



FIFO Generator (Beta Release)

Version 0.1

November 23, 2023

Copyright

Copyright © 2021 Rapid Silicon. All rights reserved. This document may not, in whole or part, be reproduced, modified, distributed, or publicly displayed without prior written consent from Rapid Silicon ("Rapid Silicon").

Trademarks

All Rapid Silicon trademarks are as listed at www.rapidsilicon.com. Synopsys and Synplify Pro are trademarks of Synopsys, Inc. Aldec and Active-HDL are trademarks of Aldec, Inc. Modelsim and Questa are trademarks or registered trademarks of Siemens Industry Software Inc. or its subsidiaries in the United States or other countries. All other trademarks are the property of their respective owners.

Disclaimers

NO WARRANTIES: THE INFORMATION PROVIDED IN THIS DOCUMENT IS "AS IS" WITHOUT ANY EXPRESS OR IMPLIED WARRANTY OF ANY KIND INCLUDING WARRANTIES OF ACCURACY, COMPLETENESS, MERCHANTABILITY, NONINFRINGEMENT OF INTELLECTUAL PROPERTY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL RAPID SILICON OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER (WHETHER DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL, INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OF OR INABILITY TO USE THE INFORMATION PROVIDED IN THIS DOCUMENT, EVEN IF RAPID SILICON HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME JURISDICTIONS PROHIBIT THE EXCLUSION OR LIMITATION OF CERTAIN LIABILITY, SOME OF THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU.

Rapid Silicon may make changes to these materials, specifications, or information, or to the products described herein, at any time without notice. Rapid Silicon makes no commitment to update this documentation. Rapid Silicon reserves the right to discontinue any product or service without notice and assumes no obligation to correct any errors contained herein or to advise any user of this document of any correction if such be made. Rapid Silicon recommends its customers obtain the latest version of the relevant information to establish that the information being relied upon is current and before ordering any products.

Contents

| | |
|---|-----------|
| IP Summary | 3 |
| Introduction | 3 |
| Features | 3 |
| Overview | 4 |
| FIFO Generator | 4 |
| IP Specification | 5 |
| Synchronous versus Asynchronous Clock | 6 |
| Built-in FIFO versus Distributed Memory | 6 |
| Symmetric versus Asymmetric Data Width | 6 |
| Pessimistic Full and Empty Conditions | 7 |
| Standards | 7 |
| IP Support Details | 8 |
| Parameters | 8 |
| Port List | 9 |
| Resource Utilization | 10 |
| Design Flow | 11 |
| IP Customization and Generation | 11 |
| Parameters Customization | 12 |
| Synthesis and PR | 13 |
| Test Bench | 13 |
| Release | 16 |
| Release History | 16 |

IP Summary

Introduction

The FIFO Generator IP offers a superior alternative to custom Verilog-based designs, streamlining the development of high-quality FIFOs. It incorporates years of expertise, best practices, and optimizations for easy integration.

Beyond fundamental FIFO operations, this IP boasts configurable asymmetric data widths, adjustable depths, and synchronous/asynchronous clock domain crossing options. This flexibility empowers designers to tailor the FIFO Generator IP to specific requirements, making it indispensable across a wide range of applications.

Designers choose this IP to overcome challenges in data flow and synchronization. Its pre-established architecture expedites development, sparing engineers from intricate custom design nuances. The IP's versatility is evident in applications, from complex communication protocols to intricate memory management, prioritizing data integrity and synchronization. By adhering to the FIFO principle, technical systems can streamline processes, optimize resource utilization, and achieve systematic organization of data and tasks. This manual provides a comprehensive understanding of FIFO's implementation in Raptor Design Suite, empowering users across various technical domains.

Features

- Configurable FIFO Depth from 3 - 523264.
- Configurable Data Width from 1 - 1024 bits.
- Support for Synchronous and Asynchronous Clocks.
- Support for programmable flags that indicates when the FIFO is empty and full.
- Support for implementation on either Built-in FIFO or Distributed Memory.
- Support for First-Word-Fall-Through or Standard Mode.
- Native standard FIFO interface.
- Built-in Status Indicators.
- Limited Asymmetric Data Widths supported with Built-in FIFO.
- Dynamic selection between half or full Built-in FIFO bank based on the chosen configuration.

Overview

FIFO Generator

The FIFO Generator is a dedicated hardware module for effortless integration of FIFO functionality into digital systems. It offers versatile configuration options, including First-Word-Fall-Through, Programmable Full/Empty Flags, and support for Asynchronous Write/Reads with Asymmetric Data Widths.

- First-Word-Fall-Through enables the reading device to know in advance what the next chunk of data looks like without issuing a pop command, and hence increasing the robustness and data integrity of the FIFO.
- Asynchronous Reads and Writes in the FIFO facilitate seamless operation in systems with distinct clock domains. Employing a pair of Flip Flop Synchronizers ensures synchronization without encountering meta-stability issues in a two-clock domain system, ensuring optimal FIFO efficiency.
- There are a couple of programmable empty and programmable full signals the thresholds of which the user can modify accordingly. This provides a greater level of control over the status of FIFO memory, combined with the usual signals of Empty and Full.
- Asymmetric data widths in FIFOs empower designers to seamlessly integrate components with varying data processing capabilities, providing adaptability to diverse data types and optimizing system-level performance.

The IP also provides the option to select the implementation utilizing either Built-in FIFO or Distributed Memory making the FIFO generator a versatile and FPGA optimized IP Core. A block diagram for a top level FIFO generated from the FIFO Generator IP is shown in Figure 1.

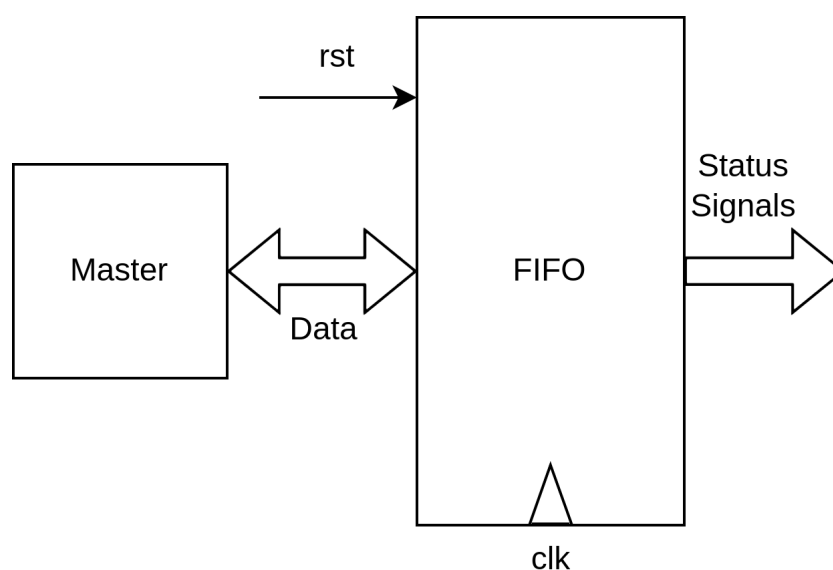


Figure 1: FIFO Block Diagram

IP Specification

The FIFO Generator is armed with advanced algorithms making the FIFO in a circular buffer ensuring that the no empty place is gone unnoticed. The synchronous pointers work on binary logic while the asynchronous pointers work on a gray code encoding logic so that the synchronizers can work without creating a meta stable condition. Because of this reason, the configurable depth for the Distributed RAM in the asynchronous mode is not continuous but rather in the exponents of 2. This is to ensure that the gray encoding logic does not break. This limitation is mitigated in the Built-in FIFO design of the asynchronous FIFO where the pointers are converted back to binary before giving them to their respective counters and hence the configurable depth is continuous from 3 - 523264 in the Symmetric Mode.

By offering support for both Built-in FIFO and Distributed Memory implementations, users gain the flexibility to choose between mapping the FIFO memory onto the Built-in FIFO modules or utilizing the logic/LUTs of the FPGA board. This versatility makes the FIFO generator IP suitable for a wide range of application requirements. The synchronous reset ensures a cycle between read and write operations, safeguarding against potential data loss in transitional cycles. A more detailed description of the internal workings of the FIFO can be read from [here](#), the publication of **Clifford E. Cummings** in his publication titled "**Simulation and Synthesis Techniques for Asynchronous FIFO Design**". An internal macro block diagram for the FIFO IP can be seen in Figure 2 that is referenced from the publication of Clifford E. Cummings linked earlier.

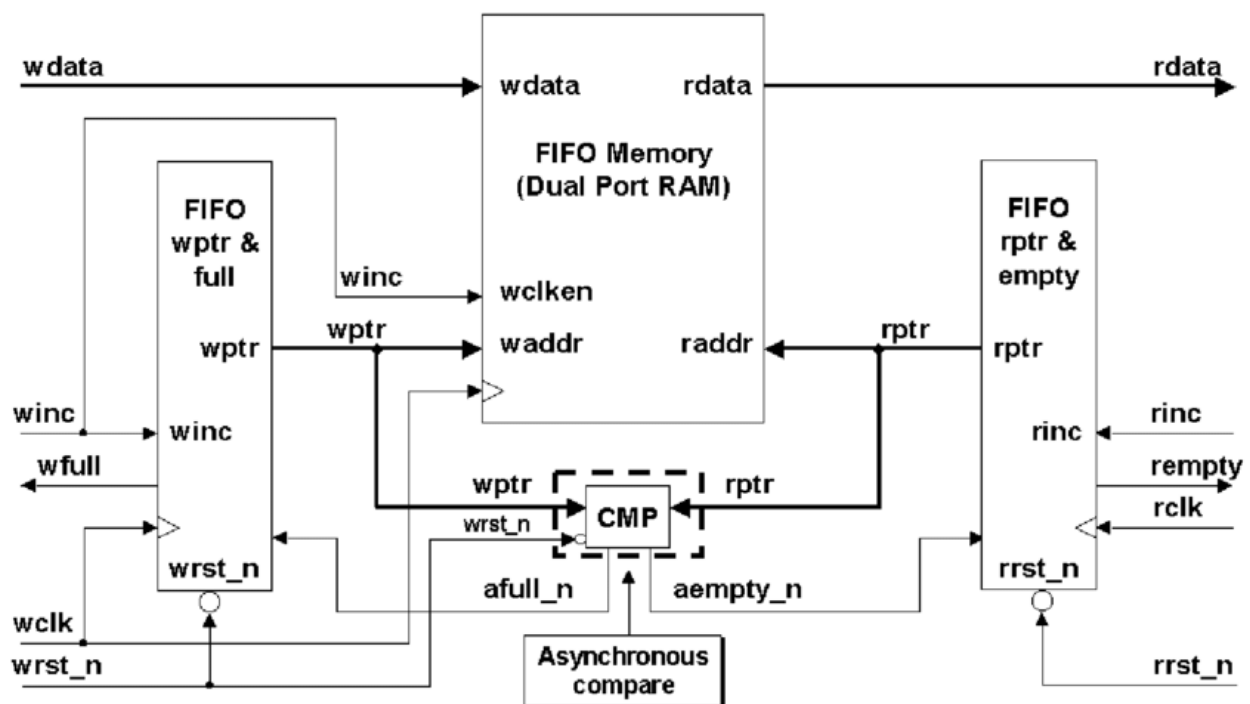


Figure 2: FIFO Internal Diagram

Synchronous versus Asynchronous Clock

Synchronous clock refers to a clock signal that is shared among all the memory blocks in the device. In contrast, an Asynchronous clock refers to a clock signal that is unique to each memory block.

- The use of a Synchronous clock in memory blocks has several advantages. First, it simplifies the design and implementation of the memory blocks as there is only one clock signal to operate. Secondly, it ensures that all the memory blocks are synchronized, which is critical for high-speed data transfer and avoiding errors. However, the disadvantage of using a Synchronous clock is that it may limit the flexibility of the design, as all the memory blocks must operate at the same clock frequency.
- The use of an Asynchronous clock in memory blocks allows for more flexibility in the design, as each memory block can operate at a different clock frequency. This can be useful in designs that require multiple memory blocks with different timing requirements. However, the disadvantage of using an Asynchronous clock is that it increases the complexity of the design, as each clock signal must be operated independently, and timing issues may arise.

Overall, the choice between using a Synchronous clock or an Asynchronous clock in memories using FIFO Generator depends on the specific requirements of the design. If the design requires all memory blocks to operate at the same frequency, a Synchronous clock may be the best choice. However, if the design requires flexibility in timing requirements, an Asynchronous clock may be more appropriate.

Built-in FIFO versus Distributed Memory

Two types of memory blocks can be generated with FIFO Generator, i.e., Built-in FIFO and Distributed RAM. Built-in FIFO uses the Block RAM which is a large, rectangular array of memory cells optimized for high-speed access, while Distributed RAM is a flexible set of smaller, distributed memory cells that are suitable for storing small chunks of data. The choice between these two types of memory blocks depends on the specific requirements of the design.

Symmetric versus Asymmetric Data Width

Symmetric and asymmetric data widths in FIFOs represent two distinct approaches to managing data flow in digital systems. Symmetric data widths involve uniform bit sizes for both the read and write interfaces, providing simplicity and ease of integration. This uniformity ensures straightforward data handling, but it might lead to under-utilization or over-utilization of resources when the application demands different widths for reading and writing. On the other hand, asymmetric data widths allow flexibility by accommodating varying bit sizes for read and write interfaces. This versatility optimizes resource utilization, making it suitable for scenarios where the data width requirements differ. However, the complexity of managing asymmetry introduces additional considerations in design and synchronization. The choice between symmetric and asymmetric data widths depends on the specific needs of the application, weighing simplicity against flexibility and resource efficiency. Due to this complexity, the asymmetric data width mode only supports limited number of data widths, i.e., 9, 18, 36, 72, 144, 288 and 576 whereas the data width is

continuous from 3 to 523264 in the symmetric mode. The introduction of asymmetry in data widths in FIFOs introduces a more intricate operational structure, particularly in scenarios where the read data width exceeds the write data width. This asymmetry leads to a pipelined operation, and it comes at the cost of requiring additional clock cycles, the actual amount of clock cycles required can be seen in the Raptor Design Suite while configuring the IP. In practical terms, when utilizing FIFOs with asymmetric data widths in cases where the read data width surpasses the write data width, the operation becomes inherently more complex and necessitates a higher number of clock cycles for effective execution. While this extended operation demands more clock cycles, it offers the advantage of optimizing hardware resources.

Pessimistic Full and Empty Conditions

The empty signal of the FIFO is de-asserted when at least one chunk of data is available to read. When using the Built-in FIFO module, it requires one clock cycle for processing this data and making it available for output. Thus, an additional clock cycle is needed after the first data chunk is written into the memory. The asynchronous mode inherently introduces this delay due to synchronizers, resulting in a two-cycle delay. Additionally, in asymmetric operation, the empty signal de-asserts when one complete word, considering the read data width, is written into the memory. Similarly, the Full signal de-asserts when one complete word is read from the FIFO, taking into account the write data width. This calls for the implementation of "Pessimistic Empty and Pessimistic Full" implementation of these status signals in accordance with the workings of *Clifford E. Cummings* in his publication *Simulation and Synthesis Techniques for Asynchronous FIFO Design*. This implies that the Full and Empty signals are asserted promptly but experience a two-clock cycle delay during de-assertion. This delay is consistent across both Symmetric and Asymmetric modes. Even in Symmetric mode, where the delay is introduced for the memory to process data before it's ready for reading, which also aligns it with the asynchronous mode.

Note:

- *The pessimistic implementation of the empty and full signals holds true exclusively in cases where the Built-in FIFO is employed or when Distributed Memory is used in the Synchronous Mode.*

Standards

The FIFO Generator soft IP supports the native interface with the standard DATA_IN and DATA_OUT ports along with the supporting clocks and reset signals in the port list.

IP Support Details

The Table 1 gives the support details for FIFO Generator.

| Compliance | | IP Resources | | | | | Tool Flow | | |
|------------|-----------|--------------|-----------------|-----------|------------------|-----------------|-------------------------|-----------------|-----------|
| Device | Interface | Source Files | Constraint File | Testbench | Simulation Model | Software Driver | Analyze and Elaboration | Simulation | Synthesis |
| GEMINI | Native | Verilog | - | Verilog | VVP | Iverilog | Raptor (Surelog) | Raptor (Icarus) | Raptor |

Table 1: IP Details

Parameters

Table 2 lists the parameters of the FIFO Generator.

| Parameter | Values | Default Value | Description |
|-------------------------|------------------------------|---------------|--|
| DEPTH | 2 - 523264 | 1024 | FIFO Depth |
| DATA WIDTH | 1 - 1024 | 36 | FIFO Write / Read Data Width in Symmetric Mode |
| FULL VALUE | 2 - (DEPTH - 1) | 1000 | Programmable Full Value |
| EMPTY VALUE | 1 - (DEPTH - 1) | 20 | Programmable Empty Value |
| SYNCHRONOUS | 0 / 1 | 1 | Synchronous / Asynchronous Implementation |
| FIRST WORD FALL THROUGH | 0 / 1 | 0 | First Word Fall Through Support |
| FULL THRESHOLD | 0 / 1 | 0 | Programmable Full Enable |
| EMPTY THRESHOLD | 0 / 1 | 0 | Programmable Empty Enable |
| BUILTIN FIFO | 0 / 1 | 1 | Distributed RAM / Built-in FIFO |
| ASYMMETRIC | 0 / 1 | 0 | Symmetric or Asymmetric Read and Write Data Widths |
| DATA WIDTH READ | 9, 18, 36, 72, 144, 288, 576 | 36 | Read Data Width for Asymmetric Access |
| DATA WIDTH WRITE | 9, 18, 36, 72, 144, 288, 576 | 36 | Write Data Width for Asymmetric Access |

Table 2: Parameters

Note:

- Both the **FULL VALUE** and **EMPTY VALUE** parameters are dependent on the **DEPTH** of the FIFO and are configured accordingly.
- The **DATA WIDTH READ** and **DATA WIDTH WRITE** parameters are only accessible when in **Built-in FIFO** and **ASYMMETRIC** mode.
- The ranges for **DEPTH** and **DATA WIDTHS** are dependent on each other.

Port List

Table 3 lists the top interface ports of the FIFO Generator.

| Signal Name | I / O | Description |
|------------------------|-------|--|
| din {DATA_WIDTH_WRITE} | I | Data Input |
| dout {DATA_WIDTH_READ} | O | Data Output |
| rst | I | Reset |
| wr_en | I | Write Enable |
| rd_en | I | Read Enable |
| full | O | FIFO Full |
| empty | O | FIFO Empty |
| underflow | O | FIFO Underflow |
| overflow | O | FIFO Overflow |
| prog_full | O | FIFO Programmable Full |
| prog_empty | O | FIFO Programmable Empty |
| rd_clk | I | Read Clock for Asynchronous Operation |
| wrt_clk | I | Write Clock for Asynchronous Operation |
| clk | I | Common Clock for Synchronous Operation |

Table 3: FIFO Generator Interface

Note:

- *DATA_WIDTH* is the bit-width of the data bus of the FIFO generated.
- *prog_full* and *prog_empty* depends on the *FULL THRESHOLD* and *EMPTY THRESHOLD* parameters respectively.
- *rd_clk* and *wrt_clk* depends on *SYNCHRONOUS* parameter being off, otherwise *clk* would be in the portlist for *SYNCHRONOUS* mode.

Resource Utilization

The parameters for computing the maximum and minimum resource utilization are given in Table 4, remaining parameters have been kept at their default values.

| Tool | Raptor Design Suite | | | |
|------------------|-------------------------|---------------|----------------------|----------|
| FPGA Device | GEMINI | | | |
| Configuration | | | Resource Utilization | |
| Minimum Resource | Options | Configuration | Resources | Utilized |
| | DATA WIDTH | 1 | FIFO | 0.5 |
| | DEPTH | 3 | | |
| | FULL THRESHOLD | 0 | | |
| | EMPTY THRESHOLD | 0 | LUTs | 17 |
| | BUILT-IN FIFO | 1 | | |
| | SYNCHRONOUS | 1 | Registers | 11 |
| | FIRST WORD FALL THROUGH | 0 | | |
| Maximum Resource | Options | Configuration | Resources | Utilized |
| | DATA WIDTH | 128 | FIFO | 127.5 |
| | DEPTH | 34816 | | |
| | FULL VALUE | 1000 | LUTs | 6158 |
| | EMPTY VALUE | 20 | | |
| | BUILT-IN FIFO | 1 | Registers | 469 |
| | SYNCHRONOUS | 0 | | |
| | FIRST WORD FALL THROUGH | 1 | Carry Adders | 34 |
| | ASYMMETRIC | 0 | | |

Table 4: Resource Utilization

Design Flow

IP Customization and Generation

FIFO Generator IP core is a part of the Raptor Design Suite Software. A customized FIFO IP can be generated from the Raptor's IP configurator window as shown in Figure 3.

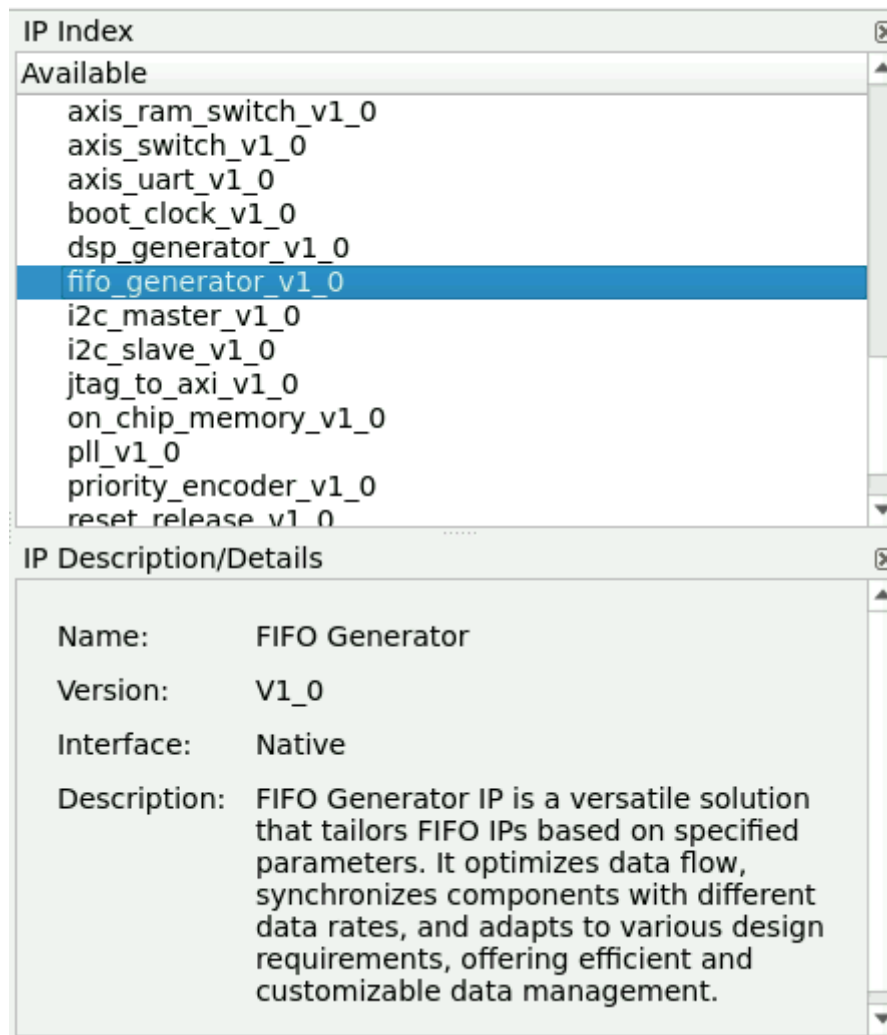
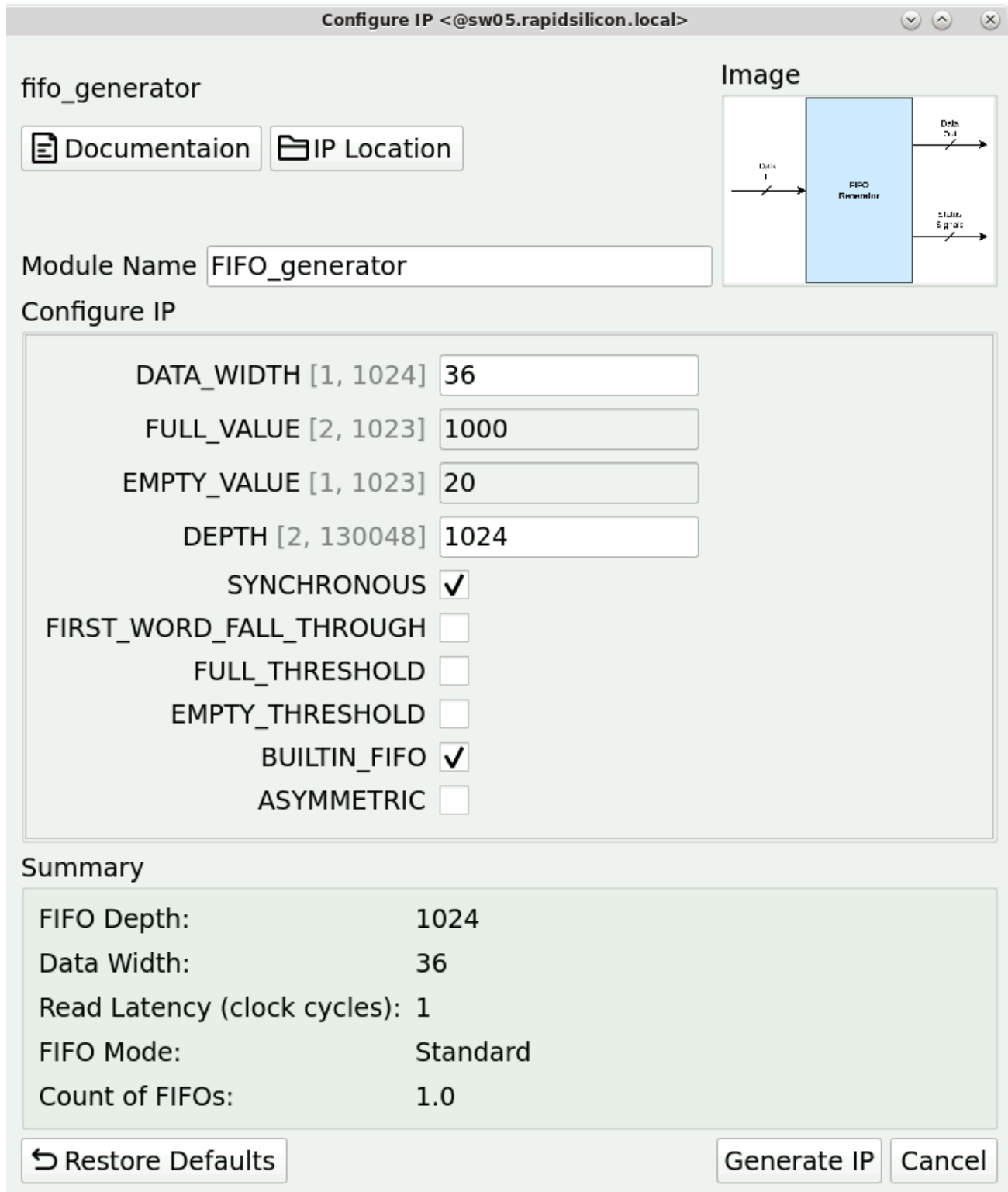


Figure 3: IP list

Parameters Customization

From the IP configuration window, the parameters of FIFO Generator can be configured and IP features can be enabled for generating a FIFO IP core that suits the user application requirement as shown in Figure 4. After IP Customization, the generated IP is made available to the user to be used in applications.



Configure IP <@sw05.rapid silicon.local>

fifo_generator

Documentaion IP Location

Module Name

Image

Block diagram showing Data In entering a FIFO Generator block, which outputs Data Out and Status Signals.

Configure IP

DATA_WIDTH [1, 1024]

FULL_VALUE [2, 1023]

EMPTY_VALUE [1, 1023]

DEPTH [2, 130048]

SYNCHRONOUS ☒

FIRST_WORD_FALL_THROUGH ☐

FULL_THRESHOLD ☐

EMPTY_THRESHOLD ☐

BUILTIN_FIFO ☒

ASYMMETRIC ☐

Summary

FIFO Depth: 1024

Data Width: 36

Read Latency (clock cycles): 1

FIFO Mode: Standard

Count of FIFOs: 1.0

Restore Defaults Generate IP Cancel

Figure 4: IP Configuration

Synthesis and PR

Raptor Suite is armed with tools for Synthesis and the generated post-synthesis net-lists can be viewed and analyzed from within the Raptor. The generated bit-stream can then be uploaded on an FPGA device to be utilized in hardware applications.

Test Bench

The FIFO IP, based on Verilog HDL, can be stimulated by any number of industry standard means. These may include simple Verilog test benches or stimulating the FIFO via an OS or via bare-metal firmware. The bundled test-bench for this IP is a Verilog based testbench that can be manipulated according to the configuration of the generated FIFO IP. After the generation of the IP, the source files and the simulation files are made available to the user along with the steps to simulate it via the bundled simulator by clicking on the "Simulate IP" button as shown in figure 5.

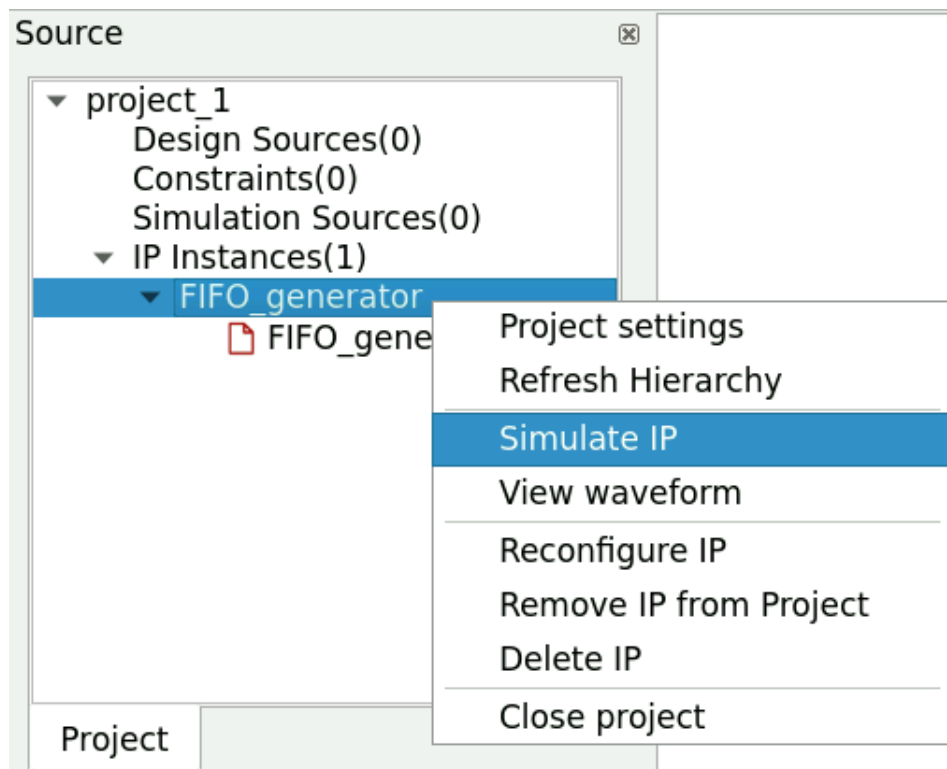


Figure 5: Simulate IP Window

To make sure that the generated FIFO IP works as intended, the testbench writes the same data in a memory array created inside the testbench. Then when the read happens from the FIFO, the output data is compared with the data that was written earlier in the memory array. The waveform can then be viewed in the integrated wave-viewer by clicking the "View waveform" as shown in the figure 6.

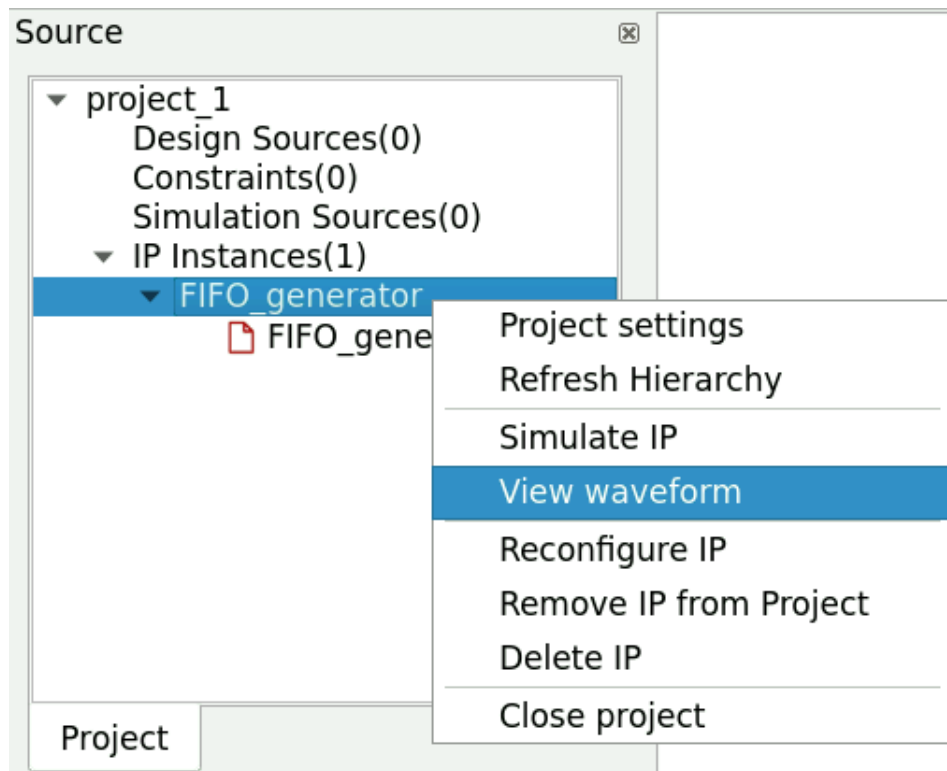


Figure 6: View Waveform Window

A sample waveform can be seen in the Figure 7.

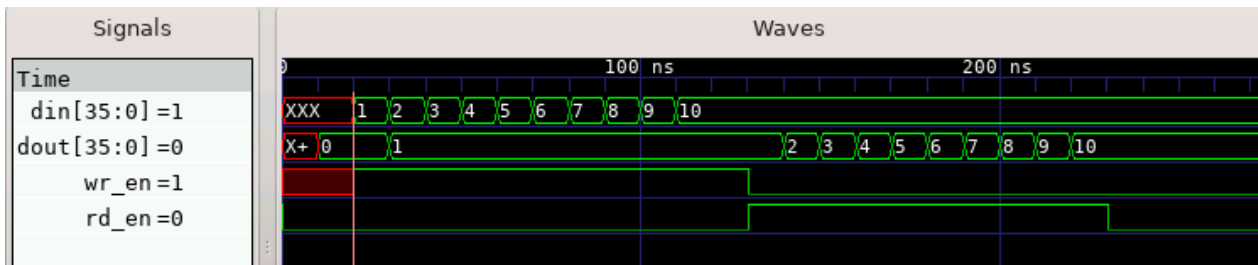
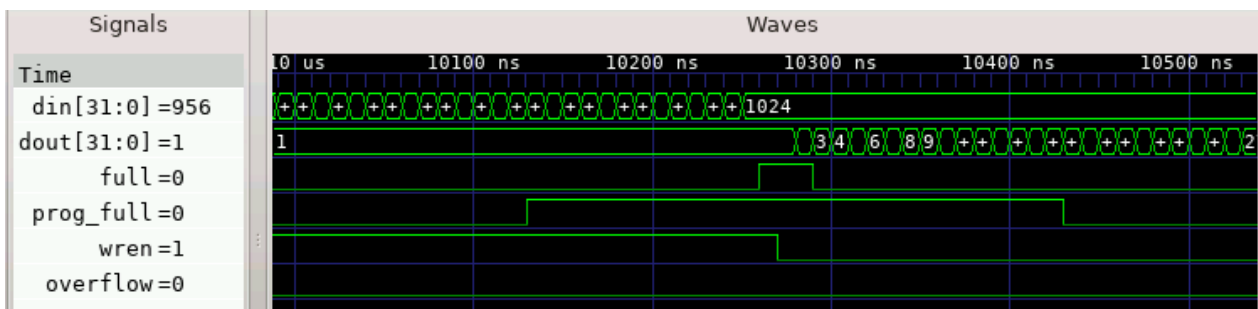


Figure 7: Test Results

The full and empty conditions can be seen in Figure 8.



The empty and programmable full conditions can be seen in Figure 9.

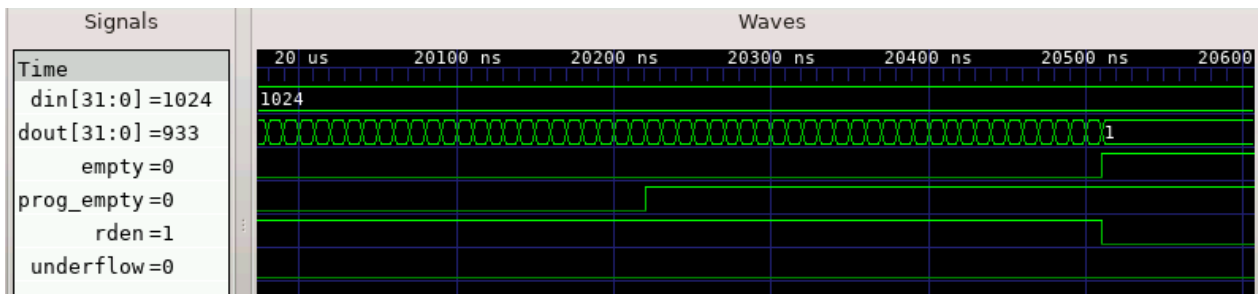


Figure 9: Empty conditions

The simulation results are then shown on the console window as shown in the figure 10.

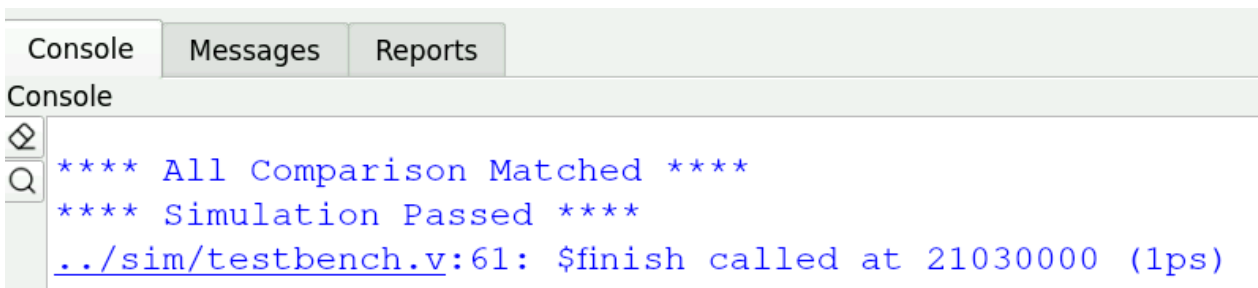


Figure 10: Simulation Results

Release

Release History

| Date | Version | Revisions |
|----------------------|---------|--|
| November 23, 2023 | 0.1 | Initial version FIFO Generator User Guide Document |