

## Pulse Width Modulation

Pulse Width Modulation (PWM) is a technique to provide a logic “1” and logic “0” for a controlled period of time. Pulse Width Modulation is used in many applications such as controlling the speed of a DC motor.

This application note describes the implementation of simple PWM using Atmel’s FPGAs. The basic principle is to use a register to store the value which is loaded on to the Up/Down Counter whenever the counter reaches its terminal count. The terminal counter is used to generate the pulse width modulation.

### Functional Description

A data register is used to store the value for the counter, this value determines the pulse width. The Up/Down Counter is loaded with a new value from the data register when the counter reaches its terminal count; a Toggle Flip-flop generates the PWM output.

When the data value is first loaded, the counter begins to count down from the data value to 0. During this phase of operation the terminal count and PWM signals are Low. When the counter transitions through 0, the terminal count is generated and it triggers the Toggle Flip-flop to drive the PWM signal High. The data value is re-loaded and counting proceeds up to the maximum value. Again a terminal count will be generated when the counter reaches its maximum value, driving the PWM signal to toggle from High to Low. The data value is re-loaded and the cycle repeats. The direction of the counter is controlled by the PWM signal: the counter is set to count down when the PWM is Low, and count up when the PWM is High. The terminal count controls the data value that loads to the counter from the data register. Data is loaded when the terminal count is High.

The duty cycle of the PWM signal is controlled by the data value loaded to the up/down counter. The duty cycle of the PWM output can be varied by specifying various data values, the higher the data value, higher the duty cycle (see Table 1).

**Table 1.** Data Values for Different Duty Cycles

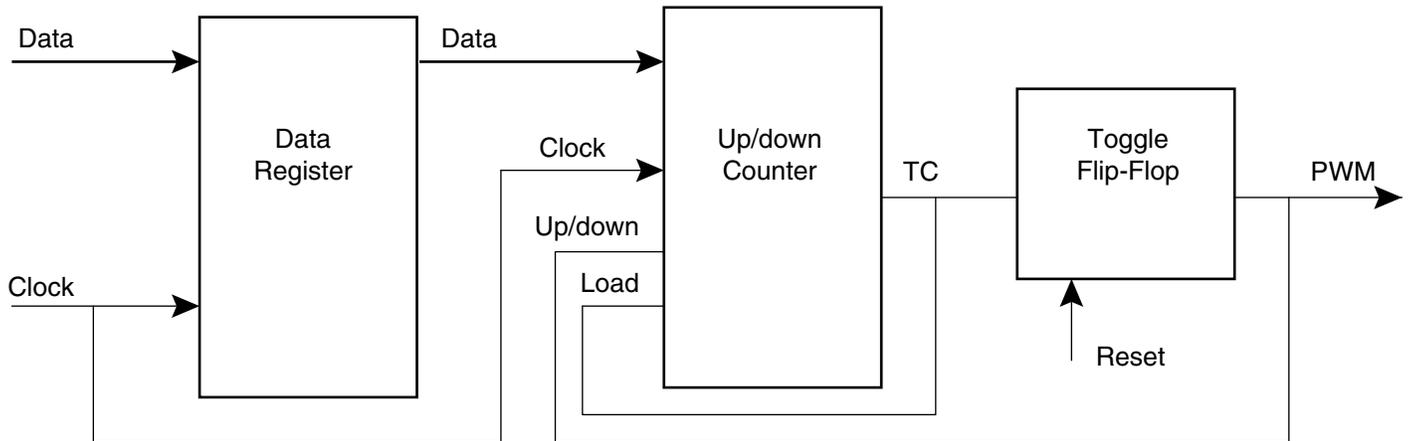
Data Value	Duty Cycle (%)
11100110	90
11000000	75
10000000	50
01000000	25
00011001	10

**Programmable  
SLI  
AT40K  
AT40KAL  
AT94K**

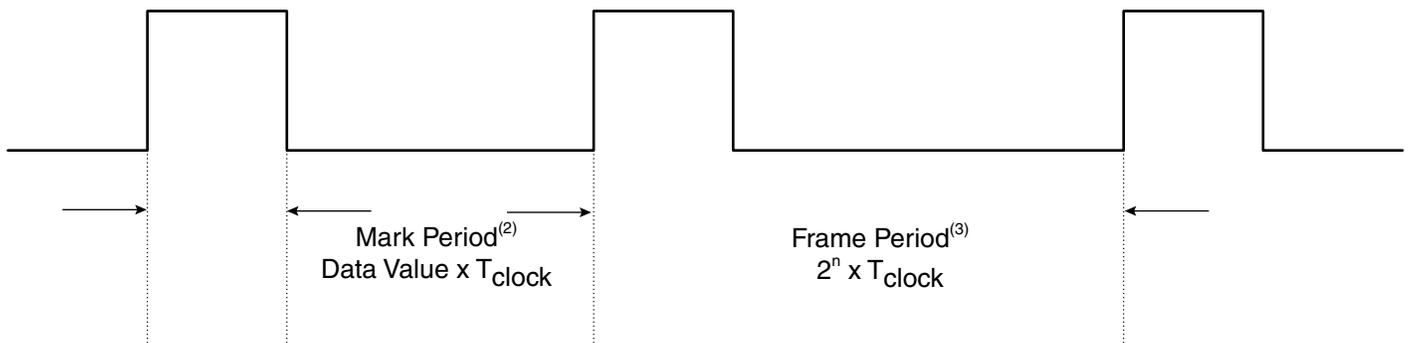
**Application  
Note**



## Block Diagram



**Figure 1.** Sample PWM Output Waveform<sup>(1)</sup>



- Notes:
1. Duty Cycle is calculated by taking the ratio of Mark Period and Frame Period:  

$$\text{Duty Cycle} = \text{Mark Period} / \text{Frame Period} = \text{Data Value} / 2^n.$$
  2. Mark Period = Data Value  $\times T_{\text{clock}}$ .
  3. Frame Period =  $T_{\text{clock}} \times 2^n$ , where "n" is the binary counter width.

## Design Implementation

An 8-bit PWM counter is implemented using VHDL. The design uses an 8-bit data register, an 8-bit up/down counter and comparator logic. The designer can easily modify the design for a different resolution by changing the data register and the counter width. The VHDL source files, testbench files and the user packages can be downloaded from Atmel's web site. Table 2 provides a list of the software requirements for the design implementation.

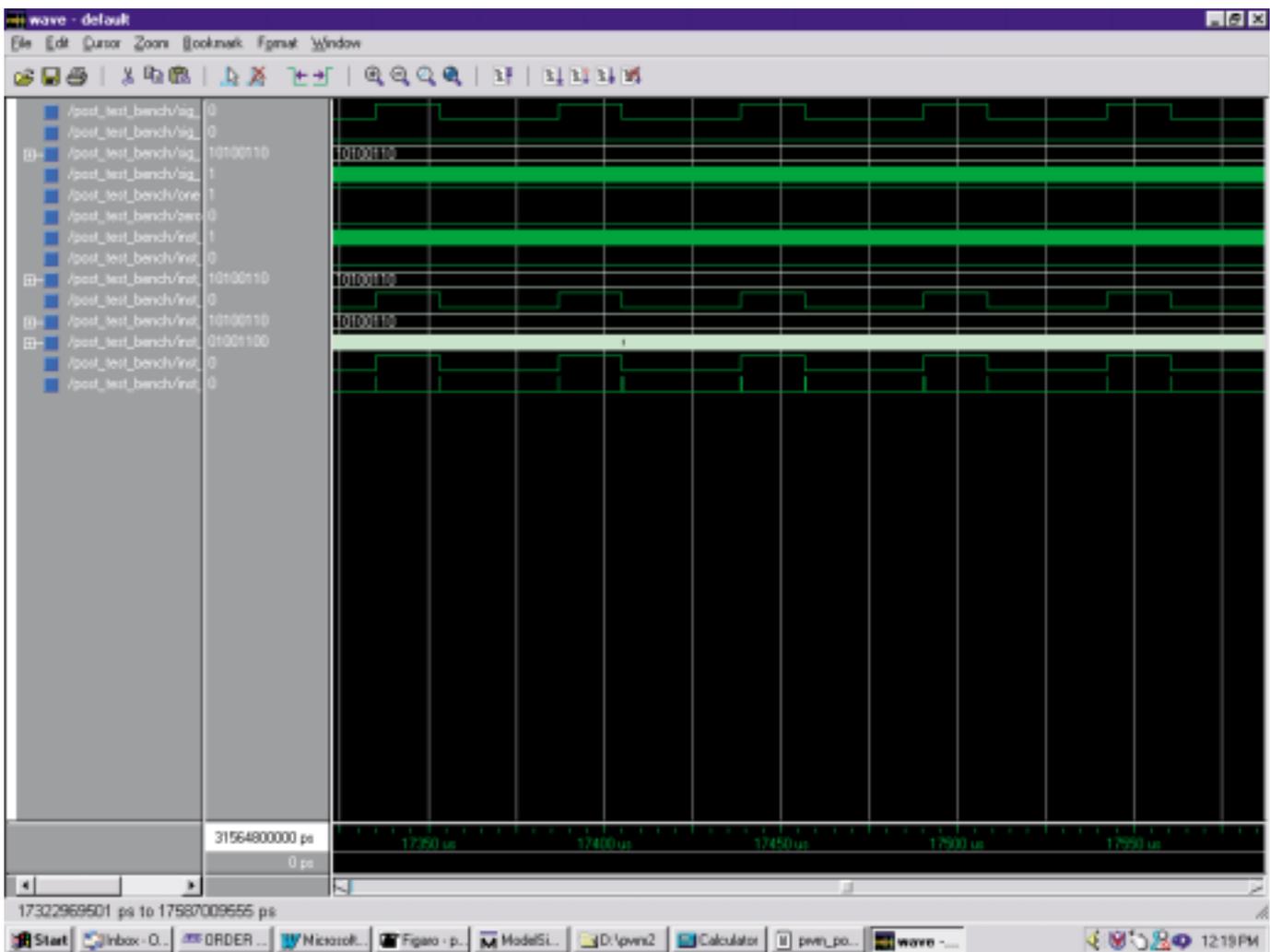
**Table 2.** Software Requirements

Tool	Requirement
VHDL Synthesizer	Exemplar's LeonardoSpectrum™ or any Synthesis tool which supports the Atmel AT40K architecture
Place & Route	Atmel's IDS 6.0 and above or System Designer™
Simulator	ModelSim® simulator or any simulator tool which supports VITAL VHDL

## Sample Design Using LeonardoSpectrum, IDS Figaro and ModelSim Simulator

1. Copy the source file pwm\_fpga.vhd to your project or design directory. This file can be downloaded from Atmel's web site, at <http://www.atmel.com/atmel/products/prod102.htm>
2. Start LeonardoSpectrum.
3. Select AT40K under the Technology window and click on open files to select pwm\_fpga.vhd from your project directory.
4. Click on run for Leonardo to read the design file, map to Atmel architecture and synthesize.
5. The successful compilation will generate the pwm\_fpga.edf.
6. Import pwm\_fpga to the IDS Figaro for placement and routing.
7. Generate the bistream and use this bistream to configure the FPSLIC™. The resulting PWM signal is shown in Figure 2.

Figure 2. ModelSim Simulator Result for 65% Duty Cycle PWM Wave





## Source Files

### pwm\_fpga.vhd

```
Library IEEE;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all ;
USE work.user_pkg.all;

ENTITY pwm_fpga IS
PORT ( clock,reset: in STD_LOGIC;
      Data_value : in std_logic_vector(7 downto 0);
      pwm       : out STD_LOGIC
      );
END pwm_fpga;

ARCHITECTURE arch_pwm OF pwm_fpga IS

SIGNAL reg_out : std_logic_vector(7 downto 0);
SIGNAL cnt_out_int: std_logic_vector(7 downto 0);
SIGNAL pwm_int, rco_int: std_logic;

BEGIN

-- 8-bit data register to store the data values .The data values
-- will determine the duty cycle of PWM output

PROCESS (clock,reg_out,reset)

BEGIN
IF (reset = '1') THEN
reg_out <="00000000";
ELSIF (rising_edge(clock)) THEN
reg_out <= data_value;
END IF;
END PROCESS;

--8-bit up/down counter. Counts up or down based on the pwm_int signal
--and generates terminal count whenever counter reaches the
--maximum value or when it transists through zero. Terminal
--count is used to automatically load the data value to generate
--different pwm out with different duty cycle

--INC and DEC are the two functions which are used for up and
--down counting. they are defined in sepearate user_pakge library

PROCESS (clock,cnt_out_int,rco_int,reg_out)

BEGIN

IF (rco_int = '1') THEN
cnt_out_int <= reg_out;
```

```
ELSIF rising_edge(clock) THEN
    IF (rco_int = '0' and pwm_int = '1' and cnt_out_int < "11111111") THEN
        cnt_out_int <= INC(cnt_out_int);
    ELSE
        IF (rco_int = '0' and pwm_int = '0' and cnt_out_int > "00000000") THEN
            cnt_out_int <= DEC(cnt_out_int);
        END IF;
    END IF;
END IF;
END PROCESS;

PROCESS(cnt_out_int, rco_int, clock, reset)
BEGIN

    IF (reset = '1') THEN
        rco_int <= '1';
    ELSIF rising_edge(clock) THEN
        IF ((cnt_out_int = "11111111") or (cnt_out_int = "00000000")) THEN
            rco_int <= '1';
        ELSE
            rco_int <= '0';
        END IF;
    END IF;

END PROCESS;

-- Logic to Generate the PWM output.

PROCESS (clock, rco_int, reset)
BEGIN
    IF (reset = '1') THEN
        pwm_int <= '0';
    ELSIF rising_edge(rco_int) THEN
        pwm_int <= NOT(pwm_int);
    ELSE
        pwm_int <= pwm_int;
    END IF;
END PROCESS;
pwm <= pwm_int;

END arch_pwm;
```

## User\_pkg\_inc\_dec.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

PACKAGE user_pkg IS
    function INC(X: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;
    function DEC(X: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;
END user_pkg ;

PACKAGE BODY user_pkg IS

function INC(X: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR is
    variable XV: STD_LOGIC_VECTOR(X'LENGTH - 1 downto 0);
    begin
        XV := X;
        for I in 0 to XV'HIGH LOOP
            if XV(I) = '0' then
                XV(I) := '1';
                exit;
            else XV(I) := '0';
            end if;
        end loop;

    return XV;
end INC;

function DEC(X: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR is
    variable XV: STD_LOGIC_VECTOR(X'LENGTH - 1 downto 0);
    begin
        XV := X;
        for I in 0 to XV'HIGH LOOP
            if XV(I) = '1' then
                XV(I) := '0';
                exit;
            else XV(I) := '1';
            end if;
        end loop;

    return XV;
end DEC;

END user_pkg;
```

## Post-layout Testbench

### pwm\_posttb.vhd

#### Post-layout Testbench File

- Design: pwm\_fpga
- Program: Figaro
- Version: Atmel 7.2 (patch level 3 applied)
- Vendor: Atmel
- Created: May 31, 2001 at : 11:45:56 am

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

use IEEE.VITAL_timing.all;

library AT40K;
use AT40K.VCOMPONENTS.all;

entity post_test_bench is
end post_test_bench;

architecture arch_test_bench of post_test_bench is

component pwm_fpga
  port (
    clock : in STD_LOGIC := '0';
    reset : in STD_LOGIC := '0';
    Data_value : in STD_LOGIC_VECTOR(7 downto 0) := "00000000";
    pwm : out STD_LOGIC
  );
end component;

signal sig_pwm: STD_LOGIC;
signal sig_reset: STD_LOGIC;
signal sig_Data_value: STD_LOGIC_VECTOR(7 downto 0);
signal sig_clock: STD_LOGIC;
signal one : STD_LOGIC := '1';
signal zero : STD_LOGIC := '0';
shared variable ENDSIM: boolean:=false;
constant clk_period:TIME:=200 ns;

BEGIN

clk_gen: process
begin
  if ENDSIM = FALSE THEN
    sig_clock <= '1';
    wait for clk_period/2;
    sig_clock <= '0';
```

```
        wait for clk_period/2;
    else
        wait;
    end if;
end process;

inst_pwm_fpga:pwm_fpga
port map (
clock => sig_clock,
reset => sig_reset,
Data_value => sig_Data_value,
pwm => sig_pwm
);

stimulus_process: PROCESS

BEGIN

    sig_reset <= '1';
    wait for 100 ns;
    sig_reset <= '0';
    sig_data_value <= "11000000";
    wait for 50 us;
    sig_data_value <= "10000000";
    wait for 50 us;
    sig_data_value <= "01000000";
    wait for 50 us;
    sig_data_value <= "10000000";

wait;

END PROCESS stimulus_process;

end arch_test_bench;
```



## Atmel Headquarters

**Corporate Headquarters**  
2325 Orchard Parkway  
San Jose, CA 95131  
TEL (408) 441-0311  
FAX (408) 487-2600

### Europe

Atmel SarL  
Route des Arsenaux 41  
Casa Postale 80  
CH-1705 Fribourg  
Switzerland  
TEL (41) 26-426-5555  
FAX (41) 26-426-5500

### Asia

Atmel Asia, Ltd.  
Room 1219  
Chinachem Golden Plaza  
77 Mody Road Tsimhatsui  
East Kowloon  
Hong Kong  
TEL (852) 2721-9778  
FAX (852) 2722-1369

### Japan

Atmel Japan K.K.  
9F, Tonetsu Shinkawa Bldg.  
1-24-8 Shinkawa  
Chuo-ku, Tokyo 104-0033  
Japan  
TEL (81) 3-3523-3551  
FAX (81) 3-3523-7581

## Atmel Product Operations

### Atmel Colorado Springs

1150 E. Cheyenne Mtn. Blvd.  
Colorado Springs, CO 80906  
TEL (719) 576-3300  
FAX (719) 540-1759

### Atmel Grenoble

Avenue de Rochepleine  
BP 123  
38521 Saint-Egreve Cedex, France  
TEL (33) 4-7658-3000  
FAX (33) 4-7658-3480

### Atmel Heilbronn

Theresienstrasse 2  
POB 3535  
D-74025 Heilbronn, Germany  
TEL (49) 71 31 67 25 94  
FAX (49) 71 31 67 24 23

### Atmel Nantes

La Chantrerie  
BP 70602  
44306 Nantes Cedex 3, France  
TEL (33) 0 2 40 18 18 18  
FAX (33) 0 2 40 18 19 60

### Atmel Rousset

Zone Industrielle  
13106 Rousset Cedex, France  
TEL (33) 4-4253-6000  
FAX (33) 4-4253-6001

### Atmel Smart Card ICs

Scottish Enterprise Technology Park  
East Kilbride, Scotland G75 0QR  
TEL (44) 1355-357-000  
FAX (44) 1355-242-743

---

### Atmel Programmable SLI Hotline

(408) 436-4119

### Atmel Programmable SLI e-mail

fpga@atmel.com

### FAQ

Available on web site

### e-mail

literature@atmel.com

### Web Site

<http://www.atmel.com>

### BBS

1-(408) 436-4309

### © Atmel Corporation 2001.

Atmel Corporation makes no warranty for the use of its products, other than those expressly contained in the Company's standard warranty which is detailed in Atmel's Terms and Conditions located on the Company's web site. The Company assumes no responsibility for any errors which may appear in this document, reserves the right to change devices or specifications detailed herein at any time without notice, and does not make any commitment to update the information contained herein. No licenses to patents or other intellectual property of Atmel are granted by the Company in connection with the sale of Atmel products, expressly or by implication. Atmel's products are not authorized for use as critical components in life support devices or systems.

Atmel® is the registered trademark of Atmel. System Designer™ and FPSLIC™ are the trademarks of Atmel.

Leonardo® and ModelSim® are the registered trademarks of Mentor Graphics Corporation. LeonardoSpectrum™ is the trademark of Mentor Graphics Corporation. Other terms and product names may be the trademarks of others.



Printed on recycled paper.