Lab 7c: Pulse Modulator

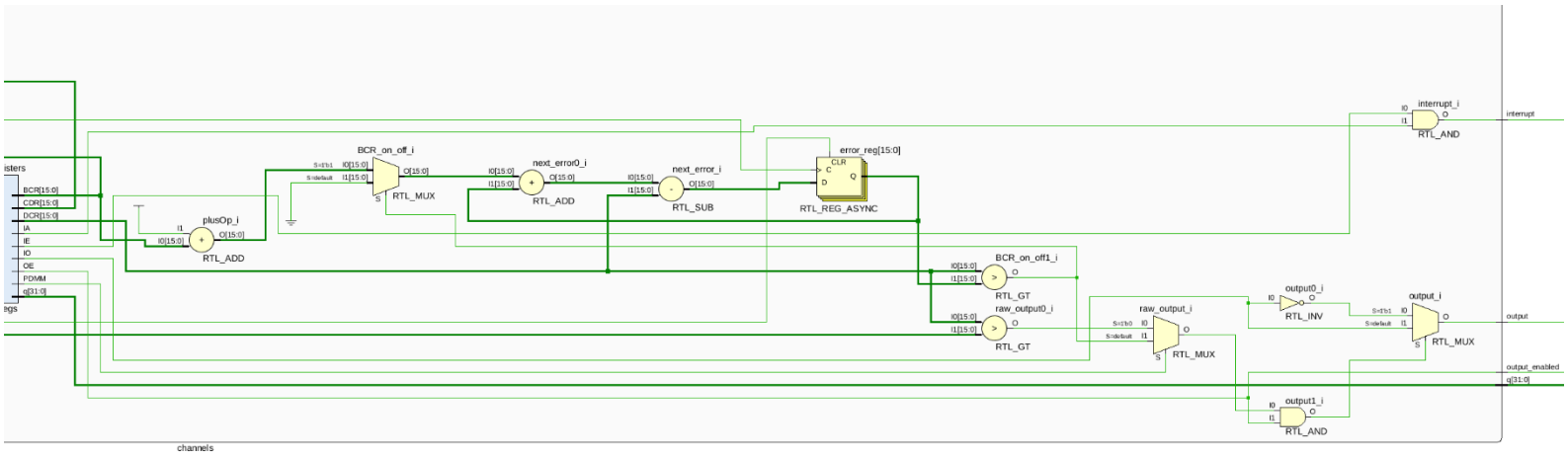
-Overview-

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-

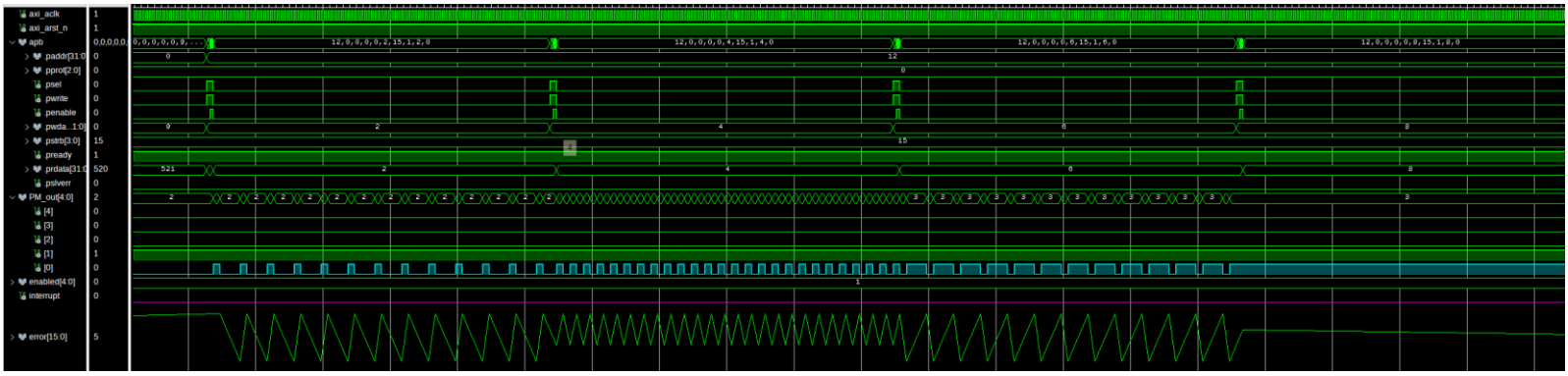
Channel Level

The only part that changed from the previous designs was the addition of the PDM logic that is selected by the PDMM. To make the PDM, an error register was set up to track how far off the value of the output was from the wanted value over time. If the error value was less than the DCR, the PDM would output a 1 as the error is not high enough to matter. Once the error accumulates greater than the DCR, the value has been high for long enough and can go down to 0 for a while. Then the error decreases until it is lower than the DCR, were it triggers the PDM output high again. Essentially, when the PDM output is 0, the error decreases at the rate of the DCR value (higher DCR value means the error decreases faster and the output switches to 1 faster). When the PDM output is 1, the error increases at a rate that is equal to the max value (BCR+1) minus the DCR (the higher the DCR, the slower the error increases and the longer the output stays high).

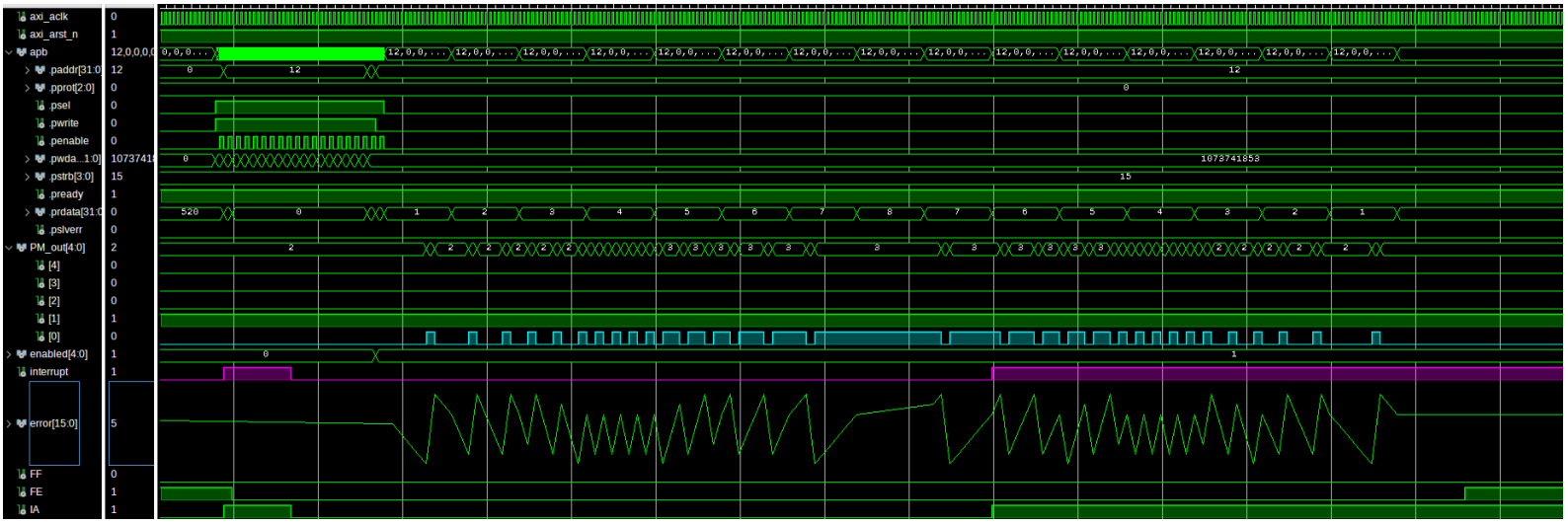


-Simulation –

In this simulation, you can see the DCR value increasing over time, and the output (cyan) changes with the DCR. This looks a lot like PWM, as the values aren't changing. At the bottom, you can see the error value change overtime. It decreases slower and increases faster when the value of the DCR is smaller, and it decreases faster and increase slower when the DCR is bigger.



In this simulation, both the PDM and FIFO are being used. On the left, the values are loaded into the FIFO and enabled shortly after. You can see the output (cyan), slowly increases in density and decreases in density as the DCR goes up and down. This is different from the PWM, as the signal doesn't always end up going high when the base counter resets but can go high when it is signaled to by the error. You can see that the error values go from a left leaning peak (increase faster, decrease slower), to a right leaning peak (increase slower and decrease faster), and back to a left leaning peak. In the middle, you can even see the error flatten out as the error doesn't increase at all when the DCR = BCR +1. Finally, you can see the interrupt go high when the FIFO has less than 8 values left.



-Conclusion-

Most of my time was taken wrapping my head around how the PDM worked. Once I got a grasp of it, implementing it was easy and it only too about 6 lines of code. Overall, this part went great.

-Appendix-

Main VHDL

LDP_001_PM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use ieee.std_logic_misc.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);
    interrupt       :out std_logic
    );
  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 ) := (others => '0');
    signal interrupt_vector : std_logic_vector( channels-1 downto 0 ) := (others => '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),
                   interrupt => interrupt_vector(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 ) ) ) );

--interrupt
interrupt <= or_reduce ( interrupt_vector );

end Behavioral;
```

Channel

```
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, interrupt : out std_logic
          );
end channels;

architecture Behavioral of channels is
    signal div_clk_vector : std_logic_vector ( 15 downto 0 );
    signal div_reset, base_reset, reset_div, reset_base, compared : std_logic;
    signal base_clk : std_logic := '1';
    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 : std_logic_vector (15 downto 0 );
    signal DCR : std_logic_vector (15 downto 0 ) := (others => '0' );
    signal BCR_on_off, error : std_logic_vector ( 15 downto 0 ) := x"0000";
    signal next_error : std_logic_vector ( 15 downto 0 ) := x"0000";
    signal pwm_out, pdm_out, raw_output : std_logic;

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, base_reset => base_reset, div_reset => div_reset,
               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 );

div_reset <= '1' when div_clk_vector = CDR else '0';
reset_div <= div_reset or reset;
modulo_clock_div : entity work.generic_counter ( Behavioral )
    generic map ( bits => 16 )
    port map ( clk => clk, reset => reset_div, q => div_clk_vector );

--base counter
base_reset <= '1' when compare_val = BCR else '0';
reset_base <= base_reset or reset;
base_clk <= clk when CDR = x"0000" else div_reset when rising_edge(clk);
base_counter : entity work.generic_counter ( Behavioral )
    generic map ( bits => 16 )
    port map ( clk => base_clk, reset => reset_base, q => compare_val );

--PWM
compared <= '1' when DCR > compare_val else '0';
pwm_out <= '1' when compared = '1' else '0';

--PDM
--error register
error <= x"0000" when reset = '1' else next_error when rising_edge(base_clk);

BCR_on_off <= std_logic_vector ( unsigned( BCR ) + 1 ) when pdm_out = '1' else x"0000";

next_error <= std_logic_vector ( unsigned ( BCR_on_off ) + unsigned ( error )  - unsigned ( DCR ) ) ;

pdm_out <= '1' when DCR > error else '0';


raw_output <= pwm_out when PDMM = '0' else pdm_out;
--outputs

output <= not IO when ( raw_output and OE )= '1' else IO;
output_enabled <= OE;
interrupt <= IE and IA;

end Behavioral;
```

PM_Registers
(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;
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);
           base_reset, div_reset : 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 cdr_en, bcr_en, dcr_en : std_logic;
    signal OE_i, SLFM_i, IE_i, RF_i, FF_i, FE_i, IA_i  : std_logic;
    signal FIL_i : std_logic_vector (4 downto 0);
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_i, FE_in => FE_i, IA_in => IA_i,
                   q => array_reg(0), OE => OE_i, IO =>IO, SLFM => SLFM_i, PDMM => PDMM, IE => IE,
                   RF => RF_i, FF => FF, FE => FE, IA => IA, FIL => FIL_i );

    cdr_en <= decoded(1) and not OE_i;
    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 OE_i;
    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_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));


    --DCR fifo
    dcr_en <= decoded(3);
    DCR_fifo: entity work. generic_FIFO ( behavioral )
    generic map ( bits => 16, depth => 32 )
    port map ( clk => clk, rst => RF_i, wdata => d(15 downto 0), wen => dcr_en, ren => base_reset and div_reset and OE_i, enable_fifo => SLFM_i, threshold => FIL_i,
               data => array_reg(3)(15 downto 0), empty => FE_i, full => FF_i, below_threshold => IA_i);


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

    --outputs
    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);

    OE <= OE_i;
    SLFM <= SLFM_i;
    FIL <= FIL_i;

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
    -- disable output
    APB_write(axi_aclk,x"00000000",x"00000000",apb,pready,pslverr,prdata);
    wait for 200 ns;
    -- enable output, PDM mode
    APB_write(axi_aclk,x"00000000",x"00000009",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;
```

```vhdl
    ------------------------------------------------------------------
    -- turn on PDM fifo mode and enable interrupts at level 4
    APB_write(axi_aclk,x"00000000",x"00000000",apb,pready,pslverr,prdata);
    wait for 200 ns;
    APB_write(axi_aclk,x"00000000",x"4000001C",apb,pready,pslverr,prdata);

    -- go to 0% duty
    APB_write(axi_aclk,x"0000000C",x"00000000",apb,pready,pslverr,prdata);
    -- go to 12.5% duty
    APB_write(axi_aclk,x"0000000C",x"00000001",apb,pready,pslverr,prdata);
    -- go to 25% duty
    APB_write(axi_aclk,x"0000000C",x"00000002",apb,pready,pslverr,prdata);
    -- go to 37.5% duty
    APB_write(axi_aclk,x"0000000C",x"00000003",apb,pready,pslverr,prdata);
    -- go to 50.5% duty
    APB_write(axi_aclk,x"0000000C",x"00000004",apb,pready,pslverr,prdata);
    -- go to 62.5% duty
    APB_write(axi_aclk,x"0000000C",x"00000005",apb,pready,pslverr,prdata);
    -- go to 755% duty
    APB_write(axi_aclk,x"0000000C",x"00000006",apb,pready,pslverr,prdata);
    -- go to 87.5% duty
    APB_write(axi_aclk,x"0000000C",x"00000007",apb,pready,pslverr,prdata);
    -- go to 100% duty
    APB_write(axi_aclk,x"0000000C",x"00000008",apb,pready,pslverr,prdata);

    -- ramp back down
    APB_write(axi_aclk,x"0000000C",x"00000007",apb,pready,pslverr,prdata);
    -- go to 12.5% duty
    APB_write(axi_aclk,x"0000000C",x"00000006",apb,pready,pslverr,prdata);
    -- go to 25% duty
    APB_write(axi_aclk,x"0000000C",x"00000005",apb,pready,pslverr,prdata);
    -- go to 37.5% duty
    APB_write(axi_aclk,x"0000000C",x"00000004",apb,pready,pslverr,prdata);
    -- go to 50.5% duty
    APB_write(axi_aclk,x"0000000C",x"00000003",apb,pready,pslverr,prdata);
    -- go to 62.5% duty
    APB_write(axi_aclk,x"0000000C",x"00000002",apb,pready,pslverr,prdata);
    -- go to 755% duty
    APB_write(axi_aclk,x"0000000C",x"00000001",apb,pready,pslverr,prdata);
    -- go to 87.5% duty
    APB_write(axi_aclk,x"0000000C",x"00000000",apb,pready,pslverr,prdata);

    APB_write(axi_aclk,x"00000000",x"4000001D",apb,pready,pslverr,prdata);
```

Overview of System Design