Freescale Semiconductor
Application Note

AN3731
Rev. 0, 06/2008

FlexTimer and ADC Synchronization

How FlexTimer is Used to Synchronize PWM Reloading and Hardware ADC Triggering

by: Eduardo Viramontes
Systems and Applications Engineering
Freescale Technical Support

1 Introduction

There are embedded applications that require several modules to work together in specific order and with precise timing. If timing requirements are too strict for software to be able to fulfill them, then hardware synchronization is needed to ensure the correct execution of an application.

This application note describes the hardware synchronization and trigger capability of the FlexTimer module (FTM) for applications that benefit from the use of that module, for better or more precise operation.

The FTM is not explained in detail in this application note, only the parts that deal with synchronization. For further detail on the FTM consult Section 5, "References," in this document.

Contents

2 ADC Conversion Triggers

Applications that require constant analog-to-digital conversions (real-time control and processing applications) need to save as much processor core time as possible. There are two reasons for this:

1. These kinds of applications have repetitive events that consume resources on a periodic basis.
2. Real-time applications need to perform mathematical and/or logical operations that also use processor resources.

Also related to ADC synchronization is the fact that real-time control requires constant, repetitive, and precise data sampling. Precise periodic sampling means predictable and precise control output. Without hardware triggering, sampling times vary with software instruction differences — with hardware triggering, sampling can be precisely timed.

With such requirements a processor may run out of resources if extra functions are added (for example, if a processor is expected not only to run a motor but also to control the user interface and a couple of valves). If dedicated hardware is able to trigger ADC conversions without the need for CPU intervention, then time is saved for other tasks. An added benefit is the possibility of precisely timing these conversions, whereas software may introduce an error in timing.

Processors that have the FTM (such as the MCF51ACxxx family) also include an ADC module that can be configured to be either software-triggered or hardware-triggered. The FTM is a source for hardware triggers to the ADC.

Generation of the hardware triggers to ADC is enabled when:

1. FTM_ENABLE = 1 (in the FTMx_MODE register) — enables FTM features. This bit is used to ensure backwards compatibility with the Timer/PWM Module (TPM), the precursor of the FTM. When this bit is 0 the FTM works as a TPM; when it is 1, the FTM features (such as ADC hardware triggers) are enabled.
2. CHj_TRIG (j is 2, 3, 4, 5, 6, or 7) bits (in the FTMx_ADC_TRIGGER register) are set — indicates if the match of the timer channel j (when FTM counter = FTMxCjVH:L) will generate a hardware trigger to the ADC.
3. ADHWTS(1:0) = 0b10 (in the SOPT2 register) — configures the FTM as a source for ADC hardware triggers.
4. ADTRG = 1 (in ADCSC2 register) — enables hardware triggers in the ADC module.
5. ADCH = selected channel (in ADCSC1 register) — selects the desired conversion channel that will be triggered by the FTM.

After all other elements have been configured, write the modulo and channel values and turn on the FTM by configuring the reference clock in FTM1SC. After this has been done, the next timer channel match (on a configured channel) will cause a conversion to start. If a Conversion Complete interrupt has been configured, this interrupt subroutine (ISR) is the first time the CPU uses resources to manage ADC conversions. After the initial configuration overhead, the periodic samples that follow require only the ISR overhead, which can be kept to a minimum if the software architecture allows it.

2.1 ADC Hardware Trigger Application — Example

The application example given here shows the benefits of ADC hardware triggers.

Here are some key points of this application:

- An industrial system generates PWM signals with several FTM channels to control the speed of a motor.
- The system needs to measure the output voltage from a sensor.
- Samples need to be obtained every PWM period to update the duty cycle.

Without hardware triggers, the time frame to obtain each sample is as shown in Figure 1.

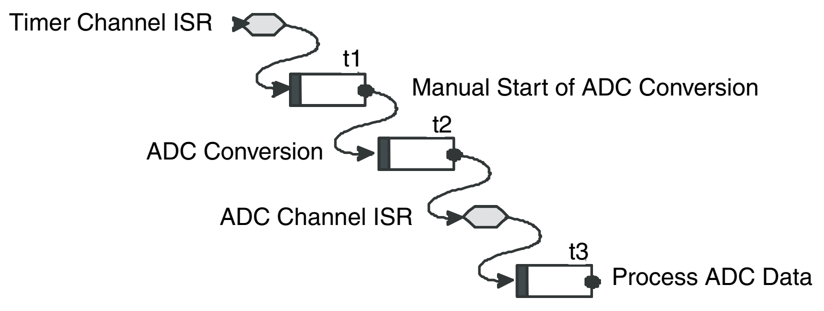

Figure 1. Time to Obtain an ADC Sample Without Hardware Trigger

With hardware triggers, the time frame is as shown in Figure 2.

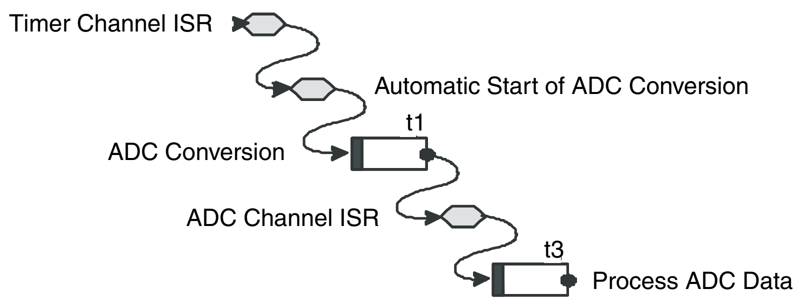

Figure 2. Time to Obtain an ADC Sample With Hardware Trigger

Without a hardware trigger, the total elapsed time from timer channel interrupt to a processed output is t1 + t2 + t3, and total CPU intervention is t2 + t3. With a hardware trigger, total time is t1 + t3, and total CPU intervention is only t3, the time required to process data.

The saved time means two things:

- Accumulated time is time that the CPU can use for other tasks.
- Instant time saved, period by period, means faster PWM signals can be achieved. Having faster PWM periods means that there is greater control resolution.

Figure 3 and Figure 4 show measurements based on the application example.

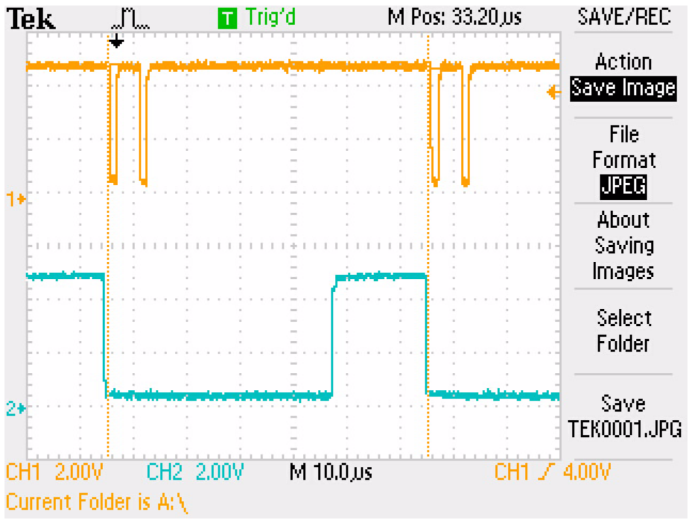

Figure 3. Timing Measurements Without Hardware Triggers

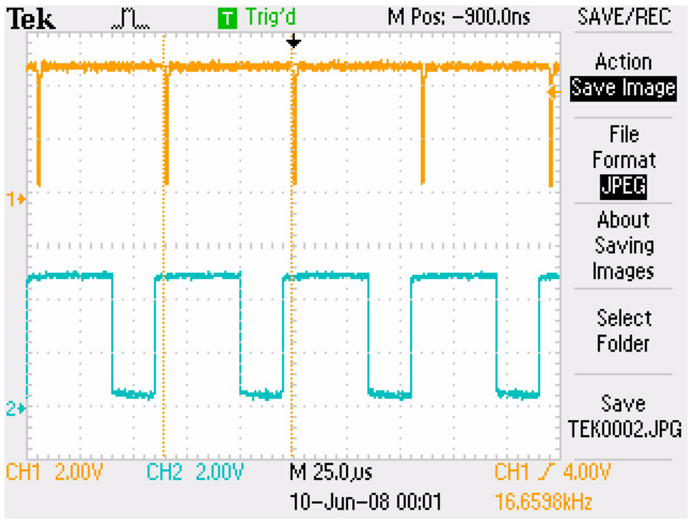

Figure 4. Timing Measurements With Hardware Triggers

In both cases, a general-purpose output pin is used to measure CPU load. A logic high is idle time, and as soon as the processor is used, the signal is driven low. The PWM signal is shown as a reference to the data being processed (for every sample, the value read by the ADC is multiplied by two and loaded as the PWM duty cycle).

The hardware-triggered version uses 1.4 µs versus 58.6 µs idle time, so CPU availability is 97.66%. The software-triggered version uses 2.8 µs versus 57.2 µs idle time, so CPU availability is 95.33%. The software-triggered version uses double resources for the same task and requires two interrupts separated by less than 5 µs. This timing can also affect the execution of other tasks that depend on this one, for example processing the ADC data. With the FTM, several trigger sources are available, so conversion can be set up at the beginning or end of the pulse width, the beginning or end of the PWM period, or even something in between, by using an extra auxiliary channel as trigger. With software triggering, these kinds of configurations are difficult to obtain and prone to errors.

The software used to generate these signals is included in the files that accompany this document. If the DEMOACKIT demo board is used, the outputs for PWM and CPU load pins are connected to LEDs in the demo board. The analog signal used for the measurement is controlled by the potentiometer in the board.

3 PWM Duty Cycle Reload Synchronization

Control applications based on PWM generation need to constantly adjust the PWM duty cycle to maintain the expected control level. If duty cycle compare values are changed at random times, the PWM signals may cause undesired output during one PWM period. Side effects of this behavior are ripples in the controlled output (meaning unexpected variations in the controlled output) and in some cases (such as motor control) an increase in mechanical noise.

The FTM includes a function that allows an ADC event to synchronize the reloading of FTM registers that are buffered. These are the FTM modulo value (which defines the PWM period), the channel values (which define the duty cycle), and the FTM counter initial value register. If FTM synchronization is enabled (SYNCH_ENABLE = 1) and the FTM is enabled, writes to these registers cause the values to be stored in a temporary buffer, then loaded after the synchronization event when the next PWM period starts (if FTM synchronization is not enabled, changes to the channel values cause the duty cycle to be updated after they were written).

The ADC event that causes synchronization is the setting of the Conversion Complete (COCO) bit. COCO will be set after a conversion is completed or, if the compare function is enabled (ACFE = 1), after a conversion is finished and the comparison is true.

3.1 PWM Reload Synchronization Application — Example

The application example given here uses ADC conversions to synchronize PWM duty cycle reloading.

Here are some key points of this application:

- A motor is being controlled with a PWM channel.
- A sensor outputs a voltage level that determines the speed of the motor.
- If the voltage level is above 2.5 V then the duty cycle must be 30% — if the voltage level is below 2.5 V, the duty cycle must be 70%.
- ADC channel-reading sensor voltage has been configured with the compare function and continuous conversion to set the COCO bit whenever the voltage equals 2 V.

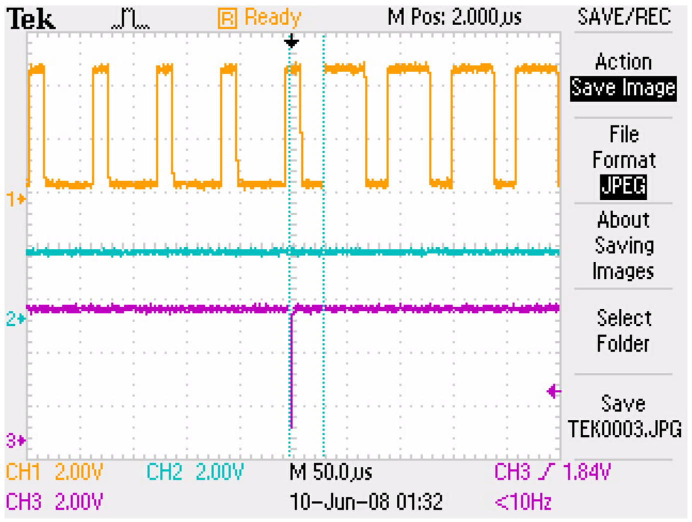

Figure 5. Change From 30% to 70% Duty Cycle

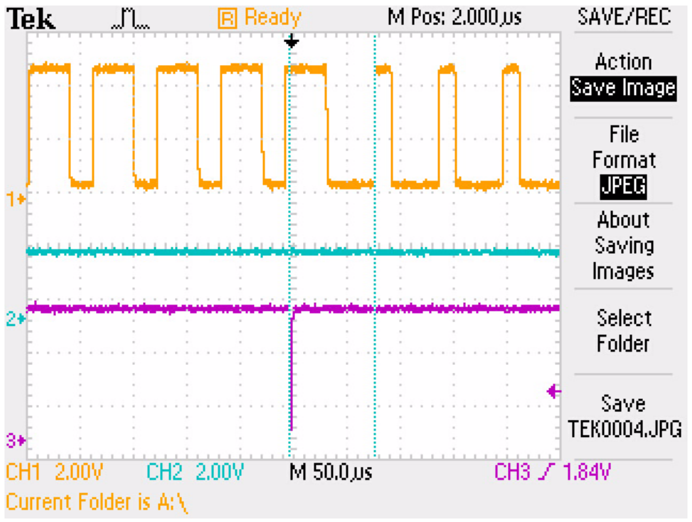

Figure 6. Change From 70% to 30% Duty Cycle

Figure 5 shows the PWM duty cycle change from 30% to 70% when the ADC comparator detects a change from a lower voltage to 2.5 V. Figure 6 shows the PWM duty cycle change from 70% to 30% when the ADC comparator detects voltage lower than 2.5 V. In both figures, from left to right, the first line marks when 2.5 V is measured and the second line shows the duty cycle change. The time difference between the first and second events is related to the FTM waiting for the next PWM period to reload the duty cycle, making the duty cycle change consistent. This example also uses the FTM channel to trigger the conversion.

The software used to generate these signals is included in the files that accompany this document.

4 Conclusions

- Hardware synchronization of ADC and timing modules allows control applications to work more precisely and frees processor resources to execute extra tasks that add value to the application.

- Better timing capability means higher control accuracy.
- The examples used in this document are simplified applications that make the advantage of the hardware approach more easily understood. More complex applications take more advantage of the functionality shown here. Some applications can only function successfully in this way, because software approaches are too slow or too imprecise.

5 References

- MCF51AC256RM — *MCF51AC256 Reference Manual*
- AN3729 — *Using FlexTimer in ACIM/PMSM Motor Control Applications*
- AN3732 — *Migrating Between MC9S08AC and MCF51AC Flexis Devices*

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3731
Rev. 0
06/2008