

Section 1.2

Asynchronous Circuits

Synthesis

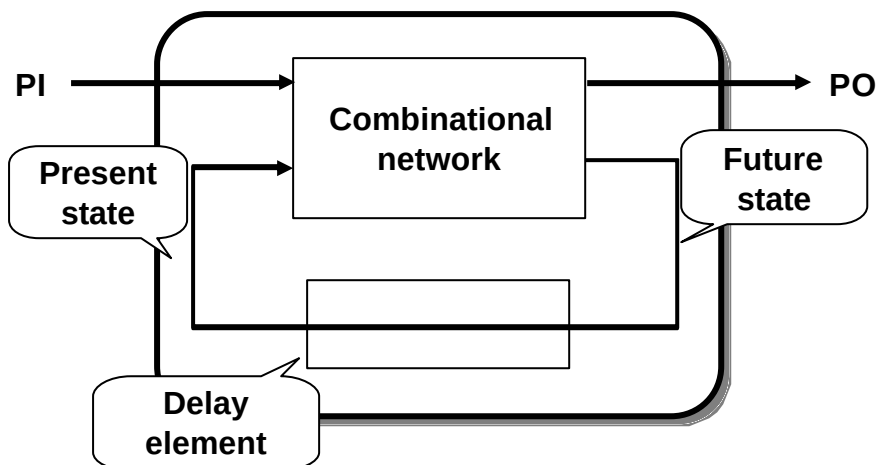
Alfredo Benso
Politecnico di Torino, Italy
Alfredo.benso@polito.it

Modeling an asynchronous circuit

How do I model an Asynchronous Circuit?

- An asynchronous circuit is defined by:
 - State variables
 - Equations that define the output function and the future state function

Asynchronous Circuits Model



How do I model an Asynchronous Circuit?

- **Future state function: modeled using a transition table**
- **Output function: modeled using an output table**

Transition Table

- **We call Transition Table a table that has:**
 - **One row for each possible combination of state variables (a row for each state)**
 - **One column for each combination of input variables**
- **Each cell c_{ij} of the table stores the future state of the circuit**

Example

	PI	00	01	11	10	
A		A	C	C	B	Future state
B		B	B	C	D	
C		D	C	C	A	
D		D	D	B	D	

Present state

Transition table

- **State:** a particular configuration of the state variables;
- **Stable State:** a state in which, for a certain input configuration, the future state is equal to the present state.

	PI	00	01	11	10
y_1y_2					
A		●	C	C	B
B		●	●	C	D
C		D	●	●	A
D		●	●	B	B

Y_1Y_2

Output Table

- The output table has:
 - One row for each state
 - One column for each input variables configuration
- Each cell c_{ij} of the output table stores the circuit outputs corresponding to state i and input configuration j .

		PI			
		00	01	11	10
y ₁ y ₂	A	0	1	1	0
	B	1	1	0	1
	C	0	1	0	0
	D	0	1	1	1
		U			

How does it work?

- Suppose that you are in a given state of the transition table and a primary input changes, you have to move on the same row to the column corresponding to the new input configuration
- Two possible cases:
 1. We end up in a stable state. In this case the state does not change until the next input change

How does it work?

- The state is unstable: in this case I have to move vertically to the row corresponding to the new state. I keep moving until I reach a stable state. If there is no stable state, then the circuit will not be stable until the next input change

y_1y_2 \ PI 00 01 11 10 A (A) C C B B (B) (B) C D C D (C) (C) A D (D) (D) B B Input: 00 PS: A Y_1Y_2	y_1y_2 \ PI 00 01 11 10 A (A) C C B B (B) (B) C D C D (C) (C) A D (D) (D) B B Input: 00 $\rightarrow$ 01 FS: C Is stable Y_1Y_2
y_1y_2 \ PI 00 01 11 10 A (A) C C B B (B) (B) C D C D (C) (C) A D (D) (D) B B Input: 00 $\rightarrow$ 10 FS: B D B ... unstable Y_1Y_2	y_1y_2 \ PI 00 01 11 10 A (A) C C B B (B) (B) C D C D (C) (C) A D (D) (D) B (B) Input: 10 $\rightarrow$ 11 PS: D Y_1Y_2

Modeling using only one table

- It is possible to use only one table, where in each cell it is possible to specify the future state and the outputs

State \ PI				
	00	01	11	10
A	A,0	B,0	-, -	C,0
B	B,1	B,1	E, -	D,1
C	A, -	C,1	F, -	D, -
...	...	...	...	...

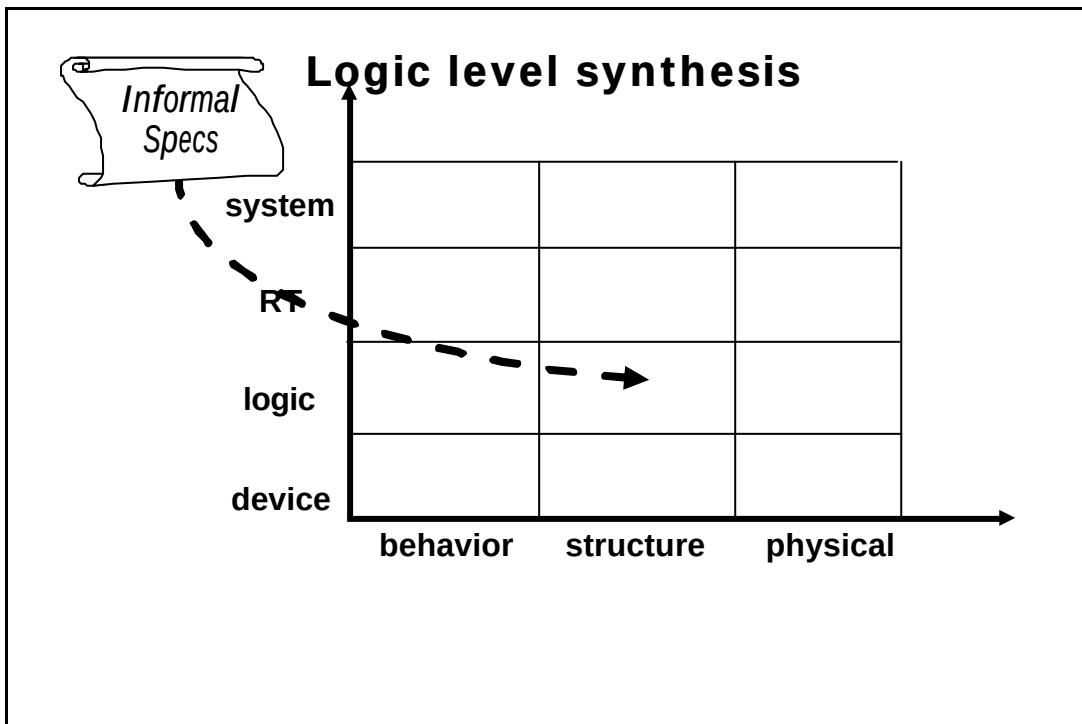
Problem: oscillation

- An asynchronous sequential circuit may become unstable and oscillate between unstable states because of the presence of feedback. The instability condition can be detected from the transition table.

Manual Synthesis

Manual synthesis

- **The manual synthesis of async circuits functioning in fundamental mode is done at the logic level**



Logic level synthesis

- The available logic blocks are:
 - Elementary or complex logic gates
- In the following we will have a first look at the synthesis steps, which will be detailed in the next section

Synthesis steps

1. RT-level formalization:

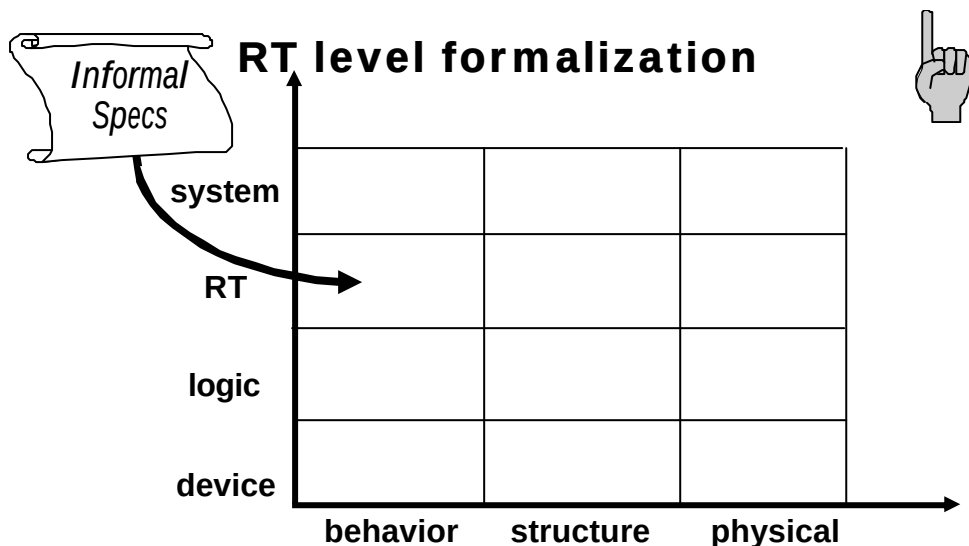
1. Obtain a primitive flow table from the given specification.
2. Reduce the flow table by merging rows in the primitive flow table.

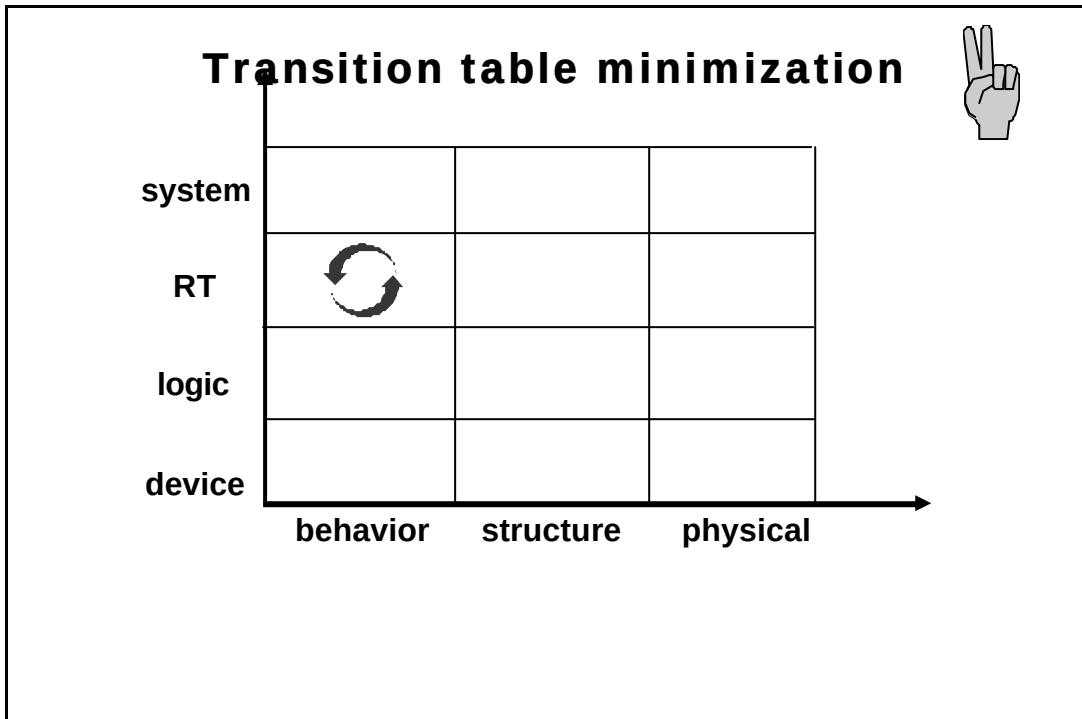
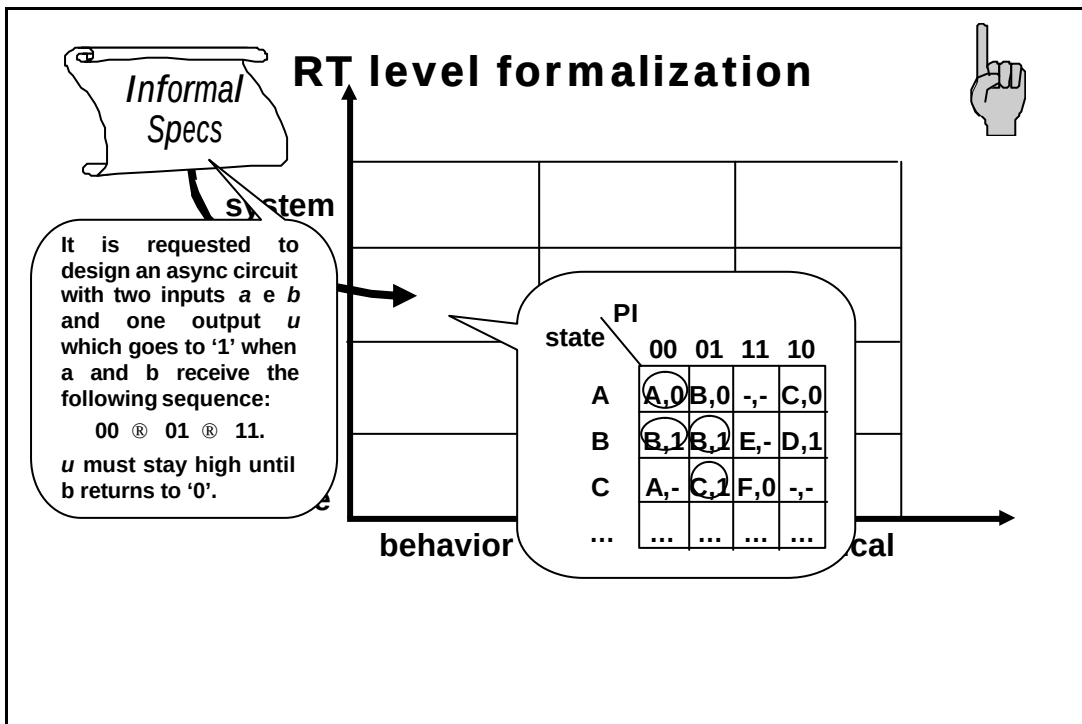
2. State /Output assignment:

1. Assign binary states variables to each row of the reduced flow table to obtain the transition table.
2. Assign output values to the dashes associated with the unstable states to obtain the output maps.

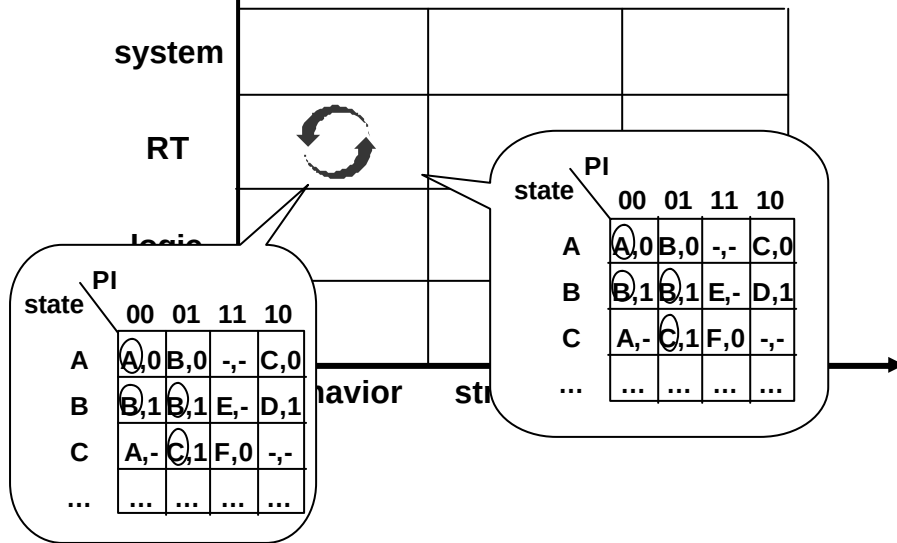
3. Library binding:

1. Simplify the Boolean functions of the state and output variables and draw the logic diagram.

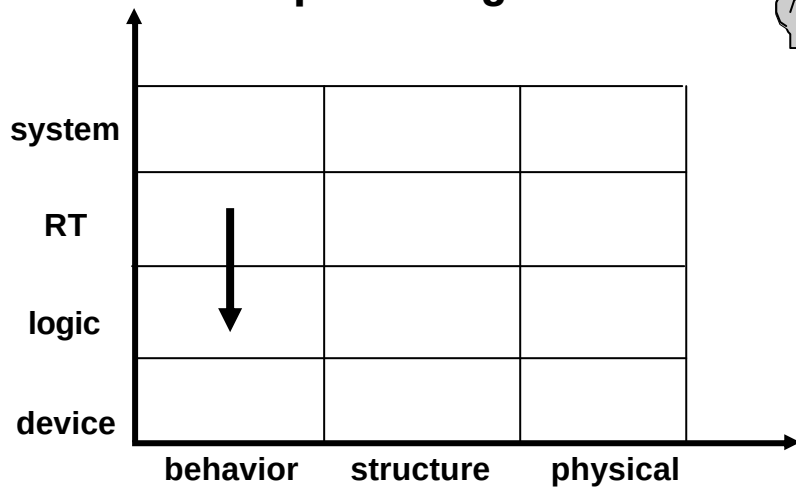




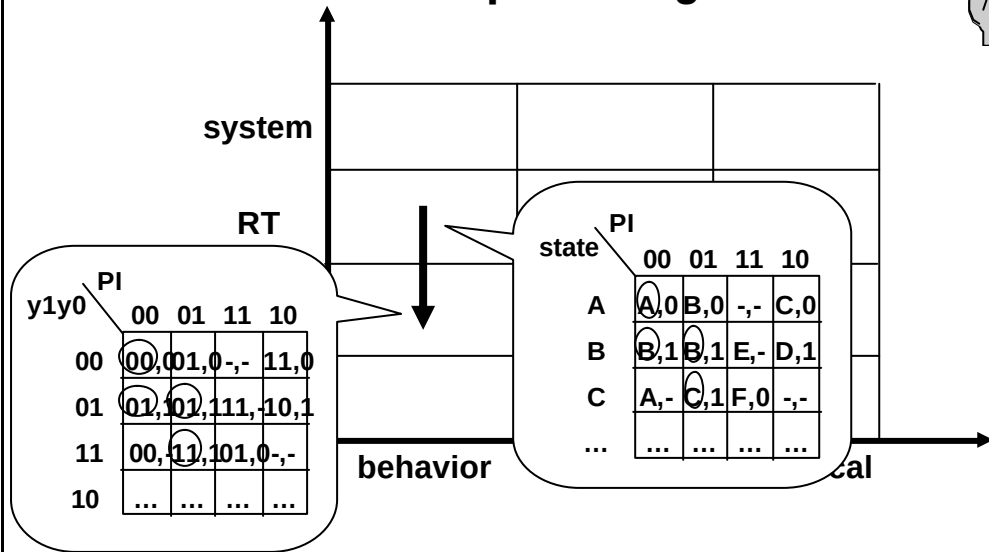
Transition table minimization



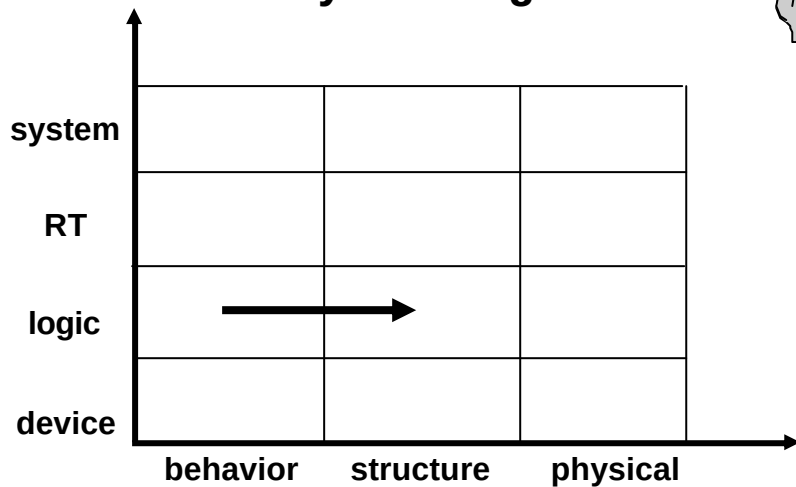
State / Output assignment



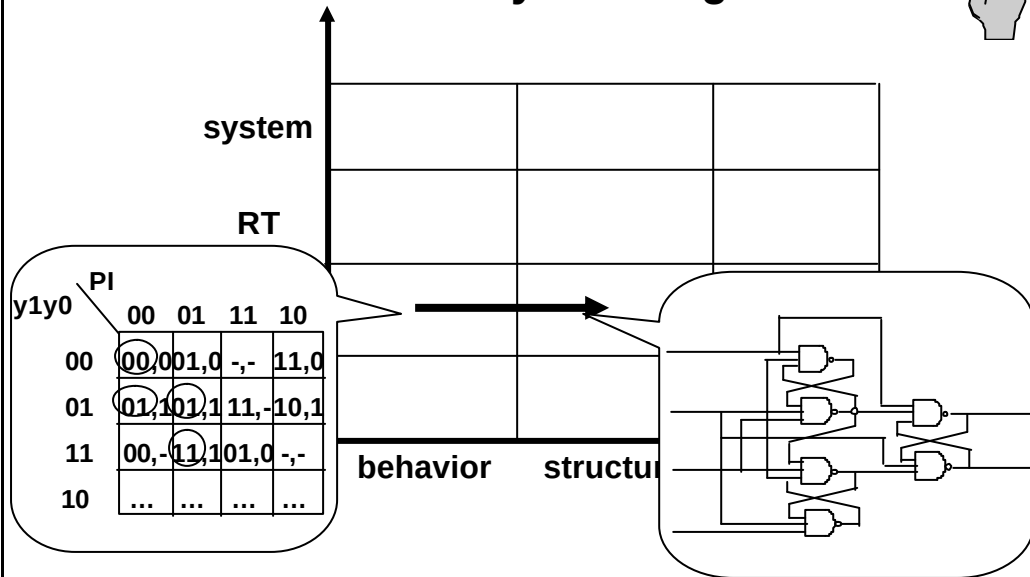
State / Output assignment



Library binding



Library binding

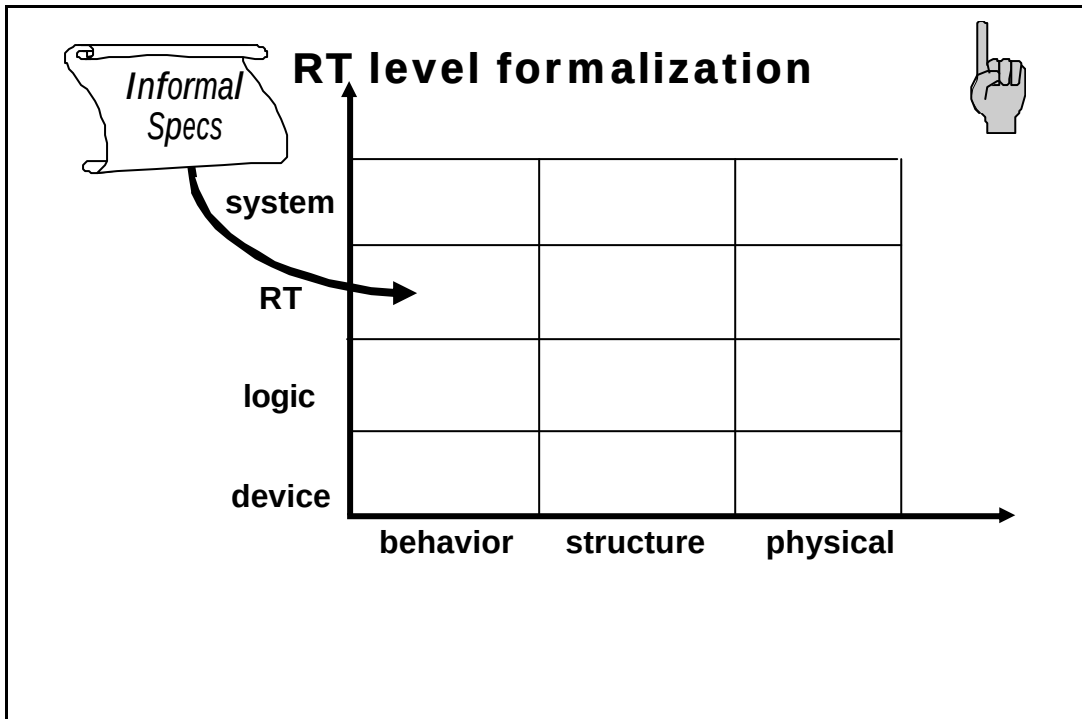


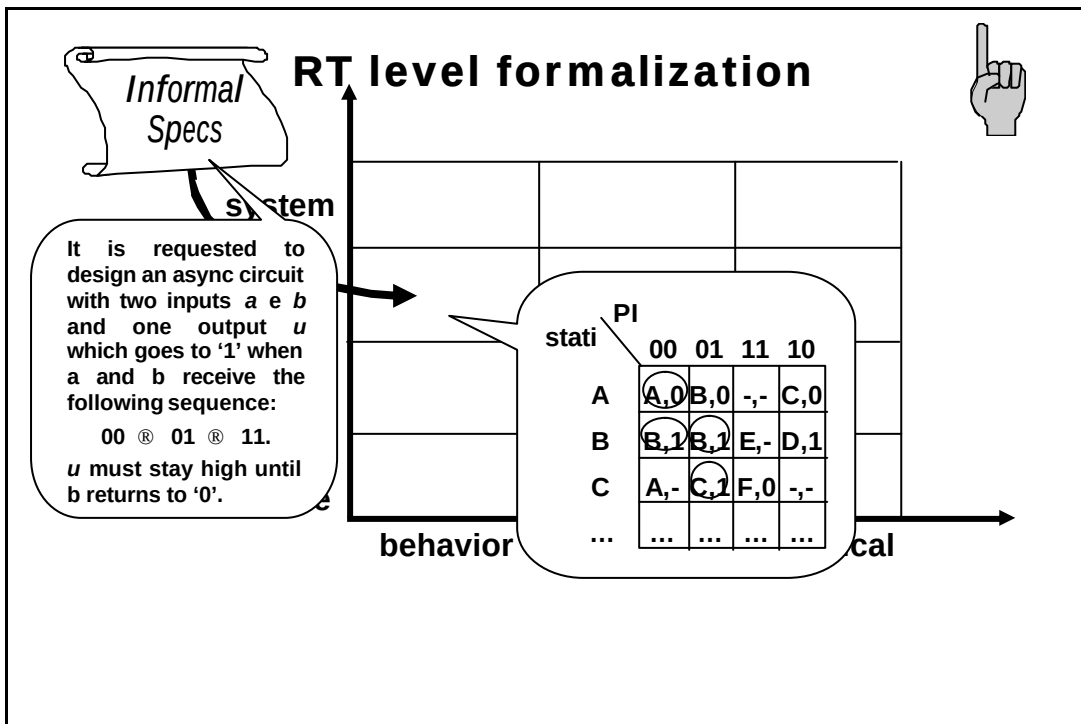
Note

- From this point on, the synthesis steps are the same as for combinational circuits, taking particular care in avoiding hazards.

Let's see the details

- Build the transition table
- Minimize the transition table
- State coding
- Library binding





Building the transition table

- **Suggestions:**
 - At first, design the transition table with only one stable in each row.
 - When designing sequence recognizers, it is easier to specify first the behavior of the circuit when it receives the correct sequence

Example

00-01-11

- **Formal specifications:**

- Design an async circuit with two inputs a and b and one output signal u . The output u is equal to '1' when on the two input signals receive the following sequence: 00 ® 01 ® 11.
- The output u changes back to '0' only when b returns to '0'.

Transition table

00-01-11

The circuit is a sequence recognizer. Let's start with the behavior when receiving the correct sequence

00

		ab			
state		00	01	11	10
	A	(A,0)			
...		...	...	...	...

Transition table

00-01-11

00 - 01

state \ ab				
	00	01	11	10
A	(A,0)	B		
B		(B,0)		
...	...	...	...	...

Transition table

00-01-11

00 - 01 - 11

state \ ab				
	00	01	11	10
A	(A,0)	B		
B		(B,0)	C	
C			(C,1)	
...	...	...	...	...

“Out of sequence state”

- Suggestion

- When designing a sequence recognizer, it is useful to have a single state to reach in case the input goes “out of sequence”. From this “out of sequence” state, it is possible to exit only when the inputs reach again the beginning of the correct sequence.

Transition table

00-01-11

- We can now complete the transition table of the sequence recognizer.

stati	ab			
	00	01	11	10
A	A,0	B	-	F
B	A	B,0	C	-
C	A	C,1	C,1	F
F	A	F,0	F,0	F,0

Outputs of unstable states

- The output value has to be specified for all states, including the unstable ones
- In the case of unstable states it is better to do as follows:
 - The two stable states involved in the transition have the same output: in order to avoid spikes, it is better to give the unstable state the same output as the stable ones.
 - The two stable states involved in the transition have different outputs: in this case it is better to use a don't care value, that will be assigned during the coverage of the output table

Example

00-01-11

state \ ab				
	00	01	11	10
A	A,0	B,0	-	F,0
B	A,0	B,0	C	-
C	A	C,1	C,1	F
F	A,0	F,0	F,0	F,0

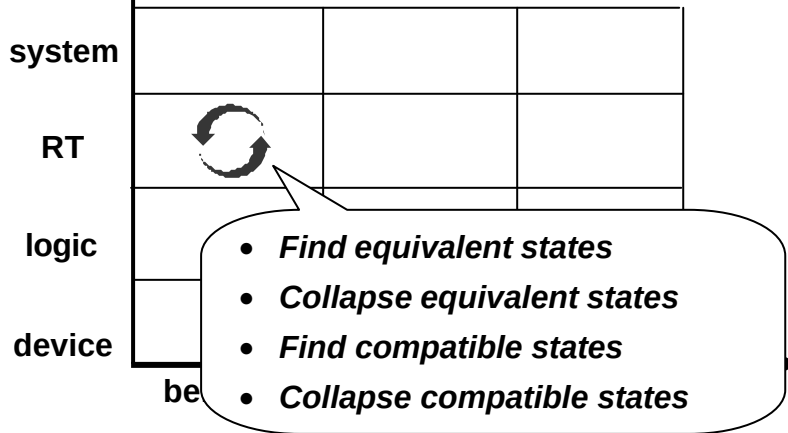
state	<i>ab</i>			
	00	01	11	10
A	A,0	B,0	-, -	F,0
B	A,0	B,0	C, -	-, -
C	A, -	C,1	C,1	F, -
F	A,0	F,0	F,0	F,0

Transition table minimization

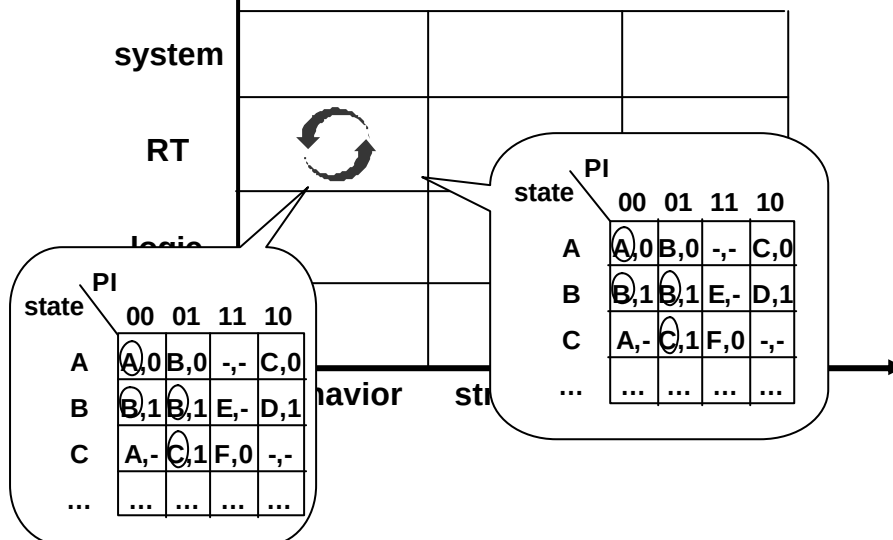


system			
RT			
logic			
device			
	behavior	structure	physical

Transition table minimization

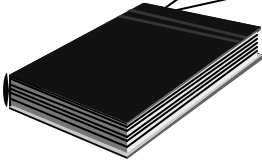


Transition table minimization



Equivalent States

- Two states are equivalent iff, with the same inputs, they have the same outputs and they transit to the same equivalent states.



Equivalent States

state \ ab		
	0	1
A	C,0	B,1
B	D,0	A,1
C	A,1	D,0
D	B,1	D,0

- A and B have the same outputs, and go to the same equivalent states (C, D)
- C and D have the same outputs, and go to the same equivalent states (A, B)

$(A, B) \stackrel{R}{\sim} X$

$(C, D) \stackrel{R}{\sim} Y$

state \ ab		
	0	1
X	Y,0	X,1
Y	X,0	Y,1

Collapsing equivalent states

- To collapse k equivalent states:
 - Choose one state (P) as the representative of the equivalence class
 - Delete the k-1 rows corresponding to the equivalent states
 - Substitute k-1 the references to the k-1 states with references to P.

state	ab	
	0	1
~~A~~	~~C~~ ,0 ~~B~~ ,1	
B	D,0	A,1
~~C~~	~~A~~ ,1 ~~D~~ ,0	
~~D~~	~~B~~ ,1 ~~D~~ ,0	

state	ab	
	0	1
X	Y,0	X,1
Y	X,0	Y,1

Compatible states

- Two states P e Q are compatible and collapsable in a new state T if, for each column, one of the following conditions is verified:

P	-	-	-	R,u	R,u	P,u	P, u	P,u
Q	-	P	R	R,u	R,-	P	-	Q,u
T	-	T	R	R,u	R,u	T,u	T,u	T,u

Note

- **To understand if two states are compatible, it is necessary to look at the outputs also in the unstable states. For this reason it is better to wait to assign the output to unstable states.**

Collapsing compatible states

- **Build a compatibility graph where:**
 - **Nodes represent states**
 - **Edges represent compatibilities**
- **In general, a state may be compatible with more than one state. One collapsing might be globally more efficient than others. So how do I find the best collapsing scheme? It is a complex optimization problem.**

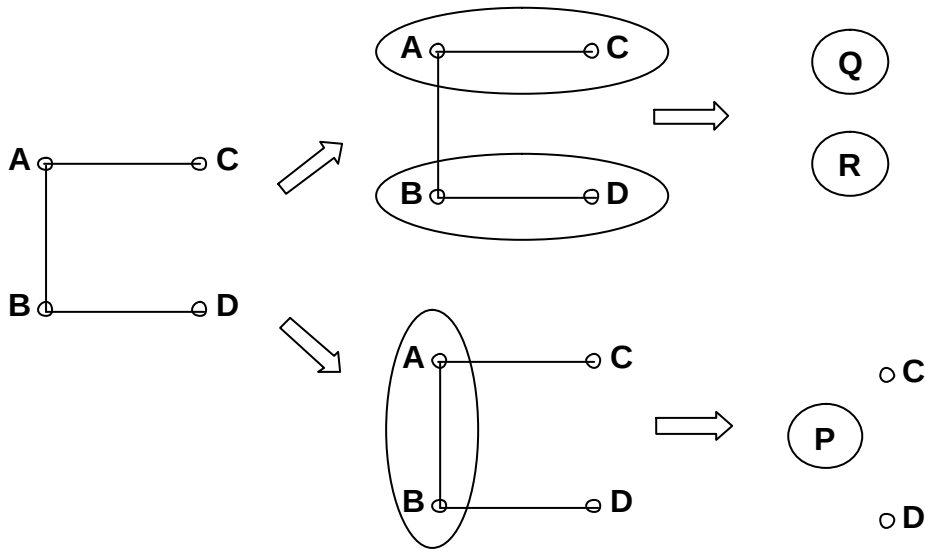
Greedy algorithm

- **while (there are cliques)**
- **{**
 - **Identify a clique;**
 - **Delete all the edges that connect the nodes of the clique with external nodes;**
 - **Collapse the clique into one state;**
- **}**

Greedy algorithm

- **The number of final states depends on the order in which I analyzed the graph.**

Greedy algorithm



Example

— 00-01-11 —

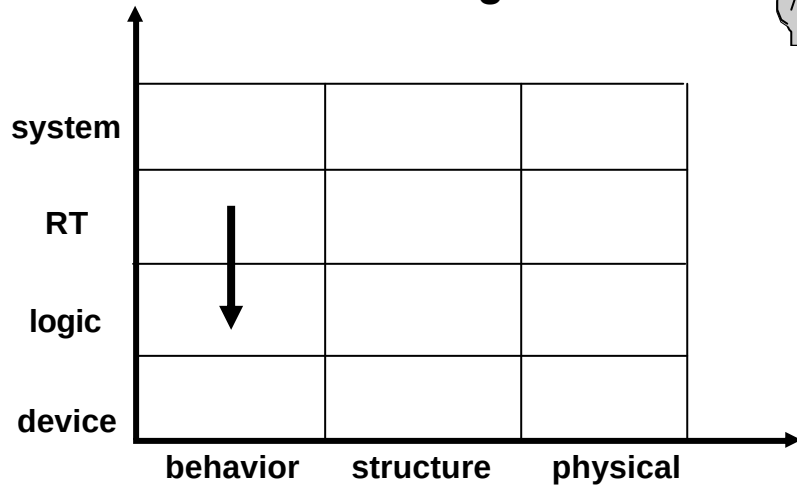
- In the transition table of the sequence recognizer there are no equivalent states, but there are two compatible states (A and B)

state	ab			
	00	01	11	10
A	A,0	B,0	-, -	F,0
B	A,0	B,0	C, -	-, -
C	A, -	C,1	C,1	F, -
F	A,0	F,0	F,0	F,0

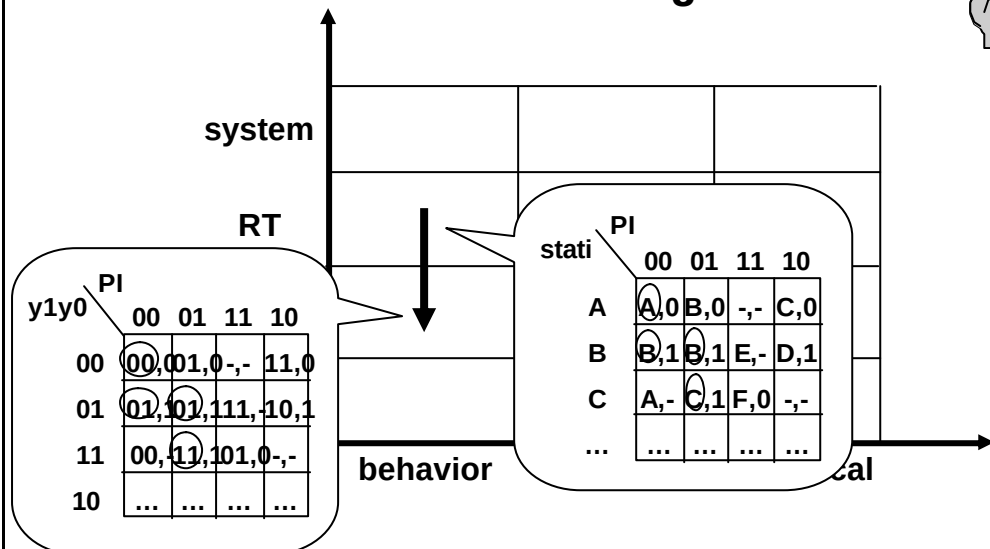
→

state	ab			
	00	01	11	10
D	D,0	D,0	C, -	F,0
C	D, -	C,1	C,1	F, -
F	D,0	F,0	F,0	F,0

State coding



State coding



State coding

- Two steps:
 - Compute the minimum number of required state variables = $\lceil \log_2 k \rceil$
 - Code the states, avoiding critical races.

Problem: Race

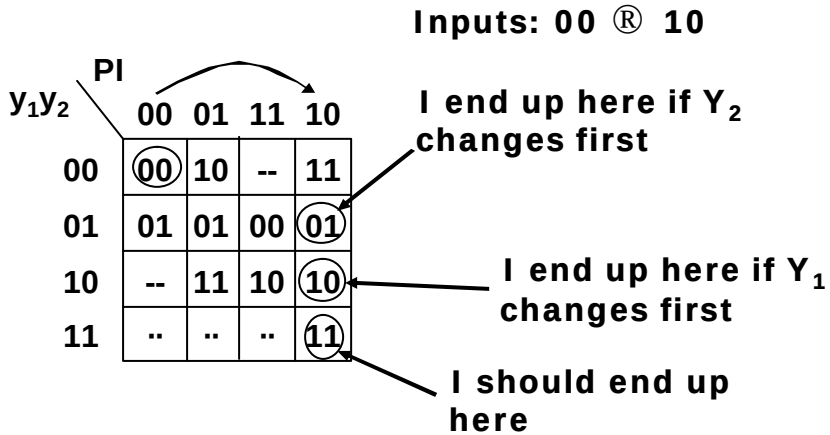
- A race condition exists in an asynchronous circuit when two or more state variables change value in response to a change in an input variable.
- When unequal delays are encountered, a race condition may cause the state variable to change in an unpredictable manner.

The diagram shows a Karnaugh map for a 2-bit state variable y_1y_2 and a 1-bit input variable PI . The map is a 2x4 grid. The columns are labeled with PI values: 00, 01, 11, 10. The rows are labeled with y_1y_2 values: 00, 01. The cells contain the next state values. A race condition is indicated by a double-headed arrow between the 00 and 11 states in the $PI=0$ column.

$y_1y_2 \backslash PI$	00	01	11	10
00	00	10	--	11
01	01	01	00	11

Problem: Critical race

- A situation in which the final state depends on the order in which the state variables change



Solution

- Choose a state coding that avoids critical races.

How to avoid races

- To avoid a critical race it is necessary to make sure that:
 1. For each transition between two states, all the traversed states must be adjacent (the coding of two adjacent states differs for 1 bit)
 2. Hazard free map coverage
- The first problem can be solved identifying the state pairs which must be adjacent.

Example

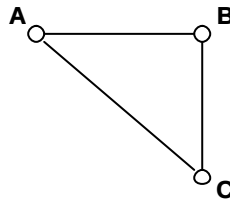
- A must be adjacent to B and D
- B must be adjacent to A and C
- ...

stat	ab			
	00	01	11	10
A	Ⓐ	B	-	D
B	A	Ⓑ	C	-
...	...	...	...	...

Vicinity Graph

- Since there might be many conditions to satisfy, it is better to resort to a graph where:
 - Nodes represent states
 - Edges connect adjacent states. (Each edge is weighted with the number of adjacencies that it represents).

		ab			
stati		00	01	11	10
	A	(A)	B	-	C
	B	A	(B)	C	-
	C	A	B	(C)	(C)

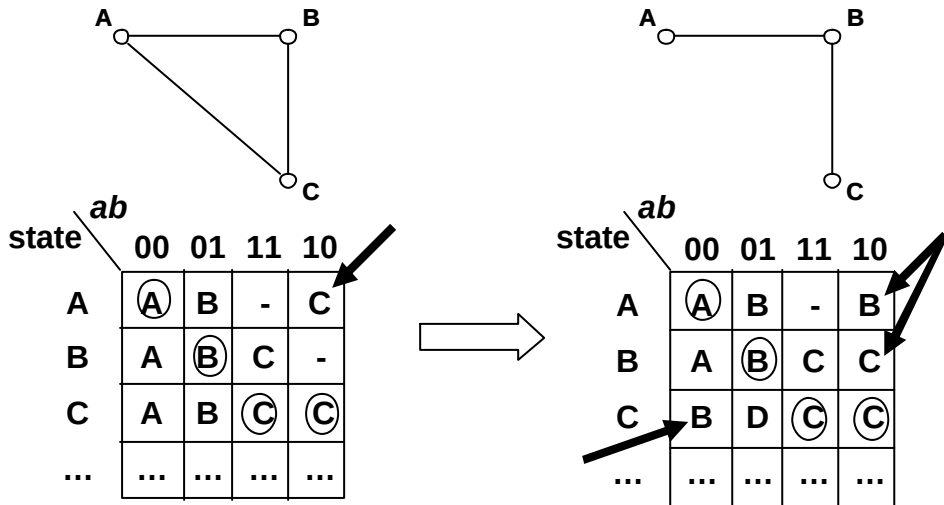


Required condition

- To satisfy all adjacencies, the number of state variables must be greater or equal to the order of the Graph.
- **#state variables** = $\max(\lceil \log_2 \#states \rceil, \max(\text{Outdegree}))$

Note

- Some adjacencies can be eliminated going through unstable states



Choosing which adjacencies to eliminate

- In choosing which adjacencies to eliminate, it is advisable to proceed as follows:
 - Identify the required adjacencies
 - Try to eliminate not-required adjacencies, starting from the nodes with a higher degree (higher number of edges)

Required adjacencies

- Required adjacencies are usually the ones imposed by the columns with higher number of stable states.

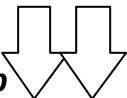


Diagram illustrating required adjacencies. The table shows states (A, B, C, ...) for different values of 'ab' (00, 01, 11, 10). The states are arranged in a grid, with some states circled to indicate required adjacencies. The label 'ab' is above the columns, and 'stati' is to the left of the rows.

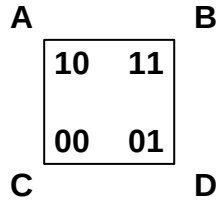
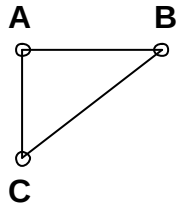
	00	01	11	10
A	(A)	B	-	C
B	(B)	(B)	C	A
C	A	(C)	(C)	(C)
...	...	...	...	...

Final State Coding

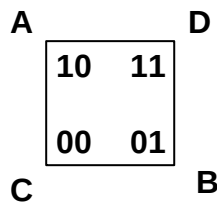
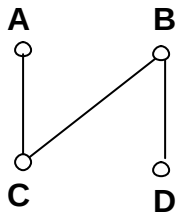
- A formal method to find a correct final state coding, consists in drawing the vicinity graph on a n-dimension cube.
- The procedure is as follows:
 - We build a cube of dimension n, where $n = \max(S_m, G)$, where S_m is the number of states and G is the degree of the vicinity graph.
 - We try to assign to each vertex of the cube a state so that all the adjacencies reported in the adjacencies graph are respected.

Example

- Certain adjacencies can be respected passing through sequences of adjacent states.
- The adjacency between B and C can be fulfilled following the path: B - D - C



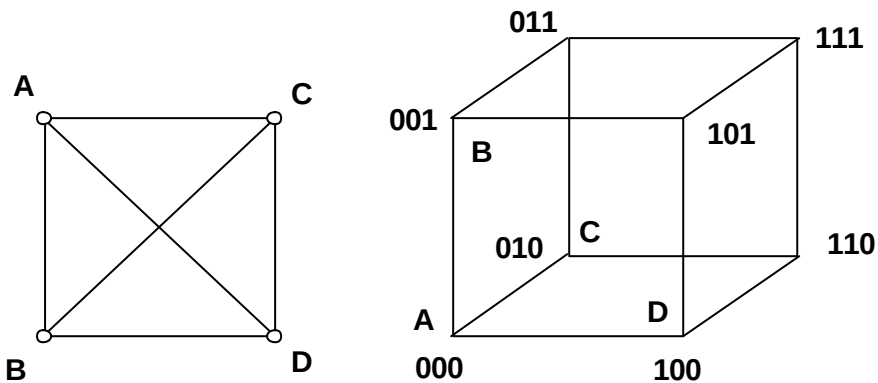
Example



Theorem

- It is always possible to design an async circuit having at most $2 \cdot S_m - 1$ state variables and no critical races.

Example

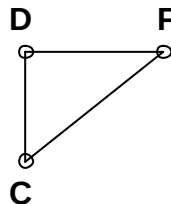


Example

00-01-11

- From the reduced transition table of the sequence recognizer, it is possible to derive the following vicinity graph:

stati \ ab	00	01	11	10
D	(D,0)	(D,0)	C,-	F,0
C	D,-	(C,1)	(C,1)	F,-
F	D,0	(F,0)	(F,0)	(F,0)



00-01-11

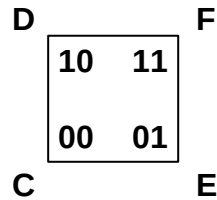
- The adjacency between C and F can be respected with an unstable transition trough E.
- State and output tables are therefore:

state \ ab	00	01	11	10
D	D	D	C	F
C	D	C	C	E
F	D	F	F	F
E	-	-	-	F

state \ ab	00	01	11	10
D	0	0	-	0
C	-	1	1	-
F	0	0	0	0
E	-	-	-	-

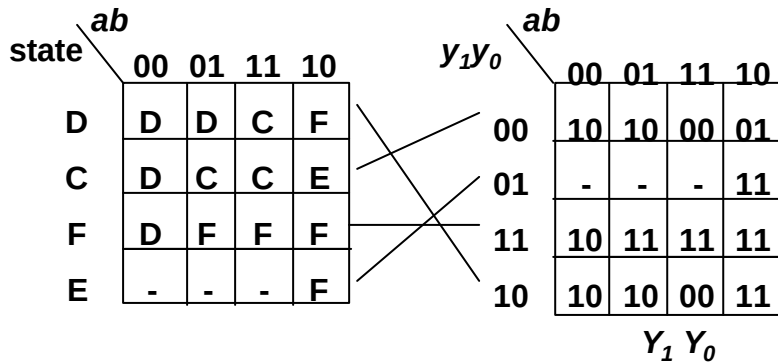
u

- Assigning the following coding scheme:

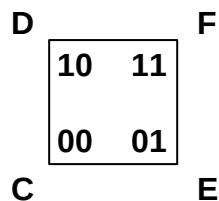


00-01-11

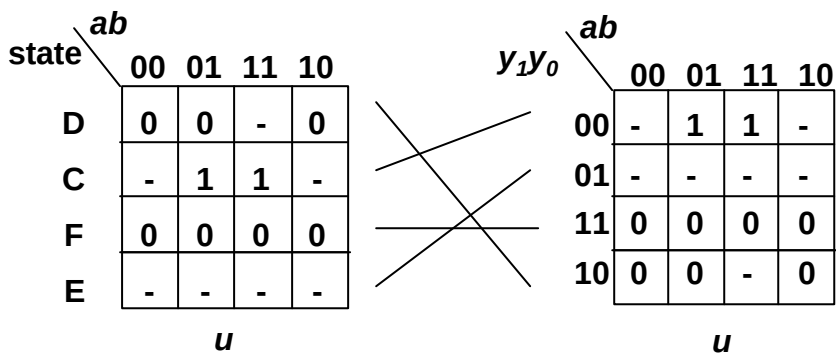
- The result is:



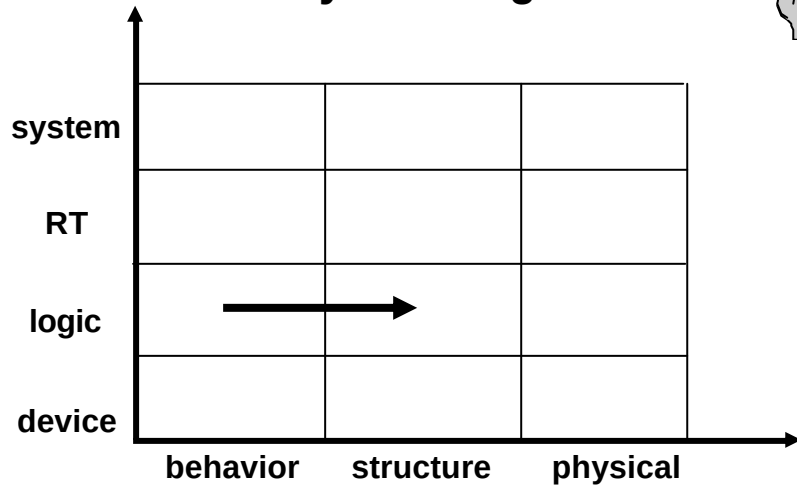
- For the output u:



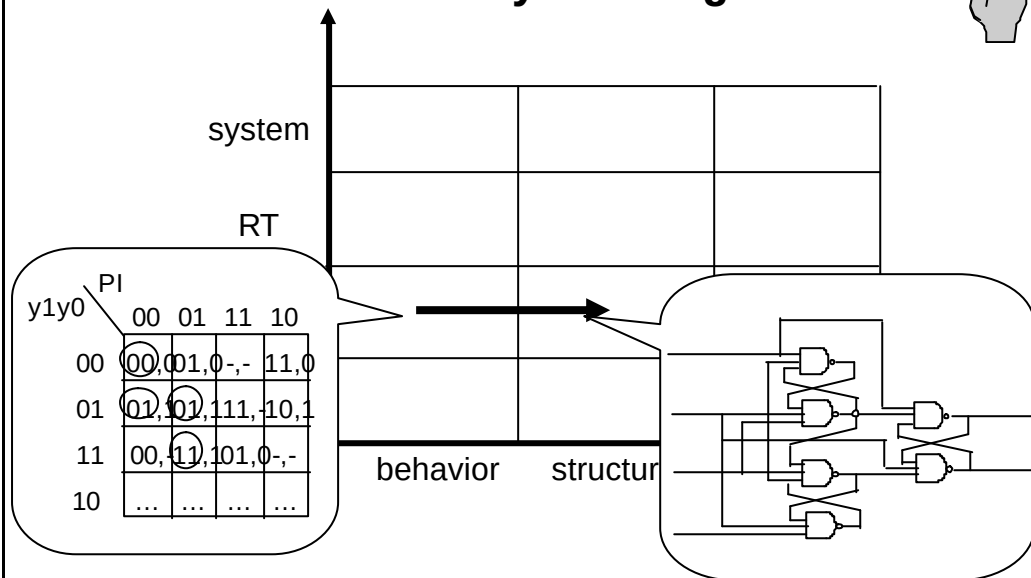
00-01-11



Library binding



Library binding



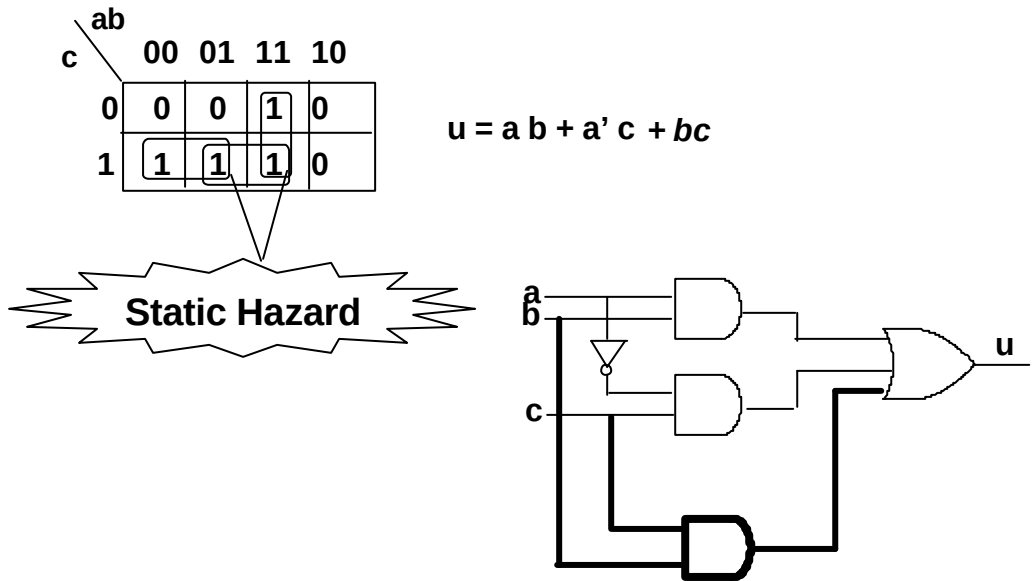
Library binding

- **The goal is to implement the combinational networks that implement the tables designed until now.**
- **When finding minterms, it is important to avoid hazards.**
- **Hazards are unwanted switching transients that may appear at the output of a circuit because different paths exhibit different propagation delays.**

How to avoid hazards?

- **We have a static hazard whenever two adjacent '1' in the K-map are covered by different interms.**
- **The solution to avoid hazards is therefore to use redundant minterms so that every pair of adjacent '1' is covered by the same minterm.**

Example



Note

- Sometimes hazards can be avoided carefully choosing the minterms (and without introducing redundant ones).