PISO

April 17, 2025

Jay Convertino

Contents

1 Usage

1.1 Introduction

This core converts parallel input data into serial output data. This is done on a positive clock edge in relation to a active high enable. PISO is useful for many parallel to serial operations such as UARTS, SPI, i2c, 1553, etc. This is a fusesoc 2.X compatable core and its VLNV is AFRL:simple:piso:X.X.X.

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 designed to be used on the same clock domain as the input parallel data. The serial output rate is set by the enable. The enable should only be pulsed for one clock cycle. If the clock is the rate the enable should be tied high. Load has to be used to load the input data to the register and reset the output count. The data count provides other cores a bit count to know what bit currently being output from the core. On load no data is output till the enable is pulsed. Once pulsed the output will be the data bit. Load will zero the output, also can be zeroed by just continuing to use enable past the full bit count (internal reg is filled with zeros). The series of steps to use the core are as follows.

1. Set pdata to input data, and set load to 1 (data will be loaded on positive clock edge to internal registers).
2. Set load to 0 (dcount will now be the max number of bits in the word loaded).
3. Pulse enable to 1 that is synced to the main clock. Only hold high for one period of the master clock.

4. Data bit will be output on the serial line and held till next enable or load.

5. Repeat enable pulse until dcount is equal to 0.

Adding the core to a fusesoc project, outside of adding the verilog module to your code, requires the core be added as dependency. Example:

```
dep:
  depend:
    - ">=AFRL:simple:piso:1.0.0"
```

Module instantion:

```
piso #(
  .BUS_WIDTH(4)
) inst_piso (
  .clk(clk),
  .rstn(rstn),
  .ena(ena),
  .load(load),
  .pdata(pdata),
  .sdata(sdata),
  .dcount(dcount)
);
```

2 Architecture

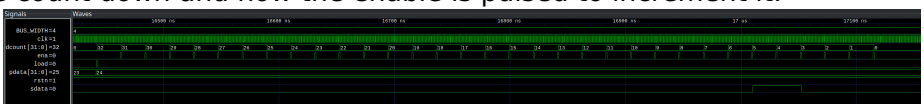
This core is made to interface parallel to serial data. Data is loaded into the core, and then enable is pulsed to push data out in a serial fashion. There is a count output that tells how many bits have been output. Data is only output when the enable is toggled. The counter will start at the max number of bits. For a 32 bit word this will be the decimal 32. When toggled the counter will state the number of the bit present at the output. Starting at 32 means once enable is active, and the counter is 31, bit 31 of the loaded word will now be output on the serial output. When the counter reaches 0 it means the internal register has output all data since the last bit, 0, is now present on the serial output. Any further enable pulses will not change the count but will shift the internal register. This will result in a 0 at the output since the register shifts in 0 bits as each bit is output. All data is shifted in MSb to LSb. Meaning Bit 31 is output first and 0 is last. The only module is the piso module. It is listed below.

- **piso** Convert parallel data to serial data. (see core for documentation).

Please see 5 for more information.

2.1 Waveform

Below is a waveform in a typical application of the core. This shows the count down and how the enable is pulsed to increment it.



3 Building

The PISO 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 as a dependency. Be 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 into 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/piso.v
- tb
  - tb/tb_piso.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. The backend used for testing is iverilog for verilog or cocotb simulations. Usually GTKWave is used to view the fst waveform output. Cocotb are the unit tests that attempt to give a pass/fail verification to the core operation.

4.1 iverilog

iverilog is used for simple test benches for quick visual verification of the core. This will autofinish after it has run up to a certain number of words have been output.

4.2 cocotb

This method allows for quick writing of test benches that acutally assert and check the state of the core. These tests are much more conclusive since it will run all test vectors and generate a report if they pass or fail. All tests output waves to a single fst file. The method of launching the tests is to use fusesoc. These have not been written to use a python runner method or makefiles. To use the cocotb tests you must install the following python libraries.

```
$ pip install cocotb
```

The targets available are listed below.

- **sim_cocotb** Standard simulation for PISO.

The targets above can be run with various parameters. This test will check the input/output against each other to validate core operation.

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

5 Module Documentation

- **piso** PISO converter
- **tb_piso-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.

piso.v

AUTHORS

JAY CONVERTINO

DATES

2025/04/15

INFORMATION

Brief

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

License MIT

piso

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

parllel in serial 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 output rate. Enable serial shift output.
load	load parallel data into core. Reset for next data message to send. This can be done at any time.
pdata	parallel data input, registered at load only.
sdata	serialized data output.
dcount	Number of bits to shift out. When the count hits zero, the parallel data register is empty and last bit is output on sdata.

tb_piso.v

AUTHORS

JAY CONVERTINO

DATES

2025/04/15

INFORMATION

Brief

Test bench for parallel to serial core.

License MIT

tb_piso

```
module tb_piso ()
```

Test bench for piso. Pump in data to parallel port on each count == 0.

INSTANTIATED MODULES

piso

```
piso #(
```

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

Device under test, parallel to serial

tb_cocotb.py

AUTHORS

JAY CONVERTINO

DATES

2025/03/04

INFORMATION

Brief

Cocotb test bench

License MIT

sipo

sipo

Convert serial data to parallel

VARIABLES

_idle_read

```
self._idle_read
```

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

2025/04/15

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, input [BUS_WIDTH*8-1:0] p
```

PISO Wrapper

Parameters

BUS_WIDTH Width of the parallel input in bytes
parameter

Ports

clk	Clock
rstn	negative reset
ena	enable for core, use to change output rate. Enable serial shift output.
load	load parallel data into core. This can be done at any time.
pdata	parallel data input, registered at load only.
sdata	serialized data output.
dcount	Number of bits to shift out (what is left, first bit is always available).

INSTANTIATED MODULES

dut

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

Device under test, piso