Configuring CorePWM Using RTL Blocks

Introduction

This application note describes the configuration of CorePWM using custom RTL blocks. A design example is provided to illustrate how a simple finite state machine (FSM) can be used to control the pulse-width modulation (PWM) outputs of CorePWM.

The basic architecture of the design example is illustrated in Figure 1. The top-level design is *FSM_CorePWM*, which consists of *FSM* and *corepwm* instantiations. For both the top-level design and the custom FSM implementation, HDL code is provided at the end of this application note.

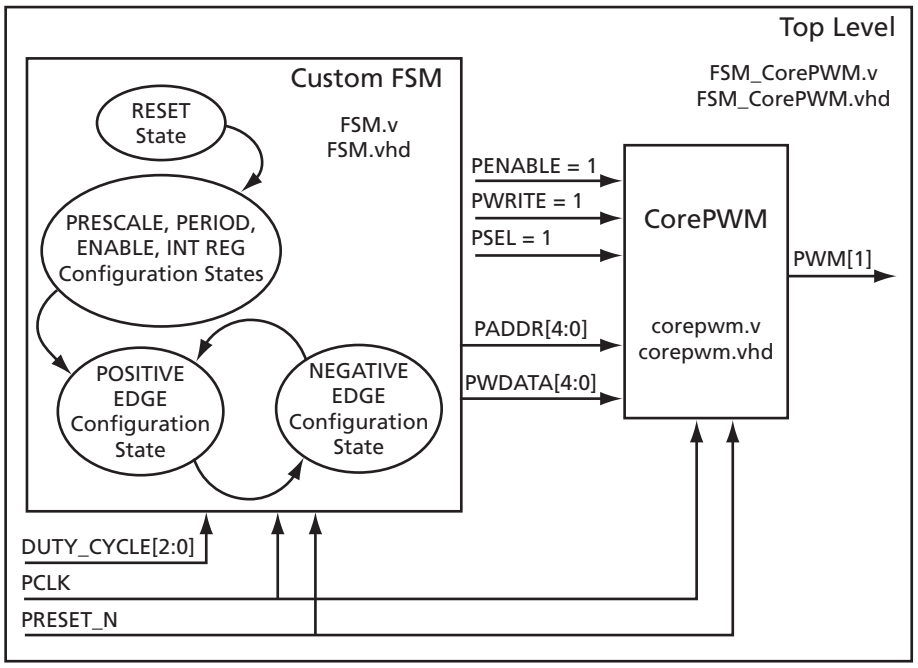

Figure 1 • **Example of Custom FSM Application Using CorePWM**

To run the design example, the Actel Libero® Integrated Design Environment (IDE) and CoreConsole software tools are required. To obtain the necessary RTL directory for the core, a CorePWM IP license used with CoreConsole is necessary as well.

CorePWM Overview

CorePWM is a PWM core that can be used in a number of embedded applications, including heating and cooling, motor control, voltage output adjustment, and sound generation.

CorePWM offers a low cost PWM solution with up to eight PWM output channels and 0–100% duty cycle capability. All PWM outputs are double-edge controlled and based on a configurable 8-bit PWM PERIOD value and an 8-bit PRESCALE value between PERIOD ticks.

CorePWM has an Advanced Peripheral Bus (APB) interface, which can be used to configure CorePWM with a microcontroller such as Core8051. For applications that do not require a microcontroller, CorePWM can be configured in write-only mode using a simple, low-tile-count state machine.

CorePWM is available in Actel Flash and antifuse FPGA devices. For more information, refer to the *CorePWM datasheet*.

General Description

As shown in Figure 2, CorePWM consists of the Register Interface, Timebase Generation, and PWM Waveform Generation blocks. A simple FSM can be used to control the inputs to the Register Interface block for PWM register configuration and updating.

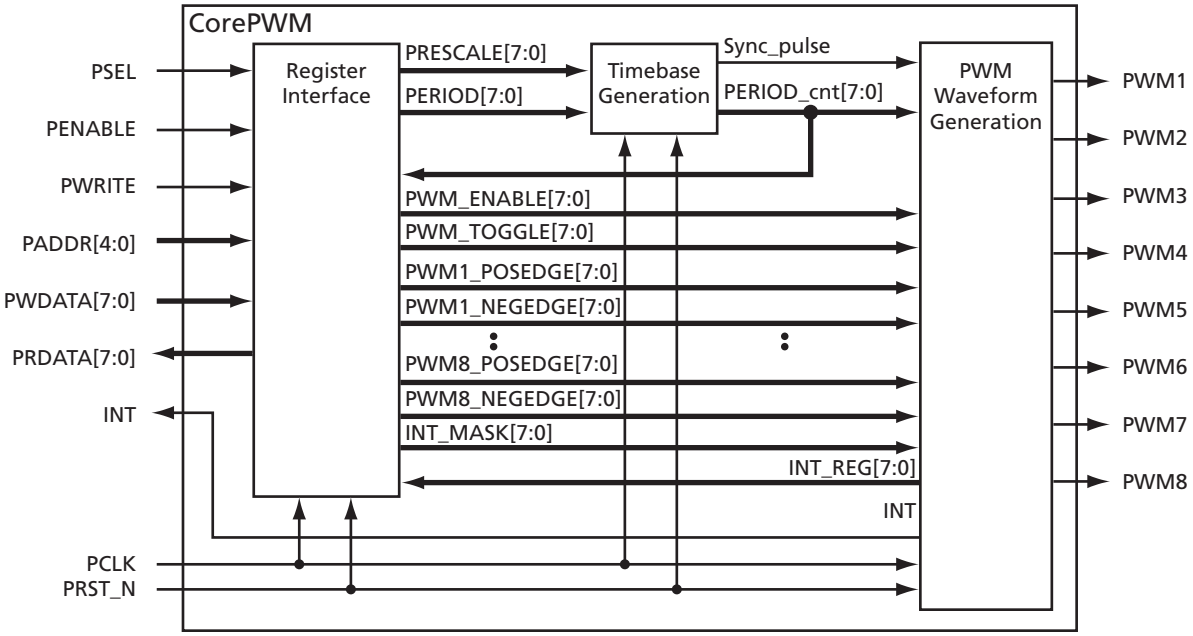

Figure 2 • **CorePWM Block Diagram**

The port signals for CorePWM, illustrated in Figure 2, are defined in Table 1.

Table 1 • **CorePWM I/O Signal Descriptions**

Signal Name	Description
PRESET_N	Active low asynchronous reset
PCLK	System clock; all operations and status are synchronous to rising edge.
PSEL	Select line for CorePWM
PENABLE	Read output enable
PWRITE	Write enable
PADDR[4:0]	Register address
PWDATA[7:0]	Write data input
PRDATA[7:0]	Read data output
INT	ORed interrupt signal; set to '1' at each PWM output transition.
PWM[PWM_NUM:1]	PWM output(s)—up to 8

The Timebase Generation block accepts PRESCALE and PERIOD register values and produces a period count (PERIOD_cnt) from 0 to 255. The number of system clock pulses between period counts is equal to the PRESCALE value.

Figure 3 shows an example of PWM waveform configuration. The example uses a PRESCALE register value of 1 and a PERIOD register value of 14.

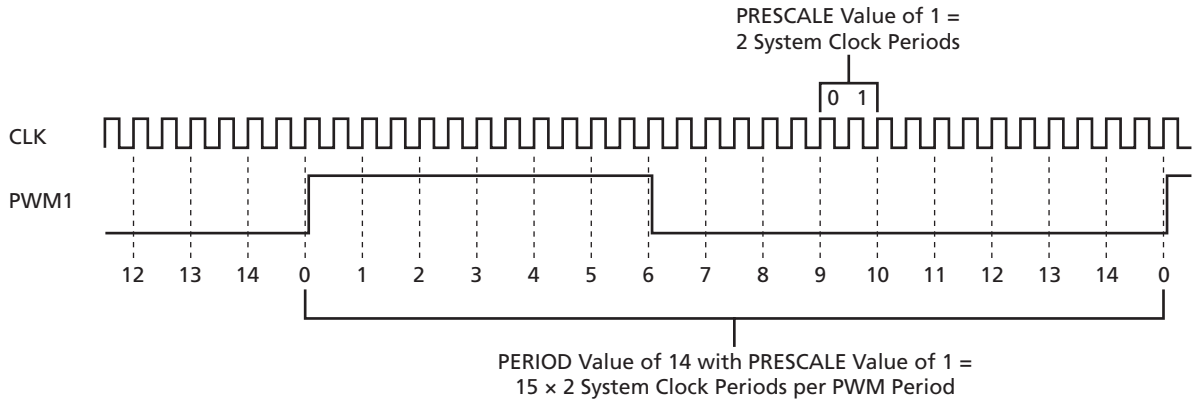

Figure 3 • **CorePWM Generation Example**

The PWM Waveform Generation block takes the input period count value and compares it with the positive- and negative-edge register values. When the count value is equal to any of these registers, the respective PWM output waveform is set to the correct value (high, low, or toggle), and the interrupt register is updated. With the use of PWM Waveform Generation block, PWM waveform updates occur only at the beginning of a PWM period, preventing erroneous pulse generation.

Duty Cycle Calculator

A Duty Cycle Calculator assists in calculating the PWM POSEDGE and NEGEDGE register values given a requested duty cycle. The calculator is provided on the Actel website as a Microsoft® Excel spreadsheet: http://www.actel.com/documents/duty_cycle_calc.zip.

CorePWM Configuration

A simple FSM can be used to configure and control the PWM outputs. The output of the FSM to CorePWM consists of the address lines (PADDR) and write data (PWDATA). The rest of the CorePWM input signals (PENABLE, PWRITE, and PSEL) are tied high, as CorePWM is configured in write-only mode.

Table 2 on page 4 gives descriptions and addresses for the CorePWM registers.

Table 2 • **CorePWM Register Definitions**

Register Name	Register Address PADDR[4:0]	Description	Default Value
PRESCALE	0h00	The system clock cycle is multiplied by the PRESCALE value, resulting in the minimum period count timebase.	0h08
PERIOD	0h01	The PRESCALE value is multiplied by the PERIOD value, yielding the PWM waveform cycle.	0h08
PWM_ENABLE	0h02	Logic '1' enables each PWM output.	0h00
INT_MASK	0h03	Logic '1' masks the respective bit in the INTERRUPT register.	0h00
INTERRUPT	0h04	Each interrupt bit is set to '1' at either edge of a PWM output.	0h00
PWM1_POSEDGE	0h05	Sets positive edge of each PWM1 output with respect to the PERIOD resolution.	0h00
PWM1_NEGEDGE	0h06	Sets negative edge of each PWM1 output with respect to the PERIOD resolution.	0h00
PWM2_POSEDGE	0h07	Sets positive edge of each PWM2 output with respect to the PERIOD resolution.	0h00
PWM2_NEGEDGE	0h08	Sets negative edge of each PWM2 output with respect to the PERIOD resolution.	0h00
PWM3_POSEDGE	0h09	Sets positive edge of each PWM3 output with respect to the PERIOD resolution.	0h00
PWM3_NEGEDGE	0h0A	Sets negative edge of each PWM3 output with respect to the PERIOD resolution.	0h00
PWM4_POSEDGE	0h0B	Sets positive edge of each PWM4 output with respect to the PERIOD resolution.	0h00
PWM4_NEGEDGE	0h0C	Sets negative edge of each PWM4 output with respect to the PERIOD resolution.	0h00
PWM5_POSEDGE	0h0D	Sets positive edge of each PWM5 output with respect to the PERIOD resolution.	0h00
PWM5_NEGEDGE	0h0E	Sets negative edge of each PWM5 output with respect to the PERIOD resolution.	0h00
PWM6_POSEDGE	0h0F	Sets positive edge of each PWM6 output with respect to the PERIOD resolution.	0h00
PWM6_NEGEDGE	0h10	Sets negative edge of each PWM6 output with respect to the PERIOD resolution.	0h00
PWM7_POSEDGE	0h11	Sets positive edge of each PWM7 output with respect to the PERIOD resolution.	0h00
PWM7_NEGEDGE	0h12	Sets negative edge of each PWM7 output with respect to the PERIOD resolution.	0h00
PWM8_POSEDGE	0h13	Sets positive edge of each PWM8 output with respect to the PERIOD resolution.	0h00
PWM8_NEGEDGE	0h14	Sets negative edge of each PWM8 output with respect to the PERIOD resolution.	0h00

Note: *0h = hexadecimal; 0b = binary.*

As an example used in this application note, the following parameters are set: PERIOD = 14, PRESCALE = 1, ENABLE = 1. A three-bit input value (DUTY_CYC) determines the duty cycle of the PWM[1] output. As predefined by the user, each DUTY_CYC value is associated with a specific duty cycle value. Based on this duty cycle value (Table 3), registers are set for the positive and negative edges of the PWM[1] output with respect to the PERIOD resolution. These register values can easily be determined using the Duty Cycle Calculator mentioned in the "Duty Cycle Calculator" section on page 3.

Table 3 • **Duty Cycle Example**

Desired Duty Cycle	DUTY_CYC Input	PWM1_POSEDGE Value	PWM1_NEGEDGE Value
0%	3'b000	15	0
20%	3'b001	0	3
40%	3'b010	0	6
60%	3'b011	0	9
80%	3'b100	0	12
100%	3'b101	0	15

HDL Implementation of Design Example

In the *FSM* module, the PERIOD, PRESCALE, and ENABLE registers are configured first (Figure 1 on page 1). This is followed by writing to the PWM[1] POSEDGE and NEGEDGE registers with the data specified in Table 3.

The CorePWM subsystem (*corepwm.v/corepwm.vhd*) is generated using CoreConsole, a design entry tool that enables IP blocks. CoreConsole exports synthesizable and simulatable RTL for CorePWM into Libero IDE.

The designer then needs to create RTL code for the FSM and connect it to the CoreConsole-created CorePWM subsystem. In addition to the FSM RTL code, the top-level design (*FSM_CorePWM* HDL file) is included with this application note in "Appendix A – Verilog" on page 7 and "Appendix B – VHDL" on page 10.

The design example has been verified with simulation, as shown in Figure 4.

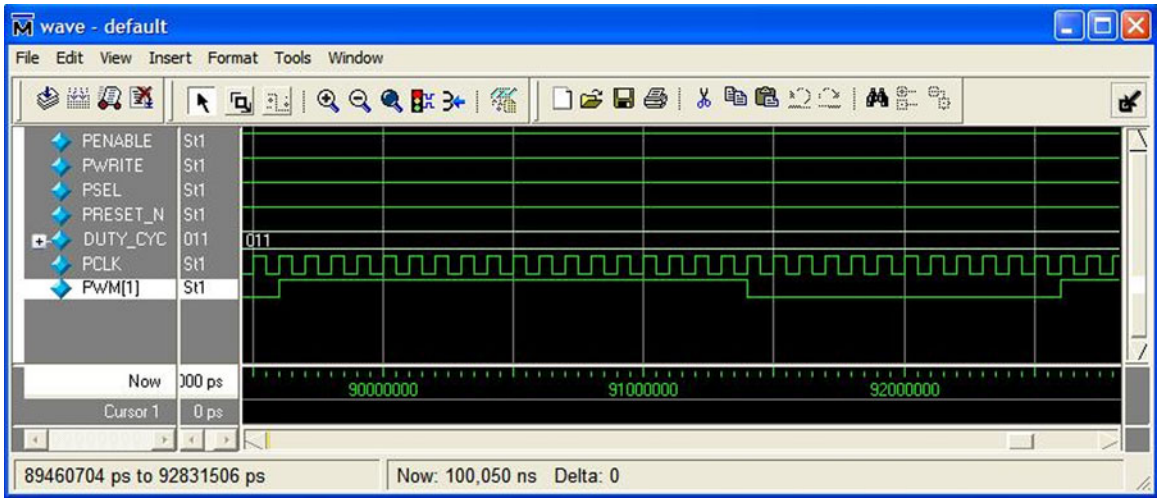

Figure 4 • **Simulation Waveform for Design Example**

Since there is only one PWM output used in the example, the PWM_NUM parameter is set to '1'. Furthermore, the FIXED_REG_SEL parameter/generic is set to '0' because the CorePWM registers will need to interface to the FSM.

Note that since this application employs the CorePWM write-only mode, the Register Interface's read data bus (PRDATA) and the interrupt line (INT), which are CorePWM outputs, are not used and can be commented out of the *corepwm* module.

Conclusion

CorePWM is a general purpose pulse-width modulator that can be used in many different applications. The CorePWM digital outputs can be configured and controlled with the use of a simple FSM. CorePWM has been implemented in all of the most popular Flash and antifuse Actel FPGA devices.

List of Changes

The following table lists critical changes that were made in the current version of the document.

Previous Version	Changes in Current Version (51900141-1/9.06*)	Page
51900141-0/6.06	Parameter "PWM_NUM" was updated in the RTL. Refer to "Appendix A – Verilog" and "Appendix B – VHDL" to see the updated code.	7 and 10

Note: **The part number is located on the last page of the document.*

Appendix A – Verilog

Top-Level Design (integration of FSM with CorePWM): *FSM_CorePWM.v*

```
module FSM_CorePWM #(
parameter PWM_NUM = 1)
(PCLK, PRESET_N, PSEL, PENABLE, PWRITE, DUTY_CYC, PWM, PRDATA, INT);
   input PCLK, PRESET_N, PSEL, PENABLE, PWRITE;
   input [2:0] DUTY_CYC;
   output [PWM_NUM:1] PWM;
   output [7:0] PRDATA;
   output INT;
   wire [4:0] PADDR_TOP;
   wire [7:0] PWDATA_TOP;

   FSM FSM_TOP (.PCLK(PCLK),.PRESET_N(PRESET_N),.DUTY_CYC(DUTY_CYC),
      .PADDR(PADDR_TOP[4:0]),.PWDATA(PWDATA_TOP[7:0]));
   corepwm #(.PWM_NUM(PWM_NUM))
   COREPWM_TOP
(.PCLK(PCLK),.PRESET_N(PRESET_N),.PSEL(PSEL),.PENABLE(PENABLE),.PWRITE(PWRITE),

.PADDR(PADDR_TOP[4:0]),.PWDATA(PWDATA_TOP[7:0]),.PWM(PWM),.PRDATA(PRDATA[7:0]),.INT(IN
T));
endmodule
```

FSM: *FSM.v*

```
module FSM (DUTY_CYC, PRESET_N, PCLK, PADDR, PWDATA);
   input PCLK, PRESET_N;
   input [2:0] DUTY_CYC;
   output [4:0] PADDR;
   output [7:0] PWDATA;
   reg [4:0] PADDR;
   reg [7:0] PWDATA;
   parameter /* Configuration States of FSM */
      S_RESET = 3'b000,
      S_CONFIG_PRE = 3'b001,
      S_CONFIG_PER = 3'b010,
      S_CONFIG_EN = 3'b011,
      S_CONFIG_MASK = 3'b100,
      S_RUN_POS = 3'b101,
      S_RUN_NEG = 3'b110;
   reg [2:0] STATE;
   parameter PRESCALE = 8'b0001;        /* PRESCALE=1 */
   parameter PERIOD = 8'b1110;          /* PERIOD=14 */
```

```
parameter PWM_ENABLE = 8'b0001;
parameter INT_MASK = 8'b0000;

always @(posedge PCLK)
begin: FSM_CORE
   if (!PRESET_N)
      STATE = S_RESET;
   case (STATE)
      S_RESET:
         if (PRESET_N)
            STATE = S_CONFIG_PRE;
      S_CONFIG_PRE:    /*PRESCALE Register Configuration*/
         begin
            PADDR = 5'b000;
            PWDATA = PRESCALE;
            STATE = S_CONFIG_PER;
         end
      S_CONFIG_PER:    /*PERIOD Register Configuration*/
         begin
            PADDR = 5'b001;
            PWDATA = PERIOD;
            STATE = S_CONFIG_EN;
         end
      S_CONFIG_EN:     /*CorePWM Enable Reg Configuration*/
         begin
            PADDR = 5'b010;
            PWDATA = PWM_ENABLE;
            STATE = S_CONFIG_MASK;
         end
      S_CONFIG_MASK:   /*INT Register Configuration*/
         begin
            PADDR = 5'b011;
            PWDATA = INT_MASK;
            STATE = S_RUN_POS;
         end
      S_RUN_POS:       /*Positive Edge Register Configuration*/
         begin
            PADDR = 5'b101;
            case (DUTY_CYC)
            /*For Register Values based on Duty Cycle Input, refer to
               Table 3 on page 5*/
               3'b000: PWDATA = 8'b1111;
               3'b001: PWDATA = 8'b0000;
```

```
                3'b010: PWDATA = 8'b0000;
                3'b011: PWDATA = 8'b0000;
                3'b100: PWDATA = 8'b0000;
                3'b101: PWDATA = 8'b0000;
             endcase
             STATE = S_RUN_NEG;
          end
       S_RUN_NEG:       /*Negative Edge Register Configuration*/
          begin
             PADDR = 5'b110;
             case (DUTY_CYC)
             /*For Register Values based on Duty Cycle Input, refer to
                Table 3 on page 5*/
                3'b000: PWDATA = 8'b0000;
                3'b001: PWDATA = 8'b0011;
                3'b010: PWDATA = 8'b0110;
                3'b011: PWDATA = 8'b1001;
                3'b100: PWDATA = 8'b1100;
                3'b101: PWDATA = 8'b1111;
             endcase
             STATE = S_RUN_POS;
          end
      endcase
   end
endmodule
```

Appendix B – VHDL

Top-Level Design (integration of FSM with CorePWM): *FSM_CorePWM.vhd*

```
library ieee;
use ieee.std_logic_1164.all;

entity FSM_CorePWM is
    GENERIC (PWM_NUM :  integer := 1);
   port (PCLK, PRESET_N, PSEL, PENABLE, PWRITE: in std_logic;
           DUTY_CYC: in std_logic_vector (2 downto 0);
           PWM: out std_logic_vector (PWM_NUM downto 1);
           PRDATA: out std_logic_vector (7 downto 0);
           INT: out std_logic);
end entity FSM_CorePWM;

architecture HIERARCHICAL of FSM_CorePWM is
   component FSM
      port (PCLK, PRESET_N: in std_logic;
         DUTY_CYC: in std_logic_vector (2 downto 0);
         PADDR: out std_logic_vector (4 downto 0);
         PWDATA: out std_logic_vector (7 downto 0));
   end component;

   component corepwm
      GENERIC (PWM_NUM :  integer := 8);
         port (PCLK, PRESET_N, PSEL, PENABLE, PWRITE: in std_logic;
               PADDR: in std_logic_vector(4 downto 0);
               PWDATA: in std_logic_vector(7 downto 0);
               PWM: out std_logic_vector(PWM_NUM downto 1);
               PRDATA: out std_logic_vector(7 downto 0);
               INT: out std_logic);

   signal PADDR_TOP: std_logic_vector(4 downto 0);
   signal PWDATA_TOP: std_logic_vector(7 downto 0);

begin
   FSM_TOP: FSM port map (PCLK=>PCLK, PRESET_N=>PRESET_N, DUTY_CYC=>DUTY_CYC,
      PADDR=>PADDR_TOP, PWDATA=>PWDATA_TOP);
   COREPWM_TOP: corepwm
   GENERIC MAP (PWM_NUM =>PWM_NUM)
   port map (PCLK=>PCLK,PRESET_N=>PRESET_N,PSEL=>PSEL,PENABLE=>PENABLE,PWRITE=>PWRITE,
PADDR=>PADDR_TOP,PWDATA=>PWDATA_TOP,PWM=>PWM,PRDATA=>PRDATA,INT=>INT);
end HIERARCHICAL;
```

FSM: *FSM.vhd*

```
library IEEE;
use IEEE.STD_Logic_1164.all;

entity FSM is
   port (PCLK, PRESET_N: in std_logic;
      DUTY_CYC: in std_logic_vector (2 downto 0);
      PADDR: out std_logic_vector (4 downto 0);
      PWDATA: out std_logic_vector (7 downto 0));
end entity FSM;

architecture RTL of FSM is

   --Configuration States of FSM
   type CurrentState is (S_RESET, S_CONFIG_PRE, S_CONFIG_PER, S_CONFIG_EN,
      S_CONFIG_MASK, S_RUN_POS, S_RUN_NEG);
   signal STATE: CurrentState;

   --PRESCALE = 1
   constant PRESCALE: std_logic_vector(7 downto 0):="00000001";

   --PERIOD = 14
   constant PERIOD: std_logic_vector(7 downto 0):="00001110";

   constant PWM_ENABLE: std_logic_vector(7 downto 0):="00000001";
   constant INT_MASK: std_logic_vector(7 downto 0):="00000000";

begin
   FSM_CORE: process (PCLK)
   begin
      if rising_edge (PCLK) then
         if (PRESET_N='0') then
            STATE <= S_RESET;
         end if;

         case (STATE) is
            when S_RESET => if (PRESET_N='1') then
                  STATE <= S_CONFIG_PRE;
               end if;
```

```
--PRESCALE Register Configuration
when S_CONFIG_PRE => PADDR <= "00000";
   PWDATA <= PRESCALE;
   STATE <= S_CONFIG_PER;

--PERIOD Register Configuration
when S_CONFIG_PER => PADDR <= "00001";
   PWDATA <= PERIOD;
   STATE <= S_CONFIG_EN;

--CorePWM Enable Reg Configuration
when S_CONFIG_EN => PADDR <= "00010";
   PWDATA <= PWM_ENABLE;
   STATE <= S_CONFIG_MASK;

--INT Register Configuration
when S_CONFIG_MASK => PADDR <= "00011";
   PWDATA <= INT_MASK;
   STATE <= S_RUN_POS;

--Positive Edge Register Configuration
when S_RUN_POS => PADDR <= "00101";
   case DUTY_CYC is
   --For Register Values based on Duty Cycle Input, refer to
      Table 3 on page 5
      when "000"=>PWDATA(3 downto 0)<="1111";
      when "001"=>PWDATA(3 downto 0)<="0000";
      when "010"=>PWDATA(3 downto 0)<="0000";
      when "011"=>PWDATA(3 downto 0)<="0000";
      when "100"=>PWDATA(3 downto 0)<="0000";
      when "101"=>PWDATA(3 downto 0)<="0000";
      when others=>PWDATA(3 downto 0)<="0000";
   end case;
   STATE <= S_RUN_NEG;

--Negative Edge Register Configuration
when S_RUN_NEG=>PADDR<="00110";
   case (DUTY_CYC) is
   --For Register Values based on Duty Cycle Input, refer to
      Table 3 on page 5
      when "000"=>PWDATA(3 downto 0)<="0000";
      when "001"=>PWDATA(3 downto 0)<="0011";
      when "010"=>PWDATA(3 downto 0)<="0110";
```

```
                when "011"=>PWDATA(3 downto 0)<="1001";
                when "100"=>PWDATA(3 downto 0)<="1100";
                when "101"=>PWDATA(3 downto 0)<="1111";
                when others=>PWDATA(3 downto 0)<="0000";
            end case;
            STATE <= S_RUN_POS;
        end case;
    end if;
  end process;
end RTL;
```

www.actel.com

Actel Corporation

2061 Stierlin Court
Mountain View, CA
94043-4655 USA
Phone 650.318.4200
Fax 650.318.4600

Actel Europe Ltd.

Dunlop House, Riverside Way
Camberley, Surrey GU15 3YL
United Kingdom
Phone +44 (0) 1276 401 450
Fax +44 (0) 1276 401 490

Actel Japan
www.jp.actel.com

EXOS Ebisu Bldg. 4F
1-24-14 Ebisu Shibuya-ku
Tokyo 150 Japan
Phone +81.03.3445.7671
Fax +81.03.3445.7668

Actel Hong Kong
www.actel.com.cn

Suite 2114, Two Pacific Place
88 Queensway, Admiralty
Hong Kong
Phone +852 2185 6460
Fax +852 2185 6488

51900141-1/9.06