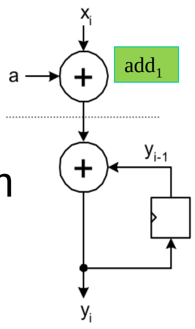
EECS150 - Digital Design
Lecture 10 - Interfacing

Oct. 1, 2013
Prof. Ronald Fearing
Electrical Engineering and Computer Sciences
University of California, Berkeley

(slides courtesy of Prof. John Wawrzynek)

http://www-inst.eecs.berkeley.edu/~cs150

Recap and Topics



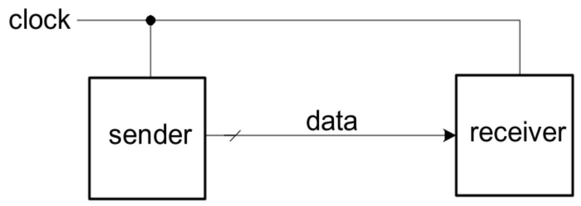
- Thu 9/29: pipelining and parallelism

Today

- How do components (modules) communicate?
- Synchronizing
- FIFO

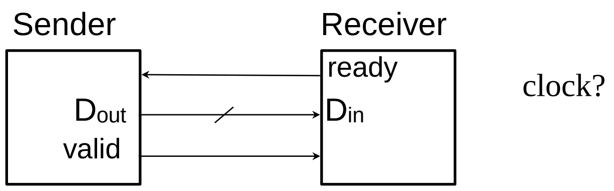
Synchronous Data Transfer

- In synchronous systems, the clock signal is used to coordinate the movement of data around the system.
- Take for example, transferring from module to module:



- By design, the clock period is sufficiently long to accommodate wire delay and time to get the data into the receiver.
- Assumes:
  - sender is ready to send data on each cycle & receiver is ready to receive data on each cycle
- What if the communication is sporadic?

Data Transfer with Control



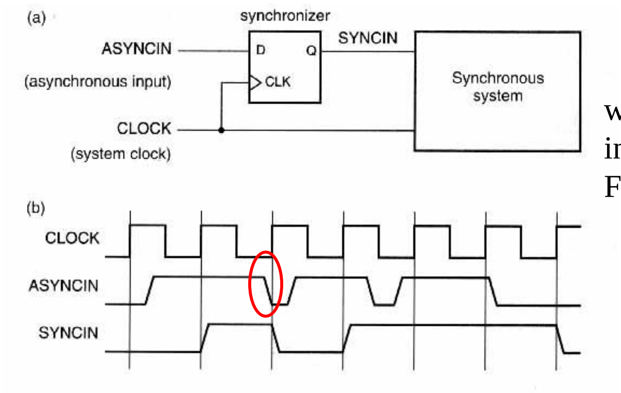
- After checking to see if the receiver is ready to receive, the sender asserts the “valid” signal to indicate that the data lines hold data for transferring
- As with synchronous transfer, the sender assumes that the transfer happens successfully
- Design constraint: the ready signal needs to be stable early enough in the cycle to allow the sender to respond with data and the valid signal before set-up time of receiver.
- Can we pipeline the control?

Communicating Across Clock Boundaries

- 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:

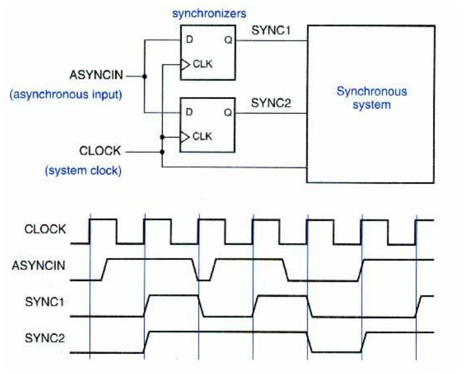


what if async input went to FSM?

- 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

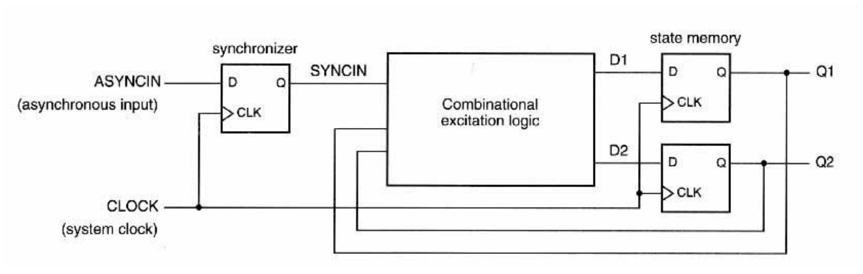
- It is essential for asynchronous inputs to be synchronized at only one place.



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

“Synchronizer” Circuit

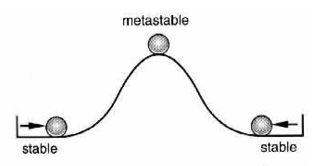
- Single point of synchronization is even more important when input goes to a combinational logic block (ex. FSM)
- The CL block can accidentally hide the fact that the signal is synchronized at multiple points.
- The CL magnifies the chance of the multiple points of synchronization seeing different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

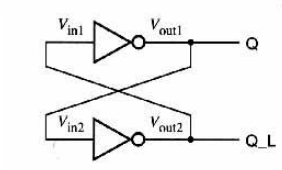


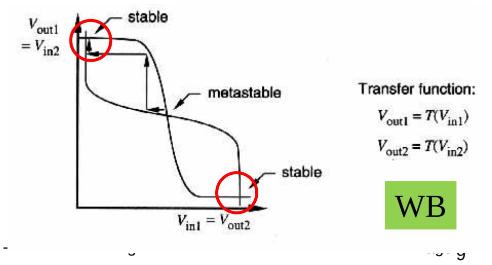
- Sounds simple, right?

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 number of trials is high!







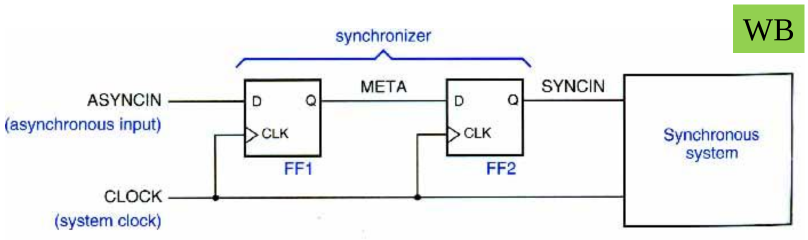
Synchronizer Failure & Metastability

- If the system uses a synchronizer output while the output is still in the metastable state ➔ 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:

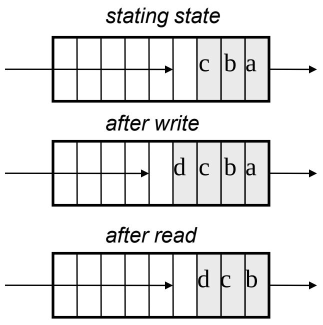
WB



- 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.

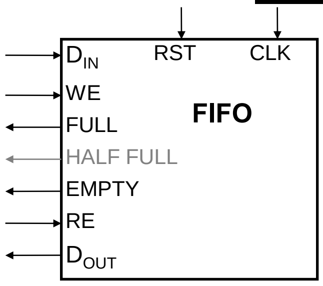
First-in-first-out (FIFO) Memory

- Used to implement *queues*.
- These find common use in computers and communication circuits.
- Generally, used to “decouple” actions of producer and consumer:



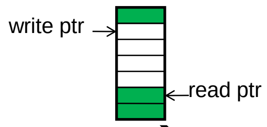
- Producer can perform many writes without consumer performing any reads (or vis versa). However, because of finite buffer size, on average, need equal number of reads and writes.
- Typical uses:
  - interfacing I/O devices. Example network interface. Data bursts from network, then processor bursts to memory buffer (or reads one word at a time from interface). Operations not synchronized.
  - Example: Audio output. Processor produces output samples in bursts (during process swap-in time). Audio DAC clocks it out at constant sample rate.

FIFO Interfaces

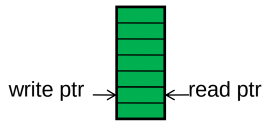


- After write or read operation, FULL and EMPTY indicate status of buffer.
- Used by external logic to control own reading from or writing to the buffer.
- FIFO resets to EMPTY state.
- HALF FULL (or other indicator of partial fullness) is optional.

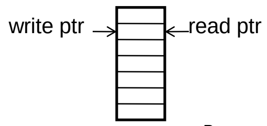
- Address pointers are used internally to keep next write position and next read position into a dual-port memory.



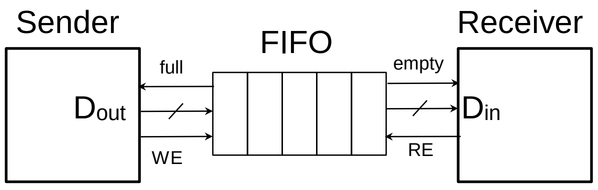
- If pointers equal after write → FULL:



- If pointers equal after read → EMPTY:



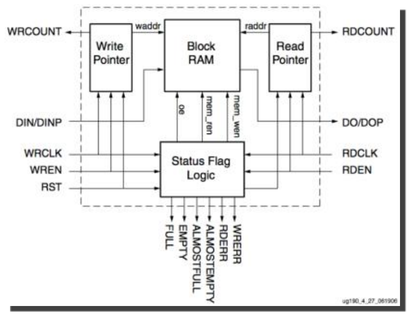
Decoupled Communication



- Allow sender and receiver to transfer data independently
- Assumes, on average, equal number of sends and receives
- FIFO buffer must be large enough to accommodate instantaneous difference in send and receive rate

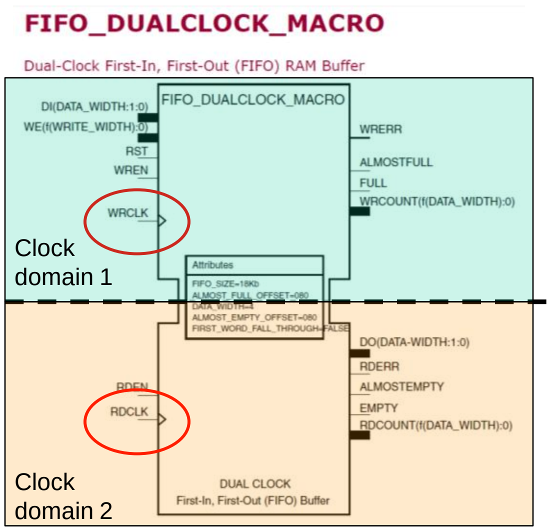
Xilinx Virtex5 FIFOs

- Virtex5 BlockRAMS include dedicated circuits for FIFOs (details in User Guide (ug190)).
- Takes advantage of separate dual ports and **independent ports clocks**.



- Often used for crossing clock boundaries.

Xilinx Virtex FIFO


FIFO_DUALCLOCK_MACRO

Dual-Clock First-In, First-Out (FIFO) RAM Buffer

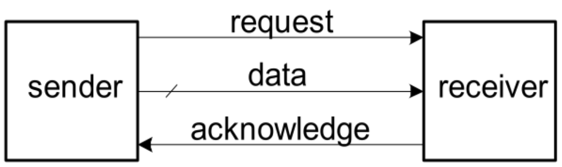
Attributes
FIFO_SIZE=18Kb
ALMOST_FULL_OFFSET=080
DATA_WIDTH=4
ALMOST_EMPTY_OFFSET=080
FIRST_WORD_FALL_THROUGH=FALSE

Virtex5HDL p. 41

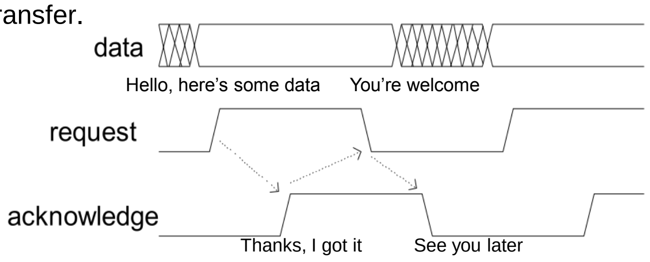
Purely Asynchronous Circuits

- 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.
- 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.
- In GALS (globally asynchronous locally synchronous) systems, independently synchronous islands communicate asynchronously.

Delay Insensitive (self-timed transfer)



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

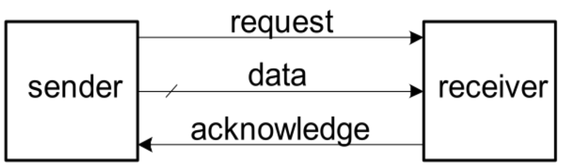


4-cycle ("return-to-zero") signaling

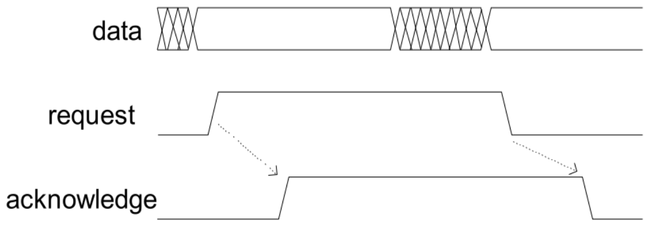
WB

- 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.

Summary

- Asynchronous inputs ➔ need synchronizer
- multiple data sources and sinks with different clocks ➔ dual clock FIFO
- Asynchronous data transfer: full handshaking