

Lab 7b: Pulse Modulator

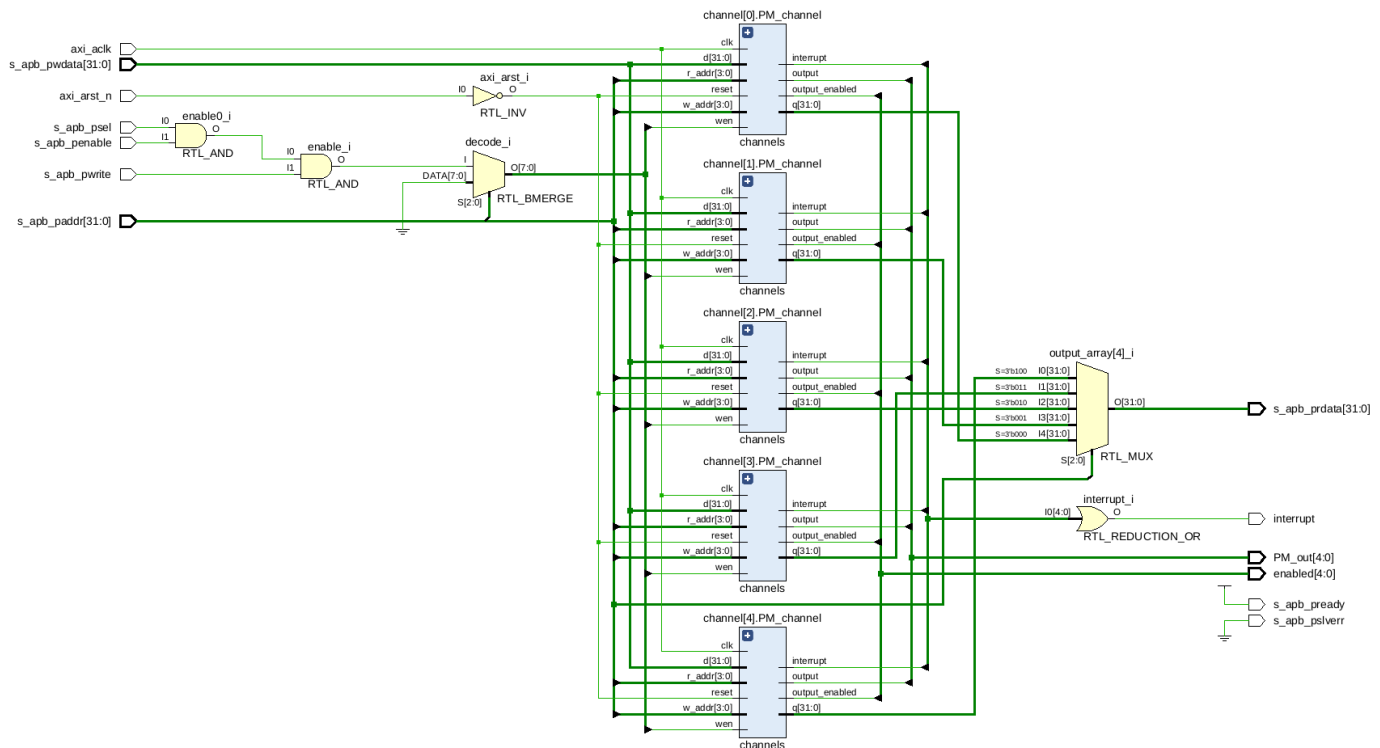
-Overview-

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-

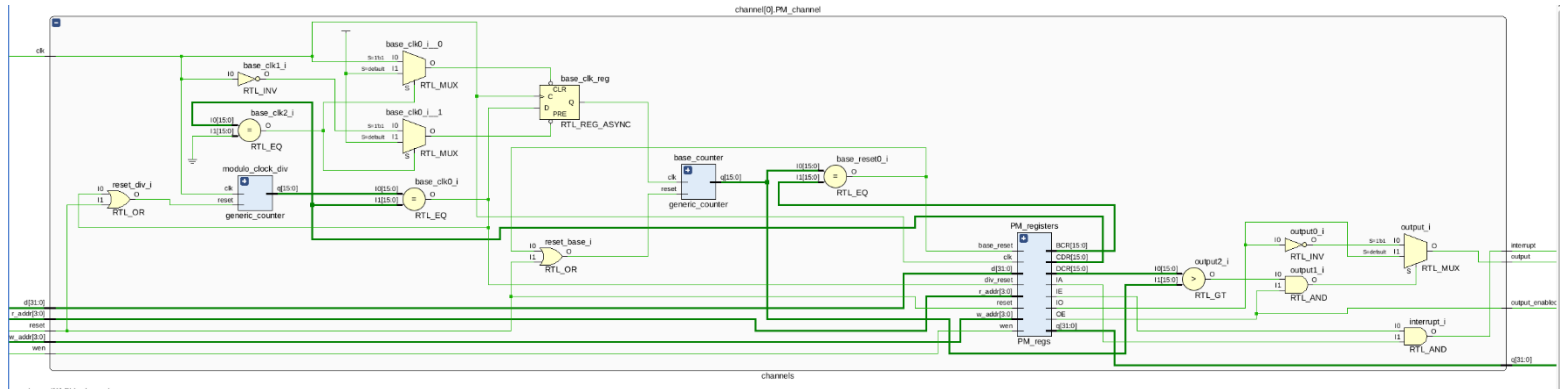
Highest Level

The only thing 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.



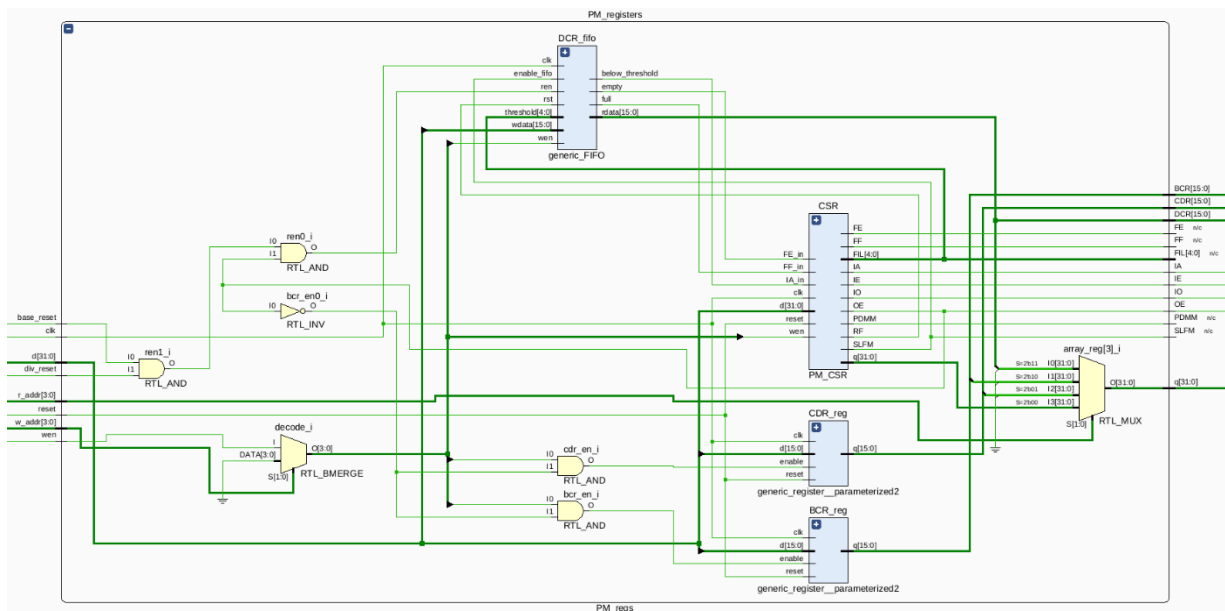
Channel Level

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 base 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.



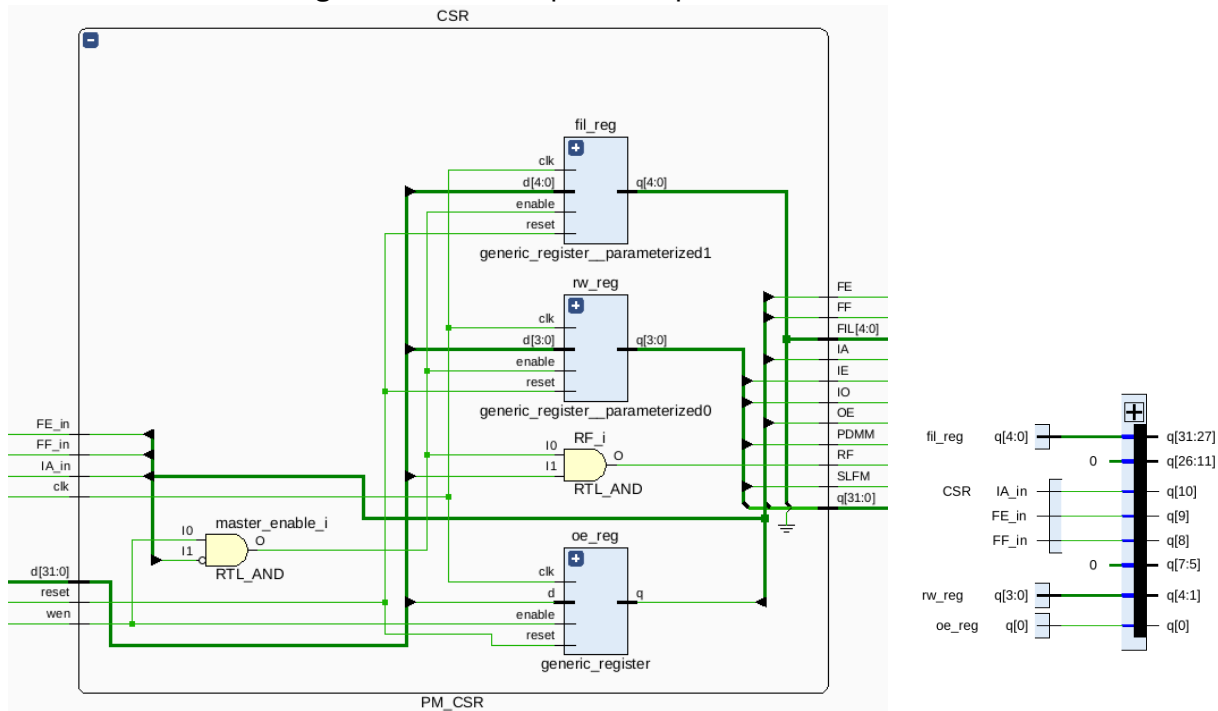
PM Registers Level

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.



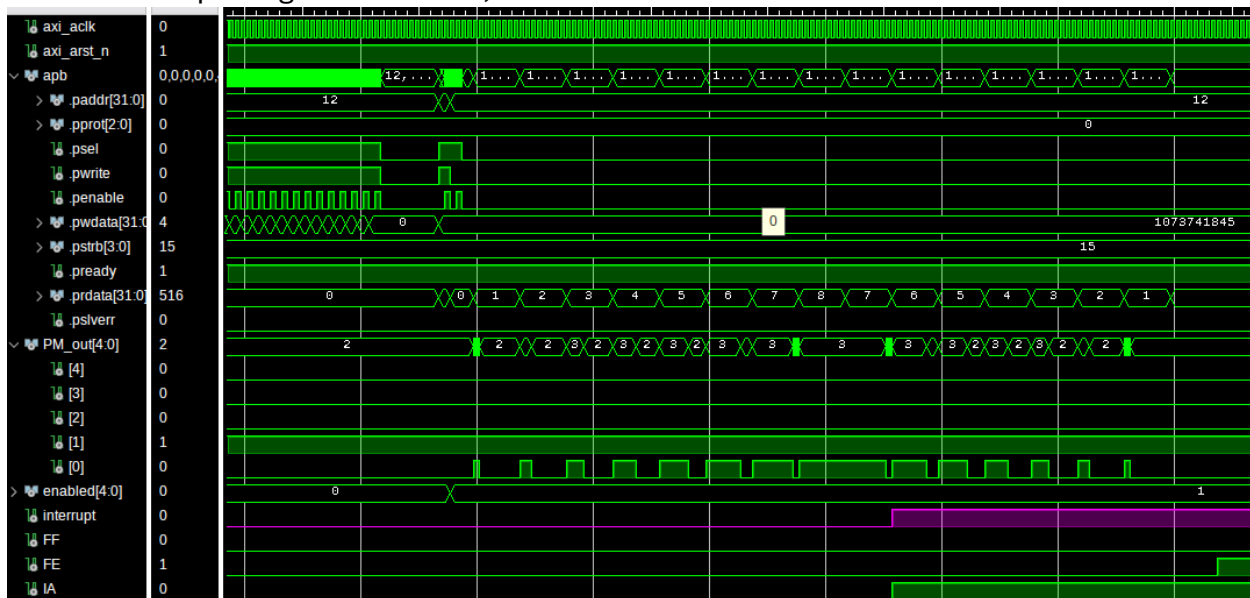
CSR Register

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

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.numeric_std.all;
4 use IEEE.std_logic_misc.all;
5 use work.m1_package.all;
6
7 entity LDP_001_PM is
8   generic ( channels : integer := 5 );
9   port (
10     -- Clock and reset
11     axi_aclck      :in  std_logic;
12     axi_arst_n     :in  std_logic;
13     -- The APB port
14     s_apb_paddr    :in  std_logic_vector(31 downto 0);
15     s_apb_psel     :in  std_logic;
16     s_apb_penable  :in  std_logic;
17     s_apb_pwrite   :in  std_logic;
18     s_apb_pwdata   :in  std_logic_vector(31 downto 0);
19     s_apb_pready   :out std_logic := '0';
20     s_apb_prdata   :out std_logic_vector(31 downto 0) := (others => '0');
21     s_apb_pslverr  :out std_logic := '0';
22     -- Port to drive the LEDs
23     PM_out         :out std_logic_vector(channels-1 downto 0);
24     enabled        :out std_logic_vector(channels-1 downto 0);
25     interrupt      :out std_logic
26   );
27   ATTRIBUTE X_INTERFACE_INFO : STRING;
28 end LDP_001_PM;
29
30 architecture Behavioral of LDP_001_PM is
31   ATTRIBUTE X_INTERFACE_INFO of s_apb_paddr :SIGNAL is
32     "xilinx.com:interface:apb:1.0 S_APB_PADDR";
33   ATTRIBUTE X_INTERFACE_INFO of s_apb_psel :SIGNAL is
34     "xilinx.com:interface:apb:1.0 S_APB_PSEL";
35   ATTRIBUTE X_INTERFACE_INFO of s_apb_penable :SIGNAL is
36     "xilinx.com:interface:apb:1.0 S_APB_PENABLE";
37   ATTRIBUTE X_INTERFACE_INFO of s_apb_pwrite :SIGNAL is
38     "xilinx.com:interface:apb:1.0 S_APB_PWRITE";
39   ATTRIBUTE X_INTERFACE_INFO of s_apb_pwdata :SIGNAL is
40     "xilinx.com:interface:apb:1.0 S_APB_PWDATA";
41   ATTRIBUTE X_INTERFACE_INFO of s_apb_pready :SIGNAL is
42     "xilinx.com:interface:apb:1.0 S_APB_PREADY";
43   ATTRIBUTE X_INTERFACE_INFO of s_apb_prdata :SIGNAL is
44     "xilinx.com:interface:apb:1.0 S_APB_PRDATA";
45   ATTRIBUTE X_INTERFACE_INFO of s_apb_pslverr :SIGNAL is
46     "xilinx.com:interface:apb:1.0 S_APB_PSLVERR";
47
48   -- Define any signals that your architecture needs.
49   signal axi_arst :std_logic;
50   signal enable_port :std_logic_vector ( 7 downto 0 ) := (others => '0');
51   signal interrupt_vector :std_logic_vector(channels-1 downto 0) := (others => '0');
52   type signal_array is array ( 4 downto 0 ) of std_logic_vector ( 31 downto 0 );
53   signal output_array :signal_array;
54 begin
55   enable <= s_apb_psel and s_apb_penable and s_apb_pwrite;
56   axi_arst <= not(axi_arst_n);
57   enable_port <= decode ( s_apb_paddr( 6 downto 4 ), enable );
58
59
60   channel : for i in 0 to channels-1 generate
61     PM_channel: entity work.channels ( Behavioral )
62     Port map ( clk => axi_aclck,
63               reset => axi_arst,
64               wen => enable_port(i),
65               w_addr => s_apb_paddr(31 downto 0),
66               r_addr => s_apb_paddr(31 downto 0),
67               d => s_apb_pwdata,
68               q => output_array(i),
69               output => PM_out(i),
70               output_enabled => enabled(i),
71               interrupt => interrupt_vector(i)
72             );
73   end generate;
74
75   --APB Outputs
76   s_apb_pready <= '1';
77   s_apb_pslverr <= '0';
78   s_apb_prdata <= output_array( to_integer ( unsigned ( s_apb_paddr (31 downto 4) ) ) );
79
80   --interrupt
81   interrupt <= or_reduce ( interrupt_vector );
82
83 end Behavioral;

```

Channel

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.Numeric_std.ALL;
4
5 entity channels is
6   Port (
7     clk, reset, wen : in std_logic;
8     w_addr, r_addr : in std_logic_vector ( 3 downto 0 );
9     d : in std_logic_vector ( 31 downto 0 );
10    q : out std_logic_vector ( 31 downto 0 );
11    output, output_enabled, interrupt : out std_logic
12  );
13 end channels;
14
15 architecture Behavioral of channels is
16   signal div_clk_vector :std_logic_vector ( 15 downto 0 );
17   signal div_reset, base_reset, reset_div, reset_base, compared : std_logic;
18   signal base_clk : std_logic := '1';
19   signal compare_val : std_logic_vector (15 downto 0 );
20   signal OE, IO, SLFM, PDMM, IE, RF, FF, FE, IA : std_logic;
21   signal FIL : std_logic_vector ( 4 downto 0 );
22   signal CDR, BCR, DCR : std_logic_vector (15 downto 0 );
23
24 begin
25
26   PM_registers : entity work.PM_regs ( Behavioral )
27   port map
28     ( clk => clk, reset => reset, wen => wen, w_addr => w_addr, r_addr => r_addr, d => d, base_reset => base_reset, div_reset => div_reset,
29       q => q, OE => OE, IO => IO, SLFM => SLFM, PDMM => PDMM, IE => IE, RF => RF,
30       FF => FF, FE => FE, IA => IA, FIL => FIL,
31       CDR => CDR, BCR => BCR, DCR => DCR );
32
33   div_reset <= '1' when div_clk_vector = CDR else '0';
34   reset_div <= div_reset or reset;
35   modulo_clock_div : entity work.generic_counter ( Behavioral )
36   generic map ( bits => 16 )
37   port map ( clk => clk, reset => reset_div, q => div_clk_vector );
38
39   --base counter
40   base_reset <= '1' when compare_val = BCR else '0';
41   reset_base <= base_reset or reset;
42   base_clk <= clk when CDR = x"0000" else div_reset when rising_edge(clk);
43   base_counter : entity work.generic_counter ( Behavioral )
44   generic map ( bits => 16 )
45   port map ( clk => base_clk, reset => reset_base, q => compare_val );
46
47   --outputs
48   compared <= '1' when DCR > compare_val else '0';
49   output <= not IO when (compared and OE) = '1' else IO;
50   output_enabled <= OE;
51   interrupt <= IE and IA;
52
53 end Behavioral;
54
55

```

PM_Registers

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

```

1: library IEEE;
2: use IEEE.STD_LOGIC_1164.ALL;
3: use IEEE.NUMERIC_STD.ALL;
4: use work.m1_package.all;
5:
6: --use UNISIM.VComponents.all;
7:
8: entity PM_regs is
9:   Port (
10:     clk, reset, wen : in STD_LOGIC;
11:     w_addr, r_addr : in STD_LOGIC_VECTOR (3 downto 0);
12:     d : in STD_LOGIC_VECTOR (31 downto 0);
13:     base_reset, div_reset : in std_logic;
14:
15:     q : out STD_LOGIC_VECTOR (31 downto 0);
16:     OE, IO, SLMF, PDMM, IE, RF, FF, FE, IA : out STD_LOGIC;
17:     FIL : out std_logic_vector (4 downto 0);
18:     DCR, BCR, DCR : out std_logic_vector (15 downto 0);
19:   );
20: end PM_regs;
21:
22: architecture Behavioral of PM_regs is
23:   type signal_array is array (3 downto 0) of std_logic_vector ( 31 downto 0 );
24:   signal array_reg: signal_array;
25:   signal decoded : std_logic_vector ( 3 downto 0 );
26:   signal cdr_en, bcr_en, dcr_en : std_logic;
27:   signal OE_i : SLMF_i, IE_i, RF_i, FF_i, FE_i, IA_i : std_logic;
28:   signal FIL_i : std_logic_vector (4 downto 0);
29: begin
30:   decoded <= decode( w_addr(3 downto 2), wen );
31:
32:   CSR : entity work.PM_CSR ( Behavioral )
33:   port map ( clk => clk, reset => reset, wen => decoded(0), d => d,
34:             FF_in => FF_i, FE_in => FE_i, IA_in => IA_i,
35:             q => array_reg(0), OE => OE_i, IO => IO, SLMF => SLMF_i, PDMM => PDMM, IE => IE,
36:             RF => RF_i, FF => FF_i, FE => FE_i, IA => IA_i, FIL => FIL_i );
37:
38:   cdr_en <= decoded(1) and not OE_i;
39:   DCR_reg: entity work.generic_register ( Behavioral )
40:   generic map ( bits => 16 )
41:   port map ( clk => clk, reset => reset, enable => cdr_en,
42:             d => d(15 downto 0), q => array_reg(1)(15 downto 0));
43:
44:   bcr_en <= decoded(2) and not OE_i;
45:   BCR_reg: entity work.generic_register ( Behavioral )
46:   generic map ( bits => 16 )
47:   port map ( clk => clk, reset => reset, enable => bcr_en,
48:             d => d(15 downto 0), q => array_reg(2)(15 downto 0));
49:
50:   DCR_reg: entity work.generic_register ( Behavioral )
51:   generic map ( bits => 16 )
52:   port map ( clk => clk, reset => reset, enable => dcr_en,
53:             d => d(15 downto 0), q => array_reg(3)(15 downto 0));
54:
55:   --DCR fifo
56:   dcr_fifo : entity work.generic_FIFO ( Behavioral )
57:   generic map ( bits => 16, depth => 32 )
58:   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 => SLMF_i, threshold => FIL_i,
59:             rdata => array_reg(3)(15 downto 0), empty => FE_i, full => FF_i, below_threshold => IA_i);
60:
61:   array_reg(1)(31 downto 16) <= ( others => '0' );
62:   array_reg(2)(31 downto 16) <= ( others => '0' );
63:   array_reg(3)(31 downto 16) <= ( others => '0' );
64:
65:   --outputs
66:   q <= array_reg ( to_integer( unsigned( r_addr (3 downto 2) ) ));
67:   DCR <= array_reg(1)(15 downto 0);
68:   BCR <= array_reg(2)(15 downto 0);
69:   DCR <= array_reg(3)(15 downto 0);
70:
71:   OE <= OE_i;
72:   SLMF <= SLMF_i;
73:   FIL <= FIL_i;
74:
75: end Behavioral;

```

PM_CSR

```

1: library IEEE;
2: use IEEE.STD_LOGIC_1164.ALL;
3: use IEEE.NUMERIC_STD.ALL;
4:
5:
6:
7: entity PM_CSR is
8:   Port ( clk, reset, wen : in std_logic;
9:         d : in STD_LOGIC_VECTOR (31 downto 0);
10:         FF_in, FE_in, IA_in : in STD_LOGIC;
11:         q : out std_logic_vector (31 downto 0);
12:         OE, IO, SLMF, PDMM, IE, RF, FF, FE, IA : out STD_LOGIC;
13:         FIL : out std_logic_vector (4 downto 0)
14:       );
15: end PM_CSR;
16:
17: architecture Behavioral of PM_CSR is
18:   signal master_enable : std_logic;
19:   signal q_oe : std_logic_vector (0 downto 0);
20:   signal q_rw : std_logic_vector (3 downto 0);
21:   signal q_fil : std_logic_vector (4 downto 0);
22:
23: begin
24:   master_enable <= wen and not q_oe(0);
25:   --output enable register
26:   oe_reg: entity work.generic_register ( Behavioral )
27:   generic map ( bits => 1 )
28:   port map ( clk => clk, reset => reset, enable => wen,
29:             d => d(0 downto 0), q => q_oe );
30:
31:   --regular read/write data register
32:   rw_reg: entity work.generic_register ( Behavioral )
33:   generic map ( bits => 4 )
34:   port map ( clk => clk, reset => reset, enable => master_enable,
35:             d => d(4 downto 1), q => q_rw );
36:
37:   --fifo interrupt level
38:   fil_reg: entity work.generic_register ( Behavioral )
39:   generic map ( bits => 5 )
40:   port map ( clk => clk, reset => reset, enable => master_enable,
41:             d => d(31 downto 27), q => q_fil );
42:
43:   --map outputs
44:   q <= q_fil & "0000000000000000" & IA_in & FE_in & FF_in & "000" & q_rw & q_oe;
45:   OE <= q_oe(0);
46:   IO <= q_rw(0);
47:   SLMF <= q_rw(1);
48:   PDMM <= q_rw(2);
49:   IE <= q_rw(3);
50:   RF <= master_enable and d(5);
51:   FIL <= q_fil;
52:   FF <= FF_in;
53:   FE <= FE_in;
54:   IA <= IA_in;
55:
56: end Behavioral;

```

Testbench VHDL

```

126: -----
127: -- turn on fifo mode and enable interrupts at level 8
128: APB_write(axi_aclck,x"00000000",x"00000000",apb,pready,pslverr,prdata);
129: APB_write(axi_aclck,x"00000000",x"00000004",apb,pready,pslverr,prdata);
130:
131:
132: wait for 40 ns;
133: -- go to 0% duty
134: APB_write(axi_aclck,x"00000000",x"00000000",apb,pready,pslverr,prdata);
135: -- go to 12.5% duty
136: APB_write(axi_aclck,x"00000000",x"00000001",apb,pready,pslverr,prdata);
137: -- go to 25% duty
138: APB_write(axi_aclck,x"00000000",x"00000002",apb,pready,pslverr,prdata);
139: -- go to 37.5% duty
140: APB_write(axi_aclck,x"00000000",x"00000003",apb,pready,pslverr,prdata);
141: -- go to 50% duty
142: APB_write(axi_aclck,x"00000000",x"00000004",apb,pready,pslverr,prdata);
143: -- go to 62.5% duty
144: APB_write(axi_aclck,x"00000000",x"00000005",apb,pready,pslverr,prdata);
145: -- go to 75% duty
146: APB_write(axi_aclck,x"00000000",x"00000006",apb,pready,pslverr,prdata);
147: -- go to 87.5% duty
148: APB_write(axi_aclck,x"00000000",x"00000007",apb,pready,pslverr,prdata);
149: -- go to 100% duty
150: APB_write(axi_aclck,x"00000000",x"00000008",apb,pready,pslverr,prdata);
151:
152: -- ramp back down
153: APB_write(axi_aclck,x"00000000",x"00000007",apb,pready,pslverr,prdata);
154: -- go to 87.5% duty
155: APB_write(axi_aclck,x"00000000",x"00000006",apb,pready,pslverr,prdata);
156: -- go to 75% duty
157: APB_write(axi_aclck,x"00000000",x"00000005",apb,pready,pslverr,prdata);
158: -- go to 62.5% duty
159: APB_write(axi_aclck,x"00000000",x"00000004",apb,pready,pslverr,prdata);
160: -- go to 50% duty
161: APB_write(axi_aclck,x"00000000",x"00000003",apb,pready,pslverr,prdata);
162: -- go to 37.5% duty
163: APB_write(axi_aclck,x"00000000",x"00000002",apb,pready,pslverr,prdata);
164: -- go to 25% duty
165: APB_write(axi_aclck,x"00000000",x"00000001",apb,pready,pslverr,prdata);
166: -- go to 12.5% duty
167: APB_write(axi_aclck,x"00000000",x"00000000",apb,pready,pslverr,prdata);
168: wait for 100 ns;
169: APB_write(axi_aclck,x"00000000",x"40000015",apb,pready,pslverr,prdata);
170:
171: --read from DCR
172: APB_read(axi_aclck,x"00000000",apb,pready,pslverr,prdata);
173:
174: -----
175: wait for 6000 ns;
176: --disable fifo, run
177: APB_write(axi_aclck,x"00000000",x"00000000",apb,pready,pslverr,prdata);
178: APB_write(axi_aclck,x"00000000",x"00000001",apb,pready,pslverr,prdata);
179: wait for 1000 ns;
180:

```

