DMA

Copying data using 2 pointers and a for

```
#include <stm32f10x.h>

void copyContents(char *destP, char *srcP, uint16_t count)
{
        while(count > 0)
        {
                *destP = *srcP;
                destP++;
                srcP++;
                count--;
        }
}

int main()
{
        char source[10]="ABCDEDFHI";
        char dest[10];

        copyContents(dest, source, 10);

        while(1);
}
```

It takes around 12 machine cycles to copy a byte

Example

- A 640x480 picture is stored in the flash memory. Assuming that BPP is 24-bit and the CPU crystal is 72MHz, how long does it take to copy the picture from flash to memory?

Solution:

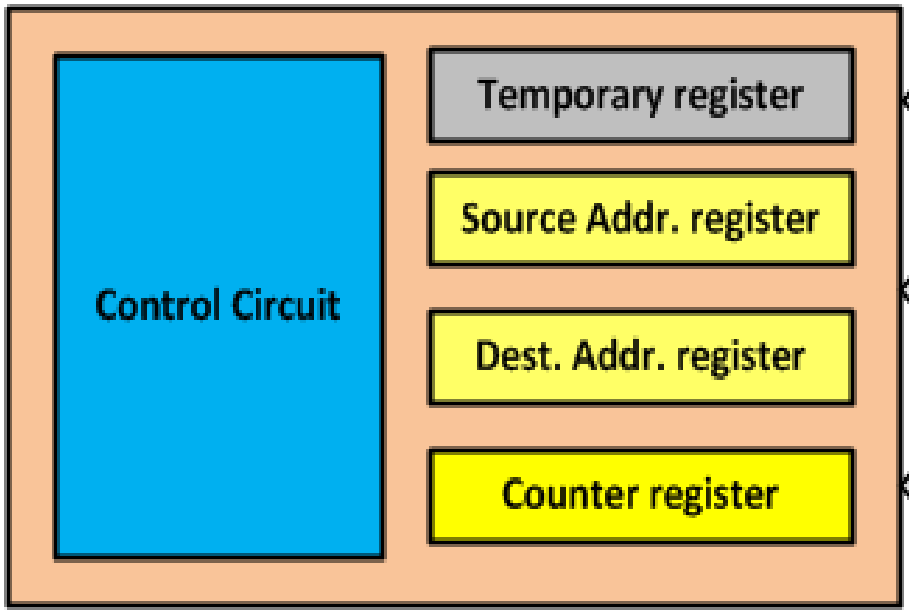
Copying each b

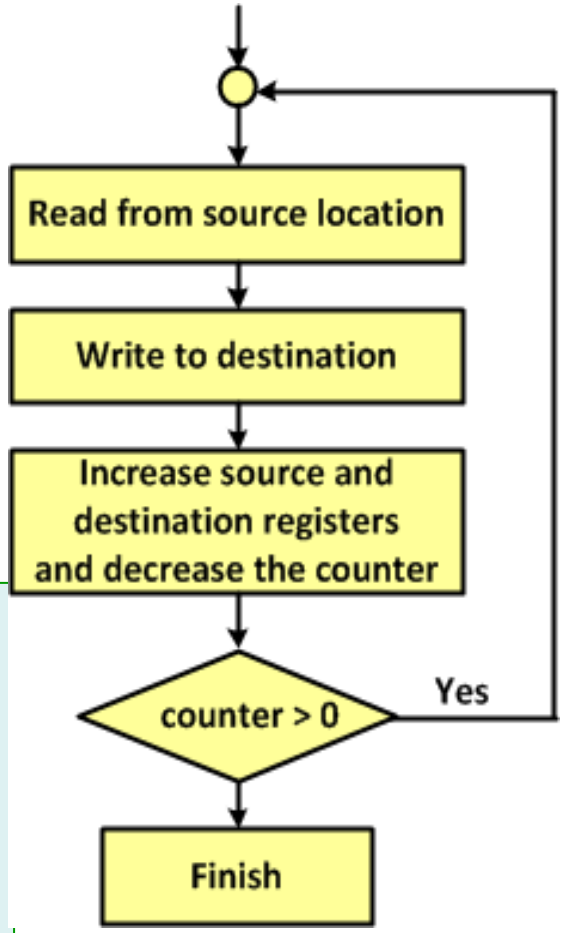
Number of byte

921600 × 166 ns = 152985600 ns = 0.152985600 seconds

- Too time consuming!
- No animation or movie
- CPU becomes busy

Copying using a simple hardware (A simplified DMA)





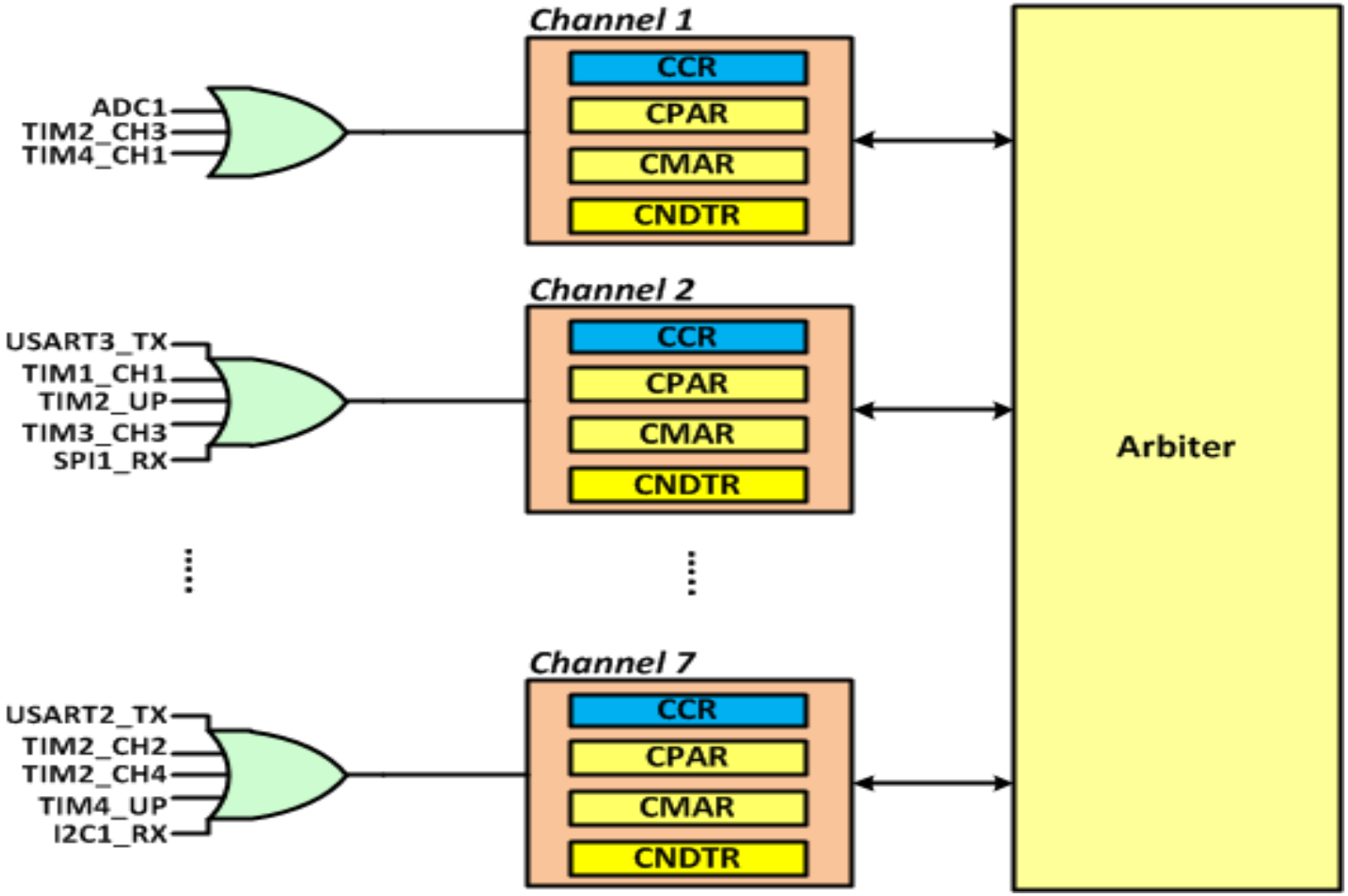
- The program initializes source, destination and counter registers. Then, the CPU sits idle until the data is copied.

In real DMA controllers

- There are more than 1 channel for copying
- Peripherals can ask the DMA to begin copying (they can send DMA request)

DMA1 in STM32F10x

- In STM32F10x, DMA1 has 7 channels



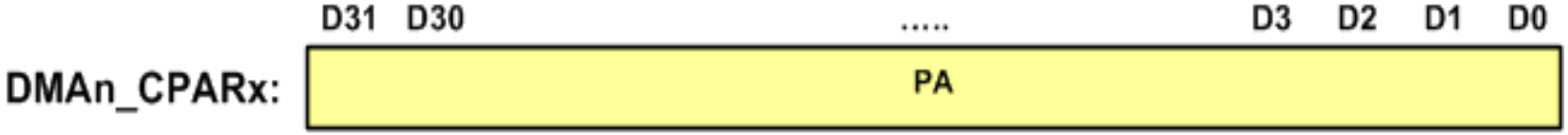
DMA2 in STM32F10x

- DMA2 has 5 channels

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5
ADC3(1)	-	-	-	-	ADC3
SPI/I2S3	SPI/I2S3_RX	SPI/I2S3_TX	-	-	-
UART4	-	-	UART4_RX	-	UART4_TX
SDIO(1)	-	-	-	SDIO	-
TIM5	TIM5_CH4 TIM5_TRIG	TIM5_CH3 TIM5_UP	-	TIM5_CH2	TIM5_CH1
TIM6/ DAC_Channel1	-	-	TIM6_UP/ DAC_Channel1	-	-
TIM7	-	-	-	TIM7_UP/ DAC_Channel2	-
TIM8	TIM8_CH3 TIM8_UP	TIM8_CH4 TIM8_TRIG TIM8_COM	TIM8_CH1	-	TIM8_CH2

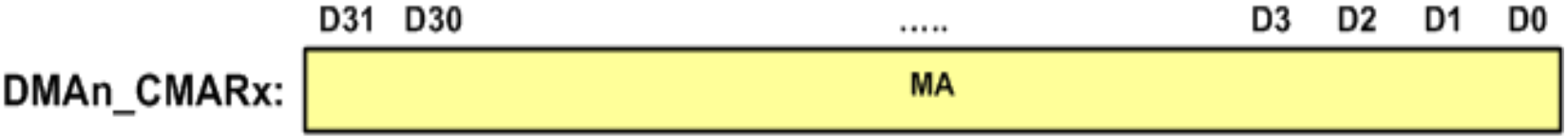
DMA Channel Registers

- CPAR (Channel Peripheral Address Register)



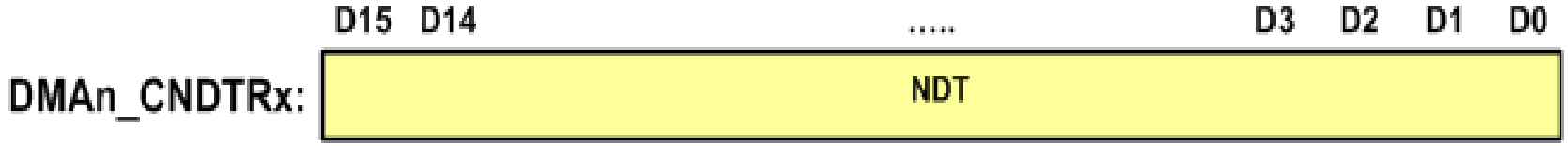
DMAn_Channelx->CPAR

- CMAR (Channel Memory Address Register)



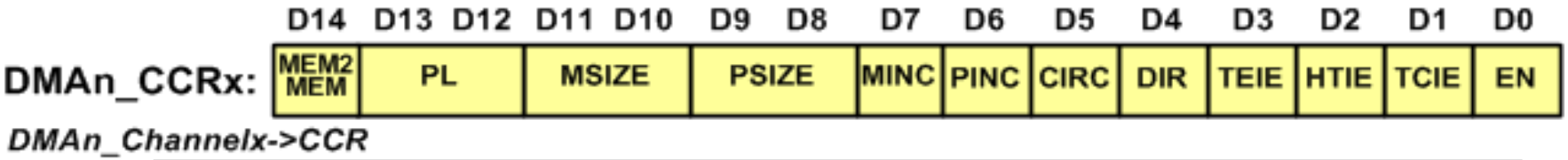
DMAn_Channelx->CMAR

- CNDTR (Channel Number of Data Register)



DMAn_Channelx->CNDTR

CCRn (Channel Control Register)



DMAn_Channelx->CCR

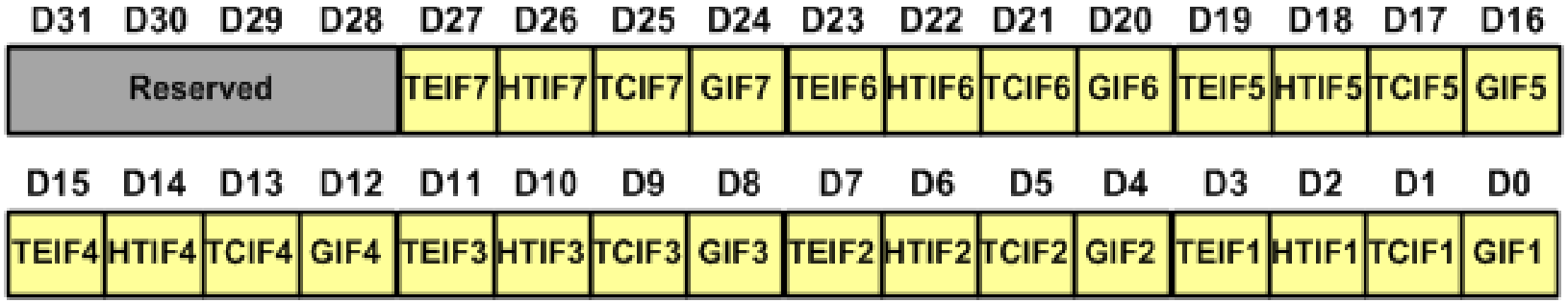
Field	Bit	Description
MEM2MEM	14	Memory to memory (0: Peripheral-memory, 1: Memory to memory) If the bit is set, the DMA channel copies data from source to destination until the counter becomes zero and transfer finishes. If the bit is zero, before each transfer, the channel waits to receive a DMA request from the peripherals and then it makes a transfer.
PL	13-12	Priority Level (00: Low, 01: Medium, 10: High, 11: very high) The bits are used to set the priority of the channel.
MSIZE	11-10	Memory size (00: 8-bit, 01: 16-bit, 10: 32-bit, 11: reserved)
PSIZE	9-8	Peripheral size (00: 8-bit, 01: 16-bit, 10: 32-bit, 11: reserved)

MSIZE	Mode	Read/write size	CMAR Increment	Alignment
00	8-bit	8-bit	+1	Byte
01	16-bit	16-bit	+2	Half-word
10	32-bit	32-bit	+4	word

		If the bit is 1, when the counter becomes zero, the CPAR, CMAR, and CNDTR are reloaded with their initial values and the channels transfers data from beginning. Otherwise, when the counter becomes zero, the channel stops.
DIR	4	Direction (0: copy from CPAR to CMAR, 1: copy from CMAR to CPAR)

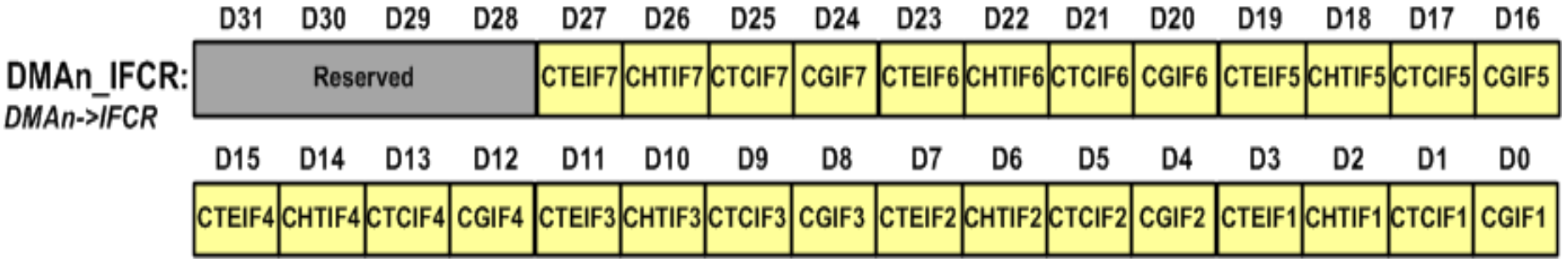
DMA_ISR (Interrupt Status Register)

DMAn_ISR:
DMAn->ISR



Field	Description
TEIFx	Channel x Transfer-Error flag: the bit is set when there is an error.
HTIFx	Channel x Half-transfer complete flag: It is set when half of data is transferred.
TCIFx	Channel x Transfer-Complete flag: It is set when the transfer is completed.
GIFx	Channel x Global flag: It is set when any flags of the channel is set.

DMA_IFCR (Interrupt Flag Clear Register)



DMA1->IFCR = (1<<5); /* clear TCIF2 (bit 5 of DMA_ISR) */

Writing Programs for DMAs

- Initialize CPAR and CMAR with source and destination addresses.
- Initialize CNDTR with the number of data to be copied. The value of CNDTR is decremented after each transfer and the channel stops transferring when the CNDTR becomes zero.
- Using the CCR register configure the channel priority, transfer direction, data size, memory and peripheral increment mode, circular mode, and the interrupts.
- Enable the channel by setting the Enable bit of the CCR register.

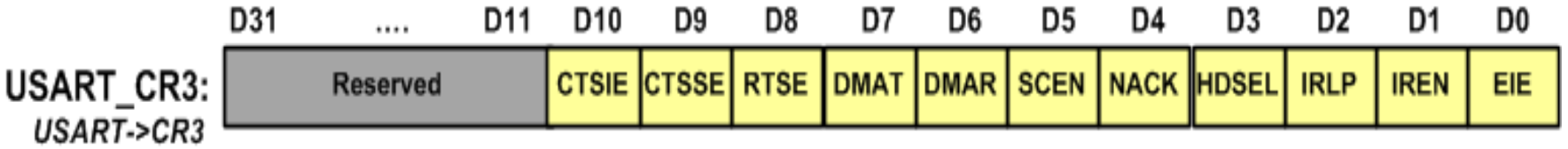
Program 2: Copying data from memory to memory

```
#include <stm32f10x.h>

volatile char transmitComplete = 0;
int main( )
{
  char source[10]="ABCDEFGHI";
  char dest[10];

  RCC->AHBENR = (1<<0); /* DMA clock enable (RCC_AHBENR_DMA1EN) */
  DMA1_Channel1->CPAR = (uint32_t) source; /* CPAR = the addr. of source
array */
  DMA1_Channel1->CMAR = (uint32_t) dest; /* CMAR = the addr. of dest
array */
  DMA1_Channel1->CNDTR = 10;  /* number of bytes to be copied = 10 */
  DMA1_Channel1->CCR = (1<<14)|(1<<7)|(1<<6)|(1<<1); /* mem2mem=1, mem
inc.=1, per. inc.=1, TCIE=1 */
  DMA1_Channel1->CCR |= 1; /* enable the channel */
  NVIC_EnableIRQ(DMA1_Channel1_IRQn); /* enable the int. for channel 1 */
  while(transmitComplete == 0); /* wait until the trans. is complete */
  while(1);
}
void DMA1_Channel1_IRQHandler(void)  {
  if((DMA1->ISR&(1<<1)) != 0) { /* is transmit complete */
    DMA1->IFCR = 1<<1; /* clear the TCIF flag */
```

Program 3: Copying data from memory to memory

```
#include <stm32f10x.h>

volatile char transmitComplete = 0;

int main( ) {
  long source[10]={0x11111111,0x22222222,0x33333333,
  0x44444444,0x55555555,0x66666666,0x77777777,
  0x88888888,0x99999999,0xaaaaaaaa};
  long dest[10];

  RCC->AHBENR = (1<<0); /* DMA clock enable */

  DMA1_Channel1->CPAR = (uint32_t) source; /* CPAR = the addr. of source
array */
  DMA1_Channel1->CMAR = (uint32_t) dest; /* CMAR = the addr. of dest
array */
  DMA1_Channel1->CNDTR = 10;  /* number of bytes to be copied = 10 */
  DMA1_Channel1->CCR = (1<<14)|(1<<11)|(1<<9)|(1<<7)|(1<<6)|(1<<1); /*
mem2mem=1,MSIZE=10 (32-bit), PSIZE=10 (32-bit), MInc=1, PInc=1, TCIE=1 */
  DMA1_Channel1->CCR |= 1; /* enable the channel */
  NVIC_EnableIRQ(DMA1_Channel1_IRQn); /* enable the int. for channel1 */

  while(transmitComplete == 0); /* wait until the transmit complete */
```

Using DMA for USART



Field	Bit	Description
DMAT	7	DMA enable Transmitter 0: Disable the DMA for USART transmit 1: Enable the DMA for USART transmit
DMAR	6	DMA enable receiver 0: the DMA for USART receive is disabled 1: the DMA for USART receive is enabled

USART1 DMA Channels

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

Program 4: Sending data via USART1 using DMA

```
#include <stm32f10x.h>

int main()
{
  char ourMsg[]="ABCDEFGHIJKL";

  RCC->AHBENR = (1<<0); /* DMA1 clock enable */
  RCC->APB2ENR |= 0xFC | (1<<14); //enable GPIO clocks

  //USART1 init.
  GPIOA->ODR |= (1<<10);  /* pull-up PA10 */
  GPIOA->CRH = 0x444448B4; /* RX1=input with pull-up, TX1=alt. func. output */
  USART1->BRR = 7500;      /* 72MHz/9600bps = 7500 */
  USART1->CR1 = 0x200C; /* enable usart transmit and receive */
  USART1->CR3 = (1<<7); /* DMA Trans. enable */

  DMA1_Channel4->CPAR = (uint32_t) &USART1->DR; /* to USART1->DR */
  DMA1_Channel4->CMAR = (uint32_t) ourMsg; /* from ourMsg array in mem. */
  DMA1_Channel4->CNDTR = 12;       /* copy 12 bytes */
  DMA1_Channel4->CCR = (1<<7)|(1<<4);      /* mem inc., read from mem */
  DMA1_Channel4->CCR |= 1; /* enable channel 4 */

  while(1)
  {
  }
}
```