Developing a PWM Interface using LabVIEW FPGA

Publish Date: Feb 26, 2013

Overview

Engineers and designers who develop and test automotive electronics, avionics, digital sensors and other similar devices often need to measure and simulate devices that generate pulse width modulation (PWM) signals. The LabVIEW FPGA Module is a tool that you can use to create PWM interfaces for your test and measurement system. Unlike dedicated PWM I/O devices, LabVIEW FPGA allows you to customize the PWM channel features and behavior to your application, and to integrate and synchronize them with other measurement devices.

Table of Contents

1. Application Overview

The LabVIEW FPGA module and reconfigurable I/O card can be used to implement a variety of custom interfaces, including:
- timing and trigger functionality for other measurement devices,
- digital communication protocols
- device simulation in rapid prototyping or hardware-in-the-loop applications
- AC and DC sensor simulation

In addition to these applications, you can also use LabVIEW FPGA to implement pulse width modulation (PWM) inputs and outputs. PWM signals are not unique to a particular class of devices, but are used in various applications to transfer a wide range of measurements. This application note will show you how to build PWM interfaces using the reconfigurable I/O board and LabVIEW FPGA module.

Traditionally, PWMs signals have been measured and generated using counters. Using a general-purpose counter for PWM signals presents a challenge in programming the application, due to the hardware capabilities and application programming interface (API) that are optimized for a wide range of counter applications. LabVIEW FPGA allows you to design PWM I/O channels with hardware and software interfaces that are customized to your application, which makes them easy to integrate into your test or measurement application.

2. Pulse Width Modulation

Pulse Width Modulation (PWM) is a modulation method which encodes a value using the width of a pulse or continuous pulse train. Commonly a PWM signal uses a continuous square wave signal of constant frequency and variable duty cycle. The duty cycle or individual pulse widths represent the value of the signal. This value has a defined range of 0 to 1 or 0 to 100 % duty cycle. This PWM value corresponds to a defined range of an engineering value used in an application, such as the rotational speed of a wheel in RPMs. Because the value range of the PWM signal is not open-ended the engineering value also has a limited range. For example the range of 0 to 1 of the PWM value may correspond to 0 to 8000 RPMs of a sensor. Rotational speeds greater than 8000 RPMs could not be measured with this sensor.

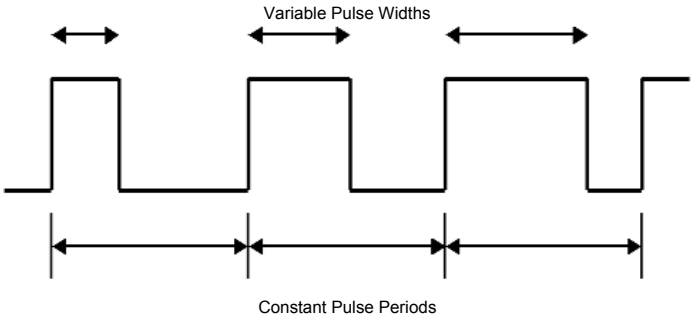

Figure 1: Example of a Pulse Width Modulation (PWM) Signal

3. Simple PWM Input

In the following examples, a digital I/O line is used to acquire the PWM signal, which is processed on the FPGA to extract the PWM value. Essentially, the code looks for consecutive rising and falling edges and based on the time passed between edges we determine the pulse width and pulse period. Dividing the pulse width by the pulse period gives us the PWM value, from which the engineering value can be derived.

The following diagram shows the implementation of a simple PWM input. The VI monitors a digital line for falling and rising edges and measures the time between each pair of edges. The length of each phase of the signal is written into a front panel cluster which can be retrieved by the host application. The frequency and duty cycle of the PWM signal is calculated from the length of the high and low phase in the host application. By placing the PWM calculation on the host application, we can reduce the FPGA space required by the implementation, allowing us to place a larger number of PWM inputs or other code on one card. This also reduces the amount of time required in the loop, which enables us to measure smaller pulse widths and pulse periods.

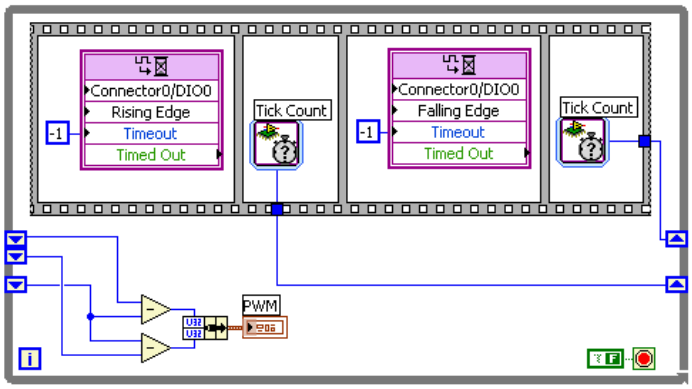

Figure 2: A simple PWM input

To further optimize the performance of the loop, we pipeline the code between two iterations of the loop. Detecting and timestamping the digital pulse edges is done in the first iteration and the timestamp values are passed to a pair of shift registers. The next iteration of the loop retrieves the timestamp values and calculates the pulse lengths, writing them to a cluster on the front panel for retrieval by the host application. Splitting these task between two loop operations reduces the overall loop cycle time. We use a cluster on the front panel to ensure that both values, the high pulse length and low pulse length, are updated at the same time and read by the host application together.

A common enhancement to the simple PWM input is to add a latch option. The latch Boolean is used to lock or latch the current reading in the output register. Rather than updating the output registers with new PWM values, the VI stores the value until the latch is released by the host. This feature is commonly used to read the measurement at a specific time controlled by the latch operation from the host. A different option would be to latch the PWM value based on another digital signal or trigger such as a pulse signal read from the RTSI/PXI trigger bus. This option would provide better synchronization to an external process compared to latching the reading from the host application.

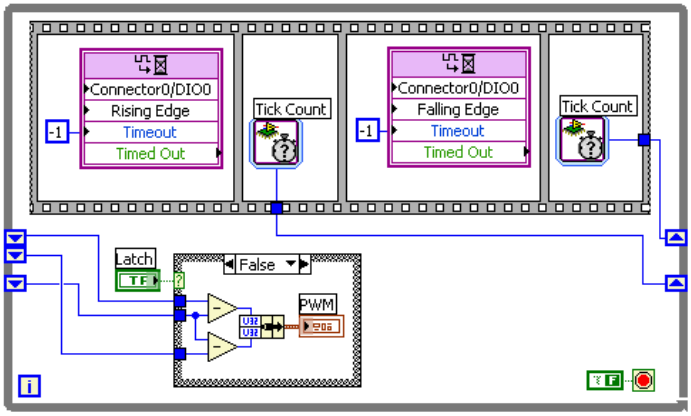

Figure 3: Simple PWM Input with Latch

Instead of reading the PWM value intermittently and using the latch operation, an application might need to read the PWM value for every period of the PWM signal. In this case we need to buffer the PWM data at the end of each iteration of the loop which corresponds to one cycle of the PWM signal. Buffering data on the FPGA before sending back to the host application is illustrated in the referenced buffering example.

To test your PWM input, you will need a PWM signal generator. If a suitable signal source is not available, you can use an FPGA-generated PWM output as described in the PWM output section of this application note.

See Also:
PWM Input Examples in LabVIEW FPGA and the 7831R

4. Advanced PWM Input

In the advanced PWM input implementation, the calculation of the PWM value is moved from the host application to the FPGA VI. Based on the measured high and low phases of the signal, the VI determines the PWM value using integer math, due to the lack of floating-point math support on the FPGA. The PWM value is returned as a 16-bit unsigned integer corresponding to the full range of the PWM value. For example, a value of 32768 corresponds to 0.5 or 50% PWM value.

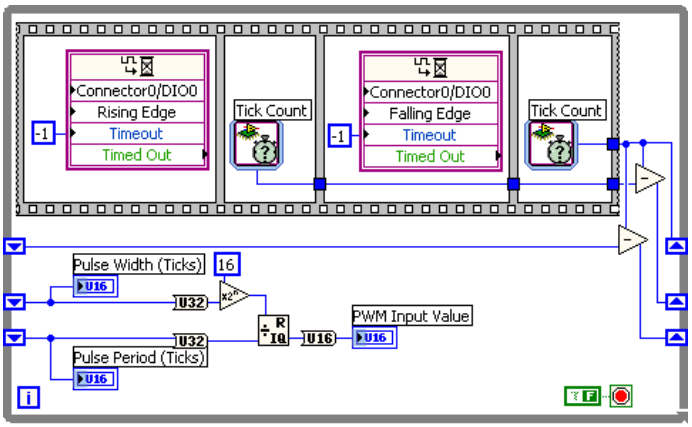

Figure 4: Advanced PWM Input

To achieve reasonable resolution on the calculated PWM value, the time stamps are measured as 16-bit values, allowing us to use the extended 32-bit range for the division of the two pulse length values. This means we have a limited range for the pulse lengths that we can measure compared to the simple PWM input implementation which uses 32-bit integers for time stamping. In each specific application, you can choose the time units of the Tick Count functions that are used to time stamp the edges on the PWM signal. By selecting ticks, microseconds or milliseconds for the Tick Count function, you can best match the PWM input implementation to the expected PWM signal with regards to the frequency of the signal.

See Also:
PWM Input Examples in LabVIEW FPGA and the 7831R

5. Multiplexed PWM Input

In some applications, it may be necessary to monitor a large number of PWM signals. Due to space limitations on the FPGA however it may not be feasible to implement a large number of independent PWM inputs. An alternate solution for this situation is to build a multiplexed PWM input, in which you monitor a limited number of PWM signals selected from a large number of available signals. Similar to a multiplexer which switches one of many signal to an input, this option select one or several PWM signals to process from many signals connected to the different DIO lines of the 7831R reconfigurable I/O board.

The following diagram shows a multiplexed PWM input which allows the host application to select from 40 different input signals routed from the 40 digital lines of one of the DIO connectors on the 7831R board. Each microsecond, the VI acquires all forty digital lines grouped in 5 ports of 8 lines each. The current state of all 40 lines is stored using a Boolean array with 40 elements, which is passed to the shift register of the loop. The VI processes data for the PWM signals selected by the Active PWMs control. For each selected PWM the VI retrieves the current and most recent state of the signal and detects rising or falling edges of the signal. Each edge in the signal is time stamped and the information stored in the cluster array. The cluster array contains timestamp information about the most recent edges of each active signal as well as the length of the most recent complete high and low pulse of the signal. Based on the pulse lengths of each PWM signal, the host application calculates the PWM value for each active signal.

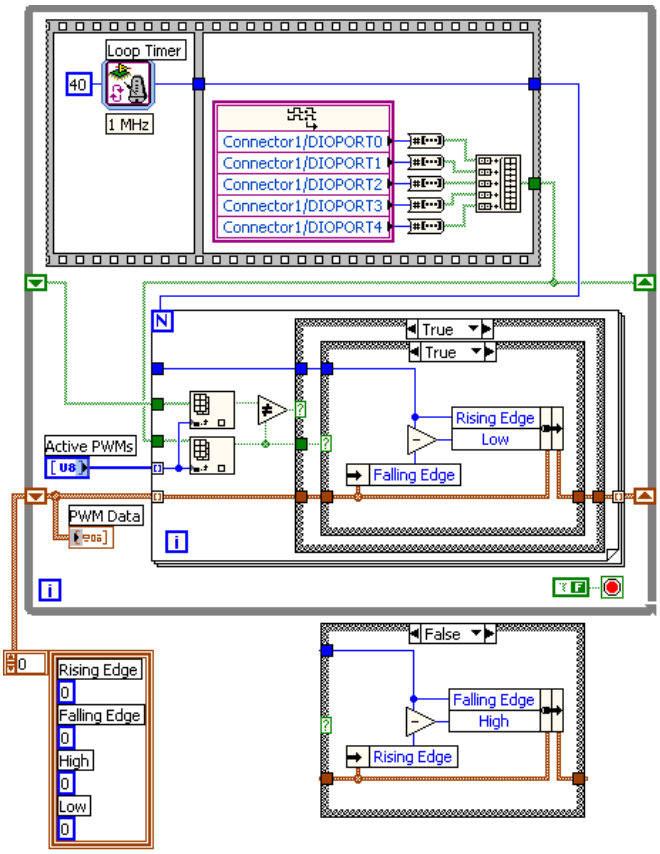

Figure 5: Multiplexed PWM Input

Depending on the space available on the FPGA, the developer can set the number of active PWM signals that are processed at one time by setting the array size of the cluster array constant on the diagram, as well as the Active PWMs array and PWM Data array on the front panel. These three arrays must have the same number of elements corresponding to the total number of Active PWM signals. In addition the VI needs to be able to process all of the active PWM signals within the time set for the loop timer. If you increase the number of active PWM signals you may also need to increase the loop time to give the VI adequate time for processing.

See Also:
Multiplexed PWM Inputs with LabVIEW FPGA and the 7831R

6. PWM Output

Many applications require the test system to also generate PWM outputs. These signals can be used to simulate devices connected to the device-under-test in its normal operating environment.

The simple PWM output implementation in figure 6 is configured by setting the period of the PWM signal and the pulse width. As with the PWM input example, both parameters are specified in clock cycles (ticks) of the FPGA, although micro- or millisecond Loop Timer and Wait parameters can also be used.

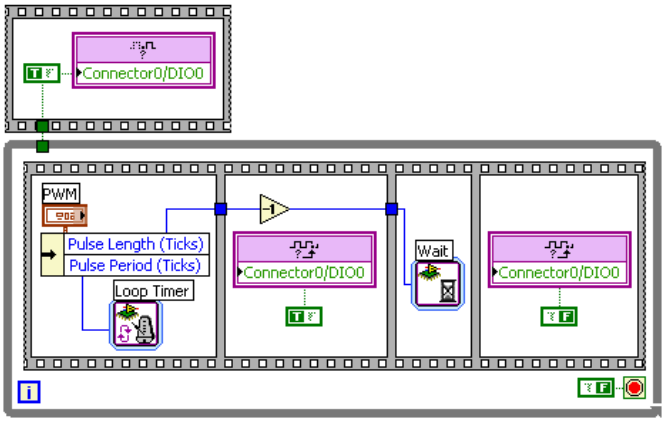

Figure 6: PWM Output

Even if you do not need PWM outputs in your application, they allow you to quickly test and characterize your PWM input implementation by connecting outputs to inputs a loopback configuration. LabVIEW FPGA also allows you to write an output level to a digital line using one part of your diagram while reading the same digital line in another part of the diagram, eliminating the need for an external loopback connection.

See Also:
PWM Output with LabVIEW FPGA

7. Multiple PWM Output

In applications that require multiple PWM outputs we can duplicate the simple PWM output algorithm. However if we have many output signals this approach duplicates a lot of code and makes this implementation inefficient with respect to space on the FPGA. For multiple PWM outputs we can design an implementation that combines the logic for multiple lines. The following implementation shows how to update eight PWM signals using one loop. The eight lines are contained within one digital port and the port is updated as a whole.

Rather than using front panel controls to store the current settings of each of the PWM signals we are using memory available on the reconfigurable I/O card, which is accessed using the memory extension utility available for the LabVIEW FPGA Module. This utility allows us to access larger blocks of memory and to access the memory as 8-, 16-, or 32-bit registers. Using the onboard memory to store the PWM settings greatly reduces the amount of FPGA space required by this implementation, especially when using multiple ports for additional output channels. On one of the DIO connectors of the 7831R reconfigurable I/O card we can implement up to 40 PWM outputs.

The top loop in the following diagram is used to communicate from the host application to the user memory to write the PWM settings to the card. Each PWM output uses two memory addresses. The first address is used to store the pulse width in microseconds, the second to store the width of the pause between pulses, also in microseconds. The two values added together represent the pulse period. Addresses 0 and 1 represent the first PWM output, addresses 2 and 3 the second PWM output, and so on.

Because we need to process all of our PWM output signals in the lower loop, the loop is configured using the Loop Timer functions to run at one microsecond intervals. Compiled for a 80 MHz clock rate of the FPGA, this gives us 80 clock cycles of the FPGA to do all of our processing.

Inside the main loop we maintain and update a state array which stores the current state and the time until the next state change for each PWM output, eight in all. Inside the main (While) loop, a For loop is used to process the state for each individual PWM output. We decrement the Next Edge value in the state array during each cycle of the main loop until it reaches zero. At this state transition the length of the following pulse or pause is read from the user memory and the state array is updated. The For loop also combines the state of each PWM output into a Boolean array which is passed to a shift register, converted to an 8-bit integer, and used to update the digital port. Due to the microsecond timing of the loop, the resolution of this implemention is one microsecond for the pulse and pause cycle of the PWM signal.

If the PWM settings in the memory buffer are uninitialized or set to zero the output of the PWM line is set low.

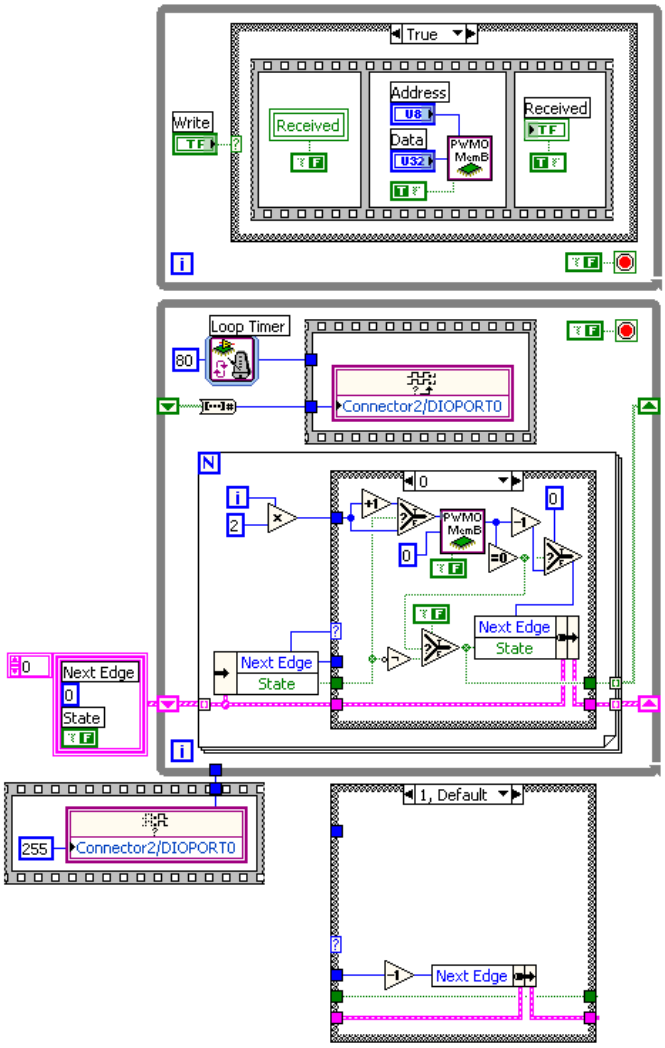

Figure 7: Multiplexed PWM Output

See Also:

PWM Output with LabVIEW FPGA

8. Special Considerations

Values near both ends of the PWM range (near 0 and 1) can create a difficult situation for PWM demodulation due to extremely small pulse widths they produce. Many PWM input devices, including the implementations shown here, have a minimum and maximum PWM value that they can measure.

In the above LabVIEW FPGA PWM input implementations, we measure the time between consecutive edges of the PWM signal. Due to the nature of the FPGA and its internal clock, as well as the individual implementation, there are limits to the minimum pulse widths that can be measured. These may vary between the high pulse and low pulse of the signal. The measurable range is determined by these minimum pulse widths and the frequency (pulse period) of the PWM signal.

For example, if in your implementation the minimum measurable pulse width is 5 cycles of the FPGA clock and the PWM frequency is 5 kHz, the following range of PWM values can be measured.

Pulse Width: 5 * (1 / 40MHz) = 5 * 25ns = 125ns = 0.125us
Pulse Period: 1 / 5kHz = 200us

Minimum PWM Value: 0.125us / 200 us = 0.000625 = 0.0625%
Maximum PWM Value: 1 - 0.000625 = 0.999375 = 99.9375%

(us = microsecond)

The specific measurable PWM range is dependent on the implementation as well as the clock rate of the FPGA. Therefore it is important that the developer is aware of these limitation and designs the code according to the requirements of the application.

For the PWM input, another boundary condition exists when the internal timer reaches its maximum value and rolls over back to zero. In this situation for one cycle there will be an incorrect value calculated for the duration of the pulse and period. This error can be eliminated with some additional programming by checking the consecutive timing values for a rollover condition and skipping the update of the measured value.

9. Conclusion

LabVIEW FPGA and the reconfigurable I/O board provide broad flexibility for designing PWM inputs and outputs that are optimized suitable for each individual application. Whether focusing on extreme measurement ranges or ease-of-use, multiplexing inputs or running parallel channels, a developer can customize the design for optimal performance. The ability to integrate the PWM interface with other signal types and measurements, and to synchronize PWM measurements with internal and external timing and trigger signals, further extends the range and functionality of these basic building blocks.