Contents

Chapter 1

Examples

1.0.1 Latch Circuit

1.1 Verification of Latch Circuit

Latch is an electronic circuit (a bistable multivibrator) that has two stable states and thereby is capable of serving as one bit of memory. Today, the term *flip-flop* has come to mostly denote non-transparent (clocked or edge-triggered) devices, while the simpler transparent ones are often referred to as latches; however, as this distinction is quite new, the two words are sometimes used interchangeably (see history). There are various types of latch and flip flop circuits as shown in pages 402-414 of [**?**].

Figure 1.1 shows a *pass gate latch* which is a very robust transparent latch, which is static, all nodes swing rail to rail, the state noise is isolated from output noise. However, it requires that the input noise should be controlled. Figure 1.2 shows a *clocked CMOS latch (C^2MOS)*,which is ususally slightly faster.

Figure 1.3 is the *differential sense-amplifier latch* or *Strong Arm latch*. The

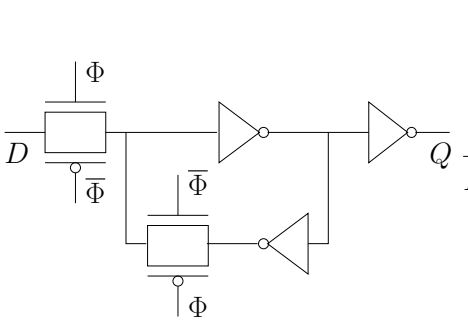


Figure 1.1: The pass gate latch circuit

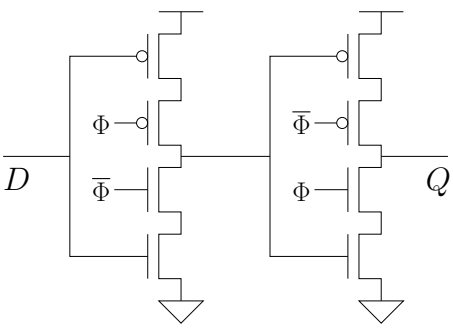


Figure 1.2: The C^2MOS latch circuit

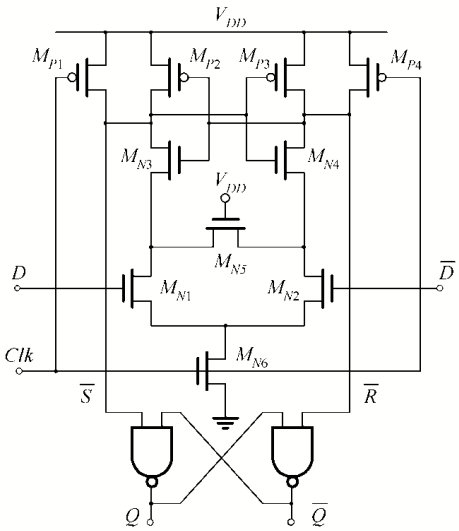


Figure 1.3: The differential latch circuit

internal weak transistor is used to fully staticize the outoput even if the inputs switch while the clock is high.

1.1.1 Specification

For a general latch, it is required that the input should be stable when the clock falls, otherwise, it may take long time for the internal node to settle. (input can only be in region 1 or 3 of brockett annulus when the clock is in region 4.) For the output, it should be stable when the clock is low. (output can only be in region 1 or 3 of brockett annulus while the clock is low.) Fiures 1.4 and 1.5 shows the specification of input and output.

As shown in figure 1.6, we can see

1. Both clock and data input specified by brockett annulu have four regions $low, rise, high, fall$, thus 16 states totally.
2. The signal changes in the order of $low \rightarrow rise \rightarrow high \rightarrow fall \rightarrow low$.
3. A trajectory can enter a region from its top or left face, leave a region through the bottom or right face.
4. The red regions ($< \Phi_{fall}, D_{rise} >, < \Phi_{fall}, D_{fall} >$)indicate that data input must be stable when clock is falling. Thus, the trajactories to/from these two states voilate the input specification.
5. Any trajectories must stay in low or $high$ region for a minimum amount of time T. Therefore, we have two kinds of transitions for each region.

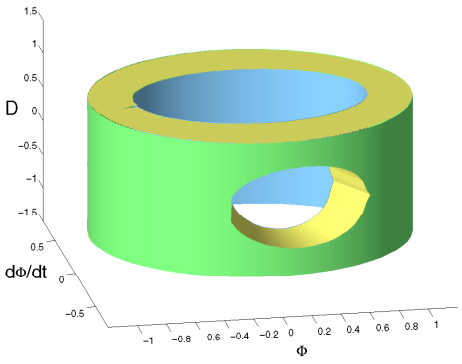


Figure 1.4: The input specification

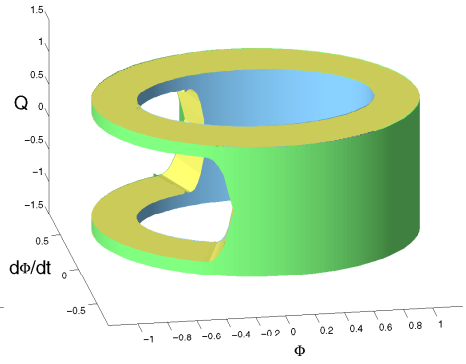


Figure 1.5: The output specification

Take $< \Phi_{rise}, D_{low} >$ for example, the dotted arrow is for transition that Φ has been in *low* region for long enough time and enter into *rise* region but the D input has not stay in *low* region for T time, thus D signal can not enter into *rise* region. (can not enter $< \Phi_{rise}, D_{rise} >$). The solid arrow is for transition that both Φ and D have been in stable region for long time thus can enter into *rise* region.

However, it is possible when the trace leaves the $< \Phi_{low}, D_{low} >$ region, D has not only stayed in *low* region for $T - \delta$ time. After a while, D satisfies the minimum stable time requirement and can enter into *rise* region. Therefore, it is possible that the dotted transition can go downward. This make the transition extremely complicated. How to record the time spent in stable region?

We use a simple solution: reduce T to $T - t_{trans}$ where t_{trans} is the maximum transition time from *low* to *high* region over all possible traces. Then, for the dotted transition, D will never statisfies the minium stable time T until it arrives Φ_{high}, D_{low} state. For the solid transition, it leaves the stable region earlier than requirement, thus an over approximated result is produced.

Therefore, there is no dotted arrow in the states where both Φ and D are changing ($< \Phi_{rise}, D_{rise} >$ and $< \Phi_{rise}, D_{fall} >$).

1.1.2 Simulation

The brockett annulus of d, q, x and $\bar{\Phi}$ are shown in figures 1.7, 1.8, 1.9 and 1.10.

The value of x signal in different states (brockett regions) is shown in figure 1.11.

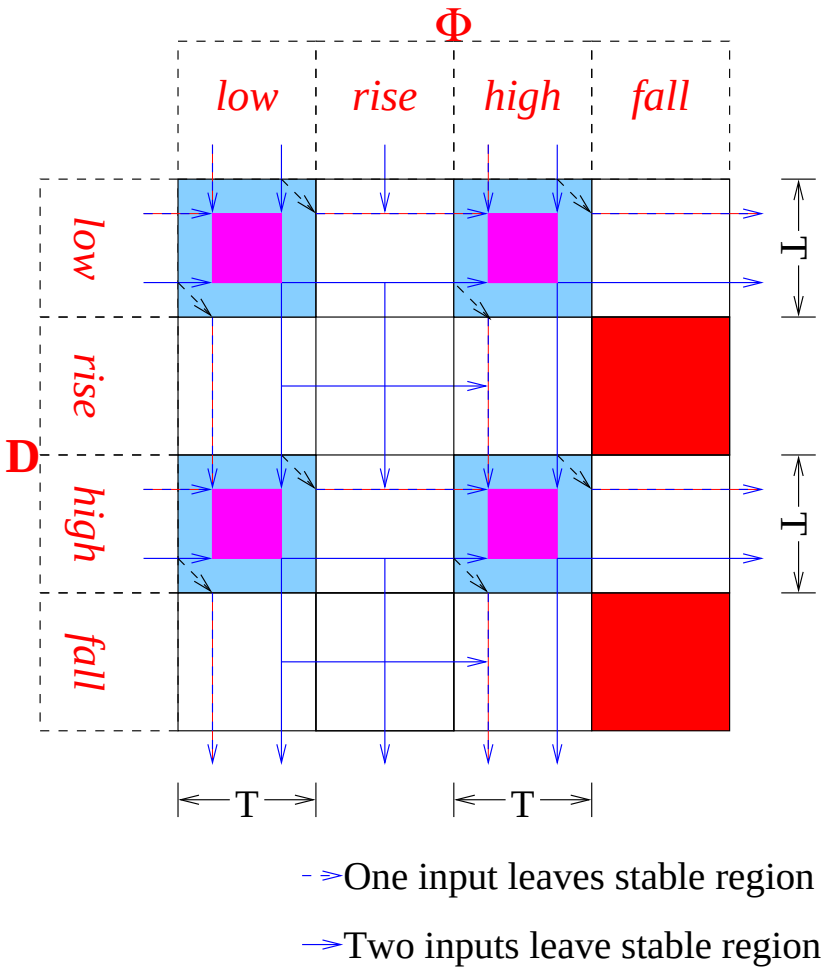


Figure 1.6: The state transition diagram

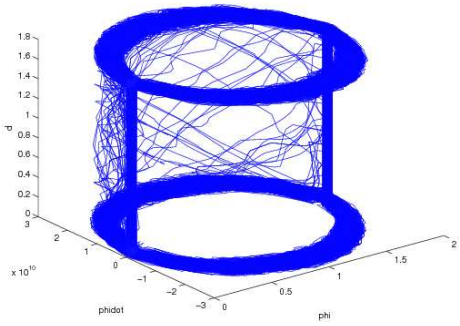


Figure 1.7: The input d specification

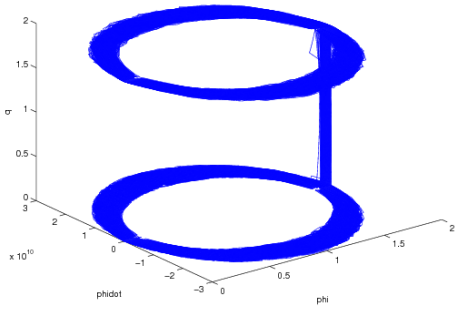


Figure 1.8: The output q specification

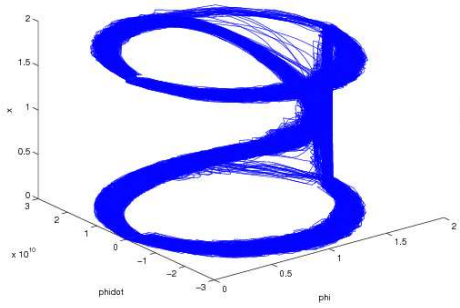


Figure 1.9: The x signal specification

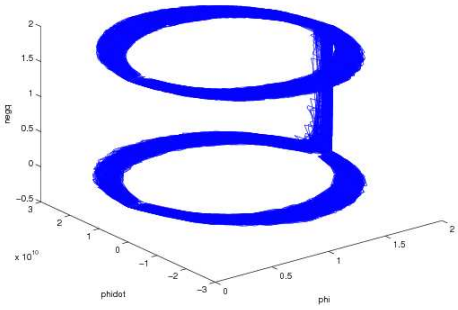


Figure 1.10: The $\bar{q}$ signal specification

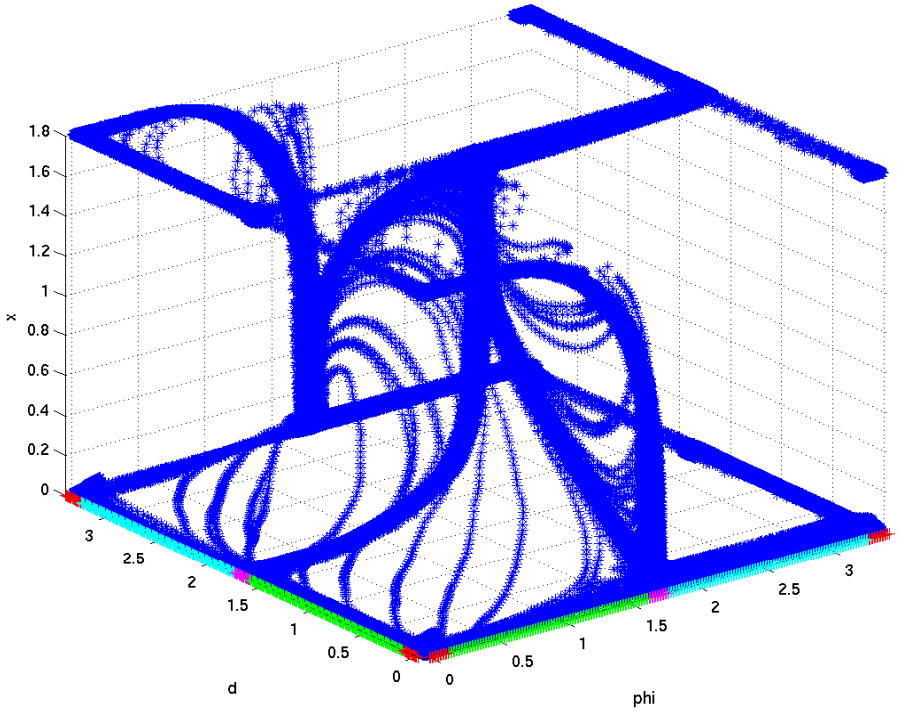


Figure 1.11: The x signal in different brockett region of simulation

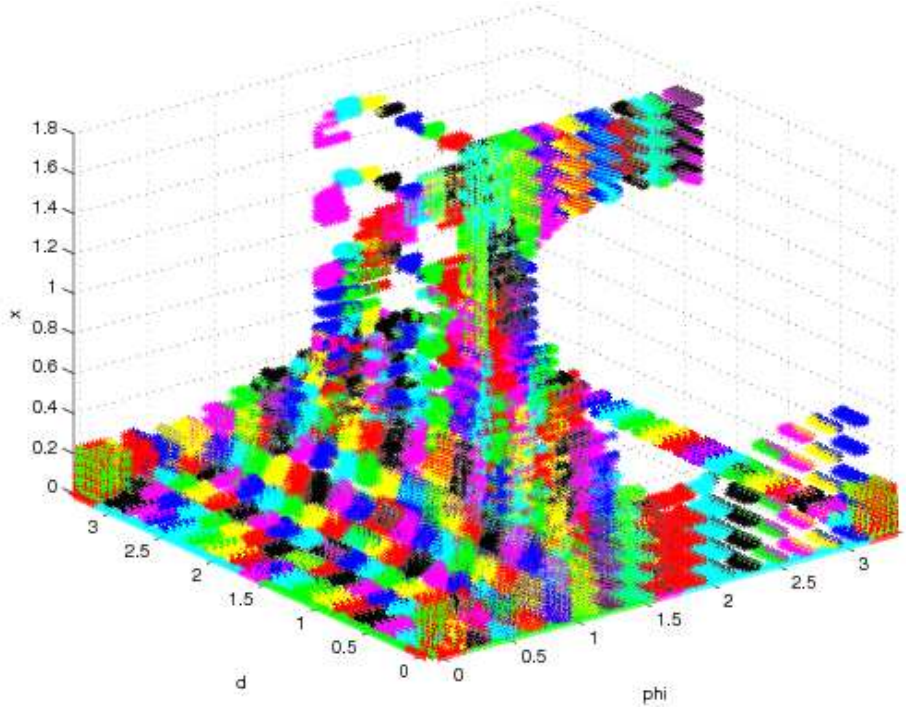


Figure 1.12: The x signal in different brockett region of coho

1.1.3 Verification

The first try

The initial stable is x is low, after one run, the result is shown in figure 1.12.

The first problem is that when d is low, why x is always increasing as Φ from region 1 to region 4, as shown in figure 1.13[1]. Similarly, when d is high, x is also in a large range.

- The inverter for $\overline{\Phi}$ is too small?

  The $\Phi - \overline{\Phi}$ plot is shown in figure 1.14. It is correct because of the circuit delay. (simulate it later).

  The size of transister only affect the delay of $\overline{\Phi}$. However, when the input is low, x should be low no matter what the value of Φ and $\overline{\Phi}$ is.

- Sum of error term is large?

  There are four transistors connects to x signal. However, I do not think it causes the problem because the error is not large for signal with three transistors before.

[1] The blue and red for x and cyan and green lines for $\overline{\Phi}$

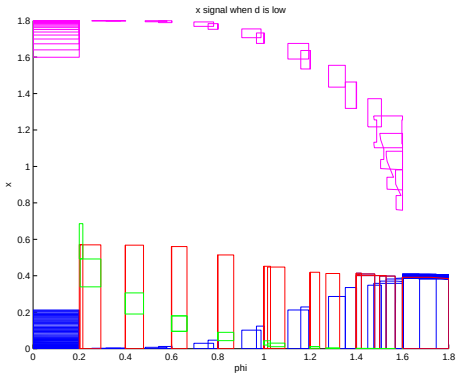


Figure 1.13: The $\Phi - x$ projection

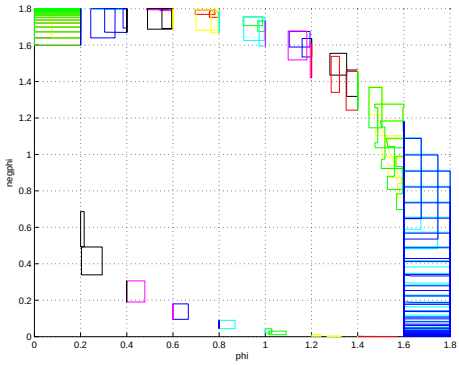


Figure 1.14: The $\Phi - \overline{\Phi}$ projection

Of course, we can use macro model later and compare the result.

- interp method's problem?

  I use the first projection in `ph_16_1_t3e1.mat` as an example to exam the problem. Its bounding bbox is

signal	**lower**	**upper**
Φ	0.3990	0.4000
d	0.0000	0.2000
$\overline{\Phi}$	0.1902	0.3061
x	0.0000	0.5682
$\bar{q}$	1.7824	1.8000
i	0.0000	0.0002

  I use `[tph,timeStep,bloatAmt] = ph_forward(ph,[],[],constraintLP)` to compare the result with differenent modeling method. The result is

Method	**maxBloat**	**timeStep**	**bloatAmt**$(\Phi, \overline{\Phi}, d)$	x
$[1, 2]$	0.1	6ps	$[-0.1, 0.1, \pm 0.08]$	[0,0.5682]
$[1, 2]$	0.2	12ps	$[-0.2, 0.2, \pm 0.16]$	[0,0.5682]
1	0.2	12ps	$[-0.2, 0.2, \pm 0.16]$	[0,0.5686]
2	0.2	12ps	$[-0.2, 0.2, \pm 0.16]$	[0,0.5682]
6	0.2	12ps	$[-0.2, 0.2, \pm 0.16]$	[0,0.5696]

  Therefore, we can see that the result does not depends on `maxBloat`, and the difference between methods is tiny. Even adding the max/min method can not remove the negative current as before because the transistor is not connected to vdd or gnd.

- Working on a face

  I pickup the $i-x$ slice, the 13^{th} face $[(0.0000, 0.5682), (0.0002, 0.5682)]$ and try to figure out why x does not drop. The bounding box of `face.modelLP` is

signal	lower	upper	bloat-	bloat+
Φ	0.3990	0.4000	0.1986	0.0000
d	0.0000	0.2000	0.1579	0.1579
$\overline{\Phi}$	0.1902	0.3061	0.0000	0.1573
x	0.5650	0.5689	0.0032	0.0007
$\bar{q}$	1.7824	1.8000	0.0144	0.0000
i	0.0000	0.0013	0.0003	0.0011

  Using this bounding box, the current through passgate should be

transistor	grid	interp	lsm
$NMos_{d-x}$	$[0.001, 0.485]e-5$	$[-0.310, 1.379]e-5$	$[-0.201, 0.626]e-5$
$PMos_{d-x}$	$[0.009, 0.196]e-5$	$[-0.166, 0.471]e-5$	$[-0.036, 0.195]e-5$
$NMos_{i-x}$	$[0.025, 0.577]e-6$	$[-0.218, 1.059]e-5$	$[-0.156, 0.717]e-6$
$PMos_{i-x}$	$[0.117, 0.117]e-7$	$[0.527, 0.533]e-6$	$[0.101, 0.121]e-7$

  We can see that all these four transistors are in (or nearly) cutoff region. The current from x to d or i can be very tiny leakage current, or relative huge current when the gate voltage is above threshold votlage. Thus the linear inclusion contains *negative* current from d, i to x even their voltage is lower. We can also see that the `interp` method produces a much larger error in the sub-threshold region.

- Try a face where transistors are conducted.

  I use `ph_9_1_t1e1.mat` where Φ is high. I use the 40^{th} projectagon where $\overline{\Phi}$ is clearly low. Then I use the $i-x$ slice, the 10^{th} face $[(0.17e-5, 0.3926), (0.60e-5, 0.3926)]$. The bounding box of `modelLP` is

signal	lower	upper	bloat-	bloat+
Φ	1.6000	1.8000	0.0738	0.0738
d	0.0000	0.2000	0.0738	0.0738
$\overline{\Phi}$	0.0000	0.0012	0.0005	0.0002
x	0.1980	0.5273	0.1946	0.1347
$\bar{q}$	1.7964	1.8000	0.0023	0.0023
i	0.0000	0.0000	0.0000	0.0000

  The current through passgate transistors should be

transistor	grid	interp	lsm
$NMos_{d-x}$	$[0.0000, 0.0047]$	$[-0.0004, 0.0050]$	$[-0.0005, 0.0050]$
$NMos_{d-x}$	$[0.0017, 0.0040]$	$[0.0009, 0.0041]$	$[0.0016, 0.0041]$
$PMos_{d-x}$	$[0.0000, 0.3103]e-4$	$[-0.1206, 0.2758]e-4$	$[-0.1124, 0.3487]e-4$
$NMos_{i-x}$	$[0.1108, 0.1423]e-9$	$[0.1353, 0.1720]e-7$	$[0.1081, 0.1881]e-9$
$PMos_{i-x}$	$[0.3042, 0.8842]e-12$	$[0.2843, 0.8817]e-12$	$[0.2858, 0.8836]e-12$

`maxDot` of x is $[-2.4839, -0.69]e10$ on the face.

We can see the error is from

1. `bloatAmt` is large, because each edge moves inward sharply, but our model has to bloat it outward the same ammount now.
2. quadratic interpolation error
3. linearization error
4. integration error

I guess the current of the second nmos using `interp` method does not contain the real value is because of large quadratic interpolation error in sub-threshold region.

The result of `lp_project` and `lp_project2` is so different. It seems *lp_project* is incorrect.

- LP project bugs

  The bug is caused by `lp_unique` which remove the `Ta2w` field of the `lp`.

- Try an face on $d-x$ face

  Because x is almost independent of i, it is better to work on a face of $d-x$ slice. I use the 7^{th} face $[(0.1413, 0.3926), (0.2000, 0.3926)]$ of the same projectagon as above. The bounding box of `modelLP` is

signal	lower	upper	bloat-	bloat+
Φ	1.6000	1.8000	0.0738	0.0738
d	0.0676	0.2738	0.0738	0.0738
$\overline{\Phi}$	0.0000	0.0012	0.0005	0.0002
x	0.1980	0.5273	0.1946	0.1347
$\bar{q}$	1.7964	1.8000	0.0023	0.0023
i	0.0000	0.0000	0.0000	0.0000

The current through passgate transistors should be

transistor	grid	interp	lsm
$NMos_{d-x}$	$[0.0017, 0.0027]$	$[0.0010, 0.0028]$	$[0.0010, 0.0028]$
$PMos_{d-x}$	$[0.0111, 0.0137]e-4$	$[-0.0454, 0.2039]e-4$	$[0.0030, 0.2529]e-4$
$NMos_{i-x}$	$[0.1292, 0.1292]e-9$	$[0.1522, 0.1603]e-7$	$[0.1264, 0.1493]e-9$
$PMos_{i-x}$	$[0.6248, 0.6248]e-12$	$[0.6277, 0.6439]e-12$	$[0.6278, 0.6455]e-12$

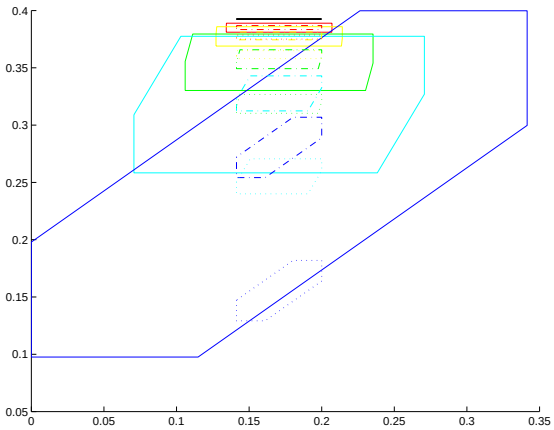


Figure 1.15: The integration of linear inclusion

`maxDot` of x is $[-3.6214, -2.9187]e10$ on the face and $[-4.8458, -1.3819]e10$ on the `modelLP`.

- Integration error

  It is weired that $\dot{x}$ is always negative, however, the projected polygon might increase! As shown in figure 1.15, we compute the forward face with different time step. We can see, at the beginning, x is decreasing, however, it turns back when time step is larger.

  To analyze the reson, we remove the error term (shown as dotted polygon) and constant term (shown as the dashed polygon) from the inclusion. For the linear system, x is always decreasing. The constant term only shifts the polygon.

  However, the error term bloat the polygon outward almost linearly as time advances. Thus, when the $\dot{x} = Ax$ decrease x slower than linear speed, the error term catch up and x value jumps back. When x is around 0.2 or above, $\dot{x}$ might be positive, thus $\dot{x} = Ax + b$ make the value of x stop to decrease. However, the error term still bloat the polygon outward. Just imagine the time step is huge, thus the error term can destroy any progress made by the linear system.

  This is cause by the integration approximation error because the input can not move all faces outmost at the same time. Thus, the time step can not too large, otherwise the integration error is huge[2].

- Linearization error

  As shown in figure 1.16, the linearization errors usually have opposite sign in the middle and at the endpoints.

[2]See Integrator chaper for details

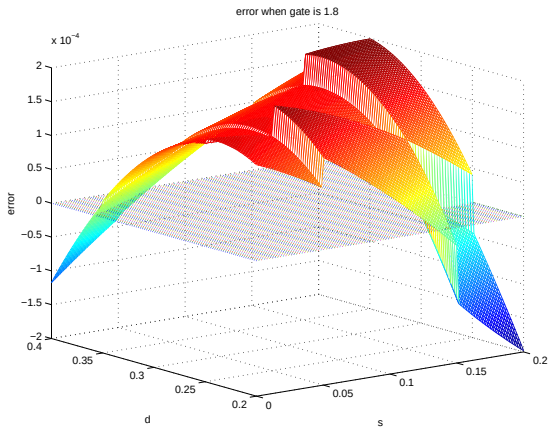


Figure 1.16: The linearization error

- Try small bloat amount

  With a smaller bloat amount (reduce 0.2 to 0.1), the problem is not solved. The result is similar as shown in figure 1.17. The value of x in 16_1 state is reduced from $[0, 0.57]$ to $[0, 0.38]$. The value in 16_9 state is reduced from $[1.40, 1.8]$ to $[1.5, 1.8]$.

The second try

I try to run coho again with smaller bloat amount. The result is better and the specification of x signal is shown in figure 1.18.

However, there is a problem that an invariant space can not be found in the $\langle high, low \rangle$ state[3]. As shown in figure 1.19, the $d - x$ polygon are in a stable region, However, the simiplification error or other approximation makes the polygon always has some point which is clearly outside the previous polygon. COHO just wanders there and can not exits.

Increasing tolerence does not help. The tolerence is $1e-3$, which can not be increase any futhure, otherwise COHO will exit before an invariant set is found.

There are several solutions

- Compute the union of all projectagon in history

  We compute the union of all previous projectagon and check if it contains the current projectagon. The problem is it may be expensive, and the union error might cause COHO exits before an invariant set is found.

- Bloat polygon vertices outside

[3]When time constraint is satisfied for the stable region, the computation can be don once an invariant set is found

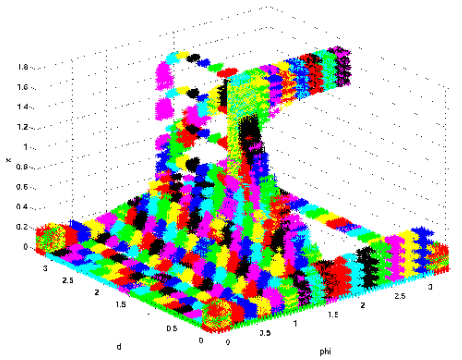


Figure 1.17: The x signal with smaller bloat amount

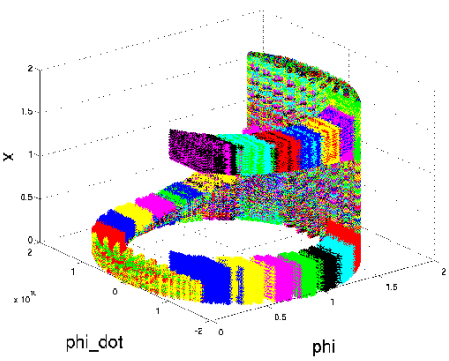


Figure 1.18: The $\Phi - \overline{\Phi} - x$ specification with smaller bloat amount

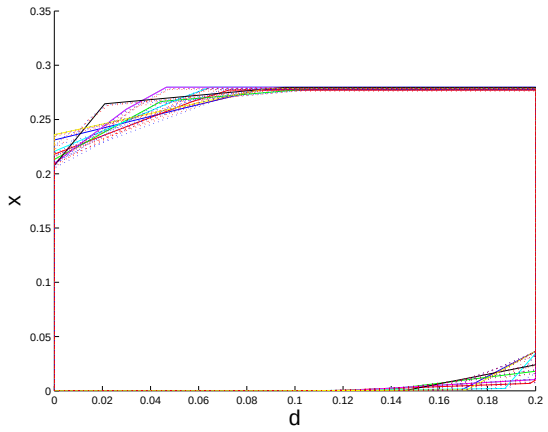


Figure 1.19: The $d - x$ polygon does not converge because of approximation

We use `inside` function to test if a point *pt* is in a polgyon *poly*. The function always return 0 if *pt* is in *poly* and return 1 if *pt* is outside *poly* and the distance is greater than *tol*, otherwise, 2 is returned. Therefore, it is the same with moving vertices of previous polygon outside. From figure 1.19, we can see that polgyon still has points outside the bloated polygon.

- User provided invariant set for each state

  User provides an invariant bounding box and once COHO reaches this bbox and exits. The idea is easy, but user has to computes such a bounding box, COHO can not do it automatically.

- Use maxsteps or ask user

  Use the same method for stable region, when the step exceeds the maximum allowed steps, exits or ask users.

The third try

I found the bug that the saved face is incorrect sometimes, therefore we rerun the verification. The result is shown in figure 1.20, 1.21.

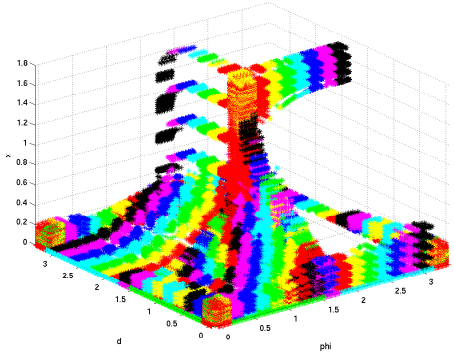


Figure 1.20: The result for low2high after fixing the face bug

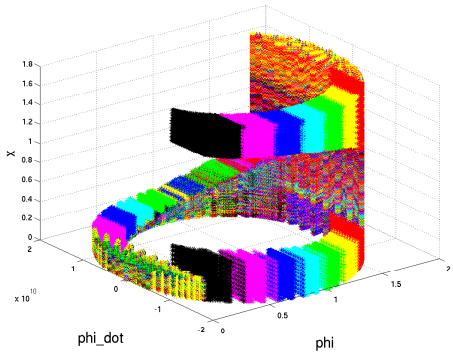


Figure 1.21: The x specification for low2high after fixing the face bug

The reachable region is a litter larger. For example, the x signal in different region is shown as

state	v2	v3
9_1_t1e1	[0.0000,0.2900]	[0.0000,0.2899]
9_9_t1e2	[1.5147,1.8000]	[1.5153,1.8000]
16_1_t3e1	[0.0000,0.3821]	[0.0000,0.3973]
16_9_t2e1	[1.5160,1.8000]	empty[4]

[4]part3 ignore the computation of 9_9_t3e1

16_9_t2e2	[1.5160,1.8000]	[1.5029,1.8000]
1_16_t2e1	[0.0000,0.0022]	[0.0000,0.0023]
2_16_t0	[0.0002,0.0043]	[0.0002,0.0040]
3_16_t0	[0.0024,0.0183]	[0.0019,0.0149]
4_16_t0	[0.0149,0.0551]	[0.0115,0.0580]
5_16_t0	[0.0463,0.1292]	[0.0401,0.1364]
6_16_t0	[0.0772,0.2143]	[0.0671,0.2350]
7_16_t0	[0.1126,0.3193]	[0.1174,0.3124]
8_16_t0	[0.1542,0.3714]	[0.1637,0.3477]
9_16_t2e1	[0.2635,0.5315]	empty
9_16_t2e2	[0.1676,1.3451]	[0.1847,1.3706]

I also compared the simulation and verification result. The verification contains all simulation trajectories as shown in figure 1.22 and 1.23. However, this might not be alway true as shown in figure 1.24. Of course, we use different ids in simulation and verification. But it is still a little weird.

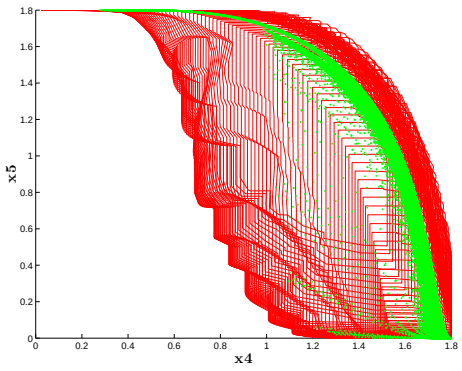


Figure 1.22: Simulation and Verification result in $\langle 9, 9\rangle$ state

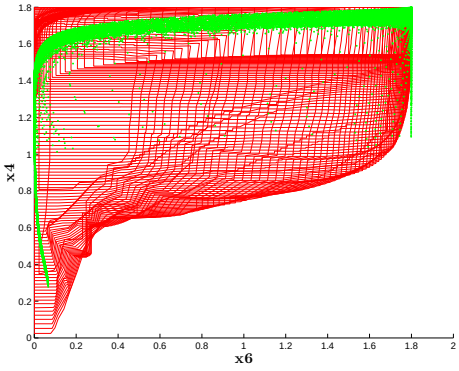


Figure 1.23: Simulation and Verification result in $\langle 9, 9\rangle$ state

Both low2high and high2low are completed. The input specification is shown in figure 1.25, the specification for other signals are shown in figures 1.26 1.27 and 1.28.

The result for x signal is similar with the simulation. However, the result for $\bar{q}$ and i are a little weird. Caused by $\overline{\Phi}$?

The forth try (v4)

The initial reachabe space we used is a little under approximated. And also some faces are ignored. Given the verification result, we can correct it and find the invariant set.

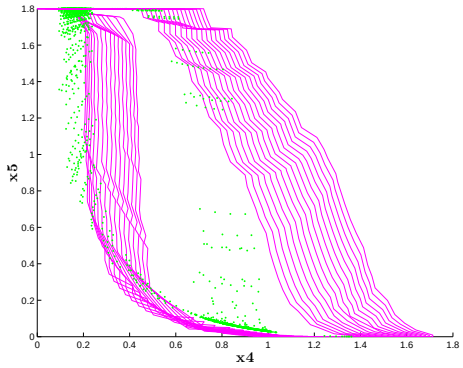


Figure 1.24: Simulation and Verification result in $\langle 9, 16/2 \rangle$ state

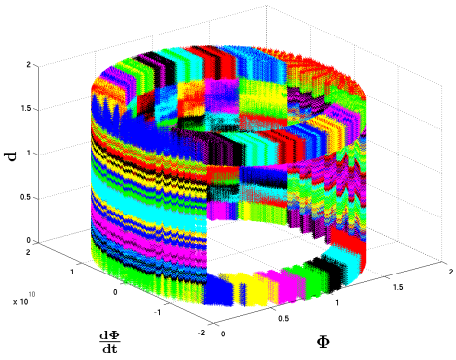


Figure 1.25: Input specification

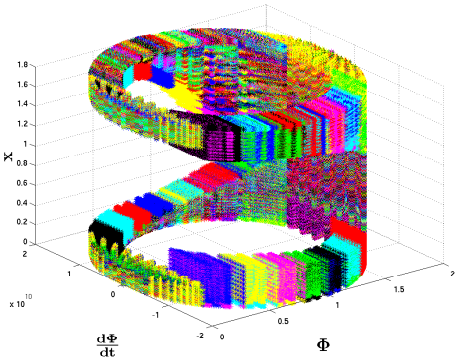


Figure 1.26: x signal specification

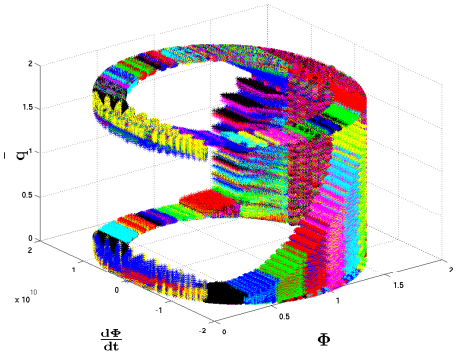


Figure 1.27: $\bar{q}$ signal specification

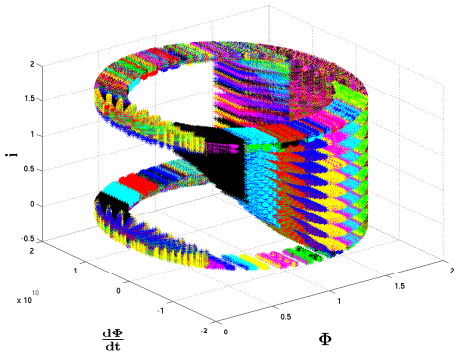


Figure 1.28: i signal specification

There are many faces to check and some of them have huge intervals. Thus, I would like to change the computation order and initial face as shown in figure 1.29.

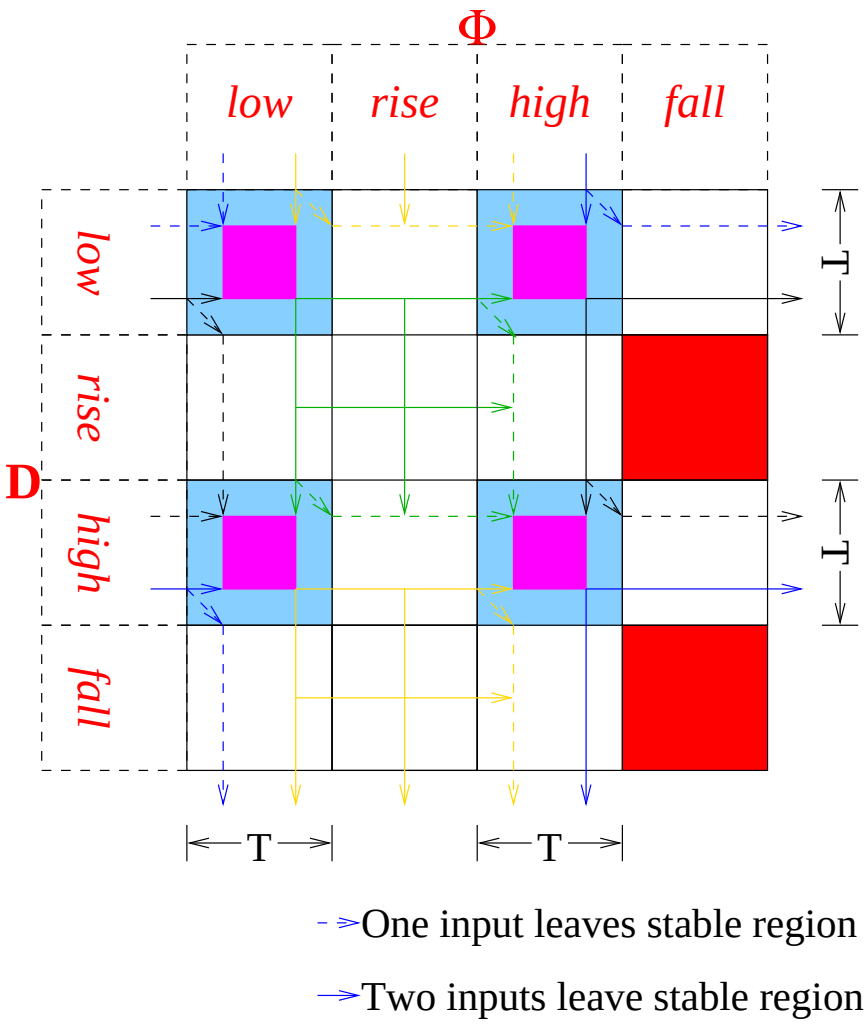


Figure 1.29: The new state transition diagram

Thus, the new computation order is

1. Given initial state, compute the reachable space in parrallel.

   (a) Start from $\langle low, low \rangle$ and compute reachable space for states shown in green color.

(b) Start from $\langle low, high\rangle$ and compute reachable space for states shown in yellow color.

(c) Start from $\langle high, low\rangle$ and compute reachable space for states shown in black color.

(d) Start from $\langle high, high\rangle$ and compute reachable space for states shown in blue color.

2. Verify the initial region and prove the space is invariant

(a) Using the result from item 1b, 1c, 1d to verify the initial region of item 1a.

(b) Using the result from item 1a, 1c, 1d to verify the initial region of item 1b.

(c) Using the result from item 1a, 1b, 1d to verify the initial region of item 1c.

(d) Using the result from item 1a, 1b, 1c to verify the initial region of item 1d.

However, after running phase 1a, 1b, 1c, 1d, I want to run 2a, 2b, 2c, 2d to check the initial guess. However, I found that the face to $\langle 1, 1\rangle$ might be high from $\langle 9, 9\rangle$. It is similar case for $\langle 1, 9\rangle$.

Thus, we can not run high2low and low2high seperately. We should run them first and verify them by using faces from both. Because the initial face for phase 1c and 1d are the same, we do not need to repeat it.

Finally, the compuation order is

1. Given initial state, compute the reachable space in parrallel.

(a) Start from $\langle low, low\rangle$ with $x = low$.

(b) Start from $\langle low, low\rangle$ with $x = high$.

(c) Start from $\langle low, high\rangle$ with $x = low$.

(d) Start from $\langle low, high\rangle$ with $x = high$.

(e) Start from $\langle high, low\rangle$ with $x = low$.

(f) Start from $\langle high, low\rangle$ with $x = high$. (same with above)

(g) Start from $\langle high, high\rangle$ with $x = low$.

(h) Start from $\langle high, high\rangle$ with $x = high$. (same with above)

2. Verify the initial region and prove the space is invariant

(a) Use result from 1e 1f, 1c, to verify the initial region of item 1a.

(b) Use result from 1g 1h, 1d, to verify the initial region of item 1b.

(c) Use result from 1e 1f, 1a, to verify the initial region of item 1c.

(d) Use result from 1g 1h, 1b, to verify the initial region of item 1d.

(e) Use result from 1a 1c, to verify the initial region of item 1e.

(f) Use result from 1b 1d, 1g, 1h, to verify the initial region of item 1f.

(g) Use result from 1a 1c, 1e, 1f to verify the initial region of item 1g.

(h) Use result from 1b 1d to verify the initial region of item 1h.

I also change the size of inverter for $\overline{\Phi}$ signal.
The result is shown in figure 1.30, 1.31, 1.32, 1.33.

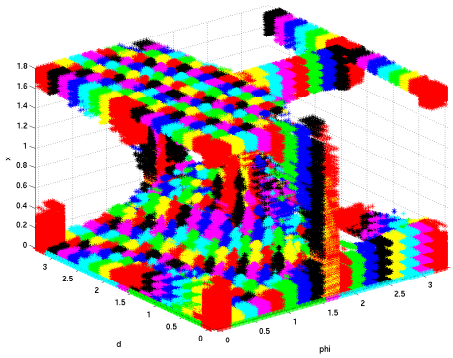


Figure 1.30: xbrock of v4

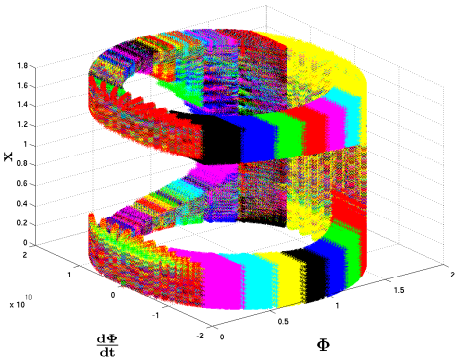


Figure 1.31: The xspec of v4

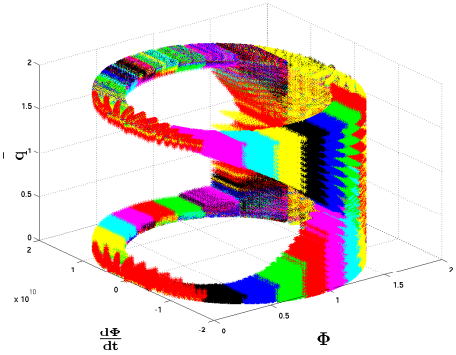


Figure 1.32: The $\bar{q}spec$ of v4

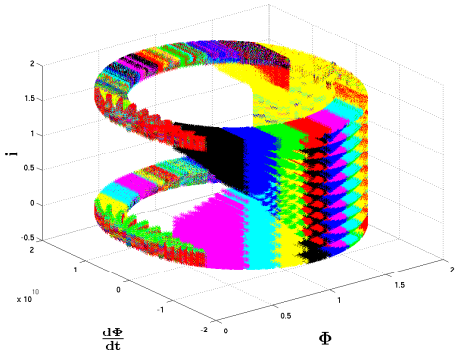


Figure 1.33: The ispec of v4

The initial and verified face is

face	low2high		high2low	
	inital	result	initial	result
clk(1,1)	0.2 0.2 0.0 0.2 1.7 1.8 0.0 0.1 1.7 1.8 0.0 0.1	0.2 0.2 0.0 0.2 1.7990 1.8 0.0 0.0016 1.8 1.8 0.0 0.0008	0.2 0.2 0.0 0.2 1.7 1.8 1.7 1.8 0.0 0.1 1.7 1.8	0.2 0.2 0.0 0.2 1.7987 1.8 1.7919 1.8 0.0 0.0 1.7958 1.8

data(1,1)	0.0 0.2 0.2 0.2 1.7 1.8 0.0 0.1 1.7 1.8 0.0 0.1	0.0 0.2 0.2 0.2 1.7990 1.8 0.0 0.0016 1.8 1.8 0.0 0.0008	0.0 0.2 0.2 0.2 1.7 1.8 1.7 1.8 0.0 0.1 1.7 1.8	0.0 0.2 0.2 0.2 1.7987 1.8 1.7919 1.8 0.0 0.0 1.7958 1.8
clk(1,9)	0.2 0.2 1.6 1.8 1.7 1.8 0.0 0.1 1.7 1.8 0.0 0.1	0.2 0.2 1.6 1.8 1.7989 1.8 0.0 0.0025 1.8 1.8 0.0 0.0013	0.2 0.2 1.6 1.8 1.7 1.8 1.7 1.8 0.0 0.1 1.7 1.8	0.2 0.2 1.6 1.8 1.7991 1.8 1.7937 1.8 0.0 0.0 1.7966 1.8
data(1,9)	0.0 0.2 1.6 1.6 1.7 1.8 0.0 0.1 1.7 1.8 0.0 0.1	0.0 0.2 1.6 1.6 1.7989 1.8 0.0 0.0025 1.8 1.8 0.0 0.0013	0.0 0.2 1.6 1.6 1.7 1.8 1.7 1.8 0.0 0.1 1.7 1.8	0.0 0.2 1.6 1.6 1.7991 1.8 1.7937 1.8 0.0 0.0 1.7966 1.8
clk(9,1)	1.6 1.6 0.0 0.2 0.0 0.1 0.0 0.3 1.7 1.8 0.0 0.1	1.6 1.6 0.0 0.2 0.0 0.0007 0.0 0.2599 1.7986 1.8 0.0 0.0	same	same
data(9,1)	1.6 1.8 0.2 0.2 0.0 0.1 0.0 0.3 1.7 1.8 0.0 0.1	1.6 1.8 0.2 0.2 0.0 0.0007 0.0 0.2599 1.7986 1.8 0.0 0.0	same	same
clk(9,9)	1.6 1.6 1.6 1.8 0.0 0.1 1.5 1.8 0.0 0.1 1.7 1.8	1.6 1.6 1.6 1.8 0.0 0.0008 1.5364 1.8 0.0 0.0006 1.7998 1.8	same	same
data(9,9)	1.6 1.8 1.6 1.6 0.0 0.1 1.5 1.8 0.0 0.1 1.7 1.8	1.6 1.8 1.6 1.6 0.0 0.0008 1.5364 1.8 0.0 0.0006 1.7998 1.8	same	same

4.1:*ph_forward*

Another improvement is about the algorithm in *ph_foward* function. Figure 1.34 showns the time step and bloat amount of each steps. It seems the time step is quite stable, we can use the one from history directly. The maximum bloat amount changes more frequently. In the code, we try to increase the bloat amount to max bloat which make the it tries twice to find a suitable pair. I guess removing this step will reduce the number to 1.

Let us focus on the phase 1d. The number of steps is 363 and the number of tries is 681, about 2 tries per step. And figure 1.35 shows the timestep and maximum bloat amount for $\langle 9, 1\rangle$ state (199 steps, 295 tries).

From the profiler, I found that there are 588 steps with 1220 calls of `ph_model` with 427 fails of finding an valid pair and 121 tmes to reduce the error, and 5 times when bloat is too small. Which means we can not guess invalid pair at the first try.

I dumped the internal data for phase 2b, with 199 steps and 276 calls of `ph_model`. I found the bloat amount osccilate, which may caused by two reasons: the first one is the projection error (or other) may made the real bloat amount jump at perticular case, when bloat amouont is increase, the real bloat amount is decrease! The second (I think the most one) is that we use the previous real bloat amount as the first try of the next step. Thus it is common that in step i, we found b_1 is too small and increase to b_2 where the real bloat amount is

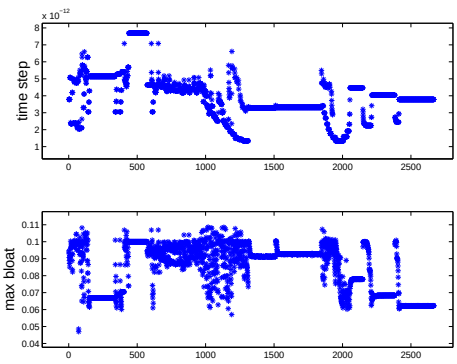


Figure 1.34: The time step and bloat amount used in COHO

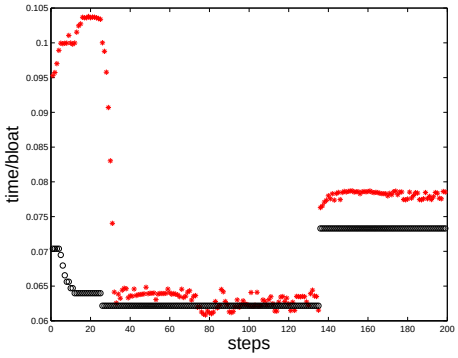


Figure 1.35: The time step and bloat amount used in $\langle 9, 1\rangle$ state

similar with b_1, thus we use b_1 and increase it to b_2 again. Thus I change to use the bloat amount rather than real bloat amount of previous steps as the initial guess for the next step.

The experimental result is

stage	textbftries/steps	ratio	**tries/steps**	ratio
(1,9)	276/199	1.3869	600/475	1.2632
(1,1)	598/421	1.4204	410/199 400/127	2.0603 3.1496
(1,9)	439/321	1.3676	224/199 405/199	1.1256 2.0352
(9,1)	1402/835	1.6790	412/199	2.0704
(9,9)	904/794	1.1385	431/199	2.1658

It seems the new algorithm is even worse. I think there are two changes: first, replace `bloatAmt` with `realBloatAmt`; second, use

```
timeStep = min(2,sqrt(factor)*counter^(-1/2))*timeStep
```

rather than

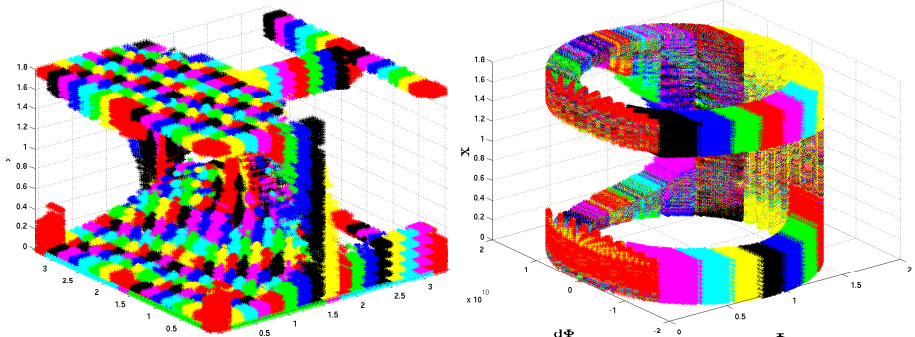
```
timeStep = min(2,factor*counter^(-1/2))*timeStep
```

to increase the time step. I run the (9,1) phase again, the result is 318/199 when restore the first one and 335/119 when restore both. Thus, I restore the first one only.

Macro model(v5)

I modified the `model_vdot` file directly. The result is similar. However, the qudratic polynomial error make the reachable space in stable state to be empty.

The result is shown in figure 1.36, 1.37, 1.38, 1.39.



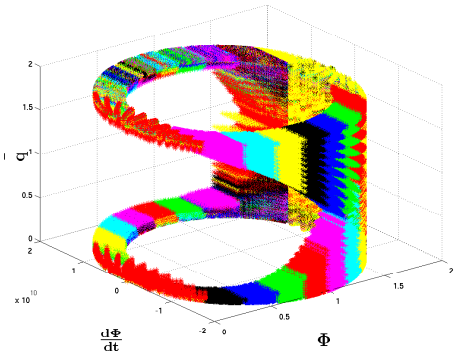


Figure 1.38: nqspec v5

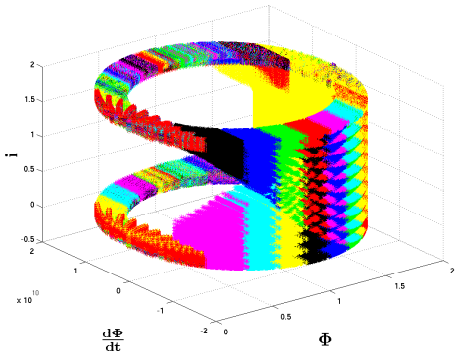


Figure 1.39: ispec v5

The result is similar, which means the error from sum of individual error is not significant.

However, I found sometime the union error is significant as shown in figure 1.40. The red line is the face from $\langle 9, 9\rangle$ and the green one is from $\langle 8, 10\rangle$, the union is shown as the magenta line.

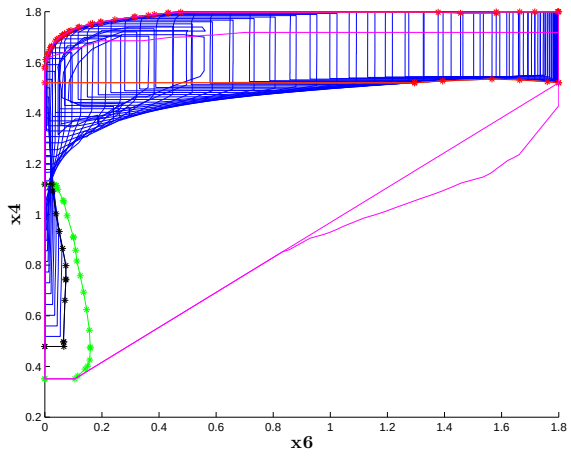


Figure 1.40: Union Larger Error

The projectagon of $\langle 9, 1-9\rangle$ in $21h$ and $\langle 9, 10-16\rangle$ in $22l$ is large, especially on $x - \bar{q}$ and $\bar{q} - i$ slice.

Using passgate macro model(v5.1)

I use the four terminal passgate model to improve speed and try to find if the error can be reduced more. I only run the $21h$ and $22l$ because they are the most difficult phase. The result is still similar without significant difference.

The running time is

	21l	**21h**	**22l**	**22h**	**23**	**24**
v4	$\frac{464}{1220} = 0.38$	$\frac{688}{1164} = 0.59$	$\frac{483}{996} = 0.48$	$\frac{413}{1307} = 0.31$	$\frac{333}{804} = 0.42$	$\frac{298}{681} = 0.44$
v5	$\frac{383}{914} = 0.42$	$\frac{534}{945} = 0.57$	$\frac{464}{959} = 0.48$	$\frac{649}{998} = 0.65$	$\frac{326}{838} = 0.39$	$\frac{263}{482} = 0.55$
v5.1		$\frac{540}{956} = 0.56$	$\frac{441}{979} = 0.45$			

It is weird that the running time is even slower. I'd better use profile to compare them later.

I rerun $22h$ with three methods at the same time. The result is:

Item	**v4**	**v5**	**v5.1**
Total Time	2.58e4	1.65e4	2.50e4
deviceIDS	2.37e4	1.53e4	2.39e4
Mn/Mp	$\frac{2.37e4}{174751\times10} = 0.0136$	$\frac{0.93}{174751\times4} = 0.0133$	0
Minv	0	$\frac{0.60e4}{174751\times3} = 0.0114$	$\frac{0.61e4}{174751\times3} = 0.0116$
Mpass	0	0	$\frac{1.78e4}{174751\times2} = 0.0509$

From the result, we can see

- both $v5$ and $v5.1$ save some time in `model_vdot` function by reducing the call number of `model_get` function
- The running time of `linfit` function depends on the model dimension and bounding box. The running time for 2D **Minv** is slightly smaller than **Mn**. But the running time of **Mpass** is almost three times larger.
- The passgate macro model make the computation slow. One reason is that the model is 4D which use more memeory and computation. Another more important reason is that a 4D bounding box is usually much larger than a 3D or 2D case, which mean the function has to handel more cubes. Thus, although $v5.1$ reduce the number of function calls, it increase the computation time of each call.
- It is not recommended to use large macro model if the error reduction is not significant.
- I should recorded the error term from three methods and compare them.

Measure the frequency

One important question is how fast can the clock be. From the input sepcification, we have the rising/falling time from 0.15 to 1.65 in the interval of

$$\begin{aligned} & \left[\frac{hh-ll}{2r_{max}}\left(\pi - 2\arccos\left(\frac{hl-lh}{hh-ll}\right)\right), \frac{hl-lh}{2r_{min}}\pi\right] \\ = & \left[\frac{1.8}{2\cdot 2e10}(\pi - 2\cdot\arccos(\frac{1.5}{1.8})), \frac{1.5}{2\cdot 1.5e10}\pi\right] \\ = & [0.115, 0.157]e{-}9\ (ns) \end{aligned}$$

The minimumal stable time is $1ns$ now. Thus the frenquency is in the range of

$$\left[\frac{1}{2\cdot(1+0.157)e\text{–}9}, \frac{1}{2\cdot(1+0.115)e\text{–}9}\right] = [432, 448]MHz$$

However, the minimum stable time only affects the computation of $26l, 26h, 27l, 27h, 28, 29$ which make the signal converge to stable values. I recheck the result and find the earliest time when the projectagon is contained by the initial guess. The result is (I use the v4 data):

state	step	time
26l(1,1)	75	0.17ns
26h(1,1)	21	0.16ns
27l(1,9)	20	0.10ns
27h(1,9)	71	0.27ns
28(9,1)	100	0.13ns
29(9,9)	90	0.14ns

$27h$ dominates the requirment of stable time. I think this is because the clock is low thus the internal node converges to stable value by the small inverter loop, and because the input face of this state is larger than others. Nowt the maximum frequcy is :

$$\left[\frac{1}{2\cdot(0.27+0.157)e\text{–}9}, \frac{1}{2\cdot(0.27+0.115)e\text{–}9}\right] = [1.17, 1.30]GHz$$

1.2 Verification of Flip-Flop

Given the result of latch circuit, we can start to verify the flip flop circuit.

From the verification of latch circuit, we know that *if the input d is stable when the clock* Φ *is falling, the output* $\bar{q}$ *is stable when* Φ *is clear low or rising.(It might not be clear low or high when* Φ *is falling because of circuit delay.)*

Thus, if we use $\bar{q}$ as the input of next latch with inversed clock $\overline{\Phi}$, we know that $\bar{q}$ is stable on the next rising edge of Φ (falling edge of $\overline{\Phi}$. Thus the output of next latch q is stable when the clock Φ is high ($\overline{\Phi}$ is low).

However, because $\bar{q}$ is stable before the next rising edge, the output p is stable much earlier as shown in figure 1.41.

As we measured, the maximum delay from when Φ starts to fall (1.6) to when $\bar{q}$ is stable is

$$\begin{aligned} t_{delay}^{d\uparrow} &= 84ps \\ t_{delay}^{d\downarrow} &= 123ps \end{aligned}$$

and the delay from when $\overline{\Phi}$ is high and $\bar{q}$ is stable to when q is stable is

$$\begin{aligned} t_{delay}^{\bar{q}\uparrow} &= 75ps \\ t_{delay}^{\bar{q}\downarrow} &= 115ps \end{aligned}$$

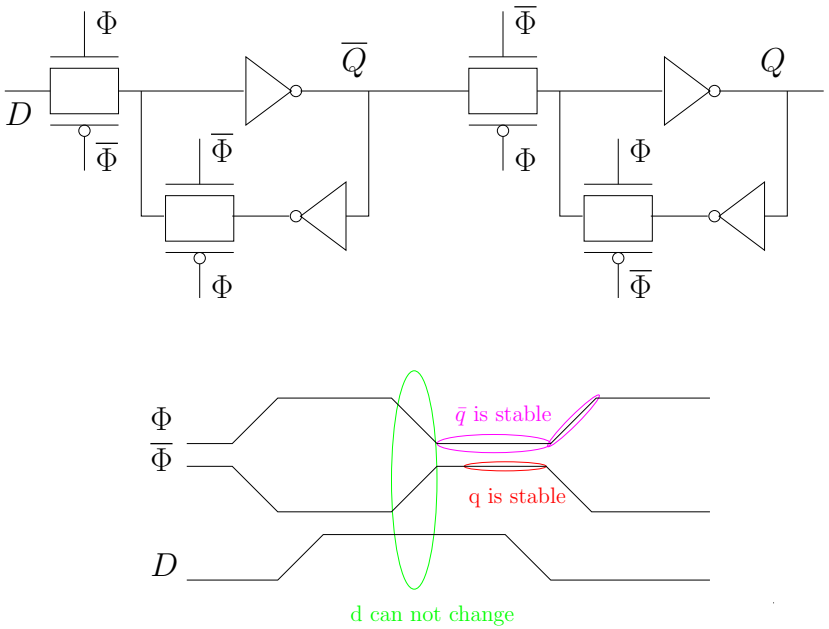


Figure 1.41: The flip flop

Both of them are smaller than the falling time of clock(input). Thus, the totally delay is bounded by

$$\begin{aligned} t_{delay} &= max(t^{d\uparrow}_{delay} + t^{\bar{q}\downarrow}_{delay}, t^{d\downarrow}_{delay} + t^{\bar{q}\uparrow}_{delay}) \\ &\approx 200ps \end{aligned} \tag{1.1}$$

Of course, this is an upper bound which can be decrease if verifing the flip-flop as a whole.