

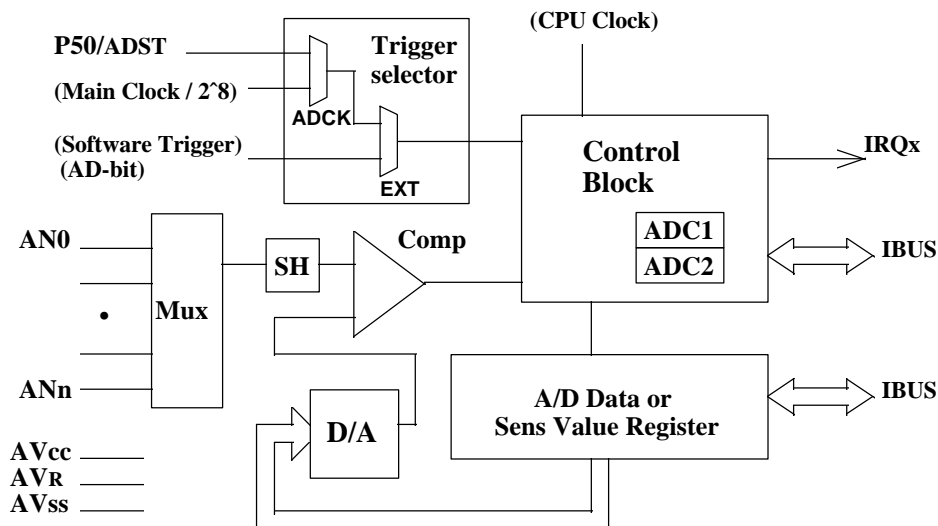
Notes on using the A/D Converter on the FFMC8L/LC Microcontrollers

© Fujitsu Mikroelektronik GmbH

Vers. 1.0 by E. Bendels

The following Application note is intended to give some hints on using the A/D converter on the FFMC8L/LC Microcontrollers.

Figure 1 shows a basic diagram of the A/D converter block.



The analog input signal (AN0..ANx) is passed via an analog multiplexer to a sample and hold stage and finally to a comparator. Analog to digital conversion is done by comparing the selected input signal with a signal derived from a D/A converter using the successive approximation approach. This means, once a conversion is started or triggered, the control block will first sample the selected input channel and keep the voltage level stable in the sample and hold stage. In the following conversion process, the control logic will start by setting the MSB in the A/D Data register, which is coupled to the D/A converter. The control logic can check via the comparator, if the sampled voltage is higher or lower. If it is lower, the bit in the A/D Data register is cleared again, otherwise it remains set. This process continues with all succeeding bits in the A/D Data register. When finished, the converted value can be read from the A/D data register.

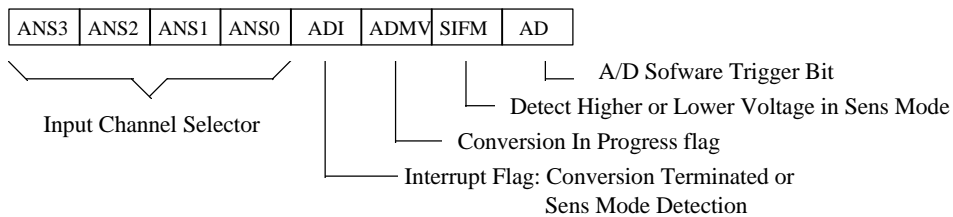
The A/D converter block can also be configured to operate in a so called Sense Mode in which a compare value is written by software into the A/D data register. In this case, the converter will simply determine if the input signal is higher or lower than the compare value.

Note that the A/D converter block has dedicated AV_{CC} , AV_{SS} and a AV_R inputs for better noise performance. AV_{CC} must be equal or higher than AV_R . The voltage applied between AV_R and AV_{SS} defines the voltage range which can be converted.

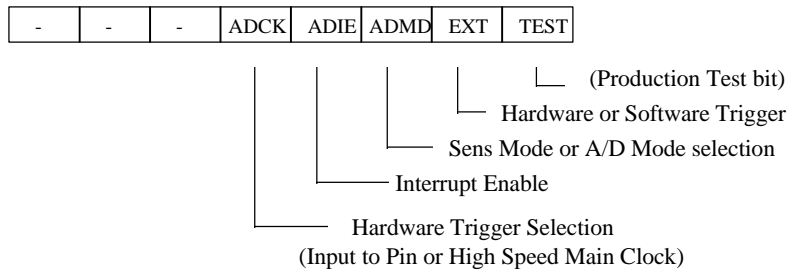
The Control Block contains registers for configuring and controlling the A/D converter operation. From a software point of view, there are various options on how to operate the A/D converter, like specifying the way how a conversion is started, how the end of a conversion is determined, and there are some peculiarities to take care about.

The following figure shows the A/D converter control registers of the MB89630 series. Some bits might differ on other series, but the basic operation is the same.

ADC1



ADC2



1) Triggering or Starting an A/D conversion

Assuming an input channel is selected by the input multiplexer, there are 3 ways different ways to actually start an A/D conversion.

a) Software Trigger:

If the Software Trigger is selected by setting the EXT-bit to 0, the A/D conversion can be started by setting the AD bit. This bit will be reset automatically.

b) Hardware Trigger (High Speed Main Clock)

If the Hardware Trigger selector is set to Main Clock (ADCK = 0) and the Hardware Trigger is active (EXT = 1), then the A/D converter is started periodically by a clock derived from the main clock oscillator.

Note: In this case, also the main-clock gear function must select the highest or 2nd highest main-clock speed, otherwise this trigger method will not work.

c) Hardware Trigger (External Input to Pin P50/ADST)

If the Hardware Trigger selector is set to the ADST Input pin (ADCK=1), and the Hardware Trigger is active (EXT = 1), then the A/D converter is started whenever there is a rising edge on this input .

On other controller series, instead of this external input signal, it is possible to use an on-chip timer to start the A/D converter.

2) Detecting the end of a conversion

a) Polling the ADMV bit

Probably the simplest way of detecting the end of an A/D conversion is to poll the ADMV bit.

If the A/D conversion was started by either the above mentioned methods, the ADMV is set to 1 as long as the conversion is in progress.

Once it becomes 0, the A/D Data register can be read out.

b) Polling the ADI bit

The ADI-bit could be used in a similar way as the ADMV-bit, but it has to be explicitly cleared before the conversion is started. Once the conversion is finished this bit becomes 1.

(If not reset by software, this bit would remain set even if a new conversion is started.)

If the ADI-bit is not actually used to generate interrupts, it is recommended to utilize the polling ADMV-bit method.

c) Interrupt Service Routine Call

If the A/D converter is configured to generate interrupts once a conversion is finished (ADIE = 1), and assuming the interrupt controller is initialized (Interrupt Level Register) and interrupts are globally enabled (SETI instruction), then the associated interrupt service routine (ISR) will be called at the end of a conversion.

Especially if the A/D conversion was triggered using 1b), it is recommended that even in the interrupt service routine, the ADMV bit is tested and waited to become 0.

The practice has shown that in this case, the ISR seems to be called before the conversion is completely finished !

Then the A/D Data register can be read out.

The ADI bit must be cleared before leaving the ISR.

3) Additional Notes

Attempts to read the A/D Data register before the conversion is finished will result in erroneous data.

Note that if the A/D conversion is started automatically on a periodic basis, the polling software or especially the ISR must be able to respond before the conversion is triggered the next time, otherwise it would read irrelevant data.

4) Sense Mode Operation

The sense mode operates in similar way to the above described A/D converter with some differences:

As an initialization task, a compare-value has to be written into the Sense Value Register (which is actually the same as the A/D Data register) to specify the compare voltage level.

The SIFM-bit specifies if the Sense Operation shall detect (set the ADI flag) if the selected analog input voltage is higher than the compare value, or shall detect if it is lower.

Note that the ADMV-bit has no function in the sense mode operation.

The sense mode is activated or triggered in the same way as described for the A/D converter mode. At each trigger (and only at each trigger), a compare operation is executed and the ADI-flag will be set if the condition specified by the SIFM bit is met.

Note that, if the condition was not met at one time and the input voltage has changed in the mean time such that the condition would meet now, it will require another trigger to actually set the ADI-flag.

Thus, the sense mode is mainly useful in one of the periodic hardware trigger modes 1b) or 1c).

5) Using the Sense Mode in a Power Down Mode

Probably the most interesting application for the Sense Mode is a voltage supervisory function which is still active in a power down mode.

If the Sense Mode detects a special condition, it will generate an interrupt and wake up the CPU core to take care about it.

Well, one requirement to do so, is that the A/D converter in Sense Mode is triggered on a periodic base to actually be able to detect the condition.

Thus, if the “Hardware Trigger by High Speed Main Clock” method is used the only possible power down mode would be the (high speed) Main-Sleep mode.

In all other power down modes this clock stops.

The alternative is the “Hardware Trigger (External Input)” method, which would also work in the Sub-Sleep mode.

In other controller series, eg.g. MB89160, instead of this external input, a 16-bit timer can be used to generate the trigger signal. Of course, then it also depends on this timer, in which power down modes it can operate.

6) Examples

Some example programs have been developed (initially used to figure out all these details mentioned above), based on the MB89630-EVAKIT and an extension board.

The program “ADC.c” demonstrates the case mentioned in 1a), .2a).

The program “ADCI.c” demonstrates the case mentioned in 1b). 2c)

The program “ADCE.c” demonstrates the case mentioned in 1c). 2c)

And finally “SENS.c” demonstrates the sense mode based on 1c) 2c)