NIM+ and Zedboard FPGA Commentary v1.0

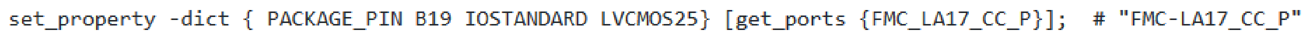
Mikhail Sharov, Constantinos Gerontis

May 22 2019

1 How to Use the Software Interface

Using the Vivado Suite allows the programming of the FPGA to perform tasks coded in VDHL, a hardware description language, and to define the inputs and outputs of the system.
Therefore, in order to use the interface, one should at minimum give a scan of the book: "VHDL for Programmable Logic" by Kevin Skahill, which should be made available to you via the e-Lab. This will give a basic overview of how to write in VHDL, although some concepts are better learned through experimentation or direct mentoring from a more experienced user. Using Vivado is at times complicated due to the complexity of the FPGA system. In any given project, when working with the ZedBoard and NIM+, there will be at least two main files: the .vhd executable code, and the zedboard master.xdc property declaration code.
The first file we will tackle is the zedboard master.xdc. This is a master file, which describes to the FPGA, which aspects of the Zedboard are being used, and at which voltages. A standard declaration of a pin to be used might look like:

Figure 1: Example Declaration of a Pin



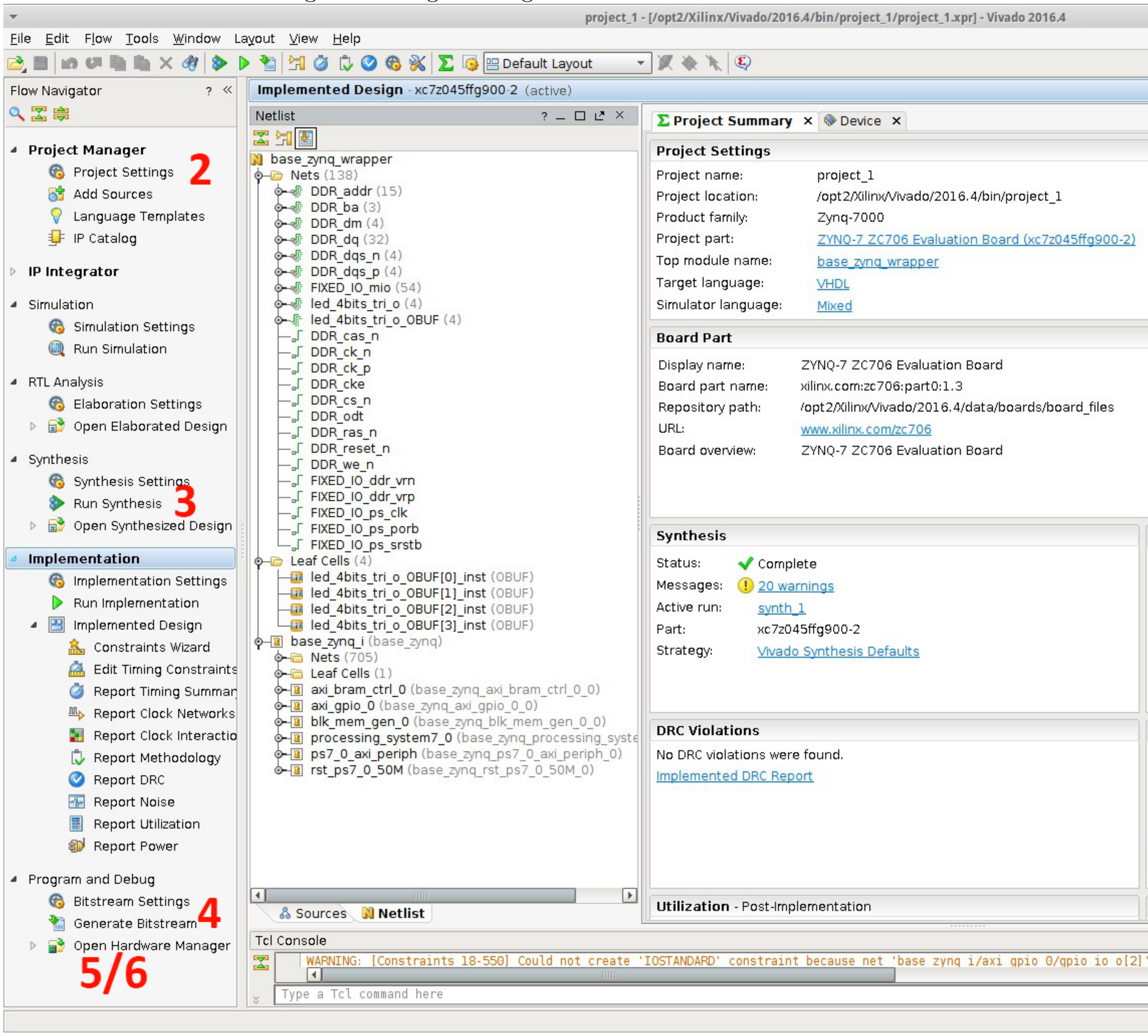
```
set_property -dict { PACKAGE_PIN B19 IOSTANDARD LVCMOS25} [get_ports {FMC_LA17_CC_P}];  # "FMC-LA17_CC_P"
```

This tells the FPGA that the pin B19 is a standard in/out with LVC MO at 2.5V. This may change depending on what kind of signal is being put through the pin, and at which allowable voltages it can run. For any pin that is to be used in the .vhd code, the pin must be declared and set to an appropriate allowable voltage in the zedboard master.xdc file. Which ports are used will depend on the path taken using the NIM+ and Zedboard schematics, where all pins and elements are labelled. Noting however, that variables and other non-physical entities do not need to be declared in this file.

The second file is the .vhd file, in this case, AND GATE.vhd. This file describes the processes and the connections that the FPGA makes in order to perform a task. This code is written in VHDL and generally begins with importing library IEEE, afterwards followed by the declaration of all ports (as described in the .xdc file) and what format they take. Third, there will be the main declaration of any variables not described as a

port. Finally, the code is written as a “behavioral” in which processes and assignments take place.
Once the .vhd has been written to produce a desired effect (such as setting a DAC) it can be implemented on the board.
1. The first step is to ensure that the board is connected to whichever computer you are using Vivado on.

Figure 2: Programming the Board



```
base_zynq_wrapper
  Nets (138)
    DDR_addr (15)
    DDR_ba (3)
    DDR_dm (4)
    DDR_dq (32)
    DDR_dqs_n (4)
    DDR_dqs_p (4)
    FIXED_IO_mio (54)
    led_4bits_tri_o (4)
    led_4bits_tri_o_OBUF (4)
    DDR_cas_n
    DDR_ck_n
    DDR_ck_p
    DDR_cke
    DDR_cs_n
    DDR_odt
    DDR_ras_n
    DDR_reset_n
    DDR_we_n
    FIXED_IO_ddr_vrn
    FIXED_IO_ddr_vrp
    FIXED_IO_ps_clk
    FIXED_IO_ps_porb
    FIXED_IO_ps_srstb
  Leaf Cells (4)
    led_4bits_tri_o_OBUF[0]_inst (OBUF)
    led_4bits_tri_o_OBUF[1]_inst (OBUF)
    led_4bits_tri_o_OBUF[2]_inst (OBUF)
    led_4bits_tri_o_OBUF[3]_inst (OBUF)
  base_zynq_i (base_zynq)
    Nets (705)
    Leaf Cells (1)
    axi_bram_ctrl_0 (base_zynq_axi_bram_ctrl_0_0)
    axi_gpio_0 (base_zynq_axi_gpio_0_0)
    blk_mem_gen_0 (base_zynq_blk_mem_gen_0_0)
    processing_system7_0 (base_zynq_processing_syste
    ps7_0_axi_periph (base_zynq_ps7_0_axi_periph_0)
    rst_ps7_0_50M (base_zynq_rst_ps7_0_50M_0)
```

Project Settings

Project name:	project_1
Project location:	/opt2/Xilinx/Vivado/2016.4/bin/project_1
Product family:	Zynq-7000
Project part:	ZYNQ-7 ZC706 Evaluation Board (xc7z045ffg900-2)
Top module name:	base_zynq_wrapper
Target language:	VHDL
Simulator language:	Mixed

Board Part

Display name:	ZYNQ-7 ZC706 Evaluation Board
Board part name:	xilinx.com:zc706:part0:1.3
Repository path:	/opt2/Xilinx/Vivado/2016.4/data/boards/board_files
URL:	www.xilinx.com/zc706
Board overview:	ZYNQ-7 ZC706 Evaluation Board

Synthesis

Status:	✓ Complete
Messages:	20 warnings
Active run:	synth_1
Part:	xc7z045ffg900-2
Strategy:	Vivado Synthesis Defaults

DRC Violations

No DRC violations were found.
Implemented DRC Report

Utilization - Post-Implementation

```
WARNING: [Constraints 18-550] Could not create 'IOSTANDARD' constraint because net 'base_zynq_i/axi_gpio_0/gpio_io_o[2]'
```

2. The second step is to open whichever project you would like to program onto the FPGA
3. The third step is to run a synthesis of the code, this will give any warning or sytax errors that may be in your code and will terminate if any are found.
4. The fourth step is to generate a bitstream, this will also terminate if any issues are found.
5. The fifth step is to open the hardware manager and select the device on which you would like to upload. In the case of the BU e-Lab setup it is the first option.
6. Finally, select program device. Ensure that the selected bitstream is the one most recently generated and run.
Congratulations! The FPGA has been programmed.

Vivado gives a detailed account of any syntax or logical issues, so fixing them is a matter to looking at the messages the programs gives back and implementing the changes.

2 Commentary on Main Code

The code in AND GATE.vhd is used to set a threshold for an event trigger for the NIM+ daughter card. This is done by communicating to the DAC(digital to analog converter) on the daughter card using the FPGA and setting the trigger level. The DAC is the AD5324. To communicate with the DAC, a 16 bit word must be sent, and the values of the bits are specified in the diagram below (with more info on page 15 of the datasheet):

Figure 3: NIM+ DAC Serial Input

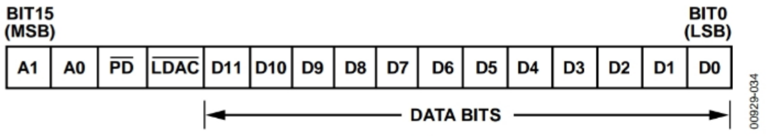
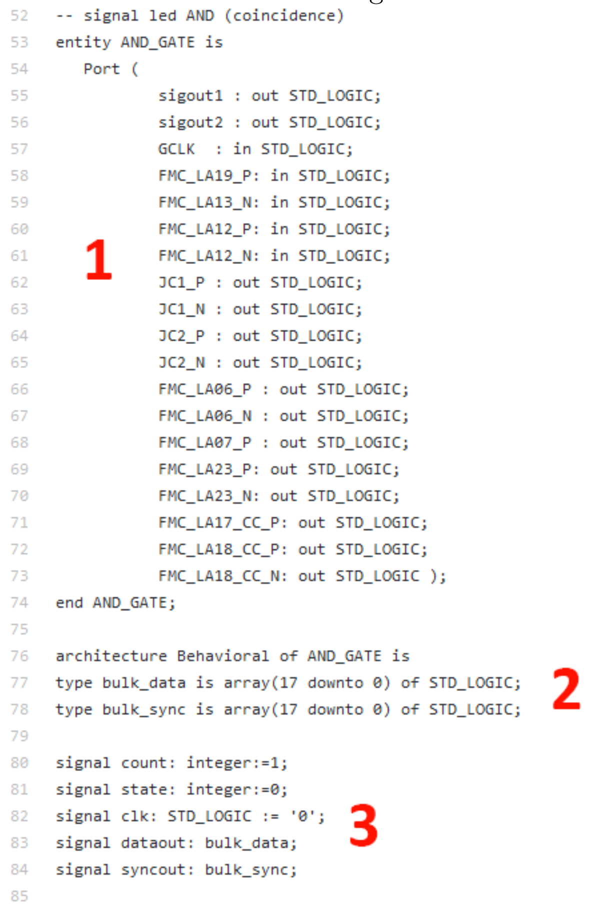


Figure 34. AD5324 Input Shift Register Contents

To effectively communicate to the DAC, the FPGA must output 3 signals: data, clock, and SYNC. The clock is used to set the communication speed and the SYNC line is used to enable communication. Since the native clock on the ZYNC-7000 board is too fast for the DAC (100 MHz) a clock divider is used to generate a slower clock signal.
The main code is split into two parts, the first of which is the declarations:

Figure 4: Main Code Section 1

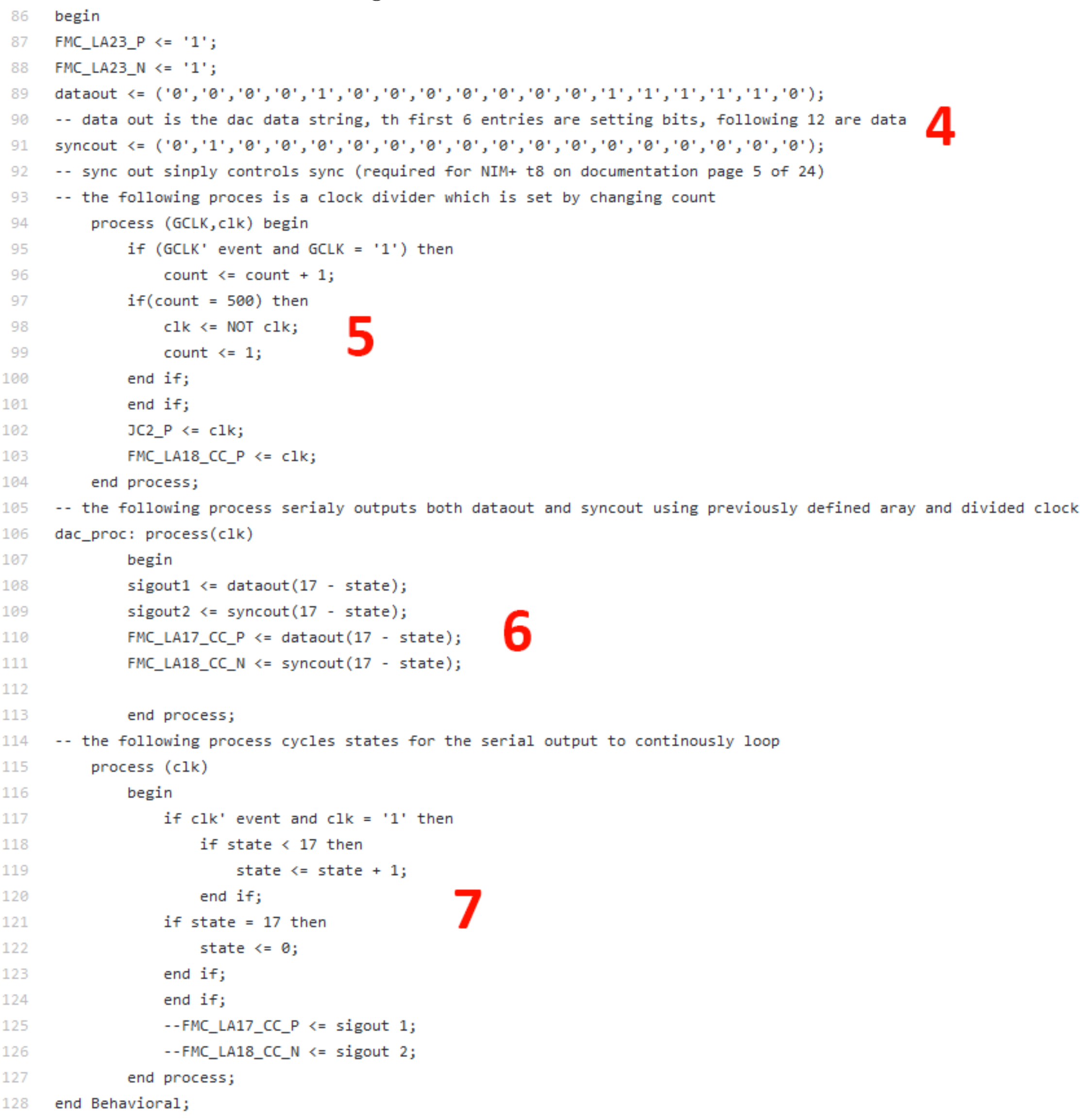


```
52  -- signal led AND (coincidence)
53  entity AND_GATE is
54     Port (
55             sigout1 : out STD_LOGIC;
56             sigout2 : out STD_LOGIC;
57             GCLK   : in STD_LOGIC;
58             FMC_LA19_P: in STD_LOGIC;
59             FMC_LA13_N: in STD_LOGIC;
60             FMC_LA12_P: in STD_LOGIC;
61             FMC_LA12_N: in STD_LOGIC;
62             JC1_P : out STD_LOGIC;
63             JC1_N : out STD_LOGIC;
64             JC2_P : out STD_LOGIC;
65             JC2_N : out STD_LOGIC;
66             FMC_LA06_P : out STD_LOGIC;
67             FMC_LA06_N : out STD_LOGIC;
68             FMC_LA07_P : out STD_LOGIC;
69             FMC_LA23_P: out STD_LOGIC;
70             FMC_LA23_N: out STD_LOGIC;
71             FMC_LA17_CC_P: out STD_LOGIC;
72             FMC_LA18_CC_P: out STD_LOGIC;
73             FMC_LA18_CC_N: out STD_LOGIC );
74  end AND_GATE;
75
76  architecture Behavioral of AND_GATE is
77  type bulk_data is array(17 downto 0) of STD_LOGIC;
78  type bulk_sync is array(17 downto 0) of STD_LOGIC;
79
80  signal count: integer:=1;
81  signal state: integer:=0;
82  signal clk: STD_LOGIC := '0';
83  signal dataout: bulk_data;
84  signal syncout: bulk_sync;
85
```

This section of the code, (1) beginning on line 52 declares various ports, described in the zedboard master.xdc, mainly as standard logic (either voltage high or voltage low). After the ports are declared, two 17 length arrays (2) are declared: the bulk data, and bulk sync, both of which are required for setting the DAC. Finally some variables that will be used later are also declared (3), including a counting variable, as state variable, a divided clock, and the outputs to the DAC.

The next section, which includes assignments and processes, tells the FPGA which pins to set to which values.

Figure 5: Main Code Section 2



```
begin
FMC_LA23_P <= '1';
FMC_LA23_N <= '1';
dataout <= ('0','0','0','0','1','0','0','0','0','0','0','0','1','1','1','1','1','0');
-- data out is the dac data string, th first 6 entries are setting bits, following 12 are data
syncout <= ('0','1','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0');
-- sync out sinply controls sync (required for NIM+ t8 on documentation page 5 of 24)
-- the following proces is a clock divider which is set by changing count
    process (GCLK,clk) begin
        if (GCLK' event and GCLK = '1') then
            count <= count + 1;
        if(count = 500) then
            clk <= NOT clk;
            count <= 1;
        end if;
        end if;
        JC2_P <= clk;
        FMC_LA18_CC_P <= clk;
    end process;
-- the following process serialy outputs both dataout and syncout using previously defined aray and divided clock
dac_proc: process(clk)
        begin
        sigout1 <= dataout(17 - state);
        sigout2 <= syncout(17 - state);
        FMC_LA17_CC_P <= dataout(17 - state);
        FMC_LA18_CC_N <= syncout(17 - state);

        end process;
-- the following process cycles states for the serial output to continously loop
    process (clk)
        begin
            if clk' event and clk = '1' then
                if state < 17 then
                    state <= state + 1;
                end if;
            if state = 17 then
                state <= 0;
            end if;
            end if;
            --FMC_LA17_CC_P <= sigout 1;
            --FMC_LA18_CC_N <= sigout 2;
        end process;
end Behavioral;
```

This section begins the behavioral part of the code, and intializes pins FMCLA23P and N to logic 1 , then the dataout and syncout are defined (4). See page 15 of the DAC datasheet for more information on why they are defined the way they are. The dataout can be set to any value following the template of page 15, in order to set a varying discriminator level. This is based on the fraction of the maximum binary number possible, and that will be a fraction of the reference value, which in this case in 3.3V. Next, the clock divider is set by changing the count every 500 ticks (5). This

sets up a slower clock which is usable by the DAC.
This value can be changed to either make a faster or slower clock and is up to the operators discretion as to which speed is needed. DAC PROC, the next process, (6) outputs the dataout and syncout serially to the pins which go into the NIM+ DAC as well as pmods to see them on the scope.
The last process (7) simply cycles states every clock tick to move the DAC dataout and syncout serially and to repeat once the sequence has ended.
This concludes the main code that has been written up to May 22 2019.

3 Outstanding Reminders

Some outstanding reminders which are useful to keep in mind:
1. ALWAYS save and backup code before major changes are committed as this will save much headache.
2. The code must be reprogrammed after the FPGA is turned off and on, everytime.
3. After any change to the code, the whole process of synthesizing and generating a bitstream must be repeated on the the code in order to apply it to the FPGA.