Published on *LabJack* (https://labjack.com)

2.9 - Timers/Counters [U6 Datasheet]

Timers / Counters Overview

The U6 has 4 timers (Timer0-Timer3) and 2 counters (Counter0-Counter1). When any of these timers or counters are enabled, they take over an FIO/EIO line in sequence (Timer0, Timer1, Timer2, Timer3, Counter0, then Counter1), starting with FIO0+TimerCounterPinOffset. The valid range for TimerCounterPinOffset is 0-8, so Timer0 can appear on FIO0-EIO0, and the highest I/O ever used would be EIO5 for Counter1 if all 4 timers and both counters are enabled. Some examples:

1 Timer enabled, Counter0 disabled, Counter1 disabled, and TimerCounterPinOffset=0:
FIO0=Timer0

1 Timer enabled, Counter0 disabled, Counter1 enabled, and TimerCounterPinOffset=0:
FIO0=Timer0
FIO1=Counter1

2 Timers enabled, Counter0 enabled, Counter1 enabled, and TimerCounterPinOffset=8:
EIO0=Timer0
EIO1=Timer1
EIO2=Counter0
EIO3=Counter1

Timers and counters can appear on various pins, but other I/O lines never move. For example, Timer1 can appear anywhere from FIO1 to EIO1, depending on TimerCounterPinOffset. On the other hand, FIO2 (for example), is always on the screw terminal labeled FIO2.

Applicable digital I/O are automatically configured as input or output as needed when timers and counters are enabled, and stay that way when the timers/counters are disabled.

Timers and counters use digital I/O hardware so the digital I/O specs from Appendix A apply. An input recognizes 0.0-0.8V as low and 2.0-5.8V as high, so rising and falling edges must transition between those levels. For more information about signal connections see Section 2.8.1.

There are special channels numbers that can be used to read timer and counter values. These channel numbers can be used most places where you would use analog input channel numbers, such as LJLogUD and LJStreamUD. See Section 3.2.1 for more information.

Timers

The timers (Timer0-Timer3) have various modes available. These are listed in the following table and more details can be found in Section 2.9.1.

Table 2.9-1. U6 Timer modes

Index (Low-level & UD)	
0	16-bit PWM output
1	8-bit PWM output
2	Period input (32-bit, rising edges)
3	Period input (32-bit, falling edges)
4	Duty cycle input
5	Firmware counter input
6	Firmware counter input (with debounce)
7	Frequency output
8	Quadrature input
9	Timer stop input (odd timers only)
10	System timer low read (Default mode)
11	System timer high read
12	Period input (16-bit, rising edges)
13	Period input (16-bit, falling edges)
14	Line-to-Line input

All timers use the same timer clock. There are 7 choices for the timer base clock:

Table 2.9-2. U6 Timer clock base options

Index (Low-level/UD)	
0/20	4 MHz
1/21	12 MHz
2/22	48 MHz (Default)
3/23	1 MHz /Divisor
4/24	4 MHz /Divisor
5/25	12 MHz /Divisor
6/26	48 MHz /Divisor

The first 3 clocks have a fixed frequency, and are not affected by TimerClockDivisor. The frequency of the last 4 clocks can be further adjusted by TimerClockDivisor, but when using these clocks Counter0 is not available. When Counter0 is not available, it does not use an external FIO/EIO pin. The divisor has a range of 0-255, where 0 corresponds to a division of 256.

Counters

Each counter (Counter0 or Counter1) consists of a 32-bit register that accumulates the number of falling edges detected on the external pin. If a counter is reset and read in the same function call, the read returns the value just before the reset.

Note that Counter0 is not available with certain timer clock base frequencies. In such a case, it does not use an external FIO/EIO pin. An error will result if an attempt is made to enable Counter0 when one of these frequencies is configured. Similarly, an error will result if an attempt is made to configure one of these frequencies when Counter0 is enabled.

2.9.1 - Timer Mode Descriptions [U6 Datasheet]

Log in or register to post comments

2.9.1.1 - PWM Output (16-Bit, Mode 0) [U6 Datasheet]

Log in or register to post comments

Outputs a pulse width modulated rectangular wave output. Value passed should be 0-65535, and determines what portion of the total time is spent low (out of 65536 total increments). That means the duty cycle can be varied from 100% (0 out of 65536 are low) to 0.0015% (65535 out of 65536 are low).

The overall frequency of the PWM output is the clock frequency specified by TimerClockBase/TimerClockDivisor divided by 2^{16}. The following table shows the range of available PWM frequencies based on timer clock settings.

Table 2.9.1.1-1. 16-bit PWM Frequency Ranges

		PWM16 Frequency Ranges	
TimerClockBase		Divisor=1	Divisor=256
0	4 MHz	61.04	N/A

1 2	12 MHz 48 MHz (Default)	138.11 732.42	N/A N/A
3	1 MHz /Divisor	15.26	0.06
4	4 MHz /Divisor	61.04	0.238
5	12 MHz /Divisor	183.11	0.715
6	48 MHz /Divisor	732.42	2.861

The same clock applies to all timers, so all 16-bitPWM channels will have the same frequency and will have their falling edges at the same time.

PWM output starts by setting the digital line to output-low for the specified amount of time. The output does not necessarily start instantly, but rather has to wait for the internal clock to roll. For 16-bit PWM output, the start delay varies from 0.0 to TimerClockDivisor*65536/TimerClockBase. For example, if TimerClockBase = 48 MHz and TimerClockDivisor = 1, PWM frequency is 732 Hz, PWM period is 1.4 ms, and the start delay will vary from 0 to 1.4 ms.

If a duty cycle of 0.0% (totally off) is required, consider using a simple inverter IC such as the CD74ACT540E from TI. Or you can switch the mode of the timer to some input mode, and add an external pull-down to hold the line low when set to input.

2.9.1.2 - PWM Output (8-Bit, Mode 1) [U6 Datasheet]

Outputs a pulse width modulated rectangular wave output. Value passed should be 0-65535, and determines what portion of the total time is spent low (out of 65536 total increments). The lower byte is actually ignored since this is 8-bit PWM. That means the duty cycle can be varied from 100% (0 out of 65536 are low) to 0.4% (65280 out of 65536 are low).

The overall frequency of the PWM output is the clock frequency specified by TimerClockBase/TimerClockDivisor divided by 2^8. The following table shows the range of available PWM frequencies based on timer clock settings.

Table 2.9.1.2-1. 8-bit PWM Frequency Ranges

		PWM8 Frequency Ranges	
TimerClockBase	_	Divisor=1	Divisor=256

0	4 MHz	15625	N/A
1	12 MHz	46875	N/A
2	48 MHz (Default)	187500	N/A
3	1 MHz /Divisor	3906.25	15.259
4	4 MHz /Divisor	15625	61.035
5	12 MHz /Divisor	46875	183.105
6	48 MHz /Divisor	187500	732.422

The same clock applies to all timers, so all 8-bitPWM channels will have the same frequency and will have their falling edges at the same time.

PWM output starts by setting the digital line to output-low for the specified amount of time. The output does not necessarily start instantly, but rather has to wait for the internal clock to roll. For 8-bit PWM output, the start delay varies from 0.0 to TimerClockDivisor*256/TimerClockBase. For example, if TimerClockBase = 48 MHz and TimerClockDivisor = 256, PWM frequency is 732 Hz, PWM period is 1.4 ms, and the start delay will vary from 0 to 1.4 ms.

If a duty cycle of 0.0% (totally off) is required, consider using a simple inverter IC such as the CD74ACT540E from TI. Or you can switch the mode of the timer to some input mode, and add an external pull-down to hold the line low when set to input.

2.9.1.3 - Period Measurement (32-Bit, Modes 2 & 3) [U6 Datasheet]

Log in or register to post comments

Mode 2: On every rising edge seen by the external pin, this mode records the number of clock cycles (clock frequency determined by TimerClockBase/TimerClockDivisor) between this rising edge and the previous rising edge. The value is updated on every rising edge, so a read returns the time between the most recent pair of rising edges.

In this 32-bit mode, the processor must jump to an interrupt service routine to record the time, so small errors can occur if another interrupt is already in progress. The possible error sources are:

- Other edge interrupt timer modes (2/3/4/5/8/9/12/13). If an interrupt is already being handled due to an edge on the other timer, delays of a few microseconds are possible.
- If a stream is in progress, every sample is acquired in a high-priority interrupt. These interrupts could cause delays on the order of 10 microseconds.
- The always active U6 system timer causes an interrupt 61 times per second. If this interrupt

happens to be in progress when the edge occurs, a delay of about 1 microsecond is possible. If the software watchdog is enabled, the system timer interrupt takes longer to execute and a delay of a few microseconds is possible.

Note that the minimum measurable period is limited by the edge rate limit discussed in Section 2.9.2.

See Section 3.2.1 for a special condition if stream mode is used to acquire timer data in this mode.

Writing a value of zero to the timer performs a reset. After reset, a read of the timer value will return zero until a new edge is detected. If a timer is reset and read in the same function call, the read returns the value just before the reset.

Mode 3 is the same except that falling edges are used instead of rising edges.

Edge Rate Limits

This edge-detecting timer mode requires processing resources as an interrupt is required to handle each edge. See more about edge rate limits in Section 2.9.2.

2.9.1.4 - Duty Cycle Measurement (Mode 4) [U6 Datasheet]

Log in or register to post comments

Records the high and low time of a signal on the external pin, which provides the duty cycle, pulse width, and period of the signal. Returns 4 bytes, where the first two bytes (least significant word or LSW) are a 16-bit value representing the number of clock ticks during the high signal, and the second two bytes (most significant word or MSW) are a 16-bit value representing the number of clock ticks during the low signal. The clock frequency is determined by TimerClockBase/TimerClockDivisor.

The appropriate value is updated on every edge, so a read returns the most recent high/low times. Note that a duty cycle of 0% or 100% does not have any edges.

To select a clock frequency, consider the longest expected high or low time, and set the clock frequency such that the 16-bit registers will not overflow. In other words, to measure 0% to 100% duty cycle for a given signal frequency, you need to set your clock frequency low enough such that the overall period of the signal is less than 65535 * 1/TimerClockFrequency. That equates to:

fclock ≤ 65535 * fsignal

Note that the minimum measurable high/low time is limited by the edge rate limit discussed in Section 2.9.2.

When using the LabJackUD driver the value returned is the entire 32-bit value. To determine the high and low time this value should be split into a high and low word. One way to do this is to do a modulus divide by 2^16 to determine the

LSW, and a normal divide by 2^16 (keep the quotient and discard the remainder) to determine the MSW.

Writing a value of zero to the timer performs a reset. After reset, a read of the timer value will return zero until a new edge is detected. If a timer is reset and read in the same function call, the read returns the value just before the reset. The duty cycle reset is special, in that if the signal is low at the time of reset, the high-time/low-time registers are set to 0/65535, but if the signal is high at the time of reset, the high-time/low-time registers are set to 65535/0. Thus if no edges occur before the next read, it is possible to tell if the duty cycle is 0% or 100%.

Edge Rate Limits

This edge-detecting timer mode requires processing resources as an interrupt is required to handle each edge. See more about edge rate limits in Section 2.9.2.

2.9.1.5 - Firmware Counter Input (Mode 5) [U6 Datasheet]

Log in or register to post comments

On every rising edge seen by the external pin, this mode increments a 32-bit register. Unlike the pure hardware counters, these timer counters require that the firmware jump to an interrupt service routine on each edge.

Writing a value of zero to the timer performs a reset. After reset, a read of the timer value will return zero until a new edge is detected. If a timer is reset and read in the same function call, the read returns the value just before the reset.

Edge Rate Limits

This edge-detecting timer mode requires processing resources as an interrupt is required to handle each edge. See more about edge rate limits in Section 2.9.2.

2.9.1.6 - Firmware Counter Input With Debounce (Mode 6) [U6 Datasheet]

Log in or register to post comments

Intended for frequencies less than 10 Hz, this mode adds a debounce feature to the firmware counter, which is particularly useful for signals from mechanical switches. On every applicable edge seen by the external pin, this mode increments a 32-bit register. Unlike the pure hardware counters, these timer counters require that the firmware jump to an interrupt service routine on each edge.

The debounce period is set by writing the timer value. The low byte of the timer value is a number from 0-255 that specifies a debounce period in 16 ms increments (plus an extra 0-16 ms of variability):

Debounce Period = (0-16 ms) + (TimerValue * 16 ms)

In the high byte (bits 8-16) of the timer value, bit 0 determines whether negative edges (bit 0 clear) or positive edges (bit 0 set) are counted.

Assume this mode is enabled with a value of 1, meaning that the debounce period is 16-32 ms and negative edges will be counted. When the input detects a negative edge, it increments the count by 1, and then waits 16-32 ms before re-arming the edge detector. Any negative edges within the debounce period are ignored. This is good behavior for a normally-high signal where the switch closure causes a brief low signal (Section 2.8.1.3). The debounce period can be set long enough so that bouncing on both the switch closure and switch open is ignored.

Writing a value of zero to the timer performs a reset. After reset, a read of the timer value will return zero until a new edge is detected. If a timer is reset and read in the same function call, the read returns the value just before the reset.

Edge Rate Limits

This edge-detecting timer mode requires processing resources as an interrupt is required to handle each edge. See more about edge rate limits in Section 2.9.2.

2.9.1.7 - Frequency Output (Mode 7) [U6 Datasheet]

Log in or register to post comments

Outputs a square wave at a frequency determined by TimerClockBase/TimerClockDivisor divided by 2*Timer#Value. The Value passed should be between 0-255, where 0 is a divisor of 256. By changing the clock configuration and timer value, a wide range of frequencies can be output, as shown in the following table:

Table 2.9.1.7-1. Mode 7 Frequency Ranges

		Mode 7 Frequency Ranges	
		Divisor=1	Divisor=1
TimerClockBase		Value =1	Value = 256
0	4 MHz	2000000	7812.5
1	12 MHz	6000000	23437.5
2	48 MHz (Default)	24000000	93750
		Divisor=1	Divisor=256
		Value =1	Value = 256
3	1 MHz /Divisor	500000	7.629
4	4 MHz /Divisor	2000000	30.518
5	12 MHz /Divisor	6000000	91.553
6	48 MHz /Divisor	24000000	366.211

The frequency output has a -3 dB frequency of about 10 MHz on the FIO lines. Accordingly, at high frequencies the output waveform will get less square and the amplitude will decrease.

The output does not necessarily start instantly, but rather has to wait for the internal clock to roll. For the Frequency Output mode, the start delay varies from 0.0 to TimerClockDivisor*256/TimerClockBase. For example, if TimerClockBase = 48 MHz and TimerClockDivisor = 256, the start delay will vary from 0 to 1.4 ms.

2.9.1.8 - Quadrature Input (Mode 8) [U6 Datasheet]

Requires 2 timer channels used in adjacent pairs (0/1 or 2/3). Even timers will be quadrature channel A, and odd timers will be quadrature channel B. Timer#Value passed has no effect. The U6 does 4x quadrature counting, and returns the current count as a signed 32-bit integer (2's complement). The same current count is returned on both even and odd timer value parameters.

Writing a value of zero to either or both timers performs a reset of both. After reset, a read of either timer value will return zero until a new quadrature count is detected. If a timer is reset and read in the same function call, the read returns the value just before the reset.

4X Counting

Quadrature mode uses the very common 4X counting method, which provides the highest resolution possible. That means you get a count for every edge (rising & falling) on both phases (A & B). Thus if you have an encoder that provides 32 PPR, and you rotate that encoder forward 1 turn, the timer Value register will be incremented by +128 counts.

Z-Phase Support

Quadrature mode supports Z-Phase. When enabled this feature will set the count to zero when the specified IO line sees a logic high.

Z-phase is controlled by the value written to the timer during initialization. To enable z-phase support set bit 15 to 1 and set bits 0 through 4 to the DIO number that Z is connected to. EG: for a Z-line on EIO3 set the timer value to 0x800B or 32779. This value should be sent to both the A and B timers.

Note that the LabJack will only check Z when it sees an edge on the A or B lines.

Z-phase support requires Firmware 1.14+.

2's Complement

Other timer modes return unsigned values, but this timer mode is unique in that it returns a signed value from -2147483648 to +2147483647. That is, a 32-bit 2's complement value. When you do a timer value read and get back a single float from the UD driver, the math is already done and you get back a value from -2147483648.0 to +2147483647.0, but when using the special channels 20x/23x/224 you get the LSW and MSW separately and have to do the math yourself. Search for 2's complement math for your particular programming language.

In a language such as C++, you start by doing using unsigned 32-bit variables & constants to compute Value = (MSW * 65536) + LSW. Then simply cast Value to a signed 32-bit integer.

In a language such as Java that does not support unsigned integers, do everything with signed 64-bit variables & constants. First calculate Value = (MSW * 65536) + LSW. If Value < 2147483648, you are done. If Value >= 2147483648, do ActualValue = -1 * (4294967296 - Value).

Edge Rate Limits

This edge-detecting timer mode requires processing resources as an interrupt is required to handle each edge. See more about edge rate limits in Section 2.9.2.

Can I Use a Simple Counter Instead?

If you need to track absolute position with changing direction (forward & reverse), you need to use the timer feature described above to decode quadrature signals, as the relationship between edges on the A & B phases indicates direction. However, if movement is only in one direction, or you always know the direction of movement, you can just connect one phase (A or B) to a hardware counter (2 available on the U3) or firmware counter (2 available on the U3). Hardware counters can handle much faster edge rates (up to 8 MHz per Appendix A). Firmware counters have the same edge rate limits as quadrature, but the firmware counter is counting 1/4th the edges of a quadrature timer and thus can handle a signal with 4x the frequency, and also is only using 1 timer rather than 2.

2.9.1.9 - Timer Stop Input (Mode 9) [U6 Datasheet]

Log in or register to post comments

This mode should only be assigned to Timer1 or Timer3. On every rising edge seen by the **external pin**, this mode increments a 16-bit register. When that register matches the specified timer value (stop count value), the adjacent even timer (Timer0 or Timer2) is stopped. The range for the stop count value is 1-65535. Generally, the signal applied to TimerOdd is from TimerEven, which is configured in some output timer mode. One place where this might be useful is for stepper motors, allowing control over a certain number of steps.

Note that the timer is counting from the external pin like other input timer modes, so you must connect something to the stop timer input pin. For example, if you are using Timer1 to stop Timer0 which is outputting pulses, you must connect a jumper from Timer0 to Timer1.

Once this timer reaches the specified stop count value, and stops the adjacent timer, the timers must be reconfigured to restart the output.

When TimerEven is stopped, it is still enabled but just not outputting anything. Thus rather than returning to whatever previous digital I/O state was on that terminal, it goes to the state “digital-input” (which has a 100 kΩ pull-up to 3.3 volts). That means the best results are generally obtained if the terminal used by TimerEven was initially configured as digital input (factory default), rather than output-high or output-low. This will result in negative going pulses, so if you need positive going pulses consider using a simple inverter IC such as the CD74ACT540E from TI.

The MSW of the read from this timer mode returns the number of edges counted, but does not increment past the stop count value. The LSW of the read returns edges waiting for.

Edge Rate Limits

This edge-detecting timer mode requires processing resources as an interrupt is required to handle each edge. See more about edge rate limits in Section 2.9.2.

2.9.1.10 - System Timer Low/High Read (Modes 10 & 11) [U6 Datasheet]

The LabJack U6 has a free-running internal 64-bit system timer with a frequency of 4 MHz. Timer modes 10 & 11 return the lower or upper 32-bits of this timer. An FIO line is allocated for these modes like normal, even though they are internal readings and do not require any external connections. This system timer cannot be reset, and is not affected by the timer clock.

If using both modes 10 & 11, read both in the same low-level command and read 10 before 11.

Mode 11, the upper 32 bits of the system timer, is not available for stream reads. Note that when streaming on the U6, the timing is known anyway (elapsed time = scan rate * scan number) and it does not make sense to stream the system timer modes 10 or 11.

2.9.1.11 - Period Measurement (16-Bit, Modes 12 & 13) [U6 Datasheet]

Similar to the 32-bit edge-to-edge timing modes described above (modes 2 & 3), except that hardware capture registers are used to record the edge times. This limits the times to 16-bit values, but is accurate to the resolution of the clock, and not subject to any errors due to firmware processing delays.

Note that the minimum measurable period is limited by the edge rate limit discussed in Section 2.9.2.

Edge Rate Limits

This edge-detecting timer mode requires processing resources as an interrupt is required to handle each edge. See more about edge rate limits in Section 2.9.2.

2.9.1.12 - Line-to-Line Measurement (Mode 14) [U6 Datasheet]

This timer mode requires firmware 1.16 or later.

Introduction:

The Line-to-Line timer mode uses two timers to measure the time between specified edges on two different lines. For instance, you can measure the time between a rising edge on Timer0 and a falling edge on Timer1. When the LabJack sees the specified edge on Timer0 it starts counting until it sees the specified edge on Timer1. High resolution up to 20.8ns can be achieved with this mode.

Line-to-Line mode is also available on timers 2 and 3.

Configuring:

To configure a LabJack for Line-to-Line mode set an even timer and the next (odd) timer to mode 14. The timer values determine the edge that the timer will respond to, 1 being rising, 0 being falling. So, if Timer0's value is 0 and Timer1's is 1 then the LabJack will measure the time between a falling edge on Timer0 to a rising edge on Timer1.

Readings:

Once configured the timer will return zero until both specified edges have been detected. The time difference in TimerClock periods is then returned by both timers until they are reset. Both timers will return the same reading, so it is only necessary to read one or the other. To convert to time, divide the value returned by the timer clock. This mode returns 16-bit values, so care should be taken to be sure that the specified condition does not exceed the maximum time. The maximum time can be calculated by 65535/TimerClock, and resolution is 1/TimerClock, so some examples are:

```
TimerClock,  MaxTime,  Resolution
    48 MHz,      1.3ms,     20.8ns
      1 MHz,       65ms,         1µs
    3906 Hz,      16.7s,      256µs
```

Resetting:

Once a measurement has been acquired the even timer needs to be reset before the LabJack will measure again. Values specified when resetting have no effect. Once reset the even timer will return zero until a new measurement has been completed. Resetting the odd timer is optional, if not reset it will continue to return the last measurement until a new one has been completed.

2.9.2 - Timer Operation/Performance Notes [U6 Datasheet]

Log in or register to post comments

Note that the specified timer clock frequency is the same for all timers. That is, TimerClockBase and TimerClockDivisor are singular values that apply to all timers. Modes 0, 1, 2, 3, 4, 7, 12, and 13, all are affected by the clock frequency, and thus the simultaneous use of these modes has limited flexibility. This is often not an issue for modes 2 and 3 since they use 32-bit registers.

The output timer modes (0, 1, and 7) are handled totally by hardware. Once started, no processing resources are used and other U6 operations do not affect the output.

The edge-detecting timer input modes do require U6 processing resources, as an interrupt is required to handle each edge. Timer modes 2, 3, 5, 6, 9, 12, and 13 must process every applicable edge (rising **or** falling). Timer modes 4 and 8 must process every edge (rising **and** falling). To avoid missing counts, keep the total number of processed edges (all timers) less than 30,000 per second. That means that in the case of a single timer, there should be no more than 1 edge per 33 µs. For multiple timers, all can process an edge simultaneously, but if for instance both timers get an edge at the same time, 66 µs should be allowed before any further edges are applied. If streaming is occurring at the same time, the maximum edge rate will be less (7,000 per second), and since each edge requires processing time the sustainable stream rates can also be reduced.