United Electronic Industries

The High-Performance Alternative

PowerDNA CT-601 User Manual

32-bit Interruptible Counter/Timer Layer for PowerDNA Cube

January 2011 Edition
PN Man-DNA-CT-601-0111

Version 3.7

Contacting United Electronic Industries

Mailing Address:

27 Renmar Avenue

Walpole, MA 02081

U.S.A.

For a list of our distributors and partners in the US and around the world, please see http://www.ueidaq.com/partners/

Support:

Telephone: (508) 921-4600
Fax: (508) 668-2350

Also see the FAQs and online "Live Help" feature on our web site.

Internet Support:

Support	support@ueidaq.com
Web-Site	www.ueidaq.com
FTP Site	ftp://ftp.ueidaq.com

Product Disclaimer:

WARNING!

DO NOT USE PRODUCTS SOLD BY UNITED ELECTRONIC INDUSTRIES, INC. AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS.

Products sold by United Electronic Industries, Inc. are not authorized for use as critical components in life support devices or systems. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness. Any attempt to purchase any United Electronic Industries, Inc. product for that purpose is null and void and United Electronic Industries Inc. accepts no liability whatsoever in contract, tort, or otherwise whether or not resulting from our or our employees' negligence or failure to detect an improper purchase.

Specifications in this document may change without notice. Check with UEI for current status.

Table of Contents

Table of Figures

Chapter 1 Introduction

This document outlines the feature-set and use of the DNA-CT-601 layer. This layer is a counter/timer module for the PowerDNA I/O Cube.

1.1 Organization of Manual

This PowerDNA DNA-CT-601 User Manual is organized as follows:

- **Introduction**
  This section provides an overview of PowerDNA Counter-Timer Series board features, the various models available and what you need to get started.
- **The DNA-CT-601 Layer**
  This chapter provides an overview of the device architecture, connectivity, and logic of the CT-601 layer.
- **Programming with the High-Level API**
  This chapter provides an overview of the how to create a session, configure the session for counter input/output, and interpret results on the DNA-CT-601 series layer.
- **Programming with the Low-Level API**
  Low-level API commands for configuring and using the DNA-CT-601 series layer.
- **Appendix A - Accessories**
  This appendix provides a list of accessories available for DNA-CT-601 layer(s).
- **Index**
  This is an alphabetical listing of the topics covered in this manual.

Conventions

To help you get the most out of this manual and our products, please note that we use the following conventions:

Tips are designed to highlight quick ways to get the job done or to reveal good ideas you might not discover on your own.

NOTE: Notes alert you to important information.

CAUTION! *Caution advises you of precautions to take to avoid injury, data loss, and damage to your boards or a system crash.*

Text formatted in **bold** typeface generally represents text that should be entered verbatim. For instance, it can represent a command, as in the following example: “You can instruct users how to run setup using a command such as **setup.exe**.”

1.2 The CT-601 Layer

The Counter/Timer DNA-CT-601 layer is based on PL-601 programmable logic layer. It contains specific counter-timer implementation on FPGA.

The CT-601 has the following features:

- 8 independent counter/timer units
- Clock and Gain Inputs for every counter
- Gate input has programmable polarity, edge sensitive
- Start/pause/stop all channels simultaneously
- 32-bit prescaler per channel; quadrature encoder support
- Multiple period counter (to 2^{32} periods) with accumulated results
- Works with either internal (66 MHz) or external (max 16.5 MHz) timebase
- 256 x 32-bit Input FIFO and 256 x 32-bit Output FIFO on each counter
- Debouncing/glitch removal on external clock and gate inputs
- 16 interrupt sources on every counter
- 8 Counting modes:
  - Timer
  - PWM generator
  - Continuously updated PWM generator (buffered)
  - Bin counter (number of pulses in specified time interval)
  - Pulse width
  - Pulse period (2^{32} periods max)
  - Quadrature encoder
  - Timebase operation
- Clock outputs of CTU0 and CTU1 only, can be divided down and internally routed to the interlayer SYNC bus*
- Clock inputs to every CTU may be connected to either the 66MHz internal timing source or to an external clock (16.5 MHz maximum)
- Protection 7 kV ESD, 350V isolation
- Input Low voltage 0.0-0.8V
- Input High voltage 2.0-5.0V
- Output Low voltage 0.0-0.8V
- Output High voltage 2.0-5.0V
- Output current drive ±12 mA
- Power consumption 2W
- *Refer to Example 601_Input_to_sync_bus.c for functional description.

1.3 Device Architecture

The CT-601 layer consists of two PCBs: the main board PL-60x and the power and isolation board PL-601. All inputs and outputs of this counter-timer are optically isolated and overvoltage protected.

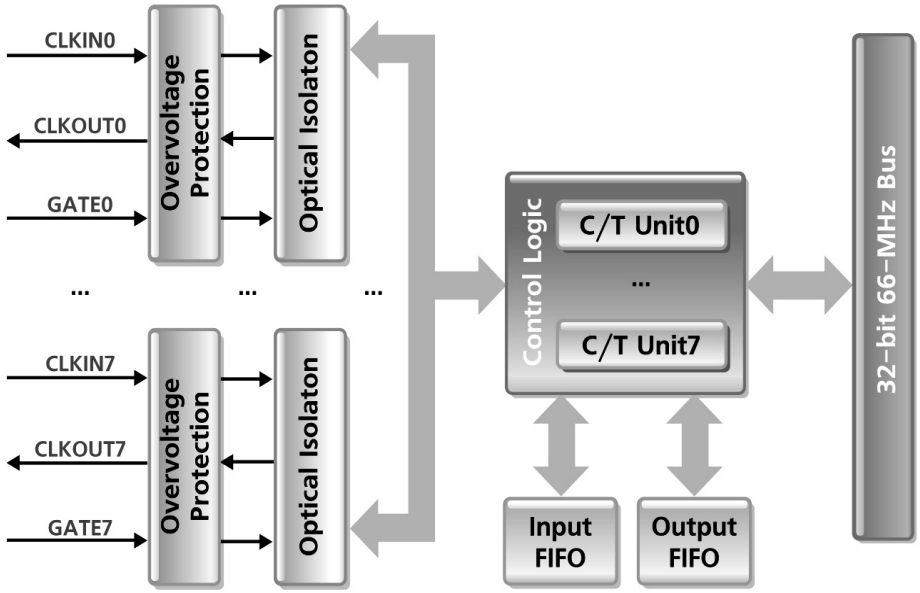

Figure 1-1. Block Diagram of DNA-CT-601

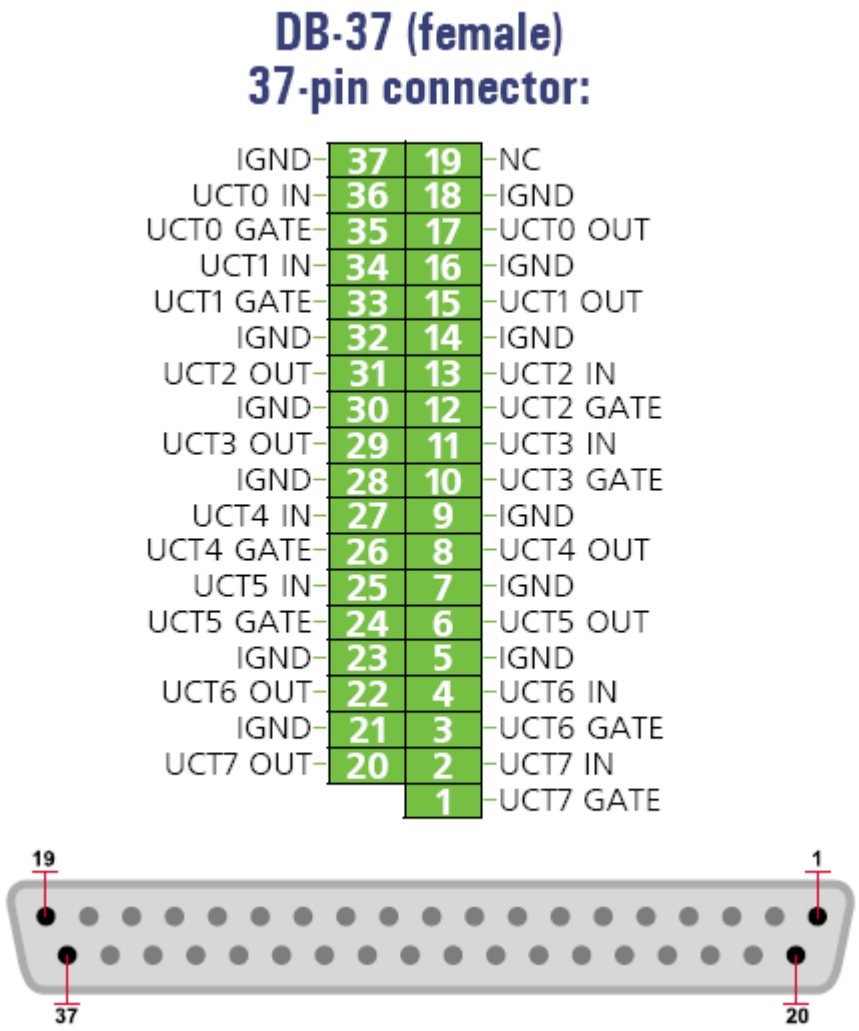

Figure 1-2. Pinout Diagram

All signals are referenced relative to isolated ground (IGND).

1.4 Layer Capabilities

The CT-601 is a very complex layer with eight modes of operation. It can be programmed for a single-point and buffered operations. In single-mode, it performs immediate measurements or one-time or continuous waveform generation.

- **Timer**
  The CT-601 measures time interval or generates clocks on the output
- **PWM Generator**
  The CT-601 generates pulse-width-modulation waveform based on values stored in its registers.
- **Continuously Updated PWM Generator (Buffered)**
  The CT-601 generates a PWM waveform. On a selected timebase counter, it takes new settings for PWM from an internal buffer and loads them into compare registers.

- **Bin Counter**
  The CT-601 bin counter counts a number of pulses in the specified time interval
- **Pulse Width**
  The CT-601 counter measures pulse width in a number of base frequency clocks
- **Pulse Period**
  The CT-601 counter measures pulse periods (2^{32} periods max)
- **Quadrature Encoder**
  The CT-601 measures relative position from quadrature encoder sensor
- **Timebase Operation**
  The CT-601 provides timebase for other counters

The following diagram represents the internal structure of each counter module:

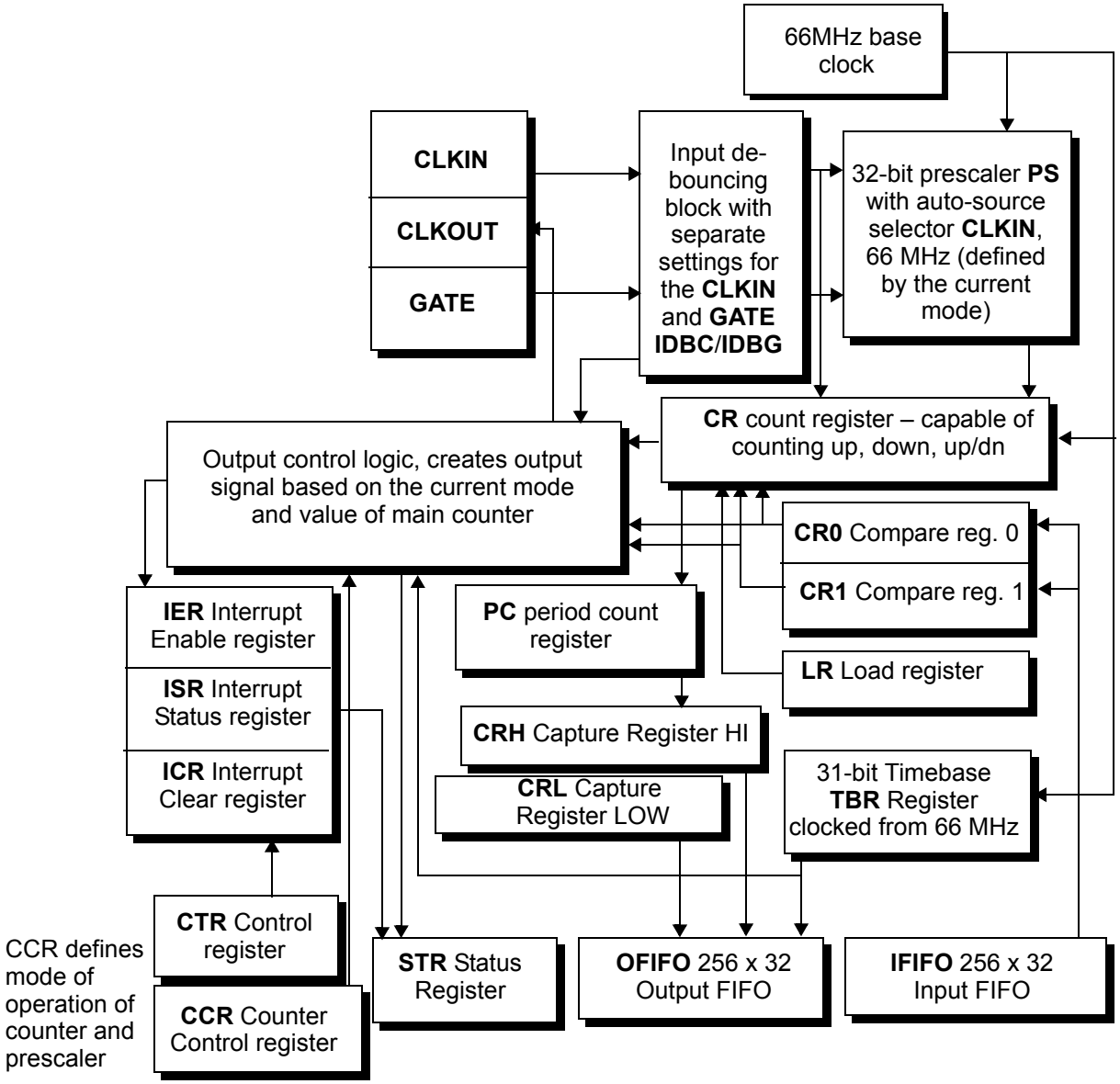

Figure 1-3. Internal Structure of Each Counter/Timer Unit

1.5 Device Description

As illustrated in **Figure 1-3**, the Counter Timer Unit (CTU) has three wires connected to its interface:

- **Input clock line** – CLKIN, used to supply a measured signal or external time base (16.5MHz fmax)
- **Output clock line** – CLKOUT, used to provide access to the output of the CTU, behaves differently depending on the mode selected (33MHz output fmax). Note that in CTU0 and CTU1 only, the clock output can be connected to the inter-layer SYNC bus, if desired.
- **External gate/direction line** – GATE, used to supply external gating signal, start/stop/restart trigger, or direction for the quadrature encoder measurement

Both input lines are connected to the de-bouncing block, which eliminates unwanted spikes from the applied signals. The de-bouncing block is programmed via **IDBC** (`DQ_CTU_IDBC`) and **IDBG** (`DQ_CTU_IDBG`) registers. Each register controls the number of 66MHz clock cycles for which the input signal must be stable before applying it to the internal circuitry. Note that the IDBG must be programmed with 8 times the number of 66 MHz clock cycles for which the clock pin must be stable and the IDBG must be programmed with 64 times the number of 66 MHz clock cycles for which gate pin must be stable.

For instance, an **IDBC** setting of 6600 will give a 12.5 usec clock pin debounce. This means 12.5 usec on the positive transition and 12.5 usec on the negative transition of the input signal giving a 40KHz cutoff. This assumes a squarewave on the input.

The main counter register (**CR**, `DQ_CTU_CR`) may use the external clock, output of the prescaler (**PS**, `DQ_CTU_PS`), or a base clock as a clock source for the counting. The **CR** register counts upward in all modes except the quadrature encoder mode in which it is capable of counting both up- and down.

The load register (**LR**, `DQ_CTU_LR`) is used to supply the initial value from which the counter starts counting.

PS is a configurable 32-bit countdown counter that starts counting from the user-selectable load value down to 0 using either the 66MHz or de-bounced CLKIN signal as a clock source. The clock source is automatically selected, depending on the mode selected. **PS** produces a single 1-internal clock-wide pulse at the end of counting and restarts itself from the load value. If **PS** is loaded with 0, it turns itself into bypass mode, in which output of the **PS** is the same as the time-base register (**TBR**, `DQ_CTU_TBR`) is used to define the pace of the counter captures in some modes.

Two compare registers, **CR0** (`DQ_CTU_CR0`) and **CR1** (`DQ_CTU_CR1`), are used to toggle the output of the counter. One or both may be used, depending on the mode selected. Generally, **CR0** is used to define how long **CLKOUT** output stays LOW and **CR1** is used to define how long it stays high.

A pair of capture registers, **CRH/CRL**, (`DQ_CTU_CRH/DQ_CTU_CRL`), is used in different measurement modes when the counter counts parameters of the input signal.

The control register (**CTR**, `DQ_CTU_CTR`) is used to enable/disable the counter, to access counter/timer pins in general purpose I/O mode, and to enable/disable the inversion mode for the I/O pins and buffered FIFO operation.

The counter control register (**CCR**, `DQ_CTU_CCR`) defines the mode of operation of the counter and prescaler.

The status register (**STR**, `DQ_CTU_STR`) reports the current status of the CTU operation.

The Interrupt Enable register (**IER**, `DQ_CTU_IER`) is used to enable/disable interrupt generation for the various interrupt conditions.

The Interrupt Status register (**ISR**, `DQ_CTU_ISR`) reports the status of the enabled interrupts.

The Interrupt Mask register (**ICR**, `DQ_CTU_ICR`) is used to clear interrupt condition(s) after a CPU processes them.

The Input/Output FIFO allows implementation of buffered I/O modes such as quadrature encoders, etc. Implementation of those modes is planned in the future releases.

Output control logic is responsible for the creation of the output waveform based on the selected mode and current value of the **CR**.

The **PC** period count register is used when measuring a signal that is too fast to read every period. Data from the **CR** is supplied only when measured data is accumulated over N (programmable) periods.

1.5.1 Terminology

- **CTU** - counter-timer unit
- **EM** - end-mode - used to describe end of the current operation performed by the CTU. The following end-modes are defined:
  - **CR** reaches **CR0**
  - **CR** reaches **CR1**
  - **CR** reaches 0xFFFFFFFF
  - X periods of input signal are captured (Note that X is defined by **PC** register and may be ½ period for the ½ period measurement modes)
  - **TBR** timebase register counts down to 0
    Once end-of-count condition is detected by the CTU logic, it may stop operation, or reload/restart it if reload mode is enabled. Optionally, an interrupt may be generated when end-mode condition is detected.
- **CM CTU** operational mode - defines one of the following modes
  - Counter/PWM generator (66MHz base clock used as a **PS** source)
  - External event counter/external clock-driven PWM generator (debounced **CLKIN** clock used as a **PS** source)
  - Capture ½ period mode (**CR** captures ½ period of the input signal starting from the rising edge of the de-glitched input and copies it into **CRH**).
  - Capture full period - **CR** captures length of the full period, copies positive part of the period into **CRH** and negative (low) into **CRL**. If Period Counter > 0, continue this process increasing **CRH/CRL** for the length of the positive/negative part of every period.
  - Quadrature decoder mode - **CR** works as an up/down counter which counts up if **GATE**=1, and down if **GATE**=0. **LR** may be used to load a default value into the counter. **CR0**/**CR1** registers may be used to set an interrupt at boundary limits.

- **Timed Pulse Period Measurement Mode —**
  measures average frequency of incoming pulses over a user-defined time interval

  Also, the first four modes may be used in conjunction with a hardware or a software trigger – one-time or re-triggerable.

  Some of the counting modes are not compatible with some of the end-modes; please refer to the description of the **CCR** register for details.
- **Functional Description of the TPPM Mode—**
  The application can set the measurement time interval and select the desired edge of the incoming signal that should be used for the measurements. The following is a description of the measurement process:

  – The measurement starts after the application configures a timer in the TPPM Mode and enables it. Alternatively, a global software or hardware trigger or a trigger from the gate input can start the operation.
  – The measurement period starts. The length of the measurement period is set by the host application and can vary from 300 nS to approximately 32 seconds.
  – The first selected (rising or falling) edge of the incoming pulse after start of the measurement period resets the internal Time Interval Counter (LTIC) and Pulse Number Counter (PNC) to zero. the TIC increments itself each 66 MHz clock. The PNC increments itself every time the selected edge of the incoming pulse train is detected.
  – Every consequential selected edge of the incoming pulse train current value of the TIC is stored in the intermediate Last Time Interval Counter (LTIC) register.
  – When the measurement period expires, the LTIC register (that represents the time interval between first and last selected edge of the incoming pulse train in 15.15(15)ns counts) and the PNC registers are copied to the Counter register High (CRH) and Counter Register Low (CRL), respectively. Measuring then restarts from #2.

if more than one pulse is detected within the measurement period, this mode of operation will accurately measure average frequency of the incoming signal. If only one or no pulses are detected — the CRH/CRL registers will return zero.

1.5.2 Counter Register (CR) Counting Modes

The following counting modes may be selected for the counter register regardless of the operation mode:

- **Reload mode** – Reload counter with **LR** after it completes the current count operation
- **Count up to CR0** – Counter will count from value loaded into **LR** up to the value in **CR0**. Then, if **LR** bit is set, it will reload itself and continue counting
- **Count up to CR1** – Counter will count from value loaded into **LR** up to the value in **CR1**. Then, if **LR** bit is set, it will reload itself and continue counting
- **Count up to 0xFFFFFFFF** – Counter will count from value loaded into **LR** up to 0xFFFFFFFF. Then, if **LR** bit is set, it will reload itself and continue counting
- **Count in capture mode** – (Capture positive and negative part of the input signal)
- **Count in quadrature encoder mode** — (Up-down mode in which **GATE** pin defines direction of the counting)

1.5.3 CR Output Modes

- **One-shot mode** – Initial value of the output is low. Counter counts from value loaded in **LR** up to value loaded in **CR0** and toggles its output. Mode may be re-triggered by pulse on **GATE** input (user-selectable polarity) or software
- **Universal PWM mode** – Counter counts from **LR** up to value loaded into **CR1** register. Output stays low until counter reaches **CR0** and then stays high until it reaches **CR1**. **GATE** line may optionally be used to start/stop output generation.

  The polarity of the output signal is user-configurable in any mode.

  All output modes may be executed in the buffered mode based on the pace provided by the **TBR**. Currently, the maximum rate/counter should not exceed 10kHz.

1.5.4 CR Input Modes

- **Event counter** – counter counts events. Optional interrupt may be generated when counter reaches **CR0** and/or **CR1**.
- **Width/Period Measurement Mode** with optional hardware trigger – timer (on the programmable edge of the input signal) starts counting the width of the positive and negative parts of the incoming signal and once counted, stores data in **CRH**/**CRL** registers.
- **Quadrature Encoder** – Counter counts incoming pulses using **GATE** line as a direction (programmable).

Chapter 2 Programming with the High-Level API

This section describes how to program the PowerDNA CT-601 using the UeiDaq Framework API.

Since the UeiDaq Framework is object oriented, its objects can be manipulated in the same manner from various development environments such as Visual C++, Visual Basic or LabVIEW.

The following section focuses on the C++ API, but the concept is the same no matter what programming language you use.

Please refer to the “UeiDaq Framework User Manual” for more information about using other programming languages.

Find and refer to the examples that come with UeiDaq Framework. The examples contain detailed and commented code that compiles and executes.

1.1 Creating a Session

The Session object controls all operations on your PowerDNA device. Therefore, the first task is to create a session object:

```
CUeiSession session;
```

1.2 Configuring Channels

Framework uses resource strings to select which device, subsystem and channels to use within a session. The resource string syntax is similar to a web URL:

```
<device class>://<IP address>/<Device Id>/<Subsystem><Channel list>
```

For PowerDNA, the device class is pdna.

For example, the following resource string selects counter input line 0 on device 1 at IP address 192.168.100.2: “pdna://192.168.100.2/Dev1/Ci0”

Use the Session object’s method “CreateCIChannel” to configure the counter/timers you wish to use in input mode.

You can select any of the following modes for input operations:

- `UeiCounterModeCountEvents`: Count pulses.
- `UeiCounterModeMeasurePulseWidth`: Measure the width of a pulse.
- `UeiCounterModeMeasurePeriod`: Measure the period of a signal.
- `UeiCounterModeQuadratureEncoder`: Measure the position of a quadrature encoder.

The source of the input signal can be configured to come from the counter’s external input pin or the counter’s clock.

You can specify when the counting operation will start and stop using an external or a software gate.

You can invert and/or divide the source signal before performing the counting operation.

```
// Configure counter 0 to count events at its external
// input pin.
// Start counting immediately.
```

```
// Don't divide or invert the input signal.
session.CreateCIChannel("pdna://192.168.100.2/Dev0/ci0",
                        UeiCounterSourceInput,
                        UeiCounterModeCountEvents,
                        UeiCounterGateInternal,
                        1, false);
```

Use the Session object's method "CreateCOChannel" to configure the counter/timers you wish to use in output mode.

You can select any of the following modes for output operations:

- `UeiCounterModeGeneratePulse`: Generate a single pulse.
- `UeiCounterModeGeneratePulseTrain`: Generate a pulse train.
- `UeiCounterModePulseWidthModulation`: Generate a pattern.

The source of the reference clock can be configured to come from the counter's external input pin or the counter's clock.

You can specify when the operation will start and stop using an external or a software gate.

For CTU0 and CTU1, you can specify that CLKOUT be internally routed to the SYNC bus.

You can invert and/or divide the clock signal before performing the operation.

You can specify the duty cycle of the signal generated.

```
// Configure counter 0 to generate a pulse train
// Use internal clock to define the shape of the pulses.
// Start generating immediately (internal gate).
// The generated pulses will be in the low state for 100
// clock ticks and in the high state for 200 clock ticks.
// Don't divide or invert the clock signal.
session.CreateCOChannel("pdna://192.168.100.2/Dev0/ci0",
                        UeiCounterSourceClock,
                        UeiCounterModeGeneratePulseTrain,
                        UeiCounterGateInternal,
                        100, 200, 1, false);
```

1.3 Configuring the Timing

You can configure the CT-601 to run in simple mode (point by point) or buffered mode.

In simple mode, the delay between samples is determined by software on the host computer.

In buffered mode, the delay between samples is determined by the CT-601 on-board clock.

The following sample shows how to configure the simple mode. Please refer to the "UeiDaq Framework User's Manual" to learn how to use the other timing modes.

```
session.ConfigureTimingForSimpleIO();
```

1.4 Reading Data

Reading data from the CT-601 is done using a reader object. There is a reader object to read raw data coming straight from the counter line.

The following sample code shows how to create a scaled reader object and read samples.

```
// Create a reader and link it to the session's stream
CUeiCounterReader reader(session.GetDataStream());

// read one scan
unsigned short countedEvents;
reader.ReadSingleScan(&countedEvents);
```

1.5 Cleaning-up the Session

The session object will clean itself up when it goes out of scope or when it is destroyed. However, you can manually clean up the session (to reuse the object with a different set of channels or parameters).

```
session.CleanUp();
```

Chapter 3 Programming with the Low-Level API

This section describes how to program the PowerDNA cube using the low-level API. The low-level API offers direct access to PowerDNA DAQBios protocol and also allows you to directly access device registers.

We recommend that you use UeiDaq Framework (see Chapter 2), which is easier to use.

You should only need to use the low-level API if you are using an operating system other than Windows.

1.1 Data Representation

Counter data is represented as 32-bit words.

1.2 Configuration Settings

Configuration setting are passed in `DqCmdSetCfg()`.

Not all configuration bits apply to the CT-601 layer.

The following bits are used:

```
#define DQ_LN_MAPPED     (1L<<15)  // For WRRD (DMAP) devices
(automatically selected)
#define DQ_LN_ACTIVE     (1L<<1)   // "STS" LED status
#define DQ_LN_ENABLED    (1L<<0)   // enable operations
```

`DQ_LN_ACTIVE` flag is needed to switch on "STS" LED on CPU layer.

`DQ_LN_ENABLE` flag enables all operations with the layer

1.3 Channel List Settings

Channel list settings are currently is not supported.

1.4 Layer-specific Commands and Parameters

The CT-601 layer API provides an extensive set of function to control layer counters. All these functions are designed to set up specific single-point modes of operation for counters. Most of these functions work directly with CT-601 registers. See register definition for details. See the PowerDNA API Reference Manual for a complete description of these functions.

- DqAdv601SetRegister()
  This function writes `value` to the selected `reg` register. `reg` specifies relative offset of the register of interest.
- DqAdv601GetRegister()
  This function reads `value` from the selected `reg` register. `reg` specifies relative offset of the register of interest.
- DqAdv601EnableAll()
  This function enables all counters on the layer.
- DqAdv601DisableAll()
  This function disables all counters on the layer.
- DqAdv601StartCounter()
  This function starts the specified counter with the given configuration.

- DqAdv601StopCounter()
  This function stops the specified counter with the given configuration.
- DqAdv601ClearCounter()
  This function disables the specified counter and clears its configuration.
- DqAdv601ReadRegisterValue()
  This function reads the value from a register `reg` of the `counter` relative to layer `devn`.
- DqAdv601ConfigCounter()
  This function sets up counter configuration.
- DqAdv601CfgForGeneralCounting()
  This function sets up the configuration for a counter in a 601 layer for general counting, without period counting or timebase division. For a working example of this function, refer to the following:
  C;\Progrm Files\UEI\PowerDNA\SDK\Examples\Visual C++ \Sample601CountEvents
- DqAdv601CfgForBinCounter()
  This function sets up the configuration for a counter in a 601 layer for getting the number of counts during a timeframe. Read register for an immediate measurement. For a working example of this function, refer to the following:
  C;\Progrm Files\UEI\PowerDNA\SDK\Examples\Visual C++ \Sample601_BinCounting
- DqAdv601CfgForQuadrature()
  This function sets up the configuration for a counter in a 601 layer for quadrature decoding. For a working example of this function, refer to the following:
  C;\Progrm Files\UEI\PowerDNA\SDK\Examples\Visual C++ \Sample601_Quadrature
- DqAdv601CfgForHalfPeriod()
  This function sets up the configuration for a counter in a 601 layer for half period capture to measure waveform width.For a working example of this function, refer to the following:
  C;\Program Files\UEI\PowerDNA\SDK\Examples\Visual C++ \Sample601_PulseWidthMeasurement
- DqAdv601CfgForPeriodMeasurement()
  This function sets up the configuration for a counter in a 601 layer for getting period measurements. For a working example of this function, refer to the following:
  C;\Program Files\UEI\PowerDNA\SDK\Examples\Visual C++ \Sample601_PeriodMeasurement
- DqAdv601CfgForPWM()
  This function sets up the configuration for a counter in a 601 layer for getting period measurements. For a working example of this function, refer to the following:
  C;\Program Files\UEI\PowerDNA\SDK\Examples\Visual C++ \Sample601_Pulse WidthModulation

- DqAdv601CfgForTPPM()
  This function sets up the configuration for a counter in a 601 layer for Timed Pulse Period Measurement Mode(TPPM). For a working example of this function, refer to the following:
  C:\Program Files\UEI\PowerDNA\SDK\Examples\Visual C++ \Sample601_TPPM

NOTE: The TPPM function described above is available on DNx-CT-601 layers that are at Logic Revision 0102101A or later

- For information about how to configure the 601 to route CLKOUT signals for CTU0 or CTU1 to the SYNC bus, refer to the working code example: C:\Program Files\UEI\PowerDNA\SDK\Examples\Visual C++\ Sample601_DividedInputToSyncBus.c

Appendices

Appendix A - Accessories

The following cables and STP boards are available for the CT-601 layer.

DNA-CBL-37

3ft, 37-way flat ribbon cable; connects DNA-CT-601 to panels

DNA-STP-37

37-way screw terminal panel

DNA-STP-37D

37-way direct-connect screw terminal panel

Appendix B - EEPROM Structures and Constants

The following structure represents content of the layer E^2PROM:

```
/* combined structure to be allocated after CMNDEVS
*/
typedef struct {
    DQEECMNDEVS ee;
    DQOPMODEPRM_601_ opmodeprm;
    DQCNAMES_601_ cname;
} DEVEEPROM_601_, *pDEVEEPROM_601_;

DQEECMNDEVS is a standard E2PROM header described
in 6.2.2.

DQOPMODEPRM_601_ contains data for operating mode.
This data is pre-loaded into the working array on
switching to configuration mode and can be
overwritten before going into operating mode.

typedef struct {
    uint32 conf;       // control word - layer API
flags
    uint32 cvclk;      // CV clock
    uint32 clclk;      // CL clock
    uint32 trig;       // trigger conditions
    int clperint;      // number of channel lists
per interrupt; ignored if <1 or invalid
} DQOPMODEPRM_601_, *pDQOPMODEPRM_601_;
```

Channel names are stored in the following structure (up to 16 characters long):

```
typedef struct {
    char cname[DQ_PL_601_CHAN][DQ_PL_601_NAMELEN];
} DQCNAMES_601_, *pDQCNAMES_601_;

User can set and store these parameters using
DqCmdSetParameters(). See the API Reference Manual
for details.
```

PowerDNA Explorer provides graphical interface for program startup and shutdown states as well as names and operation mode parameters.

Index