ET 438B-Online
Sequential Control and Data Acquisition
Laboratory 7
PLC Function Programming Using Ladder Logic

Laboratory Learning Objectives

1.) Explain the operation of on and off delay timers used in ladder logic and use them in programming exercises
2.) Explain the operation of up/down counters used in ladder logic programs and use them in programming exercises.
3.) Define operating states of a given physical system
4.) Write Boolean state equations for a physical system and use them to program ladder logic.

Technical Background

Programmable Logic Controllers have other uses along with the implementation of ladder logic in software. Two of the most common functions are those of timing processes and counting events. These functions replace the costly electromechanical and discrete electronic devices used to accomplish the tasks before the arrival of PLCs. Time driven processes we see in the home are automatic washing machines and clothes dryers. A timing device controls a sequence of events in these processes moving the appliances from one operating state to another. Boolean state equations provide an mathematical basis for describing time and event driven processes. Designs based on well thought out Boolean state equations produce control programs that are less likely to have logical errors and have clearer structures.

Timer Functions

On-delay and off-delay timers provide fundamental timing functions in sequential control systems. Figure 1 shows the schematic symbols for electromechanical timer relays and contacts. Timer coils can have either two or three-wire representations and usually have labels that indicate the device is a timer (e.g. TR). Contacts associated with the coil have the same identifier with a sequence number that indicate the linkage between the coil and the contacts (e.g. the coil TR controls TR1, TR2, TR3). Figure 1 also shows the contact symbols associated both the on and off delay devices.

On-delay timers must have their coils' energized before the timing sequence can start. All contacts associated with the coils change state after a predetermined time interval. Off-delay timers maintain contact states when their coils' are energized but start the timing period AFTER the coil is de-energized. All associate contacts will change states after the timers preset interval

when the coil is de-energized. Electromechanical/Electronic timers were expensive. PLCs implement both timing functions in electronic hardware and software at a fraction of the cost.

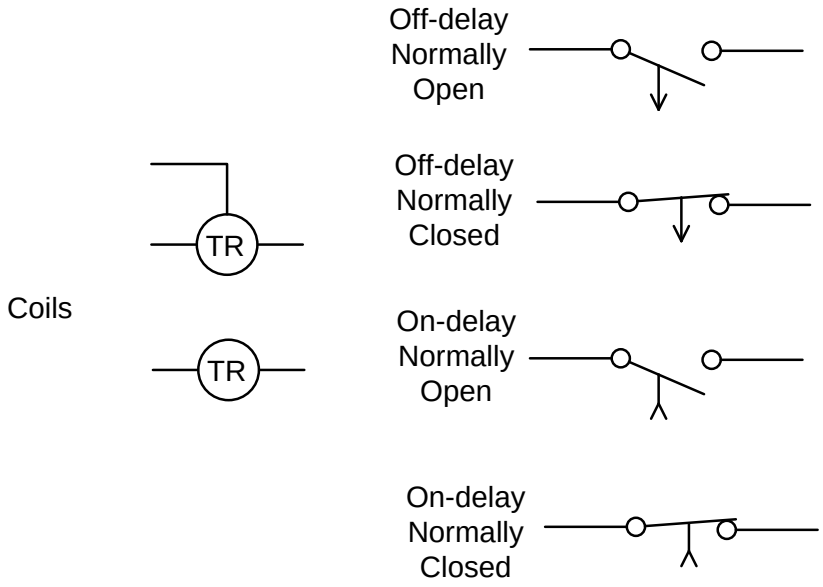

Figure 1. Electromechanical Timer Schematic Symbols.

Do-More series PLCs have both on-delay and off-delay functions accessible from the **Instruction Tool Box** and Timer/Counter/Drum menu of Do-More Designer programming software. Figure 2 shows the ladder diagram symbols for both the on and off delay timers. The ONDTMR block provides the on-delay timing function while the OFFDTMR implements

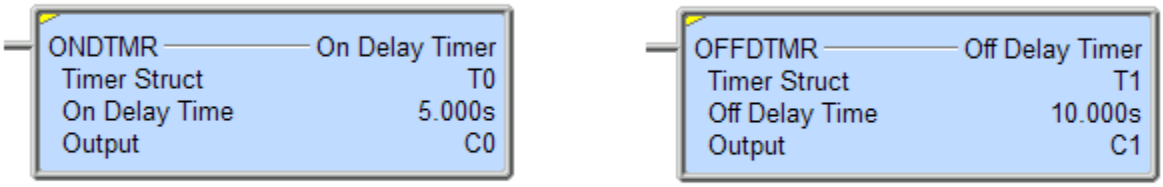

Figure 2. ONDTMR and OFFDTMR Timer Functions in the Do-More Designer Development Environment.

the off-delay timer. Timers must be the output of a rung. The Boolean result of the rung conditions and the timer type determine the action of the outputs. An off-delay timer begins its timing sequence when the timer input transitions from TRUE to FALSE. If the input rung condition changes from FALSE to TRUE before the preset time delay ends, then the timer resets.

Table 1 shows the parameters of the OFFDTMR function. The notation Tn specifies the timer structure n in the Do-More Designer where n is an integer. A TRUE to FALSE transition on this input starts the timer. The timer is setup by entering values for the Timer Structure, Off Delay Time, and Output bit.

Table-1 OFFDTMR -Off-Delay Timer Function Parameters

Parameter	Parameter Type	Data Type	Description
INPUT	Input	Bit	T to F transition starts timer F to T transition stops and resets timer
HH:MM:SS:mmm	Input	Time	Programmed Time delay where HH is number of hours, MM is minutes, SS is seconds and mmm is milliseconds.
OUTPUT	Output	Bit	Timer output. Bit stays set while timer elapsed time value is less than Off Delay Time value. Becomes reset when Timer value reaches zero..
Tn.Done	Output	Bit	This bit is reset when input is T. The timer sets this bit after time reaches zero.
Tn. Reset	Output	Bit	Bit is set if the input rung is T or if the timer is reset by a timer reset instruction.
Tn.Zero	Output	Bit	Bit is set if the accumulator value is 0
Tn.Acc	Output	32 bit signed value	Numerical value that represents the time remaining before the preset time delay value reaches zero.
Tn.Timing	Output	Bit	Bit is set when the timer is enabled by a T to F transition and Tn.Acc>0.

The NCC and NOC contact instructions can use any of the bit data types of the timer as inputs They are also valid address for the Coil/Bit Output instructions.

The timing diagram in Figure 3 shows how the OFFDTMR function's I/O signals interact. The timer remains inactive until its input rung becomes TRUE. At that time the OFFDTMR instruction sets the OUTPUT bit specified in the timer instruction. The .Done bit is reset. The output bit remains high until the input rung becomes FALSE and the Off Delay timer value reaches zero. At that time the .Done bit is set. This bit remains set until the timer is reset.

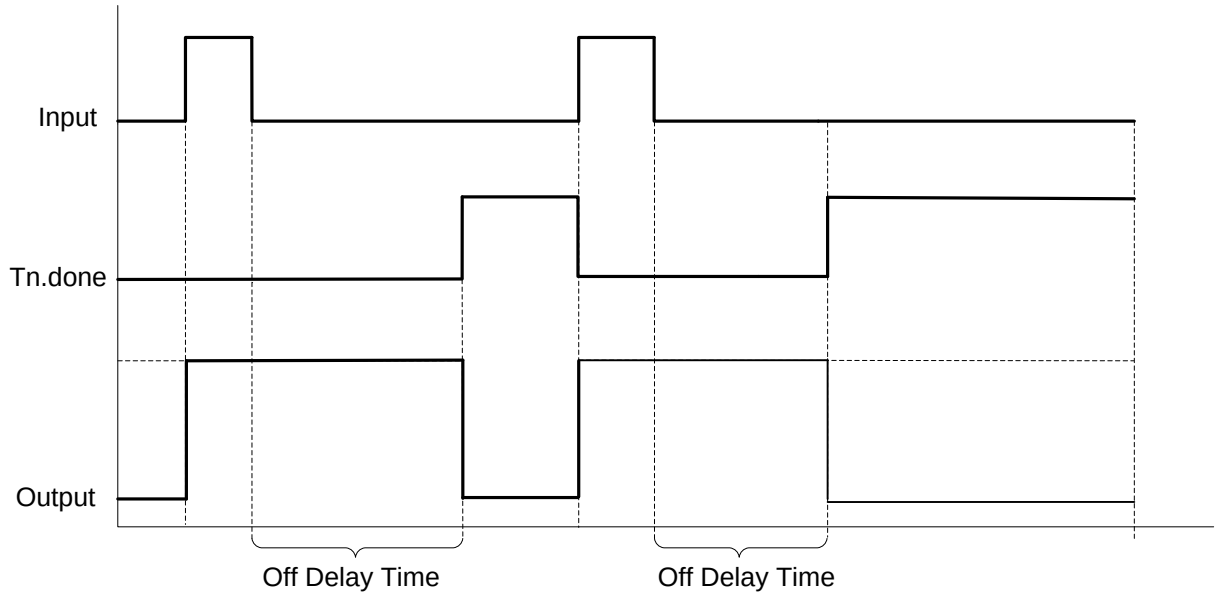

Figure 3. OFFDTMR Timing Diagram Showing the Sequence of Operation.

The on-delay timer function ONDTMR is the logical opposite of the OFFDTMR function. Table 2 describes the actions of the function's parameters. This function delays its bit output until after the timer value reaches the On-Delay time value. Figure 4 shows the timing diagram for the ONDTMR function. The ONDTMR function resets when the input parameter, INPUT, becomes FALSE before the On-Delay time value reaches zero.

Table-2 ONDTMR-On-Delay Timer Function Parameters

Parameter	Parameter Type	Data Type	Description
INPUT	Input	Boolean	F to T starts timer T to F stops and resets timer
HH:MM:SS:mmm	Input	Time	Programmed Time delay where HH is number of hours, MM is minutes, SS is seconds and mmm is milliseconds.
OUTPUT	Output	Bit	Timer output. Bit is set after timer exceeds On-Delay Time value.
Tn.Done	Output	Bit	This bit is reset when input is F. The timer sets this bit after time reaches On-Delay preset time. The bit remains set until INPUT evaluates FALSE

Parameter	Parameter Type	Data Type	Description
Tn. Reset	Output	Bit	Bit is set if the input rung is F or if the timer is reset by a timer reset instruction.
Tn.Zero	Output	Bit	Bit is set if the accumulator value is 0
Tn.Acc	Output	32 bit signed value	Numerical value that represents the time since the timer rung became TRUE. This value can exceed the preset On-Delay time and continues to increase until the timer is reset.
Tn.Timing	Output	Bit	Bit is set when the timer is enabled by a F to T transition and Tn.Acc<On-Delay time setting.

.

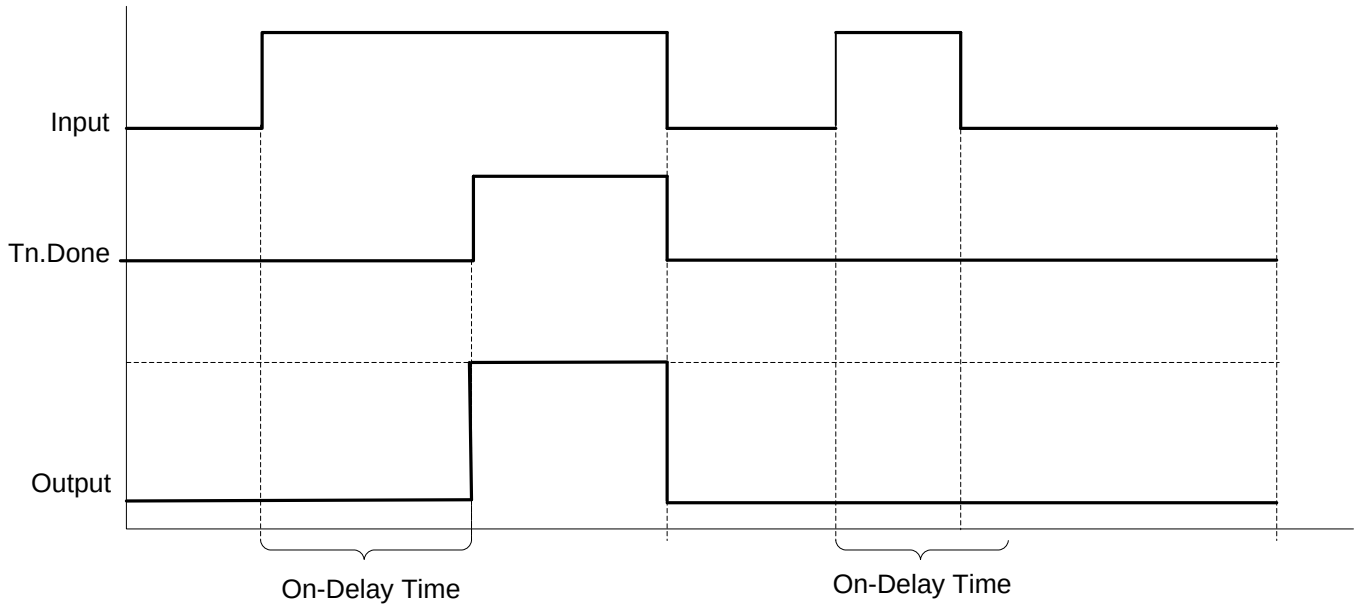

Figure 4. ONDTMR Function Timing Diagram Showing Timer Reset.

Figures 5a and 5b show rungs that include the ONDTMR and OFFDTMR functions.

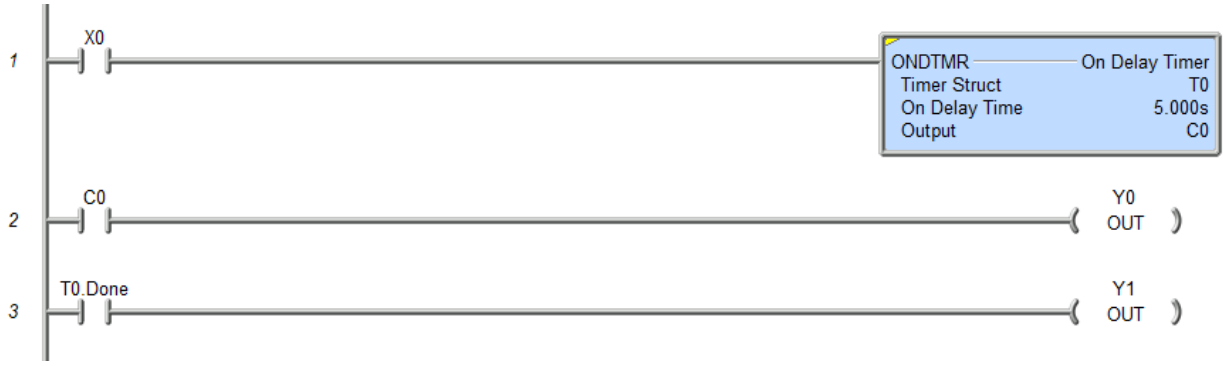

Figure 5a. On-delay Timer Application with a 5 Second Programmed Time Delay.

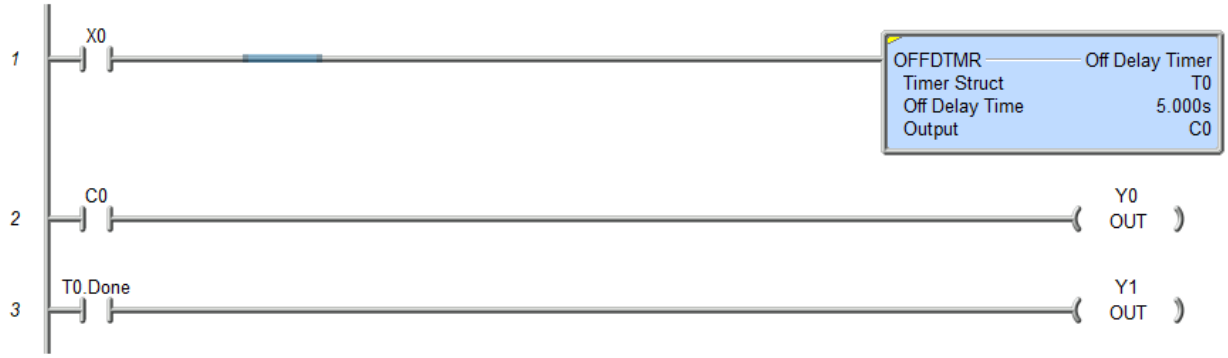

Figure 5b. Off-delay Timer Application with a 5 Second Programmed Time Delay.

Figure 5a shows an on-delay timer activated when the digital input X0 becomes TRUE. If this input produces a TRUE for more than five seconds, the values of OUTPUT, C0 and T0.Done in T0 change, setting the underlying memory bits to 1. The NOC instruction in the next rung tests the variable C0 and returns a TRUE since the addressed bit is 1. This causes the bit OUT instruction at digital address Y0 to set its bit to 1 and energize the output point. The third rung tests the T0.Done bit that is set after the timer elapsed time exceeds the preset value of five seconds. Five seconds after the input becomes TRUE, this bit is set, which causes the rung to become TRUE. The output Y1 sets the addressed bit to 1 based on the TRUE rung. If the NOC instruction at X0 becomes false before the On-Delay time elapses, the timer resets and there is no change in output.

Figure 5b shows the operation of an off-delay timer. This timer has a programmed time delay of 5 seconds. The OFFDTMR instruction actives when the input at address X0 transitions from TRUE to FALSE. The OFFDTMR function starts timing. The C0 output bit remains set until the value of Off-Delay time becomes zero. The T0.Done bit is set when the Off-Delay time

becomes zero and remains set until the timer rung changes from FALSE to TRUE. If the input point at address X0 changes from FALSE to TRUE before the elapsed time equals zero the timer resets.

The NOC addressing the C0 bit in the next rung shown in Figure 5b tests the output variable of the OFFDTMR instruction. When its addressed bit is set to 1, the NOC instruction returns a TRUE and the Y0 output is energized. The third rung uses the T0.Done bit to control the output Y1. This bit is set after the OFFDTMR instruction sees a TRUE-FALSE transition and the timer elapsed time reaches zero. It will remain set until the timer is reset by another FALSE-TRUE transition.

The Do-More Designer programming environment includes several other timing functions. The software has extensive online help topics included in the development environment, It also includes links to short online videos that demonstrate how to use the PLC instructions. These are available whenever the software is used on a computer connected to the Internet. All instructional videos are available for download to the student's local computer system for future use.

Counter Functions

Another common task implemented using a PLC system is to record the number of events that occur in a sequential system. The counter functions of the PLC replace electromechanical counters in modern industrial control processes. The Do-More Designer software and associated PLCs support a number of these functions. The most common types of counters are the up counter, CNT, the down counter, CNTDN, and the up/down counter, UDC instructions These instructions handle a variety of tasks found in sequential control design.

Figure 6 shows the function blocks of the up, down and up/down counters used for ladder diagram programming. These functions are accessible from the **Instruction Tool Box** and

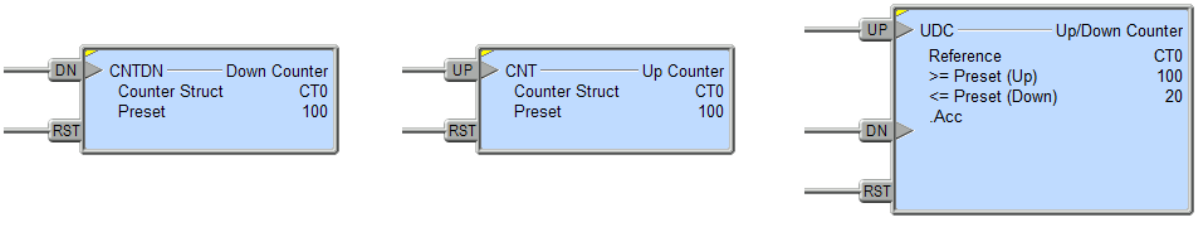

Figure 6. Counter Functions In Do-More Designer Development Software .

Timer/Counter/Drum menu of Do-More Designer programming software. These are output functions and should be located on the right side of a ladder diagram rung.

Tables 3 through 5 list the functional parameters of each structure. Table 3 shows the functional parameters of the CNTD instruction. The CNTD has two input parameters. The code CTn.xxx identifies individual bits in the counter structure where n is an integer value from 0-255. The DN input decrements the programmed value of the counter by 1 each time the rung input

Table-3 CNTD- Count-Down Counter Function Parameters

Parameter	Parameter Type	Data Type	Description
DN	Input	Boolean	Decreases CTn.Acc value by 1 when rung make a FALSE-to-TRUE transition
RST	Input	Boolean	Resets the CTn.Acc value to preset value after a FALSE-TRUE transition.
CTn.Done	Output	Bit	Bit is set when the value in CTn.Acc ≤ 0
Preset	Input	Integer	Programmed counter value
CTn.Reset	Output	Bit	Bit is set when the counter is reset by the input RST
CTn.Acc	Output	Integer	Stores a 32-bit integer that is the current number of counts remaining
CTn.Zero	Output	Bit	Bit is set when the value of CTn.Acc=0

transitions from FALSE to TRUE. The RST input clears the number of counts stored in the counter when its input transitions from FALSE to TRUE. This can be used to change the state of the output parameter, CTn.Done, before or after the programmed number of counts occurs and to reset the counter value to its Preset value after the programmed count number occurs. The instruction's outputs are:

CTn.Acc: the current count value,
CTn.Done: the bit indicating when the number of input transitions equals or is less than the preset count number,
CTn.Reset: the bit indicating when the counter was reset,
CTn.Zero: the bit indicating when the counter Acc value is at zero,

The CTn.Done bit identifies when the accumulated number of input counts has reached the preset value. This is the most used output bit of the counter since it identifies this condition.

Table 4 lists and defines the CNT instruction. This counter instruction increments a value variable until it reaches a predefined value. The Preset parameter sets the maximum number of counts the instruction registers before it sets the Tn.Done bit.

Table-4 CNT- Count-Up Counter Function Parameters

Parameter	Parameter Type	Data Type	Description
UP	Input	Boolean	Increases CTn.Acc value by 1 when rung make a FALSE-to-TRUE transition
RST	Input	Boolean	Resets the CTn.Acc value to zero value after a FALSE-TRUE transition.
CTn.Done	Output	Bit	Bit is set when the value in CTn.Acc ≥ Preset
Preset	Input	Integer	Programmed counter value
CTn.Reset	Output	Bit	Bit is set when the counter is reset by the input RST
CTn.Acc	Output	Integer	Stores a 32-bit integer that is the current number of counts recorded.
CTn.Zero	Output	Bit	Bit is set when the value of CTn.Acc=0

.

The RST input clears the counter accumulator value to zero and sets the Tn.Reset bit. This can be used to change the state of the output parameter, CTn.Done, before the programmed number of counts occurs and to reset the counter value to a zero value before or after the programmed count number occurs. The instruction's outputs are:

CTn.Acc: the current count value,
CTn.Done: the bit indicating when the number of input transitions equals or is less than the preset count number,
CTn.Reset: the bit indicating when the counter was reset,
CTn.Zero: the bit indicating when the counter Acc value is at zero,

The CTn.Done bit identifies when the accumulated number of input counts has reached the preset value. This is the most used output bit of the counter since it identifies this condition

Table 5 lists the parameters for the UOC, the up/down counter instruction. This structure combines the functions of the up and down counters into a single package. This instruction has two counter inputs, one that increments and another that decrements the current counter value. There are also two outputs to indicate if the counter has reached 0 or the programmed upper limit. The RST has the the same function in this structure as they have in the CNT and CNTDN instructions described previously. Figures 7 shows the counter functions applied in rungs of ladder logic programming.

Table 5 UDC- Count-Up/Down Counter Function Parameters

Parameter	Parameter Type	Data Type	Description
UP	Input	Boolean	Increases CTn.Acc value by 1 when rung make a FALSE-to-TRUE transition
DN	Input	Boolean	Decreases CTn.Acc value by 1 when rung make a FALSE-to-TRUE transition
RST	Input	Boolean	Resets the CTn.Acc value to zero value after a FALSE-TRUE transition.
CTn.Done	Output	Bit	Bit is set any time the counter CT.Acc ≥ Up Preset value if counter is configured to use Up Preset.
CTn.DnDone	Output	Bit	Bit is set any time the counter CT.Acc ≤ Down Preset value if counter is configured to use Down Preset
CTn.Reset	Output	Bit	Bit is set if the RST input is TRUE.
CTn.zero	Output	Bit	Bit is set any time the value in the Counter structure's accumulator (.Acc) is 0

The CNT instruction in rung 1 increments the CT0.Acc value each time the digital input X0 makes a FALSE-to-TRUE transition. The value of CT0.Acc will increase with each transition. The programmed value is 5, so the value of CT0.Done will remain FALSE until the

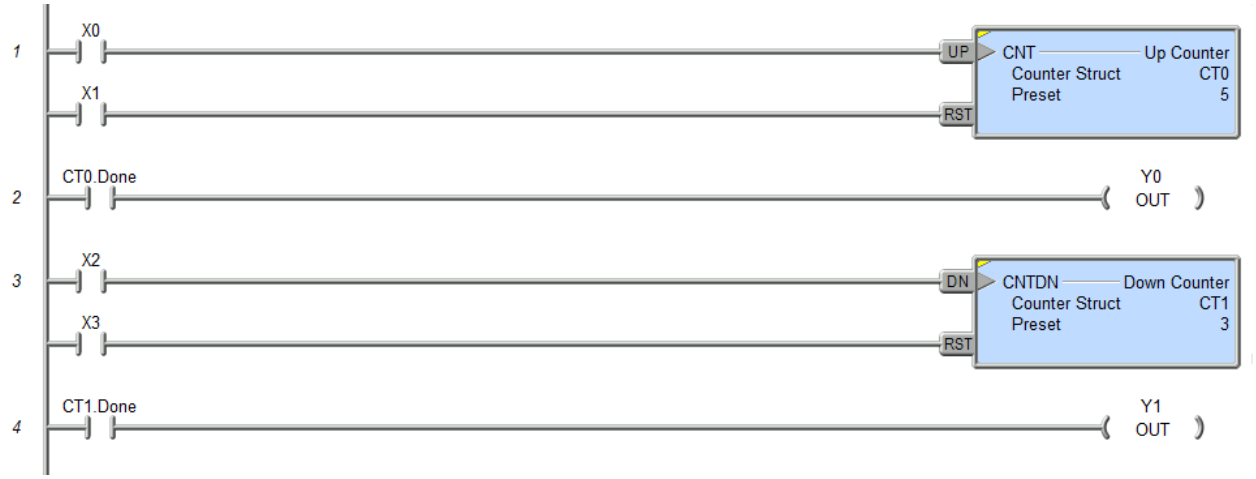

Figure 7. Up and Down Counters in Ladder Logic.

CNT input sees 5 transitions and the value of CT0.Acc=5. The input bit addressed at input X1 by the NOC instruction connects to the RST input of the counter. It will reset the counter any

time its bit value is set and that rung evaluates as TRUE. Resetting the counter causes the CT0.Acc value to become zero.

The NOC in rung 2 links to the CNT instruction in rung 1. The NOC contact addresses the bit output of the CNT instruction with the CT0.Done parameter. Each time the CNT counter instruction records 5 events the NOC in rung 2 sees a set bit at location CT0.Done. This causes the rung to evaluate TRUE and the OUT instruction to set the bit located at Y0. The output CT0.Done of the counter will remain set until input X1 evaluates TRUE and resets the counter. Other program rungs can use CT0.Done to trigger events when it is set.

Rungs 3 and 4 in Figure 7 implement a down counter using the CNTDN instruction. The input to this counter is X2 and located in rung 3. A NOC instruction tests this input and causes the counter to decrement CT1.Acc each time this rung makes a FALSE-TRUE transition. When the value stored in CT1.Acc equals zero the counter CT1.Done bit is set. This occurs after three DN input transitions take place since the preset value is 3. The NOC instruction in rung 4 will evaluate TRUE when the CT1.Done bit is set causing the Y1 output bit to be set. This will energize any device connect to this point on the PLC.

The counter RST line using the X3 input to reset the counter. The counter will reset any time the NOC instruction evaluates TRUE. The bit located at X3 must be set for the reset rung to clear the accumulator.

The Do-More Designer software has extensive online help topics included in the development environment, It also includes links to short online videos that demonstrate how to use the PLC instructions. These are available whenever the software is used on a computer connected to the Internet. All instructional videos are available for download to the student's local computer system for future use.

Boolean State Equations and Sequential System Design

Inspection and experience can provide solutions to simple sequential control problems but more complex system require structured methods. Boolean state equations coupled with Boolean algebra and state diagrams form a set of analytic tools for tackling complex sequential control problems. The solutions found from using these methods are structured and produce PLC code that is easily followed and less likely to have logical errors. A state-based design may not produce a minimal logic design if there are redundant states or the resulting Boolean equations are not reduced to minimal expressions.

A Boolean state equation provides a formal way of expressing the conditions that cause a sequential system to enter and leave a given state. An informal statement of a state equation is

State X = (Is currently in state X + Just arrived from another state)·(has not left to another state)

This expression states that a system is in state X if it was already in state X or logical conditions cause it to enter state X at this time. The ANDed statements specify logical conditions that would cause the system to leave the current state.

A formal mathematical version of the state equation gives more specific definitions to the statement above. Equation (1) shows the formal statement of a state equation. Figure 8 is a graphic interpretation of the equation.

$$S_i^+ = \left[S_i + \sum_{j=1}^{n} T(in)_{ji} \cdot S_j \right] \cdot \prod_{k=1}^{m} \left(\overline{T(out)_{ik} \cdot S_i} \right) \qquad (1)$$

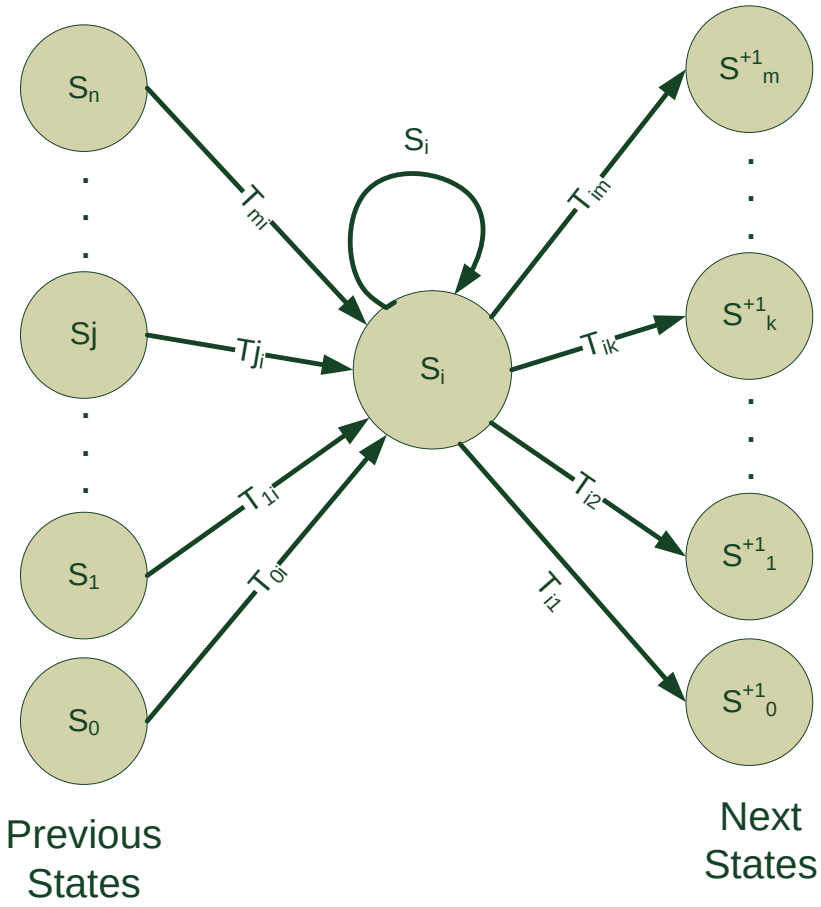

Figure 8. Graphical Representation of State Equation Showing Transitions into and out of S_i.

Where S^{+}_{i} = the next value of the state variable that describes state i
S_i = the current value of the state variable i
$T(in)_{ji}$ = logical conditions that would cause process to enter state i from state j
$T(out)_{ik}$ = logical conditions that would cause the process to leave state i for state k

State variables can take a Boolean value of 1 or 0 to indicate a sequential process is in a given state.

The transition equations $T(in)_{ji}$ represent the conditions necessary to set the state variable while the $T(out)_{ik}$ equations are the conditions necessary to reset the state variable. A state transition that depends solely on inputs is memoryless. This indicates that the transition is event driven and does not require information from a previous state. Equation 2 gives the form of a memoryless state equation. This form indicates neither set or reset functions are state dependent

$$S_i^{+1} = \left[S_i + \sum_{j=1}^{n} T(in)_{ji} \right] \cdot \prod_{k=1}^{m} (\overline{T(out)_{ik}}) \tag{2}$$

The output function relates the process states to the desired output actions. The simplest form of the output function equates the state variable to the desired output action as shown in equation (3).

$$S_i = O_i \tag{3}$$

Developing a state diagram should be the first step in designing sequential systems that have any complexity. The most critical step in developing a state diagram is identifying the states. A designer should consider the system functionality. This includes the following:

1.) normal operating state
2.) internal system behavior changes
3.) external system behavior changes
4.) system sequence of events

After determining a sequence of events, list the modes of operation where the system is performing one identifiable activity that must be started or stopped. It is possible that waiting for input is an activity, so the idle state is valid. The designer should also determine what outputs, if any, entering or leaving a state requires.

Example

A mixing process requires automation in an industrial plant. The process begins when an operator depresses a momentary contact start push button switch. When the process starts a solenoid-controlled inlet valve opens and allows a mixture to fill a tank to a specified level. A float-switch senses when the mixture reaches the correct level. The mixture must be agitated for 20 minutes and then a solenoid controlled outlet valve opens draining the contents into a holding tank for further processing. The same float switch that detects a full tank senses the tank is empty. After the tank drains, the system should be ready for another manual start. For safety reasons, the stop push button should end the process at any point in the sequence. The system should power up in the stop condition. For this process: a.) define the inputs, the outputs, and the states, b.) draw a state diagram, c.) write state equations that describe the system operation, d.) develop a PLC program that performs this control function.

Solution

a.) First define a nomenclature to use in the state diagram.

Sx = process state x , where x is 0, 1,2, 3, … n
Ix = process input x
Ox = process output x
TON(A,T) = on-delay timer function with input A and time setting T in seconds
TOF(A,T) = off-delay timer function with input A and time setting T in seconds

Now define the inputs and outputs from the process description.

Inputs: I0 = first PLC scan
I1= start push button
I2= stop push button
I3=float switch

Outputs: O0=inlet solenoid valve
O1=outlet solenoid valve
O2=agitator motor contactor

Define states for the process. Note that states are not unique. Two different designers may define the states differently but produce workable designs. The goal should be to describe system operation using a minimum number of states. Remember, each state will require a Boolean state equation.

States: S0=stop
S1=start
S2=tank filling
S3= mixture agitating
S4=tank draining

b.) The next step is to relate the inputs, outputs and states in a state diagram. A state diagram is a directed graph of nodes defining the states and arrows defining the conditions that cause state transitions. Arrows directed from the nodes that do not connect to another state indicate an output that occurs when entering or leaving the state. Figure 9 shows the state diagram for the example process. The diagram starts with an input from the first scan variable of the PLC. This variable is TRUE only for the first PLC program scan and places the process in the stop state.

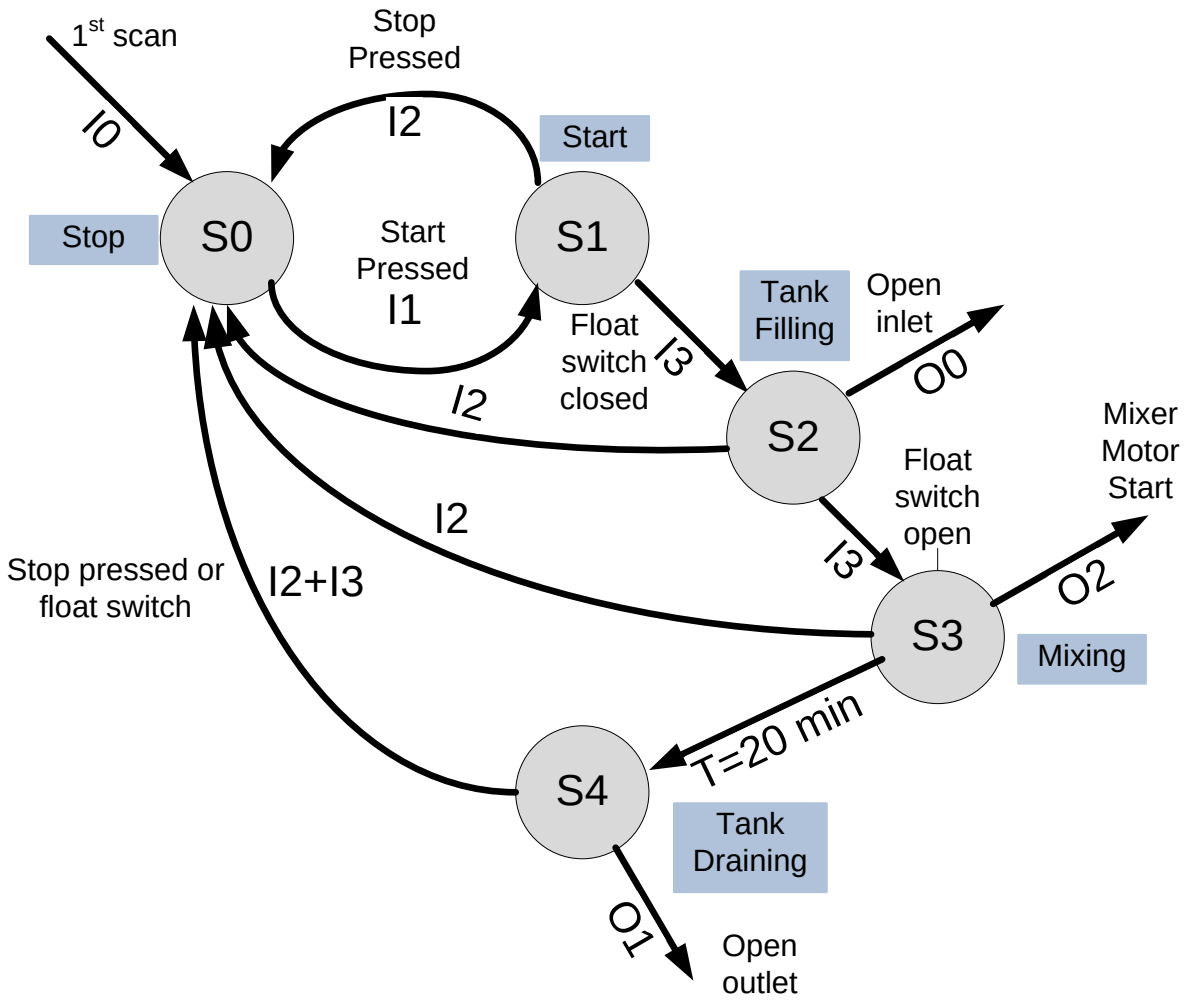

Figure 9. Example State Diagram Showing Transitions and Outputs

This guarantees that the process will be stopped even after a power outage. From this point, the process progresses to the start state if the start push button is pressed. Once in the start state, the process will transition to the tank filling state if the float switch indicates the tank is empty. If the tank is empty, the control scheme opens the inlet value allowing the tank to fill with mixture. The float switch detects a full tank and allows the process to move to the mixing state. An output energizes the mixer motor after the process enters this state. The mixing state continues until 20 minutes have elapsed, after which the process enters into the tank draining state. In this state, the drain value is opened and the state maintained until the float switch indicates the tank is empty. Pressing the stop push button at each stage of the process ends the current operation and returns the system to the stop state.

c.) State equations convert the state diagram into Boolean expressions relating the inputs to states. The first step in developing state equations is to determine the coding of the states. Individual bits can store the state variables. A number of bits, n, can store up to 2^n states. The number of states in this example is five so three bits can store all the defined states. Table 6 shows one way to code the states for this example. A selected code should attempt to only

Table-6 Coding of States In Example Problem

X1	X2	X3	Diagramed State Identifier
0	0	0	S0: Stopped State
0	0	1	S1: Start State
0	1	0	S2: Tank Filling
0	1	1	S3: Mixing State
1	1	1	S4: Tank Draining

change the value of single bit as the program transitions among the defined states. This simplifies the transitional Boolean equations. Another coding method assigns a single bit to each state. Table 7 shows this coding. This method does not take advantage of the fact the each bit can store two states and is quite wasteful of resources. It is easier to understand since each bit

Table-7 Coding of States In Example Problem

S0	S1	S2	S3	S4	Diagramed State Identifier
1	0	0	0	0	S0: Stopped State
0	1	0	0	0	S1: Start State
0	0	1	0	0	S2: Tank Filling
0	0	0	1	0	S3: Mixing State
0	0	0	0	1	S4: Tank Draining

represents a condition of the system process and it is either in the state (1) or out of the state (0). This example uses the coding from Table 7.

Examining the state diagram and utilizing the general form of the state equations given in (1) and (2) produces the system of Booleans state equations show in equations 3a-3e.

$$
\begin{aligned}
&\text{a.)} \quad S0^{+} = \left[S0 + I0 + I2 + I3 \cdot S4\right] \cdot \overline{I2} \\
&\text{b.)} \quad S1^{+} = \left[S1 + I1 \cdot S0\right] \cdot \overline{I2} \cdot \overline{I3} \\
&\text{c.)} \quad S2^{+} = \left[S2 + I3 \cdot S1\right] \cdot \overline{I2} \cdot \overline{I3} \\
&\text{d.)} \quad S3^{+} = \left[S3 + I3 \cdot S2\right] \cdot \overline{TON(S3, T)} \cdot \overline{I2} \\
&\text{e.)} \quad S4^{+} = \left[S4 + TON(S3, T) \cdot S3\right] \cdot \overline{I3} \cdot \overline{I2}
\end{aligned}
\tag{3}
$$

These equations combine memoryless and retained state memory in their implementation. The set conditions are enclosed in the square brackets and the reset conditions follow the terms enclosed in the brackets.

Output equations relate the inputs and states to the desired output actions. Using a single bit to indicate a state makes the output equations simple. Equations (4) give the output relationships for the example.

$$
\begin{aligned}
O0 &= S2 \quad (\text{Open inlet}) \\
O1 &= S4 \quad (\text{Open drain}) \\
O2 &= S3 \quad (\text{Start mixer})
\end{aligned}
\tag{4}
$$

d.) Converting the state equations to a PLC program requires the specification of the input and output points on the controller and the type of switch contact employed. Table 8 lists the I/O points used in this example.

Table-8 PLC I/O Point Assignments

Inputs		
I1	Start (N.O. contact)	X0
I2	Stop (N.C. contact)	X2
I3	Float Switch (N.C. contact)	X3
Outputs		
O0	Open Inlet	Y0
O1	Mixer Motor Start	Y1
O2	Open Outlet	Y2

The state equations produce the rungs of the ladder program with some modifications. The conditions of the specified external contacts (N.O.=normally open and N.C. normally closed)

will control the logic. A NOTed variable in the state equations my translate into a direct contact in the PLC program. The goal is to maintain the logical continuity of the rung not electrical continuity.

The state equation structure assumes that they are computed simultaneously. Remember that the PLC executes a program from top down and left to right so state equations are not updated in parallel. Adding intermediate state variables allow state equations to re-compute correctly. This example defines S0X, S1X, S2X, S3X, and S4X as the intermediate state variables. The state variable updates should appear after all state equations in the ladder program structure. Any timers or counter that depend on state variable changes follow the updates. The final section should be the output equations. Figures 10, 11, and 12 show the Do-More Designer ladder diagram program file developed for this example. Figures 10 and 11 show the ladder diagram for the five state equations. Logical OR functions in the state equations become parallel contacts in the ladder program while the AND functions become series contacts. The outputs are the intermediate variables discussed above.. These internal bits of Cn type where n is an integer

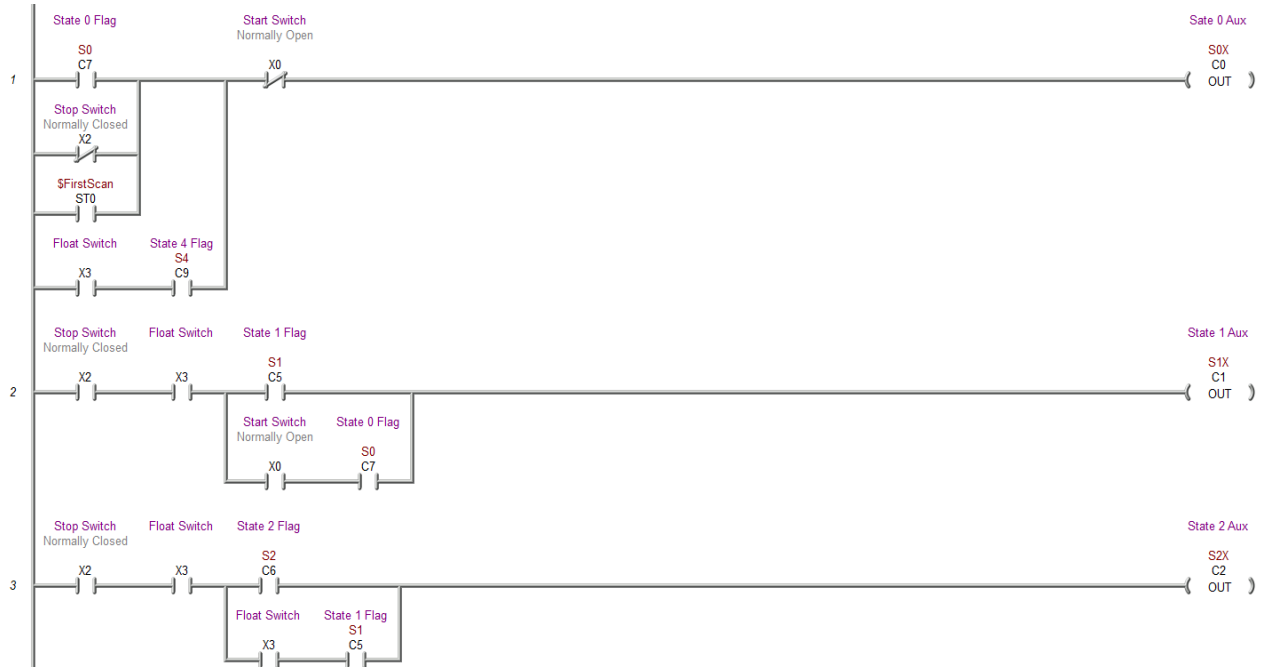

Figure 10. PLC Ladder Diagram for Example State Equations.

Figure 11 shows the section of ladder diagram program code that updates the state variables after their values are recomputed by the state equations shown above. These are very simple statements in which a NOC instruction associated with the intermediate variable changes the bit values of the OUT coil instructions whenever the intermediate variable changes. These rungs should appear after the state equations in a ladder program for proper operation.

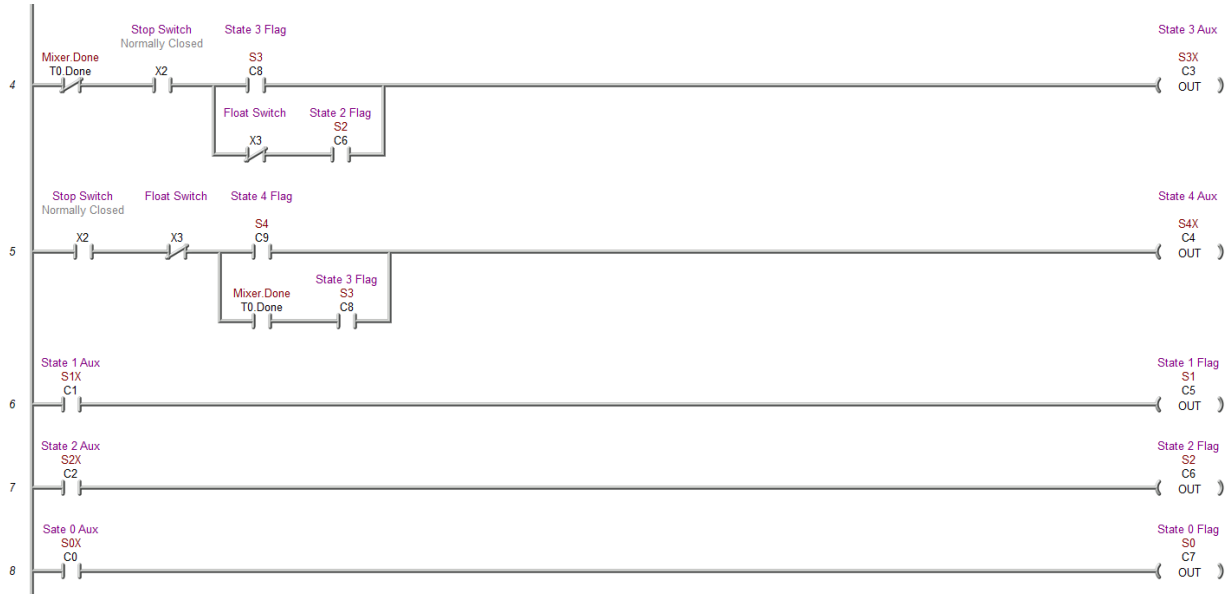

Figure 11. State Variable Updates For Example Problem.

Figure 12 shows the last section of the program. This section includes the timer function actuated by state variable S3 and the output equations. The output equations are very simple. They relate the state variables to the desired physical output points. The timer rung is the last rung in the program and is set to 10 seconds to test the program.

Figure 12. Output And Timer Equations For Example Problem.

PLC Programming Project-Sequential Control of Three Conveyors

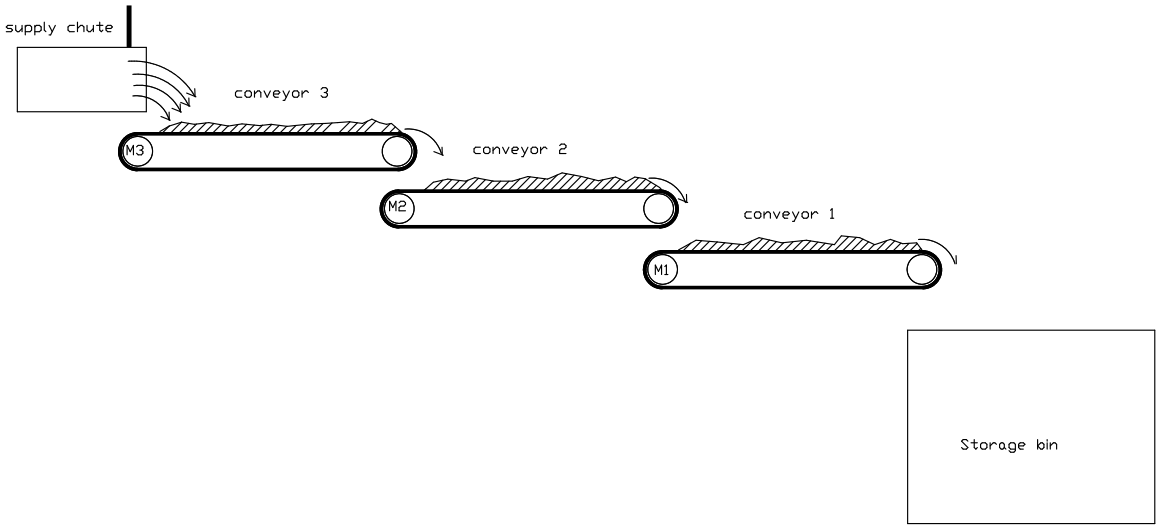

Figure 13. Sequential Conveyor Control

Program 1

Figure 13 above shows a three conveyor material transfer system. A supply chute delivers the material to conveyor 3 driven by motor M3. The flow of material is controlled by a gate that is actuated by a electro-pneumatic system. This conveyor transfers the material to conveyor 2 which is driven by motor M2. Finally the material is transferred to a storage bin by conveyor 1 driven by motor M1.

Design and code for the Do-More Designer simulator a control program for this system. A single on/off momentary-contact push-button station will control the whole process using the following procedure:

System Start Up

1.) pressing the start switch will immediately start motor M1. The motor must run for 10 seconds so the conveyor belt has time to reach operating speed. A red indicator lamp is energized when the start switch is depressed and remains on as long as the system operates.

2.) After the 10 second delay has ended, conveyor 2 starts and accelerates for 10 seconds also.

3.) Conveyor 3 is started after both conveyors 1 and 2 have been started. Finally, Conveyor 3 accelerates for 10 seconds.

4.) After all three conveyors are running the electro-pneumatic system is activated opening the material control gale allowing the material to fall onto the conveyors.

System Shut Down

1.) The stop push button is depressed.

2.) The flow of material though the gate is stopped by closing it.

3.) All motors must continue to run for 30 seconds to clear material from the conveyor belts. After this time delay all motors are de-energized and a green indicator lamp lights to show that the system is stopped. This light will remain on after the push button is released. It goes out when the start button is pressed.

Use the following I/O assignments on the trainer and develop a PLC program to implement this control.

X0 (N.O. switch) start	Y2 red run lamp
X1 (N.C. switch) stop	Y3 green stop lamp
ST0 ($FirstScan) =First Scan Bit	Y4 conveyor motor 3
	Y5 conveyor motor 2
	Y0 conveyor motor 1
	Y6 material gate open

Program 2

To save wear on the conveyor system, the above control is modified to shut down each conveyor in sequence as it empties. Tests show that conveyor 3 takes 7 seconds to clear, conveyor 2 takes 9 seconds to clear, and conveyor 1 takes 14 seconds to clear.

Modify the first program to cause the conveyors to shut down in the sequence 3-2-1 after the gate is closed with the time delays required to clear each belt.

Lab 7 Assessment

Complete and submit the following items for grading and perform the listed actions to complete this laboratory assignment.

1.) Complete the online quiz over Lab 7 technical background.
2.) Develop a state transition diagrams for programs 1 and 2
3.) Write Boolean state equations for programs 1 and 2
4.) Code programs 1 and 2 into the PLC trainer
5.) Demonstrate working programs to the lab TA
6.) Submit pdf files of the working programs, the developed state diagrams, and the state equations to the dropbox for lab 7