Lecture 13: Direct Memory Access

Seyed-Hosein Attarzadeh-Niaki

Based on Slides by ARM

Review

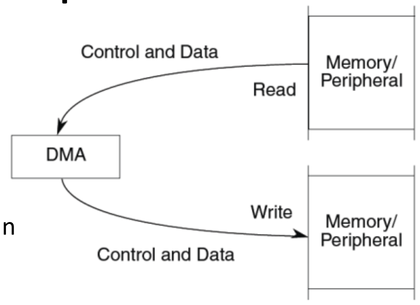
- ADC Basics
  - Concept
  - Characteristics
- Converting between analog and digital values
- STM32F4 analog interfacing peripherals
  - Analog-to-digital converter
  - Analog watchdog
  - Digital-to-analog converter
  - DAC using PWM

Outline

- Direct Memory Access
  - Channels
  - Transfer modes
  - Registers
  - Configuration

Basic Concepts

- Special hardware to *read data from a source and write it to a destination*
- Various configurable options
  - Number of data items to copy
  - Source and destination addresses can be fixed or change (e.g. increment, non-increment)
  - Size of data item
  - When transfer starts
- Operation
  - **Initialization**: Configure controller
  - **Transfer**: Data is copied
  - **Termination**: Channel indicates transfer has completed
  - **Post-transfer operation** (assert an interrupt)



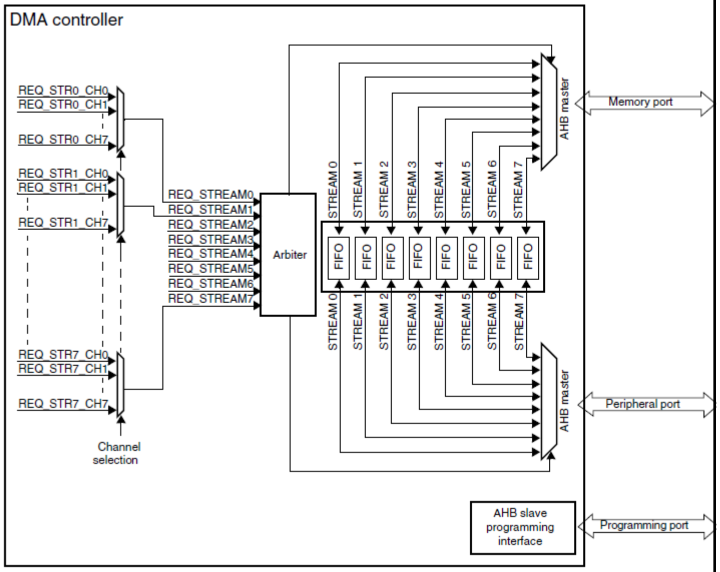
DMA Controller Features

- Two DMA controllers
- 16 streams in total (8 for each controller)
- Up to 8 channels (request) per stream
  - Each channel is responsible for specific peripheral or memory requests
- Arbiter handles the priority between DMA requests
  - 4 levels of software programmable priority
- 4 separate 32 first-in, first-out memory buffers (FIFOs) per stream

DMA Controller

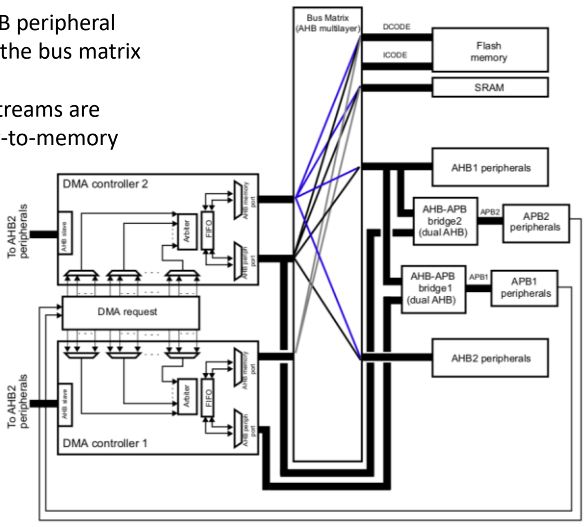
Streams are concurrent

Each stream can be only triggered by one of 8 channels at a time



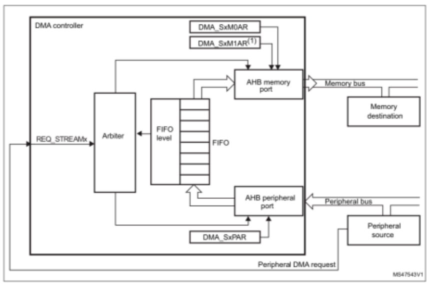
System implementation of the two DMA controllers (STM32F401xB/C and STM32F401xD/E)

- The DMA1 controller AHB peripheral port is not connected to the bus matrix like DMA2 controller.
- As a result, only DMA2 streams are able to perform memory-to-memory transfers.

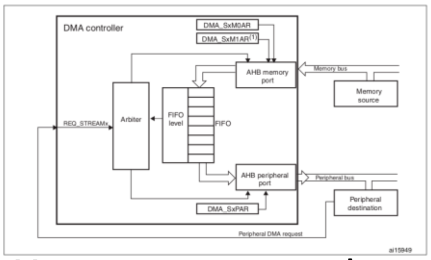


Transfer Modes

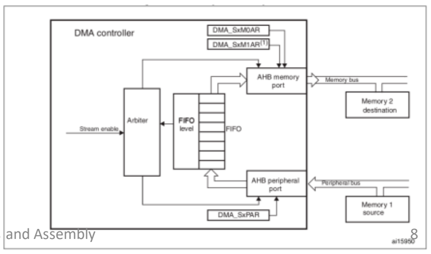
Peripheral-to-memory mode



Memory-to-peripheral mode



Memory-to-memory mode

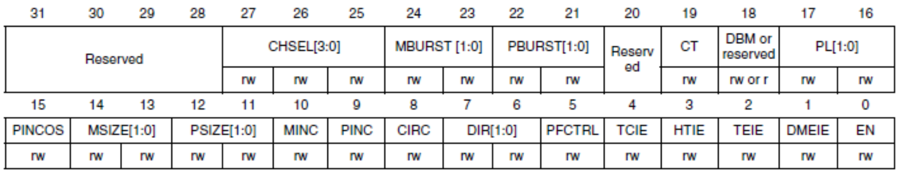


Bits DIR[1:0] of the DMA_SxCR register	Direction	Source address	Destination address
00	Peripheral-to-memory	DMA_SxPAR	DMA_SxM0AR
01	Memory-to-peripheral	DMA_SxM0AR	DMA_SxPAR
10	Memory-to-memory	DMA_SxPAR	DMA_SxM0AR
11	reserved	-	-

Registers

- DMA_SxCR
  - Stream x configuration register
- DMA_SxPAR
  - 32-bit Stream x Peripheral address register
- DMA_SxM0AR
  - 32-bit Stream x Memory address register
- Both source and destination transfers can address the entire 4 GB area, at address comprised between 0x0000 0000 and 0xFFFF FFFF

DMA Stream x Configuration Register DMA_SxCR (x=0..7)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				CHSEL[3:0]			MBURST [1:0]		PBURST[1:0]		Reserved	CT	DBM or reserved	PL[1:0]	
				rw	rw	rw	rw	rw	rw	rw		rw	rw or r	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PINCOS	MSIZE[1:0]		PSIZE[1:0]		MINC	PINC	CIRC	DIR[1:0]		PFCTRL	TCIE	HTIE	TEIE	DMEIE	EN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Configure the concerned stream
  - **CHESEL**[2:0]: Channel selection, can be written if EN is cleared.
  - **PL**[1:0]: Priority level, with 11 being very high and 00 being low.
  - **MSIZE**[1:0]: Memory data size
  - **PSIZE**[1:0]: Peripheral data size
  - **DIR**[1:0]: Data transfer direction
    - 00: P to M; 01: M to P; 10: M to M; 11: reserved
  - **EN**: Stream enable/ flag stream ready when read low

DMA Stream x Configuration Register DMA_SxCR (x=0..7)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				CHSEL[3:0]			MBURST [1:0]		PBURST[1:0]		Reserved	CT	DBM or reserved	PL[1:0]	
				rw	rw	rw	rw	rw	rw	rw		rw	rw or r	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PINCOS	MSIZE[1:0]		PSIZE[1:0]		MINC	PINC	CIRC	DIR[1:0]		PFCTRL	TCIE	HTIE	TEIE	DMEIE	EN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Configure the concerned stream
  - **MSIZE**[1:0]; **PSIZE**[1:0] data size configuration
    - 00: byte; 01:half-word; 10:word; 11:reserved
  - **TCIE**: transfer complete interrupt enable
  - Other configuration includes:
    - Direct mode
    - Error interrupt
    - Circular mode
    - Burst transfer
    - Increment

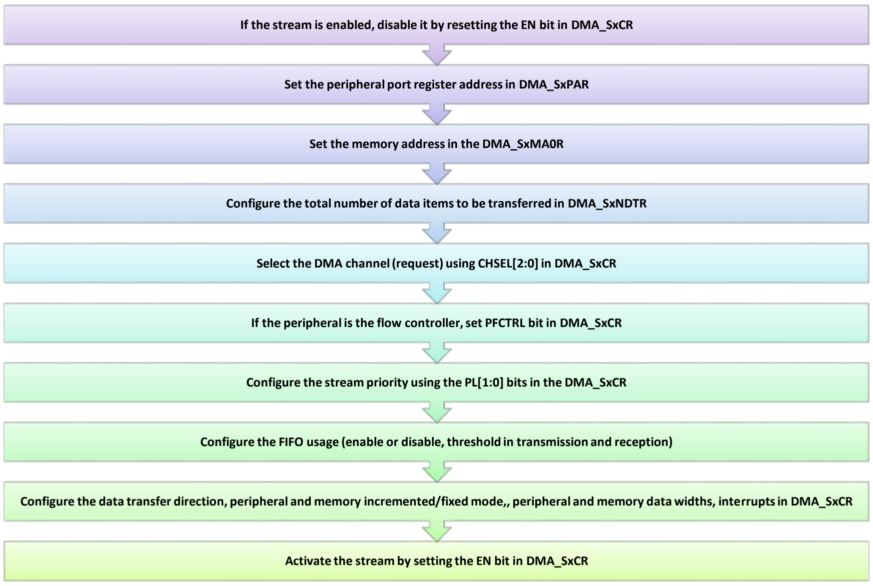
DMA stream x number of data register DMA_SxNDTR (x = 0..7)

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[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

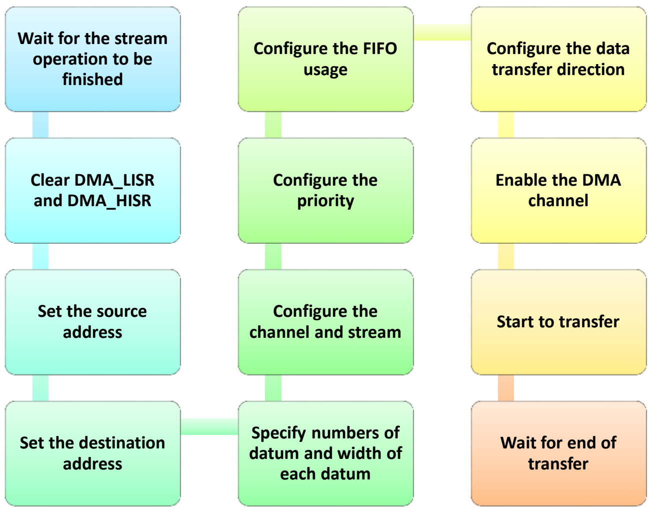
- NDT[15:0] Number of data items to transfer
  - 0 up to 65535
  - Writable only when the stream is disabled
  - When the stream is enabled, it is read-only, indicating the remaining data items to be transmitted
  - When the value is zero, no transaction will be served

General Stream Configuration Procedure



1. If the stream is enabled, disable it by resetting the EN bit in DMA_SxCR
2. Set the peripheral port register address in DMA_SxPAR
3. Set the memory address in the DMA_SxMA0R
4. Configure the total number of data items to be transferred in DMA_SxNDTR
5. Select the DMA channel (request) using CHSEL[2:0] in DMA_SxCR
6. If the peripheral is the flow controller, set PFCTRL bit in DMA_SxCR
7. Configure the stream priority using the PL[1:0] bits in the DMA_SxCR
8. Configure the FIFO usage (enable or disable, threshold in transmission and reception)
9. Configure the data transfer direction, peripheral and memory incremented/fixed mode,, peripheral and memory data widths, interrupts in DMA_SxCR
10. Activate the stream by setting the EN bit in DMA_SxCR

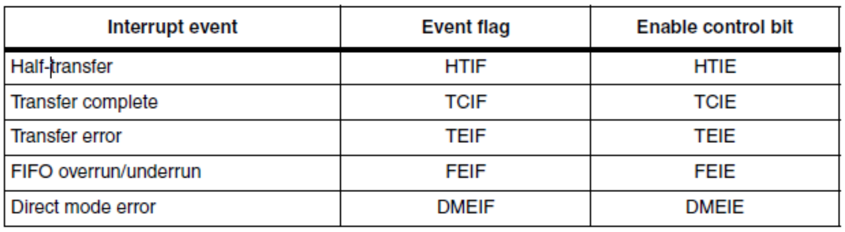
Basic Use of DMA



- Wait for the stream operation to be finished
- Clear DMA_LISR and DMA_HISR
- Set the source address
- Set the destination address
- Specify numbers of datum and width of each datum
- Configure the channel and stream
- Configure the priority
- Configure the FIFO usage
- Configure the data transfer direction
- Enable the DMA channel
- Start to transfer
- Wait for end of transfer

End of transfer

- It is possible to interrupt in case of the following events:



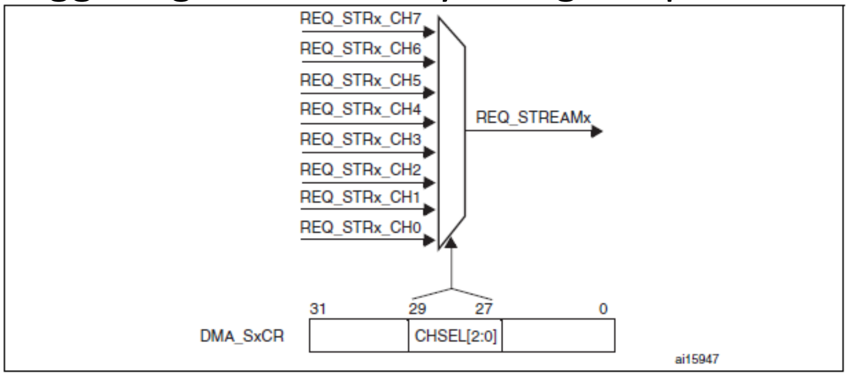
Interrupt event	Event flag	Enable control bit
Half-transfer	HTIF	HTIE
Transfer complete	TCIF	TCIE
Transfer error	TEIF	TEIE
FIFO overrun/underrun	FEIF	FEIE
Direct mode error	DMEIF	DMEIE

- If the interrupt is disabled, there are still two ways to check if the transfer has ended (in normal mode):
  - The EN bit of CR will be hardware cleared at the end of the transfer
  - The value of DMA_SxNDTR register will be 0, indicating no items to be transmitted

Recall:
DMA vs. ISR

- To communicate between CPU and peripherals, there are different approaches
  - Polling
  - Interrupt
  - DMA
  - Channel I/O
- Interrupt improves the performance of the CPU in many ways compared to polling.
  - However, the overhead of interrupt may scale up as the number of peripherals increases.
- DMA, on the other hand, takes care of the peripheral once the configuration is made by the CPU.
  - Exempt the CPU from being interrupt too frequently.

Triggering DMA Activity Using Peripherals



- In general cases, have to configure DMA and peripherals at the same time so that DMA can be triggered by a specific peripherals.
- Many peripherals have DMA supports (e.g. ADC), but it have to be at least enabled!
- Memory to peripheral or peripheral to memory mode.

DMA1 Requests

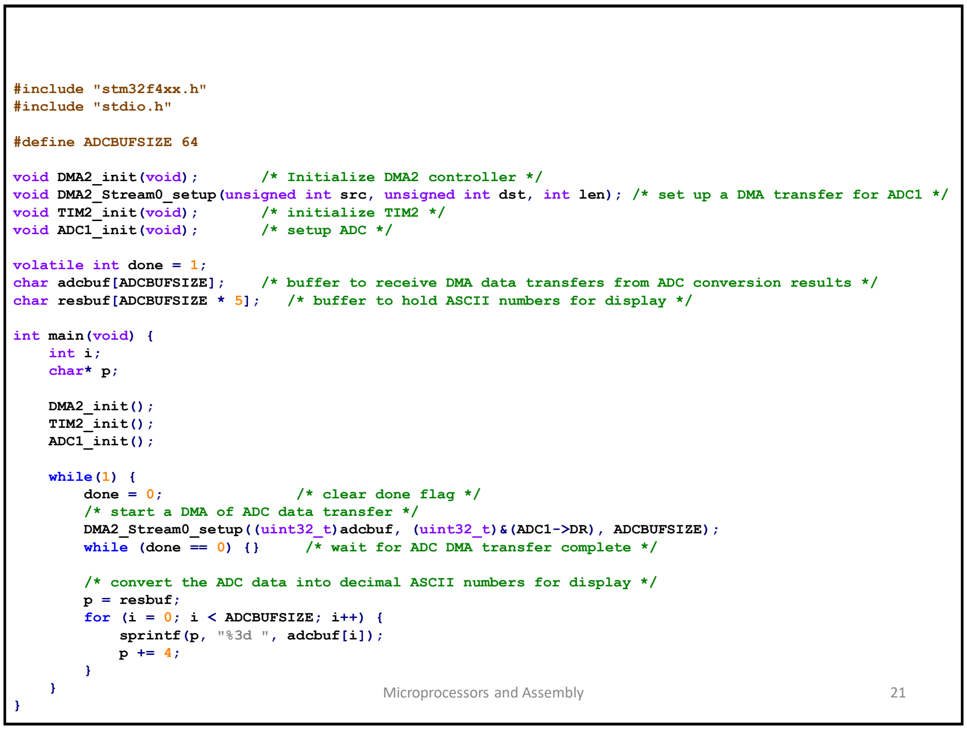
Peripheral Requests	S0	S1	S2	S3	S4	S5	S6	S7
C0	SPI3_RX		SPI3_RX	SPI2_RX	SPI2_TX	SPI3_TX		SPI3_TX
C1	I2C1_RX	I2C3_RX				I2C1_RX	I2C1_TX	I2C1_TX
C2	TIM4_CH1		I2S3_EXT_RX	TIM4_CH2	I2S2_EXT_TX	I2S3_EXT_TX	TIM4_UP	TIM4_CH3
C3	I2S3_EXT_RX	TIM2_UP TIM2_CH3	I2C3_RX	I2S2_EXT_RX	I2C3_TX	TIM2_CH1	TIM2_CH2 TIM2_CH4	TIM2_UP TIM2_CH4
C4						USART2_RX	USART2_TX	
C5			TIM3_CH4 TIM3_UP		TIM3_CH1 TIM3_TRIG	TIM3_CH2		TIM3_CH3
C6	TIM5_CH3 TIM5_UP	TIM5_CH4 TIM5_TRIG	TIM5_CH1	TIM5_CH4 TIM5_TRIG	TIM5_CH2	I2C3_TX	TIM5_UP	
C7			I2C2_RX	I2C2_RX				I2S2_TX

DMA2 Requests

Peripheral Requests	S0	S1	S2	S3	S4	S5	S6	S7
C0	ADC1				ADC1		TIM1_CH1 TIM1_CH2 TIM1_CH3	
C1								
C2								
C3	SPI1_RX		SPI1_RX	SPI1_TX		SPI1_TX		
C4	SPI4_RX	SPI4_TX	USART1_RX	SDIO		USART1_RX	SDIO	USART1_TX
C5		USART6_ RX	USART6_RX	SPI4_RX	SPI4_TX		USART6_TX	USART6_TX
C6	TIM1_TR IG	TIM1_CH 1	TIM1_CH2	TIM1_CH1	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
C7								

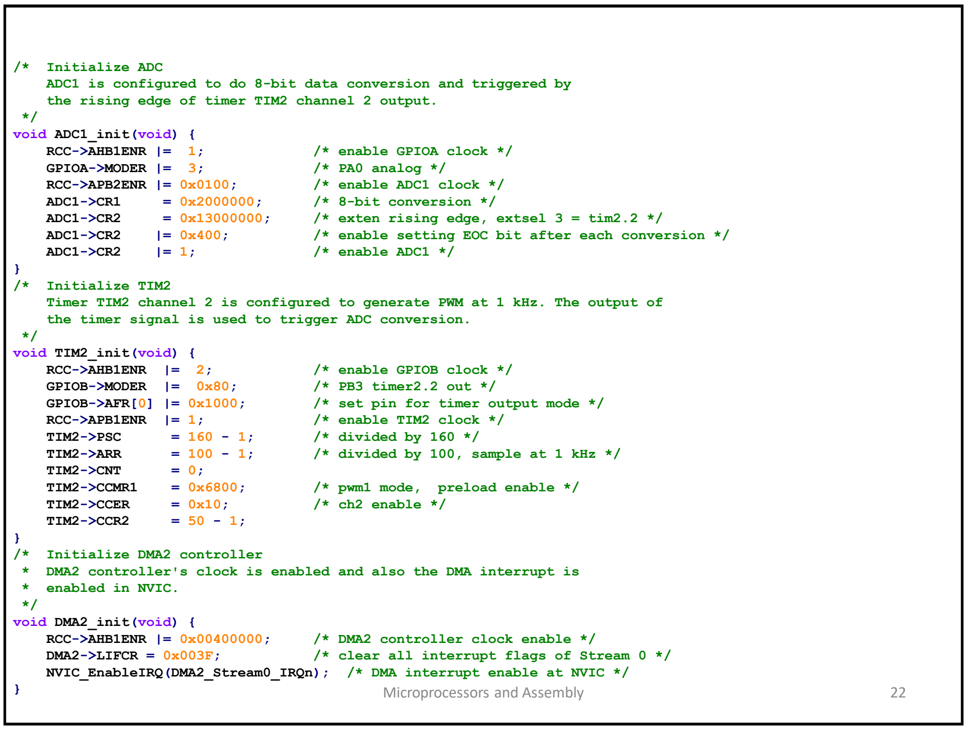
Example

- Set up the timer TIM2 channel 2 to trigger ADC1 to convert the analog input channel 0
- The output of the ADC1 is transferred to the buffer in memory by DMA
- Once the buffer if full, the DMA is stopped
- The data in the buffer is converted to decimal ASCII numbers
- A global variable, **`done`**, is used by the DMA transfer complete interrupt handler to signal the other parts of the program that a buffer full of data conversion is done.



```c
#include "stm32f4xx.h"
#include "stdio.h"

#define ADCBUFSIZE 64

void DMA2_init(void);       /* Initialize DMA2 controller */
void DMA2_Stream0_setup(unsigned int src, unsigned int dst, int len); /* set up a DMA transfer for ADC1 */
void TIM2_init(void);       /* initialize TIM2 */
void ADC1_init(void);       /* setup ADC */

volatile int done = 1;
char adcbuf[ADCBUFSIZE];    /* buffer to receive DMA data transfers from ADC conversion results */
char resbuf[ADCBUFSIZE * 5];   /* buffer to hold ASCII numbers for display */

int main(void) {
    int i;
    char* p;

    DMA2_init();
    TIM2_init();
    ADC1_init();

    while(1) {
        done = 0;                /* clear done flag */
        /* start a DMA of ADC data transfer */
        DMA2_Stream0_setup((uint32_t)adcbuf, (uint32_t)&(ADC1->DR), ADCBUFSIZE);
        while (done == 0) {}      /* wait for ADC DMA transfer complete */

        /* convert the ADC data into decimal ASCII numbers for display */
        p = resbuf;
        for (i = 0; i < ADCBUFSIZE; i++) {
            sprintf(p, "%3d ", adcbuf[i]);
            p += 4;
        }
    }
}
```

Microprocessors and Assembly 21



```c
/*  Initialize ADC
    ADC1 is configured to do 8-bit data conversion and triggered by
    the rising edge of timer TIM2 channel 2 output.
 */
void ADC1_init(void) {
    RCC->AHB1ENR |=  1;             /* enable GPIOA clock */
    GPIOA->MODER |=  3;             /* PA0 analog */
    RCC->APB2ENR |= 0x0100;         /* enable ADC1 clock */
    ADC1->CR1     = 0x2000000;      /* 8-bit conversion */
    ADC1->CR2     = 0x13000000;     /* exten rising edge, extsel 3 = tim2.2 */
    ADC1->CR2    |= 0x400;          /* enable setting EOC bit after each conversion */
    ADC1->CR2    |= 1;              /* enable ADC1 */
}
/*  Initialize TIM2
    Timer TIM2 channel 2 is configured to generate PWM at 1 kHz. The output of
    the timer signal is used to trigger ADC conversion.
 */
void TIM2_init(void) {
    RCC->AHB1ENR  |=  2;            /* enable GPIOB clock */
    GPIOB->MODER  |=  0x80;         /* PB3 timer2.2 out */
    GPIOB->AFR[0] |= 0x1000;        /* set pin for timer output mode */
    RCC->APB1ENR  |= 1;             /* enable TIM2 clock */
    TIM2->PSC      = 160 - 1;       /* divided by 160 */
    TIM2->ARR      = 100 - 1;       /* divided by 100, sample at 1 kHz */
    TIM2->CNT      = 0;
    TIM2->CCMR1    = 0x6800;        /* pwm1 mode,  preload enable */
    TIM2->CCER     = 0x10;          /* ch2 enable */
    TIM2->CCR2     = 50 - 1;
}
/*  Initialize DMA2 controller
 *  DMA2 controller's clock is enabled and also the DMA interrupt is
 *  enabled in NVIC.
 */
void DMA2_init(void) {
    RCC->AHB1ENR |= 0x00400000;     /* DMA2 controller clock enable */
    DMA2->LIFCR = 0x003F;           /* clear all interrupt flags of Stream 0 */
    NVIC_EnableIRQ(DMA2_Stream0_IRQn);  /* DMA interrupt enable at NVIC */
}
```

Microprocessors and Assembly 22

```
/*  Set up a DMA transfer for ADC
 *  The ADC1 is connected to DMA2 Stream 0. This function sets up the peripheral register address,
 *  memory address, number of transfers, data size, transfer direction, and DMA interrupts are enabled.
 *  At the end, the DMA controller is enabled, the ADC conversion complete is used to trigger DMA data
 *  transfer, and the timer used to trigger ADC is enabled.
 */
void DMA2_Stream0_setup(unsigned int src, unsigned int dst, int len) {
    DMA2_Stream0->CR &= ~1;           /* disable DMA2 Stream 0 */
    while (DMA2_Stream0->CR & 1) {} /* wait until DMA2 Stream 0 is disabled */
    DMA2->LIFCR = 0x003F;             /* clear all interrupt flags of Stream 0 */
    DMA2_Stream0->PAR    = dst;
    DMA2_Stream0->M0AR   = src;
    DMA2_Stream0->NDTR   = len;
    DMA2_Stream0->CR     = 0x00000000;  /* ADC1_0 on DMA2 Stream0 Channel 0 */
    DMA2_Stream0->CR    |= 0x00000400;  /* data size byte, mem incr, peripheral-to-mem */
    DMA2_Stream0->CR    |= 0x16;        /* enable interrupts DMA_IT_TC | DMA_IT_TE | DMA_IT_DME */
    DMA2_Stream0->S0FCR  = 0;           /* direct mode, no FIFO */
    DMA2_Stream0->CR    |= 1;           /* enable DMA2 Stream 0 */

    ADC1->CR2 |= 0x0100;                /* enable ADC conversion complete DMA data transfer */
    TIM2->CR1  = 1;                     /* enable timer2 */
}
/*  DMA2 Stream0 interrupt handler
    This function handles the interrupts from DMA2 controller Stream0. The error interrupts
    have a placeholder for error handling code. If the interrupt is from DMA data
    transfer complete, the DMA controller is disabled, the interrupt flags are
    cleared, the ADC conversion complete DMA trigger is turned off and the timer
    that triggers ADC conversion is turned off too.
 */
void DMA2_Stream0_IRQHandler(void)
{
    if (DMA2->LISR & 0x000C)        /* if an error occurred */
        while(1) {}                 /* substitute this by error handling */
    DMA2_Stream0->CR = 0;           /* disable DMA2 Stream 0 */
    DMA2->LIFCR = 0x003F;           /* clear DMA2 interrupt flags */
    ADC1->CR2 &= ~0x0100;           /* disable ADC conversion complete DMA */
    TIM2->CR1 &= ~1;                /* disable timer2 */
    done = 1;
}
```

Example: Flash to SRAM transfer

- Software-triggered by enabling the Channel
- Transfer word data buffer from Flash memory to embedded SRAM memory
- DMA2 Steam 0 Channel 0 is configured to transfer the 32-word data
- Only DMA2 Streams are able to perform memory to memory transfers
- Could be used as a fast version of memcpy function, but performed by DMA instead of CPU