AN 046: Reset Guidelines for Efinix® FPGAs

AN046-v1.1
November 2022
www.efinixinc.com

Contents

Introduction

Titanium and Trion® FPGAs enters user mode after the CDONE pin goes high and t_{USER} has elapsed. Efinix recommends that you issue a reset to your design in this initial stage to:

- Ensure that the registers are cleared.
- Avoid registers capturing unintended data input due to clock glitches or unstable clocks
- If required, you can also initialize the registers with the intended values during reset.

You can use an external or internally-generated signal as the reset signal. Either way, the reset should only be released after the device has gone into user mode. This application note explains how to implement reset in your design.

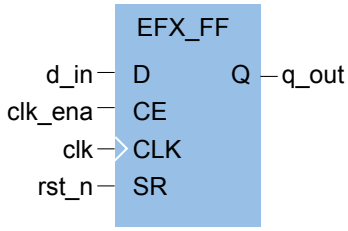
Flipflop Set/Reset

Flipflops in eXchangeable Logic and Routing (XLR) cells are used to implement register functions in synchronous designs. Through the set/reset (SR) port, flipflops support the following functions without the need for additional logic resources:

- Synchronous reset
- Synchronous set
- Asynchronous reset
- Asynchronous set

Additionally, you can configure the flipflops to expect an active-high or active-low set/reset signals. The set/reset signals can drive the SR port directly.

Figure 1: Typical Flipflop Port Connections



Learn more: Refer to the Quantum Titanium Primitives User Guide and Quantum Trion Primitives User Guide for details on the EFX_FF primitive, flipflop ports, and parameters.

The following table shows the flopflop set/reset implementations examples and the corresponding parameter settings for EFX_FF primitive.

Table 1: Flipflop Reset Implementation Examples and EFX_FF Primitive Settings

Reset Implementation	EFX_FF Parameter Setting	Verilog HDL Example	VHDL Example

Active-Low Synchronous Reset	SR_POLARITY = 0
	SR_SYNC = 1
	SR_VALUE = 0
		`always@(posedge clk)`
		`begin`
		`  if (!rst_n)`
		`  q_out <= 1'b0;`
		`  else`
		`  q_out <= d_in;`
		`end`
			`process (clk)`
			`begin`
			`  if rising_edge(clk) then`
			`  if rst_n = '0' then`
			`  q_out <= '0';`
			`  else`
			`  q_out <= d_in;`
			`  end if;`
			`  end if;`
			`end process;`

Active-High Synchronous Reset	SR_POLARITY = 1
	SR_SYNC = 1
	SR_VALUE = 0
		`always@(posedge clk)`
		`begin`
		`  if (rst)`
		`  q_out <= 1'b0;`
		`  else`
		`  q_out <= d_in;`
		`end`
			`process (clk)`
			`begin`
			`  if rising_edge(clk) then`
			`  if rst = '1' then`
			`  q_out <= '0';`
			`  else`
			`  q_out <= d_in;`
			`  end if;`
			`  end if;`
			`end process;`

Active-Low Synchronous Set	SR_POLARITY = 0
	SR_SYNC = 1
	SR_VALUE = 1
		`always@(posedge clk)`
		`begin`
		`  if (!rst_n)`
		`  q_out <= 1'b1;`
		`  else`
		`  q_out <= d_in;`
		`end`
			`process (clk)`
			`begin`
			`  if rising_edge(clk) then`
			`  if rst_n = '0' then`
			`  q_out <= '1';`
			`  else`
			`  q_out <= d_in;`
			`  end if;`
			`  end if;`
			`end process;`

Active-Low Asynchronous Reset	SR_POLARITY = 0
	SR_SYNC = 0
	SR_VALUE = 0
		`always@(posedge clk,`
		` negedge rst_n)`
		`begin`
		`  if (!rst_n)`
		`  q_out <= 1'b0;`
		`  else`
		`  q_out <= d_in;`
		`end`
			`process (clk, rst_n)`
			`begin`
			`  if rst_n = '0' then`
			`  q_out <= '0';`
			`  elsif rising_edge(clk) then`
			`  q_out <= d_in;`
			`  end if;`
			`end process;`

Active-Low Asynchronous Set	SR_POLARITY = 0
	SR_SYNC = 0
	SR_VALUE = 1
		`always@(posedge clk,`
		` negedge rst_n)`
		`begin`
		`  if (!rst_n)`
		`  q_out <= 1'b1;`
		`  else`
		`  q_out <= d_in;`
		`end`
			`process (clk, rst_n)`
			`begin`
			`  if rst_n = '0' then`
			`  q_out <= '1';`
			`  elsif rising_edge(clk) then`
			`  q_out <= d_in;`
			`  end if;`
			`end process;`

Note: Avoid implementing multiple conditions in your source code as it requires additional logic. Examples of multiple conditions include implementing both reset and set synchronously or asynchronously.

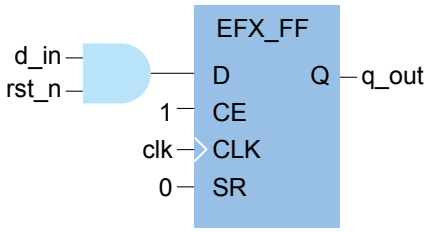
infer-sync-set-reset Synthesis Option

The reset signal can drive the flipflop SR port directly. However, the Efinity® software can also use additional resources so that the set/reset signal feeds the D input port instead of the SR port for more flexibility. To allow Efinity software to implement this feature, set the **--infer-sync-set-reset** synthesis option to 0 (disable) in the Project Editor's **Synthesis** tab.

Learn more: Refer to the Efinity Synthesis User Guide for additional details on the synthesis options.

The following figure shows the example of an additional logic implementation where the reset signal is connected to the D input port through an additional logic.

Figure 2: Reset Signal Feeding D Input Port



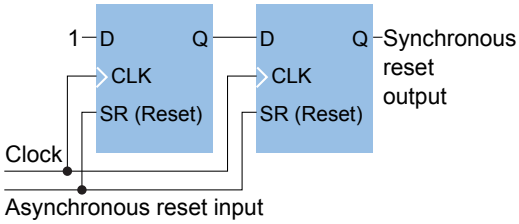
Reset Synchronizer

You can use a reset synchronizer circuitry to synchronize an external asynchronous reset signal to the clock domain of your synchronous logic design. With the reset synchronizer, the reset assertion to the flipflops is asynchronous, but the de-assertion is synchronous to the clock. The reset synchronizer can also filter glitches that might occur in an asynchronous reset signal and prevent metastability issues. Different clock domains should not share the same reset synchronizer.

The following examples describe common reset synchronizer circuitries with active-low or active-high reset input and output. You can extend the duration of the reset signal assertion by adding additional flipflop stages.

Active-Low Reset Input and Active-Low Reset Output

Figure 3: Active-Low Reset Input and Active-Low Reset Output Connections



Verilog HDL Example

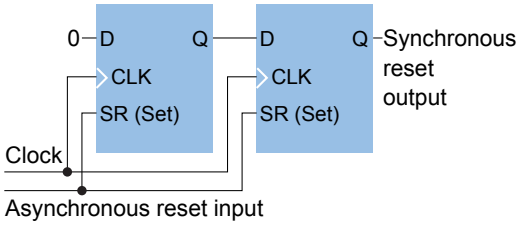
```
always@(posedge clk, negedge rst_n)
begin
    if (!rst_n)
        begin
            q_1 <= 1'b0;
            q_2 <= 1'b0;
        end
    else
        begin
            q_1 <= 1'b1;
            q_2 <= q_1;
        end
end
```

VHDL Example

```
process (clk, rst_n)
begin
    if rst_n = '0' then
        q_1 <= '0';
        q_2 <= '0';
    elsif rising_edge(clk) then
        q_1 <= '1';
        q_2 <= q_1;
    end if;
end process;
```

Active-Low Reset Input and Active-High Reset Output

Figure 4: Active-Low Reset Input and Active-High Reset Output Connections



Verilog HDL Example

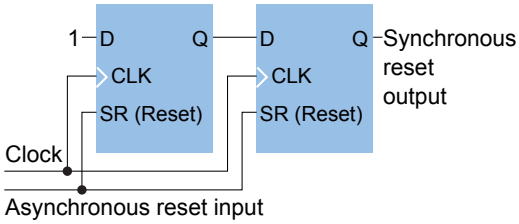
```
always@(posedge clk, negedge rst_n)
begin
    if (!rst_n)
        begin
            rst_q_1 <= 1'b1;
            rst_q_2 <= 1'b1;
        end
    else
        begin
            rst_q_1 <= 1'b0;
            rst_q_2 <= rst_q_1;
        end
end
```

VHDL Example

```
process (clk, rst_n)
    begin
        if rst_n = '0' then
            q_1 <= '1';
            q_2 <= '1';
        elsif rising_edge(clk) then
            q_1 <= '0';
            q_2 <= q_1;
        end if;
    end process;
```

Active-High Reset Input and Active-Low Reset Output

Figure 5: Active-High Reset Input and Active-Low Reset Output Connections



Verilog HDL Example

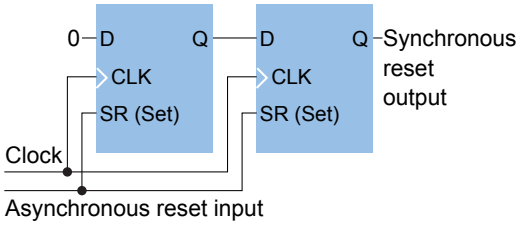
```
always@(posedge clk, posedge rst)
begin
    if (rst)
        begin
            rst_q_1 <= 1'b0;
            rst_q_2 <= 1'b0;
        end
    else
        begin
            rst_q_2 <= rst_q_1;
            rst_q_1 <= 1'b1;
        end
end
```

VHDL Example

```
process (clk, rst)
    begin
        if rst = '1' then
            q_1 <= '0';
            q_2 <= '0';
        elsif rising_edge(clk) then
            q_1 <= '1';
            q_2 <= q_1;
        end if;
    end process;
```

Active-High Reset Input and Active-High Reset Output

Figure 6: Active-High Reset Input and Active-High Reset Output Connections



Verilog HDL Example

```
always@(posedge clk, posedge rst)
begin
    if (rst)
        begin
           q_1 <= 1'b1;
           q_2 <= 1'b1;
        end
    else
        begin
            q_1 <= 1'b0;
            q_2 <= q_1;
        end
end
```

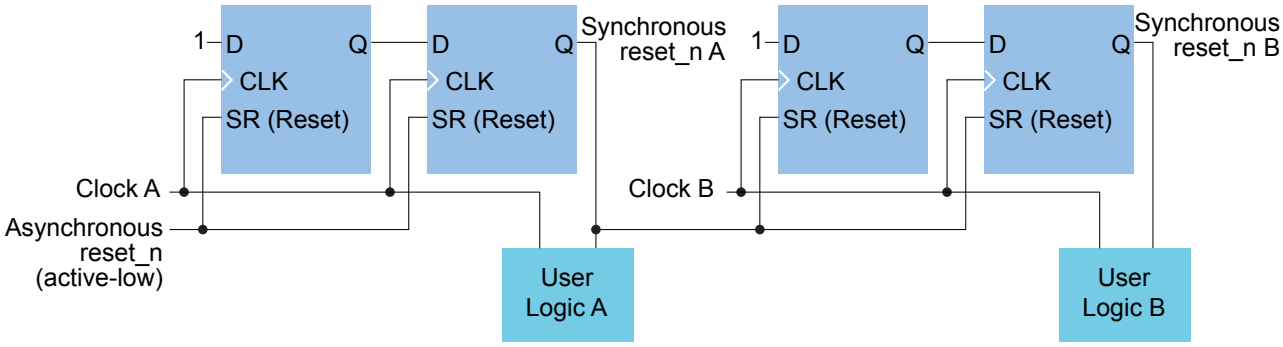
VHDL Example

```
process (clk, rst)
    begin
        if rst = '1' then
            q_1 <= '1';
            q_2 <= '1';
        elsif rising_edge(clk) then
            q_1 <= '0';
            q_2 <= q_1;
        end if;
    end process;
```

Reset Sequencing

Some designs with more than one clock domain requires the logic in different clock domains to be released from reset in a particular order. For these designs, you can implement reset sequencing. The reset sequencing is done by cascading reset synchronizers of the respective clock domains. The following figure illustrates an example where the reset for logic A under the clock domain A is released first, then followed by logic B under the clock domain B.

Figure 7: Reset Sequencing



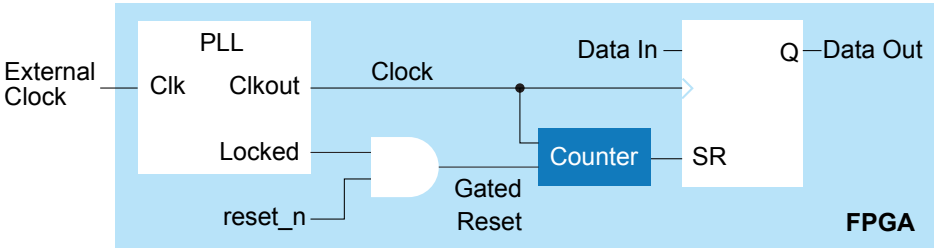
Generating Reset Signals Internally

This section describes ways to implement internally generated reset signals. You can use a PLL, a counter, or an output pin to trigger resets.

Using the PLL Locked Signal

If you use a PLL to clock the flipflops, the flipflops must be held in reset until the PLL is locked and the clock output is stable. You can use the PLL locked signal to gate the reset signal feeding the flipflops. Add a counter to hold the registers in reset even after configuration is done until t_{USER} has elapsed. The counter also acts as a reset synchronizer, ensuring the reset de-assertion is synchronous to the PLL clock output.

Figure 8: Using the PLL Locked Signal as Gated Reset



You need to calculate the number of PLL output clock cycles required to ensure the reset signal is de-asserted after t_{USER} elapsed. The following example calculates the required number of PLL output clock cycles for a Titanium FPGA (t_{USER} = 25 μs) with a PLL output clock frequency of 50 MHz.

Clock cycles required = t_{USER}/ PLL output clock period = 25 μs / (1/50 MHz) = 1,250

The counter should hold the reset for at least 1,250 clock cycles.

Verilog HDL Example

```
always@(posedge clkout, negedge rst_n)
begin
   if (!rst_n)
   begin
      rst_n_sync <= 1'b0;
      counter <= 11'b0;
   end
   else
   begin
      if (counter < 1250)
         counter <= counter + 1'b1;
      else
         rst_n_sync <= 1'b1;
   end
end
```

VHDL Example

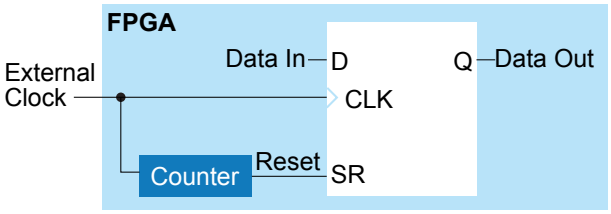
```
process (clk, rst_n)
begin
   if rst_n = '0' then
      rst_n_sync <= '0';
      counter <= (others => '0');
   elsif rising_edge(clk) then
      if (counter <1250) then
         counter <= counter + 1;
      else
         rst_n_sync <= '1';
      end if;
   end if;
end process;
```

There may be times when the PLL loses lock, or the `reset_n` signal is triggered while the FPGA is in user mode. In these situations, the reset counter restarts the count on the number of cycles to wait before releasing the registers from reset. The counter starts after the PLL regains lock or the `reset_n` signal is de-asserted.

Using a Counter

You can use a counter to generate a reset signal to hold your design in reset after configuration has completed and automatically releases the design after t_{USER} has elapsed.

Figure 9: Using a Counter to Perform a Reset

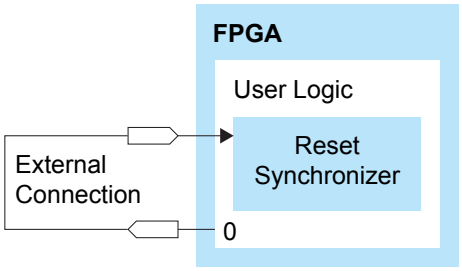


Using I/O Pins

You can also use an output pin to generate a signal that indicates when the device goes into the user mode. This signal can be used as a reset signal to your design.

Feeding the Reset Input Pin

Figure 10: Driving the Reset Input Pin with an Output Pin



Overview flow of driving the reset input pin with an output pin:

1. The I/O pins are tri-stated with an internal weak pull-up during configuration.

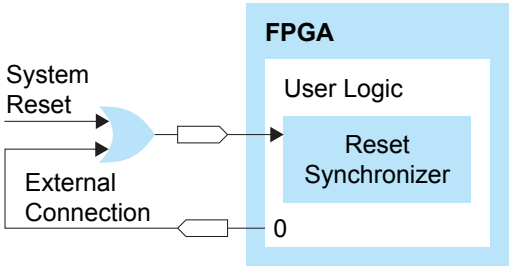
   Important: Include a weak pull-up resistor around 10 kΩ on the board if both reset input and output pins used do not support weak pull-up during configuration, for example, LVDS configured as GPIO pins in Trion FPGAs.

2. The core logic is released first when the configuration is complete, while the I/O pins are still tri-stated with weak pull-up.
3. The synchronous logic in your design is held in reset until the I/O pins are released, during which the output pin generates an asynchronous reset signal by driving the input pin low.
4. The asynchronous reset signal feeds a reset synchronizer to generate a synchronous active-high or active-low reset signal to the synchronous logic in your design.

To generate the reset signal and feed the reset input pin of your design, connect an output pin to the reset input pin at the board level. Drive this output pin low internally in your design. Before compiling your design, turn off the **Release Tri-States before Reset** programming option in the Project Editor's **Bitstream Generation** tab. Turning this option off ensures the core logic is ready before the I/O pins are released and prevents glitches at output pins when the configuration completes and the device transitions into user mode.

You can also combine the reset from the output pin with other external resets such as the system reset.

Figure 11: Driving the Reset Input Pin with an Output Pin and the System Reset

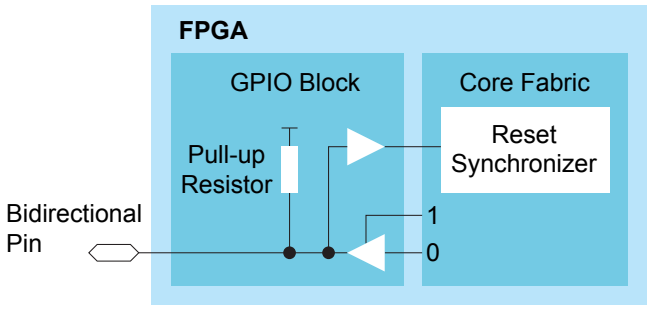


Single Pin Reset Generation

You can combine the input and output pin functions as discussed in the previous section into one single pin. To perform a reset with a single pin, add a bidirectional pin in your design in the Efinity Interface Designer. In your RTL code, drive the pin's output buffer low, and the output enable (OE) high. Connect the pin's input buffer to the reset synchronizer as the reset input.

Important: To prevent contention, do not drive this pin externally.

Figure 12: Driving the Reset with a Single Bidirectional Pin



Note: The internal weak pull-up resistor is turned on automatically during configuration. You do not need to enable the weak pull-up of the input buffer setting in the Interface Designer.

Important: LVDS configured as GPIO pins in Trion FPGAs do not have an internal weak pull-up. You must include an external 10 kΩ pull-up resistor on the board.

The following example code shows how the bidirectional pin is connected to a design with the reset driving a reset synchronizer.

Verilog HDL Example

```
module single_pin_rst (
input clk, bidir_IN,
output bidir_OUT, bidir_OE,
…
);
…
reg rst_q_1, rst_q_2;
wire bidir_OUT = 1'b0;
wire bidir_OE = 1'b1;
…
always@(posedge clk, posedge
 bidir_IN)
begin
    if (bidir_IN)
        begin
            rst_q_1 <= 1'b1;
            rst_q_2 <= 1'b1;
        end
    else
        begin
            rst_q_1 <= 1'b0;
            rst_q_2 <= rst_q_1;
        end
end
…
endmodule
```

VHDL Example

```
library IEEE;
use IEEE.std_logic_1164.all;

entity single_pin_rst is
    port (clk, bidir_IN: in
 STD_LOGIC;
    bidir_OUT: out STD_LOGIC:='0';
    bidir_OE: out STD_LOGIC:='1';
    …
    );
end single_pin_rst;

architecture reset_sync of
 single_pin_rst is
signal rst_q_1, rst_q_2: STD_LOGIC;

begin
    process (clk, bidir_IN)
    begin
        if bidir_IN = '1' then
            rst_q_1 <= '1';
            rst_q_2 <= '1';
        elsif rising_edge(clk) then
            rst_q_1 <= '0';
            rst_q_2 <= rst_q_1;
        end if;
    end process;
    …
end reset_sync;
```

Initializing Flipflop Value

The flipflops are cleared when the FPGA enters user mode. If required, you can initialize the flipflops with a logic 1 using RTL code or a reset.

Note: It is prefereable to use a reset to initialize the flipflop value because the hardware implementation does not require additional logic compared to using RTL. Additionally, using a reset allows you to re-initialize the flipflops if needed.

Initializing Flipflops with a Reset

You can use the reset signal to initialize the flipflops in your design. The following code example shows how resetting the design also sets the flipflop to a logic 1 before starting an operation.

Verilog HDL Example

```
always@(posedge clk)
begin
    if (!rst_n)
        q_out <= 1'b1;
    else
         q_out <= d_in;
end
```

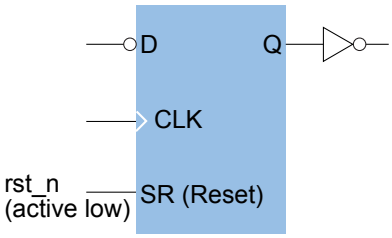
VHDL Example

```
process (clk)
begin
    if rising_edge(clk) then
        if rst_n = '0' then
            q_out <= '1';
        else
            q_out <= d_in;
        end if;
    end if;
end process;
```

Initializing Flipflops with RTL

Typically, you initialize flipflops with RTL codes for simulation but it is also synthesizable by the Efinity software. The Efinity software implements additional logic during synthesis to ensure the output is initialized to the desired value. The following figure and example codes show the implementation result with additional logic to initialize the output to logic 1.

Figure 13: Initializing Flipflops with Additional Logic in RTL



Verilog HDL Example 1

```
reg q_out = 1'b1;
always@(posedge clk)
begin
    if (!rst_n)
        q_out <= 1'b0;
    else
         q_out <= d_in;
end
```

Verilog HDL Example 2

```
initial
begin
    q_out <= 1'b1;
end

always@(posedge clk)
begin
    if (!rst_n)
        q_out <= 1'b0;
    else
         q_out <= d_in;
end
```

VHDL Example

```
signal q_out: STD_LOGIC := '1';
…
    process (clk)
    begin
        if rising_edge(clk) then
            if rst_n = '0' then
                q_out <= '0';
            else
                q_out <= d_in;
            end if;
        end if;
    end process;
```

Resetting other Blocks

Other than the flipflops, you can also reset and/or initialize other logic blocks such as the shift register, embedded memory, and DSP/multiplier block.

Note: Efinix recommends that you keep the reset asserted until the FPGA is in user mode and the clock is stable to avoid the embedded memory and DSP/multiplier blocks driving out unintended data.

8-bit Shift Registers

The EFX_SRL8 primitive in Titanium FPGAs allows you to implement an 8-bit shift register function in XLR cells. The 8-bit shift register content can be initialized with the EFX_SRL8 primitive, so the reset is not required for content initialization.

Additional logic is required if you implement the 8-bit shift register function in the XLR cell through your own RTL code.

Learn more: Refer to the **Quantum Titanium Primitives User Guide** for details on instantiating 8-bit shift register using the EFX_SRL8 primitive, and on how to initialize the shift register content.

Embedded Memory Blocks

The Titanium FPGA memory blocks support asynchronous and synchronous reset of the RAM output, and asynchronous reset of the RAM output register. The RAM output and RAM output register reset share the same reset input port but can be enabled independently. The embedded memory block in the true-dual-port mode has two reset ports, one for port A and the other for port B. Asserting the reset signal does not clear the RAM content.

You can initialize the memory block content for Trion and Titanium FPGAs when instantiating through the primitives.

Learn more: Refer to the **Quantum Titanium Primitives User Guide** for details on instantiating the embedded memory block using the EFX_RAM10 or EFX_DPRAM10 primitive and how to reset or initialize the RAM content.

Learn more: Refer to the **Quantum Trion Primitives User Guide** for details on instantiating the embedded memory block using the EFX_RAM_5K or EFX_DPRAM_5K primitive and how to initialize the RAM content.

DSP and Multiplier Blocks

The Titanium DSP blocks include a reset port that supports synchronous and asynchronous reset. This reset port can drive the reset for registers A, B, C, OP, P, W, and O. You can enable or disable the reset for each register independently. You can enable the synchronous reset and reset the registers together with other sequential logic in your design using the system reset when the FPGA goes into user mode.

The Trion multiplier blocks include reset ports to set or reset registers A, B, and O. The registers support synchronous and asynchronous reset or set. You can enable the synchronous reset and reset the registers together with other sequential logic in your design through the system reset when the FPGA goes into user mode.

Learn more: Refer to the **Quantum Titanium Primitives User Guide** for details on instantiating the DSP block using the EFX_DSP48, EFX_DSP24 or EFX_DSP12 primitive and how to enable reset for the registers.

Learn more: Refer to the **Quantum Trion Primitives User Guide** for details on instantiating the multiplier block using the EFX_MULT primitive and details on the reset/set ports.

Revision History

Table 2: Revision History

Date	Version	Description
November 2022	1.1	Fixed typo in the Verilog HDL example for Using the PLL Locked Signal. (DOC-1034)
June 2022	1.0	Initial release.