

April 11, 1995

Application Brief BY GARY LAWMAN

Summary

This Application Brief demonstrates how to build a variable Pulse-Width Modulation (PWM) waveform using a counter and a storage register. PWM is used in such areas as DC motor drive control and digital-to-analog conversion in bit stream DACs. In many of these applications, the single bit digital output is subject to a low-pass filter which results in an analog output level. The output level is the analog equivalent of the digital PWM's duty-cycle.

Xilinx Family

All

Demonstrates

X-BLOX for arbitrary bus sizing

Novel use of a counter's terminal count

Introduction

Pulse-Width Modulation (PWM) is used in many diverse areas such as controlling the speed of DC motors, and the analog translation of digital audio using the 1-bit DAC of a CD player. The design presented in this brief uses a register to store the desired 'mark' value, which is automatically loaded into a down counter upon reaching its terminal count. The PWM Frame Period is the product of the counter's clock period and the terminal count value, being the sum of the 'mark' and 'space' periods. The design is readily scaled up or down simply by changing the width of the register and counter.

Functional Description

The example design is made up of three separate components. The data register, MARK_REGISTER, is used to store the counter value. This value determines the period that a Mark is generated, or the Pulse Width. The up/down counter, called FRAME_COUNTER is loaded with the value stored in the MARK_REGISTER whenever the counter reaches its terminal count, TC.

During initial operation, the counter loads the Mark_Value and counting proceeds downwards, so that a 'One', (Mark), is seen on the output during this period. See Figure 1. When the counter transitions through Zero, the counter reaches its terminal count, TC. The terminal count then changes the direction of counting to upwards on the next clock. At the same time, the high TC signal forces the Mark_Value to be re-loaded, so that now the counter counts from Mark_Value up to TC. During this time, a 'Zero', (Space), is driven by the toggle flip-flop. Finally, the entire cycle repeats once TC is reached.

In the design example, an n -bit binary counter is used. The value of n is set to eight bits wide using the attribute 'BOUNDS=[7:0]' associated with the Mark_Value INPUTS symbol. X-BLOX propagates the BOUNDS attribute throughout the design, so that the MARK_REGISTER and the FRAME_COUNTER are

also sized automatically to 8 bits. If this attribute were instead set to 'BOUNDS=[31:0]', this design would generate a 32-bit data register and 32-bit counter.

Other types of counters could be used, but care should be taken to ensure that only valid data is written to the MARK_REGISTER. The following calculations are valid for a binary counter, and should be modified accordingly if other counter types are used. For example, a Johnson counter's TC is $2n$, and thus the Mark_Value in this case must be less than $2n$.

The Terminal Count signal, TC, drives a toggle flip-flop which then produces the PWM signal at its output. This output is modulated according to the size of the counter and the value stored in the MARK_REGISTER.

Unless a Reset is added to the Counter, a new Mark_Value only takes effect after the current Mark or Space Count is completed.

The FRAME period may be calculated directly from the period of the Clock, T_{CLK} , and the counter modulus, TC, so that:

$$\text{FRAME PERIOD} = T_{CLK} \times TC$$

Alternatively, this may be expressed as:

$$\text{FRAME PERIOD} = T_{CLK} \times 2^n$$

for an n -bit binary counter where n is the counter width in bits. Likewise for the mark period:

$$\text{MARK PERIOD} = T_{CLK} \times \text{Mark_Value}$$

and so the duty cycle is the ratio of the two:

$$\text{DUTY CYCLE} = \frac{\text{Mark_Value}}{2^n}$$

Implementing the design without X-BLOX

A generic version of the design may be implemented simply by replacing the X-BLOX macros with standard Xilinx macros.

Macro Size

The size of the PWM macro, implemented using binary counters, is:

$$2n+1 \text{ Flip-Flops}$$

which corresponds to $n+1$ XC4000 CLBs or $2n+1$ XC7000 macrocells.

Output Signal Conditioning

The digital PWM output is not of itself an analog signal. In many applications a basic low pass filter, such as an RC network, will be sufficient to provide this conversion and reduce the high frequency components in the output. High fidelity applications need much more sophisticated output signal conditioning to remove the intrinsically high frequency components such as T_{CLK} . This is often achieved using digital filters along with other techniques such as decimation. In all cases the user must take care to match the filter time constants to those of the PWM generator itself.

The topic of output conditioning is beyond the scope of this brief.

Using the Design Files

This design is available from the Xilinx BBS (see "Xilinx Technical Bulletin Board" in Section 6 of the **Xilinx Data Book**). This section describes what software is required to run the design and the steps involved. Also, please read through the **Limitations and Restrictions** section.

Software Requirements

The following software is required to process this design:

- PKUNZIP 2.04e, or later, unarchiving program.
- VIEWdraw or VIEWdraw-LCA schematic editor. This software is required in order to modify the schematics.
- Xilinx XACT 5.0, or later, FPGA development system, including the PPR place and route program and the X-BLOX module generator.

Using the Design on Your System

1. Create a new directory called **PWM** on your hard disk.
2. Copy the files and sub-directories from the **/MISCAPPS/PWM/DESIGNS** directory on the **Programmable Logic Breakthrough '95** CD-ROM into your **PWM** directory.

3. Edit the **VIEWDRAW.INI** file. Make sure that the VIEWlogic design library pointers are set appropriately for your machine. You will find the library pointers near the end of the file.
4. Invoke XDM.
5. Set the part type to XC4003A-6PC84C, (the design will work with any other part).
6. Run XMAKE on PWM.MAK to process the design. The schematic file is named PWM.1.

Limitations and Restrictions

WARNING: THIS IS AN UNTESTED DESIGN.

Xilinx, Inc. does not make any representation or warranty regarding this design or any item based on this design. Xilinx disclaims all express and implied warranties, including but not limited to the implied fitness of this design for a particular purpose and freedom from infringement. Without limiting the generality of the foregoing, Xilinx does not make any warranty of any kind that any item developed based on this design, or any portion of it, will not infringe any copyright, patent, trade secret or other intellectual property right of any person or entity in any country. It is the responsibility of the user to seek licenses for such intellectual property rights where applicable. Xilinx shall not be liable for any damages arising out of or in connection with the use of the design including liability for lost profit, business interruption, or any other damages whatsoever.

Design Support and Feedback

This application note may undergo future revisions and additions. If you would like to be updated with new versions of this application note, or if you have questions, comments, or suggestions please send an E-mail to

apps@xilinx.com

or a FAX addressed to "PWM Application Brief Developers" sent to

1+(408) 879-4442.

IMPORTANT: Please be sure to include which version of the application note you are using. The version number is in the lower right-hand corner of page 1.

Acknowledgments

The rather creative use of the Terminal Count to control the generation of the Pulse Width was proposed by Bernie New at Xilinx.

Figure 1. Eight-bit, scalable generator using PWM macro.