Sandbox 5 32 bit Adders

5 adders

fulladdA..... Textbook adder
fulladdB..... Adder that enables null wavefront carry play ahead
fulladdC..... Canonical adder mapped directly from canonical equations
fulladdD..... Halfadd, halfadd, OR - to illustrate combinational oscillation networks
fulladdQ..... quaternary (radix 4, 4-rail) full adder

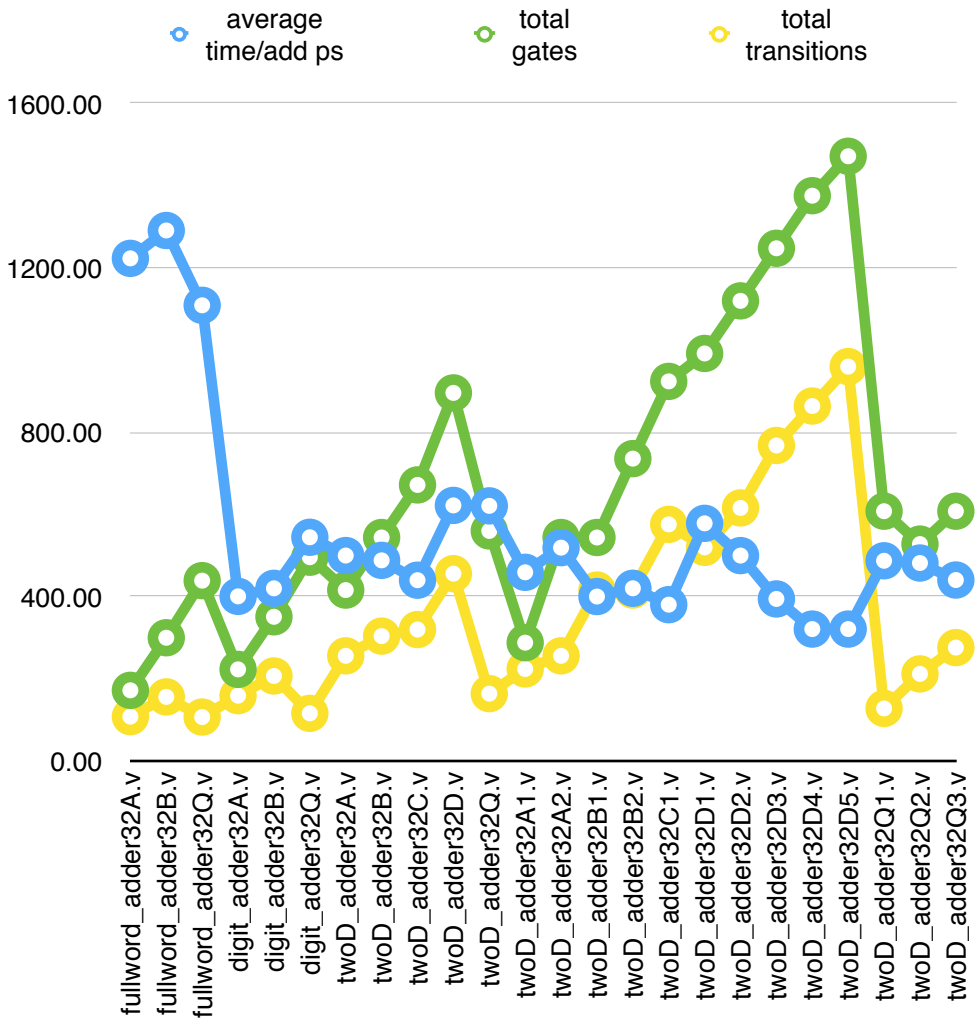
4 flow path structures

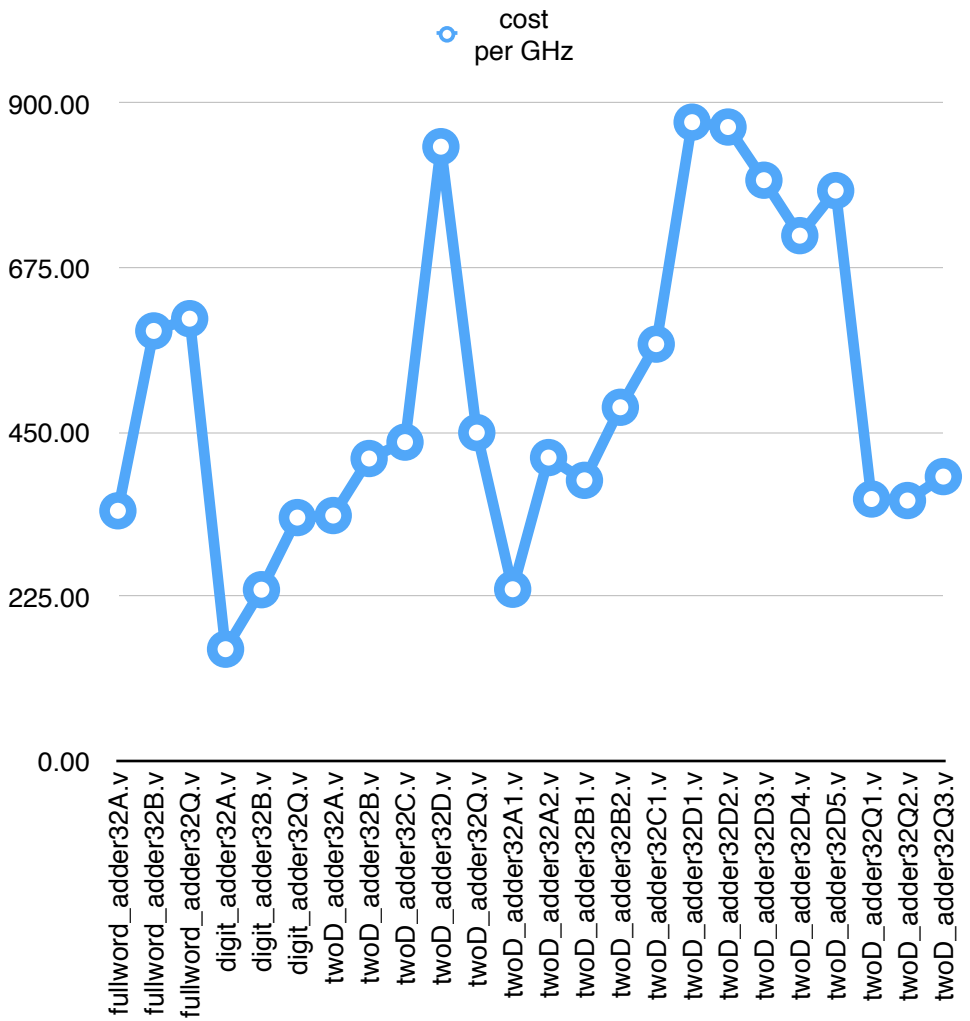
full word completeness..... what you are used to with a clock
two D pipelined..... free flowing digits with numbers separated logically by null
digit pipelined..... optimized two D pipelining
fully integrated..... every gate is flow control, some also do combinational duty

1 composition example

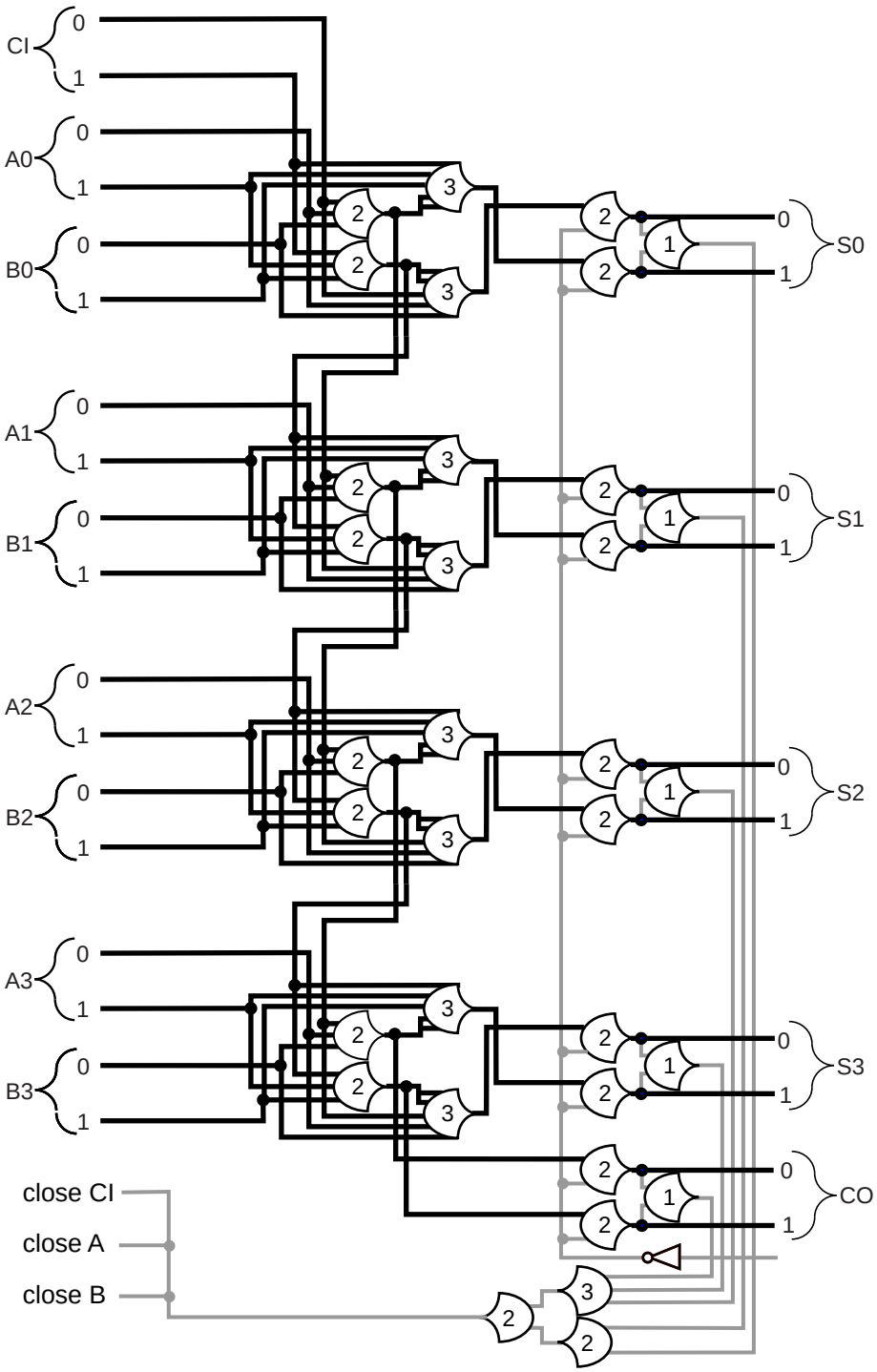
32 bit adder fed by 2 32 bit counters

Graphs of the statistics of the various adders and configurations from the spreadsheet.



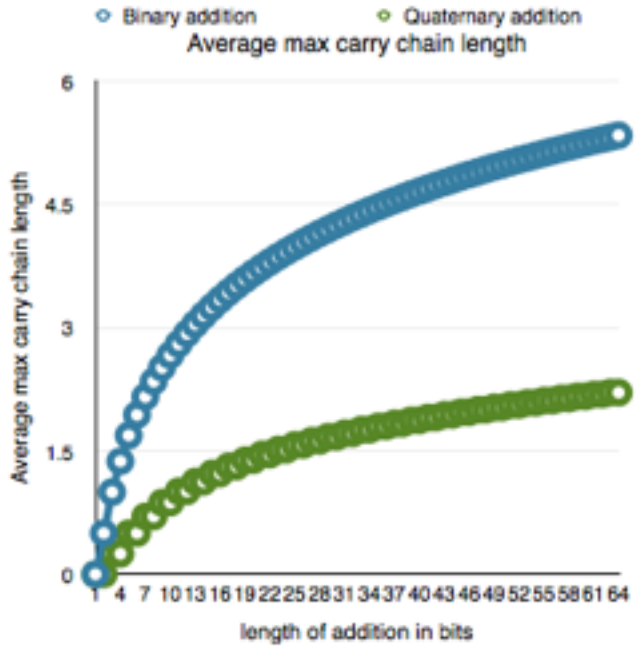


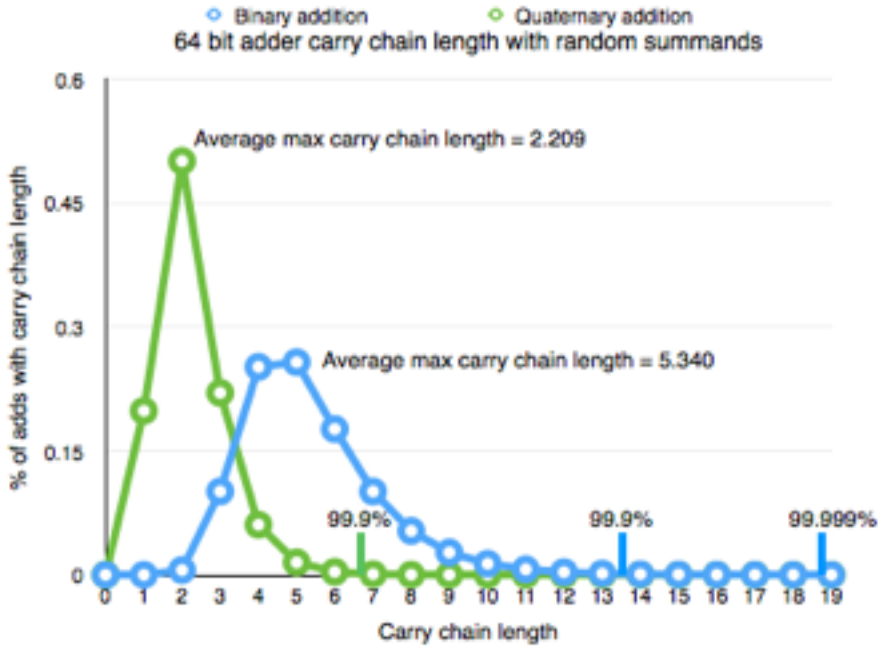
Full Word Completeness

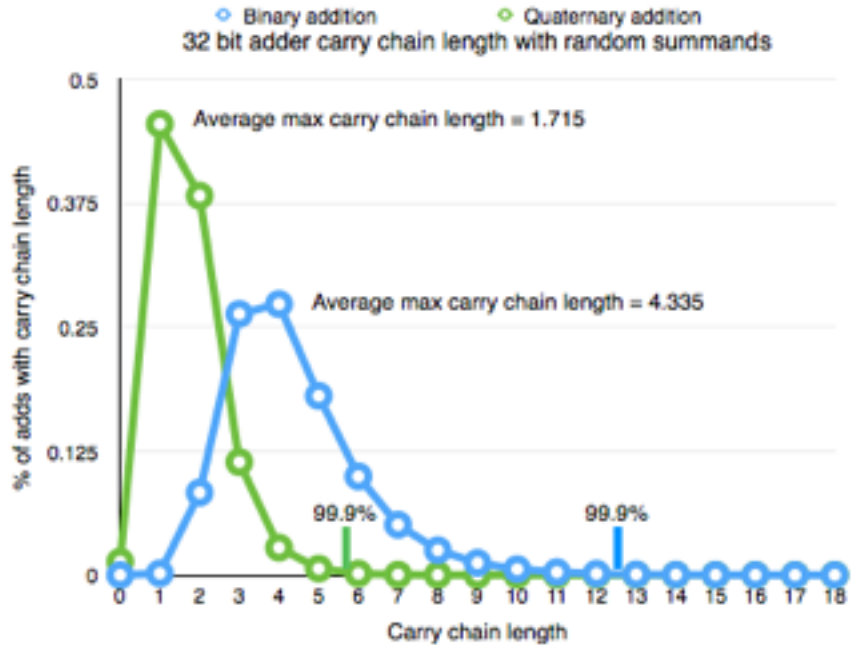


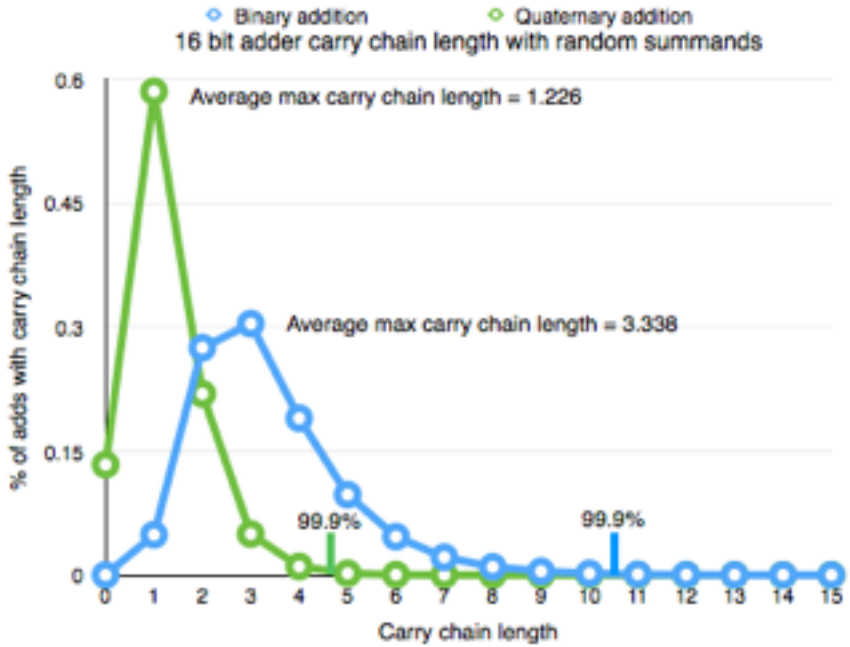
Average Carry Chain Analysis for NCL Adders

No need to waste time waiting on worst case behavior (critical path)
or to waste logic speeding up worst case behavior. (carry look ahead)



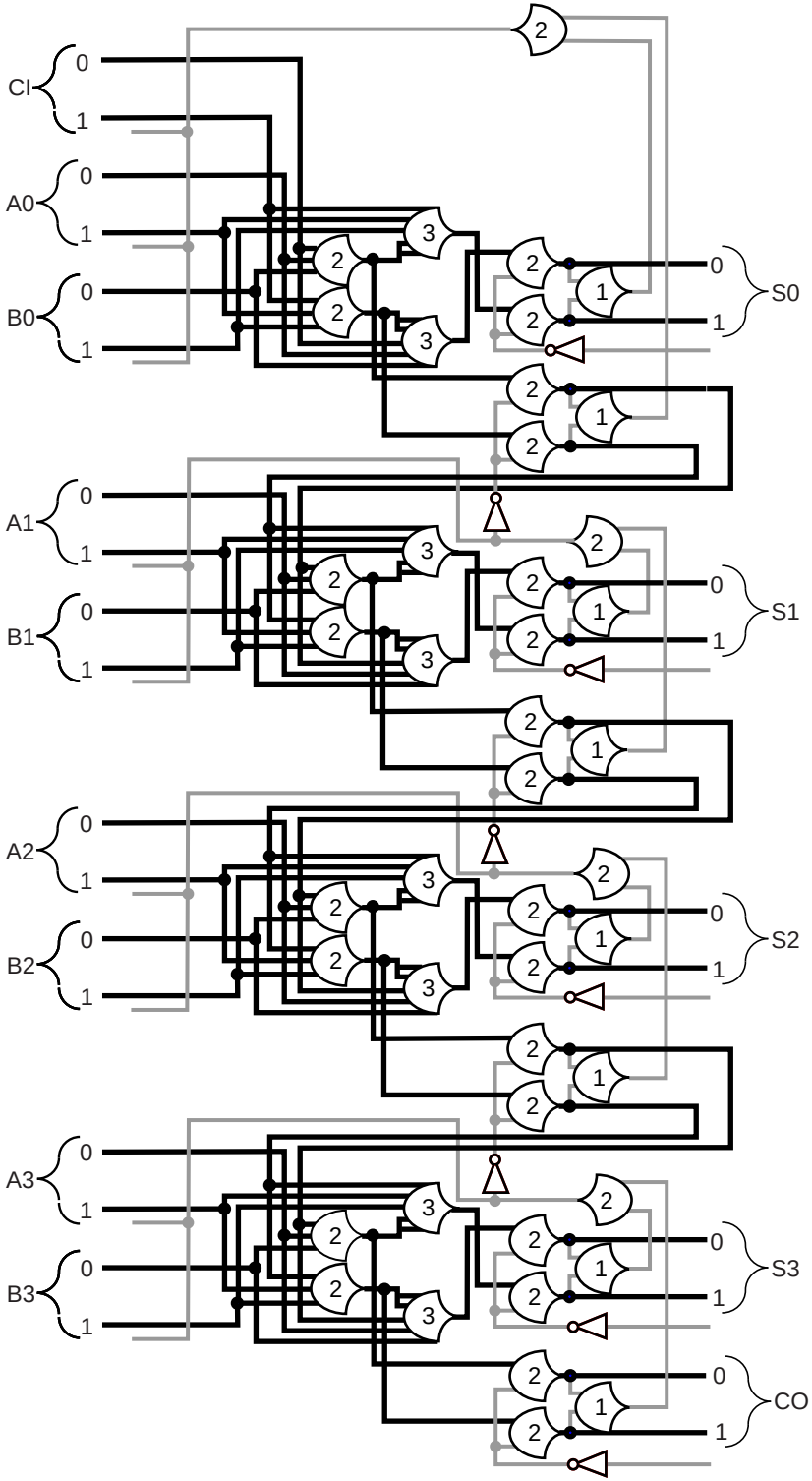




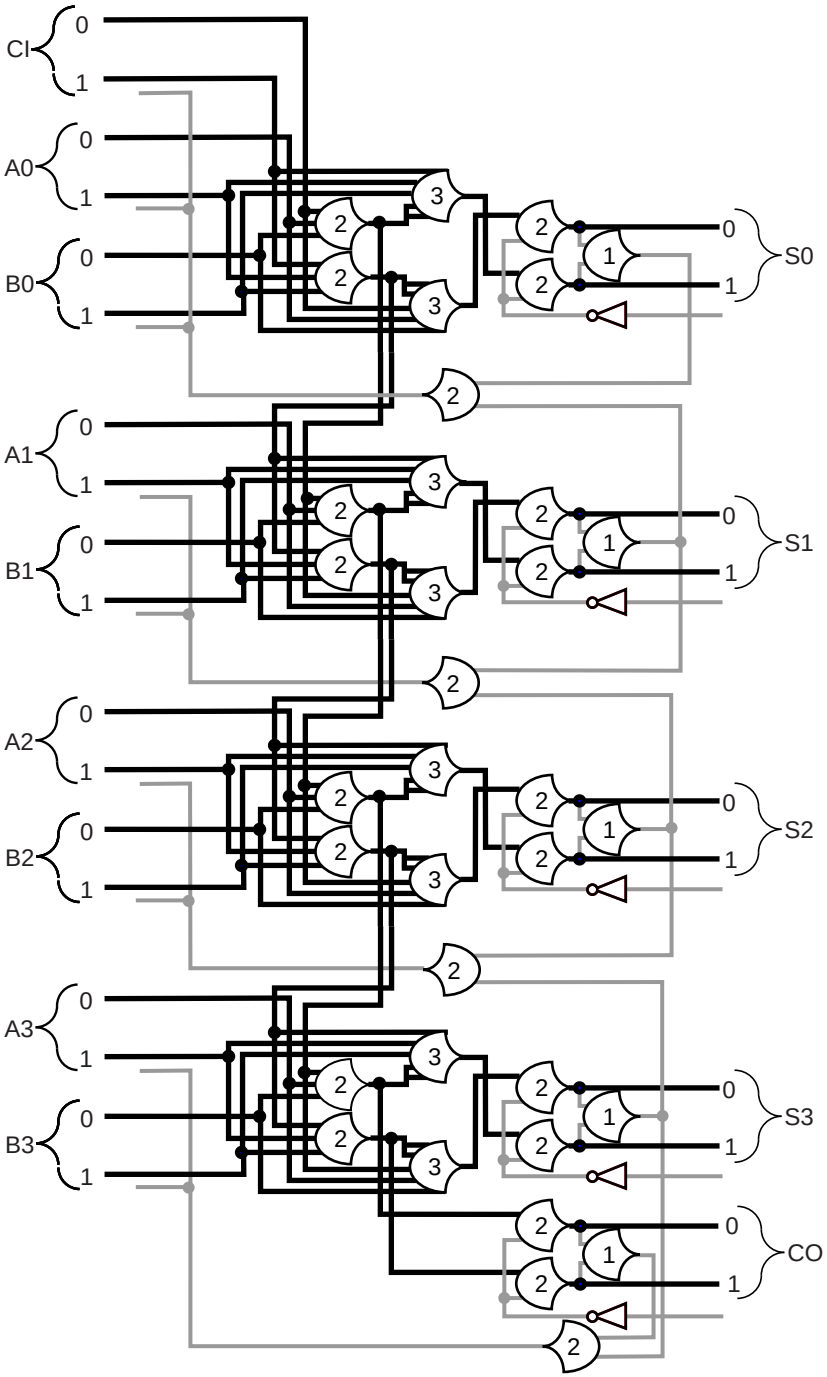


A. W. Burks, H. H. Goldstine, J. von Neumann, "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument," Institute for Advanced Study, Princeton, NJ, 1946.

2D Pipelined

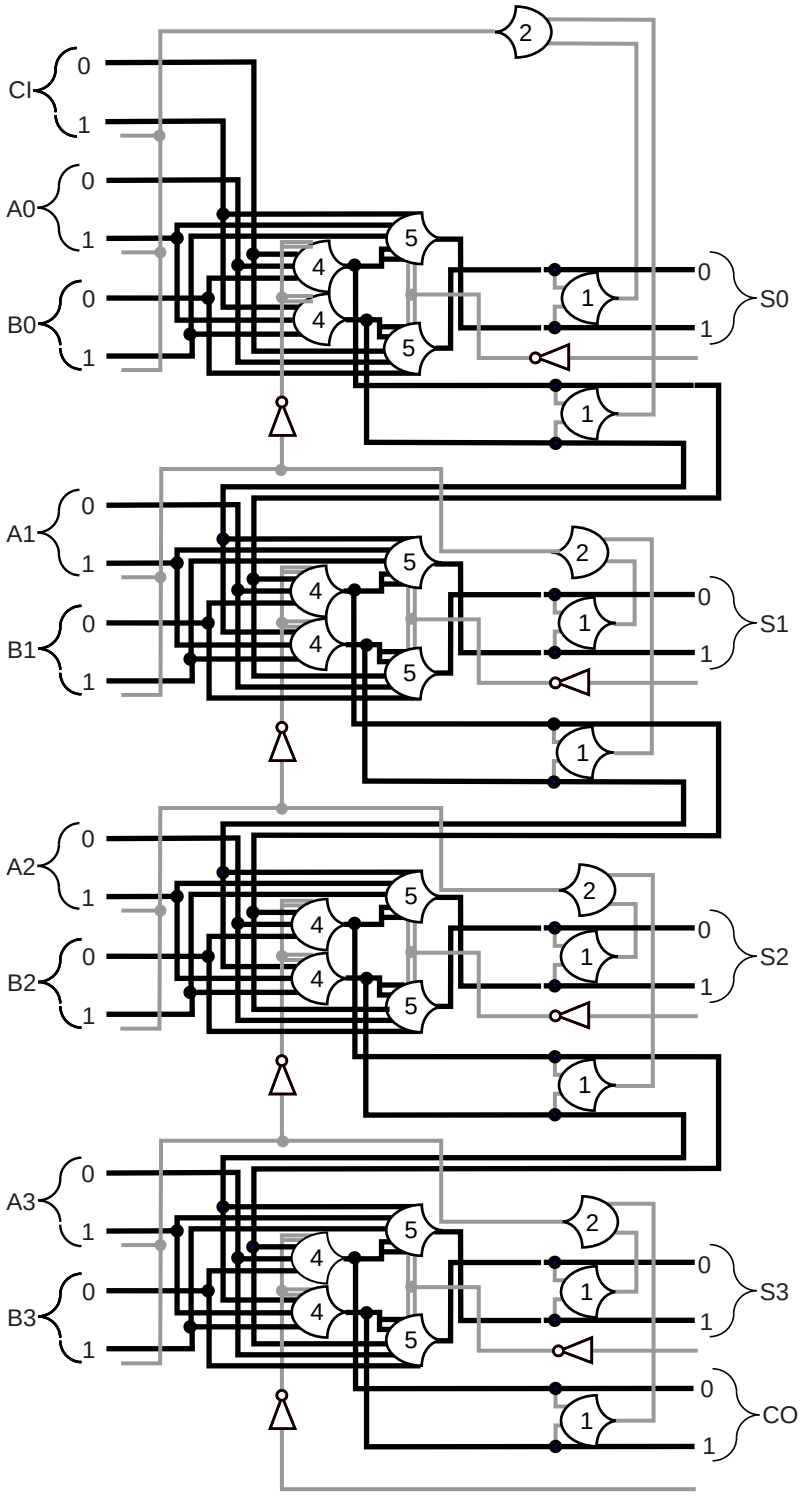


Digit Completeness



The carry is pipelined indirectly by sharing a completeness behavior

Integrated Completeness



The enable ranks of the links are integrated with the combinational logic

The binary fulladd Component - the sum and carry equations

The equations

carryin/ \ [A/ , B/]	00	01	10	11
0	0	1	1	0
1	1	0	0	1

sum/

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

carryin/ \ [A/ , B/]	00	01	10	11
0	0	0	0	1
1	0	1	1	1

carryout/

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

The fully explicit comonent

```
fulladd(A, B, carryin -> sum, carryout){
flow [A, B, carryin] -> [sum, carryout];
token {0:1} A, B, carryin, sum, carryout;
   {[A/0, B/0, carryin/0], [A/1, B/1, carryin/0], [A/1, B/0, carryin/1], [A/0, B/1, carryin/1]} -> sum/0;
   {[A/1, B/0, carryin/0], [A/0, B/1, carryin/0], [A/0, B/0, carryin/1], [A/1, B/1, carryin/1]} -> sum/1;
   {[A/0, B/0, carryin/0], [A/1, B/0, carryin/0], [A/0, B/1, carryin/0], [A/0, B/0, carryin/1]} -> carryout/0;
   {[A/1, B/1, carryin/1], [A/1, B/1, carryin/0], [A/0, B/1, carryin/1], [A/1, B/0, carryin/1]} -> carryout/1;
close A <- [sum/#, carryout/#];
close B <- [sum/#, carryout/#];
close carryin <- [sum/#, carryout/#];
close sum <- ?/#;
close carryout <- ?/#;
}
```

with default AND related **flow** and **close**

component with implied **flow** and **close** AND relations

```
fulladd(A, B, carryin -> sum, carryout){
token {0:1} A, B, carryin, sum, carryout;
   {[A/0, B/0, carryin/0], [A/1, B/1, carryin/0], [A/1, B/0, carryin/1], [A/0, B/1, carryin/1]} -> sum/0;
   {[A/1, B/0, carryin/0], [A/0, B/1, carryin/0], [A/0, B/0, carryin/1], [A/1, B/1, carryin/1]} -> sum/1;
   {[A/0, B/0, carryin/0], [A/1, B/0, carryin/0], [A/0, B/1, carryin/0], [A/0, B/0, carryin/1]} -> carryout/0;
   {[A/1, B/1, carryin/1], [A/1, B/1, carryin/0], [A/0, B/1, carryin/1], [A/1, B/0, carryin/1]} -> carryout/1;
}
```

Composing the Addition Path

path composition

```
adder(A, B, carryin -> sum, carryout){
flow [A, B, carryin] -> [sum, carryout];
token [0:31]{0:1} sum, A, B;
token {0:1} carryin, carryout;
token carry[0:32]{0:1};
path sum/i <
  (carryin -> carry/0);
  falladdA(A/i, B/i, carry/i -> sum/i, carry/i+1)
 (carry/max(i)+1 -> carryout ));
>
close A <- [sum/#, carryout/#];
close B <- [sum/#, carryout/#];
close carryin <- [sum/#, carryout/#];
close sum <- ?/#;
close carryout <- ?/#;
}
```

composition of the canonical component below results in the canonical 2D pipelining structure

fullword completeness is a relaxation of the 2D structure

digit pipelining and integrated pipelining are optimizations and enhancements of the 2D structure

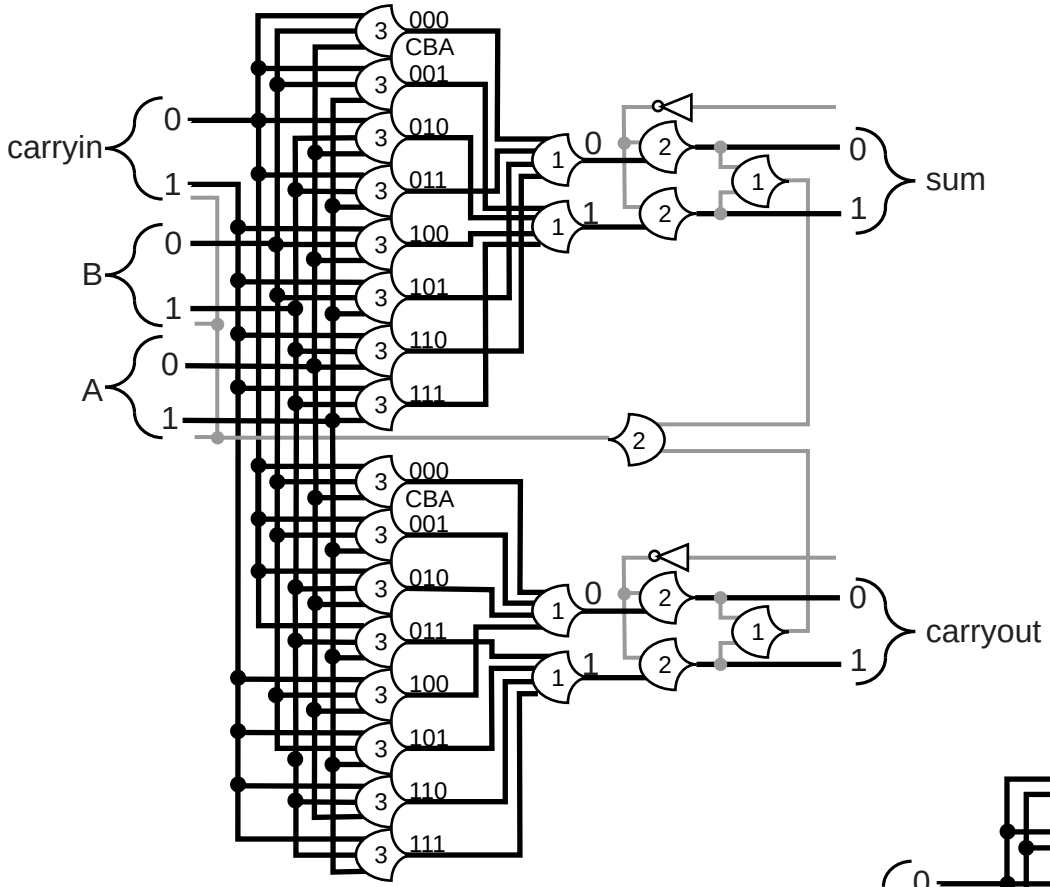
fulladd component

```
fulladd(A, B, carryin -> sum, carryout){
flow [A, B, carryin] -> [sum, carryout];
token {0:1} A, B, carryin, sum, carryout;
   {[A/0, B/0, carryin/0], [A/1, B/1, carryin/0], [A/1, B/0, carryin/1], [A/0, B/1, carryin/1]} -> sum/0;
   {[A/1, B/0, carryin/0], [A/0, B/1, carryin/0], [A/0, B/0, carryin/1], [A/1, B/1, carryin/1]} -> sum/1;
   {[A/0, B/0, carryin/0], [A/1, B/0, carryin/0], [A/0, B/1, carryin/0], [A/0, B/0, carryin/1]} -> carryout/0;
   {[A/1, B/1, carryin/1], [A/1, B/1, carryin/0], [A/0, B/1, carryin/1], [A/1, B/0, carryin/1]} -> carryout/1;
close A <- [sum/#, carryout/#];
close B <- [sum/#, carryout/#];
close carryin <- [sum/#, carryout/#];
close sum <- ?/#;
close carryout <- ?/#;
}
```

fulladdC - canonical component - direct mapping of equations

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

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



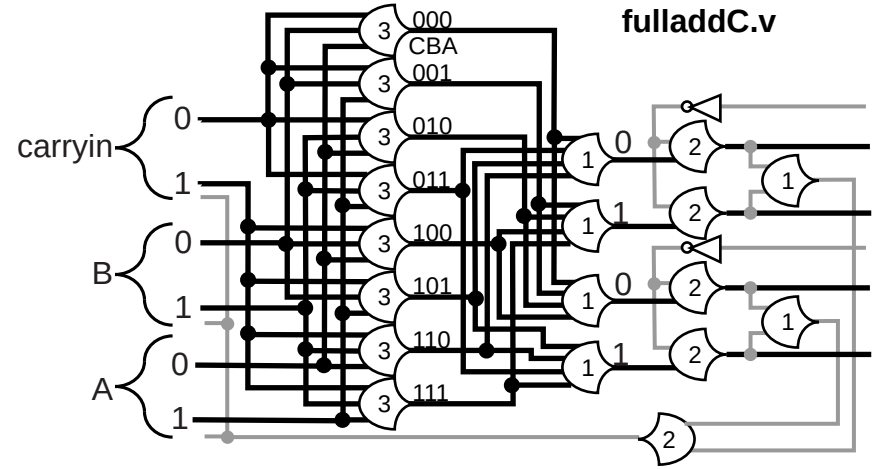
0 internal oscillations
29 cells
11 signal transitions

Oscillation periods
TBD

common subexpressions
common minterm

token min{000, 001, 010, 011, 100, 101, 110, 111};
[A/0, B/0, carryin/0] -> min/000
[A/0, B/0, carryin/1] -> min/001
[A/0, B/1, carryin/0] -> min/010
[A/0, B/1, carryin/1] -> min/011;
[A/1, B/0, carryin/0] -> min/100
[A/1, B/0, carryin/1] -> min/101
[A/1, B/1, carryin/0] -> min/110
[A/1, B/1, carryin/1] -> min/111;
{min/000, min/110, min/101, min/011} -> sum/0;
{min/100, min/010, min/001, min/111} -> sum/1;
{min/000, min/100, min/010, min/001} -> carryout/0;
{min/111, min/110, min/011, min/101} -> carryout/1;

fulladdC.v



0 internal oscillations
21 cells
10 signal transitions

Oscillation periods
TBD

fulladdC integrated

Pipeline the AND OR ranks

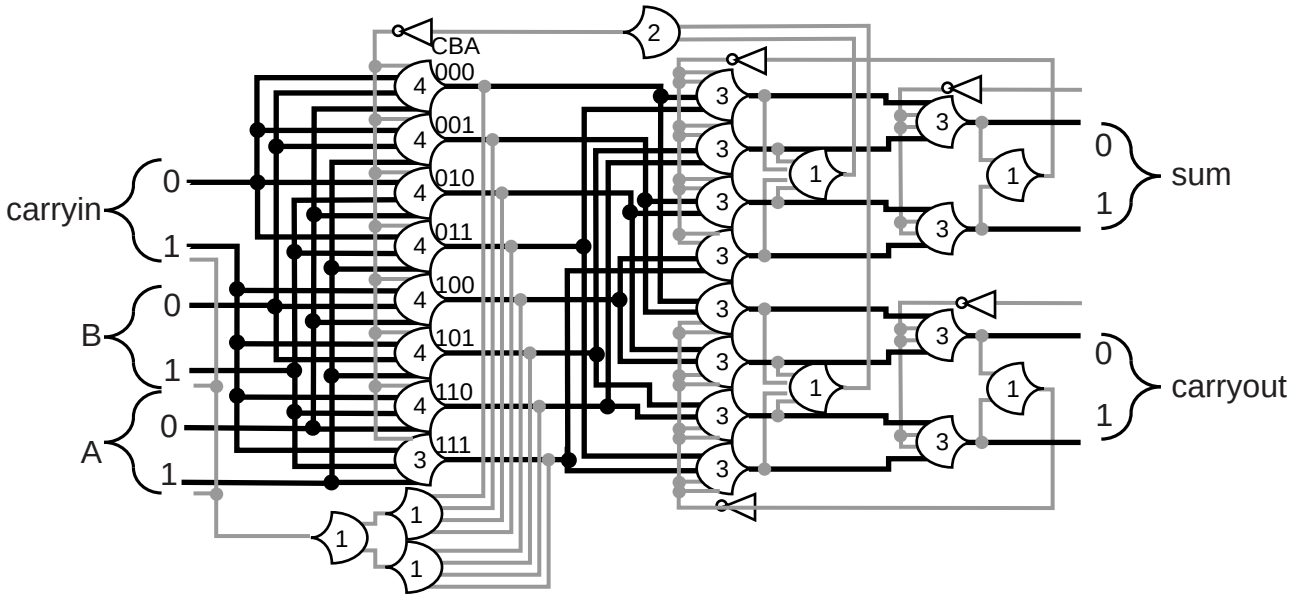
3 internal oscillations
33 cells
18 signal transitions

Oscillation periods

min	400 ps
mid	300

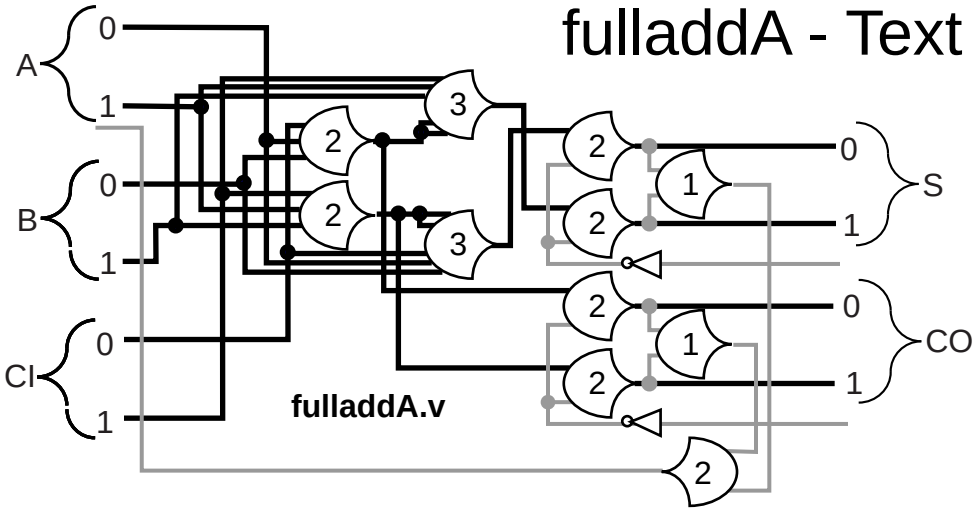
```
fulladdC(A, B carryin -> sum, carryout){
token {0:1} A, B, carryin, sum, carryout;
token min{000, 001, 010, 011, 100, 101, 110, 111};
token {0:3} midA, midB;
(A, B, carryin -> min){
[A/0, B/0, carryin/0] -> min/000
[A/0, B/0, carryin/1] -> min/001
[A/0, B/1, carryin/0] -> min/010
[A/0, B/1, carryin/1] -> min/011;
[A/1, B/0, carryin/0] -> min/100
[A/1, B/0, carryin/1] -> min/101
[A/1, B/1, carryin/0] -> min/110
[A/1, B/1, carryin/1] -> min/111;
}
(min -> midA, midB){
{min/000, min/110} -> midA/0
{min/101, min/011} -> midA/1;
{min/100, min/010} -> midA/2;
{min/001, min/111}  -> midA/3;
{min/000, min/100} -> midB/0;
{min/010, min/001} -> midB/1;
{min/111, min/110} -> midB/2;
{min/011, min/101} -> midB/3;
}
(midA -> sum){
{midOR/0, midOR/1} -> sum/0;
{midOR/2, midOR/3} -> sum/1;
}
(midB -> carryout){
{midOR/4, midOR/5} -> carryout/0;
{midOR/6, midOR/7} -> carryout/1;
}
}
```

fulladdC1.v



convert the 4 input ORs into a tree to make room for the closures

fulladdA - Text book fulladd component



0 internal oscillations
13 cells
8 signal transitions

Oscillation periods
TBD

difficult to derive
easy to prove

from the function equations we can recover the defining equations

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

TH23

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

TH34W2

expand carryout

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

The canonical equations are derived so the sum equations are correct and fulfill the completeness criterion.

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

TH23

restore don't cares

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

expand

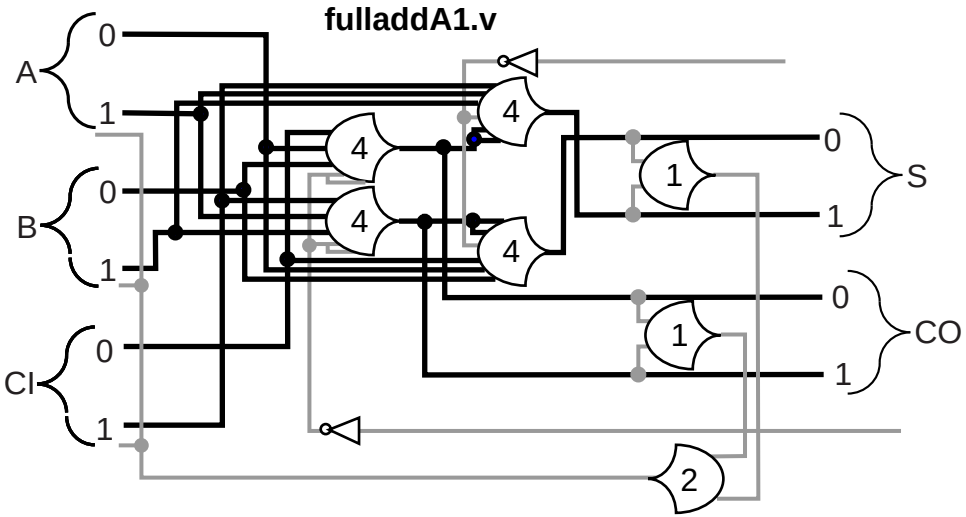
{[A/0, B/0, carryin/0], [A/0, B/0, carryin/1], [A/0, carryin/0, B/0], [A/0, carryin/0, B/1], [carryin/0, B/0, A/0], [carryin/0, B/0, A/1]} -> carryout/0;
{[A/1, B/1, carryin/0], [A/1, B/1, carryin/1], [A/1, carryin/1, B/0], [A/1, carryin/1, B/1], [carryin/1, B/1, A/0], [carryin/1, B/1, A/1]} -> carryout/1;

remove redundant terms

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

The canonical equations are derived so the sum equations are correct and fulfill the completeness criterion.

fulladdA - integrated



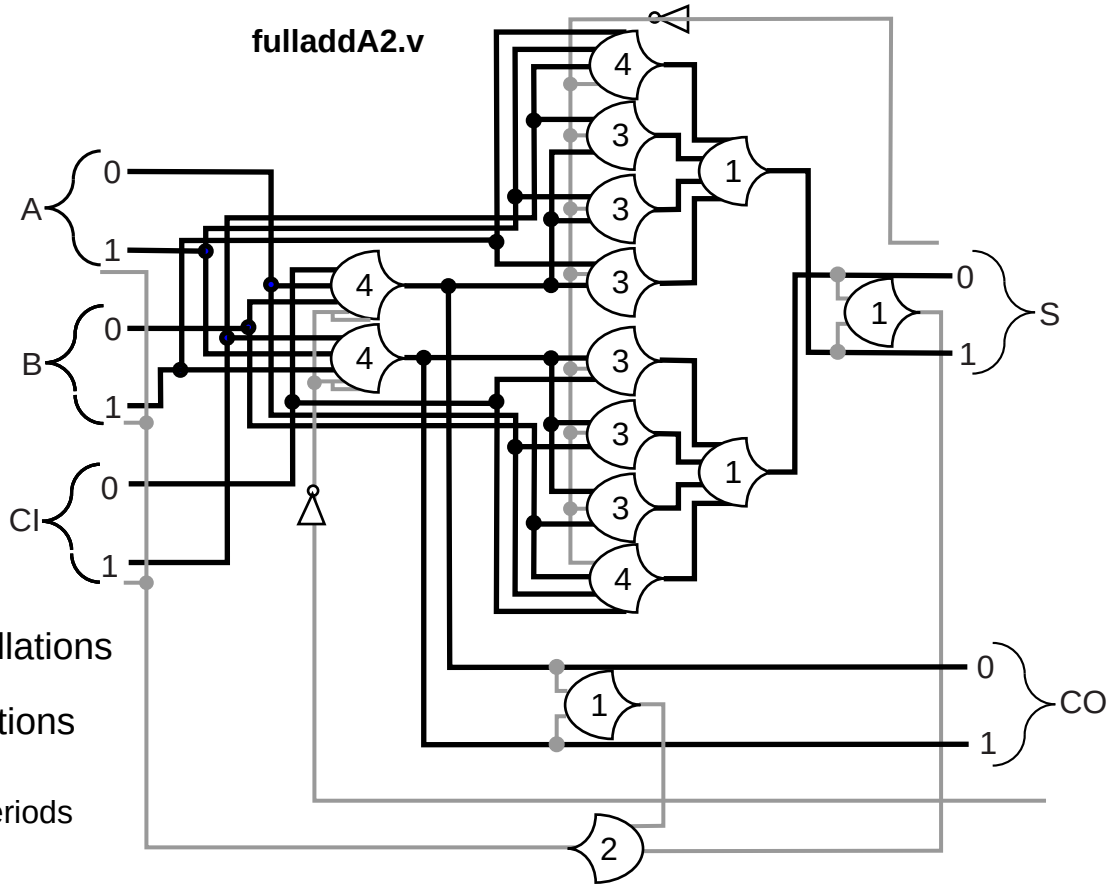
TH34W2 to TH44W2 to accept the closure.

TH44W2 is a 5 input gate not in the standard library and not practical for CMOS

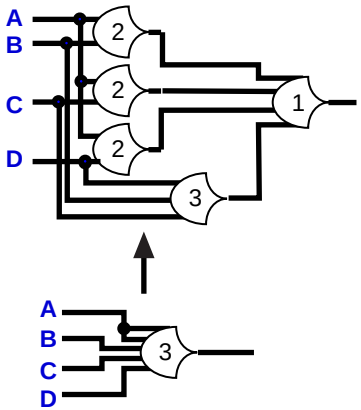
0 internal oscillations
9 cells
7 signal transitions

Oscillation periods
160
170

So we back the logic out of TH34W2 to get lower input gates that are in the NCL library that can accept the closure.



TheTH34W2 as a circuit of smaler functions.



14. AB + AC + AD + BCD
TC: 22 TH34W2

0 internal oscillations
17 cells
8 signal transitions

Oscillation periods
TBD

Two half adders derived from from Sum Equations

association - common sub

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

[A/0, B/0] -> AB0

[A/0, B/1] -> AB1

[A/1, B/0] -> AB2

[A/1, B/1] -> AB3

substitution

{[AB0, carryin/0], [AB3, carryin/0], [AB2, carryin/1], [AB1, carryin/1]} -> sum/0;
{[AB2, carryin/0], [AB1, carryin/0], [AB0, carryin/1], [AB3, carryin/1]} -> sum/1;

substitution

distribute carryin

{[carryin/0, {AB0, AB3}], [carryin/1, {AB2, AB1}]} -> sum/0;
{[carryin/0, {AB2, AB1}], [carryin/1, {AB0, AB3}]} -> sum/1;

factor out common terms

common sub

{AB0, AB3} -> suma0

{AB2, AB1} -> suma1

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

first halfadder

substitution

{[carryin/0, suma0], [carryin/1, suma1]} -> sum/0;
{[carryin/0, suma1], [carryin/1, suma0]} -> sum/1;

second halfadder

Carry Equations for fulladdB

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

common subexpression

[A/0, B/0] -> AB000

[A/0, B/1] -> AB001

[A/1, B/0] -> AB010

[A/1, B/1] -> AB011

substitution

{[AB00, carryin/0], [AB10, carryin/0], [AB01, carryin/0], [AB00, carryin/1]} -> carryout/0;
{[AB11, carryin/1], [AB11, carryin/0], [AB01, carryin/1], [AB10, carryin/1]} -> carryout/1;

factor out common terms

{[AB00, {carryin/0, carryin/1}], [{AB01, AB10}, carryin/0]} -> carryout/0;
{[AB11, {carryin/1, carryin/0}], [{AB01, AB10}, carryin/1]} -> carryout/1;

remove don't cares

{AB00, [{AB01, AB10}, carryin/0]]} -> carryout/0;
{AB11, [{AB01, AB10}, carryin/1]} -> carryout/1;

common subexpression from sum

{AB00, AB11} -> suma0

{AB10, AB01} -> suma1

substitution

{AB00, [suma1, carryin/0]]} -> carryout/0;
{AB11, [suma1, carryin/1]} -> carryout/1;

{[A/0, B/0], [suma1, carryin/0]]} -> carryout/0;
{[A/1, B/1], [suma1, carryin/1]} -> carryout/1;

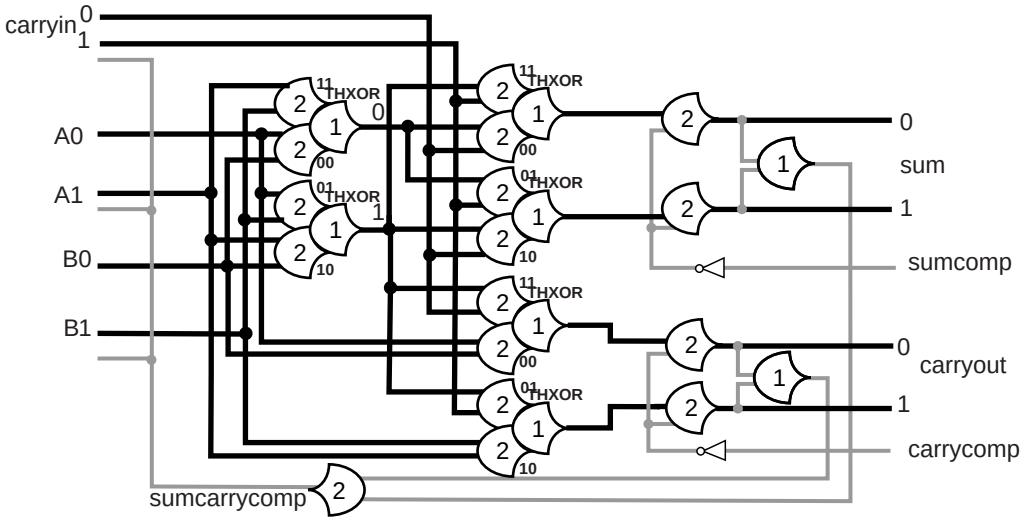
If sum fulfills the completeness criterion and the completeness of sum and carryout is ANDED then completeness of carryout does not have to imply input completeness

FulladdB - enable null carry play ahead

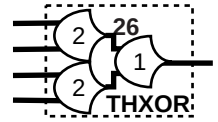
useful only for full word width completion and for additions with long chains of 00 and 11

{[A/0, B/0], [suma1, carryin/0]} -> carryout/0;
{[A/1, B/1], [suma1, carryin/1]} -> carryout/1;

0 internal oscillations
15 cells
10 signal transitions

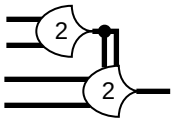


Oscillation periods
TBD



26. AB + CD

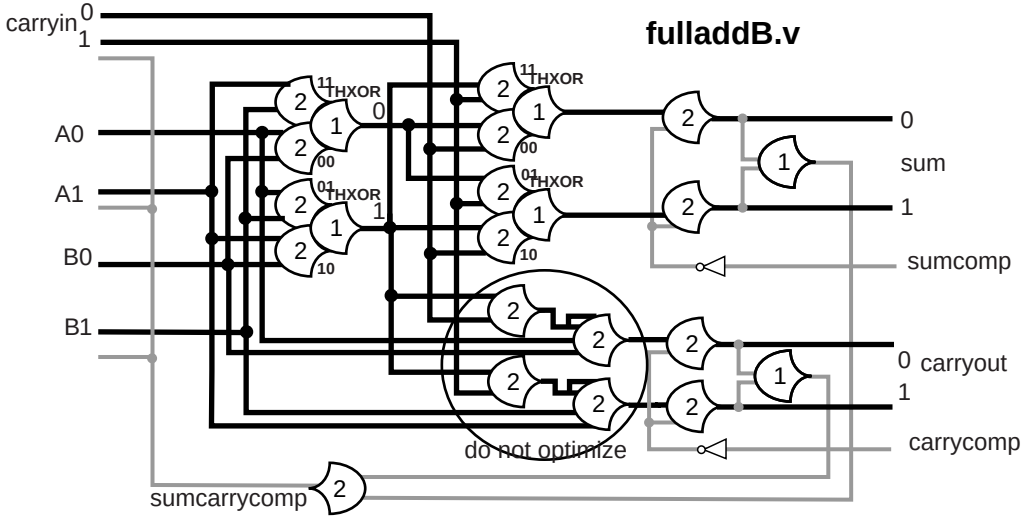
Back logic out of THXOR
to isolate carryin



There is another consideration not evident in the equations. The data wavefront carry will play ahead for [A/0, B/0] and for [A/1, B/1]. The carryin orphan will arrive holding the THXOR gate at data causing all fulladds to wait for the complete carry ripple with the null wavefront. Breaking the THXOR into two gates will isolated the carry when it is an orphan so that null carryins will play ahead also.

This is useful only for full word width completion and for additions with long chains of 00 and 11. The effect of the carry orphan is illustrated with the counter examples. For random numbers it does not seem to be an advantage. With free flowing digits all carry propagation is fully shadowed and is inconsequential .

fulladdB.v



0 internal oscillations
16 cells
11 signal transitions

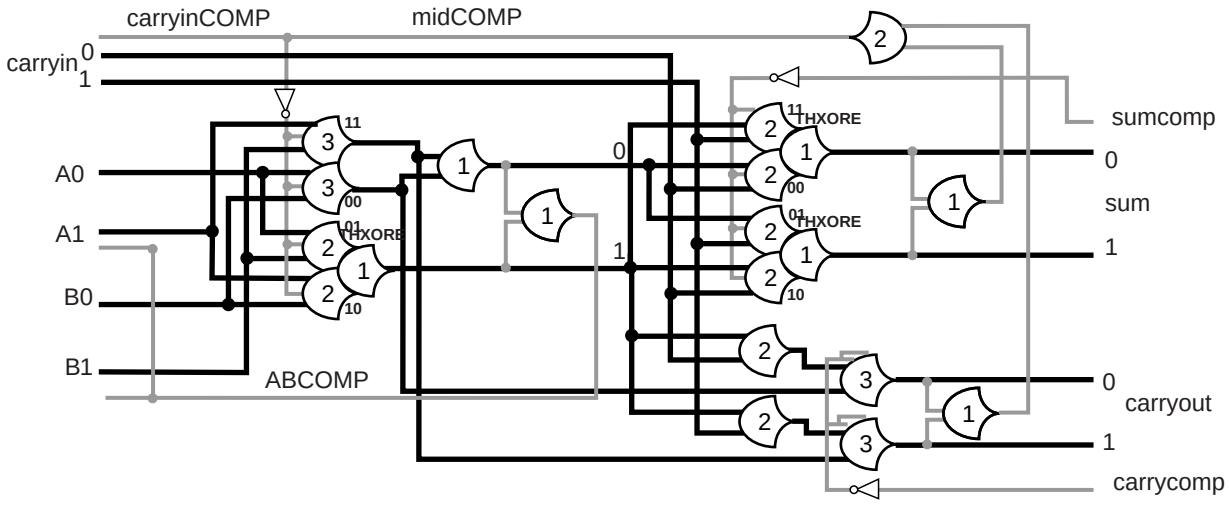
Oscillation periods
TBD

FulladdB - integrated

1 internal oscillations
17 cells
13 signal transitions

with 5 input THXORE

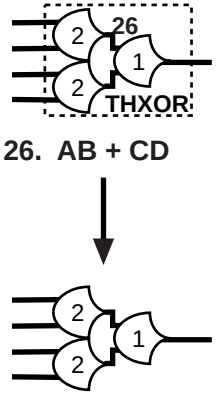
fulladdB1.v



Oscillation periods
midCOMP 420 ps

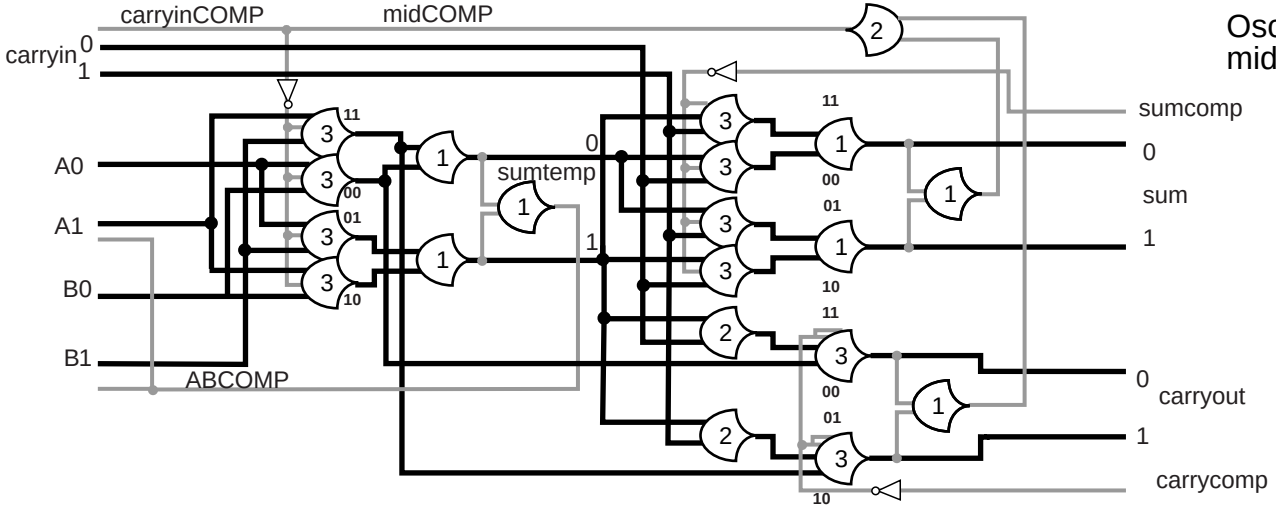
1 internal oscillations
23 cells
13 signal transitions

Back logic out of THXOR
to accommodate closure



with no 5 input gates

fulladdB2.v



Oscillation periods
midCOMP 440 ps

FulladdD as halfadd - halfadd - OR

oscillation

sum

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

first halfadder

{[carryin/0, suma/0], [carryin/1, suma/1]} -> sum/0;
{[carryin/0, suma/1], [carryin/1, suma/0]} -> sum/1;

second halfadder

carry

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

first halfadder

{[suma/0, carryin/0], [suma/0, carryin/1], [suma/1, carryin/0]} -> carryb/0
[suma/1, carryin/1] -> carryb/1

second halfadder

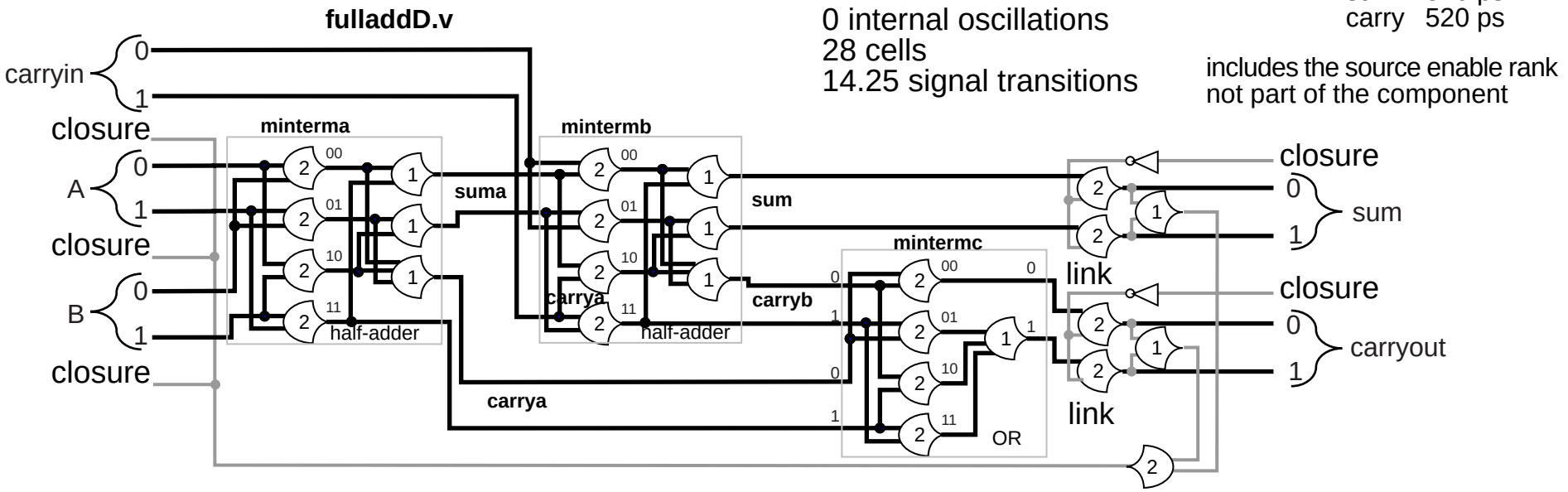
[carrya/0, carryb/0] -> carryout/0
{[carrya/1, carryb/1], [carrya/0, carryb/1], [carrya/1, carryb/0]} -> carryout/1

OR

Each token is either an input, and output or an internal result
The token flow dependence is established and defines oscillation boundaries

Oscillation periods
sum 540 ps
carry 520 ps

includes the source enable rank
not part of the component



FulladdD Single Component Specification

```
fulladdD(A, B carryin -> sum, carryout){
token {0:1} A, B, carryin, sum, carryout, suma, carrya, carryb;
token {0:3} minterma, mintermb, mintermc;
[A/0, B/0] -> minterma/0;
[A/0, B/1] -> minterma/1;
[A/1, B/0] -> minterma/2;
[A/1, B/1] -> minterma/3;
{minterma/0, minterma/3} -> suma/0;
{minterma/1, minterma/2} -> suma/1;
{minterma/0, minterma/1, minterma/2} -> carrya/0;
minterma/3 -> carrya/1;
[carryin/0, suma/0] -> mintermb/0;
[carryin/0, suma/1] -> mintermb/1;
[carryin/1, suma/0] -> mintermb/2;
[carryin/1, suma/1] -> mintermb/3;
{[mintermb/0, mintermb/1} -> sum/0;
{[mintermb/1, mintermb/2} -> sum/1;
{[mintermb/0, mintermb/1, mintermb/2} -> carryb/0;
mintermb/3 -> carryb/1;
[carrya/0, carryb/0] -> mintermc/0;
[carrya/0, carryb/1] -> mintermc/1;
[carrya/1, carryb/0] -> mintermc/2;
[carrya/1, carryb/1] -> mintermc/3;
{[mintermc/1, mintermc/2, mintermc/3} -> carryout/1;
mintermc/0 -> carryout/0;
}
```

fulladdD1

2 internal oscillations
31 cells
16.25 signal transitions

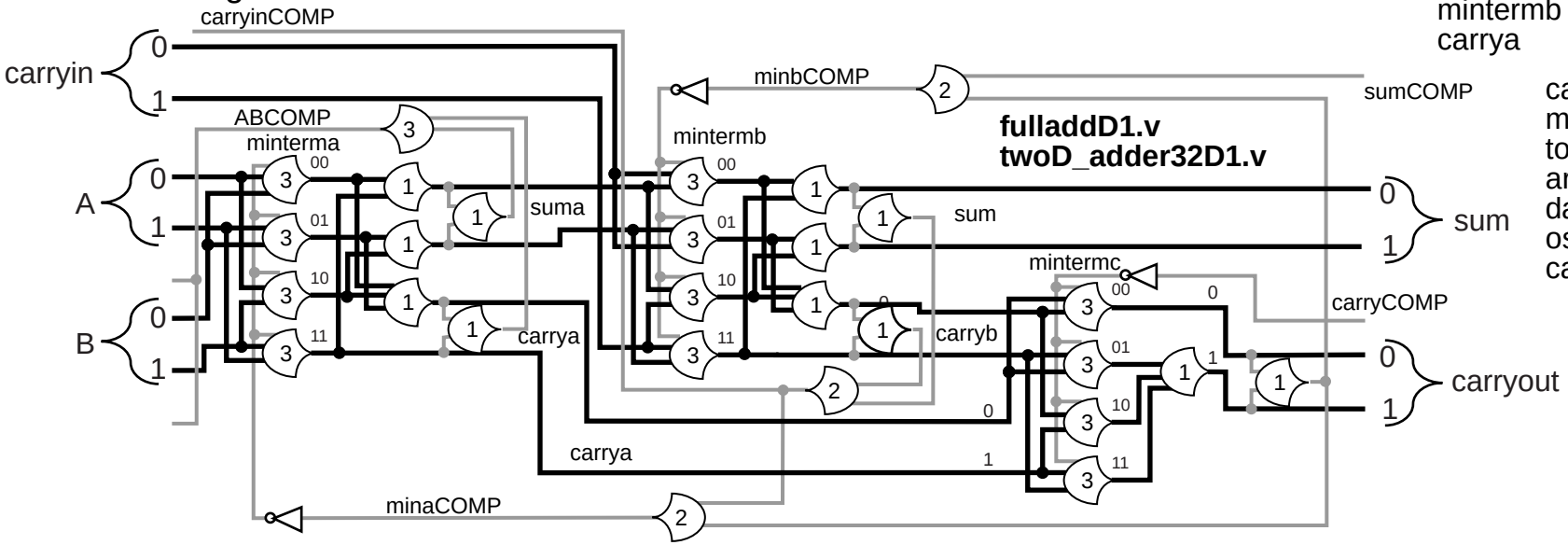
Throughput increases with
smaller shorter period oscillations

carrya is the pacing oscillation period.

Oscillation periods

minterma	460 ps
mintermb	500
carrya	610

carrya is presented to mintermc and then has to wait for carryb to arrive so the effective data path for the carrya oscillation is the the carryb path.



fulladdD2

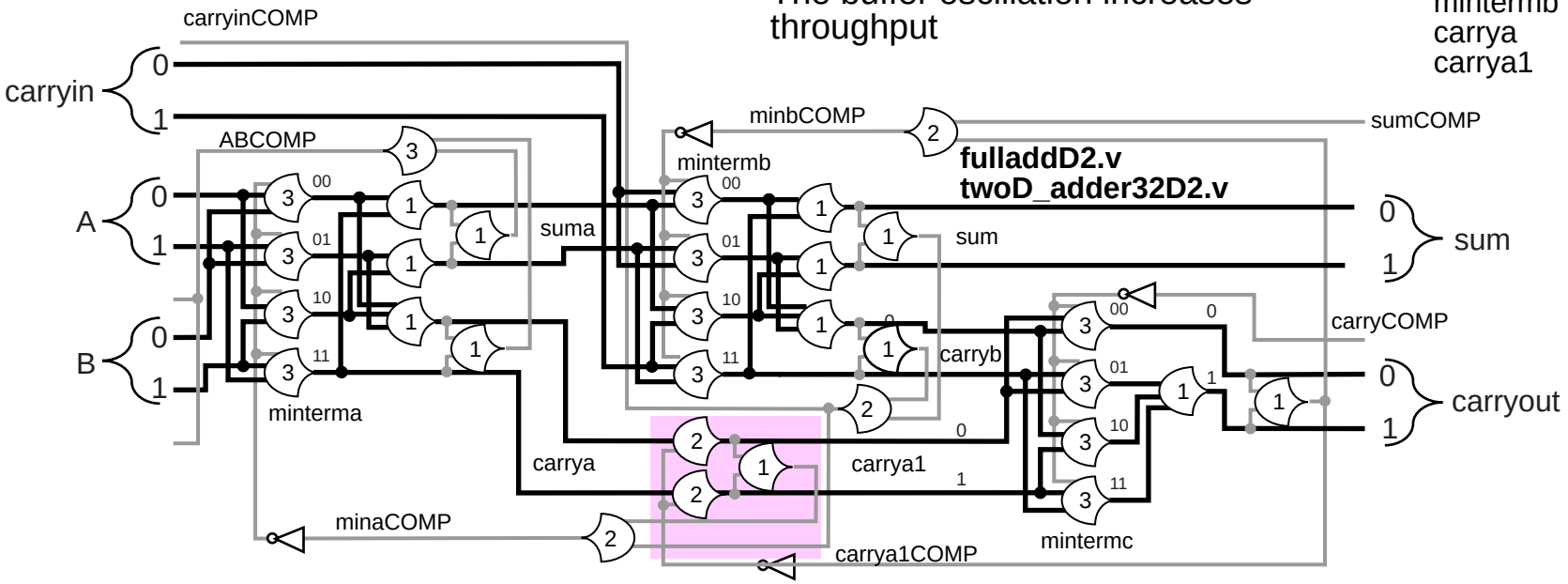
3 internal oscillations
35 cells
19.25 signal transitions

The buffer oscillation increases
throughput

minterma is now the pacing oscillation period.

Oscillation periods

minterma	460 ps
mintermb	500
carrya	380
carrya1	320



FulladdD1 Three Internal Component Specification

```
fulladdD1(A, B carryin -> sum, carryout){
token {0:1} A, B, carryin, sum, carryout, suma, carrya, carryb;
token {0:3} minterma, mintermb, mintermc;
(A, B -> suma, carrya){
[A/0, B/0] -> minterma/0;
[A/0, B/1] -> minterma/1;
[A/1, B/0] -> minterma/2;
[A/1, B/1] -> minterma/3;
{minterma/0, minterma/3} -> suma/0;
{minterma/1, minterma/2} -> suma/1;
{minterma/0, minterma/1, minterma/2} -> carrya/0;
minterma/3 -> carrya/1;
}
(suma, carryin -> sum, carryb){
[carryin/0, suma/0] -> mintermb/0;
[carryin/0, suma/1] -> mintermb/1;
[carryin/1, suma/0] -> mintermb/2;
[carryin/1, suma/1] -> mintermb/3;
{[mintermb/0, mintermb/1} -> sum/0;
{[mintermb/1, mintermb/2} -> sum/1;
{[mintermb/0, mintermb/1, mintermb/2} -> carryb/0;
mintermb/3 -> carryb/1;
}
(carrya, carryb -> carryout){
[carrya/0, carryb/0] -> mintermc/0;
[carrya/0, carryb/1] -> mintermc/1;
[carrya/1, carryb/0] -> mintermc/2;
[carrya/1, carryb/1] -> mintermc/3;
{[mintermc/1, mintermc/2, mintermc/3} -> carryout/1;
mintermc/0 -> carryout/0;
}
}
```

FulladdD2 Three Internal Component Specification with Buffer Component

```
fulladdD2(A, B carryin -> sum, carryout){
token {0:1} A, B, carryin, sum, carryout, suma, carrya, carrya1, carryb;
token {0:3} minterma, mintermb, mintermc;
(A, B -> suma, carrya){
[A/0, B/0] -> minterma/0;
[A/0, B/1] -> minterma/1;
[A/1, B/0] -> minterma/2;
[A/1, B/1] -> minterma/3;
{minterma/0, minterma/3} -> suma/0;
{minterma/1, minterma/2} -> suma/1;
{minterma/0, minterma/1, minterma/2} -> carrya/0;
minterma/3 -> carrya/1;
}
(suma, carryin -> sum, carryb){
[carryin/0, suma/0] -> mintermb/0;
[carryin/0, suma/1] -> mintermb/1;
[carryin/1, suma/0] -> mintermb/2;
[carryin/1, suma/1] -> mintermb/3;
{[mintermb/0, mintermb/1} -> sum/0;
{[mintermb/1, mintermb/2} -> sum/1;
{[mintermb/0, mintermb/1, mintermb/2} -> carryb/0;
mintermb/3 -> carryb/1;
}
(carrya -> carrya1)                    // buffer component
(carrya1, carryb -> carryout){
[carrya/0, carryb/0] -> mintermc/0;
[carrya/0, carryb/1] -> mintermc/1;
[carrya/1, carryb/0] -> mintermc/2;
[carrya/1, carryb/1] -> mintermc/3;
{[mintermc/1, mintermc/2, mintermc/3} -> carryout/1;
mintermc/0 -> carryout/0;
}
}
```

fulladdD3

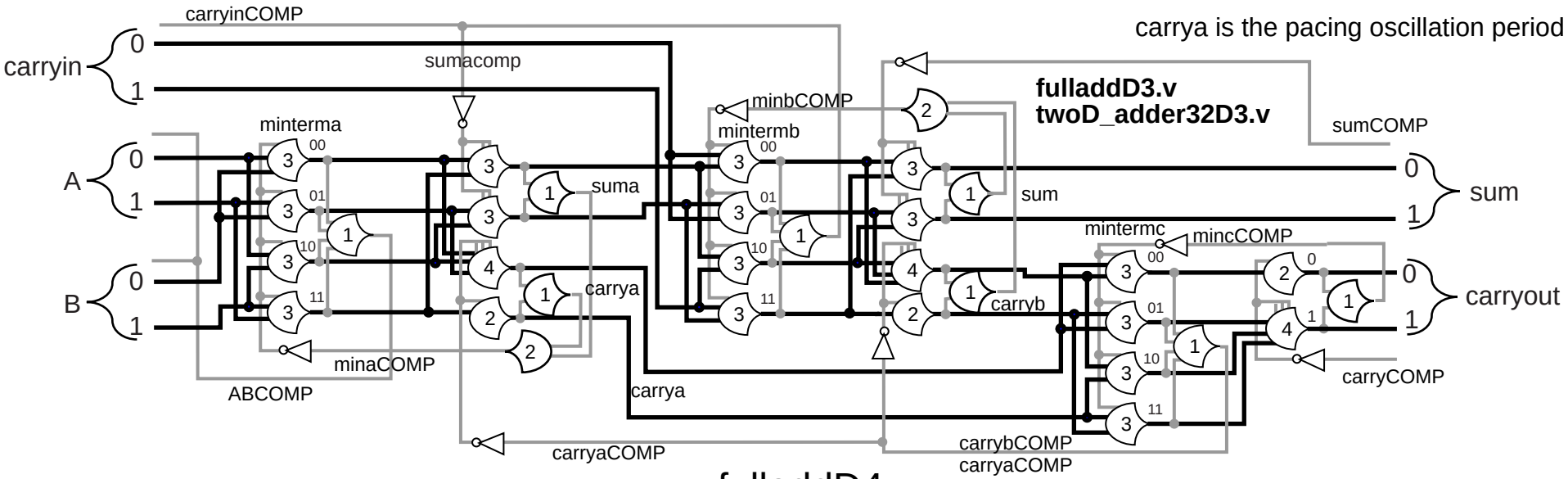
6 internal oscillations
39 cells
24 signal transitions

A long dependency path passing two oscillations has to wait on the later closure creating a slowest oscillation that paces the flow through the structure.

Oscillation periods

mintermb	280 ps
carryb	320
suma	300
carrya	480

carrya is the pacing oscillation period again



fulladdD4

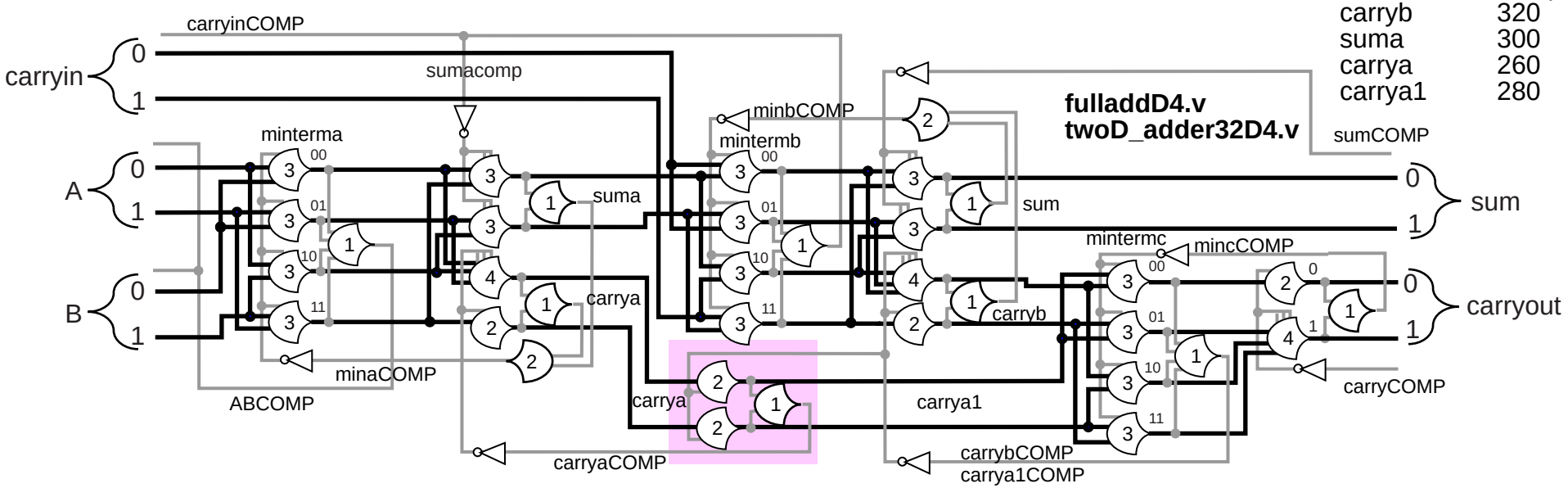
13 oscillations
42 cells
27 signal transitions

Adding a buffer component to a long dependency path can speed it up.

carryb is now the pacing oscillation period.

Oscillation periods

mintermb	280 ps
carryb	320
suma	300
carrya	260
carrya1	280



FulladdD3 Six Internal Component Specification

```
fulladdD3(A, B carryin -> sum, carryout){
token {0:1} A, B, carryin, sum, carryout, suma, carrya,
carryb;
token {0:3} minterma, mintermb, mintermc;
(A, B -> minterma){
[A/0, B/0] -> minterma/0;
[A/0, B/1] -> minterma/1;
[A/1, B/0] -> minterma/2;
[A/1, B/1] -> minterma/3;
}
(minterma -> suma, carrya){
{minterma/0, minterma/3} -> suma/0;
{minterma/1, minterma/2} -> suma/1;
{minterma/0, minterma/1, minterma/2} -> carrya/0;
minterma/3 -> carrya/1;
}
(suma, carryin -> mintermb){
[carryin/0, suma/0] -> mintermb/0;
[carryin/0, suma/1] -> mintermb/1;
[carryin/1, suma/0] -> mintermb/2;
[carryin/1, suma/1] -> mintermb/3;
}
(mintermb -> sum, carryb){
{[mintermb/0, mintermb/1} -> sum/0;
{[mintermb/1, mintermb/2} -> sum/1;
{[mintermb/0, mintermb/1, mintermb/2} -> carryb/0;
mintermb/3 -> carryb/1;
}
(carrya, carryb -> mintermc){
[carrya/0, carryb/0] -> mintermc/0;
[carrya/0, carryb/1] -> mintermc/1;
[carrya/1, carryb/0] -> mintermc/2;
[carrya/1, carryb/1] -> mintermc/3;
}
(mintermc -> carryout){
{[mintermc/1, mintermc/2, mintermc/3} -> carryout/1;
mintermc/0 -> carryout/0;
}
}
```

FulladdD4 Six Internal Component Specification with Buffer Component

```
fulladdD4(A, B carryin -> sum, carryout){
token {0:1} A, B, carryin, sum, carryout, suma, carrya,
carrya1, carryb;
token {0:3} minterma, mintermb, mintermc;
(A, B -> minterma){
[A/0, B/0] -> minterma/0;
[A/0, B/1] -> minterma/1;
[A/1, B/0] -> minterma/2;
[A/1, B/1] -> minterma/3;
}
(minterma -> suma, carrya){
{minterma/0, minterma/3} -> suma/0;
{minterma/1, minterma/2} -> suma/1;
{minterma/0, minterma/1, minterma/2} -> carrya/0;
minterma/3 -> carrya/1;
}
(suma, carryin -> mintermb){
[carryin/0, suma/0] -> mintermb/0;
[carryin/0, suma/1] -> mintermb/1;
[carryin/1, suma/0] -> mintermb/2;
[carryin/1, suma/1] -> mintermb/3;
}
(mintermb -> sum, carryb){
{[mintermb/0, mintermb/1} -> sum/0;
{[mintermb/1, mintermb/2} -> sum/1;
{[mintermb/0, mintermb/1, mintermb/2} -> carryb/0;
mintermb/3 -> carryb/1;
}
(carrya -> carrya1)                    // buffer component
(carrya1, carryb -> mintermc){
[carrya/0, carryb/0] -> mintermc/0;
[carrya/0, carryb/1] -> mintermc/1;
[carrya/1, carryb/0] -> mintermc/2;
[carrya/1, carryb/1] -> mintermc/3;
}
(mintermc -> carryout){
{[mintermc/1, mintermc/2, mintermc/3} -> carryout/1;
mintermc/0 -> carryout/0;
}
}
```

fulladdD5

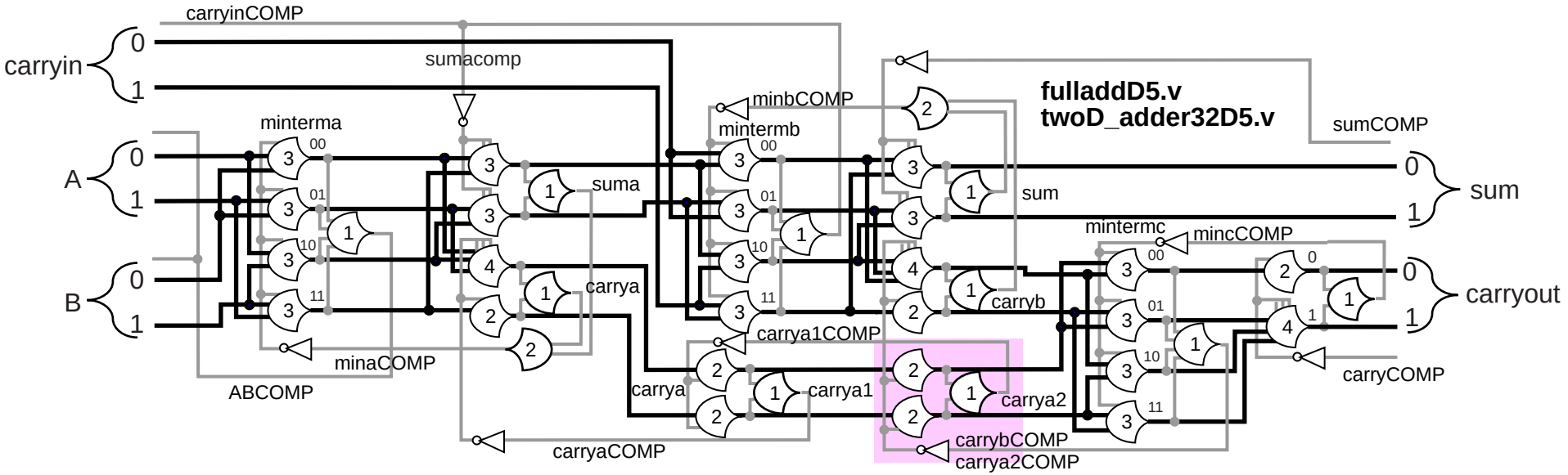
8 oscillations
46 cells
30 signal transitions

Adding the extra buffer did not speed it up.
This is not obvious without doing the numbers.

carryb is still the pacing oscillation period.

Oscillation periods

mintermb	280 ps
carryb	320
suma	300
carrya	260
carrya1	220
carrya2	280



FulladdD5 Six Internal Component Specification with Two Buffer Components

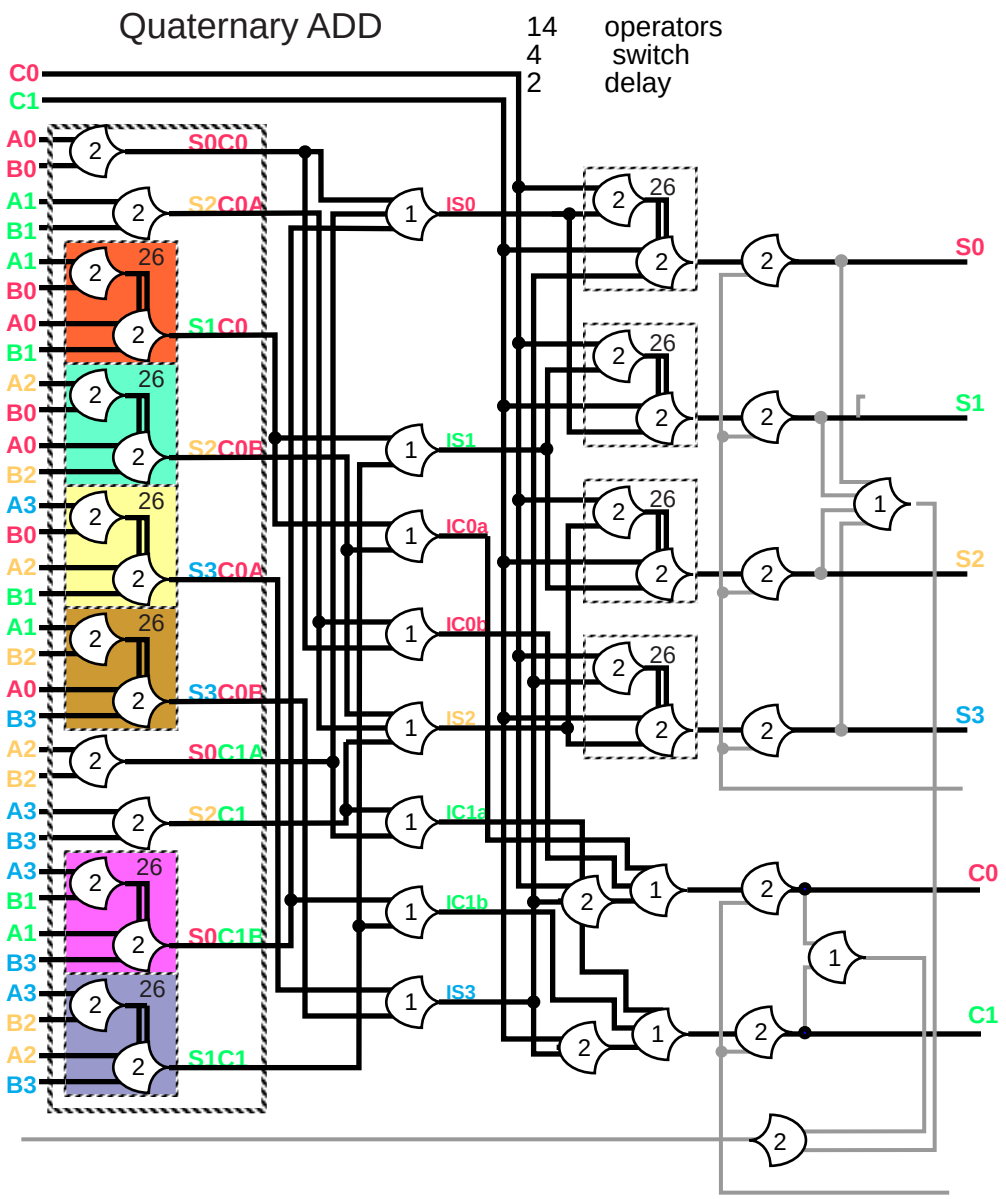
```
fulladdD5(A, B carryin -> sum, carryout){
token {0:1} A, B, carryin, sum, carryout, suma, carrya,
carrya1, carrya2, carryb;
token {0:3} minterma, mintermb, mintermc;
(A, B -> minterma){
[A/0, B/0] -> minterma/0;
[A/0, B/1] -> minterma/1;
[A/1, B/0] -> minterma/2;
[A/1, B/1] -> minterma/3;
}
(minterma -> suma, carrya){
{minterma/0, minterma/3} -> suma/0;
{minterma/1, minterma/2} -> suma/1;
{minterma/0, minterma/1, minterma/2} -> carrya/0;
minterma/3 -> carrya/1;
}
(suma, carryin -> mintermb){
[carryin/0, suma/0] -> mintermb/0;
[carryin/0, suma/1] -> mintermb/1;
[carryin/1, suma/0] -> mintermb/2;
[carryin/1, suma/1] -> mintermb/3;
}
(mintermb -> sum, carryb){
{[mintermb/0, mintermb/1} -> sum/0;
{[mintermb/1, mintermb/2} -> sum/1;
{[mintermb/0, mintermb/1, mintermb/2} -> carryb/0;
mintermb/3 -> carryb/1;
}
(carrya -> carrya1)                 // buffer component
(carrya1 -> carrya2)               // buffer component
(carrya2, carryb -> mintermc){
[carrya/0, carryb/0] -> mintermc/0;
[carrya/0, carryb/1] -> mintermc/1;
[carrya/1, carryb/0] -> mintermc/2;
[carrya/1, carryb/1] -> mintermc/3;
}
(mintermc -> carryout){
{[mintermc/1, mintermc/2, mintermc/3} -> carryout/1;
mintermc/0 -> carryout/0;
}
}
```

fulladdQ

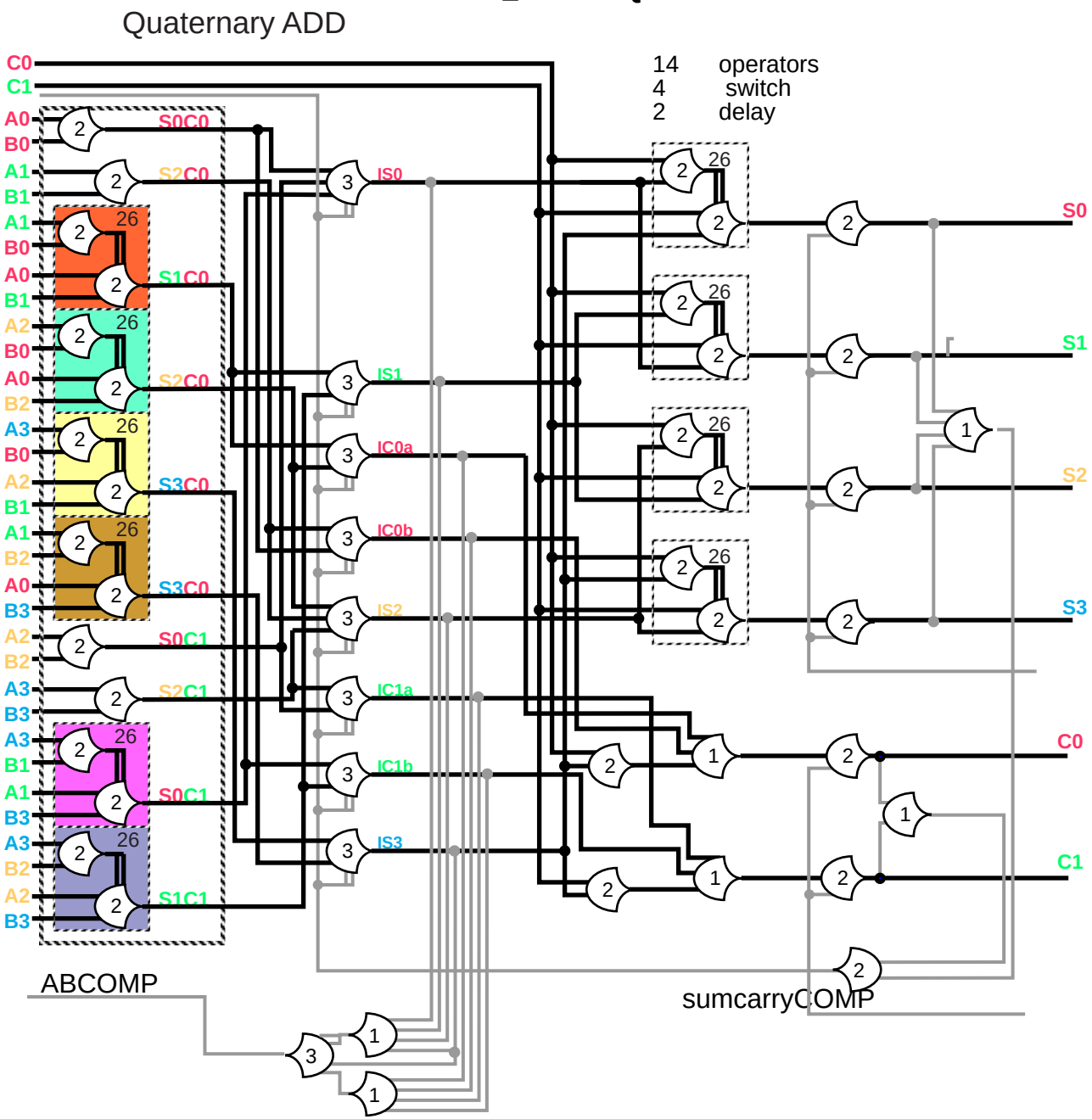
The Quaternary ADD is derived in LDD chapter 9

fulladdQ.v
twoD_adder32Q.v

```
fulladdQ(A, B carryin -> sum, carryout){
token {0:3} A, B, sum;
token {0:1} carryin, carryout;
token S0C0, S2C0A, S1C0, S2C0B, S3C0A, S3C0B,
S0C1A, S2C1, S0C1B, S1C1, IS0, IS1, IS2, IS3, ICOA,
ICOB, IC1A, IC1B;
(A, B -> minterma){
[A/0, B/0] -> S0C0;
[A/1, B/1] -> S2C0A;
{[A/1, B/0], [A/0, B/1]} -> S1C0;
{[A/2, B/0], [A/0, B/2]} -> S2C0B;
{[A/3, B/0], [A/2, B/1]} -> S3C0A;
{[A/1, B/2], [A/0, B/3]} -> S3C0B;
[A/2, B/2] -> S0C1A;
[A/3, B/3] -> S2C1;
{[A/3, B/1], [A/1, B/3]} -> S0C1B;
{[A/3, B/2], [A/2, B/3]} -> S1C1;
{S0C0, S0C1A, S0C1B} -> IS0;
{S1C0, S1C1} -> IS1;
{S2C0A, S2C0B, S2C1} -> IS2;
{S3C0A, S3C0B} -> IS3;
{S1C0, S2C0B} -> IC0A;
{S2C0A, S0C0} -> IC0B;
{S2C1, S0C1A} -> IC1A;
{S0C1B, S1C1} -> IC1B;
{[carryin/0, IS0], [carryin/1, IS3]} -> sum/0;
{[carryin/0, IS1], [carryin/1, IS0]} -> sum/1;
{[carryin/0, IS2], [carryin/1, IS1]} -> sum/2;
{[carryin/0, IS3], [carryin/1, IS2]} -> sum/3;
{[IC0A, IC0B, [carryin/0, IS3]} -> carryout/0;
{[IC1A, IC1B, [carryin/1, IS3]} -> carryout/0;
}
```

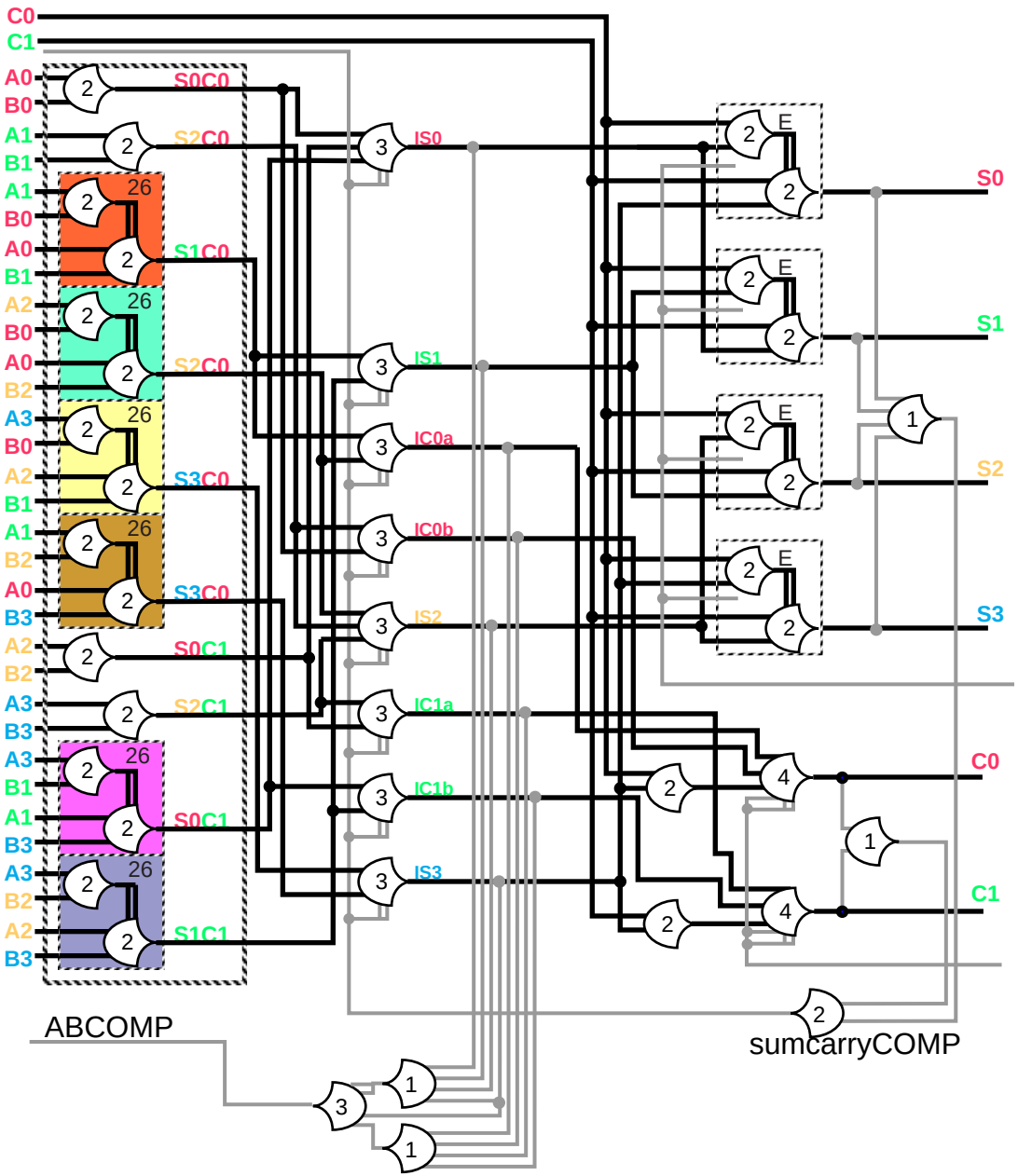


fulladdQ1.v
twoD_adder32Q1.v

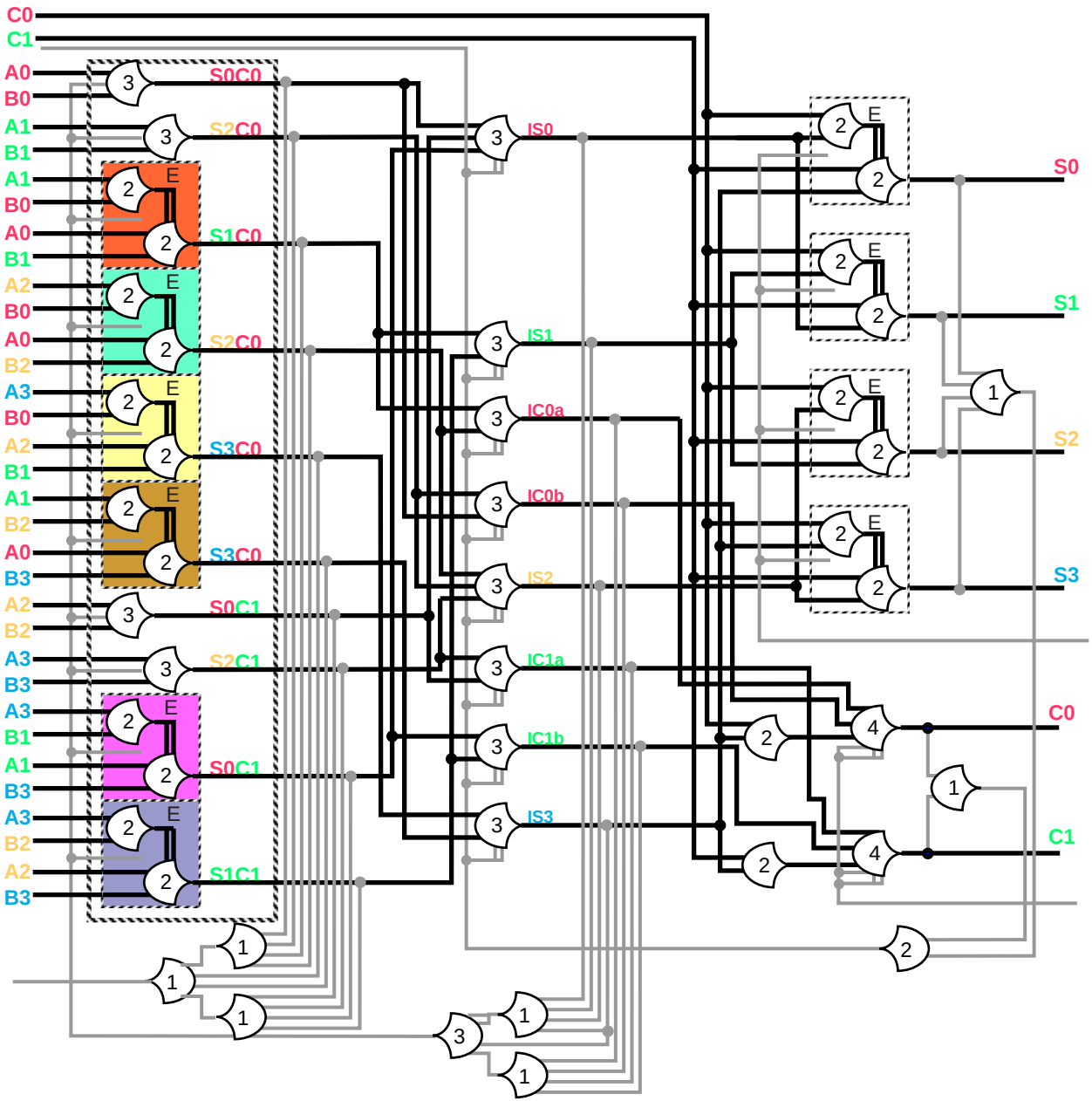


fulladdQ2.v
twoD_adder32Q2.v

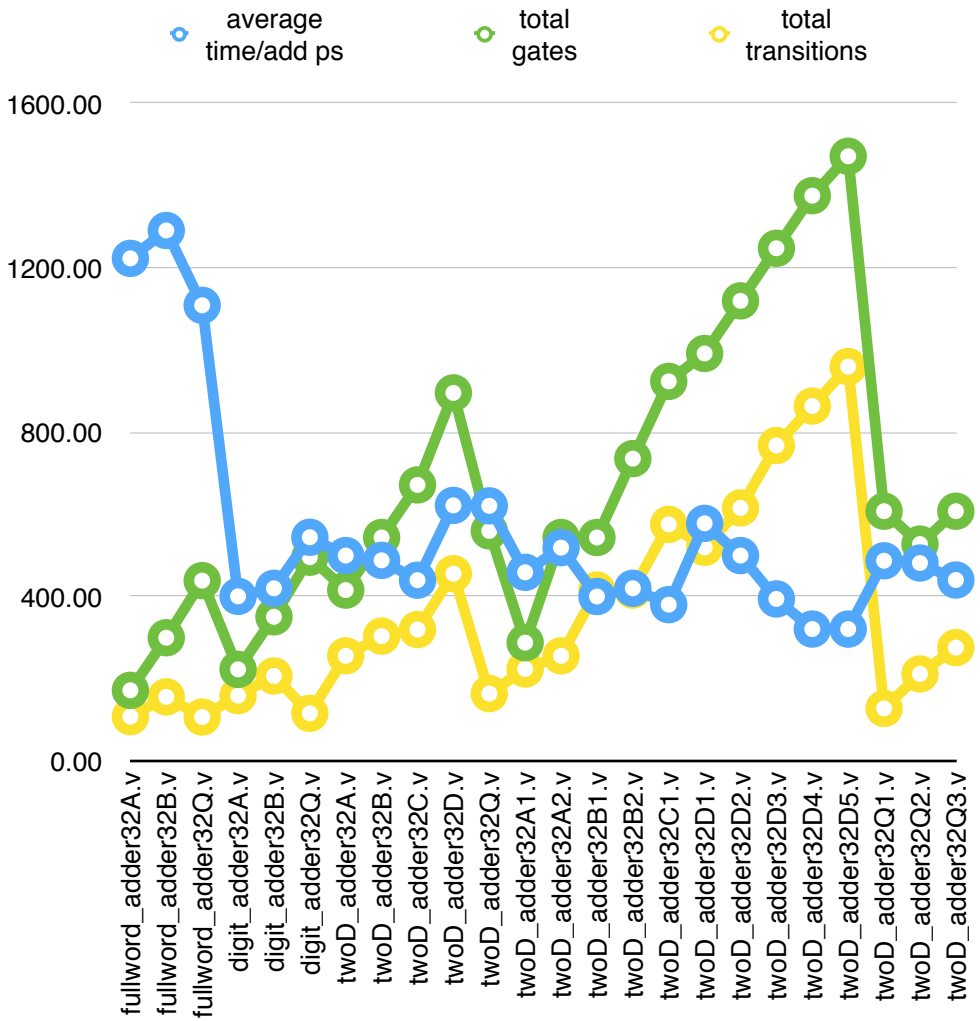
14 operators
4 switch
2 delay

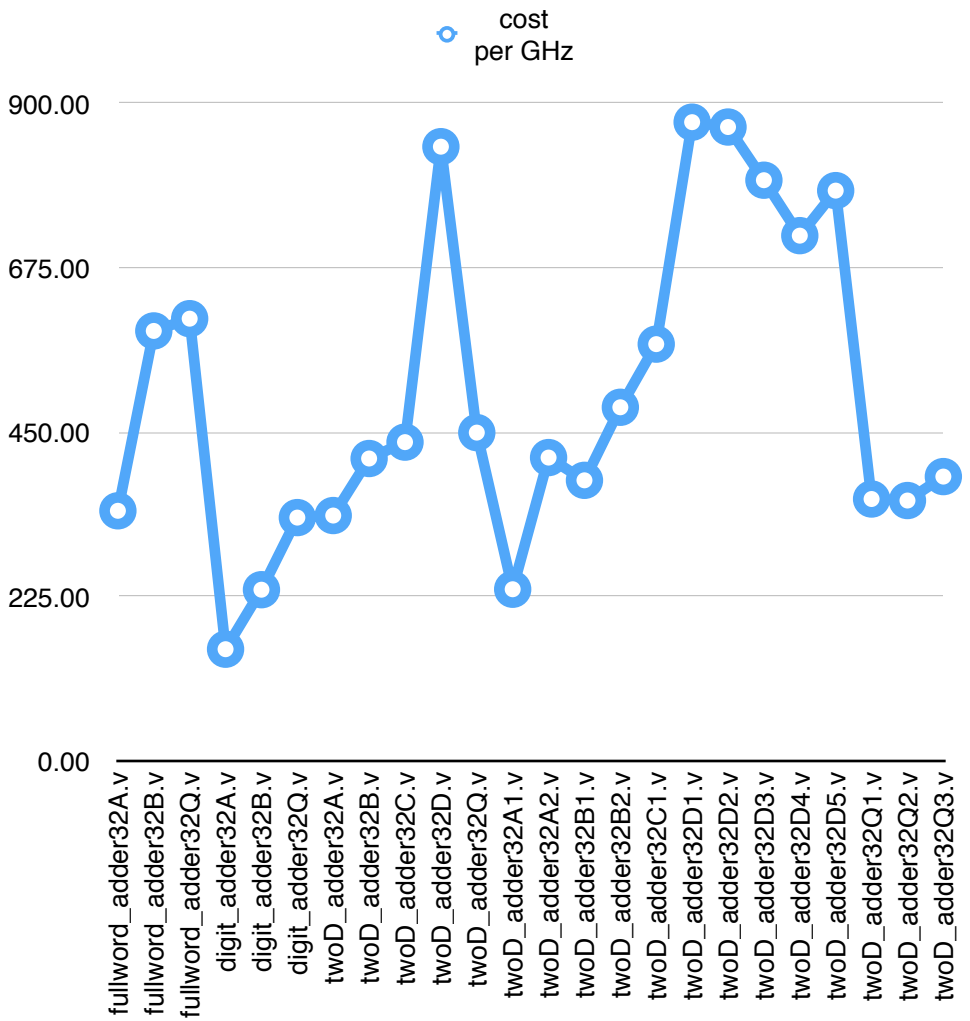


fulladdQ3.v
twoD_adder32Q3.v



Graphs of the statistics of the various adders and configurations from the spreadsheet.





count add

Two 32 bit counters driving a 32 bit adder illustrates plug and play

count_adder32A.v