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.


Implementing the Conditioner

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

Testing the Conditioner

# esterel -simul cond.strl
# gcc -o cond cond.c -lcsimul # may need -L ./cond
Conditioner> ;
--- Output: Y
--- Output: A; # Rising edge
--- Output: Y
--- Output: A; # Doesn't generate a pulse
--- Output: ; # Reset
--- Output: Y
--- Output: A;
--- Output: ;
--- Output: Y
--- Output: A;
--- Output: Y

Implementing the Counter: First Try

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
endmodule

Testing the Counter

Counter> ;
--- Output: EMPTY
--- Output: ADD SUB;
--- Output: EMPTY
--- Output: ADD;
--- Output: FULL
--- Output: ADD;
--- Output: FULL;
--- Output: ;
--- Output: # Oops: still FULL

Counter, second try

module Counter:
input ADD, SUB;
output FULL, EMPTY;
var c := 0 : integer in
loop
  present ADD then if c < 3 then c := c + 1 end end;
  present SUB else
    if c > 0 then c := c - 1 end
  end;
  if c = 0 then emit EMPTY end;
  if c = 3 then emit FULL end;
  pause
end
endmodule

Testing the second counter

Counter> ;
--- Output: EMPTY
--- Output: ADD SUB;
--- Output: EMPTY
--- Output: ADD;
--- Output: FULL
--- Output: ADD;
--- Output: FULL;
--- Output: ADD;
--- Output: FULL;
--- Output: ;
--- Output: # Working
--- Output: ADD SUB;
--- Output: FULL
--- Output: ADD;
--- Output: FULL;
--- Output: COMPLETE
--- Output: SUB;
--- Output: COMPLETE
--- Output: EMPTY
--- 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
endmodule

### 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.

\[
\begin{align*}
N &= \quad D = \quad \\
\text{GUM} &= \quad \\
\end{align*}
\]

Source: Katz, Contemporary Logic Design, 1994, p. 389

### Vending Machine Solution

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
  emit GUM; pause
endmodule

### Alternative Solution

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

### Tail Lights Example

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

\[
\begin{align*}
\text{Tail Lights} & \quad \text{There are three inputs, LEFT, RIGHT, and HAZ, that initiate the sequences, and six outputs, LA, LB, LC, RA, RB, and RC. The flashing sequence is} \\
\text{A Single Tail Light} & \quad \text{module Lights:} \\
\text{The T-Bird Controller Interface} & \quad \text{module Thunderbird :}
\end{align*}
\]

### Tail Light Behavior

Tail Lights

<table>
<thead>
<tr>
<th>LC</th>
<th>LB</th>
<th>LA</th>
<th>RA</th>
<th>RB</th>
<th>RC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

Traffic-Light Controller Example

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.