**2020年新工科联盟-Xilinx暑期学校团队项目设计文档**

**设计文稿提交格式**

**(Project Paper Submission Template)**

|  |  |
| --- | --- |
| **作品名称** | 虚拟示波器 |
| **板卡型号** | *Xc7s15ftgb196-1* |
| **所在班级** | 东南大学 |
| **成员姓名、学号、学校** | 施雨潇 06017204 东南大学  甄涵文 06117117 东南大学 |
| **Github链接** | https://github.com/shialice884/virtual-oscilloscope-based-on-sea-board.git |

**第一部分**

设计概述 /Design Introduction

本设计为仪器仪表-虚拟示波器，虚拟示波器是利用高性能的模块化硬件，结合高效灵活的软件来完成各种测试、测量和自动化的应用。虚拟示波器主要包括信号采集与控制，数据分析和处理，测量结果的显示三大部分。信号采集与控制是由计算机和仪器硬件组成硬件平台，实现对信号的采集、测量、转换与控制;数据分析与处理则是表现在虚拟示波器充分利用计算机的存储、运算功能，并通过软件实现对输入数据信号的分析与处理;虚拟示波器测量结果的显示是利用计算机的资源，如显示器、存储器等，把测量结果进行多种方式的表达与输出，也可以利用计算机进行数据的传输和利用。

虚拟示波器作为一个完整的软硬件系统，它和传统的示波器相比，比传统示波器价格要便宜，并具有波形触发、存储、显示、测量、波形数据分析处理等独特优点。

本设计与一般虚拟模拟器相比，主要区别以及特点体现在采用蓝牙模块进行数据的传输，这就使得虚拟示波器与计算机接口更加方便，通信速度更高。也使得虚拟示波器的扩展性更加良好，可以兼容绝大部分硬件，降低了示波器的使用条件，只需一台电脑即可观察波形，更好的满足人们对示波器的使用要求。

因其具有硬件条件要求不高，扩展性强，兼容性良好，价格低廉等特点，可以被广泛应用在学生实验室以及一些电子爱好者自己的实验室里面。

小组成员贡献比1:1

作品展示见视频/报告第三部分

**第二部分**

系统组成及功能说明 /System Construction & Function Description

Esp32通信模块

频率计算，最大值计算模块

ADC采样模拟信号模块

系统框图：ADC采样模拟信号模块——————频率计算/最大值计算模块————————esp32通信模块

我们ADC实验目的是让模拟信号显示出来，这里显示既包括在实际屏幕上的显示，同时也包括在虚拟电脑屏幕上的显示。对于前者在屏幕上的显示，我们主要参考了实验13，分析示波器的功能需求，主要涉及在于ADC模拟信号转成数字信号，数字信号存储模块以及如何让频率以及幅度显示模块，存储模块我们主要用了RAMIP核，频率模块包含了转换成方波计算周期从而表现频率，幅度显示模块我们通过最大值与最小值的差距计算得出。对于后者模拟示波器，我们主要借助了Arduino串口绘图器，着重进行了esp32通信模块的编写。具体代码分析见下。

ADC数据读取模块（将模拟信号转变为数字信号，再读取数据，数据存储在RAM中）

module Driver\_ADC(

input clk\_100MHz,

input clk\_system, （时钟读取信号）

input Rst,

input[7:0]ADC\_Data, （ADC采样后的数据）

input[17:0]Read\_Addr, （读取信号地址）

input[7:0]Trigger\_Gate, （触发阈值）

output[17:0]Period,

output clk\_ADC,

output ADC\_En,

output [7:0]ADC\_Data\_Out

);

parameter Sampling\_Num=38400;

reg [15:0]Addr\_Cnt=0;

reg[15:0]Addr\_Read\_Real=0;

assign ADC\_En=~Rst;

Clk\_Division Clk\_Division\_ADC(

.clk\_100MHz(clk\_100MHz),

.clk\_mode(200),

.clk\_out(clk\_ADC)

);

always@(posedge clk\_ADC or negedge Rst)

begin

**if**(!Rst)

Addr\_Cnt<=0;

**else if**(Addr\_Cnt==Sampling\_Num-1)

Addr\_Cnt<=0;

**else**

Addr\_Cnt<=Addr\_Cnt+1;

end

Freq\_Cal Freq\_Cal0(

添加 clk\_wiz\_0（分频器）的 IP 核，设置参数如下：

（1）时钟产生方式：MMCM。

（2）输入时钟：100MHz。

（3）输出时钟 1：148.5MHz。

（4）输出时钟 2：100MHz。

.clk\_100MHz(clk\_100MHz),

.Rst(Rst),

.ADC\_Data(ADC\_Data),

.F\_Gate(Trigger\_Gate),

.Period(Period)

);

（波形信号存储到RAM中）

（1）IP 核名称为：Wave\_Ram。

（2）接口类型为:Native。

（3）存储类型为：普通双端口 RAM

（4）数据位宽：8 位（采样数据匹配）

（5）数据长度：38400

（6）操作模式：写优先。

（7）端口使能:默认有效。

Wave\_Ram Sampling\_38400\_0(

.clka(clk\_ADC),

.wea(Rst),

.addra(Addr\_Cnt),

.dina(ADC\_Data),

.clkb(clk\_system),

.addrb(Read\_Addr),

.doutb(ADC\_Data\_Out)

);

Endmodule

计算频率的模块：

module Freq\_Cal(

input clk\_100MHz,

input Rst,

input [7:0]ADC\_Data,（模拟信号已经转换为数字信号，输入数字信号）

input [7:0]F\_Gate, （设定一个阈值）

output reg[20:0]Period=1

);

parameter Measure\_Num=5;

wire Signal\_Pulse=ADC\_Data>F\_Gate?1:0; （规定大于阈值记为1，小于阈值记为0）

reg [31:0]Measure\_Cnt=0;

reg [19:0]Measure\_Num\_Cnt=0; （寄存脉冲计数）

reg [31:0]Measure\_Delta\_Cnt=0; （周围干扰计数）

reg Measure\_Delta\_Clear=0;

reg Delta\_Clear\_Flag=0;

always@(posedge clk\_100MHz or negedge Rst)

begin

**if**(!Rst) （低电平）

begin

Measure\_Delta\_Cnt<=0;

Delta\_Clear\_Flag<=0;

end （开始先清零）

**else if**(Measure\_Delta\_Clear)

begin

Measure\_Delta\_Cnt<=0;

Delta\_Clear\_Flag<=1;

end

**else**

begin

Measure\_Delta\_Cnt<=Measure\_Delta\_Cnt+1;

Delta\_Clear\_Flag<=0;

end

end

下面为开始脉冲计数模块

always@(posedge Signal\_Pulse or negedge Rst or posedge Delta\_Clear\_Flag)

begin

**if**(!Rst)

begin

Measure\_Num\_Cnt<=0;

Measure\_Delta\_Clear<=0;

Measure\_Cnt<=0;

Period<=0;

end

**else if**(Delta\_Clear\_Flag)

Measure\_Delta\_Clear<=0;

**else**

begin

**if**(Measure\_Num\_Cnt==Measure\_Num-1)

begin

**if**(Measure\_Cnt<200)

Period<=1;

**else if**(Measure\_Cnt>1000000)

Period<=5000;

**else**

Period<=Measure\_Cnt/200;

Measure\_Num\_Cnt<=0;

Measure\_Delta\_Clear<=1;

Measure\_Cnt<=0;

end

**else**

begin

Measure\_Num\_Cnt<=Measure\_Num\_Cnt+1;

Measure\_Cnt<=Measure\_Cnt+Measure\_Delta\_Cnt;

end

end

end

endmodule

显示模块：根据坐标点，来显示读到的波形。

module Wave\_Generator(

input RGB\_VDE,

input [17:0]Offset,

input [11:0]Set\_X,

input [11:0]Set\_Y,

input [7:0]ADC\_Data\_Out,

output reg[17:0]Read\_Addr,

output reg[23:0]RGB\_Data=0 //RBG

);

**always**@(\*)

begin

Read\_Addr=Set\_X+Offset;

**if**(Set\_Y>=283&&Set\_Y<797)

**if**(Set\_Y==ADC\_Data\_Out+284||Set\_Y==ADC\_Data\_Out+283||Set\_Y==ADC\_Data\_Out+285)

RGB\_Data<=24'hff00ff; （符合计数值，y坐标显示具体计数值）

else

RGB\_Data<=24'h000000; （不符合计数值，不显示）

**else**

RGB\_Data<=24'h000000;

end

endmodule

整体综合模块

module ADC\_Demo(

input clk\_100MHz,

input [7:0]ADC\_Data,

output clk\_ADC,

output ADC\_En,

output TMDS\_Tx\_Clk\_N,

output TMDS\_Tx\_Clk\_P,

output [2:0]TMDS\_Tx\_Data\_N,

output [2:0]TMDS\_Tx\_Data\_P

);

wire clk\_100MHz\_system;

wire clk\_system;

wire [23:0]RGB\_Data;

wire [23:0]RGB\_In;

wire RGB\_HSync;

wire RGB\_VSync;

wire RGB\_VDE;

wire [11:0]Set\_X;

wire [11:0]Set\_Y;

wire[17:0]Read\_Addr;

wire[7:0]ADC\_Data\_Out;

wire [20:0]Offset;

clk\_wiz\_0 clk\_10(.clk\_out1(clk\_system),.clk\_out2(clk\_100MHz\_system),.clk\_in1(clk\_100MHz));

rgb2dvi\_0 rgb2dvi(

.TMDS\_Clk\_p(TMDS\_Tx\_Clk\_P),

.TMDS\_Clk\_n(TMDS\_Tx\_Clk\_N),

.TMDS\_Data\_p(TMDS\_Tx\_Data\_P),

.TMDS\_Data\_n(TMDS\_Tx\_Data\_N),

.aRst\_n(1),

.vid\_pData(RGB\_Data),

.vid\_pVDE(RGB\_VDE),

.vid\_pHSync(RGB\_HSync),

.vid\_pVSync(RGB\_VSync),

.PixelClk(clk\_system));

（HDMI接口显示模块）

Driver\_HDMI\_0 Driver\_HDMI0(

.clk(clk\_system),

.Rst(1),

.Video\_Mode(0),

.RGB\_In(RGB\_In),

.RGB\_Data(RGB\_Data),

.RGB\_HSync(RGB\_HSync),

.RGB\_VSync(RGB\_VSync),

.RGB\_VDE(RGB\_VDE),

.Set\_X(Set\_X),

.Set\_Y(Set\_Y)

);

Driver\_ADC Driver\_ADC0(

.clk\_100MHz(clk\_100MHz\_system),

.clk\_system(clk\_system),

.Rst(1),

.ADC\_Data(ADC\_Data),

.Read\_Addr(Read\_Addr),

.Trigger\_Gate(128),

.Period(Offset),

.clk\_ADC(clk\_ADC),

.ADC\_En(ADC\_En),

.ADC\_Data\_Out(ADC\_Data\_Out)

);

（波形显示模块）

Wave\_Generator Wave\_Generator0(

.RGB\_VDE(RGB\_VDE),

.Offset(Offset),

.Set\_X(Set\_X),

.Set\_Y(Set\_Y),

.ADC\_Data\_Out(ADC\_Data\_Out),

.Read\_Addr(Read\_Addr),

.RGB\_Data(RGB\_In)

);

endmodule

Esp32通信模块

实验中在 ESP32 中进行 QSPI 主机的配置，配置如下：

使用模式 0 （CPOL=0，CPHA=0）：在总 QSPI 线空闲时间，qspi\_clk 为低电平，qspi\_cs 为高电平数据在上升沿被读取，在下降沿写入。

工作模式采用半双工模式：以 ESP32 为 QSPI 主机端，FPGA 为 QSPI 从机端，进行 QSPI 的通信。

指令配置：使用 8bit 指令配置，单线传输模式，每个通信时钟周期传输 1bit 指令。

地址配置：使用 32bit 地址配置，四线传输模式，每个通信时钟周期传输 4bit 地址。

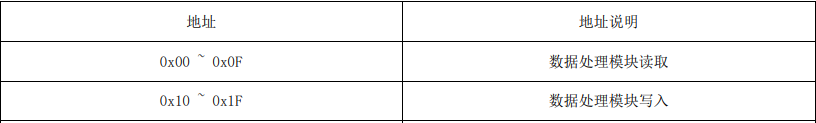
空周期配置：空周期用于 QSPI 交互模块地址的建立，配置为 4 个空周期。

数据配置：使用 8-bit 数据配置，四线传输模式，每个通信时钟传输 4-bit 数据，允许多字节数据传输；每传输 1 字节， RAM 的地址端自动加 1，直到 CS 拉高（QSPI 数据传输完成）为止。

存储模块 RAM 的配置：

本设计的存储模块只配置了一个双口 RAM，分别连接于 ESP32（间接连接）和 FPGA，可供 ESP32 的 QSPI 指令读写，也可用于 FPGA 内部模块的读写。

存储地址分配：



Arduino 端 QSPI 实现

在 Arduino 端对 ESP32 的 QSPI 主机进行配置和调试，QSPI 接口不变。

Arduino 端加载头文件 sea\_esp32\_qspi.h 后，使用 SeaTrans 进行 QSPI 的交互。SeaTrans 是 sea\_esp32\_qspi 的实 例，使用的 QSPI 控制器和 QSPI 的 IO 与 SEA 板设计一致；读写函数声明如下：

（1）初始化 SeaTrans:SeaTrans .begin ( )

（2）释放 QSPI 总线：SeaTrans .end ( )

（4）将 CS 拉低，然后依次进行指令（读指令设置为 0x）、地址、空周期、数据进行 QSPI 传输；将 FPGA 中的存储模块读出 数据到 ESP32 的接收缓存区中，其中接收数据长度最多为 32 字节。 SeaTrans .read ( 地址,接收变量,接收字节长度 )

对 FPGA 内部 RAM 地址 0x00 ~ 0x01 进行数据的读取

代码如下：

#include "sea\_esp32\_qspi.h"

void setup()

{

uint8\_t data2[8] = {0, 0};

Serial.begin(115200);

SeaTrans.begin();

SeaTrans.read(0, data2, 2);

Serial.printf("%d %d\r\n",data2[0],data2[1]); }

void loop() { }

本模块首先在 ESP32 上使用 Micropython 实现 QSPI 接口，利用Arduino IDE 完成 ESP32 QSPI 接口调用，在 FPGA 侧使用 Verilog HDL 语言完成 QSPI 从机的电路设计，Arduino 侧通过 QSPI 接口，进行读写 FPGA 内部 RAM 的 0x00 ~ 0x01 数据。FPGA侧，将ADC模块得到的数据存放在0x10 ~ 0x1F 的地址中。利用esp32通过QSPI通信协议读取FPGA上存储的波形信号并发送给上位显示模块进行显示。

Arduino试验版块（最基本的电脑验证模块）

int temp0;

void setup() {

Serial.begin(9600);

pinMode(A0,INPUT);// put your setup code here, to run once:

}

void loop() {

temp0=analogRead(A0);

Serial.println(temp0);

delay(0.1);// put your main code here, to run repeatedly:

}

**第三部分**

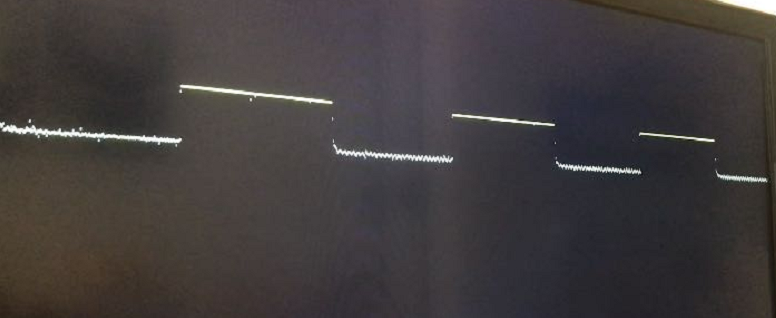
完成情况及性能参数 /Final Design & Performance Parameters

完成情况：分模块完成作品（已实现的功能）：

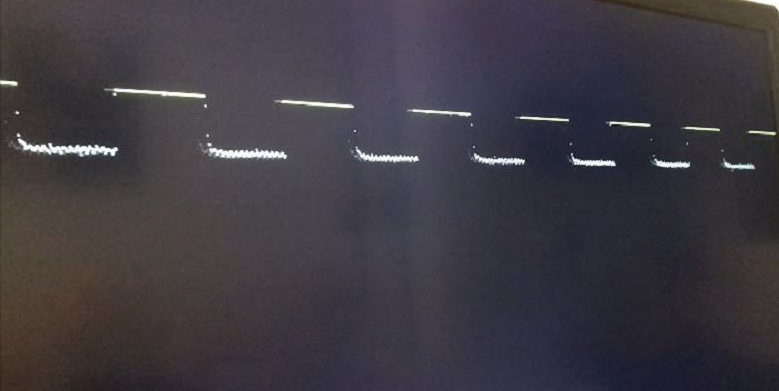
经过我们五天的设计，功能基本实现，能够实现信号发生器发生模拟信号，通过ADC转换成数字信号，通过minihdml在屏幕显示。信号波形也能通过电脑虚拟验证。但对于esp32通信模块我们还是没有能够很好的完全实现功能。具体实现功能见我们拍摄的验证视频。

1. 屏幕显示模块

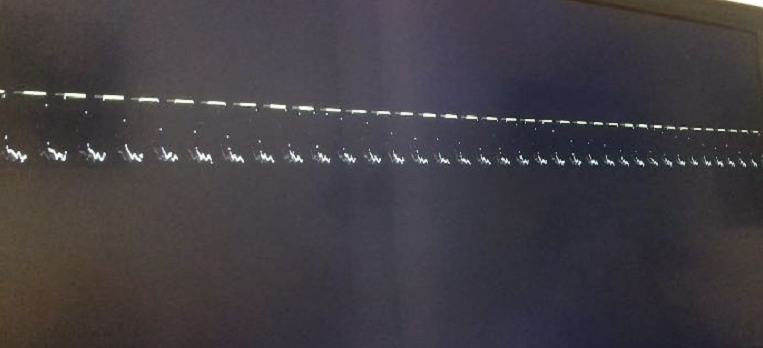
1000Hz



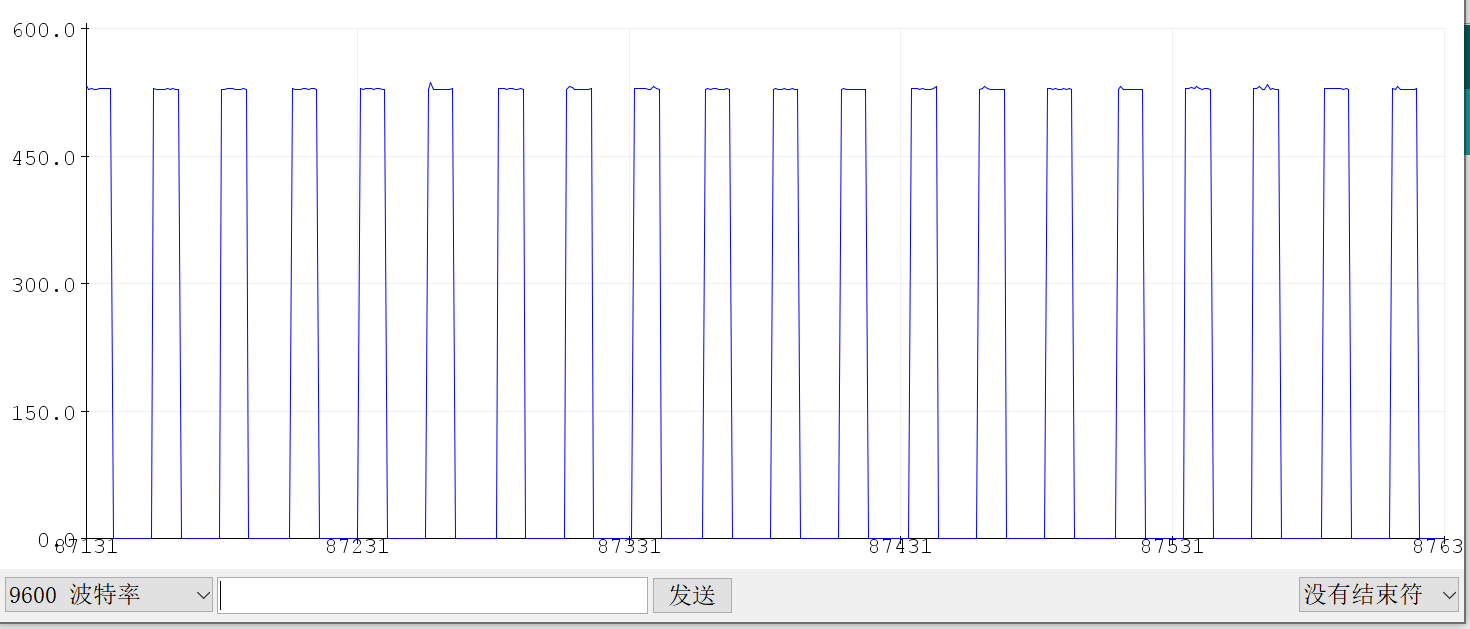
2000Hz



10000hz



arduino串口绘图（通过串口绘图器能够显示在屏幕上。）



**第四部分**

总结 /Conclusions

个人总结（甄涵文）：

本设计基本完成了蓝牙通信的虚拟示波器设计，本次设计中，我了解到了虚拟仪器的发展现状和前景，为以后的发展提供了更多的方向，而本次设计中用到的FPGA以及esp32等模块的使用，也是近几年以及未来几年的热门技术，并且越来越多的应用到实际的电子产品中。在以后的学习中，我会基于市场重新审视每一个需求，选择最优技术加以应用。

个人总结（施雨潇）

通过本次模拟示波器的制作，我将之前Xilinx课程上所学习的知识加以应用，并更好的了解了FPGA的设计和制作。在之前我只初步接触到了基于basys的设计，而这次是将fpga和esp32结合在一起，所以也能更好的发挥esp32的通信功能。在项目的完成过程中我们能够发现问题，比如ip核的升级更新，ip核的封装等从而更好的设计和解决问题。在本次项目的完成过程中，我也是第一次学习arduino编程语言，因为是初次学习，所以编程语言掌握并不是非常理想，这也是在结束这次Xilinx暑期学校，我仍然需要学习的地方。同时在做项目的过程中，我也不断反思，曾经我对于FPGA的应用认识比较片面，通过这次学习，我知道了其和esp32结合能更好发挥在物联网领域的作用。我相信本次的项目制作只是简单的一个开始，之后要学习的实践的依然很多，我会不断学习，更好的掌握FPGA在现代技术中的应用。