UT32M0R500 32-bit Arm™ Cortex® M0+ Microcontroller - ADC Unit

PRODUCT NAME	MANUFACTURER PART NUMBER	SMD #	DEVICE TYPE	INTERNAL PIC NUMBER
Arm Cortex M0+	UT32M0R500	5962-17212	ADC Unit	QS30

Table 1: Cross Reference of Applicable Products

1.0   Overview

To interface to the "natural world", which is analog in nature, the UT32M0R500 provides a Delta- sigma ADC. The Delta-sigma ADC accumulates the difference between the analog input and the measured voltage/current at the summing junction. The ADC output is decimated and filtered by one of two Digital Decimation Filters (DDF) to produce the 12-bit wide parallel data output.

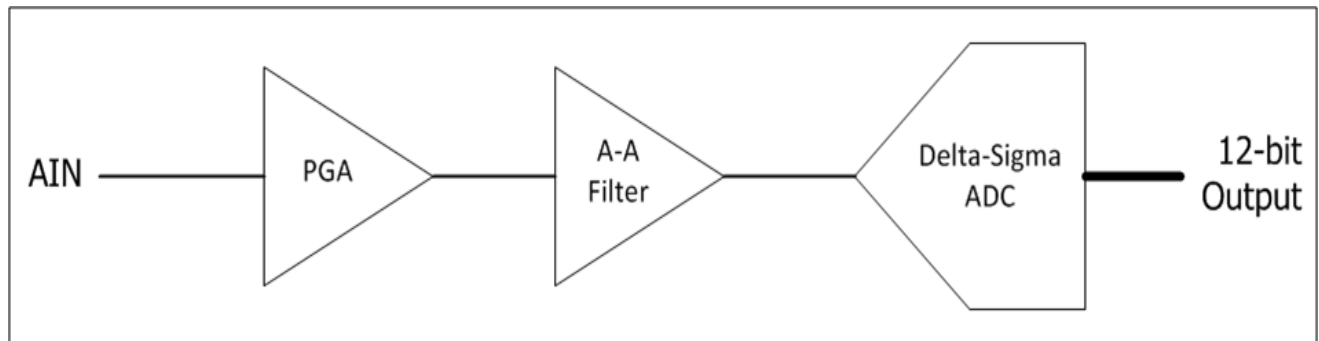Figure 1 shows the basic diagram of a Delta-Sigma ADC channel output.

Figure 1: ADC basic diagram

The Modulator converts the input signal to a bit stream. The bit stream is clocked by a multiple rate-

-the oversample rate (OSR), of 2fmax (Nyquist frequency). The modulator output bit stream is followed by a digital filter, COI3 or SINC4 for the UT32M0R500 ADC module, see Figure 2.
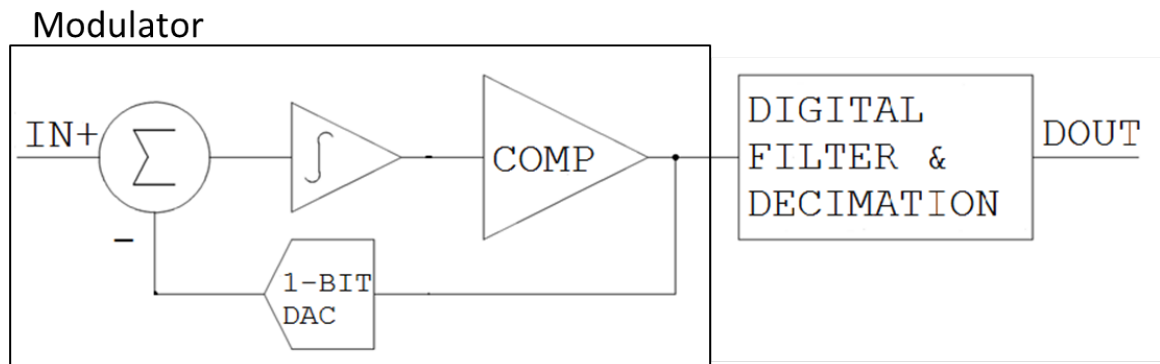

Figure 2: Modulator and Filtering

CAES
PIONEERING
ADVANCED
ELECTRONICS

UT32M0R500 32-bit Arm™ Cortex® M0+ Microcontroller - ADC Unit

1.1 ADC Single Chip Selection

To keep precision and less chip area size, the UT32M0R500 has only one ADC, but it has an analog mux with 16 analog inputs configured as either single-ended or up to 8 differential input pairs. A temperature sensor is provided as an additional 17th selectable input to the ADC.

1.2 ADC Quantization

The UT32M0R500 Sigma-Delta ADC has a sampling depth of 12 bits. The 12-bit ADC quantizes the input signal to 2^12 levels. To recover the signal, the sampling rate, fsamp, must be at least twice the maximum input frequency, fmax. This is known as the Nyquist sampling theorem. The UT32M0R500 Sigma-Delta ADC provides an "anti-alias filter" for removing signals above fmax, and therefore, guarding against false signals or "aliasing", see Figure 1.

The ADC result can be calculated as follows:

$$Digital\ value = round(2^n * \frac{V - Vrl}{Vrh - Vrl})$$

Where Vrl is 0 Volts or Ground for single ended, it becomes:

$$Digital\ value = round(2^n * \frac{V}{Vrh})$$

1.3 ADC Sampling

The ADC samples the continuous analog signal at fixed time intervals, therefore, the ADC result has sampling errors. The errors are divided into DC and AC. Some of the common DC errors are Offset, Gain, DNL (Differential Non-Linearity) and INL (Integral Non-Linearity), see Figure 3. Some of the common AC errors are THD (Total Harmonic Distortion), SNR (Signal To Noise Ratio), SFDR (Spurious Free Dynamic Range), SINAD (Signal To Noise And Distortion Ration) and ENOB (Effective Number Of Bits), see Figure 4.
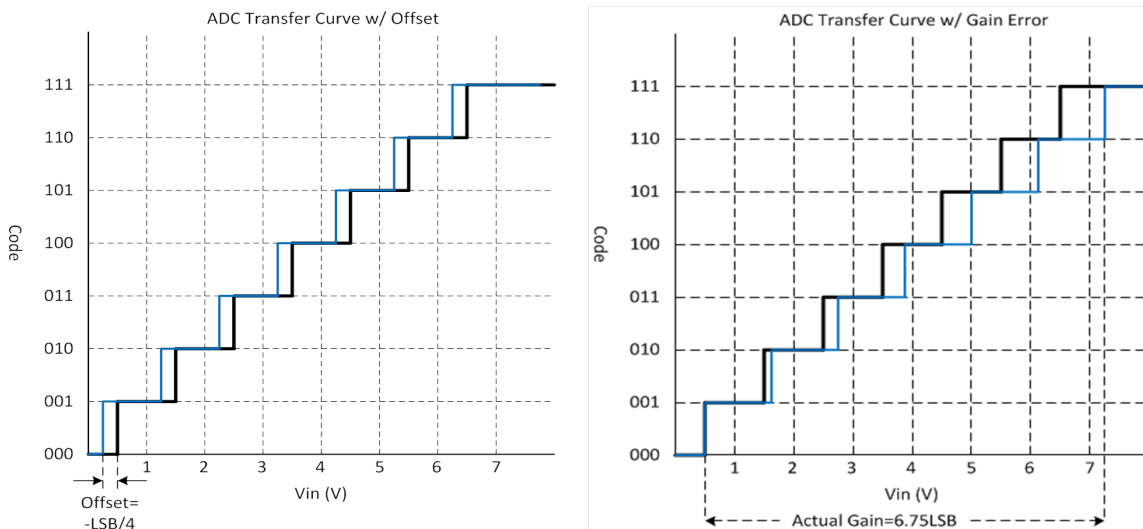
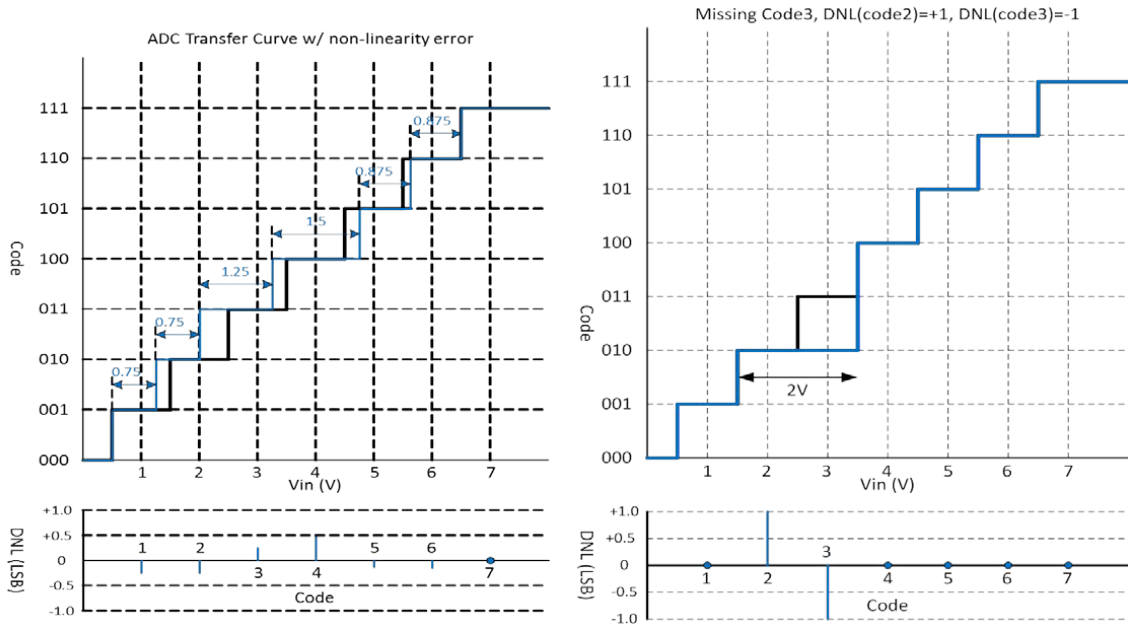

Figure 3: Some Common DC Errors

Figure 4: Some Common AC Errors

1.4   ADC Noise Shaping

Besides oversampling and digital filtering, the Sigma-Delta ADC uses the noise shaping technique to reduce noise. In the frequency domain, It pushes the quantize noise to higher frequencies above the Nyquist (fmax) frequency, see figure 5.
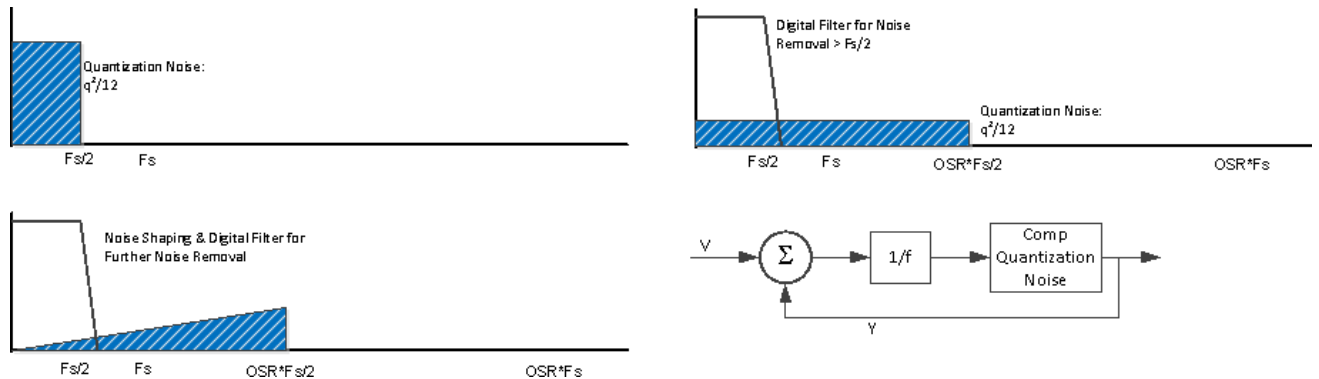

Figure 5: Modulator and Filtering

4350 Centennial Blvd. | Colorado Springs CO 80907 USA | 1-800-645-8862 | cobhamaes.com

UT32M0R500 32-bit Arm™ Cortex® M0+ Microcontroller - ADC Unit

2.0   Application Note Layout

This application note (AN) provides a brief description of the ADC unit's memory map, configuration and programming.

3.0   ADC SPB Configuration Register 0

The Configuration Register 0 (**ADC_SPB_CFG_REG0**) sets the corresponding bit to generate an interrupt, bits [31:27, 24]; the interrupts are cleared by reading either the Status or DataOut registers. Modulator delay to start conversion after reset, bit [31:27]. Sweep mode either single or continuous, bit [13]. It also configures all the power-down modes, bits [10:0].

3.1   ADC SPB Configuration Register 1

The Configuration Register 1 (**CADC_SPB_CFG_REG1**) triggers and monitors the ADC conversion process.

3.2   ADC Timer Control Register

The Timer Control Register 0 (**ADC_TIM_REG**) sets the Oscillator divider (OSCDIV) and the oversample rate (OSR), see **Table 2** and **Table 3** respectively.

Bit [9] and Bit [8] Value	Divider	Modulator Clock Frequency
00	4	12.5 MHz (default)
01	8	6.2 MHz
10	16	3.125 MHz
11	2	25 MHz

Table 2. 50 MHz Oscillator divider

Optimal COI3 OSR	Optimal SINC4 OSR
63	58
80	69
101	82
127	98
160	116
202	138
255	164
	195
	232

Table 3. OSR optimal settings

3.2.1 ADC Sequence Control Register

The Sequence Control Register (**ADC_SEQ_CTRL_REG**) sets the delay between the next channel enable and the start of filter accumulation, with 4 PCLK periods minimum required. The minimum delay is given by the following formula:

$$ADC\_SEQDLY_{min} = (4 \times (ADC\_CK\ freq / SysClk\ freq) - OSR) / 25$$

3.2.2 ADC DSM Digital Stability Control Register

The Digital Stability Control Register (**ADC_STAB_CTRL_REG**) monitors the stability of the modulator and if it becomes unstable, it will assert a modulator reset signal.

ADC Interrupt Status Register

The Interrupt Status Register (**ADC_INT_STATUS**) is for reading the status of the combine ADC_INT output signal; bit values of 1 are for active interrupts. Reading either the ADC_DATAOUT_REG or the ADC_INT_STATUS register of the particular channel clears the interrupt

4.0 ADC Unit Initialization

Code 1 initializes ADC 0 for single-ended input with complete conversion interrupt, gain of one and CIO3 filter, see Code 1. To follow along with the example code and specifics on the API's, download UT32M0R500_API_vx_x_x from: www.cobhamaes.com/hirel

```
// Turn on bit 2 Low Noise Voltage Reference,
// bit 1 Current reference and bit 0 precision current source to ADC/DAC/Comparators
SYSCON->ANALOG_SHUTDOWNS &= ~(SYS_AS_LNREF_SHUTDOWN | SYS_AS_IREF_SHUTDOWN |
SYS_AS_PCS_SHUTDOWN);

//Enable the ADC interrupt in the NVIC interrupt controller.
NVIC_SetPriority (ADC_IRQn, 2);
NVIC_EnableIRQ (ADC_IRQn);

// init the ADC
ADC_StructInit (&ADC_InitStruct);

// Channel 0 input
channelNum = ADC_SE_CHAN_0; dataNum = ADC_SE_DATA_0;

// Oversample rate (OSR), 1-255 possible values
ADC_InitStruct.ModulatorSamples = 232; // SINC4

ADC_InitStruct.SweepType = ADC_SWEEP_SINGLE; // Sweep continuous or single

ADC_InitStruct.OscillatorDivisor = ADC_OSCDIV_BY_4;// 12.5 MHz
ADC_InitStruct.SequenceDelay = 0; // slow down the ADC for multiple channels...

ADC_Init(ADC, &ADC_InitStruct);
```

```
// set up the ADC channel
ADC_ChanCfgStruct.Enable = ENABLE;

ADC_ChanCfgStruct.Gain = ADC_GAIN_1VperV; // Gain amplifier -> 0.5, 1, 2, 4, 8, 16 V/V

// Filter COI3 or SINC4
ADC_ChanCfgStruct.UseDDF2 = TRUE; // use SINC4

ADC_SetChannelConfig(ADC, channelNum, &ADC_ChanCfgStruct);

// Enable ADC conversion complete interrupt
//ADC->SPB_CFG_0 |= ADC_CFG0_CONV_COMPL_IEN;
ADC_IntConfig (ADC, ADC_CONV_COMPL_INT, ENABLE);

//ADC->SPB_CFG_0 |= ADC_CFG0_ADC_INTR_EN;
ADC_IntConfig (ADC, ADC_ADC_GLOBAL_INT, ENABLE);
```

Code 1: ADC 0 Initialization

5.0 ADC Unit Programming

Section 3.0 presented some of the basic configurations for the ADC core.   The following sections show programming examples by making use of CAES API's for the UT32RM0R500.

Service Interrupt Service Routine (ISR)

Interrupt 21 in the Vector Table maps to the ADC ISR; the software checks which interrupt happened using linked list, see Code 2.

```
void ADC_IRQHandler(void)
{
    NodePtr Ptr;
    GPIO_WriteOutputDataBit (GPIO2, PIN_15, SET);
    Ptr = &ADCTasks[0];// Points to first task in linked list
    while (Ptr){// Handles all ADC interrupt requests
    if(ADC->INT_STATUS & (Ptr->Mask)){
        (*Ptr->ADCHandler)();// Execute ADC handler
        }
    Ptr = Ptr->Next;// Poll next device
    }
}
```

Code 2: ADC Interrupt Handler

5.2 Conversion Complete Handler

The handler clear the interrupt flag and sets the sets the service flag to read the ADC value in the main loop, see Code 3.

```
void CONVHandler(void){
    // Clear the flag by reading either ADC_DATAOUT_REG or ADC_INT_STATUS registers.
    ADC->INT_STATUS;

    //ADC->SPB_CFG_1 = ADC_CFG1_CONV_COMPLETE; // write a '1' to the bit to clear it
    ADC_ConvComplFlag=1;
}
```

Code 3: ADC Conversion Complete Handler

5.3 ADC Read

In the main loop, the software check to see if the ADC is ready to read a new value from the specified ADC single-ended channel, see Code 4.

```
if((ADC_ConvComplFlag)&&(!TimerTickExpired)){
    ADC_ConvComplFlag=0;

    Error=ADC_ReadChannel (ADC, dataNum, &ADC_Data, &ChannelData);
    ADC_Data = (ADC_Data & ADC_DATA_OUT_MASK);
    if(ADC_Data & 0x800){
        ADC_Data &= ~(0x800);
    }
    else{
        ADC_Data |= 0x800;
    }

twosComplement = ~ADC_Data;
twosComplement += 1;
GPIO_WriteOutputDataBit (GPIO2, PIN_15, RESET);
DAC_WriteChannel (DAC, DAC_CHAN_0, twosComplement);
GPIO_WriteOutputDataBit (GPIO2, PIN_15, SET);

if(Error){
    // Print the error
    // set GPIO 47, turn on LED
    GPIO_WriteOutputDataBit (GPIO2, PIN_13, SET);
    __ASM volatile ("nop");
    // clear GPIO 47, turn off LED
    GPIO_WriteOutputDataBit (GPIO2, PIN_13, RESET);
    }
}
```

Code 4: ADC Read

Putting it all together, Code 5 shows the main subroutine with a placeholder for initialization from the previous sections and an endless loop for reading an ADC value. The ADC_IRQHandler is the interrupt service routine for handling the particular ADC channel input. Figure 6 shows the ADC channel 0 output for the given example.

```c
int main (void){
    // Initialization and settings go here from previous sections.
    for(;;){
        if(TimerTickExpired){
            TimerTickExpired=0;
            ADC->SPB_CFG_1 |= ADC_CFG1_TRIGGER;
            // clear GPIO 47, turn off LED
            GPIO_WriteOutputDataBit (GPIO2, PIN_15, RESET);
        }
        if((ADC_ConvComplFlag)&&(!TimerTickExpired)){
            ADC_ConvComplFlag=0;
            Error=ADC_ReadChannel (ADC, dataNum, &ADC_Data, &ChannelData);
            ADC_Data = (ADC_Data & ADC_DATA_OUT_MASK);
            if(ADC_Data & 0x800){
                ADC_Data &= ~(0x800);
            }
            else{
                ADC_Data |= 0x800;
            }

            twosComplement = ~ADC_Data;
            twosComplement += 1;
            GPIO_WriteOutputDataBit (GPIO2, PIN_15, RESET);
            DAC_WriteChannel (DAC, DAC_CHAN_0, twosComplement);
            GPIO_WriteOutputDataBit (GPIO2, PIN_15, SET);

            if(Error){
                // Print the error
                // set GPIO 47, turn on LED
                GPIO_WriteOutputDataBit (GPIO2, PIN_13, SET);
                __ASM volatile ("nop");
                // clear GPIO 47, turn off LED
                GPIO_WriteOutputDataBit (GPIO2, PIN_13, RESET);
            }
        }
    }
}
```

Code 5: Sample program for the ADC
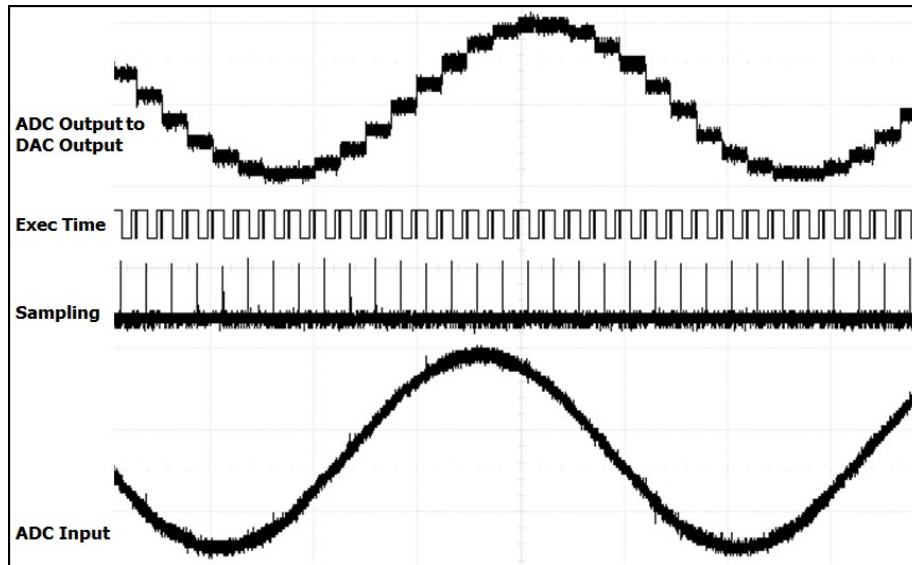
CAES PIONEERING ADVANCED ELECTRONICS

Figure 6: ADC Input and Output to DAC

6.0   Summary and Conclusion

At low frequencies, Delta-sigma ADC performs well in high resolution and noise shaping.

For more information about our UT32M0R500 microcontroller and other products, please visit our website: www.cobhamaes.com/HiRel

APPLICATION NOTE

7.0 REVISION HISTORY

Date	Rev. #	Author	Change Description
12/06/18	1.0.0	JA	Initial Release

The following United States (U.S.) Department of Commerce statement shall be applicable if these commodities, technology, or software are exported from the U.S.: These commodities, technology, or software were exported from the United States in accordance with the Export Administration Regulations. Diversion contrary to U.S. law is prohibited.

12/6/2018

CAES PIONEERING ADVANCED ELECTRONICS

4350 Centennial Blvd. | Colorado Springs CO 80907 USA | 1-800-645-8862 | cobhamaes.com