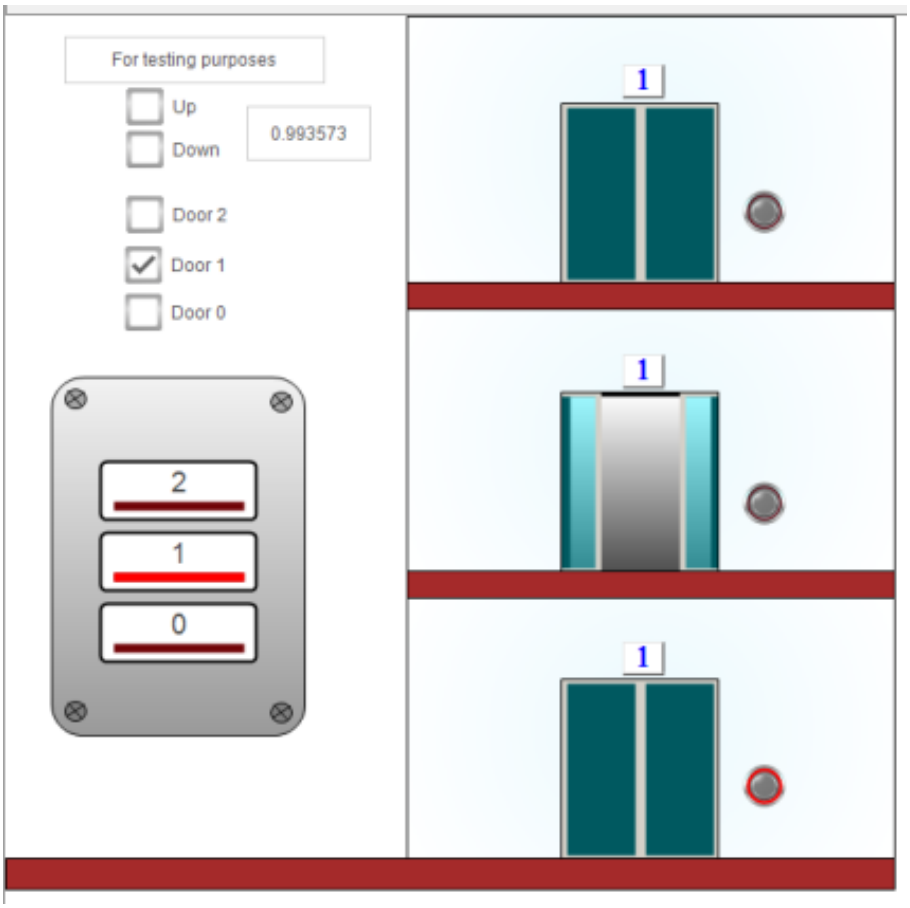
PLC Elevator



Basic Controller

Basic controller uses two nested state machines to operate the system. One state machine is operating the lift movement and the other is operating the doors, thus only together they can provide working solution for the lift problem.

First, looking the state machine for lift movement, which consists of three possible states: state_stopped, state_going_up and state_going_down. State_move is used to determinate the active state machine. State_stopped equals to the state when the lift is physically stopped to a floor. For achieving that, the state sets Boolean variables “up” and “down” as False, since they are for the movement of the lift. For the system, state_stopped is the initial state for the controller, and it has two possible predecessor states: state_going_up and state_going_down. This can be seen also from the figure 1. Active state is changed to state_going_up only if all doors has been closed (allClosed) and the next call is coming from upper floors than the lift current floor is (x>y). Also, there should be calls (x>=0) and the lift should be on a floor (y>=0). Other way around, active state is changed to state_going_down only if all doors has been closed (allClosed) and the next call is coming from lower floors than the lift current floor is (x>y). Also, like previous, there should be calls (x>=0) and the lift should be on a floor (y>=0).

Next, looking the state for upwards movement: state_going_up. This state initially sets variable “up” as true and “down” as false. So, as seen in figure 1, the lift is rising as long as the conditional variable (doStop) is false, meaning that the elevator is not at the same time on a floor, in which a call has been made. State_going_down is working in the same logic as state_going_up, but it operates the lift downwards. The state initially sets variable “up” as false and “down” as true. So, the lift is descending as long as the conditional variable (doStop) is false, meaning that the elevator is not at the same time on a floor, in which a call has been made.

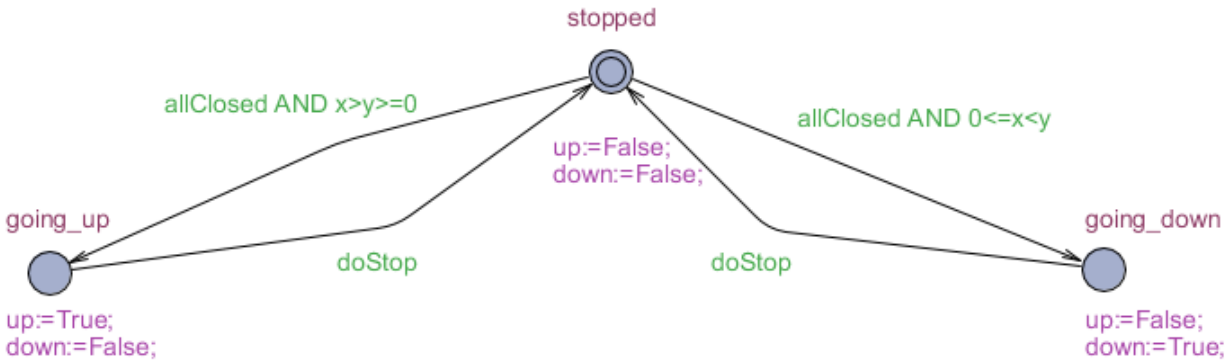


Figure 1: State machine diagram for cabin movement.

Moving on to the state machine for door operation. The state machine consists of six different states: state_closed, state_opening0, state_opening1, state_opening2, state_open and state_closing. State_door is used to describe the active state. The basic sequence for doors is that when doors are closed, one door starts opening and then fully open. After a while, the door starts closing and then all of them are fully closed. Taking a deeper engineering view, the initial state for the doors is state_closed. It represents a situation when all doors are closed.

As seen in figure 2, state_closed is changed to one of state_opening if there is a true condition (for either doOpen0, doOpen1 or doOpen2). DoOpen variables are true only when the lift is on the same floor as where a call has been made. All the three state_opening states opens the corresponding door and after that changes active state to state_open, because door is not closed (NOT doorClosed). State_open is a state for doors to be open so passengers can board. This implementation is using timer on-delay function to set the doors to stay open for three seconds. After the time period the active state changes to be state_closing. That state is closing all the doors and when that condition comes true (allClosed), state_closed is active again.

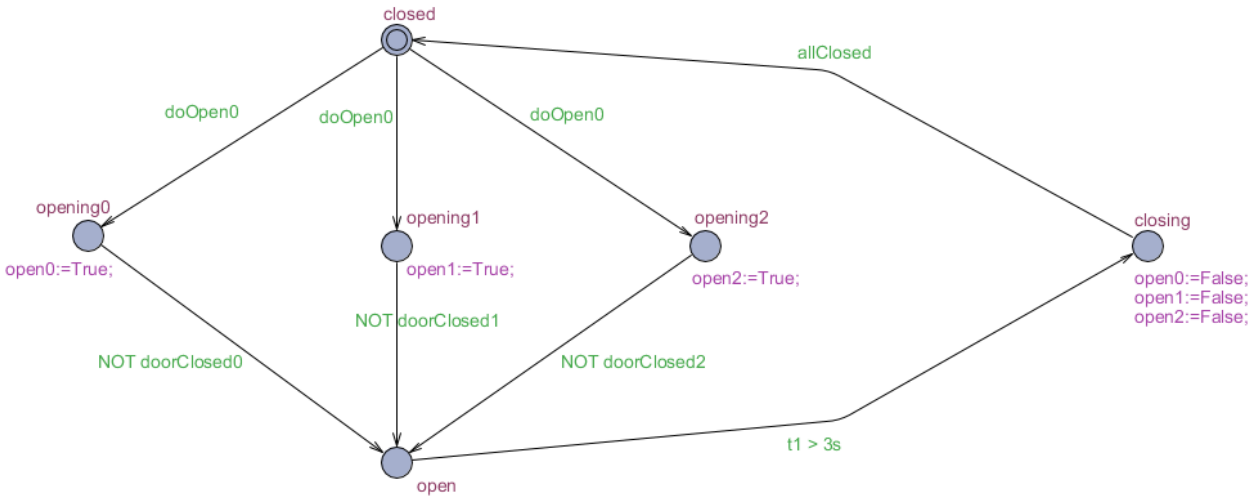


Figure 2: State machine diagram for movement of the doors.

Extended controller

The changes made for the basic controller were minor. A new Boolean variable “skipOne” was added for determining if there is calls that needs the skipping for a floor 1. Furthermore “doStop” needed small modifications. For the door control “doOpen{0…2}”, “keepOpen{0…2}”, and “keepOpen” where added to control the doors state machine transitions.

Appendix A: Code used in basic controller

```
PROGRAM SysFunc1

VAR
                t1: TON;
                allClosed: BOOL;
                y: INT;
                doStop: BOOL;

                doOpen0: BOOL;
                doOpen1: BOOL;
                doOpen2: BOOL;

                state_move: INT := 1;
                state_stopped: INT := 1;
                state_going_up: INT := 2;
                state_going_down: INT := 3;

                state_door:         INT := 1;
                state_closed:       INT := 1;
                state_opening_0: INT := 2;
                state_opening_1: INT := 3;
                state_opening_2: INT := 4;
                state_open:         INT := 5;
                state_closing:      INT := 6;
END_VAR

//Place your implementation of the state-based controller here
allClosed := doorclosed0 AND doorclosed1 AND doorclosed2;
doStop := (y=0 AND (button0 OR call0))
                                OR (y=1 AND (button1 OR call1))
                                OR (y=2 AND (button2 OR call2));

doOpen0 := (y=0 AND (button0 OR call0));
doOpen1         := (y=1 AND (button1 OR call1));
doOpen2 := (y=2 AND (button2 OR call2));
```

```
IF call0 OR button0 THEN
                x:=0;
ELSIF call1 OR button1 THEN
                x:=1;
ELSIF call2 OR button2 THEN
                x:=2;
ELSE
                x:=-2;
END_IF

IF onfloor0 THEN
                y:=0;
ELSIF onfloor1 THEN
                y:=1;
ELSIF onfloor2 THEN
                y:=2;
ELSE
                y:=-2;
END_IF

IF state_move = state_stopped THEN
                up:=FALSE;
                down:=FALSE;
                IF allClosed AND x < y AND x >= 0 THEN
                                state_move := state_going_down;
                ELSIF allClosed AND x > y AND x >= 0 THEN
                                state_move := state_going_up;
                END_IF
ELSIF state_move = state_going_up THEN
                up:=TRUE;
                down:=FALSE;
                IF doStop THEN
                                state_move := state_stopped;
                END_IF
ELSIF state_move = state_going_down THEN
                up:=FALSE;
```

```
                down:=TRUE;
                IF doStop THEN
                                state_move := state_stopped;
                END_IF
END_IF

IF state_door = state_closed THEN
                IF doOpen0 THEN
                                state_door := state_opening_0;
                ELSIF doOpen1 THEN
                                state_door := state_opening_1;
                ELSIF doOpen2 THEN
                                state_door := state_opening_2;
                END_IF

ELSIF state_door = state_opening_0 THEN
                open0 := TRUE;
                IF NOT doorClosed0 THEN
                                state_door := state_open;
                END_IF
ELSIF state_door = state_opening_1 THEN
                open1 := TRUE;
                IF NOT doorClosed1 THEN
                                state_door := state_open;
                END_IF
ELSIF state_door = state_opening_2 THEN
                open2 := TRUE;
                IF NOT doorClosed2 THEN
                                state_door := state_open;
                END_IF
ELSIF state_door = state_open THEN
                t1.IN := TRUE;
                t1.PT := T#3S;
                t1();
                IF t1.Q THEN
                                t1.IN := FALSE;
                                t1();
                                state_door := state_closing;
```

```
                END_IF

ELSIF state_door = state_closing THEN
                open0 := FALSE;
                open1 := FALSE;
                open2 := FALSE;
                IF allClosed THEN
                                state_door := state_closed;
                END_IF
END_IF
```

Appendix B: Code used in extended controller

```
PROGRAM Extended
VAR
                t1: TON;

                allClosed: BOOL;
                skipOne: BOOL;
                x: INT;
                y: INT;
                lastFloor: INT;
                doStop: BOOL;

                doOpen0: BOOL;
                doOpen1: BOOL;
                doOpen2: BOOL;
                keepOpen0: BOOL;
                keepOpen1: BOOL;
                keepOpen2: BOOL;
                keepOpen: BOOL;

                state_move: INT := 1;
                state_stopped: INT := 1;
                state_going_up: INT := 2;
                state_going_down: INT := 3;
                state_prio_going_up: INT := 4;
                state_prio_going_down: INT := 5;
```

```
                state_door:         INT := 1;
                state_closed:       INT := 1;
                state_opening_0: INT := 2;
                state_opening_1: INT := 3;
                state_opening_2: INT := 4;
                state_open:         INT := 5;
                state_closing:      INT := 6;
END_VAR

//Place your implementation of the state-based controller here
allClosed := doorclosed0 AND doorclosed1 AND doorclosed2;
skipOne := (call0 AND lastFloor > 1) OR (call2 AND lastFloor < 1);
doStop := (y=0 AND (button0 OR call0))
                                    OR (y=1 AND ((button1 AND NOT skipOne) OR call1))
                                    OR (y=2 AND (button2 OR call2));

doOpen0 := (y=0 AND (button0 OR call0));
doOpen1         := (y=1 AND ((button1 AND NOT skipOne) OR call1));
doOpen2 := (y=2 AND (button2 OR call2));
keepOpen0 := (y=0 AND (button0 OR call0));
keepOpen1 := (y=1 AND (button1 OR call1));
keepOpen2 := (y=2 AND (button2 OR call2));
keepOpen := keepOpen0 OR keepOpen1 OR keepOpen2;

IF call0 THEN
                x:=0;
ELSIF call1 THEN
                x:=1;
ELSIF call2 THEN
                x:=2;
ELSIF button0 THEN
                x:=0;
ELSIF button1 THEN
                x:=1;
ELSIF button2 THEN
                x:=2;
ELSE
```

```
                x:=-2;
END_IF



IF onfloor0 THEN
                y:=0;
                lastFloor:=0;
ELSIF onfloor1 THEN
                y:=1;
ELSIF onfloor2 THEN
                y:=2;
                lastFloor:=2;
ELSE
                y:=-2;
END_IF

IF state_door = state_closed THEN
                IF doOpen0 THEN
                                state_door := state_opening_0;
                ELSIF doOpen1 THEN
                                state_door := state_opening_1;
                ELSIF doOpen2 THEN
                                state_door := state_opening_2;
                END_IF

ELSIF state_door = state_opening_0 THEN
                open0 := TRUE;
                IF NOT doorClosed0 THEN
                                state_door := state_open;
                END_IF
ELSIF state_door = state_opening_1 THEN
                open1 := TRUE;
                IF NOT doorClosed1 THEN
                                state_door := state_open;
                END_IF
ELSIF state_door = state_opening_2 THEN
                open2 := TRUE;
                IF NOT doorClosed2 THEN
```

```
                    state_door := state_open;
        END_IF
ELSIF state_door = state_open THEN
        t1.IN := TRUE;
        t1.PT := T#3S;
        t1();
        IF t1.Q THEN
                    t1.IN := FALSE;
                    t1();
                    state_door := state_closing;
        END_IF

ELSIF state_door = state_closing THEN
        open0 := FALSE;
        open1 := FALSE;
        open2 := FALSE;
        IF allClosed OR keepOpen THEN
                    state_door := state_closed;
        END_IF
END_IF


IF state_move = state_stopped THEN
        up:=FALSE;
        down:=FALSE;
        IF allClosed AND x < y AND x >= 0 THEN
                    state_move := state_going_down;
        ELSIF allClosed AND x > y AND x >= 0 THEN
                    state_move := state_going_up;
        END_IF
ELSIF state_move = state_going_up THEN
        up:=TRUE;
        down:=FALSE;
        IF doStop THEN
                    state_move := state_stopped;
        END_IF
ELSIF state_move = state_going_down THEN
        up:=FALSE;
```

```
                down:=TRUE;
                IF doStop THEN
                                state_move := state_stopped;
                END_IF
END_IF
```