第 7 章 AD 转换和 DA 转换的 C 语言程序设计 68

7.1 ADC 和 DAC 的硬件特性 68

7.1.1 ADC 输入接口的结构 68

7.1.2 ADC 的直流电气特性 69

7.1.3 DAC 音频输出的结构 70

7.2 ADC 和 DAC 设置的寄存器 70

7.2.1 MIC 输入的存储单元 70

7.2.2 ADC 的控制端口 70

7.2.3 ADC 多通道控制单元 71

7.2.4 模拟电压输入的存储单元 72

7.2.5 DAC 数据存放的单元 72

7.2.6 DAC 音频输出方式的控制单元 72

7.3 ADC 和 DAC 设置的 C 函数 73

7.4 ADC 和 DAC 的应用实例 77

7.4.1 AD 转换的牛刀小试 77

7.4.2 锯齿波的产生 78

7.4.3 声音的录入和播放 79

第7章 AD 转换和 DA 转换的 C 语言程序设计

7.1 ADC 和 DAC 的硬件特性

SPCE061A 内置 8 通道 10 位 ADC 和 2 通道 10 位 DAC。

7.1.1 ADC 输入接口的结构

SPCE061A 内置 8 通道 10 位模-数转换器，其中 7 个通道用于将模拟量信号 (例如电压信号)转换为数字量信号, 可以直接通过引线(IOA[0~6])输入。另外一个通道 IOA7 只用于语音输入，即通过内置自动增益控制的麦克风通道(MIC_IN)输入。实际上可以把模数转换器(ADC，Analog to Digital Converter)看作是一个实现模/数信号转换的编码器。在 ADC 内，由数模转换器 DAC0 和逐次逼近寄存器 SAR 组成逐次逼近式模-数转换器。向 P_ADC_Ctrl(写)($7015H)单元第 0 位(ADE)写入“1”用以激活 ADC。系统默认设置为 ADE=0，即屏蔽 ADC。

ADC 采用自动方式工作。硬件 ADC 的最高速率限定为(Fosc/32/12)Hz，如果速率超过此值，当从 P_ADC(读)($7014H)单元读出数据时会发生错误。

表 7.1 列出了 ADC 在各种系统时钟频率下的响应速率。

表7.1 ADC 在各种系统时钟频率下的响应速率。

FOSC(MHz)	20.48	24.576	32.768	40.96	49.152
ADC 响应率(KHz)	640	768	1024	1280	1536

在 ADC 自动方式被启用后，会产生出一个启动信号，即 RDY=0。此时，DAC0 的电压模拟量输出值与外部的电压模拟量输入值进行比较，以尽快找出外部电压模拟量的数字量输出值。逐次逼近式控制首先将 SAR 中数据的最高有效位试设为‘1’，而其它位则全设为‘0’，即 10 0000 0000B。这时，DAC0 输出电压 V_{DAC0}(1/2 满量程)就会与输入电压 V_{in} 进行比较。如果 Vin>V_{DAC0}，则保持原先设置为‘1’的位(最高有效位)仍为‘1’；否则，该位会被清‘0’。接着，逐次逼近式控制又将下一位试设为‘1’，其余低位依旧设为‘0’，即 110000 0000B，V_{DAC0} 与 V_{in} 进行比较的结果若 Vin>V_{DAC0}，则仍保持原先设置位的值，否则便清‘0’该位。这个逐次逼近的过程一直会延续到 10 位中的所有位都被测试之后，A/D 转换的结果保存在 SAR 内。

当 10 位 A/D 转换完成时，RDY 会被置‘1’。此时，用户通过读取 P_ADC (7014H)或 P_ADC_MUX_Data(702CH)单元可以获得 10 位 A/D 转换的数据。而从该单元读取数据后，又会使 RDY 自动清‘0’来重新开始进行 A/D 转换。若未读取 P_ADC (7014H) 或 P_ADC_MUX_Data(702CH)单元中的数据，RDY 仍保持为‘1’，则不会启动下一次的 A/D 转换。外部信号由 LIN_IN[1~7]即 IOA[0~6]或通道 MIC_IN 输入。从 LIN_IN[1~7]输入的模拟信号直接被送入缓冲器 P_ADC_MUX_Data(702CH)；从 MIC_IN 输入的模拟信号则要经过缓冲器和放大器。AGC 功能将通过 MIC_IN 通道输入的模拟信号的放大值控制在一定范围内，然后放大信号经采样-保持模块被送至比较器参与 A/D 转换值的确定，最后送入 P_ADC (7014H)。

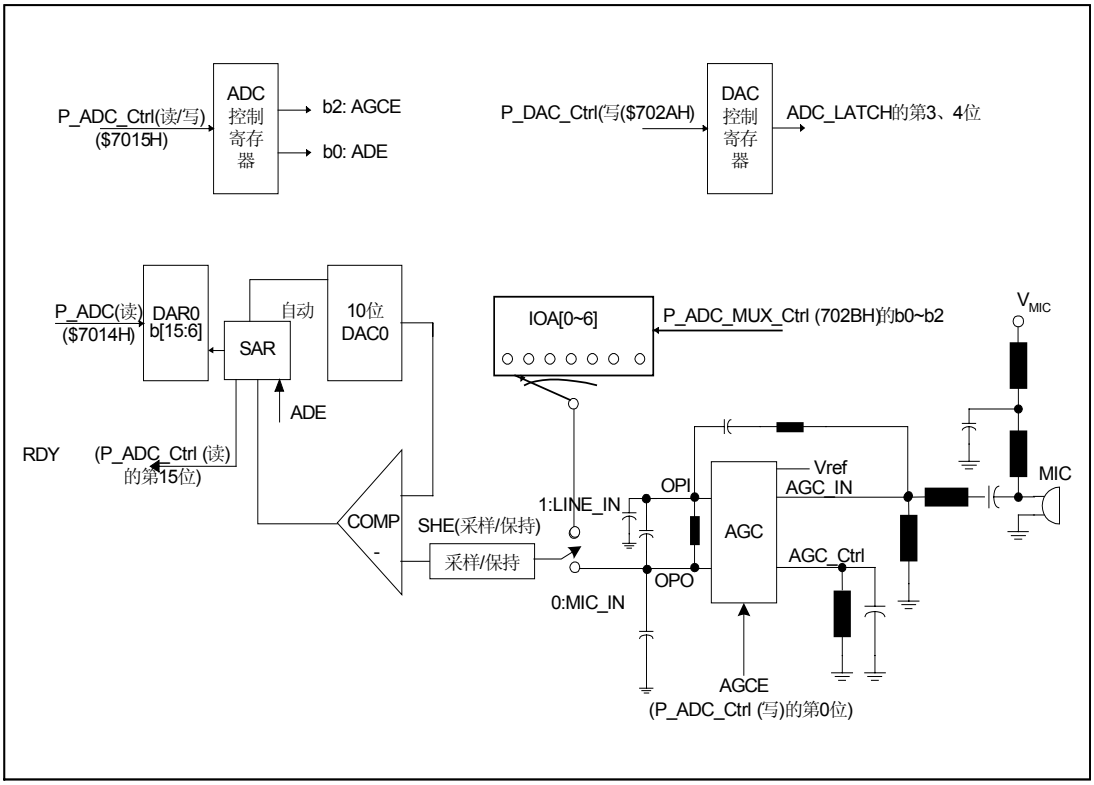

图7.1　ADC 输入接口的结构

7.1.2 ADC 的直流电气特性

表7.2　ADC 直流电气特性

ADC 直流电气项目	项目符号	最小值	典型值	最大值	单位
ADC 分辨率	RESO		10		bit
ADC 有效位数	ENOB	8			bit
ADC 信噪比	SNR	50			dB
ADC 积分非线性	INL		±4		LSB[1]
ADC 差分非线性	DNL		±0.5		LSB
ADC 转换率	F_{CONV}			96K[2]	Hz
电源电流 @Vdd=3V	I_{ADC}		3.4		mA
功耗 @Vdd=3V	P_{ADC}		10.2		mW

注：

[1] LSB 表示为最小有效单位，在 VRT=3V 的情况下，1LSB 为 2.93 mv。

[2] 此由最大采样率(Samplerate_max)得来，即 Samplerate_max =ADC 响应率/16=1536KHz/16=96KHz。

7.1.3 DAC 音频输出的结构

SPCE061A 提供的音频输出方式为双通道 DAC 方式。在此方式下，DAC1、DAC2 转换输出的模拟量电流信号分别通过 AUD1 和 AUD2 管脚输出，输出的数字量分别写入 P_DAC1(写)($7017)和 P_DAC2(写) ($7016)单元。音频输出的结构如图 7.2 所示。

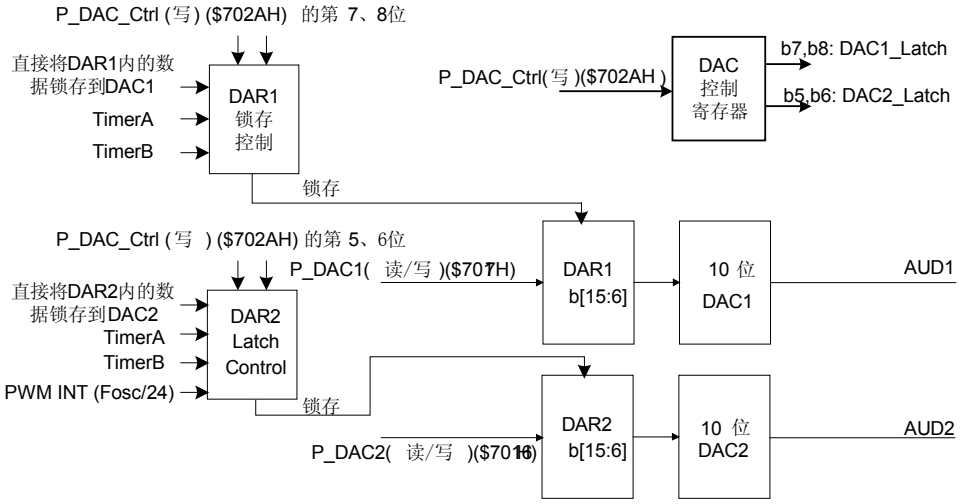

图7.2 音频输出的结构

7.2 ADC 和 DAC 设置的寄存器

7.2.1 MIC 输入的存储单元

P_ADC(读/写)($7014H)

P_ADC 单元（如表 7.3 所示）储存 MIC 输入的 A/D 转换的数据。逐次逼近式的 ADC 由一个 10 位 DAC(DAC0) 、一个 10 位缓存器 DAR0、一个逐次逼近寄存器 SAR 和一个比较器 COMP 组成。

表7.3 P_ADC 单元

b15 – b6	b5 – b0
DAR0(读/写)	---

P_ADC(读)：读出本单元实际为 A/D 转换输出的 10 位数字量。而且，如果 P_DAC_Ctrl (702AH)单元第 3、4 位被设为‘00’，那么在转换过程里读出本单元(7014H)亦会触发 A/D 转换重新开始。

7.2.2 ADC 的控制端口

P_ADC_Ctrl(读/写)($7015H)

P_ADC_Ctrl 单元（如表 7.4 所示）为 ADC 的控制端口。

表7.4 P_ADC_Ctrl 单元

b15[2]	b6	B2	b0	控制功能描述
RDY (读)	DAC_I (写)	AGCE (写)	ADE (写)	
0	-	-	-	10 位模/数转换未完成
1	-	-	-	10 位模/数转换完成，输出 10 位数字量
	0			DC 电流= 3mA @V_{DD}=3V[1]
	1			DAC 电流= 2mA @V_{DD}=3V
-	-	0	-	取消自动增益控制功能[3]
-	-	1	-	设置自动增益控制功能
-	-	-	0	禁止模/数转换工作
-	-	-	1	允许模/数转换工作

注：

[1] 此为 DAC_I 的缺省选择。

[2] b15 只用于 MIC_IN 通道输入。

[3] 当模拟信号通过麦克风的 MIC_IN 通道输入时，可选择 AGCE 为‘1’，即运算放大器的增益可在其线性区域内自动调整。AGCE 缺省选择为‘0’，即取消自动增益控制功能。

[4] 写入时需注意 b5 = 1, b4=1, b3 =1 和 b1=0。

7.2.3 ADC 多通道控制单元

P_ADC_MUX_Ctrl (读/写)(702BH)

ADC 多通道控制是通过对 P_ADC_MUX_Ctrl (702BH)单元（如表 7.5 所示）编程实现的。

表7.5 P_ADC_MUX_Ctrl 单元

b15	b14	b13-b3	b2	b1	b0	控制功能描述
Ready_MUX (读)[1]	FAIL (读) [2]	-	Channel_sel (读/写)			
0		-	-	-	-	10 位模/数转换未完成
1	0	-	-	-	-	10 位模/数转换完成
-		-	0	0	0	模拟电压信号通过 MIC_IN 输入
-		-	0	0	1	模拟电压信号通过 LINE_IN1 输入
-		-	0	1	0	模拟电压信号通过 LINE_IN2 输入
-		-	0	1	1	模拟电压信号通过 LINE_IN3 输入
-		-	1	0	0	模拟电压信号通过 LINE_IN4 输入
-		-	1	0	1	模拟电压信号通过 LINE_IN5 输入
-		-	1	1	0	模拟电压信号通过 LINE_IN6 输入
-		-	1	1	1	模拟电压信号通过 LINE_IN7 输入

注 :

[1]. Ready_MUX 只用于 Line_in[7:1].

[2]. 一般情况下，该位总为‘0’。以下情况除外：由于 MIC_IN 的优先级高于 AD LINE_IN，所以在 LIN_IN AD 转换过程里又有 MIC_IN 时，若 AD 切换到 MIC 输入，原 LINE_IN 的数据会出现问题，此时 FAIL 被置为‘1’。 MIC AD 完成之后，该位被清为‘0’。

ADC 的多路 LINE_IN 输入将与 IOA[0~6]共用，即：

表7.6 多路 LINE_IN 输入与 IOA[0~6]共用

IOA6	IOA5	IOA4	IOA3	IOA2	IOA1	IOA0
LIN_IN 7	LIN_IN 6	LIN_IN 5	LIN_IN 4	LIN_IN 3	LIN_IN 2	LIN_IN 1

7.2.4 模拟电压输入的存储单元

P_ADC_MUX_Data(读) ($702CH)

P_ADC_MUX_Data 单元用于读出 LINE_IN[7:1]10 位 ADC 转换的数字数据，即：

表7.7 10 位 ADC 转换的数字数据存放格式

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6
D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

7.2.5 DAC 数据存放的单元

P_DAC2(读/写)($7016H)

在 DAC 方式下，该单元是一个带 10 位缓冲寄存器 DAR2 的 10 位 D/A 转换单元(DAC2)。

表7.8 P_DAC2 单元

b15 – b6	b5 – b0
DA2_Data(读/写)	---

P_DAC2(写)：通过此单元直接写入 10 位数据到 10 位缓存器 DAR2，来锁存 DAC2 的输入数字量值(无符号数)。

P_DAC2(读)：从 DAR2 内读出 10 位数据。

P_DAC1(读/写)($7017H)

该单元为一个带 10 位缓存器(DAR1)的 10 位 D/A 转换单元(DAC1)。用于向 DAR1 写入或从其中读出 10 位数据。

表7.9 P_DAC1 单元

b15 – b6	b5 – b0
DA1_Data(读/写)	---

7.2.6 DAC 音频输出方式的控制单元

P_DAC_Ctrl(写) ($702AH)

DAC 音频输出方式的控制单元（如表 7.10 所示），其中第 5~8 位用于选择 DAC 输出方式下的数据锁存方式；第 3、4 位用来控制 A/D 转换方式；第 1 位总为‘0’，用于双 DAC 音频输出。b9~b15 为保留位。表 7.10 详细地列出了 P_DAC_Ctrl 单元的 b3~b8 的控制功能。

表7.10 P_DAC_Ctrl 单元

b8	b7	b6	b5	b4	b3
DAC1_Latch(写)		DAC2_Latch(写)		AD_Latch(写)	

00：直接将 DAR1 内数据锁存到 DAC1 内(缺省设置) 01：通过 TimerA 溢出 DAR1 内的数据锁存到 DAC1 内 10：通过 TimerB 溢出将 DAR1 内的数据锁存到 DAC1 内 11：通过 TimerA 或 TimerB 的溢出将 DAR1 内的数据锁存到 DAC1 内	00：直接将 DAR2 内的数据锁存到 DAC2 内(缺省设置) 01：通过 TimerA 溢出将 DAR2 内的数据锁存到 DAC2 内 10：通过 TimerB 溢出将 DAR2 内的数据锁存到 DAC2 内 11：通过 TimerA 或 TimerB 的溢出将 DAR2 内的数据锁存到 DAC2 内	00：通过读 ADC(读)($7014H)触发 ADC 自动转换(缺省设置) 01：通过 TimerA 溢出触发 A/D 转换 10：通过 TimerB 溢出触发 A/D 转换 11：通过 TimerA 或 TimerB 的溢出触发 A/D 转换

7.3 ADC 和 DAC 设置的 C 函数

函数原型

void　Set_ADC_Data(unsigned int);

功能说明　Set ADC Data，for Microphone input data

用法

Set_ADC_Data(ADC_Data);

参数

ADC_Data = b15 .. b6

函数原型

unsigned int Get_ADC_Data(void);

功能说明　Get ADC Data，for Microphone input data

用法

ADC Data = Get_ADC_Data();

返回值

ADC_Data = b15 .. b6

函数原型

unsigned int Get_ADC_Ctrl(void);

unsigned int Get_ADC_MUX_Ctrl(void);

功能说明　Get ADC Conversion Ready

用法

ADC Ready1 = Get_ADC_Ctrl ();

ADC Ready2 = Get_ADC_MUX_Ctrl();

返回值

ADC Ready1 =　b15

1: Ready

0: Converting

ADC Ready2 = b15,b14

b15 →

1: Ready

0: Converting

b14 →

1: Conversion completed successfully

0: Conversion failed

函数原型

void Set_ADC_Ctrl(unsigned int);

功能说明　Set ADC Control Register

用法

Set_ADC_Ctrl(ADC_Ctrl_Data);

参数

ADC_Ctrl_Data →

C_ADCE	→	Enable A/D converter
C_ADCN	→	Disable A/D converter
C_MIC_ENB	→	Enable Microphone
C_MIC_DIS	→	Disable Microphone
C_AGCE	→	Enable AGC Function
C_AGCN	→	Disable AGC Function
C_DAC_OUT2mA	→	DAC Output Current 2mA
C_DAC_OUT3mA	→	DAC Output Current 2mA
C_VEXTREF_ENB	→	VREF From Ext REF Voltage
C_VEXTREF_DIS	→	VREF From Internal REF Voltage
C_V2VREFB_DIS	→	Nor Generate 2V Voltage
C_V2VREFB_ENB	→	Generate 2V Voltage

Note:　ADC_Ctrl_Data is a write only data, so any change of data must reload each component.

ADC Enable →

Set_ADC_Data(C_ADCE);

Microphone Enable →

Set_ADC_Data(C_ADCE+ C_MIC_ENB);

Microphone Enable With AGC →

Set_ADC_Data(C_ADCE+ C_MIC_ENB+ C_AGCE);

函数原型

void　Set_ADC_MUX_Ctrl(unsigned int);

功能说明　Set ADC Line In Channel

用法

Set_ADC_MUX_Ctrl(Channel Number);

参数

Channel Number →

C_ADC_CH_MICin → MIC_IN
C_ADC_CH1 → Line In1 (IOA0)
C_ADC_CH2 → Line In2 (IOA1)
C_ADC_CH3 → Line In3 (IOA2)
C_ADC_CH4 → Line In4 (IOA3)
C_ADC_CH5 → Line In5 (IOA4)
C_ADC_CH6 → Line In6 (IOA5)
C_ADC_CH7 → Line In7 (IOA6)

函数原型

unsigned int Get_ADC_LineIn_Data(void);

功能说明　Get ADC Line In Data

用法

ADC Line In Data = Get_ADC_LineIn_Data();

返回值

ADC Line In Data = b15 .. b6

函数原型

void _Mic_Input_Ctrl(unsigned int,unsigned int);

功能说明　Set Microphone Input Control

用法

Mic_Input_Ctrl(Trigger Type, AGC On/Off);

参数

Trigger Type = DAC trigger type + Mic AD trigger type

If user use microphone and DAC output, please select DAC trigger type.

Ex:

Trigger Type = C_DA1_LatchA+C_DA2_LatchB+C_AD_LatchA

Trigger Type = C_DA2_LatchAB+C_AD_LatchA

DAC Trigger Type → DAC1 Latch + DAC2 Latch

DAC1_Latch →

C_DA1_Direct → Direct apply DAR1 data (Default)
C_DA1_LatchA → Latch DAR1 by Timer A
C_DA1_LatchB → Latch DAR1 by Timer B
C_DA1_LatchAB → Latch DAR1 by Timer A or Timer B

DAC2_Latch →

C_DA2_Direct → Direct apply DAR2 data (Default)
C_DA2_LatchA → Latch DAR2 by Timer A
C_DA2_LatchB → Latch DAR2 by Timer B
C_DA2_LatchAB → Latch DAR2 by Timer A or Timer B

AD_ Latch →

C_AD_Direct → Read P_ADC(R)($70114H) (Default)
C_AD_LatchA → Conversion by Timer A
C_AD_LatchB → Conversion by Timer B
C_AD_LatchAB → Conversion by Timer A or Timer B

AGC On/Off → Select AGC enable/disable

C_AGCE → AGC enable

C_AGCN → AGC disable

函数原型

void _Line_Input_Ctrl(unsigned int,unsigned int);

功能说明　Set Line Input Control

用法

Line_Input_Ctrl(Line In Channel , Vref On/Off);

参数

Line in channel →

C_ADC_CH1 → Line In1 (IOA0)

C_ADC_CH2 → Line In2 (IOA1)

C_ADC_CH3 → Line In3 (IOA2)

C_ADC_CH4 → Line In4 (IOA3)

C_ADC_CH5 → Line In5 (IOA4)

C_ADC_CH6 → Line In6 (IOA5)

C_ADC_CH7 → Line In7 (IOA6)

Vref On/Off = Vextref enable/disable + V2 Ref enable/disable

C_VEXTREF_ENB → Ext REF Voltage Enable

C_VEXTREF_DIS → Ext REF Voltage Disable (AVDD)(default)

C_V2VREFB_ENB → 2V Voltage output enable

C_V2VREFB_DIS → 2V Voltage output disable

函数原型

void Set_DAC1_Data(unsigned int);

void Set_DAC2_Data(unsigned int);

功能说明　Set DAC Data

用法

Set_DAC1_Data(DAC1 Data);

Set_DAC2_Data(DAC2 Data);

参数

DAC1 Data = b15 .. b6

DAC2 Data = b15 .. b6

函数原型

unsigned int Get_DAC1_Data(void);

unsigned int Get_DAC2_Data(void);

功能说明　Get DAC Data

用法

DAC1 Data = Get_DAC1_Data();

DAC2 Data = Get_DAC2_Data();

返回值

DAC1 Data = b15 .. b6
DAC2 Data = b15 .. b6

函数原型

void Set_DAC_Ctrl(unsigned int);

功能说明　Set DAC Control Register

用法

Set_DAC_Ctrl(DAC Ctrl Data);

参数

DAC Ctrl Data = DAC1_Latch + DAC2_Latch + AD_Latch

DAC1_Latch1 →

C_DA1_Direct	→ Direct apply DAR1 data (Default)
C_DA1_LatchA	→ Latch DAR1 by Timer A
C_DA1_LatchB	→ Latch DAR1 by Timer B
C_DA1_LatchAB	→ Latch DAR1 by Timer A or Timer B

DAC1_Latch2 →

C_DA2_Direct	→ Direct apply DAR2 data (Default)
C_DA2_LatchA	→ Latch DAR2 by Timer A
C_DA2_LatchB	→ Latch DAR2 by Timer B
C_DA2_LatchAB	→ Latch DAR2 by Timer A or Timer B

AD_ Latch →

C_AD_Direct	→ Read P_ADC(R)($70114H) (Default)
C_AD_LatchA	→ Conversion by Timer A
C_AD_LatchB	→ Conversion by Timer B
C_AD_LatchAB	→ Conversion by Timer A or Timer B

Note: A/D Latch → MIC_IN conversion

7.4 ADC 和 DAC 的应用实例

7.4.1 AD 转换的牛刀小试

例 7.1

通过模拟量输入口 LINE_IN 输入电压值，通过读取 P_ADC_MUX_Data 单元可以获得 10 位 A/D 转换的数据。而从该单元读取数据后，又会使 RDY 自动清'0'来重新开始进行 A/D 转换。若未读取 P_ADC_MUX_Data 单元中的数据 RDY 仍保持为'1'，则不会启动下一次的 A/D 转换。

本例中，电压信号从 IOA0 输入。参考电压为 3.3V。每调用一次子函数 AD()，将得到一个 AD 转换的值。例程中调用 AD()共 16 次，取平均值作为 AD 转换的结果。fVoltage 保存的就是换算后的电压值。

程序代码如下所示：

```
#include "SPCE061V004.H"
unsigned int AD(void);
```

```
main()
{
    unsigned long int uiData;
    unsigned int    i;
    float fVoltage;
    fVoltage=0.0;
    *P_ADC_MUX_Ctrl=C_ADC_CH1;              //LINE_IN-------IOA0
    *P_ADC_Ctrl= C_ADCE;                    //ADC enable
    uiData=*P_ADC_LINEIN_Data;              //ADC start
    uiData=0;
    for(i=0;i<16;i++)
    {
        *P_Watchdog_Clear=C_WDTCLR;
        uiData += AD();
    }
    uiData >>= 4;
    fVoltage=(float)uiData/0xffc0*3.3;
}

unsigned int AD(void)
{
    unsigned int uiData;
    while(!(*P_ADC_MUX_Ctrl&0x8000));       //wait,until ADC complete
    uiData=*P_ADC_LINEIN_Data;
    return(uiData&0xffc0);
}
```

7.4.2 锯齿波的产生

例 7.2

本实验采取直接方式，通过编程实现一个锯齿波，将 DAC 输出端接示波器可以观察到锯齿波形,同时也可以听到 AUD1 和 AUD2 两端的扬声器有持续间断的声音。

程序代码如下：

```
#include "SPCE061V004.H"
main()
{
    unsigned int uiData,i;
    __asm("int off");                       //disable interrupt
    *P_DAC_Ctrl=C_DA1_Direct;               //Latch to DAC1 directly
    uiData=0x0;
    while(1)
    {
        *P_Watchdog_Clear=C_WDTCLR;
```

```
            *P_DAC1=uiData;                    //output
            i=32;
            while(i--);
            uiData+=0x0040;
        }
    }
```

7.4.3 声音的录入和播放

例 7.3

本例采用自动方式即定时器 A 溢出执行 ADC 转换，通过 A/D 将 MIC_IN 输入的语音信号转换为数字信号，再通过 D/A 的两个通道 AUD1 和 AUD2 播放。

本例是即录即放，没有进行滤波处理，效果不是很好。要达到比较好的效果，可以使用凌阳提供 DVR 函数，详见第九章。

本例的程序代码不多，主程序如下：

```
#include  "SPCE061V004.H"
#define     TIMER_DATA_FOR_8KHZ (0xffff - 1500)     // 时钟 Fosc/2,采样率 8kHz
main()
{
    __asm("int off");
    *P_TimerA_Ctrl = 0x0030;                        // 时钟频率为 CLKA 的 Fosc/2
    *P_TimerA_Data = TIMER_DATA_FOR_8KHZ;           // 数据采样率为 8kHz

    *P_ADC_Ctrl = 0x0015;                           // 设置 AGC 、自动方式
                                                    //  MIC_IN 通道输入

    *P_DAC_Ctrl = 0x00A8;                           // 通过 TimerA 溢出触发
                                                    //  ADC 自动方式转换

    *P_INT_Ctrl = 0x1000;                           // 设置 IRQ1_TM 中断
    __asm("int irq");

    while(1)
        *P_Watchdog_Clear=C_WDTCLR;
}
```

中断服务程序如下：

```
#include  "SPCE061V004.H"
void IRQ1(void) __attribute__ ((ISR));
void IRQ1(void)
{
    unsigned int uiData;
    uiData=*P_ADC;                  //读取 A/D 转换的数值
    *P_DAC1 = uiData;               //DAC 输出
```

```
    *P_DAC2 = uiData;
    *P_INT_Clear = C_IRQ1_TMA;              //清中断标识位
}
```