Configuration and Status API Overview

EDM04-34

Protection Against Harmful Interference

When present on equipment this document pertains to, the statement "This device complies with part 15 of the FCC rules" specifies the equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the Federal Communications Commission [FCC] Rules.

These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment.

This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction document, may cause harmful interference to radio communications.

Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at their own expense.

Extra Components and Materials

The product that this manual pertains to may include extra components and materials that are not essential to its basic operation, but are necessary to ensure compliance to the product standards required by the United States Federal Communications Commission, and the European EMC Directive. Modification or removal of these components and/or materials, is liable to cause non compliance to these standards, and in doing so invalidate the user’s right to operate this equipment in a Class A industrial environment.

Disclaimer

Whilst every effort has been made to ensure accuracy, neither Endace Technology Limited nor any employee of the company, shall be liable on any ground whatsoever to any party in respect of decisions or actions they may make as a result of using this information.

Endace Technology Limited has taken great effort to verify the accuracy of this document, but nothing herein should be construed as a warranty and Endace shall not be liable for technical or editorial errors or omissions contained herein.

In accordance with the Endace Technology Limited policy of continuing development, the information contained herein is subject to change without notice.

Website

http://www.endace.com

Copyright 2011 Endace Technology Ltd. All Rights reserved.

Use of the Endace products described in this document is subject to the Endace Terms of Trade and the Endace End User License Agreement (EULA).

Contents

Introduction

Overview

The Endace Configuration and Status Application Programming Interface (API) enable developers to configure the components and associated attributes of an Endace Data Acquisition and Generation (DAG) card.

It allows third-party developers to perform the following tasks from within their own application software:

- Reset a DAG card.
- Load firmware images onto a DAG card.
- Set and retrieve the hardware configuration.
- Retrieve status and statistics information.

Version

The information in this document is up to date as of DAG software version 3.4.2. Please see the release notes for your software version for a list of supported DAG cards and operating systems.

Purpose

The purpose of this Overview is to:

- Provide general information about the Configuration & Status API.
- Describe the use of components and attributes associated with DAG cards by the API.
- Describe the types of functions provided by the API.
- Describe the use of data structures and constants by the API functions.

Thread Safety

Note:
The routines described in this Programming Guide are ***not*** *thread safe or re-entrant. If you are using multiple threads, Endace strongly recommends that you use wrapper functions to serialize access to the Endace supplied routines.*

Support

Endace Website

In the event that you experience problems with any Endace supplied hardware, or software, it is recommended that you visit the Endace website at http://www.endace.com. This website includes a *Support* page which offers a range of online assistance options including a Public Knowledge Base. It also allows you to submit a problem report online via the *Online Case Submission* link.

If you have a support contract with Endace you have access to the secure support website. Use your support logon details to gain access. This contains the latest versions of software, device drivers, firmware, user manuals, and release notes.

For more information about the Endace Support Package, or how to obtain (or change) your secure support website login details, please contact sales@endace.com.

If you are unable to resolve a problem using the information on the website, you can email Endace Technical Support at support@endace.com for further assistance.

Reporting Problems

When reporting a problem please supply as much information as possible. The more information you supply the quicker Endace Technical Support are able to effectively respond to you. Although the exact information available may be limited by the type of problem you are experiencing, you should try to supply the following:

- DAG card model and serial number.
- DAG software version in use as returned by `rpm -q dag-base`
- System log messages generated when DAG device driver is loaded. These can be collected from command `dmesg`, or from log file `/var/log/syslog`.
- Output of `daginf`.
- Firmware versions from `dagrom -x`
- Card configuration as reported by: `dagconfig`
- Network link statistics reported by: `dagconfig -sei`
- Network link configuration from the router where available.
- Contents of any scripts in use.
- Complete output of session where error occurred including any error messages from DAG tools. The `typescript` Unix utility may be useful for this.
- A small section of captured packet traces illustrating the problem.

API Overview

Overview

Each DAG card consists of multiple processing *modules,* each of which can have several *configurations.* In order to bring a module into a required configuration, control data must be written into registers inside the DAG card. The Configuration and Status API provides a high-level method of accessing these registers and allows the user to control the behavior of the DAG cards within their C or C++ programs.

Components

The model of a DAG card reported by the API is a hierarchical tree of *components* where each component corresponds to a functional block such as packet processor, PCI burst manager, physical interface, or hardware monitor. The top component in the tree is called the *root* component which contains a reference to the attached DAG card and subcomponents.

Card Configuration

Attribute Reference

Before a DAG card's components configuration can be modified it is necessary to:

1. obtain a reference to the DAG card,
2. obtain a reference to the desired component, and
3. obtain a reference to the component's attribute

that you wish to change.

The attribute reference can then be used to retrieve and modify the attribute value.

For example, to see if a particular port is active, first obtain a reference to the DAG card, then a reference to the port component, and finally a reference to the `active` attribute.

Alternatively, you can directly access the attribute on a DAG card:

1. obtain a reference to the DAG card, then
2. obtain the reference to the attribute by specifying its code and index in the DAG card.

Attribute Value

Reading the value returned by the attribute reference provides information about the attribute status. Writing a value to the attribute reference configures the attribute status.

A sample program is shown in Example Program (page 11).

Attribute Type

There are two types of attributes associated with components on DAG cards.

1. Configuration attributes: used to represent configuration information.
2. Status attributes: used to represent status and statistics information.

The `dag_config_get_attribute_config_status` function may be used to identify if an attribute is marked as a status or configuration attribute. More information is available on this function in `dag_config.h`.

Configuration Attribute

Configuration attributes represent properties of the DAG card that can be modified. They include such items as:

- POS or ATM mode for SONET DAG cards
- Auto-negotiation mode on/off for Ethernet DAG cards
- Variable or fixed-length packet capture
- Snap length for packet capture
- Amount of memory allocated to each receive and transmit stream

Status Attribute

Status attributes represent the card properties that are read-only and can not be modified. They include such items as:

- Physical layer error indicators.
- PCI bus speed.
- Number of frames that failed the Frame Checksum.
- Number of receive and transmit streams supported by the firmware.

Note:
The precise set of attributes and components presented by the API depends on the model of DAG card and the capabilities of the loaded firmware image(s) see Displaying a DAG Card's Components and Attributes (page 9).

Using the API

Pre-requisites

This document assumes the user has experience of the C programming language and is familiar with the operating systems and distribution installed.

Header Files

In order to use the Configuration & Status API the following header files must be included:

- `dag_config.h`
  Contains routines that relate to the card as a whole e.g. getting an initial reference to the card, loading firmware, finding a component by name, as well as routines that retrieve and set values on attributes.
- `dag_component.h`
  Contains routines that operate on components, e.g. getting the root component, getting subcomponents, getting attributes of a component.
- `dag_component_codes.h`
  Contains the codes used to reference components. e.g. `kComponentStream`
- `dag_attribute_codes.h`
  Contains the codes used to refer to attributes and enumerated types for attributes that have a restricted range of valid values. e.g. `kBooleanAttributeVarlen`

Alternatively, include the file `dag_config_api.h` which includes the four header files listed above.

FreeBSD/Linux

On FreeBSD or Linux operating systems the header files are installed in `/usr/local/include` by default. Library files are installed in `/usr/local/lib` by default.

These locations can be changed when running the `configure` script.

Windows

On Windows operating systems the header files are installed in `<Program Files>\Endace\dag-x.y.z\include`. Stub library files are installed in `<Program Files>\Endace\dag-x.y.z\lib\windows\VCproject\Release` and Runtime library files are installed in `<System>`.

Note:
The phrases in <> are standard system locations and may vary from machine to machine.

Components and Attributes

Overview

The model of a DAG card reported by the API is a hierarchical tree of components. The top component in the tree is called the root component which contains a reference to the attached DAG card and subcomponents.

Each subcomponent of the root has a set of attributes associated with it which defines the configuration of the module at any point in time. Changing the value of the component attributes directly changes the behavior of the corresponding modules.

Note:
Not all components and attributes are common to all DAG Cards.

Displaying a DAG Card's Components and Attributes

To display a list of a DAG card's components and attributes, run this command at a prompt:

```
dagconfig -T -v2
```

Below is an example output of this command, taken from a DAG 4.5G4 card. The DAG card's components (`card_info`, `gpp` and `pbm` in the example below) and its associated attributes (`user_fw`, `factory_fw`, etc.) are displayed.

Notes:

- *The output file is in CSV (comma-separated value) format which allows you to import the data into an Excel spreadsheet for easier use.*
- *Some output has been omitted for simplicity.*

```
Component: name= card_info, code=41(kComponentCardInfo), description= <undescribed> ,
Attributes: count= 8,
Attribute: name  ,  type  ,  value  ,  description
  user_fw        , status , edag45g4pci_dso_v2_3 2vp30ff1152 2008/04/03 20:43:33 , User
firmware
  factory_fw     , status , edag45g4pci_dso_v2_3 2vp30ff1152 2008/04/03 20:43:33 , Factory
firmware
  active_fw      , status , factory    , active firmware
  serial_id      , status , 3007076    , Card Serial ID
  copro_type     , status , Not Supported , Co-processor type
  pci_info       , status , 0000:05:01.0 , Physical slot information
  pci_device_code, status , 0x454e     , PCI device code
  board_rev      , status , 2          , Board revision.

Component: name= gpp, code=11(kComponentGpp), description= The size reduced gpp. ,
Attributes: count= 12,
Attribute: name  ,    type , value , description
  snap_length    , config  , 10240 , Get/set the snaplength. Accepts any value, but the
value will be rounded to a multiple of the ERF record alignment.
  varlen         , config  , on    , Enable or disable variable length capture
  interface_count, status  , 4     , Number of interfaces in the card.
  align64        , config  , on    , Align/pad the received ERF record to the next 64-bit
boundary.
  drop_count0    , status  , 0     , A count of the packets dropped on a port
  drop_count1    , status  , 0     , A count of the packets dropped on a port
  drop_count2    , status  , 0     , A count of the packets dropped on a port
  drop_count3    , status  , 0     , A count of the packets dropped on a port
  active0        , config  , on    , Enable/Disable Port0
  active1        , config  , on    , Enable/Disable Port1
  active2        , config  , on  , Enable/Disable Port2
  active3        , config  , on  , Enable/Disable Port3

Component: name= pbm, code=24(kComponentPbm), description= The PCI Burst Manager ,
Attributes: count= 6,
Attribute: name   , type   , value  , description
  pci_bus_speed   , status , 133MHz , A number representing the PCI bus speed
  buffer_size     , status , 32     , The size of the buffer allocated to the DAG card.
  tx_stream_count , status , 1      , The number of transmit streams
  rx_stream_count , status , 2      , The number of receive streams.
  overlap         , config , off    , Share the memory hole between the receive and 
transmit streams.
  drop            , config , off    , If on dropping of packets occurs at the individual 
stream that has filled up. If off dropping occurs at the gpp.
```

Example Program

The following program illustrates how to use the Configuration & Status API. The program performs the following actions:

- Executes the DAG card's default configuration routine
- Counts the number of ports on the DAG card
- Finds the line rate attribute of each port, and
- Sets the line rate to 100Mbps.

For the sake of clarity, the error-handling code has been omitted from this example.

Notes:

- *This example is only applicable to DAG cards capable of 100Mbps line rates, please refer to your DAG Card User Guide for more information.*
- *The line rate being set at the new value is dependant on the DAG card communicating with the SFP.*

```
#include "dag_config_api.h"
#include <stdio.h>
#include <stdlib.h>

int main(int argc, const char* argv[])
{
        dag_card_ref_t card_ref = NULL;
        dag_component_t root_component = NULL;
        uint32_t count;
        uint32_t i;

        /* Get a reference to the card. */
        card_ref = dag_config_init("/dev/dag0");

        /* Get a reference to the root component. */
        root_component = dag_config_get_root_component(card_ref);

        /* Configure the card to default state. */
        dag_config_default(card_ref);

        /* Count the ports on the card. */
        count = dag_component_get_subcomponent_count_of_type(root_component,
kComponentPort);

        for (i = 0; i < count; i++)
        {
                dag_component_t port = NULL;
                attr_uuid_t line_rate_uuid = 0;
                uint32_t val = kLineRateEthernet100;
                dag_err_t err_status = 0;

                /* Get a reference to the port. */
                port = dag_component_get_subcomponent(root_component,
kComponentPort, i);

                /* Get a reference to the line rate attribute of the port. */
                line_rate_uuid = dag_component_get_config_attribute_uuid(port,
kUint32AttributeLineRate);

                /* Set the value of the attribute. */
                dag_config_set_uint32_attribute(card_ref, line_rate_uuid, val);
        }
        /* Dispose of the card. */
        dag_config_dispose(card_ref);

        return EXIT_SUCCESS;
}
```

Functions

Overview

This chapter describes:

- the purpose of the different functions,
- where to find the functions available, and
- the common designators (parameter names) which are used by each function type.

The Configuration and Status API contains five types of functions:

- Card configuration functions
- Component functions
- Attribute accessor functions
- Modifier functions
- Firmware functions.

Card Configuration Functions

Card configuration functions directly configure the DAG card. The DAG card configuration functions are located in `dag_config.h`.

The following designators are used in DAG card configuration functions:

Designator	Description
`card`	Refers to a DAG card.
`uuid`	An attribute identifier.
`component`	Refers to a component.
`device name`	The name of the DAG card. In Linux this should look like `/dev/dag0` and in Windows like `dag0`.
`String`	The value for the string in attribute form.
`attr_code`	The code of the attribute to retrieve.
`attr_index`	The index of the attribute to retrieve. Index starts from 0.

Component Functions

Component functions refer to functions which configure or retrieve components on the DAG card. The component functions are located in `dag_component.h`.

The following designators are used in component functions:

Designator	Description
`attribute`	The code of the attribute to retrieve.
`component`	Refers to a component.
`component code`	See the card specific chapters earlier in this programming guide for a list of valid component codes.
`index`	The index of the attribute to return.
`name`	The name of the sub-component to return.
`code`	The desired sub-component to count.

Attribute Accessor Functions

Attribute accessor functions retrieve the value of an attribute. The only difference between the functions is the type of value they return. The accessor functions are located in `dag_config.h`.

The following designators are used in accessor functions:

Designator	Description
`card`	Refers to a DAG card.
`uuid`	An attribute identifier.
`component`	Refers to a component.

Modifier Functions

The modifier functions assign a value to an attribute. The only difference between them is the type of value they assign. The modifier functions are located in `dag_config.h`.

The following designators are used in modifier functions:

Designator	Description
`card`	Refers to a DAG card.
`uuid`	An attribute identifier.
`value`	The value to assign to the attribute.

The following values are returned by modifier functions:

- `kDagErrInvalidCardRef` is returned if the card reference is invalid.
- `kDagErrNone` is returned on success.

Firmware Functions

The firmware functions load or read firmware on a DAG card. The functions all return the same following function: `kDagErrNone`. The firmware functions are located in `dag_config.h`.

The following designators are used in firmware functions:

Designator	Description
`name`	The name of the device.
`card ref`	A valid pointer to a `dag_ref_t`.
`filename`	The name of the image to load.
`whch_pp`	The index starting from 0 of the packet processor to load.
`buffer`	A buffer to hold the SWID read from the DAG card. It should be at least 128 bytes.
`length`	The size of the buffer in bytes.
`key`	The key to match the key in the ROM. If this key does not match, the Software ID (SWID) write will fail.

Data Structures and Constants

Data Structures are used by the API functions to refer to modules on the DAG card.

Attribute accessor data structures are declared in `dag_attribute_codes.h`, component data structures are declared in `dag_component_codes.h` and other data structures are declared in `dag_config.h`.

Version History

Version	Date	Reason
1	September 2009	First release.
2	November 2009	Updated Displaying DAG cards components and attributes section
3	November 2011	Updated branding.

endace.com