An example project for FIR function implement on KV31 with SDK and CMSIS

1. Introduction

This example project shows how to use CMSIS DSP functions to implement a digital filter. The project is based on Kinetis KV31F512.

It first generates a mixed signal which is composed by two sine waves. Then it uses PDB to trigger DAC every 200 microsecond and output the signal through DAC, so that the mixed signal can be displaying on oscilloscope.

With QEDesign tool, a lowpass filter is designed, the filter coefficients are used directly to initialize the FIR function supplied by CMSIS DSP library.

This example calls the FIR functions in CMSIS DSP lib, and the filtered signal is output though DAC module.

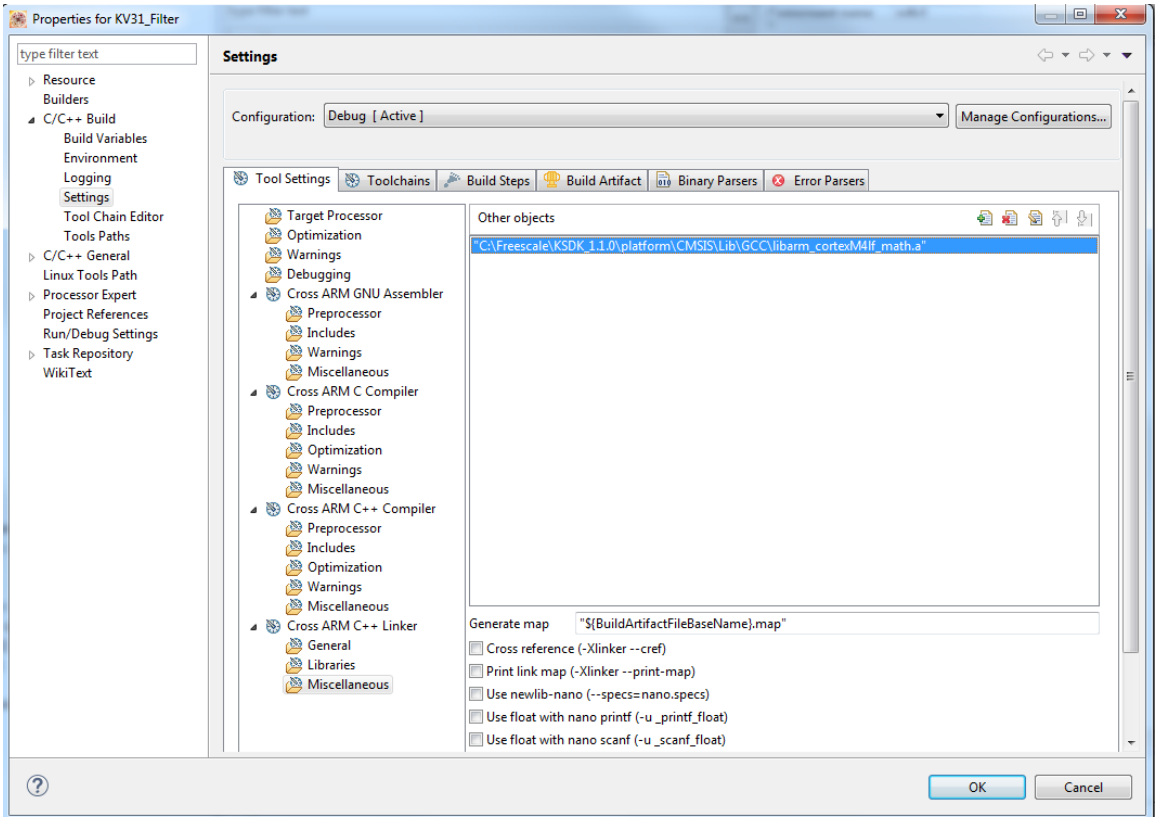
2. KDS+SDK development environment

This example project is developed on KDS2.0.0 and SDK1.1.0. The method is same for the latest KDS and SDK versions.

First download KDS from FSL website, and install. Then download SDK and install. In KDS, click on <Help> menu->Check for updates to install all updates. In order to get the SDK and PE stationery support in KDS for more targets, please install KSDK 1.1.0 Eclipse Update in KDS. For details, please refer to section < 5.2 Install Eclipse update> in:

C:\Freescale\KSDK_1.x.0\doc\Getting Started with Kinetis SDK (KSDK) v.1.x.pdf

This example project also include the CMSIS(Cortex Microcontroller Software Interface Standard) library. Please add the math lib into the project in C/C++ Build Settings:



3. Software Design

3.1 mixed signal generation

The project first generates a mixed signal which is composed by two sine waves, the frequency of one sine wave is 50Hz, the other is 500Hz. These two signals are mixed in function Init_SinTable(), and are sampled periodically. Please see the source file main.c.

The project uses PDB to trigger DAC0 every 200 microsecond and output the signal through DAC0. This is implemented in PDB ISR PDB0_IRQHandler() as below:

```
DAC_DRV_Output(FSL_DACONV0,DAC0_OuputData[MyPdbIntCounter]);
```

3.2 Filter Implement

The filter is initialized with the coefficients got by QEDesign (for details, please see the section 4 in this doc) in function Init_Filter(), which is called at the beginning of main().

The filter is implemented in PDB ISR named PDB0_IRQHandler(). In this example, it calls the fixed-point function with Q15 fractional format. And then add the offset and scales the result to output it though DAC. The source code is as below:

```
  arm_fir_q15(&ArmFirInstance, &SignalOnePeriod[MyPdbIntCounter],
&FilterOutData, 1);
    FilterOutData =  (FilterOutData+0x7FFF+0x1);
    FilterOutData = FilterOutData >> 4;

    DAC_DRV_Output(FSL_DACONV1,FilterOutData);
```

3.3 DAC and PDB initialization

The mixed signal is output though DAC0 channel, the signal after lowpass filtering is output though DAC1 channel. It uses PDB0 to trigger both DAC0 and DAC1 every 200ms. This example uses SDK driver to initialize PDB0, DAC0 and DAC1 as below:

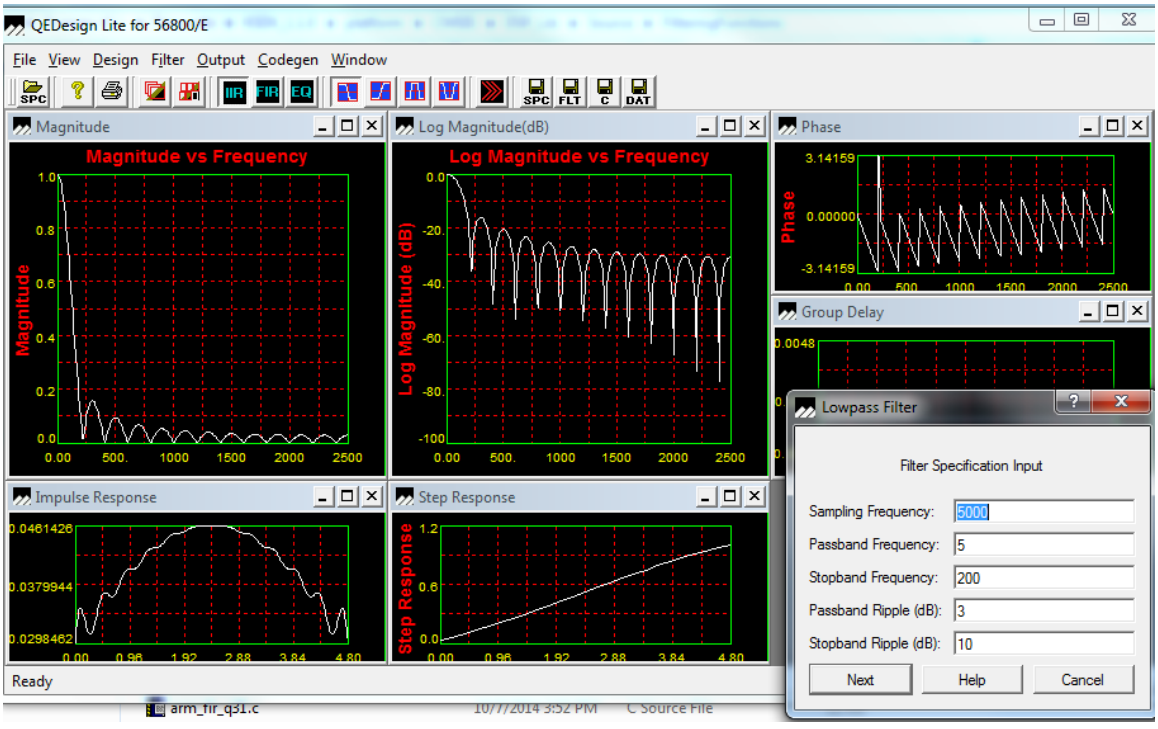
```
/* configure DAC0, DAC1 */
  /* DAC0 */
  daConv0_InitConfig0.triggerMode = 0; /* Select DACREF_1 as the reference
voltage. */
  daConv0_InitConfig0.lowPowerEnable = false; /* Select hardware trigger. */
  daConv0_InitConfig0.refVoltSrcMode = 1; /* Select DACREF_2 as the reference
voltage. */
  DAC_DRV_Init(0, &daConv0_InitConfig0);
  DAC_DRV_DisableBuff(FSL_DACONV0);

  /* DAC1 */
  DAC_DRV_StructInitUserConfigNormal(&daConv0_InitConfig0);

  daConv1_InitConfig0.triggerMode = 0; /* Select DACREF_1 as the reference
voltage. */
  daConv1_InitConfig0.lowPowerEnable = false; /* Select hardware trigger. */
  daConv1_InitConfig0.refVoltSrcMode = 1; /* Select DACREF_2 as the reference
voltage. */
  DAC_DRV_Init(1, &daConv1_InitConfig0);
  DAC_DRV_DisableBuff(FSL_DACONV1);


  /* Initialize PDB */
  PDB_DRV_StructInitUserConfigForSoftTrigger(&pdly1_InitConfig0);

  pdly1_InitConfig0.multFactorMode = kPdbMultFactorAs1; /*Multiplication
factor is 1*/
  pdly1_InitConfig0.clkPrescalerDivMode = kPdbClkPreDivBy1;
  pdly1_InitConfig0.continuousModeEnable = true;
  pdly1_InitConfig0.delayValue = 0x0040U;       // insignificance
 // pdly1_InitConfig0.pdbModulusValue = 0x1F40U; // PDB clock 40M, sampling 5K
  pdly1_InitConfig0.pdbModulusValue =  4000 ;//0x100U; // PDB clock 40M or
20M ??, 50Hz & 500Hz
```

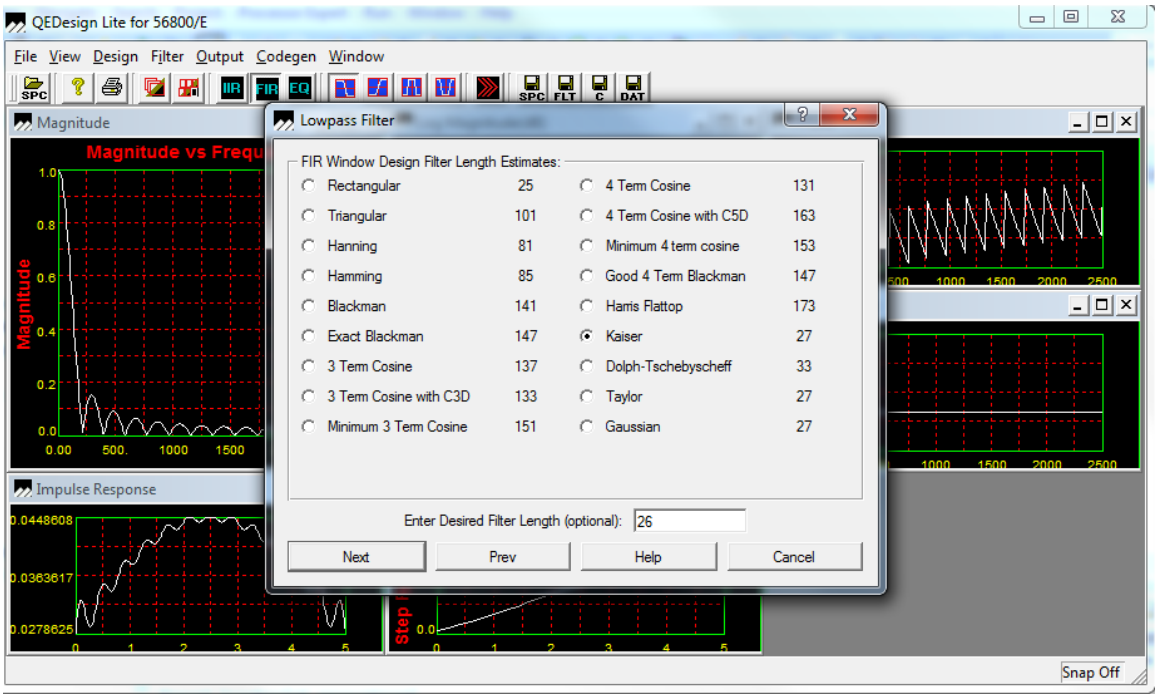
```
 // pdly1_InitConfig0.triggerSrcMode = kPdbSoftTrigger; /* Select software
trigger*/
  PDB_DRV_Init(FSL_PDLY1, &pdly1_InitConfig0);


  // Initialize the DAC trigger. //
  pdly1_DacTrigInitConfig0.extTriggerInputEnable = false;
 // pdly1_DacTrigInitConfig0.intervalValue = 0x300U;
  pdly1_DacTrigInitConfig0.intervalValue = 0x50U;

  PDB_DRV_EnableDacTrigger(FSL_PDLY1, 0, &pdly1_DacTrigInitConfig0);
  PDB_DRV_EnableDacTrigger(FSL_PDLY1, 1, &pdly1_DacTrigInitConfig0);

  /* Start PDB Counter*/
  PDB_DRV_SoftTriggerCmd(FSL_PDLY1);
```

4. Filter Design

The lowpass filter is designed by QEDesign tool, which is a tool for digital filter design included in Classic CodeWarrior for 56800/E. The filter's stopband frequency is configured as 200Hz, the filter order is 26, and it uses rectangular window.

The screenshot in below shows the filter specifications in this example.





QEDesign generates the filter coefficients in Q15 fractional format as below. They can be used directly with Cortex M4 CMSIS functions.

Frac16 filter[] = { 0x0391, //{coeff 0}

 0x03dd, //{coeff 1}

 0x0426, //{coeff 2}

 0x046a, //{coeff 3}

 0x04aa, //{coeff 4}

 0x04e4, //{coeff 5}

 0x0519, //{coeff 6}

 0x0547, //{coeff 7}

 0x056e, //{coeff 8}

 0x058d, //{coeff 9}

 0x05a5, //{coeff 10}

 0x05b6, //{coeff 11}

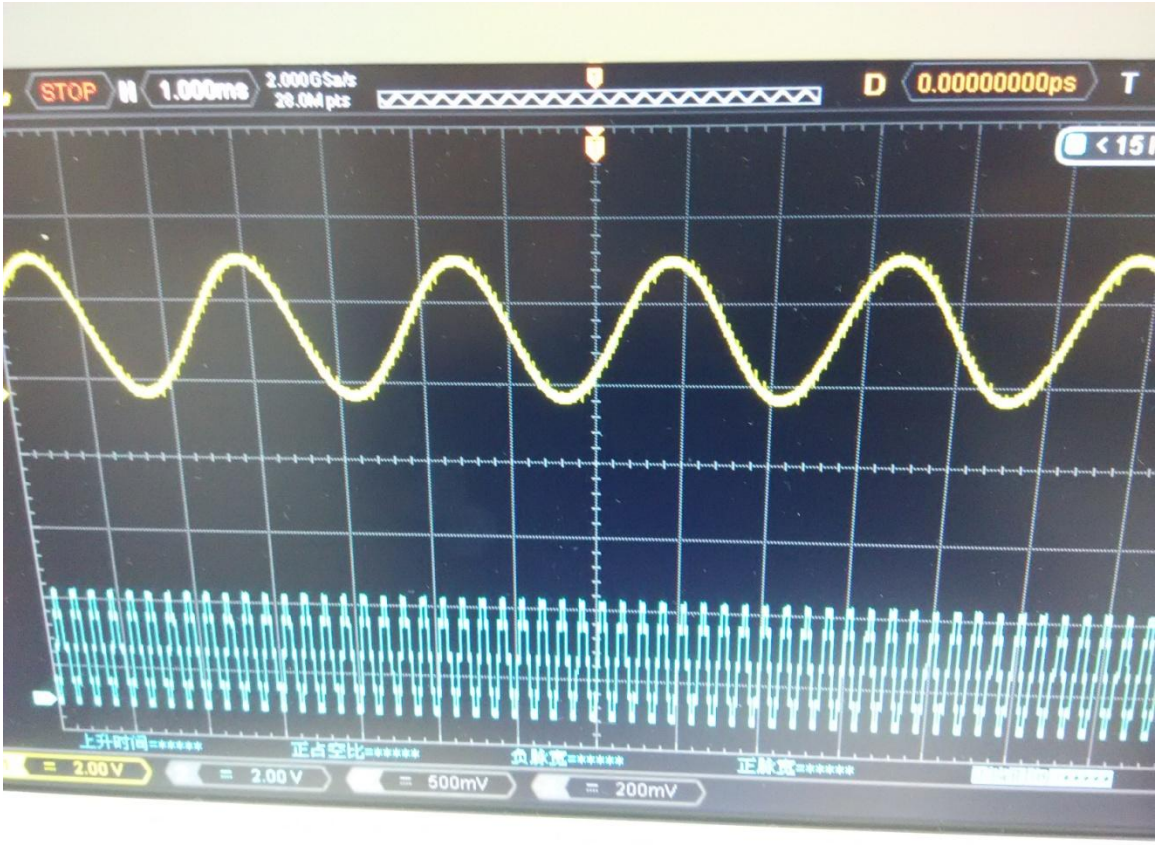
 0x05be, //{coeff 12}

```
0x05be, //{coeff 13}
0x05b6, //{coeff 14}
0x05a5, //{coeff 15}
0x058d, //{coeff 16}
0x056e, //{coeff 17}
0x0547, //{coeff 18}
0x0519, //{coeff 19}
0x04e4, //{coeff 20}
0x04aa, //{coeff 21}
0x046a, //{coeff 22}
0x0426, //{coeff 23}
0x03dd, //{coeff 24}
0x0391}; //{coeff 25}
```

5. Test result

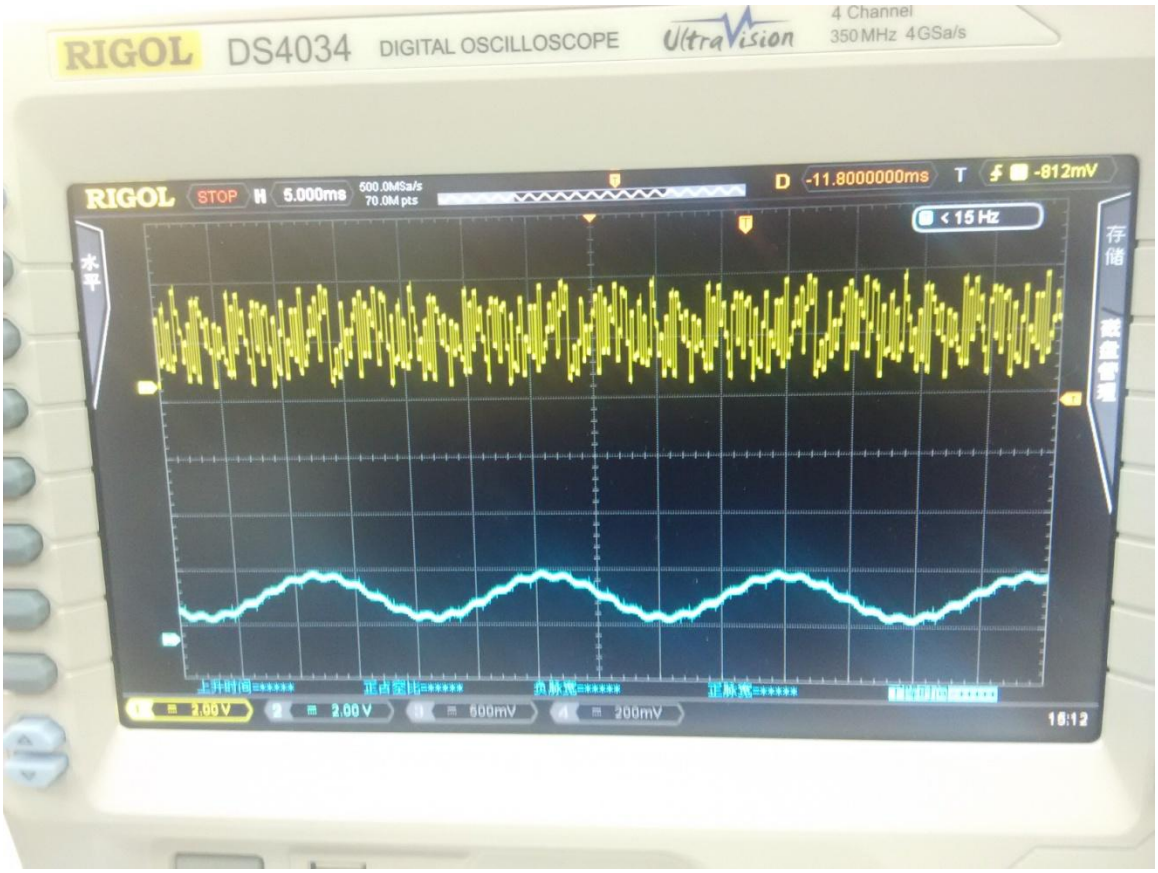
The example project runs on TWR-KV31F120M. The mixed signal before filtering is output though DAC0, the low frequency signal after filtering is output through DAC1 module on KV31F512.

The mixed signal is composed by a 50Hz sin wave (the upper one shown in the oscilloscope as below) and 500Hz sin wave (shown in lower graph).



The mixed signal which is composed by the two waves above is output though DAC0, shown as the upper graph in below.

After filter, the 500Hz signal is removed, and the 50Hz signal is kept. The lower graph in below shows the signal output from DAC1 after lowpass filtering.


RIGOL
DS4034
DIGITAL OSCILLOSCOPE
UltraVision
4 Channel
350 MHz 4 GSa/s
RIGOL
STOP
H
5.000ms
500.0MSa/s
70.0M pts
D
-11.8000000ms
T
-812mV
< 15 Hz
水平
存储
上升时间=*****
正占空比=*****
负脉宽=*****
正脉宽=*****
2.00 V
2.00 V
600mV
200mV
16:12