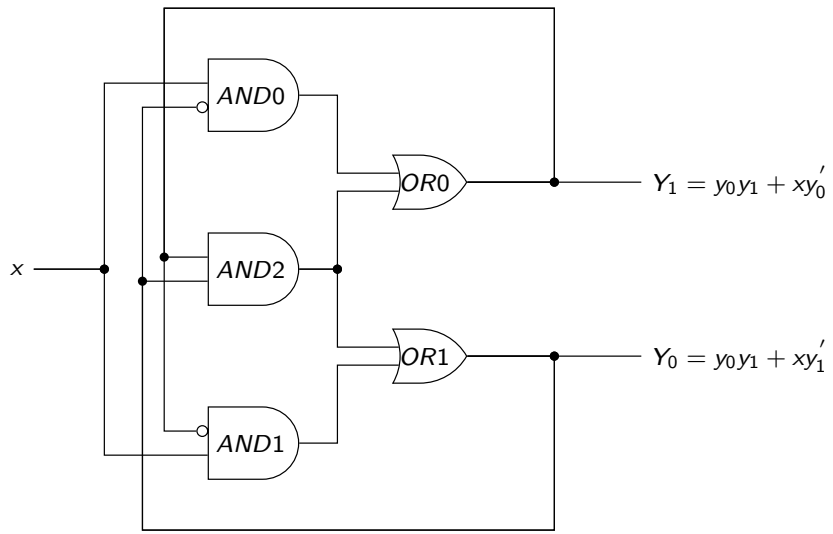
Races

- A **race condition** occurs in an asynchronous circuit when two or more state variables are required to change at the same time in response to a change in a circuit input.
- Unequal circuit delays (due to delay through logic gates and along wires) can imply that the state variables might not change at the same time and this can potentially cause problems.
- Let us assume that several state variables are required to change:
  - If the circuit reaches the correct final stable state regardless of the order in which the state variables change then the race is **not critical**.
  - If the circuit reaches can reach different final stable states depending on the order in which the state variables then the race is **critical**.
- Therefore, one issue that we should address when designing an asynchronous circuit is to *avoid designs which have critical races*. Fortunately, we can fix and avoid critical races.

Races — example of a critical race

- Example of a critical race. Assume the input x is 0 and we are currently in the stable state $y_2 y_1 = 00$. The circuit and transition table are shown:

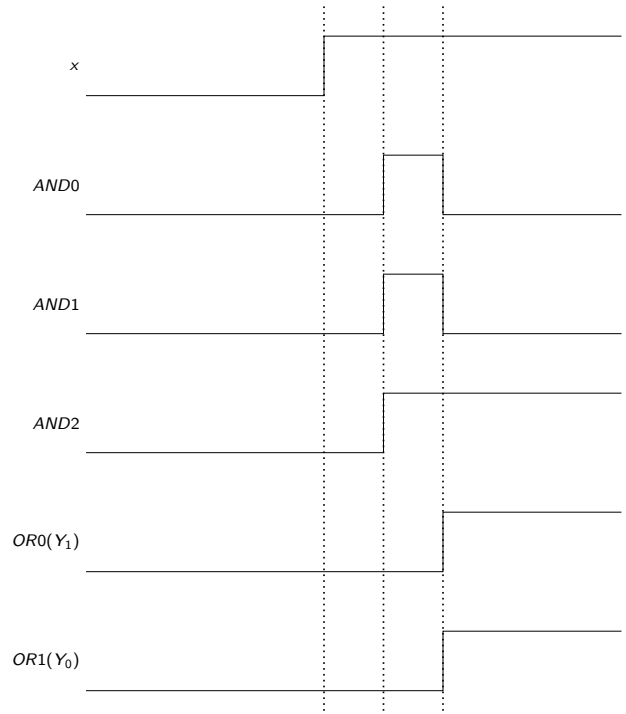


Current state $y_1 y_0$	**Next state** $(Y_1 Y_0)$ $x = 0$	$x = 1$
00	**00**	11
01	00	**01**
11	**11**	**11**
10	00	**10**

- Assume that the input changes from $x = 0 \rightarrow 1$.
- From the transition table, we see that we should transition to stable state $y_1 y_0 = 11$ when $x = 1$.

Races — example of a critical race

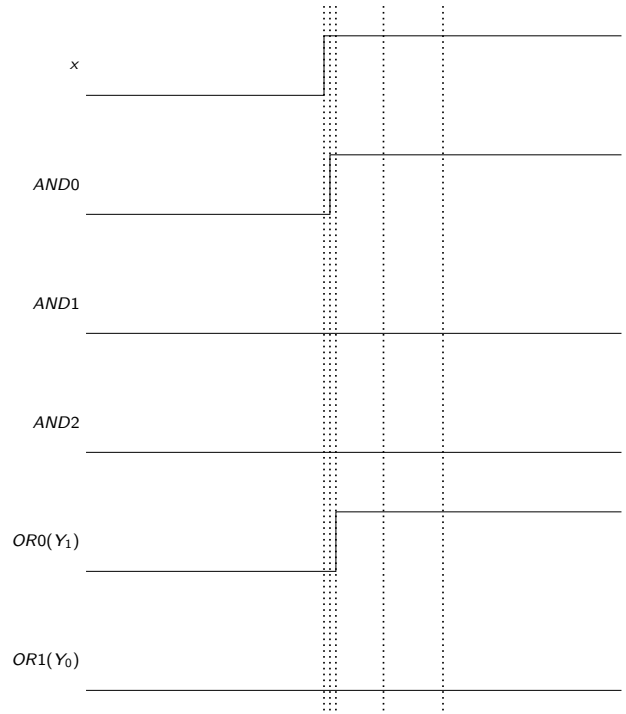
- Case I: Assume that all the delays through the gates in the circuit are the same.
- We would see the various signals change as follows:



- So if we looking at the outputs $Y_1 Y_0$ it would they would appear to have changed $\boxed{00} \rightarrow \boxed{11}$ which is expected.

Races — example of a critical race

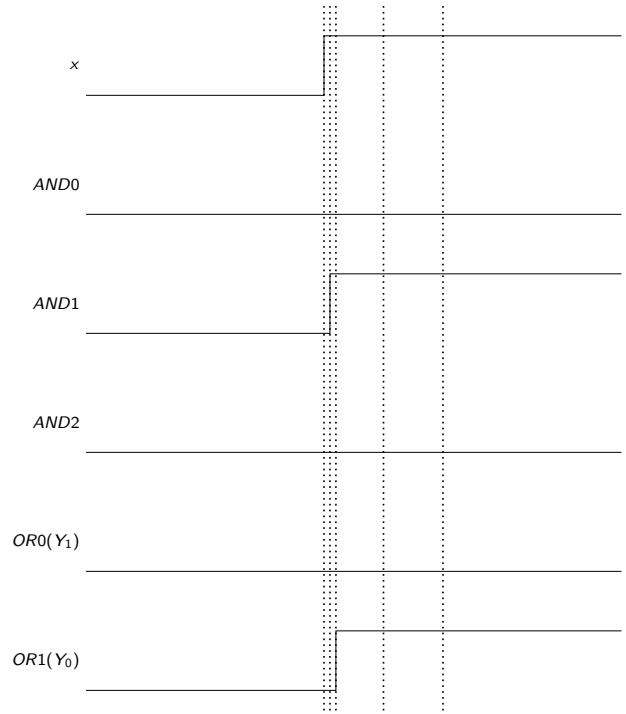
- Case II: Now assume that the delay through $AND0$ and $OR0$ are both very fast.
- We would see the various signals change as follows:



- Output Y_1 will change very quickly from $0 \rightarrow 1$. The value of 1 will propagate back around to the inputs of $AND1$ (inverted) and $AND2$. The value at the inverted input of $AND1$ will prevent $AND1$ from changing its output which means that Y_0 will not change due to x changing from $0 \rightarrow 1$.
- So if we looking at the outputs $Y_1 Y_0$ it would they would appear to have changed $\boxed{00} \rightarrow \boxed{10}$ **which is the wrong behaviour according to the transition table.** which is expected.

Races — example of a critical race

- Case III: Now assume that the delay through $AND1$ and $OR1$ are both very fast.
- We would see the various signals change as follows:



- Output Y_0 will change very quickly from $0 \rightarrow 1$. The value of 1 will propagate back around to the inputs of $AND0$ (inverted) and $AND2$. The value at the inverted input of $AND0$ will prevent $AND0$ from changing its output which means that Y_1 will not change due to x changing from $0 \rightarrow 1$.
- So if we looking at the outputs $Y_1 Y_0$ it would they would appear to have changed $\boxed{00} \rightarrow \boxed{01}$ **which is the wrong behaviour according to the transition table.** which is expected.

Races — example of a non critical race

- Example of a non-critical race. Consider the following transition table (we can ignore the actual circuit):

Current state	**Next state** $(Y_1 Y_0)$	
$y_1 y_0$	$x = 0$	$x = 1$
00	$\boxed{00}$	11
01	11	$\boxed{01}$
11	$\boxed{10}$	01
10	10	11

- Assume that $x = 0$ and we are in the stable state $y_1 y_0 = 00$. Now assume that x changes $0 \rightarrow 1$.
- We could see the following things are the output: $\boxed{00} \rightarrow \boxed{01}$, $\boxed{00} \rightarrow 11 \rightarrow \boxed{01}$ or $\boxed{00} \rightarrow 10 \rightarrow 11 \rightarrow \boxed{01}$.
- Regardless of how the state variables change, we always end up in stable state 01 which is the expected behaviour.
- This race is therefore non-critical because the circuit will behave correctly and is insensitive to the delays in the circuit.

Races

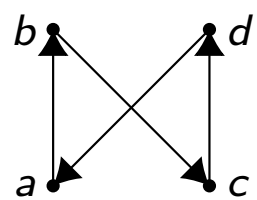
- Races are a direct result of multiple state variables having to change due to a change in an input variable.
- When examining races we used transitions tables and not flow tables.
- Races **do not exist** in flow tables because there is no binary state assignment in a flow table (and consequently multiple state variables do not change in a flow table due to a change in an input variable).
- Consequently, races **are a result of state assignment**. Therefore, we can **avoid** races by being more careful during state assignment.
  - Note that avoiding races is a **preemptive** strategy — races which are non-critical are okay. However, by proper state assignment we can avoid all races and be secure that we will not have any issues due to race conditions.
- If we are already given a transition table we can consider "fiddling" with the transition table slightly in order to try and remove the critical races (and we can ignore the non-critical races).

Transition diagrams

- The **transition diagram** helps to visualize when a race can potentially occur.
- The purpose of the transition diagram is to illustrate transitions between stable states.
- For small problems, the transition diagram is best drawn as a multi-dimensional cube. States are labeled on the corners of the cube while transitions between stable states are drawn as arrows between stable states.
- Example... Consider the following flow table:

Current state	**Next state** $x = 0$	$x = 1$
a	$\boxed{a}$	b
b	c	$\boxed{b}$
c	$\boxed{c}$	d
d	a	$\boxed{d}$

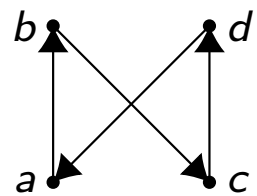
- One transition diagram:



- The 4 states are labelled and arrows are drawn between stable state trajectories. For example, if $x = 0$ and we are in stable state a and x changes to 1, we transition to stable stat b. Therefore, we include an arrow from $a \rightarrow b$.

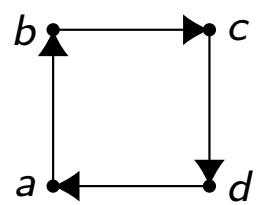
Transition diagrams

- If we consider the corners of the transition diagram to have coordinates, we then have a state assignment.
- The arrows in the transition diagram will then immediately tell us if there will be a race condition — diagonal edges in the transition diagram require *geq*2 state variables to change at the same time which indicates a race.
- The transition diagram:



clearly indicates that the state assignment $a = 00$, $b = 01$, $c = 10$ and $d = 11$ is will have races.

- We can try to rearrange the states and redraw the transition diagram to avoid diagonals:



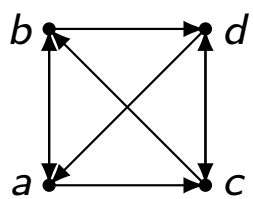
which has no diagonal arrows. Therefore, the state assignment $a = 00$, $b = 01$, $c = 11$ and $d = 10$ will not have races.

Avoiding races during state assignment — use of transition diagrams

- Races can be avoided by using a transition diagram — the objective is to assign states to the corners of a cube and then adjust the flow table to move from state to state along the edges of the cube rather than along diagonals in the cube.
- This approach is might require the addition of temporary unstable states.
- Consider the following flow table:

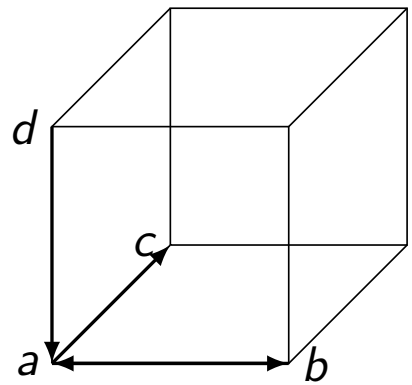
Current state	Next state $x_1x_0 = 00$	$x_1x_0 = 01$	$x_1x_0 = 11$	$x_1x_0 = 10$
a	$\boxed{a}$	$\boxed{a}$	c	b
b	a	$\boxed{b}$	d	$\boxed{b}$
c	$\boxed{c}$	b	$\boxed{c}$	d
d	c	a	$\boxed{d}$	$\boxed{d}$

- If you construct the transition diagram for this flow table, you will clearly see that it is impossible to even consider rearranging the states on the corners of the cube to remove races (diagonals).

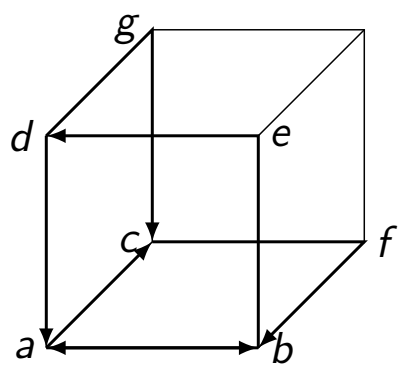


Avoiding races during state assignment — use of transition diagrams

- Draw a 3 dimensional cube and attempt to label states. Add extra states if required.
- Transition diagram showing possible transitions without diagonals and without extra states:



- Transition diagram showing all transitions without diagonals but without (required) extra states:



Extra state e to get from $b \rightarrow d$, f to get from $c \rightarrow b$, g to get from $c \rightarrow d$ and $d \rightarrow c$.

Avoiding races during state assignment — use of transition diagrams

- Final flow table (state assignment shown — taken from coordinates of the cube):

	Current state	**Next state** $x_1x_0 = 00$	$x_1x_0 = 01$	$x_1x_0 = 11$	$x_1x_0 = 10$
000	*a*	[*a*]	[*a*]	c	b
100	*b*	*a*	[*b*]	e	[*b*]
010	*c*	[*c*]	*f*	[*c*]	g
001	*d*	*g*	*a*	[*d*]	[*d*]
101	*e*	—	—	d	-
110	*f*	—	*b*	-	-
011	*g*	*c*	—	-	d

Avoiding races during state assignment — one hot encoding

- The idea of one-hot encoding can be used to get a race free state assignment.
- Assume a change in an input variable requires moving from stable state i to stable state b.
- Assume that every state k is encoded as $0\ldots0\ \underbrace{1}_{k\text{-th bit}}\ 0\ldots0$. Only a single bit is a 1.
- Then, to move from stable state i to stable state j requires *exactly* 2 bits to change (bit i must change from 1 to 0 and bit j must change from 0 to 1.
- This causes a race, but we can *always avoid the race* by introducing a new unstable state in which both bits i and j are 1; we are effectively forcing the bits i and j to change one *after* the other.
- To be clear... We want to move from state i to state j:

$$0\ldots0\ \underbrace{1}_{\text{bit } i}\ 0\ldots0\ \underbrace{0}_{\text{bit } j}\ 0\ldots0 \rightarrow 0\ldots0\ \underbrace{0}_{\text{bit } i}\ 0\ldots0\ \underbrace{1}_{\text{bit } j}\ 0\ldots0$$

but instead we do this:

$$0\ldots0\ \underbrace{1}_{\text{bit } i}\ 0\ldots0\ \underbrace{0}_{\text{bit } j}\ 0\ldots0 \rightarrow 0\ldots0\ \underbrace{1}_{\text{bit } i}\ 0\ldots0\ \underbrace{1}_{\text{bit } j}\ 0\ldots0 \rightarrow 0\ldots0\ \underbrace{0}_{\text{bit } i}\ 0\ldots0\ \underbrace{1}_{\text{bit } j}\ 0\ldots0$$

so we are forcing bit j to change before bit i.

Avoiding races during state assignment — one hot encoding

- Example...

State assignment	Current state	Next state $x_2x_1 = 00$	$x_2x_1 = 01$	$x_2x_1 = 11$	$x_2x_1 = 10$
0001	*a*	[*a*]	[*a*]	*c*	b
0010	*b*	*a*	[*b*]	*d*	[*b*]
0100	*c*	[*c*]	*b*	[*c*]	d
1000	*d*	*c*	*a*	[*d*]	[*d*]

- Races exist if we converted to a transition table using this state assignment.

Avoiding races during state assignment — one hot encoding

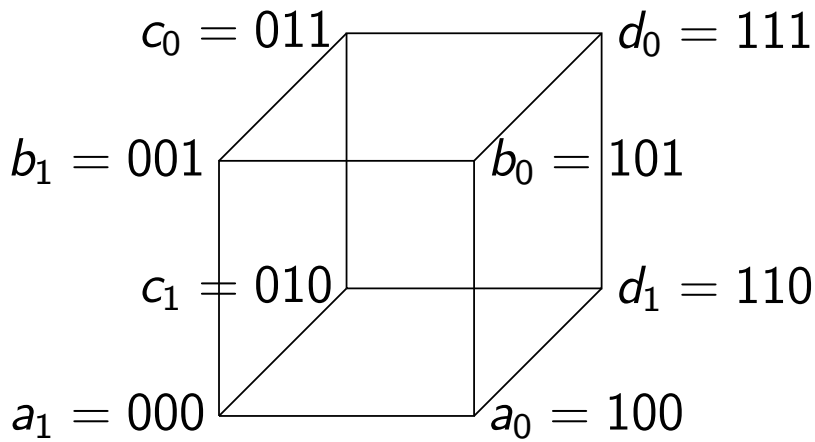
- But, introducing new unstable states...

State assignment	Current state	Next state $x_2x_1 = 00$	$x_2x_1 = 01$	$x_2x_1 = 11$	$x_2x_1 = 10$
0001	a	**a**	**a**	e	f
0010	b	f	**b**	g	**b**
0100	c	**c**	h	**c**	i
1000	d	i	j	**d**	**d**
0101	e	—	—	c	-
0011	f	a	—	—	b
1010	g	—	—	d	-
0110	h	—	b	—	-
1100	i	c	—	—	d
1001	j	—	a	—	-

- The always unstable state e permits transitions between states a and c; f permits transitions between states a and b; g permits transitions between states b and d; h permits transitions between states b and c; i permits transitions between states c and d; and j permits transitions between states a and d.

Avoiding races during state assignment — state duplication

- This method works if the number of states is ≤ 4. It works by duplicating each state into a pair of equivalent states.
- It is easy to visualize if you draw a cube and label the corners of the cube with their coordinates. Assume we have states a, b, c and d and we will duplicate each state; e.g., a becomes a_1 and a_0, and so forth.



- Notice two things: i. One can always move between equivalent states by changing only 1 bit because they are adjacent on the cube; ii. Each original state is adjacent to every other state once we take into account equivalent states and can therefore get from any state to any other state by changing only 1 bit.

Avoiding races during state assignment — state duplication

- Example...

Current state	Next state $x_2x_1 = 00$	$x_2x_1 = 01$	$x_2x_1 = 11$	$x_2x_1 = 10$
a	a	a	c	b
b	a	b	d	b
c	c	b	c	d
d	c	a	d	d

- With state duplication and some minor modifications to the flow table we get a new flow table (with a state assignment):

State assignment	Current state	Next state $x_2x_1 = 00$	$x_2x_1 = 01$	$x_2x_1 = 11$	$x_2x_1 = 10$
000	a_1	a_1	a_1	c_1	b_1
100	a_0	a_0	a_0	a_1	b_0
001	b_1	a_1	b_1	b_0	b_1
101	b_0	a_0	b_0	d_0	b_0
010	c_1	c_1	c_0	c_1	d_1
011	c_0	c_0	b_1	c_0	d_0
110	d_1	c_1	a_0	d_1	d_1
111	d_0	c_0	d_1	d_0	d_0

- We change unstable entries as required to sometimes first pass through the equivalent state rather than move directly to the desired state. An example would be stable state c_1 with $x_2x_1 = 00 \rightarrow 01$ — rather than trying to go directly to state b_1 or b_0 we temporarily pass through state c_0 (which then goes to b_1).