In Andrew Brown and Alex Yakovlev (Eds.) *We're Going to Need a Bigger Computer: Essays dedicated to Steve Furber, on the Occasion of his Retirement. At Last.* University of Manchester Print Unit, pages 66-81, 12 January 2024.

Micropipelines United

Marly Roncken, Ivan Sutherland, and Ebelechukwu Esimai

Asynchronous Research Center, Maseeh College of Engineering and Computer Science, Portland State University, 1900 SW Fourth Avenue, Suite 105, Portland, OR 97201, USA

Keywords: Micropipeline, transition signaling, jolly good fellow

1. Summary

We dedicate this Festschrift salute to our esteemed retiring friend and *Jolly Good* Fellow of the Royal Society, Steve Furber (*JG*FRS). This salute unites the two most prominent self-timed families for transition signaling, Micropipeline and Mousetrap. We bring in Click elements to bridge the gap between transition and level signaling. Using the latest developments in our unifying Link-Joint model we embrace both 2- and 4-phase level-based protocols, providing an alternative to the 4-phase Micropipeline latch control circuits from Manchester. Last but not least, we combine test solutions by both the University of Utah and the University of Manchester and bring these into the Link-Joint at-speed test approach.

2. Micropipelines

Following publication of Micropipelines [18], many people, including Steve's group at the University of Manchester [5, 6, 10, 11] and including the Tangram team at Philips Research in Eindhoven (Natlab) [2, 3], devised a plethora of self-timed protocols and circuit families for transition and level signaling, including Mousetrap [15] and Click [9]. Keeping track of all that diversity is next to impossible, but the Sparsø-Furber books [16, 17] provide a great service in doing just that.

In 2015, we introduced the Link-Joint model [12] to provide a protocol, family, and circuit independent way to think about self-timed systems. Test methods for self-timed systems drove conception of the Link-Joint model and proved to be remarkably easy to apply. Because the model separates state, in Links, from action, in Joints, testing is easy. Only Links hold state and only Joints do actions and so one can test by merely initializing or scanning Links after forbidding action by enough Joints to stabilize the values stored in Links. The Seitz arbiter in the 1979 Mead-Conway book can bring a self-timed system to a safe stop [14].

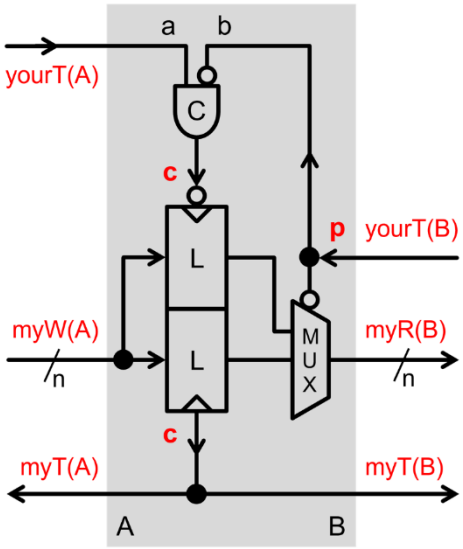
The five graphic-style chapters on the subsequent pages unite early research contributions in Micropipeline transition signaling by the University of Manchester, UK [6, 10, 11] and by the University of Utah in Salt Lake City, USA [8] with subsequent research contributions in Mousetrap transition signaling from Columbia University in New York City, USA, and the University of North Carolina in Chapel Hill, USA [1, 7, 15]. We mix these together with the final and long-lasting research contribution of Click by Philips Handshake Solutions in Eindhoven, The Netherlands [9]. We view this mix of asynchronous protocols and self-timed circuit families through our developing Link-Joint model [1, 4, 13]. Reflections and a change of view are often productive of new ideas, and so it is that these graphic-style chapters contain something old, something new, a lot borrowed, and a little blue — more on this after the graphics, in Section 3.

We hope this salute entertains as well as helps its viewers to appreciate the diversity of self-timed systems, the creativity of the people who devised them, and a few of their subtle relationships. We thank Steve Furber (*JG*FRS) for years of friendship and any help he may (or may not) anonymously have given us in years past.

*Authors for correspondence (mroncken@pdx.edu, ivans@cecs.pdx.edu, esimai@pdx.edu).

CHAPTER 1

Where we meet Micropipeline and Mousetrap and their different ways to implement exactly the same transition signaling protocol...



This is how we will model a simple Micropipeline stage as introduced by Ivan Sutherland in his CACM 1989 Turing Award lecture.

The visible differences with Mousetrap below are the C element and the capture-pass data latches.

The transition signaling protocol is under control of a Muller C element, and captures data myW(A) alternately in the top latch when *c* goes low and in the bottom latch when *c* goes high. The protocol passes data as myR(B), reading the top latch when $p=0$ and the bottom latch when $p=1$.

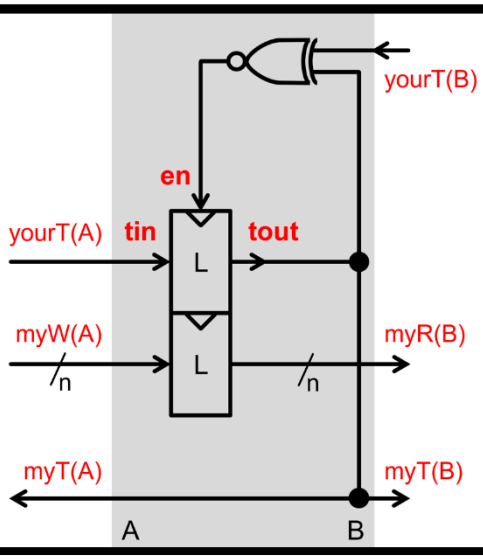
The stage is EMPTY and transparent for data when $c=p$, and FULL and opaque when $c\neq p$.

Here is a simple Mousetrap stage by Montek Singh and Steve Nowick worked out in detail in TVLSI 2007.

Visibly different from Micropipeline are the XNOR and *tin*-*tout* transition latch.

To see a change in yourT(A) or *tin*, the latch must be transparent, with $en=1$, which means that the stage is EMPTY.

The stage becomes FULL and opaque, with $en=0$, when *tout or* myT(B) changes.



Despite these differences, Micropipeline and Mousetrap are remarkably similar.

Both propagate data from A to B.

Both pair new data with a transition:
data myW(A) pair with yourT(A) — myR(B) with myT(B).

Both store the new state
before changing myT(A) or myT(B).

Both change myT(A) and myT(B)
to permit a predecessor to change data myW(A) and
to let a successor know that data myR(B) are available.

Both maintain their state
until yourT(B) changes.

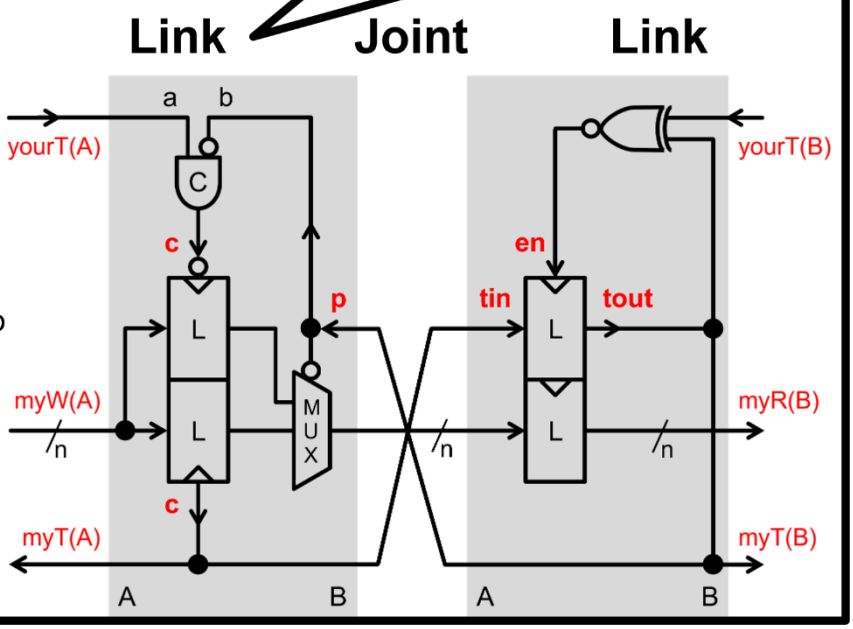
Both are EMPTY when myT(B)=yourT(B)
and FULL when myT(B)≠yourT(B).

Ahhh, you mean that both have the same interface!

This fits in the Link-Joint model that we introduced in ASYNC 2015, where Links store and transfer states between Joints for compute and flow control.

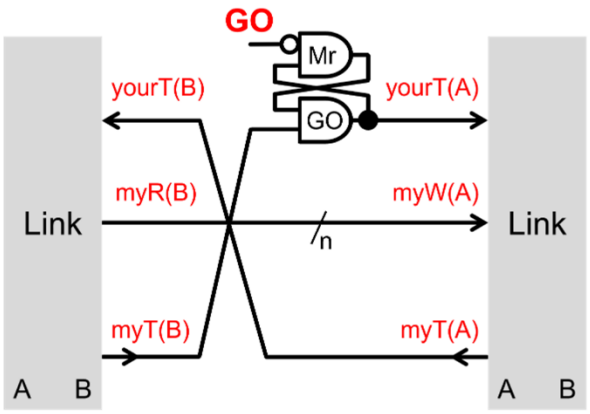
Circuit families such as Micropipeline or Mousetrap may differ but will use the same Link-Joint interface for the same protocol.

Here are a Micropipeline and Mousetrap Link with a COPY Joint in between.



CHAPTER 2

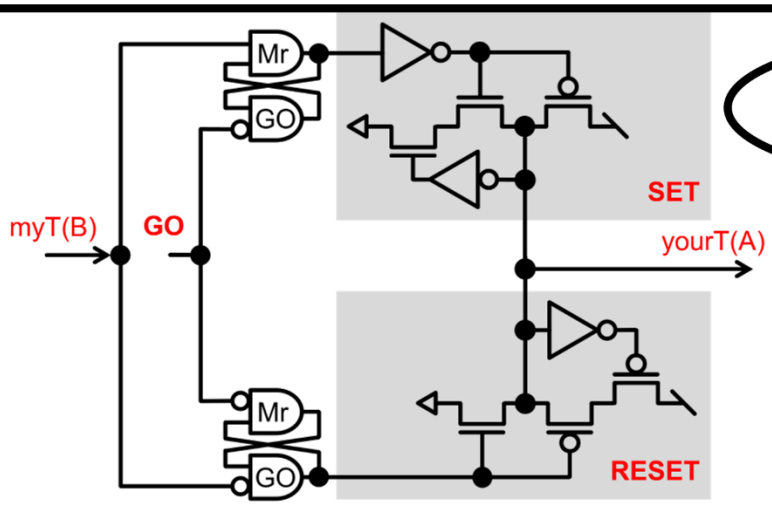
Where we meet MrGO in the COPY Joint and discover its use for initializing the design and for at-speed test...



Joints control the actions in a Link-Joint model. The ability to stop a self-timed action from outside makes it possible to initialize the circuit for design and test.

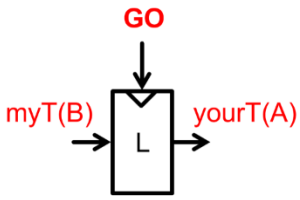
MrGO (pronounced “Mister GO”) provides stop-ability and comes in two versions: *(1) a proper stopper* that arbitrates between stop (¬GO) and a transition propagating from myT(B) to yourT(A), and *(2) a crowbar stopper* that merely stops.

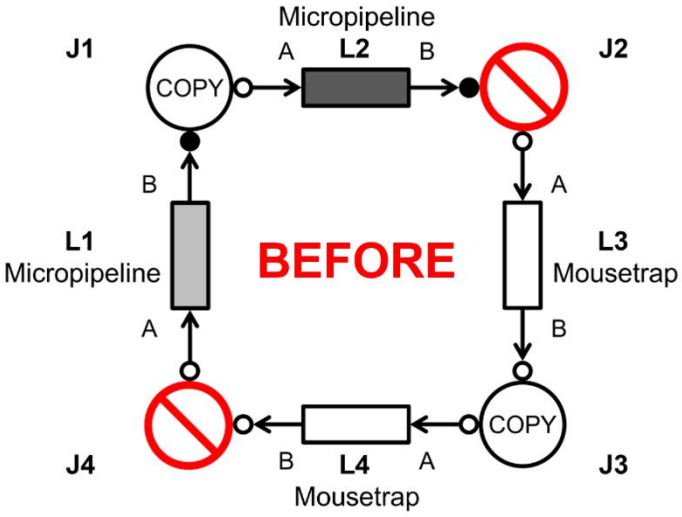
Each algorithmic loop must have a Joint with arbitrated MrGO to stop the loop.



The arbitrated proper stopper version of MrGO for transitions uses arbitrated level-sensitive MrGOs from our ASYNC 2015 paper, connected to a DriveHighKeepLow (set) latch for myT(B)=1 and a DriveLowKeepHigh (reset) latch otherwise.

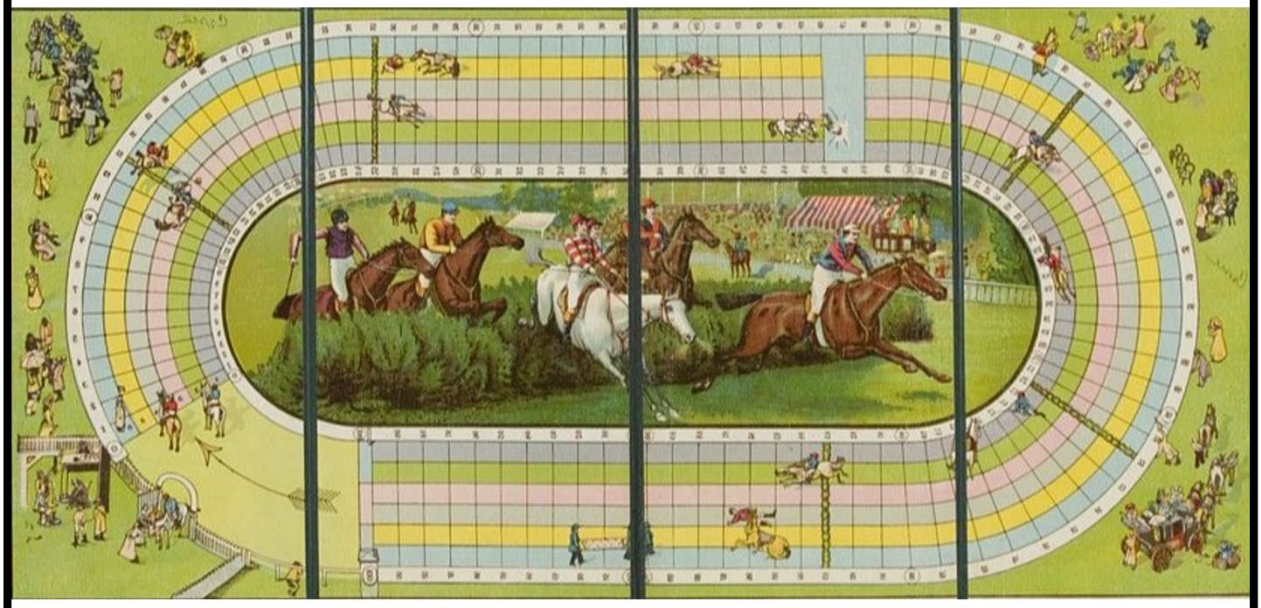
The crowbar version is a GO-enabled standard latch.

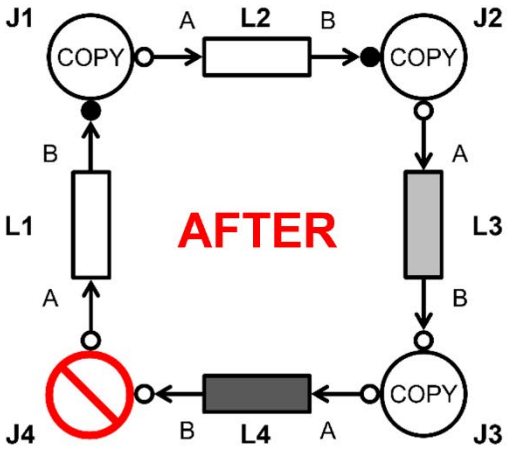




With MrGO, we can create a racetrack around a ring FIFO. Links L1 and L2 are FULL with light and dark grey data, and Links L3 and L4 are EMPTY.

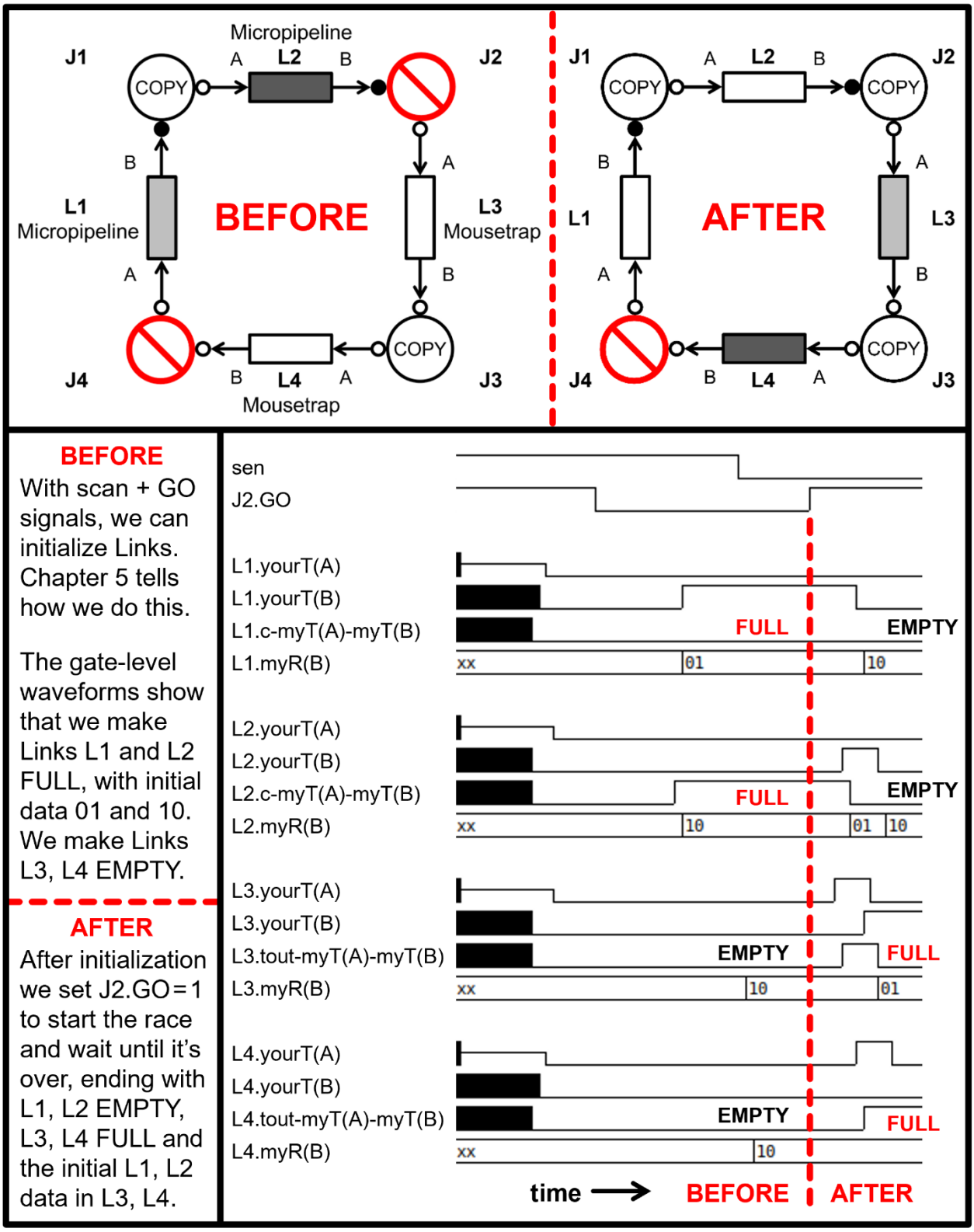
The STOP signs for COPY Joints J2 and J4 indicate that their GO=0. With J2.GO=0, J2 serves as the gatekeeper that prevents data in the take-off runway L1-J1-L2 from entering the landing runway L3-J3-L4.





As soon as we make J2.GO=1, data race at full speed from the take-off runway L1-J1-L2 through J2 to land in L3-J3-L4. Because our track is a FIFO, data remain in sequence.

With J4.GO=0, data cannot escape the landing runway.


Micropipeline
J1
J2
J3
J4
L1
Micropipeline
L2
L3
Mousetrap
L4
Mousetrap
COPY
A
B
BEFORE
AFTER
BEFORE
With scan + GO signals, we can initialize Links. Chapter 5 tells how we do this.
The gate-level waveforms show that we make Links L1 and L2 FULL, with initial data 01 and 10. We make Links L3, L4 EMPTY.
AFTER
After initialization we set J2.GO = 1 to start the race and wait until it's over, ending with L1, L2 EMPTY, L3, L4 FULL and the initial L1, L2 data in L3, L4.
sen
J2.GO
L1.yourT(A)
L1.yourT(B)
L1.c-myT(A)-myT(B)
L1.myR(B)
L2.yourT(A)
L2.yourT(B)
L2.c-myT(A)-myT(B)
L2.myR(B)
L3.yourT(A)
L3.yourT(B)
L3.tout-myT(A)-myT(B)
L3.myR(B)
L4.yourT(A)
L4.yourT(B)
L4.tout-myT(A)-myT(B)
L4.myR(B)
FULL
EMPTY
xx
01
10
time
BEFORE
AFTER

CHAPTER 3

Where we find an elephant in the room and get a bee to scare the gentle giant away...

Instead of doing half-rounds, let's race around the track and count how often data zip by L2 when there are 0, 1, 2, 3, or 4 data elements in the ring.

Without data, nothing zips by, so let's start with 1 data element in the ring FIFO — same setup as before, but only Link L2 is FULL.

Oops, I can't do that!

Let me try...

L1 is EMPTY, so L1.myT(B)=L1.yourT(B). Joint J1 connects L1.yourT(B) to L2.myT(A). And with myT(A)=myT(B) for each state, we should see L1.myT(B)=L2.myT(B).

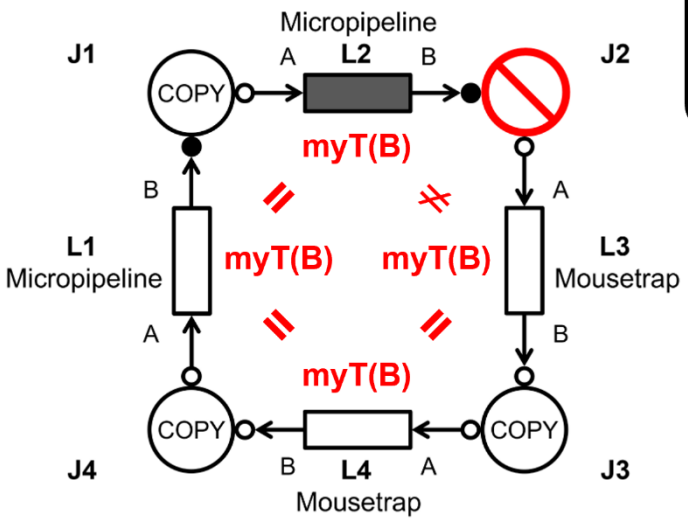
L2 is FULL, so L2.myT(B)≠L2.yourT(B). Joint J2 connects L2.yourT(B) to L3.myT(A). And with myT(A)=myT(B) for each state, we should see L2.myT(B)≠L3.myT(B).

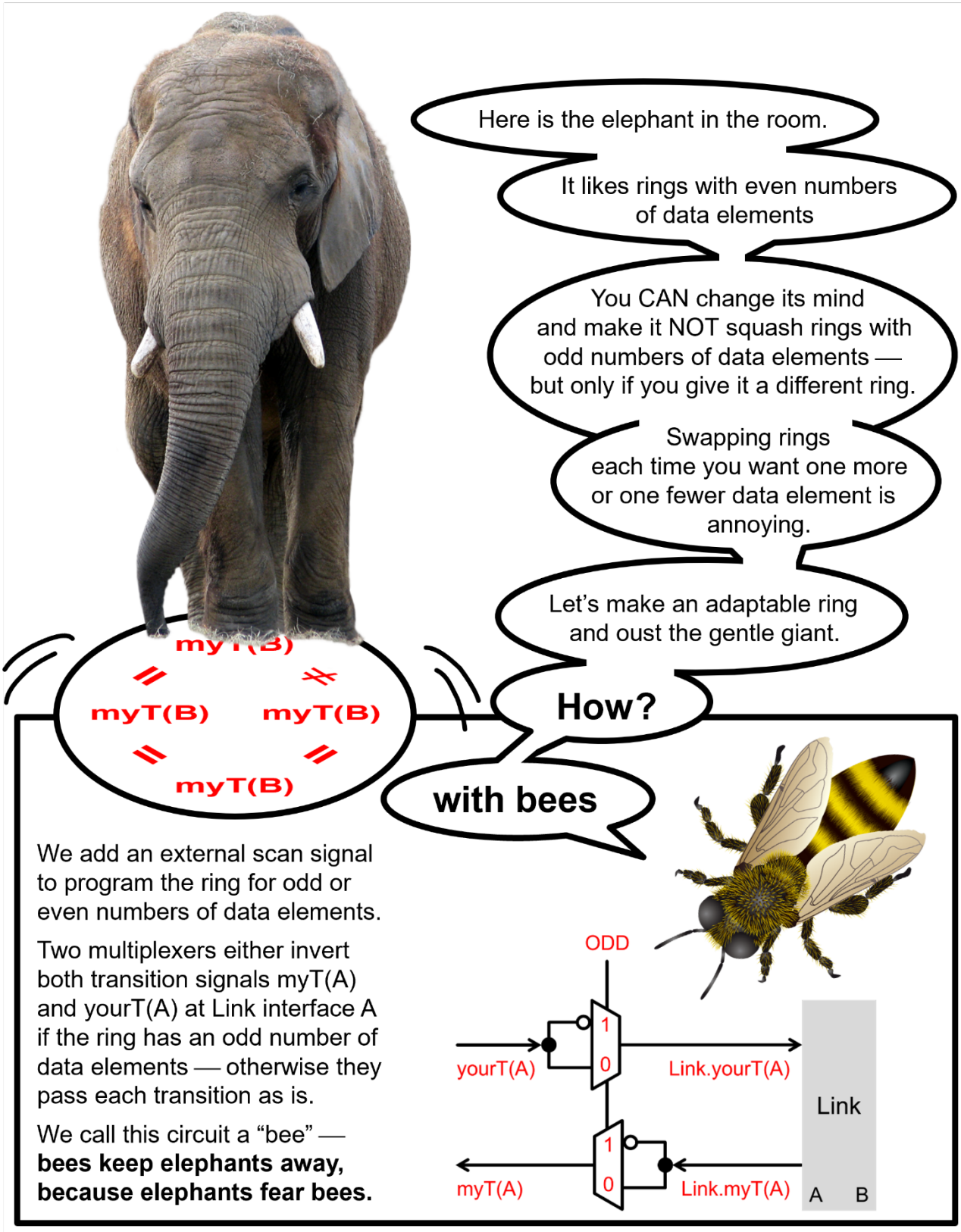
L3 and L4 are as EMPTY as L1 so L3.myT(B)=L4.myT(B) and L4.myT(B)=L1.myT(B).

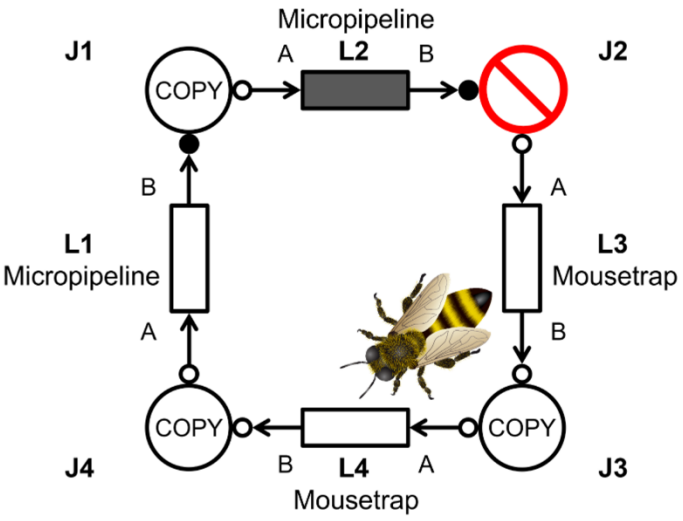
But that would mean L2.myT(B)≠L3.myT(B) and L3.myT(B)=L2.myT(B) and that's

Impossible!

And the fault is NOT in mixing Micropipeline and Mousetrap families, but in transition signaling itself.




Here is the elephant in the room.
It likes rings with even numbers of data elements
You CAN change its mind and make it NOT squash rings with odd numbers of data elements — but only if you give it a different ring.
Swapping rings each time you want one more or one fewer data element is annoying.
Let's make an adaptable ring and oust the gentle giant.
How?
with bees
myT(B)
myT(B)
myT(B)
myT(B)
We add an external scan signal to program the ring for odd or even numbers of data elements.
Two multiplexers either invert both transition signals myT(A) and yourT(A) at Link interface A if the ring has an odd number of data elements — otherwise they pass each transition as is.
We call this circuit a "bee" — **bees keep elephants away, because elephants fear bees.**
ODD
1
0
yourT(A)
Link.yourT(A)
Link
1
0
myT(A)
Link.myT(A)
A
B



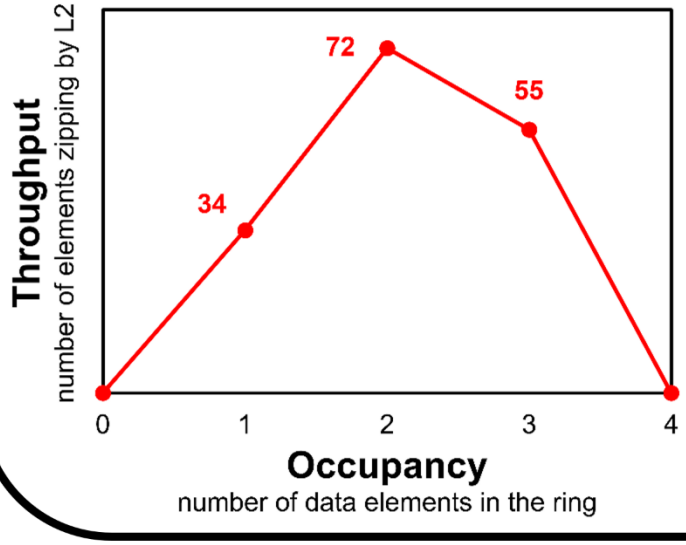
Time to race around the track and count how often data zip by L2 when there are 0, 1, 2, 3, or 4 data elements in the ring FIFO. Here is a ring FIFO with 1 element.

Our bee is at work at the interface between Joint J3 and Link L4. We set bee.ODD=1 before we put 1 or 3 elements in the ring FIFO, and bee.ODD=0 before we put 0, 2, or 4 elements in the ring.

The race starts as soon as we make J2.GO=1. Then data in L2 take off and circulate at full speed until we reset J2.GO=0, using the arbitrated version of MrGO.

CANOPY graph

Jennette Gill and Montek Singh named Canopy graphs and explored them in detail (ICCAD 2008). But the idea of the graphs is from Ted Williams and Mark Horowitz who characterized performance of a self-timed design by graphing it against occupancy (ARVLSI 1987).



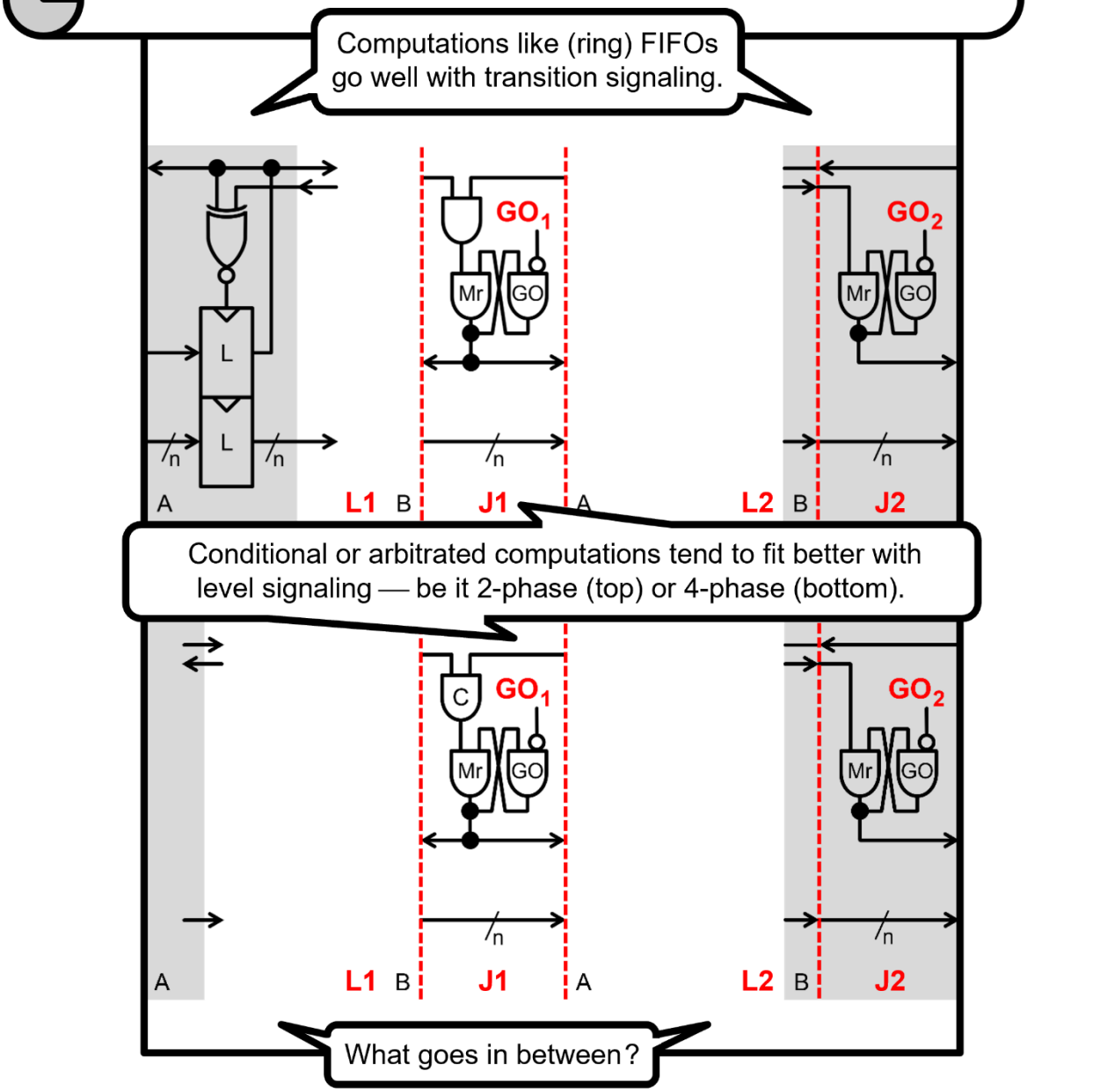
Data need space to move into. Without data (0 data elements) or space (4 data elements) nothing moves and nothing zips by L2.

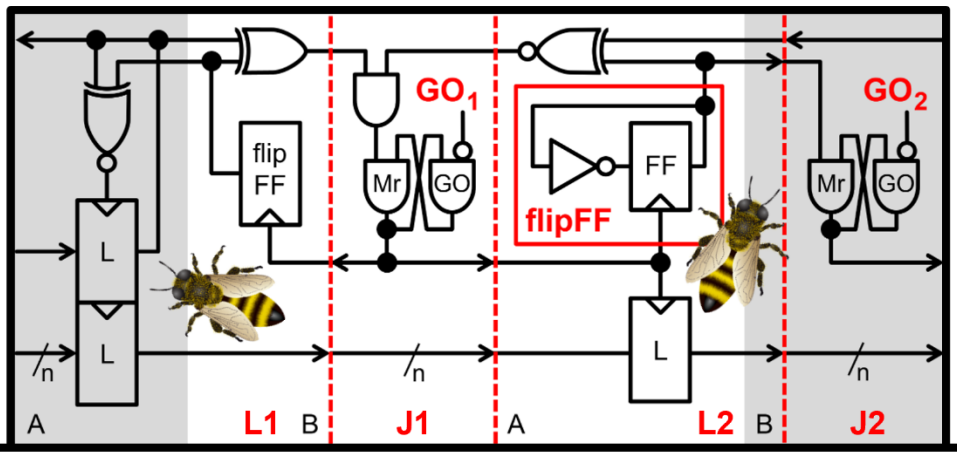
Gate-level simulations show data zipping by L2 for 34 times when we circulate one data element for a given run time. This becomes 72 times with two data elements and 55 times with three.

This CANOPY graph reports the simulation results. The take-away is that our bee does great work. *Our simulation uses abstract gate and delay models, so take the counts with a grain of salt.*

CHAPTER 4

Where Click takes us from transition to level signaling, and back...





Click's flipping flipflop, flipFF, introduced by Ad Peeters et al. in ASYNC 2010 is ideal for connecting transition and level signaling. The Click version that we use comes from our ASYNC 2015 paper and is explored in detail in ASYNC 2019 by Adrian Mardari, Zuzana Jelčicová, and Jens Sparsø as *phase-decoupled.*

We use the flipFF outputs here as both a transition signal and a level signal. For the transition-based part of the design the flipFF output is a transition, but the level-based part sees it as a level signal.

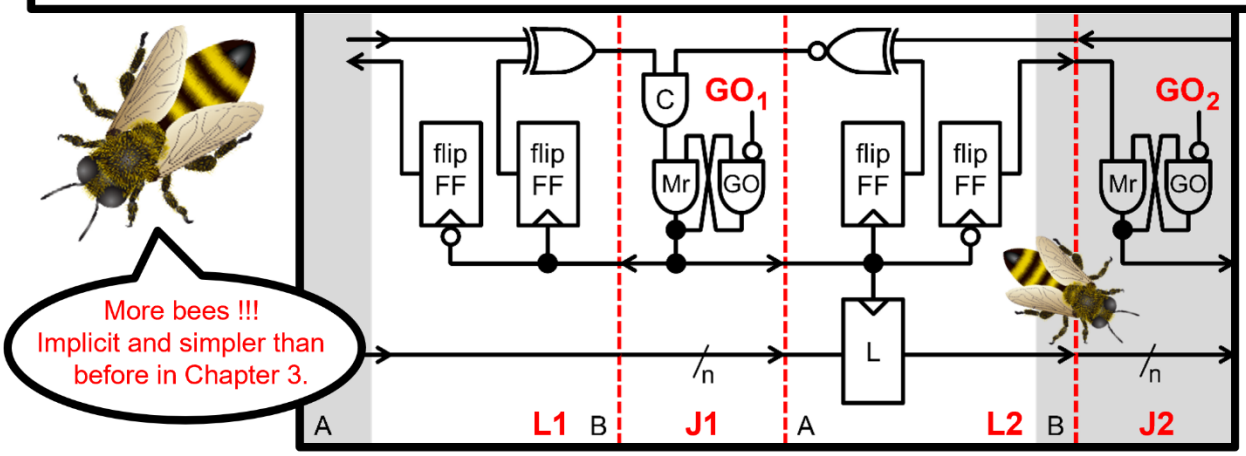
The flipFF does double-duty as "bee" and produces the desired transitions provided we initialize its state properly.

The top and bottom pictures use a grey background for transition-based parts, and white for level-based parts.

Link **L1** combines Mousetrap with Click to forward data from a design part that uses a transition signaling protocol to Joint **J1** which uses a 2-phase (top) or 4-phase (bottom) level signaling protocol. The data are stored by the Mousetrap — replaceable with Micropipeline.

L2 uses Click to store and forward data elements from level-based Joint **J1** to transition-based Joint **J2**.

The 4-phase Click circuits (bottom) are based on our ASYNC 2023 paper.



CHAPTER 5

Where we learn to scan Link states for initialization and at-speed test...

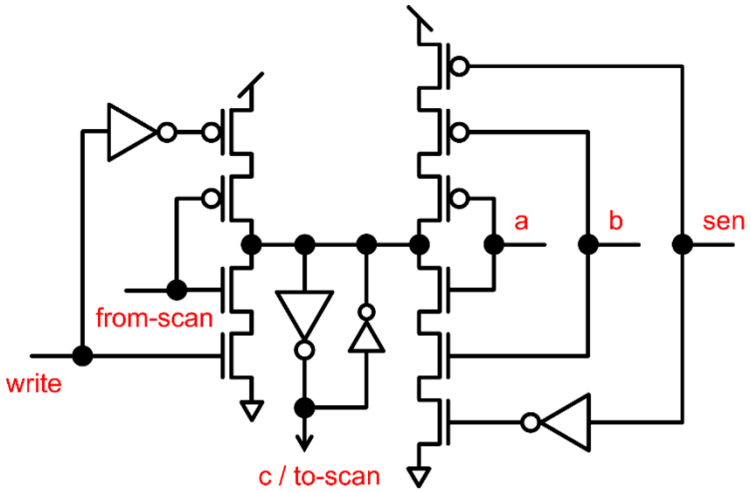
Our scan solution combines ASYNC 1994 by Ajay Khoche and Erik Brunvand with VTEST 1995 by the Manchester team of Oleg Petlin and Steve Furber and brings these into our ASYNC 2015 test approach.

Like Khoche-Brunvand, we scan the C element that controls Micropipeline.

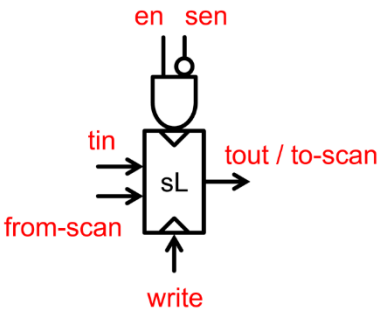
But we borrow the dominant scan mode from Petlin-Furber — as scan enable, *sen*.

And we use parallel scan chains to read and write Link control states and Joint GO signals as we explain in Chapter 7 of the 2020 IET book by Jia Di and Scott Smith.

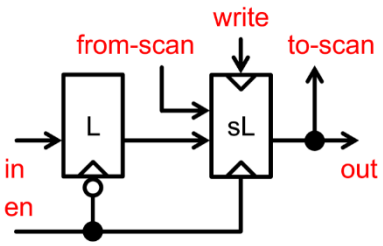
Micropipeline C element

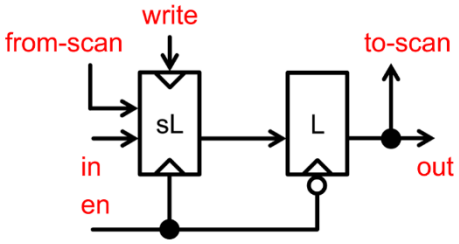


Mousetrap transition latch

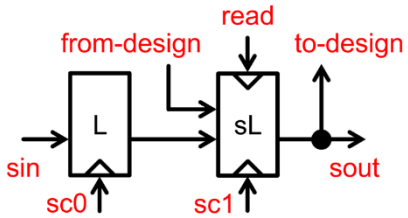


Click (flipping) flipflops





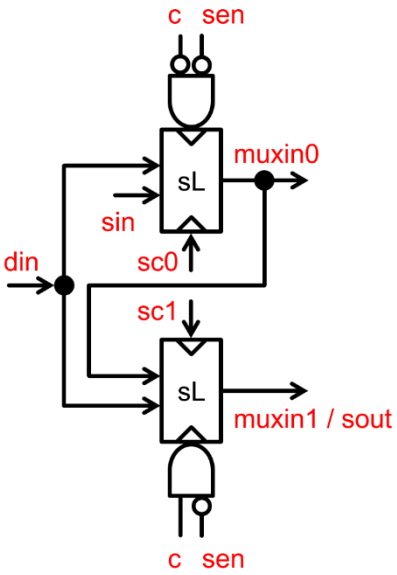
parallel scan chain



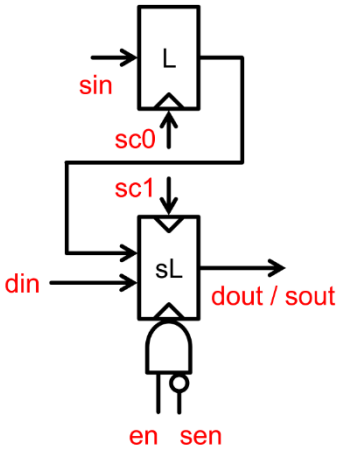
A Click flipping flipflop, flipFF, stores control state. We read and write a flipFF using a parallel scan chain to keep its state stable during scan shifts. Because the flipFF enable signal, *en*, is driven by a Joint with a level signaling protocol, setting GO=0 makes *en*=0 and tells us which latch of the flipflop to write into. Therefore, Click has no need for a dominant scan enable signal.

Like Khoche-Brunvand and Petlin-Furber we shift serially through the data latches.

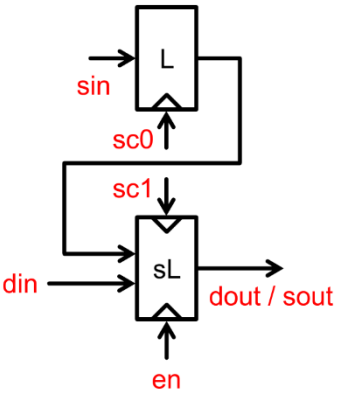
Micropipeline data latch



Mousetrap data latch



Click data latch



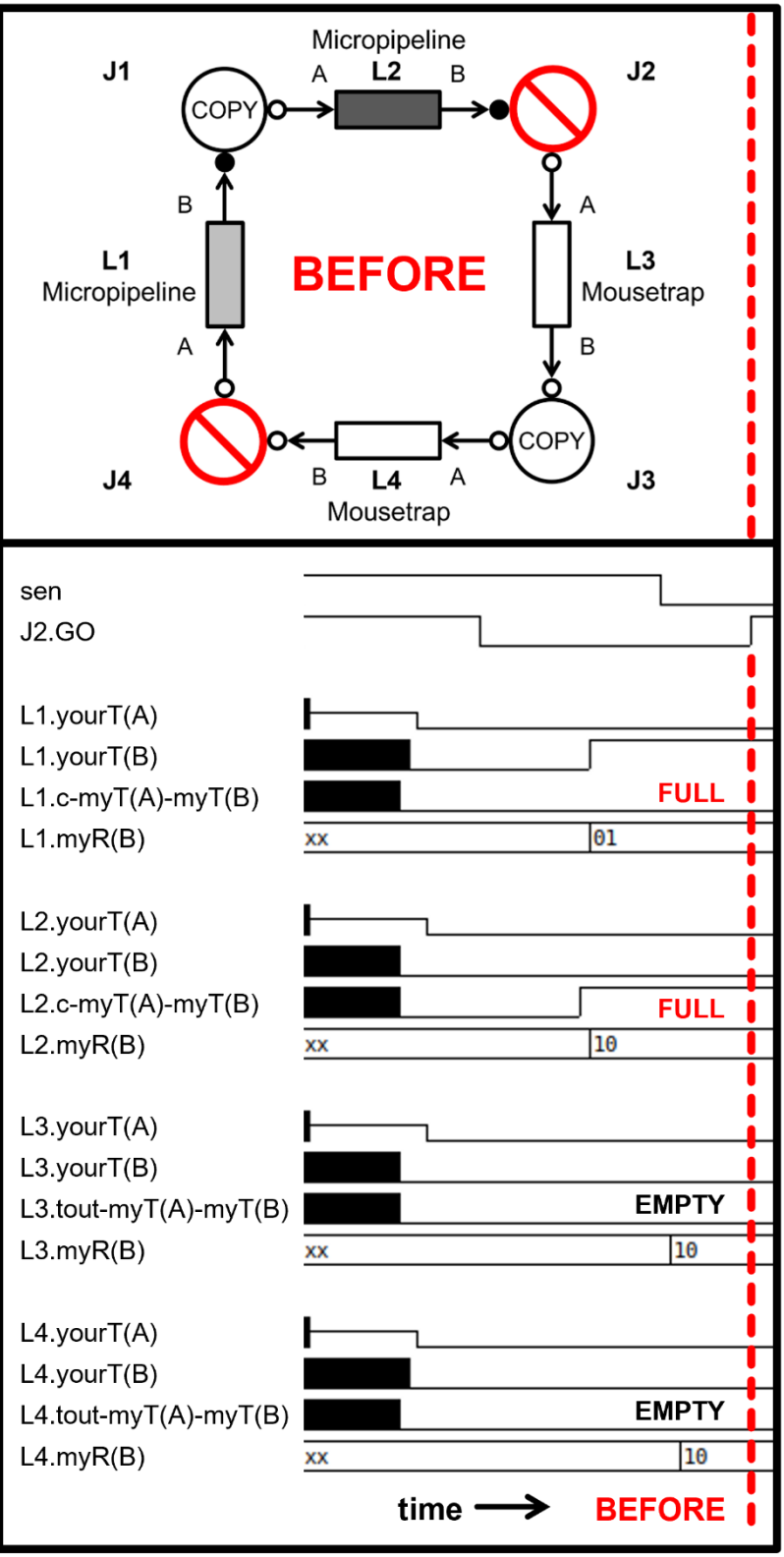
So, how did we use scan and GO signals to initialize the racetrack in Chapter 2?

We scan Link states and Joint GO signals only, so NOT the arbiters NOR the latches in a Joint MrGO for transition signaling.

We know each Link's initial state: FULL for L1-L2, and EMPTY for L3-L4. In level-signaling, we can set these states once GO=0 for each Joint, but transition signaling requires more preparation because MrGO must stop a specific transition.

So before scanning in the initial Link states, we first scan in the initial state of each MrGO, i.e., the initial state of C element input *a* or transition latch input *tin*. These are set by output *c* or *tout* of the predecessor Link, which we scan while GO=1. This is the point where scan enable signal, *sen*, comes in: ***sen* does double duty as a stop signal when GO can't play that role!**

Once each MrGO output is set, we freeze it with GO=0. Then, we scan in the initial Link states and data, make runways, and start the race by setting J2.GO=1.



3. Something Old, Something New, Lots Borrowed, a Little Blue

The graphic-style chapters on the previous pages show something old that was already known, something new that we added, and a lot that we borrowed and copied un-exactly from the old into the new.

Although the partitioning of Micropipeline and Mousetrap stages into Links and Joints in Chapter 1 is new, the effect on simple circuits like (ring) FIFOs is unsurprising. We omitted partitioning arbitrated or conditional Micropipeline and Mousetrap computations into Links and Joints, for two reasons. One, explaining more complex computations distracts from what we want to say. Second, most designers know how to implement arbitrated and conditional computations using level signaling. Therefore, as a new intermediary (or final) solution, Chapter 4 shows how to use Click or Click-enhanced Links to connect transition-based Joints to level-based Joints. Thus, we can still mix and match (ring) FIFOs and arithmetic data computations for which signaling protocols work well with computations that involve flow control.

The elephant in the room in Chapter 3 is an old friend, too often ignored. The Sparsø-Furber books alert designers of transition-signaling systems of a potential problem with odd-even numbers of elements in a ring (Figure 9.8)[17]. The scan-programmable solution in Chapter 3, called *bee*, though simple, may be new. That the Click flipFF in Chapter 4 does double-duty as *bee* is a new revelation, favorable to scaling. We expect similar "double-duty" *bees* in the complex Mousetrap stages that Montek Singh presented at the ASYNC 2022 Summer School [1]. This bodes well for transition systems at large.

Also new is the MrGO circuit in Chapter 2, which makes a self-timed transition stop-able. The racetrack simulation waveforms and canopy graph in Chapters 2, 3 and 5 show its use for initialization and test. The scan solutions in Chapter 5 borrow key elements from early test work by both the University of Utah [8] and the University of Manchester [10, 11], and combine these with the new MrGO into an at-speed test approach. New here is the interplay between the GO signals and the scan enable signal, *sen* — *sen* does double-duty as a stop signal when GOs can't play that role because they must allow transitions, as we explain in Chapter 5.

Ethical Statement

No Micropipeline, Mousetrap or Click circuits, nor any transition or level signals, and certainly no elephants and no bees were harmed in the making of this Festschrift salute.

References

1. ASYNC Summer School. 2022. https://asyncsymposium.org/async2022/

2. K. van Berkel, R. Burgess, J. Kessels, M. Roncken, F. Schalij, and A. Peeters. 1994. Asynchronous Circuits for Low Power: A DCC Error Corrector. *IEEE Des. Test*, **11**, 22-32. (doi:10.1109/54.282442)

3. K. van Berkel, R. Burgess, J. Kessels, A. Peeters, M. Roncken, F. Schalij, and R. van de Wiel. 1995. A Single-Rail Re-Implementation of a DCC Error Detector Using a Generic Standard-Cell Library. *Proc. WCADM*, 72-79. (doi:10.1109/WCADM.1995.514644)

4. E. Esimai and M. Roncken. 2023. Flexible Compilation and Refinement of Asynchronous Circuits. *Proc. ASYNC,* 109-119. (doi:10.1109/ASYNC58294.2023.10239623)

5. S.B. Furber, P. Day, J.D. Garside, N.C. Paver, and J.V. Woods. 1994. AMULET1: A Micropipelined ARM. *Proc. COMPCON*, 476-485. (doi:10.1109/CMPCON.1994.282880)

6. S.B. Furber and P. Day. 1996. Four-Phase Micropipeline Latch Control Circuits. *TVLSI* **4**, 247-253. (doi:10.1109/92.502196)

7. G. Gill, V. Gupta, and M. Singh. 2008. Performance Estimation and Slack Matching for Pipelined Asynchronous Architectures with Choice, *Proc. ICCAD*, 449-456. (doi:10.1109/ICCAD.2008.4681614)

8. A. Khoche and E. Brunvand. 1994. Testing Micropipelines. *Proc. ASYNC*, 239-246. (doi:10.1109/ASYNC.1994.656316)

9. A. Peeters, F. te Beest, M. de Wit, and W. Mallon. 2010. Click Elements: An Implementation Style for Data-Driven Compilation. *Proc. ASYNC*, 3-14. (doi:10.1109/ASYNC.2010.11)

10. O.A. Petlin and S.B. Furber. 1995. Scan Testing of Micropipelines. *Proc. VTEST*, 296-301. (doi:10.1109/VTEST.1995.512652)

11. O.A. Petlin and S.B. Furber. 1995. Scan Testing of Asynchronous Sequential Circuits. *Proc. GLSVLSI*, 224-229. (doi:10.1109/GLSV.1995.516057)

12. M. Roncken, S. Mettala Gilla, H. Park, N. Jamadagni, C. Cowan, and I. Sutherland. 2015. Naturalized Communication and Testing. *Proc. ASYNC*, 77-84. (doi:10.1109/ASYNC.2015.20).

13. M. Roncken and I. Sutherland. 2020. Chapter 7: Design and Test of High-Speed Asynchronous Circuits. In J. Di and S.C. Smith (eds.) Asynchronous Circuit Applications, *IET, London, UK*, 113–171. (doi:10.1049/PBCS061E)

14. C. Seitz. Chapter 7: System Timing. 1979. In C. Mead and L. Conway (eds) Introduction to VLSI Systems, pages 218–262. *Addison-Wesley*. (ISBN:978-0-201-04358-7)

15. M. Singh and S.M. Nowick. 2007. MOUSETRAP: High-Speed Transition-Signaling Asynchronous Pipelines. *TVLSI* **15**, 684-698. (doi:10.1109/TVLSI.2007.898732)

16. J. Sparsø and S. Furber (eds). 2001. Principles of Asynchronous Circuit Design — A Systems Perspective. *Kluwer Academic Publishers. (ISBN:0-7923-7613-7)*

17. J. Sparsø. 2020. Introduction to Asynchronous Circuit Design. *DTU Compute*, Technical University of Denmark. (ISBN:978-87-643-2001-5)

18. I.E. Sutherland. 1989. Micropipelines. *CACM* **32**, 720-738. (10.1145/63526.63532)

19. T.E. Williams, M. Horowitz, R.L. Alverson, and T.S. Yang. 1987. A Self-Timed Chip for Division. *Proc. ARVLSI*, 75-95.