Asynchronous Dataflow: MOUSETRAP Pipelines

Montek Singh

UNC Chapel Hill

ASYNC 2022 Summer School

Week 2: June 13

MOUSETRAP Pipelines

[Singh/Nowick, ICCD 2001 & TVLSI 2007

Simple asynchronous implementation style, uses…

- *transparent D-latches* + *standard combinational function logic*
- *simple control:* 1 gate/pipeline stage

Uses a "capture protocol": Latches are …

- normally transparent while waiting for data
- become opaque after data arrives

Control Signaling: *transition-signaling* = *2-phase*

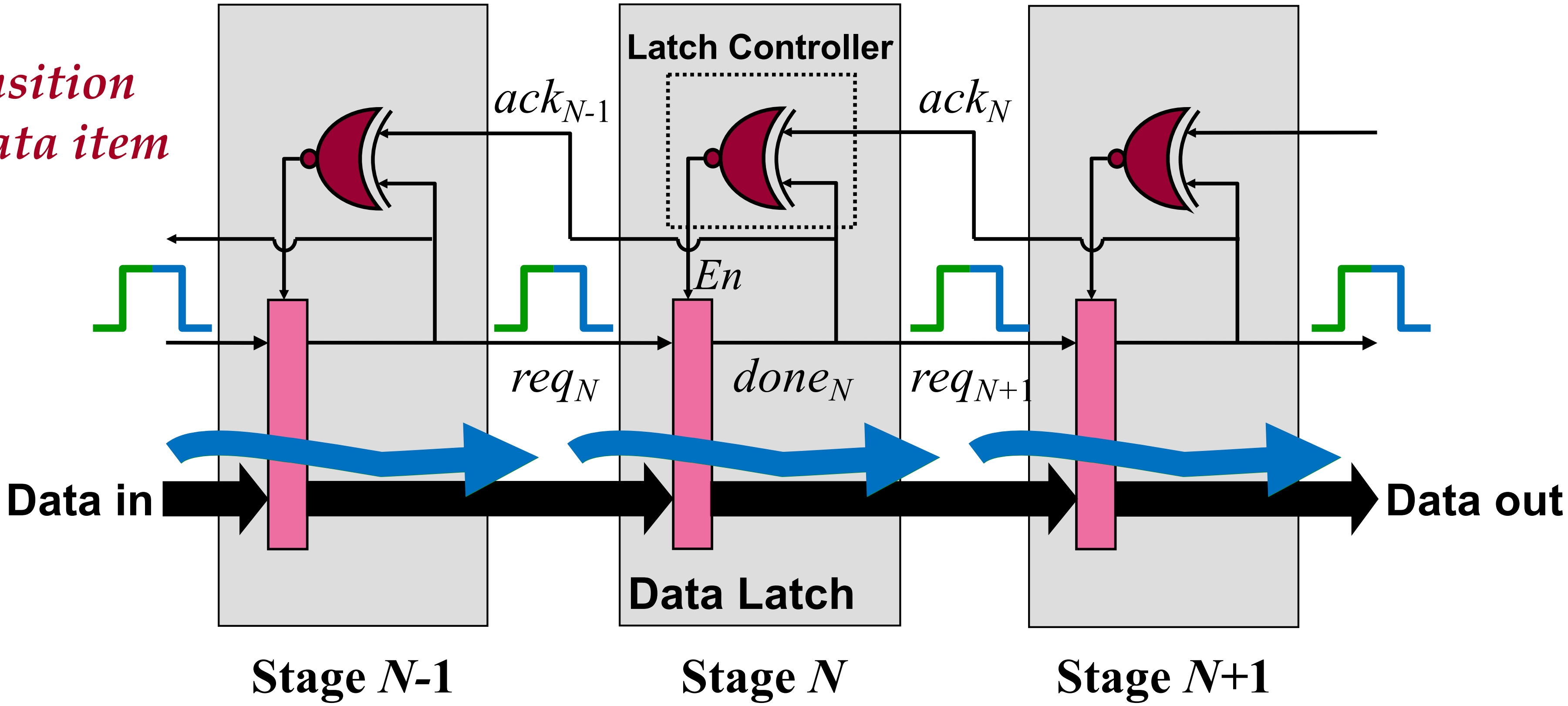
Goals:

- fast cycle time
- simple inter-stage communication
- standard cell implementation

MOUSETRAP: A Basic FIFO

Stages communicate using *transition-signaling:*

1 transition per data item



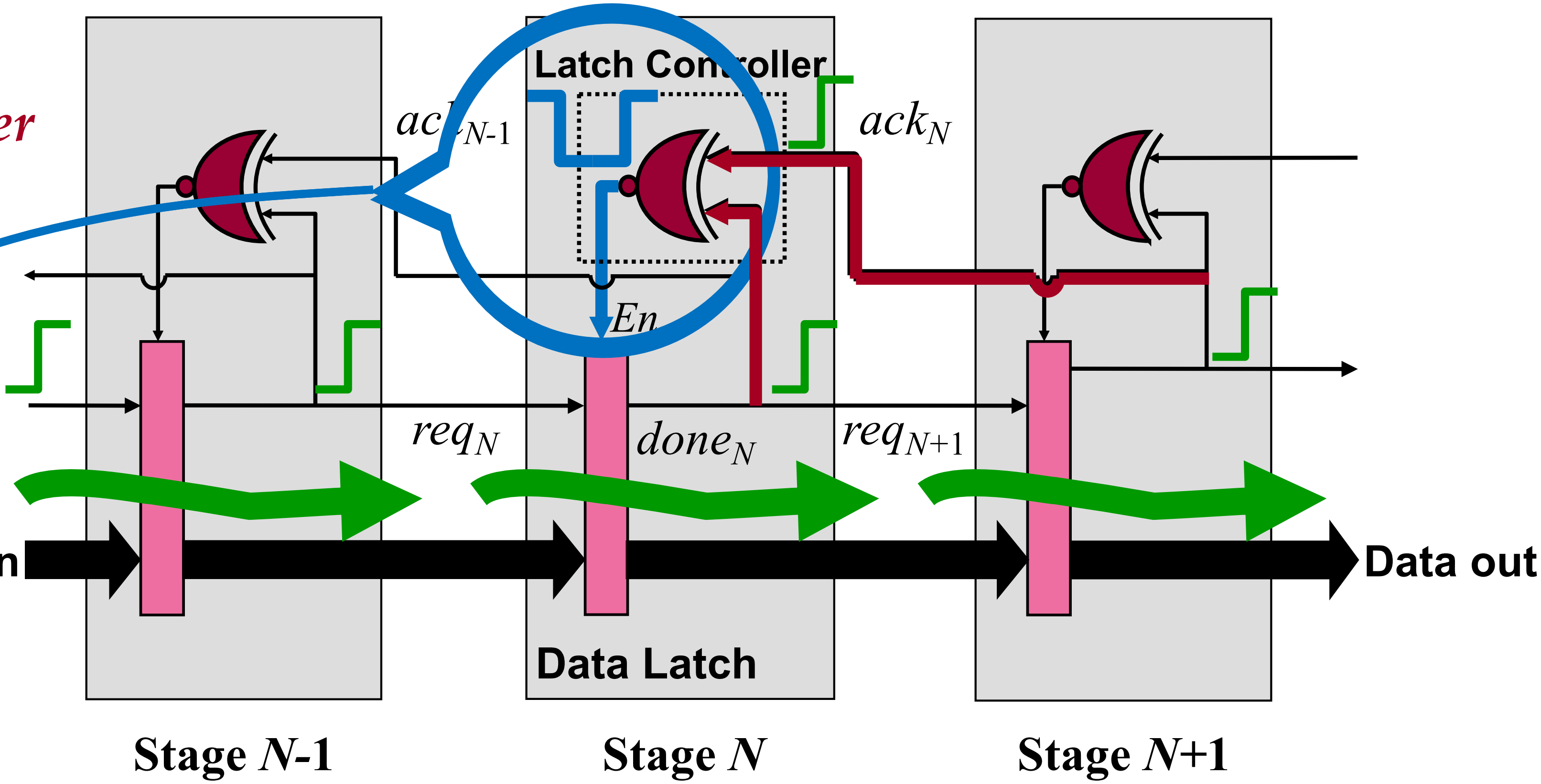
2nd data item flowing through the pipeline

MOUSETRAP: A Basic FIFO (contd.)

Latch controller (XNOR) acts as *"phase converter"*:

- 2 distinct transitions (up or down) ➔ pulsed latch enable

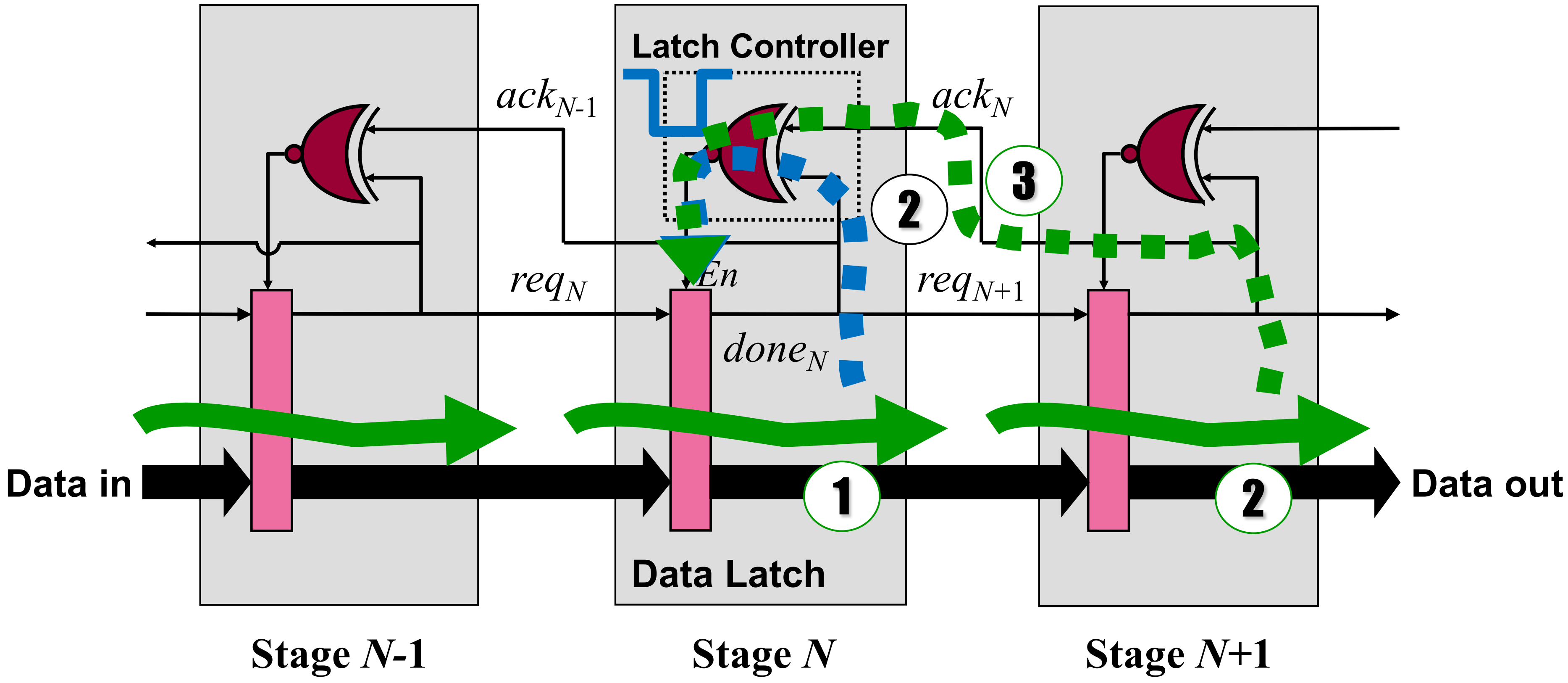
2 transitions per latch cycle



Latch is disabled when *current stage is "done"*

Latch is re-enabled when *next stage is "done"*

MOUSETRAP: FIFO Cycle Time



N re-enabled to compute

:
self

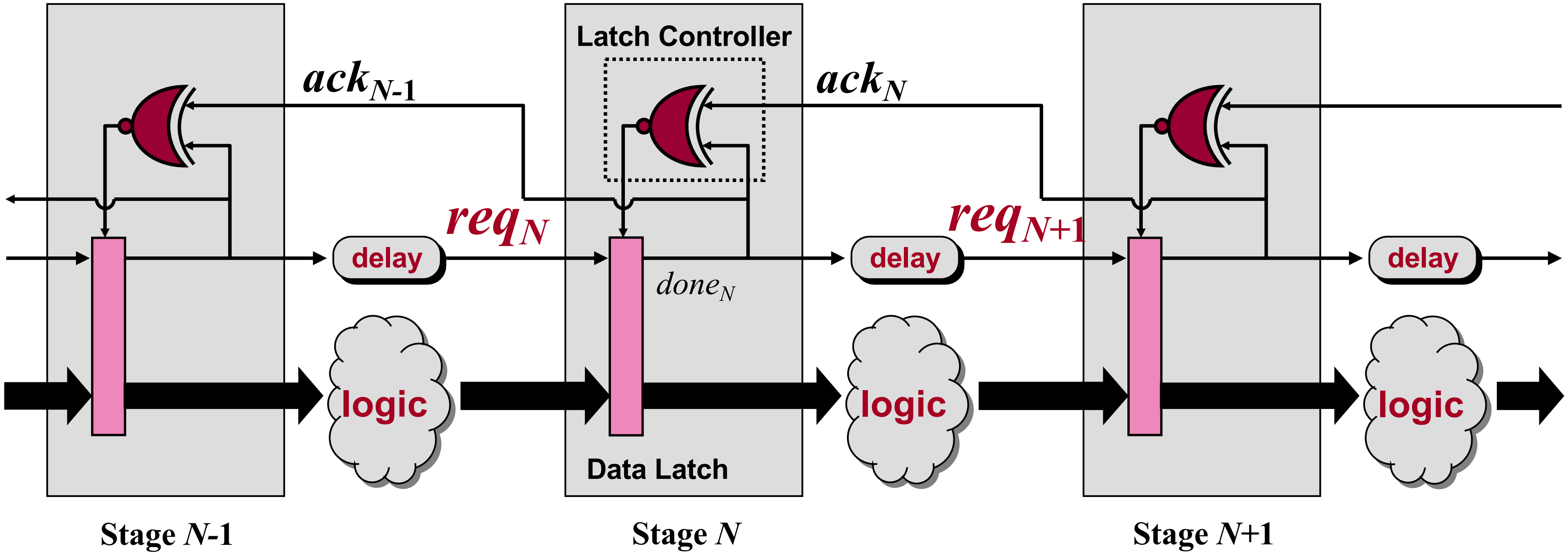
N+1 computes

Cycle Time = $2\,T_{LATCH} + T_{XNOR}$

MOUSETRAP: Pipeline With Logic

Simple Extension to FIFO:

insert *logic block* + *matching delay* in each stage

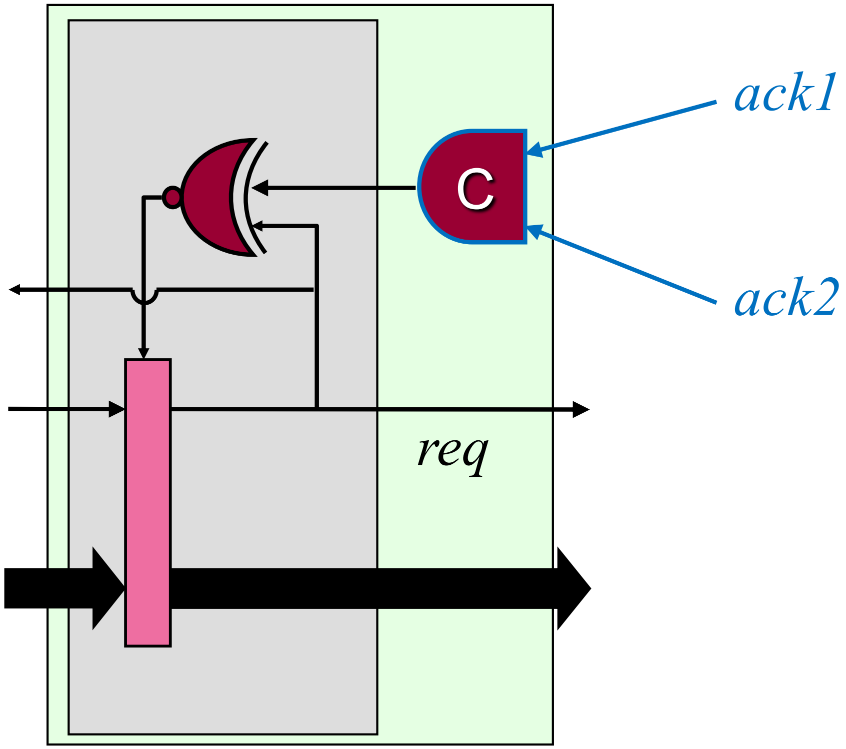


"Bundled Data" Requirement:

- each *"req"* must arrive *after* data inputs valid and stable

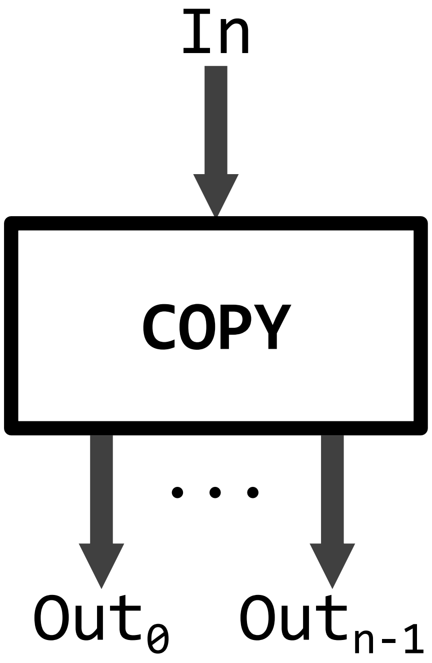
Fork

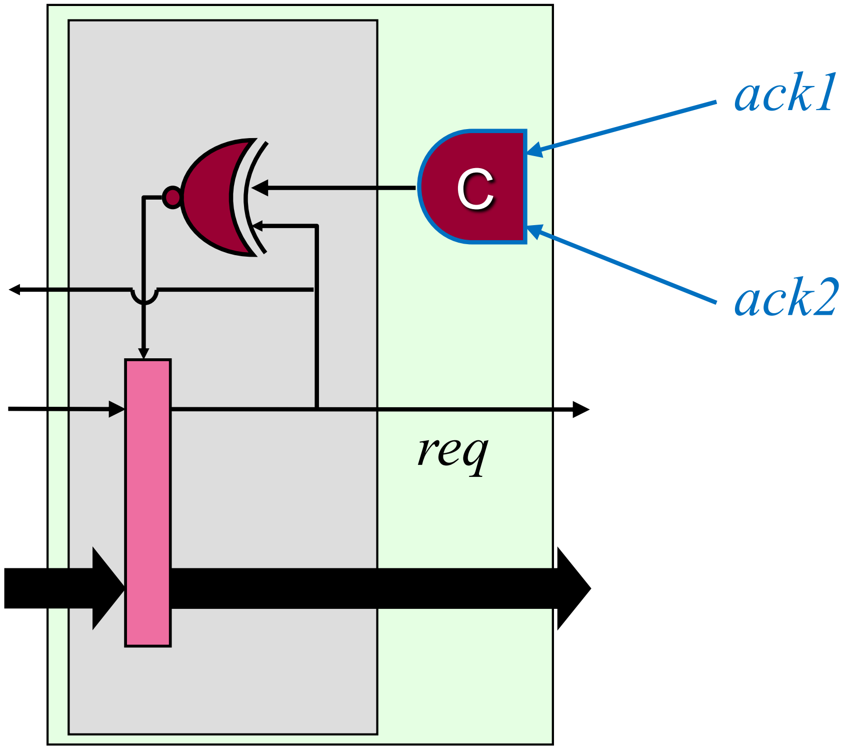
- A fork stage has two (or more) successors
  - same data and req sent to all
  - wait for ack from all



Fork Stage

Fork





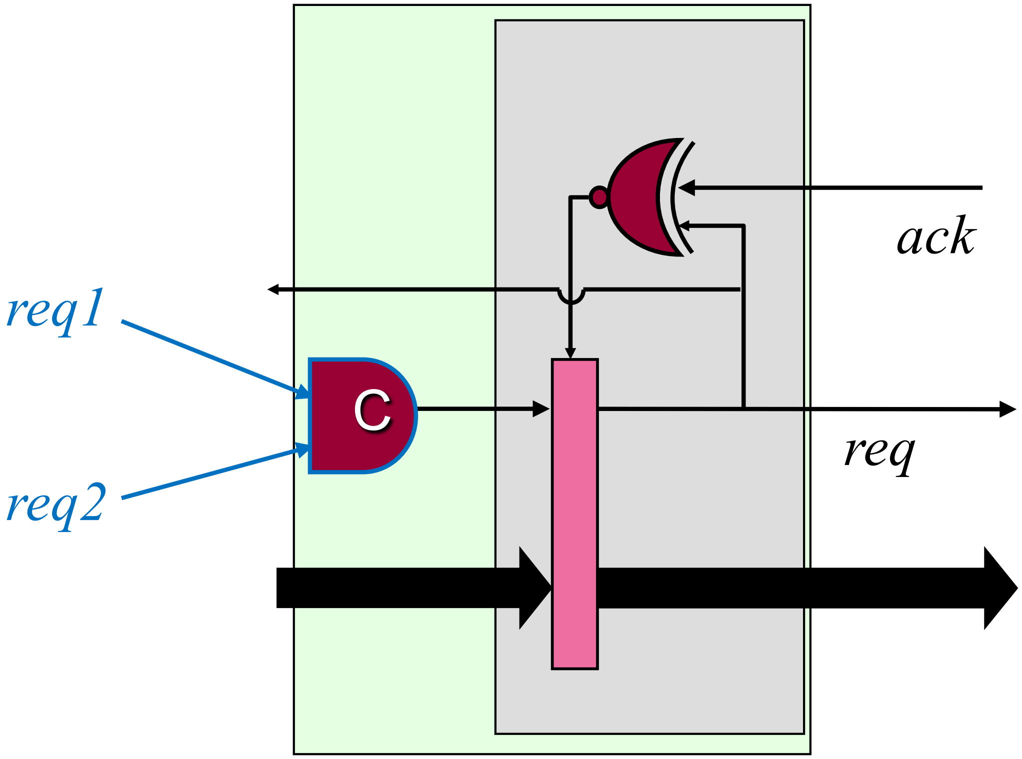
Fork Stage

*[In?x; Out$_0$!x, …, Out$_{n-1}$!x]

In Ben’s talk: COPY, n-way link

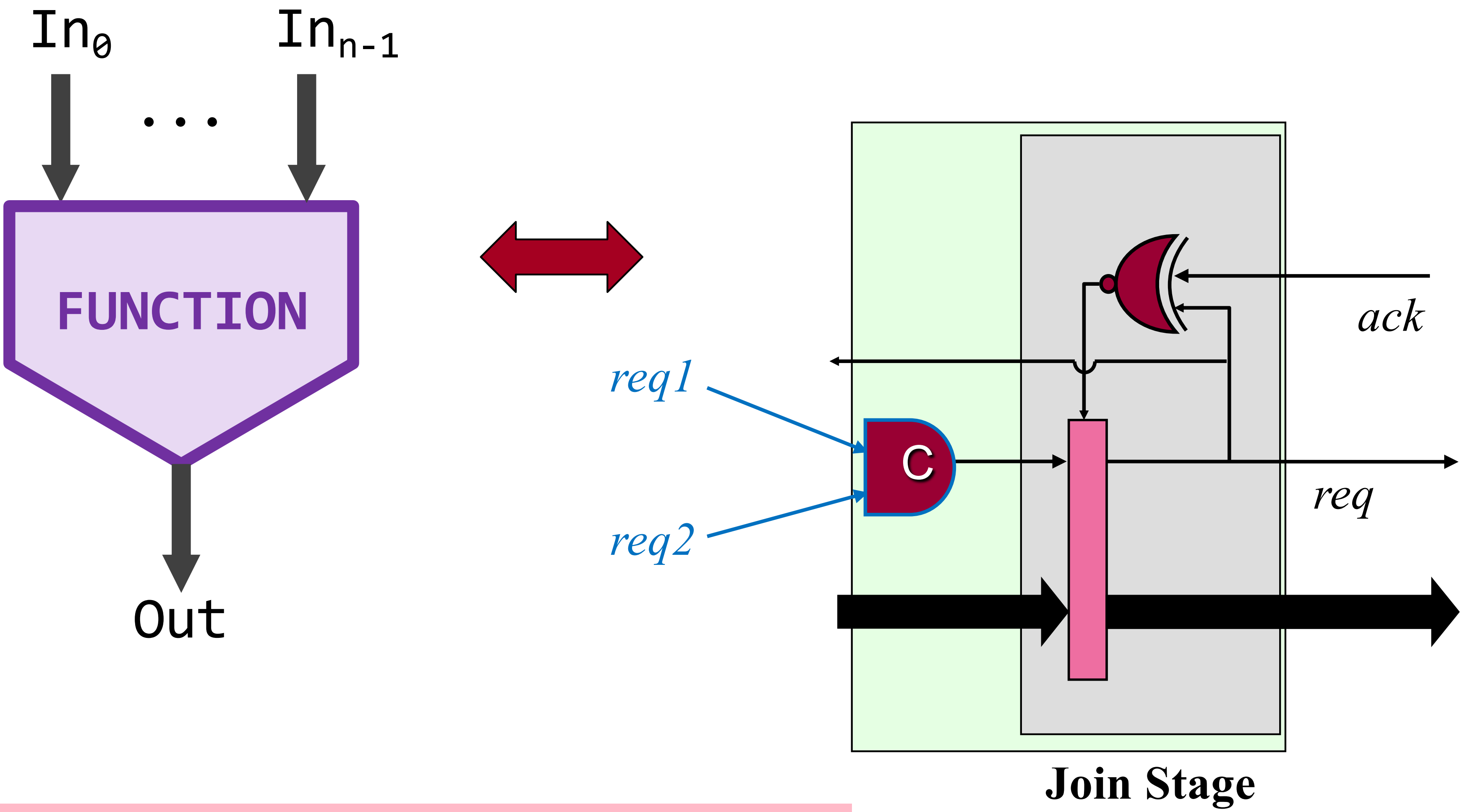
Join

- A join stage has two (or more) predecessors
  - wait for data from all
  - same ack sent to all



Join Stage

Join

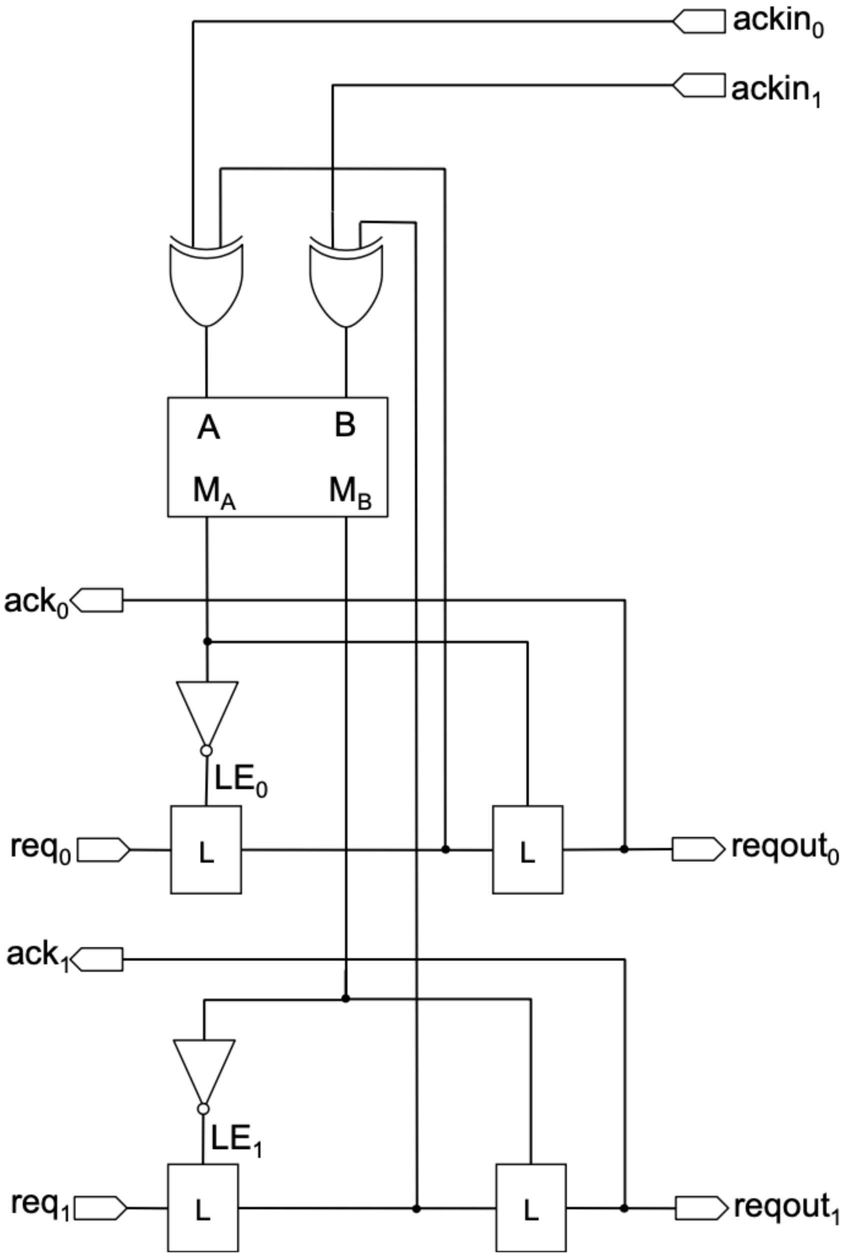


```
*[ In_0?arg_0, In_1?arg_1, … , In_{n-1}?arg_{n-1};
   Out!func(arg_0,arg_1,…,arg_{n-1})
 ]
```

In Ben’s talk: FUNCTION, OPERATOR

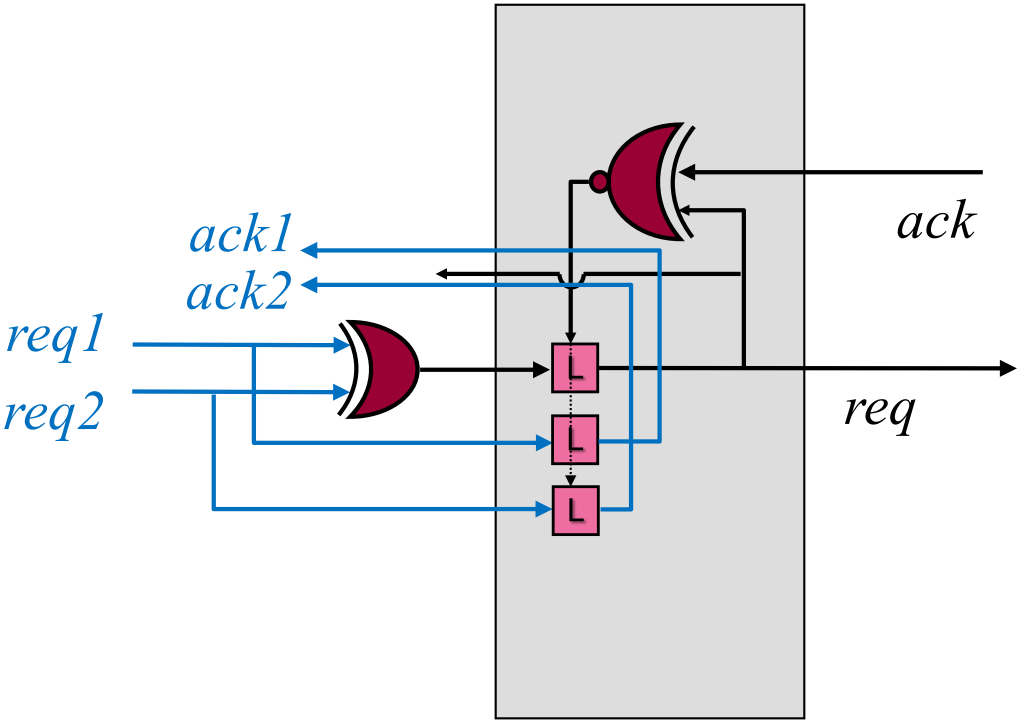
Arbitration Stage

- Two input channels, two output channels
- Only one input read
  - whichever arrives first
  - … goes out on the corresponding output
- Seitz' Mutex is the core
  - [from Brunvand's thesis]
  - surrounding logic adapts it to transition signals



Merge without (or after) Arbitration

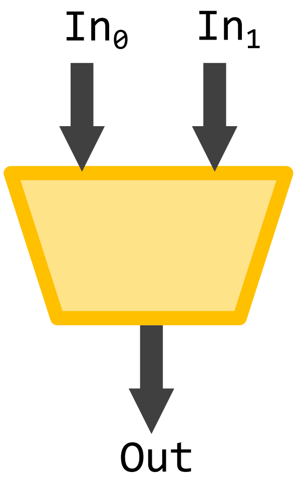
- A merge stage has two (or more) predecessors
  - data is taken from whichever input channel has a new request
- Assumption:
  - no arbitration needed
  - input channels are mutually exclusive



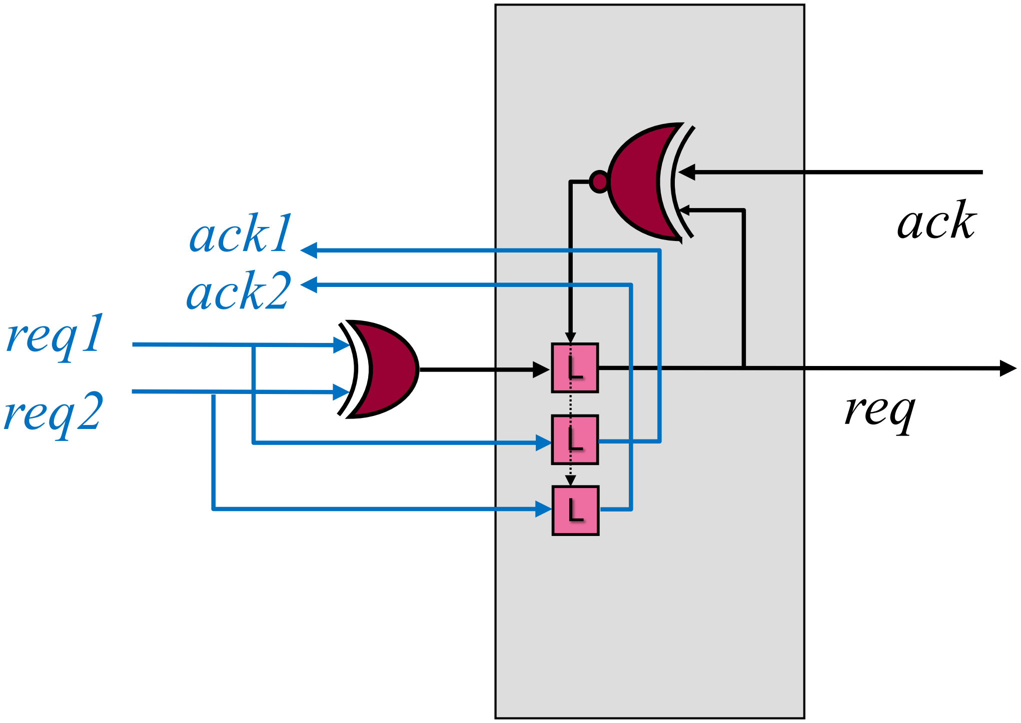
Merge w/o arbitration

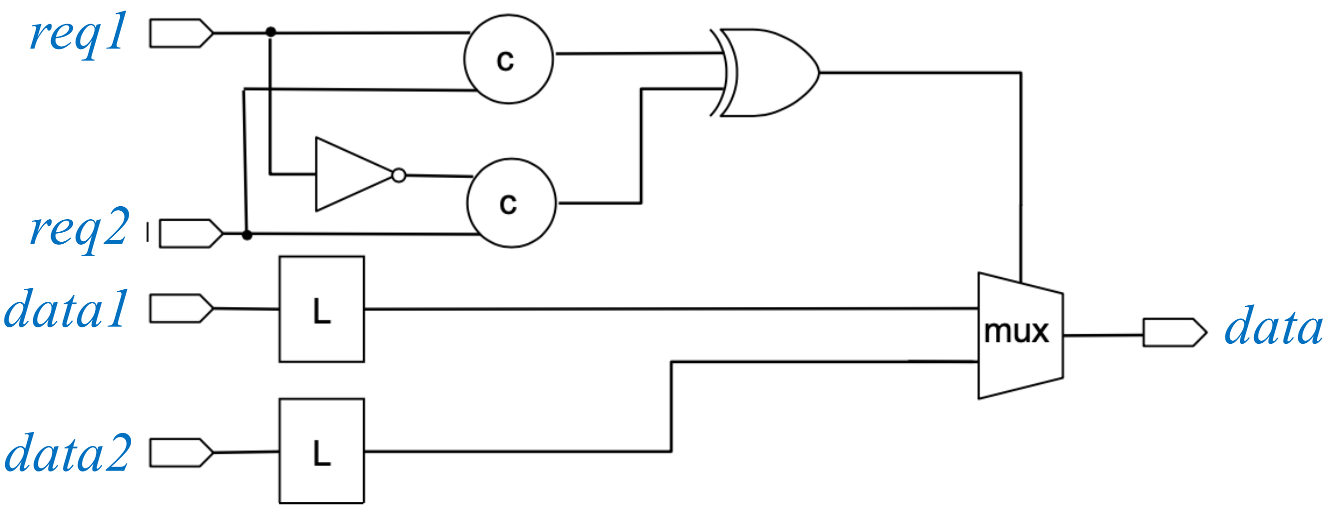
Merge without (or after) Arbitration

- Datapath
  - mux controlled by change detectors on input channels



```
*[ [  #In₀ -> In₀?x
   [] #In₁ -> In₁?x
   ];
   Out!x
 ]
```

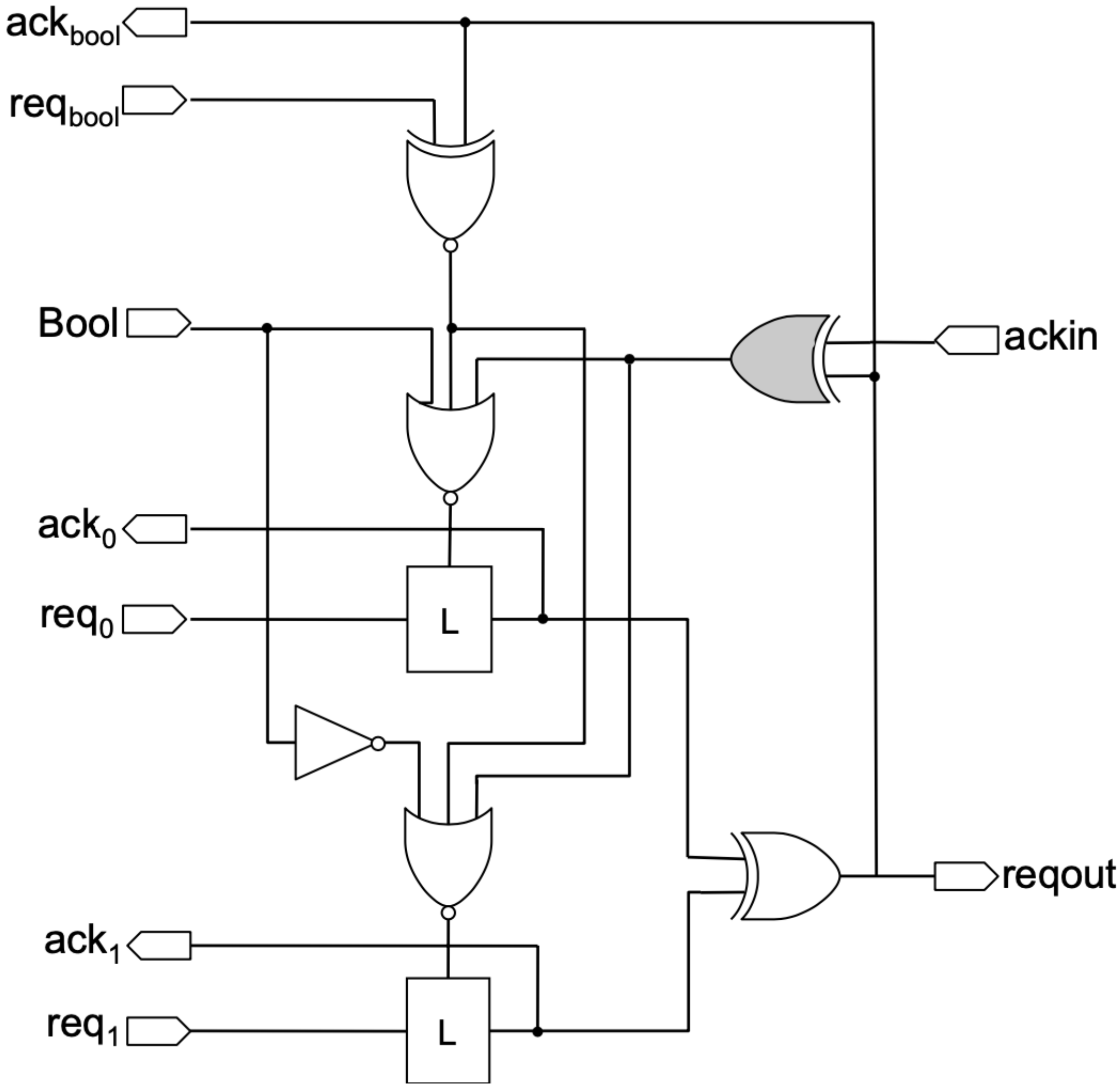




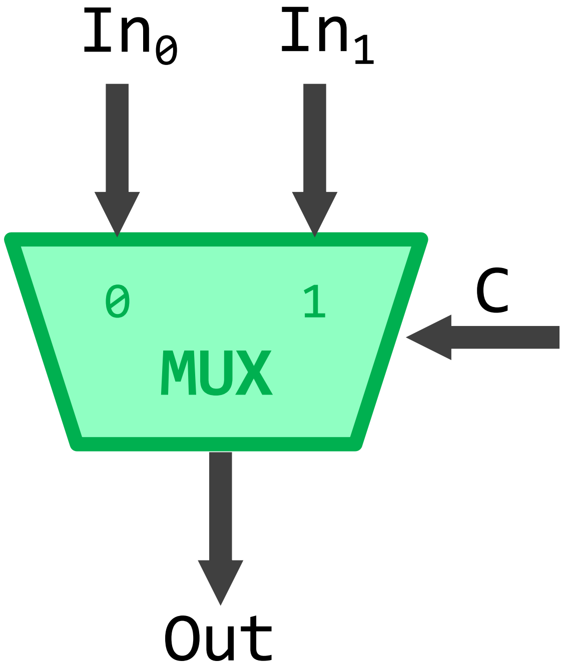
In Ben’s talk: MERGE, MIXER

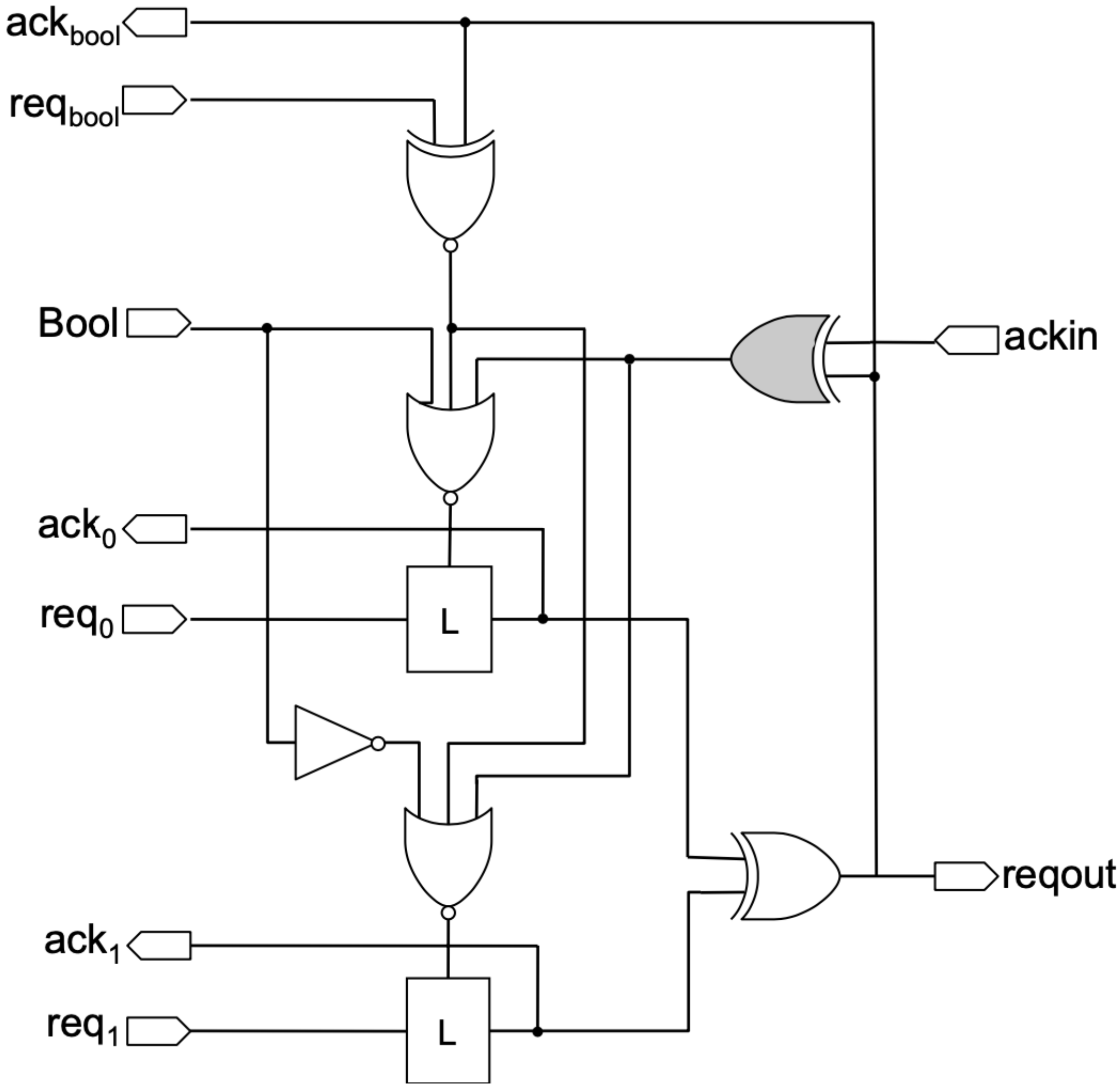
Conditional Select (or event mux)

- Two data inputs and one select
  - first read select
  - then:
    - based on select, read one of the inputs
    - do not read the other input
  - datapath
    - mux + latch



Conditional Select (or event mux)



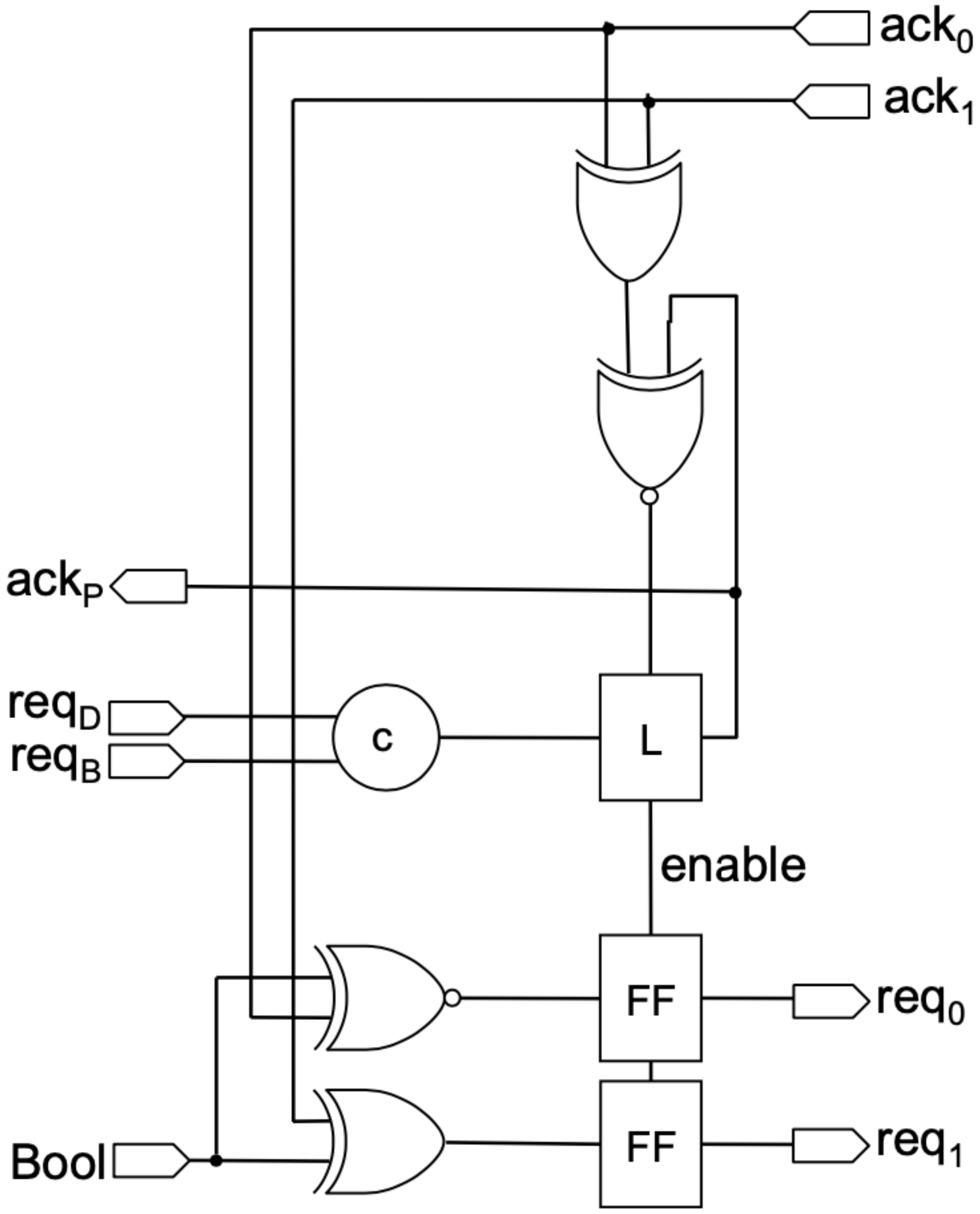


```
*[C?c;
   [  c=0 -> In₀?x
   [] c=1 -> In₁?x
   ];
  Out!x
 ]
```

In Ben's talk: MUX, CONTROLLED MERGE

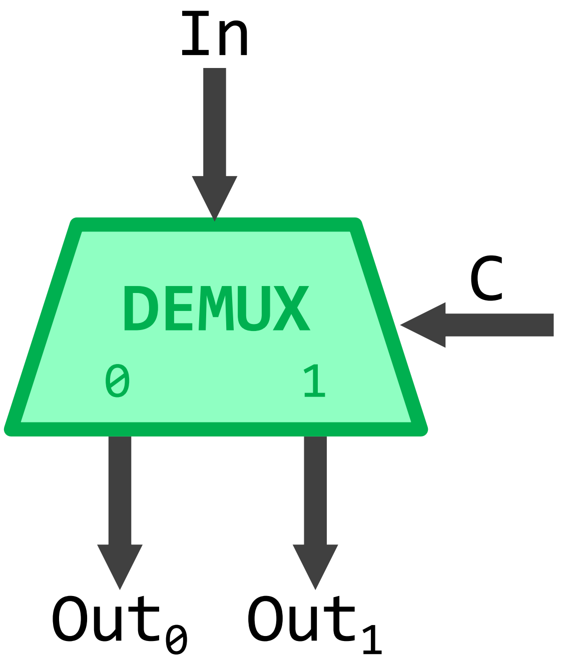
Conditional Split (or router)

- One data input and one select
  - first read data + select
  - then:
    - based on select, send data along one output channel

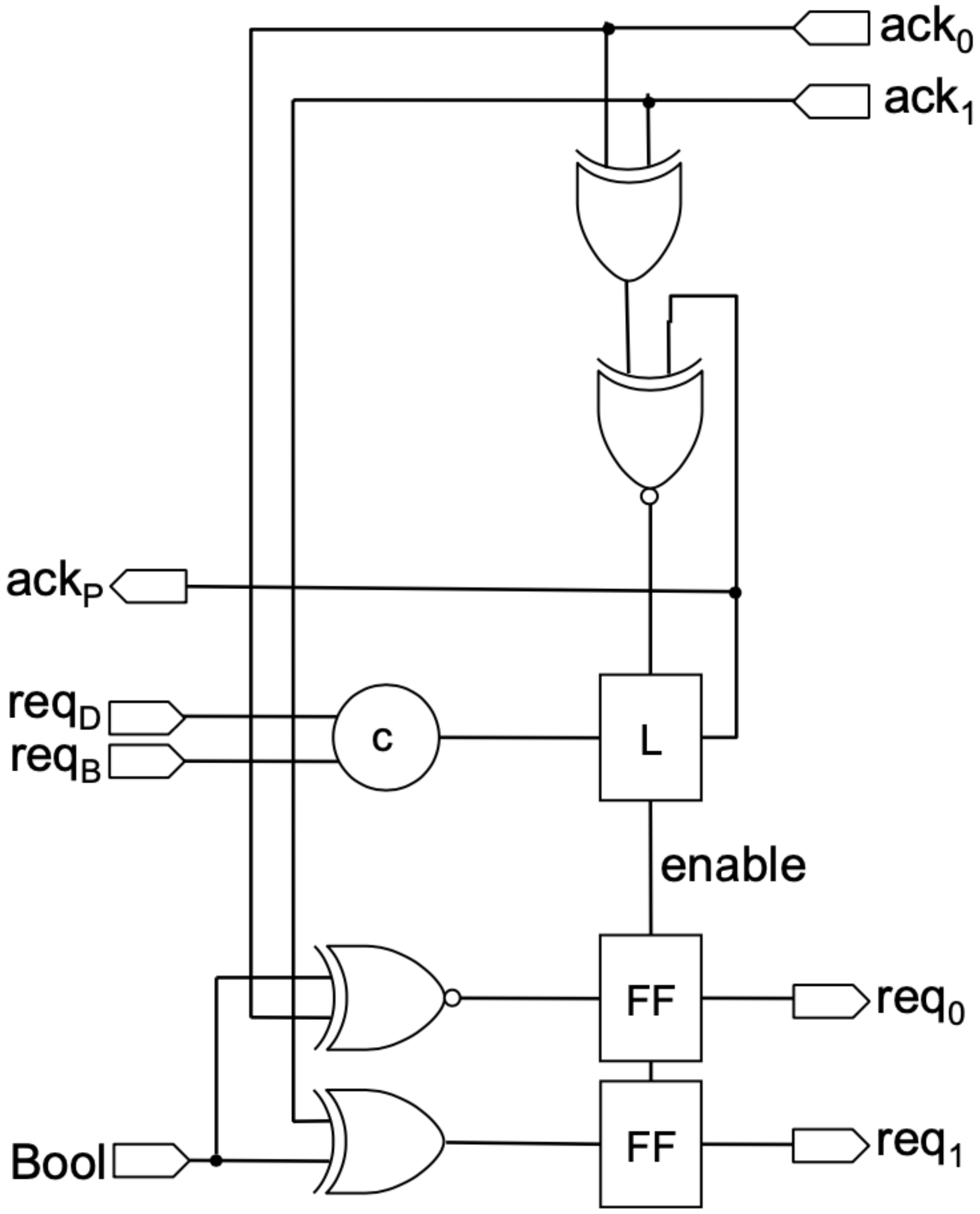


In Ben’s talk: DEMUX, SPLIT

Conditional Split (or router)



```
*[In?x, C?c;
   [  c=0 -> Out0!x
   [] c=1 -> Out1!x
   ]
 ]
```



In Ben’s talk: DEMUX, SPLIT

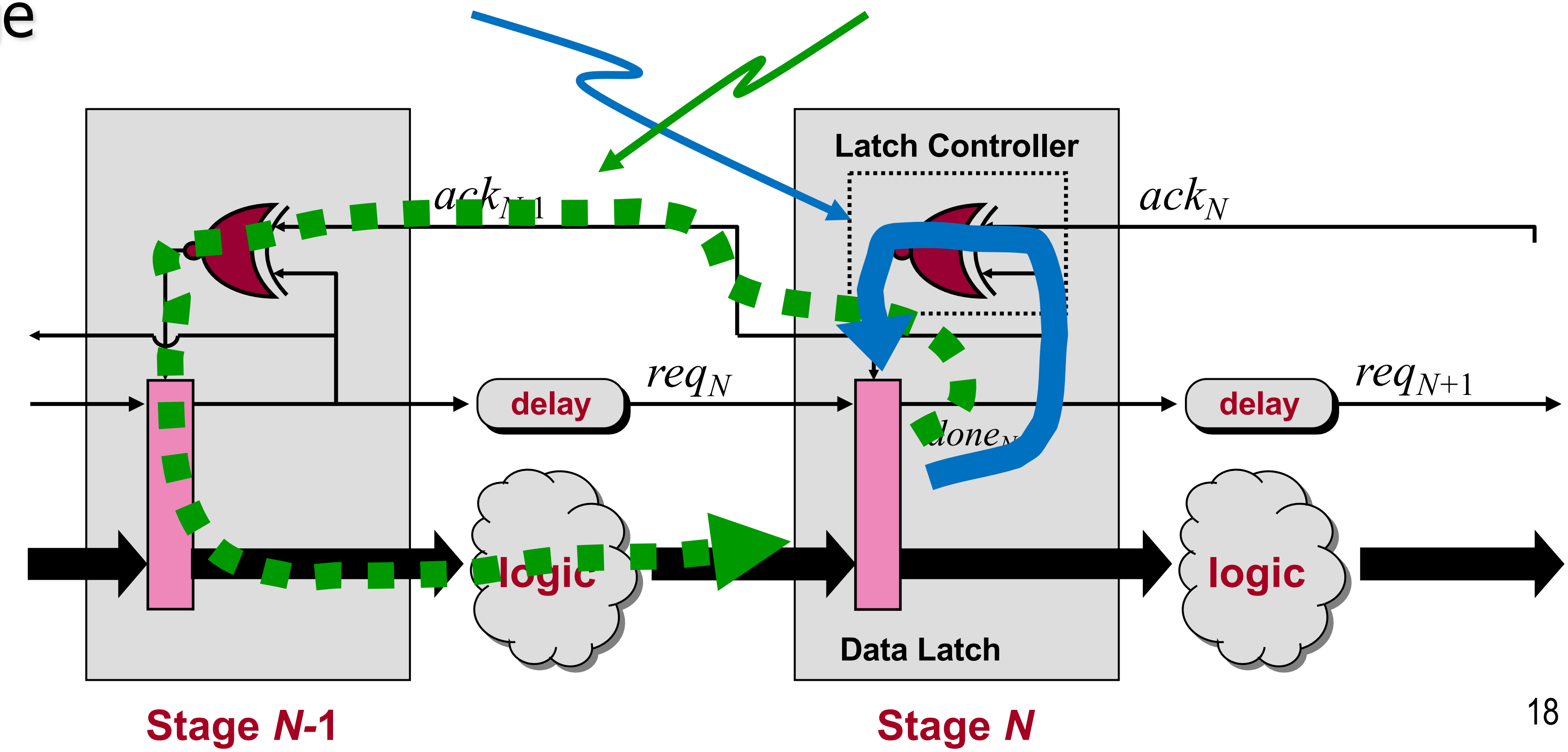
Timing Analysis

Setup constraint: matched delay

Hold Time constraint:

Data must be safely "captured" by Stage N before new inputs arrive from Stage N-1

- Stage N's "self-loop" faster than entire path through previous stage



Example: Greatest Common Divider

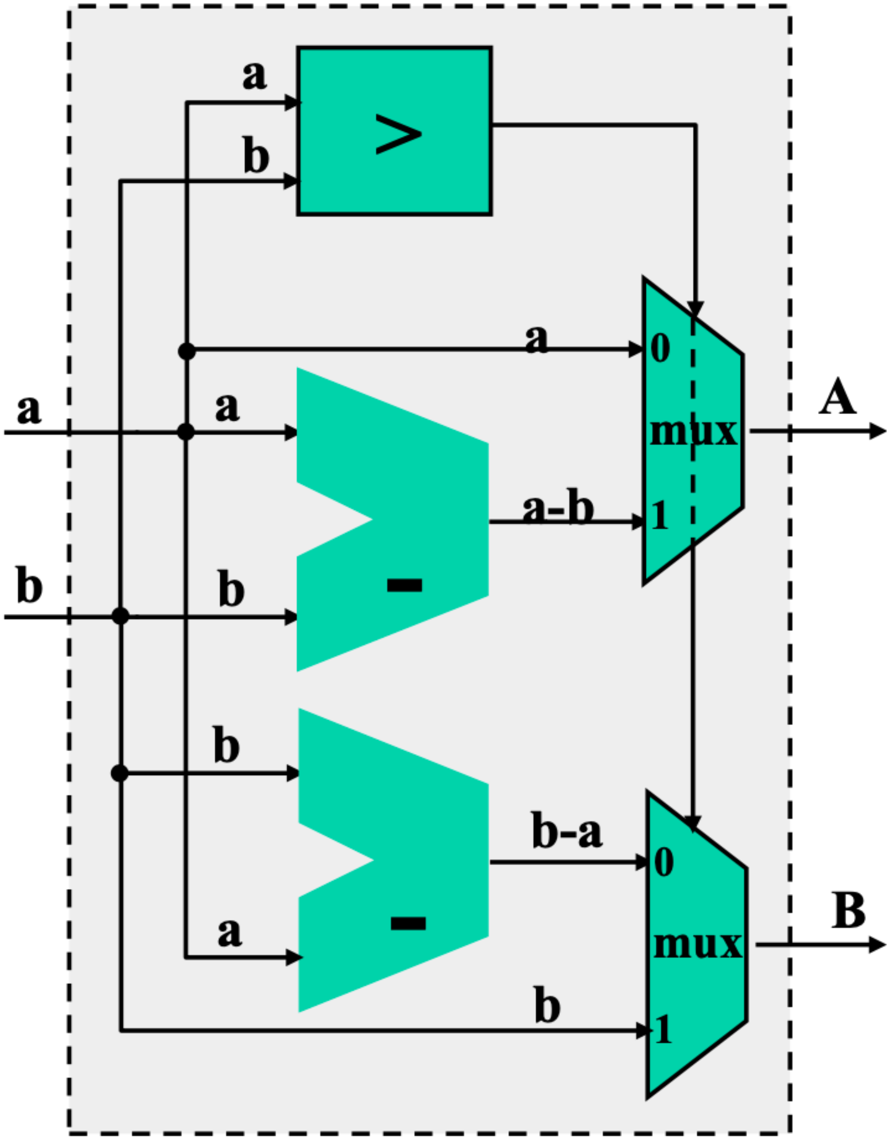
Euclid’s GCD algorithm

```
gcd(a, b)
      while (b != 0)
            if(a>b)
                  a = a – b
            else
                  b = b – a
      return a
```

- Example
  - gcd(42, 28)
  - (14, 28)
  - (14, 14)
  - (14, 0)
  - → 14

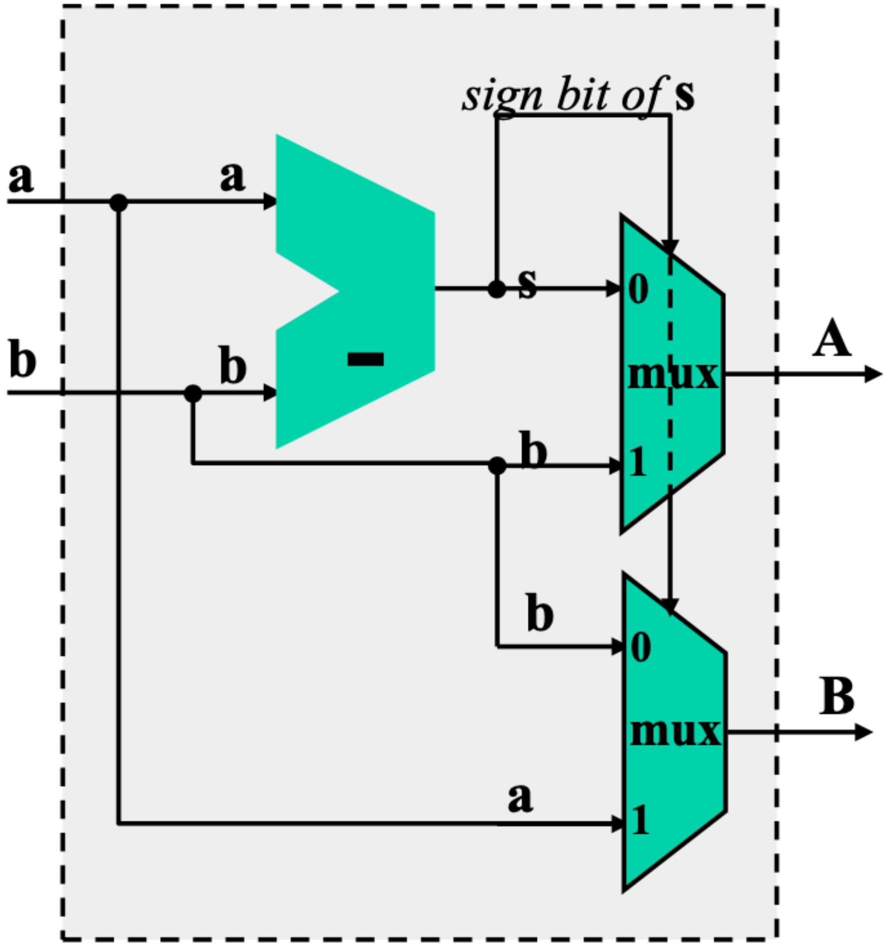
Euclid’s GCD algorithm

```
gcd(a, b)
        while (b != 0)
                if(a>b)
                        a = a – b
                else
                        b = b – a
        return a
```



Better area version

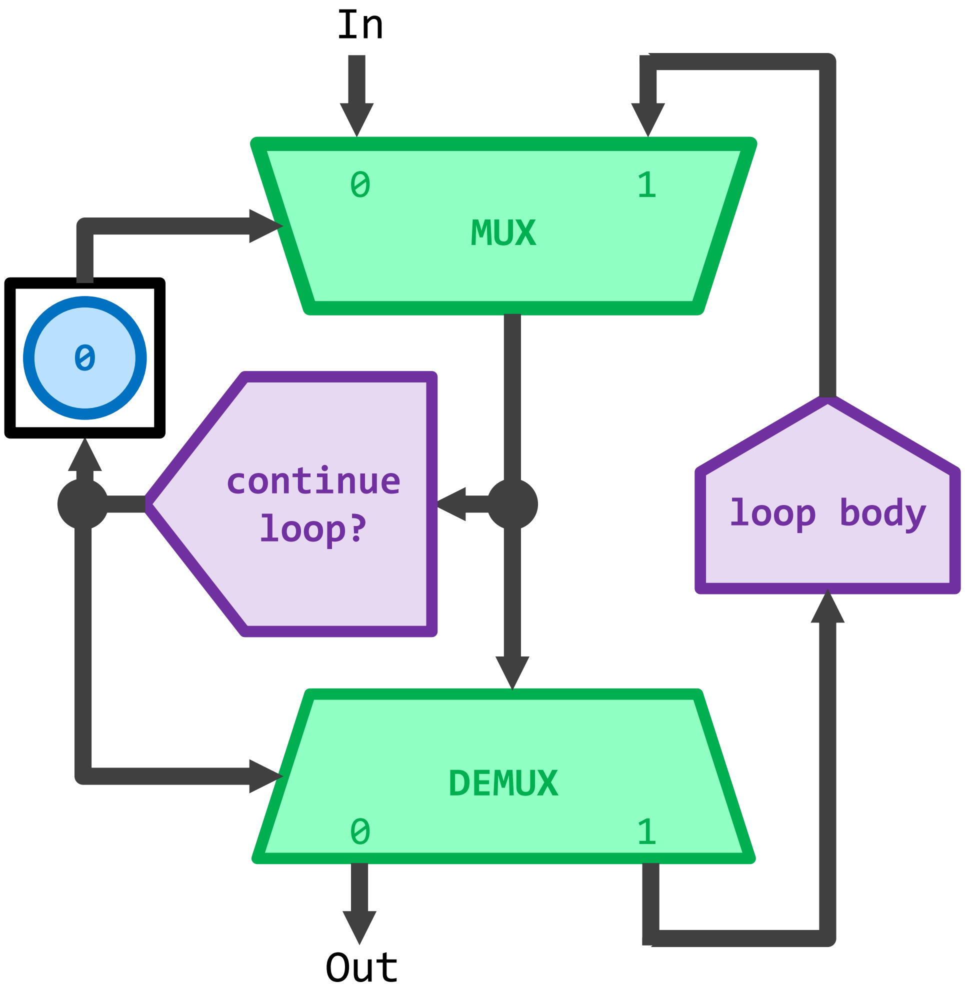
```
gcd(a, b)
      while (b != 0)
            s = a-b
            if(s<0)
                  swap(a,b)
            else
                  a = s
      return a
```



- Example
  - gcd(42, 28)
  - (14, 28)
  - (28, 14)
  - (14, 14)
  - (0, 14)
  - (14, 0)
  - ➔ 14

Dataflow: WHILE loop structure

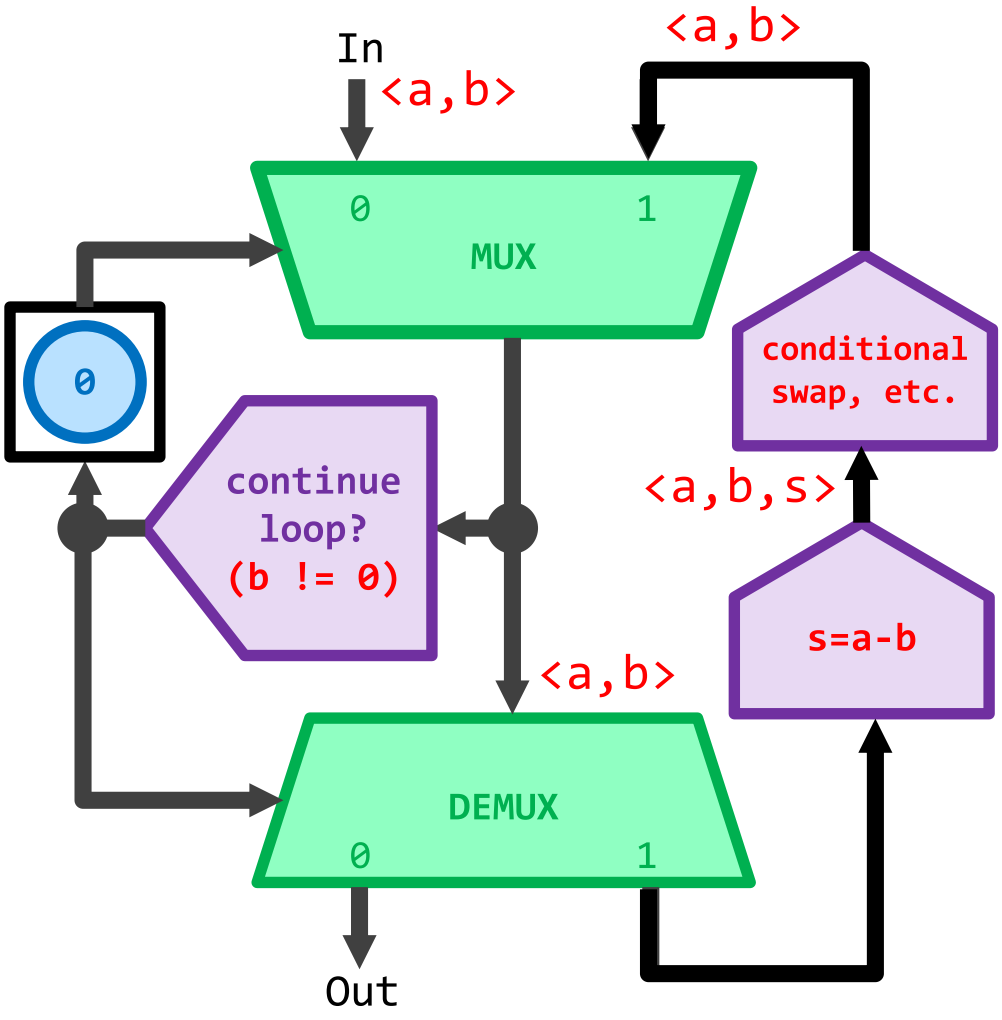
```
while (b != 0)
        s = a-b
        if(s<0)
                swap(a,b)
        else
                a = s
return a
```



From Ben’s talk Week 1

GCD implementation

```
while (b != 0)
        s = a-b
        if(s<0)
                swap(a,b)
        else
                a = s
return a
```



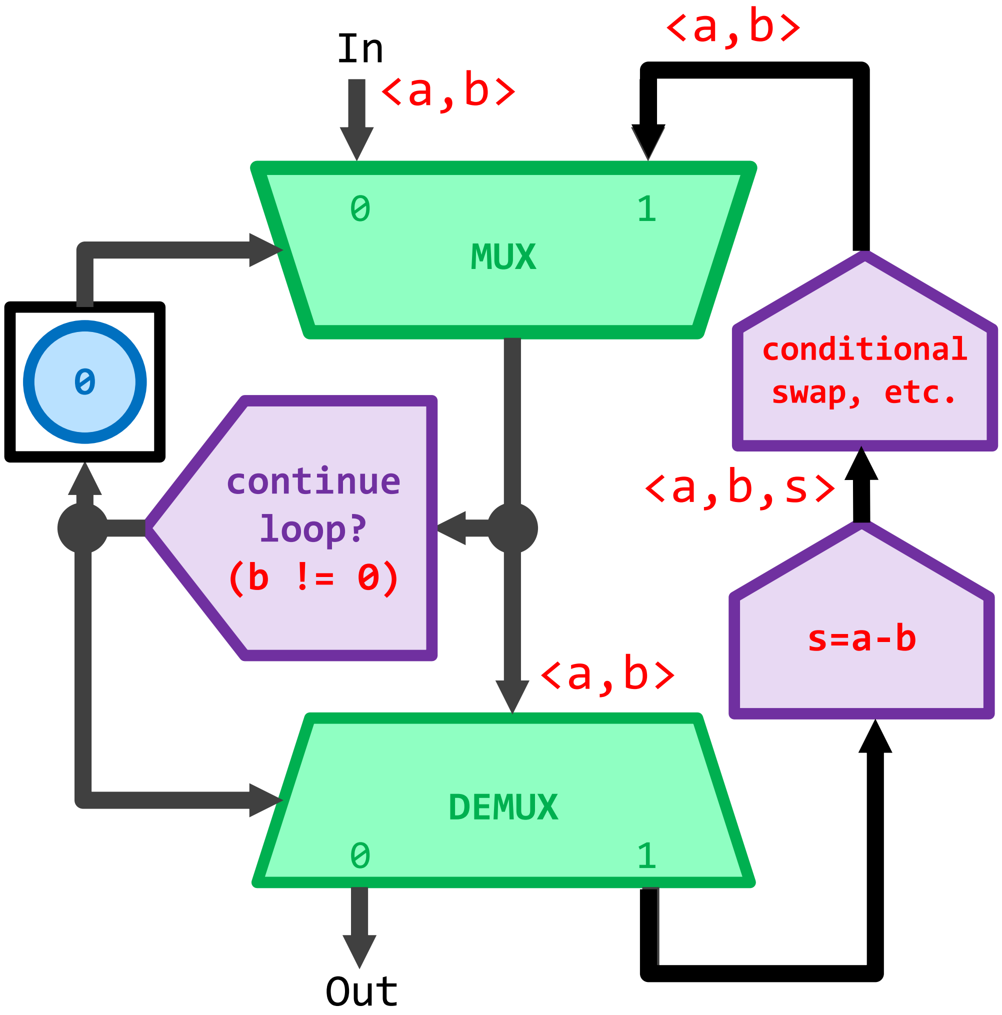
GCD implementation

```
while (b != 0)
       s = a-b
       if(s<0)
              swap(a,b)
       else
              a = s
return a
```

Unroll 8 times!

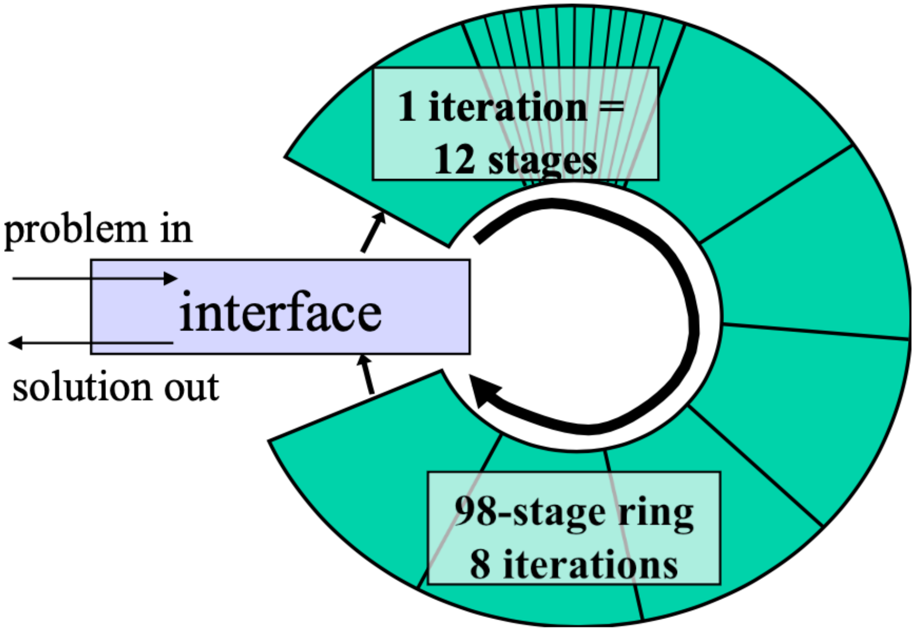
Pipeline subtract
into 8 stages

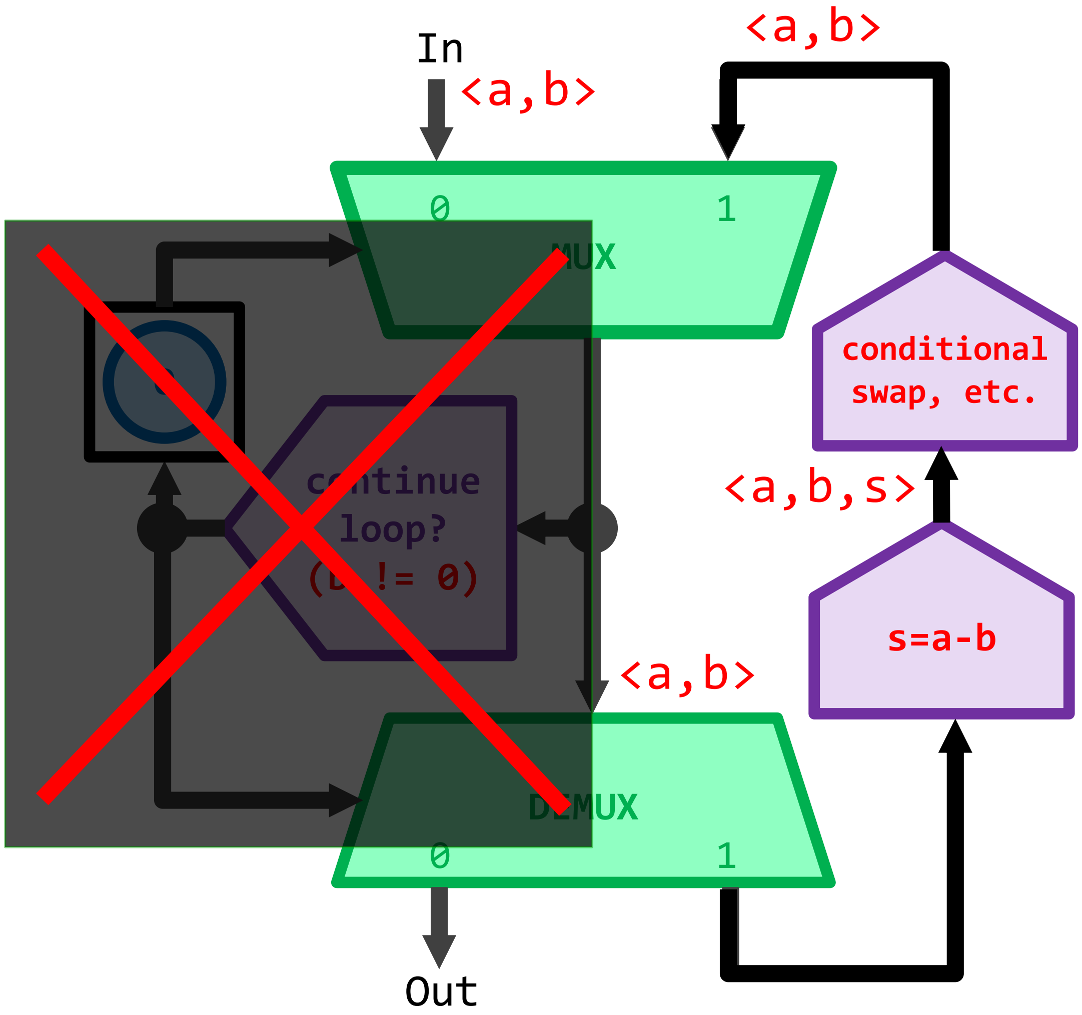
Pipeline swap
into 4 stage



GCD implementation

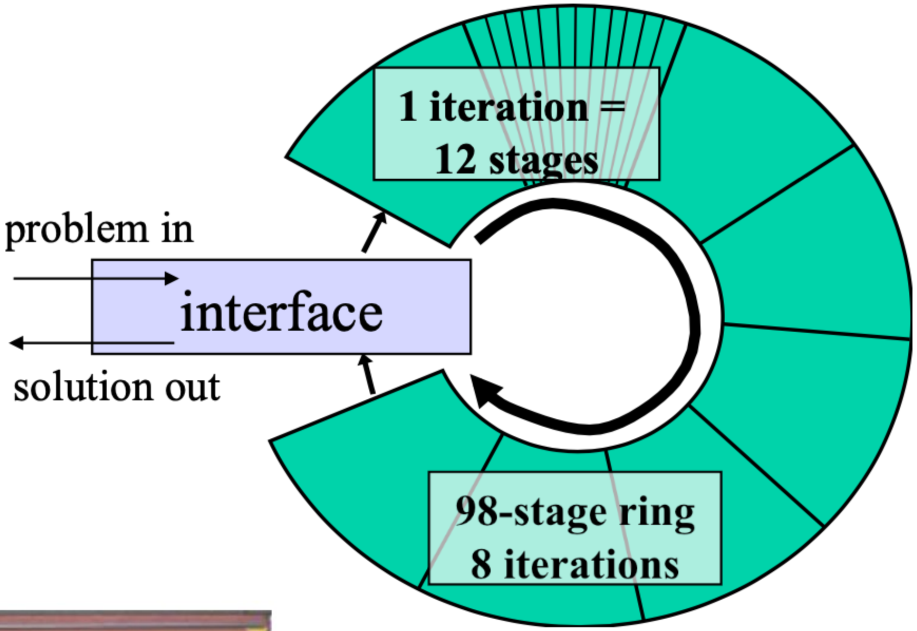
```
while (b != 0)
        s = a-b
        if(s<0)
                swap(a,b)
        else
                a = s
return a
```

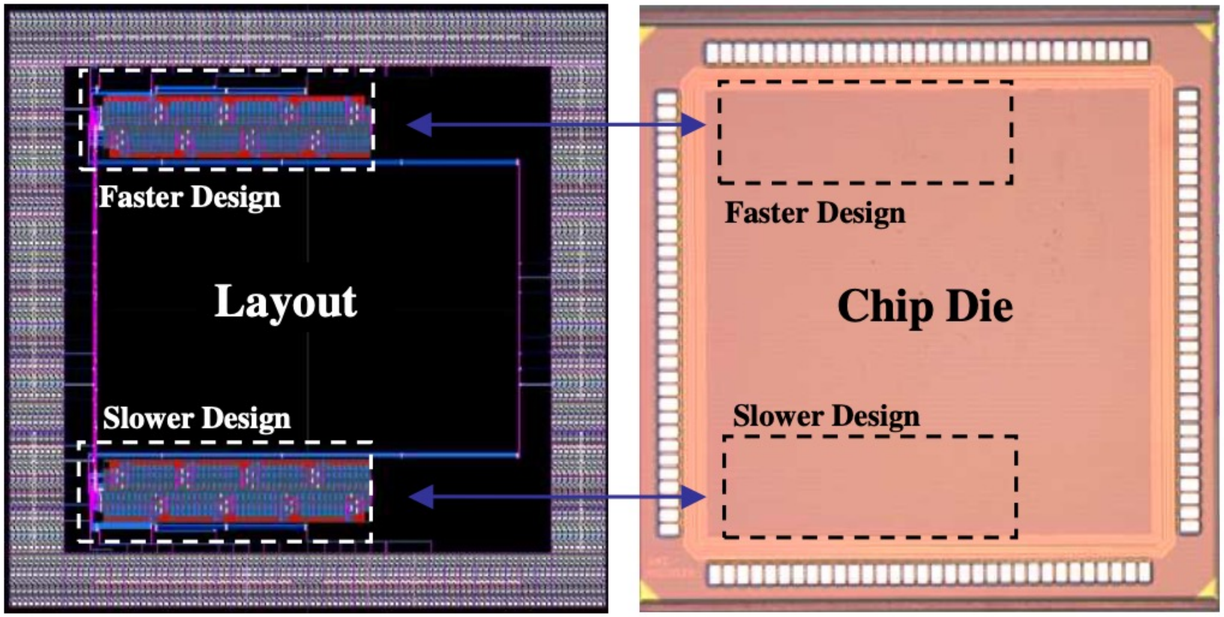




GCD chip

- Layout and fab:
  - 0.13um, standard cell

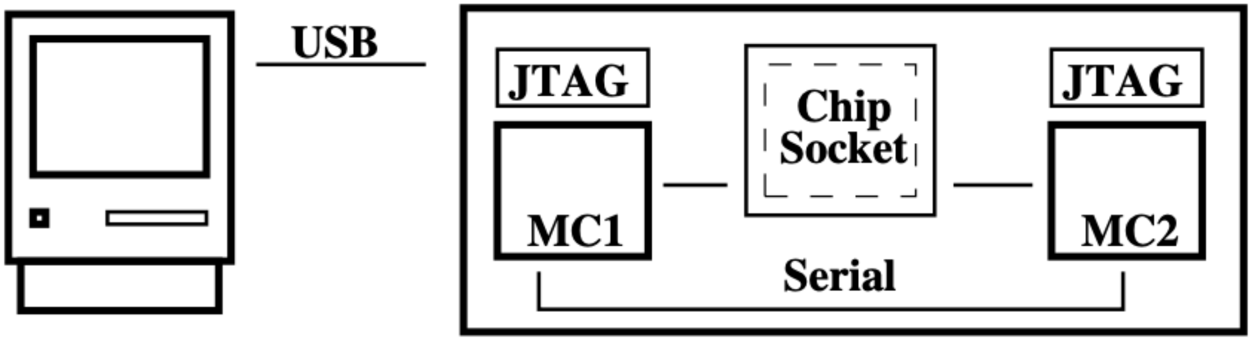




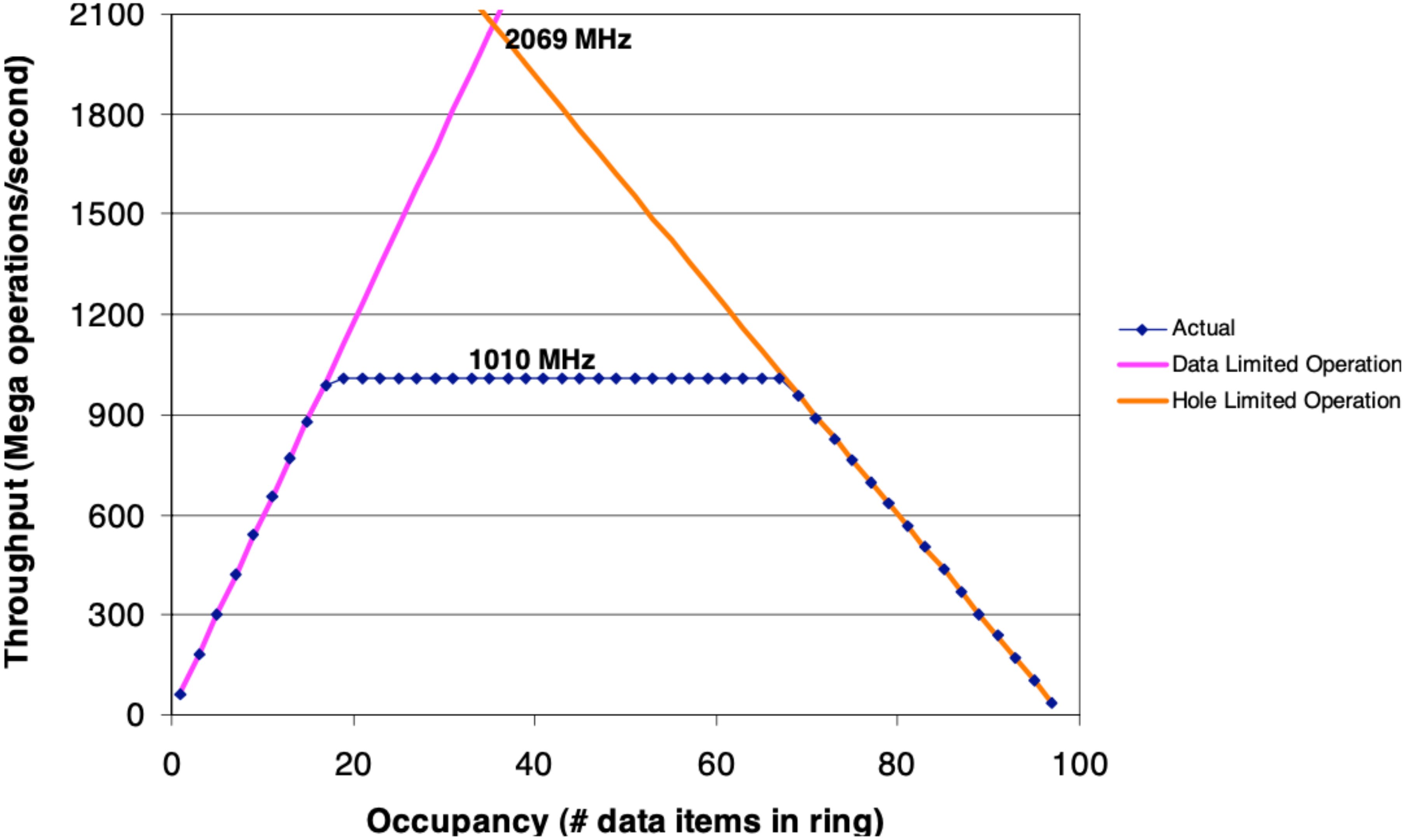
GCD chip

- Testing:
  - Full scan for latches
  - Combinational test patterns generated (Shi 2005) gave 98% coverage
  - Functional tests for timing violations (Gill 2006)

- Evaluation:

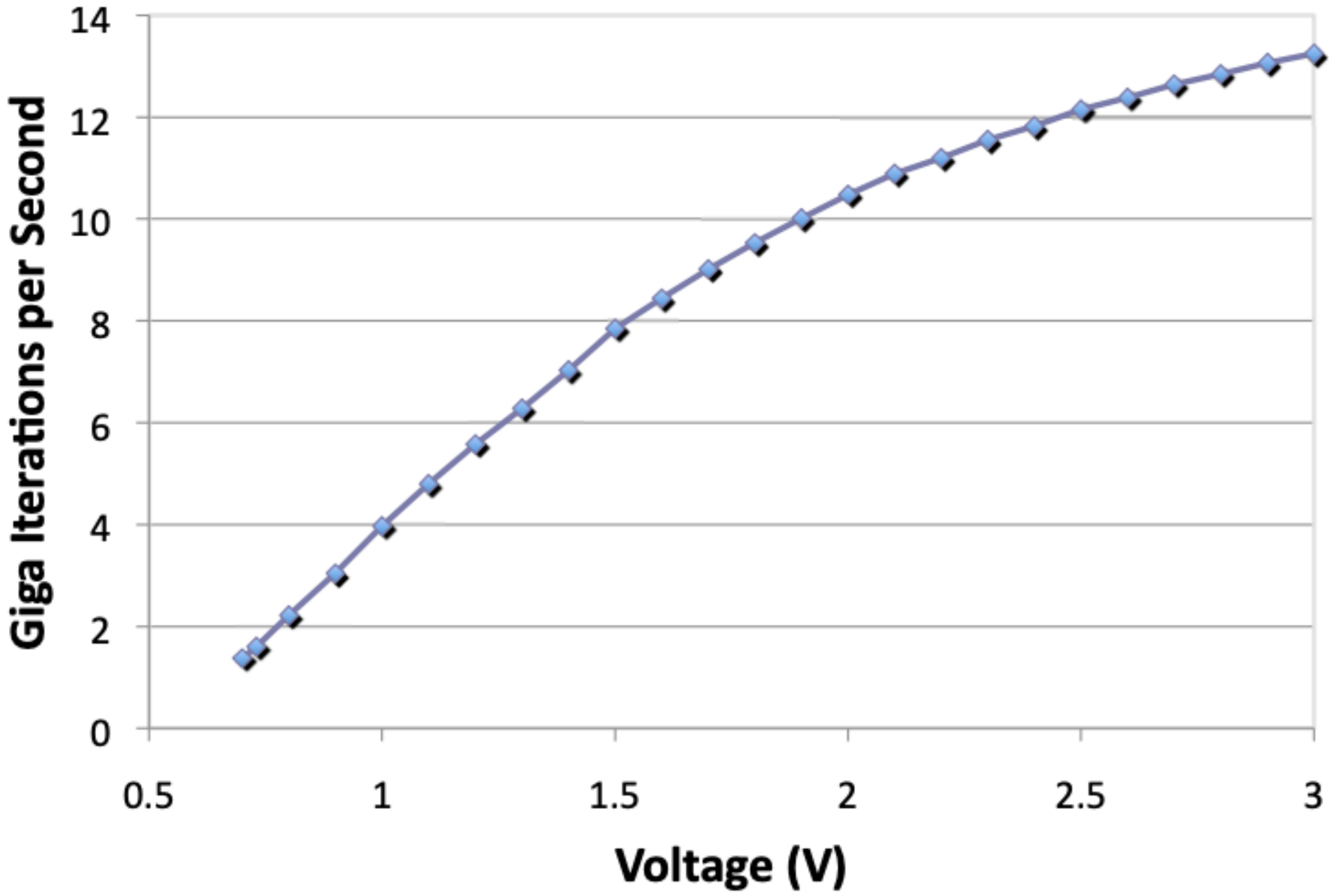


GCD chip: Results



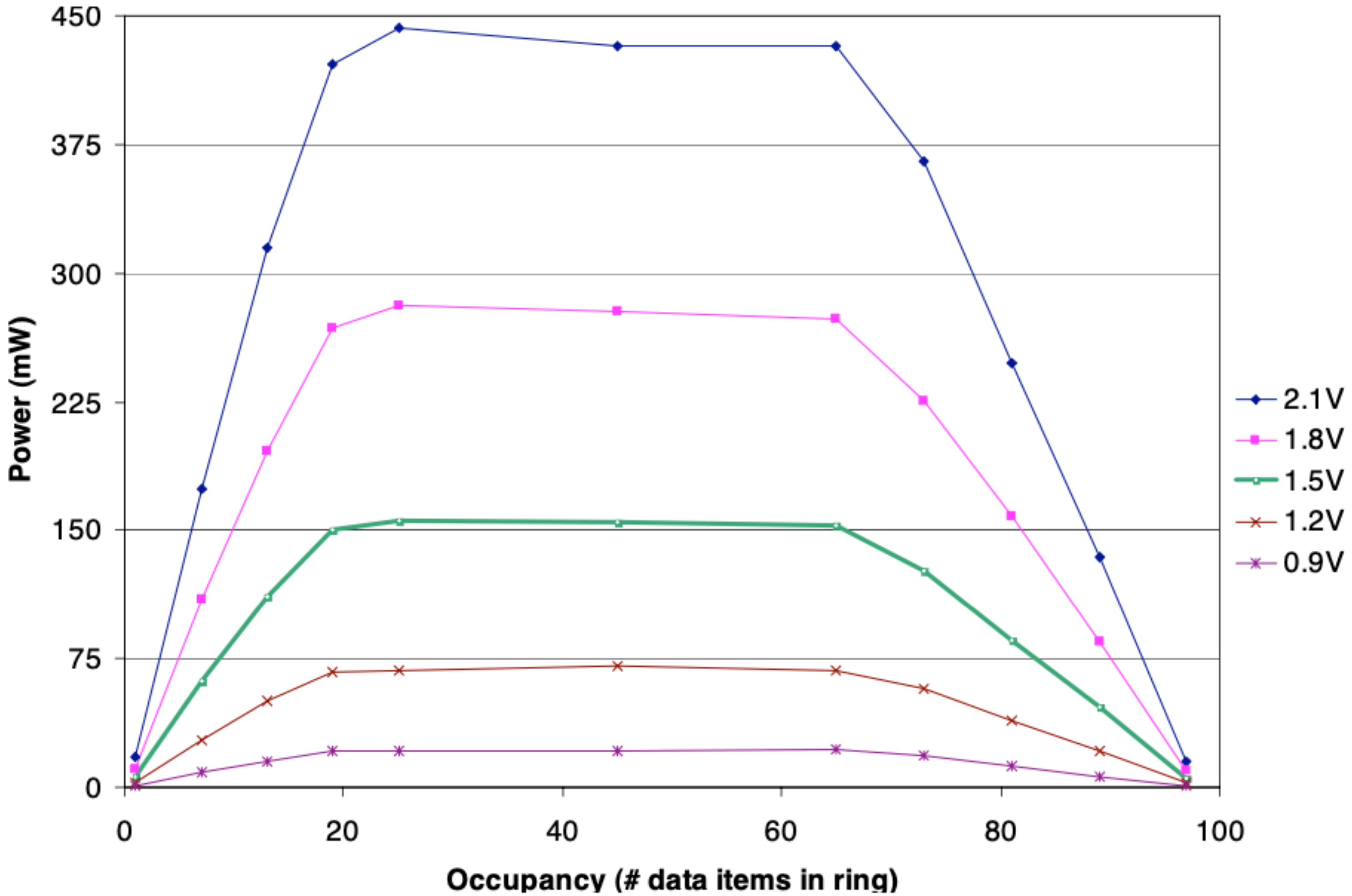
GCD chip: Results

- Operates correctly over a wide voltage range



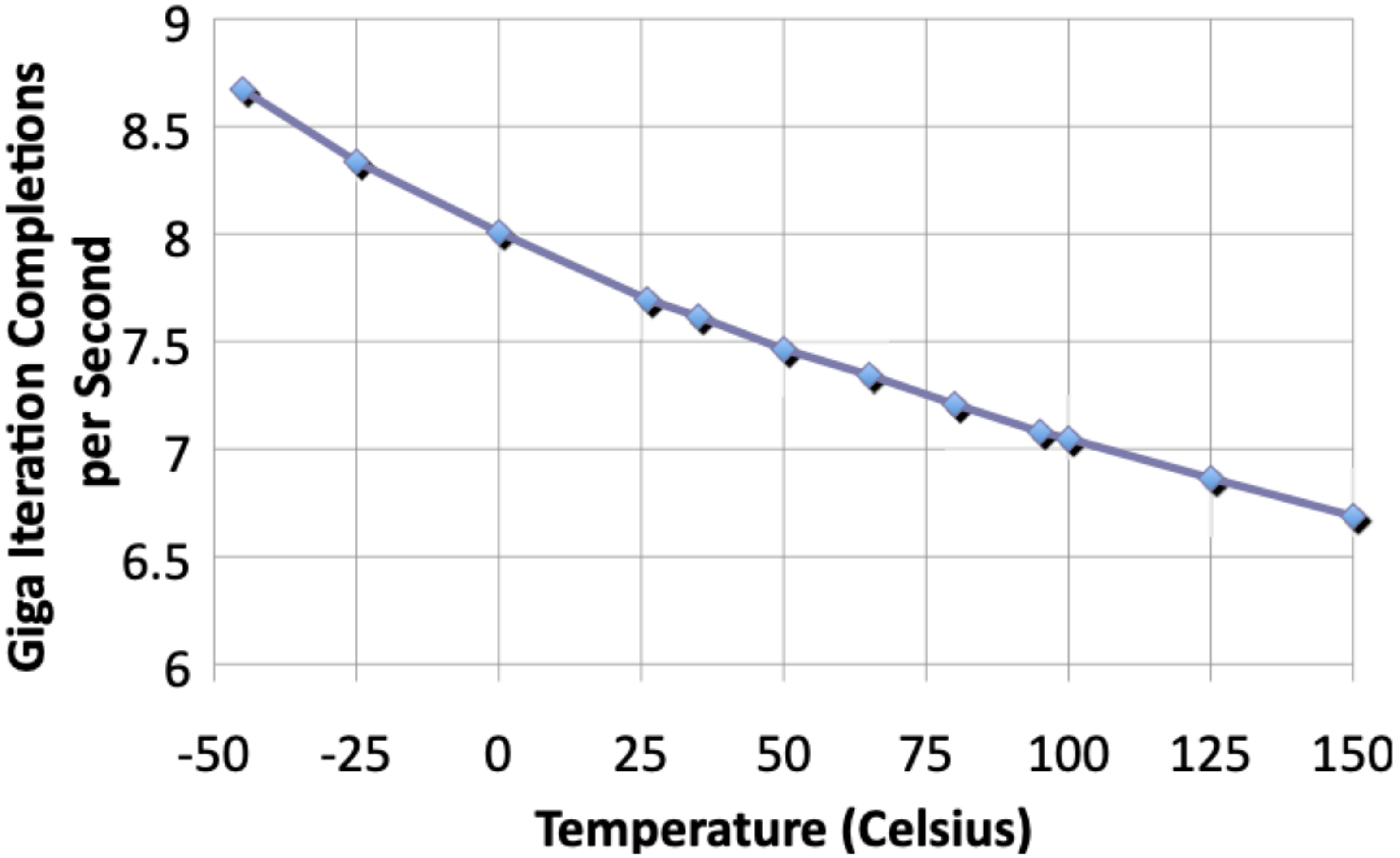
GCD chip: Results

- Impact of voltage variation on power consumption



GCD chip: Results

- Impact of temperature performance



References

- Feng Shi, Yiorgos Makris, Steven M. Nowick, and Montek Singh. *"Test generation for ultra-high-speed asynchronous pipelines."* ITC 2005.

- Gennette Gill. Analysis and Optimization for Pipelined Asynchronous Systems. PhD thesis. UNC Chapel Hill. 2010.

- Gennette Gill, A. Agiwal, M. Singh, F. Shi, Y. Makris, *"Low-Overhead Testing of Delay Faults in High-Speed Asynchronous Pipelines."* ASYNC 2006.

- Montek Singh and Steven Nowick. *"MOUSETRAP: Ultra-High-Speed Transition-Signaling Asynchronous Pipelines."* ICCD 2001.

- Montek Singh and Steven Nowick. *"MOUSETRAP: High-Speed Transition-Signaling Asynchronous Pipelines."* TVLSI 2007.

- Gennette Gill, J. Hansen, A. Agiwal, L. Vicci and M, Singh. *"A High-Speed GCD Chip: A Case Study in Asynchronous Design."* ISVLSI 2009.