

I2C 接口 EEPROM 实验

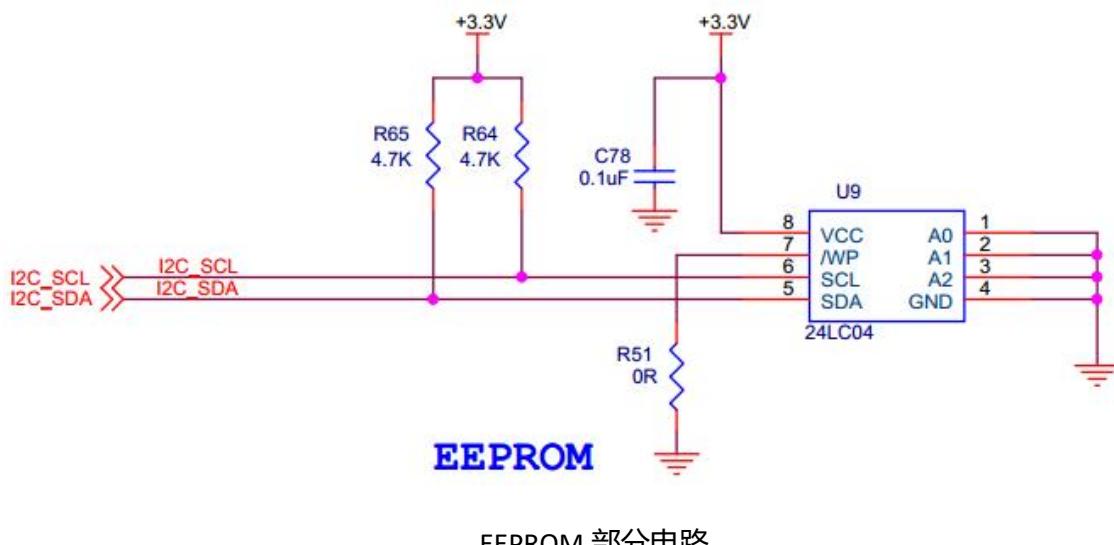
1 实验简介

本实验通过使用开源软件 opencores 上的 I2C master 控制器去控制 I2C 接口的 EEPROM 读写，练习如何有效的使用开源代码提升开发效率。

2 实验原理

2.1 硬件电路

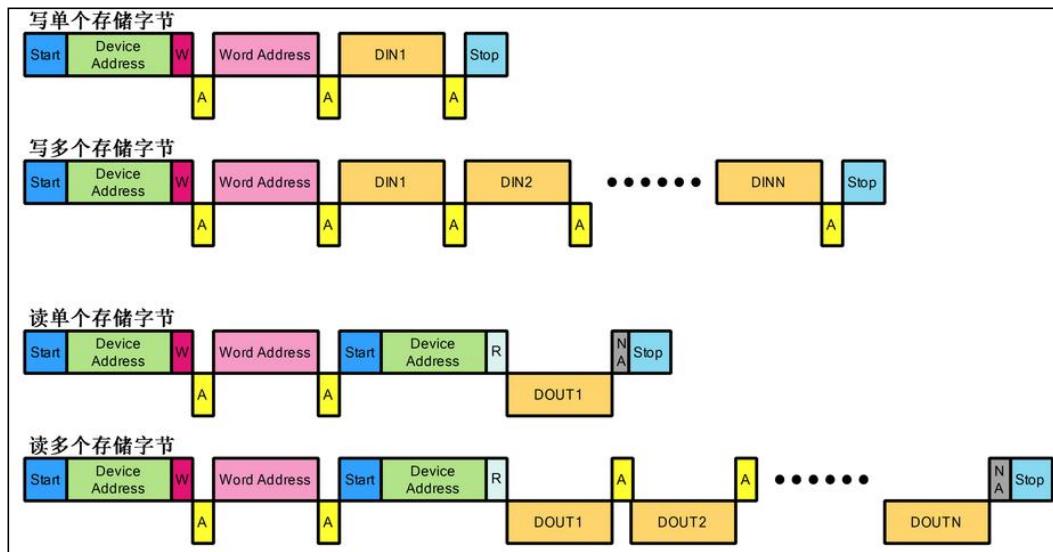
在开发板上，FPGA 芯片通过 I2C 总线连接 EEPROM 24LC04, I2C 的两根总线各上拉一个 4.7K 的电阻到 3.3V，所以当总线上没有输出时会被拉高，24LC04 的写保护没有使能，不然 FPGA 会无法写入数据。因为在电路上 A0~A2 都为低，所以 24LC04 的设备地址为 0xA0。



2.2 I2C 的总线协议和时序

I2C 标准速率为 100kbit/s，快速模式 400kbit/s，支持多机通讯，支持多主控模块，但同一时刻只允许有一个主控。由数据线 SDA 和时钟 SCL 构成串行总线；每个电路和模块都有唯一的地址。

在这里以 AT24C04 为例说明 I2C 读写的基本操作和时序，I2C 设备的操作可分为写单个存储字节，写多个存储字节，读单个存储字节和读多个存储字节。各个操作如下图所示。



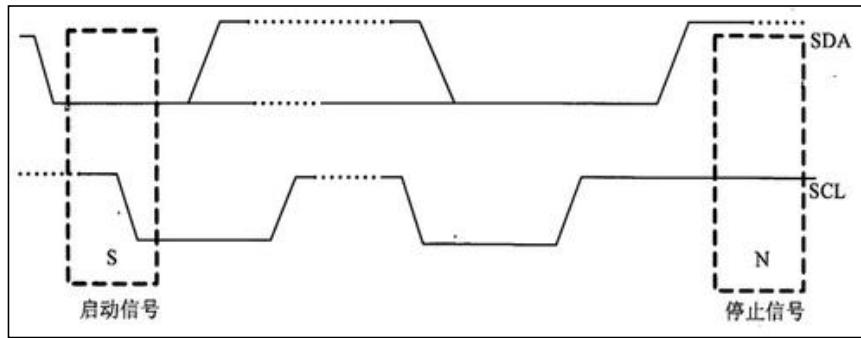
下面对 I2C 总线通信过程中出现的几种信号状态和时序进行分析。

①总线空闲状态

I2C 总线总线的 SDA 和 SCL 两条信号线同时处于高电平时，规定为总线的空闲状态。此时各个器件的输出级场效应管均处在截止状态，即释放总线，由两条信号线各自的上拉电阻把电平拉高。

②启动信号(Start)

在时钟线 SCL 保持高电平期间，数据线 SDA 上的电平被拉低（即负跳变），定义为 I2C 总线总线的启动信号，它标志着一次数据传输的开始。启动信号是由主控器主动建立的，在建立该信号之前 I2C 总线必须处于空闲状态，如下图所示。



③停止信号(Stop)

在时钟线 SCL 保持高电平期间，数据线 SDA 被释放，使得 SDA 返回高电平（即正跳变），称为 I2C 总线的停止信号，它标志着一次数据传输的终止。停止信号也是由主控器主动建立的，建立该信号之后，I2C 总线将返回空闲状态。

④数据位传送

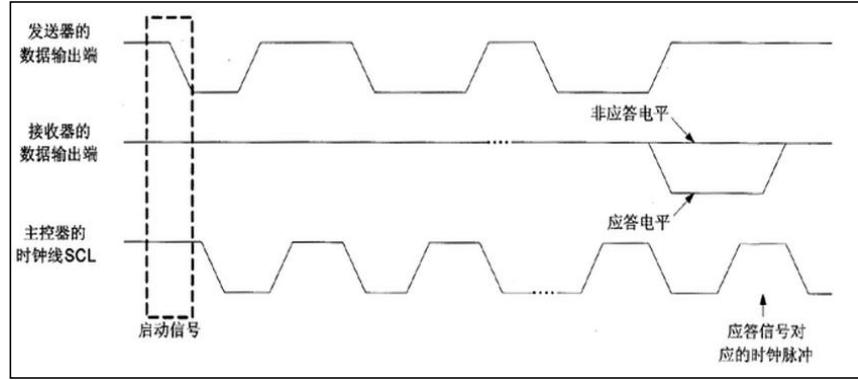
在 I2C 总线上传送的每一位数据都有一个时钟脉冲相对应（或同步控制），即在 SCL 串行时钟的配合下，在 SDA 上逐位地串行传送每一位数据。进行数据传送时，在 SCL 呈现高电平期间，SDA 上的电平必须保持稳定，低电平为数据 0，高电平为数据 1。只有在 SCL 为低电平期间，才允许 SDA 上的电平改变状态。

⑤应答信号 (ACK 和 NACK)

I2C 总线上的所有数据都是以 8 位字节传送的，发送器每发送一个字节，就在时钟脉冲 9 期间释放数据线，由接收器反馈一个应答信号。应答信号为低电平时，规定为有效应答位（ACK 简称应答位），表示接收器已经成功地接收了该字节；

应答信号为高电平时，规定为非应答位（NACK），一般表示接收器接收该字节没有成功。对于反馈有效应答位 ACK 的要求是，接收器在第 9 个时钟脉冲之前的低电平期间将 SDA 线拉低，并且确保在该时钟的高电平期间为稳定的低电平。

如果接收器是主控器，则在它收到最后一个字节后，发送一个 NACK 信号，以通知被控发送器结束数据发送，并释放 SDA 线，以便主控接收器发送一个停止信号。



3 程序设计

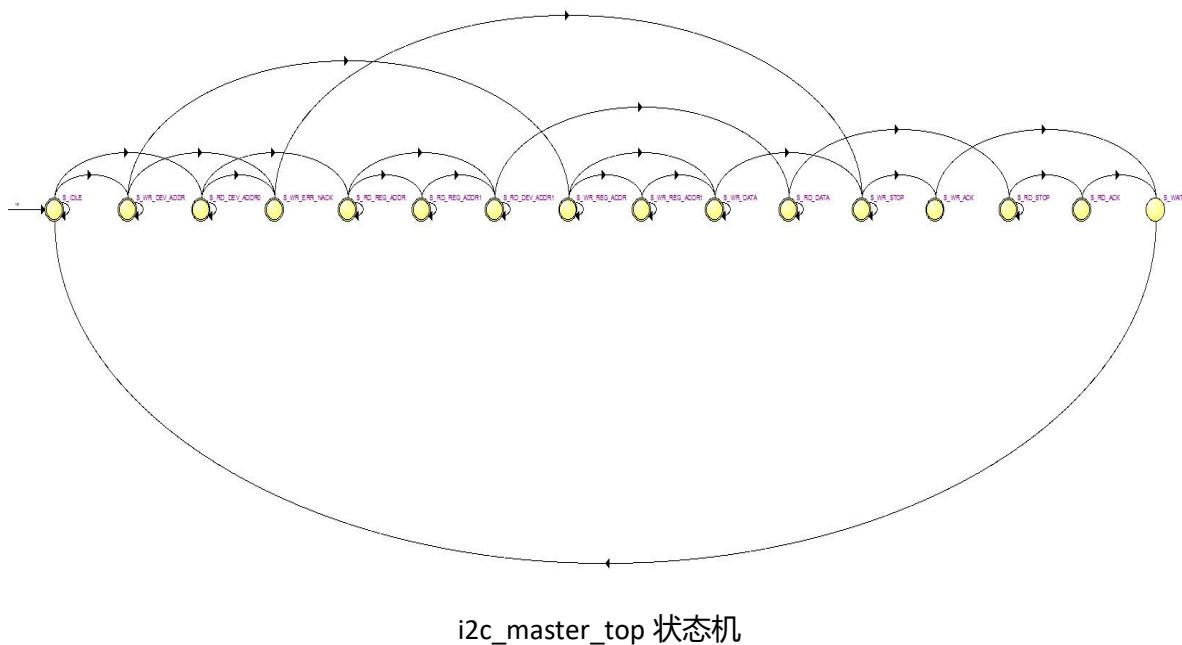
I2C 时序虽然简单，但是写的不好也会出现很多问题，在开源网站 <http://opencores.org/> 上我们可以找到很多非常好的代码，这些代码大部分都提供详细的文档和仿真。俗话说，他山之石，可以攻玉，恰当的使用开源代码，不光能提升我们的开发效率，也能学习别人的开发思路。由于代码大部分都是经过很长时间反复修改，反复精炼后的，所以有些代码理解起来可能比较困难，在不能很好的理解别人代码的时候，最好的办法就是仿真。

Gamepads	Stats		GPL	
General-Purpose I/O (GPIO) Core	Stats	wbc		
GPIB (IEEE-488) controller	Stats		GPL	
Hardware Assisted IEEE 1588 IP Core	Stats	wbc	LGPL	
HDB3/B3ZS Encoder+Decoder	Stats		BSD	
HDLC controller	Stats	wbc		
HyperTransport Tunnel	Stats			
I2C controller core	Stats	wbc	OCCP	BSD B.3
I2C Master Slave Core	Stats		BSD	
I2C master/slave Core	Stats	wbc		
I2C Multiple Bus Controller	Stats	wbc		
I2C Repeater	Stats		LGPL	
I2C Slave	Stats		GPL	
I2C Traffic Logger	Stats			
i2c_gpio	Stats		LGPL	
i2c_to_wb	Stats	wbc	LGPL	
I2S Interface	Stats	wbc	GPL	
I2S to Parallel ADC/DAC controller	Stats		GPL	
I2S to Parallel Interface	Stats		GPL	
I2S to WishBone	Stats		LGPL	

从 IP core 文档得知，i2c_master_byte_ctrl 模块主要完成一个字节的读写，我们只需要按照 I2C 读写的要求，完成设备地址、寄存器地址、数据等读写即可。

i2c_master_top 模块是对 i2c_master_byte_ctrl 模块的再次封装，完成一个寄存器的读写，由于不同的设备寄存器可能是 8bit，也可能是 16bit，这里 i2c_addr_2byte 信号来控制寄存器地址是 8 位还是 16 位。

i2c_master_top 模块状态机，如果是写寄存器操作，先写一个字节设备地址（写操作），再写 1 个字节或 2 个字节的寄存器地址，再写一个字节的数据；如果是读操作，先写一个字节的设备地址（写操作），再写 1 个字节或 2 字节的寄存器地址，完成地址的写入，再次写设备地址（读操作），然后读取一个字节的数据。不管怎么说，程序设计都是要满足芯片时序要求的，所以在阅读程序之前最好先把芯片的数据手册仔细阅读一遍。



信号名称	方向	说明
clk	in	时钟输入
rst	in	异步复位输入，高复位
clk_div_cnt	in	I2C 时钟分频因子，等于系统时钟频率 / (5 * I2C 时钟频率) - 1。例如 50Mhz 系统时钟，100Khz 的 I2C，配置为 99，400Khz 的 I2C，配置为 24。
scl_pad_i	in	I2C 时钟数据输入，本实验可忽略
scl_pad_o	out	I2C 时钟输出
scl_padoen_o	out	I2C 时钟输出使能，低有效，I2C 外部有上拉电阻，

		如果输出高阻态，则会被拉到高电平，在本实验中，高电平输出时输出高阻
sda_pad_i	in	I2C 数据输入
sda_pad_o	out	I2C 数据输出
sda_padoen_o	out	I2C 数据输出使能，低有效。在本实验中，高电平输出时输出高阻。
i2c_addr_2byte	in	寄存器地址是 8 位还是 16 位，1：16 位，0:8 位
i2c_read_req	in	I2C 寄存器读请求
i2c_read_req_ack	out	I2C 寄存器读请求应答
i2c_write_req	in	I2C 寄存器写请求
i2c_write_req_ack	out	I2C 寄存器写请求应答
i2c_slave_dev_addr	in	I2C 设备地址，8bit，最低位忽略，有效数据位是高 7 位。
i2c_slave_reg_addr	in	寄存器地址，8 位地址时，低 8 位有效
i2c_write_data	in	写寄存器数据
i2c_read_data	out	读寄存器数据
error	out	设备无应答错误

i2c_master_top 模块端口

i2c_eeprom_test 模块完成 EEPROM 的读写，EEPROM 设备地址是 A0，程序中将地址 00 的数据读出，然后通过 LED 显示，在 KEY2 按下时，数字加一并再次写入 EEPROM 并显示出来。在 I2C 控制器中，代码的大部分功能在备注中也有很多批注。

4 实验现象

下载实验程序后，可以看到 LED 显示一个二进制数字，这个数字是存储在 EEPROM 中 00 地址的数据，数据是随机的，这个时候按键 KEY2 按下，数字加一，并写入了 EEPROM，再次下载程序，可以看到直接显示更新后的数据。



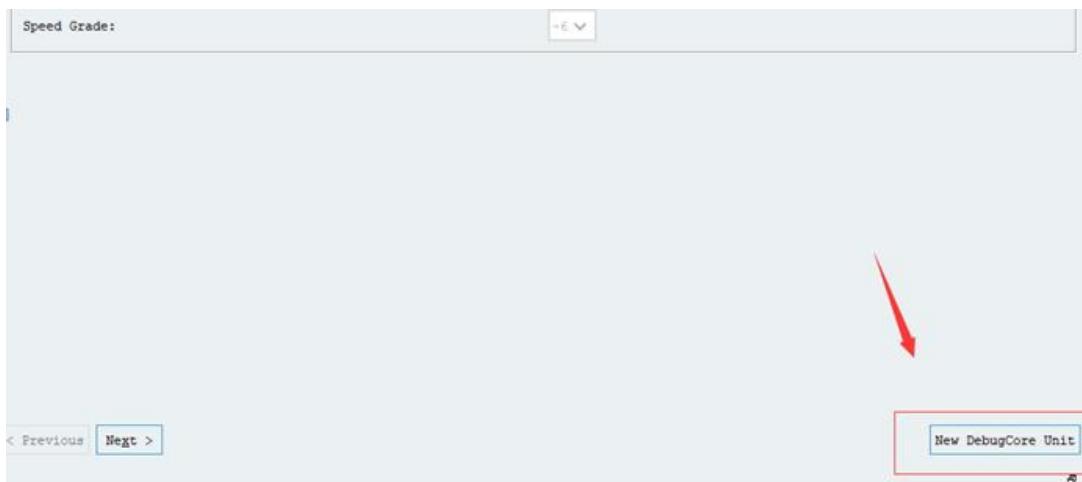
5 使用在线调试 Fabric Debugger 观察信号

使用 Fabric Debugger 可以非常直观的看到程序在开发板上运行时各个信号的变化，在本例程中添加一个 Debug Core 来观察程序运行时各数据线的变化情况。

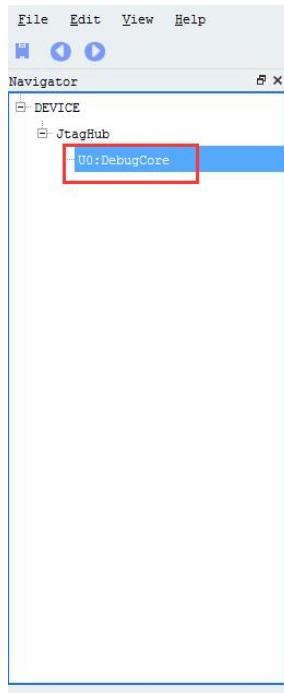
按红色标记，在单击 “Inserter”：



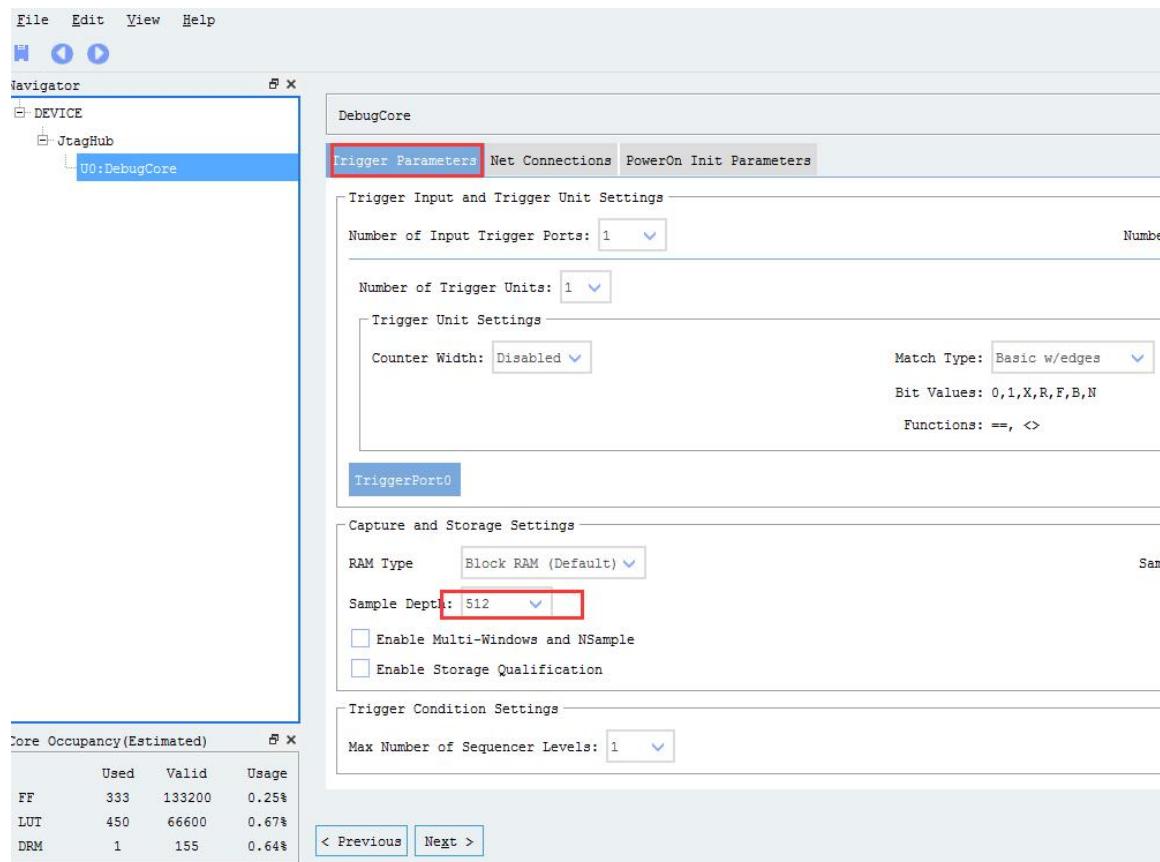
在弹出如下界面下选择 New DebugCore Unit 来新建一个 Debug Core：



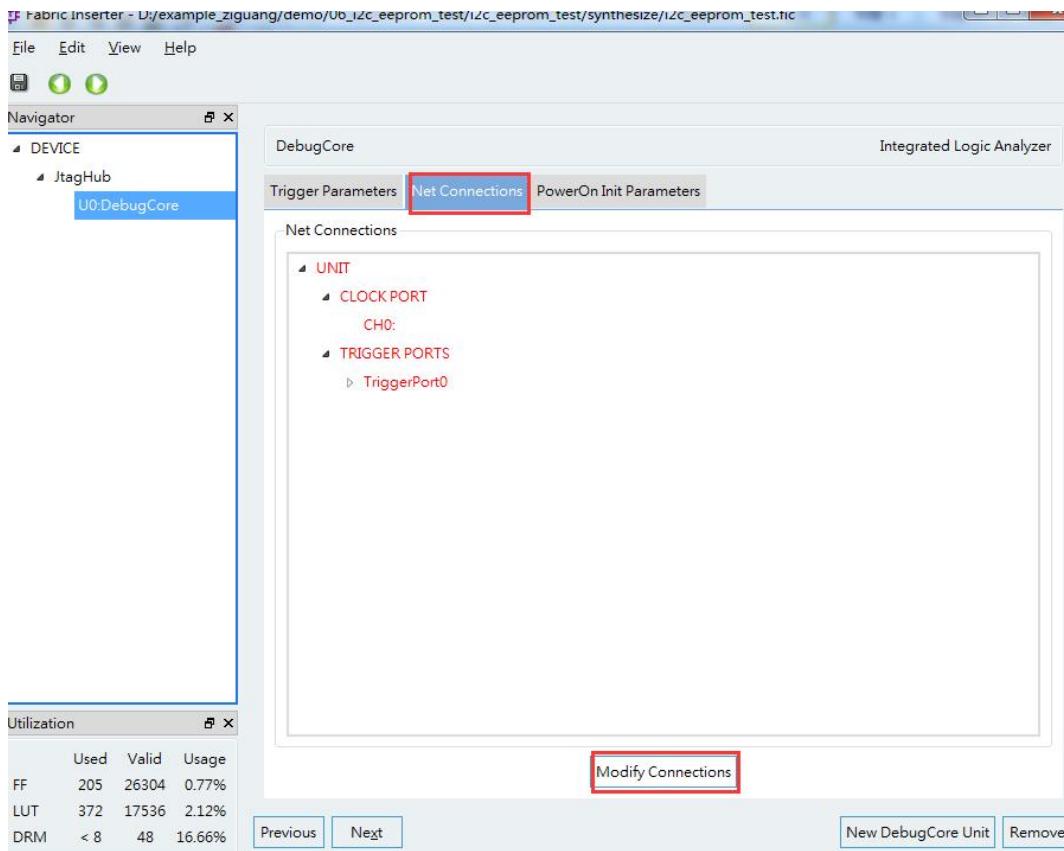
单击 U0: DebugCore：



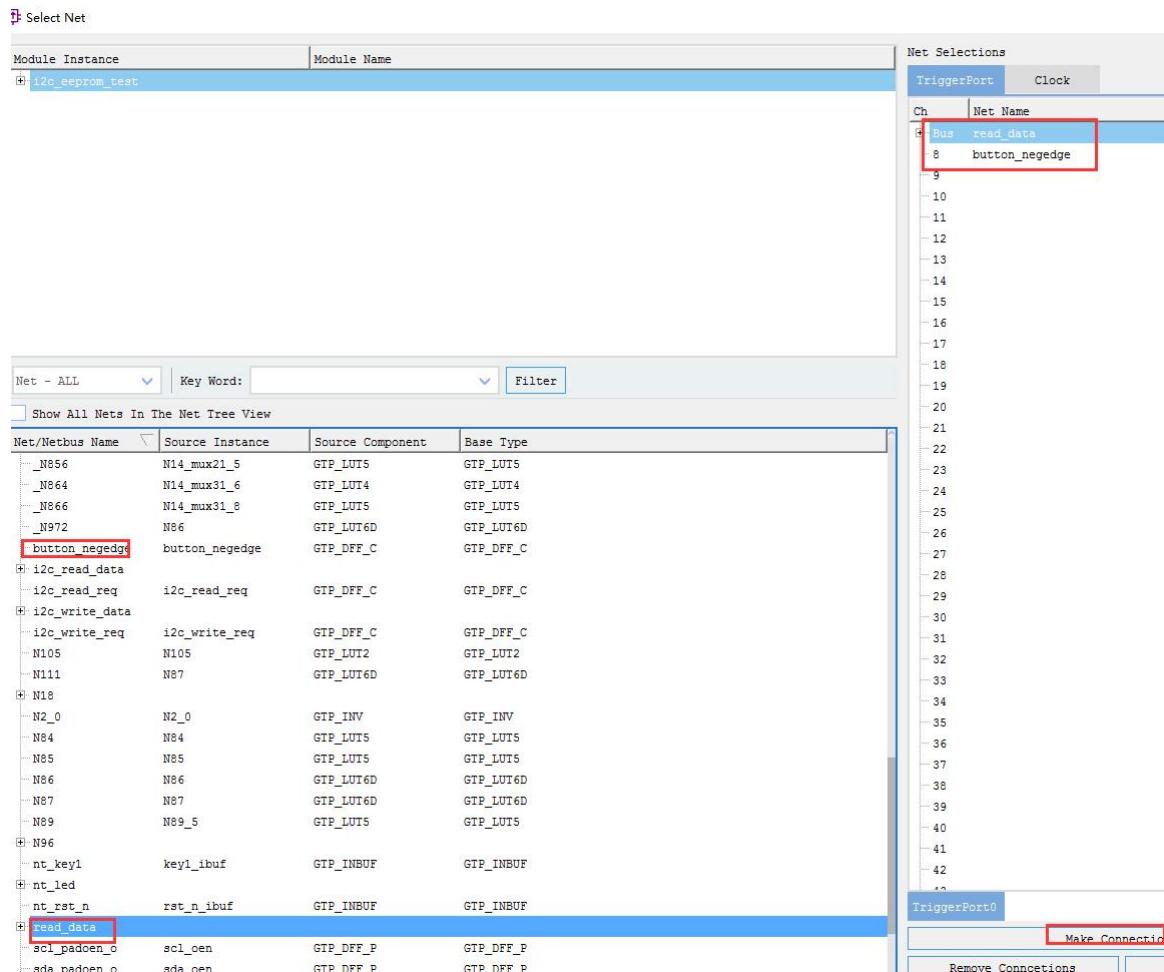
会跳到参数设置界面，在"Trigger Parameters"栏中 Sample Depth 采样深度自己根据需求选择，这里选择默认，其它的设置保持不变。



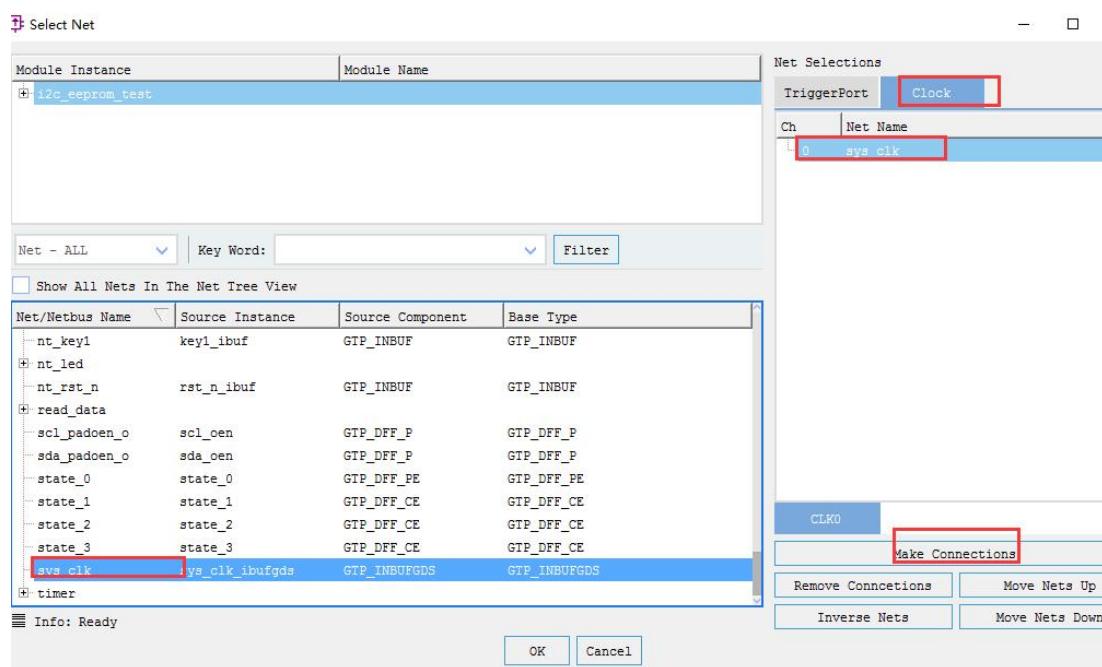
在"Net Connections"栏中选择 Modify Connections,进行 Net 修改;



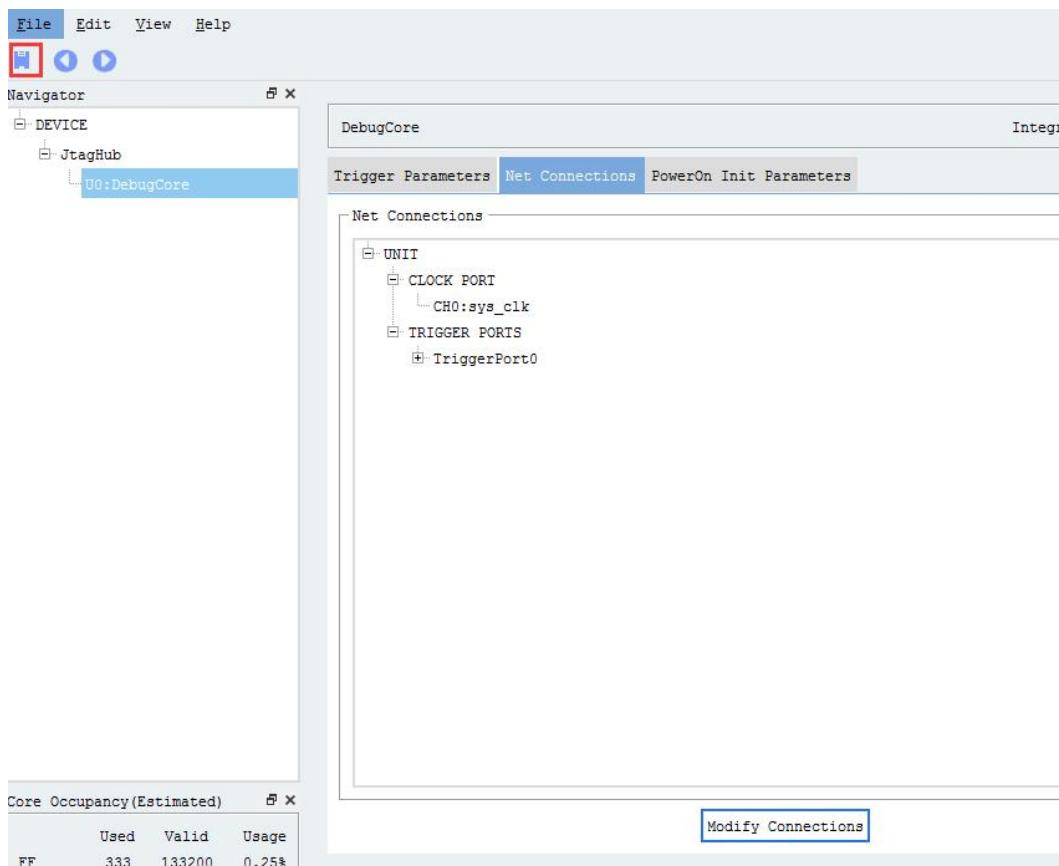
弹出的界面中首先对 TriggerPort 栏中观测信号关联，在这里，选择左侧 read_data 和 button_nededge,然后单击 Make Connections,可以在 TriggerPort 栏中看到关联的观测信号 read_data 和 button_nededge;



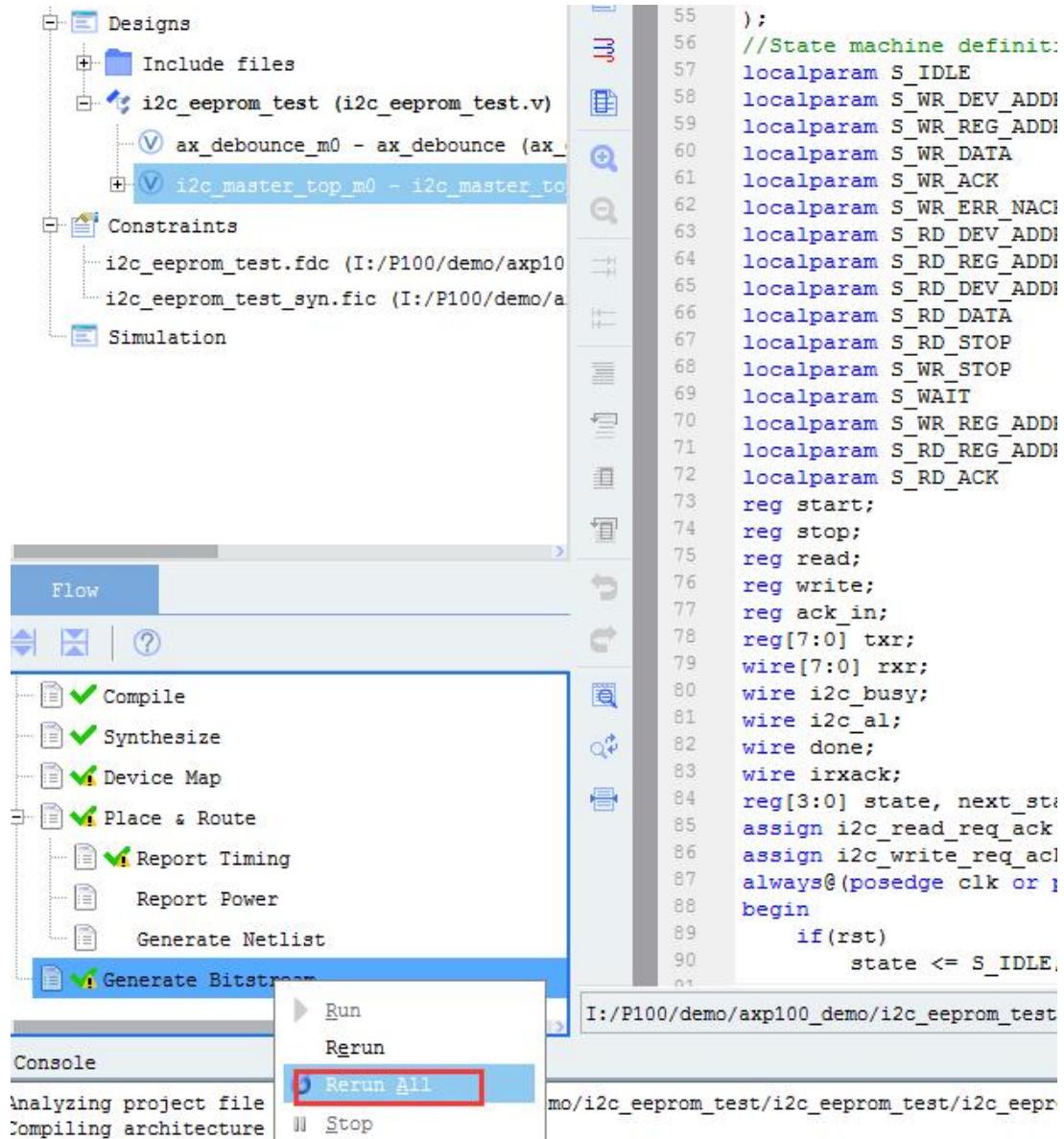
以同样的方式对 Clock 栏中时钟信号关联，选择左侧 sys_clk,然后单击 Make Connections,可以在 Clock 栏中看到关联的观测信号 sys_clk,然后单击 OK;



再在如下界面中单击保存按钮进行保存并关闭即可。



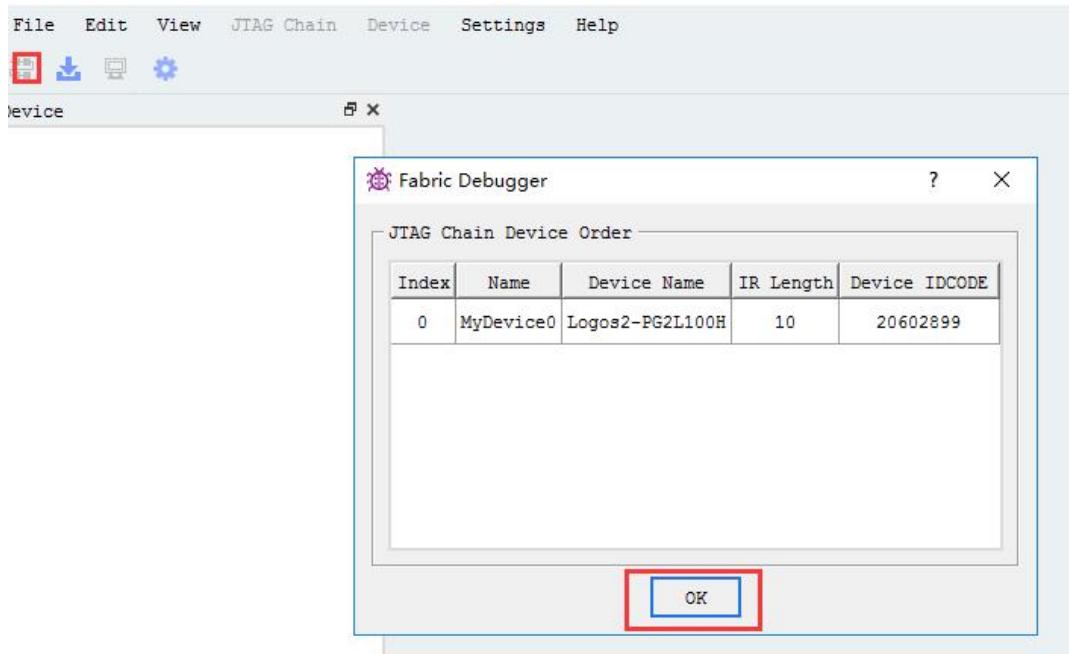
接下来重新生成.sbit 文件，双击 Generate Bitstream，会生成带有在线调试的 DebugCore。



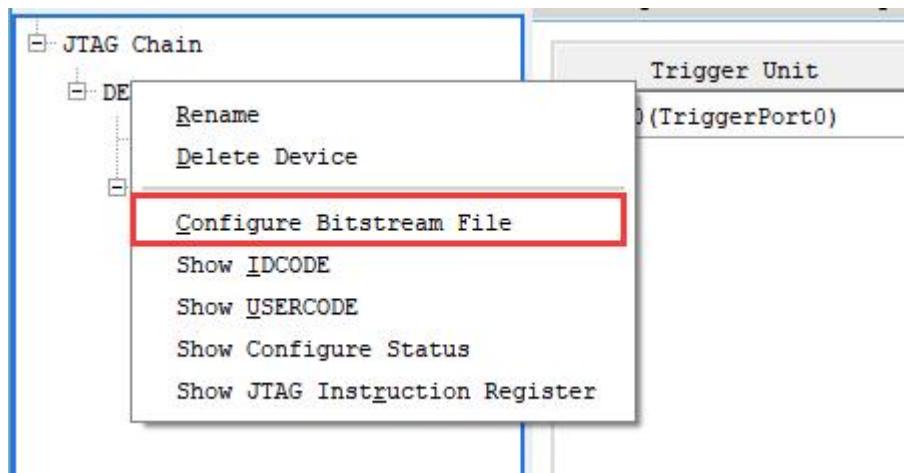
上面的工作准备好后，接下来才是正式开始调试，单击 Debugger（确保开发板上电并连接了下载器），



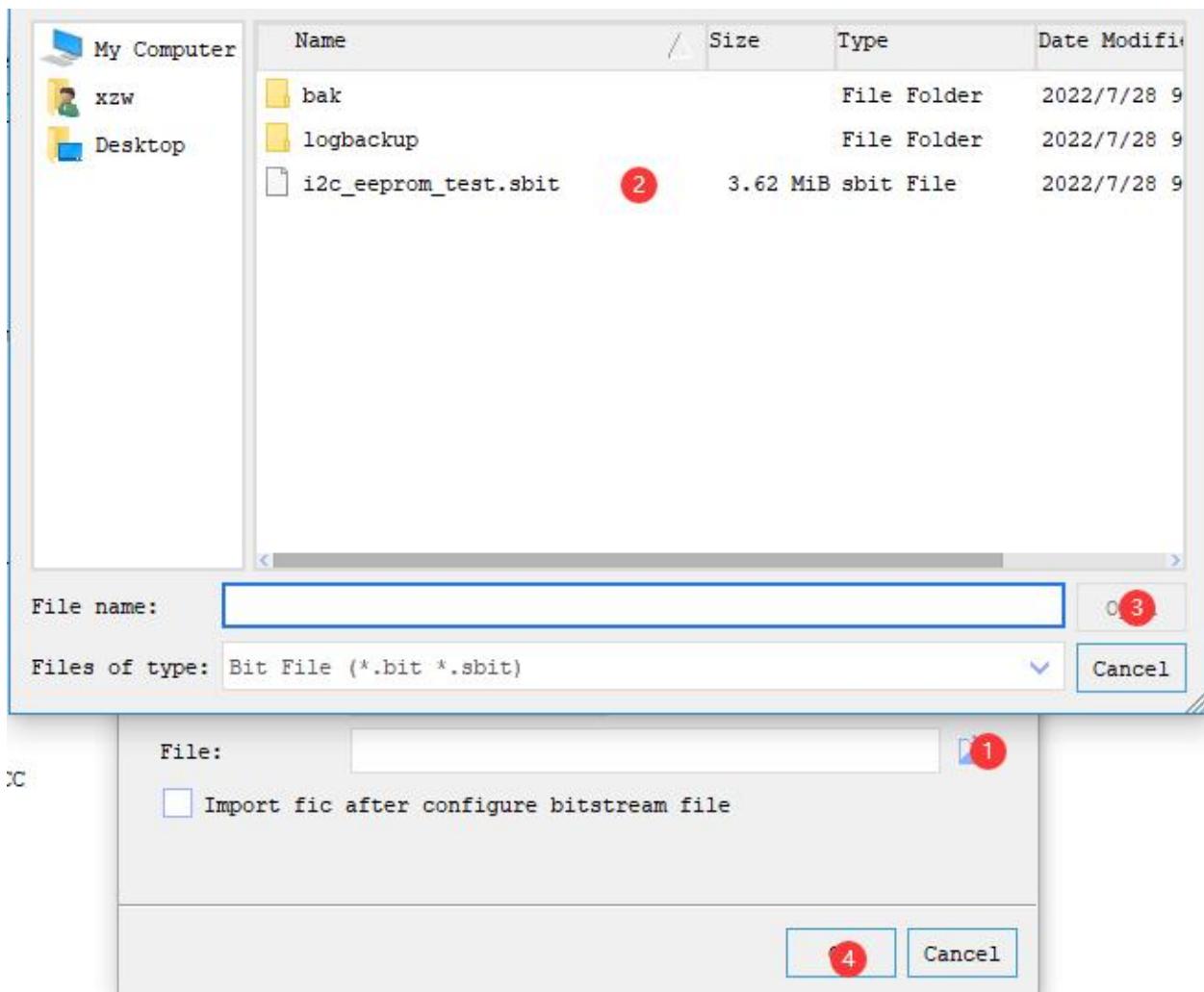
单击 JTAG 扫描按钮，在找到后单击 OK；



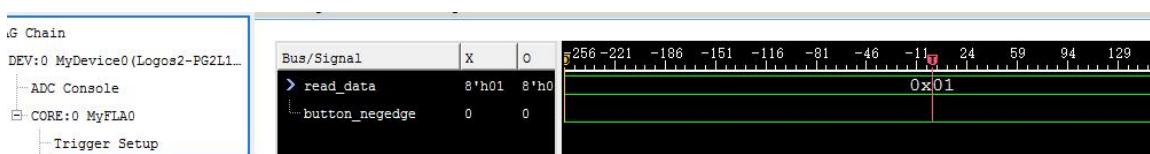
右击 DEV:0MyDevice0，在下拉菜单中单击 Configure Bitstream File



再按如下图中进行设置，设置完成后按 OK 键；



到此，程序已下载到 FPGA 中，可以在线观察所需要的信号，选择 Waveform，选择触发模式为连续，并单击 run；



可以看到 read_data 数据，每按一次 KEY2 键运行一次可以看到数据增加 1。

