear circuit would likely get the result out first because the result would come out one block delay after X_0, Y_0 were stable.

In the logarithmic comparator the result has to go through $\log_2(n)$ blocks after the last two input bits are stable.

▪ The Logarithmic Comparator ▪



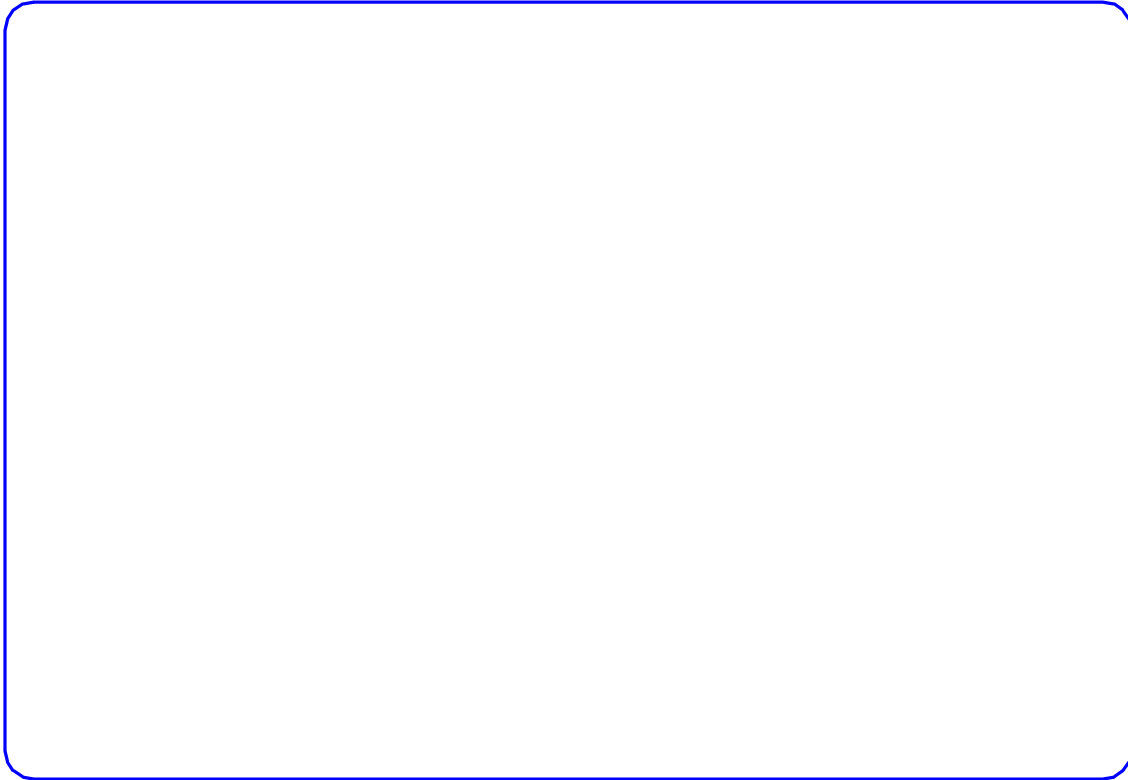
▪ The Logarithmic Comparator ▪



▪ The Logarithmic Comparator ▪



▪ The Logarithmic Comparator ▪



▪ The Logarithmic Comparator ▪



▪ The Logarithmic Comparator ▪

