LDP-001 Pulse Modulator
User Manual

Dr. Larry D. Pyeatt

March 24, 2025

Chapter 1

Introduction

The LDP-001 Pulse Modulator is a configurable VHDL component that can be instantiated to control any number of pulse modulated outputs, referred to as "channels". In the SDSMT CENG 448 platform, it is configured to provide five channels. Channel 0 is routed to the audio jack on the Digilent Nexys A7 FPGA board. The audio jack output on the NEXYS A7 FPGA board provides a hardware low-pass filter that will help to convert a pulse modulated signal into an audio waveform. The remaining channels (1 through 4) of the LDP-001 Pulse Modulator are routed to PMOD header JB as shown in Figure 1.1.

Each channel is controlled by four registers. The following table shows the register names of each channel and their offsets relative to the base address of the channel:

Name	**Alias**	**Offset**
Control and Status Register	CSR	B_c
Clock Divisor Register	CDR	$B_c + \texttt{0x4}$
Base Cycle Time Register	BCR	$B_c + \texttt{0x8}$
Duty Cycle Register	DCR	$B_c + \texttt{0xC}$

where B_c is the base address of the channel, which can be calculated as

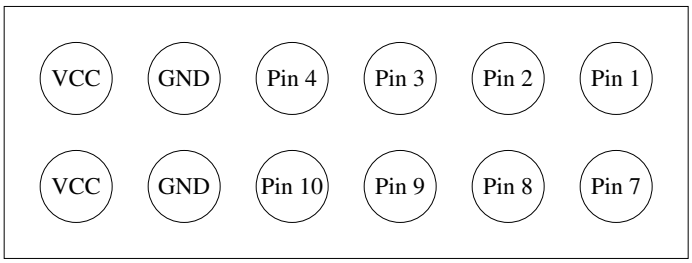
$$B_c = B_d + 16n,$$

where B_d is the base address of the pulse modulator device, and n is the channel number.

Writing to the Control and Status Register (CSR) will enable/disable the channel, set the type of modulation, enable/disable the FIFO and control the interrupt settings for the channel. When read, the CSR provides the FIFO and interrupt status of the channel. The Clock Divisor Register (CDR) can be used to divide the system clock to provide a slower clock, so that lower base frequencies can be achieved. The Base Cycle Time Register (BCR) sets the base cycle time, and the Duty Cycle Register (DCR) sets the duty cycle. The following sections provide more information on the individual registers.

1.1 FIFO

Each channel contains a FIFO, so that duty cycle settings can be buffered. If the FIFO is enabled, then a value as taken from the FIFO at the beginning of each base cycle. This is useful for applications such as streaming audio. For more information, see Section 1.1.



Pin 1: PWM Channel 1 Output
Pin 2: PWM Channel 2 Output
Pin 3: PWM Channel 3 Output
Pin 4: PWM Channel 4 Output
Pin 7: PWM Channel 1 Enabled
Pin 8: PWM Channel 2 Enabled
Pin 9: PWM Channel 3 Enabled
Pin 10: PWM Channel 4 Enabled

Figure 1.1: Pin assignments for PMOD header JB.

1.2 Modes

Each channel can be independently configured for Pulse Width Modulation (PWM), or Pulse Density Modulation (PDM).

Chapter 2

Registers

This section describes the operation of each of the four registers in a channel.

2.1 Control and Status Register (CSR)

The CSR is a 32-bit read/write register that is used to configure the channel, read the status of the FIFO, and read the interrupt status of the channel. Bits in the Control and Status Register (CSR) are listed in the following table.

Bit(s)	Name	R/W	Description
0	OE	rw	Output enable
1	IO	rw	Invert output
2	SLFM	rw	synchronous load/FIFO mode
3	PDMM	rw	PDM mode
4	IE	rw	Interrupt enable.
5	RF	wo	Reset FIFO
6–7	Unused		
8	FF	ro	FIFO full
9	FE	ro	FIFO empty
10	IA	ro	Interupt active.
11–26	Unused		
27–31	FIL	rw	FIFO interrupt level

OE When this bit is set to '1', pulse modulated output is enabled for this channel. Otherwise, the channel continuously outputs a '0' if the IO bit is set to '0', or a '1' if he IO bit is set to '1'. When this bit is set to '1', none of the other bits in the CSR can be written to (except for the IE bit), nor can the CDR or BCR be modified. In order to change the contents of the CDR, BCR, or other bits in the CSR, this bit must first be set to '0'.

IO When this bit is set '1', the output is inverted, producing a '1' when it normally would produce a '0', and vice-versa.

SLFM When this bit is set to '0', the FIFO is disabled. A write to the duty cycle register (DCR) has immediate effect. When this bit is set to '1' the FIFO is enabled. A write to the DCR is

routed into the FIFO instead of directly to the DCR. At the beginning of every base cycle, the next value from the FIFO will be moved into the DCR.

PDMM Writing '1' to this bit will configure the channel to produce pulse *density* modulated signal. Writing '0' will configure it for pulse *width* modulation.

IE This bit controls whether or not this channel is allowed to send interrupts. It is only relevant when the channel is in FIFO mode. Writing a '1' to this bit enables interrupts from this channel.

RF This write-only bit will reset the FIFO, losing all its contents, if a '1' is written. Otherwise the FIFO will be unaffected. This bit always reads as '0'.

FF This read-only bit indicates whether or not the FIFO is full. If the FIFO is full, then this bit will be set to '1'. If there is still room in the FIFO, then this bit will be set to '0'.

FE This read-only bit indicates whether or not the FIFO is empty. If the FIFO is empty, then this bit will be set to '1'. If the FIFO contains any data, then this bit will be set to '0'.

IA This read only bit indicates whether or not the channel is signaling an interrupt. The channel will signal an interrupt when it is in FIFO mode and the number of items in the FIFO is less than the number specified in the FIL field of the CSR. A '1' indicates that this channel is signaling an interrupt. Writing enough data to the channel so that the number of items in the FIFO is greater than the number in the FIL field of the CSR will clear the interrupt condition for the channel.

FIL This field sets the FIFO interrupt level. If FIFO mode is enabled for this channel, and the IE bit for this channel is set to '1', then this channel will signal an interrupt whenever the number of items in the FIFO is less than the number in the FIL field.

2.2 Clock Divisor Register (CDR)

The CDR is a 32/16-bit register used to control division of the system PM clock. The register is 32 bits wide, but only the least-significant 16 bits are used. This register is used to provide additional control over the base cycle time and duty cycle resolution. Each pulse modulator channel has two free-running counters, **clockdiv** and **base**. The first counter (**clockdiv**) is incremented on each rising edge of the PM clock (provided by the system clock circuitry). If the **clockdiv** counter equals the CDR, then the **base** counter is incremented and the **clockdiv** counter is reset to zero at the beginning of the next *pmclk* cycle.

The CDR cannot we written when the OE bit in the CSR is set to '1'. Writing a value of zero into the CDR causes the **base** counter to increment on each rising edge of the system PM clock. Writing a value of one into the CDR causes the **base** counter to increment at half the rate of the system PM clock, and so on. The rate at which the **base** counter is incremented is calculated as

$$baseclk = \frac{pmclk}{CDR+1}, \tag{2.1}$$

where *baseclk* is the frequency (in Hz) at which the **base** counter is incremented, *pmclk* is the frequency (in Hz) of the system PM clock, and *CDR* is the value that has been loaded into the Clock Divisor Register (CDR).

2.3 Base Cycle Time Register (BCR)

The BCR is a 32/16-bit register used to set the cycle time (also known as the base frequency) of the pulse modulator channel. The register is 32 bits wide, but only the least-significant 16 bits are used. The **base** counter counts from zero up to the value in the BCR at the rate set by *baseclk*. When the value in the base counter matches the value in the BCR, the base counter is reset to zero at the beginning of the next *baseclk* cycle. The base cycle time can be calculated as

$$bct = \frac{BCR + 1}{baseclk}, \tag{2.2}$$

where *BCR* is the value that has been loaded into the base cycle time register (BCR) and *baseclk* is obtained from Equation 2.1. The base frequency is calculated as

$$bcf = bct^{-1} = \frac{baseclk}{BCR + 1}. \tag{2.3}$$

The BCR cannot we written when the OE bit in the CSR is set to '1'.

2.4 Duty Cycle Register (DCR)

The DCR is a 32/16-bit register used to set the duty cycle of the pulse modulator channel. The register is 32 bits wide, but only the least-significant 16 bits are used. The DCR controls the amount of time in each base cycle that the channel spends with its output at '1' vs the amount of time that it spends with its output at '0'.

For correct operation, the value stored in the DCR must be between 0 and $BCR + 1$. For example, if the BCR is loaded with a value of four, then there are five valid values for the DCR as shown in the table below:

Value Stored in DCR	Duty Cycle %
0	0%
1	25%
2	50%
3	75%
4	100%

Each channel in the pulse modulation device has two modes of operation: pulse *width* modulation (PM), and pulse *d*ensity modulation (PDM). In both modes, the DCR controls the total time that the output is '1' versus '0', but they produce different pulse trains.

Chapter 3

Pulse Modulation

In some situations, it is useful to have the ability to turn a device on at varying levels. For instance, it could be useful to control a motor at any required speed, or control the brightness of a light source. One way that this can be accomplished is through pulse modulation.

The basic idea is that the computer sends a stream of pulses to the device. The device acts as a low-pass filter, which averages the digital pulses into an analog voltage. By varying the percentage of time that the pulses are high versus low, the computer can control how much average energy is sent to the device. Varying the percentage of time that the pulses are high versus low is known as the *duty cycle*. Varying the duty cycle is referred to as *modulation*. There are two major types of pulse modulation: pulse density modulation (PDM) and pulse width modulation (PWM). Most pulse modulation devices are configured in three steps as follows:

1. The base frequency of the clock that drives the PM device is configured. This step is usually optional.

2. The mode of operation for the pulse modulation device is configured by writing to one or more configuration registers in the pulse modulation device.

3. The cycle time is set by writing a "range" value into a register in the pulse modulation device. This value is usually set as a multiple of the base clock cycle time.

Once the device is configured, the duty cycle can be changed easily by writing to one or more registers in the pulse modulation device.

In pulse width modulation mode, all of the '1' pulses are clustered at the beginning of the base cycle. When the duty register is greater than the base counter, the PM output is '1'. Otherwise it is '0'. This can be reversed by setting the IO bit in the CSR.

In pulse width modulation, the frequency of the pulses remains fixed, but the duration of the positive pulse (the pulse width) is modulated. When using PM devices, the programmer typically sets the device cycle time t_c in a register, then uses another register to specify the number of base clock cycles, d, for which the output should be high. The percentage $\frac{d}{t_c} \times 100$ is typically referred to as the duty cycle and d must be chosen such that $0 \leq d \leq t_c$. For instance, if $t_c = 1024$, then the device cycle time is 1024 times the cycle time of the clock that drives the device. If $d = 512$, then the device will output a high signal for 512 clock cycles, then output a low signal for 512 clock cycles. It will continue to repeat this pattern of pulses until d is changed.

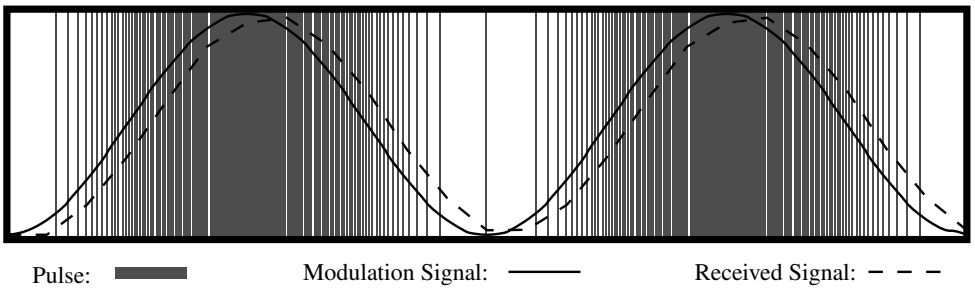


Figure 3.1: Pulse Density Modulation.

3.1 Pulse Density Modulation (PDM)

With Pulse Density Modulation (PDM), also known as Pulse Frequency Modulation (PFM), the duration of the positive pulses does not change, but the time between them (the pulse density) is modulated. Each pulse has the same duration as one period of the base cycle clock.

Figure 3.1 shows a signal (solid sine wave) that is being sent using PDM, and the resulting set of pulses. Each pulse transfers a fixed amount of energy to the device. When the pulses arrive at the device, they are filtered using a low pass filter. The resulting received signal is also shown (dashed sine wave). Notice that the received signal has a delay, or phase shift, caused by the low-pass filtering. This approach is suitable for controlling certain types of devices, such as lights and speakers.

However, when driving such devices directly with the digital pulses, care must be taken that the minimum frequency of pulses remains above the threshold that can be detected by human senses. For instance, when driving a speaker, the minimum pulse frequency must be high enough that the individual pulses cannot be distinguished by the human ear. This minimum frequency is around 40 KHz. Likewise, when driving an LED directly, the minimum frequency must be high enough that the eye cannot detect the individual pulses, because they will be seen as a flickering effect. That minimum frequency is around 70 Hz. To reduce or alleviate this problem, designers may add a low-pass filter between the PM device and the device that is being driven.

An LDP-001 Pulse Modulator channel in pulse density modulation mode will output one pulse for each value in the **base** counter. The DCR register controls the number of clock periods where the output is '1' versus the number of clock periods were the output is '0', but the pulses are spread out over the base cycle time. The polarity can be reversed by setting the IO bit in the CSR.

3.2 Pulse Width Modulation (PWM)

Figure 3.2 shows a sine wave signal that is being sent using pulse width modulation. The pulses are also shown. Each pulse transfers some energy to the device. The width of each pulse determines how much energy is transferred. When the pulses arrive at the device, they are effectively filtered using a low pass filter. The resulting received signal is shown by the dashed line. As with PDM, the received signal has a delay, or phase shift, caused by the low-pass filtering.

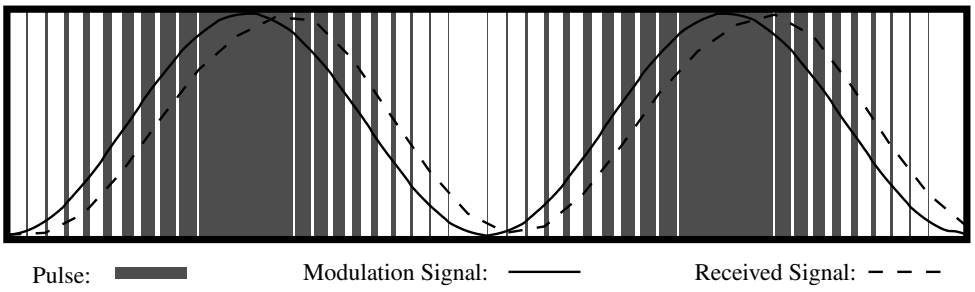


Figure 3.2: Pulse Width Modulation.

3.3 PWM vs PDM

One advantage of PWM over PDM is that the digital circuit is not as complex. Also, when driving motors it is usually necessary to match the pulse frequency to the size and type of motor. Mismatching the frequency can cause loss of efficiency, as well as overheating of the motor and drive electronics. In severe cases, this can cause premature failure of the motor and/or drive electronics. With PWM, it is easier for the programmer to control the base frequency, and thereby avoid those problems. In audio applications, PDM may be preferred. PDM can send very short pulses at higher frequencies. The higher frequency pulses are more easily low-pass filtered so that the individual pulses cannot be detected by human senses.

Chapter 4

Programming

4.1 Example 1

Suppose our PM clock is 100 MHz and we want to control a DC motor. We have determined that the motor works well with a base frequency of 100 Hz. We would like to control motor speed with 1000 possible duty cycle settings. We could load 99 into the clock divisor register (CDR), which would give a 1 MHz clock to the **base** counter. We could then load 999 into the BCR, which would give 1000 divisions of the 1 MHz clock, with each division being $\frac{1}{1000}$ s. We can now load the duty cycle register with any value between 0 and 1000 to set the duty cycle, where 0 is always off (0%), and 1000 is always on (100%).

4.2 Example 2

Suppose that our PM clock is running at 150 MHz and we want to play a 16-bit audio stream sampled at 44.1 KHz using pulse modulation. We would need a base frequency of 44.1 KHz, and we would want 16 bits to be used in the duty register.

In order to achieve this, the **base** counter would need to be incremented at a rate of $44.1\,\text{KHz} \times 2^{16} = 2,890,137,600\,\text{Hz}$. That is the speed that we would need to run the PM clock, and it is almost 20 times faster than the 150 MHz clock that we have. Therefore, we cannot achieve 16-bit audio at 44.1 KHz. However, if we are willing to accept a lower audio depth (the number of bits in each sample), and a lower sample rate, then we may be able to produce acceptable audio.

If we reduce our audio depth from 16 bits to 12 bits, then we only need to run our PM device at 180,224,000 Hz (180.224 KHz) in order to play 44000 samples per second. That is still slightly out of our range. The maximum sample rate that we can achieve with 12 bits of audio depth and a 150 MHz PM clock is

$$f_{max} = \frac{150\text{MHz}}{2^{12}} = 36621.09375\text{Hz}.$$

We can re-sample our audio signal and play it at 36.621 KHz and 12 bits of depth. Note that we could get 13 bits of audio depth if we reduce the sample rate to 18310.546875 Hz. We can trade lower audio depth for higher sample rate and vice versa.

4.3 Interrupt-driven Operation with an RTOS

The LDP-001 Pulse Modulator device has multiple channels, and we would like to set up a driver so that each channel may be controlled by a different task. The driver should keep track of which task owns each channel and prevent other tasks from accessing the channel. The task that owns a channel is allowed to control the channel configuration and specify the interrupt handler(s) for that channel(s). This will result in good code modularization. The tasks can request PM channel(s) and have the ISR for the associated channel(s) in the same file as the task code.

All channels on the LDP-001 Pulse Modulator device are connected to the same hardware interrupt, so the software driver must provide a single interrupt service routine (ISR) that polls each channel. The ISR must call the associated interrupt handler for each channel that is currently signaling an interrupt. An efficient method for implementing the driver is to define a device control block, or descriptor, for each channel. The descriptor should should include

- the address of the channel's CSR,
- the owner task handle, (so that error checking can be performed), and
- a pointer to the channel's ISR handler.

The driver code should provide some functions for allocating and using channels. The following listing provides a suggested API for the pulse modulator.

```
#ifndef PULSE_MODULATOR_DRIVER_H
#define PULSE_MODULATOR_DRIVER_H

#include <FreeRTOS.h>

// Get exclusive access to the pulse modulator channel.
BaseType_t PM_acquire(int channel);

// Release the channel. This function also disables the channel.
void PM_release(int channel);

// Set the interrupt handler function for the channel.
void PM_set_handler(int channel, void (*handler)(void));

// Set the base frequency and number of divisions for the
// channel.
void PM_set_cycle_time(int channel, int divisions, int
   base_frequency);

// Put the channel in PDM mode.
void PM_set_PDM_mode(int channel);

// Put the channel in PWM mode.
void PM_set_PWM_mode(int channel);

```

```
25 // Enable the FIFO. Return zero if the handler, cycle time, or
26 // mode have not been set.
27 int PM_enable_FIFO(int channel);
28
29 // Disable the FIFO.
30 void PM_disable_FIFO(int channel);
31
32 // Enable output on the channel. Return zero if the channel is
33 // not fully configured.
34 int PM_enable(int channel);
35
36 // Disable output on the channel.
37 void PM_disable(int channel);
38
39 // Enable interrupts on the channel.
40 int PM_enable_interrupt(int channel);
41
42 // Disable interrupts on the channel.
43 void PM_disable_interrupt(int channel);
44
45 // Set the duty cycle for the channel.  If the channel is in
46 // FIFO mode, this function writes to the FIFO.  Otherwise,
47 // it writes directly to the Duty Cycle Register.
48 void PM_set_duty(int channel,int duty);
49
50 // If the channel is in FIFO mode, this function returns 1 if
51 // the FIFO is full. In all other cases, it returns zero.
52 int PM_FIFO_full(int channel);
53
54 #endif
```

Every function in the driver code (except `PM_acquire`) should check to make sure that the currently running task owns the channel (use `xGetCurrentTaskHandle`). The `PM_acquire` function shoud verify that the owner is NULL and hang the system in an infinite loop for debugging if two tasks try to acquire the same PM channel, and set the owner handle if everything is okay. Additionally, the acquire and release functions should be protected by a mutex global to the PM code. i.e. The `channel` array should only be modified within a mutex.