An Elevator Controller

solution

```
--------------------------------------------------------------------------
-- AN ELEVATOR CONTROLLER                                               --
--------------------------------------------------------------------------
-- This SMV program describes an elevator system for a 4-floors building.
-- It includes modules both for the physical system (reservation buttons,
-- cabin, door), and for the controller.

--------------------------------------------------------------------------
-- BUTTON                                                               --
--------------------------------------------------------------------------
-- For each floor there is a button to request service, that can be
-- pressed. A pressed button stays pressed unless reset by the
-- controller. A button that is not pressed can become pressed
-- nondeterministically.

MODULE Button(reset)
  VAR
    pressed : boolean;
  ASSIGN
    init(pressed) := 0;
    next(pressed) :=
      case
        pressed & reset  : 0;
        pressed & !reset : 1;
        !pressed         : {0,1};
      esac;

  -- REQ: The controller must not resets a button that is not pressed.
  INVARSPEC (reset -> pressed)

--------------------------------------------------------------------------
-- CABIN                                                                --
--------------------------------------------------------------------------
-- The cabin can be at any floor between 1 and 4. It is equipped with an
-- engine that has a direction of motion, that can be either standing, up
-- or down. The engine can receive one of the following commands: nop, in
-- which case it does not change status; stop, in which case it becomes
-- standing; up (down), in which case it goes up (down).

MODULE Cabin(move_cmd)
  VAR
    floor     : { 1,2,3,4 };
    direction : { standing, moving_up, moving_down };
```

```
  ASSIGN
    init(direction) := standing;
    next(direction) :=
      case
        move_cmd = stop      : standing;
        move_cmd = move_up   : moving_up;
        move_cmd = move_down : moving_down;
        move_cmd = nop       : direction;
      esac;

    next(floor) :=
      case
        next(direction) = standing    : floor;
        next(direction) = moving_up   : case
                                          floor = 4       : 4;
                                          1               : floor + 1;
                                        esac;
        next(direction) = moving_down : case
                                          floor = 1       : 1;
                                          1               : floor - 1;
                                        esac;
      esac;

  -- REQ: The controller can issue a stop command only if the direction
  --      is up or down.
  INVARSPEC (move_cmd = stop  ->  direction in {moving_up,moving_down})

  -- REQ: The controller can issue a move command only if the
  --      direction is standing.
  INVARSPEC (move_cmd in {move_up,move_down}  ->  direction = standing)

  -- REQ: The cabin can move up only if the floor is not 4.
  SPEC AG (floor = 4 -> AX(direction != moving_up))

  -- REQ: The cabin can move down only if the floor is not 1.
  SPEC AG (floor = 1 -> AX(direction != moving_down))

-------------------------------------------------------------------------
-- DOOR                                                                --
-------------------------------------------------------------------------
-- The cabin is also equipped with a door, that can be either open
-- or closed. The door can receive either open, close or nop commands
-- from the controller, and it responds opening, closing, or
-- preserving the current state.

MODULE Door(door_cmd)
  VAR
    status : { open, closed };

  ASSIGN
    next(status) :=
      case
```

```
        door_cmd = open      : open;
        door_cmd = close     : closed;
        door_cmd = nop       : status;
      esac;

  -- REQ: The controller can issue an open command only if the door is closed.
  INVARSPEC (door_cmd = open   ->  status = closed)

  -- REQ: The controller can issue a close command only if the door is open.
  INVARSPEC (door_cmd = close  ->  status = open)

-----------------------------------------------------------------------------
-- CONTROLLER                                                              --
-----------------------------------------------------------------------------
-- The controller takes in input (as sensory signals) the floor and the
-- direction of motion of the cabin, the status of the door, and the
-- status of the four buttons. It decides the controls to the engine, to
-- the door and to the buttons.

MODULE CTRL(floor, dir, door, pressed_1, pressed_2, pressed_3, pressed_4)
  VAR
    move_cmd : {move_up, move_down, stop, nop};
    door_cmd : {open, close, nop};
    reset_1 : boolean;
    reset_2 : boolean;
    reset_3 : boolean;
    reset_4 : boolean;

  -- Button N is reset only if it is pressed, we are at floor N, and
  -- the door is open.
  ASSIGN
    reset_1 := (pressed_1 & floor = 1 & door = open);
    reset_2 := (pressed_2 & floor = 2 & door = open);
    reset_3 := (pressed_3 & floor = 3 & door = open);
    reset_4 := (pressed_4 & floor = 4 & door = open);

  -- Check whether there are pending requests at the current floor,
  -- at a higher floor, and at a lower floor.
  DEFINE
    pending_here := (floor = 1 & pressed_1) | (floor = 2 & pressed_2) |
                    (floor = 3 & pressed_3) | (floor = 4 & pressed_4) ;

    pending_up   := (floor = 1 & ( pressed_2 | pressed_3 | pressed_4 )) |
                    (floor = 2 & (             pressed_3 | pressed_4 )) |
                    (floor = 3 & (                         pressed_4 )) ;

    pending_down := (floor = 4 & ( pressed_1 | pressed_2 | pressed_3 )) |
                    (floor = 3 & ( pressed_1 | pressed_2             )) |
                    (floor = 2 & ( pressed_1                        )) ;

  -- * If the cabin is moving, do not send commands to the door.
  -- * If there is a pending request at the current floor and
  --   the door is closed, open it.
```

```
-- * If there are pending requests at different floors and the
--   door is open, close it.
-- * Otherwise, do not send commands to the door.
ASSIGN
  door_cmd :=
    case
      dir != standing              : nop;
      pending_here & door = closed : open;
      pending_up   & door = open   : close;
      pending_down & door = open   : close;
      1                            : nop;
    esac;

-- Variable "last_dir" records the last movement direction of the cabin.
VAR
  last_dir : {moving_up,moving_down};
ASSIGN
  next(last_dir) :=
    case
      dir = standing : last_dir;
      1              : dir;
    esac;

-- * If the door is open, do not send move commands to the cabin.
-- * If there is a pending request at the current floor
--   and the cabin is moving, stop it.
-- * If there are pending requests both at higher and at lower floors,
--   keep moving in "last_dir".
-- * If there are pending requests at higher (lower) floors,
--   move up (down).
-- * Otherwise, do not send commands to the cabin.
ASSIGN
  move_cmd :=
    case
      door = open               : nop;
      pending_here              : case
                                    dir != standing : stop;
                                    1               : nop;
                                  esac;
      pending_up & pending_down : case
                                    dir != standing        : nop;
                                    last_dir = moving_up   : move_up;
                                    last_dir = moving_down : move_down;
                                  esac;
      pending_up                : case
                                    dir != standing      : nop;
                                    1                    : move_up;
                                  esac;
      pending_down              : case
                                    dir != standing      : nop;
                                    1                    : move_down;
                                  esac;
      1                         : nop;
```

```
      esac;

-----------------------------------------------------------------------------
-- MAIN                                                                    --
-----------------------------------------------------------------------------
-- The main module shows the connection between modules.

MODULE main
  VAR
    cabin : Cabin(ctrl.move_cmd);
    door  : Door(ctrl.door_cmd);
    button_1 : Button(ctrl.reset_1);
    button_2 : Button(ctrl.reset_2);
    button_3 : Button(ctrl.reset_3);
    button_4 : Button(ctrl.reset_4);
    ctrl : CTRL(cabin.floor, cabin.direction, door.status,
                button_1.pressed, button_2.pressed,
                button_3.pressed, button_4.pressed);

-----------------------------------------------------------------------------
-- REQUIREMENTS                                                            --
-----------------------------------------------------------------------------
-- The controller must satisfy the following requirements.

-- REQ: No button can reach a state where it remains pressed forever.
  SPEC AG AF ! button_1.pressed
  SPEC AG AF ! button_2.pressed
  SPEC AG AF ! button_3.pressed
  SPEC AG AF ! button_4.pressed

-- REQ: No pressed button can be reset until the cabin stops at the
--      corresponding floor and opens the door.
  SPEC AG (button_1.pressed ->
    A [button_1.pressed U (cabin.floor = 1 & door.status = open)])
  SPEC AG (button_2.pressed ->
    A [button_2.pressed U (cabin.floor = 2 & door.status = open)])
  SPEC AG (button_3.pressed ->
    A [button_3.pressed U (cabin.floor = 3 & door.status = open)])
  SPEC AG (button_4.pressed ->
    A [button_4.pressed U (cabin.floor = 4 & door.status = open)])

-- REQ: A button must be reset as soon as the cabin stops at the
--      corresponding floor with the door open.
  SPEC AG ((button_1.pressed & cabin.floor = 1 & door.status = open) ->
    AX ! button_1.pressed)
  SPEC AG ((button_2.pressed & cabin.floor = 2 & door.status = open) ->
    AX ! button_2.pressed)
  SPEC AG ((button_3.pressed & cabin.floor = 3 & door.status = open) ->
    AX ! button_3.pressed)
  SPEC AG ((button_4.pressed & cabin.floor = 4 & door.status = open) ->
    AX ! button_4.pressed)

-- REQ: The cabin can move only when the door is closed.
```

```
  INVARSPEC (door.status = open -> cabin.direction = standing)

-- REQ: If no button is pressed, the controller must issue no commands
--      and the cabin must be standing.
  INVARSPEC ((((! button_1.pressed) & (! button_2.pressed) &
              (! button_3.pressed) & (! button_4.pressed))
                   -> (ctrl.door_cmd = nop & ctrl.move_cmd = nop))

-------------------------------------------------------------------------
```