南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Embedded System and Microcomputer Principle

LAB13 DMA

2023 Fall
wangq9@mail.sustech.edu.cn

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

CONTENTS

01

DMA Principle Description

1. DMA Principle Description

-- Introduction

- DMA：Direct Memory Access
- Used in order to provide high-speed data transfer between peripherals and memory as well as memory to memory.
- Data can be quickly moved by DMA without any CPU actions. This keeps CPU resources free for other operations.

1. DMA Principle Description
-- DMA of STM32F103

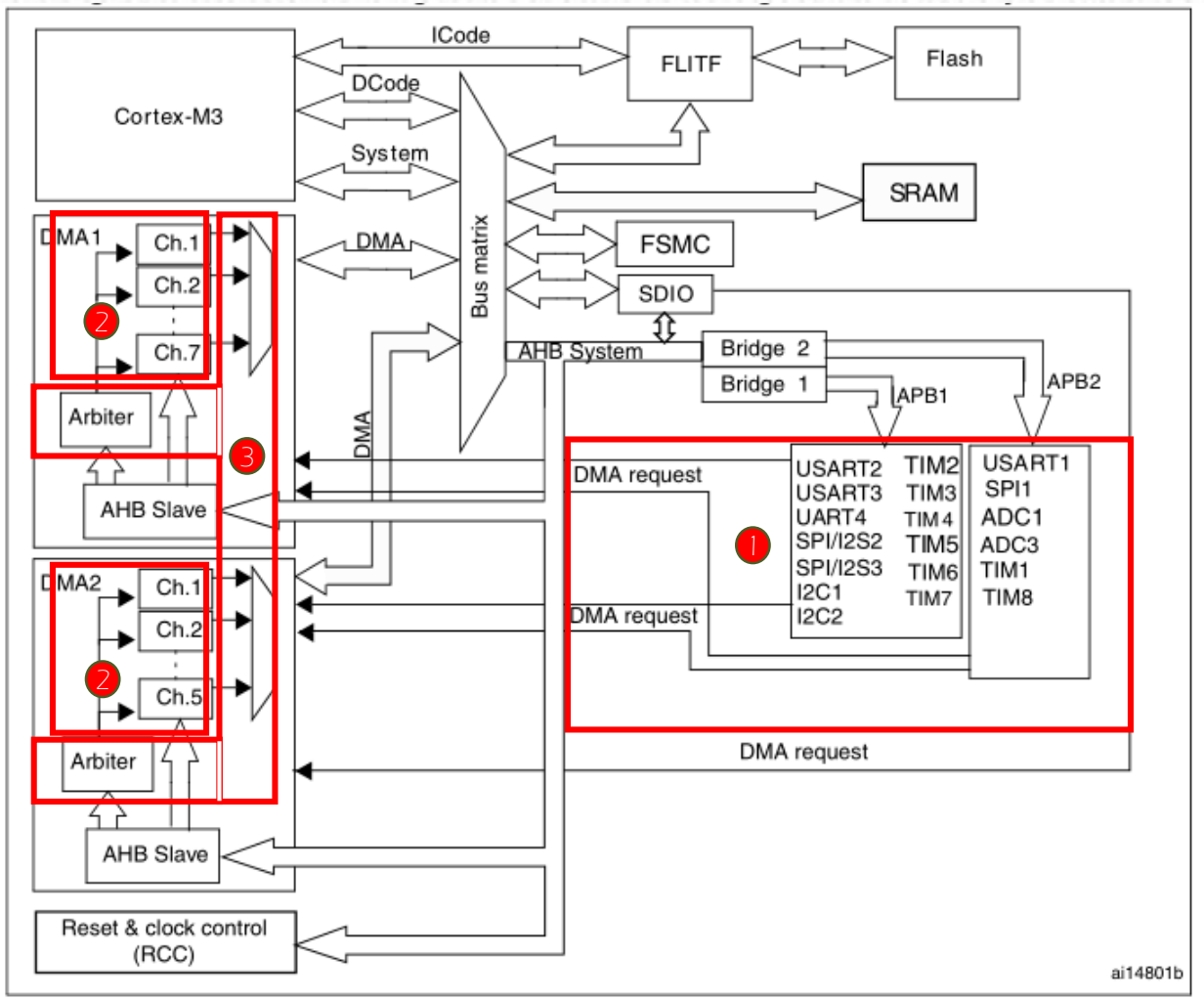
- Two DMA controllers in STM32F103
- 12 channels in total (7 for DMA1 and 5 for DMA2)
- Each channel is connected to dedicated to managing memory access requests from one or more peripherals.
- It has an arbiter for handling the priority between DMA requests.
- Priorities between requests from channels of one DMA are software programmable (4 levels consisting of very high, high, medium, low) or hardware in case of equality (request 1 has priority over request 2, etc.)
- Independent source and destination transfer size (byte, half word, word), emulating packing and unpacking. Source/destination addresses must be aligned on the data size.

1. DMA Principle Description
-- DMA of STM32F103 (continued)

- Support for circular buffer management
- 3 event flags (DMA Half Transfer, DMA Transfer complete and DMA Transfer Error) logically ORed together in a single interrupt request for each channel
- 3 directions: peripheral device -> memory，memory -> peripheral device ，memory -> memory
- Access to Flash, SRAM, APB1, APB2 and AHB peripherals as source and destination
- Programmable number of data to be transferred: up to 65536

1. DMA Principle Description -- DMA diagram

- ① DMA request
- ② DMA channel
- ③ DMA priorities
- The DMA2 controller is available only in high-density and XL-density devices.
- ADC3, SPI/I2S3, UART4, SDIO, TIM5, TIM6, DAC, TIM7, TIM8 DMA requests are available only in high-density devices



1. DMA Principle Description -- DMA transaction

- After an event, the peripheral sends a request signal to DMA Controller.
- DMA controller serves the request depending on the channel priorities.
- DMA Controller accesses the peripheral, and sends an Acknowledge.
- The peripheral releases its request as soon as it gets the Acknowledge from the DMA Controller.
- DMA Controller release the Acknowledge.
- If there are more requests, the peripheral can initiate the next transaction.

1. DMA Principle Description -- Arbiter

- The arbiter manages the channel requests based on their priority and launches the peripheral/memory access sequences.
- The priorities are managed in two stages:
  - Software: each channel priority can be configured in the DMA_CCRx register. There are four levels: very high priority, high priority, medium priority, low priority.
  - Hardware: if 2 requests have the same software priority level, the channel with the lowest number will get priority versus the channel with the highest number. For example, channel 2 gets priority over channel 4.
- Note: In high-density, XL-density and connectivity line devices, the DMA1 controller has priority over the DMA2 controller.

1. DMA Principle Description
-- DMA channels

- Each channel can handle DMA transfer between a peripheral register located at a fixed address and a memory address.
- The amount of data to be transferred (up to 65535) is programmable.
- The register which contains the amount of data items to be transferred is decremented after each transaction.
- Transfer data sizes of the peripheral and memory are fully programmable through the PSIZE and MSIZE bits in the DMA_CCRx register.
- Peripheral and memory pointers can optionally be automatically post-incremented after each transaction depending on the PINC and MINC bits in the DMA_CCRx register.
- If incremented mode is enabled, the address of the next transfer will be the address of the revious one incremented by 1, 2 or 4 depending on the chosen data size.

1. DMA Principle Description -- DMA channels (continued)

- The first transfer address is the one programmed in the DMA_CPARx/DMA_CMARx registers.
- During transfer operations, these registers keep the initially programmed value. The current Peripheral and memory pointers can optionally be automatically post-incremented after each transaction depending on the PINC and MINC bits in the DMA_CCRx register.
- If incremented mode is enabled, the address of the next transfer will be the address of the previous one incremented by 1, 2 or 4 depending on the chosen data size. The first transfer address is the one programmed in the DMA_CPARx/DMA_CMARx registers. During transfer operations, these registers keep the initially programmed value. The current transfer addresses (in the current internal peripheral/memory address register) are not accessible by software.

1. DMA Principle Description
-- Channel configuration procedure

- How to configure a DMA channelx (where x is the channel number).
- 1. Set peripheral register address in DMA_CPARx. The data will be moved from/ to this address to/ from the memory after the peripheral event.
- 2. Set the memory address in DMA_CMARx register. The data will be written to or read from this memory after the peripheral event.
- 3. Configure the total number of data in the DMA_CNDTRx register. After each peripheral event, this value will be decremented.
- 4. Configure the channel priority using the PL[1:0] bits in DMA_CCRx.
- 5. Configure data transfer direction, circular mode, peripheral & memory incremented mode, peripheral & memory data size, and interrupt after half and/or full transfer in the DMA_CCRx register.
- 6. Activate the channel by setting the ENABLE bit in DMA_CCRx register.
- As soon as the channel is enabled, it can serve any DMA request from the peripheral connected on the channel.

1. DMA Principle Description
-- DMA Interrupts

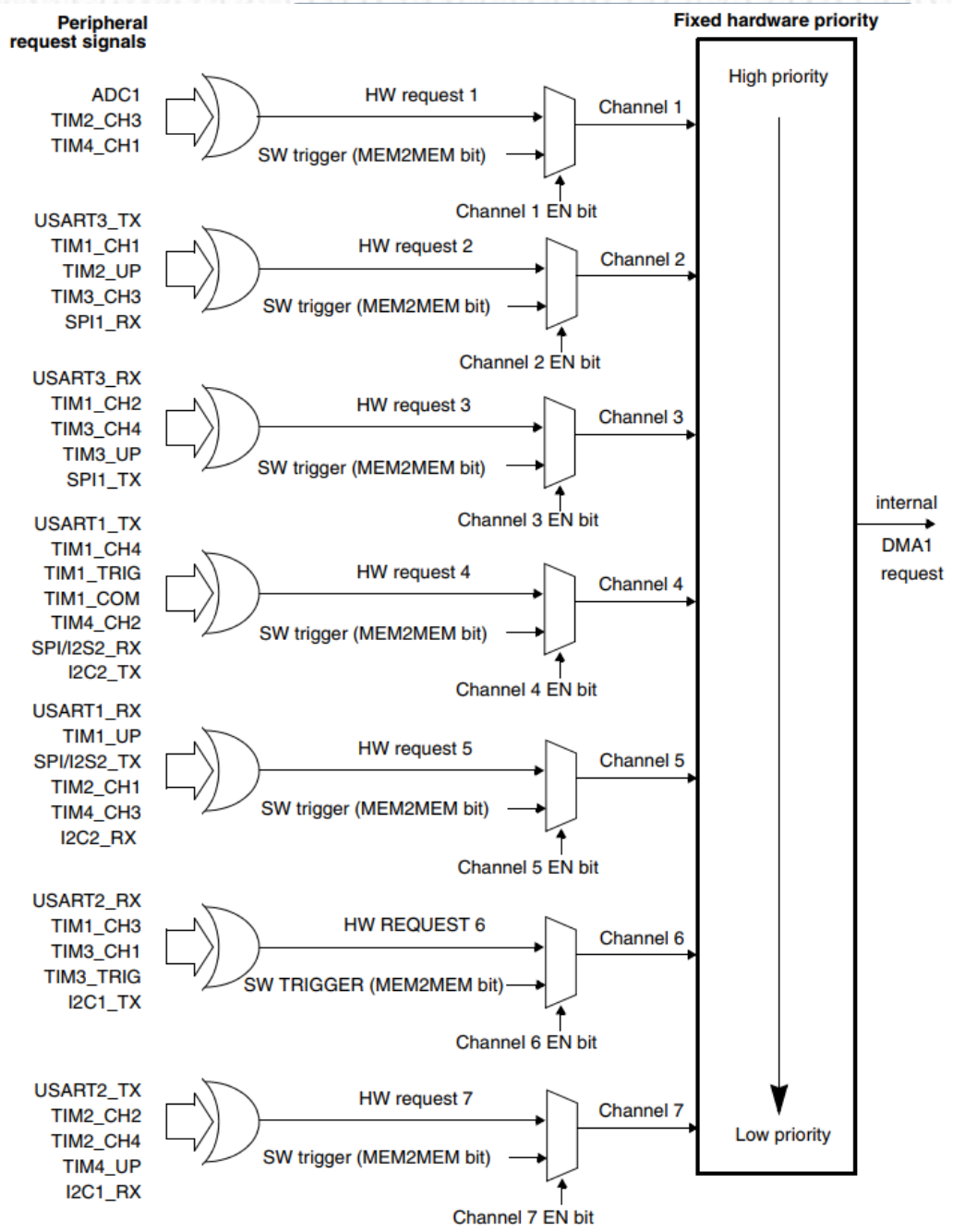
- An interrupt can be produced on a Half-transfer, Transfer complete or Transfer error for each DMA channel.
- Separate interrupt enable bits are available for flexibility.

Interrupt event	Event flag	Enable Control bit
Half-transfer	HTIF	HTIE
Transfer complete	TCIF	TCIE
Transfer error	TEIF	TEIE

1. DMA Principle Description -- DMA1 requests for channels

- The 7 requests from the peripherals (TIMx[1,2,3,4], ADC1, SPI1, SPI/I2S2, I2Cx[1,2] and USARTx[1,2,3]) are simply logically ORed before entering the DMA1, this means that only one request must be enabled at a time.

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
ADC1	ADC1	-	-	-	-	-	-
SPI/I^2S	-	SPI1_RX	SPI1_TX	SPI2/I2S2_RX	SPI2/I2S2_TX	-	-
USART	-	USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
I^2C	-	-	-	I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX
TIM1		TIM1_CH1	-	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
TIM2	TIM2_CH3	TIM2_UP	-	-	TIM2_CH1	-	TIM2_CH2 TIM2_CH4
TIM3	-	TIM3_CH3	TIM3_CH4 TIM3_UP	-	-	TIM3_CH1 TIM3_TRIG	-
TIM4	TIM4_CH1	-	-	TIM4_CH2	TIM4_CH3	-	TIM4_UP



1. DMA Principle Description

-- DMA configuration steps

- Enable DMA clock
- Initialize DMA
- Enable DMA transmission, and start DMA transfer
- Query DMA transmission status
- Take use of DMA interrupts

02

DMA Registers

2. DMA Registers

Register	Full name	Function
DMA_CCRx	DMA channel x configuration register	Used to configure DMA (core control register)
DMA_ISR	DMA interrupt status register	Used to query the current DMA transmission status
DMA_IFCR	DMA interrupt flag clear register	Used to clear DMA_ ISR flag bit
DMA_CNDTRx	DMA channel x number of data register	Used to control the amount of data transmitted by DMA channel x each time
DMA_CPARx	DMA channel x peripheral address register	Used to store STM32 peripheral address
DMA_CMARx	DMA channel x memory address register	Used to store STM32 memory address
USART_CR3	USART controller register 3	Used to enable DMA transmission of usart

2. DMA Registers
-- DMA_CCRx

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	MEM2MEM	PL[1:0]		MSIZE[1:0]		PSIZE[1:0]		MINC	PINC	CIRC	DIR	TEIE	HTIE	TCIE	EN
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:15 Reserved, must be kept at reset value.

Bit 14 **MEM2MEM:** Memory to memory mode
This bit is set and cleared by software.
0: Memory to memory mode disabled
1: Memory to memory mode enabled

Bits 13:12 **PL[1:0]:** Channel priority level
These bits are set and cleared by software.
00: Low
01: Medium
10: High
11: Very high

Bits 11:10 **MSIZE[1:0]:** Memory size
These bits are set and cleared by software.
00: 8-bits
01: 16-bits
10: 32-bits
11: Reserved

Bits 9:8 **PSIZE[1:0]:** Peripheral size
These bits are set and cleared by software.
00: 8-bits
01: 16-bits
10: 32-bits
11: Reserved

Bit 7 **MINC:** Memory increment mode
This bit is set and cleared by software.
0: Memory increment mode disabled
1: Memory increment mode enabled

Bit 6 **PINC:** Peripheral increment mode
This bit is set and cleared by software.
0: Peripheral increment mode disabled
1: Peripheral increment mode enabled

Bit 5 **CIRC:** Circular mode
This bit is set and cleared by software.
0: Circular mode disabled
1: Circular mode enabled

Bit 4 **DIR:** Data transfer direction
This bit is set and cleared by software.
0: Read from peripheral
1: Read from memory

Bit 3 **TEIE:** Transfer error interrupt enable
This bit is set and cleared by software.
0: TE interrupt disabled
1: TE interrupt enabled

Bit 2 **HTIE:** Half transfer interrupt enable
This bit is set and cleared by software.
0: HT interrupt disabled
1: HT interrupt enabled

Bit 1 **TCIE:** Transfer complete interrupt enable
This bit is set and cleared by software.
0: TC interrupt disabled
1: TC interrupt enabled

Bit 0 **EN:** Channel enable
This bit is set and cleared by software.
0: Channel disabled
1: Channel enabled

2. DMA Registers
-- DMA_ISR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				TEIF7	HTIF7	TCIF7	GIF7	TEIF6	HTIF6	TCIF6	GIF6	TEIF5	HTIF5	TCIF5	GIF5
				r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TEIF4	HTIF4	TCIF4	GIF4	TEIF3	HTIF3	TCIF3	GIF3	TEIF2	HTIF2	TCIF2	GIF2	TEIF1	HTIF1	TCIF1	GIF1
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:28 Reserved, must be kept at reset value.

Bits 27, 23, 19, 15, 11, 7, 3 **TEIFx:** Channel x transfer error flag (x = 1 ..7)
This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in the DMA_IFCR register.
0: No transfer error (TE) on channel x
1: A transfer error (TE) occurred on channel x

Bits 26, 22, 18, 14, 10, 6, 2 **HTIFx:** Channel x half transfer flag (x = 1 ..7)
This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in the DMA_IFCR register.
0: No half transfer (HT) event on channel x
1: A half transfer (HT) event occurred on channel x

Bits 25, 21, 17, 13, 9, 5, 1 **TCIFx:** Channel x transfer complete flag (x = 1 ..7)
This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in the DMA_IFCR register.
0: No transfer complete (TC) event on channel x
1: A transfer complete (TC) event occurred on channel x

Bits 24, 20, 16, 12, 8, 4, 0 **GIFx:** Channel x global interrupt flag (x = 1 ..7)
This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in the DMA_IFCR register.
0: No TE, HT or TC event on channel x
1: A TE, HT or TC event occurred on channel x

Error occurs

Half-transfer

Transfer complete

2. DMA Registers

-- DMA_IFCR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				CTEIF7	CHTIF7	CTCIF7	CGIF7	CTEIF6	*CHTIF6*	CTCIF6	CGIF6	CTEIF5	CHTIF5	CTCIF5	CGIF5
				w	w	w	w	w	w	w	w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTEIF4	CHTIF4	CTCIF4	CGIF4	CTEIF3	CHTIF3	CTCIF3	CGIF3	CTEIF2	CHTIF2	CTCIF2	CGIF2	CTEIF1	CHTIF1	CTCIF1	CGIF1
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:28 Reserved, must be kept at reset value.

Bits 27, 23, 19, 15, 11, 7, 3 **CTEIFx:** Channel x transfer error clear (x = 1 ..7)
This bit is set and cleared by software.
0: No effect
1: Clears the corresponding TEIF flag in the DMA_ISR register

Bits 26, 22, 18, 14, 10, 6, 2 **CHTIFx:** Channel x half transfer clear (x = 1 ..7)
This bit is set and cleared by software.
0: No effect
1: Clears the corresponding HTIF flag in the DMA_ISR register

Bits 25, 21, 17, 13, 9, 5, 1 **CTCIFx:** Channel x transfer complete clear (x = 1 ..7)
This bit is set and cleared by software.
0: No effect
1: Clears the corresponding TCIF flag in the DMA_ISR register

Bits 24, 20, 16, 12, 8, 4, 0 **CGIFx:** Channel x global interrupt clear (x = 1 ..7)
This bit is set and cleared by software.
0: No effect
1: Clears the GIF, TEIF, HTIF and TCIF flags in the DMA_ISR register

2. DMA Registers

-- DMA_CNDTRx

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NDT															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **NDT[15:0]:** Number of data to transfer

Number of data to be transferred (0 up to 65535). This register can only be written when the channel is disabled. Once the channel is enabled, this register is read-only, indicating the remaining bytes to be transmitted. This register decrements after each DMA transfer.

Once the transfer is completed, this register can either stay at zero or be reloaded automatically by the value previously programmed if the channel is configured in auto-reload mode.

If this register is zero, no transaction can be served whether the channel is enabled or not.

2. DMA Registers

-- DMA_CPARx

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PA																															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **PA[31:0]:** Peripheral address

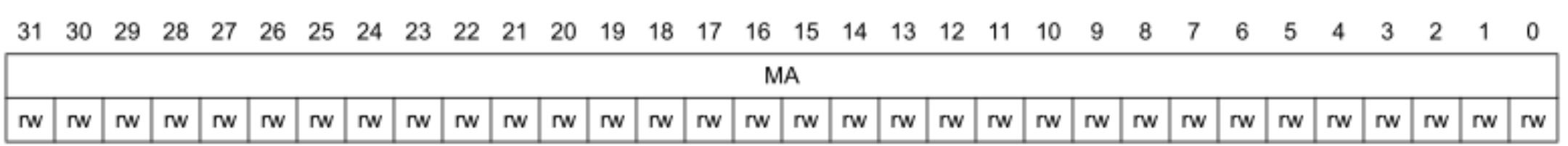
Base address of the peripheral data register from/to which the data will be read/written.

When PSIZE is 01 (16-bit), the PA[0] bit is ignored. Access is automatically aligned to a half-word address.

When PSIZE is 10 (32-bit), PA[1:0] are ignored. Access is automatically aligned to a word address.

2. DMA Registers

-- DMA_CMARx



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MA																															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **MA[31:0]:** Memory address

Base address of the memory area from/to which the data will be read/written.

When MSIZE is 01 (16-bit), the MA[0] bit is ignored. Access is automatically aligned to a half-word address.

When MSIZE is 10 (32-bit), MA[1:0] are ignored. Access is automatically aligned to a word address.

03

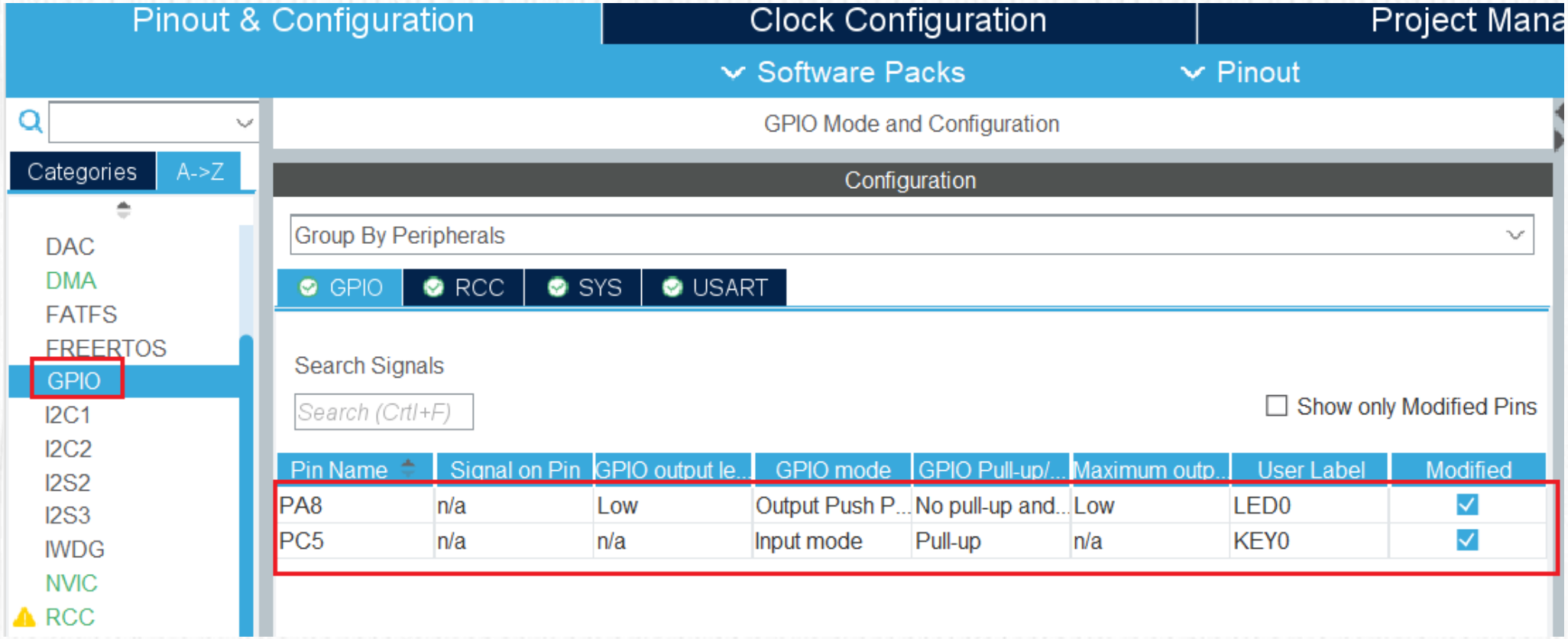
How to Program

3. How to Program

- Our Goal
  - Using GPIO KEY0 to control DMA transmission
  - Each time KEY0 is pressed, data is transferred to USART1 through DMA
  - DMA data transfer direction: memory -> peripheral devices
  - Display the current transfer progress on TFTLCD

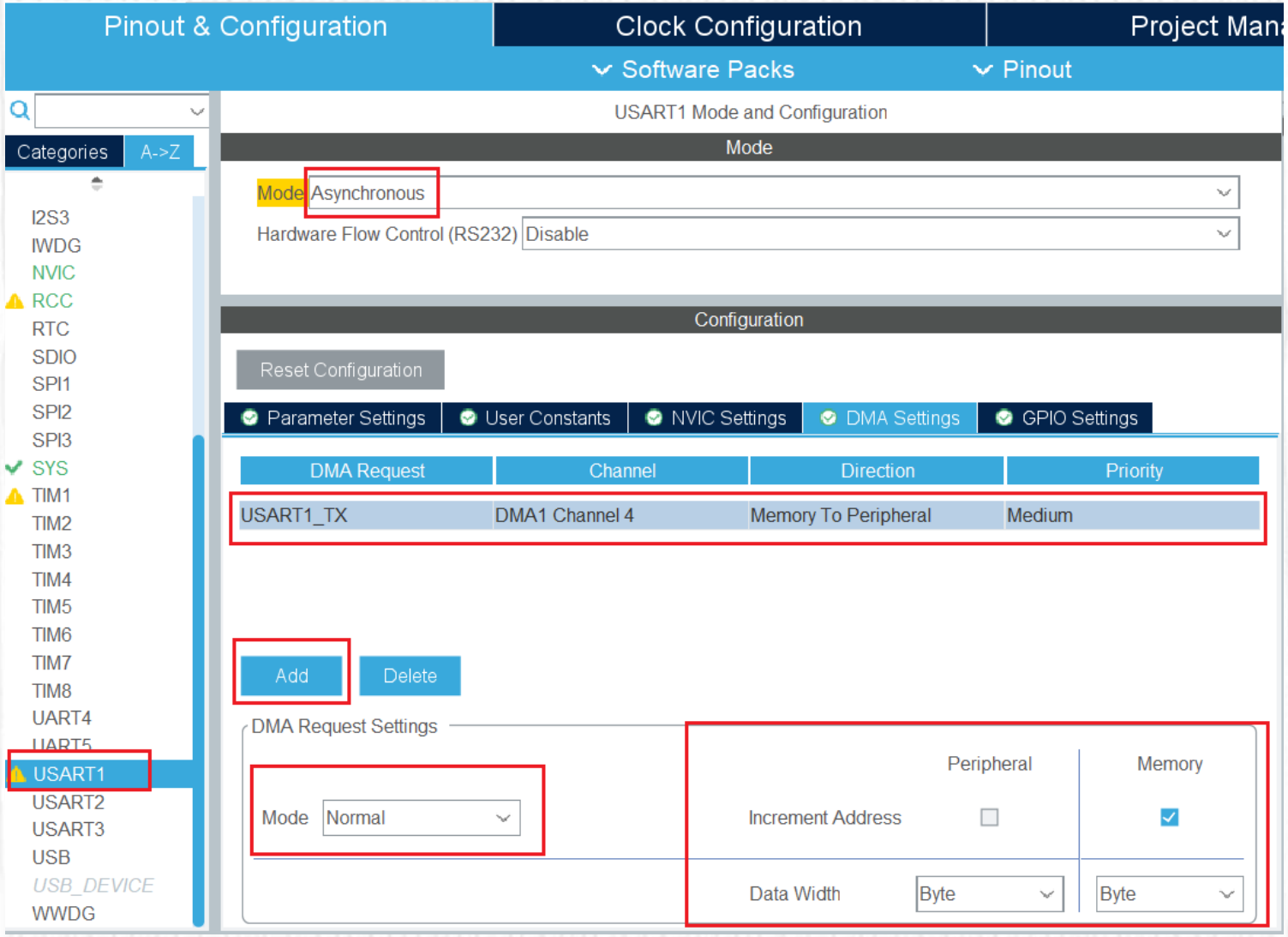
3. How to Program

- Configure GPIO
  - Set KEY0, LED0



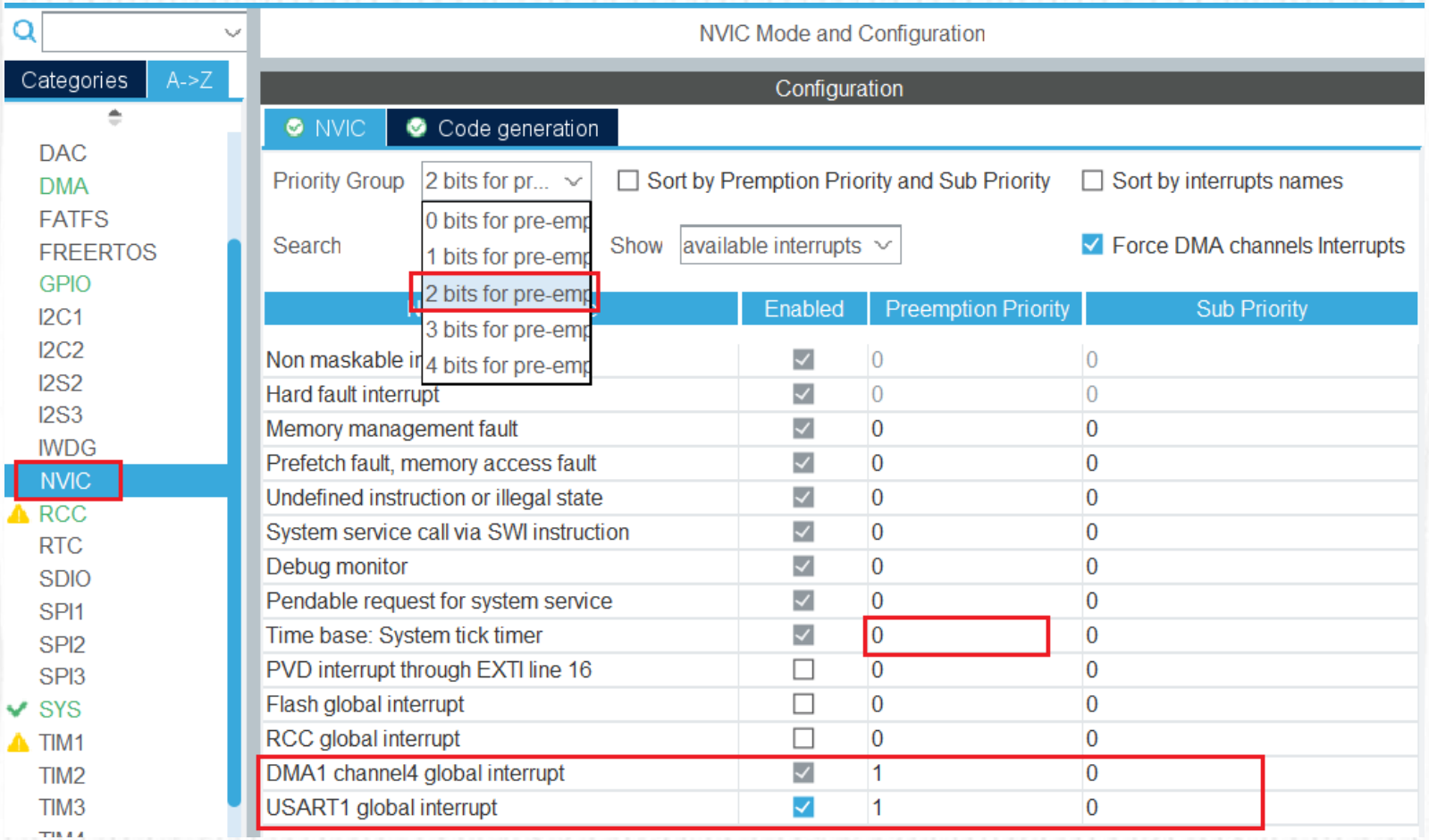
3. How to Program

- Configure Usart1
  - Enable Usart1
  - Add USART1_TX



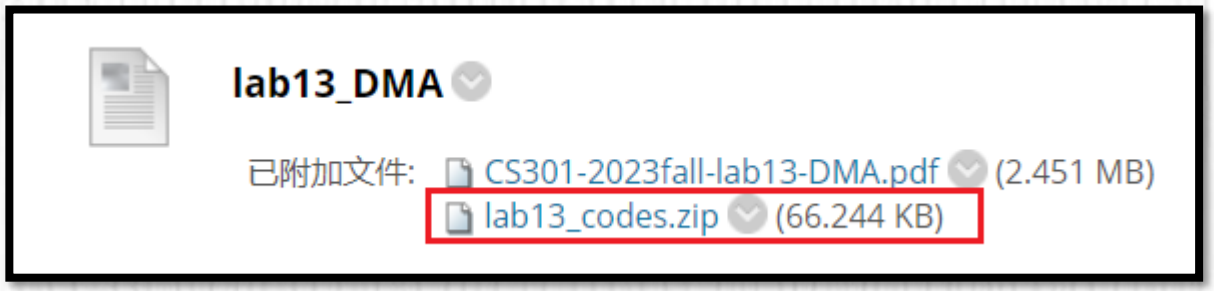
3. How to Program

- Configure NVIC
  - Enable Usart1 interrupt



3. How to Program

- Download lab13_codes.zip from Blackboard.
- Put **.c files in Src folder, and **.h files in Inc folder.



3. How to program

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Structures that used
  - DMA_HandleTypeDef
  - DMA_InitTypeDef

```
typedef struct{
    uint32_t Direction;
    uint32_t PeriphInc;
    uint32_t MemInc;
    uint32_t PeriphDataAlignment;
    uint32_t MemDataAlignment;
    uint32_t Mode;
    uint32_t Priority;
}DMA_InitTypeDef;
```

3. How to program

- DMA initialization (in stm32f1xx_hal_msp.c)

```
void HAL_UART_MspInit(UART_HandleTypeDef* huart){
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(huart->Instance==USART1){
    __HAL_RCC_USART1_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    /**USART1 GPIO Configuration:PA9     ------> USART1_TX; PA10     ------> USART1_RX*/
    …… ……
    /* USART1 DMA Init */    /* USART1_TX Init */
    hdma_usart1_tx.Instance = DMA1_Channel4;                    /* USART1_TX使用的DMA通道为: DMA1_Channel4 */
    hdma_usart1_tx.Init.Direction = DMA_MEMORY_TO_PERIPH;       /* DIR = 1 , 存储器到外设模式 */
    hdma_usart1_tx.Init.PeriphInc = DMA_PINC_DISABLE;           /* 外设非增量模式 */
    hdma_usart1_tx.Init.MemInc = DMA_MINC_ENABLE;               /* 存储器增量模式 */
    hdma_usart1_tx.Init.PeriphDataAlignment = DMA_PDATAALIGN_BYTE; /* 外设数据长度:8位 */
    hdma_usart1_tx.Init.MemDataAlignment = DMA_MDATAALIGN_BYTE;  /* 存储器数据长度:8位 */
    hdma_usart1_tx.Init.Mode = DMA_NORMAL;                      /* DMA模式:正常模式 */
    hdma_usart1_tx.Init.Priority = DMA_PRIORITY_MEDIUM;         /* 中等优先级 */
    if (HAL_DMA_Init(&hdma_usart1_tx) != HAL_OK){Error_Handler();}
    __HAL_LINKDMA(huart,hdmatx,hdma_usart1_tx); /* 将DMA与USART1联系起来(发送DMA) */
    HAL_NVIC_SetPriority(USART1_IRQn, 1, 0);
    HAL_NVIC_EnableIRQ(USART1_IRQn);
  }
}
```

3. How to program

- HAL functions

HAL API	Related registers	Function
__HAL_RCC_DMAx_CLK_ENABLE(…)	RCC_AHBENR	Enable DMAx clock
HAL_DMA_Init(…)	DMA_CCR	Initialize the DMA
HAL_DMA_Start_IT(…)	DMA_CCR/CPAR/CMAR/CNDTR	Start the DMA Transfer
__HAL_LINKDMA(…)		Connect DMA and peripheral handles
HAL_UART_Transmit_DMA(…)	CCR/CPAR/CMAR/CNDTR/USART_CR3	Sends an amount of data in DMA mode
__HAL_DMA_GET_FLAG(…)	DMA_ISR	Get the DMA Channel pending flags
__HAL_DMA_ENABLE(…)	DMA_CCR(EN)	Enable the specified DMA Channel
__HAL_DMA_DISABLE(…)	DMA_CCR(EN)	Disable the specified DMA Channel

3. How to program

- Main program

```
len = sizeof(TEXT_TO_SEND);
…… ……//copy TEXT_TO_SEND to g_sendbuf, and g_sendbuf is the string to transmit
while (1) {
    if (HAL_GPIO_ReadPin(KEY0_GPIO_Port, KEY0_Pin) == GPIO_PIN_RESET) {      /* KEY0按下 */
      HAL_UART_Transmit_DMA(&huart1, g_sendbuf, SEND_BUF_SIZE);
      /* 等待DMA传输完成，此时CPU空闲，可以处理其他任务；实际应用中，传输数据期间，可以执行另外的任务 */
      while (1){
              if ( __HAL_DMA_GET_FLAG(&hdma_usart1_tx, DMA_FLAG_TC4)) {/*等待DMA1_Channel4传输完成 */
                    __HAL_DMA_CLEAR_FLAG(&hdma_usart1_tx, DMA_FLAG_TC4); /* 清除传输完成标志 */
                    HAL_UART_DMAStop(&huart1);       /* 传输完成以后关闭串口DMA */
                    break;
              }
              pro = __HAL_DMA_GET_COUNTER(&hdma_usart1_tx); /* 得到当前还剩余多少个数据 */
              len = SEND_BUF_SIZE;          /* 总长度 */
              pro = 1 - (pro / len);        /* 得到百分比 */
              pro *= 100;                   /* 扩大100倍 */
              LCD_ShowNum(30, 150, pro, 3, 16); /* 显示传输进度百分比*/
      }
}
```

3. How to program -- Results

XCOM V2.6

is late, and there havebeen fewer kits. ThunderClan needs morewarriors if it is to survive.""But the year is only just beginning,"Spottedleaf pointed out calmly. "There will bemore kits when greenleaf comes."The gray cat twitched her broad shoulders."Perhaps. But training our young to becomewarriors takes time. If ThunderClan is to defendits territory, it must have new warriors as soon aspossible.""Are you asking StarClan for answers?"meowed Spottedleaf gently, following Bluestar'sgaze and staring up at the swath of stars glitteringin the dark sky."It is at times like this we need the words ofancient warriors to help us. Has StarClan spokento you?" Bluestar asked."Not for some moons, Bluestar."Suddenly a shooting star blazed over thetreetops. Spottedleaf's tail twitched and the furalong her spine bristled.Bluestar's ears pricked but she remained silentas Spottedleaf continued to gaze upward.After a few moments, Spottedleaf lowered herhead and turned to Bluestar. "It was a messagefrom StarClan," she murmured. A distant lookcame into her eyes. "Fire alone can save our Clan.""Fire?" Bluestar echoed. "But fire is feared byall the Clans! How can it save us?"Spottedleaf shook her head. "I do not know,"she admitted. "But this is the message StarClanhas chosen to share with me."The ThunderClan leader fixed her clear blueeyes on the medicine cat. "You have never beenwrong before, Spottedleaf," she meowed. "IfStarClan has spoken, then it must be so. Fire willsave our Clan."

串口选择: COM3:USB-SERIAL CH34C
波特率: 115200
停止位: 1
数据位: 8
校验位: None
串口操作: 关闭串口
保存窗口 | 清除接收
16进制显示 | DTR | RTS | 自动保存 | 时间戳 1000 ms

单条发送 | 多条发送 | 协议传输 | 帮助

发送 | 清除发送

定时发送 周期: 1000 ms | 打开文件 | 发送文件 | 停止发送
16进制发送 | 发送新行 | 0%
【火爆全网】正点原子DS100手持示波器上市

www.openedv.com | S:0 | R:6083 | CTS=0 DSR=0 DCD=0 | 当前时间 20:19:17

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

04

Practice

4. Practice

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Run the demo on MiniSTM32 board.
- We have taken use of DMA1_Channel4 in the demo, please make another DMA channel available in your practice.
- We have provided a method to enable DMA1_Channel5 in following pages, you can refer to it, or you can also take use of any other DMA channels.
- In the next demo, we have used both DMA1_Channel4 and DMA1_Channel5. The data is transferred from peripheral device (Usart1) to memory by DMA1_Channel4, and then moved from memory to peripheral device (Usart1) via DMA1_Channel5.

4. Practice

- Take use of DMA1_Channel5.

DMA Request	Channel	Direction	Priority
USART1_RX	DMA1 Channel 5	Peripheral To Memory	Medium
USART1_TX	DMA1 Channel 4	Memory To Peripheral	Medium

```
void HAL_UART_MspInit(UART_HandleTypeDef* huart){
    …… ……
    /* USART1_RX Init*/
    hdma_usart1_rx.Instance = DMA1_Channel5;                    /* USART1_RX使用的DMA通道为: DMA1_Channel5 */
    hdma_usart1_rx.Init.Direction = DMA_PERIPH_TO_MEMORY;           /* DIR = 0 , 外设到存储器到模式 */
    hdma_usart1_rx.Init.PeriphInc = DMA_PINC_DISABLE;               /* 外设非增量模式 */
    hdma_usart1_rx.Init.MemInc = DMA_MINC_ENABLE;                   /* 存储器增量模式 */
    hdma_usart1_rx.Init.PeriphDataAlignment = DMA_PDATAALIGN_BYTE;  /* 外设数据长度:8位 */
    hdma_usart1_rx.Init.MemDataAlignment = DMA_MDATAALIGN_BYTE;     /* 存储器数据长度:8位 */
    hdma_usart1_rx.Init.Mode = DMA_NORMAL;                          /* DMA模式:正常模式 */
    hdma_usart1_rx.Init.Priority = DMA_PRIORITY_MEDIUM;             /* 中等优先级 */
    if (HAL_DMA_Init(&hdma_usart1_rx) != HAL_OK){
      Error_Handler();
    }
    __HAL_LINKDMA(huart,hdmarx,hdma_usart1_rx);} /* 将DMA与USART1联系起来*/
    …… ……
}
```

4. Practice

- Enable RXNE interrupt and IDLE interrupt of Usart1. (in main.c)
- Receive data through DMA.

```
static void MX_USART1_UART_Init(void)
{
  ……
  /* USER CODE BEGIN USART1_Init 2 */

  __HAL_UART_ENABLE_IT(&huart1, UART_IT_RXNE);
  __HAL_UART_ENABLE_IT(&huart1, UART_IT_IDLE);
  HAL_UART_Receive_DMA(&huart1,rx_buffer,BUFFER_SIZE);

  /* USER CODE END USART1_Init 2 */
}
```

4. Practice

- Add the codes into USART1_IRQHandler( ) function. (in stm32f1xx_it.c)

```
void USART1_IRQHandler(void){
  /* USER CODE BEGIN USART1_IRQn 0 */
  uint32_t tmp_flag = 0;
  uint32_t temp;
  /* USER CODE END USART1_IRQn 0 */
  HAL_UART_IRQHandler(&huart1);
  /* USER CODE BEGIN USART1_IRQn 1 */
  tmp_flag =__HAL_UART_GET_FLAG(&huart1,UART_FLAG_IDLE);
  if((tmp_flag != RESET)) {
          __HAL_UART_CLEAR_IDLEFLAG(&huart1);
          HAL_UART_DMAStop(&huart1);
          temp  = __HAL_DMA_GET_COUNTER(&hdma_usart1_rx);
          rx_len =  BUFFER_SIZE_1 - temp;
          recv_end_flag = 1;
  }
  __HAL_UART_ENABLE_IT(&huart1, UART_IT_RXNE);
  __HAL_UART_ENABLE_IT(&huart1, UART_IT_IDLE);
  HAL_UART_Receive_DMA(&huart1,(uint8_t *)rx_buffer,BUFFER_SIZE_1);
  /* USER CODE END USART1_IRQn 1 */
}
```

4. Practice

- Main program (in main.c)

```
int main(void){
        …… ……

        LCD_Init();
        LCD_ShowString (30, 50, 200, 16, 16, "CS301");
        LCD_ShowString (30, 70, 200, 16, 16, "DMA TEST");

        while (1){
                if(recv_end_flag){
                        recv_end_flag = 0;
                        HAL_UART_Transmit_DMA(&huart1, rx_buffer, BUFFER_SIZE);
                        LCD_ShowString (30, 110, 200, 16, 16, "The data is: ");
                        LCD_ShowString (30, 130, 200, 16, 16, rx_buffer);
                        memset(rx_buffer, 0, rx_len);
                        rx_len = 0;
                }
        HAL_UART_Receive_DMA(&huart1, rx_buffer,BUFFER_SIZE);
}
```

4. Practice

- Results