EECS150 - Digital Design

Lecture 29 - Asynchronous Sequential Circuits

May 6, 2003

John Wawrzynek

Outline

- Synchronizers

  Figures from

  "Digital Design", John F. Wakerly

  Prentice Hall, 2000

  An excellent treatment of the topic.

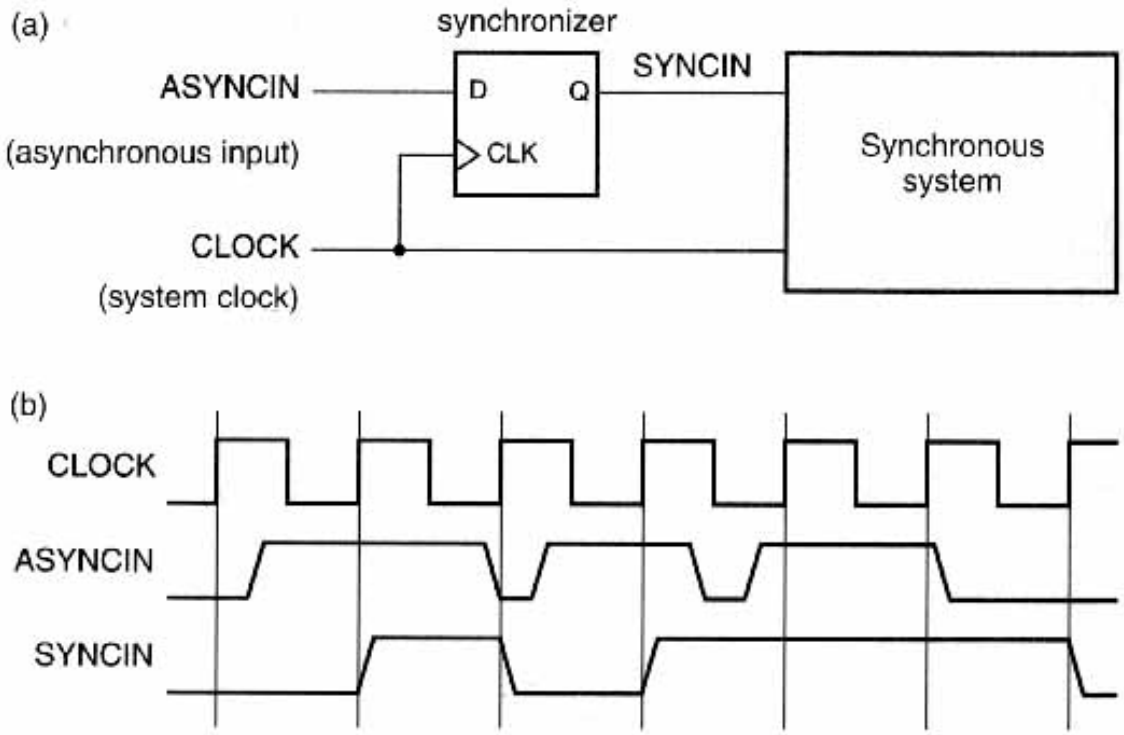
- Purely asynchronous circuits
  - "self-timed" circuits
  - Mano has another class of asynchronous circuits (not covered in class)

Asynchronous Inputs to Synchronous Systems

- Many synchronous systems need to interface to asynchronous input signals:
  - Consider a computer system running at some clock frequency, say 1GHz with:
    - Interrupts from I/O devices, keystrokes, etc.
    - Data transfers from devices with their own clocks
      - Ethernet has its own 100MHz clock
      - PCI bus transfers, 66MHz standard clock.
  - These signals could have no known timing relationship with the system clock of the CPU.

"Synchronizer" Circuit

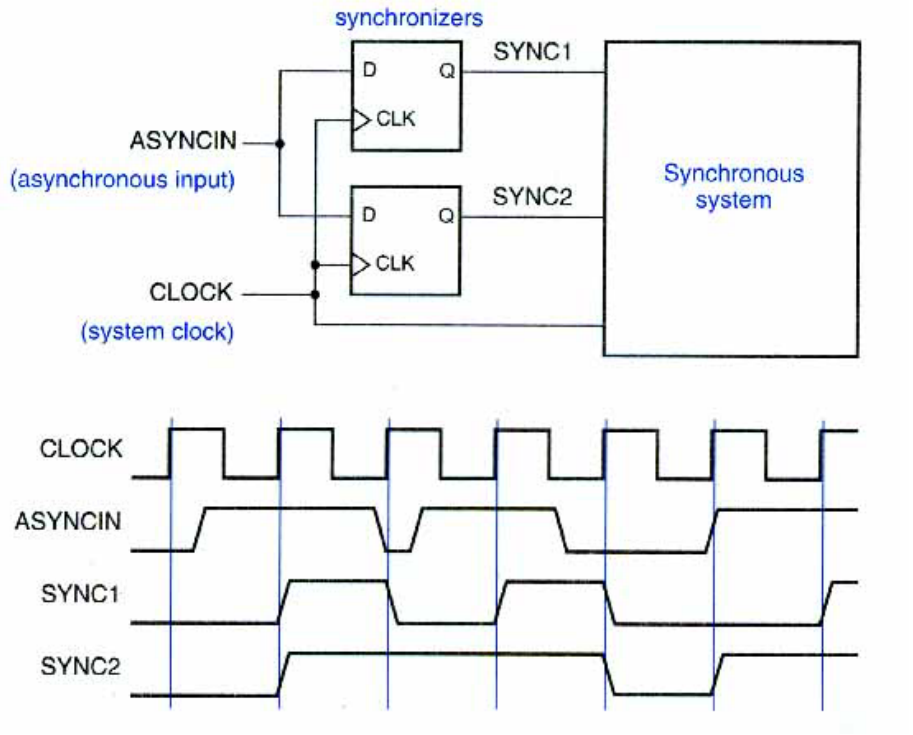
- For a single asynchronous input, we use a simple flip-flop to bring the external input signal into the timing domain of the system clock:



- The D flip-flop samples the asynchronous input at each cycle and produces a synchronous output that meets the setup time of the next stage.

"Synchronizer" Circuit

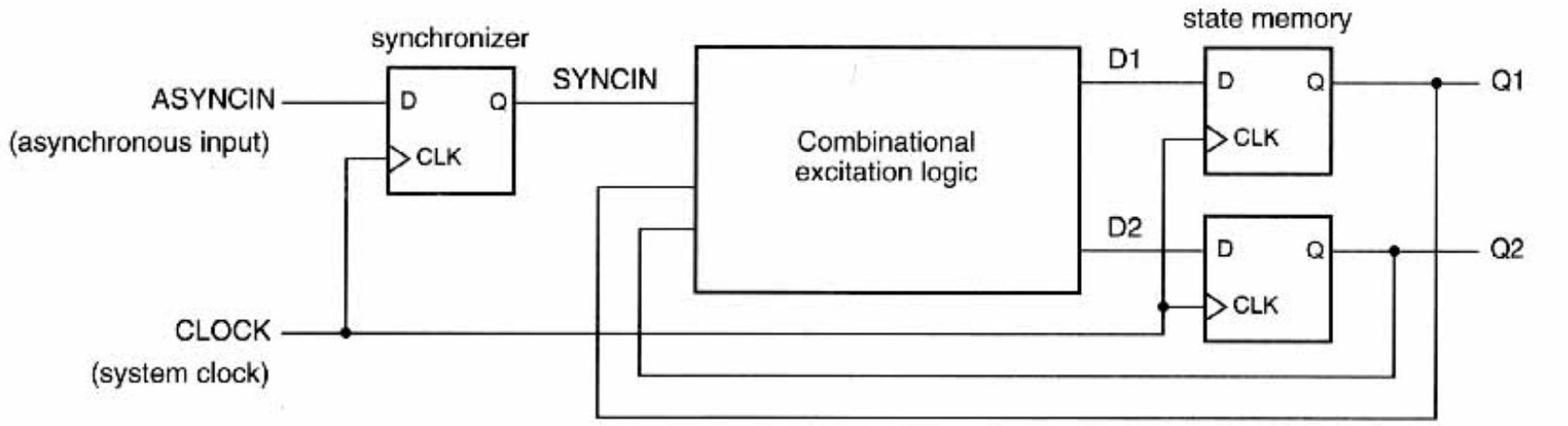
- It is essential for asynchronous inputs to be synchronized at only one place.



- Two flip-flops may not receive the clock and input signals at precisely the same time (clock *and* data skew).
- When the asynchronous changes near the clock edge, one flip-flop may sample input as 1 and the other as 0.

“Synchronizer” Circuit

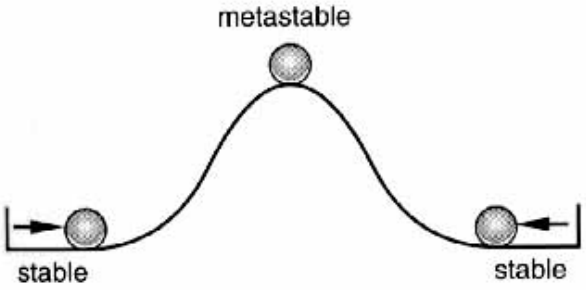
- Single point of synchronization is even more important when input goes to a combinational logic block (ex. FSM)
- The CL block can accidentally hide the fact that the signal is synchronized at multiple points.
- The CL magnifies the chance of the multiple points of synchronization seeing different values.

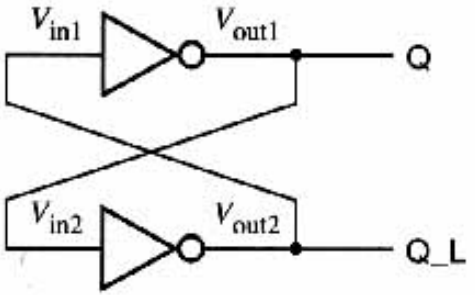


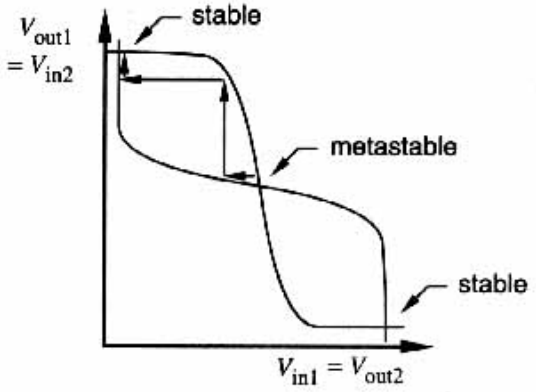
- Sounds simple, right?

Synchronizer Failure & Metastability

- We think of flip-flops having only two stable states - but all have a third *metastable* state halfway between 0 and 1.
- When the setup and hold times of a flip-flop are not met, the flip-flop could be put into the metastable state.
- Noise will be amplified and push the flip-flop one way or other.
- However, in theory, the time to transition to a legal state is unbounded.
- Does this really happen?
- The probability is low, but
  the number of trials is high!







Transfer function:

$V_{out1} = T(V_{in1})$

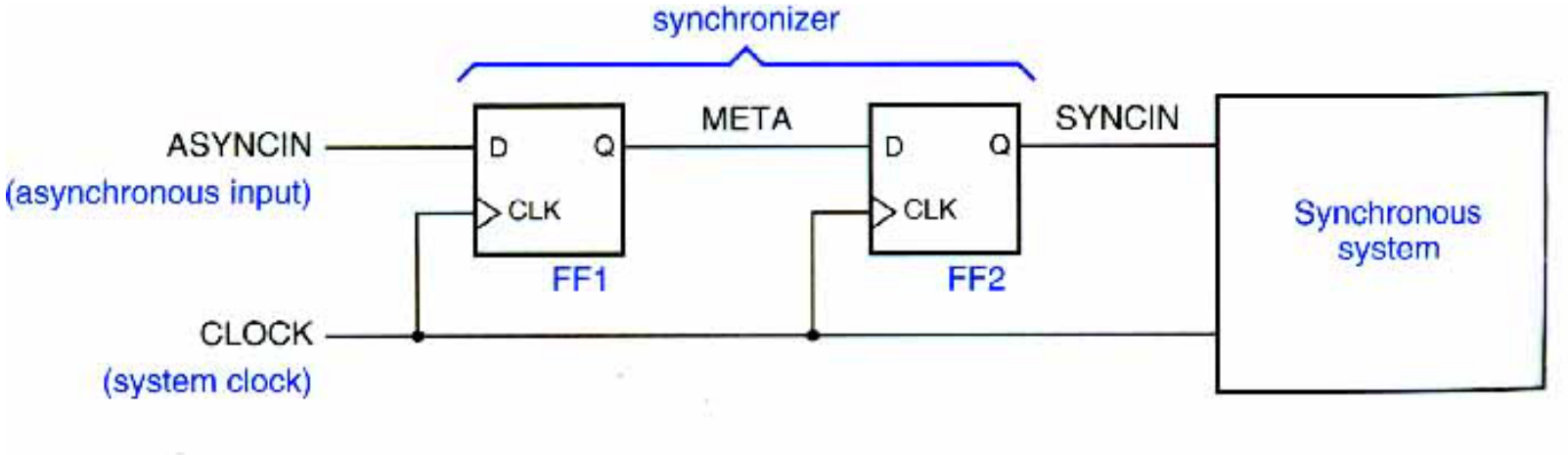
$V_{out2} = T(V_{in2})$

Synchronizer Failure & Metastability

- If the system uses a synchronizer output while the output is still in the metastable state $\Rightarrow$ synchronizer failure.
- Initial versions of several commercial ICs have suffered from metastability problems - effectively synchronization failure:
  - AMD9513 system timing controller
  - AMD9519 interrupt controller
  - Zilog Z-80 Serial I/O interface
  - Intel 8048 microprocessor
  - AMD 29000 microprocessor
- To avoid synchronizer failure wait long enough before using a synchronizer's output. *"Long enough", according to Wakerly, is so that the mean time between synchronizer failures is several orders of magnitude longer than the designer's expected length of employment!*
- In practice all we can do is reduce the probability of failure to a vanishing small value.

Reliable Synchronizer Design

- The probability that a flip-flop stays in the metastable state decreases exponentially with time.
- Therefore, any scheme that delays using the signal can be used to decrease the probability of failure.
- In practice, delaying the signal by a cycle is usually sufficient:



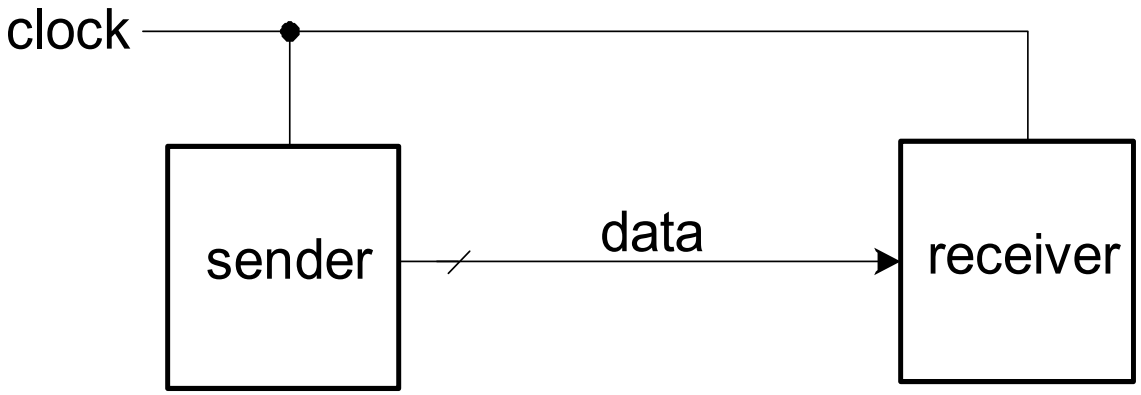
- If the clock period is greater than metastability resolution time plus FF2 setup time, FF2 gets a synchronized version of ASYNCIN.
- Multi-cycle synchronizers (using counters or more cascaded flip-flops) are even better – but often overkill.

Purely Asynchronous Circuits

- Many researchers (and a few industrial designers) have proposed a variety of circuit design methodologies that **eliminate the need for a globally distributed clock**.
- They cite a variety of important potential advantages over synchronous systems (will list later).
- To date, these attempts have remained mainly in Universities.
- A few commercial asynchronous chips/systems have been build.
- Sometimes, asynchronous blocks sometimes appear inside otherwise synchronous systems.
- Asynchronous techniques have long been employed in DRAM and other memory chips for generation internal control without external clocks. (Precharge/sense-amplifier timing based on address line changes.)
- These techniques are generally interesting, and if nothing else help put synchronous design in perspective.

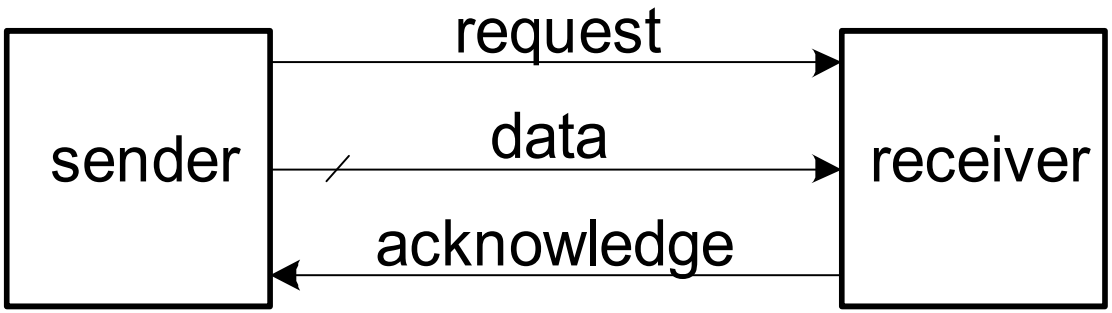
Synchronous Data Transfer

- In synchronous systems, the clock signal is used to coordinate the movement of data around the system.
- If we are going to eliminate the clock, we need to substitute some technique for managing the flow of data.
- Take for example, transferring data across a bus:

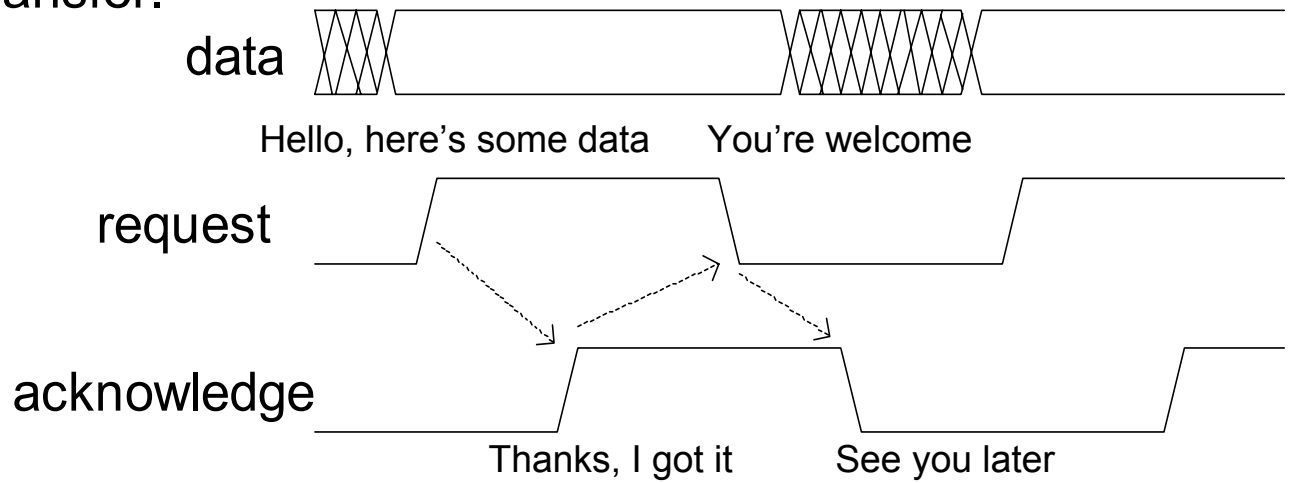


- By design, the clock period is sufficiently long to accommodate wire delay and time to get the data into the receiver.

Delay Insensitive (self-timed transfer)



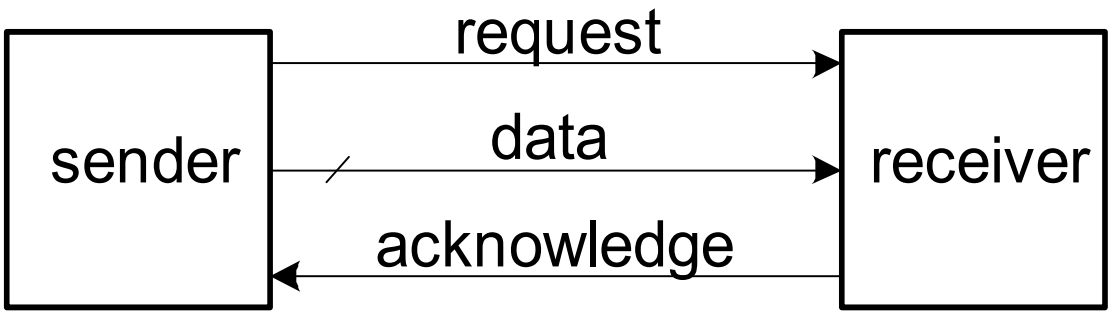
- Request/acknowledge "handshake" signal pair used to coordinate data transfer.



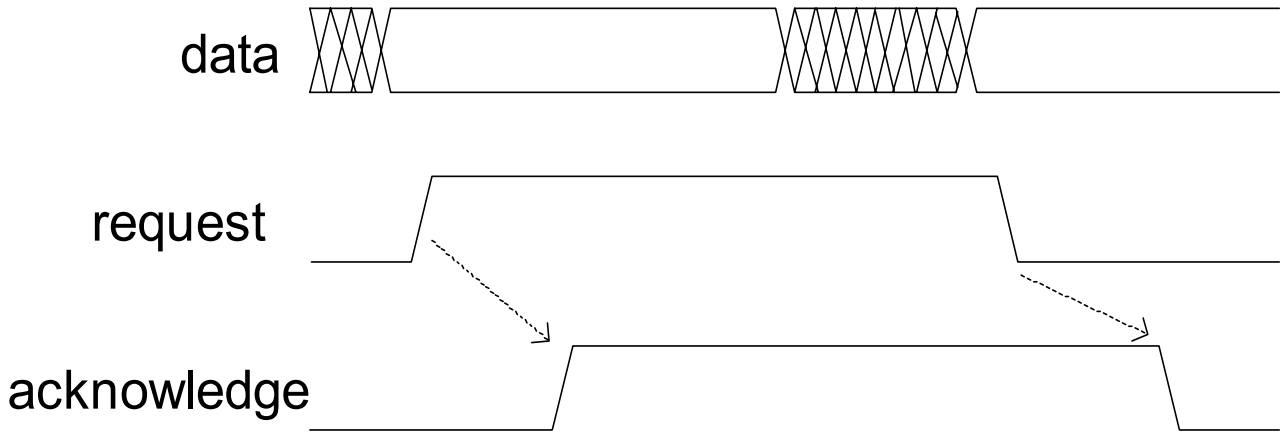
4-cycle ("return-to-zero") signaling

- Note, transfer is insensitive to any delay in sending and receiving.

Delay Insensitive (self-timed transfer)



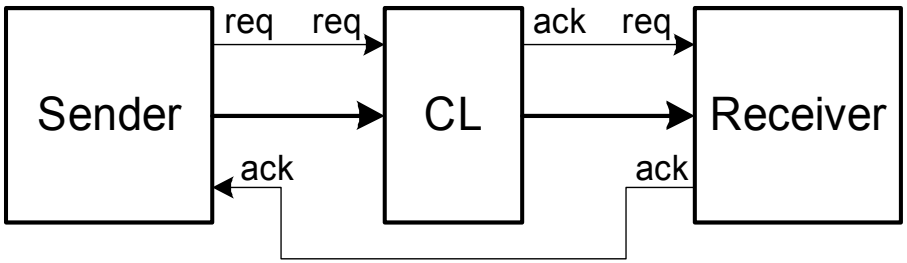
2-cycle ("non-return-to-zero") signaling



- Only two transitions per transfer. Maybe higher performance.
- More complex logic. 4-cycle return to zero can usually be overlapped with other operations.

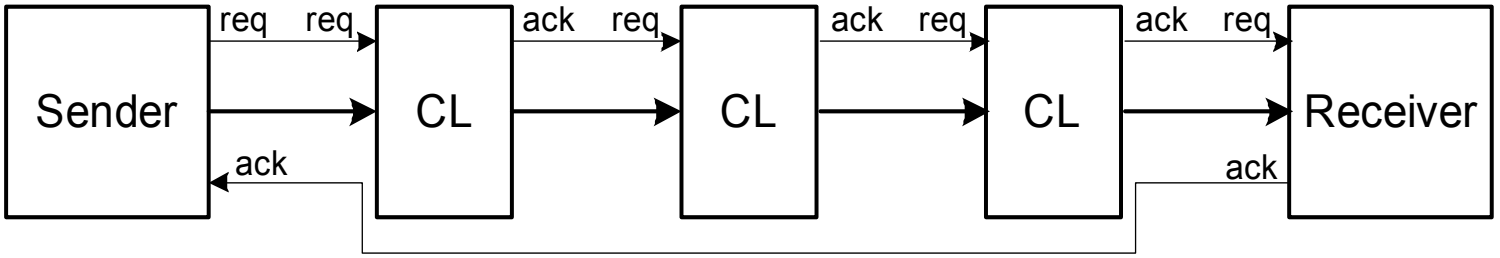
Self-timed Processing

- Of course, a processing elements can be inserted. Req signal starts it, and it generates a "completion" (ack) signal when its output data is ready.

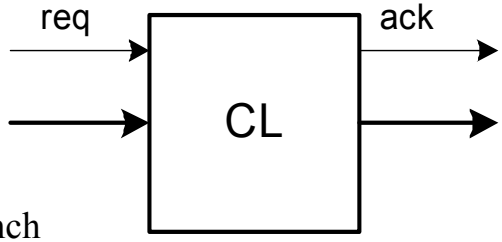


"req" = "go"
"ack" = "done"

- The output ack becomes the request for the receiver or next stage:

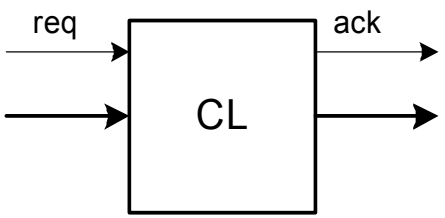


- Note, three cascaded CL blocks as a composite preserves the signaling convention:

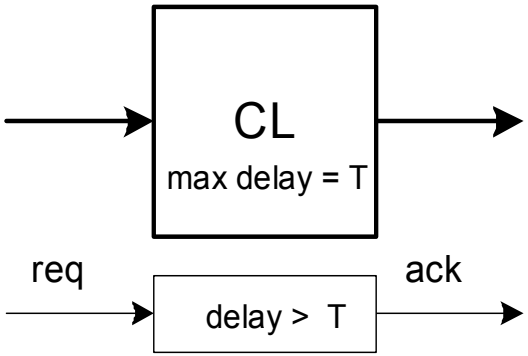


Completion Signal Generation

- Output ack signal is generated one of several ways:
  - **derived** from handshake signals of sub-blocks,
  - **fixed delay**, arranged to match delay of logic circuit.



1. Fixed delay

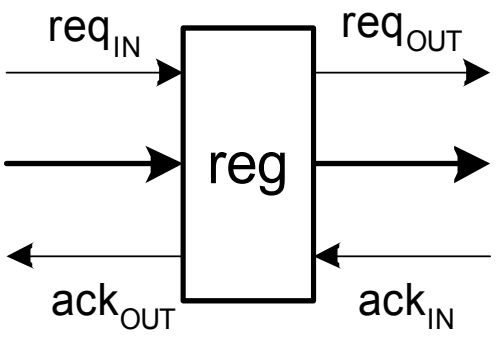


   - A fixed delay (for instance a chain of gates) greater than the worst case circuit delay is used.
   - Works best for regular structures (memories, PLAs) where dummy circuits can be used to mimic block delay.

2. Derived ack signal offers potential performance advantages, because it does not need to be worst case. Example, adder circuit.

Self-timed Processing Compositions

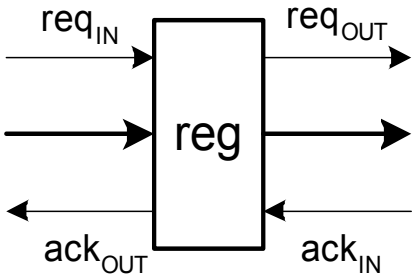
- Other interesting compositions are possible:
- Fan-in: req is “and” of requests from incoming blocks. Data is ready with *all* sets of data is ready. Send ack to all blocks.
- Fan-out: send req to all block receiving output data. Returning acks get “anded”.
- Pipelines: Need to define self-timed register.



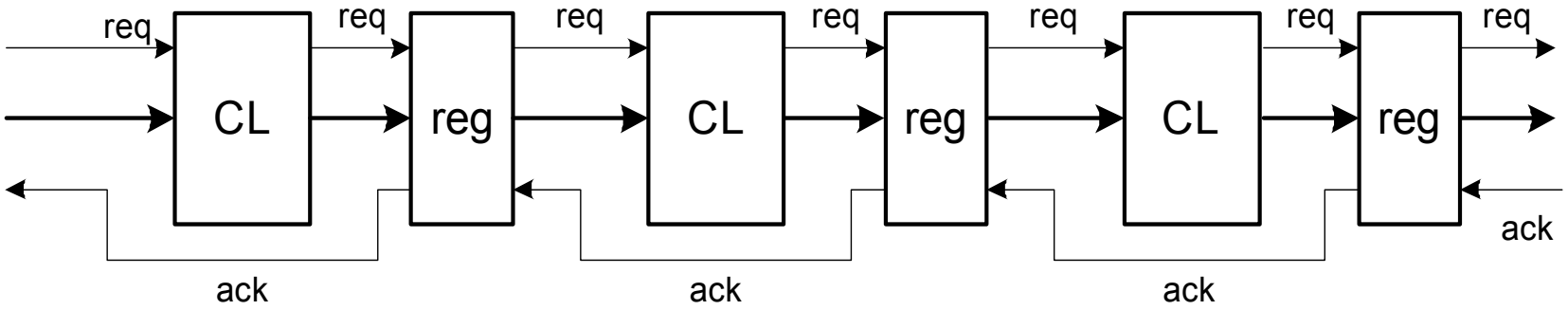
Keeps one bit of state, “empty”

On req_{IN} if empty {
load data,
clear empty,
assert req_{OUT},ack_{OUT}}
else wait for ack_{IN}
On ack_{IN} {
deassert req_{out}
set empty}

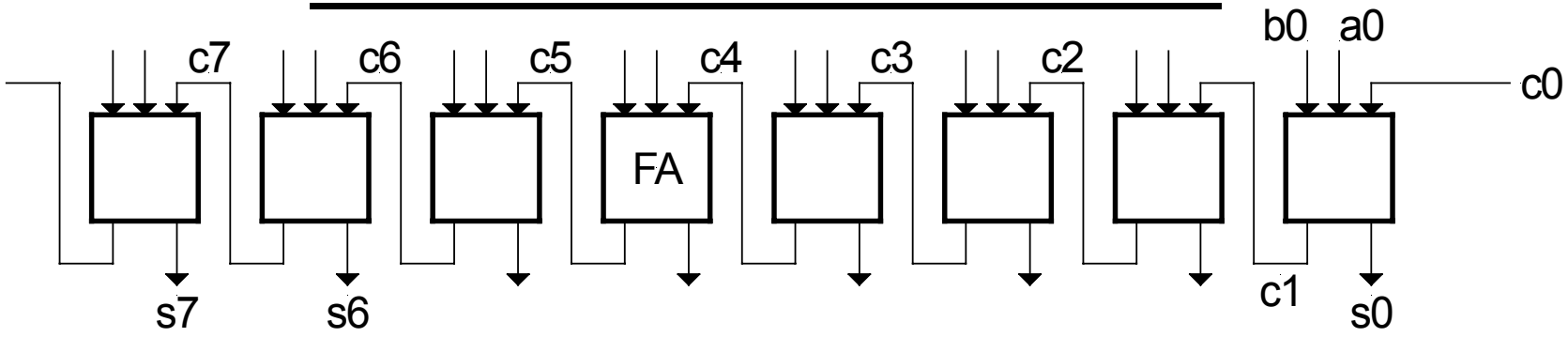
Self-timed Pipeline



- Registers *pipeline* data and handshake signals:



Self-timed Adder Scheme



- Include an req signal at each input and ack on each output.
- Completion signal for each carry out can be generated "early" when ever a=b (carry kill or carry generate). No need to wait for carry in.

a b c_i	c_{i+1}	s	
0 0 0	0	0	carry "kill"
0 0 1	0	1	$k_i = a_i' \, b_i'$
0 1 0	0	1	
0 1 1	1	0	carry "propagate"
1 0 0	0	1	$p_i = a_i \oplus b_i$
1 0 1	1	0	
1 1 0	1	0	carry "generate"
1 1 1	1	1	$g_i = a_i \, b_i$

- Therefore entire adder completion time is a function of the input data.
- On average, number of stages propagating carry bounded by log(n). Therefore on average delay is proportional to log(n) instead of n.
- Demonstrates important principle of self-timed circuits. Often avoid worst-case behavior.

Asynchronous Logic Pluses and Minuses

- Advocates make the following claims (Al Davis):
  1. Achieve average case performance
  2. Consume power only when needed
  3. Provide easy modular composition
  4. Do not require clock alignment at interfaces
  5. Metastability has time to resolve
  6. Avoid clock distribution problems
  7. Exploit concurrency more gracefully
  8. Provide intellectual challenge
  9. Exhibit intrinsic elegance
  10. Global synchrony does not exist anyway!

- The above claims are often debated. Also, known disadvantages:
  1. Time/area overhead.
  2. Not well supported by CAD tools.
  3. Lack of clock complicates debugging and verification.

Caltech Asynchronous Microprocessor

- 1998, Alain Martin and students.
- Completely asynchronous implementation of a MIPS R3000.
- 32-bit RISC CPU with memory management unit.
- 2 -KB caches.
- Used 0.6um CMOS process
- Results:
  - 180 MIPS and 4W at 3.3V
  - 100 MIPS and 850nW at 2.0V
  - 60 MIPS and 220mW at 1.5V
- Some layout bugs, but still …
- Around 2.5X performance of commercial processor of the same type and in equivalent technology.
- Some of the students formed Fulcrum Microsystems to commercialize asynchronous chips (http://www.fulcrummicro.com).