Silicon Compilation and Test for Dataflow implementations in GasP and Click

Swetha Mettala Gilla

PhD Dissertation Defense
Asynchronous Research Center (ARC)
Maseeh College of Engineering and Computer Science
Portland State University
3 November 2017

Acknowledgements

Advisors

Dr. Xiaoyu Song	(ECE)
Drs. Marly Roncken	(CS, ARC)

Committee

Dr. Douglas V. Hall	(ECE)
Dr. Robert Daasch	(ECE)
Dr. Ivan Sutherland	(ECE, ARC)
Dr. Mark Jones	(CS, Graduate Office Representative)

ARCwelders

Hoon Park
Chris Cowan
Chris Chen
Yong Hei

Publications

Journals:

- Swetha Mettala Gilla, Marly Roncken, and Ivan Sutherland
  Hoi Polloi Mutual Exclusion, IEEE Transactions on VLSI
  plan to submit by end of 2017.

Conferences and books:

- Marly Roncken, Swetha Mettala Gilla, Hoon Park, Navaneeth Jamadagni, Chris Cowan, and Ivan Sutherland
  Naturalized Communication and Testing, ASYNC 2015, pages 77-84, 2015.
- Swetha Mettala Gilla
  Testing with MrGO, ASYNC 2015 web site.
- Marly Roncken, Swetha Mettala Gilla, Hoon Park, Robert Daasch, Xiaoyu Song, Chris Cowan, and Ivan Sutherland
  Beyond Carrying Coal to Newcastle: Dual Citizens and Circuits
  Andrey Mokhov (Ed.) This Asynchronous world – essays dedicated to Alex Yakovlev
  Newcastle University, pages 241– 293, July 2016.
- Swetha Mettala Gilla, Marly Roncken, and Ivan Sutherland
  Long Range GasP with Charge Relaxation
  ASYNC 2010, pages 185-195, 2010.
- Swetha Mettala Gilla
  Library Characterization and Static Timing Analysis of Single-Track Circuits in GasP
  M.Sc. Thesis, Electrical and Computer Engineering, Portland State University, October 2010.

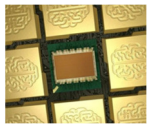
Outline

- Overview
- PART I: Silicon compilation for GasP and Click
- INTERMEZZO: New point of view for design and test
- PART II: Test and debug with state and action control
- Contributions

Motivation: why asynchronous?

- Many modern computer systems are distributed over space.
- Examples:
  - **Internet of things** *[Wikipedia]*
    The network of objects or "things" embedded with electronics, software, sensors, and network connectivity, which enables exchange of data.

  - **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.



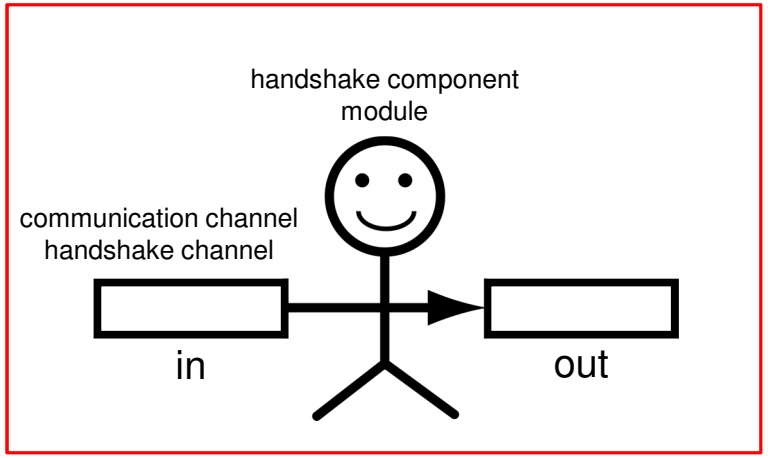
  - **Intel's Loihi chip**
    Energy efficient chip that mimics how the brain functions by learning to operate based on feedback from the environment.

Motivation: why asynchronous?

- 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.
- This is a scalable system.

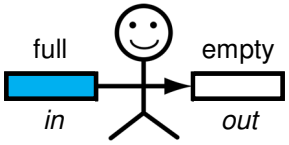
Building blocks
(analogy reminder)

Building blocks

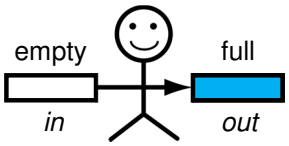


Building blocks: action

WHEN to act:
in is full
and
out is empty

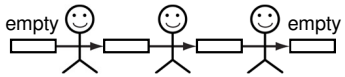


WHAT to do:
- copy data
- drain *in*
- fill *out*



Systems of building blocks

WHEN to act:
in is full
and
out is empty

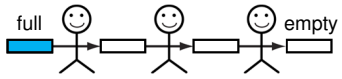


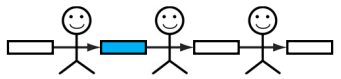
WHAT to do:
- copy data
- drain *in*
- fill *out*

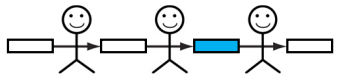
Systems of building blocks

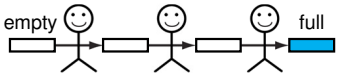
WHEN to act:
in is full
and
out is empty

external fill







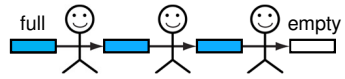


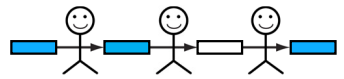
WHAT to do:
- copy data
- drain *in*
- fill *out*

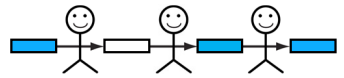
Systems of building blocks

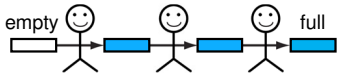
WHEN to act:
in is full
and
out is empty

external drain



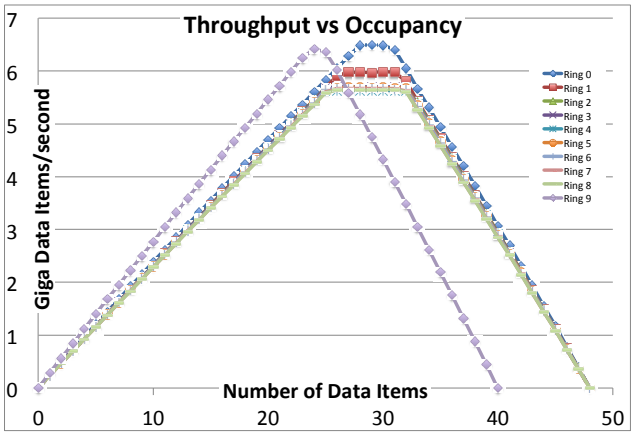






WHAT to do:
- copy data
- drain *in*
- fill *out*

Performance without clocks (Weaver)

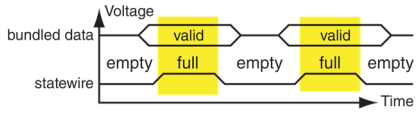


Building blocks at start of thesis

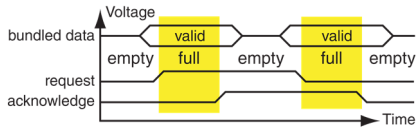
- My building blocks use GasP and Click circuits which differ in handshake protocols and gates.
- GasP circuits are highly customized. They have better logical effort, low power and high speed.
- Click has the most synchronous asynchronous circuits we know. These circuits work better with standard industry tools for static timing analysis, placement and routing.

Handshake protocols

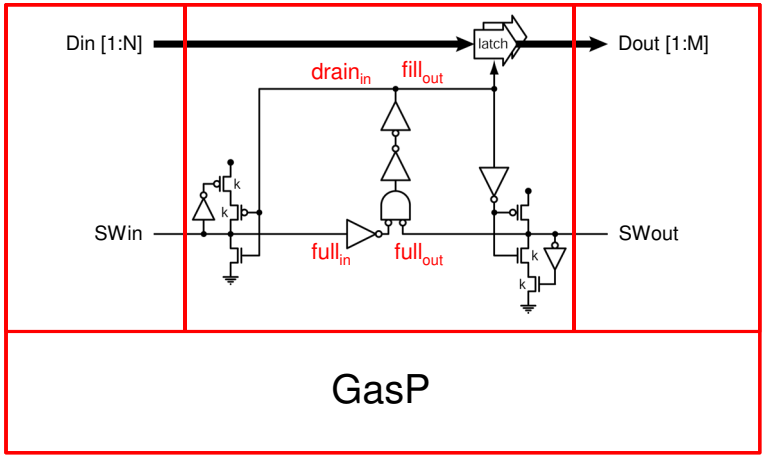
- Handshakes *encode* full, empty, and valid data when full
- Examples:
  - 2-phase return-to-zero (RTZ) with bundled data (used in GasP)
    full: statewire is high / empty: statewire is low



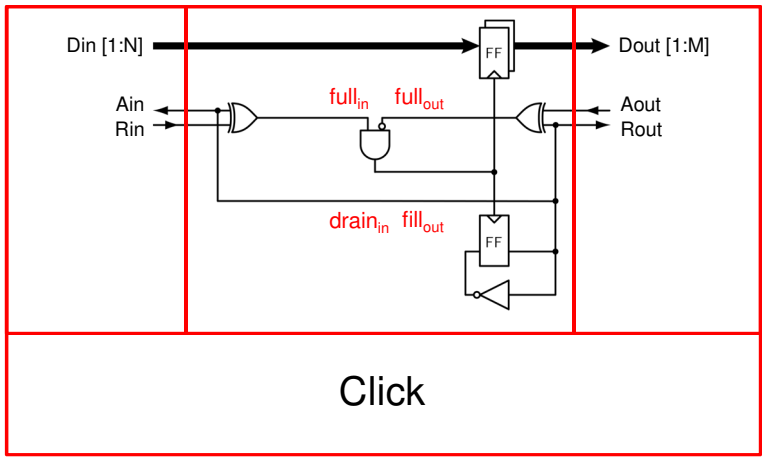
  - 2-phase non-RTZ with bundled data (used in Click)
    full: request ≠ acknowledge / empty: request = acknowledge



Building blocks with handshake interfaces



Building blocks with handshake interfaces



State-of-the-ARC at the start of my thesis

- For design, we used:
  - Electric schematic editor to build circuits in GasP.
  - ARCwelder compiler to build circuits in Click.
- For test, we knew that:
  - Sun-Oracle Labs used a partial scan and functional test to characterize the performance of GasP circuits.
  - Philips used full scan and structural test ("one-shot test") to detect stuck-at faults in Click circuits.

Focus of my thesis

GOAL:

- Create a general automated design and test solution
- for self-timed distributed systems
- that works with any self-timed circuit family
- including mixed systems with GasP and Click.

Focus of my thesis – continued

PART I:

GOAL:

- Extend ARCwelder from Click to **Click and GasP.**

APPROACH:

1. Extend GasP style to support Telescope GasP (TGasP) to match all existing Click designs in ARCwelder.
2. Adapt ARCwelder to generate GasP and TGasP.

OBSERVATION:

- Adaptation required too many code changes.

Focus of my thesis – continued

INTERMEZZO – NEW POINT OF VIEW:

PART I influenced how the ARC views design and test

DESIGN:

- Previous : various handshake interfaces
  Now : common full-empty interfaces
- Previous : components have both communication and computation
  Now : separate communication (links) and computation (joints)

TEST:

- Previous : state-centric test control (scan)
  Now : state control (scan) and action control (MrGO, go)

Focus of my thesis – continued

PART II:

GOAL:

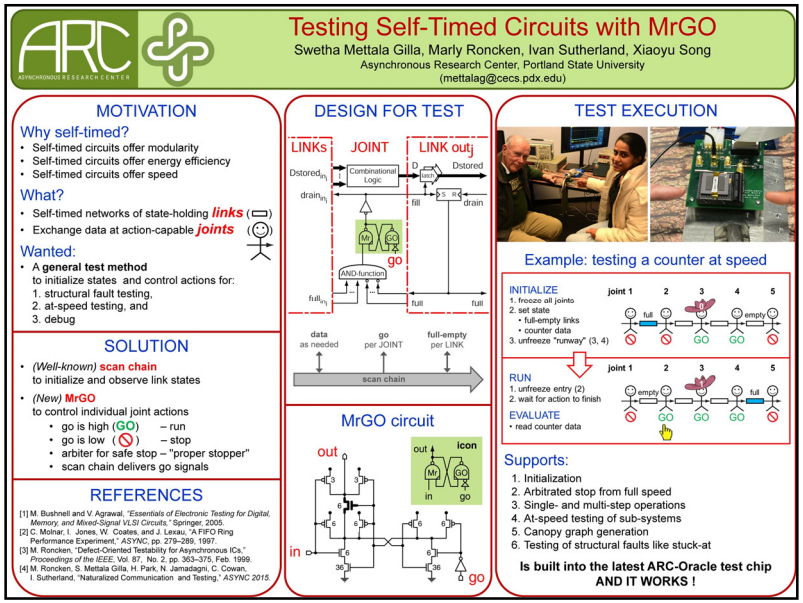
- Work out the new test point of view.

APPROACH:

1. Implement action control: MrGO and go signals.
2. Extend scan test for states to **states and actions.**
3. Create a standard test interface.
4. Demo all of this on real silicon.

OBSERVATION:

- It works perfectly ! (see Poster).



My contributions

System design

- My research influenced ARC's new design and test point of view.
- I created Telescope GasP – a GasP extension still relevant today.
- I extended ARCwelder to support (T)GasP.

Arbitrated circuits

- With MrGO, arbiters are everywhere now.
- I improved the noise tolerance for arbiter inputs and outputs.
- I created a mathematical foundation to size arbiters for speed.

Test, debug, and performance characterization

- I implemented MrGO and scan.
- I built and demo-ed the combined MrGO-scan solution on silicon.
- I proposed an initialization solution which works at power-up.

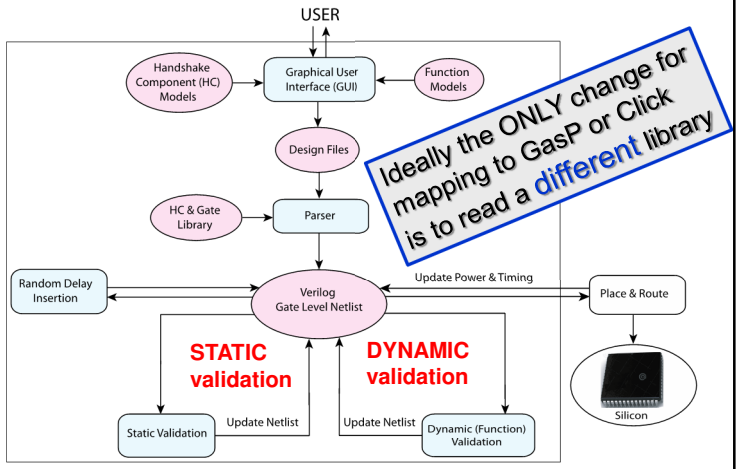
Outline

- Overview
- PART I: Silicon compilation for GasP and Click
- INTERMEZZO: New point of view for design and test
- PART II: Test and debug with state and action control
- Contributions

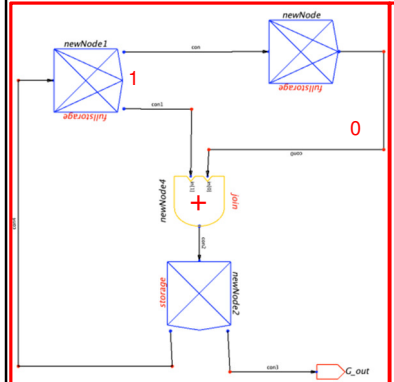
PART I

Compile dataflow designs to GasP circuits using
- Electric to build GasP libraries
- ARCwelder to map the designs onto the libraries

Silicon Compilation: ARCwelder



ARCwelder Graphical User Interface(GUI)



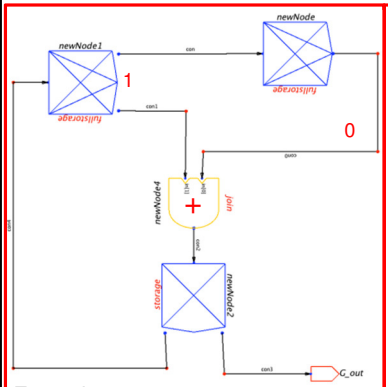
Example:
- Fibonacci number generator

Graphical design representation with:
- network of handshake components
- connected by handshake channels

Textual design representation for:
- data types
- data computations
- component handshake behavior
- channel handshake communication

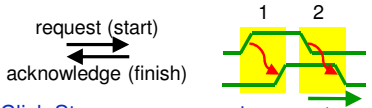
GUI can simulate the design:
- showing output data + handshakes
- for user input data + handshakes
- for GUI built-in timing model

ARCwelder Parser to Click Storage

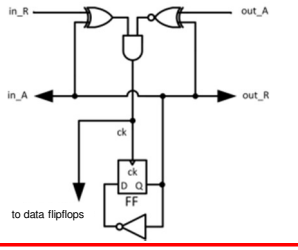


Example:
- Fibonacci number generator

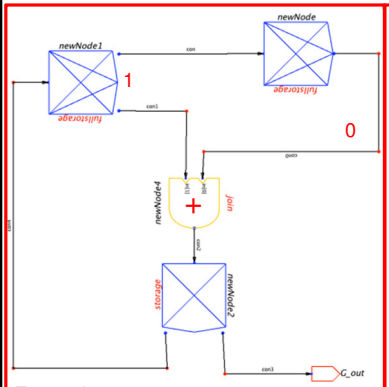
Click Handshake channel:
- Two wires, 2-phase non-RTZ handshake



Click Storage component:

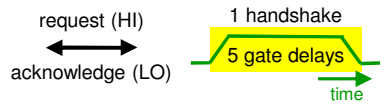


ARCwelder Parser to GasP Storage

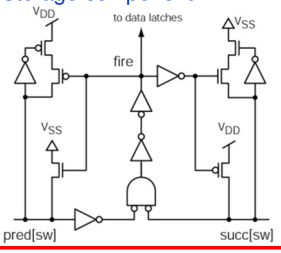


Example:
- Fibonacci number generator

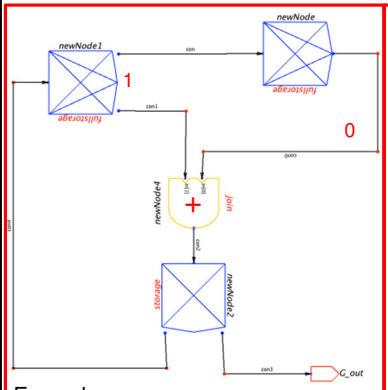
GasP Handshake channel:
- One (bidirectional) wire, 2-phase RTZ



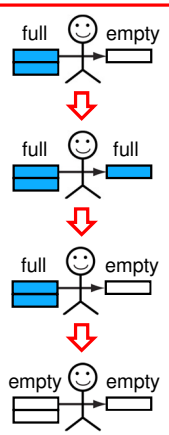
GasP Storage component:



ARCwelder Parser to Click Join

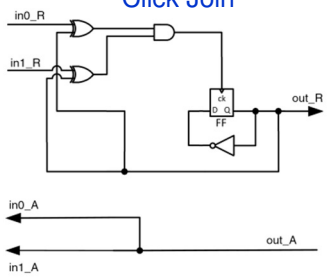


Example:
- Fibonacci number generator



ARCwelder to Click and GasP Join

Click Join



- Telescope handshake behavior: inputs finish AFTER outputs finish

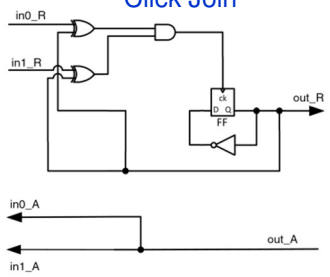
GasP Join

Solution:
new GasP module collection
TGasP (Telescope GasP)

- GasP has only parallel handshakes: inputs complete WITH start of output

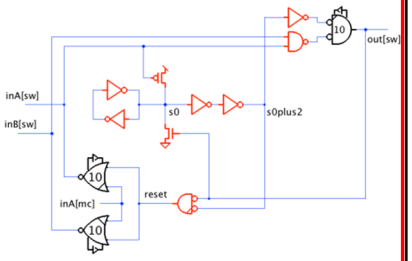
ARCwelder Parser to Click and TGasP Join

Click Join



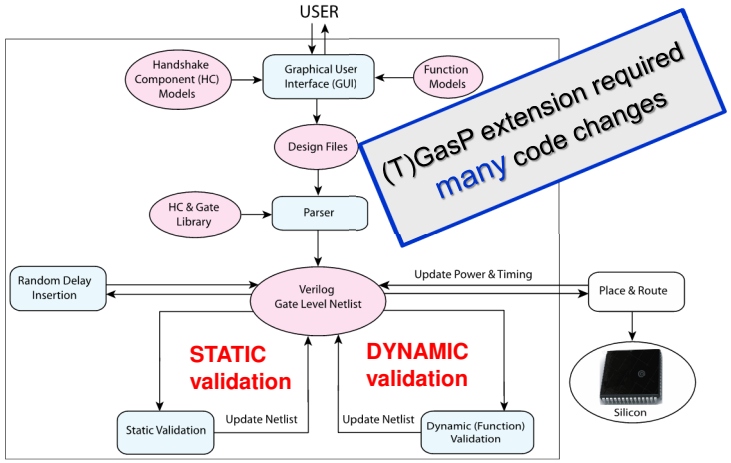
- Telescope handshake behavior: inputs finish AFTER outputs finish

TGasP Join



- Same handshake behavior in GUI but uses GasP handshake channels

ARCwelder extension: bad news



ARCwelder extension: good news

- New TGasP family
  PRO:
  - GasP + TGasP provide a 1-1 replacement for Click.
  - Previous GUI designs can now be mapped to GasP + TGasP.
  - Telescope protocols provide safe dataflow by design.

  CON:
  - TGasP is slower than GasP.
- ARCwelder extension works for the entire flow including validation
  - Static: combinational GasP loops require special care.
  - Dynamic: custom GasP requires powerful simulator like Modelsim.
- Initial experimental results for TGasP are promising
  - (T)GasP is 24-34% faster than Click for no-delay datapaths.
  - Speeds are more similar when datapaths take more time.

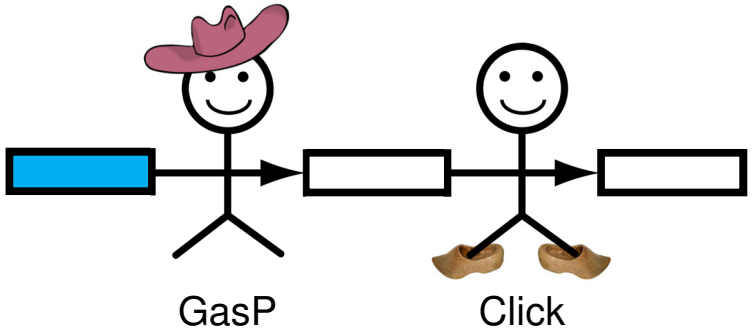
Outline

- Overview
- PART I: Silicon compilation for GasP and Click
- INTERMEZZO: New point of view for design and test
- PART II: Test and debug with state and action control
- Contributions

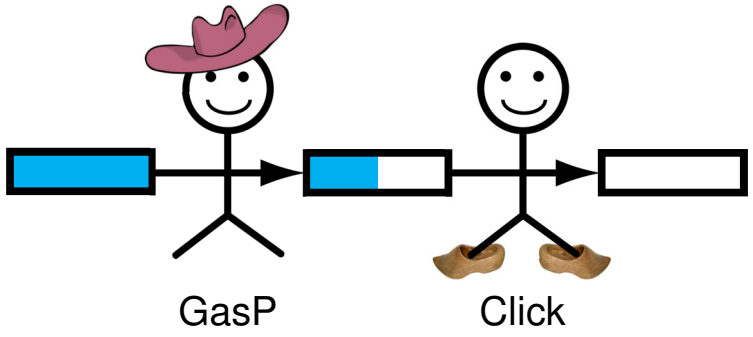
INTERMEZZO

INTERMEZZO I: New point of view for DESIGN
- common full-empty interfaces
- separate communication (links) and computation (joints)

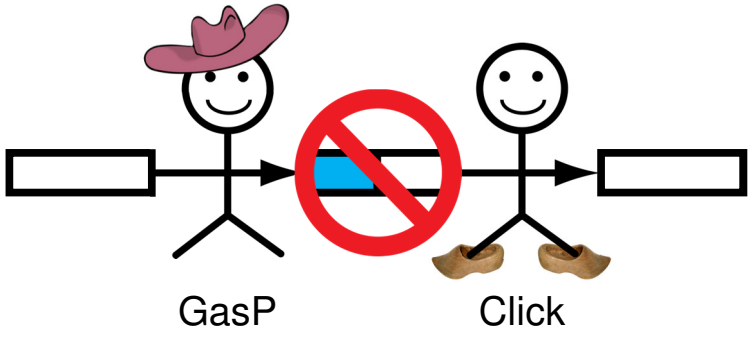
The problem with handshake interfaces



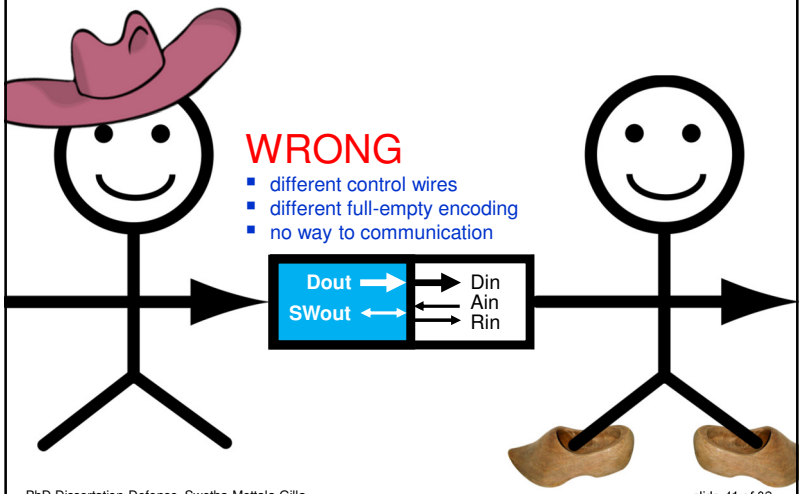
The problem with handshake interfaces



The problem with handshake interfaces



The problem with handshake interfaces

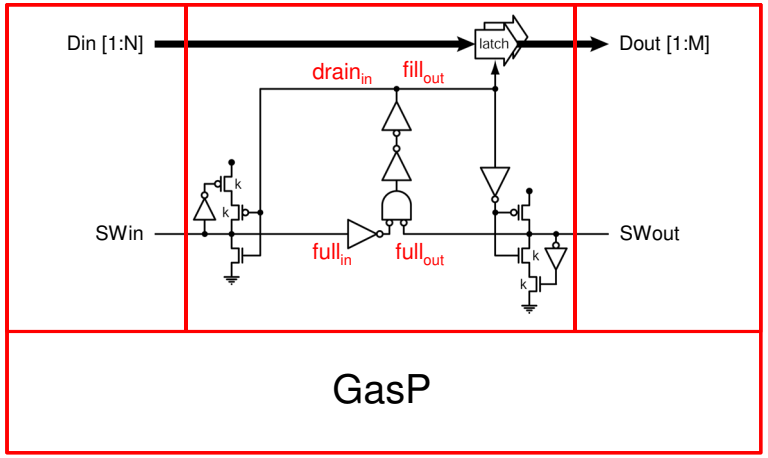


The problem with handshake interfaces

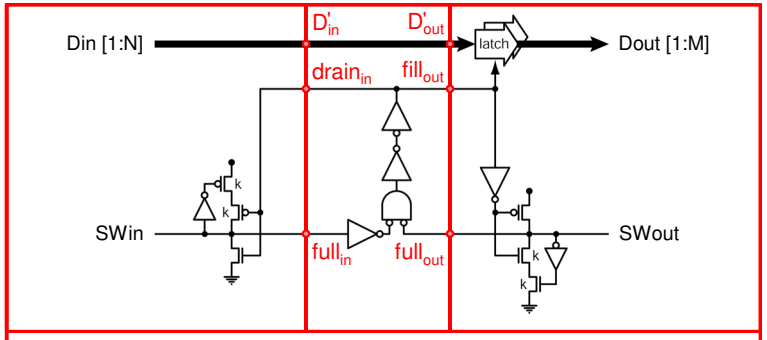
if you mix handshakes, you need full-empty translators

GasP revisited

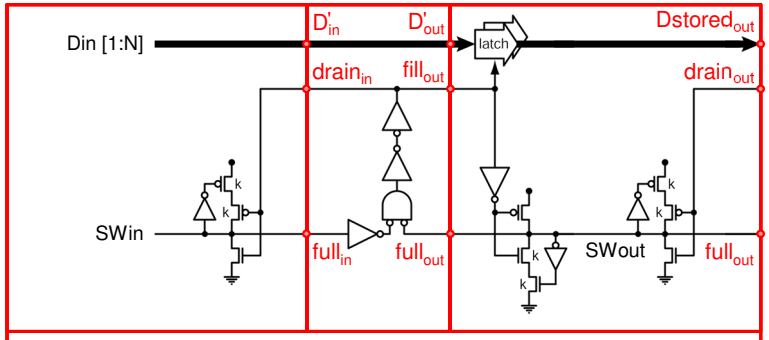
Building block interfaces ...from



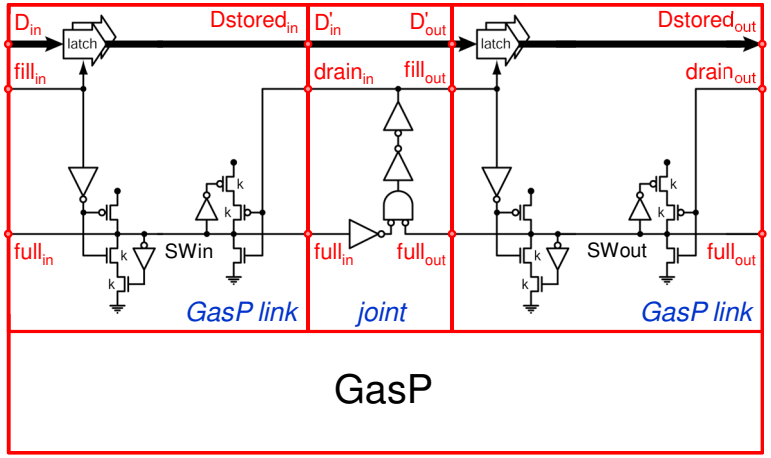
Building block interfaces ...to



Building block interfaces ...to

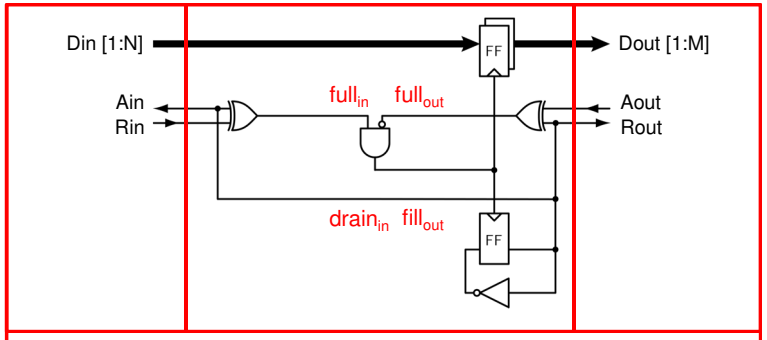


Building block interfaces ...to



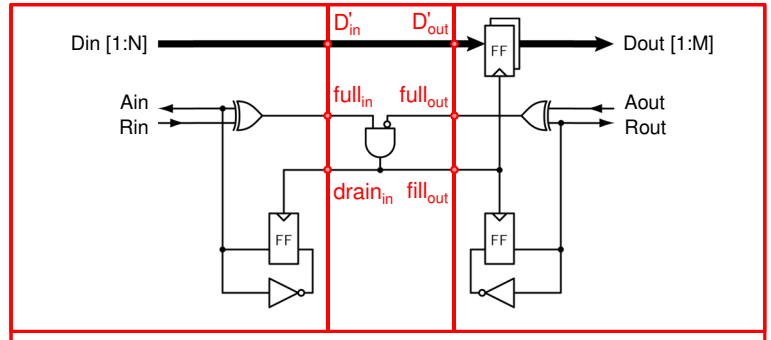
Click revisited

Building block interfaces ...from



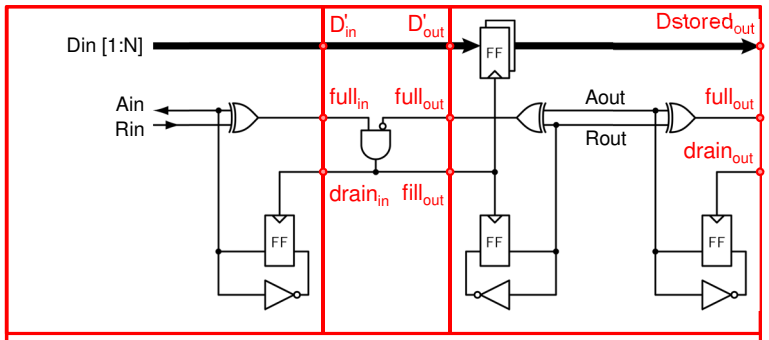
Click

Building block interfaces ...to



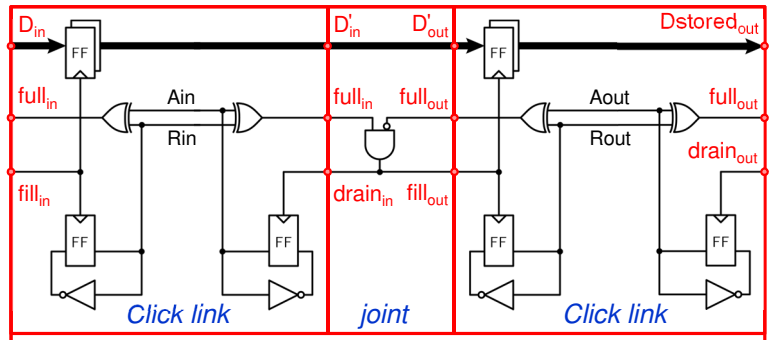
Click

Building block interfaces ...to



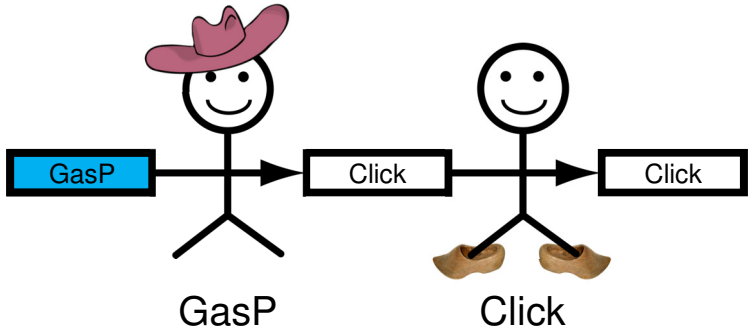
Click

Building block interfaces ...to

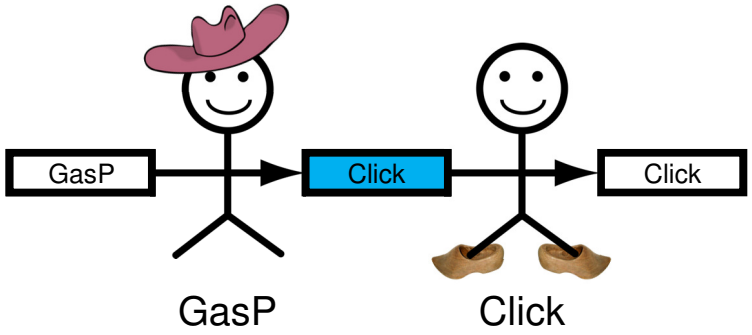


Click

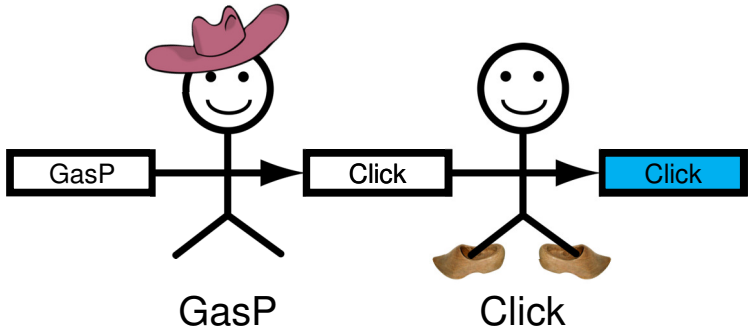
Solution with full-empty interfaces



Solution with full-empty interfaces



Solution with full-empty interfaces



Solution with full-empty interfaces

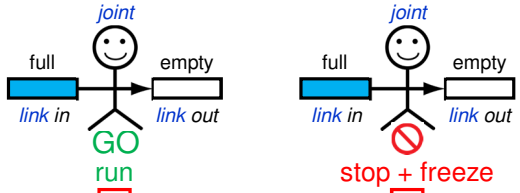
- Provides translation-free communication.
- Simplifies collaboration and design re-use.

INTERMEZZO

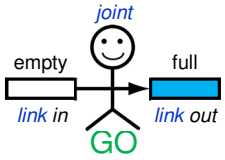
INTERMEZZO II: New point of view for TEST
- recognize and control actions individually

Building blocks: action with GO control

WHEN to act:
in is full
and
out is empty
and
GO

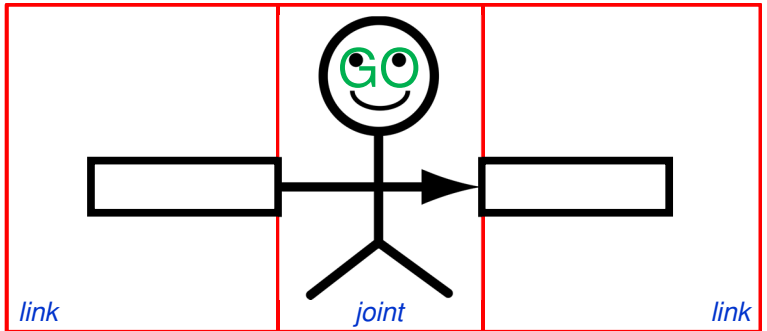


WHAT to do:
- copy data
- drain *in*
- fill *out*



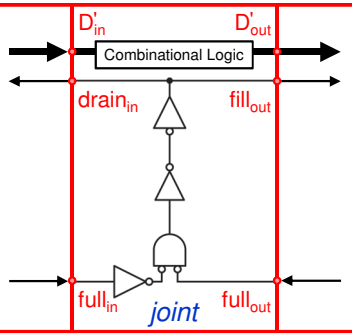
no action

Building blocks: design with GO control



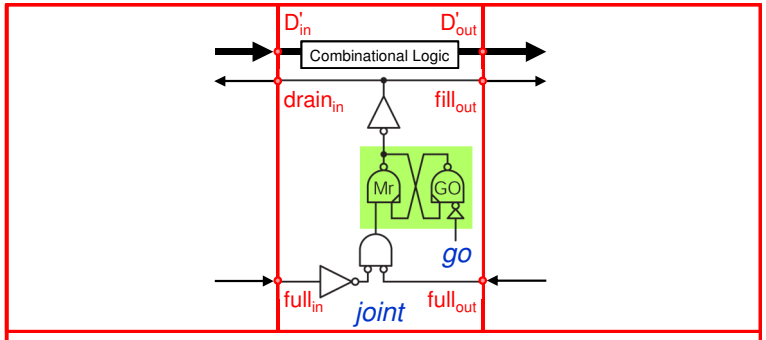
design reminder

Building blocks: design with GO control



design reminder

Building blocks: design with GO control

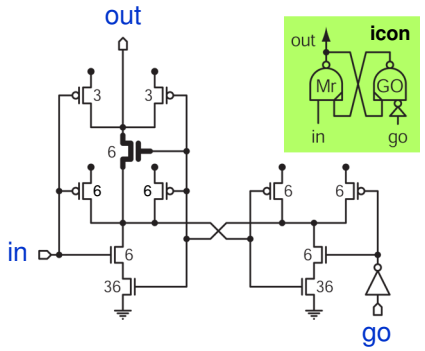


Solution MrGO:
pronounced "Mister GO"

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

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



Outline

- Overview
- PART I: Silicon compilation for GasP and Click
- INTERMEZZO: New point of view for design and test
- PART II: Test and debug with state and action control
- Contributions

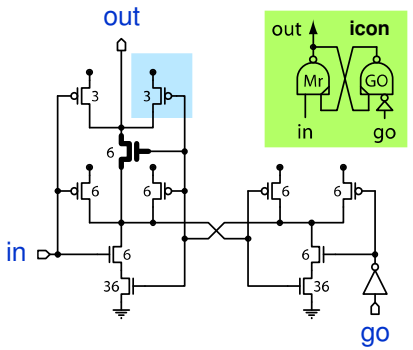
PART II

Work out the new test solution

- optimize arbiters for design and MrGO use
- combine scan with go control
- demo the combined solution on silicon

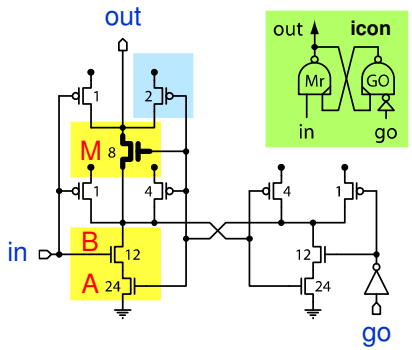
Optimize arbiters for noise and speed

- Added keepers to drive *out* when *in* waits to be granted.



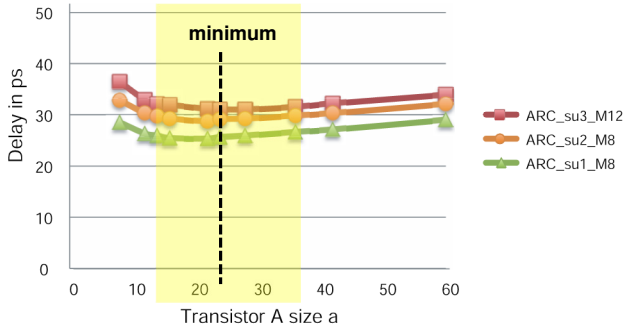
Optimize arbiters for noise and speed

- Added keepers to drive *out* when *in* waits to be granted.
- Developed mathematics to size for uncontested grant.



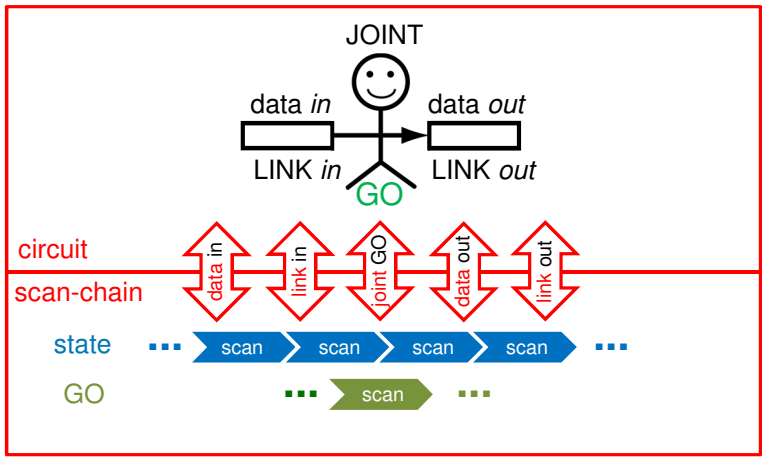
Optimize arbiters for noise and speed

- ARC arbiter delay graphs in SPICE
- for 90nm CMOS with typical gate delay ~20 ps
- for uncontested *in* HI to *out* LO
- with B fixed to size 12
- show nearly constant delay
- for a range of A and M sizes and arbiter stepups.

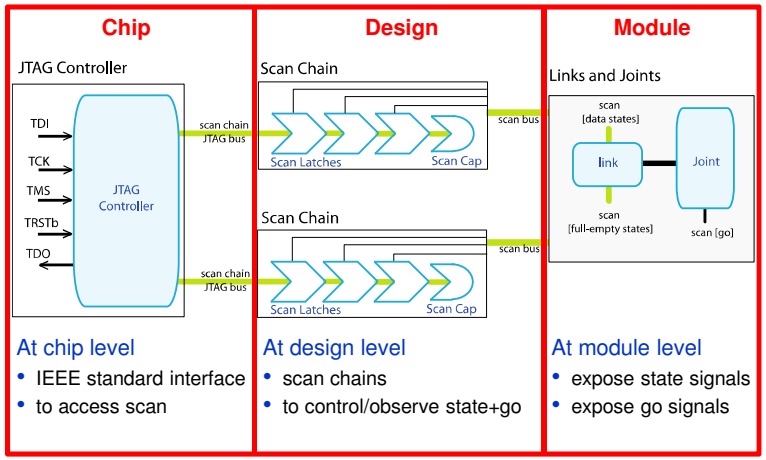


Combine scan with *go* control

Combine scan with go control

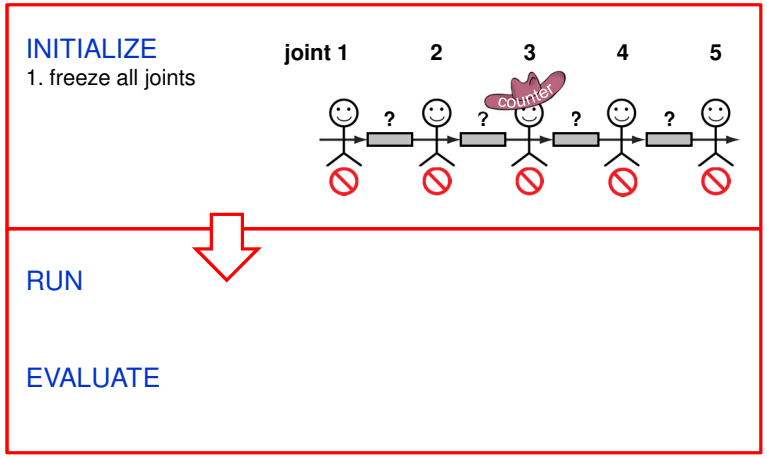


Standard scan test interface



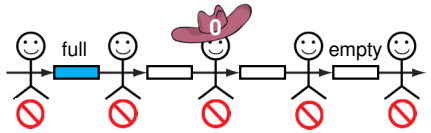
Demo the combined solution on silicon

Testing a counter at speed



Testing a counter at speed

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

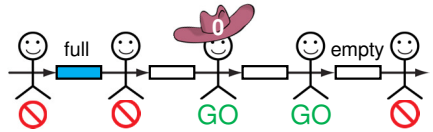


RUN

EVALUATE

Testing a counter at speed

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

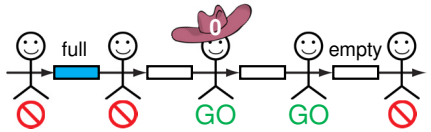


RUN

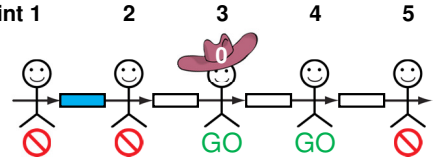
EVALUATE

Testing a counter at speed

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



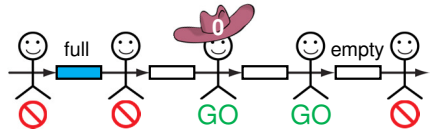
RUN



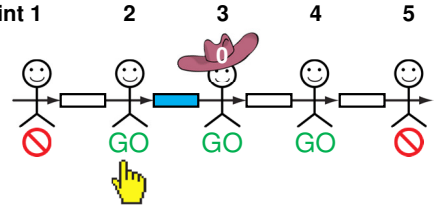
EVALUATE

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



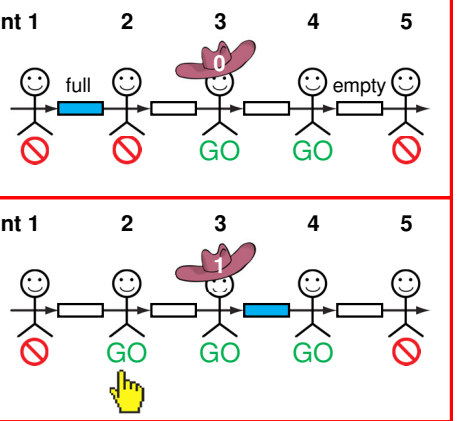
EVALUATE

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

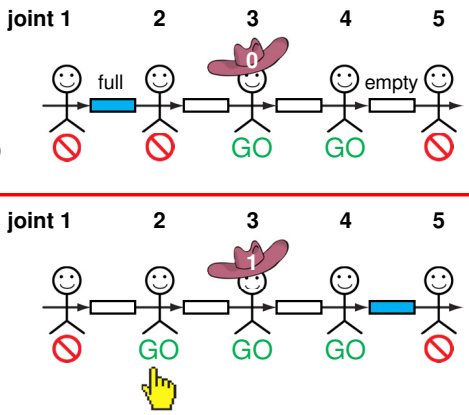


Testing a counter at speed

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

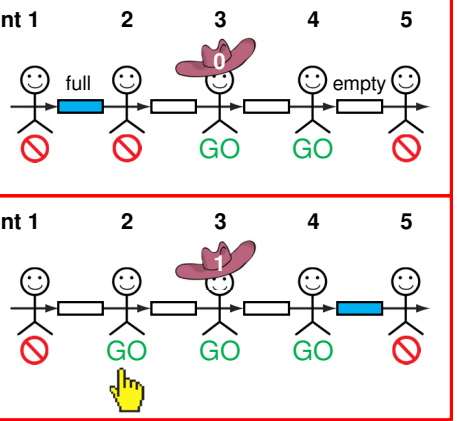


Testing a counter at speed

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

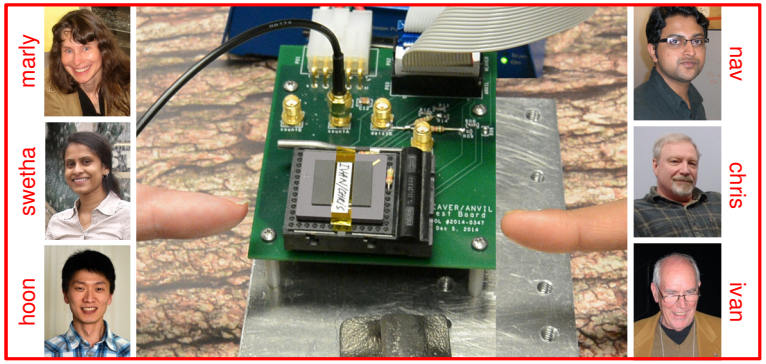
EVALUATE
- read counter data



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.



My contributions (reminder)

System design

- My research influenced ARC's new design and test point of view.
- I created Telescope GasP – a GasP extension still relevant today.
- I extended ARCwelder to support (T)GasP.

Arbitrated circuits

- With MrGO, arbiters are everywhere now.
- I improved the noise tolerance for arbiter inputs and outputs.
- I created the mathematical foundation to size arbiters for speed.

Test, debug, and performance characterization

- I implemented MrGO and scan.
- I built and demo-ed the combined MrGO-scan solution on silicon.
- I proposed an initialization solution which works at power-up.

Publications (reminder)

Journals:

- Swetha Mettala Gilla, Marly Roncken, and Ivan Sutherland
  Hoi Polloi Mutual Exclusion, IEEE Transactions on VLSI
  plan to submit by end of 2017.

Conferences and books:

- Marly Roncken, Swetha Mettala Gilla, Hoon Park, Navaneeth Jamadagni, Chris Cowan, and Ivan Sutherland
  Naturalized Communication and Testing, ASYNC 2015, pages 77-84, 2015.
- Swetha Mettala Gilla
  Testing with MrGO, ASYNC 2015 web site.
- Marly Roncken, Swetha Mettala Gilla, Hoon Park, Robert Daasch, Xiaoyu Song, Chris Cowan, and Ivan Sutherland
  Beyond Carrying Coal to Newcastle: Dual Citizens and Circuits
  Andrey Mokhov (Ed.) This Asynchronous world – essays dedicated to Alex Yakovlev
  Newcastle University, pages 241– 293, July 2016.
- Swetha Mettala Gilla, Marly Roncken, and Ivan Sutherland
  Long Range GasP with Charge Relaxation
  ASYNC 2010, pages 185-195, 2010.
- Swetha Mettala Gilla
  Library Characterization and Static Timing Analysis of Single-Track Circuits in GasP
  M.Sc. Thesis, Electrical and Computer Engineering, Portland State University, October 2010.

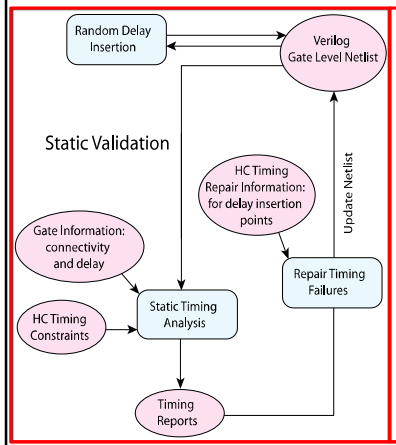
BACKUP

Timeline

	Research Topic	Completion Date
PART I	Development TGasP	April 2012
	First draft (T)GasP extension of ARCwelder compiler	April 2013
	Compiler coding DONE in agreement with PhD mentors	
PART II	Test and debug implementation for GasP	February 2014
	PSU-Oracle test chip evaluation	May 2015
	Test research DONE in agreement with PhD mentors	August 2015
	Arbiter speed optimization for uncontested grant	May 2017
	Thesis (submitted) and journal publication written	December 2017

BACKUP
silicon compilation

ARCwelder Static Validation

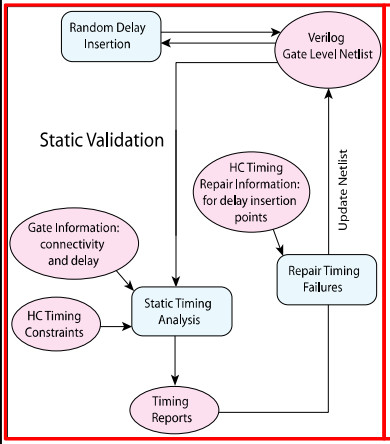


Static Timing Analysis (STA):
- Check relative timings (RT)
  - "POD → short path < long path"
  - RT generation: Hoon Park's PhD
- Using gate delay and connectivity
- Report short-to-long delay difference

Repair Static Timing Failures:
- Read report and repair failing RTs
  - insert delay in longer path
  - using RT insertion point
- Repeat until all RTs are valid

Random Delay Insertion:
- Random delay at random locations
- Run STA + Repair + Dynamic Validation
  - if Repair doesn't converge or Dynamic Validation test fails
  - then unsound or incomplete RTs

Static Validation changes for (T)GasP

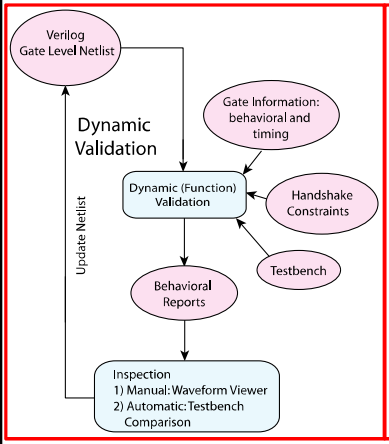


- New RT libraries for (T)GasP
- Asymmetric delay insertions
  - leave 5-gate-delay reset loops intact
- Extend path delay calculations to:
  - bidirectional statewires
  - asymmetric delay insertions
  - start-stop paths at non-flipflops
- New Verilog models
  - to convey delay and connectivity
  - in a uni-directional way
  - without up-down transition info

Conclusion

- ARCwelder's static analysis code needs major changes to integrate (T)GasP systematically with sensible code-sharing

ARCwelder Dynamic Validation



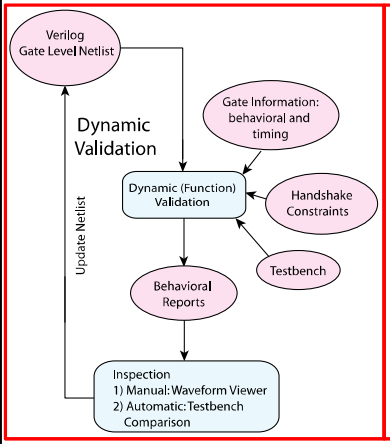
Dynamic (Function) Validation:

- Simulate Verilog netlist using
  - gate delay + connectivity + behavior
  - handshake constraints to monitor, e.g.
    data must be stable in handshake
  - test environment for inputs-outputs
- Report results

Inspection of simulation results:

- Manually
  - by viewing the Verilog waveforms
- Automatically
  - by checking the handshake reports
  - by using test scripts

Dynamic Validation changes for (T)GasP



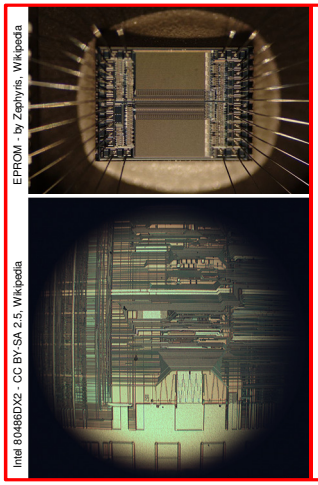
- Too many special Verilog models
  - Willem Mallon's ARCwelder version uses an Icarus Verilog simulator – "free Verilog simulator on a laptop"
  - Icarus Verilog doesn't do custom
  - (T)GasP uses many custom cells
  - Running Icarus Verilog on (T)GasP requires too much modeling to relate the Verilog results to silicon

Conclusion

- Use more powerful Verilog simulators
- Mentor Graphics Modelsim can handle Electric generated Verilog models down to simple gates + a few special models

BACKUP
test and debug

so MANY signals - so FEW pins



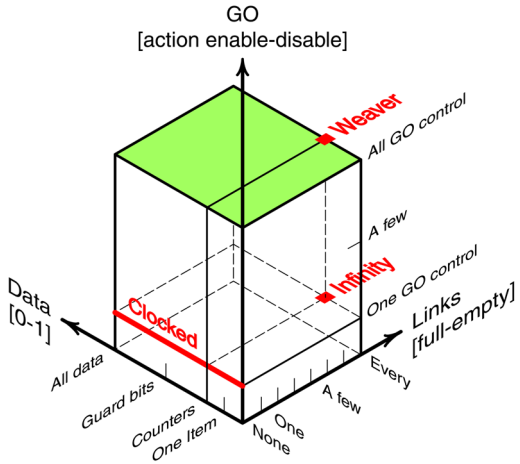
- Isn't this similar to testing software?
  - so many lines - so few exports
- for which the answer is:
  - use an interactive code debugger
  - to set break points, single-step code, etc.

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

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

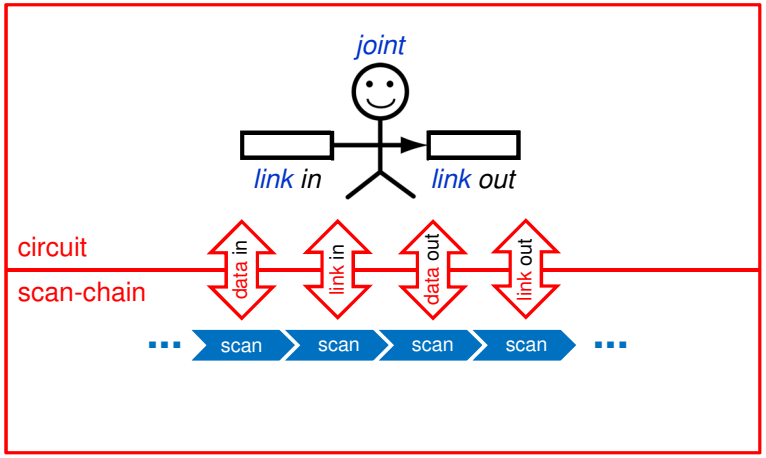
- scan+GO: initialization + single-step and multi-step and at-speed test and debug



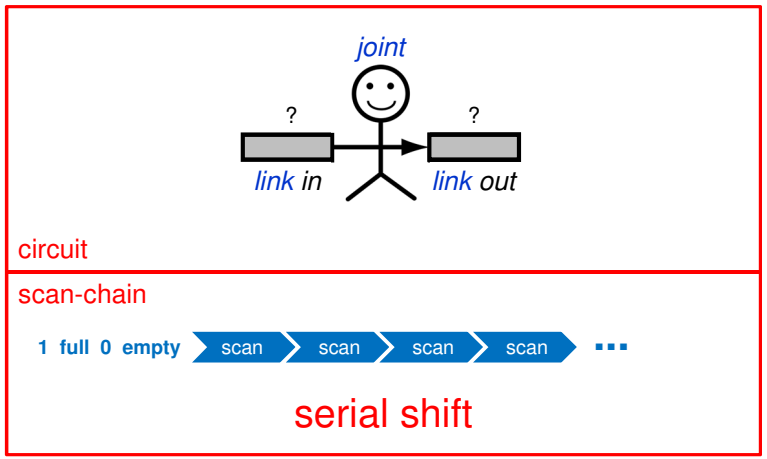
SCAN: pros and cons

- 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

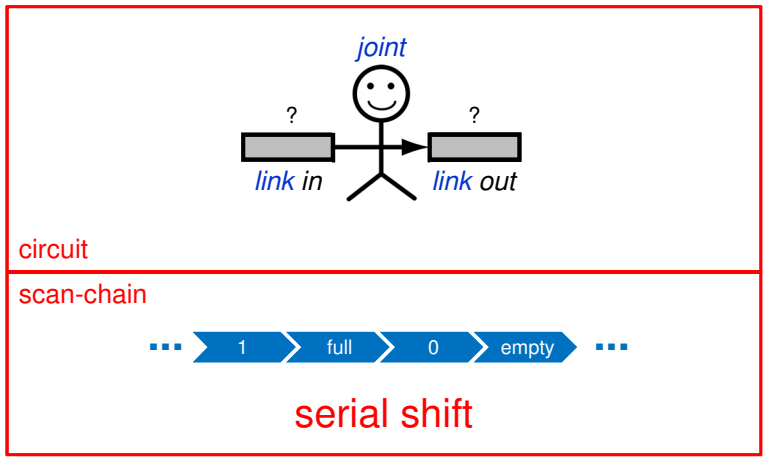
Scan design



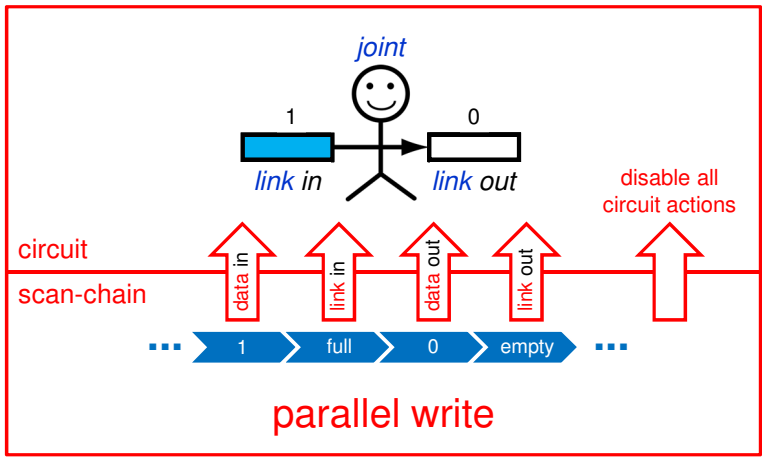
Scan operations: shift in stimuli (1/5)



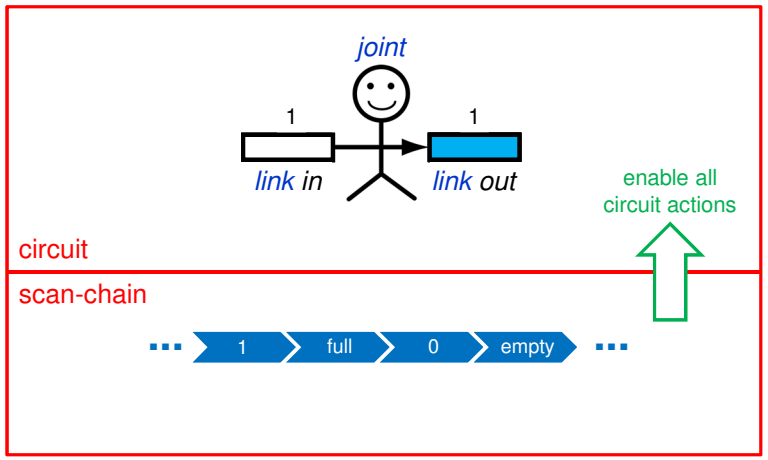
Scan operations: shift in stimuli (5/5)



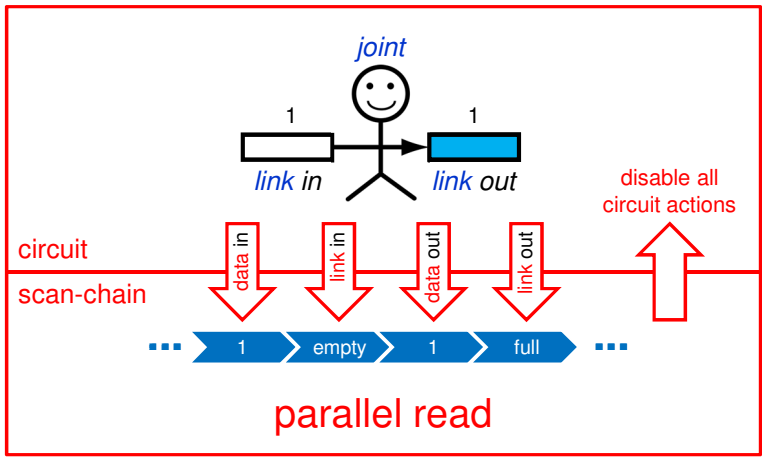
Scan operations: write stimuli to circuit



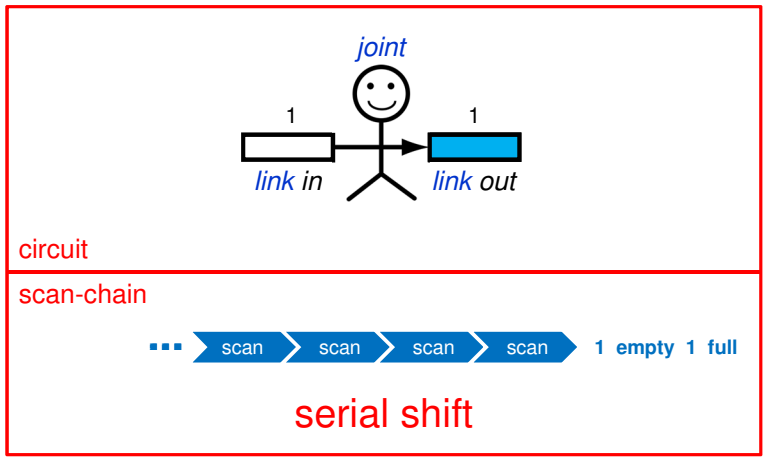
Scan operations: let the circuit run



Scan operations: read results from circuit

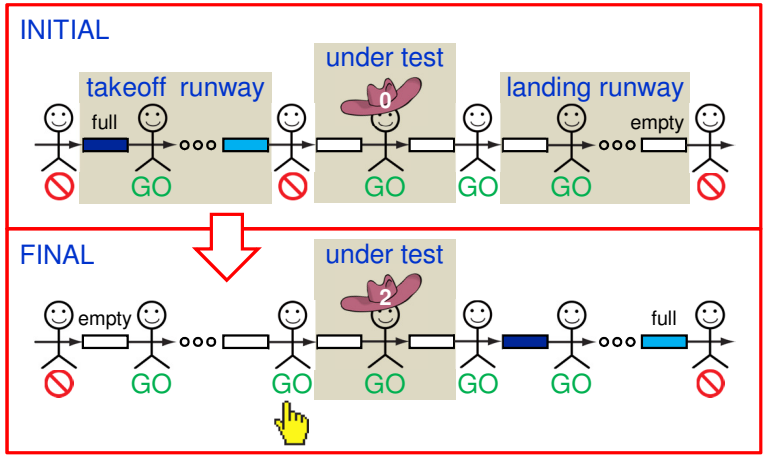


Scan operations: shift out results



BACKUP
testing a burst of data at speed

Testing a burst of data at speed

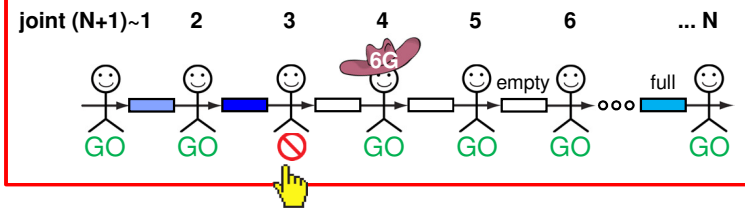


BACKUP
characterization of throughput

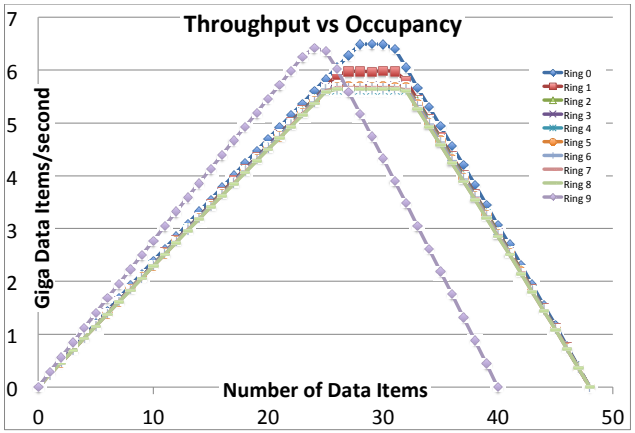
Performance characterization

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

FINAL for *i* ~ 60% links

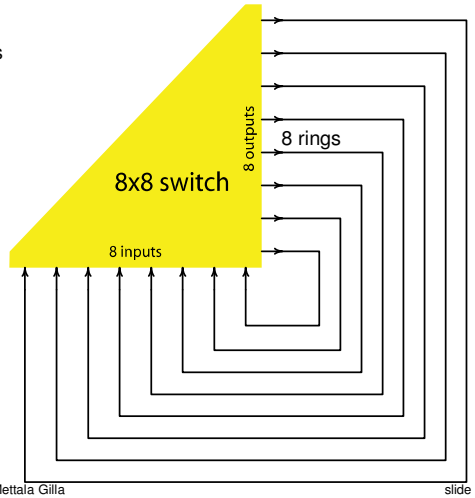


Canopy graphs for all Weaver rings



Weaver overview

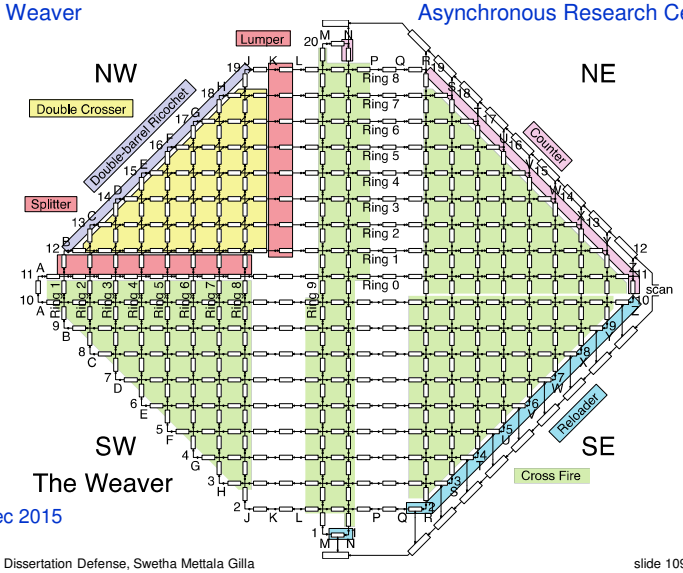
8 recirculating rings provide high-speed data for the switch



Not shown:
2 more rings without switches

The Weaver

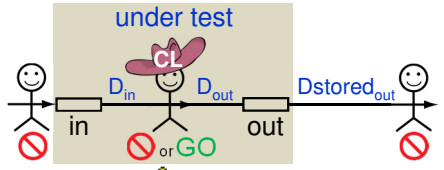
Asynchronous Research Center



Dec 2015

BACKUP
structural fault testing ("one-shot test")

Testing stuck-at faults



TEST control logic

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

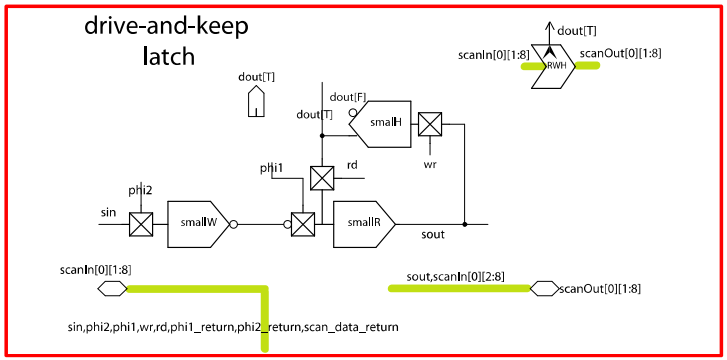
TEST datapath (normally opaque)

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

BACKUP
built-in scan test with standard interface

Scan test at module level:

shift, drive-keep, read, write

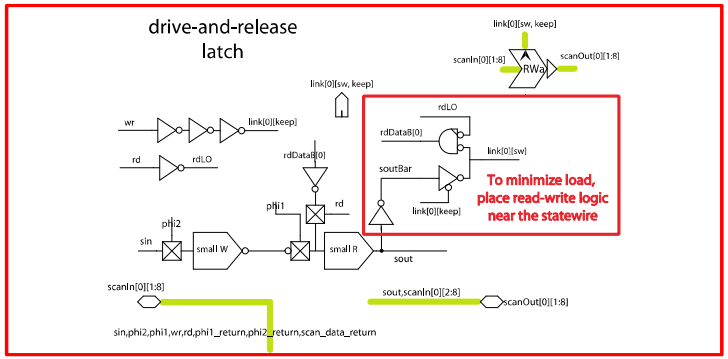


Scan design for signals not changed by GasP, like *go*

- Master-slave latch pair to shift, clocked by *phi1* and *phi2*
- Write/drive-keep latch, clocked by *wr*
- Read control to slave latch, clocked by *rd*

Scan test at module level:

shift, drive-release, read, write

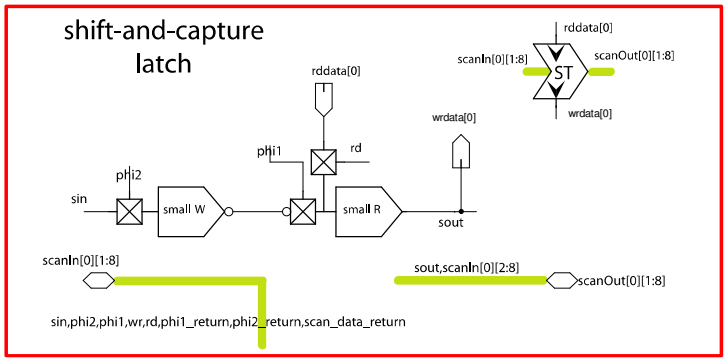


Scan design for state signals changed by GasP, like *full*

- Master-slave latch pair to shift, clocked by *phi1* and *phi2*
- Write/drive-release control, clocked by *wr*
- Read control to slave latch, clocked by *rd*

Scan test at module level:

shift, drive-mux, read, write

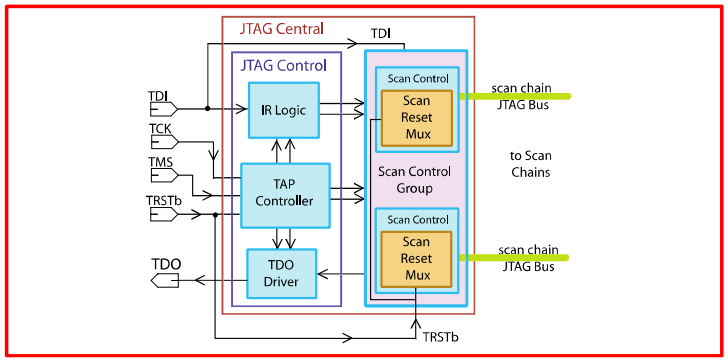


Scan design for state signals with mux control, like *data*

- Master-slave latch pair to shift, clocked by *phi1* and *phi2*
- Write/drive-mux control (muxed in datapath), clocked by *wr*
- Read control to slave latch, clocked by *rd*

Scan test at chip level:

standard test interface



IEEE 1149.1 scan test interface: industry standard

- 4 input pins, 1 output pin, internal bus interfaces to scan chains
- TAP controller to program scan operations
- Immediate control for stable-state initialization at power-up