AN12351

LPC55(S)xx Dual-DMA Usage

Rev. 3.0 — 30 September 2025 Application note

Document information

Information	Content
Keywords	AN12351, LPC55S69, DMA
Abstract	This application note provides a basic introduction to the LPC55(S)xx Dual-DMA feature.

1 Introduction

This application note provides a basic introduction to the LPC55(S)xx Dual-DMA feature. This document also describes the required registers for configuring a specific direct memory access (DMA) channel, along with the SDK-based configuration steps, hardware platform setup, and program verification. For detailed technical specifications, refer to the LPC55(S)xx data sheet and user manual.

The LPC55(S)xx microcontroller family, based on the Arm Cortex-M33 core, targets a wide range of embedded applications. These devices offer a set of features including:

- Up to 320 kB of on-chip SRAM
- Up to 640 kB on-chip flash
- High-speed and full-speed USB host and device interface with crystal-less operation for full-speed
- Five general-purpose timers
- One SCTimer/PWM
- One RTC/alarm timer
- One 24-bit multirate timer (MRT)
- One windowed watchdog timer (WWDT)
- Eight flexible serial communication peripherals (each of which can be a USART, SPI, I2C, or I2S interface)
- One 16-bit with1.0 Msps analog-to-digital converter (ADC)
- Temperature sensor

The Arm Cortex- M33 provides a security foundation, offering isolation to protect valuable IP and critical data with TrustZone technology.

2 LPC55(S)xx DMA introduction

The LPC55(S)xx series supports DMA for efficient data transfers between memory and peripherals without CPU intervention, which helps free up CPU resources. LPC55(S)xx series also features dual DMA controllers (DMA0 and DMA1) with multiple channels and trigger sources. With flexible channel multiplexing, priority arbitration, and support for up to 1024-word transfer, the DMA system enables high-speed data movement with minimal processor load.

2.1 DMA overview

DMA enables high-speed data transfers between memory and peripherals or between memory locations, without involving the CPU. This data offloading frees the CPU to perform other operations, improving overall system efficiency.

Note: *DMA handles only data transfers. Its transmission efficiency can be lower than the MCU core.*

Each DMA channel supports one DMA request line and one trigger input. For example, both USART0_RX and USART0_TX act as request *input* and generate request *signal*. This request signal produced by a specific peripheral connects to a specific DMA channel. Users must refer to the DMA requests table to configure the corresponding channel.

Table 1 summarizes the various DMA transfer modes, highlighting the corresponding source and destination ports involved in each type of data transfer.

Table 1. DMA transfer mode

DMA transfer mode	Source	Destination
Memory to memory	AHB memory port	AHB memory port
Memory to peripheral	AHB memory port	AHB peripheral port
Peripheral to memory	AHB peripheral port	AHB memory port

2.2 Dual DMA block overview

The LPC55(S)xx microcontroller integrates a Dual DMA architecture featuring two instances of the SmartDMA controller. Users configure each controller as either secure or nonsecure, depending on application requirements.

The following are the key features of the DMA0 controller:

- Supports 22 channels, each with a dedicated multiplexer for selecting from 22 trigger sources.
- Each Flexcomm interface provides separate DMA RX and DMA TX request to the DMA controller.
- The ADC connects to two distinct DMA request channels.
- SCTimer, selected general-purpose timers, and pin interrupts serve as other DMA triggers.
- Offers four selectable DMA triggers from all available DMA channel output triggers.
- Secure hash algorithm 2 (SHA-2) and advanced encryption standard (AES) modules offer DMA channel and trigger interfaces for secure data handling.

The following are the key features of the DMA1 controller:

- Supports 10 channels with multiplexers for 15 trigger sources.

The following features are supported across both DMA0 and DMA1:

- Users assign priority levels to each channel, with up to eight levels supported.
- The system performs continuous priority arbitration.
- Each channel supports single transfers up to 1,024 words.
- Address increment options allow efficient packing and unpacking of data.

2.3 Basic configuration of DMA block

The shows the configuration and interaction of its main components.

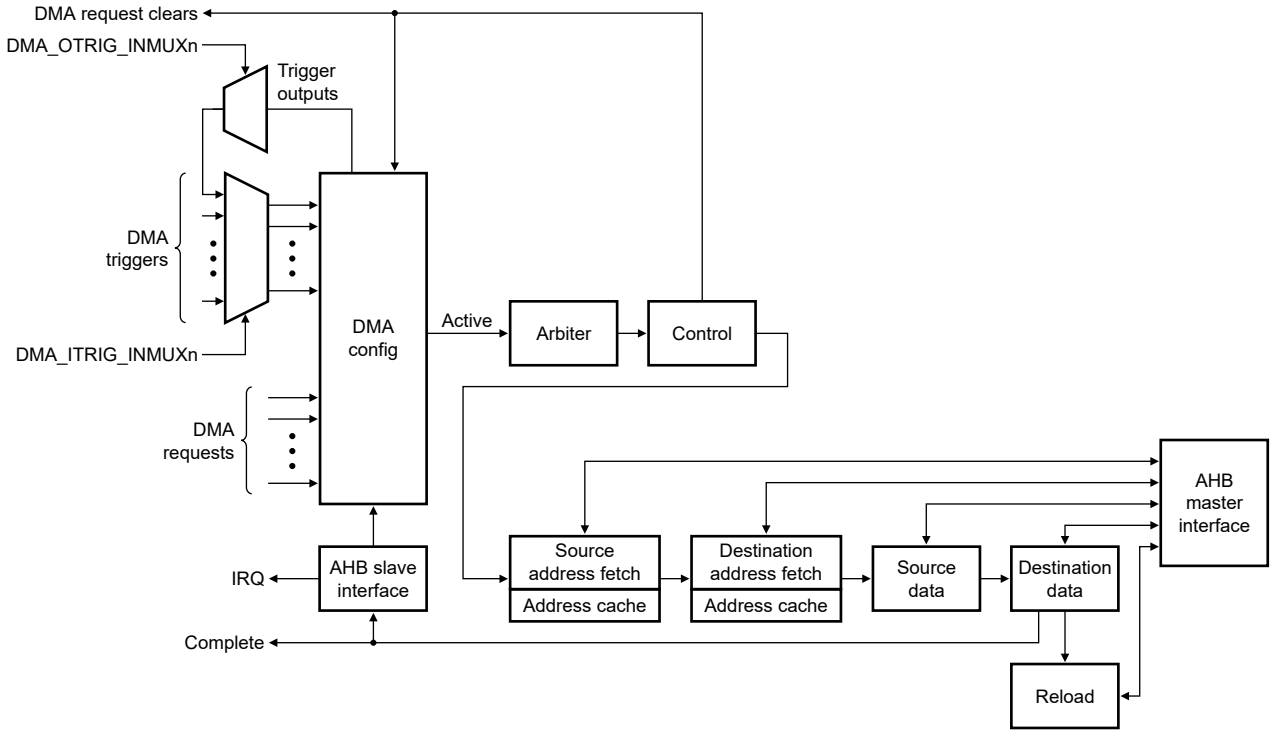

Figure 1. DMA block diagram

Note: *DMA requests are directly connected to the peripherals. Each channel supports one DMA request line and one trigger input. Some DMA requests allow a selection of requests sources. DMA triggers are selected from many possible input sources.*

2.3.1 DMA request and trigger multiplexers

DMA requests regulate the pace of data transfers to match the capabilities of the target peripheral, including any internal FIFO. DMA triggers initiate the actual transfer. Depending on the design requirements, users can select either software or hardware triggers.

The requests and trigger muxes for DMA0 and DMA1 are shown in the DMA0 requests and trigger multiplexers table.

Table 2. DMA0 request and trigger multiplexers

DMA channel	Request input	DMA trigger mux
0	Hash-Crypt DMA request	DMA0_ ITRIG_ INMUX0
1	Spare channel, no request connected	DMA0_ ITRIG_ INMUX1
2	High speed SPI (Flexcomm 8) RX	DMA0_ ITRIG_ INMUX2
3	High speed SPI (Flexcomm 8) TX	DMA0_ ITRIG_ INMUX3
4	Flexcomm interface 0 RX/I2C target	DMA0_ ITRIG_ INMUX4
5	Flexcomm interface 0 TX/I2C controller	DMA0_ ITRIG_ INMUX5
6	Flexcomm interface 1 RX/I2C target	DMA0_ ITRIG_ INMUX6
7	Flexcomm interface 1 TX/I2C controller	DMA0_ ITRIG_ INMUX7
8	Flexcomm interface 3 RX/I2C target	DMA0_ ITRIG_ INMUX8

Table 2. DMA0 request and trigger multiplexers*...continued*

DMA channel	Request input	DMA trigger mux
9	Flexcomm interface 3 RTX/I2C controller	DMA0_ ITRIG_ INMUX9
10	Flexcomm interface 2 RX/I2C target	DMA0_ ITRIG_ INMUX10
11	Flexcomm interface 2 TX/I2C controller	DMA0_ ITRIG_ INMUX11
12	Flexcomm interface 4 RX/I2C target	DMA0_ ITRIG_ INMUX12
13	Flexcomm interface 4 TX/I2C controller	DMA0_ ITRIG_ INMUX13
14	Flexcomm interface 5 RX/I2C target	DMA0_ ITRIG_ INMUX14
15	Flexcomm interface 5 TX/I2C controller	DMA0_ ITRIG_ INMUX15
16	Flexcomm interface 6 RX/I2C target	DMA0_ ITRIG_ INMUX16
17	Flexcomm interface 6 TX/I2C controller	DMA0_ ITRIG_ INMUX17
18	Flexcomm interface 7 RX/I2C target	DMA0_ ITRIG_ INMUX18
19	Flexcomm interface 7 TX/I2C controller	DMA0_ ITRIG_ INMUX19
20	Spare channel, no request connected	DMA0_ ITRIG_ INMUX20
21	ADC0 FIFO 0	DMA0_ ITRIG_ INMUX21
22	ADC0 FIFO 1	DMA0_ ITRIG_ INMUX22

Table 3 provides details on DMA channels trigger source assignment, including their request inputs and the respective DMA trigger MUX configurations.

Table 3. DMA1 request and trigger multiplexers

DMA channel	Request input	DMA trigger mux
0	Hash-Crypt DMA request	DMA1_ ITRIG_ INMUX0
1	Spare channel, no request connected	DMA1_ ITRIG_ INMUX1
2	High speed SPI (Flexcomm 8) RX	DMA1_ ITRIG_ INMUX2
3	High speed SPI (Flexcomm 8) TX	DMA1_ ITRIG_ INMUX3
4	Flexcomm interface 0 RX/I2C target	DMA1_ ITRIG_ INMUX4
5	Flexcomm interface 0 TX/I2C controller	DMA1_ ITRIG_ INMUX5
6	Flexcomm interface 1 RX/I2C target	DMA1_ ITRIG_ INMUX6
7	Flexcomm interface 1 TX/I2C controller	DMA1_ ITRIG_ INMUX7
8	Flexcomm interface 2 RX/I2C target	DMA1_ ITRIG_ INMUX8
9	Flexcomm interface 2 RTX/I2C controller	DMA1_ ITRIG_ INMUX9

Table 4 outlines the available DMA trigger sources, specifying both the input source for each trigger and its corresponding output configuration within the system.

Table 4. DMA trigger sources

DMA trigger	DMA0 trigger input	DMA1 trigger input
0	Pin interrupt 0	Pin interrupt 0
1	Pin interrupt 1	Pin interrupt 1
2	Pin interrupt 2	Pin interrupt 2
3	Pin interrupt 3	Pin interrupt 3

Table 4. DMA trigger sources*...continued*

DMA trigger	DMA0 trigger input	DMA1 trigger input
4	Timer CTIMER0 Match 0	Timer CTIMER0 Match 0
5	Timer CTIMER0 Match 1	Timer CTIMER0 Match 1
6	Timer CTIMER1 Match 0	Timer CTIMER2 Match 0
7	Timer CTIMER1 Match 1	Timer CTIMER4 Match 0
8	Timer CTIMER2 Match 0	DMA output trigger 0
9	Timer CTIMER2 Match 1	DMA output trigger 1
10	Timer CTIMER3 Match 0	DMA output trigger 2
11	Timer CTIMER3 Match 1	DMA output trigger 3
12	Timer CTIMER4 Match 0	SCT0 DMA request 0
13	Timer CTIMER4 Match 1	SCT0 DMA request 1
14	Comparator 0 output	Hash-Crypt output DMA
15	DMA output trigger 0	NA
16	DMA output trigger 1	NA
17	DMA output trigger 2	NA
18	DMA output trigger 3	NA
19	SCT0 DMA request 0	NA
20	SCT0 DMA request 1	NA
21	Hash-Cryptoutput DMA	NA

3 DMA configuration API

This section lists the DMA configuration APIs and provides Data API examples.

3.1 DMA configuration using API list

The LPC55(S)xx microcontroller provides a structured set of DMA configuration APIs that simplify the setup process without requiring changes to the SDK driver. These APIs allow users to enable and manage DMA functionality.

DMA registers fall into two main categories:

- DMA control, interrupt, and status registers
- DMA channel-specific register

Each DMA transfer channel is managed through a dedicated set of registers, including:

- CFG[0:29] – Configuration registers
- CTRLSTAT[0:29] – Control and status registers
- XFERCFG[0:29] – Transfer configuration register

The device includes two DMA controllers:

- DMA0
- DMA1 (secure)

This section explains how to configure these registers using the provided APIs. The main function prototypes are shown in the DMA API calls table.

Table 5. DMA API calls

Function prototype	API description
void DMA_Init (DMA_Type *base)	This function enable the DMA clock, sets the descriptor table, and enables DMA peripheral. • param base: DMA peripheral base address.
void DMA_CreateHandle (dma_handle_t *handle, DMA_Type *base, uint32_t channel)	This function initializes the internal state of a DMA handle when using the transaction API for DMA operations. • param handle: DMA handle pointer that stores the callback functions and parameters • param base: Base address of the DMA peripheral • param channel: DMA channel number
Void DMA_EnableChannel (DMA_Type *base) uint32_t channel)	This function enables the specified DMA channel. • param base: Base address of the DMA peripheral • param channel: DMA channel number
void DMA_SetCallback (dma_handle_t *handle, dma_callback callback, void *userData)	This callback function executes within the DMA IRQ handler. It allows users to perform actions after the current major loop transfer completes. • param handle: DMA handle pointer. • param callback: Pointer to the DMA callback function • param userData: Parameter for the callback function.
void DMA_PrepareTransfer (dma_transfer_config_t *config, void *srcAddr, void *dstAddr, uint32_t byteWidth, uint32_t transferBytes, dma_transfer_ type_t type, void *nextDesc)	This function prepares the DMA transfer configuration structure based on user input. It sets the source and destination addresses, transfer size, and other parameters required for the transfer. • param config: User-defined configuration structure of type `dma_transfer_t` • param srcAddr: Source address for the DMA transfer • param dstAddr: Destination address for the DMA transfer • param byteWidth: Width of each transfer unit in bytes for the DMA transfer • param transferBytes: Number of DMA bytes to transfer • param type: Type of DMA transfer • param nextDesc: Chain custom descriptor to transfer
status_t DMA_SubmitTransfer (dma_handle_t *handle, dma_transfer_config_t *config)	This function submits a DMA transfer request based on the transfer configuration structure. • param handle: DMA handle pointer • param config: Pointer to the DMA transfer configuration structure
void DMA_StartTransfer (dma_handle_t *handle)	This function enables the channel request. • param handle: DMA handle pointer

3.2 Data API example

The following examples demonstrate how to use the DMA data API for different types of data transfers:

- Memory to memory
- Memory to peripheral
- Peripheral to memory

3.2.1 Memory to memory

This example demonstrates how to configure and execute a memory-to-memory DMA transfer using the LPC55(S)xx SDK APIs. It outlines the key steps required to initialize the DMA controller, set up the transfer configuration, and monitor completion.

1. Initialize DMA controller (DMA_Init):
   Enable the DMA clock, set the descriptor table, and activate the DMA peripheral.

```
/** Peripheral DMA0 base address */
    #define DMA0_BASE_NS (0x40082000u)
    /** Peripheral DMA0 base pointer */
    #define DMA0_NS                  ((DMA_Type *)DMA0_BASE_NS)
    volatile DMA_Type *DMA0_NS_Base = (DMA_Type *)DMA0_NS;
    DMA_Init(DMA0);
```

2. Create DMA handle (DMA_CreateHandle):
   Create a DMA handle for the selected channel.

```
DMA_CreateHandle(&g_DMA_Handle, DMA0, 0);
```

   Note: *The user can use DMA1. For example, calling* `DMA_CreateHandle(&g_DMA_Handle, DMA1, 0)` *selects DMA1_Channel0 as the channel for data transfer.*

3. Enable DMA channel (DMA_EnableChannel):
   This function enables the DMA channel. To enable the selected channel, the user calls the API using DMA0_Channel0, which is selected at step 2.

```
DMA_EnableChannel(DMA0, 0);
```

   Note: *The LPC55(S)xx includes two DMA controllers: DMA0 and DMA1. When using DMA0, the user must reference COMMON[0]; for DMA1, the correct structure is COMMON[1]. The DMA_Type register layout typedef provides the complete definition of all related registers.*

```
/** DMA - Register Layout Typedef */
typedef struct {
  __IO uint32_t CTRL;                                /**< DMA control., offset:
 0x0 */
  __I  uint32_t INTSTAT;                             /**< Interrupt status.,
 offset: 0x4 */
  __IO uint32_t SRAMBASE;                            /**< SRAM address of the
 channel configuration table., offset: 0x8 */
       uint8_t RESERVED_0[20];
  struct {                                           /* offset: 0x20, array
 step: 0x5C */
    __IO uint32_t ENABLESET;                           /**< Channel Enable read
 and Set for all DMA channels., array offset: 0x20, array step: 0x5C */
         uint8_t RESERVED_0[4];
    __O  uint32_t ENABLECLR;                           /**< Channel Enable
 Clear for all DMA channels., array offset: 0x28, array step: 0x5C */
         uint8_t RESERVED_1[4];
    __I  uint32_t ACTIVE;                              /**< Channel Active
 status for all DMA channels., array offset: 0x30, array step: 0x5C */
         uint8_t RESERVED_2[4];
    __I  uint32_t BUSY;                                /**< Channel Busy status
 for all DMA channels., array offset: 0x38, array step: 0x5C */
         uint8_t RESERVED_3[4];
    __IO uint32_t ERRINT;                              /**< Error Interrupt
 status for all DMA channels., array offset: 0x40, array step: 0x5C */
         uint8_t RESERVED_4[4];
    __IO uint32_t INTENSET;                            /**< Interrupt Enable
 read and Set for all DMA channels., array offset: 0x48, array step: 0x5C */
```

```
        uint8_t RESERVED_5[4];
    __O  uint32_t INTENCLR;                          /**< Interrupt Enable
 Clear for all DMA channels., array offset: 0x50, array step: 0x5C */
        uint8_t RESERVED_6[4];
    __IO uint32_t INTA;                              /**< Interrupt A status
 for all DMA channels., array offset: 0x58, array step: 0x5C */
        uint8_t RESERVED_7[4];
    __IO uint32_t INTB;                              /**< Interrupt B status
 for all DMA channels., array offset: 0x60, array step: 0x5C */
        uint8_t RESERVED_8[4];
    __O  uint32_t SETVALID;                          /**< Set ValidPending
 control bits for all DMA channels., array offset: 0x68, array step: 0x5C */
        uint8_t RESERVED_9[4];
    __O  uint32_t SETTRIG;                           /**< Set Trigger control
 bits for all DMA channels., array offset: 0x70, array step: 0x5C */
        uint8_t RESERVED_10[4];
    __O  uint32_t ABORT;                             /**< Channel Abort
 control for all DMA channels., array offset: 0x78, array step: 0x5C */
  } COMMON[DMA_COMMON_COUNT];
      uint8_t RESERVED_1[900];
 struct {                                        /* offset: 0x400, array
 step: 0x10 */
    __IO uint32_t CFG;                               /**< Configuration
 register for DMA channel ., array offset: 0x400, array step: 0x10, irregular
 array, not all indices are valid */
    __I  uint32_t CTLSTAT;                           /**< Control and status
 register for DMA channel ., array offset: 0x404, array step: 0x10, irregular
 array, not all indices are valid */
    __IO uint32_t XFERCFG;                           /**< Transfer
 configuration register for DMA channel ., array offset: 0x408, array step:
 0x10, irregular array, not all indices are valid */
        uint8_t RESERVED_0[4];
  } CHANNEL[DMA_CHANNEL_COUNT];
} DMA_Type;
```

4. Set DMA callback (DMA_SetCallback):
   This function sets a callback that executes within the DMA IRQ handler. The system uses this callback to check interrupt A or interrupt B flags and monitor whether the software sets them using the following API. using the following API.

```
DMA_SetCallback(&g_DMA_Handle, DMA_Callback, NULL);
```

5. Prepare DMA transfer (DMA_PrepareTransfer):
   This function prepares the transfer configuration structure according to the user input. The following code represents that the user can configure DMA transfer source address, destination address, DMA transfer type, and custom descriptor according to data transferring requirements.

```
DMA_PrepareTransfer(&transferConfig,srcAddr,destAddr,sizeof(srcAddr[0]),sizeof(srcAddr)
 kDMA_MemoryToMemory,NULL);
```

   Note: *The sixth parameter determines whether the source and destination addresses must increment during the transfer. For example, both addresses must increment to copy data sequentially between memory locations.*

```
config->xfercfg.srcInc = 1;
config->xfercfg.dstInc = 1;
config->isPeriph = false;
```

6. Submit DMA transfer (DMA_SubmitTransfer):
   This function submits a DMA transfer request based on the specified transfer configuration structure. The following code shows that the user completes the DMA transfer configuration by setting up 'dma_xfercfg_t'

structure. This structure includes the option to reload channel configuration, perform a software trigger, and specify other information.

```
/*! @brief DMA transfer configuration */
typedef struct _dma_xfercfg
{
    bool valid;             /*!< Descriptor is ready to transfer */
    bool reload;            /*!< Reload channel configuration register after
                                 current descriptor is exhausted */
    bool swtrig;            /*!< Perform software trigger. Transfer if fired
                                 when 'valid' is set */
    bool clrtrig;           /*!< Clear trigger */
    bool intA;              /*!< Raises IRQ when transfer is done and set
 IRQA status register flag */
    bool intB;              /*!< Raises IRQ when transfer is done and set
 IRQB status register flag */
    uint8_t byteWidth;      /*!< Byte width of data to transfer */
    uint8_t srcInc;         /*!< Increment source address by 'srcInc' x
 'byteWidth' */
    uint8_t dstInc;         /*!< Increment destination address by 'dstInc' x
 'byteWidth' */
    uint16_t transferCount; /*!< Number of transfers */
} dma_xfercfg_t;
```

Then, the user can create a specific DMA descriptor to be used in a chain in transfer by using the following API.

```
DMA_StartTransfer(&g_DMA_Handle);
```

Note: *The default value of bit 1 (HWTRIGEN) in the CFG register is '0'. This means that the hardware trigger is unused. To trigger a DMA transfer, such as Pin interrupt 0, Timer CTIMER0 Match 0, and others, the user must follow the following steps. These steps include completing extra configuration using hardware.*

a. *Set bit 1 (HWTRIGEN) of the CFG register must be set by using the following pseudo code.*

```
Base->CHANNEL[x]. CFG  |= 0x2U;
```

b. *Bit 2 of the `XFERCFGn` register, which enables the software trigger, must be disabled so that the channel can wait for a hardware trigger signal. The following pseudo code demonstrates this configuration:*

```
Base->CHANNEL[x]. XFERCFG |=~ 0x4U
```

c. *User must choose 'DMA trigger sources' by configuring register 'DMA0_ITRIG_INMUX[0:22]' or 'DMA1_ITRIG_INMUX[0:9]'. The following pseudo code means that we can choose hardware trigger 2 'Pin interrupt 2' for DMA0_Channlx's request input. The user selects the DMA trigger source by configuring the DMA0_ITRIG_INMUX[0:22] or DMA1_ITRIG_INMUX[0:9] register. The following pseudo code shows how to choose hardware trigger 2 (Pin Interrupt 2) for the request input of DMA0_Channelx.*

```
Base->DMA0_ITRIG_INMUX[x] = 0x2;
```

With the above configuration of six main steps, the DMA is ready to transfer data. Now, the user waits for the DMA transfer to complete by monitoring the value of g_Transfer_Done, which is updated in the DMA_Callback function.

```
void DMA_Callback(dma_handle_t *handle, void *param, bool transferDone,
 uint32_t tcds)
{
          if (transferDone){
             g_Transfer_Done = true;
          }
}
```

```
while (g_Transfer_Done != true)
          {}
```

3.2.2 Memory to peripheral

The user transfers data from memory to a peripheral. For example, the application transfers data from RAM to `USART0_FIFOWR`. The main steps follow the same procedure as described in Section 3.2.1, except for List item.

```
/** Peripheral USART0 base address  */
#define USART0_BASE_NS (0x40086000u)
/** Peripheral USART0 base pointer */
#define USART0_NS  ((USART_Type *)USART0_BASE_NS)
volatile USART_Type *USART0_NS_Base = (USART_Type *)USART0_NS;
DMA_PrepareTransfer(&transferConfig,srcAddr,
                    USART0_NS_Base ->FIFOWR, sizeof(srcAddr[0]),sizeof(srcAddr),
                    kDMA_MemoryToPeripheral,NULL);
```

Note: *As the peripheral (`USART0_FIFOWR`) has a fixed base address (`0x40086000h + 0xE20`), the destination address does not increment. The source address must increment when using memory (RAM) as the source.*

```
/* Peripheral register - destination doesn't increment */
    config->xfercfg.srcInc = 1;
    config->xfercfg.dstInc = 0;
    config->isPeriph = true;
```

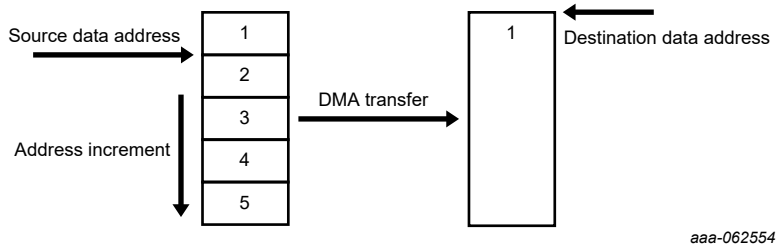

Figure 2. Source address must increment

3.2.3 Peripheral to memory

The user transfers data from a peripheral to memory. For example, when a sensor communicates with a microcontroller unit through `USART0`, then the user can store the sensor the data collected by the sensor into Memory by transferring data from `USART0_FIFORD` to RAM. The main steps follow the same procedure as described in Section 3.2.1, except for step 5.

- The application uses bits from 0 to 8 of the FIFORD register to store received data from FIFO. The number of bits used depends on the DATALEN and PARITYSEL settings.
- For example, a user can transfer a character from USART0 > FIFO1D to RAM by using the following API to configure the DMA_PrepareTransfer function.

```
DMA_PrepareTransfer(&transferConfig, (void *)(&USART0_NS_Base->FIFORD),desAddr,
 4, 4, kDMA_PeripheralToMemory, NULL);
```

Note: *As the peripheral (USART0_NS > FIFORD) has a fixed base address (0x4008 6000 h+ 0xE30), its address does not increment. When using it as the source address and RAM as the destination data address, the destination address must increment.*

```
/*Peripheral register - source doesn't increment */
    config->xfercfg.srcInc = 0;
    config->xfercfg.dstInc = 1;
    config->isPeriph = true;
```

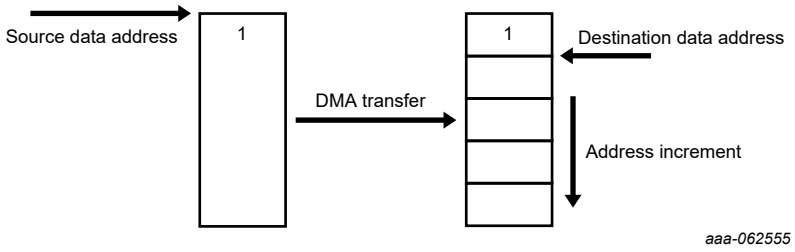

Figure 3. Source address increment

4 DMA verification

This application note provides examples to complete DMA transfer operations in Section 3.2.1, Section 3.2.2, and Section 3.2.3 modes. It uses the LPC55S6x as a reference to demonstrate the functionality of Dual DMA.

All demos are based on the IAR. The MCUXpresso IDE from NXP also supports these examples. The user can download a related project to the LPC55S6x to verify Dual DMA functionality. The demo uses USB Virtual COM (VCOM) to display log information.

4.1 Hardware design

This NXP LPCXpresso development board features an LPC5500 series Arm Cortex-M microcontroller as the main control unit. It integrates an LPC4322 debug probe for program downloading and debugging.

The development board provides rich interface resources, including multiple USB connectors, a micro-SD card slot, audio input or output jacks, and various expansion connectors. These interface options include LPCXpresso and Mikroe Click interfaces.

The board includes functional buttons, such as ISP boot, reset, user, and wake buttons. An RGB LED indicator enhances usability, making it a complete embedded system development platform.

To download and debug the project and view serial port data, the user can connect USB port P6 with the PC by using the USB interface.

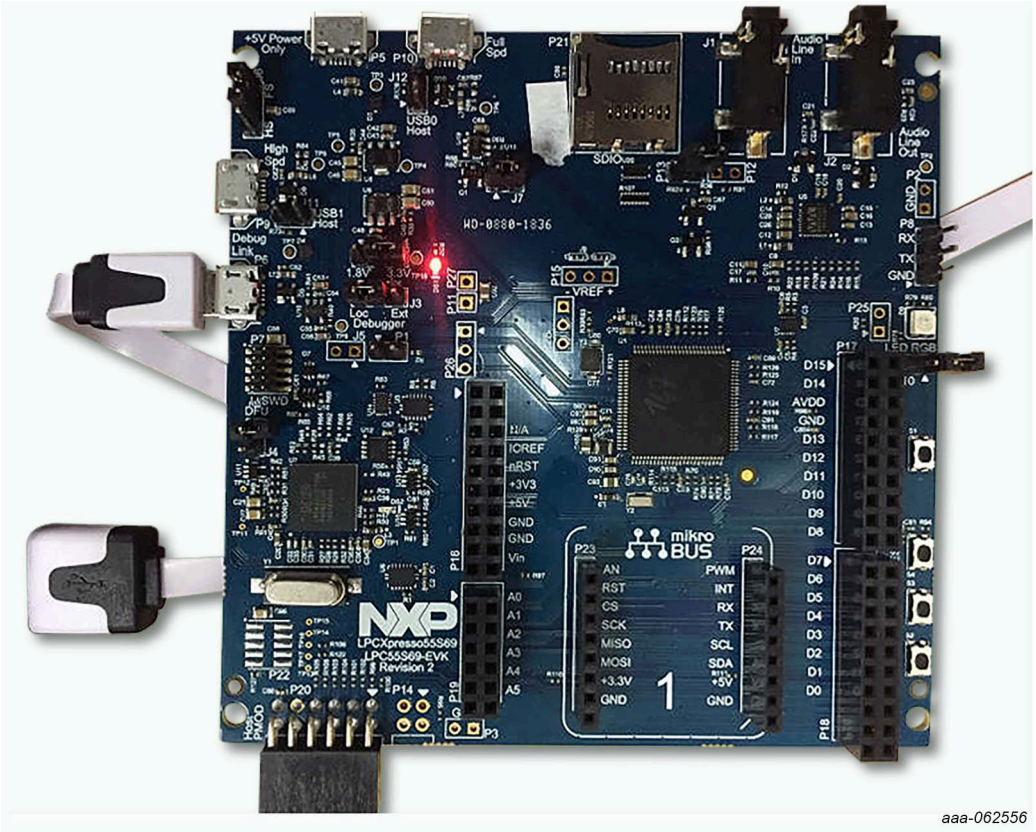

Figure 4. Connect LPCXpresso55S69 with PC

4.2 Software setup

To verify the sample DMA projects, the user must first set up the software environment using supported development tools.

Two IDEs are used to verify the sample DMA projects:

- IAR Embedded Workbench v8.32.1.
- MCUXpresso IDE v10.3.0 (available at https://www.nxp.com).

To compile and load the software example-using IAR IDE, follow the steps below:

1. Connect the USB port P6 of the LPCXpresso55S69 board to the PC (set J3 to Loc, P4 to 3.3 V, and J6 to FS).
2. Unzip the project folder `...\LPC55S6x Dual-DMA.zip` for testing.
3. Compile all projects by clicking *Compile (Ctrl+F7)* or *Make (F7)* from the Quickstart menu.
4. Start the debugger by clicking *Download and Debug (Ctrl+D)*. This step compiles, flashes the code, and launches the debugger.
5. Use terminal software such as *teraterm* to view log output via USB Virtual COM (VCOM).

4.3 Program verification

After downloading the DMA project, press the RESET (S4) button to start the program. The system displays a menu via the USB Virtual COM (VCOM) port, allowing you to select and verify any DMA transfer mode in any order.

The DMA selection mode menu is given below:

```
Select a DMA transfer mode
1. Memory to Memory (Software Trigger)
2. Memory to Peripheral (Hardware Trigger)
3. Memory to Peripheral (Software Trigger)
4. Peripheral to Memory (Software Trigger)
```

To use the menu, select any DMA transfer mode by entering its number and follow the instructions shown.

- To trigger DMA memory to memory transfer by software, Input 1. The below log confirms a successful memory-to-memory software trigger transfer.

```
Entering 1.M2M (Note: Do not need other operation) ...
desAddr[] Buffer:
0       0       0       0
desAddr[] Buffer:
1       2       3       4
DMA Memory to Memory(Software Trigger) example finish, pressing 'S4_RESET' to
 verify other DMA transfer.
```

- To trigger DMA memory to peripheral transfer by hardware, input 2. The below log confirms a successful memory-to-peripheral hardware trigger transfer.

```
Entering 2.M2P (Note: User need press Button 'S9_USER' to trigger DMA) ...
1234
DMA Memory to Peripheral(Hardware Trigger) example finish, pressing 'S4_RESET'
 to verify other DMA transfer.
```

- To trigger DMA memory to peripheral transfer by software, input 3. The below log confirms a successful memory to peripheral software trigger transfer.
-

```
Entering 3.M2P (Note: Do not need other operation) ...
1234
DMA Memory to Peripheral(Software Trigger) example finish, pressing 'S4_RESET'
 to verify other DMA transfer.
```

- To trigger DMA peripheral to memory by software, input 4. The below log confirms a successful peripheral to memory software trigger transfer.

```
Entering 4.P2M (Note: User need input a character) ...
Input a character for P2M and then read this character from (USART0-
>FIFOWR(RO)):4
desAddr[] Buffer: - 4
DMA Peripheral to Memory(Software Trigger) example finish, pressing 'S4_RESET'
 to verify other DMA transfer.
```

Note: *You can select and verify any DMA transfer mode in any order. After each operation, press S4_RESET to return to the menu and choose another mode as needed.*

5 Conclusion

The software example provided with this application note demonstrates the basic configuration of the LPC55S6x DMA block. The user can select either DMA0 or DMA1 channels to transfer data based on actual requirements.

6 Acronyms

Table 6 lists the acronyms used in this document along with their description.

Table 6. Acronyms

Acronym	Description
ADC	Analog-to-digital converter
AHB	Advanced high-performance bus
CFG	Configuration register
CTRLSTAT	Control and status register
DMA	Direct memory access
I2C	Inter-integrated circuit
I2S	Integrated interchip sound
PWM	Pulse-width modulation
RTC	Real-time clock
SHA-2	Secure hash algorithm 2
SPI	Serial peripheral interface
USART	Universal synchronous/ssynchronous receiver/transmitter
VCOM	Virtual COM port
WWDT	Windowed watchdog timer

7 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

8 Revision history

Table 7 summarizes the changes since the initial release.

Table 7. Revision history

Document ID	Release date	Description
AN12351 v.3.0	30 September 2025	• Updated the table, Table 2 • Made editorial changes
AN12351 v.2	27 October 2020	Updated LPC55(S)xx for LPC55S6x
AN12351 v.1	26 February 2020	Figure 1 updated
AN12351 v.0	14 February 2019	Initial release

Legal information

Definitions

Draft — A draft status on a document indicates that 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 in a draft version of a document and shall have no liability for the consequences of use of such information.

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.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

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.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

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

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

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

NXP — wordmark and logo are trademarks of NXP B.V.

Contents