Automation Systems

Digital Control Systems (DCS)

for MSST

at VGU in HCMC

SS 2011

Prof. Dr.-Ing. Hans-Werner Dorschner

Content

Copyright

Part II

Sequential Control

1. Sequential Control

Sequential control problems are controls with single steps that have to be processed in a fixed sequential manner.
The stepping from one step to the next depends on so called transition conditions that have to be fulfilled by the process.
There is always a definite relationship between each step in the control program and the technological process under control. This connection can be time based or function based. Therefore we distinguish between

⇨ time based and process based sequential controls.

Time based sequential controls use as transition conditions timer, counter etc. whereas the process based sequential controls do derive their transition conditions from process conditions like pressures, temperatures flow rates or a. o.

1.1 Status Description

The methods for designing logic control can also be used for sequential control. But it can become very difficult to respect all dependences or interlocks if we have more complex problems. Therefore a special language or design method was developed which introduces a status combined with a transition. If the transition condition it fulfilled the controller can move to the next step of his program and execute the respective commands.

1.2 Status Graph

Every controller has a special status at a definite point in time. If the controller has a memory then this status will also contain the information of the past.
The symbolic representation of those steps (status) is shown in Fig. 1-1. We see two steps (status) and the sign for the conditions (transition conditions) to step from status 1 (step 1) to status 2 (step 2).
This means the step 2 will only be reached, if the controller is in step 1 and the transition conditions for step 2 are fulfilled.

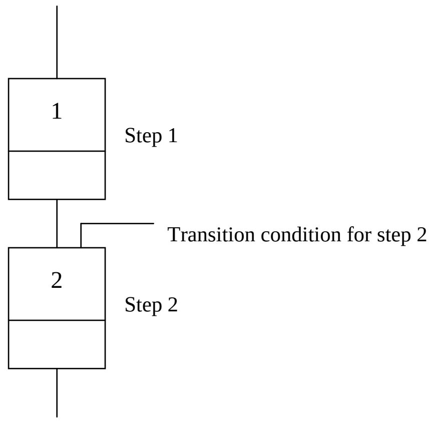


Fig. 1-1: Graph with 2 status steps

In the respective step some commands will be executed to control the process (Fig. 1-2).

Depending on the processed to be controlled there can be parallel branches in the graph or jumps to special steps (Fig. 1-3).

In case of parallel branches they have to be interlocked because the controller has to have a definite status.

In case of jumps we see the jump target step in a circle. In Fig. 1-3 we have two parallel branches.

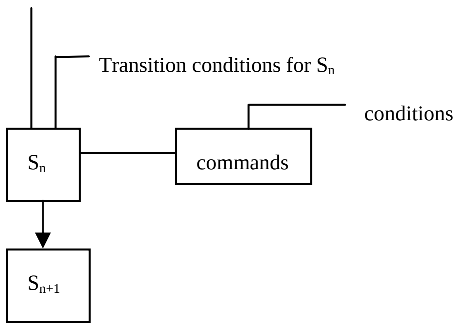


Fig. 1-2: Commands in step n

From step 2 the controller can get either to step 3 or 5. These alternatives must be interlocked. From step 4 the controller gets to step 5 and from step 6 the controller will jump – if the transition conditions are fulfilled – to step 3.

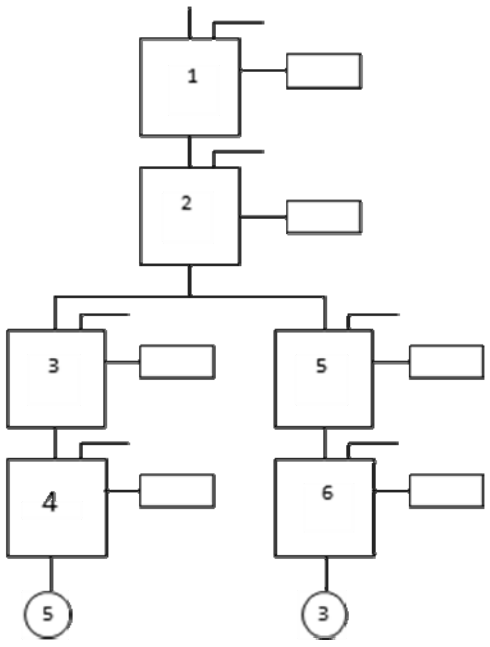


Fig. 1-3: Graph with parallel branches

1.3 Implementation of the State Graph

The state graph can be implemented in different way.
The straight forward way would be to declare one flip-flop for each state (Fig. 1-4).

The transition conditions for step 3 are led to the set input of the flip-flop of step/state 3. Here is the step 2 included in the conditions because the controller can get to step 3 only if he is in the 2 in this special example.
The following state 4 will reset the flip-flop of state 3.

Each output of the respective flip-flop represents one definite step or state. If the controller is in state 4, then the state 3 flip-flop will be reset.

At power-up of the controller the initial status of all states has to be rest. Then we get a defined initial status of the controller. The reset can either be done manually (f.e. in an OB 100) or will be done automatically with power-up depending on the CPU type.

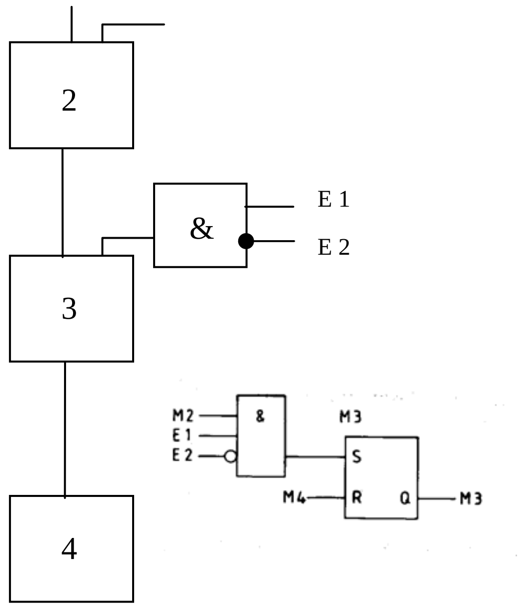


Fig. 1-4: state definition by flip-flops

As a summary we can note down the rules to implement a state graph with flip-flop:

- Every state of the state graph will be assigned a RS-flip-flop.
- The set input (S) of the flip-flop of the respective state which is a marker (memory flag) will be built by the AND combination of the preceding state and the individual transition conditions of the actual state.
- For a defined initial status all states must be reset.
- In case of branches the following states must be interlocked.
- The outputs to the process will be built as a logical function of the respective states (markers) according to the controller tasks.

1.4 Programming of Step Actions

I every step (state S_i) there will be an action processed that influences the process under control (Fig. 1-5)

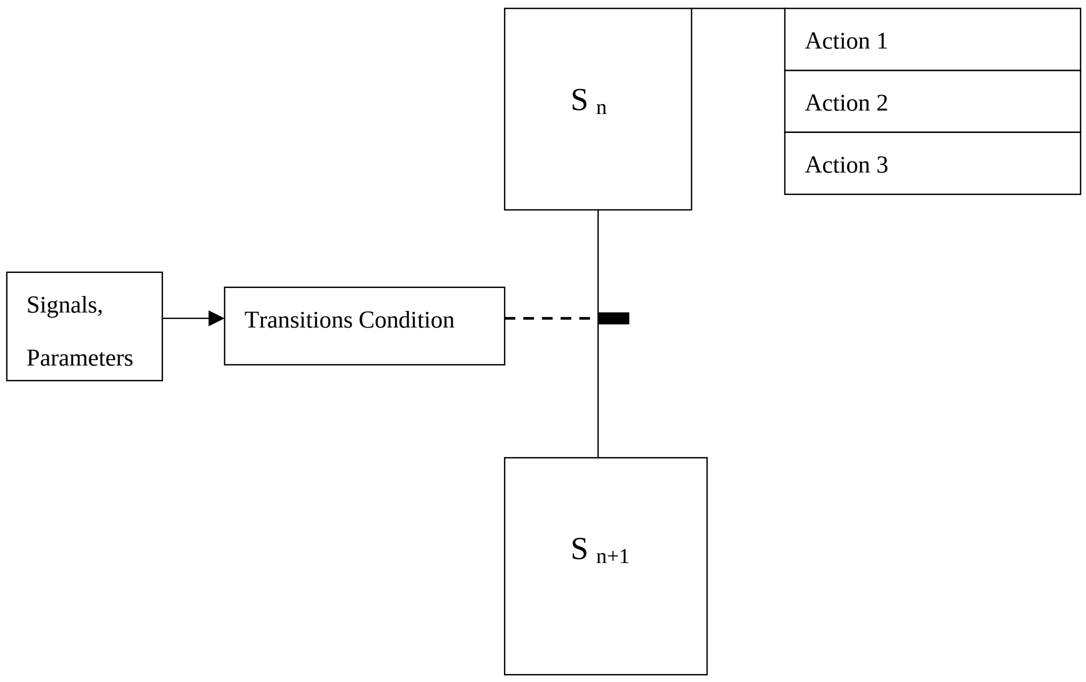


Fig. 1-5: Actions processed at a special state

An action is an operation (command) which consists of an operand and a code for the operation. As operations in a state signals can be set or reset:

- **S** **S**et output signal
- **R** **R**eset output signal
- **N** **Non holding**: Operand=1 as long as the controller is in the respective state.
- **D** **Delay:** Operand will be set to 1 after a programmed time when controller is in the respective stat and will be reset when controller leaves the state.

Besides these elementary commands there are also call-commands possible for example calling a function or a function block.

Examples will follow.

1.5 Programming or Transition Conditions

For the transition conditions we can put logical functions of signals in a conjunctive or disjunctive form (Fig. 1-6).

Besides this also compare functions are possible that compare the time in a state with a predefined value. Herewith time dependent transitions can be defined.

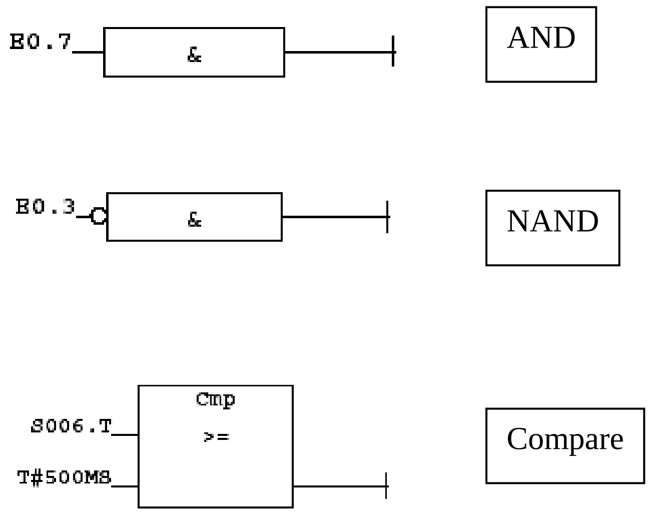


Fig. 1-6: Transition conditions programming

In general for conditional time dependent transitions there are three possibilities:

1. Using a timer: The timer gets started delayed by the defined timer word when the controller enters the state (S1). This is done in a stored manner that means that the time must be rest some when (Fig. 1-7).

   With the command SO the timer will be rest (TR timer reset), if the controller leaves the state.

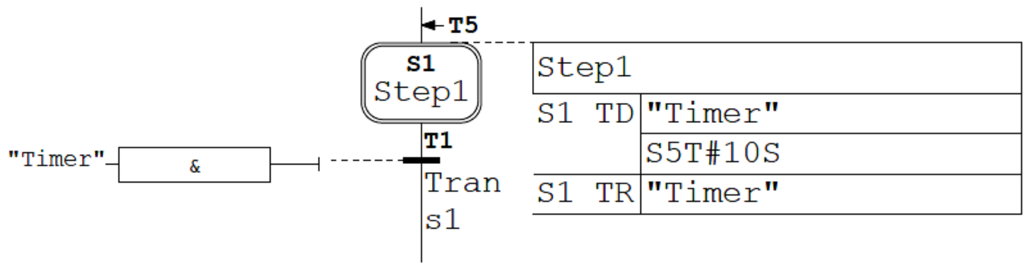


Fig. 1-7: Conditional transition time dependent

2. Using a comparing command the controller compares the time he stays in the current state compared to the programmed one (Fig. 1-8).

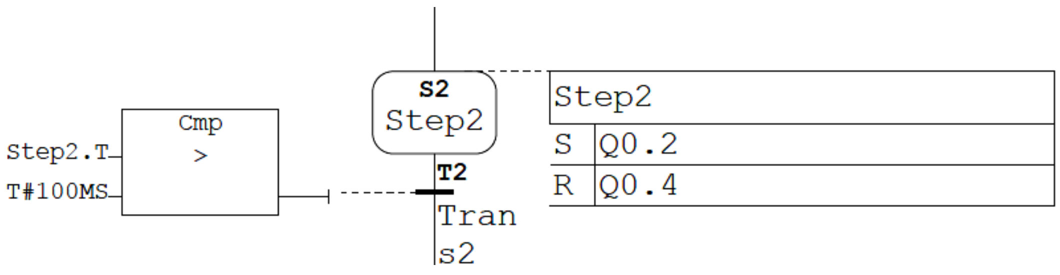


Fig. 1-8: Direct comparison with a defined time interval

3. Using a marker (memory flag). The memory flag is delayed by a predefined time value (Fig. 1-9).

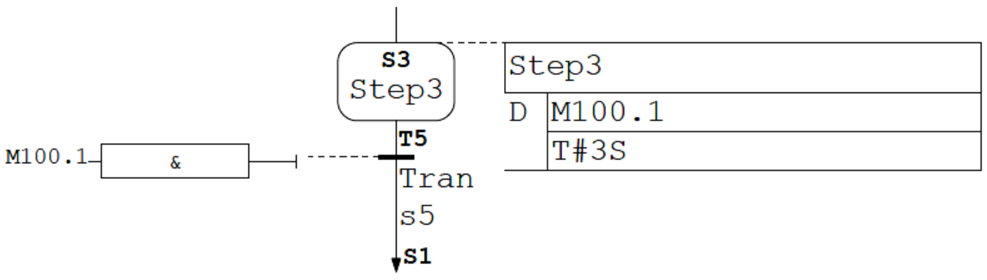


Fig. 1-9: Conditional transition using a memory flag (marker)

For clarification we will look to the following example:

Example: Filling plant

A screw conveyor transports a granulated good to a belt conveyor. The weighbridge will send a signal (S3) if the wagon is full.

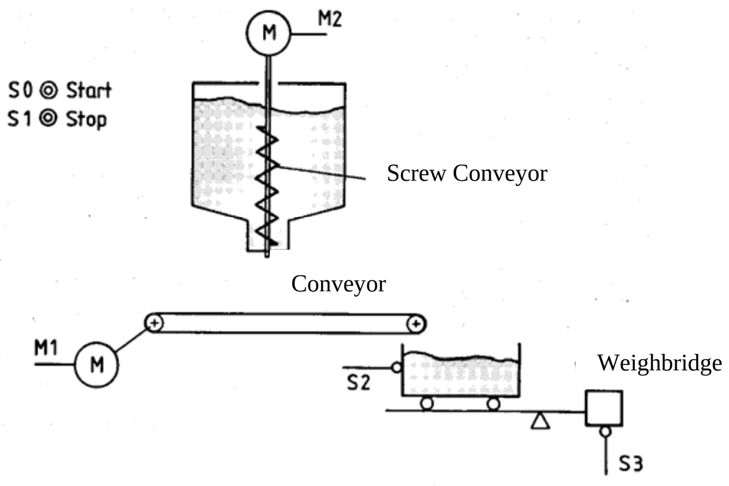


Fig. 1-10: Filling station

The filling process will be started by the switch S0 if a wagon is in his position (S2). Then the belt will be running for 3 seconds to prevent material congestion.
The filling process will be stopped if the sensor S3 indicates that the wagon is full or not in his position any more (S2) or the stop switch (S1) was operated.
In this case the belt conveyor still runs for an additional 5 seconds in order to empty the belt.
The next filling process will be started by pressing the switch S0.

We get the following allocation table:

Input Variables	**Device Tag**		**Logical Allocation**	
START switch	S0	I 0.0	operated	S0=1
STOP switch	S1	I 0.1	operated	S1=0
Sensor Ramp	S2	I 0.2	If wagon at ramp	S2=1
Sensor Weighbridge	S3	I 0.3	If wagon full	S3=1
Initialization Impulse	RI	M 40.4		
Output Variables				
Motor Conveyor Belt (Contactor Motor M1)	K1	Q 0.0	on	K1=1
Motor Screw Conveyor (Contactor Motor M2)	K2	Q 0.1	on	K2=1
Timer 1	T1		T1=3s	
Timer 2	T2		T2=5s	

With the described control task we get the following state graph (Fig. 1-11):

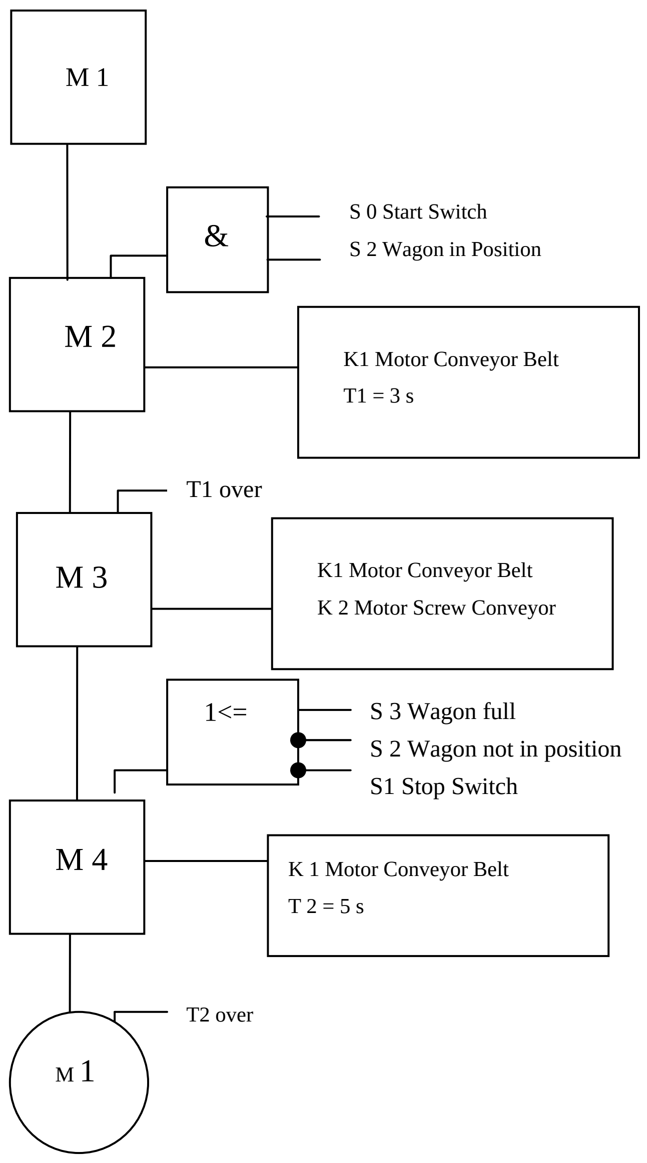


Fig. 1-11: State graph of the filling station

Exercise:

1. Derive the PLC program for a SIMATIC S7 PLC in GRAPH.
2. Derive the STL program for the state graph.

If the markers are nonremanent it will be sufficient to start step 0 with an initialization impulse.

It is important to remember that you cannot a STL or FBD directly from a state Graph (GRAPH in SIMATIC)

These SW packages are completely separately and cannot be converted into each other.

The following example demonstrates the necessity to interlock states.

Example: Traffic Light

A construction area is secured by a traffic light on the road

If the plant is powered both lights will show red.

An arriving car is detected by the respective initiator. Then after 10 s the corresponding traffic light will change to green. The green phase will last at minimum 20 s before both lights will change to 10 s on request of the other initiator which indicates oncoming traffic. Then the oncoming traffic will have a green phase of 20 s minimum.
By operating the OFF switch the plant will be shut down after finishing the actual green phase (Fig. 1-12).

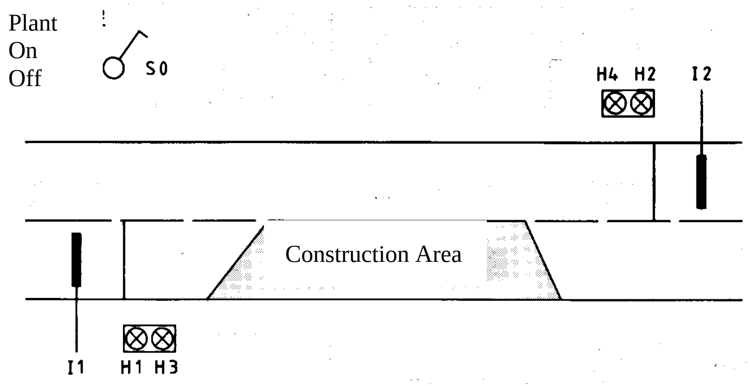


Fig. 1-12: Traffic lights at a construction area of a road

We get the following allocation table:

Input Variables	**Device Tag**		**Logical Allocation**	
On/OFF	S0	I0.0	ON	S0=1
Initiator 1	I1	I0.1	operated	I1=1
Initiator 2	I2	I0.2	operated	I2=1
Initialization Impulse	RI	M40.4		
Output Variable				
Lamp 1 green	H1	A0.0	shines	H1=1
Lamp 2 green	H2	A0.1	shines	H2=1
Lamp 1 red	H3	A0.2	shines	H3=1
Lamp 2 red	H4	A0.3	shines	H4=1

According to the description the plant can be in different states.
Because of the branch (S2 -> S3/S4) the following steps have to be interlocked. This is also true for the subsequent states 4 and 7 (Fig. 1-13).

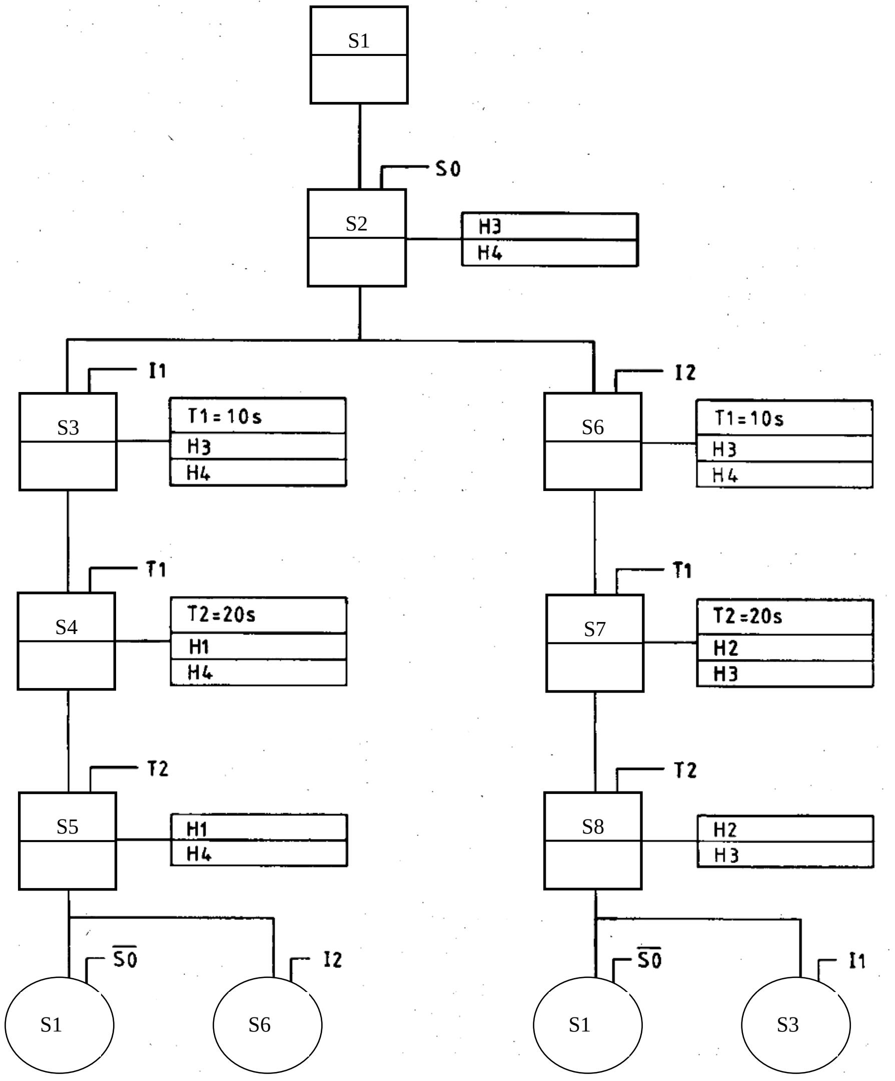


Fig. 1-13: State graph of the traffic light

Exercise:

1. Derive the PLC program for a SIMATIC S7 PLC in GRAPH (use all three different time based transition conditions)
2. Derive the STL program for the state graph.

Note:

The interlock of the states 3 and 6 is implemented automatically because of the two different branches.
If both transition conditions are fulfilled at the same time than this branch will be executed that has the highest priority that means the lowest transition number (can be assigned manually).

1.6 State Graph with Loops

If we find in a state graph a state which is in a set and reset position (like S3 and S4 in Fig. 1-14) then we call it a loop which a controller may not escape from.
Therefore the following rules have to be followed:

Erscheint in einem Zustandsgraphen eine Zustand sowohl in der Setz- wie in der Rücksetzbedingung eines zweiten Zustandes, dann können Schleifen auftreten, aus denen die Steuerung nicht mehr ausbrechen kann, wenn man nicht folgenden Regel beachtet:

In loops the respective states have to be rest with the following state and the set condition of the following state.

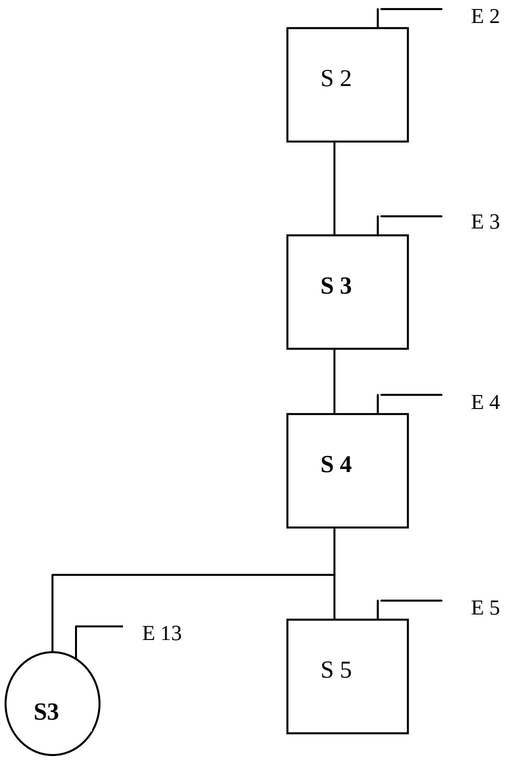


Fig. 1-14: State Graph with loops S3-S4

In Fig. 1-14 state 3 sets state 4 together with E4 but rests it as a following state. That means state 3 appears in the set and reset condition of state 4. The same is true for state 4. He acts as a set and reset -condition for state 3. But a state (signal) cannot act as set and reset input to a state (flip-flop) at the same time. Therefore we have to follow the signal wiring according to

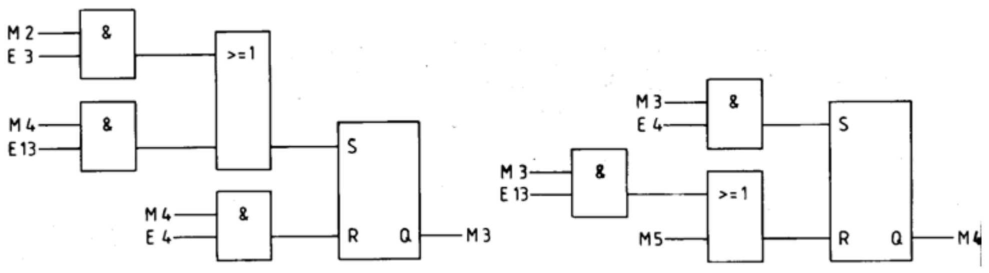


Fig. 1-15: Signal wiring in case of loops

In order to clarify the loop problematic we review an old example with the filling station.

Example: Filling station

Three reservoirs with level detectors S1, S3 and S5 (Full) and S2, S4 and S6 (Empty) can be emptied in an arbitrary order (Fig. 1-16). If any container is empty he will be filled. But the controller has to prevent that more than one container will be filled at a time. If the sensor is signaling that the container is full the filling process will be stopped. The reservoirs should be filled in the same order as they had been emptied. This is a new additional control task with respect to the old problem.

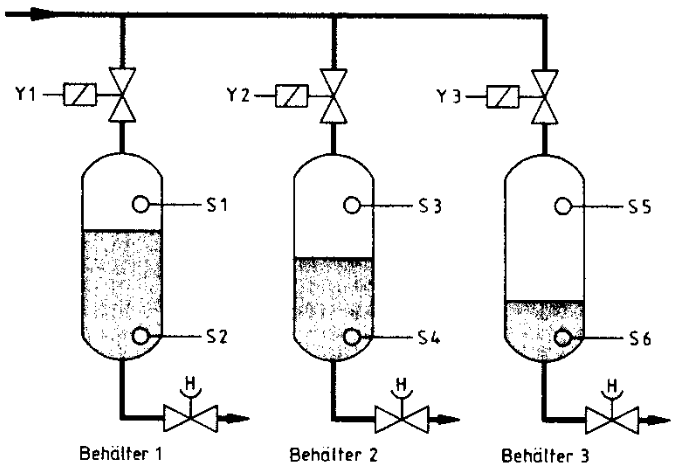


Fig. 1-16: Filling station

Für die Realisierung der Steuerung wird folgende Zuordnungstabelle zugrunde gelegt:

Input variable	Device tag		Logical assignment	
Full signal container 1	S1	I0.0	actuated	S1=1
Full signal container 2	S3	I0.2	actuated	S2=1
Full signal container 3	S5	I0.4	actuated	S3=1
Empty signal container 1	S2	I0.1	actuated	S4=1
Empty signal container 2	S4	I0.3	actuated	S5=1
Empty signal container 3	S6	I0.5	actuated	S6=1
Output variable				
Valve container 1	Y1	Q0.0	Valve open	Y1=1
Valve container 2	Y2	Q0.1	Valve open	Y2=1
Valve container 3	Y3	Q0.2	Valve open	Y3=1

The control program should be designed with SIMATIC GRAPH.

Fig. 1-17 shows state graph in a non SIMATIC manner.

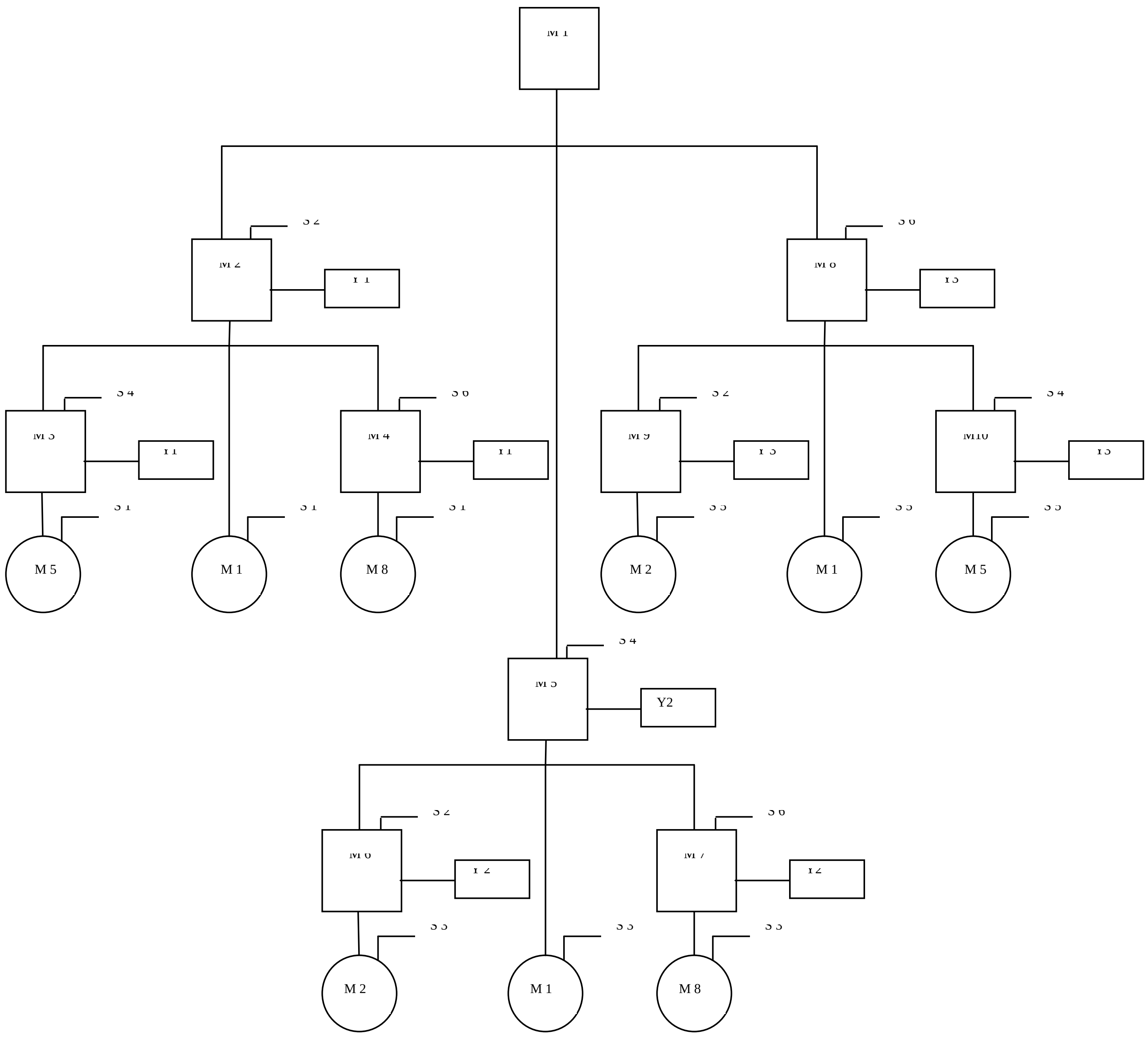


Fig. 1-17: State graph of the filling process (states are notes as M_i's)

Exercise: Implement the state graph in SIMATIC GRAPH.

From the base state (M1) the graph splits into 3 branches to implement the different filling processes for the respective reservoir.
As a following state the controller can jump to M1 so that the loops (1-2-1), (1-5-1) and (1-8-1) can occur. This has to be considered when programming the state Graph in FBD or STL!

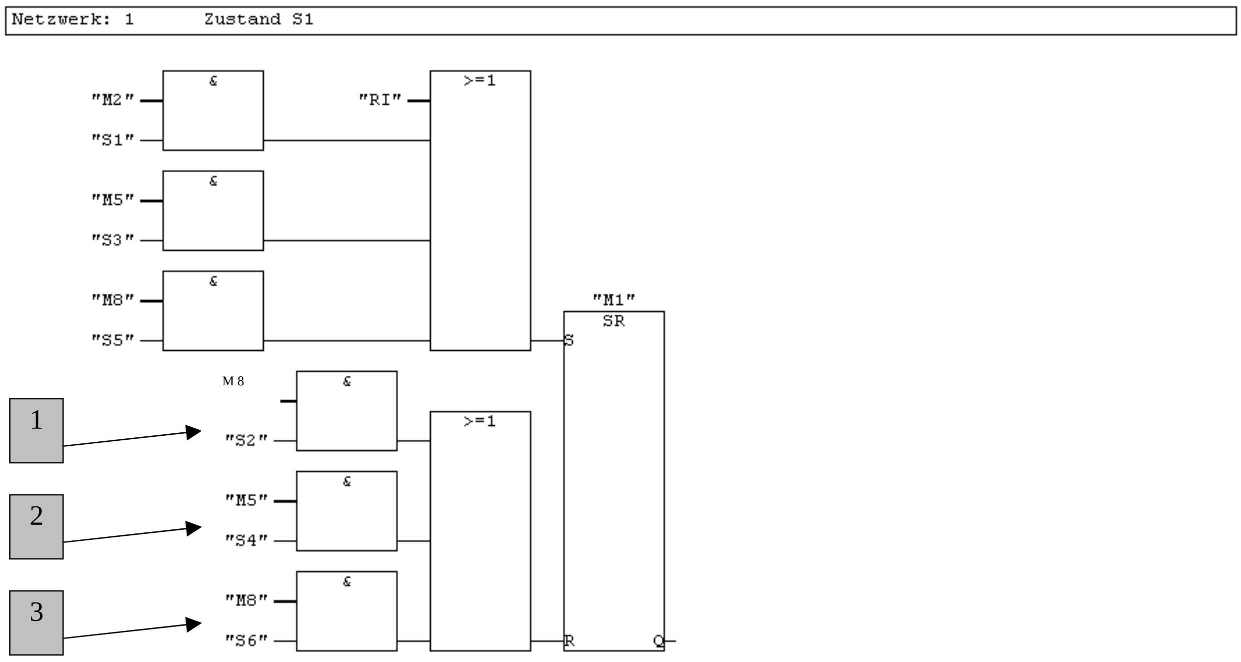
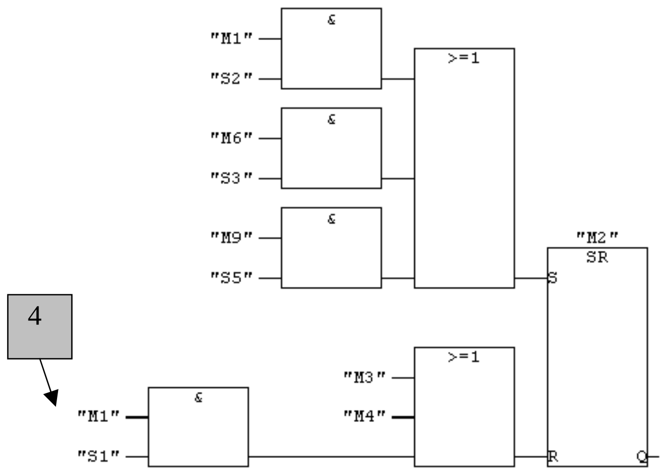
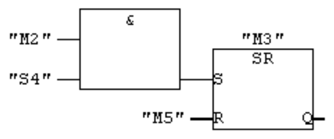


Fig. 1-18: State graph representation of the filling process in FBD, state 1

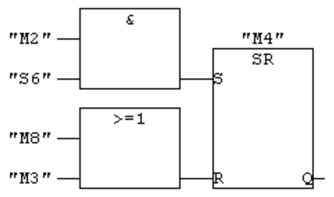
Netzwerk: 2 Zustand S2



Netzwerk: 3 Zustand S3



Netzwerk: 4 Zustand S4



Netzwerk: 5 Zustand S5

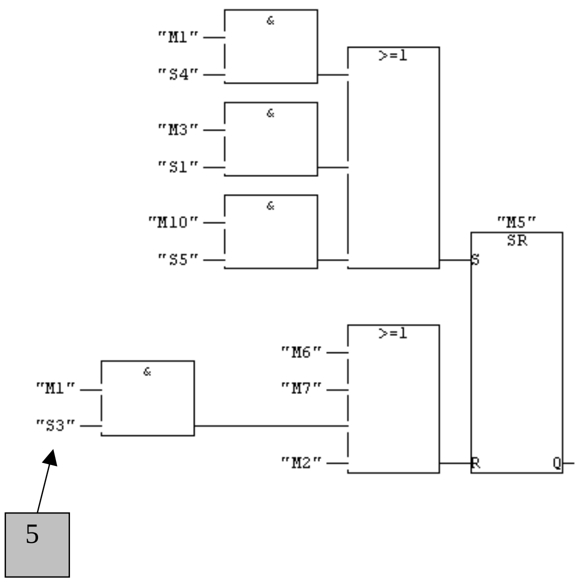


Fig. 1-19: State graph representation of the filling process in FBD, state 2 to 5

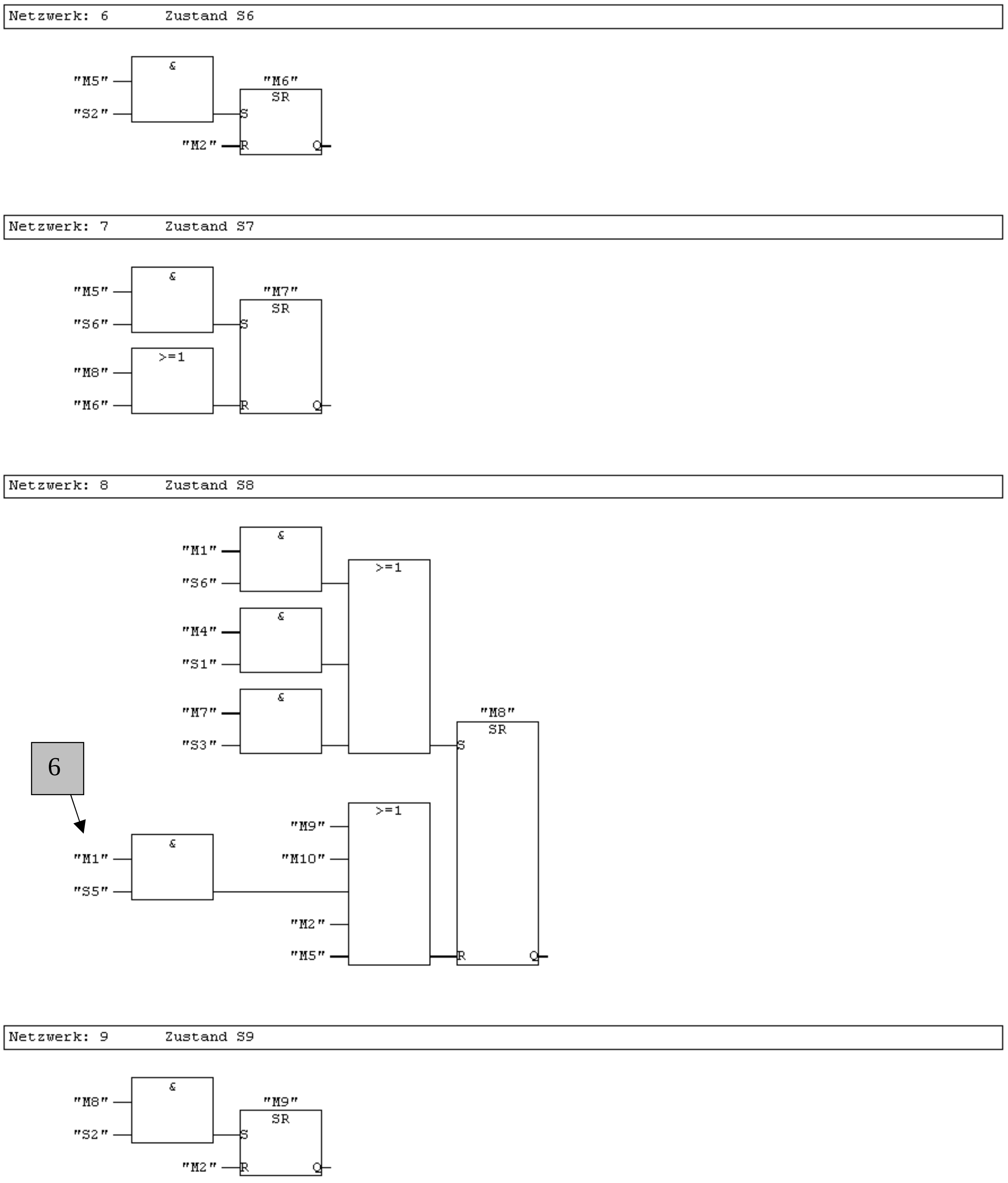


Fig. 1-20: State graph representation of the filling process in FBD, state 6 to 9

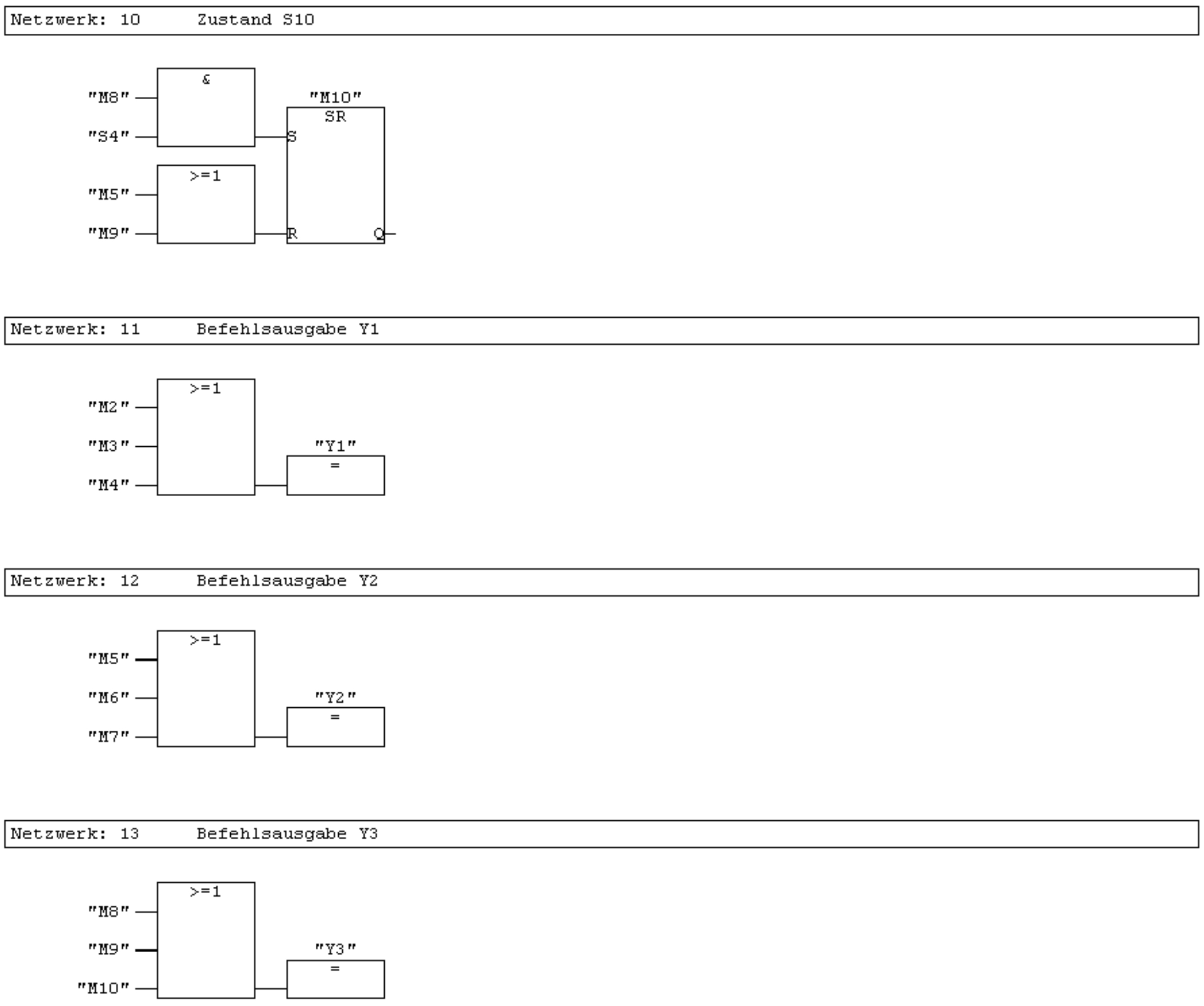


Fig. 1-21: State graph representation of the filling process in FBD, state 10 to 13

At the points 1) to 6) in the above figures we can notice that the states are reset with their following states and their set conditions. 1) and 4) implements the loops (1-2-1) and (2-1-2), 2) the loop (1-5-1) and finally the loop (1-8-1). At 5) the loop (5-1-5) is programmed and at 6) the loop (8-1-8).