Lecture 3: Model-checker NuSMV

B. Srivathsan

Chennai Mathematical Institute

NPTEL-course

July - November 2015

Model-checker

- Specify the model of the **system**
- Specify the **requirements**

Model-checker will automatically check if **system satisfies requirements**

Specifying the system

View the computation as a **sequence of states**

Specifying the system

View the computation as a **sequence of states**

A state is a **valuation of the variables**

Specifying the system

View the computation as a **sequence of states**

A state is a **valuation of the variables**

- Declare the **variables**

Specifying the system

View the computation as a **sequence of states**

A state is a **valuation of the variables**

- Declare the **variables**
- Define the **initial values** of the variables

Specifying the system

View the computation as a **sequence of states**

A state is a **valuation of the variables**

- Declare the **variables**
- Define the **initial values** of the variables
- Define the **next-state relation**

Specifying the system

View the computation as a **sequence of states**

A state is a **valuation of the variables**

- Declare the **variables**
- Define the **initial values** of the variables
- Define the **next-state relation**

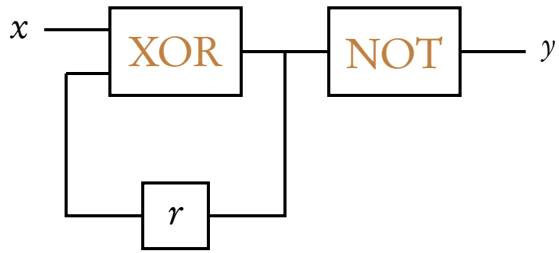
In this course: model-checker **NuSMV**

NuSMV

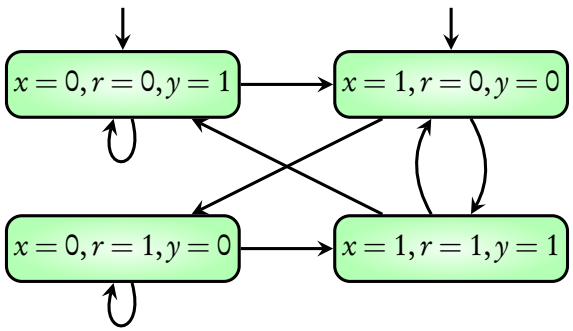
New Symbolic Model Verifier

`http://nusmv.fbk.eu/`

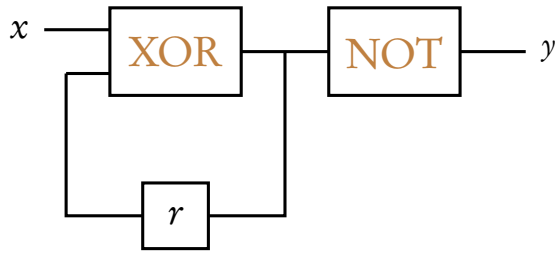
$$y = \text{NOT}\ (\ \text{XOR}\ (x, r)\ )$$
$$r_{next} = \text{XOR}\ (x, r)$$



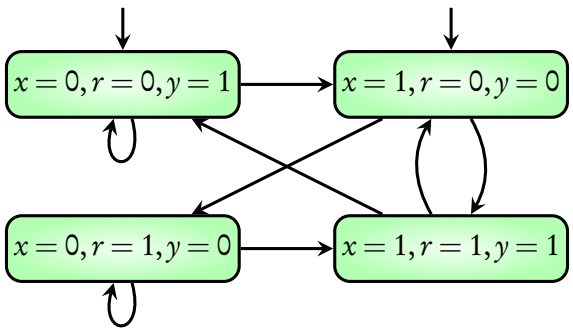
register



$$y = \text{NOT}\ (\ \text{XOR}\ (x, r)\ )$$
$$r_{next} = \text{XOR}\ (x, r)$$

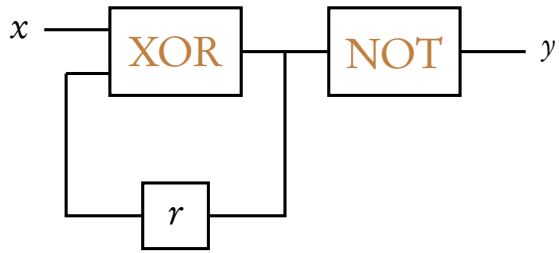


register

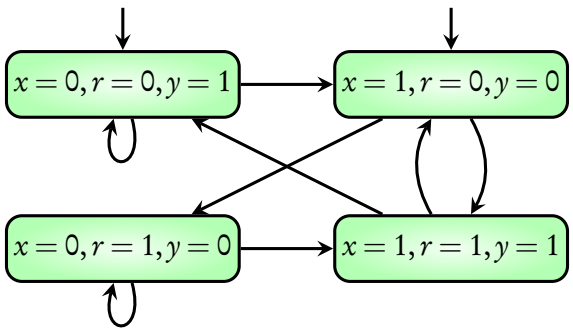



```
MODULE main
```

$$y = \text{NOT}\ (\ \text{XOR}\ (x, r)\ )$$
$$r_{next} = \text{XOR}\ (x, r)$$

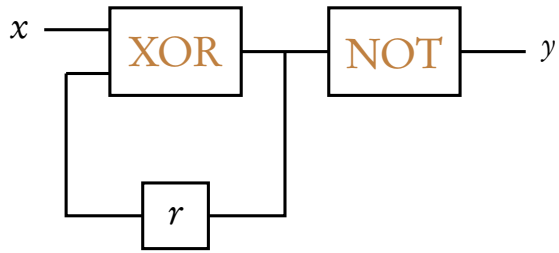


register

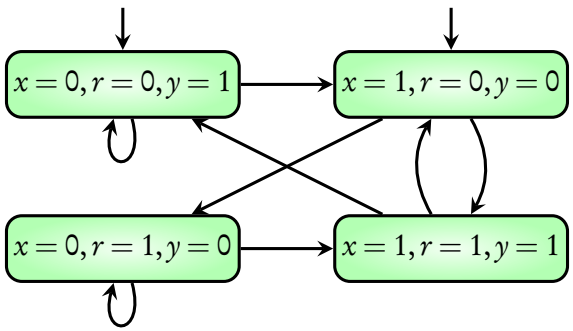



```
MODULE main
VAR
   x:  boolean;
   r:  boolean;
```

$$y = \text{NOT}\ (\ \text{XOR}\ (x, r)\ )$$
$$r_{next} = \text{XOR}\ (x, r)$$

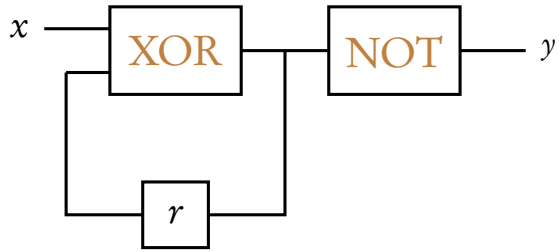


register

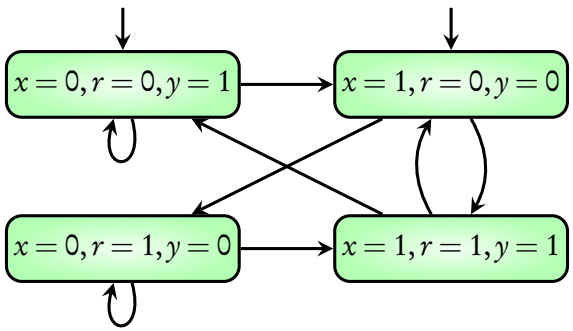



```
MODULE main
VAR
   x:  boolean;
   r:  boolean;

ASSIGN
   init(r) := FALSE;
```

$$y = \text{NOT}\ (\ \text{XOR}\ (x, r)\ )$$
$$r_{next} = \text{XOR}\ (x, r)$$

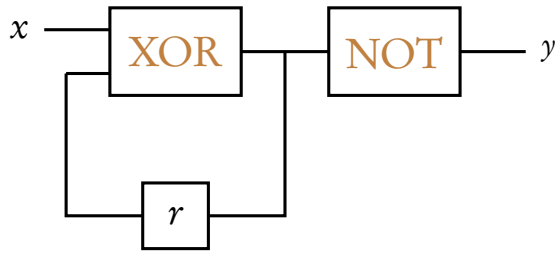


register

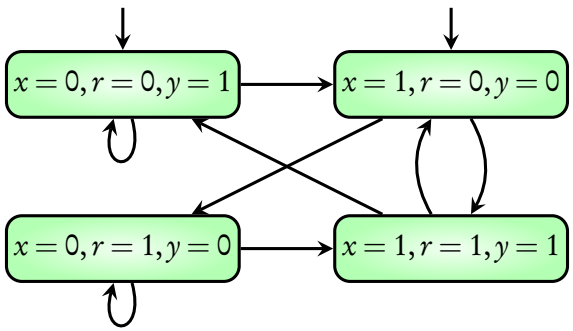



```
MODULE main
VAR
   x:  boolean;
   r:  boolean;

ASSIGN
   init(r) := FALSE;
   next(r) := x xor r;
```

$$y = \text{NOT}\ (\ \text{XOR}\ (x, r)\ )$$
$$r_{next} = \text{XOR}\ (x, r)$$

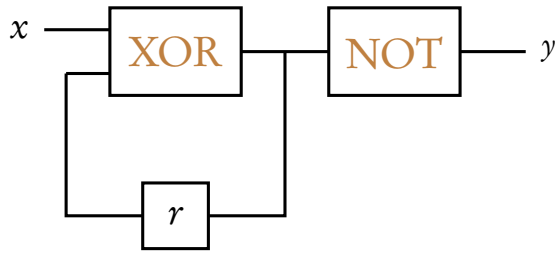


register

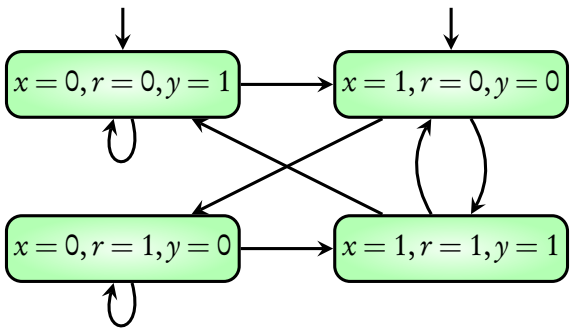



```
MODULE main
VAR
   x:  boolean;
   r:  boolean;
DEFINE
   y := !(x xor r);
ASSIGN
   init(r) := FALSE;
   next(r) := x xor r;
```

$$y = \text{NOT}\ (\ \text{XOR}\ (x, r)\ )$$
$$r_{next} = \text{XOR}\ (x, r)$$

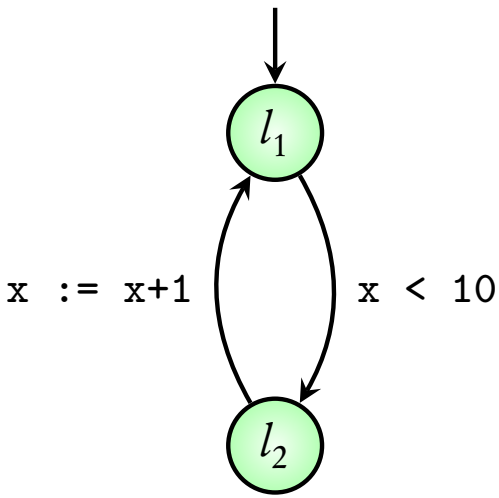


register

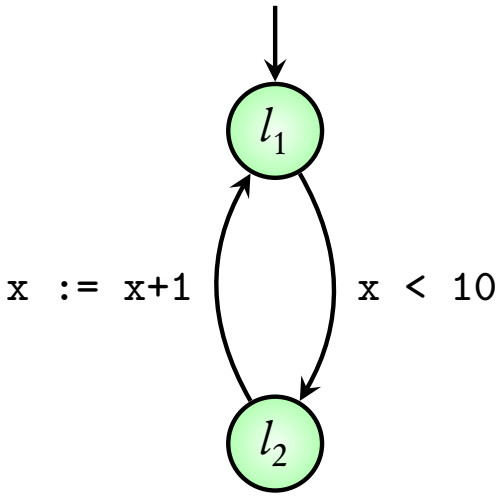



```
MODULE main
VAR
   x:  boolean;
   r:  boolean;
DEFINE
   y := !(x xor r);
ASSIGN
   init(r) := FALSE;
   next(r) := x xor r;
```

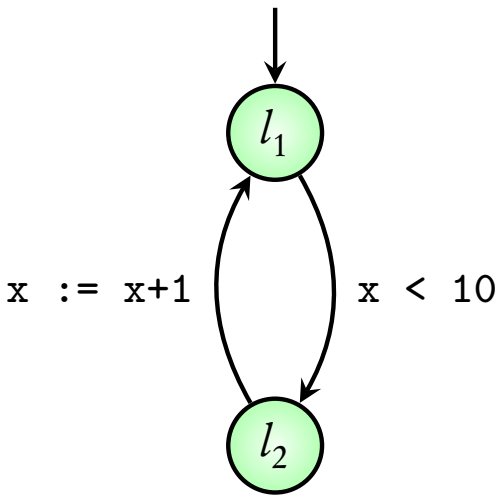
NuSMV demo: `circuit-demo1.smv`

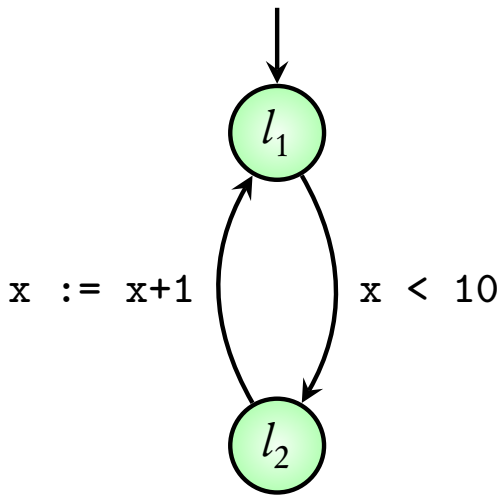

l_1
x := x+1
x < 10
l_2

```
MODULE main

VAR



ASSIGN
```

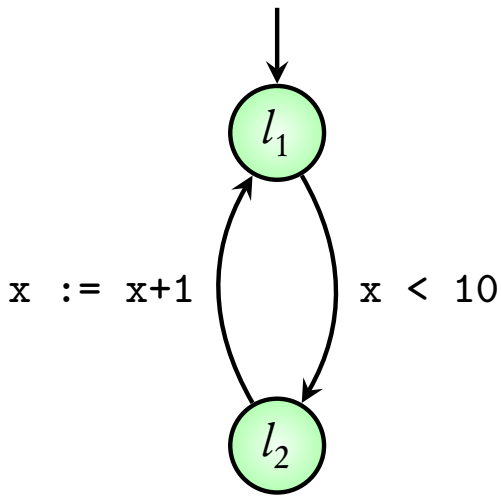


```
MODULE main
VAR
    location:  {l1,l2};
    x:  0 ..  100;
ASSIGN
```

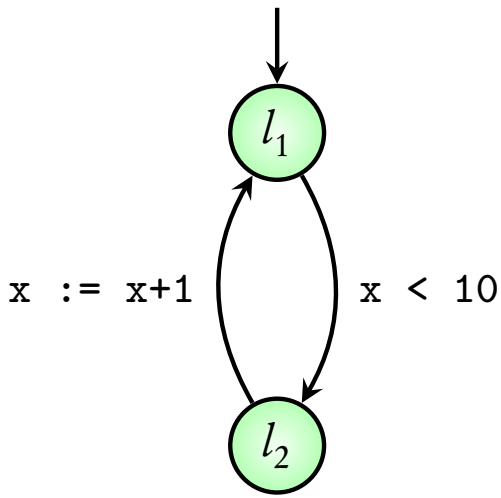




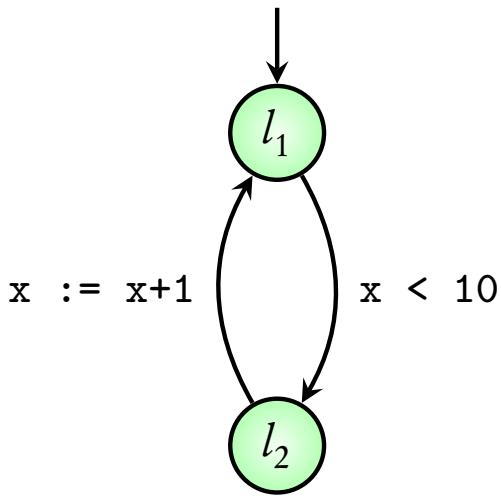

```
MODULE main
VAR
    location:  {l1,l2};
    x:  0 ..  100;
ASSIGN
    init(location) := l1;
    init(x) := 0;
```



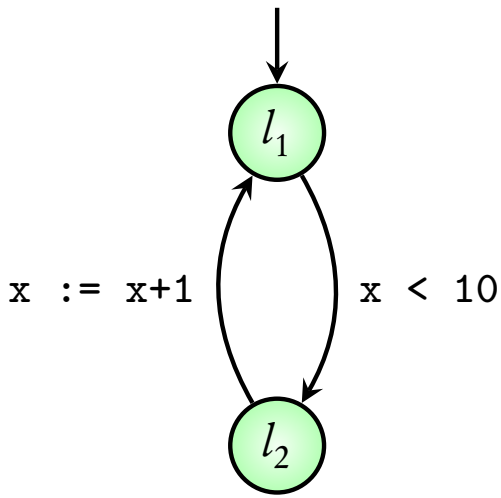

```
MODULE main
VAR
    location:  {l1,l2};
    x:  0 ..  100;
ASSIGN
    init(location) := l1;
    init(x) := 0;
    next(location) := case


                              esac;
```



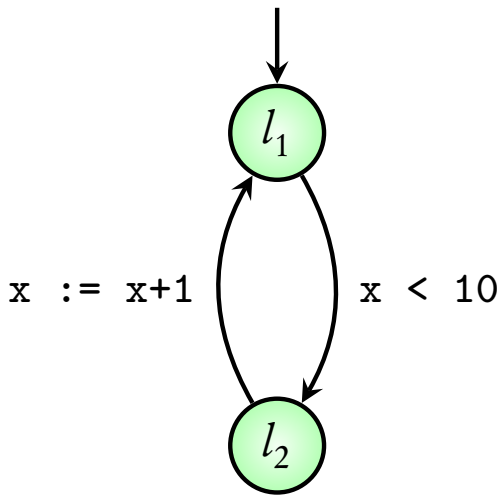

```
MODULE main
VAR
    location:  {l1,l2};
    x:  0 ..  100;
ASSIGN
    init(location) := l1;
    init(x) := 0;
    next(location) := case
                  (location = l1) & (x<10):  l2;


                                  esac;
```



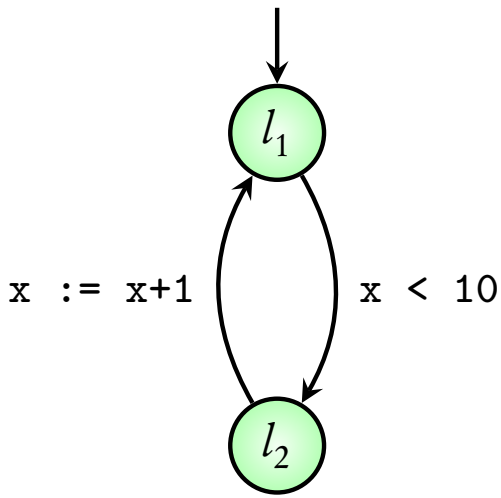

```
MODULE main
VAR
    location:  {l1,l2};
    x:  0 ..  100;
ASSIGN
    init(location) := l1;
    init(x) := 0;
    next(location) := case
                  (location = l1) & (x<10):  l2;
                  (location = l2) :  l1;

                       esac;
```



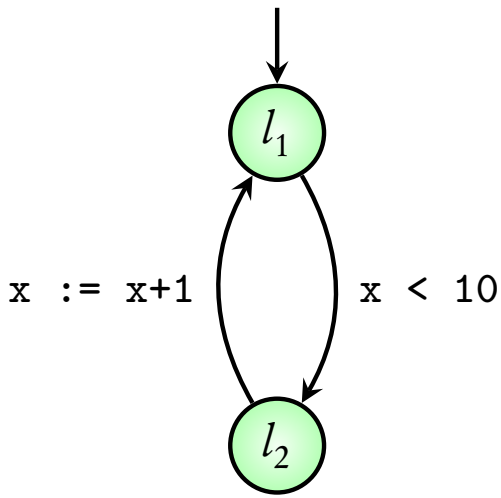

```
MODULE main
VAR
    location:  {l1,l2};
    x:  0 ..  100;
ASSIGN
    init(location) := l1;
    init(x) := 0;
    next(location) := case
                  (location = l1) & (x<10):  l2;
                  (location = l2) :  l1;
                  TRUE: location;
                       esac;
```



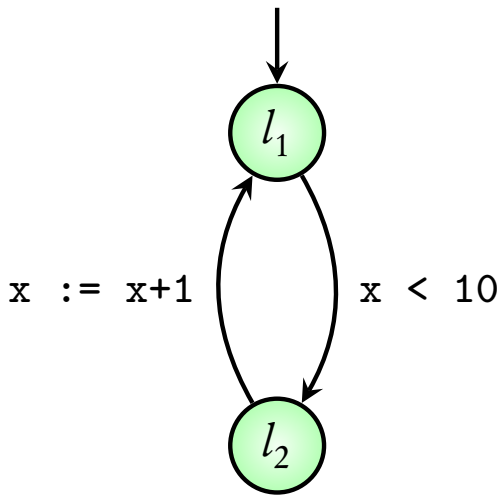

```
MODULE main
VAR
    location:  {l1,l2};
    x:  0 ..  100;
ASSIGN
    init(location) := l1;
    init(x) := 0;
    next(location) := case
                  (location = l1) & (x<10):  l2;
                  (location = l2) :  l1;
                  TRUE: location;
                       esac;
    next(x) := case


              esac;
```




```
MODULE main
VAR
    location:  {l1,l2};
    x:  0 ..  100;
ASSIGN
    init(location) := l1;
    init(x) := 0;
    next(location) := case
                  (location = l1) & (x<10):  l2;
                  (location = l2) :  l1;
                  TRUE: location;
                       esac;
    next(x) := case
                  (location = l2) & x < 100:  x+1;

                esac;
```



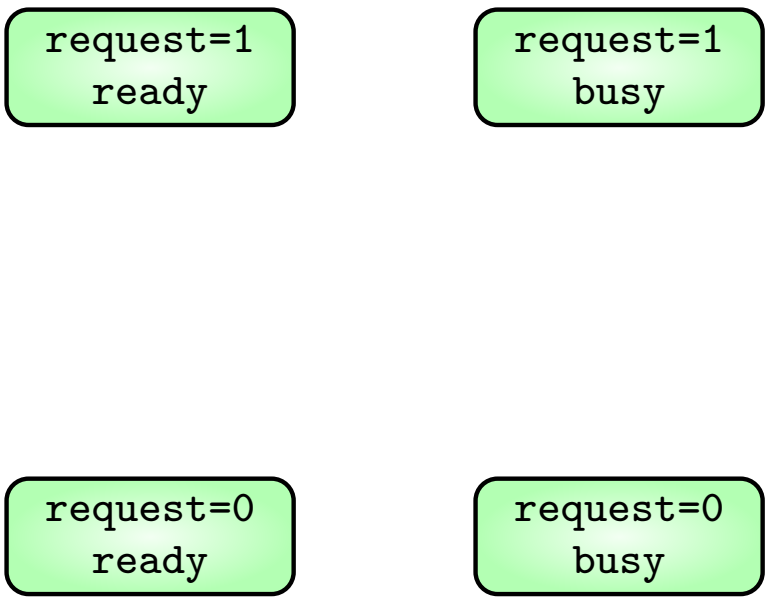

```
MODULE main
VAR
    location:  {l1,l2};
    x:  0 ..  100;
ASSIGN
    init(location) := l1;
    init(x) := 0;
    next(location) := case
                  (location = l1) & (x<10):  l2;
                  (location = l2) :  l1;
                  TRUE: location;
                       esac;
    next(x) := case
                  (location = l2) & x < 100:  x+1;
                  TRUE: x;
                esac;
```



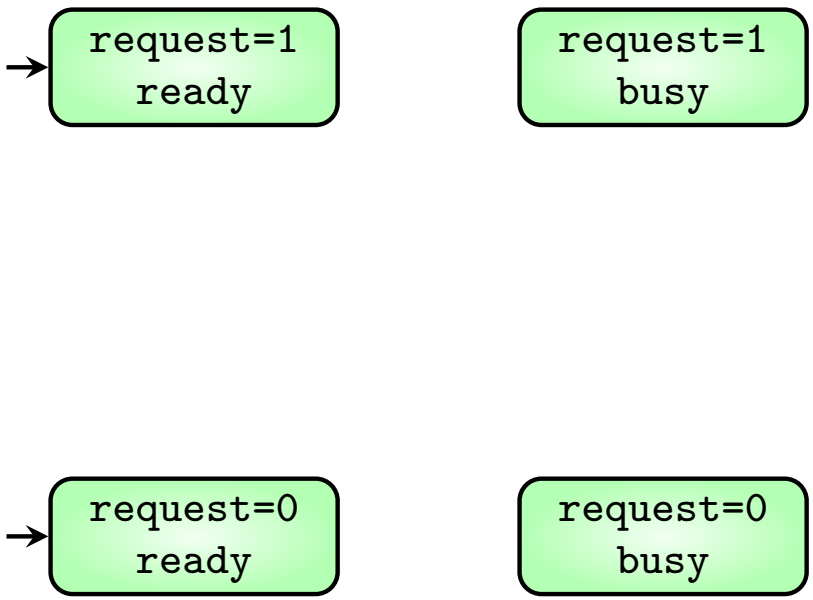

```
MODULE main
VAR
    location:  {l1,l2};
    x:  0 ..  100;
ASSIGN
    init(location) := l1;
    init(x) := 0;
    next(location) := case
                  (location = l1) & (x<10):  l2;
                  (location = l2) :  l1;
                  TRUE: location;
                       esac;
    next(x) := case
                  (location = l2) & x < 100:  x+1;
                  TRUE: x;
                esac;
```

NuSMV file: `pg-demo.smv`

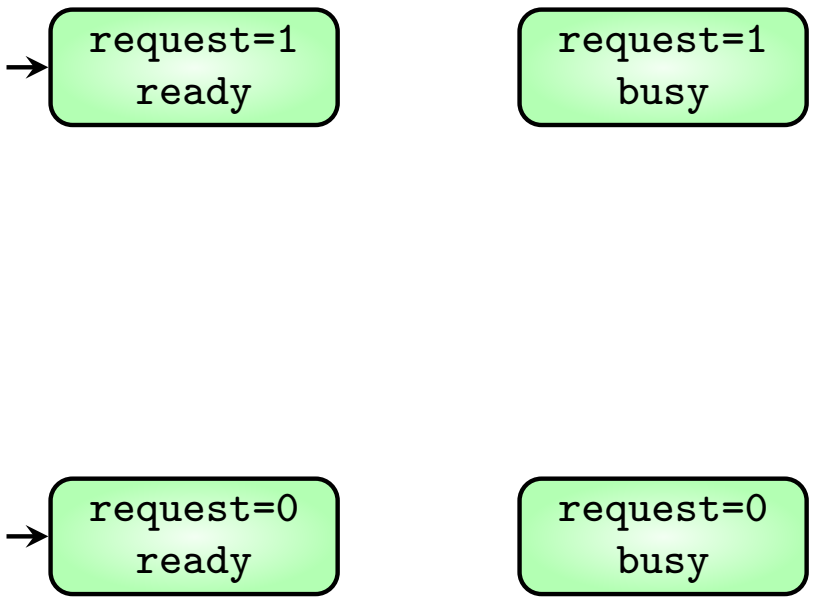
```
MODULE main
VAR
    request:  boolean;
    status:  {ready, busy}
```



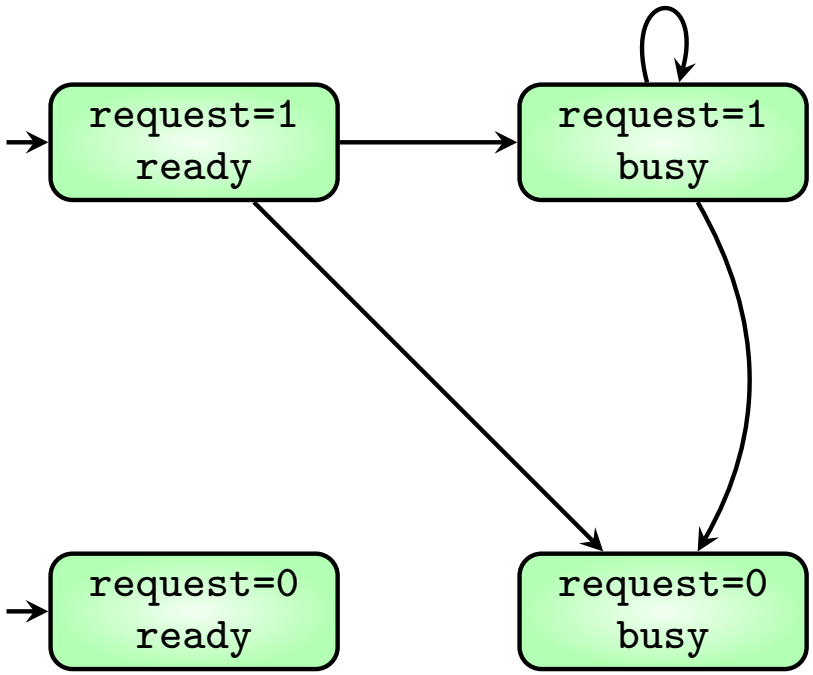

```
MODULE main
VAR
    request:  boolean;
    status:  {ready, busy}
```



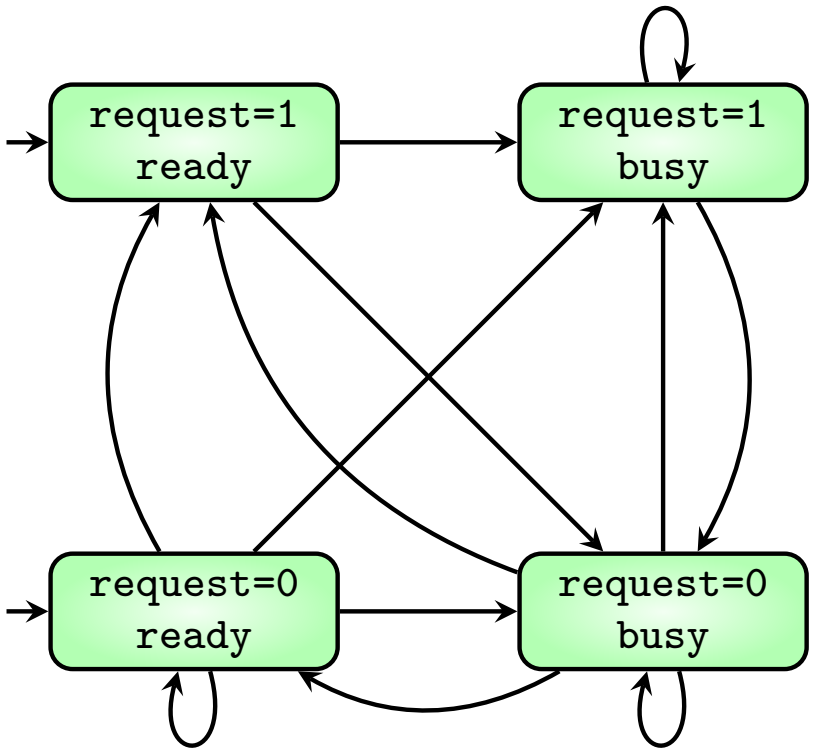

```
MODULE main
VAR
    request:  boolean;
    status:  {ready, busy}
ASSIGN
    init(status) := ready;
```




```
MODULE main
VAR
    request:  boolean;
    status:  {ready, busy}
ASSIGN
    init(status) := ready;
    next(status) := case
                    request :  busy;
                    TRUE : {ready,busy};
                          esac;
```



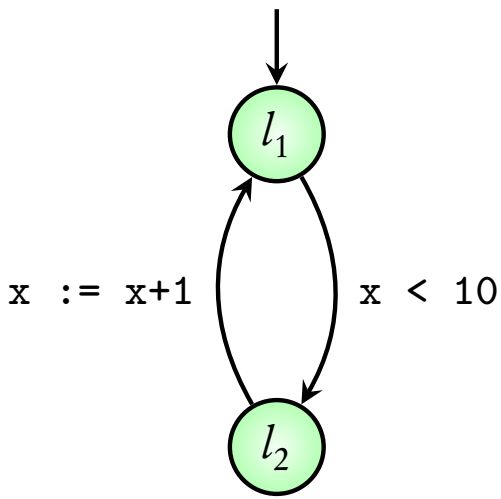

```
MODULE main
VAR
    request:  boolean;
    status:  {ready, busy}
ASSIGN
    init(status) := ready;
    next(status) := case
                    request :  busy;
                    TRUE : {ready,busy};
                          esac;
```

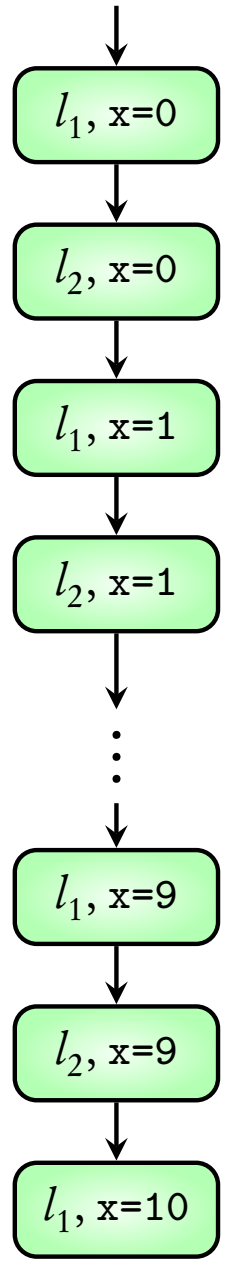



```
MODULE main
VAR
    request:  boolean;
    status:  {ready, busy}
ASSIGN
    init(status) := ready;
    next(status) := case
                    request :  busy;
                    TRUE : {ready,busy};
                          esac;
```

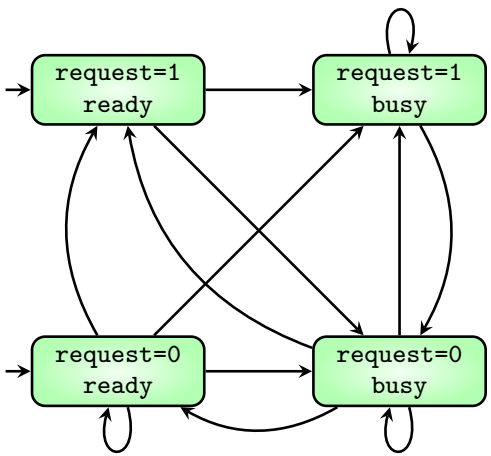
Coming next: checking **requirements** in NuSMV

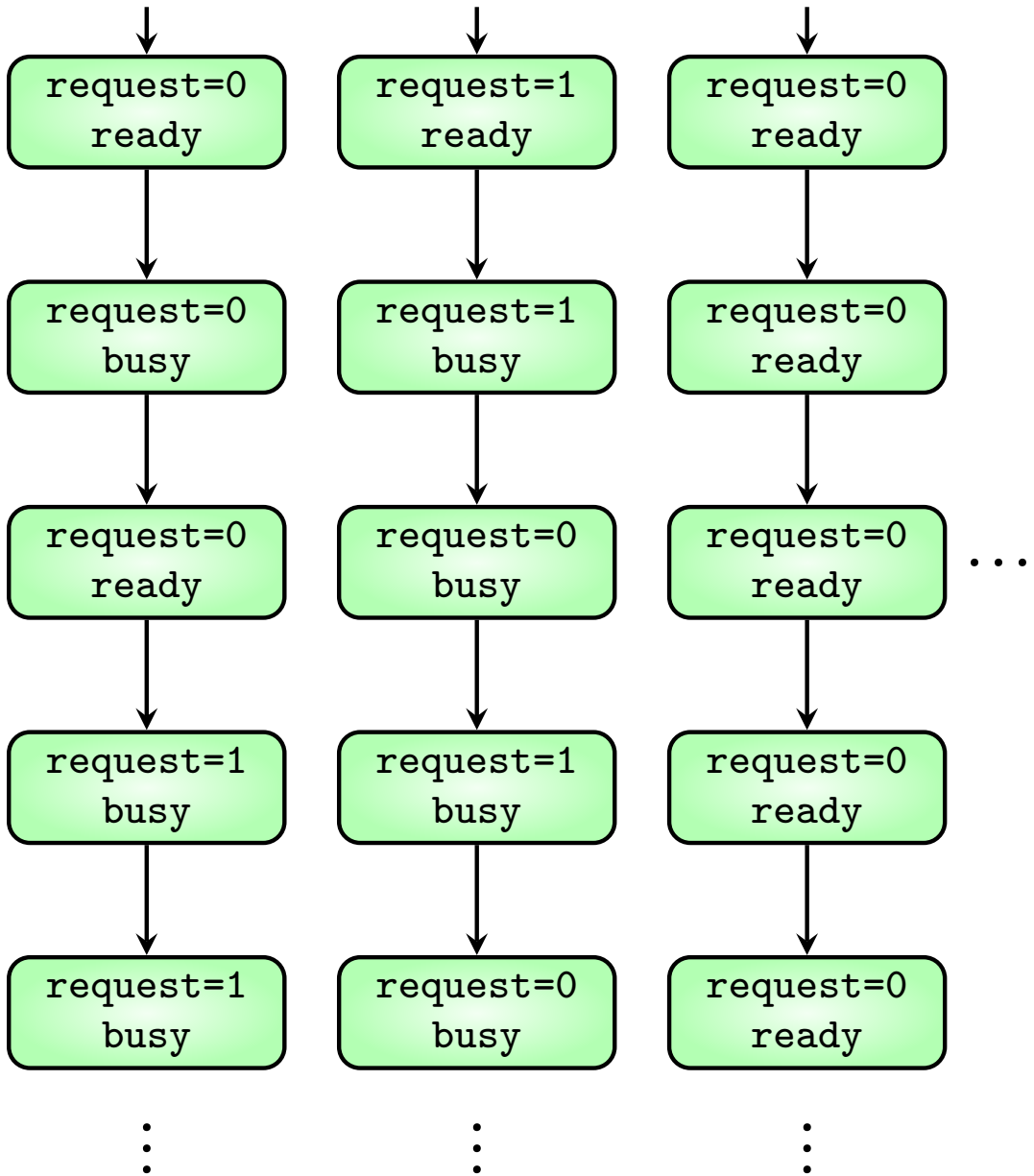
Executions





Executions


request=1
ready
request=1
busy
request=0
ready
request=0
busy


request=0
ready
request=0
busy
request=0
ready
request=1
busy
request=1
busy
request=0
busy
request=0
ready
request=1
busy
request=0
busy
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready

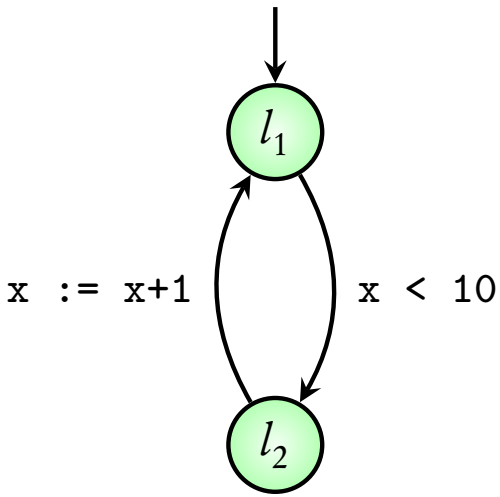
Transition system **satisfies a requirement**

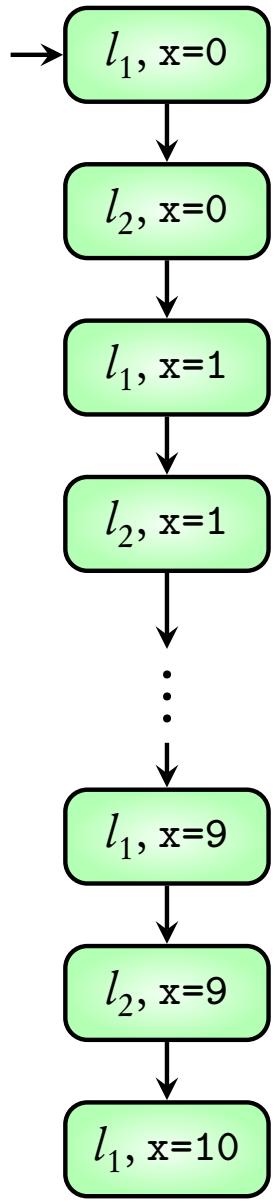
means

all its executions satisfy the requirement

Requirement type 1: G

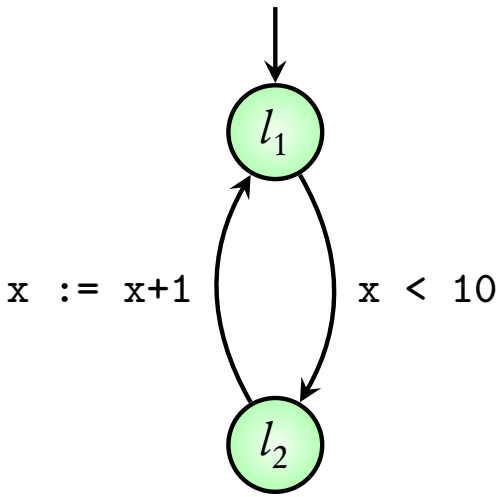
Requirement type 1: G





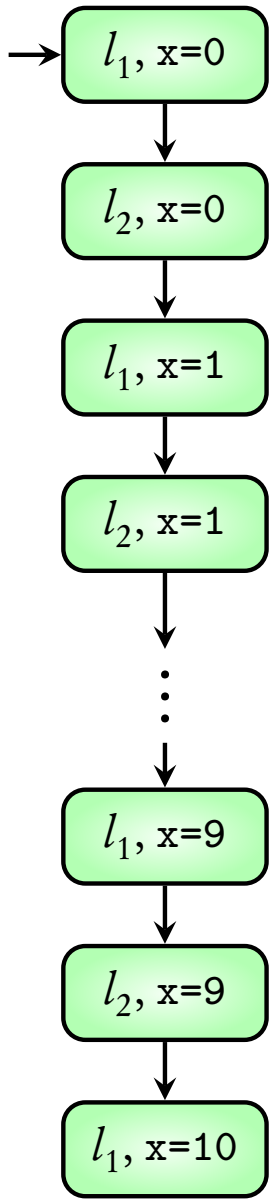
G (x >= 0)

Requirement type 1: G



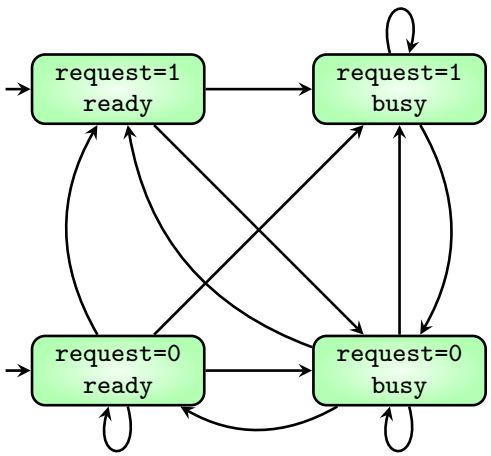
TS of above PG with initial value x=0

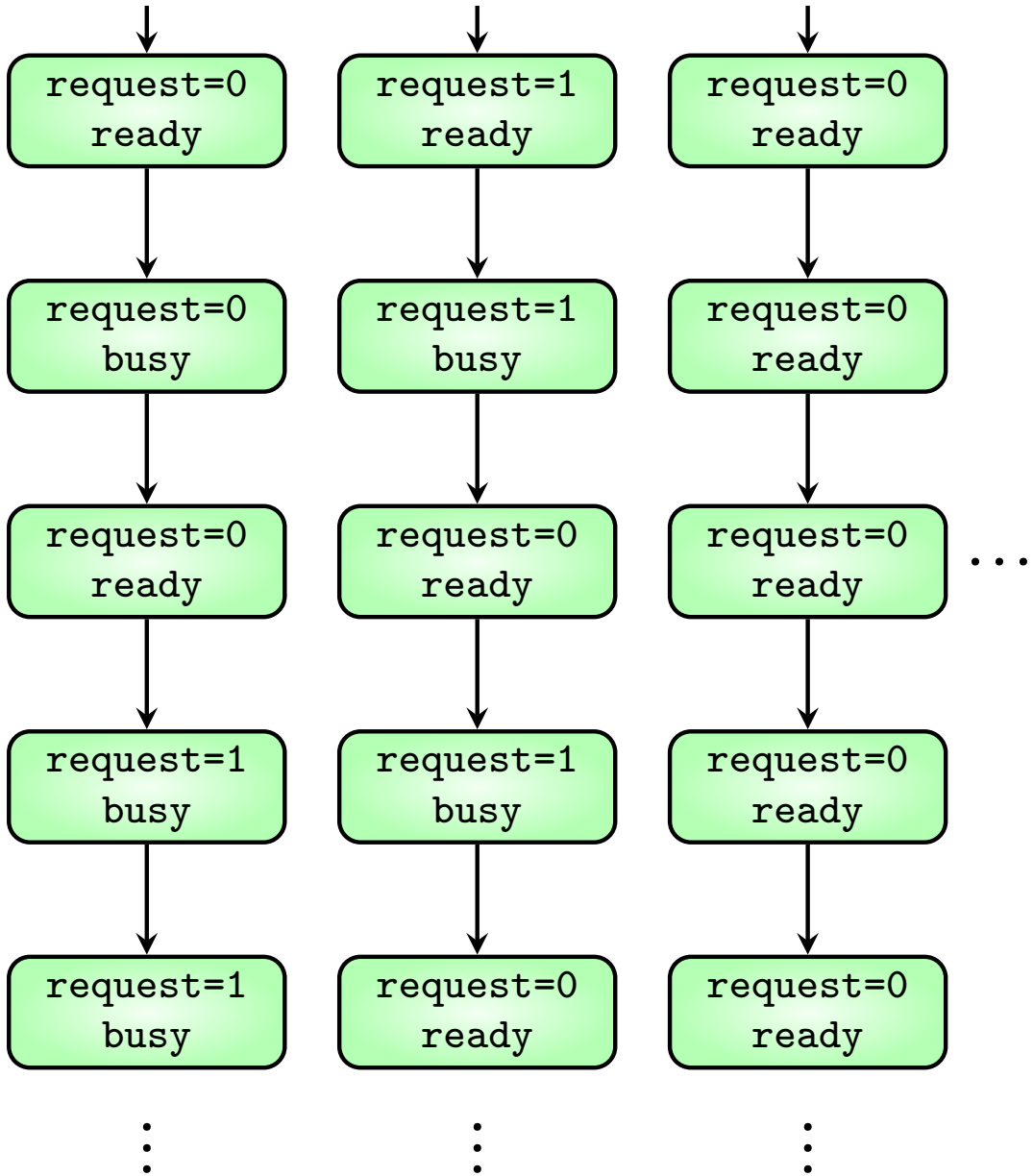
satisfies G (x >= 0)



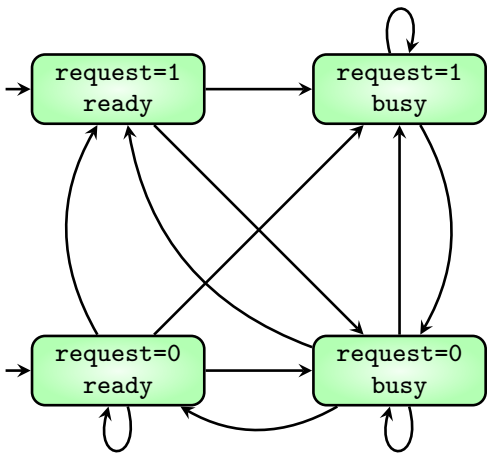
G (x >= 0)

G (request=0)

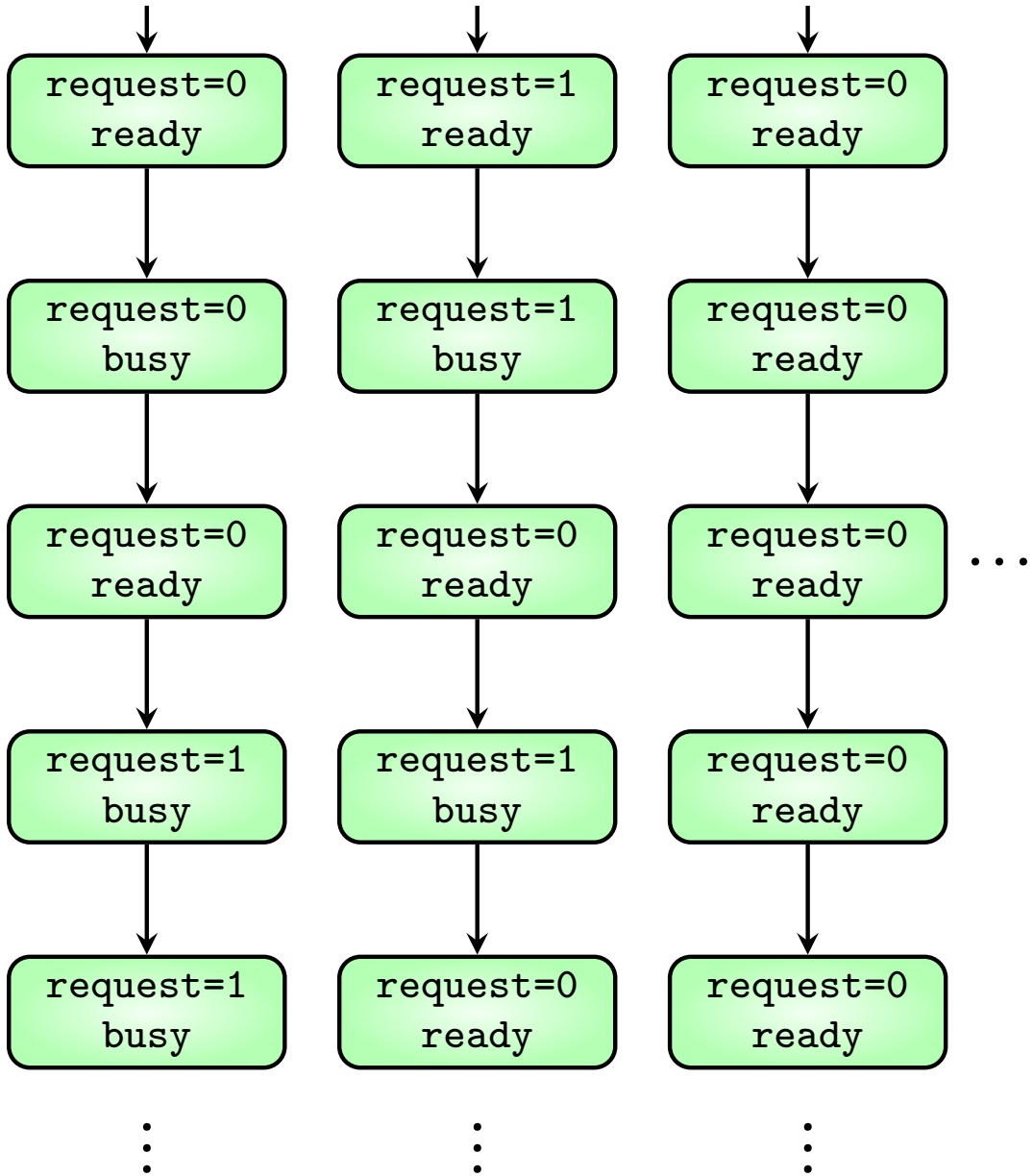

request=1
ready
request=1
busy
request=0
ready
request=0
busy


request=0
ready
request=0
busy
request=0
ready
request=1
busy
request=1
busy
request=1
busy
request=1
busy
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready
...

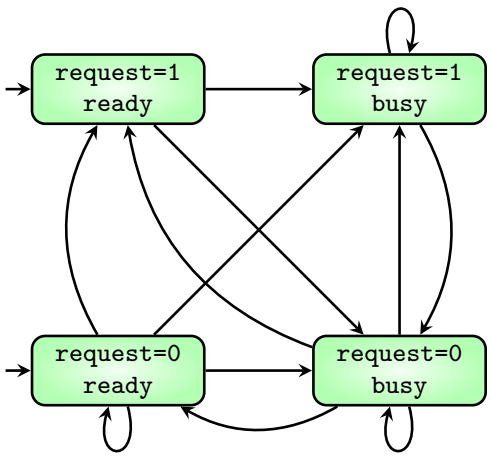
G (request=0)

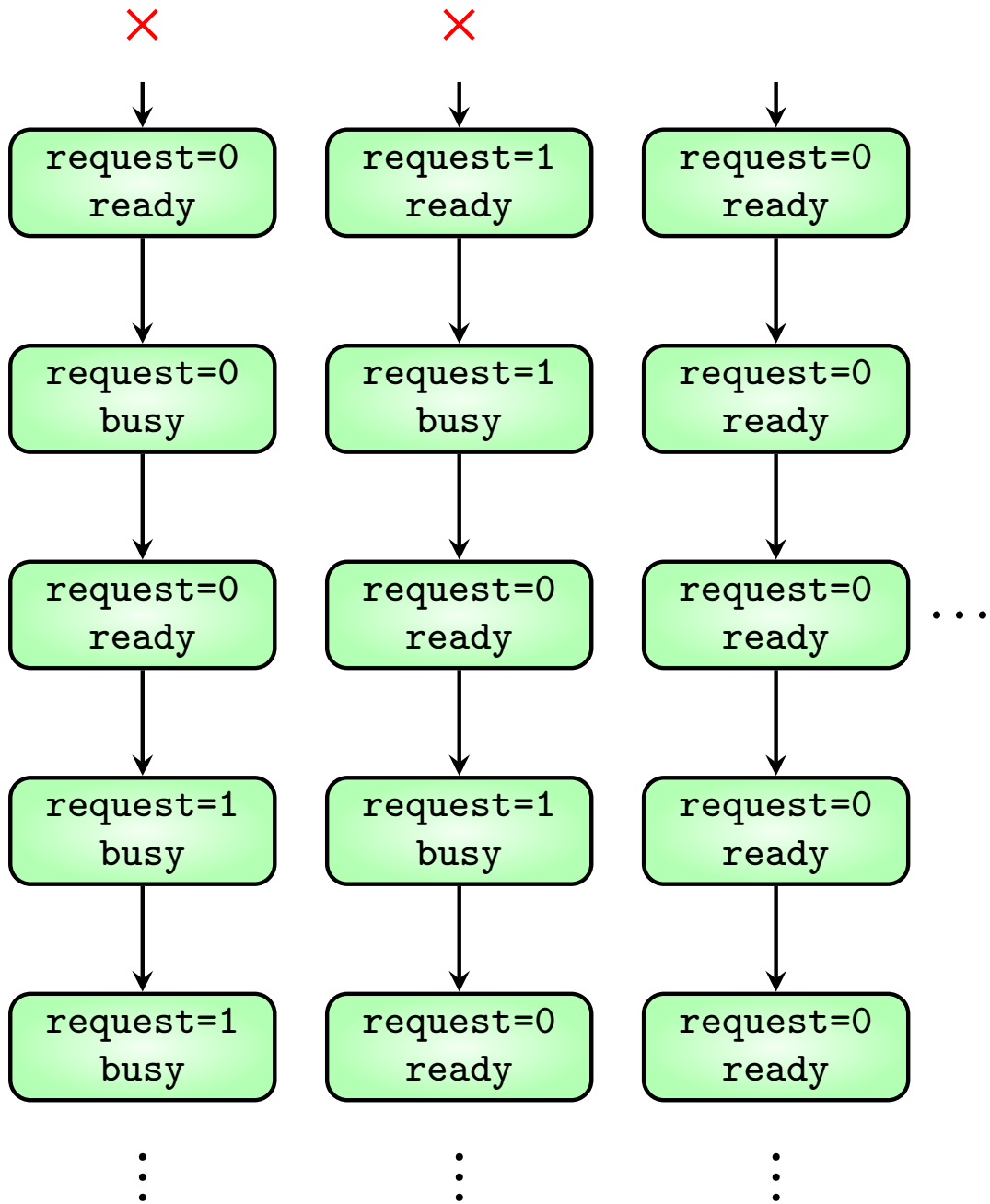

request=1
ready
request=1
busy
request=0
ready
request=0
busy

×

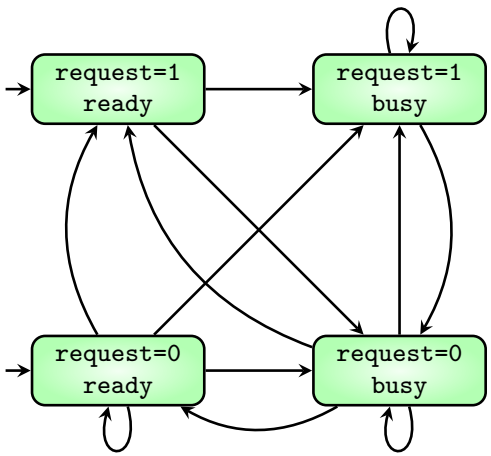

request=0
ready
request=0
busy
request=0
ready
request=1
busy
request=1
busy
request=1
ready
request=1
busy
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready

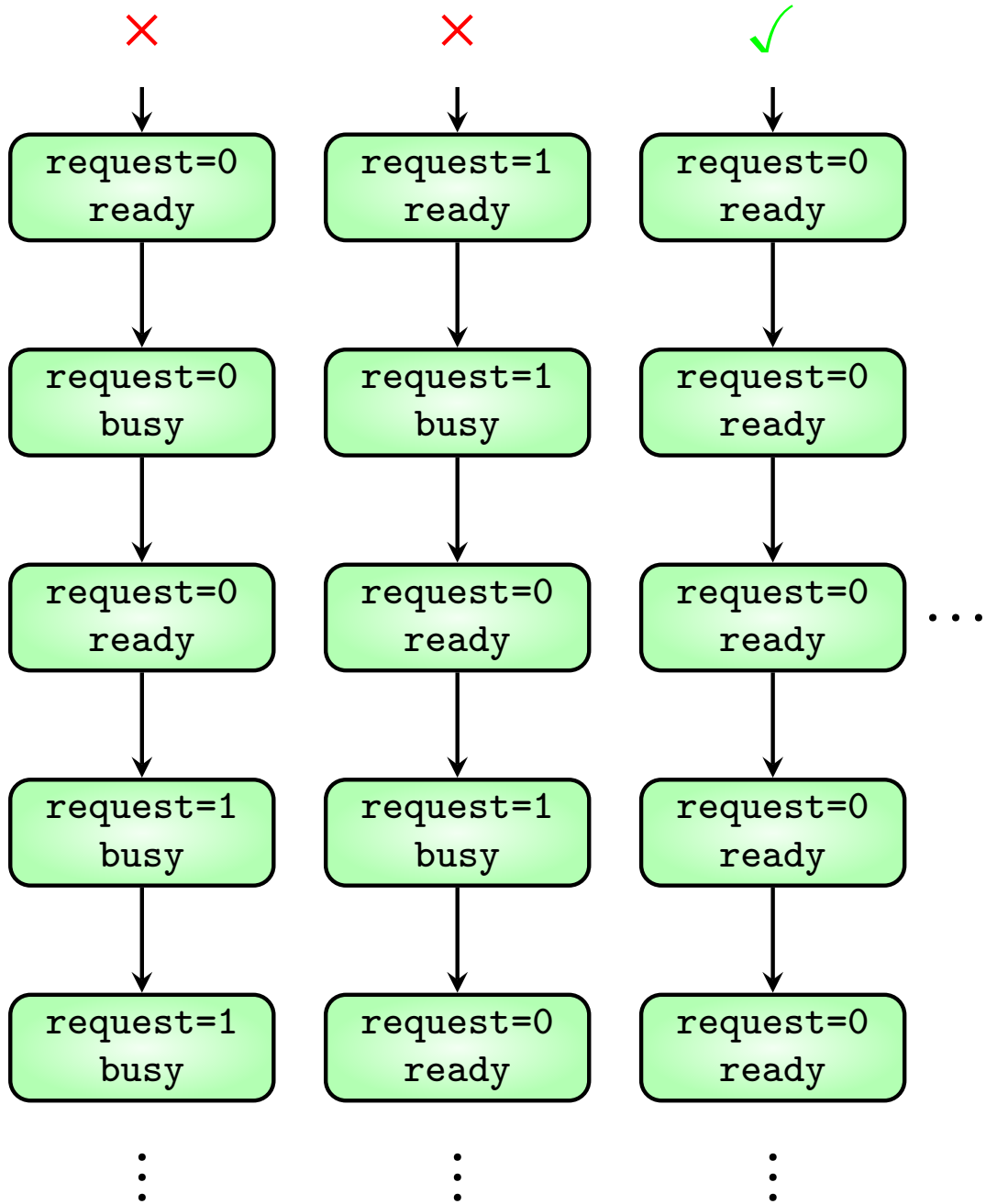
G (request=0)


request=1
ready
request=1
busy
request=0
ready
request=0
busy

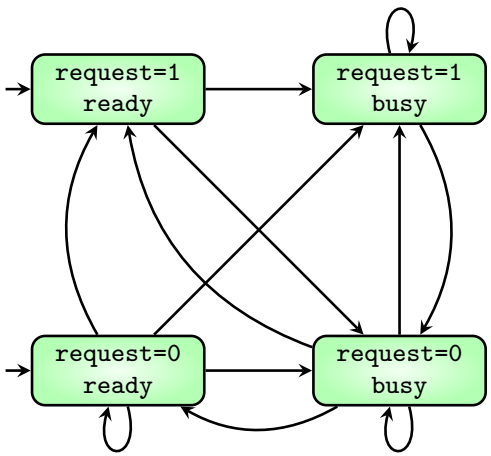

request=0
ready
request=0
busy
request=0
ready
request=1
busy
request=1
busy
request=1
ready
request=1
busy
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready

G (request=0)


request=1
ready
request=1
busy
request=0
ready
request=0
busy

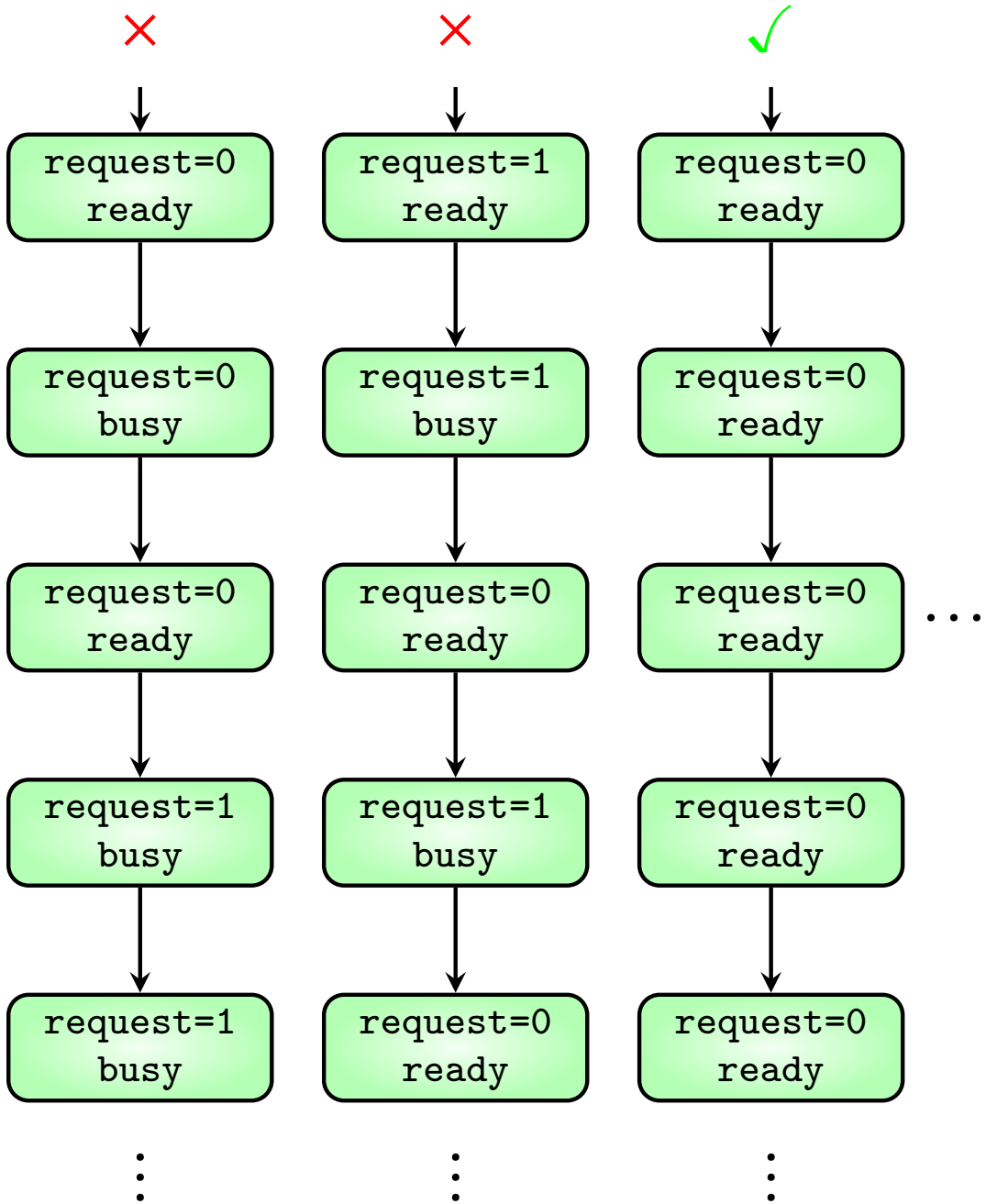

request=0
ready
request=0
busy
request=0
ready
request=1
busy
request=1
busy
request=1
ready
request=1
busy
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready

G (request=0)



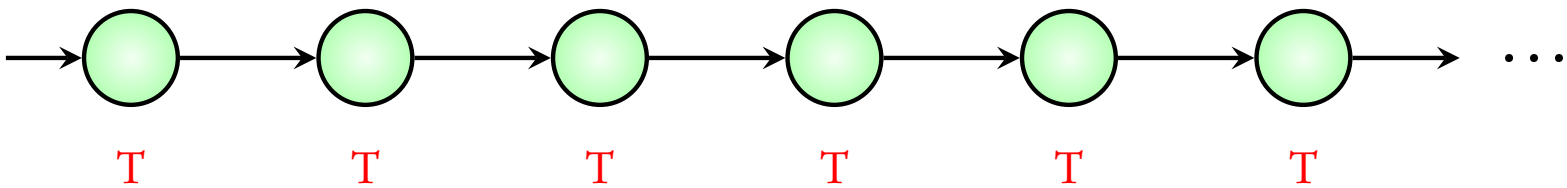
TS does not satisfy

G (request=0)



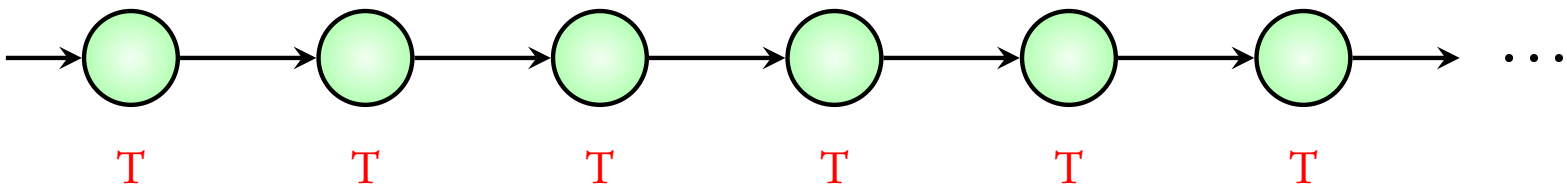
Execution **satisfies** G `(expr)` if

`expr` evaluates to T in **all its states**



Execution **satisfies** G (expr) if

expr evaluates to T in **all its states**



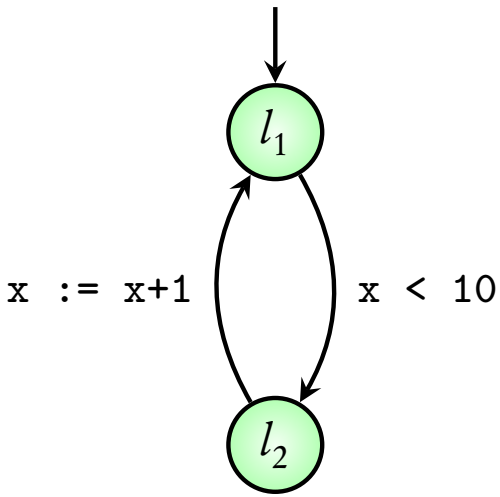
Transition system **satisfies** G (expr) if

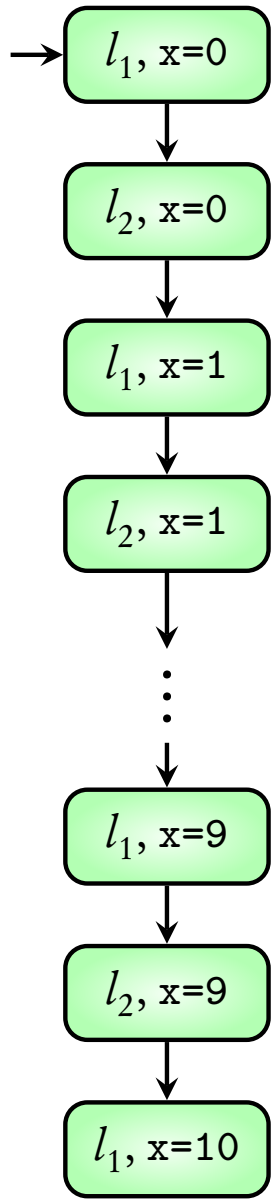
all its executions satisfy G (expr)

Checking the G requirement: NuSMV demo

Requirement type 2: F

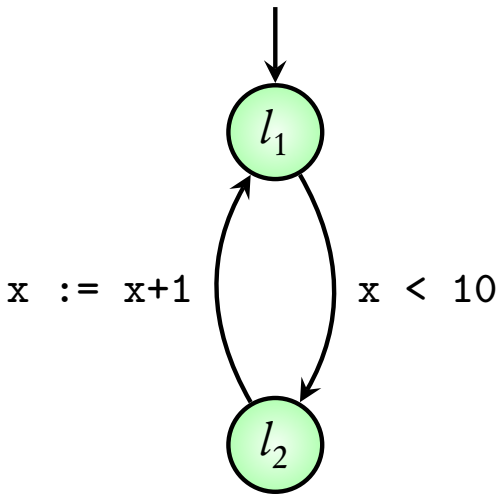
Requirement type 2: F





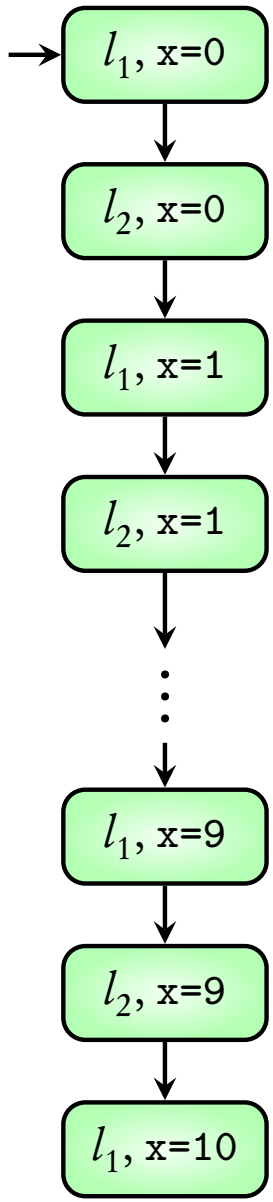
F (x >= 5)

Requirement type 2: F



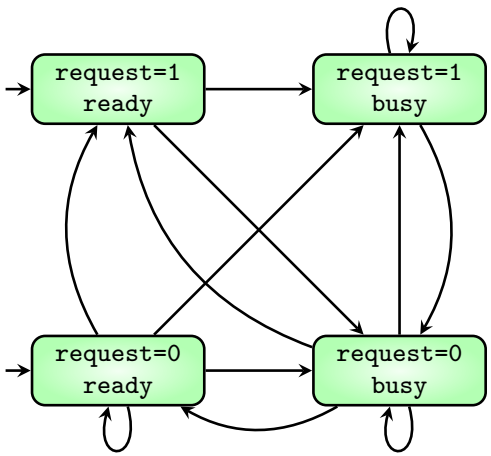
TS of above PG with initial value x=0

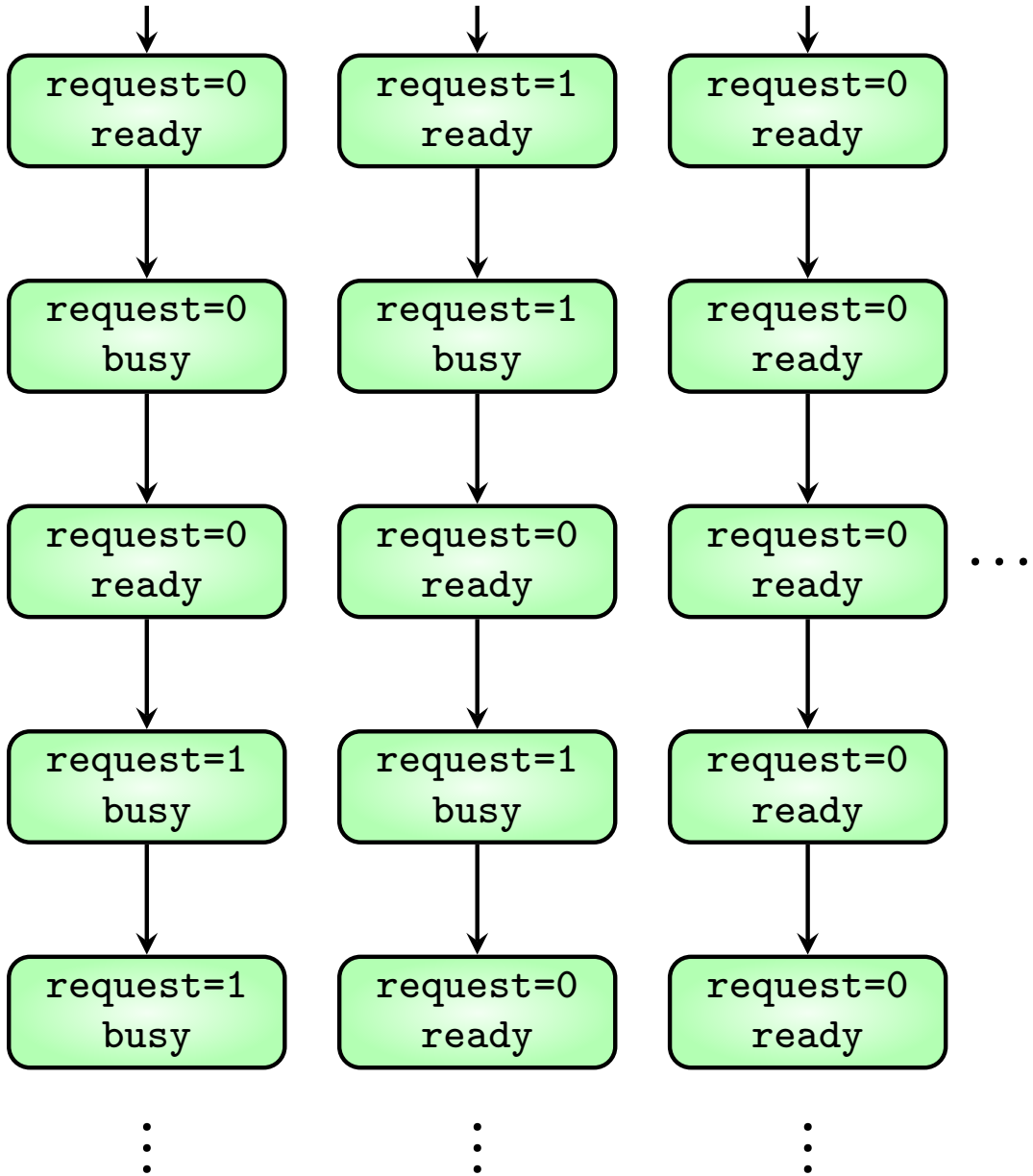
satisfies F (x >= 5)



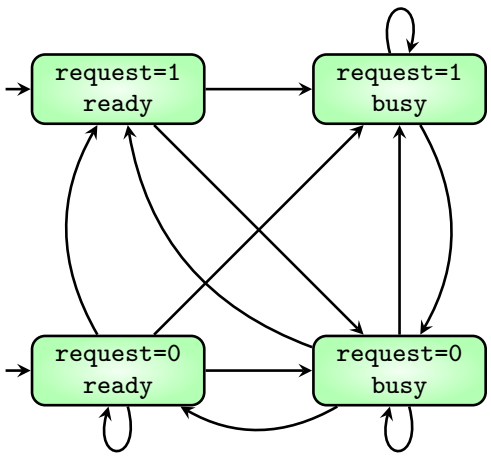
F (x >= 5)

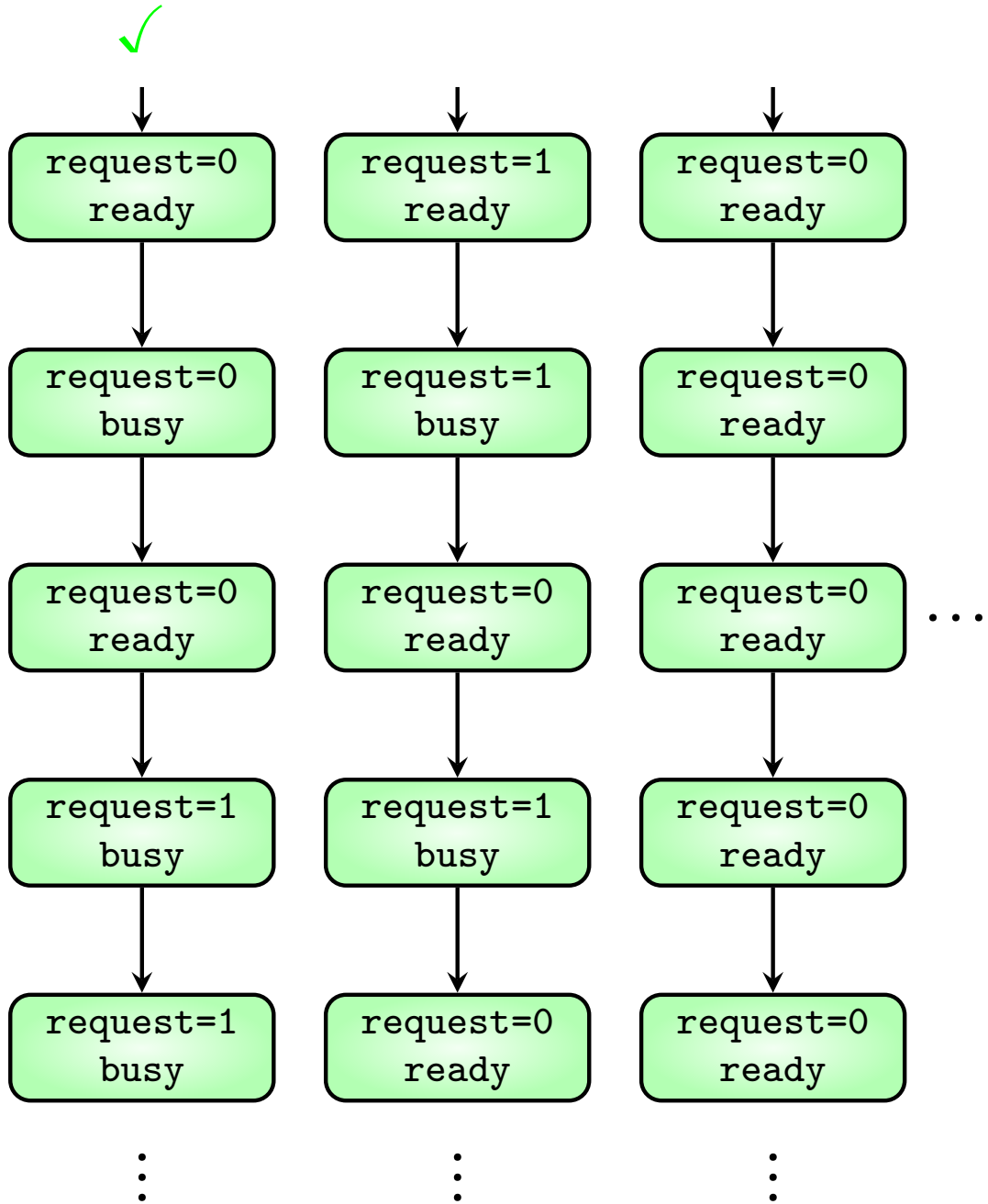
F (request=1)


request=1
ready
request=1
busy
request=0
ready
request=0
busy

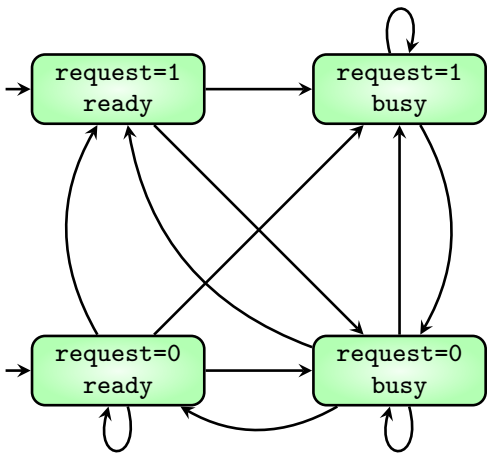

request=0
ready
request=0
busy
request=0
ready
request=1
busy
request=1
busy
request=1
busy
request=1
busy
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready
...

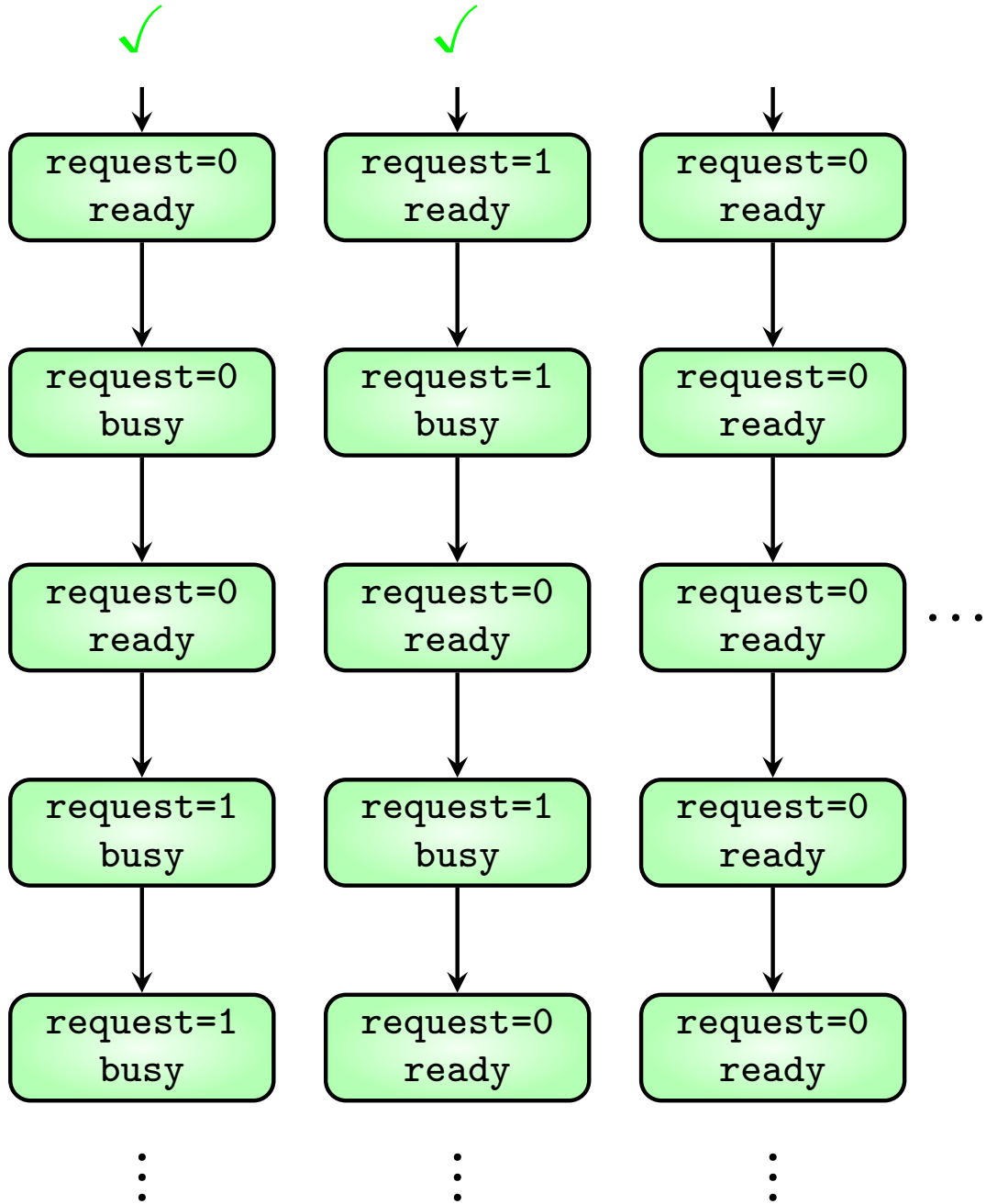
F (request=1)


request=1
ready
request=1
busy
request=0
ready
request=0
busy

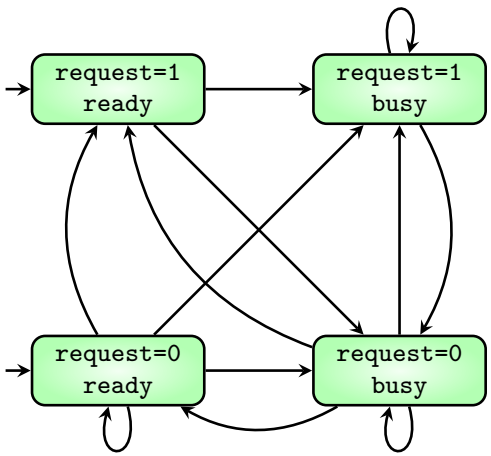

request=0
ready
request=0
busy
request=0
ready
request=1
busy
request=1
busy
request=1
busy
request=1
ready
request=1
busy
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready

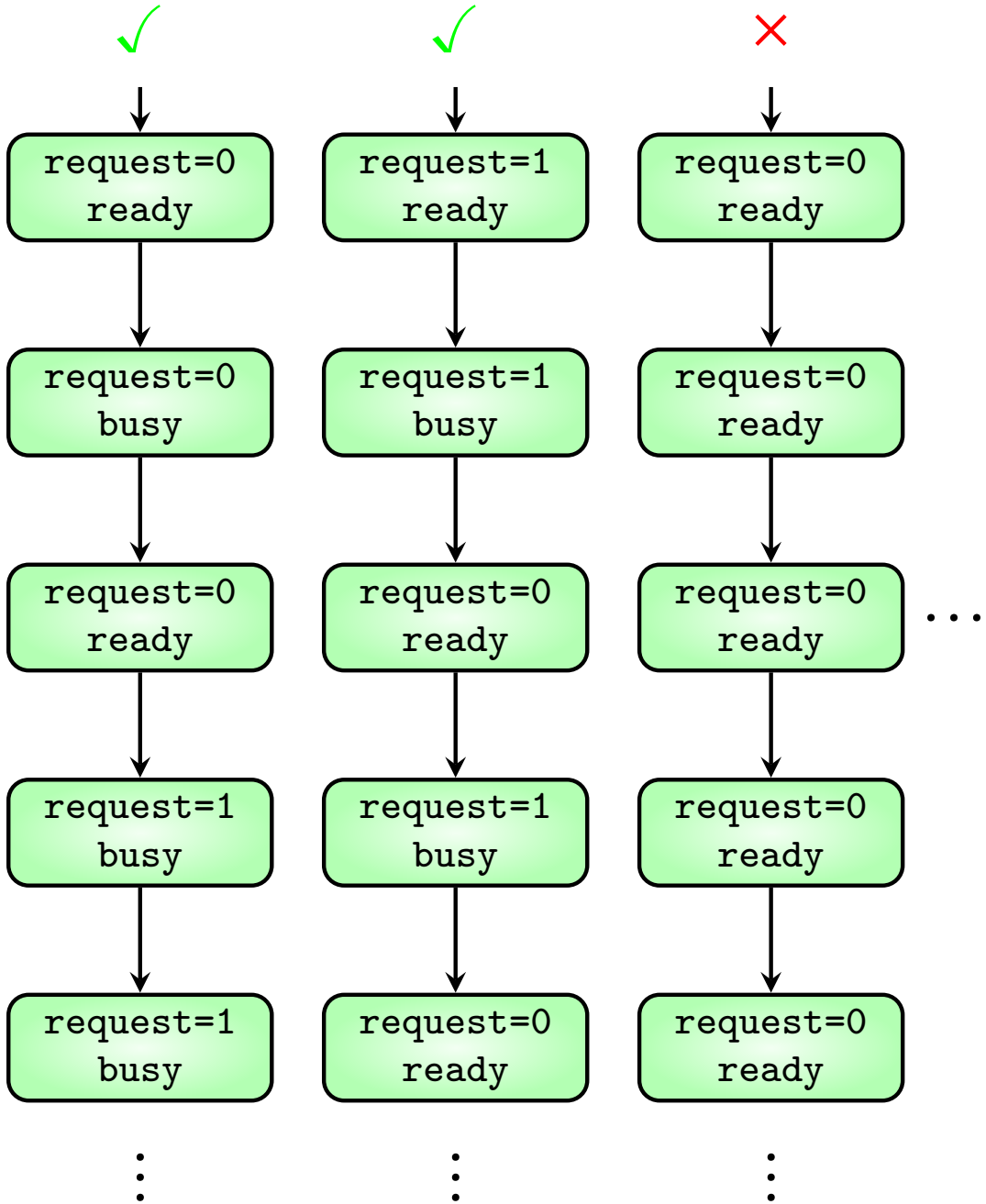
F (request=1)


request=1
ready
request=1
busy
request=0
ready
request=0
busy

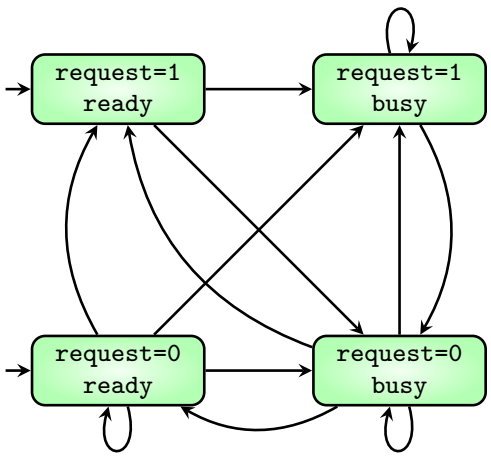

request=0
ready
request=0
busy
request=0
ready
request=1
busy
request=1
busy
request=1
busy
request=1
ready
request=1
busy
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready

F (request=1)


request=1
ready
request=1
busy
request=0
ready
request=0
busy

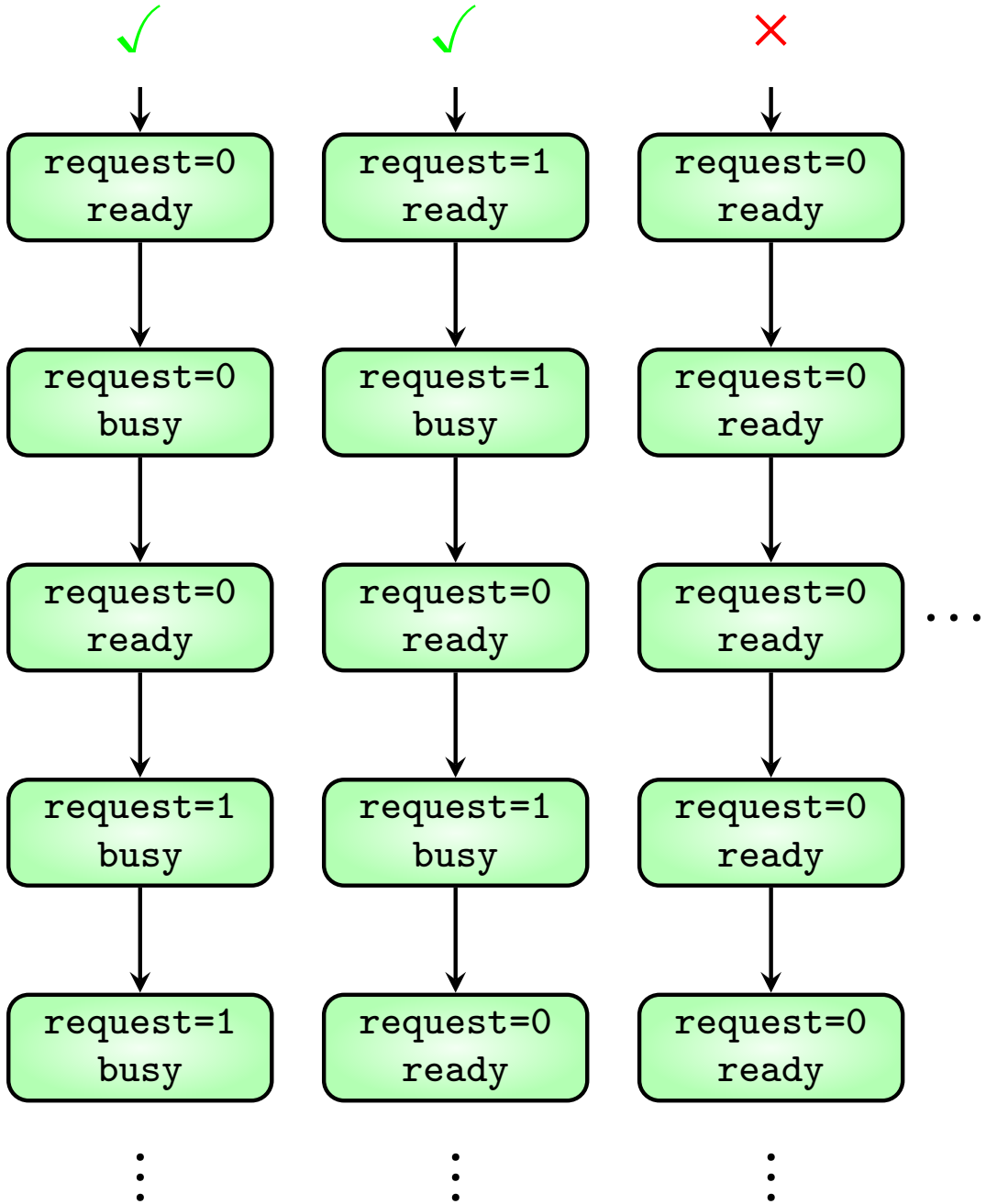

request=0
ready
request=0
busy
request=0
ready
request=1
busy
request=1
busy
request=1
busy
request=1
ready
request=1
busy
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready

F (request=1)



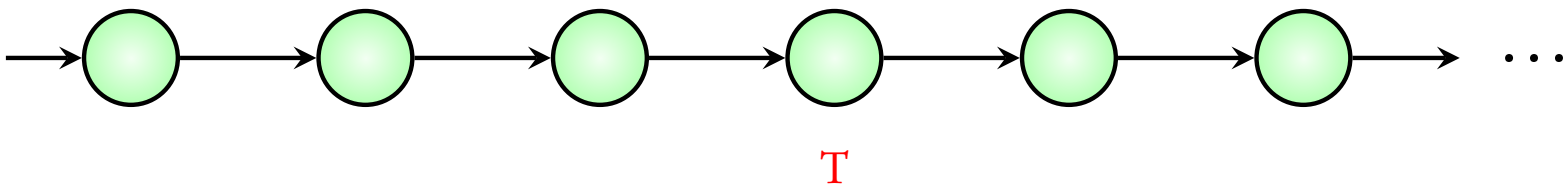
TS does not satisfy

F (request=1)



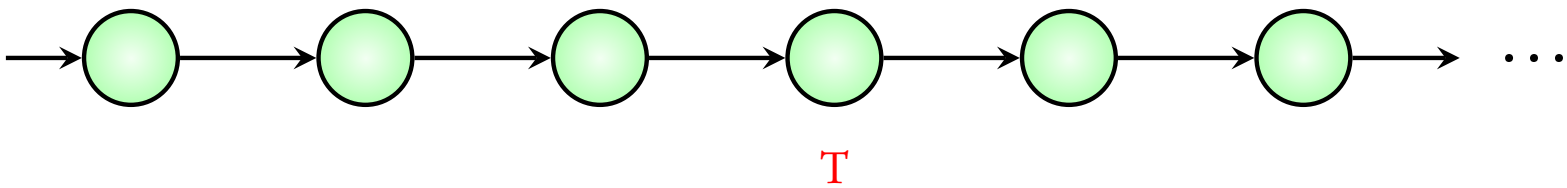
Execution **satisfies** F `(expr)` if

`expr` evaluates to T in **one of its states**



Execution **satisfies** `F (expr)` if

`expr` evaluates to T in **one of its states**



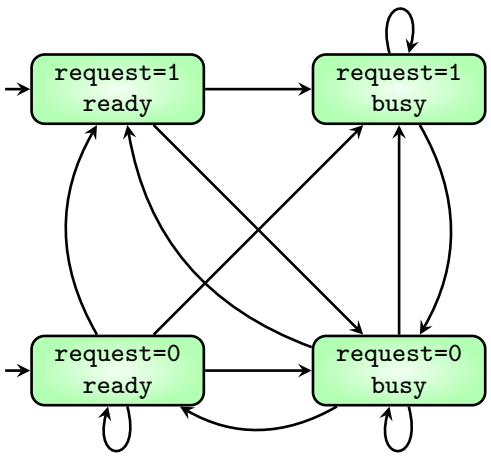
Transition system **satisfies** `F (expr)` if

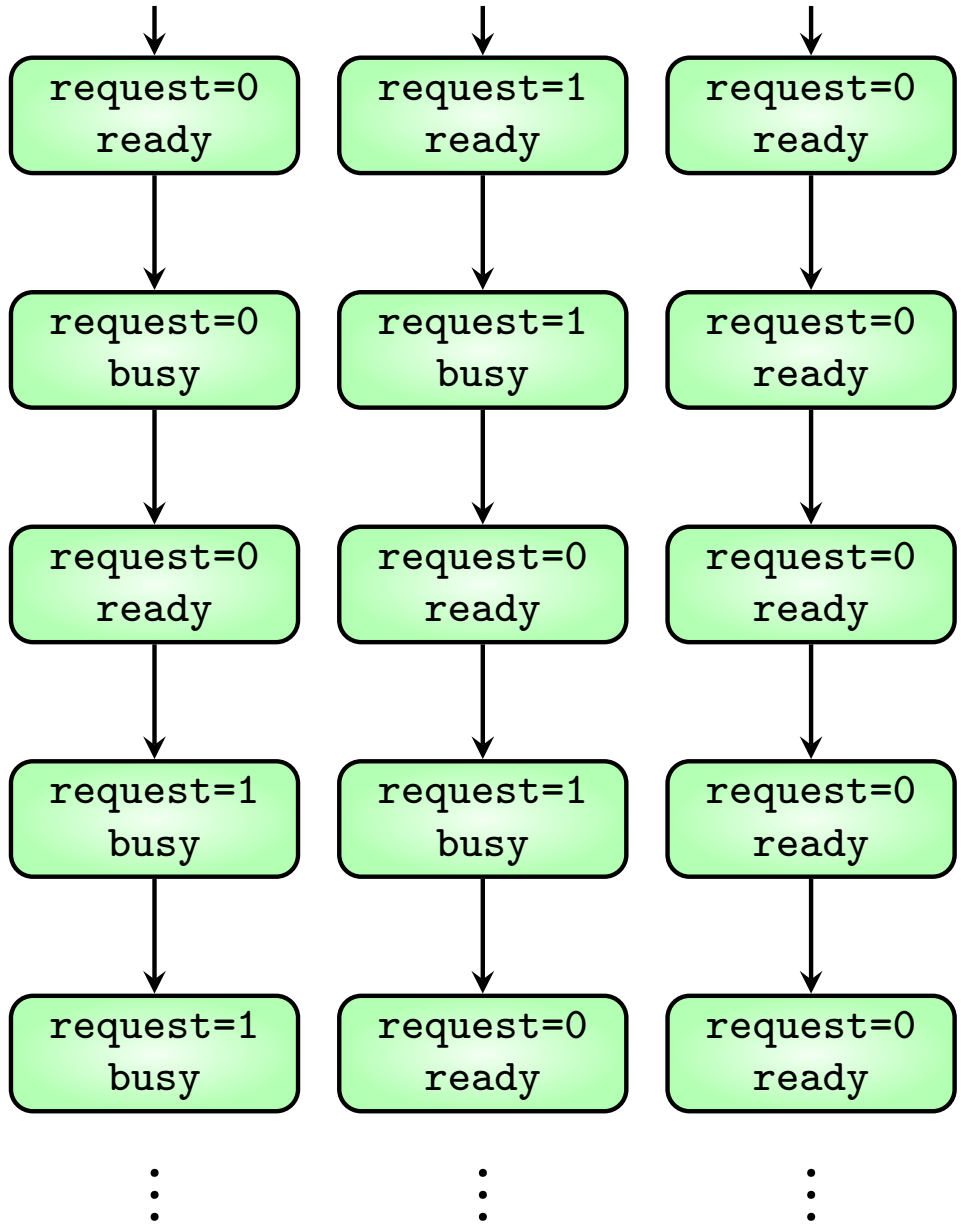
all its executions satisfy `F (expr)`

Checking the **F** requirement: **NuSMV demo**

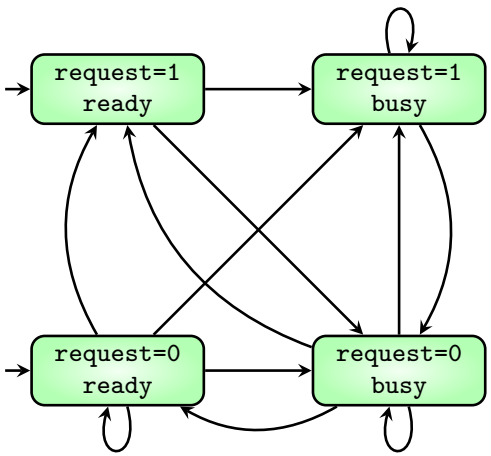
Coming next: Combining G and F

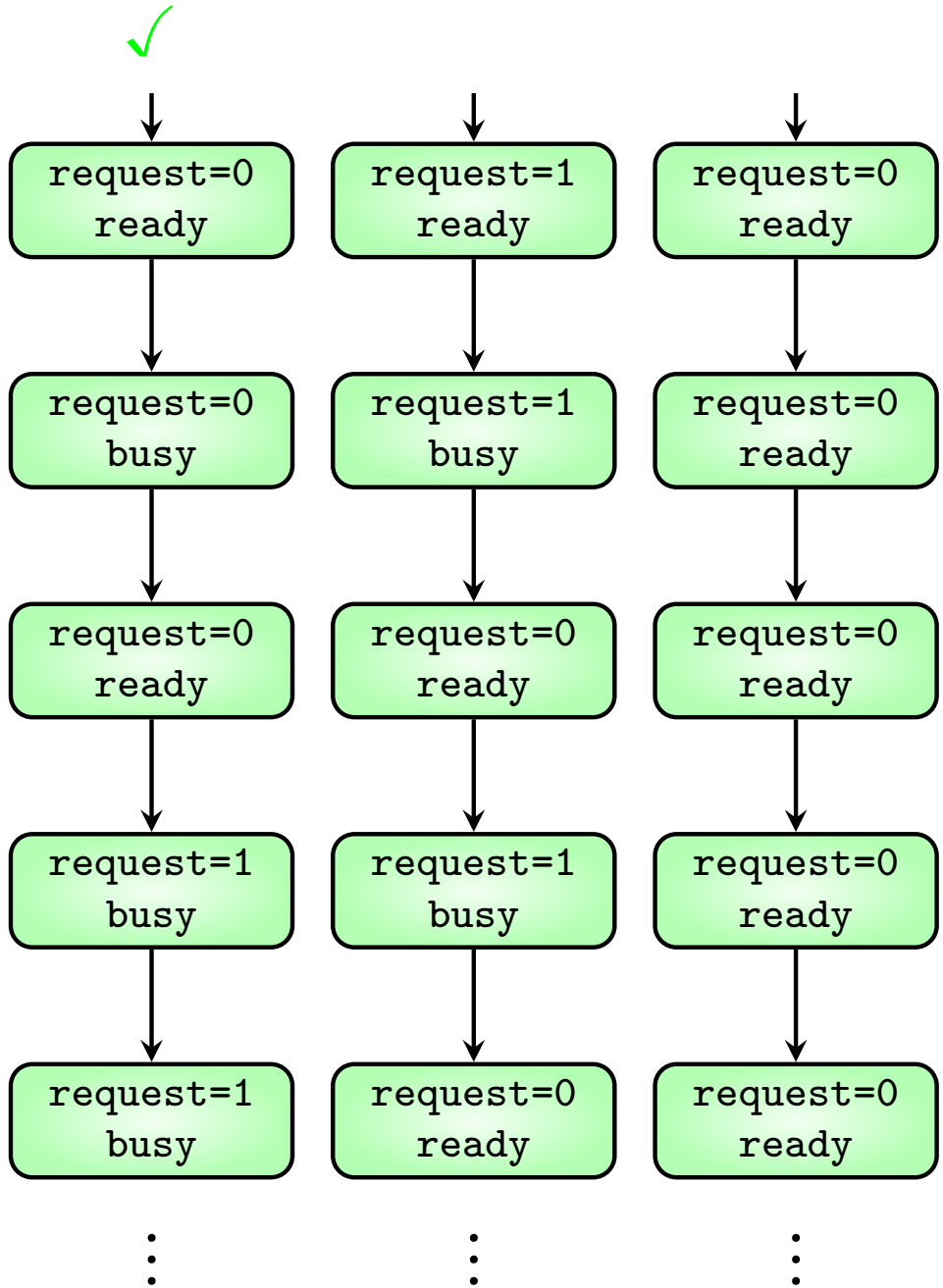
G (request=1 => F status=busy)



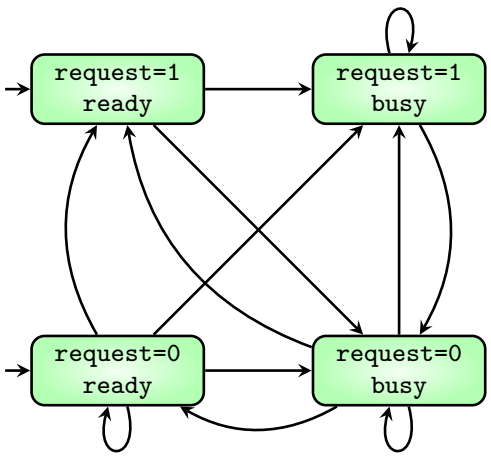


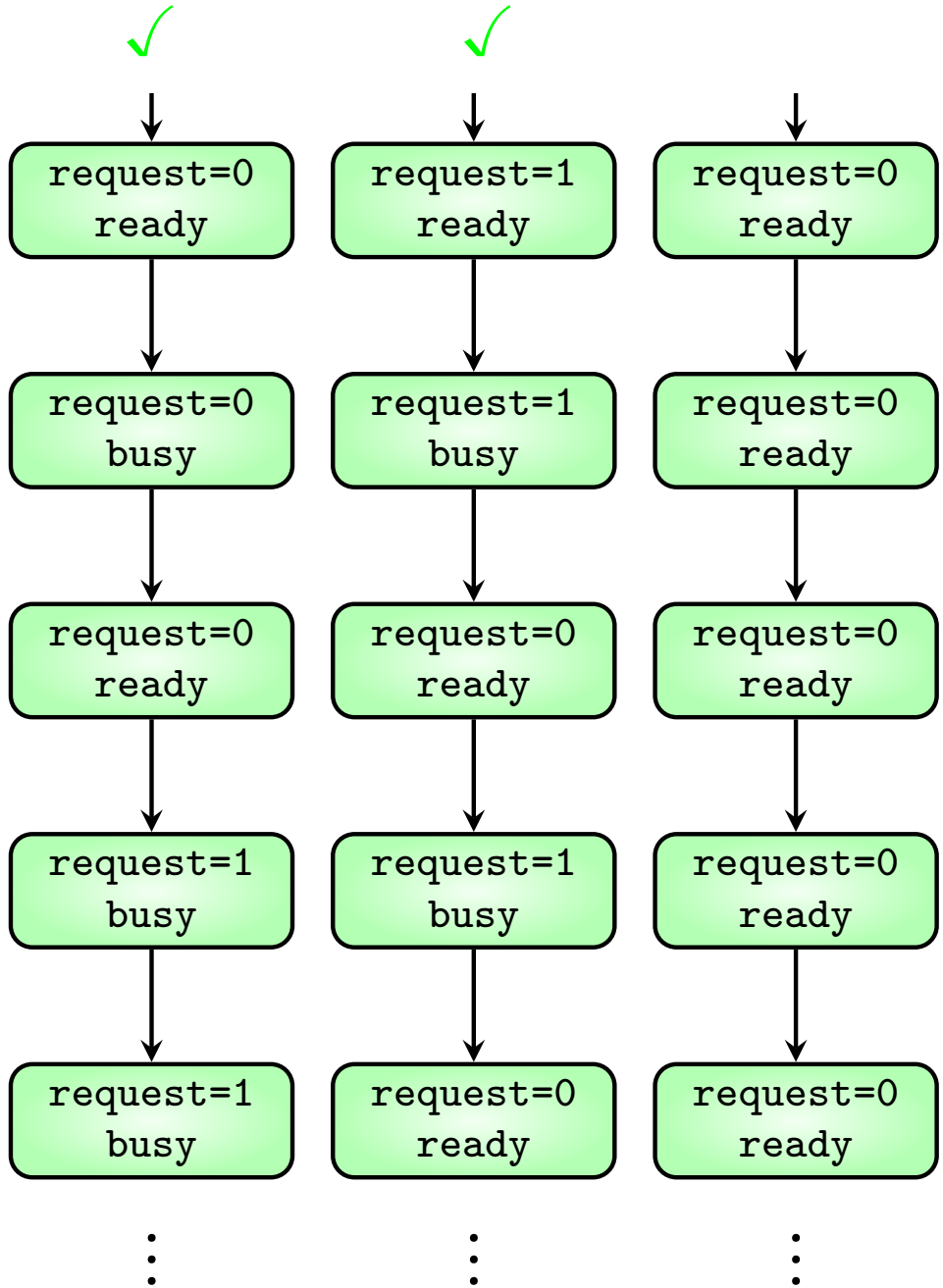
G (request=1 => F status=busy)



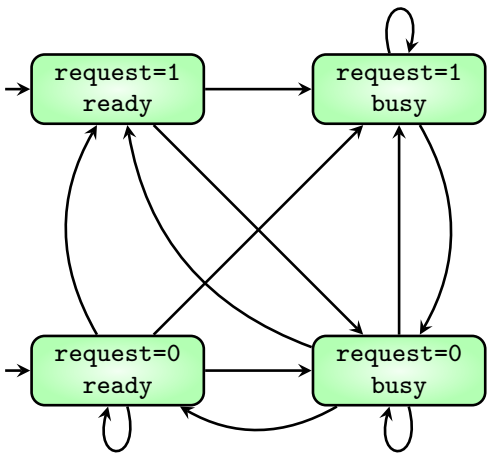


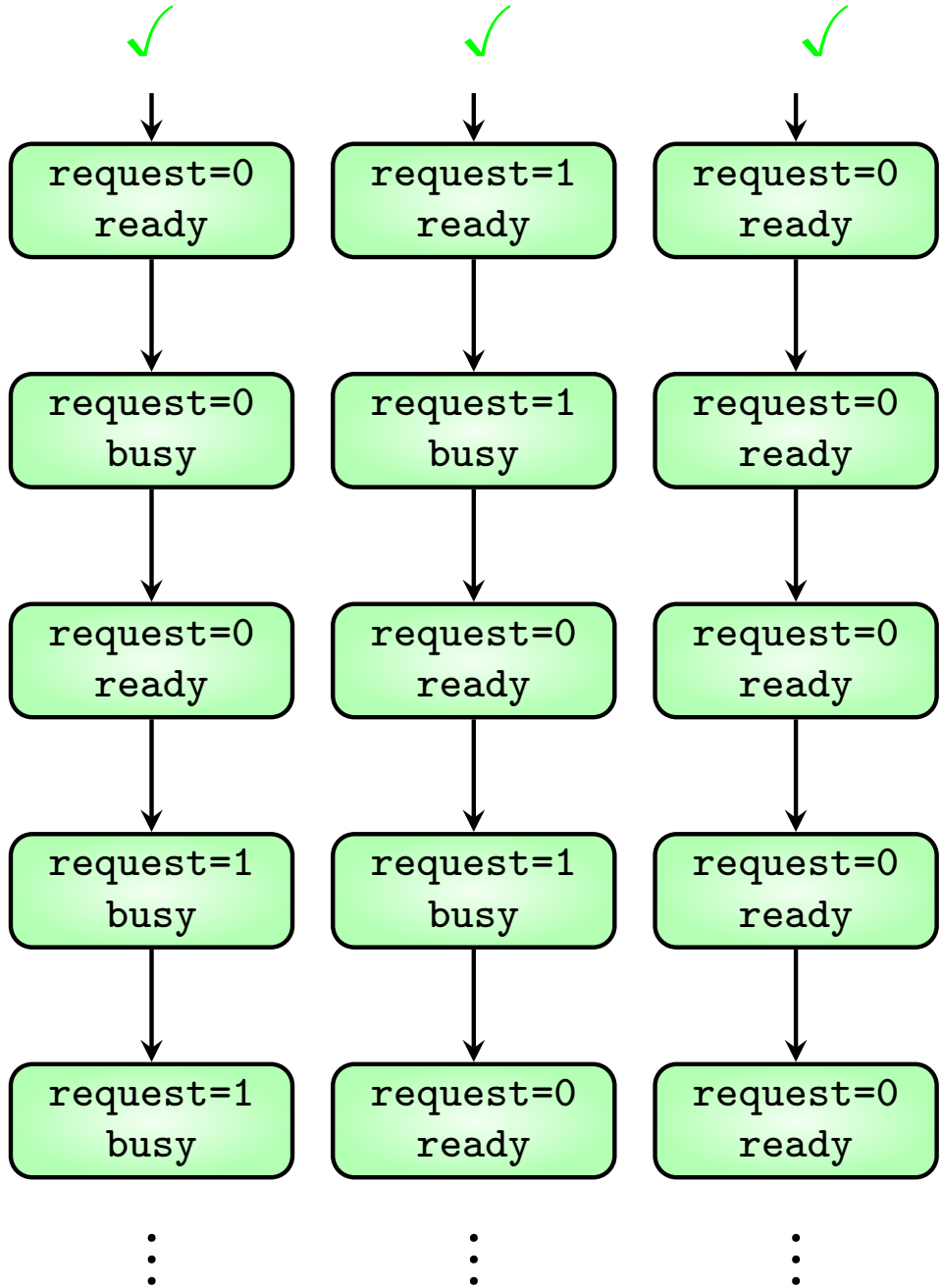
G (request=1 => F status=busy)



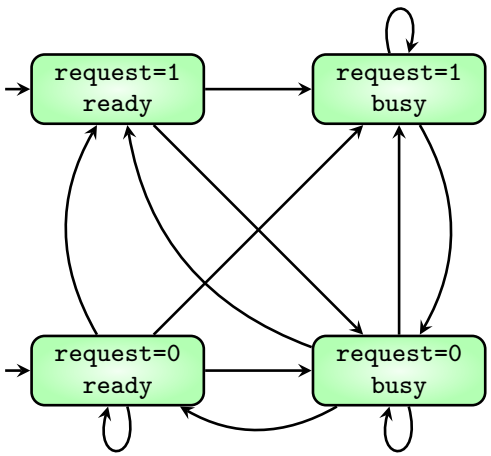


G (request=1 => F status=busy)



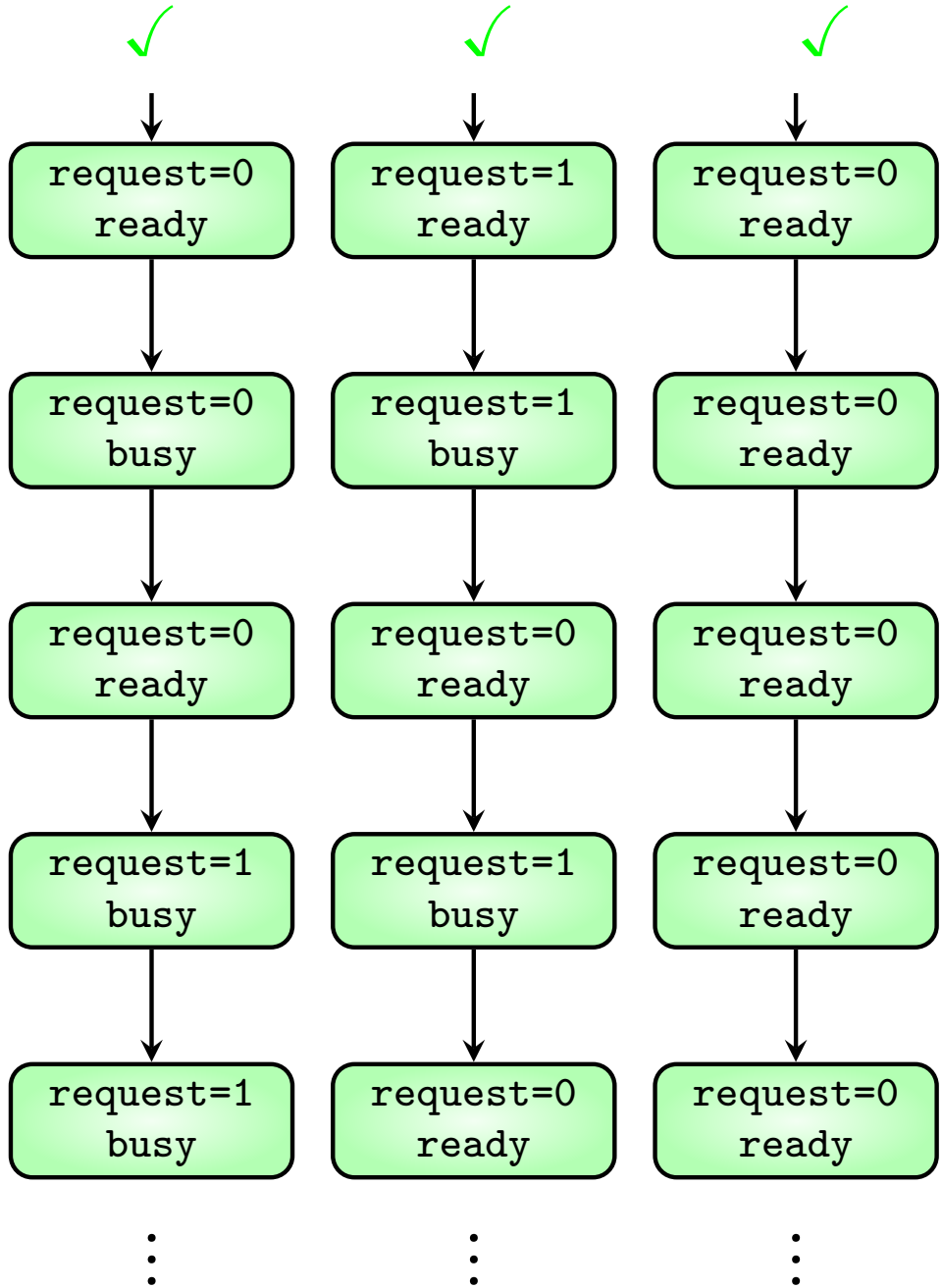


G (request=1 => F status=busy)



TS satisfies

G (request => F (status=busy))



Summary

Using NuSMV

Format for writing models

G and F requirements

Summary

Using NuSMV

Format for writing models

G and F requirements

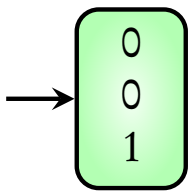
Coming next: More circuits

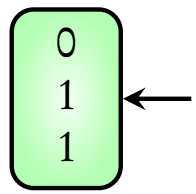

in_1
in_2
NAND
out

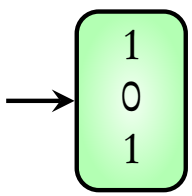

```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
DEFINE
   -- ZERO DELAY
   out := !(in1 & in2);
```

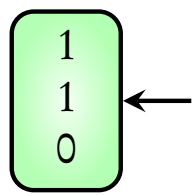
in_1	in_2	out
0	0	1
0	1	1
1	0	1
1	1	0

```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
DEFINE
   -- ZERO DELAY
   out := !(in1 & in2);
```

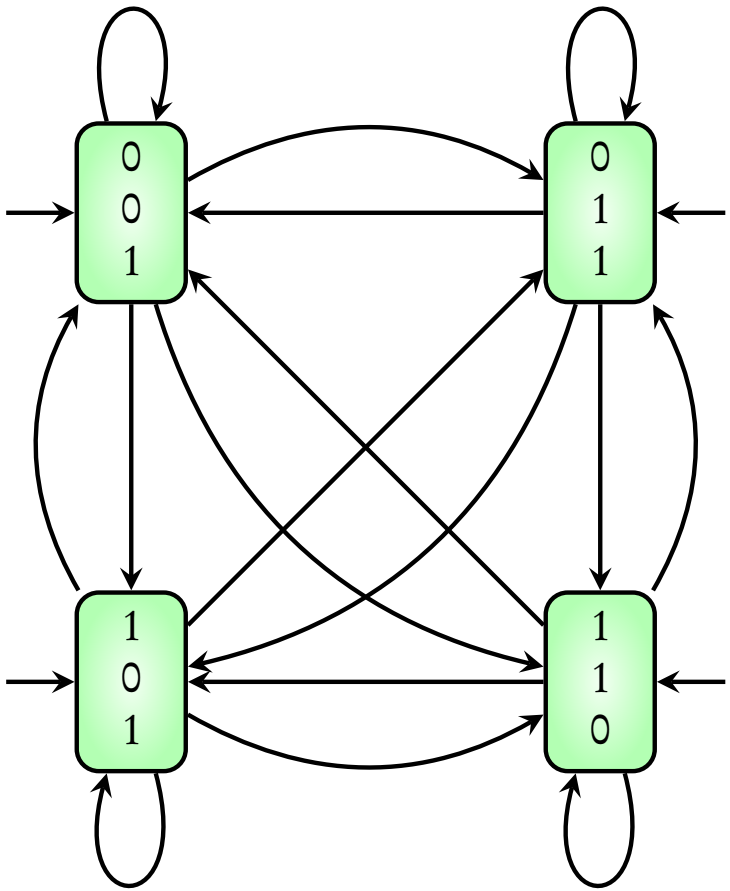






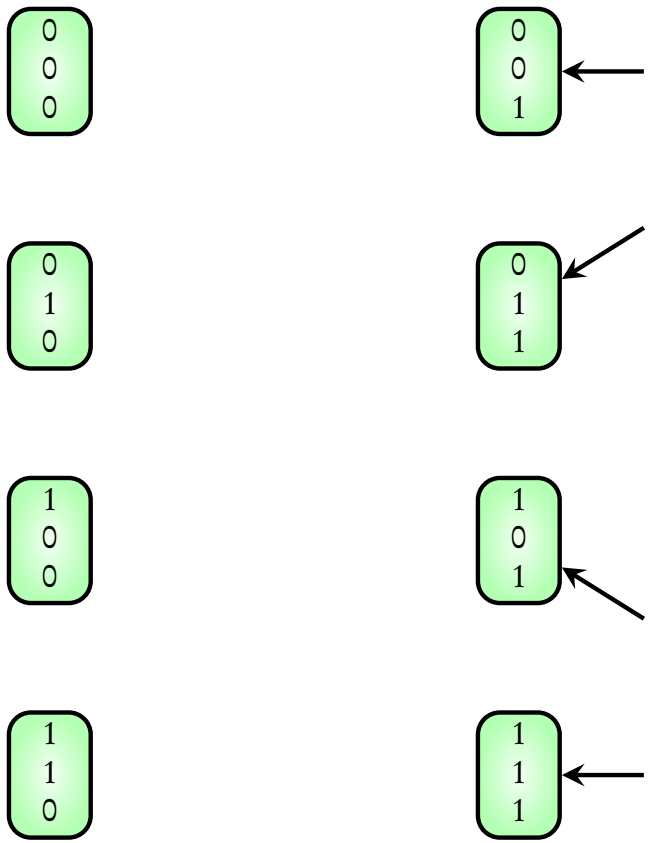



```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
DEFINE
   -- ZERO DELAY
   out := !(in1 & in2);
```

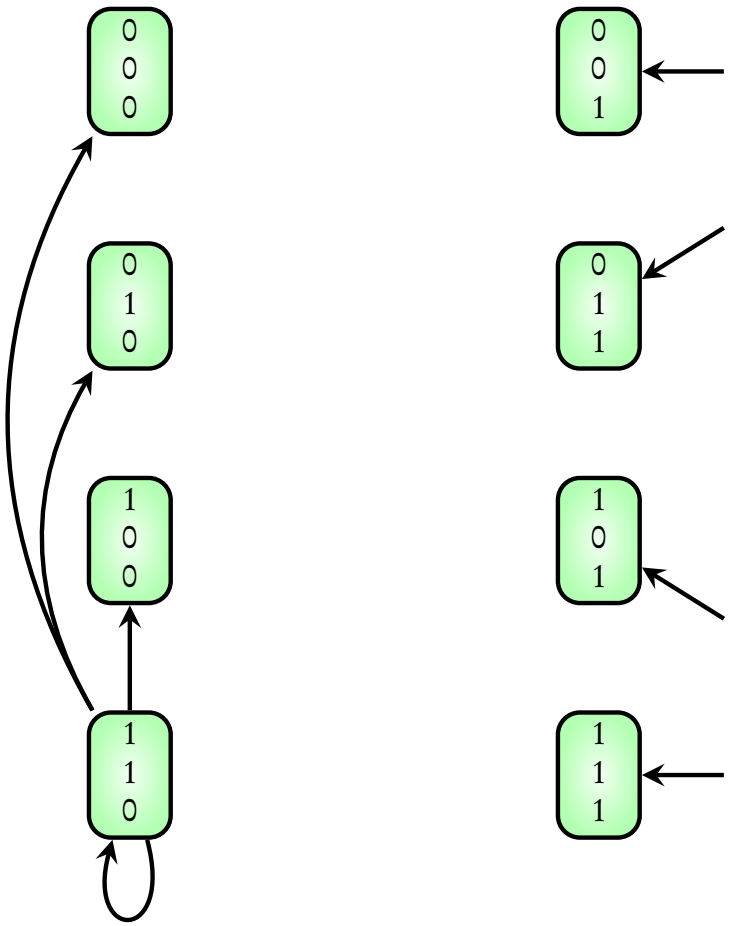



```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
DEFINE
   -- ZERO DELAY
   out := !(in1 & in2);
```

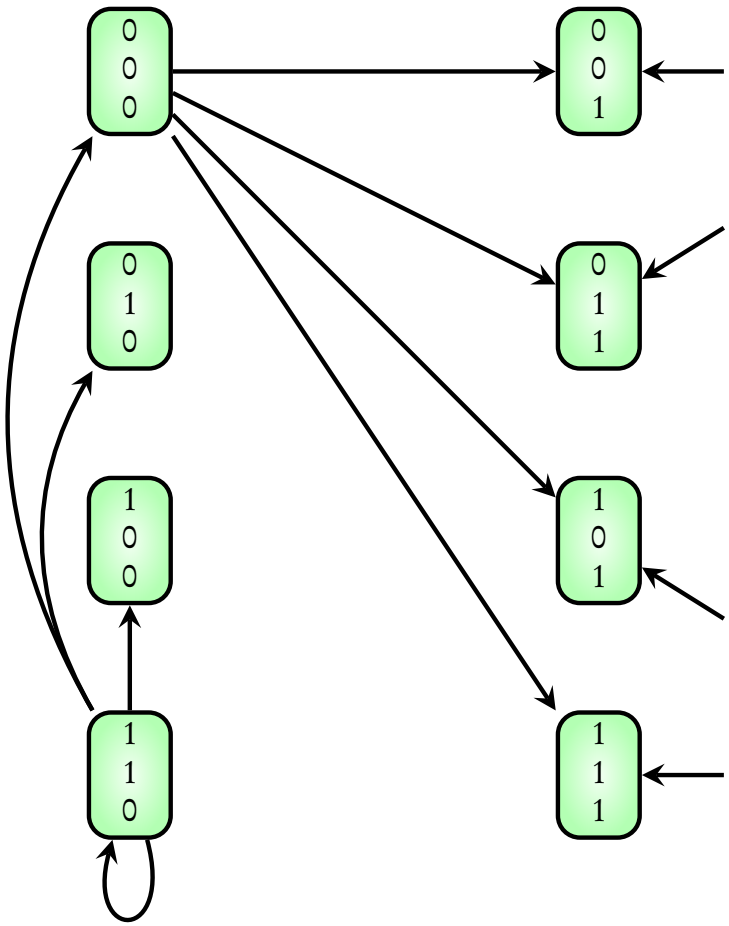

```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```



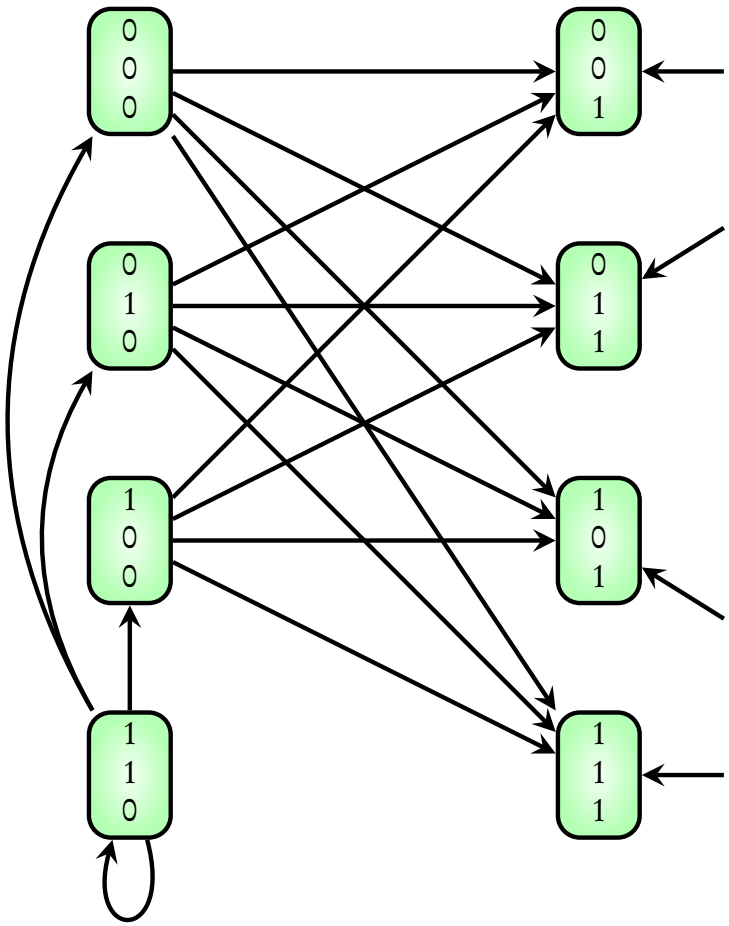

```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```



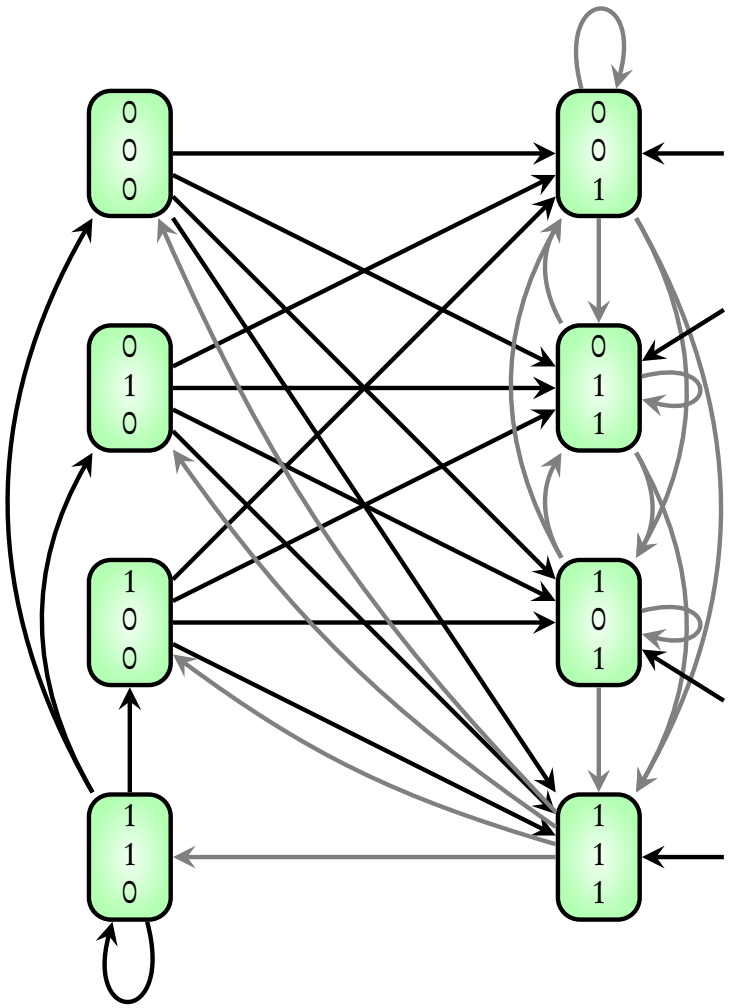

```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```



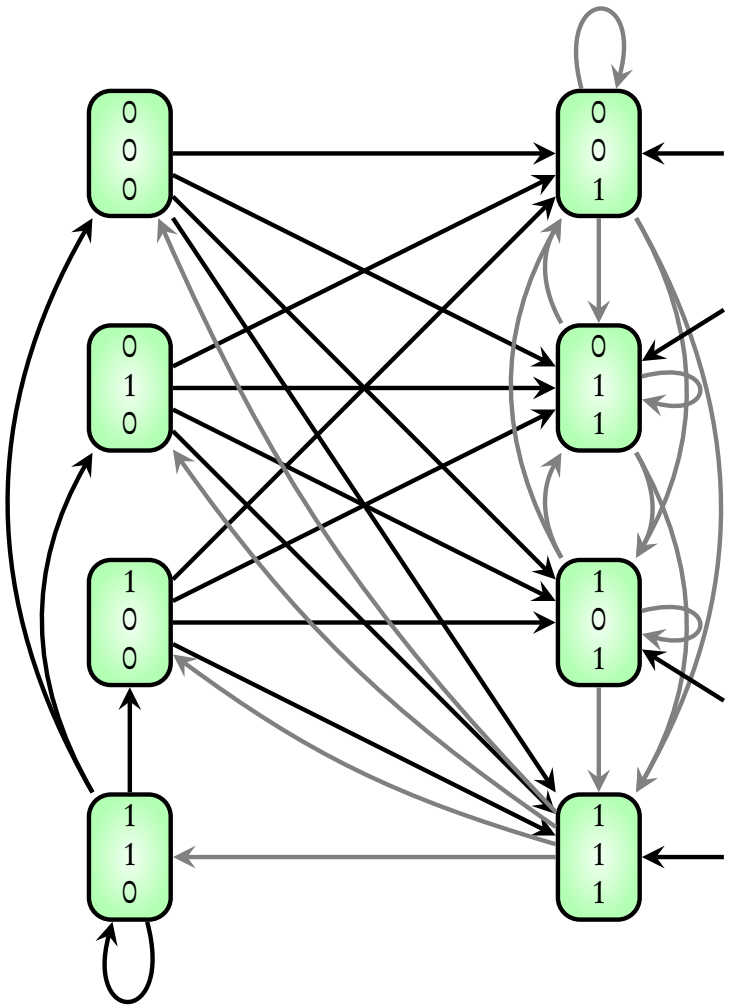

```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```



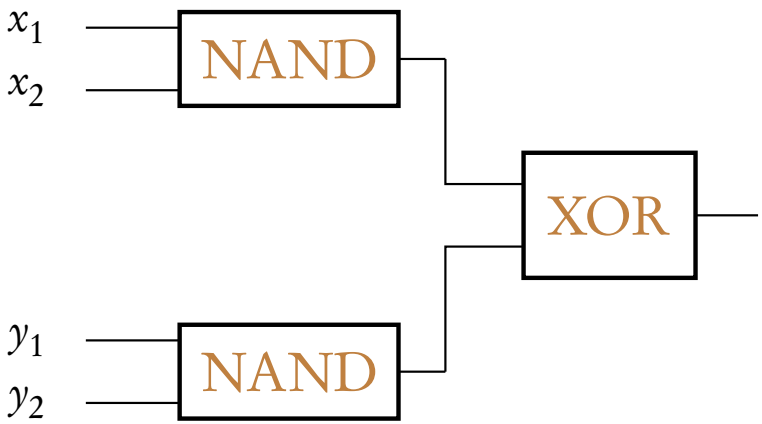

```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```



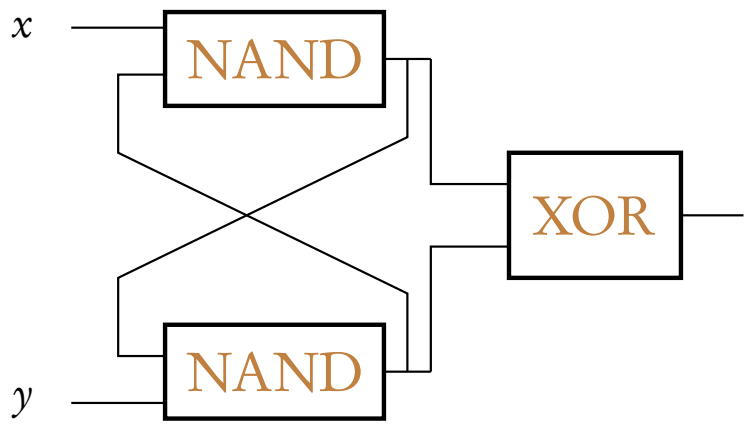

```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```




```
MODULE main
VAR
   input1:  boolean;
   input2:  boolean;
   q:  nand2(input1, input2);

MODULE nand2(in1, in2)
VAR
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```



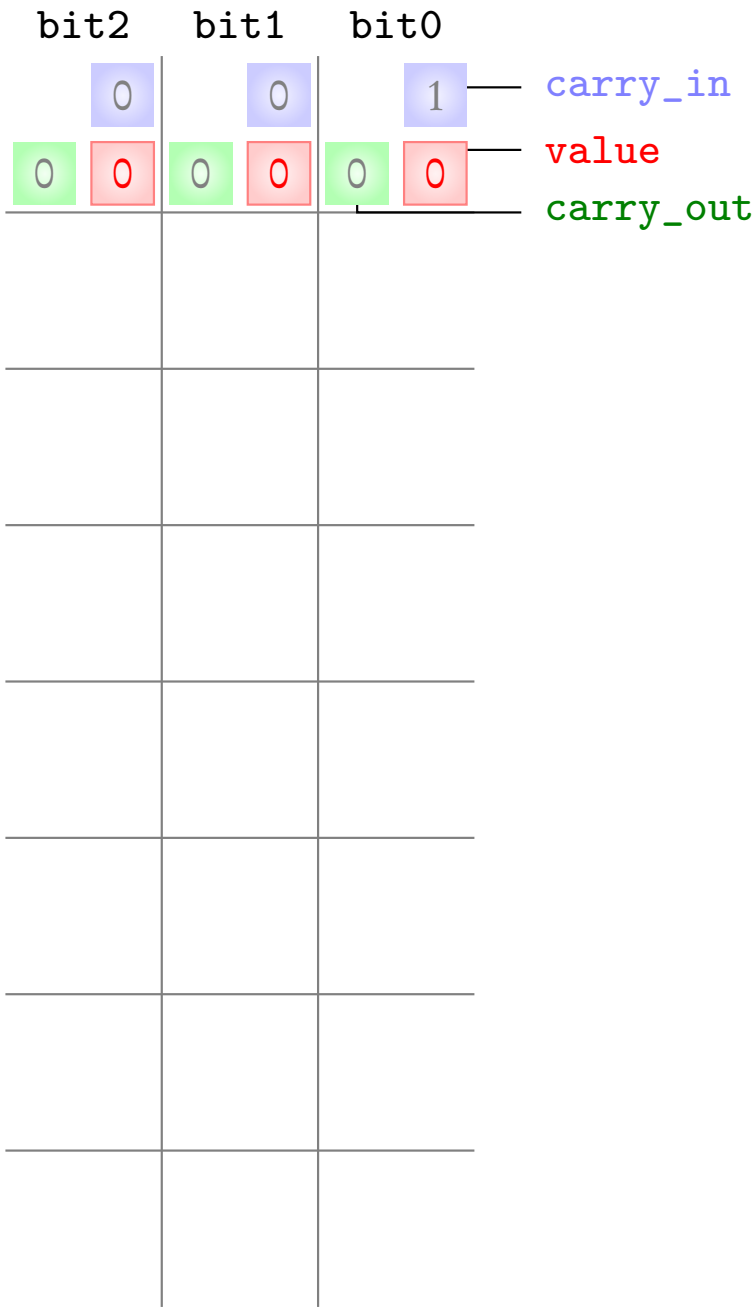

```
MODULE main
VAR
   x1:  boolean; x2:boolean;
   y1:  boolean; y2:boolean;
   q1:  nand2(x1, x2);
   q2:  nand2(y1, y2);
DEFINE
   -- ZERO DELAY
   fout := q1.out xor q2.out;


MODULE nand2(in1, in2)
VAR
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```



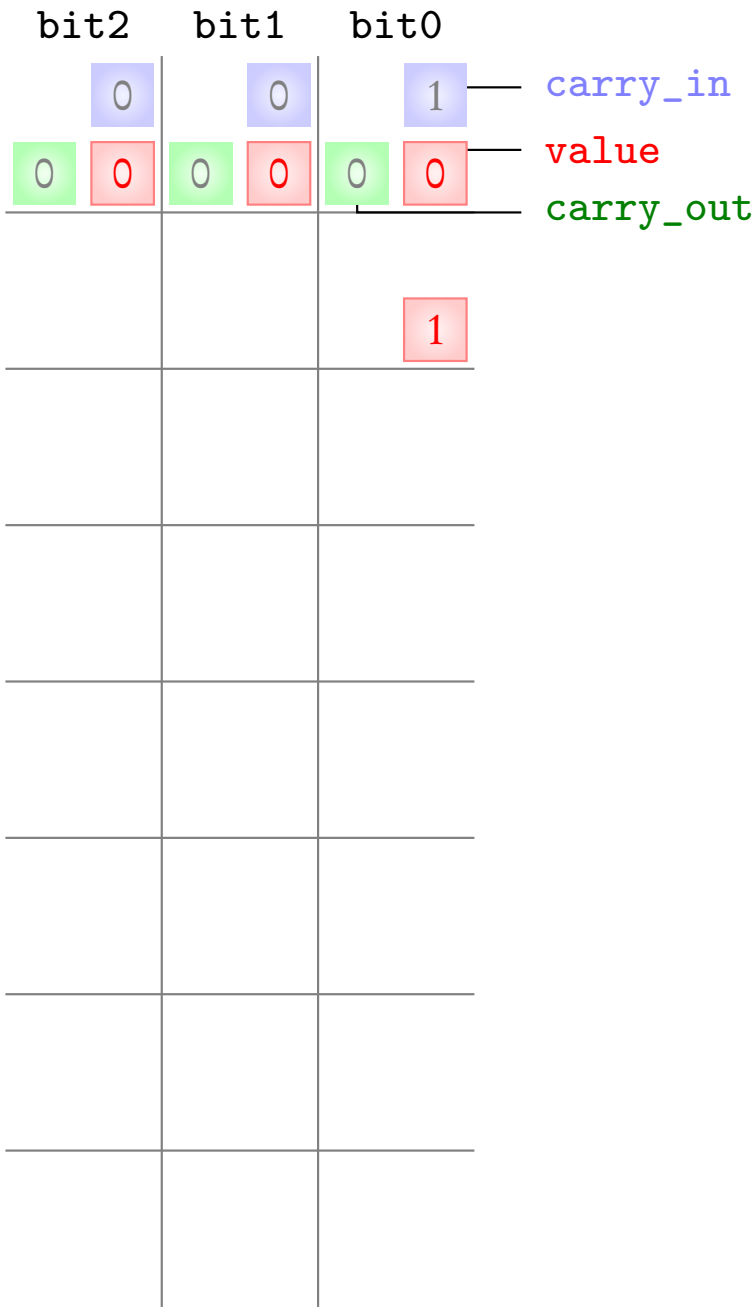

```
MODULE main
VAR
   x:  boolean;
   y:  boolean;
   q1:  nand2(x, q2.out);
   q2:  nand2(q1.out, y);
DEFINE
   -- ZERO DELAY
   fout := q1.out xor q2.out;

MODULE nand2(in1, in2)
VAR
   out:  boolean
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```

Coming next: Three-bit adder

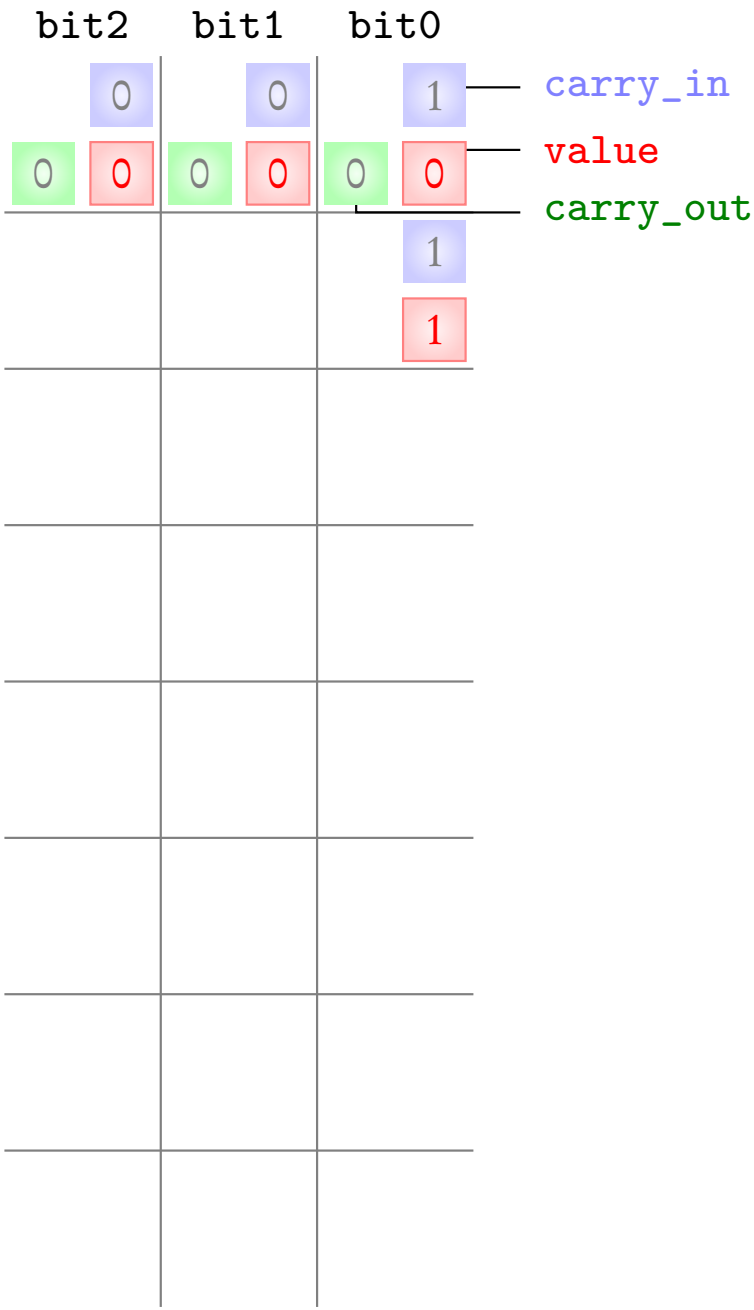
```
MODULE counter_cell(carry_in)
VAR
   value:boolean;
ASSIGN
   init(value):=FALSE;
   next(value):= value xor carry_in;
DEFINE
   carry_out := carry_in & value;

MODULE main
VAR
   bit0:counter_cell(TRUE);
   bit1:counter_cell(bit0.carry_out);
   bit2:counter_cell(bit1.carry_out);
```



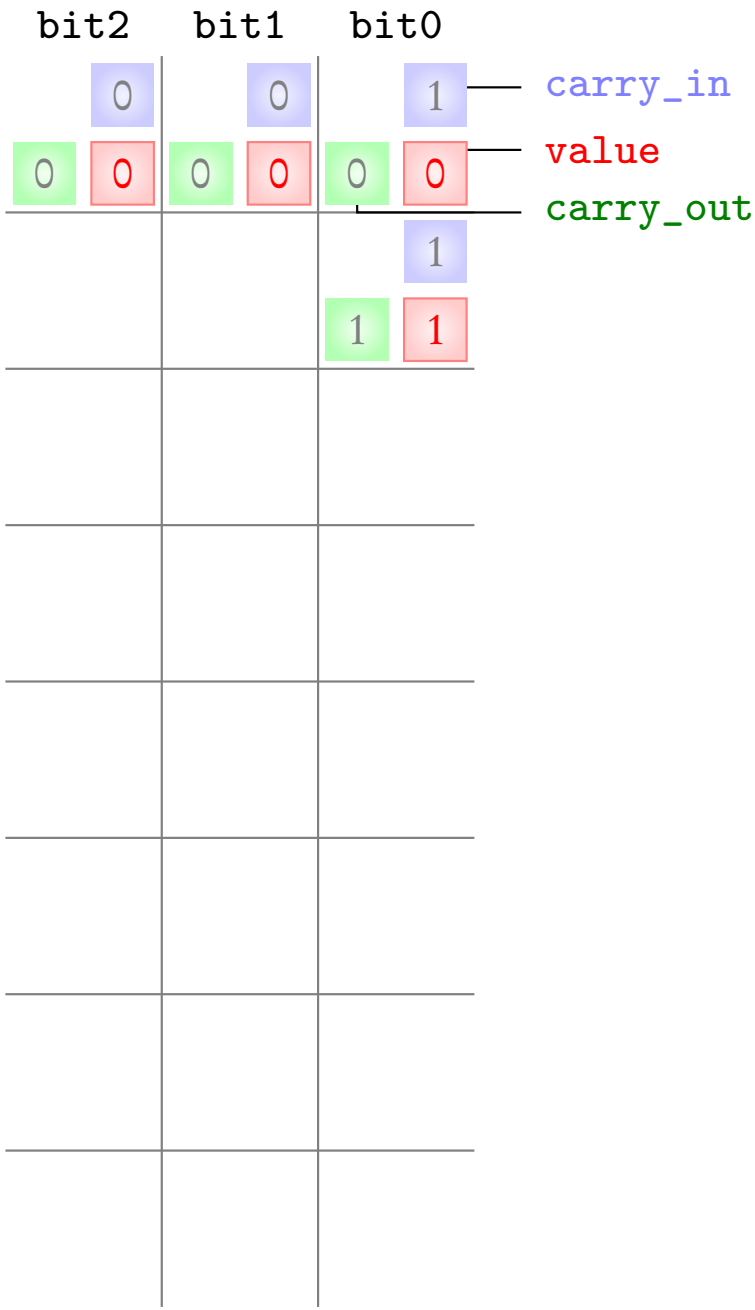

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


MODULE main
VAR
    bit0:counter_cell(TRUE);
    bit1:counter_cell(bit0.carry_out);
    bit2:counter_cell(bit1.carry_out);
```



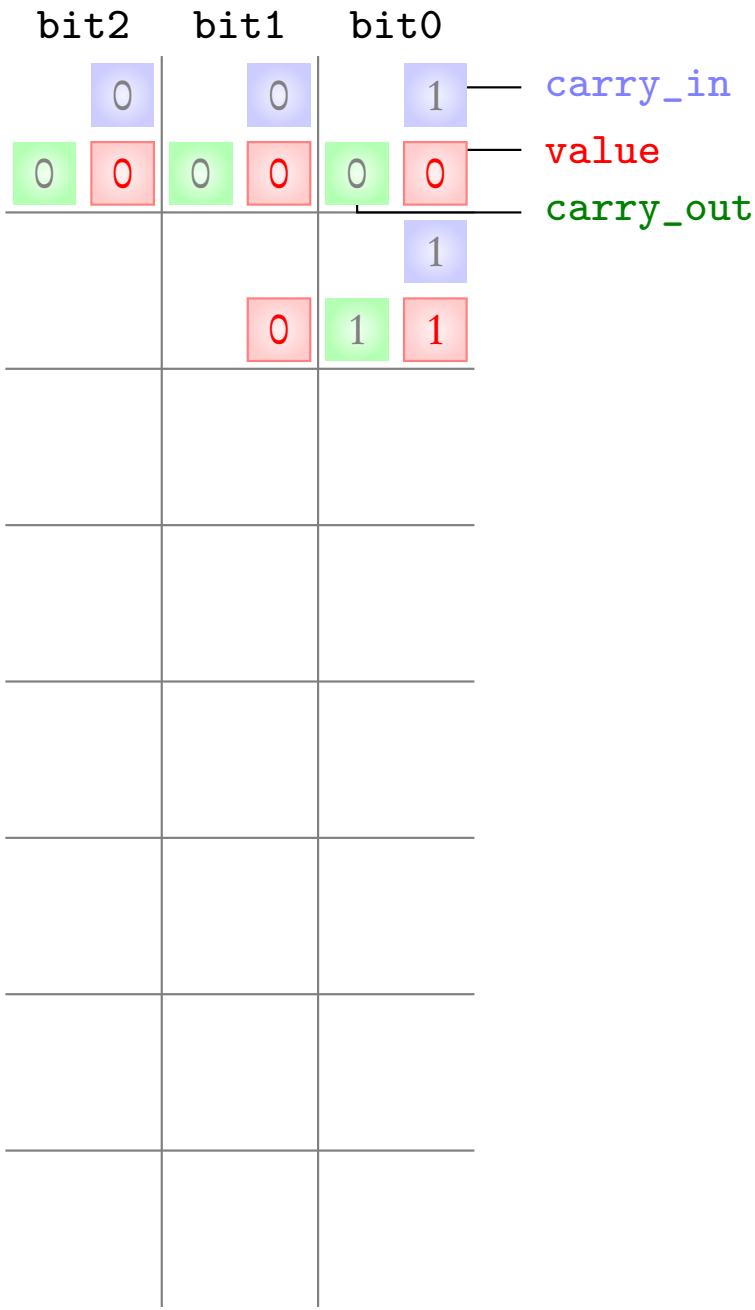

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


MODULE main
VAR
    bit0:counter_cell(TRUE);
    bit1:counter_cell(bit0.carry_out);
    bit2:counter_cell(bit1.carry_out);
```



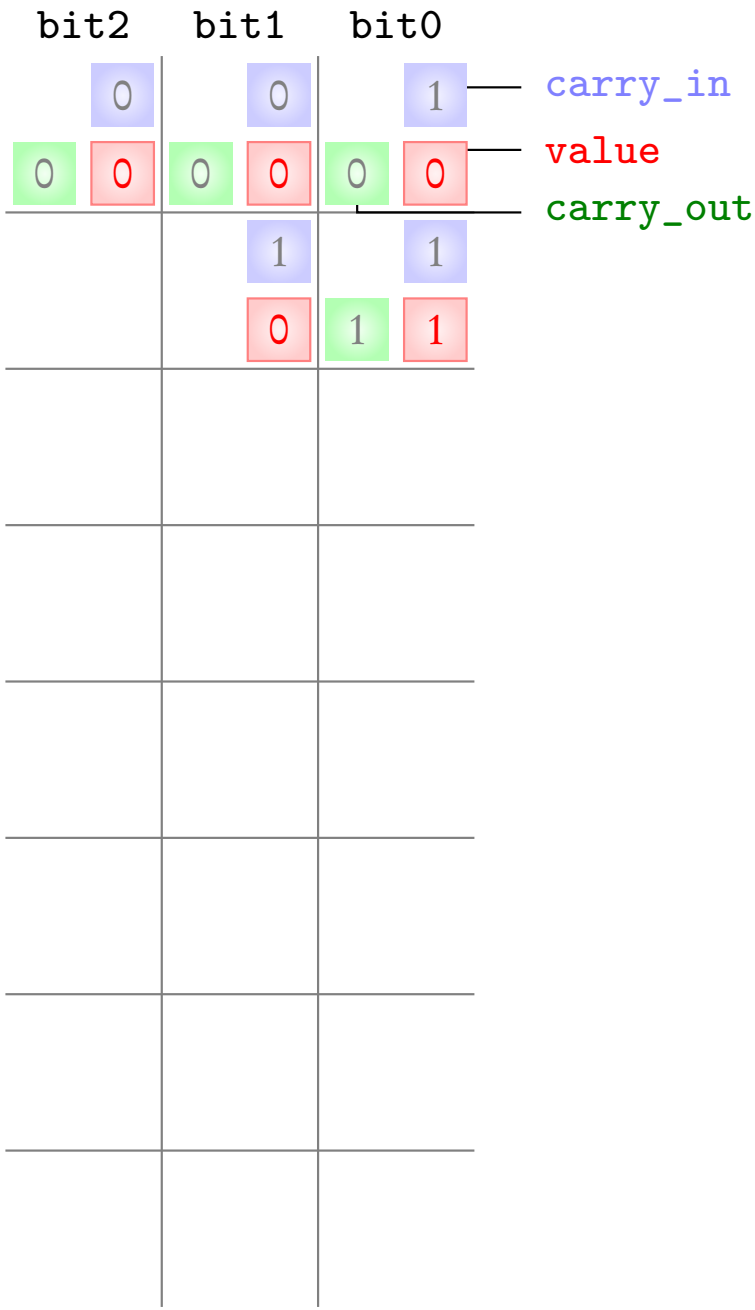

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


MODULE main
VAR
    bit0:counter_cell(TRUE);
    bit1:counter_cell(bit0.carry_out);
    bit2:counter_cell(bit1.carry_out);
```



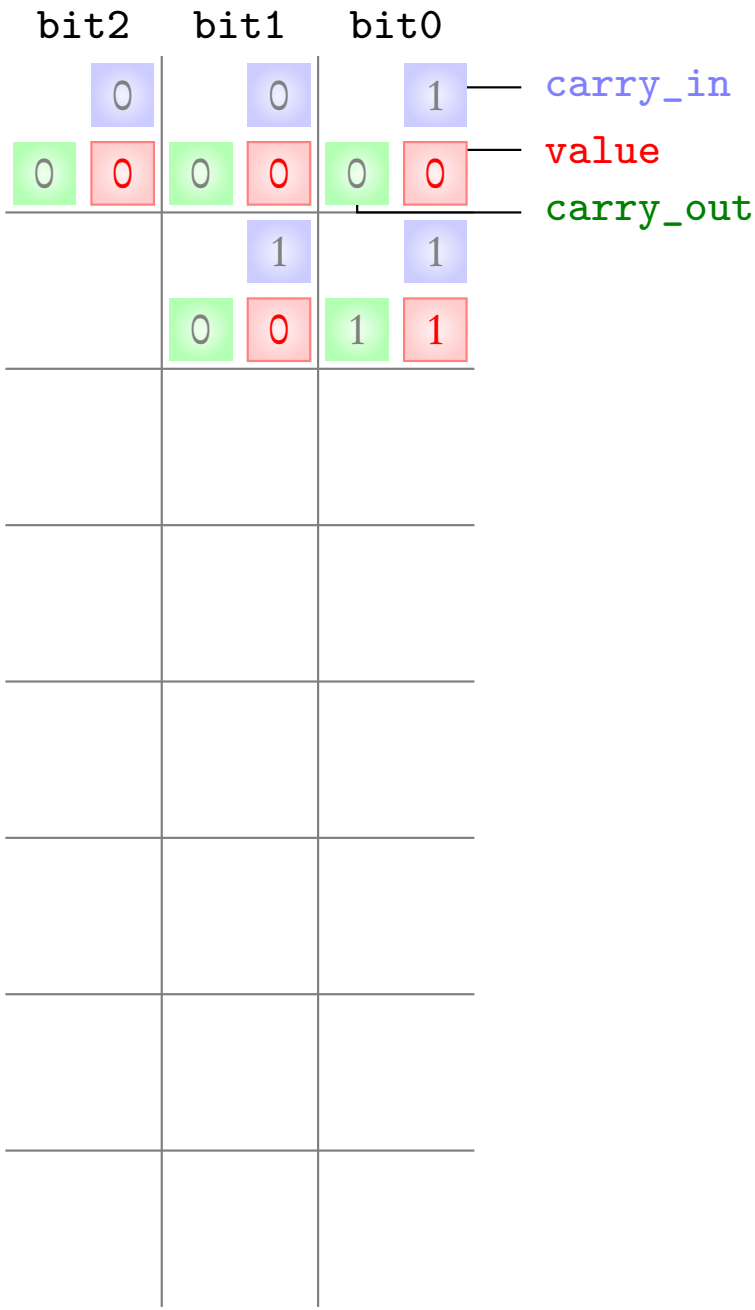

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


MODULE main
VAR
    bit0:counter_cell(TRUE);
    bit1:counter_cell(bit0.carry_out);
    bit2:counter_cell(bit1.carry_out);
```



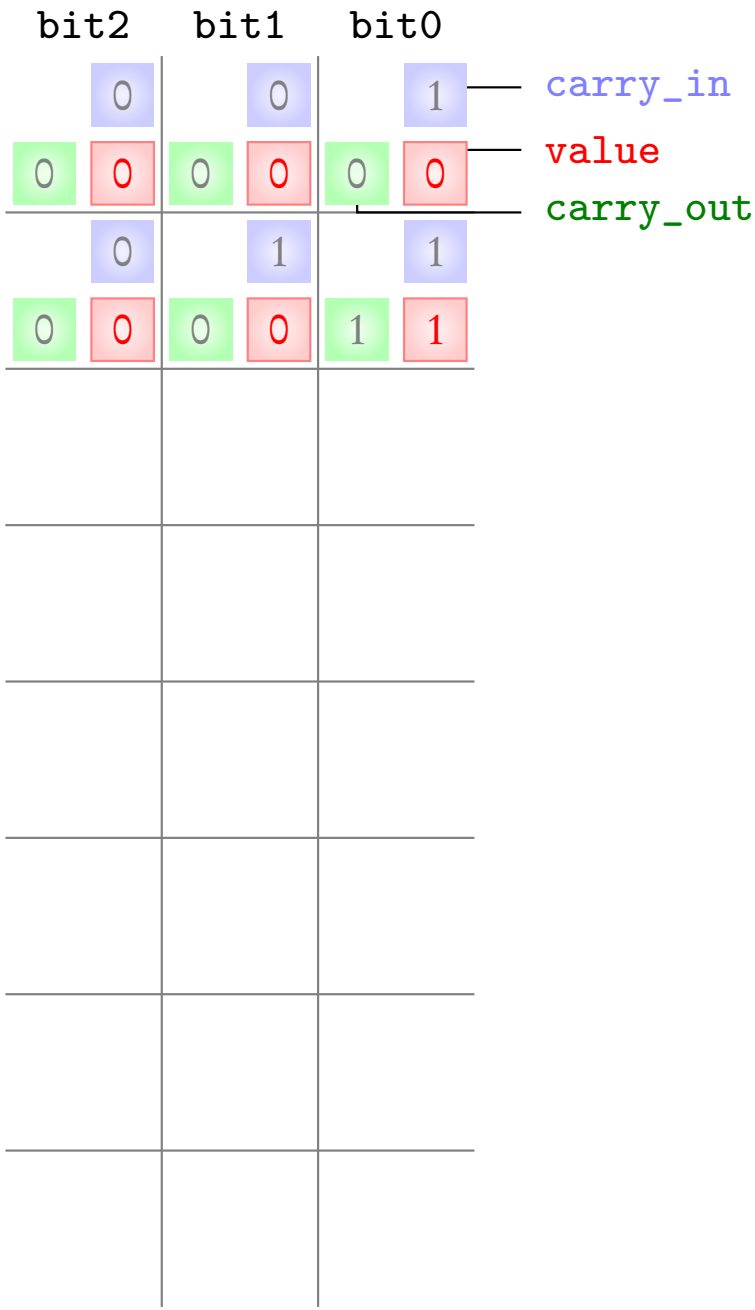

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


MODULE main
VAR
    bit0:counter_cell(TRUE);
    bit1:counter_cell(bit0.carry_out);
    bit2:counter_cell(bit1.carry_out);
```



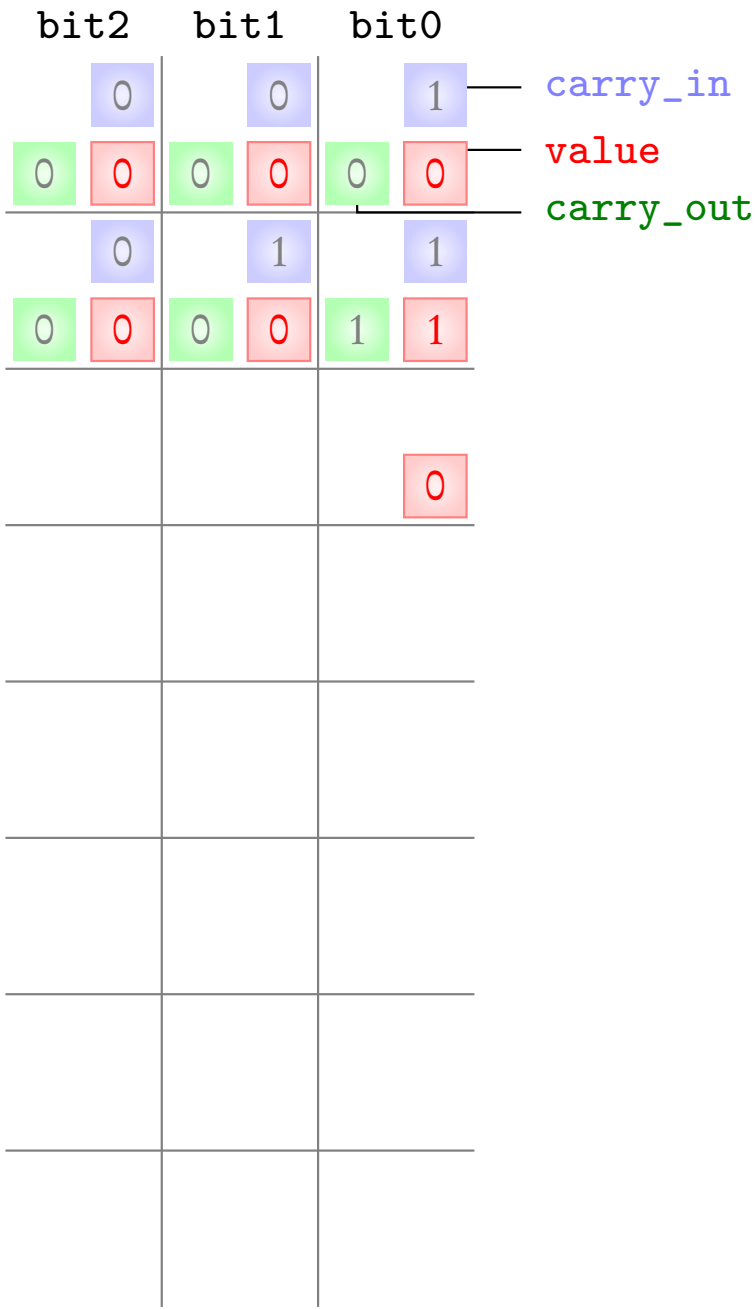

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


MODULE main
VAR
    bit0:counter_cell(TRUE);
    bit1:counter_cell(bit0.carry_out);
    bit2:counter_cell(bit1.carry_out);
```



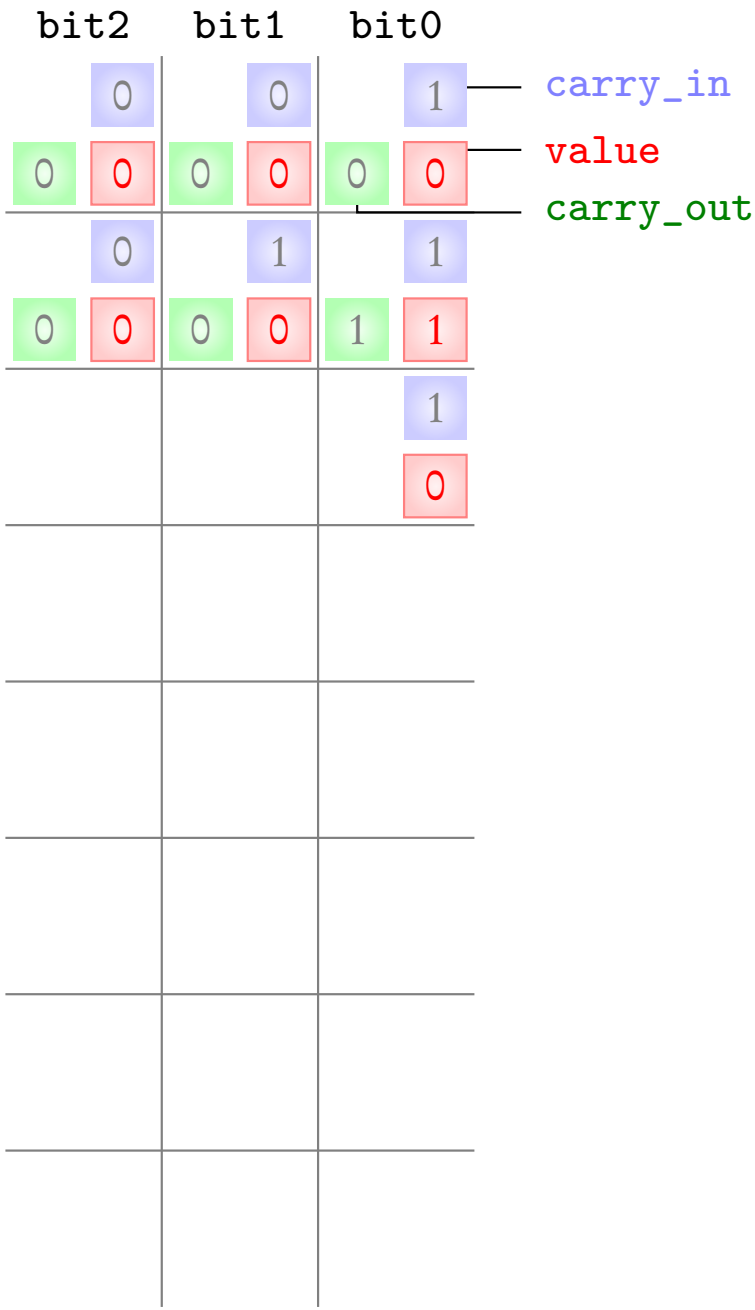

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


MODULE main
VAR
    bit0:counter_cell(TRUE);
    bit1:counter_cell(bit0.carry_out);
    bit2:counter_cell(bit1.carry_out);
```



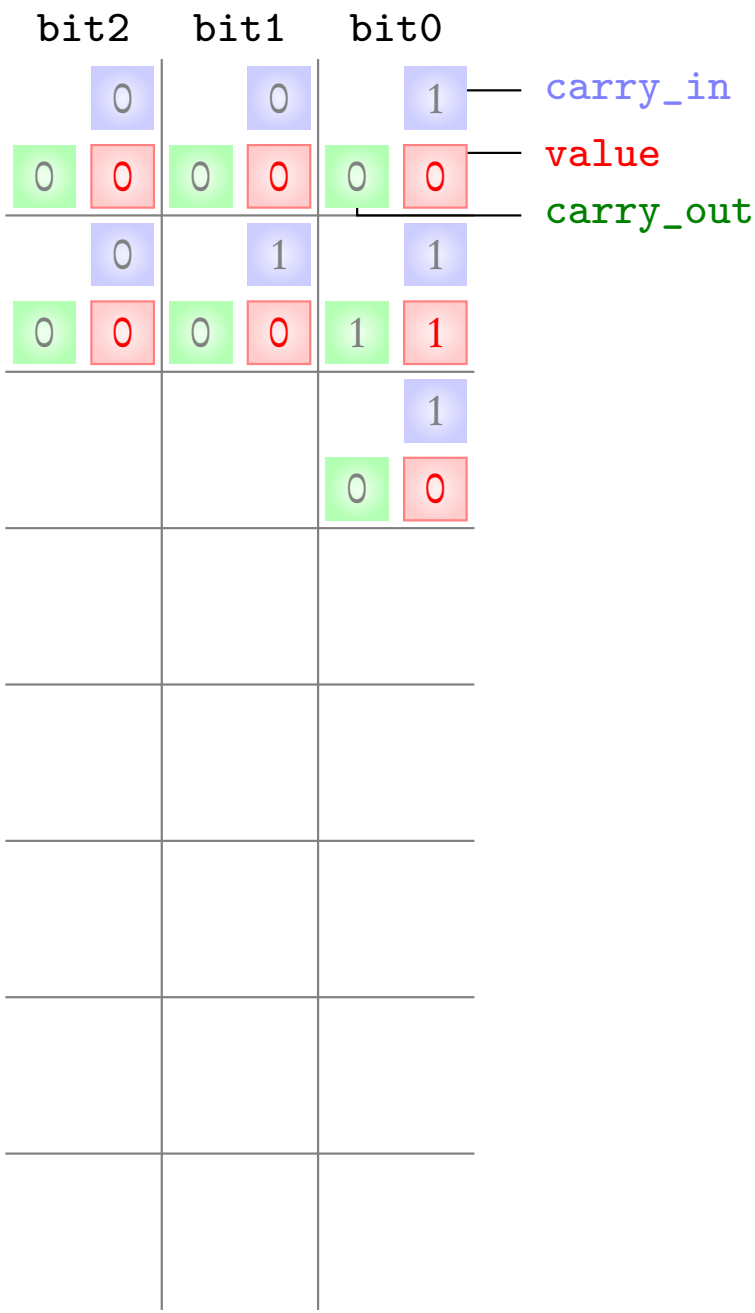

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


MODULE main
VAR
    bit0:counter_cell(TRUE);
    bit1:counter_cell(bit0.carry_out);
    bit2:counter_cell(bit1.carry_out);
```



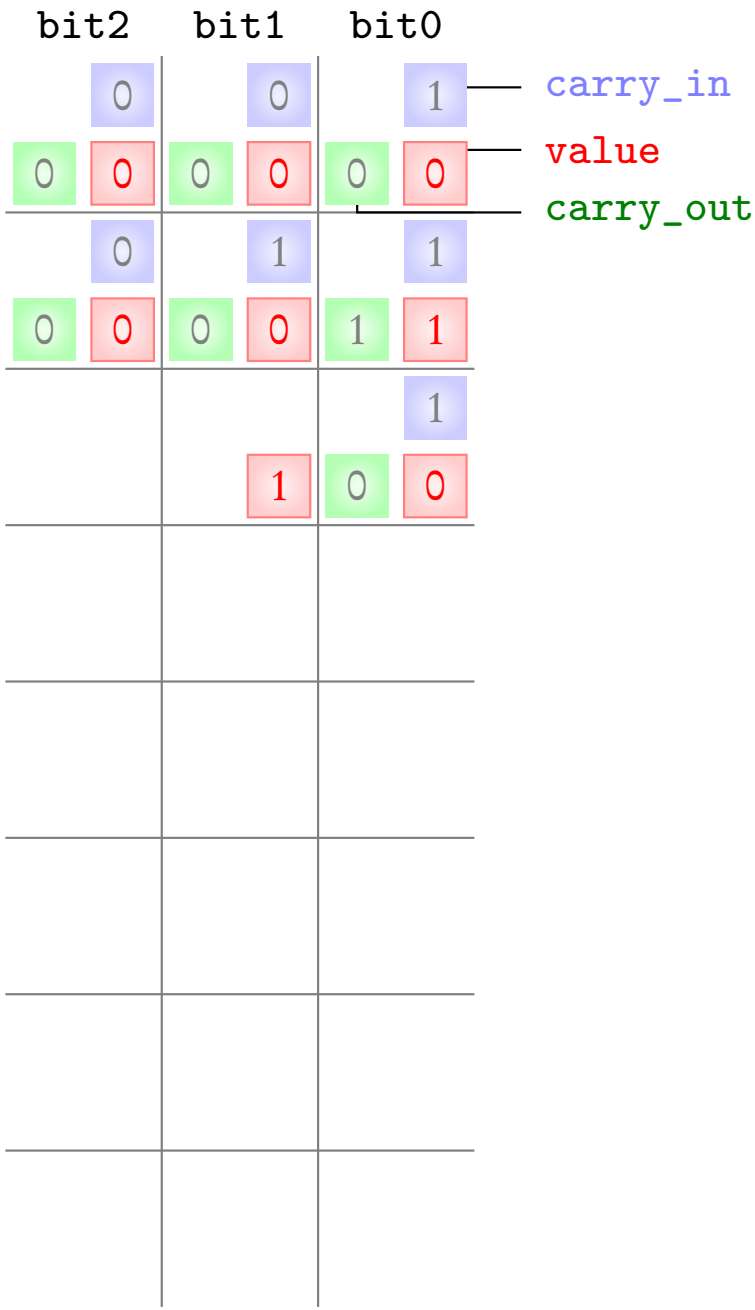

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


MODULE main
VAR
    bit0:counter_cell(TRUE);
    bit1:counter_cell(bit0.carry_out);
    bit2:counter_cell(bit1.carry_out);
```



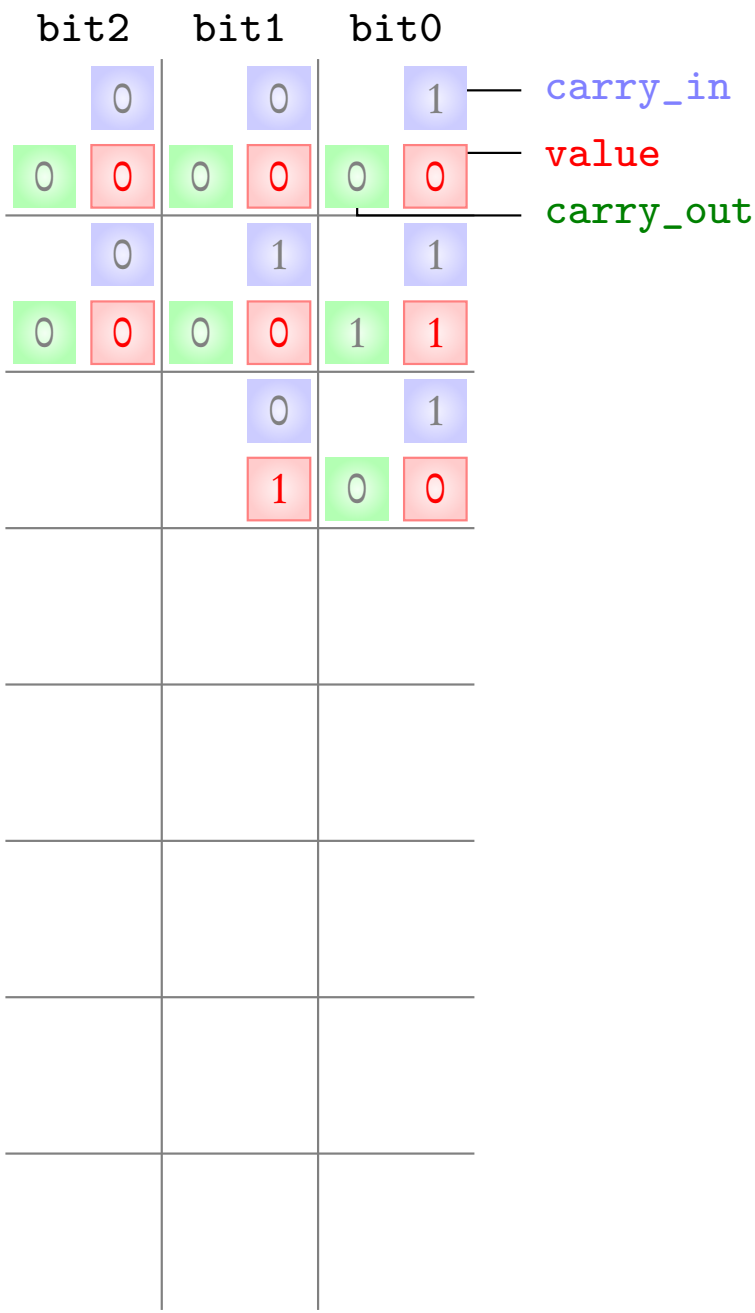

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


MODULE main
VAR
    bit0:counter_cell(TRUE);
    bit1:counter_cell(bit0.carry_out);
    bit2:counter_cell(bit1.carry_out);
```



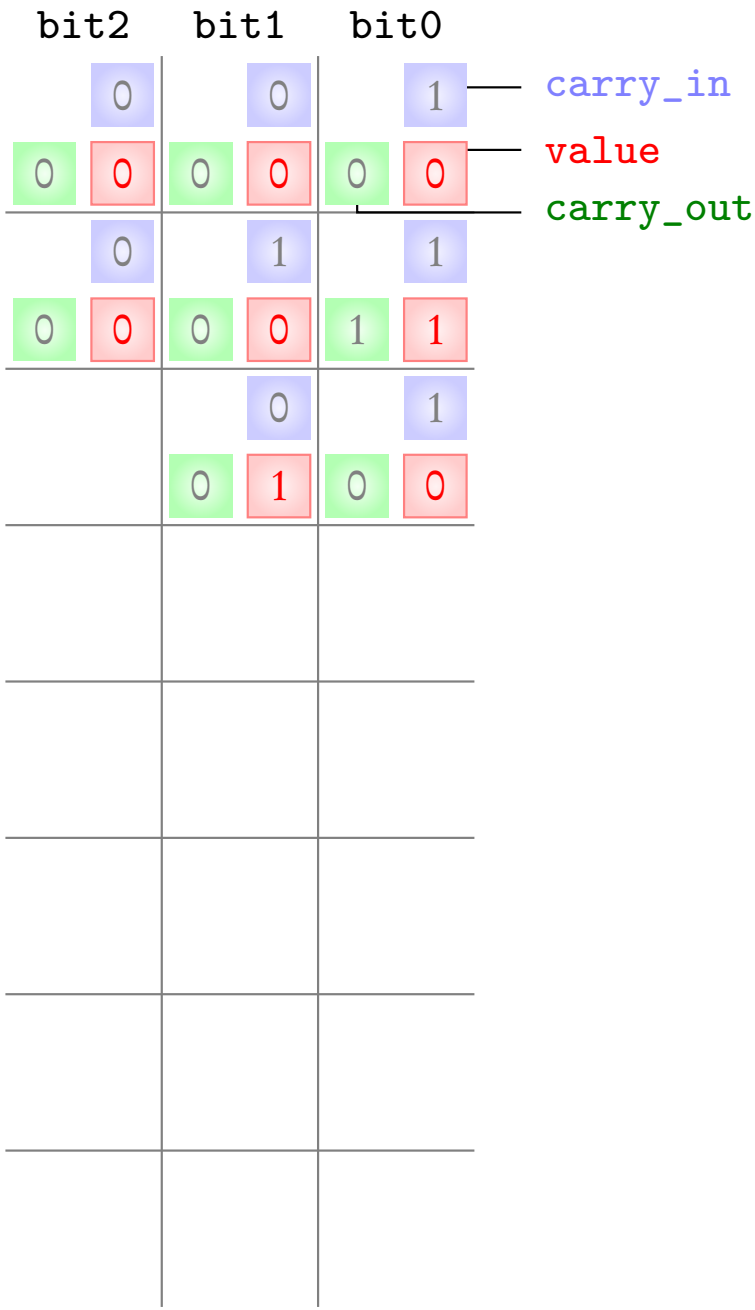

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


MODULE main
VAR
    bit0:counter_cell(TRUE);
    bit1:counter_cell(bit0.carry_out);
    bit2:counter_cell(bit1.carry_out);
```



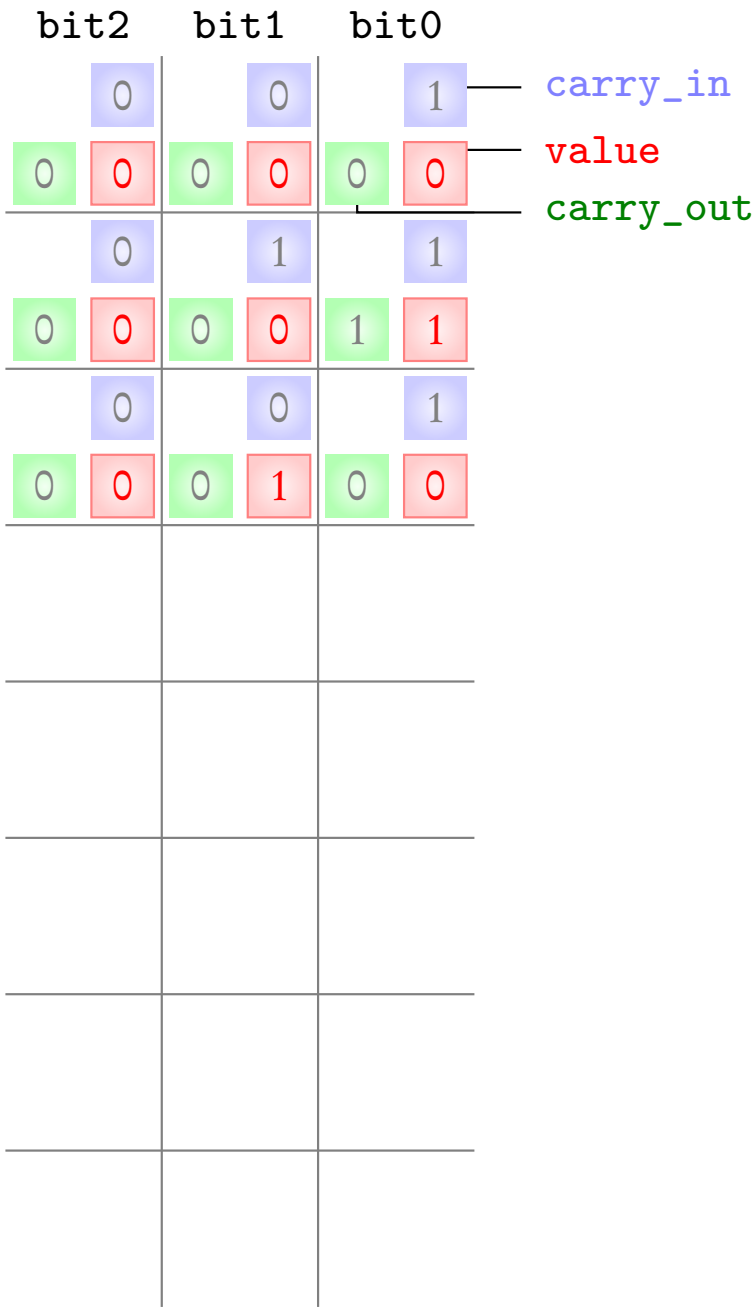

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


MODULE main
VAR
    bit0:counter_cell(TRUE);
    bit1:counter_cell(bit0.carry_out);
    bit2:counter_cell(bit1.carry_out);
```



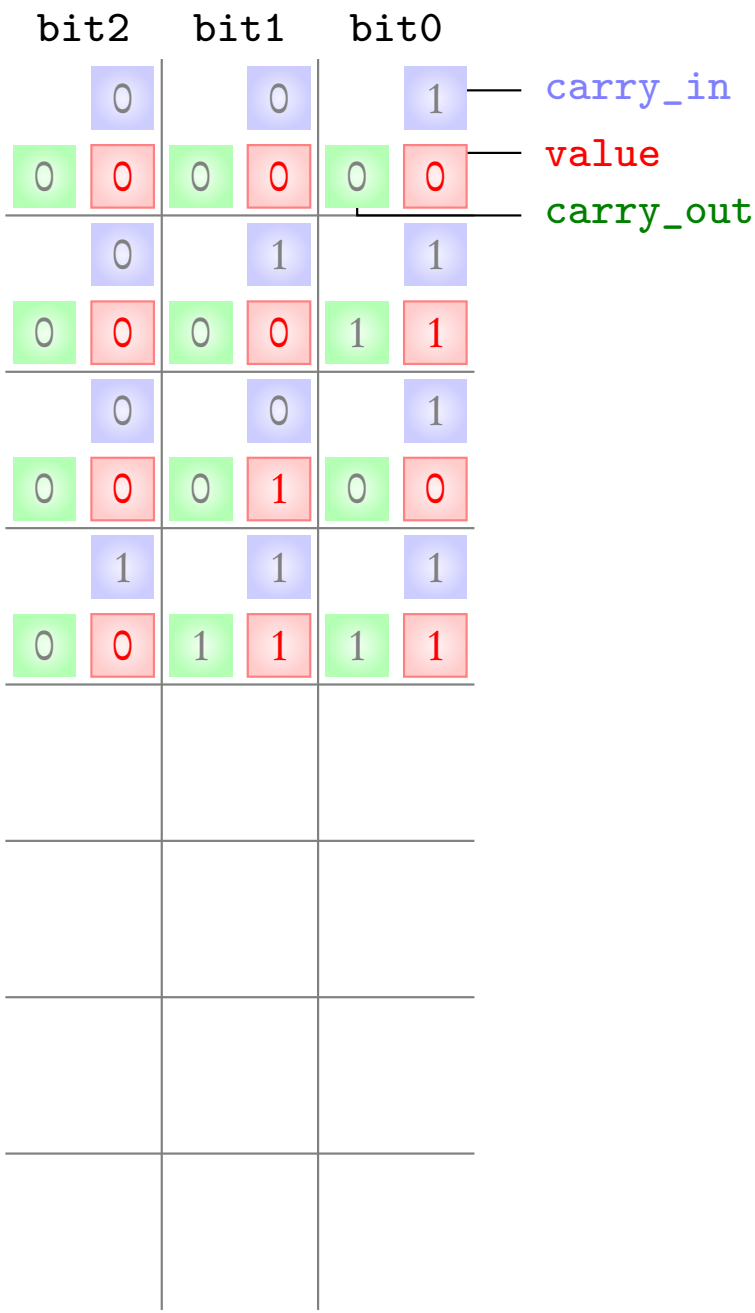

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


MODULE main
VAR
    bit0:counter_cell(TRUE);
    bit1:counter_cell(bit0.carry_out);
    bit2:counter_cell(bit1.carry_out);
```



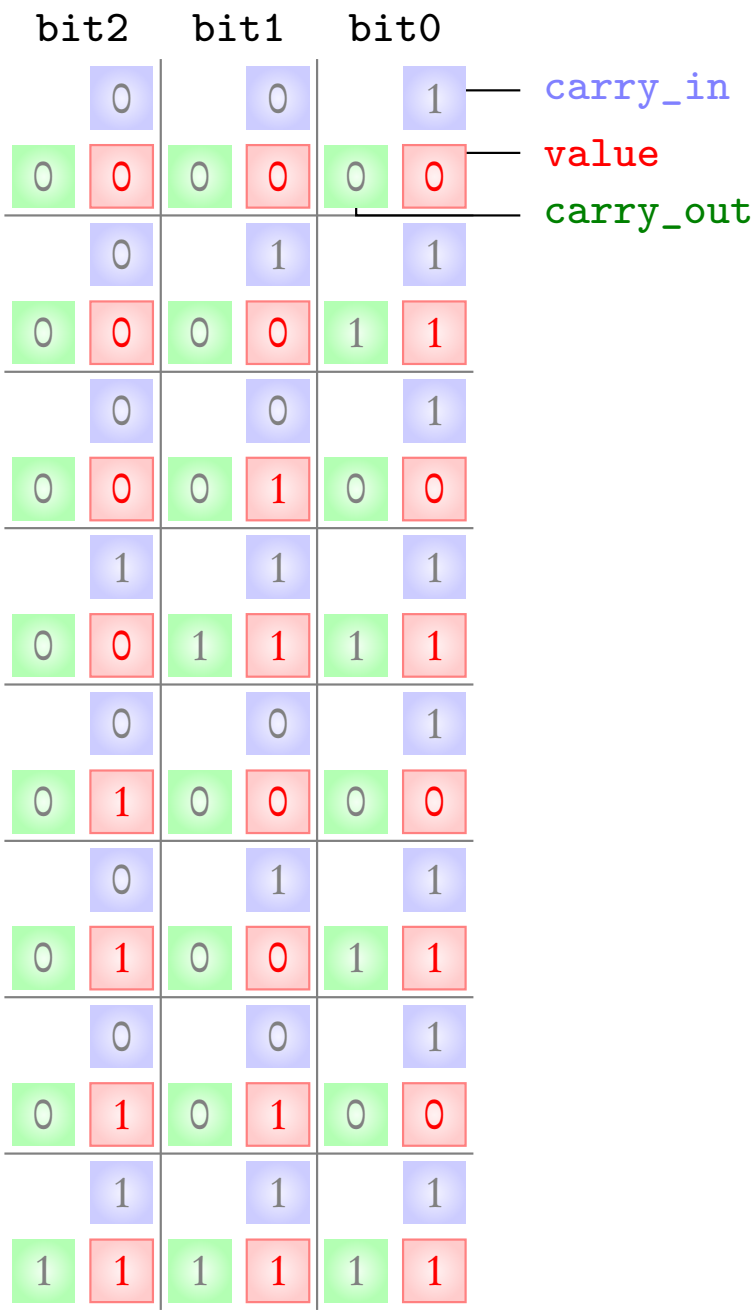

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


MODULE main
VAR
    bit0:counter_cell(TRUE);
    bit1:counter_cell(bit0.carry_out);
    bit2:counter_cell(bit1.carry_out);
```



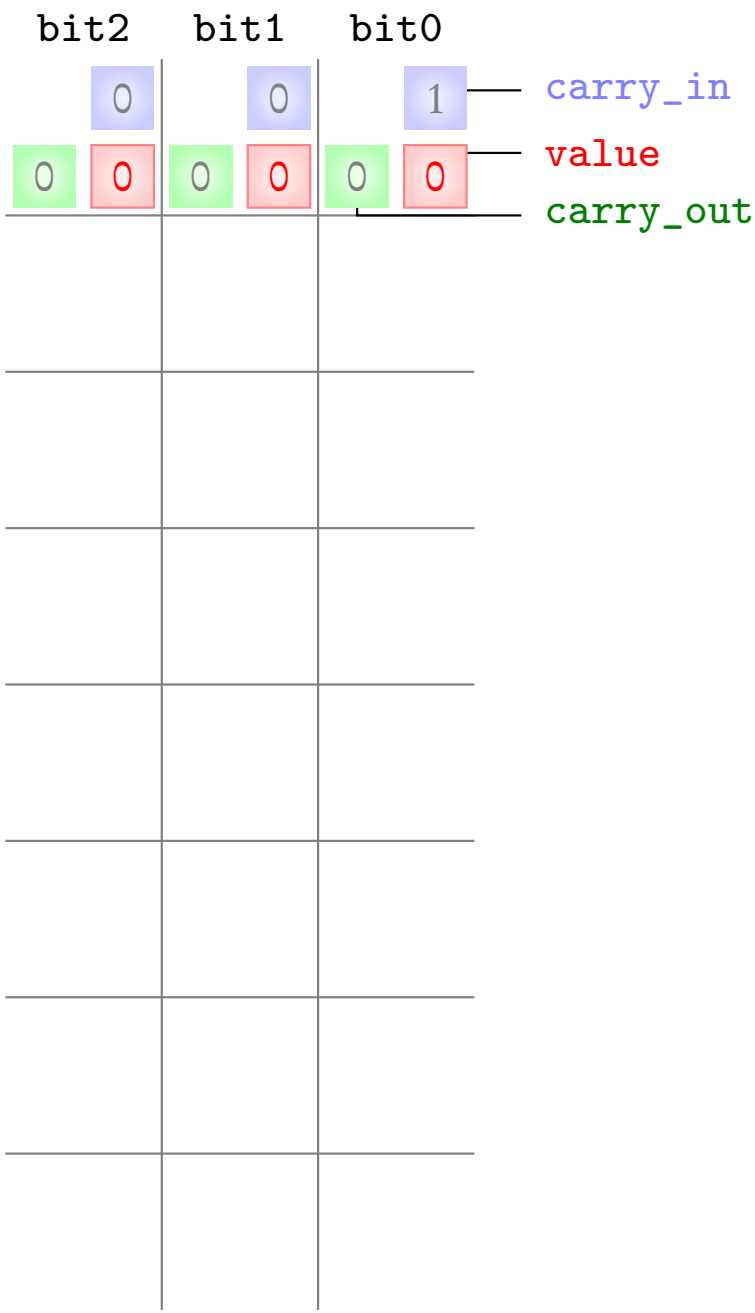

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


MODULE main
VAR
    bit0:counter_cell(TRUE);
    bit1:counter_cell(bit0.carry_out);
    bit2:counter_cell(bit1.carry_out);
```

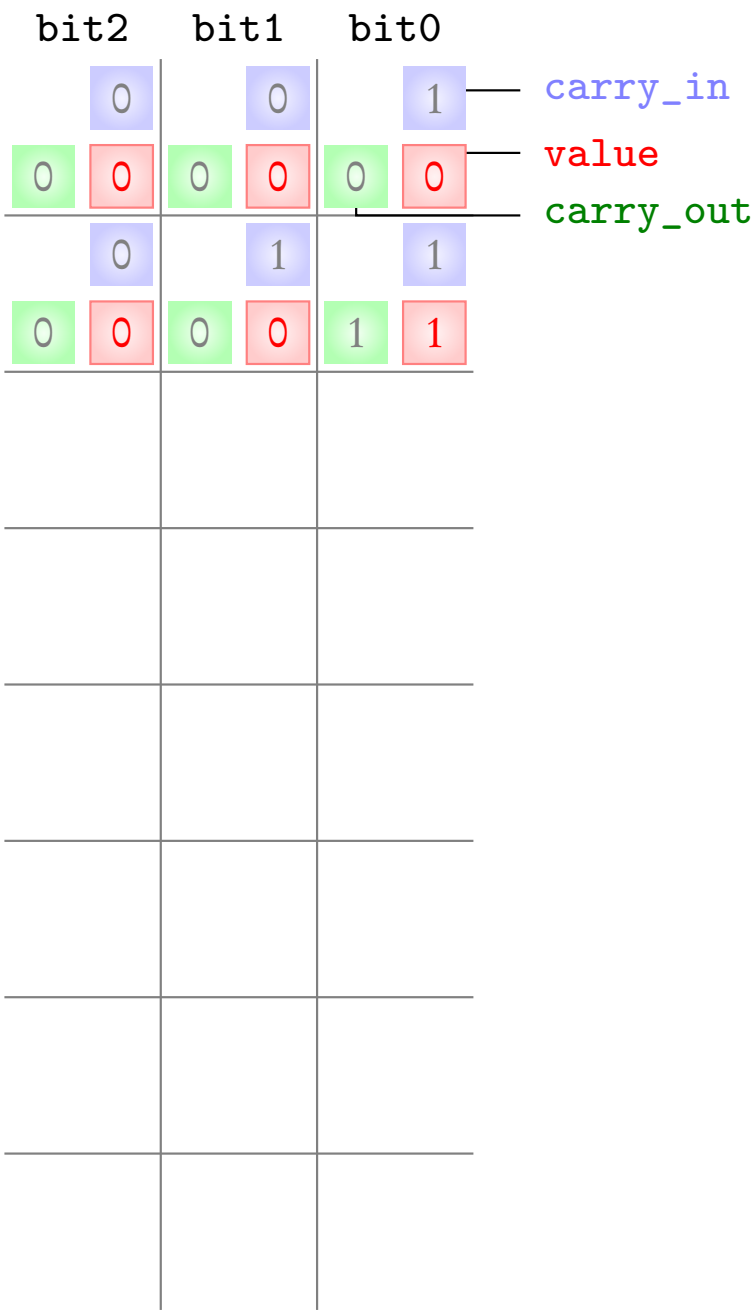


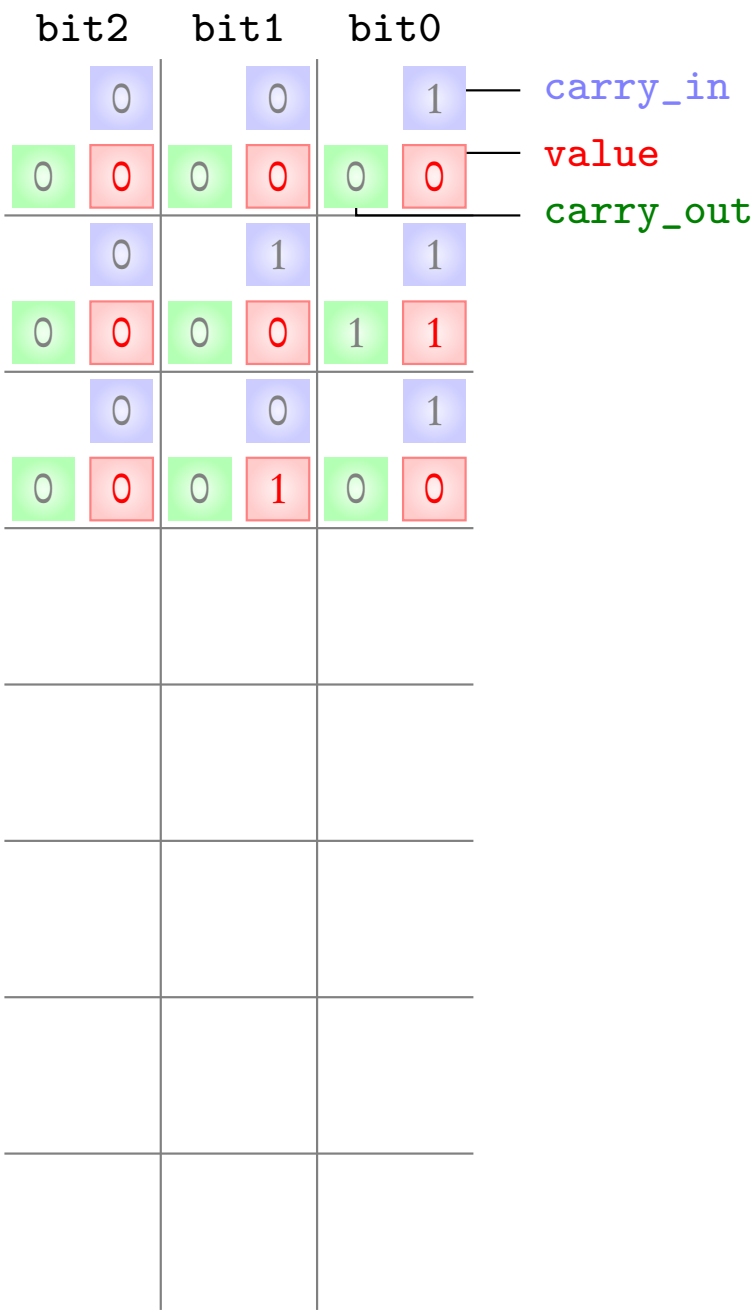

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


MODULE main
VAR
    bit0:counter_cell(TRUE);
    bit1:counter_cell(bit0.carry_out);
    bit2:counter_cell(bit1.carry_out);
```

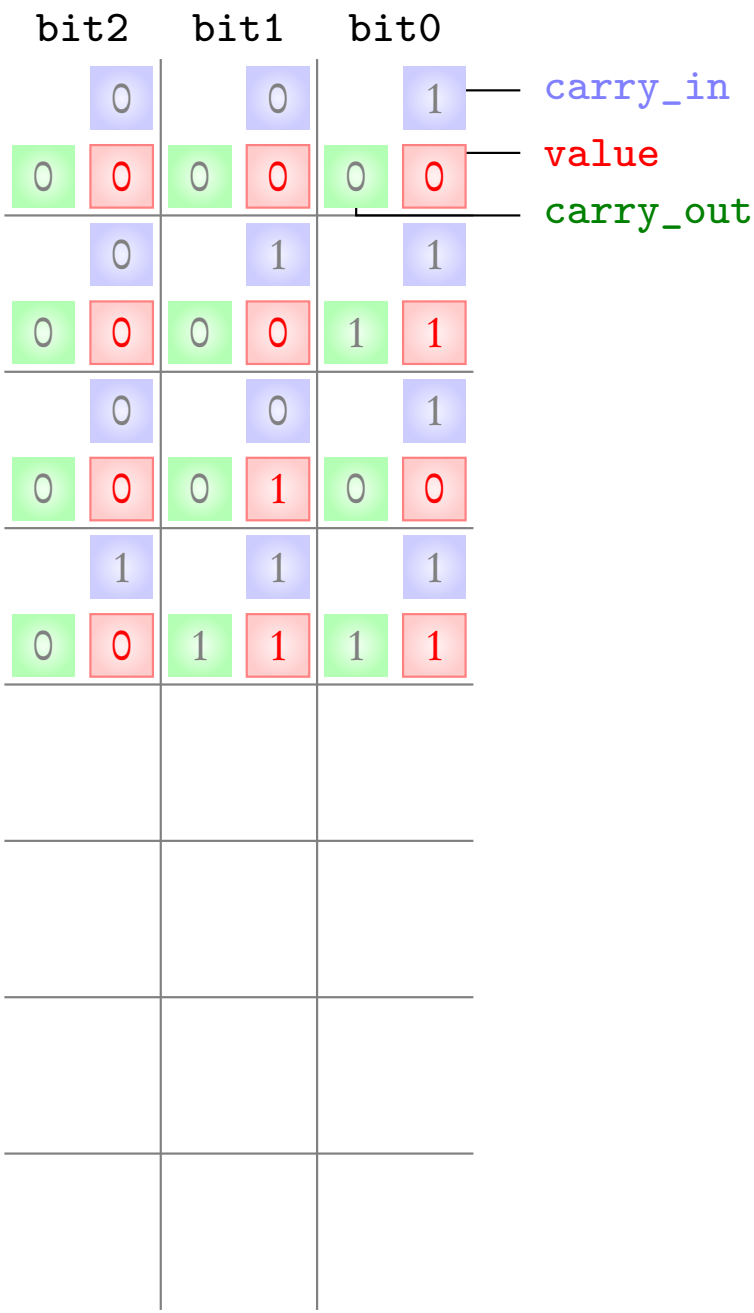


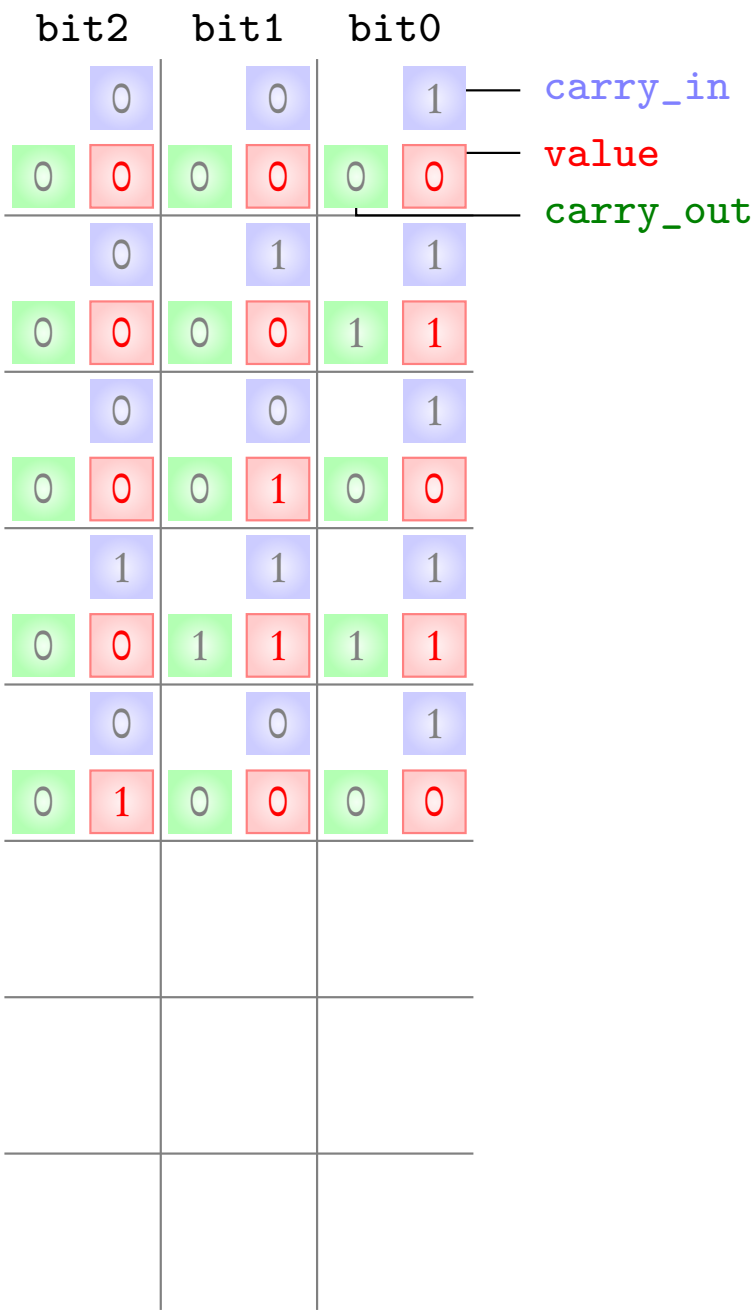

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


MODULE main
VAR
    bit0:counter_cell(TRUE);
    bit1:counter_cell(bit0.carry_out);
    bit2:counter_cell(bit1.carry_out);
```

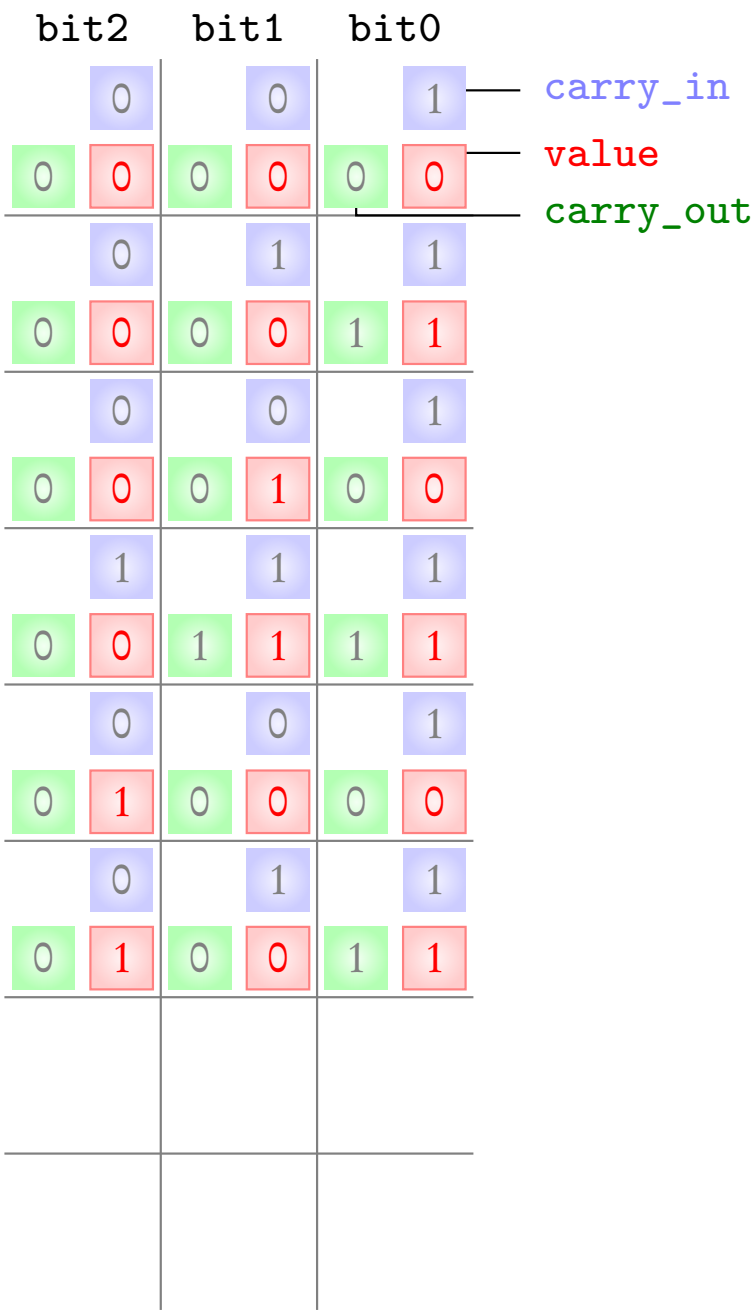

bit2
bit1
bit0
0
0
1
carry_in
0
0
0
0
0
0
value
carry_out

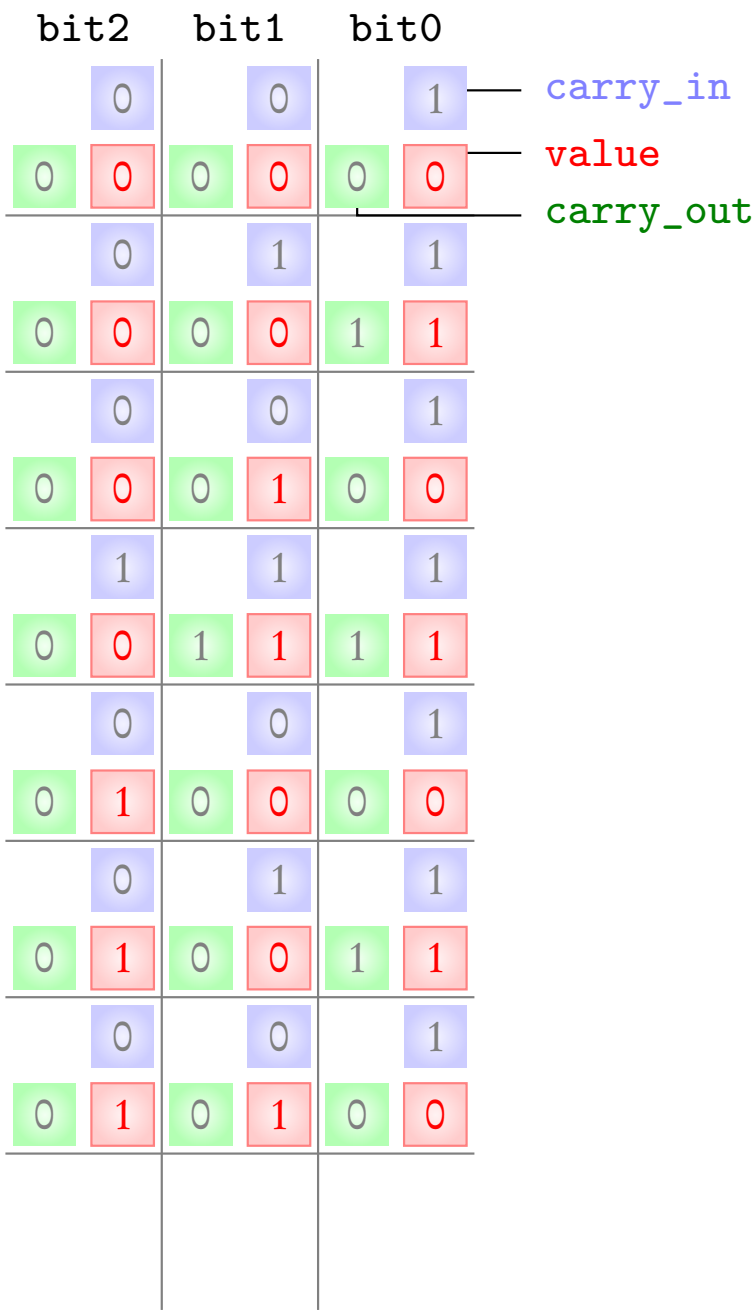

bit2
bit1
bit0
0
0
1
carry_in
0
0
0
0
0
0
value
carry_out
0
1
1
0
0
0
0
1
1

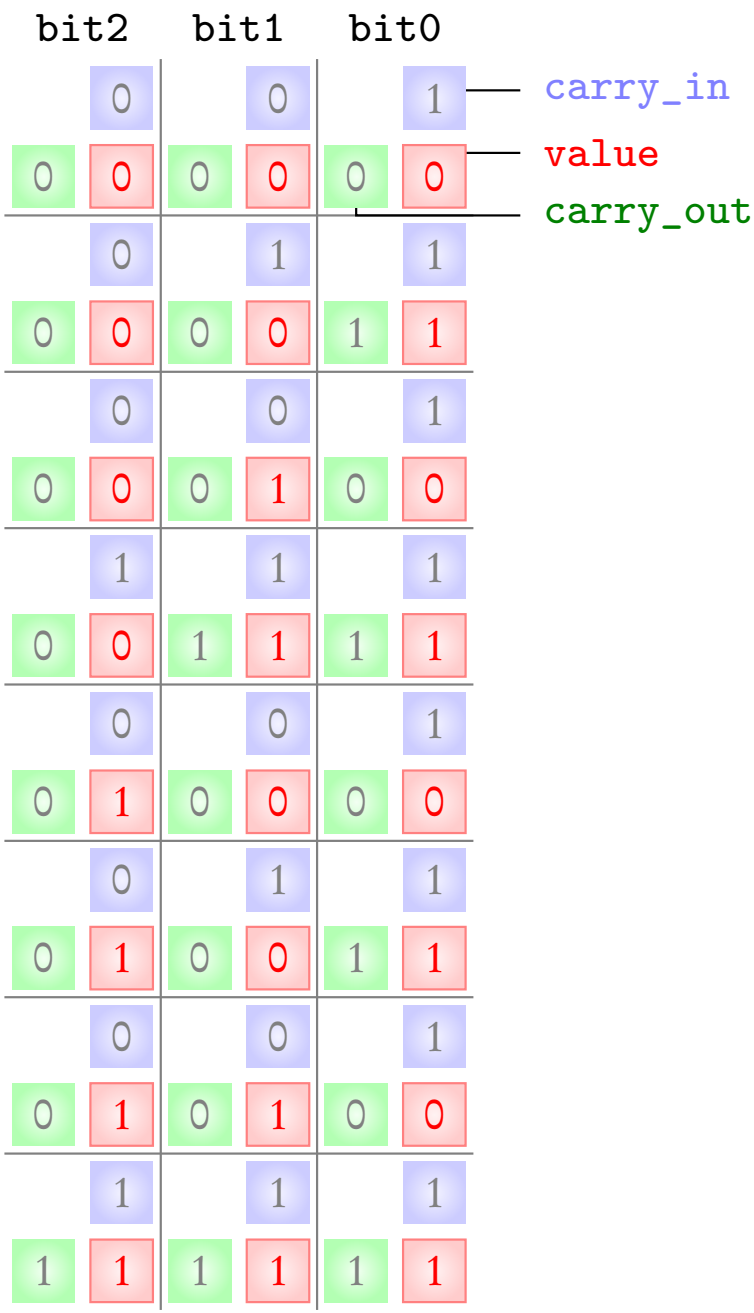

bit2
bit1
bit0
carry_in
value
carry_out

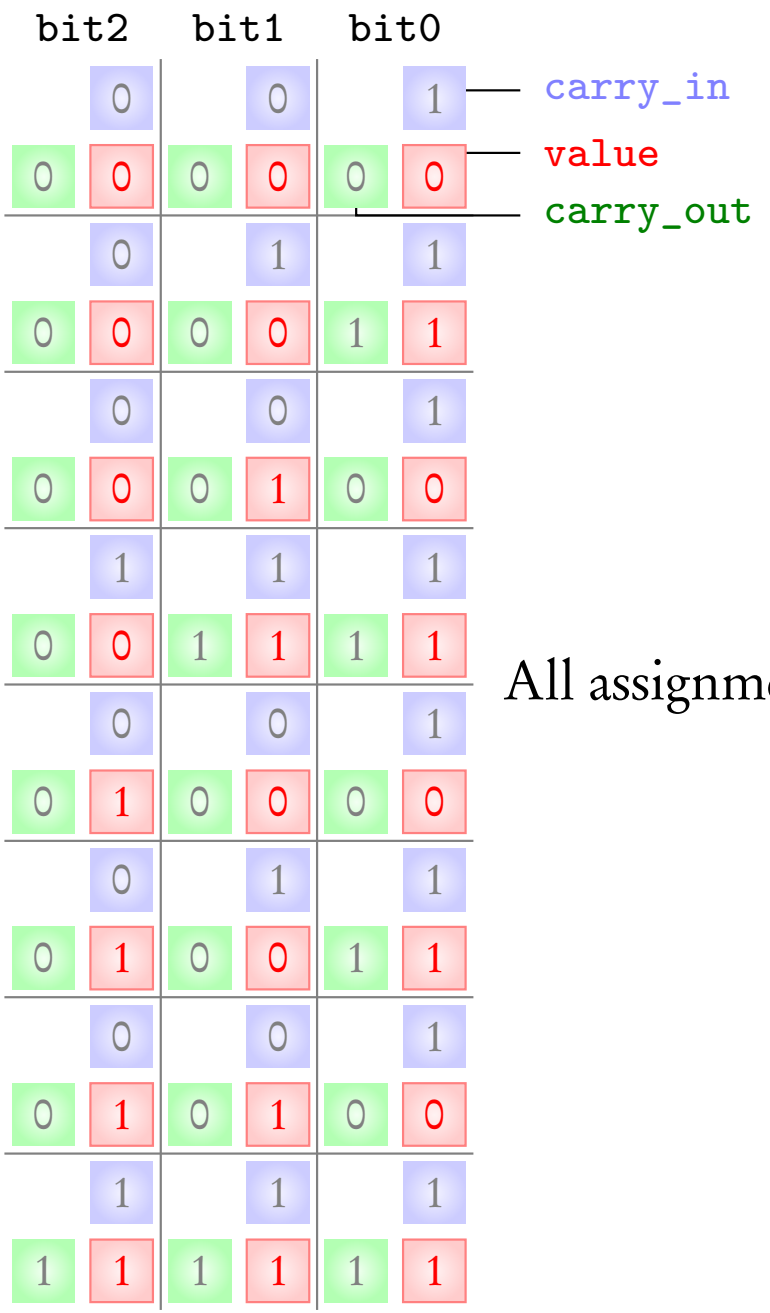

bit2
bit1
bit0
carry_in
value
carry_out


bit2
bit1
bit0
carry_in
value
carry_out


bit2
bit1
bit0
carry_in
value
carry_out


bit2
bit1
bit0
carry_in
value
carry_out


bit2
bit1
bit0
carry_in
value
carry_out



Synchronous composition

All assignments to all `MODULES` occur simultaneously

(more about this later)

Summary

Hierarchical designs

Use of `MODULE`

Synchronous composition

Take-away

- Computation as a **sequence of states**
- Specify **initial values** for variables
- Specify **next-state relation**: how the variables change given the current valuation
- **NuSMV models** of simple systems
- Requirements **G** and **F**