

## 12. STATE BASED DESIGN

Topics:

- Describing process control using state diagrams
- Conversion of state diagrams to ladder logic
- MCR blocks

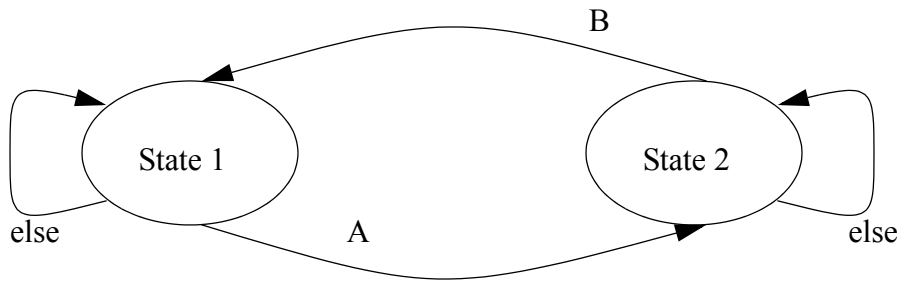
Objectives:

- Be able to construct state diagrams for a process.
- Be able to convert a state diagram to ladder logic directly.
- Be able to convert state diagrams to ladder logic using equations.

### 12.1 INTRODUCTION

A system state is a mode of operation. Consider a bank machine that will go through very carefully selected states. The general sequence of states might be idle, scan card, get secret number, select transaction type, ask for amount of cash, count cash, deliver cash/return card, then idle.

A State based system can be described with system states, and the transitions between those states. A state diagram is shown in Figure 138. The diagram has two states, *State 1* and *State 2*. If the system is in state 1 and *A* happens the system will then go into state 2, otherwise it will remain in State 1. Likewise if the system is in state 2, and *B* happens the system will return to state 1. As shown in the figure this state diagram could be used for an automatic light controller. When the power is turned on the system will go into the lights off state. If motion is detected or an on push button is pushed the system will go to the lights on state. If the system is in the lights on state and 1 hour has passed, or an off push button is pushed then the system will go to the lights off state. The else statements are omitted on the second diagram, but they are implied.



This diagram could describe the operation of energy efficient lights in a room operated by two push buttons. State 1 might be lights off and state 2 might be lights on. The arrows between the states are called transitions and will be followed when the conditions are true. In this case if we were in state 1 and A occurred we would move to state 2. The *else* loop indicate that a state will stay active if a transition are is not followed. These are so obvious they are often omitted from state diagrams.

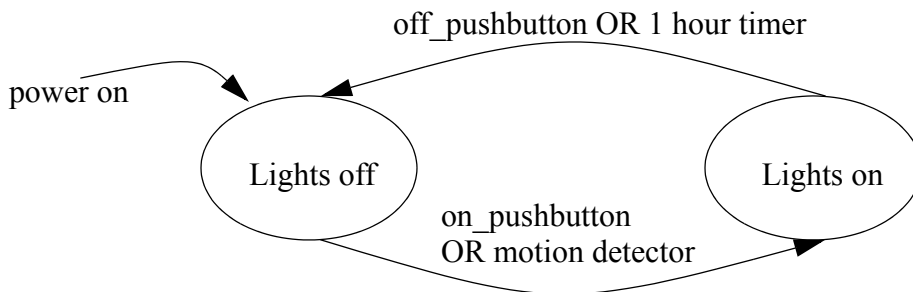


Figure 138 A State Diagram

The most essential part of creating state diagrams is identifying states. Some key questions to ask are,

1. Consider the system,
  - What does the system do normally?
  - Does the system behavior change?
  - Can something change how the system behaves?
  - Is there a sequence to actions?
2. List *modes* of operation where the system is doing one identifiable activity that will start and stop. Keep in mind that some activities may just be to wait.

Consider the design of a coffee vending machine. The first step requires the identification of vending machine states as shown in Figure 139. The main state is the idle state. There is an inserting coins state where the total can be displayed. When enough coins have been inserted the user may select their drink of choice. After this the make coffee state will be active while coffee is being brewed. If an error is detected the service needed state will be activated.

## STATES

idle - the machine has no coins and is doing nothing

inserting coins - coins have been entered and the total is displayed

user choose - enough money has been entered and the user is making coffee selection

make coffee - the selected type is being made

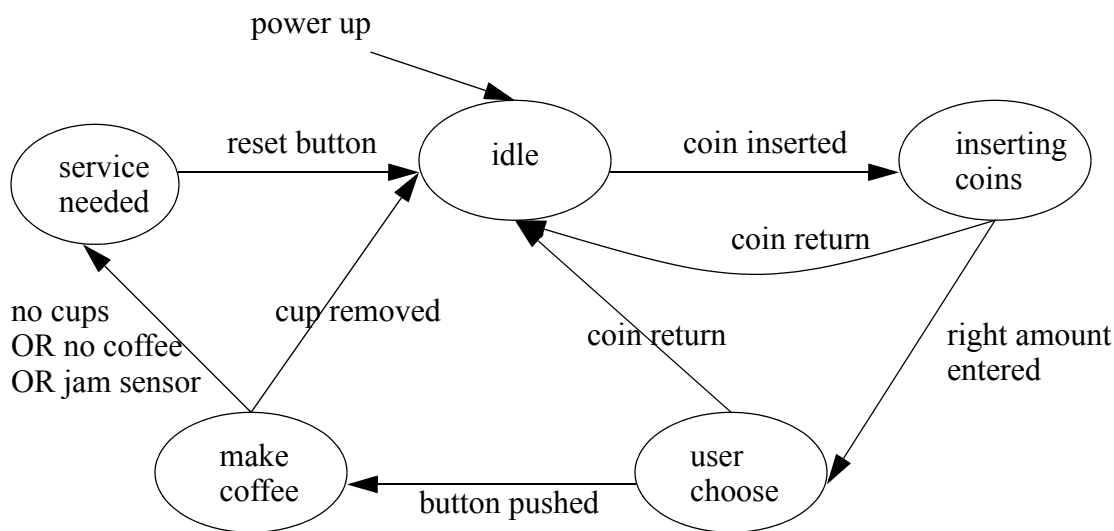
service needed - the machine is out of coffee, cups, or another error has occurred

Notes:

1. These states can be subjective, and different designers might pick others.
2. The states are highly specific to the machine.
3. The previous/next states are not part of the states.
4. There is a clean difference between states.

*Figure 139* Definition of Vending Machine States

The states are then drawn in a state diagram as shown in Figure 140. Transitions are added as needed between the states. Here we can see that when powered up the machine will start in an idle state. The transitions here are based on the inputs and sensors in the vending machine. The state diagram is quite subjective, and complex diagrams will differ from design to design. These diagrams also expose the controller behavior. Consider that if the machine needs maintenance, and it is unplugged and plugged back in, the service needed statement would not be reentered until the next customer paid for but did not receive their coffee. In a commercial design we would want to fix this oversight.



*Figure 140* State Diagram for a Coffee Machine

### 12.1.1 State Diagram Example

Consider the traffic lights in Figure 141. The normal sequences for traffic lights are a green light in one direction for a long period of time, typically 10 or more seconds. This is followed by a brief yellow light, typically 4 seconds. This is then followed by a similar light pattern in the other direction. It is understood that a green or yellow light in one direction implies a red light in the other direction. Pedestrian buttons are provided so that when pedestrians are present a cross walk light can be turned on and the duration of the green light increased.

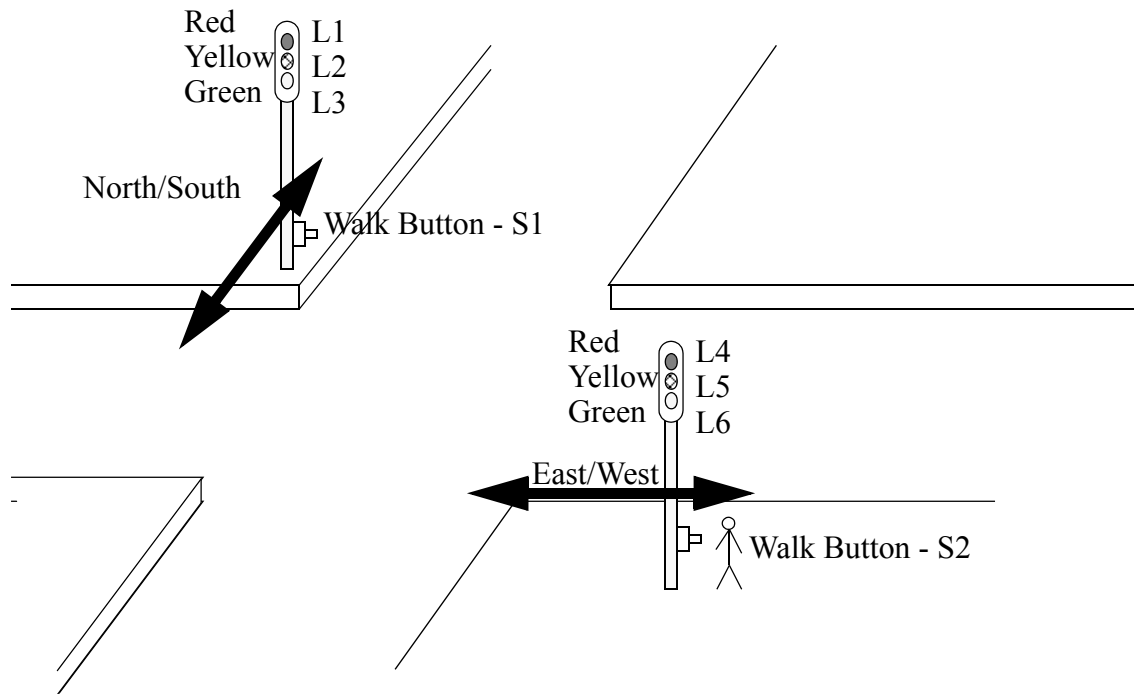
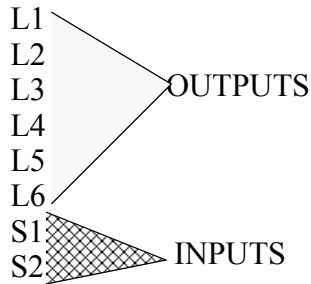


Figure 141 Traffic Lights

The first step for developing a controller is to define the inputs and outputs of the system as shown in Figure 142. First we will describe the system variables. These will vary as the system moves from state to state. Please note that some of these together can define a state (alone they are not the states). The inputs are used when defining the transitions. The outputs can be used to define the system state.

We have eight items that are ON or OFF



Note that each state will lead to a different set of outputs. The inputs are often part, or all of the transitions.

A simple diagram can be drawn to show sequences for the lights

Figure 142 Inputs and Outputs for Traffic Light Controller

Previously state diagrams were used to define the system, it is possible to use a state table as shown in Figure 143. Here the light sequences are listed in order. Each state is given a name to ease interpretation, but the corresponding output pattern is also given. The system state is defined as the bit pattern of the 6 lights. Note that there are only 4 patterns, but 6 binary bits could give as many as 64.

Step 1: Define the System States and put them (roughly) in sequence

System State		L1 L2 L3 L4 L5 L6						A binary number 0 = light off 1 = light on
State Description	#	L1	L2	L3	L4	L5	L6	
Green East/West	1	1	0	0	0	0	1	Here the four states determine how the 6 outputs are switched on/off.
Yellow East/West	2	1	0	0	0	1	0	
Green North/South	3	0	0	1	1	0	0	
Yellow North/South	4	0	1	0	1	0	0	

Figure 143 System State Table for Traffic Lights

Transitions can be added to the state table to clarify the operation, as shown in Figure 144. Here the transition from Green E/W to Yellow E/W is S1. What this means is that a cross walk button must be pushed to end the green light. This is not normal, normally the lights would use a delay. The transition from Yellow E/W to Green N/S is caused by a 4 second delay (this is normal.) The next transition is also abnormal, requiring that the cross walk button be pushed to end the Green N/S state. The last state has a 4 second delay before returning to the first state in the table. In this state table the sequence will always be the same, but the times will vary for the green lights.

Step 2: Define State Transition Triggers, and add them to the list of states

Description	#	L1	L2	L3	L4	L5	L6	transition
Green East/West	1	1	0	0	0	0	1	S1
Yellow East/West	2	1	0	0	0	1	0	delay 4sec
Green North/South	3	0	0	1	1	0	0	S2
Yellow North/South	4	0	1	0	1	0	0	

Figure 144 State Table with Transitions

A state diagram for the system is shown in Figure 145. This diagram is equivalent to the state table in Figure 144, but it can be valuable for doing visual inspection.

Step 3: Draw the State Transition Diagram

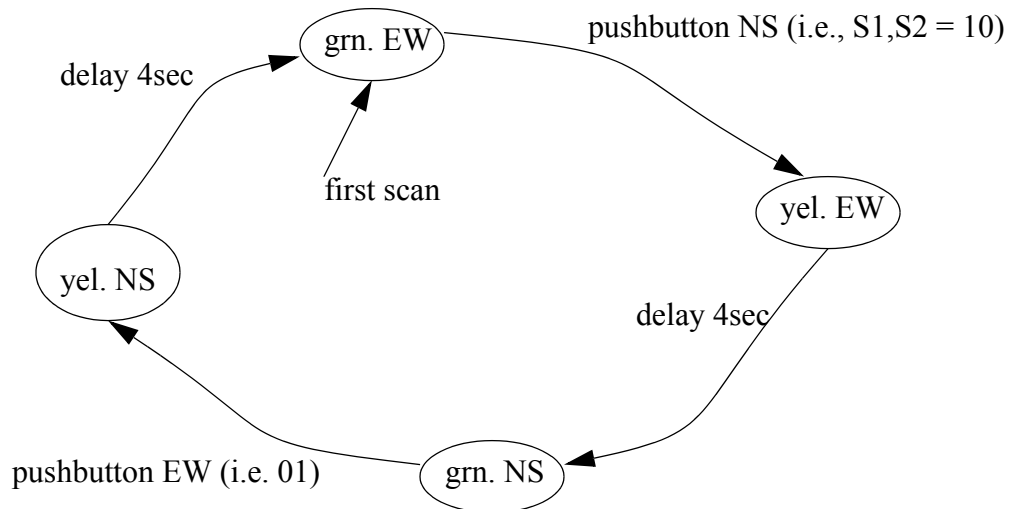


Figure 145 A Traffic Light State Diagram

## 12.1.2 Conversion to Ladder Logic

### 12.1.2.1 - Block Logic Conversion

State diagrams can be converted directly to ladder logic using block logic. This technique will produce larger programs, but it is a simple method to understand, and easy to debug. The previous traffic light example is to be implemented in ladder logic. The inputs and outputs are defined in Figure 146, assuming it will be implemented on an Allen Bradley Micrologix. *first scan* is the address of the first

scan in the PLC. The locations state\_1 to state\_4 are internal memory locations that will be used to track which states are on. The behave like outputs, but are not available for connection outside the PLC. The input and output values are determined by the PLC layout.

STATES	OUTPUTS	INPUTS
state_1 - green E/W	L1 - red N/S	S1 - cross
state_2 - yellow E/W	L2 - yellow N/S	S2 - cross
state_3 - green N/S	L3 - green N/S	S:FS - first scan
state_4 - yellow N/S	L4 - red E/W	
	L5 - yellow E/W	
	L6 - green E/W	

Figure 146 Inputs and Outputs for Traffic Light Controller

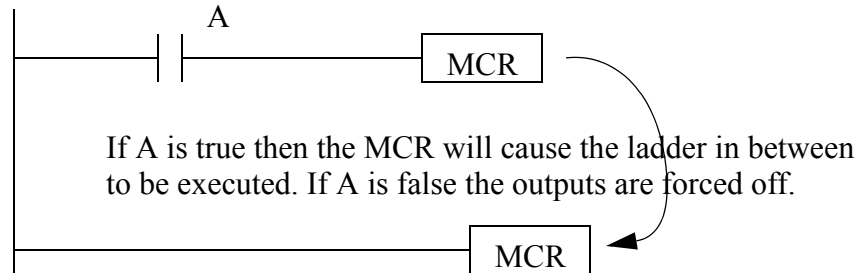
The initial ladder logic block shown in Figure 147 will initialize the states of the PLC, so that only state 1 is on. The first scan indicator *first scan* will execute the MCR block when the PLC is first turned on, and the latches will turn on the value for state\_1 and turn off the others.

#### RESET THE STATES



Figure 147 Ladder Logic to Initialize Traffic Light Controller

Note: We will use MCR instructions to implement some of the state based programs. This allows us to switch off part of the ladder logic. The one significant note to remember is that any normal outputs (not latches and timers) will be FORCED OFF. Unless this is what you want, put the normal outputs outside MCR blocks.



The next section of ladder logic only deals with outputs. For example the output *O/1* is the N/S red light, which will be on for states 1 and 2, or *B3/1* and *B3/2* respectively. Putting normal outputs outside the MCR blocks is important. If they were inside the blocks they could only be on when the MCR block was active, otherwise they would be forced off. Note: Many beginners will make the careless mistake of repeating outputs in this section of the program.

#### TURN ON LIGHTS AS REQUIRED

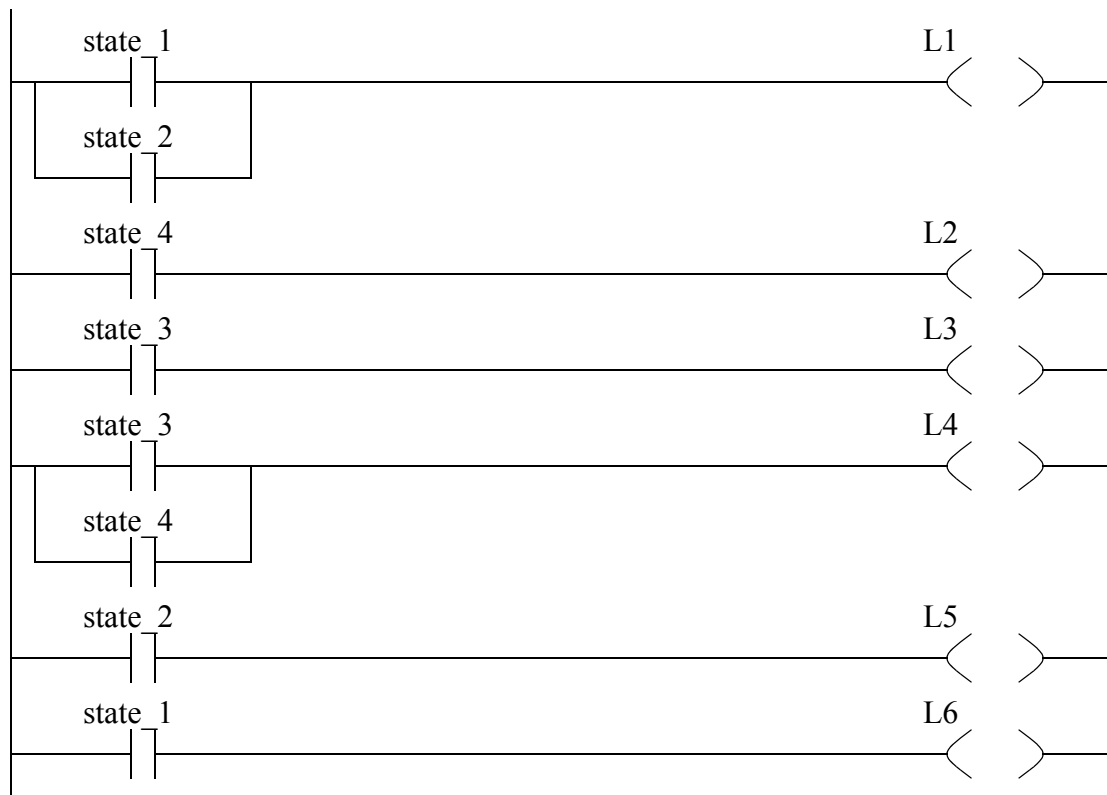


Figure 148 General Output Control Logic

The first state is implemented in Figure 147. If state\_1 is active this will be active. The transition is S1 which will end state\_1 and start state\_2.

#### FIRST STATE WAIT FOR TRANSITIONS



Figure 149 Ladder Logic for First State

The second state is more complex because it involves a time delay, as shown in Figure 150. When the state is active the TON timer will be timing. When the timer is done state 2 will be unlatched, and state 3 will be latched on. The timer is nonretentive, so if state\_2 is off the MCR block will force all of the outputs off, including the timer, causing it to reset.

## SECOND STATE WAIT FOR TRANSITIONS

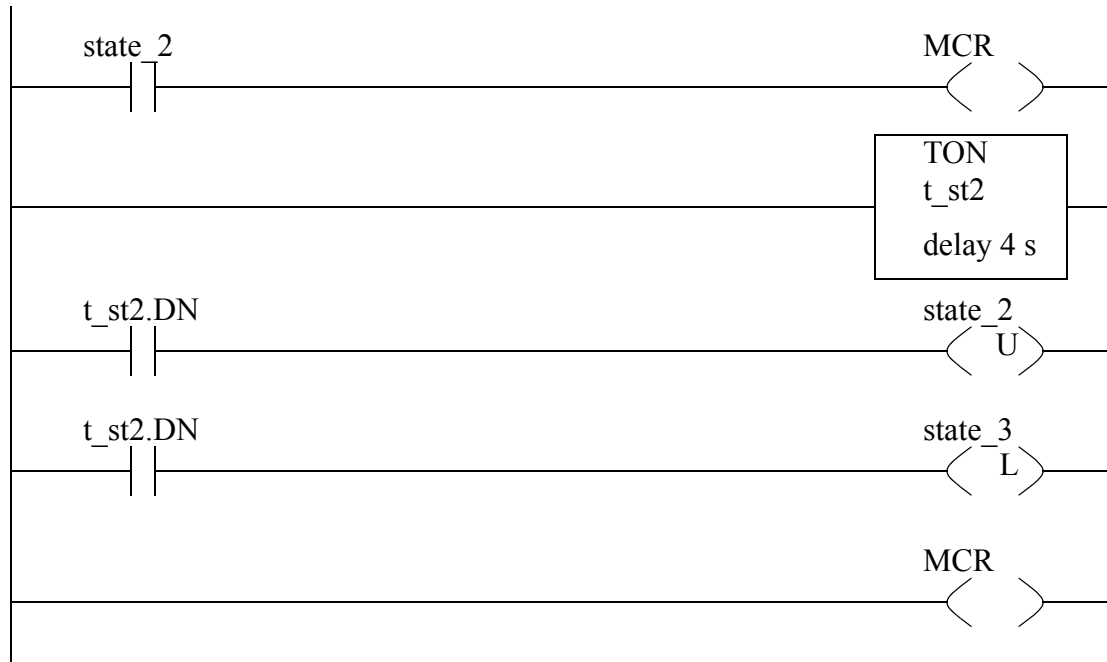


Figure 150 Ladder Logic for Second State

The third and fourth states are shown in Figure 151 and Figure 152. Their layout is very similar to that of the first two states.

## THIRD STATE WAIT FOR TRANSITIONS

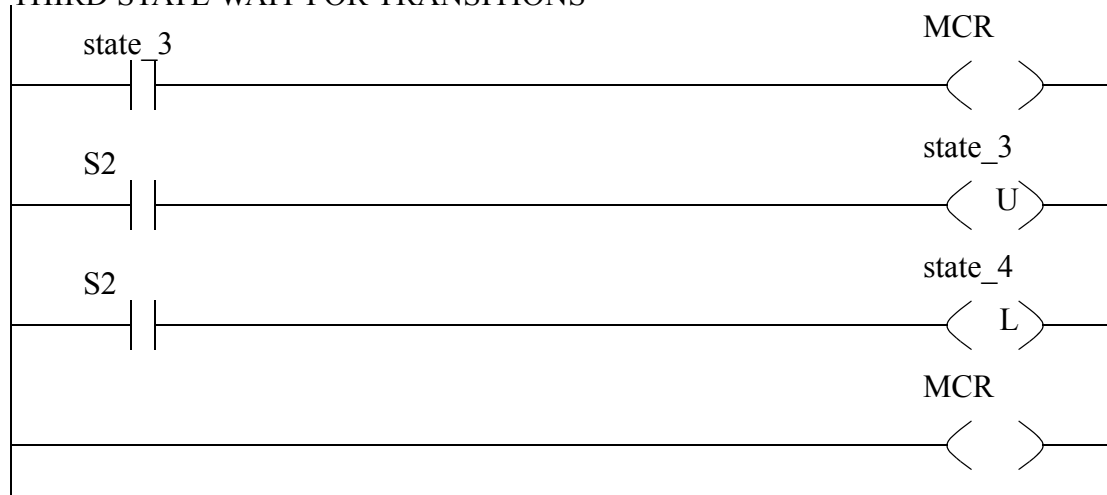


Figure 151 Ladder Logic for State Three

## FOURTH STATE WAIT FOR TRANSITIONS

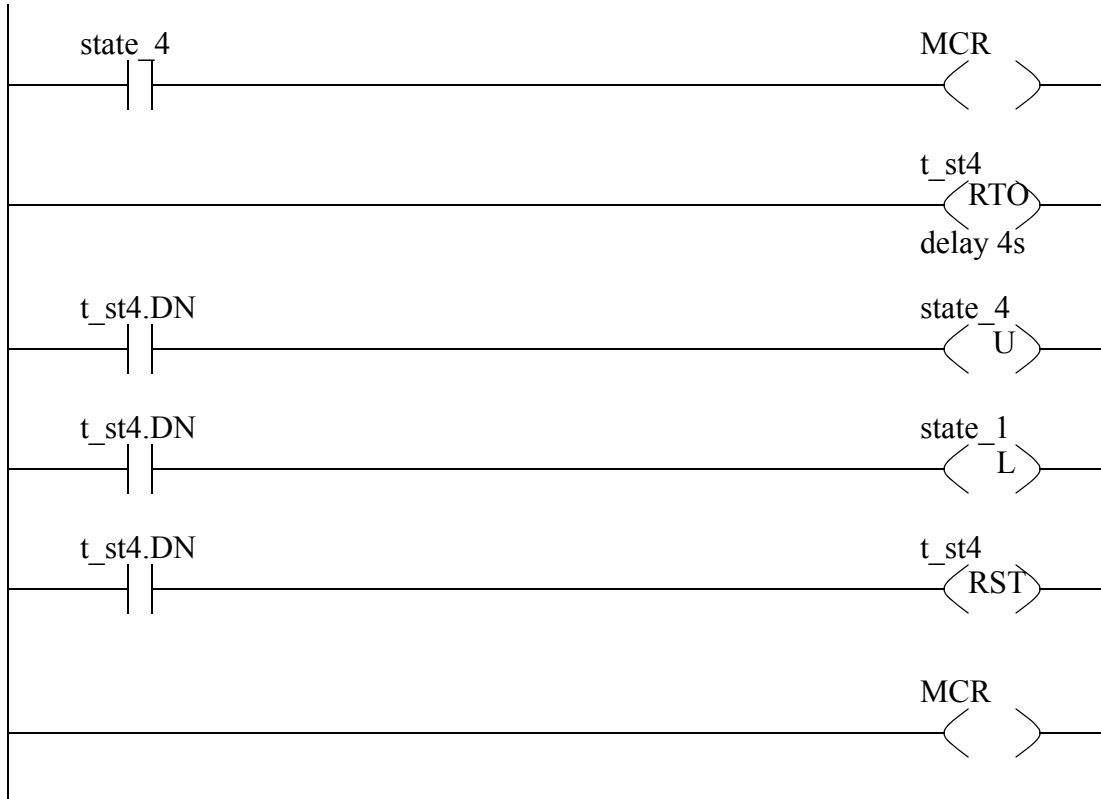


Figure 152 Ladder Logic for State Four

The previous example only had one path through the state tables, so there was never a choice between states. The state diagram in Figure 153 could potentially have problems if two transitions occur simultaneously. For example if state *STB* is active and *A* and *C* occur simultaneously, the system could go to either *STA* or *STC* (or both in a poorly written program.) To resolve this problem we should choose one of the two transitions as having a higher priority, meaning that it should be chosen over the other transition. This decision will normally be clear, but if not an arbitrary decision is still needed.

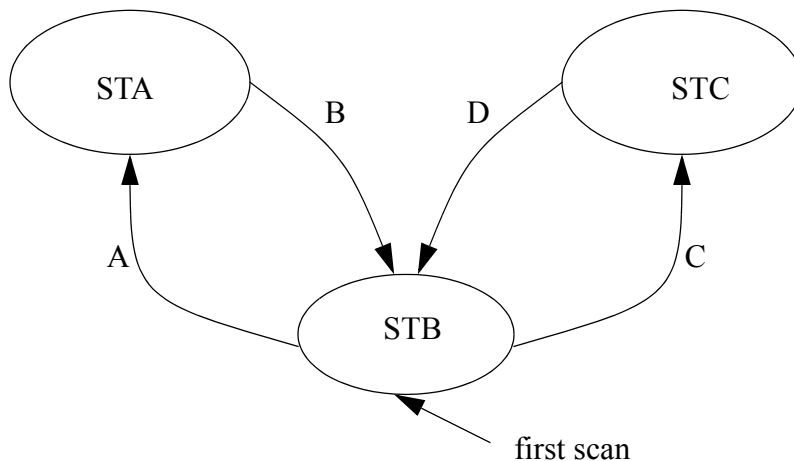


Figure 153 A State Diagram with Priority Problems

The state diagram in Figure 153 is implemented with ladder logic in Figure 154 and Figure 155. The implementation is the same as described before, but for state *STB* additional ladder logic is added to disable transition *A* if transition *C* is active, therefore giving priority to *C*.

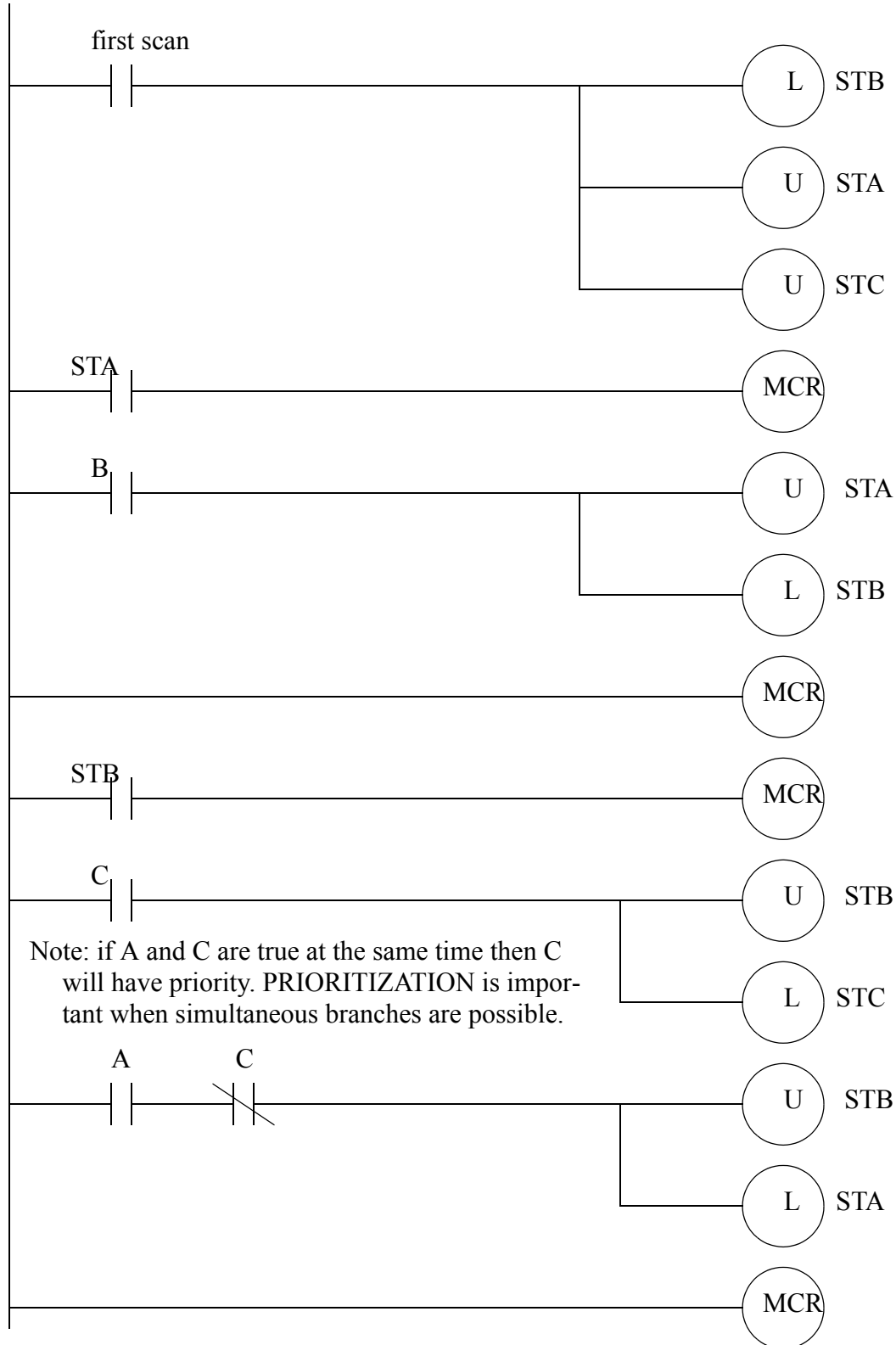


Figure 154 State Diagram for Prioritization Problem

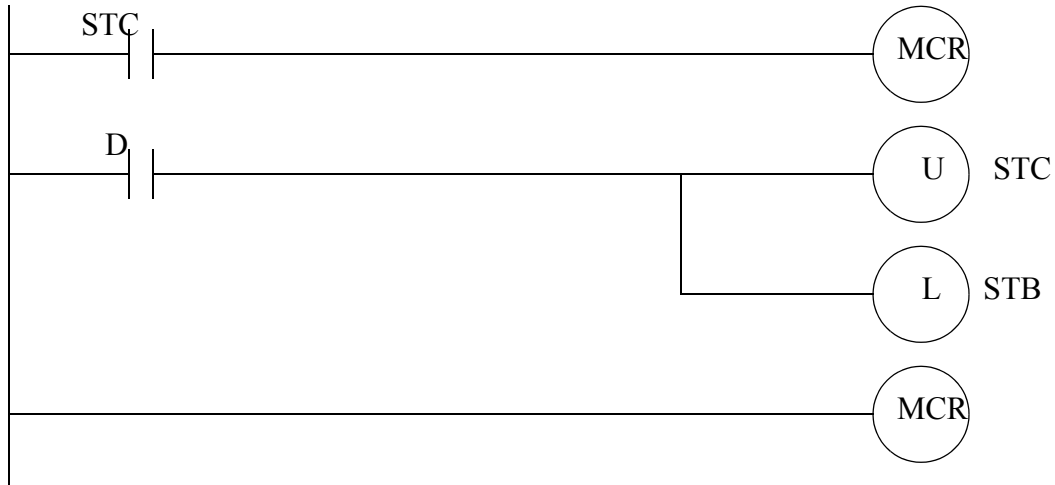


Figure 155 State Diagram for Prioritization Problem

The Block Logic technique described does not require any special knowledge and the programs can be written directly from the state diagram. The final programs can be easily modified, and finding problems is easier. But, these programs are much larger and less efficient.

### 12.1.2.2 - State Equations

State diagrams can be converted to Boolean equations and then to Ladder Logic. The first technique that will be described is state equations. These equations contain three main parts, as shown below in Figure 156. To describe them simply - a state will be on if it is already on, or if it has been turned on by a transition from another state, but it will be turned off if there was a transition to another state. An equation is required for each state in the state diagram.

Informally,

State X = (State X + just arrived from another state) and has not left for another state

Formally,

$$STATE_i = \left( STATE_i + \sum_{j=1}^n (T_{j,i} \bullet STATE_j) \right) \bullet \prod_{k=1}^m \overline{(T_{i,k} \bullet STATE_i)}$$

where,  $STATE_i$  = A variable that will reflect if state i is on

$n$  = the number of transitions to state i

$m$  = the number of transitions out of state i

$T_{j,i}$  = The logical condition of a transition from state j to i

$T_{i,k}$  = The logical condition of a transition out of state i to k

*Figure 156* State Equations

The state equation method can be applied to the traffic light example in Figure 145. The first step in the process is to define variable names (or PLC memory locations) to keep track of which states are on or off. Next, the state diagram is examined, one state at a time. The first equation is for ST1, or *state 1 - green NS*. The start of the equation can be read as ST1 will be on if it is on, or if ST4 is on, and it has been on for 4s, or if it is the first scan of the PLC. The end of the equation can be read as ST1 will be turned off if it is on, but S1 has been pushed and S2 is off. As discussed before, the first half of the equation will turn the state on, but the second half will turn it off. The first scan is also used to turn on ST1 when the PLC starts. It is put outside the terms to force ST1 on, even if the exit conditions are true.

Defined state variables:

$ST1$  = state 1 - green NS

$ST2$  = state 2 - yellow NS

$ST3$  = state 3 - green EW

$ST4$  = state 4 - yellow EW

The state entrance and exit condition equations:

$$ST1 = (ST1 + ST4 \cdot TON_2(ST4, 4s)) \cdot \overline{ST1} \cdot S1 \cdot \overline{S2} + FS$$

$$ST2 = (ST2 + ST1 \cdot S1 \cdot \overline{S2}) \cdot \overline{ST2} \cdot TON_1(ST2, 4s)$$

$$ST3 = (ST3 + ST2 \cdot TON_1(ST2, 4s)) \cdot \overline{ST3} \cdot \overline{S1} \cdot S2$$

$$ST4 = (ST4 + ST3 \cdot \overline{S1} \cdot S2) \cdot \overline{ST4} \cdot TON_2(ST4, 4s)$$

Note: Timers are represented in these equations in the form  $TON_i(A, delay)$ .  $TON$  indicates that it is an on-delay timer,  $A$  is the input to the timer, and  $delay$  is the timer delay value. The subscript  $i$  is used to differentiate timers.

Figure 157 State Equations for the Traffic Light Example

The equations in Figure 157 cannot be implemented in ladder logic because of the NOT over the last terms. The equations are simplified in Figure 158 so that all NOT operators are only over a single variable.

Now, simplify these for implementation in ladder logic.

$$ST1 = (ST1 + ST4 \cdot TON_2(ST4, 4)) \cdot (\overline{ST1} + \overline{S1} + S2) + FS$$

$$ST2 = (ST2 + ST1 \cdot S1 \cdot \overline{S2}) \cdot (\overline{ST2} + \overline{TON_1(ST2, 4)})$$

$$ST3 = (ST3 + ST2 \cdot TON_1(ST2, 4)) \cdot (\overline{ST3} + S1 + \overline{S2})$$

$$ST4 = (ST4 + ST3 \cdot \overline{S1} \cdot S2) \cdot (\overline{ST4} + \overline{TON_2(ST4, 4)})$$

Figure 158 Simplified Boolean Equations

These equations are then converted to the ladder logic shown in Figure 159 and Figure 160. At the top of the program the two timers are defined. (Note: it is tempting to combine the timers, but it is better to keep them separate.) Next, the Boolean state equations are implemented in ladder logic. After this we use the states to turn specific lights on.

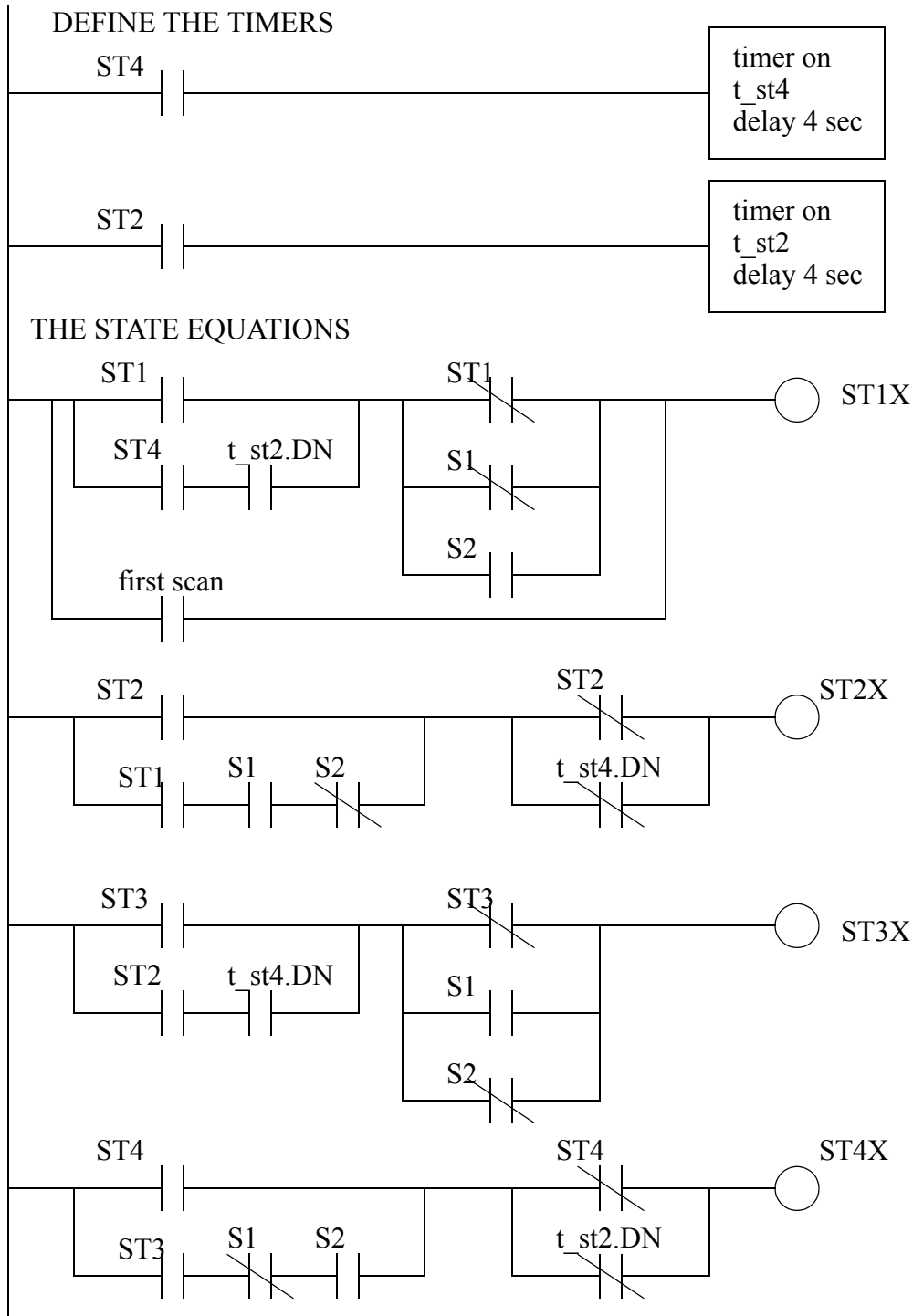


Figure 159 Ladder Logic for the State Equations

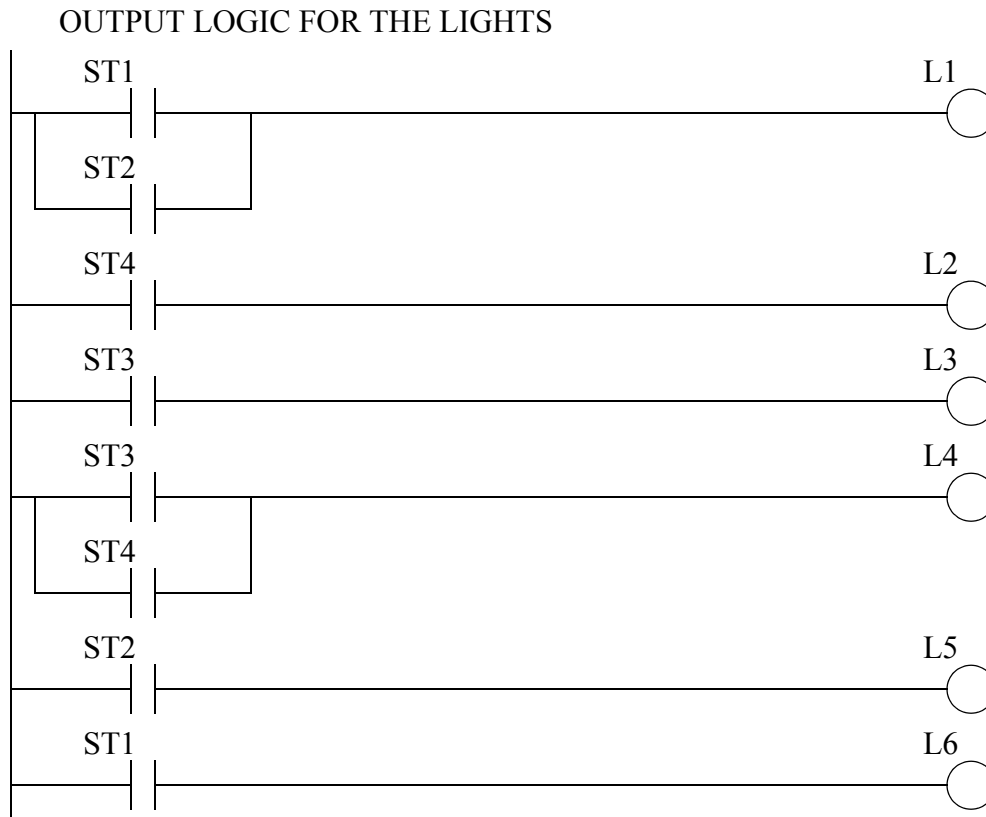


Figure 160 Ladder Logic for the State Equations

This method will provide the most compact code of all techniques, but there are potential problems. Consider the example in Figure 160. If push button *S1* has been pushed the line for ST1 should turn off, and the line for ST2 should turn on. But, the line for ST2 depends upon the value for *ST1* that has just been turned off. This will cause a problem if the value of ST1 goes off immediately after the line of ladder logic has been scanned. In effect the PLC will get *lost* and none of the states will be on. This problem arises because the equations are normally calculated in parallel, and then all values are updated simultaneously. To overcome this problem the ladder logic could be modified to the form shown in Figure 161. Here some temporary variables are used to hold the new state values. After all the equations are solved the states are updated to their new values.

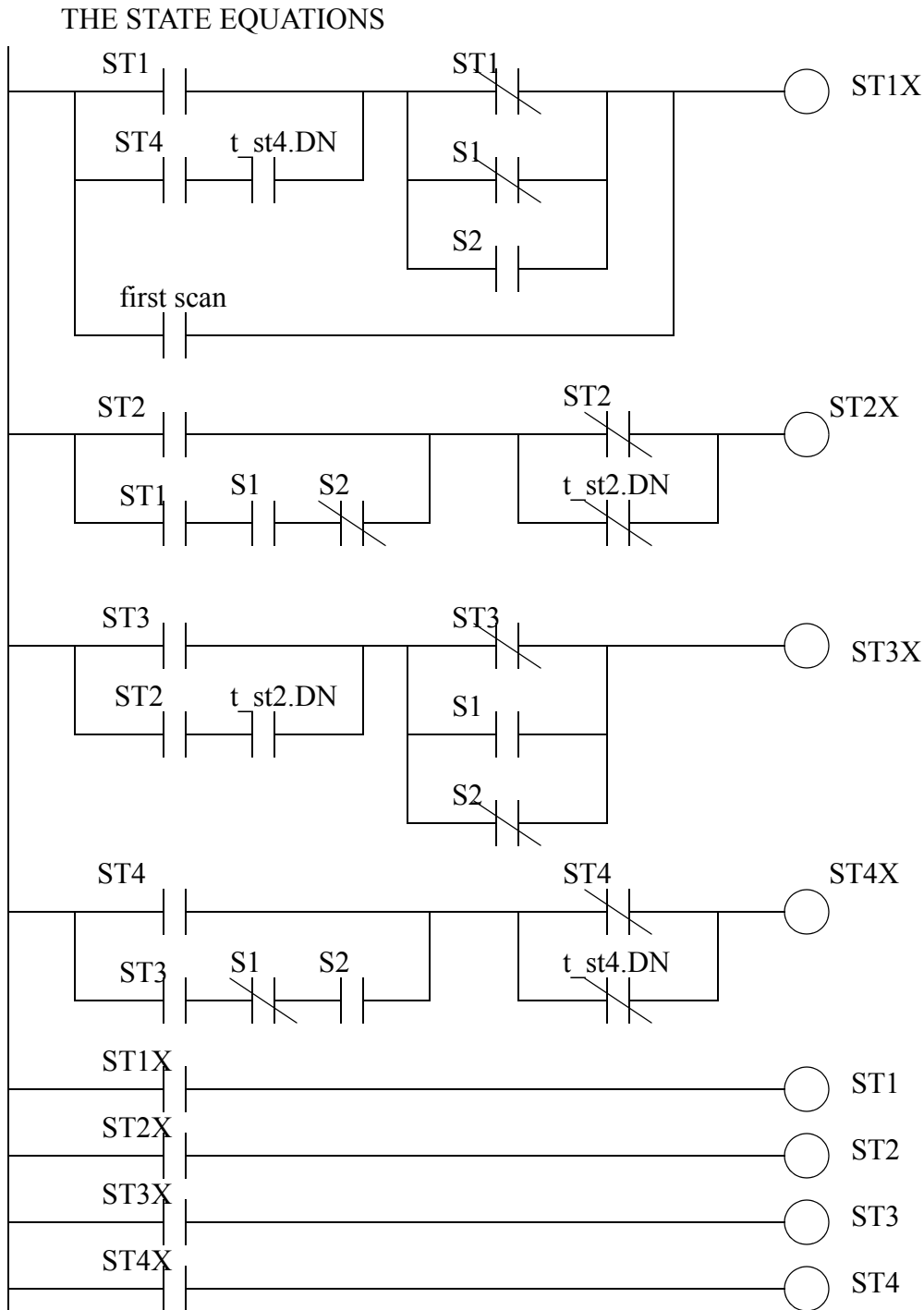
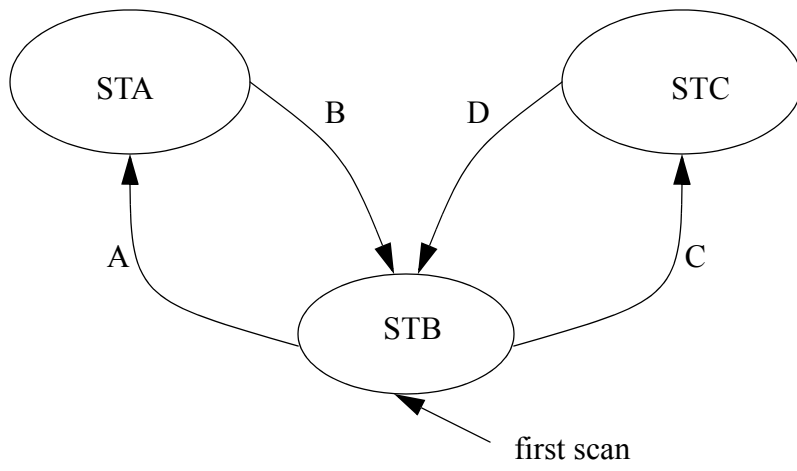


Figure 161 Delayed State Updating

When multiple transitions out of a state exist we must take care to add priorities. Each of the alternate transitions out of a state should be given a priority, from highest to lowest. The state equations can then be written to suppress transitions of lower priority when one or more occur simultaneously. The state diagram in Figure 162 has two transitions *A* and *C* that could occur simultaneously. The equations have been written to give *A* a higher priority. When *A* occurs, it will block *C* in the equation for *STC*. These equations have been converted to ladder logic in Figure 163.



$$STA = (STA + STB \cdot A) \cdot \overline{STA \cdot B}$$

$$STB = (STB + STA \cdot B + STC \cdot D) \cdot \overline{STB \cdot A} \cdot \overline{STB \cdot C} + FS$$

$$STC = (STC + STB \cdot C \cdot \bar{A}) \cdot \overline{STC \cdot D}$$

Figure 162 State Equations with Prioritization

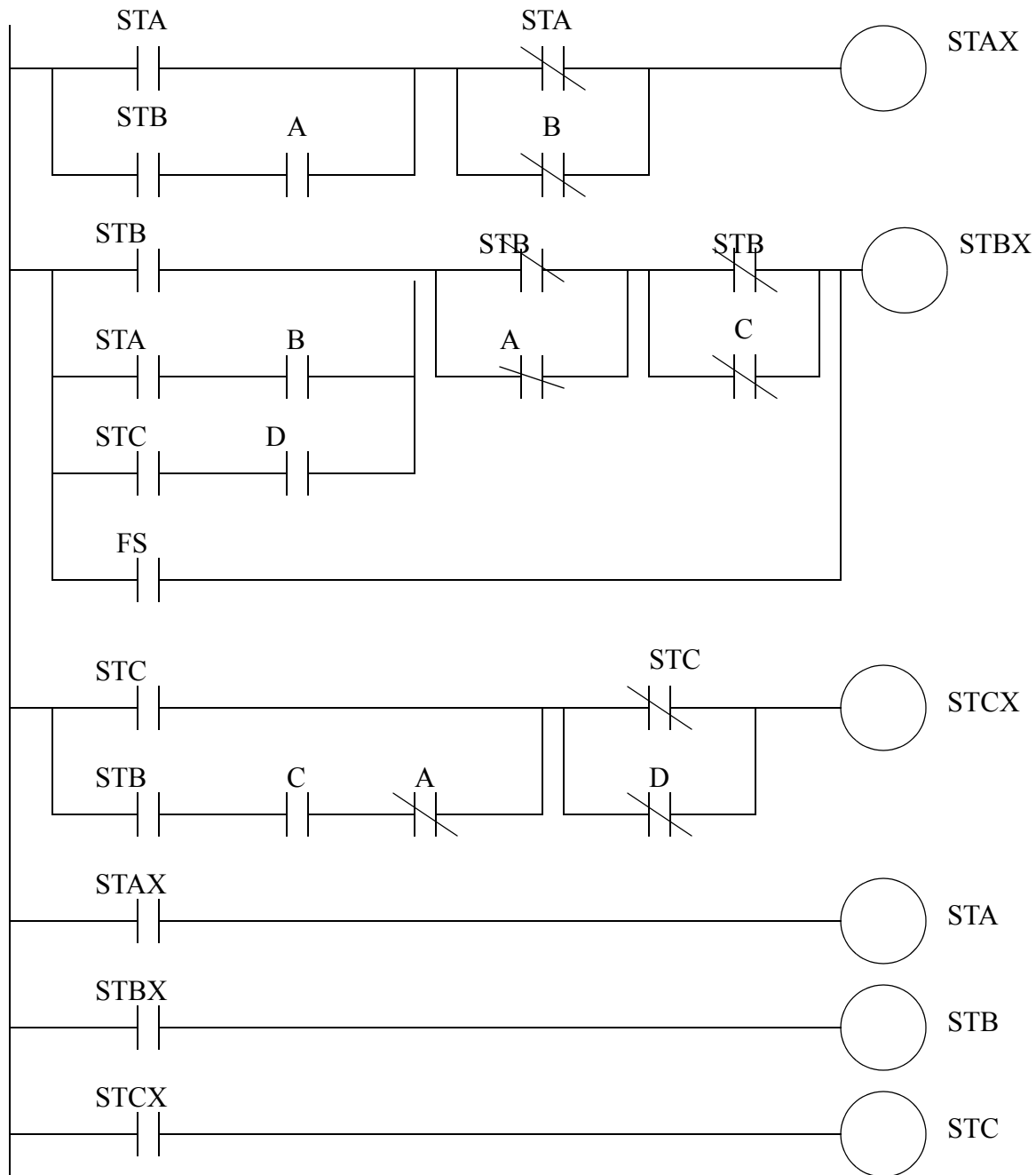


Figure 163 Ladder Logic with Prioritization

### 12.1.2.3 - State-Transition Equations

A state diagram may be converted to equations by writing an equation for each state and each transition. A sample set of equations is seen in Figure 164 for the traffic light example of Figure 145. Each state and transition needs to be assigned a unique variable name. (Note: It is a good idea to note these on the diagram) These are then used to write the equations for the diagram. The transition equations are written by looking at the each state, and then determining which transitions will end that state.

For example, if  $ST1$  is true, and crosswalk button  $S1$  is pushed, and  $S2$  is not, then transition  $T1$  will be true. The state equations are similar to the state equations in the previous State Equation method, except they now only refer to the transitions. Recall, the basic form of these equations is that the state will be on if it is already on, or it has been turned on by a transition. The state will be turned off if an exiting transition occurs. In this example the first scan was given it's own transition, but it could have also been put into the equation for  $T4$ .

defined state and transition variables:

$ST1$  = state 1 - green NS

$T1$  = transition from  $ST1$  to  $ST2$

$ST2$  = state 2 - yellow NS

$T2$  = transition from  $ST2$  to  $ST3$

$ST3$  = state 3 - green EW

$T3$  = transition from  $ST3$  to  $ST4$

$ST4$  = state 4 - yellow EW

$T4$  = transition from  $ST4$  to  $ST1$

$T5$  = transition to  $ST1$  for first scan

state and transition equations:

$$T4 = ST4 \cdot TON_2(ST4, 4)$$

$$ST1 = (ST1 + T4 + T5) \cdot \overline{T1}$$

$$T1 = ST1 \cdot S1 \cdot \overline{S2}$$

$$ST2 = (ST2 + T1) \cdot \overline{T2}$$

$$T2 = ST2 \cdot TON_1(ST2, 4)$$

$$ST3 = (ST3 + T2) \cdot \overline{T3}$$

$$T3 = ST3 \cdot \overline{S1} \cdot S2$$

$$ST4 = (ST4 + T3) \cdot \overline{T4}$$

$$T5 = FS$$

*Figure 164* State-Transition Equations

These equations can be converted directly to the ladder logic in Figure 165, Figure 166 and Figure 167. It is very important that the transition equations all occur before the state equations. By updating the transition equations first and then updating the state equations the problem of state variable values changing is negated - recall this problem was discussed in the State Equations section.

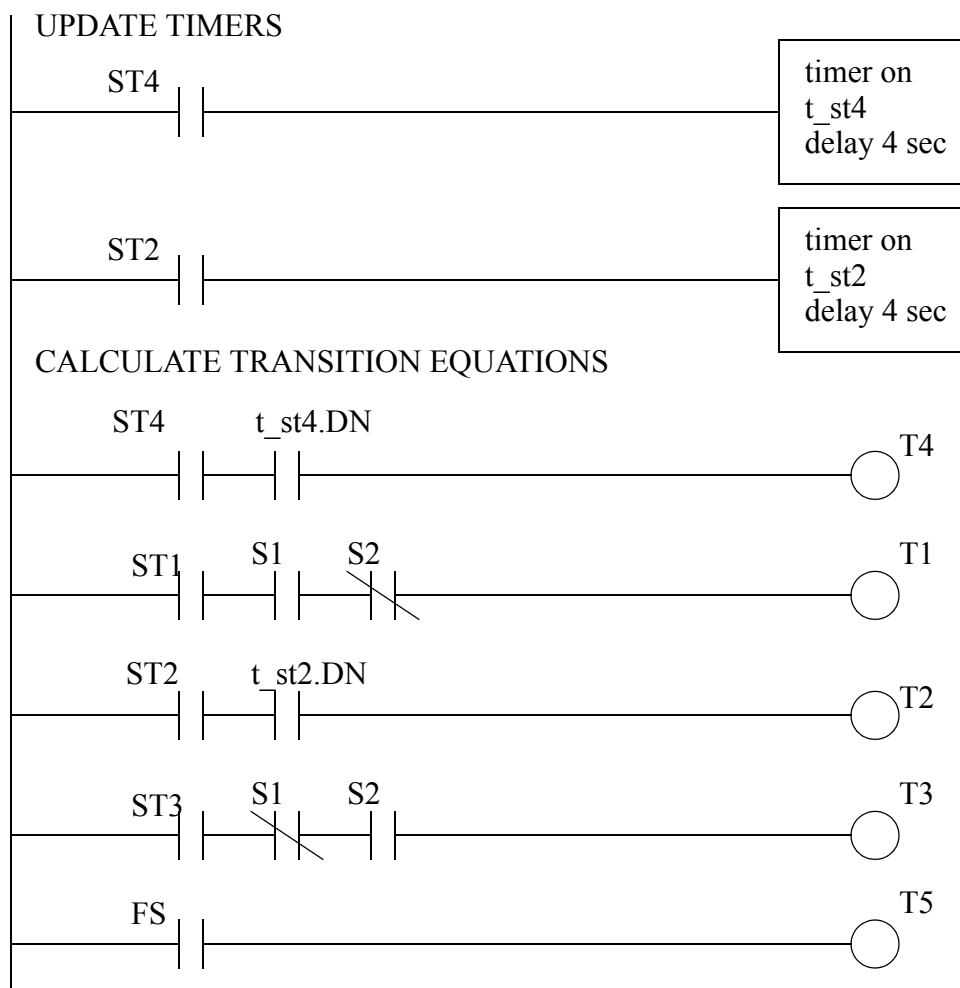


Figure 165 Ladder Logic for the State-Transition Equations

## CALCULATE STATE EQUATIONS

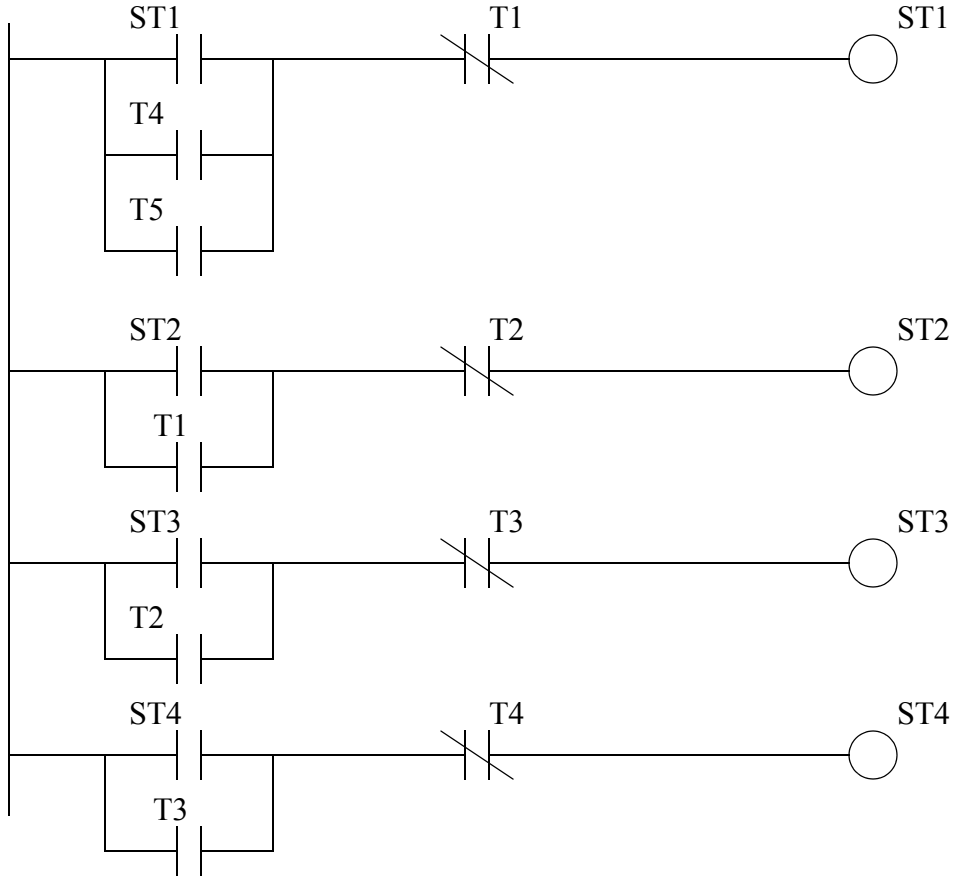


Figure 166 Ladder Logic for the State-Transition Equations

## UPDATE OUTPUTS

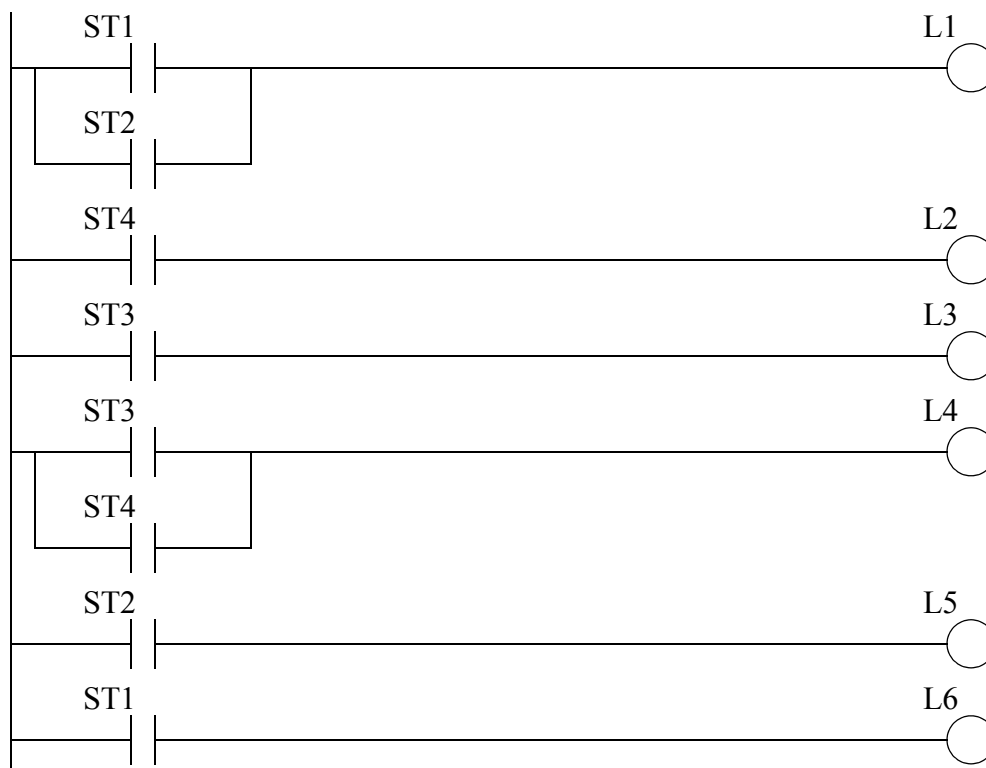


Figure 167 Ladder Logic for the State-Transition Equations

The problem of prioritization also occurs with the State-Transition equations. Equations were written for the State Diagram in Figure 168. The problem will occur if transitions  $A$  and  $C$  occur simultaneously. In the example transition  $T2$  is given a higher priority, and if it is true, then the transition  $T3$  will be suppressed when calculating  $STC$ . In this example the transitions have been considered in the state update equations, but they can also be used in the transition equations.

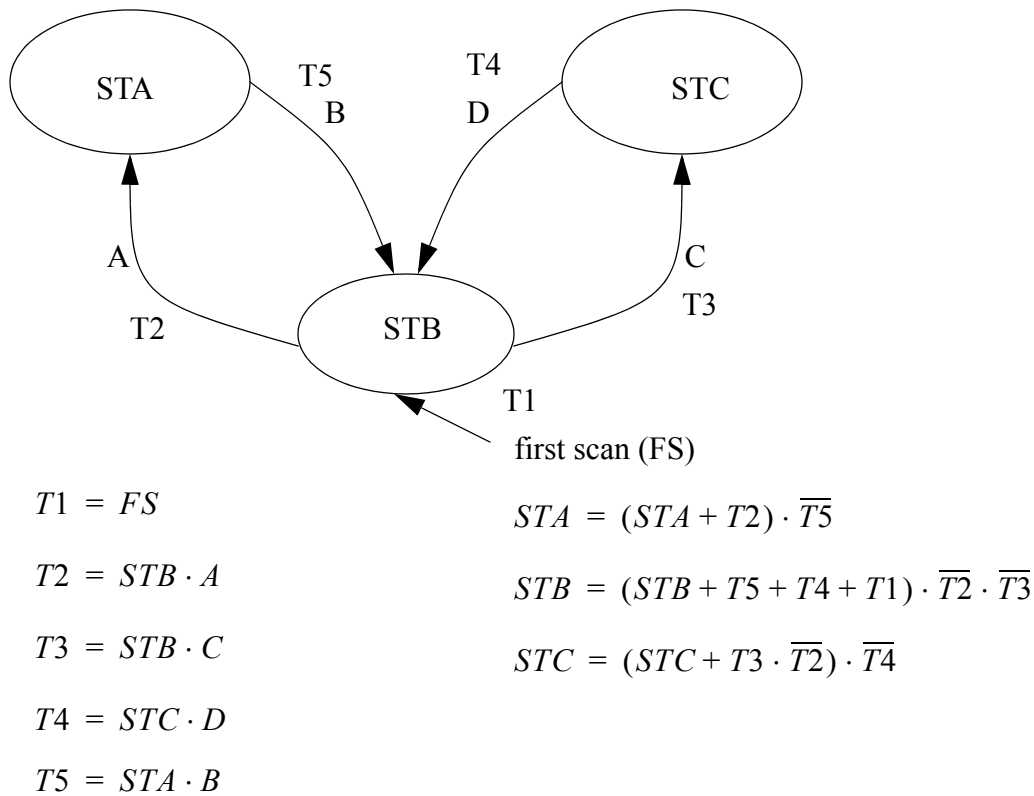


Figure 168 Prioritization for State Transition Equations

## 12.2 SUMMARY

- State diagrams are suited to processes with a single flow of execution.
- State diagrams are suited to problems that has clearly defines modes of execution.
- Controller diagrams can be converted to ladder logic using MCR blocks
- State diagrams can also be converted to ladder logic using equations
- The sequence of operations is important when converting state diagrams to ladder logic.

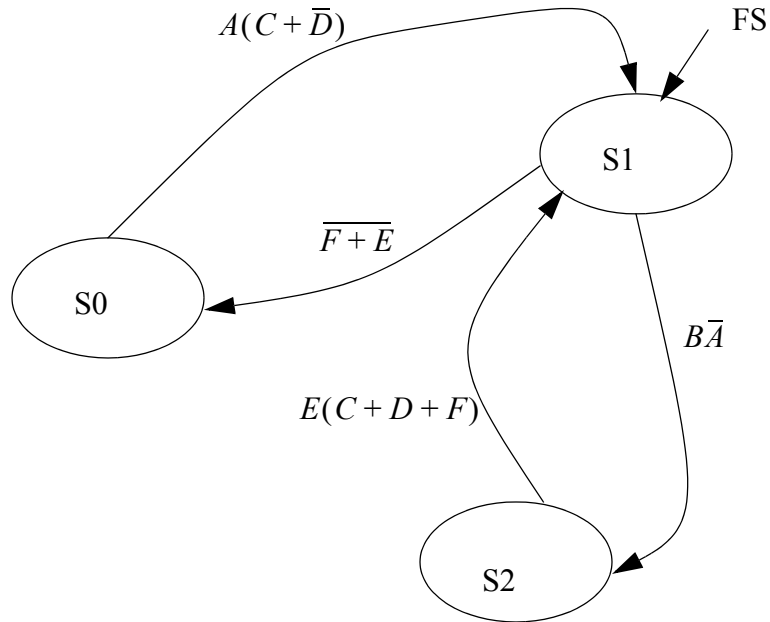
## 12.3 PRACTICE PROBLEMS

1. Draw a state diagram for a microwave oven.

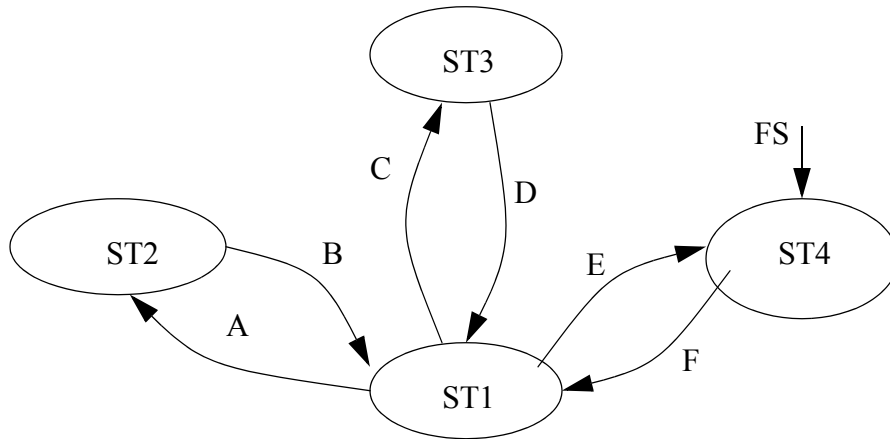
2. Convert the following state diagram to equations.

Inputs      Outputs  
 A            P  
 B            Q  
 C            R  
 D  
 E  
 F

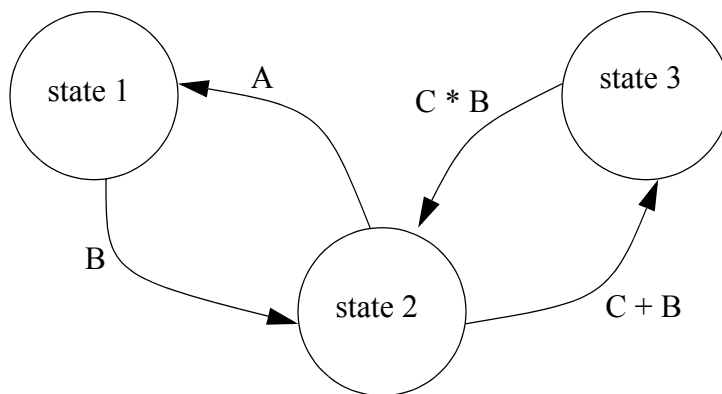
state	P	Q	R
S0	0	1	1
S1	1	0	1
S2	1	1	0



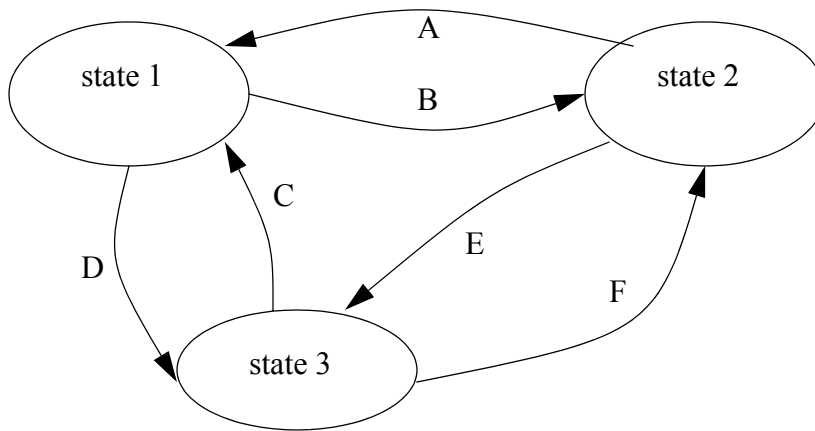
3. Implement the following state diagram with equations.



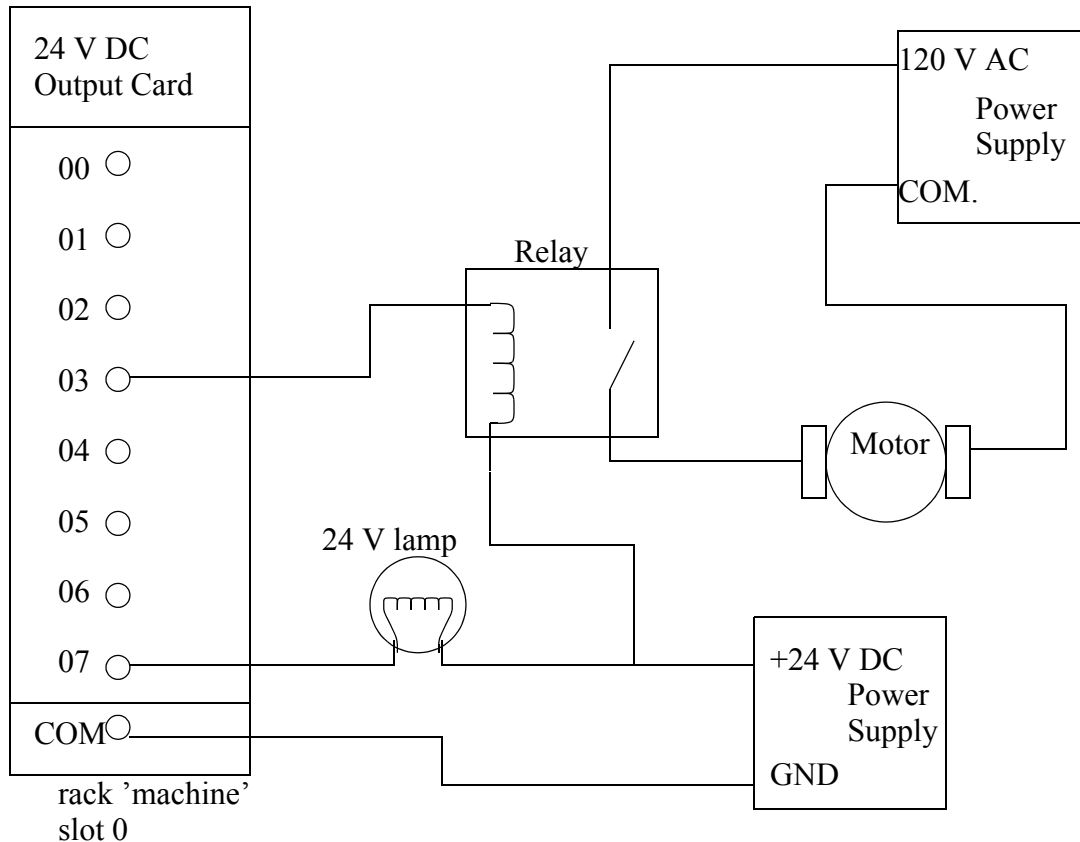
4. Given the following state diagram, use equations to implement ladder logic.

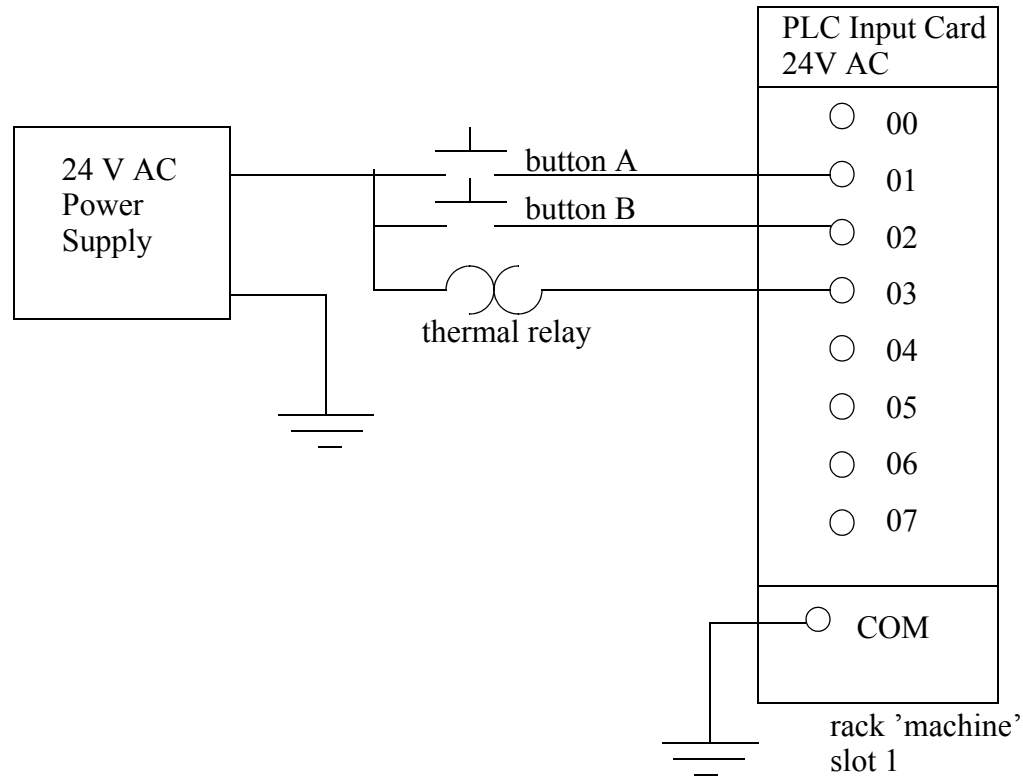


5. Convert the following state diagram to logic using equations.



6. You have been asked to program a PLC that is controlling a handicapped access door opener. The client has provided the electrical wiring diagram below to show how the PLC inputs and outputs have been wired. Button A is located inside and button B is located outside. When either button is pushed the motor will be turned on to open the door. The motor is to be kept on for a total of 15 seconds to allow the person to enter. After the motor is turned off the door will fall closed. In the event that somebody gets caught in the door the thermal relay will go off, and the motor should be turned off. After 20,000 cycles the door should stop working and the light should go on to indicate that maintenance is required.

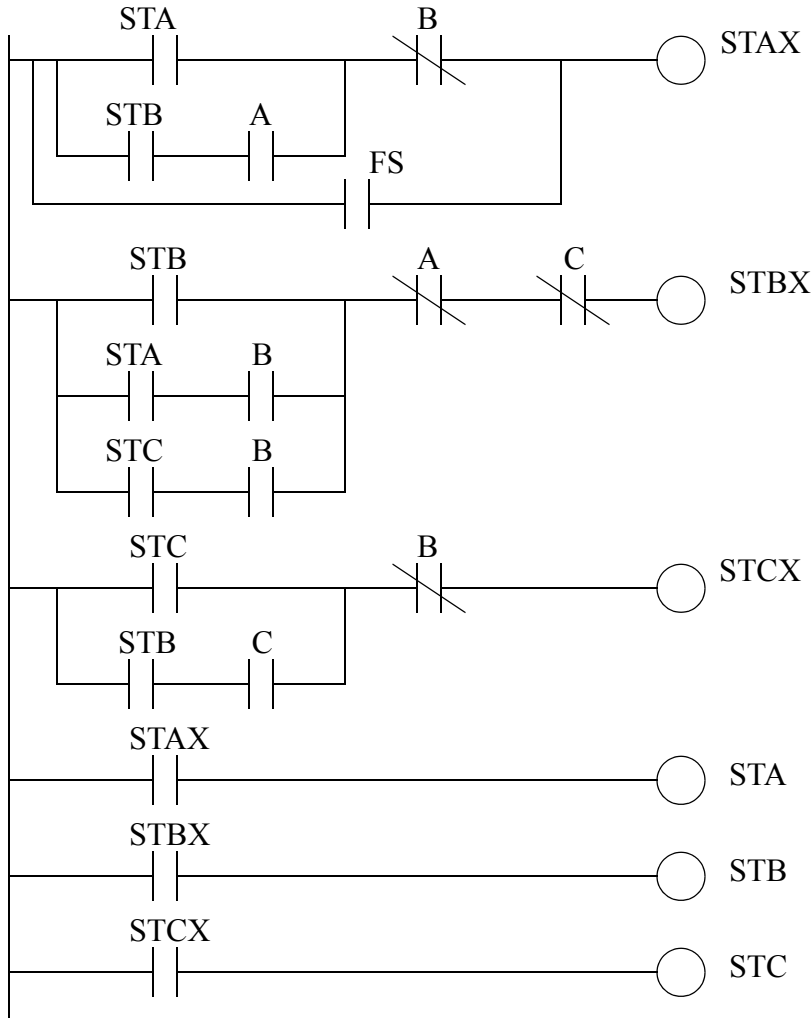




- Develop a state diagram for the control of the door.
- Convert the state diagram to ladder logic. (list the input and the output addresses first)
- Convert the state diagram to Boolean equations.

7. Design a garage door controller using a) block logic, and b) state-transition equations. The behavior of the garage door controller is as follows,
- there is a single button in the garage, and a single button remote control.
  - when the button is pushed the door will move up or down.
  - if the button is pushed once while moving, the door will stop, a second push will start motion again in the opposite direction.
  - there are top/bottom limit switches to stop the motion of the door.
  - there is a light beam across the bottom of the door. If the beam is cut while the door is closing the door will stop and reverse.
  - there is a garage light that will be on for 5 minutes after the door opens or closes.

8. Convert the following ladder logic to Boolean equations and then draw the state diagram for the system. Is something missing from the system?



9. A program is to perform the following actions for a self-service security check. The device will allow bags to be inserted to the test chamber through an entrance door. If the bag passes the check it can be removed through an exit door, otherwise an alarm is sounded. Create a state diagram using the steps below.

1. The machine starts in an 'idle' state. The 'open\_entry' output is activated to open the input door. The 'open\_exit' output is deactivated to close the output door.
2. When a bag is inserted the 'bag\_detected' input goes high. The 'open\_entry' output should be deactivated to close the door.
3. When the 'entry\_door\_closed' and 'exit\_door\_closed' inputs are active then a 'test' output will be set high to start a scan of the bags.
4. When the scan of the bags is complete a 'scan\_done' input is set. The 'test' output should be turned off.
5. The scan results in two real values 'nitrates' and 'mass'. The calculation below is performed. If the 'risk' is below 0.3, or above 23.5, then the machine enters an alarm state (step 8), otherwise it continues to step 6.

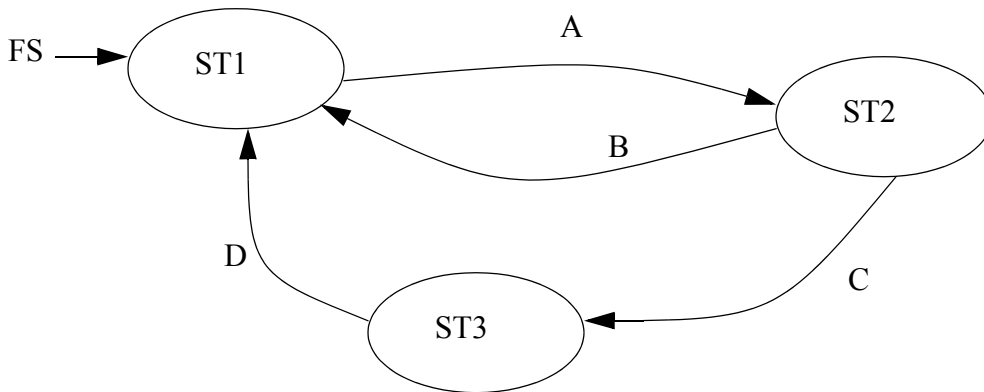
$$risk = 4^{nitrates} + \sqrt{mass}nitrates$$

6. The 'open\_exit' output is activated to open the exit door. The machine waits until the 'bag\_detected' input goes low.

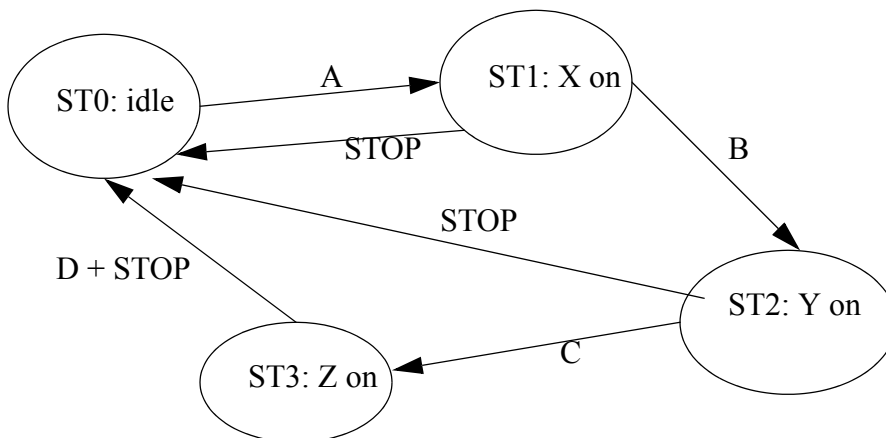
7. The 'open\_exit' output is deactivated to close the door. The machine waits until the 'exit\_door\_closed' input is high before returning to the 'idle' state.
8. In the alarm state an operator input 'key' must be active to open the exit door. After this input is released the door will close and return to the 'idle' state.

## 12.4 ASSIGNMENT PROBLEMS

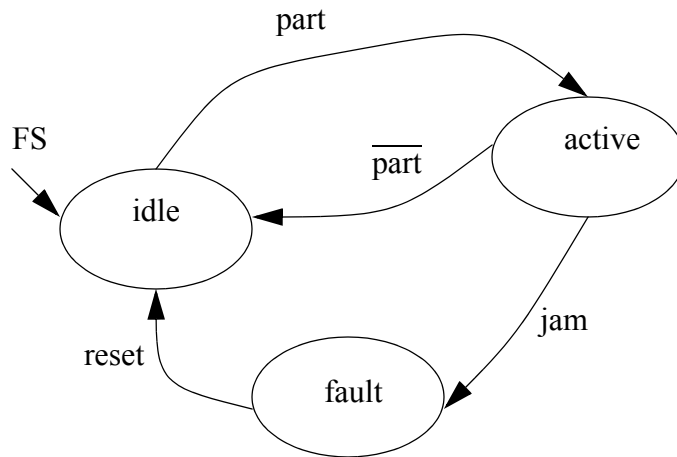
1. Describe the difference between the block logic, delayed update, and transition equation methods for converting state diagrams to ladder logic.
2. Write the ladder logic for the state diagram below using the block logic method.



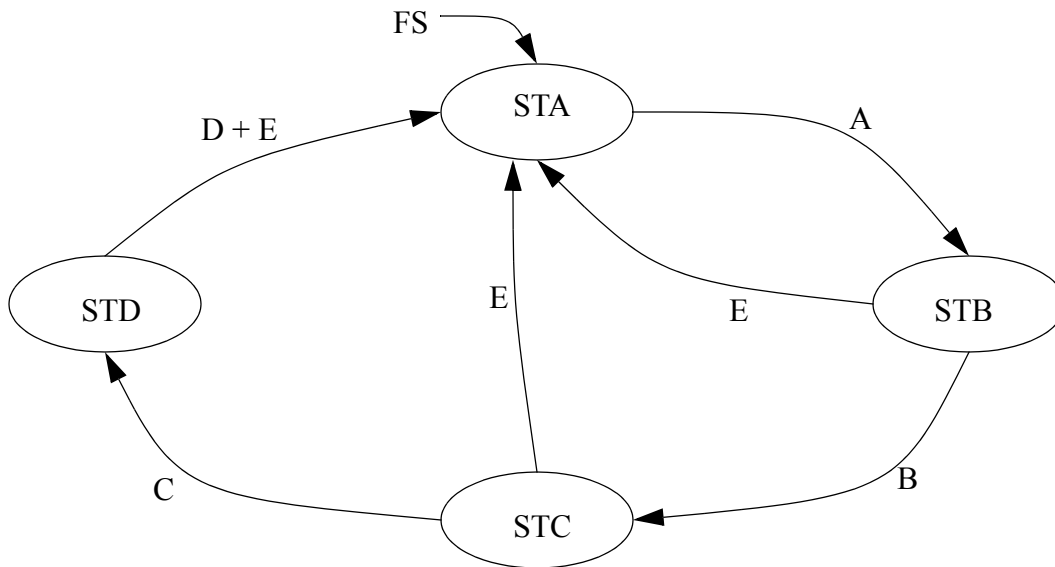
3. Convert the following state diagram to ladder logic using the block logic method. Give the stop button higher priority.



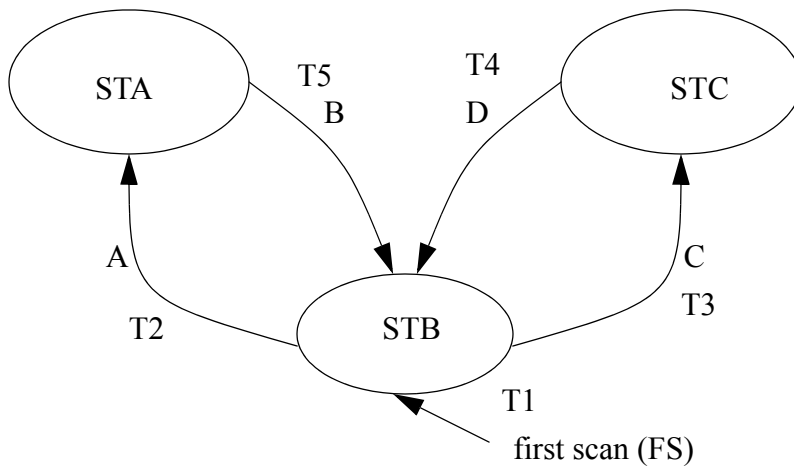
4. Convert the following state diagram to ladder logic using the delayed update method.



5. Use equations to develop ladder logic for the state diagram below using the delayed update method. Be sure to deal with the priority problems.



6. Implement the State-Transition equations in the figure below with ladder logic.



$$T1 = FS$$

$$T2 = STB \cdot A$$

$$T3 = STB \cdot C$$

$$T4 = STC \cdot D$$

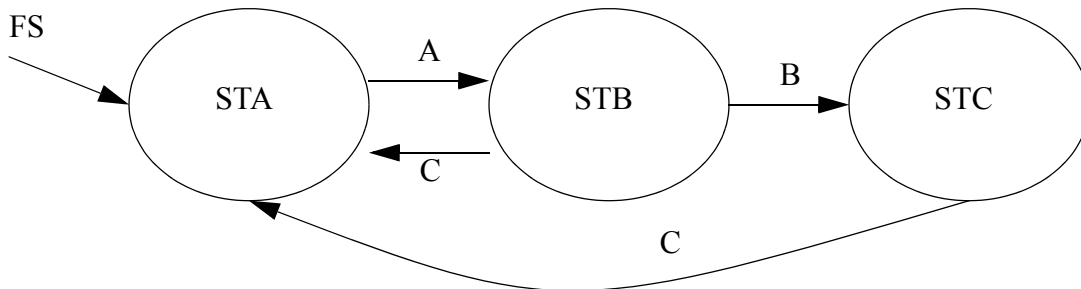
$$T5 = STA \cdot B$$

$$STA = (STA + T2) \cdot \overline{T5}$$

$$STB = (STB + T5 + T4 + T1) \cdot \overline{T2} \cdot \overline{T3}$$

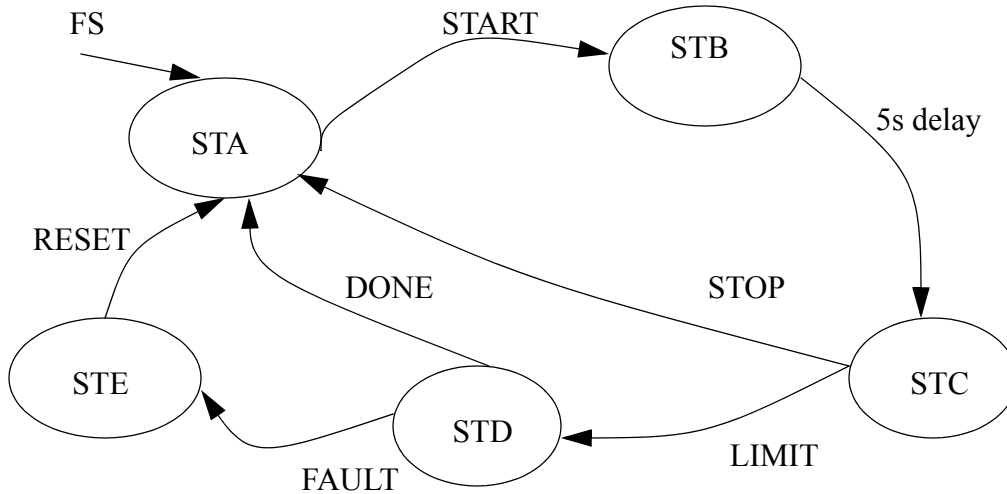
$$STC = (STC + T3 \cdot \overline{T2}) \cdot \overline{T4}$$

7. Write ladder logic to implement the state diagram below using state transition equations.

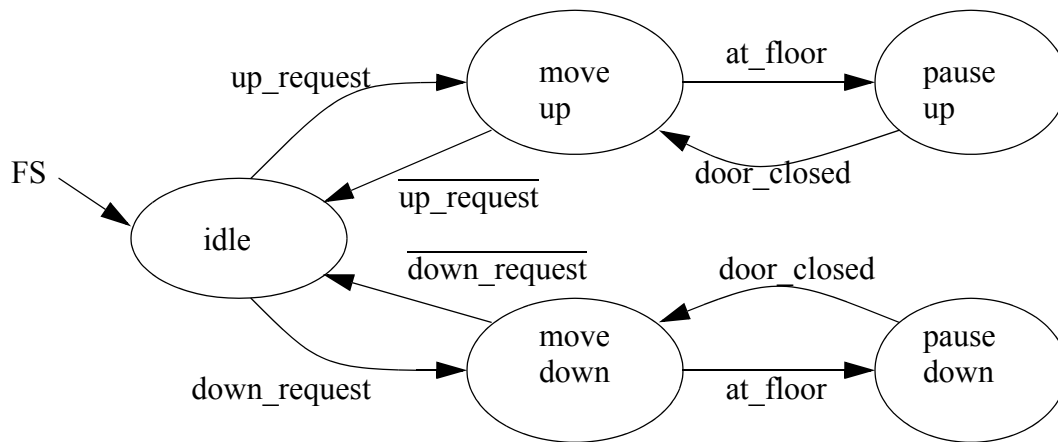


8. Convert the following state diagram to ladder logic using a) an equation based method, b) a method that is

not based on equations.

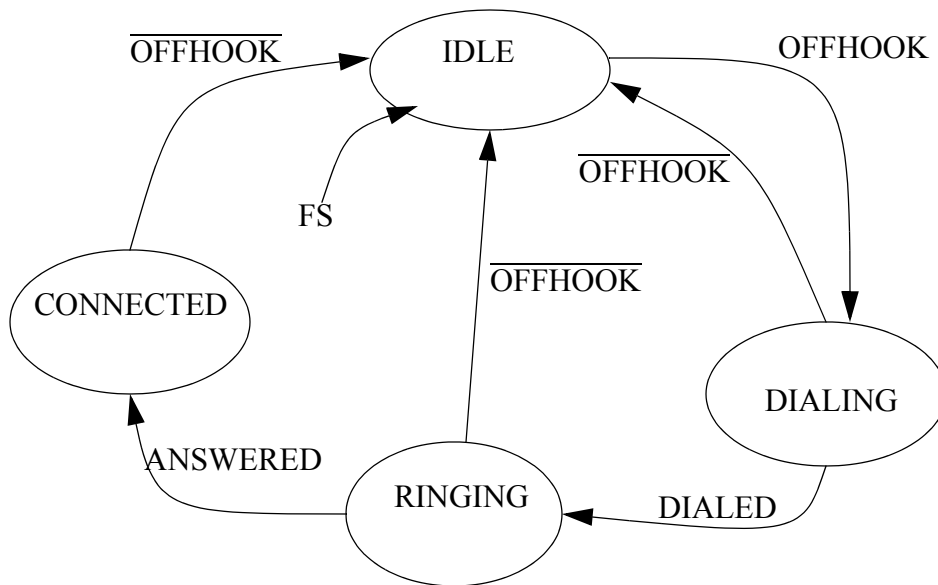


9. The state diagram below is for a simple elevator controller. a) Develop a ladder logic program that implements it with Boolean equations. b) Develop the ladder logic using the block logic technique. c) Develop the ladder logic using the delayed update method.



10. Write ladder logic for the state diagram below a) using an equation based method. b) without using an

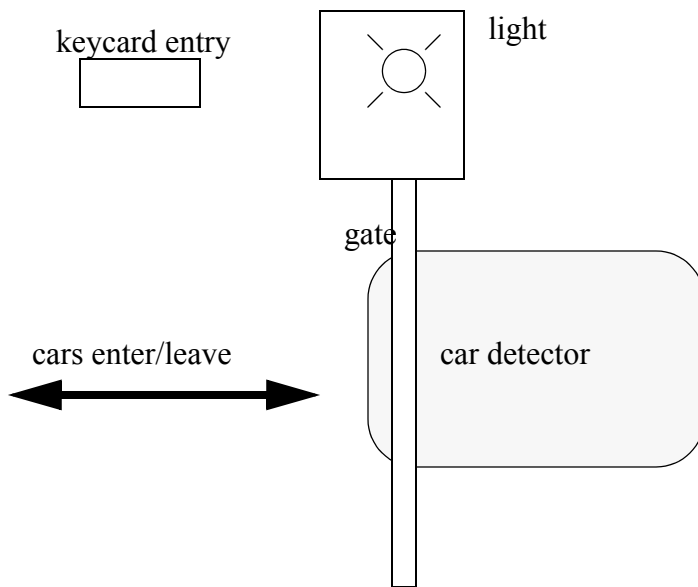
equation based method.



11. For the state diagram for the traffic light example, add a 15 second green light timer and speed up signal for an emergency vehicle. A strobe light mounted on fire trucks will cause the lights to change so that the truck doesn't need to stop. Modify the state diagram to include this option. Implement the new state diagram with ladder logic.
12. Design a program with a state diagram for a hydraulic press that will advance when two palm buttons are pushed. Top and bottom limit switches are used to reverse the advance and stop after a retract. At any time the hands removed from the palm button will stop an advance and retract the press. Include start and stop buttons to put the press in and out of an active mode.
13. In dangerous processes it is common to use two palm buttons that require an operator to use both hands to start a process (this keeps hands out of presses, etc.). To develop this there are two inputs (P1 and P2) that must both be turned on within 0.25s of each other before a machine cycle may begin.

Develop ladder logic with a state diagram to control a process that has a start (START) and stop (STOP) button for the power. After the power is on the palm buttons (P1 and P2) may be used as described above to start a cycle. The cycle will consist of turning on an output (MOVE) for 2 seconds. After the press has been cycled 1000 times the press power should turn off and an output (LIGHT) should go on.

14. Use a state diagram to design a parking gate controller.



- the gate will be raised by one output and lowered by another. If the gate gets stuck an over current detector will make a PLC input true. If this is the case the gate should reverse and the light should be turned on indefinitely.
- if a valid keycard is entered a PLC input will be true. The gate is to rise and stay open for 10 seconds.
- when a car is over the car detector a PLC input will go true. The gate is to open while this detector is active. If it is active for more that 30 seconds the light should also turn on until the gate closes.

15. This morning you received a call from Mr. Ian M. Daasprate at the Old Fashioned Widget Company. In the past when they built a new machine they would used punched paper cards for control, but their supplier of punched paper readers went out of business in 1972 and they have decided to try using PLCs this time. He explains that the machine will dip wooden parts in varnish for 2 seconds, and then apply heat for 5 minutes to dry the coat, after this they are manually removed from the machine, and a new part is put in. They are also considering a premium line of parts that would call for a dip time of 30 seconds, and a drying time of 10 minutes. He then refers you to the project manager, Ann Nooyed.

You call Ann and she explains how the machine should operate. There should be start and stop buttons. The start button will be pressed when the new part has been loaded, and is ready to be coated. A light should be mounted to indicate when the machine is in operation. The part is mounted on a wheel that is rotated by a motor. To dip the part, the motor is turned on until a switch is closed. To remove the part from the dipping bath the motor is turned on until a second switch is closed. If the motor to rotate the wheel is on for more that 10 seconds before hitting a switch, the machine should be turned off, and a fault light turned on. The fault condition will be cleared by manually setting the machine back to its initial state, and hitting the start button twice. If the part has been dipped and dried properly, then a done light should be lit. To select a premium product you will use an input switch that needs to be pushed before the start button is pushed. She closes by saying she will be going on vacation and you need to have it done before she returns.

You hang up the phone and, after a bit of thought, decide to use the following outputs and inputs,

#### INPUTS

- I/1 - start push button
- I/2 - stop button
- I/3 - premium part push button
- I/4 - switch - part is in bath on wheel
- I/5 - switch - part is out of bath on wheel

#### OUTPUTS

- O/1 - start button
- O/2 - in operation
- O/3 - fault light
- O/4 - part done light
- O/5 - motor on
- O/6 - heater power supply

- a) Draw a state diagram for the process.
- b) List the variables needed to indicate when each state is on, and list any timers and counters used.
- c) Write a Boolean expression for each transition in the state diagram.
- d) Do a simple wiring diagram for the PLC.
- e) Write the ladder logic for the state that involves moving the part into the dipping bath.

16. Design ladder logic with a state diagram for the following process description.

- a) A toggle start switch (TS1) and a limit switch on a safety gate (LS1) must both be on before a solenoid (SOL1) can be energized to extend a stamping cylinder to the top of a part. Should a part detect sensor (PS1) also be considered? Explain your answer.
- b) While the stamping solenoid is energized, it must remain energized until a limit switch (LS2) is activated. This second limit switch indicates the end of a stroke. At this point the solenoid should be de-energized, thus retracting the cylinder.
- c) When the cylinder is fully retracted a limit switch (LS3) is activated. The cycle may not begin again until this limit switch is active. This is one way to ensure that a new part is present, is there another?
- d) A cycle counter should also be included to allow counts of parts produced. When this value exceeds some variable amount (from 1 to 5000) the machine should shut down, and a job done light lit up.
- e) A safety check should be included. If the cylinder solenoid has been on for more than 5 seconds, it suggests that the cylinder is jammed, or the machine has a fault. If this is the case the machine should be shut down, and a maintenance light turned on.
- f) Implement the ladder diagram on a PLC in the laboratory.
- g) Fully document the ladder logic and prepare a short report - This should be of use to another engineer that will be maintaining the system.