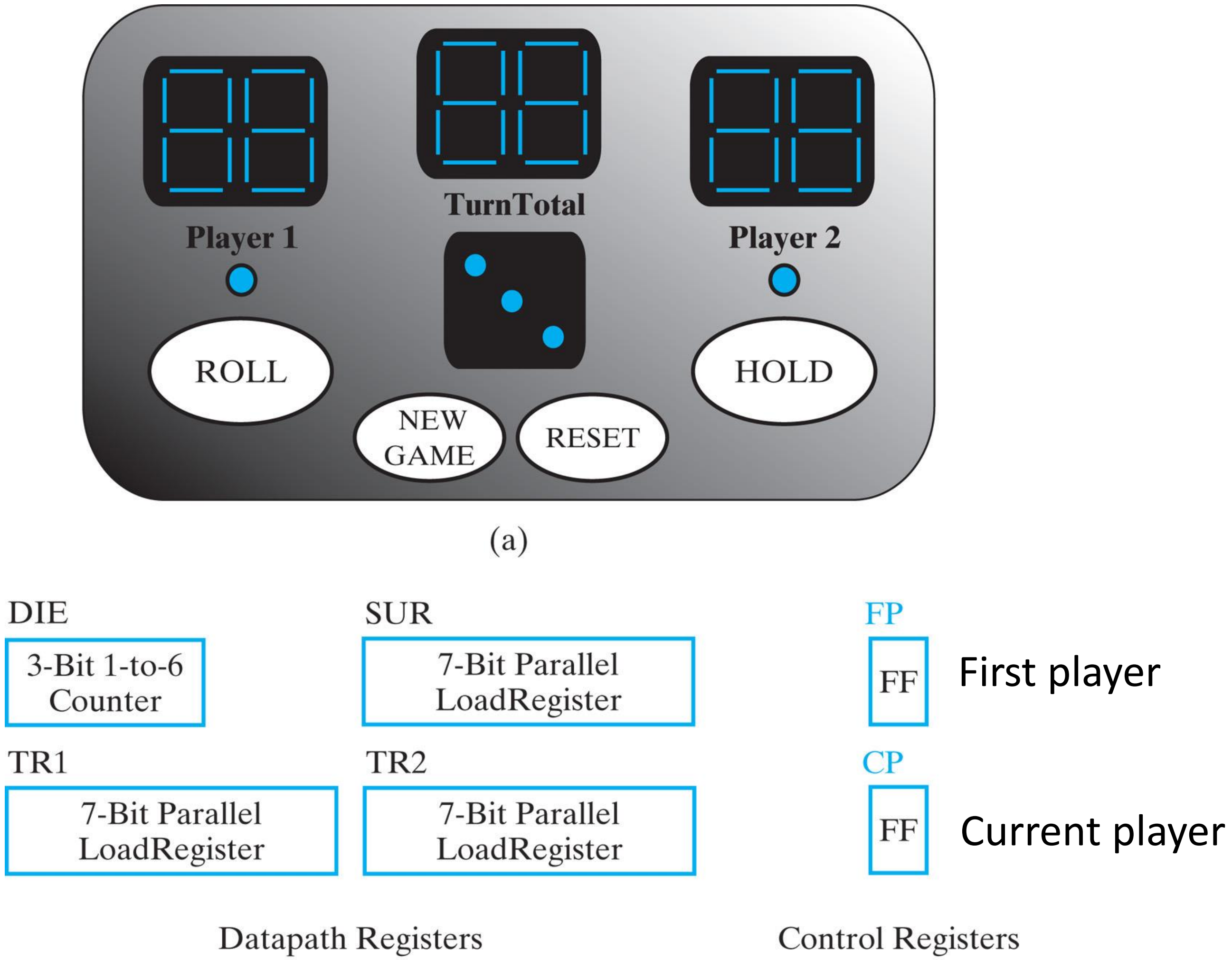
LAB 2

PIG GAME

FIGURE 6-28 PIG: (a) Exterior View of PIG, (b) PIG Registers



(b)

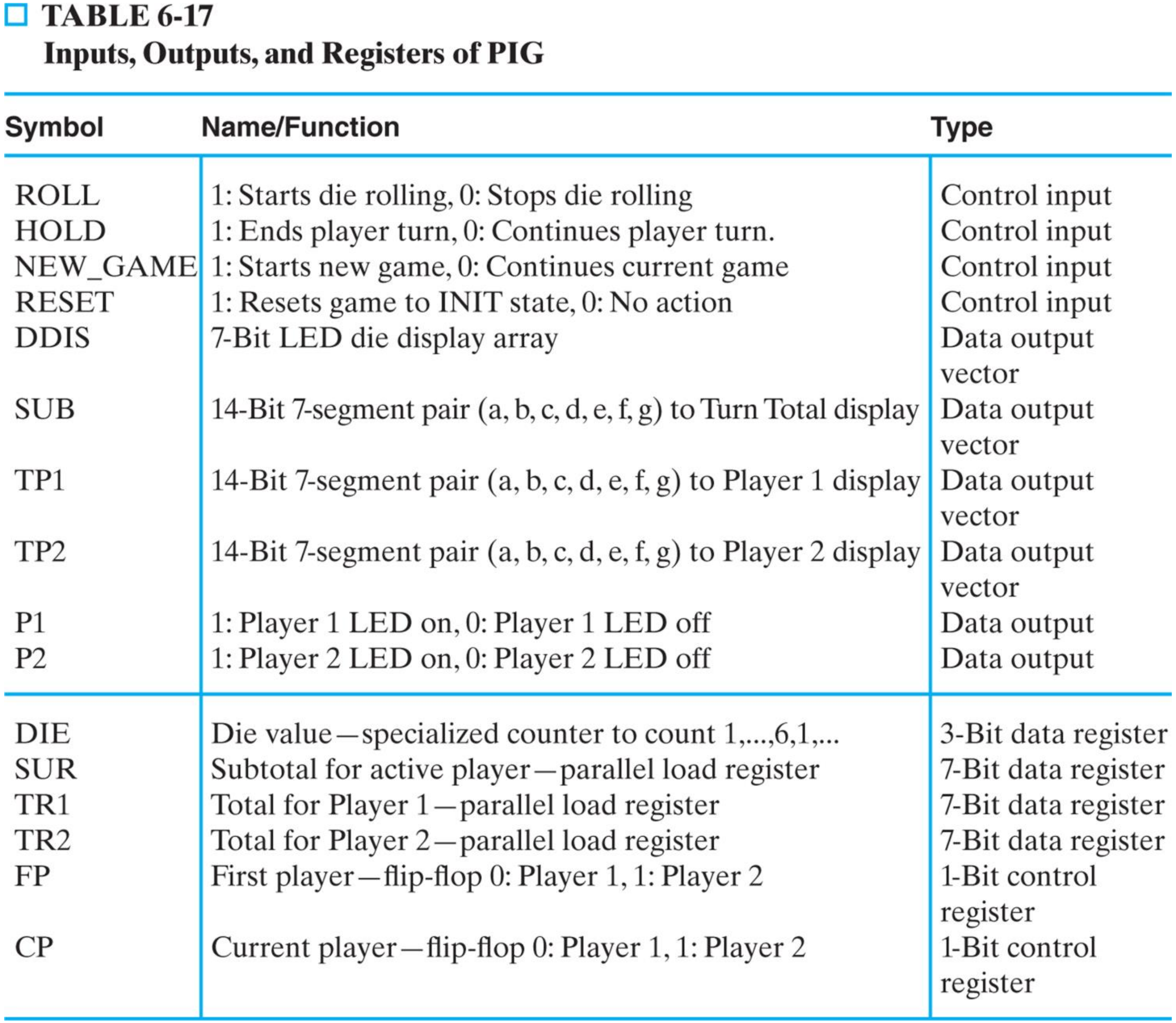
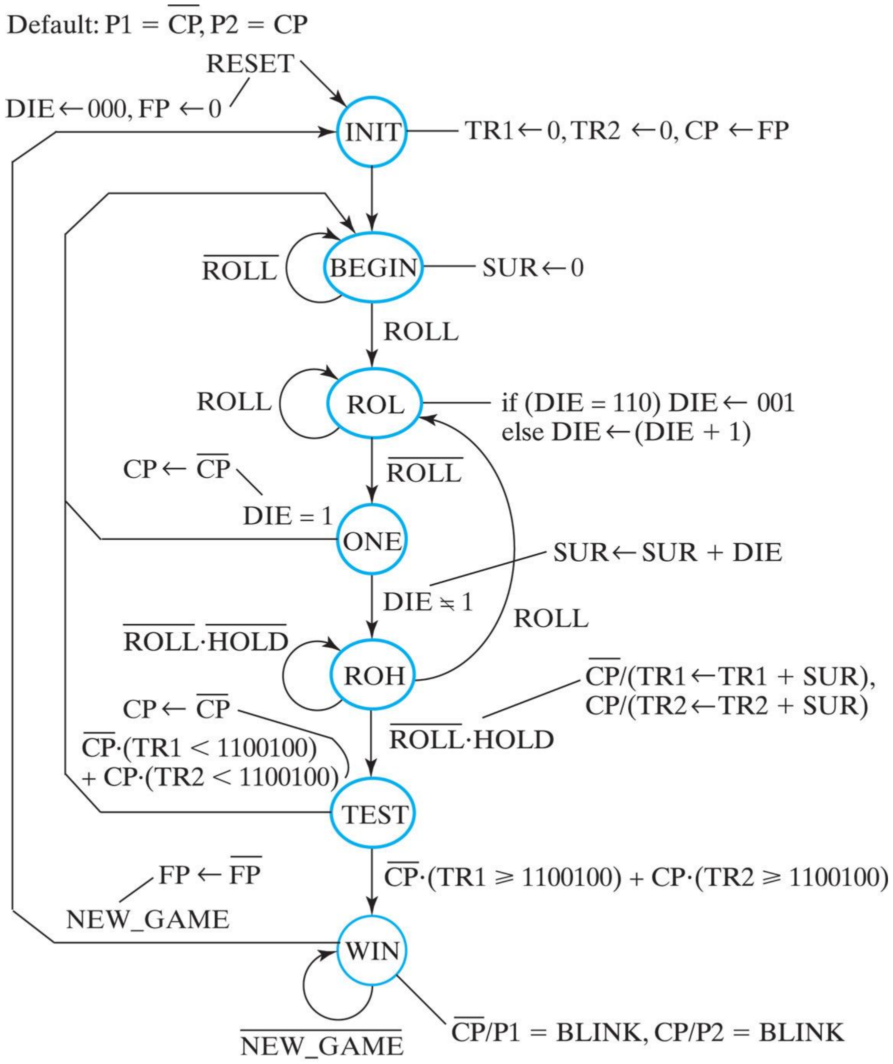


TABLE 6-17
Inputs, Outputs, and Registers of PIG

Symbol	Name/Function	Type
ROLL	1: Starts die rolling, 0: Stops die rolling	Control input
HOLD	1: Ends player turn, 0: Continues player turn.	Control input
NEW_GAME	1: Starts new game, 0: Continues current game	Control input
RESET	1: Resets game to INIT state, 0: No action	Control input
DDIS	7-Bit LED die display array	Data output vector
SUB	14-Bit 7-segment pair (a, b, c, d, e, f, g) to Turn Total display	Data output vector
TP1	14-Bit 7-segment pair (a, b, c, d, e, f, g) to Player 1 display	Data output vector
TP2	14-Bit 7-segment pair (a, b, c, d, e, f, g) to Player 2 display	Data output vector
P1	1: Player 1 LED on, 0: Player 1 LED off	Data output
P2	1: Player 2 LED on, 0: Player 2 LED off	Data output
DIE	Die value—specialized counter to count 1,...,6,1,...	3-Bit data register
SUR	Subtotal for active player—parallel load register	7-Bit data register
TR1	Total for Player 1—parallel load register	7-Bit data register
TR2	Total for Player 2—parallel load register	7-Bit data register
FP	First player—flip-flop 0: Player 1, 1: Player 2	1-Bit control register
CP	Current player—flip-flop 0: Player 1, 1: Player 2	1-Bit control register


Default: P1 = $\overline{CP}$, P2 = CP
RESET
DIE ← 000, FP ← 0
INIT
TR1 ← 0, TR2 ← 0, CP ← FP
$\overline{ROLL}$
BEGIN
SUR ← 0
ROLL
ROLL
ROL
if (DIE = 110) DIE ← 001
else DIE ← (DIE + 1)
$\overline{ROLL}$
CP ← $\overline{CP}$
DIE = 1
ONE
SUR ← SUR + DIE
DIE ≠ 1
ROLL
$\overline{ROLL}$·$\overline{HOLD}$
ROH
$\overline{CP}$/(TR1 ← TR1 + SUR),
CP/(TR2 ← TR2 + SUR)
CP ← $\overline{CP}$
$\overline{ROLL}$·HOLD
$\overline{CP}$·(TR1 < 1100100)
+ CP·(TR2 < 1100100)
TEST
$\overline{CP}$·(TR1 ⩾ 1100100) + CP·(TR2 ⩾ 1100100)
FP ← $\overline{FP}$
NEW_GAME
WIN
$\overline{NEW\_GAME}$
$\overline{CP}$/P1 = BLINK, CP/P2 = BLINK

□ TABLE 6-18
Datapath Output Actions and Control and Status Signals for PIG

Action or Status	Control or Status Signals	Meaning for Values 1 and 0

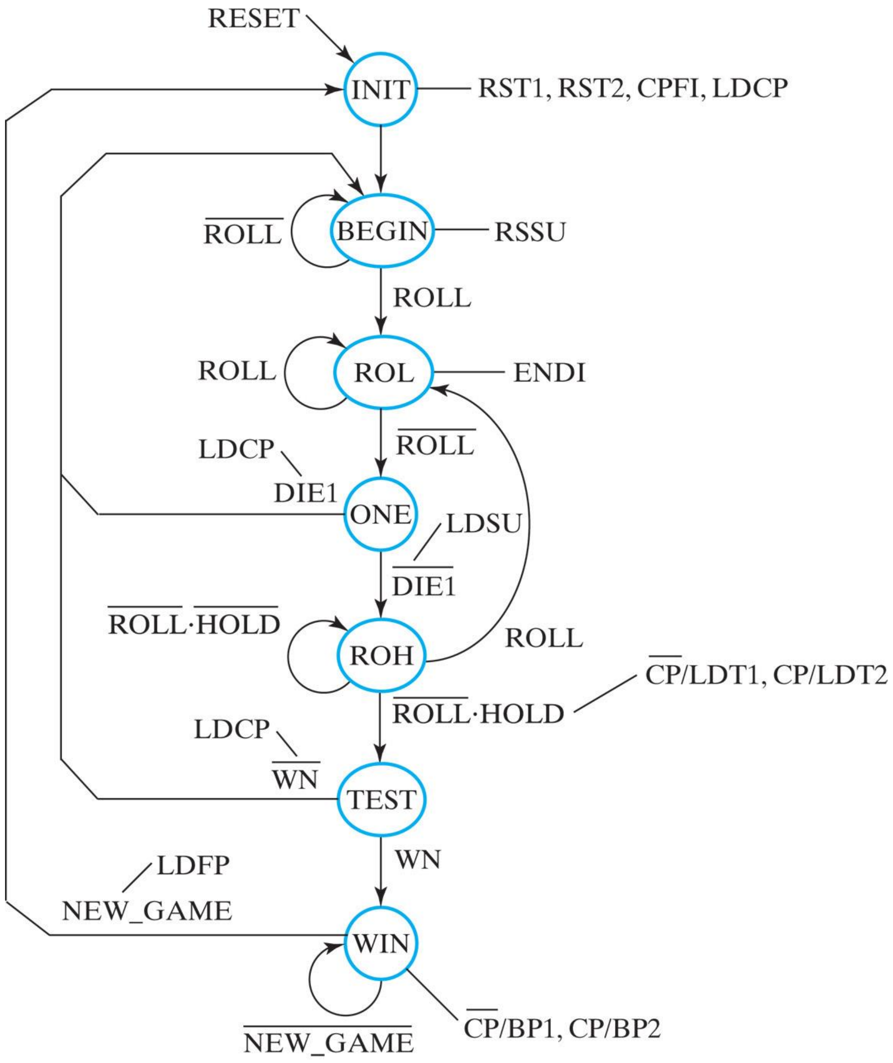
TR1 ← 0
TR1 ← TR1 + SUR
	RST1
	LDT1
		1: Reset TR1 (synchronous reset), 0: No action
		1: Add SUR to TR1, 0: No action

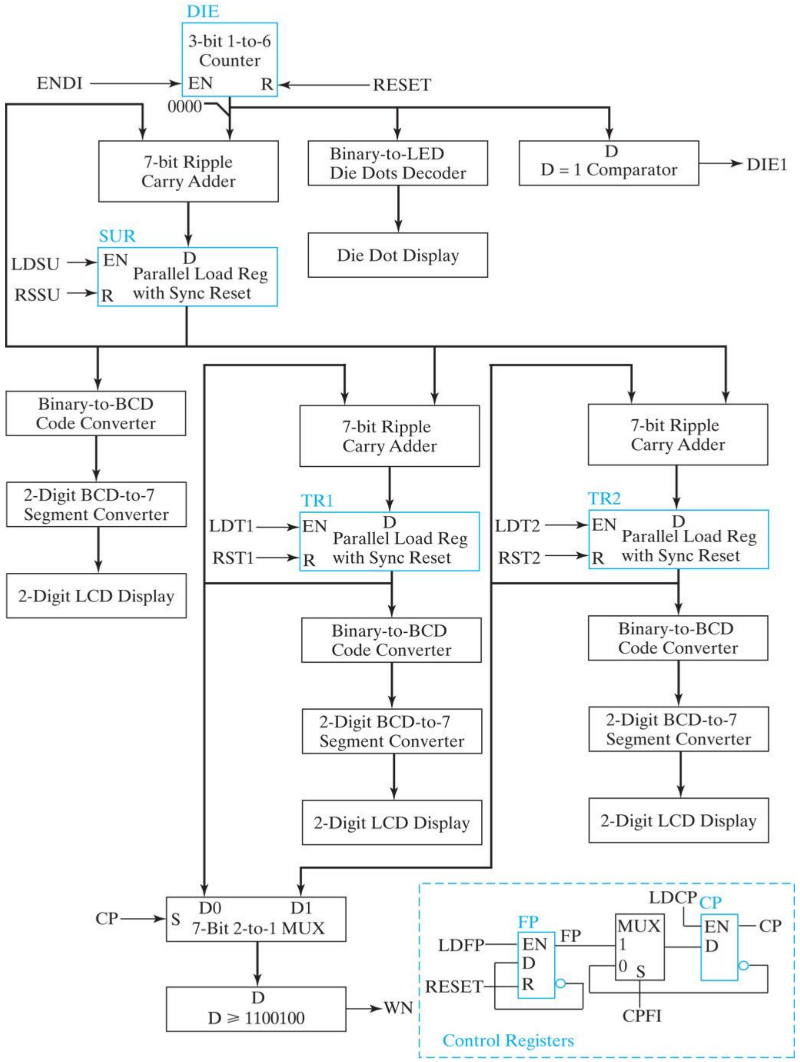
TR2 ← 0
TR2 ← TR2 + SUR
	RST2
	LDT2
		1: Reset TR2 (synchronous reset), 0: No action
		1: Add SUR to TR2, 0: No action

SUR ← 0
SUR ← SUR + DIE
	RSSU
	LDSU
		1: Reset SUR (synchronous reset), 0: No action
		1: Add DIE to SUR, 0: No action

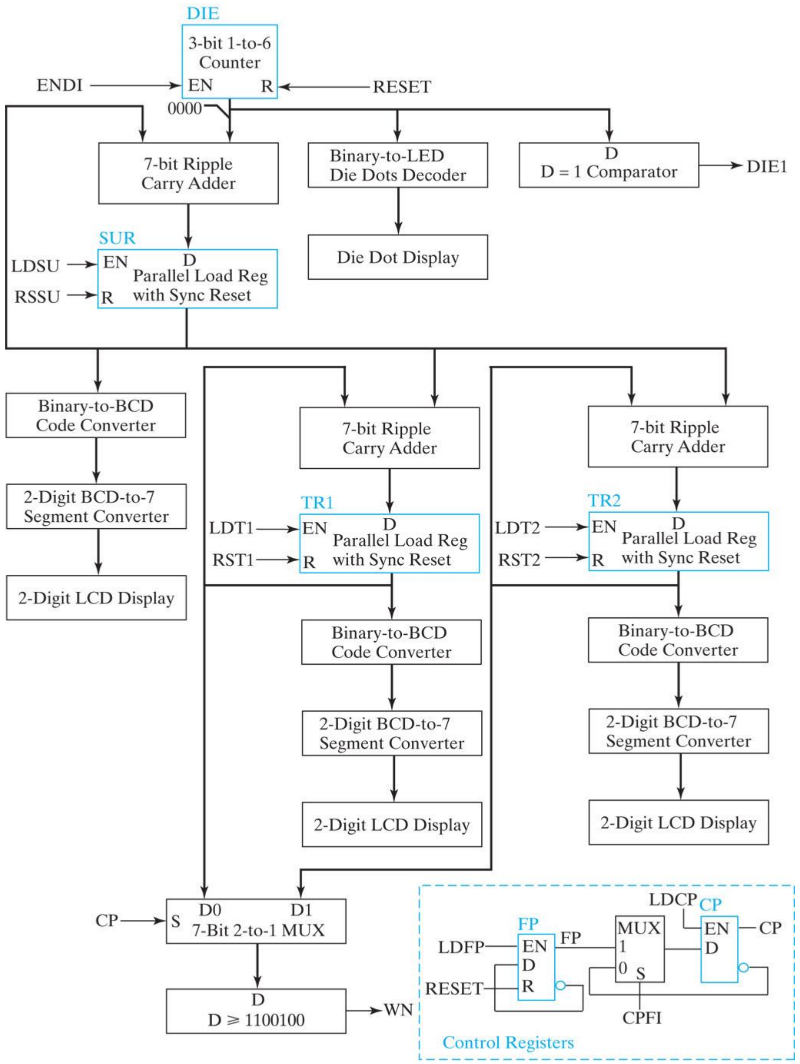
DIE ← 000
if (DIE = 110)
DIE ← 001
else DIE ← DIE + 1
	RESET
	ENDI
		1: Reset DIE to 000 (asynchronous reset)
		1: Enable DIE to increment, 0: Hold DIE value

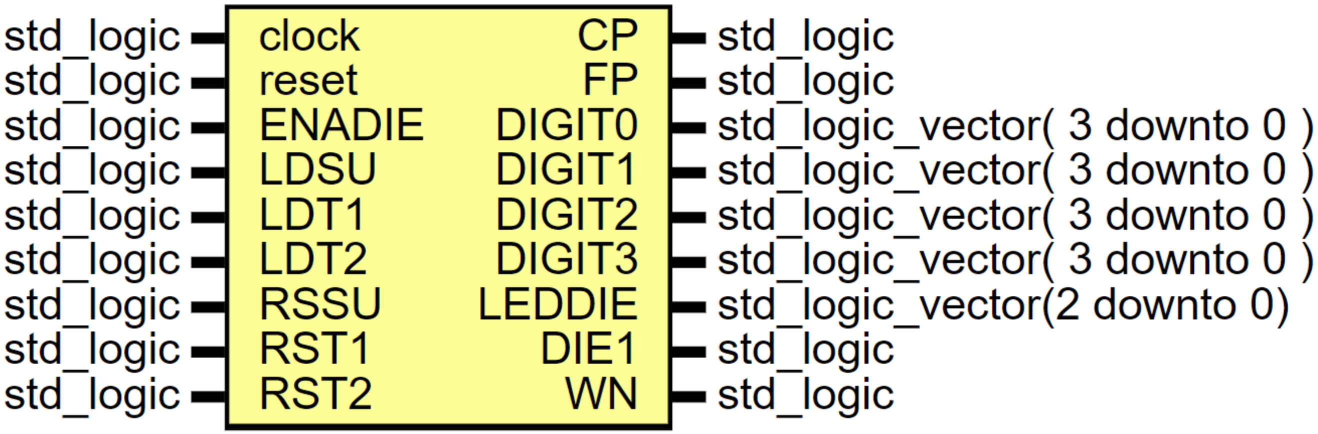
P1 = BLINK	BP1	1: Connect P1 to BLINK, 0: Connect P1 to 1



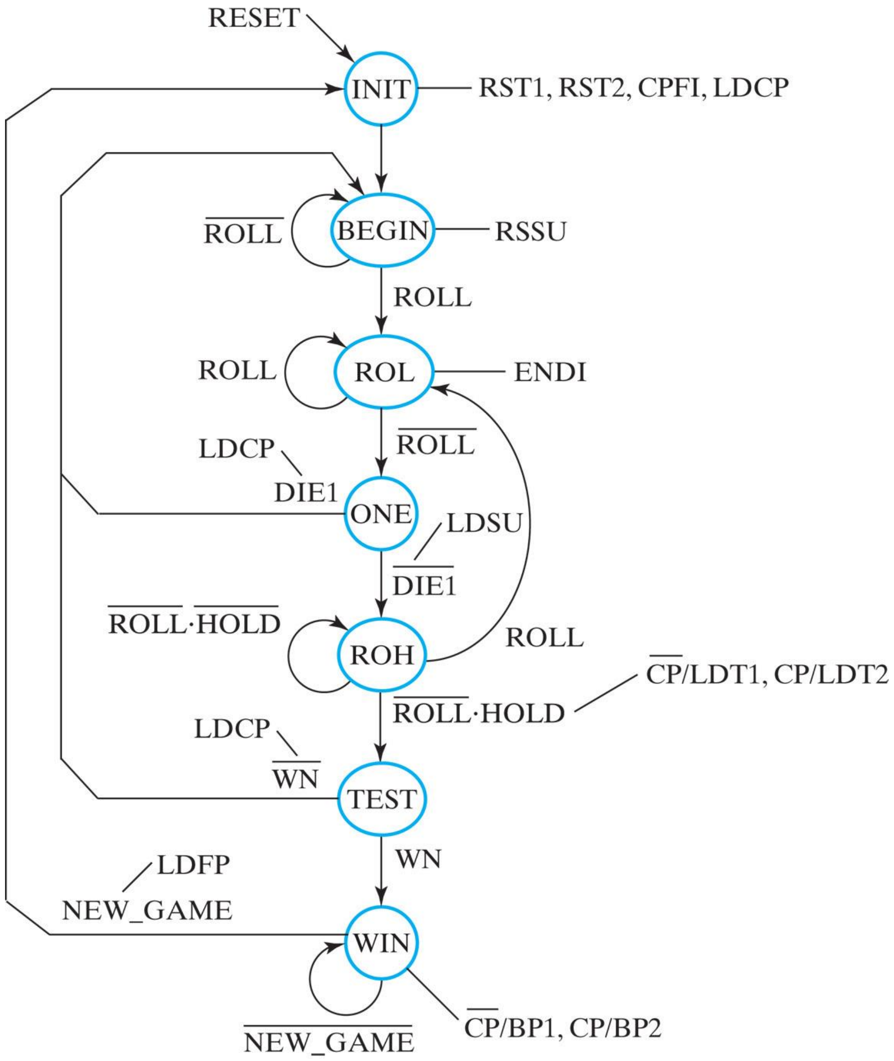


What are the inputs and outputs?

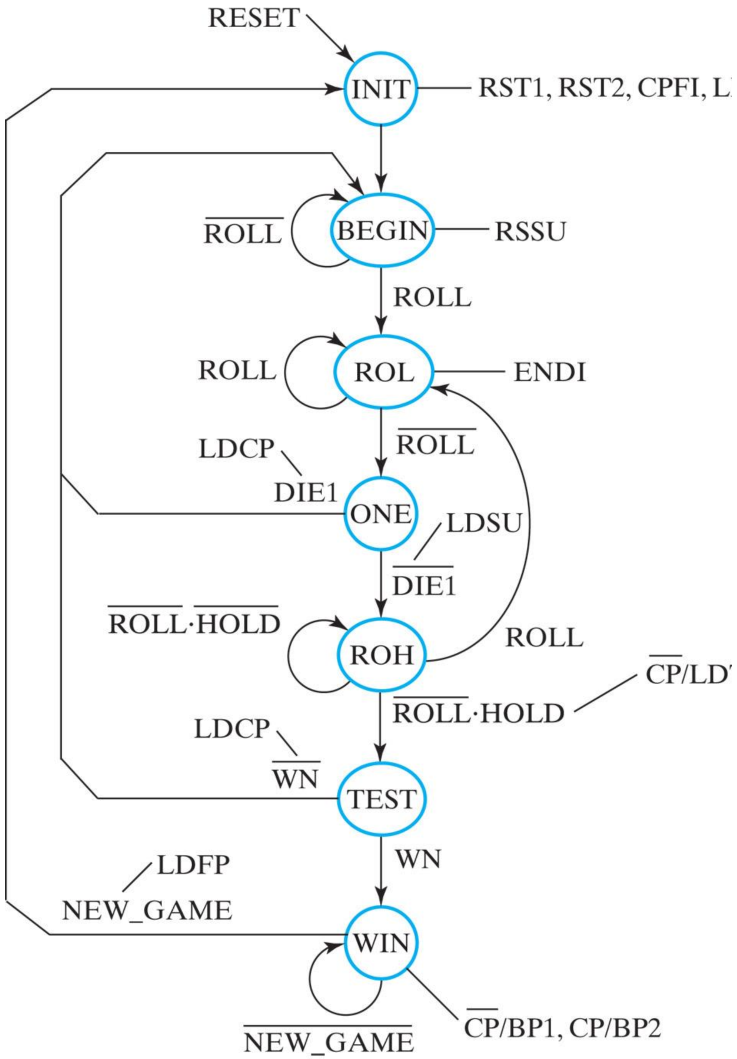


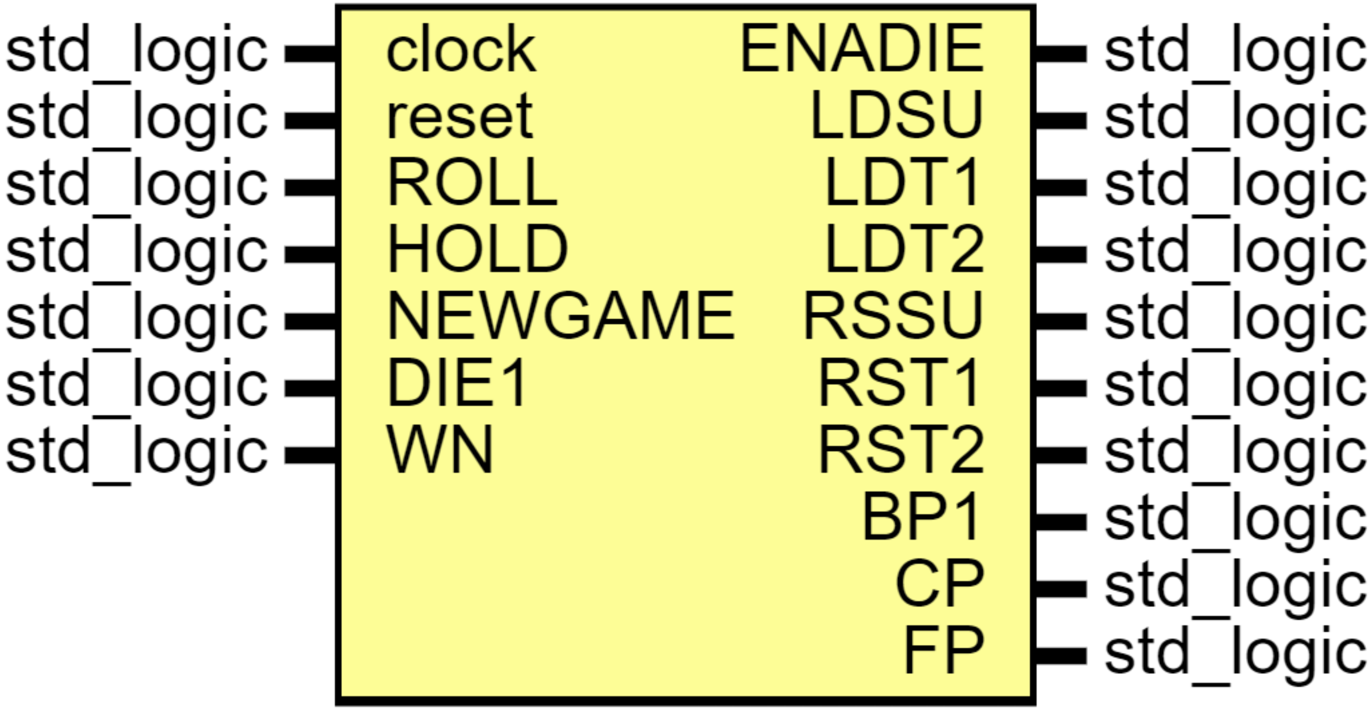


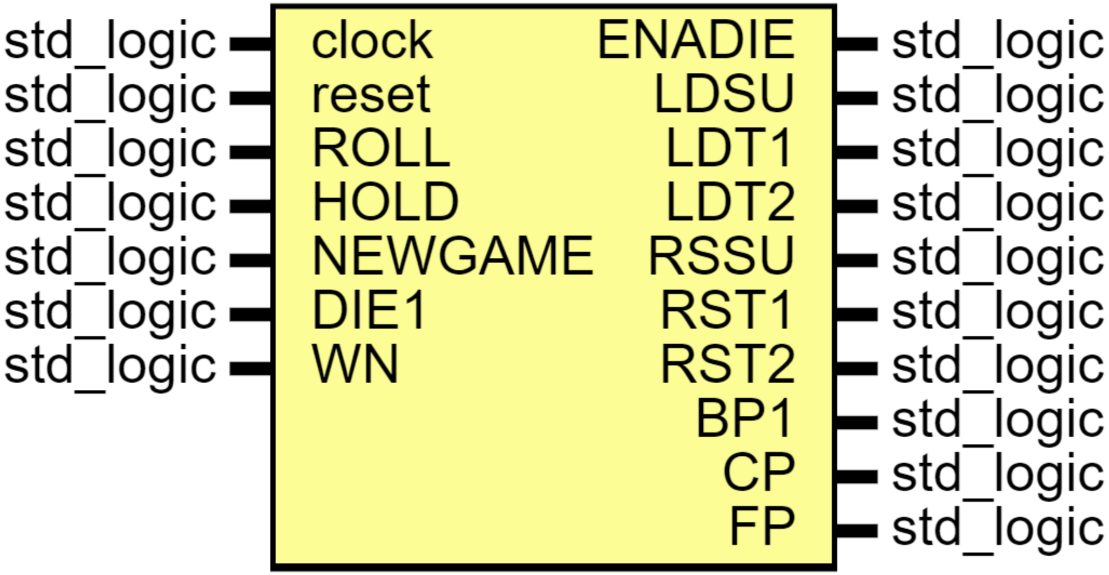
std_logic	clock	CP	std_logic
std_logic	reset	FP	std_logic
std_logic	ENADIE	DIGIT0	std_logic_vector(3 downto 0)
std_logic	LDSU	DIGIT1	std_logic_vector(3 downto 0)
std_logic	LDT1	DIGIT2	std_logic_vector(3 downto 0)
std_logic	LDT2	DIGIT3	std_logic_vector(3 downto 0)
std_logic	RSSU	LEDDIE	std_logic_vector(2 downto 0)
std_logic	RST1	DIE1	std_logic
std_logic	RST2	WN	std_logic

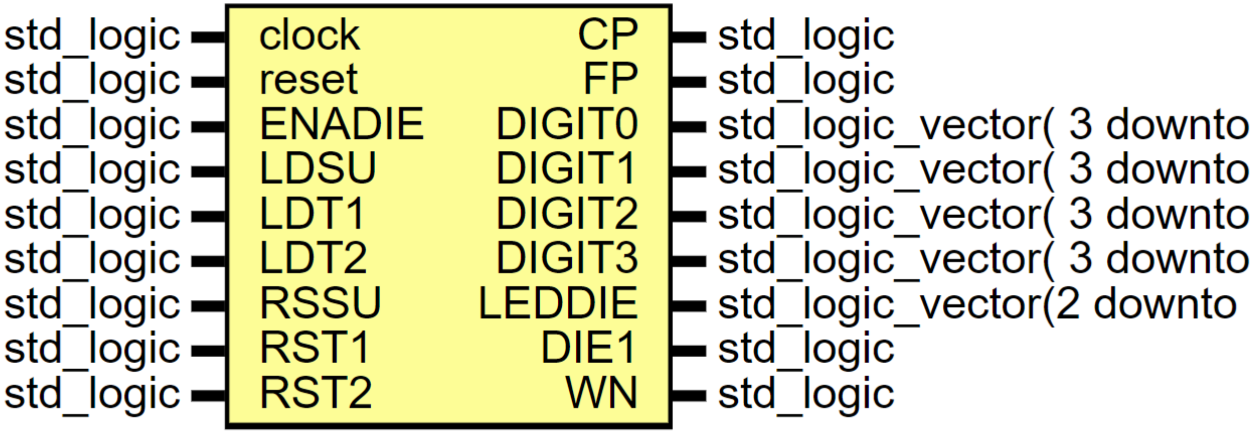


What are the inputs and outputs?

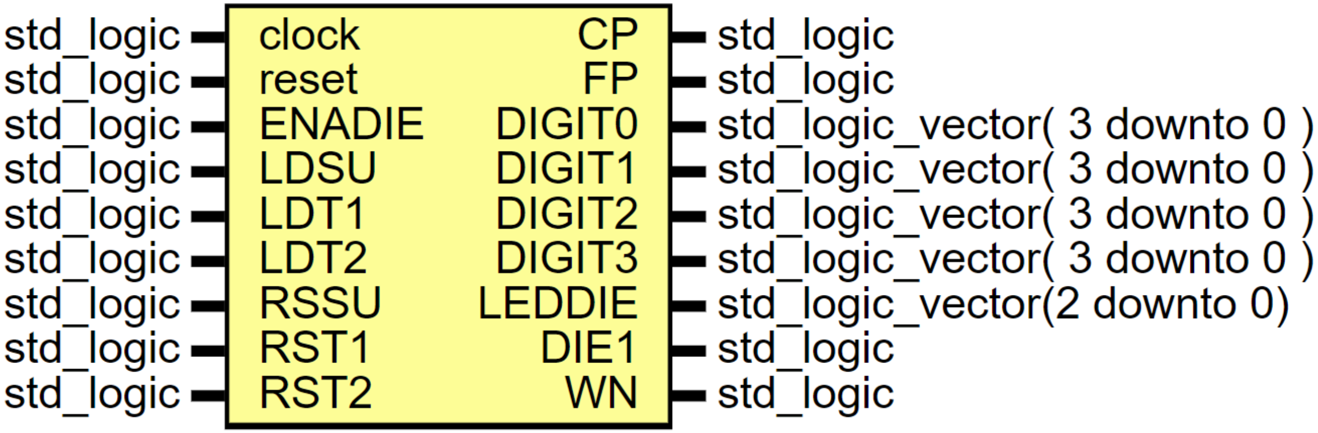

RESET
INIT
RST1, RST2, CPFI, L
BEGIN
$\overline{\text{ROLL}}$
RSSU
ROLL
ROL
ROLL
ENDI
$\overline{\text{ROLL}}$
LDCP
DIE1
ONE
LDSU
$\overline{\text{DIE1}}$
$\overline{\text{ROLL}}\cdot\overline{\text{HOLD}}$
ROH
ROLL
$\overline{\text{ROLL}}\cdot\text{HOLD}$
$\overline{\text{CP}}$/LD
LDCP
$\overline{\text{WN}}$
TEST
WN
LDFP
NEW_GAME
WIN
$\overline{\text{NEW\_GAME}}$
$\overline{\text{CP}}$/BP1, CP/BP2


std_logic clock
std_logic reset
std_logic ROLL
std_logic HOLD
std_logic NEWGAME
std_logic DIE1
std_logic WN
ENADIE std_logic
LDSU std_logic
LDT1 std_logic
LDT2 std_logic
RSSU std_logic
RST1 std_logic
RST2 std_logic
BP1 std_logic
CP std_logic
FP std_logic


std_logic clock
std_logic reset
std_logic ROLL
std_logic HOLD
std_logic NEWGAME
std_logic DIE1
std_logic WN
ENADIE std_logic
LDSU std_logic
LDT1 std_logic
LDT2 std_logic
RSSU std_logic
RST1 std_logic
RST2 std_logic
BP1 std_logic
CP std_logic
FP std_logic


std_logic clock
std_logic reset
std_logic ENADIE
std_logic LDSU
std_logic LDT1
std_logic LDT2
std_logic RSSU
std_logic RST1
std_logic RST2
CP std_logic
FP std_logic
DIGIT0 std_logic_vector(3 downto
DIGIT1 std_logic_vector(3 downto
DIGIT2 std_logic_vector(3 downto
DIGIT3 std_logic_vector(3 downto
LEDDIE std_logic_vector(2 downto
DIE1 std_logic
WN std_logic

```
entity datapath is
   port(
      clock  : in std_logic; --! Clock
      reset  : in std_logic; --! Reset
      ENADIE : in std_logic; --! Enable Die to increment
      LDSU   : in std_logic; --! Add DIE to SUR register
      LDT1   : in std_logic; --! Add SUR to TR1 register
      LDT2   : in std_logic; --! Add SUR to TR2 register
      RSSU   : in std_logic; --! Reset SUR register
      RST1   : in std_logic; --! Reset TR1 register
      RST2   : in std_logic; --! Reset TR2 register
      CP     : inout std_logic; --! current player (register outside)
      FP     : inout std_logic; --! First player (register outside)
      DIGIT0 : out std_logic_vector( 3 downto 0 ); --! digit to the right
      DIGIT1 : out std_logic_vector( 3 downto 0 ); --! 2nd digit to the left
      DIGIT2 : out std_logic_vector( 3 downto 0 ); --! 3rd digit to the left
      DIGIT3 : out std_logic_vector( 3 downto 0 ); --! digit to the left
      LEDDIE : out std_logic_vector(2 downto 0); --! LEDs to display the die value
      DIE1   : out std_logic; --! signal that a one has been obtained
      WN     : out std_logic --! WIN has been achieved by a player
   );
end entity datapath;
```

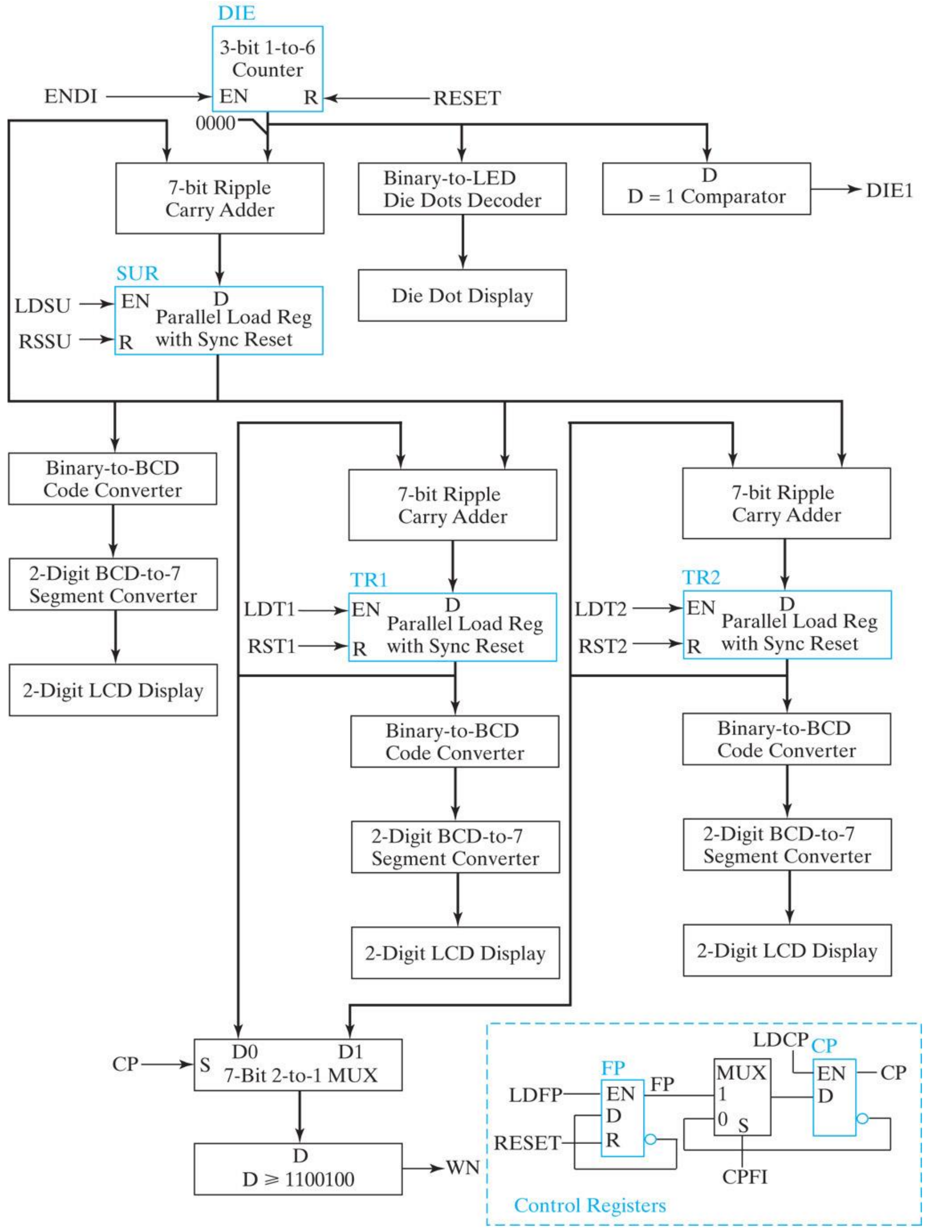


std_logic clock CP std_logic
std_logic reset FP std_logic
std_logic ENADIE DIGIT0 std_logic_vector(3 downto 0)
std_logic LDSU DIGIT1 std_logic_vector(3 downto 0)
std_logic LDT1 DIGIT2 std_logic_vector(3 downto 0)
std_logic LDT2 DIGIT3 std_logic_vector(3 downto 0)
std_logic RSSU LEDDIE std_logic_vector(2 downto 0)
std_logic RST1 DIE1 std_logic
std_logic RST2 WN std_logic

DATAPATH

```
Main_process : process(clock,reset) begin
  if reset = '1' then
    -- reset
    DIE <= (others => '0'); --! asynchronous reset
  else
    if rising_edge(clock) then
      if RST1 = '1' then
        TR1 <= (others => '0');
      end if;
      if RST2 = '1' then
        TR2 <= (others => '0');
      end if;
      if RSSU = '1' then
        SUR <= (others => '0');
      end if;
      if LDT1 = '1' then
        TR1 <= TR1 + SUR;
      end if;
      if LDT2 = '1' then
        TR2 <= TR2 + SUR;
      end if;
      if ENADIE = '1' then
        case DIE is
          when "110" => DIE <= "001";
          when others => DIE <= DIE +1;
        end case;
      end if;
```

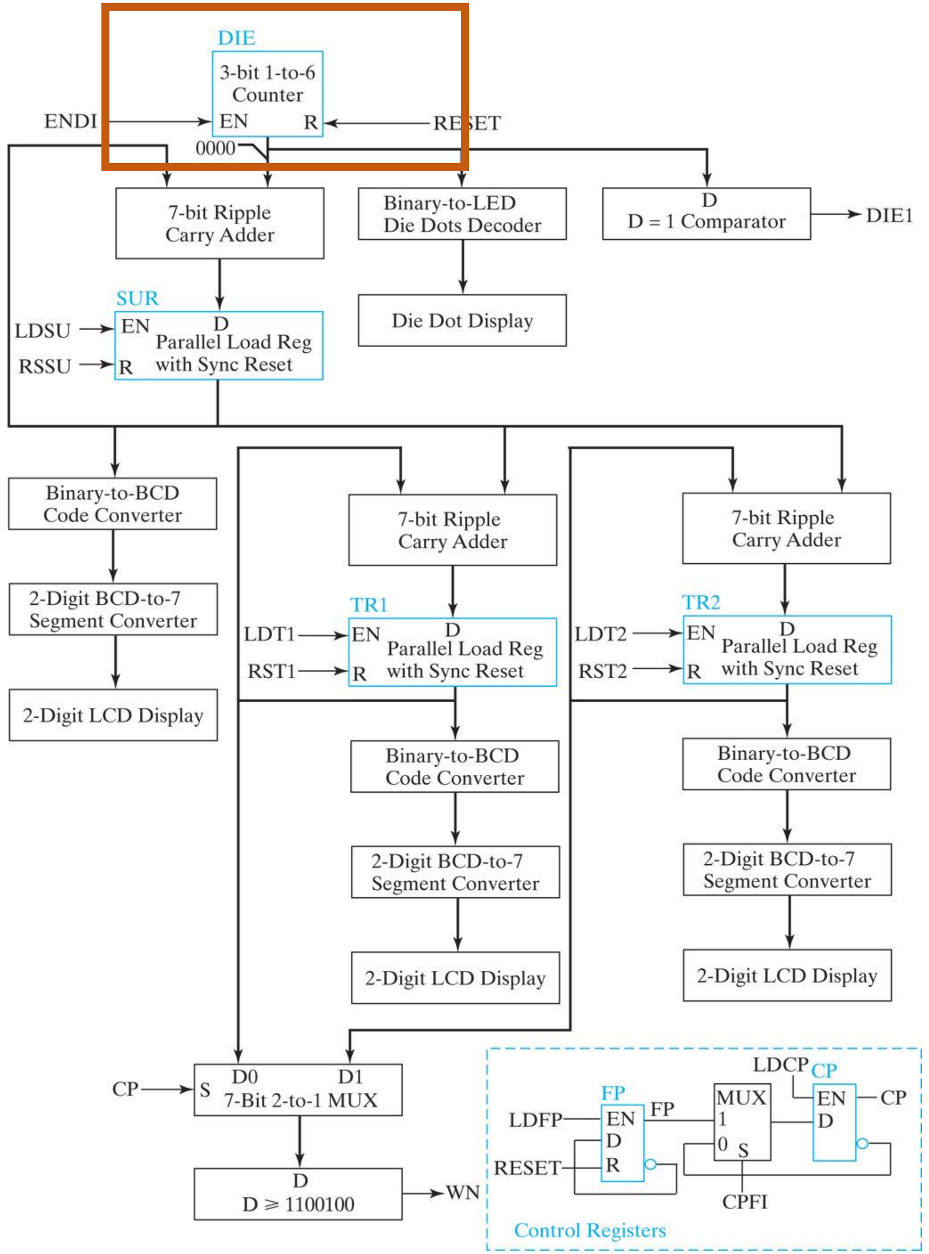
```
if DIE ="001" then
        DIE1 <= '1';
      else
        DIE1 <= '0';
      end if;
      if LDSU = '1' then
        SUR <= SUR + (frontbits & DIE);
      end if;
      if CP ='1' then
        D <= TR2;
      else
        D <= TR1;
      end if;

      if (D > "1100011") then
        WN <= '1';
       else
        WN <= '0';
      end if;

    end if;
  end if;
--! connection to displays
LEDDIE <= DIE;
DIGIT0 <= bcd1;
DIGIT1 <= bcd2;
DIGIT2 <= bcd3;
DIGIT3 <= bcd4;
end process;
```



Copyright ©2016 Pearson Education, All Rights Reserved

```
Main_process : process(clock,reset) begin
   if reset = '1' then
      -- reset
      DIE <= (others => '0'); --! asynchronous reset
   else
      if rising_edge(clock) then
         if RST1 = '1' then
            TR1 <= (others => '0');
         end if;
         if RST2 = '1' then
            TR2 <= (others => '0');
         end if;
         if RSSU = '1' then
            SUR <= (others => '0');
         end if;
         if LDT1 = '1' then
            TR1 <= TR1 + SUR;
         end if;
         if LDT2 = '1' then
            TR2 <= TR2 + SUR;
         end if;
         if ENADIE = '1' then
            case DIE is
               when "110" => DIE <= "001";
               when others => DIE <= DIE +1;
            end case;
         end if;
```

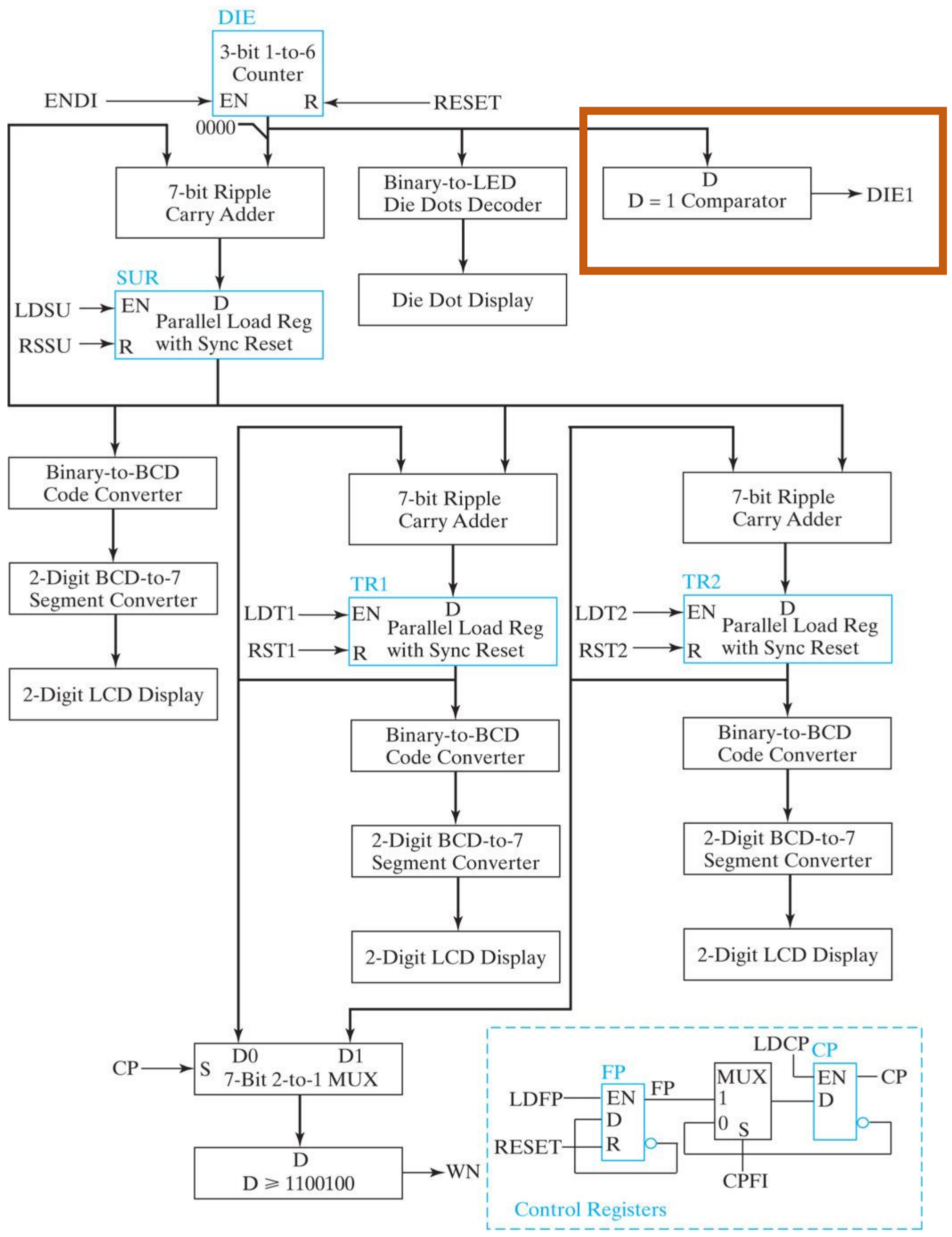
```
if DIE ="001" then
     DIE1 <= '1';
    else
     DIE1 <= '0';
    end if;
 if LDSU = '1' then
     SUR <= SUR + (frontbits & DIE);
end if;
    if CP ='1' then
     D <= TR2;
    else
     D <= TR1;
    end if;

    if (D > "1100011") then
     WN <= '1';
     else
     WN <= '0';
    end if;

  end if;
 end if;
--! connection to displays
LEDDIE <= DIE;
DIGIT0 <= bcd1;
DIGIT1 <= bcd2;
DIGIT2 <= bcd3;
DIGIT3 <= bcd4;
end process;
```



Copyright ©2016 Pearson Education, All Rights Reserved

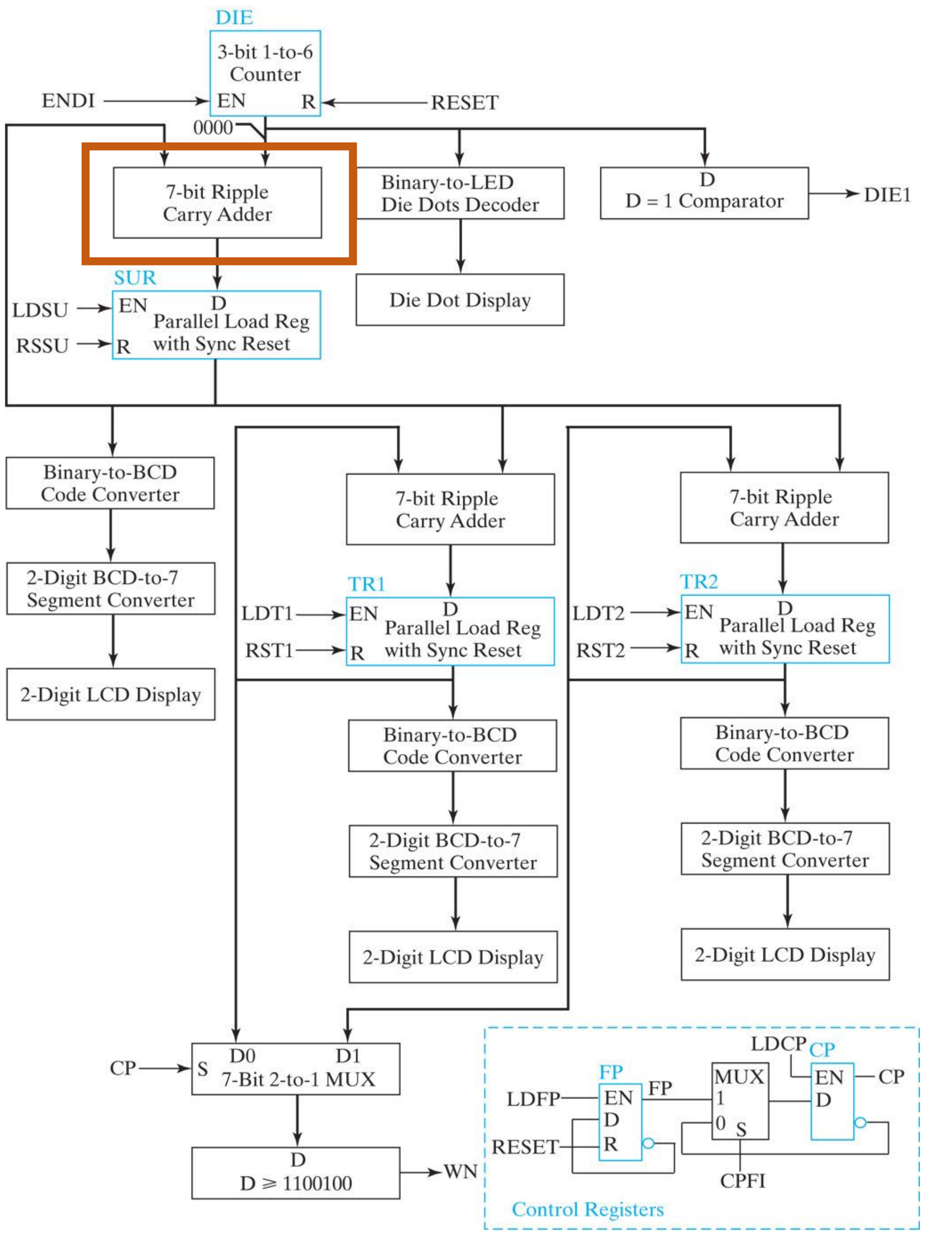
```
Main_process : process(clock,reset) begin
   if reset = '1' then
      -- reset
      DIE <= (others => '0'); --! asynchronous reset
   else
      if rising_edge(clock) then
         if RST1 = '1' then
            TR1 <= (others => '0');
         end if;
         if RST2 = '1' then
            TR2 <= (others => '0');
         end if;
         if RSSU = '1' then
            SUR <= (others => '0');
         end if;
         if LDT1 = '1' then
            TR1 <= TR1 + SUR;
         end if;
         if LDT2 = '1' then
            TR2 <= TR2 + SUR;
         end if;
         if ENADIE = '1' then
            case DIE is
               when "110" => DIE <= "001";
               when others => DIE <= DIE +1;
            end case;
         end if;
```

```
if DIE ="001" then
           DIE1 <= '1';
        else
           DIE1 <= '0';
end if;
 if LDSU = '1' then
           SUR <= SUR + (frontbits & DIE);
 end if;
        if CP ='1' then
           D <= TR2;
        else
           D <= TR1;
        end if;

        if (D > "1100011") then
           WN <= '1';
         else
           WN <= '0';
        end if;

     end if;
   end if;
--! connection to displays
LEDDIE <= DIE;
DIGIT0 <= bcd1;
DIGIT1 <= bcd2;
DIGIT2 <= bcd3;
DIGIT3 <= bcd4;
end process;
```



```
Main_process : process(clock,reset) begin
  if reset = '1' then
    -- reset
    DIE <= (others => '0'); --! asynchronous reset
  else
    if rising_edge(clock) then
      if RST1 = '1' then
        TR1 <= (others => '0');
      end if;
      if RST2 = '1' then
        TR2 <= (others => '0');
      end if;
      if RSSU = '1' then
        SUR <= (others => '0');
      end if;
      if LDT1 = '1' then
        TR1 <= TR1 + SUR;
      end if;
      if LDT2 = '1' then
        TR2 <= TR2 + SUR;
      end if;
      if ENADIE = '1' then
        case DIE is
          when "110" => DIE <= "001";
          when others => DIE <= DIE +1;
        end case;
      end if;
```

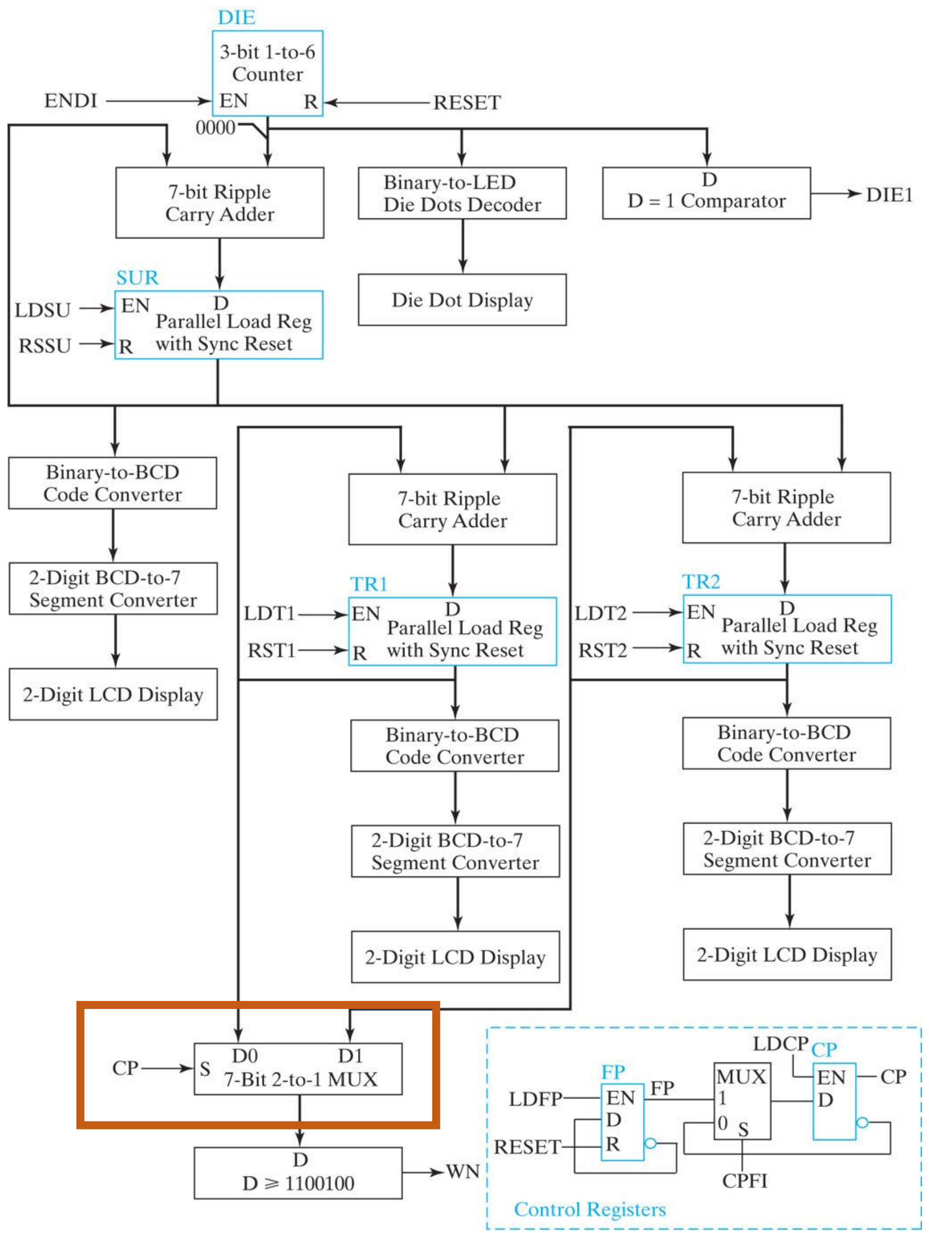
```
if DIE ="001" then
        DIE1 <= '1';
      else
        DIE1 <= '0';
      end if;
if LDSU = '1' then
        SUR <= SUR + (frontbits & DIE);
end if;
      if CP ='1' then
        D <= TR2;
      else
        D <= TR1;
      end if;

      if (D > "1100011") then
        WN <= '1';
       else
        WN <= '0';
      end if;

    end if;
  end if;
--! connection to displays
LEDDIE <= DIE;
DIGIT0 <= bcd1;
DIGIT1 <= bcd2;
DIGIT2 <= bcd3;
DIGIT3 <= bcd4;
end process;
```



Copyright ©2016 Pearson Education, All Rights Reserved

```
Main_process : process(clock,reset) begin
   if reset = '1' then
      -- reset
      DIE <= (others => '0'); --! asynchronous reset
   else
      if rising_edge(clock) then
         if RST1 = '1' then
            TR1 <= (others => '0');
         end if;
         if RST2 = '1' then
            TR2 <= (others => '0');
         end if;
         if RSSU = '1' then
            SUR <= (others => '0');
         end if;
         if LDT1 = '1' then
            TR1 <= TR1 + SUR;
         end if;
         if LDT2 = '1' then
            TR2 <= TR2 + SUR;
         end if;
         if ENADIE = '1' then
            case DIE is
               when "110" => DIE <= "001";
               when others => DIE <= DIE +1;
            end case;
         end if;
```

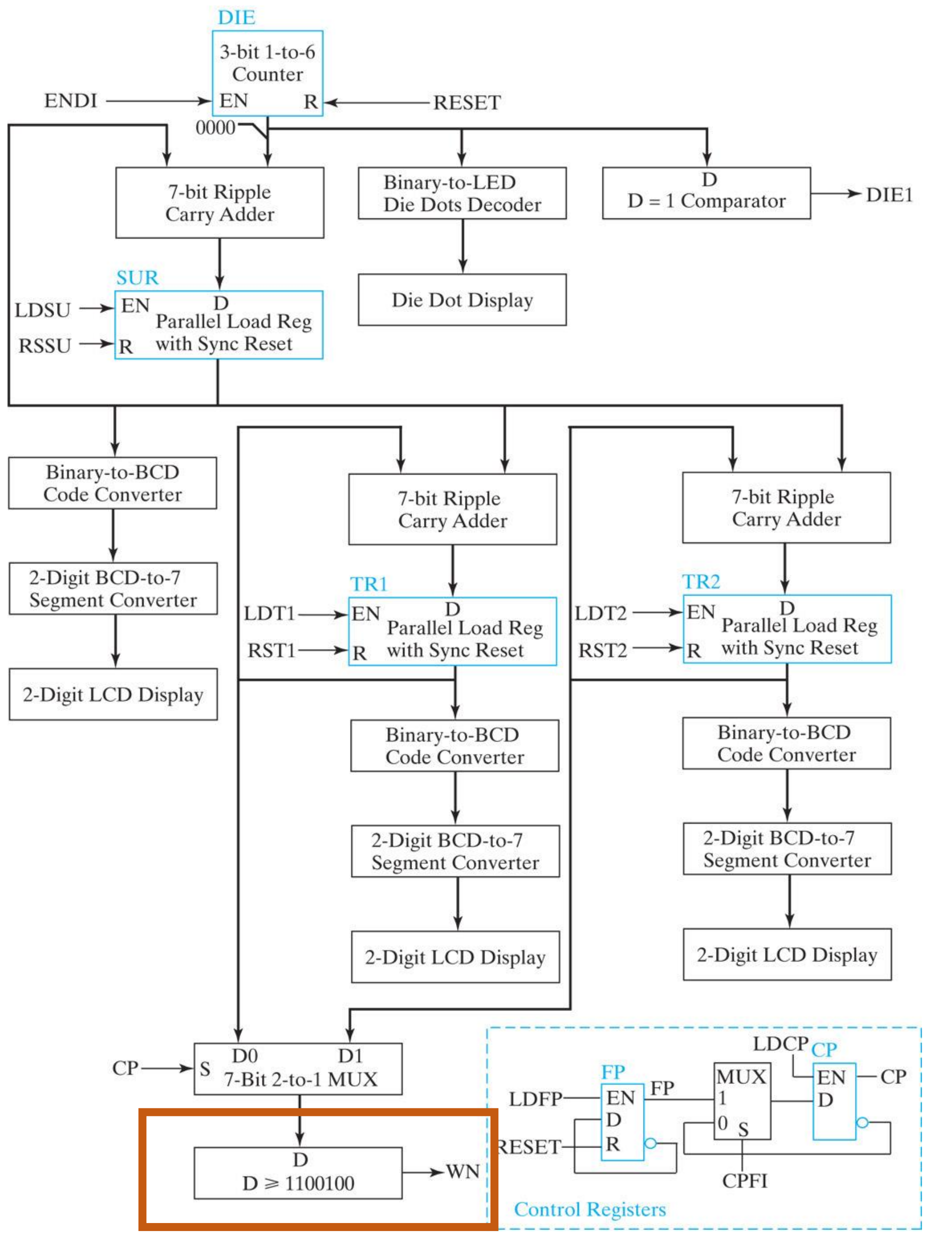
```
if DIE ="001" then
            DIE1 <= '1';
         else
            DIE1 <= '0';
         end if;
         if LDSU = '1' then
            SUR <= SUR + (frontbits & DIE);
         end if;
if CP ='1' then
            D <= TR2;
         else
            D <= TR1;
end if;

         if (D > "1100011") then
            WN <= '1';
          else
            WN <= '0';
         end if;

      end if;
   end if;
--! connection to displays
LEDDIE <= DIE;
DIGIT0 <= bcd1;
DIGIT1 <= bcd2;
DIGIT2 <= bcd3;
DIGIT3 <= bcd4;
end process;
```



Copyright ©2016 Pearson Education, All Rights Reserved

```
Main_process : process(clock,reset) begin
   if reset = '1' then
      -- reset
      DIE <= (others => '0'); --! asynchronous reset
   else
      if rising_edge(clock) then
         if RST1 = '1' then
            TR1 <= (others => '0');
         end if;
         if RST2 = '1' then
            TR2 <= (others => '0');
         end if;
         if RSSU = '1' then
            SUR <= (others => '0');
         end if;
         if LDT1 = '1' then
            TR1 <= TR1 + SUR;
         end if;
         if LDT2 = '1' then
            TR2 <= TR2 + SUR;
         end if;
         if ENADIE = '1' then
            case DIE is
               when "110" => DIE <= "001";
               when others => DIE <= DIE +1;
            end case;
         end if;
```

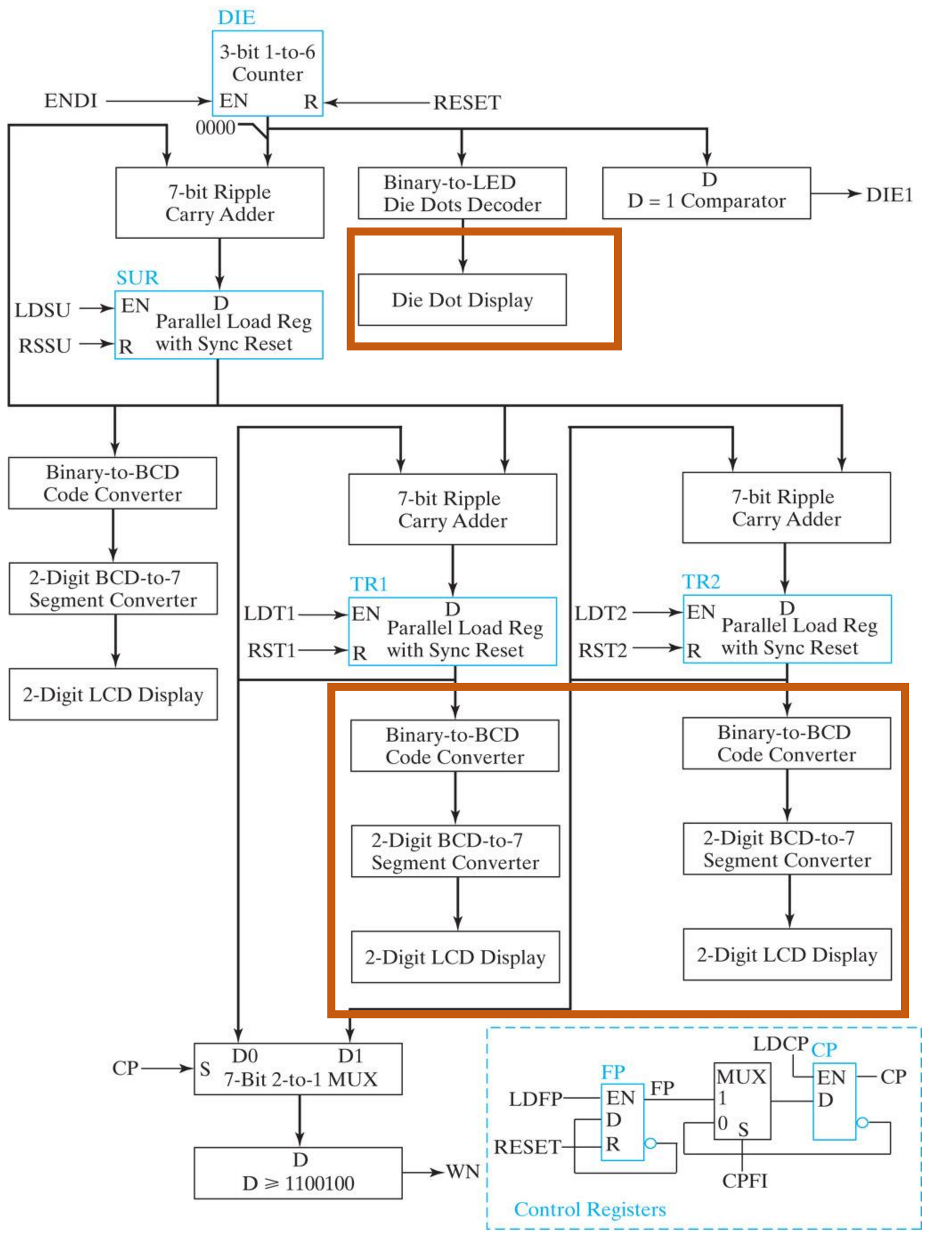
```
 if DIE ="001" then
            DIE1 <= '1';
         else
            DIE1 <= '0';
         end if;
         if LDSU = '1' then
            SUR <= SUR + (frontbits & DIE);
         end if;
         if CP ='1' then
            D <= TR2;
         else
            D <= TR1;
         end if;
if (D > "1100011") then
            WN <= '1';
          else
            WN <= '0';
 end if;

      end if;
   end if;
--! connection to displays
LEDDIE <= DIE;
DIGIT0 <= bcd1;
DIGIT1 <= bcd2;
DIGIT2 <= bcd3;
DIGIT3 <= bcd4;
end process;
```



Copyright ©2016 Pearson Education, All Rights Reserved

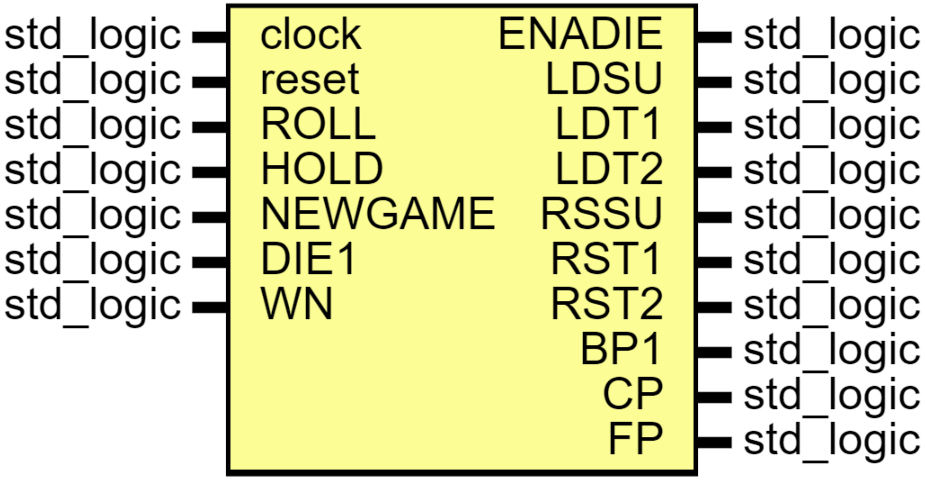
```
Main_process : process(clock,reset) begin
   if reset = '1' then
      -- reset
      DIE <= (others => '0'); --! asynchronous reset
   else
      if rising_edge(clock) then
         if RST1 = '1' then
            TR1 <= (others => '0');
         end if;
         if RST2 = '1' then
            TR2 <= (others => '0');
         end if;
         if RSSU = '1' then
            SUR <= (others => '0');
         end if;
         if LDT1 = '1' then
            TR1 <= TR1 + SUR;
         end if;
         if LDT2 = '1' then
            TR2 <= TR2 + SUR;
         end if;
         if ENADIE = '1' then
            case DIE is
               when "110" => DIE <= "001";
               when others => DIE <= DIE +1;
            end case;
         end if;
```

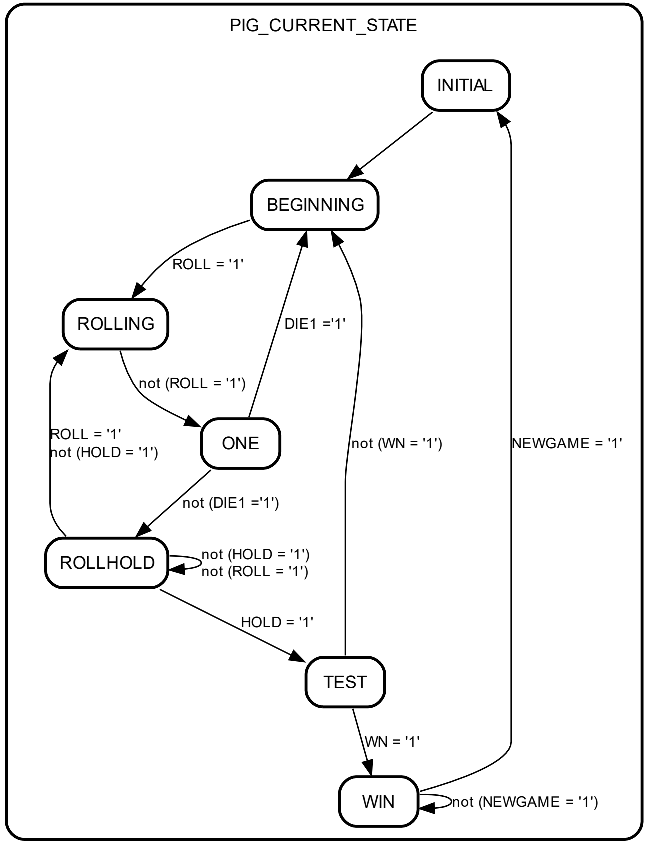
```
 if DIE ="001" then
            DIE1 <= '1';
         else
            DIE1 <= '0';
         end if;
         if LDSU = '1' then
            SUR <= SUR + (frontbits & DIE);
         end if;
         if CP ='1' then
            D <= TR2;
         else
            D <= TR1;
         end if;
if (D > "1100011") then
            WN <= '1';
          else
            WN <= '0';
 end if;

      end if;
   end if;
--! connection to displays
LEDDIE <= DIE;
DIGIT0 <= bcd1;
DIGIT1 <= bcd2;
DIGIT2 <= bcd3;
DIGIT3 <= bcd4;
end process;
```

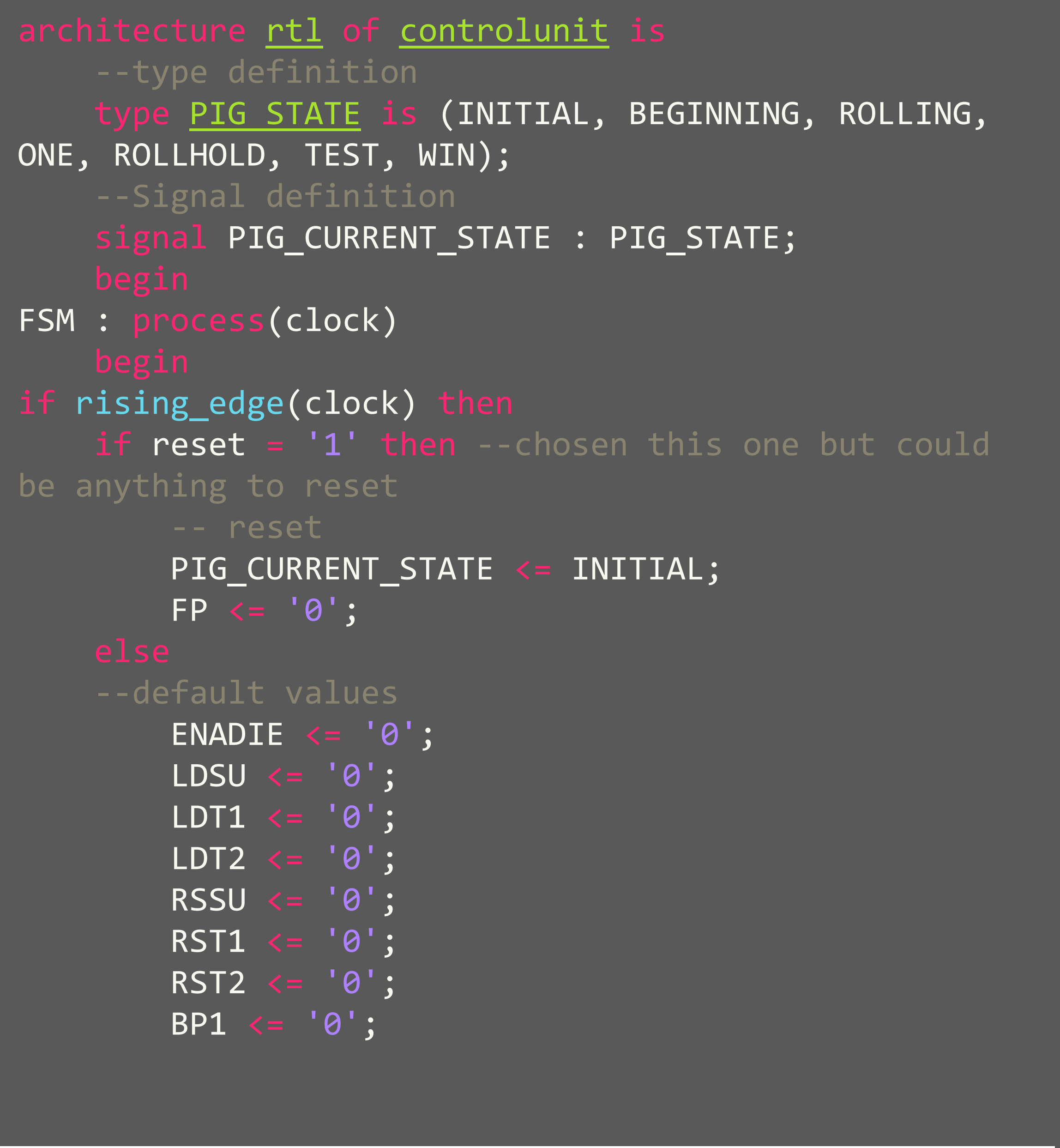


CONTROL UNIT

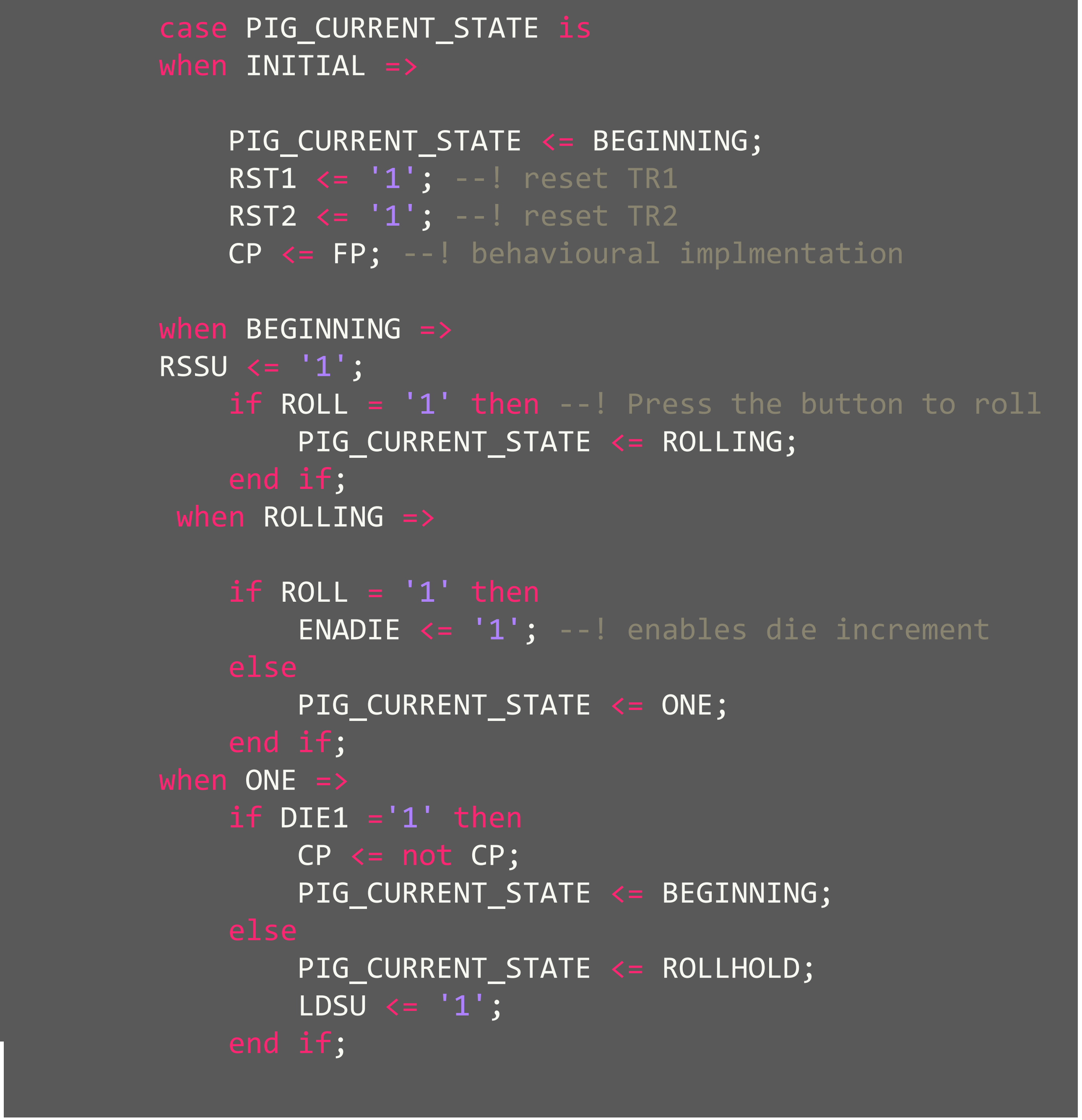
```
entity controlunit is
   port(
      clock  : in std_logic; --! Clock
      reset  : in std_logic; --! Reset
      ROLL   : in std_logic; --! button for the roll
      HOLD   : in std_logic; --! button for hold
      NEWGAME: in std_logic; --! button for new game
      ENADIE : out std_logic; --! Enable Die to increment
      LDSU   : out std_logic; --! Add DIE to SUR register
      LDT1   : out std_logic; --! Add SUR to TR1 register
      LDT2   : out std_logic; --! Add SUR to TR2 register
      RSSU   : out std_logic; --! Reset SUR register
      RST1   : out std_logic; --! Reset TR1 register
      RST2   : out std_logic; --! Reset TR2 register
      BP1    : out std_logic; --! enables blinking
      CP     : inout std_logic; --! current player (register outside)
      FP     : inout std_logic; --! First player (register outside)
      DIE1   : in std_logic; --! signal that the die is at one
      WN     : in std_logic --! WIN has been achieved by a player
   );
end entity controlunit;
```



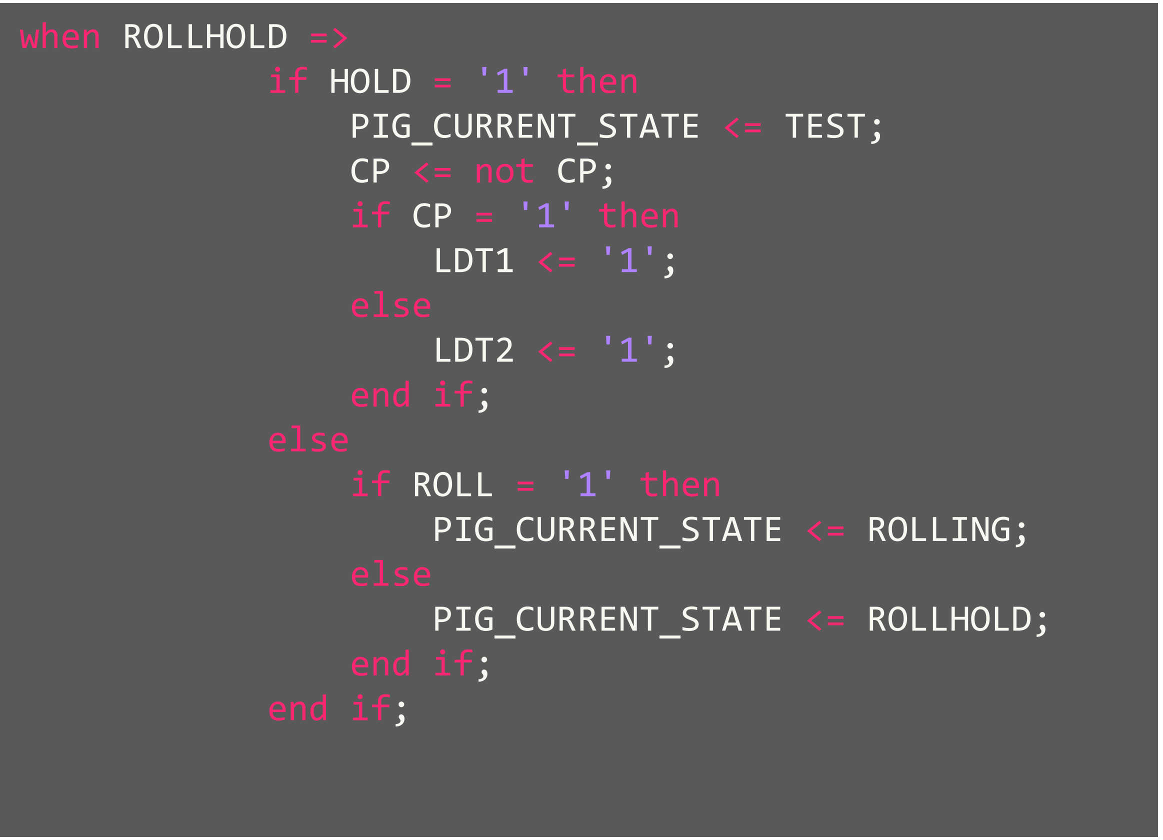




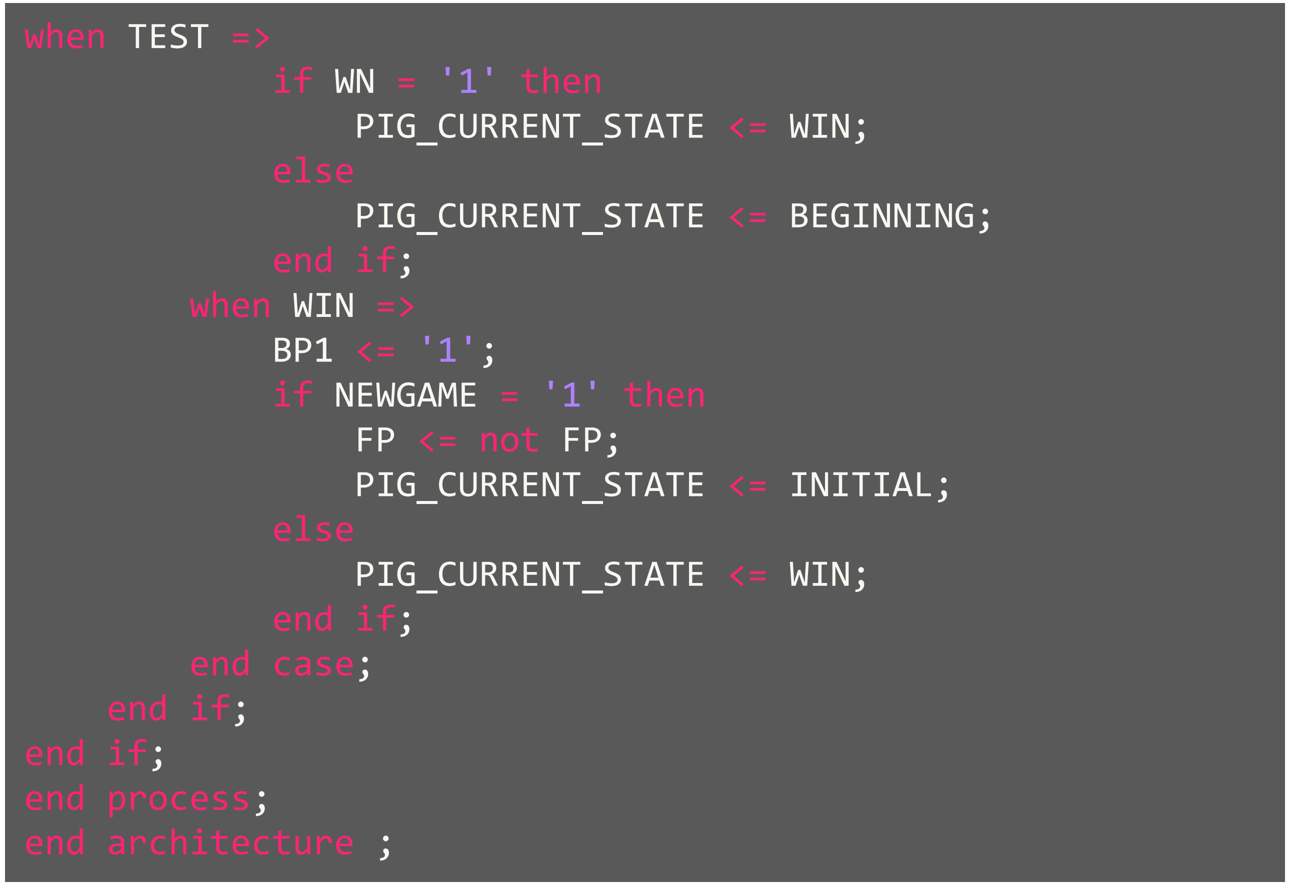
```
architecture rtl of controlunit is
    --type definition
    type PIG_STATE is (INITIAL, BEGINNING, ROLLING,
ONE, ROLLHOLD, TEST, WIN);
    --Signal definition
    signal PIG_CURRENT_STATE : PIG_STATE;
    begin
FSM : process(clock)
    begin
if rising_edge(clock) then
    if reset = '1' then --chosen this one but could
be anything to reset
        -- reset
        PIG_CURRENT_STATE <= INITIAL;
        FP <= '0';
    else
    --default values
        ENADIE <= '0';
        LDSU <= '0';
        LDT1 <= '0';
        LDT2 <= '0';
        RSSU <= '0';
        RST1 <= '0';
        RST2 <= '0';
        BP1 <= '0';
```



```
case PIG_CURRENT_STATE is
when INITIAL =>

    PIG_CURRENT_STATE <= BEGINNING;
    RST1 <= '1'; --! reset TR1
    RST2 <= '1'; --! reset TR2
    CP <= FP; --! behavioural implmentation

when BEGINNING =>
RSSU <= '1';
    if ROLL = '1' then --! Press the button to roll
        PIG_CURRENT_STATE <= ROLLING;
    end if;
 when ROLLING =>

    if ROLL = '1' then
        ENADIE <= '1'; --! enables die increment
    else
        PIG_CURRENT_STATE <= ONE;
    end if;
when ONE =>
    if DIE1 ='1' then
        CP <= not CP;
        PIG_CURRENT_STATE <= BEGINNING;
    else
        PIG_CURRENT_STATE <= ROLLHOLD;
        LDSU <= '1';
    end if;
```

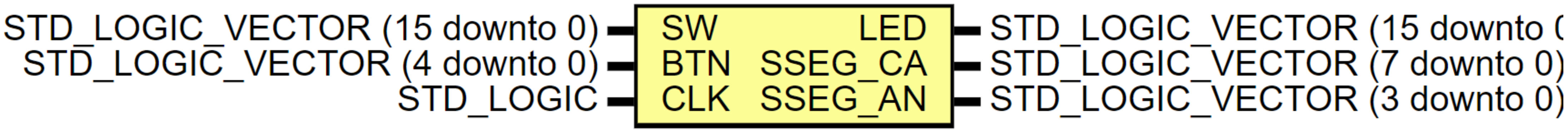


```vhdl
when ROLLHOLD =>
            if HOLD = '1' then
                PIG_CURRENT_STATE <= TEST;
                CP <= not CP;
                if CP = '1' then
                    LDT1 <= '1';
                else
                    LDT2 <= '1';
                end if;
            else
                if ROLL = '1' then
                    PIG_CURRENT_STATE <= ROLLING;
                else
                    PIG_CURRENT_STATE <= ROLLHOLD;
                end if;
            end if;
```



```vhdl
when TEST =>
            if WN = '1' then
                PIG_CURRENT_STATE <= WIN;
            else
                PIG_CURRENT_STATE <= BEGINNING;
            end if;
        when WIN =>
            BP1 <= '1';
            if NEWGAME = '1' then
                FP <= not FP;
                PIG_CURRENT_STATE <= INITIAL;
            else
                PIG_CURRENT_STATE <= WIN;
            end if;
        end case;
    end if;
end if;
end process;
end architecture ;
```

Main



STD_LOGIC_VECTOR (15 downto 0) SW LED STD_LOGIC_VECTOR (15 downto (

STD_LOGIC_VECTOR (4 downto 0) BTN SSEG_CA STD_LOGIC_VECTOR (7 downto 0)

STD_LOGIC CLK SSEG_AN STD_LOGIC_VECTOR (3 downto 0)

What’s inside?