10 PDS 软件中 FIFO 存储器 IP 配置和使用

工程源码	
相关视频课程	

章节导读

FIFO（First In First Out），即先进先出。FPGA 或者 ASIC 中使用到的 FIFO 一般指的是对数据的存储具有先进先出特性的一个缓存器，常被用于数据的缓存或者高速异步数据的交互。它与普通存储器的区别是没有外部读写地址线，这样使用起来相对简单，但缺点就是只能顺序写入数据，顺序的读出数据，其数据地址由内部读写指针自动完成，不能像普通存储器那样可以由地址线决定读取或写入某个指定的地址。

本章我们将侧重介绍 FIFO 的基础知识并以双时钟 FIFO 为例讲解，来向大家介绍 Pango FIFO IP 核的使用方法。

10.1 FIFO 相关知识

10.1.1FIFO 结构

FIFO 从读写时钟上来分有两类结构：单时钟 FIFO（同步 FIFO）和双时钟 FIFO（异步 FIFO）。单时钟 FIFO 具有一个时钟（读写共用一个时钟）输入，因此所有输入信号的读取都是在这个时钟的上升沿进行的，所有输出信号的变化也是在这个时钟信号的上升沿的控制下进行的，即单时钟 FIFO 的所有输入输出信号都是同步这个时钟信号的。而在双时钟 FIFO 结构中，写端口和读端口分别有独立的时钟，所有与写相关的信号都是同步于写时钟 wr_clk 的，所有与读相关的信号都是同步于读时钟 rd_clk 的。下面图是双时钟 FIFO 的整体框图和内部实现的框图，有单独的模块对读写时钟域进行同步处理。

10.1.2FIFO 应用场景

（1）单时钟 FIFO：

单时钟 FIFO 常用于片内数据交互。例如，在 FPGA 的控制下从外部传感器读取到的一连串传感器数据，首先被写入 FIFO 中，然后再以 UART 串口波特率

将数据依次发送出去。由于传感器的单次读取数据可能很快，但并不是时刻都需要采集数据，例如某传感器使用 SPI 接口的协议，FPGA 以 2M 的 SPI 数据速率从该传感器中读取 20 个数据，然后以 9600 的波特率通过串口发送出去。因为 2M 的数据速率远高于串口 9600 的波特率，因此需要将从传感器中采集到的数据首先用 FIFO 缓存起来，然后再以串口的数据速率缓慢发送出去。这里，由于传感器数据的读取和串口数据的发送都是可以同步于同一个时钟的，因此可以使用单时钟结构的 FIFO 来实现此功能。

（2）双时钟 FIFO：

双时钟 FIFO 的一个典型应用就是异步数据的收发，所谓异步数据是指数据的发送端和接收端分别使用不同的时钟域。使用双时钟 FIFO 能够将不同时钟域中的数据同步到所需的时钟域系统中。例如，在一个高速数据采集系统中，实现将高速 ADC 采集的数据通过千兆以太网发送到 PC 机。ADC 的采样时钟(CLK1)由外部专用锁相环芯片产生，则高速 ADC 采样得到的数据就是同步于该 CLK1 时钟信号，在 FPGA 内部，如果 FPGA 工作时钟(CLK2)是由独立的时钟芯片加片上锁相环产生的，则 CLK1 和 CLK2 就是两个不同域的时钟，他们的频率和相位没有必然的联系，假如 CLK1 为 65M，CLK2 为 125M，那么就不能使用 125M 的数据来直接采集 65M 速率的数据，因为两者数据速率不匹配，在采集过程中会出现包括亚稳态问题在内的一系列问题，所以这里就可以使用一个具备双时钟结构的 FIFO 来进行异步数据的收发。

10.1.3FIFO 常见参数

1) FIFO 的宽度：即 FIFO 一次读写操作的数据位；
2) FIFO 的深度：指的是 FIFO 可以存储多少个 N 位的数据（如果宽度为 N）。
3) 满标志：FIFO 已满或将要满时由 FIFO 的状态电路送出的一个信号，以阻止 FIFO 的写操作继续向 FIFO 中写数据而造成溢出。
4) 空标志：FIFO 已空或将要空时由 FIFO 的状态电路送出的一个信号，以阻止 FIFO 的读操作继续从 FIFO 中读出数据而造成无效数据的读出。
5) 读时钟：读操作所遵循的时钟，在每个时钟沿来临时读数据。
6) 写时钟：写操作所遵循的时钟，在每个时钟沿来临时写数据。

10.1.4实现 FIFO 的方法

使用 FIFO 实现用户功能设计主要有三种实现方式。

第一种为用户根据需求自己编写 FIFO 逻辑，当对于 FIFO 的功能有特殊需求时，可以使用此种方式实现。

第二种方式为使用第三方提供的开源 IP 核，此种 IP 核以源码的形式提供，能够快速的应用到用户系统中，当用户对 FIFO 功能有特殊需求时，可以在此源码的基础上进行修改，以适应自己的系统需求。

第三种方式为使用 PDS 软件提供的免费 FIFO IP 核，此种方式下，PDS 软件为用户提供了友好的图形化界面方便用户对 FIFO 的各种参数和结构进行配置。由于该 FIFO IP 核已经提供了大部分设计所需的所有功能，因此在系统设计中，推荐使用该 FIFO IP 核进行系统设计。

10.2程序设计

新建 PDS 工程，首先在菜单栏里选择“Tools”然后单击 “IP Compiler”选项，如图 10-1 所示。

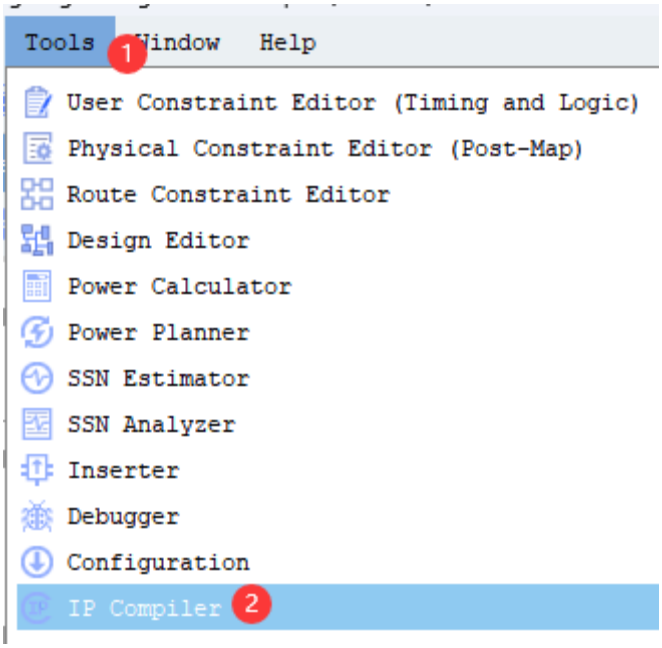

图 10-1“IP Compiler”页面

点击图 10-1“IP Compiler”页面中的“IP Compiler”后会跳转图 10-2 页面。

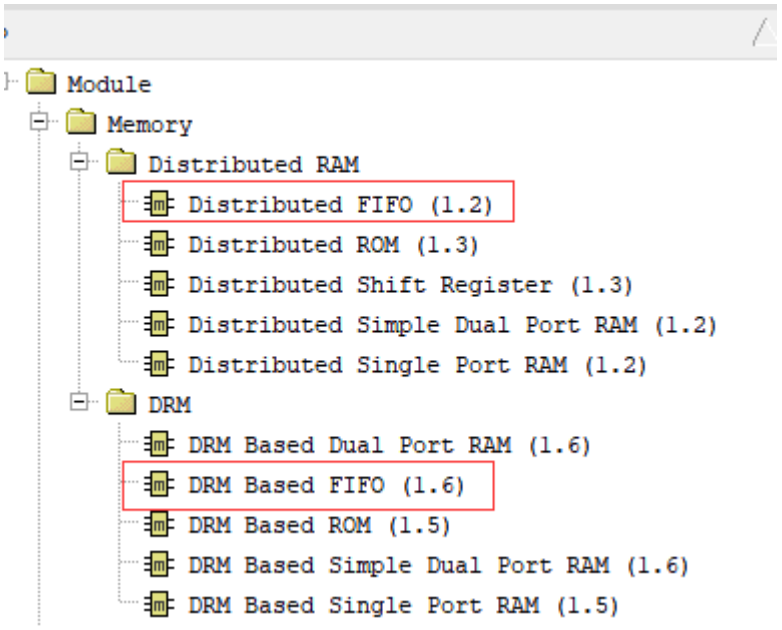

图 10-2 FIFO 类型

在图 10-2 中，我们可以看出有两种 FIFO IP 可以供我们选择使用，一种是 Distributed FIFO（分布式 FIFO），另一种是 DRM Based FIFO(基于 DRM 的 FIFO)。Distributed FIFO 是紫光同创基于 FPGA 片内资源设计的 IP。先简单说说两个的差别，两者最主要的差别是生成的 Core 所占用的 FPGA 资源不一样。

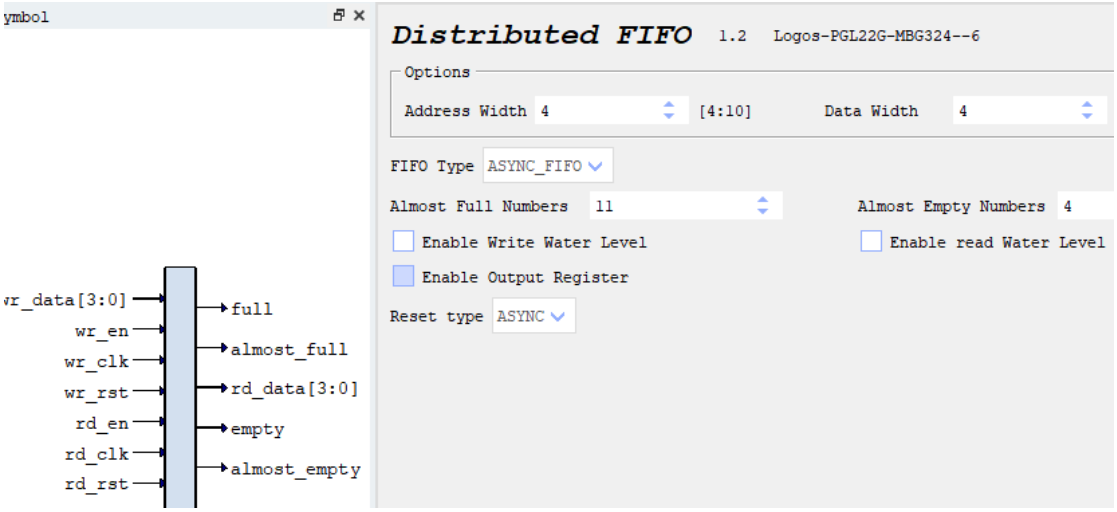

图 10-3 Distributed FIFO IP 界面

在图 10-3 中可以看出 “Address Width”(地址位宽)，地址位宽的合法范围是 4-10， 即 FIFO 存储的存储容量最大为 1023 个存储单元，而且该 FIFO 的读写数据位宽必须保持一致。

DRM Based FIFO IP 是基于 DRM(Dedicated RAM Module)即专用 RAM 模块

设计的 IP，通过对 DRM 的级联调用实现 FIFO IP 的设计。 Logos 系列 FPGA 的 DRM 有高达 18K bits 的存储单元并且容量可被独立配置为 2 个 9K bits 或者 1 个 18K bits。每个 DRM 都能支持 DP（True Dual Port，双口）模式即读写数据的位宽不一致，同时也以被配置为 SP（Single Port，单口）模式即读写数据位宽一致，以及同步\异步 FIFO 模式。DRM 资源还支持输入寄存器（IR）、输出寄存器（OR）以及 Core Latch，这使得 DRM 级联使用时拥有更加出色的性能表现。

在图 10-4 中我们首先点击标签 1 中的“DRM Based FIFO”按钮，选择创建“DRM Based FIFO” IP 核。在标签 2“Pathname”选项里面可以修改 IP 生成的路径，这里保持默认路径即可。再将标签 3 中“Instance Name”选项命名为“fifo”。最后点击标签 4 中的“Customize”按钮进入 FIFO IP 参数配置页面。

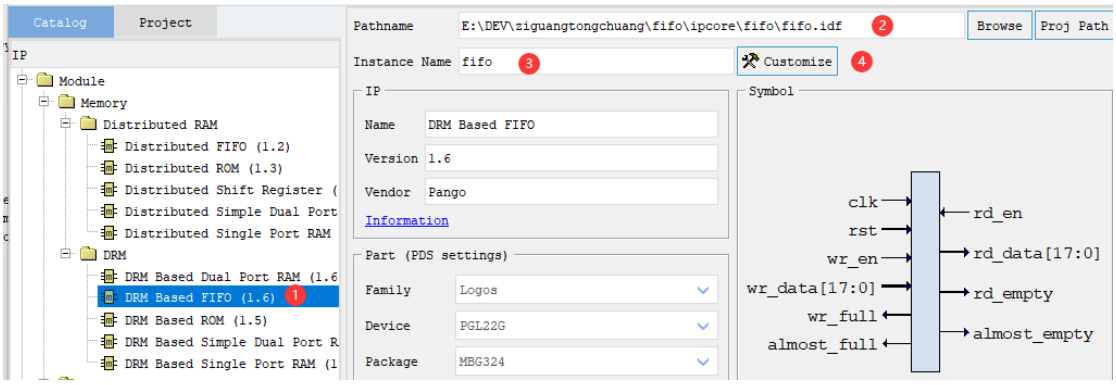

图 10-4 DRM Based FIFO IP

点击图 10-4 中的“Customize”按钮进入图 10-5 中。

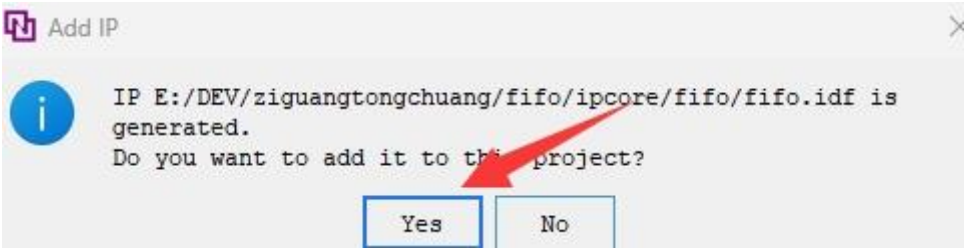

图 10-5 Add IP 弹窗

图 10-5 中表示的是 fifo.idf 文件已经生成，询问我们是否将该文件加载到我们的工程中。因为这个 IP 我们后续是要在工程中用到的，因此这里直接点击图 10-5 中的“Yes”按钮即可。然后软件会自动进入 DRM Based FIFO IP 的参数配置界面，如图 10-6 所示。

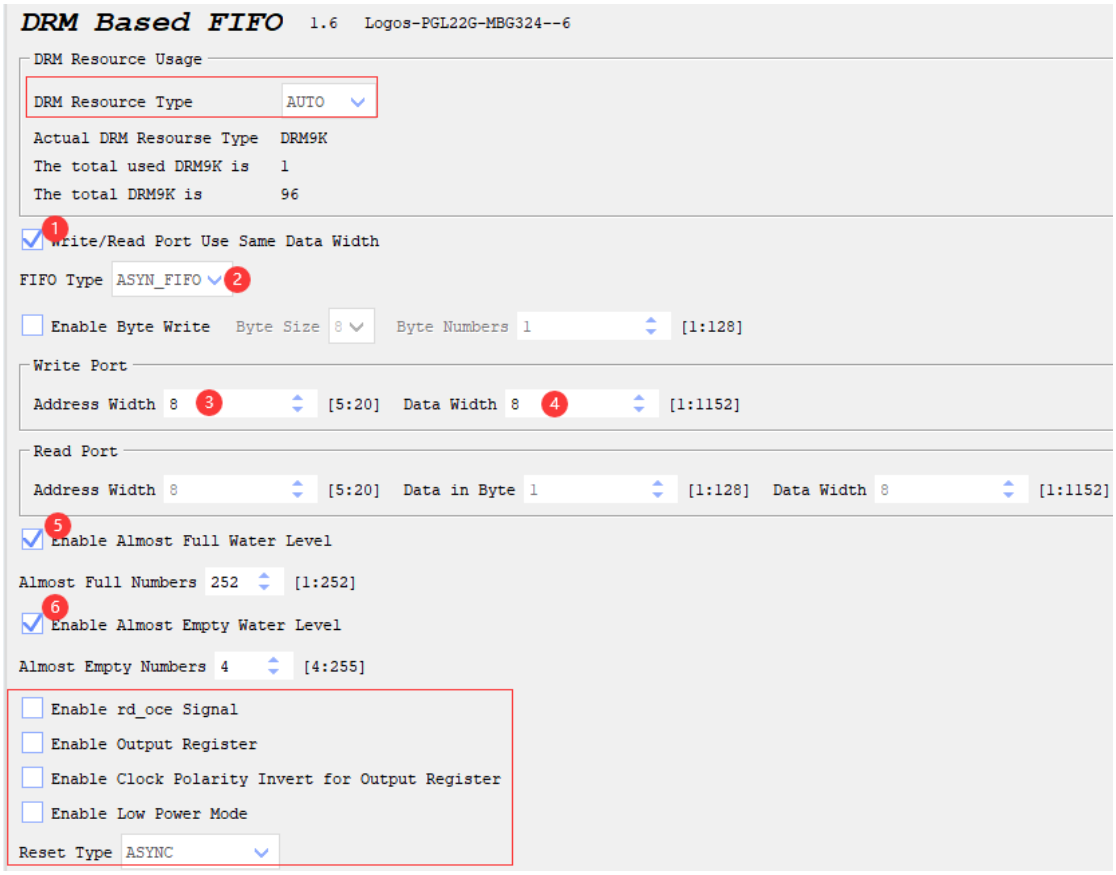

图 10-6 fifo 配置页面

图中展示了 FIFO 存储器的若干个相关参数，包括数据位宽，地址位宽，以下分别介绍：

标签 1 中“DRM Resource Type”（DRM 资源类型），一般我们设置为“AUTO”模式,即自动模式让 PDS 软件自动选择 DRM 资源类型；

标签 2 中“Write/Read Port Use Same Data Width”（读写端口位宽是否一致），这里我们对该选项进行勾选，勾选后，我们在本实验中只需要对写入端口的地址位宽和数据位宽进行设置即可；

标签 3 中 “Address Width”(地址位宽)，地址位宽的合法范围是 5-20，该实验我们设置成 8, 则 FIFO 存储的存储容量为 255 个存储单元；

标签 4 中“Data Width”(数据位宽)，代表了 FIFO 存储器每个地址（存储单元）中存储的数据的位宽，可根据实际应用需求在 1 位到 1152 位之间任意设定,这里我们设置为 8；

标签 5 中“Enable Almost Full Water Level”：是否配置使能 wr_water_level 信号，使能 wr_water_level 信号可以对写端口数据进行计数；

标签 6 中“Enable Almost Empty Water Level”：是否配置使能 wr_empty_level 信号，使能 wr_empty_level 信号可以对读端口数据进行计数；

在本章实验中对 FIFO IP 做出以上六种设置，在图 10-6 中还有一些其它的设置没有提及，如：“Enable Byte Write”（使能 Byte Write 功能），以字节为单位写入数据。当勾选该选项时需要对“Byte Size”（字节位宽，字节位宽可以设置为 8 或 9）和“Byte Numbers”（字节个数）进行配置。举个例子，当写入数据是 32bit,“Byte Size”设置为 8，那么就需要 4 个写使能信号。这个使能信号与输入数据各位的对应关系如下表所示。从表中不难看出，当 we[0]有效时，会将输入数据的低 8-bit 写入到目标地址中；而当 we[3]有效时，只会将输入数据的高 8-bit 写入到目标地址；本章不使用 Byte Write 功能，保持默认不用勾选。

表 1 使能信号与输入数据各位的对应关系

WE[3]				WE[2]				WE[1]				WE[0]			
31	30	…	24	23	22	…	16	15	14	…	8	7	6	…	0

Almost Full Numbers：配置 FIFO Almost Full 个数，设置为该选项后面的允许的最大值，本实验不做设置，保持默认 252 设置即可。

Almost Empty Numbers：配置 FIFO Almost Empty 个数，设置为该选项后面的允*许的最*小值，本实验不做设置，保持默认即可。

Enable rd_oce Signal：是否配置使能 rd_oce（输出寄存器选项）信号，输出寄存使能信号为高时对应地址有效，读数据会寄存输出，若输出寄存使能信号为低时对应地址无效，读数据保持。本实验不需要使能读信号，保持默认不用勾选。

Enable Output Register：输出寄存器选项。如果勾选了“Enable Output Register”信号， 虽然会改善代码的时序性能，但会使输出的数据延迟一拍，这不利于我们在仿真窗口中直观清晰地观察信号，所以本实验未使用保持默认不用勾选。

Enable Clock Polarity Invert for Output Register：配置读使能端口输出时钟反向极性，端使能读口输出时钟反向极性时，必须要勾选读使能端口输出寄存（Enable Output Register），所以本实验保持默认不用勾选。

Enable Low Power Mode：配置是否使能低功耗模式，本实验不需要配置成低功耗模式，本实验保持默认不用勾选。

Reset Type：配置复位方式："ASYNC"异步复位，"SYNC"同步复位，"Sync_Internally "异步复位同步 释放，本实验保持默认选项"ASYNC"异步复位（低有效）。

值得注意的是紫光的 FIFO 的读模式是在给了读数据使能之后，读数据才出来，可以结合读模式的时序图来进行理解。

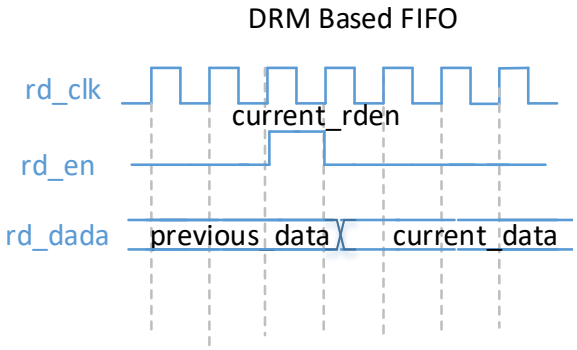

图 10-7 fifo 读相关的工作模式

至此本实验所需要的异步 FIFO IP 已介绍并配置完成，接下来点击图 10-8“Customize IP”窗口左上角的 “Generate”按钮即可。

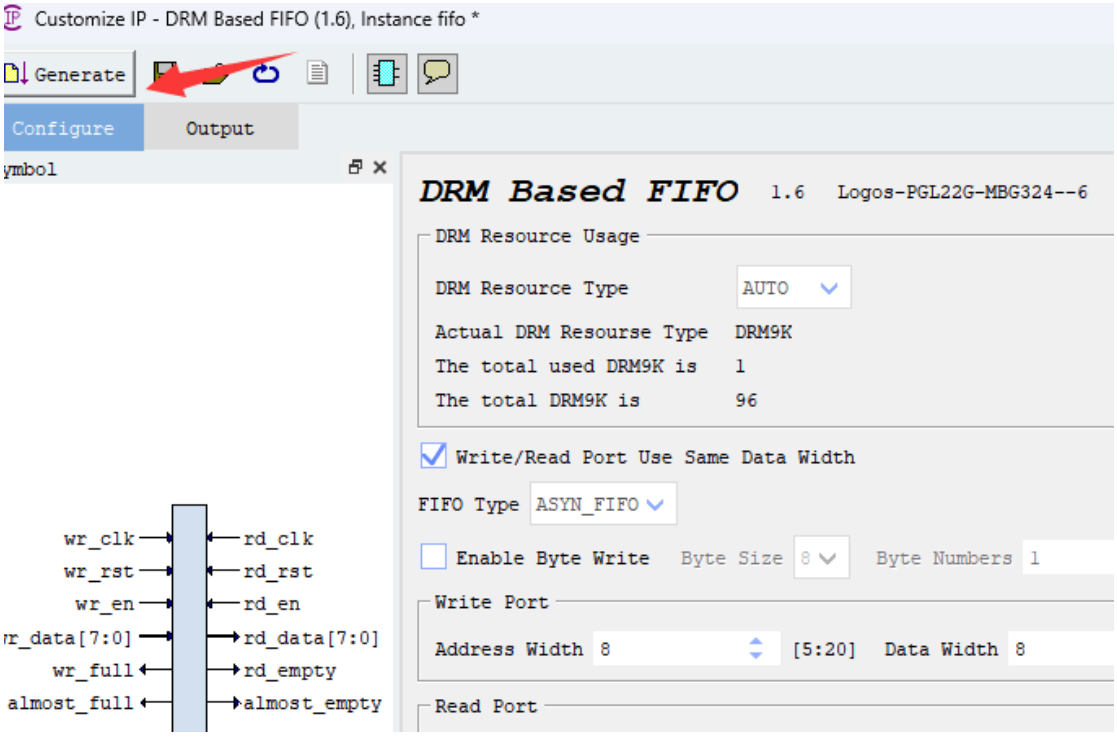

图 10-8“Customize IP”窗口

点击图 10-8 中的“Generate”按钮，进入图 10-9 界面。至此，FIFO IP 核

配置成功。

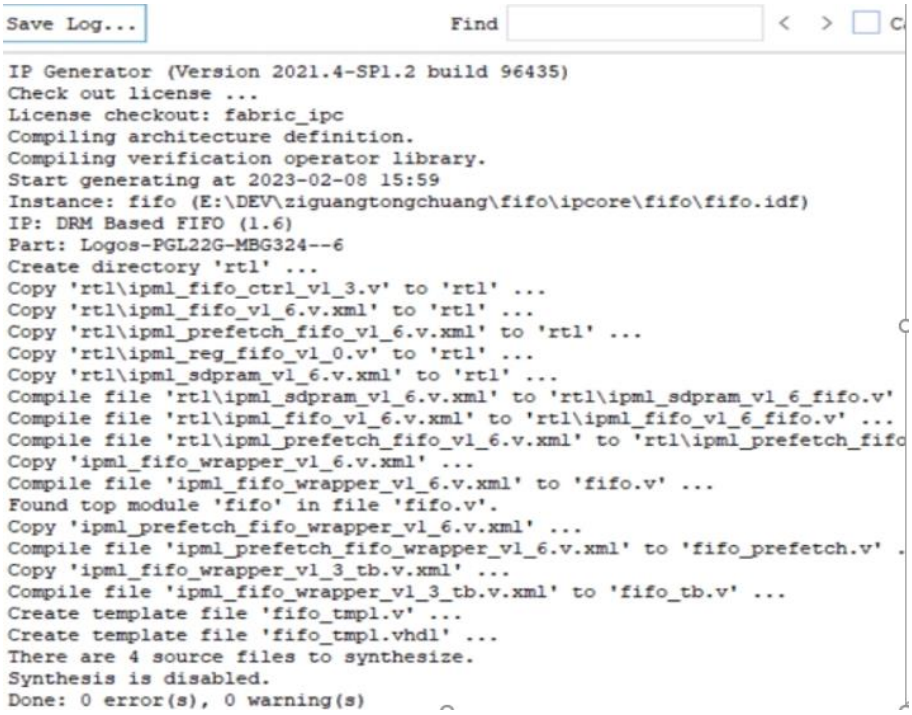

图 10-9 FIFO IP 配置成功

IP 核配置成功后会自动弹出图 10-10 FIFO IP 的例化模板文件，接下来我们需要例化这些自动生成的代码。

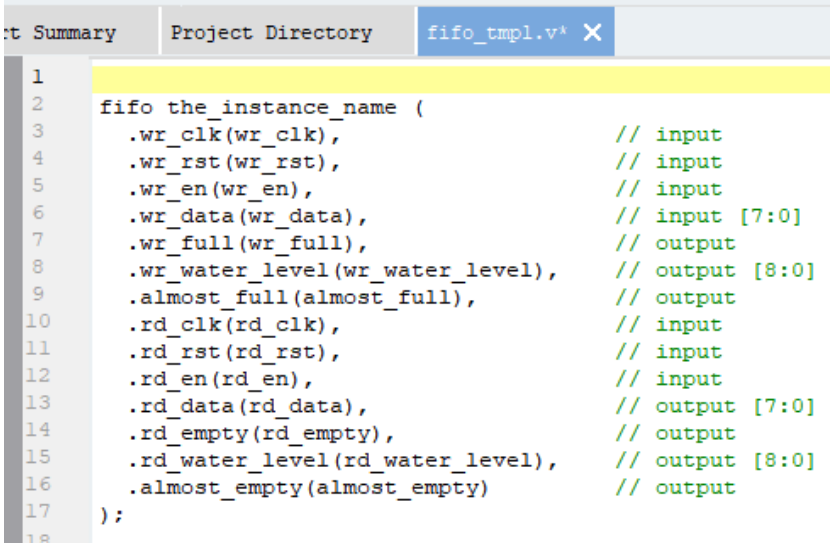

图 10-10 IP 的例化模板文件

将 IP 核配置过程中弹出的页面关闭，返回 Source 面板，如图 10-11 所示。

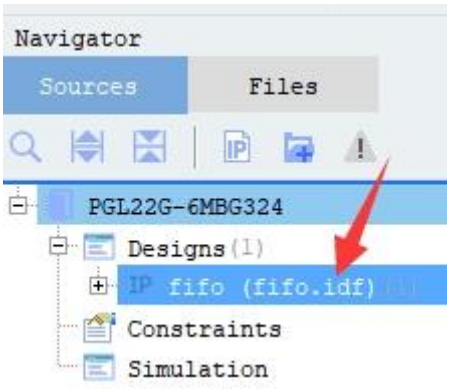

图 10-11 Source 面板

点击图 10-11 页面中箭头指向的 fifo.idf 文件左侧加号位置，可以看到 fifo.idf 核下有四个文件，点击图 10-12 中箭头所指的位置，打开 fifo.v 文件。

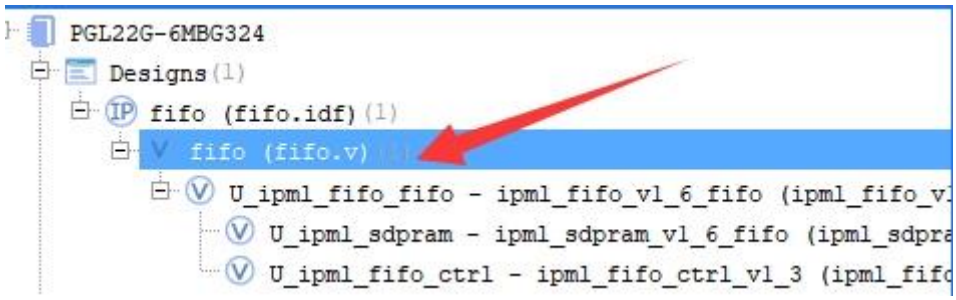

图 10-12 IP 核包含的文件

部分 fifo.v 文件内容如图 10-13 所示，若要调用 FIFO IP，我们可以选择将图 10-13 中的端口例化到需要调用 FIFO IP 的模块中，或者是直接例化图 10-10 中内容。

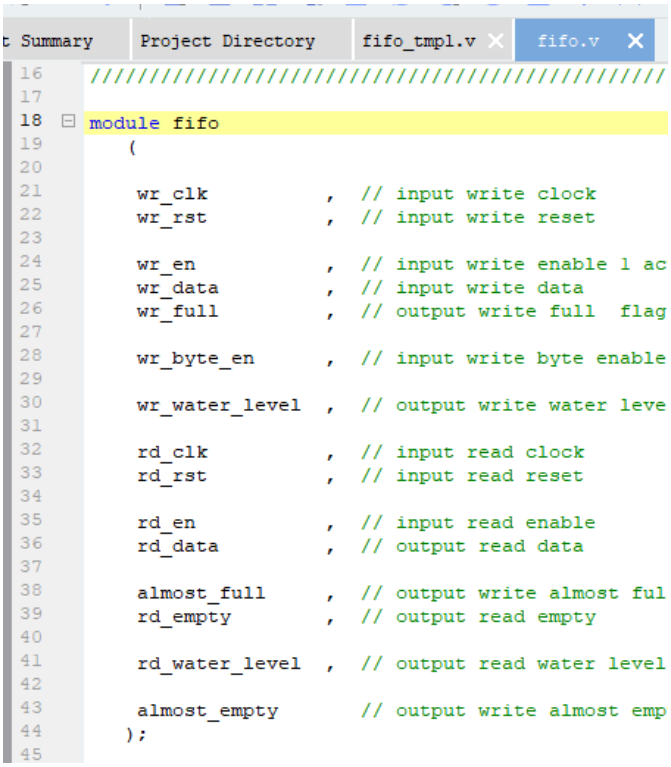

图 10-13 fifo.v 文件部分内容

至此，对于如何配置双时钟 FIFO IP 以及如何使用该 IP 已经介绍完毕。

10.2.1代码设计

接下来我们创建一个 verilog 源文件，其名称 top_fifo，本设计的端口列表和源码比较简单，代码的主要作用就是例化 FIFO IP 以及 PLL IP 代码如下：

```
module top_fifo(
     clk ,
     reset_n,
     full,
     almost_full,
     empty,
     almost_empty,
     wr_data_count,
     rd_data_count
);

 input clk;
 input reset_n;
```

```
 output full;
 output almost_full;
 output empty;
 output almost_empty;
 output [8:0]wr_data_count;
 output [8:0]rd_data_count;

wire wr_clk;
wire locked;
reg rd_en ;
reg wr_en;
reg[9:0]counter;
reg[7:0] wr_data;
wire rd_clk;
wire [7:0]rd_data;

pll pll  (
    .clkin1   (clk),
    .pll_rst  (~reset_n),
    .clkout0  (rd_clk),
    .clkout1  (wr_clk),
    .pll_lock (locked)
    );


fifo   fifo
   (
   . wr_clk        (wr_clk) ,  // input write clock
   . wr_rst        (~reset_n) ,  // input write reset
   . wr_en         (wr_en) ,  // input write enable 1 active
   . wr_data       (wr_data) ,  // input write data
   . wr_full       (full) ,  // output write full  flag 1 active
   . wr_water_level(wr_data_count) ,  // output write water level
   . rd_clk        (rd_clk) ,  // input read clock
   . rd_rst        (~reset_n) ,  // input read reset
   . rd_en         (rd_en) ,  // input read enable
   . rd_data       (rd_data) ,  // output read data
   . almost_full   (almost_full) ,  // output write almost full
   . rd_empty      (empty) ,  // output read empty
```

```
    . rd_water_level(rd_data_count),  // output read water level
    . almost_empty  (almost_empty)    // output write almost empty
    );


always@(posedge wr_clk or negedge reset_n)
 if(!reset_n)
  counter<=0;
 else if(counter==10'd1023)
  counter<=0;
 else
  counter<= counter+1'b1;


always@(posedge wr_clk or negedge reset_n)
 if(!reset_n)
  wr_en<=0;
else if(counter<=255)
  wr_en<=1;
 else
  wr_en<=0;


always@(posedge wr_clk or negedge reset_n)
 if(!reset_n)
  wr_data<=8'hff;
 else if(wr_en)
  wr_data<= wr_data-1'b1;
 else
  wr_data<=wr_data;

always@(posedge wr_clk or negedge reset_n)
 if(!reset_n)
  rd_en<=0;
 else if(counter>10'd511)
  rd_en<=1;
 else
  rd_en<=0;
endmodule
```

可以看出创建的 FIFO IP 采用的是独立时钟，分别为 wr_clk（写时钟）和

rd_clk（读时钟），其中 wr_clk 的时钟周期通过系统时钟控制为 20ns，rd_clk 的时钟周期通过 PLL IP 产生一个 25MHz 的时钟（时钟周期为 40ns）。由于 rd_clk 的周期是 wr_clk 周期的 2 倍，所以 rd_en 拉高的时间长度必须是 wr_en 拉高时间的 2 倍才能将写入的数据读完。

使用 PDS 将上面的设计内容进行分析和综合直至没有错误以及警告后，按开发流程接下来进入激励创建及仿真测试环节。作为一个 FPGA 数字逻辑的完整的开发流程，仿真环节是必不可少的，这一要求请务必引起各位初学者的重视。

10.3激励创建及仿真测试

为了测试仿真编写测试激励文件，添加并新建仿真文件命名为 top_tb.v。我们可以通过仿真方式来对该 IP 进行测试，周期时间值通过宏定义 define 设置为 20ns。具体代码如下：

```
`timescale 1ns / 1ns
`define  CLK_PERIOD 20

module top_tb();
  reg       clk;
  reg       reset_n;
  wire      full;
  wire      almost_full;
  wire      empty;
  wire      almost_empty;
  wire  [8:0] wr_data_count;
  wire  [8:0] rd_data_count;

reg grs_n;
 GTP_GRS GRS_INST(
.GRS_N (grs_n)
);

initial begin
 grs_n = 1'b0;
 #50000;
 grs_n = 1'b1;
```

```
 end


 initial clk = 1;
 always #(`CLK_PERIOD/2) clk = ~clk;

 initial begin
   reset_n=0;
   #21;
   reset_n=1;
   #2000000;
 end

top_fifo  top_fifo(
     .clk(clk) ,
     .reset_n(reset_n),
     .full(full),
     .almost_full(almost_full),
     .empty(empty),
     .almost_empty(almost_empty),
     .wr_data_count(wr_data_count),
     .rd_data_count(rd_data_count)
);

endmodule
```

testbench 代码例化了一个 GTP_GRS 模块，GTP_GRS 是一个全局复位模块，在 FIFO 或者 RAM IP 中使用了这个全局复位，所以使用了 FIFO 或者 RAM IP 的文件的仿真代码里面就需要对 GTP_GRS 模块进行例化，不然联合仿真会报错。

如何对模块仿真这里我们不再赘述，仿真波形如图 10-14 所示

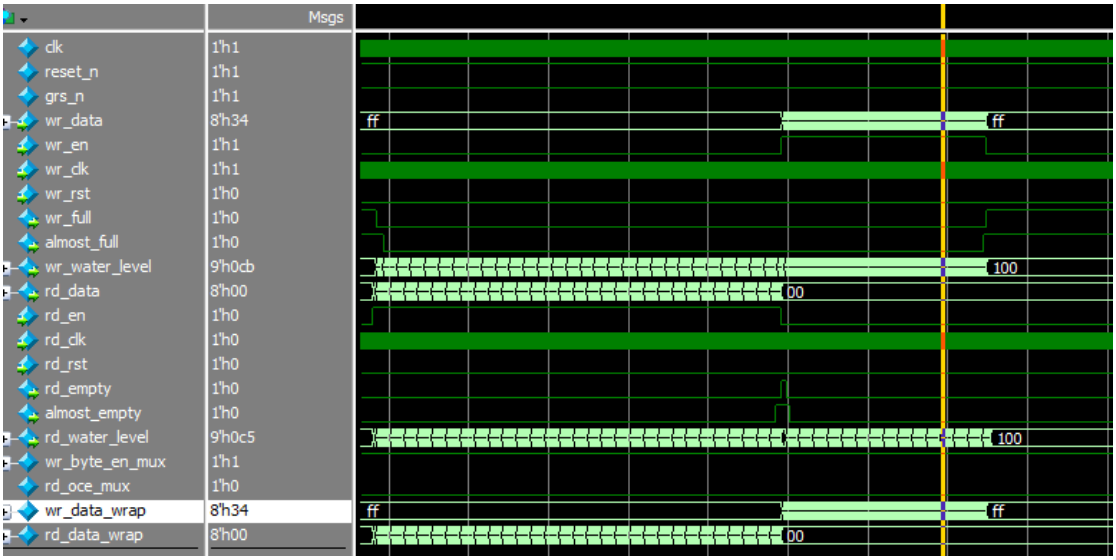

图 10-14 仿真波形全貌

对写数据部分放大后的波形，如图 10-15 所示：

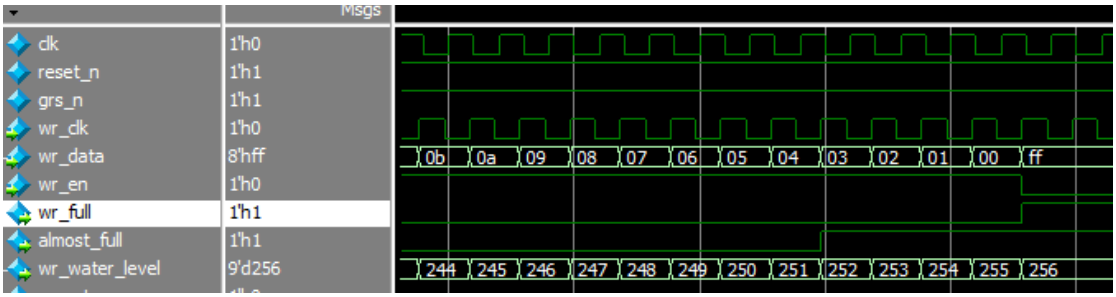

图 10-15 数据写入时放大的波形

wr_water_level 是对写入的数据进行计数，当写入第 252 个数据后，almost_full 信号开始拉高，这与图 10-6 中的 FIFO IP 设置一致，当数据写满 255 个数据后，wr_full 也开始拉高。

放大读取部分数据波形，如图 10-16 所示

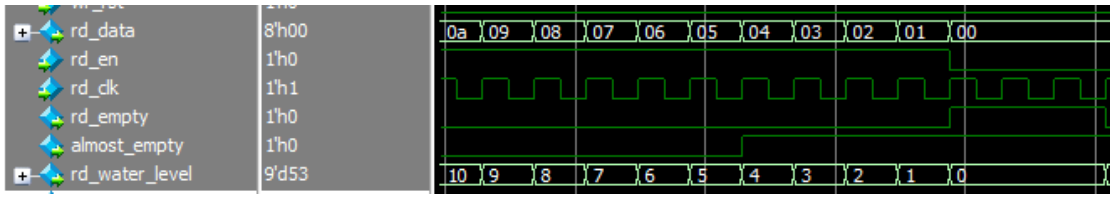

图 10-16 数据读出时放大的波形

由图 10-16 可知，当读完 251 个数据后，almost_empty 信号就被拉高，这是因为读完 251 个数据后，FIFO 还剩余 4 个数据，与图 10-6 中设置一致；当读完 255 个数据后，rd_empty 信号也被拉高。

通过对比以上的波形可以看到我们设置的 FIFO IP 处于正常工作状态。

10.4板级调试

本实验的板级验证环节，主要验证以下几个目标：

1. 能否正确将生成的 bit 文件下载到 PGL22G 开发板；
2. FIFO IP 数据读写以及相关信号变化情况；

系统所需硬件：

1. PGL22G 开发板；
2. 电源电缆一根；
3. 硬件条件符合实验要求，具有完全开发功能的 PC 机一台；

10.4.1添加 I/O 约束

通常，一个设计中的 FPGA 不会是独立使用的，FPGA 一定会与其他外设、接口相连接，比如时钟，按键等。因此，FPGA 设计需要指定对应的 IO 引脚位置信息。

添加 IO 约束的方法非常简单：点击图 10-17 上方工具栏“Tools”栏中“User Constranit Editor(Timing and Logic)”后点击 “Pre Synthesize USE ”选项。

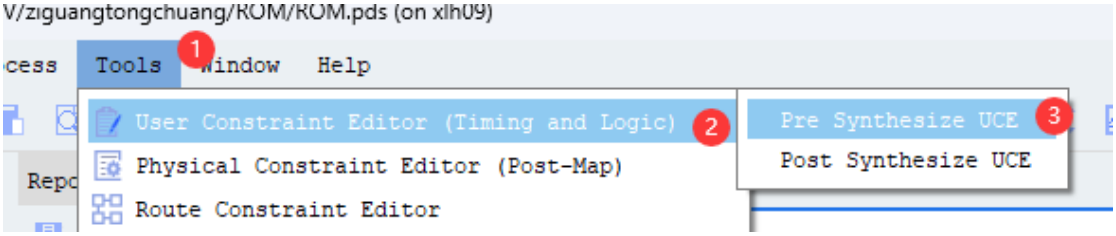

图 10-17“Tools”工具栏

点击图 10-17 中的“Pre Synthesize USE ”选项后，弹出图 10-18 界面，首先点击 “Pre Synthesize USE ” 选项，然后点击“Device”选项，最后点击“I/O”选项，进入管脚分配页面。

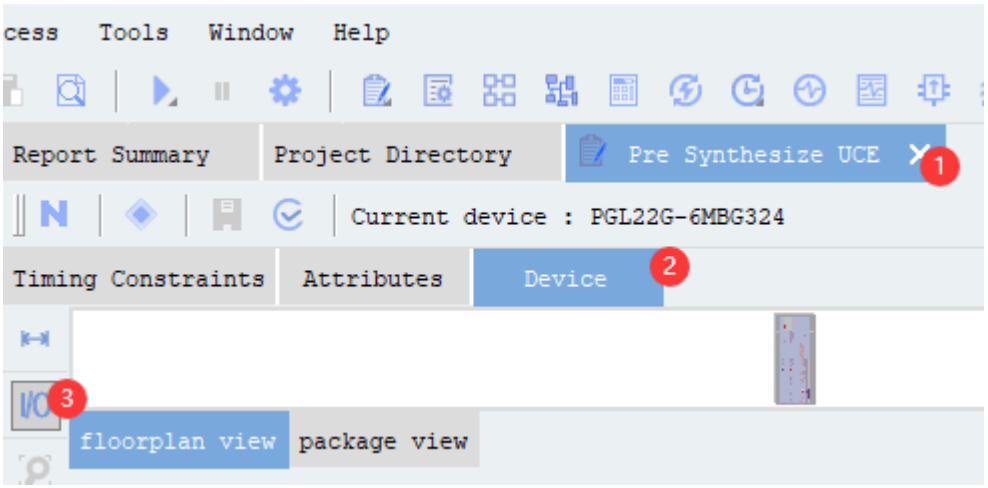

图 10-18 管脚分配入口

打开管脚约束窗口如图 10-19，其中 LOC，VCCIO，IOSTANDARD 是我们要约束的内容。这里，参考我们提供的管脚约束表即可完成管脚绑定。

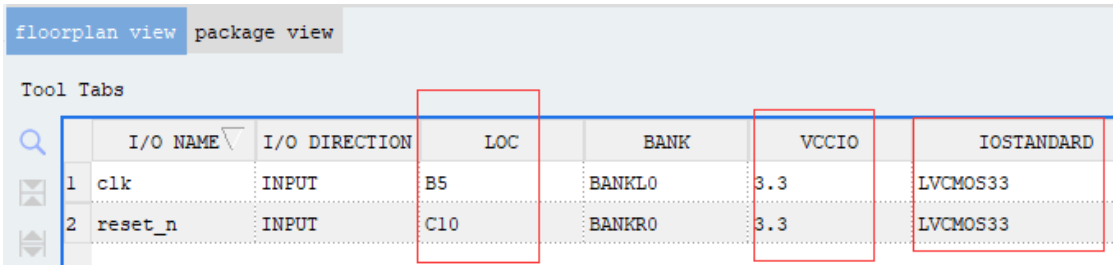

图 10-19 管脚分配页面

这样管脚约束添加完成了。但是此时约束内容保存在内存中，还没有写入文件，点击工具栏保存按钮(图 10-20 中箭头指向的位置)，或者直接点击 Ctrl+S。

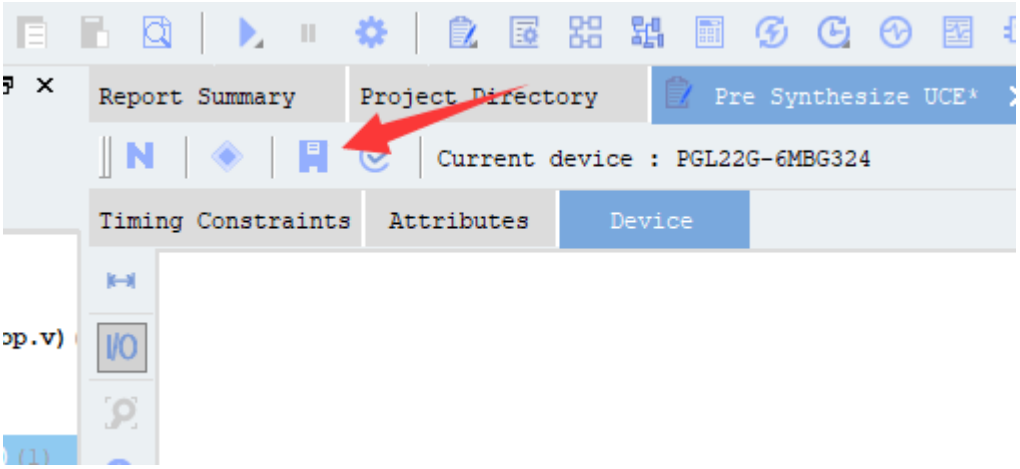

图 10-20 保存文件

将约束文件命名为 fifo 后就可以在 Source 窗口的 Constraints 下可找到刚保

存的 fifo.fdc 文件。双击可以打开约束文件。

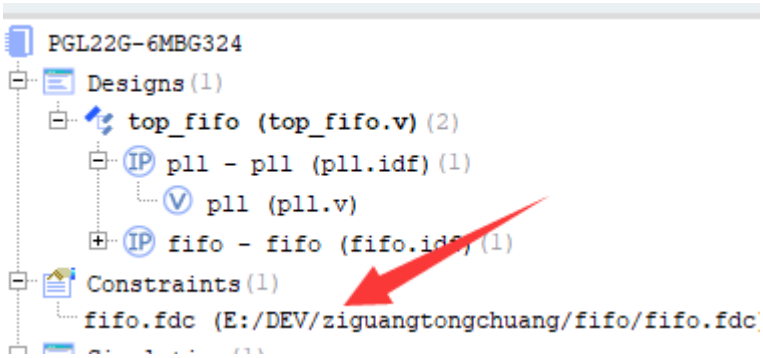

图 10-21 Source 面板

10.4.2下载验证

新建 DebugCore，将 wr_full，almost_full，wr_data_count，almost_empty，rd_data_count，rd_empty, wr_en 以及 rd_en 这些信号添加至观察列表中，新建 DebugCore 核的方法这里不再赘述。

编译工程并生成比特流.sbit 文件后，此时将下载器一端连接电脑，另一端与开发板上的 JTAG 下载口连接，连接电源线，并打开开发板的电源开关。

点击 PDS 工具栏的下载按钮，在弹出的 Fabric Configuration 界面中双击“Boundary Scan”，我们将生成好的 sbit 流文件下载到开发板中去。

在线调试配置界面将 wr_en 信号设置为高电平触发，然后点击图 10-22 箭头指向的触发按钮。

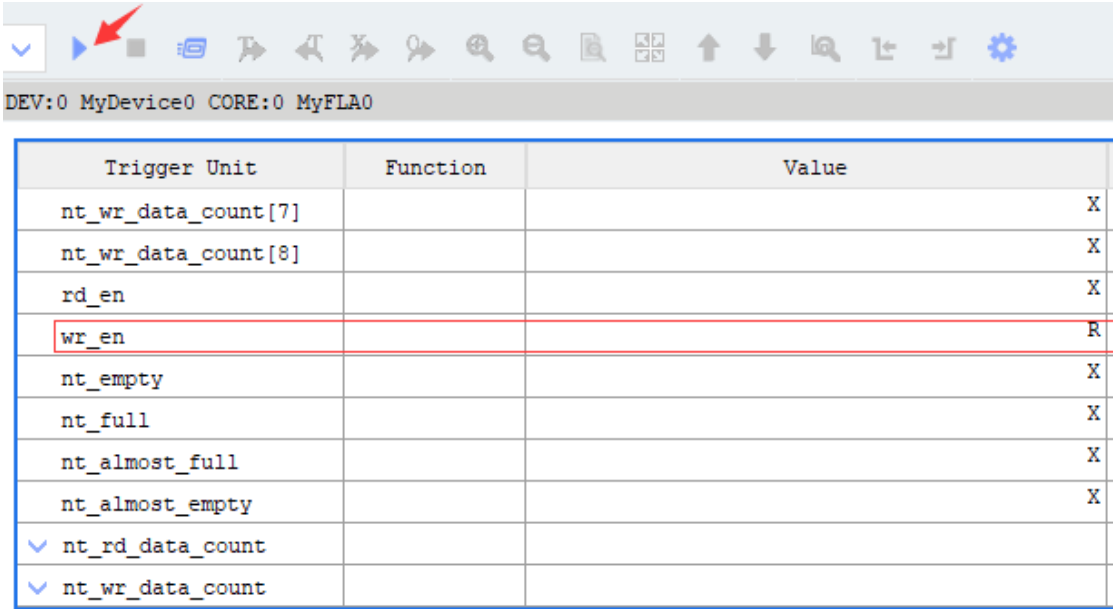

图 10-22 设置触发信号

点击图 10-22 中的触发按钮后，等待触发时按下开发板的复位按键，可以看到如图 10-23 所示一张整体的在线调试运行图。

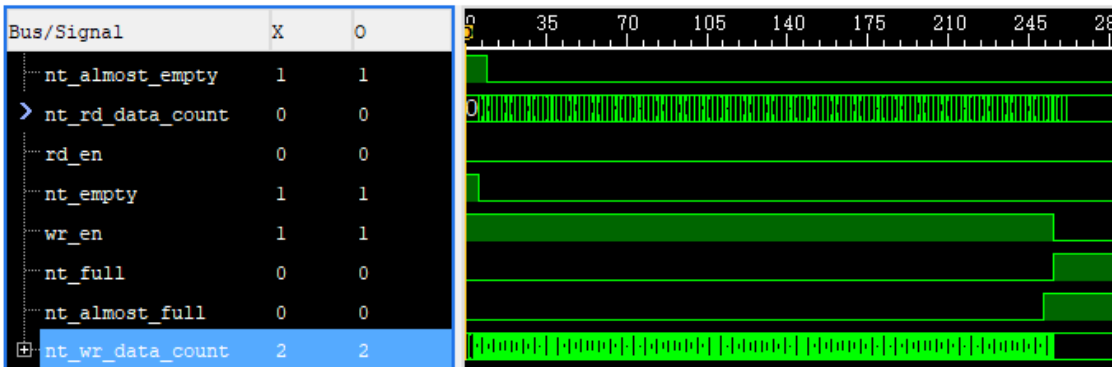

图 10-23 仿真波形全貌

对写数据部分放大后的波形，如图 10-24 所示

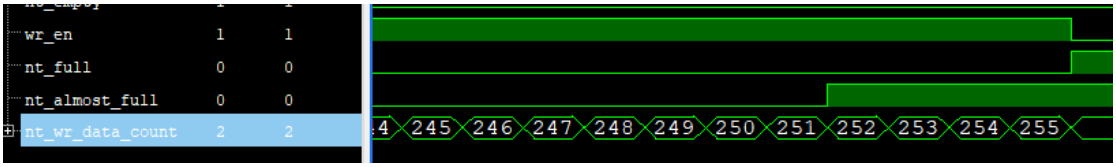

图 10-24 数据写入放大的波形

当写第 252 个数据后，almost_full 信号开始拉高，这与图 10-6 中的 FIFO IP 设置一致，当数据写满 255 个数据后，wr_full 也开始拉高。

对读数据部分放大后的波形，如图 10-25 所示

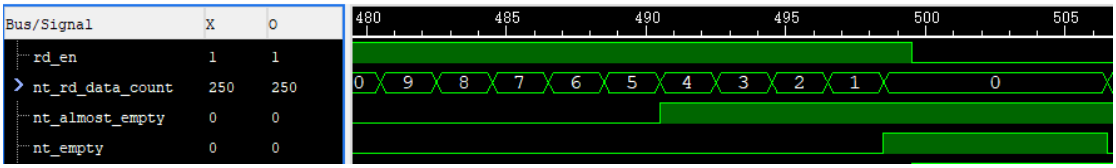

图 10-25 数据读出放大的波形

当读完 251 个数据后，almost_empty 信号就被拉高，这是因为读完 251 个数据后，FIFO 还剩余 4 个可读数据，与图 10-6 中设置一致；当读完 255 个数据后，FIFO 还剩余 0 个可读数据，rd_empty 信号也被拉高。

我们可以发现，上图中的数据变化同样和 PDS 仿真的波形是一致的。本次实验的 IP 核之 FIFO 读写实验验证成功。

10.5常见问题说明

1. 对于一个完整的 FPGA 工程开发流程，仿真是一个重要而必不可少的步

骤；

2. FIFO IP 中使用了 GTP_GRS 是一个全局复位模块，在对涉及到 FIFO IP 代码进行仿真时，要在仿真代码里面对 GTP_GRS 模块进行例化；

10.6总结

本章介绍了 FIFO 的分类、区别以及相关概念，通过调用 IP 核的方式实现了一个双时钟 FIFO。本章属于理论学习与上板实验相结合，建议读者能够跟随本实验内容，完整的进行整个实验。