FPGA-based hysteresis current controller for three-phase inverter

TN120 | Posted on April 2, 2021 | Updated on May 7, 2025

Benoît STEINMANN
Software Team Leader
imperix • in

Julien ORSINGER
Power Applications Specialist
imperix • in

Table of Contents

This technical note provides an example of how a fast hysteresis current controller can be implemented, leveraging the possibility of editing the FPGA firmware for rapid control prototyping applications.

This example implements the direct current control of a three-phase passive load. It relies on manually-generated VHDL code. The automated generation of VHDL code is presented in TN121, which addresses the same application from Simulink and using Matlab HDL Coder.

Instructions on how to set up the FPGA development toolchain for imperix controllers are given in the dedicated note PN168. Quick-start recommendations regarding FPGA firmware customization can also be found in the getting started with FPGA control.

To find all FPGA-related notes, you can visit FPGA development homepage.

Software resources

hysteresis_current_control_simulinkDownload
hysteresis_current_control_FPGA_vivado_projectDownload

The first file is the Simulink model used to generate the CPU code, and the second file contains a Vivado project that implements the FPGA logic described below.

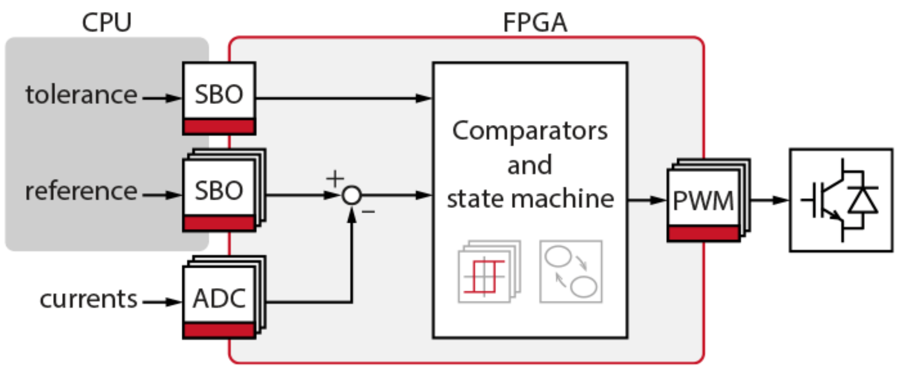
Implementation of the hysteresis current control

This implementation stems from the following choices:

- The current references are generated inside the CPU at a 40 kHz interval.
- The measured 3-phase currents are sampled at 400 kHz.

- The hysteresis control is implemented inside the programmable logic area (FPGA) by comparing the “fast” measured currents (400 kHz) with the “slow” generated references (40 kHz).
- The hysteresis comparators are 2-level comparators.
- The PWM output states are directly derived from the comparator state (see logic below).
- A counter prevents the PWM output from switching excessively rapidly between states (state change limiter). This ensures that the switching frequency of each phase is kept below a defined threshold.

Both CPU and FPGA tasks are illustrated below. The current references and the tolerance values are computed in the CPU and transferred to the FPGA with SBO blocks.



FPGA logic of the hysteresis controller

The following control logic is implemented in VHDL. Details on how to edit the firmware of the B-Box RCP are given in PN116.

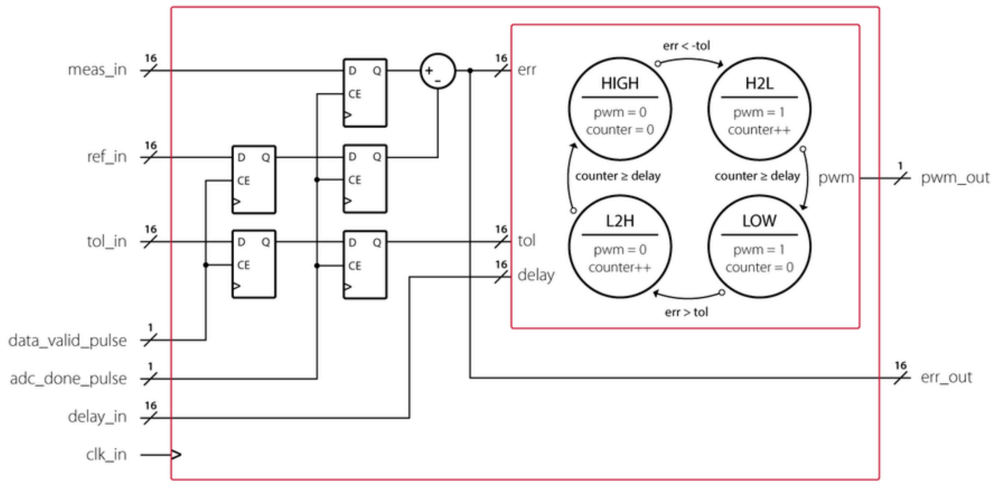
Source code of dcc.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity DirectCurrentControl is
    Port (
        -- Measure
        meas_in : in std_logic_vector(15 downto 0) := (others => '0');
        -- Reference
        ref_in : in std_logic_vector(15 downto 0) := (others => '0');
        -- Tolerence
        tol_in  : in std_logic_vector(15 downto 0) := (others => '0');
        -- Delay
        delay_in  : in std_logic_vector(15 downto 0) := (others => '0');
        -- Error = Measure - Reference
        err_out  : out std_logic_vector(15 downto 0);

        -- Timing pulses
        adc_done_pulse_in : in std_logic;
        data_valid_pulse_in : in std_logic;

        -- PWM output
        pwm_out : out std_logic;

        -- Main clock running at 250 MHz
        clk_in : in std_logic
    );
end DirectCurrentControl;
```

```vhdl
architecture impl of DirectCurrentControl is

    ATTRIBUTE X_INTERFACE_INFO : STRING;
    ATTRIBUTE X_INTERFACE_INFO of clk_in: SIGNAL is "xilinx.com:signal:clock:1.0 clk CLK";

    signal meas_ff1 : std_logic_vector(15 downto 0) := (others => '0');
    signal ref_ff1 : std_logic_vector(15 downto 0) := (others => '0');
    signal ref_ff2 : std_logic_vector(15 downto 0) := (others => '0');
    signal tol_ff1 : std_logic_vector(15 downto 0) := (others => '0');
    signal tol_ff2 : std_logic_vector(15 downto 0) := (others => '0');

    signal counter_reg : unsigned(15 downto 0) := (others => '0');

    signal err : signed(15 downto 0);

    type type_fsm_state is (STATE_HIGH, STATE_LOW, STATE_H2L, STATE_L2H);
    signal fsm_state : type_fsm_state := STATE_LOW;

begin

    INPUT : process(clk_in)
    begin
        if rising_edge(clk_in) then

            -- new SBO data are available
            if data_valid_pulse_in = '1' then
                ref_ff1 <= ref_in;
                tol_ff1 <= tol_in;
            end if;

            -- new measure is available
            if adc_done_pulse_in = '1' then
                meas_ff1 <= meas_in;
                ref_ff2 <= ref_ff1;
                tol_ff2 <= tol_ff1;
            end if;

        end if;
    end process INPUT;

    err <= signed(meas_ff1) - signed(ref_ff2);
    err_out <= std_logic_vector(err);

    HYSTERESIS : process(clk_in)
    begin
        if rising_edge(clk_in) then

            case fsm_state is

                when STATE_HIGH =>
                    pwm_out <= '0';
                    counter_reg <= (others => '0');
                    if err < -signed(tol_ff2) then
                        fsm_state <= STATE_H2L;
                    end if;

                when STATE_H2L =>
                    pwm_out <= '1';
                    counter_reg <= counter_reg + 1;
                    if counter_reg >= unsigned(delay_in) then
                        fsm_state <= STATE_LOW;
                    end if;

                when STATE_LOW =>
```

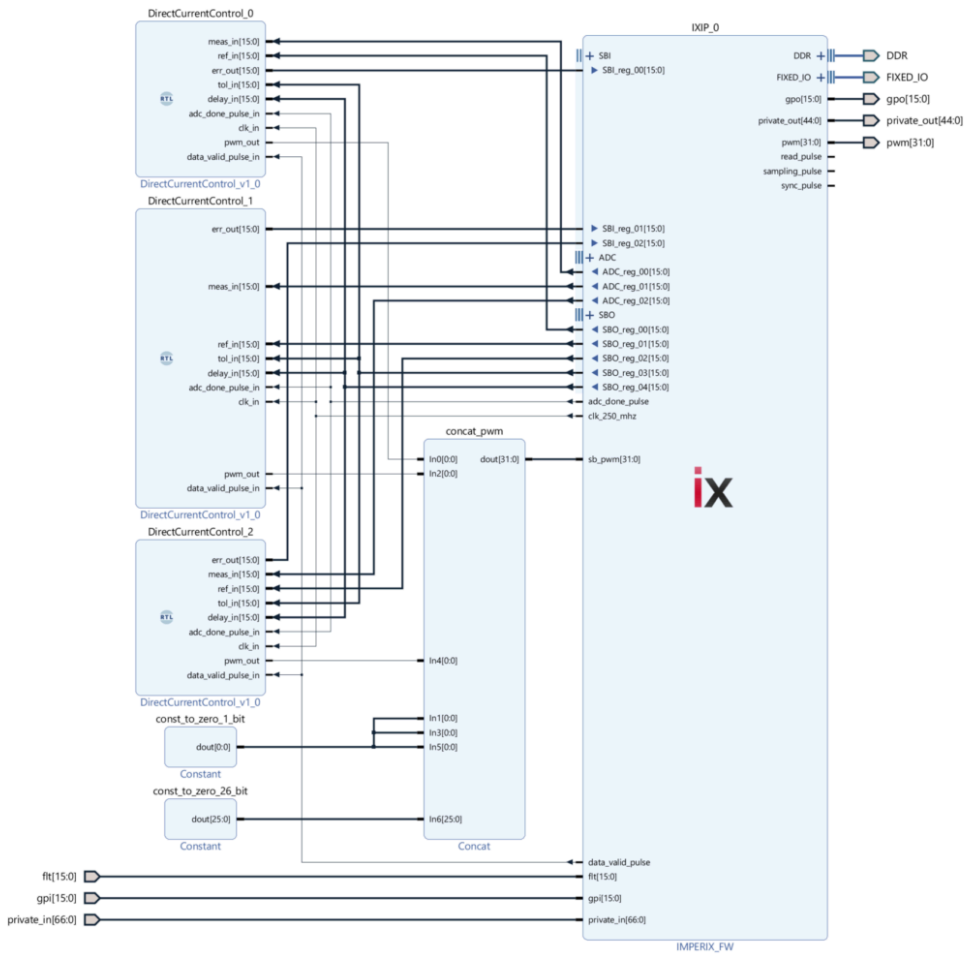
```vhdl
                    pwm_out <= '1';
                    counter_reg <= (others => '0');
                    if err > signed(tol_ff2) then
                        fsm_state <= STATE_L2H;
                    end if;

                when STATE_L2H =>
                    pwm_out <= '0';
                    counter_reg <= counter_reg + 1;
                    if counter_reg >= unsigned(delay_in) then
                        fsm_state <= STATE_HIGH;
                    end if;

                when others => null;
        end case;

      end if;
    end process HYSTERESIS;

end impl;
```
Code language: VHDL (vhdl)

The resulting logic is illustrated in the figure below. The current error is simply obtained by comparing the current reference with the actual measurement. Subsequently, as basic state machine controls the state transitions of the `pwm_out` signal, enforcing a minimum delay to limit the converter switching frequency.



This subsystem is instantiated three times, one for each phase. Connections are made as shown in the next figure:

- `meas_in` are connected to `ADC_reg_00`, `ADC_reg_01` and `ADC_reg_02`
- `ref_in` are connected to `SBO_reg_00`, `SBO_reg_01`and `SB0_reg_02`
- `tol_in` are connected to `SBO_reg_03`
- `delay_in` are connected to `SBO_reg_04`
- `err_out` are connected to a `SBI_reg_00`, `SBI_reg_01` and `SBI_02`
- `pwm_out` are connected to `sb_pwm[0]`, `sb_pwm[2]` and `sb_pwm[4]`

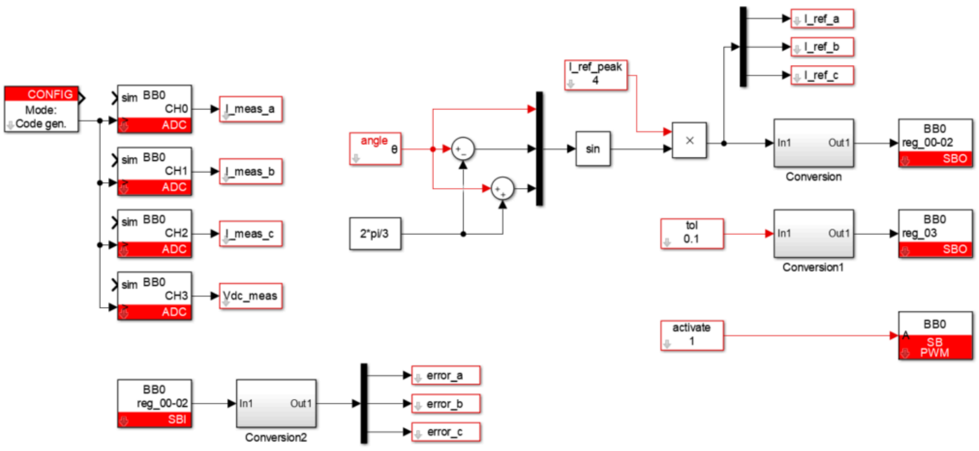


CPU implementation (using the imperix blockset for Simulink)

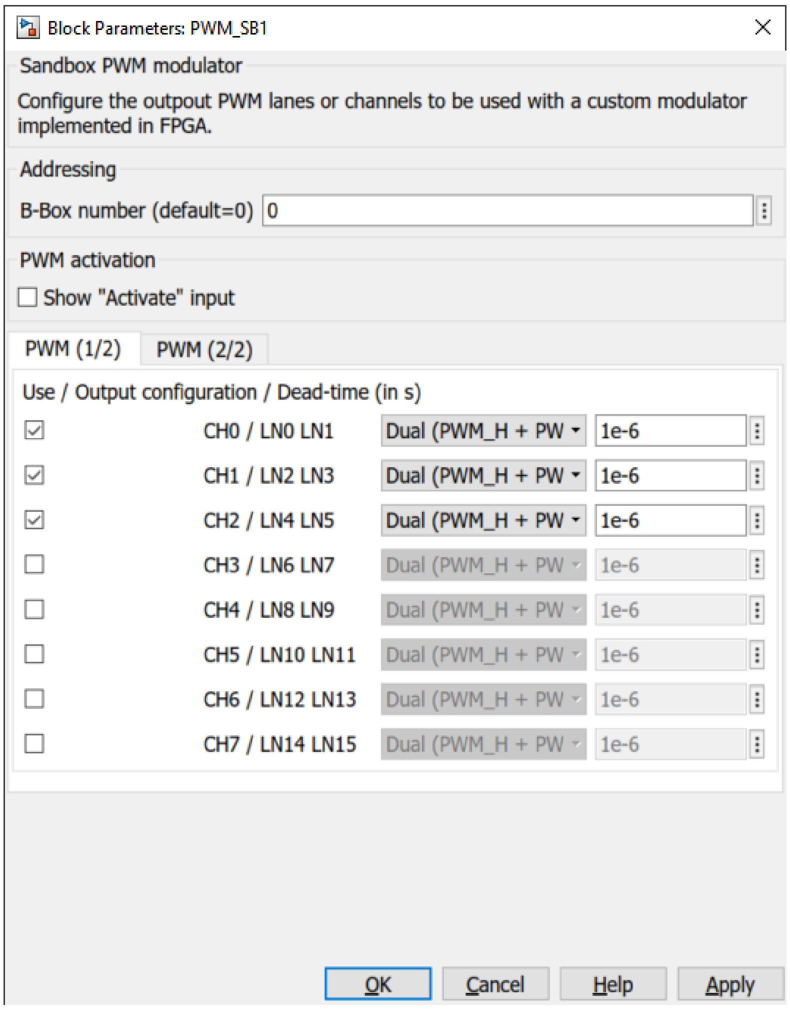
The following configuration is used:

- CLOCK_0 (main clock, controlling the sampling frequency): 400 kHz
- Interrupt postscaler: 5 (i.e. control interrupt executed at 40 kHz)
- SBO registers 00 to 02: current references (real-time registers)
- SBO register 03: hysteresis tolerance (real-time register)
- SBO register 04: state counter period (configuration register)
- SBI registers (read from FPGA) 00 to 02: current errors (for monitoring purpose only)
- SB_PWM: PWM channels 0 to 2 configured with dual outputs with a 1 us dead time
- State change limiter: maximum switching frequency: 40 kHz (i.e. H2L and L2H states have a counter period of 3125. Counter period is 4 ns)

The CPU code is generated automatically from Simulink, using the following model:



The PWM output configuration is the following. Further information on sandbox PWM configuration is available on the custom FPGA PWM page.


Block Parameters: PWM_SB1

Sandbox PWM modulator

Configure the outpout PWM lanes or channels to be used with a custom modulator implemented in FPGA.

Addressing

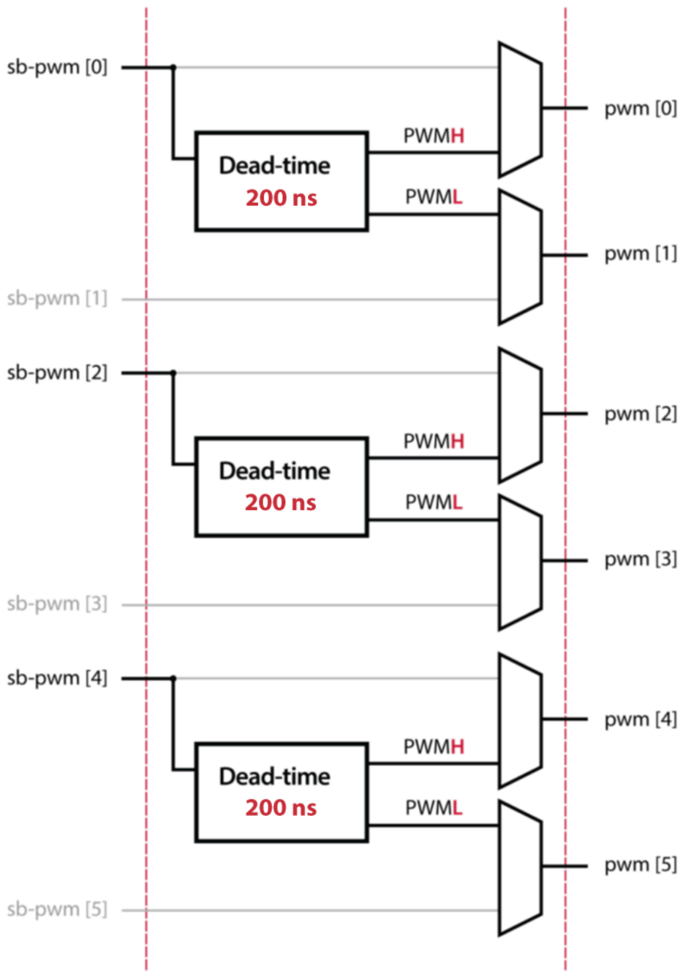
B-Box number (default=0) 0

PWM activation

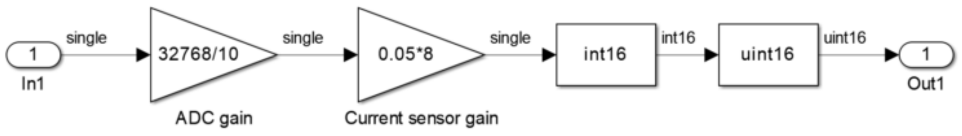
☐ Show "Activate" input

PWM (1/2) | PWM (2/2)

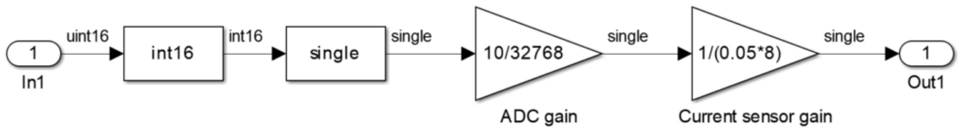
Use	Output configuration / Dead-time (in s)		
✓	CH0 / LN0 LN1	Dual (PWM_H + PW	1e-6
✓	CH1 / LN2 LN3	Dual (PWM_H + PW	1e-6
✓	CH2 / LN4 LN5	Dual (PWM_H + PW	1e-6
☐	CH3 / LN6 LN7	Dual (PWM_H + PW	1e-6
☐	CH4 / LN8 LN9	Dual (PWM_H + PW	1e-6
☐	CH5 / LN10 LN11	Dual (PWM_H + PW	1e-6
☐	CH6 / LN12 LN13	Dual (PWM_H + PW	1e-6
☐	CH7 / LN14 LN15	Dual (PWM_H + PW	1e-6



This model generates 3-phase current references with a user-defined amplitude and a frequency of 50Hz. These references are then converted into `unsigned int16` format to match the format of the FPGA registers. The conversion from Amperes to bits or inversely is done with the following operations. It considers the sensitivity of the current sensors and an input voltage range of ±10V with 16 bits ADCs. The intermediate conversion to `int16` ensures that the signed float number is coded or interpreted correctly in 2's complement.



Data type conversion for CPU to FPGA transfer



Data type conversion for FPGA to CPU transfer

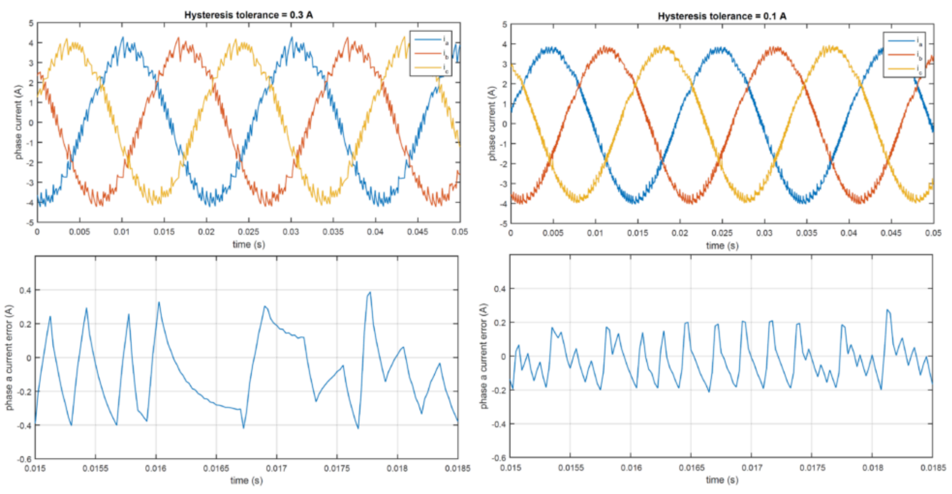
All probe variables are used for monitoring and debug purposes only.

Experimental results and validation of the hysteresis current control

The subsequent results consider the following parameters:

- V_{DC} = 70 V
- L_{load} = 5 mH
- R_{load} = 8 Ohm

The resulting current waveforms are shown below, for a current reference of 4 A peak and a hysteresis tolerance of ±0.3 A and ±0.1 A :



It can be clearly seen that reducing the hysteresis tolerance reduces the current ripples but increases the switching frequency. It also appears that the current error is not kept fully within the hysteresis band. This is unavoidable in digital implementations, for the following 2 reasons:

- A finite sampling frequency implies a maximum delay of Ts. In the present implementation, this corresponds to an additional current error of at most $1.5*V_{DC}*T_s/L_{load}$ = 50 mA
- The ADCs have a non-zero conversion delay. In the present implementation, the ADC have a 2 µs delay, which leads to an additional current error of at most $1.5*V_{DC}*T_{delay}/L_{load}$ = 42 mA

All in all, this explains a current error of approx. 100 mA beyond the hysteresis bounds.

Back to FPGA development homepage