Creating new ASP algorithms using FIL

Performance Audio - EEE

ABSTRACT

This application report shows how to create Audio Stream Processing (ASP) Algorithms using TI-PA Filter Library/Framework, abbreviated as 'FIL', within the Performance Audio Framework from Texas Instruments (TI).

Contents

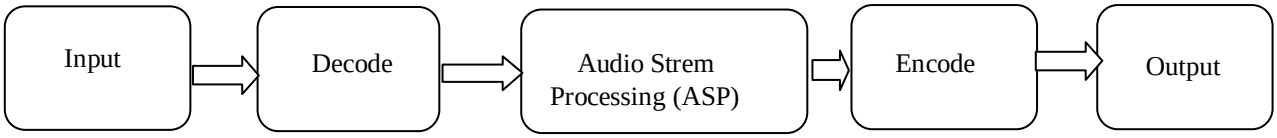
Figures

1 Brief introduction to various components

1.1 Performance Audio Framework (PA/F)

The major components in processing an audio stream, as implemented using the Performance Audio Framework (PA/F), are shown in Figure 1. The PA/F is described in the *PA User's Guide* (Reference 1). This application report discusses how to create ASP Algorithms that use FIL components, by explaining various FIL Audio Examples (FE) that can be finally plugged into the ASP chain of PA framework.

Figure 1. PA/F Audio Stream Components



1.2 PA/F Audio Stream Processing (ASP) interface

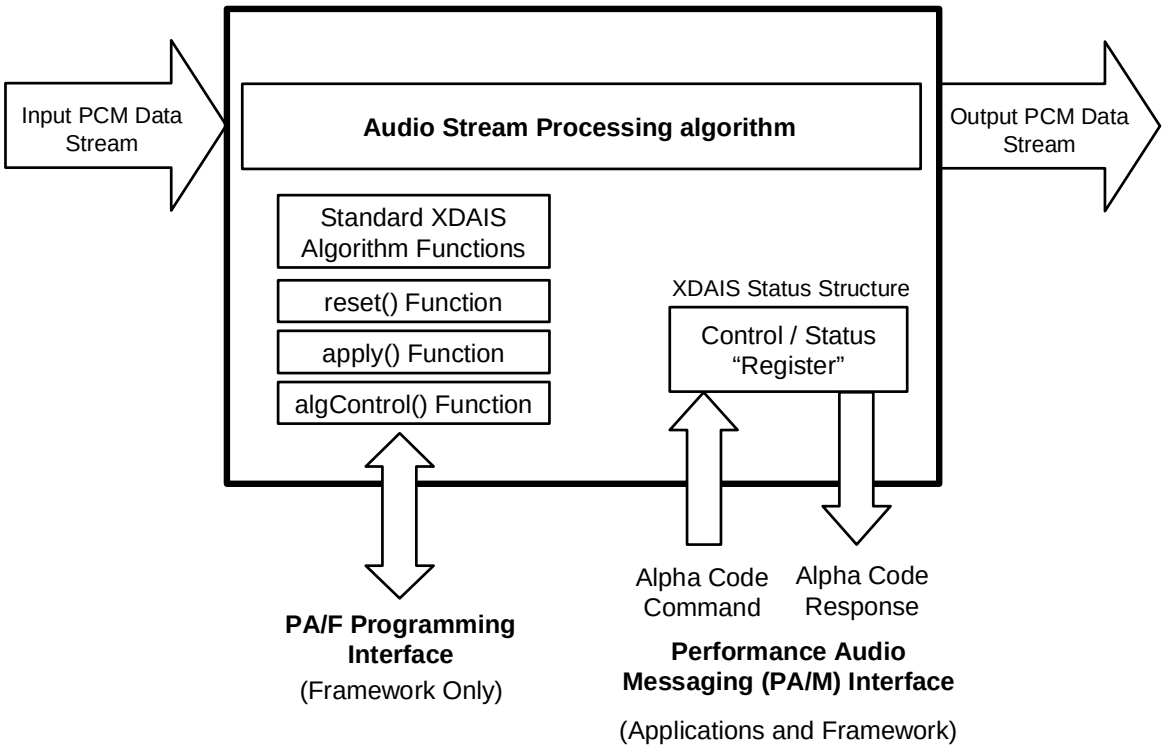


Figure 2. XDAIS-compliant Audio Stream Processing (ASP) Algorithm

Algorithm/Programming interface:

The PA/F ASP algorithm interface is an extended form of standard XDAIS Algorithm interface, in a fully–compliant way, to provide the required functionality for digital audio processing. As shown in the figure above, an ASP Algorithm extends the standard XDAIS Algorithm functions with three additional functions:

- reset( )
- apply( )
- algControl( )

The data structures used in these function interfaces are also standardized. The ASP Algorithm is instantiated and initialized using the XDAIS standard techniques. For more details refer to "*Audio Stream Processing*" Application Report(Reference 3).

The **reset()** function is called to initialize (or re-initialize) the "control data", state registers, etc associated with an algorithm. It is always called before the **apply()** function.

The **apply()** function essentially causes a frame of audio data to be processed (and to also handle the "control data" needed to manage the "processing chain").

Strictly speaking, the **algControl()** function is a standard XDAIS function, but its use is optional. When used, it is a standard way of providing algorithm–specific functionality, so it is listed here. The **algControl()** function must always provide an algorithm-specific *ALG_GETSTATUSADDRESS1* command, that returns the address of the control/status register inside the Algorithm. This is how the framework "connects" the Alpha Code messaging mechanism to the Algorithm.

Application/Messaging interface :

The **Performance Audio Messaging (PA/M) Interface** forms the Applications Interface ("API") to the Algorithm, and is the main method of communication between the "application" and the algorithm objects. The word "application", as used here, can mean an application program running on a separate host computer, or user code running on the same (or a different) DSP, inside the (perhaps embedded in the) audio processing system. Note, that the PA/F can also communicate with algorithms through the messaging interface (instead of directly through the programming interface). This "API" is described in more detail in the *PA User's Guide* (Reference 1). Standard PA/F Algorithms are documented in the appendices of the User's Guide. Typically, you send and receive Alpha Codes using the Galfa or Calfa tools on the PC (also described in the User's Guide).

1.3 TI-PA Filter Library/Framework (FIL)

Digital filters are undoubtedly the basic building blocks of Audio Signal Processing and the TI-PA Filter Library/Framework, abbreviated as “FIL”, aims at providing an easy tool/library for programmers to develop and implement Memory- and MIPS-efficient filter-based audio algorithms. FIL includes a library of filters for the TMS320C67xx platform, written in C, serial assembly, and hand assembly, and optimized for single/double precision data types. FIL has a filter-independent generic interface provided in two layers: i) XDAIS-ASP interface layer and ii) a simple filter function-call layer. These are explained in detail in the *FIL Application Report* (Reference 4). In layer-1, FIL control code takes care of calling the best filter implementation possible, with the given filter parameters. This helps to hide the TI-PA Framework and Filter Library specific details from the developer and make his job easier. The FIL Audio Example ASPs being developed here show how to use the XDAIS-ASP interface layer (layer-1) and the simple filter layer (layer-2) of FIL.

2 FIL Audio Examples

In the following pages, a number of Audio Stream Processing Algorithms will be presented which are implemented using FIL. Each **F**IL **E**xample (**FE**) has been given a single alphabetic index, starting from A to Z. Thus the first FIL Example will be called **FEA**. Further, different version of the same example will be indexed using a version number 0 to N. Hence the initial version of FEA will be **FEA0**.

2.1 FEA – Simple FIR filter

The FIL-ASP Audio Example (FEA) developed here is a simple 64 tap, Low Pass FIR filter, applied on Left, Right and Center channels of PA/F audio stream.

Filter design

Filter : Low Pass FIR filter of order 64 (excluding input gain ‘bo’).

Sampling frequency = 48KHz

F_{pass} *= 2KHz,* F_{stop} *= 4KHz*

Implementation structure is given below :

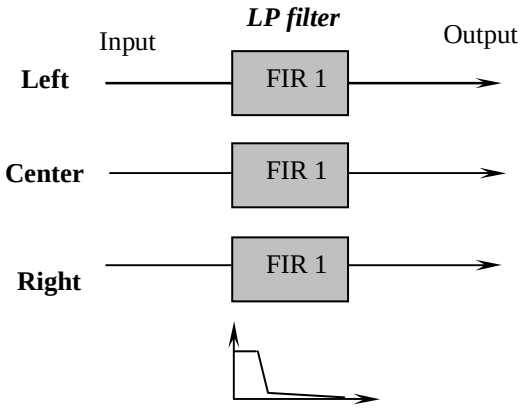


Figure 3. FIR implementation structure

As you can see in the above implementation structure, the same FIR coefficient set is applied on all the three channels. This property, uni-coefficient multiple channel filtering, is considered while implementing the filters, using FIL.

FEA0 : FIR implementation using FIL layer-1 interface

Only few of the Basic ASP Example Algorithm files were modified and they are listed below, with brief description about the modifications done. The modified code sections have been colored, for clarity.

Header files

1. ifea.h - Included FIL header files fil.h and fil_tii.h. Added FIL params and function pointers into FEB params and function pointer structures. Declared IFIL_PARAMS_FEA as extern.

```
#include "paftyp.h"
#include "fil.h"
#include "fil_tii.h"
```

```
typedef struct IFEA_Params {
    Int size;
    const IFEA_Status *pStatus;
    IFEA_Config *pConfig;
    const IFIL_Params *pFilParams;
} IFEA_Params;
```

```
extern const IFIL_Params IFIL_PARAMS_FEA;

/*
 *  ======== IFEA_Fxns ========
 *  All implementation's of FEA must declare and statically
 *  initialize a constant variable of this type.
 *
 *  By convention the name of the variable is FEA_<vendor>_IFEA, where
 *  <vendor> is the vendor name.
 */
typedef struct IFEA_Fxns {
    /* public */
    IALG_Fxns   ialg;
    Int         (*reset)(IFEA_Handle, PAF_AudioFrame *);
    Int         (*apply)(IFEA_Handle, PAF_AudioFrame *);
    /* private */
    IFIL_Fxns   *filFxns;
} IFEA_Fxns;
```

2. fea_tii_priv.h – Declared FEA_TII_numAlloc() as extern and included the FIL_Handle as an element in the FEA object handle.

```
extern Int  FEA_TII_numAlloc();

typedef struct FEA_TII_Obj {
    IALG_Obj alg;            /* MUST be first field of all XDAS algs */
    IRTC_Mask mask;          /* current test/diag mask setting */
    FEA_TII_Status *pStatus; /* public interface */
    FEA_TII_Config config; /* private interface */
    FIL_Handle firHandle;    /* Pointer to the FIL object handle */
```

```
} FEA_TII_Obj;
```

Source files

3. ifea.c – Defined default values of Status, Config. and Param structures of FEA.

```
const IFEA_Params IFEA_PARAMS = {
    sizeof(IFEA_Params),     /* size */
    &IFEA_PARAMS_STATUS,     /* pStatus */
    &IFEA_PARAMS_CONFIG,     /* config */
    &IFIL_PARAMS_FEA,
};
```

4. fea_coef.c - This contains the initialization values for creating the FIL objects. The FIL structures are initialized for “uni-coefficient, multi-channel, single precision, in-place, 64 tap FIR filtering, of possible channels L, R and Cnt” case.

Note that, since the same coefficient array is reused for all sample rates, we specify the FIL coefficient structure to be ‘single sample rate’ by setting the Multi-sample rate (MS) bit of ‘type’ field to 0. Refer to the *FIL Application Report* (Reference 4) for more details.

```
/* Headers */
#include "std.h"
#include "fil.h"
#include "fil_tii.h"

#define FIL_FIR_TYPE_FEA 0x200C0803 /* SP, 64 tap FIR, unicoeff, ch=1. */
#define FIL_CH_SELECT_FEA 0x7 /* Ch - L, R, Cnt */

Float gFIR_Coeff_FEA[65] = { /* Not shown */
};

/* FEA's FIL-coefficient struct, similar to PAF_FilCoef struct */
typedef struct FilCoef_FEA{
    LgUns type;       /* Filter type field */
    LgUns sampRate;   /* Sample rate bit field */
    Float *coef;      /* Coefficient ptr */
} FilCoef_FEA;

/* FEA's FIR initial coefficient structure */
FilCoef_FEA FilCoefFIR_FEA = {
    FIL_FIR_TYPE_FEA, /* Filter type field */
    0x0,              /* Sample rate bit field */

    gFIR_Coeff_FEA,   /* Coefficient ptr */
};

/* Status */
IFIL_Status IFIL_Status_FEA = {
             sizeof(IFIL_Status), /* size */
             FIL_CH_SELECT_FEA,   /* mode, ch - L,R,C */
             0x1,                 /* use, default is 'Enable' */
             FIL_CH_SELECT_FEA,   /* mask select, ch - L,R,C */
             0x0                  /* mask status */
};

/* Config */
IFIL_Config IFIL_Config_FEA = {
    sizeof(IFIL_Config),          /* size */
    (PAF_FilCoef *)&FilCoefFIR_FEA /* FIL-FIR filter Coefficients */
};
```

```
/* Param */
const IFIL_Params IFIL_PARAMS_FEA = {
    sizeof(IFIL_Params), /* size */
    1,                          /* use */
    &IFIL_Status_FEA,
    &IFIL_Config_FEA,
};

/* EOF */
```

5. fea_tii_ialg.c - Modified the algorithm functions for memory allocation and initialization and added numAlloc() function, which returns the number of memTabs created, and specified its memory section.

Function FEA_TII_free() will be called when the module is deleted. In this function, the memTabs are filled, in the same manner as done while requesting for memory in the alloc function, but with the base addresses set.

```
#pragma CODE_SECTION(FEA_TII_numAlloc,   ".text:algNumAlloc")
```

```
/*
 *  ======== FEA_TII_alloc ========
 */
Int FEA_TII_alloc(const IALG_Params *algParams,
                IALG_Fxns **pf, IALG_MemRec memTab[])
{
    const IFEA_Params *params = (Void *)algParams;
    Int n;

    if (params == NULL) {
        params = &IFEA_PARAMS;  /* set default parameters */
    }

    /* Request memory for FEA object */
    memTab[0].size = (sizeof(FEA_TII_Obj) + 3) /4*4 +
                     (sizeof(FEA_TII_Status) + 3 ) /4*4;
    memTab[0].alignment = 4;
    memTab[0].space = IALG_EXTERNAL;
    memTab[0].attrs = IALG_PERSIST;

    /* Call FIL alg alloc, with the next available memTab ptr */
    n = FEA_TII_IFEA.filFxns->ialg.algAlloc((const IALG_Params *)params->pFilParams,
                                                                    pf, &memTab[1]);

    return (1 + n); /* Return the number of memory requests */
}

/*
 *  ======== FEA_TII_deactivate ========
 */
Void FEA_TII_deactivate(IALG_Handle handle)
{
}

/*
 *  ======== FEA_TII_free ========
 */
Int FEA_TII_free(IALG_Handle handle, IALG_MemRec memTab[])
{
```

```
    FEA_TII_Obj *fea = (Void *)handle;

    memTab[1].base = fea->firHandle;

    /* Call alloc function to set the remaining parameters of memTabs */
    return (*handle->fxns->algAlloc)(NULL, NULL, memTab);
}

/*
 *  ======== FEA_TII_initObj ========
 */
Int FEA_TII_initObj(IALG_Handle handle,
                const IALG_MemRec memTab[], IALG_Handle p,
                const IALG_Params *algParams)
{
    FEA_TII_Obj *fea = (Void *)handle;
    const IFEA_Params *params = (Void *)algParams;
    FIL_Handle fil;

    if (params == NULL)
    {
        params = &IFEA_PARAMS;  /* set default parameters */
    }

    /* Get the status and config. pointers */
    fea->pStatus = (FEA_TII_Status *)((char *)fea + (sizeof(FEA_TII_Obj)+3)/4*4);
    *fea->pStatus = *params->pStatus;
    fea->config = *params->pConfig;

    fil = (FIL_Handle)memTab[1].base;

    /* Get the FIR-FIL handle into the status structure */
    fea->firHandle = fil;

    /* Initialize the fxns pointer */
    fil->fxns = (IFIL_Fxns *)((IFEA_Fxns *)(fea->alg.fxns))->filFxns;

    /* Call FIL algInit function */
    fil->fxns->ialg.algInit((IALG_Handle)fil, &memTab[1], p,
                            ( const IALG_Params *)params->pFilParams);

    return (IALG_EOK);
}
```

FEA_TII_numAlloc() function returns the number of memTab structures required for memory allocation for FEA. This is called by the ALG_create() XDAIS function. It can be called by any other function as well.

```
Int FEA_TII_numAlloc()
{
    return(1 + FEA_TII_IFEA.filFxns->ialg.algNumAlloc());
}
```

6. fea_tii_ifea.c – FEA_TII_apply() and FEA_TII_reset() implementations.

FIL object instantiation is expected to do “uni-coefficient, multi-channel, single precision, in-place, 64 tap FIR filtering, of possible channels L, R and Cnt”, by calling the FIL apply( ) function. The filtering specifications are given using the following parameters, which are set while initializing the FIL objects using values defined in the file fea_coef.c. Refer to the *FIL Application Report* (Reference 4) for more details.

Coefficient type field – "uni-coefficient, multi-channel, 64 tap IIR filtering “

Channel select template/mask – "filtering of possible channels L, R and Cnt"

FIL control code – It checks the coefficient and sample data types and also whether the processing is in-place or not, before invoking the appropriate filter function.

A simple reset logic is also added to the apply function, which resets the filter states only when FEA enters from 'Disable' mode to 'Enable' mode. The FEA reset function resets the filter states by calling FIL reset function.

```
Int FEA_TII_apply(IFEA_Handle handle, PAF_AudioFrame *pAudioFrame)
{
    FEA_TII_Obj * restrict fea = (Void *)handle; /* Module handle */
    FIL_Handle fil = fea->firHandle;

    /* On disabling FEA module, set the reset flag; so it resets the when enabled */
    if((fea->pStatus->mode==0) && (fea->pStatus->unused==0))
        fea->pStatus->unused = 1;

    if (fea->pStatus->mode == 0)
        return 0;   /* If mode is zero, ASP is Disabled, so exit. */

    /* Reset the filter states if FEA is nabled when reset flag is '1'*/
    if((fea->pStatus->mode==1) && (fea->pStatus->unused==1))
    {
        fil->fxns->reset(fil, pAudioFrame);
        fea->pStatus->unused = 0;
    }

    fil->fxns->apply(fil,pAudioFrame);

    return 0;
}

*
 *  ======== FEA_TII_reset ========
 *  TII's implementation of the reset function, which is like an "information"
 *  operation.  This is always called by the framework before calling the
 *  FEA_TII_apply function, and only deals with control data, filter states etc
 *  and not sample data.
 */

Int FEA_TII_reset(IFEA_Handle handle, PAF_AudioFrame *pAudioFrame)
{
    FEA_TII_Obj * restrict fea = (Void *)handle; /* FEA handle */
    FIL_Handle fil = fea->firHandle;

    fil->fxns->reset(fil, pAudioFrame);
    fea->pStatus->unused = 0; /* Set the reset flag to '0' */

    return 0;
}

/* EOF */
```

7. fea_tii_ialgv.c - – Added numAlloc in to IALGFXNS and added FIL functions table pointer, &FIL_TII_IFIL, into FEA functions table.

```
#define IALGFXNS \
    &FEA_TII_IALG,       /* module ID */                    \
    FEA_TII_activate,    /* activate */                     \
    FEA_TII_alloc,       /* alloc */                        \
    FEA_TII_control,     /* control */                       \
    FEA_TII_deactivate,  /* deactivate */                   \
```

```
    FEA_TII_free,        /* free */                              \
    FEA_TII_initObj,     /* init */                              \
    FEA_TII_moved,       /* moved */                             \
    FEA_TII_numAlloc     /* numAlloc (NULL => IALG_MAXMEMRECS) */\

/*
 *  ======== FEA_TII_IFEA ========
 *  This structure defines TII's implementation of the IFEA interface
 *  for the FEA_TII module.
 */
IFEA_Fxns FEA_TII_IFEA = {       /* module_vendor_interface */
    IALGFXNS,
    FEA_TII_reset,
    FEA_TII_apply,
    &FIL_TII_IFIL,
};
```

E.g. If L ,R and Cnt. channels were present in the audio stream, then it would have been a '3 channel, single precision, uni-coefficient, in-place, 64-tap FIR' filtering case for FIL.

So referring to Table 3 in *FIL Application Report* (Reference 4), it says 0.5 clks / tap/ channel. Thus for 64 tap filtering of 3 channels at a sampling rate of 48KHz will require,

MClks = 0.5*64*ch*samplingRate/10^6 = 0.5*64*3*48,000/10^6 = 4.608 MClks.

Thus the worst case MClks for the filtering part, excluding cache misses, is theoretically expected to be 4.608 MClks, which is around 2.048% load for a 225MHz DSP.

With this, the example ASP algorithm (FEA0), the simple LP-FIR implementation, is ready to be plugged into the ASP chain of PA/F.See ASP Porting Guide for more information on this.

FEA1 : FIR implementation using FIL layer-2 interface

The following will be the modifications to basic example ASP, if FEA is implemented using FIL layer-2, i.e. using *filter function* layer.

Header files

1. ifea.h - Included FIL header files fil.h and fil_tii.h. Header file fil_table.h contains the declaration of FIL filter functions. Added the *firCoeff* pointer element into *IFEA_Config* structure.

```
#include "fil.h"
#include "fil_tii.h"
#include "fil_table.h"
```

```
typedef struct IFEA_Config {
    Int size; /* Size of IFEA_Config */
    Float *firCoeff; /* Pointer to FIR filter coefficient structure */
} IFEA_Config;
```

2. fea_tii_priv.h – Declared FEA_TII_numAlloc() as extern and included FIL layre-2 handle and other few required elements into the *FEA_TII_Obj* structure. The purpose of the structure elements is explained in the comments.

```
extern Int  FEA_TII_numAlloc();

#define CH_FEA 3
```

```
#define FIR_CH_MASK_FEA 0x7
#define FIR_TAPS_FEA 64
#define FIL_FIR_TYPE_FEA 0x200C0801

typedef struct FEA_TII_Obj {
    IALG_Obj alg;           /* MUST be first field of all XDAS algs */
    IRTC_Mask mask;         /* current test/diag mask setting */
    FEA_TII_Status *pStatus; /* public interface */
    FEA_TII_Config config;   /* private interface */
    PAF_FilParam fil; /* FIL layer 2 handle */
    PAF_ChannelMask chMask; /* Bit mask of Filtered Ch */
    Void *chPtr[3*CH_FEA]; /* Ch data pointer array */
    Void *chPtrInit[2*CH_FEA]; /* Initial Ch pointer array */
    Int varPerCh;
} FEA_TII_Obj;
```

Source files

3. ifea.c – Defined default values of Config. and Param structures of FEA.

```
/* FIL default config */
IFEA_Config IFEA_PARAMS_CONFIG = {
    sizeof(IFEA_Config),     /* size */
    (Float *)&gFIR_Coeff_FEA[0],
};

const IFEA_Params IFEA_PARAMS = {
    sizeof(IFEA_Params),     /* size */
    &IFEA_PARAMS_STATUS,     /* pStatus */
    &IFEA_PARAMS_CONFIG,     /* config */
};
```

fea_coef.c – FIR coefficient array declaration.

```
/* FIR filter coefficients */
Float gFIR_Coeff_FEA[65] = {
/* Not shown */
}
```

fea_tii_ialg.c – The memory requirements for the filter is obtained by calling the FIL function *FIL_varsPerCh()*. Since the data pointers involved in FIL layer-2 handle vary depending on the PAF channel configuration, we maintain the initial data pointer values in the array *chPtrInit.*

```
#pragma CODE_SECTION(FEA_TII_numAlloc,   ".text:algNumAlloc")
```

```
Int FEA_TII_alloc(const IALG_Params *algParams,
                  IALG_Fxns **pf, IALG_MemRec memTab[])
{
    const IFEA_Params *params = (Void *)algParams;

    if (params == NULL) {
        params = &IFEA_PARAMS;  /* set default parameters */
    }

    /* Request memory for FEA object */
    memTab[0].size = (sizeof(FEA_TII_Obj) + 3) /4*4 +
                     (sizeof(FEA_TII_Status) + 3 ) /4*4;
    memTab[0].alignment = 4;
    memTab[0].space = IALG_EXTERNAL;
```

```
    memTab[0].attrs = IALG_PERSIST;

    /* Allocation for filter states */
    memTab[1].size = FIL_varsPerCh(FIL_FIR_TYPE_FEA)*CH_FEA;
    memTab[1].alignment = sizeof(Double);
    memTab[1].space = IALG_EXTERNAL;
    memTab[1].attrs = IALG_PERSIST;

    return (2); /* Return the number of memory requests */
}


Int FEA_TII_free(IALG_Handle handle, IALG_MemRec memTab[])
{
    FEA_TII_Obj *fea = (Void *)handle;

    memTab[1].base = fea->chPtrInit[0]; /* FIL-FEA var ptr */

    /* Call alloc function to set the remaining parameters of memTabs */
    return (*handle->fxns->algAlloc)(NULL, NULL, memTab);
}


Int FEA_TII_initObj(IALG_Handle handle,
                const IALG_MemRec memTab[], IALG_Handle p,
                const IALG_Params *algParams)
{
    FEA_TII_Obj *fea = (Void *)handle;
    const IFEA_Params *params = (Void *)algParams;
    Int i;

    if (params == NULL)
    {
        params = &IFEA_PARAMS;  /* set default parameters */
    }

    /* Get the status and config. pointers */
    fea->pStatus = (FEA_TII_Status *)((char *)fea + (sizeof(FEA_TII_Obj)+3)/4*4);
    *fea->pStatus = *params->pStatus;
    fea->config = *params->pConfig;

    /* Initialize the layer-2 filter handle structure elements */
    fea->fil.pOut = fea->fil.pIn = &fea->chPtr[0];
    fea->fil.pVar = &fea->chPtr[CH_FEA];
    fea->fil.channels = CH_FEA; /* L, R, and Cnt for FEA module */
    fea->fil.pCoef = fea->chPtr[2*CH_FEA];
    fea->fil.use = FIR_TAPS_FEA; /* FIR Filter taps */

    /* FEA Obj intialization */
    fea->chMask = 0x0;
    fea->varPerCh = FIL_varsPerCh(FIL_FIR_TYPE_FEA);

    /* Initialize the FEA chPtrInit structure */
    for(i = 0; i < CH_FEA; i++)
    {
        fea->chPtrInit[i] = (Float *)memTab[1].base + i*FIR_TAPS_FEA;
        fea->chPtrInit[CH_FEA + i] = params->pConfig->firCoeff; /* Unic-coeff */
    }

    return (IALG_EOK);
}
```

FEA_TII_numAlloc() function returns the number of memTab structures required for memory allocation for FEA. This is called by the ALG_create() XDAIS function. It can be called by any other function as well.

```
Int FEA_TII_numAlloc()
{
    return(2);
}
```

4. fea_tii_ifea.c – FEA_TII_apply(), FEA_TII_reset() and FIL_setup_FEA() implementations.

As you can see in the below code section, the FIR filter function *Filter_firTNChN*()is called directly using FIL layer-2 handle. Refer to respective filter appendices in *FIL Application Report* (reference 4) to get the proper filter function names, given in the tables.

The function *FIL_setup_FEA*() is called to setup the FIL layer-2 handle depending on the PAF stream's channel configuration. The ch data pointers are rearranged properly with respect to the channel configuration changes. For more details on FIL layer-2, refer to *FIL Application Report* (reference 4).

```
static Int FIL_setup_FEA(IFEA_Handle handle, PAF_AudioFrame *pAudioFrame);

Int FEA_TII_apply(IFEA_Handle handle, PAF_AudioFrame *pAudioFrame)
{
    FEA_TII_Obj * restrict fea = (Void *)handle; /* Module handle */

    /* On disabling FEA module, set the reset flag; so it resets the when enabled */
    if((fea->pStatus->mode==0) && (fea->pStatus->unused==0))
        fea->pStatus->unused = 1;

    if (fea->pStatus->mode == 0)
        return 0;   /* If mode is zero, ASP is Disabled, so exit. */

    /* Reset the filter states if FEA is nabled when reset flag is '1'*/
    if((fea->pStatus->mode==1) && (fea->pStatus->unused==1))
    {
        FIL_memReset((Void *)fea->chPtrInit[0],
                                fea->varPerCh*CH_FEA );
        fea->pStatus->unused = 0; /* Reset flag is disabled */
    }


    FIL_setup_FEA((IFEA_Handle)fea, pAudioFrame); /* Setup PAF_FilParam structure */

    Filter_firTNChN(&fea->fil); /* Calls FIR filter function directly */

    return 0;
}

Int FEA_TII_reset(IFEA_Handle handle, PAF_AudioFrame *pAudioFrame)
{
    FEA_TII_Obj * restrict fea = (Void *)handle; /* FEA handle */

    FIL_memReset((Void *)fea->chPtrInit[0], fea->varPerCh*CH_FEA );
    fea->pStatus->unused = 0; /* Set the reset flag to '0' */

    return 0;
}

static Int FIL_setup_FEA(IFEA_Handle handle, PAF_AudioFrame *pAudioFrame)
{
    FEA_TII_Obj * restrict fea = (Void *)handle; /* Module handle */
```

```
    PAF_ChannelMask satMask, subWMask, PAF_chMask, FEA_chMask;
    Int i;

    /* Satellite channel bit mask. */
    satMask =
          pAudioFrame->pChannelConfigurationMaskTable->sat.pMask
          [
           (XDAS_UInt8 )pAudioFrame->channelConfigurationStream.part.sat >=
           pAudioFrame->pChannelConfigurationMaskTable->sat.length      ?
           0 : (XDAS_UInt8 )pAudioFrame->channelConfigurationStream.part.sat
          ];

    /* SubWoofer channel bit mask. */
    subWMask =
          pAudioFrame->pChannelConfigurationMaskTable->sub.pMask
          [
           (XDAS_UInt8 )pAudioFrame->channelConfigurationStream.part.sub >=
           pAudioFrame->pChannelConfigurationMaskTable->sub.length      ?
           0 : (XDAS_UInt8 )pAudioFrame->channelConfigurationStream.part.sub
          ];

    PAF_chMask = satMask | subWMask;
    FEA_chMask = FIR_CH_MASK_FEA;

    if( fea->chMask != (PAF_chMask & FIR_CH_MASK_FEA) )
        /* Use the FIL function for resetting the filter states */
        FIL_memReset((Void *)fea->chPtrInit[0],
                                    fea->varPerCh*CH_FEA);

    fea->chMask = FIR_CH_MASK_FEA & PAF_chMask;
    fea->fil.channels = 0;

    for(i = 0; FEA_chMask; )
    {
        if( (PAF_chMask & 0x1) && (FEA_chMask & 0x1) )
        {
            fea->fil.pIn[i]   = pAudioFrame->data.sample[i];
            fea->fil.pOut[i]  = fea->fil.pIn[i]; /* Inplace */
            fea->fil.pVar[i]  = fea->chPtrInit[i];
            fea->fil.pCoef[i] = fea->chPtrInit[CH_FEA + i];
            ++fea->fil.channels;
            i++;
        }

        PAF_chMask >>= 1;
        FEA_chMask >>= 1;
    }

    fea->fil.sampleCount =  pAudioFrame->sampleCount;

    return 0;
}

/* EOF */
```

5. fea_tii_ialgv.c - – Added numAlloc in to IALGFXNS.

```
#define IALGFXNS \
    &FEA_TII_IALG,       /* module ID */                    \
    FEA_TII_activate,    /* activate */                     \
    FEA_TII_alloc,       /* alloc */                        \
    FEA_TII_control,     /* control */                      \
    FEA_TII_deactivate,  /* deactivate */                   \
    FEA_TII_free,        /* free */                         \
```

```
    FEA_TII_initObj,     /* init */                              \
    FEA_TII_moved,       /* moved */                             \
    FEA_TII_numAlloc     /* numAlloc (NULL => IALG_MAXMEMRECS) */\
```

The performance is almost same as of using FIL layer-1 implementation.

2.2 FEB – A ten band Graphic-Equalizer

The FIL-ASP Audio Example (FEB) developed here is a simple 10-band Graphic-Equalizer, applied commonly to all active channels (7.1 channel configuration), in PA/F audio stream.

Equalizer design

The implementation structure of the Equalizer (cascade form) is given below.

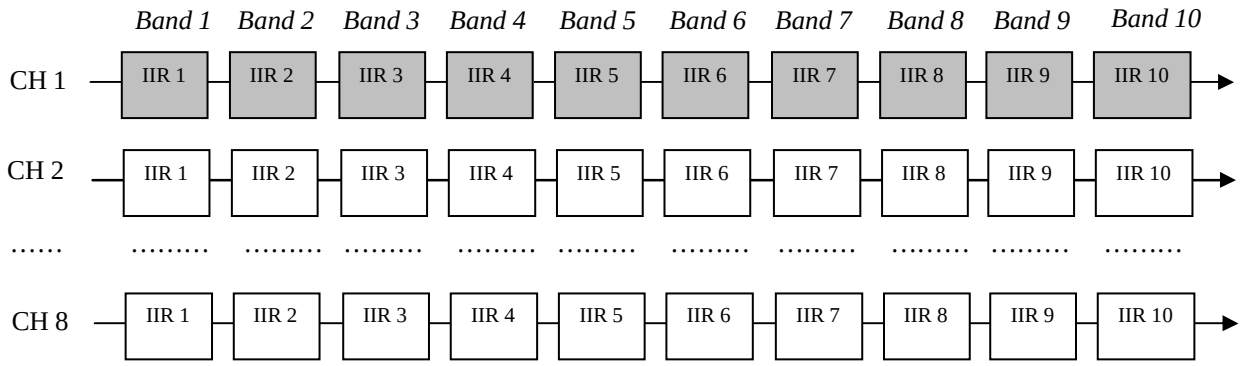


Figure 4. Implementation structure of the Equalizer

The coefficients of each IIR filter (biquad section) is dependent on two parameters, i) Band gain and ii) Sampling rate. As you can see in the implementation structure, for each equalizer band, the same IIR coefficient set is applied on multiple channels. This property, called uni-coefficient, is considered while implementing the filters.

The basic equations involved in calculating the coefficients for a single band are given below. These calculations are done in the function FEB_EQSetup (), which is explained later.

$$Z\text{-Transform, } H(z) = \frac{b0 + b1*z^{-1} + b2*z^{-2}}{a0 + a1*z^{-1} + a2*z^{-2}}$$

$Q = 1.2$

$w = 2\Pi * fc/fs$ *where, fc = cutoff freq. and fs = sampling freq.*

$c = -2\cos(w)$

$alpha = \sin(w) / (2Q)$

$A = sqrt(10^{(bandGain / 20)})$ *where, bandGain is assumed to be in dB units.*

$A0 = 1.0 + alpha / A$

$a0 = 1$ $a1 = c / A0$ $a2 = (1.0 - (alpha/A)) / A0$

*b0 = (1.0 + (alpha*A)) / A0 b1 = c / A0 b2 = (1.0 - (alpha*A)) / A0*

FEB1 : Equalizer implementation using FIL layer-1 interface

Only few of the FEA0 files were modified to create FEB1, and they are listed below with brief description about the modifications done.

Header files

6. ifeb.h – Declared Band Gain data type and included the Band Gain elements into FEB Status structure.

```
/*
 *  ======== GainDatatype ========
 *  This defines both the precision and datatype of the band gain.
 */
typedef signed char GainDatatype;

/*
 *  ======== IFEB_Status ========
 *  This Status structure defines the parameters that can be changed or read
 *  during real-time operation of the algorithm.  This structure is actually
 *  instantiated and initialized in ieq.c.
 */
typedef volatile struct IFEB_Status {

    Int size;             /* This value must always be here, and must be set to the
                             total size of this structure in 8-bit bytes, as the
                             sizeof() operator would do. */
    XDAS_Int8 mode;       /* This is the 8-bit FEB Mode Control Register.  All
                             Algorithms must have a mode control register.  */
    XDAS_Int8 unused;     /* This 8-bit register is unused, and is not only useful as a
                             "spare", but also provides 16-bit alignment to the next
                             value. */
    GainDatatype bandGain[10];/* Array of target band gains */
    Float gainStatus[10];    /* Array of present band gains */

} IFEB_Status;

/*
 *  ======== IFEB_Config ========
 *  Config structure defines the parameters that cannot be changed or read
 *  during real-time operation of the algorithm.
 */
typedef struct IFEB_Config {
    Int size; /* Size of IFEB_Config */
} IFEB_Config;
```

7. feb_tii_priv.h – Included the FIL handle, FIL coef and coefficient arrays in the FEB_TII object handle and declared FEB global variables as extern.

```
typedef struct FEB_TII_Obj {
    IALG_Obj alg;             /* MUST be first field of all XDAS algs */
    IRTC_Mask mask;           /* current test/diag mask setting */
    FEB_TII_Status *pStatus;  /* public interface */
    FEB_TII_Config config;    /* private interface */
    FIL_Handle filHandle;     /* Pointer to the FIL object handle */
    PAF_FilCoef_Void filCoef; /* FIL coef structure */
    Float coef[41];           /* Total 4*10bands + 1(gain) coefficient */
} FEB_TII_Obj;

extern Double  gFEB_EQFreqs[10];
```

```
extern Float gByEqSampleRate[15];
```

8. feb_a.h –This contains additional alpha codes for varying the gain of one band at a time and also for varying all the 10 band gains at a time.

```
/* 'Band gain' access alpha codes */
#define readFEBGainAll  0xf600+CUS_BETA_FEB,0x060a

/* Change all band gains */
#define writeFEBGainAll(A,B,C,D,E,F,G,H,I,J) 0xfe00+CUS_BETA_FEB, 0x060a, \
                     (0x00FF & A)|(B<<8),( 0x00FF & C)|(D<<8),( 0x00FF & E)|(F<<8), \
                     ( 0x00FF & G)|(H<<8),( 0x00FF & I)|(J<<8)
/* Change one band gain */
#define writeFEBBandGain(band,gain) 0xfe00+CUS_BETA_FEB, 0x0600+(band<<8)+0x01, gain
```

Source files

9. feb_coef.c – This contains the initialization values for creating the FIL objects. The center frequencies for the Bands are fixed.

```
/* Headers */
#include "std.h"
#include "fil.h"
#include "fil_tii.h"

/* TYPE : Filter group=IR Cascade(0x1) and Filter sub-group=IIR-SOS-DF2(0x1) */
/*        SP,10 Biquad Cascades(tap=20), unicoeff, ch=1                      */
#define FIL_IIRSOSDF2_TYPE_FEB 0x208C0281

/* Channel select template */
#define FIL_CH_SELECT_FEB 0x1F07    /* Ch - L,R,C,Ls,Rs,Lb,Rb,Sbw */

/* Band centre frequecies */
Double  gFEB_EQFreqs[10] ={
32.7, 65.4, 130.8, 261.6, 523.3, 1046.5, 2093.0, 4186.0, 8372.0, 16744.0
};

/* PA/F sample rates, corresponding to its index order */
Double gByEqSampleRate[15] = {
0.00003125,          /* 1/32000.0  */
2.26757369614512e-5, /* 1/44100.0  */
2.08333333333333e-5, /* 1/48000.0  */
1.13378684807256e-5, /* 1/88200.0  */
1.04166666666666e-5, /* 1/96000.0  */
5.20833333333333e-6, /* 1/192000.0 */
0.000015625,         /* 1/64000.0  */
0.0000078125,        /* 1/128000.0 */
5.66893424036281e-6, /* 1/176400.0 */
0.000125,            /* 1/8000.0   */
9.07029478458049e-5, /* 1/11025.0  */
8.33333333333333e-5, /* 1/12000.0  */
0.0000625,           /* 1/16000.0  */
4.53514739229024e-5, /* 1/22050.0  */
4.16666666666666e-5, /* 1/24000.0  */
};

/* FEB coefficient struct, similar to PAF_FilCoef_PAF struct */
typedef struct FilCoef_FEB {
    LgUns type;     /* Filter type field */
    LgUns sampRate; /* Sample rate bit field */
```

```
    Float *coef;    /* Coefficient ptr */
} FilCoef_FEB;

/* Initial value */
FilCoef_FEB FilCoefEQ_FEB =
{
    FIL_IIRSOSDF2_TYPE_FEB, /* Filter type field */
    0x0,                    /* Sample rate bit field */
    0,                      /* Coefficient ptr */
};

/* Status */
IFIL_Status IFIL_Status_FEB = {
    sizeof(IFIL_Status),/* size */
    FIL_CH_SELECT_FEB,  /* mode, ch - L,R,C,Ls,Rs,Lb,Rb,Sbw */
    0x1,                /* use, default is 'enable' */
    FIL_CH_SELECT_FEB,  /* mask select, ch - L,R,C,Ls,Rs,Lb,Rb,Sbw */
    0x0                 /* mask status */
};

/* Config */
IFIL_Config IFIL_Config_FEB = {
    sizeof(IFIL_Config),          /* size */
    (PAF_FilCoef *)&FilCoefEQ_FEB /* Eq filter Coefficients */
};

/* Param */
const IFIL_Params IFIL_PARAMS_FEB = {
    sizeof(IFIL_Params),    /* size */
    1,                        /* use */
    &IFIL_Status_FEB,
    &IFIL_Config_FEB
};

/* EOF */
```

10. feb_tii_ialg.c - As we have 10 cascaded biquads per channel, a FIL filter object, for FIL cascaded biquad(DF2) filter type, is created.

```
Int FEB_TII_alloc(const IALG_Params *algParams,
                  IALG_Fxns **pf, IALG_MemRec memTab[])
{
    Int i, n = 0;
    const IFEB_Params *params = (Void *)algParams;

    if (params == NULL) {
        params = &IFEB_PARAMS;  /* set default parameters */
    }

    /* Request memory for FEB object */
    memTab[0].size = (sizeof(FEB_TII_Obj) + 3) /4*4 +
                     (sizeof(FEB_TII_Status) + 3 ) /4*4;
    memTab[0].alignment = 4;
    memTab[0].space = IALG_EXTERNAL;
    memTab[0].attrs = IALG_PERSIST;

    n = FEB_TII_IFEB.filFxns->ialg.algAlloc((const IALG_Params *)params->pFilParams,
                                                                pf, &memTab[1]);
    return (1 + n); /* Return the number of memory requests */
}
```

```
Int FEB_TII_free(IALG_Handle handle, IALG_MemRec memTab[])
```

```
{
    FEB_TII_Obj *feb = (Void *)handle;
    FIL_TII_Obj *fil = (FIL_TII_Obj *)feb->bandHandle[0];
    Int i, n;

    /* FIL memTabs are set for free by calling FIL free function */
    fil = (FIL_Handle)feb->filHandle;
    fil->fxns->ialg.algFree((IALG_Handle)fil, &memTab[1]);

    /* Call alloc function to set the remaining parameters of memTabs */
    return (*handle->fxns->algAlloc)(NULL, NULL, memTab);
}
```

As is shown in the below code section, the FIL filter object is created and the FIL handle is stored in FEB handle as feb->filHandle.

```
Int FEB_TII_initObj(IALG_Handle handle,
                const IALG_MemRec memTab[], IALG_Handle p,
                const IALG_Params *algParams)
{
    FEB_TII_Obj *feb = (Void *)handle;
    const IFEB_Params *params = (Void *)algParams;
    FIL_Handle fil;
    Int i, n;

    if (params == NULL)
    {
        params = &IFEB_PARAMS;  /* set default parameters */
    }

    /* Get the status and config. pointers */
    feb->pStatus = (FEB_TII_Status *)((char *)feb + (sizeof(FEB_TII_Obj)+3)/4*4);
    *feb->pStatus = *params->pStatus;
    feb->config = *params->pConfig;

    /* Reset the filter coefficients */
    for(i = 0; i < 41; i++)
        feb->coef[i] = 0;

    /* Get FIL handle ptr of FIL band object */
    feb->filHandle = fil = (FIL_Handle)memTab[1].base;

    /* Initialize the fxns pointer */
    fil->fxns = ((IFEB_Fxns *)feb->alg.fxns)->filFxns;

    /* Call FIL algInit function */
    fil->fxns->ialg.algInit((IALG_Handle)fil, &memTab[1],
                            p,( const IALG_Params *)params->pFilParams);

    /* Initialize the band coefficients to behave as All Pass filters */
    feb->coef[0] = 1.0; /* B0 */

    feb->filCoef = *(PAF_FilCoef_Void *)IFIL_PARAMS_FEB.pConfig->pCoefs;

    /* Set the coefficient pointer for the filter object */
    feb->filCoef.cPtr[0] = (Void *)&feb->coef[0];

    ((FIL_TII_Obj *)fil)->pConfig->pCoefs = (PAF_FilCoef *)&feb->filCoef;

    for(i = 0; i < 10; i++)
    {
        /* Initialize the band gains to 0dB */
        feb->pStatus->bandGain[i]   = 0;
```

```
        feb->pStatus->gainStatus[i] = 0;
    }

    return (IALG_EOK);
}
```

```
Int FEB_TII_numAlloc()
{
    return(1 + FEB_TII_IFEB.filFxns->ialg.algNumAlloc());
}
```

11. feb_tii_ifeb.c – Added the FEB_EQSetup () function, which sets the filter coefficients, before calling the FIL apply() function.

FIL filter object instantiation is expected to do “uni-coefficient, multi-channel, cascaded(10) biquad(DF2) sections, single precision, in-place, IIR filtering, of possible channels L, R, C, Sbw, Ls, Rs, Lb and Rb”, by calling the FIL apply( ) function.

```
#include <math.h>

/* This function is made private to the application */
static Void FEB_EQSetup(FEB_TII_Obj *feb, PAF_AudioFrame *pAudioFrame);

Int FEB_TII_apply(IFEB_Handle handle, PAF_AudioFrame *pAudioFrame)
{
    FEB_TII_Obj * restrict feb = (Void *)handle; /* Module handle */

    /* On disabling FEB module, set the reset flag; so it resets the when enabled */
    if((feb->pStatus->mode==0) && (feb->pStatus->unused==0))
        feb->pStatus->unused = 1;

    if (feb->pStatus->mode == 0)
        return 0;   /* If mode is zero, ASP is Disabled, so exit. */

    /* Reset the filter states when, FEB is enabled while the reset flag is '1'*/
    if((feb->pStatus->mode==1) && (feb->pStatus->unused==1))
    {
          /* Reset the filter objects by calling FIL reset function */
          for( i = 0; i < 10; i++ )
              ((IFIL_Fxns *)((IFEB_Fxns *)(feb->alg.fxns))->filFxns)->reset
                                  (feb->bandHandle[i], pAudioFrame);

          feb->pStatus->unused = 0;
    }

    /* Update equalizer coeffs, depending on band gain and fs, before filtering */
    FEB_EQSetup(feb, pAudioFrame);

    /* FIL apply function */
   ((IFIL_Fxns *)((IFEB_Fxns *)(feb->alg.fxns))->filFxns)->apply
                                     (feb->filHandle,pAudioFrame);
    return 0;
}
```

FEB_EQSetup() function deals with the logic of designing and updating the filter coefficients, for the given band gains of the Equalizer. The function’s logic has 3 parts,

i) To check whether redesign of coefficients is required, if Equalizer gains have been changed.

ii) Increment the current Equalizer gains in a granular fashion; currently it is 0.05 of the difference, to reach the requested gain. This enables smooth transition from one Equalizer setup to another. During each iteration, the pAudioFrame->data.samsiz[ch], which informs the PA Framework about internal gains applied to channels, is updated with the maximum of the Equalizer band gains.

iii) Design of coefficients.

```
/*
 *  ======== FEB_EQSetup ========
 */
static Void FEB_EQSetup(FEB_TII_Obj *feb, PAF_AudioFrame *pAudioFrame)
{
    Int i, bands = 10;
    Double pi = 3.142857142857;
    Double bySampleRate;
    Double by_2Q = 1.0/(2.0*1.2); /* Q factor of the filter */
    Float maxGain, delta = 0.05; /* delta is the grain size of gain */
    Uint filteredCh, mask;
    Float gain, gainTarget, maxGainDiff;
    Float *cPtr;
    Int coeffFlag = 0;
    Double by_a0, by_b0;

    /* Check whether coefficient calculation is required */
    for (i = 0; i < bands; i++)
    {
        maxGainDiff = feb->pStatus->bandGain[i] - feb->pStatus->gainStatus[i];

        /* Is target gain of any band much different from its current status */
        if( (maxGainDiff > 0.01) || (maxGainDiff < -0.01))
            coeffFlag = 1;

    }

    /* Reset maxGain, which is used to update samSize */
    maxGain = 0;

    /* The gain is made to reach its target in a granular manner */
    for (i = 0; i < bands; i++)
    {
        /* Get the present and target gains */
        gainTarget = feb->pStatus->bandGain[i];
        gain       = feb->pStatus->gainStatus[i];

        /* Calculate the changed gain */
        gain = (gainTarget - gain)*delta + gain;

        feb->pStatus->gainStatus[i] = gain;

        /* Update maxGain */
        if(gain > maxGain)
            maxGain = gain;
    }

    /* Update the samSize of the filtered channels */
    mask = 1;
    filteredCh = ((FIL_TII_Obj *)(feb->filHandle))->pStatus->maskStatus;

    for(mask = 1, i = 0; i < sizeof(PAF_ChannelMask); i++, mask << 1 )
```

```
    {
        /* Reduce samSize by (maxGain/10.0)*2 */
        if(filteredCh & mask)
            pAudioFrame->data.samsiz[i] -= (PAF_AudioSize)(maxGain*0.2);
    }

    /* If coeff. calculation is not required, return without processing */
    if(!coeffFlag)
        return;

    /* Get the sample rate by referring to PA/F handle sample rate index */
    bySampleRate = gByEqSampleRate[pAudioFrame->sampleRate -2];

    /* Initialy set input gain as 1.0 */
    cPtr = (Float *)((PAF_FilCoef_Void *)(((FIL_TII_Obj *)(feb->filHandle))
                                          ->pConfig->pCoefs))->cPtr[0];
    cPtr[0] = 1.0;

    /* Calculate the coefficients for each band */
    for (i = 0; i < bands; i++)
    {
        Double w, c, alpha, A;
        Double a0, a1, a2, b0, b1, b2;

        /* Get the band gain, after factoring by 10 */
        gain = feb->pStatus->gainStatus[i]*0.1;

        /* w = 2Pi * cutOffFreq/sampFreq */
        w = 2.0 * pi * gFEB_EQFreqs[i] * bySampleRate;

        c = -2.0 * cos(w);

        alpha = sin(w) * by_2Q;

        A     = pow(10.0, gain*0.025);
        a0    = 1.0 + alpha/A;
        by_a0 = 1.0/a0;

        a1 = c * by_a0;
        a2 = (1.0 - (alpha/A)) * by_a0;
        b0 = (1.0 + (alpha*A)) * by_a0;
        b1 = a1;
        b2 = (1.0 - (alpha*A)) * by_a0;
        a0 = 1.0;

        /* b0 of each band-biquad is made a single input gain */
        by_b0          = 1.0/b0;
        cPtr[0 ]       = cPtr[0 ]*b0;
        cPtr[4*i + 1 ] = b1*by_b0;
        cPtr[4*i + 2 ] = b2*by_b0;
        cPtr[4*i + 3 ] = -a1;
        cPtr[4*i + 4 ] = -a2;

    } /* for (i = 0; i < bands; i++) */

} /* FEB_EQSetup() */
```

E.g. If all the above channels were present in the audio stream, then it would have been an ‘8 channel, single precision, uni-coefficient, in-place, cascaded(10) biquad(DF2)’ filtering case for FIL.

So referring to Table 4 in *FIL Application Report* (Reference 4), it says in worst case 3 clks / biquad/sample/ channel. Thus for filtering 8 channels at a sampling rate of 48KHz will require,

MClks/band = 3*ch*samplingRate/10^6 = 3*8*48,000/10^6 = 1.152 MClks/band.

Thus for a 10-Band equalizer, the worst case MClks for the filtering part, excluding cache misses, is theoretically expected to be 11.52 MClks, which is around 5.12% load for a 225MHz DSP.

References

1. *PA User's Guide*, Texas Instruments, Inc., 2009
2. *TI-PA Filter Library / Framework (FIL)* Application Report, Texas Instruments, Inc., 2002
3. *Making DSP Algorithms Compliant with the TMS320 DSP Algorithm Standard* (SPRA579B), Texas Instruments, Inc., November 2000
4. Baudendistel, Kurt, *Performance Audio Messaging* Application Report, Momentum Data Systems, Inc., 2002