# SIPO


April 16, 2025

Jay Convertino

# Contents

# 1 Usage

## 1.1 Introduction

This core converts the serial data to parallel data. This is useful for SPI and other serial data protocols. All data actions in the core are done on a rising edge clock. The enable is used to change the rate based upon the input clock.

## 1.2 Dependencies

The following are the dependencies of the cores.

- fusesoc 2.X
- iverilog (simulation)
- cocotb (simulation)

### 1.2.1 fusesoc_info Depenecies

- dep
  - AFRL:utility:helper:1.0.0

## 1.3 In a Project

This core is made to interface for serial to parallel data. On each enable pulse on a positive clock edge the input serial line will be sampled. The counter starts at 0 and the shift is complete once it hits BUS_WIDTH * 8. For a 32 bit word you will need 32 enable pulses. These will correspond to counter output 0 to 31. 32 will signal the shift is done and the parallel output data is valid. Once it is valid data can be read and load asserted. Load will reset the core and clear it for the next shift operation.

1. Set ena to 1 at rising clock edge for one clock cycle
2. Continue till the counter hits the number of bits needed.
3. Read the parallel data output to capture data.
4. Assert load to 1 to restart counter and clear parallel register.

# 2 Architecture

The only module is the sipo module. It is listed below.

- **sipo** Convert serial data to parallel by shifting in data on each enable pulse on a rising clock. (see core for documentation).

Please see 5 for more information.

# 3 Building

The SIPO core is written in Verilog 2001. They should synthesize in any modern FPGA software. The core comes as a fusesoc packaged core and can be included in any other core. Be sure to make sure you have meet the dependencies listed in the previous section.

## 3.1 fusesoc

Fusesoc is a system for building FPGA software without relying on the internal project management of the tool. Avoiding vendor lock in to Vivado or Quartus. These cores, when included in a project, can be easily integrated and targets created based upon the end developer needs. The core by itself is not a part of a system and should be integrated into a fusesoc based system. Simulations are setup to use fusesoc and are a part of its targets.

## 3.2 Source Files

### 3.2.1 fusesoc_info File List

- src
  - src/sipo.v
- tb
  - tb/tb_sipo.v
- tb_cocotb
  - 'tb/tb_cocotb.py': 'file_type': 'user', 'copyto': '.'
  - 'tb/tb_cocotb.v': 'file_type': 'verilogSource'

## 3.3 Targets

### 3.3.1 fusesoc_info Targets

- default

  Info: Default for IP intergration.

- sim

  Info: Base simulation using icarus as default.

- sim_cocotb

  Info: Cocotb unit tests

## 3.4 Directory Guide

Below highlights important folders from the root of the directory.

1. **docs** Contains all documentation related to this project.
   - **manual** Contains user manual and github page that are generated from the latex sources.
2. **src** Contains source files for the core
3. **tb** Contains test bench files for iverilog and cocotb
   - **cocotb** testbench files

# 4 Simulation

There are a few different simulations that can be run for this core.

## 4.1 iverilog

iverilog is used for simple test benches for quick verification, visually, of the core.

## 4.2 cocotb

To use the cocotb tests you must install the following python libraries.

```
$ pip install cocotb
```

- **sim_cocotb** Standard simulation of SIPO conversion.

Then you must use the cocotb sim target. The targets above can be run with various parameters.

```
$ fusesoc run --target sim_cocotb AFRL:simple:sipo
    ↪ :1.0.0
```

# 5 Module Documentation

- **sipo** SIPO converter
- **tb_sipo-v** Verilog test bench
- **tb_cocotb-py** Cocotb python test routines
- **tb_cocotb-v** Cocotb verilog test bench

The next sections document the module in detail.

# sipo.v

## AUTHORS

### JAY CONVERTINO

## DATES

### 2025/15/04

## INFORMATION

### Brief

SIPO (serial in parallel out) The idea is to keep this core simple, and let the control logic be handled outside of this core.

### License MIT



### sipo

```
module sipo #(
parameter
BUS_WIDTH
 =
1
) ( input clk, input rstn, input ena, input load, output [BUS_WIDTH*8-1:0]
```

serial in parallel out

#### Parametes

**BUS_WIDTH** width of the parallel data input in bytes.

### Ports

| | |
|---|---|
| **clk** | global clock for the core. |
| **rstn** | negative syncronus reset to clk. |
| **ena** | enable for core, use to change input rate. Enable serial shift input. |
| **load** | load parallel data from core, and reset counters for next incoming serial data. |
| **pdata** | parallel data output, valid when dcount is BUS_WIDTH*8. |
| **sdata** | serialized data input. |
| **dcount** | Number of bits to shift out. BUS_WIDTH*8 means all bits have been sampled and put into the register. |

# tb_sipo.v

## AUTHORS

### JAY CONVERTINO

## DATES

### 2025/04/16

## INFORMATION

### Brief

Test bench for Serial in Parallel out core.

### License MIT



### tb_sipo

```
module tb_sipo ()
```

Test bench for serial in parallel out

## INSTANTIATED MODULES

### sipo

```
sipo #(
```

```
                                                                        .
BUS_WIDTH(1)
) dut ( .clk(tb_clk), .rstn(tb_rstn), .ena(tb_ena), .load(tb_load), .pdata(
```

Device under test, serial to parallel data conversion.

# tb_cocotb.py

## AUTHORS

### JAY CONVERTINO

## DATES

### 2025/04/16

## INFORMATION

### Brief

Cocotb test bench

### License MIT



# piso

**piso**

Convert parallel data to serial

## VARIABLES

### wqueue

```
self.wqueue
```

Queue to store write requests

### _idle_write

```
self._idle_write
```

Event trigger for cocotb read

## FUNCTIONS

### _restart

```
def _restart(
self
)
```

kill and restart _run thread.

### random_bool

```
def random_bool()
```

Return a infinte cycle of random bools

Returns: List

### start_clock

```
def start_clock(
dut
)
```

Start the simulation clock generator.

#### Parameters

**dut** Device under test passed from cocotb test function

### reset_dut

```
async def reset_dut(
dut
)
```

Cocotb coroutine for resets, used with await to make sure system is reset.

### increment test

Coroutine that is identified as a test routine. Write data, on one clock edge, read on the next.

### Parameters

**dut** Device under test passed from cocotb.

## in_reset

```
@cocotb.test()
async def in_reset(
dut
)
```

Coroutine that is identified as a test routine. This routine tests if device stays in unready state when in reset.

### Parameters

**dut** Device under test passed from cocotb.

## no_clock

```
@cocotb.test()
async def no_clock(
dut
)
```

Coroutine that is identified as a test routine. This routine tests if no ready when clock is lost and device is left in reset.

### Parameters

**dut** Device under test passed from cocotb.

# tb_cocotb.v

## AUTHORS

### JAY CONVERTINO

## DATES

### 2025/04/16

## INFORMATION

### Brief

Test bench wrapper for cocotb

### License MIT



### tb_cocotb

```
module tb_cocotb #(
parameter
BUS_WIDTH
  =
4
) ( input clk, input rstn, input ena, input load, output [BUS_WIDTH*8-1:0]
```

SIPO interface DUT

#### Parameters

**BUS_WIDTH** Width of the APB3 bus data port in bytes.
parameter

### Ports

| | |
|---|---|
| **clk** | Clock |
| **rstn** | negative reset |
| **ena** | enable for core, use to change input rate. Enable serial shift input. |
| **load** | load parallel data from core, and reset counters for next incoming serial data. |
| **pdata** | parallel data output, valid when dcount is BUS_WIDTH*8. |
| **sdata** | serialized data input. |
| **dcount** | Number of bits to shift out. BUS_WIDTH*8 means all bits have been sampled and put into the register. |

## INSTANTIATED MODULES

### dut

```
sipo #(
                                                                                .
BUS_WIDTH(BUS_WIDTH)
) dut ( .clk(clk), .rstn(rstn), .ena(ena), .load(load), .pdata(pdata), .sda
```

Device under test, sipo