UP_APB3

March 26, 2025

Jay Convertino

Contents

1 Usage

1.1 Introduction

This core converts the APB3 bus to the uP bus. This allows any core with a uP bus to be interfaced with a APB3 bus. These busses are very similar and is done with combinatoral logic only.

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 APB3 bus to uP based device cores. This is part of a family of converters based on Analog Devices uP specification. Using this allows usage of Analog Devices AXI Lite core, AFRL APB3, AFRL Wishbone Classic, and AFRL Wishbone Pipeline converters. Meaning any uP core can be easily customized to any bus quickly. These are made for relativly slow speed bus device interfaces. An example of a Verilog uP interface provided below.

```
//output signals assigned to registers.
assign up_rack   = r_up_rack & up_rreq;
assign up_wack   = r_up_wack & up_wreq;
assign up_rdata = r_up_rdata;
assign irq       = r_irq;

assign s_rx_ren = ((up_raddr[3:0] == RX_FIFO_REG) &&
    ↪ up_rreq ? r_up_rack & r_rx_ren : 0);

//up registers decoder
always @(posedge clk)
begin
  if(rstn == 1'b0)
```

```
begin
  r_up_rack    <= 1'b0;
  r_up_wack    <= 1'b0;
  r_up_rdata   <= 0;

  r_rx_ren     <= 1'b0;

  r_overflow   <= 1'b0;

  r_control_reg <= 0;
end else begin
  r_up_rack    <= 1'b0;
  r_up_wack    <= 1'b0;
  r_tx_wen     <= 1'b0;
  r_rx_ren     <= 1'b0;
  r_up_rdata   <= r_up_rdata;
  //clear reset bits
  r_control_reg[RESET_RX_BIT] <= 1'b0;
  r_control_reg[RESET_TX_BIT] <= 1'b0;

  if(rx_full == 1'b1)
  begin
    r_overflow <= 1'b1;
  end

  //read request
  if(up_rreq == 1'b1)
  begin
    r_up_rack <= 1'b1;

    case(up_raddr[3:0])
      RX_FIFO_REG: begin
        r_up_rdata <= rx_rdata & {{(BUS_WIDTH*8-
            ↪ DATA_BITS){1'b0}}, {DATA_BITS{1'b1}}};
        r_rx_ren <= 1'b1;
      end
      STATUS_REG: begin
        r_up_rdata <= {{(BUS_WIDTH*8-8){1'b0}},
            ↪ s_parity_err, s_frame_err, r_overflow,
            ↪ r_irq_en, tx_full, tx_empty, rx_full,
            ↪ rx_valid};
        r_overflow <= 1'b0;
      end
      default:begin
        r_up_rdata <= 0;
      end
```

```
      endcase
    end

    //write request
    if(up_wreq == 1'b1)
    begin
      r_up_wack <= 1'b1;

      //only allow write once ack (Analog Devices does
          ↪ the same)
      if(r_up_wack == 1'b1) begin
        case(up_waddr[3:0])
          TX_FIFO_REG: begin
            r_tx_wdata  <= up_wdata;
            r_tx_wen    <= 1'b1;
          end
          CONTROL_REG: begin
            r_control_reg <= up_wdata;
          end
          default:begin
          end
        endcase
      end
    end
  end
end

//up control register processing and fifo reset
always @(posedge clk)
begin
  if(rstn == 1'b0)
  begin
    r_rstn_rx_delay <= ~0;
    r_rstn_tx_delay <= ~0;
    r_irq_en <= 1'b0;
  end else begin
    r_rstn_rx_delay <= {1'b1, r_rstn_rx_delay[
        ↪ FIFO_DEPTH-1:1]};
    r_rstn_tx_delay <= {1'b1, r_rstn_rx_delay[
        ↪ FIFO_DEPTH-1:1]};

    if(r_control_reg[RESET_RX_BIT])
    begin
      r_rstn_rx_delay <= {FIFO_DEPTH{1'b0}};
    end
```

```
      if(r_control_reg[RESET_TX_BIT])
      begin
        r_rstn_tx_delay <= {FIFO_DEPTH{1'b0}};
      end

      if(r_control_reg[ENABLE_INTR_BIT] != r_irq_en)
      begin
        r_irq_en <= r_control_reg[ENABLE_INTR_BIT];
      end
    end
  end
```

2 Architecture

The only module is the up_apb3 module. It is listed below.

- **up_apb3** Convert APB3 to the Analog Devices uP BUS. (see core for documentation).

2.1 cocotb

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

```
$ pip install cocotb
$ pip install cocotbext-axi
```

Each module has a cocotb based simulation. These use the cocotb extensions made by Alex. The two extensions used are cocotbext-axi and cocotbext-uart. These provide outside verification of the implimentation. These tests consist of 3 different fusesoc targets.

- **sim_cocotb_full** Standard simulation of TX/RX passing data to and from cocotbexts.
- **sim_cocotb_rx** Simulation of data receive using cocotbext.
- **sim_cocotb_tx** Simulation of data transmit using cocotbext.

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

```
$ fusesoc run --target AFRL:device_converter:axis_uart
    ↪ :1.0.0
```

This core only uses combinatoral methods to convert a few signals between the uP bus the APB3.

Please see 5 for more information.

3 Building

The APB3 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/up_apb3.v
- tb
  - tb/tb_apb3.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.
    - **specs** Contains specifications for the bus.
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
$ pip install cocotbext-up
$ pip install cocotbext-apb
```

Each module has a cocotb based simulation. These use the cocotb extensions APB and uP. To install these locally use the following.

```
$ pip install --break-system-packages -e .
```

- **sim_cocotb** Standard simulation APB3 to uP conversion using cocotbexts.

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

```
$ fusesoc run sim_cocotb --target AFRL:bus:up_apb3
    ↪ :1.0.0
```

5 Module Documentation

There is a single async module for this core.

- **up_apb3** APB3 to uP converter
- **tb_cocotb-py** Cocotb python test routines
- **tb_cocotb-v** Cocotb verilog test bench

The next sections document the module in great detail.

up_apb3.v

AUTHORS

JAY CONVERTINO

DATES

2024/03/19

INFORMATION

Brief

APB3 slave to uP interface

License MIT

up_apb3

```
module up_apb3 #(
parameter
ADDRESS_WIDTH
 =
16,
parameter
BUS_WIDTH
 =
4
) ( input clk, input rstn, input [ADDRESS_WIDTH-1:0] s_apb_paddr, input [0:(
```

APB3 slave to uP interface

Parameters

ADDRESS_WIDTH	parameter	Width of the APB3 address port in bits.
BUS_WIDTH	parameter	Width of the APB3 bus data port in bytes.

Ports

clk	Clock
rstn	negative reset
s_apb_paddr	APB3 address bus, up to 32 bits wide.
s_apb_psel	APB3 select per slave (1 for this core).
s_apb_penable	APB3 enable device for multiple transfers after first.
s_apb_pready	APB3 ready is a output from the slave to indicate its able to process the request.
s_apb_pwrite	APB3 Direction signal, active high is a write access. Active low is a read access.
s_apb_pwdata	APB3 write data port.
s_apb_prdata	APB3 read data port.
s_apb_pslverror	APB3 error indicates transfer failure, not implimented.
up_rreq	uP bus read request
up_rack	uP bus read ack
up_raddr	uP bus read address
up_rdata	uP bus read data
up_wreq	uP bus write request
up_wack	uP bus write ack
up_waddr	uP bus write address
up_wdata	uP bus write data

VARIABLES

valid

```
assign valid = s_apb_psel & s_apb_penable & rstn
```

This will add an extra clock cycle. since enable happens after select. both are needed to use the device.

s_apb_pslverror

```
assign s_apb_pslverror = 1'b0
```

APB3 error is always 0, no error.

up_waddr

```
assign up_waddr = s_apb_paddr[ADDRESS_WIDTH-1:shift]
```

up_waddr and s_apb_addr are a direct mapping.

up_waddr

up_raddr and s_apb_addr are a direct mapping.

up_wdata

```
assign up_wdata = s_apb_pwdata
```

up_wdata and s_apb_pwdata are a direct mapping.

s_apb_prdata

```
assign s_apb_prdata = up_rdata
```

s_apb_prdata and up_rdata are a direct mapping.

up_wreq

```
assign up_wreq = valid & s_apb_pwrite
```

uP write request is a combination of the APB3 valid and APB3 write select (active high is write).

up_rreq

```
assign up_rreq = valid & ~s_apb_pwrite
```

uP read request is a combination of the APB3 valid and APB3 write select (active low is read).

s_apb_pready

```
assign s_apb_pready = (
                                                                  up_wack |
up_rack |
                                                                         ~
valid
) & rstn
```

Diagrams seem to indicate that we should indicate ready when not sel and enable, which is why valid is complimented.

tb_cocotb.py

AUTHORS

JAY CONVERTINO

DATES

2025/03/04

INFORMATION

Brief

Cocotb test bench

License MIT

FUNCTIONS

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.

increment test stream

Coroutine that is identified as a test routine. Write data, in a stream to registers, then read back stream.

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_coctb.v

AUTHORS

JAY CONVERTINO

DATES

2025/03/26

INFORMATION

Brief

Test bench wrapper for cocotb

License MIT

tb_cocotb

```
module tb_cocotb #(
parameter
ADDRESS_WIDTH
 =
16,
parameter
BUS_WIDTH
 =
4
) ( input clk, input rstn, input [ADDRESS_WIDTH-1:0] s_apb_paddr, input [0:(
```

APB3 slave to uP interface DUT

Parameters

ADDRESS_WIDTH
parameter
Width of the APB3 address port in bits.

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

Ports

clk	Clock
rstn	negative reset
s_apb_paddr	APB3 address bus, up to 32 bits wide.
s_apb_psel	APB3 select per slave (1 for this core).
s_apb_penable	APB3 enable device for multiple transfers after first.
s_apb_pready	APB3 ready is a output from the slave to indicate its able to process the request.
s_apb_pwrite	APB3 Direction signal, active high is a write access. Active low is a read access.
s_apb_pwdata	APB3 write data port.
s_apb_prdata	APB3 read data port.
s_apb_pslverror	APB3 error indicates transfer failure, not implimented.
up_rreq	uP bus read request
up_rack	uP bus read ack
up_raddr	uP bus read address
up_rdata	uP bus read data
up_wreq	uP bus write request
up_wack	uP bus write ack
up_waddr	uP bus write address
up_wdata	uP bus write data

INSTANTIATED MODULES

dut

```
up_apb3 #(
                                                                        .
ADDRESS_WIDTH(ADDRESS_WIDTH),
                                                                        .
BUS_WIDTH(BUS_WIDTH)
) dut ( .clk(clk), .rstn(rstn), .s_apb_paddr(s_apb_paddr), .s_apb_psel(s_apl
```

Device under test, up_apb3