EE273 Lecture 15
Asynchronous Design
November 16, 1998

William J. Dally
Computer Systems Laboratory
Stanford University
billd@csl.stanford.edu

Today's Assignment

- Term Project
  - see project update handout on web
  - checkpoint 1 (signaling components) due Wednesday (11/18)
  - checkpoint 2 (timing components) due on 11/25
- Reading
  - no new reading, complete Section 10.4 if you haven't already done so
- FYI
  - meso- means middle
    - timing midway between sync and async?
  - plesio- means near
    - nearly mesochronous?

A Quick Overview

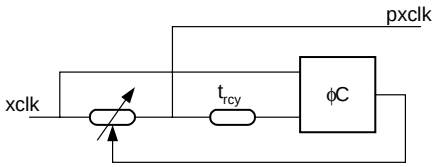
- Periodic Synchronizers
  - clock prediction - looking into the future
- Asynchronous design
  - most synchronization is unnecessary
  - can be avoided by using asynchronous design
- A stoppable clock
  - stop clock between *events*
  - start clock when an input event occurs
- Combinational asynchronous modules
  - inputs and outputs encode *events* not just values
  - modules obey the weak conditions
    - number of events on each output is equal to or one less than the number of events on each input
- Composition
  - weak conditions are closed under acyclic composition

Periodic Timing

- Transmit and receive clocks are periodic but at unrelated frequencies
  - e.g., modules in a system operate off of separate oscillators with independent frequencies
  - case where one is rationally derived from the other is an interesting special case
- In this situation, a single synchronization won't last forever (like mesochronous) or even for a long time (like plesiochronous)
- However, we can still look into the future and predict clock conflicts far enough ahead to reduce synchronizer delay

Clock-Predictor Circuit

- Suppose we want to know the value of xclk, one rclk cycle (t_{rcy}) in the future
- This is just a phase shift of t_{xcy}-t_{rcy}
- It is easy to generate this phase shift using a simple timing loop
- Note that we could just as easily predict xclk several rclk cycles in the future
- So how do we build a synchronizer using this?



Asynchronous Timing

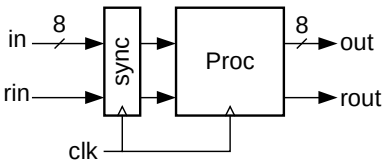
- Sometimes we need to sample a signal that is truly asynchronous
- We can still move the synchronization out of the datapath by using an asynchronous FIFO synchronizer
- However this still incurs a high latency on the full and empty signals as we have to wait for a brute force synchronizer to make its decision
- We can still avoid delay in this case if we don't *really* need to synchronize
  - often synchronization is just an expensive convenience

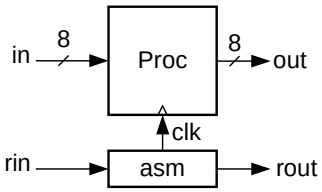
Asynchronous Design

- Most often we synchronize just to align an event to a clock
  - it doesn't really matter if we handle the event on clock i or i+1
- We can avoid this unnecessary synchronization by processing events asynchronously
- The arrival of an event triggers its handling

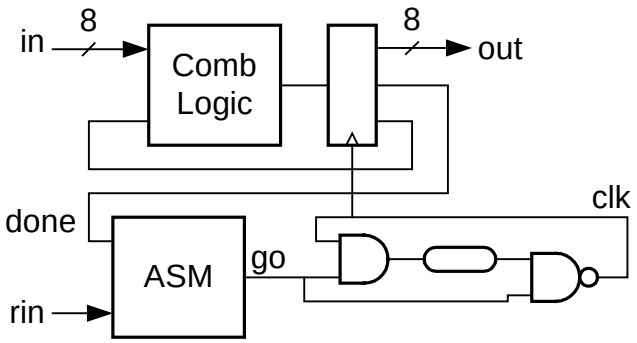
Example, A Stoppable Clock

- Suppose our input is an asynchronous 8-bit signal
- We need to wait for 128 events on the input and then output the average value
- We could synchronize to a local clock on each sample
  - this is an example of unnecessary synchronization
- We could stop our clock and restart it on each sample
  - no probability of synchronization failure
  - lower power





Stoppable Clock Circuit

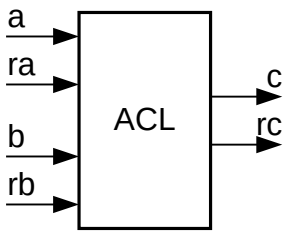


Stoppable Clocks
Some Questions

- What do the waveforms on rin, go, clk, and done look like?
- Where is rout generated?
- What are the constraints on input timing? On output timing?
- How do we design the ASM block?

An Asynchronous Module

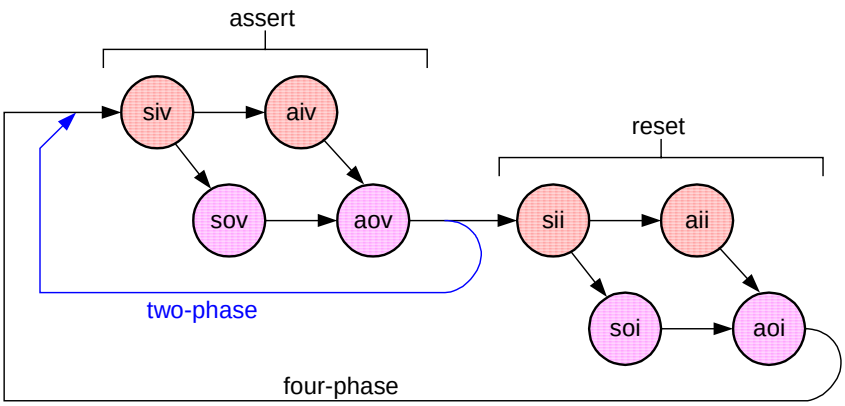
- To ensure correct design of asynchronous circuits (and to simplify verification) we need to impose some *discipline* on our designs
- We start by specifying the properties of a *combinational* asynchronous module
  - some number of *inputs*
    - bundled or dual-rail
  - some number of *outputs*
    - bundled or dual rail
  - a constraint on input and output events



The Weak Conditions

- Inputs and outputs transition in the following order
  1. some input(s) become valid
  2. some output(s) become valid
  3. all inputs become valid
  4. all outputs become valid
  5. some input(s) become invalid
  6. some output(s) become invalid
  7. all input(s) become invalid
  8. all output(s) become invalid
- So, for example it is not allowed for
  - any output to become valid before any input becomes valid
  - all outputs to become valid while any input is invalid
  - any output to become invalid while all inputs are valid
  - all outputs to become invalid while any input is valid

The Weak Conditions State Diagram

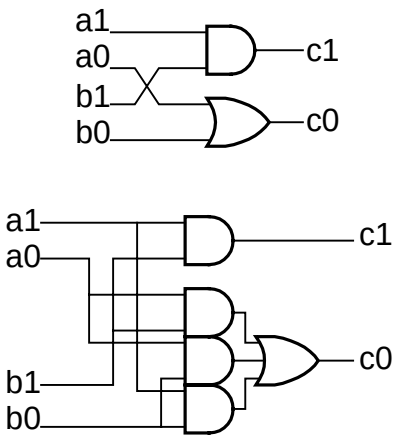


The Weak Conditions Some Questions

- What does it mean for a signal to be *valid* or *invalid*?
- Which conditions apply to the circuit and which apply to the environment?
- Is it sufficient for *one* output to always *lag* all inputs?
- What do the weak conditions say about the number of events on each input and output?

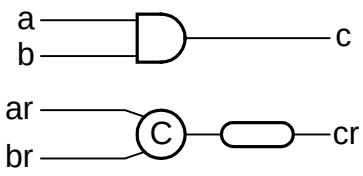
The Weak Conditions
Example: Self-Timed AND Gate

- Suppose we want to build an asynchronous AND gate using this discipline.
  - inputs dual-rail
- Can we just use an AND gate and an OR gate?
  - why not?
- Is the lower circuit OK?
  - why?
  - isn't it logically equivalent to the upper circuit?
  - what's different?



A Bundled Self-Timed AND Gate

- If the inputs are *bundled* instead of dual rail, how do we generate a bundled output?
- A Muller C-element
  - output follows input when both inputs are equal
- This is an example of a *matched-delay* circuit
  - delay of C element plus delay line must be ≥ delay of AND gate
- Output cr is an example of a *completion signal*



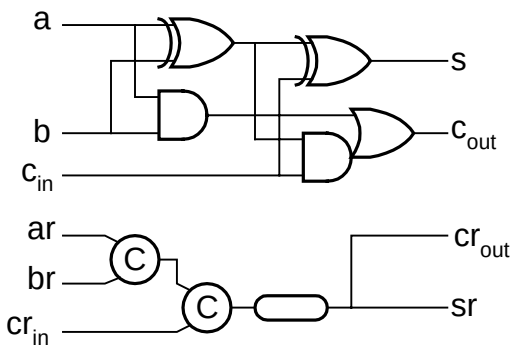
A Matched-Delay Full Adder

- Suppose we want to build a full adder with bundled inputs using the matched-delay design style
- Can we generate a fast carry and still obey the weak conditions?

C_{in}	A	B	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

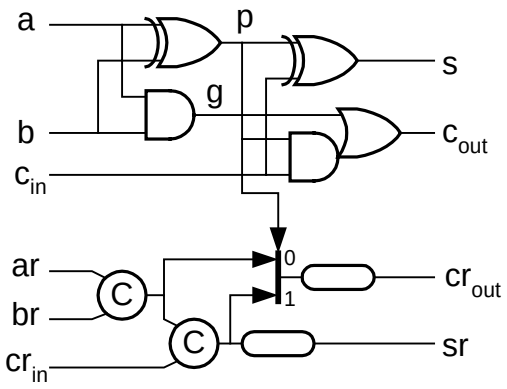
Matched-Delay Full Adder First Attempt

- Does this work?
- Does it give good performance?



Matched-Delay Full Adder with Fast Carry

- Does this work?
- Does it obey the weak conditions?
- Is there a completion signal?
- Under what conditions is the carry *fast*?

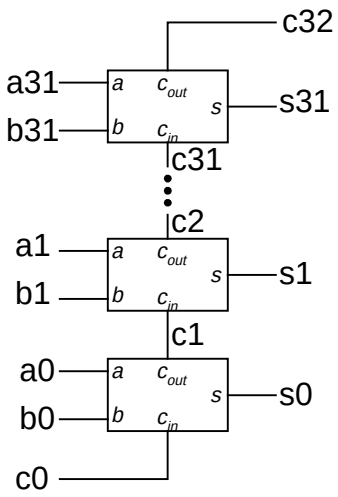


Composition of Asynchronous Modules

- The weak conditions are *closed* under acyclic composition
  - An acyclic composition of modules that obey the weak conditions also obeys the weak conditions
- Only true if the circuit is *fully connected*
  - Two independent parallel circuits do *not* obey the weak conditions
- Why is this true?

Example, An Asynchronous Adder

- Each line here represents two wires: value and request
- What is the **average** delay of this adder with the first full adder design? the second?
- Is there a completion signal? If not how could we generate one?
- Can we *factor out* some of the matched-delay completion logic?



Next Time

- Asynchronous State Machines
- Asynchronous Pipelines
- Iterative Asynchronous Circuits