How Do I Reset My FPGA?

Devising the best reset structure can improve the density, performance and power of your design.

by E. Srikanth
Solutions Development Engineer
Xilinx, Inc.
serusal@xilinx.com

In an FPGA design, a reset acts as a synchronization signal that sets all the storage elements to a known state. In a digital design, designers normally implement a global reset as an external pin to initialize the design on power-up. The global reset pin is similar to any other input pin and is often applied asynchronously to the FPGA. Designers can then choose to use this signal to reset their design asynchronously or synchronously inside the FPGA.

But with the help of a few hints and tips, designers will find ways to choose a more suitable reset structure. An optimal reset structure will enhance device utilization, timing and power consumption in an FPGA.

UNDERSTANDING THE FLIP-FLOP RESET BEHAVIOR

Before we delve into reset techniques, it is important to understand the behavior of flip-flops inside an FPGA slice. Devices in the Xilinx® 7 series architecture contain eight registers per slice, and all these registers are D-type flip-flops. All of these flip-flops share a common control set.

The control set of a flip-flop is the clock input (CLK), the active-high chip enable (CE) and the active-high SR port. The SR port in a flip-flop can serve as a synchronous set/reset or an asynchronous preset/clear port (see Figure 1).

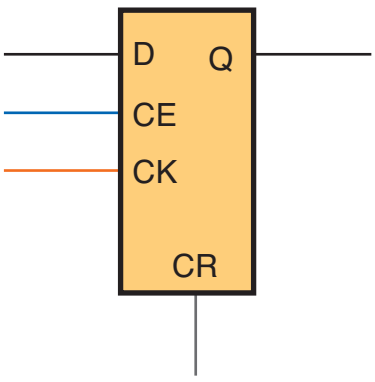

Figure 1 – Slice flip-flop control signals

The RTL code that infers the flip-flop also infers the type of reset a flip-flop will use. The code will infer an asynchronous reset when the reset signal is present in the sensitivity list of an RTL process (as shown in Figure 2a). The synthesis tool will infer a flip-flop with an SR port configured as a preset or clear port (represented by the FDCE or FDPE flip-flop primitive). When the SR port is asserted, the flip-flop output is immediately forced to the SRVAL attribute of the flip-flop.

In the case of synchronous resets, the synthesis tool will infer a flip-flop whose SR port is configured as a set or reset port (represented by an FDSE or FDRE flip-flop primitive). When the SR port is asserted, the flip-flop output is forced to the SRVAL attribute of the flip-flop on the next rising edge of the clock.

In addition, you can initialize the flip-flop output to the value the INIT attribute specifies. The INIT value is loaded into the flip-flop during configuration and when the global set reset (GSR) signal is asserted.

The flip-flops in Xilinx FPGAs can support both asynchronous and synchronous reset and set controls. However, the underlying flip-flop can natively implement only one set/reset/preset/clear at a time. Coding for more than one set/reset/preset/clear condition in the RTL code will result in the implementation of one condition using the SR port of the flip-flop and the other conditions in fabric logic, thus using more FPGA resources.

If one of the conditions is synchronous and the other is asynchronous, the asynchronous condition will be implemented using the SR port and the synchronous condition in fabric logic. In general, it's best to avoid more than one set/reset/preset/clear condition. Furthermore, only one attribute for each group of four flip-flops in a slice determines if the SR ports of flip-flops are synchronous or asynchronous.

```
signal Q:std_logic:='1';
....
async: process (CLK, RST)
begin
  if  (RST='1')then
    Q <= '0';
  else
    if (CLK'event and CLK='1') then
      Q <= D;
    end if;
  end if;
end process async;
```

(a)

INIT

SRVAL

```
signal Q:std_logic:='1';
....
sync: process (CLK)
begin
  if (CLK'event and CLK='1') then
    if (RST='1') then
      Q <= '0';
    else
      Q <= D;
    end if;
  end if;
end process sync;
```

(b)

Figure 2 – SRVAL and INIT attributes define flip-flop reset and initialization: here, VHDL code to infer asynchronous (a) and synchronous (b) reset.

RESET METHODOLOGY

Regardless of the reset type used (synchronous or asynchronous), you will generally need to synchronize the reset with the clock. As long as the duration of the global reset pulse is long enough, all the device flip-flops will enter the reset state. However, the deassertion of the reset signal must satisfy the timing requirements of the flip-flops to ensure that the flip-flops transition cleanly from their reset state to their normal state. Failure to meet this requirement can result in flip-flops entering a metastable state.

Furthermore, for correct operation of some subsystems, like state

machines and counters, all flip-flops must come out of reset on the same clock edge. If different bits of the same state machine come out of reset on different clocks, the state machine may transition into an illegal state. This reinforces the need to make the deassertion of reset synchronous to the clock.

For designs that use a synchronous reset methodology for a given clock domain, it is sufficient to use a standard metastability resolution circuit (two back-to-back flip-flops) to synchronize the global reset pin onto a particular clock domain. This synchronized reset signal can then initialize all storage elements in the clock domain by using the synchronous SR port on the flip-flops. Because both the synchronizer and the flip-flops to be reset are on the same clock domain, the standard PERIOD constraint of the clock covers the timing of the paths between them. Each clock domain in the device needs to use a separate synchronizer to generate a synchronized version of the global reset for that clock domain.

Now let's get down to brass tacks. Here are some specific hints and tips that will help you arrive at the best reset strategy for your design.

Tip 1: When driving the synchronous SR port of flip-flops, every clock domain requires its own localized version of the global reset, synchronized to that domain.

Sometimes a portion of a design is not guaranteed to have a valid clock. This can occur in systems that use recovered clocks or clocks that are sourced by a hot-pluggable module. In such cases, the storage elements in the design may need to be initialized with an asynchronous reset using the asynchronous SR port on the flip-flops. Even though the storage elements use an asynchronous SR port, the deasserting edge of the reset must still be synchronous to the clock. This requirement is characterized by the reset-recovery timing arc of the flip-flops, which is similar to a setup requirement of the deasserting edge of an asynchronous SR to the rising edge of the clock. Failure to meet this timing arc can cause flip-flops to enter a metastable state and synchronous subsystems to enter unwanted states.

The reset bridge circuit shown in Figure 3 provides a mechanism to assert reset asynchronously (and hence take effect even in the absence of a valid clock) and deassert reset synchronously. In this circuit, it is assumed that the SR ports of the two flip-flops have an asynchronous preset functionality (SRVAL=1).

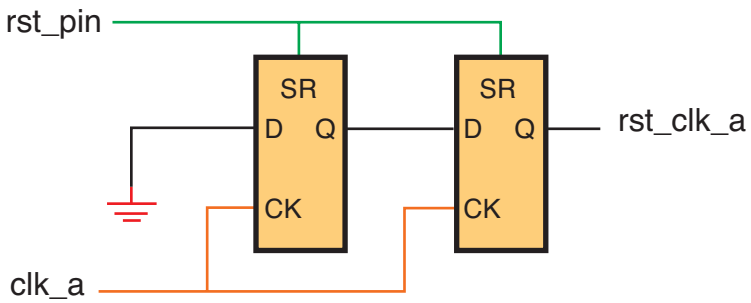

Figure 3 – Reset bridge circuit asserts asynchronously and deasserts synchronously.

You can use the output of such a reset bridge to drive the asynchronous reset for a given clock domain. This synchronized reset can initialize all storage elements in the clock domain by using the asynchronous SR port on the flip-flops. Again, each clock domain in the device needs a separate, synchronized version of the global reset generated by a separate reset bridge.

Tip 2: A reset bridge circuit provides a safe mechanism to deassert an asynchronous reset synchronously. Every clock domain requires its own localized version of the global reset with the use of a reset bridge circuit.

The circuit in Figure 3 assumes that the clock (clk_a) for clocking the reset bridge and the associated logic is stable and error free. In an FPGA, clocks can come directly from an off-chip clock source (ideally via a clock-capable pin), or can be generated internally using an MMCM or phase-locked loop (PLL). Any MMCM or PLL that you've used to generate a clock requires calibration after it is reset. Hence, you may have to insert additional logic in the global reset path to stabilize that clock.

Tip 3: Ensure that the clock the MMCM or PLL has generated is stable and locked before deasserting the global reset to the FPGA.

Figure 4 illustrates a typical reset implementation in an FPGA.

The SR control port on Xilinx registers is active high. If the RTL code describes active-low set/reset/preset/clear functionality, the synthesis tool will infer an inverter before it can directly drive the control port of a register. You must accomplish this inversion with a lookup table, thus taking up a LUT input. The additional logic that active-low control signals infers may

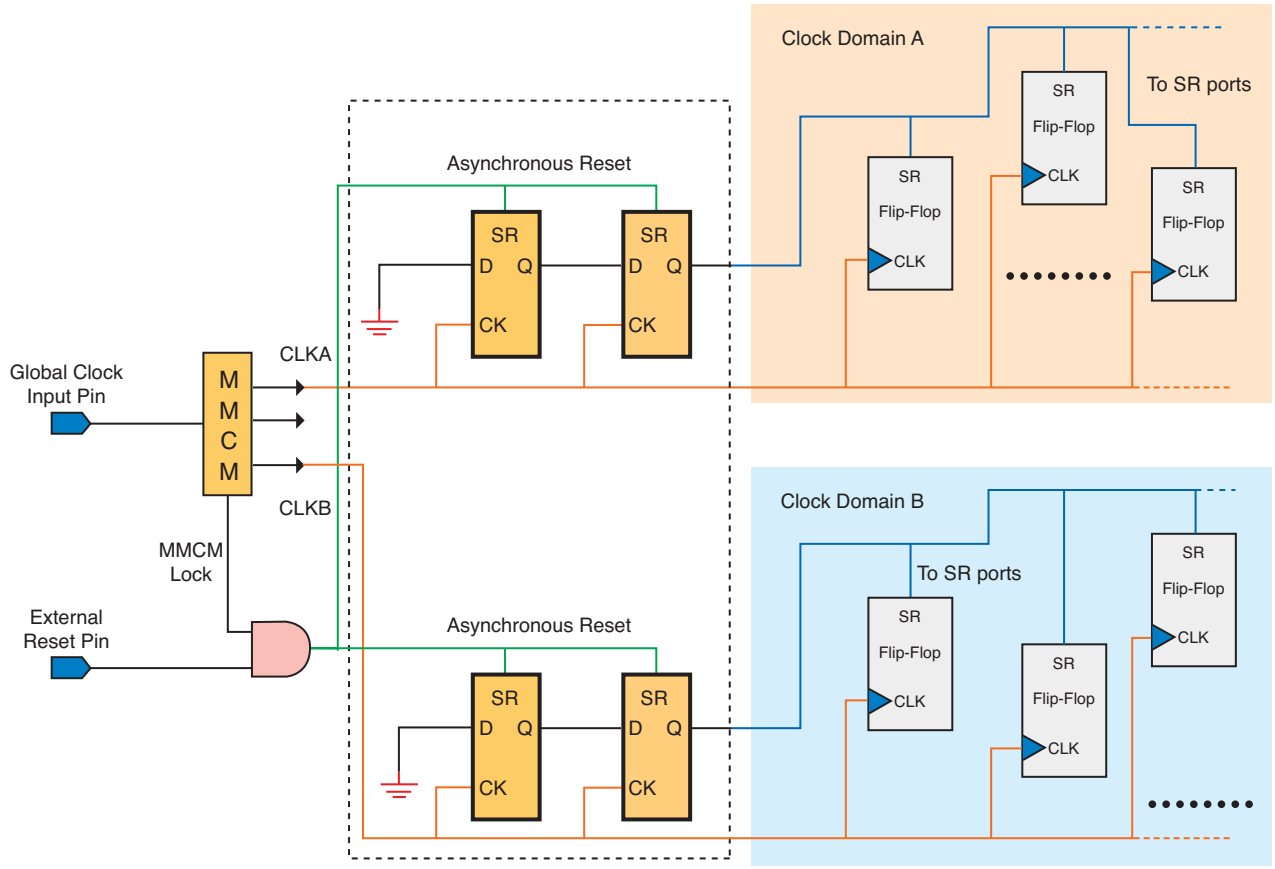

Figure 4 – Typical reset implementation in FPGAs

lead to longer runtimes and result in poorer device utilization. It will also affect timing and power.

The bottom line? Use active-high control signals wherever possible in the HDL code or instantiated components. When you cannot control the polarity of a control signal within the design, you need to invert the signal in the top-level hierarchy of the code. When described in this manner, the inferred inverter can be absorbed into the I/O logic without using any additional FPGA logic or routing.

Tip 4: Active-high resets enable better device utilization and improve performance.

It's important to note that FPGAs do not necessarily require a global reset. Global resets compete for the same routing resources as other nets in a design. A global reset would typically have high fanout because it needs to be propagated to every flip-flop in the design. This can consume a significant amount of routing resources and can have a negative impact on device utilization and timing performance. As a result, it is worth exploring other reset mechanisms that do not rely on a complete global reset.

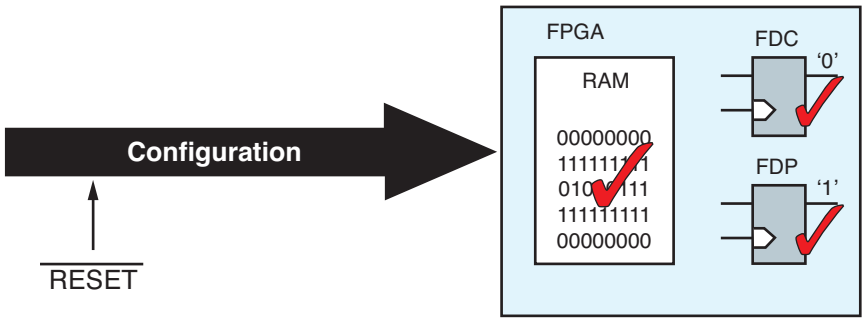

Figure 5 – FPGA initialization after configuration

When a Xilinx FPGA is configured or reconfigured, every cell (including flip-flops and block RAMs) is initialized as shown in Figure 5. Hence, FPGA configuration has the same effect as a global reset in that it sets the initial state of every storage element in the FPGA to a known state.

```
signal reg: std_logic_vector (7 downto 0) := (others <= '0');
....
process (clk) begin
   if (clk'event and clk= '1') then
     if (rst= '1') then
        reg <= '0';
     else
        reg <= D;
     end if;
   end if;
end process;
```

Figure 6 – Signal initialization in RTL code (VHDL)

You can infer flip-flop initialization values from RTL code. The example shown in Figure 6 demonstrates how to code initialization of a register in RTL. FPGA tools can synthesize initialization of the signals even though it is a common misconception that this is not possible. The initialization value of the underlying VHDL signal or Verilog reg becomes the INIT value for the inferred flip-flop, which is the value loaded into the flip-flop during configuration.

As with registers, you can also initialize block RAMs during configuration. With the increase in embedded RAMs in processor-based systems, BRAM initialization has become a useful feature. This is because a predefined RAM facilitates easier simulation setup and eliminates the requirement to have boot-up sequences to clear memory for embedded designs.

The global set reset (GSR) signal is a special prerouted reset signal that holds the design in the initial state while the FPGA is being configured. After the configuration is complete, the GSR is released and all of the flip-flops and other resources now possess the INIT value. In addition to operating it during the configuration process, a user design can access the GSR net by instantiating the STARTUPE2 module and connecting to the GSR port. Using this port, the design can reassert the GSR net, which will return all storage elements in the FPGA to the state specified by their INIT property.

The deassertion of GSR is asynchronous and can take several clocks to affect all flip-flops in the design. State machines, counters or any other logic that can change state autonomously will require an explicit reset that deasserts synchronously to the user clock. As a result, using GSR as the sole reset mechanism can result in an unreliable system.

Hence, you are better served by adopting a mixed approach to manage the startup effectively.

Tip 5: A hybrid approach that relies on the built-in initialization the GSR provides, along with explicit resets for portions of the design that can start autonomously, will result in better utilization and performance.

After using the GSR to set the initial state of the entire design, use explicit resets for logic elements, like state machines, that require a synchronous reset. Generate the synchronized version of the explicit reset using either a standard metastability resolution circuit or a reset bridge.

USE APPROPRIATE RESETS TO MAXIMIZE UTILIZATION

The style of reset used in RTL code can have a significant impact on the ability of the tools to map a design to the underlying FPGA resources. When writing RTL code, it is important that designers tailor the reset style of their subdesign to enable the tools to map to these resources.

Other than using the GSR mechanism for initialization, you cannot reset the contents of SRLs, LUTRAMs and block RAMs using an explicit reset. Thus, when writing code that is expected to map to these resources, it is important to code

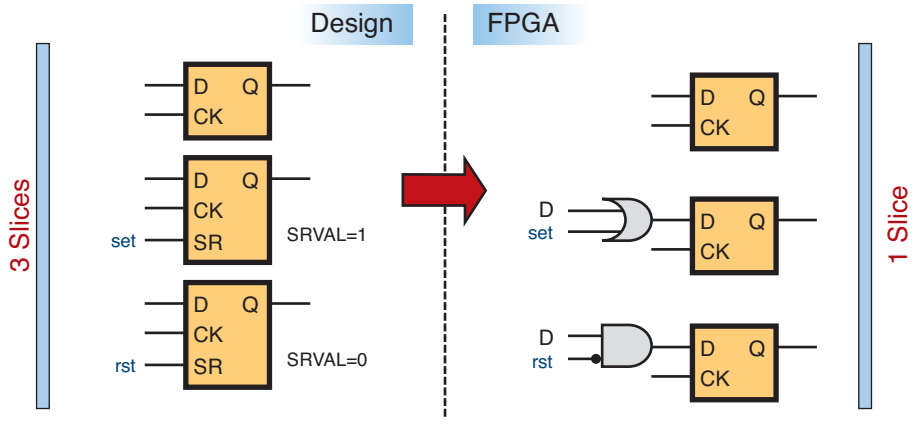

Figure 7 – Control set reduction on SR

specifically without reset. For example, if RTL code describes a 32-bit shift register with an explicit reset for the 32 stages in the shift register, the synthesis tool would not be able to map this RTL code directly to an SRL32E because it cannot meet the requirements of the coded reset using this resource. Instead, it would either infer 32 flip-flops or infer some additional circuitry around an SRL32E in order to implement the required reset functionality. Both of these solutions would require more resources than if you had coded the RTL without reset.

Tip 6: When mapping to SRLs, LUTRAMs or block RAMs, do not code for a reset of the SRL or RAM array.

In 7 series devices, you cannot pack flip-flops with different control signals into the same slice. For low-fanout resets, this can have a negative impact on overall slice utilization. With synchronous resets, the synthesis tool can implement the reset functionality using LUTs (as shown in Figure 7) rather than control ports of flip-flops, thereby removing the reset as a control port. This allows you to pack the resulting LUT/flip-flop pair with other flip-flops that do not use their SR ports. This may result in higher LUT utilization but improved slice utilization.

Tip 7: Synchronous resets enhance FPGA utilization. Use them in your designs rather than asynchronous resets.

Some of the larger dedicated resources (namely block RAMs and DSP48E1 cells) contain registers that can be inferred as part of the dedicated resource functionality. Block RAMs have optional output registers that you can use to improve clock frequency by means of an additional clock of latency. DSP48E1 cells have many registers that you can use both for pipelining, to increase maximum clock speed, as well as for cycle delays (Z-1). However, these registers only have synchronous set/reset capabilities.

Tip 8: Using synchronous resets allows the synthesis tool to use the registers inside dedicated resources like DSP48E1 slices or block RAMs. This improves overall device utilization and performance for that portion of the design, and also reduces overall power consumption.

If the RTL code describes asynchronous set/reset, then the synthesis tool will not be able to use these internal registers. Instead, it will use slice flip-flops since they can implement the requested asynchronous set/reset functionality. This will not only result in poor device utilization but will also have a negative impact on performance and power.

MANY OPTIONS

Various reset options are available for FPGAs, each with its own advantages and disadvantages. The recommendations outlined here will help designers choose a suitable reset structure for their design. An optimal reset structure will enhance the device utilization, timing and power consumption of an FPGA.

Many of the tips explained in this article are described in the Designing with the 7 Series Families training course. More information on Xilinx courses is available at *www.xilinx.com/training*.

About the Author

Srikanth Erusalagandi is currently working as a solutions development engineer on Xilinx's Global Training Solutions team. Srikanth develops content for the Xilinx training courses. His areas of expertise are FPGA design and connectivity. Prior to joining Xilinx in January 2010, he spent close to six years at MosChip Semiconductors as an applications engineer.