Sandbox 6 Steering

NCL Steering
A data wavefront, flowing through a background of null emptiness, can be directed to one of several possible paths. The data wavefront will flow through the one path causing transitions while the other paths remain quiescent - null - empty and do not transition.

The logical structures of steering
The steering column

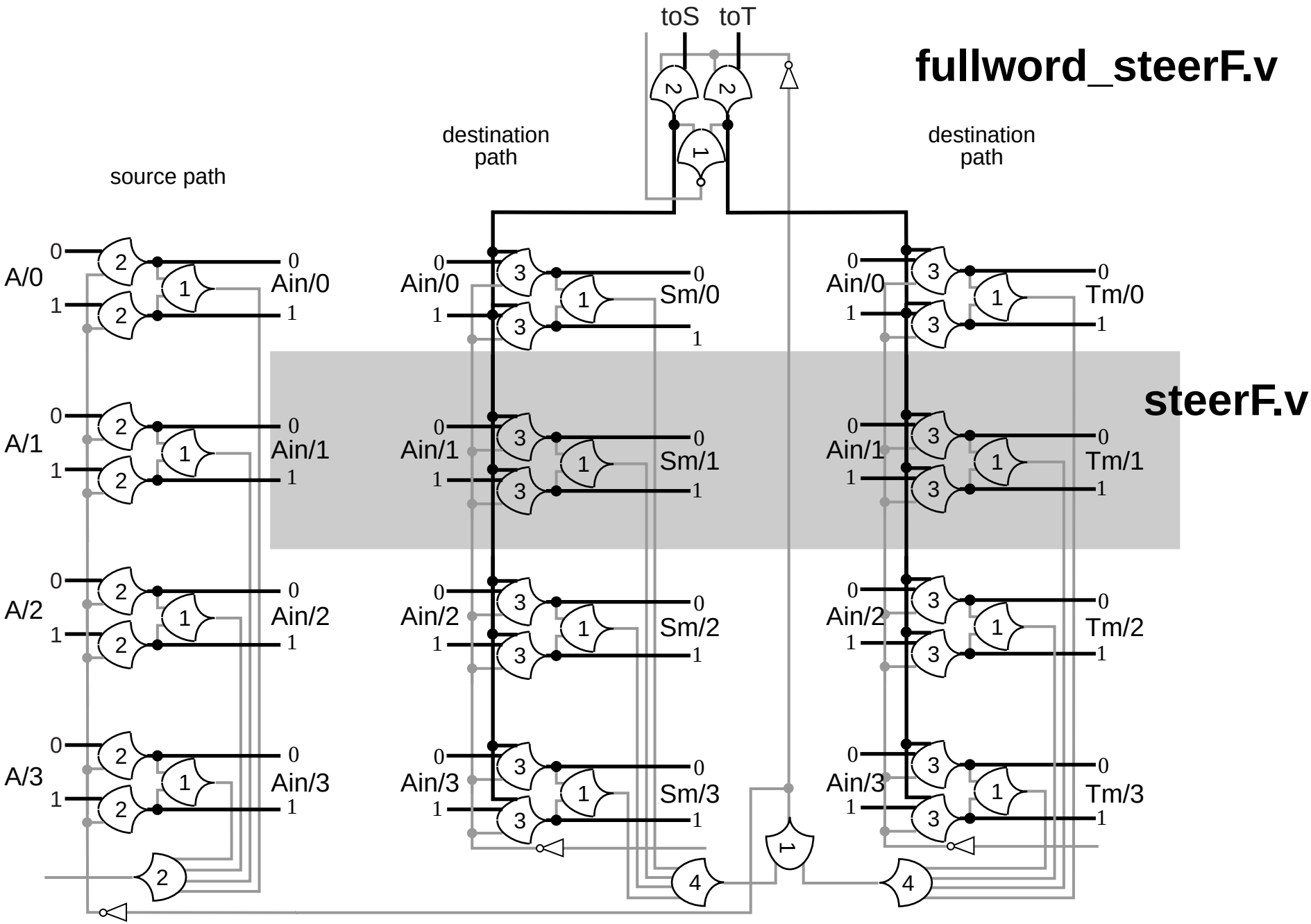
OR Related Flow Paths
Steering creates OR related flow paths. The data wavefront is steered to A OR B OR C and so on.

Races
Or related flow paths can be a source of races if the paths converge without logical constraint.

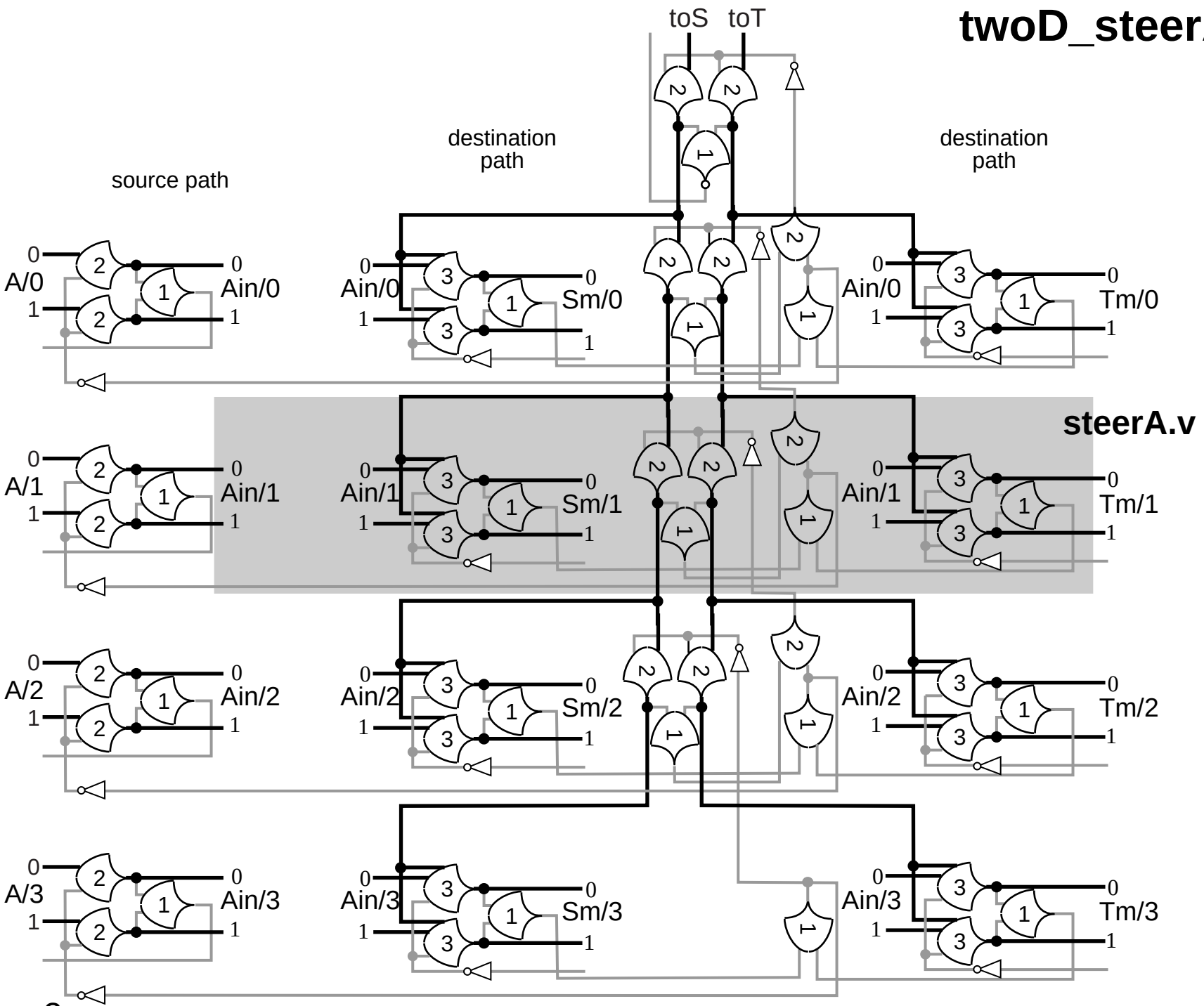
Managing OR flow convergence
There are two logical protocols to manage OR flow convergence without races

Baseline ALU model
Steering through multiple functions

Steer Column Component for Full Word Completeness



Steer Column Component for 2D Pipelining

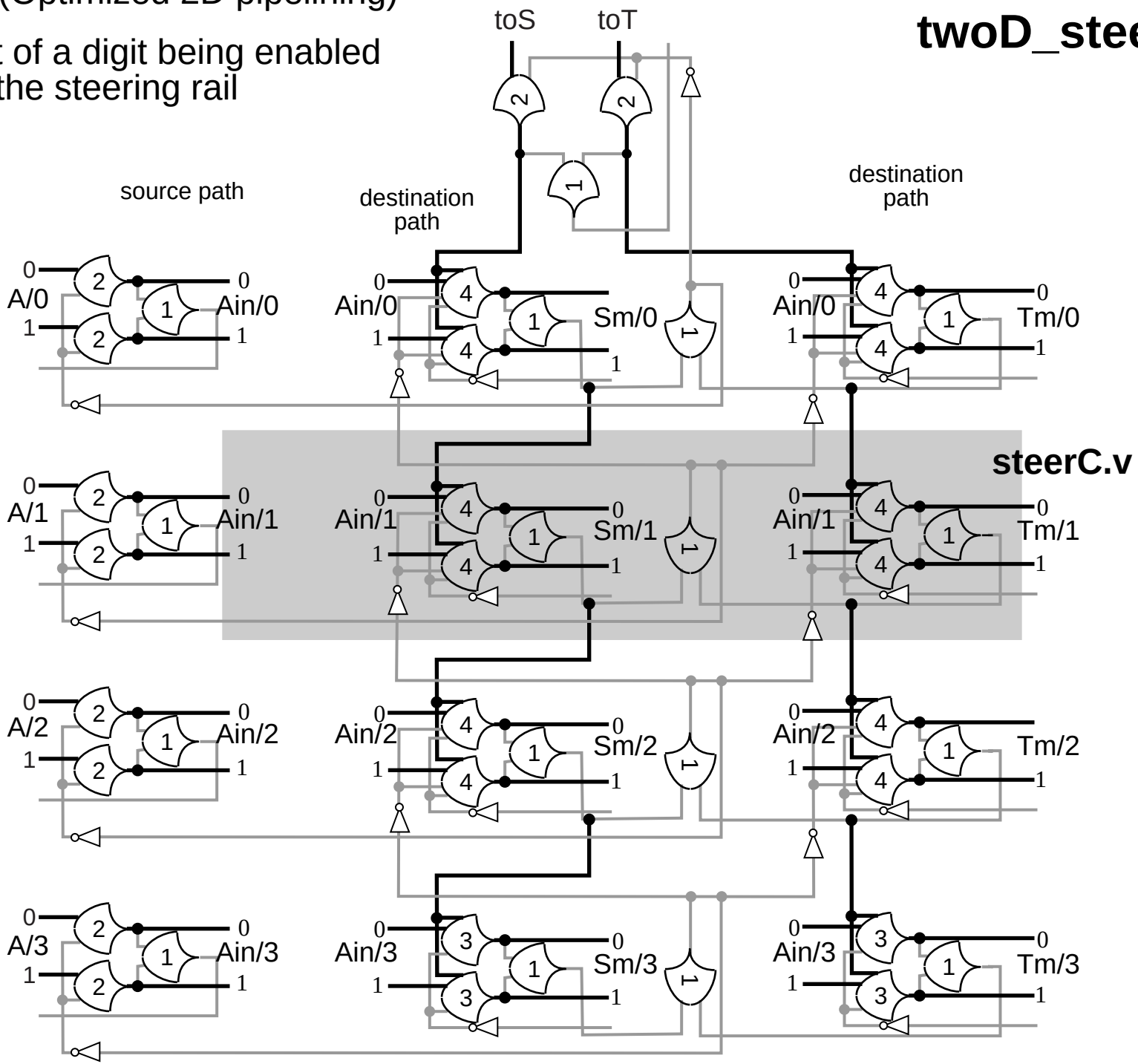


Steer Column Component for 2D Digit Pipelining

(Optimized 2D pipelining)

The fact of a digit being enabled implies the steering rail

twoD_steerC.v

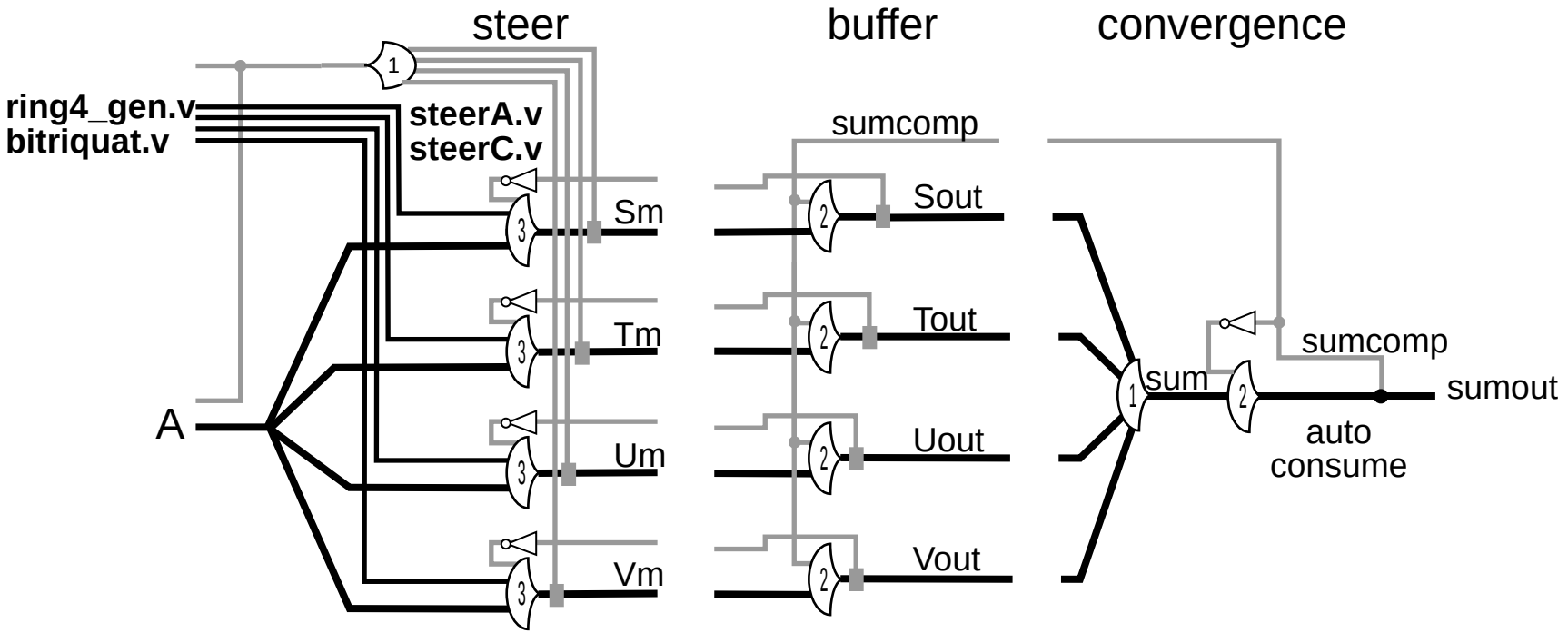


Steering Creates OR Related Paths with OR related Wavefront Flow

twoD_steer32A.v
ring4_gen.v steer with 4 state sequencer component .
steerA.v The steer token flows down the digits with an explicit flow path

twoD_steer32B.v
bitriquat.v steer with output of bitriquat adder.
steerA.v The steer token flows down the digits with an explicit flow path

twoD_steer32C.v
bitriquat.v steer with output of bitriquat adder.
steerC.v The steer token is regenerated from completeness of each digit

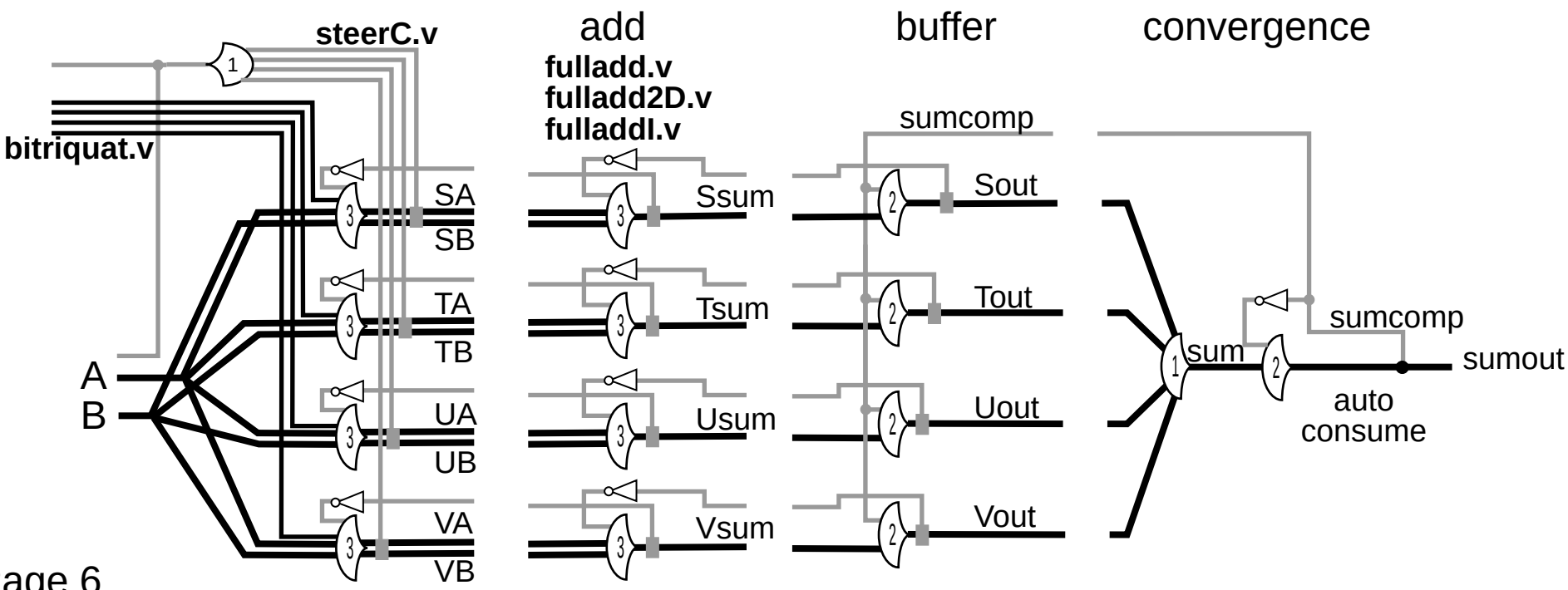


Insert 32 Bit Adders in the Steered Flow Paths

twoD_steer32D1.v
bitriquat.v steer with output of bitriquat adder.
steerC.v The steer token is regenerated from completeness of each digit
fulladd.v A 32 bit 2D digit piped adder is inserted in each flow path

twoD_steer32D2.v
bitriquat.v steer with output of bitriquat adder.
steerC.v The steer token is regenerated from completeness of each digit
fulladd2D.v A 32 bit 2D piped adder is inserted in each flow path

twoD_steer32D3.v
bitriquat.v steer with output of bitriquat adder.
steerC.v The steer token is regenerated from completeness of each digit
fulladdI.v A 32 bit 2D integrated adder is inserted in each flow path



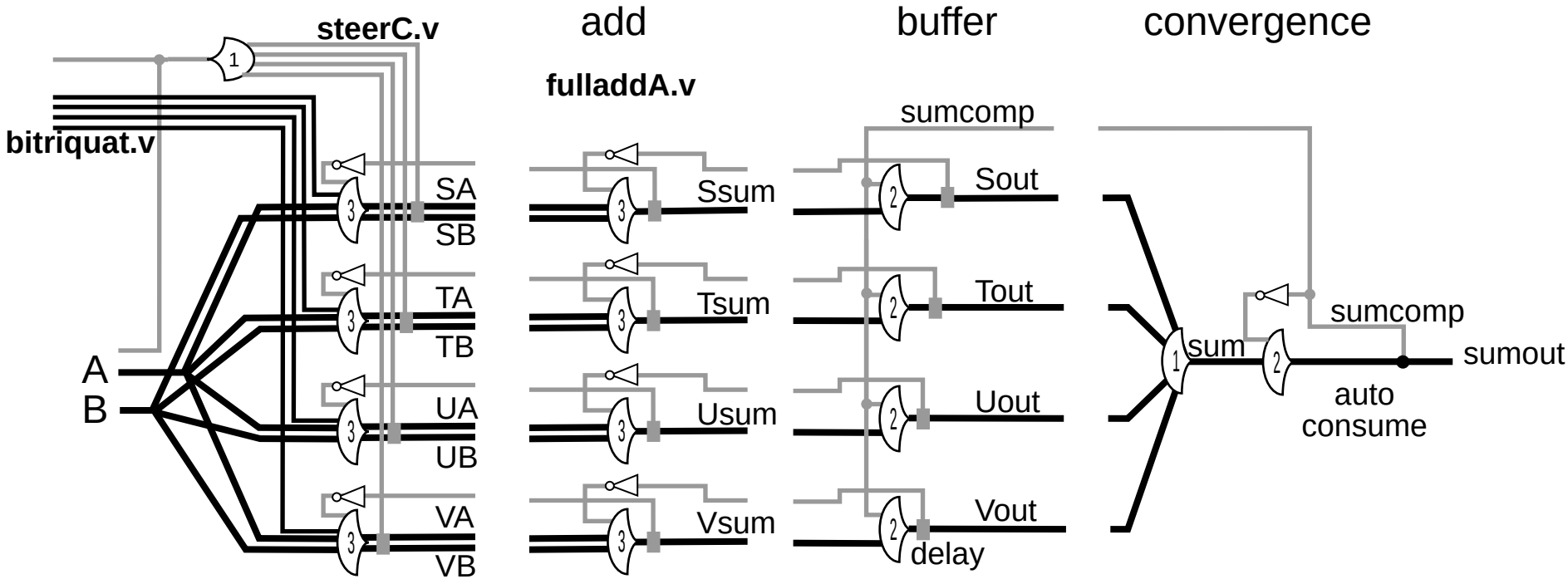
The OR Flow Convergence Race

The steering creates OR related paths which, flowing without coordination, will race into the OR convergence. We have gotten by so far because all four paths have been identical and the simultation maintains exact timing through each path.

We place a delay in the V path to expose the race. A TH22A gate has a 300 ps delay in contrast to the 30 ps delay for TH22.

The result is that the other paths race past V into the OR convergence, digits get mixed up and the test bench begins failing.

twoD_steer32E.v
bitriquat.v steer with output of bitriquat adder.
steerC.v The steer token is regenerated from completeness of each digit
fulladd.v A 32 bit 2D digit piped adder is inserted in each flow path
TH22A gates in V path to create an unbalancing delay



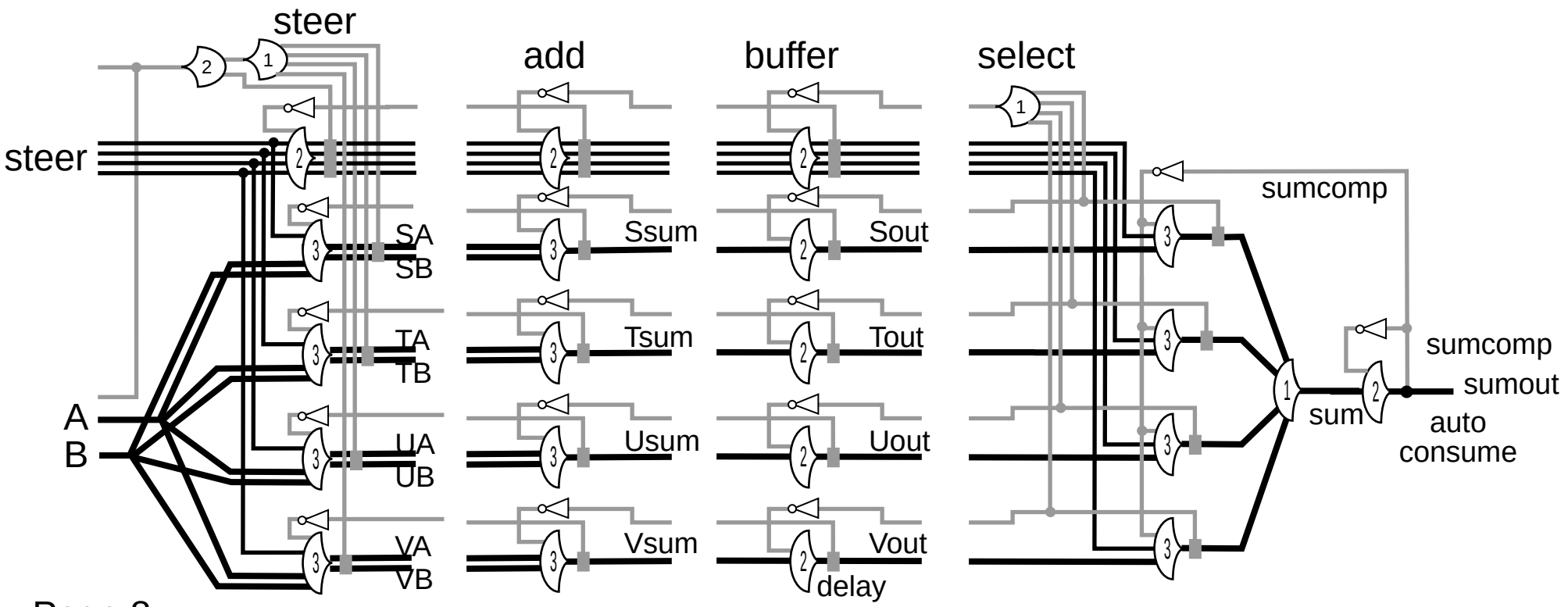
OR Related wavefronts Explicitly Ordered into the Convergence

Use the steer variable to enable the OR flows into the convergence in the same order that they were steered into the OR related paths

twoD_steer32F.v with delay
bitriquat.v steer with output of bitriquat adder.
steerC.v The steer token is regenerated from completeness of each digit
fulladd.v A 32 bit 2D digit piped adder is inserted in each flow path
TH22A gates in V parh to create an unbalancing delay
selectA.v select component to order flows into the OR converge

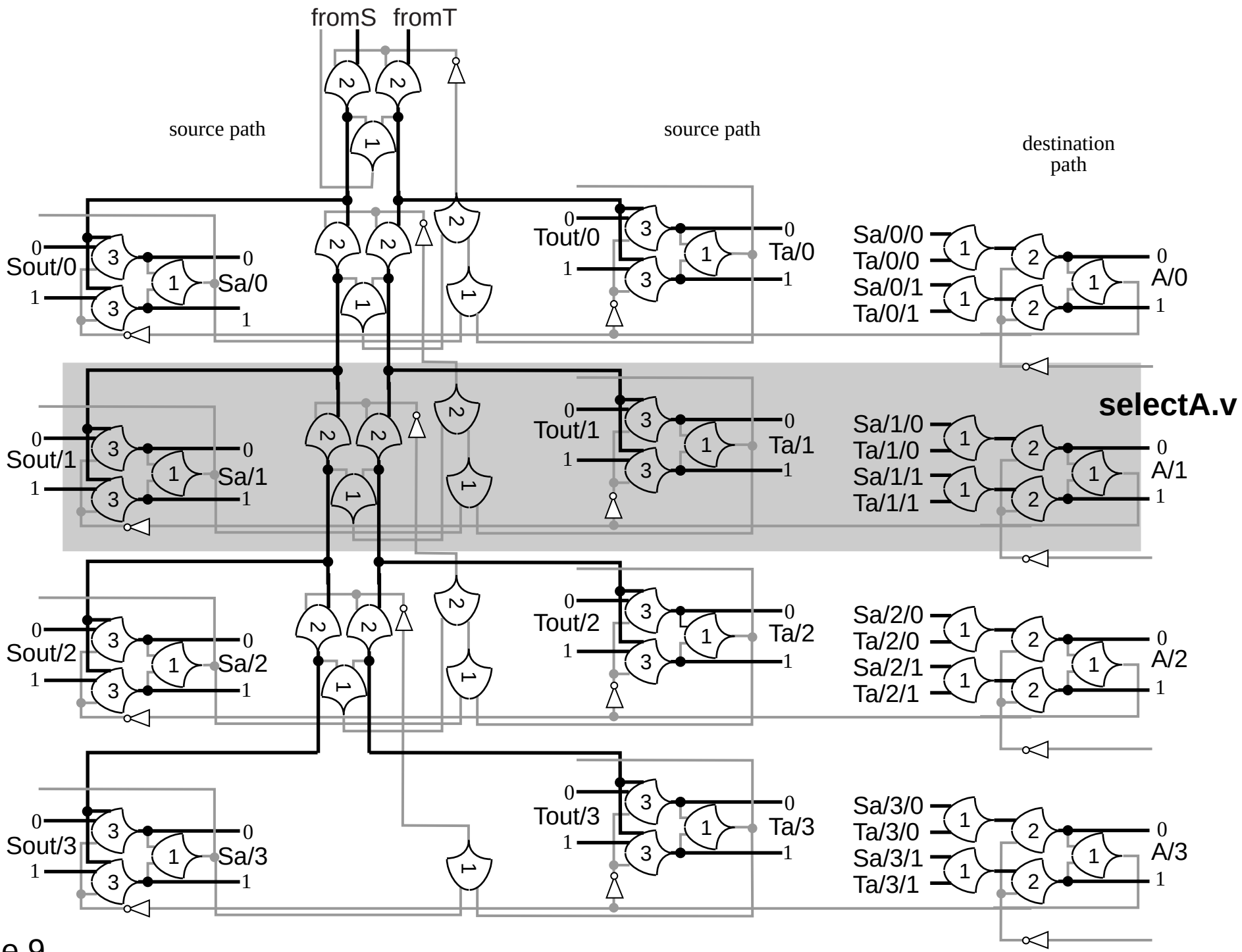
twoD_steer32FC.v uses **selectC.v** with delay present

twoD_steer32FN.v delay is removed



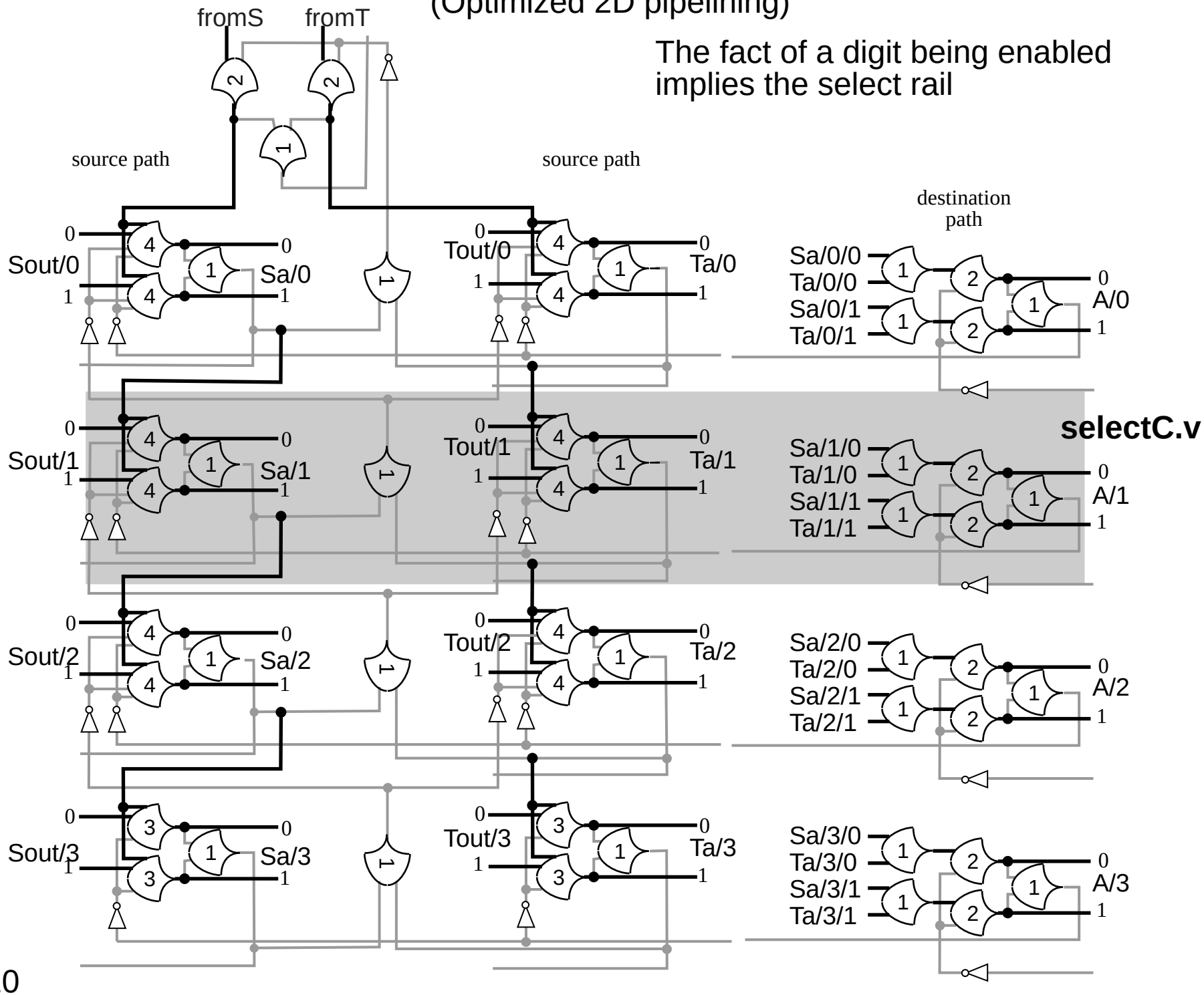
Select Column Component for 2D Pipelining

twoD_steerF.v



Select Column Component for 2D Digit Pipelining

(Optimized 2D pipelining)



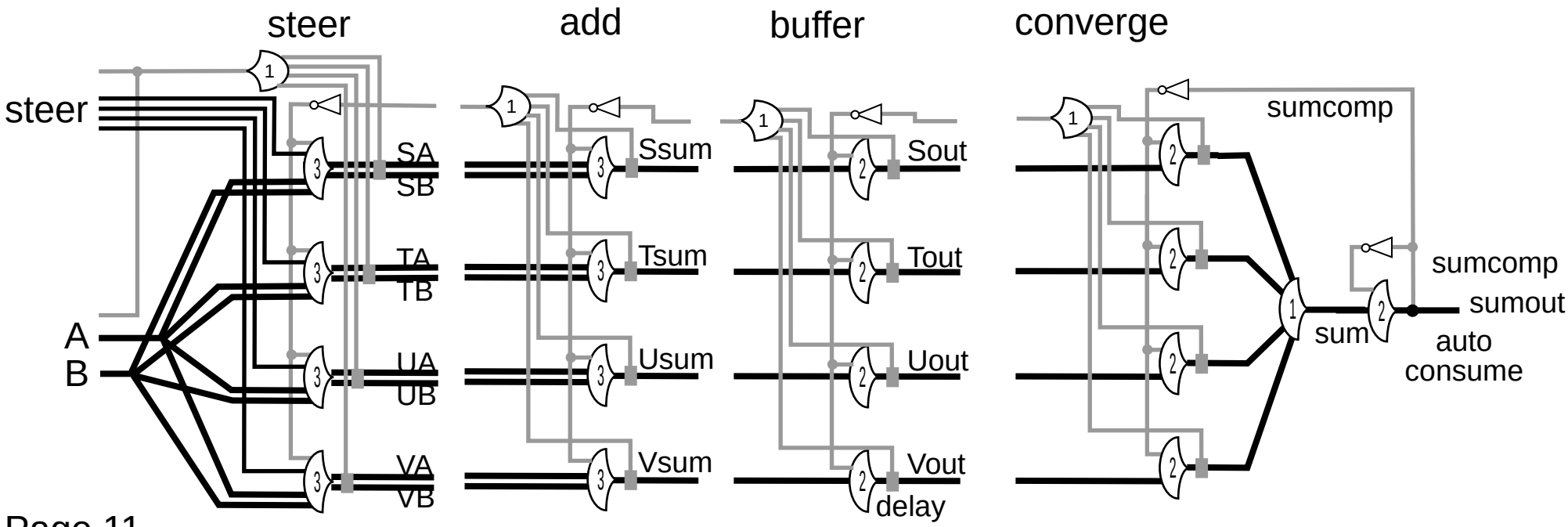
OR Related Paths Encompassed by Each Oscillation

The OR related paths do not free flow in relation to each other. Each wavefront flowing through the OR related paths is treated as a single coherent wavefront regulating the OR related wavefronts in Nthness ordered flow. An N+1th wavefront cannot flow past an Nth wavefront.

This is how the flow of the OR related rails of a multi-rail variable is managed.

twoD_steer32G1.v with delay
bitriquat.v steer with output of bitriquat adder.
steerC.v The steer token is regenerated from completeness of each digit
fulladd.v A 32 bit 2D digit piped adder is inserted in each flow path
TH22A gates in V parh to create an unbalancing delay

twoD_steer32G2.v delay removed
bitriquat.v steer with output of bitriquat adder.
steerC.v The steer token is regenerated from completeness of each digit
fulladd.v A 32 bit 2D digit piped adder is inserted in each flow path



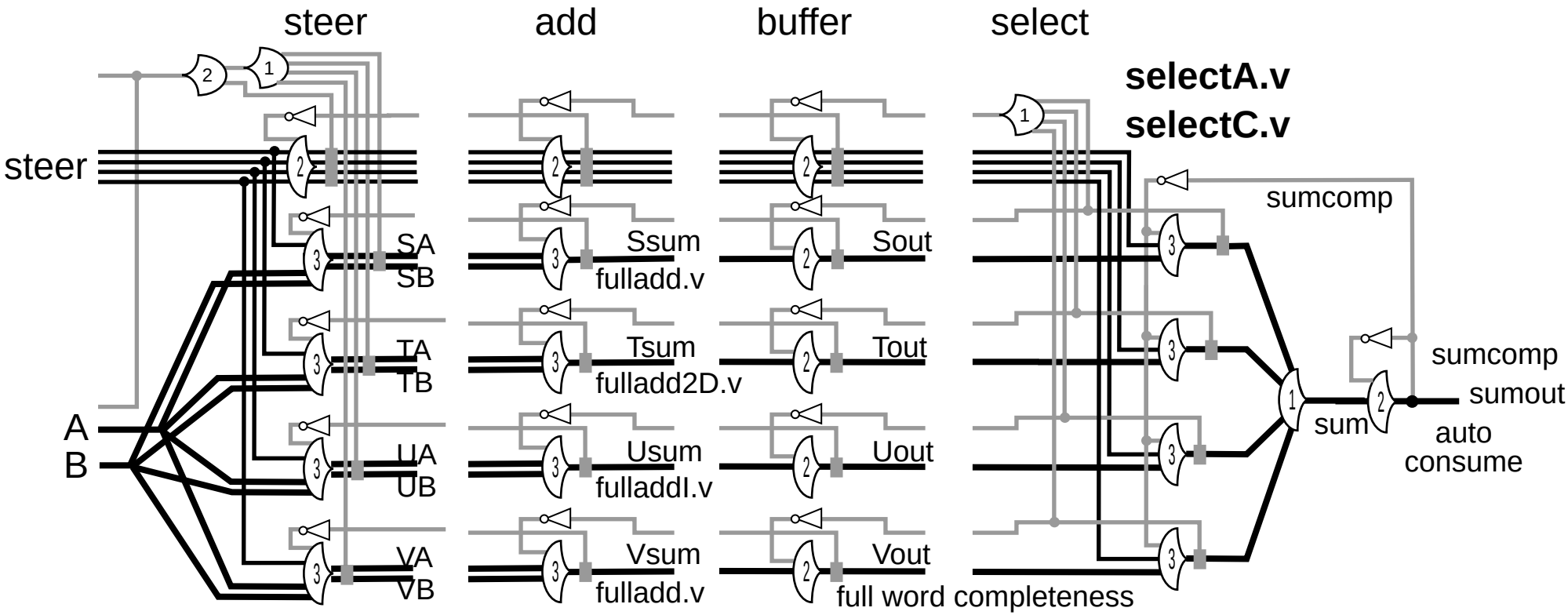
A Conditional Function Column Component

3 Different 32 bit Adders in three paths with different delay properties demonstrate a prototype ALU.

twoD_steer32K.v
bitriquat.v steer with output of bitriquat adder.
steerC.v The steer token is regenerated from completeness of each digit
fulladd.v A 32 bit 2D digit piped adder is inserted in each flow path
fulladd2D.v A 32 bit 2D piped adder is inserted in each flow path
fulladdI.v A 32 bit 2D integrated piped adder is inserted in each flow path
selectA.v select component to order flows into the OR converge

twoD_steer32K2.v adds full word completeness in path V additionally exercising the logical determination.

twoD_steer32K3.v uses **selectC.v**



```
steerdigit(A, steerin -> S, T, U, V, steerout){
flow [A, steerin] -> [{S, T, U, V}, steerout];
token {1:0} A, S, T, U, V;
token {toS, toT, toU, toV} steerin, steerout;
  [A, steerin/toS] -> S;
  [A, steerin/toT] -> T;
  [A, steerin/toU] -> U;
  [A, steerin/toV] -> V;
  steerin -> steerout;
close A <- [{S/#, T/#, U/#, V/#}, steerout/#];
close steerin <- [{S/#, T/#, U/#, V/#}, steerout/#];
close S <- ?/#;
close T <- ?/#;
close U <- ?/#;
close V <- ?/#;
close steerout <- ?/#;
}
```

```
steer2D(A, steerin -> S, T, U, V){
flow [A, steerin] -> {S, T, U, V};
token [31:0]{1:0} A, S, T, U, V;
token {toS, toT, toU, toV} steerin;
token [32:0]steerin steer;
path A/i<
  steerin -> steer/0;
  steerdigit (A/i, Steer/i -> S/i, T/i, U/i, V/i, steer/i+1)
  steer/i+1 -> ;
close A /i<- {S/i/#, T/i/#, U/i/#, V/i/#};
close steerin <- steer/0/#;
close S/i <- ?/i/#;
close T/i <- ?/i/#;
close U/i <- ?/i/#;
close V/i <- ?/i/#;
>
}
```

The Add Column Component

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

```
add32(A, B, carryin -> sum, carryout){
flow [A, B, carryin] -> [sum, carryout];
token [31:0]{1:0} A, B, sum;
token [32:0]{1:0} carry;
token {1:0} caryin, carryout;
token {toS, toT, toU, toV} steerin;
token [32:0]steerin steer;
path A/i<
  carryin -> carry/0;
  adddigit (A/i, B/i, carry/i -> sum/i, carry/i+1)
  cary/i+1 -> carryout;
close A /i<- {sum/i/#, carryout/#};
close A /i<- {sum/i/#, carryout/#};
close carryin <- carry/0/#;
close sum/i <- ?/i/#;
close carryout <- ?/#;
>
}
```

The Select Column Component

```
selectdigit(S, T, selectin -> A, selectout){
flow [{S, T}, selectin] -> [A, selectout];
token {1:0} A, S, T;
token {fromS, fromT} selectin, selectout;
  {[S, selectin/fromS], [T, selectin/fromT], [U, selectin/fromU],
[V, selectin/fromV]} -> A;
 selectin -> selectout;
close S <- [A/#, selectout/fromS];
close T <- [A/#, selectout/fromT];
close U <- [A/#, selectout/fromU];
close V <- [A/#, selectout/fromV];
close selectin <- [A/#, selectout/#];
close A <- ?/#;
close selectout <- ?/#;
```

```
select2D(S, T, U, V, selectin -> A){
flow [{S, T, U, V}, selectin] -> A;
token [31:0]{1:0} A, S, T, U, V;
token {fromS, fromT, fromU, fromV} selectin;
token [32:0]selectin select;
path A/i<
  selectin -> select/0;
  selectdigit (S/i, T/i, U/i, V/iA/i, select/i -> A/i, select/i+1)
  select/i+1 -> ;
close S /i<- A/i/#;
close T /i<- A/i/#;
close U /i<- A/i/#;
close V /i<- A/i/#;
close selectin <- select/0/#;
close A/i <- ?/i/#;
>
}
```

The Conditional Function Component

Composed from the previously defined components.

```
addpipe(A, B, steer -> sum){
flow [A, B] -> sum;
token [31:0]{1:0} A, B, sum;
token {toS, toT, toU, toV} steerin, steerin1, steerin2;
token {fromS, fromT, fromU, fromV} selectin;
token [31:0]{1:0} SA, TA, UA, VA, SB, TB, UB, VB;
token {1:0} Scarryin, Scarryout, Tcarryin, Tcarryout, Ucarryin, Ucarryout, Vcarryin, Vcarryout;
token [31:0]{1:0} Ssuma, Tsuma, Usuma, Vsuma, Ssum, Tsum, Usum, Vsum;
( -> steer);
steer2D(A, steer -> SA, TA, UA, VA);
steer2D(B, steer -> SB, TB, UB, VB);
(0 -> Scarryin);
add32(SA, SB, Scarryin -> Ssuma, Scarryout);
(Scarryout -> );
(0 -> Tcarryin);
add32(TA, TB, Tcarryin -> Tsuma, Tcarryout);
(Tcarryout -> );
(0 -> Ucarryin);
add32(UA, UB, Ucarryin -> Usuma, Ucarryout);
(Ucarryout -> );
(0 -> Vcarryin);
add32(VA, VB, Vcarryin -> Vsuma, Vcarryout);
(Vcarryout -> );
buffer2D(Ssuma -> Ssum);
buffer2D(Tsuma -> Tsum);
buffer2D(Usuma -> Usum);
buffer2D(Vsuma -> Vsum);
buffer(steerin -> steerin1);
buffer(steerin1 -> steerin2);
buffer(steerin2 -> selectin);
select2D(Ssum, Tsum, Usum, Vsum, selectin -> sum);
close A <- {SA/#, TA/#, UA/#, VA/#};
close B <- {SB/#, TB/#, UB/#, VB/#};
close sum <- ?/#;
}
```

```
buffer2D(A -> B){
flow A -> B;
token [31:0]{1:0} A, B;
path A/i<
 (A/i -> B/i)
close A /i <- B/i/#;
close B/i <- ?/i/#;
>
}
```

The fullword steer Component

```
steerdigit(A, steerin -> S, T, U, V){
flow [A, steerin] -> {S, T, U, V};
token {1:0} A, S, T, U, V;
token {toS, toT, toU, toV} steerin, steerout;
  [A, steerin/toS] -> S;
  [A, steerin/toT] -> T;
  [A, steerin/toU] -> U;
  [A, steerin/toV] -> V;
close A <- {S/#, T/#, U/#, V/#};
close steerin <- {S/#, T/#, U/#, V/#};
close S <- ?/#;
close T <- ?/#;
close U <- ?/#;
close V <- ?/#;
}
```

```
steerfull(A, steerin -> S, T, U, V){
flow [A, steerin] -> {S, T, U, V};
token [31:0]{1:0} A, S, T, U, V;
token {toS, toT, toU, toV} steerin;
token steer [32:0]steerin;
path A/i<
 steerdigit (A/i, Steerin -> S/i, T/i, U/i, V/i)
>
close A <- {S/#, T/#, U/#, V/#};
close steerin <- {S/#, T/#, U/#, V/#};
close S <- ?/#;
close T <- ?/#;
close U <- ?/#;
close V <- ?/#;
}
```