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

13.2 DIO Extended Features [T-Series Datasheet]

DIO Extended Features Overview

Basics: DIO Extended Features, commonly refered to as "DIO-EF", allow T-Series devices to measure and generate digital waveforms that are more advanced than logic high or logic low. They expose features such as PWM output for servo motor control, Quadrature input for incremental/quadrature encoders, and more.

Register Numbering: DIO-EFs are configured and used through the DIO#(0:22)_EF registers. The numbering of these registers corresponds with the DIO numbers documented in section 13.0 Digital I/O.

Configuration and how to use: The meanings of each of the DIO#_EF_CONFIG registers and DIO#_EF_READ registers changes depending on what DIO-EF index (DIO#_EF_INDEX) is configured, however the general configuration process is the same and is described below. It is helpful to think of DIO-EF features as "sub-systems" that need to be configured before they are started. Once they are started, they can be interacted with by reading the system state and updating system configurations.

DIO-EF System Configurations:

1. Select a feature and determine the number of required DIO lines using table 13.2-1 or 13.2-2.
2. Ensure that the DIO-EF is disabled by writing a 0 to the appropriate DIO#_EF_ENABLE register.
3. If required by the selected DIO-EF feature, configure the DIO-EF clock source.
4. Write the selected feature's index value to the appropriate DIO#_EF_INDEX register.
5. If required by the selected DIO-EF feature, write to the DIO#_EF_OPTIONS register.
6. If required by the selected DIO-EF feature, write to the DIO#_EF_CONFIG registers.
7. Enable the selected DIO-EF feature by writing a 1 to the appropriate DIO#_EF_ENABLE register.

Once a DIO-EF has been started, it can be interacted with using the following registers

- If the selected DIO-EF produces data, read the results from the DIO#_EF_READ registers.
  - *(E.g., if DIO6 is configured as an Interrupt Counter, you can read the current count from DIO6_EF_READ_A.)*
- If the selected DIO-EF can be updated on the fly, write to the DIO#_EF_CONFIG registers.
  - *(E.g., if DIO0 is configured as a PWM Out, you can update the duty cycle by writing to DIO0_EF_CONFIG_A.)*

Available DIO-EF Features: A brief overview of each of the features is as follows:

- **0:** PWM Out
- **1:** PWM Out with Phase
- **2:** Pulse Out
- **3,4:** Frequency In
- **5:** Pulse Width In
- **6:** Line-to-Line In

- **7:** High-Speed Counter
- **8:** Interrupt Counter
- **9:** Interrupt Counter with Debounce
- **10:** Quadrature In
- **11:** Interrupt Frequency In
- **12:** Conditional Reset

Kipling Walkthroughs: Kipling's Register Matrix can be used to perform DIO-EF features. Some examples:

- Configuring & Reading a Counter
- Configuring & Reading Frequency
- Configuring a PWM Output

DIO-EF Enable/Disable

This register is used to disable a DIO-EF feature (in order to configure it) and also used to start or enable the DIO-EF subsystem.

A DIO-EF doesn't always need to be disabled for it to be configured, depending on the DIO-EF being enabled.

Name		Start Address	Type	Access
DIO#(0:22)_EF_ENABLE	1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.	44000	UINT32	R/W

&print=true

DIO-EF Index (Feature Selection)

This register is used to select the extended feature that will get enabled on a given DIO line. The valid DIO lines differ by device. For more specific details look at reference tables 13.2-1 and 13.2-2 as well as the appropriate DIO-EF feature subsection.

Name		Start Address	Type	Access
DIO#(0:22)_EF_INDEX	An index to specify the feature you want.	44100	UINT32	R/W

&print=true

DIO-EF Options and Clock Source Selection

This register isn't used by all DIO-EF features.

If a DIO-EF feature requires the configuration or selection of a clock source (such as PWM Out does), the configuration of this register is required, since it is required for selecting a clock source. See 13.2.1 EF Clock Source for more details about clock source selection.

Name		Start Address	Type	Access
DIO#(0:22)_EF_OPTIONS	Function dependent on selected feature index.	44200	UINT32	R/W

&print=true

DIO-EF Configuration

Configuration registers serve two purposes. They provide a location for settings that need to be configured upon DIO-EF enable and they provide a location for settings that users may need to use to update a DIO-EF feature once it has been enabled.

Initial Configuration: Configuration is the initial setup of the Extended Feature. Configuration requires that any DIO-EF running at the pin in question first be disabled. Options can then be loaded. Then the DIO-EF can be enabled.

Update: Some DIO#_CONFIG registers can be updated while a DIO-EF is running. Updating allows the DIO-EF to change its operation parameters without restarting. Note that the clock source and feature index cannot be changed in an update. Depending on the feature, reads and writes to the update registers have small differences. See the Update portion of each feature for more information.

Name		Start Address	Type	Access
DIO#(0:22)_EF_CONFIG_A	Function dependent on selected feature index.	44300	UINT32	R/W
DIO#(0:22)_EF_CONFIG_B	Function dependent on selected feature index.	44400	UINT32	R/W
DIO#(0:22)_EF_CONFIG_C	Function dependent on selected feature index.	44500	UINT32	R/W
DIO#(0:22)_EF_CONFIG_D	Function dependent on selected feature index.	44600	UINT32	R/W

&print=true

DIO-EF Basic Read Registers

Some DIO-EF produce results or provide status information that can be read. This information is usually a binary integer. When possible, the T-series device will convert the binary integer into a real-world unit such as seconds. When available, converted values can be read from the registers designated with “_F”.

Name		Start Address	Type	Access

Name		Start Address	Type	Access
DIO#(0:21)_EF_READ_A	Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.	3000	UINT32	R
DIO#(0:21)_EF_READ_B	Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.	3200	UINT32	R
DIO#(0:21)_EF_READ_A_F	Reads a floating point value. The meaning of value is dependent on selected feature index.	3500	FLOAT32	R
DIO#(0:21)_EF_READ_B_F	Reads a floating point value. The meaning of value is dependent on selected feature index.	3700	FLOAT32	R

&print=true

DIO-EF Read-and-Reset Registers

Some DIO-EF can be reset while they are running. Resetting can have different results depending on the feature. For instance, counters are reset to zero.

Name		Start Address	Type	Access
DIO#(0:21)_EF_READ_A_AND_RESET	Reads the same value as DIO#(0:22)_EF_READ_A and forces a reset.	3100	UINT32	R
DIO#(0:21)_EF_READ_A_F_AND_RESET	Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.	3600	FLOAT32	R

&print=true

Streaming DIO-EF Results

Though all operations discussed in this section are supported in command-response mode, some DIO-EF features can be read fast enough to be streamed:

- Frequency In
- Pulse Width In
- High-Speed Counter
- Interrupt Counter
- Interrupt Counter with Debounce
- Quadrature In
- Interrupt Frequency In

In stream mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in 3.2 Stream, those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

Other Considerations

Specifications

See Appendix A-2 for specs including:

- Frequency Output
- Counter Input Frequency
- Minimum High & Low Time
- "Interrupt" Total Edge Rate

System Timer

Complications can occur if streaming while enabling a DIO-EF that requires the use of a system timer. Please contact LabJack support if you need to do this.

Available DIO-EF By Device/Reference Tables

T4

Table 13.2-1.T4 Digital I/O Extended Features

T4 Digital I/O Extended Features		AIN (0-3)				FIO (4-7)				EIO (0-7)								CIO (0-3)			
						DIO															
Feature	**Index#**					**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
PWM Out	**0**							✔	✔												
PWM Out with Phase	**1**							✔	✔												
Pulse Out	**2**							✔	✔												
Frequency In	**3,4**					✔	✔														
Pulse Width In	**5**					✔	✔														
Line-to-Line In*	**6**					✔	✔														
High-Speed Counter	**7**																	✔	✔	✔	✔
Interrupt Counter	**8**					✔	✔	✔	✔	✔	✔										
Interrupt Counter with Debounce	**9**					✔	✔	✔	✔	✔	✔										
Quadrature In*	**10**					✔	✔	✔	✔	✔	✔										
Interrupt Frequency In	**11**					✔	✔	✔	✔	✔	✔										
Conditional Reset	**12**					✔	✔	✔	✔	✔	✔										

T7

Table 13.2-2.T7 Digital I/O Extended Features

T7 Digital I/O Extended Features		FIO (0-7)								EIO (0-7)								CIO (0-3)				MIO (0-2)		
		DIO																						
Feature	**Index#**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**
PWM Out	**0**	✔		✔	✔	✔	✔																	
PWM Out with Phase	**1**	✔		✔	✔	✔	✔																	
Pulse Out	**2**	✔		✔	✔	✔	✔																	
Frequency In	**3,4**	✔	✔																					
Pulse Width In	**5**	✔	✔																					
Line-to-Line In*	**6**	✔	✔																					
High-Speed Counter	**7**																	✔	✔	✔	✔			
Interrupt Counter	**8**	✔	✔	✔	✔			✔	✔															
Interrupt Counter with Debounce	**9**	✔	✔	✔	✔			✔	✔															
Quadrature In*	**10**	✔	✔	✔	✔			✔	✔															
Interrupt Frequency In	**11**	✔	✔	✔	✔			✔	✔															
Conditional Reset	**12**	✔	✔	✔	✔			✔	✔															

* Line-to-Line In and Quadrature In both require two DIO lines.

13.2.1 EF Clock Source [T-Series Datasheet]

Overview

The clock source settings produce the reference frequencies used to generate output waveforms and measure input waveforms. They control output frequency, PWM resolution, maximum measurable period, and measurement resolution.

```
Clock#Frequency = CoreFrequency / DIO_EF_CLOCK#_DIVISOR    // typically 80M/Divisor
```

CoreFrequency is always 80 MHz at this time, but in the future some low-power operational modes might result in different core frequencies. The valid values for DIO_EF_CLOCK#_DIVISOR are 1, 2, 4, 8, 16, 32, 64, or 256, and a value of 0 (default) equates to a divisor of 1.

There are 3 DIO-EF clock sources available. Each clock source has an associated bit size and several mutual exclusions. Mutual exclusions exist because the clock sources share hardware with other features. A clock source is created from a hardware counter. CLOCK1 uses COUNTER_A (CIO0) and CLOCK2 uses COUNTER_B (CIO1). The 32-bit clock source (CLOCK0) is created by combining the 2 16-bit clock sources (CLOCK1 CLOCK2). The following list provides clock source bit sizes and mutual exclusions:

- CLOCK0: 32-bit. Mutual Exclusions: CLOCK1, CLOCK2, COUNTER_A (CIO0), COUNTER_B(CIO1)
- CLOCK1: 16-bit. Mutual Exclusions: CLOCK0, COUNTER_A (CIO0)
- CLOCK2: 16-bit. Mutual Exclusions: CLOCK0, COUNTER_B (CIO1)

The clock source is not a DIO-EF feature, but the four basic operations of Configure, Read, Update, and Reset still apply.

Configure

There are four registers associated with the configuration of clock sources:

Digital EF Clock Source

Name		Start Address	Type	Access
DIO_EF_CLOCK0_ENABLE	1 = enabled. 0 = disabled. Must be disabled during configuration.	44900	UINT16	R/W
DIO_EF_CLOCK0_DIVISOR	Divides the core clock. Valid options: 1,2,4,8,16,32,64,256.	44901	UINT16	R/W
DIO_EF_CLOCK0_OPTIONS	Bitmask: bit0: 1 = use external clock. All other bits reserved.	44902	UINT32	R/W
DIO_EF_CLOCK0_ROLL_VALUE	The clock will count to this value and then start over at zero. The clock pulses counted are those after the divisor. 0 results in the max roll value possible. This is a 32-bit value (0-4294967295) if using a 32-bit clock, and a 16-bit value (0-65535) if using a 16-bit clock.	44904	UINT32	R/W

&print=true

A clock source can be enabled after DIO#_EF_INDEX has been configured. This allows several DIO-EFs to be started at the same time.

DIO_EF_CLOCK0_ROLL_VALUE is a 32-bit value (0-4294967295) if using a 32-bit clock, and a 16-bit value (0-65535) if using a 16-bit clock. 0 results in the max roll value possible.

Read

To read the clock, read DIO_EF_CLOCK0_COUNT:

Digital EF Clock Source

Name		Start Address	Type	Access
DIO_EF_CLOCK0_COUNT	Current tick count of this clock. Will read between 0 and ROLL_VALUE-1.	44908	UINT32	R

&print=true

This can be useful for generating timestamps.

Update

Both the ROLL_VALUE and the DIVISOR can be written while a clock source is running. As long as the clock source's period is greater than 50 µs, the clock will seamlessly switch to the new settings.

Reset

At this time there are no reset operations available for the DIO-EF clock sources.

Example

Configure CLOCK0 as a 10 MHz clock source with a roll value of 1000000.

```
DIO_EF_CLOCK0_ENABLE = 0
DIO_EF_CLOCK0_DIVISOR = 8
DIO_EF_CLOCK0_ROLL_VALUE = 1000000
DIO_EF_CLOCK0_ENABLE = 1
```

With this clock configuration, PWM output (index=0) will have a frequency of 10 Hz. A frequency input measurement (index=3/4) will be able to count from 0-999999 with each count equal to 0.1 microseconds, and thus a max period of about 0.1 seconds.

Advanced

If CLOCK0 is enabled and CLOCK1 and CLOCK2 are disabled, you can still select CLOCK1 or CLOCK2 as the source for a DIO-EF channel. CLOCK1 and CLOCK2 are actually the LSW & MSW of CLOCK0. The frequency of CLOCK1 is the same as CLOCK0. If DIO_EF_CLOCK0_ROLL_VALUE is >= 2^16, then the frequency of CLOCK2 is CLOCK0_freq divided by the modulus (remainder portion) of CLOCK0_freq / 2^16. If (CLOCK0_ROLL_VALUE - 1) is < 2^16, then the frequency of CLOCK2 is 0. CLOCK1_ROLL_VALUE is the modulus of (CLOCK0_ROLL_VALUE - 1) / 2^16 and CLOCK2_ROLL_VALUE is the quotient (integer portion) of (CLOCK0_ROLL_VALUE - 1) / 2^16.

13.2.2 PWM Out [T-Series Datasheet]

Overview

> T4 Capable DIO: **DIO6, DIO7** (aka FIO6, FIO7)

> T7 Capable DIO: **DIO0, DIO2, DIO3, DIO4, DIO5** (aka FIO0, FIO2, FIO3, FIO4, FIO5)

Requires Clock Source: **Yes**

Index: **0**

Streamable: **No**

This PWM Out Extended Feature generates a pulse width modulated wave form.

Operation

PWM output will set the DIO high and low relative to the clock source's count. When the count is zero the DIO line will be set high. When the count matches Config A the line will be set low. Therefore Config A is used to control the duty cycle and the resolution is equal to the roll value.

```
Clock#Frequency = CoreFrequency / DIO_EF_CLOCK#_DIVISOR    // typically 80M/Divisor
PWMFrequency = Clock#Frequency / DIO_EF_CLOCK#_ROLL_VALUE
DutyCycle% = 100 * DIO#_EF_CONFIG_A / DIO_EF_CLOCK#_ROLL_VALUE
```

For the common case of CoreFrequency = 80 MHz we can rewrite output frequency as:

```
PWMFrequency = 80M / (DIO_EF_CLOCK#_DIVISOR * DIO_EF_CLOCK#_ROLL_VALUE)
```

... and for 50% duty-cycle (square wave) simply set:

```
DIO#_EF_CONFIG_A = DIO_EF_CLOCK#_ROLL_VALUE / 2
```

CoreFrequency is always 80 MHz at this time, but in the future some low-power operational modes might result in different core frequencies.

The valid values for DIO_EF_CLOCK#_DIVISOR are 1, 2, 4, 8, 16, 32, 64, or 256, and a value of 0 (default) equates to a divisor of 1. For more details about Clock#Frequency and DIO_EF_CLOCK#_DIVISOR, see the DIO-EF Clock Source section.

PWM Out is capable of glitch-free updates in most situations. A glitch-free update means that the PWM will finish the current period consisting of the high time then the low time before loading the new value. The next period will then have the new duty cycle. This is true for all cases except zero. When setting the duty cycle to zero, the line will be set low regardless of the current position. This means that a single high pulse with duration between zero and the previous high time can be output before the line goes low.

Configure

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 0
DIO#_EF_OPTIONS: Bits 0-2 specify which clock source to use ... 000 for Clock0, 001 for Clock1, and 010 for Clock2. All other bits are reserved and should be set to 0.
DIO#_EF_CONFIG_A: When the specified clocks source's count matches this value, the line will transition from high to low.
DIO#_EF_CONFIG_B: Not used.
DIO#_EF_CONFIG_C: Not used.
DIO#_EF_CONFIG_D: Not used.

Update

The duty cycle can be updated at any time. To update, write the new value to DIO#_EF_CONFIG_A. The new value will not be used until the clock source rolls to zero. This means that at the end of the current period, the new value will be loaded—resulting in a glitch-free transition.

Read

No information is returned by PWM Out.

Reset

Reset has no affect on this feature.

Example

To generate a 10 kHz PWM starting at 25% DC, first configure the clock source. The higher the roll value, the greater the duty cycle resolution will be. For the highest resolution, we want to maximize the roll value, so use the smallest clock divisor that will not result in a roll value greater than the clock source's maximum (32-bits or 16-bits). With a divisor of 1, the roll value will be 8000:

```
80 MHz / (1 * 8000) = 10 kHz
```

Now set the clock registers accordingly:

```
DIO_EF_CLOCK0_ENABLE = 0
DIO_EF_CLOCK0_DIVISOR = 1
DIO_EF_CLOCK0_ROLL_VALUE = 8000
DIO_EF_CLOCK0_ENABLE = 1
```

Once the clock source is configured, we can use the roll value to calculate CONFIG_A:

```
DC = 25% = 100 * CONFIG_A / 8000
```

...So CONFIG_A = 2000. Now the PWM can be turned on by writing the proper registers:

```
DIO0_EF_ENABLE = 0
DIO0_EF_INDEX = 0
DIO0_EF_CONFIG_A = 2000
DIO0_EF_ENABLE = 1
```

For a more detailed walkthrough, see Configuring a PWM Output.

T7 PWM Output Config

Unprintable Content

This page contains material that is only viewable online. https://labjack.com/support/.

Support - General Psuedocode Compiler

Support - T7 PWM Output Config Script

13.2.3 PWM Out with Phase [T-Series Datasheet]

Overview

> T4 Capable DIO: **DIO6, DIO7** (aka FIO6, FIO7)

> T7 Capable DIO: **DIO0, DIO2, DIO3, DIO4, DIO5** (aka FIO0, FIO2, FIO3, FIO4, FIO5)

Requires Clock Source: **Yes**

Index: **1**

Streamable: **No**

This PWM Out with Phase Extended Feature is similar to thePWM Out DIO-EF, but allows for phase control.

Operation

PWM Output with Phase control generates PWM waveforms with the pulse positioned at different points in the period. This is achieved by setting the DIO line high and low relative to the clock source's count.

```
Clock#Frequency = CoreFrequency / DIO_EF_CLOCK#_DIVISOR   // typically 80M/Divisor
PWMFrequency = Clock#Frequency / DIO_EF_CLOCK#_ROLL_VALUE
DutyCycle% = 100 * (DIO#_EF_CONFIG_A - DIO#_EF_CONFIG_B) / DIO_EF_CLOCK#_ROLL_VALUE
PhaseOffset = 360º * DIO#_EF_CONFIG_A / DIO_EF_CLOCK#_ROLL_VALUE
```

When the count matches CONFIG_B, the DIO line will be set high. When the count matches CONFIG_A, the line will be set low. Therefore CONFIG_B minus CONFIG_A controls the duty cycle.

CoreFrequency is always 80 MHz at this time, but in the future some low-power operational modes might result in different core frequencies. The valid values for DIO_EF_CLOCK#_DIVISOR are 1, 2, 4, 8, 16, 32, 64, or 256, and a value of 0 (default) equates to a divisor of 1. For more details about Clock#Frequency and DIO_EF_CLOCK#_DIVISOR, see the DIO-EF Clock Source section.

Configure

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 1
DIO#_EF_OPTIONS: Bits 0-2 specify which clock source to use ... 000 for Clock0, 001 for Clock1, and 010 for Clock2. All other bits reserved and should be set to 0.
DIO#_EF_CONFIG_A: When the clock source's count matches this value the line will transition from high to low.
DIO#_EF_CONFIG_B: When the clock source's count matches this value the line will transition from low to high.
DIO#_EF_CONFIG_C: Not used.
DIO#_EF_CONFIG_D: Not used.

Update

The duty cycle can be updated at any time. To update, write the new value to CONFIG_A then CONFIG_B. The value written to CONFIG_A is stored until CONFIG_B is written. After writing CONFIG_B, the new value will be loaded at the start of the next period. Updates are glitch-less unless switching from a very high to very low duty cycle or a very low to very high duty cycle.

DIO#_EF_CONFIG_A: Values written here will set the new falling position. The new value will not take effect until CONFIG_B is written.
DIO#_EF_CONFIG_B: Values written here will set the new rising position. When CONFIG_B is written, the new CONFIG_A is also loaded.

Read

No information is returned by PWM Out with Phase.

Reset

Reset has no affect on this feature.

Example

See 13.2.2 PWM Out for an example.

13.2.4 Pulse Out [T-Series Datasheet]

Overview

> T4 Capable DIO: **DIO6, DIO7** (aka FIO6, FIO7)

> T7 Capable DIO: **DIO0, DIO2, DIO3, DIO4, DIO5** (aka FIO0, FIO2, FIO3, FIO4, FIO5)

Requires Clock Source: **Yes**

Index: **2**

Streamable: **No**

Operation

Pulse output will generate a specified number of pulses. The high time and the low time are specified relative to the clock source the same way as PWM with Phase.

```
Clock#Frequency = CoreFrequency / DIO_EF_CLOCK#_DIVISOR     // typically 80M/Divisor
PulseOutFrequency = Clock#Frequency / DIO_EF_CLOCK#_ROLL_VALUE
DutyCycle% = 100 * (DIO#_EF_CONFIG_A - DIO#_EF_CONFIG_B) / DIO_EF_CLOCK#_ROLL_VALUE     // if A > B
```

For the common case of CoreFrequency = 80 MHz and CONFIG_B fixed at 0, we can rewrite these as:

```
PulseOutFrequency = 80M / (DIO_EF_CLOCK#_DIVISOR * DIO_EF_CLOCK#_ROLL_VALUE)
DutyCycle% = 100 * DIO#_EF_CONFIG_A / DIO_EF_CLOCK#_ROLL_VALUE
```

... and thus for 50% duty cycle simply set:

```
DIO#_EF_CONFIG_A = DIO_EF_CLOCK#_ROLL_VALUE / 2
```

CoreFrequency is always 80 MHz at this time, but in the future some low-power operational modes might result in different core frequencies. The valid values for DIO_EF_CLOCK#_DIVISOR are 1, 2, 4, 8, 16, 32, 64, or 256, and a value of 0 (default) equates to a divisor of 1. For more details about Clock#Frequency and DIO_EF_CLOCK#_DIVISOR, see the DIO-EF Clock Source section.

Configure

DIO#: First set the DIO line low (DIO#=0). The line must start low for proper pulse generation.
DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 2
DIO#_EF_OPTIONS: Bits 0-2 specify which clock source to use ... 000 for Clock0, 001 for Clock1, and 010 for Clock2. All other bits are reserved and should be set to 0.
DIO#_EF_CONFIG_A: When the specified, clock source's count matches this value the line will transition from high to low.
DIO#_EF_CONFIG_B: When the specified, clock source's count matches this value the line will transition from low to high.
DIO#_EF_CONFIG_C: The number of pulses to generate.
DIO#_EF_CONFIG_D: Not used.

Update

DIO#_EF_CONFIG_A: Sets a new high to low transition point. Will take effect when writing CONFIG_C.
DIO#_EF_CONFIG_B: Sets a new low to high transition point. Will take effect when writing CONFIG_C.
DIO#_EF_CONFIG_C: Writing to this value will start a new pulse sequence. If a sequence is already in progress it will be aborted. Numbers previously written to CONFIG_A or CONFIG_B will take effect when CONFIG_C is written.

Read

Results are read from the following registers.

DIO#_EF_READ_A: The number of pulses that have been completed.
DIO#_EF_READ_B: The target number of pulses.

Reset

DIO#_EF_READ_A_AND_RESET: Reads number of pulses that have been completed, then restarts the pulse sequence.

Example

First configure a clock source to drive the pulse generator. Assuming the core frequency is 80 MHz, writing the following registers will produce a 1 kHz pulse frequency.

```
DIO_EF_CLOCK0_DIVISOR = 8
DIO_EF_CLOCK0_ROLL_VALUE = 10000
DIO_EF_CLOCK0_ENABLE = 1
```

Thus the clock frequency is:

```
Clock0Frequency = 80 MHz / 8 = 10 MHz
```

and PWM frequency is:

```
PWMFrequency = 10 MHz / 10000 = 1 kHz
```

Now that we have a clock to work with, we can configure our pulse.

```
DIO0_EF_ENABLE = 0
DIO0 = 0                // set DIO0 to output-low
DIO0_EF_INDEX = 2       // pulse out type index
DIO0_EF_CONFIG_A = 2000   // high to low count
DIO0_EF_CONFIG_B = 0      // low to high count
DIO0_EF_CONFIG_C = 5000   // number of pulses
DIO0_EF_ENABLE = 1
```

Thus, the duty cycle is:

```
duty cycle = 100 * (2000 - 0) / 10000 = 20%
```

The LabJack will now output 5000 pulses over 5 seconds at 20% duty cycle.

13.2.5 Frequency In [T-Series Datasheet]

Overview

> T4 Capable DIO: **DIO4, DIO5** (aka FIO4, FIO5)

> T7 Capable DIO: **DIO0, DIO1** (aka FIO0, FIO1)

Requires Clock Source: **Yes**

Index: **3 (positive edges) or 4 (negative edges)**

Streamable: **Yes—integer READ registers only.**

Operation

Frequency In will measure the period/frequency of a digital input signal by counting the number of clock source ticks between two edges: rising-to-rising (index=3) or falling-to-falling (index=4). The number of ticks can be read from DIO#_EF_READ_A.

```
Clock#Frequency = CoreFrequency / DIO_EF_CLOCK#_DIVISOR     //typically 80M/Divisor
Period (s) = DIO#_EF_READ_A / Clock#Frequency
Frequency (Hz) = Clock#Frequency / DIO#_EF_READ_A
```

```
Resolution(s) = 1 / Clock#Frequency
Max Period(s) = DIO_EF_CLOCK#_ROLL_VALUE / Clock#Frequency
```

CoreFrequency is always 80 MHz at this time, but in the future some low-power operational modes might result in different core frequencies. The valid values for DIO_EF_CLOCK#_DIVISOR are 1, 2, 4, 8, 16, 32, 64, or 256, and a value of 0 (default) equates to a divisor of 1. For more details about Clock#Frequency and DIO_EF_CLOCK#_DIVISOR, see the DIO-EF Clock Source section.

Roll value for this feature would typically be left at the default of 0, which is the max value (2^32 for the 32-bit Clock0), but you might be forced to use a lower roll value due to another needed feature such as PWM Out.

A couple of typical scenarios with roll value = 0 and using the 32-bit clock (Clock0):

- Divisor = 1, Resolution = 12.5 nanoseconds, MaxPeriod = 53.7 seconds
- Divisor = 256, Resolution = 3.2 microseconds, MaxPeriod = 229 minutes

By default, Frequency In operates in one-shot mode where it will measure the frequency once after being enabled and a new measurement only once after each read of a READ_A register. The other option is continuous mode, where the frequency is constantly measured (every edge is processed) and READ registers return the most recent result. Running in continuous mode puts a greater load on the processor.

If you do another read before a new edge has occurred, you will get the same value as before. Some applications will want to use the read-and-reset option so that a value is only returned once and extra reads will return 0. (See Reset below.)

Configure

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 3 or 4
DIO#_EF_OPTIONS: Default = 0. Bits 0-2 specify which clock source to use ... 000 for Clock0, 001 for Clock1, and 010 for Clock2. All other bits reserved and should be set to 0.
DIO#_EF_CONFIG_A: Default = 0. Bit 1: 0 = one-shot, 1 = continuous. All other bits reserved.
DIO#_EF_CONFIG_B: Not used.
DIO#_EF_CONFIG_C: Not used.
DIO#_EF_CONFIG_D: Not used.

Update

No update operations can be performed on Frequency In.

Read

Results are read from the following registers.

DIO#_EF_READ_A: Returns the period in ticks. If a full period has not yet been observed this value will be zero.
DIO#_EF_READ_B: Returns the same value as READ_A.
DIO#_EF_READ_A_F: Returns the period in seconds. If a full period has not yet been observed this value will be zero.
DIO#_EF_READ_B_F: Returns the frequency in Hz. If a full period has not yet been observed this value will be zero.

Note that all "READ_B" registers are capture registers. All "READ_B" registers are only updated when any "READ_A" register is read. Thus it would be unusual to read any B registers without first reading at least one A

register.

Stream Read

All operations discussed in this section are supported in command-response mode. In stream mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in the Stream Section those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

Reset

DIO#_EF_READ_A_AND_RESET: Returns the same data as DIO#_EF_READ_A and then clears the result so that zero is returned by subsequent reads until another full period is measured (2 new edges).

Note that even in continuous mode, with reads happening faster than the signal frequency using a _RESET read will result in measurements of every other cycle not every cycle.

Example

Most applications can use default clock settings, so to configure frequency input on DIO0 you can simply write to 3 registers:

```
DIO_EF_CLOCK0_ENABLE = 1
DIO0_EF_ENABLE = 0
DIO0_EF_INDEX = 3
DIO0_EF_ENABLE = 1
```

Now you can read the period in seconds from a 4th register DIO0_EF_READ_A_F.

Roll value considerations:

Sometimes, other DIO-EF might interact with this feature. For example, roll value would usually be set to 0 to provide the maximum measurable period, but assume that we have to use 10000 because it is set to that for PWM output on another channel:

```
DIO_EF_CLOCK0_DIVISOR = 8          // Clock0Frequency = 80M/8 = 10 MHz.
DIO_EF_CLOCK0_ROLL_VALUE = 10000   // Roll value needed for PWM output on another channel.
DIO_EF_CLOCK0_ENABLE = 1
```

This clock configuration results in:

```
Resolution = 1 / 10M = 0.1 us
```

and

```
MaxPeriod = 10000 / 10M = 1 ms
```

For a more detailed walkthrough, see Configuring & Reading Frequency.

Rate Limits

T-Series IC Rate Limits

The maximum measurable frequency varies based on the one-shot setting, concurrent stream rate, and other DIO-EFs set to an input mode.

One-shot or Continuous: When one-shot is enabled, the T-series devices will take a single measurement, then wait for a READ_A register to be read before taking another measurement. This means that with one-shot, only a small fraction of the total periods of a signal are measured. One-shot allows for higher maximum measurable frequency than continuous does.

Stream: The stream process is the highest priority process on T-series devices. When stream needs the processor, all other operations are put on hold. That hold occurs more frequently at higher stream speeds. When a DIO-EF process has to wait, the max frequency that can be measured is reduced.

Multiple DIO-EFs: DIO-EFs on other different lines also require processor time. The amount of processor time required depends on the signal being processed and the DIO-EF's settings. The maximum frequencies in the below table give the total max frequency for all running DIO-EFs—divide the max frequencies below by the number of enabled DIO-EFs to get the max frequency for each individual DIO-EF.

Refer to the following table for maximum measurable frequencies in various combinations of stream and one-shot.

Index	One-shot or Continuous	Stream Rate	Max Frequency
3 or 4	Continuous	Stream not running	200 kHz
3 or 4	One-shot	Stream not running	750 kHz
3 or 4	Continuous	10 kHz	75 kHz
3 or 4	One-shot	10 kHz	750 kHz
3 or 4	Continuous	100 kHz	20 kHz
3 or 4	One-shot	100 kHz	250 kHz
5	Continuous	Stream not running	200 kHz
5	One-shot	Stream not running	750 kHz
5	Continuous	10 kHz	75 kHz
5	One-shot	10 kHz	750 kHz
5	Continuous	100 kHz	20 kHz
5	One-shot	100 kHz	250 kHz

13.2.6 Pulse Width In [T-Series Datasheet]

Overview

> T4 Capable DIO: **DIO4, DIO5** (aka FIO4, FIO5)

> T7 Capable DIO: **DIO0, DIO1** (aka FIO0, FIO1)

Requires Clock Source: **Yes**

Index: **5**

Streamable: **Yes—integer READ registers only.**

Operation

Pulse Width In will measure the high time and low time of a digital input signal, by counting the number of clock source ticks while the signal is high and low. This could also be referred to as duty-cycle input or PWM input. The number of high ticks can be read from DIO#_EF_READ_A and the number of low ticks can be read from DIO#_EF_READ_B.

```
Clock#Frequency = CoreFrequency / DIO_EF_CLOCK#_DIVISOR    //typically 80M/Divisor
HighTime(s) = DIO#_EF_READ_A / Clock#Frequency
LowTime(s) = DIO#_EF_READ_B / Clock#Frequency
Resolution(s) = 1 / Clock#Frequency
Max High or Low Time(s) = DIO_EF_CLOCK#_ROLL_VALUE / Clock#Frequency
```

CoreFrequency is always 80 MHz at this time, but in the future some low-power operational modes might result in different core frequencies. The valid values for DIO_EF_CLOCK#_DIVISOR are 1, 2, 4, 8, 16, 32, 64, or 256, and a value of 0 (default) equates to a divisor of 1. For more details about Clock#Frequency and DIO_EF_CLOCK#_DIVISOR, see the DIO-EF Clock Source section.

Roll value for this feature would typically be left at the default of 0, which is the max value (2^32 for the 32-bit Clock0), but you might be using a lower roll value for another feature such as PWM Out.

A couple typical scenarios with roll value = 0 and using the 32-bit clock (Clock0):

- Divisor = 1, Resolution = 12.5 nanoseconds, MaxTime = 53.7 seconds
- Divisor = 256, Resolution = 3.2 microseconds, MaxTime = 229 minutes

Once this feature is enabled, a new measurement happens on every applicable edge and both result registers are updated on every rising edge. If you do another read before a new rising edge has occurred, you will get the same values as before. Many applications will want to use the read-and-reset option so that a value is only read once and extra reads will return 0. (See Reset below.)

Configure

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 5
DIO#_EF_OPTIONS: Bits 0-2 specify which clock source to use ... 000 for Clock0, 001 for Clock1, and 010 for Clock2. All other bits reserved and should be set to 0.
DIO#_EF_CONFIG_A: Bit 1: 1=continuous, 0=OneShot. All other bits reserved.
DIO#_EF_CONFIG_B: Not used.
DIO#_EF_CONFIG_C: Not used.
DIO#_EF_CONFIG_D: Not used.

Update

No update operations can be performed on Pulse Width In.

Read

Results are read from the following registers.

DIO#_EF_READ_A: Returns the measured high time in clock source ticks and saves the low time so that it can be read later. If a full period has not yet been observed this value will be zero.
DIO#_EF_READ_B: Returns the measured low time in clock source ticks. This is a capture register ... it is only updated when one of the READ_A registers is read.
DIO#_EF_READ_A_F: Returns the measured high time in seconds and saves the low time so that it can be read later. If a full period has not yet been observed this value will be zero.
DIO#_EF_READ_B_F: Returns the measured low time in seconds. This is a capture register ... it is only updated when one of the READ_A registers is read.

Only reading DIO#_EF_READ_A or DIO#_EF_READ_A_F triggers a new measurement.

Stream Read

All operations discussed in this section are supported in command-response mode. In stream mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in the Stream Section those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

Reset

DIO#_EF_READ_A_AND_RESET: Performs the same read as READ_A, but then also clears the register so that zero is returned until another full period is measured.

DIO#_EF_READ_A_F_AND_RESET: Performs the same read as READ_A_F, but then also clears the register so that zero is returned until another full period is measured.

Example

First, configure the clock source. Roll value would usually be set to 0 to provide the maximum measurable period, but assume for this example that we have to use 10000 because of PWM output on another channel:

```
DIO_EF_CLOCK0_DIVISOR = 8    // Clock0Frequency = 80M/8 = 10 MHz
DIO_EF_CLOCK0_ROLL_VALUE = 10000
DIO_EF_CLOCK0_ENABLE = 1
```

This clock configuration results in:

```
Resolution = 1 / 10M = 0.1 us
```

and

```
MaxPeriod = 10000 / 10M = 1 ms
```

Now configure the DIO_EF on DIO0 as pulse width input.

```
DIO0_EF_ENABLE = 0
DIO0_EF_INDEX = 5    // Pulse width input feature.
DIO0_EF_OPTIONS = 0  // Set to use clock source zero.
DIO0_EF_ENABLE = 1   // Enable the DIO_EF
```

After a full period (rising edge, falling edge, rising edge) has occurred, the values are stored in the result registers. This repeats at each rising edge. READ_A and READ_A_F both return the high time and save the low

time that can be read from READ_B and READ_B_F. This ensures that both readings are from the same waveform cycle.

Rate Limits

T-Series IC Rate Limits

The maximum measurable frequency varies based on the one-shot setting, concurrent stream rate, and other DIO-EFs set to an input mode.

One-shot or Continuous: When one-shot is enabled, the T-series devices will take a single measurement, then wait for a READ_A register to be read before taking another measurement. This means that with one-shot, only a small fraction of the total periods of a signal are measured. One-shot allows for higher maximum measurable frequency than continuous does.

Stream: The stream process is the highest priority process on T-series devices. When stream needs the processor, all other operations are put on hold. That hold occurs more frequently at higher stream speeds. When a DIO-EF process has to wait, the max frequency that can be measured is reduced.

Multiple DIO-EFs: DIO-EFs on other different lines also require processor time. The amount of processor time required depends on the signal being processed and the DIO-EF's settings. The maximum frequencies in the below table give the total max frequency for all running DIO-EFs—divide the max frequencies below by the number of enabled DIO-EFs to get the max frequency for each individual DIO-EF.

Refer to the following table for maximum measurable frequencies in various combinations of stream and one-shot.

Index	One-shot or Continuous	Stream Rate	Max Frequency
3 or 4	Continuous	Stream not running	200 kHz
3 or 4	One-shot	Stream not running	750 kHz
3 or 4	Continuous	10 kHz	75 kHz
3 or 4	One-shot	10 kHz	750 kHz
3 or 4	Continuous	100 kHz	20 kHz
3 or 4	One-shot	100 kHz	250 kHz
5	Continuous	Stream not running	200 kHz
5	One-shot	Stream not running	750 kHz
5	Continuous	10 kHz	75 kHz
5	One-shot	10 kHz	750 kHz
5	Continuous	100 kHz	20 kHz
5	One-shot	100 kHz	250 kHz

13.2.7 Line-to-Line In [T-Series Datasheet]

Overview

> T4 Capable DIO: **DIO4, DIO5** (aka FIO4, FIO5)

> T7 Capable DIO: **DIO0, DIO1** (aka FIO0, FIO1)

Requires Clock Source: **Yes**

Index: **6**

Streamable: **No**

Operation

Line-to-Line In measures the time between an edge on one DIO line and an edge on another DIO line by counting the number of clock source ticks between the two edges. The edges can be individually specified as rising or falling.

```
Clock#Frequency = CoreFrequency / DIO_EF_CLOCK#_DIVISOR    //typically 80M/Divisor
Time(s) = DIO#_EF_READ_A / Clock#Frequency
Resolution(s) = 1 / Clock#Frequency
Max Time(s) = DIO_EF_CLOCK#_ROLL_VALUE / Clock#Frequency
```

CoreFrequency is always 80 MHz at this time, but in the future some low-power operational modes might result in different core frequencies. The valid values for DIO_EF_CLOCK#_DIVISOR are 1, 2, 4, 8, 16, 32, 64, or 256, and a value of 0 (default) equates to a divisor of 1. For more details about Clock#Frequency and DIO_EF_CLOCK#_DIVISOR, see the DIO-EF Clock Source section.

Roll value for this feature would typically be left at the default of 0, which is the max value (2^32 for the 32-bit Clock0), but you might be using a lower roll value for another feature such as PWM Out.

A couple typical scenarios with roll value = 0 and using the 32-bit clock (Clock0):

- Divisor = 1, Resolution = 12.5 nanoseconds, MaxPeriod = 53.7 seconds
- Divisor = 256, Resolution = 3.2 microseconds, MaxPeriod = 229 minutes

Line-to-Line In operates in a one-shot mode. Once the specified combination of edges is observed, the data is saved and measuring stops. Another measurement can be started by resetting or performing the configuration procedure again.

Configure

Configuring Line-to-Line In requires configuring two digital I/O lines (DIO0 and DIO1 only) as Line-to-Line In feature index 6. The first DIO configured should be the one expecting the first edge. Any extended features on either DIO should be disabled before beginning configuration.

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 6
DIO#_EF_OPTIONS: Bits 0-2 specify which clock source to use ... 000 for Clock0, 001 for Clock1, and 010 for Clock2. All other bits are reserved and should be set to 0.
DIO#_EF_CONFIG_A: 0 = falling edge. 1 = rising edge.

DIO#_EF_CONFIG_B: Not used.
DIO#_EF_CONFIG_C: Not used.
DIO#_EF_CONFIG_D: Not used.

Update

No update operations can be performed on Line-to-Line In.

Read

Results are read from the following registers.

DIO#_EF_READ_A: Returns the one-shot measured time in clock source ticks. If the specified combination of edges has not yet been observed this value will be zero. DIO0 & DIO1 both return the same value.
DIO#_EF_READ_A_F: Returns the time in seconds.

Reset

DIO#_EF_READ_A_AND_RESET: Performs the same operation as DIO#_EF_READ_A, then clears the result and starts another measurement.

Example

First, configure the clock source:

```
DIO_EF_CLOCK0_DIVISOR = 1     // Clock0Frequency = 80M/1 = 80 MHz
DIO_EF_CLOCK0_ROLL_VALUE = 0
DIO_EF_CLOCK0_ENABLE = 1
```

This clock configuration results in:

```
Resolution = 1 / 80M = 12.5 ns
```

and

```
MaxPeriod = 2^32 / 80M = 53.7 seconds
```

Now configure the DIO-EF on DIO0 and DIO1 as line-to-line:

```
DIO0_EF_ENABLE = 0
DIO1_EF_ENABLE = 0

DIO0_EF_INDEX = 6     // Index for line-to-line feature.
DIO0_EF_OPTIONS = 0   // Select the clock source.
DIO0_EF_CONFIG_A = 0  // Detect falling edge.
DIO0_EF_ENABLE = 1    // Turn on the DIO-EF

DIO1_EF_INDEX = 6     // Index for line-to-line feature.
DIO1_EF_OPTIONS = 0   // Select the clock source.
DIO1_EF_CONFIG_A = 0  // Detect falling edge.
DIO1_EF_ENABLE = 1    // Turn on the DIO-EF
```

At this point the device is watching DIO0 for a falling edge. Once that happens it watches for a falling edge on DIO1. Once that happens it stores the time between those 2 edges, which you can read from the READ registers described above.

To do another measurement, repeat the DIO0-EF configuration above, or read from DIO0_EF_READ_A_AND_RESET.

Rate Limits

Line-to-line can achieve high resolution while requiring very little processor time. The time between the edges can be as little as 50 ns. Once a measurement has been completed the system will not measure again until reconfigured or reset. Unless you are reading at a high rate, line-to-line will have little impact on other systems.

13.2.8 High-Speed Counter [T-Series Datasheet]

Overview

Capable DIO: **DIO16, DIO17, DIO18, DIO19** (aka CIO0, CIO1, CIO2, CIO3)

Requires Clock Source: **No**

Index: **7**

Streamable: **Yes—integer READ registers only.**

T-series devices support up to 4 high-speed rising-edge counters that use hardware to achieve high count rates. These counters are shared with other resources as follows:

- CounterA (DIO16/CIO0): Used by EF Clock0 & Clock1.
- CounterB (DIO17/CIO1): Used by EF Clock0 & Clock2.
- CounterC (DIO18/CIO2): Always available.
- CounterD (DIO19/CIO3): Used by stream mode.

Configure

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 7
DIO#_EF_OPTIONS: Not used.
DIO#_EF_CONFIG_A: Not used.
DIO#_EF_CONFIG_B: Not used.
DIO#_EF_CONFIG_B: Not used.
DIO#_EF_CONFIG_B: Not used.

Update

No update operations can be performed with High-Speed Counter.

Read

Results are read from the following register.

DIO#_EF_READ_A: Returns the current count which is incremented on each rising edge.

Stream Read

All operations discussed in this section are supported in command-response mode. In stream mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in the Stream Section those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

Reset

DIO#_EF_READ_A_AND_RESET: Reads the current count then clears the counter. There is a brief period of time between reading and clearing during which edges can be missed. During normal operation this time period is 10-30 µs. If missed edges at this point are not acceptable, then do not use reset but rather just note the "virtual reset" counter value in software and subtract it from other values.

Frequency Measurement

Counters are often used to measure frequency by taking change in count over change in time:

```
Frequency = (CurrentCount - PreviousCount) / (CurrentTimestamp - PreviousTimestamp)
```

Typically the timestamps are from the host clock (software), but for more accurate timestamps read the CORE_TIMER register (address=61520, UINT32) in the same Modbus packet as the READ registers. CORE_TIMER is a 32-bit system timer running at 1/2 the core speed, and thus is normally 80M/2 => 40 MHz.

Also note that other digital extended features are available to measure frequency by timing individual pulses rather than counting over time.

Example

Enable CounterC on DIO18/CIO2:

```
DIO18_EF_ENABLE = 0
DIO18_EF_INDEX = 7
DIO18_EF_ENABLE = 1
```

Results can be read from the READ registers defined above.

Edge Rate Limits

See Appendix A-2.

13.2.9 Interrupt Counter [T-Series Datasheet]

Overview

> T4 Capable DIO: **DIO4, DIO5, DIO6, DIO7, DIO8, DIO9** (aka FIO4, FIO5, FIO6, FIO7, EIO0, EIO1)

> T7 Capable DIO: **DIO0, DIO1, DIO2, DIO3, DIO6, DIO7** (aka FIO0, FIO1, FIO2, FIO3, FIO6, FIO7)

Requires Clock Source: **No**

Index: **8**

Streamable: **Yes—integer READ registers only.**

Interrupt Counter counts the rising edge of pulses on the associated IO line. This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. See the discussion of edge rate limits at the bottom of this page.

Configure

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 8
DIO#_EF_OPTIONS: Not used.
DIO#_EF_CONFIG_A: Not used.
DIO#_EF_CONFIG_B: Not used.
DIO#_EF_CONFIG_B: Not used.
DIO#_EF_CONFIG_B: Not used.

There are 3 basic techniques for device configuration:

1. Power-up defaults. Most registers related to I/O configuration are part of the IO Config system where their startup values can be defined by the user. This is easily done with the Power-up Defaults tab in Kipling.

2. Real time software configuration. Software can write any needed configuration values at the beginning of execution, but you might have to consider how to handle if the device later reboots during software execution. An advantage to this method is that a factory device will work out-of-the-box without requiring the user to first configure the device.

3. Startup script. A Lua script can be set to run at startup and can write any values to any registers.

Update

No update operations can be performed on Interrupt Counter.

Read

Results are read from the following register.

DIO#_EF_READ_A: Returns the current Count

Stream Read

All operations discussed in this section are supported in command-response mode. In stream mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in the Stream Section those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

Reset

DIO#_EF_READ_A_AND_RESET: Reads the current count then clears the counter. Note that there is a brief period of time between reading and clearing during which edges can be missed. During normal operation this time period is 10-30 µs. If missed edges at this point can not be tolerated then reset should not be used.

Frequency Measurement

Counters are often used to measure frequency by taking change in count over change in time:

```
Frequency = (CurrentCount - PreviousCount) / (CurrentTimestamp - PreviousTimestamp)
```

Typically the timestamps are from the host clock (software), but for more accurate timestamps read the CORE_TIMER register (address=61520, UINT32) in the same Modbus packet as the READ registers. CORE_TIMER is a 32-bit system timer running at 1/2 the core speed, and thus is normally 80M/2 => 40 MHz.

Also note that other digital extended features are available to measure frequency by timing individual pulses rather than counting over time.

Example

Enable a counter on DIO0:

```
DIO0_EF_ENABLE = 0
DIO0_EF_INDEX = 8
DIO0_EF_ENABLE = 1
```

Use the Register Matrix in Kipling to write those 2 registers above, and also add DIO0_EF_READ_A so you see its value. Now take a wire connected to a GND terminal and tap that wire to the inside-back of the DIO0 screw-terminal (labeled "FIO0"). DIO0_EF_READ_A should increment by 1 or more counts each time you tap the ground wire in DIO0.

As another test you can set DAC1_FREQUENCY_OUT_ENABLE = 1 to enable the 10 Hz test signal (T7 requires firmware 1.0234+). Jumper DAC1 to DIO0 and you should see DIO0_EF_READ_A increment by about 10 counts per second.

For a more detailed walkthrough, see Configuring & Reading a Counter.

Edge Rate Limits

This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. This makes it substantially slower than other DIO-EF that are purely hardware-based. To avoid missed edges, the aggregate limit for edges seen by all interrupt-based DIO-EF is 70k edges/second. If stream mode is active, the limit is reduced to 20k edges/second. Excessive processor loading (e.g. a busy Lua script) can also reduce these limits. Note that interrupt features must process all edges, rising & falling, even if a given feature is configured to only look at one or the other.

The more proper way to think of the edge limit, and understand error that could be introduced when using multiple interrupt-based DIO-EF, is to consider that the interrupt that processes an edge can take up to 14 µs to complete. When a particular channel sees an applicable edge, an IF (interrupt flag) is set for that channel that tells the processor it needs to run an ISR (interrupt service routine) for that channel. Once an ISR is started, it runs to completion and no other ISR can run until it is done (except that stream interrupts are higher priority and will preempt other interrupts). When an ISR completes, it clears the IF for that channel. So it is okay to have edges on multiple channels at the same time, as long as there is not another edge on any of those channels before enough time to process all the initial edges.

Say that channel A & B have an edge occur at the same time and an ISR starts to process the edge on channel A. If channel A has another edge during the first 14 µs, that edge will be lost. If channel B has another edge during the first 14 µs, the initial edge will be lost. If channel B has another edge during the second 14 µs (during the ISR for channel B), the new edge will be lost.

13.2.10 Interrupt Counter with Debounce [T-Series Datasheet]

Overview

> T4 Capable DIO: **DIO4, DIO5, DIO6, DIO7, DIO8, DIO9** (aka FIO4, FIO5, FIO6, FIO7, EIO0, EIO1)

> T7 Capable DIO: **DIO0, DIO1, DIO2, DIO3, DIO6, DIO7** (aka FIO0, FIO1, FIO2, FIO3, FIO6, FIO7)

Requires Clock Source: **No**

Index: **9**

Streamable: **Yes—integer READ registers only.**

Interrupt Counter with Debounce will increment its count by 1 when it receives a rising edge, a falling edge, or any edge (2x counting). After seeing an applicable edge, any further edges will be ignored during the debounce time.

This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. See the discussion of edge rate limits at the bottom of this page.

Debounce Modes (DIO#_EF_CONFIG_B)

The exact behavior of the counting/debouncing is controlled by an index value written to DIO#_EF_CONFIG_B.

- 0: Count falling, debounce all, self-restarting timeout.
- 1: Count rising, debounce all, self-restarting timeout.

- 2: Count & debounce all, self-restarting timeout.
- 3: Count & debounce falling, fixed timeout.
- 4: Count & debounce rising, fixed timeout.
- 5: Timeout starts on falling edge. During timeout, a rising edge cancels and a falling edge restarts the timeout. After timeout any edge causes a count.
- 6: Timeout starts on rising edge. During timeout, a falling edge cancels and a rising edge restarts the timeout. After timeout any edge causes a count.

Self-restarting timeout means that during timeout any edge will restart the timeout with the value specified with DIO#_EF_CONFIG_A.

Mode 0 is commonly used with a normally-open push-button switch that is actuated by a person. We only want to count the push (falling edge), but expect bounce on the push & release (falling & rising edges) so need to debounce both.

Mode 4 might be used with some sort of device that outputs a fixed length positive pulse. For example, say a device provides a 1000 µs pulse, and there is always at least 5000 µs between pulses. Set the debounce timeout to 2000 µs so that the timeout period safely covers the entire pulse, but the timeout will for sure be done before another pulse can occur.

Modes 5 & 6 implement a requirement that the state of the line must remain low or high for some amount of time. For example, if you use mode 5 with a push-button switch and set DIO#_EF_CONFIG_A = 50000, that means that someone must push the switch and hold it down solidly for at least 50ms, and then the count will occur when they release the switch. An advantage to these modes is that they will ignore brief transient signals.

Configure

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 9
DIO#_EF_OPTIONS: Not used.
DIO#_EF_CONFIG_A: Debounce timeout in microseconds (µs, 0-1000000).
DIO#_EF_CONFIG_B: Debounce mode index.
DIO#_EF_CONFIG_B: Not used.
DIO#_EF_CONFIG_B: Not used.

Update

No update operations can be performed on Interrupt Counter with Debounce.

Read

Results are read from the following register.

DIO#_EF_READ_A: Returns the current Count

Stream Read

All operations discussed in this section are supported in command-response mode. In stream mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in the Stream Section those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

Reset

DIO#_EF_READ_A_AND_RESET: Reads the current count then clears the counter. Note that there is a brief period of time between reading and clearing during which edges can be missed. During normal operation this time period is 10-30 µs. If missed edges at this point can not be tolerated then reset should not be used.

Example

Enable a debounce counter on DIO0:

```
DIO0_EF_ENABLE = 0
DIO0_EF_INDEX = 9
DIO0_EF_CONFIG_A = 20000   // 20 ms debounce time
DIO0_EF_CONFIG_B = 0       // count falling, debounce all, self-restarting timeout
DIO0_EF_ENABLE = 1
```

Results can be read from the READ registers defined above.

Edge Rate Limits

This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. This makes it substantially slower than other DIO-EF that are purely hardware-based. To avoid missed edges, the aggregate limit for edges seen by all interrupt-based DIO-EF is 70k edges/second. If stream mode is active, the limit is reduced to 20k edges/second. Excessive processor loading (e.g. a busy Lua script) can also reduce these limits. Note that interrupt features must process all edges, rising & falling, even if a given feature is configured to only look at one or the other.

The more proper way to think of the edge limit, and understand error that could be introduced when using multiple interrupt-based DIO-EF, is to consider that the interrupt that processes an edge can take up to 14 µs to complete. When a particular channel sees an applicable edge, an IF (interrupt flag) is set for that channel that tells the processor it needs to run an ISR (interrupt service routine) for that channel. Once an ISR is started, it runs to completion and no other ISR can run until it is done (except that stream interrupts are higher priority and will preempt other interrupts). When an ISR completes, it clears the IF for that channel. So it is okay to have edges on multiple channels at the same time, as long as there is not another edge on any of those channels before enough time to process all the initial edges.

Say that channel A & B have an edge occur at the same time and an ISR starts to process the edge on channel A. If channel A has another edge during the first 14 µs, that edge will be lost. If channel B has another edge during the first 14 µs, the initial edge will be lost. If channel B has another edge during the second 14 µs (during the ISR for channel B), the new edge will be lost.

13.2.11 Quadrature In [T-Series Datasheet]

Overview

> T4 Capable DIO: **DIO4, DIO5, DIO6, DIO7, DIO8, DIO9** (aka FIO4, FIO5, FIO6, FIO7, EIO0, EIO1)

> T7 Capable DIO: **DIO0, DIO1, DIO2, DIO3, DIO6, DIO7** (aka FIO0, FIO1, FIO2, FIO3, FIO6, FIO7)

Requires Clock Source: **No**

Index: **10**

Streamable: **Yes—integer READ registers only.**

Quadrature input uses two DIOs to track a quadrature signal. Quadrature is a directional count often used in rotary encoders when you need to keep track of absolute position with forward & reverse movement. If you have movement in only one direction, or another way to know direction, you can simply use a normal counter with one phase of the encoder output.

T-series devices uses 4x quadrature decoding, meaning that every edge observed (rising & falling on both phases) will increment or decrement the count.

This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. See the discussion of edge rate limits at the bottom of this page.

Quadrature is prone to error if the edge rate is exceeded. This is particularly likely during direction change where the time between edges can be very small. Errors can occur when two edges come in too quickly for the device to process, which can result in missed counts or missed change in direction. These errors will be recorded and the quantity encountered can be read. If three edges come in too quickly an undetectable error can occur.

Some quadrature encoders also include a third output channel, called a zero (Z-phase) or index or reference signal, which supplies a single pulse per revolution. This single pulse is used for absolute determination of position. T-series devices support resets according to this reference signal. Z-phase will reset the count when a high state is detected on the specified DIO line at the same time any edge occurs on A or B phases. If the reference pulse is wider than A/B pulses, consider using the Conditional Reset feature instead of this Z-phase support. If the reference pulse is only high in between A/B edges, consider some sort of RC circuit to elongate it or consider using the Conditional Reset feature. If set to one-shot mode, Z-phase will clear the count only once and must be "re-armed" by disabling/re-enabling the feature or a read from DIO#_EF_READ_A_AND_RESET.

Configure

Two DIO must be configured for Quadrature In. The two lines must be an adjacent even/odd pair:

> T4: DIO4/5, DIO6/7, and DIO8/9 are valid pairs.

> T7: DIO0/1, DIO2/3, and DIO6/7 are valid pairs.

The even IO line will be phase A and the odd will be phase B.

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 10
DIO#_EF_OPTIONS: Not used.
DIO#_EF_CONFIG_A: Z-phase control: 0 = Z-phase off, 1 = Z-phase on, 3 = Z-phase on in one shot mode.
DIO#_EF_CONFIG_B: Z-phase DIO number.
DIO#_EF_CONFIG_C: Not used.
DIO#_EF_CONFIG_D: Not used.

Update

No update operations can be performed with Quadrature In.

Read

The quadrature count and error count are available from the even channel. The odd channel will return zeros.

Results are read from the following registers.

DIO#_EF_READ_A - Returns the current count as a signed 2's complement value.
DIO#_EF_READ_B – Returns the number of detected errors.
DIO#_EF_READ_A_F - Returns the count in a single precision float (float32).

Only reading DIO#_EF_READ_A or DIO#_EF_READ_A_F triggers a new measurement.

Stream Read

All operations discussed in this section are supported in command-response mode. In stream mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in the Stream Section those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

Reset

DIO#_EF_READ_A_AND_RESET – Performs the same operation as DIO#_EF_READ_A, then sets the count to zero.

Example

Configure DIO6 and DIO7 as quadrature inputs:

```
DIO6_EF_ENABLE = 0    //Make sure DIO-EF is disabled on DIO6
DIO7_EF_ENABLE = 0    //Make sure DIO-EF is disabled on DIO7

DIO6_EF_INDEX = 10    //Set feature index for DIO6 to quadrature.
DIO7_EF_INDEX = 10    //Set feature index for DIO7 to quadrature.

DIO6_EF_ENABLE = 1    //Enable quadrature DIO-EF on DIO6, for A phase.
DIO7_EF_ENABLE = 1    //Enable quadrature DIO-EF on DIO7, for B phase.
```

Edges on the two lines will now be decoded and the count will be incremented or decremented according to the edge sequence.

The current count can be read from DIO6_EF_READ_A or DIO6_EF_READ_A_AND_RESET.

To enable z-phase connected to DIO5 you would do:

```
DIO6_EF_CONFIG_A=1
DIO6_EF_CONFIG_B=5
DIO7_EF_CONFIG_A=1
DIO7_EF_CONFIG_B=5
```

Testing

On some LabJack devices you can physically tap a GND wire into the two DIO channels to cause some counts, but that does not work on T-series devices. Testing needs to be done with a proper quadrature signal.

If testing with an actual encoder, start with DIO-EF enabled and simply watch (e.g. in Kipling) the digital inputs as you slowly turn the encoder to see if the inputs change between high and low. That confirms a valid electrical connection.

You can test using 2 digital inputs to create a quadrature sequence. For example, configure quadrature on DIO6/7 as shown above, and connect DIO0 to DIO6 and DIO1 to DIO7.

```
DIO0 = 1    //Initialize DIO0 to output-high.
DIO1 = 1    //Initialize DIO1 to output-high.

DIO6_EF_ENABLE = 0    //Make sure DIO-EF is disabled on DIO6
DIO7_EF_ENABLE = 0    //Make sure DIO-EF is disabled on DIO7
DIO6_EF_INDEX = 10    //Set feature index for DIO6 to quadrature.
DIO7_EF_INDEX = 10    //Set feature index for DIO7 to quadrature.
DIO6_EF_ENABLE = 1    //Enable quadrature DIO-EF on DIO6, for A phase.
DIO7_EF_ENABLE = 1    //Enable quadrature DIO-EF on DIO7, for B phase.
```

Now we can simulate a quadrature signal by toggling DIO0 and DIO1—aka FIO0 and FIO1—in the proper sequence. We will use the FIO_STATE register (address=2500) which operates on all 8 FIO bits at once, but we will set the inhibit bits for FIO2-FIO7 (bits 10-15) so they are not affected. To set bits 10-15 we can simply add 64512 to the desired FIO0/1 state value:

```
Write FIO_STATE = 3 + 64512, then should read DIO6_EF_READ_A =  0
Write FIO_STATE = 1 + 64512, then should read DIO6_EF_READ_A =  1
Write FIO_STATE = 0 + 64512, then should read DIO6_EF_READ_A =  0
Write FIO_STATE = 2 + 64512, then should read DIO6_EF_READ_A = -1
Write FIO_STATE = 3 + 64512, then should read DIO6_EF_READ_A = -2
Write FIO_STATE = 2 + 64512, then should read DIO6_EF_READ_A = -1
Write FIO_STATE = 0 + 64512, then should read DIO6_EF_READ_A =  0
Write FIO_STATE = 1 + 64512, then should read DIO6_EF_READ_A =  1
Write FIO_STATE = 3 + 64512, then should read DIO6_EF_READ_A =  2
Write FIO_STATE = 2 + 64512, then should read DIO6_EF_READ_A =  3
Write FIO_STATE = 0 + 64512, then should read DIO6_EF_READ_A =  4
Write FIO_STATE = 1 + 64512, then should read DIO6_EF_READ_A =  5
Write FIO_STATE = 3 + 64512, then should read DIO6_EF_READ_A =  6
Write FIO_STATE = 2 + 64512, then should read DIO6_EF_READ_A =  7
```

Quadrature Decoding or Simple Counting?

Quadrature decoding is only needed when you need to keep track of absolute position with changes in direction included.

If you want to track absolute position but the direction does not change, or for whatever reason you always know the direction of movement, then you can just count the pulses from one phase (A or B) using a simple counter. The interrupt counters available on 6 of the FIO0 lines have the same 70k edge rate limit discussed below, but

they only do 1x counting and only use 1 timer. The high-speed counters on the 4 CIO lines can handle up to 5 MHz per counter.

If you are just trying to measure frequency, you can use a counter as described above and note the change in count over some time interval, or you can use a DIO-EF that measures the time of individual pulses. Either way, just one phase (A or B) is needed.

Edge Rate Limits

Keep in mind that T-series devices do 4x quadrature counting. For example, a 100 pulses/revolution encoder connected to a pair of DIO will generate 400 edges/revolution.

This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. This makes it substantially slower than other DIO-EF that are purely hardware-based. To avoid missed edges, the aggregate limit for edges seen by all interrupt-based DIO-EF is 70k edges/second. If stream mode is active, the limit is reduced to 20k edges/second. Excessive processor loading (e.g. a busy Lua script) can also reduce these limits. Note that interrupt features must process all edges, rising & falling, even if a given feature is configured to only look at one or the other.

The more proper way to think of the edge limit, and understand error that could be introduced when using multiple interrupt-based DIO-EF, is to consider that the interrupt that processes an edge can take up to 14 µs to complete. When a particular channel sees an applicable edge, an IF (interrupt flag) is set for that channel that tells the processor it needs to run an ISR (interrupt service routine) for that channel. Once an ISR is started, it runs to completion and no other ISR can run until it is done (except that stream interrupts are higher priority and will preempt other interrupts). When an ISR completes, it clears the IF for that channel. So it is okay to have edges on multiple channels at the same time, as long as there is not another edge on any of those channels before enough time to process all the initial edges.

Say that channel A & B have an edge occur at the same time and an ISR starts to process the edge on channel A. If channel A has another edge during the first 14 µs, that edge will be lost. If channel B has another edge during the first 14 µs, the initial edge will be lost. If channel B has another edge during the second 14 µs (during the ISR for channel B), the new edge will be lost.

For faster quadrature tracking, one option is to use a chip such as theLS7366R-S from US Digital and then use the SPI ability of the T-series device to talk to that chip. In fact, a Lua script can be used to handle the SPI communication with the chip and periodically put the current count in a user-ram register than can be easily read by any host application or Modbus client. If you don't find an example for the LS7366 ask us about it.

13.2.12 Interrupt Frequency In [T-Series Datasheet]

Overview

> T4 Capable DIO: **DIO4, DIO5, DIO6, DIO7, DIO8, DIO9** (aka FIO4, FIO5, FIO6, FIO7, EIO0, EIO1)

> T7 Capable DIO: **DIO0, DIO1, DIO2, DIO3, DIO6, DIO7** (aka FIO0, FIO1, FIO2, FIO3, FIO6, FIO7)

Requires Clock Source: **No. Uses core clock / 2.**

Index: **11**

Streamable: **Yes—integer READ registers only.**

Interrupt Frequency In will measure the frequency of a signal on the associated DIO line.

This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. See the discussion of edge rate limits at the bottom of this page.

To measure the frequency, the T-series device will measure the duration of one or more periods. There are several option available to control the way the LabJack does this. The number of periods to be averaged, the edge direction to trigger on, and whether to measure continuously or in one-shot mode can all be specified.

The clock source for this feature is simply half the core frequency:

```
ClockFrequency = CoreFrequency / 2  //Typically 80M/2 = 40 MHz
Period(s) = DIO#_EF_READ_A / ClockFrequency
Frequency (Hz) = ClockFrequency / DIO#_EF_READ_A
```

The maximum measurable time is 107 s. The number of periods to be averaged multiplied by the maximum expected period must be less than 107 s or the result will overflow:

```
107 > (NumToAverage * MaxPeriod)
```

By default, Interrupt Frequency In operates in one-shot mode where it will measure the frequency once after being enabled and a new measurement only once after each read. The other option is continuous mode, where the frequency is constantly measured (every edge is processed) and READ registers return the most recent result. Running in continuous mode puts a greater load on the processor.

Configure

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 11
DIO#_EF_OPTIONS: Not Used.
DIO#_EF_CONFIG_A: Default = 0. Bit 0: Edge select; 0 = falling, 1 = rising. Bit 1: 0 = one-shot, 1 = continuous.
DIO#_EF_CONFIG_B: Default = 0 which is equates to 1. Number of periods to be measured and averaged.
DIO#_EF_CONFIG_C: Not used.
DIO#_EF_CONFIG_D: Not used.

Update

No update operations can be performed with Interrupt Frequency In.

Read

Results are read from the following registers.

DIO#_EF_READ_A: Returns the average period per cycle in ticks (core clock ticks / 2).
DIO#_EF_READ_B: Returns the total core clock tick count.
DIO#_EF_READ_A_F: Returns the average period per cycle in seconds. Takes into account the number of periods to be averaged and the core clock speed.
DIO#_EF_READ_B_F: Returns the average frequency in Hz. Takes into account the number of periods to be averaged and the core clock speed.

Note that all "READ_B" registers are capture registers. All "READ_B" registers are only updated when any "READ_A" register is read. Thus it would be unusual to read any B registers without first reading at least one A register.

Stream Read

All operations discussed in this section are supported in command-response mode. In stream mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in the Stream Section those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

Reset

DIO#_EF_READ_A_AND_RESET: Returns the same data as DIO#_EF_READ_A and then clears the result so that zero is returned by subsequent reads until another full period is measured (2 new edges).
DIO#_EF_READ_A_AND_RESET_F: Returns the same data as DIO#_EF_READ_A_F and then clears the result so that zero is returned by subsequent reads until another full period is measured (2 new edges).

Note that even in continuous mode, with reads happening faster than the signal frequency using a _RESET read will result in measurements of every other cycle not every cycle.

Example

To configure Interrupt Frequency In on DIO6 you can simply write to 2 registers:

```
DIO6_EF_ENABLE = 0
DIO6_EF_INDEX = 11
DIO6_EF_ENABLE = 1
```

Now you can read the period in seconds from a 4th register DIO6_EF_READ_A_F.

Edge Rate Limits

This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. This makes it substantially slower than other DIO-EF that are purely hardware-based. To avoid missed edges, the aggregate limit for edges seen by all interrupt-based DIO-EF is 70k edges/second. If stream mode is active, the limit is reduced to 20k edges/second. Excessive processor loading (e.g. a busy Lua script) can also reduce these limits. Note that interrupt features must process all edges, rising & falling, even if a given feature is configured to only look at one or the other.

The more proper way to think of the edge limit, and understand error that could be introduced when using multiple interrupt-based DIO-EF, is to consider that the interrupt that processes an edge can take up to 14 µs to complete. When a particular channel sees an applicable edge, an IF (interrupt flag) is set for that channel that tells the processor it needs to run an ISR (interrupt service routine) for that channel. Once an ISR is started, it runs to completion and no other ISR can run until it is done (except that stream interrupts are higher priority and

will preempt other interrupts). When an ISR completes, it clears the IF for that channel. So it is okay to have edges on multiple channels at the same time, as long as there is not another edge on any of those channels before enough time to process all the initial edges.

Say that channel A & B have an edge occur at the same time and an ISR starts to process the edge on channel A. If channel A has another edge during the first 14 µs, that edge will be lost. If channel B has another edge during the first 14 µs, the initial edge will be lost. If channel B has another edge during the second 14 µs (during the ISR for channel B), the new edge will be lost.

13.2.13 Conditional Reset [T-Series Datasheet]

Overview

> T4 Capable DIO: **DIO4, DIO5, DIO6, DIO7, DIO8, DIO9** (aka FIO4, FIO5, FIO6, FIO7, EIO0, EIO1)

> T7 Capable DIO: **DIO0, DIO1, DIO2, DIO3, DIO6, DIO7** (aka FIO0, FIO1, FIO2, FIO3, FIO6, FIO7)

Requires Clock Source: **No**

Index: **12**

Streamable: **No**

DIO-EF Conditional Reset will reset a specified DIO-EF after a specified number of edges have been detected.

Configure

To set up a DIO-EF Conditional Reset is simple. Just set the DIO number of the DIO-EF you would like to reset and then set the other options. More options are likely to be added, so be sure to leave unused bits cleared to zero.

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 12
DIO#_EF_OPTIONS: Not used.
DIO#_EF_CONFIG_A: Reset Options bitmask:

- bit 0: Edge select. 1 = rising, 0 = falling
- bit 1: reserved
- bit 2: OneShot. 1 = only reset once. 0 = reset every n edges.

DIO#_EF_CONFIG_B: Number of edges per reset.
DIO#_EF_CONFIG_C: IO number of DIO-EF to be reset.

DIO#_EF_CONFIG_D: Not used.

Update

No update operations can be performed on Conditional Reset.

Read

Results are read from the following registers.

DIO#_EF_READ_A – Returns the current count.

Example

This example assumes that DIO0 has a running extended feature such as quadrature or a counter. Now we will set up DIO2 as a falling edge trigger that will reset the count of DIO0_EF.

```
DIO2_EF_ENABLE = 0    // Ensure that the DIO-EF is not running so that it can be configured.
DIO2_EF_INDEX = 12    // Set to Conditional Reset
DIO2_EF_CONFIG_A = 0  // Falling edges
DIO2_EF_CONFIG_B = 1  // Reset every edges
DIO2_EF_CONFIG_C = 0  // Reset events clear the count of DIO0_EF
DIO2_EF_ENABLE = 1    // Turn on the DIO-EF
```

Now falling edges on DIO2 will set the count of DIO0_EF to zero.

For a more detailed walkthrough, see Configuring & Reading a Counter.

Edge Rate Limits

This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. This makes it substantially slower than other DIO-EF that are purely hardware-based. To avoid missed edges, the aggregate limit for edges seen by all interrupt-based DIO-EF is 70k edges/second. If stream mode is active, the limit is reduced to 20k edges/second. Excessive processor loading (e.g. a busy Lua script) can also reduce these limits. Note that interrupt features must process all edges, rising & falling, even if a given feature is configured to only look at one or the other.

The more proper way to think of the edge limit, and understand error that could be introduced when using multiple interrupt-based DIO-EF, is to consider that the interrupt that processes an edge can take up to 14 µs to complete. When a particular channel sees an applicable edge, an IF (interrupt flag) is set for that channel that tells the processor it needs to run an ISR (interrupt service routine) for that channel. Once an ISR is started, it runs to completion and no other ISR can run until it is done (except that stream interrupts are higher priority and will preempt other interrupts). When an ISR completes, it clears the IF for that channel. So it is okay to have edges on multiple channels at the same time, as long as there is not another edge on any of those channels before enough time to process all the initial edges.

Say that channel A & B have an edge occur at the same time and an ISR starts to process the edge on channel A. If channel A has another edge during the first 14 µs, that edge will be lost. If channel B has another edge during the first 14 µs, the initial edge will be lost. If channel B has another edge during the second 14 µs (during the ISR for channel B), the new edge will be lost.