power_sink
Data Sheet

Content

Table of Contents

Figures

1 Introduction

This component implements chains of FFs, SRLs and BRAMs that toggle at a given pattern. This allows draining much power for testing purposes.

The number of elements to toggle ins configurable.

1.1 Purpose

The document here describes this very generic firmware developed for all kinds of projects.

1.2 Scope

This document provides a detailed overview of the firmware interface and specifies the user interface.

1.3 Definitions, acronyms, and abbreviations

This document is based on the “IEEE Recommended Practice for Software Requirements Specifications” [1].

FPGA	Field Programmable Gate Array. Programmable logic device.
FF	Flip-Flop
SRL	Shift register implemented in LUTs
BRAM	Block RAM

1.4 References

[1] IEEE Std 830-1998, Recommended Practice for Software Requirements Specifications.

2 IP Description

2.1 Vivado Component

The *power_sink* feature is loaded into a System On Chip (SOC) as a AXI slave component. Different parameters can be configured in a simple GUI.

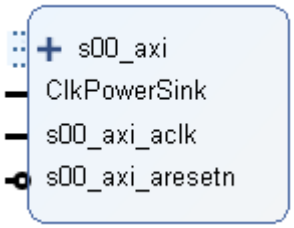

Figure 1: Component Overview

The parameters are pretty self-explaining

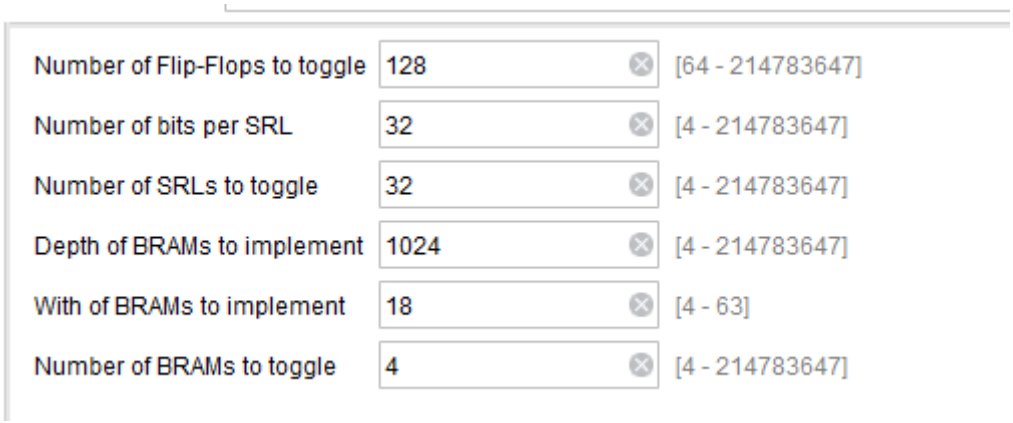

Figure 2: Component Configuration GUI

2.2 Architecture

2.2.1 Pattern Generation

The pattern is generated by a 32-bit shift register. The initial content of the shift register can be modified by the user to have some control over the toggle-rate.

A pattern of 0xAAAAAAAA or 0x55555555 leads to maximum toggle rate (all values toggle every clock cycle). The minimum toggle rate can be selected by using the pattern 0x0000FFFF or 0xFFFF0000 which leads to bits toggling only every 16th cycle.

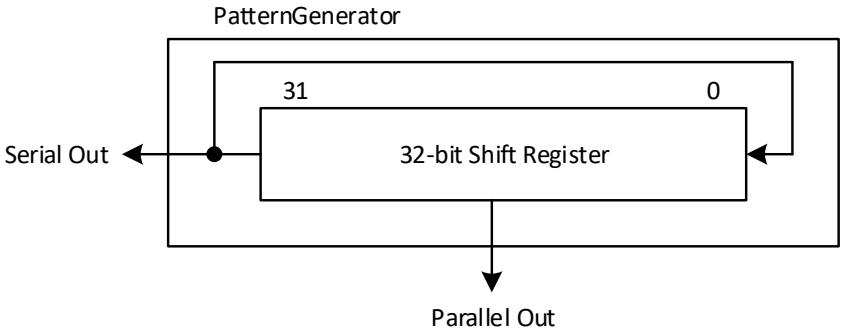

Figure 3: Pattern Generator

2.2.2 FF Implementation

For the FF toggling, a chain of FFs (with attributes that prevent optimization) is implemented. It is fed by the serial output of the pattern generator. Its output is fed to an AXI register only to prevent optimization (but reading this AXI register does not make any sense to the user).

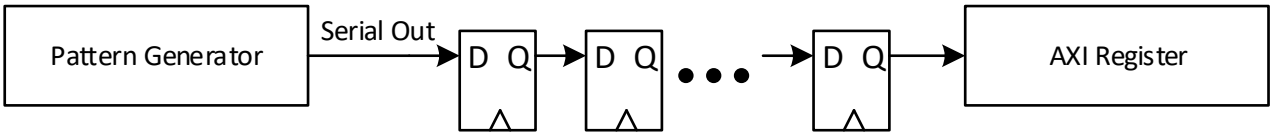

Figure 4: FF Implementation

2.2.3 SRL Implementation

The implementation for SRLs is the same as for FFs, just with SRLs and attributes to enforce SRL implementation.

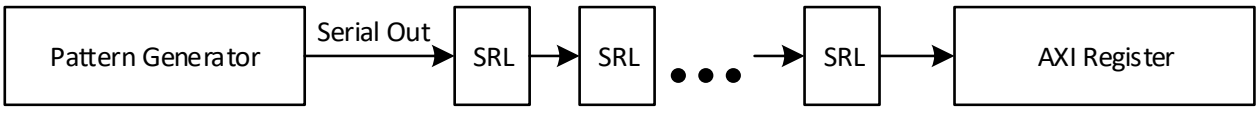

Figure 5: SRL Implementation

2.2.4 BRAM Implementation

For the BRAMs, the pattern generator is duplicated. The second pattern generator is initialized with the inverted version of the requested pattern. This results in the same number of toggling events.

An address counter counts through all addresses available and the pattern is written/read constantly. The address counter is distributed to all the BRAMs in the chain in pipelined fashion to prevent timing problems.

Both ports of the BRAMs are used. The addresses for both ports loop over the whole address range of the RAMs but the addresses are shifted by half the address range.

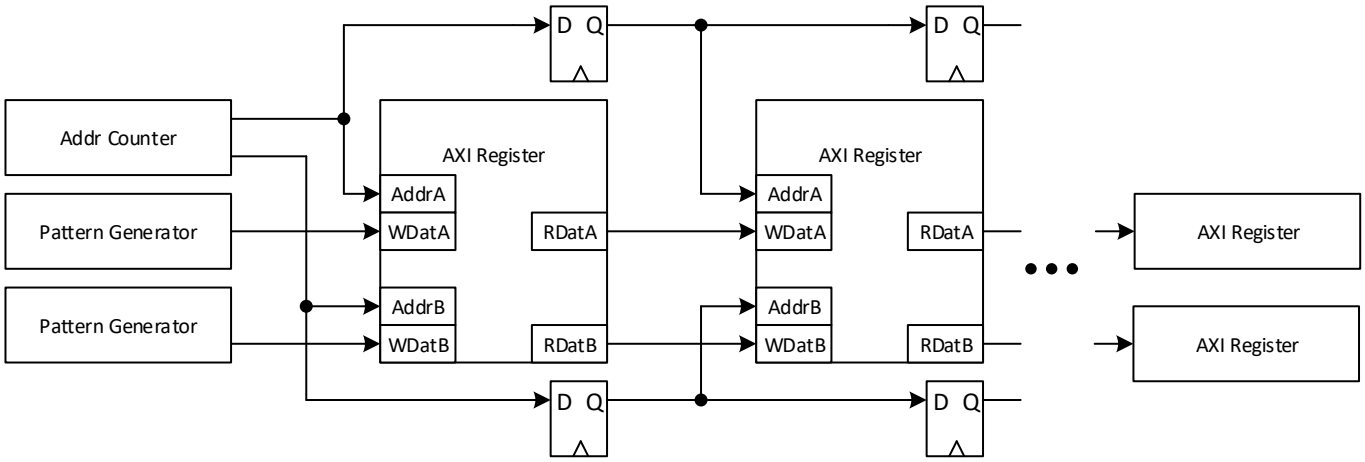

Figure 6: BRAM Implementation

3 Register Bank Description

3.1 Configuration

The following short notations for component parameters are used:

N SPI Transfer size in bits
M $\log_2(Depth_{FIFO})$

Address offset	R/W	Bit	Name	Description
0x00	R/W	0	EnaFf	Enable FF chain (reset value = 1) Enabled if EnaFf = 1 and EnaGlobal = 1
0x04	R/W	0	EnaSrl	Enable SRL chain (reset value = 1) Enabled if EnaSrl = 1 and EnaGlobal = 1
0x08	R/W	0	EnaBram	Enable BRAM chain (reset value = 1) Enabled if EnaBram = 1 and EnaGlobal = 1
0x0C	R/W	0	EnaGlobal	Enable/Disable all logic
0x10	RW	31:0	PatternFf	Pattern for the FF chain
0x14	RW	31:0	PatternSrl	Pattern for the SRL chain
0x18	RW	31:0	PatternBram	Pattern for the BRAM chain

Table 1: Registers

Normally direct access to the registers is not required because a low-level driver for the IP-Core exists.

4 Developer Information

4.1 Generate Fast Transients

Fast transients in power consumption are the most critical situation for the power supplies.

If the Core is enabled the first time, the power consumption will not increase immediately because all the FFs, SRLs and BRAMs are initialized with zeros. So first let the core run for a while to fill all cells with the pattern, then disable it and after a while re-enable it to generate transients of maximum steepness.

Use the global enable to enable all chains at the same time to generate one quick transient.

4.2 Packaging

To simplify re-packaging of the IP-Core after changes and avoid trouble with the Vivado GUI, a packaging script was written. To re-package the IP-Core, follow the steps below:

1. Open Vivado
2. In the TCL console, navigate to the “scripts” directory (using “cd <path>”)
3. Execute “source ./package.tcl”