Composition Example

Two Bit Multiplier

A two bit multiplier design is composed in terms of

connecting correspondingly named component boundary tokens

and is analyzed by

tracking the evolving boundary specifications,

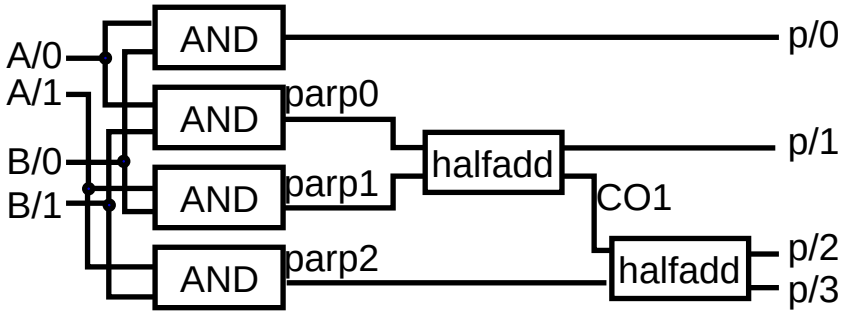
flow relations and

closure equations

Two bit multiplier design

Language specification

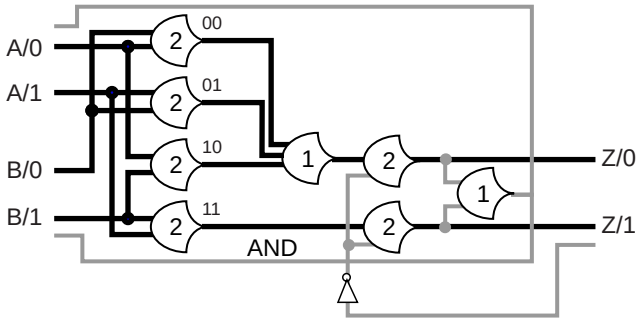
```
twobitmul(A, B -> p){
flow [A, B] -> p;
token dual{0,1};
token (A[0,1], B[0,1], p[0-3])[dual];
token (parp0, parp1, parp2, CO1)[dual];
AND(A/0, B/0 -> p/0);
AND(A/0, B/1 -> parp0);
AND(A/1, B/0 -> parp1);
AND(A/1, B/1 -> parp2);
HA(parp0, parp1 -> p/1, CO1);
HA(CO1, parp2 -> p/2,  p/3);
}
```



components

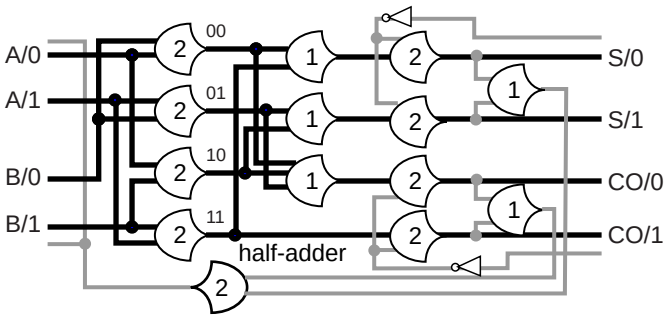
Component circuits

```
AND(A, B -> Z){   // AND
token (A, B, Z){0,1};
flow [A, B] -> Z;
{[A/0, B/0], [A/0, B/1], [A/1, B/0]} -> Z/0;
[A/1, B/1] -> Z/1;
}
close A <- #Z;
close B <- #Z;
close Z <- #?;
```




```
HA(A, B -> S, CO){   // half adder
token (A, B, S, CO){0,1};
flow [A, B] -> [S, CO];
{[A/0, B/0], [A/1, B/1]} -> S/0;
{[A/0, B/1], [A/1, B/0]} -> S/1;
{[A/0, B/0], [A/0, B/1], [A/1, B/0]} -> CO/0;
[A/1, B/1] -> CO/1;
}
close A <- [#S, #CO];
close B <- [#S, #CO];
close S <- #?;
close CO <- #?;
```

If the input flow relation is honored the output flow relation will be fulfilled



The composition method

Vet primitive components:

orphan isolation (no orphans cross the boundary),

completeness fulfilliment (flow relation, closures, equations and boundary all match),

and flow sufficiency (if the input flow relation is met the output flow relation will be fulfilled).

Composition step:

Connect correspondingly named boundary token half oscillations forming a closed internal oscillation.

Merge remaining boundary tokens into the single boundary of the new component.

Merge closure equations

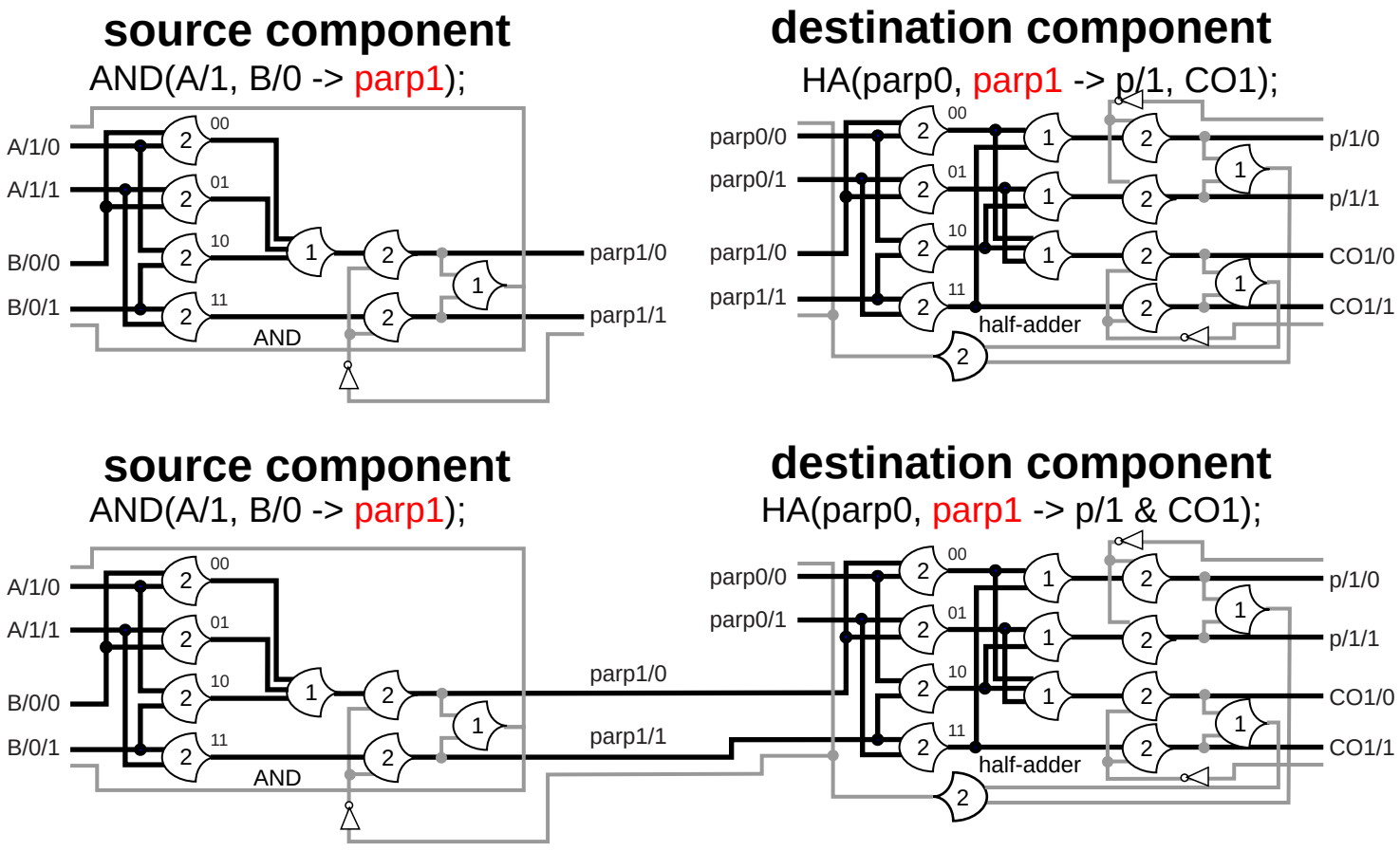
Validate the connection as deadlock free, race free and flow sufficient by tracking the boundary evolution.

A validated flow network grows, validated connection by validated connection.

If the composed result matches the specification and no errors or ambiguities were reported then the specified flow network is implemented and validated

Connect parp1: The Composition

```
twobitmul(A, B -> p){
flow [A, B] -> p;
token dual{0,1};
token (A[0,1], B[0,1], p[0-3])[dual];
token (parp0, parp1, parp2, CO1)[dual];
AND(A/0, B/0 -> p/0);
AND(A/0, B/1 -> parp0);
AND(A/1, B/0 -> parp1);
AND(A/1, B/1 -> parp2);
HA(parp0, parp1 -> p/1, CO1);
HA(CO1, parp2 -> p/2, p/3);
}
```



Connect parp1: The Symbolic Analysis

```
twobitmul(A, B -> p){
flow [A, B] -> p;
token dual{0,1};
token (A[0,1], B[0,1], p[0-3])[dual];
token (parp0, parp1, parp2, CO1)[dual];
AND(A/0, B/0 -> p/0);
AND(A/0, B/1 -> parp0);
AND(A/1, B/0 -> parp1);
AND(A/1, B/1 -> parp2);
HA(parp0, parp1 -> p/1, CO1);
HA(CO1, parp2 -> p/2, p/3);
}
```

If parp0 and parp1 are present then p/1 and CO1 will flow.
If A/1 and B/0 are present then parp1 will flow.
Therefore, if A/1 and B/0 and parp0 are present at the new input boundary then p/1 and CO1 will flow

source component

```
AND(A/1, B/0 -> parp1);
flow [A/1, B/0] -> parp1;
close A/1 <- #parp1;
close B/0 <- #parp1;
close parp1 <- #?;
```

destination component

```
HA(parp0, parp1 -> p/1, CO1);
flow [parp0, parp1] -> [p/1, CO1];
close parp0 <- [#p/1,  #CO1];
close parp1 <- [#p/1, #CO1];
close p/1 <- #?;
close CO1 <- #?;
```

The boundaries are merged, minus parp1, to form a single boundary of a new network component

boundary composition

```
(A/1, B/0, parp0 -> p/1, CO1){
flow [A/1, B/0, parp0] -> [p/1, CO1];
        AND(A/1, B/0 -> parp1);
        HA(parp0, parp1 -> p/1, CO1);
}
close A/1 <- #parp1;
close B/0 <- #parp1;
osc parp1 <- [#p/1, #CO1];
close parp0 <- [#p/1, #CO1];
close p/1 <- #?;
close CO1 <- #?;
```

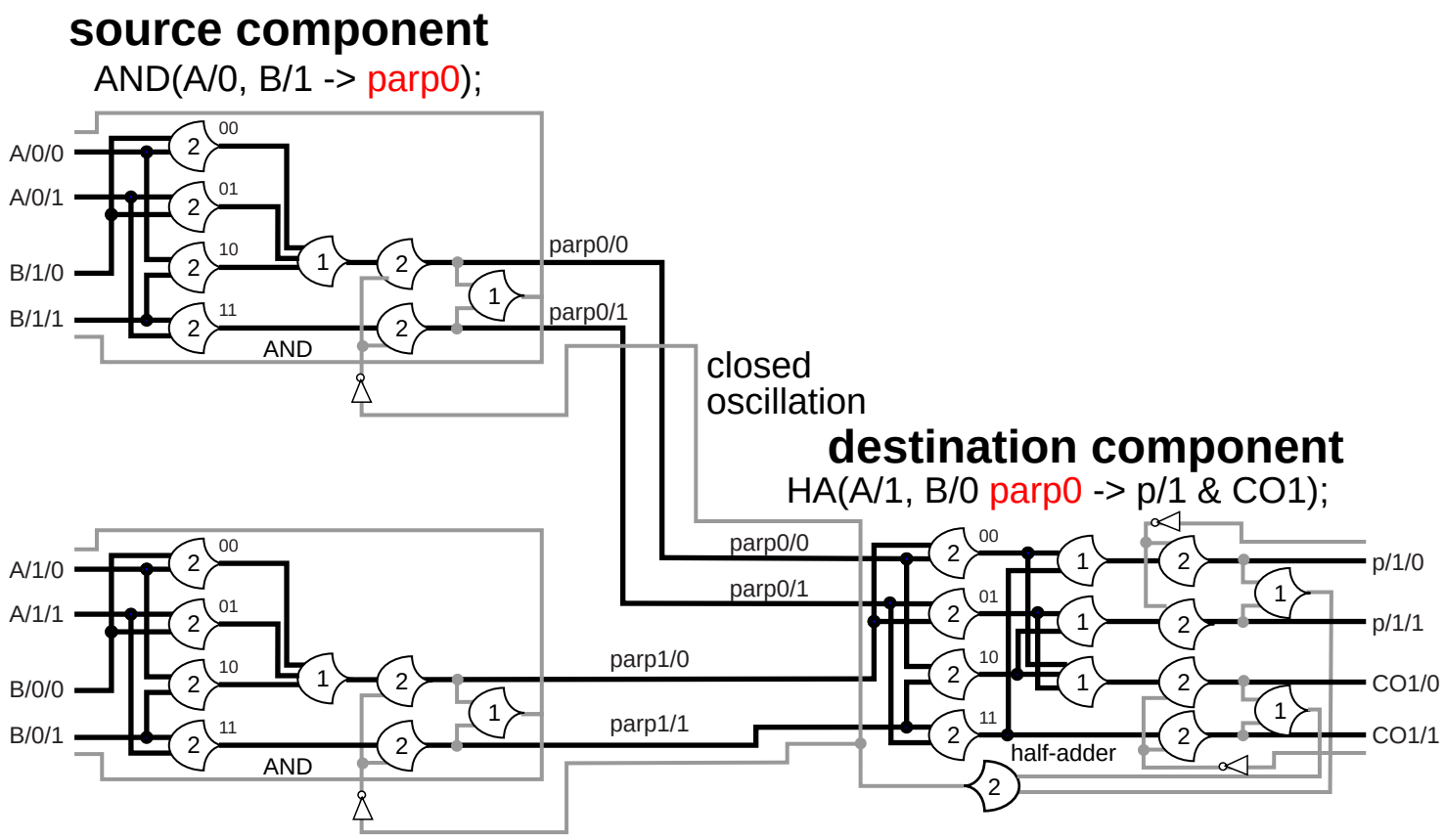
The AND flow relation [parp0, parp1] in the destination component propagates to the new boundary, minus parp1, as [A/1, B/0, parp0]

Closures are merged
parp1 becomes internal oscillation

Each component is certified as flow sufficient and the composition did not introduce any deadlock, race or behavior ambiguity **so the new network component is flow sufficient with the new boundary flow relation**

Connect parp0: The Composition

```
twobitmul(A, B -> p){
flow [A, B] -> p;
token dual{0,1};
token (A[0,1], B[0,1], p[0-3])[dual];
token (parp0, parp1, parp2, CO1)[dual];
AND(A/0, B/0 -> p/0);
AND(A/0, B/1 -> parp0);
AND(A/1, B/0 -> parp1);
AND(A/1, B/1 -> parp2);
HA(parp0, parp1 -> p/1, CO1);
HA(CO1, parp2 -> p/2, p/3);
}
```



Connect parp0: The Symbolic Analysis

```
twobitmul(A, B -> p){
flow [A, B] -> p;
token dual{0,1};
token (A[0,1], B[0,1], p[0-3])[dual];
token (parp0, parp1, parp2, CO1)[dual];
AND(A/0, B/0 -> p/0);
AND(A/0, B/1 -> parp0);
AND(A/1, B/0 -> parp1);
AND(A/1, B/1 -> parp2);
HA(parp0, parp1 -> p/1, CO1);
HA(CO1, parp2 -> p/2, p/3);
}
```

If A/1 and B/0 and parp0 are present then p/1 and CO1 will flow.
If A/0 and B/1 are present then parp0 will flow.
Therefore, if A/1 and B/0 and A/0 and B/1 are present at the new input boundary then p/1 and CO1 will flow

source component

```
AND(A/0, B/1 -> parp0);
flow [A/0, B/1] -> parp0;
close A/0 <- #parp0;
close B/1 <- #parp0;
close parp0 <- #?;
```

destination component

```
(A/1, B/0, parp0 -> p/1, CO1){
flow [A/1, B/0, parp0] -> [p/1, CO1];
        AND(A/1, B/0 -> parp1);
        HA(parp0, parp1 -> p/1, CO1);
}
close A/1 <- #parp1;
close B/0 <- #parp1;
close parp0 <- [# p/1, # CO1];
osc parp1 <- [#p/1, #CO1];
close p/1 <- #?;
close CO1 <- #?;
```

The boundaries are merged, minus parp0, to form a single boundary of a new network component

boundary composition

The AND flow relation [A/1, B/0, parp0] in the destination component propagates to the new boundary, minus parp0, as [A/1, B/0, A/0, B/1]

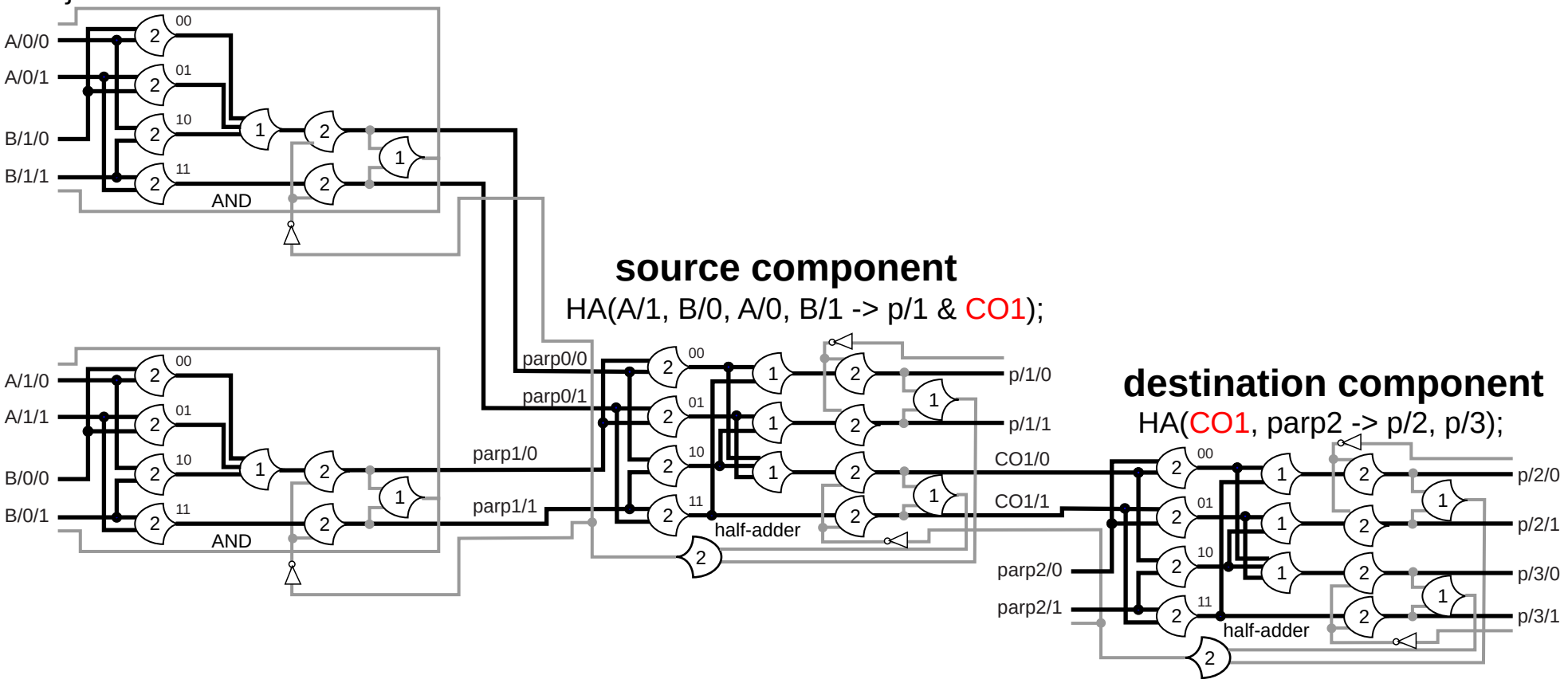
Closures are merged
parp0 becomes internal oscillation

```
(A/1, B/0, A/0, B/1 -> p/1, CO1){
flow [A/1, B/0, A/0, B/1] -> [p/1, CO1];
        AND(A/0, B/1 -> parp0);
        AND(A/1, B/0 -> parp1);
        HA(parp0, parp1 -> p/1, CO1);
}
close A/1 <- #parp1;
close B/0 <- #parp1;
close A/0 <- #parp0;
close B/1 <- #parp0;
osc parp1 <- [#p/1, #CO1];
osc parp0 <- [#p/1, #CO1];
close p/1 <- #?;
close CO1 <- #?;
```

Each component is certified as flow sufficient and the composition did not introduce any deadlock, race or behavior ambiguity **so the new network component is flow sufficient with the new boundary flow relation**

Connect CO1: The Composition

```
twobitmul(A, B -> p){
flow [A, B] -> p;
token dual{0,1};
token (A[0,1], B[0,1], p[0-3])[dual];
token (parp0, parp1, parp2, CO1)[dual];
AND(A/0, B/0 -> p/0);
AND(A/0, B/1 -> parp0);
AND(A/1, B/0 -> parp1);
AND(A/1, B/1 -> parp2);
HA(parp0, parp1 -> p/1, CO1);
HA(CO1, parp2 -> p/2, p/3);
}
```



Connect CO1: The Symbolic Analysis

```
twobitmul(A, B -> p){
flow [A, B] -> p;
token dual{0,1};
token (A[0,1], B[0,1], p[0-3])[dual];
token (parp0, parp1, parp2, CO1)[dual];
AND(A/0, B/0 -> p/0);
AND(A/0, B/1 -> parp0);
AND(A/1, B/0 -> parp1);
AND(A/1, B/1 -> parp2);
HA(parp0, parp1 -> p/1, CO1);
HA(CO1, parp2 -> p/2, p/3);
}
```

If CO1 and parp2 are present then p/2 and p/3 will flow.
If A/1 and B/0 and A/0 and B/1 are present then p/1 and CO1 will flow.
Therefore, if A/1 and B/0 and A/0 and B/1 and parp2 are present at the new input boundary then p/1 and p/2 and p/3 will flow

source component

```
(A/1, B/0, A/0, B/1 -> p/1, CO1){
flow [A/1, B/0, A/0, B/1] -> [p/1, CO1];
        AND(A/1, B/0 -> parp1);
        HA(parp0, parp1 -> p/1, CO1);
}
close A/1 <- #parp1;
close B/0 <- #parp1;
close A/0 <- #parp0;
close B/1 <- #parp0;
osc parp1 <- [#p/1, #CO1];
osc parp0 <- [#p/1, #CO1];
close p/1 <- #?;
close CO1 <- #?;
```

destination component

```
HA(CO1, parp2 -> p/2, p/3);
flow [CO1, parp2] -> [p/2, p/3];
close CO1 <- [#p/2,  #p/3];
close parp2 <- [#p/2, #p/3];
close p/2 <- #?;
close p/3 <- #?;
```

boundary composition

The boundaries are merged, minus CO1, to form a single boundary of a new network component

```
(A/1, B/0, A/0, B/1, parp2 -> p/1, p/2, p/3){
flow [A/1, B/0, A/0, B/1, parp2] -> [p/1, p/2, p/3];
        AND(A/0, B/1 -> parp0);
        AND(A/1, B/0 -> parp1);
        HA(parp0, parp1 -> p/1, CO1);
        HA(CO1, parp2 -> p/2, p/3);
}
close A/1 <- #parp1;
close B/0 <- #parp1;
close A/0 <- #parp0;
close B/1 <- #parp0;
close parp2 <- [#p/2, #p/3];
osc parp1 <- [#p/1, #CO1];
osc parp0 <- [#p/1, #CO1];
osc CO1 <- [#p/2, #p/3];
close p/1 <- #?;
close p/2 <- #?;
close p/3 <- #?;
```

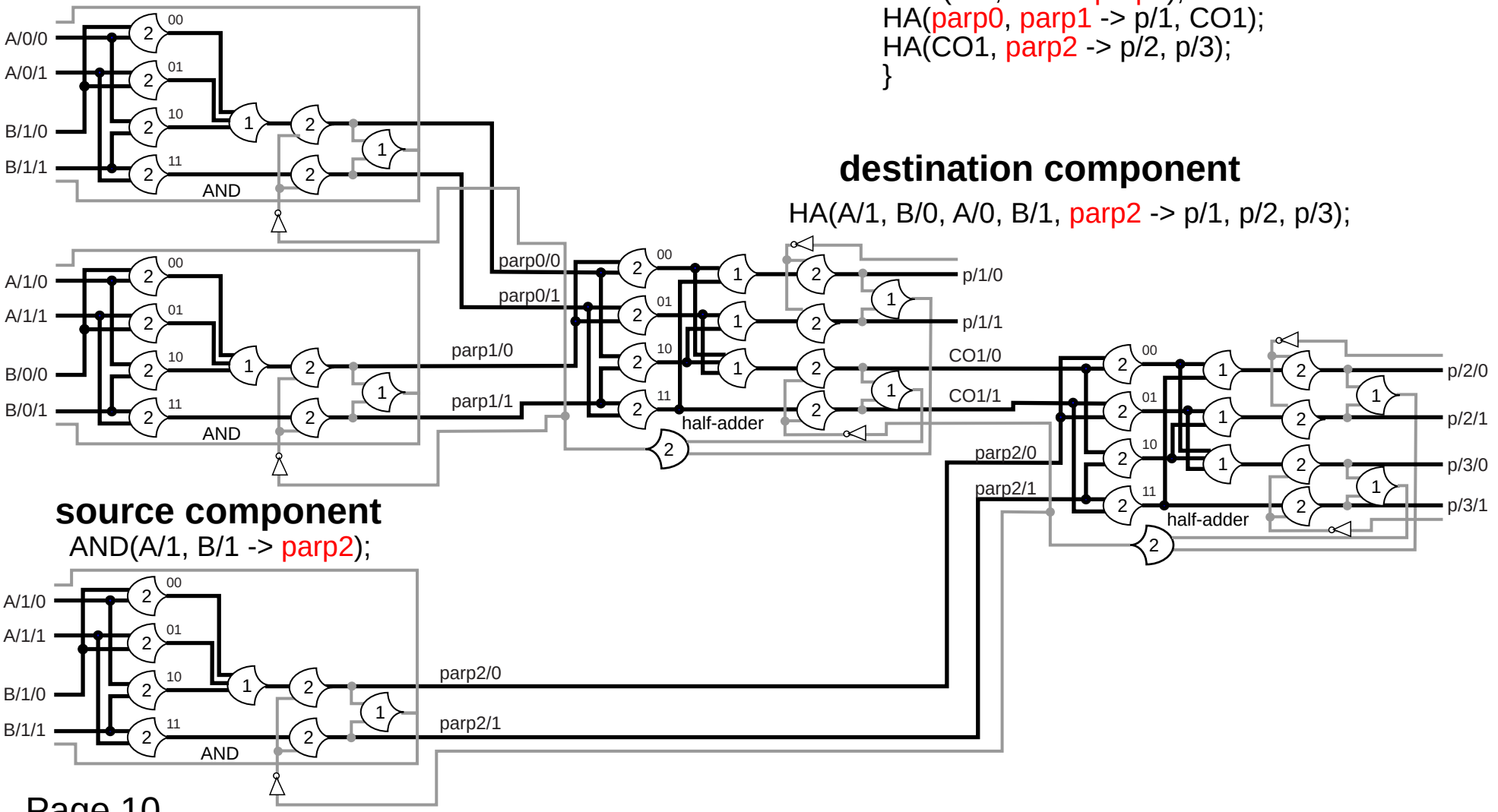
The AND flow relation [CO1, parp2] in the destination component propagates to the new boundary, minus CO1, as [A/1, B/0, A/0, B/1, parp2]

Closures are merged
CO1 becomes internal oscillation

Each component is certified as flow sufficient and the composition did not introduce any deadlock, race or behavior ambiguity **so the new network component is flow sufficient with the new boundary flow relation**

Connect parp2: The Composition

```
twobitmul(A, B -> p){
flow [A, B] -> p;
token dual{0,1};
token (A[0,1], B[0,1], p[0-3])[dual];
token (parp0, parp1, parp2, CO1)[dual];
AND(A/0, B/0 -> p/0);
AND(A/0, B/1 -> parp0);
AND(A/1, B/0 -> parp1);
AND(A/1, B/1 -> parp2);
HA(parp0, parp1 -> p/1, CO1);
HA(CO1, parp2 -> p/2, p/3);
}
```



destination component

HA(A/1, B/0, A/0, B/1, parp2 -> p/1, p/2, p/3);

source component

AND(A/1, B/1 -> parp2);

Connect parp2: The Symbolic Analysis

If A/1 and B/0 and A/0 and B/1 and parp2 are present then p/1 and p/2 and p/3 will flow.
If A/1 and B/1 are present then parp2 will flow.
Therefore, if A/1 and B/0 and A/0 and B/1 and A/1 and B/1 are present at the new input boundary then p/1 and p/2 and p/3 will flow

```
twobitmul(A, B -> p){
flow [A, B] -> p;
token dual{0,1};
token (A[0,1], B[0,1], p[0-3])[dual];
token (parp0, parp1, parp2, CO1)[dual];
AND(A/0, B/0 -> p/0);
AND(A/0, B/1 -> parp0);
AND(A/1, B/0 -> parp1);
AND(A/1, B/1 -> parp2);
HA(parp0, parp1 -> p/1, CO1);
HA(CO1, parp2 -> p/2, p/3);
}
```

destination component

```
(A/1, B/0, A/0, B/1, parp2 -> p/1, p/2, p/3){
flow [A/1, B/0, A/0, B/1, parp2] -> [p/1, p/2, p/3];
        AND(A/0, B/1 -> parp0);
        AND(A/1, B/0 -> parp1);
        HA(parp0, parp1 -> p/1, CO1);
        HA(CO1, parp2 -> p/2, p/3);
}
close A/1 <- #parp1;
close B/0 <- #parp1;
close A/0 <- #parp0;
close B/1 <- #parp0;
close parp2 <- [#p/2, #p/3];
osc parp1 <- [#p/1, #CO1];
osc parp0 <- [#p/1, #CO1];
osc CO1 <- [#p/2, #p/3];
close p/1 <- #?;
close p/2 <- #?;
close p/3 <- #?;
```

```
AND(A/1, B/1 -> parp2);
flow [A/1, B/1] -> parp2;
close A/1 <- #parp2;
close B/1 <- #parp2;
close parp2 <- #?;
```

boundary composition

```
(A/1, B/0, A/0, B/1, A/1, B/1 -> p/1, p/2, p/3){
flow [A/1, B/0, A/0, B/1, A/1, B/1] -> [p/1, p/2, p/3];
        AND(A/0, B/1 -> parp0);
        AND(A/1, B/0 -> parp1);
        AND(A/1, B/1 -> parp2);
        HA(parp0, parp1 -> p/1, CO1);
        HA(CO1, parp2 -> p/2, p/3);
}
close A/1 <- #parp1;
close B/0 <- #parp1;
close A/0 <- #parp0;
close B/1 <- #parp0;
close A/1 <- #parp2;
close B/1 <- #parp2;
osc parp1 <- [#p/1, #CO1];
osc parp0 <- [#p/1, #CO1];
osc CO1 <- [#p/2, #p/3];
osc parp2 <- [#p/2, #p/3];
close p/1 <- #?;
close p/2 <- #?;
close p/3 <- #?;
```

The boundaries are merged, minus parp2, to form a single boundary of a new network component

The AND flow relation [A/1, B/0, A/0, B/1, parp2] in the destination component propagates to the new boundary, minus parp2, as [A/1, B/0, A/0, B/1, A/1, B/1] -> [p/1, p/2, p/3]

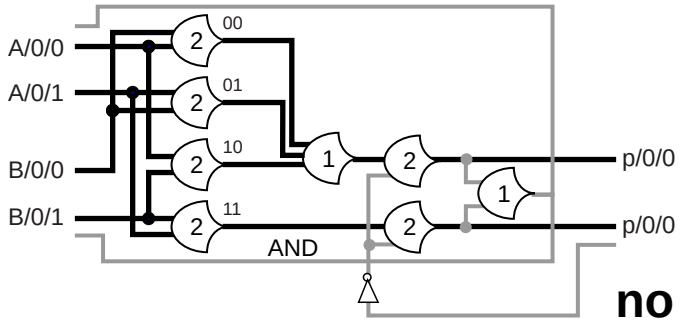
Closures are merged
parp2 becomes internal oscillation

Each component is certified as flow sufficient and the composition did not introduce any deadlock, race or behavior ambiguity **so the new network component is flow sufficient with the new boundary flow relation**

Add Unconnected: The Composition

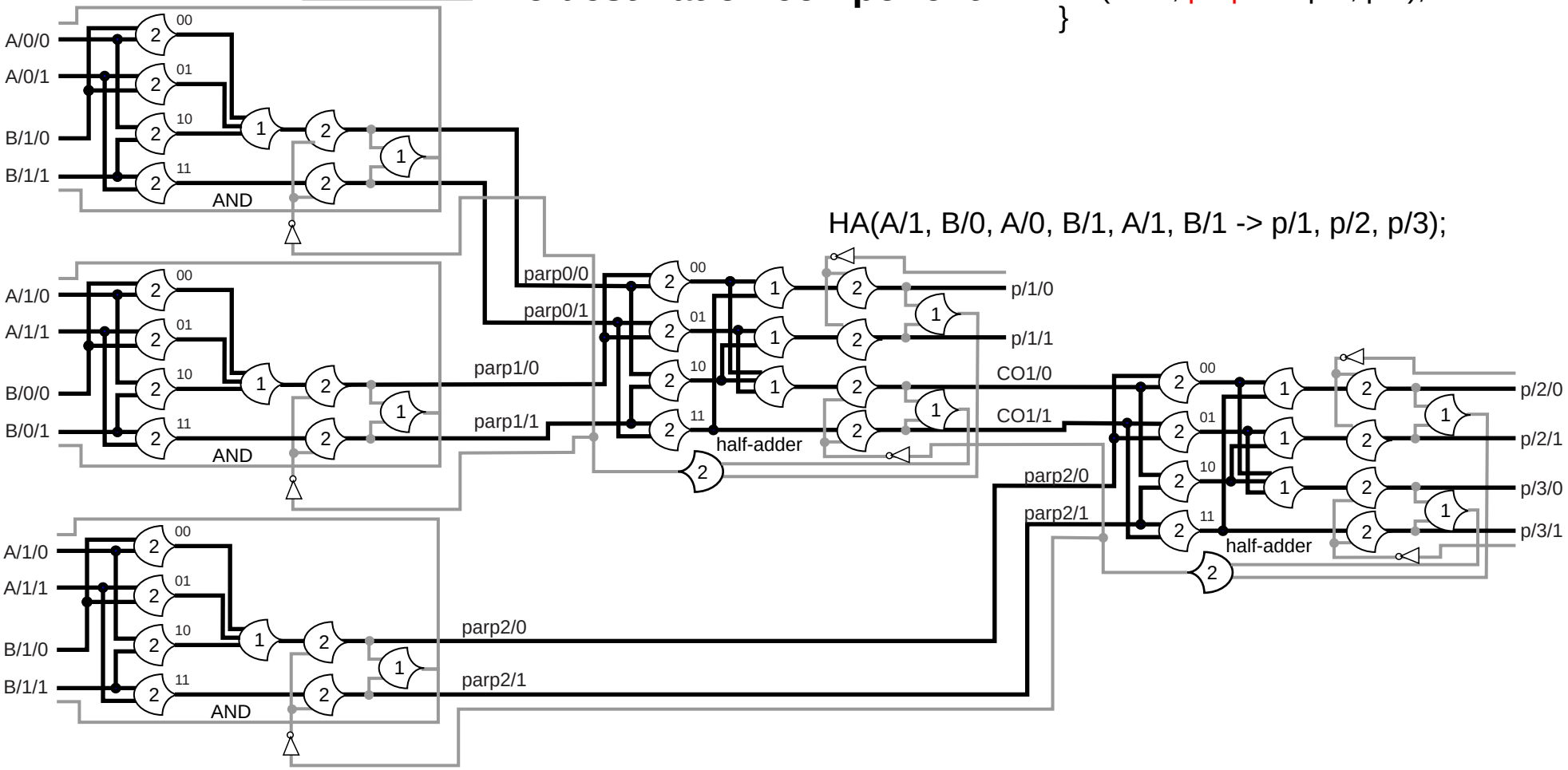
source component

AND(A/0, B/0 -> p/0);




```
twobitmul(A, B -> p){
flow [A, B] -> p;
token dual{0,1};
token (A[0,1], B[0,1], p[0-3])[dual];
token (parp0, parp1, parp2, CO1)[dual];
AND(A/0, B/0 -> p/0);
AND(A/0, B/1 -> parp0);
AND(A/1, B/0 -> parp1);
AND(A/1, B/1 -> parp2);
HA(parp0, parp1 -> p/1, CO1);
HA(CO1, parp2 -> p/2, p/3);
}
```

no destination component



Add Unconnected: The Symbolic Analysis

If A/1 and B/0 and A/0 and B/1 and A/1 and B/1 are present then p/1 and p/2 and p/3 will flow.
If A/0 and B/0 are present then p/0 will flow.
Therefore, if A/1 and B/0 and A/0 and B/1 and A/1 and B/1 and A/0 and B/0 are present at the new input boundary then p/0 and p/1 and p/2 and p/3 will flow

```
twobitmul(A, B -> p){
flow [A, B] -> p;
token dual{0,1};
token (A[0,1], B[0,1], p[0-3])[dual];
token (parp0, parp1, parp2, CO1)[dual];
AND(A/0, B/0 -> p/0);
AND(A/0, B/1 -> parp0);
AND(A/1, B/0 -> parp1);
AND(A/1, B/1 -> parp2);
HA(parp0, parp1 -> p/1, CO1);
HA(CO1, parp2 -> p/2, p/3);
}
```

source component

```
AND(A/0, B/0 -> p/0);
flow [A/0, B/0] -> p/0;
close A/0 <- #p/0;
close B/0 <- #p/0;
close p/0 <- #?;
```

destination component
none
does not connect internally

boundary composition

```
(A/1, B/0, A/0, B/1, A/1, B/1, A/0, B/0 -> p/0, p/1, p/2, p/3){
flow [A/1, B/0, A/0, B/1, A/1, B/1, A/0, B/0] -> [p/0, p/1, p/2, p/3];
        AND(A/0, B/0 -> p/0);
        AND(A/0, B/1 -> parp0);
        AND(A/1, B/0 -> parp1);
        AND(A/1, B/1 -> parp2);
        HA(parp0, parp1 -> p/1, CO1);
        HA(CO1, parp2 -> p/2, p/3);
}
close A/1 <- #parp1;
close B/0 <- #parp1;
close A/0 <- #parp0;
close B/1 <- #parp0;
close A/1 <- #parp2;
close B/1 <- #parp2;
close A/0 <- #p/0;
close B/0 <- #p/0;
osc parp1 <- [#p/1, #CO1];
osc parp0 <- [#p/1, #CO1];
osc CO1 <- [#p/2, #p/3];
osc parp2 <- [#p/2, #p/3];
close p/0 <- #?;
close p/1 <- #?;
close p/2 <- #?;
close p/3 <- #?;
```

The source component flows from input to output and does not connect internally so it just updates the boundary

The flows are assumed to be AND related unless otherwise indicated.

Closures are merged no internal oscillation

Each component is certified as flow sufficient and the composition did not introduce any deadlock, race or behavior ambiguity **so the new network component is flow sufficient with the new boundary flow relation**

The Final Fully Vetted Component: Symbolic

```
(A/1, B/0, A/0, B/1, A/1, B/1, A/0, B/0 -> p/0, p/1, p/2, p/3){
flow [A/1, B/0, A/0, B/1, A/1, B/1, A/0, B/0] -> [p/0, p/1, p/2, p/3];
        AND(A/0, B/0 -> p/0);
        AND(A/0, B/1 -> parp0);
        AND(A/1, B/0 -> parp1);
        AND(A/1, B/1 -> parp2);
        HA(parp0, parp1 -> p/1, CO1);
        HA(CO1, parp2 -> p/2, p/3);
}
close A/1 <- #parp1;
close B/0 <- #parp1;
close A/0 <- #parp0;
close B/1 <- #parp0;
close A/1 <- #parp2;
close B/1 <- #parp2;
close A/0 <- #p/0;
close B/0 <- #p/0;
osc parp1 <- [#p/1, #CO1];
osc parp0 <- [#p/1, #CO1];
osc CO1 <- [#p/2, #p/3];
osc parp2 <- [#p/2, #p/3];
close p/0 <- #?;
close p/1 <- #?;
close p/2 <- #?;
close p/3 <- #?;
```

Consolidate closures

Consolidate input tokens with fan-out assumption of identically named AND related tokens

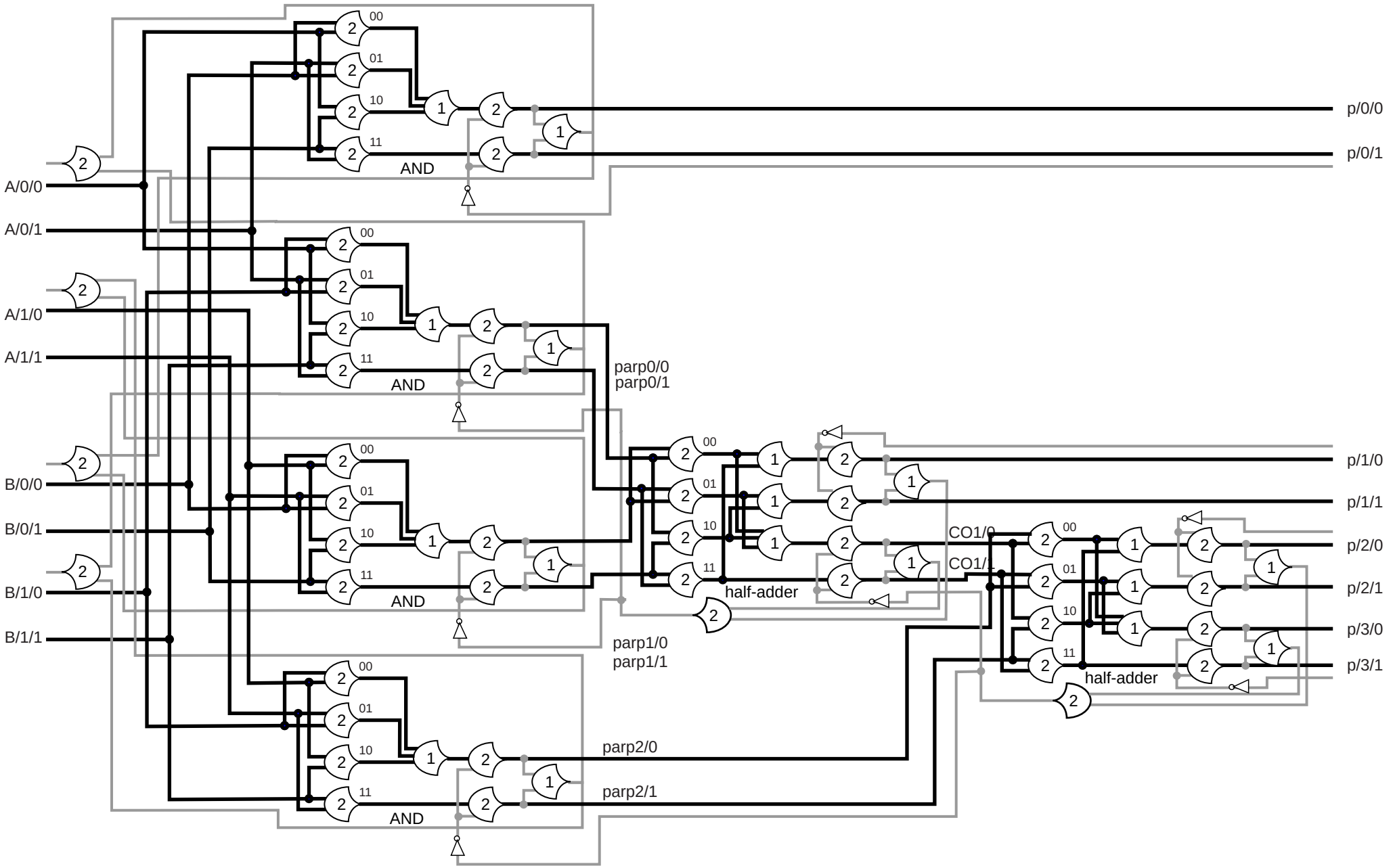
```
(A/0, A/1, B/0, B/1 -> p/0, p/1, p/2, p/3){
flow [A/0, A/1, B/0, B/1] -> [p/0, p/1, p/2, p/3];
        AND(A/0, B/0 -> p/0);
        AND(A/0, B/1 -> parp0);
        AND(A/1, B/0 -> parp1);
        AND(A/1, B/1 -> parp2);
        HA(parp0, parp1 -> p/1, CO1);
        HA(CO1, parp2 -> p/2, p/3);
}
close A/1 <- [#parp1, #parp2];
close B/0 <- [#parp1, #p/0];
close A/0 <- [#parp0, #p/0];
close B/1 <- [#parp0, #parp2];
osc parp1 <- [#p/1, #CO1];
osc parp0 <- [#p/1, #CO1];
osc CO1 <- [#p/2, #p/3];
osc parp2 <- [#p/2, #p/3];
close p/0 <- #?;
close p/1 <- #?;
close p/2 <- #?;
close p/3 <- #?;
```

From the declarations:
[A/0, A/1] is the same as A
[B/0, B/1] is B and
[p/0, p/1,p/2, p/3] is p

```
(A, B -> p){
flow [A, B] -> p;
        AND(A/0, B/0 -> p/0);
        AND(A/0, B/1 -> parp0);
        AND(A/1, B/0 -> parp1);
        AND(A/1, B/1 -> parp2);
        HA(parp0, parp1 -> p/1, CO1);
        HA(CO1, parp2 -> p/2, p/3);
}
close A/1 <- [#parp1, #parp2];
close B/0 <- [#parp1, #p/0];
close A/0 <- [#parp0, #p/0];
close B/1 <- [#parp0, #parp2];
osc parp1 <- [#p/1, #CO1];
osc parp0 <- [#p/1, #CO1];
osc CO1 <- [#p/2, #p/3];
osc parp2 <- [#p/2, #p/3];
close p/0 <- #?;
close p/1 <- #?;
close p/2 <- #?;
close p/3 <- #?;
```

We have reconstructed the original specification as a fully vetted flow network component ready to be further composed into a larger network .

The Final Network



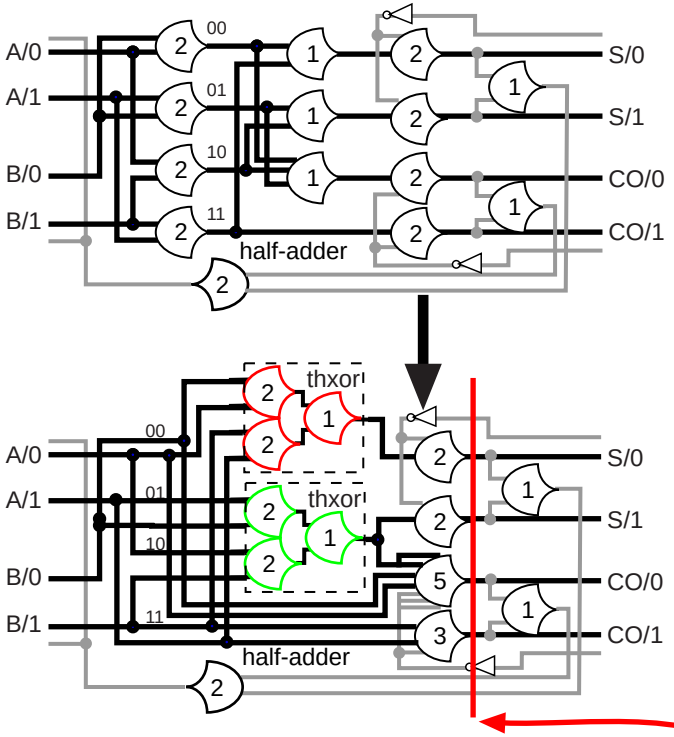
Using the Component Specification

From the composed 2 bit multiplier:

- Optimize component logic
- Structure the network of linked oscillations for optimal throughput
- Relax the granularity of oscillations: less area, lower throughput
- Optimize relaxed logic

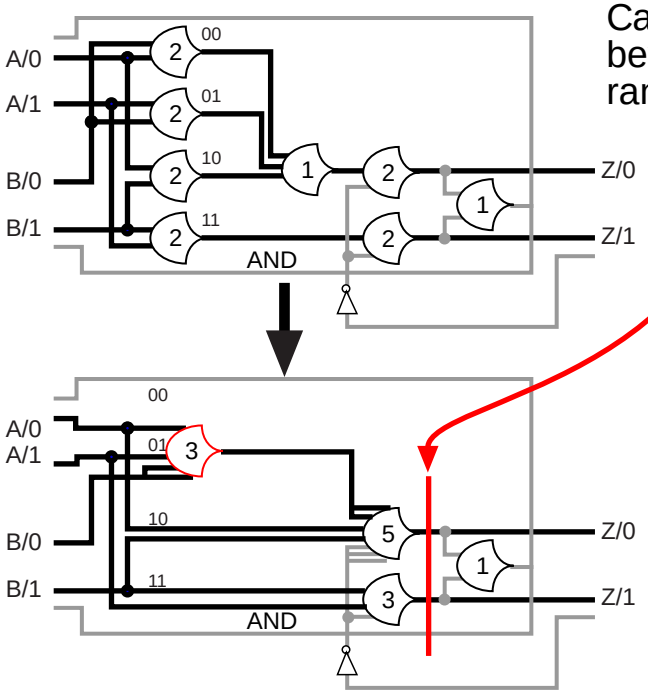
Optimizing the Primitive Component Logic

```
HA(A, B -> S, CO){   // half adder
token (A, B, S, CO){0,1};
flow [A, B] -> [S, CO];
[S/close, {[A/0, B/0], [A/1, B/1]}] -> S/0;
[S/close, {[A/0, B/1], [A/1, B/0]}] -> S/1;
[CO/close, {[A/0, B/0], [A/0, B/1], [A/1, B/0]}] -> CO/0;
[CO/close, [A/1, B/1]] -> CO/1;
}
close A <- [#S, CO];
close B <- [#S, CO];
close S <- #?;
close CO <- #?;
```

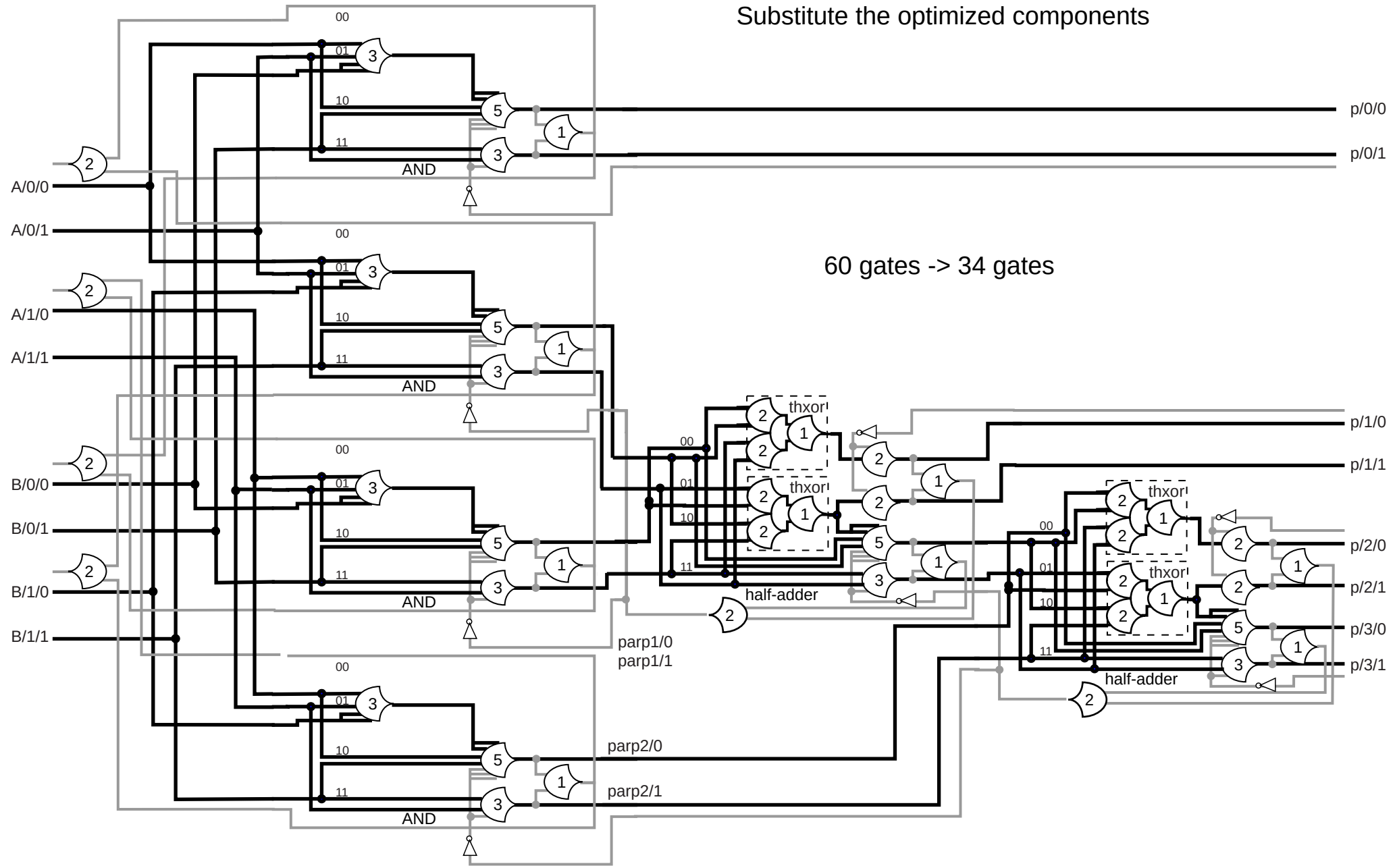


Can optimize into but not beyond the link enable rank

```
AND(A, B -> Z){   // AND
token (A, B, Z){0,1};
flow [A, B] -> Z;
[Z/close, {[A/0, B/0], [A/0, B/1], [A/1, B/0]}] -> Z/0;
[Z/close, [A/1, B/1]] -> Z/1;
}
close A <- #Z;
close B <- #Z;
close Z <- #?;
```



Optimized Flow Network Multiplier



Structure the Network for Optimal Flow
The Network Flow Grid

Tuning the structure of linked oscillations for optimal flow.

Minimize waiting

- Oscillations are roughly equal
- Linking structure is a regular grid

The flow grid

- Irregular grid derived from the dependency relationships of the linked oscillations
- Transform to a regular grid of linked oscillations

Grid numbers

All inputs are assigned an input grid number, in this case 0.

Grid numbers progress with connections

Network outputs are assigned the highest grid number.

Add buffers to fill in grid numbers

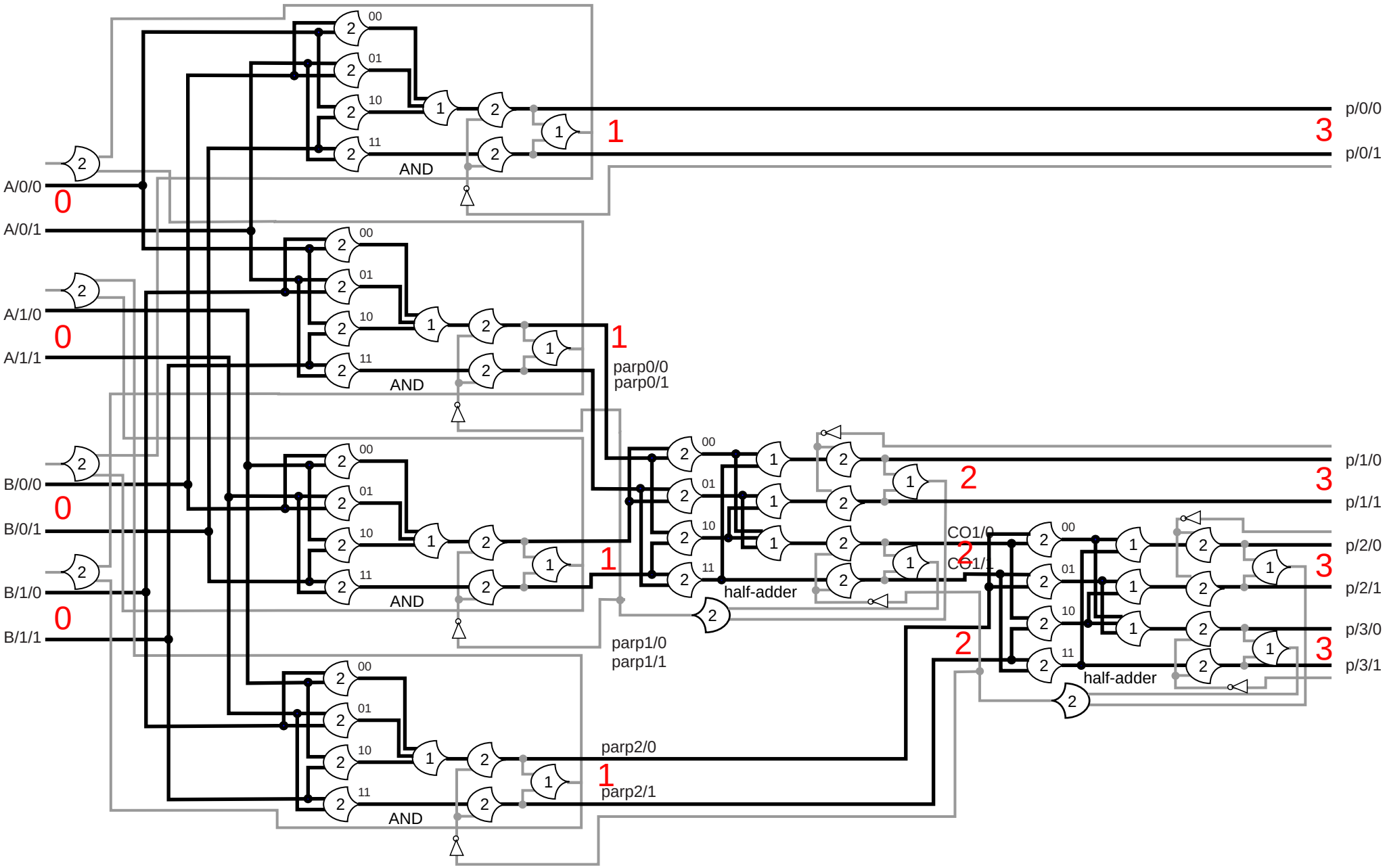
Multiplier specification with derived grid numbers

```
(A, B -> p){
flow [A, B] -> p;
(0)    AND(A/0, B/0 -> p/0);              (1)
(0)    AND(A/0, B/1 -> parp0);            (1)
(0)    AND(A/1, B/0 -> parp1);            (1)
(0)    AND(A/1, B/1 -> parp2);            (1)
(1)    HA(parp0, parp1 -> p/1, CO1);      (2)
(2)    HA(CO1, parp2 -> p/2, p/3);        (3)
}
close A/1 <- [#parp1, #parp2];
close B/0 <- [#parp1, #p/0];
close A/0 <- [#parp0, #p/0];
close B/1 <- [#parp0, #parp2];
osc parp1 <- [#p/1, #CO1];
osc parp0 <- [#p/1, #CO1];
osc CO1 <- [#p/2, #p/3];
osc parp2 <- [#p/2, #p/3];
close p/0 <- #?;
close p/1 <- #?;
close p/2 <- #?;
close p/3 <- #?;
```

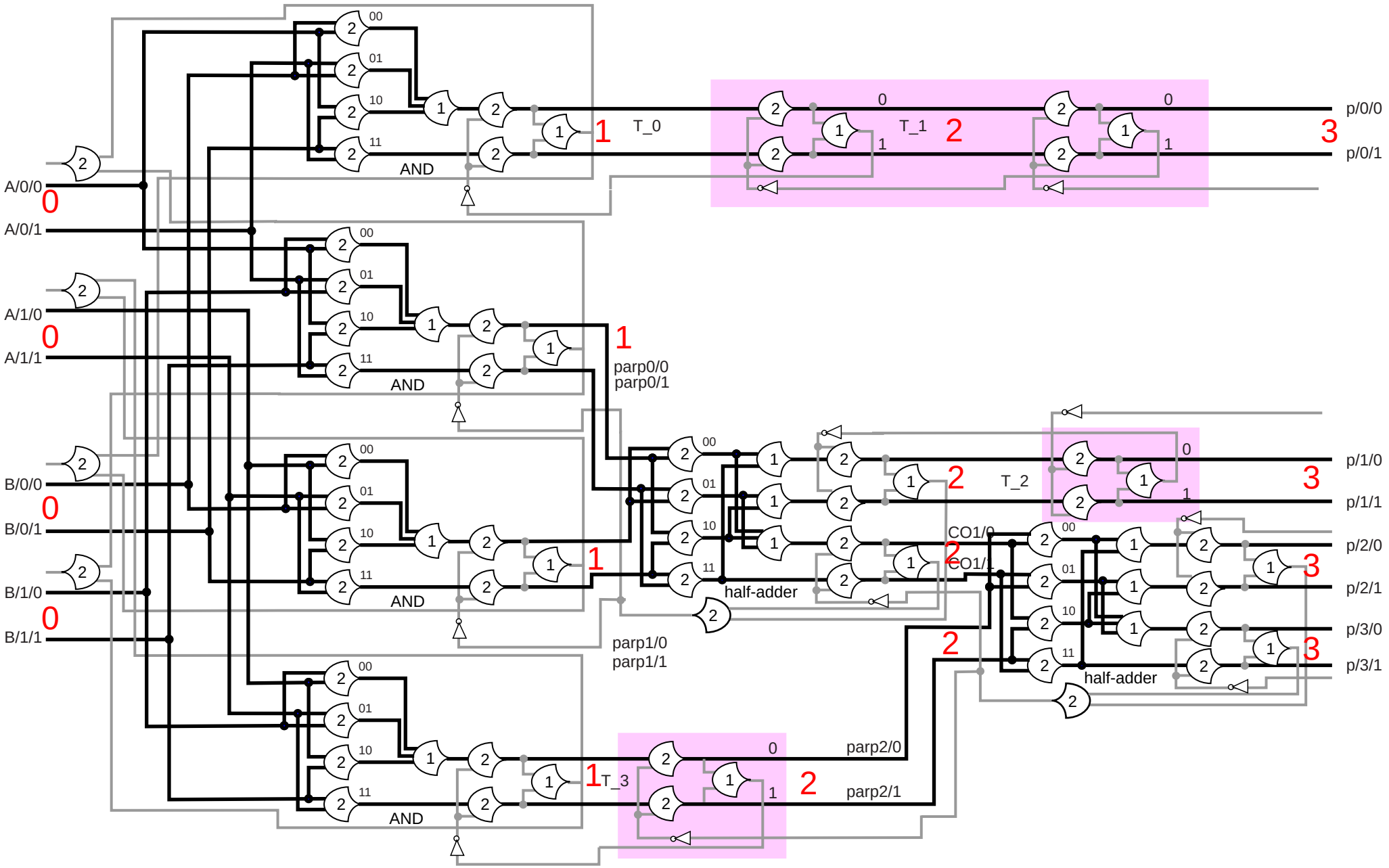
Add buffer components and update closures

```
(A, B -> p){
flow [A, B] -> p;
(0)    AND(A/0, B/0 -> T0);               (1)
(1)    (T0 -> T1);                        (2)
(2)    (T1 -> p/0);                       (3)
(0)    AND(A/0, B/1 -> parp0);            (1)
(0)    AND(A/1, B/0 -> parp1);            (1)
(0)    AND(A/1, B/1 -> T3);               (1)
(1)    (T3 -> PARP2);                     (2)
(1)    HA(parp0, parp1 -> T2, CO1);       (2)
(2)    (T2 -> p/1);                       (3)
(2)    HA(CO1, parp2 -> p/2, p/3);        (3)
}
close A/1 <- [#parp1, #T3];
close B/0 <- [#parp1, #T0];
close A/0 <- [#parp0, #T0];
close B/1 <- [#parp0, #T3];
osc parp1 <- [#T2, #CO1];
osc parp0 <- [#T2, #CO1];
osc CO1 <- [#p/2, #p/3];
osc parp2 <- [#p/2, #p/3];
osc T0 <- #T1];
osc T1 <-  #p/0];
osc T2 <-  #p/1];
osc T3 <- #parp2];
close p/0 <- #?;
close p/1 <- #?;
close p/2 <- #?;
close p/3 <- #?;
```

multiplier with flow field numbers

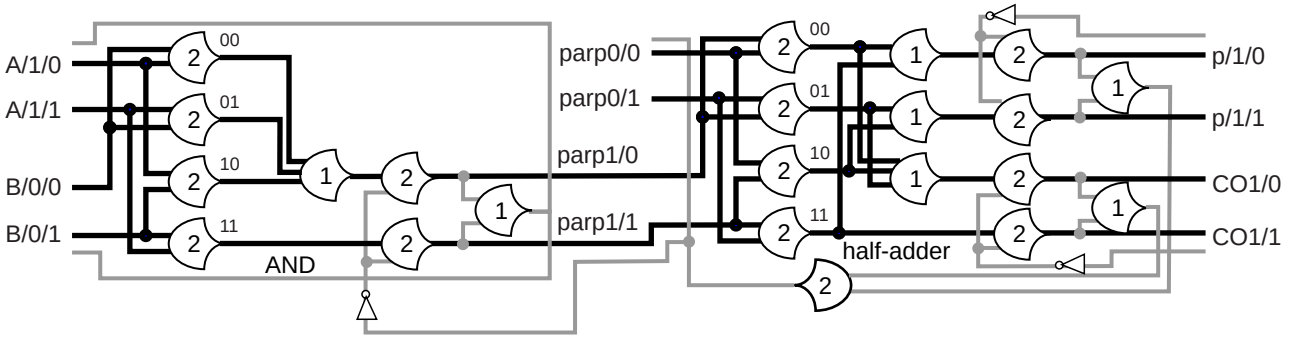


multiplier with buffers to complete flow field array



Relaxing Oscillation Links

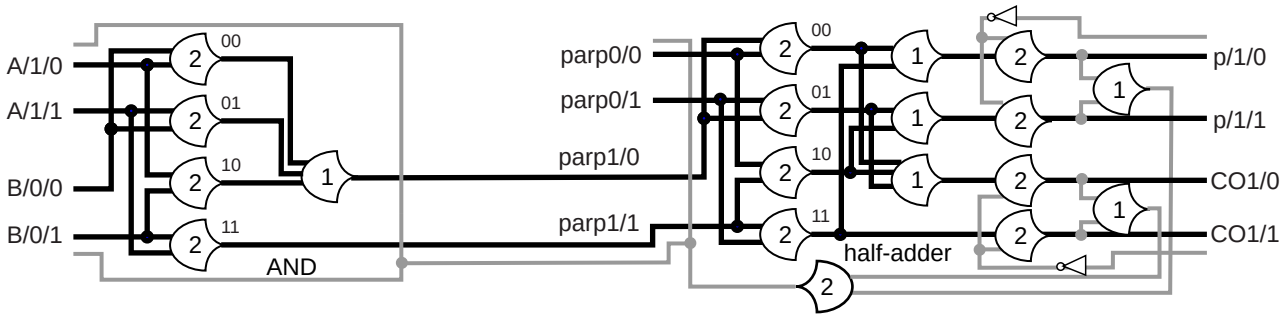
component with internal oscillation




```
(A/1, B/0, parp0 -> p/1, CO1){
flow [A/1, B/0, parp0] -> [p/1, CO1];
        AND(A/1, B/0 -> parp1);
        HA(parp0, parp1 -> p/1, CO1);
}
close A/1 <- #parp1;
close B/0 <- #parp1;
osc parp1 <- [#p/1, #CO1];
close parp0 <- [#p/1, #CO1];
close p/1 <- #?;
close CO1 <- #?;
```

the substitution removes a link and combines oscillations

removing link combines two oscillations




```
(A/1, B/0, parp0 -> p/1, CO1){
flow [A/1, B/0, parp0] -> [p/1, CO1];
        AND(A/1, B/0 -> parp1);
        HA(parp0, parp1 -> p/1, CO1);
}
close A/1 <- [#p/1, #CO1];
close B/0 <- [#p/1, #CO1];
close parp0 <- [#p/1, #CO1];
close p/1 <- #?;
close CO1 <- #?;
```

Merging the Exposed Equations

AND and HA are no longer components and no longer have boundaries so their equations are exposed into the body of the new component.

```
(A/1, B/0, parp0 -> p/1, CO1){
flow [A/1, B/0, parp0] -> [p/1, CO1];
        AND(A/1, B/0 -> parp1);
        HA(parp0, parp1 -> p/1, CO1);
}
close A/1 <- [#p/1, #CO1];
close B/0 <- [#p/1, #CO1];
close parp0 <- [#p/1, #CO1];
close p/1 <- #?;
close CO1 <- #?;
```

```
(A/1, B/0, parp0 -> p/1, CO1){
flow [A/1, B/0, parp0] -> [p/1, CO1];
{[A/1/0, B/0/0], [A/1/0, B/0/1], [A/1/1, B/1/0]} -> parp1/0;
[A/1/1, B/0/1] -> parp1/1;
{[parp0/0, parp1/0], [parp0/1, parp1/1]} -> p/1/0;
{[parp0/0, parp1/1], [parp0/1, parp1/0]} -> p/1/1;
{[parp0/0, parp1/0], [parp0/0, parp1/1], [parp0/1, parp1/0]} -> CO1/0;
[parp0/1, parp1/1] -> CO1/1;
}
close A/1 <- [#p/1, #CO1];
close B/0 <- [#p/1, #CO1];
close parp0 <- [#p/1, #CO1];
close p/1 <- #?;
close CO1 <- #?;
```

name substituted components

```
AND(A/1, B/0 -> parp1){
flow [A/1, B/0] -> parp1;
{[A/1/0, B/0/0], [A/1/0, B/0/1], [A/1/1, B/1/0]} -> parp1/0;
[A/1/1, B/0/1] -> parp1/1;
}
close A/1 <- #parp1;
close A/1 <- #parp1;
close B/0 <- #parp1;
close parp1 <- #?;
```

```
HA(parp0, parp1 -> p/1, CO1){
flow [parp0, parp1] -> [p/1, CO1];
{[parp0/0, parp1/0], [parp0/1, parp1/1]} -> p/1/0;
{[parp0/0, parp1/1], [parp0/1, parp1/0]} -> p/1/1;
{[parp0/0, parp1/0], [parp0/0, parp1/1], [parp0/1, parp1/0]} -> CO1/0;
[parp0/1, parp1/1] -> CO1/1;
}
close parp0 <- [#p/1,  #CO1];
close parp1 <- [#p/1, #CO1];
close p/1 <- #?;
close CO1 <- #?;
```

Relax the Links of Two Bit Mult

remove internal oscillation links and propagate their completeness to the boundary

closure equations from composition

```
close A/1 <- [#parp1, #parp2];
close B/0 <- [#parp1, #p/0];
close A/0 <- [#parp0, #p/0];
close B/1 <- [#parp0, #parp2];
osc parp1 <- [#p/1, #CO1];
osc parp0 <- [#p/1, #CO1];
osc CO1 <- [#p/2, #p/3];
osc parp2 <- [#p/2, #p/3];
close p/0 <- #?;
close p/1 <- #?;
close p/2 <- #?;
close p/3 <- #?;
```

substitute parp1 and parp0

```
close A/1 <- [#p/1, #CO1, #parp2];
close B/0 <- [#p/1, #CO1, #p/0];
close A/0 <- [#p/1, #CO1, #p/0];
close B/1 <- [#p/1, #CO1, #parp2];
osc CO1 <- [#p/2, #p/3];
osc parp2 <- [#p/2, #p/3];
close p/0 <- #?;
close p/1 <- #?;
close p/2 <- #?;
close p/3 <- #?;
```

substitute parp2 and CO1

```
close A/1 <- [#p/1, #p/2, #p/3, #p/2, #p/3];
close B/0 <- [#p/1, #p/2, #p/3, #p/0];
close A/0 <- [#p/1, #p/2, #p/3, #p/0];
close B/1 <- [#p/1, #p/2, #p/3, #p/2, #p/3];
close p/0 <- #?;
close p/1 <- #?;
close p/2 <- #?;
close p/3 <- #?;
```

Consoldate the Boundary

consolidate closure of input tokens

```
close B <- [#p/1, #p/2, #p/3, #p/0, #p/1, #p/2, #p/3, #p/2, #p/3];
close A <- [#p/1, #p/2, #p/3, #p/0, #p/1, #p/2, #p/3, #p/2, #p/3];
close p <- #?;
```

remove redundancies

```
close B <- [#p/0, #p/1, #p/2, #p/3];
close A <- [#p/0, #p/1, #p/2, #p/3];
close p <- #?;
```

scale token reference: update boundary specification, flow relation and closure equations

```
(A, B -> p){
flow [A, B] -> p;
        AND(A/0, B/0 -> p/0);
        AND(A/0, B/1 -> parp0);
        AND(A/1, B/0 -> parp1);
        AND(A/1, B/1 -> parp2);
        HA(parp0, parp1 -> p/1, CO1);
        HA(CO1, parp2 -> p/2, p/3);
}
close B <- #p;
close A <- #p;
close p <- #?;
```

Merge the Exposed Equations of Two Bit Mult

AND and HA are no longer components and no longer have boundaries so their equations are exposed.

```
(A, B -> p){
flow [A, B] -> p;
        AND(A/0, B/0 -> p/0);
        AND(A/0, B/1 -> parp0);
        AND(A/1, B/0 -> parp1);
        AND(A/1, B/1 -> parp2);
        HA(parp0, parp1 -> p/1, CO1);
        HA(CO1, parp2 -> p/2, p/3);
}
close B <- #p;
close A <- #p;
close p <- #?;
```

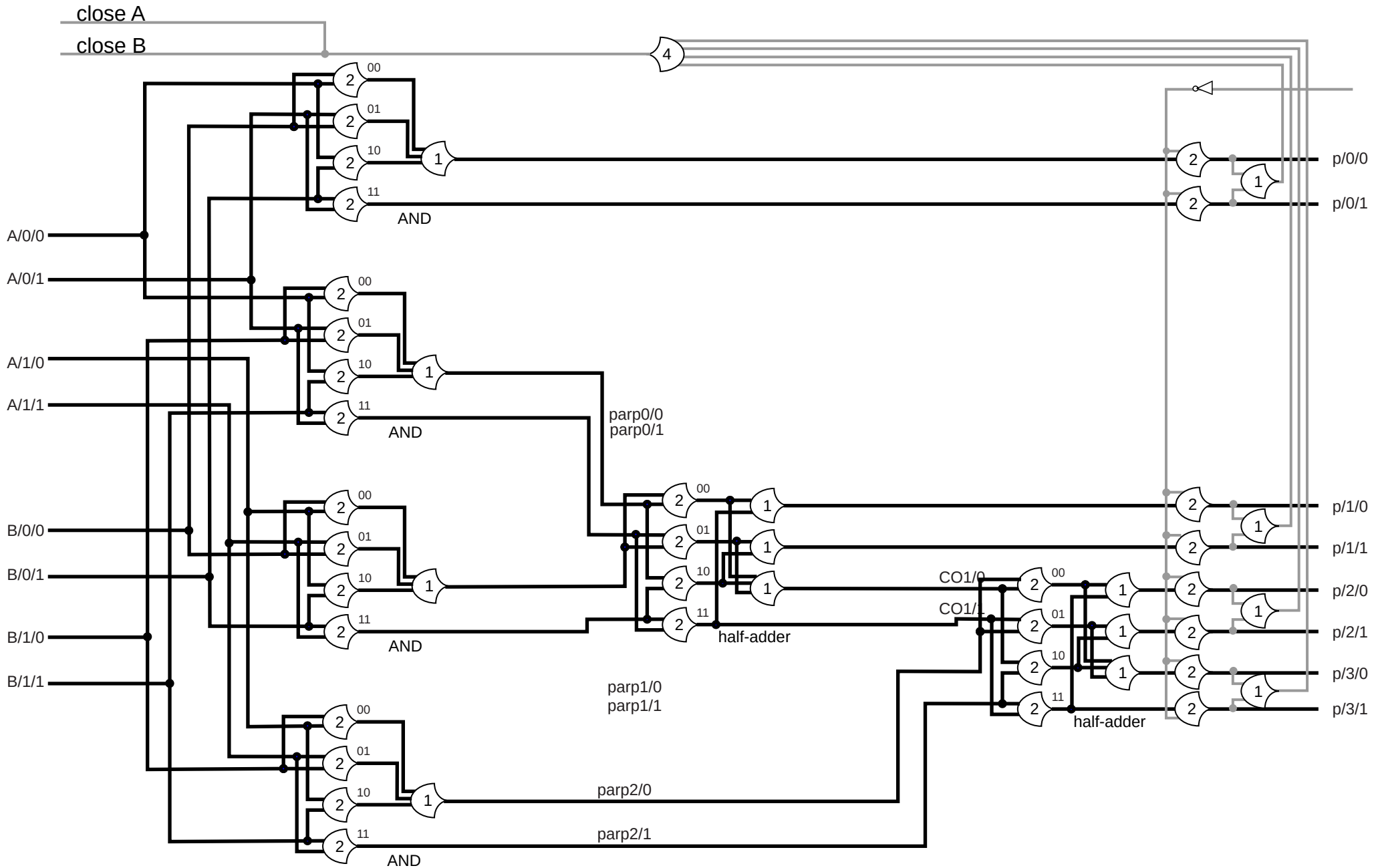
component with exposed equations with appropriate token name substitution

```
twobitmul(A, B -> p){
flow [A, B] -> p;
token dual{0,1};
token (A[0,1], B[0,1], p[0-3])[dual];
token (parp0, parp1, parp2, CO1)[dual];
{[A/0/0, B/0/0], [A/0/0, B/0/1], [A/0/1, B/0/0]} -> p/0/0;
[A/0/1, B/0/1] -> p/0/1;
{[A/0/0, B/1/0], [A/0/0, B/1/1], [A/0/1, B/1/0]} -> parp0/0;
[A/0/1, B/1/1] -> parp0/1;
{[A/1/0, B/0/0], [A/1/0, B/0/1], [A/1/1, B/1/0]} -> parp1/0;
[A/1/1, B/0/1] -> parp1/1;
{[A/1/0, B/1/0], [A/1/0, B/1/1], [A/1/1, B/1/0]} -> parp2/0;
[A/1/1, B/1/1] -> parp2/1;
{[parp0/0, parp1/0], [parp0/1, parp1/1]} -> p/1/0;
{[parp0/0, parp1/1], [parp0/1, parp1/0]} -> p/1/1;
{[parp0/0, parp1/0], [parp0/0, parp1/1], [parp0/1, parp1/0]} -> CO1/0;
[parp0/1, parp1/1] -> CO1/1;
{[CO1/0, parp2/0], [CO1/1, parp2/1]} -> p/2/0;
{[CO1/0, parp2/1], [CO1/1, parp2/0]} -> p/2/1;
{[CO1/0, parp2/0], [CO1/0, parp2/1], [CO1/1, parp2/0]} -> p/3/0;
[CO1/1, parp2/1] -> p/3/1;
}
close B <- #p;
close A <- #p];
close p <- #?;
```

The program is vetted for linked oscillation structure it is also vetted for relaxed structure. The equations of each component fulfil the completeness criterion just like each thershold function does and the equations of the components can be combined just like threshold gates.

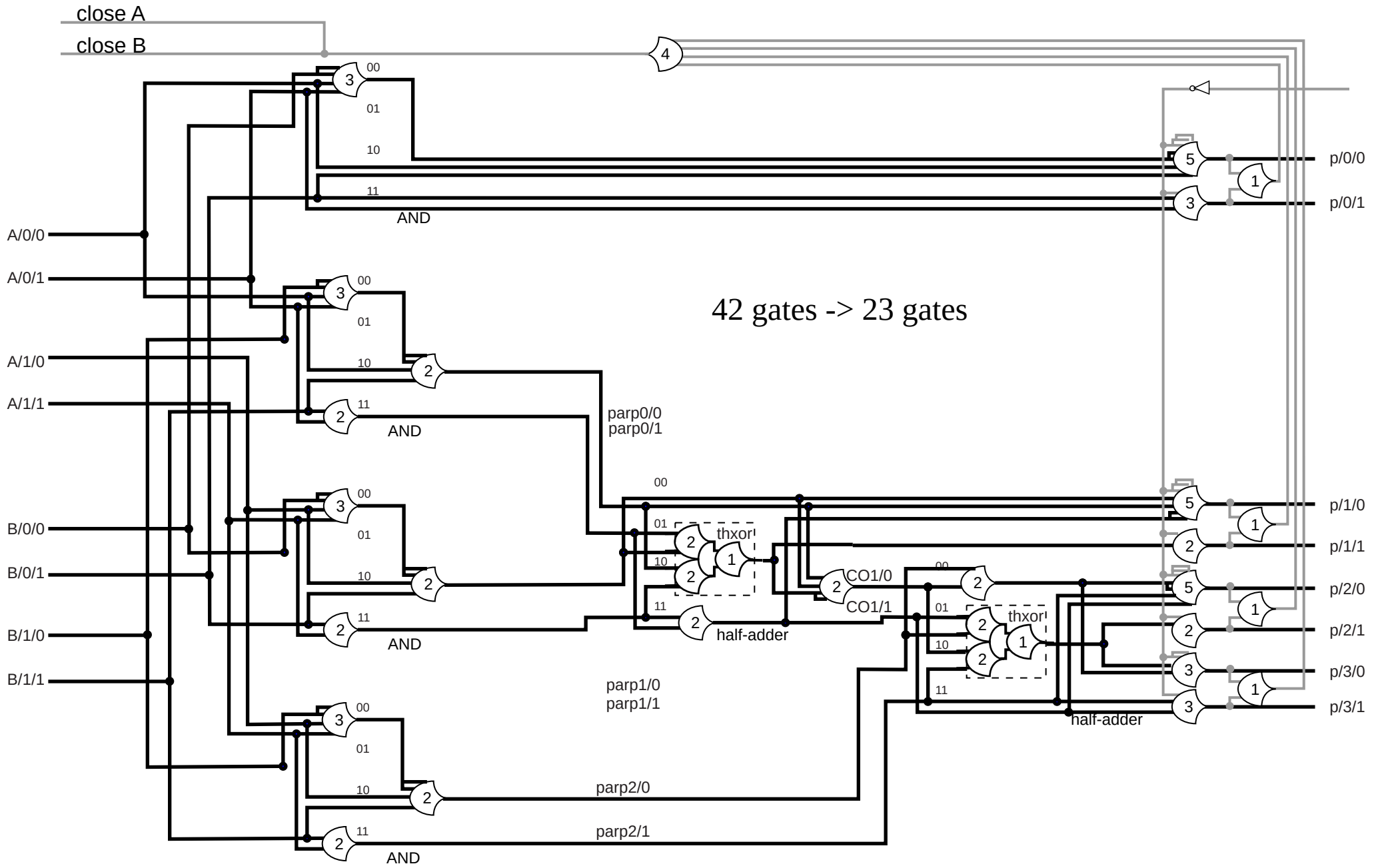
The Fully Relaxed 2 Bit Multiplier Component

The fully relaxed circuit remains a component with a boundary of half oscillations with internal completeness behavior coupling the oscillations



Optimizing the Relaxed 2 Bit Multiplier Component

The combined equations are iin a single component and can now be optimized all together.



Optimizing the Relaxed 2 Bit Multiplier Component