JIEE Časopis priemyselnej elektrotechniky
Journal of Industrial Electrical Engineering
ISSN 2454-0900
www.jiee.eu

The DMA controller description and configuration of the STM32 microcontrollers

[1]*Patrik JACKO*, [2]*Dobroslav KOVÁČ*

[1,2] Department of Theoretical and Industrial Electrical Engineering, FEI TU of Košice, Slovak Republic

[1] patrik.jacko.2@tuke.sk, [2] dobroslav.kovac@tuke.sk

***Abstract* — One of the main problem which is resolving since first processors, computers or microcontrollers is speed of processing data, speed of transmitting data etc. – speed is a crucial element of processors technic. The direct memory access or DMA controller is nowadays one of most used controllers in fast processor electronics. It is used in the computers, microcontrollers for high-speed data transmission and it serves to descramble the processor of the transfer operations. The DMA controller provides quick read and write data from or into the memory. This article describes DMA controller used in the STM32 microcontrollers and its configuration.**

***Keywords* — DMA controller, SPI interface, STM32 microcontroller**

I. INTRODUCTION

The microcontrollers of STMicroelectronics are day to day more popular thanks its speed, reliability and cash affordability. There are a lot of families ST microcontrollers, but we will describe STM32F4 microcontroller family. This family includes a number of various processors too. Our focus will be microcontroller STM32F446RE and its peripherals with DMA (Direct Memory Access) controller. This article describes using of DMA for the communication interface and AD conversions which is very important to digital signal processing and analog values processing. We show tested measurement of SPI interface with DMA and there will be described source code for the correct configuration DMA controller.

II. THE DMA CONTROLLER

If I/O pins of the microcontroller are controlled directly by using an interrupt system, transmission speed between peripherals and memory is limited by processor speed. Of course, processor interrupts take some time too, therefore transmission can take longer time. Each elementary transfer requires several instructions. This requires some memory access that needs to be performed in addition to reading or writing the data. The data frame transmission from/to I/O pins is repeat single operation that is possible to perform using special hardware component – DMA. [1]

The DMA was developed for the devices required high-speed service, which processor is impossible ensure. The DMA controller usually serving group of channels. One channel means communication between memory and any peripheral device (AD converter, USART, SPI etc.) for example. Microprocessor before start the transmission sets address and distance of frames which will be transmitted by using DMA. There is a necessary set direction of transmitting data too (memory to memory, peripheral to memory, memory to peripheral or peripheral to peripheral). After the configurations, the DMA is ready for data transmission. [2]

The data that are transmitted through the DMA are completely bypassed by the processor. This advantage helps to CPU because the CPU can then do other operations. The process of data transmitting by using DMA is shown in the Fig. 1.

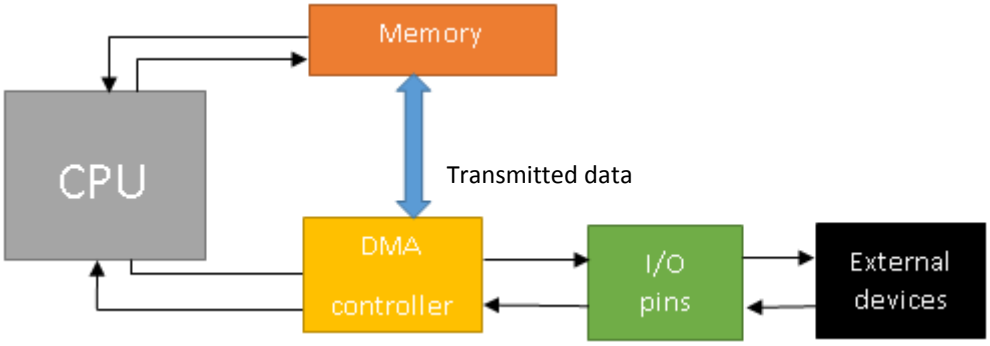

Fig. 1 The microcontroller blocking scheme with DMA

III. THE STM32F446RE MICROCONTROLLER AND ITS DMA CONTROLLER

The microcontroller STM32F446RE includes two general-purpose dual-port DMA controllers with 8 streams each. They are able to manage memory to memory, peripheral to memory and memory to peripheral transfers. The microcontroller STM32F446RE operates on the 180 MHz frequency, and all devices and peripherals in the microcontroller are connected to the AHB bus (Advanced High-performance Bus) - operates on 180 MHz, APB2 bus (Advanced Peripheral Bus) – operates on frequency 90 MHz and low speed APB1 operates on the 45 MHz. The both DMA controllers are connected to the AHB bus, which means operations on the 180 MHz.

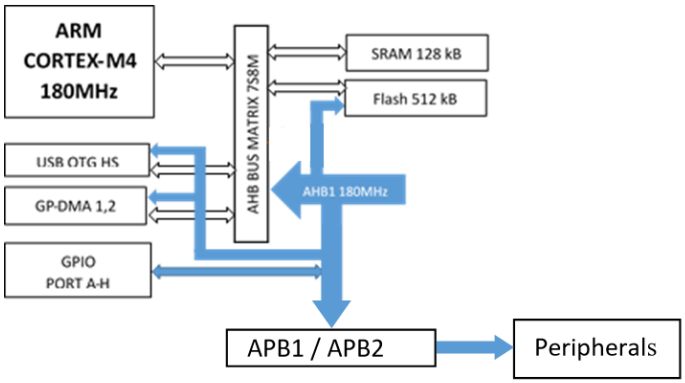

Fig. 2 The STM32F4 architecture with DMA controllers connected

We can see part of STM32F446RE architecture in the Fig. 2 where the DMA controllers are connected with primary AHB BUS Matrix and microcontroller's Flash and SRAM memories.

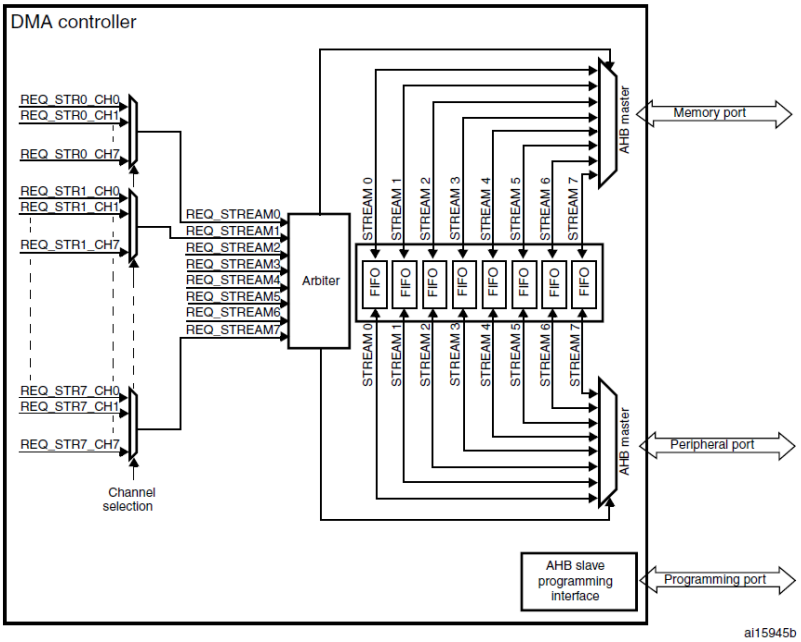

Fig. 3 The DMA controller in the STM32F446RE microcontroller with streams and their cannels [5]

In the Fig. 3 is shown DMA controller of STM32F4 microcontrollers included 8 streams, where each stream includes 8 channels connected to the peripherals. If a programmer wants correctly configure DMA controller, he must know right stream and the channel of peripherals. Table of peripherals connected to the concretely stream and the channel is shown in the Table I.

Table I. DMA2 request mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	ADC1	SAI1_A	TIM8_CH1 TIM8_CH2 TIM8_CH3	SAI1_A	ADC1	SAI1_B	TIM1_CH1 TIM1_CH2 TIM1_CH3	SAI2_B
Channel 1	-	DCMI	ADC2	ADC2	SAI1_B	-	-	DCMI
Channel 2	ADC3	ADC3	-	-	-	-	-	-
Channel 3	SPI1_RX	-	SPI1_RX	SPI1_TX	SAI2_A	SPI1_TX	SAI2_B	QUADSPI
Channel 4	SPI4_RX	SPI4_TX	USART1_RX	SDIO		USART1_RX	SDIO	USART1_TX
Channel 5	-	USART6_RX	USART6_RX	SPI4_RX	SPI4_TX	-	USART6_TX	USART6_TX
Channel 6	TIM1_TRIG	TIM1_CH1	TIM1_CH2	TIM1_CH1	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	-
Channel 7	-	TIM8_UP	TIM8_CH1	TIM8_CH2	TIM8_CH3	-	-	TIM8_CH4 TIM8_TRIG TIM8_COM

If we want to use a DMA controller for transmission TX SPI4 data, we must enable DMA2 peripheral clock and set Stream1 and Channel4. So this way we can enable and configured another peripherals such as A/D converters, D/A converters, Timers, USART/UART peripherals, etc. Choice of the stream and correct channel is only first configuration. If user wants to correctly set DMA controller, he has to set other bits too. The DMA of STM32F446RE microcontroller has more registers, but there are only four registers which is necessarily set correctly. It is *DMA stream x configuration register*, *DMA stream x number of data register*, *DMA stream x peripheral address register* and *DMA stream x memory 0/1 address register*, where "*x"* present stream number.

The DMA stream x memory 0 address register (M0AR) served for sets memory storage. If is chosen *peripheral to memory* direction, M0AR present register where will be data saved. If there is memory to peripheral direction set, M0AR includes data which will be transferred to the peripheral. For example: we want use DMA to the SPI4_TX (TX – transmit data, RX – receive data) and data that we want to transmit to the other device through SPI are saved in *data_tx* variable. The command will look like this:

- *DMA2_Stream1->M0AR = (uint32_t)&data_tx;*

The next register is DMA stream x peripheral address register (PAR). This register points to the peripheral interface where will be data transfer perform (in the *memory to peripheral* direction), or from where will be data come to the peripheral (in the *peripheral to memory* direction). The command for peripheral address setting will look like this:

- *DMA2_Stream1->PAR = (uint32_t)&(SPI4->DR);*

The DMA stream x number of data register (NDTR) served for determining number of data transfer. If we want send buffer where are saved 10 values, it need set NDTR register to number 10. The command for data number configuration will look like this:

- *DMA2_Stream1->NDTR = 1;*

The last register needed to the configuration DMA is DMA stream x configuration register (CR). There are more bits that are necessary sets. The first is channel selection. If we want to set SPI4 on the stream 1 and channel 4, we must set a bit:

- *DMA2_Stream1->CR |= DMA_SxCR_CHSEL_2;*

The data transmission direction is set in the same register by using the command: *DMA2_Stream1->CR |= DMA_SxCR_DIR_0;* In this case is chosen *memory to peripheral* direction, but we know change the configuration by *peripheral to memory* using by *~DMA_SxCR_DIR* or *memory to memory* with the command *DMA2_Stream1->CR |= DMA_SxCR_DIR_1;* There are the other bits too. For configuration memory and peripheral size, we must set bits *MSIZE and PSIZE* for transmitting size determining.

There are three possibilities for selecting memory and peripheral size 8-bit, half-word (16-bit) and word (32-bit) length. After transmitting complete processor can generate interrupt with set bit *TCIE* by using command *DMA2_Stream1->CR |= DMA_SxCR_TCIE;* There is possible to generate interrupt after the half transfer by using *HTIE* bit. If we want repeat data transmission after sent the last data, it is possible set *CIRC* bit for circular mode. The *CT* bit set target memory (Memory 0 – default or Memory 1). The *MINC* bit provides memory increment. It means that we want to sent a data buffer included 100 valued, this bit will be increment buffer position and program than sends all buffer values. The last necessary bit is *EN* uses for enabling DMA.

```
121  void dma_spi_config()
122  {
123    RCC->AHB1ENR |= RCC_AHB1ENR_DMA2EN;          //clock enabling to the DMA2
124    //TX
125    DMA2_Stream1->CR |= DMA_SxCR_CHSEL_2;        //channel4    dma_TX
126    DMA2_Stream1->M0AR = (uint32_t)&data_tx;     //Memory with data stored
127    DMA2_Stream1->PAR = (uint32_t)&(SPI4->DR);   //used register for transmitt
128    DMA2_Stream1->NDTR = 100;                    //number of conversions
129    DMA2_Stream1->CR &= ~DMA_SxCR_CIRC;          //circular mode disabled
130    DMA2_Stream1->CR &= ~DMA_SxCR_CT;            //target select - Memory 0
131    DMA2_Stream1->CR |= DMA_SxCR_DIR_0;          //direction - peripheral to memory
132    DMA2_Stream1->CR |= DMA_SxCR_MINC;           //memory increment enable
133    DMA2_Stream1->CR |= DMA_SxCR_MSIZE_0;        //16-bit (half-word)-memory
134    DMA2_Stream1->CR |= DMA_SxCR_PSIZE_0;        //16-bit (half-word)-peripheral
135    DMA2_Stream1->CR |= DMA_SxCR_TCIE;           //transmit complete interrupt enable
136    DMA2_Stream1->CR |= DMA_SxCR_EN;             //DMA enable
137
138    NVIC_EnableIRQ(DMA2_Stream1_IRQn);           //interrupt enable to stream4
139    NVIC_SetPriority(DMA2_Stream1_IRQn,0);
140  }
```

Fig. 4 DMA configuration to the SPI4_TX

Of course, there are a lot of bits, but for correctly function suffice mentioned bits and registers. The Fig. 4 shows a full DMA configuration for SPI4_TX. It is necessary enable a clock to the DMA peripheral using by *RCC->AHB1ENR |= RCC_AHB1ENR_DMA2EN;* without it DMA will not work.

In the next figure we can see the main configuration register and its bitmap.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	CHSEL[2:0]			MBURST [1:0]		PBURST[1:0]		Res.	CT	DBM	PL[1:0]	
				rw	rw	rw	rw	rw	rw	rw		rw	rw	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

Fig. 5 DMA configuration register [5]

There are some bits which were not mentioned in the previous text. The *DMEIE* bit enables *Direct mode error interrupt enable*. If occurs direct mode error the *DMEIF* is set in the status register. The flag is set when a DMA request occurs while the previous data have not yet been fully transferred into the memory.

The *PFCTRL* bit means *Peripheral flow controller*. This bit is cleared in default configuration because the number of data items to be transferred is programed by software into the *DMA_SxNDTR* register. If the *PFCTRL* bit is set, the number of data items to be transferred is unknown. The peripheral indicates by hardware to the DMA controller when the last data are beginning transferred. The next not mentioned bit is *PINC* bit. If this bit is cleared, the peripheral address pointer is fixed. If this bit is set, the peripheral address pointer is incremented after each data transfer. The *PINCOS* bit - *Peripheral increment offset size* has meaning only if *PINC* bit is set. If the *PINCOS* bit is cleared, the offset size of the peripheral address calculation is linked to the *PSIZE*. This bit is used to align the increment offset size with the data size on the peripheral or on a 32-bit address. If this bit is set the offset size for the peripheral address calculation is fixed to 4. The 16 and 17 bits are for priority configurations – *PL*. There are 4 possibilities for determining priority: *Very high priority* by set log. 1 to the both bits, *High priority* by set log. 1 to the 17. bit, *Medium priority* request sets log. 1 to the 16. bit and if there is cleared the both bits it is set *Low priority*. By using priority, we can set what the DMA request will be performed if the request comes in the same time. If two requests have the same

software priority level, the stream with the lower number takes priority over the stream with the higher number. For example, Stream 2 takes priority over Stream 4. When the *DBM – Double buffer mode* bit is enabled, the Circular mode is automatically enabled and at each end of the transaction, the memory pointers are swapped. In this mode, the DMA controller swaps from one memory target to another at each end of the transaction. The last bits *21* to *24 – PBURST* and *MBURST* indicates number of beats in the burst. There is possible chose single transfers (single transfers – without PBURST or MBURST) or incremental burst transfers of 4, 8 or 16. Depending on the single or burst configuration, each DMA request initiates a different number of transfers on the peripheral port:

- When the peripheral port is configured for single transfers, each DMA request generates a data transfer of a byte, half-word or word depending on the PSIZE bits.
- When the peripheral port is configured for burst transfers, each DMA request generates 4, 8 or 16 beats of byte, half word or word transfers depending on the PBURST and PSIZE bits.[5]

The STM32F446RE microcontroller has 2 status registers in a DMA controller and 2 interrupt flag clear registers. The status registers are divided to *DMA low interrupt status registers* (*DMA_LISR*) used by Stream 0 to Stream 3 and *DMA high interrupt status registers* (*DMA_HISR*) used by Stream 4 to Stream 7. The interrupt flag clear register is divided to the low and high register as such as status registers. The status registers include 6 bites for each stream: The *FEIFx* (Stream x FIFO error interrupt flag) bit informs about FIFO error. If this bit is set interrupt occurs. The next bit is *DMEIFx* (Stream x direct mode error interrupt flag) which is set if an error occurs in the Direct mode. The Transfer error interrupt flag – *TEIFx* bit is set when an error occurs pending data transfer. The last two bits – *HTIFx* (Stream x half transfer interrupt flag) and *TCIFx* (Stream x transfer complete interrupt flag) inform the user about complete transfer or half data transfer. The status register served only for read. There are interrupt flag clear registers (only write) for clearing flags in the status registers. [5]

DMA high interrupt status register (DMA_HISR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	TCIF7	HTIF7	TEIF7	DMEIF7	Res.	FEIF7	TCIF6	HTIF6	TEIF6	DMEIF6	Res.	FEIF6
				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
Res.	Res.	Res.	Res.	TCIF5	HTIF5	TEIF5	DMEIF5	Res.	FEIF5	TCIF4	HTIF4	TEIF4	DMEIF4	Res.	FEIF4
				r	r	r	r		r	r	r	r	r		r

DMA low interrupt flag clear register (DMA_LIFCR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	CTCIF3	CHTIF3	CTEIF3	CDMEIF3	Res.	CFEIF3	CTCIF2	CHTIF2	CTEIF2	CDMEIF2	Res.	CFEIF2
				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
Res.	Res.	Res.	Res.	CTCIF1	CHTIF1	CTEIF1	CDMEIF1	Res.	CFEIF1	CTCIF0	CHTIF0	CTEIF0	CDMEIF0	Res.	CFEIF0
				w	w	w	w		w	w	w	w	w		w

Fig. 6 High interrupt status register (upper) and Low interrupt flag clear register of STM32F4

The upper was mentioned that clear registers are divided to high and low register such as status registers. The Stream 0 to Stream 3 are in the *DMA low interrupt flag clear register* (*DMA_LIFCR*) and Stream 4 to Stream 7 are in the *DMA high interrupt flag clear register* (*DMA_HIFCR*). The both registers include bits for clearing status registers: *CFEIFx* – Stream x clear FIFO error interrupt flag, *CDMEIFx* – Stream x clear direct mode error interrupt flag, *CTEIFx* – Stream x clear transfer error interrupt flag, *CHTIFx* – Stream x clear half transfer interrupt flag, and the last *CTIFx* bit named Stream x clear transfer complete interrupt flag. [5]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	FEIE	Res.	FS[2:0]			DMDIS	FTH[1:0]	
								rw		r	r	r	rw	rw	rw

Fig. 7 FIFO control register

The last DMA register of STM32F446RE, shown in the Fig. 7 is DMA stream x FIFO control register (*DMA_SxFCR*). This register consists of three states (only read) bits and 4 read/write bits. Each stream has an independent 4-word FIFO and the threshold level are software-configurable between 1/4, 1/2, 3/4, or full. It is possible to configure by using *FTH* (FIFO threshold selection) bits. To enable the use of the FIFO threshold level, the direct mode must be disabled by setting the DMDIS bit in this register. The *FS* (FIFO status) bits uses only to read and these bits inform us about FIFO level. The last bit – *FEIE* (FIFO error interrupt enable) bit use for enabling interrupt when some error occurs. [5]

IV. Conclusion

The STM32F446RE microcontroller is a very fast and reliable system and with DMA controller it can work faster. There are a lot of possibilities where we can use DMA controller – for fast data processing of AD converter, it can be used for fast communication data transmit, for digital signal processing, etc. However, DMA controller is thanks its ability implemented to the all modern microcontrollers and systems using high speed data processing.

Acknowledgment

We support research activities in Slovakia / Project is co-financed from EU funds. This paper was developed within the Project "Centre of Excellence of the Integrated Research & Exploitation the Advanced Materials and Technologies in the Automotive Electronics ", ITMS 26220120055

References

[1] Řízení IO přenosu DMA řadičem, [online], <http://home.zcu.cz/~dudacek/Pot/dma_kanal.pdf>

[2] Hrůza P., Lízal V., SYSMAN , Korsorcium SYSTEM SOFT, Systémový manuál, verze 1.10 [online] <http://www.umel.feec.vutbr.cz/~adamek/komp/data/4.htm>

[3] http://www.umel.feec.vutbr.cz/~adamek/komp/data/4.htm

[4] STMicroelectronics, *STM32F446xC/E Datasheet*, [online], <http://www.st.com/en/microcontrollers/stm32f446re.html>

[5] STMicroelectronic, RM0090 Reference manual, [online], <http://www.st.com/content/ccc/resource/technical/document/reference_manual/4d/ed/bc/89/b5/70/40/dc/DM00135183.pdf/files/DM00135183.pdf/jcr:content/translations/en.DM00135183.pdf>

[6] Guzan M., Špaldonová D., Hodulíková A., Tomčíková I., Gladyr A.: *Boundary Surface and Load Plane of the Ternary Memory*, In: Electromechanical and energy saving systems. Vol. 15, no. 3 (2011), p. 163-167. - ISSN 2072–2052

[7] Dziak,J. : *Linear circuit simulation using MATLAB and modeling of nonlinear elements*, In: SCYR 2014 Proceeding from Comference: 20.5.2014: Herľany, S. 70 - 71, Košice : Technická univerzita v Košiciach, 2014 /978-80-553-1714-4/.

[8] Kováčová I.,: *EMC of power DC electrical drives* - 2005. In: Journal of Electrical Engineering. Vol. 5, no. 1 (2005), p. 61-66. - ISSN 1582-4594

[9] Vince T., Molnár J., Bučko R.: *Real-time regulation systems based on internet - optimization algorithm* - 2010. In: Transactions of KMOSU. Vol. 4, no. 3 (2010), p. 150-153. - ISSN 2072-8263

[10] Bereš M.: *Vstupné a výstupné charakteristiky znižovaco- zvyšovacích impulzových DC-DC meničov* - 2016. In: Electrical Engineering and Informatics 7 : proceedings of the Faculty of Electrical Engineering and Informatics of the Technical University of Košice. - Košice : FEI TU, 2016 S. 793-796. - ISBN 978-80-553-2599-6

[11] Schweiner D.: *A parallel connection of the DCDC converters and the current-sharing methods* - 2017. In: SCYR 2017. - Košice : TU, 2017 S. 14-17. - ISBN 978-80-553-3162-1