

Cortex-M4原理与实践实验报告

题目： 并行ADC与DAC

**作者姓名 章金皓**

**指导教师**  **付明磊**

**专业班级 自动化1901**

**学 院 信息工程学院**

**提交日期** 2021年12月5日

**一、实验目的**

1. 熟悉并口 AD/DA 芯片的结构及工作方式

2. 熟悉并行口的扩展编程

**二、实验设备**

1.计算机一台，操作系统为WindosXP或Windos7，装有CCSv6.0软件。

2.EK-TM4C1294XL实验开发板一块。

3.USB连接线一条。

**三、实验原理和流程**

**1、Dac7821芯片原理**

（1）Dac7821 芯片简介

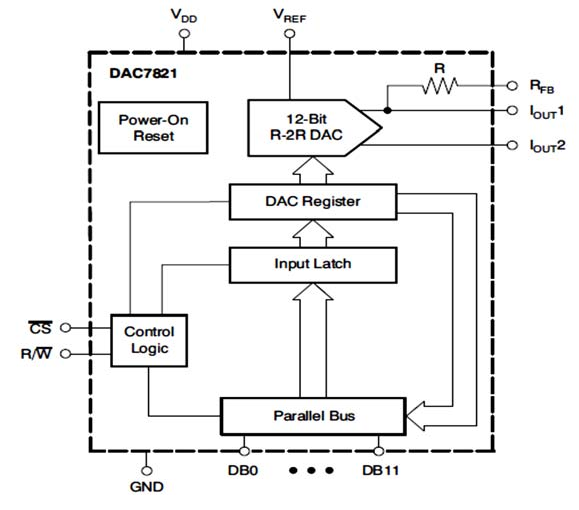
Dac7821 是一个 12 位的电流输出的数模转换器，器件的工作电压为 2.5V 到 5.5V。芯片 是通过快速的并口接口工作。如图 1为芯片的功能框图。

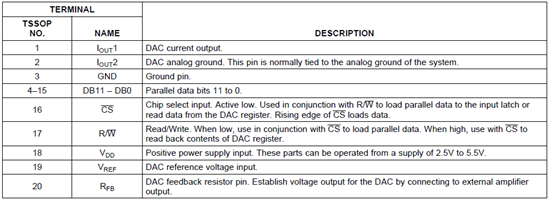
图 1 Dac7821功能框图

图示

描述已自动生成 （2）Dac7821时序图

图 2 Dac7821时序图

（3）Dac7821时序要求

图 3 DA时序要求

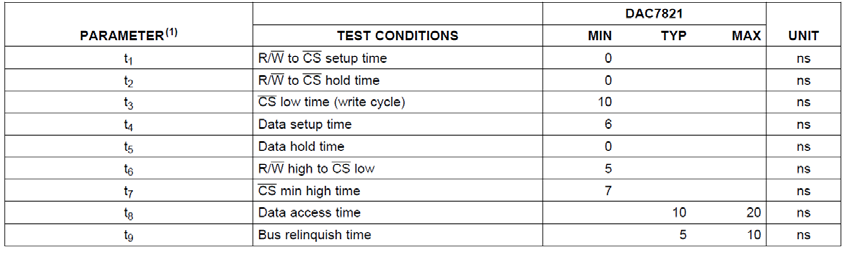
 （4）Dac7821时序要求

图 4 Dac7821管脚定义

**2、ADS7881芯片原理**

（1）ADS7881芯片简介

图示

描述已自动生成 ADS7881是12位4-MSPS AD转换器，提供2.5V的内部参考电压。该芯片提供了一个12位的并行接口与额外的与8位模式的接口。如图5为改芯片的功能框图。

图 5 ADS7881功能框图

（2）ADS7881管脚定义

图示

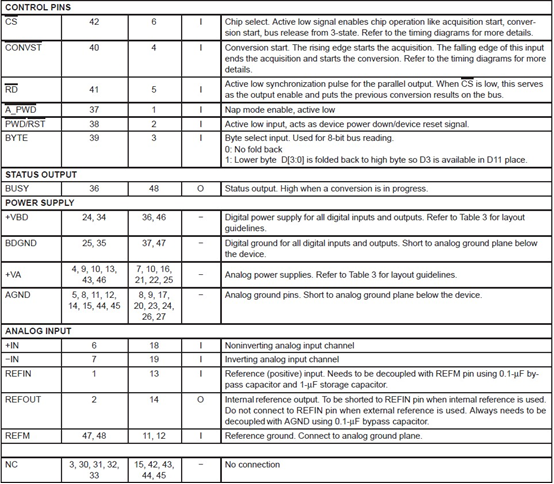
描述已自动生成 （3）ADS7881时序图

图 6ADS7881管脚定义

图 7 ADS7881时序图

图形用户界面, 表格

中度可信度描述已自动生成 （4）ADS7881时序要求

图 8 AD时序要求

**流程图如下：**



图 9 DA实验流程图



图 10 AD实验流程图

**四、实验代码、注释**

**1、DA实验**

**实验代码：**

#include <stdbool.h>

#include <stdint.h>

#include "inc/hw\_gpio.h"

#include "inc/hw\_memmap.h"

#include "inc/hw\_types.h"

#include "driverlib/gpio.h"

#include "driverlib/epi.h"

#include "driverlib/rom\_map.h"

#include "driverlib/sysctl.h"

#include "driverlib/timer.h"

#include "driverlib/rom.h"

#include "inc/tm4c1294ncpdt.h"

#define DAC\_DATAH\_PERIPH\_G SYSCTL\_PERIPH\_GPIOG

#define DAC\_DATAH\_BASE\_G GPIO\_PORTG\_BASE

#define DAC\_DATAH\_D11 0x40026004

#define DAC\_DATAH\_D11\_S(a) ((a) >> 11)

#define DAC\_DATAH\_D10 0x40026008

#define DAC\_DATAH\_D10\_S(a) ((a) >> 9)

#define DAC\_DATAH\_PERIPH\_A SYSCTL\_PERIPH\_GPIOA

#define DAC\_DATAH\_BASE\_A GPIO\_PORTA\_BASE

#define DAC\_DATAH\_D9 0x40004200

#define DAC\_DATAH\_D9\_S(a) ((a) >> 2)

#define DAC\_DATAH\_D8 0x40004100

#define DAC\_DATAH\_D8\_S(a) ((a) >> 2)

#define DAC\_DATAH\_PERIPH\_C SYSCTL\_PERIPH\_GPIOC

#define DAC\_DATAH\_BASE\_C GPIO\_PORTC\_BASE

#define DAC\_DATAH\_D7 0x40006040

#define DAC\_DATAH\_D7\_S(a) ((a) >> 3)

#define DAC\_DATAH\_D6 0x40006080

#define DAC\_DATAH\_D6\_S(a) ((a) >> 1) //

#define DAC\_DATAH\_D5 0x40006100 //0000 0100 //

#define DAC\_DATAH\_D5\_S(a) ((a) << 1) //0001 0000

#define DAC\_DATAH\_D4 0x40006200 //0000 0100 //

#define DAC\_DATAH\_D4\_S(a) ((a) << 3) //0001 0000

#define DAC\_DATAH\_PERIPH\_K SYSCTL\_PERIPH\_GPIOK

#define DAC\_DATAH\_BASE\_K GPIO\_PORTK\_BASE

#define DAC\_DATAH\_D3\_S(a) ((a) << 3) //0001 1011 0000 1000

#define DAC\_DATAH\_D3 0x40061020 //1000 0000

#define DAC\_DATAH\_D2\_S(a) ((a) << 1)

#define DAC\_DATAH\_D2 0x40061010

#define DAC\_DATAH\_D1\_S(a) ((a) >> 1)

#define DAC\_DATAH\_D1 0x40061008

#define DAC\_DATAH\_D0\_S(a) ((a) >> 3)

#define DAC\_DATAH\_D0 0x40061004

#define DAC\_DATAH\_PIN\_11 GPIO\_PIN\_0

#define DAC\_DATAH\_PIN\_10 GPIO\_PIN\_1

#define DAC\_DATAH\_PIN\_9 GPIO\_PIN\_7

#define DAC\_DATAH\_PIN\_8 GPIO\_PIN\_6

#define DAC\_DATAH\_PIN\_7 GPIO\_PIN\_4

#define DAC\_DATAH\_PIN\_6 GPIO\_PIN\_5

#define DAC\_DATAH\_PIN\_5 GPIO\_PIN\_6

#define DAC\_DATAH\_PIN\_4 GPIO\_PIN\_7

#define DAC\_DATAH\_PIN\_3 GPIO\_PIN\_3

#define DAC\_DATAH\_PIN\_2 GPIO\_PIN\_2

#define DAC\_DATAH\_PIN\_1 GPIO\_PIN\_1

#define DAC\_DATAH\_PIN\_0 GPIO\_PIN\_0

#define DAC\_WR\_CS\_RD\_PERIPH SYSCTL\_PERIPH\_GPIOE

#define DAC\_WR\_CS\_RD\_BASE GPIO\_PORTE\_BASE

#define DAC\_WR\_PIN GPIO\_PIN\_2

#define DAC\_WR\_PIN\_REG 0x40024010

#define DAC\_CS\_PIN GPIO\_PIN\_1

#define DAC\_CS\_PIN\_REG 0x40024008

#define SET\_DAC\_DATA(ui16Byte) \

{ \

HWREG(DAC\_DATAH\_D11) = DAC\_DATAH\_D11\_S(ui16Byte); \

HWREG(DAC\_DATAH\_D10) = DAC\_DATAH\_D10\_S(ui16Byte); \

HWREG(DAC\_DATAH\_D9) = DAC\_DATAH\_D9\_S(ui16Byte); \

HWREG(DAC\_DATAH\_D8) = DAC\_DATAH\_D8\_S(ui16Byte); \

HWREG(DAC\_DATAH\_D7) = DAC\_DATAH\_D7\_S(ui16Byte); \

HWREG(DAC\_DATAH\_D6) = DAC\_DATAH\_D6\_S(ui16Byte); \

HWREG(DAC\_DATAH\_D5) = DAC\_DATAH\_D5\_S(ui16Byte); \

HWREG(DAC\_DATAH\_D4) = DAC\_DATAH\_D4\_S(ui16Byte); \

HWREG(DAC\_DATAH\_D3) = DAC\_DATAH\_D3\_S(ui16Byte); \

HWREG(DAC\_DATAH\_D2) = DAC\_DATAH\_D2\_S(ui16Byte); \

HWREG(DAC\_DATAH\_D1) = DAC\_DATAH\_D1\_S(ui16Byte); \

HWREG(DAC\_DATAH\_D0) = DAC\_DATAH\_D0\_S(ui16Byte); \

}

uint32\_t g\_ui32SysClock;

void WriteDataGPIO(uint16\_t ui16Data)

{

// 数据写入

HWREG(DAC\_CS\_PIN\_REG)=0xff; //CS高电平

SysCtlDelay(3);

HWREG(DAC\_WR\_PIN\_REG) = 0; // WR低电平

SysCtlDelay(1);

HWREG(DAC\_CS\_PIN\_REG) = 0; // CS低电平

SET\_DAC\_DATA(ui16Data);

SysCtlDelay(5);

HWREG(DAC\_CS\_PIN\_REG) = 0xFF; // CS高电平

SysCtlDelay(1);

HWREG(DAC\_WR\_PIN\_REG) = 0xFF; //WR高电平

SysCtlDelay(6);

// Pull CS Low.

HWREG(DAC\_CS\_PIN\_REG) = 0; //CS低电平

SysCtlDelay(3);

HWREG(DAC\_CS\_PIN\_REG)=0xff; //CS高电平

}

void main()

{

volatile uint32\_t ui32Loop,ui32val,ui32Loop0,ui32Loop1,ui32Loop2,ui32Loop3,ui32Loop4;

SYSCTL\_RCGCGPIO\_R |= SYSCTL\_RCGCGPIO\_R5;

// 使能各模块

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

GPIOPinTypeGPIOOutput(GPIO\_PORTF\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_4);

SysCtlPeripheralEnable(DAC\_DATAH\_PERIPH\_A);

SysCtlPeripheralEnable(DAC\_DATAH\_PERIPH\_C);

SysCtlPeripheralEnable(DAC\_DATAH\_PERIPH\_G);

SysCtlPeripheralEnable(DAC\_DATAH\_PERIPH\_K);

SysCtlPeripheralEnable(DAC\_WR\_CS\_RD\_PERIPH);

SysCtlGPIOAHBEnable(SYSCTL\_PERIPH\_GPIOA);

SysCtlGPIOAHBEnable(SYSCTL\_PERIPH\_GPIOC);

SysCtlGPIOAHBEnable(SYSCTL\_PERIPH\_GPIOG);

SysCtlGPIOAHBEnable(SYSCTL\_PERIPH\_GPIOK);

SysCtlGPIOAHBEnable(SYSCTL\_PERIPH\_GPIOE);

// 配置功能输出

GPIOPinTypeGPIOOutput(DAC\_WR\_CS\_RD\_BASE, DAC\_CS\_PIN);

GPIOPinTypeGPIOOutput(DAC\_WR\_CS\_RD\_BASE, DAC\_WR\_PIN);

GPIOPinTypeGPIOOutput(DAC\_DATAH\_BASE\_C, DAC\_DATAH\_PIN\_7 |

DAC\_DATAH\_PIN\_6 | DAC\_DATAH\_PIN\_5 | DAC\_DATAH\_PIN\_4);

GPIOPinTypeGPIOOutput(DAC\_DATAH\_BASE\_K, DAC\_DATAH\_PIN\_3 |

DAC\_DATAH\_PIN\_2 | DAC\_DATAH\_PIN\_1 | DAC\_DATAH\_PIN\_0);

GPIOPinTypeGPIOOutput(DAC\_DATAH\_BASE\_A, DAC\_DATAH\_PIN\_9 | DAC\_DATAH\_PIN\_8);

GPIOPinTypeGPIOOutput(DAC\_DATAH\_BASE\_G, DAC\_DATAH\_PIN\_11 | DAC\_DATAH\_PIN\_10);

// 设置系统时钟

g\_ui32SysClock = SysCtlClockFreqSet((SYSCTL\_OSC\_INT | SYSCTL\_USE\_PLL |

SYSCTL\_CFG\_VCO\_320), 120000000);

ui32val = 2000;

while(1)

{

WriteDataGPIO(ui32val);

SysCtlDelay(100);

for(ui32Loop = 0; ui32Loop < 2000000; ui32Loop++)

{

}

ui32val += 100;

if(ui32val >= 4000)

ui32val = 0;

}

}

**实验现象：**

因没有示波器无法查看

**2、AD实验**

**实验代码：**

#include "stdint.h"

#include "stdbool.h"

#include "driverlib/debug.h"//调试

#include "inc/hw\_types.h"//数据类型设置，寄存器访问封装

#include "inc/hw\_memmap.h"//基址

#include "driverlib/fpu.h"

#include "driverlib/sysctl.h"//使能寄存器

#include "driverlib/rom.h"

#include "driverlib/pin\_map.h"

#include "grlib/grlib.h"

#include "driverlib/gpio.h"

#include "driverlib/adc.h"

#include "inc/hw\_gpio.h"

#include "inc/tm4c1294ncpdt.h"

#include "driverlib/uart.h"

#include "utils/uartstdio.h"

volatile uint32\_t ui32SysClock;

void InitConsole(void)//UART初始化

{

// 由于UART0使用PA0,PA1两个引脚，因此需要使能GPIOA模块

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

// 因为有引脚复用，所以要对PA0和PA1两个引脚的功能进行选择

// 这里将它们选择为执行UART0模块的功能

GPIOPinConfigure(GPIO\_PA0\_U0RX);

GPIOPinConfigure(GPIO\_PA1\_U0TX);

// 对于PA0和PA1两个引脚，在将它们作为UART功能使用之前，需要对它们做一些有关UART的配置

GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

UARTStdioConfig(0, 115200, ui32SysClock);

}

void GPIOint()//GPIO口初始化

{

GPIO\_PORTA\_AHB\_DIR\_R = 0x00; //PA7,PA6，对应并口线D8,D9

GPIO\_PORTC\_AHB\_DIR\_R = 0x00; //PC7-4，对应并口线D4-D7

GPIO\_PORTD\_AHB\_DIR\_R = 0x30;//PD4,PD5，对应AD\_convst,AD\_byte

GPIO\_PORTE\_AHB\_DIR\_R = 0x03; //PE0对应RD，PE1对应CS

GPIO\_PORTG\_AHB\_DIR\_R = 0x00; //PG1,PG0，对应并口线D10,D11

GPIO\_PORTK\_DIR\_R = 0x00; //PK3-0，对应并口线D0-D3

//启用数字I/O功能

GPIO\_PORTA\_AHB\_DEN\_R = 0xc0; //1100 0000 PA7,PA6置1

GPIO\_PORTC\_AHB\_DEN\_R = 0xf0;//1111 0000,PC7-4置1

GPIO\_PORTD\_AHB\_DEN\_R = 0x70;//0111 0000,PD4,PD5置1

GPIO\_PORTE\_AHB\_DEN\_R = 0x03;//0000 0011,PE0对应RD，PE1对应CS 置1

GPIO\_PORTG\_AHB\_DEN\_R = 0x03;//0000 0011,PG1,PG0置1

GPIO\_PORTK\_DEN\_R = 0x0f;//0000 1111,PK3-0置1

}

int main(void)

{

volatile uint32\_t ui32Loop,ui32Loop0,ui32Loop1,ui32Loop2,ui32Loop3,ui32Loop4;

ui32SysClock=SysCtlClockFreqSet((SYSCTL\_OSC\_INT | SYSCTL\_USE\_PLL | SYSCTL\_CFG\_VCO\_320), 120000000);//初始化系统时钟

SYSCTL\_RCGCGPIO\_R |= (SYSCTL\_RCGCGPIO\_R9 | SYSCTL\_RCGCGPIO\_R6 | SYSCTL\_RCGCGPIO\_R4 |SYSCTL\_RCGCGPIO\_R3| SYSCTL\_RCGCGPIO\_R2 | SYSCTL\_RCGCGPIO\_R1 | SYSCTL\_RCGCGPIO\_R0);

//使能GPIO用到的端口时钟

ui32Loop = SYSCTL\_RCGCGPIO\_R;//时钟信号赋值

GPIOint();//GPIO口初始化

InitConsole();//UART初始化

while(1)//根据时序图给相应的IO赋高低电平

{

GPIOPinWrite(GPIO\_PORTD\_BASE,GPIO\_PIN\_6|GPIO\_PIN\_5|GPIO\_PIN\_4,0x10);

GPIOPinWrite(GPIO\_PORTE\_BASE,GPIO\_PIN\_0|GPIO\_PIN\_1,0x03);

GPIOPinWrite(GPIO\_PORTE\_BASE,GPIO\_PIN\_0|GPIO\_PIN\_1,0x01);

GPIOPinWrite(GPIO\_PORTD\_BASE,GPIO\_PIN\_6|GPIO\_PIN\_5|GPIO\_PIN\_4,0x10);

GPIOPinWrite(GPIO\_PORTD\_BASE,GPIO\_PIN\_6|GPIO\_PIN\_5|GPIO\_PIN\_4,0x40);

GPIOPinWrite(GPIO\_PORTE\_BASE,GPIO\_PIN\_0|GPIO\_PIN\_1,0x03);

GPIOPinWrite(GPIO\_PORTD\_BASE,GPIO\_PIN\_6|GPIO\_PIN\_5|GPIO\_PIN\_4,0x00);

GPIOPinWrite(GPIO\_PORTE\_BASE,GPIO\_PIN\_0|GPIO\_PIN\_1,0x01);

GPIOPinWrite(GPIO\_PORTD\_BASE,GPIO\_PIN\_6|GPIO\_PIN\_5|GPIO\_PIN\_4,0x10);

SysCtlDelay(10);

GPIOPinWrite(GPIO\_PORTE\_BASE,GPIO\_PIN\_0|GPIO\_PIN\_1,0x00);

SysCtlDelay(10);

GPIOPinWrite(GPIO\_PORTD\_BASE,GPIO\_PIN\_6|GPIO\_PIN\_5|GPIO\_PIN\_4,0x50);

ui32Loop1 = (GPIO\_PORTG\_AHB\_DATA\_R&0x03);//PG1,PG0，并口线D10,D11

ui32Loop2 = (GPIO\_PORTA\_AHB\_DATA\_R&0xc0);//PA7,PA6，并口线D8,D9

ui32Loop3 = (GPIO\_PORTC\_AHB\_DATA\_R&0xf0);//PC7-4，并口线D4-D7

ui32Loop4 = (GPIO\_PORTK\_DATA\_R&0x0f) ;//PK3-0，并口线D0-D3

ui32Loop0 = (ui32Loop1%2\*8+ui32Loop1/2\*4+ui32Loop2/64)\*256+((ui32Loop3&0x10)/16\*8+(ui32Loop3&0x20)/32\*4+(ui32Loop3&0x40)/64\*2+(ui32Loop3&0x80)/128)\*16+ui32Loop4;

UARTprintf("%04d \n",ui32Loop0);

SysCtlDelay(10);

GPIOPinWrite(GPIO\_PORTE\_BASE,GPIO\_PIN\_0|GPIO\_PIN\_1,0x03);

}

}

**实验现象：**

当跳帽套上时J32的23管脚的时候，串口的数值与滚轮装置的转变而转变。在跳线帽取下时，串口的数值直接为4095。

**五、思考题**

1、将 AD 实验的代码改成用函数库实现。

#include "stdint.h"

#include "stdbool.h"

#include "driverlib/debug.h"//调试

#include "inc/hw\_types.h"//数据类型设置，寄存器访问封装

#include "inc/hw\_memmap.h"//基址

#include "driverlib/fpu.h"

#include "driverlib/sysctl.h"//使能寄存器

#include "driverlib/rom.h"

#include "driverlib/pin\_map.h"

#include "grlib/grlib.h"

#include "driverlib/gpio.h"

#include "driverlib/adc.h"

#include "inc/hw\_gpio.h"

#include "inc/tm4c1294ncpdt.h"

#include "driverlib/uart.h"

#include "utils/uartstdio.h"

volatile uint32\_t ui32SysClock;

void InitConsole(void)

{

// 由于UART0使用PA0,PA1两个引脚，因此需要使能GPIOA模块

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

// 因为有引脚复用，所以要对PA0和PA1两个引脚的功能进行选择

// 这里将它们选择为执行UART0模块的功能

GPIOPinConfigure(GPIO\_PA0\_U0RX);

GPIOPinConfigure(GPIO\_PA1\_U0TX);

// 对于PA0和PA1两个引脚，在将它们作为UART功能使用之前，需要对它们做一些有关UART的配置

GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

UARTStdioConfig(0, 115200, ui32SysClock);

}

void GPIOint() // gpio初始化

{

GPIOPinTypeGPIOInput(GPIO\_PORTA\_AHB\_BASE, GPIO\_PIN\_7|GPIO\_PIN\_6);\

GPIOPinTypeGPIOInput(GPIO\_PORTC\_AHB\_BASE,GPIO\_PIN\_7|GPIO\_PIN\_6|GPIO\_PIN\_5|GPIO\_PIN\_4);

GPIOPinTypeGPIOOutput(GPIO\_PORTD\_AHB\_BASE, GPIO\_PIN\_4|GPIO\_PIN\_5);

GPIOPinTypeGPIOOutput(GPIO\_PORTE\_AHB\_BASE, GPIO\_PIN\_0|GPIO\_PIN\_1);

GPIOPinTypeGPIOInput(GPIO\_PORTG\_AHB\_BASE, GPIO\_PIN\_0|GPIO\_PIN\_1);

GPIOPinTypeGPIOInput(GPIO\_PORTK\_BASE, GPIO\_PIN\_0|GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3);

}

int main(void)

{

volatile uint32\_t ui32Loop,ui32Loop0,ui32Loop1,ui32Loop2,ui32Loop3,ui32Loop4;

ui32SysClock=SysCtlClockFreqSet((SYSCTL\_OSC\_INT | SYSCTL\_USE\_PLL | SYSCTL\_CFG\_VCO\_320), 120000000);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);//使能GPIOA

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOC);//使能GPIOC

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOD);//使能GPIOD

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOE);//使能GPIOE

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOG);//使能GPIOG

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOK);//使能GPIOK

ui32Loop = SYSCTL\_RCGCGPIO\_R;

GPIOint();

InitConsole();

while(1)

{

GPIOPinWrite(GPIO\_PORTD\_BASE,GPIO\_PIN\_6|GPIO\_PIN\_5|GPIO\_PIN\_4,0x10);

GPIOPinWrite(GPIO\_PORTE\_BASE,GPIO\_PIN\_0|GPIO\_PIN\_1,0x03);

GPIOPinWrite(GPIO\_PORTE\_BASE,GPIO\_PIN\_0|GPIO\_PIN\_1,0x01);

GPIOPinWrite(GPIO\_PORTD\_BASE,GPIO\_PIN\_6|GPIO\_PIN\_5|GPIO\_PIN\_4,0x10);

GPIOPinWrite(GPIO\_PORTD\_BASE,GPIO\_PIN\_6|GPIO\_PIN\_5|GPIO\_PIN\_4,0x40);

GPIOPinWrite(GPIO\_PORTE\_BASE,GPIO\_PIN\_0|GPIO\_PIN\_1,0x03);

GPIOPinWrite(GPIO\_PORTD\_BASE,GPIO\_PIN\_6|GPIO\_PIN\_5|GPIO\_PIN\_4,0x00);

GPIOPinWrite(GPIO\_PORTE\_BASE,GPIO\_PIN\_0|GPIO\_PIN\_1,0x01);

GPIOPinWrite(GPIO\_PORTD\_BASE,GPIO\_PIN\_6|GPIO\_PIN\_5|GPIO\_PIN\_4,0x10);

SysCtlDelay(2);//2

GPIOPinWrite(GPIO\_PORTE\_BASE,GPIO\_PIN\_0|GPIO\_PIN\_1,0x00);

SysCtlDelay(3);//3

GPIOPinWrite(GPIO\_PORTD\_BASE,GPIO\_PIN\_6|GPIO\_PIN\_5|GPIO\_PIN\_4,0x50);

SysCtlDelay(1);//1

ui32Loop1 = GPIOPinRead(GPIO\_PORTG\_AHB\_BASE,GPIO\_PIN\_1)+GPIOPinRead(GPIO\_PORTG\_AHB\_BASE,GPIO\_PIN\_0);

ui32Loop2 = GPIOPinRead(GPIO\_PORTA\_AHB\_BASE,GPIO\_PIN\_7)+GPIOPinRead(GPIO\_PORTA\_AHB\_BASE,GPIO\_PIN\_6);

ui32Loop3 = GPIOPinRead(GPIO\_PORTC\_AHB\_BASE,GPIO\_PIN\_7)+GPIOPinRead(GPIO\_PORTC\_AHB\_BASE,GPIO\_PIN\_6)+GPIOPinRead(GPIO\_PORTC\_AHB\_BASE,GPIO\_PIN\_5)+GPIOPinRead(GPIO\_PORTC\_AHB\_BASE,GPIO\_PIN\_4);

ui32Loop4 = GPIOPinRead(GPIO\_PORTK\_BASE,GPIO\_PIN\_3)+GPIOPinRead(GPIO\_PORTK\_BASE,GPIO\_PIN\_2)+GPIOPinRead(GPIO\_PORTK\_BASE,GPIO\_PIN\_1)+GPIOPinRead(GPIO\_PORTK\_BASE,GPIO\_PIN\_0);

ui32Loop0 = (ui32Loop1%2\*8+ui32Loop1/2\*4+ui32Loop2/64)\*256+((ui32Loop3&0x10)/16\*8+(ui32Loop3&0x20)/32\*4+(ui32Loop3&0x40)/64\*2+(ui32Loop3&0x80)/128)\*16+ui32Loop4;

UARTprintf("%04d \n",ui32Loop0);

GPIOPinWrite(GPIO\_PORTE\_BASE,GPIO\_PIN\_0|GPIO\_PIN\_1,0x03);

SysCtlDelay(10);

}

}

**六、实验体会与心得**

经过这次曲折的实验，我深入接触了并行总线的数据传输，和以往的串行总线还是有很大的区别的。串行数据传输时，数据是一位一位地在数据线上传输的，它只需要很少的连接线即可达到数据传输的目的，但这也限制了它的最大传输速率。而在并行通信传输中有多个数据位，同时在两个设备之间传输。发送设备将这些数据位通过一一对应的数据线传送给接收设备。接收设备可同时接收到这些数据，不需要做任何变换就可直接使用。并行方式主要用于近距离通信，且速率大大提高，但同时占用了很多的单片机引脚。

提早写完程序后，曾试图将DAC和ADC结合起来，观察实验指导书上的原理图发现，J32跳帽接到的12时，DAC的输出经过一个运算放大器后连接到了ADC上，所以将两个程序合在一起后，用串口输出ADC采集到的值发现和我设置DAC输出的值相差不大，正当我高兴的时候，同学跟我说了个问题，ADC与DAC的并行总线与单片机相连的是同一批线， ADC还没将总线上的数据覆盖，单片机就去读取数据，那不就是当时DAC设置的值了。当时老师也觉得有道理，我也没有多想。现在回顾一下发现不影响，因为整个程序是串行运行的，在给DAC设置的时候，单片机不会去和ADC交互，而且在读取ADC的时候已经为了确保ADC的数据采集而延时一段时间，所以和我得到的实验现象也差不多，不是ADC和DAC完全一样的，还是有点误差的。