Tristan Niewenhuis Due: 12/11/2024 Ceng 242: Dr. Pyeatt

Lab 7a: Pulse Modulator

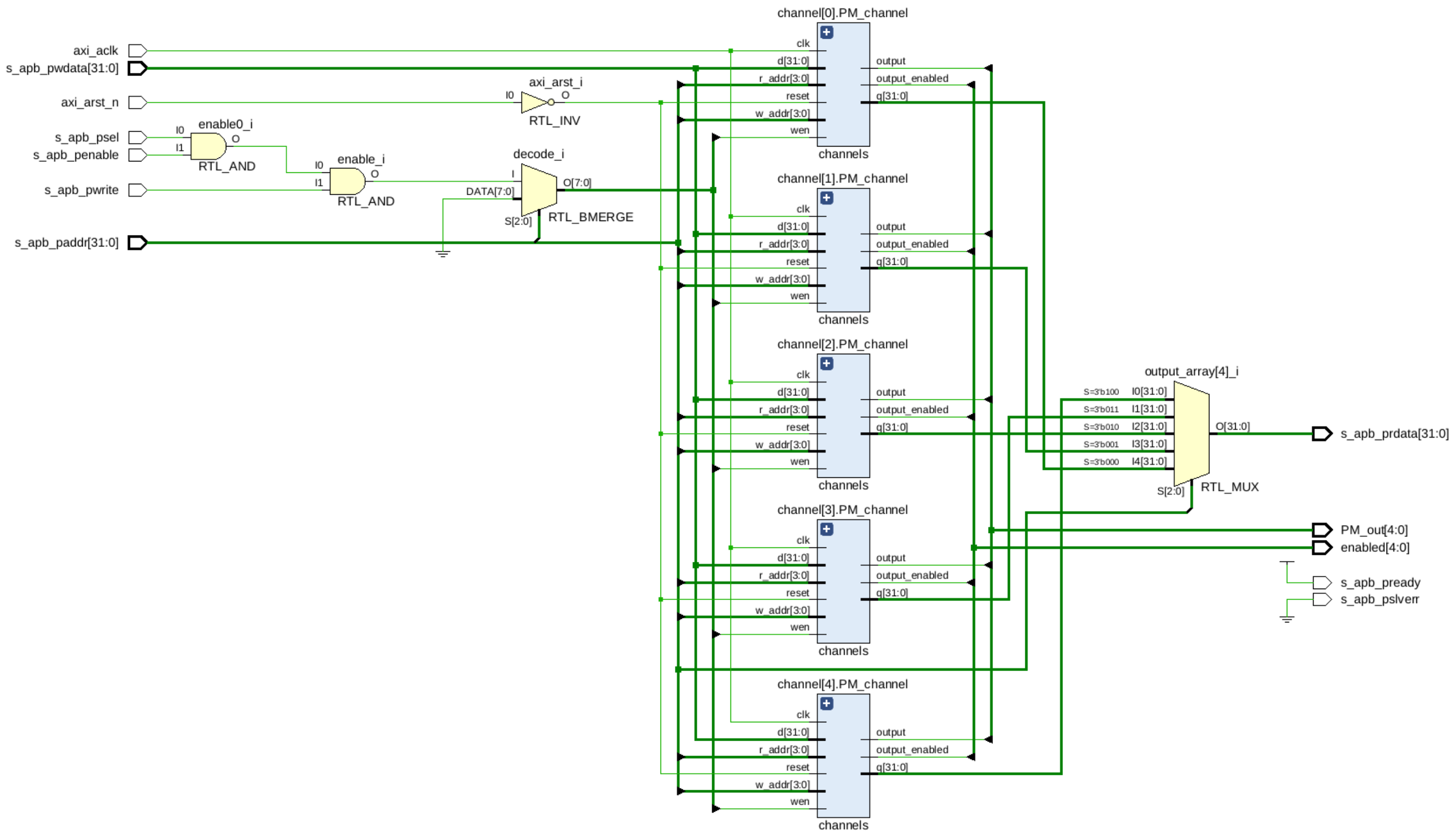
-Overview-

In this multi-week lab, we will design and implement an APB pulse modulation (PM) device to integrate with a microprocessor system. Starting with a basic pulse-width modulator (PWM), we will progressively enhance the design by adding features like a FIFO memory with interrupts and Pulse Density Modulation (PDM). Using VHDL in Vivado, we will create a top-level entity that supports multiple channels, implement a decoder and multiplexer for APB communication, and test the device through simulation and on an FPGA board with provided code. This lab will help us improve our VHDL skills, build moderately complex hardware, and interface custom modules with a CPU. Our final submission will include a PDF report, VHDL code, and a hardware demonstration.

-Design-

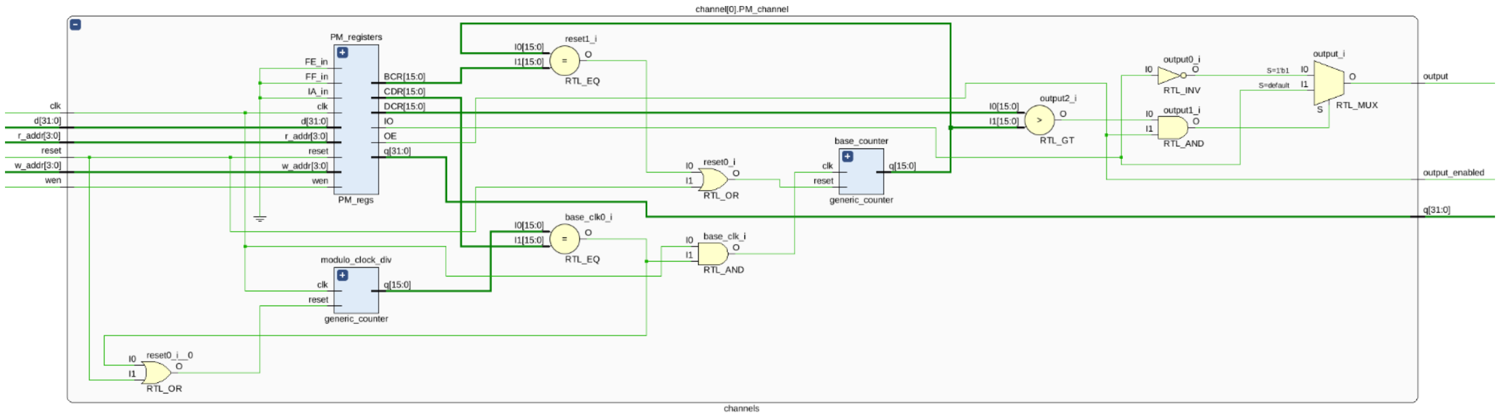
Highest Level

This level just organizes all the signals in and out of each channel and onto the apb bus. The reset is un-inverted. The select, enable, and write bits are ANDed together and decoded to select the appropriate channel to write to. To read, all the signals are routed through a mux controlled by the appropriate apb address bites.



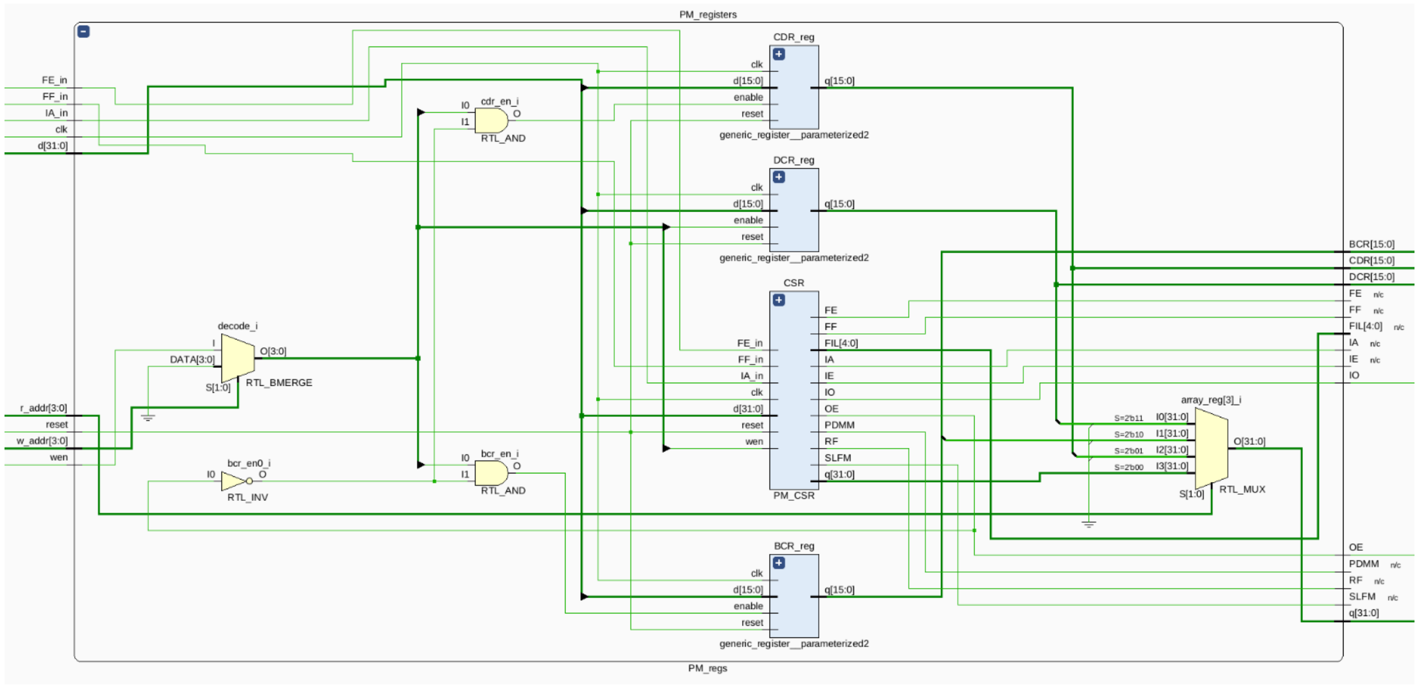
Channel Level

At this level, you can see the clock divider that divides the clock based on the value of the CDR. Then that clock goes into the base counter, that counts to a value before it resets at the value of the BCR. The base counter value is compared to the DCR, and when the DCR is greater than the base counter value, the output will be high. Right before it exits the channel, it goes through an AND gate and mux that controls whether the output is enabled or inverted based on the values of IO and OE.



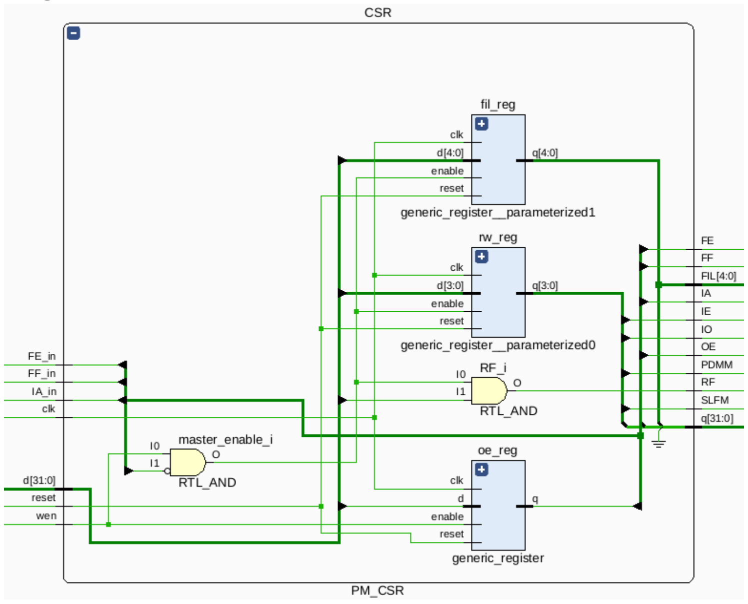
PM Registers Level

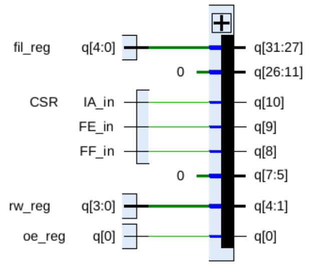
These registers are organized a lot like a register file. The output is selected by the address with a mux routing each register output. The 2 AND gates are used to only write to the CDR and BCR if OE is low, and they are selected to write. The decoder is to select the correct register.



CSR Register

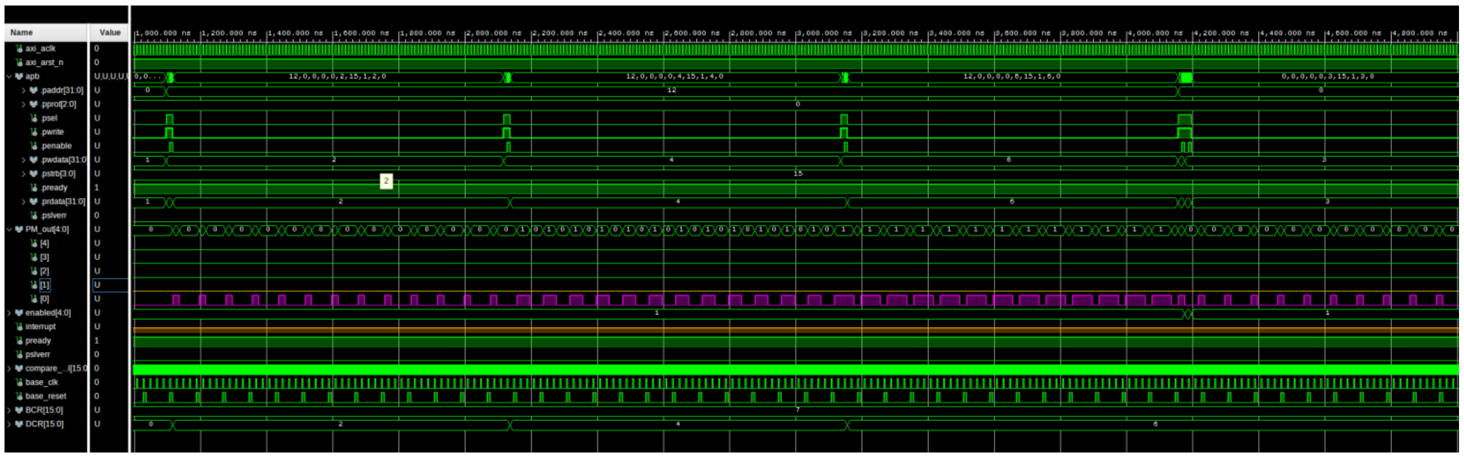
In the CSR, there are 3 main registers that are used to store the read/write values, the FIL value, and the OE. The other register can only be written to if the OE is low, which is the purpose of the AND gate master enable. The reset FIFO is write only, so no storage is needed and only needs to be high if the CSR is enabled and the bit is written high. On the right, you can see that the output is organized so the top 5 bits are the FIL register, then some 0 padding, some FIFO information that is just passed through, some more 0 padding including the reset FIFO bit, and the reset of the read/write bits including OE.



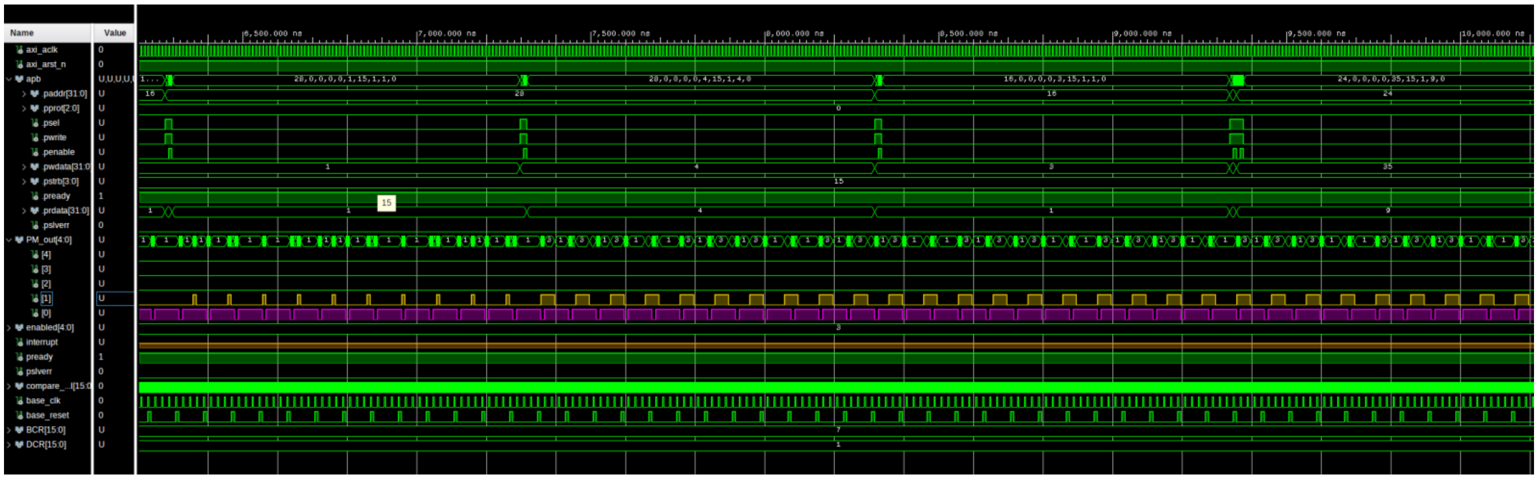


-Simulation –

In this first simulation, you can see that the Output signal (purple) is properly adjusting to the values being written to in the DCR. Furthermore, the base clock at the bottom is being properly divided. The enable is bit is also high when on as it should be. Toward the end, you can see the device is disabled for a clock cycle and the enable goes low. After the output is disabled, the signal is inverted, and you can see the output is inverted to what it was before.

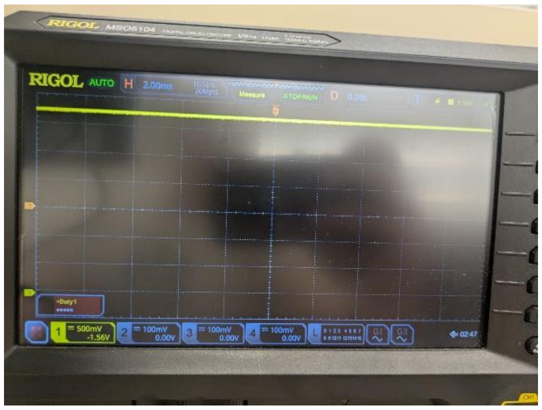


In this simulation, you can see that another channel (yellow) can be operated completely independent of the others. During that last part of the simulation, I tried to write to the CDR and BCR, and the values did not change as the device was enabled.

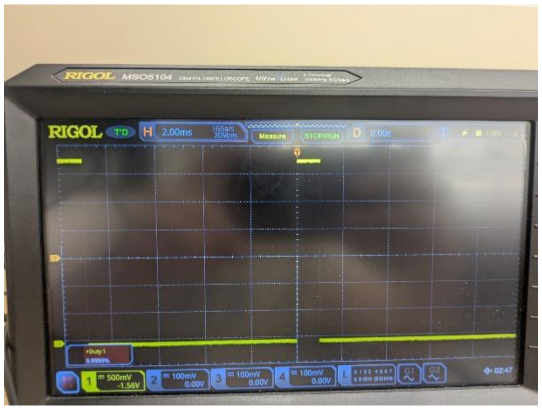


-Implemented Design Verification-

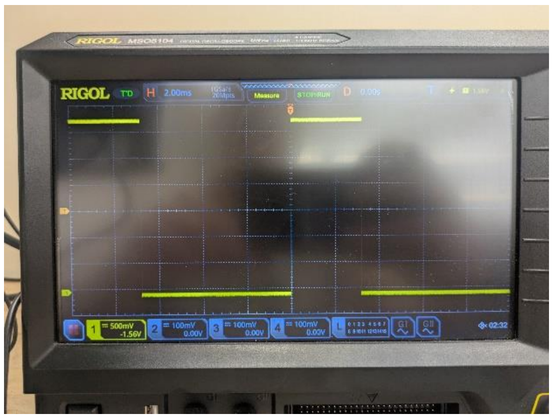
Device at 100%



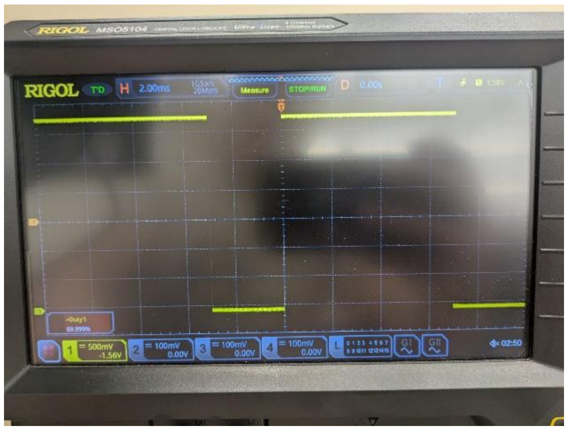
Device at 10%



Device at 30%



Device at 70%



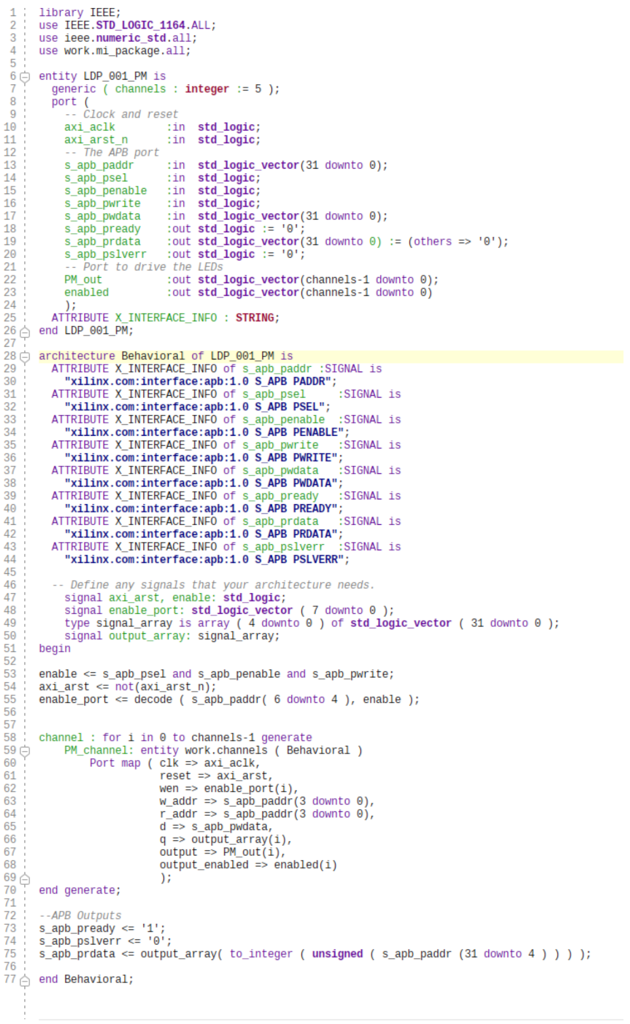
-Conclusion-

Overall, the lab took some time but went pretty well. It took a bit to wrap my head around the organization of all the registers and such, as I am not used to getting a guide like the one provided and working off that. The programming was definitely the most finicky part as I could not get the terminal to write to the FPGA. In the end, I just hard coded the values to the registers, and it worked as expected.

-Appendix-

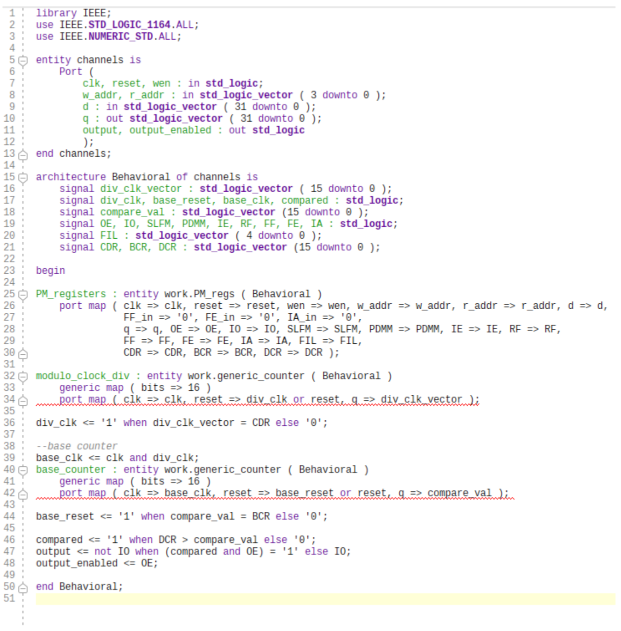
Main VHDL

LDP_001_PM



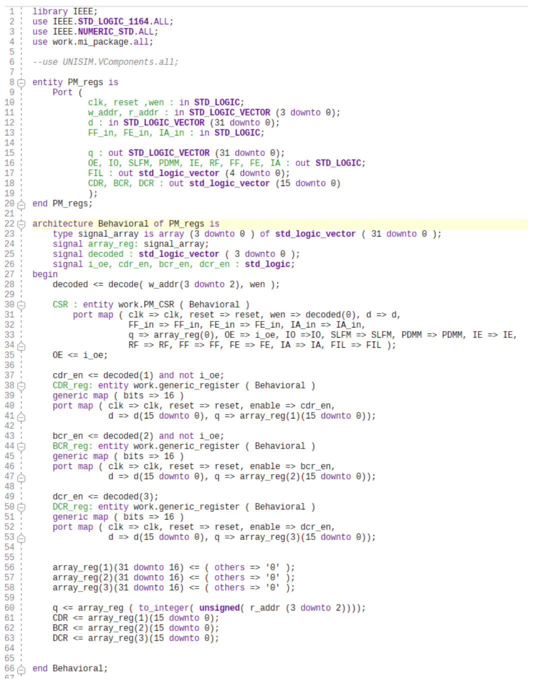
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use work.mi_package.all;

entity LDP_001_PM is
  generic ( channels : integer := 5 );
  port (
    -- Clock and reset
    axi_aclk        :in  std_logic;
    axi_arst_n      :in  std_logic;
    -- The APB port
    s_apb_paddr     :in  std_logic_vector(31 downto 0);
    s_apb_psel      :in  std_logic;
    s_apb_penable   :in  std_logic;
    s_apb_pwrite    :in  std_logic;
    s_apb_pwdata    :in  std_logic_vector(31 downto 0);
    s_apb_pready    :out std_logic := '0';
    s_apb_prdata    :out std_logic_vector(31 downto 0) := (others => '0');
    s_apb_pslverr   :out std_logic := '0';
    -- Port to drive the LEDs
    PM_out          :out std_logic_vector(channels-1 downto 0);
    enabled         :out std_logic_vector(channels-1 downto 0)
    );
  ATTRIBUTE X_INTERFACE_INFO : STRING;
end LDP_001_PM;

architecture Behavioral of LDP_001_PM is
  ATTRIBUTE X_INTERFACE_INFO of s_apb_paddr :SIGNAL is
    "xilinx.com:interface:apb:1.0 S_APB PADDR";
  ATTRIBUTE X_INTERFACE_INFO of s_apb_psel     :SIGNAL is
    "xilinx.com:interface:apb:1.0 S_APB PSEL";
  ATTRIBUTE X_INTERFACE_INFO of s_apb_penable  :SIGNAL is
    "xilinx.com:interface:apb:1.0 S_APB PENABLE";
  ATTRIBUTE X_INTERFACE_INFO of s_apb_pwrite   :SIGNAL is
    "xilinx.com:interface:apb:1.0 S_APB PWRITE";
  ATTRIBUTE X_INTERFACE_INFO of s_apb_pwdata   :SIGNAL is
    "xilinx.com:interface:apb:1.0 S_APB PWDATA";
  ATTRIBUTE X_INTERFACE_INFO of s_apb_pready   :SIGNAL is
    "xilinx.com:interface:apb:1.0 S_APB PREADY";
  ATTRIBUTE X_INTERFACE_INFO of s_apb_prdata   :SIGNAL is
    "xilinx.com:interface:apb:1.0 S_APB PRDATA";
  ATTRIBUTE X_INTERFACE_INFO of s_apb_pslverr  :SIGNAL is
    "xilinx.com:interface:apb:1.0 S_APB PSLVERR";

  -- Define any signals that your architecture needs.
    signal axi_arst, enable: std_logic;
    signal enable_port: std_logic_vector ( 7 downto 0 );
    type signal_array is array ( 4 downto 0 ) of std_logic_vector ( 31 downto 0 );
    signal output_array: signal_array;
begin

enable <= s_apb_psel and s_apb_penable and s_apb_pwrite;
axi_arst <= not(axi_arst_n);
enable_port <= decode ( s_apb_paddr( 6 downto 4 ), enable );


channel : for i in 0 to channels-1 generate
    PM_channel: entity work.channels ( Behavioral )
        Port map ( clk => axi_aclk,
                   reset => axi_arst,
                   wen => enable_port(i),
                   w_addr => s_apb_paddr(3 downto 0),
                   r_addr => s_apb_paddr(3 downto 0),
                   d => s_apb_pwdata,
                   q => output_array(i),
                   output => PM_out(i),
                   output_enabled => enabled(i)
                   );
end generate;

--APB Outputs
s_apb_pready <= '1';
s_apb_pslverr <= '0';
s_apb_prdata <= output_array( to_integer ( unsigned ( s_apb_paddr (31 downto 4 ) ) ) );

end Behavioral;
```

Channel (although there are error lines is the code, it was more of a warning and the code elaborated, synthesized, and implemented fine)



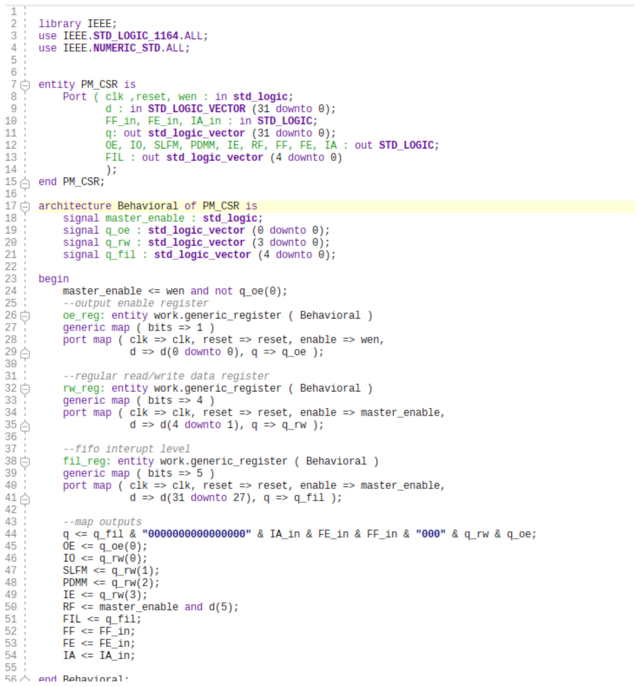
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity channels is
    Port (
        clk, reset, wen : in std_logic;
        w_addr, r_addr : in std_logic_vector ( 3 downto 0 );
        d : in std_logic_vector ( 31 downto 0 );
        q : out std_logic_vector ( 31 downto 0 );
        output, output_enabled : out std_logic
        );
end channels;

architecture Behavioral of channels is
    signal div_clk_vector : std_logic_vector ( 15 downto 0 );
    signal div_clk, base_reset, base_clk, compared : std_logic;
    signal compare_val : std_logic_vector (15 downto 0 );
    signal OE, IO, SLFM, PDMM, IE, RF, FF, FE, IA : std_logic;
    signal FIL : std_logic_vector ( 4 downto 0 );
    signal CDR, BCR, DCR : std_logic_vector (15 downto 0 );

begin

PM_registers : entity work.PM_regs ( Behavioral )
    port map ( clk => clk, reset => reset, wen => wen, w_addr => w_addr, r_addr => r_addr, d => d,
               FF_in => '0', FE_in => '0', IA_in => '0',
               q => q, OE => OE, IO => IO, SLFM => SLFM, PDMM => PDMM, IE => IE, RF => RF,
               FF => FF, FE => FE, IA => IA, FIL => FIL,
               CDR => CDR, BCR => BCR, DCR => DCR );

modulo_clock_div : entity work.generic_counter ( Behavioral )
    generic map ( bits => 16 )
    port map ( clk => clk, reset => div_clk or reset, q => div_clk_vector );

div_clk <= '1' when div_clk_vector = CDR else '0';

--base counter
base_clk <= clk and div_clk;
base_counter : entity work.generic_counter ( Behavioral )
    generic map ( bits => 16 )
    port map ( clk => base_clk, reset => base_reset or reset, q => compare_val );

base_reset <= '1' when compare_val = BCR else '0';

compared <= '1' when DCR > compare_val else '0';
output <= not IO when (compared and OE) = '1' else IO;
output_enabled <= OE;

end Behavioral;
```

PM_Registers



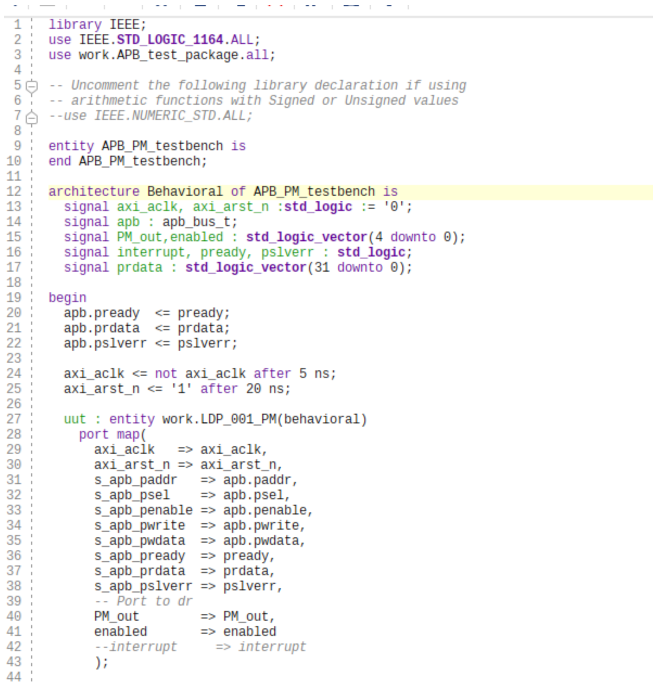
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.mi_package.all;

--use UNISIM.VComponents.all;

entity PM_regs is
    Port (
            clk, reset ,wen : in STD_LOGIC;
            w_addr, r_addr : in STD_LOGIC_VECTOR (3 downto 0);
            d : in STD_LOGIC_VECTOR (31 downto 0);
            FF_in, FE_in, IA_in : in STD_LOGIC;

            q : out STD_LOGIC_VECTOR (31 downto 0);
            OE, IO, SLFM, PDMM, IE, RF, FF, FE, IA : out STD_LOGIC;
            FIL : out std_logic_vector (4 downto 0);
            CDR, BCR, DCR : out std_logic_vector (15 downto 0)
            );
end PM_regs;

architecture Behavioral of PM_regs is
    type signal_array is array (3 downto 0 ) of std_logic_vector ( 31 downto 0 );
    signal array_reg: signal_array;
    signal decoded : std_logic_vector ( 3 downto 0 );
    signal i_oe, cdr_en, bcr_en, dcr_en : std_logic;
begin
    decoded <= decode( w_addr(3 downto 2), wen );

    CSR : entity work.PM_CSR ( Behavioral )
        port map ( clk => clk, reset => reset, wen => decoded(0), d => d,
                   FF_in => FF_in, FE_in => FE_in, IA_in => IA_in,
                   q => array_reg(0), OE => i_oe, IO =>IO, SLFM => SLFM, PDMM => PDMM, IE => IE,
                   RF => RF, FF => FF, FE => FE, IA => IA, FIL => FIL );
    OE <= i_oe;

    cdr_en <= decoded(1) and not i_oe;
    CDR_reg: entity work.generic_register ( Behavioral )
    generic map ( bits => 16 )
    port map ( clk => clk, reset => reset, enable => cdr_en,
               d => d(15 downto 0), q => array_reg(1)(15 downto 0));

    bcr_en <= decoded(2) and not i_oe;
    BCR_reg: entity work.generic_register ( Behavioral )
    generic map ( bits => 16 )
    port map ( clk => clk, reset => reset, enable => bcr_en,
               d => d(15 downto 0), q => array_reg(2)(15 downto 0));

    dcr_en <= decoded(3);
    DCR_reg: entity work.generic_register ( Behavioral )
    generic map ( bits => 16 )
    port map ( clk => clk, reset => reset, enable => dcr_en,
               d => d(15 downto 0), q => array_reg(3)(15 downto 0));


    array_reg(1)(31 downto 16) <= ( others => '0' );
    array_reg(2)(31 downto 16) <= ( others => '0' );
    array_reg(3)(31 downto 16) <= ( others => '0' );

    q <= array_reg ( to_integer( unsigned( r_addr (3 downto 2))));
    CDR <= array_reg(1)(15 downto 0);
    BCR <= array_reg(2)(15 downto 0);
    DCR <= array_reg(3)(15 downto 0);


end Behavioral;
```

PM_CSR

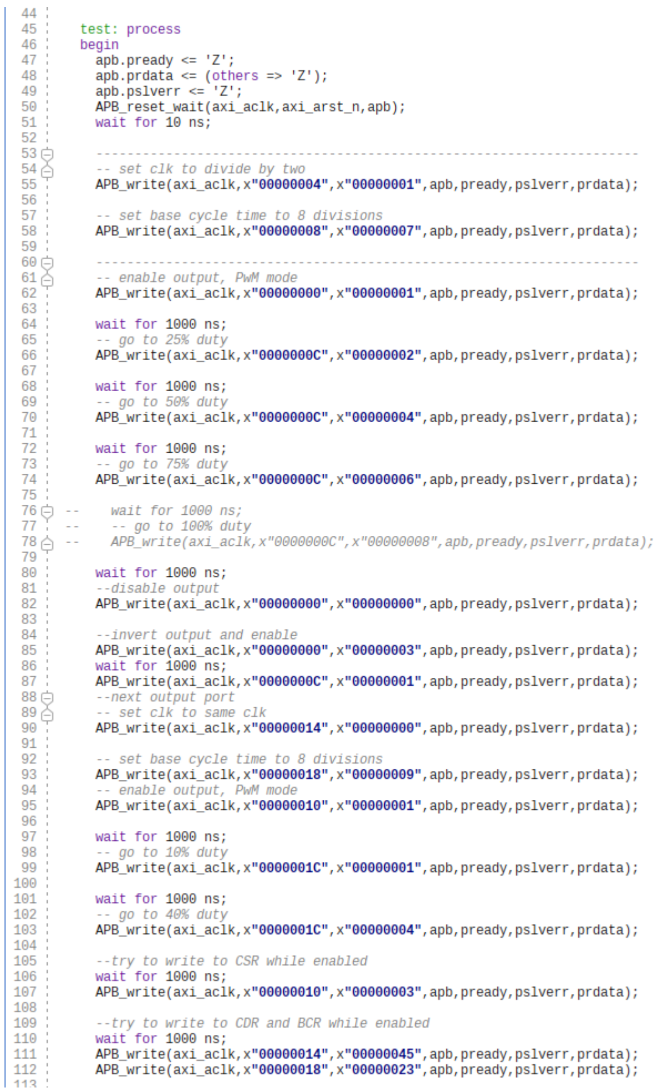


```vhdl

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


entity PM_CSR is
    Port ( clk ,reset, wen : in std_logic;
           d : in STD_LOGIC_VECTOR (31 downto 0);
           FF_in, FE_in, IA_in : in STD_LOGIC;
           q: out std_logic_vector (31 downto 0);
           OE, IO, SLFM, PDMM, IE, RF, FF, FE, IA : out STD_LOGIC;
           FIL : out std_logic_vector (4 downto 0)
           );
end PM_CSR;

architecture Behavioral of PM_CSR is
    signal master_enable : std_logic;
    signal q_oe : std_logic_vector (0 downto 0);
    signal q_rw : std_logic_vector (3 downto 0);
    signal q_fil : std_logic_vector (4 downto 0);

begin
    master_enable <= wen and not q_oe(0);
    --output enable register
    oe_reg: entity work.generic_register ( Behavioral )
    generic map ( bits => 1 )
    port map ( clk => clk, reset => reset, enable => wen,
               d => d(0 downto 0), q => q_oe );

    --regular read/write data register
    rw_reg: entity work.generic_register ( Behavioral )
    generic map ( bits => 4 )
    port map ( clk => clk, reset => reset, enable => master_enable,
               d => d(4 downto 1), q => q_rw );

    --fifo interupt level
    fil_reg: entity work.generic_register ( Behavioral )
    generic map ( bits => 5 )
    port map ( clk => clk, reset => reset, enable => master_enable,
               d => d(31 downto 27), q => q_fil );

    --map outputs
    q <= q_fil & "0000000000000000" & IA_in & FE_in & FF_in & "000" & q_rw & q_oe;
    OE <= q_oe(0);
    IO <= q_rw(0);
    SLFM <= q_rw(1);
    PDMM <= q_rw(2);
    IE <= q_rw(3);
    RF <= master_enable and d(5);
    FIL <= q_fil;
    FF <= FF_in;
    FE <= FE_in;
    IA <= IA_in;

end Behavioral;
```

Testbench VHDL

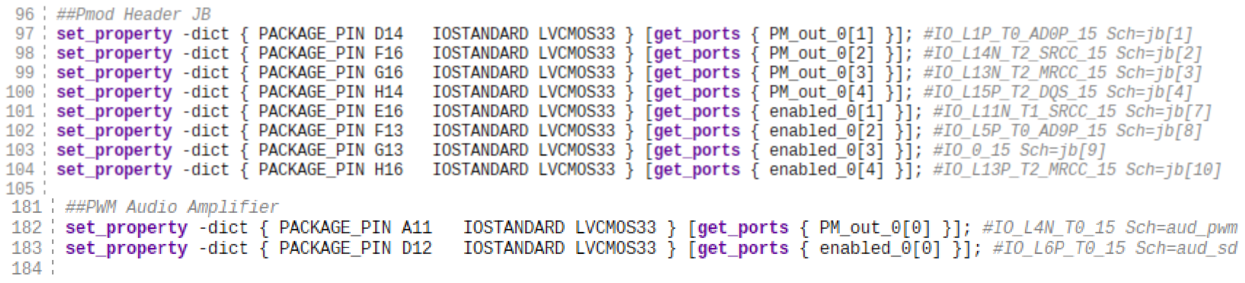


```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.APB_test_package.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

entity APB_PM_testbench is
end APB_PM_testbench;

architecture Behavioral of APB_PM_testbench is
  signal axi_aclk, axi_arst_n :std_logic := '0';
  signal apb : apb_bus_t;
  signal PM_out,enabled : std_logic_vector(4 downto 0);
  signal interrupt, pready, pslverr : std_logic;
  signal prdata : std_logic_vector(31 downto 0);

begin
  apb.pready  <= pready;
  apb.prdata  <= prdata;
  apb.pslverr <= pslverr;

  axi_aclk <= not axi_aclk after 5 ns;
  axi_arst_n <= '1' after 20 ns;

  uut : entity work.LDP_001_PM(behavioral)
    port map(
      axi_aclk    => axi_aclk,
      axi_arst_n => axi_arst_n,
      s_apb_paddr   => apb.paddr,
      s_apb_psel    => apb.psel,
      s_apb_penable => apb.penable,
      s_apb_pwrite  => apb.pwrite,
      s_apb_pwdata  => apb.pwdata,
      s_apb_pready  => pready,
      s_apb_prdata  => prdata,
      s_apb_pslverr => pslverr,
      -- Port to dr
      PM_out        => PM_out,
      enabled       => enabled
      --interrupt      => interrupt
      );

```



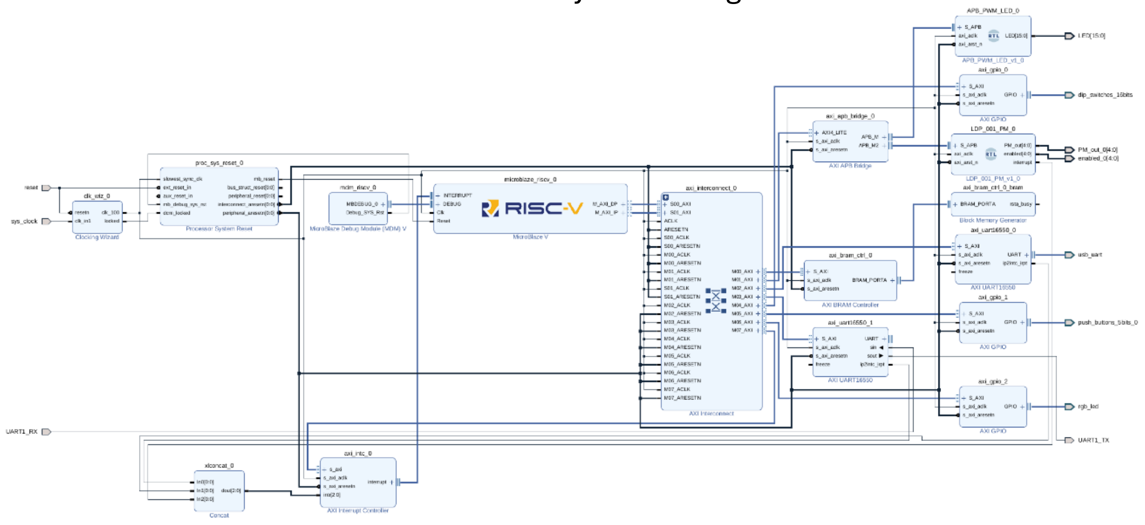
```
  test: process
  begin
    apb.pready <= 'Z';
    apb.prdata <= (others => 'Z');
    apb.pslverr <= 'Z';
    APB_reset_wait(axi_aclk,axi_arst_n,apb);
    wait for 10 ns;

    ----------------------------------------------------------------------
    -- set clk to divide by two
    APB_write(axi_aclk,x"00000004",x"00000001",apb,pready,pslverr,prdata);

    -- set base cycle time to 8 divisions
    APB_write(axi_aclk,x"00000008",x"00000007",apb,pready,pslverr,prdata);

    ----------------------------------------------------------------------
    -- enable output, PWM mode
    APB_write(axi_aclk,x"00000000",x"00000001",apb,pready,pslverr,prdata);

    wait for 1000 ns;
    -- go to 25% duty
    APB_write(axi_aclk,x"0000000C",x"00000002",apb,pready,pslverr,prdata);

    wait for 1000 ns;
    -- go to 50% duty
    APB_write(axi_aclk,x"0000000C",x"00000004",apb,pready,pslverr,prdata);

    wait for 1000 ns;
    -- go to 75% duty
    APB_write(axi_aclk,x"0000000C",x"00000006",apb,pready,pslverr,prdata);

--    wait for 1000 ns;
--    -- go to 100% duty
--    APB_write(axi_aclk,x"0000000C",x"00000008",apb,pready,pslverr,prdata);

    wait for 1000 ns;
    --disable output
    APB_write(axi_aclk,x"00000000",x"00000000",apb,pready,pslverr,prdata);

    --invert output and enable
    APB_write(axi_aclk,x"00000000",x"00000003",apb,pready,pslverr,prdata);
    wait for 1000 ns;
    APB_write(axi_aclk,x"0000000C",x"00000001",apb,pready,pslverr,prdata);
    --next output port
    -- set clk to same clk
    APB_write(axi_aclk,x"00000014",x"00000000",apb,pready,pslverr,prdata);

    -- set base cycle time to 8 divisions
    APB_write(axi_aclk,x"00000018",x"00000009",apb,pready,pslverr,prdata);
    -- enable output, PWM mode
    APB_write(axi_aclk,x"00000010",x"00000001",apb,pready,pslverr,prdata);

    wait for 1000 ns;
    -- go to 10% duty
    APB_write(axi_aclk,x"0000001C",x"00000001",apb,pready,pslverr,prdata);

    wait for 1000 ns;
    -- go to 40% duty
    APB_write(axi_aclk,x"0000001C",x"00000004",apb,pready,pslverr,prdata);

    --try to write to CSR while enabled
    wait for 1000 ns;
    APB_write(axi_aclk,x"00000010",x"00000003",apb,pready,pslverr,prdata);

    --try to write to CDR and BCR while enabled
    wait for 1000 ns;
    APB_write(axi_aclk,x"00000014",x"00000045",apb,pready,pslverr,prdata);
    APB_write(axi_aclk,x"00000018",x"00000023",apb,pready,pslverr,prdata);
```

Constraint File



```
##Pmod Header JB
set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33 } [get_ports { PM_out_0[1] }]; #IO_L1P_T0_AD0P_15 Sch=jb[1]
set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 } [get_ports { PM_out_0[2] }]; #IO_L14N_T2_SRCC_15 Sch=jb[2]
set_property -dict { PACKAGE_PIN G16   IOSTANDARD LVCMOS33 } [get_ports { PM_out_0[3] }]; #IO_L13N_T2_MRCC_15 Sch=jb[3]
set_property -dict { PACKAGE_PIN H14   IOSTANDARD LVCMOS33 } [get_ports { PM_out_0[4] }]; #IO_L15P_T2_DQS_15 Sch=jb[4]
set_property -dict { PACKAGE_PIN E16   IOSTANDARD LVCMOS33 } [get_ports { enabled_0[1] }]; #IO_L11N_T1_SRCC_15 Sch=jb[7]
set_property -dict { PACKAGE_PIN F13   IOSTANDARD LVCMOS33 } [get_ports { enabled_0[2] }]; #IO_L5P_T0_AD9P_15 Sch=jb[8]
set_property -dict { PACKAGE_PIN G13   IOSTANDARD LVCMOS33 } [get_ports { enabled_0[3] }]; #IO_0_15 Sch=jb[9]
set_property -dict { PACKAGE_PIN H16   IOSTANDARD LVCMOS33 } [get_ports { enabled_0[4] }]; #IO_L13P_T2_MRCC_15 Sch=jb[10]

##PWM Audio Amplifier
set_property -dict { PACKAGE_PIN A11   IOSTANDARD LVCMOS33 } [get_ports { PM_out_0[0] }]; #IO_L4N_T0_15 Sch=aud_pwm
set_property -dict { PACKAGE_PIN D12   IOSTANDARD LVCMOS33 } [get_ports { enabled_0[0] }]; #IO_L6P_T0_15 Sch=aud_sd
```

Overview of System Design


reset
sys_clock
UART1_RX
clk_wiz_0
resetn
clk_in1
clk_100
locked
Clocking Wizard
proc_sys_reset_0
slowest_sync_clk
ext_reset_in
aux_reset_in
mb_debug_sys_rst
dcm_locked
mb_reset
bus_struct_reset[0:0]
peripheral_reset[0:0]
interconnect_aresetn[0:0]
peripheral_aresetn[0:0]
Processor System Reset
mdm_riscv_0
MBDEBUG_0
Debug_SYS_Rst
MicroBlaze Debug Module (MDM) V
microblaze_riscv_0
INTERRUPT
DEBUG
Clk
Reset
M_AXI_DP
M_AXI_IP
MicroBlaze V
axi_interconnect_0
S00_AXI
S01_AXI
ACLK
ARESETN
S00_ACLK
S00_ARESETN
M00_ACLK
M00_ARESETN
M01_ACLK
M01_ARESETN
S01_ACLK
S01_ARESETN
M02_ACLK
M02_ARESETN
M03_ACLK
M03_ARESETN
M04_ACLK
M04_ARESETN
M05_ACLK
M05_ARESETN
M06_ACLK
M06_ARESETN
M07_ACLK
M07_ARESETN
M00_AXI
M01_AXI
M02_AXI
M03_AXI
M04_AXI
M05_AXI
M06_AXI
M07_AXI
AXI Interconnect
xlconcat_0
In0[0:0]
In1[0:0]
In2[0:0]
dout[2:0]
Concat
axi_intc_0
s_axi
s_axi_aclk
s_axi_aresetn
intr[2:0]
interrupt
AXI Interrupt Controller
axi_apb_bridge_0
AXI4_LITE
s_axi_aclk
s_axi_aresetn
APB_M
APB_M2
AXI APB Bridge
axi_bram_ctrl_0
S_AXI
s_axi_aclk
s_axi_aresetn
BRAM_PORTA
AXI BRAM Controller
axi_uart16550_1
S_AXI
s_axi_aclk
s_axi_aresetn
freeze
UART
sin
sout
ip2intc_irpt
AXI UART16550
APB_PWM_LED_0
S_APB
axi_aclk
axi_arst_n
RTL
LED[15:0]
APB_PWM_LED_v1_0
axi_gpio_0
S_AXI
s_axi_aclk
s_axi_aresetn
GPIO
AXI GPIO
LDP_001_PM_0
S_APB
axi_aclk
axi_arst_n
RTL
PM_out[4:0]
enabled[4:0]
interrupt
LDP_001_PM_v1_0
axi_bram_ctrl_0_bram
BRAM_PORTA
rsta_busy
Block Memory Generator
axi_uart16550_0
S_AXI
s_axi_aclk
s_axi_aresetn
freeze
UART
ip2intc_irpt
AXI UART16550
axi_gpio_1
S_AXI
s_axi_aclk
s_axi_aresetn
GPIO
AXI GPIO
axi_gpio_2
S_AXI
s_axi_aclk
s_axi_aresetn
GPIO
AXI GPIO
LED[15:0]
dip_switches_16bits
PM_out_0[4:0]
enabled_0[4:0]
usb_uart
push_buttons_5bits_0
rgb_led
UART1_TX