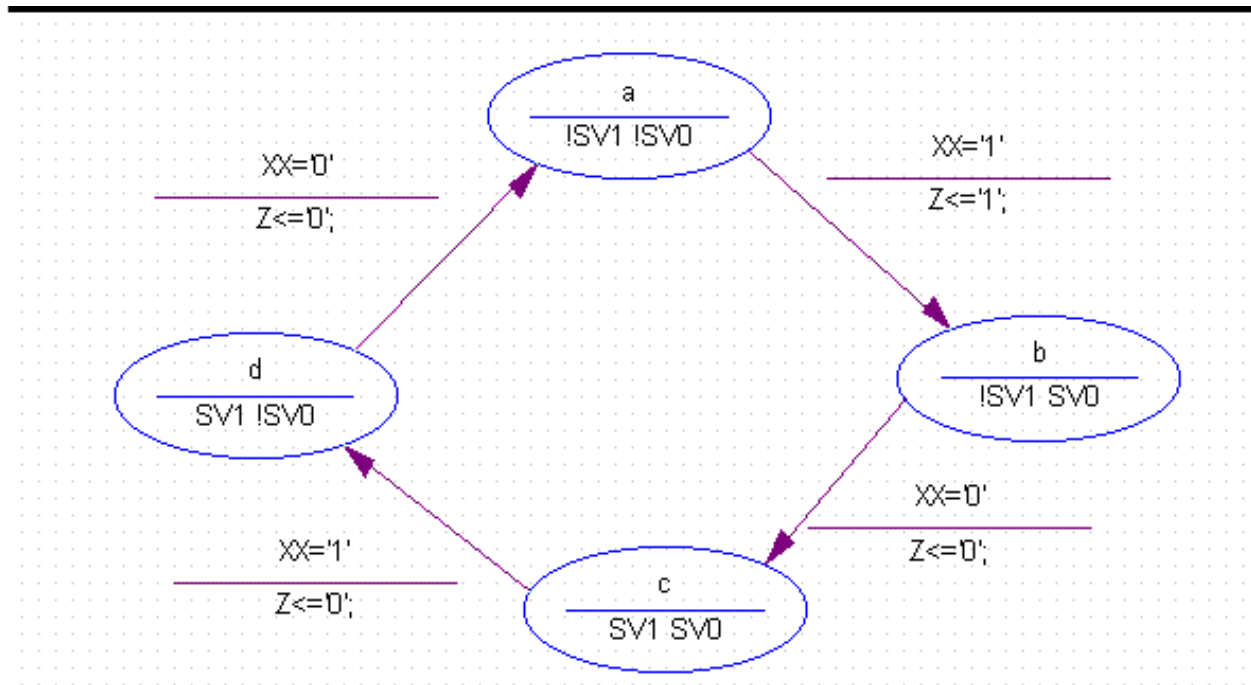


THE MECHANICS OF ASYNCHRONOUS DESIGN AND STATECAD AND VHDL

This set of notes does not deal with the concepts, theory or design of asynchronous state machines. It covers only how the tools we have been using may be used for this type of machine as well. Actually, VHDL can do any kind of machine. What we are primarily looking at is StateCAD which is designed to do only synchronous machines.

Below is the StateCAD representation of the machine shown on page 596 of Sandige's text (it was drawn flipped over, by mistake). It looks like all of the other synchronous state machines we have designed because StateCAD thinks it is synchronous. One thing that is different here is that the STATE VARIABLES have been assigned in each state. Unlike outputs which are just listed in the states where they are asserted, state variables (if listed at all) must be listed in every state. If they are zero, a ! is placed in front of the name. This is STILL a synchronous machine.



When StateCAD produces a VHDL description, it looks like the other descriptions. It is shown below. Several parts have been placed in bold type and underscored.

```
-- F: \ASYNCH\ASYNCH. VHD
-- VHDL code created by Visual Software Solution's StateCAD Version 3.2
-- Fri Oct 24 10:11:14 1997

-- This VHDL code (for use with IEEE compliant tools) was generated using:
-- manual 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;
ENTITY ASYNCH IS
    PORT (CLK, XX: IN std_logic;
          Z : OUT std_logic);
END;

ARCHITECTURE BEHAVIOR OF ASYNCH IS
    SIGNAL sreg : std_logic_vector (1 DOWNTO 0);
    SIGNAL next_sreg : std_logic_vector (1 DOWNTO 0);
    CONSTANT a : std_logic_vector (1 DOWNTO 0) := "00";
    CONSTANT b : std_logic_vector (1 DOWNTO 0) := "10";
    CONSTANT c : std_logic_vector (1 DOWNTO 0) := "11";
    CONSTANT d : std_logic_vector (1 DOWNTO 0) := "01";
BEGIN
    PROCESS (CLK, next_sreg)
    BEGIN
        IF CLK='1' AND CLK'event THEN
            sreg <= next_sreg;
        END IF;
    END PROCESS;
    PROCESS (sreg, XX)
    BEGIN
        Z <= '0';
        next_sreg<=a;
        CASE sreg IS
            WHEN a =>
                IF ( XX='1' ) THEN
                    next_sreg<=b;
                    Z<='1';
                ELSE
                    next_sreg<=a;
                    Z<='0';
                END IF;
            WHEN b =>
                IF ( XX='0' ) THEN
                    next_sreg<=c;
                    Z<='0';
                ELSE
                    next_sreg<=b;
                    Z<='0';
                END IF;
            WHEN c =>
                IF ( XX='1' ) THEN
                    next_sreg<=d;
                    Z<='0';
                ELSE
                    next_sreg<=c;
                    Z<='0';
                END IF;
            WHEN d =>
                IF ( XX='0' ) THEN
                    next_sreg<=a;
                    Z<='0';
                ELSE
                    next_sreg<=d;
                    Z<='0';
                END IF;
            WHEN OTHERS =>
                END CASE;
        END PROCESS;
    END BEHAVIOR;

```

If this is an asynchronous machine, CLK should not be listed as an input. We may edit it out of the PORT statement. The other thing that is wrong is the first process listed.

```

PROCESS (CLK, next_sreg)
BEGIN
IF CLK='1' AND CLK'event THEN
    sreg <= next_sreg;
END IF;
END PROCESS;

```

The machine can't change states on the rising edge of the clock is there is none. To take care of this, just delete the parts that are crossed out above. This just leaves the process setting sreg to the value of next_sreg.

This will work fine. If you are a purist, there is a more complicated solution. Eliminate the above process entirely. Eliminate the SIGNAL declaration of next_sreg. Then, replace all appearances of next_sreg in the module with sreg. The VHDL module then will appear as below.

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY ASYNCH IS
    PORT (XX: IN std_logic;
          Z : OUT std_logic);
END;

ARCHITECTURE BEHAVIOR OF ASYNCH IS
    SIGNAL sreg : std_logic_vector (1 DOWNTO 0);
    CONSTANT a : std_logic_vector (1 DOWNTO 0) := "00";
    CONSTANT b : std_logic_vector (1 DOWNTO 0) := "10";
    CONSTANT c : std_logic_vector (1 DOWNTO 0) := "11";
    CONSTANT d : std_logic_vector (1 DOWNTO 0) := "01";

BEGIN
    PROCESS (sreg, XX)
    BEGIN
        Z <= '0';
        sreg<=a;
        CASE sreg IS
            WHEN a =>
                IF ( XX='1' ) THEN
                    sreg<=b;
                    Z<='1';
                ELSE
                    sreg<=a;
                    Z<='0';
                END IF;
            WHEN b =>
                IF ( XX='0' ) THEN
                    sreg<=c;
                    Z<='0';
                ELSE
                    sreg<=b;
                    Z<='0';
                END IF;
            WHEN c =>
                IF ( XX='1' ) THEN
                    sreg<=d;
                    Z<='0';

```

```

        ELSE
            sreg<=c;
            Z<='0';
        END IF;
    WHEN d =>
        IF ( XX='0' ) THEN
            sreg<=a;
            Z<='0';
        ELSE
            sreg<=d;
            Z<='0';
        END IF;
    WHEN OTHERS =>
        END CASE;
    END PROCESS;
END BEHAVIOR;

```

When compiled, the report file obtained is as shown below (abbreviated).

Circuit simplification

Substituting virtuals - pass 1:

Note: Virtual signal sreg_1 Creating a cycle.

Substituting virtuals - pass 2:

Note: Virtual signal sreg_0 Creating a cycle.

Substituting virtuals - pass 3:

The compiler recognized that this is feedback logic!

Completed Successfully

PLD Optimizer Software: DSGNOPT.EXE 01/MAR/97 [v4.00] 4 IR x77

OPTIMIZATION OPTIONS (10:05:46)

Messages:

- Information: Process virtual 'sreg_0' ... converted to NODE.
- Information: Process virtual 'sreg_1' ... converted to NODE.
- Information: Optimizing logic using best output polarity for signals:
 sreg_0 sreg_1
- Information: Selected logic optimization OFF for signals:
 z

Completed Successfully

DESIGN EQUATIONS (10:05:49)

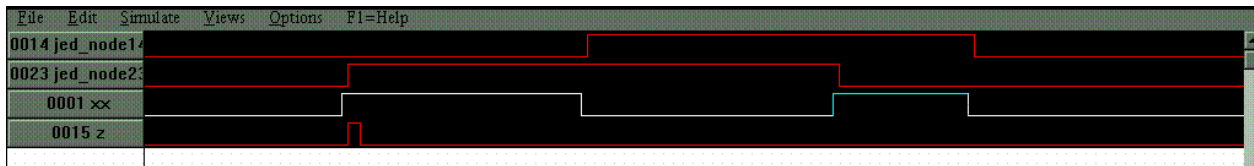
```

z = xx * /sreg_1 * /sreg_0
sreg_0 = xx * sreg_0 + /xx * sreg_1
sreg_1 = /xx * sreg_1 + xx * /sreg_0

```

xx	=	1		24	*	not used
not used	*	2		23	=	(sreg_1)
not used	*	3		22	*	not used
not used	*	4		21	*	not used
not used	*	5		20	*	not used
not used	*	6		19	*	not used
not used	*	7		18	*	not used
not used	*	8		17	*	not used
not used	*	9		16	*	not used
not used	*	10		15	=	z
not used	*	11		14	=	(sreg_0)
not used	*	12		13	*	not used

The NOVA simulation of this is shown below and is correct. Note that this is a weird machine. The way it is specified, all you get for an output is a glitch in going from state a to b.



In closnig, note that either the quick solution or the “purist” solution yields the same results. WARP2 software will just eliminate the next_sreg in the reduction process anyway - so why not let it do it instead of you? Actually, the documentation is a little clearer if you leave the next_sreg in the module.

Finally, none of this should have been a great surprise. The only technical difference between synchronous and asynchronous state machines is the presence or absence of a clock! There is a great difference in state assignment techniques but these occur before what we have done here.