DMA utils Library on EVE and DSP

User Guide

May 2017

IMPORTANT NOTICE

1 Read This First

1.1 About This Manual

This document describes how to install and work with Texas Instruments' (TI) DMA Utils Module implemented for EVE and DSP subsystems on TI's device. It also provides a detailed Application Programming Interface (API) reference and information on the sample application that accompanies this component.

1.2 Intended Audience

This document is intended for system engineers who want to integrate TI's vision and imaging algorithms with other software to build a high level vision system using EVE or DSP subsystem.

This document assumes that you are fluent in the C language, have a good working knowledge of embedded system and basic computer architecture concepts like DMA.

1.3 How to Use This Manual

This document includes the following chapters:

- **Chapter 2 - Introduction**, provides a brief introduction DMA Utils Library. It also provides an overview of supported features.
- **Chapter 3 - Installation Overview**, describes how to install, build, and run the algorithm.
- **Chapter 4 - API Reference**, describes the data structures and interface functions used in the algorithm.
- **Chapter 5 - Sample Usage**, describes the sample usage of the library.
- **Chapter 6 - Frequently Asked Questions,** provides answers to frequently asked questions related to using DMA Utils Library.

1.4 Related Documentation from Texas Instruments

The following documents describe EVE subsystem details. To obtain a copy of any of these TI documents, visit the Texas Instruments website at www.ti.com.

- Enhanced Direct Memory Access (EDMA3) Controller User's Guide (literature number SPRUEQ5) for complete details on the EDMA3

1.5 Abbreviations

The following abbreviations are used in this document.

Table 1 List of Abbreviations

Abbreviation	Description
API	Application Programming Interface

Abbreviation	Description
DMA	Direct Memory Access
EDMA	Enhanced DMA
DSP	Digital Signal Processing
EVE	Embedded Vision Engine
DDR	Double Data Rate
SRAM	Static random-access memory

1.6 Text Conventions

The following conventions are used in this document:

- Text inside back-quotes (‘‘) represents pseudo-code.
- Program source code, function and macro names, parameters, and command line commands are shown in a `mono-spaced` font.

1.7 Product Support

When contacting TI for support on this product, quote the product name (DMA Utils Module) and version number. The version number of the DMA Utils Module is included in the Title of the Release Notes that accompanies the product release.

1.8 Trademarks

Code Composer Studio, DSP/BIOS, eXpressDSP, TMS320, EVE are trademarks of Texas Instruments.

2 Introduction

This chapter provides a brief introduction to EDMA3. It also provides an overview of TI's implementation of DMA Utils library for C66x DSP/EVE and its supported features.

2.1 Overview of EDMA3

The EDMA3 is the primary DMA engine for transfers between system memory (DDR and/or L3 SRAM) and EVE/DSP internal memories. Channel controller (CC) is the front end programmer interface, to the DMA engine. It is a small processor that accepts a program from a memory region called PaRAM (parameter memory) to describe the attributes of the data transfer and efficiently calculates new addresses for regular data transfers. Transfer controller (TC) is an actual physical channel that services the requests programmed on the Channel controller. The transfer controller aspect is transparent to the user/programmer, hence user need not worry about it apart from knowing that EVE/DSP subsystem EDMA has two TC channels (in TDA2x device) to ensure maximum concurrency for data transfers along with computation. Mapping of a transfer to particular TC is programmable.

2.2 Overview of DMA Utils library

DMA utility is a software offering to ease the EDMA programming from user . Idea behind DMA utility is to provide utility functions for the most common DMA data flow usecases in order to hide EDMA programming from the user.

Most of the vision and imaging algorithms are compute intensive and hence it is always desirable to overlap compute intensive part with data transfer so as to fully utilize both the hardware resources in parallel. It has been observed that most of the vision algorithms are implemented with some common set of data transfer's . EDMA utilities provide utility functions for such common EDMA data flow's which are generally used in many vision algorithms.

Following are the most common EDMA data flow usecases used in vision algorithms:

- Auto-Increment : Horizontal
- Auto-Increment : Horizontal (with ping pong support)
- Auto-Increment : 1D (with circular buffering)
- Scatter Gather
- Blocking/Non-Blocking 2D memcpy

2.3 Supported Services and Features

This user guide accompanies TI's implementation of DMA Utils for EVE/DSP core of TI devices.

This version of the Lane Detection has the following supported features of the standard:

- ❑ Supports 2D Auto-increment usecase.

- Supports 2D Auto-increment with inbuilt buffer aliasing.
- Supports 1D Auto-increment usecase.
- Supports scatter gather usecase.
- Supports 2D memcpy usecase.

3 Installation Overview

This chapter provides a brief description on the system requirements and instructions for installing DMA Utils library.

3.1 System Requirements

This section describes the hardware and software requirements for the normal functioning of the dma utils.

3.1.1 Hardware

This module has been built and tested on EVE and C66x DSP on TDA2x platform. The library shall work on any device hosting EVE or C66x DSP.

3.1.2 Software

Please refer to section 3.3.1 for the softwares/tools required for the stand alone functioning of the dma utils module.

3.2 Installing the Component

The DMA Utils component is released as install executable. Following sub sections provided details on installation along with directory structure.

3.2.1 Installing the compressed archive

The algorithm component is released as a compressed archive. To install the algorithm, extract the contents of the exe file onto your local hard disk. The exe file extraction creates a top-level directory called REL.DMAUTILS.xx.xx.xx.xx . Folder structure of this top level directory is shown in below figure.

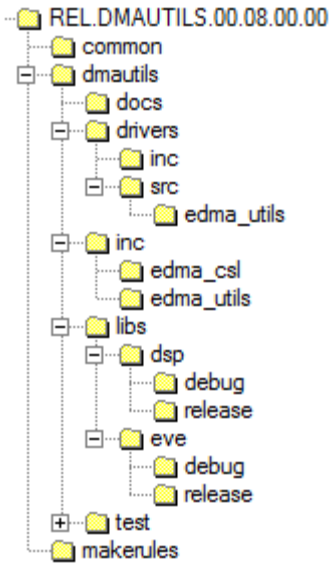

Figure 1 Component Directory Structure

Table 2 Component Directories

Sub-Directory	Description
\common	Common files for building test code for EVE and DSP
\dmautils\docs	Contains docs related to DMA Utils
\dmautils\drivers\inc	Contains private headers for DMA Utils
\dmautils\drivers\src	Contains source files for lower level DMA API's
\dmautils\drivers\src\edma_utils	Contains source files for DMA Utility API's
\dmautils\inc	Contains public header files for lower level DMA API's
\dmautils\inc\edma_csl	Contains header files for EDMA Chip support Library
\dmautils\inc\edma_utils	Contains public header files for DMA Utility API's
\dmautils\libs\dsp\debug	Contains debug library of DMA Utils for DSP
\dmautils\libs\dsp\release	Contains relese library of DMA Utils for DSP
\dmautils\libs\eve\debug	Contains debug library of DMA Utils for EVE
\dmautils\libs\eve\release	Contains release library of DMA Utils for EVE
\dmautils\test	Contains sample test code for DMA Utility.
\makerules	Makefiles for building DMA Utility.

3.3 Building DMA Utils Library

As part of DMA Utils software, pre-built libraries are available but for the users who want to modify and build, this section is relevant. The package currently supports both windows and linux build.

3.3.1 Dependent software components

The following are the software requirements to build and use the EVE software. Please refer DMAUtilsLibrary_ReleaseNotes.pdf to know the exact version of tools used for validation:

- **Development Environment:** Code Composer Studio (Code Composer Studio).
- **ARP32 Code Generation Tools**
- **C6000 Code Generation Tools**
- **GNU Make**

3.3.2 Installation of Dependent software components

3.3.2.1 Code Composer Studio (Code Composer Studio v5)

CCS can be installed using the instruction in following link :

http://processors.wiki.ti.com/index.php/Category:Code_Composer_Studio_v5

3.3.2.2 ARP32 Code Generation Tools

Installation of ARP32 tools is done by updating CCS tools using P2 server

1. In the CCSv5 main window, select Help->Install New Software...

   - This brings up the Available Software window.

2. In the Available Software window, type the EVE P2 installation server URL into the "Work with:" text box, and press Enter.

Server URL for windows:

http://software-dl.ti.com/dsps/dsps_public_sw/sdo_ccstudio/codegen/Updates/p2win32

Server URL for Linux:

http://software-dl.ti.com/dsps/dsps_public_sw/sdo_ccstudio/codegen/Updates/p2linux

The contents of the site should appear in the selection area in the middle of the window.

You should see "CCSv5 Windows updates".

3. Expand "CCSv5 Windows updates".

- You should see "EVE Compiler Tools vx.x.x".

4. Select the EVE Compiler Tools checkbox, then click "Next>".

- The Install Details window will be displayed. If the "Details" section says "Cannot complete the install because of a conflicting dependency", try the following steps (The exact wording may differ):

a. Click the "Back" button on the Install Details window. This takes you back to the Available Software window.

b. Uncheck the box marked "Group items by category" in the lower left portion of the Available Software window.

- You should now see "REQUIRED CCS Update Installation Tool" as well as "EVE Compiler Tools vx.x.x".

c. Select the checkbox for both of these items, then click "Next>".

- This will bring up the Installation Details window. The window may indicate that your original request has been modified; that is okay.

d. Continue through the installation process as described above.

5. Continue through the install process, accept the license agreement, and finish the installation.

6. Restart CCS to execute the installation.

NOTE: The installer for the EVE Compiler Tools is executed in silent mode. If the copy of the XML file called "ARP32.xml" required to activate the EVE tools in CCS fails, you will not be notified. The easiest way to tell if this occurred is to create a New CCS Project and look for EVE in the Device Family drop down menu. If the EVE tools are not shown here, please refer to the README.txt included in the release directory (<CCSInstallDir>\ccsv5\tools\compiler\ arp32_1.0.x). If the directory does not exist, then installation is not completed

User should check for updates by going to Help->Check for Updates to update to the latest version of ARP32 CG tools

3.3.2.3 C6000 Code Generation Tools

https://www-a.ti.com/downloads/sds_support/TICodegenerationTools/download.htm
After installing the CG tools, set the environment variable named "DSP_TOOLS" to the installed directory like <install directory>\<cgtools_directory>.

3.3.2.4 GNU Make

Available as part of CCS installation (ccsv5\utils\bin\gmake.exe)

3.3.3 Building DMA Utils library

Before building DMA Utils library following variables as defined in makerules/config.mk file needs to be set. Following are the defaults value used for the variables used for building DMA Utils library. If all the dependent components are installed in the same location as default then user does not need to set any

additional variable. If the dependent components location is not the same default then user needs to set the corresponding variables in makerules/config.mk file.

```
DSP_TOOLS          ?="C:\ti\ccsv5\tools\compiler\c6000_7.4.2"

ARP32_TOOLS        ?="C:\ti\ccsv5\tools\compiler\arp32_1.0.7"

UTILS_PATH         ?="C:\ti\ccsv5\utils\cygwin"
```

After setting the above variables. You can build dmautils library by going to <SW ROOT>/dmautils directory and using following commands for various DMA Utils library :

For building DMA Utils library in release mode for DSP subsystem

```
gmake CORE=dsp
```

For building DMA Utils library in debug mode for DSP subsystem

```
gmake CORE=dsp TARGET_BUILD=debug
```

For building DMA Utils library in release mode for EVE subsystem

```
gmake CORE=eve
```

For building DMA Utils library in debug mode for EVE subsystem

```
gmake CORE=eve TARGET_BUILD=debug
```

For building DMA Utils library in debug host emulation mode for DSP subsystem

```
gmake CORE=dsp TARGET_PLATFORM=PC
```

For building DMA Utils library in debug host emulation mode for EVE subsystem

```
gmake CORE=eve TARGET_PLATFORM=PC
```

Library will be created and placed at:
<DMAUTILS_PATH>/dmautils/libs/<CORE>/<TARGET_BUILD>/dmautils.lib

Host Emulation library will be created and placed at

<DMAUTILS_PATH>/dmautils/libs/PC/<TARGET_BUILD>/dmautils.lib

4 API reference

This chapter provides a detailed description of various API provided for DMA Utils library.

4.1 Lower Level EDMA API's

These are the API's to directly configure EDMA hardware. EDMA API's are divided into two parts: One for allocating the resources (present in dma_resource_allocator.h) and other for configure EDMA registers (present in dma_funcs.h).

dma_resource_allocator.h

API NAME	Description
DMA_resourceAllocator_initResources	This functions initializes resets all the allocated resources of EDMA. After this call all the EDMA resources are freshly available
DMA_resourceAllocator_allocateResources	This functions allocates EDMA resources for given number of channels. It also needs to know the EDMA attribute (EDMA/QDMA channel). How many param sets needed for each channel and which hardware que each channel should go to.
DMA_resourceAllocator_deallocateResources	This functions deallocates EDMA resources which are allocated using DMA_resourceAllocator_allocateResources API

dma_functions.h

API NAME	Description
DMA_funcs_hardwareRegSetup	This function updates the actual hardware register based on DMA_resourceStruct. For QDMA we will be setting up QCHMAP, QDMAQNUM, QEESR. Similarly for EDMA we will be setting up, DCHMAP, DMAQNUM. Independent of QDMA and EDMA we configure QUEWTHRA, QUEPRI and QUETCMAP registers. This implementation assumes one to one mapping between transfer controllers and hardware queues. TC0 is mapped to Q0 and TC1 is mapped to Q1, this means Q0 submits on TC0 and Q1 submits on TC1. More over Q0 is given a higher priority compare to Q1.
DMA_funcs_writeTransferParams	Updates paramset entry based on user input
QDMA_SUBMIT	This function is used to trigger QDMA transfer. It's user of this functions

	responsibility to provide the address of the trigger word writing at which will trigger the QDMA transfers. This function is only aplicable for QDMA channels and should not be used for EDMA channels
QDMA_WAIT	This function waits for the completion of QEDMA transfers indicated by bit position in wait word. It also clears the IPR register by writing to 1 to the bits position indicated by wait word
DMA_SUBMIT	Submit the DMA on this virtual handle
DMA_WAIT	Wait for the DMA to complete, by polling
pack2	Helper function to pack two 16-bit values into a single 32 bit entry

All EDMA example uses edma_csl API's which are present in edma_csl folder in dmautils/inc directory.

Paths:

Sources – dma_funcs.c in dmautils/drivers/src

dma_resource_allocator.c in dmautils/drivers/src/

Prototypes – dma_funcs.h in dmautils/inc

dma_resource_allocator.h in dmautils/inc

dma_resource.h in dmautils/inc

4.2 DMA Utils API's

Following are the set of API's which are expected to be provided for each utility:

- EDMA_UTILS_setEdma3RmHandle : This API should be called first for a set of Utility functions used inside an applet. Users can provide handle as NULL if they don't want to use EDMA3 LLD
- EDMA_UTILS_globalReset : This API should be called once per frame. This will reset all the utilities state. It is important that this should only be called once for IN or OUT transfer.
- EDMA_UTILITY_<usecase>_getContextSize: To request the utility to give the size of internal context used by it. It is always advisable to allocate the utility context in internal memory for better performance
- EDMA_UTILITY_<usecase>_init : To initialize the usecase internal context with user provided transfer property for a specific usecase
- EDMA_UTILITY_<usecase>_update: Update transfer property of usecase

- EDMA_UTILITY_<usecase>_configure : Actually configure EDMA hardware based on usecase requirement
- EDMA_UTILITY_<usecase>_deconfigure : Release the EDMA resources allocated during configure call of a usecase
- EDMA_UTILITY__<usecase>_trigger : Trigger the DMA transfer for particular usecase
- EDMA_UTILITY_<usecase>_wait : Wait for transfer to complete

There can be some additional API's for separating the INPUT and OUTPUT transfers for trigger and wait. The reason for keeping them separate is to remove overheads of checking whether its INPUT transfer or OUTPUT transfer

Sources – EDMA_UTILS_<utilityname>.c in dmautils/drivers/src/edma_utils
Prototypes – EDMA_UTILS_<utilityname>.h in dmautils/inc/edma_utils

4.2.1 EDMA Utility Autoincrement

Auto-increment Data flow means that every time user triggers a DMA transfer it should fetch a block without user/CPU intervention. All the addresses updating is hidden from the user and is either done by EDMA hardware itself or done by utility for the cases where it cannot be done by hardware. EDMA autoincrement utility provides init/configure function which will configure hardware once and later user is expected to only call EDMA trigger and EDMA wait. Figure 3-1displays basic horizontal auto-increment. Note that blocks need not be overlapping as it is shown in the figure and also the internal memory organization is not necessary to be same.

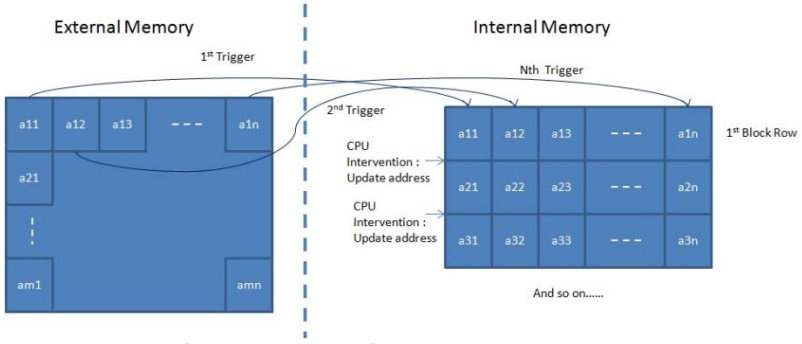

Figure 3-1. Autoincrement Usecase

Auto-increment can also be of two types:

- Horizontal auto-increment: In this usecase, blocks are incremented in horizontal direction first followed by vertical direction.
- Vertical auto-increment: In this usecase, blocks are first incremented in vertical direction then in horizontal direction.

This usecase is most common data flow, which is needed by vision algorithms. For example any filtering operation generally works on blocks which are incremented either in horizontal or vertical direction. In the subsequent section, we will be describing various autoincrement utilities that are offered by dma utils.

This utility handles both overlapping and non-overlapping increments. Lets look at the interface for this utility.

Following is the data type for configuring EDMA_AUTO_INCREMENT utility

```
typedef struct
{
    uint16_t    roiWidth;
    uint16_t    roiHeight;
    uint16_t    blkWidth;
    uint16_t    blkHeight;
    uint16_t    extBlkIncrementX;
    uint16_t    extBlkIncrementY;
    uint16_t    intBlkIncrementX;
    uint16_t    intBlkIncrementY;
    uint32_t    roiOffset;
    uint8_t     *extMemPtr;
    uint8_t     *interMemPtr;
    uint16_t    extMemPtrStride;
    uint16_t    interMemPtrStride;
    uint8_t     dmaQueNo;
}EDMA_UTILS_autoIncrement_transferProperties;
```

This structure specifies the properties of the transfer for auto increment usecase.

roiWidth : Width of the region of interest in an image frame. User should be careful that for non-overlapping increment usecase roiWidth should not include the last increment in X direction in order to satisfy the above stated equation.

roiHeight : Height of the region of interest in an image frame. User should be careful that for non-overlapping increment usecase roiHeight should not include the last increment in Y direction.

blkWidth : Block Width

blkHeight : Block Height

extBlkIncrementX : Block Increment in X direction for external memory. If extBlkIncrementX = 0 then number of blocks for which autoincrement with run in horizontal directions is 1.

extBlkIncrementY: Block Increment in Y direction for external memory.If extBlkIncrementY = 0 then number of blocks for which autoincrement with run in vertical directions is 1.

intBlkIncrementX : Block Increment in X direction for internal memory

intBlkIncrementY : Block Increment in Y direction for internal memory

roiOffset : Offset from the base pointer of the image to the point from where ROI starts

extMemPtr : Pointer to external memory buffer

interMemPtr :Pointer to internal memory buffer. When using autoincrement utility from BAM user need not provide this pointer as it is internally allocated by BAM and automatically gets set during setMemRec function

extMemPtrStride : Stride/Pitch for external memory buffers

interMemPtrStride: Stride/Pitch for internal memory buffers

dmaQueNo: Queue/TC number to be used (0 or 1)

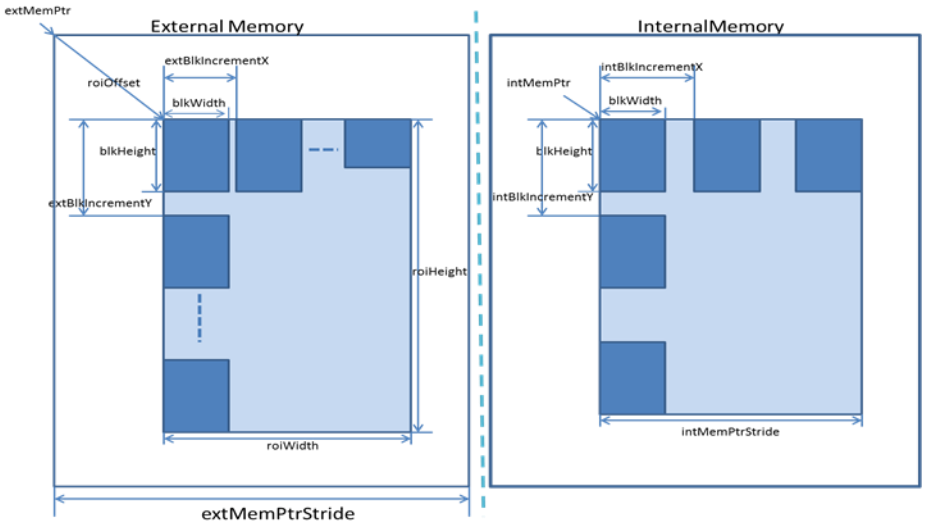

Figure 3-1. Autoincrement Utility Init parameters

API NAME	Description
EDMA_UTILS_autoIncrement_getContextSize	Returns the size needed by the edma utils autoincrement internal context structure. If you want to statically allocate the memory for this context you can also get the size of context from the edma_utils_context_size.h header file located at inc/edma_utils folder.
EDMA_UTILS_autoIncrement_init	Initializes the EDMA autoincrement context based on the user provided initialization parameters
EDMA_UTILS_autoIncrement_configure	This API actually configures EDMA with autoincrement init params provided during init time. This API can be called separately for IN and OUT transfers or for both INOUT transfer. It is expected that you configure IN transfers followed by OUT transfers.
EDMA_UTILS_autoIncrement_deconfigure	This API release all the EDMA resources which were requested duing EDMA_UTILS_autoIncrement_configure call. This API can be called separately for IN and OUT transfers or for both INOUT transfer. It is expected that you deconfigure IN transfers followed by OUT transfers.
EDMA_UTILS_autoIncrement_update	This API is an optional API which can be called to update the external memory

	address.
EDMA_UTILS_autoIncrement_triggerInChannel	This API triggers all the input transfers
EDMA_UTILS_autoIncrement_triggerOutChannel	This API triggers all the output transfers. This API also returns the status of 1 when the second last block is reached in autoincrement. This condition should be used to come out of while loop. First time call to this function will not result any dma transfer and is more like a dummy call.
EDMA_UTILS_autoIncrement_waitInChannel	This API waits for all the input transfers to complete. This is a blocking call
EDMA_UTILS_autoIncrement_waitOutChannel	This API waits for all the output transfers to complete. This is a blocking call

Following is the expected order in which the following API's are expected :

- First call either call EDMA_UTILS_autoIncrement_getContextSize to get the context size or read it from *edma_utils_context_size.h* (incase user wants to statically allocate this memory) . User is then expected to use this to allocate memory for autoincrement context. For all further API's this context should be given as an input argument

  Note: It is advisable to allocate the auto increment context size in EVE internal memory(DMEM) for performance reasons. This is because this context is read multiple types for every block in the autoincrement loop and hence keeping it in internal memory will have better performance.

- Once context is allocated you are ready to initialize auto increment utility with the transfers you want to configure. This can be done using EDMA_UTILS_autoIncrement_init API. Note that this API can be called separately for IN and OUT transfers or together for INOUT transfer. It is expected that IN transfers are configured first followed by OUT transfers when we are initializing IN and OUT channels separately

- There is an optional API provided to update DDR/external memeory pointer for the cases when you don't have these addresses available during init time. The API to update external memory pointers is EDMA_UTILS_autoIncrement_update

- Once you have initialized the autoincrement context with the transfers you need then you can use this API to configure autoincrement which internally programs EDMA registers with the user provided configuration

- After the above steps we are ready to start autoincrement. Use trigger and wait API's to trigger and wait for all transfers.

- EDMA autoincrement uses EDMA_UTILS_autoIncrement_triggerOutChannel to communicate the arrival of second last block. It returns 1 when it reaches the second last block. This condition should be used as an exit condition to come out of the autoincrement loop.

4.2.2 EDMA Utility Autoincrement V2

This utility is exactly same as EDMA Utility Autoincrement except for the case that this utility can handle ping pong of 2D buffers for the cases where hardware doesn't support aliasing of IO buffers (for example DSP subsystem).

Please refer the EDMA Uitlity Autoincrement section 4.2.1 for the usage of the varius API's for this utility. All the API's of edma utils autoincrement are suffixed by _v2 for this utility. Init params of V2 API is slightly different from the edma utils autoincrement. Following is the init param for V2 API :

```
typedef struct
{
    EDMA_UTILS_autoIncrement_initParam initParams;
    uint32_t   pingPongOffset;
}EDMA_UTILS_autoIncrement_initParam_v2;
```

Here initParams is same as initParams of autoincrement utility.

pingPongOffset is the offset in bytes between the ping and the pong buffer.

4.2.3 EDMA Utility Autoincrement 1D

This EDMA utility specifically handles 1D auto-increment usecase. Another feature in which this utility is different from 2D auto-increment is the fact that it supports circular buffering. Most of the interface and usage for this utility is same as EDMA_UTILITY_AUTOINCREMENT. Kindly refer to that for its usage.

Following is the data type for configuring EDMA_AUTO_INCREMENT utility

```
/** ============================================================
 *  @name   EDMA_UTILS_autoIncrement1D_transferProperties
 *
 *  @desc   This structure specifies the properties of the transfer for
 *          auto increment usecase.
 *
 *  @field totalLength
 *             Total Length of the 1D data in which auto increment is expected
 *             to rum
 *
 *  @field t numBytes
 *             Number of bytes of 1D data to be transferred per DMA trigger
 *
 *  @field extMemIncrement
 *              Increment in external memory. If extMemIncrement = 0
 *              then number of segments to be transfered is 1
 *
 *
 *  @field intMemIncrement
 *              Increment in internal memory.
 *
 *  @field extMemPtr
 *              Pointer to external memory buffer
 *
 *  @field interMemPtr
 *              Pointer to internal memory buffer.
 *              When using this autoincrement utility from BAM user need not provide
 *              this pointer as it is internally allocated by BAM and automatically
 *              gets set during setMemRec function
 *
 *  @field numCircBuf
 *              Number of buffers for circular buffering
 *
 *  @field dmaQueNo
 *              DMA Que number to which particular transfer is expected to go
 *
 *  ============================================================
 */
typedef struct
{
    uint32_t    totalLength;
    uint16_t    numBytes;
    uint16_t    extMemIncrement;
    uint16_t    intMemIncrement;
    uint8_t     *extMemPtr;
    uint8_t     *interMemPtr;
    uint8_t     numCircBuf;
    uint8_t     dmaQueNo;
}EDMA_UTILS_autoIncrement1D_transferProperties;
```

This utility also expects the following equation to hold true :

(totalLength – numBytes) % extMemIncrement = 0

API NAME	**Description**
EDMA_UTILS_autoIncrement1D_getContextSize	Returns the size needed by the edma utils autoIncrement1D internal context structure. If you want to statically allocate the memory for this context you can also get the size of context from the edma_utils_context_size.h header file located at inc/edma_utils folder.
EDMA_UTILS_ autoIncrement1D _init	Initializes the EDMA autoIncrement1D context based on the user provided initialization parameters
EDMA_UTILS_ autoIncrement1D _configure	This API actually configures EDMA with autoIncrement1D init params provided during init time. This API can be called separately for for IN and OUT transfers or for both INOUT transfer. It is expected that you configure IN transfers followed by OUT transfers.
EDMA_UTILS_ autoIncrement1D _deconfigure	This API release all the EDMA resources which were requested duing

	EDMA_UTILS_autoIncrement1D_configure call.. This API can be called separately for for IN and OUT transfers or for both INOUT transfer. It is expected that you deconfigure IN transfers followed by OUT transfers.
EDMA_UTILS_ autoIncrement1D _update	This API is an optional API which can be called to update the external memory address.
EDMA_UTILS_ autoIncrement1D _triggerInChannel	This API triggers all the input transfers
EDMA_UTILS_ autoIncrement1D _triggerOutChannel	This API triggers all the output transfers. This API also returns the status of 1 when the second last block is reached in autoincrement. This condition should be used to come out of while loop. First time call to this function will not result any dma transfer and is more like a dummy call.
EDMA_UTILS_ autoIncrement1D _waitInChannel	This API waits for all the input transfers to complete. This is a blocking call
EDMA_UTILS_ autoIncrement1D _waitOutChannel	This API waits for all the output transfers to complete. This is a blocking call

This utility can be used to do ping/pong or circular buffering even with out using ALIAS view of EVE memory.

4.2.4 EDMA Utility Memcpy 2D

This is another EDMA utility provided by DMA Utils. This is actually generic 2D memcpy. This utility is different from all other utility in a sense that which is a stand alone utility and it doesn't need any initialization and can be called from anywhere. This utility internally uses dedicated EDMA resources. It is to be noted that this utility is a blocking call. Following is the description of various parameters which user needs to be provide for this utility.

```
/* ================================================
 *  @func     EDMA_UTILS_memcpy2D
 *
 *  @desc     This function used EDMA module of eve subsytem to do a 2D memcpy
 *
 *  @modif
 *
 *  @inputs   This function takes following Inputs
 *                    dstPtr :
 *                            Pointer to destination
 *                    srcPtr :
 *                            Pointer to source
 *                    width :
 *                            width of 2D block to be transfered
 *                    height :
 *                            height of 2D block to be transfered
 *                    dstStride :
 *                            Stride/Pitch for dstPtr
 *                    srcStride :
 *                            Stride/Pitch for srcPtr
 *
 *  @outputs NONE
 *
 *  @return    0 : Success
 *                    -1 : Failure
 *
 *  ================================================
 */

int32_t EDMA_UTILS_memcpy2D(void * dstPtr, void * srcPtr , uint32_t width, uint32_t height,
                            int32_t dstStride, int32_t srcStride);
```

4.2.5 EDMA Utility Scatter Gather

Another popular usecase which is commonly used in vision algorithms is scatter gather usecase. In Scatter gather data flow, input blocks for processing are scattered across the whole image without any periodicity. It is expected that every trigger will fetch these scattered block to a particular memory. It is to be noted that this utility need not expect that all blocks are gathered at one location only. Instead it is flexible in a sense that even in destination pointers for each block could be scattered across the image.

Following is the data type for describing a single transfer for configuring EDMA_SCATTERGATHER utility. Each field in this structure is a list of entries describing transfer property for the complete list of transfer that needs to be done in one trigger

```
/** ============================================================
 *  @name   EDMA_UTILS_scattherGather_transferProperties
 *
 *  @desc   This structure specifies the properties of the transfer for
 *          scatter gather usecase.
 *
 *  @field updateMask
 *              Mask of fields telling which fields needs to be updated.
 *              Refer to EDMA_UTILS_SCATTERGATHER_UPDATE_TYPE for valid
 *              values. User can provided more than one field to be updated
 *              by ORing the above enum. This field is a don't care(ignored) during
 *              EDMA_UTILS_scattherGather_init
 *
 *  @field dmaQueNo
 *              DMA Que number to which particular transfer is expected to go.
 *              This is dont care (ignored) during EDMA_UTILS_scattherGather_updateNtrigger
 *
 *  @field srcPtr
 *             Pointer to the list of source pointer
 *
 *  @field t dstPtr
 *            Pointer to the list of destination pointer
 *
 *  @field srcPtrStride
 *              Pointer to the list of stride for the source pointer
 *
 *  @field dstPtrStride
 *              Pointer to the list of stride for the destination pointer
 *
 *  @field blkWidth
 *              Pointer to the list of Block width for the transfer
 *
 *  @field blkHeight
 *              Pointer to the list of  Block Height for the transrer
 *
 *  ============================================================
 */
typedef struct
{
    uint8_t     updateMask;
    uint8_t     dmaQueNo;
    uint8_t     **srcPtr;
    uint8_t     **dstPtr;
    uint16_t    *srcPtrStride;
    uint16_t    *dstPtrStride;
    uint16_t    *blkWidth;
    uint16_t    *blkHeight;
}EDMA_UTILS_scattherGather_transferProperties;
```

API NAME	Description

EDMA_UTILS_scattherGather_getContextSize	Returns the size needed by the edma utils scattergather internal context structure. If you want to statically allocate the memory for this context you can also get the size of context from the edma_utils_context_size.h header file located at inc/edma_utils folder.
EDMA_UTILS_scattherGather_init	This function configures EDMA hardware based initParams. It is important to note that this particular utility configures the hardware during this call instead of configure call
EDMA_UTILS_scattherGather_deinit	This function releases all the allocated EDMA resources for this particular usecase.
EDMA_UTILS_scattherGather_configure	Dummy function
EDMA_UTILS_scattherGather_updateNtrigger	This function can selectively update four properties of the transfers and actually trigger the transfer. These prioperties are are sourceAddress, Destination Address, block widht and block Height. Which property needs to be updated can be given in EDMA_UTILS_scattherGather_updateParams. Refer this structure for futher details on each individual fields. This function can only be called after EDMA_UTILS_scattherGather_init has been called. Size of array update param should be same as what has already been initialized in EDMA_UTILS_scattherGather_init.
EDMA_UTILS_scattherGather_updateSrcNtrigger	This function updates the source pointer for all the transfers and trigger the transfer. This function should be used when only source pointer is getting updated after initializing other parameters. This function can only be called after EDMA_UTILS_scattherGather_init has been called. Size of array update param should be same as what has already been initialized in EDMA_UTILS_scattherGather_init
EDMA_UTILS_scattherGather_updateDstNtrigger	This function updates the destination pointer for all the transfers and trigger the transfer. This function should be used when only destination pointer is getting updated after initializing other parameters. This function can only be called after EDMA_UTILS_scattherGather_init has been called. Size of array update param should be same as what has already been initialized in EDMA_UTILS_scattherGather_init
EDMA_UTILS_scattherGather_wait	This function waits for the DMA transfer to be completed and can be used for any of the above trigger.

Following is the expected order in which the following API's are expected :

- First call either call EDMA_UTILS_scatterGather_getContextSize to get the context size or read it from *edma_utils_context_size.h* (incase user wants to statically allocate this memory) . User is then expected to use this to allocate memory for this utility. For all further API's this context should be given as an input argument

  Note: It is advisable to allocate the scatterGather context size in EVE internal memory(DMEM) for performance reasons. This is because this context is read multiple types for every block in the autoincrement loop and hence keeping it in internal memory will have better performance.

- Once context is allocated you are ready to initialize scatter gather utility with the transfers you want to configure. This can be done using EDMA_UTILS_scatterGather_init API.This API actually configures EDMA hardware based on init parameters

- After initialization it will depend on the usecase to decide on which trigger API to use. Any of the following trigger API can be used :

  EDMA_UTILS_scattherGather_updateNtrigger

  EDMA_UTILS_scattherGather_updateSrcNtrigger

  EDMA_UTILS_scattherGather_updateDstNtrigger

- To wait for the completion of transfer use EDMA_UTILS_scattherGather_wait API.

5 Sample Usage

This chapter provides a detailed description of the sample test examples that accompanies this dmautls library. All dmauils test examples are locate at dmautils/test folder.

5.1 edma_utils_autoincrement_test :

This is a simple example to demonstrate usage of DMA utilities for autoincrement (V2 version) on DSP. The test application uses autoincrement V2 utility to do ping pong buffering in internal memory of DSP. Each trigger fetches one 2D block from input data buffer and performs a flip operation and store the output back to DDR again using autoincrement V2 utility.

This test applicaton also uses makerules.mk file for building the example. For build the example in release mode use the following command from dmautils/test/ edma_utils_autoincrement_test directory :

gmake

To build the example in debug mode use the following command :

gmake TARGET_BUILD=debug

To build the example in host emulation mode you will need to install the c6xsim which is available at http://processors.wiki.ti.com/index.php/Run_Intrinsics_Code_Anywhere . Set C6XSIM_PATH to point to the relative location of c6xsim host emulation internsic. Use the following command to build in host emulation :

gmake TARGET_PLATFORM=PC

The generated executable will be generated at the following location :

dmautils/test/ edma_utils_autoincrement_test/elf_out

5.2 edma_utils_autoincrement_1d_test :

This is a simple example to demonstrate usage of DMA utilities for autoincrement 1d on DSP. The test application uses autoincrement 1d utility to do ping pong buffering in internal memory of DSP. Each trigger fetches one 1D block from input data buffer and performs a flip operation and store the output back to DDR again using the same utility.

This test applicaton also uses makerules.mk file for building the example. For build the example in release mode use the following command from dmautils/test/ edma_utils_autoincrement_1d_test directory :

gmake

To build the example in debug mode use the following command :

gmake TARGET_BUILD=debug

To build the example in host emulation mode you will need to install the c6xsim which is available at http://processors.wiki.ti.com/index.php/Run_Intrinsics_Code_Anywhere . Set

C6XSIM_PATH to point to the relative location of c6xsim host emulation internsic. Use the following command to build in host emulation :

gmake TARGET_PLATFORM=PC

The generated executable will be generated at the following location :

dmautils/test/ edma_utils_autoincrement_1d_test/elf_out

6 Frequently Asked Questions

This chapter provides answers to few frequently asked questions related to using this dma utils library.

6.1 Release Package

Question	Answer
Can this dmautils library release be used on any eve based platform?	Yes, you can use it. But there are some examples involving multiple EVE's and hence can only be validated on vayu platform
Where are the host(PC) API's in this release	Host API's are same as the target API's for dma utils. You just have to link using correct library from the libs folder (Libs folder contain separate libraries for PC hosts). It is to be noted that not all dmautils API's are supported from host. For the list of API's supported on PC kindly refer to section 4.2

6.2 Code Build and Execution

Question	Answer
Build is not working and not giving any error	Ensure that you have set proper environment variables as described in section 3.3.3.
dmautils build is not working on windows, giving error "gmake: *** No rule to make target"	Ensure that you have set proper environment variables as described in section 3.3.3. Make sure that your path is not too long. If this path is too long in windows, build can fail. Other way to fix this problem is to create a link to this path. For this you can use subst utility in windows. This will create a new drive with the content of you directory. subst <DRIVE NAME> PATH e.g. subst X: $(PATH_MODULE)
dmautils build not working on windows: Makefile:9: D:workdmautils/dmautils/drivers/edma_utils_autoincrement.c: No such file or directory	Make sure in your environment variable all paths use forward slash "/" instead of backward slash "\". Also do not use any quotes or spaces in the path names. If there are some spaces in path then usethe following way, if we want to set ARP32 compiler path as C:/Program Files/Texas Instruments/ARP32_tools, use set ARP32_TOOLS=C:/PROGRA~1/TEXASI~1/ARP32_tools

6.3 Issues with tools

Question	Answer
What tools are required for using this dma utils release?	Kindly see section 3.3.1 for the list of tools required.