CODE

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

entity game is

PORT(clock, r:in bit;

n,s,e,w,v: in bit;

sw,win,die,s0,s1,s2,s3,s4,s5,s6:out bit);

end entity game;

architecture operation of game is

type state\_type is (Ccacophony,TTunnel,RRiver,SStash,DDen,GGraveyard,VVault,gameout);

signal State : State\_Type;

begin

process(clock,r,n,s,e,w)

begin

if (r='1') then

State<= Ccacophony;

s0<='1';

s1<='0';

s2<='0';

s3<='0';

s4<='0';

s5<='0';

s6<='0';

win<='0';

die<='0';

sw<='0';

else if(clock'EVENT AND clock='1') then

case state is

when Ccacophony =>

if (e='1') and (n='0') and (s='0') and (w='0') then

State <= TTunnel;

s0<='0';

s1<='1';

s2<='0';

s3<='0';

s4<='0';

s5<='0';

s6<='0';

win<='0';

die<='0';

sw<='0';

else

State <= Ccacophony;

s0<='1';

s1<='0';

s2<='0';

s3<='0';

s4<='0';

s5<='0';

s6<='0';

win<='0';

die<='0';

sw<='0';

end if;

when TTunnel =>

if (s='1') and (n='0') and (e='0') and (w='0') then

State <= RRiver;

s0<='0';

s1<='0';

s2<='1';

s3<='0';

s4<='0';

s5<='0';

s6<='0';

win<='0';

die<='0';

sw<='0';

else if (w='1') and (n='0') and (s='0') and (e='0') then

State <= Ccacophony;

s0<='1';

s1<='0';

s2<='0';

s3<='0';

s4<='0';

s5<='0';

s6<='0';

win<='0';

die<='0';

sw<='0';

else

State <= TTunnel;

s0<='0';

s1<='1';

s2<='0';

s3<='0';

s4<='0';

s5<='0';

s6<='0';

win<='0';

die<='0';

sw<='0';

end if;

end if;

when RRiver =>

if (e='1') and (n='0') and (s='0') and (w='0') then

State <= DDen;

s0<='0';

s1<='0';

s2<='0';

s3<='0';

s4<='1';

s5<='0';

s6<='0';

win<='0';

die<='0';

sw<='0';

else if (n='1') and (s='0') and (e='0') and (w='0') then

state<=TTunnel;

s0<='0';

s1<='1';

s2<='0';

s3<='0';

s4<='0';

s5<='0';

s6<='0';

win<='0';

die<='0';

sw<='0';

else if (w='1') and (n='0') and(s='0') and (w='0') then

State <= SStash;

s0<='0';

s1<='0';

s2<='0';

s3<='1';

s4<='0';

s5<='0';

s6<='0';

win<='0';

die<='0';

sw<='1';

else

State <= RRiver;

s0<='0';

s1<='0';

s2<='1';

s3<='0';

s4<='0';

s5<='0';

s6<='0';

win<='0';

die<='0';

sw<='0';

end if;

end if;

end if;

when DDen =>

if (e='1') and (n='0') and (w='0') and (s='0') and (v='0') then

State <=GGraveyard;

s0<='0';

s1<='0';

s2<='0';

s3<='0';

s4<='0';

s5<='1';

s6<='0';

win<='0';

die<='1';

sw<='0';

else if (e='1') and (n='0') and (s='0') and(w='0') and (v='1') then

state<=VVault;

s0<='0';

s1<='0';

s2<='0';

s3<='0';

s4<='0';

s5<='0';

s6<='1';

win<='1';

die<='0';

sw<='0';

end if;

end if;

when SStash =>

if (e='1') and (n='0') and (w='0') and (s='0') then

State <= RRiver;

s0<='0';

s1<='0';

s2<='1';

s3<='0';

s4<='0';

s5<='0';

s6<='0';

win<='0';

die<='0';

sw<='0';

else

State <= SStash;

s0<='0';

s1<='0';

s2<='0';

s3<='1';

s4<='0';

s5<='0';

s6<='0';

win<='0';

die<='0';

sw<='1';

end if;

when others => NULL;

end case;

end if;

end if;

end process;

end architecture operation;

TEST BENCH

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

ENTITY game\_vhd\_tst IS

END game\_vhd\_tst;

ARCHITECTURE game\_arch OF game\_vhd\_tst IS

-- constants

-- signals

SIGNAL clock : STD\_LOGIC;

SIGNAL die : STD\_LOGIC;

SIGNAL e : STD\_LOGIC;

SIGNAL n : STD\_LOGIC;

SIGNAL r : STD\_LOGIC;

SIGNAL s : STD\_LOGIC;

SIGNAL s0 : STD\_LOGIC;

SIGNAL s1 : STD\_LOGIC;

SIGNAL s2 : STD\_LOGIC;

SIGNAL s3 : STD\_LOGIC;

SIGNAL s4 : STD\_LOGIC;

SIGNAL s5 : STD\_LOGIC;

SIGNAL s6 : STD\_LOGIC;

SIGNAL sw : STD\_LOGIC;

SIGNAL v : STD\_LOGIC;

SIGNAL w : STD\_LOGIC;

SIGNAL win : STD\_LOGIC;

COMPONENT game

PORT (clock : IN STD\_LOGIC;

die : OUT STD\_LOGIC;

e : IN STD\_LOGIC;

n : IN STD\_LOGIC;

r : IN STD\_LOGIC;

s : IN STD\_LOGIC;

s0 : OUT STD\_LOGIC;

s1 : OUT STD\_LOGIC;

s2 : OUT STD\_LOGIC;

s3 : OUT STD\_LOGIC;

s4 : OUT STD\_LOGIC;

s5 : OUT STD\_LOGIC;

s6 : OUT STD\_LOGIC;

sw : OUT STD\_LOGIC;

v : IN STD\_LOGIC;

w : IN STD\_LOGIC;

win : OUT STD\_LOGIC

);

END COMPONENT;

BEGIN

i1 : game

PORT MAP (

-- list connections between master ports and signals

clock => clock,

die => die,

e => e,

n => n,

r => r,

s => s,

s0 => s0,

s1 => s1,

s2 => s2,

s3 => s3,

s4 => s4,

s5 => s5,

s6 => s6,

sw => sw,

v => v,

w => w,

win => win

);

init : PROCESS

-- variable declarations

BEGIN

-- code that executes only once

WAIT;

END PROCESS init;

always : PROCESS

-- optional sensitivity list

-- ( )

-- variable declarations

BEGIN

-- code executes for every event on sensitivity list

clock<='1';

wait for 50 ns;

e<='1';

wait for 100 ns;

s6<='1';

wait for 100 ns;

v<='1';

wait for 100 ns;

win<='1';

wait for 100 ns;

WAIT;

END PROCESS always;

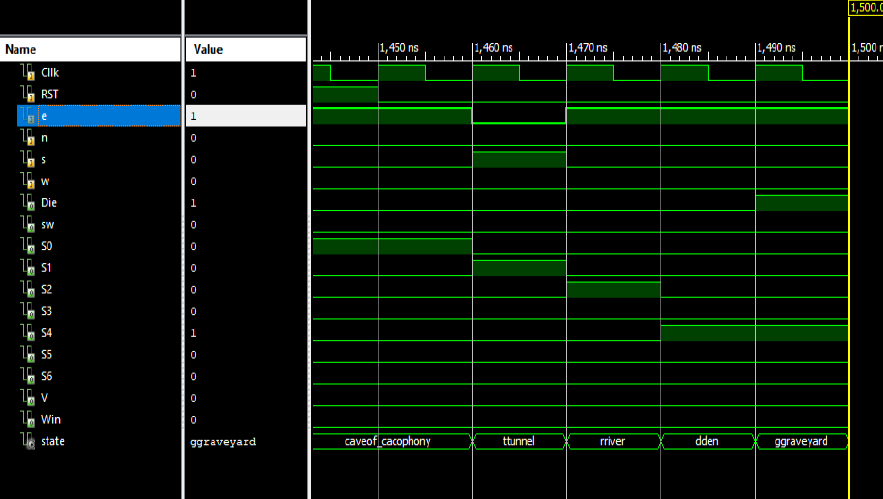
END game\_arch;

WAVEFORM:

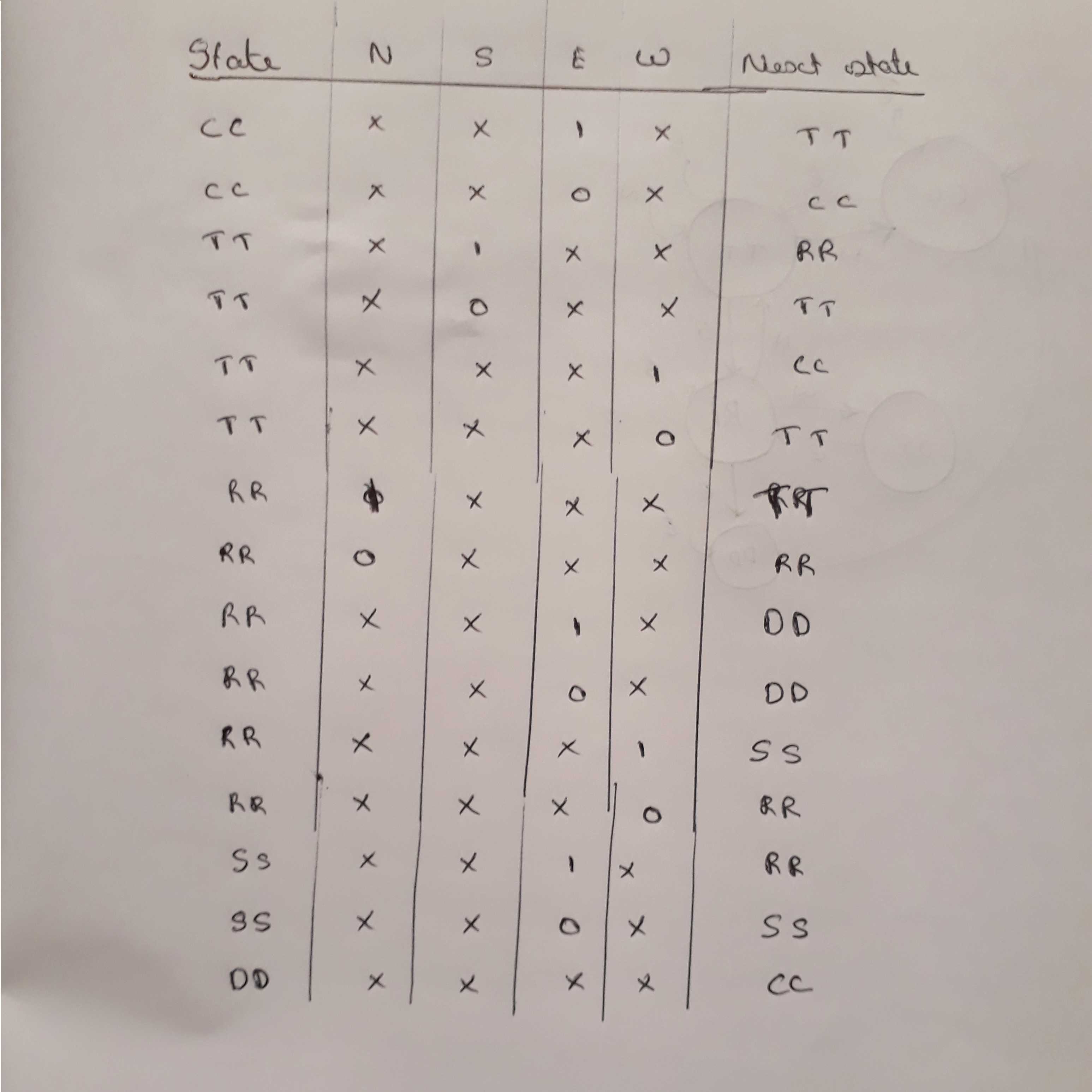
**Sequence for win:**



**Sequence for die:**

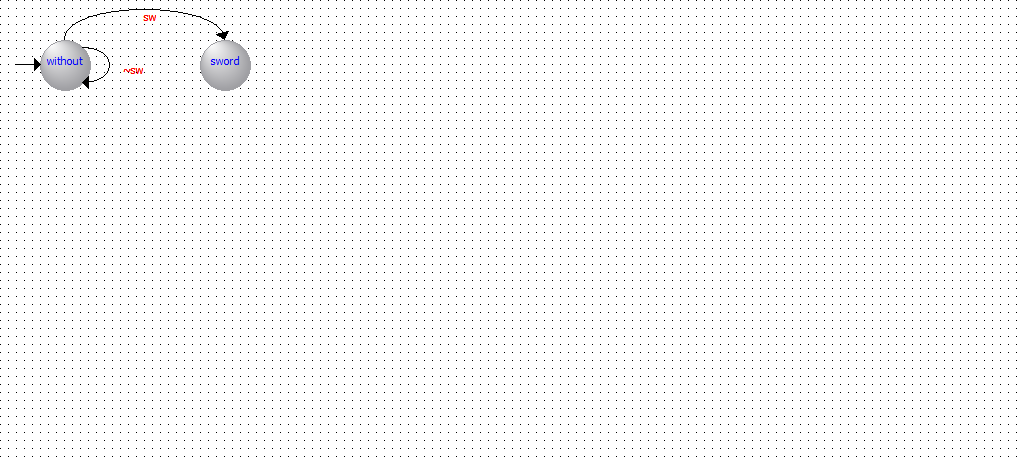


STATE TRANSITION TABLE



STATE TRANSITION DIAGRAM

SWORD



ROOM

