Half Adder Component

A/ \ B/	0	1
0	0	1
1	1	0

sum/

{[A/0, B/0], [A/1, B/1]} -> sum/0;
{[A/1, B/0], [A/0, B/1]} -> sum/1;

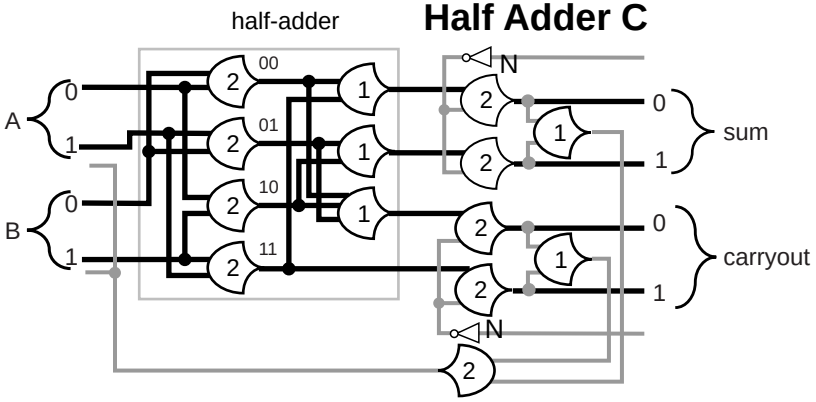
A/ \ B/	0	1
0	0	0
1	0	1

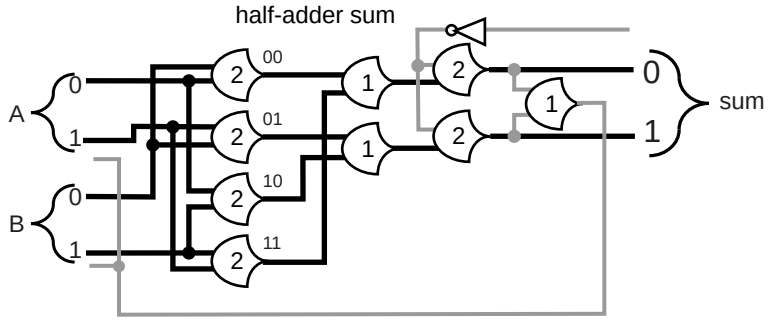
carry/

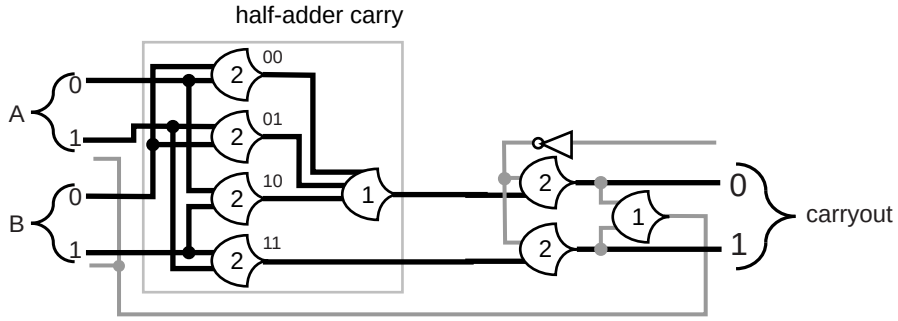
{[A/0, B/0], [A/1, B/0], [A/0, B/1]} -> carryout/0;
[A/1, B/1] -> carryout/1;

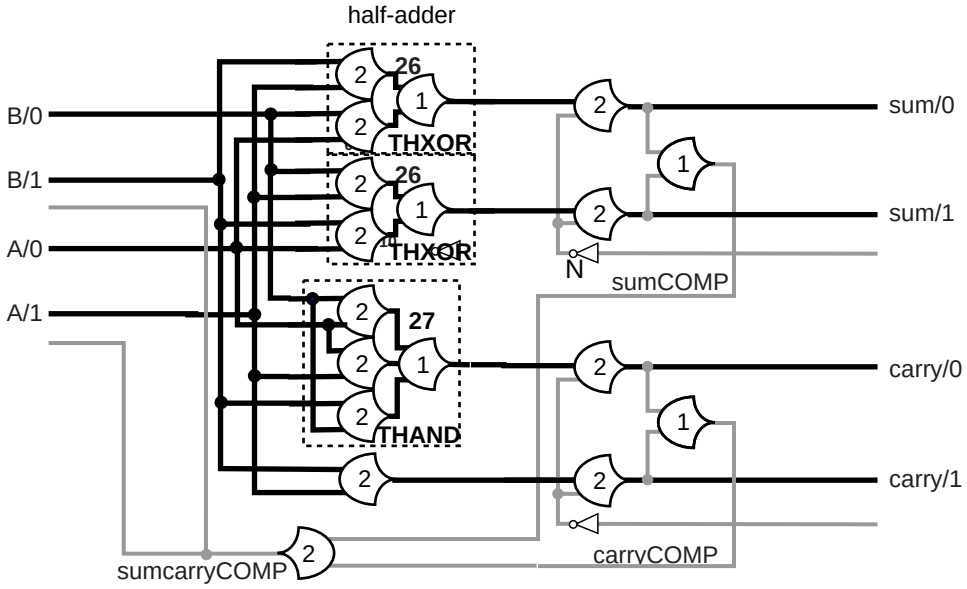
Symbolic specification

```
// specify the half add component
halfadd(A, B -> sum, carryout){
flow [A, B] -> [sum, carryout];
token {0:1} A, B, sum, carryout;
   {[A/0, B/0], [A/1, B/1]} -> sum/0;
   {[A/1, B/0], [A/0, B/1]} -> sum/1;
   {[A/0, B/0], [A/1, B/0], [A/0, B/1]} -> carryout/0;
   [A/1, B/1] -> carryout/1;
close A <- [sum/#, carryout/#];
close B <- [sum/#, carryout/#];
}
```





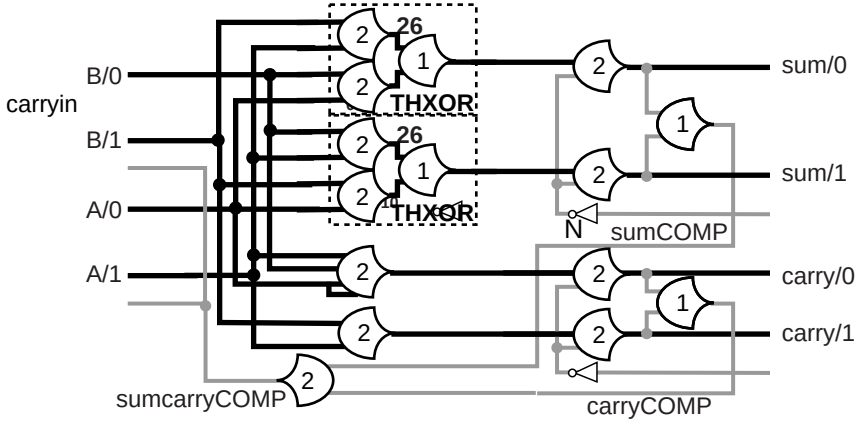




input incompleteness for carry

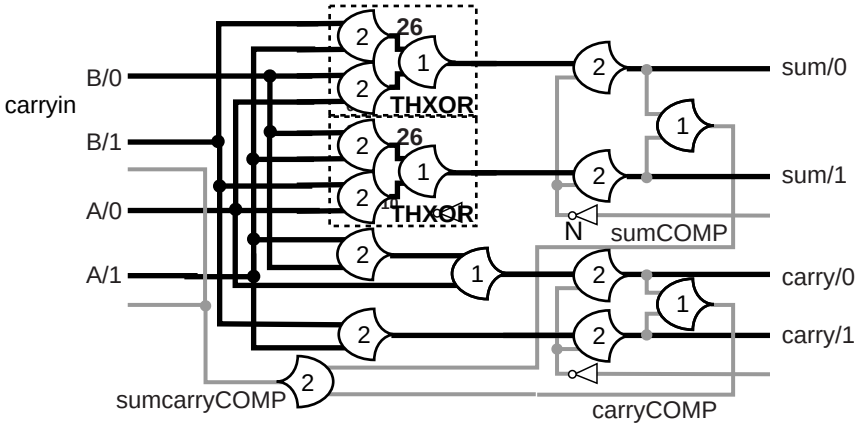
Given two output flow paths of a single shared completeness behavior If the completeness of transition one output flow implies the completeness of transition of the input flow then the other output flow need not imply the completeness of input but only the completeness of propagation and the correcctness of the output. The AND of the completeness of the two output flows closes with the input and fulfills the completeness criterion

In the case of the half adder the sum implies the completeness of the input so the carry can be generated with partial input. Specificaly, if A/0 then carry/0 and it is not necessary to wait on B to generate the carry. In the case of the counter B will be the carryin from the lower order digit. So to generate carryout half the time we will not have to wait on carryin.



Half Adder A

The carry orphan causes a wait for the null carry ripple

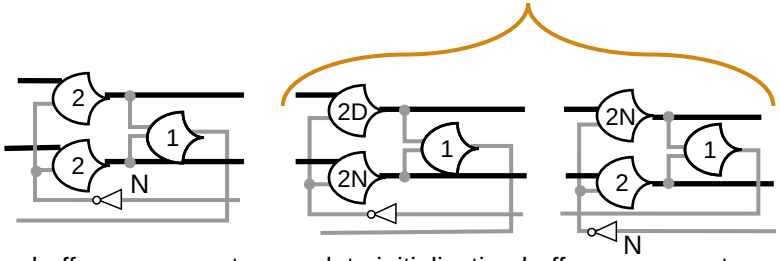


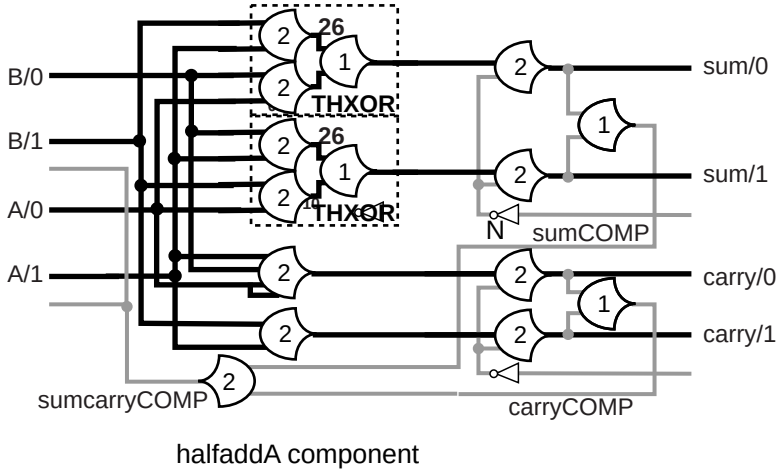
Half Adder B

The carry orphan does not cause a wait for the null carry ripple

Composing the half adder ring

initialize data wavefront





halfadd(B, carryin -> count,carryout)

Symbolic specification

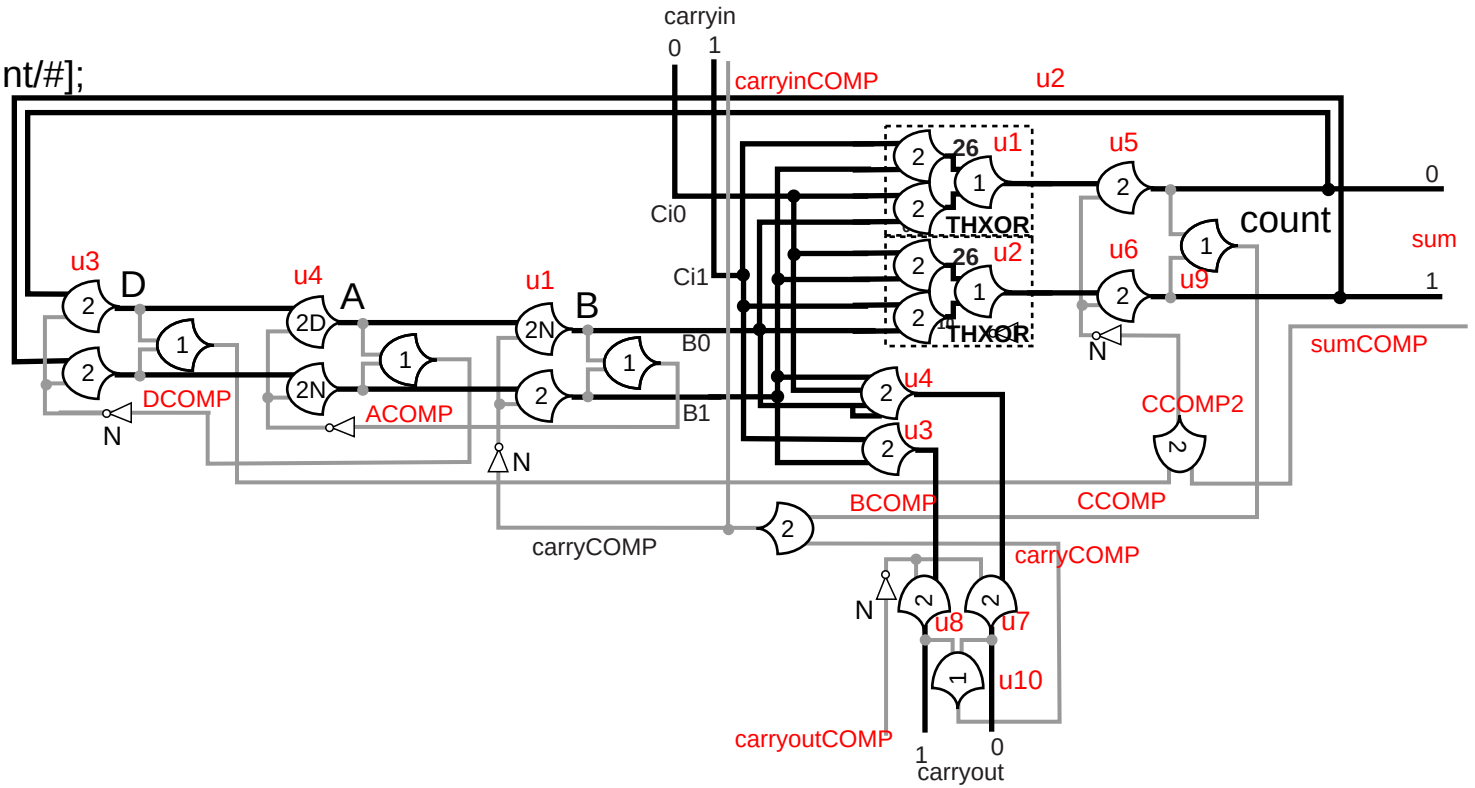
```
// compose the digit halfadd ring
digit_count(carryin -> carryout, count){
flow carryin -> [count, carryout];
token {0:1} carryin, carryout, count, B, D, A(0);
   halfadd(B, carryin -> count,carryout)
    (count -> D);
    (D -> A);
    (A -> B);
close carryin <- {carryout/#, count/#];
close count <- [D/#, ?/#];
close carryout <- ?/#;
}
```

Red names indicate corespondences in the verilog code

twoD-counter_ringA.v
twoD-counter_ringB.v
twoD-counter_ringC.v

.

halfadd ring counter component



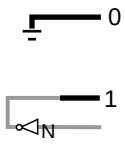
Canonical composition

direct mapping from symbolic specification

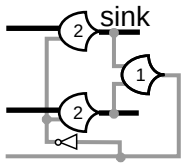
Symbolic specification

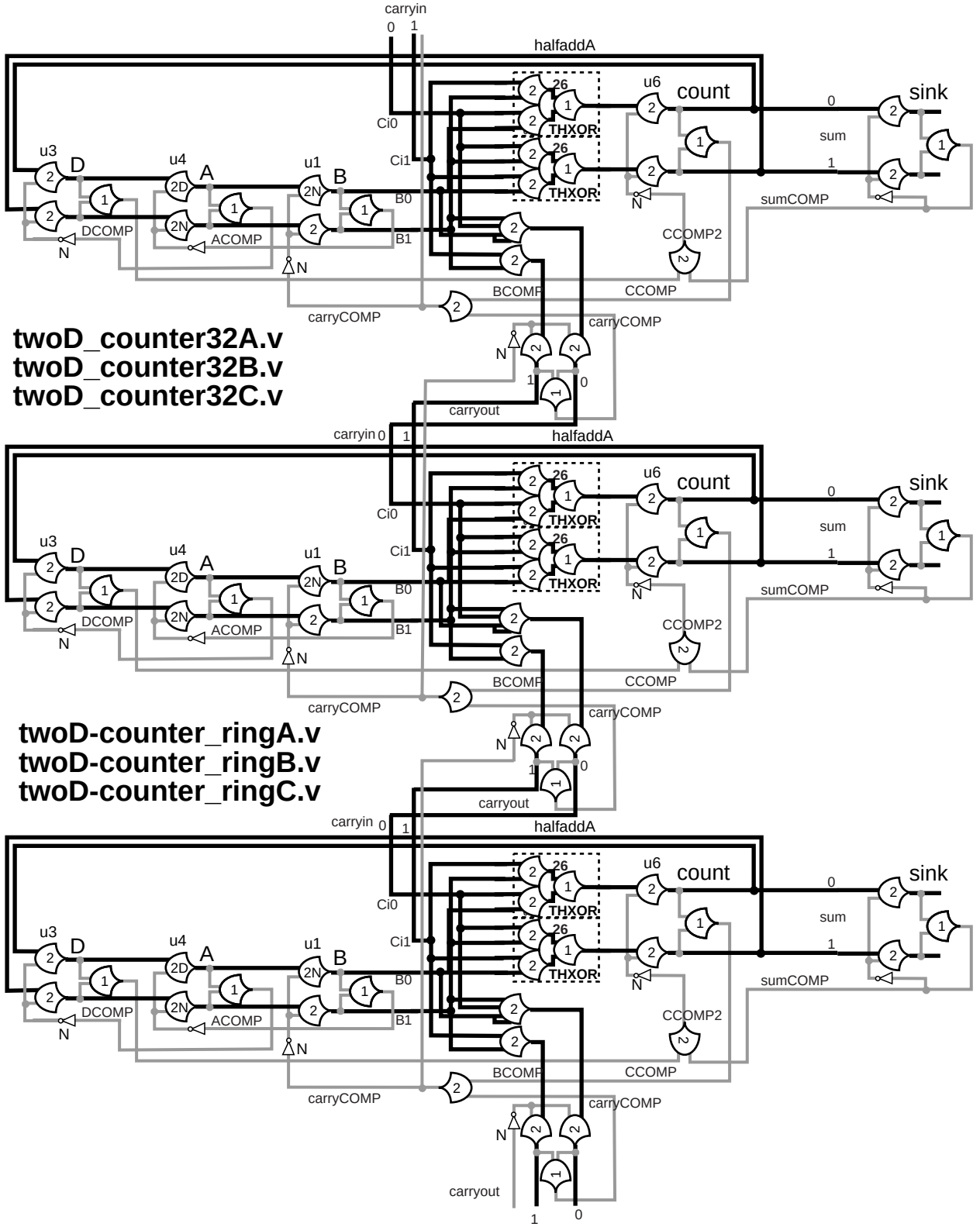
```
// sink the flow path
(count -> ){
flow count -> ;
token count[0:31]{0:1};
path count/i <
  (count/i -> );
>
}

// compose the flow path
counter( -> count){
flow -> count;
token count[0:31]{0:1};
token carry[0:32]{0:1};
path count/i <
  (1 -> carry/0);
  digit_count(carry/i -> count/i, carry/i+1)
 (carry/max(i)+1 -> ));
>
close count <- ?/#;
}
```

constant 1 component



sink component

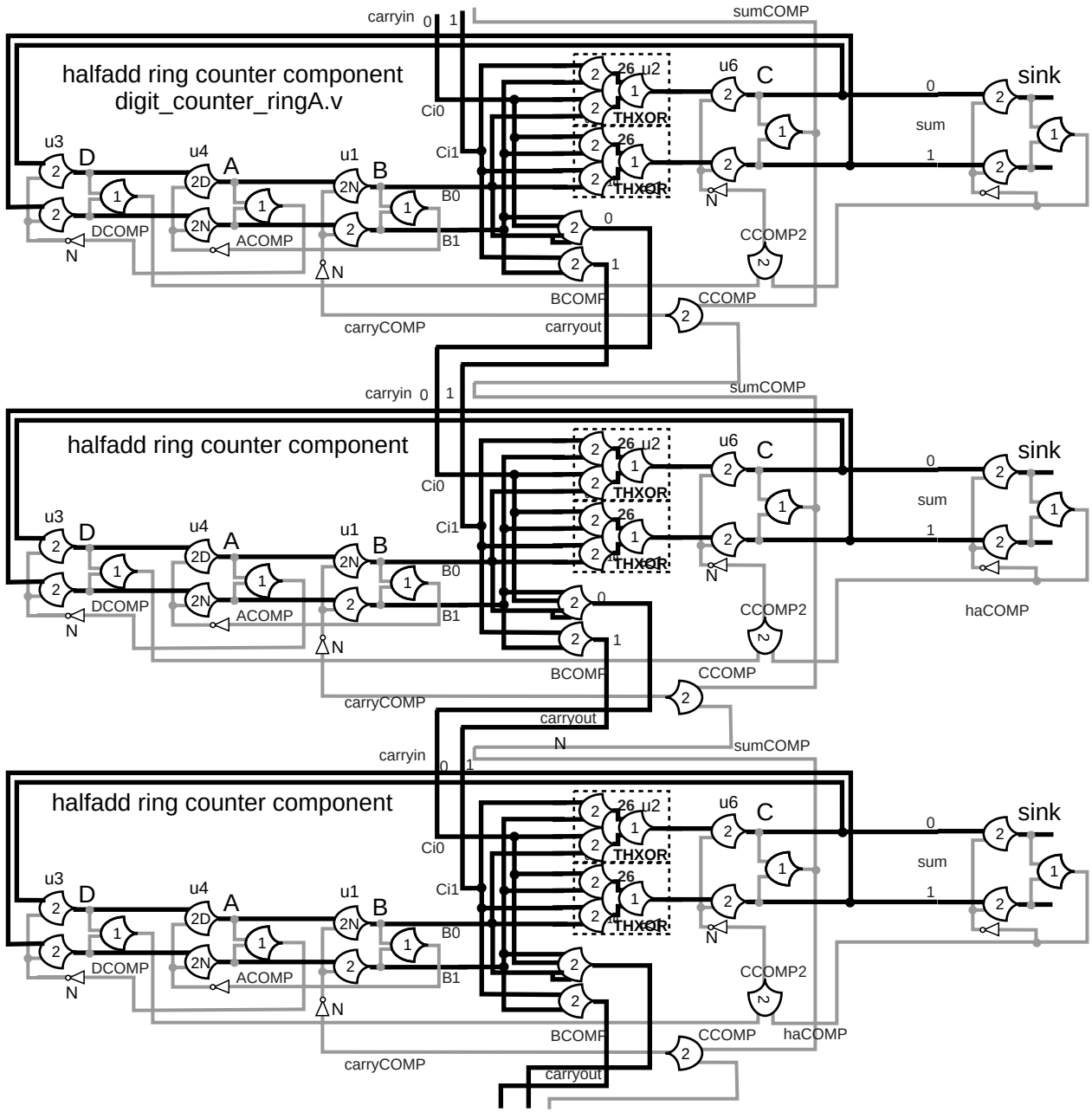




Cost reduction of carry pipeline

digit_counter32A.v
digit_counter32B.v

digit-counter_ringA.v
digit-counter_ringB.v



Imposition of full token completeness

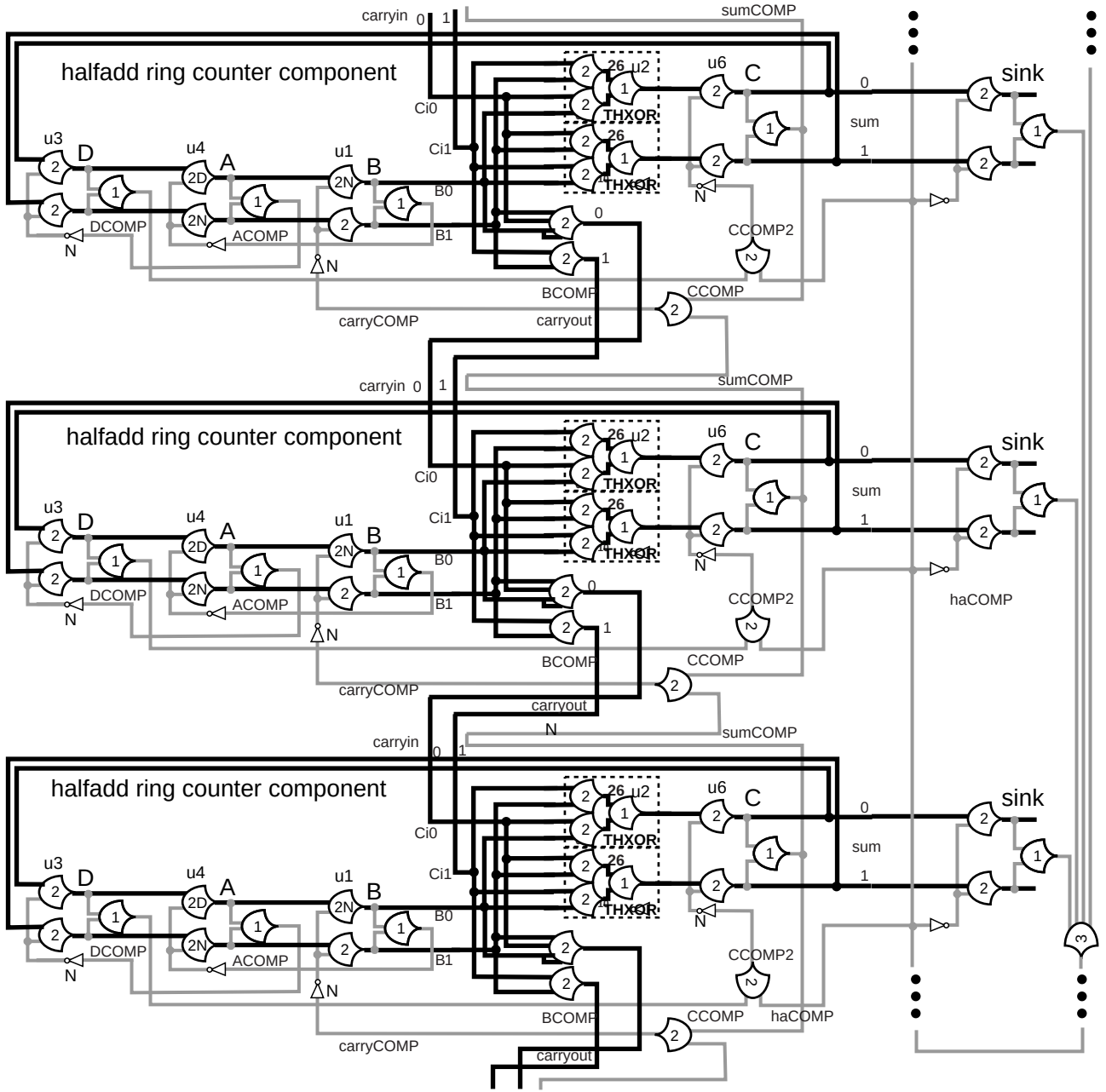
fullword_counter32A.v
fullword_counter32B.v

fullword-counter_ringA.v
fullword-counter_ringB.v

The flowing wavefronts will conform to whatever logical constraint is imposed

```
counter( -> count){
flow -> count;
token count[0:31]{0:1};
token carry[0:32]{0:1};
path count/i <
  (1 -> carry/0);
  digit_count(carry/i -> count/i, carry/i+1)
 (carry/max(i)+1 -> ));
>
close count <- ?/#;
}
```

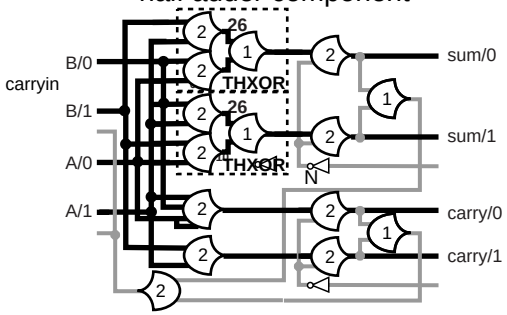
Close count <- ?/#
specifies that the complete token has to be closed each oscillation but it does not specify how the closure must occur.



Fully integrated combinational logic

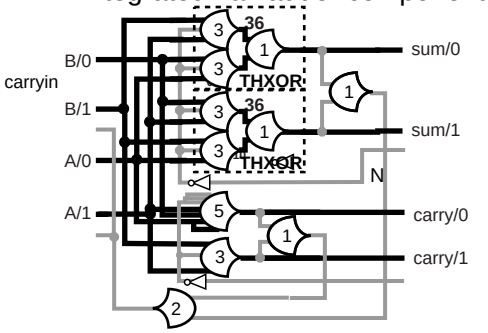
No gate in the circuit is performing solely combination service. Every gate in the circuit is performing flow coordination duty and a few are also performing combinational duty.

half adder component



A long delay gate creates a long period oscillation which paces the rest of the circuit. Consequently it is slower than the non integrated circuit. But it is fewer gates.

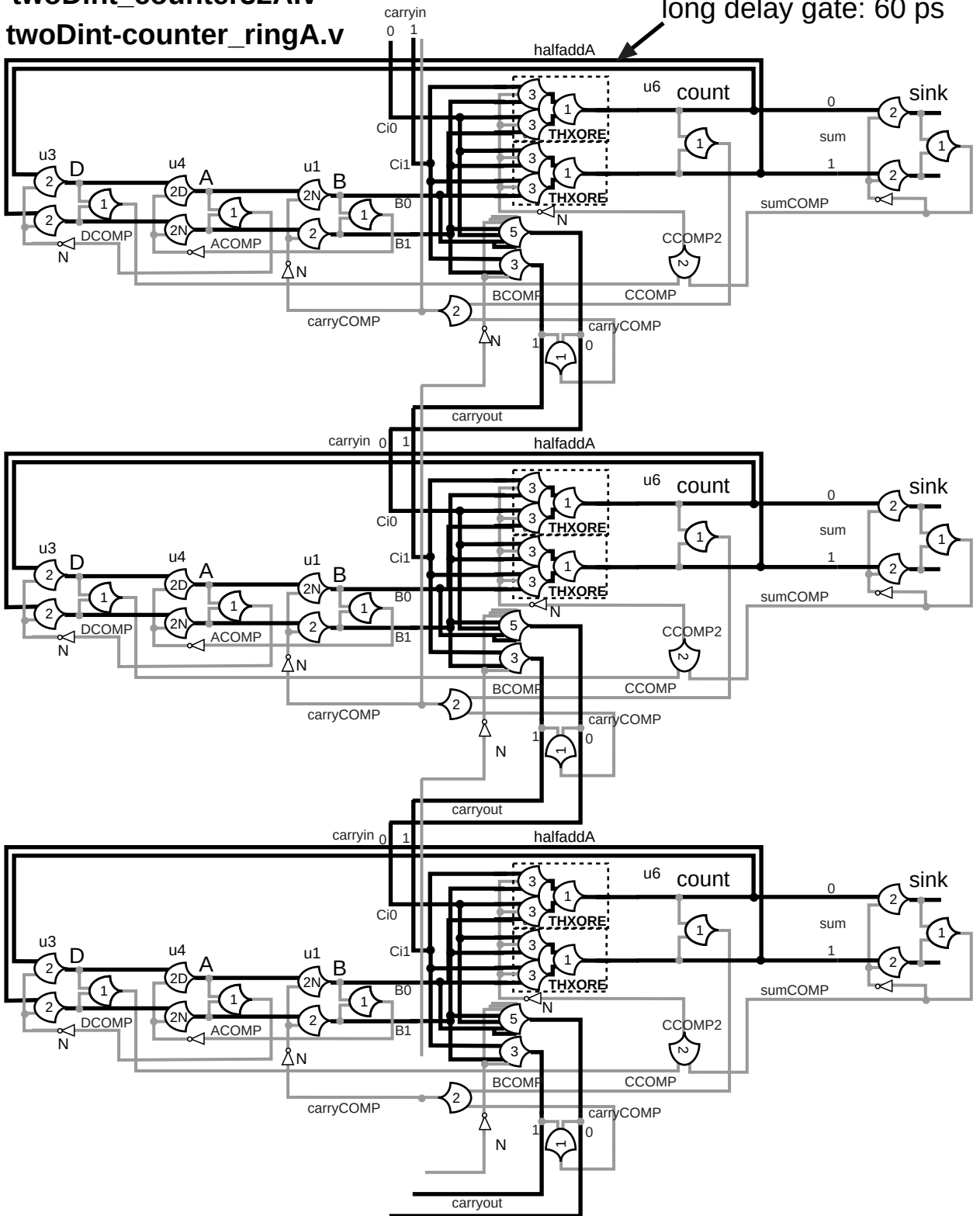
integrated half adder component



twoDint_counter32A.v

twoDint-counter_ringA.v

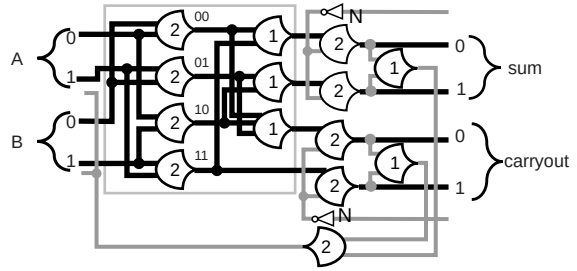
long delay gate: 60 ps



Fully integrated combinational logic

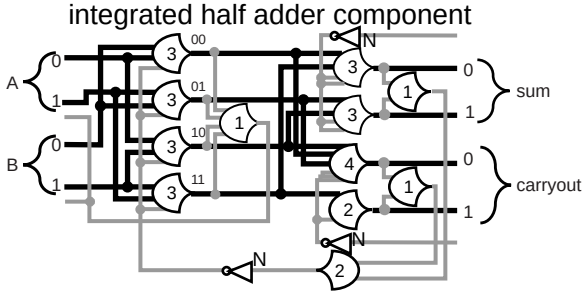
No gate in the circuit is performing solely combination service. Every gate in the circuit is performing flow coordination duty and a few are also performing combinational duty.

half adder component



The combinational work is distributed over two oscillations with shorter periods. The circuit as a whole flows faster. but it has more gates and more transitions.

integrated half adder component



twoDint_counter32C.v

twoDint_counter_ringC.v

shorter delay gate: 2 oscillations