Unit-2: Model-checker NuSMV

B. Srivathsan

Chennai Mathematical Institute

NPTEL-course

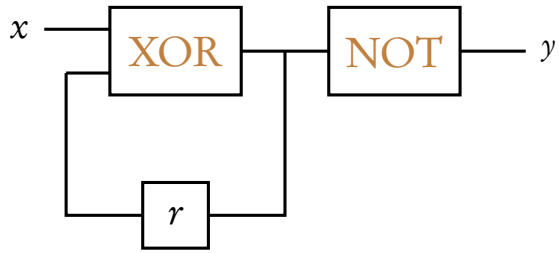
July - November 2015

Module 3:

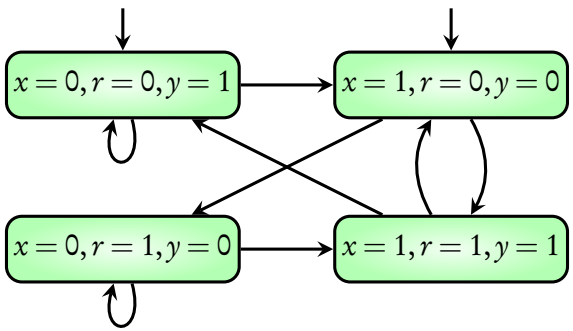
Hardware verification using NuSMV



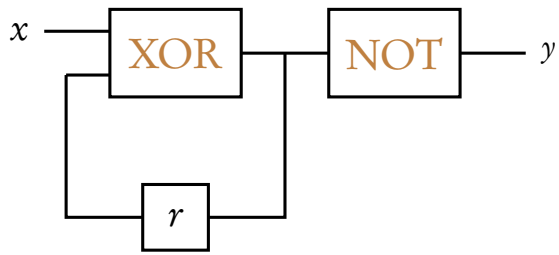
$$y = \text{NOT}\ (\ \text{XOR}\ (x, r)\ )$$
$$r_{next} = \text{XOR}\ (x, r)$$



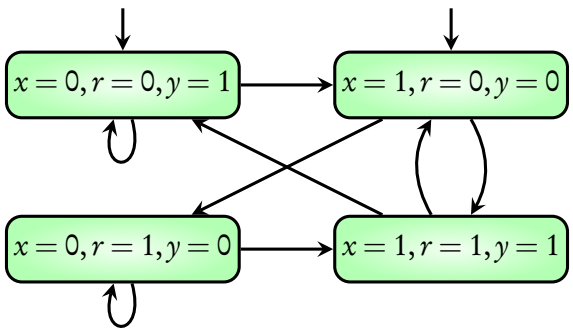
register



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

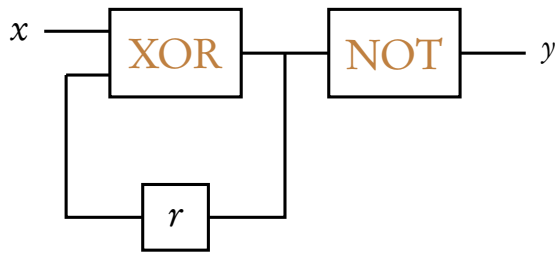


register

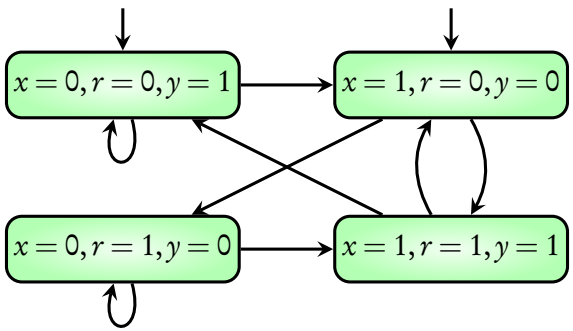



```
MODULE main
VAR
   x:  boolean;
   r:  boolean;
```

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



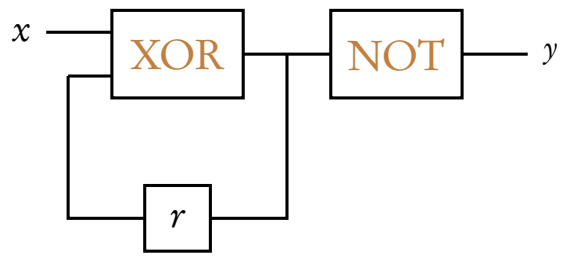
register



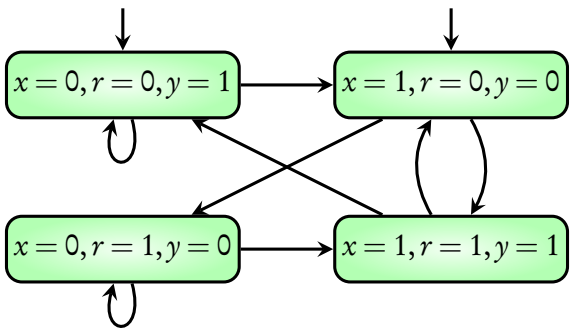

```
MODULE main
VAR
   x:  boolean;
   r:  boolean;
```

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

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

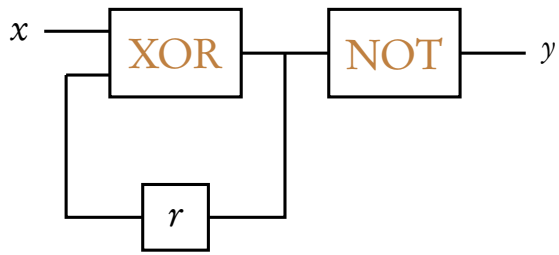


register

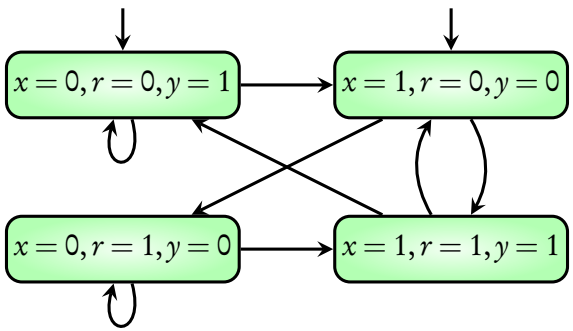



```
MODULE main
VAR
   x:  boolean;
   r:  boolean;

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;
```

Simple circuit

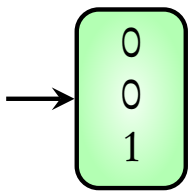
Use of `DEFINE`

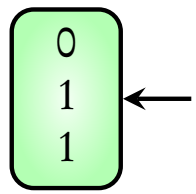

in_1
in_2
NAND
out

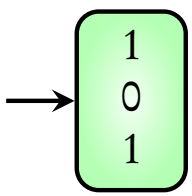

```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
DEFINE
   -- ZERO DELAY
   out := !(in1 & in2);
```

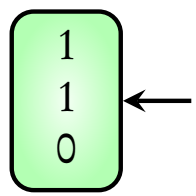
0 0 1		0 1 1
1 0 1		1 1 0

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

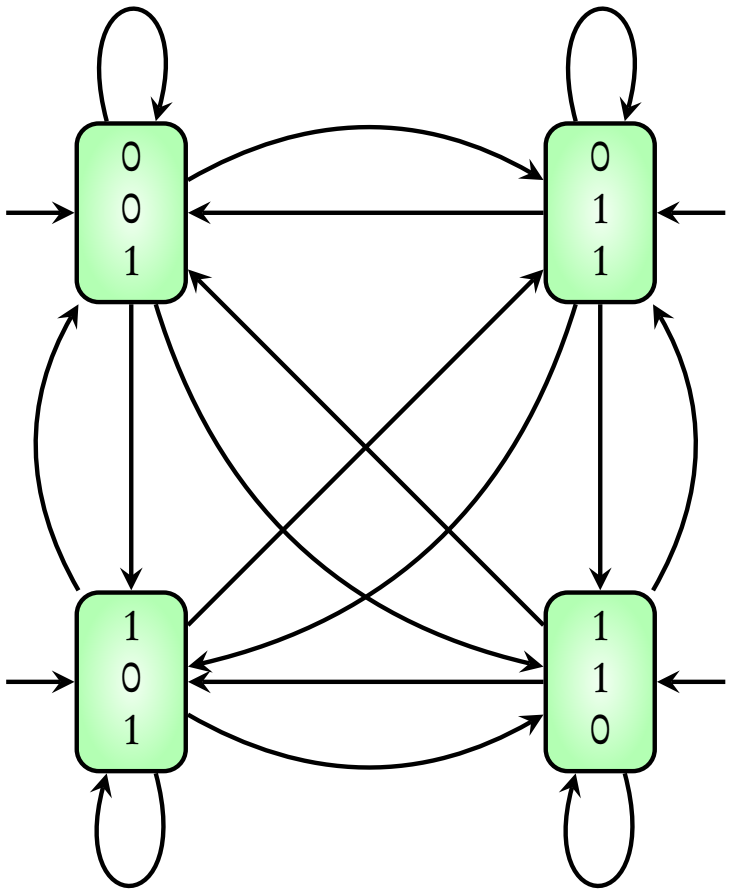






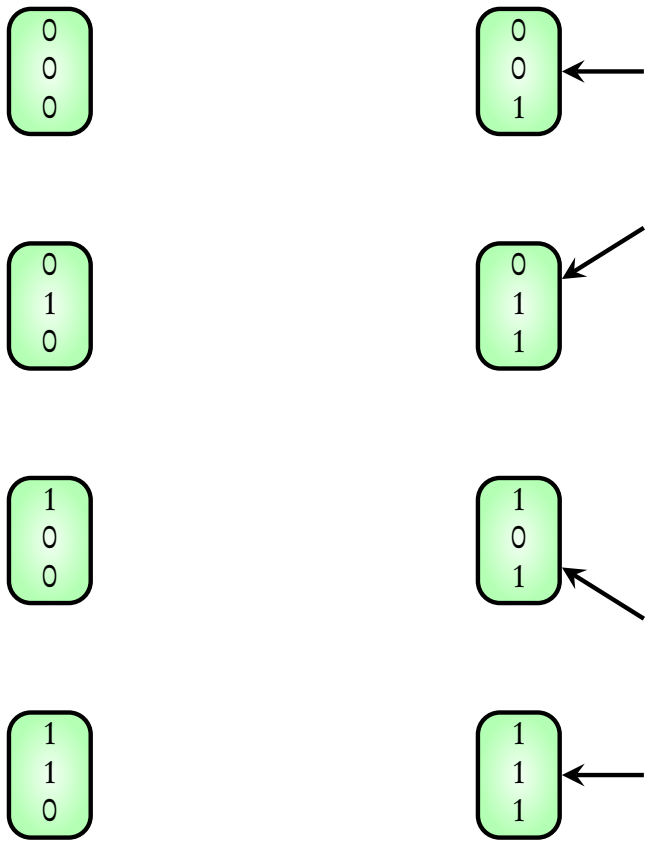



```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
DEFINE
   -- ZERO DELAY
   out := !(in1 & in2);
```

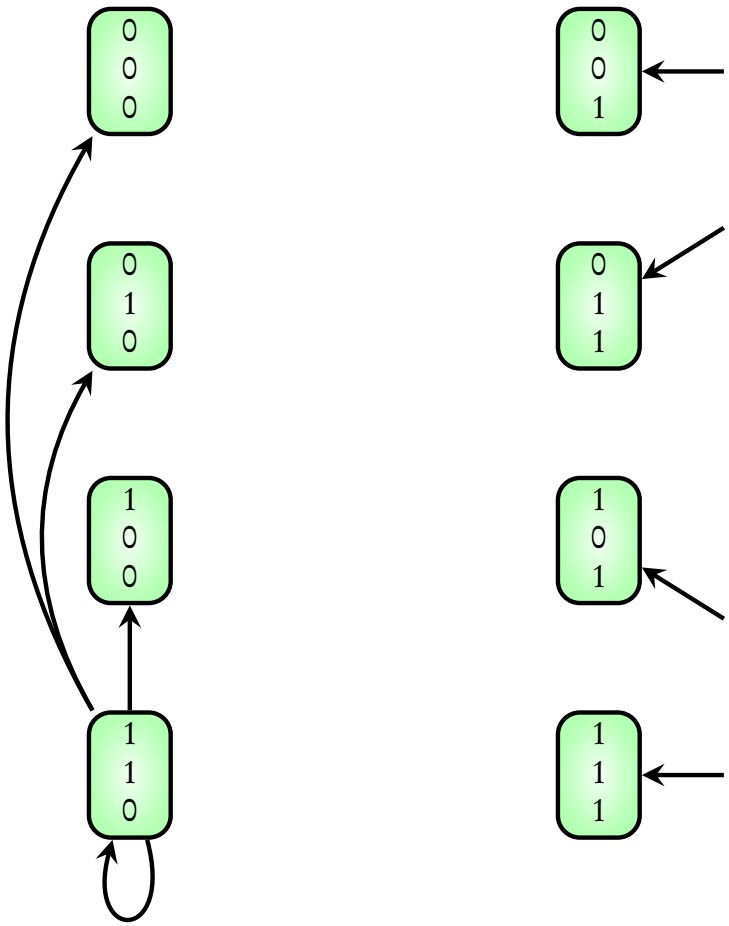



```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
DEFINE
   -- ZERO DELAY
   out := !(in1 & in2);
```

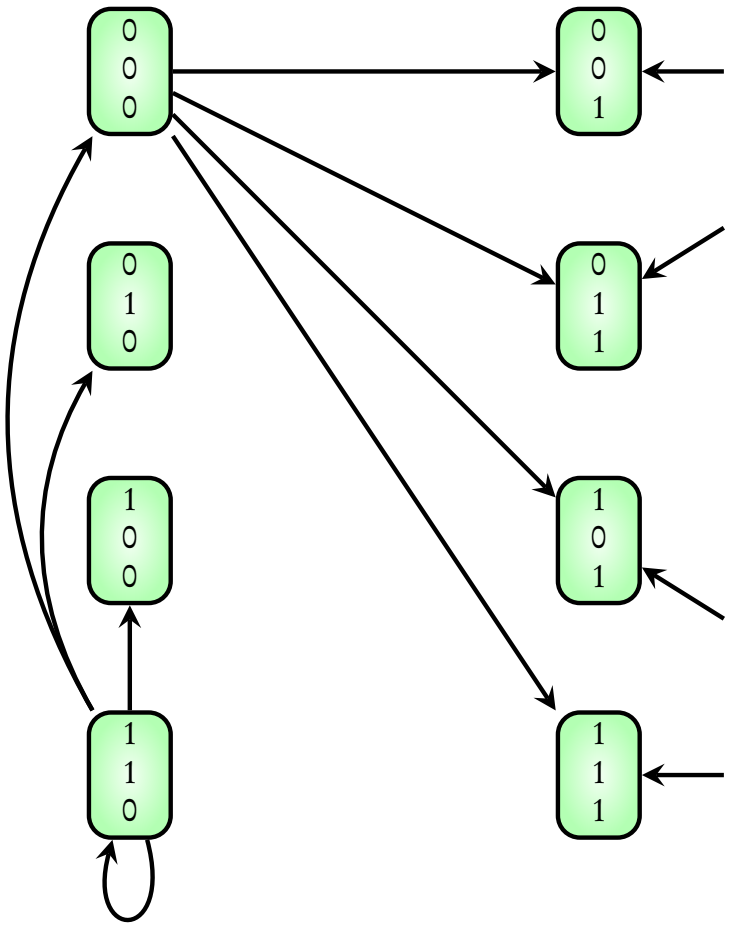

```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```



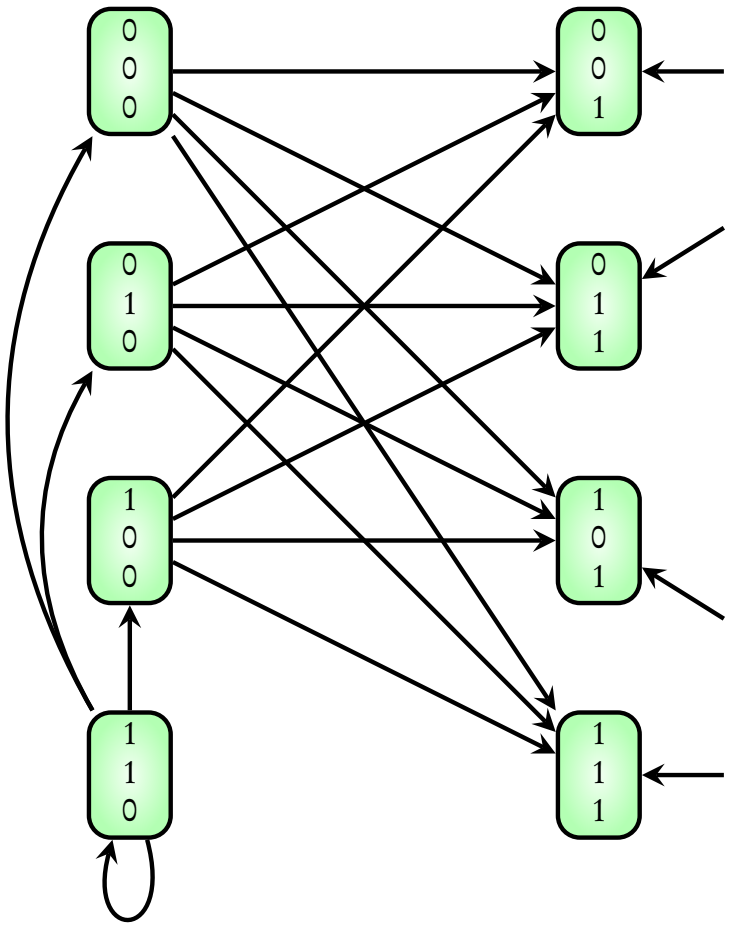

```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```



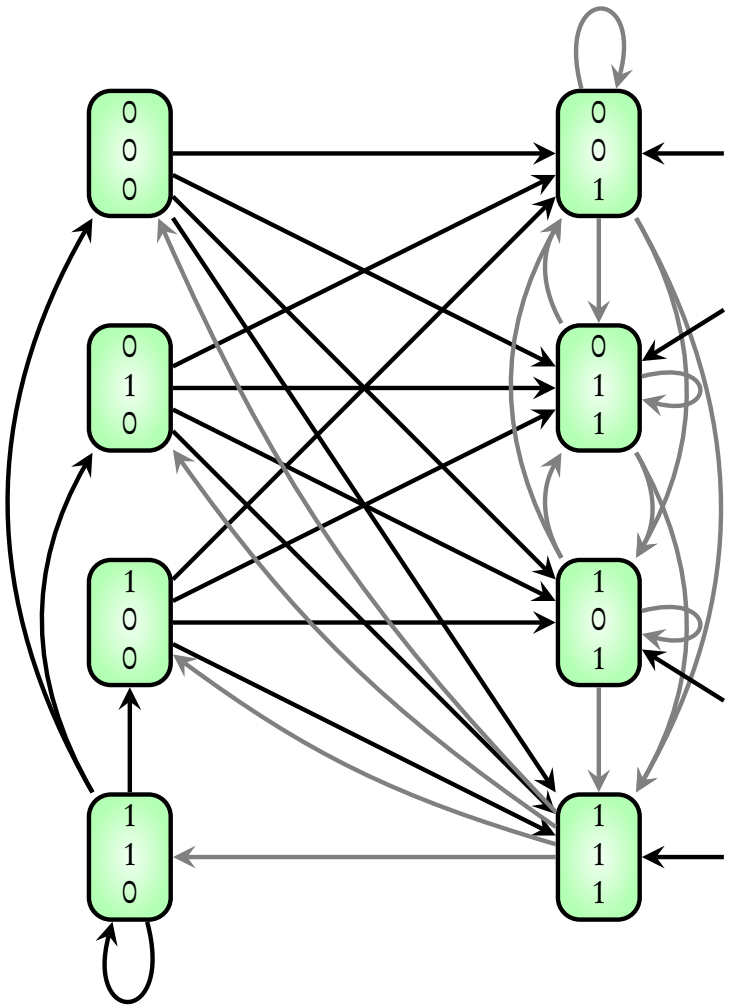

```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```



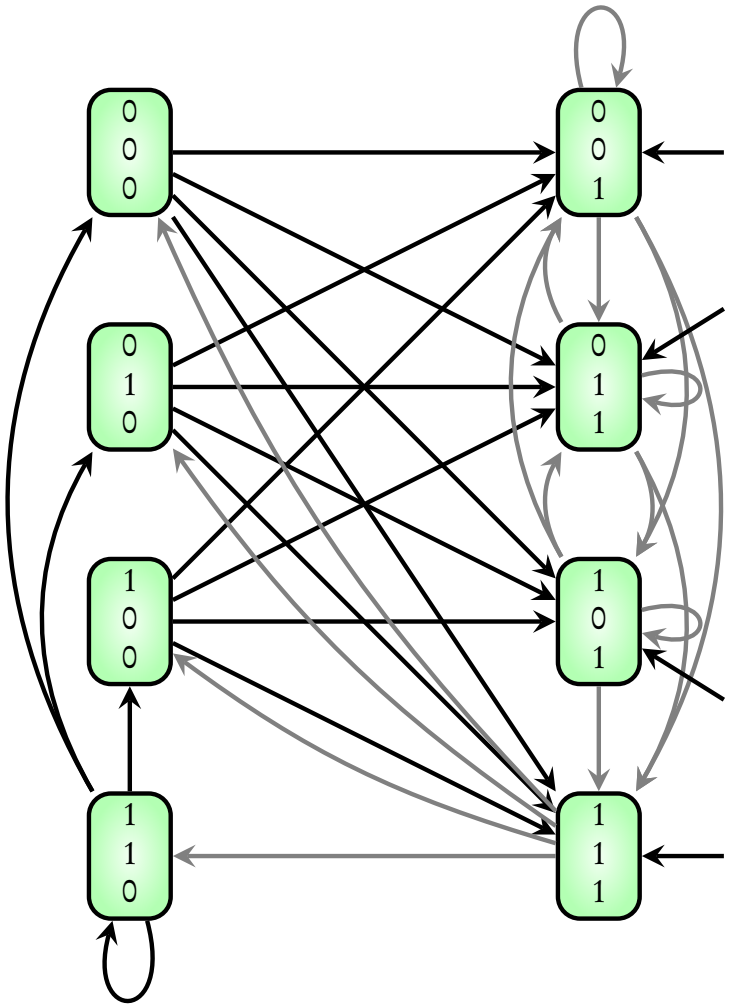

```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```



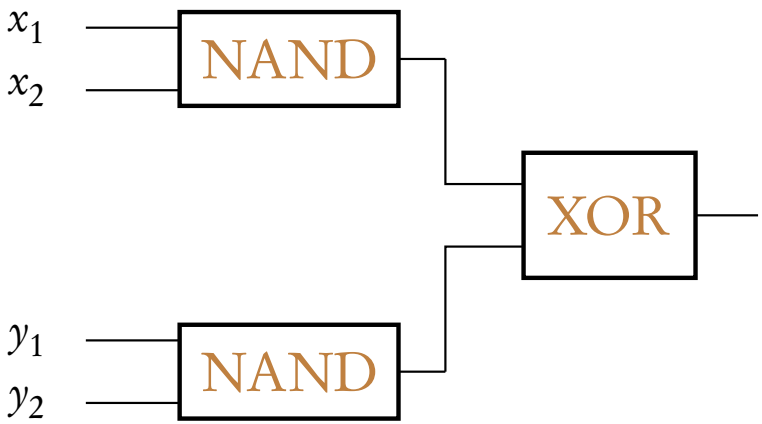

```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```



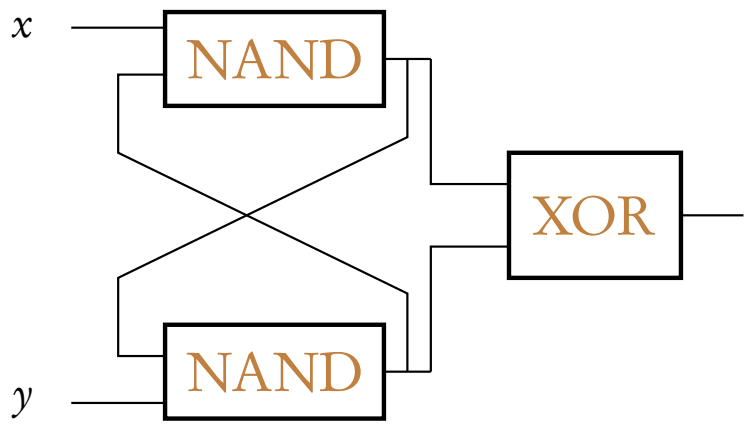

```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```




```
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);
```

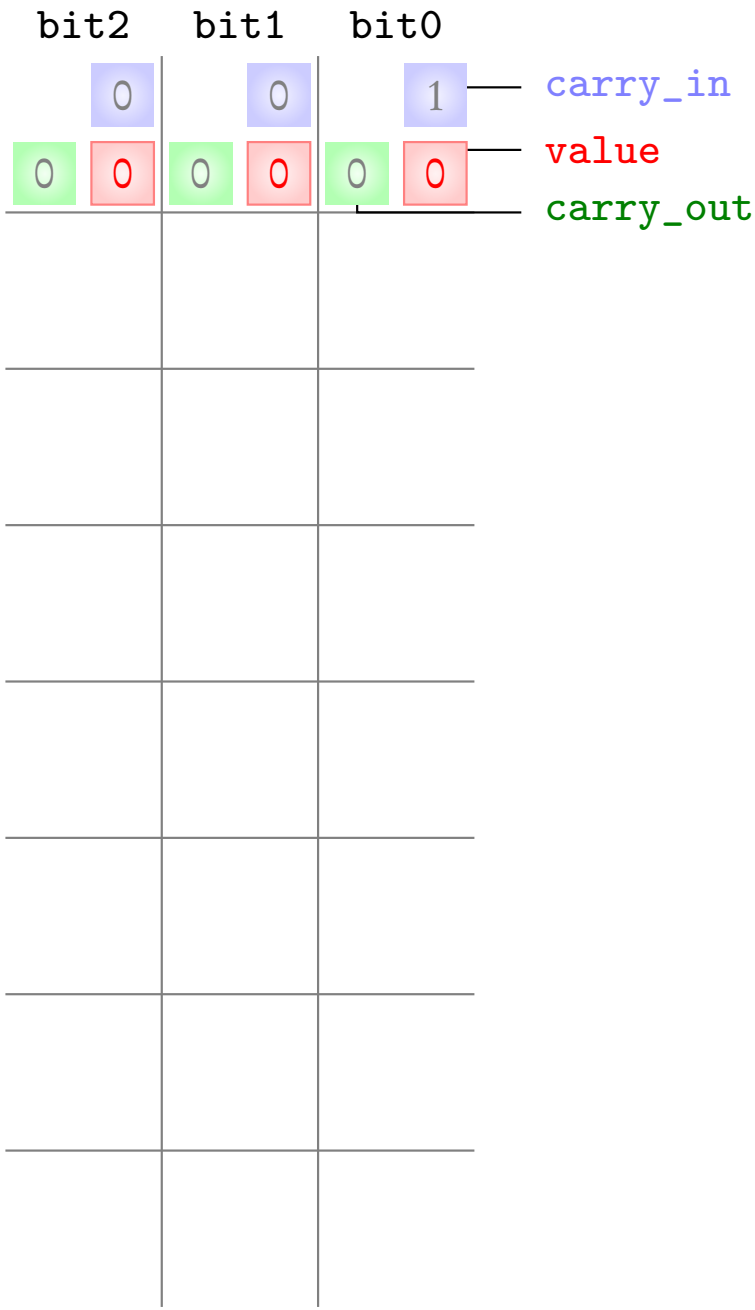
Simple circuit

Use of `DEFINE`

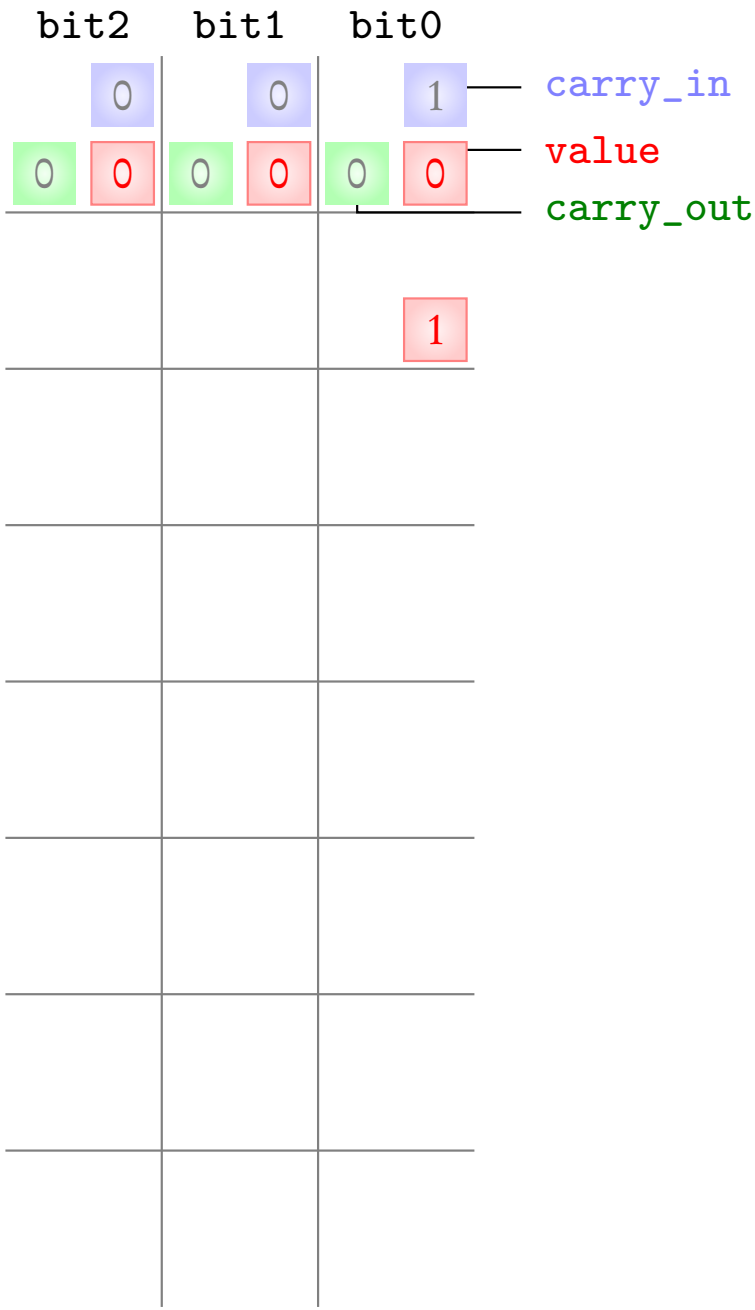
Hierarchical designs

Use of `MODULE`

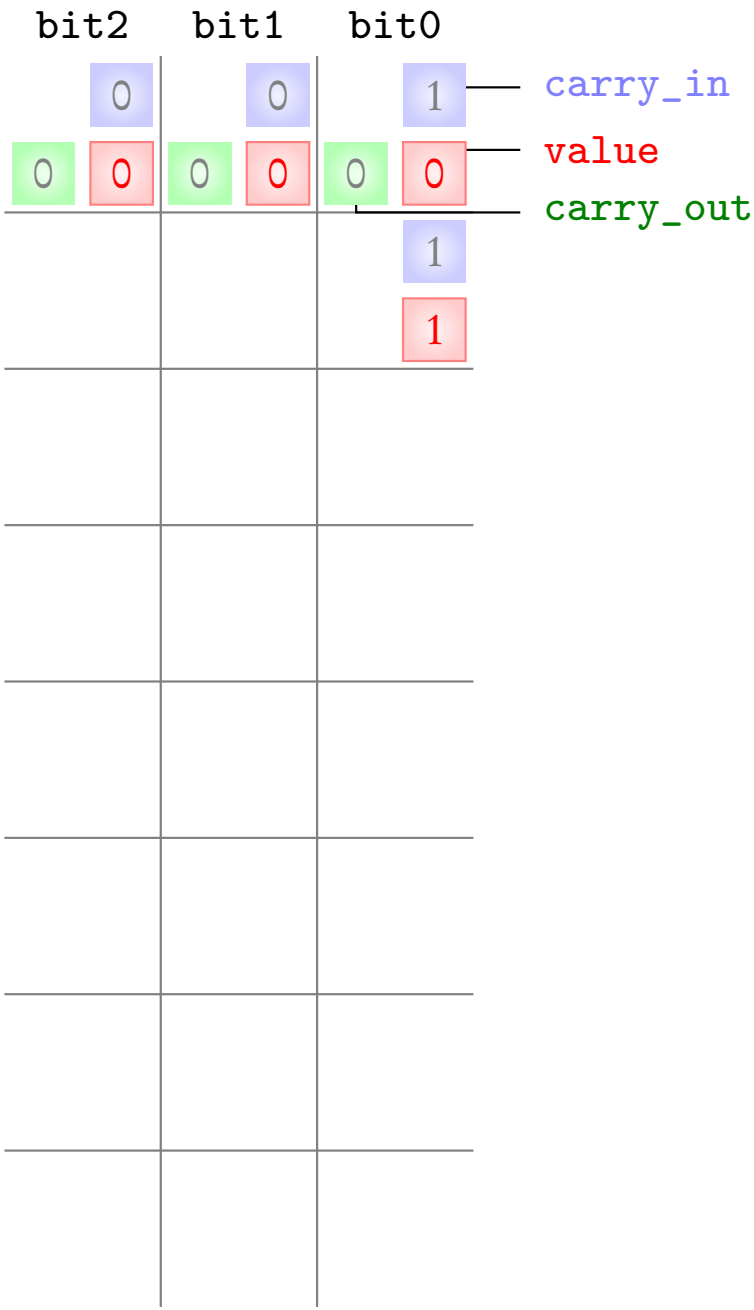
```
MODULE counter_cell(carry_in)
VAR
   value:boolean;
ASSIGN
   init(value):=FALSE;
   next(value):= value xor carry_in;
DEFINE
   carry_out := carry_in & value;

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



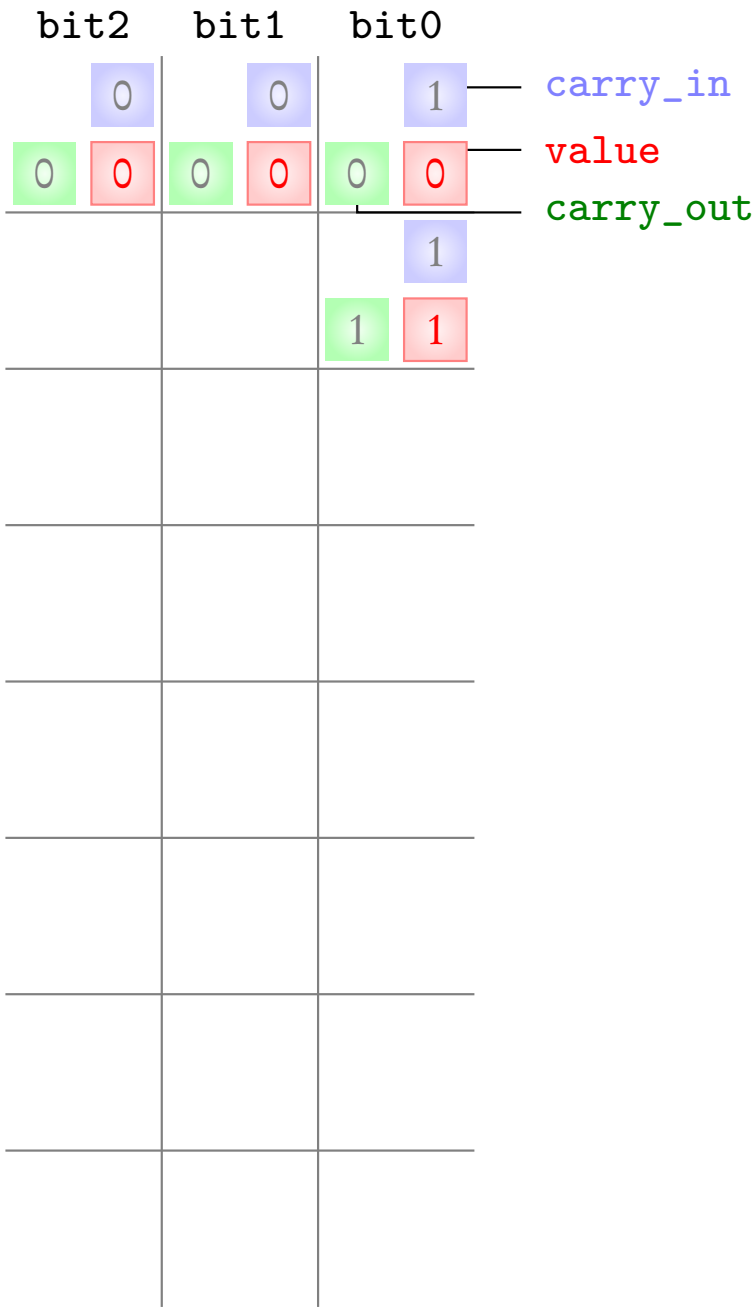

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



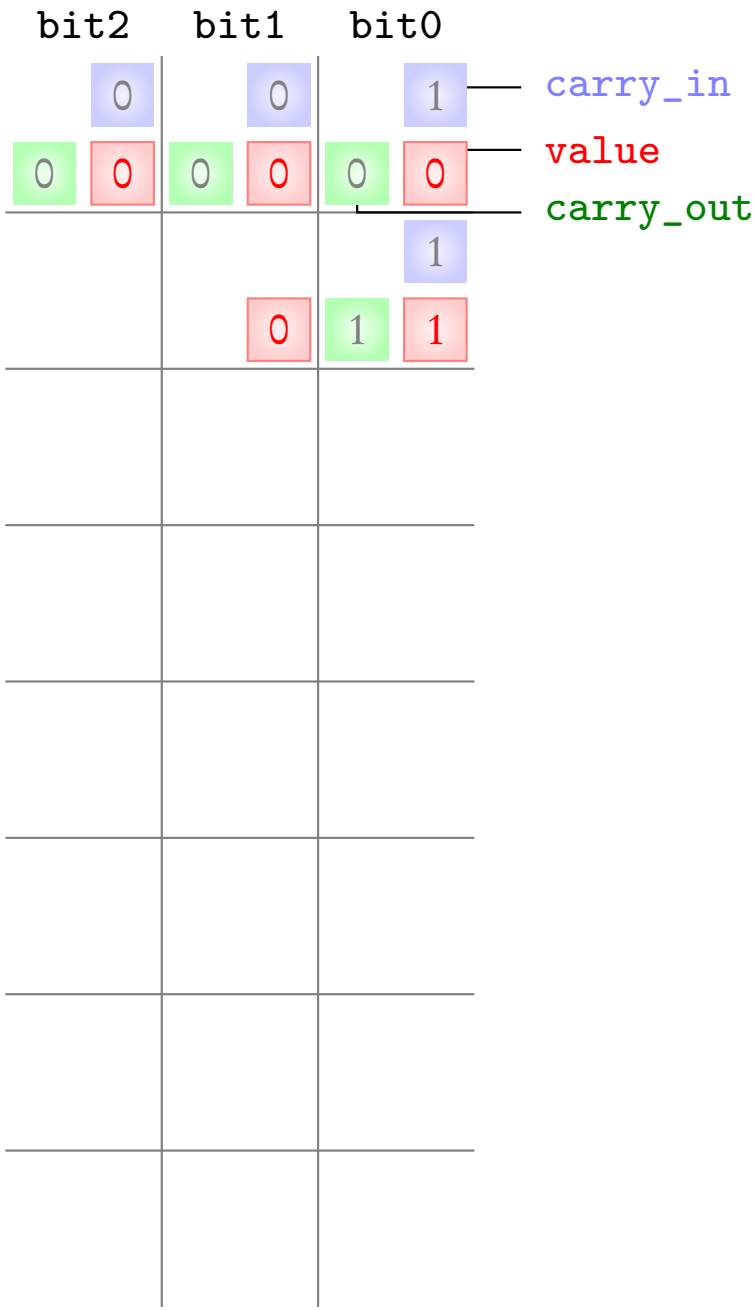

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



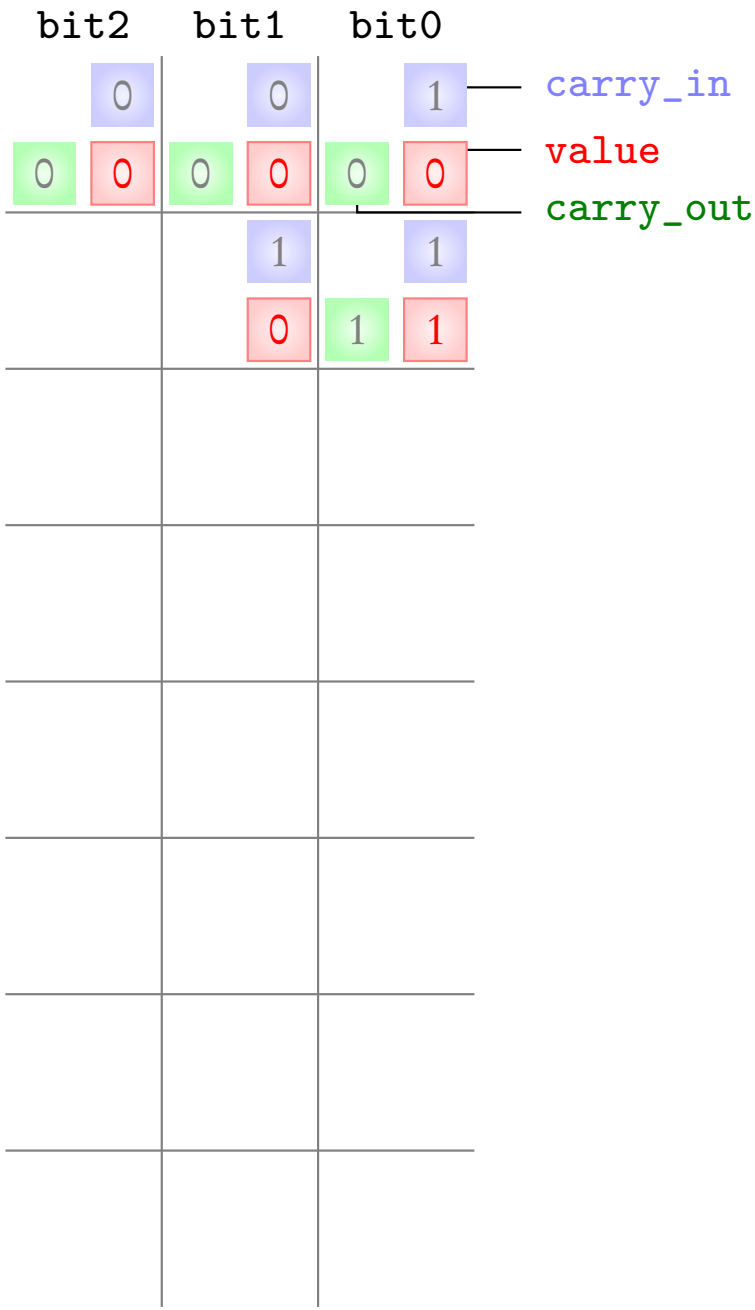

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



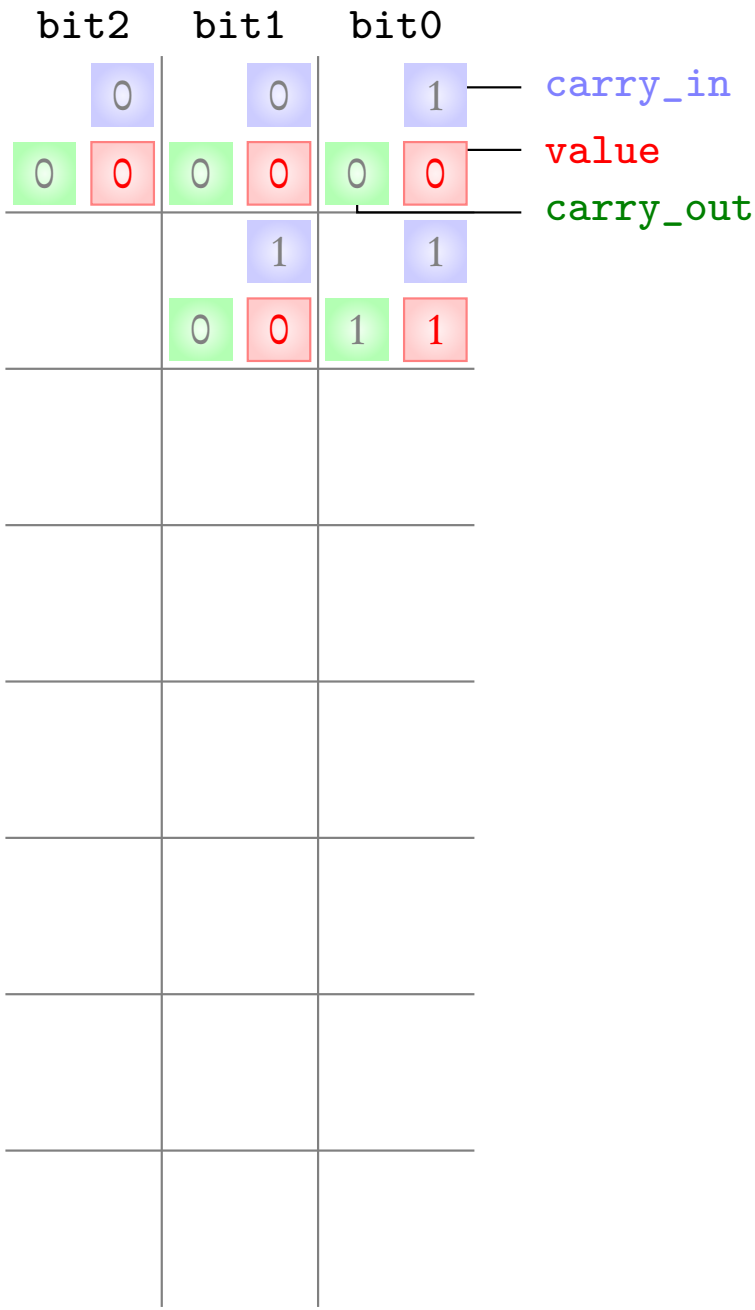

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



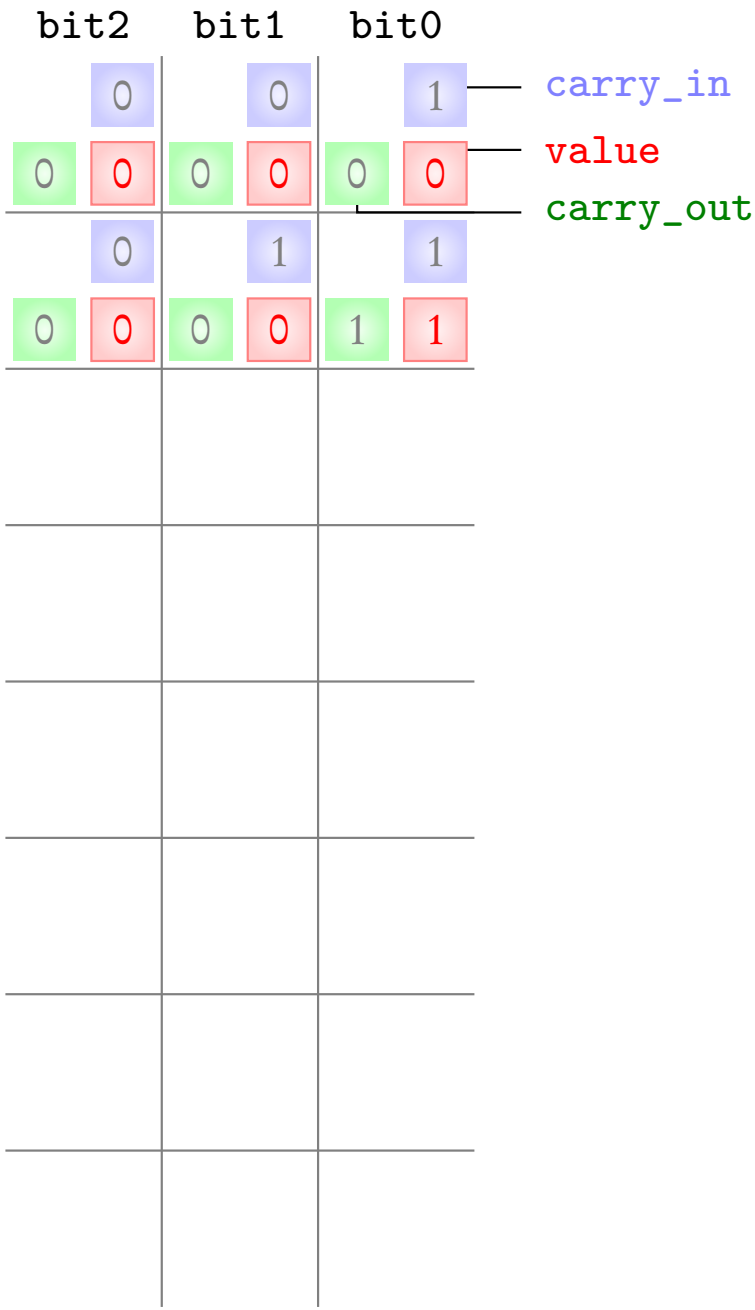

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



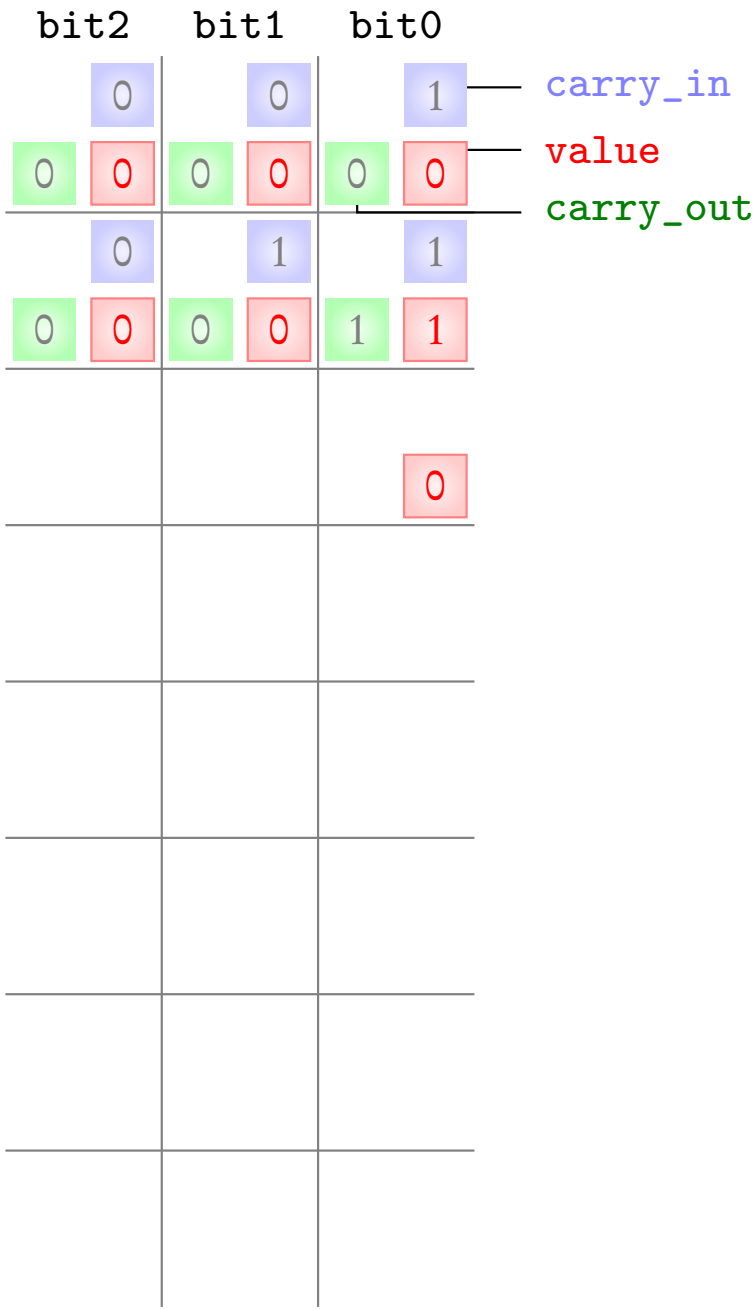

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



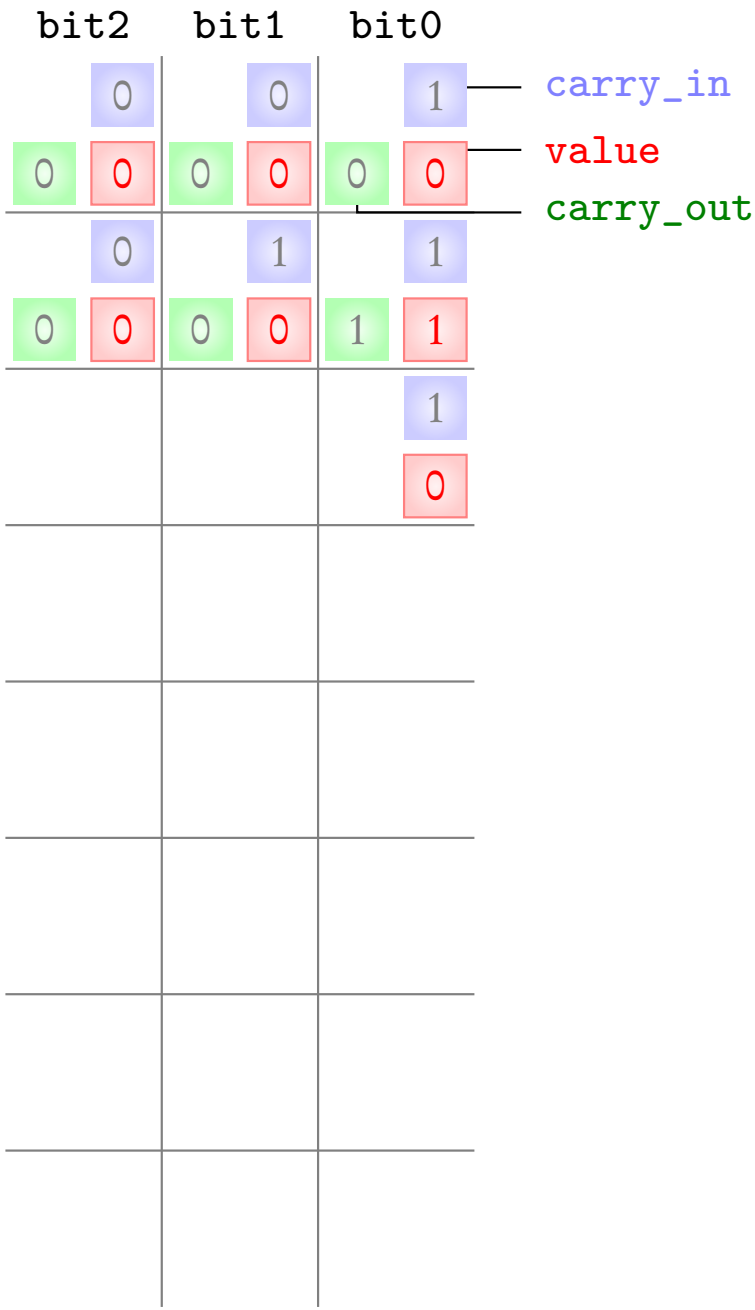

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



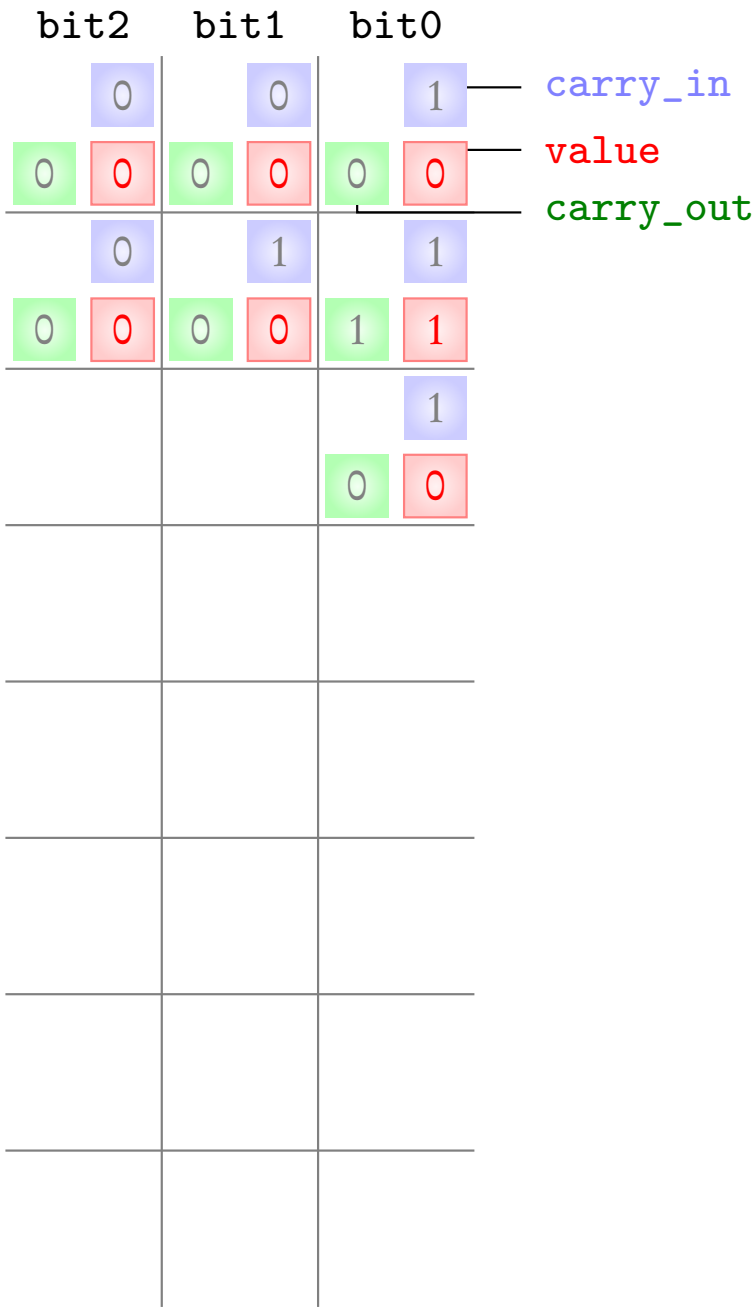

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



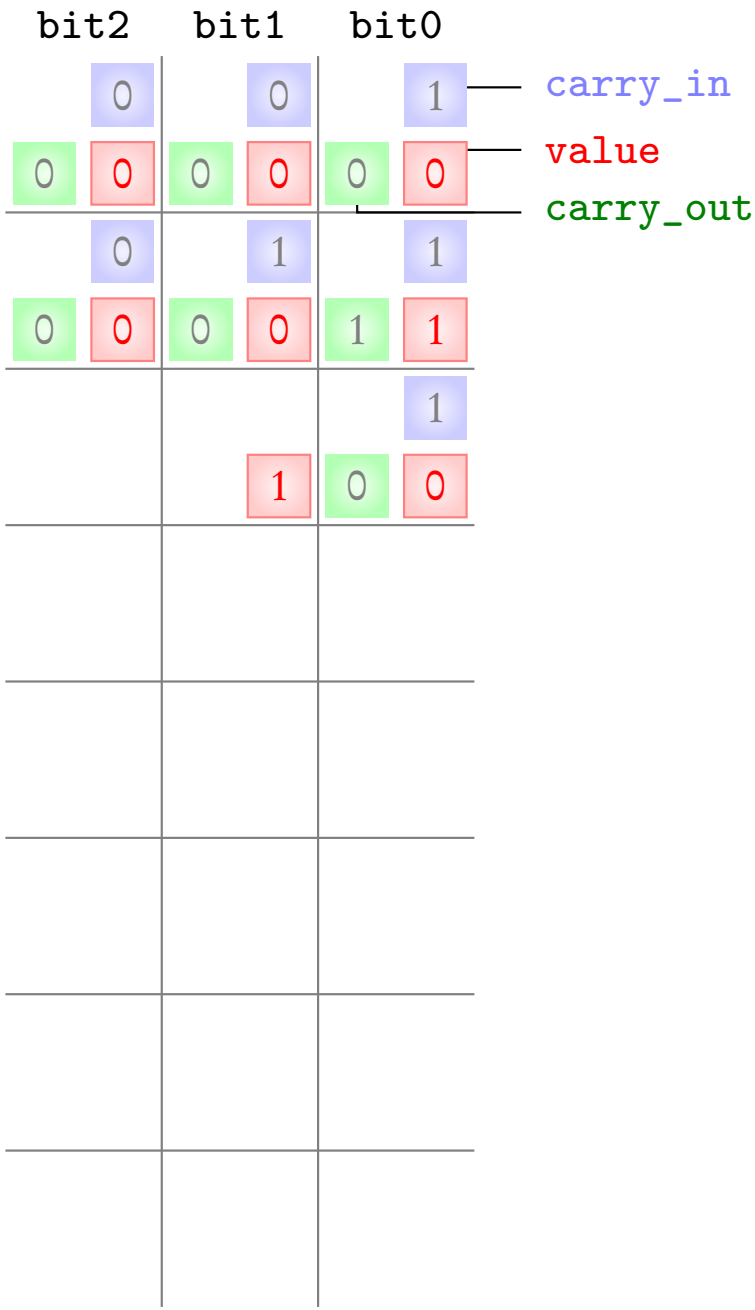

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



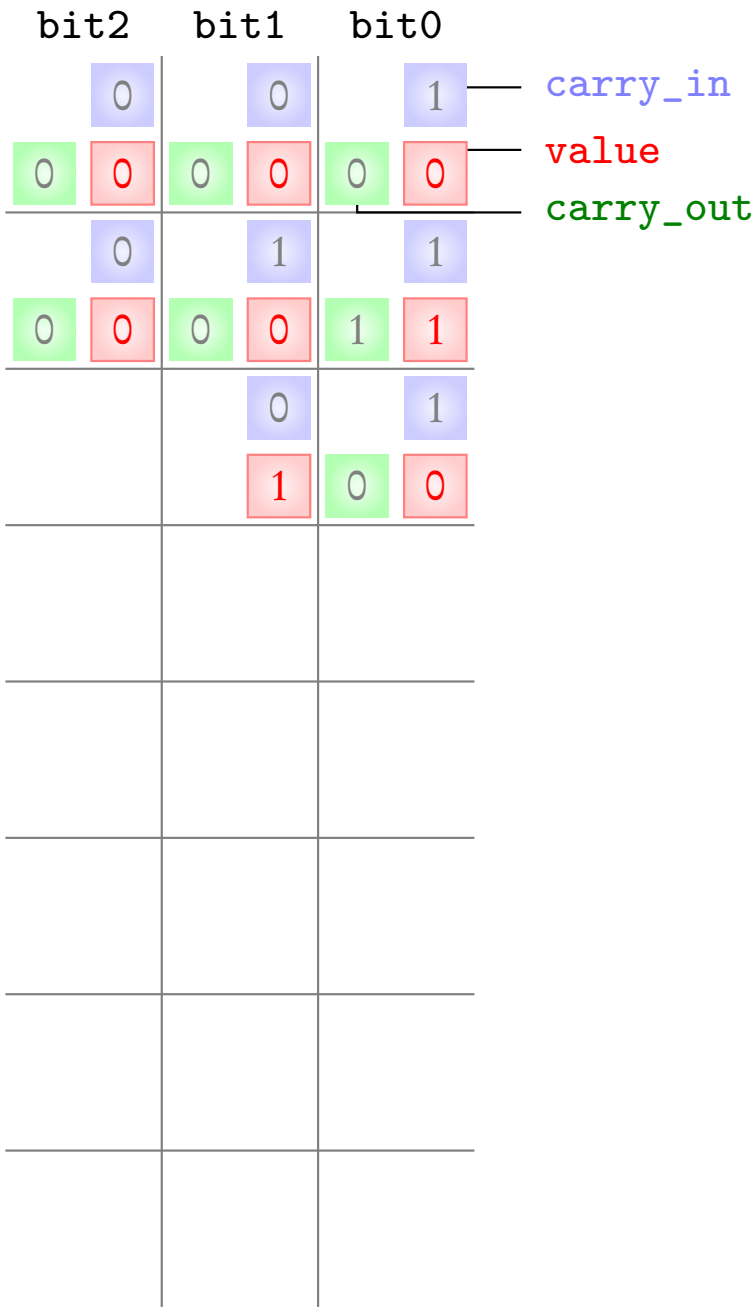

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



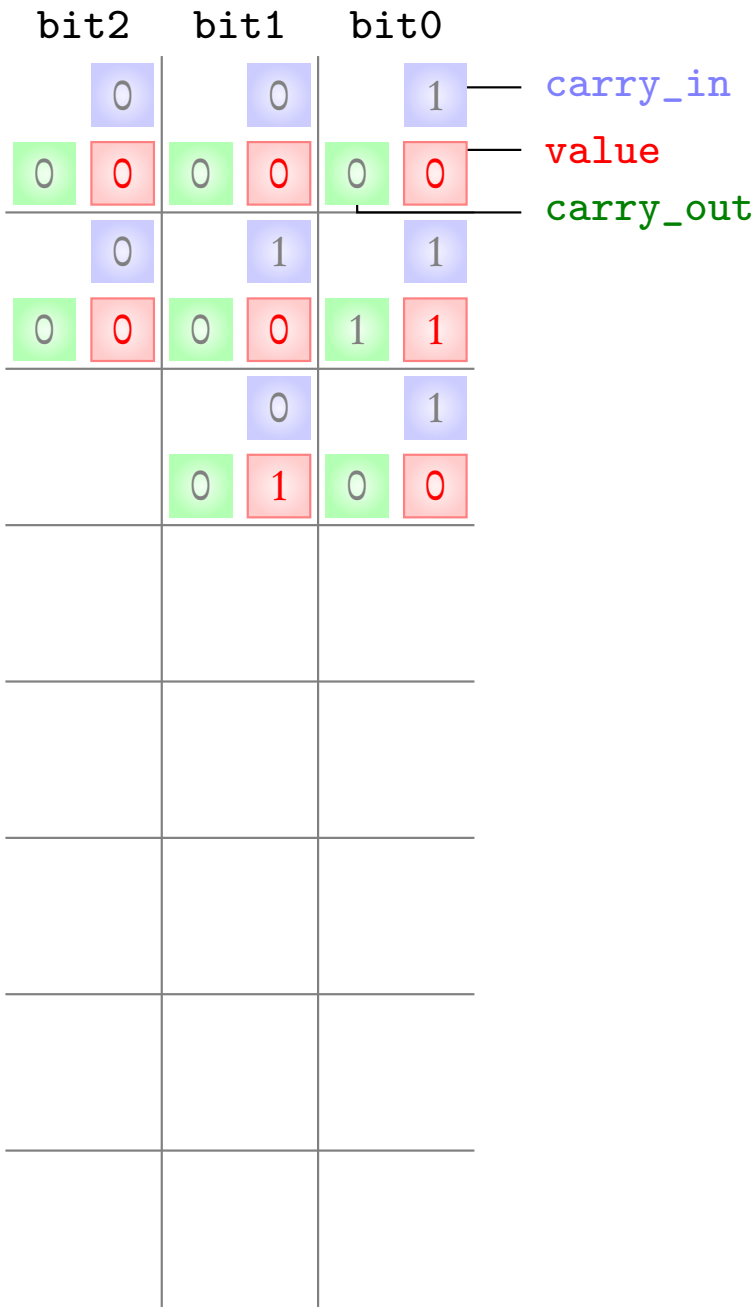

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



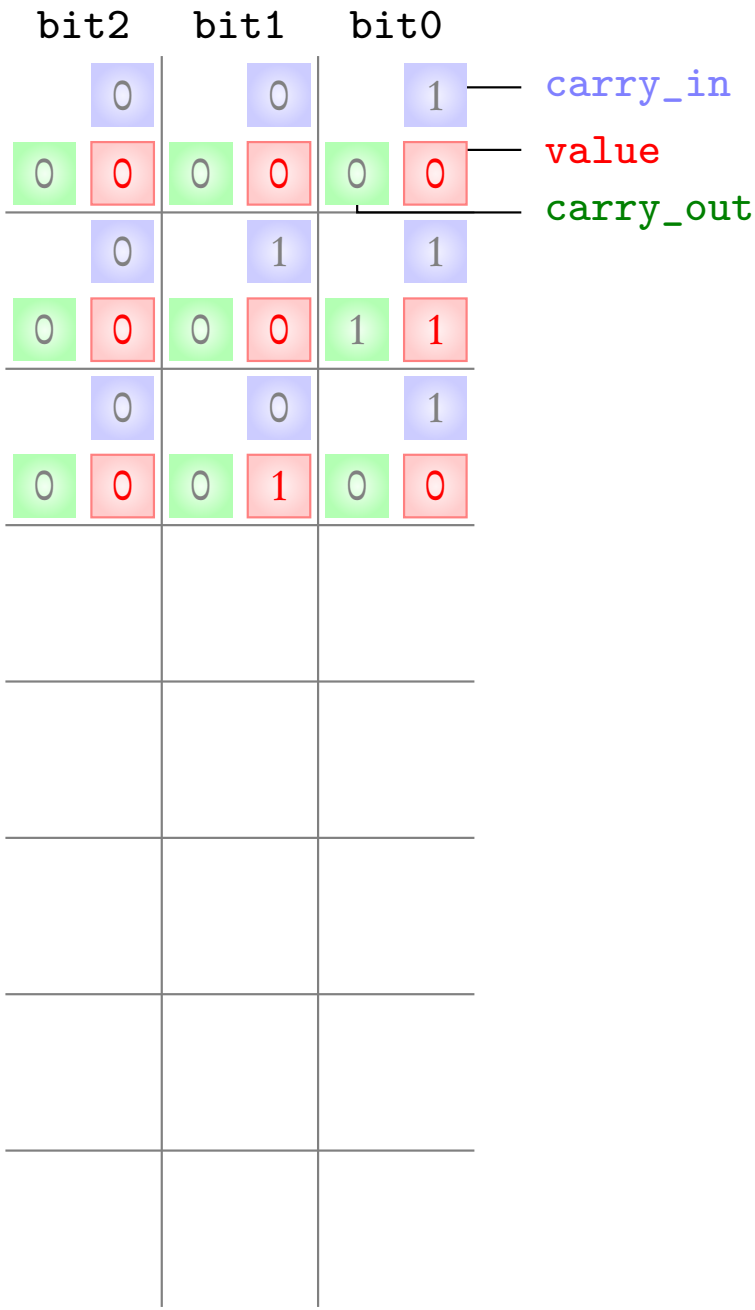

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



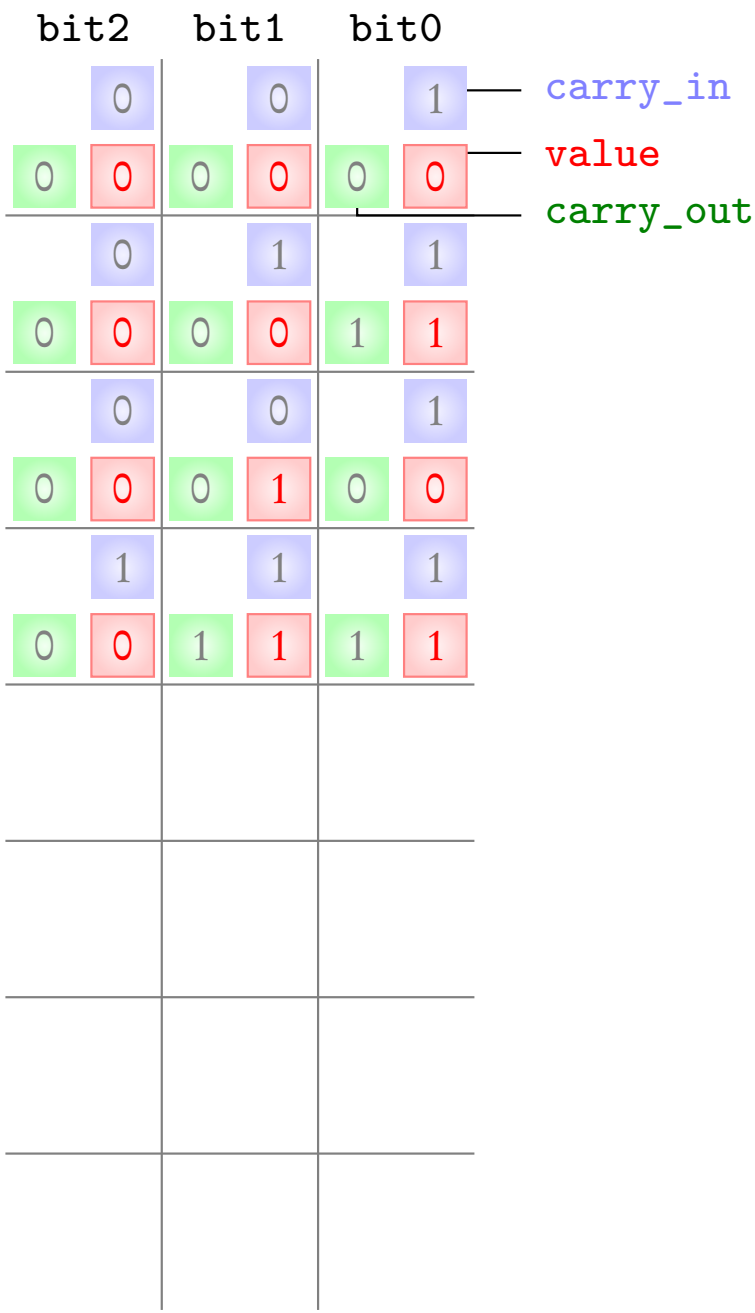

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



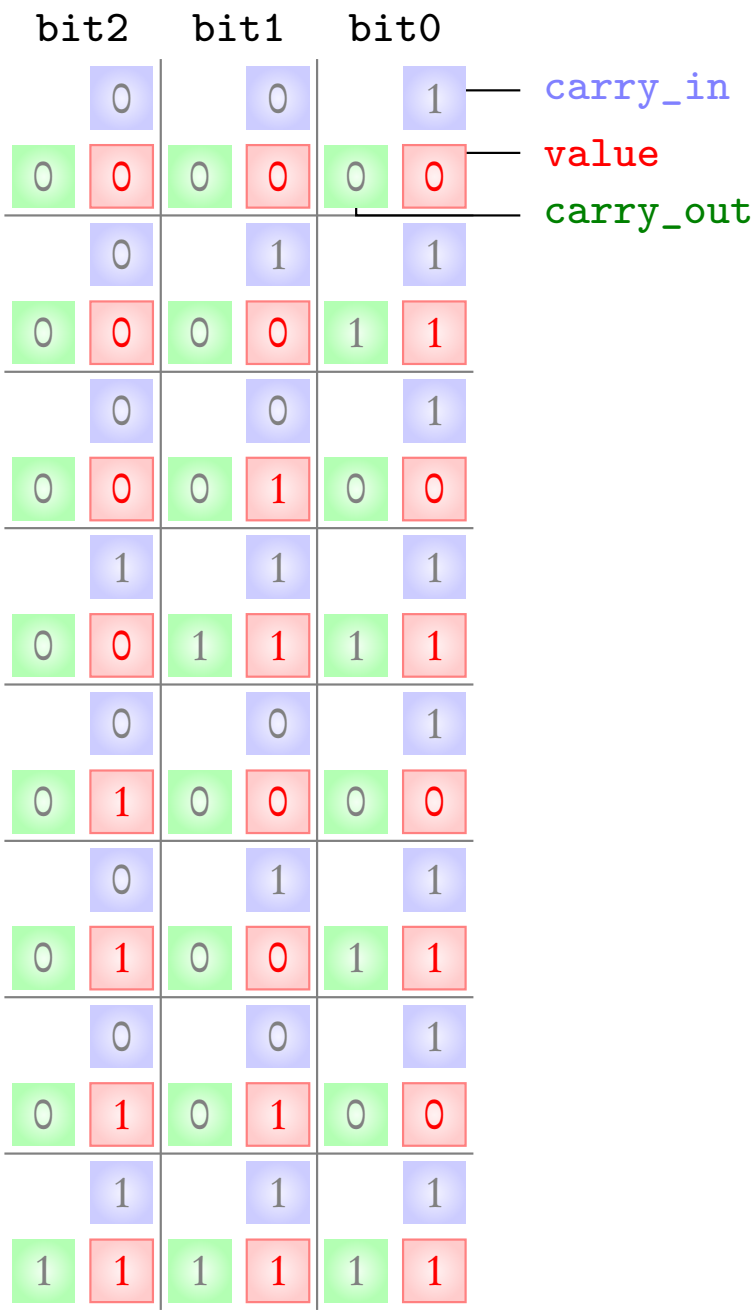

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



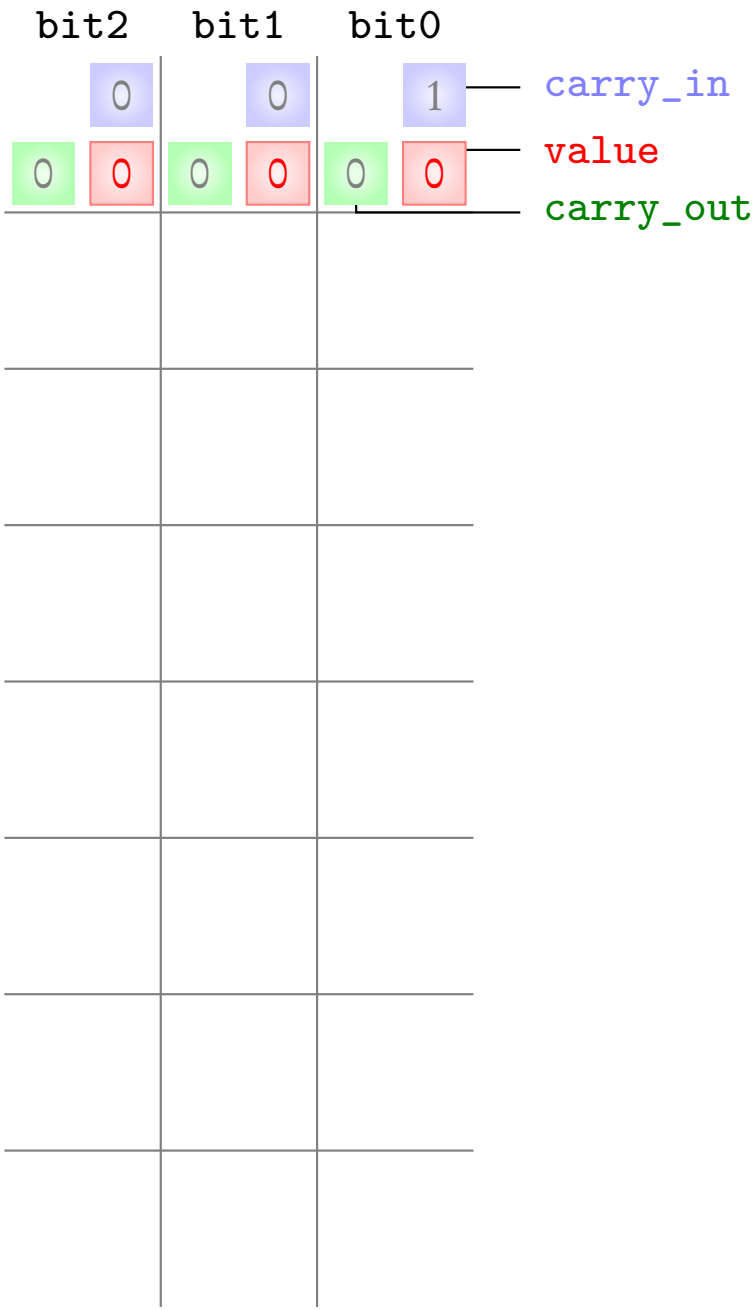

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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

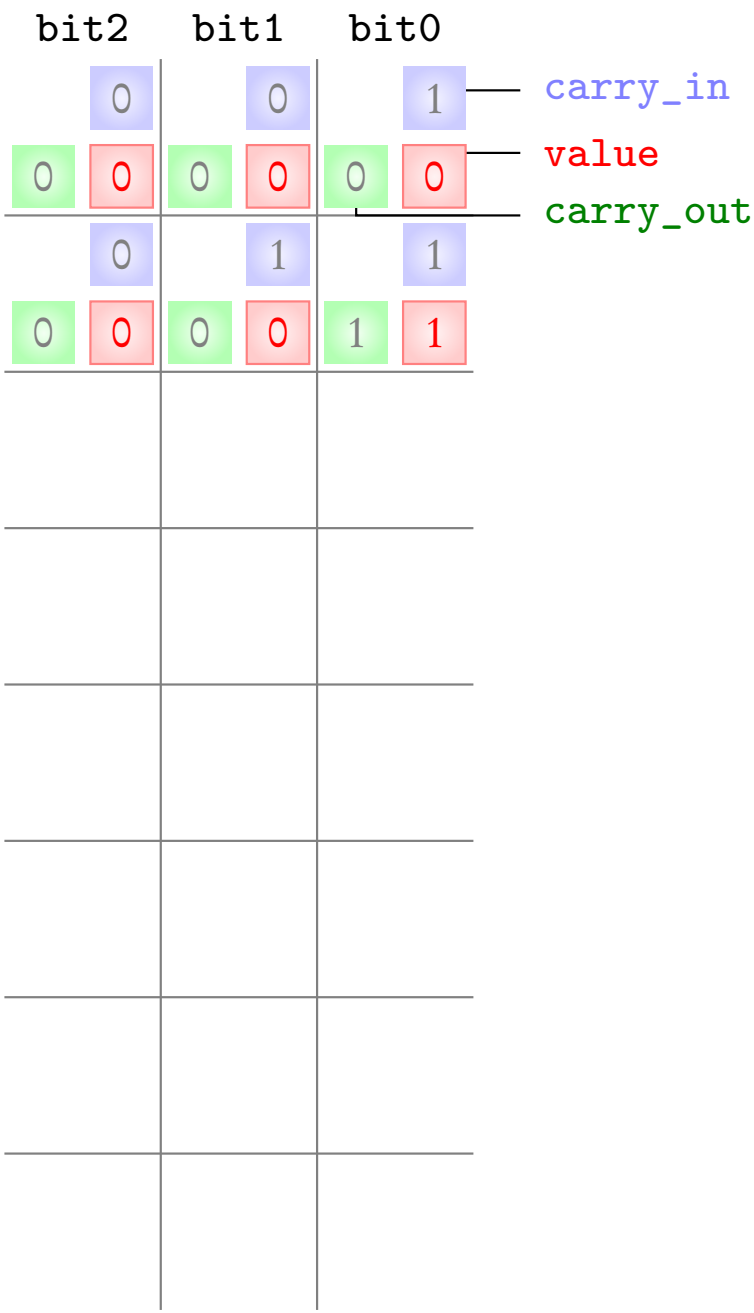


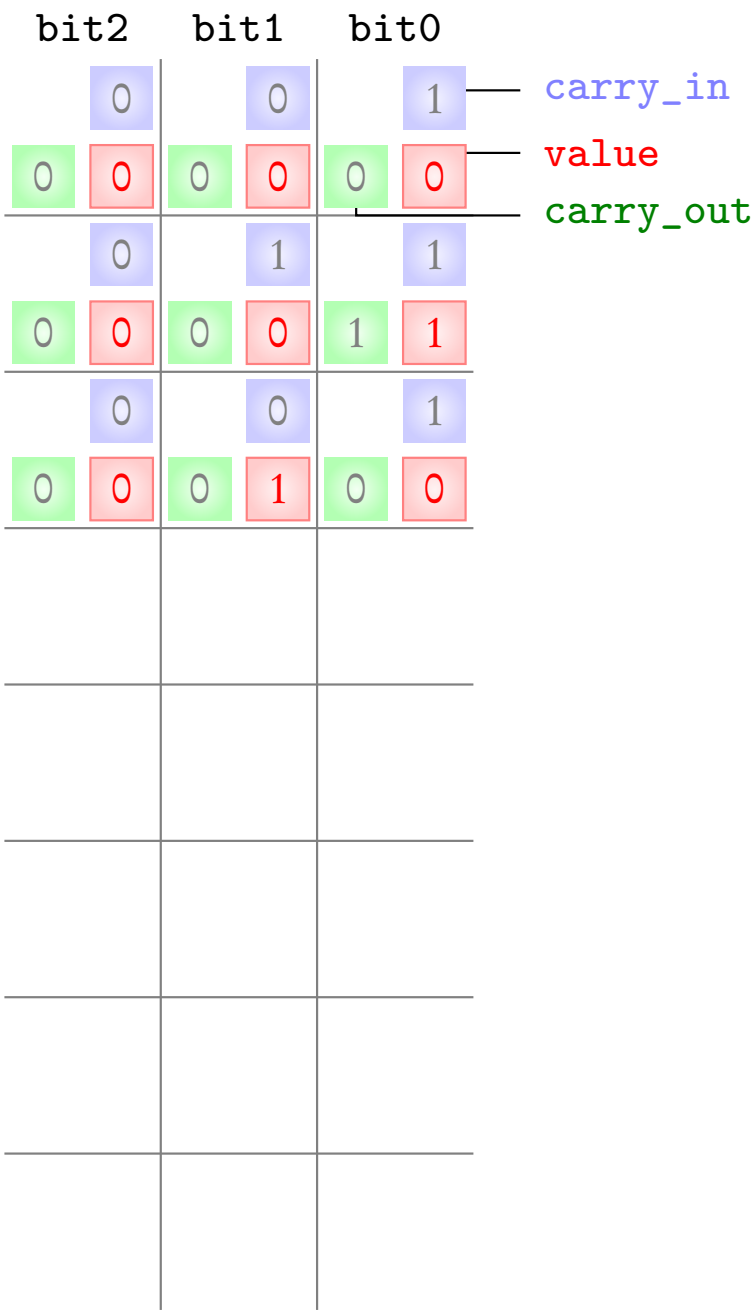

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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

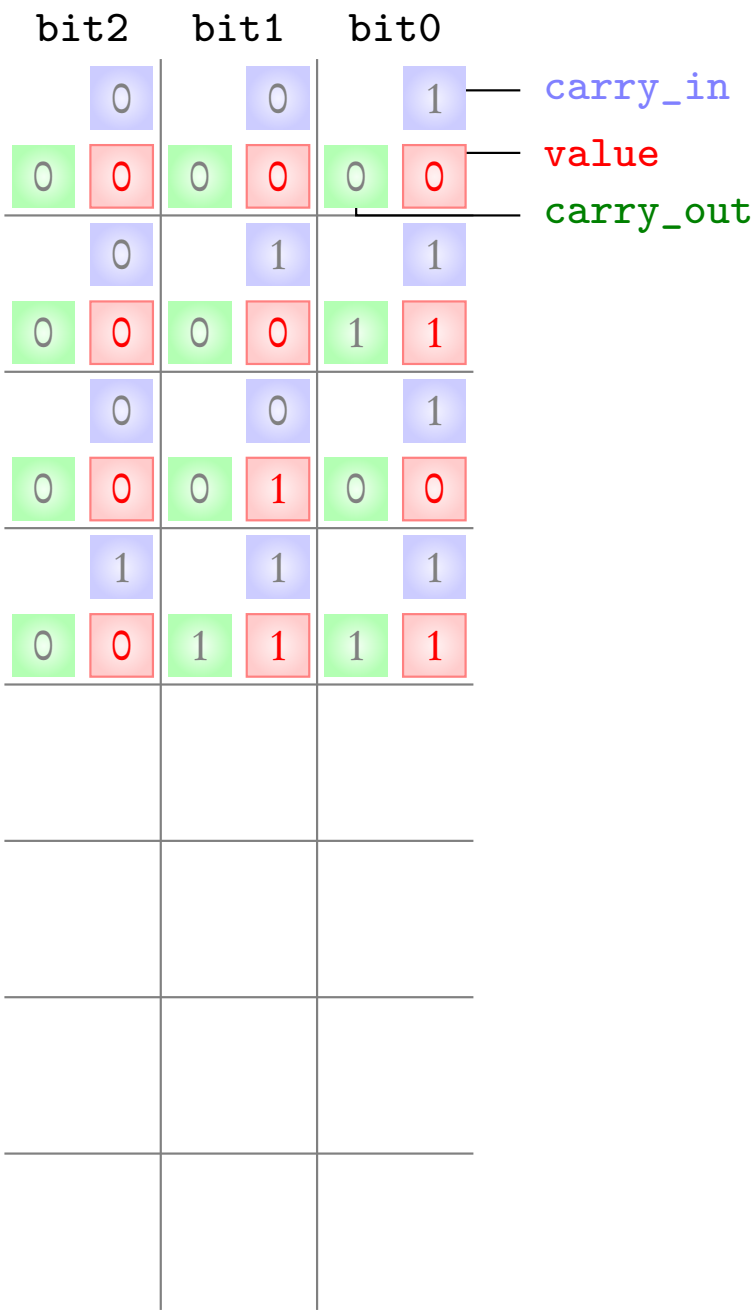


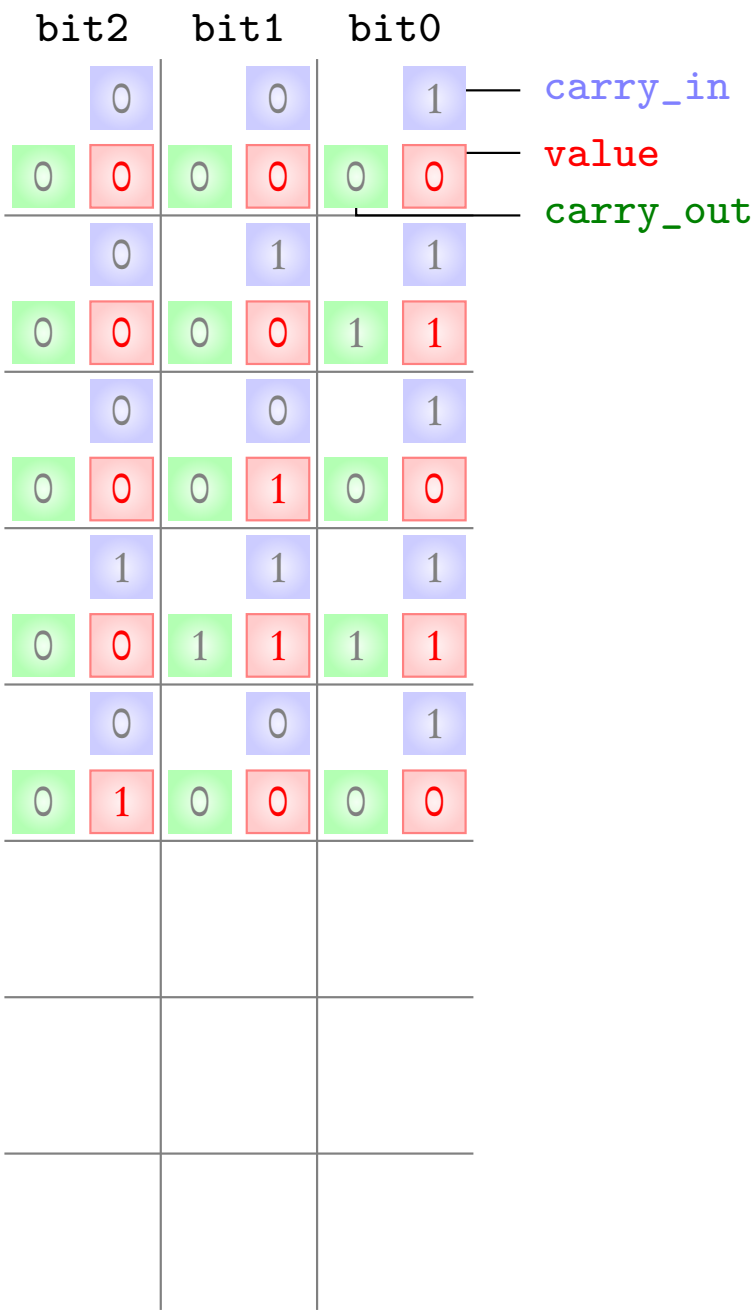

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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

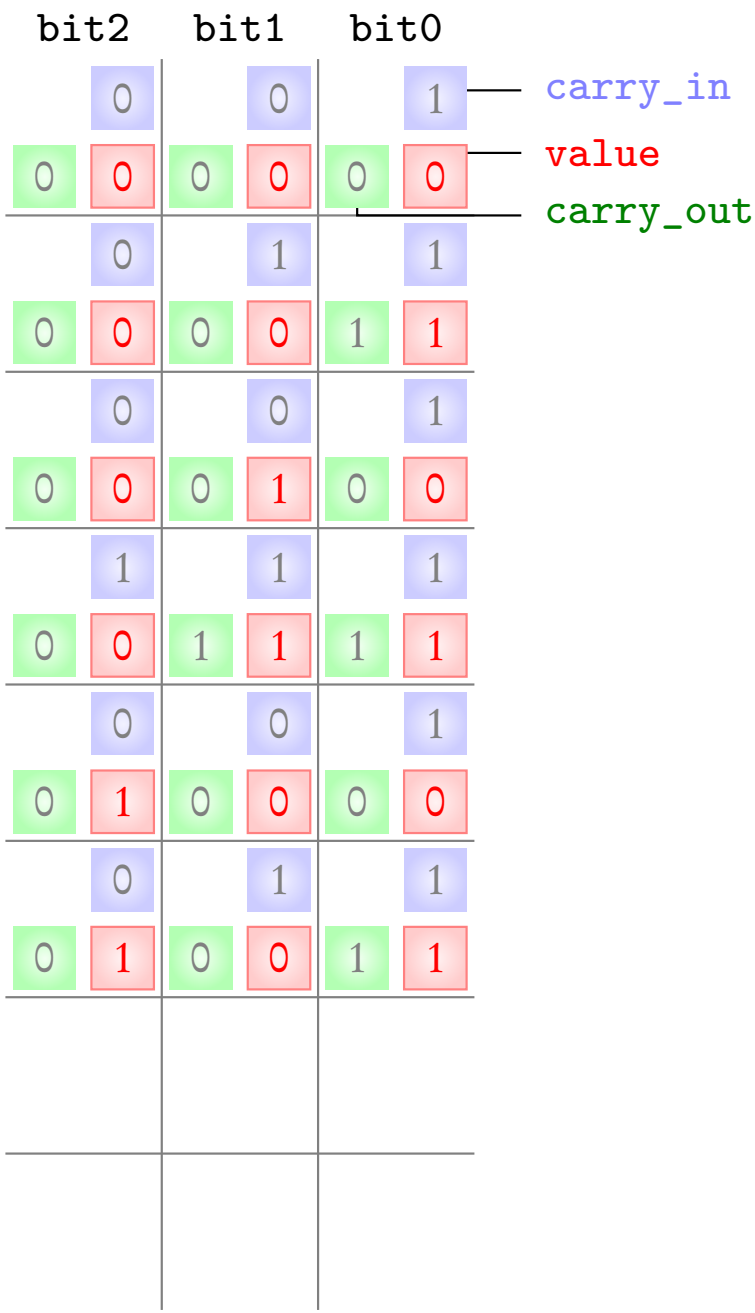

bit2
bit1
bit0
0
0
1
carry_in
0
0
0
0
0
0
value
carry_out

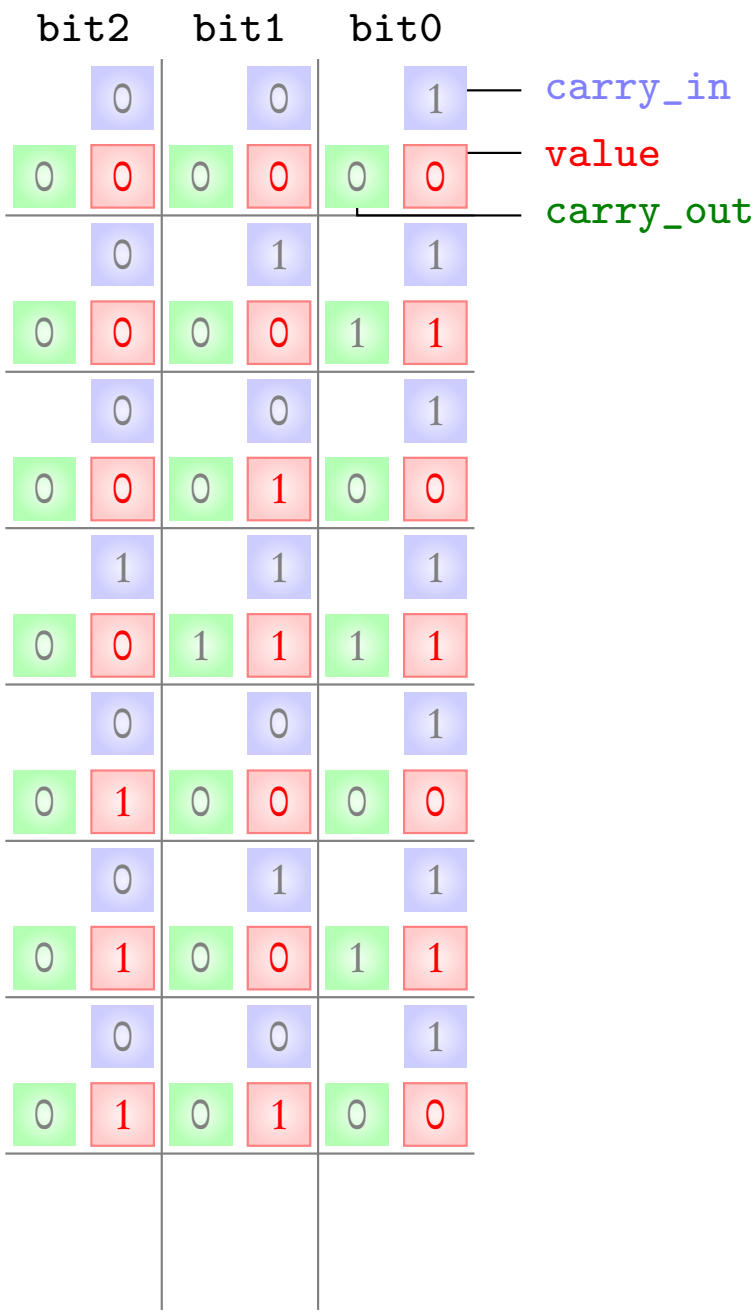

bit2
bit1
bit0
0
0
1
carry_in
0
0
0
0
0
0
value
carry_out
0
1
1
0
0
0
0
1
1

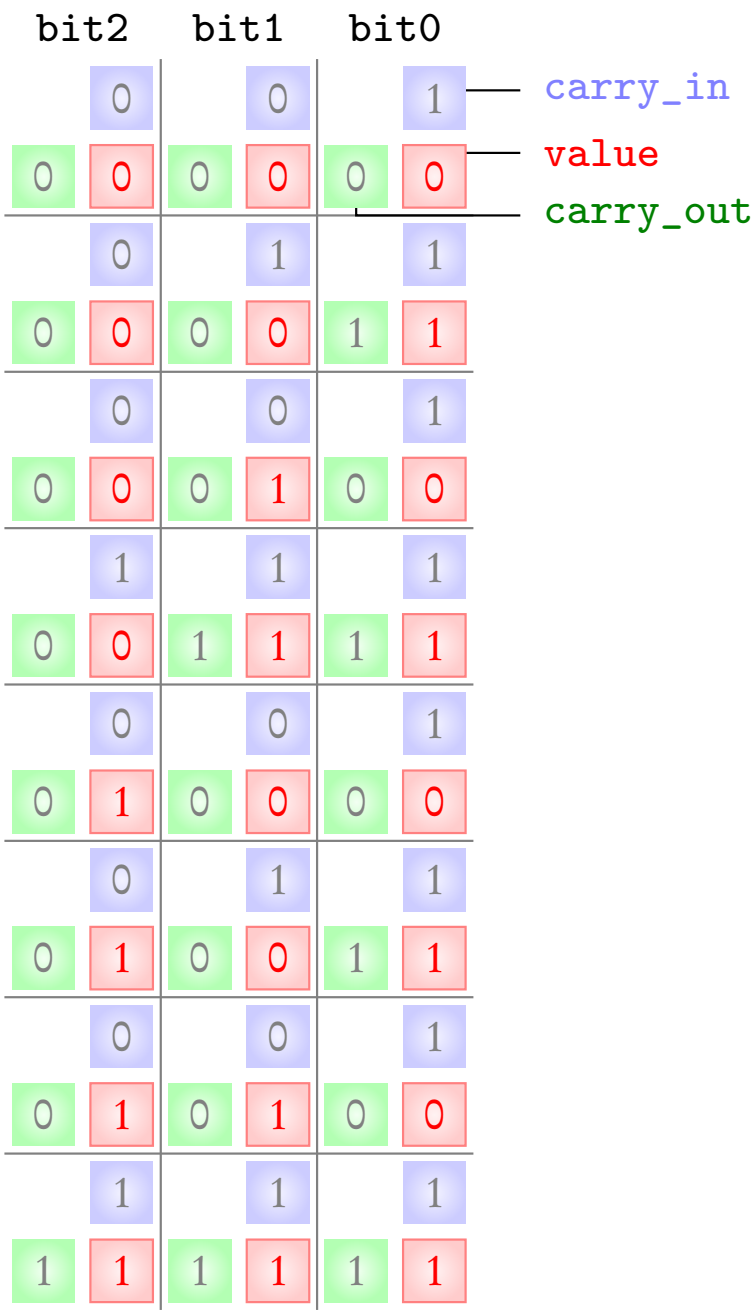

bit2
bit1
bit0
carry_in
value
carry_out
0 0 0 0 0 1
0 0 0 0 0 0
0 0 0 1 1 1
0 0 0 0 1 1
0 0 0 1 0 0

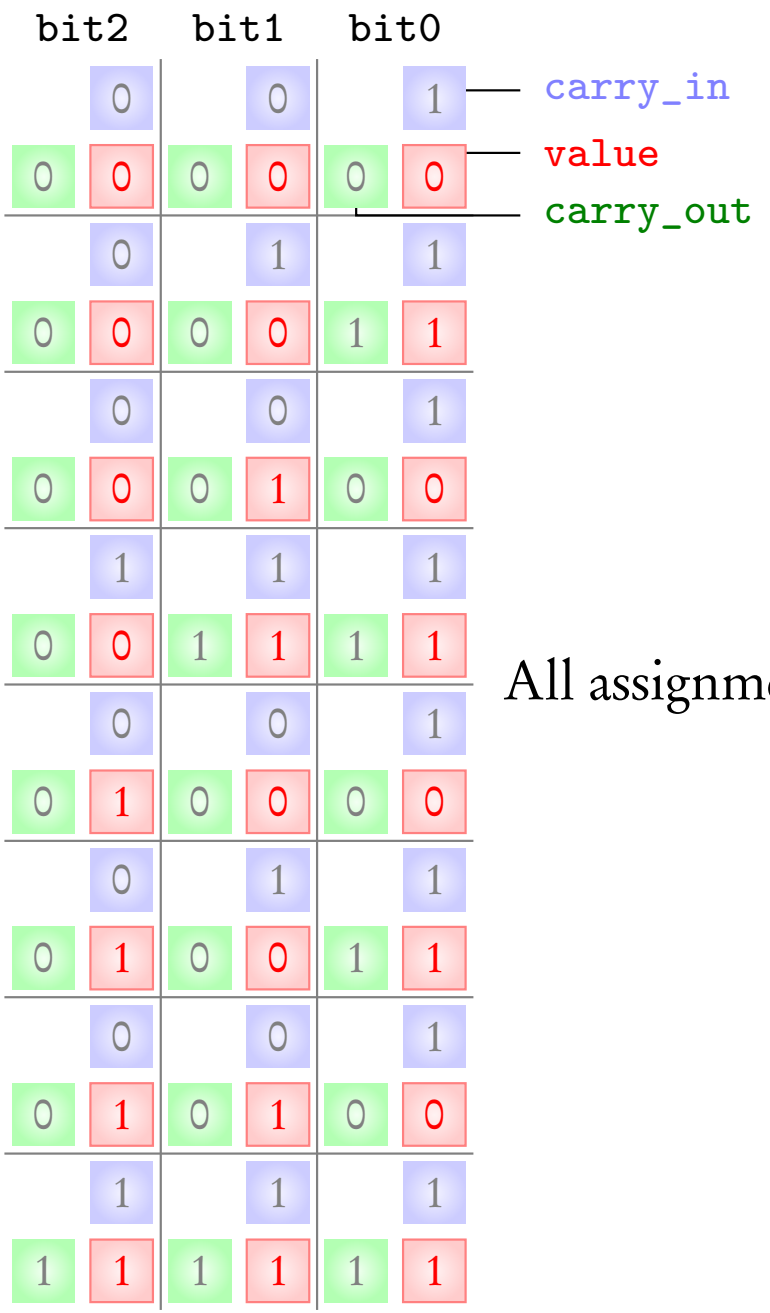

bit2
bit1
bit0
carry_in
value
carry_out


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)

Simple circuit

Use of `DEFINE`

Hierarchical designs

Use of `MODULE`

Counter

Synchronous composition of modules