

```

/*
    Texas Instruments
    Author - Phanindra, Texas Instruments
    v0    2020-01-02 - 1. Added conditional compilation
directive(TIENV) for TI test environment
                                2. Removed FBNCORaw: Obsolete
function
                                3. Removed functions clearAgcStatus,
getAgcLnaStatus. For Internal AGC mode. Please confirm that they
are not used
                                4. Removed repeat(and wrong)
instances of sleepModeConfiguration(), setTxDsaAttn()
                                5. Updated sleepModeConfiguration to
include blocks enable during wake up
                                6. Corrected following functions:
Corrected page selection logic      GetCurrentTxAttenuation:
Corrected Error bit field          getJesdRxDsaFifoErrors:
Corrected Error bit field          getJesdRxDsaMiscSerdesErrors:
the register address              maskJesdTxFifoErrors: Corrected
functionality                    tempSensorEnable: Corrected
type cast in frequency calculation findpllFrequency: Added Float
formula
                                txSigGenTone: Corrected
functionality
                                readFbNco: Corrected register
read combining logic
                                clearSrAlarms: Added clear for
all channels
                                getSrAlarms: Corrected Mapping
of channel to register read out
    v1    2020-01-14 - 1. Added tdd force parameters to override
Sleep function
    v1p1  2020-01-08 - 1. Updated the maskSrPapAlarms function:
Changed TX channel value to 0-3
                                2. Corrected following functions:
Removed memory shutdown          fbDataCapture and RxDataCapture:
selection error in FB            adcJesdRamp: Corrected step size
Corrected digital attenuation calculation
                                GetCurrentTxAttenuation:
AttackThreshold calculation      readFovrSettings: Corrected
                                3. Added following functions:
                                ConfigFovr function
                                readEnableDgc function
                                getSerdesRxDsaEyeMarginValue

```

function
getIntAlarms function
2020-01-10 - 1. Updated the configurePll function: DC PLL configuration is not supported
2. Corrected following functions:
configSrBlock: corrected PAP

alarm mask field
2020-01-13 - 1. Removed clearSrAlarms: similar function exists(clearPapAlarms)
2. Added following functions:
overrideAlarmPin function
overrideRelDetPin function
overrideDigPkDetPin function

2020-01-16 - 1. Updated following functions:
overrideAlarmPin: Added

configuration of GPIO
overrideRelDetPin: Added

configuration of GPIO
overrideDigPkDetPin: Added

configuration of GPIO and corrected channel selection
v1p2 2020-01-24 - 1. Added following functions:
BigStepAttackConfig
BigStepDecayConfig
2. Updated following functions:
SmallStepDecayConfig: Corrected

threshold
SmallStepAttackConfig: Corrected

threshold
txSigGenTone: Corrected signal

power
2020-01-28 - 1. Added following functions:
setLowIfNcoForAllTxFb
setAdcJesdRampPattern
getSerdesStatus
2020-02-03 - 1. Added following functions:
deepSleepModeConfiguration
deepSleepEn
2020-04-10 - 1. Updated txSigGenTone: corrected

functionality
2020-04-22 - 1. Added getSerdesEye function
Added setPllLoFbNcoFreezeQec
function: To configure pll when TX-QEC or RX-QEC is enabled
2. Updated setLowIfNcoForAllTxFb:
Added freeze and unfreeze of TX QEC
v1p3 2020-05-08 - 1. Updated ExtLnaControlConfig: corrected
functionality
Updated configureFovr: corrected
functionality

v1p4 2020-06-29 - 1. Updated setPllLoFbNcoFreezeQec: Tdd status is restored

v1p5 2020-07-13 - 1. Removed TxDsaAttn and TxDsaDigAttn: Obsolete function

digital gain

v1p6 2020-08-18 - 1. Removed configureFovr: Similar function exists

2. Updated setTxDsaAttn: Supports

2. Changed name of readconfigureFovr to readBigStepAttackSettings

3. Functions are arranged according to functionality

4. Added sleepEn function
Added rxPowDet function
Added txPowDet function

5. Fixed bug in getSerdesEye function

v1p7 2020-08-28 - 1. Added readSmallStepAttackConfig function

Added readSmallStepDecayConfig function

Added readBigStepDecayConfig function

Added auxAdcInSel function
Added readAuxAdc function
Added waitForMacroAck function
Added relinkJesd function for jesd204c mode

2. Added rxDsaCalibration function

2. Updated readFbNco function:
Updated getIntAlarms function:
Updated FbNCO function: Band1

Added band 1 frequency read

Separate readback for each pin

NCO frequency is programmable

Updated waitForMacroDone function: Added error code detection

Updated setPllLoFbNcoFreezeQec function: Added FB NCO band1 program

Minor updates to some functions: comments and return value

v1p8 2020-09-04 - 1. Updated rxDsaCalibration function
Updated relinkJesd function
Updated setPllLoFbNcoFreezeQec function

Updated checkPllLockStatus function: Removed unnecessary lines

Updated rxPowDet function

2. Added overrideTdd function
Added getSerdesEyeInmV function
Added rxPowerAttackConfig function

Added readRxPowerAttackConfig function

Added rxPowerDecayConfig function

```

function
    Added readRxPowerDecayConfig
v1p9 2020-09-25 - 1. Added txIqmcEstimatorStartDelay
function
    Added checkSysrefFlags function
    2. Updated configurePll function
    Updated freezeTxIqmc function:
Added separate control for AB and CD
    Updated freezeTxLol function:
Added separate control for AB and CD
    Updated rxPowDet function: Added
AGC freeze/unfreeze, fixed power calculation
    3. Removed setPllLoFbNco function:
Obsolete function

*/

#include <stdio.h>
#include <stdint.h>
#include <sys/types.h>
#include <math.h>
#include "interface.h"
#include <time.h>

typedef unsigned char      U8;
typedef unsigned short    U16;
typedef unsigned int      U32;
typedef unsigned long long U64;
typedef int               S32;

#ifdef TIENV

int AFE77xx_RegIfOpen(char *name){
    int status = ftdi_open(name);
    return status;
}

int AFE77xx_RegIfClose(){
    int status = ftdi_close();
    return status;
}

void delay(int milliseconds){
    long pause;
    clock_t now, then;

    pause = milliseconds*(CLOCKS_PER_SEC/1000);
    now = then = clock();
    while( (now-then) < pause )
        now = clock();
}

```

```

}

int AFE77xx_RegWrite(int fd,int address, U32 data){
    int writesstatus = ftdi_writeReg(address,data);
    return writesstatus;
}

int AFE77xx_RegRead(int fd,int address, U32 *data){
    delay(100);
    if(address<0x8000){
        address = 0x8000+address;
    }
    U32 val = ftdi_readReg(address);
    *data = val;
    return (int)val;
}

int AFE77xx_RegReadWrite(int fd, U16 address, U8 data, U8 lsb, U8
msb){
    U32 temp = 0;
    AFE77xx_RegRead(fd,address,&temp);
    for(int i=lsb;i<=msb;i=i+1){
        temp = (temp&(0xff-(1<<i)))|(data&(0x00+(1<<i)));
    }
    AFE77xx_RegWrite(fd,address,temp);
    return 0;
}

int sysUsDelay(int waittime){
    delay(waittime/1000);
    return 0;
}

#else

int AFE77xx_RegWrite(int fd,int address, U32 data){
    /*return writeOp(fd,address,data);*/
    printf("0x%4x\t0x%2x\n",address,data);
}

int AFE77xx_RegRead(int fd,int address, U32 *data){
    /*return readOp(fd,address,*data);*/
    printf("Read 0x%4x\n",address);
}

int AFE77xx_RegReadWrite(int fd, U16 addr, U8 data, U8 lsb, U8
msb){
    printf("Poll0x%4x\t0x%2x\n",addr,data);
    /*readWriteOp(fd, addr, data, lsb, msb)*/
}

int sysUsDelay(int waittime){

```

```

        /*return waitMs(waittime/1000);*/
        printf("Wait %d\n",waittime);
    }
#endif

U32 SerdesWrite(int fd, U32 serdesInstance, U32 address, U32
data){
    address=(address+0x2000)&0x3fff;
    address=address<<1;
    if (serdesInstance==0){
        AFE77xx_RegWrite(fd,0x0015,0x04);
    }
    else{
        AFE77xx_RegWrite(fd,0x0015,0x40);
    }
    AFE77xx_RegWrite(fd,address+1,data>>8);
    AFE77xx_RegWrite(fd,address,data&0xff);
    AFE77xx_RegWrite(fd,0x0015,0x00);
    return 0;
}

U32 SerdesLaneWrite(int fd, U32 ulLaneno, U32 address, U32 data){
    /* ulLaneno is the JESD lane Number from 0 to 7. address and
data are of the format 0x8032 */
    U32 jesdToSerdesLaneMapping[8]={3,2,0,1,1,0,2,3};
    address=address+(0x100*jesdToSerdesLaneMapping[ulLaneno]);
    SerdesWrite(fd,(ulLaneno>>2),address,data);
    return 0;
}

U32 SerdesRead(int fd, U32 serdesInstance, U32 address){
    U32 ulRegValue = 0;
    U32 ulValue=0;
    address=(address+0x2000)&0x3fff;
    address=address<<1;
    if (serdesInstance==0){
        AFE77xx_RegWrite(fd,0x0015,0x04);
    }
    else{
        AFE77xx_RegWrite(fd,0x0015,0x40);
    }
    /* It is important to read each Byte twice, but only the
second matters. */
    AFE77xx_RegRead(fd,address+1,&ulRegValue);
    AFE77xx_RegRead(fd,address+1,&ulRegValue);
    ulValue=ulValue+(ulRegValue<<8);
    AFE77xx_RegRead(fd,address,&ulRegValue);
    AFE77xx_RegRead(fd,address,&ulRegValue);
    ulValue=ulValue+(ulRegValue&0xff);
    AFE77xx_RegWrite(fd,0x0015,0x00);
    return ulValue;
}

```

```

U32 SerdesLaneRead(int fd, U32 ulLaneno, U32 address){
    /* ulLaneno is the JESD lane Number from 0 to 7. address and
data are of the format 0x8032 */
    U32 ulValue=0;
    U32 jesdToSerdesLaneMapping[8]={3,2,0,1,1,0,2,3};
    address=address+(0x100*jesdToSerdesLaneMapping[ulLaneno]);
    ulValue=SerdesRead(fd, (ulLaneno>>2), address);
    return ulValue;
}

int twos_comp(int val,int bits){
    if((val &(1<<(bits-1))) != 0){
        val = val-(1<<bits);
    }
    return val;
}

int waitForMacroReady(fd){
    U32 regValue=0;
    U16 count=0;
    AFE77xx_RegWrite(fd,0x13,0x10);
    while (count<0xffff){
        AFE77xx_RegRead(fd,0xf0,&regValue);
        if ((regValue&1)==1){
            break;
        }
        sysUsDelay(1000);
        count++;
    }
    AFE77xx_RegWrite(fd,0x13,0x00);
    if (count>=0xffff){
        printf("ERROR: Macro Ready Wait, time out");
        return -1;
    }
    else{
        return 0;
    }
}

int waitForMacroDone(fd){
    U32 regValue=0;
    U32 regValue2=0;
    U16 count=0;
    AFE77xx_RegWrite(fd,0x13,0x10);
    while (count<0xffff){
        AFE77xx_RegRead(fd,0xf0,&regValue);
        if ((regValue&4)==4){
            if((regValue&0x8)==0x8){
                AFE77xx_RegRead(fd,0xf1,&regValue2);
                printf("ERROR: Macro Execution Error\n");
                printf("ERROR in Opcode: %d\n",regValue2);
            }
        }
        count++;
    }
}

```

```

        AFE77xx_RegRead(fd,0xf2,&regValue2);
        printf("ERROR Code: %d\n",regValue2);
    }
    else
    {
        printf("PASS: Macro Execution Successful
\n");
    }
    break;
}
sysUsDelay(1000);
count++;
}
AFE77xx_RegWrite(fd,0x13,0x00);
if (count>=0xffff){
    printf("ERROR: Macro Done Wait, time out");
    return -1;
}
else{
    return 0;
}
}

int waitForMacroAck(fd){
    U32 regValue=0;
    U16 count=0;
    AFE77xx_RegWrite(fd,0x13,0x10);
    while (count<0xffff){
        AFE77xx_RegRead(fd,0xf0,&regValue);
        if ((regValue&2)==2){
            break;
        }
        sysUsDelay(1000);
        count++;
    }
    AFE77xx_RegWrite(fd,0x13,0x00);
    if (count>=0xffff){
        printf("ERROR: Macro Ack Wait, time out");
        return -1;
    }
    else{
        return 0;
    }
}

/* JESD-SERDES related functions */
U32 SetSerdesTxCursor(int fd, U32 ulLaneno, U32
mainCursorSetting, U32 preCursorSetting, U32 postCursorSetting){
    /* ulLaneno is SERDES lane from 0-7.*/
    U32 cursorSetting;
    cursorSetting=(mainCursorSetting<<5)
+(postCursorSetting<<11) + (preCursorSetting<<8);

```



```

        SerdesLaneWrite(fd, ulLaneno, 0x80f6, cursorSetting);
        return 0;
    }

U32 getSerdesStatus(int fd, int laneNo){
    U32 eyeMarginValue=0;
    U32 cdrLock=0;
    U32 CTLEVal=0;
    eyeMarginValue=SerdesLaneRead(fd, laneNo, 0x8030)&0xfff;
    printf("The Eye Margin Value fpr lane %d is %ld. This value
*0.5mV is the approximate Eye Height.\n", laneNo, eyeMarginValue);
    cdrLock=(SerdesLaneRead(fd, laneNo, 0x804e)>>15)&1;
    if (cdrLock==1){
        printf("The CDR for lane %d is locked\n", laneNo);
    }
    else{
        printf("The CDR for lane %d is Not locked\n", laneNo);
    }
    CTLEVal=(SerdesLaneRead(fd, laneNo, 0x801d)>>4)&7;
    printf("The CTLE Value for lane %d is %d. The lower the
value is, the higher the loss of the lane.\n", laneNo, CTLEVal);
    return 1;
}

double getSerdesEyeInmV(int fd, int laneNo){
    U32 eyeReg;
    U32 dacReg;
    U32 dacVal;
    double eyeInmV;
    eyeReg=(SerdesLaneRead(fd, laneNo, 0x802c)&0xfff);
    dacReg=(SerdesLaneRead(fd, laneNo, 0x801D)&0x1000);
    if (dacReg==1){
        dacVal=((dacReg&0xf00)>>8);
    }
    else{
        dacVal=((SerdesLaneRead(fd, laneNo, 0x8023)&0xf0)>>4);
    }
    eyeInmV((((float) (eyeReg)/2048.0)*(200.0+(50*(float)
(dacVal)))));
    printf("Eye value in mV:%f\n", eyeInmV);
    return eyeInmV;
}

U32 getJesdRxlaneErrors(int fd, U32 ulLaneno){
    int ulRegValue, a;
    const char *laneErrorBits[8];

    if ((ulLaneno&0x4)==0){
        AFE77xx_RegWrite(fd, 0x0015, 0x02);
    }
    else{
        AFE77xx_RegWrite(fd, 0x0015, 0x20);
    }
}

```

```

    }
    AFE77xx_RegRead(fd, 0x164+(ulLaneno&0x3), &ulRegValue);
    AFE77xx_RegWrite(fd, 0x0015, 0x00);

    laneErrorBits[7] = "multiframe alignment error";
    laneErrorBits[6] = "frame alignment error";
    laneErrorBits[5] = "link configuration error";
    laneErrorBits[4] = "elastic buffer overflow (bad RBD
value)";
    laneErrorBits[3] = "elastic buffer match error. The first
no-/K/ does not match 'match_ctrl' and 'match_data' programmed
values";
    laneErrorBits[2] = "code synchronization error";
    laneErrorBits[1] = "JESD 204B: 8b/10b not-in-table code
error. JESD 204C: sync_header_invalid_err";
    laneErrorBits[0] = "JESD 204B: 8b/10b disparity error. JESD
204C: sync_header_parity_err";
    for( a = 0; a < 8; a = a + 1 ){
        if (((ulRegValue>>a)&1)==1){
            printf("JESD RX Lane Error for lane %d seen: %s
\n", ulLaneno, laneErrorBits[a]);
        }
    }
    return 0;
}

U32 getJesdRxFifoErrors(int fd, U32 ulLaneno){
    int ulRegValue, a;
    const char *fifoErrorBits[2];

    if ((ulLaneno&0x4)==0){
        AFE77xx_RegWrite(fd, 0x0015, 0x02);
    }
    else{
        AFE77xx_RegWrite(fd, 0x0015, 0x20);
    }
    AFE77xx_RegRead(fd, 0x162+((ulLaneno&3)>>1), &ulRegValue);
    AFE77xx_RegWrite(fd, 0x0015, 0x00);

    ulRegValue=(ulRegValue>>(4*(ulLaneno&1)));
    ulRegValue=ulRegValue&0xf;

    fifoErrorBits[1] = "write_error : High if write request and
FIFO is full (NOTE: only released when JESD block is initialized
with init_state)";
    fifoErrorBits[0] = "read_error : High if read request with
empty FIFO (NOTE: only released when JESD block is initialized
with init_state)";

    for( a = 0; a < 2; a = a + 1 ){
        if (((ulRegValue>>a)&1)==1){
            printf("JESD RX Lane Fifo Error for lane %d seen:

```

```

%s\n", ulLaneno, fifoErrorBits[a]);
        }
    }
    return 0;
}

U32 getJesdRxMiscSerdesErrors(int fd, U32 jesdNo){
    /* jesdNo is 0 for top 4 lanes and 1 for bottom 4 lanes */
    int ulRegValue, a, ulLaneno, ulErrorValue;
    const char *serdesErrorBits[2], *miscErrorBits[4];

    if (jesdNo==0){
        AFE77xx_RegWrite(fd, 0x0015, 0x02);
    }
    else{
        AFE77xx_RegWrite(fd, 0x0015, 0x20);
    }

    miscErrorBits[0]="JESD sysref error";
    miscErrorBits[1]="JESD shorttest alarm";
    miscErrorBits[2]="NA";
    miscErrorBits[3]="SERDES PLL loss of lock";

    AFE77xx_RegRead(fd, 0x160, &ulRegValue);

    for( a = 0; a < 4; a = a + 1 ){
        if ((ulRegValue>>a)&1)==1){
            printf("JESD RX Serdes Error seen: %s
\n", miscErrorBits[a]);
        }
    }

    AFE77xx_RegRead(fd, 0x161, &ulRegValue);
    AFE77xx_RegWrite(fd, 0x0015, 0x00);

    serdesErrorBits[1] = "JESD frame_sync_err";
    serdesErrorBits[0] = "LOS indicator";

    for (ulLaneno=0;ulLaneno<4;ulLaneno=ulLaneno+1){
        ulErrorValue=((ulRegValue>>(ulLaneno))&
1)+((ulRegValue>>(ulLaneno+3))&2));
        for( a = 0; a < 2; a = a + 1 ){
            if ((ulErrorValue>>a)&1)==1){
                printf("JESD RX Lane Serdes Error for lane
%d seen: %s\n", ulLaneno+(jesdNo<<2), serdesErrorBits[a]);
            }
        }
    }
    return 0;
}

U32 getJesdRxAlarms(int fd){

```

```

U32 ulLaneno;

for (ulLaneno=0;ulLaneno<8;ulLaneno=ulLaneno+1){
    getJesdRxLaneErrors (fd,ulLaneno);
    getJesdRxLaneFifoErrors (fd,ulLaneno);
}

for (ulLaneno=0;ulLaneno<2;ulLaneno=ulLaneno+1){
    getJesdRxMiscSerdesErrors (fd,ulLaneno);
}
}

U32 getJesdRxLinkStatus(int fd){
    /* Reads the lane enables and accordingly returns the status
of the link
    Return Value is 4 bits. 2 bits for top 4 lanes and 2
bits for bottom 4 lanes.
state.
characters mode.
=0 Idle state. No change in
=1 CGS Passed. Still in K
=2 Link is up.
*/
    int laneEna0, laneEna1;
    int csState0, csState1;
    int expectedCsState0, expectedCsState1;
    int expectedFsState0, expectedFsState1;
    int fsState0, fsState1;
    int linkStatus0, linkStatus1;
    AFE77xx_RegWrite (fd, 0x0015, 0x02);
    AFE77xx_RegRead (fd, 0x7c, &laneEna0);
    AFE77xx_RegRead (fd, 0x12a, &csState0);
    AFE77xx_RegRead (fd, 0x12c, &fsState0);
    AFE77xx_RegWrite (fd, 0x0015, 0x00);

    expectedCsState0=0;
    expectedFsState0=0;
    for (int i=0;i<4;i++){
        if ((laneEna0>>i)&1)!=0){
            expectedCsState0=expectedCsState0+(2<<(i<<1));
            expectedFsState0=expectedFsState0+(1<<(i<<1));
        }
    }

    if (expectedCsState0==csState0){
        if (expectedFsState0==fsState0){
            linkStatus0=2;
        }
        else{
            linkStatus0=1;
        }
    }
}

```

```

else{
    linkStatus0=0;
}

AFE77xx_RegWrite(fd,0x0015,0x20);
AFE77xx_RegRead(fd,0x7c,&laneEnal);
AFE77xx_RegRead(fd,0x12a,&csStatel);
AFE77xx_RegRead(fd,0x12c,&fsStatel);
AFE77xx_RegWrite(fd,0x0015,0x00);

expectedCsStatel=0;
expectedFsStatel=0;
for (int i=0;i<4;i++){
    if (((laneEnal>>i)&1)!=0){
        expectedCsStatel=expectedCsStatel+(2<<(i<<1));
        expectedFsStatel=expectedFsStatel+(1<<(i<<1));
    }
}

if (expectedCsStatel==csStatel){
    if (expectedFsStatel==fsStatel){
        linkStatus1=2;
    }
    else{
        linkStatus1=1;
    }
}
else{
    linkStatus1=0;
}
return ((linkStatus1<<2)+linkStatus0);
}

U32 getJesdTxFifoErrors(int fd, U32 jesdNo){
    U32 ulLaneno;
    U32 ulRegValue=0;
    if (jesdNo==0){
        AFE77xx_RegWrite(fd,0x0015,0x01);
    }
    else
    {
        AFE77xx_RegWrite(fd,0x0015,0x10);
    }
    AFE77xx_RegRead(fd,0x005e,&ulRegValue);

    for (ulLaneno=0;ulLaneno<4;ulLaneno=ulLaneno+1){
        if (((ulRegValue>>ulLaneno)&1)==1){
            printf("JESD TX Lane Serdes FIFO Error for lane %
d.\n", ulLaneno);
        }
    }
}

```

```

AFE77xx_RegWrite(fd,0x0015,0x00);
}

U32 jesdTxSetSyncOverride(int fd, int override, int sendData){
    /* overrideWhen override is 1, the sync pin will be
overridden.
        In this case, if sendData is 0, K28.5 characters will
be sent by ASIC on the lanes, else data will be sent.
    */
AFE77xx_RegWrite(fd,0x0015,0x11);
if (override==1 && sendData==1){
AFE77xx_RegWrite(fd,0x0072,0xFF);
}
else if (override==1 && sendData==0){
AFE77xx_RegWrite(fd,0x0072,0xF0);
}
else{
AFE77xx_RegWrite(fd,0x0072,0x00);
}
AFE77xx_RegWrite(fd,0x0015,0x00);
return 0;
}

U32 maskJesdRxDxLaneErrors(int fd, U32 ulLaneno,U32 maskValue){
    /*
maskValue: The bits made 1 will be masked and not reflect on
pin.
laneErrorBits[7] = "multiframe alignment error";
laneErrorBits[6] = "frame alignment error";
laneErrorBits[5] = "link configuration error";
laneErrorBits[4] = "elastic buffer overflow (bad RBD
value)";
laneErrorBits[3] = "elastic buffer match error. The first
no-/K/ does not match 'match_ctrl' and 'match_data' programmed
values";
laneErrorBits[2] = "code synchronization error";
laneErrorBits[1] = "8b/10b not-in-table code error";
laneErrorBits[0] = "8b/10b disparity error";
    */
int ulRegValue;

if ((ulLaneno&0x4)==0){
AFE77xx_RegWrite(fd,0x0015,0x02);
}
else{
AFE77xx_RegWrite(fd,0x0015,0x20);
}
ulLaneno=ulLaneno&3;
AFE77xx_RegWrite(fd,0x174+ulLaneno,maskValue);
AFE77xx_RegWrite(fd,0x0015,0x00);
return 0;
}

```

```

U32 maskJesdRxFifoErrors(int fd, U32 ulLaneno,U32 maskValue){
    /*
        maskValue: The bits made 1 will be masked and not reflect on
pin.
        fifoErrorBits[3] = "write_error : High if write request and
FIFO is full (NOTE: only released when JESD block is initialized
with init_state)";
        fifoErrorBits[2] = "write_full : FIFO is FULL";
        fifoErrorBits[1] = "read_error : High if read request with
empty FIFO (NOTE: only released when JESD block is initialized
with init_state)";
        fifoErrorBits[0] = "read_empty : FIFO is empty";
    */
    int ulRegValue;

    if ((ulLaneno&0x4)==0){
        AFE77xx_RegWrite(fd,0x0015,0x02);
    }
    else{
        AFE77xx_RegWrite(fd,0x0015,0x20);
    }
    ulLaneno=ulLaneno&3;
    AFE77xx_RegRead(fd,0x172+(ulLaneno>>1), &ulRegValue);
    ulRegValue=ulRegValue&(0xf<<(4*(1-(ulLaneno&1))));
    ulRegValue=ulRegValue+(maskValue<<(4*(ulLaneno&1)));
    AFE77xx_RegWrite(fd,0x172+(ulLaneno>>1),ulRegValue);
    AFE77xx_RegWrite(fd,0x0015,0x00);
    return 0;
}

```

```

U32 maskJesdRxDmaSerdesErrors(int fd, U32 jesdNo,U32
miscErrorsMaskValue,U32 serdesErrorsMaskValue){
    /*
        miscErrorBits: The bits made 1 will be masked and not
reflect on pin.
        miscErrorBits[0]="JESD sysref error"
        miscErrorBits[1]="JESD shorttest alarm"
        miscErrorBits[2]="NA"
        miscErrorBits[3]="SERDES PLL loss of lock"
    */
    /* for serdesErrorsMaskValue: The bits made 1 will be masked
and not reflect on pin.
        bit7 = Lane 3 JESD frame_sync_err
        bit6 = Lane 2 JESD frame_sync_err
        bit5 = Lane 1 JESD frame_sync_err
        bit4 = Lane 0 JESD frame_sync_err
        bit3 = Lane 3 LOS indicator
        bit2 = Lane 2 LOS indicator
        bit1 = Lane 1 LOS indicator
        bit0 = Lane 0 LOS indicator"
    */

```

```

int ulRegValue;

if (jesdNo==0){
    AFE77xx_RegWrite(fd,0x0015,0x02);
}
else{
    AFE77xx_RegWrite(fd,0x0015,0x20);
}
AFE77xx_RegWrite(fd,0x170,miscErrorsMaskValue);
AFE77xx_RegWrite(fd,0x171,serdesErrorsMaskValue);
AFE77xx_RegWrite(fd,0x0015,0x00);
return 0;
}

U32 maskJesdTxFifoErrors(int fd, U32 jesdNo,U32 maskValue){
    /* If the corresponding bit is 1, the alarm will not go out
    onto pin.
        bit 0 is for lane 0 errors,
        bit 1 for lane 1 errors
        bit 2 for lane 2 errors
        bit 3 for lane 3 errors */
    U32 ulLaneno;
    if (jesdNo==0){
        AFE77xx_RegWrite(fd,0x0015,0x01);
    }
    else{
        AFE77xx_RegWrite(fd,0x0015,0x10);
    }
    AFE77xx_RegWrite(fd,0x00b2,maskValue);
    AFE77xx_RegWrite(fd,0x0015,0x00);
}

U32 maskAllJesdAlarms(int fd, U32 alarmNo, U32 maskValue){
    /* This masks all the JESD alarms
        alarmNo=0 is pin C17.(INT1) alarmNo=1 is pin
E17(INT0).
        maskValue=1 will not get the alarm out onto pins*/
    U32 address=0;
    U32 ulRegValue=0;
    maskValue=!maskValue;
    if (alarmNo==0){
        address=0x118;
    }
    else{
        address=0x11c;
    }
    AFE77xx_RegWrite(fd,0x0015,0x80);
    AFE77xx_RegRead(fd,address,&ulRegValue);
    AFE77xx_RegWrite(fd,address,(ulRegValue&0xfe)+maskValue);
    AFE77xx_RegWrite(fd,0x0015,0x00);
}

```



```

U32 getSerdesRxLaneEyeMarginValue(int fd, int laneNo){
    U32 regValue=0;
    regValue=SerdesLaneRead(fd, laneNo, 0x8030);
    regValue=regValue&0xfff;
    printf("Eye Margin Value read out is on lane %d is %
d", laneNo, regValue);
}

U32 enableSerdesRxPrbsCheck(int fd, int laneNo, int prbsMode, int
enable){
    /*
        laneNo is from 0-7 for SRX1-SRX8.
        prbsMode is 0 for PRBS9, 1 for PRBS15, 2 for PRBS23
and 3 for PRBS31.
        enable = 1 will enable the check, 0 will disable the
check.
    */
    U32 readValue=0, writeValue=0;
    readValue=SerdesLaneRead(fd, laneNo, 0x8042);
    writeValue=(readValue&0xffd1)+((prbsMode&3)<<2)+((enable&1)
<<1)+(1<<5);
    SerdesLaneWrite(fd, laneNo, 0x8042, writeValue);
    writeValue=(readValue&0xffd1)+((prbsMode&3)<<2)+((enable&1)
<<1)+(0<<5);
    SerdesLaneWrite(fd, laneNo, 0x8042, writeValue);
}

U32 getSerdesRxPrbsError(int fd, int laneNo){
    U32 errorRegValue=0;
    errorRegValue=SerdesLaneRead(fd, laneNo, 0x804c);
    errorRegValue=errorRegValue<<16;
    errorRegValue=errorRegValue+SerdesLaneRead(fd, laneNo, 0x804d)
;
    errorRegValue=errorRegValue/3;
    printf("Number of Errors seen in lane %d are %
d", laneNo, errorRegValue);
}

U32 sendSerdesTxPrbs(int fd, int laneNo, int prbsMode, int
enable){
    /*
        laneNo is from 0-7 for STX1-STX8.
        prbsMode is 0 for PRBS9, 1 for PRBS15, 2 for PRBS23
and 3 for PRBS31.
        enable = 1 will enable send PRBS, 0 will send Data.
    */
    U32 readValue=0, writeValue=0;
    readValue=SerdesLaneRead(fd, laneNo, 0x80a0);
    writeValue=(readValue&0x14ff)+((prbsMode&3)<<8)+((enable&1)
<<11)+((enable&1)<<13)+((enable&1)<<14)+((enable&1)<<15);
    SerdesLaneWrite(fd, laneNo, 0x80a0, writeValue);
}

```

```

U32 adcJesdRamp(int fd, U32 RxFb, U32 en, U32 step){
    /* function to enable or disable the ramp from JESD of ADC
    RxFb - 0 - selects Rx
    RxFb - 1 - selects FB
    en - 0 - disable ramp
    en - 1 - enable ramp
    step - increment value - minimum value of 1.
    */
    AFE77xx_RegWrite(fd, 0x0015, 0x11);
    AFE77xx_RegWrite(fd, 0x0045, (0x02<<(RxFb*3))*en);
    AFE77xx_RegWrite(fd, 0x0050, ((step-1)<<(RxFb*4)));
    AFE77xx_RegWrite(fd, 0x0015, 0x00);
    return 0;
}

int setAdcJesdRampPattern(int fd,int chNo,int enable){
    /* chNo is 0 for RX A, RXB and FB AB channels.
    chNo is 1 for RX C, RXD and FB CD channels.
    */
    if (chNo==0){
        AFE77xx_RegWrite(fd,0x0015,0x01);
    }
    else{
        AFE77xx_RegWrite(fd,0x0015,0x10);
    }
    if (enable==1){
        AFE77xx_RegWrite(fd,0x0045,0x12);
    }
    else{
        AFE77xx_RegWrite(fd,0x0045,0x00);
    }
    AFE77xx_RegWrite(fd,0x0015,0x00);
}

U32 parse_response(int chipId, int instanceNo){
    U32 REG_CMD = 0x9815;
    U32 response=0;
    while (1){
        response = SerdesRead(chipId,instanceNo,REG_CMD);
        if (response>>12 == 0){
            break;
        }
    }
    int status = (response>>8) & 0xf;
    int data = response & 0xff;
    if (status == 0x3){
        if (data == 0x02){
            printf("Invalid input");
        }
        else if (data == 0x03){
            printf("Phy not ready");
        }
    }
}

```

```

        }
        else if (data == 0x05){
            printf("Eye monitor going on");
        }
        else if (data == 0x06){
            printf("Eye monitor cancelled");
        }
        else if (data == 0x07){
            printf("Eye monitor not started");
        }
        else{
            printf(data);
        }
    }
    return response;
}

void em_start(int chipId,int lane_num, int ber_exp, int mode){
    U32 REG_CMD = 0x9815;
    U32 REG_CMD_DETAIL = 0x9816;
    int instanceNo=lane_num>>2;
    int jesdToSerdesLaneMapping[8]={3,2,0,1,1,0,2,3};
    lane_num=jesdToSerdesLaneMapping[lane_num];
    U32 command = 0x1000 | (lane_num&0x3) | ((ber_exp&0xF)<<4);
    SerdesWrite(chipId,instanceNo,REG_CMD_DETAIL, mode);
    SerdesWrite(chipId,instanceNo,REG_CMD, command);
    int response = parse_response(chipId,instanceNo);
    int status = (response>>8) & 0xf;
    if (status == 0x0){
        printf("Eye monitor starts...");
    }
}

int em_report_progress(int chipId, int instanceNo){
    U32 REG_CMD = 0x9815;
    U32 command = 0x2000;
    SerdesWrite(chipId,instanceNo,REG_CMD, command);
    int response = parse_response(chipId,instanceNo);
    int status = (response>>8) & 0xf;
    int data = response & 0xff;
    if (status == 0x1){
        return data;
    }
    return 0;
}

unsigned int * em_read(int chipId, int instanceNo){
    U32 REG_CMD = 0x9815;
    U32 REG_CMD_DETAIL = 0x9816;
    static unsigned int ber[3135];
    /*95*33*/
    int m=0;

```

```

    for (int phase=-16;phase<17;phase++){
        int pindex = phase+16;
        for (int margin=-47;margin< 48;margin+= 16){
            SerdesWrite(chipId,instanceNo,REG_CMD_DETAIL,
margin&0xffff);
            SerdesWrite(chipId,instanceNo,REG_CMD, (phase&
0xFF) | 0x3000);
            unsigned int response =
parse_response(chipId,instanceNo);
            unsigned int status = (response>>8) & 0xf;
            unsigned int data = response & 0xff;
            if (status == 0x2){
                for (int i=0;i<16;i++){
                    m=margin+i;
                    if (m<48){
                        ber[(47+m)+(pindex*95)] =
SerdesRead(chipId,instanceNo, (0x9f00+i));
                    }
                }
            }
            else{
                for (int i=0;i<16;i++){
                    m=margin+i;
                    if (m<48){
                        ber[(47+m)+(pindex*95)] = 0;
                    }
                }
            }
            printf("%d",ber[(47+m)+(pindex*95)]);
        }
    }
    return ber;
}

void em_cancel(int chipId, int instanceNo){
    U32 REG_CMD = 0x9815;
    SerdesWrite(chipId,instanceNo,REG_CMD, 0x4000);
    int response = parse_response(chipId,instanceNo);
}

void getSerdesEye(int chipId, int laneNo){
    U32 REG_CMD_DETAIL = 0x9816;
    int instanceNo=laneNo>>2;
    unsigned int *ber;
    em_start(chipId, laneNo, 7, 1);
    /*lane0, ber_exp 10^7, mode 1*/
    int old_progress = 0;
    while (1){
        int progress = em_report_progress(chipId,instanceNo);
        if (progress == 0){
            break;
        }
    }
}

```

```

        if (old_progress != progress){
            printf("%d%%" , progress);
        }
        if (progress == 100){
            break;
        }
        old_progress = progress;
    }
    ber=em_read(chipId,instanceNo);
    U32 extent = SerdesRead(chipId,instanceNo,REG_CMD_DETAIL);
    printf("Printing BER Data");
    for(int loop = 0; loop < 3135; loop++){
        printf("%d, ", ber[loop]);
    }
    printf("\n\nEXTENT: %d",extent);
    /*getSerdesEye*/
}

```

```

U32 relinkJesd(fd){
    /* For JESD 204C linkup */
    long long unsigned alarms=0;
    U32 temp=0;
    U32 readVal=0;
    U32 readSysrefFlags[9]={0,0,0,0,0,0,0,0,0};
    U8 tddPinCtrl[12];
    const char *flagDesc[9];
    flagDesc[0]="Sysref to RX AB";
    flagDesc[1]="Sysref to RX CD";
    flagDesc[2]="Sysref to FB A";
    flagDesc[3]="Sysref to FB D";
    flagDesc[4]="Sysref to TX AB";
    flagDesc[5]="Sysref to TX CD";
    flagDesc[6]="Digital Clock";
    flagDesc[7]="Sysref to Digital";
    flagDesc[8]="Sysref to Analog";

    //Preserve "Pin control on TDD" settings
    AFE77xx_RegWrite(fd, 0x14,0x04);
    AFE77xx_RegRead(fd, 0x124,&tddPinCtrl[0]);
    AFE77xx_RegRead(fd, 0x126,&tddPinCtrl[1]);
    AFE77xx_RegRead(fd, 0x128,&tddPinCtrl[2]);
    AFE77xx_RegRead(fd, 0x12A,&tddPinCtrl[3]);
    AFE77xx_RegRead(fd, 0x12C,&tddPinCtrl[4]);
    AFE77xx_RegRead(fd, 0x12E,&tddPinCtrl[5]);
    AFE77xx_RegRead(fd, 0x125,&tddPinCtrl[6]);
    AFE77xx_RegRead(fd, 0x127,&tddPinCtrl[7]);
    AFE77xx_RegRead(fd, 0x129,&tddPinCtrl[8]);
    AFE77xx_RegRead(fd, 0x12B,&tddPinCtrl[9]);
    AFE77xx_RegRead(fd, 0x12D,&tddPinCtrl[10]);
    AFE77xx_RegRead(fd, 0x12F,&tddPinCtrl[11]);
    AFE77xx_RegWrite(fd, 0x14,0x00);
}

```

```

//START: Overriding TDD Pins internally
AFE77xx_RegWrite(fd,0x0014,0x04);
AFE77xx_RegWrite(fd,0x0124,0x01);
AFE77xx_RegWrite(fd,0x0126,0x01);
AFE77xx_RegWrite(fd,0x0128,0x01);
AFE77xx_RegWrite(fd,0x012a,0x01);
AFE77xx_RegWrite(fd,0x012c,0x01);
AFE77xx_RegWrite(fd,0x012e,0x01);
AFE77xx_RegWrite(fd,0x0125,0x00);
AFE77xx_RegWrite(fd,0x0127,0x01);
AFE77xx_RegWrite(fd,0x0129,0x00);
AFE77xx_RegWrite(fd,0x012b,0x00);
AFE77xx_RegWrite(fd,0x012d,0x01);
AFE77xx_RegWrite(fd,0x012f,0x00);
AFE77xx_RegWrite(fd,0x0014,0x00);
//END: Done overriding TDD Pins internally

AFE77xx_RegWrite(fd,0x0013,0x20);
AFE77xx_RegWrite(fd,0x01a3,0x00);
AFE77xx_RegWrite(fd,0x01a2,0x0f);
AFE77xx_RegWrite(fd,0x01a1,0x00);
AFE77xx_RegWrite(fd,0x01a0,0x00);
AFE77xx_RegWrite(fd,0x01a3,0x11);
AFE77xx_RegWrite(fd,0x01a2,0x0f);
AFE77xx_RegWrite(fd,0x01a1,0x00);
AFE77xx_RegWrite(fd,0x01a0,0x00);
AFE77xx_RegWrite(fd,0x0013,0x00);
AFE77xx_RegWrite(fd,0x0018,0x10);
AFE77xx_RegWrite(fd,0x00bc,0x00);
AFE77xx_RegWrite(fd,0x00bd,0x00);
AFE77xx_RegWrite(fd,0x0024,0x00);
AFE77xx_RegWrite(fd,0x0030,0x01);
AFE77xx_RegWrite(fd,0x0018,0x00);
AFE77xx_RegWrite(fd,0x0010,0x04);
AFE77xx_RegWrite(fd,0x0a05,0x00);
AFE77xx_RegWrite(fd,0x0010,0x08);
AFE77xx_RegWrite(fd,0x0a05,0x00);
AFE77xx_RegWrite(fd,0x0010,0x00);
AFE77xx_RegWrite(fd,0x0013,0x10);
AFE77xx_RegWrite(fd,0x00a3,0x00);
AFE77xx_RegWrite(fd,0x00a2,0x00);
AFE77xx_RegWrite(fd,0x00a1,0xff);
AFE77xx_RegWrite(fd,0x00a0,0x06);
AFE77xx_RegWrite(fd,0x0193,0x35);
AFE77xx_RegWrite(fd,0x0013,0x00);
sysUsDelay(20000);

//Start of Link up Sequence
// STEP: jesdLinkUpInit/step0

//START: Clearing JESD TX Init States
AFE77xx_RegWrite(fd,0x0015,0x01);

```

```
AFE77xx_RegWrite(fd,0x0015,0x11);
AFE77xx_RegWrite(fd,0x0045,0xe0);
AFE77xx_RegWrite(fd,0x0045,0xe0);
AFE77xx_RegWrite(fd,0x0045,0x60);
AFE77xx_RegWrite(fd,0x0045,0x20);
AFE77xx_RegWrite(fd,0x0077,0x00);
//END: Done clearing JESD TX Init States
```

```
//START: Clearing JESD RX Init States
AFE77xx_RegWrite(fd,0x0015,0x10);
AFE77xx_RegWrite(fd,0x0015,0x00);
AFE77xx_RegWrite(fd,0x0015,0x02);
AFE77xx_RegWrite(fd,0x0015,0x22);
AFE77xx_RegWrite(fd,0x0020,0x01);
AFE77xx_RegWrite(fd,0x0020,0x00);
AFE77xx_RegWrite(fd,0x0134,0x00);
//END: Done clearing JESD RX Init States
```

```
AFE77xx_RegWrite(fd,0x0015,0x20);
AFE77xx_RegWrite(fd,0x0015,0x00);
// STEP: jesdLinkUpInit/step1
```

```
//START: Clearing Sysref Flags
AFE77xx_RegWrite(fd,0x0015,0x01);
AFE77xx_RegWrite(fd,0x0015,0x01);
AFE77xx_RegWrite(fd,0x00a8,0x51);
AFE77xx_RegWrite(fd,0x00a8,0x53);
AFE77xx_RegWrite(fd,0x00a8,0x57);
AFE77xx_RegWrite(fd,0x00a8,0x5f);
AFE77xx_RegWrite(fd,0x00a8,0x5e);
AFE77xx_RegWrite(fd,0x00a8,0x5c);
AFE77xx_RegWrite(fd,0x00a8,0x58);
AFE77xx_RegWrite(fd,0x00a8,0x50);
AFE77xx_RegWrite(fd,0x0015,0x00);
AFE77xx_RegWrite(fd,0x0015,0x00);
AFE77xx_RegWrite(fd,0x0011,0xf0);
AFE77xx_RegWrite(fd,0x053f,0xff);
AFE77xx_RegWrite(fd,0x053e,0xff);
AFE77xx_RegWrite(fd,0x053f,0x00);
AFE77xx_RegWrite(fd,0x053e,0x00);
AFE77xx_RegWrite(fd,0x0011,0x00);
AFE77xx_RegWrite(fd,0x0011,0x0f);
AFE77xx_RegWrite(fd,0x053f,0xff);
AFE77xx_RegWrite(fd,0x053e,0xff);
AFE77xx_RegWrite(fd,0x053f,0x00);
AFE77xx_RegWrite(fd,0x053e,0x00);
AFE77xx_RegWrite(fd,0x0011,0x00);
AFE77xx_RegWrite(fd,0x0017,0x01);
AFE77xx_RegReadWrite(fd,0x0050,0x00,7,7);
AFE77xx_RegReadWrite(fd,0x0050,0x80,7,7);
AFE77xx_RegReadWrite(fd,0x0050,0x00,7,7);
AFE77xx_RegWrite(fd,0x0017,0x00);
```

```

AFE77xx_RegWrite(fd,0x0017,0x10);
AFE77xx_RegWrite(fd,0x0030,0x00);
AFE77xx_RegWrite(fd,0x0030,0x08);
AFE77xx_RegWrite(fd,0x0030,0x00);
AFE77xx_RegWrite(fd,0x0017,0x20);
AFE77xx_RegWrite(fd,0x0030,0x00);
AFE77xx_RegWrite(fd,0x0030,0x08);
AFE77xx_RegWrite(fd,0x0030,0x00);
AFE77xx_RegWrite(fd,0x0017,0x00);
AFE77xx_RegWrite(fd,0x0017,0x40);
AFE77xx_RegWrite(fd,0x002b,0x00);
AFE77xx_RegWrite(fd,0x002b,0x08);
AFE77xx_RegWrite(fd,0x002b,0x00);
AFE77xx_RegWrite(fd,0x0017,0x80);
AFE77xx_RegWrite(fd,0x002b,0x00);
AFE77xx_RegWrite(fd,0x002b,0x08);
AFE77xx_RegWrite(fd,0x002b,0x00);
AFE77xx_RegWrite(fd,0x0017,0x40);
AFE77xx_RegWrite(fd,0x002e,0x00);
AFE77xx_RegWrite(fd,0x002e,0x08);
AFE77xx_RegWrite(fd,0x002e,0x00);
AFE77xx_RegWrite(fd,0x0017,0x80);
AFE77xx_RegWrite(fd,0x002e,0x00);
AFE77xx_RegWrite(fd,0x002e,0x08);
AFE77xx_RegWrite(fd,0x002e,0x00);
AFE77xx_RegWrite(fd,0x0118,0x07);
AFE77xx_RegWrite(fd,0x0119,0x7f);
AFE77xx_RegWrite(fd,0x0117,0x10);
AFE77xx_RegWrite(fd,0x0116,0x00);
AFE77xx_RegWrite(fd,0x0017,0x00);
//END: Done clearing Sysref Flags

// STEP: jesdLinkUpInit/step2

//START: Sending Sysref to device from Pin/SPI

//START: Requesting/releasing SPI Access to PLL Pages
AFE77xx_RegWrite(fd,0x0015,0x80);
#ifdef TIENV
AFE77xx_RegWrite(fd,0x01d4,0x00);
AFE77xx_RegWrite(fd,0x0374,0x01);
#else
AFE77xx_RegWrite(fd,0x01d4,0x01);
AFE77xx_RegWrite(fd,0x0374,0x00);
#endif
sysUsDelay(1000);
AFE77xx_RegWrite(fd,0x0015,0x00);
//END: Done requesting/releasing SPI Access to PLL Pages

AFE77xx_RegWrite(fd,0x0014,0x01);
AFE77xx_RegWrite(fd,0x002c,0x00);
sysUsDelay(1000);

```



```
AFE77xx_RegWrite (fd, 0x0014, 0x00);
AFE77xx_RegWrite (fd, 0x0014, 0x04);
AFE77xx_RegWrite (fd, 0x0717, 0x00);
AFE77xx_RegWrite (fd, 0x0716, 0x00);
AFE77xx_RegWrite (fd, 0x0715, 0x00);
AFE77xx_RegWrite (fd, 0x0714, 0x00);
AFE77xx_RegWrite (fd, 0x0717, 0x00);
AFE77xx_RegWrite (fd, 0x0716, 0x00);
AFE77xx_RegWrite (fd, 0x0715, 0x01);
AFE77xx_RegWrite (fd, 0x0714, 0x01);
AFE77xx_RegWrite (fd, 0x0717, 0x00);
AFE77xx_RegWrite (fd, 0x0716, 0x00);
AFE77xx_RegWrite (fd, 0x0715, 0x00);
AFE77xx_RegWrite (fd, 0x0714, 0x00);
AFE77xx_RegWrite (fd, 0x0014, 0x00);
AFE77xx_RegWrite (fd, 0x0014, 0x08);
AFE77xx_RegReadWrite (fd, 0x004c, 0xfc, 0, 0);
AFE77xx_RegWrite (fd, 0x0051, 0x86);
AFE77xx_RegWrite (fd, 0x0051, 0x8e);
AFE77xx_RegWrite (fd, 0x0014, 0x10);
AFE77xx_RegReadWrite (fd, 0x004e, 0x30, 2, 2);
AFE77xx_RegWrite (fd, 0x0014, 0x08);
AFE77xx_RegReadWrite (fd, 0x004c, 0xfc, 0, 0);
AFE77xx_RegReadWrite (fd, 0x004c, 0xfd, 0, 0);
AFE77xx_RegWrite (fd, 0x0014, 0x10);
AFE77xx_RegReadWrite (fd, 0x004c, 0xfc, 0, 0);
AFE77xx_RegWrite (fd, 0x0051, 0x84);
AFE77xx_RegWrite (fd, 0x0051, 0x8c);
AFE77xx_RegReadWrite (fd, 0x004c, 0xfc, 0, 0);
AFE77xx_RegReadWrite (fd, 0x004c, 0xfd, 0, 0);
AFE77xx_RegWrite (fd, 0x0014, 0x20);
AFE77xx_RegReadWrite (fd, 0x004c, 0xfc, 0, 0);
AFE77xx_RegWrite (fd, 0x0051, 0x86);
AFE77xx_RegWrite (fd, 0x0051, 0x8e);
AFE77xx_RegWrite (fd, 0x0014, 0x10);
AFE77xx_RegReadWrite (fd, 0x004e, 0x30, 2, 2);
AFE77xx_RegWrite (fd, 0x0014, 0x20);
AFE77xx_RegReadWrite (fd, 0x004c, 0xfc, 0, 0);
AFE77xx_RegReadWrite (fd, 0x004c, 0xfd, 0, 0);
AFE77xx_RegWrite (fd, 0x0014, 0x40);
AFE77xx_RegReadWrite (fd, 0x004c, 0xfc, 0, 0);
AFE77xx_RegWrite (fd, 0x0051, 0x86);
AFE77xx_RegWrite (fd, 0x0051, 0x8e);
AFE77xx_RegWrite (fd, 0x0014, 0x10);
AFE77xx_RegReadWrite (fd, 0x004e, 0x30, 2, 2);
AFE77xx_RegWrite (fd, 0x0014, 0x40);
AFE77xx_RegReadWrite (fd, 0x004c, 0xfc, 0, 0);
AFE77xx_RegReadWrite (fd, 0x004c, 0xfd, 0, 0);
AFE77xx_RegWrite (fd, 0x0014, 0x80);
AFE77xx_RegReadWrite (fd, 0x004c, 0xfc, 0, 0);
AFE77xx_RegWrite (fd, 0x0051, 0x86);
AFE77xx_RegWrite (fd, 0x0051, 0x8e);
```

```

AFE77xx_RegWrite(fd,0x0014,0x10);
AFE77xx_RegReadWrite(fd,0x004e,0x30,2,2);
AFE77xx_RegWrite(fd,0x0014,0x80);
AFE77xx_RegReadWrite(fd,0x004c,0xfc,0,0);
AFE77xx_RegReadWrite(fd,0x004c,0xfd,0,0);

//Give Sysref Here.
sysUsDelay(1000);

AFE77xx_RegWrite(fd,0x0014,0x08);
AFE77xx_RegWrite(fd,0x0051,0x0e);
AFE77xx_RegWrite(fd,0x0051,0x06);
AFE77xx_RegWrite(fd,0x0014,0x10);
AFE77xx_RegWrite(fd,0x0051,0x0c);
AFE77xx_RegWrite(fd,0x0051,0x04);
AFE77xx_RegWrite(fd,0x0014,0x20);
AFE77xx_RegWrite(fd,0x0051,0x0e);
AFE77xx_RegWrite(fd,0x0051,0x06);
AFE77xx_RegWrite(fd,0x0014,0x40);
AFE77xx_RegWrite(fd,0x0051,0x0e);
AFE77xx_RegWrite(fd,0x0051,0x06);
AFE77xx_RegWrite(fd,0x0014,0x80);
AFE77xx_RegWrite(fd,0x0051,0x0e);
AFE77xx_RegWrite(fd,0x0051,0x06);
AFE77xx_RegWrite(fd,0x0014,0x00);
AFE77xx_RegWrite(fd,0x0014,0x01);
AFE77xx_RegWrite(fd,0x002c,0x00);

//START: Requesting/releasing SPI Access to PLL Pages
AFE77xx_RegWrite(fd,0x0014,0x00);
AFE77xx_RegWrite(fd,0x0015,0x80);
AFE77xx_RegWrite(fd,0x01d4,0x00);
AFE77xx_RegWrite(fd,0x0374,0x00);
sysUsDelay(100);
AFE77xx_RegWrite(fd,0x0015,0x00);
//END: Done requesting/releasing SPI Access to PLL Pages

//END: Done sending Sysref to device from Pin/SPI

//START: Staggered TDD Toggle
AFE77xx_RegWrite(fd,0x0014,0x04);
AFE77xx_RegWrite(fd,0x0124,0x01);
AFE77xx_RegWrite(fd,0x0126,0x01);
AFE77xx_RegWrite(fd,0x0128,0x01);
AFE77xx_RegWrite(fd,0x012a,0x01);
AFE77xx_RegWrite(fd,0x012c,0x01);
AFE77xx_RegWrite(fd,0x012e,0x01);
AFE77xx_RegWrite(fd,0x0127,0x00);
AFE77xx_RegWrite(fd,0x012d,0x00);
AFE77xx_RegWrite(fd,0x0127,0x01);
AFE77xx_RegWrite(fd,0x012d,0x01);
AFE77xx_RegWrite(fd,0x0127,0x00);

```

```

AFE77xx_RegWrite (fd, 0x012d, 0x00);
AFE77xx_RegWrite (fd, 0x0125, 0x00);
AFE77xx_RegWrite (fd, 0x012b, 0x00);
AFE77xx_RegWrite (fd, 0x0129, 0x00);
AFE77xx_RegWrite (fd, 0x012f, 0x00);
AFE77xx_RegWrite (fd, 0x0125, 0x01);
AFE77xx_RegWrite (fd, 0x012b, 0x01);
AFE77xx_RegWrite (fd, 0x0129, 0x01);
AFE77xx_RegWrite (fd, 0x012f, 0x01);
AFE77xx_RegWrite (fd, 0x0014, 0x00);
//END: Done Staggered TDD Toggle

// STEP: jesdLinkUpInit/step3

//START: Checking Sysref Flags
AFE77xx_RegWrite (fd, 0x0017, 0x40);
readSysrefFlags [0]= (AFE77xx_RegRead (fd, 0x0049, &temp) &0x3);

AFE77xx_RegWrite (fd, 0x0017, 0x80);
readSysrefFlags [1]= (AFE77xx_RegRead (fd, 0x0049, &temp) &0x3);

AFE77xx_RegWrite (fd, 0x0017, 0x40);
readSysrefFlags [2]= ((AFE77xx_RegRead (fd, 0x0049, &temp) &
0x8)>>3);

AFE77xx_RegWrite (fd, 0x0017, 0x80);
readSysrefFlags [3]= ((AFE77xx_RegRead (fd, 0x0049, &temp) &
0x8)>>3);

AFE77xx_RegWrite (fd, 0x0017, 0x00);
AFE77xx_RegWrite (fd, 0x0017, 0x10);
readSysrefFlags [4]=AFE77xx_RegRead (fd, 0x003c, &temp);

AFE77xx_RegWrite (fd, 0x0017, 0x20);
readSysrefFlags [5]=AFE77xx_RegRead (fd, 0x003c, &temp);

AFE77xx_RegWrite (fd, 0x0017, 0x00);
AFE77xx_RegWrite (fd, 0x0015, 0x01);
AFE77xx_RegWrite (fd, 0x0015, 0x01);
readSysrefFlags [6]= ((AFE77xx_RegRead (fd, 0x00a8, &temp) &
0x10)>>4);

readSysrefFlags [7]= ((AFE77xx_RegRead (fd, 0x00a8, &temp) &
0x20)>>5);

AFE77xx_RegWrite (fd, 0x0015, 0x00);
AFE77xx_RegWrite (fd, 0x0015, 0x00);
AFE77xx_RegWrite (fd, 0x0011, 0x01);
readSysrefFlags [8]=AFE77xx_RegRead (fd, 0x053d, &temp);
readSysrefFlags [8]= (readSysrefFlags [8]
<<8)+AFE77xx_RegRead (fd, 0x053c, &temp);

```

```

    U32 expectedSysrefFlags[9]={3,3,1,1,7,7,1,1,7};
    for(int i=0;i<9;i=i+1){
        printf("%s, read :%d, expected: %d
\n", flagDesc[i], readSysrefFlags[i], expectedSysrefFlags[i]);
    }
    //END: Done checking Sysref Flags

AFE77xx_RegWrite(fd,0x0011,0x00);
// STEP: jesdLinkUpInit/step4

//START: Clearing JESD TX Data and alarms
AFE77xx_RegWrite(fd,0x0015,0x01);
AFE77xx_RegWrite(fd,0x0015,0x11);
AFE77xx_RegWrite(fd,0x00b1,0x0f);
AFE77xx_RegWrite(fd,0x00b1,0x00);
AFE77xx_RegWrite(fd,0x0026,0x0f);
AFE77xx_RegWrite(fd,0x0026,0x00);
AFE77xx_RegWrite(fd,0x0045,0x00);
//END: Done clearing JESD TX Data and alarms

//START: Clearing JESD RX Data and alarms
AFE77xx_RegWrite(fd,0x0015,0x10);
AFE77xx_RegWrite(fd,0x0015,0x00);
AFE77xx_RegWrite(fd,0x0015,0x02);
AFE77xx_RegWrite(fd,0x0015,0x22);
AFE77xx_RegWrite(fd,0x0081,0xff);
AFE77xx_RegWrite(fd,0x0081,0x00);
AFE77xx_RegWrite(fd,0x002d,0xdb);
AFE77xx_RegWrite(fd,0x002d,0x9b);
AFE77xx_RegWrite(fd,0x0190,0x01);
AFE77xx_RegWrite(fd,0x0190,0x00);
AFE77xx_RegWrite(fd,0x0190,0x04);
AFE77xx_RegWrite(fd,0x0190,0x00);
//END: Done clearing JESD RX Data and alarms

//START: Reading the JESD RX states to check if link is
established

//START: Clearing JESD RX Data and alarms
AFE77xx_RegWrite(fd,0x0081,0xff);
AFE77xx_RegWrite(fd,0x0081,0x00);
AFE77xx_RegWrite(fd,0x002d,0xdb);
AFE77xx_RegWrite(fd,0x002d,0x9b);
AFE77xx_RegWrite(fd,0x0190,0x01);
AFE77xx_RegWrite(fd,0x0190,0x00);
AFE77xx_RegWrite(fd,0x0190,0x04);
AFE77xx_RegWrite(fd,0x0190,0x00);
//END: Done clearing JESD RX Data and alarms
sysUsDelay(1000);

//START: Reading the JESD RX states to check if link is
established

```

```

//START: Clearing JESD RX Data and alarms

AFE77xx_RegWrite(fd,0x0081,0xff);
AFE77xx_RegWrite(fd,0x0081,0x00);
AFE77xx_RegWrite(fd,0x002d,0xdb);
AFE77xx_RegWrite(fd,0x002d,0x9b);
AFE77xx_RegWrite(fd,0x0190,0x01);
AFE77xx_RegWrite(fd,0x0190,0x00);
AFE77xx_RegWrite(fd,0x0190,0x04);
AFE77xx_RegWrite(fd,0x0190,0x00);

//END: Done clearing JESD RX Data and alarms
sysUsDelay(1000);

AFE77xx_RegWrite(fd,0x0015,0x20);
AFE77xx_RegWrite(fd,0x0015,0x00);
AFE77xx_RegWrite(fd,0x0015,0x01);
AFE77xx_RegWrite(fd,0x0015,0x11);
AFE77xx_RegWrite(fd,0x0072,0xf0);
AFE77xx_RegWrite(fd,0x0072,0xff);

sysUsDelay(200000);

AFE77xx_RegWrite(fd,0x0072,0xf0);
AFE77xx_RegWrite(fd,0x0015,0x10);
AFE77xx_RegWrite(fd,0x0015,0x00);
// STEP: dacJesdLinkUp/step0

//START: Reading the JESD RX states to check if link is
established

//START: Clearing JESD RX Data and alarms

AFE77xx_RegWrite(fd,0x0015,0x02);
AFE77xx_RegWrite(fd,0x0015,0x02);
AFE77xx_RegWrite(fd,0x0081,0xff);
AFE77xx_RegWrite(fd,0x0081,0x00);
AFE77xx_RegWrite(fd,0x002d,0xdb);
AFE77xx_RegWrite(fd,0x002d,0x9b);
AFE77xx_RegWrite(fd,0x0190,0x01);
AFE77xx_RegWrite(fd,0x0190,0x00);
AFE77xx_RegWrite(fd,0x0190,0x04);
AFE77xx_RegWrite(fd,0x0190,0x00);

//END: Done clearing JESD RX Data and alarms

sysUsDelay(1000);

AFE77xx_RegRead(fd,0x07c,&temp);
if(temp!=0){
    printf("###Device DAC JESD-RX 0 Link Status###\n");
}

```

```

readVal=AFE77xx_RegRead(fd,0x0167,&temp);
readVal=(readVal<<8)+AFE77xx_RegRead(fd,0x0166,&temp);
readVal=(readVal<<8)+AFE77xx_RegRead(fd,0x0165,&temp);
readVal=(readVal<<8)+AFE77xx_RegRead(fd,0x0164,&temp);
readVal=(readVal<<8)+AFE77xx_RegRead(fd,0x0163,&temp);
readVal=(readVal<<8)+AFE77xx_RegRead(fd,0x0162,&temp);
readVal=(readVal<<8)+AFE77xx_RegRead(fd,0x0161,&temp);
readVal=(readVal<<8)+AFE77xx_RegRead(fd,0x0160,&temp);
if(((readVal>>3)&1)!=0){
    readVal=readVal^0x8;
}
printf("Alarms:%llu\n",readVal);

//Read    alarms=0x0;
Address(0x160[31:0],0x164[31:0],)

readVal=((AFE77xx_RegRead(fd,0x0135,&temp)&0x10)>>4);

//Read    comma_align_lock_lane0_monitor_flag=0x1;
Address(0x134[12:12],)

readVal=((AFE77xx_RegRead(fd,0x0135,&temp)&0x20)>>5);

//Read    comma_align_lock_lane1_monitor_flag=0x1;
Address(0x134[13:13],)

AFE77xx_RegRead(fd,0x012a,&temp);
printf("CS state: %d\n",temp);

//Read    jesd_cs_state_tx0=0xa;
Address(0x128[23:16],)

AFE77xx_RegRead(fd,0x012e,&temp);
printf("Buf state: %d\n",temp);

//Read    jesd_buf_state_tx0=0xf;
Address(0x12c[23:16],)
}
//END: Done reading the JESD RX states to check if link is
established

//START: Reading the JESD RX states to check if link is
established

//START: Clearing JESD RX Data and alarms

AFE77xx_RegWrite(fd,0x0015,0x00);
AFE77xx_RegWrite(fd,0x0015,0x20);
AFE77xx_RegWrite(fd,0x0081,0xff);
AFE77xx_RegWrite(fd,0x0081,0x00);
AFE77xx_RegWrite(fd,0x002d,0xdb);
AFE77xx_RegWrite(fd,0x002d,0x9b);

```

```

AFE77xx_RegWrite(fd,0x0190,0x01);
AFE77xx_RegWrite(fd,0x0190,0x00);
AFE77xx_RegWrite(fd,0x0190,0x04);
AFE77xx_RegWrite(fd,0x0190,0x00);

//END: Done clearing JESD RX Data and alarms
sysUsDelay(1000);

AFE77xx_RegRead(fd,0x007c,&temp);
if(temp!=0){
    printf("###Device DAC JESD-RX 1 Link Status###\n");
    readVal=AFE77xx_RegRead(fd,0x0167,&temp);
    readVal=(readVal<<8)+AFE77xx_RegRead(fd,0x0166,&temp);
    readVal=(readVal<<8)+AFE77xx_RegRead(fd,0x0165,&temp);
    readVal=(readVal<<8)+AFE77xx_RegRead(fd,0x0164,&temp);
    readVal=(readVal<<8)+AFE77xx_RegRead(fd,0x0163,&temp);
    readVal=(readVal<<8)+AFE77xx_RegRead(fd,0x0162,&temp);
    readVal=(readVal<<8)+AFE77xx_RegRead(fd,0x0161,&temp);
    readVal=(readVal<<8)+AFE77xx_RegRead(fd,0x0160,&temp);
    if(((readVal>>3)&1)!=0){
        readVal=readVal^0x8;
    }
    printf("Alarms:%llu\n",readVal);
    //Read    alarms=0x0;
    Address(0x160[31:0],0x164[31:0],)

    readVal=((AFE77xx_RegRead(fd,0x0135,&temp)&0x10)>>4);

    //Read    comma_align_lock_lane0_monitor_flag=0x1;
    Address(0x134[12:12],)

    readVal=((AFE77xx_RegRead(fd,0x0135,&temp)&0x20)>>5);

    //Read    comma_align_lock_lane1_monitor_flag=0x1;
    Address(0x134[13:13],)

    AFE77xx_RegRead(fd,0x012a,&temp);
    printf("CS state: %d\n",temp);
    //Read    jesd_cs_state_tx0=0xa;
    Address(0x128[23:16],)

    AFE77xx_RegRead(fd,0x012e,&temp);
    printf("Buf state: %d\n",temp);
    //Read    jesd_buf_state_tx0=0xf;
    Address(0x12c[23:16],)
}
//END: Done reading the JESD RX states to check if link is
established

AFE77xx_RegWrite(fd,0x0015,0x20);
AFE77xx_RegWrite(fd,0x0015,0x00);
// STEP: dacJesdLinkUp/step1

```

```
//START: Writing Post Link up SERDES writes
AFE77xx_RegWrite(fd,0x0015,0x04);
AFE77xx_RegWrite(fd,0x0015,0x04);
AFE77xx_RegWrite(fd,0x41fb,0x02);
AFE77xx_RegWrite(fd,0x41fa,0x6e);
AFE77xx_RegWrite(fd,0x43fb,0x02);
AFE77xx_RegWrite(fd,0x43fa,0x6e);
AFE77xx_RegWrite(fd,0x45fb,0x02);
AFE77xx_RegWrite(fd,0x45fa,0x6e);
AFE77xx_RegWrite(fd,0x47fb,0x02);
AFE77xx_RegWrite(fd,0x47fa,0x6e);
//END: Done writing Post Link up SERDES writes
```

```
//START: Writing Post Link up SERDES writes
AFE77xx_RegWrite(fd,0x0015,0x00);
AFE77xx_RegWrite(fd,0x0015,0x40);
AFE77xx_RegWrite(fd,0x41fb,0x02);
AFE77xx_RegWrite(fd,0x41fa,0x6e);
AFE77xx_RegWrite(fd,0x43fb,0x02);
AFE77xx_RegWrite(fd,0x43fa,0x6e);
AFE77xx_RegWrite(fd,0x45fb,0x02);
AFE77xx_RegWrite(fd,0x45fa,0x6e);
AFE77xx_RegWrite(fd,0x47fb,0x02);
AFE77xx_RegWrite(fd,0x47fa,0x6e);
//END: Done writing Post Link up SERDES writes
```

```
AFE77xx_RegWrite(fd,0x0015,0x40);
AFE77xx_RegWrite(fd,0x0015,0x00);
AFE77xx_RegWrite(fd,0x0013,0x10);
AFE77xx_RegWrite(fd,0x00a3,0x00);
AFE77xx_RegWrite(fd,0x00a2,0x00);
AFE77xx_RegWrite(fd,0x00a1,0xff);
AFE77xx_RegWrite(fd,0x00a0,0x07);
AFE77xx_RegWrite(fd,0x0193,0x35);
sysUsDelay(20000);
```

```
//START: Overriding TDD Pins internally
AFE77xx_RegWrite(fd,0x0013,0x00);
AFE77xx_RegWrite(fd,0x0014,0x04);
AFE77xx_RegWrite(fd,0x0124,0x01);
AFE77xx_RegWrite(fd,0x0126,0x01);
AFE77xx_RegWrite(fd,0x0128,0x01);
AFE77xx_RegWrite(fd,0x012a,0x01);
AFE77xx_RegWrite(fd,0x012c,0x01);
AFE77xx_RegWrite(fd,0x012e,0x01);
AFE77xx_RegWrite(fd,0x0125,0x00);
AFE77xx_RegWrite(fd,0x0127,0x01);
AFE77xx_RegWrite(fd,0x0129,0x00);
AFE77xx_RegWrite(fd,0x012b,0x00);
AFE77xx_RegWrite(fd,0x012d,0x01);
AFE77xx_RegWrite(fd,0x012f,0x00);
```



```
//END: Done overriding TDD Pins internally
```

```
AFE77xx_RegWrite(fd,0x0014,0x00);  
AFE77xx_RegWrite(fd,0x0010,0x04);  
AFE77xx_RegWrite(fd,0x0a05,0x01);  
AFE77xx_RegWrite(fd,0x0010,0x08);  
AFE77xx_RegWrite(fd,0x0a05,0x01);  
AFE77xx_RegWrite(fd,0x0010,0x00);  
AFE77xx_RegWrite(fd,0x0013,0x20);  
AFE77xx_RegWrite(fd,0x01a3,0x00);  
AFE77xx_RegWrite(fd,0x01a2,0x0f);  
AFE77xx_RegWrite(fd,0x01a1,0x00);  
AFE77xx_RegWrite(fd,0x01a0,0x00);  
AFE77xx_RegWrite(fd,0x01a3,0x17);  
AFE77xx_RegWrite(fd,0x01a2,0x0f);  
AFE77xx_RegWrite(fd,0x01a1,0x00);  
AFE77xx_RegWrite(fd,0x01a0,0x00);  
AFE77xx_RegWrite(fd,0x01a3,0x00);  
AFE77xx_RegWrite(fd,0x01a2,0x0f);  
AFE77xx_RegWrite(fd,0x01a1,0x00);  
AFE77xx_RegWrite(fd,0x01a0,0x00);  
AFE77xx_RegWrite(fd,0x01a3,0x00);  
AFE77xx_RegWrite(fd,0x01a2,0x0f);  
AFE77xx_RegWrite(fd,0x01a1,0x00);  
AFE77xx_RegWrite(fd,0x01a0,0x00);  
AFE77xx_RegWrite(fd,0x01a3,0x12);  
AFE77xx_RegWrite(fd,0x01a2,0x0f);  
AFE77xx_RegWrite(fd,0x01a1,0x00);  
AFE77xx_RegWrite(fd,0x01a0,0x00);
```

```
//START: Disabling TDD overrides
```

```
AFE77xx_RegWrite(fd,0x0013,0x00);  
//"Releasing Pin control on TDD/"  
AFE77xx_RegWrite(fd, 0x14,0x04);  
AFE77xx_RegWrite(fd, 0x124,tddPinCtrl[0]);  
AFE77xx_RegWrite(fd, 0x126,tddPinCtrl[1]);  
AFE77xx_RegWrite(fd, 0x128,tddPinCtrl[2]);  
AFE77xx_RegWrite(fd, 0x12A,tddPinCtrl[3]);  
AFE77xx_RegWrite(fd, 0x12C,tddPinCtrl[4]);  
AFE77xx_RegWrite(fd, 0x12E,tddPinCtrl[5]);  
AFE77xx_RegWrite(fd, 0x125,tddPinCtrl[6]);  
AFE77xx_RegWrite(fd, 0x127,tddPinCtrl[7]);  
AFE77xx_RegWrite(fd, 0x129,tddPinCtrl[8]);  
AFE77xx_RegWrite(fd, 0x12B,tddPinCtrl[9]);  
AFE77xx_RegWrite(fd, 0x12D,tddPinCtrl[10]);  
AFE77xx_RegWrite(fd, 0x12F,tddPinCtrl[11]);  
//END: Done disabling TDD overrides
```

```
//START: Clearing All Alarms
```

```
AFE77xx_RegWrite(fd,0x001a,0xff);  
AFE77xx_RegWrite(fd,0x001b,0x01);  
AFE77xx_RegWrite(fd,0x001a,0x00);  
AFE77xx_RegWrite(fd,0x001b,0x00);  
AFE77xx_RegWrite(fd,0x0014,0x00);  
AFE77xx_RegWrite(fd,0x0013,0x20);  
AFE77xx_RegWrite(fd,0x02e5,0x20);
```

```

AFE77xx_RegWrite(fd,0x02e5,0x00);

//START: Requesting/releasing SPI Access to PLL Pages
AFE77xx_RegWrite(fd,0x0013,0x00);
AFE77xx_RegWrite(fd,0x0015,0x80);
#ifdef TIENV
AFE77xx_RegWrite(fd,0x01d4,0x00);
AFE77xx_RegWrite(fd,0x0374,0x01);
#else
AFE77xx_RegWrite(fd,0x01d4,0x01);
AFE77xx_RegWrite(fd,0x0374,0x00);
#endif
sysUsDelay(1000);
//END: Done requesting/releasing SPI Access to PLL Pages

AFE77xx_RegWrite(fd,0x0015,0x00);
AFE77xx_RegWrite(fd,0x0014,0xf8);
AFE77xx_RegWrite(fd,0x0066,0x08);
AFE77xx_RegWrite(fd,0x0066,0x00);

//START: Requesting/releasing SPI Access to PLL Pages
AFE77xx_RegWrite(fd,0x0014,0x00);
AFE77xx_RegWrite(fd,0x0015,0x80);
AFE77xx_RegWrite(fd,0x01d4,0x00);
AFE77xx_RegWrite(fd,0x0374,0x00);
sysUsDelay(100);
//END: Done requesting/releasing SPI Access to PLL Pages

AFE77xx_RegWrite(fd,0x064a,0x0f);
AFE77xx_RegWrite(fd,0x064a,0x00);

//END: Done clearing All Alarms

AFE77xx_RegWrite(fd,0x0015,0x00);
}

/* RX related functions */
U32 RXDsaAttn(int fd, U32 ulChan, U32 ulRXDsa){
    U32 ulPageRegValue = 0;
    U32 ulRegValue = 0;
    U32 ulDsaReg = 0;

    ulPageRegValue = 0x10 << (ulChan>>1);
    ulDsaReg=0x50+(0x60*(ulChan&1));

    AFE77xx_RegWrite(fd,0x0016,ulPageRegValue);
    AFE77xx_RegWrite(fd,ulDsaReg,ulRXDsa);
    AFE77xx_RegWrite(fd,0x0016,0x00);
    return 0;
}

U32 RXDsaSwapAttn(int fd, U32 ulChan, U32 ulRXDsa){

```

```

U32 ulPageRegValue = 0;
U32 ulRegValue = 0;
U32 ulDsaReg = 0;

ulPageRegValue = 0x10 << (ulChan>>1);
ulDsaReg=0x5C+(0x60*(ulChan&1));

AFE77xx_RegWrite(fd,0x0016,ulPageRegValue);
AFE77xx_RegWrite(fd,ulDsaReg,ulRXDsa);
AFE77xx_RegWrite(fd,0x0016,0x00);
return 0;
}

U32 EnableInternalAgc(int fd, U32 ulChan, U32 ulEnable){
/* ulChan value is 0 for AB and 1 for CD */

/* To check if gain control is AGC or SPI */
AFE77xx_RegWrite(fd,0x0016,(0x10<<ulChan));
if(ulEnable == 1){
AFE77xx_RegWrite(fd,0x0040,0x02);
}
else{
AFE77xx_RegWrite(fd,0x0040,0x03);
}
AFE77xx_RegWrite(fd,0x0016,0);
/* To Enable or Disable AGC */
AFE77xx_RegWrite(fd,0x0010,(0x04<<ulChan));
if(ulEnable == 1){
AFE77xx_RegWrite(fd,0x0264,0x01);
AFE77xx_RegWrite(fd,0x0664,0x01);
}
else{
AFE77xx_RegWrite(fd,0x0264,0x00);
AFE77xx_RegWrite(fd,0x0664,0x00);
}
AFE77xx_RegWrite(fd,0x0010,0);
return 0;
}

U32 SetDefAttenuation(int fd, U32 ulChan, U32 ulDefGain){
U32 ulRegOffset=(0x0400*(ulChan&1));
AFE77xx_RegWrite(fd,0x0010,(0x4<<(ulChan>>1)));
AFE77xx_RegWrite(fd,0x0268 + ulRegOffset,ulDefGain);
AFE77xx_RegWrite(fd,0x0010,0);
}

U32 SetMaxAttenuation(int fd, U32 ulChan, U32 ulMaxAttn){
U32 ulRegOffset=(0x0400*(ulChan&1));
AFE77xx_RegWrite(fd,0x0010,(0x4<<(ulChan>>1)));
AFE77xx_RegWrite(fd,0x026c + ulRegOffset,ulMaxAttn);
AFE77xx_RegWrite(fd,0x0010,0);
}

```

```

U32 SetMinAttenuation(int fd, U32 ulChan, U32 ulMinAttn){
    U32 ulRegOffset=(0x0400*(ulChan&1));
    AFE77xx_RegWrite(fd,0x0010,(0x4<<(ulChan>>1)));
    AFE77xx_RegWrite(fd,0x026d + ulRegOffset,ulMinAttn);
    AFE77xx_RegWrite(fd,0x0010,0);
}

U32 SetMaxMinAttenuation(int fd, U32 ulChan, U32 ulMinAttn, U32
ulMaxAttn){
    U32 ulRegOffset=(0x0400*(ulChan&1));
    AFE77xx_RegWrite(fd,0x0010,(0x4<<(ulChan>>1)));
    AFE77xx_RegWrite(fd,0x026c + ulRegOffset,ulMaxAttn);
    AFE77xx_RegWrite(fd,0x026d + ulRegOffset,ulMinAttn);
    AFE77xx_RegWrite(fd,0x0010,0);
}

U32 BigStepAttackConfig(int fd, U32 ulChan, U32 ulEnable, U32
ulAttackStepSize, double ulAttackThreshold, U64 ulWindowLen, U32
ulHitCount){
    /* When the internal AGC is enabled, this function is used
for big step attack.
        ulChan is from 0 to 3, for each RX Channel.
        ulAttackStepSize is the step size in dBFs. Should be
multiple of 1.
        ulAttackThreshold is the attack threshold in dBFs.
        ulWindowLen is the window length. For value of N, the
window length will be N cycles of Fs/8 clocks.
        ulHitCount is the number of times the signal should be
above threshold in the window for attack to happen.
    */
    U32 ulRegValue=0;
    U32 ulRegOffset=(0x0400*(ulChan&1));
    U32 ulAttackThresholdWord=ceil(4096*pow(10.0,
ulAttackThreshold/20.0));

    AFE77xx_RegWrite(fd,0x0010,(0x4<<(ulChan>>1)));
    AFE77xx_RegWrite(fd,0x0208 + ulRegOffset,ulAttackStepSize);
    AFE77xx_RegWrite(fd,0x0235 +
ulRegOffset,ulAttackThresholdWord>>8);
    AFE77xx_RegWrite(fd,0x0234 +
ulRegOffset,ulAttackThresholdWord&0xff);
    AFE77xx_RegWrite(fd,0x0237 +
ulRegOffset,ulAttackThresholdWord>>8);
    AFE77xx_RegWrite(fd,0x0236 +
ulRegOffset,ulAttackThresholdWord&0xff);
    AFE77xx_RegWrite(fd,0x0216 + ulRegOffset,ulWindowLen>>16);
    AFE77xx_RegWrite(fd,0x0215 + ulRegOffset,(ulWindowLen>>8)&
0xff);
    AFE77xx_RegWrite(fd,0x0214 + ulRegOffset,ulWindowLen&0xff);
    AFE77xx_RegWrite(fd,0x024a + ulRegOffset,ulHitCount>>16);
    AFE77xx_RegWrite(fd,0x0249 + ulRegOffset,(ulHitCount>>8)&

```

```

0xff);
AFE77xx_RegWrite(fd,0x0248 + ulRegOffset,ulHitCount&0xff);

AFE77xx_RegRead(fd,0x0200+ulRegOffset,&ulRegValue);
if(ulEnable==1){
AFE77xx_RegWrite(fd,0x0200 + ulRegOffset,ulRegValue|
0x01);
AFE77xx_RegWrite(fd,0x0204 + ulRegOffset,ulRegValue|
0x01);
}
else{
AFE77xx_RegWrite(fd,0x0200 + ulRegOffset,ulRegValue&
0xfe);
AFE77xx_RegWrite(fd,0x0204 + ulRegOffset,ulRegValue&
0xfe);
}

AFE77xx_RegWrite(fd,0x0010,0);
EnableInternalAgc(fd,(ulChan>>1),0);
EnableInternalAgc(fd,(ulChan>>1),1);
}

U32 readBigStepAttackSettings(int fd, U32 ulChan){
/*
When the internal AGC is enabled, this function is used for
big step attack.
ulChan is from 0 to 3, for each RX Channel.
ulAttackStepSize is the step size in dBFs.
ulAttackThreshold is the attack threshold in dBFs.
ulWindowLen is the window length. For value of N, the window
length will be N cycles of Fs/8 clocks.
ulHitCount is the number of times the signal should be above
threshold in the window for attack to happen.
*/
U32 ulRegValue=0;
U32 ulRegOffset=(0x0400*(ulChan&1));
U8 ulEnable;
U32 ulAttackStepSize;
double ulAttackThreshold=0;
U64 ulWindowLen;
U32 ulHitCount;
U32 ulAttackThresholdWord=0;

AFE77xx_RegWrite(fd,0x0010,(0x4<<(ulChan>>1)));
AFE77xx_RegRead(fd,0x0208 + ulRegOffset,&ulAttackStepSize);
AFE77xx_RegRead(fd,0x0235 + ulRegOffset,&ulRegValue);
ulAttackThresholdWord=(ulRegValue<<8);
AFE77xx_RegRead(fd,0x0234 + ulRegOffset,&ulRegValue);
ulAttackThresholdWord=ulAttackThresholdWord+ulRegValue;
AFE77xx_RegRead(fd,0x0216 + ulRegOffset,&ulRegValue);
ulWindowLen=(ulRegValue<<16);
AFE77xx_RegRead(fd,0x0215 + ulRegOffset,&ulRegValue);

```

```

    ulWindowLen=ulWindowLen+(ulRegValue<<8);
    AFE77xx_RegRead(fd,0x0214 + ulRegOffset,&ulRegValue);
    ulWindowLen=ulWindowLen+ulRegValue;
    AFE77xx_RegRead(fd,0x024a + ulRegOffset,&ulRegValue);
    ulHitCount=(ulRegValue<<16);
    AFE77xx_RegRead(fd,0x0249 + ulRegOffset,&ulRegValue);
    ulHitCount=ulHitCount+(ulRegValue<<8);
    AFE77xx_RegRead(fd,0x0248 + ulRegOffset,&ulRegValue);
    ulHitCount=ulHitCount+ulRegValue;

    ulAttackThreshold=20*log10(((double)
    (ulAttackThresholdWord)/4096));

    AFE77xx_RegRead(fd,0x0200+ulRegOffset,&ulRegValue);
    if (ulRegValue&1==1){
        ulEnable=1;
    }
    else{
        ulEnable=0;
    }
    printf("ulEnable: %d\n",ulEnable);
    printf("ulAttackStepSize: %d\n",ulAttackStepSize);
    printf("ulAttackThreshold: %f\n",ulAttackThreshold);
    printf("ulWindowLen: %llu\n",ulWindowLen);
    printf("ulHitCount: %ld\n",ulHitCount);
    AFE77xx_RegWrite(fd,0x0010,0);
}

U32 SmallStepAttackConfig(int fd, U32 ulChan, U32 ulEnable, U32
ulAttackStepSize, double ulAttackThreshold, U64 ulWindowLen, U32
ulHitCount){
    /* ulChan is from 0 to 3, for each RX Channel.
    ulAttackStepSize is the step size in dBFs. Should be
multiple of 1.
    ulAttackThreshold is the attack threshold in dBFs.
    ulWindowLen is the window length. For value of N, the
window length will be N cycles of Fs/8 clocks.
    ulHitCount is the number of times the signal should be
above threshold in the window for attack to happen.
*/
    U32 ulRegValue=0;
    U32 ulRegOffset=(0x0400*(ulChan&1));
    U32 ulAttackThresholdWord=ceil(4096*pow(10.0,
ulAttackThreshold/20.0));

    AFE77xx_RegWrite(fd,0x0010,(0x4<<(ulChan>>1)));
    AFE77xx_RegWrite(fd,0x0209 + ulRegOffset,ulAttackStepSize);
    AFE77xx_RegWrite(fd,0x0239 +
ulRegOffset,ulAttackThresholdWord>>8);
    AFE77xx_RegWrite(fd,0x0238 +
ulRegOffset,ulAttackThresholdWord&0xff);
    AFE77xx_RegWrite(fd,0x023b +

```

```

ulRegOffset,ulAttackThresholdWord>>8);
AFE77xx_RegWrite(fd,0x023a +
ulRegOffset,ulAttackThresholdWord&0xff);
AFE77xx_RegWrite(fd,0x021a + ulRegOffset,ulWindowLen>>16);
AFE77xx_RegWrite(fd,0x0219 + ulRegOffset,(ulWindowLen>>8) &
0xff);
AFE77xx_RegWrite(fd,0x0218 + ulRegOffset,ulWindowLen&0xff);
AFE77xx_RegWrite(fd,0x024e + ulRegOffset,ulHitCount>>16);
AFE77xx_RegWrite(fd,0x024d + ulRegOffset,(ulHitCount>>8) &
0xff);
AFE77xx_RegWrite(fd,0x024c + ulRegOffset,ulHitCount&0xff);

AFE77xx_RegRead(fd,0x0200+ulRegOffset,&ulRegValue);
if(ulEnable==1){
AFE77xx_RegWrite(fd,0x0200 + ulRegOffset,ulRegValue|
0x02);
AFE77xx_RegWrite(fd,0x0204 + ulRegOffset,ulRegValue|
0x02);
}
else{
AFE77xx_RegWrite(fd,0x0200 + ulRegOffset,ulRegValue&
0xfd);
AFE77xx_RegWrite(fd,0x0204 + ulRegOffset,ulRegValue&
0xfd);
}

AFE77xx_RegWrite(fd,0x0010,0);
EnableInternalAgc(fd,(ulChan>>1),0);
EnableInternalAgc(fd,(ulChan>>1),1);
return 0;
}

U32 readSmallStepAttackConfig(int fd, U32 ulChan){
/* ulChan is from 0 to 3, for each RX Channel.
ulAttackStepSize is the step size in dBFs.
ulAttackThreshold is the attack threshold in dBFs.
ulWindowLen is the window length. For value of N, the
window length will be N cycles of Fs/8 clocks.
ulHitCount is the number of times the signal should be
above threshold in the window for attack to happen.
*/
U32 ulRegValue=0;
U32 ulRegOffset=(0x0400*(ulChan&1));
U8 ulEnable;
U32 ulAttackStepSize;
double ulAttackThreshold=0;
U64 ulWindowLen;
U32 ulHitCount;
U32 ulAttackThresholdWord=0;

AFE77xx_RegWrite(fd,0x0010,(0x4<<(ulChan>>1)));
AFE77xx_RegRead(fd,0x0209 + ulRegOffset,&ulAttackStepSize);

```

```

AFE77xx_RegRead(fd,0x0239 + ulRegOffset,&ulRegValue);
ulAttackThresholdWord=(ulRegValue<<8);
AFE77xx_RegRead(fd,0x0238 + ulRegOffset,&ulRegValue);
ulAttackThresholdWord=ulAttackThresholdWord+ulRegValue;
AFE77xx_RegRead(fd,0x021a + ulRegOffset,&ulRegValue);
ulWindowLen=(ulRegValue<<16);
AFE77xx_RegRead(fd,0x0219 + ulRegOffset,&ulRegValue);
ulWindowLen=ulWindowLen+(ulRegValue<<8);
AFE77xx_RegRead(fd,0x0218 + ulRegOffset,&ulRegValue);
ulWindowLen=ulWindowLen+ulRegValue;
AFE77xx_RegRead(fd,0x024e + ulRegOffset,&ulRegValue);
ulHitCount=(ulRegValue<<16);
AFE77xx_RegRead(fd,0x024d + ulRegOffset,&ulRegValue);
ulHitCount=ulHitCount+(ulRegValue<<8);
AFE77xx_RegRead(fd,0x024c + ulRegOffset,&ulRegValue);
ulHitCount=ulHitCount+ulRegValue;
AFE77xx_RegRead(fd,0x0200+ulRegOffset,&ulRegValue);
ulEnable=((ulRegValue&0x02)>>1);
AFE77xx_RegRead(fd,0x0204+ulRegOffset,&ulRegValue);
ulEnable=ulEnable&((ulRegValue&0x02)>>1);
AFE77xx_RegWrite(fd,0x0010,0);
ulAttackThreshold=20*log10(((double)
(ulAttackThresholdWord)/4096));
printf("ulEnable: %d\n",ulEnable);
printf("ulAttackStepSize: %d\n",ulAttackStepSize);
printf("ulAttackThreshold: %f\n",ulAttackThreshold);
printf("ulWindowLen: %llu\n",ulWindowLen);
printf("ulHitCount: %ld\n",ulHitCount);
return ulEnable;
}

U32 SmallStepDecayConfig(int fd, U32 ulChan, U32 ulEnable, U32
ulDecayStepSize, double ulDecayThreshold, U64 ulWindowLen, U32
ulHitCount){
    /* ulChan is from 0 to 3, for each RX Channel.
       ulDecayStepSize is the step size in dBFs. Should be
multiple of 1.
       ulDecayThreshold is the decay threshold in dBFs.
       ulWindowLen is the window length. For value of N, the
window length will be N cycles of Fs/8 clocks.
       ulHitCount is the number of times the signal should be
below threshold in the window for decay to happen.
    */

    U32 ulRegOffset=(0x0400*(ulChan&1));
    U32 ulDecayThresholdWord=ceil(4096*pow(10.0,
ulDecayThreshold/20.0));
    U32 ulRegValue=0;

    AFE77xx_RegWrite(fd,0x0010,(0x4<<(ulChan>>1)));
    AFE77xx_RegWrite(fd,0x020b + ulRegOffset,ulDecayStepSize);
    AFE77xx_RegWrite(fd,0x0241 +

```



```

ulRegOffset, ulDecayThresholdWord>>8);
    AFE77xx_RegWrite (fd, 0x0240 +
ulRegOffset, ulDecayThresholdWord&0xff);
    AFE77xx_RegWrite (fd, 0x0243 +
ulRegOffset, ulDecayThresholdWord>>8);
    AFE77xx_RegWrite (fd, 0x0242 +
ulRegOffset, ulDecayThresholdWord&0xff);
    AFE77xx_RegWrite (fd, 0x0222 + ulRegOffset, ulWindowLen>>16);
    AFE77xx_RegWrite (fd, 0x0221 + ulRegOffset, (ulWindowLen>>8) &
0xff);
    AFE77xx_RegWrite (fd, 0x0220 + ulRegOffset, ulWindowLen&0xff);
    AFE77xx_RegWrite (fd, 0x0256 + ulRegOffset, ulHitCount>>16);
    AFE77xx_RegWrite (fd, 0x0255 + ulRegOffset, (ulHitCount>>8) &
0xff);
    AFE77xx_RegWrite (fd, 0x0254 + ulRegOffset, ulHitCount&0xff);

    AFE77xx_RegRead (fd, 0x0200+ulRegOffset, &ulRegValue);
    if (ulEnable==1) {
        AFE77xx_RegWrite (fd, 0x0200 + ulRegOffset, ulRegValue |
0x08);
        AFE77xx_RegWrite (fd, 0x0204 + ulRegOffset, ulRegValue |
0x08);
    }
    else {
        AFE77xx_RegWrite (fd, 0x0200 + ulRegOffset, ulRegValue&
0xf7);
        AFE77xx_RegWrite (fd, 0x0204 + ulRegOffset, ulRegValue&
0xf7);
    }
    AFE77xx_RegWrite (fd, 0x0010, 0);

    EnableInternalAgc (fd, (ulChan>>1), 0);
    EnableInternalAgc (fd, (ulChan>>1), 1);
    return 0;
}

U32 readSmallStepDecayConfig (int fd, U32 ulChan) {
    /*    ulChan is from 0 to 3, for each RX Channel.
        ulDecayStepSize is the step size in dBFs.
        ulDecayThreshold is the decay threshold in dBFs.
        ulWindowLen is the window length. For value of N, the
window length will be N cycles of Fs/8 clocks.
        ulHitCount is the number of times the signal should be
below threshold in the window for decay to happen.
    */

    U32 ulRegValue=0;
    U32 ulRegOffset=(0x0400*(ulChan&1));
    U8 ulEnable;
    U32 ulDecayStepSize;
    double ulDecayThreshold=0;
    U64 ulWindowLen;

```

```

U32 ulHitCount;
U32 ulDecayThresholdWord=0;

AFE77xx_RegWrite(fd,0x0010,(0x4<<(ulChan>>1)));
AFE77xx_RegRead(fd,0x020b + ulRegOffset,&ulDecayStepSize);
AFE77xx_RegRead(fd,0x0241 + ulRegOffset,&ulRegValue);
ulDecayThresholdWord=(ulRegValue<<8);
AFE77xx_RegRead(fd,0x0240 + ulRegOffset,&ulRegValue);
ulDecayThresholdWord=ulDecayThresholdWord+ulRegValue;

AFE77xx_RegRead(fd,0x0222 + ulRegOffset,&ulRegValue);
ulWindowLen=(ulRegValue<<16);
AFE77xx_RegRead(fd,0x0221 + ulRegOffset,&ulRegValue);
ulWindowLen=ulWindowLen+(ulRegValue<<8);
AFE77xx_RegRead(fd,0x0220 + ulRegOffset,&ulRegValue);
ulWindowLen=ulWindowLen+ulRegValue;
AFE77xx_RegRead(fd,0x0256 + ulRegOffset,&ulRegValue);
ulHitCount=(ulRegValue<<16);
AFE77xx_RegRead(fd,0x0255 + ulRegOffset,&ulRegValue);
ulHitCount=ulHitCount+(ulRegValue<<8);
AFE77xx_RegRead(fd,0x0254 + ulRegOffset,&ulRegValue);
ulHitCount=ulHitCount+ulRegValue;
AFE77xx_RegRead(fd,0x0200+ulRegOffset,&ulRegValue);
ulEnable=((ulRegValue&0x08)>>3);
AFE77xx_RegRead(fd,0x0204+ulRegOffset,&ulRegValue);
ulEnable=ulEnable&((ulRegValue&0x08)>>3);
AFE77xx_RegWrite(fd,0x0010,0);
ulDecayThreshold=20*log10(((double)
(ulDecayThresholdWord)/4096));
printf("ulEnable: %d\n",ulEnable);
printf("ulDecayStepSize: %d\n",ulDecayStepSize);
printf("ulDecayThreshold: %f\n",ulDecayThreshold);
printf("ulWindowLen: %llu\n",ulWindowLen);
printf("ulHitCount: %ld\n",ulHitCount);
return ulEnable;
}

U32 BigStepDecayConfig(int fd, U32 ulChan, U32 ulEnable, U32
ulDecayStepSize, double ulDecayThreshold, U64 ulWindowLen, U32
ulHitCount){
    /* ulChan is from 0 to 3, for each RX Channel.
        ulDecayStepSize is the step size in dBFs. Should be
multiple of 1.
        ulDecayThreshold is the decay threshold in dBFs.
        ulWindowLen is the window length. For value of N, the
window length will be N cycles of Fs/8 clocks.
        ulHitCount is the number of times the signal should be
below threshold in the window for decay to happen.
    */

    U32 ulRegOffset=(0x0400*(ulChan&1));
    U32 ulDecayThresholdWord=ceil(4096*pow(10.0,

```

```

ulDecayThreshold/20.0));
    U32 ulRegValue=0;

    AFE77xx_RegWrite(fd,0x0010,(0x4<<(ulChan>>1)));
    AFE77xx_RegWrite(fd,0x020a + ulRegOffset,ulDecayStepSize);
    AFE77xx_RegWrite(fd,0x023d +
ulRegOffset,ulDecayThresholdWord>>8);
    AFE77xx_RegWrite(fd,0x023c +
ulRegOffset,ulDecayThresholdWord&0xff);
    AFE77xx_RegWrite(fd,0x023f +
ulRegOffset,ulDecayThresholdWord>>8);
    AFE77xx_RegWrite(fd,0x023e +
ulRegOffset,ulDecayThresholdWord&0xff);
    AFE77xx_RegWrite(fd,0x021e + ulRegOffset,ulWindowLen>>16);
    AFE77xx_RegWrite(fd,0x021d + ulRegOffset,(ulWindowLen>>8) &
0xff);
    AFE77xx_RegWrite(fd,0x021c + ulRegOffset,ulWindowLen&0xff);
    AFE77xx_RegWrite(fd,0x0252 + ulRegOffset,ulHitCount>>16);
    AFE77xx_RegWrite(fd,0x0251 + ulRegOffset,(ulHitCount>>8) &
0xff);
    AFE77xx_RegWrite(fd,0x0250 + ulRegOffset,ulHitCount&0xff);

    AFE77xx_RegRead(fd,0x0200+ulRegOffset,&ulRegValue);
    if(ulEnable==1){
        AFE77xx_RegWrite(fd,0x0200 + ulRegOffset,ulRegValue|
0x04);
        AFE77xx_RegWrite(fd,0x0204 + ulRegOffset,ulRegValue|
0x04);
    }
    else{
        AFE77xx_RegWrite(fd,0x0200 + ulRegOffset,ulRegValue&
0xfb);
        AFE77xx_RegWrite(fd,0x0204 + ulRegOffset,ulRegValue&
0xfb);
    }
    AFE77xx_RegWrite(fd,0x0010,0);

    EnableInternalAgc(fd,(ulChan>>1),0);
    EnableInternalAgc(fd,(ulChan>>1),1);
}

U32 readBigStepDecayConfig(int fd, U32 ulChan){
    /* ulChan is from 0 to 3, for each RX Channel.
        ulDecayStepSize is the step size in dBFs. Should be
multiple of 1.
        ulDecayThreshold is the decay threshold in dBFs.
        ulWindowLen is the window length. For value of N, the
window length will be N cycles of Fs/8 clocks.
        ulHitCount is the number of times the signal should be
below threshold in the window for decay to happen.
    */
    U32 ulRegValue=0;

```

```

U32 ulRegOffset=(0x0400*(ulChan&1));
U8 ulEnable;
U32 ulDecayStepSize;
double ulDecayThreshold=0;
U64 ulWindowLen;
U32 ulHitCount;
U32 ulDecayThresholdWord=0;

AFE77xx_RegWrite(fd,0x0010,(0x4<<(ulChan>>1)));
AFE77xx_RegRead(fd,0x020a + ulRegOffset,&ulDecayStepSize);
AFE77xx_RegRead(fd,0x023d + ulRegOffset,&ulRegValue);
ulDecayThresholdWord=(ulRegValue<<8);
AFE77xx_RegRead(fd,0x023c + ulRegOffset,&ulRegValue);
ulDecayThresholdWord=ulDecayThresholdWord+ulRegValue;
AFE77xx_RegRead(fd,0x021e + ulRegOffset,&ulRegValue);
ulWindowLen=(ulRegValue<<16);
AFE77xx_RegRead(fd,0x021d + ulRegOffset,&ulRegValue);
ulWindowLen=ulWindowLen+(ulRegValue<<8);
AFE77xx_RegRead(fd,0x021c + ulRegOffset,&ulRegValue);
ulWindowLen=ulWindowLen+ulRegValue;
AFE77xx_RegRead(fd,0x0251+ulRegOffset,&ulRegValue);
ulHitCount=(ulRegValue<<16);
AFE77xx_RegRead(fd,0x0251 + ulRegOffset,&ulRegValue);
ulHitCount=ulHitCount+(ulRegValue<<8);
AFE77xx_RegRead(fd,0x0250 + ulRegOffset,&ulRegValue);
ulHitCount=ulHitCount+ulRegValue;
AFE77xx_RegRead(fd,0x0200+ulRegOffset,&ulRegValue);
ulEnable=((ulRegValue&0x04)>>2);
AFE77xx_RegRead(fd,0x0204+ulRegOffset,&ulRegValue);
ulEnable=ulEnable&((ulRegValue&0x04)>>2);
AFE77xx_RegWrite(fd,0x0010,0);
ulDecayThreshold=20*log10(((double)
(ulDecayThresholdWord)/4096));
printf("ulEnable: %d\n",ulEnable);
printf("ulDecayStepSize: %d\n",ulDecayStepSize);
printf("ulDecayThreshold: %f\n",ulDecayThreshold);
printf("ulWindowLen: %llu\n",ulWindowLen);
printf("ulHitCount: %ld\n",ulHitCount);
return ulEnable;
}

U32 rxPowerAttackConfig(int fd,U32 ulChan,U32 ulEnable,U32
ulAttackStepSize,double ulAttackThreshold,U64 ulWindowLen){
/* ulChan is from 0 to 3, for each RX Channel.
ulAttackStepSize is the step size in dBFS. Should be
multiple of 1.
ulAttackThreshold is the attack threshold in dBFS.
ulWindowLen: For value of N, the window length will be
2^N cycles of Fs/8 clocks. Max value for ulWindowLen is 23.
*/
U32 ulRegOffset=(0x0400*(ulChan&1));
U32 ulAttackThresholdWord=ceil(65536*pow(10.0,

```

```

ulAttackThreshold/10.0));

AFE77xx_RegWrite(fd,0x0010,(0x4<<(ulChan>>1)));
AFE77xx_RegWrite(fd,0x020c + ulRegOffset,ulAttackStepSize);
AFE77xx_RegWrite(fd,0x0245 +
ulRegOffset,ulAttackThresholdWord>>8);
AFE77xx_RegWrite(fd,0x0244 +
ulRegOffset,ulAttackThresholdWord&0xff);
AFE77xx_RegWrite(fd,0x0224 + ulRegOffset,ulWindowLen);
AFE77xx_RegReadWrite(fd,0x0200
+ulRegOffset,(ulEnable<<4),4,4);
AFE77xx_RegReadWrite(fd,0x0204
+ulRegOffset,(ulEnable<<4),4,4);
AFE77xx_RegWrite(fd,0x0010,0x0);
return 0;
}

U32 readRxPowerAttackConfig(int fd, int ulChan){
/* ulChan is from 0 to 3, for each RX Channel.
ulAttackStepSize is the step size in dBFS.
ulAttackThreshold is the attack threshold in dBFS.
ulWindowLen: For value of N, the window length will be
2^N cycles of Fs/8 clocks.
*/
U32 ulRegValue=0;
U32 ulRegOffset=(0x0400*(ulChan&1));
U8 ulEnable;
U32 ulAttackStepSize;
double ulAttackThreshold=0;
U64 ulWindowLen;
U32 ulAttackThresholdWord=0;

AFE77xx_RegWrite(fd,0x0010,(0x4<<(ulChan>>1)));
AFE77xx_RegRead(fd,0x020c + ulRegOffset,&ulAttackStepSize);
AFE77xx_RegRead(fd,0x0245 + ulRegOffset,&ulRegValue);
ulAttackThresholdWord=(ulRegValue<<8);
AFE77xx_RegRead(fd,0x0244 + ulRegOffset,&ulRegValue);
ulAttackThresholdWord=ulAttackThresholdWord+ulRegValue;
AFE77xx_RegRead(fd,0x0224 + ulRegOffset,&ulWindowLen);
ulEnable=((AFE77xx_RegRead(fd,0x0200+ulRegOffset,
&ulRegValue)&0x10)>>4);
ulEnable=ulEnable&((AFE77xx_RegRead(fd,0x0204+ulRegOffset,
&ulRegValue)&0x10)>>4);
ulAttackThreshold=10*log10(((double)
(ulAttackThresholdWord)/65536));
printf("ulEnable: %d\n",ulEnable);
printf("ulAttackStepSize: %d\n",ulAttackStepSize);
printf("ulAttackThreshold: %f\n",ulAttackThreshold);
printf("ulWindowLen: %llu\n",ulWindowLen);
AFE77xx_RegWrite(fd,0x0010,0x0);
return ulEnable;
}

```

```

U32 rxPowerDecayConfig(int fd,U32 ulChan,U32 ulEnable,U32
ulDecayStepSize,double ulDecayThreshold,U64 ulWindowLen){
    /*    ulChan is from 0 to 3, for each RX Channel.
        ulDecayStepSize is the step size in dBFS. Should be
multiple of 1.
        ulDecayThreshold is the decay threshold in dBFS.
        ulWindowLen: For value of N, the window length will be
2^N cycles of Fs/8 clocks.
    */
    U32 ulRegOffset=(0x0400*(ulChan&1));
    U32 ulDecayThresholdWord=ceil(65536*pow(10.0,
ulDecayThreshold/10.0));

    AFE77xx_RegWrite(fd,0x0010,(0x4<<(ulChan>>1)));
    AFE77xx_RegWrite(fd,0x020d + ulRegOffset,ulDecayStepSize);
    AFE77xx_RegWrite(fd,0x0247 +
ulRegOffset,ulDecayThresholdWord>>8);
    AFE77xx_RegWrite(fd,0x0246 +
ulRegOffset,ulDecayThresholdWord&0xff);
    AFE77xx_RegWrite(fd,0x0225 + ulRegOffset,ulWindowLen);
    AFE77xx_RegReadWrite(fd,0x0cd1+(ulRegOffset/2),0x8,3,3);
    AFE77xx_RegReadWrite(fd,0x0cd7+(ulRegOffset/2),0x00,5,5);
    AFE77xx_RegReadWrite(fd,0x0200
+ulRegOffset,(ulEnable<<5),5,5);
    AFE77xx_RegReadWrite(fd,0x0204
+ulRegOffset,(ulEnable<<5),5,5);
    AFE77xx_RegWrite(fd,0x0010,0x0);
    return 0;
}

U32 readRxPowerDecayConfig(int fd,int ulChan){
    /*    ulChan is from 0 to 3, for each RX Channel.
        ulDecayStepSize is the step size in dBFS.
        ulDecayThreshold is the decay threshold in dBFS.
        ulWindowLen: For value of N, the window length will be
2^N cycles of Fs/8 clocks.
    */
    U32 ulRegValue=0;
    U32 ulRegOffset=(0x0400*(ulChan&1));
    U8 ulEnable;
    U32 ulDecayStepSize;
    double ulDecayThreshold=0;
    U64 ulWindowLen;
    U32 ulDecayThresholdWord=0;

    AFE77xx_RegWrite(fd,0x0010,(0x4<<(ulChan>>1)));
    AFE77xx_RegRead(fd,0x020d + ulRegOffset,&ulDecayStepSize);
    AFE77xx_RegRead(fd,0x0247 + ulRegOffset,&ulRegValue);
    ulDecayThresholdWord=(ulRegValue<<8);
    AFE77xx_RegRead(fd,0x0246 + ulRegOffset,&ulRegValue);
    ulDecayThresholdWord=ulDecayThresholdWord+ulRegValue;

```

```

    AFE77xx_RegRead(fd,0x0225 + ulRegOffset,&ulWindowLen);
    ulEnable=( (AFE77xx_RegRead(fd,0x0200+ulRegOffset,
&ulRegValue) &0x20)>>5);
    ulEnable=ulEnable&( (AFE77xx_RegRead(fd,0x0204+ulRegOffset,
&ulRegValue) &0x20)>>5);
    ulDecayThreshold=10*log10(((double)
(ulDecayThresholdWord)/65536));
    printf("ulEnable: %d\n",ulEnable);
    printf("ulDecayStepSize: %d\n",ulDecayStepSize);
    printf("ulDecayThreshold: %f\n",ulDecayThreshold);
    printf("ulWindowLen: %llu\n",ulWindowLen);
    AFE77xx_RegWrite(fd,0x0010,0x0);
    return ulEnable;
}

U32 ExtLnaControlConfig(int fd, U32 ulChan, U32 ulEnable, U32
ulLnaGain, U32 ulGainMargin, U32 ulBlankingTime){
    /* ulChan is from 0 to 3, for each RX Channel.
    ulLnaGain is the LNA Gain in dB.
    ulGainMargin is the LNA Gain Margin in dB.
    ulBlankingTime is the blanking time for AGC. For value
of N, the window length will be N cycles of Fs/8 clocks. This
time should be greater than 1000.
    */
    U32 ulRegValue=0;
    U32 ulRegOffset=(0x0400*(ulChan&1));
    ulLnaGain=ulLnaGain<<5;

    AFE77xx_RegWrite(fd,0x0010,(0x4<<(ulChan>>1)));
    AFE77xx_RegWrite(fd,0x0291 + ulRegOffset,ulLnaGain>>8);
    AFE77xx_RegWrite(fd,0x0290 + ulRegOffset,ulLnaGain&0xff);

    AFE77xx_RegWrite(fd,0x0276 + ulRegOffset,ulGainMargin);

    AFE77xx_RegWrite(fd,0x0275 + ulRegOffset,ulBlankingTime>>8);
    AFE77xx_RegWrite(fd,0x0274 + ulRegOffset,ulBlankingTime&
0xff);

    AFE77xx_RegRead(fd,0x0202+ulRegOffset,&ulRegValue);
    if(ulEnable==1){
        AFE77xx_RegWrite(fd,0x0202 + ulRegOffset,ulRegValue|
0x02);
    }
    else{
        AFE77xx_RegWrite(fd,0x0202 + ulRegOffset,ulRegValue&
0xfd);
    }

    AFE77xx_RegWrite(fd,0x0010,0);
    EnableInternalAgc(fd,(ulChan>>1),0);
    EnableInternalAgc(fd,(ulChan>>1),1);
    return 0;
}

```

```

}

U32 OverrideLnaPinControl(int fd, U32 ulChan, U32 ulOverride, U32
ulOverrideValue){
    /* ulOverride=1 will override the pin output of the LNA. 0
will give internal AGC the control.
    ulOverrideValue is the value to get on the pin when
ulOverride=1 */
    U32 ulRegValue=0;
    AFE77xx_RegWrite(fd,0x0015,0x80);
    AFE77xx_RegRead(fd,0x0592,&ulRegValue);
    ulRegValue=ulRegValue&(0xff^(3<<(ulChan*2)));
    ulRegValue=ulRegValue+(((ulOverride<<1)+ulOverrideValue)
<<(ulChan*2));
    AFE77xx_RegWrite(fd,0x0592,ulRegValue);
    AFE77xx_RegWrite(fd,0x0015,0);
}

U32 EnableDgc(int fd, U32 ulChan, U32 ulEnable){
    /* ulChan is from 0 to 3, for each RX Channel. */
    U32 ulPageRegValue=0;
    U32 ulRegOffset=0x0400*(ulChan&1);
    /* To Enable or Disable AGC */
    AFE77xx_RegWrite(fd,0x0010,(0x4<<(ulChan>>1)));
    AFE77xx_RegWrite(fd,0x0334+ulRegOffset,0x06+ulEnable);
    AFE77xx_RegWrite(fd,0x0010,0);
    return 0;
}

U32 readEnableDgc(int fd, U32 ulChan){
    /* ulChan is from 0 to 3, for each RX Channel. */
    U32 readValue=0;
    U32 ulRegOffset=0x0400*(ulChan&1);
    /* To Enable or Disable AGC */
    AFE77xx_RegWrite(fd,0x0010,(0x4<<(ulChan>>1)));
    AFE77xx_RegRead(fd,0x0334+ulRegOffset,&readValue);
    AFE77xx_RegWrite(fd,0x0010,0);
    readValue = readValue&1;
    return readValue;
}

U32 GetCurrentRxAttenuation(int fd, U32 ulChan){
    U32 ulRegValue=0;
    /* ulChan is from 0 to 3, for each RX Channel. */
    AFE77xx_RegWrite(fd,0x0016,(0x10<<(ulChan>>1)));
    AFE77xx_RegRead(fd,0x02f7+((ulChan&1)*0x0360),&ulRegValue);
    printf("Current Attenuation for RX Channel %d:%d\r
\n",ulChan,ulRegValue);
    AFE77xx_RegWrite(fd,0x0016,0);
    return ulRegValue;
}

```



```

U32 FreezeAgc(int fd, U32 ulChan, U32 ulFreeze){
    /* ulChan is from 0 to 3, for each RX Channel. Making
    ulFreeze as 1 will freeze the AGC in current state, till it is
    made 0 again.*/
    U32 ulPageRegValue=0;
    U32 ulRegOffset=(0x0400*(ulChan&1));
    /* To Enable or Disable AGC */
    AFE77xx_RegWrite(fd,0x0010,(0x4<<(ulChan>>1)));
    AFE77xx_RegWrite(fd,0x0266+ulRegOffset,ulFreeze);
    AFE77xx_RegWrite(fd,0x0010,0);
    return 0;
}

```

```

U32 rxDcBw(int fd, U32 ulChan, double rxOutputDataRate, double
dcFilterBw){
    /* dcFilterBw should be in Hz and rxOutputDataRate should be
in MHz

```

Data Rate: 122.88M	184.32M	245.76M
368.64M	61.44M	92.16M
dcFilterBwWord		

Supported DC bandwidth
for each data rate:

	2.62M	3.92M	5.23M	7.85M
1.31M	1.96M	3		
	1.26M	1.89M	2.53M	3.79M
631.31K	946.96K	4		
	620.96K	931.44K		1.24M
1.86M	310.48K	465.72K	5	
	308.00K	461.99K		615.99K
923.99K	154.00K	231.00K		6
	153.39K	230.08K		306.78K
460.17K	76.69K	115.04K		7
	76.54K	114.82K		153.09K
229.63K	38.27K	57.41K		8
	38.23K	57.35K		76.47K
114.70K	19.12K	28.68K		9
	19.11K	28.66K		38.22K
57.32K	9.55K	14.33K	10	
	9.55K	14.33K		19.10K
28.65K	4.78K	7.16K	11	
	4.78K	7.16K	9.55K	14.33K
2.39K	3.58K	12		
	2.39K	3.58K	4.77K	7.16K
1.19K	1.79K	13		
	1.19K	1.79K	2.39K	3.58K
0.60K	0.90K	14		
	596.84	895.26		1193.68
1790.52	298.42	447.63		15
	298.42	447.63		596.84

895.25		149.21		223.81		16
		149.21		223.81		298.42
447.62		74.6		111.91		17
		74.6		111.91		149.21
223.81		37.3		55.95		18
		37.3		55.95		74.6
						111.91
18.65	27.98		19			
		18.65		27.98		37.3
			20			55.95
9.33	13.99					
		9.33		13.99		18.65
						27.98
4.66	6.99		21			
		4.66		6.99		9.33
						13.99
2.33	3.5			22		

```

*/
U32 dcFilterBwWord=0,i;
double tempConst,tempConst1;
double bwlookupTable[20]={21321.61458333, 10275.22786458,
5053.38541667, 2506.51041667, 1248.20963542, 622.88411458,
311.19791667, 155.43619792, 77.79947917, 38.89973958,
19.36848958, 9.765625, 4.85709635, 2.42854818, 1.21419271,
0.60709635, 0.30354818, 0.15185547, 0.07584635, 0.03792318};
U32 ulRegOffset=(0x0400*(ulChan&1));
/*dcFilterBwWord=(U32) (3+(log(round(dcFilterBw/47.0)) *
1.4426950408));*/
tempConst=dcFilterBw/rxOutputDataRate;
for (i=0;i<20;i++){
    /*printf("calc: %d \t %lf\n",i,100.0
*bwlookupTable[i]/tempConst);*/
    tempConst1=1000*bwlookupTable[i]/tempConst;
    if (tempConst1<1010 && tempConst1>990){
        dcFilterBwWord=i+3;
        break;
    }
}
printf("%d\n",dcFilterBwWord);
AFE77xx_RegWrite(fd,0x0010,(0x4<<(ulChan>>1)));
AFE77xx_RegWrite(fd,0x407 + ulRegOffset,dcFilterBwWord);
AFE77xx_RegWrite(fd,0x40b + ulRegOffset,dcFilterBwWord);
AFE77xx_RegWrite(fd,0x40f + ulRegOffset,dcFilterBwWord);
AFE77xx_RegWrite(fd,0x417 + ulRegOffset,dcFilterBwWord);
AFE77xx_RegWrite(fd,0x41b + ulRegOffset,dcFilterBwWord);
AFE77xx_RegWrite(fd,0x41f + ulRegOffset,dcFilterBwWord);
AFE77xx_RegWrite(fd,0x420 + ulRegOffset,0);
AFE77xx_RegWrite(fd,0x420 + ulRegOffset,1);
AFE77xx_RegWrite(fd,0x420 + ulRegOffset,0);
AFE77xx_RegWrite(fd,0x421 + ulRegOffset,0);
AFE77xx_RegWrite(fd,0x421 + ulRegOffset,1);
AFE77xx_RegWrite(fd,0x421 + ulRegOffset,0);
AFE77xx_RegWrite(fd,0x0010,0);
return 0;
}

```

```

U32 RXDsaSwapAttnMacro(int fd, U32 ulChan, U32 ulRXDsa){
    U32 ulRegValue = 0;
    U32 ulDsaReg = 0;

    ulDsaReg=0x5C+(0x60*(ulChan&1));

    AFE77xx_RegWrite(fd,0x0013,0x10);
    AFE77xx_RegWrite(fd,0x00a0,0x13);
    AFE77xx_RegWrite(fd,0x00a1,ulDsaReg);
    AFE77xx_RegWrite(fd,0x00a2,0x00);
    AFE77xx_RegWrite(fd,0x00a3,0x00);
    if ((ulChan>>1)==0){
        AFE77xx_RegWrite(fd,0x00a4,0xae);
    }
    else{
        AFE77xx_RegWrite(fd,0x00a4,0xaf);
    }
    AFE77xx_RegWrite(fd,0x00a5,ulRXDsa);
    AFE77xx_RegWrite(fd,0x0193,0x02);
    AFE77xx_RegWrite(fd,0x0013,0x10);
    return 0;
}

U32 OnlyEnableInternalAgc(int fd, U32 ulChan, U32 ulEnable){
    /* ulChan value is 0 for A, 1 for B, 2 for C and 3 for D */

    /* To Enable or Disable AGC */
    AFE77xx_RegWrite(fd,0x0010,(0x04<<(ulChan>>1)));
    AFE77xx_RegWrite(fd,0x0264+(0x400*(ulChan&1)),ulEnable);
    AFE77xx_RegWrite(fd,0x0010,0);
    return 0;
}

U32 programLnaGainPhaseCorrection(int fd, U32 ulChan, U32
ulIndex, float gainValue, float phaseValue){
    /* ulChan is the channel number from 0-3
        ulIndex is the index of the LUT to program
        gainValue is the value in dB.
        phaseValue is the value in degrees
    */

    U32 gainValueWord,phaseValueWord,ulRegOffset;
    gainValueWord=ceil(gainValue*32);

    if (phaseValue>=0){
        phaseValueWord=ceil(phaseValue*0x400/360);
    }
    else{
        phaseValueWord=0x3ff-ceil(-phaseValue*0x400/360);
    }
}

```

```

    ulRegOffset=(0x400*(ulChan&1))+(ulIndex<<1);
    AFE77xx_RegWrite(fd,0x0010,(0x04<<(ulChan>>1)));
    AFE77xx_RegWrite(fd,0x0290+ulRegOffset,gainValueWord&0xff);
    AFE77xx_RegWrite(fd,0x0291+ulRegOffset,(gainValueWord)>>8);
    AFE77xx_RegWrite(fd,0x02d0+ulRegOffset,phaseValueWord&0xff);
    AFE77xx_RegWrite(fd,0x02d1+ulRegOffset,(phaseValueWord)>>8);
    AFE77xx_RegWrite(fd,0x0010,0);
    return 0;
}

U32 selectLnaCorrectionIndex(int fd, U32 ulChan, U32 ulIndex){
    /* ulChan is the channel number from 0-3
       ulIndex is the index of the LUT to select for
    correction
    */
    U32 ulRegOffset;
    ulRegOffset=0x400*(ulChan&1);

    AFE77xx_RegWrite(fd,0x0010,(0x04<<(ulChan>>1)));
    AFE77xx_RegWrite(fd,0x0284+ulRegOffset,ulIndex);
    AFE77xx_RegWrite(fd,0x0010,0);
}

int loadRxDsaPacket(int fd, U8 *array, U16 arraySize){
    /*Pass the Array saved during the DSA calibration and it's
    size. */
    U16 i;
    AFE77xx_RegWrite(fd,0x12,0x10);
    for (i = 0; i < arraySize; ++i) {
        AFE77xx_RegWrite(fd,0x20+i,array[i]);
    }
    AFE77xx_RegWrite(fd,0x12,0x00);
    waitForMacroReady(fd);
    AFE77xx_RegWrite(fd,0x13,0x10);
    AFE77xx_RegWrite(fd,0x00a3,0x00);
    AFE77xx_RegWrite(fd,0x00a2,0x00);
    AFE77xx_RegWrite(fd,0x00a1,0x00);
    AFE77xx_RegWrite(fd,0x00a0,0x02);
    AFE77xx_RegWrite(fd,0x0193,0x12);
    AFE77xx_RegWrite(fd,0x0192,0x00);
    AFE77xx_RegWrite(fd,0x0191,0x00);
    AFE77xx_RegWrite(fd,0x0190,0x00);
    AFE77xx_RegWrite(fd,0x13,0x00);
    waitForMacroDone(fd);
}

U32 RXDsaAttnOffset(int fd, U32 ulChan, U32 ulRXDsa){
    U32 ulPageRegValue = 0;
    U32 ulRegValue = 0;
    U32 ulDsaReg = 0;

    ulPageRegValue = 0x10 << (ulChan>>1);

```

```

    ulDsaReg=0x52+(0x60*(ulChan&1));

    AFE77xx_RegWrite(fd,0x0016,ulPageRegValue);
    AFE77xx_RegWrite(fd,ulDsaReg,ulRXDsa);
    AFE77xx_RegWrite(fd,0x0016,0x00);
    return 0;
}

void overrideRxTdd(int fd, int overrideEn, int overrideVal){
    /*
    overrideEn: Bit 0 for AB and Bit 1 for CD. If the bit is 1,
    the SPI will control TDD. If the bit is 0, pin will control TDD.
    overrideVal: Bit 0 for AB and Bit 1 for CD. If the bit is 1,
    TDD is ON and 0, TDD is off.
    */
    AFE77xx_RegWrite(fd,0x0014,0x04);
    AFE77xx_RegWrite(fd,0x0126,overrideEn&1);
    AFE77xx_RegWrite(fd,0x012C,overrideEn>>1);
    AFE77xx_RegWrite(fd,0x0127,overrideVal&1);
    AFE77xx_RegWrite(fd,0x012D,overrideVal>>1);
    AFE77xx_RegWrite(fd,0x0014,0x00);
}

unsigned int rxDsaGainRead(int fd, U32 ch, U32 dsaLimit){
    /* function to read the Rx gain after calibration selected
    by Channel
        0 - RxA
        1 - RxB
        2 - RxC
        3 - RxD
    */
    int i;
    int gainRead[dsaLimit];
    int temp;

    AFE77xx_RegWrite(fd, 0x0016, (0x10<<(ch>>1)));
    for(i=0;i<dsaLimit; i++){
        AFE77xx_RegRead(fd, (0x03B2+(0x360*(ch&0x1))+4*i),
&temp);
        gainRead[i] = temp;
        printf("Gain read for DSA %d is %d. \n", i,
gainRead[i]);
    }
    AFE77xx_RegWrite(fd, 0x0016, 0x00);
    return gainRead;
}

unsigned int rxDsaPhaseRead(int fd, U32 ch, U32 dsaLimit){
    /* function to read the Rx phase after calibration selected
    by Channel
        0 - RxA
        1 - RxB

```

```

        2 - RxC
        3 - RxD
    */
    int i;
    int phaseRead[dsaLimit];
    int temp;

    AFE77xx_RegWrite(fd, 0x0016, (0x10<<(ch>>1)));
    for(i=0;i<dsaLimit; i++){
        AFE77xx_RegRead(fd, (0x03B0+(0x360*(ch&0x1))+4*i),
&temp);

        phaseRead[i] = temp;
        AFE77xx_RegRead(fd, (0x03B1+(0x360*(ch&0x1))+4*i),
&temp);

        phaseRead[i] = phaseRead[i] + (temp<<8);
        printf("Phase read for DSA %d is %d. \n", i,
phaseRead[i]);
    }
    AFE77xx_RegWrite(fd, 0x0016, 0x00);
    return phaseRead;
}

U32 rxLolEn(int fd, U32 en, U32 ch){
    /*
    enable or disable the LO correction of Rx
    en - 0 - disable LOL
    en - 1 - enable LOL
    ch - 0 - RxA
        - 1 - RxB
        - 2 - RxC
        - 3 - RxD
    */
    AFE77xx_RegWrite(fd, 0x0010, (0x04<<(ch>>0x1)));
    AFE77xx_RegReadWrite(fd, (0x400<<(ch&0x1)), (0x01>>en), 0,
0);
    AFE77xx_RegReadWrite(fd, ((0x400<<(ch&0x1))+0x10), (0x01>>
en), 0, 0);
    AFE77xx_RegWrite(fd, 0x0010, 0x00);
    return 0;
}

U32 rxQmcEn(int fd, U32 en, U32 ch){
    /*
    enable or disable the QMC of Rx
    en - 0 - disable QMC
    en - 1 - enable QMC
    ch - 0 - RxA
        - 1 - RxB
        - 2 - RxC
        - 3 - RxD
    */
    AFE77xx_RegWrite(fd, 0x0010, (0x04<<(ch>>0x1)));

```

```

AFE77xx_RegWrite(fd, ((0x400<<(ch&0x1))+0x100), (0x01>>en));
AFE77xx_RegWrite(fd, 0x0010, 0x00);
return 0;
}

U32 RxDataCapture(fd, samples,Ch){
    /* to capture data from Rx inside the AFE. maximum samples
are 512
        Channel to capture is selected by Ch.
    */
    int done_val =0;
    int count = 0;
    int data_i[samples];
    int data_q[samples];
    int tempVal=0;

    if(samples> 512){
        printf("maximum samples allowed is 1024 \n");
        return 0;
    }
    AFE77xx_RegWrite(fd, 0x0010, (0x04<<(Ch>>1)));
    AFE77xx_RegReadWrite(fd, 0x0A03, 0x00,1,1);
    AFE77xx_RegWrite(fd, 0x0010, 0x00);

    AFE77xx_RegWrite(fd, 0x0013, 0x20);
    int retVal1 = AFE77xx_RegRead(fd, 0x0294,&tempVal);
    int retVal2 = AFE77xx_RegRead(fd, 0x0295,&tempVal);
    AFE77xx_RegWrite(fd, 0x0294, (retVal1&0xea));
    AFE77xx_RegWrite(fd, 0x0295, (retVal2&0xbf));
    AFE77xx_RegReadWrite(fd, 0x009E, 0x01, 0, 0);
    AFE77xx_RegReadWrite(fd, 0x00A4, 0x10, 4, 4);
    AFE77xx_RegReadWrite(fd, 0x00B4, 0x40, 6, 6);
    AFE77xx_RegWrite(fd, 0x00C3, 0x7F);
    AFE77xx_RegWrite(fd, 0x00C2, 0xFF);
    AFE77xx_RegReadWrite(fd, 0x00C1, 0x07<<2, 2, 5);
    AFE77xx_RegReadWrite(fd, 0x00C6, 0x07<<3, 3, 6);

    AFE77xx_RegWrite(fd, 0x00C9, 0x3F);
    AFE77xx_RegWrite(fd, 0x00C8, 0xFF);
    AFE77xx_RegWrite(fd, 0x008E, 0x3F);
    AFE77xx_RegWrite(fd, 0x008D, 0xFF);
    AFE77xx_RegWrite(fd, 0x0092, 0x3F);
    AFE77xx_RegWrite(fd, 0x0091, 0xFF);
    AFE77xx_RegWrite(fd, 0x00D9, 0x3F);
    AFE77xx_RegWrite(fd, 0x00D8, 0xFF);

    AFE77xx_RegReadWrite(fd, 0x00BA, 0x00, 0, 0);
    AFE77xx_RegReadWrite(fd, 0x00BB, 0x00, 0, 0);

    AFE77xx_RegReadWrite(fd, 0x0095, 0x01, 0, 0);
    AFE77xx_RegReadWrite(fd, 0x0095, 0x00, 0, 0);

```

```

AFE77xx_RegReadWrite(fd, 0x00F3, 0x01, 0, 0);
AFE77xx_RegReadWrite(fd, 0x00F3, 0x00, 0, 0);

AFE77xx_RegReadWrite(fd, 0x00A8, 0x0A, 0, 3);
AFE77xx_RegWrite(fd, 0x00CC, 0x00);
AFE77xx_RegReadWrite(fd, 0x00BF, 0x01, 0, 0);

printf("trigger \n");
AFE77xx_RegReadWrite(fd, 0x008C, 0x01, 0, 0);
AFE77xx_RegReadWrite(fd, 0x008C, 0x00, 0, 0);
sysUsDelay(3000);

while(count<0xff){
    AFE77xx_RegRead(fd, 0x00DC, &done_val);
    printf("done Value %d\n",done_val);
    if((done_val&(0x01<<Ch)) == (0x01<<Ch)){
        printf("done \n");
        break;
    }
    sysUsDelay(1000);
    count++;
}
if(count>=0xff){
    printf("capture not done\n");
    AFE77xx_RegWrite(fd, 0x0013, 0x00);
    return -1;
}

AFE77xx_RegReadWrite(fd, 0x00BA, 0x01, 0, 0);
AFE77xx_RegReadWrite(fd, 0x00A9, 0x01, 0, 1);

AFE77xx_RegReadWrite(fd, 0x00D7, 0x00, 2, 3);

AFE77xx_RegReadWrite(fd, 0x00AE, 0x01, 0, 0);
AFE77xx_RegReadWrite(fd, 0x00AE, 0x00, 0, 0);

sysUsDelay(1000);
count = 0;
while(count<0xff){
    AFE77xx_RegRead(fd, 0x00FE, &done_val);
    if((done_val&0x08) == 0x08){
        printf("done\n");
        break;
    }
    sysUsDelay(1000);
    count++;
}
if(count>= 0xff){
    printf("Capture not done\n");
    AFE77xx_RegWrite(fd, 0x0013, 0x00);
}

```



```

        return -1;
    }

AFE77xx_RegReadWrite(fd, 0x0108, 0x80, 7, 7);
AFE77xx_RegReadWrite(fd, 0x0108, 0x10, 4, 4);
AFE77xx_RegReadWrite(fd, 0x0109, (0x01>>(Ch>>1)), 0, 0);

AFE77xx_RegWrite(fd, 0x0013, 0x00);

AFE77xx_RegWrite(fd, 0x0013, 0x40);

int start_address_i;
int start_address_q;
if(Ch&0x1 == 0){
    start_address_i = 0x20;
    start_address_q = 0x20 + 4*1024;
}
else{
    start_address_i = 0x20+4*1024*4;
    start_address_q = 0x20+5*1025*4;
}
int val;
int temp;

for(int i =0; i<samples; i++){
    AFE77xx_RegRead(fd, start_address_i+(i*4), &temp);
    val = temp;
    AFE77xx_RegRead(fd, start_address_i+1+(i*4), &temp);
    val = val+(temp<<8);
    AFE77xx_RegRead(fd, start_address_i+2+(i*4), &temp);
    val = val+(temp<<16);
    AFE77xx_RegRead(fd, start_address_i+3+(i*4), &temp);
    val = val+(temp<<24);

    data_i[i] = twos_comp(val,32);

    AFE77xx_RegRead(fd, start_address_q+(i*4), &temp);
    val = temp;
    AFE77xx_RegRead(fd, start_address_q+1+(i*4), &temp);
    val = val+(temp<<8);
    AFE77xx_RegRead(fd, start_address_q+2+(i*4), &temp);
    val = val+(temp<<16);
    AFE77xx_RegRead(fd, start_address_q+3+(i*4), &temp);
    val = val+(temp<<24);

    data_q[i] = twos_comp(val,32);
}
AFE77xx_RegWrite(fd, 0x0294, (retVal1));
AFE77xx_RegWrite(fd, 0x0295, (retVal2));
AFE77xx_RegWrite(fd, 0x0013, 0x00);

```

```

printf("i data\n");
for(int i=0;i<samples;i++){
    printf("%d ",data_i[i]);
}
printf("\n");
printf("q data\n");
for(int i=0;i<samples;i++){
    printf("%d ",data_q[i]);
}
printf("\n");
return 0;
}

float rxPowDet(int fd,int ch){
/*ch= 0:A, 1:B, 2:C, 3:D*/
int a,b;
float pkVal;
int freezeAgcStatus;
U32 ulRegOffset=(0x0400*(ch&1));

AFE77xx_RegWrite(fd,0x0010,0x4<<(ch>>1));
freezeAgcStatus=AFE77xx_RegRead(fd,0x0266+ulRegOffset,
&freezeAgcStatus);
AFE77xx_RegWrite(fd,0x0266+ulRegOffset,1);
AFE77xx_RegWrite(fd,0x1A1,0x0F);
if(ch%2==0){
    AFE77xx_RegReadWrite(fd,0x0204,0x1A,4,4);
    AFE77xx_RegWrite(fd,0x0311,0x00);
    AFE77xx_RegWrite(fd,0x0311,0x01);
    AFE77xx_RegWrite(fd,0x0311,0x00);
    a=AFE77xx_RegRead(fd,0x0315,&a);
    b=AFE77xx_RegRead(fd,0x0314,&b);
}
else{
    AFE77xx_RegReadWrite(fd,0x0604,0x1A,4,4);
    AFE77xx_RegWrite(fd,0x0711,0x00);
    AFE77xx_RegWrite(fd,0x0711,0x01);
    AFE77xx_RegWrite(fd,0x0711,0x00);
    a=AFE77xx_RegRead(fd,0x0715,&a);
    b=AFE77xx_RegRead(fd,0x0714,&b);
}
AFE77xx_RegWrite(fd,0x1A1,0x0);
AFE77xx_RegWrite(fd,0x0266+ulRegOffset,freezeAgcStatus);
AFE77xx_RegWrite(fd,0x10,0x00);

pkVal=10*log10((float)((a*256)+b)/pow(2,15));
printf("RX Power=%lfdBFS\n",pkVal);
return pkVal;
}

int rxDsaCalibration(int fd,int CHAIN_TO_CALIB){
/*    bit 0:RXA,1:RXB,2:RXC,3:RXD

```

```

        value=0: Calib, 1: Do not calib
*/
U32 readVal[990];
U32 tddPinCtrl[12];
U32 rxDsaMode[2];
//Preserve "Pin control on TDD" settings
AFE77xx_RegWrite(fd, 0x14,0x04);
AFE77xx_RegRead(fd, 0x124,&tddPinCtrl[0]);
AFE77xx_RegRead(fd, 0x126,&tddPinCtrl[1]);
AFE77xx_RegRead(fd, 0x128,&tddPinCtrl[2]);
AFE77xx_RegRead(fd, 0x12A,&tddPinCtrl[3]);
AFE77xx_RegRead(fd, 0x12C,&tddPinCtrl[4]);
AFE77xx_RegRead(fd, 0x12E,&tddPinCtrl[5]);
AFE77xx_RegRead(fd, 0x125,&tddPinCtrl[6]);
AFE77xx_RegRead(fd, 0x127,&tddPinCtrl[7]);
AFE77xx_RegRead(fd, 0x129,&tddPinCtrl[8]);
AFE77xx_RegRead(fd, 0x12B,&tddPinCtrl[9]);
AFE77xx_RegRead(fd, 0x12D,&tddPinCtrl[10]);
AFE77xx_RegRead(fd, 0x12F,&tddPinCtrl[11]);
AFE77xx_RegWrite(fd, 0x14,0x00);

//START: Starting RX DSA Calibration
AFE77xx_RegWrite(fd,0x0015,0x80);
AFE77xx_RegWrite(fd,0x0550,0xaa);
AFE77xx_RegWrite(fd,0x0551,0x02);
AFE77xx_RegWrite(fd,0x0552,0x2a);
AFE77xx_RegWrite(fd,0x0550,0xaf);
AFE77xx_RegWrite(fd,0x0551,0x02);
AFE77xx_RegWrite(fd,0x0552,0x2a);
AFE77xx_RegWrite(fd,0x0015,0x00);
AFE77xx_RegWrite(fd,0x0014,0x04);
AFE77xx_RegWrite(fd,0x0121,0x01);
AFE77xx_RegWrite(fd,0x0122,0x00);
AFE77xx_RegWrite(fd,0x0014,0x00);
AFE77xx_RegWrite(fd,0x0016,0x10);
AFE77xx_RegRead(fd,0x0040,&rxDsaMode[0]);
AFE77xx_RegWrite(fd,0x0016,0x20);
AFE77xx_RegRead(fd,0x0040,&rxDsaMode[1]);
AFE77xx_RegWrite(fd,0x0016,0x30);
AFE77xx_RegWrite(fd,0x0040,0x03);
AFE77xx_RegWrite(fd,0x0016,0x00);
AFE77xx_RegWrite(fd,0x0010,0x04);
AFE77xx_RegWrite(fd,0x0402,0x01);
AFE77xx_RegWrite(fd,0x0802,0x01);
AFE77xx_RegWrite(fd,0x0010,0x08);
AFE77xx_RegWrite(fd,0x0402,0x01);
AFE77xx_RegWrite(fd,0x0802,0x01);
AFE77xx_RegWrite(fd,0x0010,0x00);
AFE77xx_RegWrite(fd,0x0013,0x02);
AFE77xx_RegWrite(fd,0x350d,0x64);
AFE77xx_RegWrite(fd,0x0013,0x00);
AFE77xx_RegWrite(fd,0x0013,0x02);

```

```

AFE77xx_RegWrite (fd, 0x3534, 0x99);
AFE77xx_RegWrite (fd, 0x3535, 0x32);
AFE77xx_RegWrite (fd, 0x3536, 0xe4);
AFE77xx_RegWrite (fd, 0x3537, 0x08);
AFE77xx_RegWrite (fd, 0x0013, 0x00);
AFE77xx_RegWrite (fd, 0x0016, 0x10);
AFE77xx_RegWrite (fd, 0x02fb, 0x00);
AFE77xx_RegWrite (fd, 0x065b, 0x00);
AFE77xx_RegWrite (fd, 0x0016, 0x20);
AFE77xx_RegReadWrite (fd, 0x02fb, 0x00, 0, 0);
AFE77xx_RegReadWrite (fd, 0x065b, 0x00, 0, 0);
AFE77xx_RegWrite (fd, 0x0016, 0x00);

//api_factory_calibration_control
waitForMacroReady (fd);
AFE77xx_RegWrite (fd, 0x0013, 0x10);
AFE77xx_RegWrite (fd, 0x00a3, 0x00);
AFE77xx_RegWrite (fd, 0x00a2, 0x00);
AFE77xx_RegWrite (fd, 0x00a1, 0x02);
AFE77xx_RegWrite (fd, 0x00a0, 0x01);
AFE77xx_RegWrite (fd, 0x00a7, 0x00);
AFE77xx_RegWrite (fd, 0x00a6, 0x00);
AFE77xx_RegWrite (fd, 0x00a5, CHAIN_TO_CALIB);
AFE77xx_RegWrite (fd, 0x00a4, 0x1c);
AFE77xx_RegWrite (fd, 0x00ab, 0x0e);
AFE77xx_RegWrite (fd, 0x00aa, 0x00);
AFE77xx_RegWrite (fd, 0x00a9, 0x00);
AFE77xx_RegWrite (fd, 0x00a8, 0x00);
AFE77xx_RegWrite (fd, 0x0193, 0x41);
AFE77xx_RegWrite (fd, 0x0013, 0x00);
waitForMacroAck (fd);
waitForMacroDone (fd);

U32 chainSel=0;

//api_factory_calibration_control_continue
for (int currChannel=0; currChannel<4; currChannel=currChannel+
1) {
    if ((CHAIN_TO_CALIB>>currChannel)&1)==1) {
        chainSel=1<<currChannel;
        waitForMacroReady (fd);
        AFE77xx_RegWrite (fd, 0x0013, 0x10);
        AFE77xx_RegWrite (fd, 0x00a3, 0x00);
        AFE77xx_RegWrite (fd, 0x00a2, chainSel);
        AFE77xx_RegWrite (fd, 0x00a1, 0x02);
        AFE77xx_RegWrite (fd, 0x00a0, 0x02);
        AFE77xx_RegWrite (fd, 0x00a7, 0x00);
        AFE77xx_RegWrite (fd, 0x00a6, 0x00);
        AFE77xx_RegWrite (fd, 0x00a5, chainSel);
        AFE77xx_RegWrite (fd, 0x00a4, 0x1c);
        AFE77xx_RegWrite (fd, 0x00ab, 0x00);
        AFE77xx_RegWrite (fd, 0x00aa, 0x01);
    }
}

```

```

AFE77xx_RegWrite(fd,0x00a9,0x00);
AFE77xx_RegWrite(fd,0x00a8,0x00);
AFE77xx_RegWrite(fd,0x00af,0x00);
AFE77xx_RegWrite(fd,0x00ae,0x00);
AFE77xx_RegWrite(fd,0x00ad,0x00);
AFE77xx_RegWrite(fd,0x00ac,0x00);
AFE77xx_RegWrite(fd,0x00b3,0x00);
AFE77xx_RegWrite(fd,0x00b2,0x00);
AFE77xx_RegWrite(fd,0x00b1,0x0e);
AFE77xx_RegWrite(fd,0x00b0,0x00);
AFE77xx_RegWrite(fd,0x0193,0x41);
AFE77xx_RegWrite(fd,0x0013,0x00);
waitForMacroAck(fd);
waitForMacroDone(fd);
    }
}
//END: Done Setting the parameters for factory Calibration

//api_factory_calibration_control_done
waitForMacroReady(fd);
AFE77xx_RegWrite(fd,0x0013,0x10);
AFE77xx_RegWrite(fd,0x00a3,0x00);
AFE77xx_RegWrite(fd,0x00a2,0x01);
AFE77xx_RegWrite(fd,0x00a1,0x02);
AFE77xx_RegWrite(fd,0x00a0,0x0f);
AFE77xx_RegWrite(fd,0x00a7,0x00);
AFE77xx_RegWrite(fd,0x00a6,0x00);
AFE77xx_RegWrite(fd,0x00a5,CHAIN_TO_CALIB); // RX CHAIN SEL
AFE77xx_RegWrite(fd,0x00a4,0x1c);
AFE77xx_RegWrite(fd,0x00ab,0x0e);
AFE77xx_RegWrite(fd,0x00aa,0x00);
AFE77xx_RegWrite(fd,0x00a9,0x00);
AFE77xx_RegWrite(fd,0x00a8,0x00);
AFE77xx_RegWrite(fd,0x0193,0x41);
AFE77xx_RegWrite(fd,0x0013,0x00);
waitForMacroAck(fd);
waitForMacroDone(fd);
//END: Completed Calibration and loading the packet into
memory

//START: Reading packet from silicon
//EXTERNAL-ACTION: Do the below reads and save the values
into memory.
U32 noOfChannels=0;
for (int i=0;i<4;i=i+1){
    noOfChannels=noOfChannels+((CHAIN_TO_CALIB>>i)&0x1);
}

AFE77xx_RegWrite(fd,0x0012,0x10);
for(int i=0x0020;i<(0x37+(noOfChannels*241));i=i+1){
    AFE77xx_RegRead(fd,i,&readVal[i-0x20]);
    printf("%d,",readVal[i-0x20]);
}

```

```

}

AFE77xx_RegWrite(fd,0x0012,0x00);

//EXTERNAL-ACTION: End of the reads.
//END: Done reading packet from silicon

//set_factory_calibration_data
waitForMacroReady(fd);
AFE77xx_RegWrite(fd,0x0013,0x10);
AFE77xx_RegWrite(fd,0x00a3,0x00);
AFE77xx_RegWrite(fd,0x00a2,0x00);
AFE77xx_RegWrite(fd,0x00a1,0x00);
AFE77xx_RegWrite(fd,0x00a0,0x02);
AFE77xx_RegWrite(fd,0x0193,0x12);
AFE77xx_RegWrite(fd,0x0013,0x00);
waitForMacroAck(fd);
waitForMacroDone(fd);

AFE77xx_RegWrite(fd,0x0015,0x80);
AFE77xx_RegWrite(fd,0x0550,0x05);
AFE77xx_RegReadWrite(fd,0x0551,00,1,1);
AFE77xx_RegReadWrite(fd,0x0552,28,1,1);
AFE77xx_RegWrite(fd,0x0015,0x00);
AFE77xx_RegWrite(fd,0x0016,0x10);
AFE77xx_RegWrite(fd,0x0040,rxDsaMode[0]);
AFE77xx_RegWrite(fd,0x0016,0x20);
AFE77xx_RegWrite(fd,0x0040,rxDsaMode[1]);
AFE77xx_RegWrite(fd,0x0016,0x00);
AFE77xx_RegWrite(fd,0x0014,0x04);
AFE77xx_RegWrite(fd,0x0121,0x00);
//START: Overriding TDD Pins internally
//"Releasing Pin control on TDD//"
AFE77xx_RegWrite(fd, 0x124,tddPinCtrl[0]);
AFE77xx_RegWrite(fd, 0x126,tddPinCtrl[1]);
AFE77xx_RegWrite(fd, 0x128,tddPinCtrl[2]);
AFE77xx_RegWrite(fd, 0x12A,tddPinCtrl[3]);
AFE77xx_RegWrite(fd, 0x12C,tddPinCtrl[4]);
AFE77xx_RegWrite(fd, 0x12E,tddPinCtrl[5]);
AFE77xx_RegWrite(fd, 0x125,tddPinCtrl[6]);
AFE77xx_RegWrite(fd, 0x127,tddPinCtrl[7]);
AFE77xx_RegWrite(fd, 0x129,tddPinCtrl[8]);
AFE77xx_RegWrite(fd, 0x12B,tddPinCtrl[9]);
AFE77xx_RegWrite(fd, 0x12D,tddPinCtrl[10]);
AFE77xx_RegWrite(fd, 0x12F,tddPinCtrl[11]);
//END: Done overriding TDD Pins internally
//END: Completed RX DSA Calibration
AFE77xx_RegWrite(fd,0x0014,0x00);
return 0;
}

/* TX related functions */

```

```

U32 TxDsaSwapAttn(int fd, U32 ulChan, U32 ulTXDsaA, U32 ulTXDsaB)
{
    U32 ulPageRegValue = 0;
    U32 ulRegValue = 0;

    ulPageRegValue = 0x10 << (ulChan);

    AFE77xx_RegWrite(fd,0x0016,ulPageRegValue);
    AFE77xx_RegWrite(fd,0x0104,ulTXDsaA);
    AFE77xx_RegWrite(fd,0x0124,ulTXDsaB);
    AFE77xx_RegWrite(fd,0x0016,0x00);
    return 0;
}

double GetCurrentTxAttenuation(int fd, U32 ulChan){
    U32 ulRegValue=0;
    double analogAttn,digAttn=0;
    AFE77xx_RegWrite(fd,0x0016,(0x10<<(ulChan>>1)));
    AFE77xx_RegRead(fd,0x0100+((ulChan&1)*0x020),&ulRegValue);
    printf("Current Analog Attenuation for TX Channel %d:%d\r\n",ulChan,ulRegValue);
    analogAttn=ulRegValue;
    AFE77xx_RegRead(fd,0x0101+((ulChan&1)*0x020),&ulRegValue);
    digAttn=(ulRegValue/8.0)-3;
    printf("Current Dig Attenuation for TX Channel %d:%lf\r\n",ulChan,digAttn);
    AFE77xx_RegWrite(fd,0x0016,0x00);
    printf("Current complete Attenuation for TX Channel %d:%lf\r\n",ulChan,(analogAttn+digAttn));
    return (analogAttn+digAttn);
}

U32 TXDsaSwapAttnMacro(int fd, U32 ulChan, U32 ulTXDsaA, U32
ulTXDsaB){
    U32 ulRegValue = 0;

    AFE77xx_RegWrite(fd,0x0013,0x10);
    AFE77xx_RegWrite(fd,0x00a0,0x13);
    AFE77xx_RegWrite(fd,0x00a1,0x04+((ulChan&1)*0x20));
    AFE77xx_RegWrite(fd,0x00a2,0x01);
    AFE77xx_RegWrite(fd,0x00a3,0x00);
    if ((ulChan>>1)==0){
        AFE77xx_RegWrite(fd,0x00a4,0xae);
    }
    else{
        AFE77xx_RegWrite(fd,0x00a4,0xaf);
    }
    AFE77xx_RegWrite(fd,0x00a5,ulTXDsaB);
    AFE77xx_RegWrite(fd,0x0193,0x02);
    AFE77xx_RegWrite(fd,0x0013,0x00);
    return 0;
}

```

```

U32 txSigGenTone(int fd, int channelNo, int enable, double power,
double frequency){
    /*
        channelNo = 0- TXA, 1-TXB, 2-TXC, 3-TXD
        enable     =0-don't enable. 1- Enable.
        power      = power in dBfs
        frequency  = base band offset frequency
    */
    double frequencyFactor;
    U16 frequencyWord;

    frequencyFactor=491.52/128;      /*Set the bandwidth
appropriately*/

    if (frequency<0){
        frequencyWord=ceil(-512*frequency/frequencyFactor);
    }
    else{
        frequencyWord=65536-ceil(512
*frequency/frequencyFactor);
    }
    printf("Frequency Being Programmed: %f
\n",frequencyWord*frequencyFactor/512.0);
    AFE77xx_RegWrite(fd,0x13,0x20);
    if (channelNo==0){
        AFE77xx_RegReadWrite(fd, 0x13d, (enable*3)<<2, 2, 3);
        AFE77xx_RegReadWrite(fd, 0x013a, (int)(ceil(round(16
*pow(10.0, power/20.0))))<<1, 1, 6);

        AFE77xx_RegReadWrite(fd, 0x00122,0x01,0,0);
        AFE77xx_RegReadWrite(fd,
0x0011d,frequencyWord>>8,0,7);
        AFE77xx_RegReadWrite(fd, 0x0011c,frequencyWord&
0xff,0,7);
        AFE77xx_RegReadWrite(fd, 0x00122,0x00,0,0);
        AFE77xx_RegReadWrite(fd, 0x00122,0x01,0,0);
    }
    else if (channelNo==1){
        AFE77xx_RegReadWrite(fd, 0x13d, (enable*3)<<4, 4, 5);
        AFE77xx_RegReadWrite(fd, 0x13c, ceil(round(16
*pow(10.0, power/20.0))), 0, 5);

        AFE77xx_RegReadWrite(fd, 0x012a,0x01,0,0);
        AFE77xx_RegReadWrite(fd, 0x0125,frequencyWord>>8,0,7);
        AFE77xx_RegReadWrite(fd, 0x0124,frequencyWord&
0xff,0,7);
        AFE77xx_RegReadWrite(fd, 0x012a,0x00,0,0);
        AFE77xx_RegReadWrite(fd, 0x012a,0x01,0,0);
    }
    else if (channelNo==2){
        AFE77xx_RegReadWrite(fd, 0x13e, (enable*3)<<3, 3, 4);

```



```

AFE77xx_RegReadWrite(fd, 0x132, (int) (ceil(round(16
*pow(10.0, power/20.0))))<<1, 1, 6);

AFE77xx_RegReadWrite(fd, 0x0132, 0x01, 0, 0);
AFE77xx_RegReadWrite(fd, 0x012d, frequencyWord>>8, 0, 7);
AFE77xx_RegReadWrite(fd, 0x012c, frequencyWord&
0xff, 0, 7);
AFE77xx_RegReadWrite(fd, 0x0132, 0x00, 0, 0);
AFE77xx_RegReadWrite(fd, 0x0132, 0x01, 0, 0);
}
else if (channelNo==3){
AFE77xx_RegReadWrite(fd, 0x13f, (enable*3)<<2, 2, 3);
AFE77xx_RegReadWrite(fd, 0x12a, (int) (ceil(round(16
*pow(10.0, power/20.0))))<<1, 1, 6);

AFE77xx_RegReadWrite(fd, 0x013a, 0x01, 0, 0);
AFE77xx_RegReadWrite(fd, 0x0135, frequencyWord>>8, 0, 7);
AFE77xx_RegReadWrite(fd, 0x0134, frequencyWord&
0xff, 0, 7);
AFE77xx_RegReadWrite(fd, 0x013a, 0x00, 0, 0);
AFE77xx_RegReadWrite(fd, 0x013a, 0x01, 0, 0);
}
AFE77xx_RegReadWrite(fd, 0x110, ((1<<channelNo)*enable), channelNo, channelNo);
AFE77xx_RegReadWrite(fd, 0x13f, 0x70, 4, 6);
AFE77xx_RegWrite(fd, 0x13, 0x00);
}

int loadTxDsaPacket(int fd, U8 *array, U16 arraySize){
/*Pass the Array saved during the DSA calibration and it's
size. */
U16 i;
if (arraySize==0){
return 0;
}
AFE77xx_RegWrite(fd, 0x12, 0x10);
for (i = 0; i < arraySize; ++i) {
AFE77xx_RegWrite(fd, 0x20+i, array[i]);
}
AFE77xx_RegWrite(fd, 0x12, 0x00);
waitForMacroReady(fd);
AFE77xx_RegWrite(fd, 0x13, 0x10);
AFE77xx_RegWrite(fd, 0x00a3, 0x00);
AFE77xx_RegWrite(fd, 0x00a2, 0x00);
AFE77xx_RegWrite(fd, 0x00a1, 0x00);
AFE77xx_RegWrite(fd, 0x00a0, 0x01);
AFE77xx_RegWrite(fd, 0x0193, 0x12);
AFE77xx_RegWrite(fd, 0x0192, 0x00);
AFE77xx_RegWrite(fd, 0x0191, 0x00);
AFE77xx_RegWrite(fd, 0x0190, 0x00);
AFE77xx_RegWrite(fd, 0x13, 0x00);
waitForMacroDone(fd);

```

```

}

U32 setLowIfNcoForAllTxFb(int fd, unsigned long int ulNcoFreq){
    /*ulNcoFreq is frequency in KHz*/
    unsigned long int ncoWriteVal;
    U32 ulRegValue1 = 0;
    U32 ulRegValue2 = 0;
    U32 ulRegValue3 = 0;

    AFE77xx_RegWrite(fd, 0x0016, 0x40);
    AFE77xx_RegRead(fd, 0x015b, &ulRegValue1);
    AFE77xx_RegWrite(fd, 0x0016, 0x00);

    AFE77xx_RegWrite(fd, 0x0010, 0x01);
    AFE77xx_RegRead(fd, 0x010B, &ulRegValue2);
    AFE77xx_RegRead(fd, 0x010D, &ulRegValue3);
    AFE77xx_RegWrite(fd, 0x0010, 0x00);

    if (ulNcoFreq>0){
        ncoWriteVal=4294230016+ulNcoFreq;
    }
    else{
        ncoWriteVal=4294967296+ulNcoFreq;
    }
    //TX Freeze Start
    AFE77xx_RegWrite(fd, 0x14, 0x04);
    AFE77xx_RegWrite(fd, 0x108, 0x01);
    AFE77xx_RegWrite(fd, 0x109, 0x04);
    AFE77xx_RegWrite(fd, 0x14, 0x00);

    AFE77xx_RegWrite(fd, 0x13, 0x10);
    AFE77xx_RegWrite(fd, 0xA3, 0x00);
    AFE77xx_RegWrite(fd, 0xA2, 0x00);
    AFE77xx_RegWrite(fd, 0xA1, 0xFF);
    AFE77xx_RegWrite(fd, 0xA0, 0x06);
    AFE77xx_RegWrite(fd, 0x193, 0x35);
    sysUsDelay(20000);
    AFE77xx_RegWrite(fd, 0x13, 0x00);
    //TX Freeze End

    AFE77xx_RegWrite(fd, 0x0014, 0x04);
    AFE77xx_RegWrite(fd, 0x04fd, 0x00);
    AFE77xx_RegWrite(fd, 0x04fc, 0x1e);
    AFE77xx_RegWrite(fd, 0x0014, 0x00);
    AFE77xx_RegWrite(fd, 0x0013, 0x02);
    AFE77xx_RegWrite(fd, 0x3457, (ncoWriteVal>>24)&0xff);
    AFE77xx_RegWrite(fd, 0x3456, (ncoWriteVal>>16)&0xff);
    AFE77xx_RegWrite(fd, 0x3455, (ncoWriteVal>>8)&0xff);
    AFE77xx_RegWrite(fd, 0x3454, ncoWriteVal&0xff);
    AFE77xx_RegWrite(fd, 0x345b, (ncoWriteVal>>24)&0xff);
    AFE77xx_RegWrite(fd, 0x345a, (ncoWriteVal>>16)&0xff);
    AFE77xx_RegWrite(fd, 0x3459, (ncoWriteVal>>8)&0xff);

```

```

AFE77xx_RegWrite (fd, 0x3458, ncoWriteVal&0xff);
AFE77xx_RegWrite (fd, 0x346f, (ncoWriteVal>>24) &0xff);
AFE77xx_RegWrite (fd, 0x346e, (ncoWriteVal>>16) &0xff);
AFE77xx_RegWrite (fd, 0x346d, (ncoWriteVal>>8) &0xff);
AFE77xx_RegWrite (fd, 0x346c, ncoWriteVal&0xff);
AFE77xx_RegWrite (fd, 0x3473, (ncoWriteVal>>24) &0xff);
AFE77xx_RegWrite (fd, 0x3472, (ncoWriteVal>>16) &0xff);
AFE77xx_RegWrite (fd, 0x3471, (ncoWriteVal>>8) &0xff);
AFE77xx_RegWrite (fd, 0x3470, ncoWriteVal&0xff);
AFE77xx_RegWrite (fd, 0x3468, 0x03);
AFE77xx_RegWrite (fd, 0x0013, 0x00);
waitForMacroReady();
AFE77xx_RegWrite (fd, 0x0013, 0x10);
AFE77xx_RegWrite (fd, 0x00a3, 0x00);
AFE77xx_RegWrite (fd, 0x00a2, 0x01);
AFE77xx_RegWrite (fd, 0x00a1, 0x01);
AFE77xx_RegWrite (fd, 0x00a0, 0x00);
AFE77xx_RegWrite (fd, 0x00a7, 0x00);
AFE77xx_RegWrite (fd, 0x00a6, 0x00);
AFE77xx_RegWrite (fd, 0x00a5, 0x08);
AFE77xx_RegWrite (fd, 0x00a4, 0x00);
AFE77xx_RegWrite (fd, 0x0193, 0x31);
AFE77xx_RegWrite (fd, 0x0192, 0x00);
AFE77xx_RegWrite (fd, 0x0191, 0x00);
AFE77xx_RegWrite (fd, 0x0190, 0x00);
AFE77xx_RegWrite (fd, 0x0013, 0x00);
waitForMacroDone();

AFE77xx_RegWrite (fd, 0x0016, 0x40);
AFE77xx_RegWrite (fd, 0x015b, ulRegValue1);
AFE77xx_RegWrite (fd, 0x0016, 0x80);
AFE77xx_RegWrite (fd, 0x015b, ulRegValue1);

AFE77xx_RegWrite (fd, 0x0016, 0x00);

AFE77xx_RegWrite (fd, 0x0010, 0x03);
AFE77xx_RegWrite (fd, 0x010B, ulRegValue2);
AFE77xx_RegWrite (fd, 0x010D, ulRegValue3);
AFE77xx_RegWrite (fd, 0x0010, 0x00);

AFE77xx_RegWrite (fd, 0x0015, 0x80);
#ifdef TIENV
AFE77xx_RegWrite (fd, 0x01d4, 0x00);
AFE77xx_RegWrite (fd, 0x0374, 0x01);
#else
AFE77xx_RegWrite (fd, 0x01d4, 0x01);
AFE77xx_RegWrite (fd, 0x0374, 0x00);
#endif
AFE77xx_RegWrite (fd, 0x0015, 0x00);
sysUsDelay(100);

AFE77xx_RegWrite (fd, 0x0014, 0x04);

```

```

AFE77xx_RegWrite(fd,0x04e0,0x00);
AFE77xx_RegWrite(fd,0x0014,0x10);
AFE77xx_RegWrite(fd,0x005d,0xe0);

sysUsDelay(100);

AFE77xx_RegWrite(fd,0x005d,0xc0);
AFE77xx_RegWrite(fd,0x0014,0x00);

AFE77xx_RegWrite(fd,0x0015,0x80);
AFE77xx_RegWrite(fd,0x01d4,0x00);
AFE77xx_RegWrite(fd,0x0374,0x00);
AFE77xx_RegWrite(fd,0x0015,0x00);
sysUsDelay(100);
AFE77xx_RegWrite(fd,0x0014,0x04);
AFE77xx_RegWrite(fd,0x04fd,0x07);
AFE77xx_RegWrite(fd,0x04fc,0xff);
AFE77xx_RegWrite(fd,0x0014,0x00);

// "TX UnFreeze Start"
AFE77xx_RegWrite(fd, 0x13,0x10);
AFE77xx_RegWrite(fd, 0xA3,0x00);
AFE77xx_RegWrite(fd, 0xA2,0x00);
AFE77xx_RegWrite(fd, 0xA1,0xFF);
AFE77xx_RegWrite(fd, 0xA0,0x07);
AFE77xx_RegWrite(fd, 0x193,0x35);
sysUsDelay(10000);
AFE77xx_RegWrite(fd, 0x13,0x00);

AFE77xx_RegWrite(fd, 0x16,0x08);
AFE77xx_RegWrite(fd, 0x100,0x00);
sysUsDelay(1000);
AFE77xx_RegWrite(fd, 0x100,0x01);
AFE77xx_RegWrite(fd, 0x16,0x00);

sysUsDelay(1000);

AFE77xx_RegWrite(fd, 0x12,0x10);
AFE77xx_RegWrite(fd, 0x20,0x8A);
AFE77xx_RegWrite(fd, 0x21,0x13);
AFE77xx_RegWrite(fd, 0x22,0x01);
AFE77xx_RegWrite(fd, 0x23,0x20);
AFE77xx_RegWrite(fd, 0x24,0x01);
AFE77xx_RegWrite(fd, 0x25,0x01);
AFE77xx_RegWrite(fd, 0x26,0x01);
AFE77xx_RegWrite(fd, 0x27,0x01);
AFE77xx_RegWrite(fd, 0x28,0x02);
AFE77xx_RegWrite(fd, 0x29,0x00);
AFE77xx_RegWrite(fd, 0x2A,0x00);
AFE77xx_RegWrite(fd, 0x2B,0x00);
AFE77xx_RegWrite(fd, 0x12,0x00);
AFE77xx_RegWrite(fd, 0x13,0x10);

```

```

AFE77xx_RegWrite(fd, 0xA3,0x00);
AFE77xx_RegWrite(fd, 0xA2,0x0A);
AFE77xx_RegWrite(fd, 0xA1,0x06);
AFE77xx_RegWrite(fd, 0xA0,0x10);
AFE77xx_RegWrite(fd, 0x193,0x35);
AFE77xx_RegWrite(fd, 0x192,0x00);
AFE77xx_RegWrite(fd, 0x191,0x00);
AFE77xx_RegWrite(fd, 0x190,0x00);
AFE77xx_RegWrite(fd, 0x13,0x00);
sysUsDelay(100000);
// "TX UnFreeze End//"

//Change TX QEC mode into GPIO mode
AFE77xx_RegWrite(fd,0x14,0x04);
AFE77xx_RegWrite(fd,0x108,0x00);
AFE77xx_RegWrite(fd,0x14,0x00);
}

U32 configSrBlock(int fd,int chNo,int GainStepSize,int
AttnStepSize,int AmplUpdateCycles,int threshold, int
rampDownWaitTime, int rampUpWaitTime){
/*
chNo=0,1,2,3
GainStepSize= This is the ramp step size while gaining up
(actualSampleStep=GainStepSize*(last good sample)/1024)
AttnStepSize = similar to GainStepSize while ramping down.
AmplUpdateCycles = This is the number of 368.64M clocks
after which the step will be updated.
threshold = This is the
percentage of the theshold*65536
rampDownWaitTime= This value clocks of Sampling rate/8 is
the time between the SR trigger and the ramp down starts. This
should be <=20
rampUpWaitTime = This value clocks of Sampling
rate/8 is the time between the SR trigger release and the ramp Up
starts. This should be >=30
*/
AFE77xx_RegWrite(fd,0x0010,0x01<<(chNo>>1));
AFE77xx_RegWrite(fd,0x0225+(0x200*(chNo&
1)),rampDownWaitTime>>8);
AFE77xx_RegWrite(fd,0x0224+(0x200*(chNo&
1)),rampDownWaitTime&0xff);
AFE77xx_RegWrite(fd,0x0227+(0x200*(chNo&
1)),rampUpWaitTime>>8);
AFE77xx_RegWrite(fd,0x0226+(0x200*(chNo&1)),rampUpWaitTime&
0xff);
AFE77xx_RegWrite(fd,0x0247+(0x200*(chNo&1)),0x0f);
AFE77xx_RegWrite(fd,0x0246+(0x200*(chNo&1)),0x9c);
AFE77xx_RegWrite(fd,0x02a0+(0x200*(chNo&1)),0x01);
AFE77xx_RegWrite(fd,0x022a+(0x200*(chNo&1)),0x01);
AFE77xx_RegWrite(fd,0x023b+(0x200*(chNo&
1)),GainStepSize>>8);

```

```

AFE77xx_RegWrite(fd,0x023a+(0x200*(chNo&1)),GainStepSize&
0xff);
AFE77xx_RegWrite(fd,0x0233+(0x200*(chNo&
1)),AttnStepSize>>8);
AFE77xx_RegWrite(fd,0x0232+(0x200*(chNo&1)),AttnStepSize&
0xff);
AFE77xx_RegWrite(fd,0x0228+(0x200*(chNo&
1)),AmplUpdateCycles);
AFE77xx_RegWrite(fd,0x0210+(0x200*(chNo&1)),0x03);
AFE77xx_RegWrite(fd,0x02a3+(0x200*(chNo&1)),threshold>>8);
AFE77xx_RegWrite(fd,0x02a2+(0x200*(chNo&1)),threshold&0xff);
AFE77xx_RegWrite(fd,0x022f+(0x200*(chNo&1)),0x00);
AFE77xx_RegWrite(fd,(0x022e)+(0x200*(chNo&1)),0x13);
AFE77xx_RegWrite(fd,0x0210+(0x200*(chNo&1)),0x03);
AFE77xx_RegWrite(fd,0x0010,0x00);
}

```

```

U32 freezeTxIqmc(int fd,int freeze,int txCh){
/* freeze:- 1:Freeze IQ mismatch estimation,0:Unfreeze IQ
mismatch estimation
txCh:- 1:TX AB, 2:TX CD, 3:TX ABCD
*/
waitForMacroReady(fd);
AFE77xx_RegWrite(fd,0x0013,0x10);
AFE77xx_RegWrite(fd,0x00a3,0x00);
AFE77xx_RegWrite(fd,0x00a2,0x00);
AFE77xx_RegWrite(fd,0x00a1,0xf0+((txCh&2)<<2)+((txCh&2)
<<1)+((txCh&1)<<1)+(txCh&1));
if (freeze==1){
AFE77xx_RegWrite(fd,0x00a0,0x13);
}
else{
AFE77xx_RegWrite(fd,0x00a0,0x14);
}
AFE77xx_RegWrite(fd,0x0193,0x35);
AFE77xx_RegWrite(fd,0x0192,0x00);
AFE77xx_RegWrite(fd,0x0191,0x00);
AFE77xx_RegWrite(fd,0x0190,0x00);
waitForMacroDone(fd);
}

```

```

U32 freezeTxLol(int fd,int freeze,int txCh){
/* freeze:- 1:Freeze LO leakage estimation,0:Unfreeze LO
leakage estimation
txCh:- 1:TX AB, 2:TX CD, 3:TX ABCD
*/
waitForMacroReady(fd);
AFE77xx_RegWrite(fd,0x0013,0x10);
AFE77xx_RegWrite(fd,0x00a3,0x00);
AFE77xx_RegWrite(fd,0x00a2,0x00);
AFE77xx_RegWrite(fd,0x00a1,0xf0+((txCh&2)<<2)+((txCh&2)
<<1)+((txCh&1)<<1)+(txCh&1));

```

```

    if (freeze==1){
        AFE77xx_RegWrite (fd,0x00a0,0x15);
    }
    else{
        AFE77xx_RegWrite (fd,0x00a0,0x16);
    }
    AFE77xx_RegWrite (fd,0x0193,0x35);
    AFE77xx_RegWrite (fd,0x0192,0x00);
    AFE77xx_RegWrite (fd,0x0191,0x00);
    AFE77xx_RegWrite (fd,0x0190,0x00);
    waitForMacroDone (fd);
}

int txIqmcEstimatorStartDelay(int fd,int delay){
    /*    delay: delay in DC_PLL_OUT/4 cycles. Default 750.
        Eg: for delay=750 and DC_PLL_OUT=2949.12MHz, actual
time delay=750*(4/2949.12)uSec=1.017uSec
    */
    AFE77xx_RegWrite (fd,0x0014,0x04);
    AFE77xx_RegWrite (fd,0x053F, (delay>>8));
    AFE77xx_RegWrite (fd,0x053E, (delay&0xff));
    AFE77xx_RegReadWrite (fd,0x0540,0x1,0,0);
    AFE77xx_RegWrite (fd,0x0014,0x00);
}

U32 getSrAlarm(int fd,int chN0){
    U32 readVal=0;
    AFE77xx_RegWrite (fd,0x0015, 0x80);
    AFE77xx_RegRead (fd,0x01f8, &readVal);
    readVal=(readVal>>1)&0xf;
    printf("TXA PAP: %d, TXB PAP: %d ",(readVal&
1),((readVal>>1)&1));
    AFE77xx_RegRead (fd,0x03c0, &readVal);
    printf("TXC PAP: %d, TXD PAP: %d", ((readVal>>3)&
1),((readVal>>4)&1));
    AFE77xx_RegWrite (fd,0x0015, 0x00);
}

U32 setTxDsaUpdateMode(int fd, int immediateUpdate, int delay){
    /*
        immediateUpdate is 0 for changing gain on TDD and 1 for
using the gain smoothening.
        delay*375ns will be the step time for gain smoothening.
Ignore in TDD mode*/
    AFE77xx_RegWrite (fd,0x013, 0x10);
    AFE77xx_RegWrite (fd,0x0A3, 0x00);
    AFE77xx_RegWrite (fd,0x0A2, 0x1e);
    AFE77xx_RegWrite (fd,0x0A1, delay);
    AFE77xx_RegWrite (fd,0x0A0, immediateUpdate);
    AFE77xx_RegWrite (fd,0x193, 0x2D);
    AFE77xx_RegWrite (fd,0x192, 0x00);
    AFE77xx_RegWrite (fd,0x191, 0x00);
}

```

```

AFE77xx_RegWrite(fd,0x190, 0x00);
AFE77xx_RegWrite(fd,0x013, 0x00);
waitForMacroDone();
}

U32 getTxDsaModeConfig(int fd){
    U32 regVal=0;

    AFE77xx_RegWrite(fd,0x0013,0x10);
    regVal=AFE77xx_RegRead(fd,0x0140,&regVal);
    AFE77xx_RegWrite(fd,0x0140,regVal&0xfe);
    regVal=AFE77xx_RegRead(fd,0x0144,&regVal);
    regVal=0x8+(regVal&0xE3);
    AFE77xx_RegWrite(fd,0x0144,regVal);
    AFE77xx_RegWrite(fd,0x0013,0x00);
    AFE77xx_RegWrite(fd,0x0013,0x02);
    regVal=AFE77xx_RegRead(fd,0x085,&regVal)&1;
    AFE77xx_RegWrite(fd,0x0013,0x00);
    if (regVal==0){
        printf("TX DSA gain change happens at TDD toggle.");
    }
    else{
        printf("TX DSA gain change happens immediately.");
    }
}

U32 setTxDsaAttn(int fd, U16 dsaSetting, int chainSel){
    /*dsaSetting should be 8x the required DSA attenuation.
    Range from -24 to 479 corresponding to -3dB to 59.875dB in
    steps of 0.125dB. Negative value indicates digital gain.
    chainSel is 4 bit with select per channel.
    For example, for writing the setting to A and C channels,
    chainSel=0x05
    */

    AFE77xx_RegWrite(fd,0x013, 0x10);
    AFE77xx_RegWrite(fd,0x0A3, 0x00);
    AFE77xx_RegWrite(fd,0x0A2, chainSel);
    AFE77xx_RegWrite(fd,0x0A1, dsaSetting>>8);
    AFE77xx_RegWrite(fd,0x0A0, dsaSetting&0xff);
    AFE77xx_RegWrite(fd,0x193, 0x2E);
    AFE77xx_RegWrite(fd,0x192, 0x00);
    AFE77xx_RegWrite(fd,0x191, 0x00);
    AFE77xx_RegWrite(fd,0x190, 0x00);
    AFE77xx_RegWrite(fd,0x013, 0x00);
    waitForMacroDone();
}

float txPowDet(int fd,int ch){
    /*ch= 0:A, 1:B, 2:C, 3:D*/
    U32 readVal,rdVal;
    float txPow;

```



```

    int ulRegOffset=0x200*(ch&1);
    AFE77xx_RegWrite(fd,0x10,0x01<<(ch>>1));
    AFE77xx_RegWrite(fd,0x205+ulRegOffset,0x01);
    AFE77xx_RegWrite(fd,0x207+ulRegOffset,0x00);
    AFE77xx_RegWrite(fd,0x206+ulRegOffset,0x03);
    rdVal=AFE77xx_RegRead(fd,0x0210+ulRegOffset,&rdVal);
    AFE77xx_RegWrite(fd,0x210+ulRegOffset,0x03);
    AFE77xx_RegWrite(fd,0x204+ulRegOffset,0x00);
    AFE77xx_RegWrite(fd,0x204+ulRegOffset,0x01);
    AFE77xx_RegWrite(fd,0x204+ulRegOffset,0x00);
    readVal=AFE77xx_RegRead(fd,0x020A+ulRegOffset,&readVal);
    readVal=(readVal<<8)+AFE77xx_RegRead(fd,0x0209+ulRegOffset,
&readVal);
    readVal=(readVal<<8)+AFE77xx_RegRead(fd,0x0208+ulRegOffset,
&readVal);
    AFE77xx_RegWrite(fd,0x210+ulRegOffset,rdVal);
    AFE77xx_RegWrite(fd,0x10,0x00);

    txPow=20*log10(128*sqrt(readVal)/32768.);
    printf("TX Power=%lfdBFS\n",txPow);
    return txPow;
}

/* FB related functions */
U32 FbDsaAttn(int fd, U32 ulChan, U32 ulFbDsa){
    U32 ulRegValue=0;
    AFE77xx_RegWrite(fd,0x0016,(0x10<<ulChan));
    AFE77xx_RegWrite(fd,0x0150,ulFbDsa);
    AFE77xx_RegWrite(fd,0x0016,0x00);
    return 0;
}

U32 GetCurrentFbAttenuation(int fd, U32 ulChan){
    U32 ulRegValue=0;
    AFE77xx_RegWrite(fd,0x0016,(0x10<<ulChan));
    AFE77xx_RegRead(fd,0x0150,&ulRegValue);
    printf("Current Attenuation for FB Channel %d:%d\r
\n",ulChan,ulRegValue);
    AFE77xx_RegWrite(fd,0x0016,0x00);
    return ulRegValue;
}

U32 FbNCO(int fd, double ulNcoFreq,U8 ucFbChan){
    /*
        ulNcoFreq is the frequency in MHz with raster as 1KHz.
        ucFbChan= 0:FBAB,1:FBCD,2:FBAB band1 NCO,3:FBCD band1
    NCO
    */
    U32 freqWord;
    U32 ulRegValue = 0;
    freqWord=(U32)(ulNcoFreq*1000);

```

```

AFE77xx_RegWrite (fd, 0x0016, 0x40);
AFE77xx_RegRead (fd, 0x15b, &ulRegValue);
AFE77xx_RegWrite (fd, 0x0016, 0x0);

AFE77xx_RegWrite (fd, 0x0014, 0x04);
AFE77xx_RegWrite (fd, 0x04fd, 0x00);
AFE77xx_RegWrite (fd, 0x04fc, 0x06);
AFE77xx_RegWrite (fd, 0x0014, 0x00);
AFE77xx_RegWrite (fd, 0x0013, 0x02);

if (ucFbChan < 2) {
    AFE77xx_RegWrite (fd, 0x343c + (ucFbChan * 8), freqWord &
0xff);
    AFE77xx_RegWrite (fd, 0x343d + (ucFbChan * 8), (freqWord >> 8) &
0xff);
    AFE77xx_RegWrite (fd, 0x343e + (ucFbChan *
8), (freqWord >> 16) & 0xff);
    AFE77xx_RegWrite (fd, 0x343f + (ucFbChan *
8), (freqWord >> 24) & 0xff);
    AFE77xx_RegWrite (fd, 0x0013, 0x00);
}
else if (ucFbChan < 4) {
    AFE77xx_RegWrite (fd, 0x3440 + ((ucFbChan & 1) * 8), freqWord &
0xff);
    AFE77xx_RegWrite (fd, 0x3441 + ((ucFbChan & 1) *
8), (freqWord >> 8) & 0xff);
    AFE77xx_RegWrite (fd, 0x3442 + ((ucFbChan & 1) *
8), (freqWord >> 16) & 0xff);
    AFE77xx_RegWrite (fd, 0x3443 + ((ucFbChan & 1) *
8), (freqWord >> 24) & 0xff);
    AFE77xx_RegWrite (fd, 0x0013, 0x00);
}

waitForMacroReady ();

AFE77xx_RegWrite (fd, 0x0013, 0x10);
AFE77xx_RegWrite (fd, 0x00a3, 0x00);
AFE77xx_RegWrite (fd, 0x00a2, 0x01);
AFE77xx_RegWrite (fd, 0x00a1, 0x00);
AFE77xx_RegWrite (fd, 0x00a0, 0x00);
AFE77xx_RegWrite (fd, 0x00a7, 0x00);
AFE77xx_RegWrite (fd, 0x00a6, 0x00);
AFE77xx_RegWrite (fd, 0x00a5, 0x00);
AFE77xx_RegWrite (fd, 0x00a4, 0x00);
AFE77xx_RegWrite (fd, 0x0193, 0x31);
AFE77xx_RegWrite (fd, 0x0192, 0x00);
AFE77xx_RegWrite (fd, 0x0191, 0x00);
AFE77xx_RegWrite (fd, 0x0190, 0x00);
AFE77xx_RegWrite (fd, 0x0013, 0x00);

waitForMacroDone ();

```

```

AFE77xx_RegWrite (fd, 0x0016, 0x40);
AFE77xx_RegReadWrite (fd, 0x015b, ulRegValue, 0, 2);
AFE77xx_RegWrite (fd, 0x0016, 0x80);
AFE77xx_RegReadWrite (fd, 0x015b, ulRegValue, 0, 2);

AFE77xx_RegWrite (fd, 0x0016, 0x00);

AFE77xx_RegWrite (fd, 0x0015, 0x80);
#ifdef TIENV
AFE77xx_RegWrite (fd, 0x01d4, 0x00);
AFE77xx_RegWrite (fd, 0x0374, 0x01);
#else
AFE77xx_RegWrite (fd, 0x01d4, 0x01);
AFE77xx_RegWrite (fd, 0x0374, 0x00);
#endif
AFE77xx_RegWrite (fd, 0x0015, 0x00);
sysUsDelay (100);

AFE77xx_RegWrite (fd, 0x0014, 0x04);
AFE77xx_RegWrite (fd, 0x04e0, 0x00);
AFE77xx_RegWrite (fd, 0x0014, 0x10);
AFE77xx_RegWrite (fd, 0x005d, 0xe0);

sysUsDelay (100);

AFE77xx_RegWrite (fd, 0x005d, 0xc0);
AFE77xx_RegWrite (fd, 0x0014, 0x00);

AFE77xx_RegWrite (fd, 0x0015, 0x80);
AFE77xx_RegWrite (fd, 0x01d4, 0x00);
AFE77xx_RegWrite (fd, 0x0374, 0x00);
AFE77xx_RegWrite (fd, 0x0015, 0x00);
sysUsDelay (100);

AFE77xx_RegWrite (fd, 0x0014, 0x04);
AFE77xx_RegWrite (fd, 0x04fd, 0x07);
AFE77xx_RegWrite (fd, 0x04fc, 0xff);
AFE77xx_RegWrite (fd, 0x0014, 0x00);
}

U32 readFbNco (int fd, int chNo) {
    U32 regVal=0;
    U32 fbNco=0;
    U32 fbNcoBand1=0;
    U32 temp=0;
    AFE77xx_RegWrite (fd, 0x0013, 0x02);
    fbNco=AFE77xx_RegRead (fd, 0x343c+ (chNo*8) , &temp);
    fbNco=fbNco+ ((AFE77xx_RegRead (fd, (0x343d+ (chNo*8) ) , &temp)
<<8);
    fbNco=fbNco+ ((AFE77xx_RegRead (fd, (0x343e + (chNo*8) ) , &temp)
<<16);
    fbNco=fbNco+ ((AFE77xx_RegRead (fd, (0x343f+ (chNo*8) ) , &temp)

```

```

<<24);
AFE77xx_RegWrite(fd,0x0013,0x00);
printf("Frequency in KHz: %d\n",fbNco);
// checking NCO BAND1
AFE77xx_RegWrite(fd,0x0014,0x04);
AFE77xx_RegRead(fd,0x0114,&regVal);
AFE77xx_RegRead(fd,0x0115,&temp);
AFE77xx_RegWrite(fd,0x0014,0x00);
if((chNo==0&&regVal==1)|| (chNo==1&&temp==1)){
    AFE77xx_RegWrite(fd,0x0013,0x02);
    fbNcoBand1=AFE77xx_RegRead(fd,0x3440+(chNo*8),&temp);
    fbNcoBand1=fbNcoBand1+((AFE77xx_RegRead(fd,(0x3441
+ (chNo*8)),&temp))<<8);
    fbNcoBand1=fbNcoBand1+((AFE77xx_RegRead(fd,(0x3442
+ (chNo*8)),&temp))<<16);
    fbNcoBand1=fbNcoBand1+((AFE77xx_RegRead(fd,(0x3443
+ (chNo*8)),&temp))<<24);
    AFE77xx_RegWrite(fd,0x0013,0x00);
    printf("NCO band1 Frequency in KHz: %d\n",fbNcoBand1);
}
return fbNco;
}

U32 fbDataCapture(fd, samples){
    /* to capture data from FB inside the AFE. maximum samples
are 1024 */
    int done_val =0;
    int count = 0;
    int data_i[samples];
    int data_q[samples];
    int tempVal = 0;

    if(samples> 1024){
        printf("maximum samples allowed is 1024 \n");
        return 0;
    }
    AFE77xx_RegWrite(fd, 0x0016, 0x40);
    AFE77xx_RegReadWrite(fd, 0x0268, 0x00, 0, 0);
    AFE77xx_RegReadWrite(fd, 0x016F, 0x00, 0, 2);
    AFE77xx_RegWrite(fd, 0x0016, 0x00);

    AFE77xx_RegWrite(fd, 0x0013, 0x20);
    int retVal1 = AFE77xx_RegRead(fd, 0x0294,&tempVal);
    int retVal2 = AFE77xx_RegRead(fd, 0x0295,&tempVal);
    AFE77xx_RegWrite(fd, 0x0294, (retVal1&0xea));
    AFE77xx_RegWrite(fd, 0x0295, (retVal2&0xbf));

    AFE77xx_RegReadWrite(fd, 0x009E, 0x01, 0, 0);
    AFE77xx_RegReadWrite(fd, 0x00A4, 0x10, 4, 4);
    AFE77xx_RegReadWrite(fd, 0x00B4, 0x40, 6, 6);
    AFE77xx_RegWrite(fd, 0x00C3, 0x7F);
    AFE77xx_RegWrite(fd, 0x00C2, 0xFF);

```

```

AFE77xx_RegReadWrite(fd, 0x00C1, 0x07<<2, 2, 5);
AFE77xx_RegReadWrite(fd, 0x00C6, 0x07<<3, 3, 6);

AFE77xx_RegWrite(fd, 0x00C9, 0x3F);
AFE77xx_RegWrite(fd, 0x00C8, 0xFF);
AFE77xx_RegWrite(fd, 0x008E, 0x3F);
AFE77xx_RegWrite(fd, 0x008D, 0xFF);
AFE77xx_RegWrite(fd, 0x0092, 0x3F);
AFE77xx_RegWrite(fd, 0x0091, 0xFF);
AFE77xx_RegWrite(fd, 0x00D9, 0x3F);
AFE77xx_RegWrite(fd, 0x00D8, 0xFF);

AFE77xx_RegReadWrite(fd, 0x00BA, 0x00, 0, 0);
AFE77xx_RegReadWrite(fd, 0x00BB, 0x00, 0, 0);

AFE77xx_RegReadWrite(fd, 0x0095, 0x01, 0, 0);
AFE77xx_RegReadWrite(fd, 0x0095, 0x00, 0, 0);

AFE77xx_RegReadWrite(fd, 0x00F3, 0x01, 0, 0);
AFE77xx_RegReadWrite(fd, 0x00F3, 0x00, 0, 0);

AFE77xx_RegReadWrite(fd, 0x00A8, 0x0A, 0, 3);
AFE77xx_RegWrite(fd, 0x00CC, 0x00);
AFE77xx_RegReadWrite(fd, 0x00BF, 0x01, 0, 0);

AFE77xx_RegReadWrite(fd, 0x0090, 0x02, 0, 3);

printf("trigger \n");
AFE77xx_RegReadWrite(fd, 0x008C, 0x01, 0, 0);
AFE77xx_RegReadWrite(fd, 0x008C, 0x00, 0, 0);
sysUsDelay(3000);

while(count<0xff){
    AFE77xx_RegRead(fd, 0x00DC, &done_val);
    if((done_val&0x04) == 0x04){
        printf("done \n");
        break;
    }
    sysUsDelay(1000);
    count++;
}
if(count>=0xff){
    printf("capture not done\n");
    AFE77xx_RegWrite(fd, 0x0013, 0x00);
    return -1;
}

AFE77xx_RegReadWrite(fd, 0x00BB, 0x01, 0, 0);
AFE77xx_RegReadWrite(fd, 0x00A9, 0x02, 0, 1);

AFE77xx_RegReadWrite(fd, 0x00D7, 0x00, 2, 3);

```

```

AFE77xx_RegReadWrite(fd, 0x00AE, 0x01, 0, 0);
AFE77xx_RegReadWrite(fd, 0x00AE, 0x00, 0, 0);

sysUsDelay(1000);
count = 0;
while(count<0xff){
    AFE77xx_RegRead(fd, 0x00FE, &done_val);
    if((done_val&0x08) == 0x08){
        printf("done\n");
        break;
    }
    sysUsDelay(1000);
    count++;
}
if(count>= 0xff){
    printf("Capture not done\n");
    AFE77xx_RegWrite(fd, 0x0013, 0x00);
    return -1;
}

AFE77xx_RegReadWrite(fd, 0x0108, 0x80, 7, 7);
AFE77xx_RegReadWrite(fd, 0x0108, 0x20, 5, 5);
AFE77xx_RegReadWrite(fd, 0x0109, 0x01, 0, 0);

AFE77xx_RegWrite(fd, 0x0013, 0x00);

AFE77xx_RegWrite(fd, 0x0013, 0x40);

int start_address_i    = 0x20;
int start_address_i1   = 0x20+4*1024*4;
int start_address_q    = 0x20 +4*1024;
int start_address_q1   = 0x20 +4*1024*5;
int val;
int temp;

for(int i =0; i<samples/2; i++){
    AFE77xx_RegRead(fd, start_address_i+(i*4), &temp);
    val = temp;
    AFE77xx_RegRead(fd, start_address_i+1+(i*4), &temp);
    val = val+(temp<<8);
    AFE77xx_RegRead(fd, start_address_i+2+(i*4), &temp);
    val = val+(temp<<16);
    AFE77xx_RegRead(fd, start_address_i+3+(i*4), &temp);
    val = val+(temp<<24);

    data_i[2*i] = twos_comp(val,32);

    AFE77xx_RegRead(fd, start_address_i1+(i*4), &temp);
    val = temp;
    AFE77xx_RegRead(fd, start_address_i1+1+(i*4), &temp);
    val = val+(temp<<8);
}

```

```

AFE77xx_RegRead(fd, start_address_i1+2+(i*4), &temp);
val = val+(temp<<16);
AFE77xx_RegRead(fd, start_address_i1+3+(i*4), &temp);
val = val+(temp<<24);
data_i[2*i+1] = twos_comp(val,32);

AFE77xx_RegRead(fd, start_address_q+(i*4), &temp);
val = temp;
AFE77xx_RegRead(fd, start_address_q+1+(i*4), &temp);
val = val+(temp<<8);
AFE77xx_RegRead(fd, start_address_q+2+(i*4), &temp);
val = val+(temp<<16);
AFE77xx_RegRead(fd, start_address_q+3+(i*4), &temp);
val = val+(temp<<24);

data_q[2*i] = twos_comp(val,32);

AFE77xx_RegRead(fd, start_address_q1+(i*4), &temp);
val = temp;
AFE77xx_RegRead(fd, start_address_q1+1+(i*4), &temp);
val = val+(temp<<8);
AFE77xx_RegRead(fd, start_address_q1+2+(i*4), &temp);
val = val+(temp<<16);
AFE77xx_RegRead(fd, start_address_q1+3+(i*4), &temp);
val = val+(temp<<24);
data_q[2*i+1] = twos_comp(val,32);
}
AFE77xx_RegWrite(fd, 0x0294, (retVal1));
AFE77xx_RegWrite(fd, 0x0295, (retVal2));
AFE77xx_RegWrite(fd, 0x0013, 0x00);
printf("i data\n");
for(int i=0;i<samples;i++){
    printf("%d ",data_i[i]);
}
printf("\n");
printf("q data\n");
for(int i=0;i<samples;i++){
    printf("%d ",data_q[i]);
}
printf("\n");
return 0;
}

```

/* PLL related functions */

U32 configurePll(int fd, U32 ulChan, double ulfreq, double ulRefFreq){

/*
enabled use "setPllLoFbNcoFreezeQec" function if IQMC is

ulChan is PLL index from 0 to 4.
ulChan=1 is the main data converter PLL.
ulfreq is the PLL output frequency in MHz.

```

        ulRefFreq is the input Reference Clock to the device
*/
if (ulChan == 1) {
    printf("configuring DC PLL is not supported\n");
    return 0;
}
U32 ulRegValue = 0;
double pllMulFact = 0.0;
double pllVcoInput=0;
U8 ucPLL_CAL_CNTRL = 0;
U8 ucPLL_OP_DIVVert = 0;
U8 ucEN_FRAC = 0;
U8 ucPLL_OP_DIV = 0;
U8 ucPLL_N = 0;
U32 ulPLL_D = 0x40000;
U32 ulPLL_F = 0;
U32 temp=0;
double pllVcoFreq=0.0;
double actualOutputFreq=0.0;

AFE77xx_RegWrite(fd, 0x0015, 0x80);
#ifdef TIENV
AFE77xx_RegWrite(fd, 0x01d4, 0x00);
AFE77xx_RegWrite(fd, 0x0374, 0x01);
#else
AFE77xx_RegWrite(fd, 0x01d4, 0x01);
AFE77xx_RegWrite(fd, 0x0374, 0x00);
#endif
AFE77xx_RegWrite(fd, 0x0015, 0x00);
sysUsDelay(1000);
/*step 1*/
AFE77xx_RegWrite(fd, 0x0014, 0x08<<ulChan);

/*step 2*/
if (ulChan==1) {
    pllVcoFreq=ulfreq;
}
else {
    pllVcoFreq=2*ulfreq;
}
ucPLL_OP_DIV = (U32) (ceil(5800.0/pllVcoFreq)) & 0xff;
if (ucPLL_OP_DIV > 4) {
    ucPLL_OP_DIV = 8;
}
pllVcoInput=ulRefFreq;
pllMulFact = pllVcoFreq*ucPLL_OP_DIV/pllVcoInput;
ucPLL_N = (U8) (pllVcoFreq*ucPLL_OP_DIV/pllVcoInput);

if (pllVcoInput*1000>pow(2,18)) {
    temp=(U32) ceil(pllVcoInput*1000/pow(2,18));
    ulPLL_D=(U32) (pllVcoInput*1000/temp);
}

```



```

else{
    temp=1;
    ulPLL_D=(U32) (pllVcoInput*1000);
}
ulPLL_F = floor((pllMulFact-ucPLL_N)*ulPLL_D);
if (ulPLL_F!=0){
    temp=temp*ucPLL_OP_DIV;
}
else if (((ucPLL_N*ulPLL_D)+ulPLL_F)%temp)!=0){
    if (ulPLL_F>=((ucPLL_N*ulPLL_D)+ulPLL_F)%temp){
        ulPLL_F=ulPLL_F-(((ucPLL_N*ulPLL_D)+ulPLL_F)%
temp);
    }
    else{
        ucPLL_N=ucPLL_N-1;
        ulPLL_F=ulPLL_D+ulPLL_F-
((ucPLL_N*ulPLL_D)+ulPLL_F)%temp);
    }
}
if(0 == ulPLL_F){
    ucEN_FRAC = 0;
    actualOutputFreq=pllVcoInput*ucPLL_N/2;
}
else{
    ucEN_FRAC = 1;
    actualOutputFreq=pllVcoInput*((double)
ucPLL_N+((double)ulPLL_F/ulPLL_D))/(ucPLL_OP_DIV*2);
}
if(61.440 == pllVcoInput){
    ucPLL_CAL_CNTRL = 5;
}
else if((92.160 == pllVcoInput)|| (122.880 == pllVcoInput)){
    ucPLL_CAL_CNTRL = 6;
}
else if((184.320 == pllVcoInput)|| (245.760 == pllVcoInput))
{
    ucPLL_CAL_CNTRL = 7;
}
else if((368.640 == pllVcoInput)|| (491.520 == pllVcoInput)){
    ucPLL_CAL_CNTRL = 8;
}
else{
    printf("Error: Couldn't find ucPLL_CAL_CNTRL setting.
Please check the frequency.\r\n");
    return 0;
}
AFE77xx_RegWrite(fd,0x005e,0x97);
AFE77xx_RegWrite(fd,0x0028,ucPLL_N);
AFE77xx_RegWrite(fd,0x0036,ucPLL_N);
AFE77xx_RegWrite(fd,0x0035,ucPLL_CAL_CNTRL);
AFE77xx_RegWrite(fd,0x0051,0x06);
AFE77xx_RegWrite(fd,0x0050,0x40);

```

```

AFE77xx_RegWrite(fd,0x0051,0x06);
AFE77xx_RegWrite(fd,0x0051,0x06);
AFE77xx_RegWrite(fd,0x0051,0x0e);
if (ucPLL_N<15){
    AFE77xx_RegWrite(fd,0x003c,0x41);
}
else{
    AFE77xx_RegWrite(fd,0x003c,0x40);
}
AFE77xx_RegWrite(fd,0x005e,ucEN_FRAC|0x96);
AFE77xx_RegWrite(fd,0x0033,ucEN_FRAC|0x02);
AFE77xx_RegWrite(fd,0x003f,0x0c);
AFE77xx_RegWrite(fd,0x0049,0x01);
AFE77xx_RegWrite(fd,0x0048,0x2f);
AFE77xx_RegWrite(fd,0x0044,0x27);
AFE77xx_RegWrite(fd,0x0043,0x4c);
AFE77xx_RegWrite(fd,0x0043,0x0c);
AFE77xx_RegWrite(fd,0x0042,0x04);
AFE77xx_RegWrite(fd,0x0046,0x00);
AFE77xx_RegWrite(fd,0x0040,0x28);
AFE77xx_RegWrite(fd,0x002e,(ulPLL_F >> 16)&(0xff));
AFE77xx_RegWrite(fd,0x002d,(ulPLL_F >> 8)&(0xff));
AFE77xx_RegWrite(fd,0x002c,ulPLL_F&(0xff));

AFE77xx_RegRead(fd,0x0032,&ulRegValue);
AFE77xx_RegWrite(fd,0x0032,(ulRegValue & 0xF0)+((ulPLL_D >>
16)&(0x0f)));

AFE77xx_RegWrite(fd,0x0031,(ulPLL_D >> 8)&(0xff));
AFE77xx_RegWrite(fd,0x0030,ulPLL_D&(0xff));
AFE77xx_RegWrite(fd,0x0033,ucEN_FRAC|0x02);
AFE77xx_RegWrite(fd,0x0039,0x40);
AFE77xx_RegWrite(fd,0x003f,0x14);
if(8 == ucPLL_OP_DIV ){
    ucPLL_OP_DIVVert = 6;
}
else{
    ucPLL_OP_DIVVert = ucPLL_OP_DIV;
}
AFE77xx_RegWrite(fd,0x003f,0x10|ucPLL_OP_DIVVert);

if (ucPLL_N<15){
    AFE77xx_RegWrite(fd,0x003c,0x41);
}
else{
    AFE77xx_RegWrite(fd,0x003c,0x40);
}
AFE77xx_RegWrite(fd,0x003d,0x24);
AFE77xx_RegWrite(fd,0x0052,0x10);
AFE77xx_RegWrite(fd,0x003b,0x30);
AFE77xx_RegWrite(fd,0x0034,0x03);
AFE77xx_RegWrite(fd,0x0034,0x0f);

```

```

AFE77xx_RegWrite(fd,0x0034,0x1f);
AFE77xx_RegWrite(fd,0x0034,0x1f);
AFE77xx_RegWrite(fd,0x0034,0x5f);
sysUsDelay(1000);
if(1 == ucEN_FRAC){
    if (ucPLL_N<15){
        AFE77xx_RegWrite(fd,0x005e,0x9f);
        AFE77xx_RegWrite(fd,0x003c,0x41);
        AFE77xx_RegWrite(fd,0x0065,0x10);
        AFE77xx_RegWrite(fd,0x003c,0x45);
    }
    else{
        AFE77xx_RegWrite(fd,0x005e,0x9f);
        AFE77xx_RegWrite(fd,0x003c,0x40);
        AFE77xx_RegWrite(fd,0x0065,0x10);
        AFE77xx_RegWrite(fd,0x003c,0x48);
    }
}

AFE77xx_RegRead(fd,0x0062,&ulRegValue);
if ((ulRegValue&0x80)==0){
    printf("PLL %d didn't lock\n",ulChan);
}
else{
    printf("PLL %d locked.\n",ulChan);
    AFE77xx_RegWrite(fd,0x0066,0x0f);
    AFE77xx_RegWrite(fd,0x0066,0x03);
}
if (ulChan==1){
    AFE77xx_RegWrite(fd,0x0051,0x0c);
}
AFE77xx_RegRead(fd,0x0063,&ulRegValue);
printf("value read from 0x63 is %d\n",ulRegValue);

AFE77xx_RegWrite(fd,0x0014,0x00);
printf("actualOutputFreq %lf\n",actualOutputFreq);
AFE77xx_RegWrite(fd,0x0015,0x80);
AFE77xx_RegWrite(fd,0x01d4,0x00);
AFE77xx_RegWrite(fd,0x0374,0x00);
AFE77xx_RegWrite(fd,0x0015,0x00);
sysUsDelay(1000);
return 0;
}

U32 setPllLoFbNcoFreezeQec(int fd, double Fout, double Fref, int
ulPll, int ulFbnco){
    /*    ulPll 0-4 for each channel, 5- no channel is selected
        ulFbnco= 0:FBAB,1:FBCD, 2:No channel is
selected,3:FBAB band1 NCO,4:FBCD band1 NCO
    */
    double actualOutputFreq=0;

```

```

    actualOutputFreq=Fout;/
*/findLoNcoPossibleFrequency(Fout,Fref);*/
    //Preserve "Pin control on TDD" settings
    U8 tddPinCtrl[12];
    AFE77xx_RegWrite(fd, 0x14,0x04);
    AFE77xx_RegRead(fd, 0x124,&tddPinCtrl[0]);
    AFE77xx_RegRead(fd, 0x126,&tddPinCtrl[1]);
    AFE77xx_RegRead(fd, 0x128,&tddPinCtrl[2]);
    AFE77xx_RegRead(fd, 0x12A,&tddPinCtrl[3]);
    AFE77xx_RegRead(fd, 0x12C,&tddPinCtrl[4]);
    AFE77xx_RegRead(fd, 0x12E,&tddPinCtrl[5]);
    AFE77xx_RegRead(fd, 0x125,&tddPinCtrl[6]);
    AFE77xx_RegRead(fd, 0x127,&tddPinCtrl[7]);
    AFE77xx_RegRead(fd, 0x129,&tddPinCtrl[8]);
    AFE77xx_RegRead(fd, 0x12B,&tddPinCtrl[9]);
    AFE77xx_RegRead(fd, 0x12D,&tddPinCtrl[10]);
    AFE77xx_RegRead(fd, 0x12F,&tddPinCtrl[11]);
    AFE77xx_RegWrite(fd, 0x14,0x00);

    //Freeze TX and RX QEC
    //TX Freeze Start
    AFE77xx_RegWrite(fd,0x14,0x04);
    AFE77xx_RegWrite(fd,0x108,0x01);
    AFE77xx_RegWrite(fd,0x109,0x04);
    AFE77xx_RegWrite(fd,0x14,0x00);

    AFE77xx_RegWrite(fd,0x13,0x10);
    AFE77xx_RegWrite(fd,0xA3,0x00);
    AFE77xx_RegWrite(fd,0xA2,0x00);
    AFE77xx_RegWrite(fd,0xA1,0xFF);
    AFE77xx_RegWrite(fd,0xA0,0x06);
    AFE77xx_RegWrite(fd,0x193,0x35);
    sysUsDelay(20000); //20ms
    AFE77xx_RegWrite(fd,0x13,0x00);
    //TX Freeze End

    //RX Freeze Start RX TDD force 1 during freeze
    AFE77xx_RegWrite(fd,0x014,0x04);
    AFE77xx_RegWrite(fd,0x124,0x01);
    AFE77xx_RegWrite(fd,0x126,0x01);
    AFE77xx_RegWrite(fd,0x128,0x01);
    AFE77xx_RegWrite(fd,0x12A,0x01);
    AFE77xx_RegWrite(fd,0x12C,0x01);
    AFE77xx_RegWrite(fd,0x12E,0x01);
    AFE77xx_RegWrite(fd,0x125,0x00);
    AFE77xx_RegWrite(fd,0x127,0x01);
    AFE77xx_RegWrite(fd,0x129,0x00);
    AFE77xx_RegWrite(fd,0x12B,0x00);
    AFE77xx_RegWrite(fd,0x12D,0x01);
    AFE77xx_RegWrite(fd,0x12F,0x00);
    AFE77xx_RegWrite(fd,0x14,0x00);

```

```

AFE77xx_RegWrite(fd,0x13,0x20);
AFE77xx_RegWrite(fd,0x1A2,0x0F);
AFE77xx_RegWrite(fd,0x1A1,0x00);
AFE77xx_RegWrite(fd,0x1A0,0x00);
AFE77xx_RegWrite(fd,0x1A3,0x11);
AFE77xx_RegWrite(fd,0x13,0x00);

sysUsDelay(20000);
AFE77xx_RegWrite(fd,0x18,0x10);
AFE77xx_RegWrite(fd,0xBC,0x00);
AFE77xx_RegWrite(fd,0xBD,0x00);
AFE77xx_RegWrite(fd,0x24,0x00);
AFE77xx_RegWrite(fd,0x30,0x01);
AFE77xx_RegWrite(fd,0x18,0x00);

AFE77xx_RegWrite(fd,0x10,0x0C);
AFE77xx_RegWrite(fd,0xA05,0x00);
AFE77xx_RegWrite(fd,0x10,0x00);
// "RX Freeze End"

//Configure PLL and Set FB NCO
if(ulPll<5){
    configurePll(fd,ulPll,actualOutputFreq,Fref);
}
if(ulFbnco<2){
    FbNCO(fd,actualOutputFreq,ulFbnco);
}
else if((ulFbnco==3)|| (ulFbnco==4)){
    FbNCO(fd,actualOutputFreq,(ulFbnco-1));
}

//Relatch Sysref for LO and give to Analog
AFE77xx_RegWrite(fd,0x14,0x04);
AFE77xx_RegWrite(fd,0x4fd,0x0);
AFE77xx_RegWrite(fd,0x4fc,0x7e);

AFE77xx_RegWrite(fd,0x4D4,0x0B);
AFE77xx_RegWrite(fd,0x4D5,0x0B);
AFE77xx_RegWrite(fd,0x4D6,0x0B);
AFE77xx_RegWrite(fd,0x4D8,0x0B);
AFE77xx_RegWrite(fd,0x4D9,0x0B);
AFE77xx_RegWrite(fd,0x4DA,0x0B);
AFE77xx_RegWrite(fd,0x14,0x0);

AFE77xx_RegWrite(fd,0x0015,0x80);
#ifdef TIENV
AFE77xx_RegWrite(fd,0x01d4,0x00);
AFE77xx_RegWrite(fd,0x0374,0x01);
#else
AFE77xx_RegWrite(fd,0x01d4,0x01);
AFE77xx_RegWrite(fd,0x0374,0x00);
#endif

```

```

AFE77xx_RegWrite (fd, 0x0015, 0x00);

sysUsDelay (1000);

AFE77xx_RegWrite (fd, 0x0014, 0x08);
AFE77xx_RegWrite (fd, 0x004c, 0xfc);
AFE77xx_RegWrite (fd, 0x0051, 0x8e);
AFE77xx_RegWrite (fd, 0x004e, 0x30);
AFE77xx_RegWrite (fd, 0x004d, 0x05);
AFE77xx_RegWrite (fd, 0x004c, 0xfc);
AFE77xx_RegWrite (fd, 0x004c, 0xfc);
AFE77xx_RegWrite (fd, 0x004c, 0xfd);

// "Sysref taken from pin here"
sysUsDelay (1000);

AFE77xx_RegWrite (fd, 0x0014, 0x08);
AFE77xx_RegWrite (fd, 0x0051, 0x0e);
AFE77xx_RegWrite (fd, 0x0051, 0x06);
AFE77xx_RegWrite (fd, 0x0014, 0x00);

sysUsDelay (100);

// Sysref Given to TX and FB
AFE77xx_RegWrite (fd, 0x14, 0x10);
AFE77xx_RegWrite (fd, 0x5d, 0xe0);
sysUsDelay (100);
AFE77xx_RegWrite (fd, 0x5d, 0xc0);
AFE77xx_RegWrite (fd, 0x14, 0x00);

sysUsDelay (100);

AFE77xx_RegWrite (fd, 0x0015, 0x80);
AFE77xx_RegWrite (fd, 0x01d4, 0x00);
AFE77xx_RegWrite (fd, 0x0374, 0x00);
AFE77xx_RegWrite (fd, 0x0015, 0x00);

AFE77xx_RegWrite (fd, 0x14, 0x04);
AFE77xx_RegWrite (fd, 0x4fd, 0x7);
AFE77xx_RegWrite (fd, 0x4fc, 0xff);
AFE77xx_RegWrite (fd, 0x14, 0x00);

AFE77xx_RegWrite (fd, 0x14, 0x04);
AFE77xx_RegWrite (fd, 0x124, 0x01);
AFE77xx_RegWrite (fd, 0x126, 0x01);
AFE77xx_RegWrite (fd, 0x128, 0x01);
AFE77xx_RegWrite (fd, 0x12A, 0x01);
AFE77xx_RegWrite (fd, 0x12C, 0x01);
AFE77xx_RegWrite (fd, 0x12E, 0x01);
AFE77xx_RegWrite (fd, 0x125, 0x00);
AFE77xx_RegWrite (fd, 0x127, 0x00);
AFE77xx_RegWrite (fd, 0x129, 0x00);

```

```

AFE77xx_RegWrite(fd,0x12B,0x00);
AFE77xx_RegWrite(fd,0x12D,0x00);
AFE77xx_RegWrite(fd,0x12F,0x00);
AFE77xx_RegWrite(fd,0x14,0x00);
sysUsDelay(1000);

AFE77xx_RegWrite(fd,0x14,0x04);
AFE77xx_RegWrite(fd,0x124,0x01);
AFE77xx_RegWrite(fd,0x126,0x01);
AFE77xx_RegWrite(fd,0x128,0x01);
AFE77xx_RegWrite(fd,0x12A,0x01);
AFE77xx_RegWrite(fd,0x12C,0x01);
AFE77xx_RegWrite(fd,0x12E,0x01);
AFE77xx_RegWrite(fd,0x125,0x01);
AFE77xx_RegWrite(fd,0x127,0x00);
AFE77xx_RegWrite(fd,0x129,0x01);
AFE77xx_RegWrite(fd,0x12B,0x01);
AFE77xx_RegWrite(fd,0x12D,0x00);
AFE77xx_RegWrite(fd,0x12F,0x01);
AFE77xx_RegWrite(fd,0x14,0x00);
sysUsDelay(1000);

//UnFreeze RX and TX QEC
//"TX UnFreeze Start//"
AFE77xx_RegWrite(fd, 0x13,0x10);
AFE77xx_RegWrite(fd, 0xA3,0x00);
AFE77xx_RegWrite(fd, 0xA2,0x00);
AFE77xx_RegWrite(fd, 0xA1,0xFF);
AFE77xx_RegWrite(fd, 0xA0,0x07);
AFE77xx_RegWrite(fd, 0x193,0x35);
sysUsDelay(10000);
AFE77xx_RegWrite(fd, 0x13,0x00);

AFE77xx_RegWrite(fd, 0x16,0x08);
AFE77xx_RegWrite(fd, 0x100,0x00);
sysUsDelay(1000);
AFE77xx_RegWrite(fd, 0x100,0x01);
AFE77xx_RegWrite(fd, 0x16,0x00);

sysUsDelay(1000);

AFE77xx_RegWrite(fd, 0x12,0x10);
AFE77xx_RegWrite(fd, 0x20,0x8A);
AFE77xx_RegWrite(fd, 0x21,0x13);
AFE77xx_RegWrite(fd, 0x22,0x01);
AFE77xx_RegWrite(fd, 0x23,0x20);
AFE77xx_RegWrite(fd, 0x24,0x01);
AFE77xx_RegWrite(fd, 0x25,0x01);
AFE77xx_RegWrite(fd, 0x26,0x01);
AFE77xx_RegWrite(fd, 0x27,0x01);
AFE77xx_RegWrite(fd, 0x28,0x02);
AFE77xx_RegWrite(fd, 0x29,0x00);

```

```

AFE77xx_RegWrite(fd, 0x2A, 0x00);
AFE77xx_RegWrite(fd, 0x2B, 0x00);
AFE77xx_RegWrite(fd, 0x12, 0x00);
AFE77xx_RegWrite(fd, 0x13, 0x10);
AFE77xx_RegWrite(fd, 0xA3, 0x00);
AFE77xx_RegWrite(fd, 0xA2, 0x0A);
AFE77xx_RegWrite(fd, 0xA1, 0x06);
AFE77xx_RegWrite(fd, 0xA0, 0x10);
AFE77xx_RegWrite(fd, 0x193, 0x35);
AFE77xx_RegWrite(fd, 0x192, 0x00);
AFE77xx_RegWrite(fd, 0x191, 0x00);
AFE77xx_RegWrite(fd, 0x190, 0x00);
AFE77xx_RegWrite(fd, 0x13, 0x00);
sysUsDelay(1000);
// "TX UnFreeze End//"

AFE77xx_RegWrite(fd, 0x14, 0x04);
AFE77xx_RegWrite(fd, 0x124, 0x01);
AFE77xx_RegWrite(fd, 0x126, 0x01);
AFE77xx_RegWrite(fd, 0x128, 0x01);
AFE77xx_RegWrite(fd, 0x12A, 0x01);
AFE77xx_RegWrite(fd, 0x12C, 0x01);
AFE77xx_RegWrite(fd, 0x12E, 0x01);
AFE77xx_RegWrite(fd, 0x125, 0x00);
AFE77xx_RegWrite(fd, 0x127, 0x01);
AFE77xx_RegWrite(fd, 0x129, 0x00);
AFE77xx_RegWrite(fd, 0x12B, 0x00);
AFE77xx_RegWrite(fd, 0x12D, 0x01);
AFE77xx_RegWrite(fd, 0x12F, 0x00);
AFE77xx_RegWrite(fd, 0x14, 0x00);
sysUsDelay(1000);

// "Rx UnFreeze Start//"
AFE77xx_RegWrite(fd, 0x10, 0x04);
AFE77xx_RegWrite(fd, 0xA05, 0x01);
AFE77xx_RegWrite(fd, 0x10, 0x08);
AFE77xx_RegWrite(fd, 0xA05, 0x01);
AFE77xx_RegWrite(fd, 0x10, 0x00);
AFE77xx_RegWrite(fd, 0x13, 0x20);
AFE77xx_RegWrite(fd, 0x1A2, 0x0F);
AFE77xx_RegWrite(fd, 0x1A1, 0x00);
AFE77xx_RegWrite(fd, 0x1A0, 0x00);
AFE77xx_RegWrite(fd, 0x1A3, 0x17);
sysUsDelay(10000);

AFE77xx_RegWrite(fd, 0x1A2, 0x0F);
AFE77xx_RegWrite(fd, 0x1A1, 0x00);
AFE77xx_RegWrite(fd, 0x1A0, 0x00);
AFE77xx_RegWrite(fd, 0x1A3, 0x12);
AFE77xx_RegWrite(fd, 0x13, 0x00);
// "Rx UnFreeze End//"

```



```

sysUsDelay(10000);

// "Releasing Pin control on TDD//"
AFE77xx_RegWrite(fd, 0x14, 0x04);
AFE77xx_RegWrite(fd, 0x124, tddPinCtrl[0]);
AFE77xx_RegWrite(fd, 0x126, tddPinCtrl[1]);
AFE77xx_RegWrite(fd, 0x128, tddPinCtrl[2]);
AFE77xx_RegWrite(fd, 0x12A, tddPinCtrl[3]);
AFE77xx_RegWrite(fd, 0x12C, tddPinCtrl[4]);
AFE77xx_RegWrite(fd, 0x12E, tddPinCtrl[5]);
AFE77xx_RegWrite(fd, 0x125, tddPinCtrl[6]);
AFE77xx_RegWrite(fd, 0x127, tddPinCtrl[7]);
AFE77xx_RegWrite(fd, 0x129, tddPinCtrl[8]);
AFE77xx_RegWrite(fd, 0x12B, tddPinCtrl[9]);
AFE77xx_RegWrite(fd, 0x12D, tddPinCtrl[10]);
AFE77xx_RegWrite(fd, 0x12F, tddPinCtrl[11]);
AFE77xx_RegWrite(fd, 0x14, 0x00);
sysUsDelay(10000);

// Change TX QEC mode into GPIO mode
AFE77xx_RegWrite(fd, 0x14, 0x04);
AFE77xx_RegWrite(fd, 0x108, 0x00);
AFE77xx_RegWrite(fd, 0x14, 0x00);
}

U32 configurePLLmode(int fd, U32 ulPLLMode){
    /*
        ulPLLMode is the PLL mode
        ulPLLMode=0 means Both TXLO and RXLO is PLL0. Both
        TXLO=RXLO=ulTXLO
        ulPLLMode=1 means TXLO is PLL0 and RXLO uses PLL2
    */
    // Override Tdd
    AFE77xx_RegWrite(fd, 0x14, 0x04);
    AFE77xx_RegWrite(fd, 0x124, 0x01);
    AFE77xx_RegWrite(fd, 0x126, 0x01);
    AFE77xx_RegWrite(fd, 0x128, 0x01);
    AFE77xx_RegWrite(fd, 0x12A, 0x01);
    AFE77xx_RegWrite(fd, 0x12C, 0x01);
    AFE77xx_RegWrite(fd, 0x12E, 0x01);
    AFE77xx_RegWrite(fd, 0x125, 0x00);
    AFE77xx_RegWrite(fd, 0x127, 0x00);
    AFE77xx_RegWrite(fd, 0x129, 0x00);
    AFE77xx_RegWrite(fd, 0x12B, 0x00);
    AFE77xx_RegWrite(fd, 0x12D, 0x00);
    AFE77xx_RegWrite(fd, 0x12F, 0x00);
    AFE77xx_RegWrite(fd, 0x14, 0x00);
    sysUsDelay(1000);
    // Override Tdd

    if (ulPLLMode==0){
        // Disable PLL2 VCO

```

```

AFE77xx_RegWrite(fd,0x0015,0x80);
#ifdef TIENV
AFE77xx_RegWrite(fd,0x01d4,0x00);
AFE77xx_RegWrite(fd,0x0374,0x01);
#else
AFE77xx_RegWrite(fd,0x01d4,0x01);
AFE77xx_RegWrite(fd,0x0374,0x00);
#endif
AFE77xx_RegWrite(fd,0x0015,0x00);
AFE77xx_RegWrite(fd,0x0014,0x20);
AFE77xx_RegWrite(fd,0x005E,0x1F);
AFE77xx_RegWrite(fd,0x0014,0x00);
//DONE:Disable PLL2 VCO

//Configure LOMUX
AFE77xx_RegWrite(fd,0x0014,0x01);
AFE77xx_RegWrite(fd,0x0064,0x00);
AFE77xx_RegWrite(fd,0x0062,0x03);
AFE77xx_RegWrite(fd,0x0071,0x00);
AFE77xx_RegWrite(fd,0x0060,0x01);
AFE77xx_RegWrite(fd,0x0014,0x00);
//DONE:Configure LOMUX

AFE77xx_RegWrite(fd,0x0015,0x80);
AFE77xx_RegWrite(fd,0x01d4,0x00);
AFE77xx_RegWrite(fd,0x0374,0x00);
AFE77xx_RegWrite(fd,0x0015,0x00);
}
if (ulPLLMode==1){
//configurePLL will enable the PLL2 VCO by default
//Enable PLL2 VCO
AFE77xx_RegWrite(fd,0x0015,0x80);
#ifdef TIENV
AFE77xx_RegWrite(fd,0x01d4,0x00);
AFE77xx_RegWrite(fd,0x0374,0x01);
#else
AFE77xx_RegWrite(fd,0x01d4,0x01);
AFE77xx_RegWrite(fd,0x0374,0x00);
#endif
AFE77xx_RegWrite(fd,0x0015,0x00);
AFE77xx_RegWrite(fd,0x0014,0x20);
AFE77xx_RegWrite(fd,0x005E,0x9F);
AFE77xx_RegWrite(fd,0x0014,0x00);
//DONE:Enable PLL2 VCO

//Configure LOMUX
AFE77xx_RegWrite(fd,0x0015,0x80);
#ifdef TIENV
AFE77xx_RegWrite(fd,0x01d4,0x00);
AFE77xx_RegWrite(fd,0x0374,0x01);
#else
AFE77xx_RegWrite(fd,0x01d4,0x01);

```

```

AFE77xx_RegWrite(fd, 0x0374, 0x00);
#endif
AFE77xx_RegWrite(fd, 0x0015, 0x00);

AFE77xx_RegWrite(fd, 0x0014, 0x01);
AFE77xx_RegWrite(fd, 0x0064, 0x00);
AFE77xx_RegWrite(fd, 0x0062, 0x00);
AFE77xx_RegWrite(fd, 0x0071, 0x03);
AFE77xx_RegWrite(fd, 0x0060, 0x01);
AFE77xx_RegWrite(fd, 0x0014, 0x00);
//DONE:Configure LOMUX

AFE77xx_RegWrite(fd, 0x0015, 0x80);
AFE77xx_RegWrite(fd, 0x01d4, 0x00);
AFE77xx_RegWrite(fd, 0x0374, 0x00);
AFE77xx_RegWrite(fd, 0x0015, 0x00);
}
//Release TDD Override control
sysUsDelay(1000);
AFE77xx_RegWrite(fd, 0x14, 0x04);
AFE77xx_RegWrite(fd, 0x124, 0x00);
AFE77xx_RegWrite(fd, 0x126, 0x00);
AFE77xx_RegWrite(fd, 0x128, 0x00);
AFE77xx_RegWrite(fd, 0x12A, 0x00);
AFE77xx_RegWrite(fd, 0x12C, 0x00);
AFE77xx_RegWrite(fd, 0x12E, 0x00);
AFE77xx_RegWrite(fd, 0x125, 0x00);
AFE77xx_RegWrite(fd, 0x127, 0x00);
AFE77xx_RegWrite(fd, 0x129, 0x00);
AFE77xx_RegWrite(fd, 0x12B, 0x00);
AFE77xx_RegWrite(fd, 0x12D, 0x00);
AFE77xx_RegWrite(fd, 0x12F, 0x00);
AFE77xx_RegWrite(fd, 0x14, 0x00);
}

U32 checkPllLockStatus(int fd, U32 ulChan){
/*
    ulChan is PLL index from 0 to 4.
    ulChan=1 is the main data converter PLL.
    ulfreq is the PLL output frequency in MHz.
    ulRefFreq is the input Reference Clock to the device
*/
U32 ulRegValue = 0;
AFE77xx_RegWrite(fd, 0x0015, 0x80);
#ifdef TIENV
AFE77xx_RegWrite(fd, 0x01d4, 0x00);
AFE77xx_RegWrite(fd, 0x0374, 0x01);
#else
AFE77xx_RegWrite(fd, 0x01d4, 0x01);
AFE77xx_RegWrite(fd, 0x0374, 0x00);
#endif
AFE77xx_RegWrite(fd, 0x0015, 0x00);

```

```

sysUsDelay(1000);
AFE77xx_RegWrite(fd,0x0014,0x08<<ulChan);
AFE77xx_RegRead(fd,0x0062,&ulRegValue);
if ((ulRegValue&0x80)==0){
    printf("PLL %d didn't lock\n",ulChan);
}
else{
    printf("PLL %d locked.\n",ulChan);
}
AFE77xx_RegWrite(fd,0x0014,0x00);

AFE77xx_RegWrite(fd,0x0015,0x80);
AFE77xx_RegWrite(fd,0x01d4,0x00);
AFE77xx_RegWrite(fd,0x0374,0x00);
AFE77xx_RegWrite(fd,0x0015,0x00);
sysUsDelay(1000);
return ((ulRegValue&0x80)>>7);
}

double findpllFrequency(int fd, U8 ulChan, double ulRefFreq){
    U8 ucPLL_OP_DIV = 0;
    U8 ucPLL_N = 0;
    U32 ulPLL_D = 0x40000;
    U32 ulPLL_F = 0;
    double outputFreq,multFact;
    U8 tempReg=0;

    AFE77xx_RegWrite(fd,0x0015,0x80);
    #ifdef TIENV
    AFE77xx_RegWrite(fd,0x01d4,0x00);
    AFE77xx_RegWrite(fd,0x0374,0x01);
    #else
    AFE77xx_RegWrite(fd,0x01d4,0x01);
    AFE77xx_RegWrite(fd,0x0374,0x00);
    #endif
    AFE77xx_RegWrite(fd,0x0015,0x00);
    sysUsDelay(1000);
    AFE77xx_RegWrite(fd,0x0014,0x08<<ulChan);

    AFE77xx_RegRead(fd,0x0028,&tempReg);
    ucPLL_N = tempReg;
    AFE77xx_RegRead(fd,0x002e,&tempReg);
    ulPLL_F=tempReg<<16;
    AFE77xx_RegRead(fd,0x002d,&tempReg);
    ulPLL_F=ulPLL_F+(tempReg<<8);
    AFE77xx_RegRead(fd,0x002c,&tempReg);
    ulPLL_F=ulPLL_F+tempReg;

    AFE77xx_RegRead(fd,0x0032,&tempReg);
    ulPLL_D=(tempReg & 0xF)<<16;
    AFE77xx_RegRead(fd,0x0031,&tempReg);
    ulPLL_D=ulPLL_D+(tempReg<<8);

```

```

AFE77xx_RegRead(fd,0x0030,&tempReg);
ulPLL_D=ulPLL_D+tempReg;

AFE77xx_RegRead(fd,0x003f,&ucPLL_OP_DIV);
ucPLL_OP_DIV=ucPLL_OP_DIV&7;
AFE77xx_RegWrite(fd,0x0014,0x0);

multFact=((ucPLL_N)+((float)
ulPLL_F/ulPLL_D))/ucPLL_OP_DIV);
if (ulChan==1){
    outputFreq=multFact*ulRefFreq;
}
else{
    outputFreq=multFact*ulRefFreq/2;
}
outputFreq=(float)((outputFreq*100.0)/100.0);
AFE77xx_RegWrite(fd,0x0015,0x80);
AFE77xx_RegWrite(fd,0x01d4,0x00);
AFE77xx_RegWrite(fd,0x0374,0x00);
AFE77xx_RegWrite(fd,0x0015,0x00);
sysUsDelay(1000);
printf("outputFreq %lf\n",outputFreq);
return outputFreq;
}

U32 findLoNcoPossibleFrequency(double ulfreq, double ulRefFreq){
/*
    ulChan is PLL index from 0 to 4.
    ulChan=1 is the main data converter PLL.
    ulfreq is the PLL output frequency in MHz.
    ulRefFreq is the input Reference Clock to the device
*/
U32 ulRegValue = 0;
double pllMulFact = 0.0;
double pllVcoInput=0;
U8 ucPLL_CAL_CNTRL = 0;
U8 ucPLL_OP_DIVVert = 0;
U8 ucEN_FRAC = 0;
U8 ucPLL_OP_DIV = 0;
U8 ucPLL_N = 0;
U32 ulPLL_D = 0x40000;
U32 ulPLL_F = 0;
U32 temp=0;
double pllVcoFreq=0.0;
double actualOutputFreq=0.0;

pllVcoFreq=2*ulfreq;

ucPLL_OP_DIV = ceil(5800.0/pllVcoFreq)&&0xff;
if(ucPLL_OP_DIV > 4)
{
    ucPLL_OP_DIV = 8;
}

```

```

    }
    pllVcoInput=ulRefFreq;
    pllMulFact = pllVcoFreq*ucPLL_OP_DIV/pllVcoInput;
    ucPLL_N = (U8) (pllVcoFreq*ucPLL_OP_DIV/pllVcoInput);

    if (pllVcoInput*1000>pow(2,18)) {
        temp=(U32) ceil (pllVcoInput*1000/pow(2,18));
        ulPLL_D=(U32) (pllVcoInput*1000/temp);
    }
    else{
        temp=1;
        ulPLL_D=(U32) (pllVcoInput*1000);
    }
    ulPLL_F = floor((pllMulFact-ucPLL_N)*ulPLL_D);
    if (ulPLL_F!=0){
        temp=temp*ucPLL_OP_DIV;
    }
    else if (((ucPLL_N*ulPLL_D)+ulPLL_F)%temp)!=0){
        if (ulPLL_F>=((ucPLL_N*ulPLL_D)+ulPLL_F)%temp){
            ulPLL_F=ulPLL_F-((ucPLL_N*ulPLL_D)+ulPLL_F)%
temp);
        }
        else{
            ucPLL_N=ucPLL_N-1;
            ulPLL_F=ulPLL_D+ulPLL_F-
((ucPLL_N*ulPLL_D)+ulPLL_F)%temp);
        }
    }
    if(0 == ulPLL_F){
        ucEN_FRAC = 0;
        actualOutputFreq=pllVcoInput*ucPLL_N/2;
    }
    else{
        ucEN_FRAC = 1;
        actualOutputFreq=pllVcoInput*((double)
ucPLL_N+((double)ulPLL_F/ulPLL_D))/(ucPLL_OP_DIV*2);
    }
    printf("actualOutputFreq %lf\n",actualOutputFreq);
    return actualOutputFreq;
}

/* TOP level related functions */
U32 clearSpiAlarms(int fd){
    AFE77xx_RegWrite(fd,0x001a,0xff); /*alarms_clear=0x1fff*/
    AFE77xx_RegWrite(fd,0x001b,0x01);
    AFE77xx_RegWrite(fd,0x001a,0x00); /*alarms_clear=0x0*/
    AFE77xx_RegWrite(fd,0x001b,0x00);
}

U32 clearMacroAlarms(int fd){
    AFE77xx_RegWrite(fd,0x0013,0x20);
    AFE77xx_RegWrite(fd,0x02e5,0x20);
}

```

```

AFE77xx_RegWrite(fd,0x02e5,0x00);
AFE77xx_RegWrite(fd,0x0013,0x00);
}

U32 clearJesdRxAlarms(int fd){
    /* Clearing JESD RX Data and alarms*/
    AFE77xx_RegWrite(fd,0x0015,0x02); /*PAGE:tx_jesd=0x3*/
    AFE77xx_RegWrite(fd,0x0015,0x22);
    AFE77xx_RegWrite(fd,0x0081,0xff); /*jesd_clear_data=0xff*/
    AFE77xx_RegWrite(fd,0x0081,0x00); /*jesd_clear_data=0x0*/
    AFE77xx_RegWrite(fd,0x002d,0xdb); /*serdes_fifo_err_clear=
0x1*/
    AFE77xx_RegWrite(fd,0x002d,0x9b); /*serdes_fifo_err_clear=
0x0*/
    AFE77xx_RegWrite(fd,0x0190,0x05); /*clear_all_alarms=0x1*/
    AFE77xx_RegWrite(fd,0x0190,0x00); /*clear_all_alarms=0x0*/

    AFE77xx_RegWrite(fd,0x0015,0x20); /*PAGE:tx_jesd=0x0*/
    AFE77xx_RegWrite(fd,0x0015,0x00);
}

U32 clearJesdTxAlarms(int fd){
    /* Clearing JESD TX Data and alarms*/
    AFE77xx_RegWrite(fd,0x0015,0x11);
    AFE77xx_RegWrite(fd,0x00b1,0x0f); /
*alarms_serdes_fifo_errors_clear=0xf*/
    AFE77xx_RegWrite(fd,0x00b1,0x00); /
*alarms_serdes_fifo_errors_clear=0x0*/
    AFE77xx_RegWrite(fd,0x0026,0x0f); /*jesd_clear_data=0xf*/
    AFE77xx_RegWrite(fd,0x0026,0x00); /*jesd_clear_data=0x0*/
    AFE77xx_RegWrite(fd,0x0045,0x00); /*fifo_init_state=0x0*/
    AFE77xx_RegWrite(fd,0x0015,0x10); /*PAGE:rx_jesd=0x0*/
    AFE77xx_RegWrite(fd,0x0015,0x00);
}

U32 clearPllAlarms(int fd){
    AFE77xx_RegWrite(fd,0x0015,0x80);
    #ifdef TIENV
    AFE77xx_RegWrite(fd,0x01d4,0x00);
    AFE77xx_RegWrite(fd,0x0374,0x01);
    #else
    AFE77xx_RegWrite(fd,0x01d4,0x01);
    AFE77xx_RegWrite(fd,0x0374,0x00);
    #endif
    AFE77xx_RegWrite(fd,0x0015,0x00);
    sysUsDelay(100);
    AFE77xx_RegWrite(fd,0x0014,0xf8); /*PAGE:pll=0x1f*/
    AFE77xx_RegWrite(fd,0x0066,0x08); /*lock_lost_rst=0x1*/
    AFE77xx_RegWrite(fd,0x0066,0x00); /*lock_lost_rst=0x0*/
    AFE77xx_RegWrite(fd,0x0014,0x00); /*PAGE:pll=0x00*/
    AFE77xx_RegWrite(fd,0x0015,0x80);
    AFE77xx_RegWrite(fd,0x01d4,0x00);
}

```

```

AFE77xx_RegWrite(fd,0x0374,0x00);
AFE77xx_RegWrite(fd,0x0015,0x00);
}

U32 clearPapAlarms(int fd){
AFE77xx_RegWrite(fd,0x0015,0x80);
AFE77xx_RegWrite(fd,0x064a,0x0f);
AFE77xx_RegWrite(fd,0x064a,0x00);
AFE77xx_RegWrite(fd,0x0015,0x00);
}

U32 clearAllAlarms(int fd){
clearSpiAlarms(fd);
clearMacroAlarms(fd);
clearJesdRxAlarms(fd);
clearJesdTxAlarms(fd);
clearPllAlarms(fd);
clearPapAlarms(fd);
return 0;
}

U32 maskSrPapAlarms(int fd, U32 alarmNo, U32 maskValue,U32
txChNo){
/* This masks SR/PAP alarms
alarmNo=0 is pin C17. alarmNo=1 is pin E17.
maskValue=1 will not get the alarm out onto pins.
txChNo is from 0 to 3, for each TX Channel.*/
U32 address=0;
U32 ulRegValue=0;
maskValue=!maskValue;
if (alarmNo==0){
address=0x118;
}
else{
address=0x11c;
}

AFE77xx_RegWrite(fd,0x0015,0x80);
AFE77xx_RegRead(fd,address,&ulRegValue);
if (maskValue==0){
ulRegValue=ulRegValue&(0xff^(1<<(txChNo+1)));
}
else{
ulRegValue=ulRegValue|(1<<(txChNo+1));
}
AFE77xx_RegWrite(fd,address,ulRegValue);
AFE77xx_RegWrite(fd,0x0015,0x00);
}

U32 maskPllAlarms(int fd, U32 alarmNo, U32 pllNo, U32 maskValue){
/* This masks PLL alarms
alarmNo=0 is pin C17. alarmNo=1 is pin E17.

```



```

        maskValue=1 will not get the alarm out onto pins.
        pllNo is the PLL numbers from 0 to 4. 1 is DC PLL.*/
U32 address=0;
U32 ulRegValue=0;
maskValue=!maskValue;
if (alarmNo==0){
    address=0x119;
}
else{
    address=0x11d;
}

AFE77xx_RegWrite(fd,0x0015,0x80);
AFE77xx_RegRead(fd,address,&ulRegValue);
if (maskValue==0){
    ulRegValue=ulRegValue&(0xff^(1<<pllNo));
}
else{
    ulRegValue=ulRegValue|(1<<pllNo);
}
AFE77xx_RegWrite(fd,address,ulRegValue);
AFE77xx_RegWrite(fd,0x0015,0x00);
}

U32 maskSpiAlarms(int fd, U32 alarmNo, U32 maskValue){
    /* This masks SPI alarms
        maskValue=1 will not get the alarm out onto pins.
    */
    U32 address=0;
    U32 ulRegValue=0;
    maskValue=!maskValue;
    if (alarmNo==0){
        address=0x118;
    }
    else{
        address=0x11c;
    }
    AFE77xx_RegWrite(fd,0x0015,0x80);
    AFE77xx_RegRead(fd,address,&ulRegValue);
    if (maskValue==0){
        ulRegValue=ulRegValue&0xdf;
    }
    else{
        ulRegValue=ulRegValue|0x20;
    }
    AFE77xx_RegWrite(fd,address,ulRegValue);
    AFE77xx_RegWrite(fd,0x0015,0x00);
}

U32 tempSensorEnable(int fd, U32 Enable){
    AFE77xx_RegWrite(fd,0x0015,0x80);
    AFE77xx_RegReadWrite(fd,0x0722,(Enable<<1),1,1);
}

```

```

    AFE77xx_RegWrite(fd,0x03a0,((Enable<<1)|0x7c));
    AFE77xx_RegWrite(fd,0x03a3,0x08);
    AFE77xx_RegWrite(fd,0x0015,0x000);
}

U32 tempSensorReadOut(int fd){
    U32 tempValue=0,ulRegValue;

    AFE77xx_RegWrite(fd,0x0015,0x080);
    AFE77xx_RegWrite(fd,0x08f0,0x01);
    AFE77xx_RegRead(fd,0x03ac,&ulRegValue);
    tempValue=ulRegValue;
    AFE77xx_RegRead(fd,0x03ad,&ulRegValue);
    tempValue=tempValue+(ulRegValue<<8);
    AFE77xx_RegWrite(fd,0x08f0,0x00);
    AFE77xx_RegWrite(fd,0x0015,0x000);
    return tempValue;
}

U32 checkAllAlarms(int fd){
    checkPllLockStatus(0,0);
    checkPllLockStatus(0,1);
    getJesdRxAlarms(0);
}

U32 getIntAlarms(int fd,int chNo){
    /*    chNo= 0:Alarm pin 0, 1:Alarm pin 1 */
    U32 readVal=0;
    U32 alarmValue=0;
    AFE77xx_RegWrite(fd,0x0015, 0x80);
    AFE77xx_RegRead(fd,0x01f9+0x1c8*chNo,&readVal);
    alarmValue=(readVal<<8);
    printf("RF PLL0: %d, DC PLL: %d, RF PLL2: %d, RF PLL3: %d,
RF PLL4: %d\n",readVal&1,(readVal>>1)&1,(readVal>>2)&
1,(readVal>>3)&1,(readVal>>4)&1);
    AFE77xx_RegRead(fd,0x01f8+0x1c8*chNo,&readVal);
    alarmValue=alarmValue+readVal;
    printf("JESD Alarms: %d\n",readVal&1);
    printf("TXA PAP: %d, TXB PAP: %d, TXC PAP: %d, TXD PAP: %d
\n",((readVal>>1)&1),((readVal>>2)&1),((readVal>>3)&
1),((readVal>>4)&1));
    printf("SPI Alarms: %d\n",((readVal>>5)&1));
    AFE77xx_RegWrite(fd,0x0015, 0x00);
    return alarmValue;
}

U32 overrideAlarmPin(int fd,int alarmNo, int overrideSel, int
overrideVal){
    U32 regVal=0;
    AFE77xx_RegWrite(fd,0x0015, 0x80);
    regVal=AFE77xx_RegRead(fd,0x571,&regVal);
    regVal=(regVal&0xff^(3<<((alarmNo*2)+

```

```

2)))) | (((overrideSel<<1)+overrideVal)<<((alarmNo*2)+2));
AFE77xx_RegWrite(fd,0x571,regVal);
AFE77xx_RegWrite(fd,0x0015, 0x00);
}

U32 overrideRelDetPin(int fd,int chNo, int overrideSel, int
overrideVal){
    U32 regVal=0;
    U8 offset=0;
    U32 address=0;
    offset=(chNo<<2);
    address=0x589+(offset>>3);
    offset=(chNo<<2)&0x7;
    AFE77xx_RegWrite(fd,0x0015, 0x80);
    if(chNo==0){
        AFE77xx_RegWrite(fd,0x4e8,0x42);
    }
    else if(chNo==1){
        AFE77xx_RegWrite(fd,0x4ec,0x42);
    }
    else if(chNo==2){
        AFE77xx_RegWrite(fd,0x4fc,0x42);
    }
    else if(chNo==3){
        AFE77xx_RegWrite(fd,0x504,0x42);
    }
    regVal=AFE77xx_RegRead(fd,address,&regVal);
    regVal=(regVal&(0xff^(3<<offset))) | (((overrideSel<<1)+overri
deVal)<<offset);
    AFE77xx_RegWrite(fd,address,regVal);
    AFE77xx_RegWrite(fd,0x0015, 0x00);
}

U32 overrideDigPkDetPin(int fd,int chNo, int overrideSel, int
overrideVal){
    U32 regVal=0;
    U8 offset=0;
    U32 address=0;
    address=0x586+(chNo>>1);
    if (chNo&1==1){
        offset=4;
    }
    else{
        offset=0;
    }

    AFE77xx_RegWrite(fd,0x0015, 0x80);
    if(chNo==0){
        AFE77xx_RegWrite(fd,0x4d4,0x22);
    }
    else if(chNo==1){
        AFE77xx_RegWrite(fd,0x4d0,0x22);
    }
}

```

```

    }
    else if(chNo==2){
        AFE77xx_RegWrite(fd,0x53c,0x22);
    }
    else if(chNo==3){
        AFE77xx_RegWrite(fd,0x4f0,0x22);
    }
    regVal=AFE77xx_RegRead(fd,address,&regVal);
    regVal=(regVal&(0xff^(3<<offset)))|(((overrideSel<<1)+overri
deVal)<<offset);
    AFE77xx_RegWrite(fd,address,regVal);
    AFE77xx_RegWrite(fd,0x0015, 0x00);
}

```

```

int sleepModeConfiguration(int fd, U8 rx_chain,U8 tx_chain,U8
fb_chain){
    /*

```

Making corresponding bit 1 for each of the parameter
will put it in sleep when the sleep pin goes high.

```

    rx_chain(=0xF)    4 bits, 1 each for one RX chain.
    tx_chain(=0xF)    4 bits, 1 each for one TX chain
    fb_chain(=0x3)    2 bits, 1 each for one FB chain.
*/

```

```

AFE77xx_RegWrite(fd,0x0013,0x10);
AFE77xx_RegWrite(fd,0x00a0,((rx_chain)+(tx_chain<<4)));
AFE77xx_RegWrite(fd,0x00a1,((fb_chain)+(0<<2)+(0<<7)));
AFE77xx_RegWrite(fd,0x00a2,8);
AFE77xx_RegWrite(fd,0x0193,0x39);
AFE77xx_RegWrite(fd,0x0192,0x00);
AFE77xx_RegWrite(fd,0x0191,0x00);
AFE77xx_RegWrite(fd,0x0190,0x00);
AFE77xx_RegWrite(fd,0x0013,0x00);
sysUsDelay(1000);
AFE77xx_RegWrite(fd,0x0013,0x10);
AFE77xx_RegWrite(fd,0x00a0,((rx_chain)+(tx_chain<<4)));
AFE77xx_RegWrite(fd,0x00a1,((fb_chain)+(0<<2)+(0<<7)));
AFE77xx_RegWrite(fd,0x00a2,8);
AFE77xx_RegWrite(fd,0x0193,0x3a);
AFE77xx_RegWrite(fd,0x0192,0x00);
AFE77xx_RegWrite(fd,0x0191,0x00);
AFE77xx_RegWrite(fd,0x0190,0x00);
AFE77xx_RegWrite(fd,0x0013,0x00);
}

```

```

void sleepEn(int fd, int sleepEn){
    /*
    Need to configure sleep mode using function
    "sleepModeConfiguration" before this function once.
    sleepEn=          0: Wake up.          1- sleep
    */
    U32 regVal=0;
    U32 temp=0;

```

```

AFE77xx_RegWrite(fd,0x0015,0x80);
AFE77xx_RegWrite(fd,0x0914,0x00);
AFE77xx_RegWrite(fd,0x0918,0x00);
regVal=AFE77xx_RegRead(fd,0x056d,&temp);
regVal=(regVal&0xf3)+(1<<3)+(sleepEn<<2);
AFE77xx_RegWrite(fd,0x056d,regVal);
AFE77xx_RegWrite(fd,0x0015,0x00);
sysUsDelay(2000);
}

int deepSleepModeConfiguration(int fd, U8 rx_chain,U8 tx_chain,U8
fb_chain,U8 pll,U8 serdes){
    /*
        Making corresponding bit 1 for each of the parameter
will put it in sleep when the sleep pin goes high.
        rx_chain(=0xF)    4 bits, 1 each for one RX chain.
        tx_chain(=0xF)    4 bits, 1 each for one TX chain
        fb_chain(=0x3)    2 bits, 1 each for one FB chain.
        pll(=0x1F) 5bits, 1 each for one pll
        serdes(=1) For putting Serdes in sleep
    */
    int ulRegValue = 0;
AFE77xx_RegWrite(fd,0x0015,0x80);
#ifdef TIENV
AFE77xx_RegWrite(fd,0x01d4,0x00);
AFE77xx_RegWrite(fd,0x0374,0x01);
#else
AFE77xx_RegWrite(fd,0x01d4,0x01);
AFE77xx_RegWrite(fd,0x0374,0x00);
#endif
AFE77xx_RegWrite(fd,0x0015,0x00);
sysUsDelay(1000);
for(int i=0;i<5;i++){
    if((pll&(1<<i))>0){
        AFE77xx_RegWrite(fd,0x0014,0x08<<i);
AFE77xx_RegRead(fd,0x005e,&ulRegValue);
AFE77xx_RegWrite(fd,0x0014,0x00);
        if((ulRegValue&0x80)==0){
            pll=(pll^(1<<i));
        }
    }
}

AFE77xx_RegWrite(fd,0x0015,0x80);
AFE77xx_RegWrite(fd,0x01d4,0x00);
AFE77xx_RegWrite(fd,0x0374,0x00);
AFE77xx_RegWrite(fd,0x0015,0x00);

AFE77xx_RegWrite(fd,0x0013,0x10);
AFE77xx_RegWrite(fd,0x00a0,((rx_chain&0xf)+((tx_chain&0xf)
<<4)));

```

```

AFE77xx_RegWrite(fd,0x00a1,((fb_chain&3)+((pll&0x1f)
<<2)+((pll&2)>>1)<<7)));
AFE77xx_RegWrite(fd,0x00a2,((pll&2)>>1)+((serdes&1)<<1)+
8));
AFE77xx_RegWrite(fd,0x0193,0x39);
AFE77xx_RegWrite(fd,0x0192,0x00);
AFE77xx_RegWrite(fd,0x0191,0x00);
AFE77xx_RegWrite(fd,0x0190,0x00);
AFE77xx_RegWrite(fd,0x0013,0x00);
sysUsDelay(1000);
AFE77xx_RegWrite(fd,0x0013,0x10);
AFE77xx_RegWrite(fd,0x0193,0x3a);
AFE77xx_RegWrite(fd,0x0192,0x00);
AFE77xx_RegWrite(fd,0x0191,0x00);
AFE77xx_RegWrite(fd,0x0190,0x00);
AFE77xx_RegWrite(fd,0x0013,0x00);
sysUsDelay(1000);
}

```

```

void deepSleepEn(int fd, int sleepEn){
/*
Need to configure sleep mode using function
"deepSleepModeConfiguration" before this function once.
sleepEn=      0: Wake up.      1- sleep
*/
U32 regVal=0;
U32 temp=0;

AFE77xx_RegWrite(fd,0x0015,0x80);
AFE77xx_RegWrite(fd,0x0914,0x00);
AFE77xx_RegWrite(fd,0x0918,0x00);
regVal=AFE77xx_RegRead(fd,0x056d,&temp);
regVal=(regVal&0xf3)+(1<<3)+(sleepEn<<2);
AFE77xx_RegWrite(fd,0x056d,regVal);
AFE77xx_RegWrite(fd,0x0015,0x00);
sysUsDelay(2000);

if(sleepEn==1){
int readReg,readValA,readValB;

AFE77xx_RegWrite(fd,0x0015,0x04);
AFE77xx_RegRead(fd,0x49fd,&readReg);
AFE77xx_RegRead(fd,0x49fd,&readReg);
AFE77xx_RegRead(fd,0x49fc,&readReg);
readReg = AFE77xx_RegRead(fd,0x49fc,&readReg);
AFE77xx_RegRead(fd,0x49ff,&readValA);
readValA = AFE77xx_RegRead(fd,0x49ff,&readValA);
AFE77xx_RegRead(fd,0x49fe,&readValB);
readValB = AFE77xx_RegRead(fd,0x49fe,&readValB);
int writeVal = (readValA&0xef)+((readReg&0x80)>>3);
AFE77xx_RegWrite(fd,0x49ff,writeVal);
AFE77xx_RegWrite(fd,0x49fe,readValB);
}
}

```

```

        AFE77xx_RegWrite(fd,0x0015,0x00);
    }
    if((sleepEn==0)){
        int pllLock,pllLockLost;

        AFE77xx_RegWrite(fd,0x0015,0x80);
        #ifdef TIENV
        AFE77xx_RegWrite(fd,0x01d4,0x00);
        AFE77xx_RegWrite(fd,0x0374,0x01);
        #else
        AFE77xx_RegWrite(fd,0x01d4,0x01);
        AFE77xx_RegWrite(fd,0x0374,0x00);
        #endif
        AFE77xx_RegWrite(fd,0x0015,0x00);
        sysUsDelay(1000);

        AFE77xx_RegWrite(fd,0x0014,0x10);
        pllLock = (AFE77xx_RegRead(fd,0x0062,&pllLock) &
0x80)>>7;
        pllLockLost = (AFE77xx_RegRead(fd,0x0063,
&pllLockLost) &0x20)>>1;
        AFE77xx_RegWrite(fd,0x0014,0x00);

        AFE77xx_RegWrite(fd,0x0015,0x80);
        AFE77xx_RegWrite(fd,0x01d4,0x00);
        AFE77xx_RegWrite(fd,0x0374,0x00);
        AFE77xx_RegWrite(fd,0x0015,0x00);

        if((pllLock==1) && (pllLockLost==0)){
            AFE77xx_RegWrite(fd,0x0013,0x20);
            AFE77xx_RegWrite(fd,0x0088,0x01);
            AFE77xx_RegWrite(fd,0x0013,0x00);
        }
    }
}

void overrideSleep(int fd, int overrideEn, int overrideVal, int
rxTDD, int txTDD, int fbTDD){
    /*
        overrideEn=          0: Control sleep using pins.          1-
Control sleep using SPI
        overrideVal=         1: Put in Sleep.0: WakeUp.
        rxTDD/txTDD=         0: No override
                                1: RX/TX AB override to 1
                                2: RX/TX CD override to 1
                                3: RX/TX ABCD override to 1
        fbTDD=                0: No override
                                1: FB1 override to 1
                                2: FB2 override to 1
                                3: FB1 and FB2 override to 1
    */
}

```

```

U32 regVal=0;
U32 temp=0;

AFE77xx_RegWrite(fd,0x0015,0x80);
AFE77xx_RegWrite(fd,0x0914,0x00);
AFE77xx_RegWrite(fd,0x0918,0x00);
regVal=AFE77xx_RegRead(fd,0x056d,&temp);
regVal=(regVal&0xf3)+(overrideEn<<3)+(overrideVal<<2);
AFE77xx_RegWrite(fd,0x056d,regVal);
AFE77xx_RegWrite(fd,0x0015,0x00);
sysUsDelay(2000);

if((overrideEn==1) && (overrideVal==0)){
    AFE77xx_RegWrite(fd,0x0014,0x04);
    if((rxTDD & 0x1)==1){
        AFE77xx_RegWrite(fd,0x126,0x1);
        AFE77xx_RegWrite(fd,0x127,0x1);
    }
    else{
        AFE77xx_RegWrite(fd,0x126,0x0);
        AFE77xx_RegWrite(fd,0x127,0x0);
    }
    if((rxTDD & 0x2)==2){
        AFE77xx_RegWrite(fd,0x12C,0x1);
        AFE77xx_RegWrite(fd,0x12D,0x1);
    }
    else{
        AFE77xx_RegWrite(fd,0x12C,0x0);
        AFE77xx_RegWrite(fd,0x12D,0x0);
    }
    if((txTDD & 0x1)==1){
        AFE77xx_RegWrite(fd,0x124,0x1);
        AFE77xx_RegWrite(fd,0x125,0x1);
    }
    else{
        AFE77xx_RegWrite(fd,0x124,0x0);
        AFE77xx_RegWrite(fd,0x125,0x0);
    }
    if((txTDD & 0x2)==2){
        AFE77xx_RegWrite(fd,0x12A,0x1);
        AFE77xx_RegWrite(fd,0x12B,0x1);
    }
    else{
        AFE77xx_RegWrite(fd,0x12A,0x0);
        AFE77xx_RegWrite(fd,0x12B,0x0);
    }
    if((fbTDD & 0x1)==1){
        AFE77xx_RegWrite(fd,0x128,0x1);
        AFE77xx_RegWrite(fd,0x129,0x1);
    }
    else{
        AFE77xx_RegWrite(fd,0x128,0x0);
    }
}

```



```

        AFE77xx_RegWrite(fd,0x129,0x0);
    }
    if((fbTDD & 0x2)==2){
        AFE77xx_RegWrite(fd,0x12E,0x1);
        AFE77xx_RegWrite(fd,0x12F,0x1);
    }
    else{
        AFE77xx_RegWrite(fd,0x12E,0x0);
        AFE77xx_RegWrite(fd,0x12F,0x0);
    }
    AFE77xx_RegWrite(fd,0x0014,0x00);
}
}

void overrideTdd(int fd,int overrideEn,int rxTdd,int fbTdd,int
txTdd){
    /*    overrideEn= 0:no override. control goes to pin, 1:TDD
is forced acoording to rxTdd, fbTdd and txTdd.
        rxTdd=0:RX ABCD are off, 1:RX AB is on, 2:RX CD is on,
3:RX ABCD are on
        fbTdd=0:FB 1&2 are off, 1:FB 1 is on, 2:FB 2 is on,
3:FB 1&2 are on
        txTdd=0:TX ABCD are off, 1:TX AB is on, 2:TX CD is on,
3:TX ABCD are on
    */
    AFE77xx_RegWrite(fd,0x0014,0x04);
    AFE77xx_RegWrite(fd,0x124,overrideEn);
    AFE77xx_RegWrite(fd,0x126,overrideEn);
    AFE77xx_RegWrite(fd,0x128,overrideEn);
    AFE77xx_RegWrite(fd,0x12a,overrideEn);
    AFE77xx_RegWrite(fd,0x12c,overrideEn);
    AFE77xx_RegWrite(fd,0x12e,overrideEn);
    AFE77xx_RegWrite(fd,0x125,(overrideEn&(txTdd&1)));
    AFE77xx_RegWrite(fd,0x127,(overrideEn&(rxTdd&1)));
    AFE77xx_RegWrite(fd,0x129,(overrideEn&(fbTdd&1)));
    AFE77xx_RegWrite(fd,0x12b,(overrideEn&((txTdd&2)>>1)));
    AFE77xx_RegWrite(fd,0x12d,(overrideEn&((rxTdd&2)>>1)));
    AFE77xx_RegWrite(fd,0x12f,(overrideEn&((fbTdd&2)>>1)));
    AFE77xx_RegWrite(fd,0x0014,0x00);
}

int controlBranchMute(int fd, U8 rx_chain,U8 tx_chain,U8 mute){
    /*
        Making corresponding bit 1 selects mute/unmte on those
branches
        U8 rx_chain      2 bits, 1 each for RXAB/CD chain.
        U8 tx_chain      2 bits, 1 each for TXAB/CD chain.
        U8 mute 1 bit, 1 mutes the selected branches, 0
unmutes the selected branches
    */
    if((rx_chain & 0x1)==1){
        if(mute==1){

```

```

AFE77xx_RegWrite(fd,0x0014,0x04);
AFE77xx_RegWrite(fd,0x00126,0x01);
AFE77xx_RegWrite(fd,0x00127,0x00);
AFE77xx_RegWrite(fd,0x0014,0x00);
AFE77xx_RegWrite(fd,0x0018,0x01);
AFE77xx_RegWrite(fd,0x0030,0x06);
AFE77xx_RegWrite(fd,0x0018,0x00);
}
else{
AFE77xx_RegWrite(fd,0x0018,0x01);
AFE77xx_RegWrite(fd,0x0030,0x00);
AFE77xx_RegWrite(fd,0x0018,0x00);
sysUsDelay(1000);
AFE77xx_RegWrite(fd,0x0014,0x04);
AFE77xx_RegWrite(fd,0x00126,0x01);
AFE77xx_RegWrite(fd,0x00127,0x00);
AFE77xx_RegWrite(fd,0x00127,0x01);
AFE77xx_RegWrite(fd,0x00127,0x00);
AFE77xx_RegWrite(fd,0x00126,0x00);
AFE77xx_RegWrite(fd,0x0014,0x00);
}
}
if(((rx_chain & 0x2)>>1)==1){
if(mute==1){
AFE77xx_RegWrite(fd,0x0014,0x04);
AFE77xx_RegWrite(fd,0x0012c,0x01);
AFE77xx_RegWrite(fd,0x0012d,0x00);
AFE77xx_RegWrite(fd,0x0014,0x00);
AFE77xx_RegWrite(fd,0x0018,0x01);
AFE77xx_RegWrite(fd,0x0038,0x06);
AFE77xx_RegWrite(fd,0x0018,0x00);
}
else{
AFE77xx_RegWrite(fd,0x0018,0x01);
AFE77xx_RegWrite(fd,0x0038,0x00);
AFE77xx_RegWrite(fd,0x0018,0x00);
sysUsDelay(1000);
AFE77xx_RegWrite(fd,0x0014,0x04);
AFE77xx_RegWrite(fd,0x0012c,0x01);
AFE77xx_RegWrite(fd,0x0012d,0x00);
AFE77xx_RegWrite(fd,0x0012d,0x01);
AFE77xx_RegWrite(fd,0x0012d,0x00);
AFE77xx_RegWrite(fd,0x0012c,0x00);
AFE77xx_RegWrite(fd,0x0014,0x00);
}
}
}
if((tx_chain & 0x1)==1){
if(mute==1){
AFE77xx_RegWrite(fd,0x0014,0x04);
AFE77xx_RegWrite(fd,0x00124,0x01);
AFE77xx_RegWrite(fd,0x00125,0x00);
AFE77xx_RegWrite(fd,0x0014,0x00);
}
}
}

```

```

AFE77xx_RegWrite(fd,0x0018,0x01);
AFE77xx_RegWrite(fd,0x0040,0x06);
AFE77xx_RegWrite(fd,0x0018,0x00);

}
else{
AFE77xx_RegWrite(fd,0x0018,0x01);
AFE77xx_RegWrite(fd,0x0040,0x00);
AFE77xx_RegWrite(fd,0x0028,0x00);
AFE77xx_RegWrite(fd,0x0018,0x00);
sysUsDelay(1000);
AFE77xx_RegWrite(fd,0x0014,0x04);

AFE77xx_RegWrite(fd,0x00128,0x01);
AFE77xx_RegWrite(fd,0x00129,0x00);
AFE77xx_RegWrite(fd,0x00129,0x01);
AFE77xx_RegWrite(fd,0x00129,0x00);
AFE77xx_RegWrite(fd,0x00128,0x00);

AFE77xx_RegWrite(fd,0x00124,0x01);
AFE77xx_RegWrite(fd,0x00125,0x00);
AFE77xx_RegWrite(fd,0x00125,0x01);
AFE77xx_RegWrite(fd,0x00125,0x00);
AFE77xx_RegWrite(fd,0x00124,0x00);
AFE77xx_RegWrite(fd,0x0014,0x00);

}
}
if(((tx_chain & 0x2)>>1)==1){
if(mute==1){
AFE77xx_RegWrite(fd,0x0014,0x04);
AFE77xx_RegWrite(fd,0x0012a,0x01);
AFE77xx_RegWrite(fd,0x0012b,0x00);
AFE77xx_RegWrite(fd,0x0014,0x00);
AFE77xx_RegWrite(fd,0x0018,0x01);
AFE77xx_RegWrite(fd,0x0044,0x06);
AFE77xx_RegWrite(fd,0x0018,0x00);

}
else{
AFE77xx_RegWrite(fd,0x0018,0x01);
AFE77xx_RegWrite(fd,0x0044,0x00);
AFE77xx_RegWrite(fd,0x0028,0x00);
AFE77xx_RegWrite(fd,0x0018,0x00);
sysUsDelay(1000);
AFE77xx_RegWrite(fd,0x0014,0x04);
AFE77xx_RegWrite(fd,0x00128,0x01);
AFE77xx_RegWrite(fd,0x00129,0x00);
AFE77xx_RegWrite(fd,0x00129,0x01);
AFE77xx_RegWrite(fd,0x00129,0x00);
AFE77xx_RegWrite(fd,0x00128,0x00);

AFE77xx_RegWrite(fd,0x0012a,0x01);
AFE77xx_RegWrite(fd,0x0012b,0x00);

```

```

        AFE77xx_RegWrite(fd,0x0012b,0x01);
        AFE77xx_RegWrite(fd,0x0012b,0x00);
        AFE77xx_RegWrite(fd,0x0012a,0x00);
        AFE77xx_RegWrite(fd,0x0014,0x00);
    }
}
if((tx_chain & 0x3)==3){
    if(mute==1){
        AFE77xx_RegWrite(fd,0x0014,0x04);
        AFE77xx_RegWrite(fd,0x00128,0x01);
        AFE77xx_RegWrite(fd,0x00129,0x00);
        AFE77xx_RegWrite(fd,0x0014,0x00);
        AFE77xx_RegWrite(fd,0x0018,0x01);
        AFE77xx_RegWrite(fd,0x0028,0x02);
        AFE77xx_RegWrite(fd,0x0018,0x00);
    }
    else{
        AFE77xx_RegWrite(fd,0x0018,0x01);
        AFE77xx_RegWrite(fd,0x0028,0x00);
        AFE77xx_RegWrite(fd,0x0018,0x00);
        sysUsDelay(1000);
        AFE77xx_RegWrite(fd,0x0014,0x04);
        AFE77xx_RegWrite(fd,0x00128,0x01);
        AFE77xx_RegWrite(fd,0x00129,0x00);
        AFE77xx_RegWrite(fd,0x00129,0x01);
        AFE77xx_RegWrite(fd,0x00129,0x00);
        AFE77xx_RegWrite(fd,0x00128,0x00);
        AFE77xx_RegWrite(fd,0x0014,0x00);
    }
}
}

int auxAdcInpSel(int fd, int INP,int INM){
    /*    INP= 0:Vref/2, 1:TP,2:AUX4,3:AUX5,4:AUX6,5:AVSS
        INM= 0:Vref/2, 1:TP,2:AUX1,3:AUX2,4:AUX3,5:AVSS
        Vref is typically 1.4V and it is generated internally
    */
    U32 writeVal=(INP<<5)+(INM<<2);
    AFE77xx_RegWrite(fd,0x0015,0x80);
    AFE77xx_RegReadWrite(fd,0x0102,writeVal,2,7);
    AFE77xx_RegWrite(fd,0x0015,0x00);
}

int readAuxAdc(int fd){
    /*    select input using "auxAdcInSel" function    */
    U32 regVal=0;
    AFE77xx_RegWrite(fd,0x0015,0x80);
    AFE77xx_RegWrite(fd,0x0100,0x01);
    AFE77xx_RegWrite(fd,0x0100,0x09);
    AFE77xx_RegWrite(fd,0x0100,0x01);
    AFE77xx_RegWrite(fd,0x0110,0x00);
    AFE77xx_RegWrite(fd,0x0100,0x03);
}

```

```

AFE77xx_RegWrite(fd,0x0110,0x01);
AFE77xx_RegWrite(fd,0x0100,0x00);
AFE77xx_RegRead(fd,0x010D,&regVal);
regVal=(regVal<<8)+AFE77xx_RegRead(fd,0x010C,&regVal);
AFE77xx_RegWrite(fd,0x0015,0x00);
printf("AUX ADC output code=%d",regVal);
return regVal;
}

int checkSysrefFlags(int fd){
/* Note: If FB CD is unused, do not consider FB CD */
U32 temp=0;
U32 readSysrefFlags[9]={0,0,0,0,0,0,0,0,0};
const char *flagDesc[9];
flagDesc[0]="Sysref to RX AB";
flagDesc[1]="Sysref to RX CD";
flagDesc[2]="Sysref to FB AB";
flagDesc[3]="Sysref to FB CD";
flagDesc[4]="Sysref to TX AB";
flagDesc[5]="Sysref to TX CD";
flagDesc[6]="Digital Clock";
flagDesc[7]="Sysref to Digital";
flagDesc[8]="Sysref to Analog";

AFE77xx_RegWrite(fd,0x0017,0x40);
readSysrefFlags[0]=(AFE77xx_RegRead(fd,0x0049,&temp)&0x3);
AFE77xx_RegWrite(fd,0x0017,0x80);
readSysrefFlags[1]=(AFE77xx_RegRead(fd,0x0049,&temp)&0x3);
AFE77xx_RegWrite(fd,0x0017,0x40);
readSysrefFlags[2]=((AFE77xx_RegRead(fd,0x0049,&temp)&
0x8)>>3);
AFE77xx_RegWrite(fd,0x0017,0x80);
readSysrefFlags[3]=((AFE77xx_RegRead(fd,0x0049,&temp)&
0x8)>>3);
AFE77xx_RegWrite(fd,0x0017,0x10);
readSysrefFlags[4]=AFE77xx_RegRead(fd,0x003c,&temp);
AFE77xx_RegWrite(fd,0x0017,0x20);
readSysrefFlags[5]=AFE77xx_RegRead(fd,0x003c,&temp);
AFE77xx_RegWrite(fd,0x0017,0x00);
AFE77xx_RegWrite(fd,0x0015,0x01);
readSysrefFlags[6]=((AFE77xx_RegRead(fd,0x00a8,&temp)&
0x10)>>4);
readSysrefFlags[7]=((AFE77xx_RegRead(fd,0x00a8,&temp)&
0x20)>>5);
AFE77xx_RegWrite(fd,0x0015,0x00);
AFE77xx_RegWrite(fd,0x0011,0x01);
readSysrefFlags[8]=AFE77xx_RegRead(fd,0x053d,&temp);
readSysrefFlags[8]=(readSysrefFlags[8]
<<8)+AFE77xx_RegRead(fd,0x053c,&temp);
AFE77xx_RegWrite(fd,0x0011,0x00);
U32 expectedSysrefFlags[9]={3,3,1,1,7,7,1,1,7};
for(int i=0;i<9;i=i+1){

```

```
        printf("%s, read :%d, expected: %d  
\n", flagDesc[i], readSysrefFlags[i], expectedSysrefFlags[i]);  
    }  
}
```