Lab 7b: Pulse Modulator

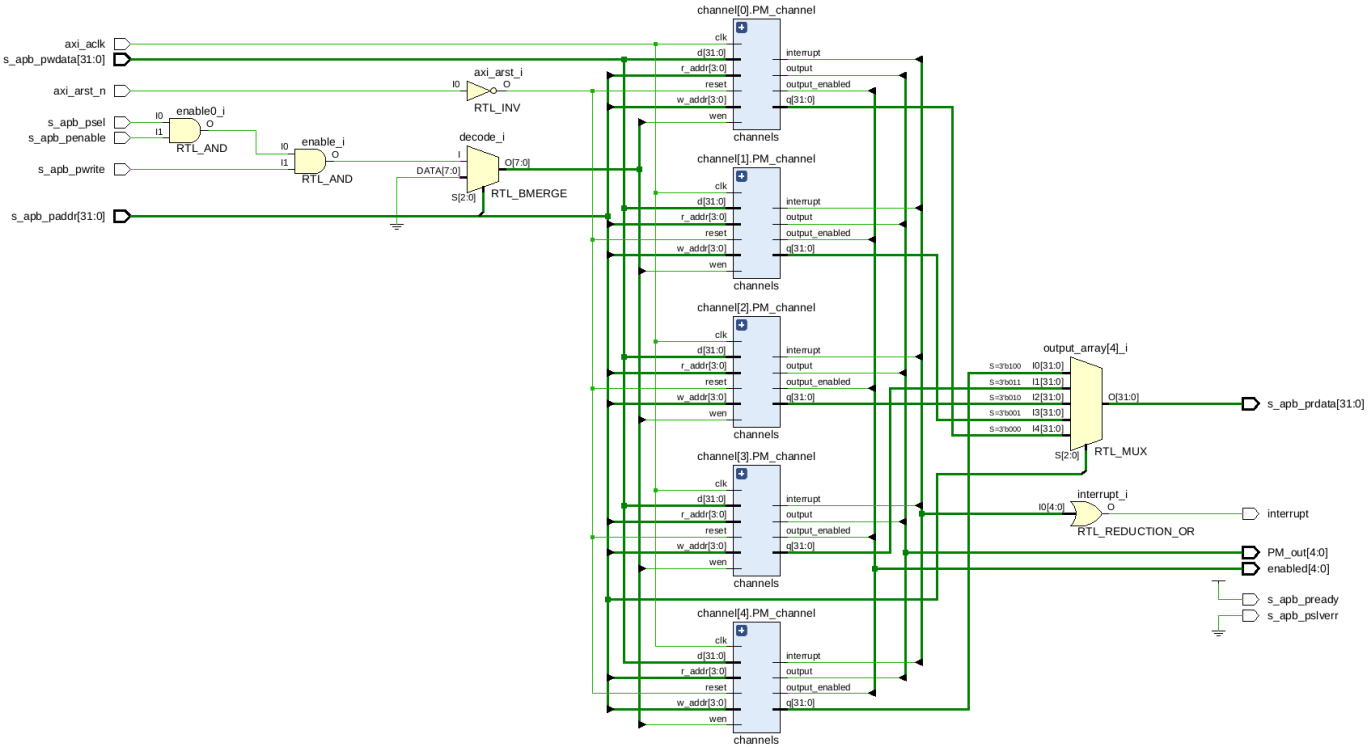
-Overview-

This week, I implemented the FIFO that will feed new values into the PWM over time, so the values can be set and the device will automatically step through the new values.

-Design-

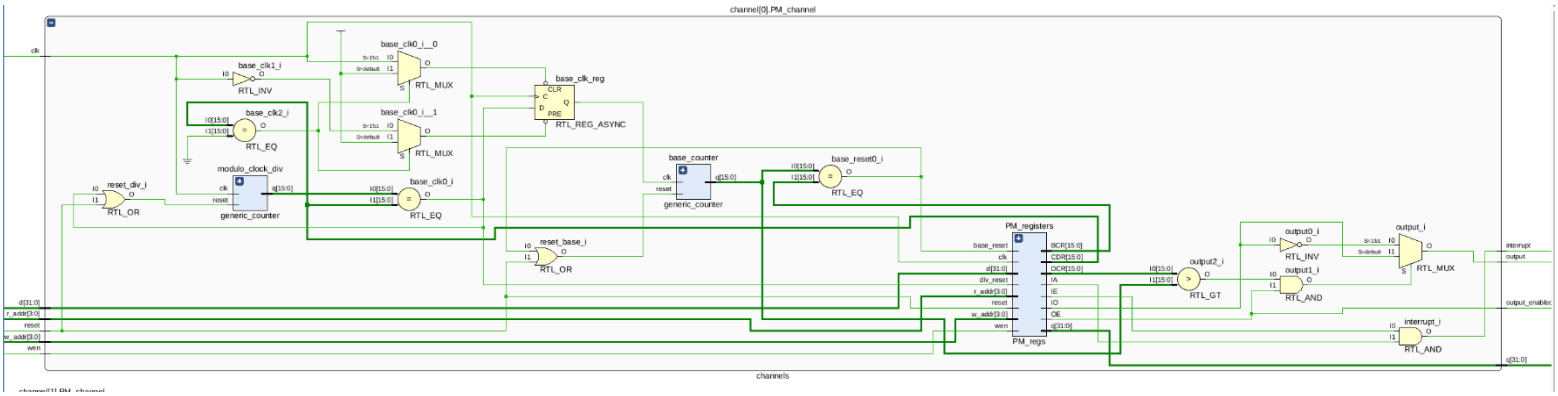
Highest Level

The only this that changed at this level is that an OR reduction was added for the interrupt. Other than that, this still just organizes and guides signals to the different channels.



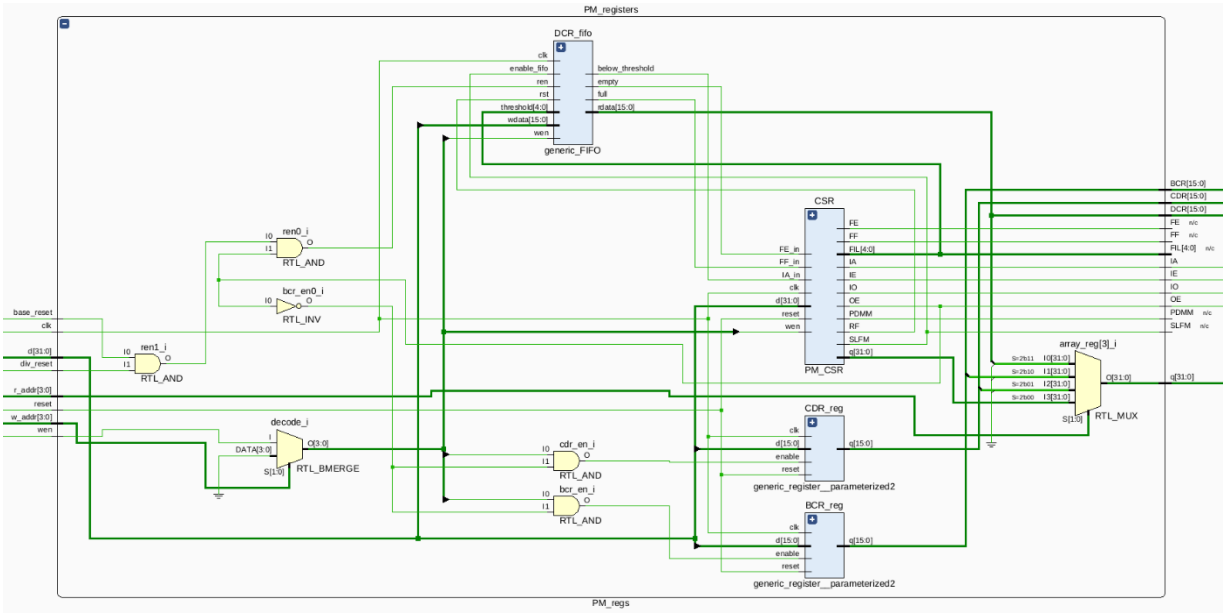
Channel Level

There are a few changes at this level. First, on the right side and interrupt was added to trigger when the interrupt was active, and when the FIFO was signaling that it is below threshold. Also, I changed the clock divider to handle when the CDR is set to 0. Previously, the base clock would just stay high the entire time as the counter was always equal to the CDR. In the new design, a register is set after the compare and doesn't affect any values 1 and above. In the special case of 0, the muxes before the register are active, and set and reset the values of the register based on the values of the clock. This makes it output the same as the global clock.



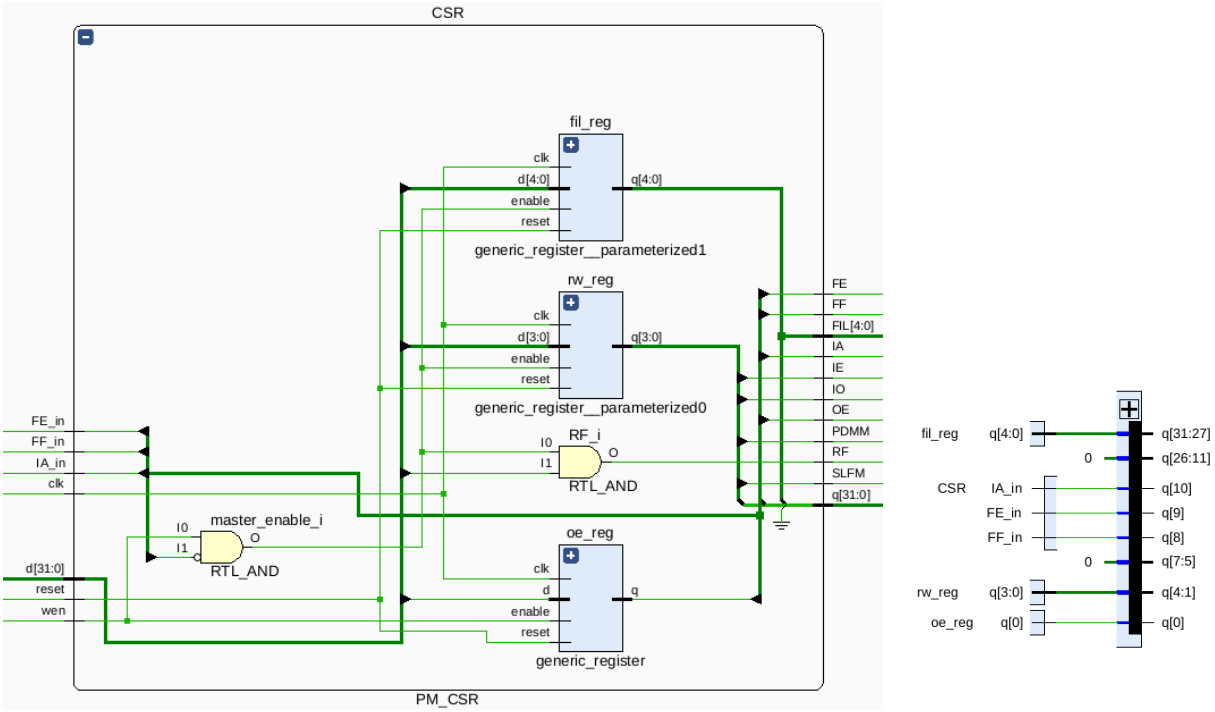
PM Registers Level

The DCR registers have been changed to the FIFO entity. When the FIFO is not active, the FIFO acts like a normal register and doesn't affect the design. When the FIFO is enabled, the values are outputted one at a time. This only happens when the output is enabled, the base counter is getting reset, and the clock divider is getting reset. This generates a one clock pulse that only cycles to the next number once a full base counter cycle. Other than that logic, the other signals just get routed into the CSR to be read by the controller or used by other parts of the channel.



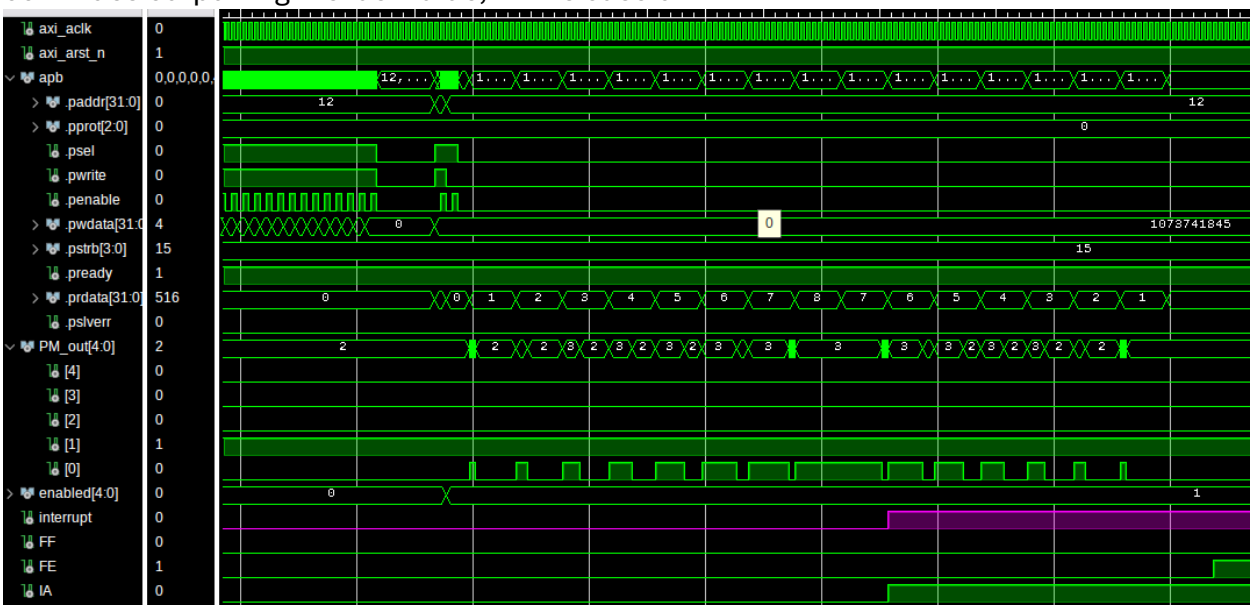
CSR Register

This didn't change at all from the previous part.



-Simulation –

On the right, you can see all the values getting loaded into the FIFO. When the output gets enabled thereafter, each value comes out of the FIFO one at a time without any input from the controller. In the interrupt activated when the amount of left in the FIFO is below 8, and the interrupt goes high (purple). You can also see at the bottom. The FE in the CSR goes high when the FIFO runs out of values. Once the FIFO runs out of values, it continues outputting the last value, in this case 0.



-Conclusion-

This part took a little time to figure out how I wanted to implement the FIFO, either replacing the DCR register or going in parallel with it. After playing with the FIFO, I saw it worked just like a register when not activated, so I decided to simply replace the DCR register with it as I figured it would be easier. Fixing the base clock took a little time to figure out, but the FIFO wasn't too much of a challenge with time.

-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, 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, 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 );



--outputs
compared <= '1' when DCR > compare_val else '0';
output <= not IO when (compared 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,
               rdata => 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
    ------------------------------------------------------------------
    -- turn on fifo mode and enable interrupts at level 8
    APB_write(axi_aclk,x"00000000",x"00000000",apb,pready,pslverr,prdata);
    APB_write(axi_aclk,x"00000000",x"00000004",apb,pready,pslverr,prdata);


    wait for 40 ns;
    -- 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 87.5% duty
    APB_write(axi_aclk,x"0000000C",x"00000006",apb,pready,pslverr,prdata);
    -- go to 75% duty
    APB_write(axi_aclk,x"0000000C",x"00000005",apb,pready,pslverr,prdata);
    -- go to 62.5% duty
    APB_write(axi_aclk,x"0000000C",x"00000004",apb,pready,pslverr,prdata);
    -- go to 50% duty
    APB_write(axi_aclk,x"0000000C",x"00000003",apb,pready,pslverr,prdata);
    -- go to 37.5% duty
    APB_write(axi_aclk,x"0000000C",x"00000002",apb,pready,pslverr,prdata);
    -- go to 25% duty
    APB_write(axi_aclk,x"0000000C",x"00000001",apb,pready,pslverr,prdata);
    -- go to 12.5% duty
    APB_write(axi_aclk,x"0000000C",x"00000000",apb,pready,pslverr,prdata);
    wait for 100 ns;
    APB_write(axi_aclk,x"00000000",x"40000015",apb,pready,pslverr,prdata);

    --read from DCR
    APB_read(axi_aclk,x"0000000C",apb,pready,pslverr,prdata);

    ------------------------------------------------------------------
     wait for 6000 ns;
     --disable fifo, run
     APB_write(axi_aclk,x"00000000",x"00000000",apb,pready,pslverr,prdata);
     APB_write(axi_aclk,x"00000000",x"00000001",apb,pready,pslverr,prdata);
     wait for 1000 ns;
```

Overview of System Design