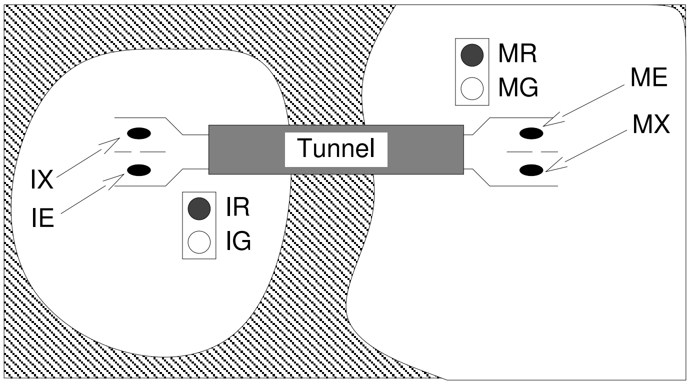
The Island Tunnel Controller

Steven D. Johnson and Kathryn Fisler
Computer Science Department
Indiana Univeristy
Bloomington, IN 47405-4101
USA
{sjohnson, kfisler}@cs.indiana.edu

This problem is an elaboration of the *traffic light control* problem and, as such, is a metaphor for coordinating access to a restricted resource. Concurrent entry/exit is allowed under certain conditions, represented by a one-lane tunnel between "The Mainland" and "The Island," that may at any time contain several vehicles, and an additional constraint on the number of vehicles that can be on The Island.



We want to design a controller for the traffic lights at the each end of this tunnel. There are four sensors for detecting the presence of vehicles: one at tunnel entrance on the island side (IE), one at tunnel exit on the island side (IX), one at tunnel entrance on the mainland side (ME), and one at tunnel exit on the mainland side (MX).

Environmental Properties

The controller design may assume the following constraints on its environment, described in terms of the behavior of vehicles that entering the system.

- E1 The vehicle sensors are conditioned signals, that is, clean pulses of significant duration, which may also be assumed synchronous in clocked designs.
- E2 Every vehicle that crosses a sensor is detected and every gap between vehicles that cross a sensor is detected.

E3 Vehicles going into the tunnel are detected before they reach the traffic light; vehicles leaving the tunnel are detected after they leave the single-lane portion of the roadway.

E4 Vehicles don't break down. They obey the traffic signals in finite time, and, if necessary, a bound may be placed on this time. Every vehicle that approaches the tunnel is committed to entering it.

E5 Having arrived on The Island every vehicle eventually tires to leave.

Design Properties

D1 *Light Behavior.* The traffic lights each follow a *red*, *green*, *red*, *green*, ... sequence. For simplicity, it is assumed that all vehicles can react instananeously to a change in the traffic light, so that *neither* "caution" lights nor duration timers are required, although these may be added if desired.

D2 *Safety.* At no time are there two vehicles in the tunnel that are traveling in opposite directions.

D3 *Liveness.* Any vehicle attempting to reach or leave the island eventually succeeds.

D3 *Access.* Traffic on the island is limited. Variants include:

- D3.1 At no time are there more than 16 vehicles on the island.
- D3.2 At no time are there more than N vhicles on the island, N a fixed but undetermined constant.
- D3.3 The system strives to keep the number of vehicles on the island below n, a dynamically varying input.

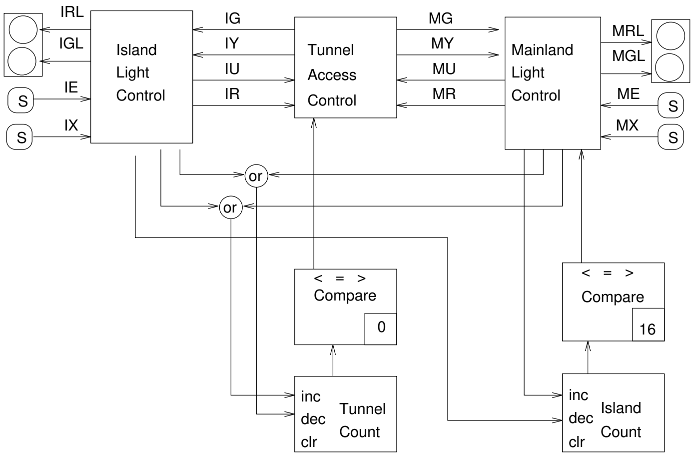


Figure 1: Three-process architecture of the Island Tunnel Control problem.

An ITC implementation

This implementation of the Island Tunnel Controller was conceived by Kathryn Fisler during work on visual inference research [2, 1]. It involves three processes, described below using *algorithmic state machine* (ASM) diagrams as developed in the textbook *The Art of Digital Design* [3]. A globally clocked synchronous model is assumed.

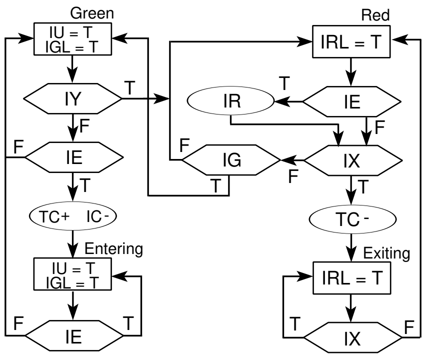
Additional control and status signals for the three-process solutions include the following: IGL and IRL are the green and red lights for the island, IU indicates that the island is using the tunnel, IR indicates that the island is requesting the tunnel, IY indicates that the island is being instructed to release control of the tunnel, and IG indicates that the island has been granted control of the tunnel.

A similar set of signals has been defined for the mainland.

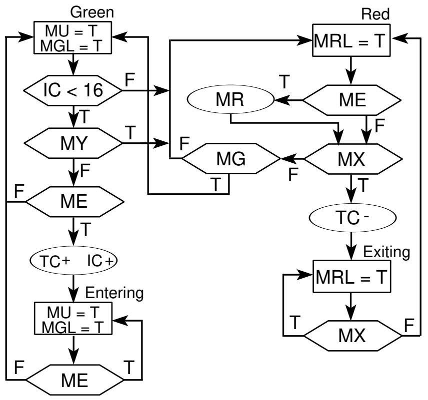
TC is a count of the number of cars presently inside the tunnel and IC is a count of the number of cars presently on the island.

In its intial/reset state, both light controllers are in their Red control states, both counters are zero, and no vehicles are in the system. Alternatively, the counters correctly reflect what vehicles are in the system.

- **Island Light Controller**



- **Mainland Light Controller**



- **Tunnel Controller**

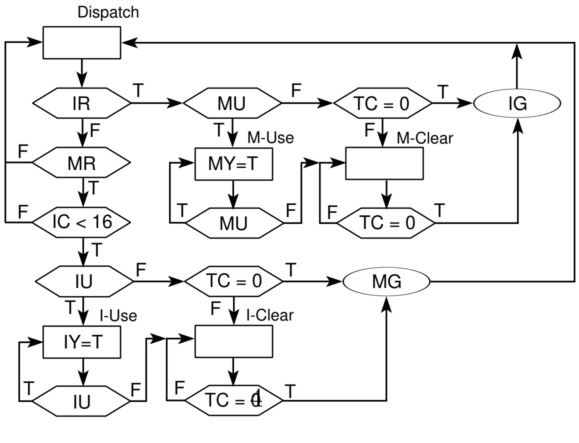


Figure 2: Three-process implementation of the Island Tunnel Control problem.

References

[1] Kathryn Fisler. Exploiting the potential of diagrams in guiding hardware reasoning. In Gerard Allwein and Jon Barwise, editors, *Logical Reasoning with Diagrams*. Oxford University Press, 1996. In press.

[2] Kathryn Fisler and Steven D. Johnson. Integrating design and verification envrionments through a logic supprting hardware diagrams. In *Procedings of the 1995 IFIP International Conference on Computer Hardware Description Languages and Their Applications*, pages 669–674. IEEE Cat. No. 95TH8102, September 1995. CHDL proceedings pp. 493-696 of the "ACV'95" held August 29 to September 1, 1995, Chiba, Japan.

[3] Franklin Prosser and David Winkel. *The Art of Digital Design*. Prentice Hall, Englewood Cliffs NJ, second edition, 1987.

Formal verification if ITC in SMV

This is the source file for a verification of the Island Tunnel Controller implementation by Kathryn Fisler. The implementation is described in Section 2.

```
-- Kathi Fisler
-- February 7, 1995
-- Island Traffic Light Controller, version 3
-- Used corrected environmental assumption as proposed by ratan

MODULE main
VAR
  istate : {igreen, ired, icar_entering, icar_exiting};
  mstate : {mgreen, mred, mcar_entering, mcar_exiting};
  tstate : {dispatch, island_use, mainland_use, iclear, mclear};
  IGL : boolean; -- the lights
  IRL : boolean;
  MGL : boolean;
  MRL : boolean;
  IU : boolean; -- in use
  MU : boolean;
  IE : boolean; -- entering tunnel sensors
  ME : boolean;
  IX : boolean; -- exiting tunnel sensors
  MX : boolean;
  IR : boolean; -- request tunnel
  MR : boolean;
  IG : boolean; -- tunnel granted
  MG : boolean;
  IY : boolean; -- tunnel yield
  MY : boolean;
  TC : {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16} ;
  IC : {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16} ;


ASSIGN
  init(TC) := 0;

-- island controller

  init(istate) := ired;
  next(istate) :=
     case
        istate = igreen & IY : ired;
```

```
        istate = igreen & !IY & !IE : igreen;
        istate = igreen & !IY & IE : icar_entering;

        istate = icar_entering & IE : icar_entering;
        istate = icar_entering & !IE : igreen;

        istate = ired & !IX & !IG : ired;
        istate = ired & !IX & IG : igreen;
        istate = ired & IX : icar_exiting;

        istate = icar_exiting & IX : icar_exiting;
        istate = icar_exiting & !IX : ired;
      esac;

  IGL := istate = igreen | istate = icar_entering;
  IRL := istate = ired | istate = icar_exiting;
  IU := istate = igreen | istate = icar_entering;
  IR := istate = ired & IE;

-- mainland controller
  init(mstate) := mred;
  next(mstate) :=
      case
        mstate = mgreen & !(IC < 16) : mred;
        mstate = mgreen & (IC < 16) & MY : mred;
        mstate = mgreen & (IC < 16) & !MY & !ME : mgreen;
        mstate = mgreen & (IC < 16) & !MY & ME : mcar_entering;

        mstate = mcar_entering & ME : mcar_entering;
        mstate = mcar_entering & !ME : mgreen;

        mstate = mred & !MX & !MG : mred;
        mstate = mred & !MX & MG : mgreen;
        mstate = mred & MX : mcar_exiting;

        mstate = mcar_exiting & MX : mcar_exiting;
        mstate = mcar_exiting & !MX : mred;
      esac;

  MGL := mstate = mgreen | mstate = mcar_entering;
  MRL := mstate = mred | mstate = mcar_exiting;
  MU := mstate = mgreen | mstate = mcar_entering;
  MR := mstate = mred & ME;
```

```
-- tunnel controller
  init(tstate) := dispatch;
  next(tstate) :=
     case
        tstate = dispatch & !IR & !MR : dispatch;
        tstate = dispatch & IR & MU : mainland_use;
        tstate = dispatch & IR & !MU & (TC = 0) : dispatch;
        tstate = dispatch & IR & !MU & !(TC = 0) : mclear;
        tstate = dispatch & !IR & MR & !(IC < 16) : dispatch;
        tstate = dispatch & !IR & MR & (IC < 16) & IU : island_use;
        tstate = dispatch & !IR & MR & (IC < 16) & !IU & (TC = 0) : dispatch;
        tstate = dispatch & !IR & MR & (IC < 16) & !IU & !(TC = 0) : iclear;

        tstate = island_use & IU : island_use;
        tstate = island_use & !IU : iclear;

        tstate = mainland_use & MU : mainland_use;
        tstate = mainland_use & !MU : mclear;

        tstate = iclear & !(TC = 0) : iclear;
        tstate = iclear & (TC = 0) : dispatch;

        tstate = mclear & !(TC = 0) : mclear;
        tstate = mclear & (TC = 0) : dispatch;
     esac;

  IY := tstate = island_use;
  MY := tstate = mainland_use;
  IG := (tstate = dispatch & IR & !MU & (TC = 0)) |
        (tstate = mclear & (TC = 0));
  MG := (tstate = dispatch & !IR & MR & (IC < 16) & !IU & (TC = 0)) |
        (tstate = iclear & (TC = 0));

DEFINE
  tcincr := (istate = igreen & !IY & IE) |
            (mstate = mgreen & !MY & ME & (IC < 16));
  tcdecr := (istate = ired & IX) | (mstate = mred & MX);
  icdecr := istate = igreen & !IY & IE;
  icincr := mstate = mgreen & !MY & ME;

  tunnel_empty := TC = 0;

ASSIGN
  next(TC) :=
```

```
    case
      tcincr & (TC < 16) : TC + 1;
      tcdecr & (TC > 0) : TC - 1;
      1 : TC;
    esac;

  next(IC) :=
    case
      icdecr & (IC > 0) : IC - 1;
      icincr & (IC < 16): IC + 1;
      1 : IC;
    esac;

DEFINE
  env1 := (IE -> AF MX);
  env2 := A[!MX U IE];
  env3 := (ME -> AF IX);
  env4 := A[!IX U ME];
  env := env1 & env2 & env3 & env4;

  conda := !((istate = icar_exiting) & (mstate = mcar_exiting));

FAIRNESS IE
FAIRNESS ME
FAIRNESS !IE
FAIRNESS !ME

-- a test case to see how things are working
SPEC (AF conda) | A[conda U !env]

-- access never granted while cars in tunnel
SPEC AG ((IG | MG) -> (TC = 0))

-- tunnel used once granted
SPEC (AF (IG -> AF IU)) | A[(IG -> AF IU) U !env]
SPEC (AF (MG -> AF MU)) | A[(MG -> AF MU) U !env]

-- requests eventually granted
SPEC (AF (IR -> AF IG)) | A[(IR -> AF IG) U !env]
SPEC (AF (MR -> AF MG)) | A[(MR -> AF MG) U !env]

-- only change counters once per car
SPEC (AF ((IE & tcincr) -> A[!tcincr U !IE]))
     | A[((IE & tcincr) -> A[!tcincr U !IE]) U !env]
```

```
-- not incrementing and decrementing counters simultaneously
--SPEC AG !(tcincr & tcdecr)
SPEC AG !(icincr & icdecr)

-- never cars travelling both directions at once
SPEC AG (((IE <-> AX (AF MX)) & (ME <-> AX (AF IX))) ->
                (((!IGL & AX IGL) -> tunnel_empty) &
                 ((!MGL & AX MGL) -> tunnel_empty) &
                 !(IGL & MGL)))

-- more generic liveness
SPEC (AF (IE -> AF IGL)) | A[(IE -> AF IGL) U !env]
SPEC (AF (ME -> AF MGL)) | A[(ME -> AF MGL) U !env]

-- never more than 16 cars on the island
SPEC (AF (IC <= 16)) | A[(IC <= 16) U !env]
SPEC (AF (icincr -> (IC < 16))) | A[(icincr -> (IC < 16)) U !env]
--SPEC (AF (((IC = 16) & ME) -> A[!MGL U (IC < 16)]))
--     | A[(((IC = 16) & ME) -> A[!MGL U (IC < 16)]) U !env]
SPEC (AF (TC + IC <= 16)) | A[(TC + IC <= 16) U !env]

-- yields acknowledged
SPEC (AF (IY -> AF !IU)) | A[(IY -> AF !IU) U !env]
SPEC (AF (MY -> AF !MU)) | A[(MY -> AF !MU) U !env]
```

VHDL description of an ITC implementation

This is a VHDL source file for the Island Tunnel Controller by Kathryn Fisler. The VHDL comes from a verification front end under development at (what used to be called) AT&T Bell Laboratories. For details contact `kfisler@cs.indiana.edu`.

```
Library IEEE, DZX;
Use IEEE.std_logic_1164.all;
Use DZX.Logic_Utils.all;
Use DZX.Bit_Arith.all;
Use DZX.Attributes.all;

entity ITLC is
  port (IX : in X01Z := '0';  -- island exit sensor
IE : in X01Z := '0';  -- island entrance sensor
MX : in X01Z := '0';  -- mainland exit sensor
ME : in X01Z := '0';  -- mainland entrance sensor
CLK : in X01Z := '0';  -- system clock
RESET : in X01Z;    -- system wide reset
MRL : out X01Z := '0'; -- mainland red light
MGL : out X01Z := '0'; -- mainland green light
IRL : out X01Z := '0'; -- island red light
IGL : out X01Z := '0'  -- island green light
       );
end ITLC;

architecture itlc_arch of ITLC is
  signal IU, MU, IY, MY, IR, MR, IG, MG, TCInc, TCDec, ICInc, ICDec : X01Z := '0';
  signal TCEnb, TCDU, ICEng, ICDU : X01Z := '0';
  signal IC_val, TC_val : integer range 0 to 16;

  component Side_CNTL
    port(CLK, RESET. SE, SX, SG, SY : in X01Z;
      Res_Level : in integer range 0 to 16;
 SRL, SGL, SU, SR, STCD, STCI, SIC : out X01Z );
  end component;

  component Tunnel_CNTL
    port(CLK, RESET, IR, MR, IU, MU : in X01Z;
 IC_val, TC_val : in integer range 0 to 16;
 IY, MY, IG, MG : out X01Z );
  end component;
```

```
  component UD_CNT
    port(CLK, RESET, Enable, UD : in X01Z;
 Q : out integer range 0 to 16;
  end component;

begin

ICNTL : Side_CNTL port map (CLK,RESET,IE,IX,IG,IY,0,IRL,IGL,IU,IR,ITCD,ITCI,IIC);
MCNTL : Side_CNTL port map (CLK,RESET,ME,MX,MG,MY,IC_val,MRL,MGL,MU,MR,MTCD,MTCI,MIC);
TCNTL : Tunnel_CNTL port map (CLK,RESET,IR,MR,IU,MU,IC_val,TC_val,IY,MY,IG,MG);
TC    : UD_CNT port map (CLK,RESET,TCEnb,TCDU,TC_val);
IC    : UD_CNT port map (CLK,RESET,ICEnb,ICDU,IC_val);

process(ITCI, ITCD, MTCI, MTCD)

begin
  TCDU <= ITCD or MTCD;
  TCEnb <= not ((ITCI or MTCI) xor (ITCD or MTCD));
end process;

process(IIC, MIC)
  ICDU <= IIC;
  ICEnb <= not (MIC or IIC);
end process;

end itlc_arch;Library IEEE, DZX;
Use IEEE.std_logic_1164.all;
Use DZX.Logic_Utils.all;
Use DZX.Bit_Arith.all;
Use DZX.Attributes.all;

entity Side_CNTL is
  port (CLK   : in X01Z := '0'; -- system clock
   RESET   : in X01Z;     -- system wide reset
SE    : in X01Z := '0'; -- entrance sensor
SX     : in X01Z := '0'; -- exit sensor
SG    : in X01Z := '0'; -- grant signal
SY    : in X01Z := '0'; -- yield signal
Res_Level : in integer range 0 to 16; -- value of max resource usage
SRL      : out X01Z := '0'; -- red light signal
SGL      : out X01Z := '0'; -- green light signal
SU      : out X01Z := '0'; -- in use signal
SR      : out X01Z := '0'; -- request signal
STCD      : out X01Z := '0'; -- tunnel counter decrement
```

```
STCI      : out X01Z := '0'; -- tunnel counter increment
SIC    : out X01Z := '0'; -- island counter change
       );
end Side_CNTL;

architecture side_arch of Side_CNTL is
  type STATE_TYPE is (RED, EXITING, GREEN, ENTERING);
  signal state, new_state : STATE_TYPE;

  process(SX, SY, SE, Res_Level, SG)
  begin
  case state is
      when RED =>
SRL <= '1';
  SGL <= '0';
SU <= '0';
STCI <= '0';
SIC <= '0';
if SX = '1' then
   new_state <= EXITING;
   STCD <= '1';
        else
   STCD <= '0';
   if SG = '1' then
      new_state <= GREEN;
   else
      new_state <= RED;
        end if;
    end if;

      when EXITING =>
SRL <= '1';
  SGL <= '0';
SU <= '0';
STDC <= '0';
STCI <= '0';
SIC <= '0';
if SX = '1' then
   new_state <= EXITING ;
else
   new_state <= RED;
end if;

      when GREEN =>
```

```
SRL <= '0';
  SGL <= '1';
SU <= '1';
STCD <= '0';
if Res_Level /= "1111" then
   if SY = '0' then
      if SE = '1' then
 new_state <= ENTERING;
 STCI <= '1';
 SIC <= '1';
      else
 new_state <= GREEN;
 STCI <= '0';
 SIC <= '0';
      end if;
   else
      new_state <= RED;
      STCI <= '0';
      SIC <= '0';
   end if;
        else
   new_state <= RED;
   STCI <= '0';
   SIC <= '0';
    end if;

      when ENTERING =>
SRL <= '0';
  SGL <= '1';
SU <= '1';
STCD <= '0';
STCI <= '0';
SIC <= '0';
       if SE = '1' then
   new_state <= ENTERING;
else
   new_state <= GREEN;
end if;

      end case;
  end process;

  process(RESET, CLK, new_state)
  begin
```

```
      if RESET = '1' then
state <= RED ;
      else
if CLK'event and CLK = '1' then
    state <= new_state;
end if;
      end if;
  end process;

end side_arch;Library IEEE, DZX;
Use IEEE.std_logic_1164.all;
Use DZX.Logic_Utils.all;
Use DZX.Bit_Arith.all;
Use DZX.Attributes.all;

entity Tunnel_CNTL is
  port (CLK   : in X01Z := '0'; -- system clock
   RESET   : in X01Z;     -- system wide reset
IR    : in X01Z := '0'; -- island requested tunnel
MR     : in X01Z := '0'; -- mainland requested tunnel
IU    : in X01Z := '0'; -- island using tunnel
MU    : in X01Z := '0'; -- mainland using tunnel
IC_val   : in integer range 0 to 16; -- value of island counter
TC_val   : in integer range 0 to 16; -- value of tunnel counter
IY      : out X01Z := '0'; -- island yield
MY      : out X01Z := '0'; -- mainland yield
IG      : out X01Z := '0'; -- island granted tunnel
MG      : out X01Z := '0'  -- mainland granted tunnel
       );
end Tunnel_CNTL;

architecture tunn_arch of Tunnel_CNTL is
  type STATE_TYPE is (DISPATCH, IUSE, MUSE, ICLEAR, MCLEAR);
  signal state, new_state : STATE_TYPE;

  process()
  begin
  case state is
      when DISPATCH =>
IY <= '0';
MY <= '0';
if IR = '0' then
   IG <= '0';
   if MR = '1' and IC_val < 16 then
```

```
      if IU = '1' then
 new-state <= IUSE;
 MG <= '0';
      else
 if TC_val = 0 then
    new_state <= DISPATCH;
    MG <= '1';
 else
    new_state <= ICLEAR;
    MG <= '0';
 end if;
      end if;
   else
      new_state <= DISPATCH;
      IG <= '0';
      MG <= '0';
   end if;
else
   MG <= '0';
   if MU = '1' then
      new_state <= MUSE;
 IG <= '0';
   else
      if TC_val = 0 then
 new_state <= DISPATCH;
 IG <= '1';
      else
 new_state <= MCLEAR;
 IG <= '0';
      end if;
   end if;
end if;

      when IUSE =>
IY <= '1';
MY <= '0';
IG <= '0';
MG <= '0';
if IU = '1' then
   new_state <= IUSE;
else
   new_state <= ICLEAR;
end if;
```

```
      when MUSE =>
IY <= '0';
MY <= '1';
IG <= '0';
MG <= '0';
if MU = '1' then
   new_state <= MUSE;
else
   new_state <= MCLEAR;
end if;

      when ICLEAR =>
IY <= '0';
MY <= '0';
IG <= '0';
if TC_val = 0 then
   new_state <= DISPATCH;
   MG <= '1';
else
   new_state <= ICLEAR;
   MG <= '0';
end if;

      when MCLEAR =>
IY <= '0';
MY <= '0';
MG <= '0';
if TC_val = 0 then
   new_state <= DISPATCH;
   IG <= '1';
else
   new_state <= MCLEAR;
   IG <= '0';
end if;

      end case;
  end process;

  process(RESET, CLK, new_state)
  begin
      if RESET = '1' then
state <= DISPATCH ;
      else
if rising_edge(CLK) then
```

```
    state <= new_state;
end if;
      end if;
  end process;

end tunn_arch;Library IEEE, DZX;
Use IEEE.std_logic_1164.all;
Use DZX.Logic_Utils.all;
Use DZX.Bit_Arith.all;
Use DZX.Attributes.all;

entity UD_CNT is
  port (CLK : in X01Z := '0';  -- system clock
  RESET : in X01Z;    -- system reset
Enable : in X01Z;    -- counter enable
UD : in X01Z := '0';  -- count up (if 0) or count down (if 1)
Q : out integer range 0 to 16
       );
end UD_CNT;

architecture cnt_arch of UD_CNT is
  constant MAXVAL : integer := 16;
  constant MINVAL : integer := 0;
begin
  process (CLK, Enable, UD)
    variable CNTR_VALUE : integer range 0 to 16;
    begin
     if RESET = '1' then
   CNTR_VALUE := 0;
elsif rising_edge(CLK) then
   if Enable = '0' then
      if UD = '0'
 if CNTR_VALUE < UPPERBOUND then
    CNTR_VALUE := CNTR_VALUE + 1;
 else
    CNTR_VALUE := CNTR_VALUE;
    end if;
      else
 if CNTR_VALUE > LOWERBOUND then
    CNTR_VALUE := CNTR_VALUE - 1;
 else
    CNTR_VALUE := CNTR_VALUE;
    end if;
      end if;
```

```
   else
      CNTR_VALUE := CNTR_VALUE;
   end if;
end if;
Q <= CNTR_VALUE;
    end process;
end cnt_arch
```