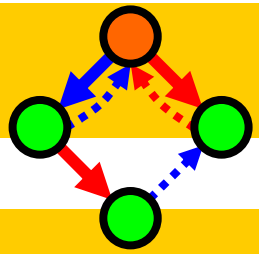
I cannot do it without comp[u]ters.
William Shakespeare – The Winter's Tale.

Artificial Biochemistry



Dipolin Circuits
Thomas Circuits

Luca Cardelli

Microsoft Research

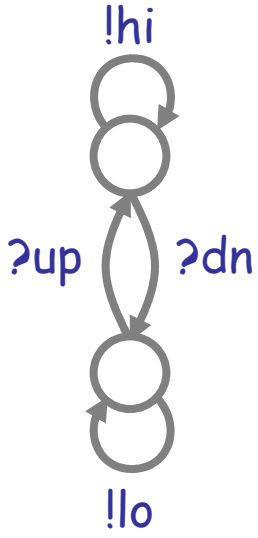
The Microsoft Research - University of Trento
Centre for Computational and Systems Biology

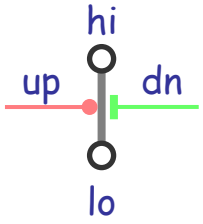
Trento, 2006-05-22..26

www.luca.demon.co.uk/ArtificialBiochemistry.htm

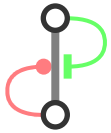
Dipolins

Dipolins

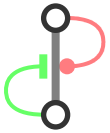




Dipolin

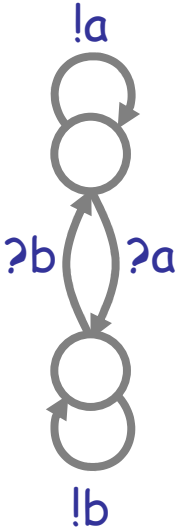


Less

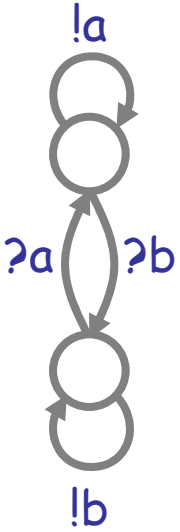


More

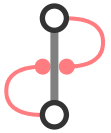
(seen as a population: each not interacting with self)



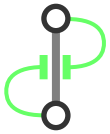
(Celebrity)



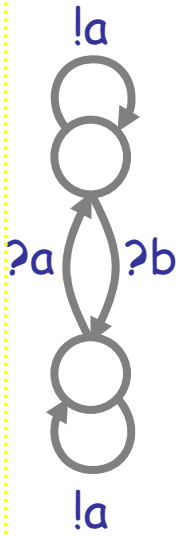
(Groupie)

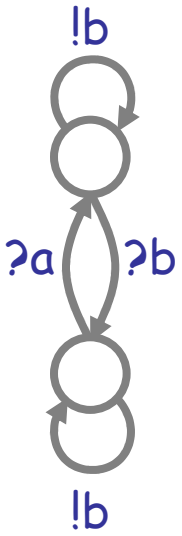


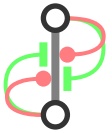
Hi
(whole population is in high state)



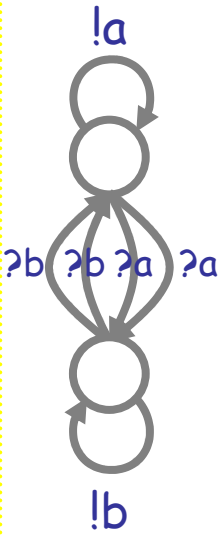
Lo
(whole population is in low state)



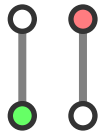




Both
(population is in both states)



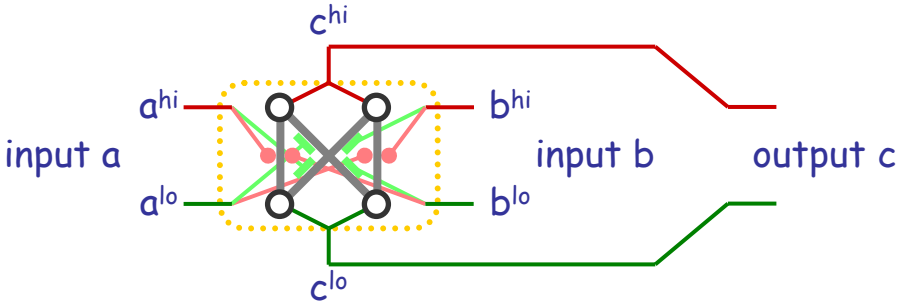
A dipolin is often drawn as an undirected up-down bipartite graph to implicitly indicate edge (up-) direction.
The current state is indicated by red (high) or green (low).



Dipolin Boolean Gates

A "dipolin boolean signal" consists of either the *presence* of a certain pole (designated "hi") in current state, or the *presence* of another pole (designated "lo") in current state, and not both.

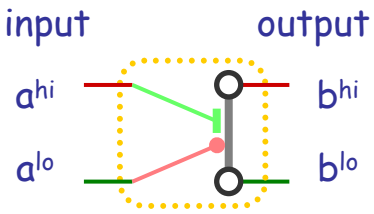
For "boolean" polin gates and circuits we need to consistently distinguish between "high poles" and "low poles".



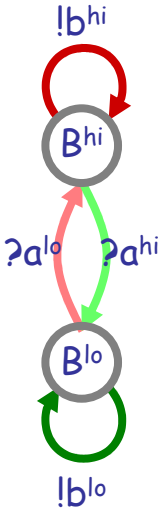
Each gate or circuit is a *population* that reacts to other populations.

(But a lone gate may work too.)

Dipolin Not Gate

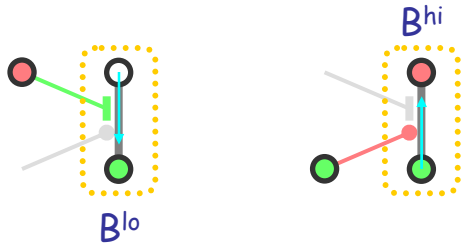


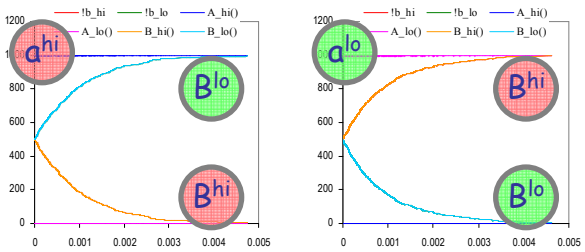
signal from a high state
signal from a low state
raising the state
lowering the state



For "boolean" polin gates and circuits we need to consistently distinguish between "high poles" and "low poles".

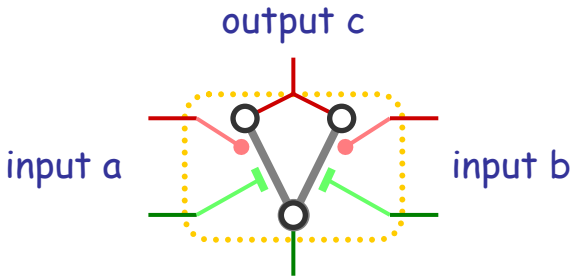
E.g. the inhibition input of a not gate should always be connected to a high pole, otherwise it would not behave like a not gate. This is represented in the diagram by drawing the starting point of the inhibition input higher, and the starting point of the excitation input lower.

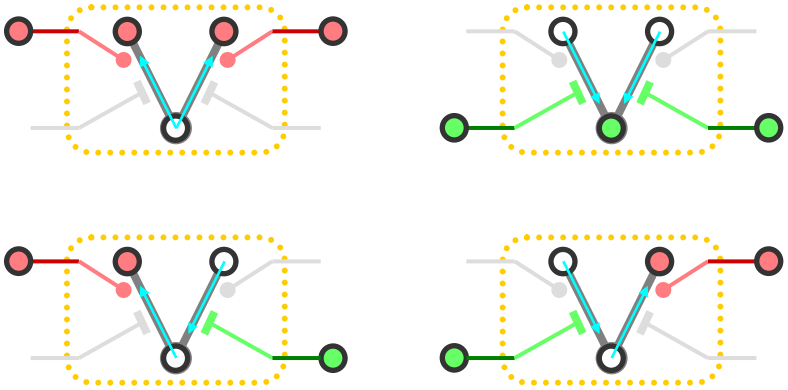


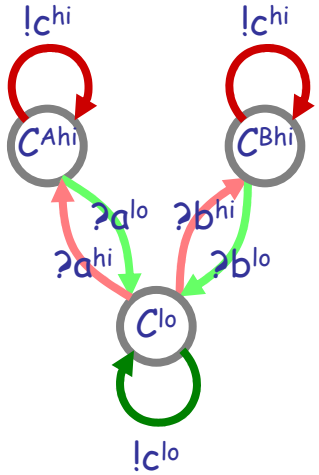



```
directive sample 0.005 10000
directive plot !b_hi; !b_lo;
A_hi(); A_lo(); B_hi(); B_lo()

new a_hi@1.0:chan() new a_lo@1.0:chan()
new b_hi@1.0:chan() new b_lo@1.0:chan()
new c_hi@1.0:chan() new c_lo@1.0:chan()

let B_hi() = do !b_hi;B_hi() or ?a_hi;B_lo()
and B_lo() = do !b_lo;B_lo() or ?a_lo;B_hi()

let A_hi() = !a_hi;A_hi()
and A_lo() = !a_lo;A_lo()

run 500 of (B_lo() | B_hi())
run 1000 of A_hi()
(* run 1000 of A_lo() *)
```

Dipolin Or Gate





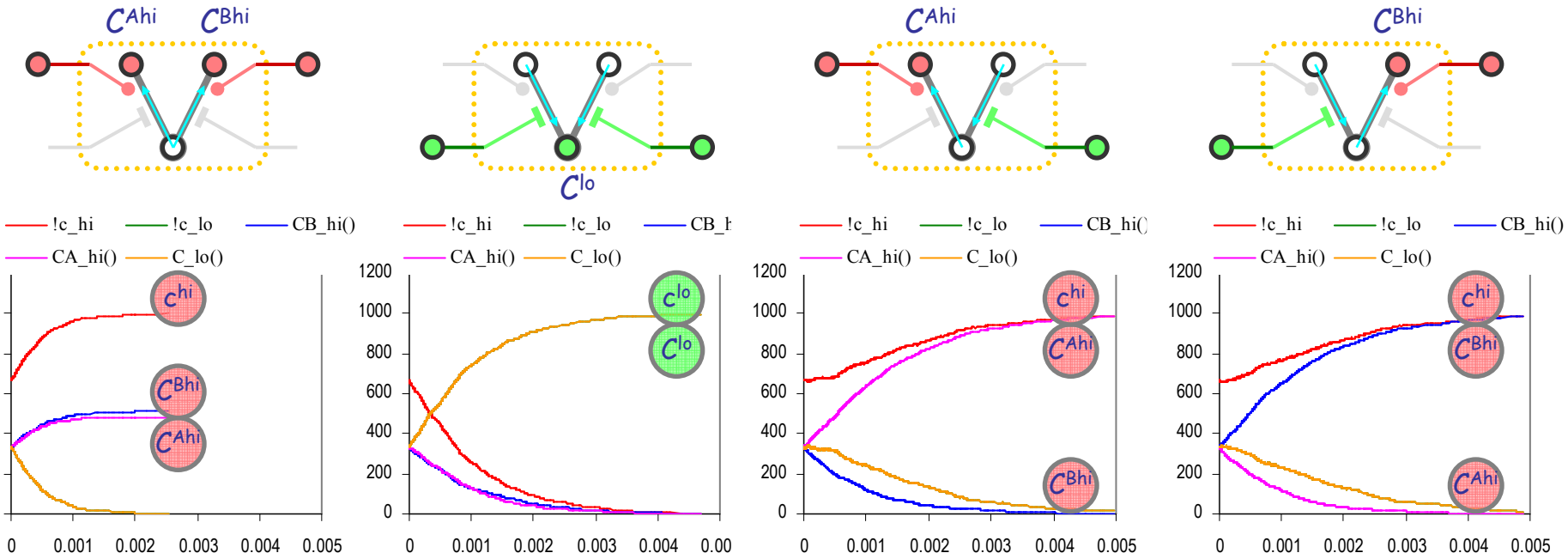


The gate is a population that reacts to two other populations.

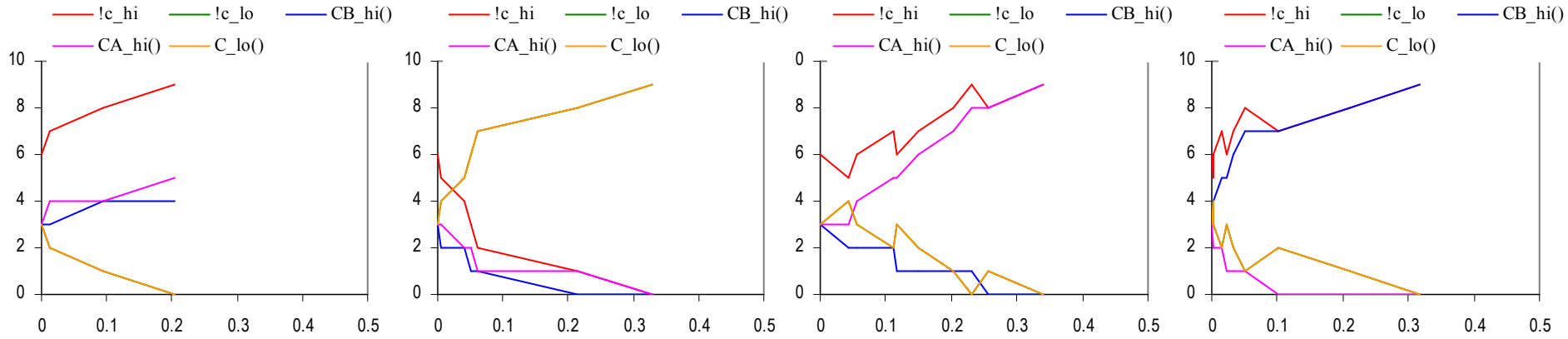
(But a lone gate works too.)

```
directive sample 0.005 10000
directive plot !c_hi; !c_lo; CB_hi(); CA_hi(); C_lo()

new a_hi@1.0:chan() new a_lo@1.0:chan()
new b_hi@1.0:chan() new b_lo@1.0:chan()
new c_hi@1.0:chan() new c_lo@1.0:chan()

let C_lo() = do !c_lo;C_lo() or ?a_hi;CA_hi() or ?b_hi;CB_hi()
and CA_hi() = do !c_hi;CA_hi() or ?a_lo;C_lo()
and CB_hi() = do !c_hi;CB_hi() or ?b_lo;C_lo()

let A_lo() = !a_lo;A_lo()   and   A_hi() = !a_hi;A_hi()
let B_lo() = !b_lo;B_lo()   and   B_hi() = !b_hi;B_hi()

run 333 of (C_lo() | CA_hi() | CB_hi())
run 1000 of (A_hi() | B_hi())
(* run 1000 of (A_lo() | B_lo()) *)
(* run 1000 of (A_lo() | B_hi()) *)
(* run 1000 of (A_hi() | B_lo()) *)
```

Or Gate Plots



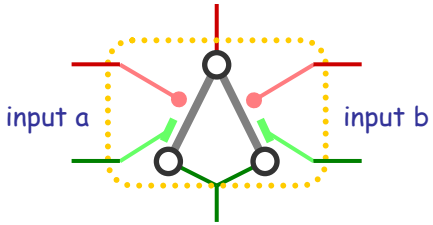
Initially: 333 Or gates in each of the three states; 1000 of each input (not plotted).



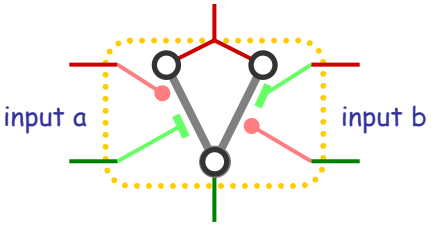
Initially: ~100x less (3 Or gates in each of the three states; 10 of each input). ~ 100x slower.

Other Dipolin Gates

From Or gate: flip the middle or flip either input.

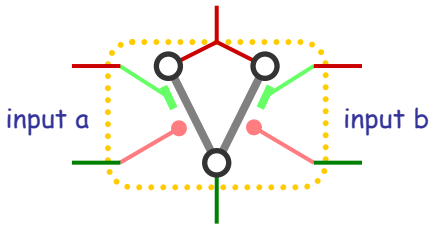


a And b

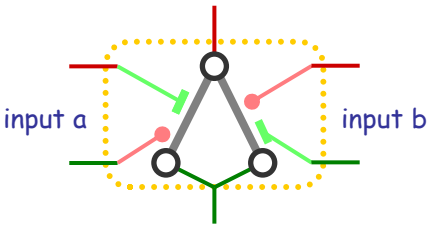


b Imply a

00-1
01-0
10-1
11-1

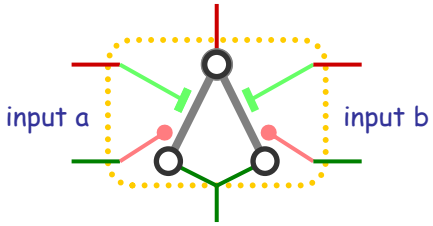


a Nand b

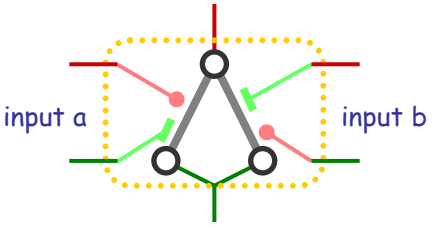


b Not a

00-0
01-1
10-0
11-0

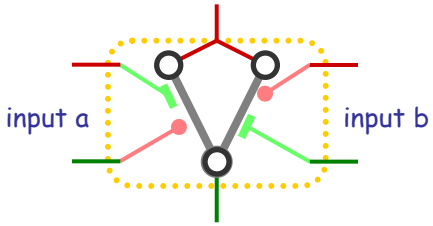


a Nor b



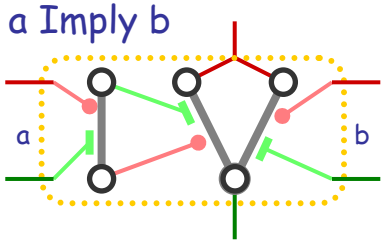
a Not b

00-0
01-0
10-1
11-0



a Imply b

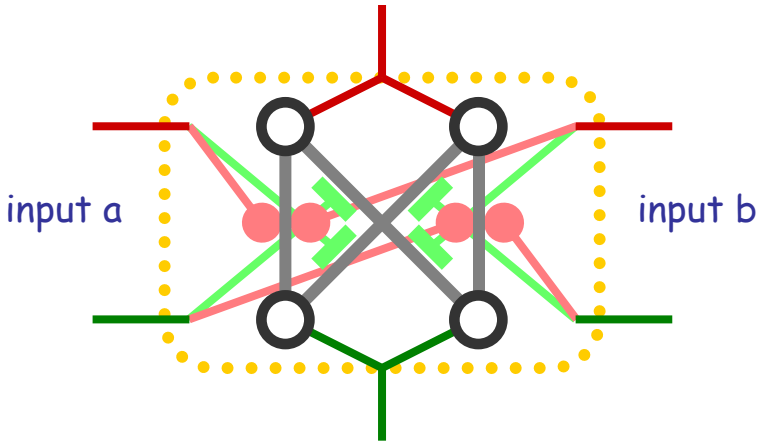
a Imply b

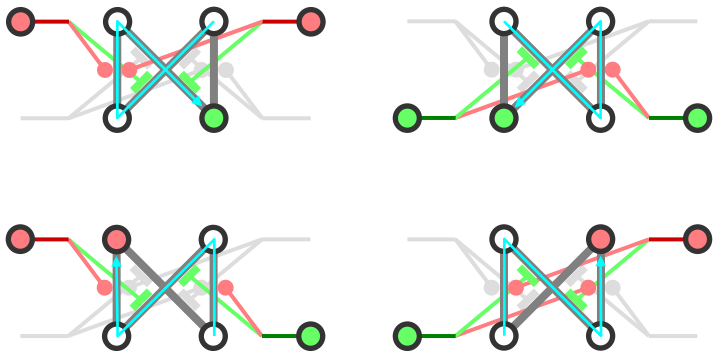


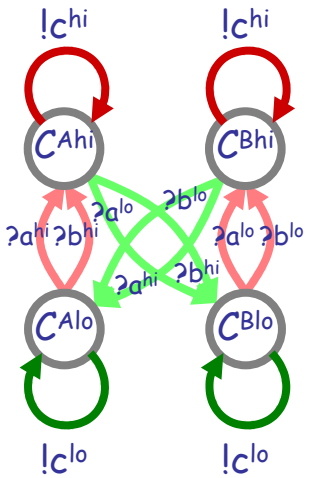
It is very convenient to "encapsulate the wiring" when building networks of gates. So that the hi output of one gate is always connected to a up input of another (and lo to dn).

2006-05-26

Dipolin Xor Gate

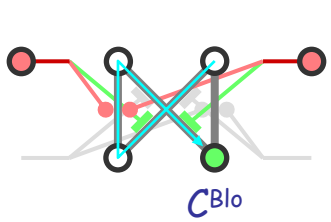


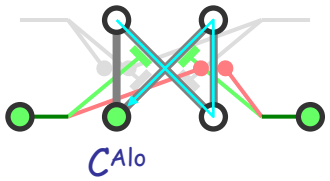


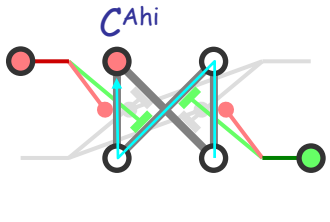


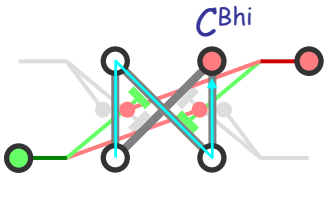

```
directive sample 0.005 10000
directive plot !c_hi; !c_lo; CB_hi(); CA_hi(); CB_lo(); CA_lo()

new a_hi@1.0:chan() new a_lo@1.0:chan()
new b_hi@1.0:chan() new b_lo@1.0:chan()
new c_hi@1.0:chan() new c_lo@1.0:chan()

let CA_lo() = do !c_lo;CA_lo() or ?a_hi;CA_hi() or ?b_hi;CA_hi()
and CA_hi() = do !c_hi;CA_hi() or ?a_lo;CB_lo() or ?b_hi;CB_lo()
and CB_lo() = do !c_lo;CB_lo() or ?a_lo;CB_hi() or ?b_lo;CB_hi()
and CB_hi() = do !c_hi;CB_hi() or ?a_hi;CA_lo() or ?b_lo;CA_lo()

let A_lo() = !a_lo;A_lo()   and   A_hi() = !a_hi;A_hi()
let B_lo() = !b_lo;B_lo()   and   B_hi() = !b_hi;B_hi()

run 250 of (CA_lo() | CB_lo() | CA_hi() | CB_hi())
run 1000 of (A_lo() | B_lo())
(* run 1000 of (A_hi() | B_hi()) *)
(* run 1000 of (A_hi() | B_lo()) *)
(* run 1000 of (A_lo() | B_hi()) *)
```

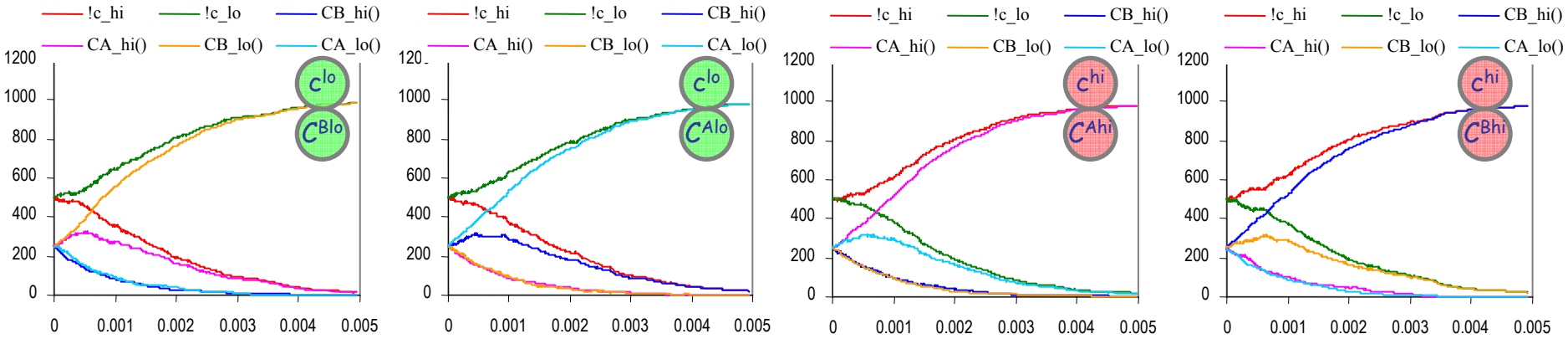
Xor Gate Plots



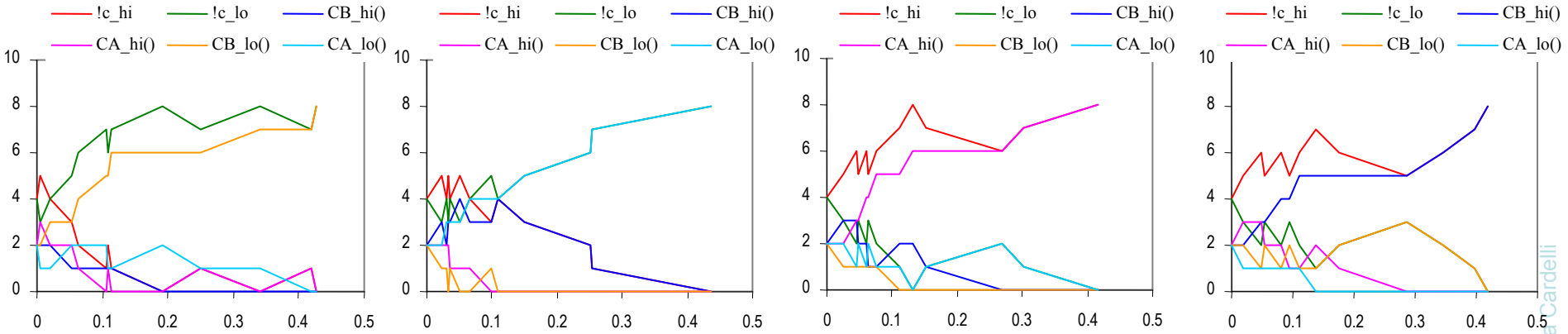








Initially: 250 Xor gates in each of the four states; 1000 of each input (not plotted).



Initially: ~100x less (2 Xor gates in each of the four states; 8 of each input). ~ 100x slower.

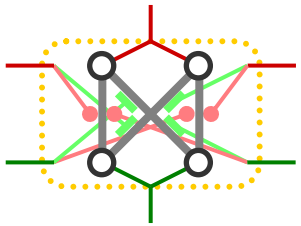
Aux

Dipolin Iff Gate

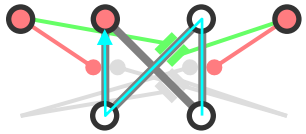
Aux

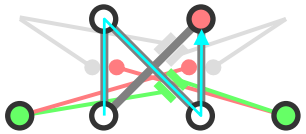
Flip the right input of Xor gate.

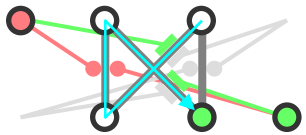
input a

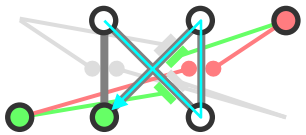


input b

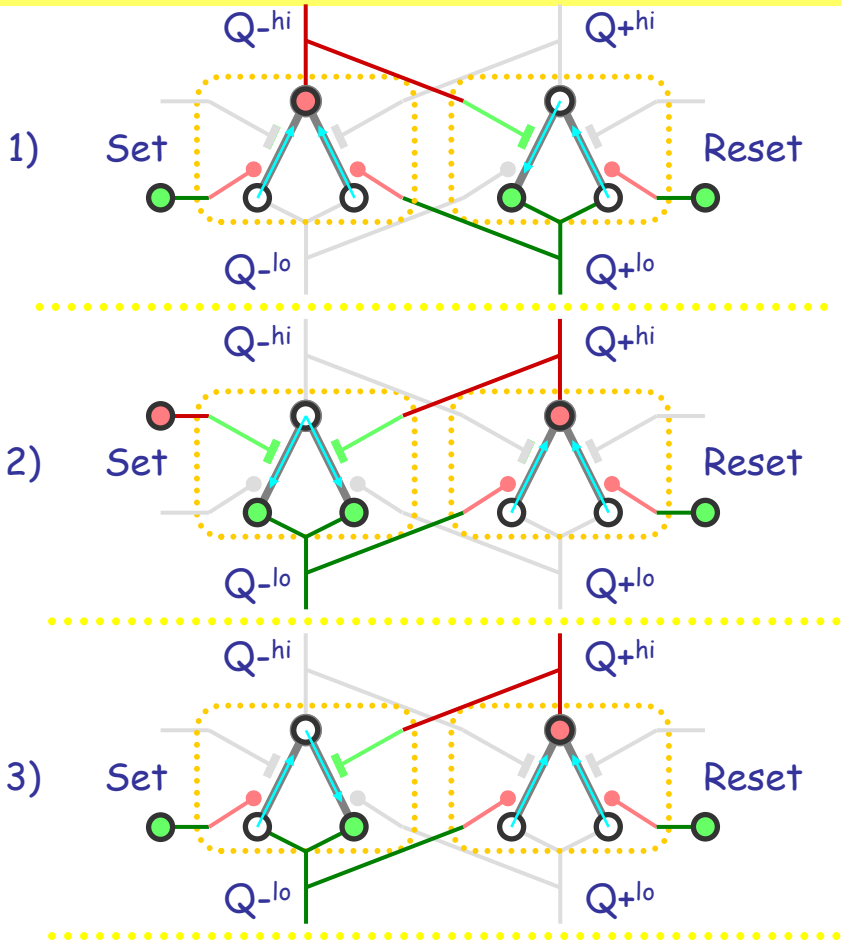


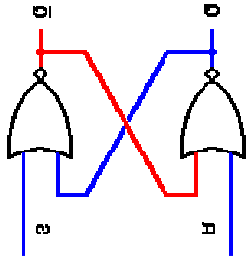






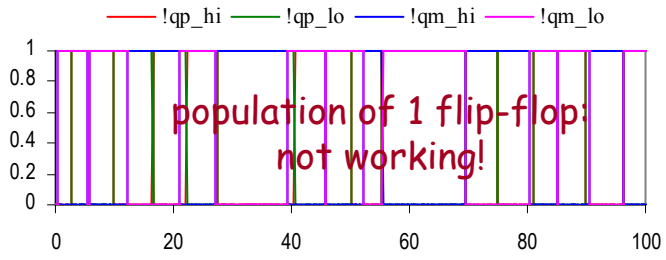
Dipolin Flip-flop



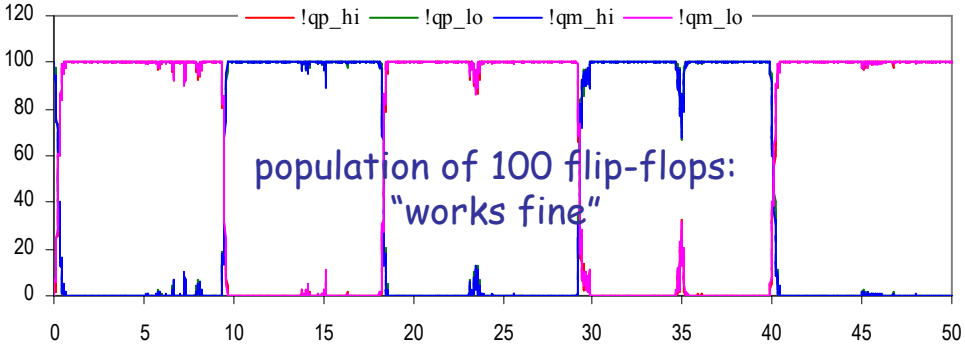


RS Nor Latch

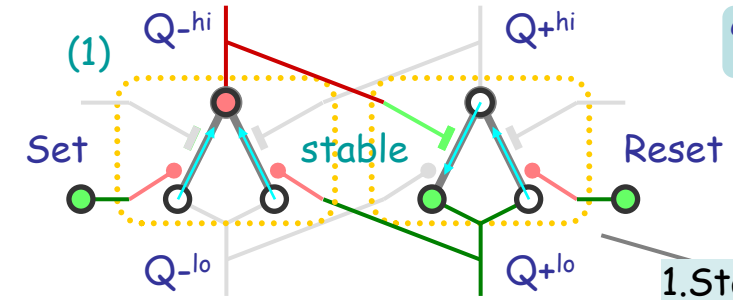
```
directive sample 50.0 10000
directive plot !qp_hi; !qp_lo; !qm_hi; !qm_lo

let Nor_hi(a_dn:chan,a_up:chan,b_dn:chan,b_up:chan,c_lo:chan,c_hi:chan) =
  do !c_hi; Nor_hi(a_dn,a_up,b_dn,b_up,c_lo,c_hi)
  or ?a_dn; NorA_lo(a_dn,a_up,b_dn,b_up,c_lo,c_hi)
  or ?b_dn; NorB_lo(a_dn,a_up,b_dn,b_up,c_lo,c_hi)
and NorA_lo(a_dn:chan,a_up:chan,b_dn:chan,b_up:chan,c_lo:chan,c_hi:chan) =
  do !c_lo; NorA_lo(a_dn,a_up,b_dn,b_up,c_lo,c_hi)
  or ?a_up; Nor_hi(a_dn,a_up,b_dn,b_up,c_lo,c_hi)
and NorB_lo(a_dn:chan,a_up:chan,b_dn:chan,b_up:chan,c_lo:chan,c_hi:chan) =
  do !c_lo; NorB_lo(a_dn,a_up,b_dn,b_up,c_lo,c_hi)
  or ?b_up; Nor_hi(a_dn,a_up,b_dn,b_up,c_lo,c_hi)

new set_lo@1.0:chan new set_hi@1.0:chan
new ret_lo@1.0:chan new ret_hi@1.0:chan
new qp_lo@1.0:chan new qp_hi@1.0:chan
new qm_lo@1.0:chan new qm_hi@1.0:chan

let LatchLft() =
  ( Nor_hi(set_hi,set_lo,qp_hi,qp_lo,qm_lo,qm_hi)
  | NorA_lo(qm_hi,qm_lo,ret_hi,ret_lo,qp_lo,qp_hi))

let clock(t:float, tick:chan) =
  (val ti = t/100.0 val d = 1.0/ti
   let step(n:int) = if n<=0 then !tick; clock(t,tick) else delay@d; step(n-1)
   run step(100))

let s1(tick:chan) = do !set_hi; s1(tick) or !ret_lo; s1(tick) or ?tick; s2(tick)
and s2(tick:chan) = do !set_lo; s2(tick) or !ret_lo; s2(tick) or ?tick; s3(tick)
and s3(tick:chan) = do !set_lo; s3(tick) or !ret_hi; s3(tick) or ?tick; s4(tick)
and s4(tick:chan) = do !set_lo; s4(tick) or !ret_lo; s4(tick) or ?tick; s1(tick)

new tick:chan   run (clock(5.0, tick) | s1(tick))   run 100 of LatchLft()
```



set-up;set-dn;ret-up;red-dn every ~5t
Ok on set-up&ret-up, but why the instability on set-dn&ret-dn?

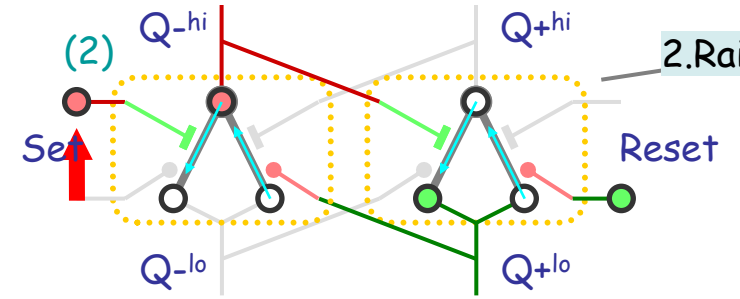


Lone Flip-flop

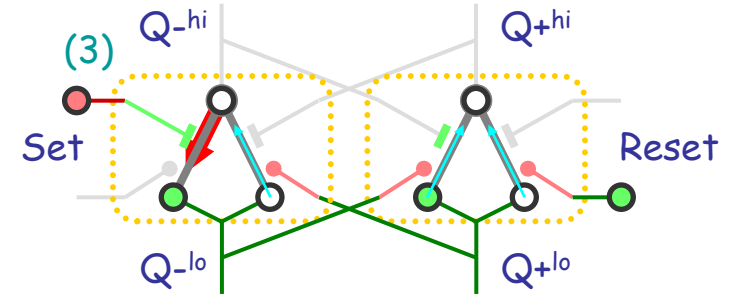


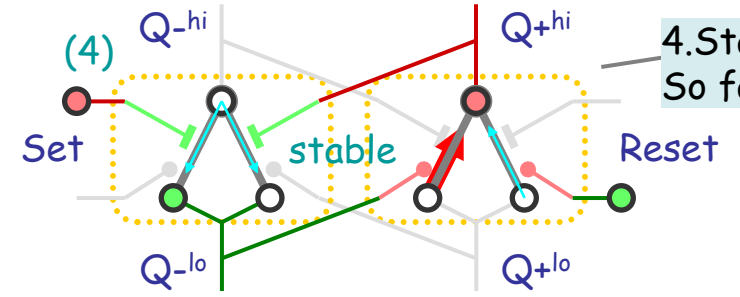
Consider a set-up;set-dn cycle on a population of just 1 flip-flop

1.Stable Q^{-hi}



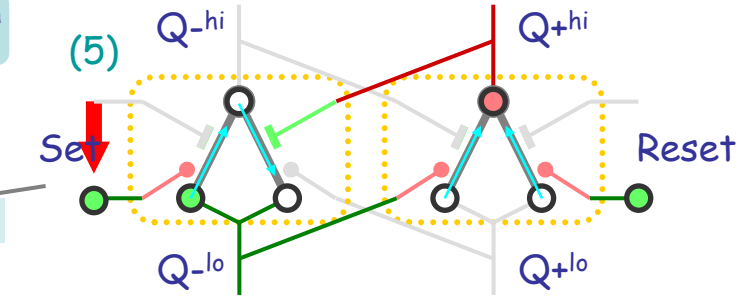
2.Raise Set

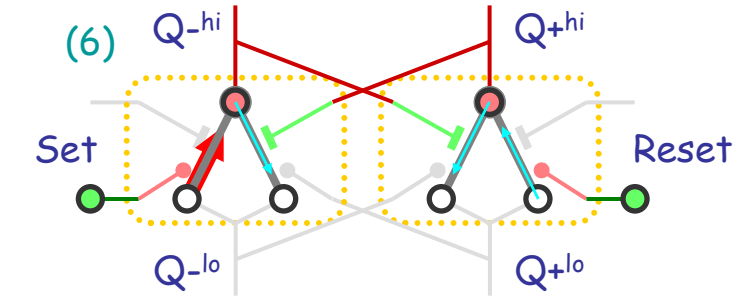




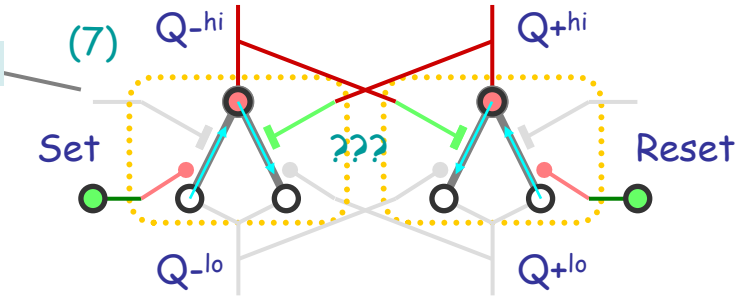
4.Stable Q^{+hi}
So far so good!

5.Lower Set





7.Back to Q^{-hi}?

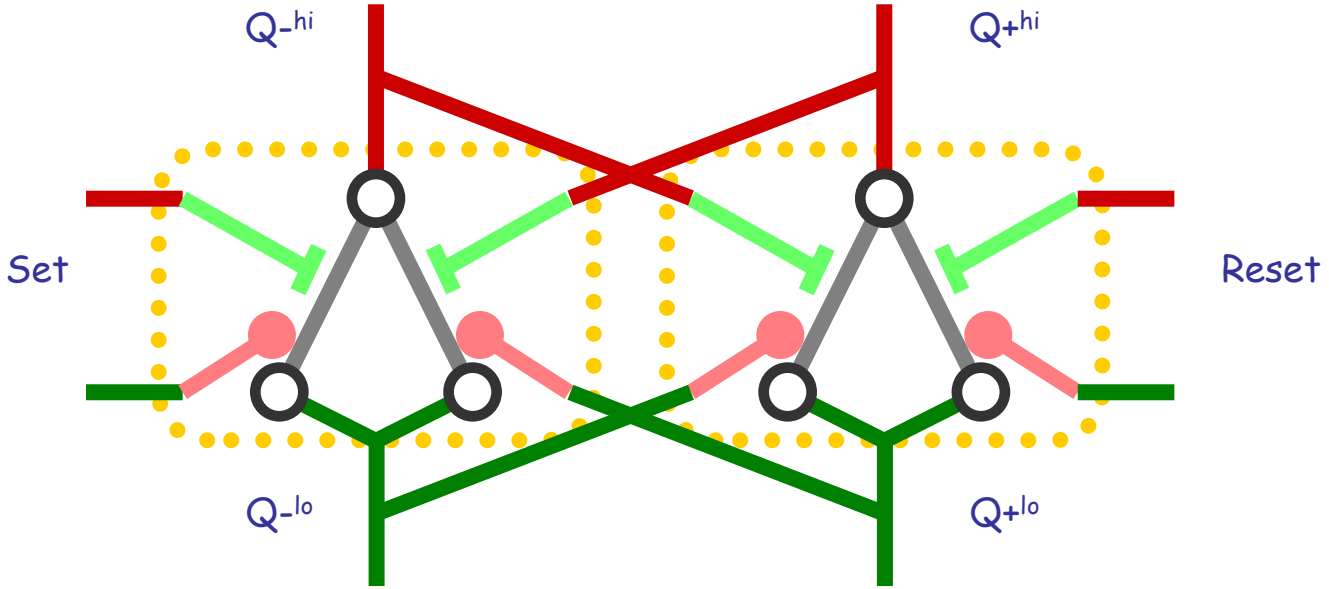


Oops, (7) is a perfectly symmetrical state that can go back to (1) in one step! There is no reason for the (expected) left transition to be taken, except that there are usually *more* Q^{+hi} states pushing on it.

The flip-flop works "more reliably" as a population!

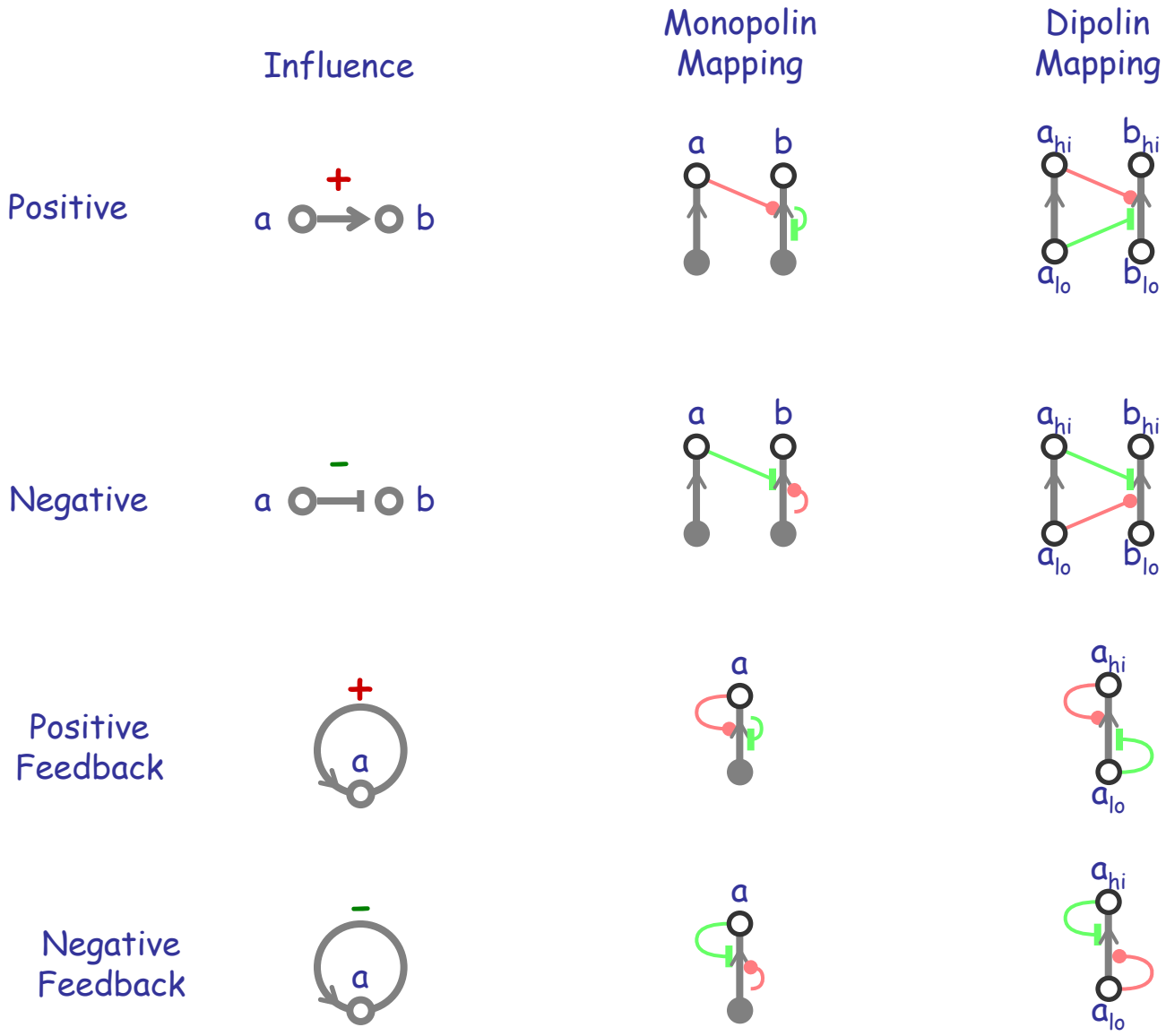
Aux

Aux



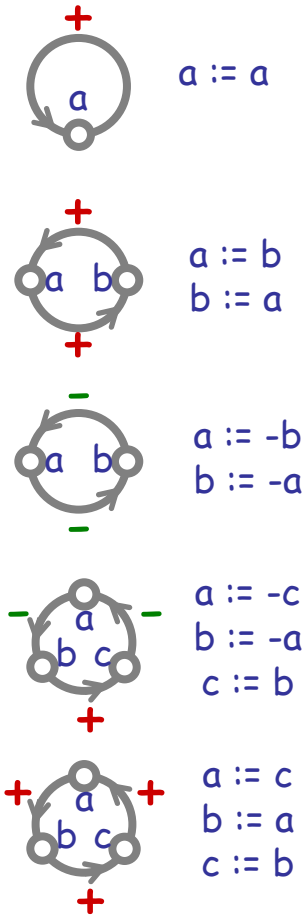
Thomas Circuits

Influence Diagrams



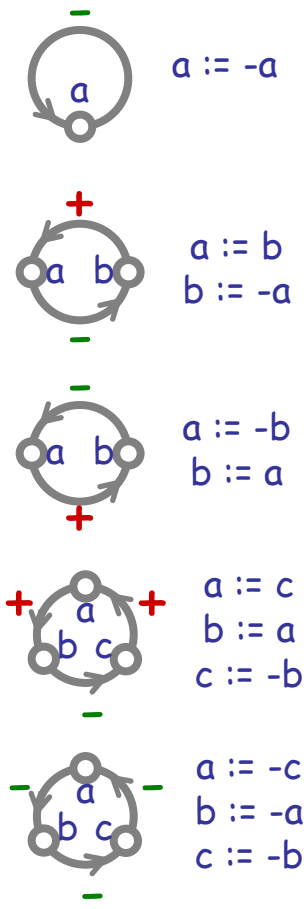
Thomas Circuits

Positive



Necessary condition for
Multistability

Negative



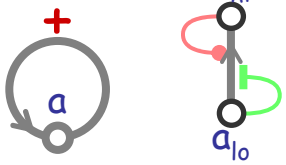
Necessary condition for
Periodicity
(stable state, limit cycle,
or chaotic attractor)

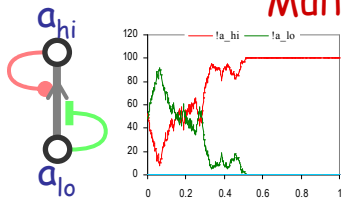
Rene Thomas circuits are about *inverting* or *restoring* polarities. A negative influence maps high to lowering and low to raising. A positive influence maps high to rising and low to lowering.

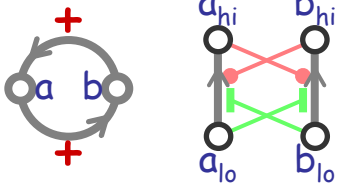
Dipolin Thomas Circuits

Positive

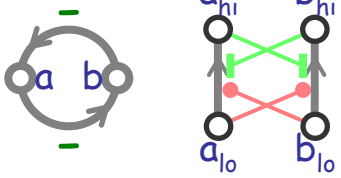
Multistable



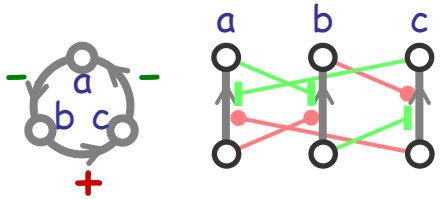


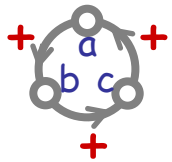


2 stable states:
a^{hi}-b^{hi} and a^{lo}-b^{lo}



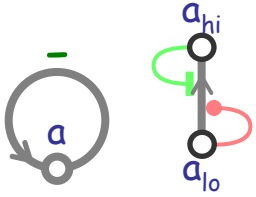
2 stable states:
a^{hi}-b^{lo} and a^{lo}-b^{hi}

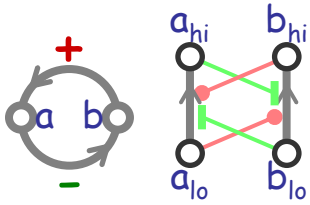


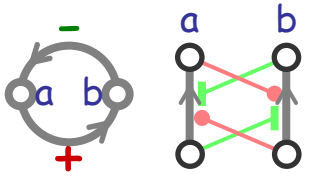


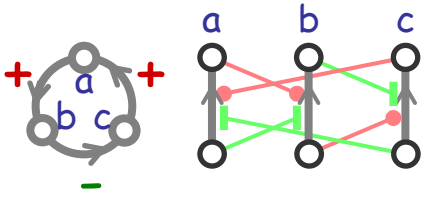
Negative

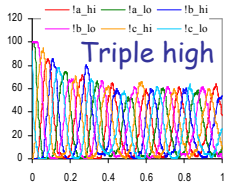
Periodic

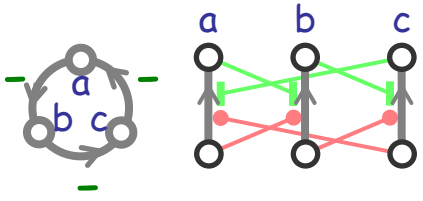


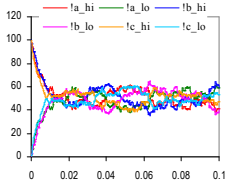


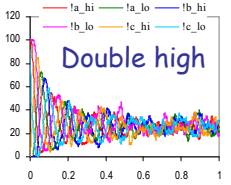


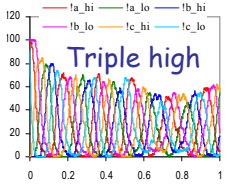


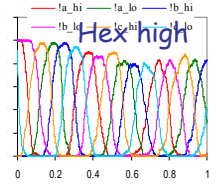






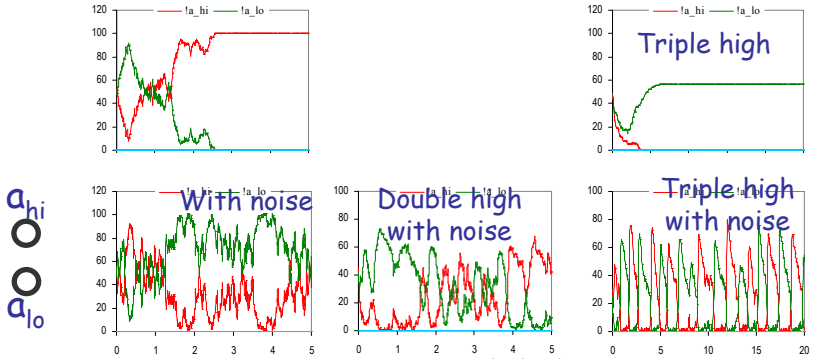


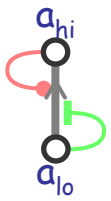


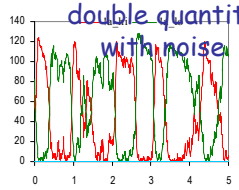


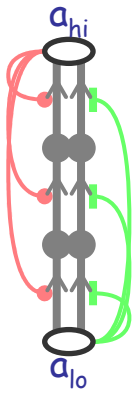

```
directive sample 5.0 1000
directive plot !a_hi; !a_lo; !b_hi; !b_lo; !c_hi; !c_lo

run replicate delay@1000.0 (* prevent simulation stopping *)

new a_hi@1.0:chan new a_lo@1.0:chan new a_up@1.0:chan new a_dn@1.0:chan
new b_hi@1.0:chan new b_lo@1.0:chan new b_up@1.0:chan new b_dn@1.0:chan
new c_hi@1.0:chan new c_lo@1.0:chan new c_up@1.0:chan new c_dn@1.0:chan

(* Single high
let Dip_hi(hi:chan, lo:chan, up:chan, dn:chan) =
 do !hi; Dip_hi(hi,lo,up,dn) or ?dn; Dip_lo(hi,lo,up,dn)
and Dip_lo(hi:chan, lo:chan, up:chan, dn:chan) =
 do !lo; Dip_lo(hi,lo,up,dn) or ?up; Dip_hi(hi,lo,up,dn)
*)
(* Double high*)
let Dip_hi(hi:chan, lo:chan, up:chan, dn:chan) =
 do !hi; Dip_hi(hi,lo,up,dn) or ?dn; ?dn; Dip_lo(hi,lo,up,dn)
and Dip_lo(hi:chan, lo:chan, up:chan, dn:chan) =
 do !lo; Dip_lo(hi,lo,up,dn) or ?up; ?up; Dip_hi(hi,lo,up,dn)

(* Triple high
let Dip_hi(hi:chan, lo:chan, up:chan, dn:chan) =
 do !hi; Dip_hi(hi,lo,up,dn) or ?dn; ?dn; Dip_lo(hi,lo,up,dn)
and Dip_lo(hi:chan, lo:chan, up:chan, dn:chan) =
 do !lo; Dip_lo(hi,lo,up,dn) or ?up; ?up; Dip_hi(hi,lo,up,dn)
*)
(* Hex high
let Dip_hi(hi:chan, lo:chan, up:chan, dn:chan) =
 do !hi; Dip_hi(hi,lo,up,dn) or ?dn; ?dn; ?dn; ?dn; ?dn; Dip_lo(hi,lo,up,dn)
and Dip_lo(hi:chan, lo:chan, up:chan, dn:chan) =
 do !lo; Dip_lo(hi,lo,up,dn) or ?up; ?up; ?up; ?up; ?up; Dip_hi(hi,lo,up,dn)
 *)

let A_lo(up:chan, dn:chan) = Dip_lo(a_hi,a_lo, up, dn)
let A_hi(up:chan, dn:chan) = Dip_hi(a_hi,a_lo, up, dn)
let B_lo(up:chan, dn:chan) = Dip_lo(b_hi,b_lo, up, dn)
let B_hi(up:chan, dn:chan) = Dip_hi(b_hi,b_lo, up, dn)
let C_lo(up:chan, dn:chan) = Dip_lo(c_hi,c_lo, up, dn)
let C_hi(up:chan, dn:chan) = Dip_hi(c_hi,c_lo, up, dn)

(* a := -a
run 100 of A_hi(a_lo,a_hi)
*)

(* a := a *)
run 100 of (A_hi(a_hi,a_lo) | A_lo(a_hi,a_lo))
run replicate !a_lo
run replicate !a_hi

(* a := b, b := -a
run 100 of (A_hi(b_hi,b_lo) | B_hi(a_lo,a_hi))
*)

(* a := c, b := a, c := -b
run 100 of (A_hi(c_hi,c_lo) | B_hi(a_hi,a_lo) | C_hi(b_lo,b_hi))
*)

(* a := -c, b := -a, c := -b
run 100 of (A_hi(c_lo,c_hi) | B_lo(a_lo,a_hi) | C_lo(b_lo,b_hi))
*)
```

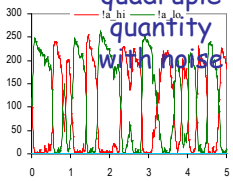
Positive Periodic! (with stochastic noise)

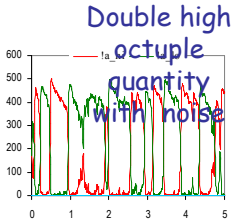




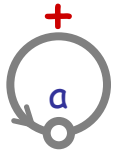








Double high is a flakey oscillator no matter what the quantity of a,b



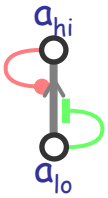
a := a

Necessary condition for
Multistability
NOT PERIODICITY

Positive

Although the Positive circuits are supposed to be multistable, and the Negative ones periodic, here we have positive circuits with stochastic noise that are periodic.

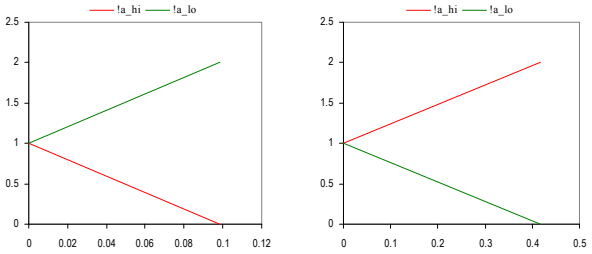
Pos: Population Behavior



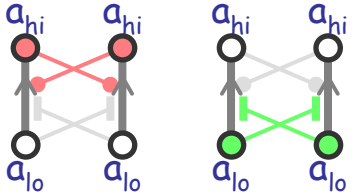
$x_i := x_j \quad (i \neq j)$ x_i range over {hi,lo}

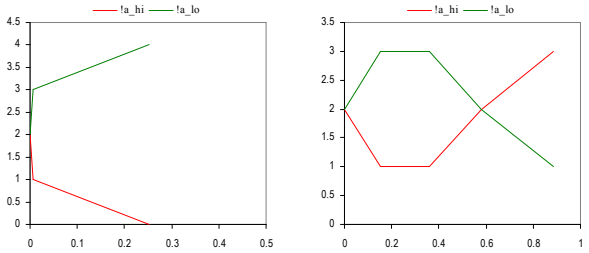
(positive = multistable)

1, inactive

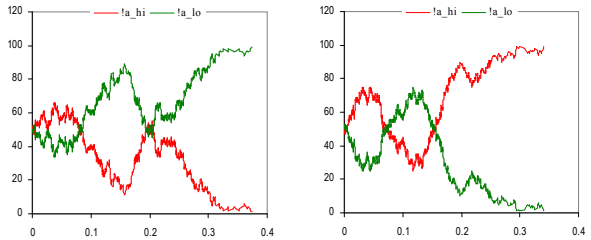


2, bistable





4, bistable



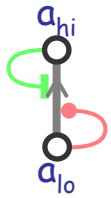
100, bistable

100+noise, fluctuate!

added noise:
$x_i := hi$
$x_i := lo$

```
directive sample 1.0
directive plot !a_hi; !a_lo

new a_hi@1.0:chan new a_lo@1.0:chan

let Dip_hi(hi:chan, lo:chan, up:chan, dn:chan) =
  do !hi; Dip_hi(hi,lo,up,dn) or ?dn; Dip_lo(hi,lo,up,dn)
and Dip_lo(hi:chan, lo:chan, up:chan, dn:chan) =
  do !lo; Dip_lo(hi,lo,up,dn) or ?up; Dip_hi(hi,lo,up,dn)

let A_hi(up:chan, dn:chan) = Dip_hi(a_hi,a_lo,up,dn)
let A_lo(up:chan, dn:chan) = Dip_lo(a_hi,a_lo,up,dn)

run 50 of (A_lo(a_hi,a_lo) | A_hi(a_hi,a_lo))
```

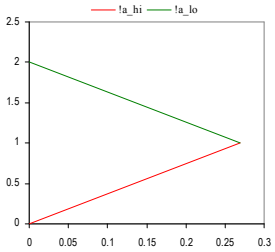
Neg: Population Behavior



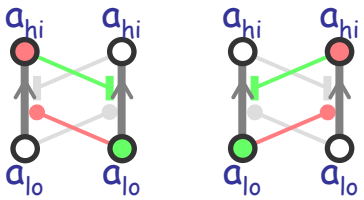
$x_i = -x_j \quad (i \neq j)$

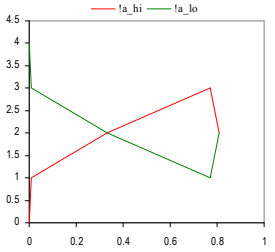
(negative = periodic)

1, inactive

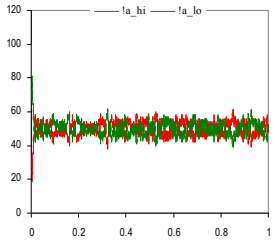


2, stable





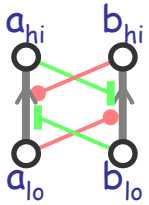
4, stable



100, stable

```
directive sample 1.0
directive plot !a_hi; !a_lo

new a_hi@1.0:chan new a_lo@1.0:chan

let Dip_hi(hi:chan, lo:chan, up:chan, dn:chan) =
  do !hi; Dip_hi(hi,lo,up,dn) or ?dn; Dip_lo(hi,lo,up,dn)
and Dip_lo(hi:chan, lo:chan, up:chan, dn:chan) =
  do !lo; Dip_lo(hi,lo,up,dn) or ?up; Dip_hi(hi,lo,up,dn)

let A_hi(up:chan, dn:chan) = Dip_hi(a_hi,a_lo,up,dn)
let A_lo(up:chan, dn:chan) = Dip_lo(a_hi,a_lo,up,dn)

run 100 of A_lo(a_lo,a_hi)
```

NegPos: Population Behavior

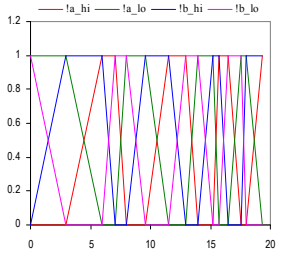


$a_i = b_j \quad (i \neq j)$

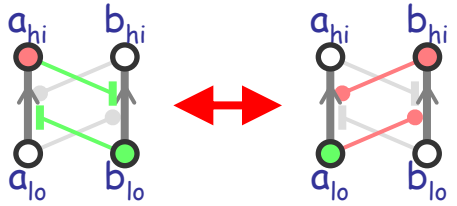
$b_i = -a_j \quad (i \neq j)$

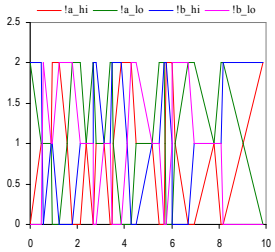
(negative = periodic)

1 inactive

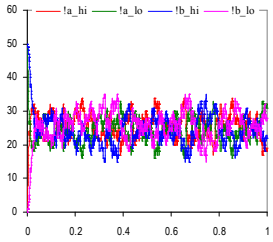


2 oscillate regularly





4 fluctuate



100 tend to stability

```
directive sample 20.0
directive plot la_hi; la_lo; lb_hi; lb_lo

new a_hi@1.0:chan new a_lo@1.0:chan
new b_hi@1.0:chan new b_lo@1.0:chan

let Dip_hi(hi:chan, lo:chan, up:chan, dn:chan) =
  do !hi; Dip_hi(hi,lo,up,dn) or ?dn; Dip_lo(hi,lo,up,dn)
and Dip_lo(hi:chan, lo:chan, up:chan, dn:chan) =
  do !lo; Dip_lo(hi,lo,up,dn) or ?up; Dip_hi(hi,lo,up,dn)

let A_hi(up:chan, dn:chan) = Dip_hi(a_hi,a_lo,up,dn)
let A_lo(up:chan, dn:chan) = Dip_lo(a_hi,a_lo,up,dn)
let B_hi(up:chan, dn:chan) = Dip_hi(b_hi,b_lo,up,dn)
let B_lo(up:chan, dn:chan) = Dip_lo(b_hi,b_lo,up,dn)

run 1 of (A_lo(b_hi,b_lo) | B_hi(a_lo,a_hi))
```

Summary

- Dipolin Circuits
  - A different logic than for Monopolin circuits
  - Apparently more robust, but still prone to stochastic glitches (Flip-Flop)

- Thomas Circuits
  - Formal influence diagrams, a very good theory of stability and periodicity
  - Can be represented by monopolin or dipolin circuits

Q?