EE178 Lecture Module 4

Eric Crabill

SJSU / Xilinx

Fall 2005

Lecture #9 Agenda

- Considerations for synchronizing signals.
  - Clocks.
  - Resets.
- Considerations for asynchronous inputs.
- Methods for crossing clock domains.

Clocks

- The “academic” clock distribution is one that would deliver clock events to all synchronous elements in the system with zero delay, zero skew, and zero jitter.
  - This is what you see in functional simulation.
  - Not representative of physical reality.
- Some designs can actually make constructive use of clock delay and clock skew.

Clocks

- Most FPGA devices have special routing intended for use with high fan-out, low skew signals such as clocks.
  - Typically a limited (precious) resource.
  - Usually driven by a “global buffer” primitive.
  - Better delay and skew characteristics than normal routing resources in the FPGA.

Clocks

- In Xilinx FPGA devices, you indicate your desire to use these resources by instantiating a "global buffer" in your design to drive the clock signal of interest.
  - Schematic designs use a primitive called BUFG.
  - HDL designs have two options:
    - Direct use of instantiated BUFG primitive from library.
    - Allow synthesis tool to identify wires which are used as clocks and automatically infer BUFG primitives.

Clocks

- Since most FPGA devices have a limited number of these clock distribution resources, it makes sense to minimize the number of unique clocks in your design.
  - Avoid “gating the clock”.
  - Avoid things like ripple counters.
  - Use clock enables instead of divided clocks.
- A side benefit is that your static timing analysis will be less complicated!

Resets

- Most designs use another synchronization signal, a “reset”, to put the system in an initial state.
- Initial state does not need to be all zero or all one, it can be whatever you need; you may not need (or want) to initialize every state element.
- Reset signals can be synchronous (to the system clock) or asynchronous.

Synchronous Resets

- A synchronous reset is synchronized to the clock.
- You may consider it as “just another synchronous input” to state elements in the design.
- The synchronous reset will have priority over other inputs, such as the D input.
- When the reset is asserted and the clock event takes place, the flip-flop will transition.

Synchronous Resets

- A synchronous reset input to a flip flop has the same timing requirements as other synchronous inputs to the flip-flop.
- If the synchronous reset signal is coming from an external source, it must meet input setup and hold requirements.
- If the synchronous reset signal is coming from an internal source (say, another flip flop), it must meet the period requirement.

Asynchronous Resets

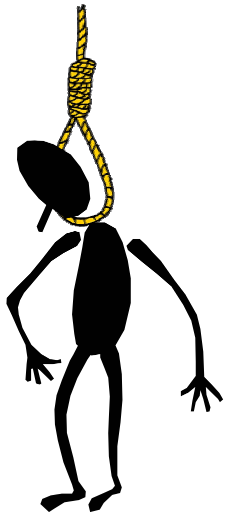
- An asynchronous reset is not synchronized to the clock; when it is asserted, the state element will immediately transition.
- Typically, these types of asynchronous control signals have priority over all other inputs to the flip-flop, even the clock.
- No clock events are required to initialize.

Asynchronous Resets

- In contrast to a synchronous reset, this reset has a potential problem when deasserted.
  - It can occur at any time, even near clock edges.
  - Skew on the signal distribution can result in different portions of the design "waking up" at different times, sending the design into some state other than what was intended.
  - Can build logic in such a way that waking up in the wrong state is either harmless or correctable.

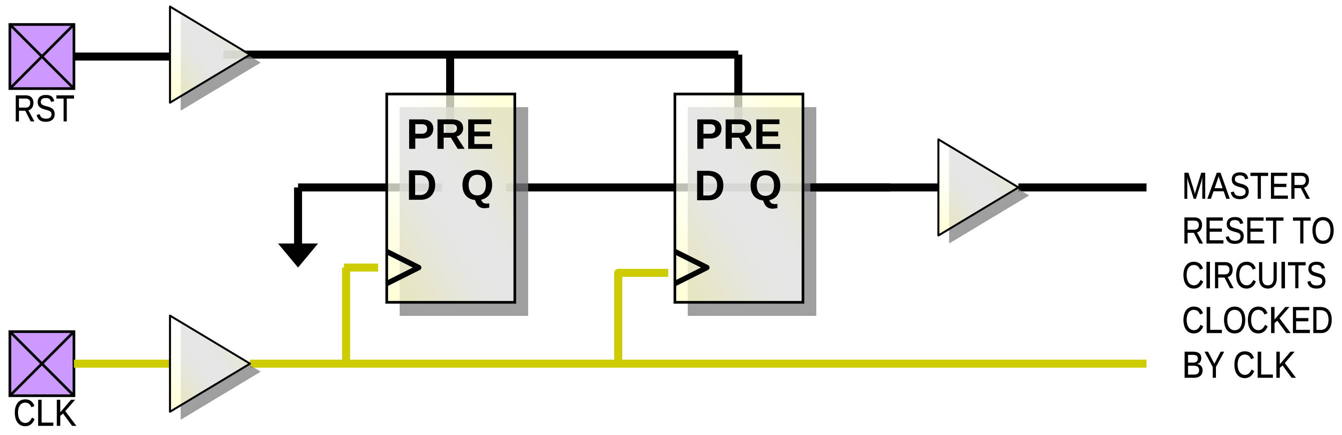
Asynchronous Resets

```
module hang_yourself (detonate_warhead, clk, rst);

output detonate_warhead;
input clk, rst;

reg flop1, flop2;
reg detonate;

always @(posedge clk or posedge rst)
begin
  if (rst) flop1 <= 1'b0;
  else flop1 <= !flop1;
end

always @(posedge clk or posedge rst)
begin
  if (rst) flop2 <= 1'b0;
  else flop2 <= !flop2;
end

always @(posedge clk or posedge rst)
begin
  if (rst) detonate_warhead <= 1'b0;
  else detonate_warhead <= flop1 ^ flop2;
end

endmodule
```



Asynchronous Solution

- Interesting circuit to drive asynchronous resets:
  - Master reset asserts asynchronously, forcing your circuit into a known state immediately.
  - Master reset de-asserts synchronously, allowing meaningful timing requirements / analysis.



Xilinx FPGA Resets

- In your design, you may design with either type of reset, and it will be implemented in the FPGA as you designed it.
- Use of synchronous resets may reduce logic.
- There are two other initialization signals in Xilinx FPGAs which are not well documented.
  - GSR, the global set/reset.
  - GTS, the global three state control.

Xilinx FPGA Resets

- At power-on, and when directed, the FPGA starts loading a configuration bitstream, which is a description of your design.
- While the bitstream is loading, the FPGA will have an incomplete description of your logic.
  - Your state could get messed up or transition.
  - Your output pins could be driven incorrectly.

Xilinx FPGA Resets

- Until the bitstream load is complete, the FPGA holds GSR and GTS asserted.
  - All flip flops are held in an initial state by GSR.
    - FDC type primitives have initial state of zero.
    - FDP type primitives have initial state of one.
  - All chip outputs are held in three-state by GTS.
- When configuration completes, GSR and GTS are released, and your design begins to operate.

Xilinx FPGA Resets

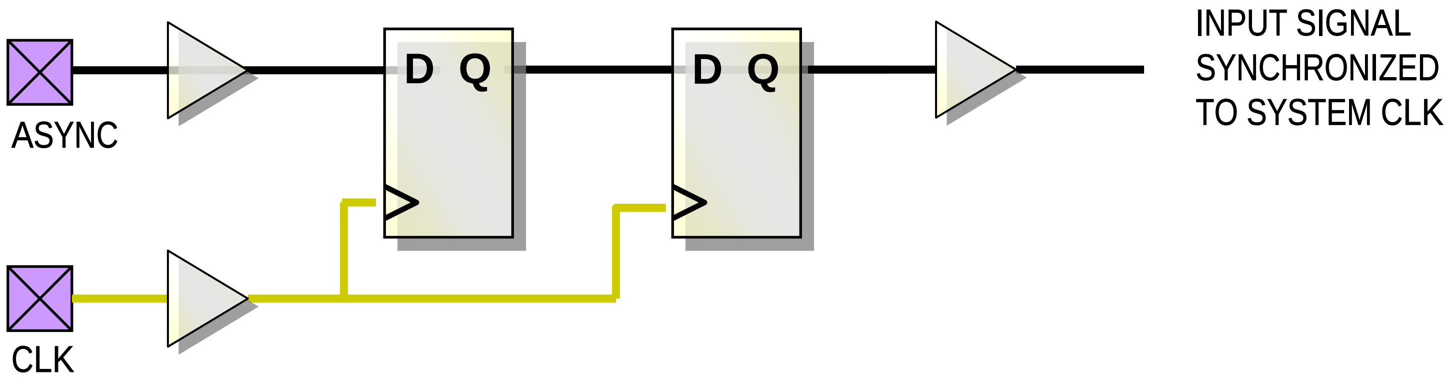
- Once the user design is active, it is possible to drive GSR and GTS under user control.
  - Asserting GSR would be rather disruptive.
  - Asserting GTS is useful to three-state the entire chip at once.
  - Control of GSR and GTS is done via the STARTUP primitive in the library which you may use in HDL and schematic designs.

Asynchronous Inputs

- Asynchronous inputs, like buttons, switches, and anything not synchronized to the system clock will inevitably cause input setup or input hold violations.
  - May not be an issue on data path circuits.
  - Can be fatal on control circuits.
  - Why is metastability a problem?

Asynchronous Inputs

- Synchronize signals to system clock using a synchronizer circuit:



Asynchronous Inputs

- Synchronizer circuits add delay (latency).
- Synchronizer circuits are not perfect guarantees.
  - Place flops close to each other to minimize net delay.
  - How good is good enough? (MTBF calculations)
- When a signal comes on-chip, synchronize it once and then fan signal out as required.
  - Do not fan out, then synchronize at multiple places.
  - Variations in timing can create different results.

Clock Domains

- A clock domain is a group of logic elements and related signals that are synchronized to one clock.
- The emphasis of this course and the labs is fully synchronous design -- that is, design with only one clock domain.
- Many designs do not fit into this “paradigm”.

Clock Domains

- Why would you have multiple clock domains?
  - Independent (sub)systems with different reference clocks, needing to share/exchange information.
  - Impractical to distribute or use a reference clock.
  - Many other reasons, I’m sure…
- How may the clocks in two domains be related?
  - Synchronous (degenerate case, same clock).
    - Same frequency.
    - Zero phase difference.

Clock Domains

- How may the clocks in two domains be related?
  - Derived, Synchronous.
    - Frequencies related to a common reference.
    - Phase difference is a function of time.
    - Example: Multiplied or divided clock from DLL or DFS.
  - Mesochronous.
    - Same frequency.
    - Constant phase difference.
    - Example: Phase shifted clock from DLL or DFS.

Clock Domains

- How may the clocks in two domains be related?
  - Plesiochronous.
    - Different frequencies, nominally the same.
    - Phase difference is slowly varying.
    - Example: Two oscillators, both marked 1.000000 MHz.
  - Asynchronous.
    - Different frequencies or non-periodic clocks.
    - Arbitrary phase difference.
    - Example: Two clocks of unknown relationship.

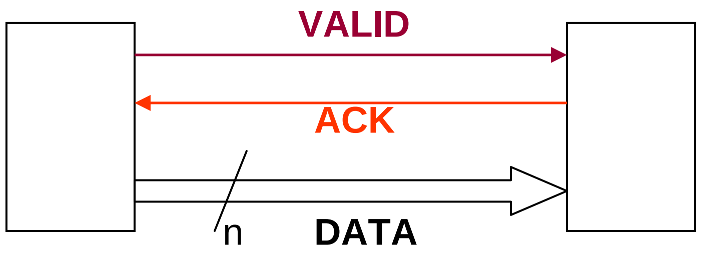
Crossing Clock Domains

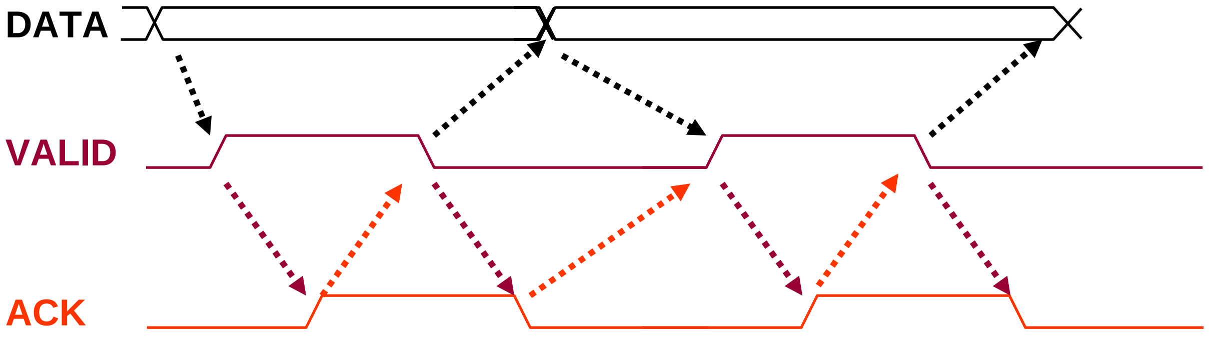
- For asynchronous clock domain relationships:
  - For a single signal, use the same two flip-flop synchronizer used for asynchronous inputs.
  - For multi-bit signals, simply synchronizing each of the bits is not sufficient because each instance of the synchronizer may resolve at different times.
    - No way to know when multi-bit quantity is valid, other than waiting a long time…
    - Use four phase or two phase handshaking (a single point of synchronization).

Crossing Clock Domains

- For asynchronous clock domain relationships:
  - Four phase handshaking (RTZ, level based flags).
    - Source domain provides DATA and asserts its VALID flag.
    - Destination domain sees synchronized VALID flag assert and takes DATA, then asserts its ACK flag.
    - Source domain sees synchronized ACK flag assert and deasserts its VALID flag.
    - Destination domain sees synchronized VALID flag deassert and deasserts its ACK flag.
    - Process can then repeat…

Four Phase Handshake



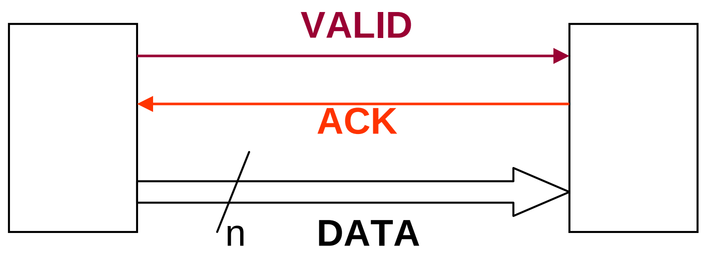


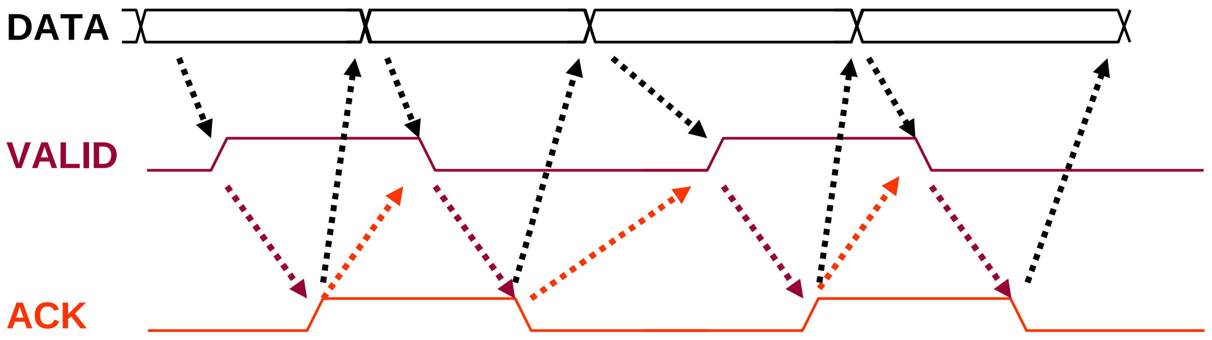
[Adapted from VLSI Architectures Spring 2004 www.ee.technion.ac.il/courses/048878 by Ran Ginosar]

Crossing Clock Domains

- For asynchronous clock domain relationships:
  - Two phase handshaking (NRZ, transition flags).
    - Source domain provides DATA and changes VALID flag.
    - Destination domain sees synchronized VALID flag change and takes DATA, then changes its ACK flag.
    - Source domain sees synchronized ACK flag change.
    - Process can then repeat…

Two Phase Handshake





[Adapted from VLSI Architectures Spring 2004 www.ee.technion.ac.il/courses/048878 by Ran Ginosar]

Crossing Clock Domains

- For asynchronous clock domain relationships:
  - For bulk data transfer, but low bandwidth, use some form of memory (dual ported is convenient…) with handshaking to indicate which domain is in control at a given time.
    - “Fill and spill” buffers -- high latency, low throughput.
    - “Ping-pong” (double buffering) -- some improvement.
  - With a dual ported RAM, can I be clever about this and start “spilling” while it’s still “filling”?

Crossing Clock Domains

- For asynchronous clock domain relationships:
  - Yes, it is called an asynchronous FIFO.
    - Usually implemented with a dual ported memory.
    - On the source (write) domain, data can be written into the FIFO as long as the FIFO is not FULL.
    - On the destination (read) domain, data can be read out of the FIFO as long as the FIFO is not EMPTY.
    - See "Simulation and Synthesis Techniques for Asynchronous FIFO Design" by Cliff Cummings.

Crossing Clock Domains

- For other clock domain relationships:
  - There are a variety of other methods to deal with clock domain crossing, if more is known about the nature of the clock signals.
  - Pretending things are asynchronous always works…