Лабораторная работа № 7: Работа с DMA

Цель работы: исследовать возможности использования DMA, основные характеристики DMA, взаимодействие с другими устройствами в составе микроконтроллера.

Оборудование и программное обеспечение: отладочная плата STM32F4 Discovery, мультиметр, реда разработки CooCox CoIDE 1.7, Keil 4.xx, инструменты построения проекта GNU Toolchain for ARM Embedded Processors, библиотека CMSIS.

Теоретические сведения

Direct Memory Access (DMA, рос. “Прямой доступ к памяти”) – механизм, используемый в контроллерах ARM для перемещения данных между памятью и периферией без участия процессора. Ключевым моментом является, то что при использовании DMA на перемещение данных не используются ресурсы процессора, что может быть особенно критичным при создании приложений, работающих с большим количеством данных и активно использующих периферию. Работа DMA обеспечивается отдельным контроллером, который выполняет определенные действия по команде процессора.

Рассматриваемый контроллер имеет в своем распоряжении сразу два контроллера DMA, которые обеспечивают в общей сложности 16 потоков (по 8 потоков на каждый контроллер), каждый из которых предназначен для управления запросами к памяти от одного или нескольких устройств. Каждый поток может обеспечивать до 8 каналов (запросов). Каждый контроллер DMA имеет устройство разрешения конфликтов для обработки запросов в соответствии с их приоритетом.

Контроллер DMA позволяет производить запись данных в трех направлениях:

- от периферии в память;

- из памяти к периферии;

- из памяти в память.

Передача ведется либо в режиме непосредственной передачи, либо в режиме очереди. Поддерживается различный размер данных для передачи, причем размер данных для приемника и источника может быть неодинаковым. В таком случае DMA определяет данную ситуацию и выполняет необходимые действия для оптимизации передачи, однако эта возможность поддерживается только в режиме очереди.

Кроме того, очень полезной может оказаться функция циклической передачи, при использовании которой передача данных начинается с начального адреса снова после передачи последней единицы данных источника.

Пример программы

Рассмотрим пример программы, которая использует DMA для передачи данных из памяти в ЦАП. Изменение уровня сигнала на выходе ЦАП происходит циклически с частотой изменения в 1 с. Изменения происходят на основе значения, которое считывается из памяти по сигналу переполнения таймера. В результате этого ЦАП отправляет запрос к DMA и получает данные, которые записываются в регистр данных, что приводит к изменению уровня сигнала. Схема взаимодействия выглядит следующим образом (рис. 3.10):

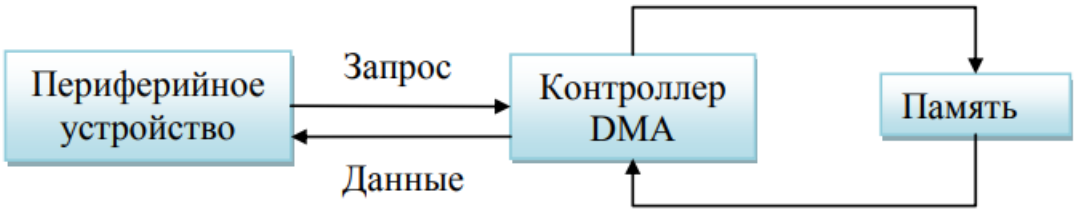

Рис. 3.10. Схема использования DMA

В самой программе следует обратить внимание на большое количество параметров, необходимых для настройки DMA по сравнению с другими устройствами. С этим связана и сложность использования DMA, поскольку необходимо учитывать большое количество возможных настроек. Тем не менее, многие из них достаточно просты (направление передачи, значения адресов), поэтому изучение DMA можно сопоставить по сложности с другими устройствами, в чем очень помогает библиотека Standard Peripheral Library

```
#include <stm32f4xx.h>
#include <stm32f4xx_rcc.h>
#include <stm32f4xx_dma.h>
#include <stm32f4xx_gpio.h>
#include <stm32f4xx_tim.h>
#include <stm32f4xx_dac.h>
uint8_t levels[] = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77};
void init(void);
void init_gpio(void);
void init_timer(void);
void init_dac(void);
void init_dma(void);
int main(void)
{
init();
 while(1)
 {
 }
}
void init(void) {
init_gpio();
init_timer();
init_dac();
```

```
init_dma();
}
void init_gpio(void) {
GPIO_InitTypeDef gpio_init;
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
gpio_init.GPIO_Mode = GPIO_Mode_AN;
gpio_init.GPIO_Pin = GPIO_Pin_4;
gpio_init.GPIO_OType = GPIO_OType_PP;
gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_Init(GPIOA, &gpio_init);
}
void init_timer(void) {
TIM_TimeBaseInitTypeDef tim_init;
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM6, ENABLE);
tim_init.TIM_CounterMode = TIM_CounterMode_Up;
tim_init.TIM_Period = 16000 - 1;
tim_init.TIM_Prescaler = 1000 - 1;
TIM_TimeBaseInit(TIM6, &tim_init);
TIM_SelectOutputTrigger(TIM6, TIM_TRGOSource_Update);
TIM_Cmd(TIM6, ENABLE);
}
void init_dac(void) {
DAC_InitTypeDef dac_init;
RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
DAC_StructInit(&dac_init);
dac_init.DAC_Trigger = DAC_Trigger_T6_TRGO;
dac_init.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
dac_init.DAC_WaveGeneration = DAC_WaveGeneration_None;
DAC_Init(DAC_Channel_1, &dac_init);
}
void init_dma(void) {
```

```
DMA_InitTypeDef dma_init;
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA1, ENABLE);
DMA_DeInit(DMA1_Stream5);
dma_init.DMA_Channel = DMA_Channel_7;
dma_init.DMA_PeripheralBaseAddr = (uint32_t)(DAC_BASE + 0x10);
dma_init.DMA_Memory0BaseAddr = (uint32_t)&levels;
dma_init.DMA_DIR = DMA_DIR_MemoryToPeripheral;
dma_init.DMA_BufferSize = 8;
dma_init.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
dma_init.DMA_MemoryInc = DMA_MemoryInc_Enable;
dma_init.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
dma_init.DMA_MemoryDataSize = DMA_PeripheralDataSize_Byte;
dma_init.DMA_Mode = DMA_Mode_Circular;
dma_init.DMA_Priority = DMA_Priority_High;
dma_init.DMA_FIFOMode = DMA_FIFOMode_Disable;
dma_init.DMA_FIFOThreshold = DMA_FIFOThreshold_HalfFull;
dma_init.DMA_MemoryBurst = DMA_MemoryBurst_Single;
dma_init.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
DMA_Init(DMA1_Stream5, &dma_init);
DMA_Cmd(DMA1_Stream5, ENABLE);
DAC_Cmd(DAC_Channel_1, ENABLE);
DAC_DMACmd(DAC_Channel_1, ENABLE);
}
```

После компиляции программы-примера и прошивки отладочной платы следует наблюдать за напряжением на выходе PA4 с помощью мультиметра. Прибор должен показать ступенчатое изменение напряжение от 0 В до определенного значения, после чего цикл повторяется. Шаг увеличения одинаков для всего цикла, что можно увидеть по массиву значений для записи в регистр данных.

Ход работы

1. Проверить работу программы-примера, скомпилировав ее в одной из сред разработки и выполнив прошивку.

2. Ознакомиться с документацией по DMA для микроконтроллера на отладочной плате.

3. Попробовать изменить определенные параметры в программе-примере.

4. Организовать взаимодействие DMA и другого периферийного устройства в соответствии с индивидуальным заданием.

Индивидуальные задания

1. Реализовать асинхронную передачу данных через USART, считывая данные через определенные

промежутки времени посредством DMA.

2. Организовать прием данных через USART с записью в память через DMA. Количество передаваемых

данных известно заранее и равно размеру массива.

3. Продемонстрировать передачу данных через SPI из памяти в циклическом режиме.

4. Продемонстрировать прием данных через SPI с записью в память. Режим записи – циклический.

5. Реализовать запись в память значений, полученных с помощью работы АЦП через определенные

промежутки времени. После записи первых 50 значений начинается новый чикл записи.