People Counter Example

Construct an Esterel program that counts the number of people in a room. People enter the room from one door with a photocell that changes from 0 to 1 when the light is interrupted, and leave from a second door with a similar photocell. These inputs may be true for more than one clock cycle.

The two photocell inputs are called ENTER and LEAVE. There are two outputs: EMPTY and FULL, which are present when the room is empty and contains three people respectively.

Overall Structure

Conditioner detects rising edges of signal from photocell. Counter tracks number of people in the room.
Implementing the Conditioner

module Conditioner:
input A;
output Y;
loop
  await A; emit Y;
  await [not A];
end
end module
Testing the Conditioner

```bash
# esterel -simul cond.strl
# gcc -o cond cond.c -lcsimul  # may need -L
# ./cond
Conditioner> ;
--- Output:
Conditioner> A;  # Rising edge
--- Output: Y
Conditioner> A;  # Doesn't generate a pulse
--- Output:
Conditioner> ;  # Reset
--- Output:
Conditioner> A;  # Another rising edge
--- Output: Y
Conditioner> ;  # Reset
--- Output:
Conditioner> A;  # Another rising edge
--- Output: Y
```
module Counter:
  input ADD, SUB;
  output FULL, EMPTY;

  var count := 0 : integer in
    loop
      present ADD then if count < 3 then
        count := count + 1 end end;
      present SUB then if count > 0 then
        count := count - 1 end end;
      if count = 0 then emit EMPTY end;
      if count = 3 then emit FULL end;
      pause
    end
  end
end module
Testing the Counter

Counter> ;
--- Output: EMPTY
Counter> ADD SUB;
--- Output: EMPTY
Counter> ADD;
--- Output: ADD
Counter> SUB;
--- Output: EMPTY
Counter> ADD;
--- Output: ADD
Counter> ADD;
--- Output: ADD
Counter> ADD;
--- Output: ADD
Counter> ADD;
--- Output: FULL
Counter> ADD SUB;
--- Output: # Oops: still FULL
module Counter:
  input ADD, SUB;
  output FULL, EMPTY;

  var c := 0 : integer in
  loop
    present ADD then
      present SUB else
        if c < 3 then c := c + 1 end
      end
    else
      present SUB then
        if c > 0 then c := c - 1 end
      end;
    end;
  if c = 0 then emit EMPTY end;
  if c = 3 then emit FULL end;
  pause
  end
end module
Testing the second counter

Counter> ;
  --- Output: EMPTY
Counter> ADD SUB;
  --- Output: EMPTY
Counter> ADD SUB;
  --- Output: EMPTY
Counter> ADD;
  --- Output: 
Counter> ADD;
  --- Output: 
Counter> ADD;
  --- Output: FULL
Counter> ADD SUB;
  --- Output: FULL
Counter> ADD SUB;
  --- Output: FULL
Counter> SUB;
  --- Output: 
Counter> SUB;
  --- Output: 
Counter> SUB;
  --- Output: EMPTY
Counter> SUB;
  --- Output: EMPTY
Assembling the People Counter

module PeopleCounter:
input ENTER, LEAVE;
output EMPTY, FULL;

signal ADD, SUB in
  run Conditioner[signal ENTER / A,
                  ADD / Y]
||
  run Conditioner[signal LEAVE / A,
                  SUB / Y]
||
  run Counter
end

end module
Vending Machine Example

Design a vending machine controller that dispenses gum once. Two inputs, N and D, are present when a nickel and dime have been inserted, and a single output, GUM, should be present for a single cycle when the machine has been given fifteen cents. No change is returned.

N =  
D =  
GUM =  

module Vending:
input N, D;
output GUM;

loop
  var m := 0 : integer in
  trap WAIT in
  loop
    present N then m := m + 5; end;
    present D then m := m + 10; end;
    if m >= 15 then exit WAIT end;
  pause
  end
end
emit GUM; pause
end
end module
Alternative Solution

loop
  await
    case immediate N do await
    case N do await
      case N do nothing
      case immediate D do nothing
    end
    case immediate D do nothing
  end
  case immediate D do await
  case immediate immediate N do nothing
  case D do nothing
end
end;
emit GUM; pause
Tail Lights Example

Construct an Esterel program that controls the turn signals of a 1965 Ford Thunderbird.

Tail Light Behavior
Tail Lights

There are three inputs, LEFT, RIGHT, and HAZ, that initiate the sequences, and six outputs, LA, LB, LC, RA, RB, and RC. The flashing sequence is:

<table>
<thead>
<tr>
<th>LC</th>
<th>LB</th>
<th>LA</th>
<th>step</th>
<th>RA</th>
<th>RB</th>
<th>RC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The sequence is LC LB LA step RA RB RC.
module Lights:
output A, B, C;

loop
emit A; pause;
emit A; emit B; pause;
emit A; emit B; emit C; pause;
pause
end

end module
The T-Bird Controller Interface

module Thunderbird :
input LEFT, RIGHT, HAZ;
output LA, LB, LC, RA, RB, RC;
...
end module
The T-Bird Controller Body

loop
    await
    case immediate HAZ do
        abort
        run Lights[signal LA/A, LB/B, LC/C]
    || run Lights[signal RA/A, RB/B, RC/C]
        when [not HAZ]
    case immediate LEFT do
        abort
        run Lights[signal LA/A, LB/B, LC/C]
        when [not LEFT]
    case immediate RIGHT do
        abort
        run Lights[signal RA/A, RB/B, RC/C]
        when [not RIGHT]
end
end
Comments on the T-Bird

I choose to use Esterel’s innate ability to control the execution of processes, producing succinct easy-to-understand source but a somewhat larger executable.

An alternative: Use signals to control the execution of two processes, one for the left lights, one for the right.

A challenge: synchronizing hazards.

Most communication signals can be either level- or edge-sensitive.

Control can be done explicitly, or implicitly through signals.
This controls a traffic light at the intersection of a busy highway and a farm road. Normally, the highway light is green but if a sensor detects a car on the farm road, the highway light turns yellow then red. The farm road light then turns green until there are no cars or after a long timeout. Then, the farm road light turns yellow then red, and the highway light returns to green. The inputs to the machine are the car sensor \( C \), a short timeout signal \( S \), and a long timeout signal \( L \). The outputs are a timer start signal \( R \), and the colors of the highway and farm road lights.

The Traffic Light Controller

module Fsm:

input C, L, S;
output R;
output HG, HY, FG, FY;

loop
  emit HG ; emit R; await [C and L];
  emit HY ; emit R; await S;
  emit FG ; emit R; await [not C or L];
  emit FY ; emit R; await S;
end

end module
module Timer:
input R, SEC;
output L, S;

loop
  weak abort
  await 3 SEC;
  [sustain S
  ||
  await 5 SEC;
  sustain L
  ]
  when R;
end

end module
module TLC:
input C, SEC;
output HG, HY, FG, FY;

signal S, L, S in
    run Fsm
||
    run Timer
end

end module