Engine knock detection using AURIX™

About this document

# Scope and purpose

This application note describes how to implement knock detection system using DSADC in AURIX™ 32-bit microcontrollers from Infineon. This offers both the opportunity for cost reduction compared to some conventional systems, and the chance to reduce CPU load. A comparison is made between AURIX™ and the Infineon Audo device.

# Intended audience

This document is intended for software engineers who are interested in engine control systems involving AURIX™ devices. Engineers should have a good understanding of DSADC, Engine control and signal processing before reading this document.

For more detailed information please refer to the following Application notes:

* AP32343\_Position\_Minus\_Time\_Request\_PMTR
* AP32222\_DSADC\_basics
* AP32015\_ Engine Knock detection using TC-1796

Table of contents

[About this document 1](#_Toc461004863)

[Table of Contents 1](#_Toc461004864)

[1 What is engine knock? 3](#_Toc461004865)

[2 Signal processing basics 4](#_Toc461004866)

[2.1 Nyquist sampling theorem 4](#_Toc461004867)

[2.2 Aliasing 4](#_Toc461004868)

[2.3 Anti-aliasing 4](#_Toc461004869)

[2.4 A/D converter 5](#_Toc461004870)

[2.5 Advantage of oversampling converter 5](#_Toc461004871)

[2.6 Filter specification 6](#_Toc461004872)

[3 System concept 8](#_Toc461004873)

[3.1 Conventional system concept with Audo 8](#_Toc461004874)

[3.2 System concept with AURIX™ 8](#_Toc461004875)

[4 Code implementation 9](#_Toc461004876)

[4.1 Resources 9](#_Toc461004877)

[4.2 Code operation sequence 9](#_Toc461004878)

[4.2.1 Knock window 10](#_Toc461004879)

[4.3 DSADC configuration 10](#_Toc461004880)

[4.4 Group delay 11](#_Toc461004881)

[4.5 Offset compensation 12](#_Toc461004882)

[4.6 DMA 12](#_Toc461004883)

[4.7 Signal extract 12](#_Toc461004884)

[5 Performance test 14](#_Toc461004885)

[5.1 Input signal 14](#_Toc461004886)

[5.2 Analog to digital conversion validation test 14](#_Toc461004887)

[5.3 DSADC cut off frequncy validation test 14](#_Toc461004888)

[5.4 Signal extraction validation test 15](#_Toc461004889)

[6 Check point 17](#_Toc461004890)

[7 Driver API 18](#_Toc461004891)

[8 Driver files and folder structure 19](#_Toc461004892)

[9 References 20](#_Toc461004893)

[Revision History 21](#_Toc461004894)



# What is engine knock?

|  |
| --- |
| C:\D\Git\Knock_Dev\Appnote\pressurevstiming.png  C:\D\Git\Knock_Dev\Appnote\cylinder.png |

1. Engine ignition timing

Generally, advanced ignition timing generates high combustion pressure. But advanced timing cannot be used unconditionally in engine control, because advanced timing leads to [the phenomena](http://endic.naver.com/enkrEntry.nhn?entryId=4c022d97b37241898afa563fec31e052&query=knocking+phenomena) of engine knocking. Therefore to get high combustion pressure in the cylinder, finding the ‘correct’ timing is very important from an engine control point of view.

In system operation the ECU (Engine Control Unit) continuously tries to find the best timing. For example, the ECU will set the initial advanced ignition timing and then, if no knock signal is detected, the ECU will gradually increase the advanced ignition timing. However, if a knock signal is detected the ECU could set a much more retarded ignition timing compared with the advanced ignition timing.

In order to distinguish normal combustion and abnormal combustion, the ECU receives a signal via a piezoelectric sensor which provides a vibration signal. According to engine RPM, mechanical structure, and cylinder atmosphere, the knock signal has a specific frequency.

# Signal processing basics

## Nyquist sampling theorem

The Nyquist sampling theorem provides a prescription for the nominal sampling interval required to avoid aliasing. It may be stated simply as follows:

The sampling frequency should be at least twice the highest frequency contained in the signal.

Where:

* fs is the sampling frequency (how often samples are taken per unit of time or space).
* fc is the highest frequency contained in the signal.

## Aliasing

An alias is a duplicate. If the sample is equal to or below the Nyquist rate, the system will be faced with a problem. In the interpolation process, which signal in the graph below is more likely to be reproduced? From the dots, both are valid signals. Two waves hide in the same bunch of samples. This situation is called aliasing.

|  |
| --- |
| C:\D\Git\Knock_Dev\Appnote\aliasing.png |

1. Aliasing

## Anti-aliasing

Input signals Nyquist frequency cannot be properly converted and they also create new signals in the base-band, which were not present in the original signal. This non-linear phenomenon is a signal distortion frequently referred to as an aliased signal.

|  |
| --- |
| C:\D\Git\Knock_Dev\Appnote\aliased.png |

1. Aliased signal

Distortion can only be prevented by running the input signal through a low-pass filter, up to the Nyquist frequency. This low-pass filter, sometimes called the anti-aliasing filter, must have a flat response over the frequency band of interest (baseband) and attenuate the frequencies above the Nyquist frequency enough to put them under the noise floor.

|  |
| --- |
| C:\D\Git\Knock_Dev\Appnote\antialiased.png |

1. Anti -aliasing

## A/D converter

Most A/D converters can be classified into two groups according to the sampling rate criteria.

Nyquist rate converters (for example, Fast ADC in the Infineon Audo product) sample analog signals which have maximum frequencies slightly less than the Nyquist frequency:

fN= fs/2

where:

* fs is the sampling frequency.

Oversampling converters (for example, DSADC in Infineon AURIX™ products) perform the sampling process at a much higher rate:

fN << Fs

where:

* Fs denotes the input sampling rate.

## Advantage of oversampling converters

The following figure shows the requirements of the anti-aliasing filter and over-sampled Nyquist rate A/D converters. Sampling at the Nyquist rate mandates the use of an anti-aliasing filter with very sharp transition in order to provide adequate aliasing protection without compromising the signal bandwidth .

The transition band of the anti-aliasing filter of an oversampled A/D converter is much wider than its passband, because anti-aliasing protection is required only for frequency bands between and .

One R-C low-pass ﬁlter is sufﬁcient for the oversampled A/D converter anti-aliasing ﬁlter.

|  |
| --- |
| C:\D\Git\Knock_Dev\Appnote\Advatage.png |

1. Advantage of over-sampling converter

## Filter specification

DSADC has the following filter response specification:

|  |
| --- |
|  |

1. DSADC frequency response

When compared to the FADC, DSADC has the superior filter response. Specifically, without compensation software FIR, the DSADC shows a flat passband when compared to the FADC.

|  |
| --- |
|  |

1. DSADC passband

|  |
| --- |
|  |

1. FADC passband

# System concept

## Conventional system concept with Audo

Using FADC in Audo, the following concept can be configured:

|  |
| --- |
| C:\D\Git\Knock_Dev\Appnote\convention.png |

1. Audo system concept

## System concept with AURIX™

Compared with the FADC concept, the external analog filter and software AAF are removed in the DSADC system concept, because DSADC provides enough filter performance to implement a knock detection system.

An AURIX™ DSADC implementation offers the opportunity for cost reductions and to lower the CPU load.

|  |
| --- |
| C:\D\Git\Knock_Dev\Appnote\DSADC.png |

1. AURIX™ system concept

# Code implementation

1. Code examples are provided ONLY for illustration purposes and are not intended to be ‘production’ ready. Any use of these code examples is entirely at your own risk.

## Resources

The main resources to implement Knock detection in the code are as follows:

* ATOM2\_CH6 in GTM
* DSADC CH0
* DMA CH0
* Port pin 32.0 (Knock window check)
* FIR filter in DSP library

|  |
| --- |
| C:\D\Git\Knock_Dev\Appnote\code implementation.png |

1. Resources for knock detection

## Code operation sequence

The overall system operation is as follows:

|  |
| --- |
| C:\D\Git\Knock_Dev\Appnote\Code operation sequence.png |

1. Code operation sequence

### Knock window

In the system the data gathering period is known as the ‘knock window open’ status. The knock window in the AURIX™ is implemented by ATOM in the GTM.

The ATOM module plays a role as a gate trigger. During the knock window open, DSADC converts the analog to digital data. The knock window open period, and the start and close points, are decided by the application (KnoCheck\_ASWControl API) and will depend on the internal atmosphere of the engine.

|  |
| --- |
|  |

1. Knock window

## DSADC configuration

Here the target output pass band is set as 104.17KHz, where = 100MHz.

To get the intended passband OSR (Oversampling Ratio) DSADC can be configured as shown in the following table.

The CIC (Cyclic Integration Comb) filter OSR is configurable by the user, but two FIR filters in DSADC have a fixed decimation rate with 2 respectively.

|  |
| --- |
| image005 |

1. Pass band configuration

## Group delay

Data that enters a digital filter needs a certain number of filter clocks before it appears at the filter’s output, where it can be used by the system. The effective group delay depends on the configuration of the filter chain. The AURIX™ user manual provides a summary table with respect to group delay. When considering group delay, data which comes in to the DSADC from the ‘knock window open’ until meaningful data is generated, should be discarded.

|  |
| --- |
| C:\D\Git\Knock_Dev\Appnote\gropu delay.png |

1. Group delay

|  |
| --- |
| C:\D\Git\Knock_Dev\Appnote\gropu delay2.png |

1. Group delay consideration in code

## Offset compensation

Fundamentally DSADC has deterministic offset. The following table demonstrates deterministic offset.

The value can vary +/-35. Each user must consider the offset when calibrating the offset value.

According to Aurix device step, offset is different. For more specific information please refer to Aurix data spec.

|  |
| --- |
| C:\D\Git\Knock_Dev\Appnote\offset.png |

1. DSADC internal offset(TC27x B&C steps)

## DMA and double buffering to reduce CPU load

Digital Signal Processing arithmetic takes a long time. To distribute the CPU load, DSP calculations only occur after 128 bits of data have entered the RAM buffer. For effective data handling a ‘double buffering’ function inside the DMA is used. Double buffering provides automatic buffer change functionality and means that during the calculation phase, while working on the available set of data, more incoming data can still be safely stored.

|  |
| --- |
| C:\D\Git\Knock_Dev\Appnote\Double buffering.png |

1. DMA double buffering concept

## Signal extract

FIR filter software is needed to get a target frequency signal. Infineon provides a DSP library. To extract the signal which has the target frequency the FIR filter API Ifx\_firRealQ15a is used. To use this API the filter should have a multiple of 8 coefficients. It takes 115.326us to calculate 128 bits of data (AURIX™ TC277).

|  |
| --- |
|  |

1. FIR filter calculation points

After extraction the signal which has the knock frequency is then manipulated.

* The signed signal is first converted to an absolute signal.
* The absolute signal is added to get an integration value.
* The more digitalized the signals are in pass band, the greater the integration value.

|  |
| --- |
| C:\D\Git\Knock_Dev\Appnote\integration.png |

1. Integration value

# Performance test

## Input signal

A sine wave signal is used for the test.

# According to the test, sinwave has following frequencys:

* 11KHz sinewave (Analog to digital conversion validation test)
* 0~200Khz sweeped sinewave for 4s(DSADC cut-off frequency validation test)
* 0~104Khz sweeped sinewave for 7s(Signal extraction validation test)

|  |
| --- |
| C:\D\Git\Knock_Dev\Appnote\InputSignal.png |

1. Test input signal

## Analog to digital conversion validation test

As shown in the figure below, the sine signal is digitalized. A digitalized signal can be checked though the variable Buffer0\_Dest, Buffer1\_Dest. An 11 KHz signal is used for the test in this figure.

|  |
| --- |
| C:\D\Git\Knock_Dev\Appnote\Digitalfsignal.png |

1. Analog to digital conversion validation test(Sinewave with 11KHz)

## DSADC cut-off frequency validation test

The target cut-off DSADC frequency is 104.17KHz in this system. The following figure shows the integration value.

For the test, a sine-wave signal which sweeps from 0 to 200KHz for 4s is used.

The cut-off frequency can be checked in the second picture below. The integration value dramatically drops down from around the 104Khz.

|  |
| --- |
| C:\D\Git\Knock_Dev\Appnote\CutoffFreqTest.png |

1. Cut-off frequency validation test

The next figure shows a re-constructed sinewave signal which has 130 KHz. The re-constructed sinewave signal is very different from the original.

|  |
| --- |
| C:\D\Git\Knock_Dev\Appnote\130KHz.png |

1. Sinewave with 130KHz

## Signal extraction validation test

For the test, a 30 KHz(+-2KHz) Bandpass filtercoefficient is used. The following figure shows the signal extraction result.(Filter type, frequency, filtering range can be changed based on project)

A 30 KHz sinewave signal has highest integration value.

|  |
| --- |
| C:\D\Git\Knock_Dev\Appnote\extract.png |

1. Signal extraction validation test

# Check points

The operating situation can be easily checked through the available variables.

# Check points

* Buffer0\_Dest
* Buffer1\_Dest
* Test\_Integrated
* Knocheck\_Integrated

1. If you are using a T32 debugger the following commands are recommended:

// v.w Buffer0\_Dest Buffer1\_Dest Test\_Integrated Knocheck\_Integrated

// v.Draw Buffer0\_Dest

// v.Draw Buffer1\_Dest

Variable Buffer0,1\_Dest shows the dizitalized knock signal.// v.Draw Test\_Integrated[0]

// v.Draw Test\_Integrated[1]Variable Test\_Integrated shows the filtered which is done by SW and integrated signal. Filter frequency is decided by variable FilterIndices[i]. Test\_Integrated[0] shows the bypassed wich means SW filter is not adjust and integrated signal in the flowing figure. Test\_Integrated[1] shows bandpass filtered which has 30KHz and intergrated signal in the flowing figure. For more detailed information please refer to ”DSADC cut-off frequency validation test” and “Signal extraction validation test”section

|  |
| --- |
| C:\D\Git\Knock_Dev\Appnote\screenshot.png |

1. Check point window

# Driver API

1. Because this document does not focus on the DPLL implementation, any comments only relate to the Sync point related API in the DPLL driver.
2. DPLL driver API

| Function | Signature | Description |
| --- | --- | --- |
| SyncPoint\_TE3\_ISR | IFX\_INTERRUPT(SyncPoint\_TE3\_ISR, 0, TE3\_INT\_NUM) | Sync point is generated every 180°. |

1. Knock driver API

| Function | Signature | Description |
| --- | --- | --- |
| Knocheck\_Ini | void Knocheck\_Ini(void); | Initialize Knock window, gain, Integration value. |
| KnoCheck\_Dma\_Ini | void KnoCheck\_Dma\_Ini(void); | Initialize Konck related DMA function |
| KnoCheck\_Dsadc\_Ini | void KnoCheck\_Dsadc\_Ini(void); | Initialize Konck related DSADC function |
| KnoCheck\_ASWControl | void KnoCheck\_ASWControl(void) | Called at Every Sync position.  Set the gain and Knock window open ,close position |
| KnoCheck\_Filter | void KnoCheck\_Filter(void); | Calculate FIR filter |
| KnoCheck\_Memcopy | void KnoCheck\_Memcopy(void); | Copy the DMA buffer value to the Ram region |
| KnoCheck\_ClearCoeffDly | void KnoCheck\_ClearCoeffDly(void); | After integration, clear the buffer in FIR filter |
| KnoCheck\_MemCopy64 | void KnoCheck\_MemCopy64(uint64\* xDest\_pu64,uint64\* xSrc\_pcu64, uint16 numBytes); | Copy the DMA buffer value to the Ram region |
| knoCheck\_GetIntegData | void knoCheck\_GetIntegData(uint8 Ch, sint32\* IntegratedValues,uint8 ValueLength); | Provide Integration result value |

# Driver files and folder structure

An outline of the driver file structure:

1. Folder structure

| Folder structure | Module | Description |
| --- | --- | --- |
|  | DPLL | Initialize and operate DPLL functionality in GTM.  This module only covers simple DPLL function. |
| DspLiba | Infineon DSP Library |
| Knock | Knock function code. |

# References

1. AP32343\_Position\_Minus\_Time\_Request\_PMTR
2. AP32222\_DSADC\_basics
3. AP32015\_ Engine Knock detection using TC-1796
4. http://wolfcrow.com/blog/professor-samplers-notes-aliasing
5. <https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem>

Revision history

# Major changes since the last revision

| Page or Reference | Description of change |
| --- | --- |
| V1.0 | First release |
|  |  |
|  |  |