Will Moore

MT 2007

C3B Digital Electronics

The course is about both practical and theoretical issues of digital design.

1. Logic Design and Key Parameters

1.1 Combinational logic: delays, fan-out, noise margins, hazards.

1.2 Sequential logic: set-up, hold and propagation times; metastability & synchronisation.

1.3 Fundamental-mode analysis: stable states, races & essential hazards.

1.4 Pulse-mode analysis: finite state machines.

1.5 Sequential logic design: capturing the specification, initialisation, minimisation, state variable assignment, synthesis.

1.6 Some Programmable Logic devices

2. Pipelining and DSP

2.1 DSP: analogue *vs.* digital; convolution *vs.* correlation.

2.2 FIR & IIR filters

2.3 DSP chips and arithmetic

2.4 VLSI implementation.

3. Signal Integrity & Thermal issues

3.1 Analogue Issues: sensitivities and coupling mechanisms.

3.2 Digital Issues: noise margins, supply decoupling, transmission line delays, crosstalk

3.3 Packaging: supply bounce.

3.4 PCB *vs.* IC issues

3.5 Clock skew and clock circuit design.

3.6 Thermal issues.

4. Testing

4.1 Defects, complexity, fault models, coverage.

4.2 Test pattern generation.

4.3 Scan path and built-in self-test. Boundary scan.

4.4 Other issues: delay-fault testing, functional testing.

Learning Outcomes

After attending this course, undertaking recommended background reading and solving the problem sheets you should:

Have an appreciation of modern digital circuit design options.

Understand the key parameters of elementary logic & sequential elements and how they constrain the design and dictate the performance of digital designs.

Understand the difficulties of fundamental-mode design, how to overcome them and how they may be avoided by pulse-mode design.

Be able to design a fundamental-mode circuit.

Understand the main types of digital architecture that are used, when each is appropriate and how this relates to available synthesis tools.

Understand why we use DSP and how it is implemented.

Understand what is meant by signal integrity, the key mechanisms involved and design techniques to overcome them.

Understand the importance of clock design and techniques to control clock skew.

Understand the importance and difficulties of testing modern electronic systems, and the importance of using a fault model and of getting good fault coverage.

Understand the basis of the D-algorithm and be able to apply it to simple circuits.

Understand the need for and the most common design techniques used to improve the testability of sequential circuits.

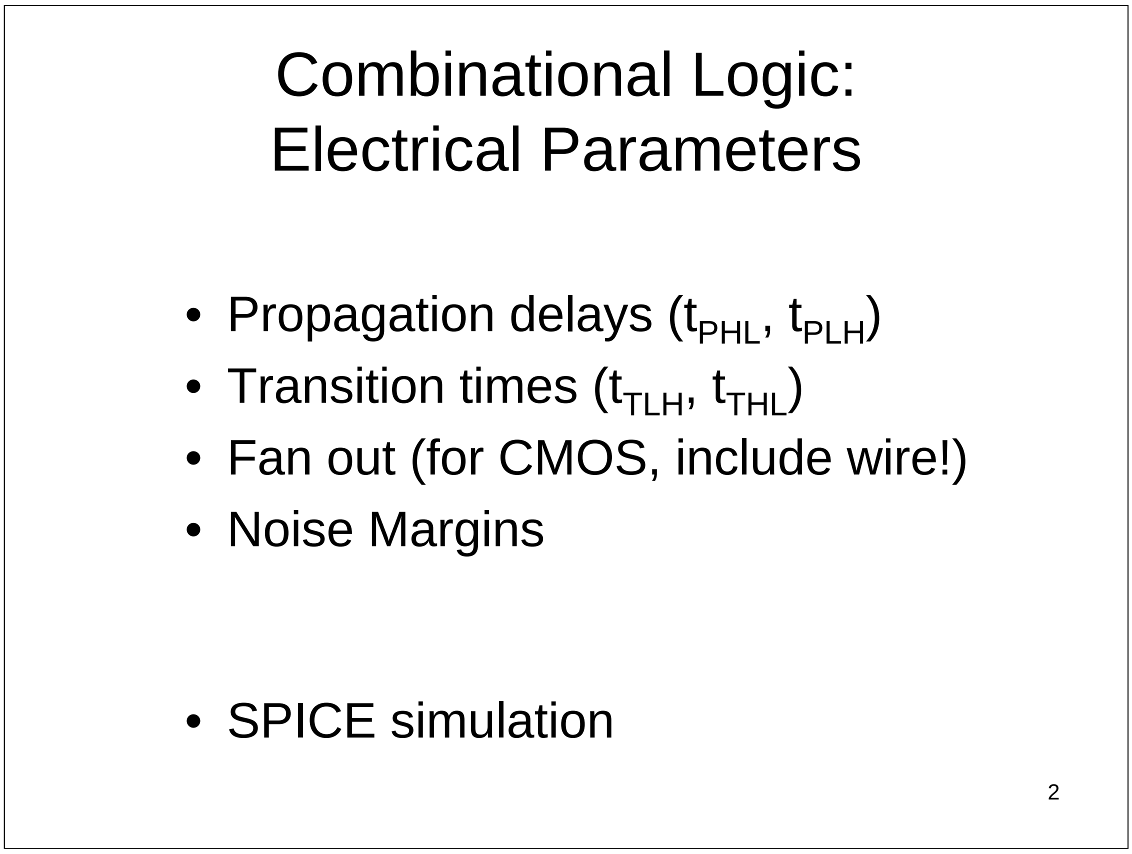
1. Logic Design & Key Parameters

1

- Combinational logic: delays, fan-out, noise margins, hazards.
- Sequential logic: set-up, hold and propagation times; metastability & synchronisation.
- Fundamental-mode analysis: stable states, races & essential hazards.
- Pulse-mode analysis: finite state machines.
- Sequential logic design: capturing the specification, initialisation, minimisation, state variable assignment, synthesis.

In this section of the course, I will stick fairly closely to E. J. McCluskey, "Logic design principles: with emphasis on testable semicustom circuits", Prentice Hall, 1986. A classic text which you should be able to find a the library but regrettably now out of print.

There are lots of other texts which also cover this material. J F Wakerly, “Digital Design: Principals and Practices”, Prentice Hall, 4th Edition 2004 is probably the best.



Combinational Logic: Electrical Parameters

- Propagation delays (t_{PHL}, t_{PLH})
- Transition times (t_{TLH}, t_{THL})
- Fan out (for CMOS, include wire!)
- Noise Margins

- SPICE simulation

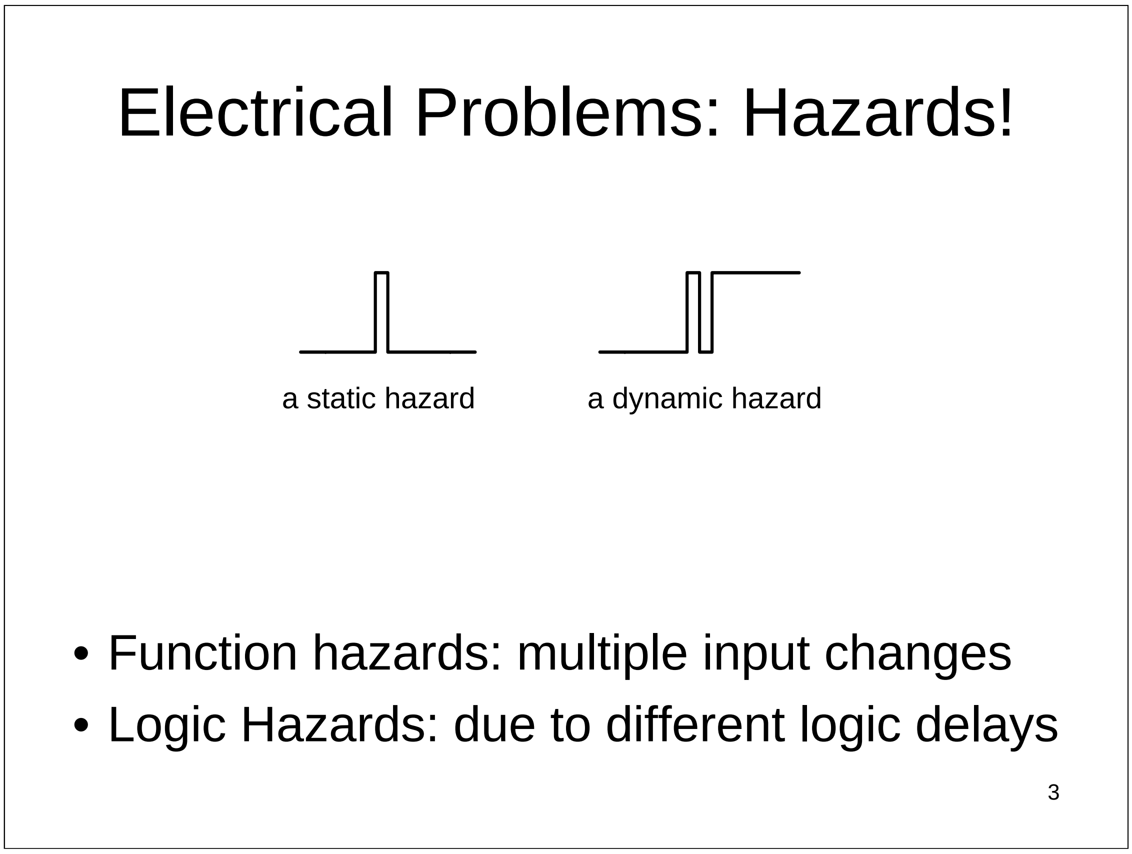
2

At the logic level, we like to assume the *physical* issues are solved but we cannot escape certain *electrical* issues which we can quantify in the terms here. Remember that nothing ever is quite 0 / 1 or quite instantaneous.

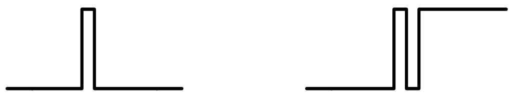
In bipolar logic, fan-out is determined by output current driving capability and the input current requirement. In CMOS is all to do with R-C charging – pretty often on ICs and on PCB’s the wiring dominates the total capacitance.

If in doubt, we might well want to *simulate* the critical parts of our circuit with an "analogue" simulator (e.g. Spice). "Critical" here most often means the part of the circuit that we think may be slowest, but it may also be a response to some other problems.

Don’t forget to make your own notes!



Electrical Problems: Hazards!



a static hazard

a dynamic hazard

- Function hazards: multiple input changes
- Logic Hazards: due to different logic delays

3

Sometimes also referred to informally as glitches, spikes, runt pulses.

Function Hazards

- Change function?   X
- Make sure only ONE input changes?
- Pulse-mode design?

4

A function hazard arises when we change more than one input “at the same time”. It can happen in any logic gate except an inverter!

Logic Hazards

- Hazard-free design; bridging terms? X
- Electrical design / simulation ?
- Pulse mode design?

5

A logic hazard may arise when there is “reconvergent fan-out” (*i.e.* more than one path from an input to an output) and there are different delays along the two paths. The classic example is the multiplexer made from three 2-input NANDs and an inverter. You CAN add an extra NAND to patch the problem – it’s called a “bridging term” because on a Karnaugh map it bridges between two islands in the original circuit. HOWEVER, (a) it adds to the expense, (b) it’s untestable, (c) it’s impossibly tedious on a reasonable sized circuit (d) it’s much better to redesign the circuit or use pulse-mode design.

Latches & Flip-flops: Electrical Parameters

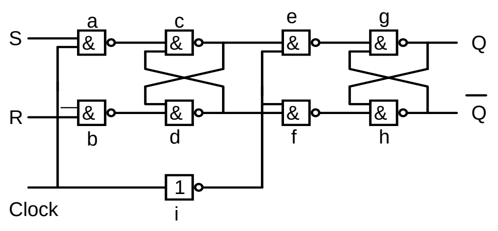
- Set-up time (t_{SU})
- Hold time (t_H)
- Propagation delay (t_P)
- Min. control pulse width (t_{WL}, t_{WH})

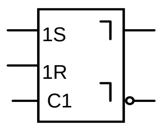
6

You probably should have a vague idea about the various different types of flip-flops (SR, JK, T, D-type) and about level vs. edge triggering*, but in practice, the dominant flip-flop is the D-type which can only be edge triggered. They are smaller, faster, cheaper

* (SR and JK flip-flops can be either level or edge triggered – i.e. if constructed as master-slave devices, the inputs are “ones-catching” – i.e. dynamic hazards while the master section is active will be remembered and affect the output when the slave becomes active. This can’t happen with a D-type because there’s only one input.)

A Master-Slave S-R Flip-Flop





Assuming each gate has a unit delay τ, then:

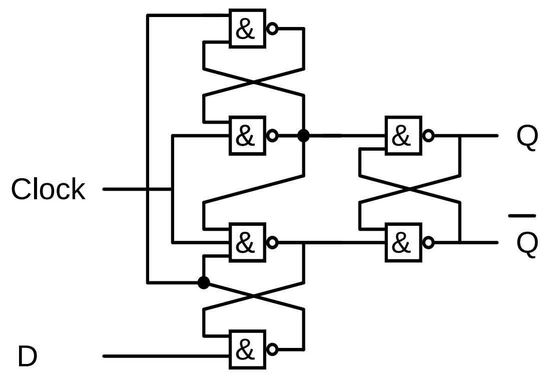
$t_{WH} = 3\tau$ (the time for a, b, c & d to settle)
$t_{WL} = t_P = 4\tau$ (the time for i, e, f, g & h to settle)
t_{SU} = Clock High period
(otherwise c & d will respond to any hazards on S or R)
$t_H = 0$ (no need to hold inputs after clock falls)

7

Obviously it is a gross simplification to assume that each gate has the same delay and in practice we ought to simulate the circuit properly using SPICE.

However, it is a useful exercise to see how typical values of these parameters might arise.

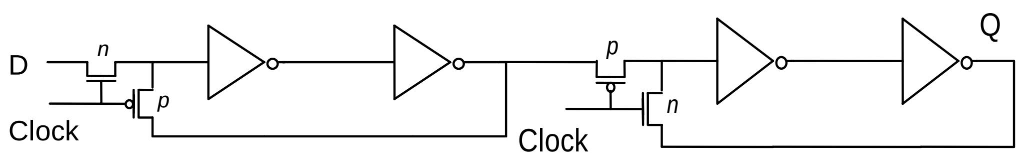
A D Flip-Flop



8

Try to work out likely parameters for this circuit yourself.

A CMOS Static Flip-flop



9

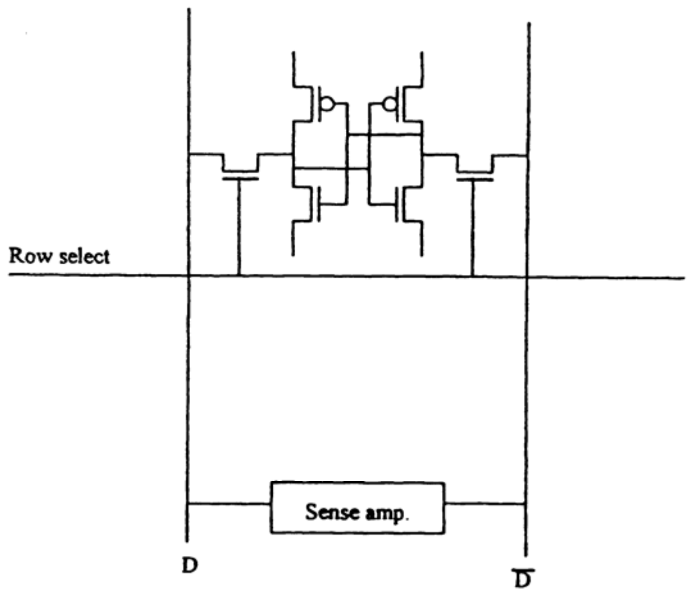
.. and for this one.

For simplicity, assume equal delays in the pass transistors and inverters.

Note that (for obvious reasons??) your unit of delay is likely to be smaller than it was for the NAND gates in the previous circuits.

This is pretty much the most common flip-flop design in use - variants include pass gates instead of pass transistors and weak driving of the feedback loop replacing the feedback transistor, notably in …

RAM

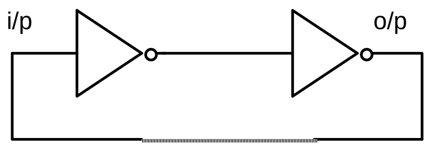


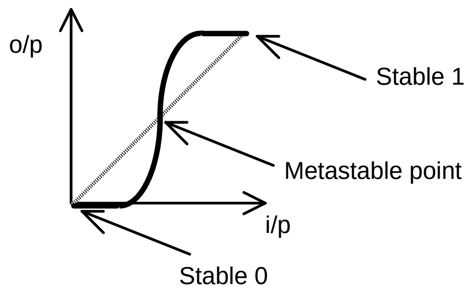
10

… the standard six-transistor Static RAM cell - a D-latch!

(For denser RAM, we might use a one-transistor Dynamic RAM cell – what would that look like?

Electrical Problems: Metastability



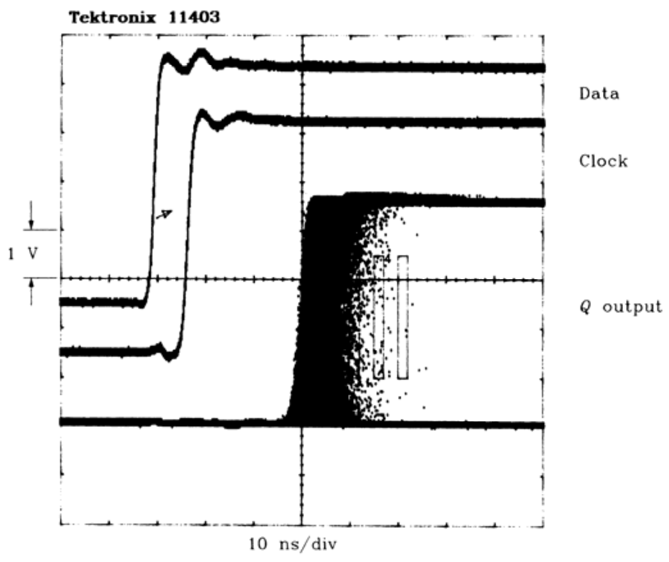


11

When signals do not meet the t_{SU}, t_H or t_W criteria, it is possible for a latch to be put in an intermediate state between "1" and "0".

This state cannot exist indefinitely, but for the indeterminate time that it does, the input cannot be interpreted reliably.

Metastability



Mask 1	30 ns	4685
Mask 2	35 ns	445
Mask 3	40 ns	42
Mask 4	45 ns	4

12

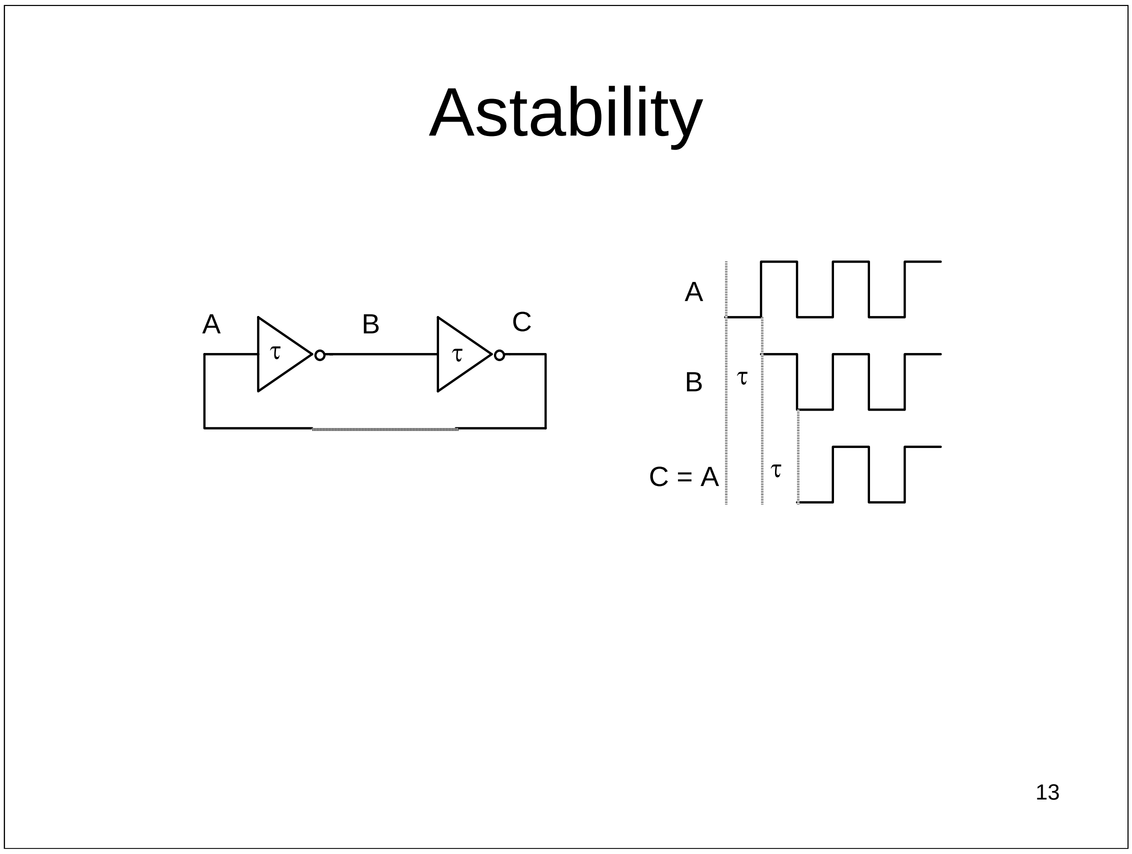
In this experiment a feedback loop was used to continuously adjust the timing of the data and clock to try and excite metastability.

The probability of the metastable state lasting for a given time is clearly reducing exponentially.

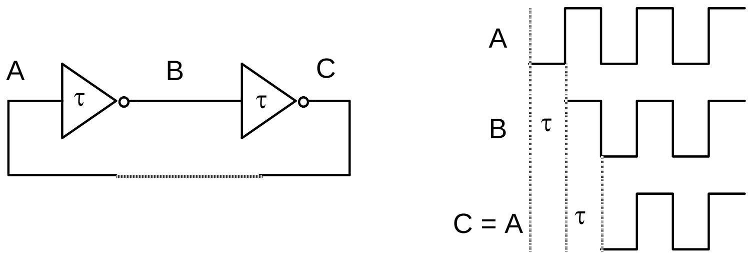
That means:

(a) It will never get to zero!

But, (b) we can make the probability of metastability as small as we like if we are prepared to wait



Astability

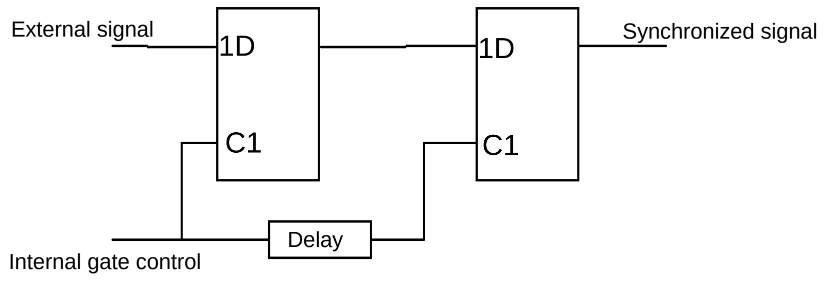


13

An alternative problem occurs when a short input or control pulse equals the gate propagation time so that the latch (or flip-flop) oscillates for an indeterminate time.

This requires that the rise and fall times of the signals is much less than the gate propagation delay which is not so often true.

Synchronizer



14

The above two phenomena may arise due to physical problems (such as power rail spikes) or due to hazards, but we will assume that we have designed these problems out.

The other common cause is an external signal arriving from another unit which may easily infringe the set-up or hold time. It is therefore common to use a "synchronizer" circuit, gating the external signal with an appropriate internal signal (which might be the clock in a synchronous logic circuit).

In case the first latch should be put in a metastable or astable state, we may often put two latches in series to reduce the already small risk to an insignificant risk as shown here. If we use the clock as our gate this will turn into a master-slave D flip-flop. Alternatively we could use two edge-triggered flip-flops in series.

Fundamental- and Pulse-Mode Circuits

15

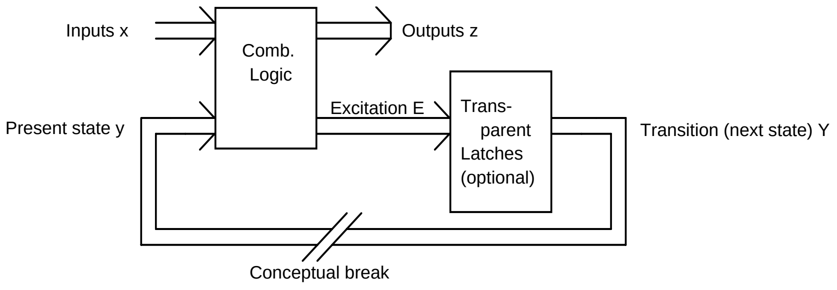
We can consider sequential circuits at two different levels.

The lower level is a "fundamental mode" and the higher level is the "pulse mode" (roughly equivalent to asynchronous design without a clock and synchronous design with a clock).

The lower level requires us to think more about the design and we will examine the problems that arise there.

(These will probably encourage you to adopt a pulse-mode style whenever possible! The point of having a design hierarchy is to hide away the lower level problems at every higher level in order cope with large designs.)

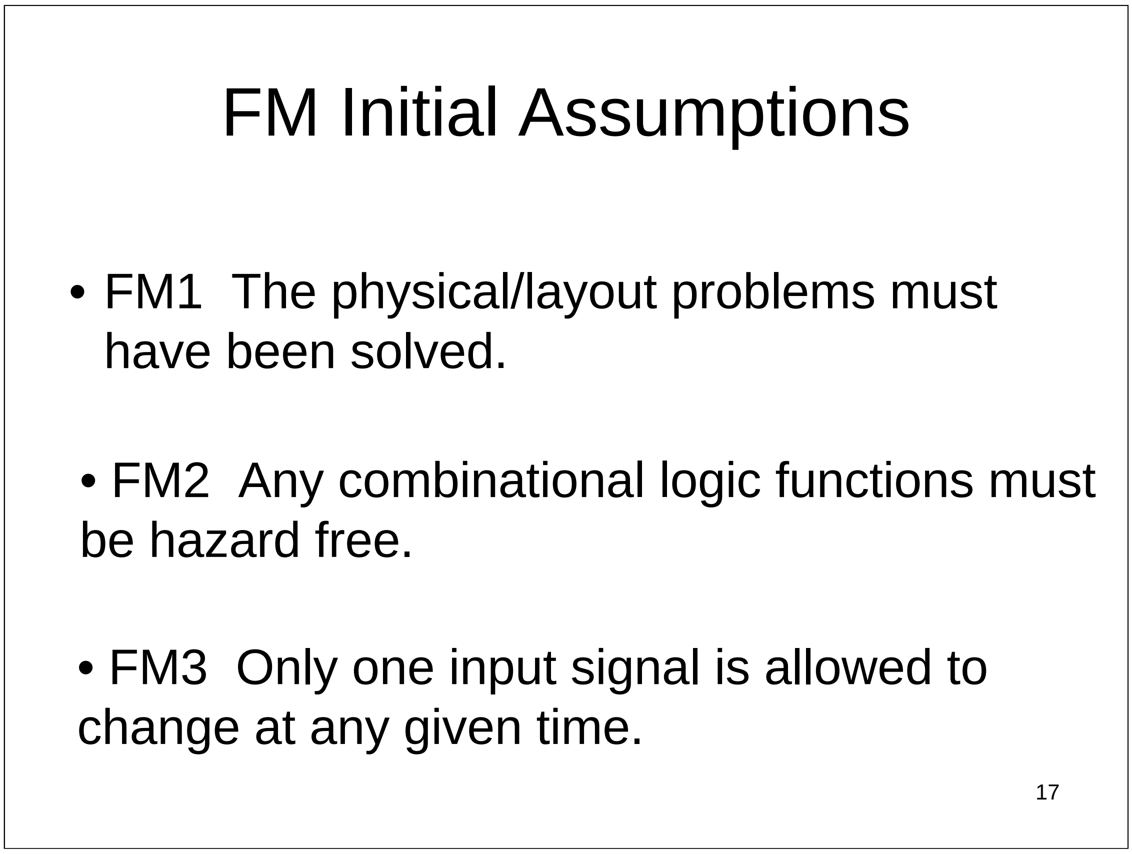
Fundamental-Mode Circuit



16

Here is a generic diagram of a fundamental-mode circuit.

From what we have learnt so far, we should sensibly start off with the following three conditions ….



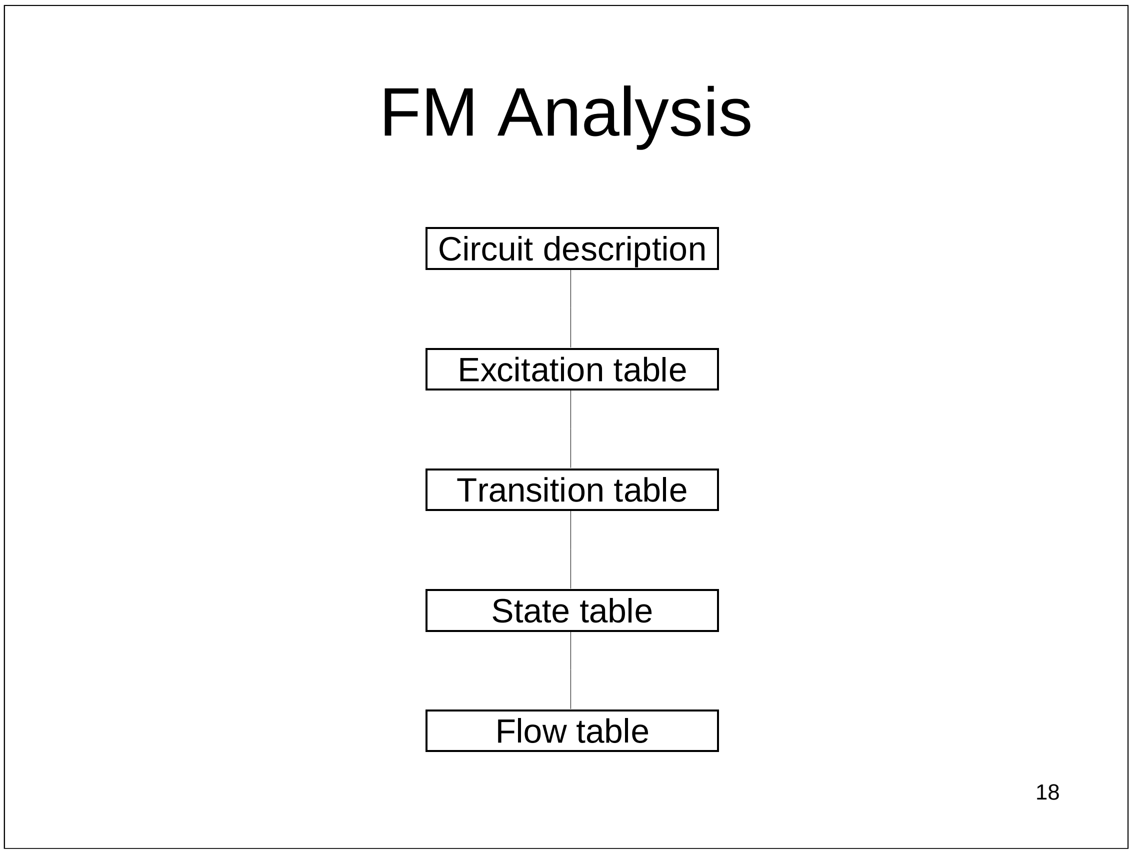
FM Initial Assumptions

- FM1 The physical/layout problems must have been solved.
- FM2 Any combinational logic functions must be hazard free.
- FM3 Only one input signal is allowed to change at any given time.

17

Why do you think we need these assumptions?

If we can satisfy these three conditions then we are on our way to making an unambiguous analysis of any sequential logic circuit by fundamental-mode analysis (as opposed to pulse-mode analysis which imposes other restrictions to gain other benefits - see later).



FM Analysis

Circuit description

Excitation table

Transition table

State table

Flow table

18

Faced with a fundamental mode circuit, it is possible to consider a standard approach to analysing it in order to understand what it is supposed to do. One such approach is summarised below. (Note that we will later consider the more common problem of *design* in which we will reverse the order of the steps).

Circuit description (Logic gates, latches and connections or Boolean description.)

> Extract combinational logic which drives the inputs to the Latches (or to artificial break in feed-back loop.

Excitation table

> Use characteristic equations of latches to predict next-state.

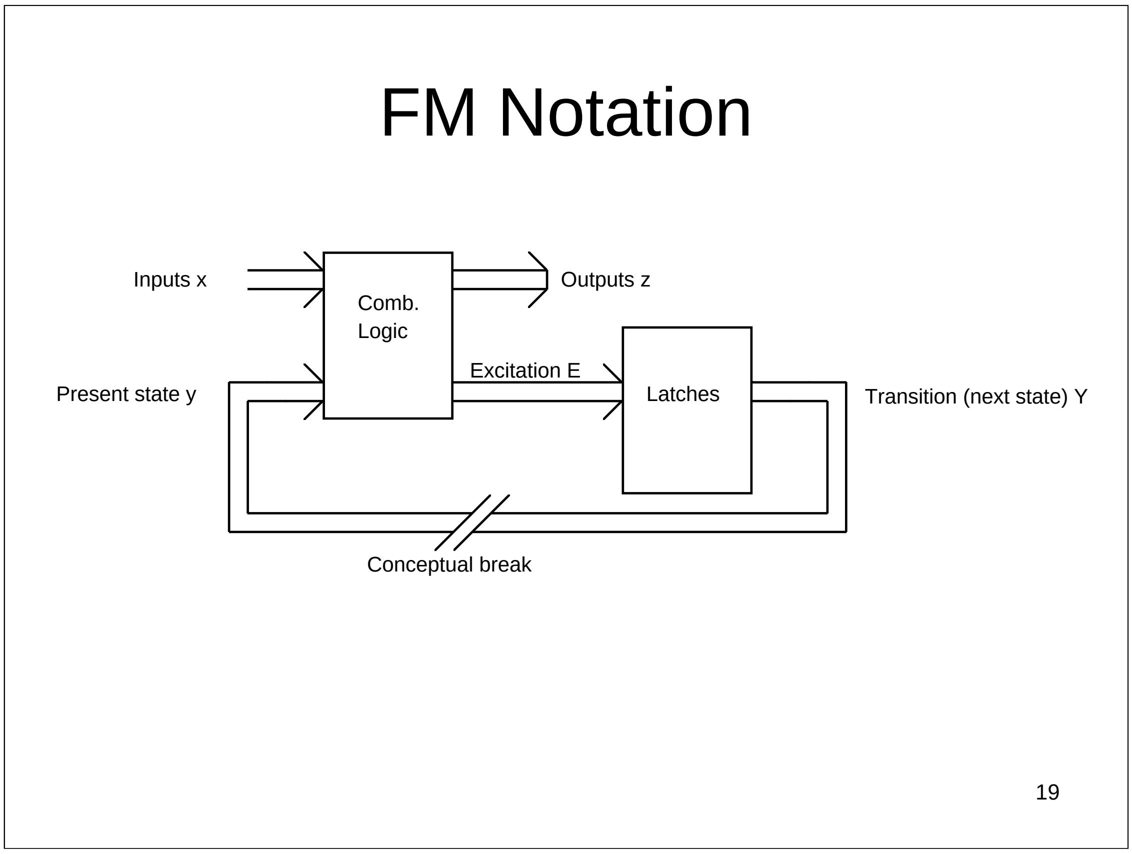
Transition table

> Abstract the system states away from specific 0/1 implementation.

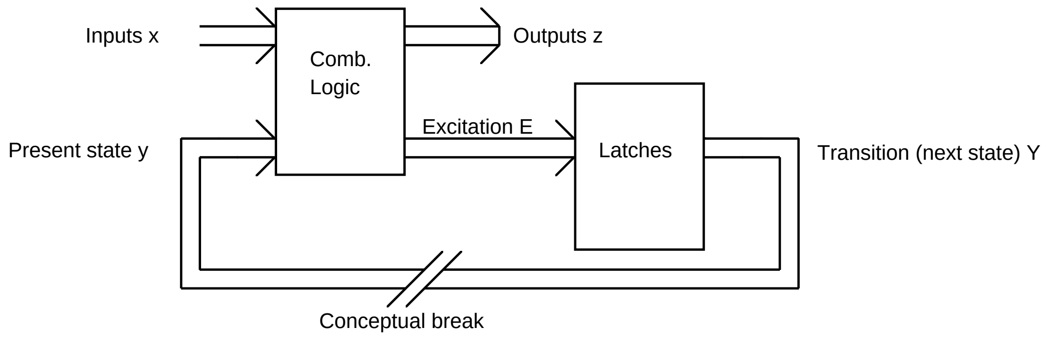
State table

> Eliminate superfluous information about transient states.

Flow table (Our most general specification of the sequential logic design.



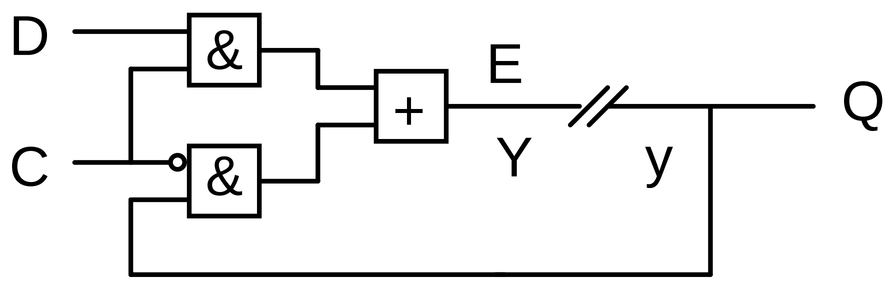
FM Notation



19

Note that for a fundamental-mode circuit, the states may arise from specified latches or simply because of feedback loops. There is no clock to help you out!

FM Analysis of a D-latch

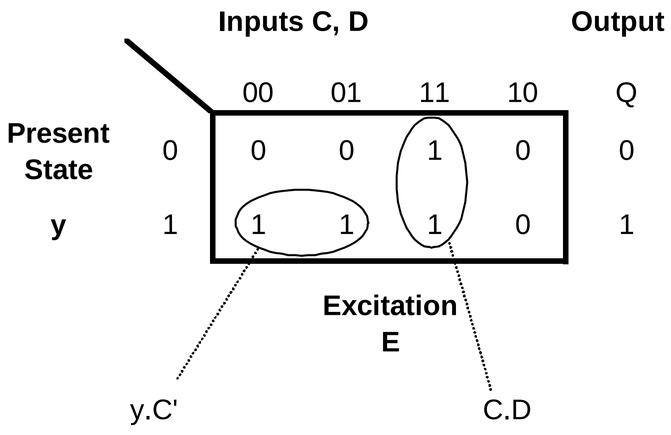


The *excitation function*, E = D.C + y.C’

In this elementary case the excitation function is the same as the *“transition function”*, Y = D.C + y.C’

(It is also sometimes called the *“characteristic function”* of the latch).

… its Excitation Table

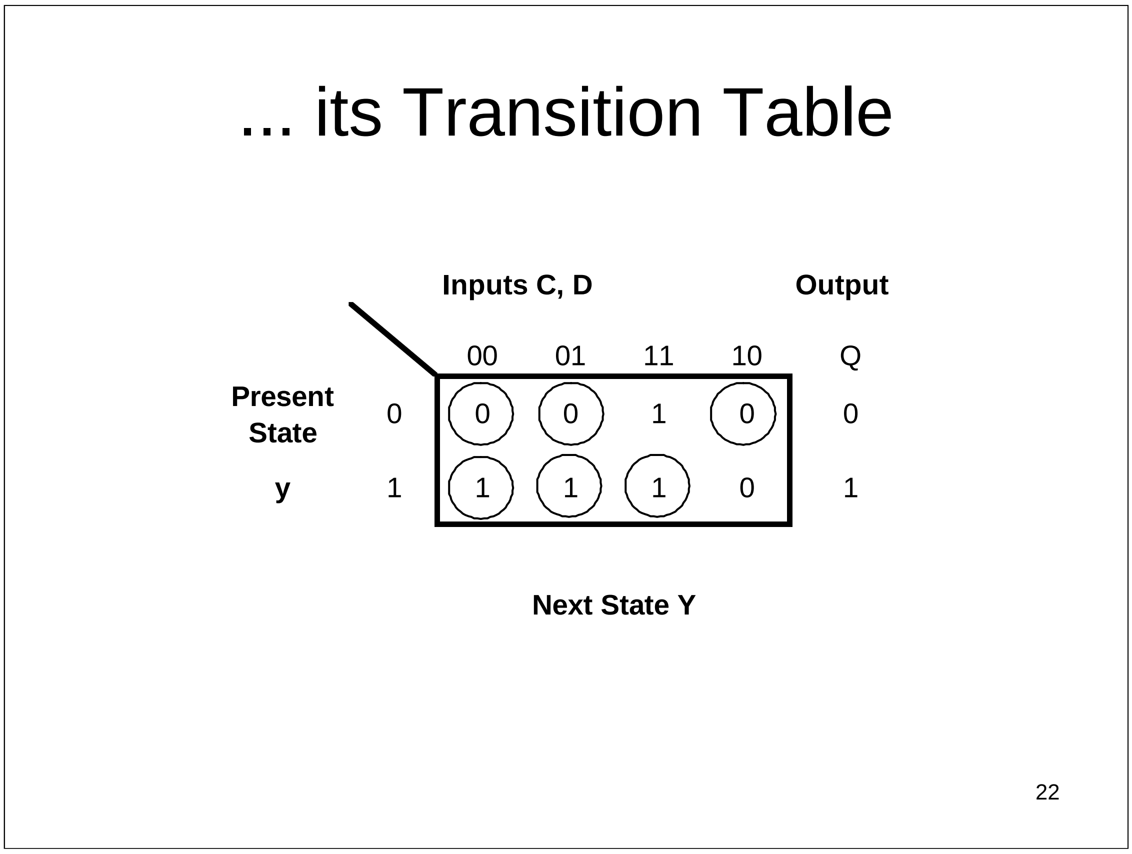


		Inputs C, D				**Output**
Present State y		00	01	11	10	Q
	0	0	0	1	0	0
	1	1	1	1	0	1

Excitation E

Note the potential logic hazard in this particular circuit as we change from C.D to y.C’

E may go to "0" causing y to go to "0" instead of "1"!



... its Transition Table

		Inputs C, D				**Output**
		00	01	11	10	Q
Present State	0	⓪	⓪	1	⓪	0
y	1	①	①	①	0	1

Next State Y

22

We usually circle the *stable* states where Y = y in order to interpret the table more easily.

If the states are not stable, we say they are “transitional”.

… and its State Table & State Diagram

		Inputs C, D				Output
		00	01	11	10	Q
Present State	A	A	A	B	A	0
s	B	B	B	B	A	1

Next State S

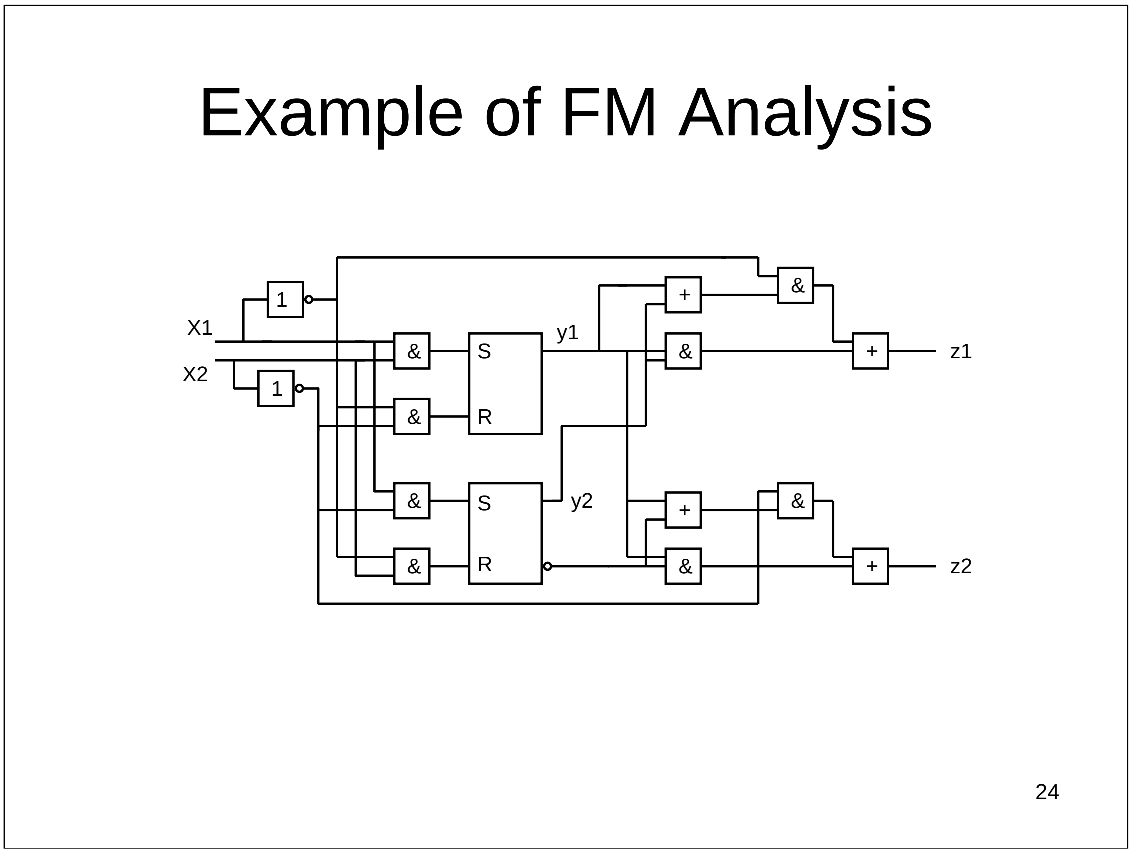
$$\begin{tikzcd}
\text{A(0)} \arrow[loop left, "\begin{matrix}00\\01\\10\end{matrix}"] \arrow[r, bend left, "11"] & \text{B(1)} \arrow[loop right, "\begin{matrix}00\\01\\11\end{matrix}"] \arrow[l, bend left, "10"]
\end{tikzcd}$$

23

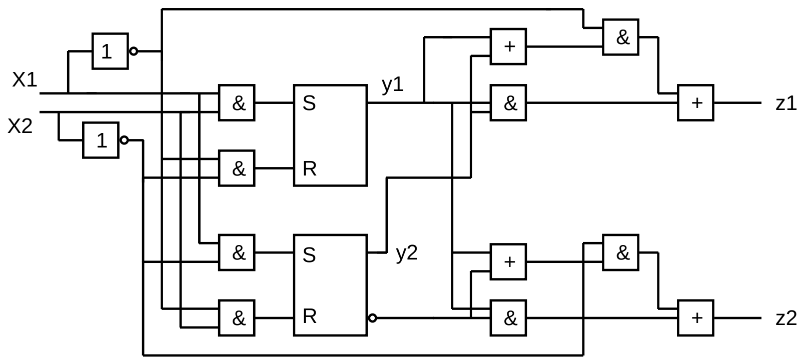
Here we perform an abstract labelling of states. This further helps to give us a better feel for what the circuit is supposed to do.

Also in the later lecture on *design*, when we follow these steps in reverse we will normally have the original specification in terms of abstract states.

The *state diagram* portrays the same information in graphical form.



Example of FM Analysis



Excitation Functions

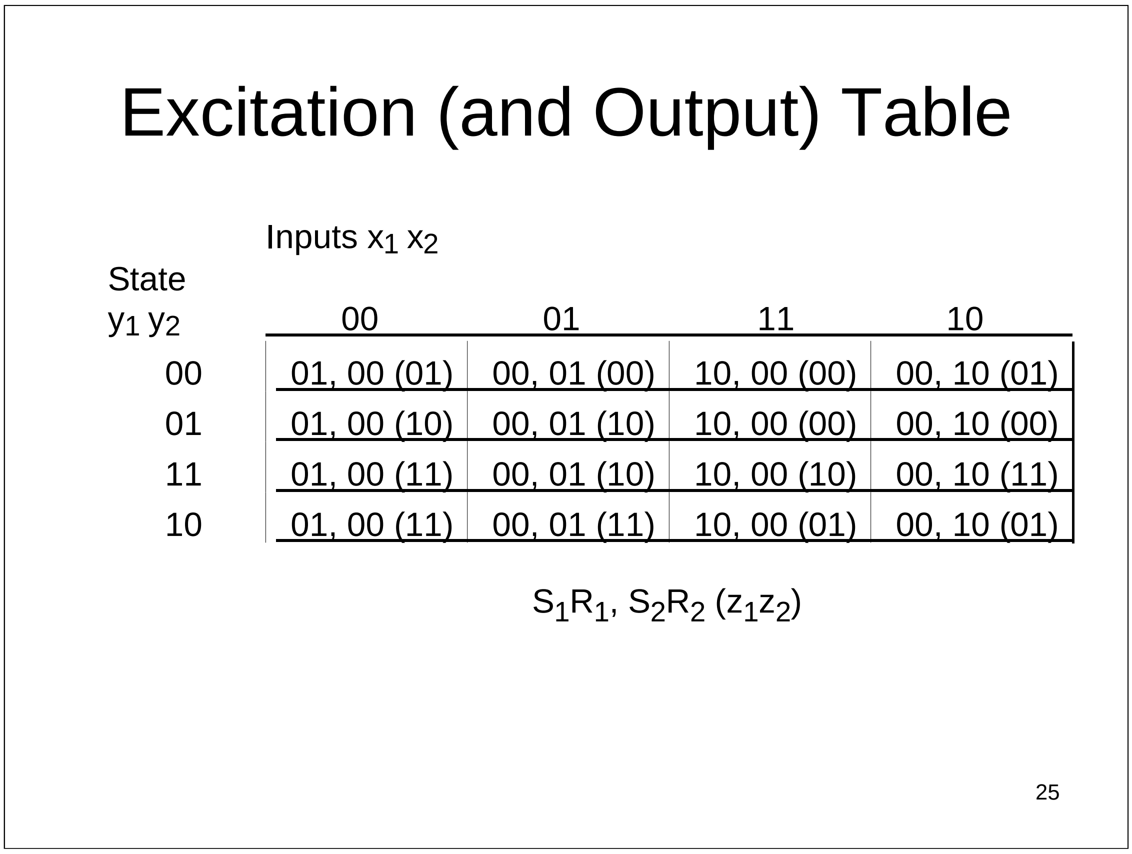
S1 = x1x2,

R1 = …etc.

Output Functions

z1 = y1. y2 + (y2 + y1).1,

z2 = ... etc.



Excitation (and Output) Table

Inputs $x_1\ x_2$

State $y_1\ y_2$	00	01	11	10
00	01, 00 (01)	00, 01 (00)	10, 00 (00)	00, 10 (01)
01	01, 00 (10)	00, 01 (10)	10, 00 (00)	00, 10 (00)
11	01, 00 (11)	00, 01 (10)	10, 00 (10)	00, 10 (11)
10	01, 00 (11)	00, 01 (11)	10, 00 (01)	00, 10 (01)

S_1R_1, S_2R_2 (z_1z_2)

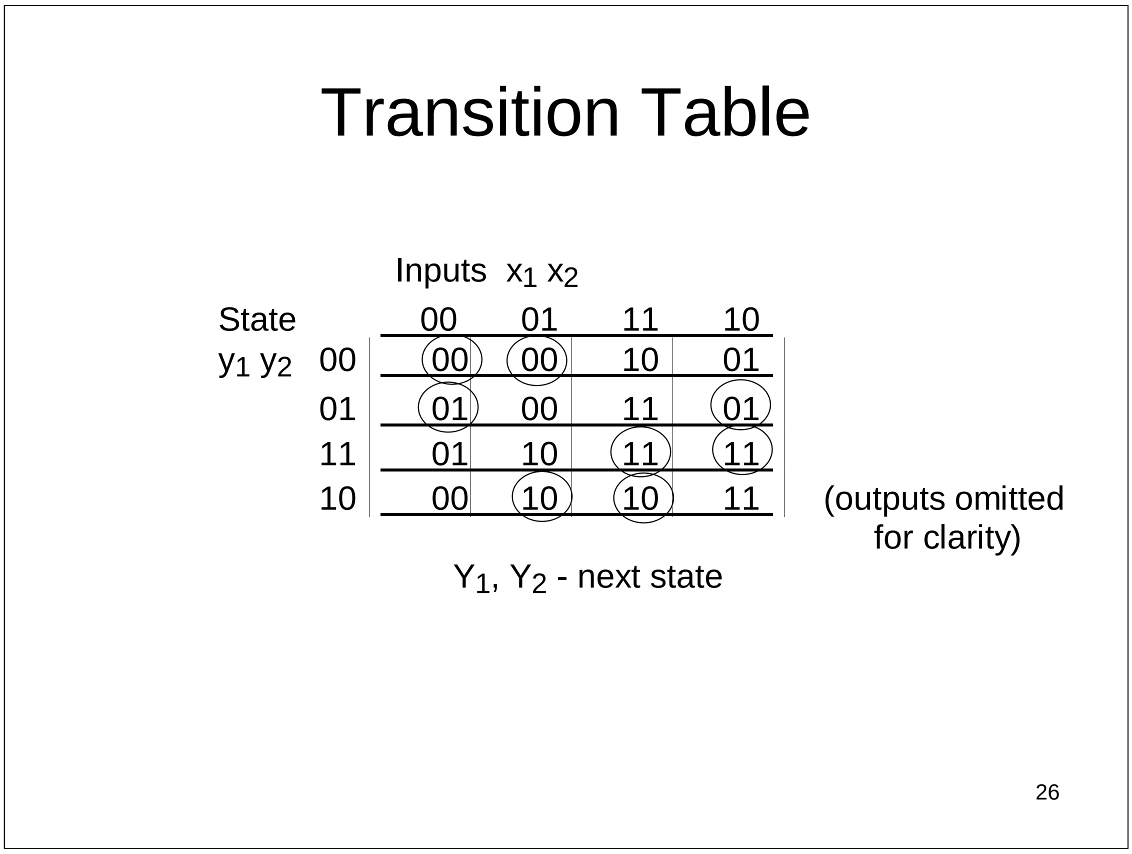
25

Then we put this together with the *Characteristic Function* (of S-R latch)

$Y = S + R'.y$ (set dominated)

….

[or reset dominated: $Y' = R + S'.y'$ - equivalent provided R=S=1 is disallowed.]



Transition Table

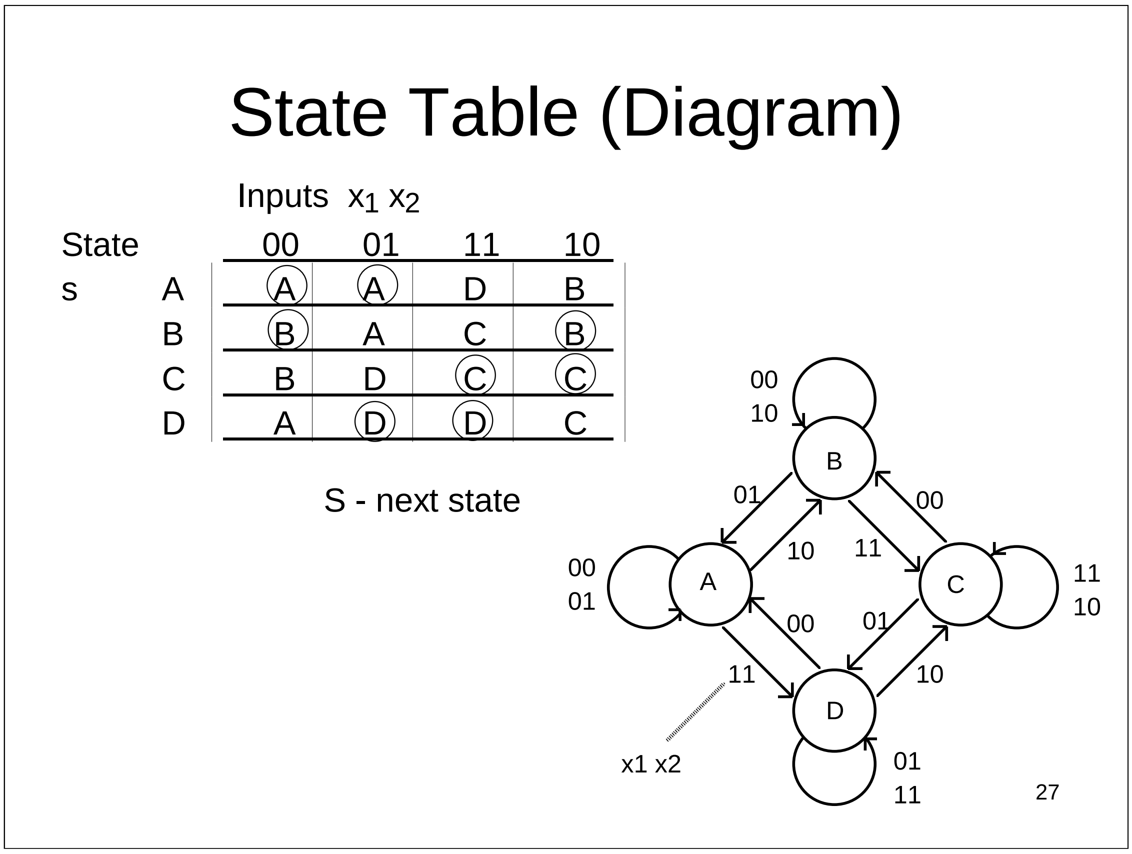
Inputs x_1 x_2

State y_1 y_2	00	01	11	10
00	$\boxed{00}$	$\boxed{00}$	10	01
01	$\boxed{01}$	00	11	$\boxed{01}$
11	01	10	$\boxed{11}$	$\boxed{11}$
10	00	$\boxed{10}$	$\boxed{10}$	11

(outputs omitted for clarity)

Y_1, Y_2 - next state

.. to give the transition table.

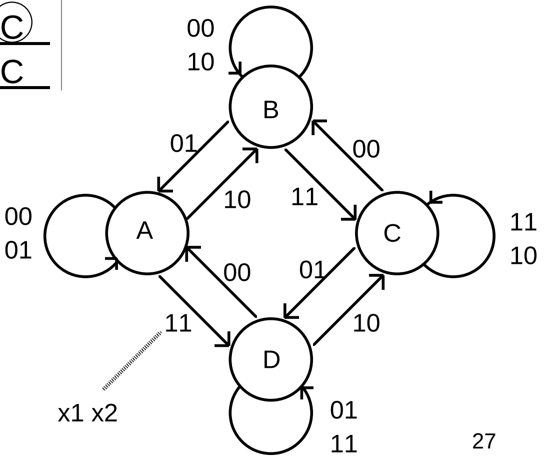


State Table (Diagram)

Inputs x_1 x_2

States		00	01	11	10
	A	Ⓐ	Ⓐ	D	B
	B	Ⓑ	A	C	Ⓑ
	C	B	D	Ⓒ	Ⓒ
	D	A	Ⓓ	Ⓓ	C

S - next state



27

In the above example the State Table is the same as the Flow Table because each entry is either a stable state or leads directly to a stable state.

See next slide for an example where they are different is.

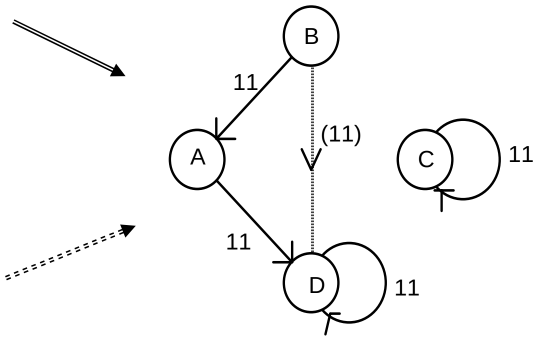
Example: Flow Table (Diagram)

State Table

	00	01	11	10
A			D	
B			A	
C			$\textcircled{C}$	
D			$\textcircled{D}$	

Flow Table

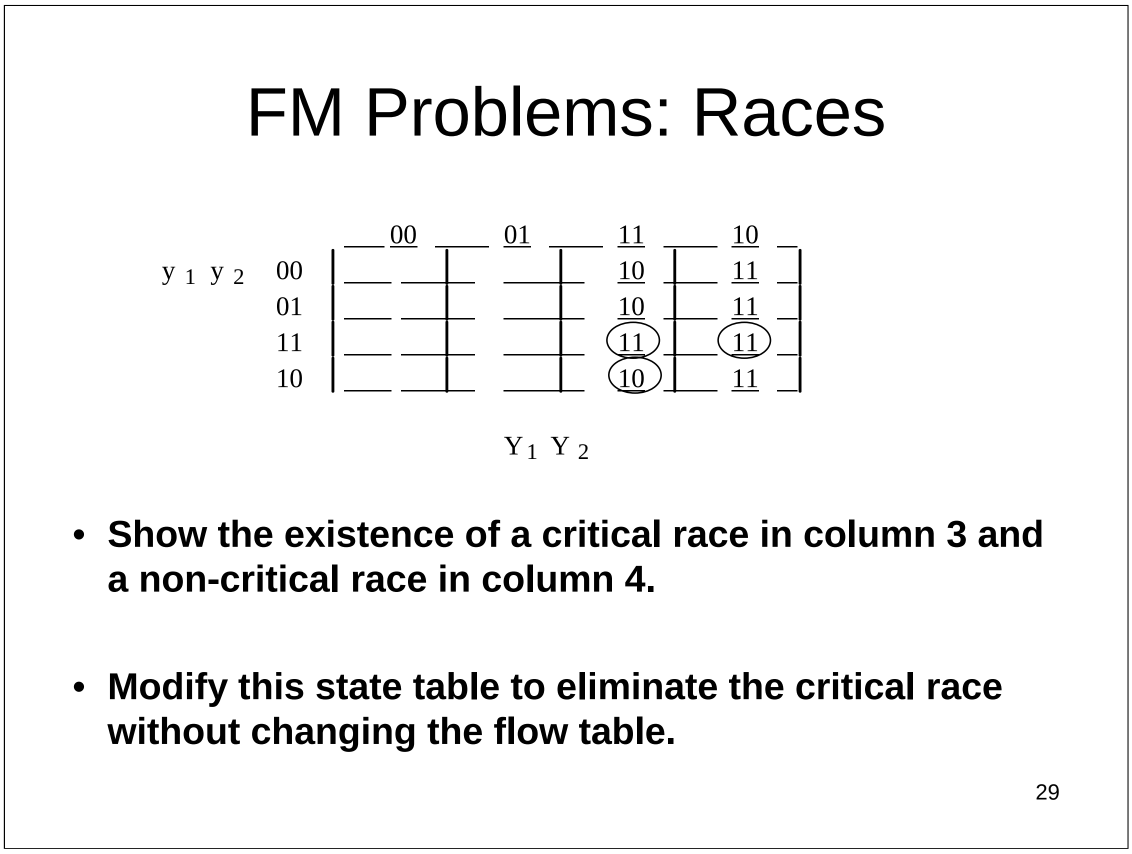
	00	01	11	10
A			D	
B			D	
C			$\textcircled{C}$	
D			$\textcircled{D}$	



28

The flow table tells us what the design is intended to do and eliminates irrelevant information about transient states.

(Though it may be that the designer has deliberately introduced these features in his implementation as we shall see below).



FM Problems: Races

$y_1 y_2$	00	01	11	10
00			10	11
01			10	11
11			11	11
10			10	11

$Y_1\ Y_2$

- **Show the existence of a critical race in column 3 and a non-critical race in column 4.**
- **Modify this state table to eliminate the critical race without changing the flow table.**

29

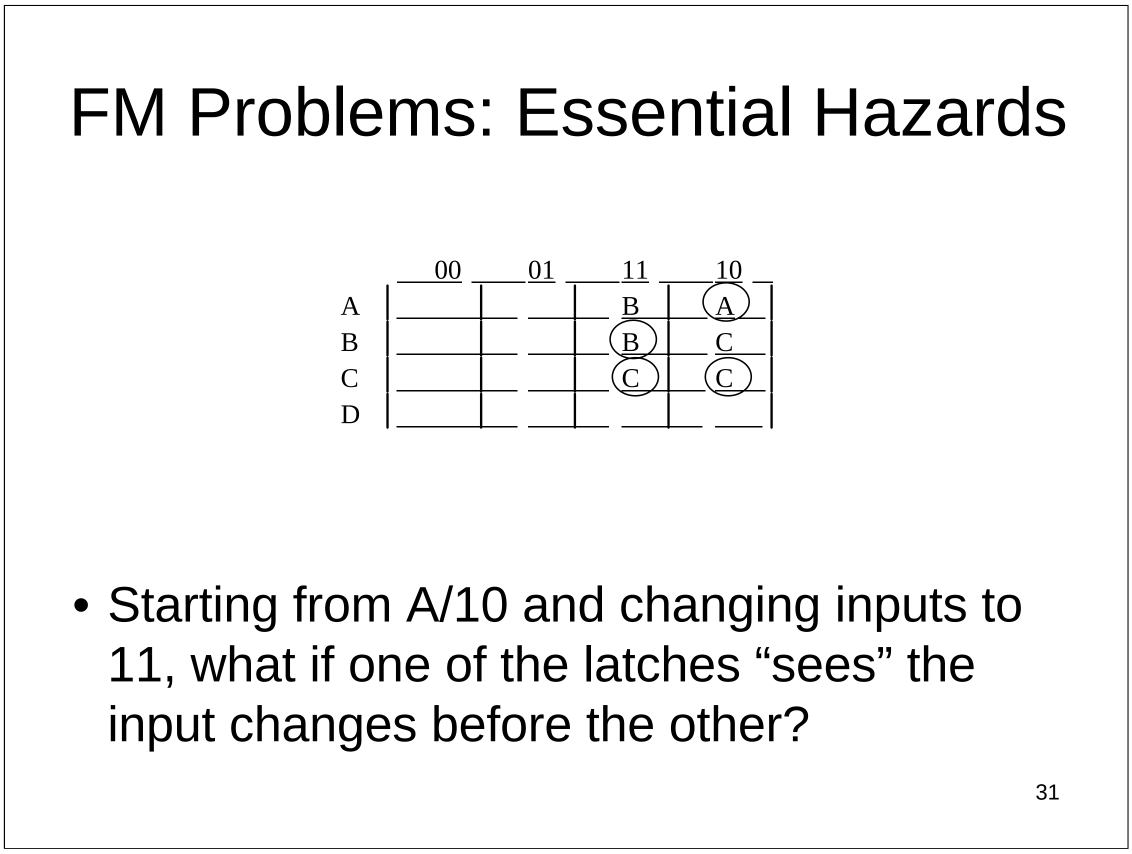
Races occur when two or more states variables change at the same time. If one actually changes fractionally before the other it is possible that the system finishes up in a different state to that intended. This is a *critical race*.

Races can often be eliminated by modifying the state table in this way without changing the flow table or else by assigning different binary state variables so that there is only ever one state variable changing at a time (this is probably easier to see on the state diagram). Sometimes we need to introduce an extra transitional state into the design (more on this in the design lecture).

We might be able to overcome the critical race by electrical design, ensuring that one particular state variable always changes first, but we would prefer not to dive down into this lower level of the design hierarchy if we can help it. Therefore we will normally add another constraint to our list of conditions for the reliable design of fundamental mode circuits:

FM Assumptions (continued)

- FM4 We must design out the possibility of critical races.



FM Problems: Essential Hazards

	00	01	11	10
A			B	Ⓐ
B			Ⓑ	C
C			Ⓒ	Ⓒ
D				

- Starting from A/10 and changing inputs to 11, what if one of the latches “sees” the input changes before the other?

31

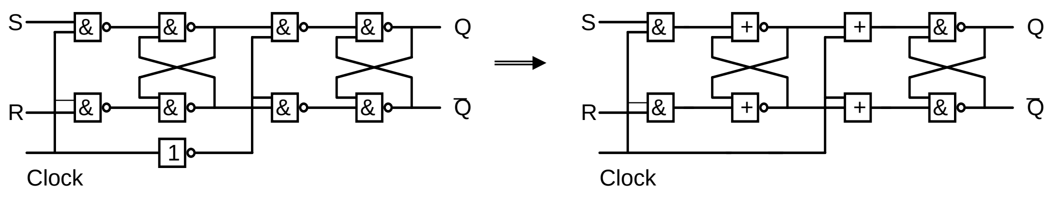
In fundamental mode circuits it is conceivable that the delay through the latches may be shorter than through the logic. It is then possible that one latch may respond to an input change and pass this new value to a second latch *before* the second latch has seen the original input change. If this causes an error, we call it an *essential hazard* and if the design is completely specified*, is **not possible** to get around the problem at this level of the design; we must control it by going down to the electrical design level (or else up to change the specification!)

We can easily test to see whether or not an essential hazard exists, by looking at each stable state in the state table and changing each input in turn. If the final state after one transition of the input is the same as the final state after a triple transition, then we are safe; if not we have an essential hazard.

* If the design is not completely specified, we may be add constraints to avoid the problem; in the above example, if the next state of B(10) had been a “don’t care” (perhaps because we hadn’t expected it to occur) we should constrain it to be A or B and not let it take the value C.

FM Problems

- Races can be designed out at the logic level, but Essential Hazards must be designed out at the electrical level, e.g.



Consider the circuit example of the master slave flip-flop on the left, this has an essential hazard in that an input change occurring just after the clock goes high could conceivably propagate through the master section and transfer directly to the slave section *before* the clock change has propagated to the close down the slave section.

The circuit on the right has been designed to reduce the risk of this happening. (Yet another reason for preferring the edge-triggered circuit!)

FM Final Assumptions

FM1 **The physical/layout problems must have been solved.**

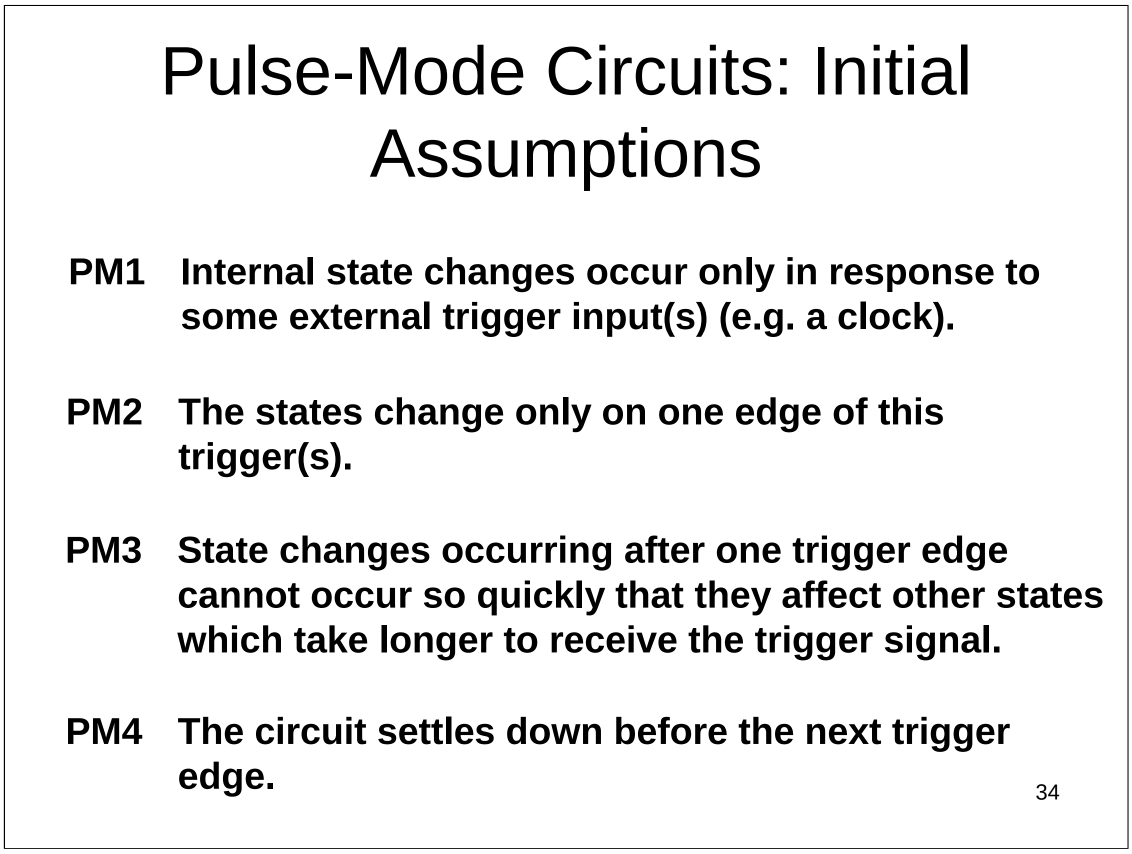
FM2 **Any combinational logic functions must be hazard free.**

FM3 **Only one input signal is allowed to change at any given time.**

FM4 **Critical Races should be designed out logically.**

FM5 **Essential Hazards must be designed out electrically.**

- *So that's why FM design is tricky - avoid it!*



Pulse-Mode Circuits: Initial Assumptions

PM1 **Internal state changes occur only in response to some external trigger input(s) (e.g. a clock).**

PM2 **The states change only on one edge of this trigger(s).**

PM3 **State changes occurring after one trigger edge cannot occur so quickly that they affect other states which take longer to receive the trigger signal.**

PM4 **The circuit settles down before the next trigger edge.**

34

We have seen that it is quite hard work designing fundamental-mode circuits in the sense that we have to start off with some assumptions and yet still have to worry about potential electrical design issues. These all arise from timing problems (more than one input changing, logic hazards, races, essential hazards etc.)

In pulse mode design we go to a higher level in our design hierarchy, imposing more rigid constraints on ourselves in order that we really can forget about the electrical details and concentrate on the sequential digital design.

Pulse mode circuits have two fundamental constraints – PM1 & PM2.

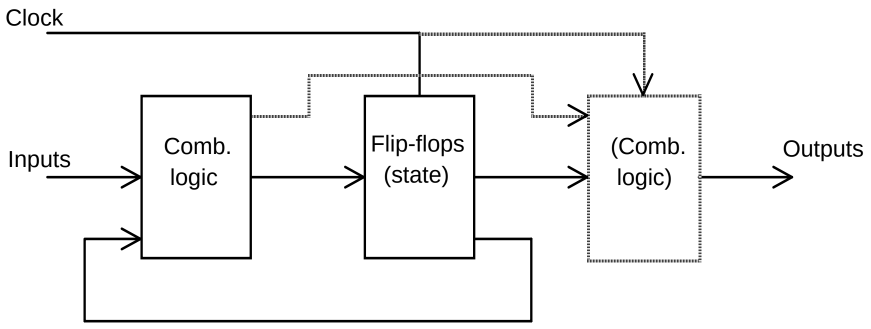
(The trigger input(s) is also called a *pulse input* as opposed to the remaining inputs which are called *level inputs).*

Our electrical problems are then simply to ensure PM3 & PM4.

PM3 is ensured either by tight control of any "clock skew" or by using "non-overlapping clocks". PM4 is ensured by limiting the maximum trigger rate (clock rate) and is eased by using edge-triggered flip-flops. We will look at these again later.

Finite State Machines

PM5 There is only one trigger input:



PM6 Preferably the outputs should come directly from the flip-flops.

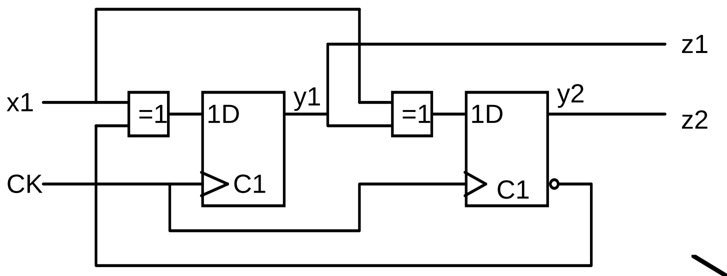
35

Within this framework we have avoided all the messy problems of fundamental mode circuits and can concentrate on nice clean mathematics!

With just one more constraint – PM5 - we can enter the fairy-tale world of *FINITE STATE MACHINES.*

The dotted parts are permitted in the so-called "Mealy machine" but the dotted parts (gating the outputs with the clock and generating outputs from combinations of the state variables) will make it more difficult to guarantee conditions PM3 and PM4. It is therefore strongly recommended that the dotted parts are omitted. This is then a so-called "Moore machine".

Example



Transition table:

(= excitation table with D-types)

<CK>↑

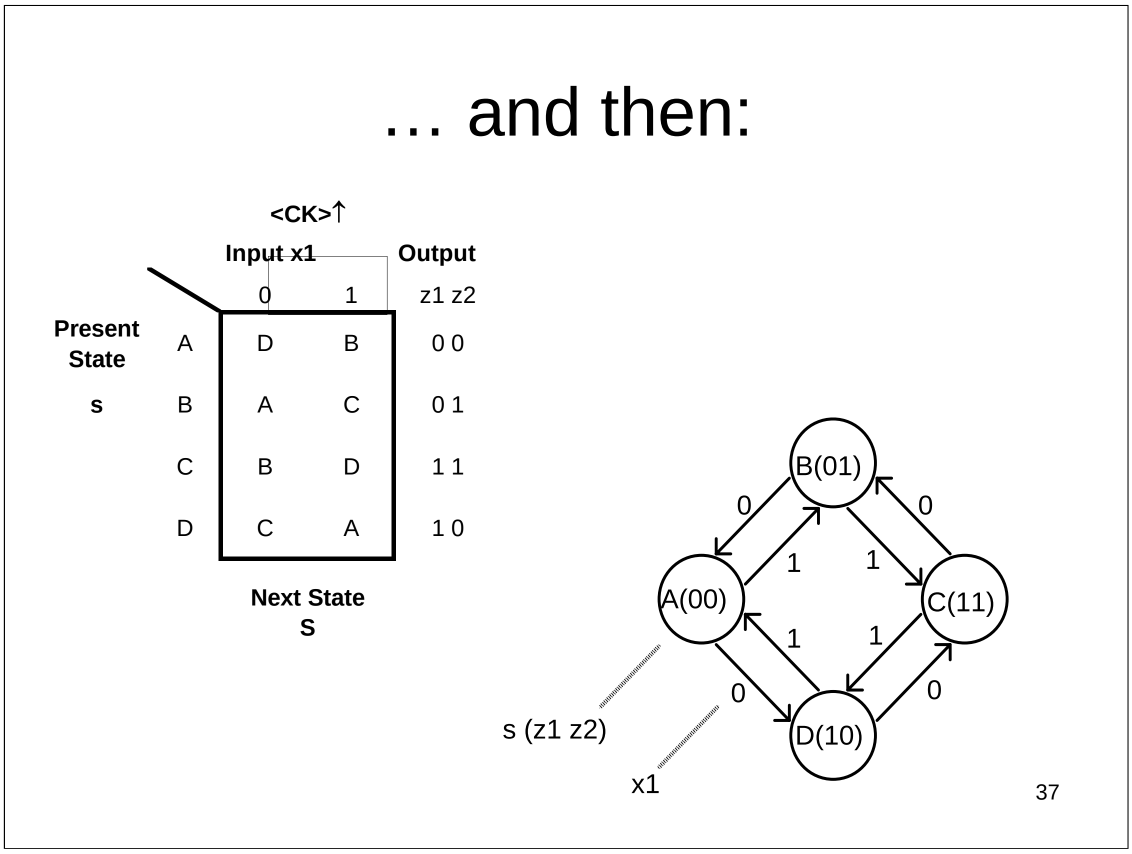
Present State $y_1\ y_2$	Input x1: 0	Input x1: 1	Output z1 z2
0 0	1 0	0 1	0 0
0 1	0 0	1 1	0 1
1 1	0 1	1 0	1 1
1 0	1 1	0 0	1 0

Next State

$Y1^+\ Y2^+$

36

Our analysis now simplifies to considering the state at successive clock intervals and we can ignore all the electrical problems of multiple input changes, logic hazards, races and essential hazards.

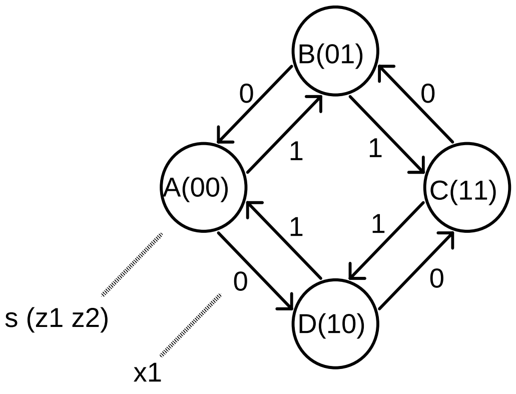


… and then:

<CK>↑

Present State s	Input x1: 0	Input x1: 1	Output z1 z2
A	D	B	0 0
B	A	C	0 1
C	B	D	1 1
D	C	A	1 0

Next State S



37

And once more we abstract to the State Table & Diagram.

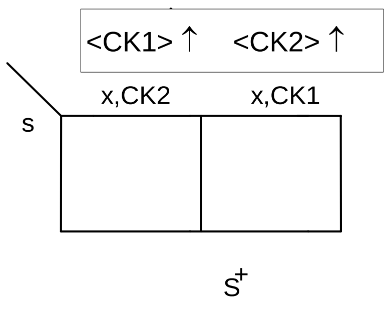
Note all states are stable in pulse-mode design so no need to circle them on the table.

Also (for same reason) there’s no need for the final step of constructing the Flow Table.

Other Pulse-Mode Designs

- Multiple Clocks?

  Multi-port registers
  Pulse counters
  Multi-phase clocks



We quite often relax the single clock assumption for the general class of pulse mode circuits **providing ONLY ONE of them changes at a time!**

Since only one clock changes at a time, a ‘double’ state table can be constructed. Here the left hand side is the state table for CK1 and has the (constant) value of CK2 as an input alongside any ordinary inputs (x); and *vice versa* for CK2 on the right hand side.

“Almost” Pulse-Mode Designs

- Using both edges of the clock (e.g. divide by 3 circuit)
- Multi-step designs (e.g. ripple counter)
- Gated clocks
- ASYNCHRONOUS RESET

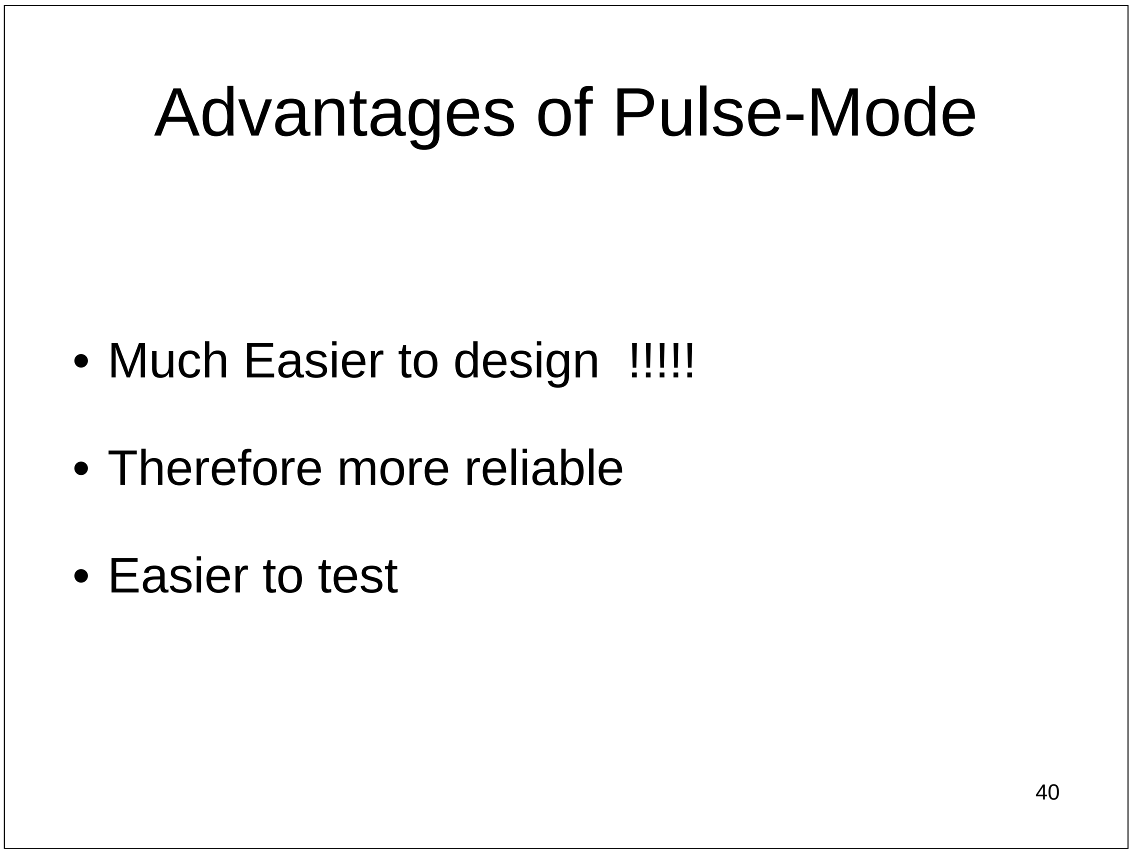
39

Some popular design tricks contravene our strict requisites for pulse-mode circuits but nevertheless can be treated in a very similar manner, **but be wary of electrical issues!**

For the ripple counter, derive State Table entries for the circuit **after** it has settled.

For gated clocks, be very careful about clock skew.

A RESET input is very important but generally we consider it independently of the main design.

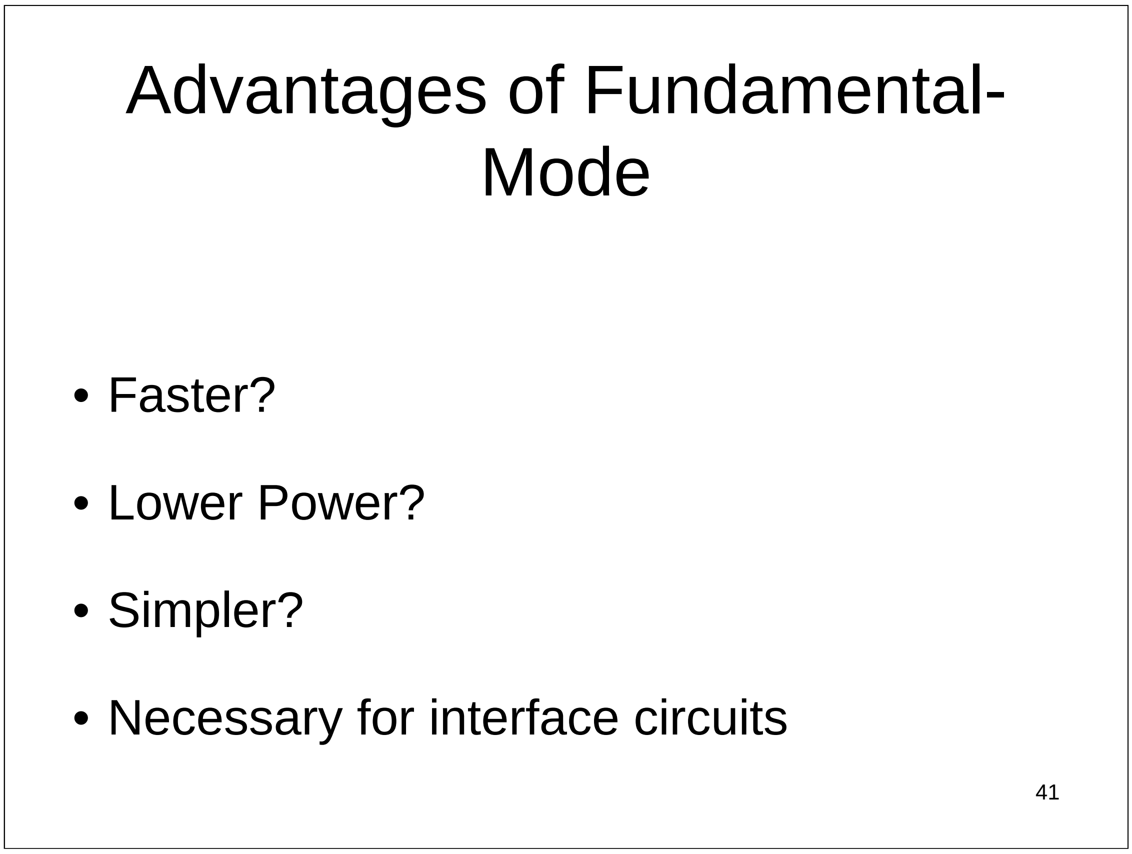


Advantages of Pulse-Mode

- Much Easier to design !!!!!
- Therefore more reliable
- Easier to test

40

Use pulse-mode design unless forced not to.



Advantages of Fundamental-Mode

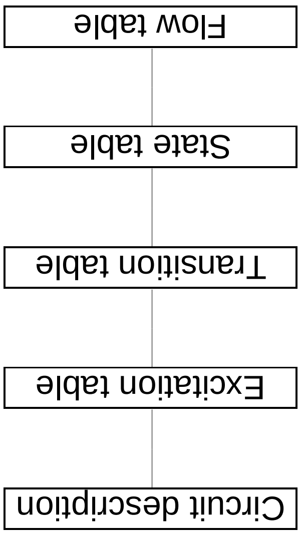
- Faster?
- Lower Power?
- Simpler?
- Necessary for interface circuits

41

Fundamental-mode circuits CAN have some advantages and are used for small parts of circuits which require some special performance. BUT they are always harder to design.

Most likely you will only come across them in interface circuits, but when you do, remember the issues of races, essential hazards, state-variable assignment etc. It is at times like these where you may be having to design quite a small circuit but also having to manage without the full support of CAD tools.

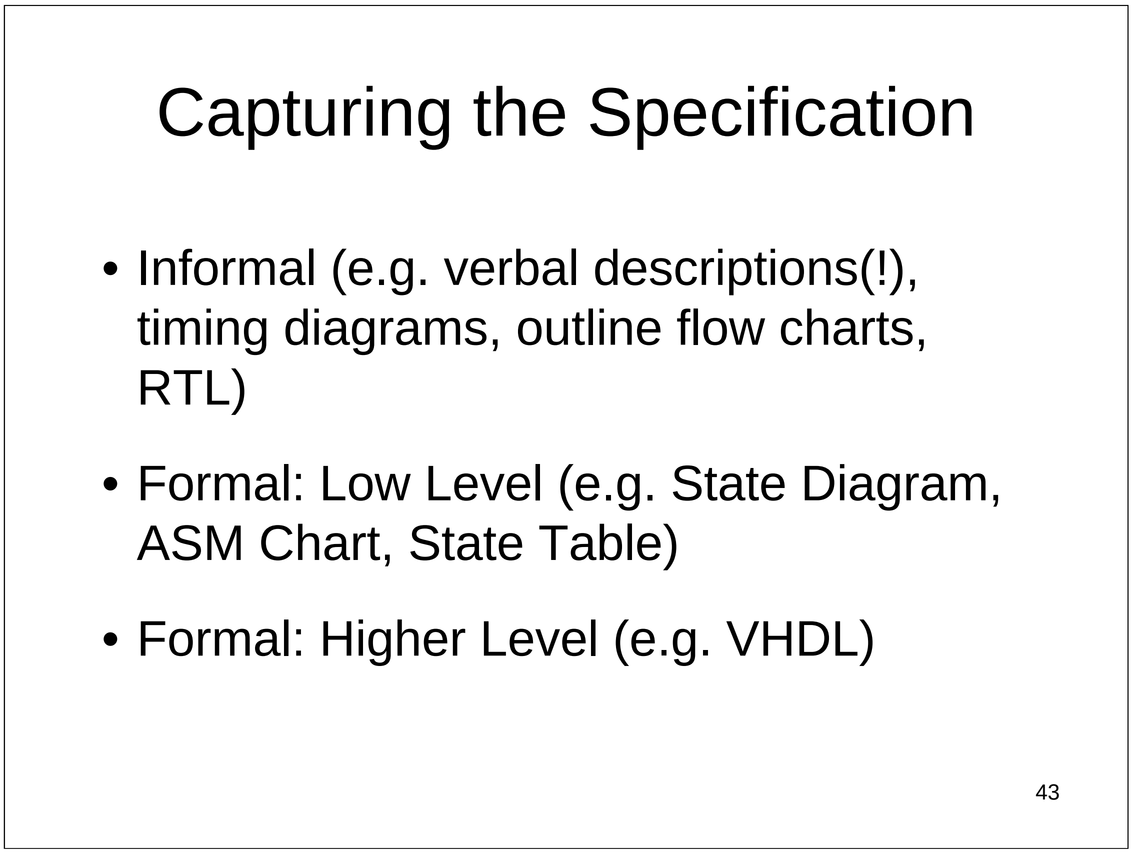
Sequential Logic *Design*



We have listed a series of transformations that are made in analysing a circuit and abstracting it to a Flow Table. The design process is basically tackling these transformations in reverse.

Of course there are lots of choices now so this is generally a more difficult task - the longer we spend and the better our CAD, the better (in principle) we may be able to make the design.

Of course the existence of cheap standard logic circuits (such as the PLA) may make it pointless to worry too much about optimisation.



Capturing the Specification

- Informal (e.g. verbal descriptions(!), timing diagrams, outline flow charts, RTL)
- Formal: Low Level (e.g. State Diagram, ASM Chart, State Table)
- Formal: Higher Level (e.g. VHDL)

The flow (and output) table is a precise statement of our requirements and may well be used as our formal specification. It is more likely, however, that the original expression of the problem will be less formal.

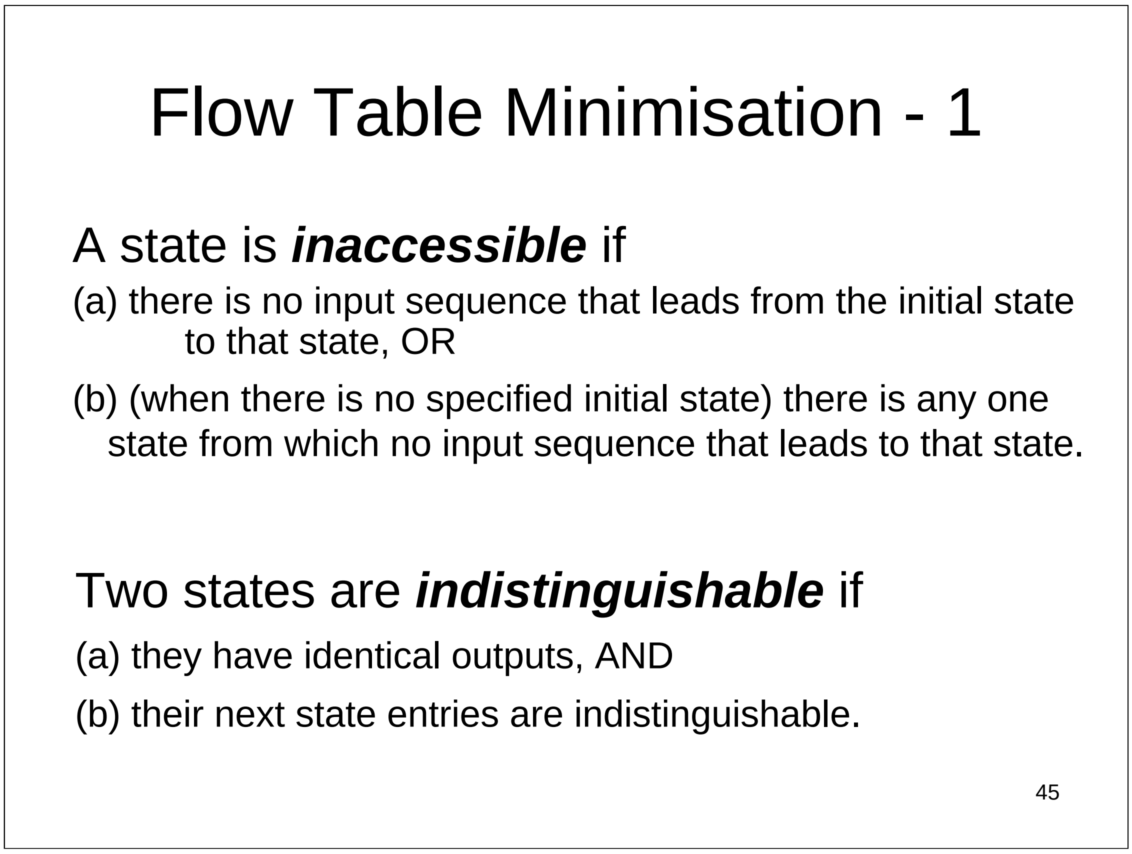
The sooner that we can a “formal” description the better.

Design

- Check the spec. (e.g. simulate)
- Minimize the number of states
- “Race-free” state variable assignment (may change the state table which achieves desired flow table)
- Derive excitation functions (use D-types unless you know better)
- Map to hardware

State minimisation is becoming less important with programmable logic.

Race-free assignment is irrelevant to pulse-mode design.



Flow Table Minimisation - 1

A state is ***inaccessible*** if

(a) there is no input sequence that leads from the initial state to that state, OR

(b) (when there is no specified initial state) there is any one state from which no input sequence that leads to that state.

Two states are ***indistinguishable*** if

(a) they have identical outputs, AND

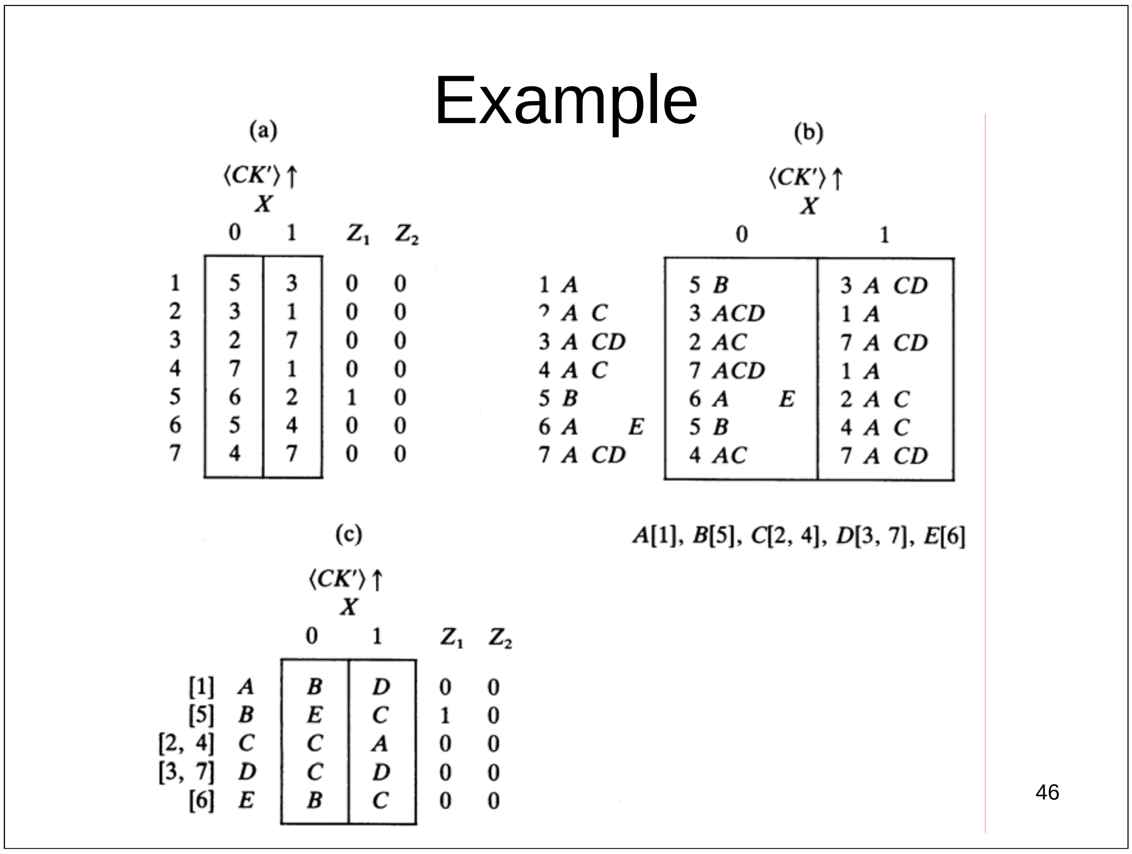
(b) their next state entries are indistinguishable.

45

Just in case you do need to minimise the number of states, this is how you go about it. (Most likely you will use CAD tools to do this for you in practise).

To start with we assume that the flow table is completely specified.

The presence of inaccessible states in the flow table is probably an indication that the specification is wrong and this should be checked first. If there is no mistake, the first thing we will do is to eliminate the inaccessible states anyway, since they obviously play no useful part in the functional design.



Example

(a)

$\langle CK' \rangle \uparrow$

X

	0	1	Z_1	Z_2
1	5	3	0	0
2	3	1	0	0
3	2	7	0	0
4	7	1	0	0
5	6	2	1	0
6	5	4	0	0
7	4	7	0	0

(b)

$\langle CK' \rangle \uparrow$

X

	0	1
1 *A*	5 *B*	3 *A CD*
2 *A C*	3 *ACD*	1 *A*
3 *A CD*	2 *AC*	7 *A CD*
4 *A C*	7 *ACD*	1 *A*
5 *B*	6 *A E*	2 *A C*
6 *A E*	5 *B*	4 *A C*
7 *A CD*	4 *AC*	7 *A CD*

A[1], *B*[5], *C*[2, 4], *D*[3, 7], *E*[6]

(c)

$\langle CK' \rangle \uparrow$

X

		0	1	Z_1	Z_2
[1]	*A*	*B*	*D*	0	0
[5]	*B*	*E*	*C*	1	0
[2, 4]	*C*	*C*	*A*	0	0
[3, 7]	*D*	*C*	*D*	0	0
[6]	*E*	*B*	*C*	0	0

46

We can minimise the number of states by examining the flow table to see if any of its states are indistinguishable from one another; if they are, we can then combine them into a single state.

McCluskey describes two fairly obvious procedures for doing this. One procedure is:

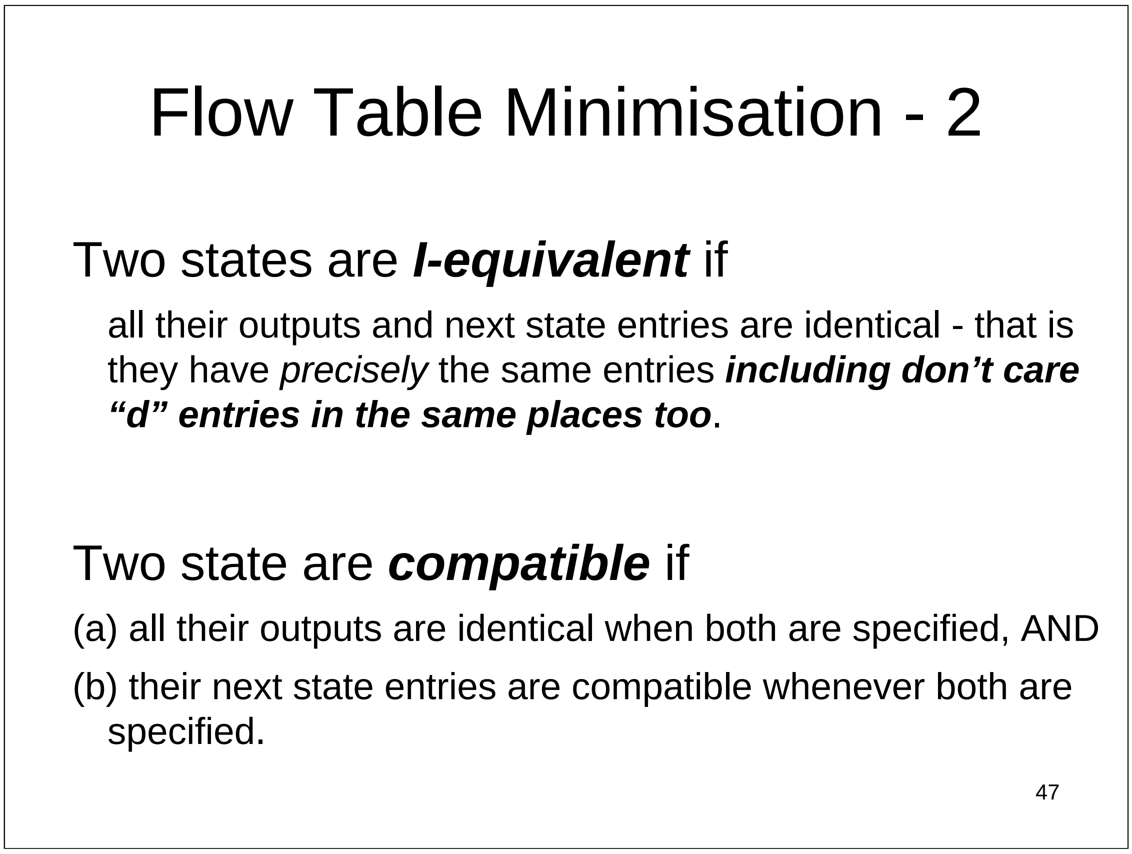
(i) Label groups of states with identical outputs;

(ii) Examine the next state entries of each group;

(iii) Subdivide groups with different next state entries and re-label the subdivisions;

(iv) Continue steps (ii) and (iii) until all groups have same next state entries;

(v) The final groups are the new states.



Flow Table Minimisation - 2

Two states are ***I-equivalent*** if

all their outputs and next state entries are identical - that is they have *precisely* the same entries ***including don't care "d" entries in the same places too***.

Two state are ***compatible*** if

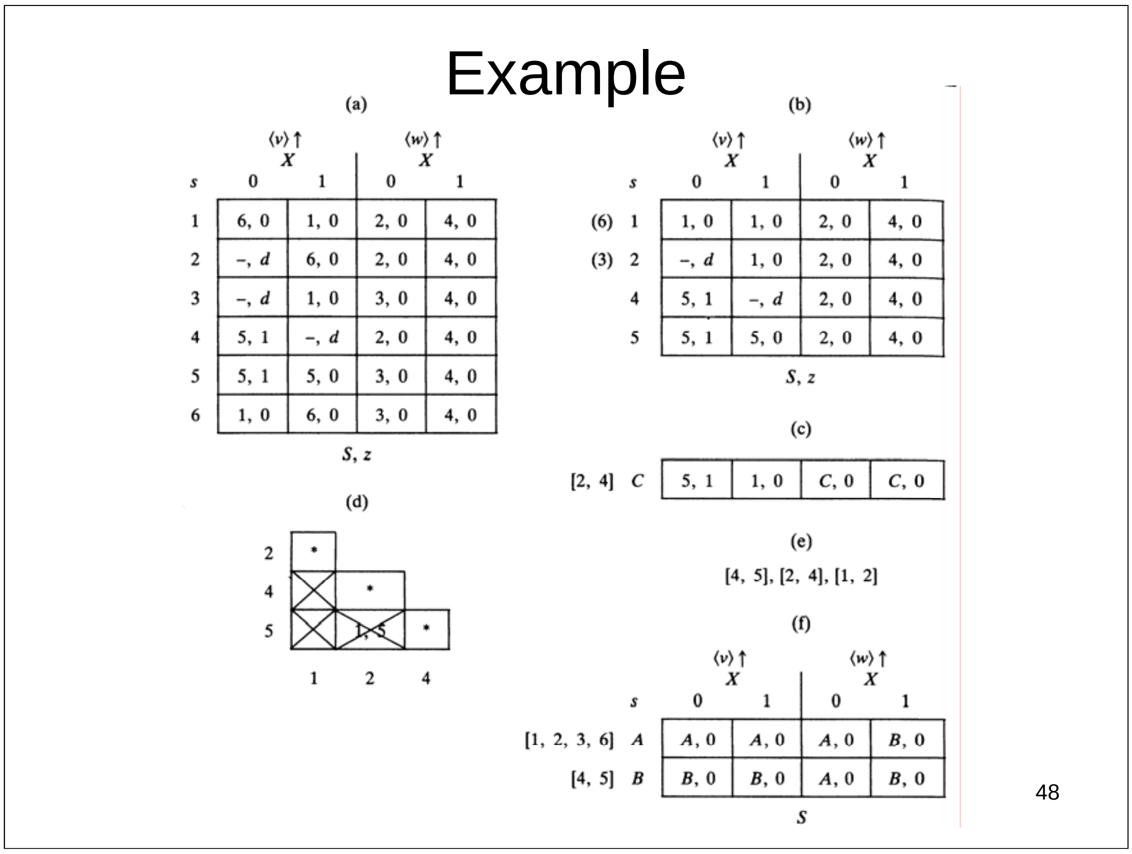
(a) all their outputs are identical when both are specified, AND

(b) their next state entries are compatible whenever both are specified.

47

For those of you who enjoy this sort of thing (i.e. I won't set an exam question on this slide), we need to make a few changes in the case that the flow table is not completely specified.

N.B. for compatibility [unlike I-equivalence], if one state has a don't care "d" entry, it doesn't matter what the other the other state has there.



Example

(a)

		⟨v⟩↑ X		⟨w⟩↑ X	
	s	0	1	0	1
	1	6, 0	1, 0	2, 0	4, 0
	2	–, d	6, 0	2, 0	4, 0
	3	–, d	1, 0	3, 0	4, 0
	4	5, 1	–, d	2, 0	4, 0
	5	5, 1	5, 0	3, 0	4, 0
	6	1, 0	6, 0	3, 0	4, 0

S, z

(b)

		⟨v⟩↑ X		⟨w⟩↑ X	
	s	0	1	0	1
(6)	1	1, 0	1, 0	2, 0	4, 0
(3)	2	–, d	1, 0	2, 0	4, 0
	4	5, 1	–, d	2, 0	4, 0
	5	5, 1	5, 0	2, 0	4, 0

S, z

(c)

[2, 4]	C	5, 1	1, 0	C, 0	C, 0

(d)

2	*		
4	✗	*	
5	✗	~~1, 5~~	*
	1	2	4

(e)

[4, 5], [2, 4], [1, 2]

(f)

		⟨v⟩↑ X		⟨w⟩↑ X	
	s	0	1	0	1
[1, 2, 3, 6]	A	A, 0	A, 0	A, 0	B, 0
[4, 5]	B	B, 0	B, 0	A, 0	B, 0

S

48

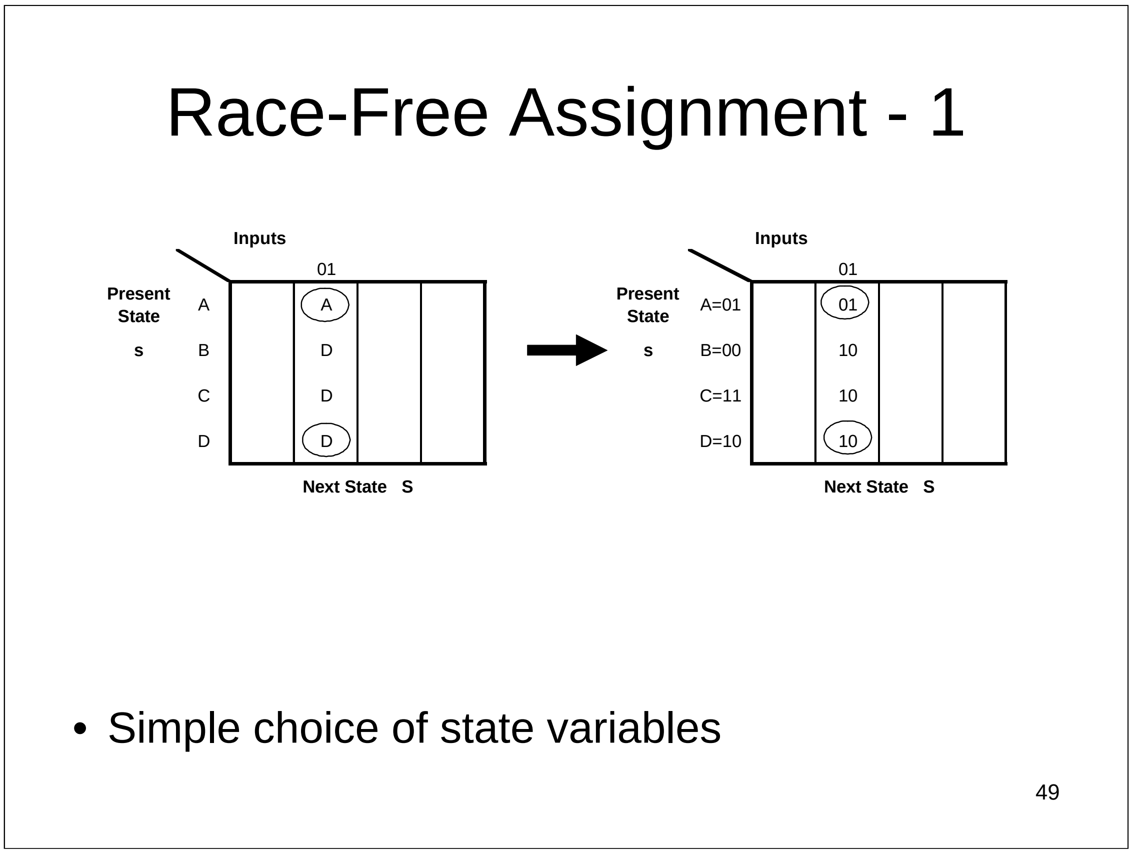
Reducing the flow graphs then comes down to the following steps

(i) Test every pair of states and combine any that are I-equivalent;

(ii) Compute pairs of compatible states in the same way as described above for pairs of indistinguishable states;

(iii) Compute Maximal Compatibility Classes - sets of states which are all compatible with each other;

(iv) Select a set of MCCs which cover all the states; these are our new states.



Race-Free Assignment - 1

Present State s \ Inputs		01		
A		$\textcircled{\scriptsize A}$		
B		D		
C		D		
D		$\textcircled{\scriptsize D}$		

Next State S

$\Rightarrow$

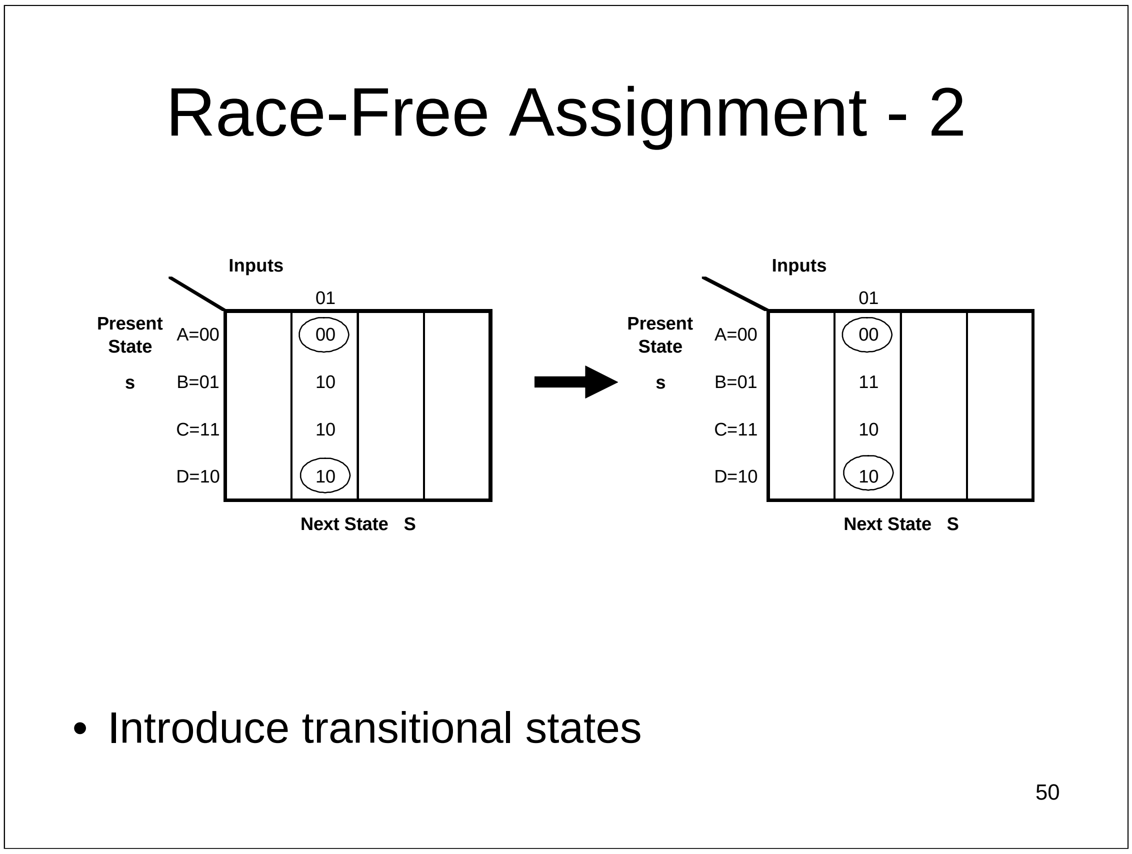
Present State s \ Inputs		01		
A=01		$\textcircled{\scriptsize 01}$		
B=00		10		
C=11		10		
D=10		$\textcircled{\scriptsize 10}$		

Next State S

- Simple choice of state variables

49

For fundamental-mode circuits, we need to make a race-free assignment of the state-variables. I.E. only one state variable changes for each transition. In this design, we see it is possible to make an appropriate choice.



Race-Free Assignment - 2

Inputs

Present State s		01		
A=00		00		
B=01		10		
C=11		10		
D=10		10		

Next State S

$\Rightarrow$

Inputs

Present State s		01		
A=00		00		
B=01		11		
C=11		10		
D=10		10		

Next State S

- Introduce transitional states

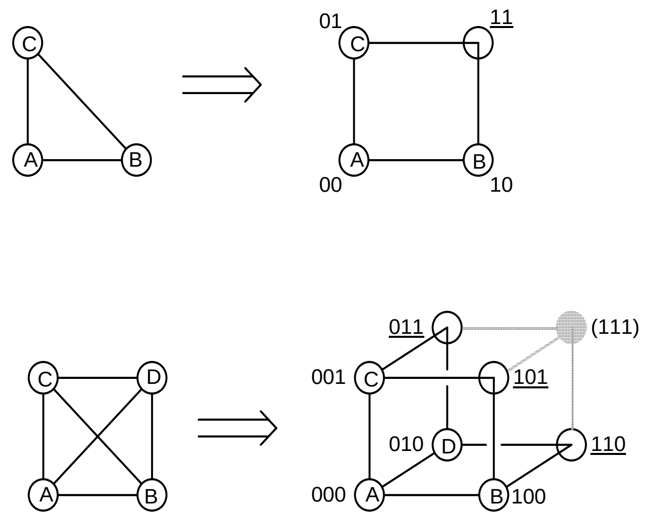
50

Where this is not possible, we try to modify the state table whilst retaining the specified flow table entries. I.E. we try to send the state via a route where only one variable changes at each step. We may be able to do this when there are some unspecified entries in that column of the flow table or when the same target state already exists somewhere else in the column.

Be careful about generating hazards at the output if the "transitional" state has a different output to that of the final state and be aware that the circuit will take longer to settle down.

This is the opposite process to that in which we earlier reduced the sate table to the flow table.

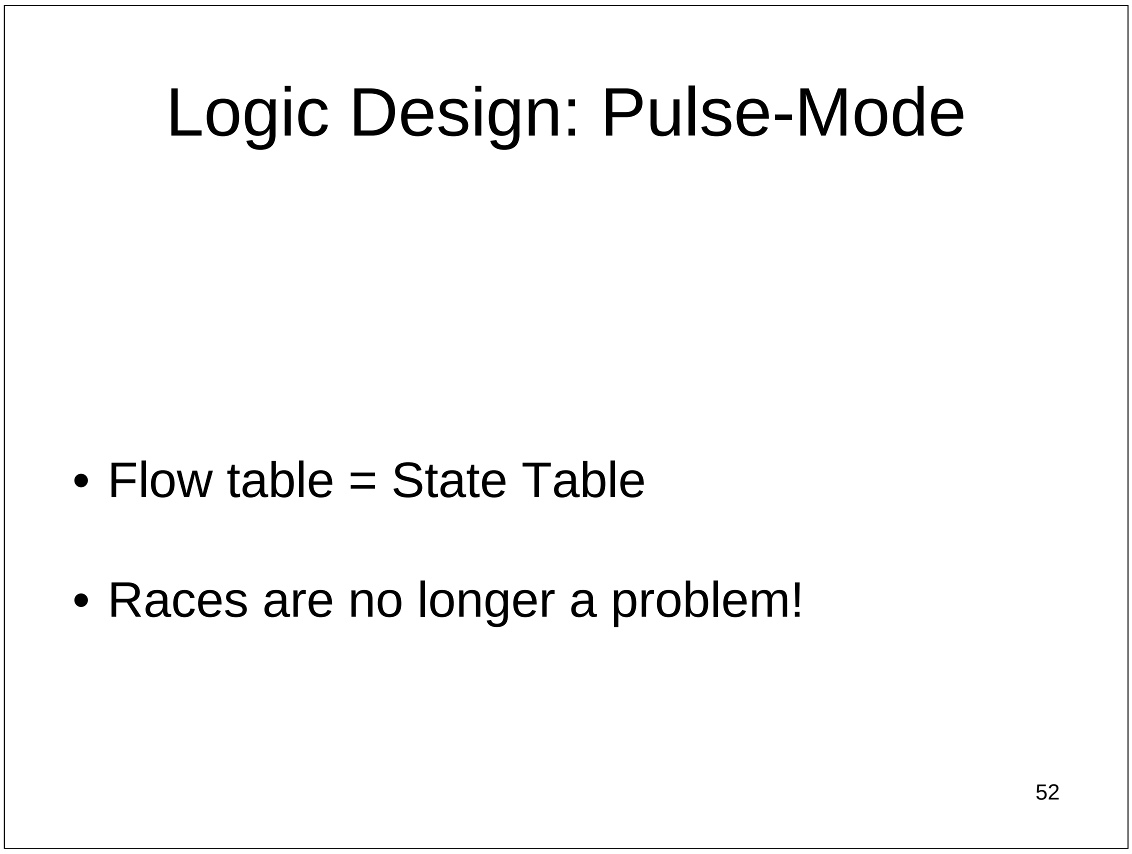
Race-Free Assignment - 3



51

If those approaches don’t work, we have to introduce one or more *new* transitional states. This is quite a tough problem and the subject of lots of theoretical papers. Commonly these are framed as a cube covering problem as shown below for two simple cases. The extra transitional states are shown underlined.

Yet another good reason why we would prefer to avoid the fundamental mode approach for anything but the simplest designs!

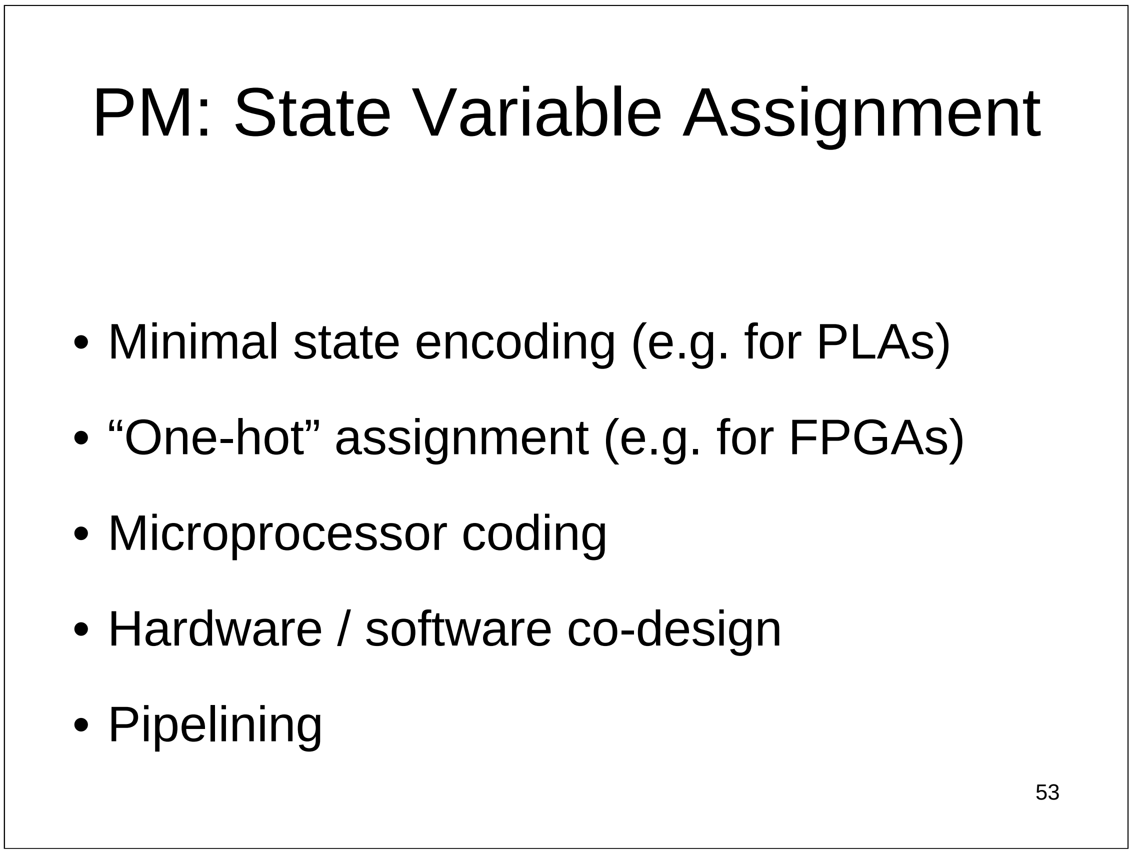


Logic Design: Pulse-Mode

- Flow table = State Table
- Races are no longer a problem!

52

Given the relative ease of designing pulse-mode circuits, we are quite likely to design big circuits where the issues of state assignment and minimisation get swamped by the actual implementation. Hence state variable assignment in pulse-mode circuits is inextricably linked to the chosen devices and the available synthesis tools as we shall see:



PM: State Variable Assignment

- Minimal state encoding (e.g. for PLAs)
- “One-hot” assignment (e.g. for FPGAs)
- Microprocessor coding
- Hardware / software co-design
- Pipelining

53

Here are some SVA / synthesis approaches:

Minimal state encoding

54

This is probably the most obvious approach; for *n* states, we choose *m* flip-flops where *m* is the smallest number for which $n < 2^m$. This is a fine choice if we are constructing the system out of discrete flip-flops and gates but this situation is increasingly rare. However, this architecture is still a good choice if we are designing with PLDs where you get an abundance of combinational logic for every flip-flop and where the size of the device is strongly related to the number of flip-flops.

In the “good old days” of discrete gates, you would also have agonised about what sort of flip-flops to use and done some logic minimization. With PLDs the flip-flops are usually included and the scope for logic minimisation more limited because you are implementing the sum-of-products.

I’ll have a few other things to say about PLDs later.

“One-hot” assignment

55

Later in your time here you learnt about RTL and found that an easy way to implement it was to have one flip-flop for every line of code (*n* flip-flops = *n* states) and to pass a “token” along the chain of flip-flops activating the logic for the appropriate line of code.

Of course, this uses a lot more flip-flops, but it turns out that this architecture is usually a good choice if you are implementing the design on an FPGA. The Xilinx devices use smallish programmable cells with fairly limited logic but two flip-flops built-in.

I’ll have a few other things to say about FPGAs later.

Microprocessor coding

56

States do not have to be *bound* to the same hardware all the time, but flip-flops and (preferably) whole *registers* can be redefined as we execute our finite state machine. This is the basis of writing a program to run on the ubiquitous microprocessor.

It is obvious that we could have taken our RTL code and compiled it to run on any old microprocessor instead. Of course the beauty is that microprocessors can be reused for many different applications and that they are so widely used that their price/performance ratio is extraordinarily high.

The "down side" is that our specific design may run much slower on a even the latest microprocessor than it would in dedicated hardware. The microprocessor is itself an amalgam of several distinct architectural styles; ROM, RAM, ALU etc. which we could elaborate on further.

Hardware / software co-design

57

We saw above that a design specified in something like RTL could be implemented in different architectures. This has lead to a lot of interest in recent years in software / hardware co-design where the code is compiled either onto a processor or onto an FPGA according to how fast we want that bit to go. Thus a lot of the slower portions of the design would finish up running on the processor whilst some critical functions would finish up running rather faster on the hardware. We may even change our split dynamically if the situation demands.

Pipelining

58

Of course, there are lots of other things we can do but the biggest gap left to plug is architectures for the very highest performance systems. Many of these are dealing with huge amounts of data and doing some digital signal processing (e.g. image compression for digital TV). You can buy “DSPs” but if these won’t do the other key thing you can think about is to pipeline the design where each piece of hardware only calculates a small part of the overall function and passes the intermediate result on to the next piece.

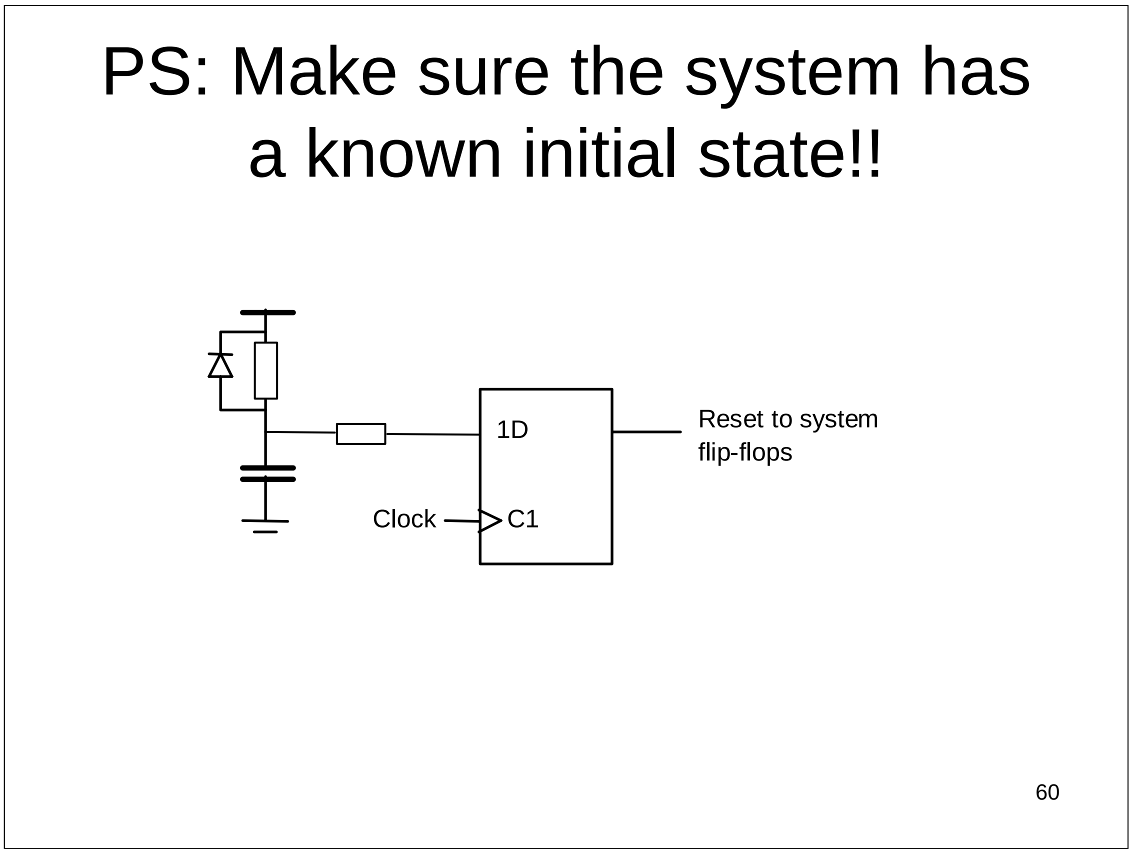
PS: Unassigned States?

59

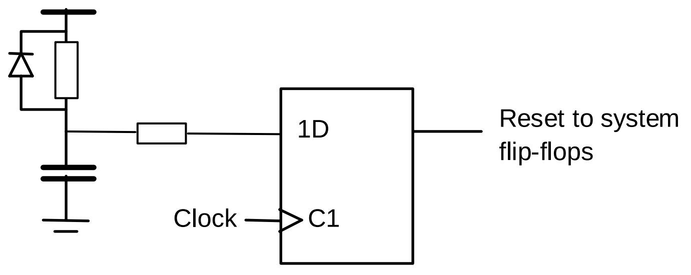
If there are *n* latches or flip-flops there are 2*n* possible states. If some of these are unused we ought to think about what might happen if some electrical disturbance pushes our design into those states. With minimum state encoding you may actually be able to examine all the unused states and see what happens if the system finds itself in them (as you did in the second year). You can even constrain the design so that the system always does something sensible (e.g. goes to an ‘error’ or a ‘reset’ state). You did this in first year.

With non-minimum encoding they will probably be so many unused states that this is quite unrealistic. In these cases we will probably want to look at some other mechanism for trapping erroneous use of these states, e.g. we may use a “watchdog timer” which checks that the state machine cycles around within an acceptable range of cycle times.

If all else fails, reach for the reset button …



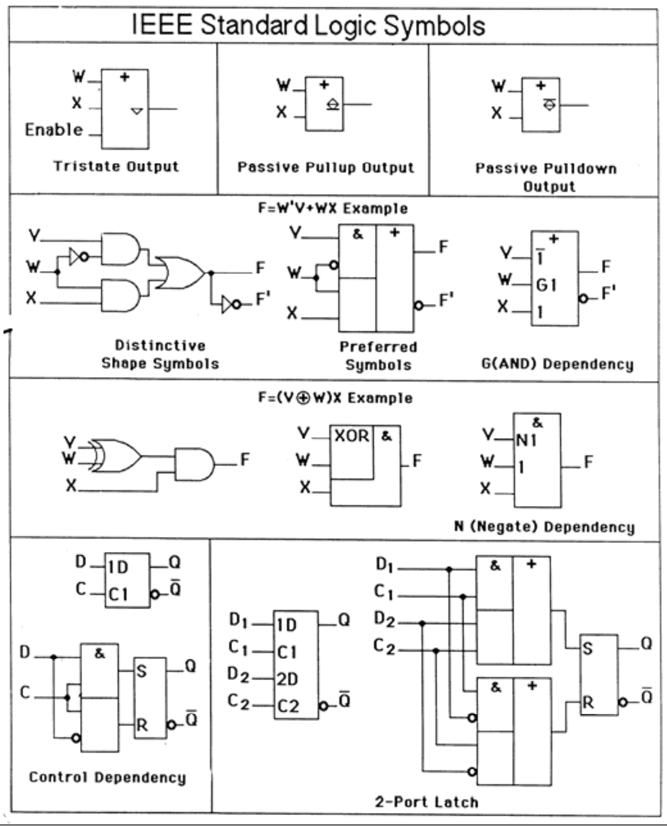
PS: Make sure the system has a known initial state!!



60

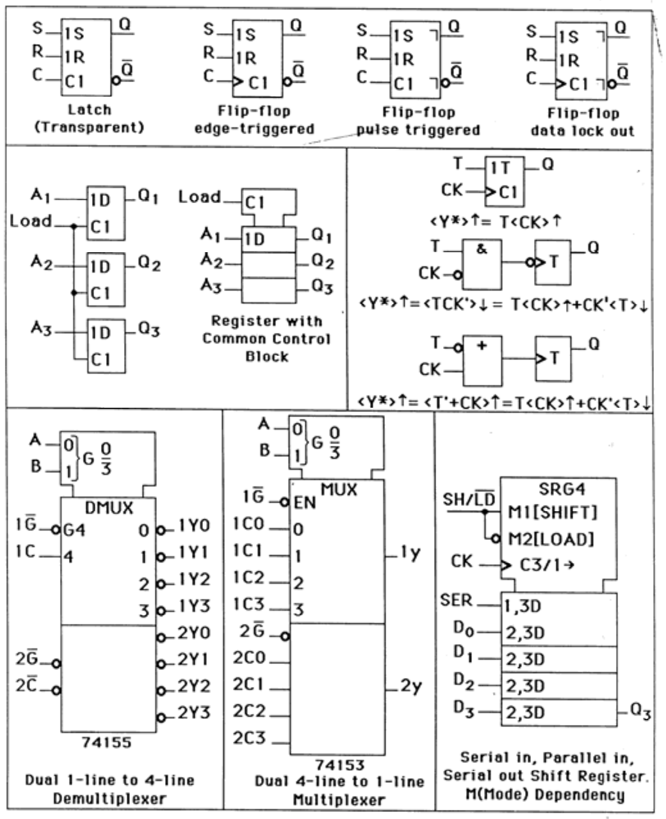
In most designs we would want to specify some initial state for the system and incorporate reset logic so that we could guarantee starting in it in a known state. Usually we would automatically perform a reset operation on power up.

If no initial state is specified, it is probably worth checking that the specification is right. Even if it is correct, it is almost certainly worth specifying an initial state anyway in order to simplify testing (see later lecture). The initialisation will normally be initiated asynchronously (e.g. by a "reset" button or by an RC delay on restoring the power supply) but it should normally be taken off synchronously in order to be sure that the system operates properly thereafter. A possible power-on reset circuit is shown here.



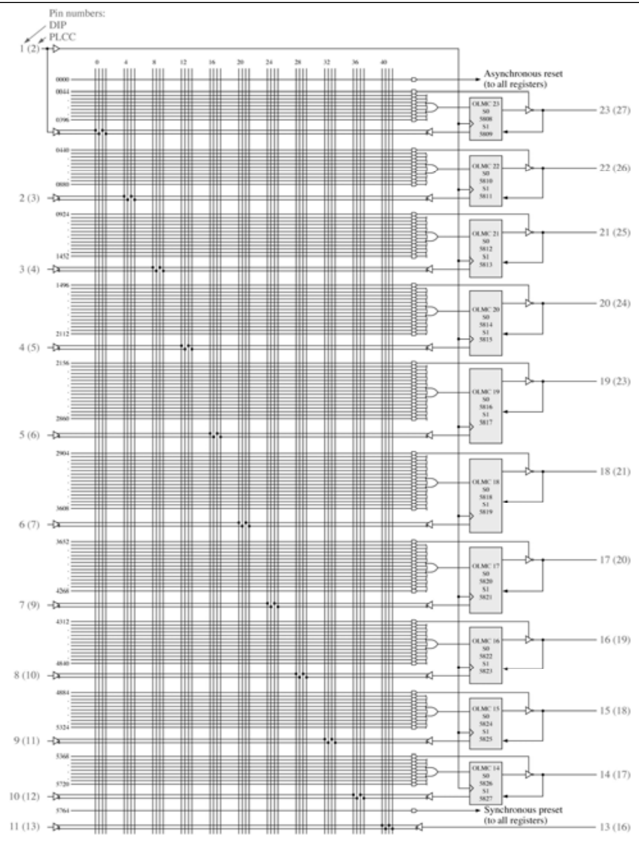
61

The “new” IEEE symbols. The “old” distinctive shapes are still in widespread use.


Latch (Transparent)
Flip-flop edge-triggered
Flip-flop pulse triggered
Flip-flop data lock out
Register with Common Control Block
<Y*>↑= T<CK>↑
<Y*>↑=<TCK'>↓ = T<CK>↑+CK'<T>↓
<Y*>↑= <T'+CK>↑=T<CK>↑+CK'<T>↓
DMUX
74155
Dual 1-line to 4-line Demultiplexer
MUX
74153
Dual 4-line to 1-line Multiplexer
SRG4
M1[SHIFT]
M2[LOAD]
C3/1→
Serial in, Parallel in, Serial out Shift Register. M(Mode) Dependency

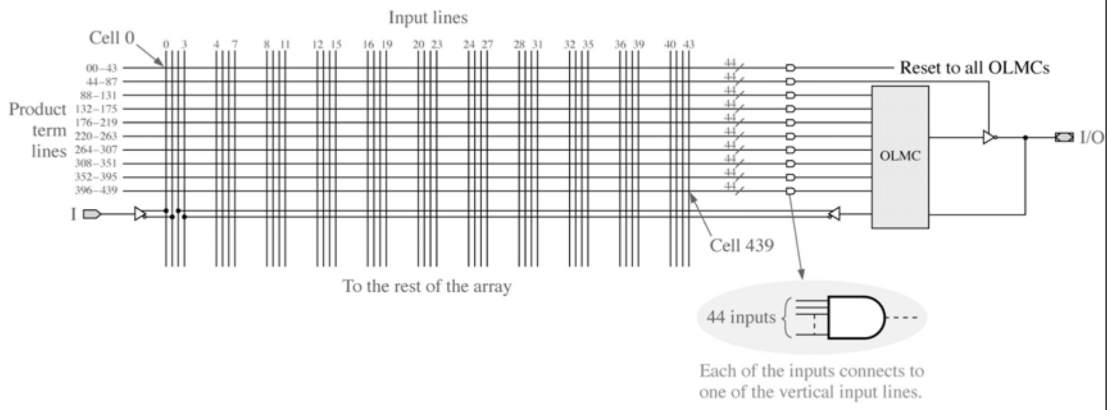
1a Some Programmable Devices

PLD

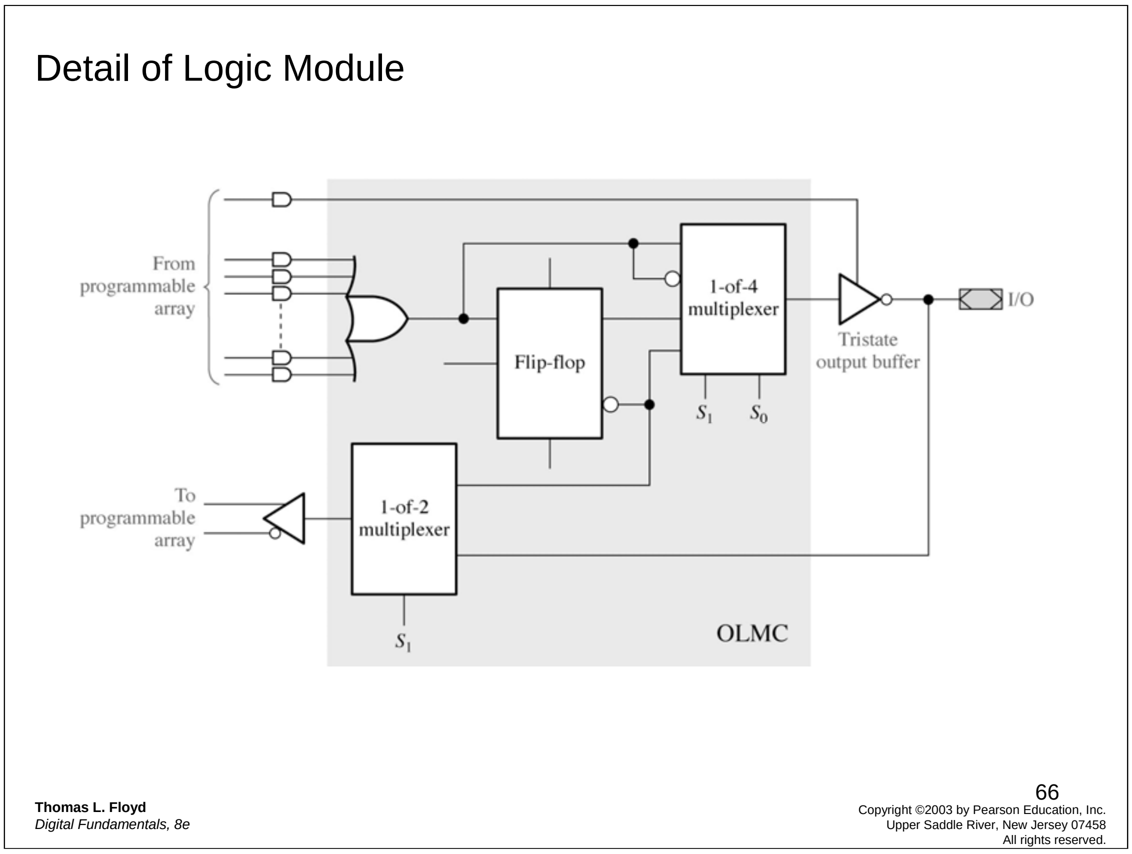


A typical small PLD …

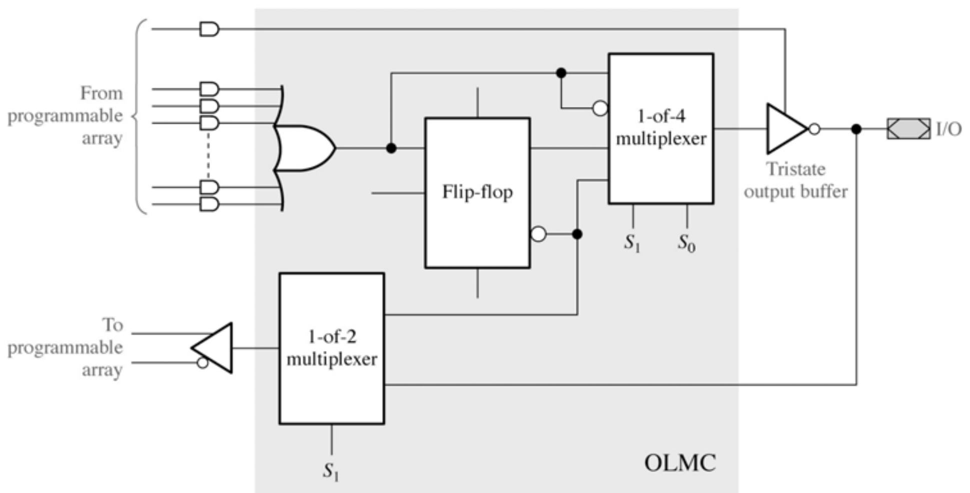
Detail of Array



… note the wide AND gates feeding fixed OR (that’s a “PAL”) …



Detail of Logic Module

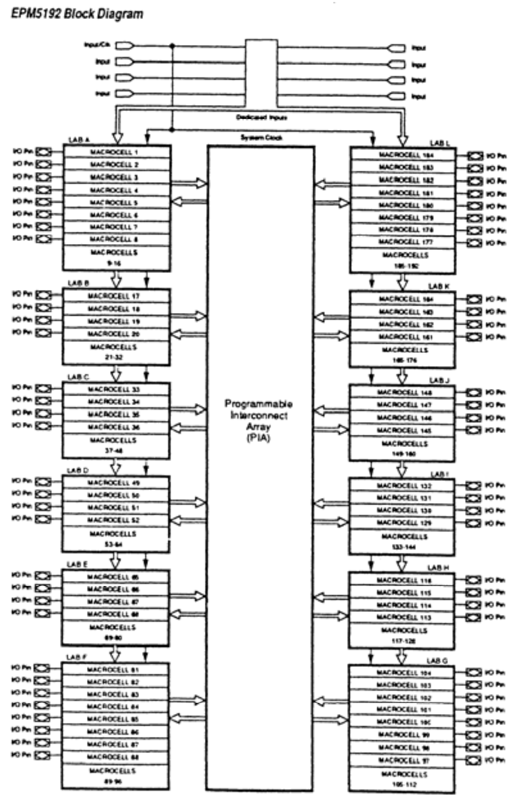


66

Thomas L. Floyd
Digital Fundamentals, 8e

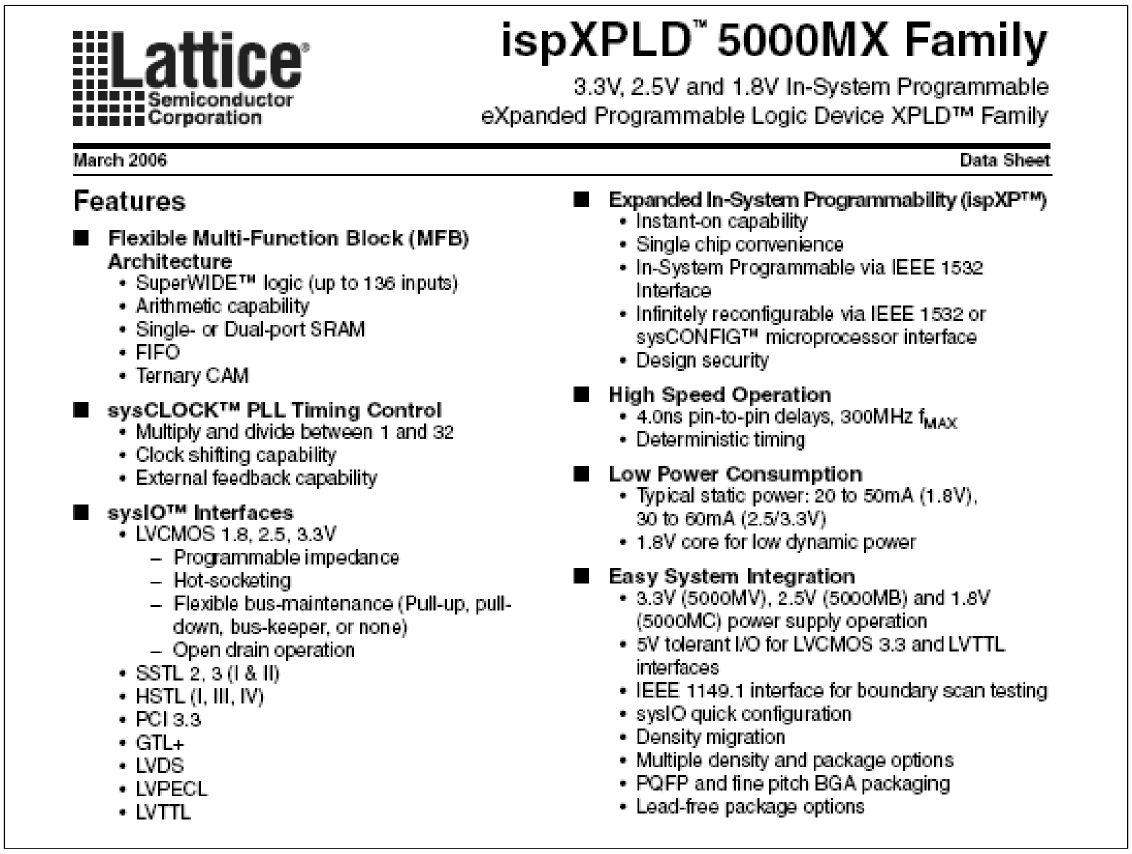
… the output logic & flip-flop …

CPLD



67

… and if you put a heap of such PLDs onto a chip with a programmable crossbar switch you get a Complex Programmable Logic Device (CPLD)).



ispXPLD™ 5000MX Family

3.3V, 2.5V and 1.8V In-System Programmable
eXpanded Programmable Logic Device XPLD™ Family

March 2006 — Data Sheet

Features

- **Flexible Multi-Function Block (MFB) Architecture**
  - SuperWIDE™ logic (up to 136 inputs)
  - Arithmetic capability
  - Single- or Dual-port SRAM
  - FIFO
  - Ternary CAM
- **sysCLOCK™ PLL Timing Control**
  - Multiply and divide between 1 and 32
  - Clock shifting capability
  - External feedback capability
- **sysIO™ Interfaces**
  - LVCMOS 1.8, 2.5, 3.3V
    - Programmable impedance
    - Hot-socketing
    - Flexible bus-maintenance (Pull-up, pull-down, bus-keeper, or none)
    - Open drain operation
  - SSTL 2, 3 (I & II)
  - HSTL (I, III, IV)
  - PCI 3.3
  - GTL+
  - LVDS
  - LVPECL
  - LVTTL
- **Expanded In-System Programmability (ispXP™)**
  - Instant-on capability
  - Single chip convenience
  - In-System Programmable via IEEE 1532 Interface
  - Infinitely reconfigurable via IEEE 1532 or sysCONFIG™ microprocessor interface
  - Design security
- **High Speed Operation**
  - 4.0ns pin-to-pin delays, 300MHz f_{MAX}
  - Deterministic timing
- **Low Power Consumption**
  - Typical static power: 20 to 50mA (1.8V), 30 to 60mA (2.5/3.3V)
  - 1.8V core for low dynamic power
- **Easy System Integration**
  - 3.3V (5000MV), 2.5V (5000MB) and 1.8V (5000MC) power supply operation
  - 5V tolerant I/O for LVCMOS 3.3 and LVTTL interfaces
  - IEEE 1149.1 interface for boundary scan testing
  - sysIO quick configuration
  - Density migration
  - Multiple density and package options
  - PQFP and fine pitch BGA packaging
  - Lead-free package options

Modern CPLDs have a host of other features …

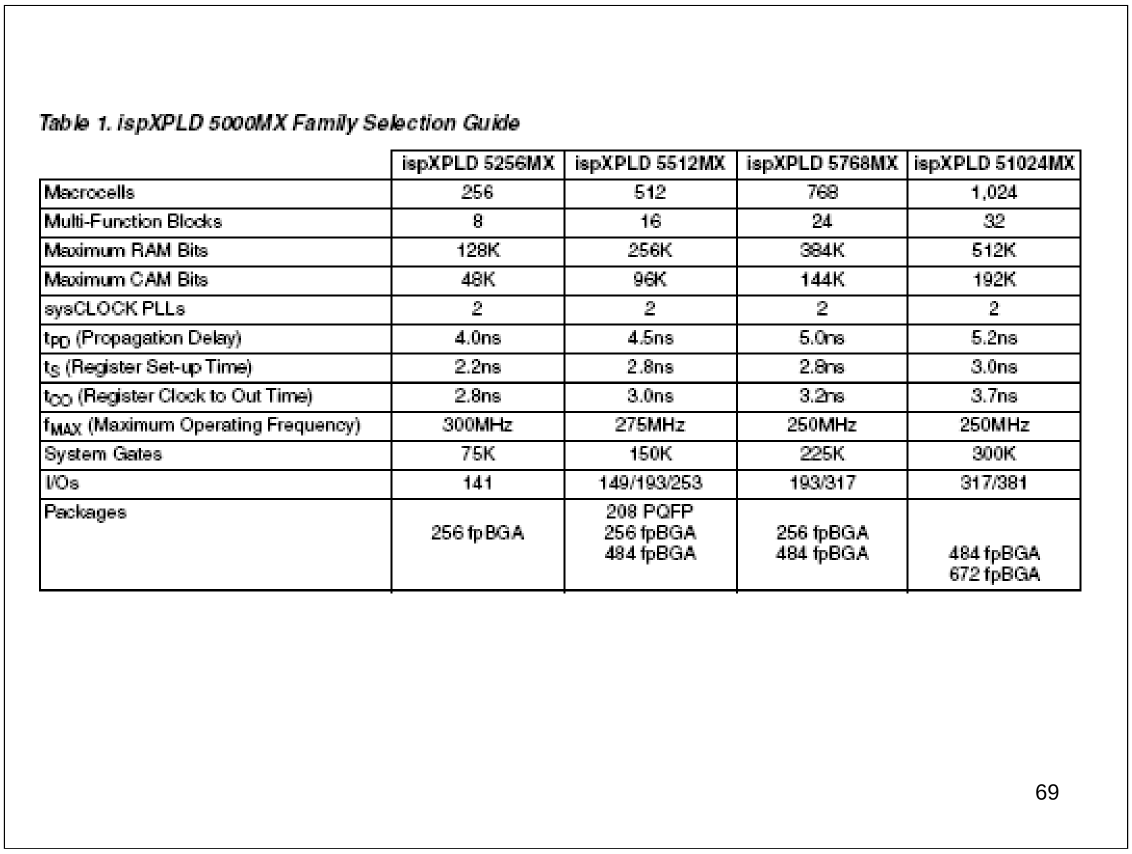
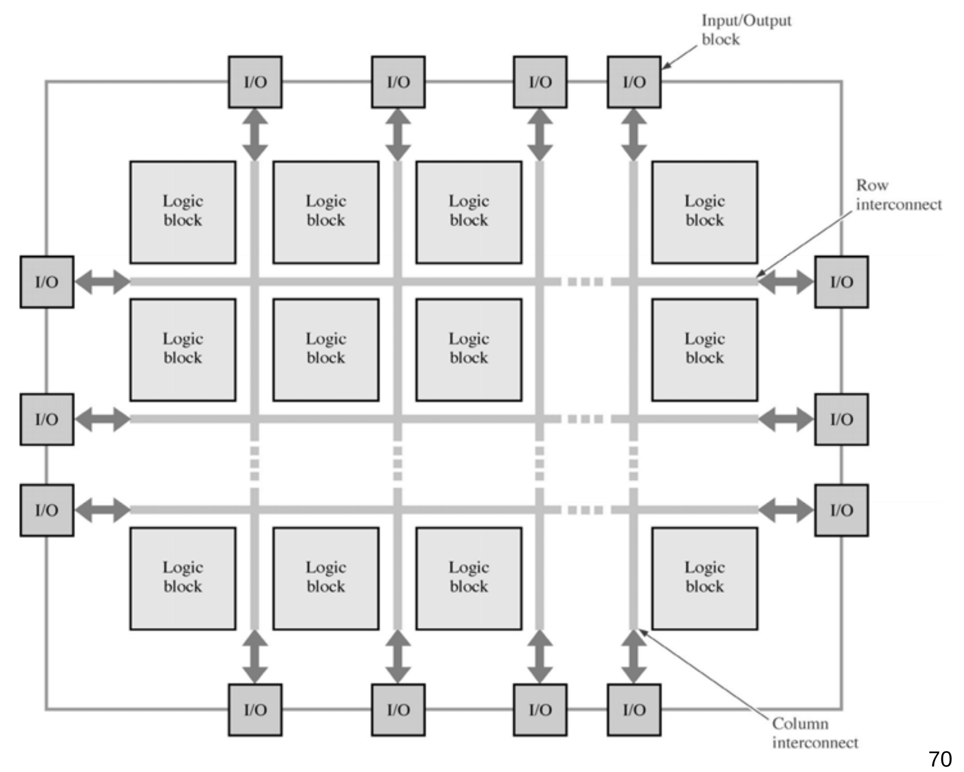


Table 1. ispXPLD 5000MX Family Selection Guide

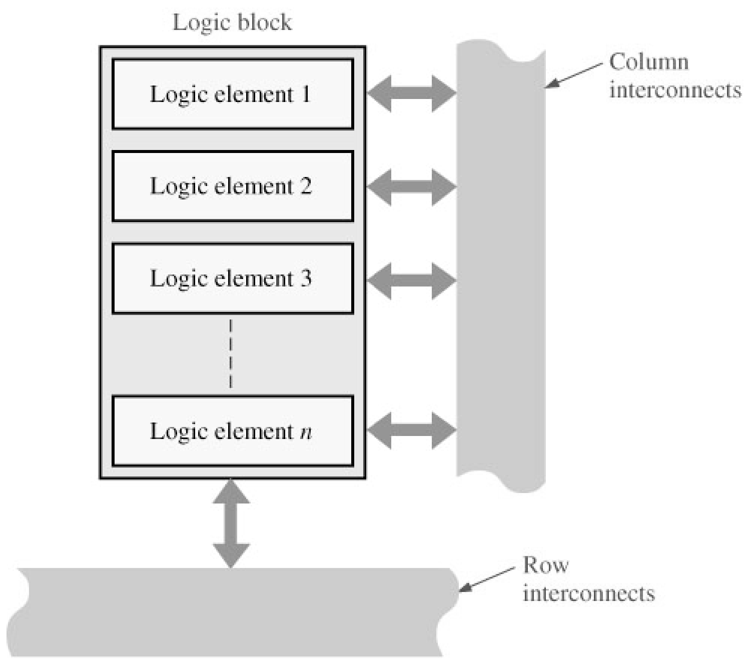
	ispXPLD 5256MX	ispXPLD 5512MX	ispXPLD 5768MX	ispXPLD 51024MX
Macrocells	256	512	768	1,024
Multi-Function Blocks	8	16	24	32
Maximum RAM Bits	128K	256K	384K	512K
Maximum CAM Bits	48K	96K	144K	192K
sysCLOCK PLLs	2	2	2	2
t_{PD} (Propagation Delay)	4.0ns	4.5ns	5.0ns	5.2ns
t_S (Register Set-up Time)	2.2ns	2.8ns	2.8ns	3.0ns
t_{CO} (Register Clock to Out Time)	2.8ns	3.0ns	3.2ns	3.7ns
f_{MAX} (Maximum Operating Frequency)	300MHz	275MHz	250MHz	250MHz
System Gates	75K	150K	225K	300K
I/Os	141	149/193/253	193/317	317/381
Packages	256 fpBGA	208 PQFP 256 fpBGA 484 fpBGA	256 fpBGA 484 fpBGA	484 fpBGA 672 fpBGA

… and are massive!

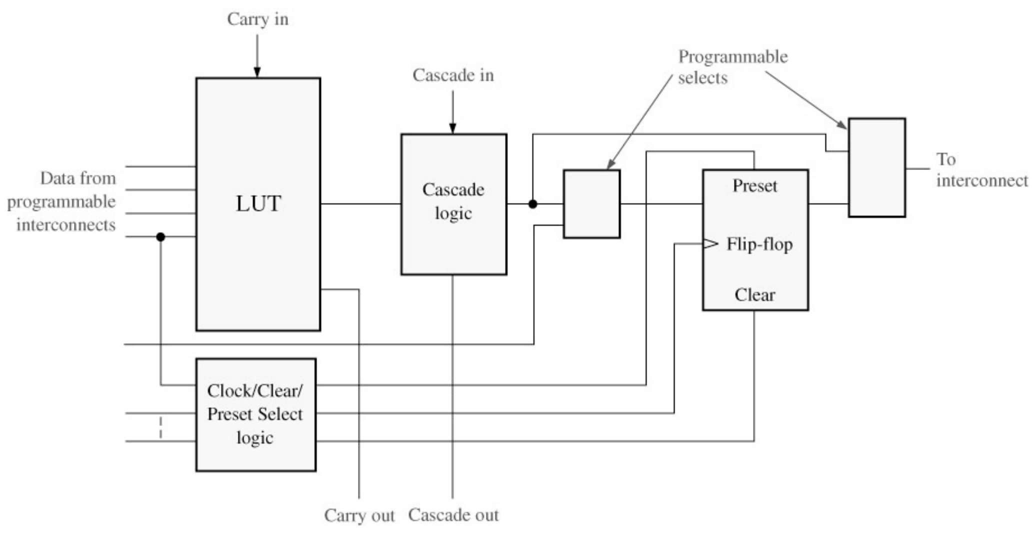


70

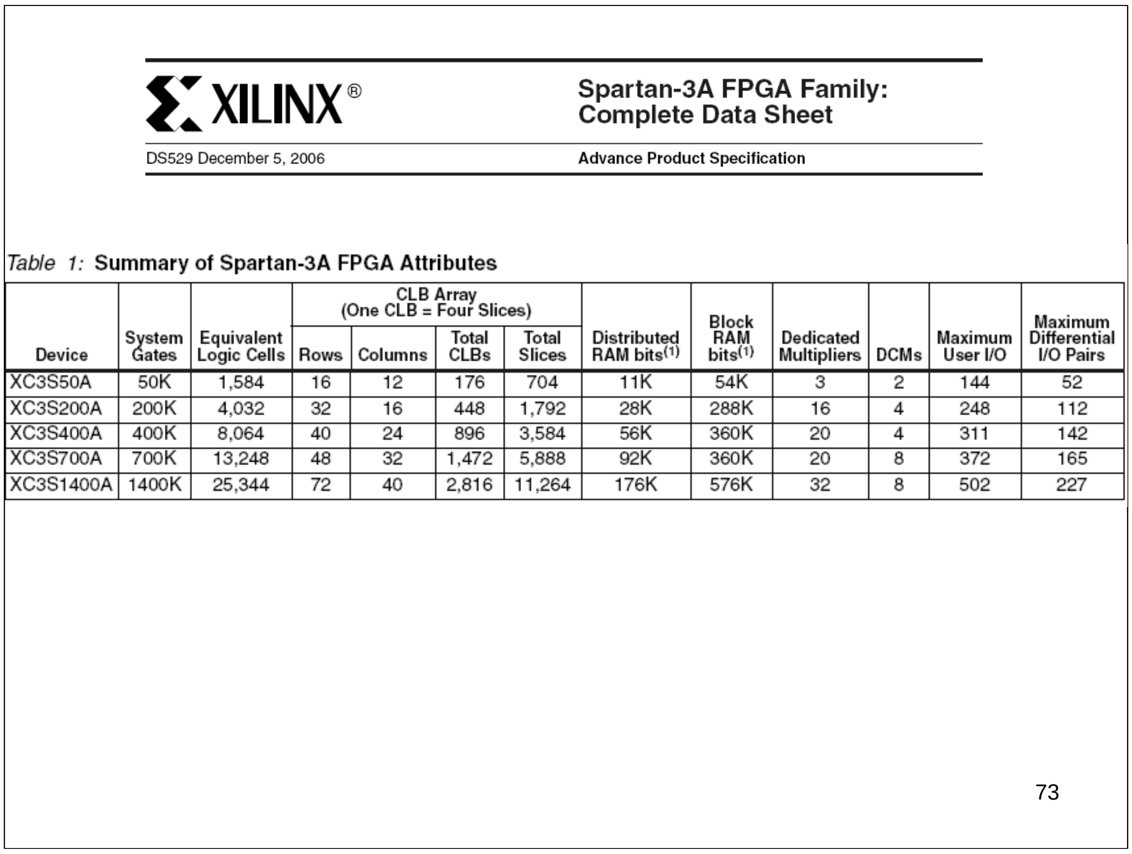
Typical Field Programmable Array (FPGA) structure …



… inside the Logic Blocks …



… a typical Logic Element.



XILINX®

Spartan-3A FPGA Family: Complete Data Sheet

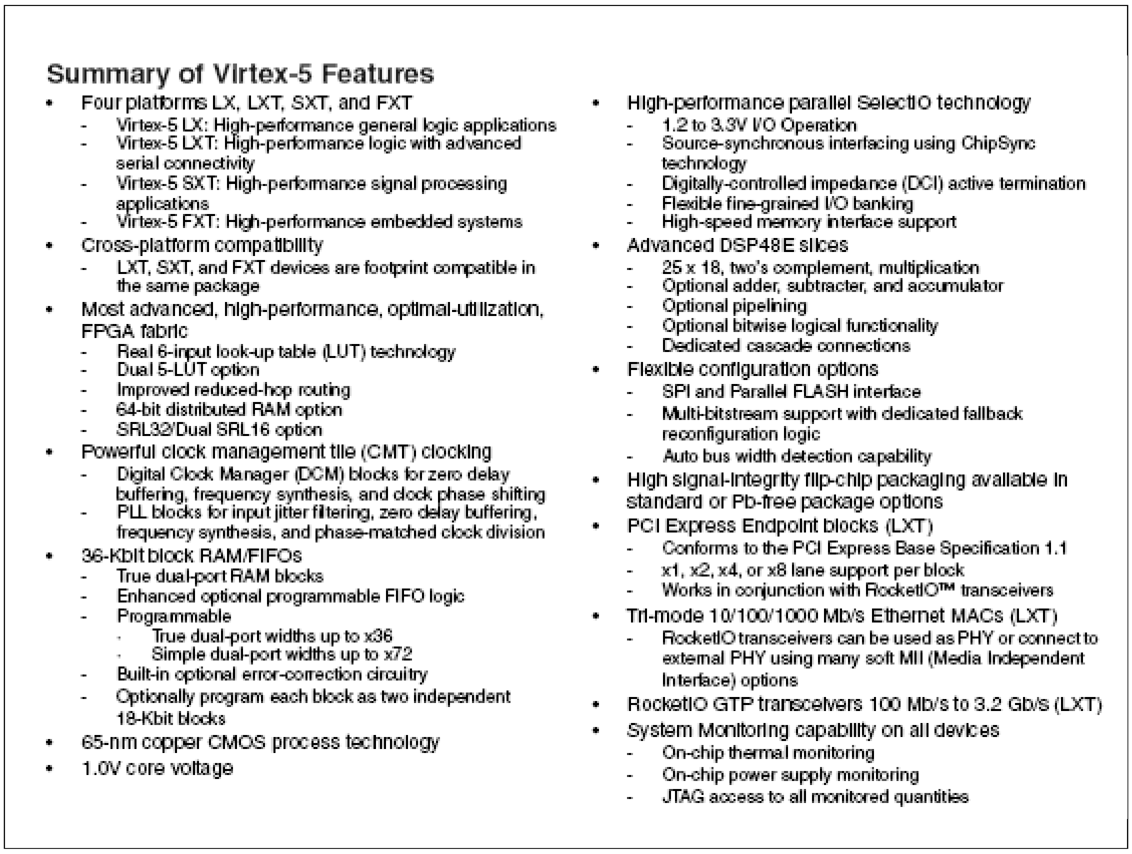
DS529 December 5, 2006

Advance Product Specification

Table 1: **Summary of Spartan-3A FPGA Attributes**

Device	System Gates	Equivalent Logic Cells	CLB Array (One CLB = Four Slices)				Distributed RAM bits(1)	Block RAM bits(1)	Dedicated Multipliers	DCMs	Maximum User I/O	Maximum Differential I/O Pairs
			Rows	Columns	Total CLBs	Total Slices						
XC3S50A	50K	1,584	16	12	176	704	11K	54K	3	2	144	52
XC3S200A	200K	4,032	32	16	448	1,792	28K	288K	16	4	248	112
XC3S400A	400K	8,064	40	24	896	3,584	56K	360K	20	4	311	142
XC3S700A	700K	13,248	48	32	1,472	5,888	92K	360K	20	8	372	165
XC3S1400A	1400K	25,344	72	40	2,816	11,264	176K	576K	32	8	502	227

… and again, you can buy some pretty massive devices.



Summary of Virtex-5 Features

- Four platforms LX, LXT, SXT, and FXT
  - Virtex-5 LX: High-performance general logic applications
  - Virtex-5 LXT: High-performance logic with advanced serial connectivity
  - Virtex-5 SXT: High-performance signal processing applications
  - Virtex-5 FXT: High-performance embedded systems
- Cross-platform compatibility
  - LXT, SXT, and FXT devices are footprint compatible in the same package
- Most advanced, high-performance, optimal-utilization, FPGA fabric
  - Real 6-input look-up table (LUT) technology
  - Dual 5-LUT option
  - Improved reduced-hop routing
  - 64-bit distributed RAM option
  - SRL32/Dual SRL16 option
- Powerful clock management tile (CMT) clocking
  - Digital Clock Manager (DCM) blocks for zero delay buffering, frequency synthesis, and clock phase shifting
  - PLL blocks for input jitter filtering, zero delay buffering, frequency synthesis, and phase-matched clock division
- 36-Kbit block RAM/FIFOs
  - True dual-port RAM blocks
  - Enhanced optional programmable FIFO logic
  - Programmable
    - True dual-port widths up to x36
    - Simple dual-port widths up to x72
  - Built-in optional error-correction circuitry
  - Optionally program each block as two independent 18-Kbit blocks
- 65-nm copper CMOS process technology
- 1.0V core voltage
- High-performance parallel SelectIO technology
  - 1.2 to 3.3V I/O Operation
  - Source-synchronous interfacing using ChipSync technology
  - Digitally-controlled impedance (DCI) active termination
  - Flexible fine-grained I/O banking
  - High-speed memory interface support
- Advanced DSP48E slices
  - 25 x 18, two's complement, multiplication
  - Optional adder, subtracter, and accumulator
  - Optional pipelining
  - Optional bitwise logical functionality
  - Dedicated cascade connections
- Flexible configuration options
  - SPI and Parallel FLASH interface
  - Multi-bitstream support with dedicated fallback reconfiguration logic
  - Auto bus width detection capability
- High signal-integrity flip-chip packaging available in standard or Pb-free package options
- PCI Express Endpoint blocks (LXT)
  - Conforms to the PCI Express Base Specification 1.1
  - x1, x2, x4, or x8 lane support per block
  - Works in conjunction with RocketIO™ transceivers
- Tri-mode 10/100/1000 Mb/s Ethernet MACs (LXT)
  - RocketIO transceivers can be used as PHY or connect to external PHY using many soft MII (Media Independent Interface) options
- RocketIO GTP transceivers 100 Mb/s to 3.2 Gb/s (LXT)
- System Monitoring capability on all devices
  - On-chip thermal monitoring
  - On-chip power supply monitoring
  - JTAG access to all monitored quantities

These days, the distinction between CPLD & FPGA has become rather blurred and the manufacturers are competing with each other for size and additional features. This is what the top end device from Xilinx can do. One page isn't enough to describe all its features and they've given up on drawing top-level diagrams!