



Trion PLL Auto-Reset Core User Guide

UG-CORE-PLL-AUTORESET-v1.0
August 2024
www.efinixinc.com



Contents

- Introduction..... 3**
- Features.....3**
- Device Support..... 3**
- Resource Utilization and Performance.....4**
- Release Notes..... 4**
- Functional Description.....5**
 - Ports..... 5
 - Parameters..... 5
 - Reset Timing.....6
- IP Manager..... 7**
- Customizing the Trion PLL Auto-Reset..... 8**
- Trion PLL Auto-Reset Example Design..... 9**
- Revision History.....10**

Introduction

The Trion PLL Auto-Reset core improves the PLL functionality by efficiently resetting the PLLs. Trion FPGAs have phase-locked loops (PLLs) situated at the corners of the FPGA, which are used to generate clock signals for your design. You can use PLLs to compensate for clock skew or delay through external or internal feedback, thus meeting the timing requirements for advanced applications.

Use the IP Manager to select IP, customize it, and generate files. The Trion PLL Auto-Reset core has an interactive wizard to help you set parameters. The wizard also has options to create a testbench and/or example design targeting an Efinix® development board.



Important: You need an open-source Java 64-bit runtime environment when generating the Trion PLL Auto-Reset core in the IP Manager. You can get the installer and instructions from one of these sources:

- <https://www.java.com/en/download/manual.jsp> (Java 8)
- <https://developers.redhat.com/products/openjdk/download> (OpenJDK 8 or 11)
- <http://jdk.java.net/16/> (OpenJDK 16)

Features

The Trion PLL Auto-Reset core includes the following features:

- Checking on the PLL lock status after the reset is triggered
- Reset pulse triggering after time-out

Device Support

Table 1: Trion PLL Auto-Reset Core Device Support

FPGA Family	Supported Device
Trion	All except T4F49, T4F81, T8F49, and T8F81
Titanium	-

Resource Utilization and Performance



Note: The resources and performance values provided are based on some of the supported FPGAs. These values are just guidance and can change depending on the device resource utilization, design congestion, and user design.

Table 2: Trion® Resource Utilization and Performance

FPGA	FSM Count	Clock	Logic Elements (Logic, Adders, Flipflops, etc.)	Memory Block	Multiplier Block	Efinity® Version ⁽¹⁾
T20 F256 C4	1	1/16 (6.3%)	106/19,728 (0.5%)	0/204 (0%)	0/36 (0%)	2024.1
	8	1/16 (6.3%)	710/19,728 (3.6%)	0/204 (0%)	0/36 (0%)	

Release Notes

You can refer to the IP Core Release Notes for more information about the IP core changes. The IP Core Release Notes are available in the [Efinity Downloads](#) page under each Efinity software release version.



Note: You must be logged in to the Support Center to view the IP Core Release Notes.

⁽¹⁾ Using Verilog HDL.

Functional Description

The Trion PLL Auto-Reset core applies to user-instantiated PLLs. The core uses the PLL's reset (`RSTN`) and locked (`LOCKED`) signals. From the core's perspective, the `LOCKED` pin is an input from PLL after the core's reset signal goes high. If the `LOCKED` pin remains low after exceeding t_{LOCK} duration, the core outputs an active-low pulse signal to the PLL's `RSTN` pin to reset the PLL.

There are 2 methods to implement the monitoring function, which includes a finite state machine (FSM).

- Instantiate one FSM for each PLL.
- Allow several PLLs to share the same FSM to reduce resource utilization. Sharing an FSM can increase the time it takes for the PLL to lock because each PLL operates independently, which may cause some PLLs to lose lock after the other PLLs are locked.

Figure 1: Trion PLL Auto-Reset Core Block Diagram



Ports

Table 3: Clock and Reset Interface

All signals are clocked by `clk_gen`

Port	Direction	Description
<code>i_pll_rstn</code>	Input	Active-low core reset signal.
<code>i_pll_locked</code>	Input	Lock signal from PLLs.
<code>o_pll_rstn</code>	Output	PLLs reset signal to PLLs.
<code>ro_clk</code>	Output	For debug purpose only.

Parameters

Table 4: Parameters

Parameter	Option	Description
<code>N_FSM</code>	Integer	Number of FSM used to monitor the lock signal.

Reset Timing

Timing Waveform

The following figure shows the typical timing waveform.

Figure 2: Typical Timing Waveform

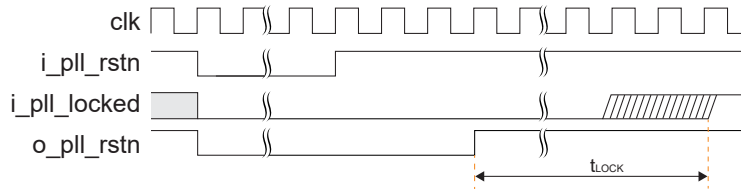
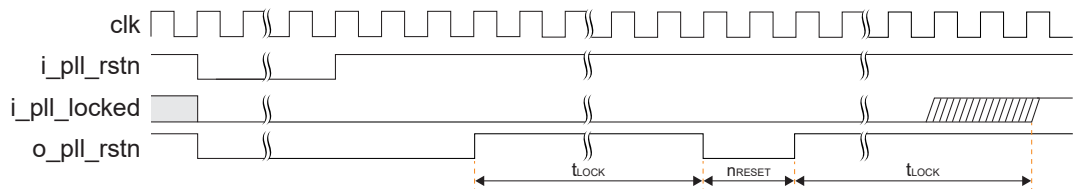


Figure 3: Auto-reset Timing Waveform



Timing Parameters

Table 5: PLL Reset Specification

Parameter	Description	Min	Typ	Max	Unit
f_{RING_OSC}	Ring oscillator frequency.	20	25	30	MHz
t_{LOCK}	PLL lock-in time ⁽²⁾ .	0.5	0.6	0.75	ms
n_{RESET}	PLL reset pulse width in clock cycle.	-	2	-	cycle



Note: You may observe that the PLL could not lock in time after a reset. The design addresses this condition by asserting an active-low reset pulse to PLL automatically. In a worst-case scenario, it may require a longer time before the PLL locks.

⁽²⁾ Refer device datasheet for PLL lock in time, t_{LOCK}

IP Manager

The Efinity® IP Manager is an interactive wizard that helps you customize and generate Efinity® IP cores. The IP Manager performs validation checks on the parameters you set to ensure that your selections are valid. When you generate the IP core, you can optionally generate an example design targeting an Efinity development board and/or a testbench. This wizard is helpful in situations in which you use several IP cores, multiple instances of an IP core with different parameters, or the same IP core for different projects.



Note: Not all Efinity IP cores include an example design or a testbench.

Generating the Trion PLL Auto-Reset Core with the IP Manager

The following steps explain how to customize an IP core with the IP Configuration wizard.

1. Open the IP Catalog.
2. Choose **Foundation IP > Trion PLL Auto-Reset** core and click **Next**. The **IP Configuration** wizard opens.
3. Enter the module name in the **Module Name** box.



Note: You cannot generate the core without a module name.

4. Customize the IP core using the options shown in the wizard. For detailed information on the options, refer to the *Customizing the Trion PLL Auto-Reset* section.
5. (Optional) In the **Deliverables** tab, specify whether to generate an IP core example design targeting an Efinity® development board and/or testbench. These options are turned on by default.
6. (Optional) In the **Summary** tab, review your selections.
7. Click **Generate** to generate the IP core and other selected deliverables.
8. In the **Review configuration generation** dialog box, click **Generate**. The Console in the **Summary** tab shows the generation status.



Note: You can disable the **Review configuration generation** dialog box by turning off the **Show Confirmation Box** option in the wizard.

9. When generation finishes, the wizard displays the **Generation Success** dialog box. Click **OK** to close the wizard.

The wizard adds the IP to your project and displays it under **IP** in the Project pane.

Generated Files

The IP Manager generates these files and directories:

- **<module name>_define.vh**—Contains the customized parameters.
- **<module name>_tpl.v**—Verilog HDL instantiation template.
- **<module name>_tpl.vhd**—VHDL instantiation template.
- **<module name>.v**—IP source code.
- **settings.json**—Configuration file.
- **<kit name>_devkit**—Has generated RTL, example design, and Efinity project targeting a specific development board.



Note: Refer to the [Efinity Software User Guide](#) for more information on Efinity IP Manager.

Customizing the Trion PLL Auto-Reset

The core has parameters so you can customize its function. You set the parameters in the **General** tab of the core's IP Configuration window.

Table 6: Trion PLL Auto-Reset Core Parameters (General Tab)

Parameter	Options	Description
FSM Count	1 - 8	Number of FSMs required. Several PLLs can share a single FSM on the condition that the start and stop times of each PLL are not a concern. Refer to Note in option 1 on page 9 for more details. Default: 2

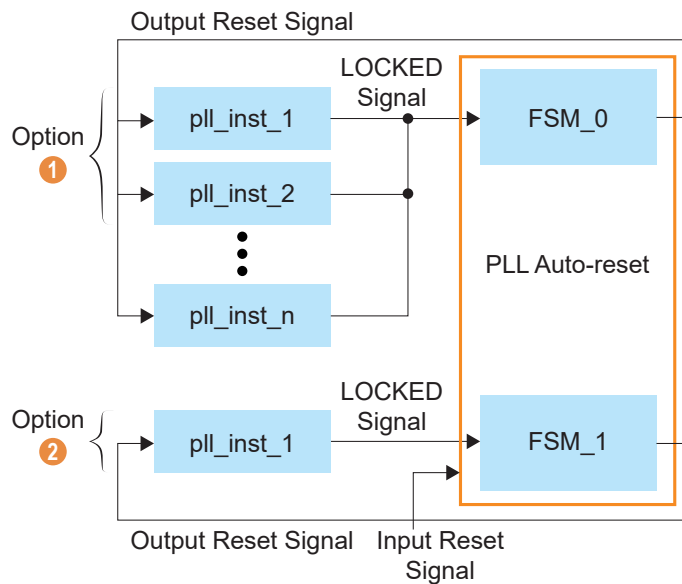
Trion PLL Auto-Reset Example Design

You can choose to generate the example design when generating the core in the IP Manager Configuration window. Compile the example design project and download the **.hex** or **.bit** file to your board. To generate example design, the **Optional Signals** option must be enabled.



Important: Efinix tested the example design generated with the default parameter options only.

Figure 4: Trion PLL Auto-Reset Core Example Design



The example design targets the Trion® T20 BGA256 Development Board.

As the figure shows, there are 2 ways to use the PLL Auto-Reset Core in your design:

- Option 1: Select one FSM for several instantiated PLLs to reduce resource utilization. You can share one FSM with all PLLs in the device if desired.



Note: You can ANDs the PLL's LOCKED signals. If one of the PLLs cannot lock within t_{LOCK} , the IP resets all PLLs. It continues to trigger reset pulse until all the PLLs are locked within t_{LOCK} after the reset pulse. If all PLLs cannot lock when using one FSM, you can create more FSMs and assign fewer PLLs to them.

- Option 2: You can select one FSM for each instantiated PLL.

The Trion PLL Auto-Reset core monitors the PLL's LOCKED signal and resets the PLL. It feeds a reset pulse signal back to the PLL if the PLL's LOCKED signal pin is not driven high.

To implement the example design:

1. Create a new project.
2. Generate the IP with `FSM_Count` set to 2.
3. Open the project file (.xml) in `<path-to-project>\ip\<ip-module-name>\T20F256_devkit\pll_autoreset.xml`.
4. Compile the design.
5. Download the design to your Trion® T20 BGA256 Development Board.

After configuration, the LED D3, D4, and D5 light up when the PLL LOCKED signal is driven high. Press the SW4 switch to reset the PLL. The PLLs LOCKED signal goes low.

Table 7: Example Design Implementation

FPGA	FSM Count	Clock	Logic Elements (Logic, Adders, Flipflops, etc.)	Memory Block	Multiplier Block	Efinity® Version ⁽³⁾
T20 F256 C4	2	5/16 (6.3%)	197/19,728 (0.5%)	0/204 (0%)	0/36 (0%)	2024.1

Revision History

Table 8: Revision History

Date	Version	Description
August 2024	1.0	Initial release. (PT-2166) Added important note in Example Design and Testbench regarding using default parameters options only. (DOC-1781)

⁽³⁾ Using Verilog HDL.