Counters and Statistics API

EDM04-25

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

Overview

This document describes the implementation of the Counters and Statistics Interface (CSI). This interface facilitates the reading of the various counters and statistic registers on any type of DAG card. The counters and registers characteristics are displayed using `dagconfig` commands.

Please be aware that this document is subject to change as additional functionality becomes available.

The CSI covers the DAG Register Bus (DRB) space and is implemented by the firmware. The software implementation considers this interface as a firmware component/module.

The CSI enables the automatic identification by software of the presence of counters or informational bits from a predefined set. Each counter and information bit in this predefined set has a set functionality, allowing different combinations of counters and functional bits to be used in different DAG cards without the requirement of additional software.

Purpose

- To make an easy translation of the existing firmware module statistics into the CSI. These counters are then recognized by the DAG software.
- To give both the firmware and software unique counter and deterministic Id's to prevent name changes and duplications across different cards and images.
- Allow customers to create specific statistics.
- Give the option for accumulated counters in near future and provide backwards compatibility with the software.

Function definitions are described in later chapters of this document.

Description

The main enumeration table stores entry(ies), of type CSI per 'counter statistics interface', which points to a CSI block in the DRB address space. The enumeration entry has different versions depending on the CSI format (type of data access: direct v0 or indirect v1).

CSI Blocks

CSI blocks are mapped to a firmware component. Each CSI block has a description field indicating:

- the number of counters,
- the type of the counters in the block (firmware module based or functional based),
- the 'latch and clear' set up.

Global latch and clear can be implemented at later stage through a single write-only DRB register instantiated in each CSI block.

Each CSI block is implemented in the Configuration and Status API with the following attributes:

- Counter Statistics Interface type (`kUint32AttributeCSIType`),
- Number of counters in CSI (`kUint32AttributeNbCounters`),
- Latch & Clear set up (`kBooleanAttributeLatchClear`),
- Counter description base address (`kUint32AttributeCounterDescBaseAdd`),
- Counter value base address (`kUint32AttributeCounterValueBaseAdd).`

The post-initialization function creates the counter(s) and initializes their state structure.

Counters

Each counter is mapped to a firmware sub-component and is associated with the CSI block.

Each individual counter has a 32-bit description entry containing:

- The counter ID which is unique and depending on the function is implemented when applicable.
- A sub-function ID which covers multiple streams, filters or interfaces ports.
- A "block value" type (counter value or address). This determines whether it is the counter value (0) or the address (1) where the counter value is stored.
- The 'Latch and Clear' information.
- The size of the counter, either 32 bits or 64 bits.
- The type of access, either Direct or Indirect.
- The Base address of the counter value.

The counter is implemented in a CSI block with these attributes:

- Counter ID (kUint32AttributeCounterID)
- Sub-function (kUint32AttributeSubFunction)
- Value type (kBooleanAttributeValueType)
- Latch and Clear information (kBooleanAttributeLatchClear)
- Counter size (kUint32AttributeCounterSize)
- Type of access (kBooleanAttributeAccess)
- Counter value (kUint32AttributeCounterValue)
- Sub-function (KUnit32AttributeSubFunctionsType)

The state structure contains:

- the index of the subcomponent,
- the address or offset (from the DRB base) of the description field,
- the address or offset (from the DRB base) of the counter value.

For example:

```
typedef struct
{
 uint32_t mIndex;
 uint32_t mValueOffset;
 uint32_t mDescrOffset;
 uint32_t* mValueAddress;
 uint32_t* mDescrAddress;
 } counter_state_t;
```

Command

The command used to display the counters relevant to your DAG card is: `dagconfig -u`

Structures

This chapter explains the various data structures which are used in the implementation of counters by the Configuration and Status API.

Counters

The following explains the structure of a counter, `dag_counter_value_t`.

Looking at the structure below:

- `dag_counter_type_t typeID` indicates the type ID,
- `int size` indicates the size of counter (32 or 64 bits),
- `dag_subfct_type_t subfct` indicates the type of sub-function,
- `int lc` indicates if there is a latch and clear bit to read the register,
- `int value_type` indicates whether it is the counter value (0) or the address where the counter value is stored (1),
- `uint64_t value` indicates the counter value,

```
typedef struct
{
  dag_counter_type_t typeID;  (see below.)
  int size;
  dag_subfct_type_t subfct;   (see below.)
  uint32_t interface_number
  int lc;
  int value_type; /* Only available for direct register */
  uint64_t value;
} dag_counter_value_t;

typedef enum
{
  kIDSubfctPort = 0x00,
  kIDSubfctStream = 0x01,
  kIDSubfctFilter = 0x02,
  kIDSubfctGeneral = 0x03,
} dag_subfct_type_t;

typedef enum
{
  kIDCounterInvalid = 0x0,
  kIDCounterRXFrame = 0x01,
  kIDCounterRXByte= 0x02,
  kIDCounterRXShort = 0x03,
  kIDCounterRXLong = 0x04,
  kIDCounterRXError = 0x05,
  kIDCounterRXFCS = 0x06,
  kIDCounterRXAbort = 0x07,
  kIDCounterTXFrame = 0x08,
  kIDCounterTXByte = 0x09,
  kIDCounterDIP4Error = 0x0A,
  kIDCounterDIP4PlError = 0x0B,
  kIDCounterBurstError = 0x0C,
  kIDCounterPlError = 0x0D,
  kIDCounterDebug = 0x0E,
  kIDCounterFilter = 0x0F,
  kIDCounterB1Error = 0x10,
  kIDCounterB2Error = 0x11,
  kIDCounterB3Error = 0x12,
  kIDCounterRXErr = 0x13,
  kIDCounterSpaceError = 0x14,
        kIDCounterContWdError = 0x15,
        kIDCounterPlContError = 0x16,
        kIDCounterTRDip4Error = 0x17,
        kIDCounterResvWd = 0x18,
        kIDCounterAddrError = 0x19,
        kIDCounterOOFPeriod = 0x1A,
        kIDCounterNbOOF = 0x1B,
        kIDCounterTXOOFPeriod = 0x1C,
        kIDCounterTXNbOOF = 0x1D,
        kIDCounterTXError = 0x1E,
        kIDCounterStatFrError = 0x1F,
        kIDCounterDip2Error = 0x20,
        kIDCounterPatternError = 0x21,
        kIDCounterRXStreamPacket = 0x22,
        kIDCounterRXStreamByte = 0x23,
        kIDCounterTXStreamPacket = 0x24,
        kIDCounterTXStreamByte = 0x25,
        kIDCounterPortDrop = 0x26,
        kIDCounterStreamDrop = 0x27,
        kIDCounterSubStreamDrop = 0x28,
        kIDCounterFilterDrop = 0x29,
      } dag_counter_type_t;
```

Types of Counters

The following is a description of each of the counter types. **dag_counter_type_t** is the name of the counter, and **typeID** is the value of the counter type enumerator.

dag_counter_type_t	typeID	Description
kIDCounterInvalid	0x0	Error code. Endace internal use only.
kIDCounterRXFrame	0x01	Number of frames received.
kIDCounterRXByte	0x02	Number of bytes received.
kIDCounterRXShort	0x03	Number of frames received which were too short.
kIDCounterRXLong	0x04	Number of frames received which were too long.
kIDCounterRXError	0x05	Number of packets received with errors (typically CRC errors).
kIDCounterRXFCS	0x06	Number of incoming packets with FCS errors.
kIDCounterRXAbort	0x07	Number of incoming packets aborted.
kIDCounterTXFrame	0x08	Number of frames transmitted.
kIDCounterTXByte	0x09	Number of bytes transmitted.
kIDCounterDIP4Error	0x0A	Endace internal use only.
kIDCounterDIP4PlError	0x0B	Endace internal use only.
kIDCounterBurstError	0x0C	Endace internal use only.
kIDCounterPlError	0x0D	Endace internal use only.
kIDCounterDebug	0x0E	Endace internal use only.
kIDCounterFilter	0x0F	Reserved value.
kIDCounterB1Error	0x10	Endace internal use only.
kIDCounterB2Error	0x11	Endace internal use only.
kIDCounterB3Error	0x12	Endace internal use only.
kIDCounterRXErr	0x13	Number of packets received with errors (typically CRC errors).
kIDCounterSpaceError	0x14	Endace internal use only.
kIDCounterContWdError	0x15	Endace internal use only.
kIDCounterPlContError	0x16	Endace internal use only.
kIDCounterTRDip4Error	0x17	Endace internal use only.
kIDCounterResvWd	0x18	Endace internal use only.
kIDCounterAddrError	0x19	Endace internal use only.
kIDCounterOOFPeriod	0x1A	Endace internal use only.
kIDCounterNbOOF	0x1B	Endace internal use only.

dag_counter_type_t	typeID	Description
`kIDCounterTXOOFPeriod`	0x1C	Endace internal use only.
`kIDCounterTXNbOOF`	0x1D	Endace internal use only.
`kIDCounterTXError`	0x1E	Endace internal use only.
`kIDCounterStatFrError`	0x1F	Endace internal use only.
`kIDCounterDip2Error`	0x20	Endace internal use only.
`kIDCounterPatternError`	0x21	Endace internal use only.
`kIDCounterRXStreamPacket`	0x22	Number of packets received per stream.
`kIDCounterRXStreamByte`	0x23	Number of bytes received per stream.
`kIDCounterTXStreamPacket`	0x24	Number of packets transmitted per stream.
`kIDCounterTXStreamByte`	0x25	Number of bytes transmitted per stream.
`kIDCounterPortDrop`	0x26	Packets dropped per port.
`kIDCounterStreamDrop`	0x27	Packets dropped per stream.
`kIDCounterSubStreamDrop`	0x28	Unused.
`kIDCounterFilterDrop`	0x29	Packets dropped by filters.
`kIDCounterIdleCell`	0x35	Unused.
`kIDCounterTxClock`	0x36	Unused.
`kIDCounterRxClock`	0x37	Unused.
`kIDCounterDuckOverflow`	0x38	Unused.
`kIDCounterPhyClockNominal`	0x39	Unused.

Blocks

The following explains the structure of a block, `dag_block_type_t`:

```
typedef enum
{
    kIDBlockDebug = 0x0,
    kIDBlockEthFramerRx = 0x11,
    kIDBlockEthFramerTx = 0x12,
    kIDBlockEthFramerRxTx = 0x13,
    kIDBlockSonetFramerRx = 0x21,
    kIDBlockSonetFramerTx = 0x22,
    kIDBlockSonetFramerRxTx = 0x23,
    kIDBlockStreamRx = 0x31,
    kIDBlockStreamTx = 0x32,
    kIDBlockStreamRxTx = 0x33,
    kIDBlockStreamDropRx = 0x41,
    kIDBlockStreamDropTx = 0x42,
    kIDBlockStreamDropRxTx = 0x43,
    kIDBlockDropRx = 0x51,
    kIDBlockDropTx = 0x52,
    kIDBlockDropRxTx = 0x53,
    kIDBlockPortDropRx = 0x61,
    kIDBlockPortDropTx = 0x62,
    kIDBlockPortDropRxTx = 0x63,
    kIDBlockFilterRx = 0x71,
    kIDBlockFilterTx = 0x72,
    kIDBlockFilterRxTx = 0x73,
    kIDBlockPatternRx = 0x81,
    kIDBlockPatternTx = 0x82,
    kIDBlockPatternRxTx = 0x83,
    kIDBlockFrontEndFrequencyReferenceRx = 0xa1,
    kIDBlockFrontEndFrequencyReferenceTx = 0xa2,
    kIDBlockFrontEndFrequencyReferenceRxTx = 0xa3
} dag_block_type_t;
```

Function Descriptions

This chapter explains the various functions which are exposed by the Configuration and Status API to read the counters.

Component Implementation

The CSI is implemented as a component called `counter_interface` in file `counter_interface_component.c` located in `/lib/libdagconf/components/`.

The usual function for a component has been implemented:

```
<component name>_get_new_component,
<component name>_post_initialize,
<component name>_reset,
<component name>_default,
<component name>_dispose,
<component name>_update_register_base.
```

Printing of Counters and Statistic Registers

In order to display the various counters and statistics, a new function has been created; `print_univ_counters in file counter_printing.c l`ocated in `tools/dagconfig/.`

Others files modified to implement the component:

```
/tools/dagconfig/process_cmdline.c
/tools/dagconfig/process_cmdline.h
/tools/dagconfig/dagconfig.c
/lib/libdagconf/cards/dagx_impl.c
```

Functions

dag_config_get_number_block function

Purpose	Return the number of block(s) of this card
Declared In	dag_config.h
Prototype	uint32_t dag_config_get_number_block(dag_card_ref_t card_ref)
Parameters	→ card_ref Reference of the DAG card
Returns	Number of block (counter statistic interface) of the DAG card.

dag_config_get_number_counters function

Purpose	Return the number of counter(s) for a particular block
Declared In	dag_config.h
Prototype	uint32_t dag_config_get_number_counters(dag_card_ref_t card_ref, dag_block_type_t block_type)
Parameters	→ card_ref Reference of the DAG card → block_type Type of csi block
Returns	Number of statistic counter(s) for the block "block type"

dag_config_get_number_all_counters function

Purpose	Return the total number of counter(s) of this card
Declared In	dag_config.h
Prototype	uint32_t dag_config_get_number_counters(dag_card_ref_t card_ref)
Parameters	→ card_ref Reference of the DAG card
Returns	Number of statistic counter(s) of the DAG card.

dag_config_get_counter_id_subfct function

Purpose	Return the id and the sub-function of counters in a specific block.
Declared In	dag_config.h
Prototype	uint32_t dag_config_get_counter_id_subfct(dag_card_ref_t card_ref, dag_block_type_t block_type, dag_counter_value_t counter_id[], uint32_t size)
Parameters	→ card_ref Reference of the DAG card → block_type Type of csi block → counter_id[] returned array of dag_counter_value_t structure. → size Size of counter_id array.
Returns	Number of statistic counter(s) found for the block "block type".

dag_config_get_all_block_id function

Purpose	Return all block ids.
Declared In	dag_config.h
Prototype	uint32_t dag_config_get_all_block_id(dag_card_ref_t card_ref, uint32_t block_id[], uint32_t size)
Parameters	→ card_ref Reference of the DAG card → block_type Type of csi block → block_id[] returned array of uint32_t. Contains all block ids. → size Size of block_id array.
Returns	Number of statistic counter(s) found for the block "block type".

dag_config_latch_clear_all function

Purpose	Latch and clear all the csi blocks.
Declared In	dag_config.h
Prototype	void dag_config_latch_clear_all(dag_card_ref_t card_ref)
Parameters	→ card_ref Reference of the DAG card
Returns	N/A
Comments	This function is called by dag_config_read_all_counters, dag_config_read_counter and print_univ_counters (counter_printing.c).

dag_config_latch_clear_block function

Purpose	Latch and clear a specific csi block.
Declared In	dag_config.h
Prototype	void dag_config_latch_clear_all(dag_card_ref_t card_ref, dag_block_type_t block_type)
Parameters	→ card_ref Reference of the DAG card → block_type Type of csi block
Returns	N/A

dag_config_read_single_block function

Purpose	Return the value of all counters in a specific csi block.
Declared In	dag_config.h
Prototype	uint32_t dag_config_read_single_block(dag_card_ref_t card_ref, dag_block_type_t block_type, dag_counter_value_t countersTab[], uint32_t size, int lc)
Parameters	→ card_ref Reference of the DAG card → block_type Type of csi block → countersTab Table of counter's structures → int size Size of countersTab → int lc Latch and clear option (0 = no latch and clear, 1 = latch and clear the block before reading the values)
Returns	Return the number of counters in the specific csi block

dag_config_read_all_counters function

Purpose	Read all counters of the card and stock their parameters in a table
Declared In	dag_config.h
Prototype	uint32_t dag_config_read_all_counters(dag_card_ref_t card_ref, dag_counter_value_t countersTab[], uint32_t size, int lc)
Parameters	→ card_ref Reference of the DAG card → countersTab Table of counter's structures → int size Size of countersTab → int lc Latch and clear option (0 = no latch and clear, 1 = latch and clear the block before reading the values)
Returns	Return the number of counters

dag_config_read_single_counter function

Purpose	Get the value of single counters on the card
Declared In	dag_config.h
Prototype	uint64_t dag_config_read_single_counter(dag_card_ref_t card_ref, dag_block_type_t block_type, dag_counter_type_t counter_type, dag_subfct_type_t subfct_type)
Parameters	→ card_ref Reference of the DAG card → block_type Type of csi block → counter_type Type ID of counter → subfct_type Type of sub-function
Returns	Return the value of a specific counter

Sample Code

The following sample code displays the counters applicable to a specified DAG card.

```
void print_univ_counters(dag_card_ref_t card)
{
    dag_counter_value_t *counters=NULL;
    uint32_t nb_count = 0;
    uint32_t nb_block = 0;
    int nb_count_s = 0;
    int i, j;
    uint32_t *block_id = NULL;
    char interface_name[10];

    nb_block = dag_config_get_number_block(card);
    nb_count = dag_config_get_number_all_counters(card);

    dag_config_latch_clear_all(card);

    if (nb_block > 0)
    {
        printf("\n**** Number of blocks: %d \nTotal number of counters: %d \n",
nb_block, nb_count);
        block_id = (uint32_t *)malloc(nb_block*sizeof(int32_t));
        for(j = 0; j < nb_block; j++)
        {
            nb_block = dag_config_get_all_block_id(card, block_id, nb_block);
            nb_count_s = dag_config_get_number_counters(card,
(dag_block_type_t)block_id[j]);

            if (nb_count > 0)
            {
                printf("Nb of counter(s) in Block ID \"%s\": %d\n",
id_block_to_string(block_id[j]), nb_count_s);
                counters =
(dag_counter_value_t*)malloc(nb_count*sizeof(dag_counter_value_t));
                nb_count_s = dag_config_read_single_block(card,
(dag_block_type_t)block_id[j], counters, nb_count_s, 0);

                /* print value of counters */
                for (i=0; i < nb_count_s; i++)
                {
                    if(counters[i].subfct == 0x00)
                        strcpy(interface_name,"Port");
                    else if(counters[i].subfct == 0x01)
                        strcpy(interface_name,"Stream");
                    else if (counters[i].subfct == 0x02)
                        strcpy(interface_name,"Filter");
                    else if (counters[i].subfct == 0x03)
                        strcpy(interface_name,"General");

                    printf(" %20s :%10s : %2d : value = %"PRIu64"\n",
id_counter_to_string(counters[i].typeID),
interface_name,counters[i].interface_number,counters[i].value);
                }
                    printf("\n");
                    free(counters);
            }
            else
                printf("No counters in block %s.\n",
id_block_to_string(block_id[j]));
```

```
        }
        free(block_id);
    }
    else
        printf("No blocks \n");
}
```

Version History

Version	Date	Reason
1	4-Jan-07	Initial revision.
2	11-July-08	Updated Subfunction Type.
2.1	18 July 2008	Updated to Endace template.
3	September 2009	Updated for software release 3.4.1. Updated front matter. Revised document order. Added Types of Counters. Added Blocks. Minor changes.
4	November 2011	Updated branding.

endace.com