Abstract

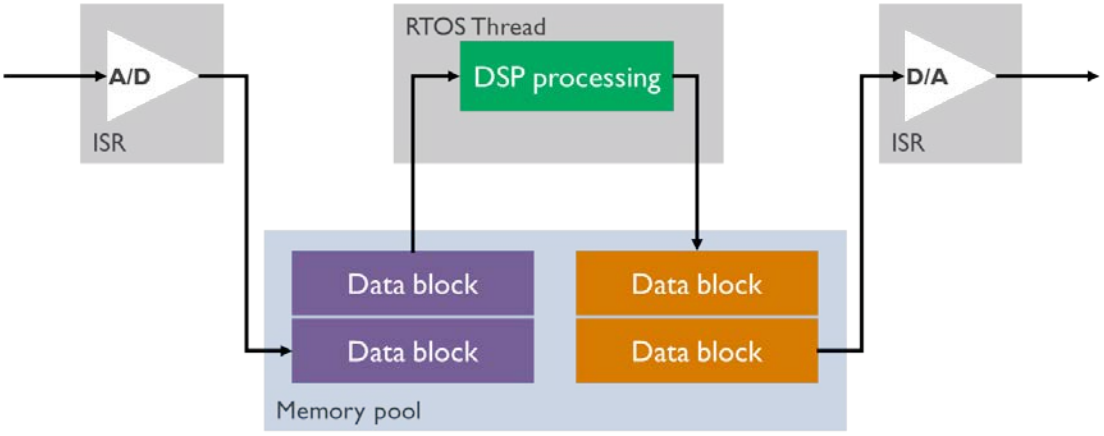
This application note describes the development of a digital filter for an analog input signal using the CMSIS-DSP library and Keil RTX5. The application is designed for an NXP LPC1768 device and can be tested using µVision simulation capabilities. The Event Recorder is used to verify the program flow.

Using CMSIS, the application can be easily ported to any other ARM® Cortex®-M processor-based device.

Contents

Introduction

This example project implements a low-pass filter using the DSP block processing mode. Incoming analog signals are converted by the A/D peripheral and are collected into data blocks by an interrupt service routine (ISR). These data blocks are saved in a memory pool and upon completion event flags are sent to an RTOS thread. The thread executes a filter algorithm and saves the processed data in a memory pool. The availability of a data block is then signaled to the ISR using another event flag. The data is then converted back to an analog signal by the D/A peripheral. Keil RTX5 coordinates the flow of the data blocks through an interrupt service routine (ISR). The CMSIS-DSP library provides the filter algorithm used in the worker thread. Since this example application uses the DSP block processing mode, the output signal gets delayed by a few milliseconds.



Prerequisites

To run through the application note's example project, you need to install the following software:

- MDK version 5.22 or higher with CMSIS 5.0.0 pack
- Keil.LPC1700_DFP.2.2.0.pack or higher

The example project runs fully in simulation without the need for any target hardware.

Project overview

The µVision example project DSP_App.uvprojx implements a digital low-pass filter. The filter is defined with the following characteristics:

- Filter Sampling Frequency [Hz]: 32000
- Passband Edge Frequency [Hz]: 3200
- Stopband Edge Frequency [Hz]: 9600
- Passband Ripple [dB]: 0.1
- Stopband Ripple [dB]: 60

Characteristics for the different filter types:

- FIR Filter Length: 15
- IIR Order Estimation: 4

Several variants for Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filters are included in the project. In µVision, the filter variants can be selected through the project targets described below:

Target	Configuration
SIM FIR FLOAT32	FIR filter using the floating-point number format
SIM FIR Q31	FIR filter using the fixed-point number format Q31
SIM FIR Q15	FIR filter using the fixed-point number format Q15
SIM IIR FLOAT32	IIR filter using the floating-point number format
SIM IIR Q31	IIR filter using the fixed-point number format Q31
SIM IIR Q15	IIR filter using the fixed-point number format Q15

The example code that is important for processing the signals is split into different modules:

Module	Description
ADC.c	Hardware abstraction layer for the A/D converter.
DAC.c	Hardware abstraction layer for the D/A converter.
DSP_App.c	Main application module, which initializes the timer, A/D and D/A converter, and starts the RTX kernel. The file contains the thread to process the data and the ISR to handle the sampling of the input and output data.
DSP_FIR.c	Definition file for the FIR filter structure and state buffers. Contains the filter coefficients. Functions to initialize the FIR data structure and functions to process the data are declared in this file.
DSP_IIR.c	Definition file for the IIR filter structure and state buffers. Contains the filter coefficients. Functions to initialize the IIR data structure and functions to process the data are declared in this file.
Timer.c	Hardware abstraction layer for the on-chip timers.

The most important functions of the main module DSP_App.c are described briefly below:

- **main()**: initializes the A/D and D/A converters, and starts the RTX kernel. It also initializes the Timer2 to trigger at 32 kHz. The timer clock rate, `#define TimerFreq`, has to be set equal to the filter sampling frequency.
- **RTX_Features_Init**: initializes the memory pool for messages `DSP_MsgPool` and the event flags `DSP_Event`. It creates the **Clock** and **SigMod** threads.
- **TIMER2_IRQHandler**: is the ISR triggered by the Timer2 interrupt. The ISR collects the values from the A/D converter and stores the data in the memory pool. The ISR also sends the output data, which have been computed by the task **SigMod**, to the D/A converter.
- **SigMod**: is the RTX5 thread that computes the data by applying the filter function of the CMSIS-DSP library and stores the computed values in a memory pool for further processing. All data buffers in the program contain 256 samples. The buffers size can be adjusted through the `#define DSP_BLOCKSIZE`.

Additional modules and functions, which are not relevant to understand the DSP implementation, have not been explained in this document. They are related to device configuration settings or peripherals.

Port the filter coefficients to the application

The project includes two files, **FIR_32K_filterCoeff_QED.flt** and **IIR_32K_filterCoeff_QED.flt**, containing the filter coefficients created using QEDesign. These filter coefficients have been copied to the example application.

Using your own FIR/IIR filter coefficients, open the file DSP_FIR.c/DSP_IIR.c in µVision and copy the coefficients to the data structure.

DSP processing

The code snippets used in this document have been simplified for demonstration purposes. Error handling and overflow checks have been omitted. The IIR Q31 filter variant is used to explain the code.

The application can be split logically into three parts:

1. Sampling input data – processed by the ISR **TIMER2_IRQHandler**.
2. Filtering the sampled data – processed by the thread **SigMod**.
3. Sampling output data – processed by the ISR **TIMER2_IRQHandler**.

The buffers for the sampled and processed data are arrays with the size `DSP_BLOCKSIZE` (set either in DSP_FIR.c or DSP_IIR.c), in this example set to 256. The memory pool contains four blocks of data.

```
/*----------------------------------------------------------------------------
   defines & typedefs for messages
 *----------------------------------------------------------------------------*/
typedef struct _DSP_DataType {
  q31_t     Sample[DSP_BLOCKSIZE];
} DSP_DataType;
```

The ISR **TIMER2_IRQHandler** samples the incoming A/D data and converts them to the selected number format, Q31 for example. The samples are stored in the data block `pDataTimIrqOut`. After storing `DSP_BLOCKKSIZE` samples, the event flag `EVENT_DATA_TIM_OUT_SIG_IN` is set.

The code below is part of the ISR **TIMER2_IRQHandler**:

```
    /* -- signal Input Section ------------------------------------------------ */
    adGdr = LPC_ADC->ADGDR;
    if (adGdr & (1UL << 31)) {          /* Data available ?                        */
      /* scale value and move it in positive/negative range. (12bit Ad = 0xFFF)
         filter in range is -1.0 < value < 1.0                                    */
      tmpFilterIn =  ((float32_t)((adGdr >> 4) & 0xFFF) / (0xFFF / 2)) - 1;
      arm_float_to_q31(&tmpFilterIn, &tmp, 1);
      pDataTimIrqOut->Sample[dataTimIrqOutIdx++] = tmp;
      if (dataTimIrqOutIdx >= DSP_BLOCKSIZE) {
```

```
        // set buffer and event
        pDataSigModIn = pDataTimIrqOut;
        flags = osEventFlagsSet(DSP_Event, EVENT_DATA_TIM_OUT_SIG_IN);
        if (flags < 0) {
          errHandler(__LINE__);
        }

        // allocate next output buffer
        pDataTimIrqOut = osMemoryPoolAlloc(DSP_MemPool, 0);
        if (pDataTimIrqOut == NULL) {
          errHandler(__LINE__);
        }

        dataTimIrqOutIdx = 0;
      }
    }
    else {
      errHandler(__LINE__);
    }
```

The **SigMod** thread waits for the event flag `EVENT_DATA_TIM_OUT_SIG_IN`. After the event flag is set, the CMSIS-DSP library filter function is executed. The received and the newly allocated data from the memory pool is directly used as parameters in the filter function. On completion of the filter function, the filtered data is saved in the memory pool and the event flag `EVENT_DATA_TIM_IN_SIG_OUT` is set.

The code below is part of the **SigMod** thread:

```
void SigMod (void __attribute__((unused)) *arg) {
  int32_t flags;
  osStatus_t status;

  for (;;) {

    // wait for data
    flags = osEventFlagsWait(DSP_Event, EVENT_DATA_TIM_OUT_SIG_IN, 0, osWaitForever);
    if (flags < 0) {
      errHandler(__LINE__);
    }

    iirExec_q31 (pDataSigModIn->Sample, pDataSigModOut->Sample);

    // free input buffer
    status = osMemoryPoolFree(DSP_MemPool, pDataSigModIn);
    if (status != osOK) {
      errHandler(__LINE__);
    }

    // data is ready
    pDataTimIrqIn = pDataSigModOut;
    flags = osEventFlagsSet(DSP_Event, EVENT_DATA_TIM_IN_SIG_OUT);
    if (flags < 0) {
      errHandler(__LINE__);
    }

    // allocate next output buffer
    pDataSigModOut = osMemoryPoolAlloc(DSP_MemPool, 0);
    if (pDataSigModOut == NULL) {
      errHandler(__LINE__);
    }
  }
}
```

The data blocks sent from the **SigMod** thread are stored in the memory pool `DSP_MemPool`. The ISR **TIMER2_IRQHandler** reads the data blocks and converts them to a format that fits the D/A converter. Then, the data is sent to the D/A converter. After processing `DSP_BLOCKKSIZE` samples, the data block memory is freed.

The code below is part of the ISR **TIMER2_IRQHandler**:

```
    /* -- signal Output Section ------------------------------------------------ */
    if (dataTimIrqInIdx == 0) {
      // check if data is available
      flags = osEventFlagsWait(DSP_Event, EVENT_DATA_TIM_IN_SIG_OUT, 0, 0);
    }
    if ((dataTimIrqInIdx > 0) || (flags==EVENT_DATA_TIM_IN_SIG_OUT)) {

      tmp = pDataTimIrqIn->Sample[dataTimIrqInIdx++];
      arm_q31_to_float(&tmp, &tmpFilterOut, 1);
      /* move value in positive range and scale it.   (10bit DA = 0x3FF)
         filter OUT range is -1.0 < value < 1.0                 */
      LPC_DAC->DACR = (((uint32_t)((tmpFilterOut + 1) * (0x03FF / 2))) & 0x03FF) <<  6;

      if (dataTimIrqInIdx >= DSP_BLOCKSIZE) {
        // free input buffer
        status = osMemoryPoolFree(DSP_MemPool, pDataTimIrqIn);
        if (status != osOK) {
          errHandler(__LINE__);
        }
        dataTimIrqInIdx = 0;
      }
    }

    LPC_TIM2->IR |= (1UL <<  0);              /* clear MR0 Interrupt  flag   */
  }
}
```

RTX5 configuration settings in RTX_Config.h

As the memory pool is made up of four objects with a size of 256 x 4 bytes, the overall **Global Dynamic Memory size [bytes]** is set to *8192*. Apart from that, only default values are used for RTX5.

Debug the application

The application can be tested with the µVision simulator and the Logic Analyzer. µVision incorporates a C-type scripting language to simulate analog input signals for the A/D converter. The VTREG AIN2 is linked to the input pin of the A/D converter. The code to generate input signals can be found in the debug command file Dbg_Sim.INI.

```
/*-----------------------------------------------------------------------------
  Generate mixed sine wave signal
    a   = amplitude of mixed sine wave
    f1  = frequency of sine wave 1
    f2  = frequency of sine wave 2
 *-----------------------------------------------------------------------------*/
signal void SineMix (float a, float f1, float f2)  {
  float   sin1;
  float   sin2;
  float   w;

  for (;;) {                                // do forever
    w    = 2 * 3.1415926 * f1;
    sin1 = __sin ( ((double)STATES / CCLK) * w);
    w    = 2 * 3.1415926 * f2;
    sin2 = __sin ( ((double)STATES / CCLK) * w);
    AIN2 =  (((sin1 + sin2) * a)) + 1.6;  // set analog value
    swatch (0.00001);                       // in 10 uSec resolution
  }                                         // end do forever
}
```

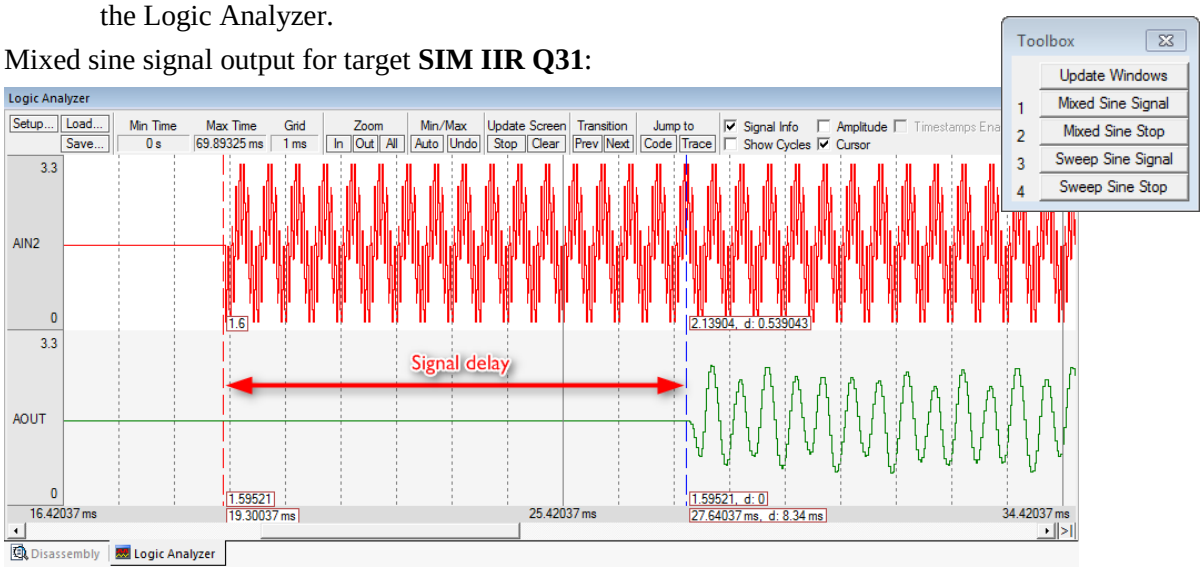
Toolbox buttons that start and stop generating sine waves are defined in the file Dbg_Sim.INI:

```
DEFINE BUTTON "Mixed Sine Signal", "SineMix   (1.0, 2000.0, 14000.0)"
DEFINE BUTTON "Mixed Sine Stop",   "SIGNAL KILL SineMix"
DEFINE BUTTON "Sweep Sine Signal", "SineSweep (1.0, 2500.0, 9700.0)"
DEFINE BUTTON "Sweep Sine Stop",   "SIGNAL KILL SineSweep"
```

To debug the program:

- Start the µVision debugger with **Debug – Start/Stop Debug Session (Ctrl+F5)**.
- Click **Debug – Run (F5)** to continue debugging the program.
- Use the Toolbox window buttons to start generating the sine wave signals. The output can be viewed in the Logic Analyzer.

Mixed sine signal output for target **SIM IIR Q31**:

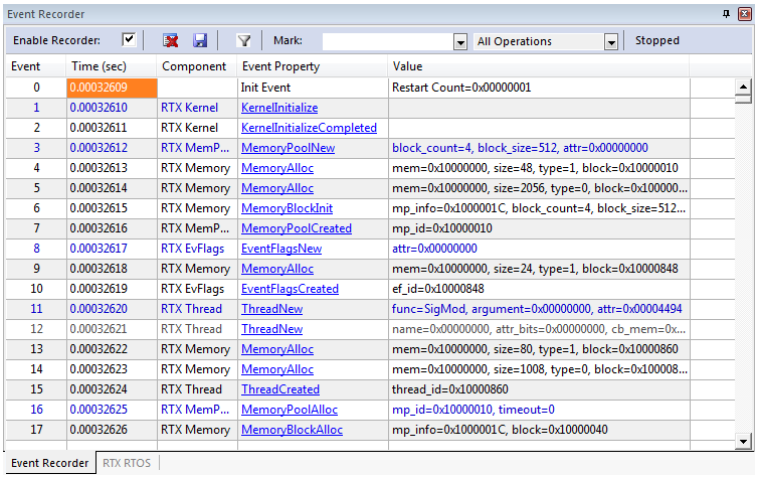


Debugging RTX5 using Event Recorder

The example project is using Event Recorder annotations. The Event Recorder shows execution status and event information, and helps to analyze the operation of RTX5.

In µVision debugger, the Event Recorder window will show all events from RTX5.

Use the filter to select only a subset of events:



Event	Time (sec)	Component	Event Property	Value
0	0.00032609		Init Event	Restart Count=0x00000001
1	0.00032610	RTX Kernel	KernelInitialize	
2	0.00032611	RTX Kernel	KernelInitializeCompleted	
3	0.00032612	RTX MemP...	MemoryPoolNew	block_count=4, block_size=512, attr=0x00000000
4	0.00032613	RTX Memory	MemoryAlloc	mem=0x10000000, size=48, type=1, block=0x10000010
5	0.00032614	RTX Memory	MemoryAlloc	mem=0x10000000, size=2056, type=0, block=0x100000...
6	0.00032615	RTX Memory	MemoryBlockInit	mp_info=0x1000001C, block_count=4, block_size=512...
7	0.00032616	RTX MemP...	MemoryPoolCreated	mp_id=0x10000010
8	0.00032617	RTX EvFlags	EventFlagsNew	attr=0x00000000
9	0.00032618	RTX Memory	MemoryAlloc	mem=0x10000000, size=24, type=1, block=0x10000848
10	0.00032619	RTX EvFlags	EventFlagsCreated	ef_id=0x10000848
11	0.00032620	RTX Thread	ThreadNew	func=SigMod, argument=0x00000000, attr=0x00004494
12	0.00032621	RTX Thread	ThreadNew	name=0x00000000, attr_bits=0x00000000, cb_mem=0x...
13	0.00032622	RTX Memory	MemoryAlloc	mem=0x10000000, size=80, type=1, block=0x10000860
14	0.00032623	RTX Memory	MemoryAlloc	mem=0x10000000, size=1008, type=0, block=0x100008...
15	0.00032624	RTX Thread	ThreadCreated	thread_id=0x10000860
16	0.00032625	RTX MemP...	MemoryPoolAlloc	mp_id=0x10000010, timeout=0
17	0.00032626	RTX Memory	MemoryBlockAlloc	mp_info=0x1000001C, block=0x10000040

Conclusion

The CMSIS-DSP library and RTX5 make the implementation of DSP algorithms straightforward. RTX5 memory pools and event flags support the DSP block processing offered by the CMSIS-DSP library.

The CMSIS-DSP library offers a common set of DSP algorithms. Only minor code changes are required for processing data with different filter types. As a consequence, development cycles are shortened and developers can focus on the application requirements and design, without having to develop and implement their own DSP algorithms.