

**19 February 2016 10:30 am**

Understanding Self-timed Circuits

Drs Ivan Sutherland & Marly Roncken
Turing Laureate Kyoto Laureate**(Computer Science 'Nobel Prize')**

Nearly all modern digital computers march to the beat of a "clock." The computer clock divides each second into a few billion "clock periods" just as a school bell divides each day into fixed-length class periods. A 55-minute class period is so useful for scheduling students and classrooms that educators rarely ask if it is best for learning. In reality, 55 minutes is either too short or too long.

We are one of a few research groups who study how to replace the rigid clock with more flexible "self-timed" regimes. Self-timed systems allow each small task to take its own natural time just as "self-paced" learning allows each student to learn at his or her own natural pace. Easy tasks finish quickly and take little energy. Difficult tasks require more time and energy.

In operation, a self-timed system is as orderly as a kindergarten playground at recess. Marly and Ivan will show how the parts of such a system interact, and how they can be tested.

**Venue: Executive
Seminar Room
(S2.2-B2-53)****Great insights from****The
'Asynchronous Duo':
Pioneers of
Self-Timed Logic****Friday
19 February, 2016
10:30 am – 12:30 pm****Refreshments
Provided****RSVP to
virtus@pmail.ntu.edu.sg
by
10 February, 2016**



Understanding Self-Timed Circuits

Talk Series Outline

Marly Roncken and Ivan Sutherland

Asynchronous Research Center (ARC)
Maseeh College of Engineering and Computer Science
Portland State University
January-February 2016

Understanding Self-Timed Circuits

PART I

TUESDAY afternoon, 16 Feb 2016

1. Introduction (Marly)
2. Self-Timed Circuits (Ivan)
3. Building Blocks and Protocols (Marly)
4. Measuring Performance (Ivan)

PART II

WEDNESDAY morning, 17 Feb 2016

5. Timing Validation (Marly)
6. Arbitration: Who wins? (Ivan)
7. Initialization, Test, and Debug (Marly)
8. The Weaver (Ivan)


Understanding Self-Timed Circuits

1. What and where is the ARC?

Marly Roncken and Ivan Sutherland

Asynchronous Research Center (ARC)
Maseeh College of Engineering and Computer Science
Portland State University
January-February 2016

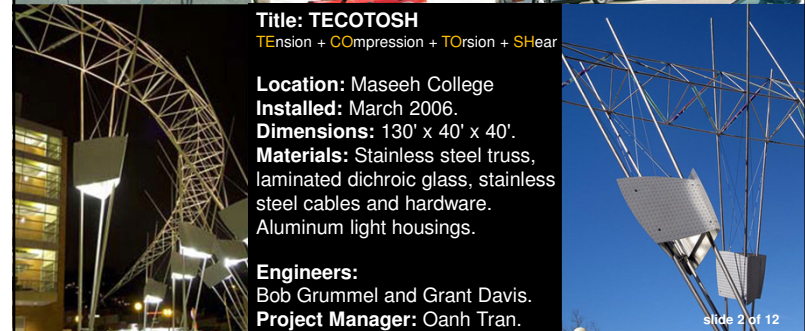
slide 1 of 12



Title: TECOTOSH
TEnsion + COmpression + TOrsion + SHear

Location: Maseeh College
Installed: March 2006.
Dimensions: 130' x 40' x 40'.
Materials: Stainless steel truss, laminated dichroic glass, stainless steel cables and hardware. Aluminum light housings.

Engineers:
Bob Grummel and Grant Davis.
Project Manager: Oanh Tran.



slide 2 of 12

Asynchronous Research Center (ARC)



Asynchronous Research Center, Portland State University, 1900 SW 4th Ave, FAB 105, Portland, OR 97201, USA

Understanding Self-Timed Circuits — 1. The ARC at the Maseeh College of Portland State University

slide 3 of 12

Asynchronous Research Center (ARC)

ARC Faculty



Marly Roncken
Director
CS Department



Ivan Sutherland
ECE Department



Willem Mollon
(2010-2012)
ARCwelder compiler
Click circuits

ARC Students



Prachi Padwal
MSc (2012)
Leakage-Power reduction



Rajesh Nerkar
MSc (2013)
DRAM interfacing



Navaneeth Jamadagni
PhD (2015)
Datapath topologies



Hoon Park
PhD (2015)
ARCTimer timing verification



Swetha Mettala Gilla
MSc (2010): STA
PhD: Compilation+Test



Chris Cowan, MD
PhD: Radiation tolerance

Visiting Students



William Koven
2009, 2011 (Harvey Mudd)
Click circuits, CPU



Jean Simatic
2013 (EP, France)
Designing with ARCwelder



Andrew Yang
2014 high-school student
ARCwelder GUI software



Ben Massey
2015 high-school student
Silicon test software



Chao Zhou
2014 Lanzhou University
ARCTimer software

Collaboration



Prof. Xiaoyu Song
ECE Department



Prof. Rob Daasch
ECE Department



Prof. Warren Hunt
UT Austin



Prof. Anping He
Lanzhou University
China

External Students

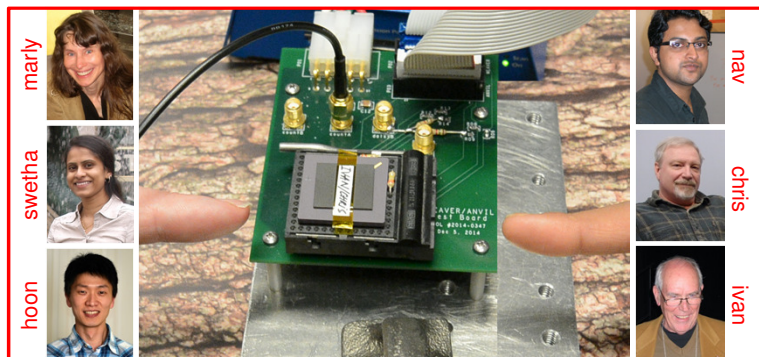


Chao Zhou
2014 Lanzhou University
ARCTimer software

Understanding Self-Timed Circuits — 1. The ARC at the Maseeh College of Portland State University

slide 4 of 12

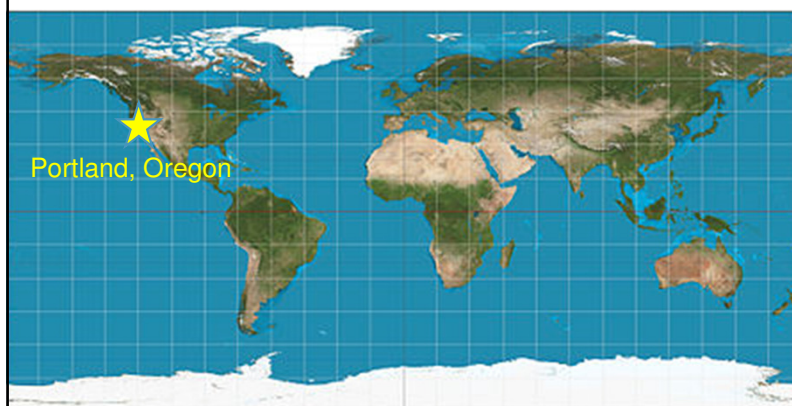
Get real!



A sample about our work

- Hoon Park, Anping He, Marly Roncken, Xiaoyu Song, Ivan Sutherland
Modular Timing Constraints for Delay-Insensitive Systems
JCST, Springer China, 31(1):77-106, 2016
- Hoon Park, Anping He, Marly Roncken, and Xiaoyu Song
Semi-modular delay model revisited in context of relative timing
IET Electronics Letters, 51(4):332-334, 2015
- Marly Roncken, Swetha Mettala Gilla, Hoon Park, Navaneeth Jamadagni, Chris Cowan, and Ivan Sutherland *Naturalized Communication and Testing*
ASYNC 2015, pages 77-84, 2015 (presentation on ASYNC 2015 web site)
 - *Poster by Hoon on Timing Validation* (see ASYNC 2015 web site)
 - *Poster by Swetha on Testing with MrGO* (see ASYNC 2015 web site)
- Ivan Sutherland *GasP Circuits that Work*
ECE 507 Seminar, Fall 2010, Asynchronous Research Center.
<http://arc.cecs.pdx.edu/fall10>

Where on earth are we? (1/2)



Equirectangular (plate carrée) projection of the world [Wikipedia]

Where on earth are we? (2/2)



the state of OREGON

Portland, the largest city in Oregon and seat of Multnomah County, is located in the northwest part of the state on the Willamette River.

Over **1,700** high-tech companies are located in the metropolitan area.

Portland is one of the fastest growing tech sectors in the country, and home to major industrial players in

- Microelectronics
- Energy and power
- Manufacturing
- Transportation and other infrastructure technology.

Portland boasts a thriving creative, recreation and culinary culture.

Regional industry



Understanding Self-Timed Circuits — 1. The ARC at the Maseeh College of Portland State University

slide 9 of 12

Maseeh College at a glance



94 faculty members with 86 teaching faculty

5 departments:

- Civil and Environmental Engineering
- Electrical and Computer Engineering
- Mechanical and Materials Engineering
- Computer Science
- Engineering and Technology Management (MS and PhD only)

2270 undergrads*

539 grad students*

~20% international students

* Totals on 19 August 2015.

Understanding Self-Timed Circuits — 1. The ARC at the Maseeh College of Portland State University

slide 10 of 12

Portland State University: GO Vikings!



Understanding Self-Timed Circuits — 1. The ARC at the Maseeh College of Portland State University

slide 11 of 12

Living in Portland



canoeing on the Willamette

excellent cuisine from fresh produce

Powell's book store - a city block of books

skiing and snowboarding on Mount Hood

Understanding Self-Timed Circuits — 1. The ARC at the Maseeh College of Portland State University

slide 12 of 12



Understanding Self-Timed Circuits

2. Self-Timed Circuits

Marly Roncken and Ivan Sutherland

Asynchronous Research Center (ARC)
Maseeh College of Engineering and Computer Science
Portland State University
January-February 2016

slide 1

Self-Timed Circuits

Asynchronous Research Center

Outline

- Kinetic Learning Activity (KLA)
- Pipelines
- Protocols
 - > Bundled Data
 - > Globally Asynchronous Locally Synchronous (GALS)

February, 2016

Slide 2

Self-Timed Circuits

Asynchronous Research Center

KLA rules

- Predecessor and successor
- Use only one hand
- Pred has object AND you don't
- **Do**: TAKE from predecessor
- **Never**: PUT to successor

February, 2016

Slide 3

Self-Timed Circuits

Asynchronous Research Center

Pipeline action

- Conditions for action
 - > predecessor proffers data
 - > successor proffers space
- Three part atomic action:
 - > copy data
 - > make successor FULL
 - > make predecessor EMPTY

February, 2016

Slide 4

Pipeline essentials

- One AND function
 - > pred FULL *and* succ EMPTY
- Data captured in latches or flip flops
- Some relative timing assumptions
- Compare with source clocking

Pipeline is:

- Logic stage “L”
- Wires “w”

w L w L w L w L w L w L w

- Wires may hold data or not:
can be “full” *or* “empty”

Bundled Data

A Data “bundle” is

- N data wires (the message)
- plus*
- Two-way handshake
 - > Validity signal – aka “request”
 - > Acknowledge – ok to send next
- with delay constraint*
- Sender and receiver alternate

Control protocols

- Two wires – four phase
 - > request HI = data valid = FULL
 - > acknowledge HI = no meaning
 - > request LO = no meaning
 - > acknowledge LO = data accepted = EMPTY
- Two wires – two phase NRZ
 - > request change = data valid = FULL
 - > like double data rate (DDR) source clock
 - > acknowledge change = EMPTY

Source clocking

- Source clocking
 - > source clock moves data forward
 - > Double Data Rate (DDR) = on each edge

Source and sink clocking

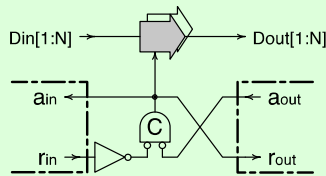
- Source clocking
 - > source clock moves data forward
 - > Double Data Rate (DDR) = on each edge
- Sink clocking
 - > sink clock moves space backward
 - > DDR = on each edge

Data Protocol

- Normally transparent
 - > Data flows forward unimpeded
 - changes may cost energy and cause noise
 - > Control locks space behind data
 - > Like rods in grocery store
- Normally opaque (I favor)
 - > Control paves the way
 - > Like a snow plow
 - > Data captured at each stage

Micropipeline (1988)

- Normally transparent
- Two wires: request & acknowledge
 - > FULL when they differ



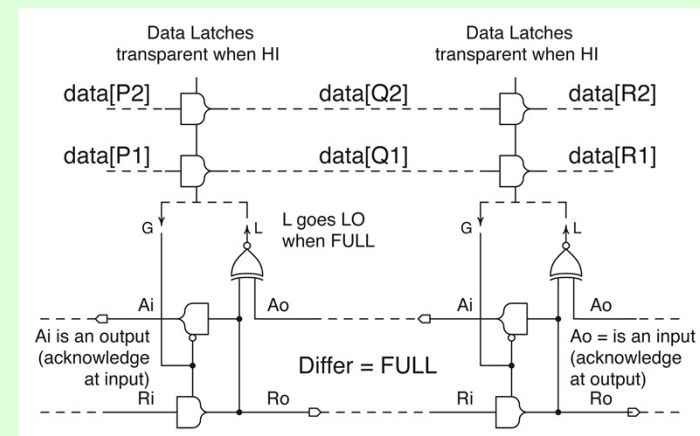
Micropipeline KLA rules

- Predecessor and successor
- IF (predecessor \neq successor)
- THEN copy predecessor
- ELSE hold value

Charlie box (2000)

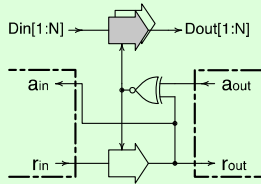
- Two wires between stages
 - > differ = FULL
 - > bundled data
 - > acknowledge after capture
- Fast in forward direction
 - > forward = 2 via one latch
 - > reverse = 4 via XOR + latch
- Normally transparent data latches

Charlie box circuit diagram



Mousetrap (2001)

- Normally transparent
- Two wires: request & acknowledge
 - > FULL when they differ
 - > Latch opaque when FULL



February, 2016

Slide 17

GasP (2002)

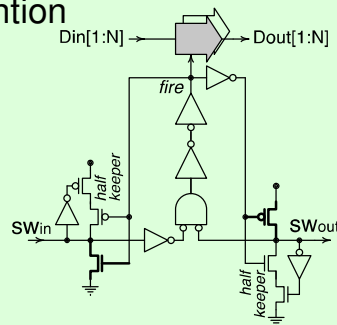
- One state wire
 - > sender changes control wire = FULL
 - > receiver changes control wire = EMPTY
 - > Conventions:
HI is FULL or LO is FULL
- Very simple pipeline control
 - > avoids XOR
 - > one AND function
 - > speed of 5 inverter ring oscillator

February, 2016

Slide 18

6-4 GasP circuit

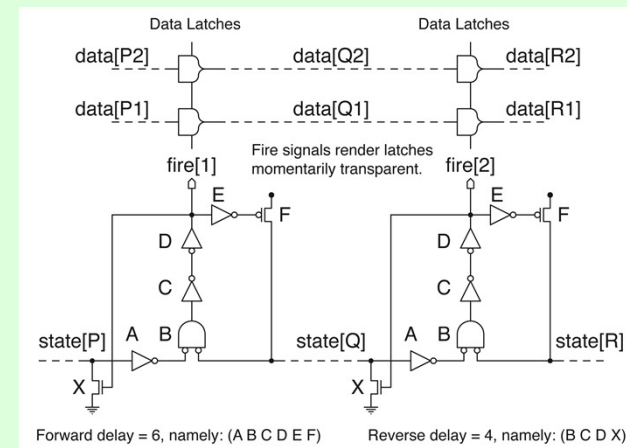
- Single control wire plus data
 - > Control wire is bidirectional
 - > HI is FULL convention
 - > Bundled data
 - > Normally opaque



February, 2016

Slide 19

GasP circuit diagram (HI = FULL)

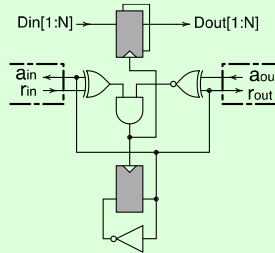


February, 2016

Slide 20

Click (2010)

- Normally opaque
- Two control wires:
 - > Request (r)
 - > Acknowledge (a)
 - > EMPTY if $r = a$
 - > FULL if $r \neq a$



February, 2016

Slide 21

Global state

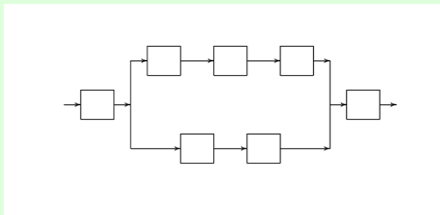
- Is an unnecessary fiction
- Handshakes isolate local actions
- Transactions are what matters
- Pipelines are easy to think about
 - > local transactions tell all
 - > avoid state explosion
 - > painless concurrency
 - > e.g. the UNIX pipe

February, 2016

Slide 22

Fork and join pipes

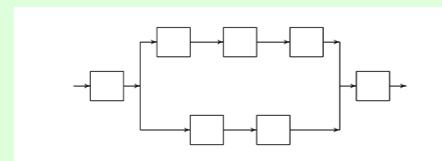
- Fork sends same data many ways
- Join combines many data inputs
- Fork then Join = parallel pipeline



February, 2016

Slide 23

Fork and join pipes



- storage capacity of the shorter
- latency of the slower
- *slack matching* avoids
 - > excess storage
 - > excess latency

February, 2016

Slide 24

Self-timing Challenges

- Tools
 - > CAD systems depend on clock
 - e.g. Static timing of logic loops
- Training
 - > Schools and texts teach clock
 - > Rigid thinking
- Management confidence
 - > Need examples

Unique validation tasks

- Combinational loops
- Slack matching
- Working with local state
- “Cycle accurate” meaningless

Unique verification tasks

- Relative delays
 - > like timing closure, but local
- Deadlock
 - > wormhole networks OK if no loops
 - > other cases?
- Non-determinism
 - > order may depend on delays

Self-timing Advantages

- Modularity
 - > Divide design and conquer
 - > Libraries
 - > Scalable to big systems
- Appropriate delay
 - > Easy cases can be fast
 - > Slow cases don't matter if rare
 - > Data dependent timing

Self-timing Advantages

- Separate function from layout
 - > Any layout works properly
 - > Layout sets speed
 - > New layout can
 - Improve performance
 - Reduce power
- Saves power

Self Timing is Inevitable

- Clocked paradigm has:
 - > Practical problem of confounding
 - Layout
 - Logic
 - Geometry
 - > Fundamental problem
 - “Simultaneous” not possible over space
- GALS is industry response
 - > Rich with synchronizers

Discussion

Understanding Self-Timed Circuits

3. Building Blocks and Protocols

Marly Roncken and Ivan Sutherland

Asynchronous Research Center (ARC)
Maseeh College of Engineering and Computer Science
Portland State University
January-February 2016

slide 1 of 68

Outline

- Intermezzo: Why asynchronous?
- Building blocks
 - Links and joints
 - Action and the role of full and empty
 - Systems of building blocks
- Building blocks with handshake interfaces
 - Handshake protocols
 - GasP and Click FIFO
 - Pros and Cons of handshake interfaces
- Building blocks with full-empty interfaces
 - GasP and Click revisited
 - Mixed GasP and Click FIFO
 - Pros and Cons of full-empty interfaces
- Summary and Conclusion

Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 2 of 68

INTERMEZZO: Why Asynchronous or Self-Timed?

Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 3 of 68

Motivation: why asynchronous? [1/9]

- Modern computer systems are distributed over space
- Examples:

- Internet of things

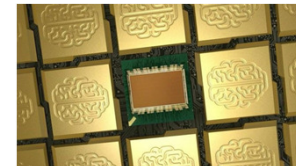
the network of physical objects or "things" embedded with electronics, software, sensors, and network connectivity, which enables these objects to collect and exchange data

[Wikipedia]



- IBM's TrueNorth

modular chips that act like neurons and form artificial neural networks to run "deep learning algorithms", like Skype's chat translator or Facebook's facial recognition



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

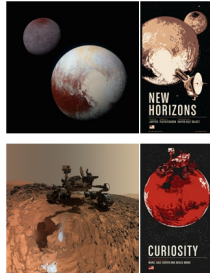
slide 4 of 68

Motivation: why asynchronous? [2/9]

- We can manage time + communication delay over space, provided
 - **synchronicity** is required only over short distances
 - long-distance communication relies on **causality** i.e. event orderings (+ communication delay)

Examples:

- It took 4 hours to get the final position commands from Earth to *New Horizons* near Pluto, in order to make this picture of Pluto and its moon Charon.
[www.planetary.org/blogs/emily-lakdawalla/2015/01300800-talking-to-pluto-is-hard.html]
- In August 2012, NASA landed *Curiosity* on Mars. At the time, Mars was 13 minutes talk-time away. The landing was on autopilot, the last 13 minutes.
[<http://mars.nasa.gov/msl>]



Curiosity "selfie"

Motivation: why asynchronous? [3/9]

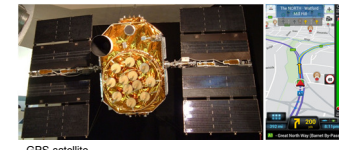
- Technology can make a space look smaller
 - so that **synchronicity** can be maintained over a longer distance

Examples:

- Rail transport took over transport by horse and introduced traveling "on time"
- Global Positioning System (GPS)
[Roger L. Easton, Ivan A. Getting, Bradford Parkinson, 1978]
- Internet (communication network)
[Internet protocols: TCP/IP Vint Cerf, Bob Kahn, 1974]
- World Wide Web (information space)
[Tim Berners-Lee, 1989]
- Email, Dropbox, Skype, TeamViewer for our Portland-Lanzhou meetings



After building a train station on Germany's railway, the hours on the lower clock in Speyer (top) no longer sufficed. An extra clock (bottom) was added to show the minutes in each hour.



GPS satellite

Motivation: why asynchronous? [4/9]

- Or technology can make a space look BIGGER
 - making us dive head on into **synchronicity limits**
- Example:

Nanometer chip technology enables more and faster transistors per chip but the wires scale less well, and are therefore much slower.

As a result:

 - Global clocks are too slow for modern chips
 - It takes **hundreds of clock domains**, ten thousand signals crossing them, and many clock ticks, to communicate within a single networking chip

[Jeanne Trinkl, IBM, Keynote Speech ASYNC 2013]

Modern chips are distributed systems
Why aren't we designing them as such?

ARC
JUST DO IT.

Motivation: why asynchronous? [5/9]

- We design and study hardware
 - distributed over self-timed components + communication protocols

Motivation: why asynchronous? [6/9]

- Inside a component

- behavior can be as chaotic as a kindergarten playground
- which is fine, because the space is small enough to synchronize events



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 9 of 68

Motivation: why asynchronous? [7/9]

- The communication protocols between the components

- are based on event orderings (+ communication delay)
- and are as orderly as a "crocodile"
- which is necessary for correctness (+ performance)



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 10 of 68

Motivation: why asynchronous? [8/9]

Summary:

We design and study hardware

- distributed over self-timed components + communication protocols

where

- Inside a component it can be as chaotic as a kindergarten playground which is fine, because components are small enough to control events
- Between components, the protocols are as orderly as a "crocodile" which guarantees that the communication is correct



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 11 of 68

Motivation: why asynchronous? [9/9]

- The playground system works, provided local supervisors

- oversee the playground to avoid accidents, bullying, fights
- ensure children use the "crocodile" between playgrounds

- Hardware analogy:

- accidents: wrong handshake protocol, ignored transitions, drive fights
- supervisors: event ordering constraints, a.k.a. **relative timing constraints**

- This is a **scalable** system because the supervisors are local



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 12 of 68

Outline

- Intermezzo: Why asynchronous?
- Building blocks
 - Links and joints
 - Action and the role of full and empty
 - Systems of building blocks
- Building blocks with handshake interfaces
 - Handshake protocols
 - GasP and Click FIFO
 - Pros and Cons of handshake interfaces
- Building blocks with full-empty interfaces
 - GasP and Click revisited
 - Mixed GasP and Click FIFO
 - Pros and Cons of full-empty interfaces
- Summary and Conclusion

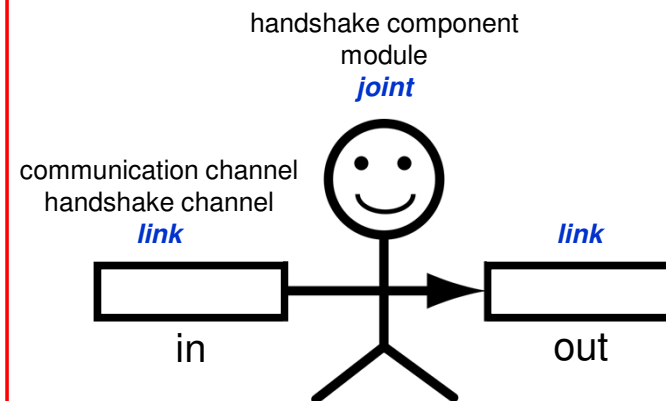
BUILDING BLOCKS

Building blocks

[Analogy reminder]



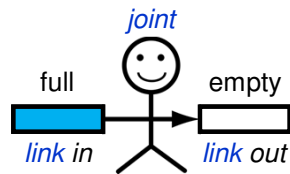
Building blocks



Building blocks: action

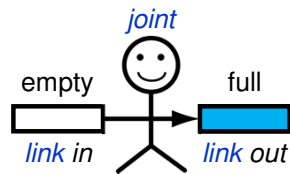
WHEN to act:

in is full
and
out is empty



WHAT to do:

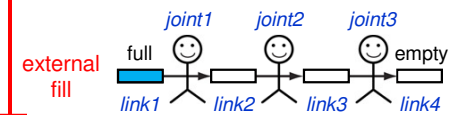
- copy data
- drain *in*
- fill *out*



Systems of building blocks (1/6)

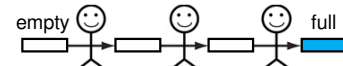
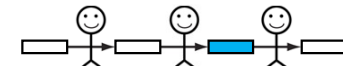
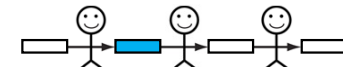
WHEN to act:

in is full
and
out is empty



WHAT to do:

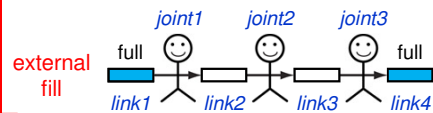
- copy data
- drain *in*
- fill *out*



Systems of building blocks (2/6)

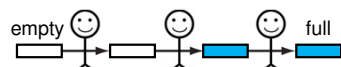
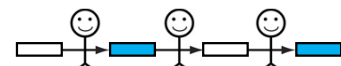
WHEN to act:

in is full
and
out is empty



WHAT to do:

- copy data
- drain *in*
- fill *out*

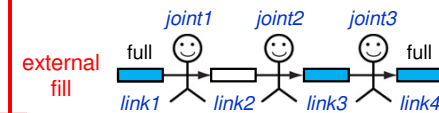


data have moved forward
as far as they can

Systems of building blocks (3/6)

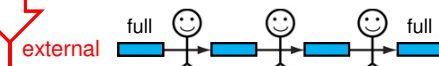
WHEN to act:

in is full
and
out is empty



WHAT to do:

- copy data
- drain *in*
- fill *out*



data have moved forward
as far as they can

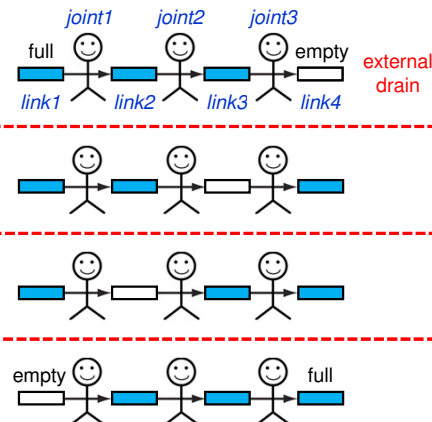
Systems of building blocks (4/6)

WHEN to act:

in is full
and
out is empty

WHAT to do:

- copy data
- drain *in*
- fill *out*



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 21 of 68

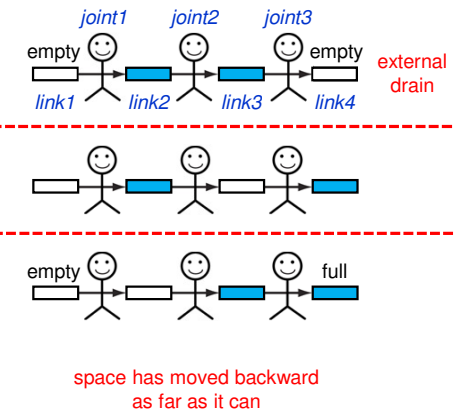
Systems of building blocks (5/6)

WHEN to act:

in is full
and
out is empty

WHAT to do:

- copy data
- drain *in*
- fill *out*



space has moved backward
as far as it can

Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 22 of 68

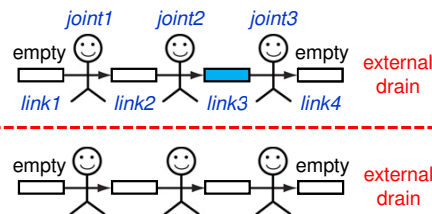
Systems of building blocks (6/6)

WHEN to act:

in is full
and
out is empty

WHAT to do:

- copy data
- drain *in*
- fill *out*



space has moved backward
as far as it can

Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 23 of 68

Outline

- Intermezzo: Why asynchronous?
- Building blocks
 - Links and joints
 - Action and the role of full and empty
 - Systems of building blocks
- Building blocks with handshake interfaces
 - Handshake protocols
 - GasP and Click FIFO
 - Pros and Cons of handshake interfaces
- Building blocks with full-empty interfaces
 - GasP and Click revisited
 - Mixed GasP and Click FIFO
 - Pros and Cons of full-empty interfaces
- Summary and Conclusion

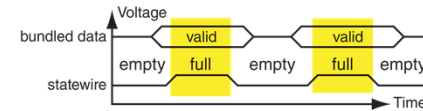
Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 24 of 68

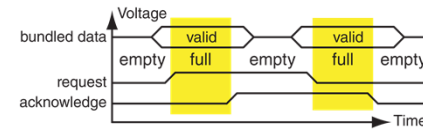
1st BUILDING BLOCK SOLUTION: with handshake interfaces

Handshake protocols

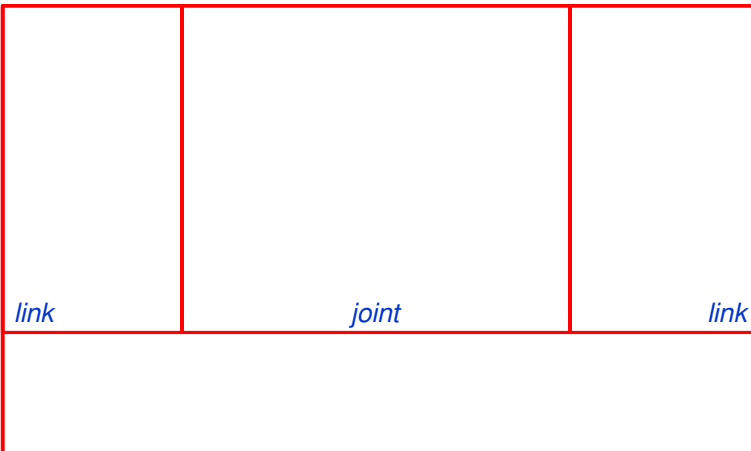
- Handshakes *encode* full, empty, and valid data when full
- Examples:
 - 2-phase return-to-zero (RTZ) with bundled data (used in GasP)
 - full: statewire is high / empty: statewire is low



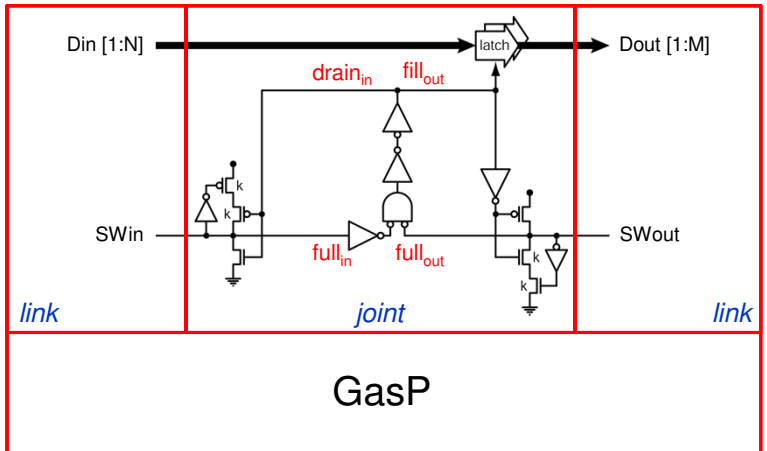
- 2-phase non-RTZ with bundled data (used in Click)
 - full: request \neq acknowledge / empty: request = acknowledge



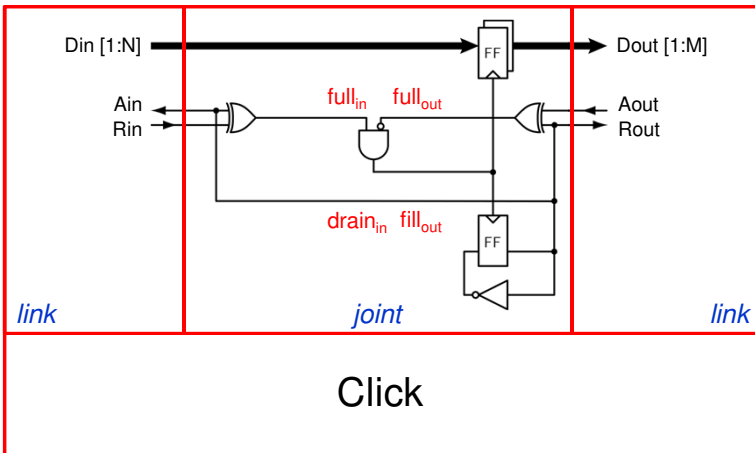
Building blocks with handshake interfaces



Building blocks with handshake interfaces



Building blocks with handshake interfaces



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

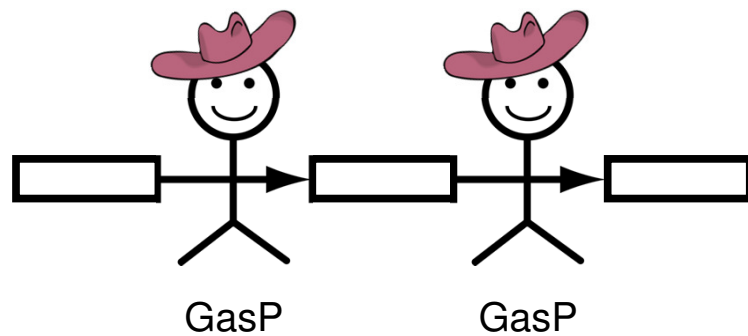
slide 29 of 68

so FAR so GOOD

Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 30 of 68

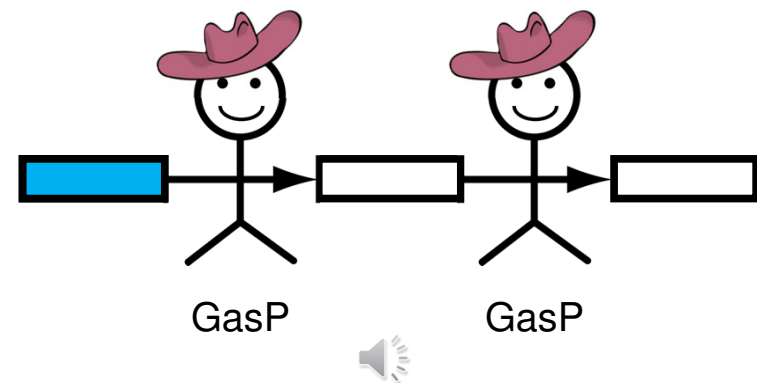
Systems with handshake interfaces (1^a/3)



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 31 of 68

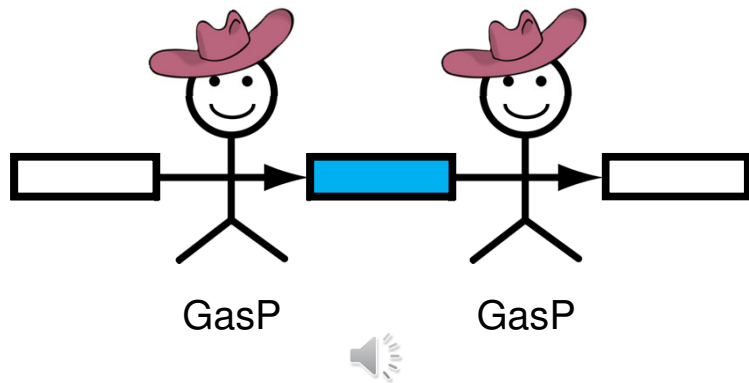
Systems with handshake interfaces (1^a/3)



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 32 of 68

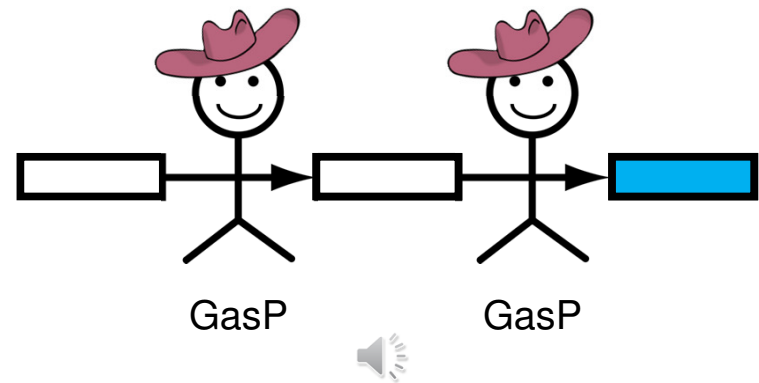
Systems with handshake interfaces ($1^b/3$)



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 33 of 68

Systems with handshake interfaces ($1^c/3$)



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

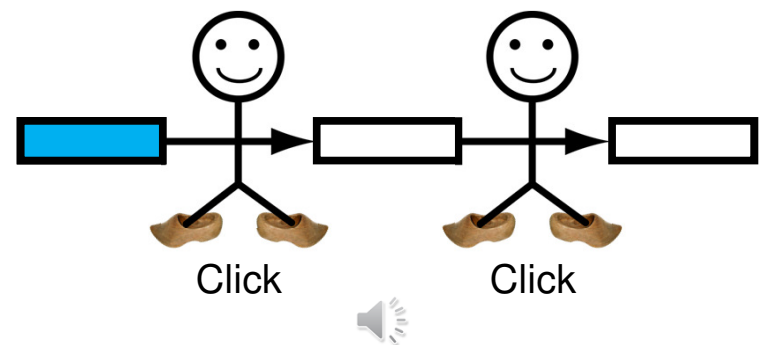
slide 34 of 68

so FAR so GOOD

Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 35 of 68

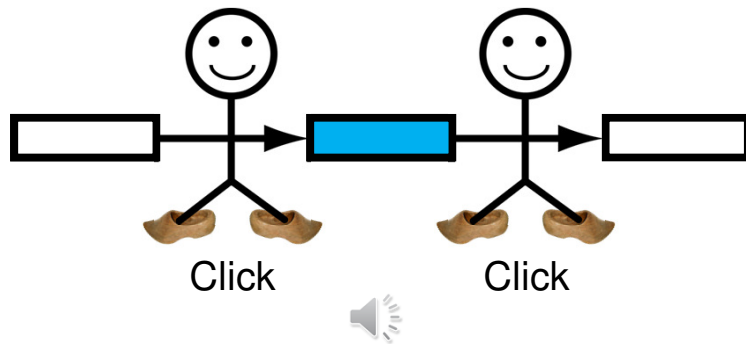
Systems with handshake interfaces ($2^a/3$)



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 36 of 68

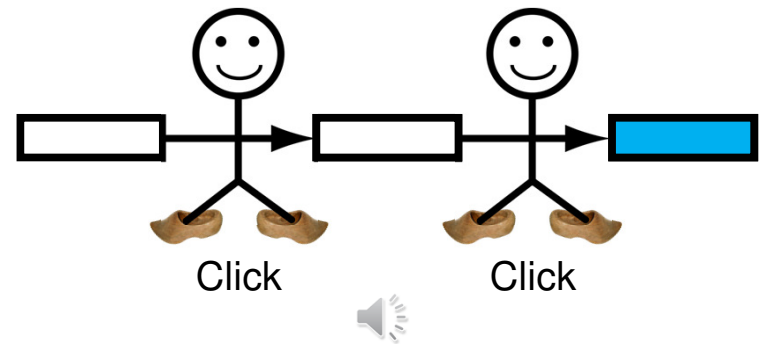
Systems with handshake interfaces ($2^b/3$)



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 37 of 68

Systems with handshake interfaces ($2^c/3$)



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

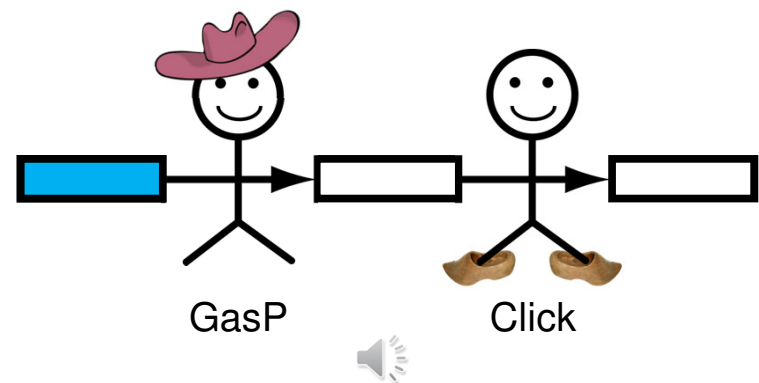
slide 38 of 68

so FAR so GOOD

Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 39 of 68

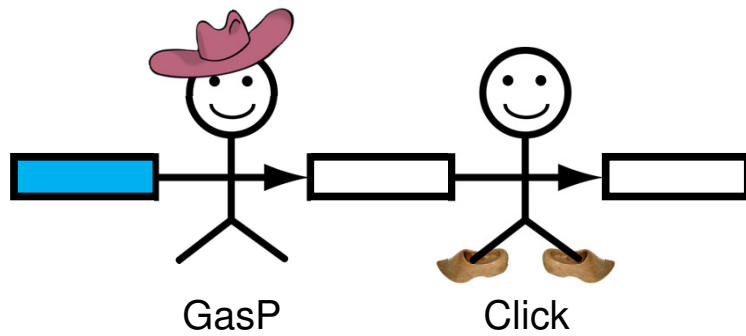
Systems with handshake interfaces ($3^a/3$)



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 40 of 68

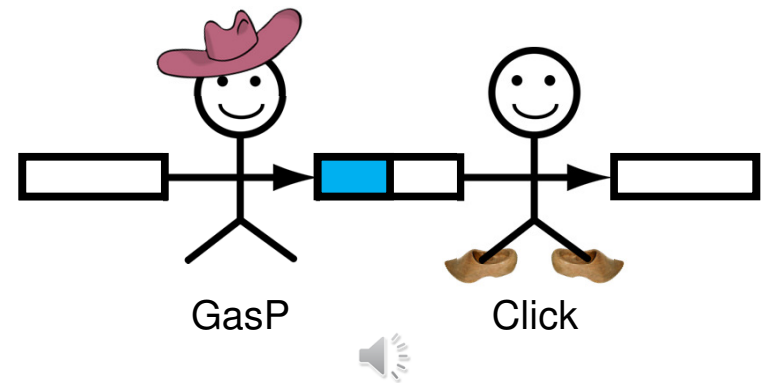
Systems with handshake interfaces ($3^b/3$)



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 41 of 68

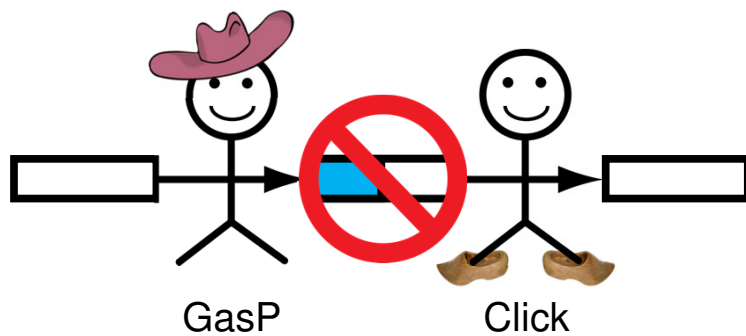
Systems with handshake interfaces ($3^b/3$)



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 42 of 68

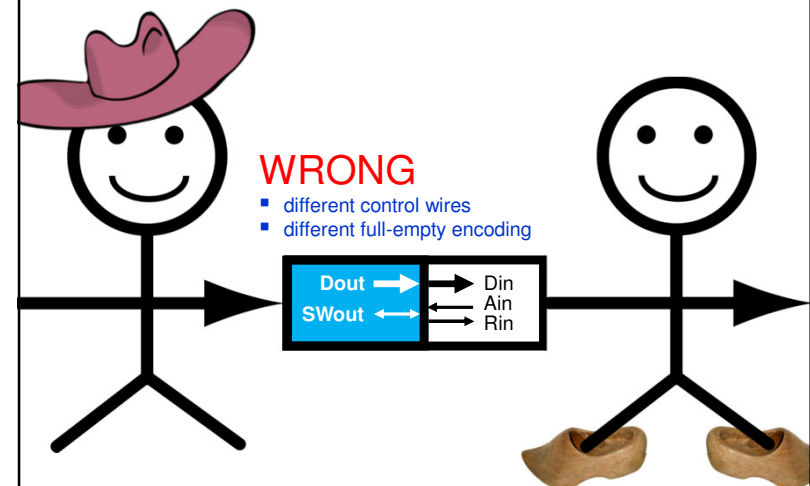
Systems with handshake interfaces ($3^c/3$)



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 43 of 68

Systems with handshake interfaces ($3^c/3$)

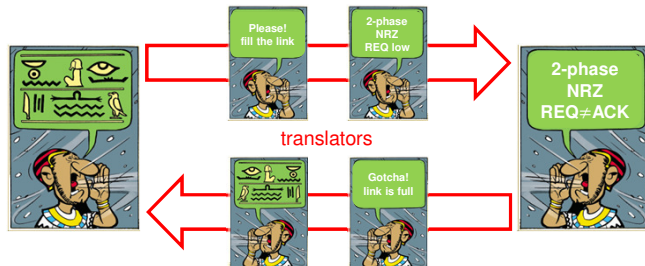


Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 44 of 68

Handshake interfaces: pros and cons

- Pros
 - Most R&D in asynchronous (self-timed) circuits use handshake interfaces
- Cons
 - Building blocks with different handshakes need translators to communicate
 - Translators cost extra validation effort + area, time, and power
 - This complicates collaboration and design re-use



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 45 of 68

Outline

- Intermezzo: Why asynchronous?
- Building blocks
 - Links and joints
 - Action and the role of full and empty
 - Systems of building blocks
- Building blocks with handshake interfaces
 - Handshake protocols
 - GasP and Click FIFO
 - Pros and Cons of handshake interfaces
- Building blocks with full-empty interfaces
 - GasP and Click revisited
 - Mixed GasP and Click FIFO
 - Pros and Cons of full-empty interfaces
- Summary and Conclusion

Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 46 of 68

2nd BUILDING BLOCK SOLUTION: with full-empty interfaces

Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

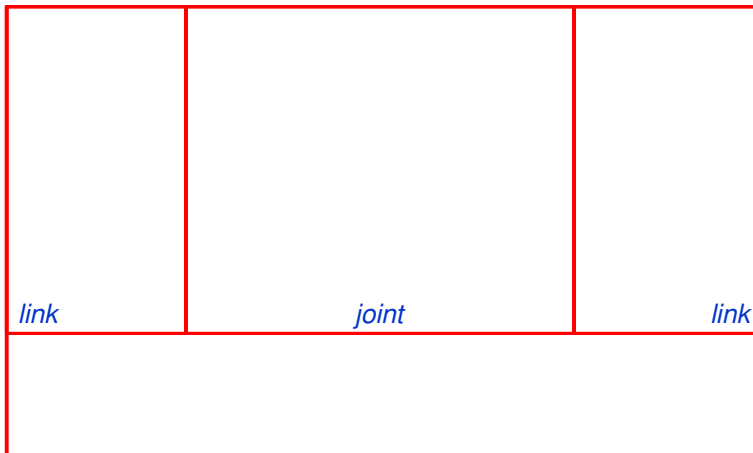
slide 47 of 68

GasP revisited

Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 48 of 68

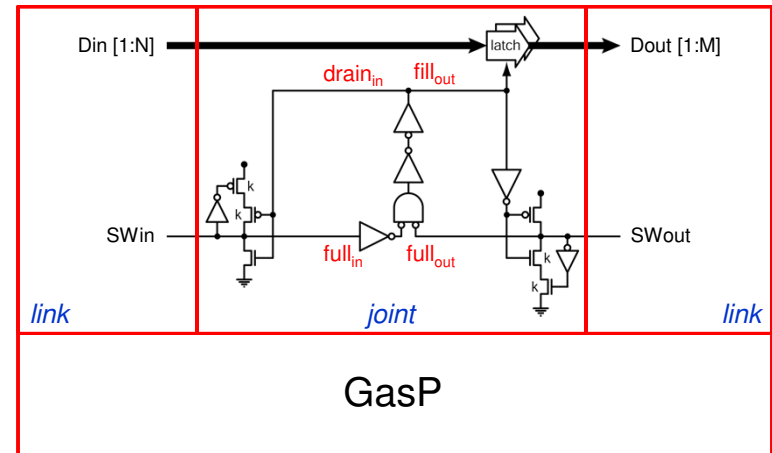
Building block interfaces revisited



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 49 of 68

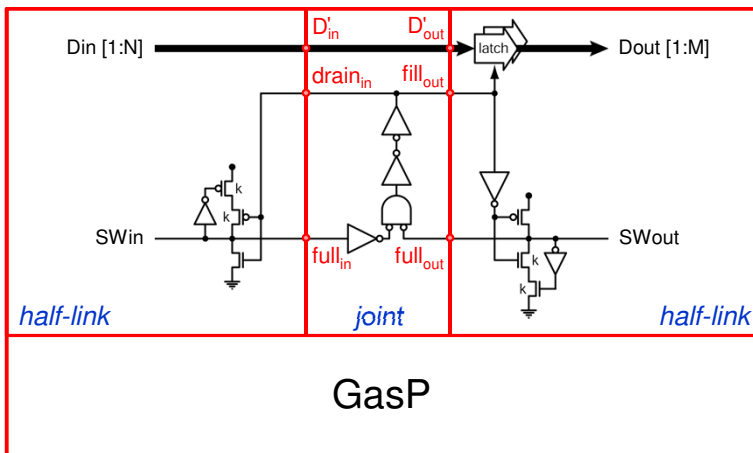
Building block interfaces ...from



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 50 of 68

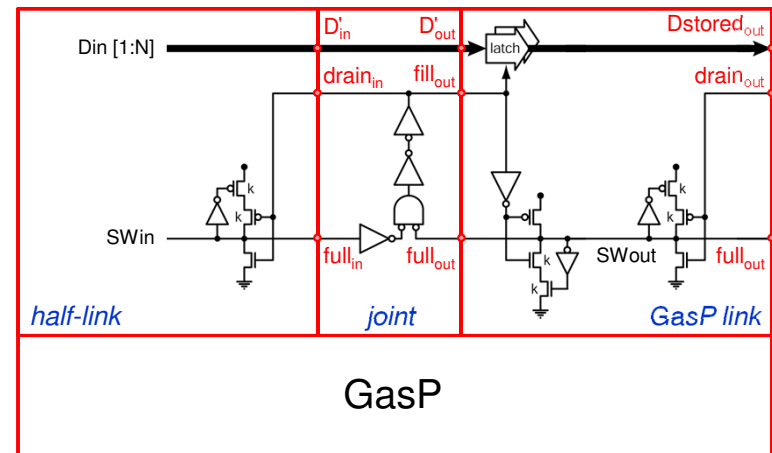
Building block interfaces ...to



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 51 of 68

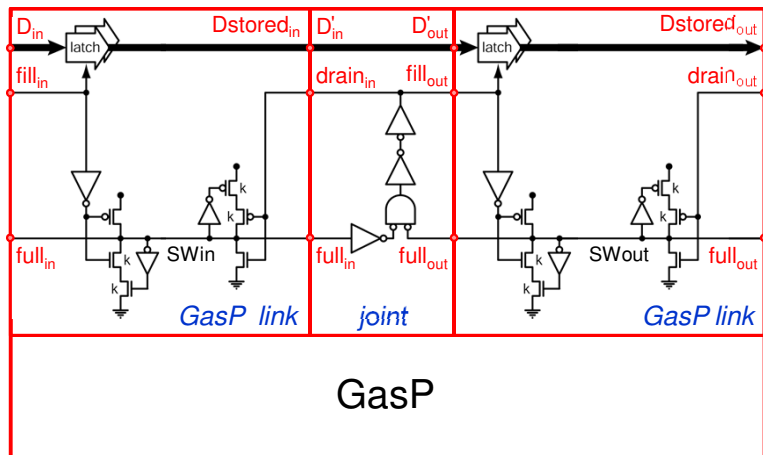
Building block interfaces ...to



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 52 of 68

Building block interfaces ...to



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

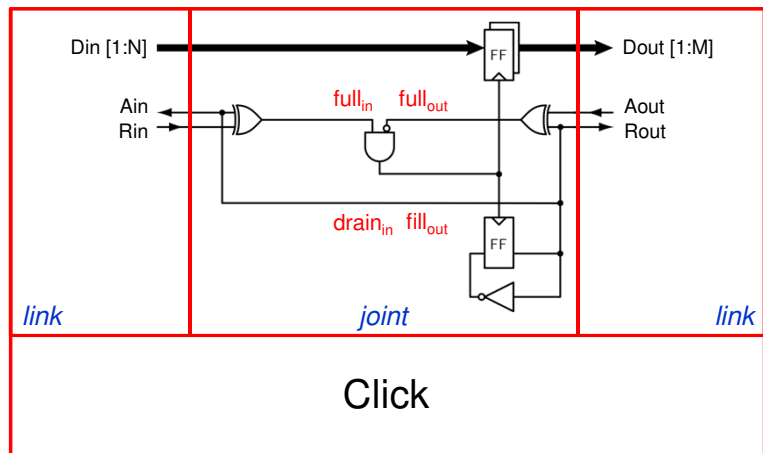
slide 53 of 68

Click revisited

Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 54 of 68

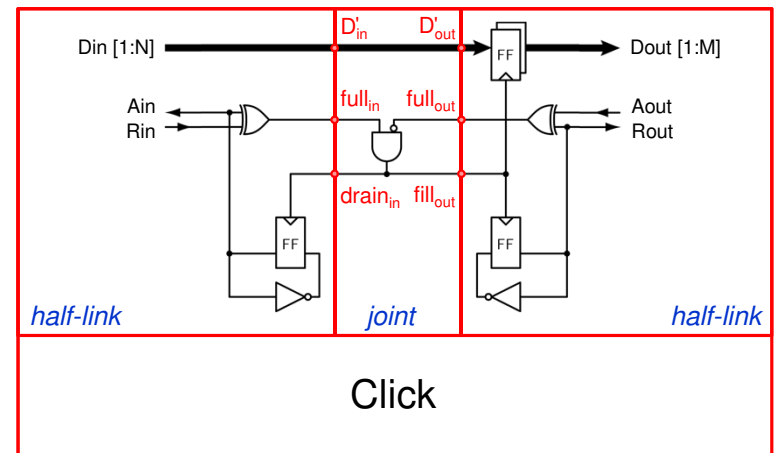
Building block interfaces ...from



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 55 of 68

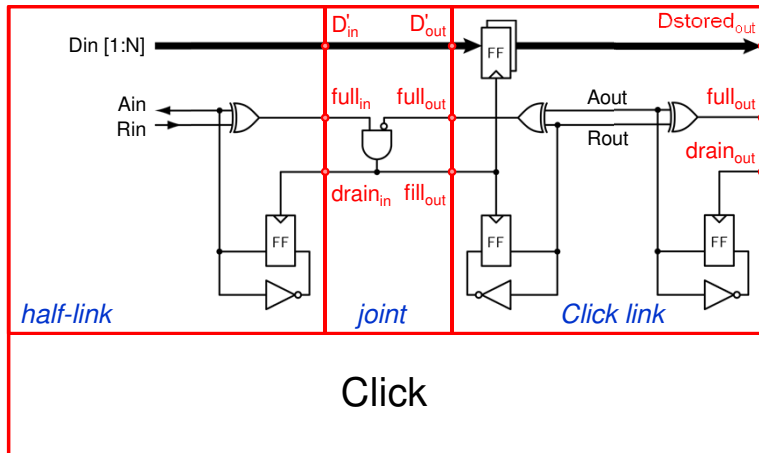
Building block interfaces ...to



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 56 of 68

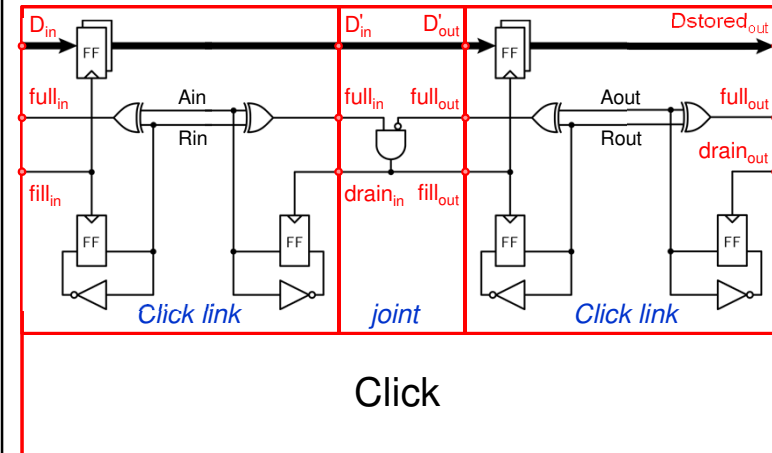
Building block interfaces ...to



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 57 of 68

Building block interfaces ...to



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

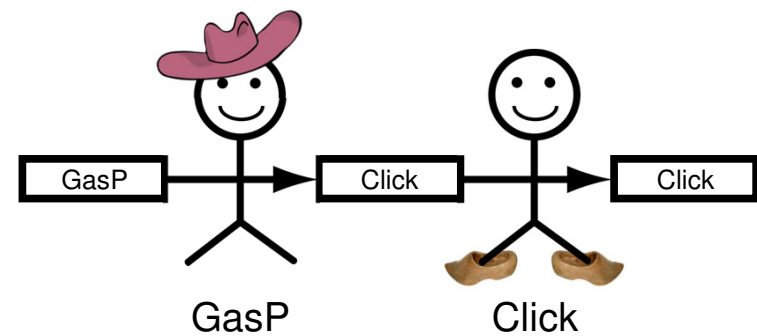
slide 58 of 68

Mixed systems revisited

Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 59 of 68

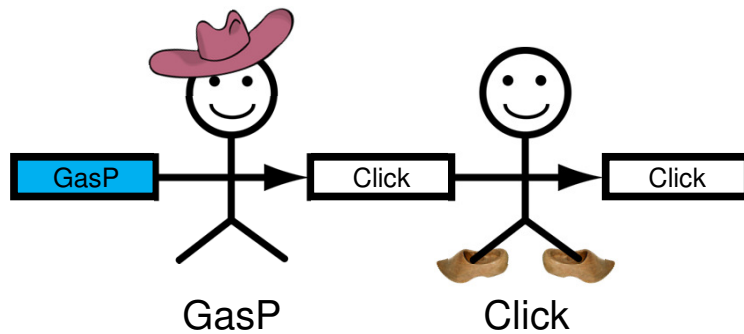
Mixed systems with full-empty interfaces



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 60 of 68

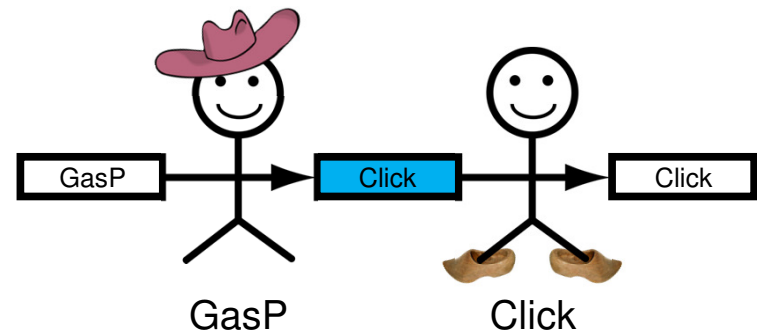
Mixed systems with full-empty interfaces



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 61 of 68

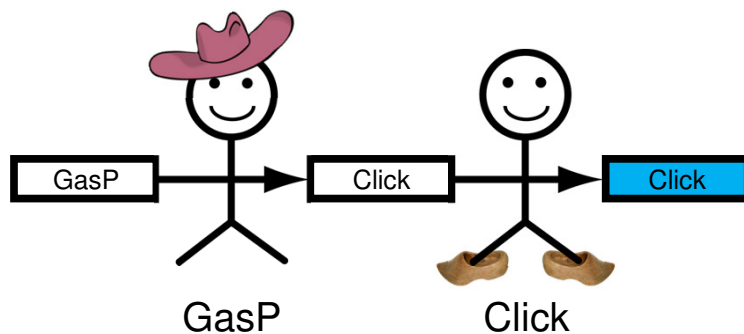
Mixed systems with full-empty interfaces



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 62 of 68

Mixed systems with full-empty interfaces



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 63 of 68

Full-empty interfaces: pros and cons

- Cons
 - Revisit existing design methods and CAD tools
- Pros
 - Translation-free communication between different self-timed circuit families
 - Simplifies collaboration and design re-use



Understanding Self-Timed Circuits — 3. Building Blocks and Protocols

slide 64 of 68

Research opportunity: ARC internship or MSc or PhD thesis



Translation-free communication for all
Explore the limits and opportunities of FULL-EMPTY interfaces for circuit families that extend the geographic reach and herald collaborative design

Outline

- Intermezzo: Why asynchronous?
- Building blocks
 - Links and joints
 - Action and the role of full and empty
 - Systems of building blocks
- Building blocks with handshake interfaces
 - Handshake protocols
 - GasP and Click FIFO
 - Pros and Cons of handshake interfaces
- Building blocks with full-empty interfaces
 - GasP and Click revisited
 - Mixed GasP and Click FIFO
 - Pros and Cons of full-empty interfaces
- Summary and Conclusion

Summary and Conclusion

- Communication delay matters!
 - Modern chips are distributed systems — so design them as such!
 - Self-timed circuits are distributed and scale over space
- Interfaces matter!
 - Design them for collaboration and re-use
Marly Roncken, Swetha Mettala Gilla, Hoon Park, Navaneeth Jamadagni, Chris Cowan, and Ivan Sutherland, *Naturalized Communication and Testing*, ASYNC 2015, pages 77-84, 2015 (presentation on ASYNC 2015 web site)
 - Think about full and empty

The importance of full and empty

- Key in asynchronous or self-timed design
 - full means data are valid
 - empty means there is space for new data
- Avoid time and power hungry polling
 - You've got mail when the mailbox flag is up
- Avoid time and power hungry memory clearing
 - Instead of clearing all register bits, declare the register empty
- Reduce the search time for a parking space in everyday life



Understanding Self-Timed Circuits

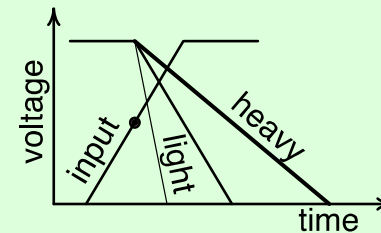
4. Measuring Performance

Marly Roncken and Ivan Sutherland
Asynchronous Research Center (ARC)
Maseeh College of Engineering and Computer Science
Portland State University
January-February 2016

slide 1

Delay in CMOS logic gates

- Turn on a transistor
- Slope of output voltage
- Depends on the load

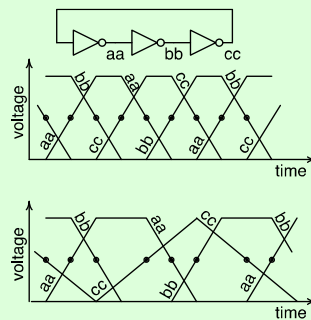


February, 2016

Slide 2

Ring oscillator

- Odd number of stages (3)
- All same size
Any size
- If different sizes
some may fail
to reach the rails



February, 2016

Slide 3

Conditions for full swing

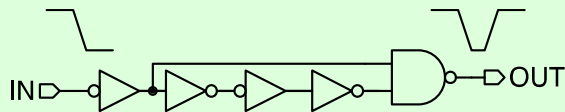
- For three stages
 - > Delays must match within 30%
- For five stages
 - > Delays must match within 300%
- We use at least 5 stages

February, 2016

Slide 4

Conditions for pulse

- Two edges with odd separation
- Need at least 3 gates between
- Better to have 5 gates

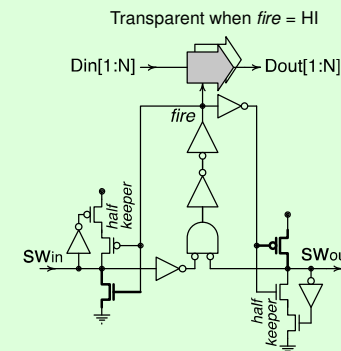


February, 2016

Slide 5

Conditions for full swing

- GasP has 2 rings of 5 stages each



February, 2016

Slide 6

Self-timed pipelines

- First-In-First-Out (FIFO) buffer
- Number of entries is variable
 - > Can be EMPTY; zero entries
 - > Can be FULL; one entry every stage
 - > Or any amount in between
- Can put in - unless FULL
- Can take out - unless EMPTY

February, 2016

Slide 7

Self-timed pipelines (cont'd)

- Data Items move toward output
 - > Remaining in sequence
 - > Data items can't overtake others
- Spaces move toward input
 - > By exchange with data items
- Analogies:
 - > one lane road
 - > queue of people at bus stop

February, 2016

Slide 8

NOT a ring buffer FIFO

- A section of RAM can be a FIFO
- Read and write pointers
- Pointer arithmetic computes FULL
- Each cycle *either* reads or writes
- Contrast with self-timed FIFO reading and writing concurrently.

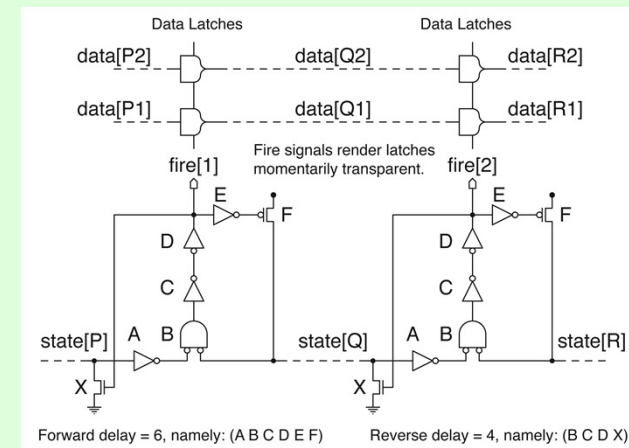
Self-timed pipelines (cont'd)

- How fast do data move?
 - > Depends on the circuit
- How fast do spaces move?
 - > Depends on the circuit
- “Canopy Graph” shows speed
- Introduced by Ted Williams (PhD 1991)
- Named by Gill and Singh (Gill's PhD 2010)

Canopy graphs

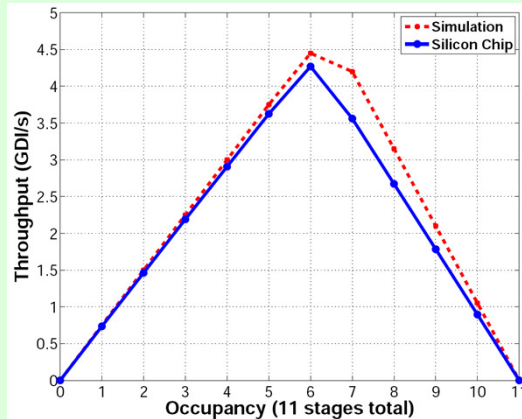
- Throughput versus occupancy
 - > throughput in items/time: GDI/s
 - > occupancy in items/stage from 0 to 1
- No occupancy = no throughput
- Full occupancy = no throughput
- Occupancy for max throughput
- Calculated for a ring

GasP circuit diagram (HI = FULL)



Throughput vs Occupancy (90nm)

Test ring has 11 GasP stages

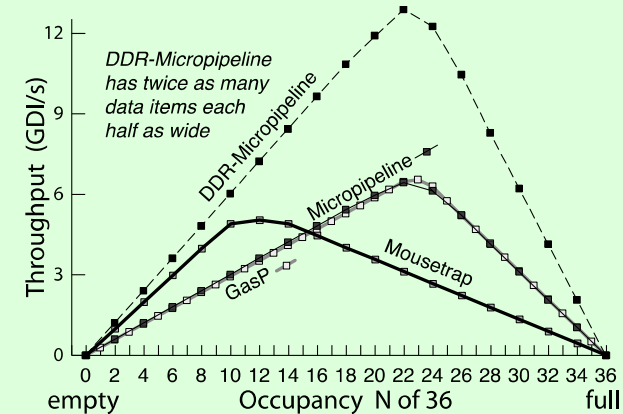


February, 2016

Slide 13

Throughput vs Occupancy (40nm)

Simulated rings of 36 stages

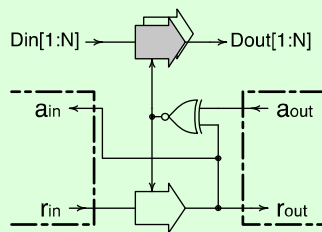


February, 2016

Slide 14

Mousetrap (2001)

- Normally transparent
- Two wires: request & acknowledge
 - FULL when they differ
 - Latch opaque when FULL

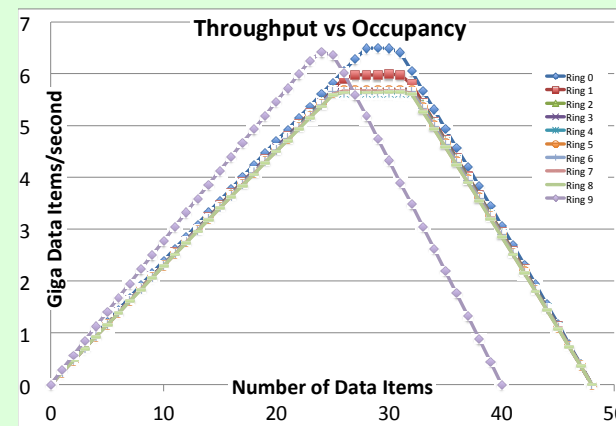


February, 2016

Slide 15

Throughput vs Occupancy (40nm)

Canopy graphs from Weaver rings



February, 2016

Slide 16

Drafting (a small effect)

- Data items form groups
 - > X X X X
 - > The first one goes slower
 - > The rest catch up
- Why?
 - > Wires fully charged for first
 - > Not yet fully charged for others

Drafting (cont'd)

- We know by many stops
 - > Where were the tokens at stop?
 - > How many got past (statistically)?
- Measured only in rings
 - > After passing billions of stages
 - > Now doing experiments to observe onset of drafting

Discussion

Understanding Self-Timed Circuits

5. Timing Validation

Marly Roncken and Ivan Sutherland
 Asynchronous Research Center (ARC)
 Maseeh College of Engineering and Computer Science
 Portland State University
 January-February 2016

slide 1 of 8

Asynchronous design: reminder (1/4)

- We design and study hardware
 - distributed over self-timed components + communication protocols
- where
 - Inside a component it can be as chaotic as a kindergarten playground which is fine, because components are small enough to control events
 - Between components, the protocols are as orderly as a "crocodile" which guarantees that the communication is correct



Understanding Self-Timed Circuits — 5. Timing Validation

slide 2 of 8

Asynchronous design: reminder (2/4)

- The playground system works, provided **local supervisors**
 - oversee the playground to avoid accidents
 - ensure children use the "crocodile" between playgrounds



Understanding Self-Timed Circuits — 5. Timing Validation

slide 3 of 8

Asynchronous design: reminder (3/4)

- Timing validation provides the **local supervisors**
- Analogy
 - playground : self-timed component
 - crocodile : channel connection with communication protocol
 - system : collection of components and communication channels
 - accident : wrong protocol
 - supervisors : event ordering constraints, a.k.a. **relative timing constraints**



Understanding Self-Timed Circuits — 5. Timing Validation

slide 4 of 8

Asynchronous design: reminder (4/4)

- Timing validation provides the local supervisors
- Analogy
 - playground : self-timed component
 - crocodile : communication channel with protocol
 - system : collection of components and communication channels
 - accident : wrong protocol
 - supervisors : event ordering constraints, a.k.a. relative timing constraints



Understanding Self-Timed Circuits — 5. Timing Validation

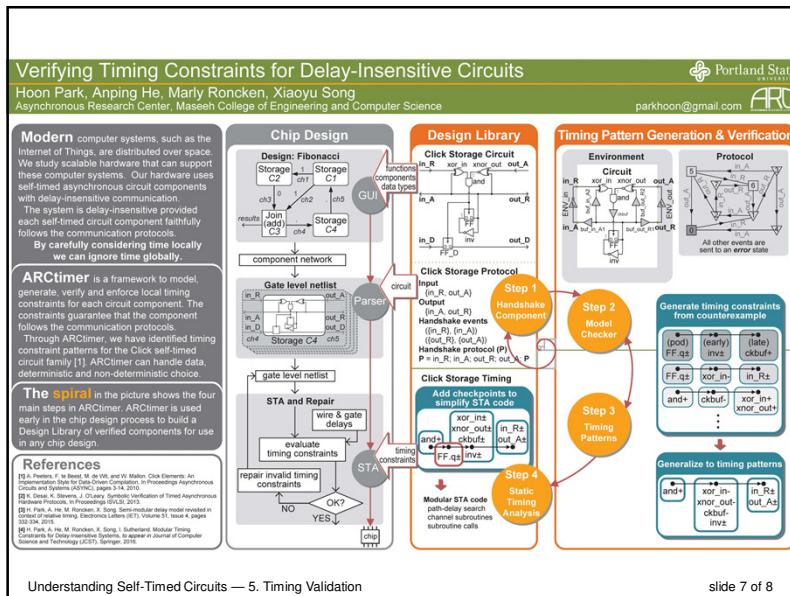
slide 5 of 8

Summary and Conclusion

- By carefully considering time locally, we can ignore time globally
- Timing constraint generation can be done in a modular way and in advance of chip design
 - as part of building a library of verified components
 - that are used again and again, for each and every chip design
 - Hoon Park, Anping He, Marly Roncken, Xiaoyu Song, and Ivan Sutherland, *Modular Timing Constraints for Delay-Insensitive Systems*, JCSST, Springer China, 31(1):77-106, 2016

Understanding Self-Timed Circuits — 5. Timing Validation

slide 6 of 8



Understanding Self-Timed Circuits — 5. Timing Validation

slide 7 of 8

Verifying Timing Constraints for Delay-Insensitive Circuits

Hoon Park, Anping He, Marly Roncken, Xiaoyu Song
Asynchronous Research Center, Maseeh College of Engineering and Computer Science

Modern computer systems, such as the Internet of Things, are distributed over space. We study scalable hardware that can support these computer systems. Our hardware uses self-timed asynchronous circuit components with delay-insensitive communication.

The system is delay-insensitive provided each self-timed circuit component faithfully follows the communication protocols.

By carefully considering time locally we can ignore time globally.

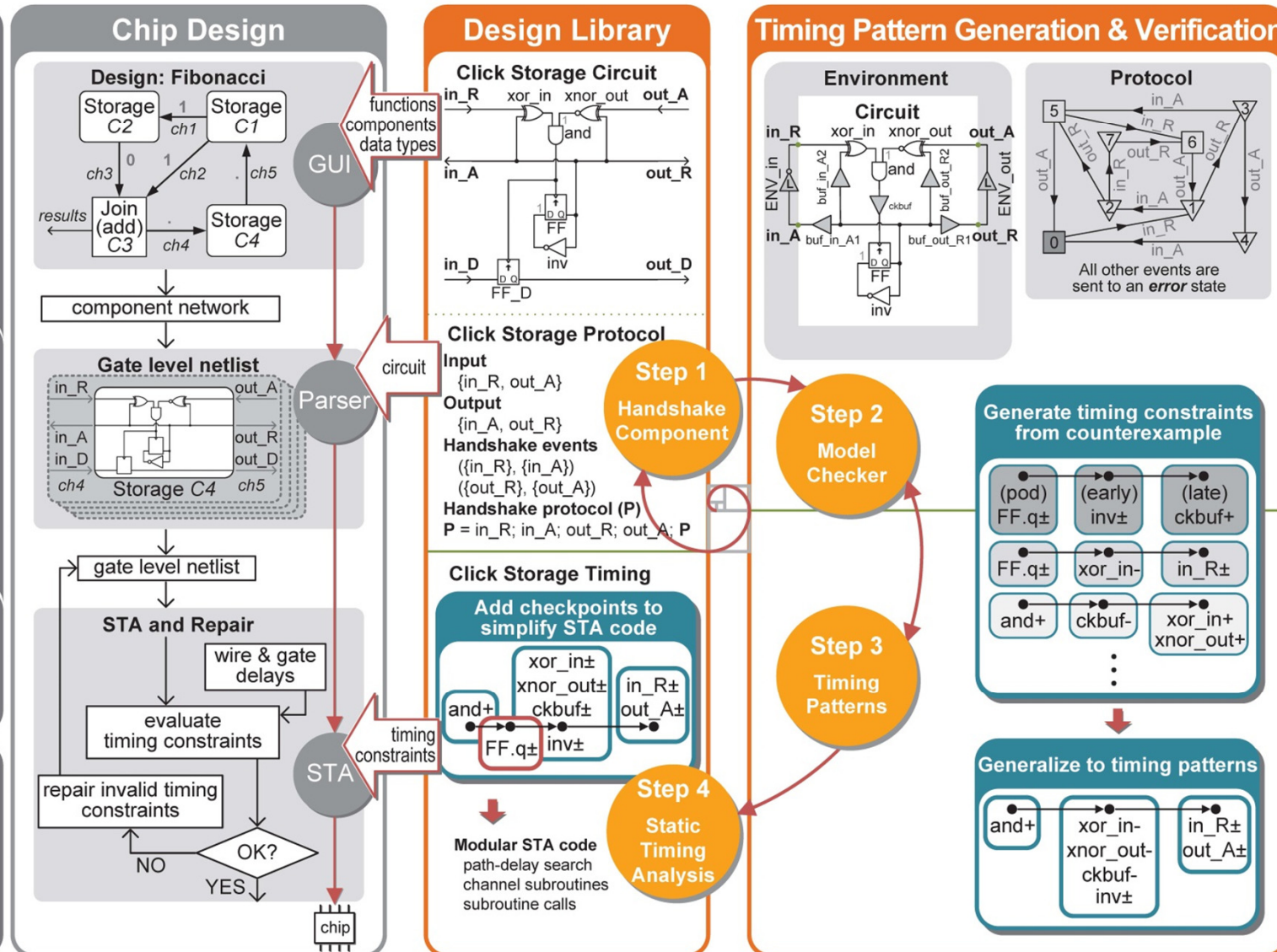
ARCTimer is a framework to model, generate, verify and enforce local timing constraints for each circuit component. The constraints guarantee that the component follows the communication protocols.

Through ARCTimer, we have identified timing constraint patterns for the Click self-timed circuit family [1]. ARCTimer can handle data, deterministic and non-deterministic choice.

The **spiral** in the picture shows the four main steps in ARCTimer. ARCTimer is used early in the chip design process to build a Design Library of verified components for use in any chip design.

References

- [1] A. Peeters, F. te Beest, M. de Wit, and W. Mallon. Click Elements: An Implementation Style for Data-Driven Compilation, In Proceedings Asynchronous Circuits and Systems (ASYNC), pages 3-14, 2010.
- [2] K. Desai, K. Stevens, J. O'Leary. Symbolic Verification of Timed Asynchronous Hardware Protocols, In Proceedings ISVLSI, 2013.
- [3] H. Park, A. He, M. Roncken, X. Song. Semi-modular delay model revisited in context of relative timing, Electronics Letters (JET), Volume 51, Issue 4, pages 332-334, 2015.
- [4] H. Park, A. He, M. Roncken, X. Song, I. Sutherland. Modular Timing Constraints for Delay-Insensitive Systems, to appear in Journal of Computer Science and Technology (JCST), Springer, 2016.



Understanding Self-Timed Circuits

6. Arbitration: Who wins?

Marly Roncken and Ivan Sutherland

Asynchronous Research Center (ARC)
Maseeh College of Engineering and Computer Science
Portland State University
January-February 2016

slide 1

Arbitration

Asynchronous Research Center

Outline

- Quantize a continuous variable
- Time is the variable
- What happened first
- Exactly the same time?
- Arbiter circuit
- Decides, but may take time

February, 2016

Slide 2

Arbitration

Asynchronous Research Center

Mutual exclusion

- Two events at “same” time
 - > which choice doesn't matter
 - > but choice must be clean
- Flip-flop can hang metastable
 - > exit is Poisson distributed
 - > may take a long time, but rarely will
- Asynchronous system can wait

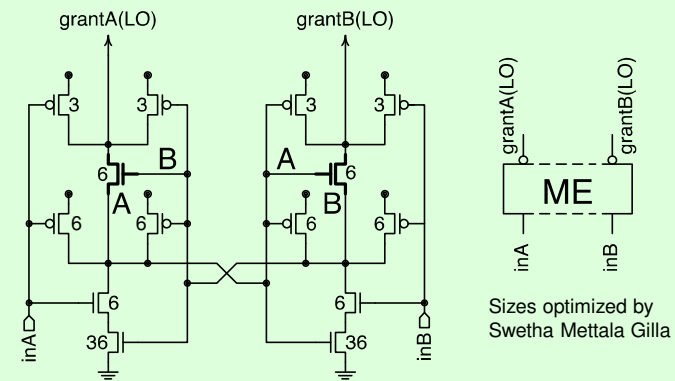
February, 2016

Slide 3

Arbitration

Asynchronous Research Center

Mutual exclusion (Seitz)

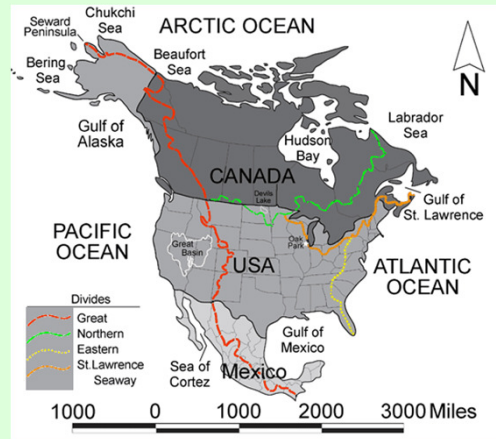


Sizes optimized by
Swetha Mettala Gilla

February, 2016

Slide 4

Continental divide



February, 2016

Slide 5

Continental divide



February, 2016

Slide 6

Continental divide



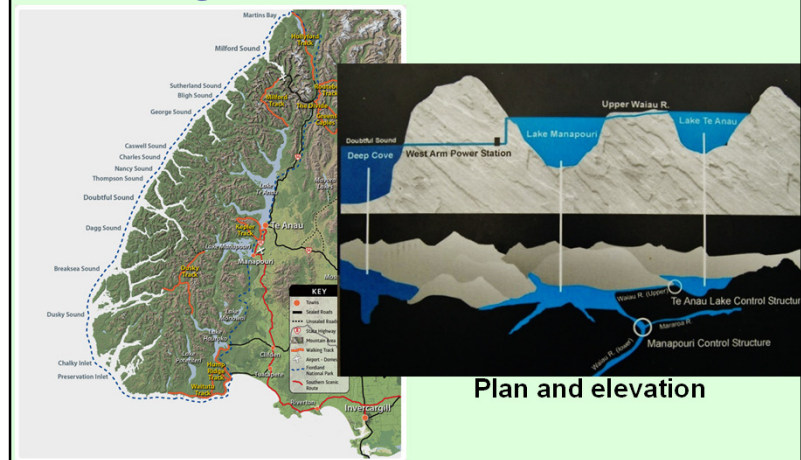
View east

View west

February, 2016

Slide 7

Moving the Continental divide



Plan and elevation

February, 2016

Slide 8

Manapouri powerplant, NZ

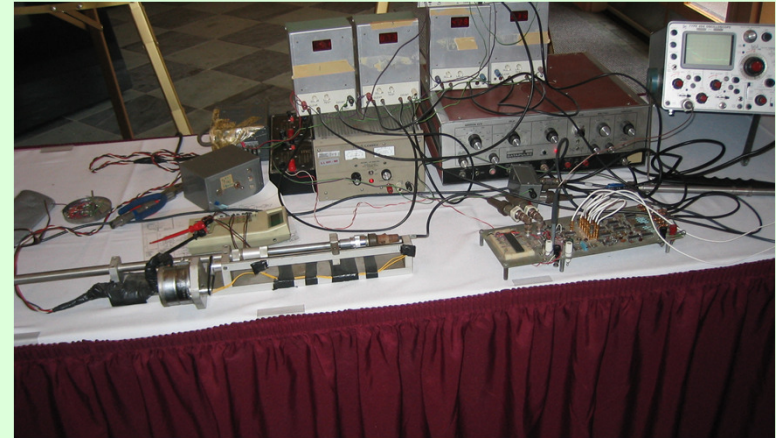


February, 2016

Slide 9

Metastability demonstration

(Chaney, St. Louis)

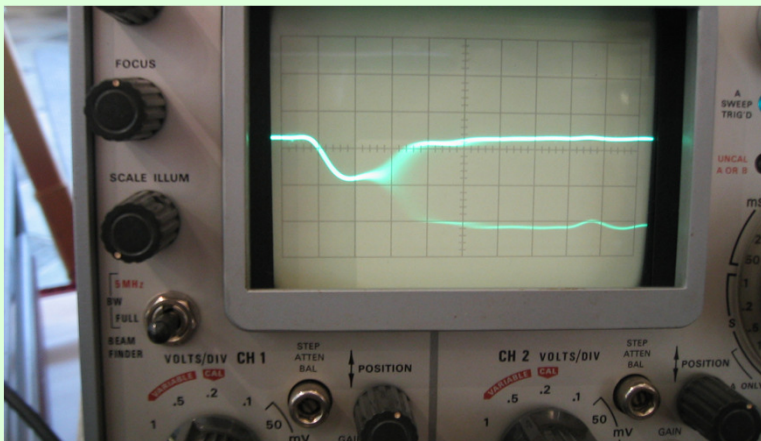


February, 2016

Slide 10

Scope trace

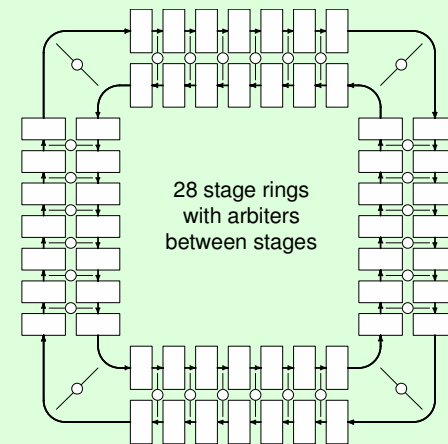
(Chaney, St. Louis)



February, 2016

Slide 11

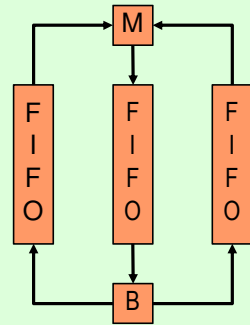
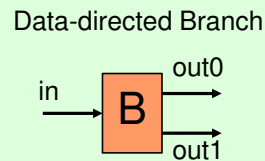
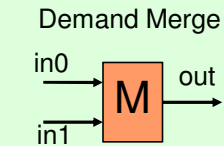
Zeke to test arbiters (1998)



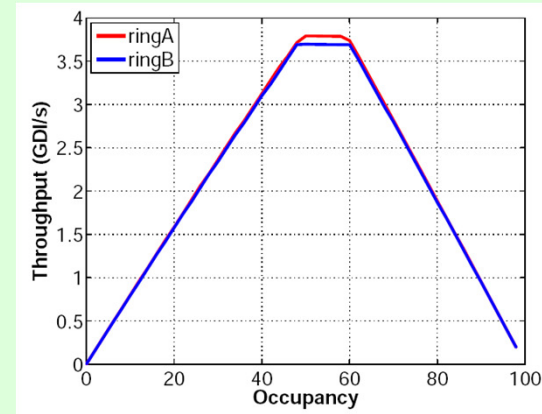
February, 2016

Slide 12

Infinity test (2008)



Infinity: Throughput vs Occupancy



Priority – a clocked idea

- Two people in an elevator
- One is “senior” and goes first
- Door opens:
 - > Senior leaves
 - > Junior leaves
- Senior has priority

Priority in a clockless world

- Two people approach a door
- One is “senior” with “priority”
- Senior arrives before junior
 - > Senior goes through first.
- Junior arrives before senior
 - > By two seconds?
 - > By two minutes?

Priority

- Not meaningful without clock
- Is equivalent to delay

“Fair” arbitration

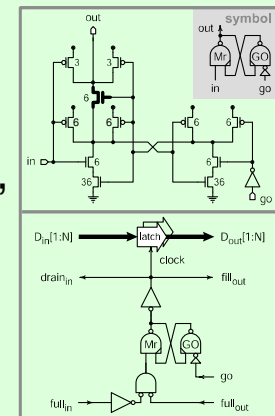
- What is “fair”
- First come first served
- If I arrive while busy and I wait will I be served next?
- Greedy requests get alternate service
- More than two users?

Stopping self-timed systems

- Clean stop requires arbitration
 - > Stop in mid action OR
 - > Finish the action, then stop
- Without arbitration, runt pulses
 - > Give chance of data error or
 - > Loss of whole data item

MrGO

- Half an arbiter
- To stop cleanly
- A “proper stopper”
- *Shall I stop now or complete this action?*



Discussion

Understanding Self-Timed Circuits

7. Initialization, Test, and Debug

Marly Roncken and Ivan Sutherland

Asynchronous Research Center (ARC)
Maseeh College of Engineering and Computer Science
Portland State University
January-February 2016

slide 1 of 65

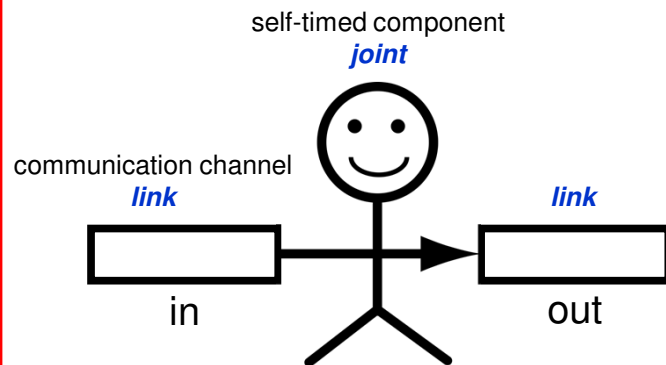
Outline

- **Reminder: Building blocks**
- What's so special about testing circuits?
- Test space and control — in 3D
 - Scan state control
 - MrGO action control
- Test examples using scan + MrGO
 - **Multi-step:**
 - Testing a counter at speed, for one data item
 - Testing a counter at speed, for a burst of data
 - Using the counter to characterize throughput
 - **Single-step:**
 - Testing stuck-at faults in building blocks
- Summary and Conclusion

Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 2 of 65

Building blocks: reminder



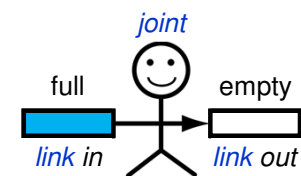
Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 3 of 65

Building blocks: action reminder

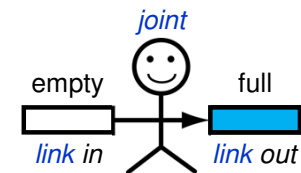
WHEN to act:

in is full
and
out is empty



WHAT to do:

- copy data
- drain *in*
- fill *out*



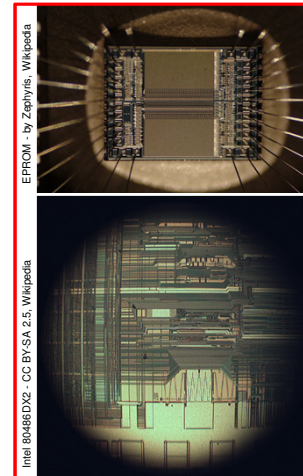
Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 4 of 65

Outline

- Reminder: Building blocks
- What's so special about testing circuits?
- Test space and control — in 3D
 - Scan state control
 - MrGO action control
- Test examples using scan + MrGO
 - Multi-step:
 - Testing a counter at speed, for one data item
 - Testing a counter at speed, for a burst of data
 - Using the counter to characterize throughput
 - Single-step:
 - Testing stuck-at faults in building blocks
- Summary and Conclusion

so MANY signals - so FEW pins



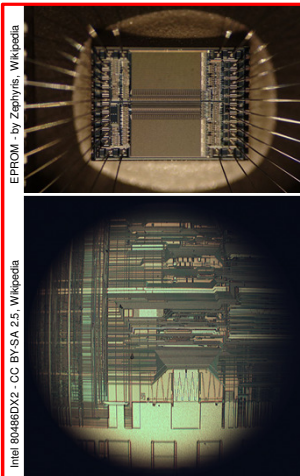
- Isn't this similar to testing software?
 - so many lines - so few exports
- for which the answer is:
 - use an **interactive code debugger**
 - to set break points, single-step code, etc.

```
//Click storage old style: with handshake interfaces
storage { in : chan[] , out : chan[] }

/*
// data path
$[ set count 0 ]$
$[ foreach (x) ($[range $#in] )$ {
  $[ foreach (y) ($[range $#in$[x]] )$ {
    ff_ff$count { .d(in$[x] [ $[y] ] ) , .q(internal_$count) }
    .op(clock) , .RESETn(RESETn);
    $[ set count $[expr $#count + 1 ]$ ]$
  } ]$
} ]$
$[foreach (x) ($[range $#out] )$ {
  $[ foreach (y) ($[range $[expr 0 $#]
    foreach (x) ($[range $#in] )$ { + $#in$[x] ) ]$ ]$ {
    out$[x]($[y]) = ASSIGN(internal_$[y]);
  } ]$
} ]$
$[foreach (x) ($[range $#in] )$ {
  delay_0 in$[x]_delay { .A(in$[x]_R) , .S(delayed_in$[x]_R);
```

Small code fragment from 50000 code lines in the ARCwelder silicon compiler for self-timed circuits

so MANY signals - so FEW pins



- Isn't this similar to testing software?
 - so many lines - so few exports
- for which the answer is:
 - use an **interactive code debugger**
 - to set break points, single-step code, etc.
- SO
- Why not test the pre-silicon code ...
 - Please do
- ... instead of testing the silicon version?
 - It's not enough, because
 - the code translation may introduce bugs
 - and manufacturing may introduce defects
- Wanted:
 - **Silicon equivalent** of a code debugger

Outline

- Reminder: Building blocks
- What's so special about testing circuits?
- Test space and control — in 3D
 - Scan state control
 - MrGO action control
- Test examples using scan + MrGO
 - Multi-step:
 - Testing a counter at speed, for one data item
 - Testing a counter at speed, for a burst of data
 - Using the counter to characterize throughput
 - Single-step:
 - Testing stuck-at faults in building blocks
- Summary and Conclusion

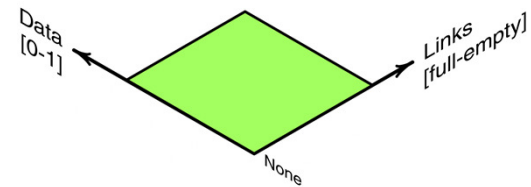
Test space and control: a 3D visualization

Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 9 of 65

Test control (1/3): none

- suffices if many internal signals are also external

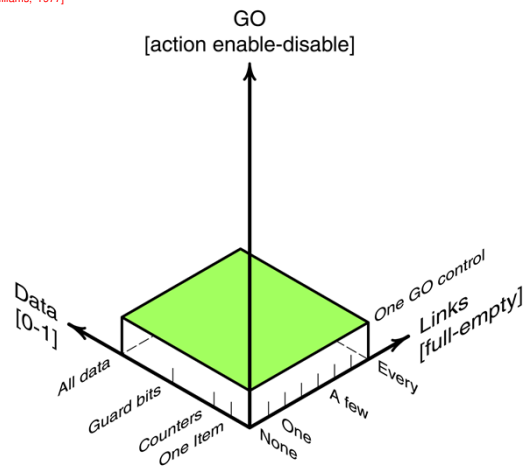


Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 10 of 65

Test control (2/3): over global state + action

- scantest (scan): initialization (+ single-step and multi-step test in clocked systems)
[Eichelberger-Williams, 1977]

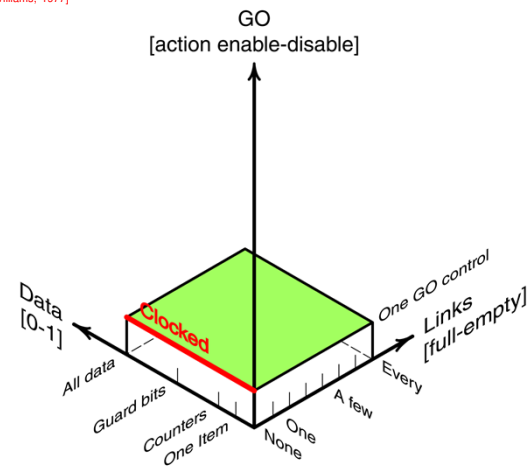


Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 11 of 65

Test control (2/3): over global state + action

- scantest (scan): initialization (+ single-step and multi-step test in clocked systems)
[Eichelberger-Williams, 1977]

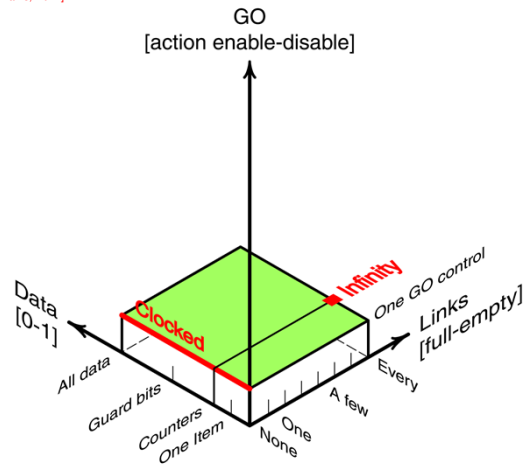


Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 12 of 65

Test control (2/3): over global state + action

- **scantest (scan):** initialization (+ single-step and multi-step test in clocked systems)
[Eichelberger-Williams, 1977]



Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 13 of 65

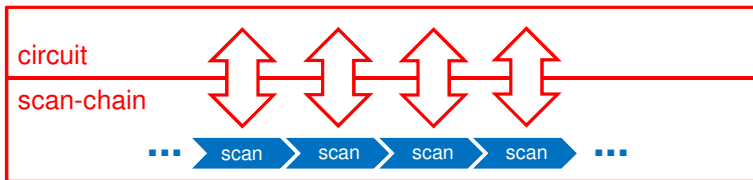
SCAN: state control

Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 14 of 65

Scan design

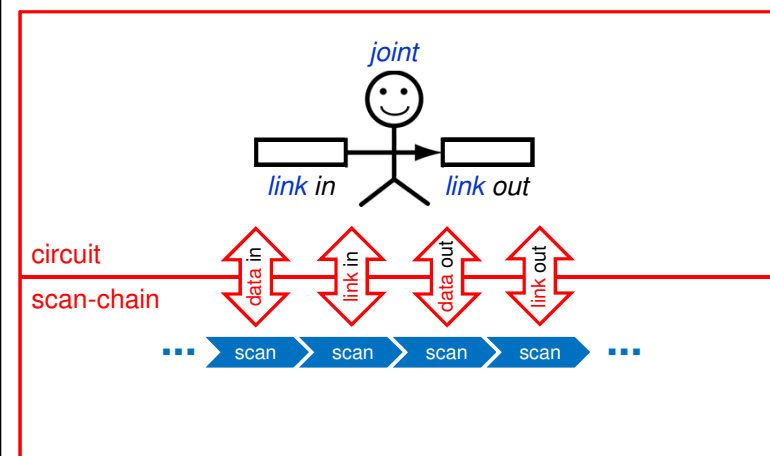
- **Scan-chain**
 - Can store + move (shift) data
 - Entry and exit are external
 - Can write data from the scan chain into internal circuit signals
 - Can read internal circuit signals and store their values in the scan chain
- **Scan-control**
 - For scan operations: **shift**, **write**, **read**, **enable-disable circuit action**
 - Can be self-timed or clocked; we use **clocked**, because
 - it doesn't matter: systems operate independent of how they're tested
 - IEEE standards are available



Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 15 of 65

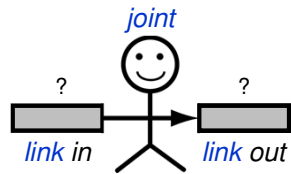
Scan design



Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 16 of 65

Scan operations: shift in stimuli (1/5)



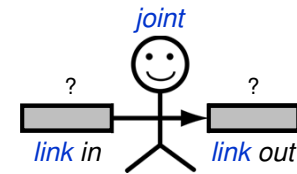
circuit

scan-chain

1 full 0 empty > scan > scan > scan > scan > ...

serial shift

Scan operations: shift in stimuli (2/5)



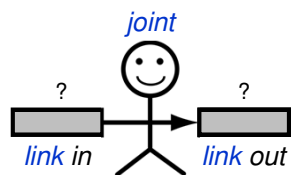
circuit

scan-chain

1 full 0 > empty > scan > scan > scan > ...

serial shift

Scan operations: shift in stimuli (3/5)



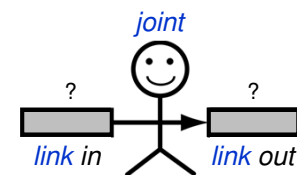
circuit

scan-chain

1 full > 0 > empty > scan > scan > ...

serial shift

Scan operations: shift in stimuli (4/5)



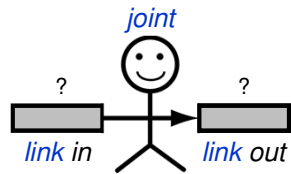
circuit

scan-chain

1 > full > 0 > empty > scan > ...

serial shift

Scan operations: shift in stimuli (5/5)



circuit

scan-chain

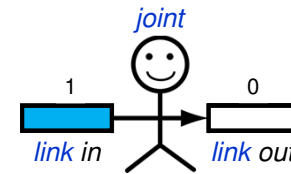


serial shift

Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

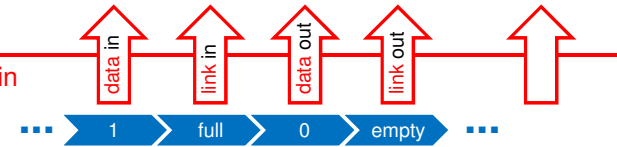
slide 21 of 65

Scan operations: write stimuli to circuit



circuit

scan-chain

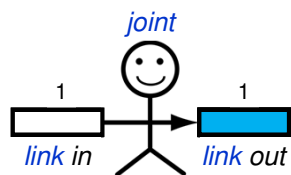


parallel write

Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 22 of 65

Scan operations: let the circuit run



circuit

scan-chain



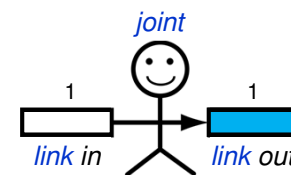
enable all
circuit actions



Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

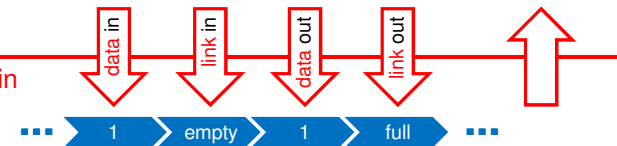
slide 23 of 65

Scan operations: read results from circuit



circuit

scan-chain

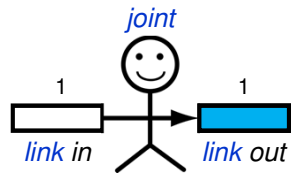


parallel read

Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 24 of 65

Scan operations: shift out results



circuit

scan-chain



serial shift

SCAN: pros and cons

Pros

- Good for circuit initialization
- Good for testing single-step circuit actions
- Good for sequential testing of multi-step circuit actions, at speed:
just "keep ticking" as many times as needed

But

- What's in a "tick" — when it's a self-timed circuit?
- Does a "tick" stop?
- If it doesn't then how can we use scan reads and writes, safely?

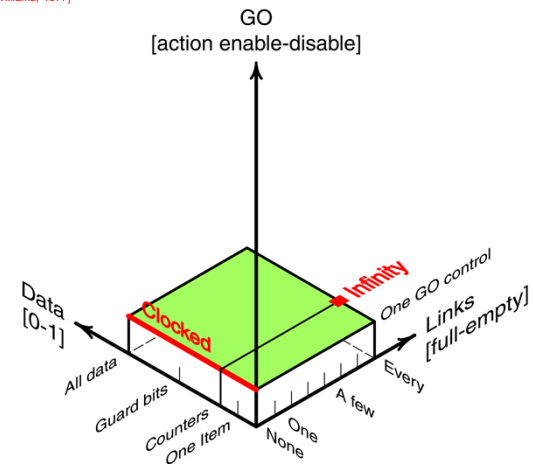
Cons

- Scan by itself isn't enough for self-timed circuits
- Example:
Handshake Solutions clocks every loop to kill self-timed action at test to regain control over initialization and single-step test (not sequential test)

What's missing?
back to 3D test space and control

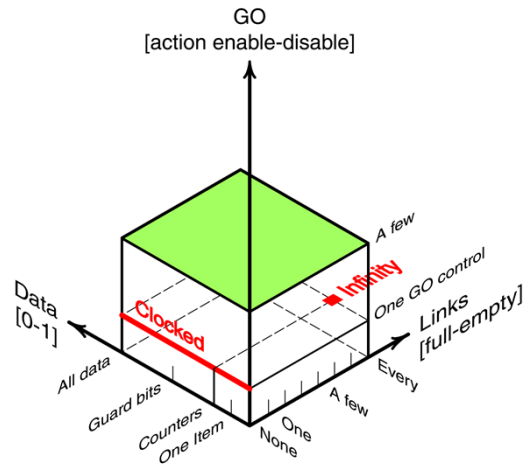
Test control (2/3): over global state + action

- scantest (scan): initialization (+ single-step and multi-step test in clocked systems)
[Eichelberger-Williams, 1977]



Test control (3/3): + distinguish local actions

- scan+GO: initialization + single-step and multi-step and at-speed test and debug

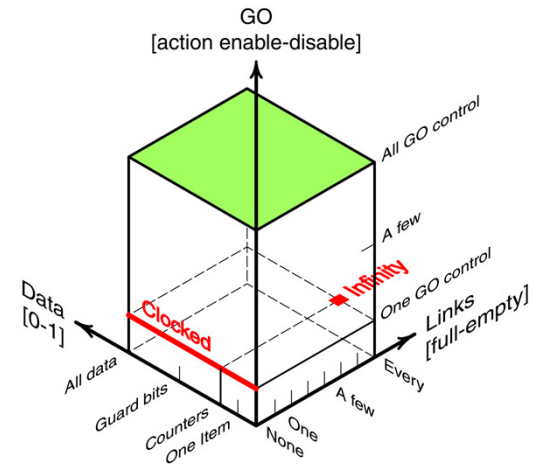


Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 29 of 65

Test control (3/3): + distinguish local actions

- scan+GO: initialization + single-step and multi-step and at-speed test and debug

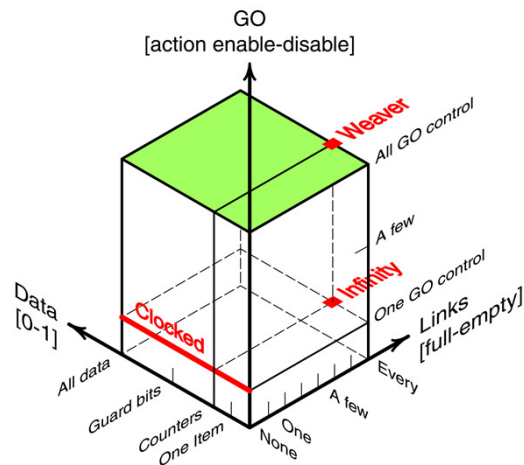


Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 30 of 65

Test control (3/3): + distinguish local actions

- scan+GO: initialization + single-step and multi-step and at-speed test and debug



Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 31 of 65

GO: (individual) local action control

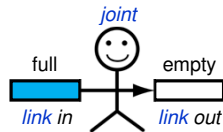
Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 32 of 65

Building blocks: action reminder

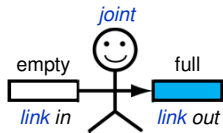
WHEN to act:

in is full
and
out is empty



WHAT to do:

- copy data
- drain *in*
- fill *out*



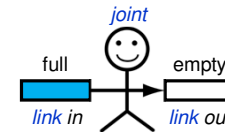
Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 33 of 65

Building blocks: action with GO control

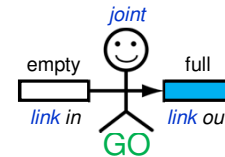
WHEN to act:

in is full
and
out is empty
and
GO



WHAT to do:

- copy data
- drain *in*
- fill *out*



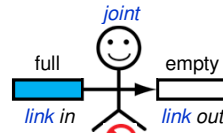
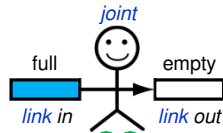
Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 34 of 65

Building blocks: action with GO control

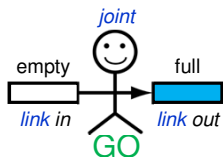
WHEN to act:

in is full
and
out is empty
and
GO



WHAT to do:

- copy data
- drain *in*
- fill *out*

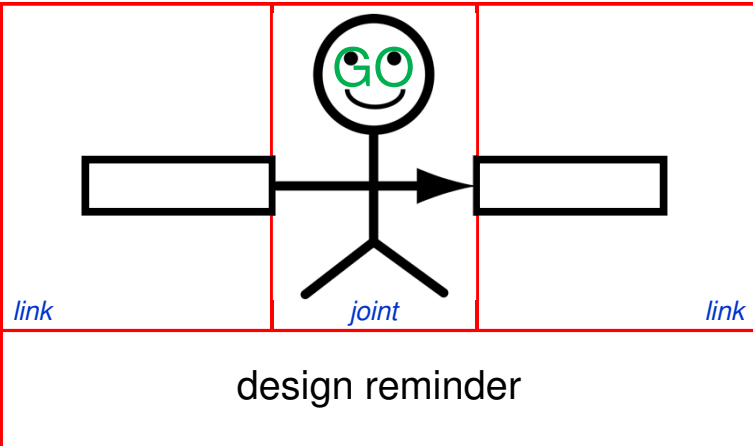


no action

Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 35 of 65

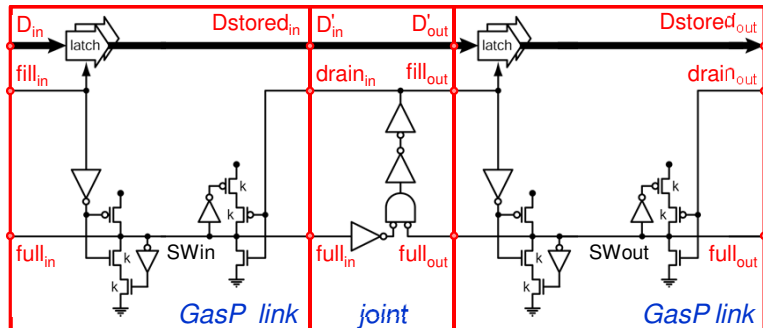
Building blocks: design with GO control



Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 36 of 65

Building blocks: design with GO control

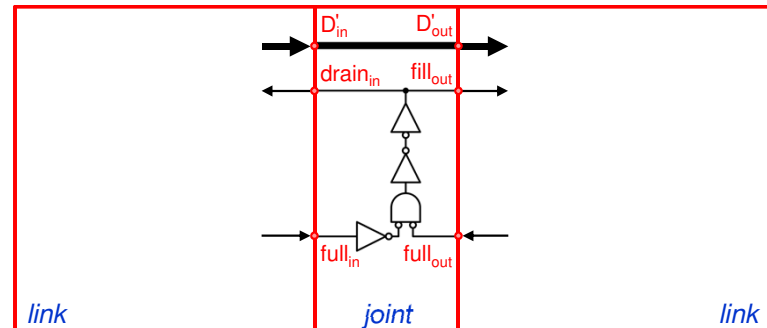


design reminder

Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 37 of 65

Building blocks: design with GO control

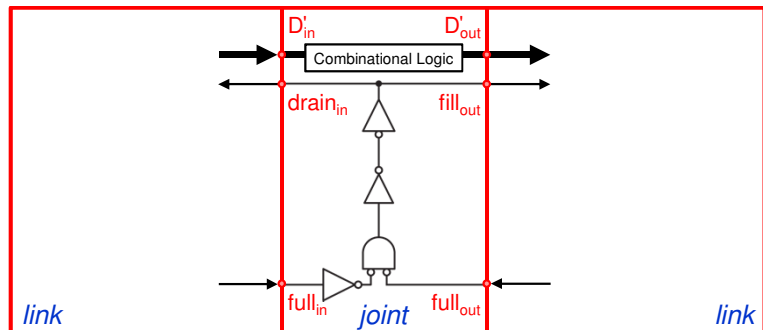


design reminder

Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 38 of 65

Building blocks: design with GO control

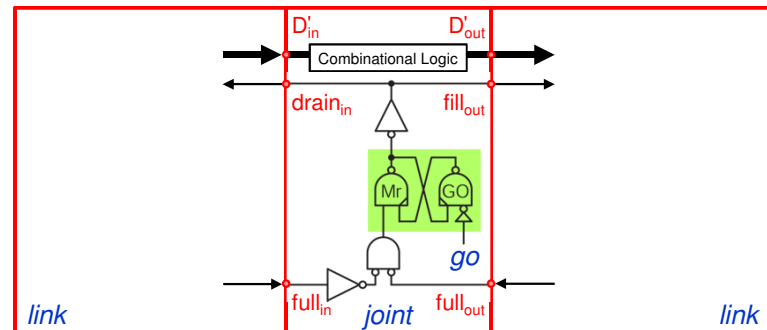


design reminder

Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 39 of 65

Building blocks: design with GO control




Solution MrGO:

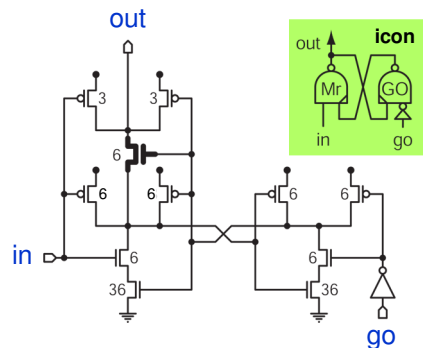
- *go* is high (GO) : run
- *go* is low () : stop and freeze
- arbiter for safe stop : "proper stopper"
- scan chain delivers *go* signals

Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 40 of 65

MrGO: dedicated action control

- go is high (GO) – start in to out
- go is low () – stop or freeze in to out
- arbiter for safe stop – "proper stopper"
- scan chain delivers go signals



Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

(backup) slide 41 of 65

Outline

- Reminder: Building blocks
- What's so special about testing circuits?
- Test space and control — in 3D
 - Scan state control
 - MrGO action control
- Test examples using scan + MrGO
 - Multi-step:
 - Testing a counter at speed, for one data item
 - Testing a counter at speed, for a burst of data
 - Using the counter to characterize throughput
 - Single-step:
 - Testing stuck-at faults in building blocks
- Summary and Conclusion

Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 42 of 65

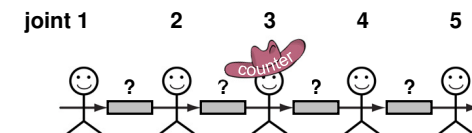
Sequential at-speed test and debug using scan + MrGO

Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 43 of 65

Testing a counter at speed

INITIALIZE



RUN

EVALUATE

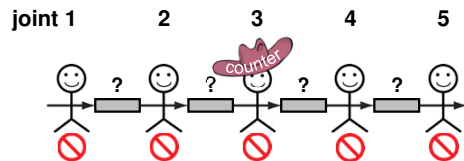
Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 44 of 65

Testing a counter at speed

INITIALIZE

1. freeze all joints



RUN

EVALUATE

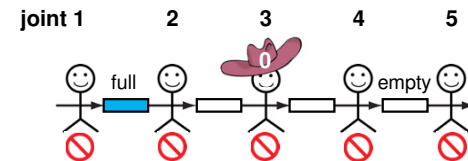
Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 45 of 65

Testing a counter at speed

INITIALIZE

1. freeze all joints
2. set state
 - full-empty links
 - counter data



RUN

EVALUATE

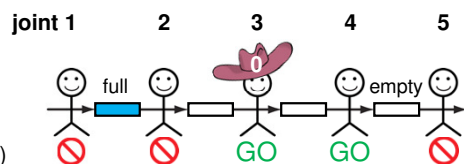
Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 46 of 65

Testing a counter at speed

INITIALIZE

1. freeze all joints
2. set state
 - full-empty links
 - counter data
3. unfreeze "runway" (3,4)



RUN

EVALUATE

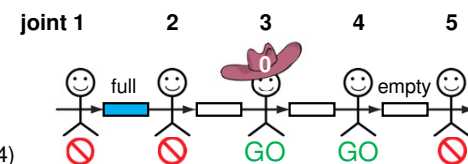
Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 47 of 65

Testing a counter at speed

INITIALIZE

1. freeze all joints
2. set state
 - full-empty links
 - counter data
3. unfreeze "runway" (3,4)



RUN

EVALUATE

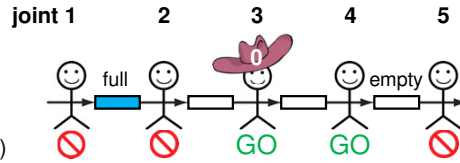
Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 48 of 65

Testing a counter at speed

INITIALIZE

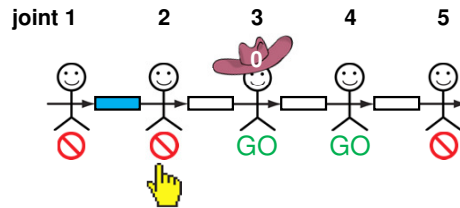
1. freeze all joints
2. set state
 - full-empty links
 - counter data
3. unfreeze "runway" (3,4)



RUN

1. unfreeze entry (2)
2. wait for action to finish

EVALUATE



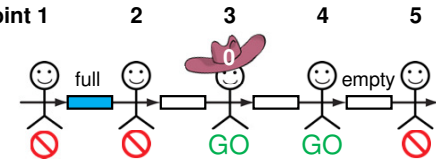
Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 49 of 65

Testing a counter at speed

INITIALIZE

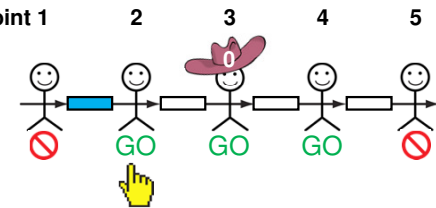
1. freeze all joints
2. set state
 - full-empty links
 - counter data
3. unfreeze "runway" (3,4)



RUN

1. unfreeze entry (2)
2. wait for action to finish

EVALUATE



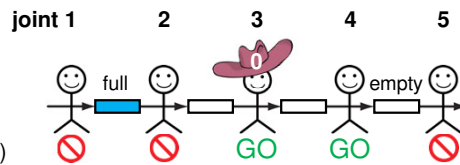
Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 50 of 65

Testing a counter at speed

INITIALIZE

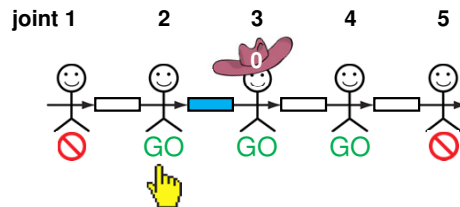
1. freeze all joints
2. set state
 - full-empty links
 - counter data
3. unfreeze "runway" (3,4)



RUN

1. unfreeze entry (2)
2. wait for action to finish

EVALUATE



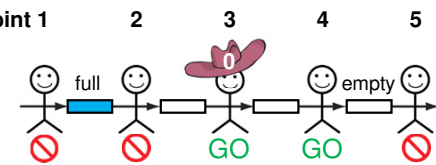
Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 51 of 65

Testing a counter at speed

INITIALIZE

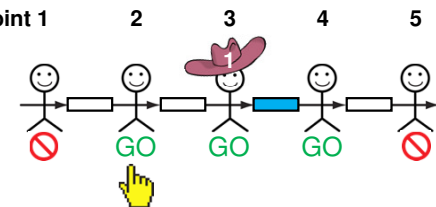
1. freeze all joints
2. set state
 - full-empty links
 - counter data
3. unfreeze "runway" (3,4)



RUN

1. unfreeze entry (2)
2. wait for action to finish

EVALUATE



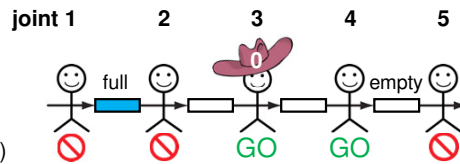
Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 52 of 65

Testing a counter at speed

INITIALIZE

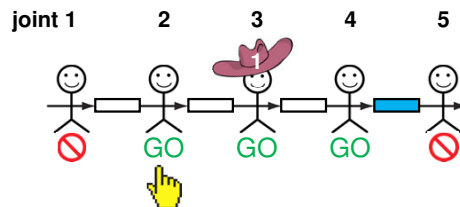
1. freeze all joints
2. set state
 - full-empty links
 - counter data
3. unfreeze "runway" (3,4)



RUN

1. unfreeze entry (2)
2. wait for action to finish

EVALUATE



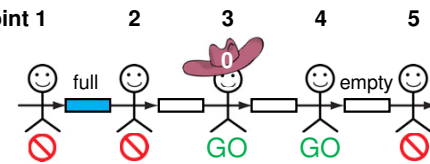
Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 53 of 65

Testing a counter at speed

INITIALIZE

1. freeze all joints
2. set state
 - full-empty links
 - counter data
3. unfreeze "runway" (3,4)

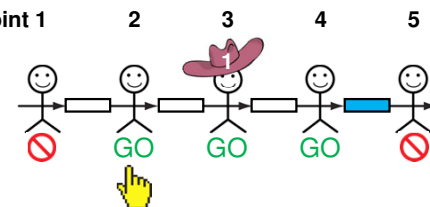


RUN

1. unfreeze entry (2)
2. wait for action to finish

EVALUATE

- read counter data



Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 54 of 65

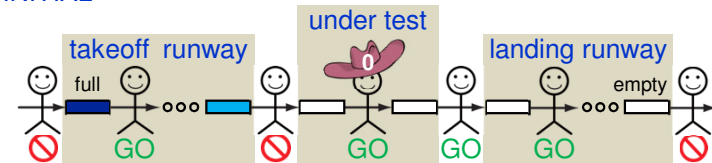
Sequential at-speed test and debug data burst

Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

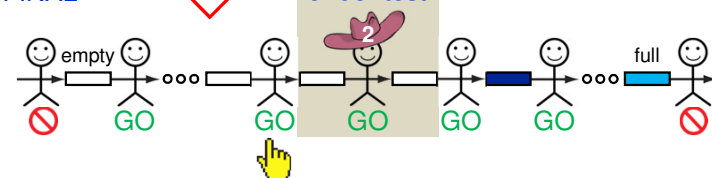
(backup) slide 55 of 65

Testing a burst of data at speed

INITIAL



FINAL



Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

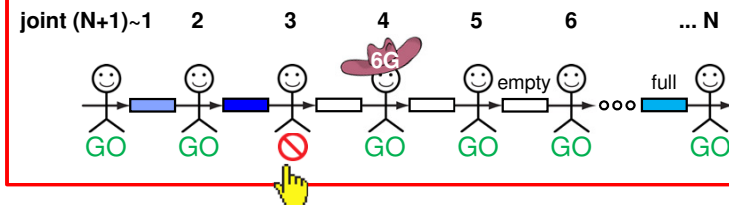
(backup) slide 56 of 65

Characterization of throughput using scan + MrGO

Performance characterization

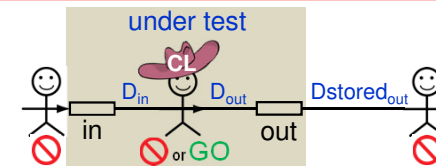
DO (ALL $i > 0$ links)
 counter=0
 run 1 second with i full links
 arbitrated stop
 read counter
 OD

FINAL for $i \sim 60\%$ links



Structural fault testing using scan + MrGO

Testing stuck-at faults



TEST control logic

DO (ALL full-empty link combos)
 freeze joint
 set $full_{in} = \text{combo}(\text{in})$
 $full_{out} = \text{combo}(\text{out})$
 evaluate if links remain unchanged
 unfreeze joint
 evaluate final link states
 OD

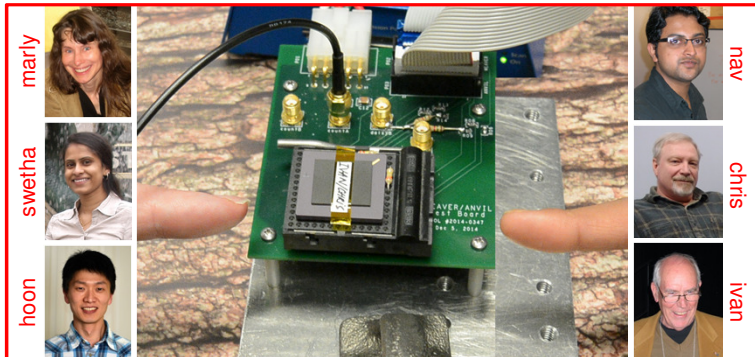
TEST datapath (normally opaque)

DO (ALL CL test inputs)
 freeze joint
 set $full_{in} = \text{TRUE}$
 $full_{out} = \text{FALSE}$
 $D_{in} = \text{test input}$
 $D_{stored_out} = \neg CL(D_{in})$
 evaluate if D_{stored_out} remains unchanged
 unfreeze joint
 evaluate if $D_{stored_out} = CL(D_{in})$
 OD

Get real!

MrGO approved

- Two working silicon experiments: Weaver and Anvil
- use building blocks with full-empty interfaces
- and MrGO + JTAG-scan for test, debug, characterization



Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 61 of 65

Outline

- Reminder: Building blocks
- What's so special about testing circuits?
- Test space and control — in 3D
 - Scan state control
 - MrGO action control
- Test examples using scan + MrGO
 - Multi-step:
 - Testing a counter at speed, for one data item
 - Testing a counter at speed, for a burst of data
 - Using the counter to characterize throughput
 - Single-step:
 - Testing stuck-at faults in building blocks

Summary and Conclusion

Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 62 of 65

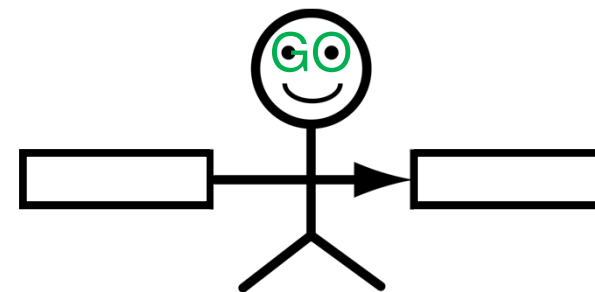
Summary and Conclusion

- Test control = state control (scan) + action control (MrGO)
- Actions matter!
 - Recognize and distinguish them
 - Control them individually for at-speed test, debug, and characterization
 - Marly Roncken, Swetha Mettala Gilla, Hoon Park, Navaneeth Jamadagni, Chris Cowan, and Ivan Sutherland, *Naturalized Communication and Testing*, ASYNC 2015, pages 77-84, 2015 (presentation on ASYNC 2015 web site)
 - Poster by Swetha on Testing with MrGO (see ASYNC 2015 web site)

Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 63 of 65

Research opportunity: ARC internship or MSc or PhD thesis



Interactive action control

Explore the limits and opportunities of dedicated action control à la MrGO for silicon test and debug and characterization of distributed VLSI systems

Understanding Self-Timed Circuits — 7. Initialization, Test, and Debug

slide 64 of 65



Testing Self-Timed Circuits with MrGO

Swetha Mettala Gilla, Marly Roncken, Ivan Sutherland, Xiaoyu Song
Asynchronous Research Center, Portland State University
(mettalag@cecs.pdx.edu)

MOTIVATION

Why self-timed?

- Self-timed circuits offer modularity
- Self-timed circuits offer energy efficiency
- Self-timed circuits offer speed

What?

- Self-timed networks of state-holding **links** (□)
- Exchange data at action-capable **joints** (⊙)

Wanted:

- A **general test method** to initialize states and control actions for:
 1. structural fault testing,
 2. at-speed testing, and
 3. debug

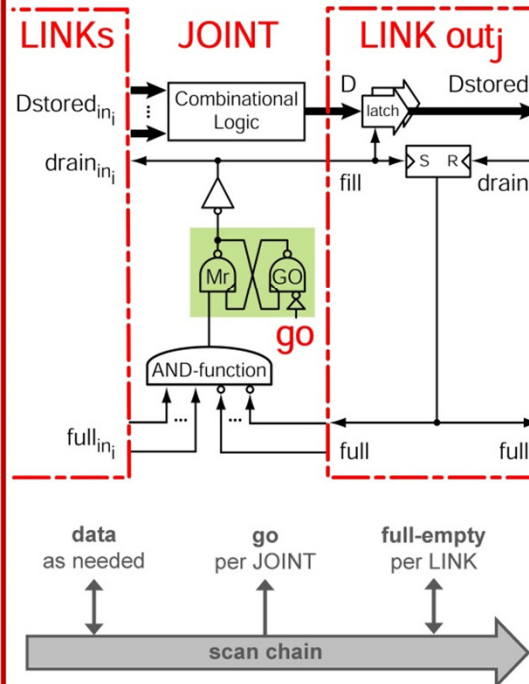
SOLUTION

- (Well-known) **scan chain** to initialize and observe link states
- (New) **MrGO** to control individual joint actions
 - go is high (GO) – run
 - go is low (⊘) – stop
 - arbiter for safe stop – "proper stopper"
 - scan chain delivers go signals

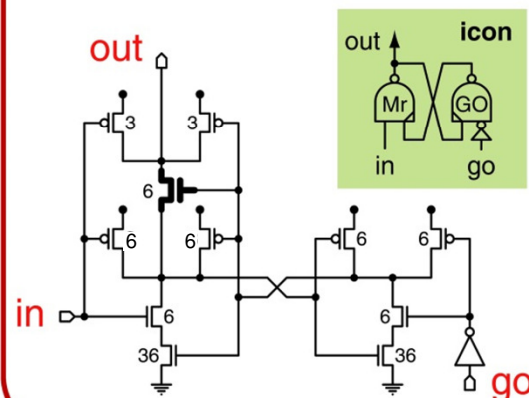
REFERENCES

- [1] M. Bushnell and V. Agrawal, "Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits," Springer, 2005.
- [2] C. Molnar, I. Jones, W. Coates, and J. Lexau, "A FIFO Ring Performance Experiment," ASYNC, pp. 279–289, 1997.
- [3] M. Roncken, "Defect-Oriented Testability for Asynchronous ICs," *Proceedings of the IEEE*, Vol. 87, No. 2, pp. 363–375, Feb. 1999.
- [4] M. Roncken, S. Mettala Gilla, H. Park, N. Jamadagni, C. Cowan, I. Sutherland, "Naturalized Communication and Testing," ASYNC 2015.

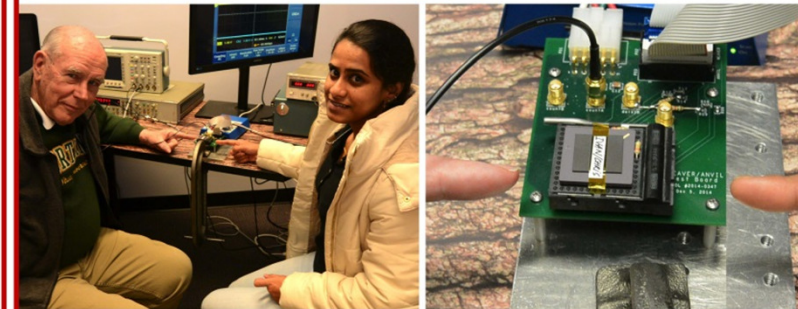
DESIGN FOR TEST



MrGO circuit



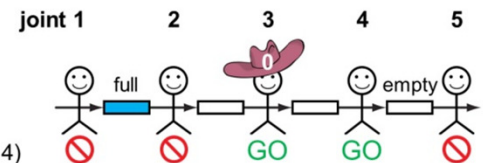
TEST EXECUTION



Example: testing a counter at speed

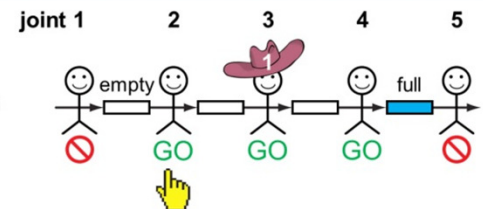
INITIALIZE

1. freeze all joints
2. set state
 - full-empty links
 - counter data
3. unfreeze "runway" (3, 4)



RUN

1. unfreeze entry (2)
2. wait for action to finish



EVALUATE

- read counter data

Supports:

1. Initialization
2. Arbitrated stop from full speed
3. Single- and multi-step operations
4. At-speed testing of sub-systems
5. Canopy graph generation
6. Testing of structural faults like stuck-at

**Is built into the latest ARC-Oracle test chip
AND IT WORKS !**

Understanding Self-Timed Circuits

8. The Weaver

Marly Roncken and Ivan Sutherland

Asynchronous Research Center (ARC)
Maseeh College of Engineering and Computer Science
Portland State University
January-February 2016

slide 1

A View of Self-Timed Systems

Asynchronous Research Center

Subject of this talk

- **Weaver**: a non-blocking 8x8 crossbar
- Built in 40 nm TSMC technology
- Less than 1 ns latency
- **3.4 Terabits/second max throughput**
Which is 6 billion data items (max)/sec/channel
times 72 bits per data item, times 8 channels
- **Uses MrGO Test and debug**

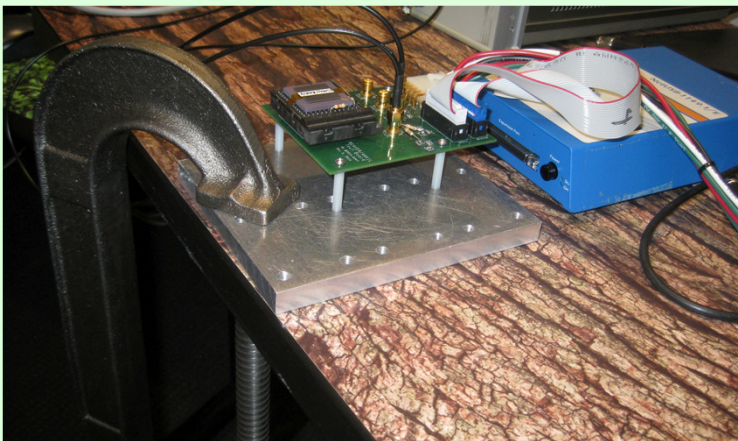
June 2015

Slide 2

A View of Self-Timed Systems

Asynchronous Research Center

Test setup



June 2015

Slide 3

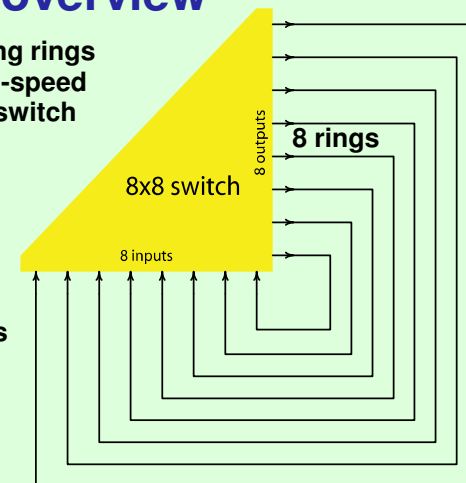
A View of Self-Timed Systems

Asynchronous Research Center

Weaver overview

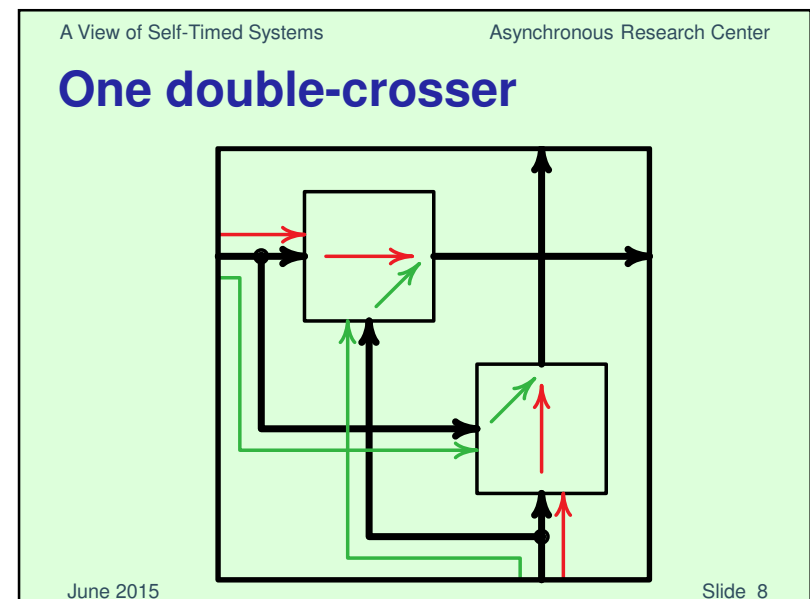
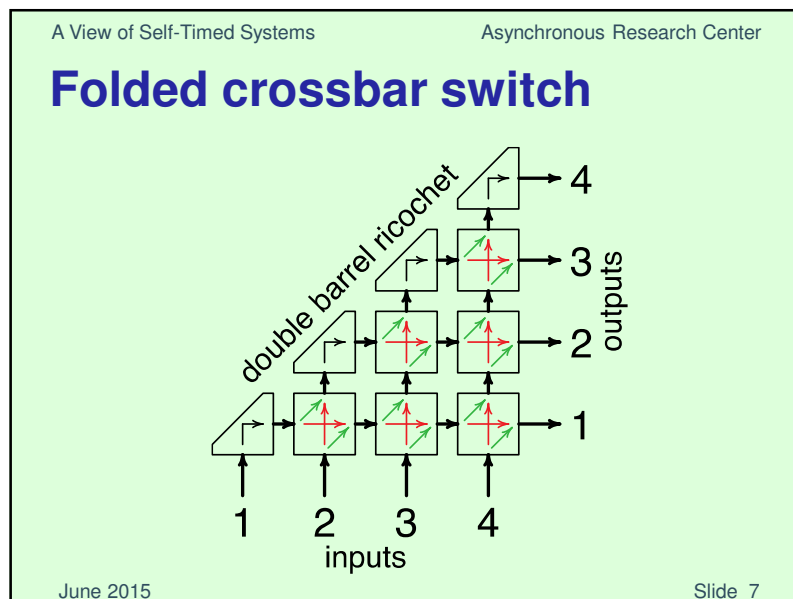
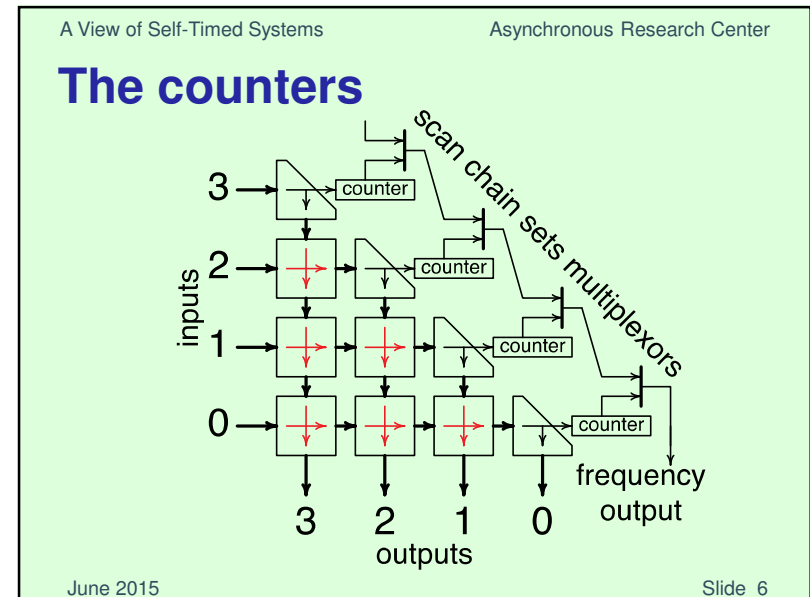
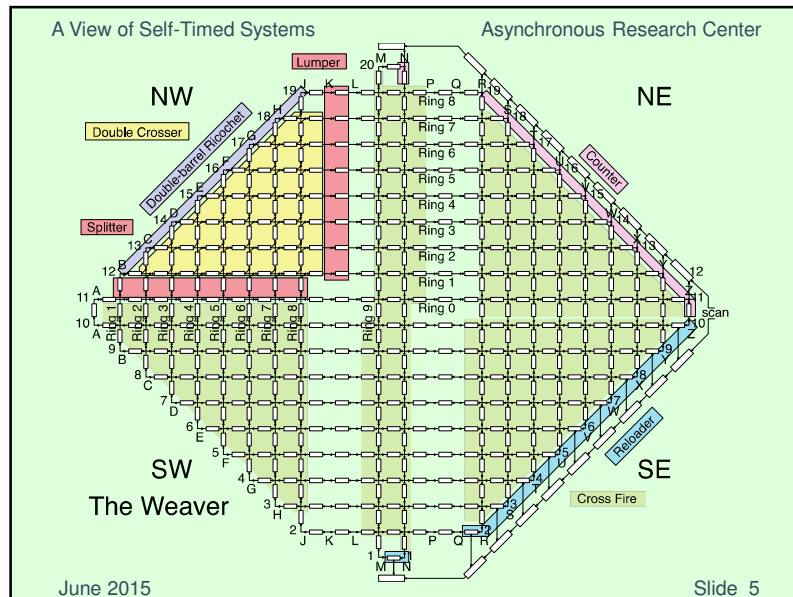
8 recirculating rings
provide high-speed
data for the switch

Not shown:
2 more rings
without
switches

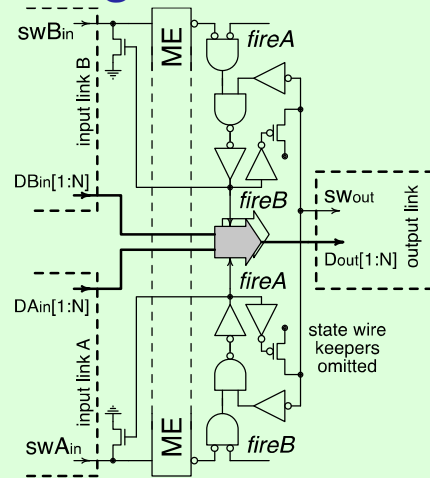


June 2015

Slide 4



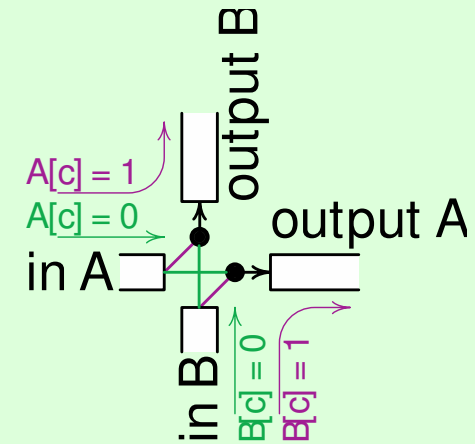
Demand merge



June 2015

Slide 9

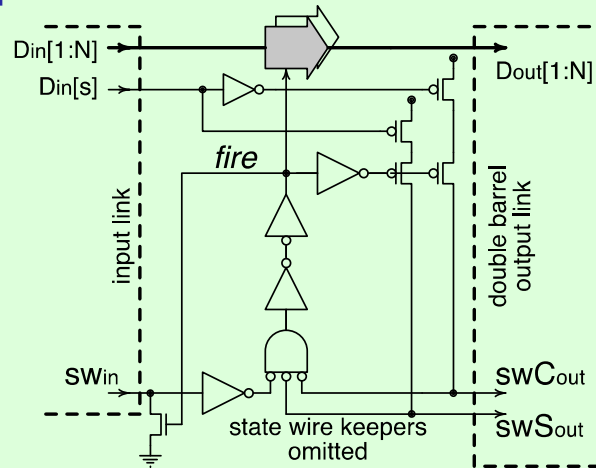
A Double Crosser



June 2015

Slide 10

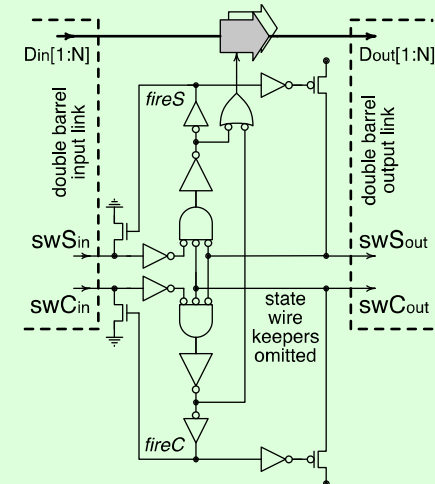
Splitter – double-barrel SW



June 2015

Slide 11

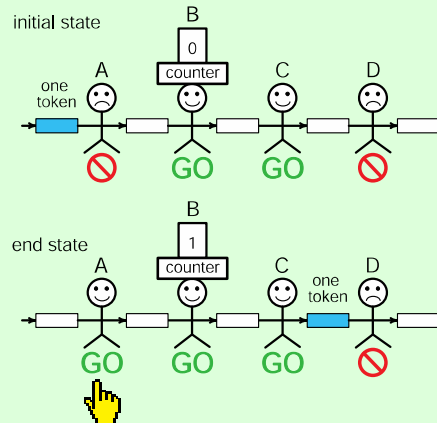
Double-barrel ricochet



June 2015

Slide 12

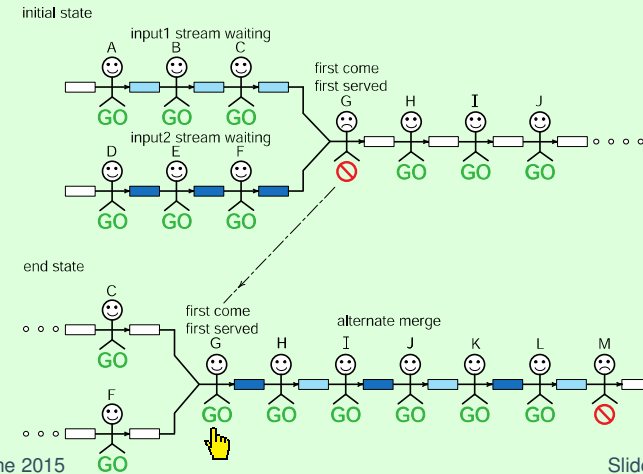
The counter experiment



June 2015

Slide 13

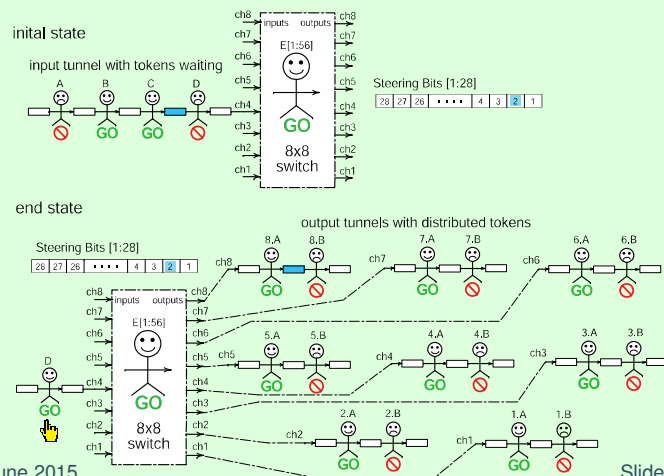
Arbiters alternate if overloaded



June 2015

Slide 14

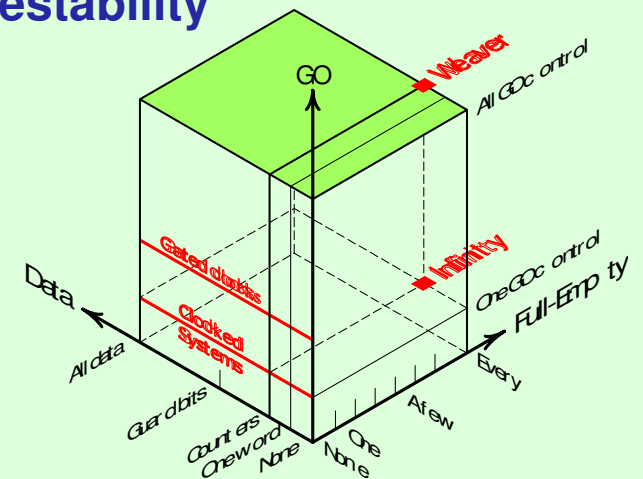
Crossbar steering bit test



June 2015

Slide 15

Testability



June 2015

Slide 16

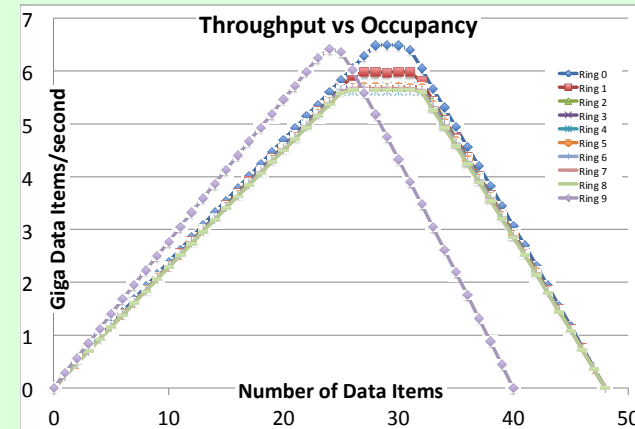
Weaver performance

- Canopy graphs
 - Horizontal axis = occupancy
 - no occupants: no action
 - jammed full: no action
 - Vertical axis = throughput, power, etc.
- Shape reveals details

June 2015

Slide 17

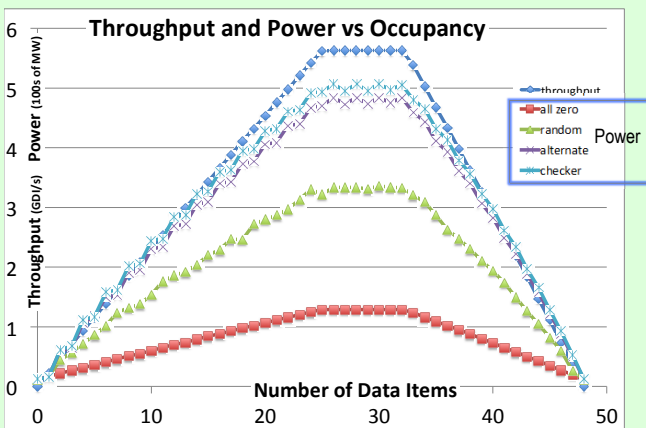
Throughput for all 10 rings



June 2015

Slide 18

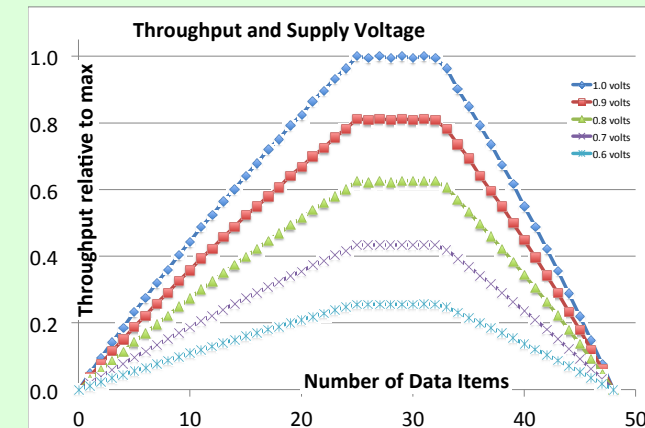
Power vs. data (ring 3)



June 2015

Slide 19

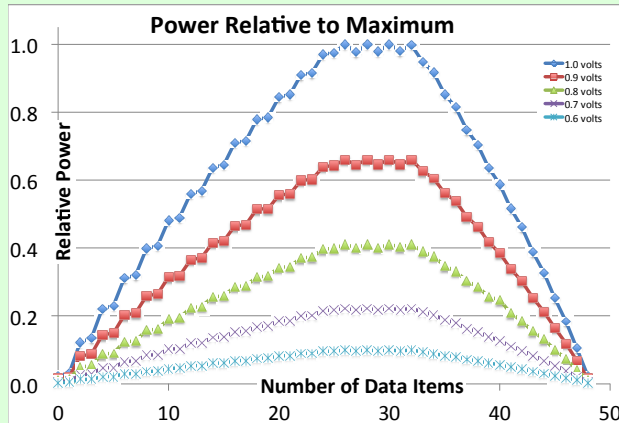
Throughput vs. voltage (ring 3)



June 2015

Slide 20

Power vs. voltage (ring 3)



June 2015

Slide 21

Our 40 nm TSMC chip exhibits

- First example of MrGO control for full test and debug
- 3.4 Terabits/second max throughput
- Combination of **testability** and **speed** suits our circuits to on-chip networks
 - > High throughput
 - > Reduced latency
 - > Low energy
 - > Span multiple clock domains
 - > Fewer synchronizers

June 2015

Slide 22

Acknowledgements – thanks to:

- Oracle – paid for the Weaver chip
- Jon Lexau – lots of help in finishing
- Portland State – for space and students
- Sponsors of the ARC:
 - > Oracle
 - > DARPA, Microsoft
 - > Private Individuals

June 2015

Slide 23

Discussion

June 2015

Slide 24



Fudan University, Shanghai