

TEST UNITS

These notes are a continuation of the notes on testing. Here we will consider how the process may be automated. To begin, let us tabulate the tests that we need to make

RST	HECK	KEY	MOT	BRK	Comments
1	0(-)	(-)	(-)	(-)	Reset to IDLE state
0	0	0	0	0	In IDLE, goto RUN_FAST
0	1	0	1	0	In RUN_FAST with MOTOR, goto STOP
0	0	0	0	1	In STOP with BRAKES, goto IDLE
0	1	0	0	0	In IDLE, goto START
0	1	1	0	0	In START with KEY_ON, stay here
0	0	1	0	0	In START with KEY_ON, goto RUN_SLOW
0	0(-)	0	1	0	In RUN_SLOW with MOTOR, goto RUN_F
0	0	0	1	0	In RUN_FAST with MOTOR, goto RUN_S
0	0(-)	0	1	0	In RUN_SLOW with MOTOR, goto RUN_F
0	1	0	1	0	In RUN_FAST with MOTOR, goto STOP
0	1	0	0	1	In STOP with BRAKES, goto START
0	0(-)	1	0	0	In START with KEY_ON, end of test.

We may write a VHDL module to create a FSM to test this machine as shown below.

```

library ieee;
use ieee.std_logic_1164.all;
entity tester is port(
-- Note that the inputs to this machine are outputs of the other
    KEY_ON, MOTOR, BRAKES, CLK, RST: in std_logic;
-- and vice-versa
    RESET, HECK, ERR_OUT: out std_logic);
end tester;

architecture testunit of tester is
    type test_vector_array is array(0 to 12)
        of std_logic_vector(4 downto 0);
-- The elements of the test vector (4 downto 0) correspond to:
-- RESET, HECK, KEY_ON, MOTOR, and BRAKES.
-- There is one array element for each test

    constant tva: test_vector_array := (

```

```

--RH & KMB These headings help identify the constants below.
"10"&"000", --reset machine
"00"&"000", --to state run_fast
"01"&"010", --to stop, check motor
"00"&"001", --to idle, check brake
"01"&"000", --to start
"01"&"100", --remain at start, test key_on
"00"&"100", --to run_slow, test key-on
"00"&"010", --to run_fast, test key_on
"00"&"010", --return to run_slow, test motor
"00"&"010", --return to run_fast, test motor
"01"&"010", --to stop, test motor
"01"&"001", --to start, test key_on
"10"&"100");--test key_on
-- This completes the "test vectors" (inputs and outputs) for the machine.

signal i:integer(0 to 15); -- This is a counter to sequence the test
begin
  process(rst, i, clk, tva, key_on, motor, brakes) begin
    if CLK'event and CLK='1' then
-- perform the test
      RESET<=tva(I)(4); -- Issue RESET if bit 4 is set
      HECK <=tva(I)(3); -- Issue HECK is bit 3 is set
      if (tva(i)(2) /= KEY_ON) -- Check if KEY_ON equals bit 2
      or (tva(i)(1) /= MOTOR) -- and if MOTOR equals bit 1
      or (tva(i)(0) /= BRAKES) -- and if BRAKES equals bit 0.
      then ERR0UT<='1'; else ERR0UT<='0'; -- If not, flag ERROR!
      end if;
-- The rest is for incrementing to the next test.
      if rst='1' then i<=0; elsif i<12 then i<=i+1; else i<=0;
      end if;
    end if;
  end process;
end testunit;

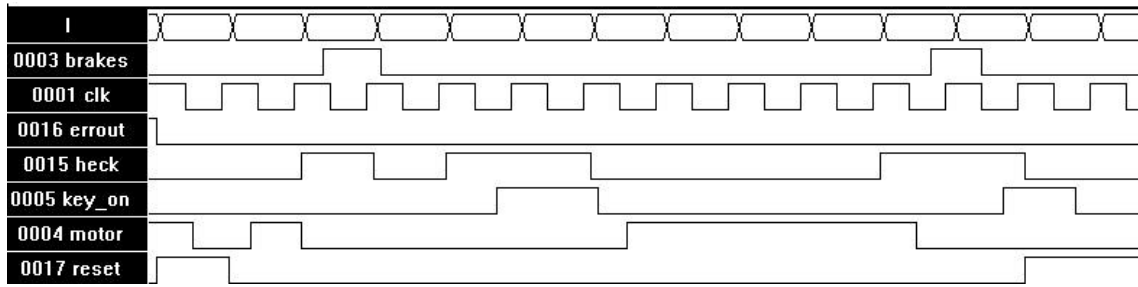
```

When this is compiled into a 22V10, the following usage is reported.

Information: Macrocell Utilization.

Description	Used	Max
Dedicated Inputs	4	11
Clock/Inputs	1	1
I/O Macrocells	7	10
12 / 22 = 54 %		

So, the entire tester will fit into a 22V10. A simulation of the results is shown in the figures below.



The above shows the test of a good machine. ERRROUT is low (after the initial reset) for the entire test. The diagram below shows a machine in which KEY_ON is stuck low. The ERRROUT signal is asserted for the three tests where KEY_ON should be asserted.

