西安电子科技大学

数字电子技术基础 课程设计报告

实验名称 模数-数模转换控制器

机电工程学院 学院 1504032 班

姓名 吕昕远 学号 15040310079

同作者

实验日期 2017 年 10 月 9 日

成 绩

指导教师评语：   指导教师： 年 月 日
实验报告内容基本要求及参考格式 一、实验目的 二、实验所用仪器（或实验环境） 三、实验基本原理及步骤（或方案设计及理论计算） 四、实验数据记录（或仿真及软件设计） 五、实验结果分析及回答问题（或测试环境及测试结果）

一、实验目的

1. 掌握模数、数模转换相关概念
2. 实现 FPGA 对 ADC 和 DAC 的控制

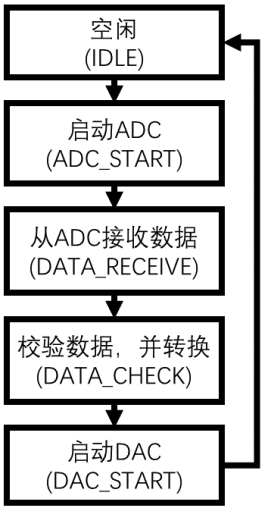
二、实验所用仪器（或实验环境）

Quartus II 9.1

三、实验基本原理及步骤（或方案设计及理论计算）

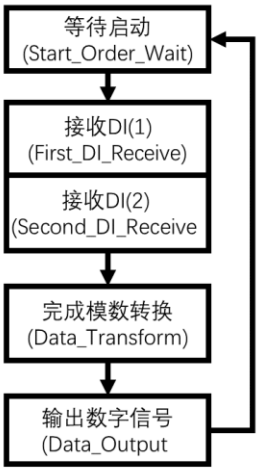
控制器部分采用有限状态机形式进行设计，并模拟 ADC0832 工作时序，**实现了“ADC0832”测试模块**。

控制器状态转移如下图：



为了更好的测试控制器的工作状态，参照 ADC0832 的工作时序[1]和使用方法[2]实现了“ADC0832”测试模块。

“ADC0832”测试模块工作状态转换如下图



四、实验数据记录（或仿真及软件设计）

控制器（AD_DA.vhd[3]）

```
library ieee;
use ieee.STD_LOGIC_ARITH.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity AD_DA is
	port(
		--控制 ADC0832 的相关端口
		-- port of ADC0832
		clk_1 : in std_logic; --clock
		CH0_1 : in integer; --analog data entrance
		CH1_1 : in integer; --analog data entrance
		CS_output : out std_logic;
		DI_output : out std_logic;
		state_signal_1 : buffer std_logic_vector(3 downto 0); --状态编号
		DO_1 : buffer std_logic; --Data Out
		DataA_output: out std_logic_vector(7 downto 0);
		DataB_output: out std_logic_vector(7 downto 0);
		stu_no:buffer integer; --学号
		--控制 DAC0832 的相关端口
		--port of DAC0832
		D : out std_logic_vector(7 downto 0);
		WR1 : out std_logic;
		XFER : out std_logic;
		WR2 : out std_logic
		);
	signal DI_1 : std_logic:='0';
	signal CS_1 : std_logic:='1'; --使能端
end AD_DA;

architecture behavior of AD_DA is
	--ADC0832 测试模块
	COMPONENT ADC0832
		port(
		clk : in std_logic; --clock
		CH0,CH1 : in integer; --analog data entrance
		CS : in std_logic; --CHIP Select
		DI : In std_logic; --Data In
		state_signal : buffer std_logic_vector(3 downto 0); --state Out
		DO : buffer std_logic --Data Out
		);
```

```
    END COMPONENT ADC0832;
    --状态定义
    type state is (IDLE,ADC_START,DATA_RECEIVE,DATA_CHECK,DAC_START);
    signal current_state:state; --现态
    signal next_state:state; --次态
    signal DataA : std_logic_vector(7 downto 0); --正向接收数据
    signal DataB : std_logic_vector(7 downto 0); --反向接收数据
    signal receive_order:std_logic; --接收数据进程启动的控制信号
    signal start_order:std_logic; --ADC 启动的控制信号
    signal temp:integer;
    signal Parallel_Data:std_logic_vector(7 downto 0); --并行数据
    shared VARIABLE COUNT1:INTEGER:=0;
    shared VARIABLE COUNT2:INTEGER:=0;
    shared VARIABLE COUNT3:INTEGER:=0;
begin
    --实例化“ADC0832”
    u1:ADC0832 PORT MAP(clk=>clk_1, CH0=>CH0_1, CH1=>CH1_1, CS=>CS_1,DI=>DI_1,state_signal=>state
_signal_1,DO=>DO_1);
    --时序电路
    counter : process(clk_1)
    begin
        if(clk_1'event and clk_1 = '0') then
            current_state <= next_state;
        end if;
    end process;
    --组合电路
    controller : process(current_state)
    begin

            case current_state is
                when IDLE =>
                    --初始化各个信号
                    --init the signal
                    CS_1 <= '0';
                    WR1 <= '1';
                    receive_order <= '0';
                    start_order  <= '0';
                    --DI_1 <= '0';
                    --设置次态
                    next_state <= ADC_START;
                when ADC_START =>
                    --启动 ADC0832
                    --send start order to ADC0832
                    if(COUNT2 < 4) then
```

```
                start_order <= '1';
                next_state <= ADC_START;
            else
                start_order <= '0';
                next_state <= DATA_RECEIVE;
            end if;

        when DATA_RECEIVE =>
            --从 ADC0832 接收转换完成的串行数据
            --receive data from ADC0832
            if(COUNT1 < 15) then
                --启动数据接收进程
                receive_order <= '1';
                next_state <= DATA_RECEIVE;
            else
                receive_order <= '0';
                next_state <= DATA_CHECK;
            end if;
        when DATA_CHECK =>
            --校验数据
            --check if the data is correct
            if(DataA = DataB) then
                --计算出学号
                temp <= CONV_INTEGER(DataA);
                stu_no <= (255 - temp)/2;
                next_state <= DAC_START;
            else
                stu_no <= -1;
                next_state <= DAC_START;
            end if;
            Parallel_Data <= CONV_STD_LOGIC_VECTOR((255 - temp)/2,8);
        when DAC_START =>
            --启动 DAC0832
            --send start order to DAC0832
            D <= Parallel_Data;
            WR1 <= '0';
            WR2 <= '0';
            XFER <= '0';
            next_state <= DAC_START;
        when others =>
            next_state <= IDLE;

end case;
DataA_output <= DataA;
```

```
            DataB_output <= DataB;
            CS_output <= CS_1;
            DI_output <= DI_1;

end process;


--ADC0832 控制进程
adc_work : process(start_order,clk_1)
begin
      if(start_order = '1') then
            if(clk_1'event and clk_1 = '0') then
                  if(COUNT2 = 0) then
                        --START_BIT
                        DI_1 <= '1';
                  end if;
                  if(COUNT2 = 1) then
                        --First DI
                        DI_1 <= '0';
                  end if;
                  if(COUNT2 = 2) then
                        --Second DI
                        DI_1 <= '1';
                  end if;
                  if(COUNT2 = 3) then
                        --Transform
                        DI_1 <= '0';
                  end if;
                  COUNT2:=COUNT2+1;
            end if;
      end if;
end process;
--接收 ADC0832 数据进程
receive : process(receive_order,clk_1)
begin
      if(receive_order = '1') then
            if(clk_1'event and clk_1='0') then
                  --先从高位到低位，再从高位到低位
                  if(COUNT1<7) then
                        DataA <= DataA(6 downto 0) & DO_1;
                  end if;
                  if(COUNT1=7) then
                        DataA <= DataA(6 downto 0) & DO_1;
                        DataB <= DO_1 & DataB(7 downto 1);
```

```
                    end if;
                    if(COUNT1<15 and COUNT1 >7) then
                        DataB <= DO_1 & DataB(7 downto 1);
                    end if;
                    COUNT1:=COUNT1+1;
                end if;
            end if;
        end process;
end behavior;
```

ADC0832 测试模块（ADC0832.vhd[4]）

```
library ieee;
use ieee.STD_LOGIC_ARITH.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity ADC0832 is
    port(
        clk : in std_logic;
        CH0,CH1 : in integer; --“模拟”信号输入
        CS : in std_logic; --CHIP Select
        DI : In std_logic; --Data In
        state_signal : buffer std_logic_vector(3 downto 0); --state Out
        DO : buffer std_logic --Data Out
        );

    signal output_index:std_logic_vector(3 downto 0):="0000";
    signal data_input_model : std_logic_vector(1 downto 0);
    signal data : std_logic_vector(7 downto 0);
    signal receive_data : boolean:=false;

end ADC0832;

architecture behavior of ADC0832 is
    --状态定义
    type state is (Start_Order_Wait,First_DI_Receive,Second_DI_Receive,Data_Transform,Data_Output);
    signal current_state:state; --现态
    signal next_state:state:=Start_Order_Wait; --次态
    signal DI1,DI0:std_logic:='0';
    signal output_order:std_logic;
begin
    --时序电路
    synch:process(clk)
```

```
begin
	--change state with clock
	if(clk'event and clk = '0') then


		current_state <= next_state;


	end if;
end process;
--组合电路
state_trans:process(CS,DI,current_state)
begin
	state_signal <= "1111";
	--work only if CS is low
	if(CS = '0') then
		case current_state is
			when Start_Order_Wait =>
				--等待启动命令
				-- wait the START BIT
				if(DI = '1')then
					state_signal <= "0001";
					next_state <= First_DI_Receive;
				else
					state_signal <= "0010";
					next_state <= Start_Order_Wait;
				end if;
				output_order <= '1';
			when First_DI_Receive =>
				--数据通道选择
				state_signal <= "0011";
				DI1 <= DI;
				next_state <= Second_DI_Receive;
			when Second_DI_Receive =>
				--数据通道选择
				state_signal <= "1000";
				DI0 <= DI;
				data_input_model(1) <= DI1;
				data_input_model(0) <= DI0;
				next_state <= Data_Transform;
			when Data_Transform =>
				--完成模数转换
				state_signal <= "0101";
				case data_input_model is
```

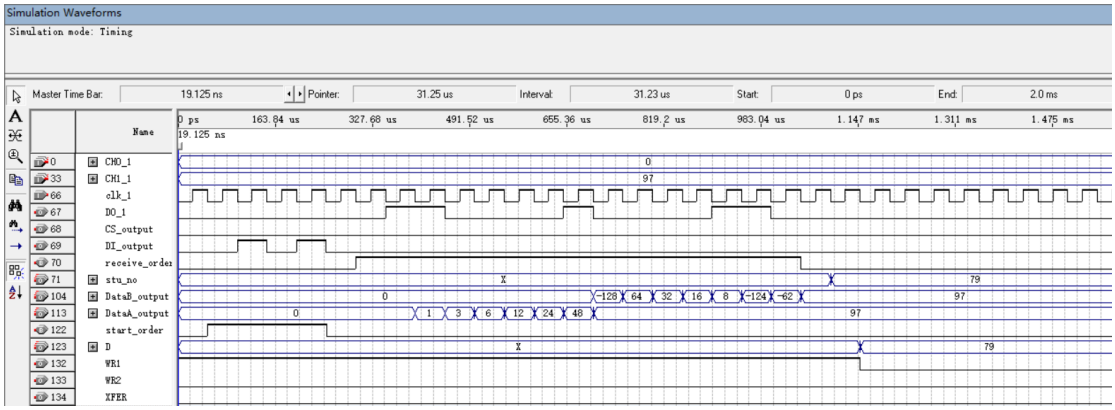
```
                        when "00" =>
                            data<=CONV_STD_LOGIC_VECTOR(CH0-CH1,8);
                        when "01" =>
                            data<=CONV_STD_LOGIC_VECTOR(CH1-CH0,8);
                        when "10" =>
                            data<=CONV_STD_LOGIC_VECTOR(CH0,8);
                        when "11" =>
                            data<=CONV_STD_LOGIC_VECTOR(CH1,8);
                    end case;
                    next_state <= Data_Output;
                    output_order <= '0';

                when Data_Output =>
                    --串行输出
                    state_signal <= "0110";
                    output_order <= '0';
                    next_state <= Data_Output;
                when others =>
                    state_signal <= "1001";
                    next_state <= Start_Order_Wait;
            end case;
        else
            next_state <= Start_Order_Wait;
        end if;
end process;

--数据输出进程
output:process(clk,output_order)
begin
            if(clk'event and clk = '0') then
                if(output_order = '0')then
                        --先高位在前，再低位在前
                        case output_index is
                            when "0000" =>
                                DO <= data(7);
                                output_index <= output_index + "0001";
                            when "0001" =>
                                DO <= data(6);
                                output_index <= output_index + "0001";
                            when "0010" =>
                                DO <= data(5);
                                output_index <= output_index + "0001";
                            when "0011" =>
                                DO <= data(4);
```

```
                output_index <= output_index + "0001";
            when "0100" =>
                DO <= data(3);
                output_index <= output_index + "0001";
            when "0101" =>
                DO <= data(2);
                output_index <= output_index + "0001";
            when "0110" =>
                DO <= data(1);
                output_index <= output_index + "0001";
            when "0111" =>
                DO <= data(0);
                output_index <= output_index + "0001";
            when "1000" =>
                DO <= data(1);
                output_index <= output_index + "0001";
            when "1001" =>
                DO <= data(2);
                output_index <= output_index + "0001";
            when "1010" =>
                DO <= data(3);
                output_index <= output_index + "0001";
            when "1011" =>
                DO <= data(4);
                output_index <= output_index + "0001";
            when "1100" =>
                DO <= data(5);
                output_index <= output_index + "0001";
            when "1101" =>
                DO <= data(6);
                output_index <= output_index + "0001";
            when "1110" =>
                DO <= data(7);
                output_index <= output_index + "0001";
            when "1111" =>
                DO <= '0';
                output_index<=output_index;
        end case;
    end if;
  end if;
 end process;
end behavior;
```

波形仿真



为满足输出学号的要求，在 ADC0832 测试模块的“模拟”输入端输入 97。启动后，控制器先进入 ADC_START 状态，start_order 置为高电平，adc_work 进程启动 DI_output 依次输出“101”，启动 ADC0832 并完成通道选择选择，然后进入 DATA_RECEIVE 状态，receive_order 置为高电平，receive 进程启动，经过 15 个周期完成数据的接收和转换计算处理，然后进入 DAC_START 状态，将并行的数据 79 传送至 D，WR1 置为低电平，使 DAC0832 在单缓冲工作模式下开始转换。

Flow Summary

五、实验结果分析及回答问题（或测试环境及测试结果）

适用的速度及精度场合

从速度方面看，ADC0832 工作频率为 250KHZ，而 DAC0832 的转换时间约为 1us，即 DAC0832 的转换速度远快与 DAC0832，因此可使用 250KHZ 的时钟源作为两者共同的时钟源。

从精度方面看，作为单通道模拟信号输入时 ADC0832 的输入电压是 0~5V 且 8 位分辨率时的电压精度为 19.53mV。如果作为由 IN+与 IN-输入的输入时，可是将电压值设定在某一个较大范围之内，从而提高转换的宽度。但值得注意的是，在进行 IN+与 IN-的输入时，如果 IN-的电压大于 IN+的电压则转换后的数据结果始终为 00H。[5]

总结

通过这次课程设计，又一次复习了之前在数电课程中学习到的 VHDL 语言程序设计，也让我对数字电路有了更深刻的理解和感悟。

由于开始时对这次课程设计题目理解不到位，本以为是要用 VHDL 程序模拟 ADC 和 DAC 的工作过程，而后经过与同学们的讨论才明白是要为 FPGA 编程从而控制 ADC0832 和 DAC0832 工作。至此我才明白了 FPGA 在电路设计中的作用，以及数字电路和模拟电路是如何协同工作的。

在设计程序的过程中，由于 ADC0832 和 DAC0832 工作时序的特殊性，要以有限状态机的形式设计程序，在自己分析设计状态转换的过程中，将状态转换的程序设计和时序电路的状态转换联系了起来。而编程的过程中，由于对硬件语言的理解不到位，用高级语言的思维去理解时导致了一些问题，在解决问题的过程中加深了对程序设计底层实现的理解。

在完善注释和文档的过程中，也锻炼了协同工作的能力，当项目规模达到需要团队协作完成时，文档和注释就显得尤为重要。尤其对于后续开发，若是没有完善的文档，就要增加额外的学习成本，降低开发效率。

总的来说，这次课程设计对我的实践能力有很大的提升，仅仅掌握理论知识是远远不够的，能将理论知识实践才真正的能力。

写给曾经的自己

当你看到这段话时，也许你即将开始数电的学习，并不想告诉你什么学习这门课的奇技淫巧，只想说说自己对这门课的感悟。

从与非门，到数据选择器，再到移位寄存器，当信息抽象为二进制的高低电平，在纷繁复杂的电路里有条不紊的流动，当亿万自由的电子被囚禁在指甲盖大的芯片，携带着各自的使命不停奔腾，我只得叹服先人的伟大。

自然创造了人类，人类为自然安排秩序。从简单的门电路到单片机周围的最小系统，再到 PC 中各种超大规模电路，都是先人们一个世纪以来研究的结晶。

学习这些知识，不应畏惧，而是应该怀揣着敬意和兴趣。当你看着 LED 灯在计数器的控制下灵性的跳动，我相信你也会感受到先人钻研时的那份激情。

Trust the process!

参考文献

[1]. 百度文库.ADC0832 芯片介绍[DB/OL].https://wenku.baidu.com/view/15613a8183d049649b665870.html,2011-10-29/2017-10-9

[2]. 新浪博客.ADC0832 的使用方法[DB/OL].http://blog.sina.com.cn/s/blog_62b33cb901015zyy.html,2012-09-11/2017-10-09

[3]. 吕昕远.AD_DA.vhd[DB/OL].https://github.com/pokerfaceSad/DigitalCircuitsCourseDesign/blob/master/AD_DA.vhd,2017-10-09/2017-10-09

[4]. 吕昕远.ADC0832.vhd[DB/OL].https://github.com/pokerfaceSad/DigitalCircuitsCourseDesign/blob/master/ADC0832.vhd,2017-10-09/2017-10-09

[5]. 百度百科.ADC0832[DB/OL].https://baike.baidu.com/item/ADC0832/4795234?fr=Aladdin,2017-08-12/2017-10-09