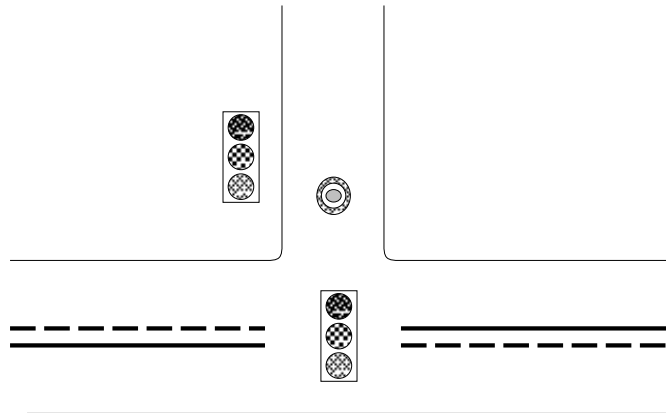
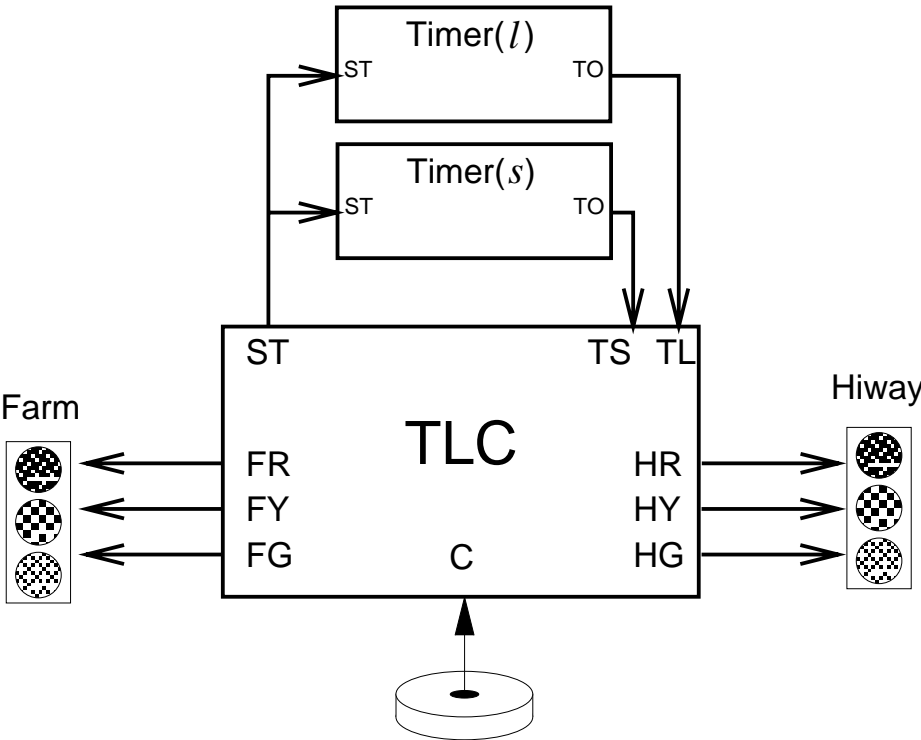


Traffic Light Controller

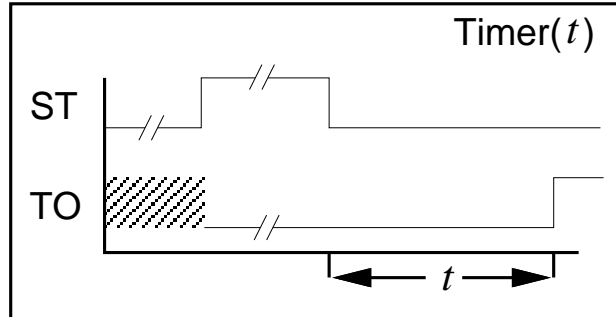
We want to design a controller for a traffic light at the intersection of a side-road, and a more heavily used “highway”. There is a sensor on the side road to detect the presence of a vehicle, and there are two signals to control flow.



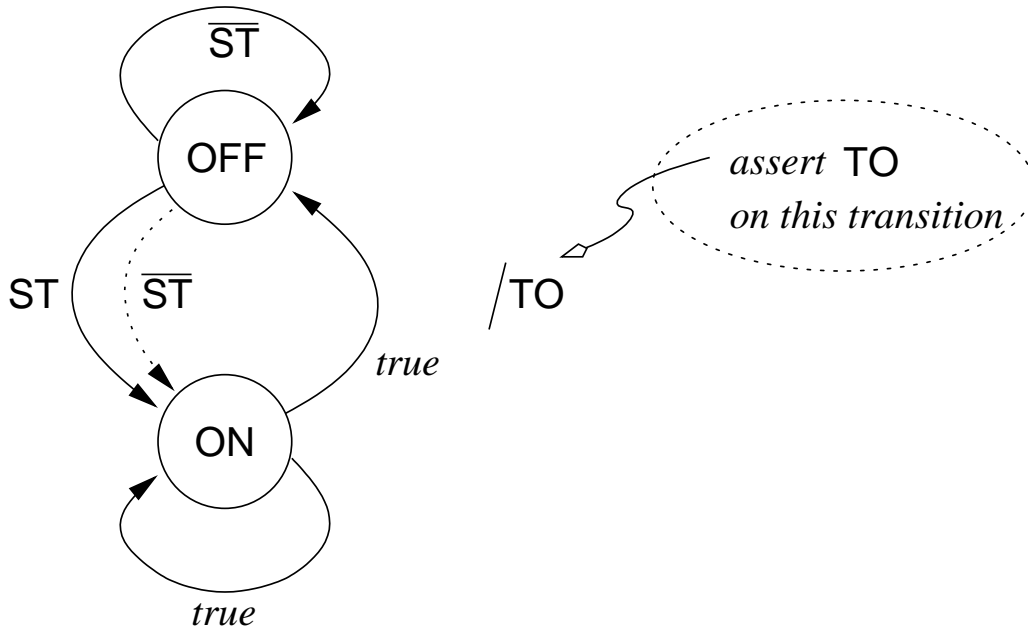
The architecture of the design includes the sensor input and signal outputs, and a pair of timers is used to determine the period when a light is on.



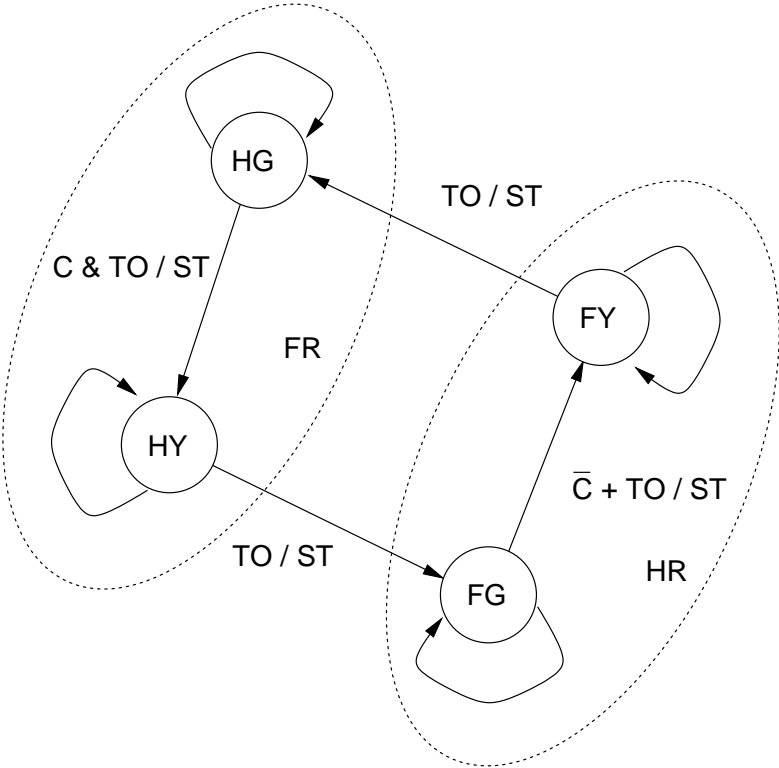
All we need to know about a timer is that, if the controller asserts a *start* signal (ST), then after a certain period of time, t , the *time-out* alarm (TO) is asserted by the timer.



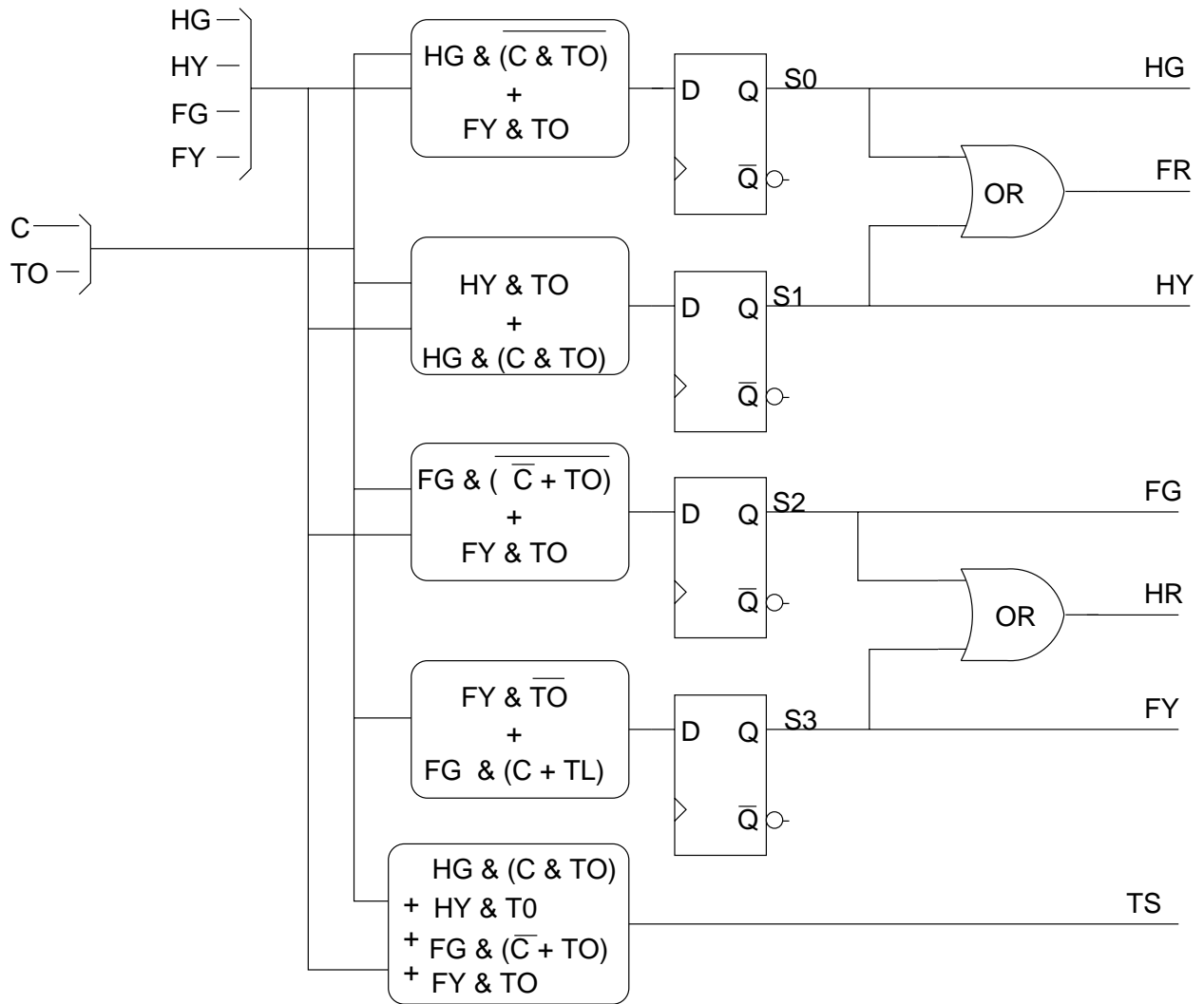
Since we don't need to know the actual duration of the two timers, we will just use one timer process, modeled to time out nondeterministically:

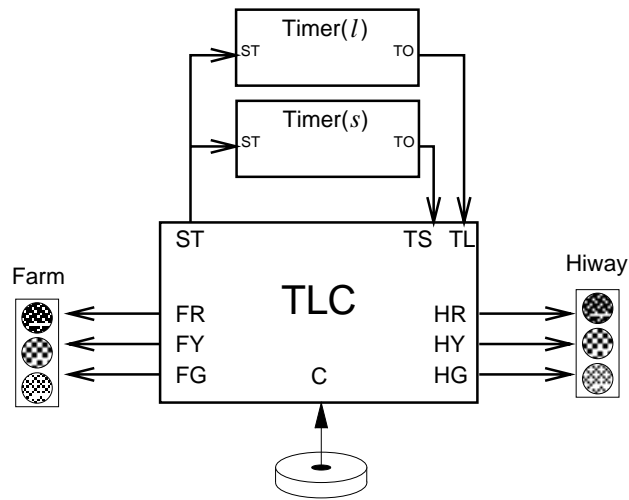


Here is a description of the controller (TLC) given as a finite state machine:



Here is a circuit implementing the controller:





```
MODULE main
```

```
VAR
```

```
  ctl : {HG, HY, FG, FY};
```

```
  sensor : boolean;
```

```
  timer: {OFF, ON};
```

```
DEFINE
```

```
  TO := (timer=ON) & (next(timer)=OFF);
```

```
  ST := case
```

```
    (ctl = HG) & (sensor & TO)   : 1;
```

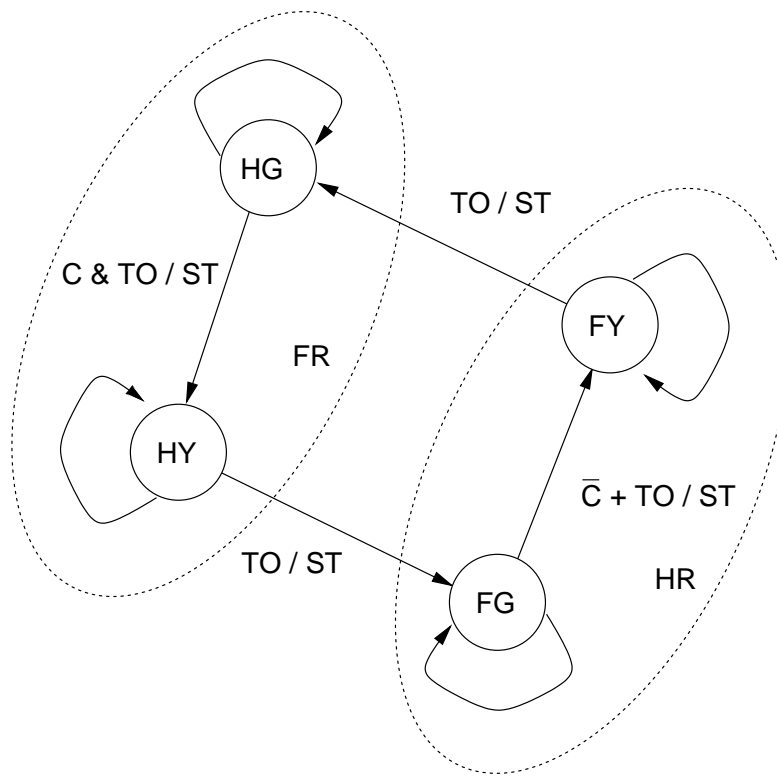
```
    (ctl = HY) & TO              : 1;
```

```
    (ctl = FG) & (!sensor | TO) : 1;
```

```
    (ctl = FY) & TO             : 1;
```

```
    1                           : 0;
```

```
  esac;
```



ASSIGN

```

init(ctl) := HG;
next(ctl) := case
    (ctl = HG) & (sensor & T0) : HY;
    (ctl = HY) & T0             : FG;
    (ctl = FG) & (!sensor | T0) : FY;
    (ctl = FY) & T0             : HG;
    1                           : ctl;
esac;

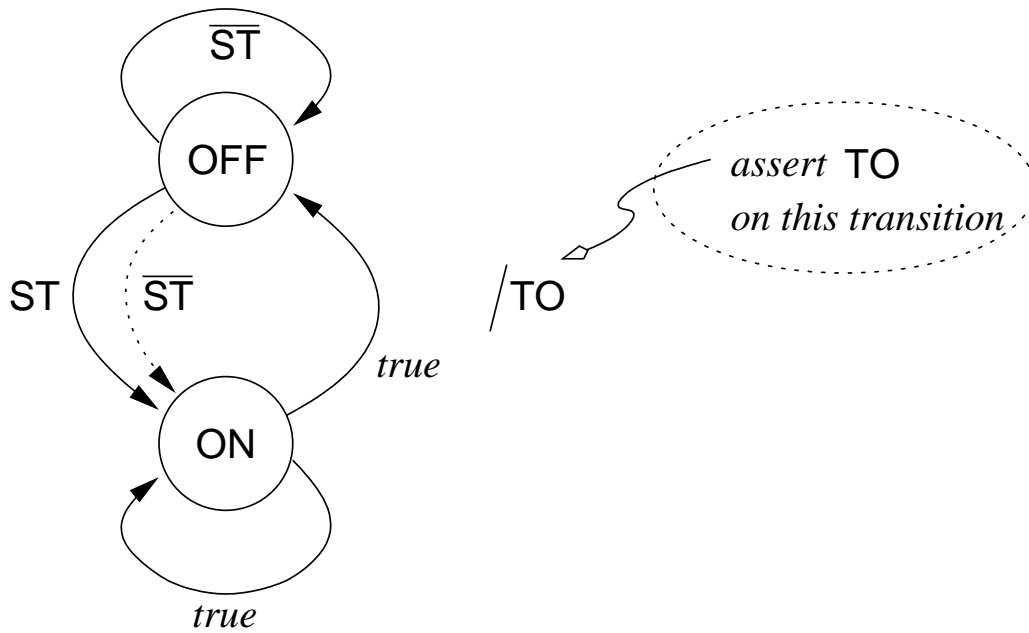
```

```

init(timer) := OFF;
next(timer) := case
    ST : ON;
    1  : {ON, OFF};
esac;

```

The timer should be specified as weakly as possible.



```
init(timer) := OFF;  
next(timer) := case  
    ST : ON;  
    1  : {ON, OFF};  
esac;
```

FAIRNESS timer = OFF -- The timer eventually times out.

DEFINE

```
fg := ctl = FG;           -- farm light is green
fy := ctl = FY;           -- farm light is yellow
fr := (ctl = HY) | (ctl = HG); -- farm light is red
hg := ctl = HG;           -- hiway light is green
hy := ctl = HY;           -- hiway light is yellow
hr := (ctl = FY) | (ctl = FG); -- hiway light is red
```

SPEC -- TLC is safe

```
AG !(fg & hg)
```

SPEC -- TLC is live

```
AG (sensor -> AF (fg))
```

SPEC -- Highway light behaves

```
AG (hg -> A[hg U hy]) -- yellow follows green
```

SPEC

```
AG (hy -> A[hy U hr]) -- red follows yellow
```

SPEC

```
AG (hr -> A[hr U hg]) -- green follows red.
```

SPEC -- Highway light behaves

```
AG (fg -> A[fg U fy]) -- yellow follows green
```

SPEC

```
AG (fy -> A[fy U fr]) -- red follows yellow
```

SPEC

```
AG (fr -> A[fr U fg]) -- green follows red
```

SPEC

AG (fr \rightarrow A[fr U fg]) -- green follows red

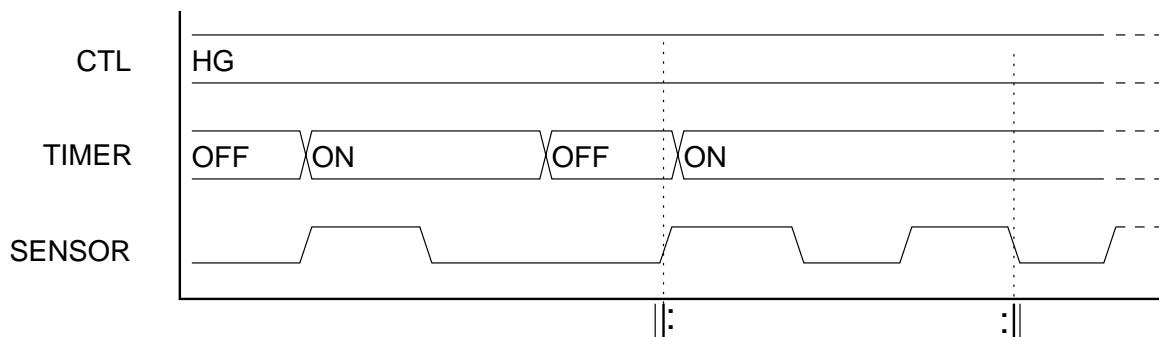
Fails! [TLCa.smv] The *environment* must assure that something trips the sensor.

FAIRNESS sensor

FAIRNESS !sensor

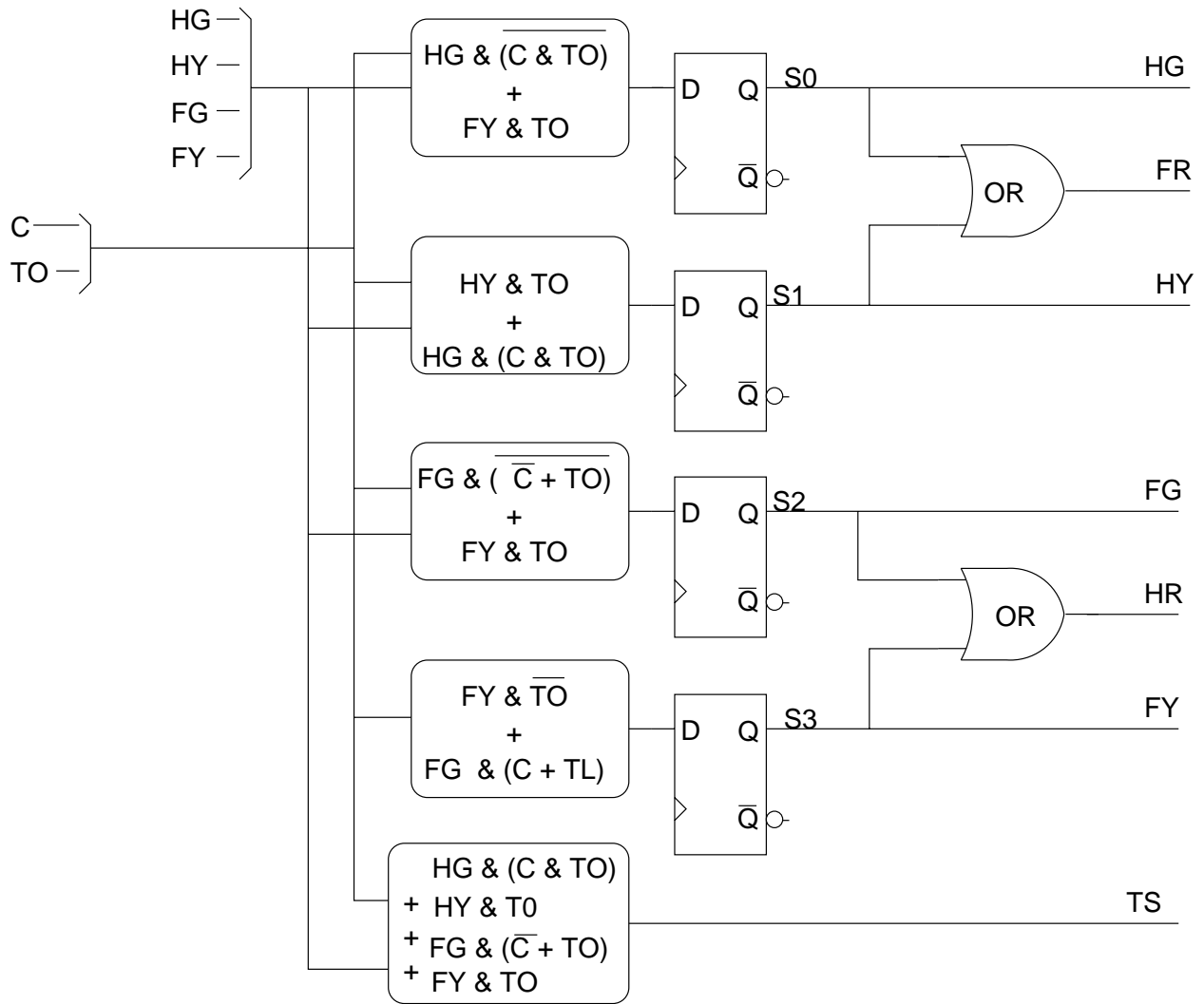
Doesn't work! [TLCd.smv]

TLCd.smv: AG(sensor \rightarrow AF(fg))



The *environment* must constrain cars to behave. [TLCd.smv]

```
next(sensor) :=  
  case  
    sensor & !fg : 1;  
    1           : {0, 1};  
  esac;
```



VAR

```
cs0 : boolean;  
cs1 : boolean;  
cs2 : boolean;  
cs3 : boolean;
```

DEFINE

```
cHG := cs0;  
cHY := cs1;  
cHR := cs2 | cs3;  
cFG := cs2;  
cFY := cs3;  
cFR := cs0 | cs1;  
cST := (cHG & (sensor & T0))  
      | (cHY & T0)  
      | (cFG & (!sensor | T0))  
      | (cFY & T0);
```

ASSIGN

```
init(cs0) := 1;  
init(cs1) := 0;  
init(cs2) := 0;  
init(cs3) := 0;
```

```
next(cs0) := (cHG & !(sensor & T0)) | (cFY & T0);  
next(cs1) := (cHG & (sensor & T0)) | (cHY & T0);  
next(cs2) := (cFG & !(sensor & T0)) | (cFY & T0);  
next(cs3) := (cFG & (sensor | T0)) | (cFY & !T0);
```

SPEC

AG((cHG = hg)
 & (cHY = hy)
 & (cHR = hr)
 & (cFG = fg)
 & (cFY = fy)
 & (cFR = fr))

SPEC

AG(cST = fST)

SPEC

AF(cST)