DMA_Linked_List_Mode_1 for KIT_AURIX_TC297_TFT

DMA Linked List Mode usage

AURIX™ TC2xx Microcontroller Training
V1.0.1

Scope of work

DMA Linked Lists are used to execute a series of DMA transactions without CPU intervention.

In this training, four DMA transactions are configured in DMA Linked List mode.
Triggering one DMA transaction leads to the execution of all DMA transactions consecutively.
An interrupt is triggered at the completion of the last DMA transaction
If DMA has correctly transferred the data, the LED driven by port pin 13.3 toggles and a new cycle starts again. Otherwise, the LED driven by port pin 13.2 turns ON and no more DMA transfers are done.

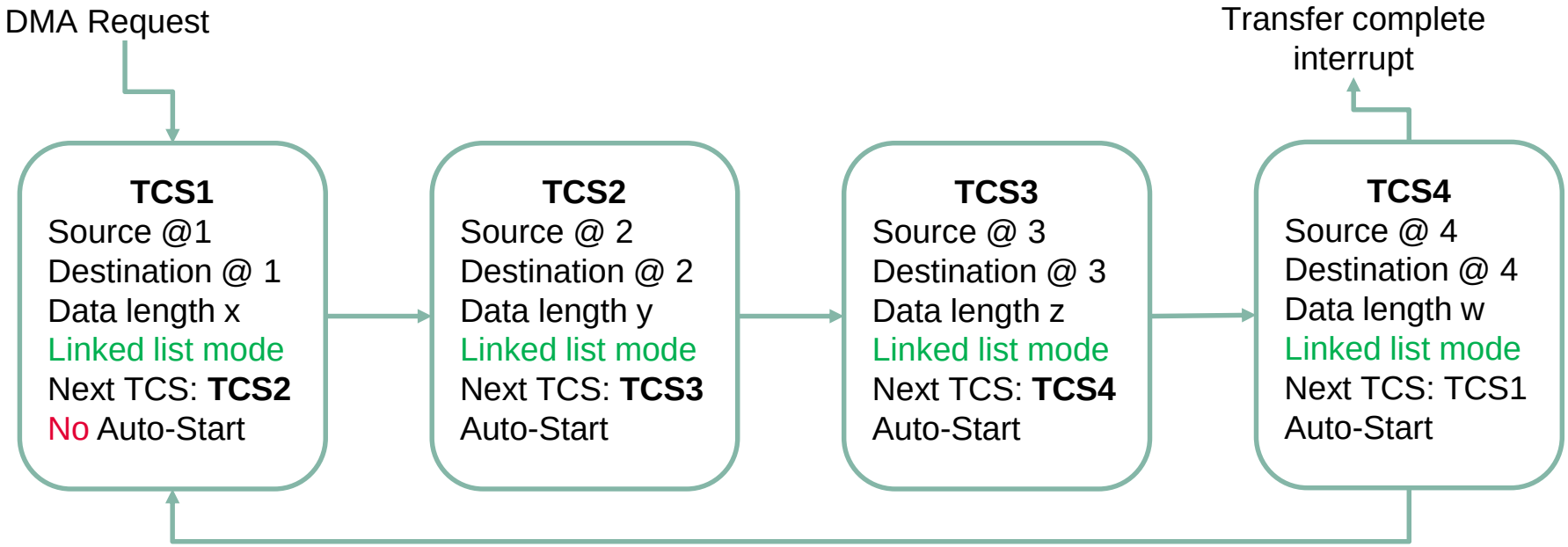
Introduction

- The Direct Memory Access (DMA) moves data from source locations to destination locations **without** the intervention of the CPU or any other on-chip devices.
- Among other features, the DMA has the capability to execute a series of DMA transactions by the same DMA channel; this is ensured by the Linked List operations.
- A DMA transaction in Linked List mode can be configured to **auto-start**.

Introduction

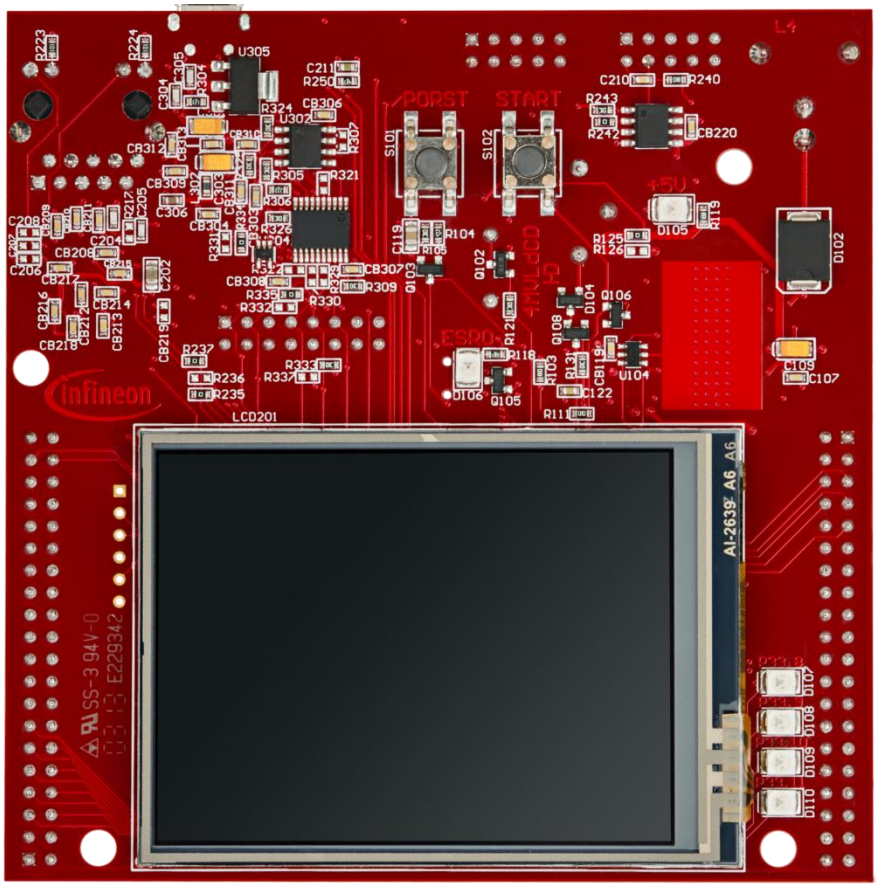
A typical DMA Linked List use case is illustrated in the figure below

- Several DMA transactions are configured and stored in RAM
- Configure DMA channel with the first Transaction Control Set (TCS1)
- Set a HW or SW DMA Channel request
- All configured DMA transactions will be executed consecutively
- Trigger an interrupt after the completion of the last DMA transaction



Hardware setup

This code example has been developed for the board KIT_AURIX_TC297_TFT_BC-Step.



Implementation

Initialization phase

The initialization phase is ensured by the ***initDmaLinkedList()*** function as following:

1. DMA module initialization

› Create the DMA module configuration:
IfxDma_Dma_Config dmaConfig;
IfxDma_Dma_initModuleConfig(&dmaConfig, &MODULE_DMA);

› Initialize the DMA software module:
IfxDma_Dma dma;
IfxDma_Dma_initModule(&dma, &dmaConfig);

The functions described above are provided by the iLLD header ***IfxDma_Dma.h***

Implementation

Initialization phase

2. DMA Channel Configuration

› Create the DMA channel default configuration:
IfxDma_Dma_ChannelConfig cfg;
IfxDma_Dma_initChannelConfig(&cfg, &dma);

› DMA Linked List configuration:
- Channel selection:
cfg.channelId = IfxDma_ChannelId_0;
- Number of DMA transfers in the DMA transaction (16 DMA transfers per DMA transaction)
cfg.transferCount = NUM_TRANSFERED_WORDS;
- Channel move data width (one DMA move = 32 bit)
cfg.moveSize = IfxDma_ChannelMoveSize_32bit;
- Channel trigger mode (one DMA transaction or one DMA transfer per trigger request)
cfg.requestMode = IfxDma_ChannelRequestMode_completeTransactionPerRequest;

The functions described above are provided by the iLLD header ***IfxDma_Dma.h***

Implementation

- DMA Linked List configuration (Cont.):
  - Channel operation mode (DMA Linked List mode)
    cfg.shadowControl = IfxDma_ChannelShadow_linkedList;
  - Source and Destination buffers addresses of each DMA transaction
    cfg.sourceAddress = source[i];
    cfg.destinationAddress = destination[i];
  - Address of the next Transaction Control Set (TCS) in the DMA Linked List
    cfg.shadowAddress = (uint32)&linkedList[(i + 1) % NUM_LINKED_LIST_ITEMS];
  - Enable channel interrupt for the last DMA transaction
    cfg.channelInterruptEnabled = TRUE;
  - Store the configuration into RAM memory in a transaction control set format
    IfxDma_Dma_initLinkedListEntry((void *)&linkedList[i], &cfg);
  - Configure DMA channel registers with the first DMA transaction parameters (***if i == 0***)
    IfxDma_Dma_initChannel(&chn, &cfg);
  - Enable Auto-Start feature for the subsequent DMA transactions (***if i != 0***)
    linkedList[i].CHCSR.B.SCH = 1;

The functions described above are provided by the iLLD header ***IfxDma_Dma.h***

Implementation

3. DMA Channel Interrupt configuration

- Get the DMA Channel Interrupt configuration register:
  DMA_CH0_SRC = IfxDma_Dma_getSrcPointer(&chn);

- Set Interrupt Service Provider (CPU0) and Priority (50):
  IfxSrc_init(DMA_CH0_SRC, IfxSrc_Tos_cpu0, ISR_PRIORITY_DMA_CH0);

- Enable Interrupt:
  IfxSrc_enable(DMA_CH0_SRC);

The function ***IfxDma_Dma_getSrcPointer()*** is provided by the iLLD header ***IfxDma_Dma.h***
IfxSrc_init() and ***IfxSrc_enable()*** functions are provided by the iLLD header ***IfxSrc.h***

Implementation

4. LEDs Configuration:

› LEDs definition to be user friendly (Pre-processor defines):
#define PASS_LED &MODULE_P13,3
#define FAIL_LED &MODULE_P13,2

› Configure port pins connected to LEDs in push-pull output mode:
IfxPort_setPinMode(PASS_LED, IfxPort_Mode_outputPushPullGeneral);
IfxPort_setPinMode(FAIL_LED, IfxPort_Mode_outputPushPullGeneral);

› Switch off LEDs (initial state):
IfxPort_setPinHigh(PASS_LED);
IfxPort_setPinHigh(FAIL_LED);

The functions described above are provided by the iLLD header ***IfxPort.h***

Implementation

Transfer phase

The data transfer phase is launched by calling the ***startDmaLinkedListTransfer()*** function and it includes the following:

1. Fill the DMA source buffers with data to be sent
2. Trigger a software DMA request:
   IfxDma_Dma_startChannelTransaction(&chn);

The function ***IfxDma_Dma_startChannelTransaction()*** is provided by the iLLD header ***IfxDma_Dma.h***

Implementation

Interrupt Service Routine

The Interrupt Service Routine (ISR) is a user function executed when the interrupt is triggered. In this example, the DMA channel interrupt is triggered after the completion of the last DMA transaction of the linked list.
The implemented ISR ***ISR_dma_ch0*** ensure the following:

- Compare destination buffers to source buffers
  - In case of a data mismatch :
    - Switch On FAIL_LED***: IfxPort_setPinLow(FAIL_LED)***
    - Switch Off PASS_LED***: IfxPort_setPinHigh(PASS_LED)***
  - In case of a data match:
    - Toggle PASS_LED: ***IfxPort_togglePin(PASS_LED)***
    - Clear Destination buffers
    - Trigger a new transfer request: ***startDmaLinkedListTransfer()***

Note: One second delay is added between every DMA transfer operation.
The delay is ensured by **wait_ms()** function and based on STM Timer.

Run and Test

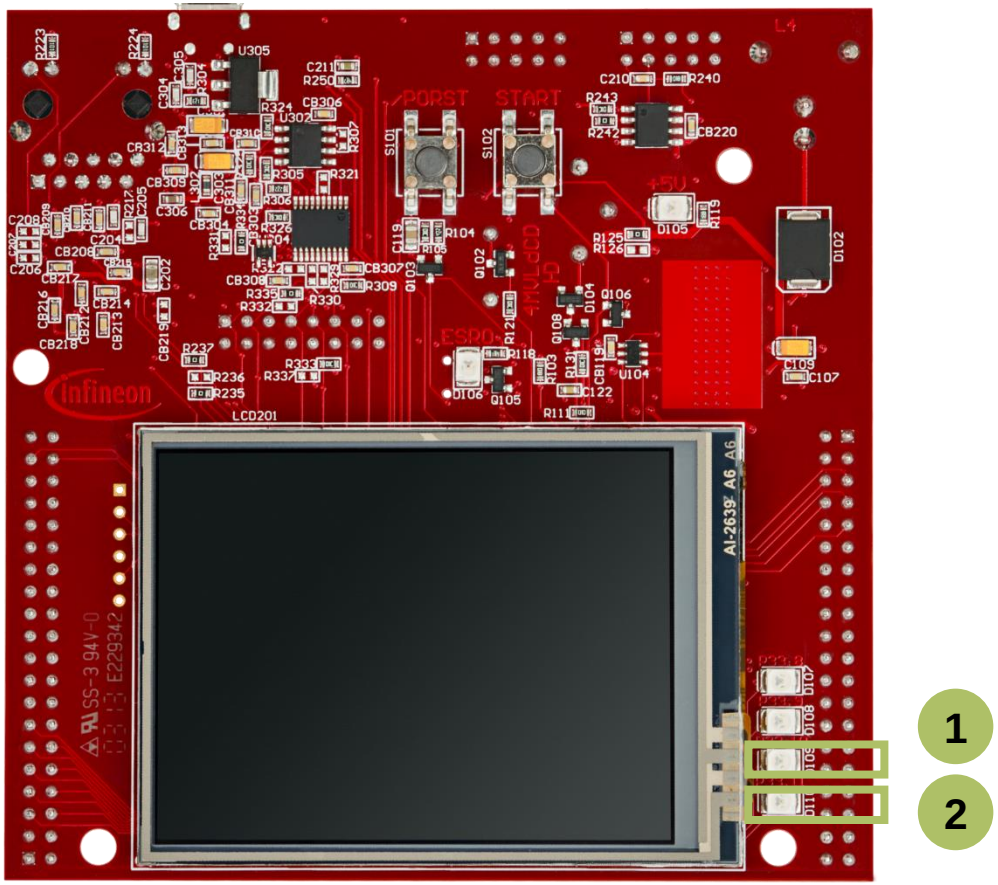
After code compilation and flashing the device, observe the LEDs' behavior:

Execution error:

- LED D109 (1) is On
- LED D110 (2) is Off

No Execution error:

- LED D109 (1) is Off
- LED D110 (2) is toggling every **one second** approximately.



References

- AURIX™ Development Studio is available online:
- https://www.infineon.com/aurixdevelopmentstudio
- Use the *„Import...“* function to get access to more code examples.

- More code examples can be found on the GIT repository:
- https://github.com/Infineon/AURIX_code_examples

- For additional trainings, visit our webpage:
- https://www.infineon.com/aurix-expert-training

- For questions and support, use the AURIX™ Forum:
- https://www.infineonforums.com/forums/13-Aurix-Forum

Revision history

Revision	Description of change
V1.0.1	Update of version to be in line with the code example’s version
V1.0.0	Initial version

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2020-12
Published by
Infineon Technologies AG
81726 Munich, Germany

Do you have a question about this document?
Email: erratum@infineon.com

Document reference
DMA_Linked_List_Mode_1
_KIT_TC297_TFT

IMPORTANT NOTICE
The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics ("Beschaffenheitsgarantie") .

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer's compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer's products and any use of the product of Infineon Technologies in customer's applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (www.infineon.com).

WARNINGS
Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.