Atmel AVR1617: Frequency Measurement with Atmel AVR XMEGA Family Devices

Features

- **Atmel® AVR® XMEGA® family devices**
- **Modular C functions easily integrated into customer application code**
- **Effective use of two event channels, two 16-bit counters, and one interrupt**
- **Function calls are reuseable between most AVR XMEGA devices**
- **Includes software option to expand to 32-bit result**
- **Measure frequencies up to 16MHz using a 32MHz clk_{CPU}**

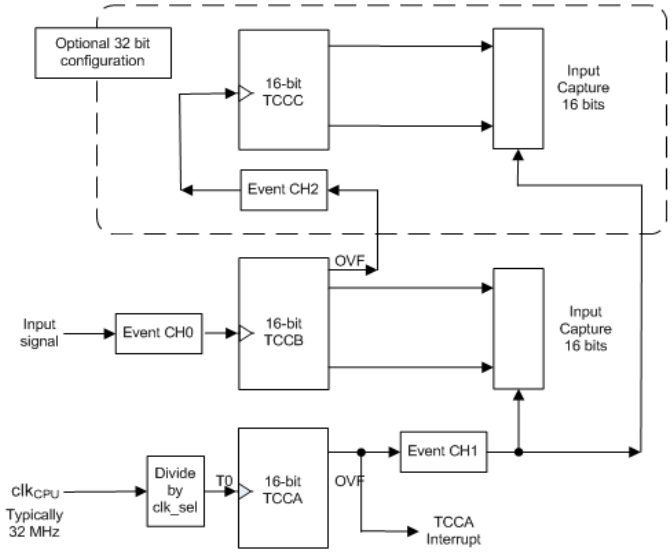
1 Introduction

Frequency measurement is a function commonly needed in today's consumer and industrial applications, and is easily implemented by using the XMEGA Event System and counter/timers. A typical scenario is shown in Figure 1-1. A signal whose frequency is to be measured is applied to an XMEGA input pin, which is assigned to one channel of the XMEGA Event System. The event channel output is routed to TCCB, a 16-bit timer/counter configured in a basic count-up mode.

A second 16-bit timer/counter, TCCA, is also configured as a simple count mode timer, which allows a counting period of, typically, 125ms. This period is configurable by a function call to adapt to a wide range on input frequencies.

As timer TCCA reaches its overflow (OVF), the 16-bit value in TCCB is latched into a 16-bit input capture register.

Figure 1-1. Frequency counter block diagram.



8-bit Atmel Microcontrollers

Application Note

Rev. 8383A-AVR-06/11

In order to provide higher resolution, the basic method is expandable to 32 bits of counting range with the addition of a third event channel, a third timer/counter, TCCC, and its input capture register, as shown in the upper portion of Figure 1-1.

A clk_{CPU} of 32MHz is used in this application example because it permits an input frequency as high as 16MHz to be measured. The `xmega_freq_cntr.c` driver can accommodate other AVR device clock sources including a crystal oscillator.

2 Prerequisite

The frequency measurement demo discussed in this document requires basic familiarity with following:

- C programming language for embedded systems
- Compiling C projects with WinAVR GCC compiler and Atmel AVR Studio® 4.18 or Atmel AVR Studio 5 integrated design environment (IDE)
- Atmel STK®600 with XMEGA adapter socket and one of various XMEGA devices
- Atmel AVR JTAGICE mkII debugger, optional for debugging support
- General familiarity with frequency counters as used in industry or a lab environment

3 Limitations

- The Atmel AVR XMEGA device operating with the recommended 32MHz internal RC oscillator is limited to measuring clock frequencies up to 32MHz / 2, or 16MHz
- The software solution with this application note was tested with AVR Studio 4.18. Newer or older versions of AVR Studio and the GCC Compiler may require some modifications
- This technique does not use the XMEGA device feature of measuring frequency by counting the time between input signal edges and then taking the reciprocal, which would require floating point calculations. For more information, see Chapter 7, “Frequency capture option”.

4 XMEGA target device resource requirements

Table 4-1. Peripheral requirements.

Peripheral	Pin(s)	Configurable?
Two or three [1] 16-bit timer/counters (TC)	Not applicable	Yes, other XMEGA TCs may be used
One I/O pin as input	One of many I/O pins	Yes
Three or four [1] event channels	None	Yes

Note: 1. If a 32-bit result is desired, then an additional third timer/counter and fourth event channel must be used.

Table 4-2. Memory requirements [1].

Memory	Typical size	Maximum size
Program memory	1016 bytes	1404 bytes with `EXPAND_TO_32_BITS = 1`
Data memory	5 bytes	7 bytes with `EXPAND_TO_32_BITS = 1`
Internal EEPROM memory	None required	

Note: 1. Exact memory requirements depend on a variety of factors, such as compiler version, optimization levels, and addition or removal of configurable functionality.

Table 4-1 describes the typical peripheral requirements for this implementation of a frequency counter. The minimum timer/counter requirement is two.

Table 4-2 describes the memory requirements. These numbers are the result of no compiler optimization. The program memory requirements would decrease if a different compiler optimization was selected. Also if `EXPAND_TO_32_BITS = 0` is used, then the peripheral and code requirements will be less.

5 Frequency counter using the Event System: How it works

5.1 Incoming signal to be measured connects through Event System to timer clock

Figure 5-1. Atmel AVR XMEGA input circuit and synchronizer.

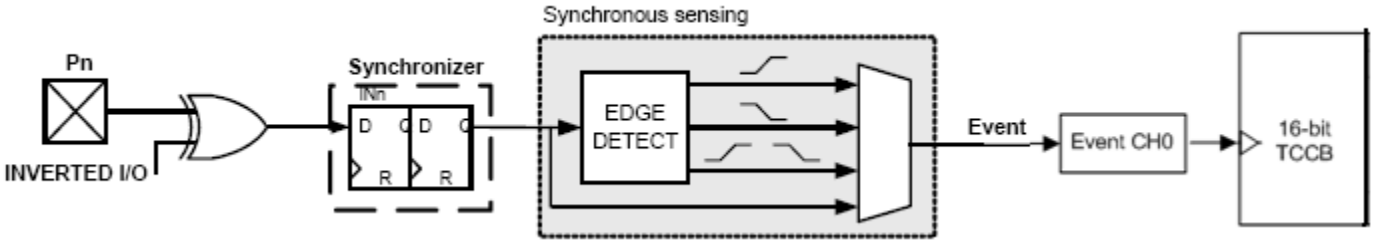


Figure 5-1 illustrates how synchronous inputs are handled by the AVR XMEGA device. The process is as follows:

The signal to be measured is synchronized by the AVR XMEGA device pin input circuit block before being submitted to the Event System. Then, two successive D flip-flops essentially eliminate a meta-stable condition from being allowed into the event detection system.

NOTE

In this case, rising and falling edges of the input signal both cause counter TCCB to increment.

The input signal is connected to the event channel with the following C statement

```
EVSYS.CH0MUX = EVSYS_CHMUX_PORTD_PIN0_gc;
```

In this specific example, Port D, Pin 0 is assigned to Channel 0 of the Event System.

The event channel is routed to the timer/counter with the following C statement:

```
TCC1.CTRLA = TC_CLKSEL_EVCH0_gc;
```

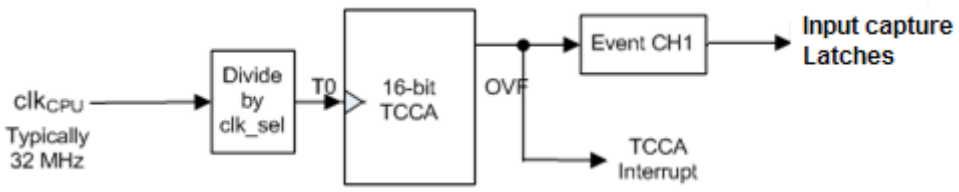
The instruction in `xmega_freq_cntr.c` is a slight variation of the instruction above to allow different timer/counters to be used.

5.2 Timer TCCA used to generate the counting interval; that is, the gate interval time

Referring to Figure 5-2, below, an AVR XMEGA device core clock of 32MHz is recommended for the best possible range of gate interval times.

Before each frequency measurement, counter TCCA is preloaded with a value such that it counts up to an 0xFFFF overflow condition (OVF), and then triggers Event Channel 1 and the TCCA interrupt, which is used by an interrupt service routine (ISR) in the `xmega_freq_cntr.c` driver.

Figure 5-2. Counting interval generated by 16-bit timer/counter.



The `divide-by-clk_sel` divider is programmed by the function `xmega_tcca_clk_freq_sel(CLK_SEL)`, which is described in `xmega_freq_cntr.h`. This function sets a register value inside TCCA using the specified divisor.

In addition, the TCCA preload value must be selected by the user in `xmega_freq_cntr.h` as part of the `xmega_freq_cntr.c` driver initialization. In this example, 125ms is used. This value may be changed if a different gate interval time is desired.

5.3 Input capture used with TCCB

Figure 5-3. Diagram of TCCA and its input capture latch.

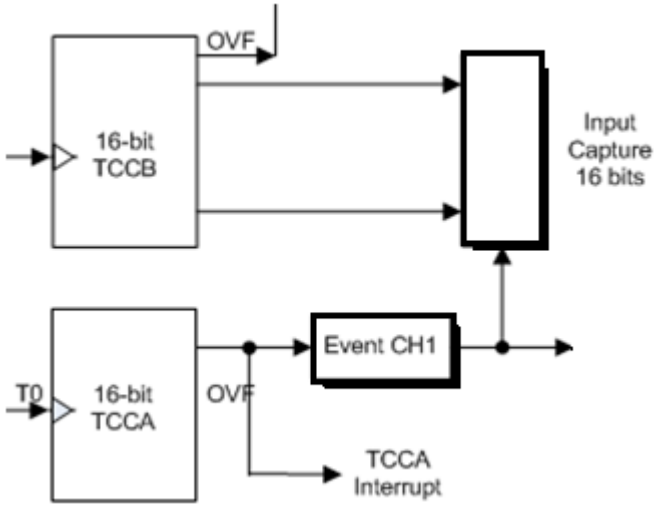


Figure 5-3 describes how the 16-bit input capture latch will clock the contents of TCCB via a clock pulse from Event Channel 1 as a result of a TCCA OVF event. The event channel is set up in the driver with the following instruction:

```
EVSYS.CH1MUX = EVSYS_CHMUX_TCC0_OVF_gc;
```

In the `xmega_freq_cntr.c` driver, TCC0 has been changed to TCCA so that the user may choose a different timer, if desired:

```
EVSYS.CH1MUX = EVSYS_CHMUX_TCCA_OVF_gc;
```

Next, Event Channel 1 is routed to TCCB via the following C instruction:

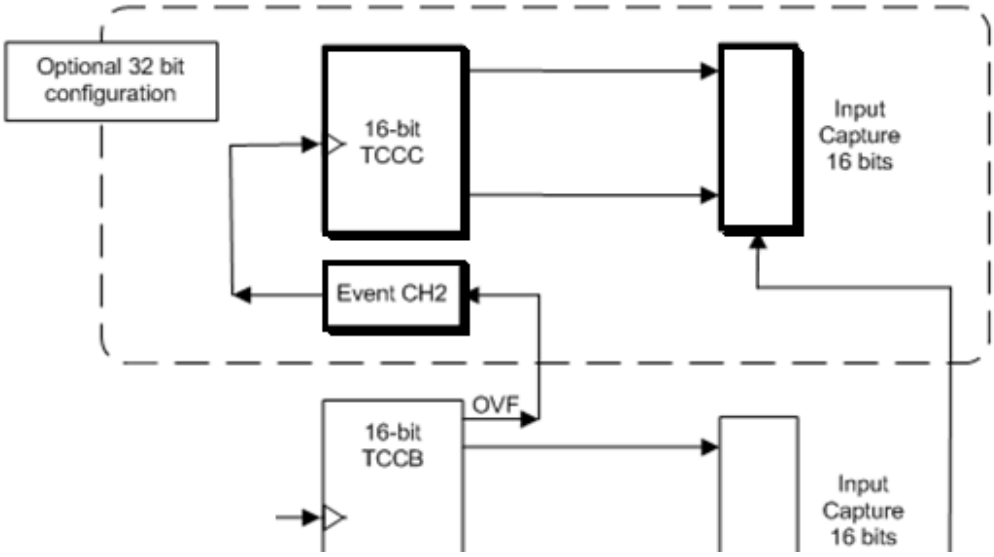
```
TCCB.CTRLD = (TC1_EVACT_gm & TCB_EVACT0_bm) | (TCB_EVSEL_gm &
(TCB_EVSEL3_bm | TCB_EVSEL0_bm));
```

This instruction has been modified to allow flexibility in the choice of Timer/Counter B. The #define for TCCB, located in `xmega_freq_cntr.h`, is as follows:

```
#define TCCB      TCC
```

5.4 Expansion to 32-bit result, uses TCCC

Figure 5-4. Block diagram of 32-bit expansion by an additional 16-bit timer/counter.



If a 32-bit frequency counter result is desired, change the #define in `xmega_freq_cntr.h` as follows:

```
#define result_32_bit   1
```

This statement will instruct the C compiler to use additional resources, as indicated in Figure 5-4, which include Event Channel 2 and an additional 16-bit timer/counter, noted above as TCCC.

6 How to build and run the software

6.1 Installing the Atmel AVR1617-freq_meas-xmega software package

From www.atmel.com, download the AVR1617.zip file. Unpack it into a working directory. The following files are included in the zip file:

`avr1617_xmega_freq_cntr.c`	The application demo code
`xmega_freq_cntr-drvr.c`	The driver code
`xmega_freq_cntr-drvr-drvr.h`	Contains #define device assignments
`avr1617_xmega_freq_cntr.aps`	The AVR Studio project file

Open the `avr1617_xmega_freq_cntr.aps` file with Atmel AVR Studio 4, or import the file into Atmel AVR Studio 5 and build the project after specifying the target Atmel AVR XMEGA device in the Project Build options.

Download the project into the target AVR XMEGA device, run the program, and feed a signal of known frequency into the specified input pin. In this example, the input pin is Port D, bit 0: PORTD0

To observe the operation of the input capture registers, insert a breakpoint in `avr1617_freq_meas_xmega.c` just before the comment

```
// result available here
```

and inspect the value of the variable `result`.

6.2 Functions

The following functions are described as prototypes in `xmega_freq_cntr.h`.

```
void xmega_freq_cntr_init(void);
```

This function initializes the timer/counters and the two (optionally three) event channels. One interrupt vector related to TCCA is also defined here.

Input: none Output: none

```
void  xmega_freq_cntr_start_meas(void);
```

This function initiates the measurement process by starting the TCCA gate timer, which is set in this demo to 125ms. This gate time is easily changed using the function `xmega_tcca_clk_freq_sel()`, described below.

Input: none Output: none

```
unsigned int xmega_freq_cntr_rtn_result(void);
```

This function returns the contents of the TCCB input capture register by using the 16-bit SRAM unsigned integer `ic_result`, which is defined in `xmega_freq_cntr.h` . This SRAM location is written to by the TCCA interrupt service routine just after TCCA overflows (OVF).

If the results are too high for 16 bits and there is no 32-bit option enabled, the value 0xFFFF indicates a TCCB OVF. In this case, the gate interval should be shortened, as indicated below.

Input: none Output: none

```
void  xmega_freq_cntr_clr_result(void);
```

After the driver returns the frequency to the application, it is necessary to clear the results in the driver to set it up for the next frequency measurement. This is accomplished by the application code, by calling this function.

Input: none Output: none

```
void xmega_tcca_clk_freq_sel(unsigned char CK_SEL);
```

This function sets the TCCA prescaler to divide clk_{CPU}, as shown in Table 6-1. The millisecond values are based on a clk_{CPU} of 32MHz, although another frequency could be used, such as an external oscillator.

Input: CK_SEL Output: none

Table 6-1. Maximum time interval based on TCCA prescaler.

CK_SEL	CPU_clk_div_by	If 32MHz osc, max interval time, ms
0	OFF	
1	div by 1	2.048
2	div by 2	4.096
3	div by 4	8.192
4	div by 8	16.384
5	div by **64**	131.072 (>125ms)
6	div by 256	524.288
7	div by 1024	2097.1

6.2.1 How to set the TCCA interval time using TCCA for the frequency range

Timer TCCA is configured by the `xmega_freq_cntr.c` driver using #defines in the `xmega_freq_cntr.h` file. The timer generates the gate interval time used to measure the input frequency. Timer TCCA is loaded with a specific value so that when it reaches 0xFFFF, it will generate an OVF condition. As OVF is reached, Event Channel 1 will clock the TCCB count into the input capture register. Also, the TCCA OVF will generate an interrupt, which will read the input capture latch to get the frequency result.

To select the proper gate interval value:

1. Consider the frequency range to be measured. To properly scale the operation of the frequency counting TCCB, the gate timer, TCCA, must not time out (reach OVF) after TCCB itself overflows.
2. Choose a gate interval time that is long enough to allow the TCCB counter to offer enough digits of resolution to suit the application.

As an example, to attain a gate interval time of 125ms, the following C instruction sets the TCCA clock to be clk_{CPU} / 64:

```
TCCA.CTRLA = ( TCCA.CTRLA & ~TC0_CLKSEL_gm ) |
    TC_CLKSEL_DIV64_gc;
```

The following #define sets the TCCA OVF value to 1/8 second (125ms).

The **8** defines the 1/8 second.

```
#define TCCA_CNT_TO_OVF         (0xffffUL)-(F_CPU/(64UL*8UL))
```

The 125ms timing interval may be changed to a longer time period for lower frequencies, or as short as 2.048ms for measuring high-frequency signals.

Consider again that the AVR XMEGA Event System supplies a clock pulse to TCCB on each rising and falling edge of the input signal. This causes TCCB to be clocked at twice the input frequency. A gate interval time of 125ms will allow up to 65534 edges to be counted, for a maximum frequency input of 8/2 × 65534 = 262136Hz.

If the `expand_to_32_bits` option is set to 1, the maximum input frequency may be as high as 16MHz. This option is located in the `xmega.freq.cntr.h` file.

7 Frequency capture option

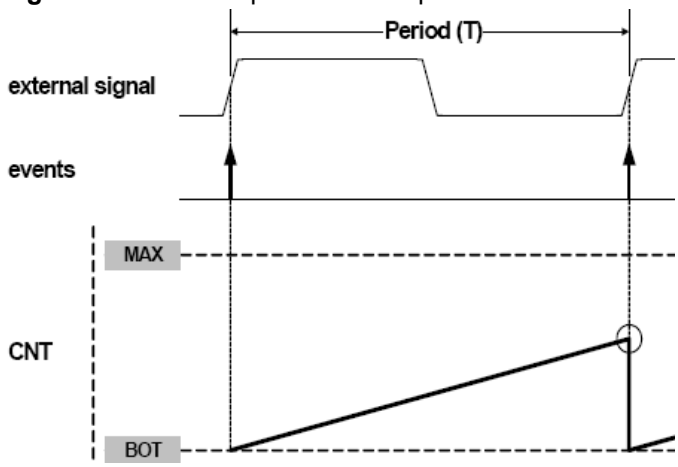
Another frequency measuring technique is described in the AVR XMEGA datasheet, although it is not implemented here. The Event System and a timer/counter are used to measure the time between two rising edges. Refer to Figure 7-1. This enables the timer/counter to use capture to measure the period or frequency of a signal directly. The capture result will be the time (T) from the previous timer/counter restart until the event occurrs. This can be used to calculate the frequency (f) of the signal:

$$f = 1/T$$

This technique is best for lower input frequencies, as the resolution of the result will decrease as the external input frequency increases. Consider the case where the external signal is 1MHz. If a 32MHz timer/counter clock is used, then the value of the CNT result would be the number 32. Then the result would be plus or minus one count, or ±3%.

Floating point software support would be required to invert CNT to arrive at the actual frequency, based on the equation above. By contrast, the method described in this application note with the 32-bit expansion and a one-second time counting interval would produce a count of 1,000,000 ± one count. This accuracy is dependent on the accuracy of the AVR XMEGA device's 32MHz oscillator.

Figure 7-1. An example where the period is measured for an external signal.



8 References

1. Atmel AVR Studio – www.atmel.com
2. The WinAVR GCC C compiler is available from http://winavr.sourceforge.net
3. AVR205: Frequency Measurement Made Easy with Atmel tinyAVR® and Atmel megaAVR®
4. Atmel AVR JTAGICE mk-II
5. National Instruments: “Frequency Measurements: How-To Guide,” http://zone.ni.com/devzone/cda/tut/p/id/7111

9 Table of contents

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: (+1)(408) 441-0311
Fax: (+1)(408) 487-2600
www.atmel.com

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Milennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chou-ku, Tokyo 104-0033
JAPAN
Tel: (+81) 3523-3551
Fax: (+81) 3523-7581

8383A-AVR-06/11