Subsystem Design

Digital FIR Filter

1 Description

This subsystem demonstrates how the internal ADC, and math accelerator (MATHACL) modules within the MSPM0G family of devices can be used to implement a simple, streaming FIR filter of an analog signal. In this configuration, noise on an analog signal can be filtered based on the desired filter order and coefficients without waiting for software floating point calculations.

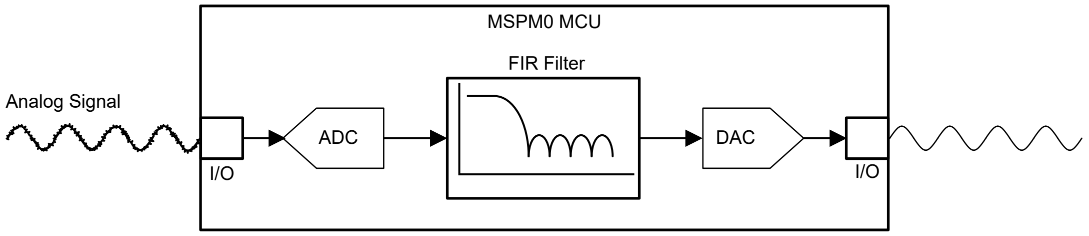

Figure 1-1. FIR Filter Functional Block Diagram

2 Required Peripherals

Required Peripherals

This application requires an integrated ADC, MathACL, and DAC12 modules.

Table 2-1. Required Peripherals

Sub-block Functionality	Peripheral Use	Notes
Analog Signal Capture	(1×) ADC	Shown as *ADC12_0_INST* in code
FIR Filter	(1×) MathACL	Shown as *MATHACL* in code
Analog Signal Output (Optional)	(1×) DAC12	Shown as *DAC12_0_INST* in code

3 Compatible Devices

Based on the requirements listed in Table 2-1, this example is compatible with the devices listed in Table 3-1. The corresponding EVM can be used for prototyping.

Table 3-1. Compatible Devices

Compatible Devices	EVM
MSPM0G35xx, MSPM0G15xx	LP-MSPM0G3507

4 Design Steps

1. Determine the desired corner frequency and filter response.
2. Set the ADC sampling frequency. This must be at least twice the expected bandwidth of the signal.
3. Calculate the desired coefficients and filter order. The filter coefficients are rational numbers which, combined with the sampling frequency, determine the pass and rejection bands of the filter.
   a. There are different methods and tools for FIR filter coefficient calculation, which is not discussed in this document.
4. Convert the filter coefficients to fixed point values.
   a. In the example code, a Q16 (16 fractional bits) representation is used. Perform this conversion using the IQMath library or by multiplying the coefficients by 2^n where *n* is the desired number of fractional bits. Verify that the selected data type can hold these values without overflowing.
   b. The filter coefficients are constant values, and as a result, can be contained in flash to save room in SRAM if desired.

5 Design Considerations

1. **Input signal bandwidth**: The bandwidth of the signal that must be resolved determines the ADC sampling frequency and the amount of data the code must process.
2. **ADC reference voltage:** The ADC reference voltage must be selected so the signal amplitude can be fully captured with good resolution.
3. **Filter order**: Every increase in filter order is another set of operations the user must perform per each sample. This increases the overall processing time between samples and limits the amount of other processes the user can perform. The result is an increase in filter rejection and increased resolution of the desired signal.

6 Software Flow Chart

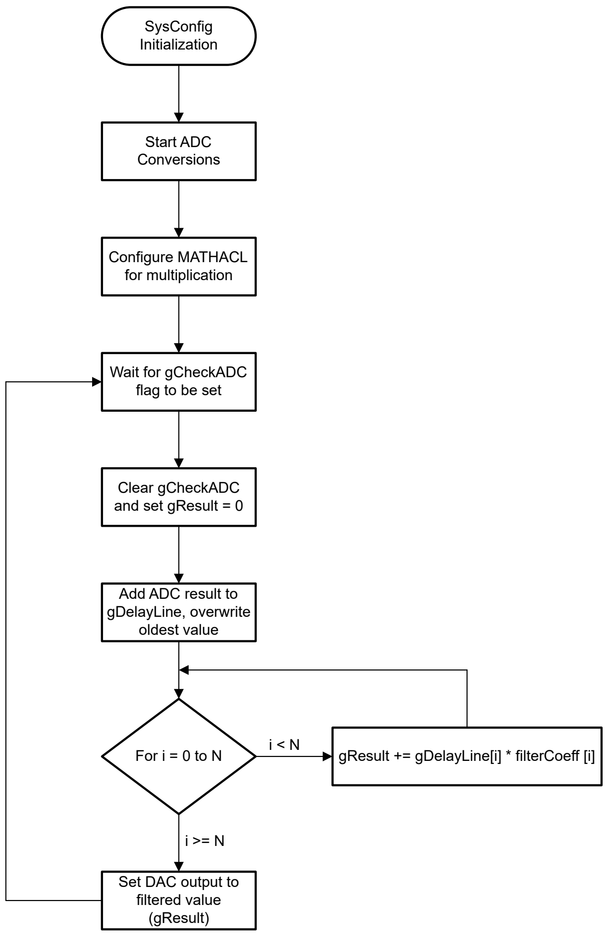
SysConfig Initialization
Start ADC Conversions
Configure MATHACL for multiplication
Wait for gCheckADC flag to be set
Clear gCheckADC and set gResult = 0
Add ADC result to gDelayLine, overwrite oldest value
For i = 0 to N
i < N
gResult += gDelayLine[i] * filterCoeff [i]
i >= N
Set DAC output to filtered value (gResult)

Figure 6-1. Example Software Sequence

7 Application Code

```
#define FILTER_ORDER 24
#define FIXED_POINT_PRECISION 16
volatile bool gCheckADC;
uint32_t gDelayLine[FILTER_ORDER];
uint32_t gResult = 0;
/* Filter coefficients are input as 16-bit Precision fixed point values  */

static int32_t filterCoeff[FILTER_ORDER] = {
  -62, -153, -56, 434, 969, 571,
  -1291, -3237, -2173, 3989, 13381, 20518,
  20518, 13381, 3989, -2173, -3237, -1291,
  571, 969, 434, -56, -153, -62
};

const DL_MathACL_operationConfig gMpyConfig = {
    .opType      = DL_MATHACL_OP_TYPE_MAC,
    .opSign      = DL_MATHACL_OPSIGN_SIGNED,
    .iterations  = 0,
    .scaleFactor = 0,
    .qType       = DL_MATHACL_Q_TYPE_Q16};

int main(void)
 {
    SYSCFG_DL_init();
    NVIC_EnableIRQ(ADC12_0_INST_INT_IRQN);
    gCheckADC = false;
    DL_ADC12_startConversion(ADC12_0_INST);

    /* Configure MathACL for Multiply */
    DL_MathACL_configOperation(MATHACL, &gMpyConfig, 0, 0 );

    while (1) {
        while (false == gCheckADC) {
            __WFE();
        }

        gCheckADC = false;
        gResult = 0;
        /* Append the most recent ADC result to the delay line */
        memmove(&gDelayLine[1], gDelayLine, sizeof(gDelayLine) - sizeof(gDelayLine[0]));
        gDelayLine[0] = DL_ADC12_getMemResult(ADC12_0_INST, DL_ADC12_MEM_IDX_0);

        /* Calculate FIR Filter Output */
        for (int i = 0; i < FILTER_ORDER; i++){
            /* Set Operand One last */
            DL_MathACL_setOperandTwo(MATHACL, filterCoeff[i]);
            DL_MathACL_setOperandOne(MATHACL, gDelayLine[i]);
            DL_MathACL_waitForOperation(MATHACL);
        }
        /* Our result should not exceed the bounds of RES1 register, in other applications you may
use both RES1 and RES2 registers */
        gResult = DL_MathACL_getResultOne(MATHACL);
        DL_DAC12_output12(DAC0, (uint32_t)(gResult));

        /* Clear Results Registers */
        DL_MathACL_clearResults(MATHACL);
    }
}

/* Set the ADC Result flag to trigger our main loop to process the new data */
void ADC12_0_INST_IRQHandler(void)
{
    switch (DL_ADC12_getPendingInterrupt(ADC12_0_INST)) {
        case DL_ADC12_IIDX_MEM0_RESULT_LOADED:
            gCheckADC = true;
            break;
        default:
            break;
    }
}
```

8 Additional Resources

- Texas Instruments, *MSPM0 G-Series 80-MHz Microcontrollers Technical Reference Manual*, technical reference manual.
- Texas Instruments, *MSPM0G350x Mixed-Signal Microcontrollers With CAN-FD Interface*, data sheet.
- Texas Instruments, *MSPM0G150x Mixed-Signal Microcontrollers*, data sheet.

9 E2E

See TI's E2E support forums to view discussions and post new threads to get technical support for using MSPM0 devices in designs.