XT32H05x

XT32 microcontroller DMA

Application notes

Rev 0.0.0

Original Release Date: 28-Sep-2023

Revised :

Revision History

Release	Date	Author	Summary of Change
V0.0.0	28/09/2023	Shirling Liu	Initial

Contents

List of Figures

List of Tables

1 Introduce

This application note serves as a comprehensive guide for software developers, offering essential information on DMA. It covers fundamental concepts and provides guidelines to ensure proper utilization of DMA in software development projects. Whether you're a beginner or an experienced developer, this document will equip you with the necessary knowledge and best practices to effectively configure and utilize DMA in your applications.

1.1 Required peripherals

This application involves modules as table 1.

Table 1. Modules in example

Sub-module	Peripheral use	Note
DMA	transfer data from a source to a destination	
UART	Only use to print result tips message.	

1.2 Compatible devices

This example is compatible with the devices in Table 2.

Table 2. Device list

Product	EVB
XT32H050	XB002823

2 Design description

2.1 Feature overview

DMA is a controller to transfer data from a source to a destination.

- 8 independent channels with flexible mapping
- Transfer data from memory to memory, from peripheral to memory, from memory to peripheral and from peripheral to peripheral.
- Flow control
- Generate interrupts

2.2 Design steps

How to transfer data from peripheral to memory or from memory to peripheral, please refer to the documents: XT32H0xxB--uart-AN23030C, XT32H0xxB--spi-AN23050C, and XT32H0xxB--i2c-AN23060C.

This example shows how to transfer data from flash memory to ram as the

following steps:

1. Configure DMA channel and parameters.
2. Register interrupt callback interface.
3. Start the DMA data transfer command.

2.3 Design considerations

2.4 Software flowchart

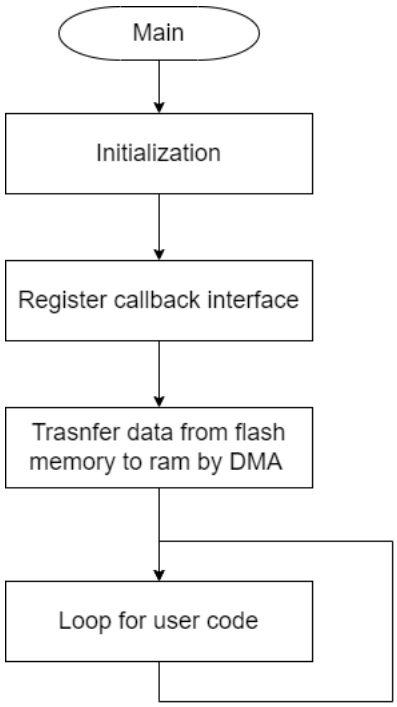

Figure 1. Application flow

2.5 Reference code

Configure DMA Peripheral using HAL_DMA_Init.

```
       /* Enable FLASH DMA transfer*/
    MODIFY_REG(SYSCFG->SYS_CFG, SYSCFG_SYS_CFG_FLASHDMA,
SYSCFG_SYS_CFG_FLASHDMA);

    hDmamem1.Instance     = DMA1_Channel4;
    hDmamem1.ChannelIndex = 4;
    hDmamem1.Init.Direction = DMA_MEMORY_TO_MEMORY_FC_DMAC;
    hDmamem1.Init.SrcInc    = DMA_SINC_INCREMENT;
    hDmamem1.Init.DstInc    = DMA_DINC_INCREMENT;
    hDmamem1.Init.SrcDataSize  = DMA_SDATAALIGN_WORD;
    hDmamem1.Init.DstDataSize  = DMA_DDATAALIGN_WORD;
    hDmamem1.Init.SrcBurstSize = DMA_SRC_MSIZE_4;
    hDmamem1.Init.DstBurstSize = DMA_DST_MSIZE_4;
    hDmamem1.Init.Mode = DMA_SGLBLK;
    hDmamem1.DMAAux = DMAAUX1;
    if (HAL_DMA_Init(&hDmamem1) != HAL_OK)
    {
      /* Error_Handler */
    }
```

XT_Dmamem_Task register interrupt callback interface and start DMA transfer process.

```
void XT_Dmamem_Task(void)
{
    /* USER CODE */
    DBG_printf("DMA Memory2memeory test Start!\n");
    /* Select Callbacks functions called after Transfer complete and
Transfer error */
```

```
    HAL_DMA_RegisterCallback(&hDmamem1, HAL_DMA_XFER_CPLT_CB_ID,
XT_Dmamem_TransferCplt);
    HAL_DMA_RegisterCallback(&hDmamem1, HAL_DMA_XFER_BLOCK_CB_ID,
XT_Dmamem_TransferBLKCplt);
    HAL_DMA_RegisterCallback(&hDmamem1, HAL_DMA_XFER_ERROR_CB_ID,
XT_Dmamem_TransferError);

    /* Configure the source, destination and buffer size DMA fields and
Start DMA Channel/Stream transfer */
    /* Enable All the DMA interrupts */
    if (HAL_DMA_Start_IT(&hDmamem1, (uint32_t)&aSRC1_FLASH_Buffer,
(uint32_t)&aDst1, DMA_BUFFER_SIZE) != HAL_OK)
    {
      /* Transfer Error */
        Error_Handle();
    }

    while(u1Dmamem_cbState==CB_DMA_IDLE);
    u1Dmamem_cbState = CB_DMA_IDLE;
    if(Buffercmp(aSRC1_FLASH_Buffer,aDst1,DMA_BUFFER_SIZE)!=0)
    {
        DBG_printf("DMA Memory2memeory test fail.\n");
        Error_Handle();
    }else
    {
        DBG_printf("DMA Memory2memeory test success.\n");
    }
}
```

2.6 Additional resources

- XT32H0xxB--reference manual

- XT32H0xxB--uart-AN23030C
- XT32H0xxB--spi-AN23050C
- XT32H0xxB--i2c-AN23060C