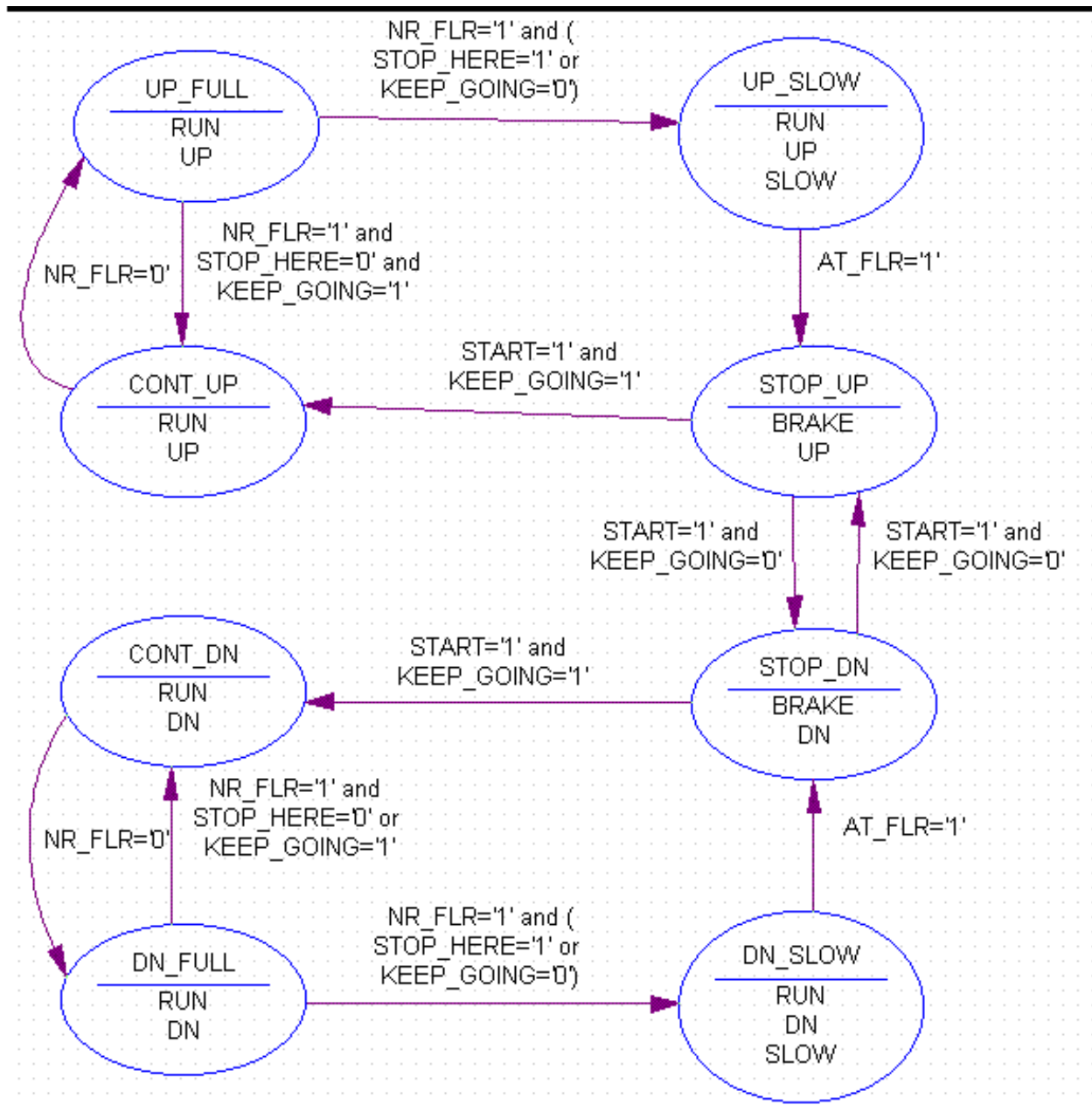


## ELEVATOR DESIGN THE MAIN CONTROLLER

The main controller for the elevator was shown in block diagram form previously. We will now look inside it. We begin with a state diagram (courtesy of StateCAD).



We begin designing (or, now, analyzing) the diagram with the UP\_FULL state where the elevator is between floors and traveling up. Why here? You can start anywhere you wish but this seemed to be the easiest place to start since nothing has to happen until it nears a floor.

When it does near a floor, if it is supposed to continue up without stopping, we go to the CONT\_UP state and continue going up. We return to the UP\_FULL state when we are no longer “near” the floor (see below). If we are supposed to stop (STOP\_HERE) we move to the UP\_SLOW state when we near a floor. The other condition was explained previously but will be noted again. If neither KEEP\_GOING nor STOP\_HERE is true, it has to be because someone pushed a DOWN button at the floor so we will have to stop and the change directions. In either case, we slow the car down so that we can put on the brakes safely when AT\_FLR is true.

The elevator stops. We will assume that the CAR Controller then opens the door, allows people to enter and leave and then closes the door and issues a START command. If the car is to continue up, we proceed to the CONT\_UP state and then UP\_FULL.

If there are no requests for higher floors, when START is asserted we move from the STOP\_UP state to the STOP\_DN state. These states are identical except that the DN (GOING\_DN) signal is asserted rather than the UP (GOING\_UP) signal. This change will cause a change in the controls from the floors and the car since they sense a change of direction. If KEEP\_GOING is true if the STOP\_DN state, it means to keep (start) going down. We thus move to the CONT\_DN and DN\_FULL states.

If no buttons have been pressed, the controller will alternate between STUP\_UP and STOP\_DN until one is pressed.

The remainder of the diagram needs no further description since the bottom half is a mirror image of the top half - only the direction is changed.

Two signals are used by the controller which have not yet been defined: NR\_FLR (Near Floor) and AT\_FLR). These are derived from a “FRONT END” for the controller which conditions the signals obtained from the elevator shaft. The VHDL for this is below.

```

library ieee;
use ieee.std_logic_1164.all;
entity shaft is port(
    f:          in    std_logic_vector(4 downto 0);
    here:       in    std_logic;
    stb:        in    std_logic;
    clk:        in    std_logic;
    flr:        out   std_logic_vector(4 downto 0);
    at_flr, nr_flr: out   std_logic);
end shaft;

architecture shaft_logic of shaft is
begin
    process(stb, f, clk, here) begin
        if (stb'event and stb='0') then flr<=f; end if;
-- NOTE THAT THE FLR FLIP-FLOPS ARE CLOCKED BY A NEGATIVE
-- TRANSITION OF STB. WITH A PAL YOU CAN HAVE ONLY ONE,
-- POSITIVE EDGE-TRIGGERED CLOCK. WITH CPLDS YOU MAY HAVE
-- SEVERAL CLOCKS.
        if (clk'event and clk='1') then
            at_flr<=here;
-- AT_FLR IS JUST A SYNCHRONIZED VERSION OF “HERE”
    end process;
end shaft_logic;

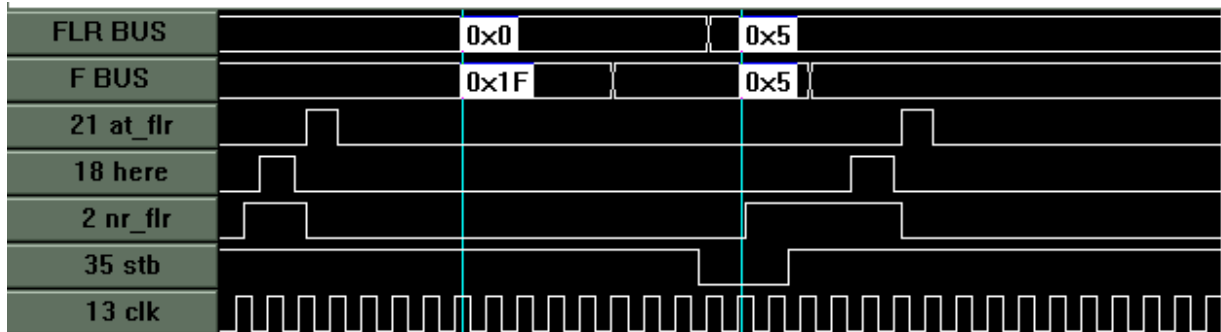
```

```

        if stb='0' then nr_flr<='1';
        elsif here='1' then nr_flr<='0';
        else here<=here; end if;
-- THE SIGNAL nr_flr IS SET WHEN THE ELEVATOR NEARS A FLOOR (AS
-- DETERMINED BY THE STB SIGNAL) AND IS CLEARED WHEN IT REACHES
-- THE FLOOR (AS INDICATED BY THE "HERE" SIGNAL).
        end if;
    end process;
end shaft_logic;

```

The above module is compiled to the work library. It provides the two new signals needed as well as the registered FLR bus. The operation is demonstrated by the simulation below.



The F BUS is all high (hex 1F except when nearing a floor. Here it nears floor five. The STB (LOW) pulse strobes this value onto the FLR bus. The STB also sets NR\_FLR. When the elevator reaches the floor, the HERE signal is synchronized as the AT\_FLR signal and NR\_FLR is cleared.

The VHDL code for the main controller was generated by StateCAD and modified as noted below.

```

-- F:\PROJS319\ELEVATOR\UPDN.VHD
-- VHDL code created by Visual Software Solution's StateCAD Version 3.2
-- Sun Jan 18 12:27:38 1998
-- This VHDL code (for use with IEEE compliant tools) was generated using:
-- binary encoded state assignment with structured code format.
-- Minimization is enabled, implied else is enabled,
-- and outputs are area optimized.

```

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.shaft;
ENTITY UPDN IS
    PORT (CLK, KEEP_GOING, START, STOP_HERE: IN std_logic;
          F: IN std_logic_vector(4 downto 0);
          HERE, STB: in std_logic;
          FLR: OUT std_logic_vector(4 downto 0);
          BRAKE, DN, RUN, SLOW, UP : OUT std_logic);
END;

```

```

ARCHITECTURE BEHAVIOR OF UPDN IS
    SIGNAL sreg : std_logic_vector (2 DOWNT0 0);
    SIGNAL next_sreg : std_logic_vector (2 DOWNT0 0);
    CONSTANT CONT_DN : std_logic_vector (2 DOWNT0 0) :="010";
    CONSTANT CONT_UP : std_logic_vector (2 DOWNT0 0) :="110";

```

```

CONSTANT DN_FULL : std_logic_vector (2 DOWNTO 0) :="011";
CONSTANT DN_SLOW : std_logic_vector (2 DOWNTO 0) :="001";
CONSTANT STOP_DN : std_logic_vector (2 DOWNTO 0) :="000";
CONSTANT STOP_UP : std_logic_vector (2 DOWNTO 0) :="100";
CONSTANT UP_FULL : std_logic_vector (2 DOWNTO 0) :="111";
CONSTANT UP_SLOW : std_logic_vector (2 DOWNTO 0) :="101";
SIGNAL NR_FLR, AT_FLR: std_logic;

BEGIN
FRONTEND: shaft PORT MAP (F, HERE, STB, CLK, FLR, AT_FLR, NR_FLR);
PROCESS (CLK, next_sreg)
BEGIN
    IF CLK=' 1' AND CLK' event THEN
        sreg <= next_sreg;
    END IF;
END PROCESS;

PROCESS (sreg, AT_FLR, KEEP_GOING, NR_FLR, START, STOP_HERE)
BEGIN
    BRAKE <= ' 0'; DN <= ' 0'; RUN <= ' 0'; SLOW <= ' 0'; UP <= ' 0';

    next_sreg<=CONT_DN;

    CASE sreg IS
        WHEN CONT_DN =>
            UP<=' 0';
            SLOW<=' 0';
            BRAKE<=' 0';
            RUN<=' 1';
            DN<=' 1';
            IF ( NR_FLR=' 0' ) THEN
                next_sreg<=DN_FULL;
            ELSE
                next_sreg<=CONT_DN;
            END IF;
        WHEN CONT_UP =>
            SLOW<=' 0';
            DN<=' 0';
            BRAKE<=' 0';
            RUN<=' 1';
            UP<=' 1';
            IF ( NR_FLR=' 0' ) THEN
                next_sreg<=UP_FULL;
            ELSE
                next_sreg<=CONT_UP;
            END IF;
        WHEN DN_FULL =>
            UP<=' 0';
            SLOW<=' 0';
            BRAKE<=' 0';
            RUN<=' 1';
            DN<=' 1';
            IF ( NR_FLR=' 0' ) THEN
                next_sreg<=DN_FULL;
            END IF;
            IF ( KEEP_GOING=' 0' AND NR_FLR=' 1' ) OR
                ( STOP_HERE=' 1' AND NR_FLR=' 1' )
            THEN
                next_sreg<=DN_SLOW;
            END IF;
            IF ( NR_FLR=' 1' AND STOP_HERE=' 0' AND
                KEEP_GOING=' 1' ) THEN
                next_sreg<=CONT_DN;
            END IF;
        WHEN DN_SLOW =>

```

```

UP<=' 0' ;
BRAKE<=' 0' ;
RUN<=' 1' ;
DN<=' 1' ;
SLOW<=' 1' ;
IF ( AT_FLR=' 1' ) THEN
    next_sreg<=STOP_DN;
ELSE
    next_sreg<=DN_SLOW;
END IF;
WHEN STOP_DN =>
    UP<=' 0' ;
    SLOW<=' 0' ;
    RUN<=' 0' ;
    BRAKE<=' 1' ;
    DN<=' 1' ;
    IF ( START=' 0' ) THEN
        next_sreg<=STOP_DN;
    END IF;
    IF ( START=' 1' AND KEEP_GOING=' 1' ) THEN
        next_sreg<=CONT_DN;
    END IF;
    IF ( START=' 1' AND KEEP_GOING=' 0' ) THEN
        next_sreg<=STOP_UP;
    END IF;
WHEN STOP_UP =>
    SLOW<=' 0' ;
    RUN<=' 0' ;
    DN<=' 0' ;
    BRAKE<=' 1' ;
    UP<=' 1' ;
    IF ( START=' 0' ) THEN
        next_sreg<=STOP_UP;
    END IF;
    IF ( START=' 1' AND KEEP_GOING=' 1' ) THEN
        next_sreg<=CONT_UP;
    END IF;
    IF ( START=' 1' AND KEEP_GOING=' 0' ) THEN
        next_sreg<=STOP_DN;
    END IF;
WHEN UP_FULL =>
    SLOW<=' 0' ;
    DN<=' 0' ;
    BRAKE<=' 0' ;
    RUN<=' 1' ;
    UP<=' 1' ;
    IF ( NR_FLR=' 0' ) THEN
        next_sreg<=UP_FULL;
    END IF;
    IF ( KEEP_GOING=' 0' AND NR_FLR=' 1' ) OR
        ( STOP_HERE=' 1' AND NR_FLR=' 1' )
    THEN
        next_sreg<=UP_SLOW;
    END IF;
    IF ( NR_FLR=' 1' AND STOP_HERE=' 0' AND
        KEEP_GOING=' 1' ) THEN
        next_sreg<=CONT_UP;
    END IF;
WHEN UP_SLOW =>
    DN<=' 0' ;
    BRAKE<=' 0' ;
    RUN<=' 1' ;
    UP<=' 1' ;
    SLOW<=' 1' ;

```

```

        IF ( AT_FLR=' 1' ) THEN
            next_sreg<=STOP_UP;
        ELSE
            next_sreg<=UP_SLOW;
        END IF;
    WHEN OTHERS =>
        END CASE;
    END PROCESS;
END BEHAVIOR;

```

The changes and additions have been underscores and put in bold type. The state assignments were changed to make for a slightly more efficient design (more on this later). The “front end module”, called “shaft” was added as an entity using the “PORT MAP” statement. The underscore inputs and outputs were added. The NR\_FLR and AT\_FLR signals were moved from the port list to internal signals.

The resulting design equations are:

```

up      = /dn. Q
brake   = /sreg_1. Q * /sreg_0. Q
slow    = /sreg_1. Q * sreg_0. Q

flr_4. D = f_4
flr_3. D = f_3
flr_2. D = f_2
flr_1. D = f_1
flr_0. D = f_0
flr_n. C = /stb (applies to all five signals)

at_flr. D = here
nr_flr. D = /here * /stb

/run = /sreg_1. Q * /sreg_0. Q
/dn. D = /sreg_1. Q * /sreg_0. Q * /keep_going * start * /up. CMB
        + /start * up. CMB + keep_going * up. CMB + sreg_0. Q * up. CMB
        + sreg_1. Q * up. CMB

sreg_1. D = /sreg_0. Q * keep_going * start
        + sreg_1. Q * keep_going * /stop_here
        + sreg_1. Q * /nr_flr. Q
        + sreg_1. Q * /sreg_0. Q
sreg_0. D = /sreg_1. Q * sreg_0. Q * /at_flr. Q
        + sreg_1. Q * sreg_0. Q * stop_here
        + sreg_1. Q * sreg_0. Q * /keep_going
        + sreg_1. Q * /nr_flr. Q

```

Note that dn.D is an alias of sreg2.D. Note also that FLR flip-flops are clocked by not STB.

The pin assignment is:

01	:	GND	12	:	GND	23	:	GND	34	:	GND
02	:	dn	13	:	clk	24	:	(nr_flr)	35	:	stb
03	:	Not Used	14	:	(sreg_1)	25	:	Not Used	36	:	flr_0
04	:	Not Used	15	:	brake	26	:	Not Used	37	:	flr_3
05	:	Not Used	16	:	run	27	:	Not Used	38	:	flr_1
06	:	Not Used	17	:	Not Used	28	:	Not Used	39	:	Not Used
07	:	Not Used	18	:	here	29	:	Not Used	40	:	f_4
08	:	slow	19	:	f_3	30	:	flr_2	41	:	f_2
09	:	up	20	:	f_1	31	:	flr_4	42	:	f_0
10	:	stop_here	21	:	(sreg_0)	32	:	start	43	:	(at_flr)
11	:	VPP	22	:	VCC	33	:	keep_going	44	:	VCC

The utilization statistics are:

Descripti on	Used	Max
Dedi cated Inputs	3	3
Clock/Inputs	2	2
I/O Macrocells	20	32
	25 /	37 = 67 %

	Requi red	Max (Avai lable)
CLOCK/LATCH ENABLE signals	2	2
Input REG/LATCH signals	0	3
Input PIN signals	3	3
Input PINs using I/O cells	6	6
Output PIN signals	14	26
Total PIN signals	25	37
Macrocells Used	14	32
Uni que Product Terms	22	160

The device is not very fully utilized.