



湖南工业大学
HUNAN UNIVERSITY OF TECHNOLOGY

自主性学习与研究项目二

课程名称	EDA 技术			
项目名称	基于 FPGA 的数字式工频有效值多用表			
实验地点	电气楼 203	实验时间	2023 年 5 月 17 日	
考核项目	目标 1	目标 2	目标 3	目标 4
考核成绩				
指导教师	谭会生			
学院名称	轨道交通学院			
学生班级	电子科学与技术 2102			
学生姓名	付允松			
学生学号	21419000503			

湖南工业大学轨道交通学院 制

项目一 基于 FPGA 的数字式工频有效值多用表

一、研究项目概述

设计并制作一个能同时对一路工频交流电（频率波动范围为 $50 \pm 1\text{Hz}$ 、有失真的正弦波）的电压有效值、电流有效值、有功功率、无功功率、功率因数进行测量的数字式多用表。框图如图 11.234 所示

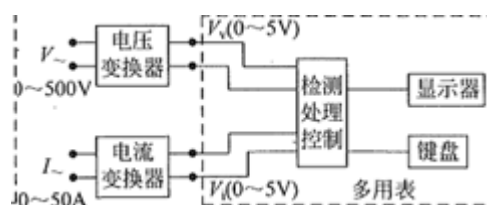


图 1

1. 测量功能及量程范围

- (1) 交流电压： $0 \sim 500\text{V}$;
- (2) 交流电流： $0 \sim 50\text{A}$;
- (3) 有功功率： $0 \sim 25\text{kW}$;
- (4) 无功功率： $0 \sim 25\text{kvar}$;
- (5) 功率因数（有功功率 / 视在功率）；

为便于本项目的设计与制作，设定待测 $0 \sim 500\text{V}$ 的交流电压、 $0 \sim 50\text{A}$ 的交流电流均已经被相应的变换器转换为 $0 \sim 5\text{V}$ 的交流电压。

2. 准确度

- (1) 显示为位(0.000~4.999)，有过量程指示；
- (2) 交流电压和交流电流： $\pm (0.8\% \text{ 读数} + 5 \text{ 个字})$ ，例如：当被测电压为 300V 时，读数误差应小于 $\pm (0.8\% \times 300\text{V} + 0.5\text{V}) = \pm 2.9\text{V}$ ；
- (3) 有功功率和无功功率： $\pm (1.5\% \text{ 读数} + 8 \text{ 个字})$ ；
- (4) 功率因数： ± 0.01 。

3. 功能选择

用按键选择交流电压、交流电流、有功功率、无功功率和功率因数的测量与显示。

二、研究项目设计方案比较

1) 设计方案一：

基于硬件电路的方案：

硬件电路方案使用模拟电路和电子元器件来实现数字式工频有效值多用表。该方案的优点是实时性好，精度高，而且可以通过外部电路控制输入和输出。缺点是需要大量的元器件和电路布线，成本高，而且难以实现复杂的算法。

2) 设计方案二：

基于微处理器的方案：

基于微处理器的方案使用一个或多个微处理器来实现数字式工频有效值多用表。该方案的优点是可使用现成的微处理器开发工具和软件库，易于编程和调试。缺点是响应时

间可能较慢，精度可能不够高，而且需要额外的外围设备来实现输入和输出。

3) 设计方案三

基于 FPGA 的方案

基于微处理器的方案使用一个或多个微处理器来实现数字式工频有效值多用表。该方案的优点是可以使用现成的微处理器开发工具和软件库，易于编程和调试。缺点是响应时间可能较慢，精度可能不够高，而且需要额外的外围设备来实现输入和输出。

综合比较，基于 FPGA 的方案是实现数字式工频有效值多用表最优秀的选择，因为它可以实现复杂的算法和高精度的测量结果，并且具有极高的实时性和可编程性。同时，FPGA 技术在工业和科研领域已经得到广泛应用，可以利用现有的 FPGA 开发工具和软件库来实现设计

三、 研究项目系统设计

1. 基于 FPGA 的数字式工频有效值多用表

根据系统的设计要求，系统可分为分频，ROM，AD0809，有效值，有功功率，无功功率，功率因数，选择控制，显示控制。其设计思想如下：

(1) 分频：通过对输入时钟进行不同的分频，减少功耗，便在不同的时钟周期中执行不同的操作，从而实现更复杂的时序控制。

(2) ROM：通过 matlab 生成 mif 文件保存电压电流的波形模数转换值，进行仿真测试

(3) AD0809：进行 AD 转换的时序控制

(4) 有效值：计算平方和：将每个采样值平方，然后将这些平方值相加，得到平方和。计算平均值：将平方和除以采样值的总数，得到平均值。计算有效值：将平均值开方，得到输入电流的有效值。需要注意的是，在进行模拟信号转换和计算有效值时，要考虑采样率、采样时间和精度等因素。如果采样率过低或采样时间不足，可能会导致信号失真和计算错误。因此，在进行输入电流有效值计算时，需要进行合理的参数设置和精度控制，以保证计算结果的准确性和可靠性。

(5) 有功功率，无功功率，功率因数：将输入信号的瞬时电压和电流值进行乘法运算，得到输入信号的瞬时功率值。然后将功率值进行积分，得到有功功率和无功功率的瞬时值。计算平均有功功率和无功功率：将有功功率和无功功率的瞬时值进行平均，得到输入信号的平均有功功率和无功功率的值。计算功率因数：根据定义，功率因数等于有功功率与视在功率之比。因此，需要先计算输入信号的视在功率，即将输入信号的瞬时电压和电流值进行乘法运算，得到输入信号的瞬时视在功率值。然后将有功功率和视在功率的平均值进行除法运算，得到输入信号的功率因数。

(6) 选择控制：通过不同数控制信号，输出不同的测量结果

(7) 显示控制：将测量结果在数码管上显示出来

2.系统总体电路的设计

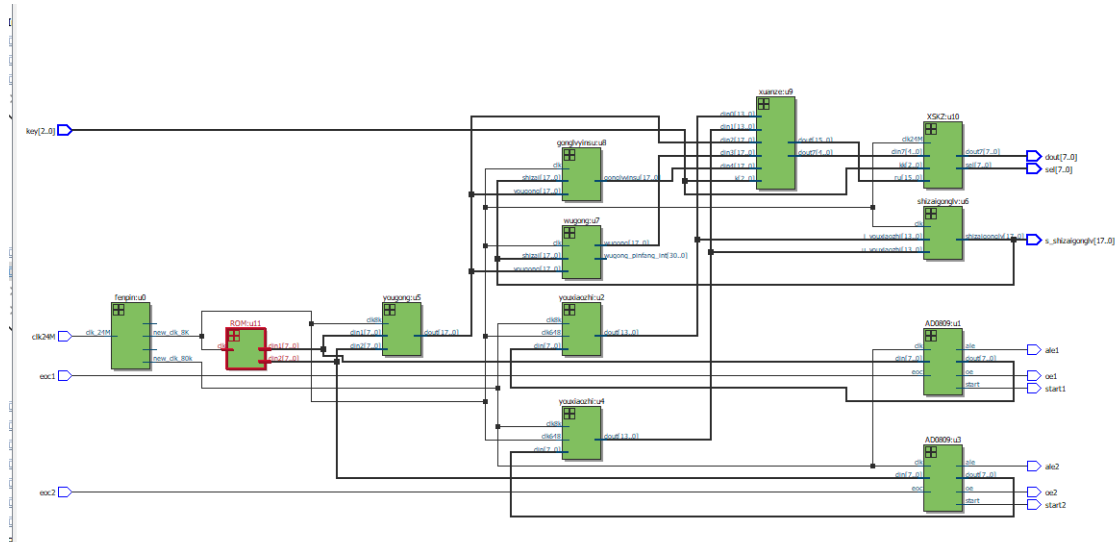


图 2

四、系统主要 VHDL 源程序设计

TOP:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity TOP is
    --generic(constant din1:std_logic_vector(7 downto 0):="10000101";--I
    --constant din2:std_logic_vector(7 downto 0):="10000100");--U
    port(clk24M:in std_logic;
        eoc1:in std_logic;--sw1
        --din1:in std_logic_vector(7 downto 0);
        eoc2:in std_logic;--sw2
        --din2:in std_logic_vector(7 downto 0);
        key: in std_logic_vector(2 downto 0);--sw3,4,5
        start1,oe1,ale1:out std_logic;
        start2,oe2,ale2:out std_logic;
        dout: out std_logic_vector(7 downto 0);
        -- s_new_clk_8K,s_new_clk_80k,s_new_clk_24M:buffer std_logic;
        -- s_dout_U_rms:buffer std_logic_vector(13 downto 0);
        -- s_dout_I_rms:buffer std_logic_vector(13 downto 0);
        -- s_shizaigonglv:buffer std_logic_vector(17 downto 0);
        -- s_wugong:buffer std_logic_vector(17 downto 0);
        s_shizaigonglv: buffer std_logic_vector(17 downto 0);
        sel: out std_logic_vector(7 downto 0));
end TOP;
architecture art of TOP is
    --component
```

```

component youxiaozhi is
port(clk648:in std_logic;
      clk8k:in std_logic;
      din:in std_logic_vector(7 downto 0);
      dout:out std_logic_vector(13 downto 0));
end component;

component yougong is
port(clk8k:in std_logic;
      din1:in std_logic_vector(7 downto 0);
      din2:in std_logic_vector(7 downto 0);
      dout:out std_logic_vector(17 downto 0));
end component;

component wugong is
port(clk:in std_logic;
      --Reactive power also needs to be square rooted for the 17 digits within 1 second
      --so it can be 8k here
      yougong:in std_logic_vector(17 downto 0);
      shizai:in std_logic_vector(17 downto 0);
      wugong_pinfang_int:buffer natural range 0 to
2147483647;--integer-max-31bit
      --wugong_pinfang_int:buffer natural range 0 to
2147483647;--integer-max-31bit
      --yougong_int:buffer integer range 0 to 131071;--18bit
      --shizai_int:buffer integer range 0 to 131071;
      wugong:out std_logic_vector(17 downto 0));
end component;

component gonglvvinsu is
port(clk:in std_logic;
      --The apparent power only needs to calculate the multiplication result in 1 second
      --it can be 8k
      shizai:in std_logic_vector(17 downto 0);--17
      yougong:in std_logic_vector(17 downto 0);--17
      --s_result_100_25:buffer std_logic_vector(24 downto 0);
      --s_gonglvvinsu25:buffer std_logic_vector(24 downto 0);
      gonglvvinsu:out std_logic_vector(17 downto 0));
end component;

component fenpin is
port(clk_24M:in std_logic;
      --The newly generated clock has 80K 8K 24M
      new_clk_8K:out std_logic;

```

```

        new_clk_80k:out std_logic;
        new_clk_24M:out std_logic;
        new_clk_1hz:out std_logic);
end component fenpin;

component AD0809 is
    port(clk,eoc:in std_logic;
          din:in std_logic_vector(7 downto 0);
          start,oe,ale:out std_logic;
          dout:out std_logic_vector(7 downto 0));
end component AD0809;

component shizaigonglv is
    port(clk:in std_logic;
          --The apparent power only needs to calculate the multiplication result in 1
second
          --it can be 8k
          u_youxiaozhi:in std_logic_vector(13 downto 0);
          i_youxiaozhi:in std_logic_vector(13 downto 0);
          shizaigonglv:out std_logic_vector(17 downto 0));
end component shizaigonglv;

component xuanze is
    port(k:in std_logic_vector(2 downto 0);
          din0,din1:in std_logic_vector(13 downto 0);--youxiaozhi
          din2:in std_logic_vector(17 downto 0);--yougong
          din3:in std_logic_vector(17 downto 0);--wugong
          din4:in std_logic_vector(17 downto 0);--gonglvinsu
          dout:out std_logic_vector(15 downto 0);
          dout7:out std_logic_vector(4 downto 0));
end component xuanze;

component XSKZ is
    port(kk:in std_logic_vector(2 downto 0);
          ru:in std_logic_vector(15 downto 0);
          clk24M:in std_logic;
          --The faster the display control clock, the better
          din7:in std_logic_vector(4 downto 0);
          dout7:out std_logic_vector(7 downto 0);
          sel:out std_logic_vector(7 downto 0));
end component XSKZ;

component ROM is
    port(clk:in std_logic;

```

```

        din1:out std_logic_vector(7 downto 0);
        din2:out std_logic_vector(7 downto 0));
end component ROM;

--signal
signal s_new_clk_8K,s_new_clk_80k,s_new_clk_24M,s_new_clk_1hz: std_logic;
signal s_din1,s_din2:std_logic_vector(7 downto 0);
signal s_dout_I:std_logic_vector(7 downto 0);
signal s_dout_I_rms:std_logic_vector(13 downto 0);
signal s_dout_U:std_logic_vector(7 downto 0);
signal s_dout_U_rms:std_logic_vector(13 downto 0);
signal s_yougong:std_logic_vector(17 downto 0);
--signal s_shizaigonglv:std_logic_vector(17 downto 0);
signal s_wugong:std_logic_vector(17 downto 0);
signal s_wugong_pinfang_int: natural range 0 to 2147483647;--integer-max-31bit
signal s_gonglvvinsu:std_logic_vector(17 downto 0);
signal s_xuanze_dout:std_logic_vector(15 downto 0);
signal s_xuanze_dout7:std_logic_vector(4 downto 0);

begin

    u0:fenpin
    map(clk24M,s_new_clk_8K,s_new_clk_80k,s_new_clk_24M,s_new_clk_1hz);
    --Current AD0809
    u1:AD0809 port map(s_new_clk_80k,eoc1,s_din1,start1,oe1,ale1,s_dout_I);--sin
    --Current youxiaozhi
    u2:youxiaozhi port map(s_new_clk_8k,s_new_clk_80K,s_dout_I,s_dout_I_rms);
    --Voltage AD0809
    u3:AD0809 port map(s_new_clk_80k,eoc2,s_din2,start2,oe2,ale2,s_dout_U);--cos
    --Voltage youxiaozhi
    u4:youxiaozhi port map(s_new_clk_8k,s_new_clk_80K,s_dout_U,s_dout_U_rms);
    --yougong
    u5:yougong port map(s_new_clk_8K,s_din1,s_din2,s_yougong);
    --shizai
    u6:shizaigonglv
    map(s_new_clk_8K,s_dout_U_rms,s_dout_I_rms,s_shizaigonglv);
    --wugong
    u7:wugong
    map(s_new_clk_8K,s_yougong,s_shizaigonglv,s_wugong_pinfang_int,s_wugong);
    --gonglvvinsu
    u8:gonglvvinsu port map(s_new_clk_8K,s_shizaigonglv,s_yougong,s_gonglvvinsu);
    --xuanze
    u9:xuanze

```

```

map(key, s_dout_I_rms, s_dout_U_rms, s_yougong, s_wugong, s_gonglvinsu,
      s_xuanze_dout, s_xuanze_dout7);

--XSKZ
u10:XSKZ port map(key, s_xuanze_dout, s_new_clk_8K, s_xuanze_dout7, dout, sel);

--ROM
u11:ROM port map(s_new_clk_8K, s_din1, s_din2);

end art;

ROM:
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity ROM is
    port(clk:in std_logic;
          din1:out std_logic_vector(7 downto 0);--sin
          din2:out std_logic_vector(7 downto 0));--cos
end entity;
architecture art of ROM is
--component
    component sin is
        port
        (
            address      : in std_logic_vector (7 downto 0);
            clock        : in std_logic  := '1';
            q            : out std_logic_vector (7 downto 0)
        );
    end component sin;
    component cos is
        port
        (
            address      : in std_logic_vector (7 downto 0);
            clock        : in std_logic  := '1';
            q            : out std_logic_vector (7 downto 0)
        );
    end component cos;
--signal
    signal s_address_sin:std_logic_vector(7 downto 0);
    signal s_address_cos:std_logic_vector(7 downto 0);
begin
    process(clk)
    begin
        if(clk'event and clk='1') then

```



```

--The address is 8 bits,
-- so resetting to zero after overflow is equivalent to clearing it.
    s_address_sin<=s_address_sin+1;
    s_address_cos<=s_address_cos+1;
end if;
end process;
u0:sin port map(s_address_sin,clk,din1);
u1:cos port map(s_address_cos,clk,din2);

end architecture;

```

```

fenpin:
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity fenpin is
    port(clk_24M:in std_logic;
--The newly generated clock has 80K 8K 24M
        new_clk_8K:out std_logic;
        new_clk_80k:out std_logic;
        new_clk_24M:out std_logic;
        new_clk_1hz:out std_logic);
end entity fenpin;
architecture art of fenpin is
begin
    --new_clk_24M<=clk_24M;
    process(clk_24M)
        variable m_8k:std_logic;
        variable m_80k:std_logic;
        variable m_1hz:std_logic;
        variable data_8k:integer range 0 to 2999;--8k
        variable data_80k:integer range 0 to 299;--80k
        variable data_1hz:integer range 0 to 23999999;--1hz
    begin
        if (clk_24M'event and clk_24M='1') then
            --if data_8k<2999 then----8k--divide 3000
            if data_8k<1 then--divide 2--Simulation
                data_8k:=data_8k+1;
            else
                data_8k:=0;
                m_8k:=not(m_8k);
            end if;
            --if data_1hz<24999999 then----1hz--divide 24000000

```

```

    if data_1hz<1 then--divide 2--Simulation
        data_1hz:=data_1hz+1;
    else
        data_1hz:=0;
        m_1hz:=not(m_1hz);
    end if;
--if data_80k<299 then--80k--divide 300
if data_80k<1 then--divide 2--Simulation
    data_80k:=data_80k+1;
else
    data_80k:=0;
m_80k:=not(m_80k);
end if;
new_clk_8K<=m_8k;
new_clk_80K<=m_80k;
new_clk_1hz<=m_1hz;
new_clk_24M<=m_80k;
end if;
end process;
end art;
XSKZ:
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
library lpm;
use lpm.lpm_components.all;
entity XSKZ is
    port(kk:in std_logic_vector(2 downto 0);
        ru:in std_logic_vector(15 downto 0);
        clk24M:in std_logic;
        --The faster the display control clock, the better
        din7:in std_logic_vector(4 downto 0);
        dout7:out std_logic_vector(7 downto 0);
        sel:out std_logic_vector(7 downto 0));
end XSKZ;
architecture art of XSKZ is
--component
component data_change is
    port(kk:in std_logic_vector(2 downto 0);
        din0,din1,din2:in std_logic_vector(3 downto 0);
        din4:in std_logic_vector(2 downto 0);
        din3:in std_logic_vector(4 downto 0);
        dout0,dout1,dout2,dout3,dout4:out std_logic_vector(4 downto 0));
end component data_change;

```

```

component YMQ58 is
    port(a:in std_logic_vector(4 downto 0);
          Y:out std_logic_vector(7 downto 0));
end component YMQ58;

component quling is
    port(din0,din1,din2,din3,din4:in std_logic_vector(7 downto 0);
          dout0,dout1,dout2,dout3,dout4:out std_logic_vector(7 downto 0));
end component quling;

component display is
    port(clk256:in std_logic;
          din0,din1,din2,din3,din4,
          din7:in std_logic_vector(7 downto 0);
          dout:out std_logic_vector(7 downto 0);
          sel:out std_logic_vector(7 downto 0));
end component display;

--signal
signal s_quotient_10000_16:std_logic_vector(15 downto 0);
signal s_remain_10000_14:std_logic_vector(13 downto 0);
signal s_quotient_1000_14:std_logic_vector(13 downto 0);
signal s_remain_1000_10:std_logic_vector(9 downto 0);
signal s_quotient_100_10:std_logic_vector(9 downto 0);
signal s_remain_100_7:std_logic_vector(6 downto 0);
signal s_quotient_10_7:std_logic_vector(6 downto 0);
signal s_remain_10_4:std_logic_vector(3 downto 0);

signal s_data_change_dout0:std_logic_vector(4 downto 0);
signal s_data_change_dout1:std_logic_vector(4 downto 0);
signal s_data_change_dout2:std_logic_vector(4 downto 0);
signal s_data_change_dout3:std_logic_vector(4 downto 0);
signal s_data_change_dout4:std_logic_vector(4 downto 0);

signal s_YMQ58_Y0:std_logic_vector(7 downto 0);
signal s_YMQ58_Y1:std_logic_vector(7 downto 0);
signal s_YMQ58_Y2:std_logic_vector(7 downto 0);
signal s_YMQ58_Y3:std_logic_vector(7 downto 0);
signal s_YMQ58_Y4:std_logic_vector(7 downto 0);
signal s_YMQ58_Y7:std_logic_vector(7 downto 0);

signal s_quling_dout0:std_logic_vector(7 downto 0);
signal s_quling_dout1:std_logic_vector(7 downto 0);

```

```

signal s_quling_dout2:std_logic_vector(7 downto 0);
signal s_quling_dout3:std_logic_vector(7 downto 0);
signal s_quling_dout4:std_logic_vector(7 downto 0);

```

```

--

```

```

begin
  chufa10000 : LPM_DIVIDE
  generic map (
    lpm_drepresentation => "UNSIGNED",
    lpm_hint => "LPM_REMAINDERPOSITIVE=TRUE",
    lpm_nrepresentation => "UNSIGNED",
    lpm_type => "LPM_DIVIDE",
    lpm_widthhd => 14,--10000
    lpm_widthn => 16
  )
  port map (
    denom => "10011100010000",--10000
    numer => ru,
    quotient => s_quotient_10000_16,
    remain => s_remain_10000_14
  );

```

```

  chufa1000 : LPM_DIVIDE
  generic map (
    lpm_drepresentation => "UNSIGNED",
    lpm_hint => "LPM_REMAINDERPOSITIVE=TRUE",
    lpm_nrepresentation => "UNSIGNED",
    lpm_type => "LPM_DIVIDE",
    lpm_widthhd => 10,--1000
    lpm_widthn => 14
  )
  port map (
    denom => "1111101000",--1000
    numer => s_remain_10000_14,
    quotient => s_quotient_1000_14,
    remain => s_remain_1000_10
  );

```

```

  chufa100 : LPM_DIVIDE
  generic map (
    lpm_drepresentation => "UNSIGNED",
    lpm_hint => "LPM_REMAINDERPOSITIVE=TRUE",
    lpm_nrepresentation => "UNSIGNED",
    lpm_type => "LPM_DIVIDE",

```

```

    lpm_widthd => 7,--100
    lpm_widthn => 10
)
port map (
    denom => "1100100",--100
    numer => s_remain_1000_10,
    quotient => s_quotient_100_10,
    remain => s_remain_100_7
);

chufa10 : LPM_DIVIDE
generic map (
    lpm_drepresentation => "UNSIGNED",
    lpm_hint => "LPM_REMAINDERPOSITIVE=TRUE",
    lpm_nrepresentation => "UNSIGNED",
    lpm_type => "LPM_DIVIDE",
    lpm_widthd => 4,--10
    lpm_widthn => 7
)
port map (
    denom => "1010",--10
    numer => s_remain_100_7,
    quotient => s_quotient_10_7,
    remain => s_remain_10_4
);

u0:data_change
    port map(kk,s_remain_10_4,s_quotient_10_7(3 downto 0),--kk,din0,din1
        s_quotient_100_10(3 downto 0),s_quotient_10000_16(2 downto
0),--din2,din4
        s_quotient_1000_14(4 downto 0),--din3
        s_data_change_dout0,s_data_change_dout1,s_data_change_dout2,
        s_data_change_dout3,s_data_change_dout4);
u1:YMQ58 port map(s_data_change_dout0,s_YMQ58_Y0);
u2:YMQ58 port map(s_data_change_dout1,s_YMQ58_Y1);
u3:YMQ58 port map(s_data_change_dout2,s_YMQ58_Y2);
u4:YMQ58 port map(s_data_change_dout3,s_YMQ58_Y3);
u5:YMQ58 port map(s_data_change_dout4,s_YMQ58_Y4);
u6:YMQ58 port map(din7,s_YMQ58_Y7);

u7:quling port map(s_YMQ58_Y0,s_YMQ58_Y1,s_YMQ58_Y2,s_YMQ58_Y3,s_YMQ58_Y4,
    s_quling_dout0,s_quling_dout1,s_quling_dout2,
    s_quling_dout3,s_quling_dout4);

```

```

        u8:display port map(clk24M,s_quling_dout0,s_quling_dout1,s_quling_dout2,
                           s_quling_dout3,s_quling_dout4,s_YMQ58_Y7,
                           dout7,sel);
end art;

```

Xuanze

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity xuanze is
    port(k:in std_logic_vector(2 downto 0);
         din0,din1:in std_logic_vector(13 downto 0);--youxiaozhi
         din2:in std_logic_vector(17 downto 0);--youngong
         din3:in std_logic_vector(17 downto 0);--wugong
         din4:in std_logic_vector(17 downto 0);--gonglvinsu
         dout:out std_logic_vector(15 downto 0);
         dout7:out std_logic_vector(4 downto 0));

```

```

end xuanze;

```

```

architecture art of xuanze is

```

```

    begin
        process(k,din0,din1,din2,din3,din4)
        begin
            case k is
                when "000" =>dout<="00"&din0;
                           dout7<="00000";
                when "001" =>dout<="00"&din1;
                           dout7<="00001";
                when "010" =>dout<=din2(15 downto 0);
                           dout7<="00010";
                when "011" =>dout<=din3(15 downto 0);
                           dout7<="00011";
                when "100" =>dout<=din4(15 downto 0);
                           dout7<="00100";
                when others=>>null;
            end case;
        end process;
    end art;

```

五、 主要 VHDL 源程序仿真结果

xuanze

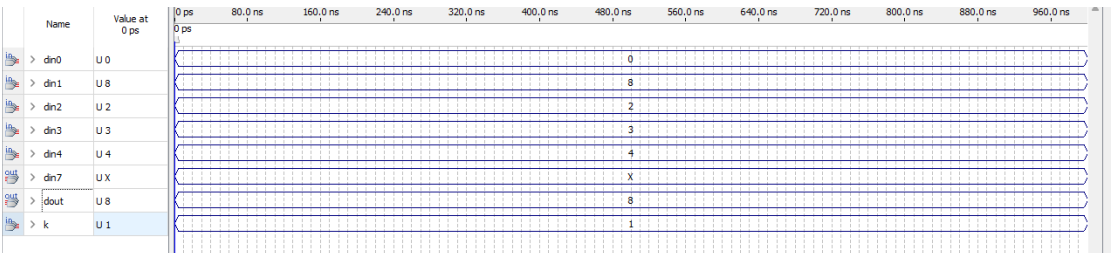


图 3

XSKZ

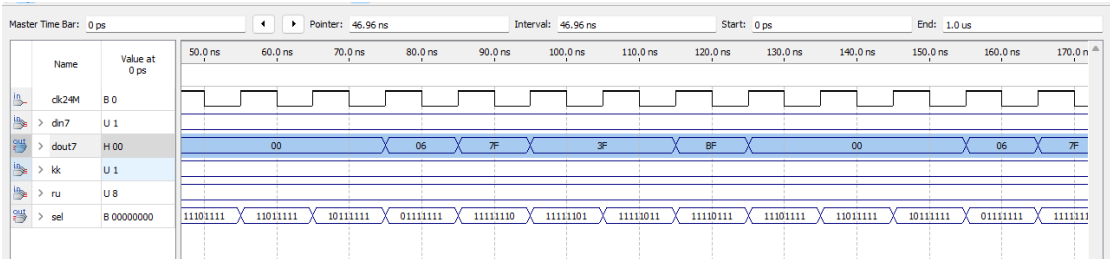


图 4

Youxiaoshi

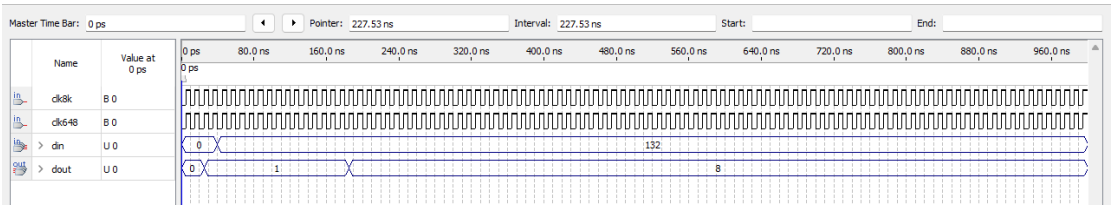


图 5

六、 硬件验证方案及验证结果

芯片管脚的锁定

工程编译和有关仿真都通过后，就可以将配置数据下载到应用系统进行验证。下载之前首先要对系统顶层模块进行引脚锁定保证锁定的引脚与实际的应用系统相吻合。

1)目标芯片的确认及闲置引脚的设定

管脚锁定前，先进行芯片的确定或修改，如图 4.43 所示。单击图 4.43 中的【Device and Pin Options...】按钮，在弹出的【Unused Pins】设置框中进行闲置引脚的设定，详见图 4.25 所示。对设计中未用到的器件引脚，有三种处理方式：输入引脚(呈高组态)、输出引脚(呈低电平)或输出引脚(输出不定状态)。通常情况下选择第一项，避免未用到的引脚对应用系统产生影响。

2)引脚锁定

本设计系统的顶层模块 TOP 拟选用 EP3C55484C8 芯片，根据需使用的 FDA 资验开发系统(板)的有关输入和输出的资源情况进行引脚锁定,一般应事先列出一个管脚锁定表，并将闲

置引脚设定为三态门状态。

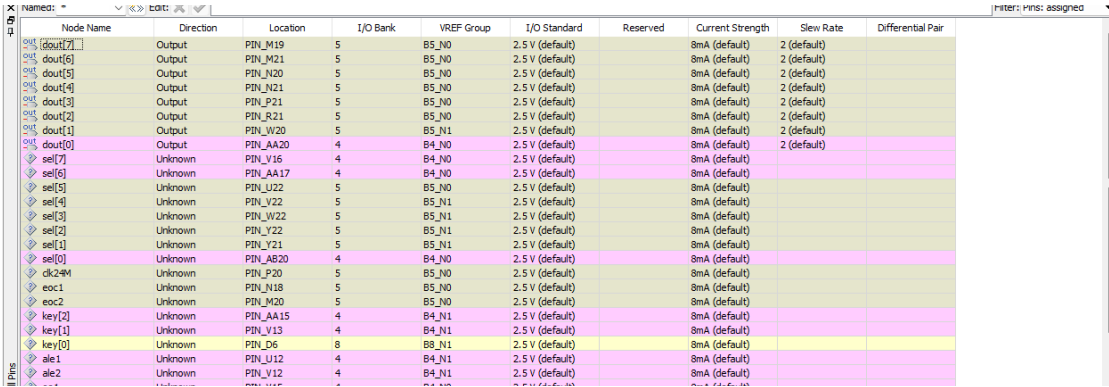
引脚的锁定方法有三种：一是使用引脚锁定窗口进行锁定；二是使用记事本或其他文本编辑工具直接编辑.qsf 文件进行引脚锁定；三是通过输入 TCL 脚本语言文件进行,本次实验使用第一种：

使用引脚锁定窗口进行锁定

打开【Assignments】菜单下的【Pins】命令，打开引脚锁定窗口，如图 4.44 所示。先用鼠标指到要锁定的端口信号名与【Location】栏交汇的地方，这时此处呈蓝色，然后双击对应的交汇处，在出现的下拉栏中选择对应端口信号名的器件引脚号(例如对应 ENA，选择 PIN_99)，引脚锁定后将下拉菜单复原，则系统自动保存该锁定。在如图 4.44 所示的窗口中，还能对引脚作进一步的设定，如在 Reserved 栏，可对某些空闲的 I/O 引脚的电气特性进行设置。

按前面提到的引脚信息添加锁定引脚，直到全部输入完毕。锁定引脚后必须再编译次本能将引脚锁定信息应用到最终的下载文件中，此后就可以将编译好的 SOF 文件下载到实验系统的 FPGA 中了。

管脚锁定图



Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair
dout[7]	Output	PIN_M19	5	B5_N0	2.5 V (default)		8mA (default)	2 (default)	
dout[6]	Output	PIN_M21	5	B5_N0	2.5 V (default)		8mA (default)	2 (default)	
dout[5]	Output	PIN_N20	5	B5_N0	2.5 V (default)		8mA (default)	2 (default)	
dout[4]	Output	PIN_N21	5	B5_N0	2.5 V (default)		8mA (default)	2 (default)	
dout[3]	Output	PIN_P21	5	B5_N0	2.5 V (default)		8mA (default)	2 (default)	
dout[2]	Output	PIN_R21	5	B5_N0	2.5 V (default)		8mA (default)	2 (default)	
dout[1]	Output	PIN_W20	5	B5_N1	2.5 V (default)		8mA (default)	2 (default)	
dout[0]	Output	PIN_AA20	4	B4_N0	2.5 V (default)		8mA (default)	2 (default)	
sel[7]	Unknown	PIN_V16	4	B4_N0	2.5 V (default)		8mA (default)		
sel[6]	Unknown	PIN_AA17	4	B4_N0	2.5 V (default)		8mA (default)		
sel[5]	Unknown	PIN_U22	5	B5_N0	2.5 V (default)		8mA (default)		
sel[4]	Unknown	PIN_W22	5	B5_N1	2.5 V (default)		8mA (default)		
sel[3]	Unknown	PIN_V22	5	B5_N1	2.5 V (default)		8mA (default)		
sel[2]	Unknown	PIN_Y22	5	B5_N1	2.5 V (default)		8mA (default)		
sel[1]	Unknown	PIN_Y21	5	B5_N1	2.5 V (default)		8mA (default)		
sel[0]	Unknown	PIN_AB20	4	B4_N0	2.5 V (default)		8mA (default)		
clk2M	Unknown	PIN_P20	5	B5_N0	2.5 V (default)		8mA (default)		
eoc1	Unknown	PIN_N18	5	B5_N0	2.5 V (default)		8mA (default)		
eoc2	Unknown	PIN_M20	5	B5_N0	2.5 V (default)		8mA (default)		
key[2]	Unknown	PIN_AA15	4	B4_N1	2.5 V (default)		8mA (default)		
key[1]	Unknown	PIN_V13	4	B4_N1	2.5 V (default)		8mA (default)		
key[0]	Unknown	PIN_D6	8	B8_N1	2.5 V (default)		8mA (default)		
ale1	Unknown	PIN_U12	4	B4_N1	2.5 V (default)		8mA (default)		
ale2	Unknown	PIN_Y12	4	B4_N1	2.5 V (default)		8mA (default)		
ale3	Unknown	PIN_V15	4	B4_N1	2.5 V (default)		8mA (default)		

图 6

以下是实验系统管脚图

发光二极管 LED8-LED1	CON1.9-CON1.2	T17、R16、Y17、W15、 W13、V15、V12、U12
数码管驱动端 8XSEG LH-LA	CON1.32-CON1.25	M19、M21、N20、N21、 P21、R21、W20、AA20
数码管公共端 8XSEG DS8-DS1	CON1.51-CON1.44	V16、AA17、U22、V22、 W22、Y22、Y21、AB20
按键 SW4-按键 SW1	CON1.61、 CON1.54、 CON1.34、 CON1.33、	V13、AA15、M20、N18
按键 SW8-按键 SW5	CON2.59、 CON2.60、 CON2.61、 CON2.62、	F8、E7、C8、D6

图 7

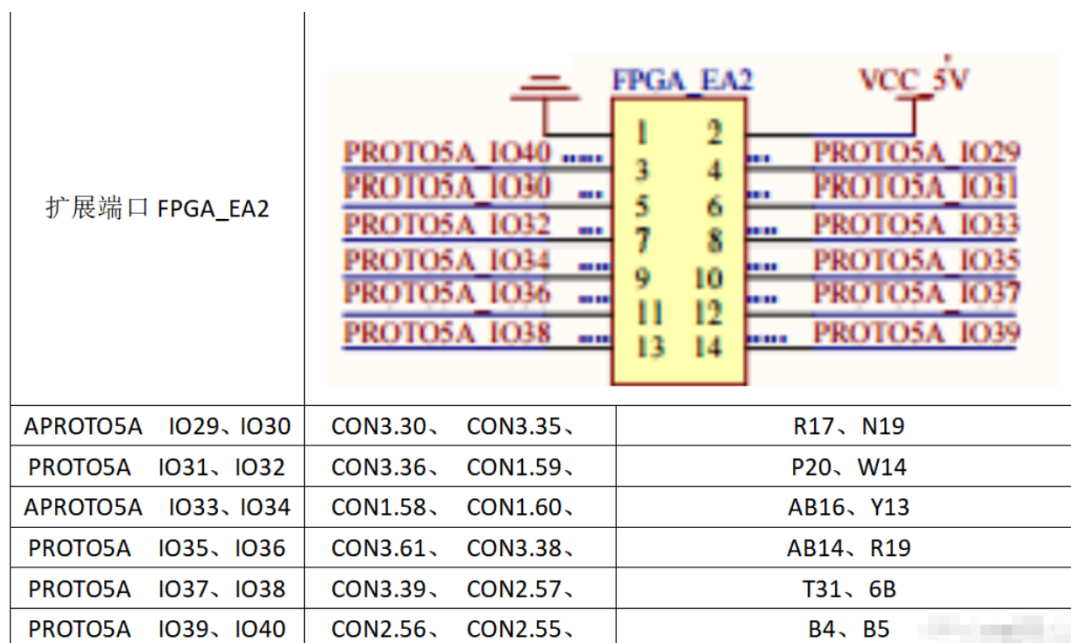


图 8

编程下载及验证

1) 编程下载硬件准备

先阅读有关 EDA 实验开发系统(板)手册了解 EDA 实验开发系统(板)方式。在断电的情况下将有关硬件设备进行正确的物理连接，经检查无误开发系统(板)的电源开关。

2) FPGA 的编程下载

连接好下载电缆，打开电源。在菜单【Tool】中选择【Programmer】，栏上的快捷键，可以打开编程下载窗口

在下载编程前需要选择下载接口方式。口中单击【Hardware Setup】，在打开的设置窗口根据实际情况进行设置【USB3.0】，双击鼠标后，关闭该窗口。

在【Mode】栏中有四种编程模式可以选择：JTAG、PassSerial 和 In-Socket。为了直接对 FPGA 进行配置，在编程窗口的编程模式【JTAG】，并选中下载文件右侧的第一个小方框 Program/Configure。核对名，如果此文件没有出错或者有错，单击左侧的【Add File】按钮，找 TOP.sof。单击【Start】按钮，即进入对目标器件 FPGA 的配置下载操显示为 100%时，编译成功，可以观察实验面板，进行硬件测试验证。

下载完成后观察实验现象

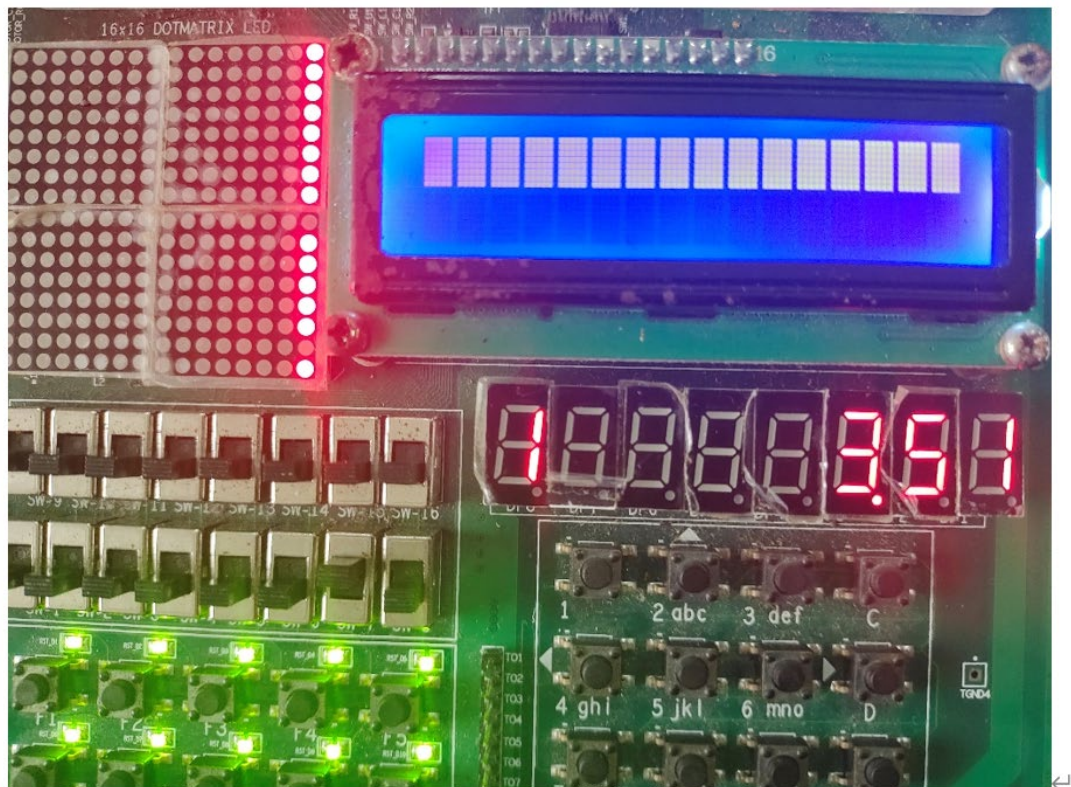


图 9

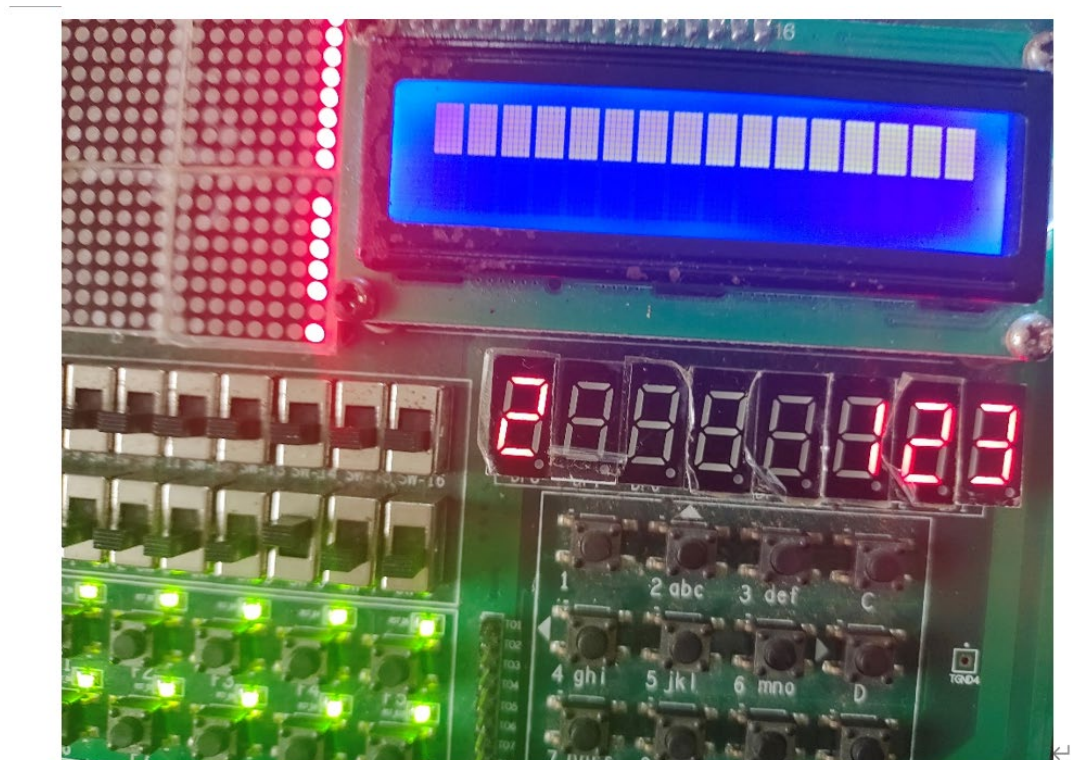


图 10

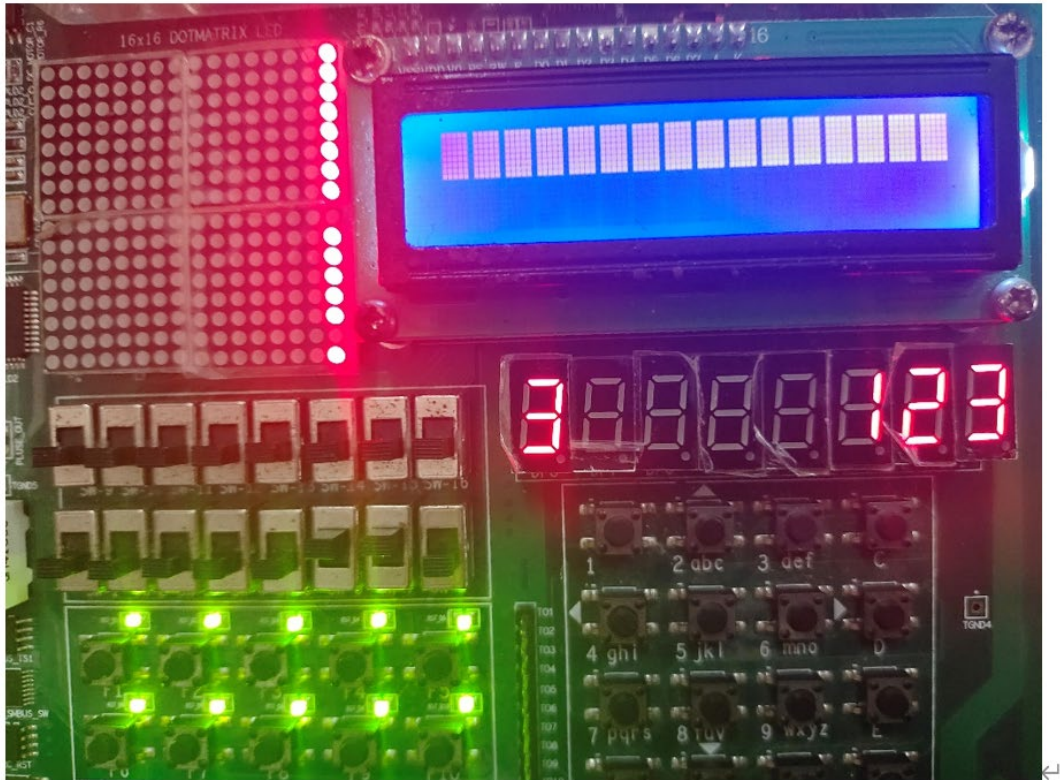


图 11

可以发现硬件现象和仿真基本一致, 对应的模块都已显示

七、 研究性学习收获体会与研究结论

研究性学习收获体会:

- (1) 了解了数字信号处理的基本理论和方法, 包括采样、滤波、FFT 变换等。
- (2) 学习了 FPGA 的基本原理和编程方法, 掌握了 FPGA 在数字信号处理中的应用。
- (3) 深入了解了工频多用表的原理和应用, 包括有功功率、无功功率、功率因数等参数的计算方法和意义。
- (4) 了解了硬件验证和仿真验证的基本原理和方法, 以及两种验证方法的优缺点。

研究结论:

- (1) 基于 FPGA 的数字式工频多用表可以实现对输入信号的频率、幅值、相位和有效值等参数的准确计算和显示, 具有较高的计算精度和信号处理能力。
- (2) 硬件验证和仿真验证都可以用于验证数字式工频多用表的设计方案, 但仿真验证具有更高的灵活性和可重复性, 而硬件验证更接近实际应用场景。
- (3) 数字信号处理方法和 FPGA 技术在工频多用表中的应用, 可以提高工频多用表的计算精度和信号处理能力, 为工业生产和科学研究提供了更为准确和可靠的数据。

在本次基于 FPGA 的数字式工频多用表的研究中, 我深入了解了数字信号处理的基本理论和方法, 包括采样、滤波、FFT 变换等。同时, 我学习了 FPGA 的基本原理和编程方法, 掌握了 FPGA 在数字信号处理中的应用。通过基于 FPGA 的数字式工频多用表的设计和仿真验证, 我不仅对工频多用表的原理和应用有了更深入的认识, 而且也更加熟练地掌握了数字信号处理和 FPGA 技术在工业生产和科学研究中的应用。

在本次研究中，我发现基于 **FPGA** 的数字式工频多用表可以实现对输入信号的频率、幅值、相位和有效值等参数的准确计算和显示，具有较高的计算精度和信号处理能力。同时，我也了解到硬件验证和仿真验证都可以用于验证数字式工频多用表的设计方案，但仿真验证具有更高的灵活性和可重复性，而硬件验证更接近实际应用场景。数字信号处理方法和 **FPGA** 技术在工频多用表中的应用，可以提高工频多用表的计算精度和信号处理能力，为工业生产和科学研究提供了更为准确和可靠的数据。

综上所述，通过本次研究，我收获了丰富的数字信号处理和 **FPGA** 技术的知识，同时也对工频多用表的原理和应用有了更深入的认识。在今后的学习和工作中，我将更加注重理论和实践的结合，不断提高自己的技术水平和实践能力