Freescale Semiconductor
Application Note

AN3226
Rev. 0, 2/2006

XGATE Library: ATD Average

Calculating a rolling average from ATD results

by: Steve McAslan
MCD Applications, East Kilbride

1 Introduction

The XGATE driver implements a simple averaging filter based on up to 64 consecutive analog-to-digital (ATD) values. At each sample time, the driver collects a new ATD conversion result and calculates a rolling average over all the most recent results. To optimize performance, the driver uses integer arithmetic only.

The XGATE handles all interrupt requests from the ATD and optionally interrupts the CPU when a new average value is available.

2 Principle of Operation

In real-world applications there are occasions when it may be inappropriate to use a particular ATD conversion result. Although the result itself is a correct conversion of the analog value that existed at the input, it may be that an external disturbance has caused it to be unrepresentative of the signal of interest. This is particularly of concern where a short series of results is disturbed by some undesired noise at the input.

Table of Contents

1 Introduction 1
2 Principle of Operation 1
3 Configuration 4
3.1 XL_ATD0_AVERAGE 4
3.2 XL_ATD1_AVERAGE 4
3.3 XL_ATD0_AVERAGECHANNELS 4
3.4 XL_ATD1_AVERAGECHANNELS 4
3.5 XL_ATD0_AVERAGEn 4
3.6 XL_ATD1_AVERAGEn 5
3.7 XL_ATD0_AVERAGETAPS 5
3.8 XL_ATD1_AVERAGETAPS 5
3.9 XL_ATD0_AVERAGEINTERRUPT 5
3.10 XL_ATD1_AVERAGEINTERRUPT 5
4 Driver Performance 6
4.1 Notes on Performance Specification 6
5 Driver Implementation 8
6 Software Example 8
7 Enhancements 10

freescale™ semiconductor

One method of addressing this problem, and of rejecting the disturbance, is to average the incoming ATD values and so extract the longer-term value of the signal. This is most effective when the desired signal changes slowly. The averaging can be applied over a number of samples, and the output based on the most recent results. Each time a result is taken from the ATD module, the oldest previous result is deleted, and the average is recalculated using the remaining results and the newest value.

Technically, the averaging procedure is a simple digital filter with a finite impulse response (FIR). The simplicity of the filter is due to the fact that each sample is multiplied by a unity filter coefficient, and so the overhead for the calculation is reduced from the typical “multiply and accumulate” to simply “accumulate”. The final step in the FIR is to normalize the result, which is done by dividing the calculated result by the sum of filter coefficients, which, in this case, is simply the number of samples to be averaged.

An FIR with this configuration is usually referred to as a “boxcar filter” (see Figure 1) and has a frequency response that gives a general low-pass characteristic, but with lots of side lobes (see Figure 2). This has the effect of preserving the desired, slowly changing (low frequency) signal in the presence of sudden changes (high frequency noise). However, it is important to bear this frequency response in mind when considering the type of noise that the averaging filter can reject.

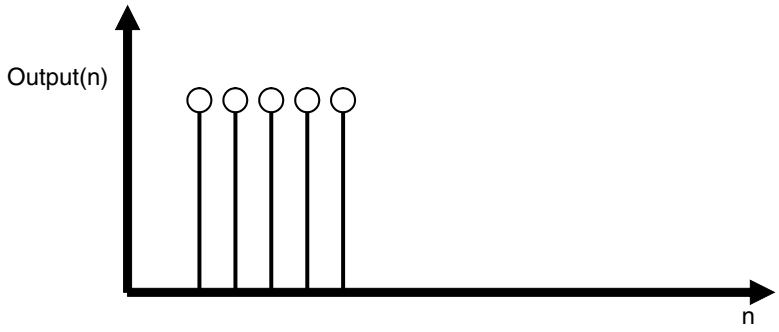

Figure 1. Impulse Response of Boxcar Filter

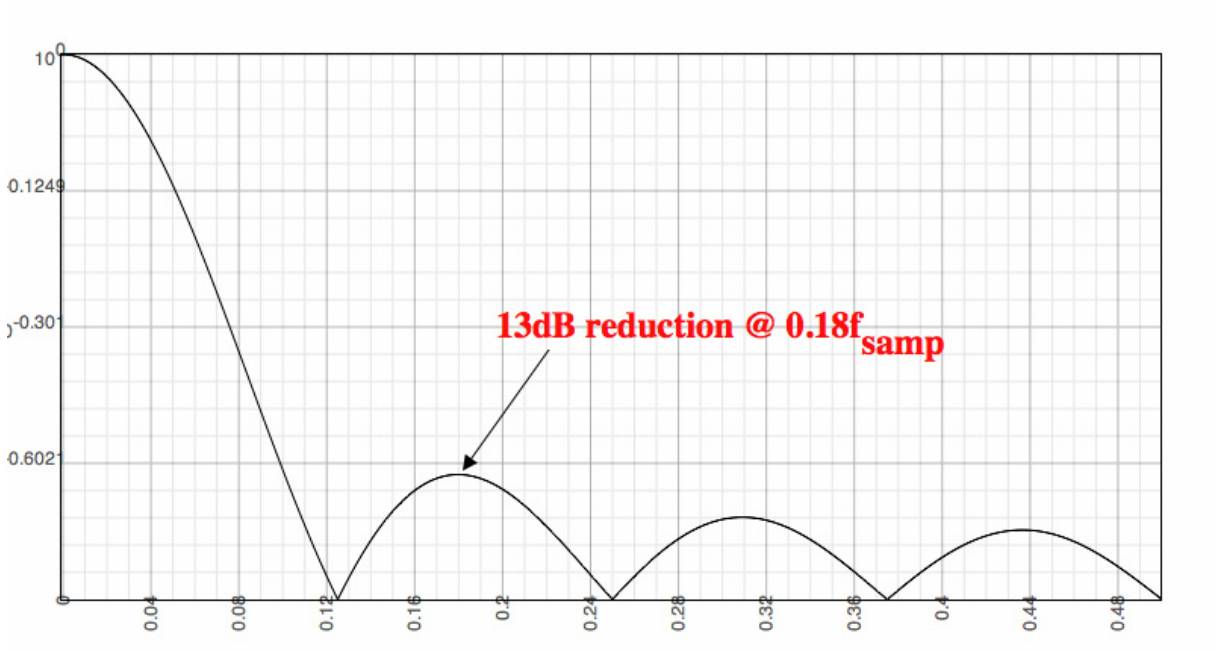

Figure 2. Frequency Response of 8-step Boxcar Filter

The driver is based on an XGATE thread that acts on a single ATD module at a time. The thread takes a pointer to a structure that defines the behavior of the driver. The structure specifies the ATD module in use, the channels of that module to process, the number of samples to filter, the accumulated ATD result buffers for each channel, and the final filtered outputs. This approach allows the XGATE to create filtered values for any combination of channels on any ATD module on the MCU.

The thread can be activated by any interrupt source; however, in most cases, the source will be the ATD module sequence complete flag. This ensures that the values in the ATD results registers will be as up-to-date as possible. In the accompanying example, two ATD modules are directly triggered by the periodic interrupt timer (PIT) every 100 µs, and the ATD sequence completion flags launch the thread. This equates to a sampling rate of 10 kHz, which allows several channels to be converted on each ATD module.

The driver data structure allows complete freedom in the choice of channels on which to perform the filtering. Each channel is allocated a data buffer large enough to store the number of samples in the filter calculation. The only limitation is that the number of samples in the buffer must be the same for all channels on single ATD module. Different ATD modules can have different buffer lengths.

The filter is compatible with 8-bit or 10-bit results, but the data must be right-aligned for correct calculation.

The CPU retrieves the output of the filter by reading a result from an array xl_atd0_FilterResults for ATD0 and xl_atd1_FilterResults for ATD1. In this implementation, each ATD conversion result is also available until it is no longer part of the average calculation. Finally, the XGATE thread can be configured to send an interrupt to the CPU on completion of the filtering operation.

3 Configuration

The filter configuration is defined in the xl_atd_filterconfigure.h file. There are up to 42 macros to configure. The maximum values for some of the filters are dependent on the number of channels available on each ATD module. For the purposes of this driver, the maximum values apply to the MC9S12XD512 MCU.

3.1 XL_ATD0_AVERAGE

If this macro is defined, the software will create filters for the ATD0 module. This macro does not require any explicit value.

```
Example: #define XL_ATD0_AVERAGE /* Create filter */
```

3.2 XL_ATD1_AVERAGE

If this macro is defined, the software will create filters for the ATD1 module. This macro does not require any explicit value.

```
Example: #define XL_ATD1_AVERAGE /* Create filter */
```

3.3 XL_ATD0_AVERAGECHANNELS

This macro defines the number of ATD0 channels that the software will filter. The value of this macro must be between 1 and the maximum number of channels on ATD0 (8).

```
Example: #define XL_ATD0_AVERAGECHANNELS 2 /* 2 filters needed on ATD0 */
```

3.4 XL_ATD1_AVERAGECHANNELS

This macro defines the number of ATD1 channels that the software will filter. The value of this macro must be between 1 and the maximum number of channels on ATD1 (16).

```
Example: #define XL_ATD1_AVERAGECHANNELS 10/* 10filters needed on ATD1 */
```

3.5 XL_ATD0_AVERAGEn

These macros define the ATD0 channel associated with each filter. There are up to eight macros for ATD0 from XL_ATD0_AVERAGE0 to XL_ATD0_AVERAGE7. The number of macros required equals the value of XL_ATD0_AVERAGECHANNELS. For example if XL_ATD0_AVERAGECHANNELS equals 4, then there must be four macros named XL_ATD0_AVERAGE0, XL_ATD0_AVERAGE1, XL_ATD0_AVERAGE2, and XL_ATD0_AVERAGE3. The value of each macro must be between 1 and the maximum number of channels on ATD0 (8).

```
Example: #define XL_ATD0_AVERAGE0 5 /* Filter 0 reads from ATD channel 5 */
```

3.6 XL_ATD1_AVERAGEn

These macros define the ATD1 channel associated with each filter. There are up to sixteen macros for ATD1 from XL_ATD1_AVERAGE0 to XL_ATD1_AVERAGE15. The number of macros required equals the value of XL_ATD1_AVERAGECHANNELS. For example if XL_ATD1_AVERAGECHANNELS equals 4 then there must be four macros named XL_ATD1_AVERAGE0, XL_ATD1_AVERAGE1, XL_ATD1_AVERAGE2, and XL_ATD1_AVERAGE3. The value of each macro must be between 1 and the maximum number of channels on ATD1 (16).

```
Example: #define XL_ATD0_AVERAGE0 5 /* Filter 0 reads from ATD channel 5 */
```

3.7 XL_ATD0_AVERAGETAPS

This macro defines the number of values to include in the average calculation. If this value is the same as the value of XL_ATD1_AVERAGETAPS, then the driver may use a binary shift to perform the normalization (if a binary shift is possible).

```
Example: #define XL_ATD0_AVERAGETAPS 4 /* Create an average from 4 values */
```

3.8 XL_ATD1_AVERAGETAPS

This macro defines the number of values to include in the average calculation. If this value is the same as the value of XL_ATD0_AVERAGETAPS, then the driver may use a binary shift to perform the normalization (if a binary shift is possible).

```
Example: #define XL_ATD1_AVERAGETAPS 10 /* Create an average from 10 values */
```

3.9 XL_ATD0_AVERAGEINTERRUPT

When defined as TRUE, this macro causes the driver to send an interrupt to the CPU after it has calculated all of the filters on ATD0. If interrupts are not required on completion, then this macro should be set to the value FALSE.

```
Example: #define XL_ATD0_AVERAGEINTERRUPT TRUE /* Interrupt CPU at each new value */
```

3.10 XL_ATD1_AVERAGEINTERRUPT

When defined as TRUE, this macro causes the driver to send an interrupt to the CPU after it has calculated all of the filters on ATD1. If interrupts are not required on completion, then this macro should be set to the value FALSE.

```
Example: #define XL_ATD1_AVERAGEINTERRUPT FALSE /* No interrupt for the CPU */
```

4 Driver Performance

Table 1. Required Resources

Parameter	Value
Required peripheral use	ATD channels as required
Optional peripheral use	PIT channel

Table 1 shows the on-chip resources required for operation of this driver. (If a specific sampling frequency is required then the ATD module must be triggered using the PIT.) Table 1 indicates how execution time and XGATE load vary across the number of channels and the number of samples in the buffer. The memory footprint data has been extracted from the map-file provided by CodeWarrior Development Studio for Freescale S12X 4.5.

4.1 Notes on Performance Specification

The driver will automatically use a binary shift instead of an integer division, where possible, and this affects both the code size and performance of the driver. To take advantage of this optimization, choose a sample buffer size of 2n for both ATD modules.

The load percentage shown assumes a sampling interval of 100 µs (10 kHz) and a bus speed of 40 MHz. In all examples, the XGATE causes an interrupt on the CPU once the filter is complete. All performance data exclude the initialization code and data requirements, since typically these are not relevant to the execution of the application, and can be executed by the CPU rather than the XGATE.

Table 2. Performance Considerations @ fbus = 40 MHz

ATD0		ATD1		Performance				
Channel	Buffer Depth	Channel	Buffer Depth	Code Size (bytes)	Data Size (bytes)	Maximum Latency	Maximum Execution Time	Load
2	4	0	N/A	154	40	95.8 µs	4.2 µs	3.9%
2	8	0	N/A	154	56	93.9 µs	6.1 µs	5.7%
2	4	1	4	154	62	~ 92 µs	4.4 µs	6.2%
2	8	4	16	196	320	~ 60 µs	29.3 µs	37.5%
0	N/A	4	4	154	76	92.1 µs	7.9 µs	7.4%
0	N/A	4	8	154	108	88.2 µs	11.8 µs	11.1%
4	4	2	4	154	116	~ 85 µs	8.2 µs	11.6%
4	8	4	16	196	280	~ 50 µs	29.3 µs	47.3%
8	4	0	N/A	154	148	84.7 µs	15.3 µs	14.2%

Table 2. Performance Considerations @ fbus = 40 MHz

ATD0		ATD1		Performance				
Channel	Buffer Depth	Channel	Buffer Depth	Code Size (bytes)	Data Size (bytes)	Maximum Latency	Maximum Execution Time	Load
8	16	0	N/A	154	340	61.3 µs	38.7 µs	36.3%
8	4	10	8	196	412	~ 20 µs	52.1 µs	80.2%
8	8	10	8	154	476	~ 50 µs	29.8 µs	50%
8	16	4	32	196	640	~ 0 µs	57.6 µs	96.8%

5 Driver Implementation

The driver contains an initialization thread that completes the configuration of the driver. This function, called xl_atd_initaverage, attaches the ATD channel in use to each buffer and also attaches the result buffers. By convention, this thread is called using software interrupt 0 (see AN3145) and is run once only.

The main thread xl_atd_average performs the averaging function. The function begins by fetching a pointer to the first ATD filter. Next, it fetches the latest ATD result from the channel associated with the current filter structure and stores this value in the results buffer. The next step is to compute the latest output from the averaging filter, which it does by adding the contents of the sample buffer. This new total is then normalized by dividing by the number of entries in the buffer.

The algorithm allows an optimization if both ATD filters use the same number of taps. In this case, the C macro value is used in place of a C variable, which means that, if a binary division is possible, then the compiler will make this step very efficient by using a single cycle XGATE shift opcode. Finally, the new filter output is written into the result register for use by the CPU.

NOTE

After initialization, the filter takes a delay equal to the number of results in the buffer multiplied by the sample time before the first filtered result is available.

If the filter is stopped, then the filter output for the same delay time will be incorrect.

6 Software Example

The driver is provided along with a software example that demonstrates how it may be used. In the example, both ATDs are enabled and a PIT channel provides the timing references for the ATD modules (see Figure 3).

The CPU can access the latest filter output for any channel, as shown in Figure 4. Finally, if the XGATE sends an interrupt to the CPU, then its interrupt flag must be cleared before the CPU completes the interrupt routine, as shown in Figure 5.

```
  /* Initialise the ATD to application requirements */
  ATD0.atdctl0.byte = 0x03;            // 4 lowest channels
  ATD0.atdctl1.byte = 0x82;            // ATD triggered from PIT0

// Turn on ATD & use external trigger
  ATD0.atdctl2.byte = ADPU|AFFC|ETRIGE|ASCIE;
  ATD0.atdctl3.byte = 0x20;            // 4 conversions - no FIFO

// For 40MHz bus, conversion rate = 2MHz, 2cycle sample, 10bit
accuracy  ATD0.atdctl4.byte = 0x09;

// Left justified, unsigned, scan multiple channels starting with 0
  ATD0.atdctl5.byte = 0x90;

  ATD1.atdctl0.byte = 0x07;            // 8 lowest channels
  ATD1.atdctl1.byte = 0x82;            // ATD triggered from PIT0
// Turn on ATD & use external trigger
  ATD1.atdctl2.byte = ADPU|AFFC|ETRIGE|ASCIE;
  ATD1.atdctl3.byte = 0x38;            // 8 conversions - no FIFO

// For 2MHz bus, conversion rate = 1MHz, 2cycle sample, 10bit accuracy
  ATD1.atdctl4.byte = 0x00;

// Left justified, unsigned, scan multiple channels starting with 0
  ATD1.atdctl5.byte = 0x90;

  /* Enable PIT 0 to provide 100us trigger to ATDs */
  PIT.pitcflmt.byte = 0x80;            // Enable PIT
  PIT.pitce.byte = 0x01;               // Enable channel 0
  PIT.pitmux.byte = 0x00;              // Use microtimer 0 for all
  PIT.pitinte.byte = 0x00;             // Disable interrupts
  PIT.pitmtld0.byte = 20;              // Microtimer count of 20
  PIT.pitld0.word = 200;               // Channel 0 counter of 200*/
```

Figure 3. Initialization of ATD and PIT modules

```
    resultATD0= xl_atd0_FilterResults[0];
    resultATD1= xl atd1 FilterResults[0];
```

Figure 4. Example of CPU Code Required to Access the Filter Output Values

```
#pragma CODE_SEG __NEAR_SEG NON_BANKED
interrupt void Filtered0(void)
{
  volatile unsigned int first_result= xl_atd0_FilterResults[0];
  // Clear interrupt - channel 69
  XGATE.xgif.bit.bits_68_6F.XGIF_69 = 1;
}
```

Figure 5. Example CPU Interrupt Service Routine for ATD0 Filter

7 Enhancements

It is possible to enhance the performance of the filter, by placing a number of filters in series such that, instead of fetching the latest result from an ATD channel, the result of another filter can be used. The resultant impulse response is the convolution of the two filters. By placing one boxcar filter immediately after another, the impulse response of the overall system becomes a triangle filter. See Figure 6 for the impulse response and Figure 7 for the frequency response of a triangle filter. In practice, this gives improved low-pass filter behavior at the expense of twice the latency of a single filter.

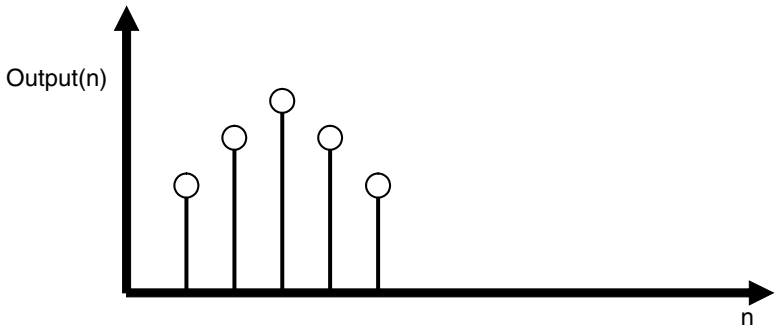

Figure 6. Triangle filter impulse response

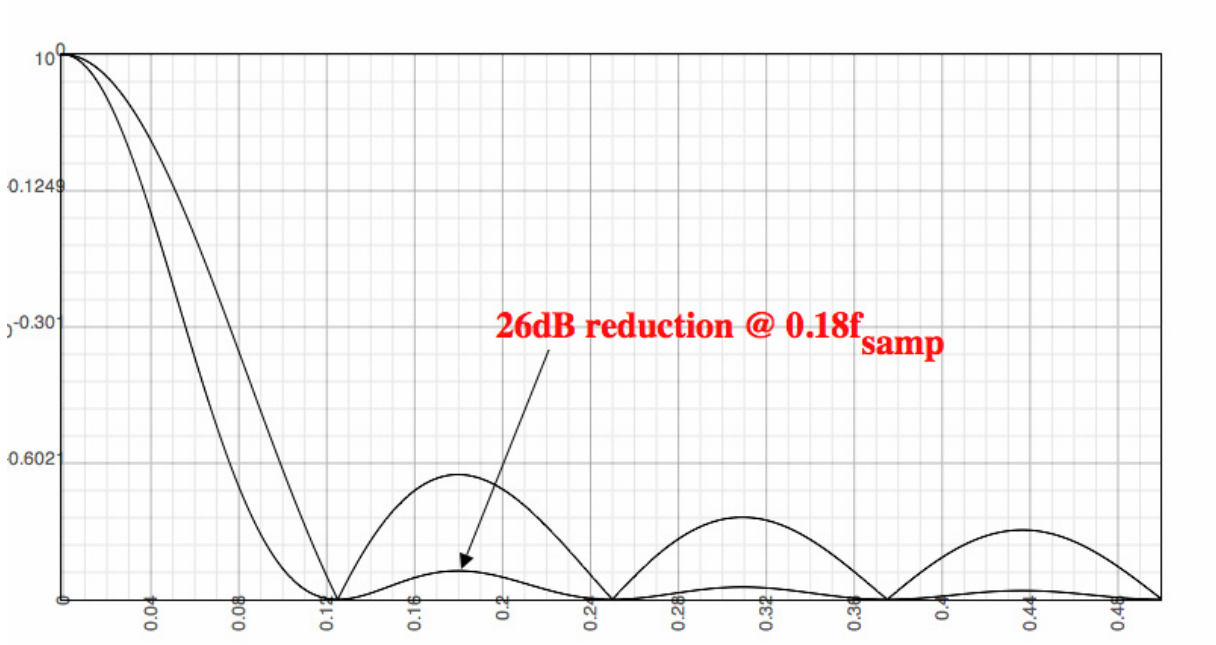

Figure 7. Triangle filter vs. Boxcar filter frequency response

For filters with longer sample buffers, it may be more efficient to subtract the oldest value and add the newest value to the buffer total, rather than calculate a new total every time a new sample is available.

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

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)
support@freescale.com

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 Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

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

AN3226
Rev. 0, 2/2006

freescale™
semiconductor