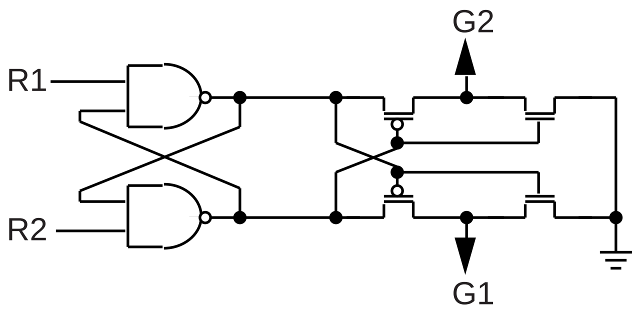
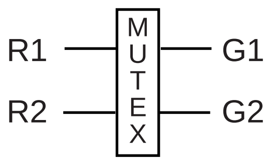
Sandbox 8

Arbitration

The MUTEX

Mutual exclusion of independent signals

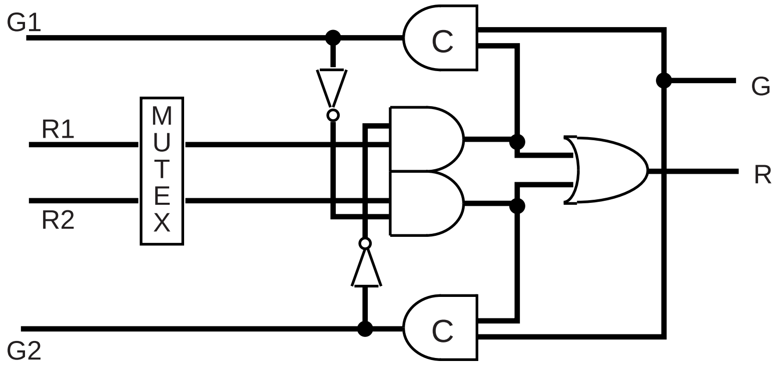




The Arbiter

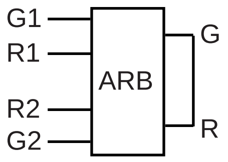
Rs can arrive at any time
Only one G at a time

The R - G behavior forms a monotonic oscillation

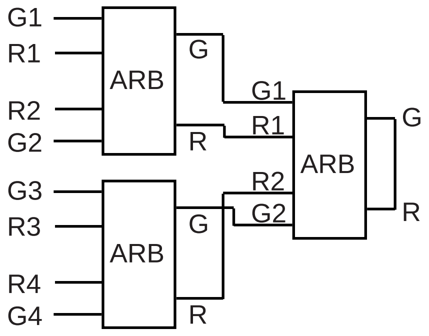


a. Standard version

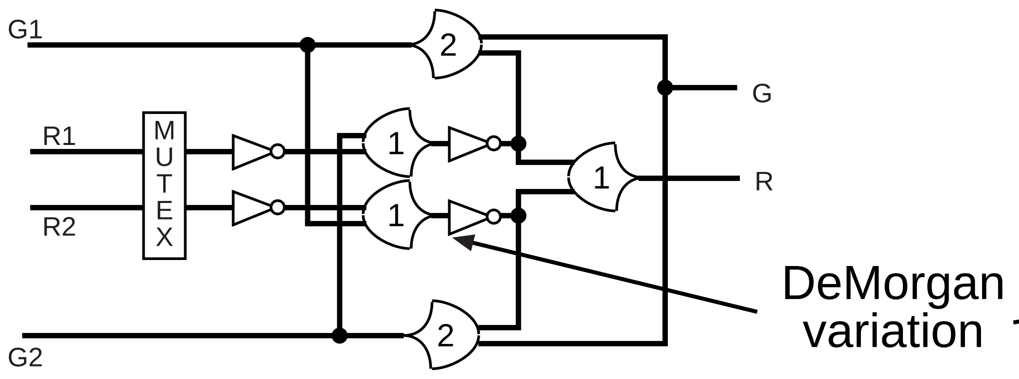
Arbitrate 2 signals



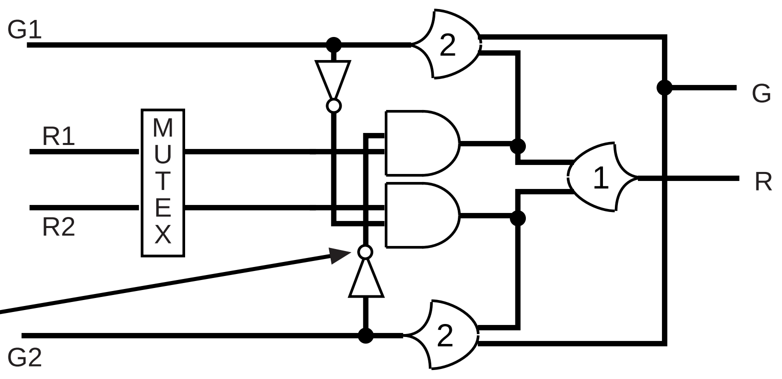
Arbitrate 4 signals - arbitrate the arbitrations



The NCL Arbiter



a. NCL version 1



b. NCL version 2

R1 and G1 behave as an oscillation
R2 and G2 behave as an oscillation

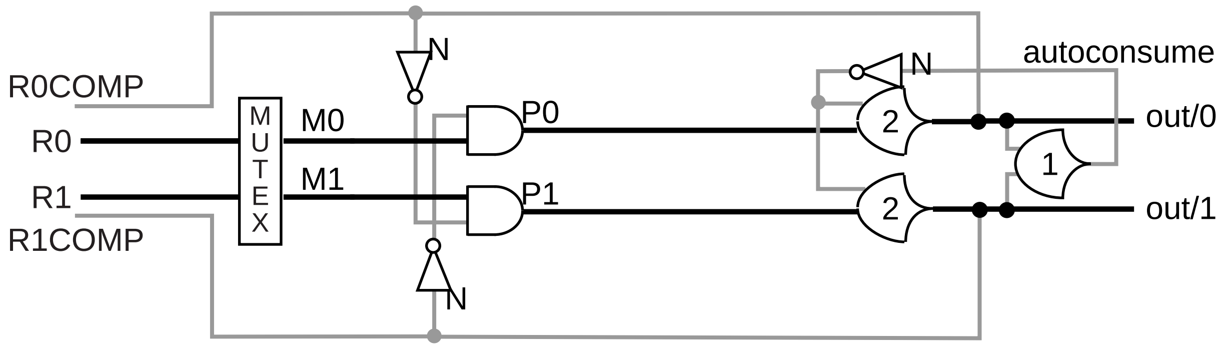
Two independent rails to one dual rail flow

Verilog code

testbench	**arbD.v**
circuit	**freetodual.v**
	make arbD

Flow arbitration component

freetodual component



Flow Specification

```
freetodual(R0, R1 -> dual){
flow {R0, R1} -> dual;
token {1:0} dual;
token R0, R1, M0, M1, P0, P1;
MUTEX(R0, R1 => M0, M1);
AND(M0, NOT(dual/1) -> P0);
AND(M1, NOT(dual/0) -> P1);
P0 -> dual/0;
P1 -> dual/1;
close R0 <- dual/0;
close R1 <- dual/1;
// autoconsume
close dual <- {dual/0, dual/1};
}
```

Three independent rails to one three rail variable

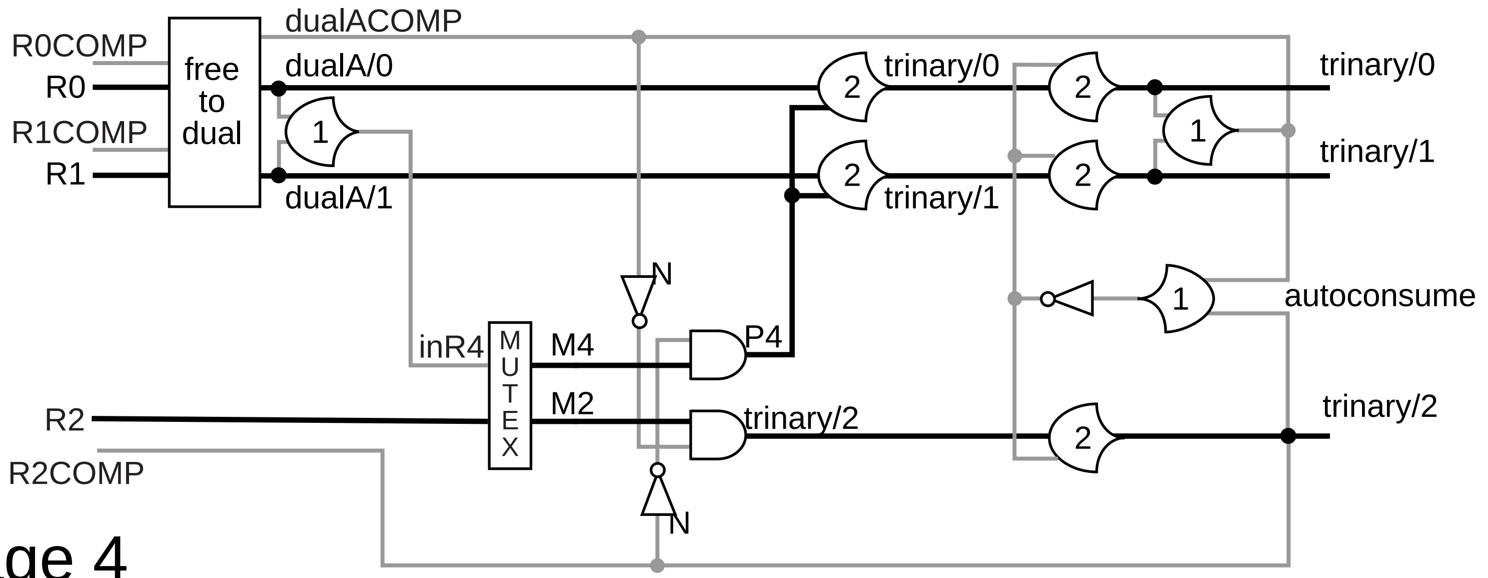
Flow Specification

```
freetotrinary(R0, R1, R2 -> trinary){
flow {R0, R1, R2} -> trinary;
token {2-0} trinary;
token {1:0} dualA,
token R2, M2, P2;
token R4

freetodual(R0, R1 -> dualA);

{dualA/0, dualA/1} -> inR4;
MUTEX(inR4, R2 => M4, M2);
AND(M4, NOT(trinary/2) -> P4);
AND(M2, NOT(dualACOMP) -> trinary/2);
[P4, dualA/0] -> trinary/0;
[P4, dualA/1] -> trinary/1;
P2 -> trinary/2;
{trinary/0, trinary/1} -> dualACOMP;

close dualA <- {trinary/0, trinary/1};
close inR2 <- trinary/2;
// auto consume
close trinary <- {trinary/0, trinary/1, trinary/2};
}
```

Verilog code

testbench	**arbT.v**
circuit	**freetotrinary.v**
	make arbT

freetotrinary component



Page 4

Four inependent rails to one quad rail variable

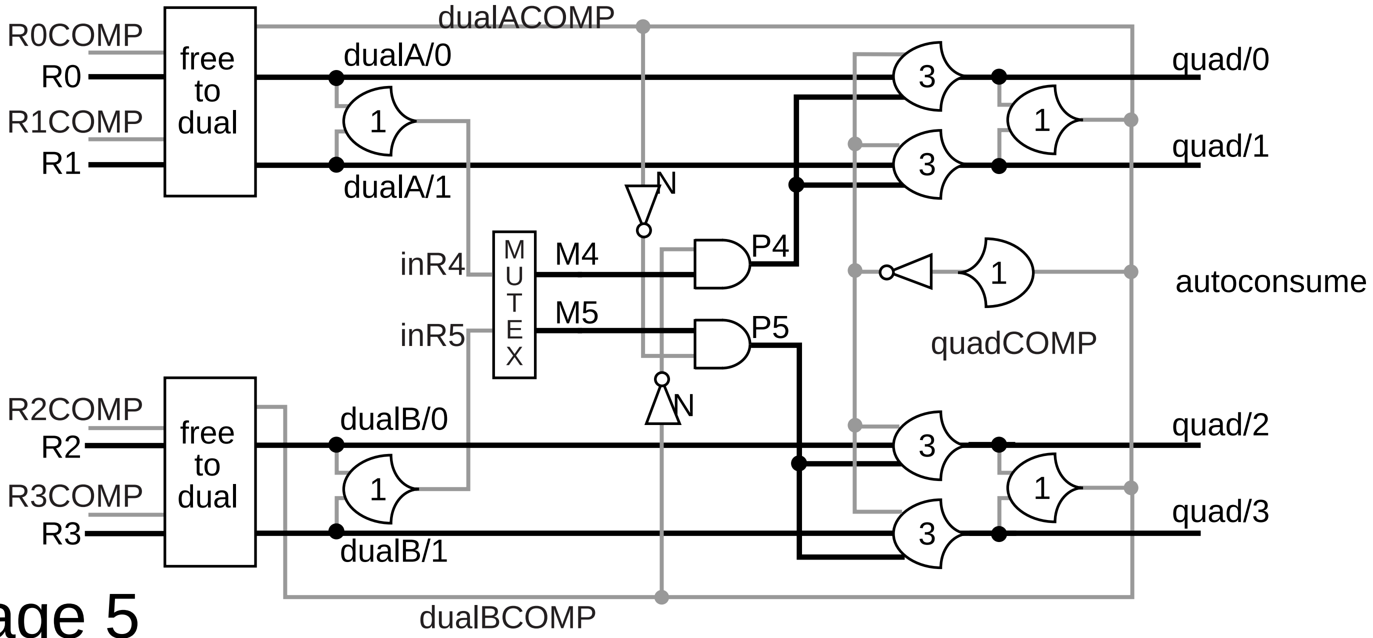
Verilog code

testbench arbQ.v
circuit freetoquad.v
make arbQ

Flow Specification

```
freetoquad(dualA, dualB -> quad){
flow {dualA, dualB} -> quad;
token {3-0} quad;
token {1:0} dualA, dualB;
token R0, R1, R2, R3, R4, R5;
token M4, M5, P4, P5;
freetodual(R0, R1 -> dualA);
freetodual(R2, R3 -> dualB);
{dualA/0, dualA/1} -> R4;
{dualB/0, dualB/1} -> R5;
MUTEX(R4, R51 => M4, M5);
AND(M4, NOT(dualACOMP) -> P4);
AND(M5, NOT(dualBCOMP) -> P5;
[dualA/0, P4] -> quad/0;
[dualA/1, P4] -> quad/1;
[dualB/0, P5] -> quad/2;
[dualB/1, P5] -> quad/3;
{quad/0, quad/1} -> dualACOMP
{quad/2, quad/3} -> dualBCOMP
close dualA <- dualACOMP;
close dualB <- dualBCOMP;
// autoconsume
close quad <- {dualACOMP, dualBCOMP};
}
```

freetoquad component



Two inependent dual rail flows to one dual rail variable

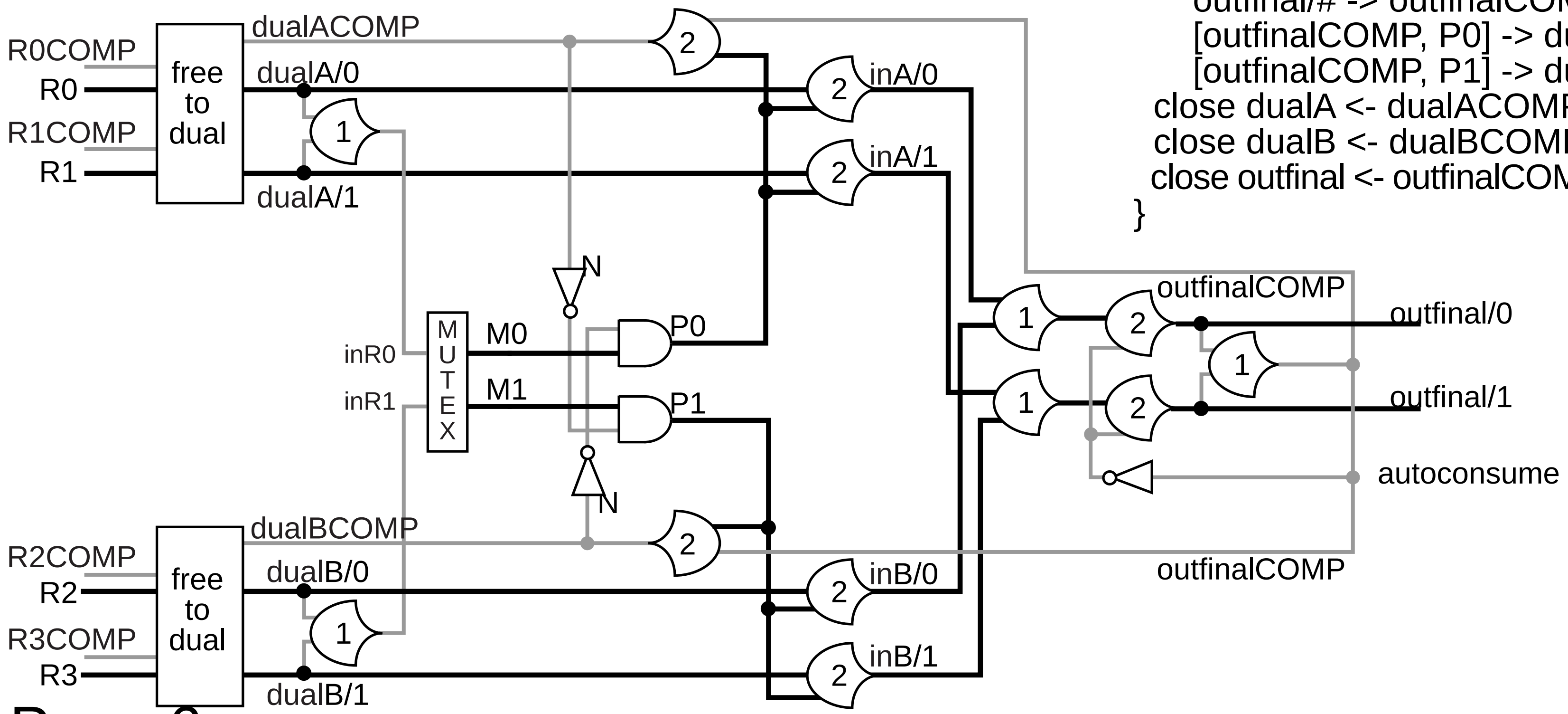
Flow Specification

```
freedualtodual(dualA, dualB -> outfinal){
  flow {dualA, dualB} -> outfinal;
    token {1:0} dualA, dualB, outfinal;
    token inR0, inR1, M0, M1, P0, P1;

// sequences the input duals into a single output
dual
      {dualA/0, dualA/1} -> inR0;
      {dualB/0, dualB/1} -> inR1;
      MUTEX(inR0, inR1 => M0, M1);
      AND(M0, NOT(dualACOMP) -> P0);
      AND(M1, NOT(dualBCOMP) -> P1);
      [dualA/0, P0] -> inA/0;
      [dualA/1, P0] -> inA/1;
      [dualB/0, P1] -> inB/0;
      [dualB/1, P1] -> inB/1;
      [outfinalCOMP, {inA/0, inB/0}] -> outfinal/0;
      [outfinalCOMP, {inA/1, inB/1}] -> outfinal/1;
      outfinal/# -> outfinalCOMP;
      [outfinalCOMP, P0] -> dualACOMP;
      [outfinalCOMP, P1] -> dualBCOMP;
  close dualA <- dualACOMP;
  close dualB <- dualBCOMP;
  close outfinal <- outfinalCOMP; // autoconsume
}
```

Verilog code

testbench	**arbDD.v**
circuit	**freedualtodual.v**
	make arbDD

freedualtodual component



R0COMP
R0
R1COMP
R1
free to dual
dualACOMP
dualA/0
dualA/1
1
2
N
inR0
inR1
MUTEX
M0
M1
P0
P1
inA/0
inA/1
R2COMP
R2
R3COMP
R3
free to dual
dualBCOMP
dualB/0
dualB/1
inB/0
inB/1
outfinalCOMP
outfinal/0
outfinal/1
autoconsume