

Program and control PA5750 Audio CODEC using the PA5750 Evaluation Board

1. Introduction

The PA5750 is a low power audio CODEC. The PA5750 requires communication to a μ C or a processor via I²C or SPI. This document will explain how to program and control the PA5750.

All programs in this document are verified by the PA5750 evaluation board. The MCU used is a P89LV51RD2, software is based on KEIL C51. It is programmed using C language, which can be easily transferred to another MCU platform.

The main contents are as followed:

- o I/O to simulate I2C and SPI
- o Using ADC
- o Using DAC
- o Using ADC, DAC
- o Using MIXER only to complete ANALOG IN TO ANALOG OUT (DAC was Bypassed)

2. I/O to simulate I²C, SPI

PA5750 provides I2C, SPI interface, which are defined as:

PA5750-QFN32		
I2C	AD0	PIN30
	SDA	PIN31
	SCL	PIN32
SPI	CS	PIN30
	DIN	PIN31
	CLK	PIN32

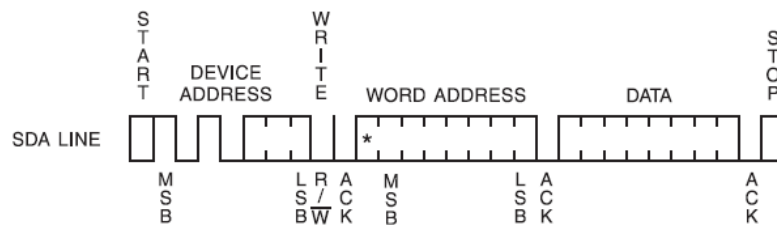
The clocking sequence of I²C in PA5750 is the same as that of the standard I²C. In applications, if I²C signals are provided by the processor, they can be used to communicate with PA5750.

User can also use I/O to simulate I²C sequence to communicate with PA5750.

Note the read/write in PA5750 is through single byte in I²C protocol.

If using SPI bus to communicate with the PA5750, only WRITE is possible, READ not possible.

Byte Write



Random Read

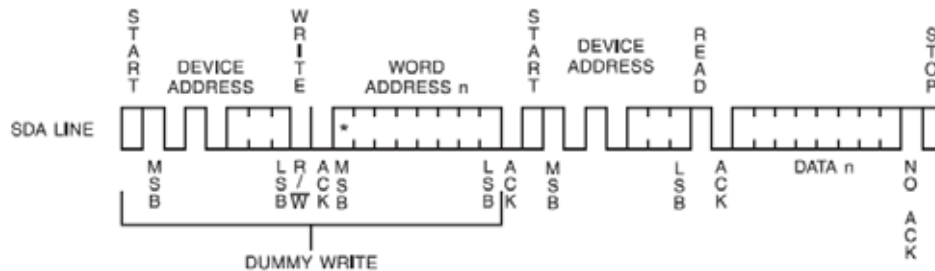


Fig.1. I²C Read/write clock

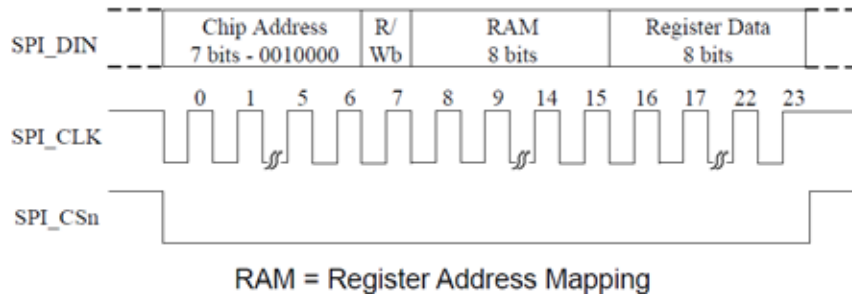


Fig. 2. WRITE in SPI

	Address							
I ² C	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
	0	0	1	0	0	0	AD0	R/W
SPI	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
	0	0	1	0	0	0	0	0

Program for I/O to simulate I²C, SPI (P0.5, P0.6, P0.7 in P89LV51RD2 are used as SDA, SCL, A0).

```
#include <reg52.h>
#include <Audio.h>

sbit bSDA    = P0^5;
sbit bSCL    = P0^6;
sbit bCS     = P0^7;
sbit bCOMSEL = P3^5;
//-----
#define nRD   0x01
#define nWR   0x00
//-----
extern void Delay(unsigned char); //delay program
//-----
UCHAR idata RDDATA;
/*****
** Function name:          I2CCLK
**
** Descriptions:         Generate I2C CLOCK
**
** Input parameters:     UCHAR i =1, CLOCK=High Level
                        =0, CLOCK=Low Level
** Returned value:      None
**
** Used global variables: None
** Calling modules:     None
**
** Created by:           ProTek Analog
** Created Date:        02/25/2010
**
** Modified by:
** Modified date:
**
*****/
void I2CCLK(UCHAR i)
{
    if(i)
```

```

{
  bSCL = 1;
}
else
{
  bSCL = 0;
}
}
/*****
** Function name:          I2CDATA
**
** Descriptions:          output I2C SDA
**
** Input parameters:      UCHAR i = 1, DATA=High Level
                        = 0, DATA=Low Level
**
** Returned value:        None
**
** Used global variables: None
** Calling modules:       None
**
** Created by:            ProTek Analog
** Created Date:          02/25/2010
** -----
** Modified by:
** Modified date:
** -----
*****/
void I2CDATA(UCHAR i)
{
  if(i)
  {
    bSDA = 1;
  }
  else
  {
    bSDA = 0;
  }
}
/*****
** Function name:          I2CSTART
**
** Descriptions:          Generate I2C START condition
**
** Input parameters:      None
** Returned value:        None
**
** Used global variables: None
** Calling modules:       None
**
** Created by:            ProTek Analog
** Created Date:          02/25/2010
** -----
** Modified by:
** Modified date:
** -----
*****/
void I2CSTART(void)
{
  I2CDATA(1);
  Delay(1);
  I2CCLK(1);
  Delay(1);
  I2CDATA(0);
}

```



```

Delay(1);
I2CCLK(0);
}
/*****
** Function name:          I2CSTOP
**
** Descriptions:         Generate I2C STOPcondition
**
** Input parameters:     None
** Returned value:       None
**
** Used global variables: None
** Calling modules:      None
**
** Created by:           ProTek Analog
** Created Date:         02/25/2010
** -----
** Modified by:
** Modified date:
** -----
*****/
void I2CSTOP(void)
{
I2CDATA(0);
Delay(1);
I2CCLK(1);
I2CDATA(0);
Delay(1);
I2CDATA(1);
}
/*****
** Function name:          I2CACK
**
** Descriptions:         Check I2C ACK
**
** Input parameters:     None
** Returned value:       UCHAR i , i =0,ACK Valid
                        i =1,ACK Invalid
**
** Used global variables: None
** Calling modules:      None
**
** Created by:           ProTek Analog
** Created Date:         02/25/2010
** -----
** Modified by:
** Modified date:
** -----
*****/
UCHAR I2CACK(void)
{
UCHAR i;

Delay(1);
I2CCLK(1);
Delay(1);
if(bSDA)
{
i=1;
}
else
{
i=0;
}
}

```



```

}
Delay(1);
I2CCLK(0);

return(i);
}
/*****
** Function name:                I2CNAK
**
** Descriptions:                I2C NAK
**
** Input parameters:            None
** Returned value:              None
**
** Used global variables:       None
** Calling modules:             None
**
** Created by:                  ProTek Analog
** Created Date:                02/25/2010
** -----
** Modified by:
** Modified date:
** -----
*****/
void I2CNAK(void)
{
I2CCLK(0);
Delay(1);
I2CCLK(1);
Delay(1);
I2CCLK(0);
}

/*****
** Function name:                I2CSENDDATA
**
** Descriptions:                I2Cwrite
**
** Input parameters:            UCHAR BData
** Returned value:              None
**
** Used global variables:       None
** Calling modules:             None
**
** Created by:                  ProTek Analog
** Created Date:                02/25/2010
** -----
** Modified by:
** Modified date:
** -----
*****/
void I2CSENDDATA(UCHAR cData)
{
UCHAR j;
for(j=0;j<8;j++)
{
Delay(1);

if((cData<<j)&0x80) //DATA=1
{
I2CDATA(1);
}
else //DATA=0;

```



```

{
I2CDATA(0);
}
Delay(1);
I2CCLK(1);
Delay(1);
I2CCLK(0);
}
}
}
/*****
** Function name:                I2CRECDATA
**
** Descriptions:                I2C read
**
** Input parameters:           None
** Returned value:             UCHAR BData
**
** Used global variables:      None
** Calling modules:            None
**
** Created by:                  ProTek Analog
** Created Date:                02/25/2010
**
-----
** Modified by:
** Modified date:
**
-----
*****/
UCHAR I2CRECDATA(void)
{
    UCHAR i,RecD;

    RecD=0;
    for(i=0;i<8;i++)
    {
        Delay(1);
        I2CCLK(1);
        Delay(1);

        if(bSDA)
        {
            RecD=RecD | (1<<(7-i));
        }
        else
        {
            RecD=RecD | 0x00;
        }
        I2CCLK(0);
    }
    return(RecD);
}
/*****
** Function name:                I2CWRNBYTE
**
** Descriptions:                Write a byte through I2C
**
** Input parameters:           UCHAR ChipID , chip address
                                UCHAR RegAdr , register address
                                UCHAR Bdata , data to be written
**
** Returned value:             UCHAR valid, = 0, I2C Successful
                                = 1, Failed
**
** Used global variables:      None

```



** Calling modules: None
**
** Created by: ProTek Analog
** Created Date: 02/25/2010

** Modified by:
** Modified date:
**

*****/

UCHAR I2CWRNBYTE(UCHAR ChipID, UCHAR RegAdr, UCHAR Bdata)

```
{
I2CSTART();
I2CSENDATA(ChipID);
if(I2CACK()==0)
{
I2CSENDATA(RegAdr);
if(I2CACK()==0)
{
I2CSENDATA(Bdata);
if(I2CACK()==0)
{
I2CSTOP();
return(0);
}
}
else
{
return(1); //Non ACK Error
}
}
else
{
return(2); //Non ACK,Error
}
}
else
{
return(3); //Non ACK,Error
}
}
```

*****/

** Function name: I2CRDNBYTE
**
** Descriptions: Read a byte through I2C, and memory the READ
** in global variables RDDATA
** Input parameters: UCHAR ChipID , Chip address
 UCHAR RegAdr , register address
** Returned value: UCHAR valid , = 0, I2C Successful
 = 1, Failed

** Used global variables: None
** Calling modules: None
**

** Created by: ProTek Analog
** Created Date: 02/25/2010

** Modified by:
** Modified date:
**

*****/

UCHAR I2CRDNBYTE(UCHAR ChipID,UCHAR RegAdr)

```
{
I2CSTART();
I2CSENDATA(ChipID);
```



```

if(I2CACK()==0)
{
I2CSENDATA(RegAdr);
if(I2CACK()==0)
{
I2CSTART();
I2CSENDATA(ChipID|0x01); //Read
if(I2CACK()==0)
{
RDDATA=I2CRECDATA();
I2CNAK();
I2CSTOP();
return(0);
}
}
else
{
I2CSTOP();
return(1); //Non ACK Error
}
}
else
{
I2CSTOP();
return(2); //Non ACK,Error
}
}
else
{
I2CSTOP();
return(3); //Non ACK,Error
}
}

```

```

/*****
** Function name:          SPISENDATA
**
** Descriptions:         SPI WRITE
**
** input parameters:     UCHAR BData
** Returned value:       None
**
** Used global variables: None
** Calling modules:      None
**
** Created by:           ProTek Analog
** Created Date:        02/25/2010
**
** Modified by:
** Modified date:
**
*****/

```

```

void SPISENDATA(UCHAR cData)
{
UCHAR j;

for(j=0;j<8;j++)
{
I2CCLK(0);
if((cData<<j)&0x80) //DATA=1
{
I2CDATA(1);
}
else //DATA=0;

```



```

{
I2CDATA(0);
}
Delay(1);
I2CCLK(1);
Delay(1);
}
}
}
/*****
** Function name:          SPI_WriteNByte
**
** Descriptions:          Write a byte through SPI
**
** Input parameters:      UCHAR ChipID, 0X20,chip address
                          UCHAR RAM, the Register Address
                          UCHAR cData, the data written
** Returned value:       None
**
** Used global variables: None
** Calling modules:      None
**
** Created by:            ProTek Analog
** Created Date:         02/25/2010
** -----
** Modified by:
** Modified date:
** -----
*****/
void SPI_WriteNByte(UCHAR ChipID, UCHAR cRAM, UCHAR cData)
{
bCS=0;
SPISENDDATA(ChipID);
SPISENDDATA(cRAM);
SPISENDDATA(cData);
Delay(1);
bCS=1;
}
/*****
** Function name:          WriteChip
**
** Descriptions:          Write PA5750, the program decides whether it is
                          I2C or SPI by P3.5, P3.5 = 0 for I2C.
** input parameters:      UCHAR RAM, the Register Address
                          UCHAR cData, the data written
** Returned value:       0, SUCESFUL
                          1, FAILED
**
** Used global variables: None
** Calling modules:      None
**
** Created by:            ProTek Analog
** Created Date:         02/25/2010
** -----
** Modified by:
** Modified date:
** -----
*****/
UCHAR WriteChip(UCHAR RAM,UCHAR cData)
{
UCHAR i;
i=0;
if(bCOMSEL) //SPI

```



```

{
  SPI_WriteNByte(0x20, RAM, cData);
  return(0);
}
else //I2C
{
  if(bCS)
  {
    i=1;
  }
  else
  {
    i=0;
  }
  i=i<<1;
  if(I2CWRNBYTE(0x20|i, RAM, cData)==0)
  {
    return(0);
  }
  else
  {
    return(1);
  }
}
}
}
/*****
** Function name:          ReadChip
**
** Descriptions:         Read a byte from PA5750 through I2C
**
** input parameters:
                        UCHAR RAM ,the Register Address
** Returned value:      0, SUCESFUL, 1, FAILED
**
** Used global variables:  None
** Calling modules:      None
**
** Created by:           ProTek Analog
** Created Date:        02/25/2010
** -----
** Modified by:
** Modified date:
** -----
*****/
UCHAR ReadChip(UCHAR RAM)
{
  UCHAR i;
  if(bCS)
  {
    i=1;
  }
  else
  {
    i=0;
  }
  i=i<<1;
  if(I2CRDNBYTE(0x20|i, RAM)==0)
  {
    return(0);
  }
  else
  {
    return(1);
  }
}

```



3. Using ADC

When only using the ADC in PA5750, the registers should be set according to the followed table.
The functions of I²C/SPI in Sect.2 is used for read/write registers in PA5750.

REGISTER	WRITE VALUE TO REGISTER(HEX)	REMARK
R2	FFH	STOP ADC&DAC STATE MACHINE AND DLL
R8	00Hor 80H	SET CHIP TO SLAVE OR MASTER MODE user defined
R0	05H	Enrefr=1,PA5750 in Playback and Record Mode.
R1	40H	Pdn_ana=0,ibiasgen_pdn=0,Vrefr_LO=0,pdn_Vrefbuf=0, start the analog bias accordingly
R3	08H	Pdn_MICB=1,Pdn_AINL=0,Pdn_AINR=0,Pdn_ADCL=0,Pdn_ADCR=0,pdn_adcbiasgen=0, if using MICROPHONE IN, set pdn_MICB to 0.
R4	C0H	Pdn_DACL=1,Pdn_DACR=1, LOUT1=0, ROUT1=0, LOUT2=0, ROUT2=0, OUT3=0, MONO=0, shut down output of DAC and analog output
R10	00H	ADC Analog IN is L/R INPUT1, user defined
R9	00H	MicAmpL=0dB,MicAmpR=0dB. R9 is used to set the gain of PGA of front ADC, user can set it according the application.
R11	02H	User defined
R12	00H	ADC IS I2S-24BIT(user defined)
R13	02H	ADC IS FS SINGLE SPEED MODE,MCLK/LRCK RAITO=256,user defined
R16	00H	R16, R17 is to set ADC VOLUME, user defined.
R17	00H	
R2	55H	Start ADC and DEM, start ADC STATE MACHINE and DLL

The followed is a sample of ADC driving program.

```
void ADC_START(void)
{
//-----
//Chip Control and Power Management registers
//-----
WriteChip(0x02,0xFF); //R2=0XF3, this is a must, to stop the state indicator and the //clock of ADC,, then parameters can be set-
able.
if(SPDIF.MS_STA==MASTER)
{
WriteChip(0x08,0x00); //PA5750 is set to SLAVE mode
}
else
{
WriteChip(0x08,0x80); //PA5750 is set to MASTERMode
}
WriteChip(0x00,0x05); // set PA5750 to PLAYBACK & RECORD mode, and EnRefr=1
WriteChip(0x01,0x40); //pdn_ana=0,ibiasgen_pdn=0
WriteChip(0x03,0x08); //ADC Power-up
WriteChip(0x04,0xc0); //ADC Power-down
//-----
//ADC Control registers
//-----
WriteChip(0x09,0x00); //ADC L/R PGA = 0dB
//-----
WriteChip(0x0A,0x00); //The inputs of LIN1/RIN1, LIN2/RIN2, LIN3/RIN3in PA5750
// is corresponding to LIN1/RIN1, LIN2/RIN2, LIN3/RIN3 in
//R10
//-----
WriteChip(0x0B,0x02); //STERO ADC
switch(SPDIF.ADCSFI)
```



```

{
case LJ :WriteChip(0x0C,0x01); //LEFT JUSTIFIED-24BIT
    break;
case I2S:WriteChip(0x0C,0x00); //I2S-24BIT
    break;
case RJ :WriteChip(0x0C,0x02); //RIGHT JUSTIFIED-24BIT
    break;
default :WriteChip(0x0C,0x02); //RIGHT JUSTIFIED-24BIT
    break;

}
switch(SPDIF.ADCRATIO)
{
case R256:WriteChip(0x0D,0x02); //SINGLE SPEED,RATIO=256
    break;
case R128:WriteChip(0x0D,0x00); //SINGLE SPEED,RATIO=128
    break;
case R512:WriteChip(0x0D,0x04); //SINGLE SPEED,RATIO=512
    break;
default :WriteChip(0x0D,0x02); //SINGLE SPEED,RATIO=256
    break;

}
WriteChip(0x10,0x00); //ADC Left Volume=0db
WriteChip(0x11,0x00); //ADC Right Volume=0db
WriteChip(0x02,0x55); //START ADC DLL AND STATE MACHINE
}

```

4. Using DAC

When only DAC in PA5750 is used, the registers must be set according to the following table. Hereafter, the I2C/SPI functions in Sect. 2 are used to READ/WRITE registers in PA5750.

REGISTER	WRITE VALUE TO REGISTER(HEX)	REMARK
R2	FFH	STOP ADC&DAC STATE MACHINE AND DLL
R8	00H or 80H	SET CHIP TO SLAVE OR MASTER MODE, user defined
R0	05H	Enrefr=1,PA5750in Playback and Record Mode.set the PA5750 in correct mode.
R1	40H	Pdn_ana=0,ibiasgen_pdn=0,Vrefr_LO=0,pdn_Vrefbuf=0 starting the analog bias
R3	FCH	Shutdown analog input and ADC, MICROPHONE BOOST
R4	3FH	Start ADC and all analog output
R23	00H	DAC IS I2S-24BIT
R24	02H	DAC IS FS SINGLE SPEED MODE,MCLK/LRCK RAITO=256
R2	AAH	Start DAC STATE MACHINE and DLL
R25	E2H	SOFT RAMP SET
R26	00H	LEFT DAC VOLUME=0DB
R27	00H	RIGHT DAC VOLUME=0DB
R38	00H	LMIXSEL,RMIXSEL TO DEFAULT
R39	B8H	LDAC TO LMIXER
R40	38H	
R41	38H	RDAC TO RMIXER
R42	B8H	
R43	B8H	UNUSE MONO MIXER
R44	38H	
R45	90H	VROI=1,OUT3=ROUT1
R46	1EH	LOUT1 VOLUME=0DB
R47	1EH	ROUT1 VOLUME=0DB
R48	1EH	LOUT2 VOLUME=0DB
R49	1EH	ROUT2 VOLUME=0DB
R50	1EH	MONO VOLUME=0DB

A sample of DAC driving program:

```
void DAC_START(void)
{
    UCHAR i;

    //-----
    //Chip Control and Power Management registers
    //-----
    WriteChip(0x02,0xFF); //R2=0XF3, this is a must, to stop the state indicator and the //clock of ADC, then parameters can be
                           //settable.
    if(SPDIF.MS_STA==MASTER)
    {
        WriteChip(0x08,0x00); // set PA5750 to SLAVE mode
    }
    else
    {
        WriteChip(0x08,0x80); //PA5750 to MASTER
    }
    WriteChip(0x00,0x05); // set PA5750 PLAYBACK & RECORD mode, and EnRefr=1
    WriteChip(0x01,0x40); //pdn_ana=0,ibiasgen_pdn=0
    WriteChip(0x03,0xFC); //Disable ADC AND Analog input
    WriteChip(0x04,0x3F); //DAC & Analog Output Power-up
    //-----
    //DAC CONTROL registers
    //-----
    switch(SPDIF.DACSFI)
    {
    case LJ :WriteChip(0x17,0x01); //LEFT JUSTIFIED-24BIT
             break;
    case I2S:WriteChip(0x17,0x00); //I2S-24BIT
             break;
    case RJ :WriteChip(0x17,0x02); //RIGHT JUSTIFIED-24BIT
             break;
    default :WriteChip(0x17,0x02); //RIGHT JUSTIFIED-24BIT
             break;

    }
    switch(SPDIF.DACRATIO)
    {
    case R256:WriteChip(0x18,0x02); //SINGLE SPEED,RATIO=256
             break;
    case R128:WriteChip(0x18,0x00); //SINGLE SPEED,RATIO=128
             break;
    case R512:WriteChip(0x18,0x04); //SINGLE SPEED,RATIO=512
             break;
    default :WriteChip(0x18,0x02); //SINGLE SPEED,RATIO=256
             break;

    }
    //-----
    //CHIP CONTROL AND POWER MANAGEMENT registers
    //-----
    WriteChip(0x02,0xAA); //START DAC DLL and State Machine
    //-----
    //DAC CONTROL registers
    //DAC VOLUME REGISTER
    //-----
    WriteChip(0x19,0xFA); //BOTH CHANNEL VOLUME CONTROL BY REGISTER

    ReadChip(0x1A); //0X1A,Disable Zero-Cross Check
    DAC_VOLL=RDDATA-1;
    ReadChip(0x1B);
    DAC_VOLR=RDDATA-1;
}
```

```

for(i=DAC_VOLL;i>0;i--)
{
WriteChip(0x1A,i); //Step-up DAC Volume
//Delay(1000);
}
WriteChip(0x1A,0x00); //DAC Left Volume=0db
WriteChip(0x1B,0x00); //DAC Right Volume=0db

WriteChip(0x19,0xE2); //SOFT RAMP RATE=128LRCKS/STEP,DISABLE
//ZERO-CROSS CHECK

//-----
//DAC CONTROL registers
//Analog OUT Volume registers
//-----
WriteChip(0x26,0x00); //LMIXSEL AND RMIXSEL SELECT
WriteChip(0x27,0xB8); //Left DAC TO Left MIXER
WriteChip(0x28,0x38);
WriteChip(0x29,0x38);
WriteChip(0x2A,0xB8); //RIGHT DAC TO RIGHT MIXER
WriteChip(0x2B,0xB8); //Left DAC TO Mono-MIXER ,
WriteChip(0x2C,0x38); //

WriteChip(0x2D,0x90); //ROUTINV=1,OUT3=VREF,VROI=1,ENABLE //HEADPHONE SWITCH
//-----
// set the VOLUME register, set to soft ramp up
//in actual applications, user can set the VOLUME to a certain value
//-----
ReadChip(0x2E); // READ LO1 , RO1 , LO2 , RO2 , MONOOUTVOLUME //control registers
LO1VOL=RDDATA;
ReadChip(0x2F);
RO1VOL=RDDATA;

ReadChip(0x30);
LO2VOL=RDDATA;
ReadChip(0x31);
RO2VOL=RDDATA;

ReadChip(0x32);
MONOVOL=RDDATA;

if(LO1VOL<RO1VOL)
{
i=LO1VOL;
}
else
{
i=RO1VOL;
}
for(;i<0x1F;i++)
{
WriteChip(0x2E,i); //Step-up LO1 Volume
WriteChip(0x2F,i); //Step-up RO1 Volume
}

if(LO2VOL<RO2VOL)
{
i=LO2VOL;
}
else
{
i=RO2VOL;
}

```

```

}
for(;i<0x1F;i++)
{
WriteChip(0x30,i); //Step-up LO2 Volume
WriteChip(0x31,i); //Step-up RO2 Volume
}

for(i=MONOVOL;i<0x1F;i++)
{
WriteChip(0x32,i); //Step-up MONOOUT Volume
}

```

5. Using DAC and ADC simultaneously

When using ADC and DAC in PA5750 simultaneously, PA5750 can be set to work in Playback and Record mode, the registers must be set according the followed table.
 Hereafter, the I²C/SPI functions in Sect. 2 are used to READ/WRITE registers in PA5750.

REGISTER	WRITE VALUE TO REGISTER(HEX)	REMARK
R2	FFH	STOP ADC&DAC STATE MACHINE AND DLL
R8	00H or 80H	SET CHIP TO SLAVE OR MASTER MODE, user defined
R0	05H	Enrefr=1, PA5750 in Playback and Record Mode. Set PA5750 to the correct working MODE
R1	40H	Pdn_ana=0,ibiasgen_pdn=0,Vrefr_LO=0,pdn_Vrefbuf=0, start the analog bias
R3	08H	Shutdown analog input and ADC, MICROPHONE BOOST, if MICROPHONE IN is needed, set pdn_MICB to 0
R4	3FH	Start ADC and all analog output
R10	00H	ADC Analog IN is L/R INPUT1, user defined
R9	00H	MicAmpL=0dB,MicAmpR=0dB. R9 is used to set the gain of PGA of front ADC, user can set it according the application.
R11	02H	User defined
R12	00H	ADC IS I2S-24BIT, user defined
R13	02H	ADC IS FS SINGLE SPEED MODE,MCLK/LRCK RAITO=256, user defined
R16	00H	R16□R17 are to set ADC VOLUME, user defined
R17	00H	
R23	00H	DAC IS I2S-24BIT
R24	02H	DAC IS FS SINGLE SPEED MODE,MCLK/LRCK RAITO=256
R2	00H	Start ADC&DAC STATE MACHINE and DLL
R25	E2H	SOFT RAMP SET
R26	00H	LEFT DAC VOLUME=0DB
R27	00H	RIGHT DAC VOLUME=0DB
R38	00H	LMIXSEL,RMIXSEL TO DEFAULT
R39	B8H	LDAC TO LMIXER
R40	38H	
R41	38H	RDAC TO RMIXER
R42	B8H	
R43	B8H	UNUSE MONO MIXER
R44	38H	
R45	90H	VROI=1,OUT3=ROUT1
R46	1EH	LOUT1 VOLUME=0DB
R47	1EH	ROUT1 VOLUME=0DB
R48	1EH	LOUT2 VOLUME=0DB
R49	1EH	ROUT2 VOLUME=0DB
R50	1EH	MONO VOLUME=0DB

A sample of the driving program.

```

WriteChip(0x02,0xFF); //R2=0XF3, a must
WriteChip(0x00,0x05); //Set PA5750 to PLAYBACK & RECORD MODE, EnRefr=1
WriteChip(0x01,0x40); //pdn_ana=0,ibiasgen_pdn=0
WriteChip(0x03,0x08); //ADC Power-up
WriteChip(0x04,0x3F); //DAC & Analog Output Power-up
if(SPDIF.MS_STA==MASTER)
{
  WriteChip(0x08,0x00); //PA5750 to SLAVE mode
}
else
{
  WriteChip(0x08,0x80); //PA5750 to MASTER mode
}
//-----
//ADC Control registers
//-----
WriteChip(0x09,0x00); //ADC L/R PGA = 0dB
WriteChip(0x0A,0x050); //input of ADC from LEFT/RIGHT channel, LIN1/RIN1
WriteChip(0x0B,0x02); //STEREO ADC
switch(SPDIF.ADCSFI)
{
  case LJ :WriteChip(0x0C,0x01); //LEFT JUSTIFIED-24BIT
    break;
  case I2S:WriteChip(0x0C,0x00); //I2S-24BIT
    break;
  case RJ :WriteChip(0x0C,0x02); //RIGHT JUSTIFIED-24BIT
    break;
  default :WriteChip(0x0C,0x02); //RIGHT JUSTIFIED-24BIT
    break;
}
switch(SPDIF.ADCRATIO)
{
  case R256:WriteChip(0x0D,0x02); //SINGLE SPEED,RATIO=256
    break;
  case R128:WriteChip(0x0D,0x00); //SINGLE SPEED,RATIO=128
    break;
  case R512:WriteChip(0x0D,0x04); //SINGLE SPEED,RATIO=512
    break;
  default :WriteChip(0x0D,0x02); //SINGLE SPEED,RATIO=256
    break;
}
WriteChip(0x10,0x00); //ADC Left Volume=0db
WriteChip(0x11,0x00); //ADC Right Volume=0db
//-----
//DAC CONTROL registers
//-----
switch(SPDIF.DACSF1)
{
  case LJ :WriteChip(0x17,0x01); //LEFT JUSTIFIED-24BIT
    break;
  case I2S:WriteChip(0x17,0x00); //I2S-24BIT
    break;
  case RJ :WriteChip(0x17,0x02); //RIGHT JUSTIFIED-24BIT
    break;
  default :WriteChip(0x17,0x02); //RIGHT JUSTIFIED-24BIT
    break;
}
switch(SPDIF.DACRATIO)
{
  case R256:WriteChip(0x18,0x02); //SINGLE SPEED,RATIO=256

```

```
break;
case R128:WriteChip(0x18,0x00); //SINGLE SPEED,RATIO=128
break;
case R512:WriteChip(0x18,0x04); //SINGLE SPEED,RATIO=512
break;
default :WriteChip(0x18,0x02); //SINGLE SPEED,RATIO=256
break;

}
//-----
//CHIP CONTROL AND POWER MANAGEMENT registers
//-----
WriteChip(0x02,0x00); //START ADC & DAC DLL and State Machine
//-----
//DAC CONTROL registers
//DAC VOLUME REGISTER
//-----
WriteChip(0x19,0xFA); //BOTH CHANNEL VOLUME CONTROL BY REGISTER

ReadChip(0x1A) //0X1A,Disable Zero-Cross Check;
DAC_VOLL=RDDATA-1;
ReadChip(0x1B);
DAC_VOLR=RDDATA-1;

for(i=DAC_VOLL;i>0;i--)
{
WriteChip(0x1A,i); //Step-up DAC Volume
//Delay(1000);
}
WriteChip(0x1A,0x00); //DAC Left Volume=0db
WriteChip(0x1B,0x00); //DAC Right Volume=0db

WriteChip(0x19,0xE2); //SOFT RAMP RATE=128LRCKS/STEP, DISABLE ZERO-
//CROSS CHECK

//-----
//DAC CONTROL registers
//Analog OUT Volume register
//-----
WriteChip(0x27,0xB8); //Left DAC TO Left MIXER
WriteChip(0x28,0x38);
WriteChip(0x29,0x38);
WriteChip(0x2A,0xB8); //RIGHT DAC TO RIGHT MIXER
WriteChip(0x2B,0xB8); //Left DAC TO Mono-MIXER
WriteChip(0x2C,0x38);

WriteChip(0x2D,0x90); //ROUTINV=1,OUT3=VREF,VROI=1,ENABLE HEADPHONE
//SWITCH

ReadChip(0x2E); //read LO1, RO1, LO2, RO2, MONOOUTVOLUME register
LO1VOL=RDDATA;
ReadChip(0x2F);
RO1VOL=RDDATA;

ReadChip(0x30);
LO2VOL=RDDATA;
ReadChip(0x31);
RO2VOL=RDDATA;

ReadChip(0x32);
MONOVOL=RDDATA;

if(LO1VOL<RO1VOL)
```

```

{
i=LO1VOL;
}
else
{
i=RO1VOL;
}
for(;i<0x1F;i++)
{
WriteChip(0x2E,i); //Step-up LO1 Volume
WriteChip(0x2F,i); //Step-up RO1 Volume
//Delay(1000);
}

if(LO2VOL<RO2VOL)
{
i=LO2VOL;
}
else
{
i=RO2VOL;
}
for(;i<0x1F;i++)
{
WriteChip(0x30,i); //Step-up LO2 Volume
WriteChip(0x31,i); //Step-up RO2 Volume
//Delay(1000);
}

for(i=MONOVOL;i<0x1F;i++)
{
WriteChip(0x32,i); //Step-up MONOOUT Volume
//Delay(1000);
}
    
```

6. Using BYPASS

"BYPASS" means that the input of analog signals is passed to analog output devices such as earphone without any transformation. In this case, both ADC and DAC are disabled.

When using BYPASS, the registers are set to the followed values as in table accordingly. Hereafter, the I2C/SPI functions in Sect. 2 are used to READ/WRITE registers in PA5750.

REGISTER	WRITE VALUE TO REGISTER(HEX)	REMARK
R2	FFH	STOP ADC & DAC STATE MACHINE AND DLL
R8	00Hor 80H	SET CHIP TO SLAVE OR MASTER MODE, user defined
R0	05H	Enrefr=1,PA5750in Playback and Record Mode. Set PA5750 to the correct mode
R1	40H	Pdn_ana=0,ibiasgen_pdn=0,Vrefr_LO=0,pdn_Vrefbuf=0, start the analog bias
R3	FCH	Pdn_MICB=1,ADC POWER DOWN, if using MICROPHONE IN, set pdn_MICB to 0, and pdn_AINL,pdn_AINR to 0.
R4	FFH	Pdn_DACL=1,Pdn_DACR=1,LOUT1=1,ROUT1=1,LOUT2=1,ROUT2=1,OUT3=1,MONO=1, disable DAC, turn on analog input
R38	00H	Set input of LEFT MIXER and RIGHT MIXE, if using MICROPHONE, R38=0x1B
R39	50H	LDAC TO LMIXER
R40	38H	
R41	38H	RDAC TO RMIXER
R42	50H	
R43	50H	UNUSE MONO MIXER
R44	38H	
R45	90H	VROI=1,OUT3=ROUT1
R46	1EH	LOUT1 VOLUME=0DB
R47	1EH	ROUT1 VOLUME=0DB

R48	1EH	LOUT2 VOLUME=0DB
R49	1EH	ROUT2 VOLUME=0DB
R50	1EH	MONO VOLUME=0DB

A sample of the driving program:

```

WriteChip(0x02,0xFF); //R2=0XF3, a must as mentioned before
if(SPDIF.MS_STA==MASTER)
{
WriteChip(0x08,0x00); //PA5750 to SLAVE mode
}
else
{
WriteChip(0x08,0x80); //PA5750 to MASTER mode
}
WriteChip(0x00,0x05); //set PA5750 to PLAYBACK & RECORD mode, EnRefr=1
WriteChip(0x01,0x40); //pdn_ana=0,ibiasgen_pdn=0
WriteChip(0x03,0xFC); //ADC Power-DOWN,DON'T USE MICROPHONE INPUT
//WriteChip(0x03,0x30); //ADC Power-DOWN, use MICROPHONE INPUT
WriteChip(0x04,0xFF); //DAC Power-DOWN,ANALOG OUT POWER-UP
//-----
//MIXER register
//-----
WriteChip(0x26,0x00); //LMIXSEL=LIN1,RMIXSEL=RIN1
//WriteChip(0x26,0x1B); // using MICROPHONE as input

WriteChip(0x27,0x50); //LMIXSEL TO Left MIXER,LMIXSEL VOLUME=0DB
WriteChip(0x28,0x38);
WriteChip(0x29,0x38);
WriteChip(0x2A,0x50); //RMIXSEL TO RIGHT MIXER,RMIXSEL VOLUME=0DB
WriteChip(0x2B,0x50); //LMIXSEL TO Mono-MIXER
WriteChip(0x2C,0x38);

WriteChip(0x2D,0x90); //ROUTINV=1,OUT3=VREF,VROI=1,DISABLE HEADPHONE //SWITCH
//-----
//SET VOLUME OF LOUT1/ROUT1/LOUT2/ROUT2/OUT3/MONOOOUT
//-----
ReadChip(0x2E); // read LO1,RO1,LO2,RO2,MONOOOUTVOLUME register
LO1VOL=RDDATA;
ReadChip(0x2F);
RO1VOL=RDDATA;

ReadChip(0x30);
LO2VOL=RDDATA;
ReadChip(0x31);
RO2VOL=RDDATA;

ReadChip(0x32);
MONOVOL=RDDATA;

if(LO1VOL<RO1VOL)
{
i=LO1VOL;
}
else
{
i=RO1VOL;
}
for(;i<0x1F;i++)
{
WriteChip(0x2E,i); //Step-up LO1 Volume
WriteChip(0x2F,i); //Step-up RO1 Volume
}

```

```
//Delay(1000);
}

if(LO2VOL<RO2VOL)
{
i=LO2VOL;
}
else
{
i=RO2VOL;
}
for(;i<0x1F;i++)
{
WriteChip(0x30,i); //Step-up LO2 Volume
WriteChip(0x31,i); //Step-up RO2 Volume
//Delay(1000);
}

for(i=MONOVOL;i<0x1F;i++)
{
WriteChip(0x32,i); //Step-up MONOOUT Volume
//Delay(1000);
}
```