DATA TRANSLATION®
A MEASUREMENT COMPUTING COMPANY

UM-26089-C

DT7816 File I/O Programming Manual

Trademark and Copyright Information

Notice

Measurement Computing Corporation does not authorize any Measurement Computing Corporation product for use in life support systems and/or devices without prior written consent from Measurement Computing Corporation. Life support devices/systems are devices or systems that, a) are intended for surgical implantation into the body, or b) support or sustain life and whose failure to perform can be reasonably expected to result in injury. Measurement Computing Corporation products are not designed with the components required, and are not subject to the testing required to ensure a level of reliability suitable for the treatment and diagnosis of people.

Table of Contents

About this Manual

This manual describes the file I/O commands that are supported by the DT7816 module in Linux user space.

Intended Audience

This document is intended for experienced Linux programmers.

What You Should Learn from this Manual

This manual provides detailed information about the file I/O operations that are supported for the DT7816 module. This manual is organized as follows:

- Chapter 1, "Overview," provides an overview of the file I/O support for the DT7816 module.
- Chapter 2, "Using the File I/O Commands," describes the I/O operations that are supported by the DT98337 module and the file I/O commands that are used to perform these operations.
- Chapter 3, "File I/O Command Reference," provides in-depth information about each of the file I/O commands that are supported by the DT7816 module, including the command syntax, arguments, and examples.
- Chapter 4, "Programming Flowcharts Using the File I/O Commands," provides a flowchart showing the order in which to use the file I/O commands to perform the supported I/O functions of the DT7816 module.
- Chapter 5, "Product Support," provides information about obtaining technical support.
- An index completes this manual.

Conventions Used in this Manual

The following conventions are used in this manual:

- Notes provide useful information or information that requires special emphasis, cautions provide information to help you avoid losing data or damaging your equipment, and warnings provide information to help you avoid catastrophic damage to yourself or your equipment.
- Command syntax is shown in `courier` font.

Related Documents

Refer to the following documents for more information:

- *DT7816 Getting Started* help file on our website (http://www.datatranslation.us/Products/ARM-Data-Acquisition/DT7816)
- *DT7816 User's Manual* (UM-26000). This manual describes the operation of the DT7816 module, as well calibration instructions, operating specifications, and pin assignments.
- Linux documentation on the AIO model (such as *//code.google.com/p/kernel/wiki/AIOUserGuide*) and on Linux file I/O commands.

Where to Get Help

Should you run into problems installing or using the DT7816 module, the Data Translation Technical Support Department is available to provide technical assistance. Refer to Chapter 5 for more information. If you are outside the United States or Canada, call your local distributor, whose number is listed on Data Translation's web site (www.mccdaq.com).

Overview

Introduction

The Linux operating system provides a powerful and flexible infrastructure for developing embedded software for the DT7816. Because it is supported by an open source community, developers can decrease development time by leveraging source code that is freely available, and reduce costs by taking advantage of software that does not require licensing or distribution fees.

Figure 1 shows how the software support is organized for the DT7816 module.

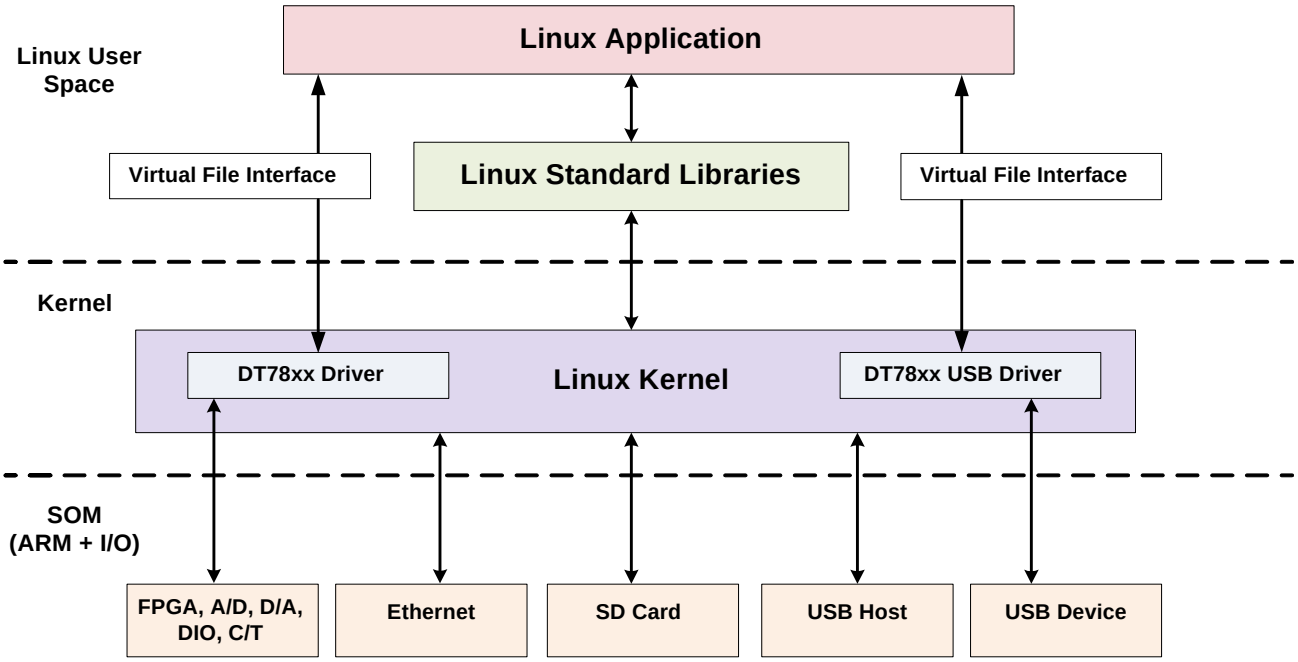


Figure 1: DT7816 Software Support

The device drivers (DT78xx driver and DT78xx USB driver), which are loaded when the module is powered up, expose the functionality of the module to Linux user space applications using virtual file interfaces, such as **/dev/DT7816-ain** for the analog input subsystem and**/dev/DT7816-ep1out** for a USB gadget OUT endpoint.

This manual describes the virtual files and the file I/O commands that are supported by the DT78xx device driver and DT78xx USB gadget driver. Using these files and file I/O commands, you can write applications in Linux user space that communicate with the subsystems on the DT7816 module to perform I/O operations.

Installing the Software

Refer to the *DT7816 Getting Started* help file on the Data Translation website for detailed information on installing and setting up a host system for use in developing application programs for the DT7816 module.

Examples

Once you have installed the DT7816 software to your Linux host computer, you can access the DT7816 example programs, which are located in the following subdirectory under the TI SDK directory: **example-applications/dt78xx-examples/**

The example programs demonstrate the capabilities of the hardware and illustrate how user programs communicate with the DT7816 drivers using file I/O commands and IOCTLS. All examples are written in ANSI C and are open source.

Each example program is located in a separate subdirectory. Each example directory includes a README.txt file that provides information about the example and a makefile, which allows you to build the example using the TI SDK toolchain and cross compiler.

You can build these example programs, use them to test the hardware, and modify them as needed to get up and running quickly. It is recommended that you refer to these examples when learning about the file I/O commands.

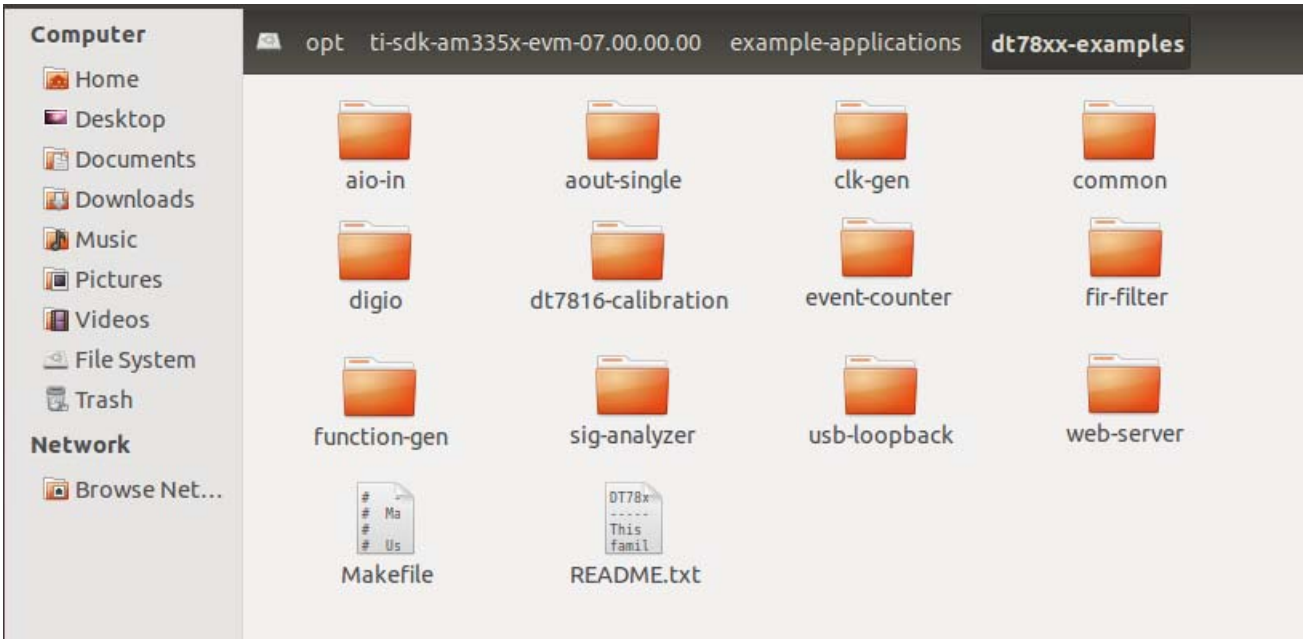


Figure 2: DT7816 Example Programs

The example programs are summarized in Table 1.

Table 1: Description of the Examples

Board Features	Example Directory	Description
Analog Input/ Input Stream	aio-in	Performs an asynchronous analog input operation and stores the data to a file.
	fir-filter	Performs an input stream and an output stream operation simultaneously and continuously. Data from the input stream is filtered and then output from the analog output channel.
	sig-analyzer	An embedded web server and signal analyzer. It acquires data from two analog input channels and performs and FFT on the data. The results are displayed to a client's web browser when connected to the device.
	web-server	An embedded web server that performs most of the functions of the DT7816 module and saves the acquired data to a file.
Analog output/ Output stream	aout-single	Performs a synchronous write operation on the analog output channel.
	fir-filter	Performs an input stream and an output stream operation simultaneously and continuously. Data from the input stream is filtered and then output from the analog output channel.
	function-gen	Performs an asynchronous analog output operation, generating one of the following waveforms using Direct Digital Synthesis: sine, triangle, sawtooth, or square. This example also outputs a pulse waveform on several bits of the digital output port.
Analog Input and Analog Output Calibration	DT7816-calibration	A command-line program that calibrates the analog input and analog output circuitry of the DT7816 module.
Digital I/O	digio	Performs a synchronous write operation on the digital output port, and then reads back the value that was output by performing a synchronous read operation on the digital input port.
	web-server	An embedded web server that performs most of the functions of the DT7816 module and saves the acquired data to a file.
	function-gen	Performs an asynchronous analog output operation, generating one of the following waveforms using Direct Digital Synthesis: sine, triangle, sawtooth, or square. This example also outputs a pulse waveform on several bits of the digital output port.

Table 1: Description of the Examples (cont.)

Board Features	Example Directory	Description
Counter/timer	clk-gen	Uses the counter/timer to generate an output clock.
	event-counter	Performs an event counting operation.
	web-server	An embedded web server that performs most of the functions of the DT7816 module and saves the acquired data to a file.
USB	usb-loopback	Demonstrates use of the gadget USB driver to send data from a host application to the DT7816 using and OUT USB pipe and to receive data from the DT7816 using IN USB pipe.
Misc	common	Includes common functions used by the other example programs.

Note: Linux is an open-source development environment. As such, our example programs may use code from other vendors. This code is for demonstration purposes only. If you want to use this code for commercial purposes, you must ensure that you resolve any licensing issues with the appropriate parties.

Refer to the *DT7816 Getting Started* help file on the Data Translation website for more information about building these example programs.

Summary of Supported File I/O Operations

Table 2 lists the supported file I/O operations for the DT7816 module.

Table 2: Summary of File I/O Operations Supported on the DT7816 Module

Subsystem	Virtual File	FIle I/O Command	Description
Analog Input	/dev/DT7816-ain	**open**	Opens the subsystem.
		close	Closes the subsystem.
		ioctl - IOCTL_AIN_CFG_SET	Configures the parameters of the analog input subsystem.
		ioctl - IOCTL_AIN_CFG_GET	Returns the configuration of the analog input subsystem.
		ioctl - IOCTL_GAIN_POT_SET	Sets the gain potentiometer for a specified analog input channel and calibration register.
		ioctl - IOCTL_GAIN_POT_GET	Returns the configuration of the gain potentiometer for a specified analog input channel and calibration register.
		ioctl - IOCTL_GAIN_POT_WIPER_SET	Sets the wiper value for the gain potentiometer associated with a specified analog input channel.
		ioctl - IOCTL_GAIN_POT_WIPER_GET	Returns the configuration of the wiper value for the gain potentiometer associated with a specified analog input channel.
		ioctl - IOCTL_LED_SET	Turns the user LEDs on the module either on or off.
		ioctl - IOCTL_LED_GET	Returns the state of the user LEDs on the module.

Table 2: Summary of File I/O Operations Supported on the DT7816 Module (cont.)

Subsystem	Virtual File	FIle I/O Command	Description
Input Stream	/dev/DT7816-stream-in	**open**	Opens the input stream.
		close	Closes the input stream.
		ioctl - IOCTL_SAMPLE_CLK_SET	Configures the sample clock for the input stream.
		ioctl - IOCTL_SAMPLE_CLK_GET	Returns the configuration of the sample clock for the input stream.
		ioctl - IOCTL_START_TRIG_CFG_SET	Configures the start trigger for the input stream.
		ioctl - IOCTL_START_TRIG_CFG_GET	Returns the configured start trigger for the input stream.
		ioctl - IOCTL_CHAN_MASK_SET	Configures the channels from which to acquire data.
		ioctl - IOCTL_CHAN_MASK_GET	Returns the channels that are enabled for acquisition.
		ioctl - IOCTL_ARM_SUBSYS	Arms the module to detect a trigger for the input stream.
		ioctl - IOCTL_START_SUBSYS	Starts continuous acquisition (an asynchronous operation) when a software trigger is specified.
		ioctl - IOCTL_ACQ_STATUS_GET	Returns the status of acquisition (armed or triggered).
		ioctl - IOCTL_STOP_SUBSYS	Stops continuous acquisition.
		io_setup	Opens an asynchronous I/O context.
		io_submit	Submits I/O requests.
		io_getevents	Processes events when I/O requests are completed.
		io_cancel	Cancels I/O requests.
		io_destroy	Destroys an asynchronous I/O context.
		ioctl - IOCTL_LED_SET	Turns the user LEDs on the module either on or off.
		ioctl - IOCTL_LED_GET	Returns the state of the user LEDs on the module.

Table 2: Summary of File I/O Operations Supported on the DT7816 Module (cont.)

Subsystem	Virtual File	FIle I/O Command	Description
Analog Output	/dev/DT7816-aout	**open**	Opens the subsystem.
		close	Closes the subsystem.
		ioctl - IOCTL_GAIN_POT_SET	Sets the gain potentiometer for a specified analog output channel and calibration register.
		ioctl - IOCTL_GAIN_POT_GET	Returns the configuration of the gain potentiometer for a specified analog output channel and calibration register.
		ioctl - IOCTL_GAIN_POT_WIPER_SET	Sets the wiper value for the gain potentiometer associated with a specified analog output channel.
		ioctl - IOCTL_GAIN_POT_WIPER_GET	Returns the configuration of the wiper value for the gain potentiometer associated with a specified analog output channel.
		ioctl - IOCTL_OFFSET_POT_SET	Sets the offset potentiometer for a specified analog output channel and calibration register.
		ioctl - IOCTL_OFFSET_POT_GET	Returns the configuration of the offset potentiometer for a specified analog output channel and calibration register.
		ioctl - IOCTL_OFFSET_POT_WIPER_SET	Sets the wiper value for the offset potentiometer associated with a specified analog output channel.
		ioctl - IOCTL_OFFSET_POT_WIPER_GET	Returns the configuration of the wiper value for the offset potentiometer associated with a specified analog output channel.
		ioctl - IOCTL_LED_SET	Turns the user LEDs on the module either on or off.
		ioctl - IOCTL_LED_GET	Returns the state of the user LEDs on the module.

Table 2: Summary of File I/O Operations Supported on the DT7816 Module (cont.)

Subsystem	Virtual File	FIle I/O Command	Description
Output Stream	/dev/DT7816-stream-out	**open**	Opens the input stream.
		close	Closes the input stream.
		ioctl - IOCTL_SAMPLE_CLK_SET	Configures the sample clock for the output stream.
		ioctl - IOCTL_SAMPLE_CLK_GET	Returns the configuration of the sample clock for the output stream.
		ioctl - IOCTL_START_TRIG_CFG_SET	Configures the start trigger for the output stream.
		ioctl - IOCTL_START_TRIG_CFG_GET	Returns the configured start trigger for the output stream.
		ioctl - IOCTL_CHAN_MASK_SET	Configures the channels that you want to update in the output stream.
		ioctl - IOCTL_CHAN_MASK_GET	Returns the output channels that are enabled in the analog output stream.
		ioctl - IOCTL_ARM_SUBSYS	Arms the module to detect a trigger for the output stream.
		ioctl - IOCTL_START_SUBSYS	Starts a continuous output operation (an asynchronous operation) when a software trigger is specified.
		ioctl - IOCTL_ACQ_STATUS_GET	Returns the status of the output operation (armed or triggered).
		ioctl - IOCTL_STOP_SUBSYS	Stops the output operation.
		io_setup	Opens an asynchronous I/O context.
		io_submit	Submits I/O requests.
		io_getevents	Processes events when I/O requests are completed.
		io_cancel	Cancels I/O requests.
		io_destroy	Destroys an asynchronous I/O context.
		ioctl - IOCTL_LED_SET	Turns the user LEDs on the module either on or off.
		ioctl - IOCTL_LED_GET	Returns the state of the user LEDs on the module.
Digital Input	/dev/DT7816-din	**open**	Opens the subsystem.
		close	Closes the subsystem.
		read	Synchronously reads a single value from the digital input port.

Table 2: Summary of File I/O Operations Supported on the DT7816 Module (cont.)

Subsystem	Virtual File	FIle I/O Command	Description
Digital Input (cont.)	/dev/DT7816-din	**ioctl - IOCTL_LED_SET**	Turns the user LEDs on the module either on or off.
		ioctl - IOCTL_LED_GET	Returns the state of the user LEDs on the module.
Digital Output	/dev/DT7816-dout	**open**	Opens the subsystem.
		close	Closes the subsystem.
		write	Synchronously writes a single value to the digital output port.
		ioctl - IOCTL_LED_SET	Turns the user LEDs on the module either on or off.
		ioctl - IOCTL_LED_GET	Returns the state of the user LEDs on the module.
Counter/ Timer	/dev/DT7816-ctr-tmr	**open**	Opens the subsystem.
		close	Closes the subsystem.
		ioclt - IOCTL_CT_CFG_SET	Configures the parameters of the counter/timer subsystem.
		ioclt - IOCTL_CT_CFG_GET	Returns the configuration of the counter/timer subsystem.
		ioctl - IOCTL_START_SUBSYS	Starts the counter/timer operation.
		ioctl - IOCTL_STOP_SUBSYS	Stops the counter/timer operation.
		read	Synchronously reads a single value from the counter/timer.
		ioctl - IOCTL_LED_SET	Turns the user LEDs on the module either on or off.
		ioctl - IOCTL_LED_GET	Returns the state of the user LEDs on the module.
Measure Counter	/dev/DT7816-measure	**open**	Opens the subsystem.
		close	Closes the subsystem.
		ioctl - IOCTL_MCTR_CFG_SET	Configures the parameters of the measure counter subsystem.
		ioctl - IOCTL_MCTR_CFG_GET	Returns the configuration of the measure counter subsystem.
		ioctl - IOCTL_LED_SET	Turns the user LEDs on the module either on or off.
		ioctl - IOCTL_LED_GET	Returns the state of the user LEDs on the module.

Table 2: Summary of File I/O Operations Supported on the DT7816 Module (cont.)

Subsystem	Virtual File	FIle I/O Command	Description
Tachometer	/dev/DT7816-tach	**open**	Opens the subsystem.
		close	Closes the subsystem.
		ioctl - IOCTL_TACH_CFG_SET	Configures the parameters of the tachometer subsystem.
		ioctl - IOCTL_TACH_CFG_GET	Returns the configuration of the tachometer subsystem.
		ioctl - IOCTL_LED_SET	Turns the user LEDs on the module either on or off.
		ioctl - IOCTL_LED_GET	Returns the state of the user LEDs on the module.
Gadget USB IN endpoints	/dev/DT7816-ep1in /dev/DT7816-ep2in	**open**	Opens the IN file.
		close	Closes the IN file.
		write	Performs a blocking write operation to send data to a USB host.
		aio_write	Performs a non-blocking, asynchronous write operation to send data to a USB host.
		flush	Removes the data from the IN endpoint.
Gadget USB OUT endpoints	/dev/DT7816-ep1out /dev/DT7816-ep2out /dev/DT7816-ep3out /dev/DT7816-ep4out /dev/DT7816-ep5out	**open**	Opens the OUT file.
		close	Closes the OUT file.
		read	Performs a blocking read operation to receive data from a USB host.
		aio_read	Performs a non-blocking, asynchronous read operation to receive data from a USB host.
		flush	Removes the data from the IN endpoint.

2

Using the File I/O Commands

Opening and Closing a File

The DT7816 device driver currently supports the following files for performing embedded I/O operations on the module:

- **/dev/DT7816-ain** – Analog input subsystem. You can configure the parameters of each analog input channel of the analog input subsystem and then read the continuous values from the analog input subsystem using the input stream file (/dev/DT7816-stream-in).
- **/dev/DT7816-aout** – Analog output subsystem. You can configure the parameters of each analog output channel of the analog output subsystem and then perform a synchronous write operation to output a single value from the analog output subsystem or output continuous values from the analog output subsystem using the output stream file (/dev/DT7816-stream-out).
- **/dev/DT7816-din** – Digital input subsystem. This subsystem supports synchronous read operations where a single value is read from the digital input port. To read the digital input data with the analog input data in the input stream, use the /dev/DT7816-stream-in file instead.
- **/dev/DT7816-dout** – Digital output subsystem. This subsystem supports synchronous write operations where a single value is written to the digital output port.
- **/dev/DT7816-tach** – Tachometer input subsystem. You can configure the parameters of the tachometer input channel. You can then read the value of the tachometer using a synchronous read operation, or you can read the tachometer input with the analog input data in the input stream using the /dev/DT7816-stream-in file instead.
- **/dev/DT7816-measure** – Measure counter subsystem. You can configure the parameters of the measure counter. You can then read the value of the measure counter using a synchronous read operation, or you can read the measure counter with the analog input data in the input stream using the /dev/DT7816-stream-in file instead.
- **/dev/DT7816-ctr-tmr** – Counter/timer subsystem. You can configure the parameters of the counter/timer. You can then read the value of the counter/timer using a synchronous read operation.
- **/dev/DT7816-stream-in** – Input stream. Use this file to perform a continuous input operation from each input channel specified in the input stream. Channels can include the analog inputs, digital input port, tachometer input, measure counter, and counter/timer.
- **/dev/DT7816-stream-out** – Output stream. Use this file to perform a continuous output operation from each input channel specified in the output stream. Channels can include the analog output channels and the digital output port.

The DT78xx USB gadget driver supports these virtual files for sending data to and receiving data from the USB host application:

- **/dev/DT7816-ep1in** – EP1 IN endpoint, address 0x81.
- **/dev/DT7816-ep1out** – EP1 OUT endpoint, address 0x01.
- **/dev/DT7816-ep2in** – EP2 IN endpoint, address 0x82.
- **/dev/DT7816-ep2out** – EP2 OUT endpoint, address 0x02.
- **/dev/DT7816-ep3out** – EP3 OUT endpoint, address 0x03.
- **/dev/DT7816-ep4out** – EP4 OUT endpoint, address 0x04.

- **/dev/DT7816-ep5out** – EP5 OUT endpoint, address 0x05.

Before you can perform an I/O operation, you must open the file that you want to read from or write to using the Linux file I/O command **open**.

You specify the pathname of the file to open and a flag that determines how to open the file, and the command returns a file descriptor that is used in subsequent calls to the file. The following flags determine how the file is opened:

- O_RDONLY –Read only
- O_WRONLY – Write only
- O_RDWR – Read and write

When you finished performing the I/O operation, you must close each subsystem and/or stream that was opened using the Linux file I/O command **close**.

Analog Input and Input Stream Operations

The DT7816 module supports continuous analog input operations using the input stream.

You can also acquire data from other input channels besides the analog input channels in the input stream, including the digital input port, tachometer, and measure counter. This section describes analog input and input stream operations on the DT7816 module.

Opening the Subsystem and Input Stream

To perform an analog input operation, open the file corresponding to the analog input subsystem (/dev/DT7816-ain) and in the input stream (/dev/DT7816-stream-in) using the Linux command **open**, described on page 22.

Configuring the Input Channels

To configure the analog input channels of the analog input subsystem, use the **ioctl - IOCTL_AIN_CFG_SET** command.

You specify the file descriptor for the analog input subsystem and then specify the parameters for each channel of the subsystem. Parameters include the following:

- Analog input channel to configure (ain) – Specify analog input channel 0 to 7.
- Gain (*gain*) – Specify a gain of 1 for an effective input range of ±10 V.
- Input type (*differential*) – Specify either the single-ended or differential input type for the analog input channel. Currently, the DT7816 supports only single-ended inputs.

If you want to read the value of the tachometer measurement in the input stream, you must also open and configure the tachometer subsystem; refer to page 39.

If you want to read the value of the measure counter, you must open and configure the measure counter subsystem; refer to page 48 for more information.

Note: The digital input port does not require configuration; therefore, you do not need to open or configure this subsystem.

You can return the current configuration of the analog input subsystem using the **ioctl - IOCTL_AIN_CFG_GET** command.

Configuring the Channel Mask for the Input Stream

Once you have configured the input channel for each subsystem that you want to use, set up the channel mask for the input stream using the **ioctl - IOCTL_CHAN_MASK_SET** command. You specify the channels that you want to enable for acquisition. Channels can include the analog input channels (bits 0 to 7 of the channel mask for analog input channels 0 to 7), tachometer (bit 8 of the channel mask), measure counter (bit 10 of the channel mask), and/or digital input port (bit 11 of the channel mask).

You can return the current configuration of the channel mask using the **ioctl - IOCTL_CHAN_MASK_GET** command.

Configuring the Sample Clock

Use the **ioctl - IOCTL_SAMPLE_CLK_SET** command to configure the clock source for the input stream.

For the DT7816, you can specify either the internal clock source on the DT7816 module or an external clock source that you must connect to the module.

If you choose an internal clock source, you must specify the frequency at which to sample the specified channels in the input stream. Supported values range from 1 Hz to 400 kHz.

If you choose the external clock source, you must specify which of the general-purpose input signals of the 40-pin I/O header to use for as the external A/D clock source. You must also specify the frequency at which to sample the specified channels in the input stream. The maximum frequency supported is 400 kHz.

Note: According to sampling theory (Nyquist Theorem), specify a frequency that is at least twice as fast as the input's highest frequency component. For example, to accurately sample a 20 kHz signal, specify a sampling frequency of at least 40 kHz to avoid aliasing.

The DT7816 driver sets the frequency as close as possible to the value that you specify. However, the value that you specify may not be the actual value that is set. To return the actual sample clock frequency that was set, use the **ioctl - IOCTL_SAMPLE_CLK_SET** command.

Configuring the Trigger that Starts Acquisition

Use the **ioctl - IOCTL_START_TRIG_CFG_SET** command to specify a start trigger that starts acquisition for the input stream. The DT7816 module supports the following sources for the start trigger:

- **Software trigger** (*trig_src_sw*) – When you specify this trigger source, the trigger event occurs when you start the analog input operation using the **ioctl - IOCTL_START_SUBSYS** command for the input stream, described on page 34.
- **External digital (TTL) trigger** (*trig_src_ext*)– When you specify this source, the trigger event occurs when the module detects a rising- or falling-edge transition on the specified general-purpose input pin on the I/O header. Using software, you can specify which edge is active and which of the following pins of the I/O header to use for the external trigger:
  - Pin 1 corresponds to bit 0 of the digital input port (value 0x1).
  - Pin 2 corresponds to bit 1 of the digital input port (value 0x2).
  - Pin 3 corresponds to bit 2 of the digital input port (value 0x4).
  - Pin 4 corresponds to bit 3 of the digital input port (value 0x8).
  - Pin 5 corresponds to bit 4 of the digital input port (value 0x10).
  - Pin 6 corresponds to bit 5 of the digital input port (value 0x20).
  - Pin 7 corresponds to bit 6 of the digital input port (value 0x40).
  - Pin 8 corresponds to bit 7 of the digital input port (value 0x80).

Note: If you assigned a general-purpose input signal as an external trigger (or counter clock or gate input), you can read the value of the signal as you would any other digital input signal. Refer to page 52 for more information on reading digital input values.

- **Threshold trigger** (*trig_src_threshold*) – When you specify this source, the trigger event occurs when the signal attached to a specified analog input channel rises above or falls below a user-specified threshold value. Using software, you specify the following parameters:
  - Edge – Specify a rising-edge threshold trigger if you want to trigger when the signal rises above a threshold level, or a falling-edge threshold trigger if you want to trigger when the signal falls below a threshold level.
  - Threshold channel – Specify any one of the analog input channels as the threshold input channel.
  - Threshold level – Specify a value between ±10 V as the threshold level. Note that in software, this value must be entered as counts.

    To convert volts to raw counts (using two's complement encoding), use this formula:

    counts = ((*volts*/10.0f) * (1 << (16 – 1)))

To convert raw counts to volts (using two's complement encoding), use this formula:

volts = ((*counts* * 10.0f)/(1 << (16 – 1)))

Note: The DT7816 driver sets the threshold level as close as possible to the value that you specify. However, the value that you specify may not be the actual value that is set. To return the actual threshold level that was set, use the **ioctl - IOCTL_START_TRIG_CNF_GET** command.

You can return the current configuration of the start trigger using the **ioctl - IOCTL_START_TRIG_CFG_GET** command.

Submitting I/O Requests

The DT7816 module works with the Linux AIO (Asynchronous Input/Output) model to submit I/O requests to the driver. The application can submit one or many requests from a thread. Submitting a request does not cause the thread to block, and instead the thread can proceed to do other computations and submit further requests to the device while the original request is in process. The application is expected to process completions and organize logical computations itself without depending on threads to organize the use of data.

For the DT7816, you must perform the following steps to set up and submit I/O requests to the driver for an input operation:

1. Open an asynchronous I/O context using the Linux command **io_setup**.
2. Allocate an array of I/O control blocks (struct iocbs).

   Each I/O control block is a structure with a number of parameters, one of which is the buffer used to store the output waveform or pattern.

   For the DT7816, buffers must be on a 32-byte boundary and the length of each buffer must be a multiple of 32 bytes.

   Each analog input and digital input sample requires two bytes, while each tachometer and measure counter requires four bytes.
3. Submit I/O requests to the asynchronous I/O context using **io_submit**.

Refer to your Linux documentation, such as Refer to the following website for more information about this model: https://code.google.com/p/kernel/wiki/AIOUserGuide, for more information on these commands and the AIO model.

Arming and Starting Continuous Operations

Once you have configured all the parameters for the subsystems used in the input stream, configured the input stream, and set up and submitted I/O requests, arm the input stream using the **ioctl - IOCTL_ARM_SUBSYS** command.

If an external trigger or threshold trigger was configured as the start trigger, the continuous input operation starts when the module detects the trigger condition.

If a software trigger was specified as the start trigger, you must explicitly start the continuous input operation using the **ioctl - IOCTL_START_SUBSYS** command. The operation starts immedately after the execution of this command.

When triggered, the DT7816 module simultaneously acquires data from all of the channels specified in the input stream.

Note: You can start a continuous operation on both the input and output streams simultaneously by specifying a non-zero value for the *pSimultaneous* variable of the **ioctl - IOCTL_START_SUBSYS** command.

Getting the Status of Acquisition

Using the **ioctl - IOCTL_ACQ_STATUS_GET** command, you can determine if the input operation has been armed or triggered, and whether the input FIFO is empty (contains no data) or is full.

Processing I/O Requests

As I/O requests are processed, events are generated. Your application must process the I/O requests as they are completed using the Linux command **io_getevents**. As each request is completed, it is up to the application to retrieve the data from the buffers and manage the I/O control blocks to ensure that buffers are available to be filled.

You can also set a timeout value for the **io_getevents** command so that the operation will stop if the request is not completed within a specified time.

Refer to your Linux documentation for more information on the **io_getevents** command.

Dealing with Input Buffers

The order of the data in the input buffer is as follows, assuming that all channels are enabled in the input stream:

- Analog input channels 0 through 7. Each analog input sample is a 16-bit, two's complement raw A/D value.
- Tachometer input. This is a 32-bit, unsigned value.

- Measure/counter. This is a 32-bit, unsigned value.
- Digital input port. This is 16-bit, unsigned value. The digital input data is in the least significant eight bits.

Refer to page 25 for more information on the channel mask for the input stream.

Enabling Buffer Error Reporting

By default, input stream buffer overrun are not reported to user programs. You can enable error reporting so that these errors are reported from the kernel device driver to the user program through signals.

To enable error reporting, perform the following steps:

1. Write your buffer error handler.
2. Open the input stream and then set the process ID or group ID that will receive the signal for input stream events. By default, SIGIO is the signal that is used to report buffer overrun errors. However, you can choose another signal, such as SIGUSR1, SIGUSR2, and so on, for reporting errors.

Note: If you want to enable buffer error reporting for both the input and output streams, it is recommended that you choose a different signal for each stream.

---

3. Get all the status flags for the input stream and then set the asynchronous flag (FASYNC) for the input stream.
4. (Optional) If you want to use a different signal (SIGUSR1, SIGUSR2, and so on) instead of the SIGIO signal, register the signal.
5. Register the signal handler for SIGIO or the signal that you register in step 4.

The following code show how to enable and handle buffer overrun errors from the input stream device file; refer to your Linux documentation for more information:

```
//include files
#define __USE_GNU   (1)
#include <signal.h>
#include <fcntl.h>

//Step 1
//Write an error handler. Note that the signal handler is
//registered in Step 5 below

void buff_overrun_handler(int sig)
{
//your code here
}
```

```
//Step 2
//Open input stream file and set the process ID for the signal
//to use for reporting errors

int fd_in = open("/dev/DT7816-stream-in", O_RDWR);
if ((fcntl(fd_in, F_SETOWN, getpid())) < 0)
{
    //handle error
}

//Step 3
//Get the flags for the input stream and set the async flag (FASYNC)
int oflags = fcntl(fd_in, F_GETFL);
if ((fcntl(fd_in, F_SETFL, oflags | FASYNC)) < 0)
{
    //handle error
}

//Step 4
//This step is optional. By default buffer errors are reported
//using the signal SIGIO.
//To use a different signal, such as SIGUSR1, SIGUSR2, and so on,
//register the signal by specifying its number

if ((fcntl(fd_in, F_SETSIG, SIGUSR1)) < 0)
{
    //handle error

//Step 5
//Register a signal handler for the default signal, SIGIO, or the
//signal specified in Step 4

struct sigaction act;
memset (&act, 0, sizeof(act));
act.sa_handler = buff_overrun_handler; //signal handler
act.sa_flags = SA_NODEFER;
sigemptyset(&act.sa_mask);
if (sigaction(SIGUSR1, &act, NULL)) //Use SIGIO if you skip step 4
{
    //handle error
}
```

Stopping Continuous Operations

Once started, a continuous operation repeats continuously until you stop it with the **ioctl - IOCTL_STOP_SUBSYS** command.

This command stops an operation that was previously started with the **ioctl - IOCTL_START_SUBSYS** command (a software trigger was specified) or was started when the specified trigger condition was detected.

This command stops the DMA engine immediately and no further data is collected. Asynchronous I/O control blocks that were submitted using **io_submit** are still in the AIO queue and will not be completed. To cancel these control blocks, use **io_cancel**.

If you want to restart the operation for the input stream, you must rearm the input stream using **ioctl - IOCTL_ARM_SUBSYS**, and, if a software trigger is specified, restart the operation using **ioctl - IOCTL_START_SUBSYS**.

Cleaning up Resources

Once you are finished acquiring data, clean up the resources used by performing the following steps:

1. Cancel any outstanding I/O requests using the Linux command **io_cancel**.
2. Destroy the asynchronous I/O context using the Linux command **io_destroy**.
3. Close the analog input subsystem, the input stream, and any other open subsystems used in the input stream, using the **close** command, described on page 23.

Refer to your Linux documentation for more information on these commands.

Analog Output and Output Stream Operations

The DT7816 has two analog output channels You can write a single value to the analog output channels using the analog output subsystem.

You can also update the analog output channels, and if desired, the digital output port continuously using the output stream.

This section describes these operations.

Performing a Single Value Operation

This section describes how to perform a single value analog output operation.

Opening the Subsystem

To perform a single value operation, open the file corresponding to the analog output subsystem (/dev/DT7816-aout) using the Linux command **open**, described on page 22.

Updating the Value of the Analog Output Channel

You can update the value of an analog output channel by performing a synchronous write operation using the Linux command **write**, specifying the file descriptor for the analog output subsystem, the variable that contains the value to write, and the size of the variable that contains the analog output value.

The write operation is blocking, in that it does not return until the value is written. The operation stops automatically once the value is written.

Closing the Subsystem

When finished, close the analog output subsystem using the **close** command, described on page 23.

Performing a Continuous Output Operation

This section describes how to perform a continuous output operation.

Opening the Output Stream

To perform a continuous analog output operation, open the file corresponding to the output stream (/dev/DT7816-stream-out) using the Linux command **open**, described on page 22.

Configuring the Channel Mask for the Output Stream

Set up the channel mask for the output stream using the **ioctl - IOCTL_CHAN_MASK_SET** command. You specify the channels that you want to enable for output. For the DT7816, you can include the analog output channels (bit 16 and 17 of the channel mask) and/or the individual lines of the digital output port (bits 24 to 31 of the channel mask) in the output stream.

You can return the current configuration of the channel mask using the **ioctl - IOCTL_CHAN_MASK_GET** command.

Configuring the Sample Clock

Use the **ioctl - IOCTL_SAMPLE_CLK_SET** command to configure the clock source for the output stream.

For the DT7816, you can specify either the internal clock source on the DT7816 module or an external clock source that you must connect to the module.

If you choose an internal clock source, you must specify the frequency at which to update the specified channels in the output stream. Supported values range from 1 Hz to 400 kHz.

If you choose the external clock source, you must specify which of the general-purpose input signals of the 40-pin I/O header to use for as the external D/A clock source. You must also specify the frequency at which to update the specified channels in the output stream. The maximum frequency supported is 400 kHz.

The DT7816 driver sets the frequency as close as possible to the value that you specify. However, the value that you specify may not be the actual value that is set. To return the actual sample clock frequency that was set, use the **ioctl - IOCTL_SAMPLE_CLK_SET** command.

Configuring the Trigger that Starts the Output Operation

Use the **ioctl - IOCTL_START_TRIG_CFG_SET** command to specify a start trigger that starts the continuous operation for the output stream. The DT7816 module supports the following sources for the start trigger:

- **Software trigger** (*trig_src_sw*) – When you specify this trigger source, the trigger event occurs when you start the analog output operation using the **ioctl - IOCTL_START_SUBSYS** command for the output stream, described on page 34.
- **External digital (TTL) trigger** (*trig_src_ext*)– When you specify this source, the trigger event occurs when the module detects a rising- or falling-edge transition on the specified general-purpose input pin on the I/O header. Using software, you can specify which edge is active and which of the following pins of the I/O header to use for the external trigger:
  - Pin 1 corresponds to bit 0 of the digital input port (value 0x1).
  - Pin 2 corresponds to bit 1 of the digital input port (value 0x2).
  - Pin 3 corresponds to bit 2 of the digital input port (value 0x4).
  - Pin 4 corresponds to bit 3 of the digital input port (value 0x8).

- Pin 5 corresponds to bit 4 of the digital input port (value 0x10).
- Pin 6 corresponds to bit 5 of the digital input port (value 0x20).
- Pin 7 corresponds to bit 6 of the digital input port (value 0x40).
- Pin 8 corresponds to bit 7 of the digital input port (value 0x80).

You can return the current configuration of the start trigger using the **ioctl - IOCTL_START_TRIG_CFG_GET** command.

Submitting I/O Requests

The DT7816 module works with the Linux AIO (Asynchronous Input/Output) model to submit I/O requests to the driver. The application can submit one or many requests from a thread. Submitting a request does not cause the thread to block, and instead the thread can proceed to do other computations and submit further requests to the device while the original request is in process. The application is expected to process completions and organize logical computations itself without depending on threads to organize the use of data.

For the DT7816, you must perform the following steps to set up and submit I/O requests to the driver for an output operation:

1. Open an asynchronous I/O context using the Linux command **io_setup**.
2. Allocate an array of I/O control blocks (struct iocbs).

   Each I/O control block is a structure with a number of parameters, one of which is the buffer used to store the data that is acquired.

   For the DT7816, buffers must be on a 32-byte boundary and the length of each buffer must be a multiple of 32 bytes.

   Each analog output and digital output sample requires two bytes.
3. Fill each buffer with the pattern that you want to output to the enabled channels in the output stream.
4. Submit I/O requests to the asynchronous I/O context using **io_submit**.

Refer to your Linux documentation, such as Refer to the following website for more information about this model: https://code.google.com/p/kernel/wiki/AIOUserGuide, for more information on these commands and the AIO model.

Arming and Starting Continuous Operations

Once you have configured the output stream and set up and submitted I/O requests, arm the output stream using the **ioctl - IOCTL_ARM_SUBSYS** command.

If an external trigger was configured as the start trigger, the continuous output operation starts when the module detects the trigger condition.

If a software trigger was specified as the start trigger, you must explicitly start the continuous input operation using the **ioctl - IOCTL_START_SUBSYS** command. The operation starts immedately after the execution of this command.

When triggered, the DT7816 module simultaneously updates all of the channels specified in the output stream.

Note: You can start a continuous operation on both the input and output streams simultaneously by specifying a non-zero value for the *pSimultaneous* variable of the **ioctl - IOCTL_START_SUBSYS** command.

Getting the Status of the Output Operation

Using the **ioctl - IOCTL_ACQ_STATUS_GET** command, you can determine if the output operation has been armed or triggered, and whether the output FIFO is empty (contains no data) or is full.

Processing I/O Requests

As I/O requests are processed, events are generated. Your application must process the I/O requests as they are completed using the Linux command **io_getevents**. As each request is completed, it is up to the application to retrieve the data from the buffers and manage the I/O control blocks to ensure that buffers are filled and available.

You can also set a timeout value for the **io_getevents** command so that the operation will stop if the request is not completed within a specified time.

Refer to your Linux documentation for more information on the **io_getevents** command.

Dealing with Output Buffers

The order of the data in the output buffer is as follows, assuming that all channels are enabled in the output stream:

- Analog output channels 0 and1. Each analog output sample is a 16-bit, two's complement value.
- Digital output port. This is 16-bit unsigned value. The digital output data is in the least significant eight bits.

Refer to page 33 for more information on the channel mask for the output stream.

Enabling Buffer Error Reporting

By default, output stream buffer underrun errors are not reported to user programs. You can enable error reporting so that these errors are reported from the kernel device driver to the user program through signals.

To enable error reporting for the output stream, perform the following steps:

1. Write your buffer error handler.

2. Open the output stream and then set the process ID or group ID that will receive the signal for output stream events. By default, SIGIO is the signal that is used to report buffer overrun errors. However, you can choose another signal, such as SIGUSR1, SIGUSR2, and so on, for reporting errors.

Note: If you want to enable buffer error reporting for both the input and output streams, it is recommended that you choose a different signal for each stream.

3. Get all the status flags for the output stream and then set the asynchronous flag (FASYNC) for the output stream.
4. (Optional) If you want to use a different signal (SIGUSR1, SIGUSR2, and so on) instead of the SIGIO signal, register the signal.
5. Register the signal handler for SIGIO or the signal that you register in step 4.

The following code show how to enable and handle buffer underrun errors from the output stream device file.

```
//include files
#define __USE_GNU   (1)
#include <signal.h>
#include <fcntl.h>

//Step 1
//Write an error handler. Note that the signal handler is
//registered in Step 5 below

void buff_underrun_handler(int sig)
{
//your code here
}

//Step 2
//Open output stream file and set the process ID for the signal
//to use for reporting errors

int fd_out = open("/dev/DT7816-stream-out", O_RDWR);
if ((fcntl(fd_out, F_SETOWN, getpid())) < 0)
{
    //handle error
}

//Step 3
//Get the flags for the output stream and set the async flag (FASYNC)
int oflags = fcntl(fd_out, F_GETFL);
if ((fcntl(fd_out, F_SETFL, oflags | FASYNC)) < 0)
{
    //handle error
}
```

```
//Step 4
//This step is optional. By default buffer errors are reported
//using the signal SIGIO.
//To use a different signal, such as SIGUSR1, SIGUSR2, and so on,
//register the signal by specifying its number

if ((fcntl(fd_out, F_SETSIG, SIGUSR1)) < 0)
{
    //handle error
}

//Step 5
//Register a signal handler for the default signal, SIGIO, or the
//signal specified in Step 4

struct sigaction act;
memset (&act, 0, sizeof(act));
act.sa_handler = buff_underrun_handler; //signal handler
act.sa_flags = SA_NODEFER;
sigemptyset(&act.sa_mask);
if (sigaction(SIGUSR1, &act, NULL)) //Use SIGIO if you skip step 4
{
    //handle error
}
```

Stopping Continuous Operations

Once started, a continuous operation repeats continuously until you stop it with the **ioctl - IOCTL_STOP_SUBSYS** command.

This command stops an operation that was previously started with the **ioctl - IOCTL_START_SUBSYS** command (a software trigger was specified) or was started when the specified trigger condition was detected.

This command stops the DMA engine immediately and no further data is output. Asynchronous I/O control blocks that were submitted using **io_submit** are still in the AIO queue and will not be completed. To cancel these control blocks, use **io_cancel**.

If you want to restart the operation for the output stream, you must rearm the output stream using **ioctl - IOCTL_ARM_SUBSYS**, and, if a software trigger is specified, restart the operation using i**octl - IOCTL_START_SUBSYS**.

Cleaning up Resources

Once you are finished outputting data, clean up the resources used by performing the following steps:

1. Cancel any outstanding I/O requests using the Linux command **io_cancel**.
2. Destroy the asynchronous I/O context using the Linux command **io_destroy**.
3. Close the output stream using the **close** command, described on page 23.

Refer to your Linux documentation for more information on these commands.

Tachometer Operations

The DT7816 module supports one tachometer input signal. You can measure the frequency or period of the tachometer input signal to calculate the rotation speed for high-level (±30 V) tachometer input signals. An internal 12 MHz counter is used for the measurement, yielding a resolution of 83 ns (1/12 MHz). The value of the tachometer (a 32-bit value) can be returned in the input stream.

To read the tachometer measurement in the input stream, set bit 8 of the channel mask of the input stream. Refer to page 32 for more information on analog input and input stream operations.

This section describes how to configure the parameters of the tachometer subsystem.

Opening the Subsystem

Open the file corresponding to the tachometer subsystem (/dev/DT7816-tach using the Linux command **open**, described on page 22.

Configuring the Tachometer Subsystem

To configure the tachometer subsystem, use the **ioctl - IOCTL_TACH_CFG_SET** command.

You specify the file descriptor for the tachometer subsystem and the following parameters:

- The edge (rising or falling) of the tachometer to use for the measurement. The number of counts betweeen two consecutive edges of the tachometer input signal is used as the tachometer measurement.
- The Stale flag (*stale_flag*) that indicates whether or not the data is new. If *stale_flag* is set as Used (1), the most significant bit (MSB) of the value is set to 0 to indicate new data; reading the value before the measurement is complete returns an MSB of 1. If *stale_flag* is set to Not Used (0), the MSB is always set to 0.

When the input operation is started, the internal 12 MHz counter starts incrementing when it detects the first starting edge of the tachometer input and stops incrementing when it detects the next starting edge; at that point, the counter stores the count. The stored count is maintained until it is read as part of the input data stream or until a new count is stored. The next tachometer measurement operation is started automatically.

If the sample rate of the input subsystem is faster than the tachometer input frequency, then the stored count retains the current value when the count is read by the input subsystem. The operation of *stale_flag* in this case is described as follows:

- If another input subsystem sample occurs before another measure completes and *stale_flag* is used, then the Stale flag is set and the stale measure count is written into the input data stream.
- If another input subsystem sample occurs before another measure completes and *stale_flag* is not used, then the Stale flag is not set and the stale measure count is written into the input data stream.

If the input sample rate is slower than the tachometer input frequency, then as each period measurement completes, a new count value is stored. When the input subsystem sample occurs, the most recently stored measure count is written into the input data stream.

A data pipeline is used in the hardware to compensate for the A/D group delay and synchronizes the value of the tachometer input with the analog input measurements so that all measurements are correlated in time.

When you read the value of the tachometer input as part of the input stream, you might see results similar to the following:

Table 3: An Example of Reading the Tachometer Input as Part of the Input Stream

Time	A/D Value	Tachometer Input Value	Status of Operation
10	5002	0	Operation started, but is not complete
20	5004	0	Operation not complete
30	5003	0	Operation not complete
40	5002	12373	Operation complete
50	5000	12373	Next operation started, but is not complete
60	5002	12373	Operation not complete
70	5004	12373	Operation not complete
80	5003	14503	Operation complete
90	5002	14503	Next operation started, but is not complete

Using the count that is returned from the tachometer input, you can determine the following:

- Frequency of a signal pulse (the number of periods per second). You can calculate the frequency as follows:
  - Frequency = 12 MHz/(Number of counts – 1)
    where 12 MHz is the internal counter/timer clock frequency

    For example, if the count is 21, the measured frequency is 600 kHz (12 MHz/20).
- Period of a signal pulse. You can calculate the period as follows:
  - Period = 1/Frequency
  - Period = (Number of counts – 1)/12 MHz
    where 12 MHz is the internal counter/timer clock frequency

You can return the current configuration of the tachometer subsystem using the **ioctl - IOCTL_TACH_CFG_GET** command.

Counter/Timer Operations

The DT7816 module supports one counter/timer channel. You can read the value of the counter (a 32-bit, unsigned value) directly using a synchronous read of the counter/timer subsystem.

This section describes counter/timer operations.

Opening the Subsystem

Open the file corresponding to the counter/timer subsystem (/dev/DT7816-ctr-tmr) using the Linux command **open**, described on page 22.

Configuring the Counter/Timer Subsystem

To configure the counter/timer subsystem, use the **ioctl - IOCTL_CT_CFG_SET** command.

You specify the file descriptor for the counter/timer subsystem and then specify the parameters specific to the counter/timer subsystem. This section describes each of the configurable parameters of the counter/timer subsystem.

You can return the current configuration of the counter/timer subsystem using the **ioctl - IOCTL_CT_CFG_GET** command.

Mode

The DT7816 module supports the following counter/timer modes:

- Event counting – described below.
- Rate generation – described on page 42
- Non-retriggerable one-shot mode – described on page 43
- Idle – described on page 44.

Event Counting Mode

Use event counting mode (*ct_mode_counter*) if you want to count the number of rising edges that occur on the counter's clock input when the counter's gate signal is active (low-level or high-level).

You can count a maximum of 4,294,967,296 events before the counter rolls over to 0 and starts counting again.

Using software, you must specify the following parameters for the event counting operation:

- Active gate type (external low level or external high level). Refer to page 44 for more information about the supported gate types.
- The general-purpose input pin (1 to 8) of the I/O header to use for the external gate signal. Ensure that you physically connect the gate signal to this input pin. Refer to the *DT7816 User's Manual* for the pin descriptions of the I/O header.
- The C/T clock source (internal or external). Note that in event counting mode, the external C/T clock is more useful than an internal C/T clock; refer to page 45 for more information about the C/T clock sources.
- The general-purpose input pin (1 to 8) of the I/O header to use for the external C/T clock input. Ensure that you physically connect the clock input signal to this input pin. Refer to the *DT7816 User's Manual* for the pin descriptions of the I/O header.

Rate Generation Mode

Use rate generation mode (*ct_mode_divider)* to generate a continuous pulse output signal from the counter's output signal. You can use this pulse output signal as an external clock to pace other operations, such as an analog input or other counter/timer operations.

The pulse output operation is enabled whenever the counter's gate signal is active. While the pulse output operation is enabled, the counter outputs a pulse of the specified type and frequency continuously. As soon as the operation is disabled, rate generation stops.

You can output pulses using a maximum frequency of 24 MHz (if using the internal C/T clock) or 5 MHz (if using the external C/T clock).

Note: The integrity of the signal degrades at frequencies greater than 10 MHz.

Using software, you must specify the following parameters for the rate generation operation:

- Active gate type (external low level or external high level). Refer to page 44 for more information about the supported gate types.
- The general-purpose input pin (1 to 8) of the I/O header to use for the external gate signal. Ensure that you physically connect the gate signal to this input pin. Refer to the *DT7816 User's Manual* for the pin descriptions of the I/O header.
- The C/T clock source (internal or external). Refer to page 45 for more information about the C/T clock sources.
- If you are using an external C/T clock source, the general-purpose input pin (1 to 8) of the I/O header to use for the external C/T clock input. Ensure that you physically connect the clock input signal to this input pin. Refer to *DT7816 User's Manual* for the pin descriptions of the I/O header.
- The period of the output pulse. Refer to page 46 for more information about the period of the output pulse.
- The pulse width (duty cycle) of the active pulse. Refer to page 46 for more information about the pulse width of the output pulse.

- The polarity of the output signal (active high or active low). Refer to page 46 for more information on the polarity of the output pulse.
- The general-purpose output pin (11 to 18) of the I/O header to use for the external C/T clock output signal. Ensure that you physically connect the C/T output signal to this output pin. Refer to *DT7816 User's Manual* for the pin descriptions of the I/O header.

Non-Retriggerable One-Shot Mode

Use non-retriggerable one-shot mode (*ct_mode_1shot*) to generate a single output pulse from the counter whenever the specified edge is detected on the counter's gate signal (after the pulse period from the previous output pulse expires). Any gate signals that occur while the pulse is being output are not detected by the module, as shown in Figure 3. The module continues to output a pulse when the specified gate edge is detected until you stop the operation. You can use this mode to clean up a poor clock input signal by changing its pulse width, and then outputting it.

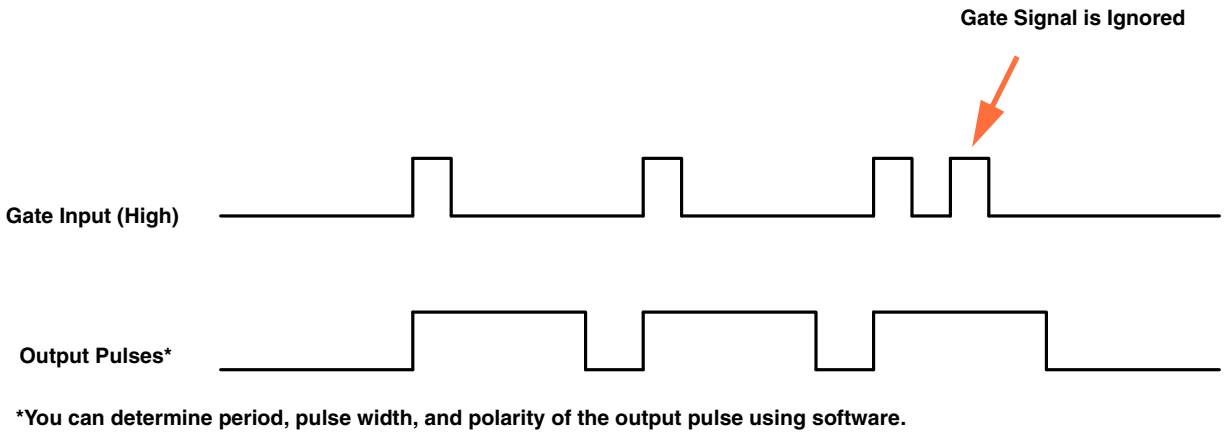


Figure 3: Non-Retriggerable One-Shot Mode

Using software, you must specify the following additional parameters to set up the non-retriggerable one-shot operation:

- Active gate type that enables the output pulse. Refer to page 44 for more information about the supported gate types.
- The general-purpose input pin (1 to 8) of the I/O header to use for the external gate signal. Ensure that you physically connect the gate signal to this input pin. Refer to the *DT7816 User's Manual* for the pin descriptions of the I/O header.
- The C/T clock source (internal or external) used to generate the pulse. Note that in non-retriggerable one-shot mode, the internal C/T clock is more useful than an external C/T clock; refer to page 45 for more information about the C/T clock sources.

- If using the external C/T clock source, the general-purpose input pin (1 to 8) of the I/O header to use for the external C/T clock input. Ensure that you physically connect the clock input signal to this input pin. Refer to the *DT7816 User's Manual* for the pin descriptions of the I/O header.
- The period of the output pulse. Refer to page 46 for more information about the period of the output pulse.
- The pulse width (duty cycle) of the output pulse. Refer to page 46 for more information about the pulse width of the output pulse.
- The general-purpose output pin (11 to 18) of the I/O header to use for the external C/T clock output signal. Ensure that you physically connect the C/T output signal to this output pin. Refer to the *DT7816 User's Manual* for the pin descriptions of the I/O header.
- Retriggerable setting of 0 for non-retriggerable one-shot mode.
- The polarity of the output signal (active high or active low). Refer to page 46 for more information on the polarity of the output pulse.

Idle Mode

When you use idle mode (*ct_mode_idle),* the counter no longer drives the clock output signal that is assigned to one of the general-purpose output signals (pins 11 to 18) of the I/O header.

Note: The value of the counter output signal can also be overwritten by writing to the digital output subsystem. Refer to page 53 for more information.

If you assigned a general-purpose input signal as a counter clock or gate input (or external trigger), you can read the value of the signal as you would any other digital input signal. Refer to page 52 for more information on reading digital input values.

Gate

The counter's gate signal determines when a counter/timer operation is enabled.

DT7816 modules provide the following choices for the gate parameter:

- **None** (*ct_gate_none*) – A software start command enables the counter/timer operation immediately after execution. (No general-purpose input signal is required if a gate type of *ct_gate_none* is selected.)
- **Low external gate input** (*ct_gate_ext_lo*) – Specifies a logic low or falling edge gate type. For event counting and rate generation mode, the operation is enabled when the counter's gate signal is low and is disabled when the counter's gate signal is high. For one-shot or repetitive one-shot mode, the operation is enabled when the counter's gate signal goes from a high to a low transition and is disabled when the counter's gate signal goes from a low to a high transition.

Using software, you specify one of the general-purpose input pins of the I/O header on the DT7816 module as the external C/T gate input (ext_gate_din). Ensure that you physically connect the external gate signal to the selected pin. (Refer to the *DT7816 User's Manual* for the pin descriptions of the I/O header.)

- **High external gate input** (*ct_gate_ext_hi*) – Specifies a logic high or rising edge gate type. For event counting and rate generation mode, the operation is enabled when the counter's gate signal is high and is disabled when the counter's gate signal is low. For one-shot mode and repetitive one-shot mode, the operation is enabled when the counter's gate signal goes from a low to a high transition and is disabled when the counter's gate signal goes from a high to a low transition.

  You specify one of the general-purpose input pins of the I/O header on the DT7816 module as the external C/T gate input (ext_gate_din). Ensure that you physically connect the external gate signal to the selected pin. (Refer to the *DT7816 User's Manual* for the pin descriptions of the I/O header.)

C/T Clock Input Sources

The following input clock sources are available for the general-purpose counter/timer:

- **Internal C/T clock** – The internal C/T clock on the DT7816 uses a 48 MHz time base. This clock source is typically used for one-shot, repetitive one-shot, and rate generation operations.
- **External C/T clock** – An external C/T clock is useful when you want to pace counter/timer operations at rates not available with the internal C/T clock or if you want to pace at uneven intervals. The frequency of the external C/T clock can range from 0.0112 Hz to 10 MHz.

  This clock source is typically used for event counting operations or rate generation operations.

  You specify one of the general-purpose input pins (1 to 8) of the I/O header on the DT7816 module as the external C/T clock (ext_clk_din). Then, physically connect the external clock signal to the selected general-purpose input pin. (Refer to the *DT7816 User's Manual* for the pin descriptions of the I/O header.) Counter/timer operations start on the rising edge of the clock input signal.

Pulse Output Period, Pulse Width, and Polarity

If you want to perform a C/T output operation, use software to define one of the general-purpose output pins (11 to 18) of the I/O header on the DT7816 module as the external C/T output signal. Then, connect the external C/T output signal to the selected general-purpose output pin. (Refer to the *DT7816 User's Manual* for the pin descriptions of the I/O header.)

For the DT7816 module, you can program the polarity of the output pulse (active high or active low). For an active high pulse, the high portion of the total pulse output period is the active portion of the counter/timer pulse output signal. For an active low pulse, the low portion of the total pulse output period is the active portion of the counter/timer pulse output signal.

You can specify the number of input clock cycles that are used to create one period of the counter clock output signal. You can also specify the number of input clock cycles used to create the active pulse width (or duty cycle) of the C/T output signal.

For example, if you are using an external C/T clock running at 10000 Hz as the input clock source of the counter/timer, and you want to generate a output signal of 1000 Hz with a 20% duty cycle, specify a period of 10 (10000 Hz divided by 10 is 1000 Hz) and a pulse width of 2 (the period of 10 multiplied by 20%). This is illustrated in Figure 4.

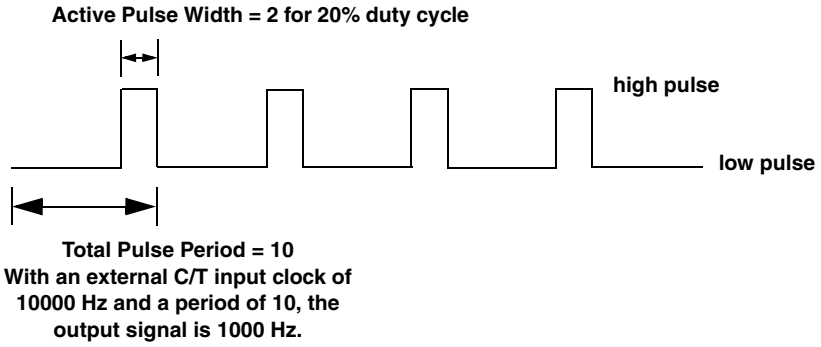


Figure 4: Example of a Pulse Output

Note: If you are using an internal C/T clock input source, you can output pulses using a maximum frequency of 24 MHz. Note, however, that the integrity of the signal degrades at frequencies greater than 10 MHz.

If you are using an external C/T clock input source, you can output pulses using a maximum frequency of 5 MHz.

Starting the Counter/Timer Operation

Once you have configured all the parameters for the counter/timer subsystem, start the counter/timer using the **ioctl - IOCTL_START_SUBSYS** command. If a software gate is specified, the operation starts immedately after the execution of this command. For other gate types, the operation starts when the specified gate type is enabled.

Reading the Counter/Timer

You can read the value of the counter using the Linux command **read**. You specify the file descriptor for the counter/timer subsystem, the variable that will store the resulting value, and the size of the variable that will store the counter value.

Stopping the Counter/Timer Operation

To stop the counter/timer operation, use the **ioctl - IOCTL_STOP_SUBSYS** command. When this command is executed, the counter/timer operation stops immediately.

Closing the Subsystem

When you are finished reading the value of the counter/timer, close the counter/timer subsystem using the **close** command, described on page 23.

Measure Counter Operations

The DT7816 module supports one measure counter whose value (a 32-bit, unsigned value) can be returned in the input stream. Using this counter, you can measure the frequency, period, or pulse width of a single signal or the time period between two signals. The measure counter is useful for correlating analog input data with digital positional data, measuring the frequency of a signal, or as a tachometer. An internal 48 MHz counter is used for the measurement, yielding a resolution of 20.83 ns (1/48 MHz).

The read the value of the measure counter in the input stream, set bit 10 of the channel mask of the input stream. Refer to page 25 for more information on analog input and input stream operations.

This section describes how to configure the parameters of the measure counter.

Opening the Subsystem

Open the file corresponding to the measure counter subsystem (/dev/DT7816-measure) using the Linux command **open**, described on page 22.

Configuring the Measure Counter Subsystem

To configure the measure counter subsystem, use the **ioctl - IOCTL_MCTR_CFG_SET** command.

You specify the file descriptor for the measure counter subsystem and then specify the following parameters:

- The signals that start and stop the measurement. Refer to Table 4 for the supported start and stop signals.

Table 4: Possible Start and Stop Signals

Signal	Connection Required
A/D conversion complete	No connection required.
Tachometer input (falling edge or rising edge)	Connect to the Tachometer input (pin 23).
Digital input 0 (falling edge or rising edge)	Connect a digital input, external A/D trigger, C/T clock input, or C/T gate input to general-purpose input 0 (pin 1) of the I/O header. By default, this is digital input 0.
Digital input 1 (falling edge or rising edge)	Connect a digital input, external A/D trigger, C/T clock input, or C/T gate input to general-purpose input 1 (pin 2) of the I/O header. By default, this is digital input 1.
Digital input 2 (falling edge or rising edge)	Connect a digital input, external A/D trigger, C/T clock input, or C/T gate input to general-purpose input 2 (pin 3) of the I/O header. By default, this is digital input 2.

Table 4: Possible Start and Stop Signals (cont.)

Signal	Connection Required
Digital input 3 (falling edge or rising edge)	Connect a digital input, external A/D trigger, C/T clock input, or C/T gate input to general-purpose input 3 (pin 4) of the I/O header. By default, this is digital input 3.
Digital input 4 (falling edge or rising edge)	Connect a digital input, external A/D trigger, C/T clock input, or C/T gate input to general-purpose input 4 (pin 5) of the I/O header. By default, this is digital input 4.
Digital input 5 (falling edge or rising edge)	Connect a digital input, external A/D trigger, C/T clock input, or C/T gate input to general-purpose input 5 (pin 6) of the I/O header. By default, this is digital input 5.
Digital input 6 (falling edge or rising edge)	Connect a digital input, external A/D trigger, C/T clock input, or C/T gate input to general-purpose input 6 (pin 7) of the I/O header. By default, this is digital input 6.
Digital input 7 (falling edge or rising edge)	Connect a digital input, external A/D trigger, C/T clock input, or C/T gate input to general-purpose input 7 (pin 8) of the I/O header. By default, this is digital input 7.

- A Stale flag (*stale_flag*) indicating whether or not the data is new. This flag is used only when the start edge and the stop edge is set to use the same pin and edge (such as pin 0 - DIN 0 rising as the start edge and pin 0 -DIN rising as the stop edge).

  If *stale_flag* is set as Used (1), the most significant bit (MSB) of the value is set to 0 to indicate new data; reading the value before the measurement is complete returns an MSB of 1. If *stale_flag* is set to Not Used (0), the MSB is always set to 0.

When the selected start edge is the same as the selected stop edge, the internal 48 MHz counter starts incrementing when it detects the first start edge of the selected input signal and stops incrementing when it detects the selected stop edge (which is the same as the start edge, in this case); at that point, the counter stores and resets the count. The stored count is maintained until it is read as part of the input data stream or until a new count is stored. Since the stop edge is the same as the start edge in this case, the stop edge for the current measurement is the start edge for the next measurement; therefore, no waveform periods are missed. The value of the measure count depends on the input subsystem sample frequency, described as follows:

- If the input subsystem sample frequency is faster than the selected input frequency, then the stored measure count retains the current value when it is read by the input subsystem. The operation of the Stale flag in this case is described as follows:
  - If another input subsystem sample occurs before another measure completes and the Stale flag is used, then the Stale flag is set and the stale measure count is written into the input data stream.
  - If another input subsystem sample occurs before another measure completes and the Stale flag is not used, then the Stale flag is not set and the stale measure count is written into the input data stream.

- If the input subsystem sample frequency is slower than the selected input frequency, then the new measure count value is stored as each period measurement completes. When an input subsystem sample occurs, then the most recently stored measure count is written into the input data stream.

When the selected start edge is not the same as the selected stop edge, the internal 48 MHz counter starts incrementing when it detects the selected start edge and stops incrementing when it detects the next selected stop edge; at that point, the counter stores and resets the count. The stored count is maintained until it is read as part of the input data stream or until a new count is stored. The value of the measure count depends on the input subsystem sample frequency, described as follows:

- If the input subsystem sample rate is faster than the selected measurement period, then the stored count retains the current value when the count is read by the input subsystem. The operation of *stale_flag* in this case is described as follows:
  - If another input subsystem sample occurs before another measure completes and *stale_flag* is used, then *stale_flag* is set and the stale measure count is written into the input data stream.
  - If another input subsystem sample occurs before another measure completes and *stale_flag* is not used, then *stale_flag* is not set and the stale measure count is written into the input data stream.
- If the input subsystem sample rate is slower than the selected measurement period, then a new count value is stored as each period measurement completes. When an input subsystem sample occurs, the most recently stored measure count is written into the input data stream.

A data pipeline is used in the hardware to compensate for the A/D group delay and synchronizes the value of the measure counter with the analog input measurements, so that all measurements are correlated in time.

When you read the value of the measure counter as part of the input data stream, you might see results similar to the following:

Table 5: An Example of Reading a Measure Counter as Part of the Input Stream

Time	A/D Value	Measure Counter Values	Status of Operation
10	5002	0	Operation started, but is not complete
20	5004	0	Operation not complete
30	5003	0	Operation not complete
40	5002	12373	Operation complete
50	5000	12373	Next operation started, but is not complete
60	5002	12373	Operation not complete

Table 5: An Example of Reading a Measure Counter as Part of the Input Stream (cont.)

Time	A/D Value	Measure Counter Values	Status of Operation
70	5004	12373	Operation not complete
80	5003	14503	Operation complete
90	5002	14503	Next operation started, but is not complete

Using the count that is returned from the measure counter, you can determine the following:

- Frequency between the start and stop signals/edges. You can calculate the frequency as follows:
  - Frequency = 48 MHz/(Number of counts – 1)
    where 48 MHz is the internal measure counter frequency

    For example, if the count is 201, the measured frequency is 240 kHz (48 MHz/200).
- Period between the start and stop signals/edges. You can calculate the period as follows:
  - Period = 1/Frequency
  - Period = (Number of counts – 1)/48 MHz
    where 48 MHz is the internal measure counter frequency
- Pulse width of the start and stop signal/edges (rising to falling edge or falling edge to rising edge). You can calculate the period as follows:
  - Pulse width = 1/Frequency
  - Pulse width = (Number of counts – 1)/48 MHz
    where 48 MHz is the internal measure counter frequency

You can return the current configuration of the measure counter subsystem using the **ioctl - IOCTL_MCTR_CFG_GET** command.

Digital Input Operations

The DT7816 module supports synchronous reads of the digital input port. You can also read the value of the digital port in the input stream by setting bit 11 of the channel mask of the input stream; refer to page 25 for more information on input stream operations.

The digital input subsystem does not need to be configured. By default, general-purpose input pins 1 to 8 of the I/O header on the DT7816 module correspond to digital input signals 0 to 7.

Note: If you assigned a general-purpose input signal as a counter clock or gate input or as an external trigger, you can read the value of the signal as you would any other digital input signal, if desired.

A digital line is high if its value is 1; a digital line is low if its value is 0. On power up or reset, a low value (0) is output from each of the digital output lines and a high value (1) is read from each of the digital input lines if the lines are not connected.

This section describes synchronous read operations of the digital input port.

Opening the Subsystem

Open the file corresponding to the digital input subsystem (/dev/DT7816-din) using the Linux command **open**, described on page 22.

Reading the Value

To perform a synchronous read of the digital input port, use the Linux command **read**, specifying the file descriptor for the digital input subsystem, the variable that will store the resulting value, and the size of the variable that will store the digital input value. For the DT7816, the digital input port is a 16-bit value.

The read operation is blocking, in that it does not return until the value is read. The operation stops automatically once the value is returned.

Closing the Subsystem

When finished, close the digital input subsystem using the **close** command, described on page 23.

Digital Output Operations

The DT7816 module supports synchronous writes to the digital output port. You can also update the individual lines of the digital output port through the output stream by setting bits 24 to 31 of the channel mask for the output stream; refer to page 33 for more information on output stream operations.

The digital output subsystem does not need to be configured. By default, general-purpose output pins 11 to 18 of the I/O header on the DT7816 module correspond to digital output signals 0 to 7.

Note: If the clock output signal of the counter/timer was assigned to one of the general-purpose output signals, you can overwrite the value of the signal by writing to the digital output subsystem. Therefore, ensure that you know the configuration of each output pin of the I/O header before writing to it or you could corrupt the signal on that pin.

A digital line is high if its value is 1; a digital line is low if its value is 0. On power up or reset, a low value (0) is output from each of the digital output lines and a high value (1) is read from each of the digital input lines if the lines are not connected.

This section describes synchronous write operations of the digital output port.

Opening the Subsystem

Open the file corresponding to the digital output subsystem (/dev/DT7816-dout) using the Linux command **open**, described on page 22.

Updating the Value of the Port

To perform a synchronous write to the digital output port, use the Linux command **write**, specifying the file descriptor for the digital output subsystem, the variable that contains the value to write, and the size of the variable that contains the digital output value. For the DT7816, the digital output port is a 16-bit value.

The write operation is blocking, in that it does not return until the value is written. The operation stops automatically once the value is written.

Closing the Subsystem

When finished, close the digital output subsystem using the **close** command, described on page 23.

Calibration

This section describes the calibration process for the analog input and analog output circuitry on the DT7816.

Note: DT7816 modules are calibrated at the factory and should not require calibration for initial use. It is recommended that you check and, if necessary, readjust the calibration of the analog circuitry every six months using the DT7816 calibration example. Refer to the *DT7816 User's Manual* for more information on this example.

This section describes the commands that the DT7816 calibration examaple uses to calibrate the module. You can write your own calibration program or modify this one, if desired, using these commands.

Analog Input Calibration

The input range of ±10 V for all channels on the DT7816 is established by a single 5 V onboard reference circuit. The overall accuracy of the measurements is affected by the gain and offset errors of each channel. The offset errors are low enough that no offset correction circuitry is provided on the module. However, the gain error must be corrected by trimming the 5 V reference; a digital gain potentiometer is provided on the module for this purpose. Varying the 5 V reference has the effect of scaling the input range above or below the ±10 V nominal voltage.

The DT7816 calibration example calibrates the voltage reference for analog input channel 0 based on the average value of the analog input channels. The accuracy of the remaining channels is dependent on the channel-to-channel matching of the ADC device, which is typically within 0.2%.

The gain potentiometer has three values: a "factory" calibration value that is programmed at the factory and represents a known accurate configuration, a "user" calibration value that you can modify, and a "wiper" value, which is the current value. In most cases, the wiper and user calibration values are the same. They differ only during the actual calibration process when the wiper value is modified until the desired value is reached.

This section describes how to calibrate the analog input circuitry using software commands.

Note: An external, precision voltage source is required to calibrate the analog input circuitry of the DT7816.

Opening the Subsystem

Open the file corresponding to the analog input subsystem (/dev/DT7816-ain) using the Linux command **open**, described on page 22.

Calibrating the Gain

The DT7816 calibration example uses the following steps to calibrate the gain of the analog input subsystem:

1. Short analog input channels 0 to 7 to analog ground.
2. Acquire a block of 1000 samples from the analog input channels at a sampling frequency of 200 kHz and compute the average ADC count value. This value should be within 10 counts of 0x0000.
3. Note the count value (called *Kos*) relative to the nominal zero count, which is halfway between 0xFFFF and 0x0000. This value can be positive or negative.
4. Remove the shorting terminations and connect a precision voltage source of +9.000 V to analog input channels 0 to 7.
5. Acquire and average a block of data and sequentially adjust the gain potentiometer for the analog input channel 0 using the **ioctl - IOCTL_GAIN_POT_SET** command until the target value, which is 0x72AE (29358 decimal) plus *Kos*, is reached.

   In the **ioctl - IOCTL_GAIN_POT_SET** command, you specify the potentiometer that you want to calibrate (for the analog input subsystem of the DT7816, this is always 0), the value to write to the potentiometer, whether the calibration type is a user or factory calibration, and the calibration register to update (for the DT7816, this value the register is always 0).

   The gain calibration register is automatically written to the wiper of the associated potentiometer.

   Note: This command will block all other operations for at least 5 ms.

6. When you are satisfied that the gain calibration value is correct, call the **ioctl - IOCTL_GAIN_POT_WIPER_SET** command. You specify the analog input channel (potentiometer) that you want to calibrate and the value to write to the potentiometer. The value is written to a calibration register in non-volatile EEPROM.

You can read the value of the gain potentiometers using the **ioctl - IOCTL_GAIN_POT_GET** and **ioctl - IOCTL_GAIN_POT_WIPER_GET** commands.

Closing the Subsystem

When finished, close the analog input subsystem using the **close** command, described on page 23.

Analog Output Calibration

Each channel of the analog output subsystem on the DT7816 requires offset and gain calibration.

Two offset potentiometers are provided: gain potentiometer 0 is for analog output channel 0 and gain potentiometer 1 is for analog output channel 1. Similarly, two gain potentiometers are also provided: offset potentiometer 0 for analog output channel 0 and offset potentiometer 1 for analog output channel 1.

Each potentiometer has three values: a "factory" calibration value that is programmed at the factory and represents a known accurate configuration, a "user" calibration value that you can modify, and a "wiper" value, which is the current value. In most cases, the wiper and user calibration values are the same. They differ only during the actual calibration process when the wiper value is modified until the desired value is reached.

This section describes how to calibrate the analog output circuitry using software commands.

Note: An external, precision digital multimeter (DMM) or voltmeter (DVM) is required to calibrate the analog output circuitry of the DT7816.

Opening the Subsystem

Open the file corresponding to the analog output subsystem (/dev/DT7816-aout) using the Linux command **open**, described on page 22.

Calibrating the Gain

The gain must be calibrated before the offset for each analog output channel.

To calibrate the gain for an analog output channel, perform the following steps:

1. Connect a DMM to the analog output channel.
2. Output a voltage of –9.9 V to the analog output channel.
3. Note the voltage that is reported on the DMM to the nearest 10 µV.
4. Output a voltage of +9.9 V to the analog output channel.
5. Use the **ioctl - IOCTL_GAIN_POT_SET** command to adjust the gain potentiometer for the analog output channel until the magnitude of the reading on the DMM is within 10 µV of 19.8 V plus the value that was noted in step 3.

   In the **ioctl - IOCTL_GAIN_POT_SET** command, you specify the analog output channel that you want to calibrate, the value to write to the potentiometer, whether the calibration type is a user or factory calibration, and the calibration register to update (for the DT7816, this value the register is always 0).

   The gain calibration register is automatically written to the wiper of the associated potentiometer.

Note: This command will block all other operations for at least 5 ms.

6. When you are satisfied that the gain calibration value is correct, call the **ioctl - IOCTL_GAIN_POT_WIPER_SET** command. You specify the analog input channel (potentiometer) that you want to calibrate and the value to write to the potentiometer. The value is written to a calibration register in non-volatile EEPROM.

Once the gain has been calibrated for the analog output channel, calibrate the offset for the same channel. Then, repeat this procedure for the remaining analog output channel.

Note that you can read the value of the gain potentiometers using the **ioctl - IOCTL_GAIN_POT_GET** and **ioctl - IOCTL_GAIN_POT_WIPER_GET** commands.

Calibrating the Offset

Once the gain is calibrated for an analog output channel, calibrate the offset for the same channel by performing the following steps:

1. With a DMM connected to the analog output channel, output a voltage of 0 V to the analog output channel.
2. Use the **ioctl - IOCTL_OFFSET_POT_SET** command to adjust the offset potentiometer for the analog output channel until the reading on the DMM is within 10 µV of 0 V.

   In the **ioctl - IOCTL_OFFSET_POT_SET** command, specify the analog output channel that you want to calibrate, the value to write to the potentiometer, whether the calibration type is a user (0) or factory (1) calibration, and the calibration register to update (for the DT7816, this value is always 0).
3. When you are satisfied that the offset calibration value is correct, call the **ioctl - IOCTL_OFFSET_POT_WIPER_SET** command. You specify the analog output channel (potentiometer) that you want to calibrate and the value to write to the potentiometer. The value is written to a calibration register in non-volatile EEPROM.

Once the offset calibration is complete, repeat the gain and offset calibration process for the remaining analog output channel.

Note that you can read the value of the offset potentiometers using the **ioctl - IOCTL_OFFSET_POT_GET** and **ioctl - IOCTL_OFFSET_POT_WIPER_GET** commands.

Closing the Subsystem

When finished, close the analog output subsystem using the **close** command, described on page 23.

Modifying the State of the User LEDs on the Module

The DT7816 has eight user LEDs, shown in Figure 5. Header J8, also shown in Figure 5, provides debug pins that correspond to the user LEDs. The value of a debug pin reflects the state of the corresponding user LED, where the pin has a value of 0 if the LED is off or a value of 1 if the LED is on.

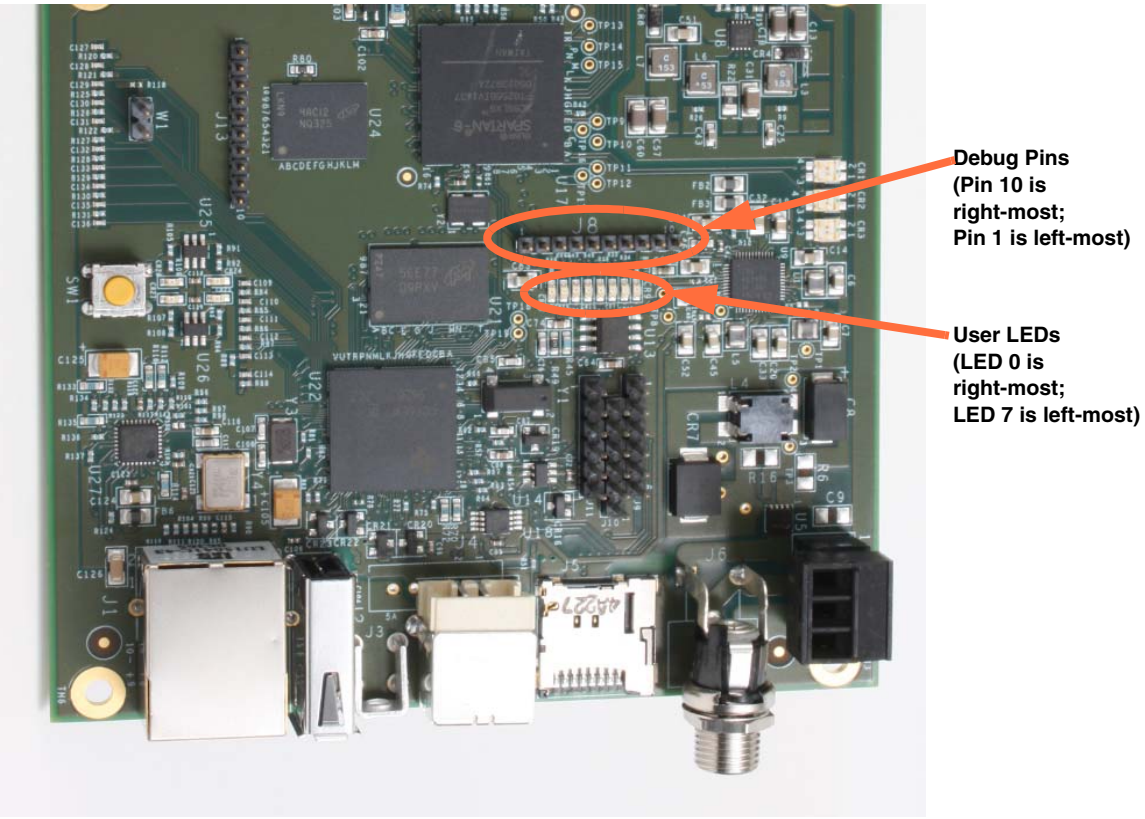


Figure 5: User LEDs and Debug Pins on the DT7816 Module

Table 6 lists the pin descriptions of header J8 on the DT7816.

Table 6: Debug Pins of Header J8

Pin	Pin Description
1	Debug_D7; corresponds to user LED 7.
2	Debug_D6; corresponds to user LED 6.
3	Debug_D5; corresponds to user LED 5.
4	Debug_D4; corresponds to user LED 4.
5	Debug_D3; corresponds to user LED 3.
6	Debug_D2; corresponds to user LED 2.
7	Debug_D1; corresponds to user LED 1.
8	Debug_D0; corresponds to user LED 0.
9	Digital Ground
10	Digital Ground

You can turn the LEDs either on or off using the **ioctl - IOCTL_LED_SET** command. You can also return the state of the user LEDs using the **ioctl - IOCTL_LED_GET** command.

Note that to execute these commands, you must first open a file corresponding to any of the supported subsystems of the DT7816 (/dev/DT7816-ain, dev/DT7816-din, /dev/DT7816-dout, /dev/DT7816-tach, dev/DT7816-measure, /dev/DT7816-ctr-tmr, or /dev/DT7816_stream_in) using the Linux command **open**, described on page 22. When finished, close the file using the **close**, described on page 23.

Sending Data to or Receiving Data from the Host USB Application

The kernel module for the DT7816 USB gadget driver is a simplified version of gadgetfs and has the same kind of functionality. When loaded, it enumerates as a USB gadget, creates a specified number of bulk USB endpoints, and exposes each end point as a virtual file in the **/dev** directory.

A user mode application can perform file operations on each virtual file to send data to the USB host through an IN endpoint and receive data from the USB host through an OUT endpoint. The module serves as a conduit for data and does not care about the data itself.

Note that the directions are relative to the USB host. Therefore, to send data to the host USB, you must write to an IN endpoint. To receive data from a USB host, you must read data from an OUT endpoint.

This section describes file I/O operations used by the USB gadget driver to send data to and receive data from the USB host.

Opening the File

Open the file corresponding to the endpoint that you want to open using the Linux command **open**, described on page 22.

The files are named as follows:

- /dev/DT7816-ep1in
- /dev/DT7816-ep1out
- /dev/DT7816-ep2in
- /dev/DT7816-ep2out
- /dev/DT7816-ep3out
- /dev/DT7816-ep4out
- /dev/DT7816-ep5out

A file can be opened by only one reader/writer at a time. However, different files can be opened simultaneously. Once a file is closed, it can be re-opened.

Sending Data to the USB Host

To send data to the USB host, you must write to file corresponding to an IN endpoint.

If you want to perform a blocking write, use the Linux command **write**. Because it is blocking, no other operation can be performed until the **write** operation is complete.

If you want to perform a non-blocking, asynchronous write operation, use the Linux command **aio_write**. Because it is non-blocking, other operations can be performed while the **aio_write** operation is in progress.

Receiving Data from the USB Host

To receive data from a USB host, you must read data from a file corresponding to an OUT endpoint.

If you want to perform a blocking read operation, use the Linux command **read**. Because it is blocking, no other operation can be performed until the **read** operation is complete.

If you want to perform a non-blocking, asynchronous read operation, use the Linux command **aio_read**. Because it is non-blocking, other operations can be performed while the **aio_read** operation is in progress.

Eliminating Data from the Endpoint

To eliminate the data from either an IN or OUT endpoint, use the Linux command **flush**.

Closing the File

When you are finished using the file, close the file using the **close** command, described on page 23.

File I/O Command Reference

Note: Refer to your Linux documentation for information on all other Linux commands, including **read**, **write**, **aio_read, aio_write, io_setup**, **io_submit**, **io_getevents**, **io_cancel**, and **io_destroy**.

close

Description Closes a file descriptor that is associated with a subsystem, stream, or endpoint of the DT7816 module.

Syntax

```
int close(int fd);
```

Arguments

Name: fd

Data Type: int

Description: The file descriptor of the subsystem, stream, or endpoint to close.

Returns 0 = Success; –1 = Failure

Notes You must open each subsystem, stream, or endpoint that you want to use with the **open** command, described on page 144, before you can configure its parameters and perform an I/O operation.

Example The following command closes the file associated with the input stream of the DT7816:

```
#define DEV_STREAM_IN "/dev/DT7816-stream-in"

close(DEV_STREAM_IN);
```

See Also **open**, described on page 144.

ioctl - IOCTL_ACQ_STATUS_GET

Description Returns the status of acquisition or the status of the output operation.

Syntax

```
int ioctl(int fd, IOCTL_ACQ_STATUS_GET,
   acq_status_t *pAcqStatus);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the input stream (/dev/DT7816-stream-in) or the output stream (/dev/DT7816-stream-out).

Name: pAcq_Status

Data Type: acq_status_t enumeration

Description: A pointer to a variable that specifies the status of acquisition (if the file /dev/DT7816-stream-in was specified as the file descriptor) or the status of the output operation (if the file /dev/DT7816-stream-out was specified as the file descriptor).

The acq_status_t enumeration is described as follows:

```
typedef enum
{
  acq_status_armed      = 0x01,
  acq_status_triggered  = 0x02,
  acq_status_fifo_empty = 0x04,
  acq_status_fifo_full  = 0x08,
}acq_status_t;
```

where,

- *acq_status_armed* specifies that the associated stream is armed and waiting for a trigger (value 0x1).
- *acq_status_triggered* specifies that the associated stream has been triggered and is active (value 0x2).
- *acq_status_fifo_empty* indicates that the FIFO is empty (no data) for the associated stream (value 0x4).
- *acq_status_fifo_full* indicates that the FIFO is full of data for the associated stream.

Returns 0 = Success; < 0 = Failure

Example The following example returns the status of acquisition for the input stream:

```
acq_status_t Acq_Status;
if (ioctl(fd_instream, IOCTL_ACQ_STATUS_GET,
   &Acq_Status))
return (printf_errno_html(conn, errno,
   "IOCTL_ACQ_STATUS_GET"));
```

ioctl - IOCTL_AIN_CNF_GET

Description Returns the configuration of the analog input subsystem.

Syntax

```
int ioctl(int fd, IOCTL_AIN_CFG_GET,
   struct dt78xx_ain_config_t *pAIN_config);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the analog input subsystem (/dev/DT7816-ain).

Name: pAIN_config

Data Type: dt78xx_ain_config_t structure

Description: A pointer to a structure that defines the configuration of the analog input subsystem.

dt78xx_ain_config_t is defined as follows:

```
typedef struct __attribute__ ((__packed__))
{
  uint16_t ain;
  uint16_t gain;
  uint16_t ac_coupling;
  uint16_t current_on;
  uint16_t differential;
  uint16_t unused;
} dt78xx_ain_config_t;
```

Structure Element Name: ain

Data Type: uint16_t

Description: Specifies which analog input channel to configure. For the DT7816, values range from 0 to 7, where 0 represents analog input channel 0 and 7 represents analog input channel 7.

Structure Element Name: gain

Data Type: uint16_t

Description: Specifies the gain to use for the specified analog input channel. For the DT9816, this value is 1 for a gain of 1.

Structure Element Name: ac_coupling

Data Type: uint16_t

Description: Specifies the coupling type for the specified analog input channel. For the DT7816, this value is ignored.

Structure Element Name: current_on

Data Type: uint16_t

Description: Specifies whether the 4 mA current source is on or off for the specified analog input channel. For the DT7816, this value is ignored.

Structure Element Name: differential

Data Type: uint16_t

Description: Specifies whether the specified analog input channel is a differential input or a single-ended input. For the DT7816, this value is 0 for single-ended inputs.

Structure Element Name: unused

Data Type: uint16_t

Description: Reserved for future use.

Returns 0 = Success; < 0 = Failure

Example The following example returns the configuration of the analog input subsystem:

```
dt78xx_ain_config_t AIN_cfg;
if (ioctl(fd_ain, IOCTL_AIN_CFG_GET, &AIN_cfg))
    {
        perror("IOCTL_AIN_CFG_GET");
        goto finish;
    }
```

See Also **IOCTL_AIN_CFG_SET**, described on page 70.

ioctl - IOCTL_AIN_CNF_SET

Description Specifies the configuration of the analog input subsystem.

Syntax

```
int ioctl(int fd, IOCTL_AIN_CFG_SET,
   struct dt78xx_ain_config_t *pAIN_config);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the analog input subsystem (/dev/DT7816-ain).

Name: pAIN_config

Data Type: dt78xx_ain_config_t structure

Description: A pointer to a structure that defines the configuration of the analog input subsystem.

dt78xx_ain_config_t is defined as follows:

```
typedef struct __attribute__ ((__packed__))
{
  uint16_t ain;
  uint16_t gain;
  uint16_t ac_coupling;
  uint16_t current_on;
  uint16_t differential;
  uint16_t unused;
} dt78xx_ain_config_t;
```

Structure Element Name: ain

Data Type: uint16_t

Description: Specifies which analog input channel to configure. For the DT9816, values range from 0 to 7, where 0 represents analog input channel 0 and 7 represents analog input channel 7.

Structure Element Name: gain

Data Type: uint16_t

Description: Specifies the gain to use for the specified analog input channel. For the DT7816, this value is 1 for a gain of 1.

Structure Element Name: ac_coupling

Data Type: uint16_t

Description: Specifies the coupling type for the specified analog input channel. For the DT7816, this value is ignored.

Structure Element Name: current_on

Data Type: uint16_t

Description: Specifies whether the 4 mA current source is on or off for the specified analog input channel. For the DT7816, this value is ignored.

Structure Element Name: differential

Data Type: uint16_t

Description: Specifies whether the specified analog input channel is a differential input or a single-ended input. For the DT7816, this value is 0 for single-ended inputs.f

Structure Element Name: unused

Data Type: uint16_t

Description: Reserved for future use.

Returns 0 = Success; < 0 = Failure

Example The following example specifies the configuration of the analog input subsystem. In this case, analog input channel 0 is configured with a gain of 1.

```
#define DEV_AIN "/dev/DT7816-ain"
fd_ain = open(DEV_AIN, O_RDWR);

dt78xx_ain_config_t AIN_cfg;
AIN_cfg.ain = 0;
AIN_cfg.gain = 1; //x1 gain
AIN_cfg.ac_coupling = 0; //ignored
AIN_cfg.current_on = 0; //ignored
if (ioctl(fd_ain, IOCTL_AIN_CFG_SET, &AIN_cfg))
  {
    fprintf(stderr, "IOCTL_AIN_CFG_SET ERROR
      %d \"%s\"\n",  errno, strerror(errno));
        goto finish;
    }
```

See Also **IOCTL_AIN_CFG_GET**, described on page 68.

ioctl - IOCTL_ARM_SUBSYS

Description Arms the input or output stream to detect the trigger that will start the continuous operation.

Syntax

```
int ioctl(int fd, IOCTL_ARM_SUBSYS, int unused);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the input stream (/dev/DT7816-stream-in) or output stream (/dev/DT7816-stream-out).

Name: unused

Data Type: int

Description: Not used; reserved for future use.

Returns 0 = Success; < 0 = Failure

Example The following example arms the input stream, *fd_instream*, that was previously opened:

```
tmp = 0;
if (ioctl(fd_in_stream, IOCTL_ARM_SUBSYS, &tmp))
    {
     err = errno;
     return (printf_errno(conn, err,
       "IOCTL_ARM_SUBSYS"));
    }
```

See Also **ioctl - IOCTL_START_SUBSYS**, described on page 129.

ioctl - IOCTL_CHAN_MASK_GET

Description Returns the channels that are enabled for acquisition in the input stream or that are enabled for output in the output stream.

Syntax

```
int ioctl(int fd, IOCTL_CHAN_MASK_GET,
   chan_mask_t *pChannels);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the input stream (/dev/DT7816-stream-in) or output stream (/dev/DT7816-stream-out).

Name: *pChannels

Data Type: chan_mask_t enumeration

Description: A pointer to a 32-bit value that specifies the channels that are enabled for acquisition in the input stream or enabled for output in the output stream. The *channel_mask_t* enumeration, described below, is used to specify the values associated with the channels in the input stream and output stream:

```
typedef enum {
  chan_mask_dout7      = (1<<31),
  chan_mask_dout6      = (1<<30),
  chan_mask_dout5      = (1<<29),
  chan_mask_dout4      = (1<<28),
  chan_mask_dout3      = (1<<27),
  chan_mask_dout2      = (1<<26),
  chan_mask_dout1      = (1<<25),
  chan_mask_dout0      = (1<<24),
  chan_mask_aout3      = (1<<19)
  chan_mask_aout2      = (1<<18),
  chan_mask_aout1      = (1<<17),
  chan_mask_aout0      = (1<<16),
  chan_mask_din        = (1<<11),
  chan_mask_meas_ctr   = (1<<10),
  chan_mask_tach       = (1<<8),
  chan_mask_ain7       = (1<<7),
  chan_mask_ain6       = (1<<6),
  chan_mask_ain5       = (1<<5),
  chan_mask_ain4       = (1<<4)
  chan_mask_ain3       = (1<<3),
  chan_mask_ain2       = (1<<2),
  chan_mask_ain1       = (1<<1),
  chan_mask_ain0       = (1<<0),
} chan_mask_t
```

Description (cont.): where,

- *chan_mask_dout7 i*s the line 7 of the digital output port (bit 31 - value 0x80000000).
- *chan_mask_dout6 i*s the line 6 of the digital output port (bit 30 - value 0x40000000).
- *chan_mask_dout5 i*s the line 5 of the digital output port (bit 29 - value 0x20000000).
- *chan_mask_dout4 i*s the line4 of the digital output port (bit 28 - value 0x10000000).
- *chan_mask_dout3 i*s the line3 of the digital output port (bit 27 - value 0x8000000).
- *chan_mask_dout2 i*s the line1 of the digital output port (bit 26 - value 0x4000000).
- *chan_mask_dout1 i*s the line1 of the digital output port (bit 25 - value 0x2000000).
- *chan_mask_dout0 i*s the line 0 of the digital output port (bit 24 - value 0x1000000).
- *chan_mask_aout3* is analog output 3 (bit 19 - value 0x80000). Note that the DT7816 does not support this channel.
- *chan_mask_aout2* is analog output 2 (bit 18 - value 0x40000). Note that the DT7816 does not support this channel.
- *chan_mask_aout1* is analog output 1 (bit 17 - value 0x20000).
- *chan_mask_aout0* is analog output 0 (bit 16 - value 0x10000).
- *chan_mask_din i*s the digital input port (bit 11 - value 0x800).
- *chan_mask_meas_ctr* is the measure counter (bit 10 - value 0x400).
- *chan_mask_tach* is the tachometer input (bit 8 - value 0x100).
- *chan_mask_ain7* is analog input channel 7 (bit 7 - value 0x80)
- *chan_mask_ain6* is analog input channel 6 (bit 6 - value 0x40)
- *chan_mask_ain5* is analog input channel 5 (bit 5 - value 0x20)
- *chan_mask_ain4* is analog input channel 4 (bit 4 - value 0x10)
- *chan_mask_ain3* is analog input channel 3 (bit 3 - value 0x8).
- *chan_mask_ain2* is analog input channel 2 (bit 2 - value 0x4).
- *chan_mask_ain1* is analog input channel 1 (bit 1 - value 0x2).
- *chan_mask_ain0* is analog input channel 0 (bit 0 - value 0x1).

Returns 0 = Success; < 0 = Failure

Example The following example returns the channels that were enabled in the input stream, *fd_instream*:

```
ret = ioctl(fd_instream, IOCTL_CHAN_MASK_GET,
   &chan_mask);
```

See Also **IOCTL_CHAN_MASK_SET**, described on page 76.

ioctl - IOCTL_CHAN_MASK_SET

Description Specifies the channels to enable for acquisition in the input stream or enabled for output in the output stream.

Syntax

```
int ioctl(int fd, IOCTL_CHAN_MASK_SET,
   chan_mask_t *pChannels);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the input stream (/dev/DT7816-stream-in) or output stream (/dev/DT7816-stream-out).

Name: *pChannels

Data Type: chan_mask_t enumeration

Description: A pointer to a 32-bit value that specifies the channels that are enabled for acquisition in the input stream or enabled for output in the output stream. The *channel_mask_t* enumeration, described below, is used to specify the values associated with the channels in the input stream and output stream:

```
typedef enum {
  chan_mask_dout7      = (1<<31),
  chan_mask_dout6      = (1<<30),
  chan_mask_dout5      = (1<<29),
  chan_mask_dout4      = (1<<28),
  chan_mask_dout3      = (1<<27),
  chan_mask_dout2      = (1<<26),
  chan_mask_dout1      = (1<<25),
  chan_mask_dout0      = (1<<24),
  chan_mask_aout3      = (1<<19)
  chan_mask_aout2      = (1<<18),
  chan_mask_aout1      = (1<<17),
  chan_mask_aout0      = (1<<16),
  chan_mask_din        = (1<<11),
  chan_mask_meas_ctr   = (1<<10),
  chan_mask_tach       = (1<<8),
  chan_mask_ain7       = (1<<7),
  chan_mask_ain6       = (1<<6),
  chan_mask_ain5       = (1<<5),
  chan_mask_ain4       = (1<<4)
  chan_mask_ain3       = (1<<3),
  chan_mask_ain2       = (1<<2),
  chan_mask_ain1       = (1<<1),
  chan_mask_ain0       = (1<<0),
} chan_mask_t
```

Description (cont.): where,

- *chan_mask_dout7 i*s the line 7 of the digital output port (bit 31 - value 0x80000000).
- *chan_mask_dout6 i*s the line 6 of the digital output port (bit 30 - value 0x40000000).
- *chan_mask_dout5 i*s the line 5 of the digital output port (bit 29 - value 0x20000000).
- *chan_mask_dout4 i*s the line4 of the digital output port (bit 28 - value 0x10000000).
- *chan_mask_dout3 i*s the line3 of the digital output port (bit 27 - value 0x8000000).
- *chan_mask_dout2 i*s the line1 of the digital output port (bit 26 - value 0x4000000).
- *chan_mask_dout1 i*s the line1 of the digital output port (bit 25 - value 0x2000000).
- *chan_mask_dout0 i*s the line 0 of the digital output port (bit 24 - value 0x1000000).
- *chan_mask_aout3* is analog output 3 (bit 19 - value 0x80000). Note that the DT7816 does not support this channel.
- *chan_mask_aout2* is analog output 2 (bit 18 - value 0x40000). Note that the DT7816 does not support this channel.
- *chan_mask_aout1* is analog output 1 (bit 17 - value 0x20000).
- *chan_mask_aout0* is analog output 0 (bit 16 - value 0x10000).
- *chan_mask_din i*s the digital input port (bit 11 - value 0x800).
- *chan_mask_meas_ctr* is the measure counter (bit 10 - value 0x400).
- *chan_mask_tach* is the tachometer input (bit 8 - value 0x100).
- *chan_mask_ain7* is analog input channel 7 (bit 7 - value 0x80)
- *chan_mask_ain6* is analog input channel 6 (bit 6 - value 0x40)
- *chan_mask_ain5* is analog input channel 5 (bit 5 - value 0x20)
- *chan_mask_ain4* is analog input channel 4 (bit 4 - value 0x10)
- *chan_mask_ain3* is analog input channel 3 (bit 3 - value 0x8).
- *chan_mask_ain2* is analog input channel 2 (bit 2 - value 0x4).
- *chan_mask_ain1* is analog input channel 1 (bit 1 - value 0x2).
- *chan_mask_ain0* is analog input channel 0 (bit 0 - value 0x1).

Returns 0 = Success; < 0 = Failure

Notes

The order of the data in the input buffer is as follows, assuming that all channels are enabled in the input stream:

- Analog input channels 0 through 7. Each analog input sample is a 16-bit, two's complement value.
- Tachometer input. This is a 32-bit unsigned value.
- Measure/counter. This is a 32-bit unsigned value.
- Digital input port. This is 16-bit unsigned value. The digital input data is in the least significant eight bits.

The order of the data in the output buffer is as follows, assuming that all channels are enabled in the output stream:

- Analog output channels 0 and 1. Each analog output sample is a 16-bit value, two's complement value.
- Digital output port. This is a 16-bit unsigned value. The digital output data is in the least significant eight bits.

Example

The following example specifies the channels to enable in the input stream. In this example, analog input channels 0 and 2 are enabled in the input stream; this equivalent to value 0x5:

```
chan_mask_t chan_mask = chan_mask_ain0|chan_mask_ain2;
if (ioctl(fd_stream, IOCTL_CHAN_MASK_SET, &chan_mask))
    {
      fprintf(stderr, "IOCTL_CHAN_MASK_SET ERROR
        %d \"%s\"\n",  errno, strerror(errno));
        goto _cleanup;
```

See Also

IOCTL_CHAN_MASK_GET, described on page 73.

ioctl - IOCTL_CT_CFG_GET

Description Returns the configuration of the counter/timer channel.

Syntax

```
int ioctl(int fd, IOCTL_CT_CFG_GET,
   struct dt78xx_ct_config_t *pCT_config);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the counter/timer subsystem (/dev/DT7816-ctr-tmr).

Name: pCT_config

Data Type: dt78xx_ct_config_t structure

Description: A pointer to a structure that defines the configuration of the counter/timer channel.

dt78xx_ct_config_t is defined as follows:

```
typedef struct __attribute__ ((__packed__))
{
  ct_mode_t    mode;
  ct_gate_t    gate;
  uint8_t      ext_gate_din;
  uint8_t      ext_clk;
  uint8_t      ext_clk_din;
  uint8_t      unused;
  union
  {
     struct __attribute__ ((__packed__))
     {
       uint32_t     period;
       uint32_t     pulse;
       uint8_t      out;
       uint8_t      out_hi;
     }divider;

     struct __attribute__ ((__packed__))
     {
       uint32_t     period;
       uint32_t     pulse;
       uint8_t      out;
       uint8_t      out_hi;
       uint8_t      retriggerable;
     }one_shot;
  };
}dt78xx_ct_config_t;
```

Structure Element Name: mode

Data Type: ct_mode_t

Description: This enumeration defines the operation mode of the counter/timer channel. It is defined as follows:

```
typedef enum
{
  ct_mode_idle,
  ct_mode_1shot,
  ct_mode_divider,
  ct_mode_counter,
  ct_mode_measure,
}ct_mode_t;
```

where,

- *ct_mode_idle* specifies idle mode. If you specify this mode, the counter no longer drives the clock output signal that is assigned to one of the general-purpose output signal (pins 11 to 18) of the I/O header.
- *ct_mode_1shot* specifies one-shot mode. If you specify this mode, values for the *one_shot* structure specified in *dt78xx_ct_config_t* are required. Specify one-shot mode if you want to generate non-retriggerable one-shot pulses.
- *ct_mode_divider* specifies rate generation mode. If you specify this mode, values for the *divider* structure specified in *dt78xx_ct_config_t* are required. In rate generation mode, a continuous, active high, pulse output signal is generated from the counter's output signal. You can use this pulse output signal as an external clock to pace other operations, such as analog input, analog output, or other counter/timer operations. The pulse output operation starts whenever the counter's gate signal is at the specified level.
- *ct_mode_counter* specifies event counting mode. In event counting mode, the number of rising edges that occur on the counter's clock input signal are counted when the counter's gate signal is active. The module can count a maximum of 4,294,967,296 events before the counter rolls over to 0 and starts counting again.
- *ct_mode_measure* is currently not supported.

Structure Element Name: gate

Data Type: ct_gate_t enumeration

Description: Defines the gate type to use for the specified counter/timer operation mode. It is defined as follows:

```
typedef enum
{
  ct_gate_none = 0x1,
  ct_gate_ext_hi = 0x2,
  ct_gate_ext_lo = 0x3
}ct_gate_t;
```

where,

- *ct_gate_none* specifies a software gate type. With this gate type, the **ioctl - IOCTL_START_SUBSYS** command, described on page 129, starts the counter/timer operation immediately after execution. (No general-purpose input signal is required if this gate type is selected.) To stop the operation, use the **ioctl - IOCTL_START_SUBSYS** command, described on page 139.
- *ct_gate_ext_hi* specifies a logic high or rising edge gate type. For event counting (*counter*) and rate generation (*divider*) mode, the operation is enabled when the counter's gate signal is high and is disabled when the counter's gate signal is low. For one-shot and repetitive one-shot (*1shot*) mode, the operation is enabled when the counter's gate signal goes from a low to a high transition and is disabled when the counter's gate signal goes from a high to a low transition.
- *ct_gate_ext_lo* specifies a logic low or falling edge gate type. For event counting (*counter*) and rate generation (*divider*) mode, the operation is enabled when the counter's gate signal is low and is disabled when the counter's gate signal is high. For one-shot and repetitive one-shot mode (*1shot*) mode, the operation is enabled when the counter's gate signal goes from a high to a low transition and is disabled when the counter's gate signal goes from a low to a high transition.

Structure Element Name: ext_gate_din

Data Type: uint8_t

Description: Specifies which general-purpose input signal (pins 1 to 8) on the I/O header to use for the external C/T gate input signal.

Pin 1 corresponds to bit 0 of the digital input port (value 0x1).

Pin 2 corresponds to bit 1 of the digital input port (value 0x2).

Pin 3 corresponds to bit 2 of the digital input port (value 0x4).

Pin 4 corresponds to bit 3 of the digital input port (value 0x8).

Pin 5 corresponds to bit 4 of the digital input port (value 0x10).

Pin 6 corresponds to bit 5 of the digital input port (value 0x20).

Pin 7 corresponds to bit 6 of the digital input port (value 0x40).

Pin 8 corresponds to bit 7 of the digital input port (value 0x80).

Structure Element Name: ext_clk

Data Type: uint8_t

Description: Specifies whether to use the internal C/T clock or an external C/T clock. When this value is 0, the internal C/T clock is used. When this value is 1, an external C/T clock is used.

Note that in one-shot mode, the internal C/T clock is more useful than an external C/T clock. However, in event counting mode, up/down counting, and measure mode, an external C/T clock source is generally used.

Structure Element Name: ext_clk_din

Data Type: uint8_t

Description: Specifies which general-purpose input signal (pins 1 to 8) on the I/O header to use for the external C/T clock input signal.

Pin 1 corresponds to bit 0 of the digital input port (value 0x1).

Pin 2 corresponds to bit 1 of the digital input port (value 0x2).

Pin 3 corresponds to bit 2 of the digital input port (value 0x4).

Pin 4 corresponds to bit 3 of the digital input port (value 0x8).

Pin 5 corresponds to bit 4 of the digital input port (value 0x10).

Pin 6 corresponds to bit 5 of the digital input port (value 0x20).

Pin 7 corresponds to bit 6 of the digital input port (value 0x40).

Pin 8 corresponds to bit 7 of the digital input port (value 0x80).

Structure Element Name: unused

Data Type: uint8_t

Description: This variable is reserved for future use.

Structure Element Name: period

Data Type: uint32_t

Description: Used by the *divider* and *one_shot* structures, specifies the number of input clock cycles used to create one period of the C/T output signal.

The period of the output pulse is determined by the C/T clock source (*ext_clk* element of the *dt78xx_ct_config_t* structure, described on page 82.) You can output pulses using a maximum frequency of 24 MHz (if using the internal C/T clock) or 5 MHz (if using the external C/T clock). Note, however, that the integrity of the signal degrades at frequencies greater than 10 MHz.

For example, if you are using an external C/T clock of10000 Hz as the input clock source of the counter/timer, and you want to generate a output signal of 1000 Hz with a 20% duty cycle, specify a period of 10 (10000 Hz divided by 10 is 1000 Hz) and a pulse width of 2 (the period multiplied by 20%). This is illustrated in Figure 6.

Note that the polarity of the output pulse can be programmed using the *out_hi* element of the *divider* or *one_shot* structure.

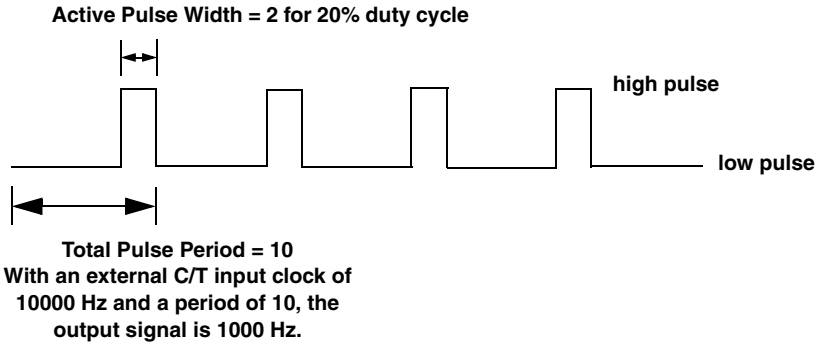


Figure 6: Example of a Pulse Width and Period

Structure Element Name: pulse

Data Type: uint32_t

Description: Used by the *divider* and *one_shot* structures, specifies the number of input clock cycles used to create the active pulse width (or duty cycle) of the C/T output signal. This value must be less than the *period*. See Figure 6 for an example.

Structure Element Name:	out
Data Type:	uint8_t
Description:	Used by the *divider* and *one_shot* structures, specifies which general-purpose output signal (pins 11 to 18) on the I/O header to use for the external C/T clock output signal. Pin 11 corresponds to bit 0 of the digital output port (value 0x1). Pin 12 corresponds to bit 1 of the digital output port (value 0x2). Pin 13 corresponds to bit 2 of the digital output port (value 0x4). Pin 14 corresponds to bit 3 of the digital output port (value 0x8). Pin 15 corresponds to bit 4 of the digital output port (value 0x10). Pin 16 corresponds to bit 5 of the digital output port (value 0x20). Pin 17 corresponds to bit 6 of the digital output port (value 0x40). Pin 18 corresponds to bit 7 of the digital output port (value 0x80).
Structure Element Name:	out_hi
Data Type:	uint8_t
Description:	Used by the *divider* and *one_shot* structures, specifies whether the output pulse for a rate generation or non-retriggerable one-shot operation is active high (1) or active low (0).
Structure Element Name:	retriggerable
Data Type:	uint8_t
Description:	Used by the *one-shot* structure, specifies whether or not to repeat the one-shot output pulse. Currently, only a value of 0 is supported. When *retriggerable* is 0, the counter generates a single pulse output signal whenever it detects the active gate signal (after the pulse period from the previous output pulse expires). Any gate signals that occur while the pulse is being output are not detected by the module. The module continues to output pulses until you stop the operation with **ioctl - IOCTL_STOP_SUBSYS**, described on page 139. You can use this mode to clean up a poor clock input signal by changing its pulse width, and then outputting it.
Returns	0 = Success; < 0 = Failure
Notes	The value of the counter clock output signal (assigned to one of the general-purpose output signals) can be overwritten by writing to the digital output subsystem. Refer to page 53 for more information. If you assigned a general-purpose input signal as a counter clock or gate input (or external trigger), you can read the value of the signal as you would any other digital input signal. Refer to page 52 for more information on reading digital input values.

Example The following example returns the configuration of the counter/timer channel:

```
if (ioctl(ct_file, IOCTL_CT_CFG_GET, &ct_cfg))
    {
        perror("IOCTL_CT_CFG_GET");
        goto finish;
    }
```

See Also **IOCTL_CT_CFG_SET**, described on page 86.

ioctl - IOCTL_CT_CFG_SET

Description Specifies the configuration of the counter/timer channel.

Syntax

```
int ioctl(int fd, IOCTL_CT_CFG_SET,
   struct dt78xx_ct_config_t *pCT_config);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the counter/timer subsystem (/dev/DT7816-ctr-tmr).

Name: pCT_config

Data Type: dt78xx_ct_config_t structure

Description: A pointer to a structure that defines the configuration of the counter/timer channel.

dt78xx_ct_config_t is defined as follows:

```
typedef struct __attribute__ ((__packed__))
{
  ct_mode_t    mode;
  ct_gate_t    gate;
  uint8_t      ext_gate_din;
  uint8_t      ext_clk;
  uint8_t      ext_clk_din;
  uint8_t      unused;
  union
  {
     struct __attribute__ ((__packed__))
     {
       uint32_t     period;
       uint32_t     pulse;
       uint8_t      out;
       uint8_t      out_hi;
     }divider;

     struct __attribute__ ((__packed__))
     {
       uint32_t     period;
       uint32_t     pulse;
       uint8_t      out;
       uint8_t      out_hi;
       uint8_t      retriggerable;
     }one_shot;
  };
}dt78xx_ct_config_t;
```

Structure Element Name: mode

Data Type: ct_mode_t

Description: This enumeration defines the operation mode of the counter/timer channel. It is defined as follows:

```
typedef enum
{
  ct_mode_idle,
  ct_mode_1shot,
  ct_mode_divider,
  ct_mode_counter,
  ct_mode_measure,
}ct_mode_t;
```

where,

- *ct_mode_idle* specifies idle mode. If you specify this mode, the counter no longer drives the clock output signal that is assigned to one of the general-purpose output signals (pins 11 to 18) of the I/O header.
- *ct_mode_1shot* specifies one-shot mode. If you specify this mode, values for the *one_shot* structure specified in *dt78xx_ct_config_t* are required. Specify one-shot mode if you want to generate non-retriggerable one-shot pulses.
- *ct_mode_divider* specifies rate generation mode. If you specify this mode, values for the *divider* structure specified in *dt78xx_ct_config_t* are required. In rate generation mode, a continuous pulse output signal is generated from the counter's output signal. You can use this pulse output signal as an external clock to pace other operations, such as analog input, analog output, or other counter/timer operations. The pulse output operation starts whenever the counter's gate signal is at the specified level.
- *ct_mode_counter* specifies event counting mode. In event counting mode, the number of rising edges that occur on the counter's clock input signal are counted when the counter's gate signal is active. The module can count a maximum of 4,294,967,296 events before the counter rolls over to 0 and starts counting again.
- *ct_mode_measure* is currently not supported.

Structure Element Name: gate

Data Type: ct_gate_t enumeration

Description: Defines the gate type to use for the specified counter/timer operation mode. It is defined as follows:

```
typedef enum
{
  ct_gate_none = 0x1,
  ct_gate_ext_hi = 0x2,
  ct_gate_ext_lo = 0x3
}ct_gate_t;
```

where,

- *ct_gate_none* specifies a software gate type. With this gate type, the **ioctl - IOCTL_START_SUBSYS** command, described on page 129, starts the counter/timer operation immediately after execution. (No general-purpose input signal is required if this gate type is selected.) To stop the operation, use the **ioctl - IOCTL_START_SUBSYS** command, described on page 139.
- *ct_gate_ext_hi* specifies a logic high or rising edge gate type. For event counting (*counter*) and rate generation (*divider*) mode, the operation is enabled when the counter's gate signal is high and is disabled when the counter's gate signal is low. For one-shot and repetitive one-shot (*1shot*) mode, the operation is enabled when the counter's gate signal goes from a low to a high transition and is disabled when the counter's gate signal goes from a high to a low transition.
- *ct_gate_ext_lo* specifies a logic low or falling edge gate type. For event counting (*counter*) and rate generation (*divider*) mode, the operation is enabled when the counter's gate signal is low and is disabled when the counter's gate signal is high. For one-shot and repetitive one-shot (*1shot*) mode, the operation is enabled when the counter's gate signal goes from a high to a low transition and is disabled when the counter's gate signal goes from a low to a high transition.

Structure Element Name: ext_gate_din

Data Type: uint8_t

Description: Specifies which general-purpose input signal (pins 1 to 8) on the I/O header to use for the external C/T gate input signal.

Pin 1 corresponds to bit 0 of the digital input port (value 0x1).

Pin 2 corresponds to bit 1 of the digital input port (value 0x2).

Pin 3 corresponds to bit 2 of the digital input port (value 0x4).

Pin 4 corresponds to bit 3 of the digital input port (value 0x8).

Pin 5 corresponds to bit 4 of the digital input port (value 0x10).

Pin 6 corresponds to bit 5 of the digital input port (value 0x20).

Pin 7 corresponds to bit 6 of the digital input port (value 0x40).

Pin 8 corresponds to bit 7 of the digital input port (value 0x80).

Structure Element Name: ext_clk

Data Type: uint8_t

Description: Specifies whether to use the internal C/T clock or an external C/T clock. When this value is 0, the internal C/T clock is used. When this value is 1, an external C/T clock is used.

Note that in one-shot and repetitive one-shot modes (*1shot)*, the internal C/T clock is more useful than an external C/T clock. In event counting (*counter*) mode, an external C/T clock source is generally used.

Either clock source can be used for rate generation mode (*divider*) depending on the application.

Structure Element Name: ext_clk_din

Data Type: uint8_t

Description: Specifies which general-purpose input signal (pins 1 to 8) on the I/O header to use for the external C/T clock input signal.

Pin 1 corresponds to bit 0 of the digital input port (value 0x1).

Pin 2 corresponds to bit 1 of the digital input port (value 0x2).

Pin 3 corresponds to bit 2 of the digital input port (value 0x4).

Pin 4 corresponds to bit 3 of the digital input port (value 0x8).

Pin 5 corresponds to bit 4 of the digital input port (value 0x10).

Pin 6 corresponds to bit 5 of the digital input port (value 0x20).

Pin 7 corresponds to bit 6 of the digital input port (value 0x40).

Pin 8 corresponds to bit 7 of the digital input port (value 0x80).

Structure Element Name: unused

Data Type: uint8_t

Description: This variable is reserved for future use.

Structure Element Name: period

Data Type: uint32_t

Description: Used by the *divider* and *one_shot* structures, specifies the number of input clock cycles used to create one period of the C/T output signal.

The period of the output pulse is determined by the C/T clock source (*ext_clk* element of the *dt78xx_ct_config_t* structure, described on page 89.) You can output pulses using a maximum frequency of 24 MHz (if using the internal C/T clock) or 5 MHz (if using the external C/T clock). Note, however, that the integrity of the signal degrades at frequencies greater than 10 MHz.

For example, if you are using an external C/T clock of10000 Hz as the input clock source of the counter/timer, and you want to generate a output signal of 1000 Hz with a 20% duty cycle, specify a period of 10 (10000 Hz divided by 10 is 1000 Hz) and a pulse width of 2 (the period multiplied by 20%). This is illustrated in Figure 7.

Note that the polarity of the output pulse can be programmed using the *out_hi* element of the *divider* or *one_shot* structure.

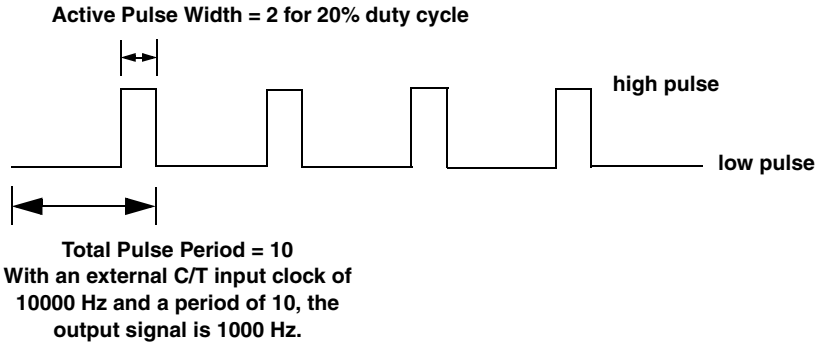


Figure 7: Example of a Pulse Width and Period

Structure Element Name: pulse

Data Type: uint32_t

Description: Used by the *divider* and *one_shot* structures, specifies the number of input clock cycles used to create the active pulse width (or duty cycle) of the C/T output signal. This value must be less than the *period*. See Figure 6 for an example.

Structure Element Name: out

Data Type: uint8_t

Description: Used by the *divider* and *one_shot* structures, specifies which general-purpose output signal (pins 11 to 18) on the I/O header to use for the external C/T clock output signal.

Pin 11 corresponds to bit 0 of the digital output port (value 0x1).

Pin 12 corresponds to bit 1 of the digital output port (value 0x2).

Pin 13 corresponds to bit 2 of the digital output port (value 0x4).

Pin 14 corresponds to bit 3 of the digital output port (value 0x8).

Pin 15 corresponds to bit 4 of the digital output port (value 0x10).

Pin 16 corresponds to bit 5 of the digital output port (value 0x20).

Pin 17 corresponds to bit 6 of the digital output port (value 0x40).

Pin 18 corresponds to bit 7 of the digital output port (value 0x80).

Structure Element Name: out_hi

Data Type: uint8_t

Description: Used by the *divider* and *one_shot* structures, specifies whether the output pulse for a rate generation or non-retriggerable one-shot operation is active high (1) or active low (0).

Structure Element Name: retriggerable

Data Type: uint8_t

Description: Used by the *one-shot* structure, specifies whether or not to repeat the one-shot output pulse.

Currently, only a value of 0 is supported.

When *retriggerable* is 0, the counter generates a single pulse output signal whenever it detects the active gate signal (after the pulse period from the previous output pulse expires). Any gate signals that occur while the pulse is being output are not detected by the module. The module continues to output pulses until you stop the operation with **ioctl - IOCTL_STOP_SUBSYS**, described on page 139. You can use this mode to clean up a poor clock input signal by changing its pulse width, and then outputting it.

Returns 0 = Success; < 0 = Failure

Notes The value of the counter clock output signal (assigned to one of the general-purpose output signals) can be overwritten by writing to the digital output subsystem. Refer to page 53 for more information.

If you assigned a general-purpose input signal as a counter clock or gate input (or external trigger), you can read the value of the signal as you would any other digital input signal. Refer to page 52 for more information on reading digital input values.

Example The following example sets up the counter/timer channel to generate a rate on the C/T clock output signal. In this example, the period of the output signal is 2 clock input signals of the internal 48 MHz C/T clock to generate a 24 MHz output signal and the pulse width of 1 clock input signal to generate a duty cycle of 50%.The operation is enabled when a logic high external gate signal is detected on general-purpose input signal 7. General-purpose output signal 7 is used for the C/T clock output signal. The output signal is active high.

```
#define CT_DEV_FILE     "//dev/DT7816-ctr-tmr"
ct_file = open(CT_DEV_FILE, O_RDWR);
dt78xx_ct_config_t ct_cfg;

ct_cfg.mode = ct_mode_divider;
ct_cfg.gate = ct_gate_ext_hi
ct_cfg.ext_gate_din = 0x80;
ct_cfg.ext_clk = 0;
ct_cfg.divider.period = 2;
ct_cfg.divider.pulse = 1;
ct_cfg.divider.out = 0x80;
ct_cfg.divider.out_high = 1;

if (ioctl(ct_file, IOCTL_CT_CFG_SET, &ct_cfg))
    {
        perror("IOCTL_CT_CFG_SET");
        goto finish;
    }
```

See Also **IOCTL_CT_CFG_GET**, described on page 79.

ioctl - IOCTL_GAIN_POT_GET

Description Returns the value of a specific potentiometer used for calibrating the gain.

Syntax

```
int ioctl(int fd, IOCTL_GAIN_POT_GET,
   struct dt78xx_cal_pot_t *pGainCalPot);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the analog input subsystem (/dev/DT7816-ain) or analog output subsystem (/dev/DT7816-aout).

Name: pGainCalPot

Data Type: dt78xx_cal_pot_t structure

Description: A pointer to a structure that defines the configuration of the calibration potentiometer.

dt78xx_cal_pot_t is defined as follows:

```
typedef struct __attribute__ ((__packed__))
{
  int          pot;
  uint32_t     value;
  struct
  {
     int       factory;
     uint32_t  reg;
  }cal;
}dt78xx_cal_pot_t;
```

Structure Element Name: pot

Data Type: int

Description: The potentiometer to calibrate. For the analog input subsystem on the DT7816, a single potentiometer (0) applies to all analog input channels. For the analog output subsystem, values are 0 for analog output channel 0 and 1 for analog output channel 1.

Structure Element Name: value

Data Type: uint32_t

Description: The value of the potentiometer. Values range from 0 to 255.

Structure Element Name: factory

Data Type: int

Description: Identifies the calibration type, where 1 is factory calibration and 0 is user calibration.

Structure Element Name: reg

Data Type: uint32_t

Description: Identifies the calibration register that stores the calibration factors for the potentiometer. Values range from 0 to 15.

For analog input and analog output calibration on the DT7816, only register 0 is used.

Returns 0 = Success; < 0 = Failure

Example The following example returns the user calibration value for the gain potentiometer that is associated with all the analog input channels on the DT7816.

```
#define AIN_DEV_FILE     "//dev/DT7816-ain"
ain_file = open(AIN_DEV_FILE, O_RDWR);
dt78xx_cal_pot_t gain_pot;

gain_pot.pot = 0;
gain_pot.value = 0;
gain_pot.factory = 0;
gain_pot.reg = 0;
if (ain_file, IOCTL_GAIN_POT_GET, &gain_pot)
  {
     fprintf(stdout, "*** ERROR reading calibration
        pot eeprom from driver\n");
     return false;
   }

*value = gain_pot.value;
return true;
```

See Also **IOCTL_GAIN_POT_SET**, described on page 95.

ioctl - IOCTL_GAIN_POT_SET

Description Sets the value of a specific potentiometer used to calibrate the gain.

Syntax

```
int ioctl(int fd, IOCTL_GAIN_POT_SET,
   struct dt78xx_cal_pot_t *pGainCalPot);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the analog input subsystem (/dev/DT7816-ain) or analog output subsystem (/dev/DT7816-aout).

Name: pGainCalPot

Data Type: dt78xx_cal_pot_t structure

Description: A pointer to a structure that defines the configuration of the calibration potentiometer.

dt78xx_cal_pot_t is defined as follows:

```
typedef struct __attribute__ ((__packed__))
{
  int          pot;
  uint32_t     value;
  struct
  {
     int       factory;
     uint32_t  reg;
  }cal;
}dt78xx_cal_pot_t;
```

Structure Element Name: pot

Data Type: int

Description: The potentiometer to calibrate. For the analog input subsystem on the DT7816, a single potentiometer (0) applies to all analog input channels. For the analog output subsystem, values are 0 for analog output channel 0 and 1 for analog output channel 1.

Structure Element Name: value

Data Type: uint32_t

Description: The value of the potentiometer. Values range from 0 to 255.

Structure Element Name: factory

Data Type: int

Description: Identifies the calibration type, where 1 is factory calibration and 0 is user calibration. Typically, you want to modify the user calibration value; therefore, a value of 0 is recommended.

Structure Element Name: reg

Data Type: uint32_t

Description: Identifies the calibration register. Values range from 0 to 15.

For analog input and analog output calibration on the DT7816, only register 0 is used.

Returns 0 = Success; < 0 = Failure

Notes You can continue to adjust the gain calibration values by calling this command with different values. The gain calibration register is automatically written to the wiper value of the associated potentiometer.

This command will block all other operations for at least 5 ms.

Example The following example sets the user calibration value for the gain potentiometer that is associated with all the analog input channels on the DT7816. In this example, the value is set to 255.

```
#define AIN_DEV_FILE     "//dev/DT7816-ain"
ain_file = open(AIN_DEV_FILE, O_RDWR);
dt78xx_cal_pot_t gain_pot;
gain_pot.pot = 0;
gain_pot.value = 255;
gain_pot.factory = 0;
gain_pot.reg =0;
if (ioctl(ain_file, IOCTL_GAIN_POT_SET, &gain_pot))
    {
      fprintf(stdout, "*** ERROR writing calibration
         pot eeprom to driver\n");
        return false;
    }
```

See Also **IOCTL_GAIN_POT_GET**, described on page 93.

ioctl - IOCTL_GAIN_POT_WIPER_GET

Description Returns the value of the wiper for a channel in a specific gain potentiometer.

Syntax

```
int ioctl(int fd, IOCTL_GAIN_POT_WIPER_GET,
   struct dt78xx_cal_pot_t *pGainCalWiperPot);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the analog input subsystem (/dev/DT7816-ain) or analog output subsystem (/dev/DT7816-aout).

Name: pGainCalWiperPot

Data Type: dt78xx_cal_pot_t structure

Description: A pointer to a structure that defines the configuration of the calibration potentiometers.

dt78xx_cal_pot_t is defined as follows:

```
typedef struct __attribute__ ((__packed__))
{
  int          pot;
  uint32_t     value;
  struct
  {
     int       factory;
     uint32_t  reg;
  }cal;
}dt78xx_cal_pot_t;
```

Structure Element Name: pot

Data Type: int

Description: The potentiometer to calibrate. For the analog input subsystem on the DT7816, a single potentiometer (0) applies to all analog input channels. For the analog output subsystem, values are 0 for analog output channel 0 and 1 for analog output channel 1.

Structure Element Name: value

Data Type: uint32_t

Description: The value of the potentiometer. Values range from 0 to 255.

Structure Element Name: factory

Data Type: int

Description: Not used for this command.

Structure Element Name: reg

Data Type: uint32_t

Description: Not used for this command.

Returns 0 = Success; < 0 = Failure

Example The following example returns the wiper value of the gain potentiometer that is associated with all the analog input channels of the DT7816:

```
#define AIN_DEV_FILE     "//dev/DT7816-ain"
ain_file = open(AIN_DEV_FILE, O_RDWR);
dt78xx_cal_pot_t gain_wiper_pot;

gain_wiper_pot.pot = 0;
gain_wiper_pot.value = 0;
if (ain_file, IOCTL_GAIN_POT_WIPER_GET,
   &gain_wiper_pot)
  {
     fprintf(stdout, "*** ERROR reading calibration
        pot eeprom from driver\n");
     return false;
   }

*value = gain_wiper_pot.value;
return true;
```

See Also **IOCTL_GAIN_POT_WIPER_SET**, described on page 99.

ioctl - IOCTL_GAIN_POT_WIPER_SET

Description Sets the value of the wiper for a channel in a specific gain potentiometer.

Syntax

```
int ioctl(int fd, IOCTL_GAIN_POT_WIPER_SET,
   struct dt78xx_cal_pot_t *pGainCalWiperPot);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the analog input subsystem (/dev/DT7816-ain) or analog output subsystem (/dev/DT7816-aout).

Name: pGainCalWiperPot

Data Type: dt78xx_cal_pot_t structure

Description: A pointer to a structure that defines the configuration of the calibration potentiometer.

dt78xx_cal_pot_t is defined as follows:

```
typedef struct __attribute__ ((__packed__))
{
  int          pot;
  uint32_t     value;
  struct
  {
     int       factory;
     uint32_t  reg;
  }cal;
}dt78xx_cal_pot_t;
```

Structure Element Name: pot

Data Type: int

Description: The potentiometer to calibrate. For the analog input subsystem on the DT7816, a single potentiometer (0) applies to all analog input channels. For the analog output subsystem, values are 0 for analog output channel 0 and 1 for analog output channel 1.

Structure Element Name: value

Data Type: uint32_t

Description: The value of the potentiometer. Values range from 0 to 255.

Structure Element Name: factory

Data Type: int

Description: Not used for this command.

Structure Element Name: reg

Data Type: uint32_t

Description: Not used for this command.

Returns 0 = Success; < 0 = Failure

Notes When you are satisfied that the gain calibration value is correct, call this command to update the calibration register in non-volatile EEPROM.

Example The following example sets the wiper value for the gain potentiometer that is associated with all the analog input channels on the DT7816 to a value of 255.

```
#define AIN_DEV_FILE     "//dev/DT7816-ain"
ain_file = open(AIN_DEV_FILE, O_RDWR);
dt78xx_cal_pot_t gain_wiper_pot;

gain_wiper_pot.pot = 0;
gain_wiper_pot.value = 255;
if (ioctl(ain_file, IOCTL_GAIN_POT_WIPER_SET,
   &gain_wiper_pot))
    {
        perror("IOCTL_GAIN_POT_WIPER_SET");
        goto finish;
    }
```

See Also **IOCTL_GAIN_POT_WIPER_GET**, described on page 97.

ioctl - IOCTL_LED_GET

Description Returns the configuration of the user LEDs on the DT7816 module.

Syntax

```
int ioctl(int fd, IOCTL_LED_GET,
   struct dt78xx_led_t *pLED_config);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with any of the subsystems on the DT7816 (/dev/DT7816-ain, /dev/DT7816-aout, /dev/DT7816-din, /dev/DT7816-dout, /dev/DT7816-tach, /dev/DT7816-measure, /dev/DT7816-ctr-tmr, /dev/DT7816-stream-in, or /dev/DT7816-stream-out).

Name: pLED_config

Data Type: dt78xx_led_t structure

Description: A pointer to a structure that defines the configuration of the user LEDs on the DT7816 module.

dt78xx_led_t is defined as follows:

```
typedef struct __attribute__ ((__packed__))
{
  uint8_t   mask;
  uint8_t   state;
} dt78xx_led_t;
```

Structure Element Name: mask

Data Type: unit_8_t

Description: An 8-bit unsigned value used by the **ioctl - IOCTL_LED_SET** command, described on page 103, to specify which user LEDs to modify. It is not used by the **ioctl - IOCTL_LED_GET** command.

Structure Element Name: state

Data Type: uint8_t

Description: An 8-bit unsigned value indicating the status of the eight user LEDs on the DT7816 module.

Each LED corresponds to a bit of the 8-bit value, as follows:

- LED 7 corresponds to bit 7 (value 0x80).
- LED 6 corresponds to bit 6 (value 0x40).
- LED 5 corresponds to bit 5 (value 0x20).
- LED 4 corresponds to bit 4 (value 0x10).
- LED 3 corresponds to bit 3 (value 0x8).
- LED 2 corresponds to bit 2 (value 0x4).
- LED 1 corresponds to bit 1 (value 0x2).
- LED 0 corresponds to bit 0 (value 0x1).

A value of 0 indicates that the LED is off, and a value of 1 indicates that the LED is on.

Returns 0 = Success; < 0 = Failure

Notes Refer to Figure 5 on page 58 for the location of the user LEDs on the DT7816 module.

Example The following example returns the configuration of the user LEDs on the DT7816 module:

```
#define AIN_DEV_FILE "//dev/DT7816-ain"
ain_file = open(AIN_DEV_FILE, O_RDWR);
dt78xx_led_t LED_cfg;

if (ioctl(ain_file, IOCTL_LED_GET, &LED_cfg))
    {
        perror("IOCTL_LED_GET");
        goto finish;
    }
```

See Also **IOCTL_LED_SET**, described on page 103.

ioctl - IOCTL_LED_SET

Description Specifies the configuration of the user LEDs on the DT7816 module.

Syntax

```
int ioctl(int fd, IOCTL_LED_SET,
   struct dt78xx_led_t *pLED_config);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with any of the subsystems on the DT7816 (/dev/DT7816-ain, /dev/DT7816-aout, /dev/DT7816-din, /dev/DT7816-dout, /dev/DT7816-tach, /dev/DT7816-measure, /dev/DT7816-ctr-tmr, /dev/DT7816-stream-in, or /dev/DT7816-stream-out).

Name: pLED_config

Data Type: dt78xx_led_t structure

Description: A pointer to a structure that defines the configuration of the user LEDs on the DT7816 module.

dt78xx_led_t is defined as follows:

```
typedef struct __attribute__ ((__packed__))
{
  uint8_t   mask;
  uint8_t   state;
} dt78xx_led_t;
```

Structure Element Name: mask

Data Type: unit_8_t

Description: An 8-bit unsigned value that specifies which of the eight user LEDs on the DT7816 module to modify. Each LED corresponds to a bit of the 8-bit value, as follows:

- LED 7 corresponds to bit 7 (value 0x80).
- LED 6 corresponds to bit 6 (value 0x40).
- LED 5 corresponds to bit 5 (value 0x20).
- LED 4 corresponds to bit 4 (value 0x10).
- LED 3 corresponds to bit 3 (value 0x8).
- LED 2 corresponds to bit 2 (value 0x4).
- LED 1 corresponds to bit 1 (value 0x2).
- LED 0 corresponds to bit 0 (value 0x1).

Specify a value of 0 for the bit to keep the state of the LED unchanged. Specify a value of 1 if you want to modify the state of the LED.

Structure Element Name: state

Data Type: uint8_t

Description: An 8-bit unsigned value indicating the state of the eight user LEDs on the DT7816 module. Each LED corresponds to a bit of the 8-bit value, as follows:

- LED 7 corresponds to bit 7 (value 0x80).
- LED 6 corresponds to bit 6 (value 0x40).
- LED 5 corresponds to bit 5 (value 0x20).
- LED 4 corresponds to bit 4 (value 0x10).
- LED 3 corresponds to bit 3 (value 0x8).
- LED 2 corresponds to bit 2 (value 0x4).
- LED 1 corresponds to bit 1 (value 0x2).
- LED 0 corresponds to bit 0 (value 0x1).

Specify a value of 0 if you want to turn the LED off. Specify a value of 1 if you want to turn the LED on.

Returns 0 = Success; < 0 = Failure

Notes Refer to Figure 5 on page 58 for the location of the user LEDs on the DT7816 module.

Example The following example modifies the state of user LEDs 0 and 2 on the DT7816 module (for a value of 0x5). LED 0 is turned off and LED 2 is turned on (for a value of 0x4):

```
#define AIN_DEV_FILE "//dev/DT7816-ain"
ain_file = open(AIN_DEV_FILE, O_RDWR);
dt78xx_led_t LED_cfg;

led_cfg.mask = 0x5;
led_cfg.state = 0x4;

if (ioctl(ain_file, IOCTL_LED_SET, &LED_cfg))
    {
      perror("IOCTL_LED_SET");
      goto finish;
    }
```

See Also **IOCTL_LED_GET**, described on page 101.

ioctl - IOCTL_MCTR_CFG_GET

Description Returns the configuration of the measure counter.

Syntax

```
int ioctl(int fd, IOCTL_MCTR_CFG_GET,
   struct dt78xx_mctr_config_t *pMCTR_config);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the measure counter subsystem (/dev/DT7816-measure).

Name: pMCTR_config

Data Type: dt78xx_mctr_config_t structure

Description: A pointer to a structure that defines the configuration of the measure counter.

dt78xx_mctr_config_t is defined as follows:

```
typedef struct __attribute__ ((__packed__))
{
  mctr_sel_t    start_sel;
  mctr_sel_t    stop_sel;
  uint8_t       stale_flag;
  };
}dt78xx_mctr_config_t;
```

Structure Element Name: start_sel

Data Type: mctr_sel_t

Description: This enumeration defines the signal and edge that starts the measure counter. It is defined as follows:

```
typedef enum
{
  mctr_sel_adc_complete = 0x00,
  mctr_sel_tach_falling = 0x10,
  mctr_sel_tach_rising  = 0x11,
  mctr_sel_din0_falling = 0x20,
  mctr_sel_din0_rising  = 0x21,
  mctr_sel_din1_falling = 0x22,
  mctr_sel_din1_rising  = 0x23,
  mctr_sel_din2_falling = 0x24,
  mctr_sel_din2_rising  = 0x25,
  mctr_sel_din3_falling = 0x26,
  mctr_sel_din3_rising  = 0x27,
  mctr_sel_din4_falling = 0x28,
  mctr_sel_din4_rising  = 0x29,
  mctr_sel_din5_falling = 0x2a,
  mctr_sel_din5_rising  = 0x2b,
  mctr_sel_din6_falling = 0x2c,
  mctr_sel_din6_rising  = 0x2d,
  mctr_sel_din7_falling = 0x2e,
  mctr_sel_din7_rising  = 0x2f,
} mctr_sel_t;
```

where,

- *mctr_sel_adc_complete* (value 0x00) – The measure counter starts when the A/D conversion is complete.
- *mctr_sel_tach_falling* (value 0x10) – The measure counter starts on the falling edge of the tachometer input signal (pin 23 of the I/O header).
- *mctr_sel_tach_rising* (value 0x11) – The measure counter starts on the rising edge of the tachometer input signal (pin 23 of the I/O header).
- *mctr_sel_din0_falling* (value 0x20) – The measure counter starts on the falling edge of general-purpose input signal 0 (pin 1 of the I/O header).
- *mctr_sel_din0_rising* (value 0x21) – The measure counter starts on the rising edge of general-purpose input signal 0 (pin 1 of the I/O header).
- *mctr_sel_din1_falling* (value 0x22) – The measure counter starts on the falling edge of general-purpose input signal 1 (pin 2 of the I/O header).

Description (cont):

- *mctr_sel_din1_rising* (value 0x23) – The measure counter starts on the rising edge of general-purpose input signal 1 (pin 2 of the I/O header).
- *mctr_sel_din2_falling* (value 0x24) – The measure counter starts on the falling edge of general-purpose input signal 2 (pin 3 of the I/O header).
- *mctr_sel_din2_rising* (value 0x25) – The measure counter starts on the rising edge of general-purpose input signal 2 (pin 3 of the I/O header).
- *mctr_sel_din3_falling* (value 0x26) – The measure counter starts on the falling edge of general-purpose input signal 3 (pin 4 of the I/O header).
- *mctr_sel_din3_rising* (value 0x27) – The measure counter starts on the rising edge of general-purpose input signal 3 (pin 4 of the I/O header).
- *mctr_sel_din4_falling* (value 0x28) – The measure counter starts on the falling edge of general-purpose input signal 4 (pin 5 of the I/O header).
- *mctr_sel_din4_rising* (value 0x29) – The measure counter starts on the rising edge of general-purpose input signal 4 (pin 5 of the I/O header).
- *mctr_sel_din5_falling* (value 0x2a) – The measure counter starts on the falling edge of general-purpose input signal 5 (pin 6 of the I/O header).
- *mctr_sel_din5_rising* (value 0x2b) – The measure counter starts on the rising edge of general-purpose input signal 5 (pin 6 of the I/O header).
- *mctr_sel_din6_falling* (value 0x2c) – The measure counter starts on the falling edge of general-purpose input signal 6 (pin 7 of the I/O header).
- *mctr_sel_din6_rising* (value 0x2d) – The measure counter starts on the rising edge of general-purpose input signal 6 (pin 7 of the I/O header).
- *mctr_sel_din7_falling* (value 0x2e) – The measure counter starts on the falling edge of general-purpose input signal 7 (pin 8 of the I/O header).
- *mctr_sel_din7_rising* (value 0x2f) – The measure counter starts on the rising edge of general-purpose input signal 7 (pin 8 of the I/O header).

Structure Element Name: stop_sel

Data Type: mctr_sel_t

Description: This enumeration defines the signal and edge that stops the measure counter. It is defined as follows:

```
typedef enum
{
  mctr_sel_adc_complete = 0x00,
  mctr_sel_tach_falling = 0x10,
  mctr_sel_tach_rising  = 0x11,
  mctr_sel_din0_falling = 0x20,
  mctr_sel_din0_rising  = 0x21,
  mctr_sel_din1_falling = 0x22,
  mctr_sel_din1_rising  = 0x23,
  mctr_sel_din2_falling = 0x24,
  mctr_sel_din2_rising  = 0x25,
  mctr_sel_din3_falling = 0x26,
  mctr_sel_din3_rising  = 0x27,
  mctr_sel_din4_falling = 0x28,
  mctr_sel_din4_rising  = 0x29,
  mctr_sel_din5_falling = 0x2a,
  mctr_sel_din5_rising  = 0x2b,
  mctr_sel_din6_falling = 0x2c,
  mctr_sel_din6_rising  = 0x2d,
  mctr_sel_din7_falling = 0x2e,
  mctr_sel_din7_rising  = 0x2f,
} mctr_sel_t;
```

where,

- *mctr_sel_adc_complete* (value 0x00) – The measure counter stops when the A/D conversion is complete.
- *mctr_sel_tach_falling* (value 0x10) – The measure counter stops on the falling edge of the tachometer input signal (pin 23 of the I/O header).
- *mctr_sel_tach_rising* (value 0x11) – The measure counter stops on the rising edge of the tachometer input signal (pin 23 of the I/O header).
- *mctr_sel_din0_falling* (value 0x20) – The measure counter stops on the falling edge of general-purpose input signal 0 (pin 1 of the I/O header).
- *mctr_sel_din0_rising* (value 0x21) – The measure counter stops on the rising edge of general-purpose input signal 0 (pin 1 of the I/O header).
- *mctr_sel_din1_falling* (value 0x22) – The measure counter stops on the falling edge of general-purpose input signal 1 (pin 2 of the I/O header).

Description (cont.):

- *mctr_sel_din1_rising* (value 0x23) – The measure counter stops on the rising edge of general-purpose input signal 1 (pin 2 of the I/O header).
- *mctr_sel_din2_falling* (value 0x24) – The measure counter stops on the falling edge of general-purpose input signal 2 (pin 3 of the I/O header).
- *mctr_sel_din2_rising* (value 0x25) – The measure counter stops on the rising edge of general-purpose input signal 2 (pin 3 of the I/O header).
- *mctr_sel_din3_falling* (value 0x26) – The measure counter stops on the falling edge of general-purpose input signal 3 (pin 4 of the I/O header).
- *mctr_sel_din3_rising* (value 0x27) – The measure counter stops on the rising edge of general-purpose input signal 3 (pin 4 of the I/O header).
- *mctr_sel_din4_falling* (value 0x28) – The measure counter stops on the falling edge of general-purpose input signal 4 (pin 5 of the I/O header).
- *mctr_sel_din4_rising* (value 0x29) – The measure counter stops on the rising edge of general-purpose input signal 4 (pin 5 of the I/O header).
- *mctr_sel_din5_falling* (value 0x2a) – The measure counter stops on the falling edge of general-purpose input signal 5 (pin 6 of the I/O header).
- *mctr_sel_din5_rising* (value 0x2b) – The measure counter stops on the rising edge of general-purpose input signal 5 (pin 6 of the I/O header).
- *mctr_sel_din6_falling* (value 0x2c) – The measure counter stops on the falling edge of general-purpose input signal 6 (pin 7 of the I/O header).
- *mctr_sel_din6_rising* (value 0x2d) – The measure counter stops on the rising edge of general-purpose input signal 6 (pin 7 of the I/O header).
- *mctr_sel_din7_falling* (value 0x2e) – The measure counter stops on the falling edge of general-purpose input signal 7 (pin 8 of the I/O header).
- *mctr_sel_din7_rising* (value 0x2f) – The measure counter stops on the rising edge of general-purpose input signal 7 (pin 8 of the I/O header).

Structure Element Name: stale_flag

Data Type: uint8_t

Description: Indicates whether or not the data is new. This flag is used only when the start edge and the stop edge is set to use the same pin and edge (such as pin 0 - DIN 0 rising as the start edge and pin 0 -DIN rising as the stop edge).

If the *stale_flag* is 1 (Used), the most significant bit (MSB) of the value is set to 0 to indicate new data; reading the value before the measurement is complete returns an MSB of 1.

If *stale_flag* is 0 (Not Used), the MSB is always set to 0.

Returns 0 = Success; < 0 = Failure

Notes By default, the general-purpose input signals are configured as digital input signals.

You read the value of the measure counter through the input stream by setting bit 10 of the channel mask using the **ioctl - IOCTL_CHAN_MASK_SET** command, described on page 76.

Example The following example returns the configuration of the measure counter to start on the rising edge of the tachometer and stop on the falling edge of general-purpose input signal 5. In this example, the general-purpose input signal is configured as a digital input line. The stale flag is not used.

```
#define MCTR_DEV_FILE     "//dev/DT7816-measure"
mctr_file = open(MCTR_DEV_FILE, O_RDWR);
dt78xx_mctr_config_t mctr_cfg;

if (ioctl(mctr_file, IOCTL_MCTR_CFG_GET, &mctr_cfg))
    {
        perror("IOCTL_MCTR_CFG_GET");
        goto finish;
    }
```

See Also **IOCTL_MCT_CFG_SET**, described on page 111.

ioctl - IOCTL_MCTR_CFG_SET

Description Specifies the configuration of the measure counter.

Syntax

```
int ioctl(int fd, IOCTL_MCTR_CFG_SET,
   struct dt78xx_mctr_config_t *pMCTR_config);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the measure counter subsystem (/dev/DT7816-measure).

Name: pMCTR_config

Data Type: dt78xx_mctr_config_t structure

Description: A pointer to a structure that defines the configuration of the measure counter.

dt78xx_mctr_config_t is defined as follows:

```
typedef struct __attribute__ ((__packed__))
{
  mctr_sel_t    start_sel;
  mctr_sel_t    stop_sel;
  uint8_t       stale_flag;
  };
}dt78xx_mctr_config_t;
```

Structure Element Name: start_sel

Data Type: mctr_sel_t

Description: This enumeration defines the signal and edge that starts the measure counter. It is defined as follows:

```
typedef enum
{
  mctr_sel_adc_complete = 0x00,
  mctr_sel_tach_falling = 0x10,
  mctr_sel_tach_rising  = 0x11,
  mctr_sel_din0_falling = 0x20,
  mctr_sel_din0_rising  = 0x21,
  mctr_sel_din1_falling = 0x22,
  mctr_sel_din1_rising  = 0x23,
  mctr_sel_din2_falling = 0x24,
  mctr_sel_din2_rising  = 0x25,
  mctr_sel_din3_falling = 0x26,
  mctr_sel_din3_rising  = 0x27,
  mctr_sel_din4_falling = 0x28,
  mctr_sel_din4_rising  = 0x29,
  mctr_sel_din5_falling = 0x2a,
  mctr_sel_din5_rising  = 0x2b,
  mctr_sel_din6_falling = 0x2c,
  mctr_sel_din6_rising  = 0x2d,
  mctr_sel_din7_falling = 0x2e,
  mctr_sel_din7_rising  = 0x2f,
} mctr_sel_t;
```

where,

- *mctr_sel_adc_complete* (value 0x00) – The measure counter starts when the A/D conversion is complete.
- *mctr_sel_tach_falling* (value 0x10) – The measure counter starts on the falling edge of the tachometer input signal (pin 23 of the I/O header).
- *mctr_sel_tach_rising* (value 0x11) – The measure counter starts on the rising edge of the tachometer input signal (pin 23 of the I/O header).
- *mctr_sel_din0_falling* (value 0x20) – The measure counter starts on the falling edge of general-purpose input signal 0 (pin 1 of the I/O header).
- *mctr_sel_din0_rising* (value 0x21) – The measure counter starts on the rising edge of general-purpose input signal 0 (pin 1 of the I/O header).
- *mctr_sel_din1_falling* (value 0x22) – The measure counter starts on the falling edge of general-purpose input signal 1 (pin 2 of the I/O header).

Description (cont):

- *mctr_sel_din1_rising* (value 0x23) – The measure counter starts on the rising edge of general-purpose input signal 1 (pin 2 of the I/O header).
- *mctr_sel_din2_falling* (value 0x24) – The measure counter starts on the falling edge of general-purpose input signal 2 (pin 3 of the I/O header).
- *mctr_sel_din2_rising* (value 0x25) – The measure counter starts on the rising edge of general-purpose input signal 2 (pin 3 of the I/O header).
- *mctr_sel_din3_falling* (value 0x26) – The measure counter starts on the falling edge of general-purpose input signal 3 (pin 4 of the I/O header).
- *mctr_sel_din3_rising* (value 0x27) – The measure counter starts on the rising edge of general-purpose input signal 3 (pin 4 of the I/O header).
- *mctr_sel_din4_falling* (value 0x28) – The measure counter starts on the falling edge of general-purpose input signal 4 (pin 5 of the I/O header).
- *mctr_sel_din4_rising* (value 0x29) – The measure counter starts on the rising edge of general-purpose input signal 4 (pin 5 of the I/O header).
- *mctr_sel_din5_falling* (value 0x2a) – The measure counter starts on the falling edge of general-purpose input signal 5 (pin 6 of the I/O header).
- *mctr_sel_din5_rising* (value 0x2b) – The measure counter starts on the rising edge of general-purpose input signal 5 (pin 6 of the I/O header).
- *mctr_sel_din6_falling* (value 0x2c) – The measure counter starts on the falling edge of general-purpose input signal 6 (pin 7 of the I/O header).
- *mctr_sel_din6_rising* (value 0x2d) – The measure counter starts on the rising edge of general-purpose input signal 6 (pin 7 of the I/O header).
- *mctr_sel_din7_falling* (value 0x2e) – The measure counter starts on the falling edge of general-purpose input signal 7 (pin 8 of the I/O header).
- *mctr_sel_din7_rising* (value 0x2f) – The measure counter starts on the rising edge of general-purpose input signal 7 (pin 8 of the I/O header).

Structure Element Name: stop_sel

Data Type: mctr_sel_t

Description: This enumeration defines the signal and edge that stops the measure counter. It is defined as follows:

```
typedef enum
{
  mctr_sel_adc_complete = 0x00,
  mctr_sel_tach_falling = 0x10,
  mctr_sel_tach_rising  = 0x11,
  mctr_sel_din0_falling = 0x20,
  mctr_sel_din0_rising  = 0x21,
  mctr_sel_din1_falling = 0x22,
  mctr_sel_din1_rising  = 0x23,
  mctr_sel_din2_falling = 0x24,
  mctr_sel_din2_rising  = 0x25,
  mctr_sel_din3_falling = 0x26,
  mctr_sel_din3_rising  = 0x27,
  mctr_sel_din4_falling = 0x28,
  mctr_sel_din4_rising  = 0x29,
  mctr_sel_din5_falling = 0x2a,
  mctr_sel_din5_rising  = 0x2b,
  mctr_sel_din6_falling = 0x2c,
  mctr_sel_din6_rising  = 0x2d,
  mctr_sel_din7_falling = 0x2e,
  mctr_sel_din7_rising  = 0x2f,
} mctr_sel_t;
```

where,

- *mctr_sel_adc_complete* (value 0x00) – The measure counter stops when the A/D conversion is complete.
- *mctr_sel_tach_falling* (value 0x10) – The measure counter stops on the falling edge of the tachometer input signal (pin 23 of the I/O header).
- *mctr_sel_tach_rising* (value 0x11) – The measure counter stops on the rising edge of the tachometer input signal (pin 23 of the I/O header).
- *mctr_sel_din0_falling* (value 0x20) – The measure counter stops on the falling edge of general-purpose input signal 0 (pin 1 of the I/O header).
- *mctr_sel_din0_rising* (value 0x21) – The measure counter stops on the rising edge of general-purpose input signal 0 (pin 1 of the I/O header).
- *mctr_sel_din1_falling* (value 0x22) – The measure counter stops on the falling edge of general-purpose input signal 1 (pin 2 of the I/O header).

Description (cont.):

- *mctr_sel_din1_rising* (value 0x23) – The measure counter stops on the rising edge of general-purpose input signal 1 (pin 2 of the I/O header).
- *mctr_sel_din2_falling* (value 0x24) – The measure counter stops on the falling edge of general-purpose input signal 2 (pin 3 of the I/O header).
- *mctr_sel_din2_rising* (value 0x25) – The measure counter stops on the rising edge of general-purpose input signal 2 (pin 3 of the I/O header).
- *mctr_sel_din3_falling* (value 0x26) – The measure counter stops on the falling edge of general-purpose input signal 3 (pin 4 of the I/O header).
- *mctr_sel_din3_rising* (value 0x27) – The measure counter stops on the rising edge of general-purpose input signal 3 (pin 4 of the I/O header).
- *mctr_sel_din4_falling* (value 0x28) – The measure counter stops on the falling edge of general-purpose input signal 4 (pin 5 of the I/O header).
- *mctr_sel_din4_rising* (value 0x29) – The measure counter stops on the rising edge of general-purpose input signal 4 (pin 5 of the I/O header).
- *mctr_sel_din5_falling* (value 0x2a) – The measure counter stops on the falling edge of general-purpose input signal 5 (pin 6 of the I/O header).
- *mctr_sel_din5_rising* (value 0x2b) – The measure counter stops on the rising edge of general-purpose input signal 5 (pin 6 of the I/O header).
- *mctr_sel_din6_falling* (value 0x2c) – The measure counter stops on the falling edge of general-purpose input signal 6 (pin 7 of the I/O header).
- *mctr_sel_din6_rising* (value 0x2d) – The measure counter stops on the rising edge of general-purpose input signal 6 (pin 7 of the I/O header).
- *mctr_sel_din7_falling* (value 0x2e) – The measure counter stops on the falling edge of general-purpose input signal 7 (pin 8 of the I/O header).
- *mctr_sel_din7_rising* (value 0x2f) – The measure counter stops on the rising edge of general-purpose input signal 7 (pin 8 of the I/O header).

Structure Element Name: stale_flag

Data Type: uint8_t

Description: Indicates whether or not the data is new. This flag is used only when the start edge and the stop edge is set to use the same pin and edge (such as pin 0 - DIN 0 rising as the start edge and pin 0 -DIN rising as the stop edge).

If the *stale_flag* is 1 (Used), the most significant bit (MSB) of the value is set to 0 to indicate new data; reading the value before the measurement is complete returns an MSB of 1.

If *stale_flag* is 0 (Not Used), the MSB is always set to 0.

Returns 0 = Success; < 0 = Failure

Notes By default, the general-purpose input signals are configured as digital input signals.

You read the value of the measure counter through the input stream by setting bit 10 of the channel mask using the **ioctl - IOCTL_CHAN_MASK_SET** command, described on page 76

Example The following example sets up the measure counter to start on the rising edge of the tachometer and stop on the falling edge of general-purpose input signal 5. In this example, the general-purpose input signal is configured as a digital input line. The stale flag is not used.

```
#define MCTR_DEV_FILE     "//dev/DT7816-measure"
mctr_file = open(MCTR_DEV_FILE, O_RDWR);
dt78xx_mctr_config_t mctr_cfg;

mctr_cfg.start_sel = mctr_sel_tach_rising;
mctr_cfg.stop_sel = mctr_sel_din5_falling;
mctr_cfg.stale_flag = o;

if (ioctl(mctr_file, IOCTL_MCTR_CFG_SET, &mctr_cfg))
    {
        perror("IOCTL_MCTR_CFG_SET");
        goto finish;
    }
```

See Also **IOCTL_MCT_CFG_GET**, described on page 105.

ioctl - IOCTL_OFFSET_POT_GET

Description Returns the value of a specific potentiometer used to calibrate the offset value associated with a specific channel.

Syntax

```
int ioctl(int fd, IOCTL_OFFSET_POT_GET,
   struct dt78xx_cal_pot_t *pOffetCalPot);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the analog output subsystem (/dev/DT7816-aout). The analog input subsystem of the DT7816 does not require offset calibration.

Name: pOffsetCalPot

Data Type: dt78xx_cal_pot_t structure

Description: A pointer to a structure that defines the configuration of the calibration potentiometer.

dt78xx_cal_pot_t is defined as follows:

```
typedef struct __attribute__ ((__packed__))
{
  int          pot;
  uint32_t     value;
  struct
  {
     int       factory;
     uint32_t  reg;
  }cal;
}dt78xx_cal_pot_t;
```

Structure Element Name: pot

Data Type: int

Description: The potentiometer to calibrate. For the analog output subsystem of the DT7816, values are 0 for analog output channel 0 and 1 for analog output channel 1.

Structure Element Name: value

Data Type: uint32_t

Description: The value of the potentiometer. Values range from 0 to 255.

Structure Element Name: factory

Data Type: int

Description: Identifies the calibration type, where 1 is factory calibration and 0 is user calibration.

Structure Element Name: reg

Data Type: uint32_t

Description: Identifies the calibration register. Values range from 0 to 15.

For analog output calibration on the DT7816, only register 0 is used.

Returns 0 = Success; < 0 = Failure

Example The following example returns the user calibration value of the offset calibration potentiometer that is associated with analog output channel 1 of the DT7816.

```
#define AIN_DEV_FILE     "//dev/DT7816-aout"
ain_file = open(AIN_DEV_FILE, O_RDWR);
dt78xx_cal_pot_t offset_pot;

offset_pot.pot = 1;
offset_pot.value = 0;
offset_pot.factory = 0;
offset_pot.reg = 0;
if (ain_file, IOCTL_OFFSET_POT_GET, &offset_pot)
  {
     fprintf(stdout, "*** ERROR reading calibration
        pot eeprom from driver\n");
     return false;
   }

*value = offset_pot.value;
return true;
```

See Also **IOCTL_OFFSET_POT_SET**, described on page 119.

ioctl - IOCTL_OFFSET_POT_SET

Description Sets the value of a specific potentiometer used to calibrate the offset value associated with a specific channel.

Syntax

```
int ioctl(int fd, IOCTL_OFFSET_POT_SET,
   struct dt78xx_cal_pot_t *pOffetCalPot);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the analog output subsystem (/dev/DT7816-aout). The analog input subsystem of the DT7816 does not require offset calibration.

Name: pOffsetCalPot

Data Type: dt78xx_cal_pot_t structure

Description: A pointer to a structure that defines the configuration of the calibration potentiometer.

dt78xx_cal_pot_t is defined as follows:

```
typedef struct __attribute__ ((__packed__))
{
  int          pot;
  uint32_t     value;
  struct
  {
     int       factory;
     uint32_t  reg;
  }cal;
}dt78xx_cal_pot_t;
```

Structure Element Name: pot

Data Type: int

Description: The potentiometer to calibrate. For the analog output subsystem of the DT7816, values are 0 for analog output channel 0 and 1 for analog output channel 1.

Structure Element Name: value

Data Type: uint32_t

Description: The value of the potentiometer. Values range from to 0 to 255.

Structure Element Name: factory

Data Type: int

Description: Identifies the calibration type, where 1 is factory calibration and 0 is user calibration.

Structure Element Name: reg

Data Type: uint32_t

Description: Identifies the calibration register. Values range from 0 to 15.

For analog output calibration on the DT7816, only register 0 is used.

Returns 0 = Success; < 0 = Failure

Notes You can continue to adjust the offset calibration values by calling this command with different values. The offset calibration register is automatically written to the wiper of the associated potentiometer.

Example The following example sets the user calibration value for the offset potentiometer that is associated with analog output channel 1 of the DT7816. In this example, the value is set to 254.

```
#define AIN_DEV_FILE     "//dev/DT7816-aout"
ain_file = open(AIN_DEV_FILE, O_RDWR);
dt78xx_cal_pot_t offset_pot;
offset_pot.pot = 1;
offset_pot.value = 254;
offset_pot.factory = 0;
offset_pot.reg = 0;

if (ioctl(ain_file, IOCTL_OFFSET_POT_SET, &offset_pot))
  {
     fprintf(stdout, "*** ERROR writing calibration
        pot eeprom \n");
     return false;
   }
```

See Also **IOCTL_OFFSET_POT_GET**, described on page 117.

ioctl - IOCTL_OFFSET_POT_WIPER_GET

Description Returns the value of the wiper for a channel in a specific offset potentiometer.

Syntax

```
int ioctl(int fd, IOCTL_OFFSET_POT_WIPER_GET,
   struct dt78xx_cal_pot_t *pOffsetCalWiperPot);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with analog output subsystem (/dev/DT7816-aout). The analog input subsystem of the DT7816 does not require offset calibration.

Name: pOffsetCalWiperPot

Data Type: dt78xx_cal_pot_t structure

Description: A pointer to a structure that defines the configuration of the calibration potentiometers.

dt78xx_cal_pot_t is defined as follows:

```
typedef struct __attribute__ ((__packed__))
{
  int          pot;
  uint32_t     value;
  struct
  {
     int       factory;
     uint32_t  reg;
  }cal;
}dt78xx_cal_pot_t;
```

Structure Element Name: pot

Data Type: int

Description: The potentiometer to calibrate. For the analog output subsystem of the DT7816, values are 0 for analog output channel 0 and 1 for analog output channel 1.

Structure Element Name: value

Data Type: uint32_t

Description: The value of the potentiometer. Values range from 0 to 255.

Structure Element Name: factory

Data Type: int

Description: Not used for this command.

Structure Element Name: reg

Data Type: uint32_t

Description: Not used for this command.

Returns 0 = Success; < 0 = Failure

Example The following example returns the wiper value of the offset potentiometer for analog output channel 1:

```
#define AIN_DEV_FILE     "//dev/DT7816-aout"
ain_file = open(AIN_DEV_FILE, O_RDWR);
dt78xx_cal_pot_t offset_wiper_pot;

offset_wiper_pot.pot = 1;
offset_wiper_pot.value = 0;
if (ain_file, IOCTL_OFFSET_POT_WIPER_GET,
   &offset_wiper_pot)
  {
     fprintf(stdout, "*** ERROR reading calibration
        pot eeprom from driver\n");
     return false;
   }

*value = offset_wiper_pot.value;
return true;
```

See Also **IOCTL_OFFSET_POT_WIPER_SET**, described on page 123.

ioctl - IOCTL_OFFSET_POT_WIPER_SET

Description Sets the value of the wiper for a channel in a specific offset potentiometer.

Syntax

```
int ioctl(int fd, IOCTL_OFFSET_POT_WIPER_SET,
   struct dt78xx_cal_pot_t *pOffsetCalWiperPot);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the analog output subsystem (/dev/DT7816-aout). The analog input subsystem of the DT7816 does not require offset calibration.

Name: pOffsetCalWiperPot

Data Type: dt78xx_cal_pot_t structure

Description: A pointer to a structure that defines the configuration of the calibration potentiometer.

dt78xx_cal_pot_t is defined as follows:

```
typedef struct __attribute__ ((__packed__))
{
  int          pot;
  uint32_t     value;
  struct
  {
     int       factory;
     uint32_t  reg;
  }cal;
}dt78xx_cal_pot_t;
```

Structure Element Name: pot

Data Type: int

Description: The potentiometer to calibrate. For the analog output subsystem of the DT7816, values are 0 for analog output channel 0 and 1 for analog output channel 1.

Structure Element Name: value

Data Type: uint32_t

Description: The value of the potentiometer. Values range from 0 to 255.

Structure Element Name: factory

Data Type: int

Description: Not used for this command.

Structure Element Name: reg

Data Type: uint32_t

Description: Not used for this command.

Returns 0 = Success; < 0 = Failure

Notes When you are satisfied that the offset calibration value is correct, call this command to update the calibration register in non-volatile EEPROM.

Example The following example sets the wiper for the offset potentiometer that is associated with analog output channel 1 to a value of 254:

```
#define AIN_DEV_FILE     "//dev/DT7816-aout"
ain_file = open(AIN_DEV_FILE, O_RDWR);
dt78xx_cal_pot_t offset_wiper_pot;

offset_wiper_pot.pot = 1;
offset_wiper_pot.value = 254;
if (ioctl(ain_file, IOCTL_OFFSET_POT_WIPER_SET,
   &offset_wiper_pot))
  {
     fprintf(stdout, "*** ERROR writing calibration
        pot eeprom \n");
     return false;
   }
```

See Also **IOCTL_OFFSET_POT_WIPER_GET**, described on page 121.

ioctl - IOCTL_SAMPLE_CLK_GET

Description Returns the configuration of the sample clock for the specified stream.

Syntax

```
int ioctl(int fd, IOCTL_SAMPLE_CLK_GET,
   dt78xx_clk_config_t *pClk_config);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the input stream (/dev/DT7816-stream-in) or output stream (/dev/DT7816-stream-out).

Name: pClk_config

Data Type: dt78xx_clk_config_t

Description: A pointer to a structure that defines the configuration of the clock.

dt78xx_clk_config_t is defined as follows:

```
typedef struct __attribute__ ((__packed__))
{
  uint8_t     ext_clk;
  uint8_t     ext_clk_din;
  uint8_t     unused[2];
  float       clk_freq;
} dt78xx_clk_config_t;
```

Structure Element Name: ext_clk

Data Type: uint8_t

Description: Specifies the clock source that is used, where 0 is the internal clock source and 1 is the external clock source.

Structure Element Name: ext_clk_din

Data Type: uint8_t

Description: Specifies which general-purpose input signal (pins 1 to 8) on the I/O header is used for the external clock signal.

Pin 1 corresponds to bit 0 of the digital input port (value 0x1).

Pin 2 corresponds to bit 1 of the digital input port (value 0x2).

Pin 3 corresponds to bit 2 of the digital input port (value 0x4).

Pin 4 corresponds to bit 3 of the digital input port (value 0x8).

Pin 5 corresponds to bit 4 of the digital input port (value 0x10).

Pin 6 corresponds to bit 5 of the digital input port (value 0x20).

Pin 7 corresponds to bit 6 of the digital input port (value 0x40).

Description (cont.): Pin 8 corresponds to bit 7 of the digital input port (value 0x80).

For the DT7816, this element is ignored if an internal clock is specified.

Structure Element Name: unused[2]

Data Type: uint8_t

Description: Reserved for future use.

Structure Element Name: clk_freq

Data Type: float

Description: Contains the actual sampling frequency that was set, in Hertz.

For the input stream, if using the internal clock, values range from 100 Hz to 400 kHz. If using the external clock, the maximum value is100 Hz; there is no minimum value.

For the output stream, if using the internal clock, values range from 1 Hz to 400 kHz. If using the external clock, the maximum value is100 Hz; there is no minimum value.

Returns 0 = Success; < 0 = Failure

Notes The actual sampling frequency that can be achieved may be different that the value that was set using the **ioctl - IOCTL_SAMPLE_CLK_SET** command, described on page 127, due to the granularity of the clock.

Example The following example returns the configuration of the sample clock, including the actual sampling frequency that was set for the input stream, *fd_instream*:

```
dt78xx_clk_config_t clk_cfg;
if (ioctl(fd_instream, IOCTL_SAMPLE_CLK_GET,
   &clk_config))
    {
        perror("IOCTL_SAMPLE_CLK_GET");
        goto finish;
    }
```

See Also **IOCTL_SAMPLE_CLK_SET**, described on page 127.

ioctl - IOCTL_SAMPLE_CLK_SET

Description Specifies the configuration of the sample clock for the specified stream.

Syntax

```
int ioctl(int fd, IOCTL_SAMPLE_CLK_SET,
   dt78xx_clk_config_t *pClk_config);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the input stream (/dev/DT7816-stream-in) or output stream (/dev/DT7816-stream-out).

Name: pClk_config

Data Type: dt78xx_clk_config_t

Description: A pointer to a structure that defines the configuration of the clock.

dt78xx_clk_config_t is defined as follows:

```
typedef struct __attribute__ ((__packed__))
{
  uint8_t     ext_clk;
  uint8_t     ext_clk_din;
  uint8_t     unused[2];
  float       clk_freq;
} dt78xx_clk_config_t;
```

Structure Element Name: ext_clk

Data Type: uint8_t

Description: Specifies which clock source is used, where 0 is the internal clock source and 1 is the external clock source.

Structure Element Name: ext_clk_din

Data Type: uint8_t

Description: Specifies which general-purpose input signal (pins 1 to 8) on the I/O header to use for the external clock signal.

Pin 1 corresponds to bit 0 of the digital input port (value 0x1).

Pin 2 corresponds to bit 1 of the digital input port (value 0x2).

Pin 3 corresponds to bit 2 of the digital input port (value 0x4).

Pin 4 corresponds to bit 3 of the digital input port (value 0x8).

Pin 5 corresponds to bit 4 of the digital input port (value 0x10).

Pin 6 corresponds to bit 5 of the digital input port (value 0x20).

Pin 7 corresponds to bit 6 of the digital input port (value 0x40).

Description (cont): Pin 8 corresponds to bit 7 of the digital input port (value 0x80).

For the DT7816, this element is ignored if an internal clock is specified.

Structure Element Name: unused[2]

Data Type: uint8_t

Description: Reserved for future use.

Structure Element Name: clk_freq

Data Type: float

Description: Specifies the sampling frequency, in Hertz.

For the input stream, if using the internal clock, values range from 100 Hz to 400 kHz. If using the external clock, the maximum value is100 Hz; there is no minimum value.

For the output stream, if using the internal clock, values range from 1 Hz to 400 kHz. If using the external clock, the maximum value is100 Hz; there is no minimum value.

Returns 0 = Success; < 0 = Failure

Notes The DT7816 driver sets the frequency of the lock as close as possible to the value that you specify. However, the value that you specify may not be the actual value that is set. To return the actual sample clock frequency that was set, use the **ioctl - IOCTL_SAMPLE_CLK_SET** command.

You can use a different clock source for both the input stream and the output stream. To do this, ensure that you specify a different general-purpose input signal for each clock source.

Example The following example specifies an external clock with a sampling frequency of 100 kHz for the input stream, *fd_instream*. In this case, the external A/D clock is connected to pin 2 (bit 1) of the I/O header:

```
dt78xx_clk_config_t clk_cfg;
clk_cfg.ext_clk = 1;
clk_cfg.ext_clk_din = 0x2;
clk_cfg.clk_freq = 100000.0f;

if (ioctl(fd_instream, IOCTL_SAMPLE_CLK_SET,
   &clk_config))
    {
        perror("IOCTL_SAMPLE_CLK_SET");
        goto finish;
    }
```

See Also **IOCTL_SAMPLE_CLK_GET**, described on page 125.

ioctl - IOCTL_START_SUBSYS

Description

Starts the I/O operation on the DT7816 module.

When the file descriptor is associated with an input stream or output stream, this command starts the operation when a software trigger is specified; it has no effect if any other trigger source is specified.

When the file descriptor is associated with a counter/timer subsystem, this command starts the counter/timer operation. If a software gate type is specified, the operation starts immediately. If the gate type is an external gate, the operation starts when the specified gate signal is detected. See the **ioctl - IOCTL_CT_CFG_SET** command, described on page 86, for more information.

Syntax

```
int ioctl(int fd, IOCTL_START_SUBSYS,
   unint32_t *pSimultaneous);
```

Include File

DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the file for the input stream (/dev/DT7816-stream-in), output stream (/dev/DT7816-stream-out), or counter/timer subsystem (/dev/DT7816-ctr-tmr) of the DT7816.

Name: *pSimultaneous

Data Type: unint32_t

Description: A pointer to a variable that specifies whether or not to start the operations on the input and output streams simultaneously. By default, this value is NULL and operations on the input and output streams are not started simultaneously.

If the file descriptor for the input stream is specified, a non-zero value starts operations on the input stream and the output stream simultaneously. If the file descriptor for the output stream is specified, a non-zero value starts the output stream and the input stream simultaneously.

If the file descriptor for the counter/timer subsystem is specified, this value is NULL.

Returns

0 = Success; < 0 = Failure

Notes

Once an operation is started, you can stop the operation using the **ioctl - IOCL_STOP_SUBSYS** command, described on page 139.

Example The following example configures the input stream, *fd_instream*, for a software trigger, arms the input stream, and then starts acquisition on the input stream.

```
dt78xx_trig_config_t trig_cfg;
    trig_cfg.src = trig_src_sw;
    if (ioctl(fd_stream, IOCTL_START_TRIG_CFG_SET,
       &trig_cfg))
       {
         perror("IOCTL_START_TRIG_CFG_SET");
         goto finish;
       }

ret = ioctl(fd_instream, IOCTL_ARM_SUBSYS, 0);

uint32_t simultaneous = 0;
ret = ioctl(fd_instream, IOCTL_START_SUBSYS,
   &simultaneous);
```

See Also **ioctl - IOCTL_STOP_SUBSYS**, described on page 139.

ioctl - IOCTL_START_TRIG_CNF_GET

Description Returns the configuration of the start trigger used by the specified stream.

Syntax

```
int ioctl(int fd, IOCTL_START_TRIG_CFG_GET,
   struct dt78xx_trig_config_t *pStartTrig_config);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the input stream (/dev/DT7816-stream-in) or output stream (/dev/DT7816-stream-out).

Name: pStartTrig_config

Data Type: dt78xx_trig_config_t structure

Description: A pointer to a structure that defines the configuration of the start trigger.

dt78xx_trig_config_t is defined as follows:

```
typedef struct __attribute__ ((__packed__))
{
  trig_src_t  src;
  union
  {
    struct __attribute__ ((__packed__))
    {
       uint16_t       edge_rising;
       uint16_t       din;
    }ext;

    struct __attribute__ ((__packed__))
    {
       uint16_t       edge_rising;
       uint16_t       ain;
       int32_t        level;
    }threshold;

  } src_cfg;
} dt78xx_trig_config_t;
```

Structure Element Name: src

Data Type: trig_src_t

Description: This enumeration defines the source of the start trigger. It is defined as follows:

```
typedef enum
{
  trig_src_sw,
  trig_src_ext,
  trig_src_threshold
}trig_src_t;
```

where,

- *trig_src_sw* specifies a software trigger as the source of the start trigger. When this trigger source is specified, the operation starts when the **ioctl - IOCTL_START_SUBSYS** command, described on page 129, is executed.
- *trig_src_ext* specifies the external, digital (TTL) trigger as the source of the start trigger. The start trigger occurs when the device detects a transition on the active edge of the external digital (TTL) trigger signal. If you specify this trigger source, the values for the elements of the *ext* structure of *dt78xx_trig_config_t*, described below, are required.
- *trig_src_threshold* specifies the threshold trigger as the source of the start trigger. The start trigger occurs when the signal attached to a specified analog input channel rises above or falls below a user-specified threshold value. If you specify this trigger source, the values for the elements of the *threshold* structure of *dt78xx_trig_config_t*, described on below, are required.

Structure Element Name: edge_rising

Data Type: uint16_t

Description: Used by the *ext* and *threshold* structures, specifies which edge of the trigger signal is the active edge, where 0 is falling edge and 1 is rising edge.

If this element is specified in the *ext* structure, the trigger signal refers the external digital (TTL) trigger.

If this element is specified in the *threshold* structure, the trigger signal refers the threshold trigger.

Structure Element Name: din

Data Type: uint16_t

Description: Used by the *ext* structure, specifies which general-purpose input signal (pins 1 to 8) on the I/O header to use for the external trigger input signal.

Pin 1 corresponds to bit 0 of the digital input port (value 0x1).

Pin 2 corresponds to bit 1 of the digital input port (value 0x2).

Pin 3 corresponds to bit 2 of the digital input port (value 0x4).

Pin 4 corresponds to bit 3 of the digital input port (value 0x8).

Pin 5 corresponds to bit 4 of the digital input port (value 0x10).

Pin 6 corresponds to bit 5 of the digital input port (value 0x20).

Pin 7 corresponds to bit 6 of the digital input port (value 0x40).

Pin 8 corresponds to bit 7 of the digital input port (value 0x80).

Structure Element Name: ain

Data Type: uint16_t

Description: Used by the *threshold* structure, specifies which analog input channel to use for the threshold trigger. For the DT9816, values range from 0 to 7, where 0 represents analog input channel 0 and 7 represents analog input channel 7.

Structure Element Name: level

Data Type: int32_t

Description: Specifies the level at which the threshold trigger occurs. Values depend on the resolution of the module. Since the DT7816 module has a 16-bit A/D converter and uses two's complement data encoding, the positive range is +10 V (0x00007fff) to one code above 0 (0x0) and the negative range is one code below 0 (0xffff8000) to –10 V (0xffffffff).

Returns 0 = Success; < 0 = Failure

Notes The threshold trigger is supported for the input stream only.

The DT7816 driver sets the threshold level as close as possible to the value that you specify. However, the value that you specify may not be the actual value that is set. This command returns the actual threshold level that was set using the **ioctl - IOCTL_START_TRIG_CNF_SET** command.

Example The following example returns the configuration of the start trigger for the input stream, *fd_instream*:

```
dt78xx_trig_config_t StartTrig_cfg;
if (ioctl(fd_instream, IOCTL_START_TRIG_CFG_GET,
   &StartTrig_cfg))
    {
        perror("IOCTL_START_TRIG_CFG_GET");
        goto finish;
    }
```

See Also **IOCTL_START_TRIG_CFG_SET**, described on page 135.

ioctl - IOCTL_START_TRIG_CNF_SET

Description Specifies the configuration of the start trigger used by the specified stream.

Syntax

```
int ioctl(int fd, IOCTL_START_TRIG_CFG_SET,
   struct dt78xx_trig_config_t *pStartTrig_config);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the input stream (/dev/DT7816-stream-in) or output stream (/dev/DT7816-stream-out).

Name: pStartTrig_config

Data Type: dt78xx_trig_config_t structure

Description: A pointer to a structure that defines the configuration of the start trigger.

dt78xx_trig_config_t is defined as follows:

```
typedef struct __attribute__ ((__packed__))
{
  trig_src_t  src;
  union
  {
    struct __attribute__ ((__packed__))
    {
       uint16_t       edge_rising;
       uint16_t       din;
    }ext;

    struct __attribute__ ((__packed__))
    {
       uint16_t       edge_rising;
       uint16_t       ain;
       int32_t        level;
    }threshold;

  } src_cfg;
} dt78xx_trig_config_t;
```

Structure Element Name: src

Data Type: trig_src_t

Description: This enumeration defines the source of the start trigger. It is defined as follows:

```
typedef enum
{
  trig_src_sw,
  trig_src_ext,
  trig_src_threshold
}trig_src_t;
```

where,

- *trig_src_sw* specifies a software trigger as the source of the start trigger. When this trigger source is specified, the operation starts when the **ioctl - IOCTL_START_SUBSYS** command, described on page 129, is executed.
- *trig_src_ext* specifies the external, digital (TTL) trigger as the source of the start trigger. The start trigger occurs when the device detects a transition on the active edge of the external digital (TTL) trigger signal. If you specify this trigger source, the values for the elements of the *ext* structure of *dt78xx_trig_config_t*, described below, are required.
- *trig_src_threshold* specifies the threshold trigger as the source of the start trigger. The start trigger occurs when the signal attached to a specified analog input channel rises above or falls below a user-specified threshold value. If you specify this trigger source, the values for the elements of the *threshold* structure of *dt78xx_trig_config_t*, described on below, are required.

Structure Element Name: edge_rising

Data Type: uint16_t

Description: Used by the *ext* and *threshold* structures, specifies which edge of the trigger signal is the active edge, where 0 is falling edge and 1 is rising edge.

If this element is specified in the *ext* structure, the trigger signal refers the external digital (TTL) trigger.

If this element is specified in the *threshold* structure, the trigger signal refers the threshold trigger.

Structure Element Name: din

Data Type: uint16_t

Description: Used by the *ext* structure, specifies which general-purpose input signal (pins 1 to 8) on the I/O header to use for the external trigger input signal.

Pin 1 corresponds to bit 0 of the digital input port (value 0x1).

Pin 2 corresponds to bit 1 of the digital input port (value 0x2).

Pin 3 corresponds to bit 2 of the digital input port (value 0x4).

Pin 4 corresponds to bit 3 of the digital input port (value 0x8).

Pin 5 corresponds to bit 4 of the digital input port (value 0x10).

Pin 6 corresponds to bit 5 of the digital input port (value 0x20).

Pin 7 corresponds to bit 6 of the digital input port (value 0x40).

Pin 8 corresponds to bit 7 of the digital input port (value 0x80).

Structure Element Name: ain

Data Type: uint16_t

Description: Used by the *threshold* structure, specifies which analog input channel to use for the threshold trigger. For the DT7816, values are 0 to 7, where 0 represents analog input channel 0 and 7 represents analog input channel 7.

Structure Element Name: level

Data Type: int32_t

Description: Specifies the level at which the threshold trigger occurs. Values depend on the resolution of the module. Since the DT7816 module has a 16-bit A/D converter and uses two's complement data encoding, the positive range is +10 V (0x00007fff) to one code above 0 (0x0) and the negative range is one code below 0 (0xffff8000) to –10 V (0xffffffff).

Returns 0 = Success; < 0 = Failure

Notes The threshold trigger is supported for the input stream only.

The DT7816 driver sets the threshold level as close as possible to the value that you specify. However, the value that you specify may not be the actual value that is set. To return the actual threshold level that was set, use the **ioctl - IOCTL_START_TRIG_CNF_GET** command.

Example The following example specifies the configuration of the start trigger for the input stream, *fd_instream*. In this example, a rising-edge threshold trigger on analog input channel 0 is used with a threshold level of approximately +0.5 V (a two's complement value of 0x0000667):

```
dt78xx_trig_config_t StartTrig_cfg;
StartTrig_cfg.src = trig_src_threshold;
StartTrig_cfg.src_cfg.threshold.edge_rising = 1;
StartTrig_cfg.src_cfg.threshold.ain = 0;
StartTrig_cfg.src_cfg.threshold.level = 0x0000667;
if (ioctl(fd_instream, IOCTL_START_TRIG_CFG_SET,
   &StartTrig_cfg))
    {
        perror("IOCTL_START_TRIG_CFG_SET");
        goto finish;
    }
```

See Also **IOCTL_START_TRIG_CFG_GET**, described on page 131.

ioctl - IOCTL_STOP_SUBSYS

Description Stops the I/O operation on the DT7816 module immediately.

Syntax

```
int ioctl(int fd, IOCTL_STOP_SUBSYS, int unused);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the input stream (/dev/DT7816-stream-in), output stream (/dev/DT7816-stream-out), or counter/timer subsystem (/dev/DT7816-ctr-tmr) of the DT7816.

Name: unused

Data Type: int

Description: Not used; reserved for future use.

Returns 0 = Success; < 0 = Failure

Notes When the file descriptor is associated with a counter/timer subsystem, this command stops an operation that was previously started with the **ioctl - IOCTL_START_SUBSYS** command, described on page 129.

When the file descriptor is associated with an input stream or output stream, this command stops an operation that was previously started with the **ioctl - IOCTL_START_SUBSYS** command (a software trigger was specified) or was started when the specified trigger condition was detected. Refer to page 129 for more information on **ioctl - IOCTL_START_SUBSYS**. This command stops the DMA engine immediately and no further data is collected. Asynchronous I/O control blocks that were submitted using **io_submit** are still in the AIO queue and will not be completed. To cancel these control blocks, use **io_cancel**.

If you want to restart the operation for the input stream or output stream, you must rearm the stream using **ioctl - IOCTL_ARM_SUBSYS**, described on page 72, and, if a software trigger is specified, restart the operation using **ioctl - IOCTL_START_SUBSYS**, described on page 129.

Before terminating the application, use **free** to free any events and I/O control blocks and **io_destroy** to destroy the asynchronous I/O context used by the stream.

Example The following example stops acquisition on the input stream, *fd_instream*:

```
ioctl(fd_instream, IOCTL_STOP_SUBSYS, 0);
```

See Also **ioctl - IOCTL_START_SUBSYS, d**escribed on page 129.

ioctl - IOCTL_TACH_CFG_GET

Description Returns the configuration of the tachometer subsystem.

Syntax

```
int ioctl(int fd, IOCTL_TACH_CFG_GET,
   struct dt78xx_tach_config_t *pTach_config);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the tachometer subsystem (/dev/DT7816-tach).

Name: pTach_config

Data Type: dt78xx_tach_config_t structure

Description: A pointer to a structure that defines the configuration of the tachometer subsystem.

dt78xx_tach_config_t is defined as follows:

```
typedef struct __attribute__ ((__packed__))
{
  uint8_t    edge_rising;
  uint8_t    stale_flag;
  };
}dt78xx_tach_config_t;
```

Structure Element Name: edge_rising

Data Type: uint8_t

Description: Specifies whether the rising edge or falling edge of the tachometer is used for the measurement. The number of counts between two consecutive edges of the tachometer input signal is used as the measurement.

If *edge_rising* is 0, the falling edge of the tachometer input signal is used for the measurement.

If *edge_rising* is 1, the rising edge of the tachometer input signal is used for the measurement.

Structure Element Name: stale_flag

Data Type: uint8_t

Description: Indicates whether or not the data is new.

If the *stale_flag* is 1 (Used), the most significant bit (MSB) of the value is set to 0 to indicate new data; reading the value before the measurement is complete returns an MSB of 1.

If *stale_flag* is 0 (Not Used), the MSB is always set to 0.

Returns 0 = Success; < 0 = Failure

Notes You read the tachometer measurement through the input stream by setting bit 8 of the channel mask using the **ioctl - IOCTL_CHAN_MASK_SET** command, described on page 76.

Example The following example returns the configuration of the tachometer subsystem, *tach_file*:

```
if (ioctl(tach_file, IOCTL_TACH_CFG_GET, &tach_cfg))
    {
        perror("IOCTL_TACH_CFG_GET");
        goto finish;
    }
```

See Also **IOCTL_TACH_CFG_SET**, described on page 142.

ioctl - IOCTL_TACH_CFG_SET

Description Specifies the configuration of the tachometer subsystem.

Syntax

```
int ioctl(int fd, IOCTL_TACH_CFG_SET,
   struct dt78xx_tach_config_t *pTACH_config);
```

Include File DT78XX_IOCTL.H

Arguments

Name: fd

Data Type: int

Description: The file descriptor associated with the tachometer subsystem (/dev/DT7816-tach).

Name: pTach_config

Data Type: dt78xx_tach_config_t structure

Description: A pointer to a structure that defines the configuration of the tachometer subsystem.

dt78xx_tach_config_t is defined as follows:

```
typedef struct __attribute__ ((__packed__))
{
  uint8_t    edge_rising;
  uint8_t    stale_flag;
  };
}dt78xx_tach_config_t;
```

Structure Element Name: edge_rising

Data Type: uint8_t

Description: Specifies whether the rising edge or falling edge of the tachometer is used for the measurement. The number of counts between two consecutive edges of the tachometer input signal is used as the measurement.

If *edge_rising* is 0, the falling edge of the tachometer input signal is used for the measurement.

If *edge_rising* is 1, the rising edge of the tachometer input signal is used for the measurement.

Structure Element Name: stale_flag

Data Type: uint8_t

Description: Indicates whether or not the data is new.

If the *stale_flag* is 1 (Used), the most significant bit (MSB) of the value is set to 0 to indicate new data; reading the value before the measurement is complete returns an MSB of 1.

If *stale_flag* is 0 (Not Used), the MSB is always set to 0.

Returns 0 = Success; < 0 = Failure

Notes You read the tachometer measurement through the input stream by setting bit 8 of the channel mask using the **ioctl - IOCTL_CHAN_MASK_SET** command, described on page 76.

Example The following example sets up the tachometer subsystem to measure between two consecutive rising edges of the tachometer signal. In this example, the stale flag is used.

```
#define TACH_DEV_FILE     "//dev/DT7816-tach"
tach_file = open(TACH_DEV_FILE, O_RDWR);
dt78xx_tach_config_t tach_cfg;

tach_cfg.edge_rising = 1;
tach_cfg.stale_flag = 1;

if (ioctl(tach_file, IOCTL_TACH_CFG_SET, &tach_cfg))
    {
        perror("IOCTL_TACH_CFG_SET");
        goto finish;
    }
```

See Also **IOCTL_TACH_CFG_GET**, described on page 140.

open

Description Given a pathname to a file that represents a subsystem, stream, or endpoint of the DT7816 module, opens the file and returns an associated file descriptor for use in subsequent calls.

Syntax

```
int open(const char *pathname, int flags);
```

Arguments

Name: *pathname

Data Type: const char

Description: A pointer to a filename that specifies the file to open. The following pathnames are supported by the DT7816 module:

- **/dev/DT7816-ain** – Analog input subsystem
- **/dev/DT7816-aout** – Analog output subsystem
- **/dev/DT7816-din** – Digital input subsystem
- **/dev/DT7816-dout** – Digital output subsystem
- **/dev/DT7816-tach** – Tachometer subsystem
- **/dev/DT7816-measure** – Measure counter subsystem
- **/dev/DT7816-ctr-tmr** – Counter/timer subsystem
- **/dev/DT7816-stream-in** – Input stream
- **/dev/DT7816-stream-out** – Output stream
- **/dev/DT7816-ep1in** – EP1 IN endpoint, address 0x81
- **/dev/DT7816-ep1out** – EP1 OUT endpoint, address 0x01
- **/dev/DT7816-ep2in** – EP2 IN endpoint, address 0x82
- **/dev/DT7816-ep2out** – EP2 OUT endpoint, address 0x02
- **/dev/DT7816-ep3out** – EP3 OUT endpoint, address 0x03
- **/dev/DT7816-ep4out** – EP4 OUT endpoint, address 0x04
- **/dev/DT7816-ep5out** – EP5 OUT endpoint, address 0x05

Name: flags

Data Type: int

Description: The access mode of the file to open. The following access modes are supported:

- O_RDONLY – Open the file for read only.
- O_WRONLY – Open the file for write only.
- O_RDWR – Open the file for read and write.

Returns The new file descriptor for the file or –1 if an error occurred.

Notes

You must open the file corresponding to each subsystem, stream, or endpoint that you want to use before you can configure its parameters and perform an I/O read or write operation.

A file can be opened by only one reader/writer at a time. However, different files can be opened simultaneously. Once a file is closed, it can be re-opened.

The file descriptor remains open until is closed with the **close** command, described on page 65.

Example

The following command opens the file associated with the input stream of the DT7816 for read only and returns a file descriptor that is associated with the input stream:

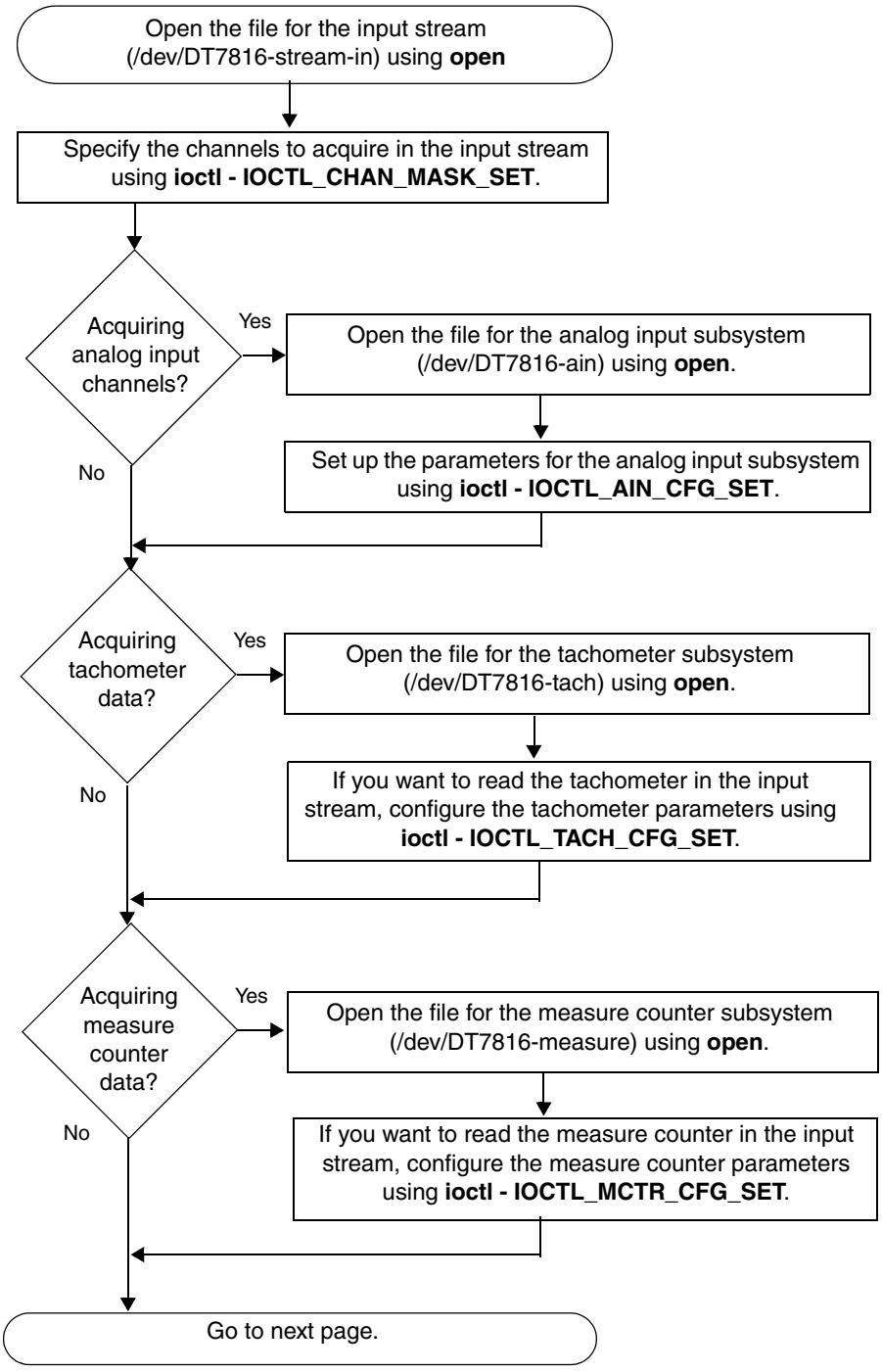
```
#define DEV_STREAM_IN "/dev/DT7816-stream-in"
fd_instream = open(DEV_STREAM_IN, O_RDONLY);
```

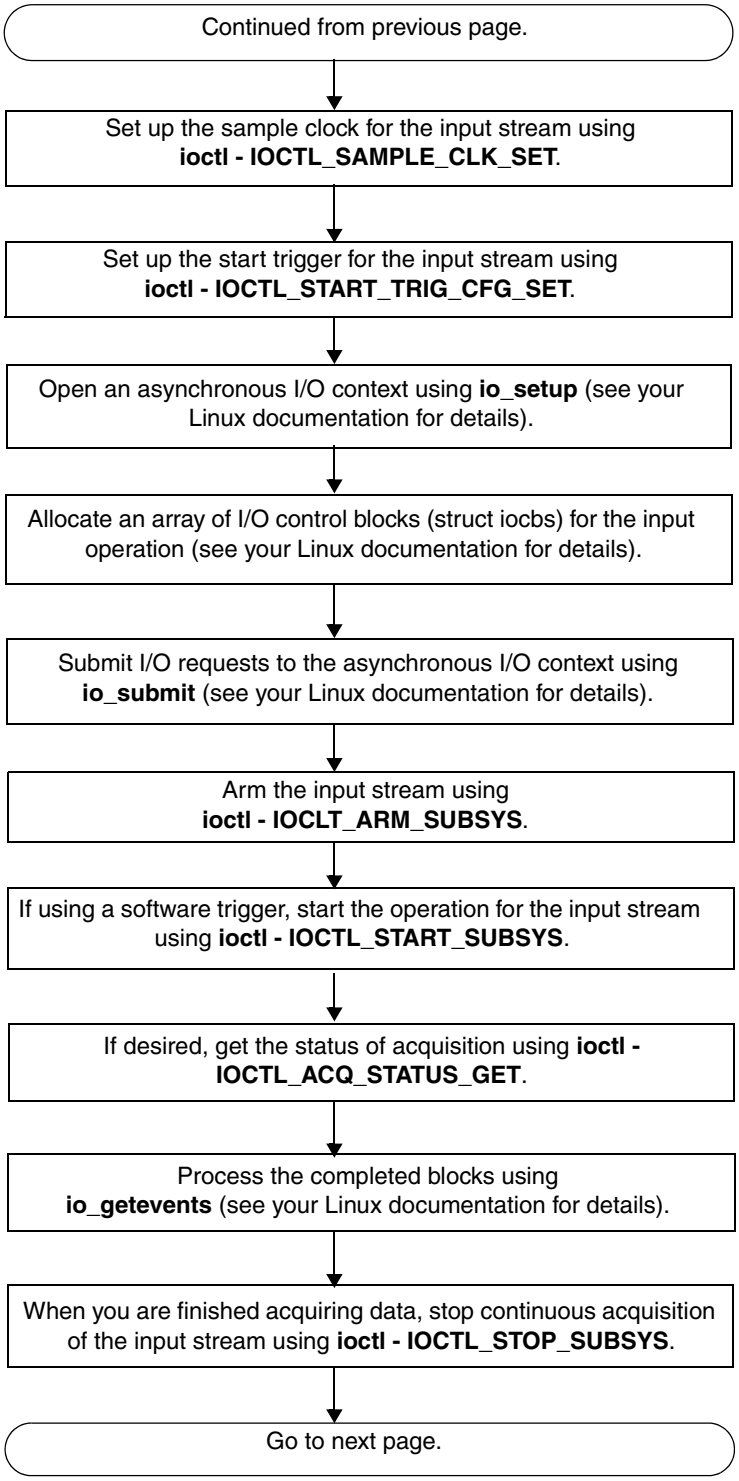
See Also

close, described on page 65.

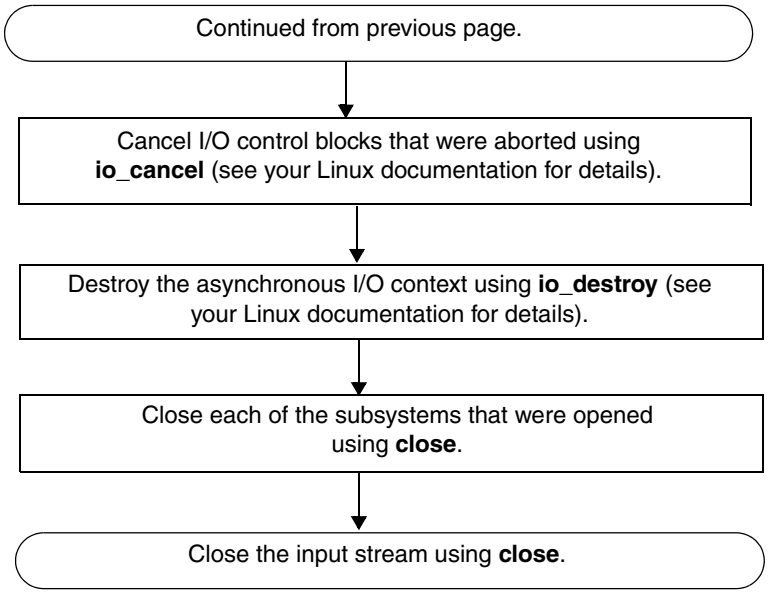
Programming Flowcharts Using the File I/O Commands

Input Stream Asynchronous Read Operations

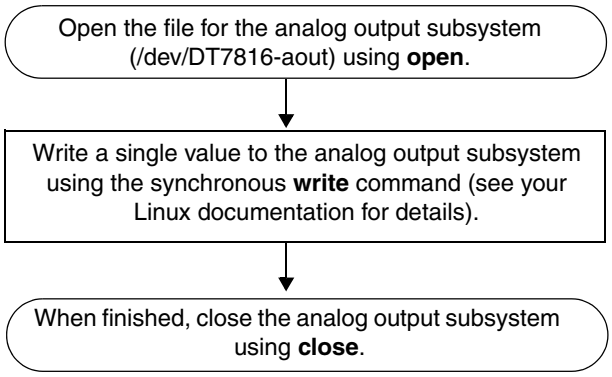




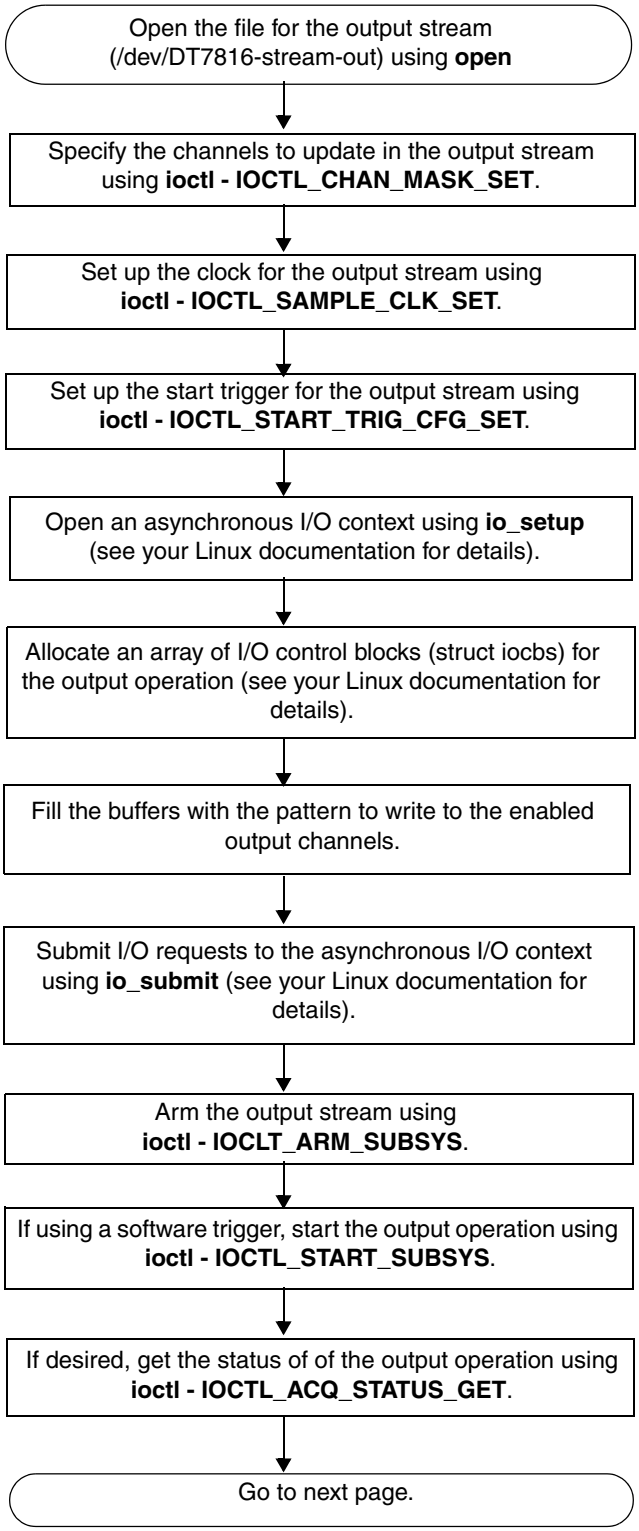
To start a continuous input operation and a continuous output operation simultaneously, specify a non-zero value for the *pSimultaneous* variable


Continued from previous page.
Cancel I/O control blocks that were aborted using **io_cancel** (see your Linux documentation for details).
Destroy the asynchronous I/O context using **io_destroy** (see your Linux documentation for details).
Close each of the subsystems that were opened using **close**.
Close the input stream using **close**.

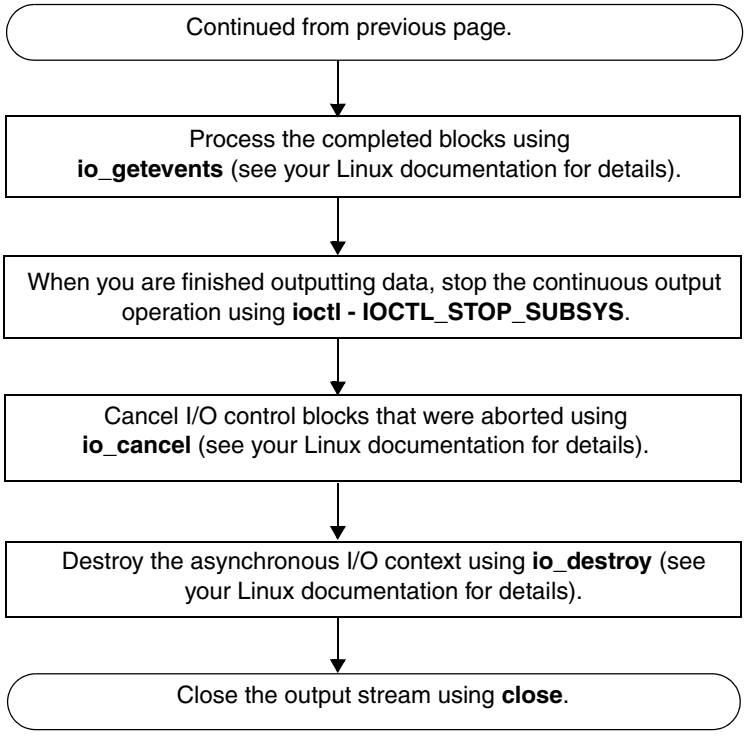
Analog Output Synchronous Write Operation



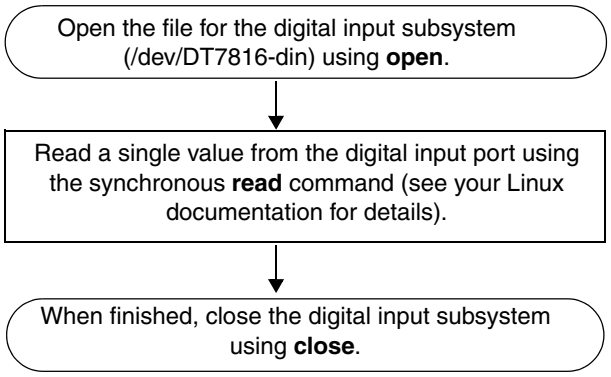
Output Stream Asynchronous Write Operations



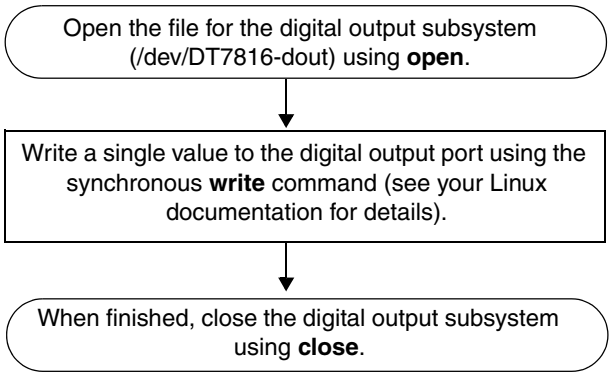
To start a continuous input operation and a continuous output operation simultaneously, specify a non-zero value for the *pSimultaneous* variable


Continued from previous page.
Process the completed blocks using **io_getevents** (see your Linux documentation for details).
When you are finished outputting data, stop the continuous output operation using **ioctl - IOCTL_STOP_SUBSYS**.
Cancel I/O control blocks that were aborted using **io_cancel** (see your Linux documentation for details).
Destroy the asynchronous I/O context using **io_destroy** (see your Linux documentation for details).
Close the output stream using **close**.

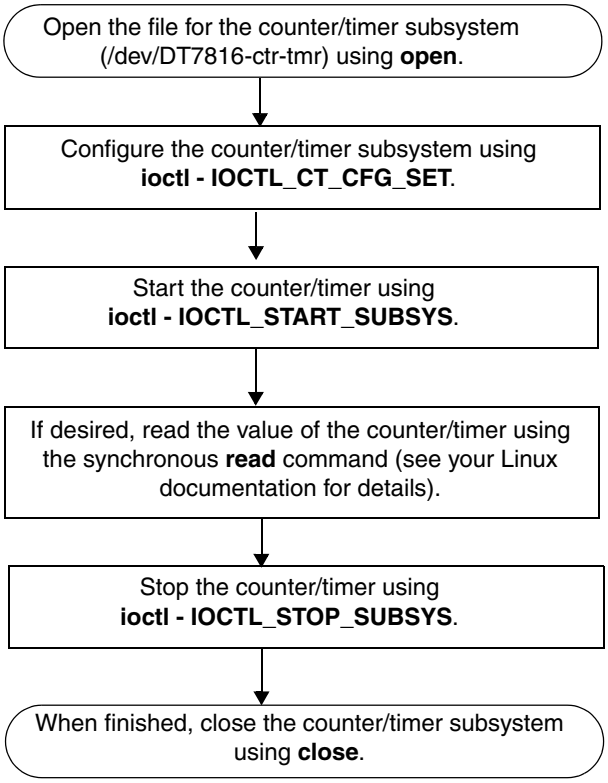
Digital Input Synchronous Read Operation



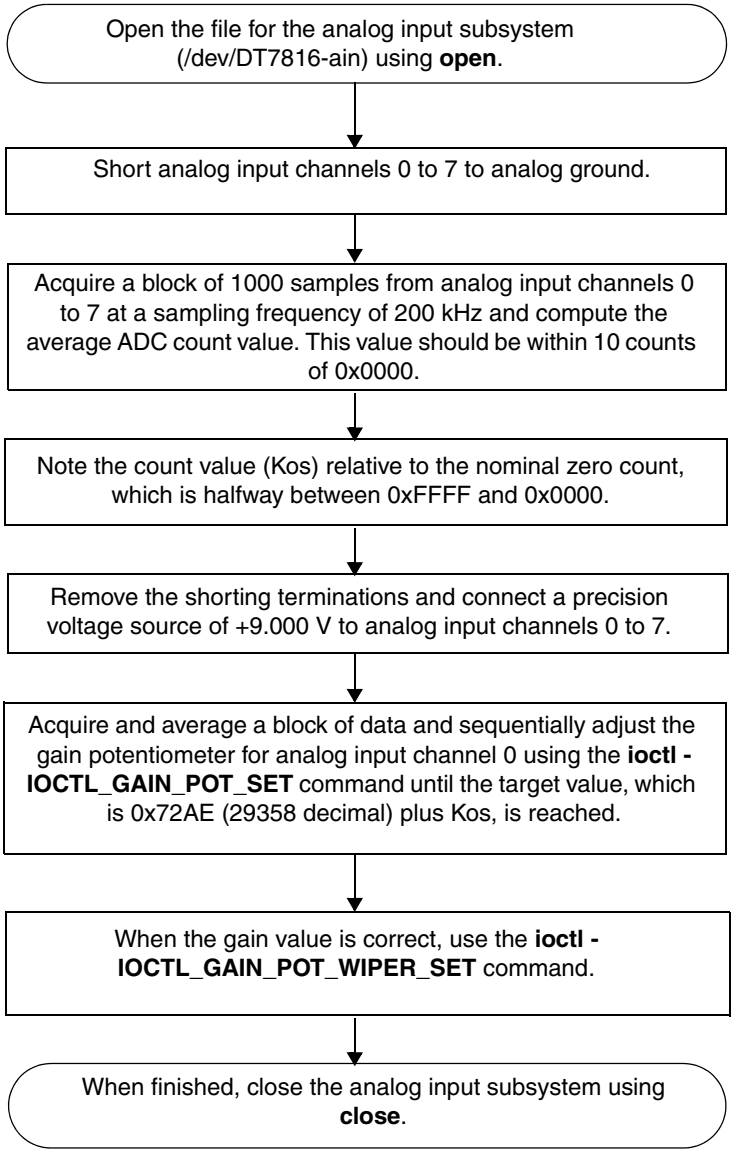
Digital Output Synchronous Write Operation



Counter/Timer Operations

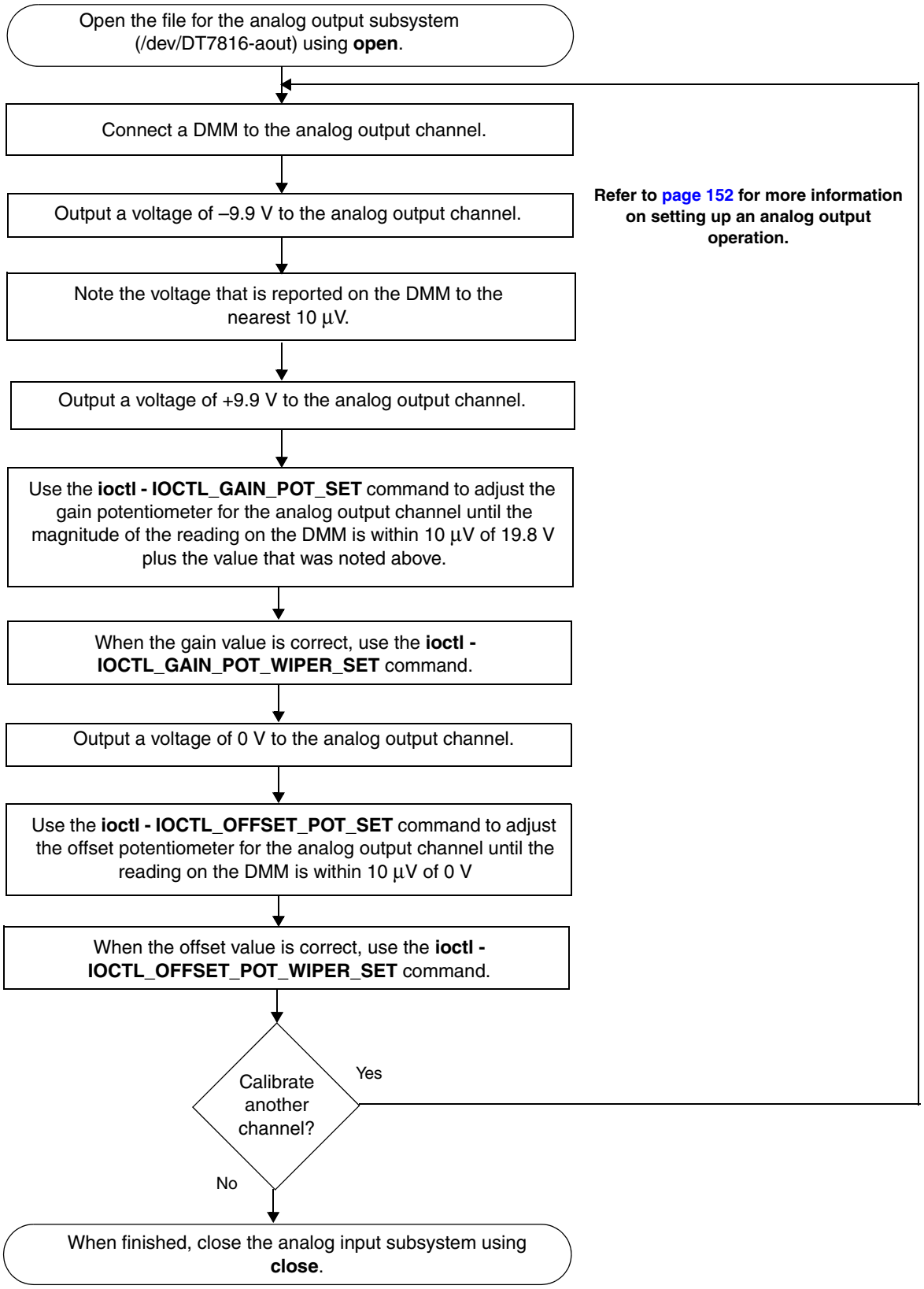


Analog Input Calibration

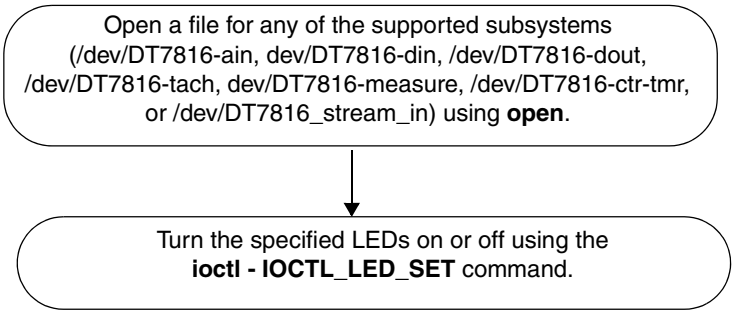


Refer to page 148 for more information on setting up a streaming operation.

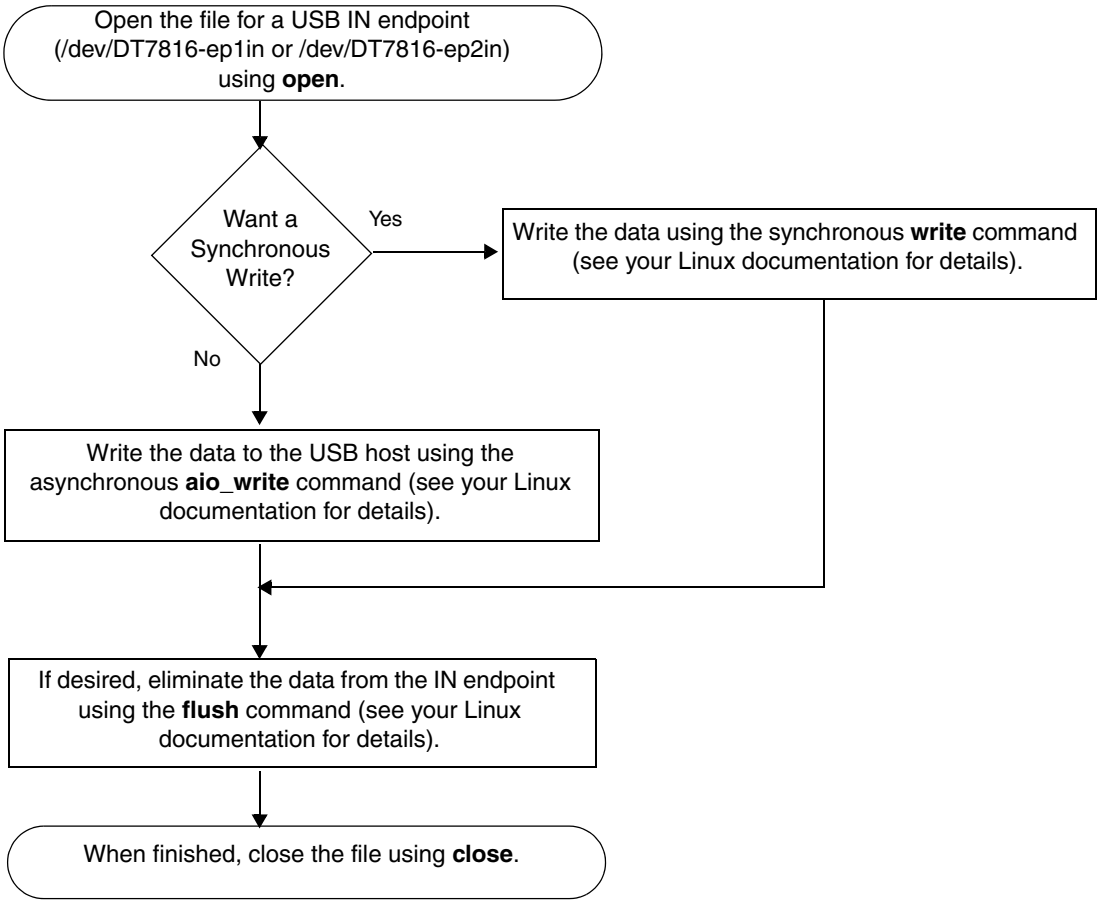
Analog Output Calibration



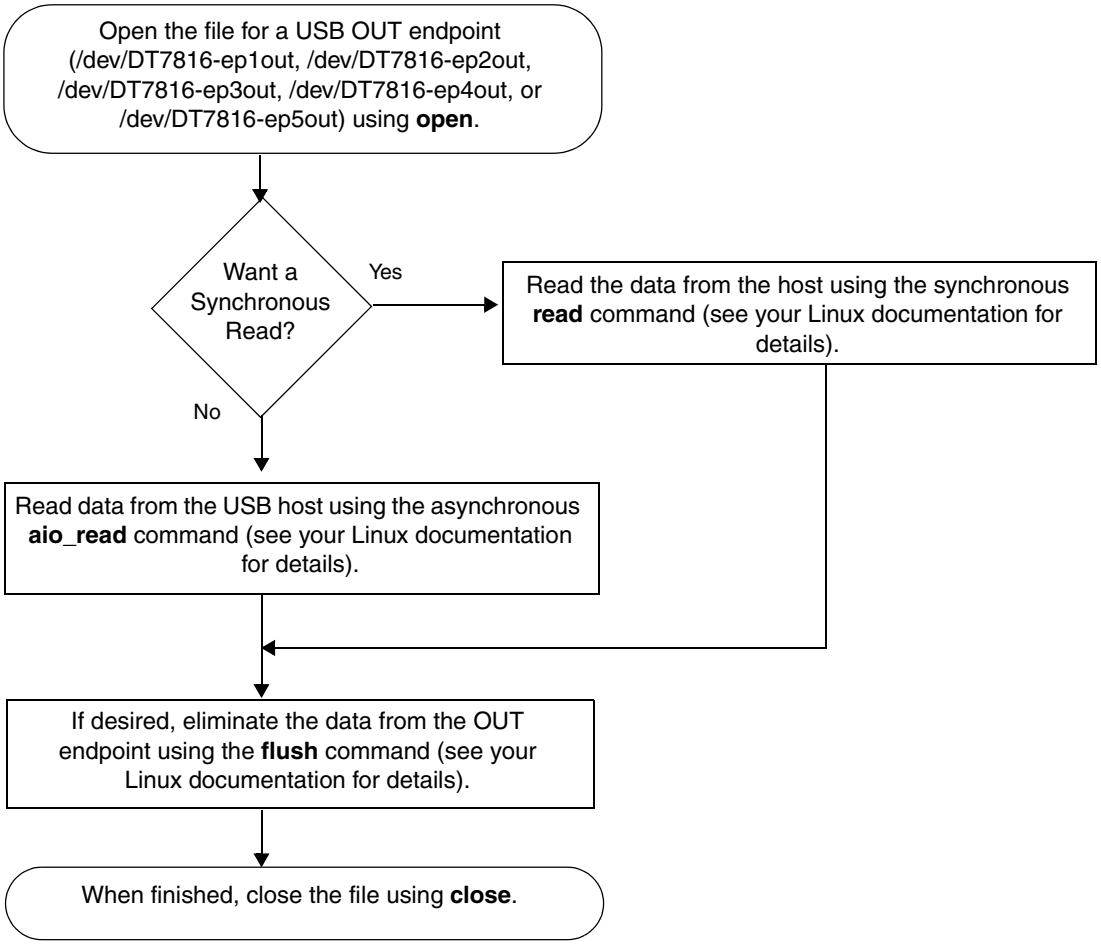
User LED Modifications



Sending Data to a USB Host



Receiving Data from a USB Host



Product Support

Should you experience problems using the file I/O commands to program a DT7816 module, follow these steps:

1. Read all the appropriate sections of this manual and the *DT7816 User's Manual*.
2. Refer to the supplied example programs for clarification.
3. Check that you have installed your hardware devices properly. For information, refer to the *DT7816 User's Manual*.
4. Check that you have installed the software properly. For information, refer to the *DT7816 User's Manual*.

If you are still having problems, Data Translation's Technical Support Department is available to provide technical assistance. To request technical support, go to our web site at http://www.mccdaq.com and click on the Support link.

When requesting technical support, be prepared to provide the following information:

- Your product serial number
- The hardware/software product you need help on

If you are located outside the USA, contact your local distributor; see our web site (www.mccdaq.com) for the name and telephone number of your nearest distributor.

Index

F

G

H

I

J

L

M

N

O

P

R

S

T

U

W