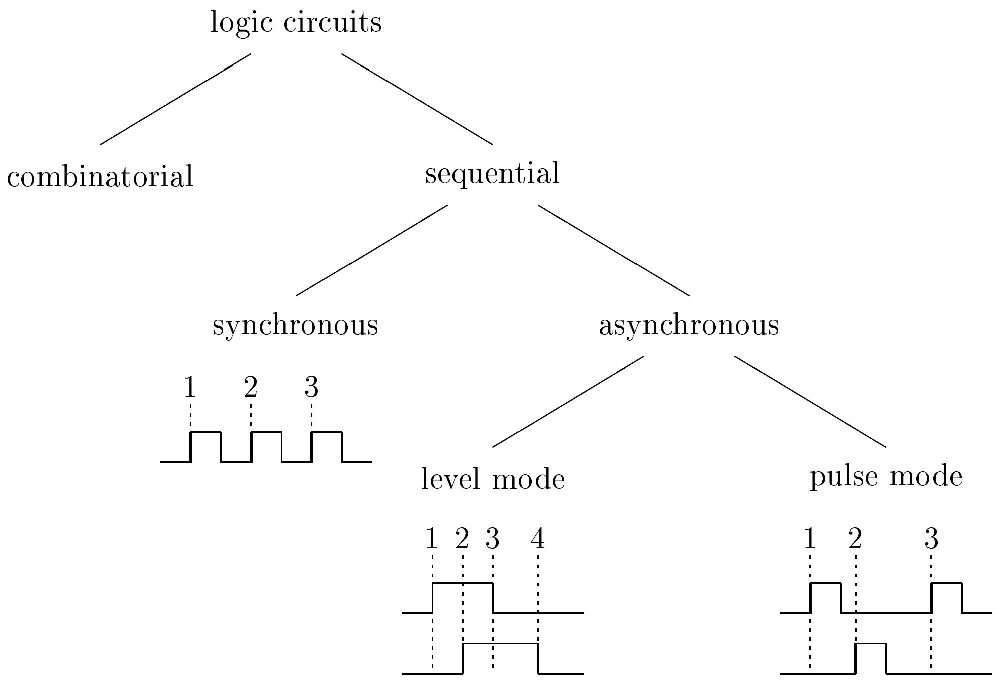
8. Sequential circuits

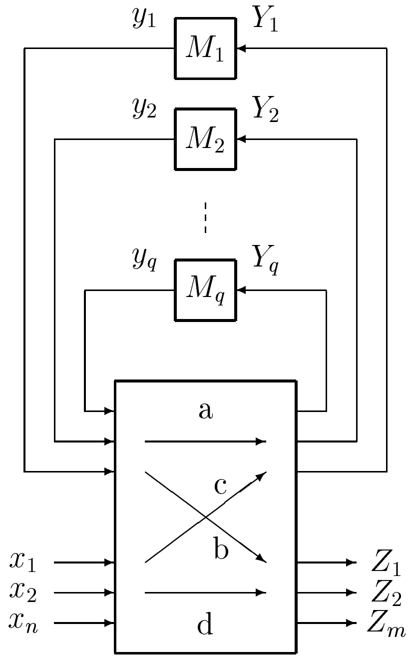
Circuits are devided between **combinatorial** and **sequential**. As already remarked before, a combinatorial circuit consists of only ordinary gates and has no feedback loops. Always: inputs $\rightarrow$ outputs, also in all subcircuits. If feedback loops are present, or if FF's are used, we speak of a sequential circuit because the next state will depend on the previous states. Note: a FF is a combinatorial circuit with two feedback loops.

Sequential circuits are further devided between **synchronous** and **asynchronous**. Synchronous means: all FF's are controlled by one common clock signal, which is mostly a periodic squarewave signal with a fixed frequency and all FF's may change at the same time, i.e. if they *must* change. This clock is absent with asynchronous circuits, so FF's always act according to their input signals. This can be on the basis of the signal levels (level mode) or on the basis of flanks or edges (pulse mode).



We have already seen examples of sequential circuits: the counters on page 47 (asynchronous level mode), and on pages 49 and 51 (asynchronous pulse mode, but can be easily made synchronous). Different types of counters are obvious examples of sequential circuits. However, they are quite common, also in the kitchen: think of your laundry washing machine. In computers they can be very complex, especially if they need to work at the highest clock speeds: memory and bus controllers in multiprocessor systems for example.

8.1 Huffman model; Mealy and Moore machines



combinatorial circuits

At the right is shown the general scheme of sequencial circuits, the **Huffman model** (D.A. Huffman, 1954). We will call:

$x_1 \ldots x_n$: primary variables
$y_1 \ldots y_q$: secondary variables
$Y_1 \ldots Y_q$: secondary functions
$Z_1 \ldots Z_m$: primary or output functions

• The M's can be flip-flops
• The M's can be simple connections (feedback loops)

The general form shown here is also called a **Mealy machine** (G.H. Mealy, 1955). If one or more of the combinatorial circuits a,b,c,d is missing, different simplifications are obtained. One very common simplification occurs when the circuit **d** is missing: the so-called **Moore machine** (E.F. Moore, 1956).

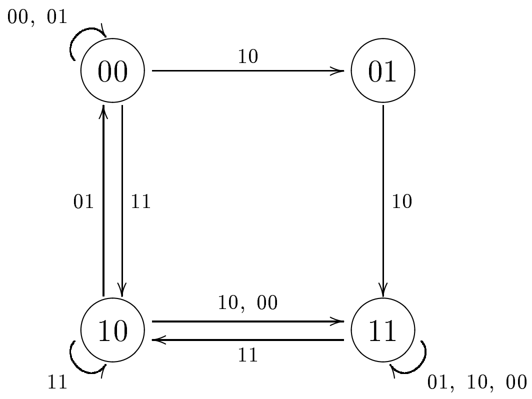
The most important simplifications are:
• There is no **d** circuit, the Z's depend only on y. This can be applied when the outputs are not important during changes of the inputs.
• If the M's are memory elements and there is no **a** circuit, then the Y's are controlled only by x's. The memory elements can "remember" only one previous state.
• If the M's are simple feedback loops and there is no **a** circuit, the memory function is lost and the circuit has become a combinatorial one.
• If circuit **b** is missing, the memory elements cannot influence the outputs. The entire circuit can be replaced by circuit **d**.
• If **c** is missing and the circuit is in a stable state, then **b** cannot change anymore. The circuit is combinatorial.
• If **c** is missing and the circuit is in an unstable state, then eventually a stable state can be reached. See previous point. However, if no stable state can be reached, the M–**a** closed loop will cause an oscillation. This is no more a sequential circuit with outputs that depend on the inputs and the previous state.

8.2 State or flow diagram

Design problems may be so complex that a simple analysis, as was done for the counters in the previous chapter, is impossible. In such cases a more systematic approach is required. In this approach we split the design problem in two steps: *analysis* and *synthesis.*

A state diagram is based on the combinations of the secondary functions Y_1 – Y_q. In the analysis these combinations are represented by a binary number and in the synthesis by a sequence or state number.

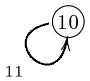
Example of analysis:



• There are two secondary functions Y_1 and Y_0. 01 means $Y_1 = 0$ and $Y_0 = 1$.

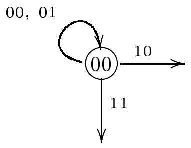
• There are two primary variables x_1 and x_0. $\xrightarrow{10}$ means inputs $x_1 = 1$ and $x_0 = 0$.

• If a combination of the secondary functions is stable, then *at least one* combination of the primary variables exists for which this state is maintained:

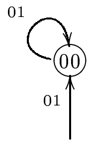


This means: combination $Y_1 = 1$ and $Y_0 = 0$ is stable for $x_1 = 1$ and $x_0 = 0$, so $Y_1 = y_1$ and $Y_0 = y_0$.

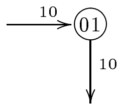
• To make a *complete* analysis, all x_1x_0 arrows for a stable state must be specified (00, 01, 10, 11).



• Furthermore, an arrow pointing at a stable state implies that the corresponding input variables are keeping that stable state.

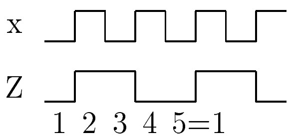


• If 01 belongs to an unstable state, this implies that the input combination leads further to a stable or another unstable state:

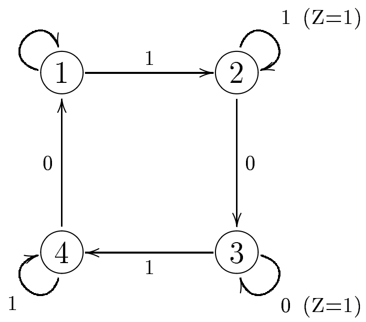


In circuit *synthesis* the Y's are not yet known at the start. Known are the stable states, because these are required. Stable states are given consecutive numbers.

Example of synthesis: design a divide-by-two counter



state	x	Z
1	0	0
2	1	1
3	0	1
4	1	0
1	0	0



If the number of states is large, a state diagram may be complicated and less suitable for simplification methods. It is for example difficult to see whether states are equivalent or not. More suitable are tables: **state-table** or flow-table. In the case of the divide-by-two counter:

secondary variables y_1y_0	primary variables $x = 0$	$x = 1$
00	(00)	01
01	11	(01)
11	(11)	10
10	00	(10)

Y_1Y_0 secondary functions

Analysis

	$x = 0$	$x = 1$
1	(1)	2
2	3	(2)
3	(3)	4
4	1	(4)

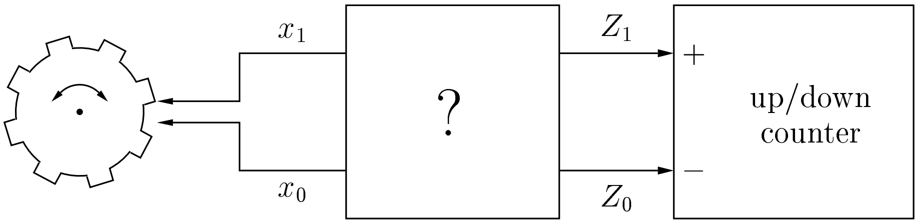
Synthesis

We see that for stable states $Y_i = y_i$. In general, a change of the primary variables leads first to another position in the same **row**, and then to another position in that **column**. The intermediate state is unstable. Exceptions will be treated later (races).

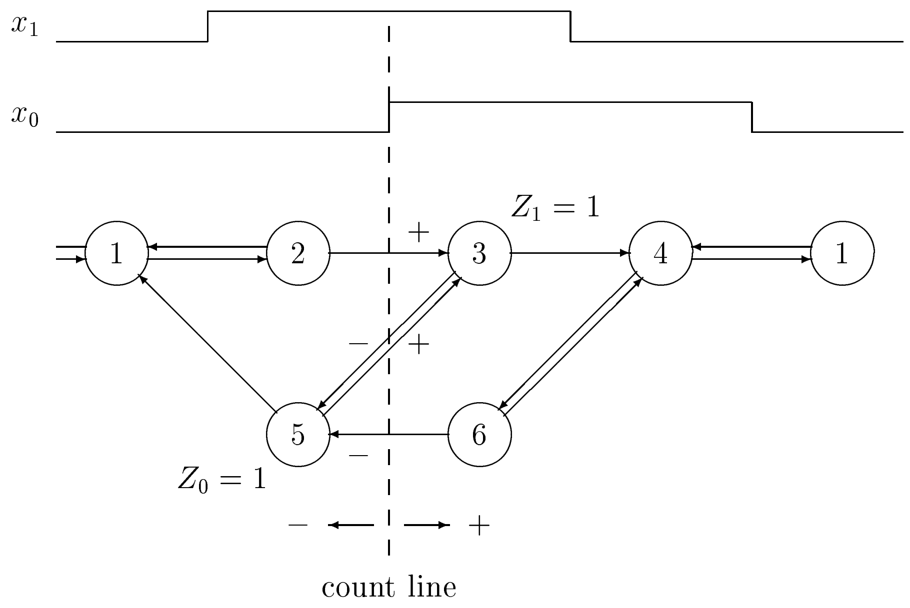
$$(00) \rightarrow 01 \;= \text{unstable}$$
$$\downarrow$$
$$(01)$$

• If there is only one stable state in each row, we speak of a *primitive* state table.

Example to be used further in the text: make a circuit that allows to read the position of an axis by means of a cam-disc and an up/down counter:



The principle is very simple. At the input we have a cam disc with two microswitches. If the disc rotates, the switches will be opened and closed in certain sequences. Depending on these sequences, the circuit must give a pulse to the counter: either increment (+ input) or decrement (– input). The circuit must determine the rotation direction of the disc, e.g.
rotation to the right: x_1 becomes 1 first, then x_0
rotation to the left: x_0 becomes 1 first, then x_1.
However, there is the possibility that the rotation orientation changes before both inputs are 1, so we first start with making a diagram with all possibilities and by making a flow diagram:



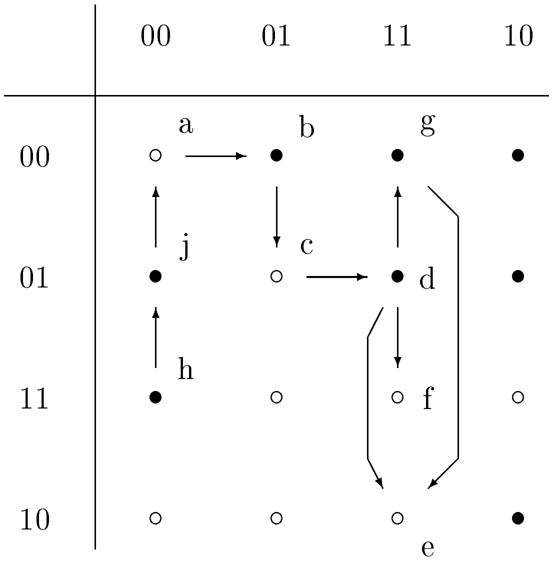
Explanation: After drawing the signals x, we start with drawing circles from left to right where the signals have the combinations 00, 10, 11, and 01, filling arbitrary numbers 1, 2, etc. Then we define a count line where both signals become 1. If we pass this line going to the right, we need to increment the counter (+), if we pass going to the left we need to decrement (–). From state 3 going to the left, we cannot pass via state 2 because the disc could stop and turn right again: we need another state 5. Try to understand the entire diagram. Once the diagram is complete, we can fill the state table:

	x_1x_0 00	01	11	10	Z_1	Z_0	
1	1	4	–	2	0	0	start state; no count information
2	1	–	3	2	0	0	ready for increment
3	–	4	3	5	1	0	passed count line: increment
4	1	4	6	–	0	0	avoid state 3 in rotating left
5	1	–	3	5	0	1	passed count line: decrement
6	–	4	6	5	0	0	ready for decrement

8.3 Races

The figure below shows the state table and the output table of a Mealy machine with two inputs, two secundary functions and one output function. Also shown is a schematic representation of the state table in order to illustrate state transitions. Stable states ($Y_i = y_i$) are shown by ∘'s, unstable states by •'s.

	x_1x_0				x_1x_0			
y_1y_0	00	01	11	10	00	01	11	10
00	00	01	10	11	0	0	0	0
01	00	01	10	11	0	0	0	0
11	01	11	11	11	0	0	0	0
10	10	10	10	11	1	0	0	0
	Y_1Y_0				Z			



Let's see what can happen. Suppose the circuit is in state **a**. The input combination 01 leads to state **b** (unstable) and then to **c** (stable).

If now the input becomes 11, the Y_1Y_0 need to change from 01 to 10, and the circuit goes to **d** (unstable). For flip-flops this means that Y_1 must be set and that Y_0 must be reset, but the outputs y_1 and y_0 are still, for a short time, 0 and 1. There are now three options:

• If both flip-flops change at *exactly* the same time, the circuit will go to state **e** (stable). However, *exactly* the same time is almost impossible in practice because of small differences in delays....

• If Y_1 changes first, we will have for a short time $y_1y_0 = 11$ and the circuit will go to state **f** (stable).

• If Y_0 changes first, we will have for a short time $y_1y_0 = 00$ and the circuit will go to state **g** (unstable), with $Y_1Y_0 = 10$ and this leads to the stable state **e**.

Hence, depending on the difference in speed between Y_1 and Y_0, either the stable state **f** or the stable state **e** is reached.

The situation in which more than one secondary variable must change is called a **race**. If the final state depends on the order in which the secondary variables change, this is called a **critical race**. If there are two secondary variables, we need to check all transitions $00 \leftrightarrow 11$ and all $01 \leftrightarrow 10$.

One solution for solving critical races is to select flip-flops with exactly the same delay, but even then, after using circuits for a certain time the characteristics may change. It is therefore better to avoid critical races. If impossible, unstable states can be added such that races become uncritical. This will be explained later on.

In the above figure: state **f** with inputs 00 leads to the unstable states **h** and **j** to **a**. No critical races are possible because for each change only one Y changes.

Check: What happens if in state **a** the inputs become 10? Is this race critical?

8.4 Synthesis of sequential circuits

With primary variables and required stable states, a **primary state table** is made first. This was already illustrated on page 59. States are defined by the state *numbers* as well as by the *row* and *column* in which they appear. Below: if we speak about a state, both state number and row/column position are meant.

8.4.1 Equivalences

An incomplete definition of the behaviour of a circuit can lead to equivalent stable states in the primary state table. **Equivalent states** have the same input combination, the same output functions and go to the same next states for all new input combinations. Equivalent states can be detected by comparing rows in the primary state table.
If for two rows:
(1) the stable states are in the same column,
(2) the output combinations are equal and
(3) all other columns have the same numbers,
then the two rows indicate equivalent states.

Theorem: equivalent rows can be reduced to a single row.
Proof: trivial, rename one stable state.

Example: the Moore machine shown to the right.
• In this table rows 2 and 4 can be unified by changing all 4's to 2's. Row 4 can be deleted.
• If all 4's have been changed to 2's, we see that now rows 1 and 3 are also equivalent: change all 3's to 1's and delete row 3.
As a result, we have eliminated 2 rows out of the seven and we obtain the reduced table shown to the right.

This analysis is simple and straightforward for small tables, but in bigger tables it may be difficult to see what happens. We then need a more systematic approach. One such approach will be studied using another example.

	x_1x_0					
	00	01	11	10	Z_1	Z_0
1	1	2	–	6	0	0
2	1	2	5	–	1	0
3	3	4	–	6	0	0
4	1	4	5	–	1	0
5	–	7	5	6	0	1
6	3	–	5	6	1	1
7	3	7	5	–	1	1

	x_1x_0					
	00	01	11	10	Z_1	Z_0
1	1	2	–	6	0	0
2	1	2	5	–	1	0
5	–	7	5	6	0	1
6	1	–	5	6	1	1
7	1	7	5	–	1	1

	x_1x_0					
	00	01	11	10	Z_1	Z_0
1	1	2	–	3	0	0
4	4	7	–	8	0	0
12	12	2	–	8	0	0
3	1	–	6	3	0	1
8	12	–	11	8	0	1
5	–	9	5	3	1	0
6	–	2	6	10	1	0
11	–	2	11	10	1	0
2	4	2	5	–	1	1
7	12	7	6	–	1	1
9	1	9	11	–	1	1
10	1	–	6	10	1	1

This is a Moore machine and the outputs are already sorted together because this is easier to see. From this table we first construct another table which is shown below.
Horizontal: make three columns of which the first two are marked "done" and $\not\equiv$ (not equivalent). Then make columns for all row combinations that *could be* equivalent because the stable states are in the same column and the outputs are the same. E.g., for outputs 00 we have 1-4, 1-12 and 4-12.
Vertical: make rows with the required equivalences for making a combination equivalent, looking at all stable states in the columns. E.g., all combinations between (1,4,12), then (3,8,10), then (5,6,11), etc.
We can now start filling the table.

done	$\not\equiv$		1-4	1-12	4-12	3-8	5-6	5-11	6-11	2-7	2-9	7-9
$\surd$	x	1-4									x	
		1-12				x						x
$\surd$	x	4-12								x		
		3-8	x	x								
$\surd$	x	5-6								x		
$\surd$	x	5-11									x	
		6-11				x						x
$\surd$	x	2-7	x		x							
$\surd$	x	2-9					x	x				
		7-9										
$\surd$	x	3-10					x	x				

1) Indicate, with a x, in each column which combinations must be equivalent to make the column combination equivalent. In the state table we can see that for combination 1-4 these are 2-7 and 3-8.
2) Start the search with a combination which is certainly *not* equivalent, e.g. because of unequal outputs. In this example we can start with 3-10. Because 3-10 is not equivalent, the combinations 5-6 and 5-11 are also not equivalent. Put a **x** in the $\not\equiv$ column in row 3-10 and then in rows 5-6 and 5-11.
3) We are ready with row 3-10, so put a $\surd$ in the "done" column.
4) Proceed with another combination which is not equivalent, e.g. 5-6 or 5-11. If 5-6 not equivalent, then 2-7 neither. And so on.
5) The rows which are empty in the "done" and $\not\equiv$ columns, after completing the procedure, indicate equivalent rows of the state table. In this example the equivalent states

are 1-12, 3-8, 6-11 and 7-9. Hence, 12 → 1, 8 → 3, 11 → 6 and 9 → 7. We can delete the rows 12, 8, 11 and 9, obtaining a reduction from 12 to 8 rows:

	x_1x_0 00	01	11	10	Z_1	Z_0
1	1	2	–	3	0	0
4	4	7	–	3	0	0
3	1	–	6	3	0	1
5	–	7	5	3	1	0
6	–	2	6	10	1	0
2	4	2	5	–	1	1
7	1	7	6	–	1	1
10	1	–	6	10	1	1

8.4.2 Pseudo equivalences

Often certain changes of the primary variables are impossible because of the conditions of the problem. These are indicated by a – (don't care) in the state table. Again, a don't care can be replaced by any combination of the secondary functions. By assuming clever states, some rows can be made equivalent with other rows. This is called a **pseudo equivalence**. This can also be done for rows which are equivalent except for output values, if the latter contain don't cares. Two examples:

1	3	–	–	0	0
2	3	–	4	0	0

→

1	3	–	4	0	0

1	3	–	4	0	0
2	3	–	4	0	–

→

1	3	–	4	0	0

Often, the reduction of the number of rows by means of pseudo equivalences is not trivial because one can choose between multiple solutions.

8.4.3 Mixing of rows

The **mixing of rows** is in fact the same procedure as pseudo equivalenc, but now by merging unstable and stable states on the basis of don't cares. In mixing a stable with an unstable state, the stable one has priority. The outputs need not be equal, but: a Moore machine turns into a Mealy machine! For a Mealy machine, mixing is only possible with the same outputs and taking don't cares into account. Example: the axis problem on pages 59-60. At the left is the original state table, at the right the result of mixing rows

1-2, 3-5 and 4-6:

	x_1x_0 00	01	11	10	Z_1	Z_0
1	1	4	–	2	0	0
2	1	–	3	2	0	0
3	–	4	3	5	1	0
4	1	4	6	–	0	0
5	1	–	3	5	0	1
6	–	4	6	5	0	0

	Y_1Y_0 (x_1x_0) 00	01	11	10	Z_1Z_0 (x_1x_0) 00	01	11	10
1	1	4	3	1	00	- -	- -	00
3	1	4	3	3	- -	- -	10	01
4	1	4	4	3	- -	00	00	- -

You can try yourself another "mix" by merging rows 1-4 and rows 3-5. Generally, multiple mixing possibilities occur. It is not easy to select the easiest one in advance. In the further synthesis *all* possibilities must be examined, finally selecting the most simple circuit.

8.4.4 Assigning the secondary functions

By means of equivalence and pseudo equivalence the number of states has been reduced. By means of mixing rows the number of combinations of the secondary functions has been reduced. For realising the sequential circuit, we must now assign flip-flops to the Y's.

Suppose there are left n different combinations of the scondary functions (n is the number of rows in the resulting state table). These n combinations can be realised with $k \geq [\log_2 n]$ functions Y_i. If $n = 7$, then $k \geq [\log_2 7] = 3$. In the case of our example "axis position" we have $n = 3$ and therefore $k = 2$. Hence, we need two flip-flops.

We can make different assignments for the states 1, 3 and 4:

AS1	Y_0 = 0	1
Y_1 = 0	1	3
1	4	a

AS2	Y_0 = 0	1
Y_1 = 0	1	3
1	b	4

AS3	Y_0 = 0	1
Y_1 = 0	1	4
1	c	3

We will start with studying assignment **AS1**. The transitions between state 1 ($Y_1 = Y_0 = 0$) and 3 ($Y_1 = 0, Y_0 = 1$), as well as between 1 and 4 ($Y_1 = 1, Y_0 = 0$) are *uni-variant.* I.e. only one secondary function changes. In these cases no critical races can occur, so we don't need to study these races in the state table. However, in the table also transitions between states 3 and 4 occur, so the Y's go from 01 to 10 and vice versa. These transitions are *bi-variant,* the two Y's change which often goes via state 1 (00) or via state a (11).

• Transition $3 \rightarrow 4$ occurs for input 01, the circuit follows the **cycle** 3 → 1 → 4. This is shown in the table below at the left. If the transition $3 \rightarrow 4$ goes via a ($01 \rightarrow 11 \rightarrow 10$), a similar cycle occurs if we add row a: in row a we place state 4 in column 01.

This is shown in the table below at the right:

		00	01	11	10
00	1	1	4	3	1
01	3	1	4	3	3
10	4	1	4	4	3

		00	01	11	10
00	1	1	4	3	1
01	3	1	4	3	3
10	4	1	4	4	3
11	a	–	4	–	

Both cycles $3 \to 1 \to 4$ and $3 \to a \to 4$ constitute an *uncritical* race, because they end both in the same state.

• Going from 4 to 3 if the input is 10, the cycle $4 \to 1 \to 3$ or $10 \to 00 \to 01$ could be followed (below left). However, 1 is a stable state, so this cannot be allowed. This can be solved by *forcing* the circuit to go always via state a (see below right): put in row 4 an a for input 10 and in row a state 3.

		00	01	11	10
00	1	1	4	3	1
01	3	1	4	3	3
10	4	1	4	4	3
11	a	–	4	–	

		00	01	11	10
00	1	1	4	3	1
01	3	1	4	3	3
10	4	1	4	4	a
11	a	–	4	–	3

Now all problems have been solved and the assignment can be included in the state table. Note that y_1y_0 has been re-ordered to make directly a Karnaugh diagram:

	x_1x_0				x_1x_0			
y_1y_0	00	01	11	10	00	01	11	10
00	00	10	01	00	00	- -	- -	00
01	00	10	01	01	- -	- -	10	01
11	- -	10	- -	01	- -	- -	- -	- -
10	00	10	10	11	- -	00	00	- -
	Y_1Y_0				Z_1Z_0			

Solution for **AS1**:
$Y_1 = y_1\bar{y}_0x_1 + \bar{x}_1x_0$
$Y_0 = y_0x_1 + \bar{y}_1x_1x_0 + y_1x_1\bar{x}_0$
$Z_1 = y_0x_0$
$Z_0 = y_0\bar{x}_0$

This solution is not necessarily the most simple one. We need to check also assignments AS2 and AS3.

For **AS2**:

	x_1x_0 00	01	11	10	x_1x_0 00	01	11	10
1	1	4	3	1	00	- -	- -	00
3	1	4	3	3	- -	- -	10	01
4	1	4	4	3	- -	00	00	- -
b	1	4	–	–	- -	- -	- -	- -

AS2	Y_0 = 0	1
Y_1 = 0	1	3
1	b	4

- $1 \to 3$ is univariant.
- $1 \to 4$ is bivariant: either $1 \to 3 \to 4$ or $1 \to b \to 4$. By putting in row b column 01 a 4, a race is created. However, this is uncritical.
- From 3 all changes are univariant.
- From 4 to 3 is univariant.
- $4 \to 1$ via 3 or via b. In row b column 00 we put a 1.

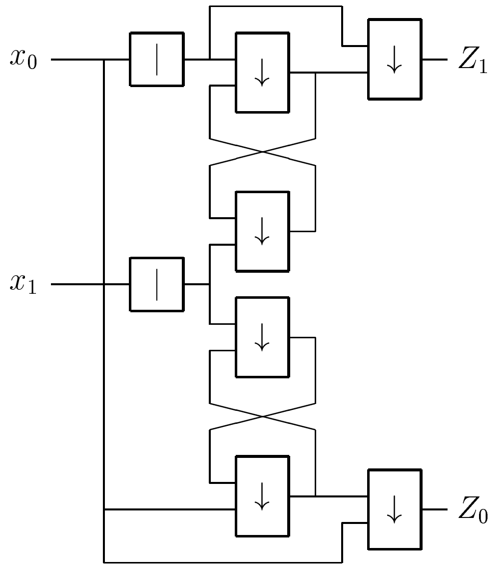
We obtain the assignment:

y_1y_0	x_1x_0 00	01	11	10	x_1x_0 00	01	11	10
00	00	11	01	00	00	- -	- -	00
01	00	11	01	01	- -	- -	10	01
11	00	11	11	01	- -	00	00	- -
10	00	11	–	–	- -	- -	- -	- -
	Y_1Y_0				Z_1Z_0			

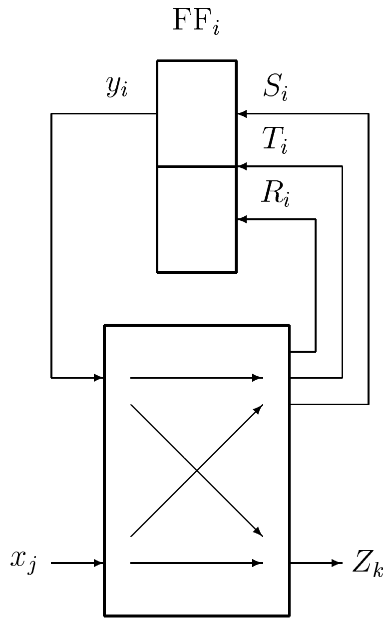
Solution for **AS2**:
$Y_1 = y_1x_0 + \bar{x}_1x_0$
$Y_0 = y_0x_1 + x_0$
$Z_1 = \bar{y}_1x_0$
$Z_0 = y_0\bar{x}_0$

We see that the AS2 solution is more simple than the AS1 solution.
Check yourself: the AS3 assignment leads to the solution
$Y_1 = y_1x_1 + y_0x_1\bar{x}_0 + \bar{y}_0x_1x_0$
$Y_0 = y_0x_1 + y_1x_1 + \bar{x}_1x_0$
$Z_1 = y_1x_0$
$Z_0 = y_1\bar{x}_0$

Because the AS3 solution is also more complex than the AS2 one, we will realise the AS2 solution by means of NOR gates:



In this realisation, the M elements in the general Huffman model are feedback loops from gate outputs to inputs. This results in a memory function. Recall that gates plus feedback form SR flip-flops. If the memory function is incorporated into the M elements, i.e. M = SR flip-flop, this results in the same circuit. But in this case an alternative approach is possible:



The FF's can be of any type, i.e. SR, SR with T, T, etc. We will take a look at realising the "axis-position" circuit by means of SR and T flip-flops.

• Using SR flip-flops, the set and reset functions can be derived directly from the state table. Applying the AS2 assignment, we get the following table for Y_1:

y_1y_0 \ x_1x_0	00	01	11	10
00	0	$\underline{1}$	0	0
01	0	$\underline{1}$	0	0
11	$\underline{0}$	1	1	$\underline{0}$
10	$\underline{0}$	1	–	–

FF1 *must* get a set if FF1=0 and $Y_1 \to 1$.
FF1 *may* get a set if FF1=1 and $Y_1 \to 1$.
If in a row with $y_1 = 0$ a $Y_1 = 1$ occurs, this means that FF1 must get a set for the corresponding input combination. This is indicated by underlining the 1 in the state table. In the rows with $y_1 = 1$ a set may be given for $Y_1 = 1$, but not necessarily: a don't care. Hence, $S_1 = \bar{x}_1x_0$.

Similar for the reset function:
FF1 *must* get a reset if $y_1 = 1$ and $Y_1 = 0$; underline the 0's.
FF1 *may* get a reset if $y_1 = 0$ and $Y_1 = 0$. Hence $R_1 = \bar{x}_0$. The analysis for FF0 is exactly the same. This procedure is much more simple than the one followed in the chapter "memory elements."

• Using the trigger function or T flip-flops; the FF always changes its state. This implies that all underlined 1's and 0's in the above table must be taken together. Now the not-underlined 1's and 0's may not be taken as don't cares, only the original don't cares at the bottom line to the right! We then get the following Karnaugh diagram for FF1, from which we obtain the function $T_1 = \bar{y}_1\bar{x}_1x_0 + y_1\bar{x}_0$.

	1		
	1		
1			1
1			1

Check yourself: for FF0 we get $S_0 = x_0$, $R_0 = \bar{x}_1\bar{x}_0$ and $T_0 = y_0\bar{x}_1\bar{x}_0 + \bar{y}_0x_0$. In this example the set and reset functions are more simple than the trigger functions, so it is better to use SR FF's instead of T FF's. Finally, the output functions Z_1 and Z_0 must be realised, but these have already been determined.

8.4.5 Increasing the number of secondary functions

Shown below is a state table which, after assigning two secondary functions, shows critical races. The states marked with an $*$ lead to critical races.

x_1x_0	00	01	11	10
1	(1)	2	3	3
2	(2)	(2)	3	4
3	(3)	4	(3)	(3)
4	1	(4)	3	(4)

Y_1 \ Y_0	0	1
0	1	2
1	4	3

y_1y_0 \ x_1x_0	00	01	11	10
00	(00)	01	11*	11*
01	(01)	(01)	11	10*
11	(11)	10	(11)	(11)
10	00	(10)	11	(10)

Y_1Y_0

In row 00, from stable state 00 with input 10 the state 11* is reached and than the stable state 11 if both Y's change at the same time. If not at the same time, the stable state 10 is reached. So this is a critical race. From 00 with input 11 a race is also entered, but this one is uncritical. There is no Y combination free which could be used to avoid the critical race. The same problems arise for all other assignments!

By assuming an additional secondary function, the number of Y combinations is doubled, so we get more possibilities to create uni-variant changes and to convert races into cycles. A new assignment with three Y's:

	Y_1Y_0	00	01	11	10
Y_2	0	1	2	c	3
	1	4	a	d	b

x_1x_0	00	01	11	10
1	1	2	3	3
2	2	2	3	a
3	3	b	3	3
4	1	4	3	4
a	–	–	–	4
b	–	4	–	–
c	–	–	–	–
d	–	–	–	–

x_1x_0	00	01	11	10
000	000	001	010	010
001	001	001	010	101
010	010	110	010	010
100	000	100	010	100
101	—	—	—	100
110	—	100	—	—
011	—	—	—	—
111	—	—	—	—

• The column with input 11 can never give critical races because it contains only 3's. So stable state 3 is reached directly or via a cycle. From 2 and input 11 the state 3 must be reached. This can via 1 because input 11 in state 1 leads also to 3. This transition can also go via c. The table is not changed.
• $1 \rightarrow 2$ and $1 \rightarrow 3$ are uni-variant.
• From 2 to 4 with input 10 is not allowed via 1 because 1 with input 10 leads to stable state 3. The transition from 2 to 4 must go via a. So, row 2 column 10 gets an a and in row a comes a 4 in the same column.
• $3 \rightarrow 4$ gives a critical race, the circuit may end up in stable state 2. This is replaced by a cycle by including state b.
After substituting the assignment we obtain the rightmost table, from which with Karnaugh the Y functions can be determined.

NOTE: This chapter is not the complete theory behind sequential circuits. There are more complications, like noise sensitivity and power-on problems (if the power is switched on, flip-flops may get a random state, and these may lead to a stable state which is not prescribed). Furthermore, sometimes a circuit should be as fast as possible. In such cases cycles must be avoided or minimised.