Product Bulletin

April 2024 **FPGA-PB-02001**

Workaround for Lattice ECP5 (LFE5UM) Known Issue with SerDes Interface Connections Due to Unstable Reset Soft Logic

Product Affected: All ECP5 and ECP5-5G with SerDes(LFE5UM and LFE5UM-5G) devices are covered by this Product Bulletin.

Customers Affected: Only customers using SerDes/PCS-based designs. It potentially affects several applications such as PCIe IP, g8B10B, and other PCS supported protocol. The workaround is organized on a single channel basis, contact your local Lattice representative for other use cases.

Background: The Lattice Semiconductor ECP5 FPGA devices may contain SerDes. These parts with SerDes can be implemented through Clarity Designer by using the PCS module. The PCS module comes with a custom soft logic reset, which helps users automatically reset the receiver portion of the PCS.

Observation: Lattice found that in specific situations, the original soft logic reset is not as stable as intended. Some unexpected behaviors include the Receiver CDR failing to lock and the receiver data being unstable. This document provides the workaround procedure to resolve these issues.

Detailed Workaround Description:

This workaround is organized on a single channel basis, contact your local Lattice representative for other use cases.

Lattice developed a workaround procedure that implements ‘extended Reset Soft Logic = extRSL’ to bring-up the SerDes/PCS at a stable state. The workaround is to remove the current receiver reset signals from the current RSL and add a new module that will drive the receiver reset signals instead.

The new source code contains the necessary functions for a detailed signal monitoring and controlled reset scenario. It is the single point of control for all the receiver resets in the PCS portion of the design. The transmitter reset signals will still be driven by the original RSL.

It supports the controlled reset sequencing for the different blocks, such as:

- **Default power-up situation**: Enhanced power-up reset sequence for higher stability. (*Refer to Figure 10*)
- **Receiver Clock Data Recovery block is locked and stable**. (*Refer to Figure 11*)
- **PCS block is locked and stable**. (*Refer to Figure 12*)

If these settings are detected, the design releases all resets, resulting to a stable design.

If any of the cases below occurs, the reset scenario starts from the beginning (*Refer to Figure 13*):

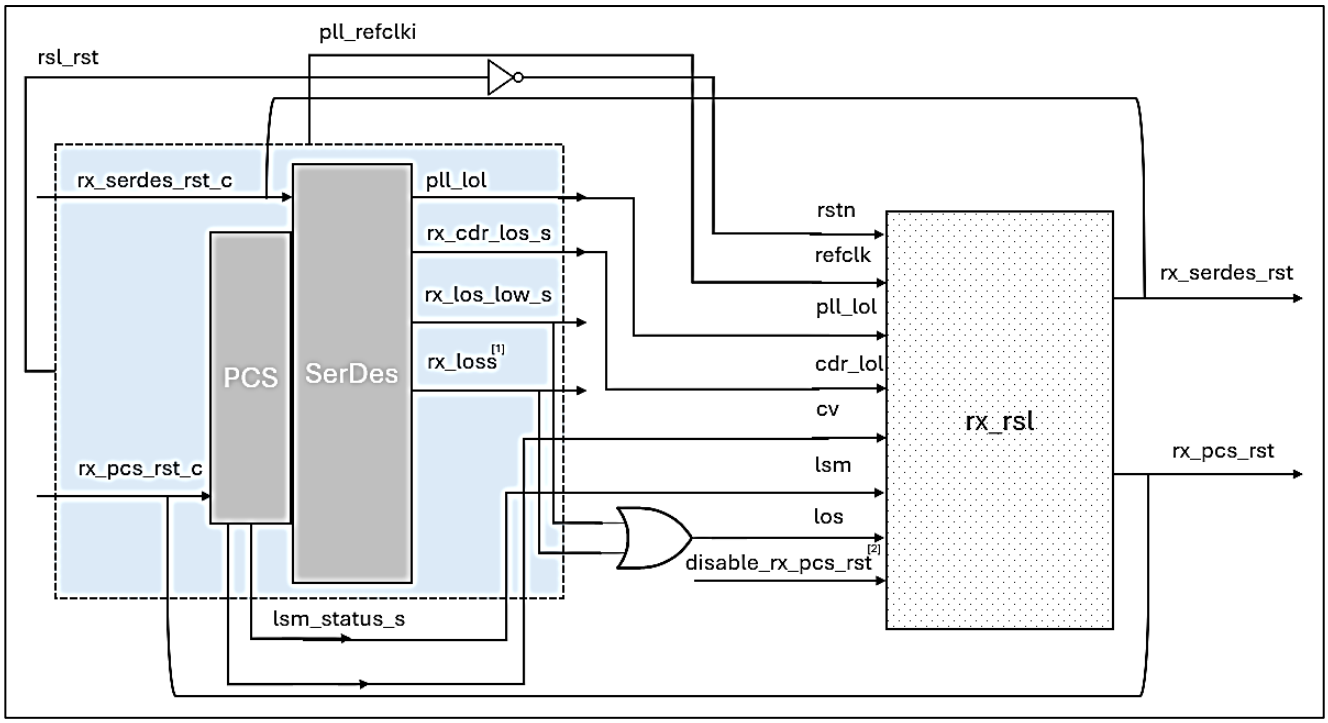
- Code violation error the FSM restarts the PCS again.
- RX loss of signal or the TX_PLL will lose its lock.

For the cascaded resets of different functions in the SerDes, the user can select default reset periods as minimum step width. These variables are counter timers for specific portions of the reset sequence and optimized for this workaround. The “Tplol” value is the amount of time

transmit PLL to lock, “Tcdr” is the amount of time receiver CDR has to lock and “Tviol” is the amount of time to wait for code violation and disparity error before concluding the PCS is correct. Note that depending on your design, these values might have to be adjusted. For example, if the reference clock used for the Transmit PLL is more jittery, the “Tplol” might have to be adjusted to a higher value. (*Refer to Figure 14*)

Note:
The extended Reset Soft Logic is instantiated into the PCS block generated from Clarity Designer. Keep in mind that once “rx_rsl” is instantiated to the PCS block, every new generation in Clarity overrides the user’s modification so make sure to save your work beforehand.

Overview Block Diagram of the Workaround:



Note:

1. Connected to the user logic. The signal should be added to the PCS module. Example: can be connected to PLL lock.
2. Debug signal.

Signal Name	Direction	Description
refclk	input	Connected to TX PLL Reference Clock.
pll_lol	input	TX PLL Loss of Lock Output.
cdr_lol	input	CDR PLL Loss of Lock Output.
cv	input	Connect this to *rx_cv_err_ch*. This is a code violation signal to indicate an error was detected with the associated data.
lsm	input	Connected to *lsm_status_s*.
los	input	Connected to *rx_los_low_s*.
disable_rx_pcs_rst	input	Test signal for debug purposes only.
rx_serdes_rst	output	Connected to the net of RX SerDes RSL signal (*rsl_rx_serdes_rst_c*)
rx_pcs_rst	output	Connected to the net of RX PCS RSL signal (rsl_rx_pcs_rst_c)

Issue Avoidance when using PCS IP:

To implement the new RSL Logic:

1. Start the PCS IP generation. After the generation, you should have the new PCS module added to your design. Then, place the PCS Module in the desired location.

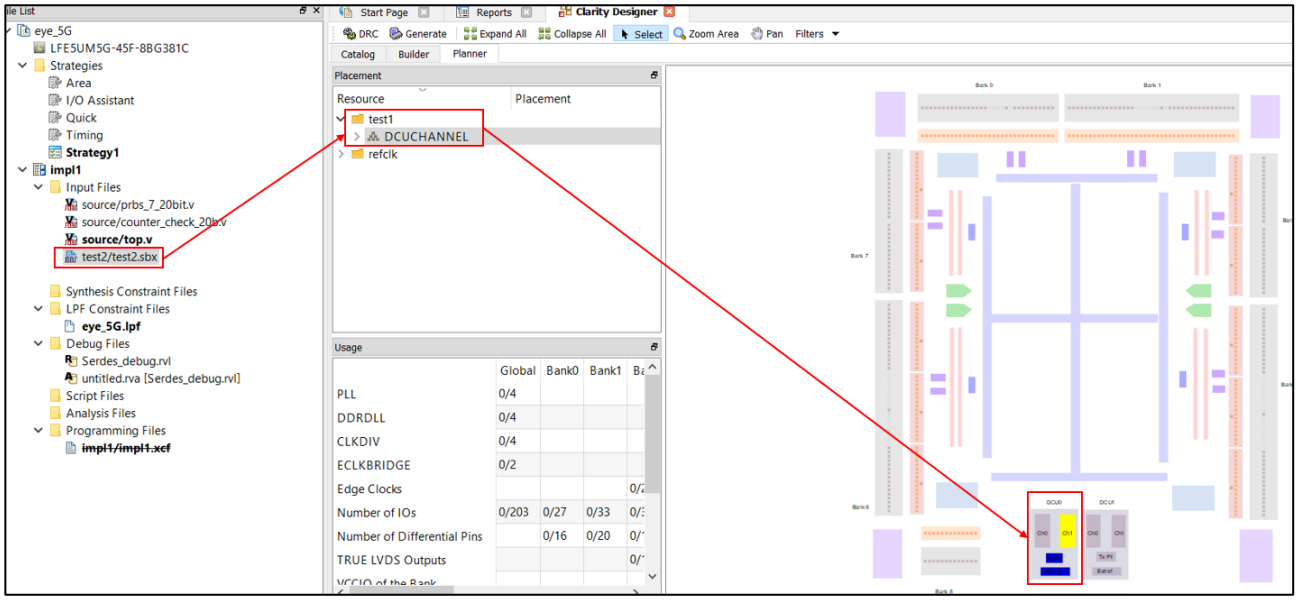


Figure 1: PCS IP assigned to DCU0 at Channel1

2. Add the new RSL logic file (rx_rsl.vhd) to the existing implementation/input files.

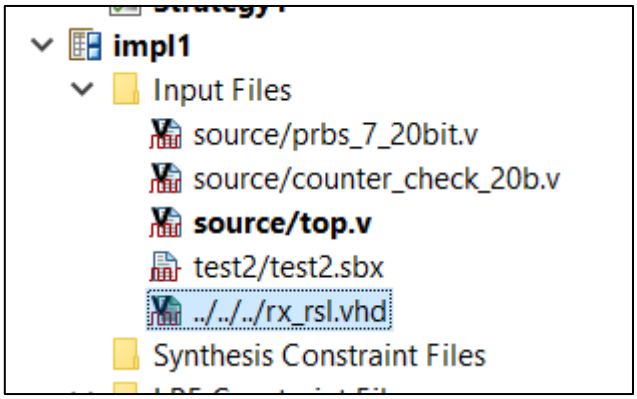


Figure 2: Addition of Source File for Extended RSL

3. Remove the current signals “rsl_rx_serdes_rst_c” and “rsl_rx_pcs_rst_c” from the PCS file and add the “rx_rsl” module instead for the RX SerDes and PCS RSL. Connections are shown below. Refer to the appendix for the source code and instantiation.

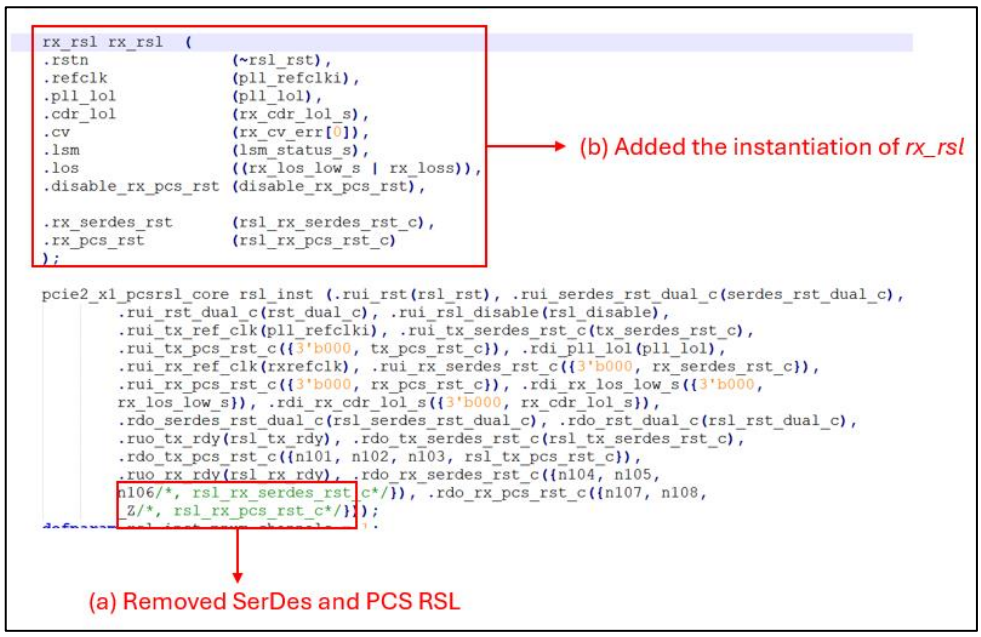


Figure 3: (a) Remove Current RSL (b) Add the rx_rsl to PCS module.

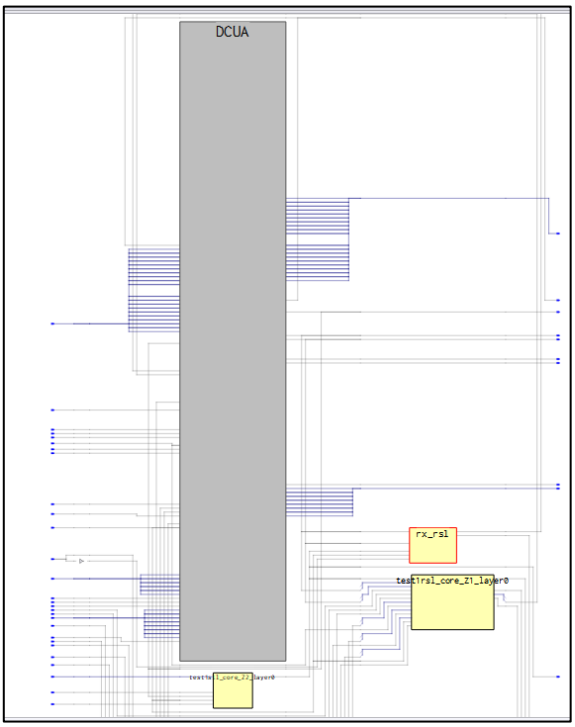


Figure 4: rx_rsl embedded into the PCS Block Generated from Clarity

Issue Avoidance when using PCIe IP:

To implement the new RSL Logic:

4. Start the PCIe IP generation. After the generation, you should have the new PCS module added to your design. Then, place the PCS Module in the desired location.

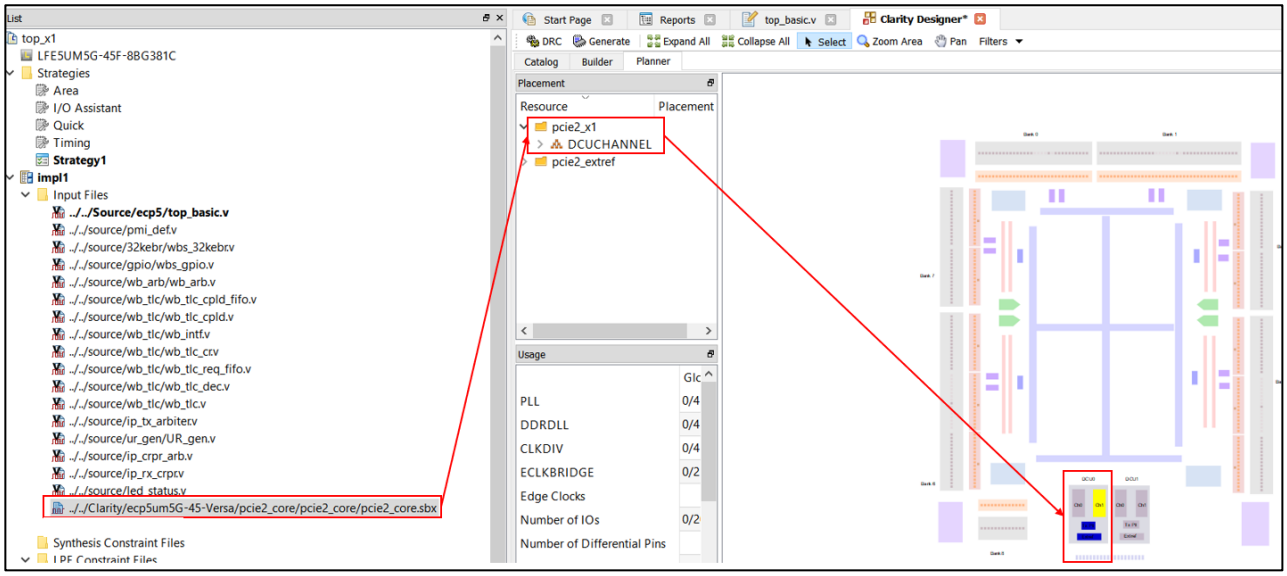


Figure 5: PCIe IP assigned to DCU0 at Channel1

5. Add the new RSL logic file (rx_rsl.vhd) to the existing implementation/input files.

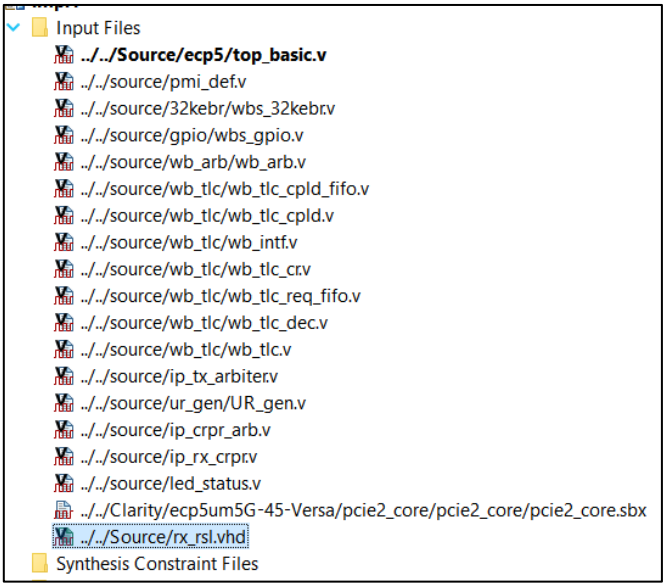


Figure 6: Addition of Source File for Extended RSL

6. Remove the current signals “rsl_rx_serdes_rst_c” and “rsl_rx_pcs_rst_c” from the PCS file and add the “rx_rsl” module instead for the RX SerDes and PCS RSL. Connections are shown below. Refer to the appendix for the source code and instantiation.

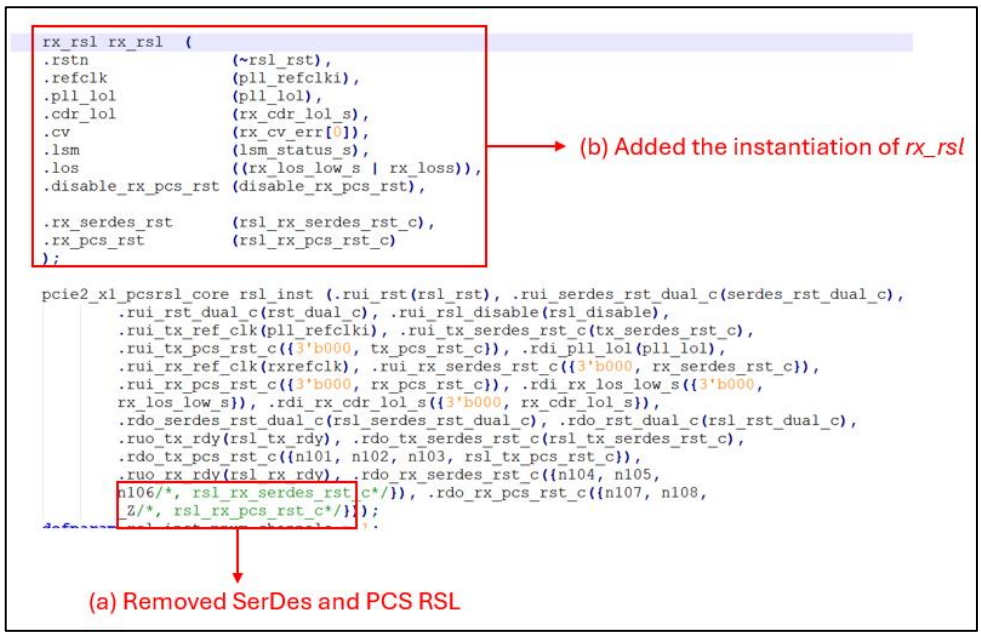


Figure 7: (a) Remove Current RSL (b) Add the rx_rsl to PCS module.

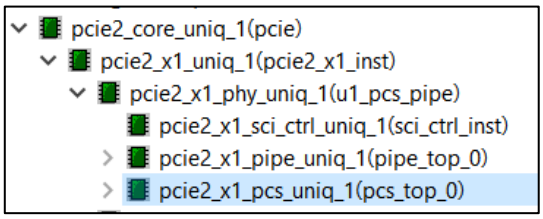


Figure 8: PCIe core to PCS sample hierarchy

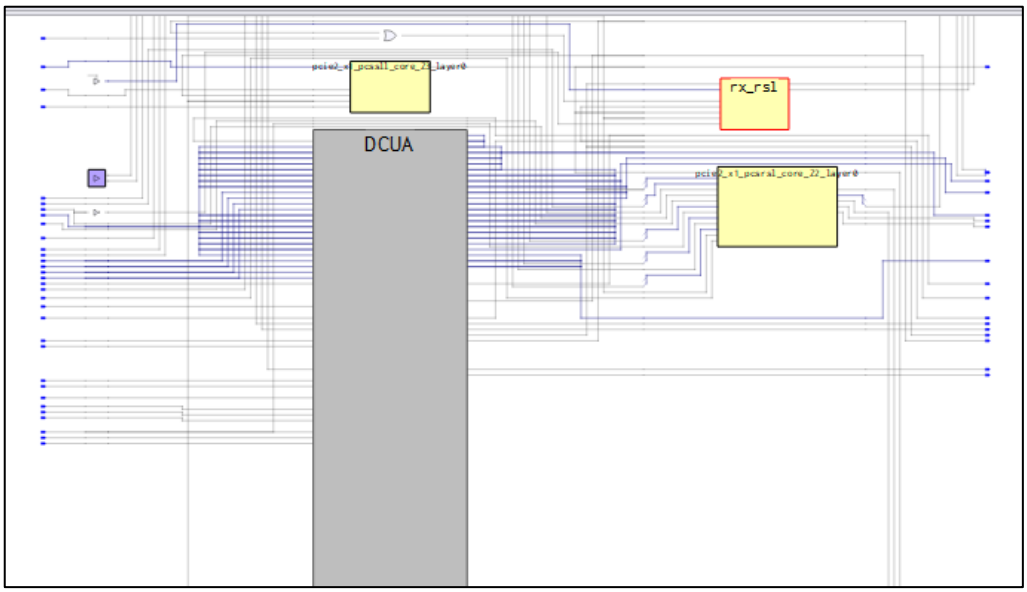


Figure 9: rx_rsl embedded into the PCS Block of the PCIe IP

Customer Acknowledgement:

By using the affected devices, the customer acknowledges that single-channel SerDes/PCS based designs have connection issues due to unstable reset soft logic and that instructions on how to avoid this scenario has been provided.

Contact:

If you have any questions or require additional information, contact Lattice Technical Support through www.latticesemi.com/techsupport.

Source code:

```
--rx_rsl.vhd content

library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

entity rx_rsl is
        port
        (
        rstn                        : in    std_logic;
        refclk                      : in    std_logic;
        pll_lol                     : in    std_logic;
        cdr_lol                     : in    std_logic;
        cv                          : in    std_logic;
        lsm                         : in    std_logic;
        los                         : in    std_logic;
        disable_rx_pcs_rst          : in    std_logic;

        rx_serdes_rst               : out   std_logic;
        rx_pcs_rst                  : out   std_logic
        );
end rx_rsl;

architecture rx_rsl_arc of rx_rsl is

attribute syn_keep : boolean;

-----------------------------------------------------------------------------------------------
--                          Constants Declaration                --
-----------------------------------------------------------------------------------------------


constant Tplol                      : std_logic_vector (31 downto 0):=x"00100000";
constant Tcdr                       : std_logic_vector (31 downto 0):=x"00100000";
constant Tviol                      : std_logic_vector (31 downto 0):=x"00100000";

-----------------------------------------------------------------------------------------------
--                                      Internal Variables                --
-----------------------------------------------------------------------------------------------
signal pll_lol_s                    : std_logic;
signal cdr_lol_s                    : std_logic;
signal cv_s                         : std_logic;
signal lsm_s                        : std_logic;
signal los_s                        : std_logic;

signal cnt                          : std_logic_vector (31 downto 0);
```

```
type rx_sm_state is (powerup, apply_cdr_rst, wait_cdr_lock, test_cdr, apply_rxpcs_rst, wait_rxpcs_lock,
test_rxpcs, idle);

signal rx_sm                            : rx_sm_state;
attribute syn_keep of rx_sm             : signal is true;

begin

------------------------------------------------------------------------------------------------
--                                  Begin Of The Design                                       --
------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------
rx_reset_proc                           : process (rstn, refclk)
begin
if rstn = '0' then
        pll_lol_s                       <= '1';
        cdr_lol_s                       <= '1';
        cv_s                            <= '1';
        lsm_s                           <= '0';
        los_s                           <= '1';

        rx_serdes_rst                   <= '1';
        rx_pcs_rst                      <= '1';

        rx_sm                           <= powerup;
        cnt                             <= (others => '0');
elsif rising_edge(refclk) then
        pll_lol_s                       <= pll_lol;
        cdr_lol_s                       <= cdr_lol;
        cv_s                            <= cv;
        lsm_s                           <= lsm;
        los_s                           <= los;

        case rx_sm is
        when powerup =>
                rx_serdes_rst           <= '1';
                rx_pcs_rst              <= '1' AND NOT(disable_rx_pcs_rst);
                if (pll_lol_s = '1') or (los_s = '1') then
                        cnt             <= (others => '0');
                else
                        if (cnt = Tplol) then
                                cnt      <= (others => '0');
                                rx_sm    <= apply_cdr_rst;
                        else
                                cnt      <= cnt + '1';
                        end if;
                end if;
------------------------------------------------------------------------------------------------
```

```
        when apply_cdr_rst =>
                rx_serdes_rst           <= '1';
                rx_pcs_rst              <= '1' AND NOT(disable_rx_pcs_rst);
                if (cnt = x"00000007") then
                        cnt             <= (others => '0');
                        rx_sm           <= wait_cdr_lock;
                else
                        cnt             <= cnt + '1';
                end if;
        when wait_cdr_lock =>
                rx_serdes_rst           <= '0';
                rx_pcs_rst              <= '1' AND NOT(disable_rx_pcs_rst);
                if (cnt = Tcdr) then
                        cnt             <= (others => '0');
                        rx_sm           <= test_cdr;
                else
                        cnt             <= cnt + '1';
                end if;
        when test_cdr =>
                rx_serdes_rst           <= '0';
                rx_pcs_rst              <= '1' AND NOT(disable_rx_pcs_rst);
                if (cdr_lol_s = '1') then
                        cnt             <= (others => '0');
                        rx_sm           <= apply_cdr_rst;
                else
                        if (cnt = Tcdr) then
                                cnt     <= (others => '0');
                                rx_sm   <= apply_rxpcs_rst;
                        else
                                cnt      <= cnt + '1';
                        end if;
                end if;
-----------------------------------------------------------------
        when apply_rxpcs_rst =>
                rx_serdes_rst           <= '0';
                rx_pcs_rst              <= '1' AND NOT(disable_rx_pcs_rst);
                if (cnt = x"00000007") then
                        cnt             <= (others => '0');
                        rx_sm           <= wait_rxpcs_lock;
                else
                        cnt             <= cnt + '1';
                end if;
        when wait_rxpcs_lock =>
                rx_serdes_rst           <= '0';
                rx_pcs_rst              <= '0';
                if (cnt = Tviol) then
                        cnt             <= (others => '0');
                        rx_sm           <= test_rxpcs;
                else
                        cnt             <= cnt + '1';
                end if;
```

```
        when test_rxpcs =>
                rx_serdes_rst           <= '0';
                rx_pcs_rst              <= '0';
                if (lsm_s = '0') or (cv_s = '1') then
                        cnt             <= (others => '0');
                        rx_sm           <= apply_rxpcs_rst;
                else
                        if (cnt = Tviol) then
                                cnt     <= (others => '0');
                                rx_sm   <= idle;
                        else
                                cnt     <= cnt + '1';
                        end if;
                end if;
-----------------------------------------------------------------
        when idle =>
                rx_serdes_rst   <= '0';
                rx_pcs_rst              <= '0';
                if (lsm_s = '0') or (cv_s = '1') then
                        rx_sm           <= apply_rxpcs_rst;
                        cnt             <= (others => '0');
                end if;
        end case;


        if (pll_lol_s = '1') or (los_s = '1') then
                rx_sm                   <= powerup;
                cnt                     <= (others => '0');
        end if;

end if;
end process rx_reset_proc;
-----------------------------------------------------------------

end rx_rsl_arc;
```

rx_rsl contents definitions:

```
-------------------------------------------------------------
rx_reset_proc       :   process (rstn, refclk)
begin
if rstn = '0' then
    pll_lol_s           <= '1';
    cdr_lol_s           <= '1';
    cv_s                <= '1';
    lsm_s               <= '0';
    los_s               <= '1';

    rx_serdes_rst       <= '1';
    rx_pcs_rst          <= '1';

    rx_sm               <= powerup;
    cnt                 <= (others => '0');
elsif rising_edge(refclk) then
    pll_lol_s           <= pll_lol;
    cdr_lol_s           <= cdr_lol;
    cv_s                <= cv;
    lsm_s               <= lsm;
    los_s               <= los;

    case rx_sm is
    when powerup =>
        rx_serdes_rst   <= '1';
        rx_pcs_rst      <= '1';
        if (pll_lol_s = '1') or (los_s = '1') then
            cnt         <= (others => '0');
        else
            if (cnt = Tplol) then
                cnt     <= (others => '0');
                rx_sm   <= apply_cdr_rst;
            else
                cnt     <= cnt + '1';
            end if;
        end if;
-------------------------------------------
```

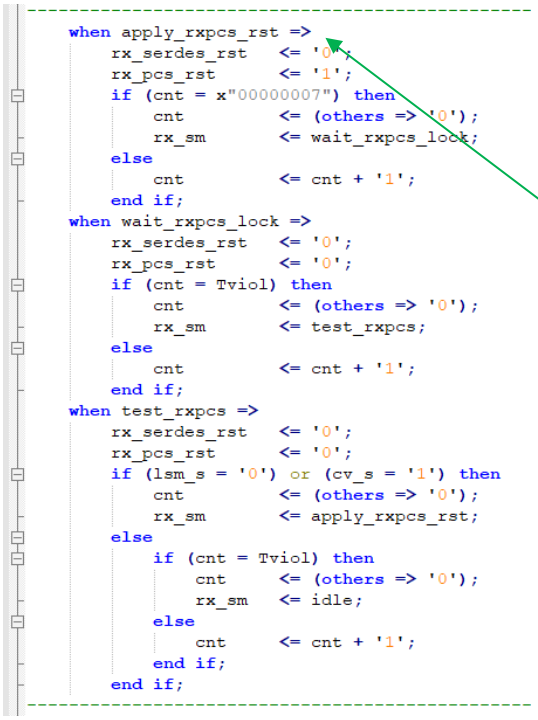
- rx_rsl.vhd
  - The controlled reset sequencing for the different blocks such as:
    - Default power up situation: (PLL is stable and RX loss of signal is detected)
    - Clock data recovery block is locked and stable.
    - PCS block is locked and stable.
  - If these settings are detected, the design releases all resets and becomes stable.
  - However, in case of any code violation error, the FSM restarts the PCS again; and
  - If any RX loss of signal or the TX_PLL will lose its lock, then the reset scenario starts from the beginning.

Figure 10: Default Power-up Situations

```
-----------------------------------------------
    when apply_cdr_rst =>
        rx_serdes_rst   <= '1';
        rx_pcs_rst      <= '1';
        if (cnt = x"00000007") then
            cnt         <= (others => '0');
            rx_sm       <= wait_cdr_lock;
        else
            cnt         <= cnt + '1';
        end if;
    when wait_cdr_lock =>
        rx_serdes_rst   <= '0';
        rx_pcs_rst      <= '1';
        if (cnt = Tcdr) then
            cnt         <= (others => '0');
            rx_sm       <= test_cdr;
        else
            cnt         <= cnt + '1';
        end if;
    when test_cdr =>
        rx_serdes_rst   <= '0';
        rx_pcs_rst      <= '1';
        if (cdr_lol_s = '1') then
            cnt         <= (others => '0');
            rx_sm       <= apply_cdr_rst;
        else
            if (cnt = Tcdr) then
                cnt     <= (others => '0');
                rx_sm   <= apply_rxpcs_rst;
            else
                cnt     <= cnt + '1';
            end if;
        end if;
-----------------------------------------------
```

- rx_rsl.vhd
  - The controlled reset sequencing for the different blocks such as:
    - Default power up situation: (PLL is stable and RX loss of signal is detected)
    - Clock data recovery block is locked and stable.
    - PCS block is locked and stable.
  - If these settings are detected, the design releases all resets and becomes stable.
  - However, in case of any code violation error, the FSM restarts the PCS again; and
  - If any RX loss of signal or the TX_PLL l its lock, then the reset scenario starts from the beginning.

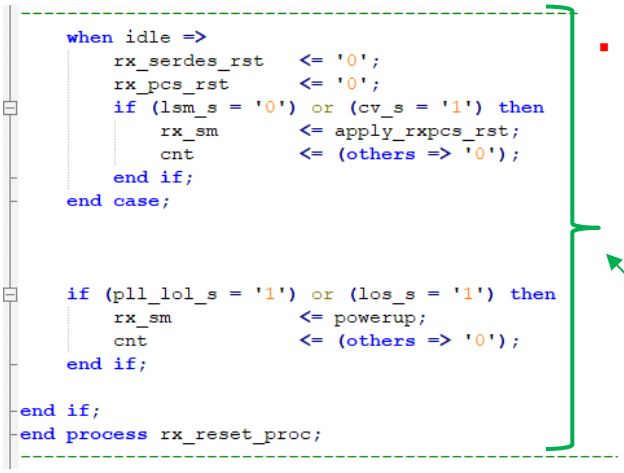
Figure 11: Clock Data Recovery Block is Locked and Stable



```
    when apply_rxpcs_rst =>
        rx_serdes_rst    <= '0';
        rx_pcs_rst       <= '1';
        if (cnt = x"00000007") then
            cnt          <= (others => '0');
            rx_sm        <= wait_rxpcs_lock;
        else
            cnt          <= cnt + '1';
        end if;
    when wait_rxpcs_lock =>
        rx_serdes_rst    <= '0';
        rx_pcs_rst       <= '0';
        if (cnt = Tviol) then
            cnt          <= (others => '0');
            rx_sm        <= test_rxpcs;
        else
            cnt          <= cnt + '1';
        end if;
    when test_rxpcs =>
        rx_serdes_rst    <= '0';
        rx_pcs_rst       <= '0';
        if (lsm_s = '0') or (cv_s = '1') then
            cnt          <= (others => '0');
            rx_sm        <= apply_rxpcs_rst;
        else
            if (cnt = Tviol) then
                cnt      <= (others => '0');
                rx_sm    <= idle;
            else
                cnt      <= cnt + '1';
            end if;
        end if;
```

- rx_rsl.vhd
  - The controlled reset sequencing for the different blocks such as:
    - Default power up situation: (PLL is stable and RX loss of signal is detected)
    - Clock data recovery block is locked and stable.
    - PCS block is locked and stable.
  - If these settings are detected, the design releases all resets and becomes stable.
  - However, in case of any code violation error, the FSM restarts the PCS again; and
  - If any RX loss of signal or the TX_PLL loses its lock, then the reset scenario start from the beginning.

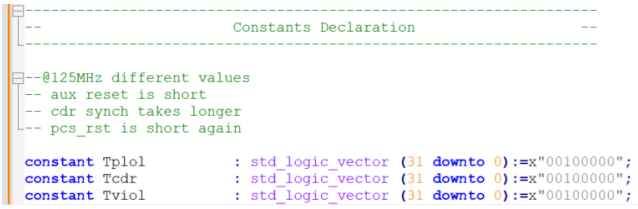
Figure 12: PCS Block is Locked and Stable



```
    when idle =>
        rx_serdes_rst    <= '0';
        rx_pcs_rst       <= '0';
        if (lsm_s = '0') or (cv_s = '1') then
            rx_sm        <= apply_rxpcs_rst;
            cnt          <= (others => '0');
        end if;
    end case;



    if (pll_lol_s = '1') or (los_s = '1') then
        rx_sm            <= powerup;
        cnt              <= (others => '0');
    end if;

end if;
end process rx_reset_proc;
```

- rx_rsl.vhd
  - The controlled reset sequencing for the different blocks such as:
    - Default power up situation: (PLL is stable and RX loss of signal is detected)
    - Clock data recovery block is locked and stable.
    - PCS block is locked and stable.
  - If these settings are detected, the design releases all resets and becomes stable.
  - However, in case of any code violation error, the FSM restarts the PCS again; and
  - If any RX loss of signal or the TX_PLL loses its lock, then the reset scenario starts from the beginning.

Figure 13: Use in Case of Any Code Violation Error



```
--                      Constants Declaration                      --

--@125MHz different values
-- aux reset is short
-- cdr synch takes longer
-- pcs_rst is short again

constant Tplol          : std_logic_vector (31 downto 0):=x"00100000";
constant Tcdr           : std_logic_vector (31 downto 0):=x"00100000";
constant Tviol          : std_logic_vector (31 downto 0):=x"00100000";
```

- rx_rsl.vhd
  - For the cascaded resets of different functions in the SerDes, user can select default reset periods as minimum step width.

Figure 14: Reset Periods as Minimum Step Width

rx_rsl Instantiation:

```
rx_rsl rx_rsl  (
   .rstn                 (~rsl_rst),
   .refclk               (pll_refclki),
   .pll_lol              (pll_lol),
   .cdr_lol              (rx_cdr_lol_s),
   .cv                   (rx_cv_err[0]),
   .lsm                  (lsm_status_s),
   .los                  ((rx_los_low_s | rx_loss)),
   .disable_rx_pcs_rst   (disable_rx_pcs_rst),

   .rx_serdes_rst        (rsl_rx_serdes_rst_c),
   .rx_pcs_rst           (rsl_rx_pcs_rst_c)
);
```

Revision History

Revision 1.1, April 2024

Section	Change Summary
All	• Updated the affected customers, parameter names, and some terminologies • Added the block diagram of the workaround • Separated the steps for PCS IP & PCIe IP

Revision 1.0, June 2021

Section	Change Summary
All	Initial release

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS and with all faults, and all risk associated with such information is entirely with Buyer. Buyer shall not rely on any data and performance specifications or parameters provided herein. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. No Lattice products should be used in conjunction with mission- or safety-critical or any other application in which the failure of Lattice's product could create a situation where personal injury, death, severe property or environmental damage may occur. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.