AN12119

Examples of optimizing the DMA Linked transfer

Rev. 1.0 — 27 August 2018 Application note

Document information

Info	Content
Keywords	LPC5460x, LPC5401x, linked transfer, SDRAM, DMA, memory-to-memory
Abstract	This application note covers application detail on optimizing memory-to-memory DMA linked transfers.

Revision history

Rev	Date	Description
1.0	20180827	Initial version

Contact information

For more information, please visit: http://www.nxp.com

1. Introduction

The LPC5460x is a family of ARM Cortex-M4 based microcontrollers for embedded applications. LPCXpresso development board for LPC5460x MCUs is used in this application note. For details of the board, see:

http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/lpc-cortex-m-mcus/lpc54000-series-cortex-m4-mcus/lpcxpresso-development-board-for-lpc5460x-mcus:OM13092

Fig 1. LPC54608 LPCXpresso development board.

This application note gives an overview of the example developed using optimized and non-optimized method to construct linked descriptors.

2. Description

Direct Memory Access (DMA) is a technique for the transfer of data between a memory and a peripheral device, or from memory to memory, or between peripherals without the need of CPU interaction. Transfer speed of the DMA controller is limited with the simultaneous access to the internal buses which could be shared between other masters or slaves.

This application note provides a brief introduction to the LPC546xx's memory-to-memory DMA transfer throughout using enhanced or non-enhanced method to build the linked descriptors. This note will describe how linked descriptors are constructed, and its effectiveness how the linked descriptors influence the overall throughout of the DMA.

2.1 DMA

Data transfer requires a lot of CPU time to perform instruction execution. When using the DMA, the CPU is freed up to execute other instructions.

The DMA controller in the LPC546xx supports memory-to-memory, memory-to-peripheral, peripheral-to-peripheral, and peripheral-to-memory transfers.

2.1.1 Linked transfers

Using the linked-lists method as shown in it enables the data transfers to/from contiguous (gather) or non-contiguous (scatter) areas of memory.

Channel Descriptor	Descriptor 1	Descriptor n-1	Descriptor n
+ 0x0 (not used)	+ 0x0 Buffer 1 transfer configuration	+ 0x0 Buffer n-1 transfer configuration	+ 0x0 Buffer n transfer configuration
+ 0x4 Source data end address	+ 0x4 Source data end address	+ 0x4 Source data end address	+ 0x4 Source data end address
+ 0x8 Destination data end address	+ 0x8 Destination data end address	+ 0x8 Destination data end address	+ 0x8 Destination data end address
+ 0xC Address of descriptor 1	+ 0xC Address of descriptor n-1	+ 0xC Address of descriptor n	+ 0xC 0

Fig 2. Linked descriptors

The 30 DMA channels have an entry for the channel descriptor in the SRAM table as shown in Fig 3 (left), and each channel must be 16-byte boundary. A channel descriptor in the SRAM is linked to number of chained descriptors if two or more descriptors are required, see Fig 3 (right). The linked transfer descriptors can be located either at internal SRAM memory or at external memory such as DRAM. A detailed usage of the use of linked transfers is described in the following section.

Depending on user configuration, an interrupt can be triggered at each descriptor when the descriptor is exhausted, or at the end of linked transfer descriptor when all descriptors are exhausted. DMA transfer time can be decreased by placing the interrupt at the end of transfer configuration descriptors. It reduces the CPU time to serve the

interrupt handler when all descriptors are exhausted. This type of method is suitable for memory-to-memory transfer.

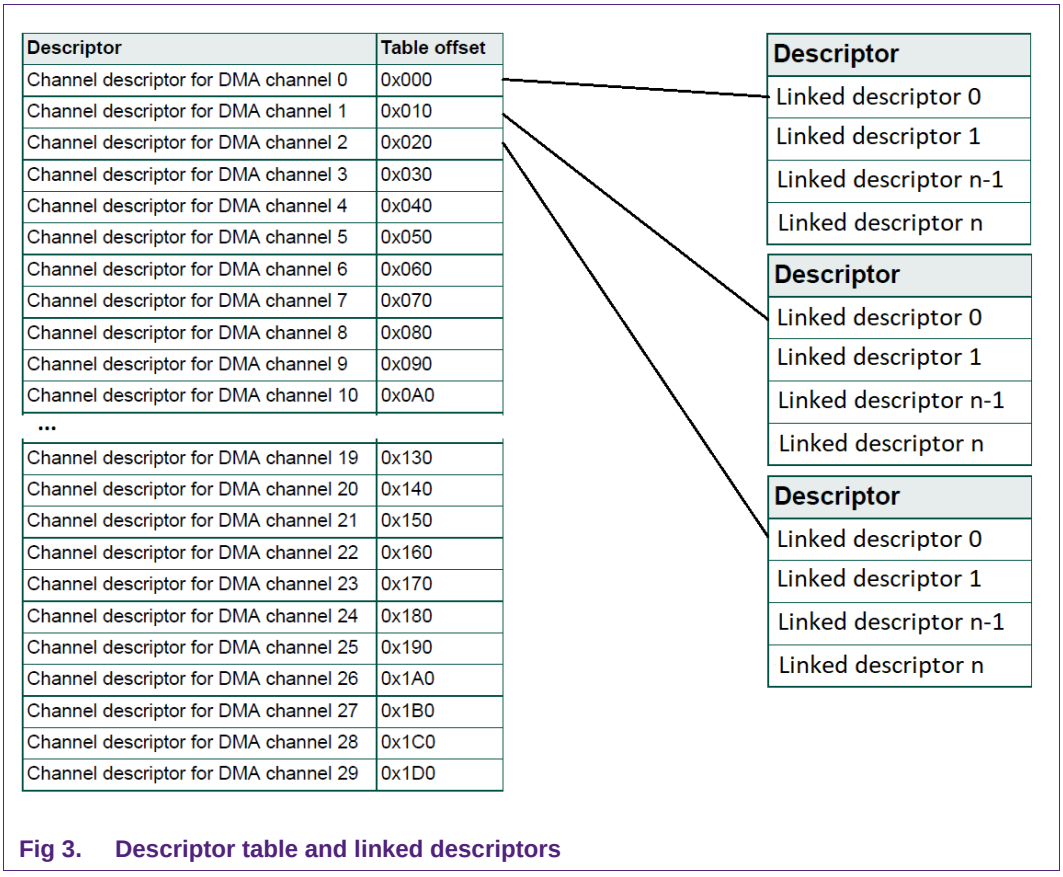

Fig 3. Descriptor table and linked descriptors

3. Application example using memory-to-memory linked transfer

The software example provided with the application note uses SRAM to allocate channel descriptor table and linked descriptors. A 2 MB source data and a 2 MB destination data will reside in the on-board SDRAM. The maximum data size that can be allowed to transfer per descriptor is 4096 bytes. Therefore, 2 MB of data requires 512 descriptors. Interrupt trigger will be placed at the 512th descriptor.

3.1 Channel transfer configuration register (XFERCFG)

Each descriptor will have the configuration valid flag (CFGVALID) and reload (RELOAD) field set, the CFGVALID indicates the descriptor is ready to be used, and the RELOAD indicates the channel control structure will be reloaded when the current descriptor is exhausted. The set interrupt flag (SETINTA or SETINTB) is set at the last descriptor. Data width is set to use 32-bit. Source (SRCINC) and destination (DSTINC) increment are set to a multiplier of 1 over data width. The total number of transfer (XFERCOUNT) to be performed is 0x3FF (2^{10}-1) where there are only 10-bit in the XFERCOUNT field.

3.2 Optimized and non-optimized method to build linked descriptors

The optimized or non-optimized method described here is not referred to the code optimization but the efficient or non-efficient way to construct linked descriptors for memory-to-memory transfer.

The overall DMA performance can be affected by how efficiently linked descriptors are constructed. Often time too much software overhead can increase the time it needs to construct the descriptors. When DMA throughout is the primary objective, linked descriptors must be optimized using small software overhead as much as possible.

3.2.1 Non-optimized method

This method is often suitable for peripheral-to-peripheral or peripheral-to-memory type of transfer. It usually takes more time to construct linked descriptors due to software overhead. For a single descriptor, the additional time added by software overhead will not be noticeable when performing peripheral-to-peripheral or peripheral-to-memory.

Using *APP_DMALinkedDescriptors()* function, see LPC54018 DMA Linked Transfer from Memory to Memory application note and SDK package, the function call *DMA_PrepareTransfer()* to setup configuration information but has not yet updated the linked descriptor until the *DMA_CreateDescriptor()* function is called. It adds extra time to setup a linked descriptor.

```
void APP_DMALinkedDescriptors(void *src, void *dest, unsigned long size)
{
    ...

    /* Setup the linked descriptors */
    for(i = 0; i < num_desc; i++)
    {
            len = (size < TRANSFER_BLOCK_SZ) ? size : TRANSFER_BLOCK_SZ;

            if(i == 0)
            {
                    /* If this is only descriptor in the chain */
                    if(i == num_desc - 1)
                    {
                            DMA_PrepareTransfer(&transferConfig, srcaddr, destaddr, BYTE_WIDTH, len, kDMA_MemoryToMemory,
                    NULL);
                    }
                    else /* Prepare and submit first descriptor */
                    {
                                    DMA_PrepareTransfer(&transferConfig, srcaddr, destaddr, BYTE_WIDTH, len,
                                    kDMA_MemoryToMemory,
                    &g_linked_desc[1]);
                    }

                    DMA_SubmitTransfer(&g_DMA_Handle, &transferConfig);
            }
            else if(i == num_desc -1)
            {
                      /* Create last descriptor in chain */
                            DMA_PrepareTransfer(&transferConfig, srcaddr, destaddr, BYTE_WIDTH, len, kDMA_MemoryToMemory,
                    NULL);
                                            DMA_CreateDescriptor(&g_linked_desc[i], &transferConfig.xfercfg, srcaddr,
                                            destaddr,
                    NULL);
            }
            else
            {
                    /* Create the descriptors in chain */
                    DMA_PrepareTransfer(&transferConfig, srcaddr, destaddr, BYTE_WIDTH, len, kDMA_MemoryToMemory,
                    &g_linked_desc[i+1]);
                    DMA_CreateDescriptor(&g_linked_desc[i], &transferConfig.xfercfg, srcaddr, destaddr,
                    &g_linked_desc[i+1]);
            }

            /* Update source address, destination address and size */
            srcaddr = (void*)((uint32_t)srcaddr + len);
            destaddr = (void*)((uint32_t)destaddr + len);
            transferConfig.srcAddr = (uint8_t*)(srcaddr);
            transferConfig.dstAddr = (uint8_t*)(destaddr);
            size = size - len;

    }

}
```

Fig 4. Non-optimized linked descriptors

3.2.2 Optimized method

Linked descriptors for memory-to-memory transfer can efficiently be constructed by reducing the software overhead into smaller single function. This method is suitable for large transfer and two or more linked descriptors.

The descriptor table and linked descriptors are all handled in a single function with no nested function calls. It results in lesser software overhead and lesser execution time.

```
void optimized_DMALinkedDescriptors (void *dst, void *src, unsigned long size)
{
    ...
    for (i = 0; i < num_desc; i++)
    {
        xfer_reg_tmp &= ~DMA_CHANNEL_XFERCFG_XFERCOUNT_MASK;
        len = (size < TRANSFER_BLOCK_SZ ? size : TRANSFER_BLOCK_SZ);

        xfer_sz = len / BYTEWIDTH;
        xfer_reg_tmp |= DMA_CHANNEL_XFERCFG_XFERCOUNT(xfer_sz - 1);

        tmp = (SRC_DST_INC * BYTEWIDTH * xfer_sz) - 1;
        srcaddr = ((uint32_t)src + tmp);
        dstaddr = ((uint32_t)dst + tmp);

        if (i == 0)
        {
            // desc table
            s_dma_descriptor_table[channel].srcEndAddr = (void*)((uint32_t)srcaddr);
            s_dma_descriptor_table[channel].dstEndAddr = (void*)((uint32_t)dstaddr);
            s_dma_descriptor_table[channel].xfercfg = 0;
            s_dma_descriptor_table[channel].linkToNextDesc = 0;

            base->CHANNEL[channel].XFERCFG = xfer_reg_tmp | DMA_CHANNEL_XFERCFG_CFGVALID_MASK;

            if (i < (num_desc - 1)) {
                s_dma_descriptor_table[channel].linkToNextDesc = (pdma_desc + 1);
                base->CHANNEL[channel].XFERCFG |= DMA_CHANNEL_XFERCFG_RELOAD_MASK;

                pdma_desc++;
                size -= len;

                src = (void*)((uint32_t)src + len);
                dst = (void*)((uint32_t)dst + len);
            }

            if (i == (num_desc - 1))
                base->CHANNEL[channel].XFERCFG |= DMA_CHANNEL_XFERCFG_SETINTA(true);
        }
        else
        {
            // descriptors
            pdma_desc->srcEndAddr = (void*)((uint32_t)srcaddr);
            pdma_desc->dstEndAddr = (void*)((uint32_t)dstaddr);
            pdma_desc->xfercfg = xfer_reg_tmp | DMA_CHANNEL_XFERCFG_CFGVALID_MASK;

            if (i < (num_desc - 1))
            {
                pdma_desc->linkToNextDesc = (pdma_desc + 1);
                pdma_desc->xfercfg |= DMA_CHANNEL_XFERCFG_RELOAD_MASK;
                pdma_desc++;
                size -= len;

                src = (void*)((uint32_t)src + len);
                dst = (void*)((uint32_t)dst + len);
            }
            else
            {
                pdma_desc->xfercfg |= DMA_CHANNEL_XFERCFG_SETINTA(true);
                pdma_desc->linkToNextDesc = 0;
            }
        }
    }

    DMA_DisableChannelPeriphRq(base, channel);
    base->COMMON[DMA_CHANNEL_GROUP(channel)].INTENSET |= 1U << DMA_CHANNEL_INDEX(channel);
}
```

Fig 5. Optimized linked descriptors

3.3 Optimized vs non-optimized method comparison

The application example measures the time before and after creating the linked descriptors. Actual DMA transfer time will not be involved because the performance of the DMA transfer is the same after the linked descriptors are created.

The following result is collected using IAR version 8.20.1. Keil and MCUXpresso may yield similar results. As expected, an increase in the number of descriptors will increase the time for the non-optimized method. The compiler optimization provides some improvement for the non-optimized method where the code has some room for a compiler to improve efficiency.

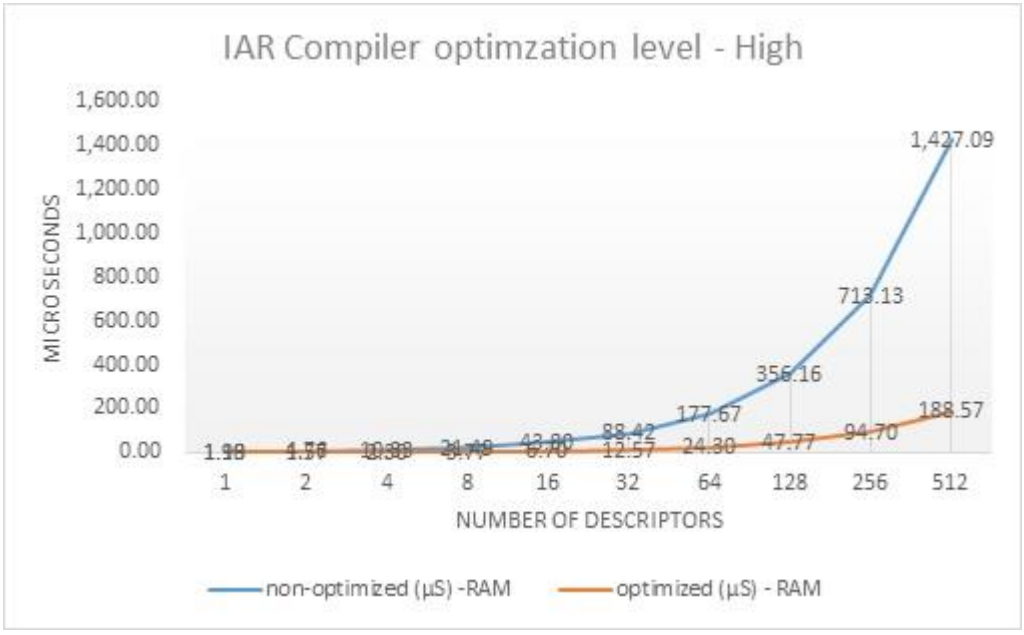

Fig 6. Non-optimized vs optimized method in RAM

4. IDE setup

4.1 IAR

Download and install IAR version of LPCXpresso54608 SDK2.3.1. Extract linked_descriptors-iar.zip and place the linked_descriptors folder under sdk/boards/lpcxpresso54608/driver_examples/dma.

4.2 MDK

Download and install Keil MDK version of LPCXpresso54608 SDK2.3.1. Extract linked_descriptors-iar.zip and place the linked_descriptors folder under sdk/boards/lpcxpresso54608/driver_examples/dma.

4.3 MCUXpresso

Select “Import project(s) from file system…”, locate and select “lpcxpresso54608_driver_examples_dma_linked_descriptors.zip” under project archive (zip), navigate and complete the project import from the file system.

5. References

[1] LPC540xx User manual

[2] LPC54018 DMA Linked Transfer from Memory to Memory

6. Legal information

6.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

6.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Translations — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

6.3 Licenses

Purchase of NXP <xxx> components
<License statement text>

6.4 Patents

Notice is herewith given that the subject device uses one or more of the following patents and that each of these patents may have corresponding patents in other jurisdictions.

<Patent ID> — owned by <Company name>

6.5 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

<Name> — is a trademark of NXP B.V.

7. List of figures

8. Contents

For more information, visit: http://www.nxp.com

Date of release: 27 August 2018
Document identifier: AN12119