blog.world3.net

Using the XMEGA DMA controller

Finally got the XMEGA DMA controller on an ATxmega128A3 working. There is not much example code for it and the datasheet is not very clear, so I wrote this.

The datasheet and examples don't really explain the terminology well. A transaction is the complete cycle of the DMA controller. To start a transaction you load up the control registers and enable the DMA channel. It will then respond to triggers. The transaction ends when TRFCNT * REPCNT bytes have been transferred.

A block is a way of dividing the transaction up to fit your buffer size. If you just want to fill one buffer from start to finish you can set the block size in TRFCNT to the full size of your buffer. There are not many situations where you would want to use blocks smaller than one whole transaction, so for simplicity you can often just ignore them and look at transactions only.

The burst length is the number of bytes copied in one go by the DMA controller and is set with the BURSTLEN bits in CTRLA. Often it is used to read/write 16 bit SFRs such as the ADC result or the DAC output level. You always want to access those 16 bit words in one go rather than as two 8 bit bytes and the burst length lets you do that.

Now comes the poorly documented bit. First we have the single shot mode bit in CTRLA (bit 2 SINGLE). If set every time the DMA channel is triggered it transfers a single burst of data, e.g. 2 bytes if you set BURSTLEN to 2BYTE. If it isn't set the DMA channel goes into free running mode and simply transfers bursts as fast as possible until the transaction ends. So if you want to save ADC readings into a buffer like I am you want single shot mode triggered by the ADC.

Next let's look at repeat mode. In repeat mode the DMA controller transfers REPCNT blocks of data. To keep it simple let's say you only have one buffer and intend to fill it in one go, so you set the block size to the buffer size. If you are not in repeat mode the buffer will be filled once and then the transaction will end and the DMA channel will be disabled. If you are in repeat mode the DMA channel will perform REPCNT block transfers, and since you set your block size to the buffer size that means it is the number of times the buffer will be filled consecutively. If you are in repeat mode and you set REPCNT to zero the DMA channel will repeat forever.

Now keep in mind that even with repeat mode enabled the DMA controller still has to be triggered. In normal mode one trigger transfers an entire buffer's worth of data, and in single shot mode each trigger transfers one burst of data.

Finally we have double buffering mode. Double buffering basically alternates between filling two buffers. When one transaction is complete the DMA channel automatically stops and its partner starts. This action is independent of repeat mode, and will set the transaction complete flag so that you can detect when it happens.

Note 1. You need to look at the transaction complete flags, not the DMA channel active or pending flags, if you want to know when the channel has finished and the buffer is ready for processing.

Note 2. If you want to use a timer to trigger a DMA channel then you must enable that timer's interrupt. You can give it a null interrupt handler (e.g. ISR(TCC1_OVF_vect){} or just RETI in assembler) but it does have to actually be enabled and triggering, otherwise the DMA channel will not be triggered either. That doesn't apply to other peripherals such as the ADC, it seems to be just the timers.

```
PORTA.DIRCLR = PIN7_bm;
PORTA.OUTCLR = PIN7_bm;

ADCA.CH0.CTRL = ADC_CH_INPUTMODE_SINGLEENDED_gc | ADC_CH_GAIN_1X_gc;
ADCA.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN7_gc;
adc_wait_8mhz(&ADCA);
ADCA.EVCTRL = ADC_SWEEP_0_gc | ADC_EVSEL_0123_gc | ADC_EVACT_CH0_gc; // event channel 0 trig
ADCA.CTRLA |= ADC_ENABLE_bm;

// set TCC1 to 11024Hz overflow, actually 11019.2838Hz (-0.052% error)
```

Search for:

Search

November 2011

M	T	W	T	F	S	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

« Oct Dec »

Meta

Log in
Entries RSS
Comments RSS
WordPress.org

Categories

audio (1)
avr (27)
BBC (2)
electronics (30)
genius (4)
hardware (24)
idiots (41)
Internet (23)
law (20)
microcontrollers (12)
networking (17)
politics (31)
privacy (19)
Retro Adapter (5)
security (17)
software (32)
Uncategorized (18)
windows (28)

Archives

July 2013
January 2013
November 2012
August 2012
July 2012
June 2012
May 2012
April 2012
March 2012
February 2012
January 2012
December 2011
November 2011
October 2011

```
TCC1.CTRLA = 0; // stop if running
TCC1.CNT = 0;
TCC1.PER = 0x02D5;

EVSYS.CH0MUX = EVSYS_CHMUX_TCC1_OVF_gc; // trigger on timer overflow

// reset DMA controller
DMA.CTRL = 0;
DMA.CTRL = DMA_RESET_bm;
while ((DMA.CTRL & DMA_RESET_bm) != 0)
;
// configure DMA controller
DMA.CTRL = DMA_CH_ENABLE_bm | DMA_DBUFMODE_CH01_gc; // double buffered with channels 0 and 1

// channel 0
// **** TODO: reset dma channels
DMA.CH0.REPCNT = 0;
DMA.CH0.CTRLA = DMA_CH_BURSTLEN_2BYTE_gc | DMA_CH_SINGLE_bm | DMA_CH_REPEAT_bm; // ADC resul
DMA.CH0.ADDRCTRL = DMA_CH_SRCRELOAD_BURST_gc | DMA_CH_SRCDIR_INC_gc | // reload source after
DMA_CH_DESTRELOAD_TRANSACTION_gc | DMA_CH_DESTDIR_INC_gc; // reload dest after every transac
DMA.CH0.TRIGSRC = DMA_CH_TRIGSRC_ADCA_CH0_gc;
DMA.CH0.TRFCNT = 2048; // always the number of bytes, even if burst length > 1
DMA.CH0.DESTADDR0 = (( (uint16_t) buffer_a) >> 0) & 0xFF;
DMA.CH0.DESTADDR1 = (( (uint16_t) buffer_a) >> 8) & 0xFF;
DMA.CH0.DESTADDR2 = 0;
DMA.CH0.SRCADDR0 = (( (uint16_t) &ADCA.CH0.RES) >> 0) & 0xFF;
DMA.CH0.SRCADDR1 = (( (uint16_t) &ADCA.CH0.RES) >> 8) & 0xFF;
DMA.CH0.SRCADDR2 = 0;

// channel 1
DMA.CH1.REPCNT = 0;
DMA.CH1.CTRLA = DMA_CH_BURSTLEN_2BYTE_gc | DMA_CH_SINGLE_bm | DMA_CH_REPEAT_bm; // ADC resul
DMA.CH1.ADDRCTRL = DMA_CH_SRCRELOAD_BURST_gc | DMA_CH_SRCDIR_INC_gc | // reload source after
DMA_CH_DESTRELOAD_TRANSACTION_gc | DMA_CH_DESTDIR_INC_gc; // reload dest after every transac
DMA.CH1.TRIGSRC = DMA_CH_TRIGSRC_ADCA_CH0_gc;
DMA.CH1.TRFCNT = 2048;
DMA.CH1.DESTADDR0 = (( (uint16_t) buffer_b) >> 0) & 0xFF;
DMA.CH1.DESTADDR1 = (( (uint16_t) buffer_b) >> 8) & 0xFF;
DMA.CH1.DESTADDR2 = 0;
DMA.CH1.SRCADDR0 = (( (uint16_t) &ADCA.CH0.RES) >> 0) & 0xFF;
DMA.CH1.SRCADDR1 = (( (uint16_t) &ADCA.CH0.RES) >> 8) & 0xFF;
DMA.CH1.SRCADDR2 = 0;

DMA.CH0.CTRLA |= DMA_CH_ENABLE_bm;
TCC1.CTRLA = TC_CLKSEL_DIV1_gc; // start timer, and in turn ADC

for (i = 0; i < 20; i++)
{
while (!(DMA.INTFLAGS & DMA_CH0TRNIF_bm));
DMA.INTFLAGS = DMA_CH0TRNIF_bm;
TERM_tx_char('A');

while (!(DMA.INTFLAGS & DMA_CH1TRNIF_bm));
DMA.INTFLAGS = DMA_CH1TRNIF_bm;
TERM_tx_char('B');
}
```

In the example the code outputs ‘A’ or ‘B’ when each buffer is full.

This entry was written by mojo, posted on 04/11/2011 at 11:47, filed under avr, electronics. Bookmark the permalink. Follow any comments here with the RSS feed for this post. Post a comment or leave a trackback: Trackback URL.

« Democratic FAIL
XMEGA TWI (I2C) bus pull-ups »

4 Comments

1. Kamul
   Posted 03/01/2013 at 16:05 | Permalink

   Thank you very much for this nice piece of article!

September 2011
July 2011
June 2011
May 2011
March 2011
January 2011
December 2010
November 2010
August 2010
July 2010
June 2010
May 2010
April 2010
March 2010
February 2010
January 2010
December 2009
November 2009
October 2009
September 2009
August 2009
July 2009
June 2009
May 2009
April 2009
March 2009
February 2009
January 2009
December 2008
November 2008
October 2008
September 2008
August 2008
July 2008
June 2008
May 2008
April 2008
March 2008
February 2008
January 2008
November 2005

Links:

Main site: world3.net

Electronics:
denki.world3.net

2. Michael
Posted 22/12/2013 at 14:52 | Permalink

Thanks a lot for providing this code. It helped me a lot to get the DMA working. I have two comments:

1.) For enabling the DMA controller, one should use “DMA.CTRL = DMA_ENABLE_bm”. DMA_CH_ENABLE_bm is meant for the channels. It just happens to be the same bit, therefore it works, but it is just pure luck.

2.) In my application, I need to read port values at regular intervals of about 16µs. I did it initially by a DMA, triggered by the overflow interrupt of a counter. As described above, the timer interrupt must be enabled, otherwise it does not work. The reason is simple, the interrupt request of the timer is not cleared otherwise. But handling an interrupt is an overhead, which I wanted to avoid by the DMA. The XMega devices provide a smart workaround: events. Now I trigger the DMA by an event and the event is triggered by the overflow of the timer. No need for generating an interrupt. Setting up the event is simple:

```
/* Initialize Event System channel 2 */
EVSYS.CH2MUX = EVSYS_CHMUX_TCE0_OVF_gc ; /* select TCE0 overflow as event source */
EVSYS.CH2CTRL = EVSYS_DIGFILT_1SAMPLE_gc ; /* active for 1 sample */
```

The DMA channel trigger is configured as follows:

```
DMA.CH0.TRIGSRC = DMA_CH_TRIGSRC_EVSYS_CH2_gc ; /* Event system channel 2 triggers DMA transfer */
```

Doing it this way, the interrupts of the timer can remain disabled. Hope this helps others who have the same application.

Michael

3. mojo
Posted 22/12/2013 at 14:59 | Permalink

Thanks, useful stuff. I have since been using events too.

4. MaS24
Posted 25/03/2015 at 11:18 | Permalink

This is only one example of use ADC with DMA. I must read 9 potentiometers via ADC. I guess, no Double Buffer is needed in my project. How to transfer all 18 bytes (2×9) from DMA to variables? Is it necessary to be a dimensioned table?

Post a Comment

Your email is *never* shared. Required fields are marked *

Name *

Email *

Website

Comment

Post Comment

Notify me of followup comments via e-mail

たとえ溺れても梦はゆめでしかない