版本信息：
版本
REV2019
时间
04/01/2019

Kintex FPGA 修炼秘籍 DDR MIG (AXI4)
MK7325FA

电子版自学资料

常州一二三电子科技有限公司
溧阳米联电子科技有限公司

米联客(MSXBO)04QQ 群：516869816
米联客(MSXBO)03QQ 群：543731097
米联客(MSXBO)02QQ 群：86730608
米联客(MSXBO)01QQ 群：34215299

版本	时间	描述
Rev2019	2019-04-01	首次更新，更新 AXI 自定义 MSXBO_FDMA 关于 DDR 读写测试，视频缓存方案的 4 个例子。

感谢您使用米联客 Kintex 系列开发板，以及配套教程。本教程使用自定义AXI4 IP MSXBO_FDMA对DDR进行读写控制。MSXBO_FDMA是基于AXI4封装的DMA控制器，可以非常方便地用于FPGA的MIG控制器和ZYNQ上并实现对ZYNQ PS 或者PL的DDR进行读写控制。和官方的DMA以及VDMA相比，FDMA具备无需驱动程序，只要会FPGA就能读写DDR的简单方便性。

软件版本：VIVADO2017.4

免费获取资料、答疑解惑到米联客(MSXBO) 官方论坛 www.osrc.cn

扫描以下二维码注册论坛：www.osrc.cn

获取最新产品发布、资料更新、技术焦点 关注 米联客 MSXBO 微信公众号

微信公众平台：米联客 MSXBO

目录

CH01 基于 FDMA 内存读写测试

软件版本:VIVADO2017.4
操作系统:WIN10
硬件平台:MK7325

1.1 概述

FDMA 是 MSXBO(米联客的)基于 AXI4 总线协议定制的一个 DMA 控制器。有了这个 IP 我们可以统一实现用 FPGA 代码直接读写 PL 的 DDR 或者 ZYNQ PS 的 DDR。

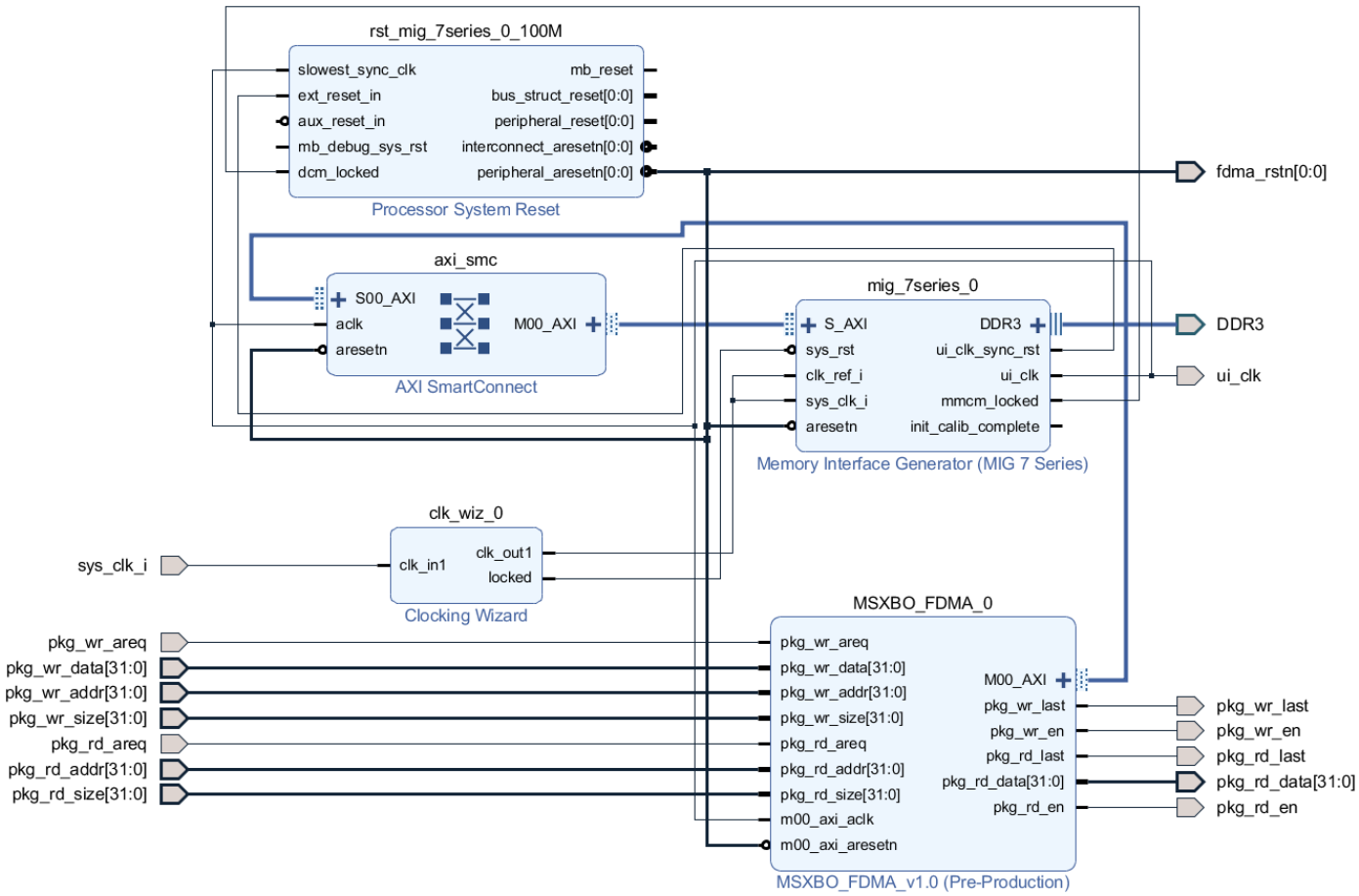
如果用过 ZYNQ 的都知道，要直接操作 PS 的 DDR 通常是 DMA 或者 VDMA,然而用过 XILINX 的 DMA IP 和 VDMA IP，总有一种遗憾，那就是不够灵活，还需要对寄存器配置，真是麻烦。对于我们搞 FPGA 的人来说，最喜欢直接了当，直接用 FPGA 代码搞定。现在 XILINX 的总线接口是 AXI4 总线，那么熟练自定义 AXI4 IP 挂到总线上就非常方便了。基于这个目的，本小编定义了一个基于 AXI4 FULL MASTER 的 IP，暂且取名为 MSXBO_FDMA。

通过这个 IP 我们可以方便地进行 AXI4 FULL MASTER 的操作，比如我们经常要读写 DDR，那么只要挂到 AXI4 总线上就可以利用这个 IP 实现。

以下是小编展示了一种在 XC7K325 FPGA 上的读写测试方案。

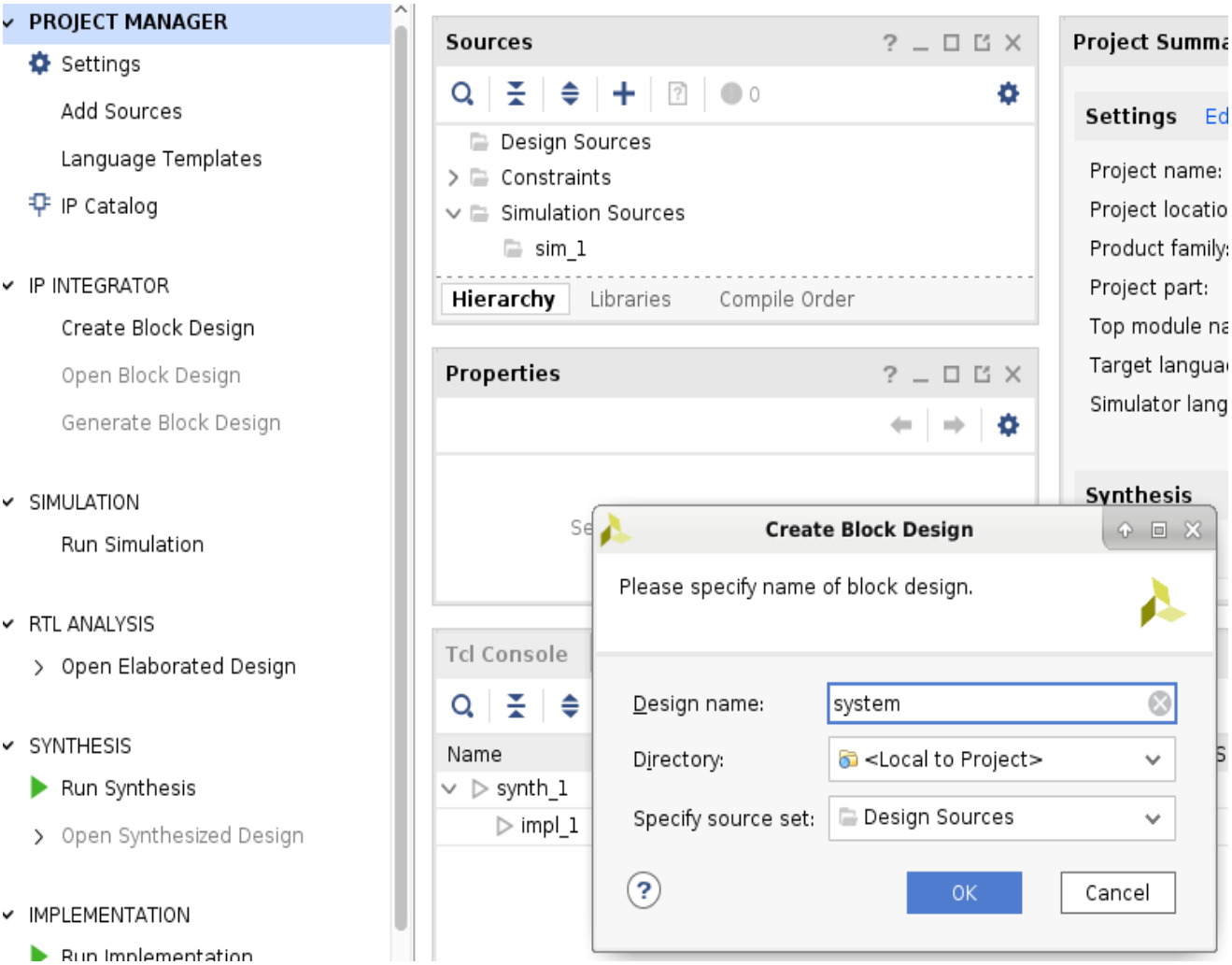
1.2 基于 FDMA 搭建的 BD 工程

首先看下基于 FDMA 搭建的 BD 工程，这里使用的是 PL 的 DDR 因此使用 MIG 控制器。

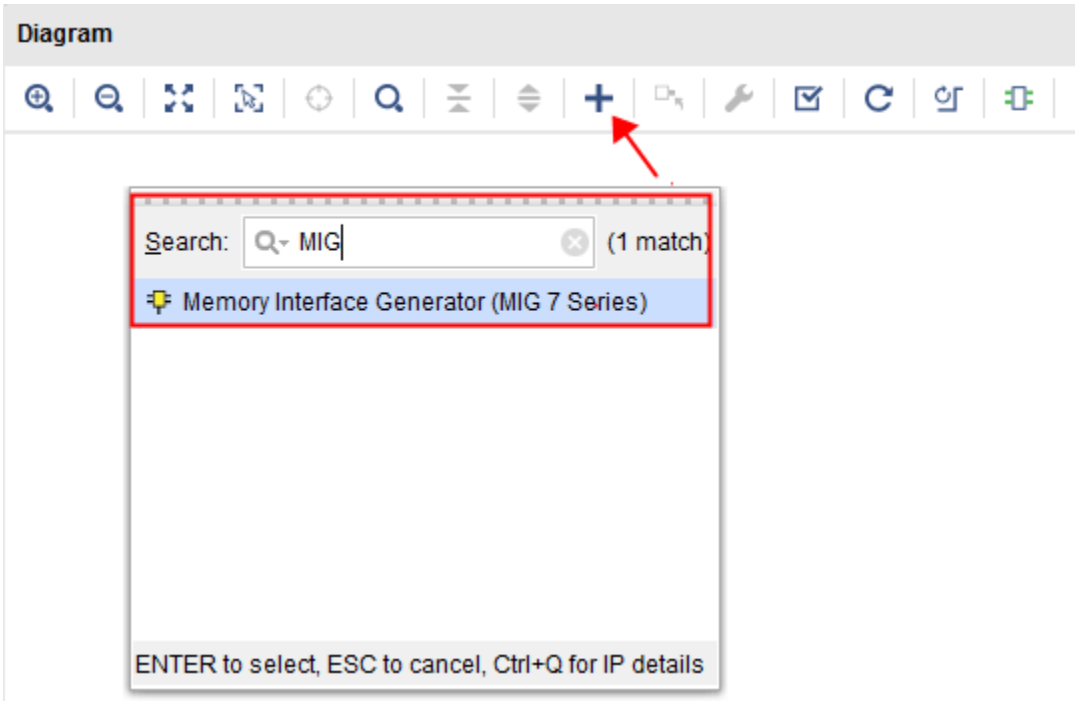


1.3Setp By Step 搭建 FPGA BD 工程

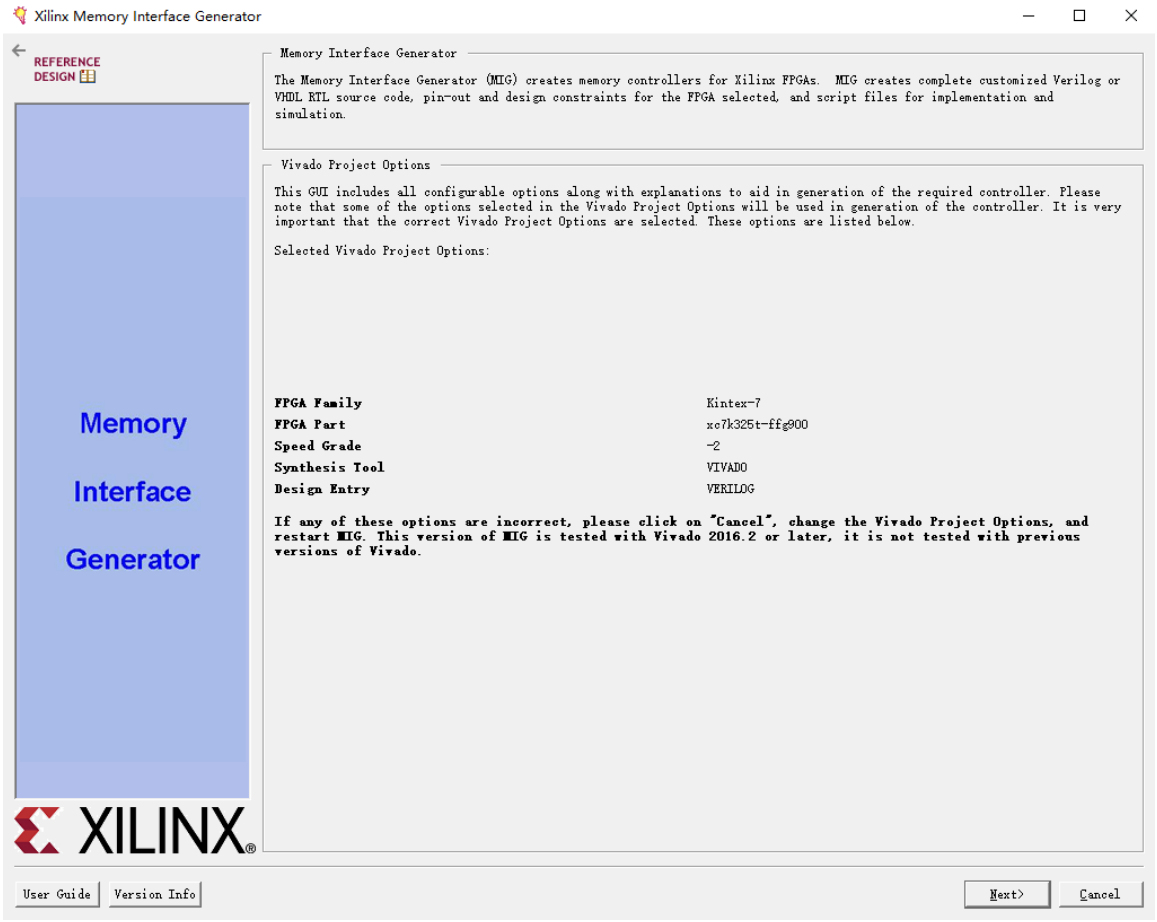
Step1:任意创建一个新的空的 BD 工程，如下图，选择 Create Block Design ， BD 的 Design name 命名为 system



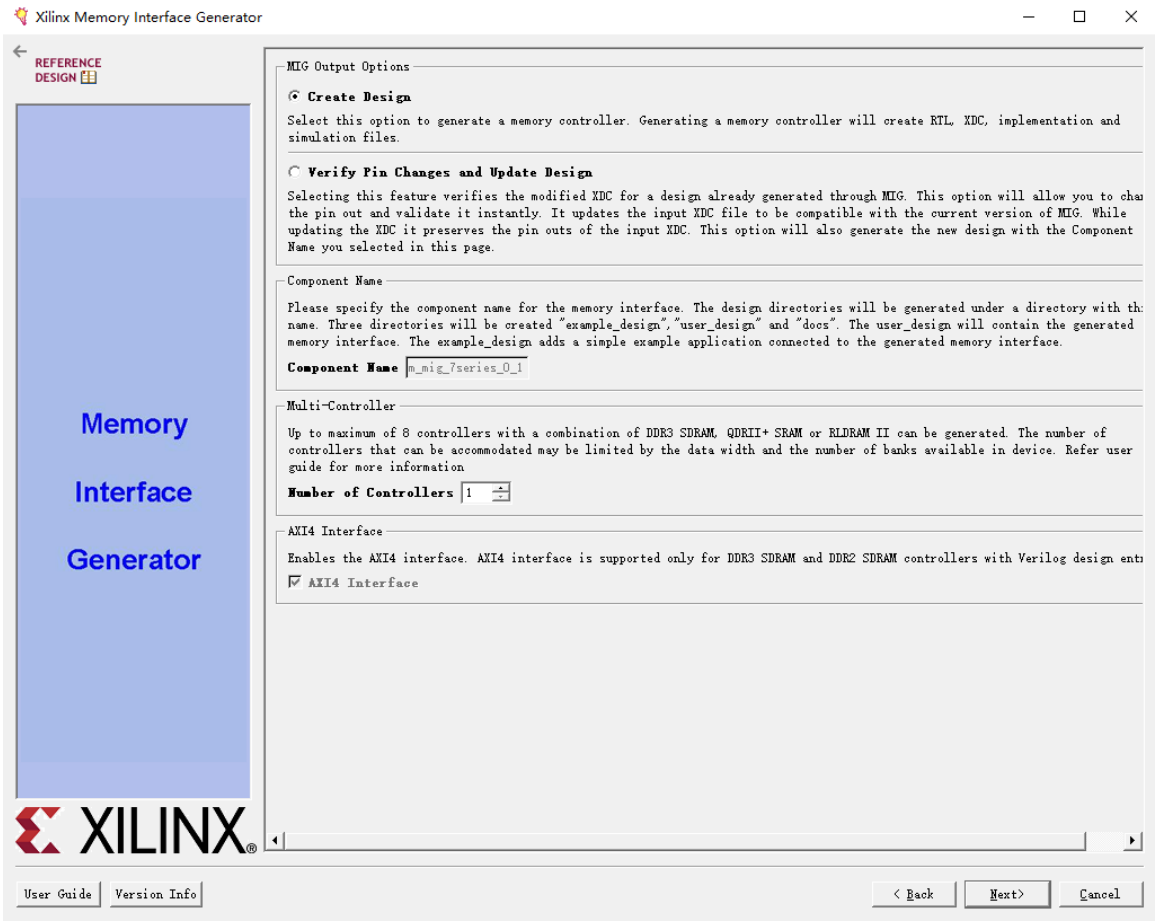
Step2:选择下图箭头所指的□，输入关键词 MIG 双击添加并且配置 MIG



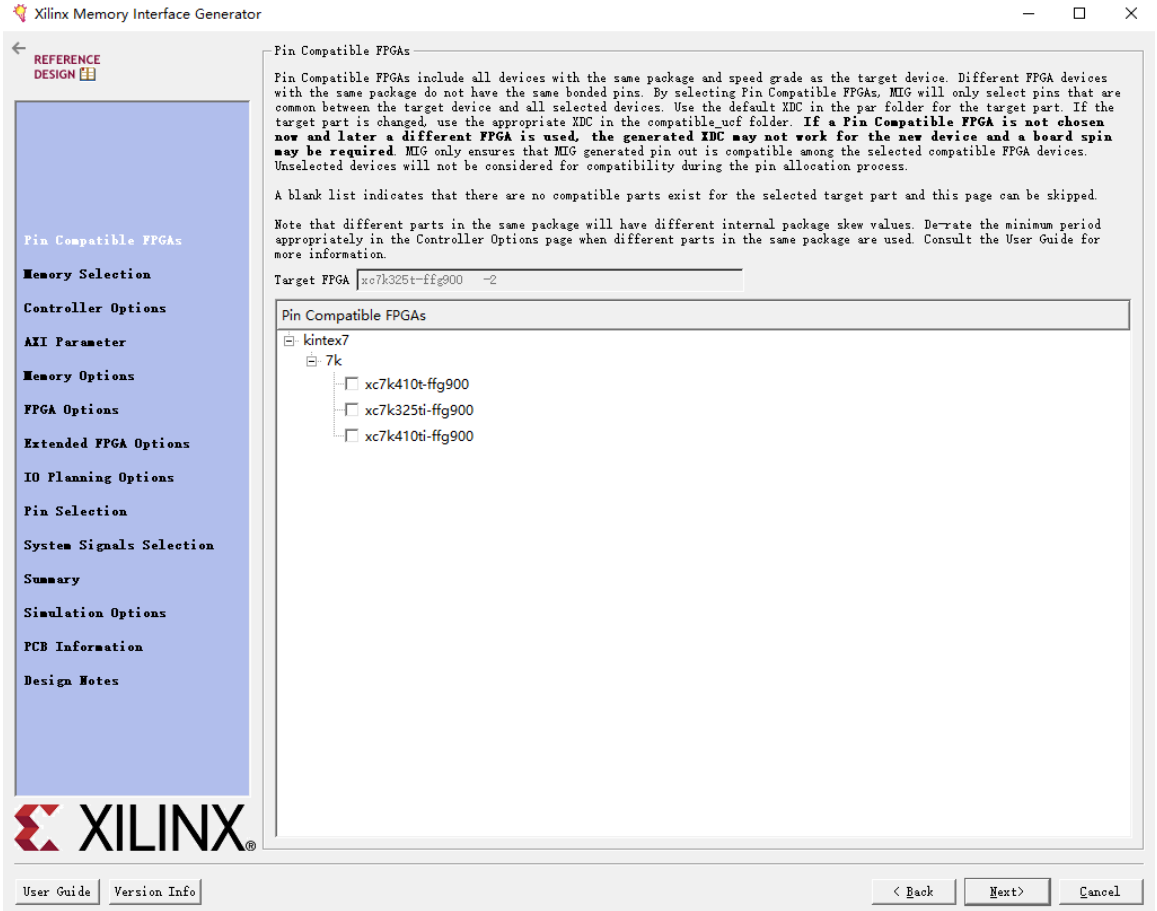
单击 NEXT



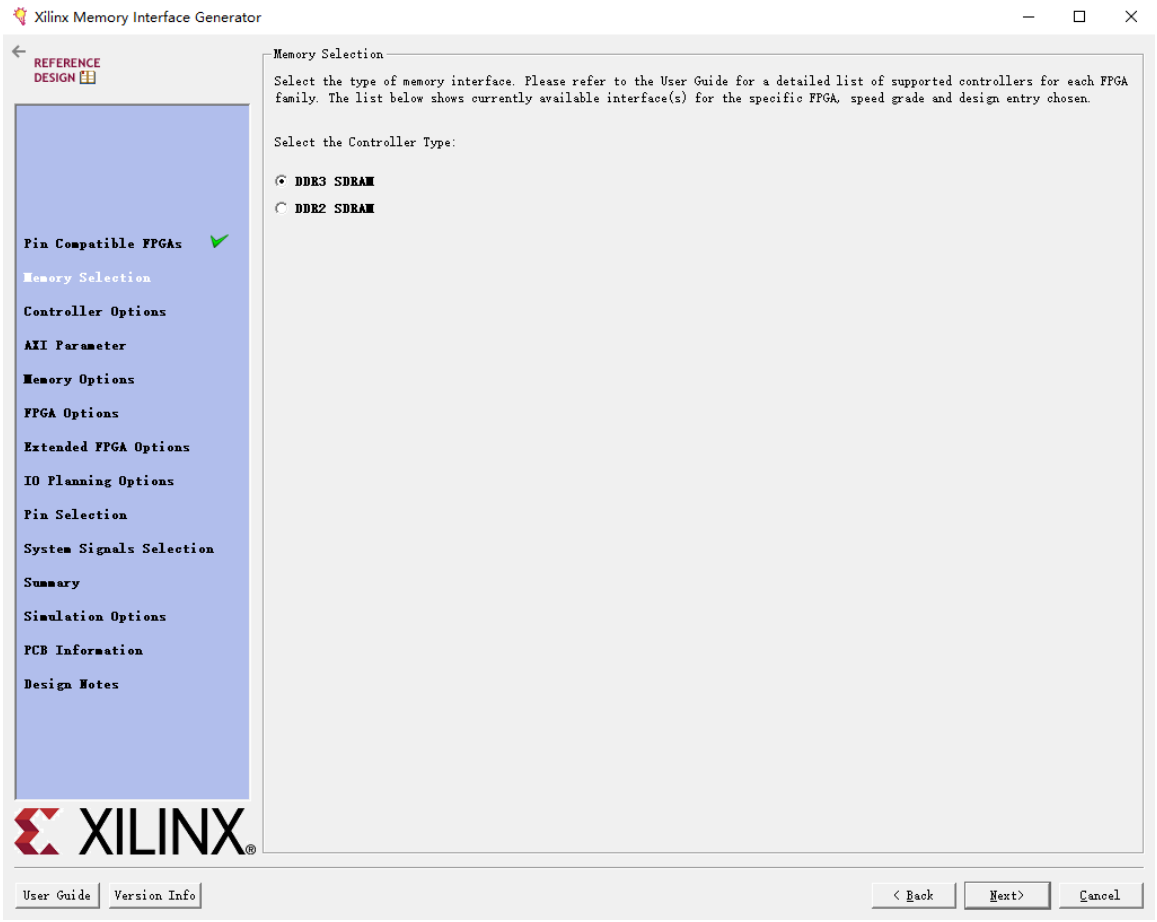
选择 Create Design 单击 NEXT



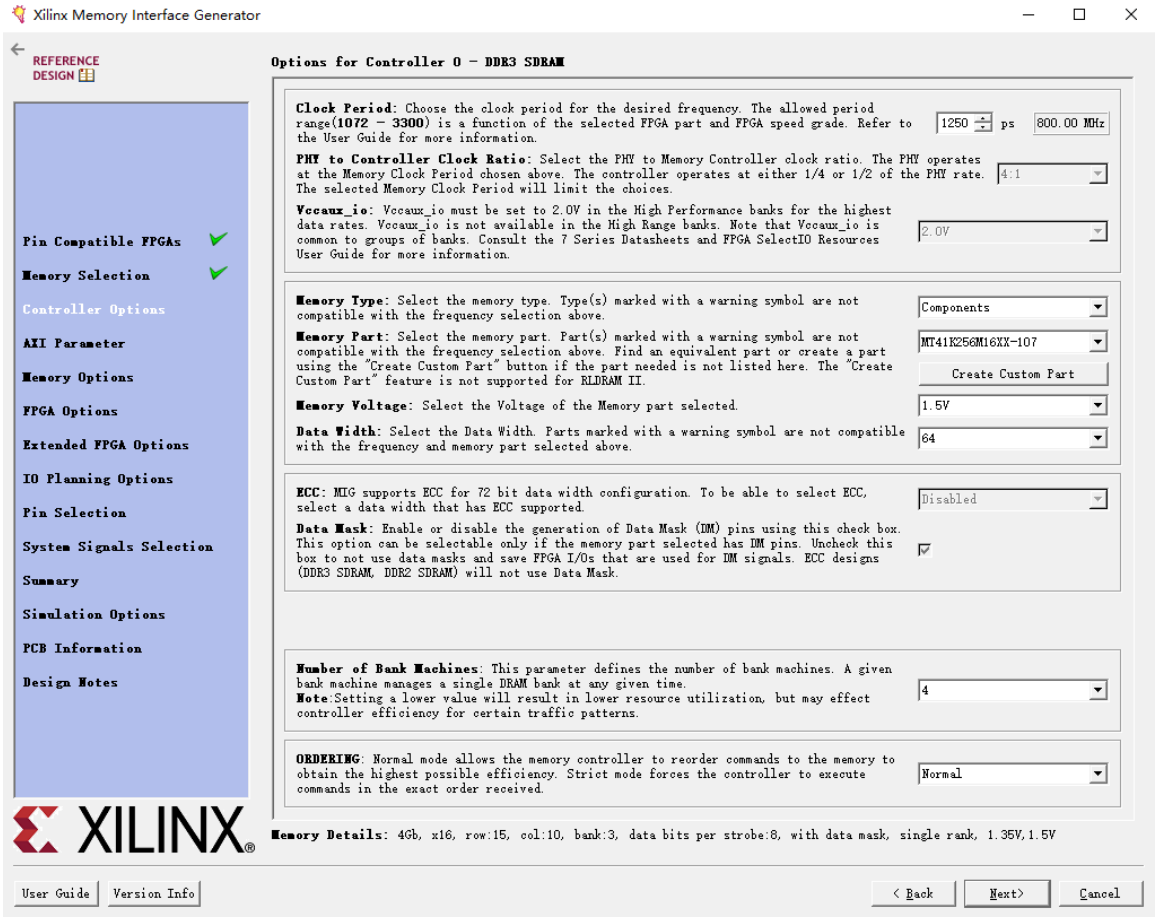
继续单击 NEXT



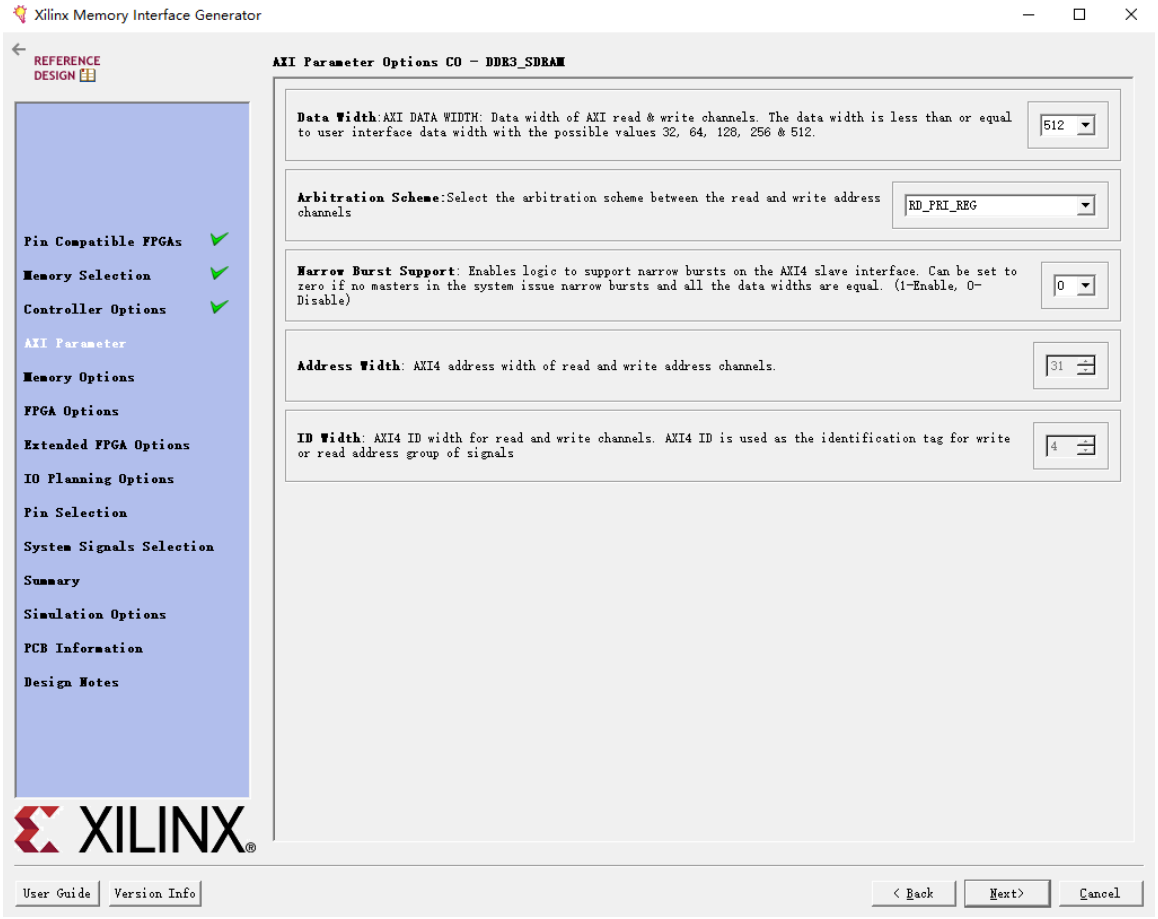
选择 DDR3 单击 NEXT



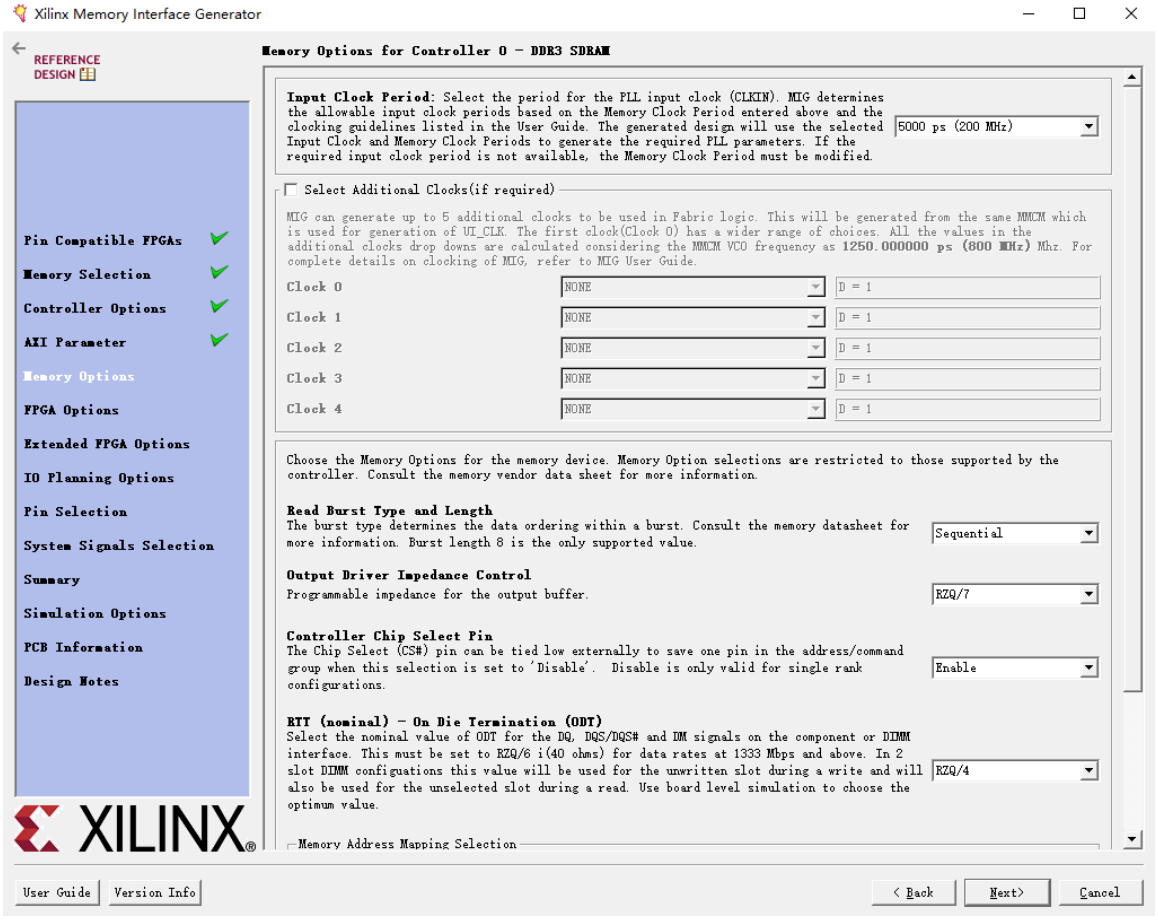
设置 MIG 内核时钟频率、内存型号、内存的数据位宽



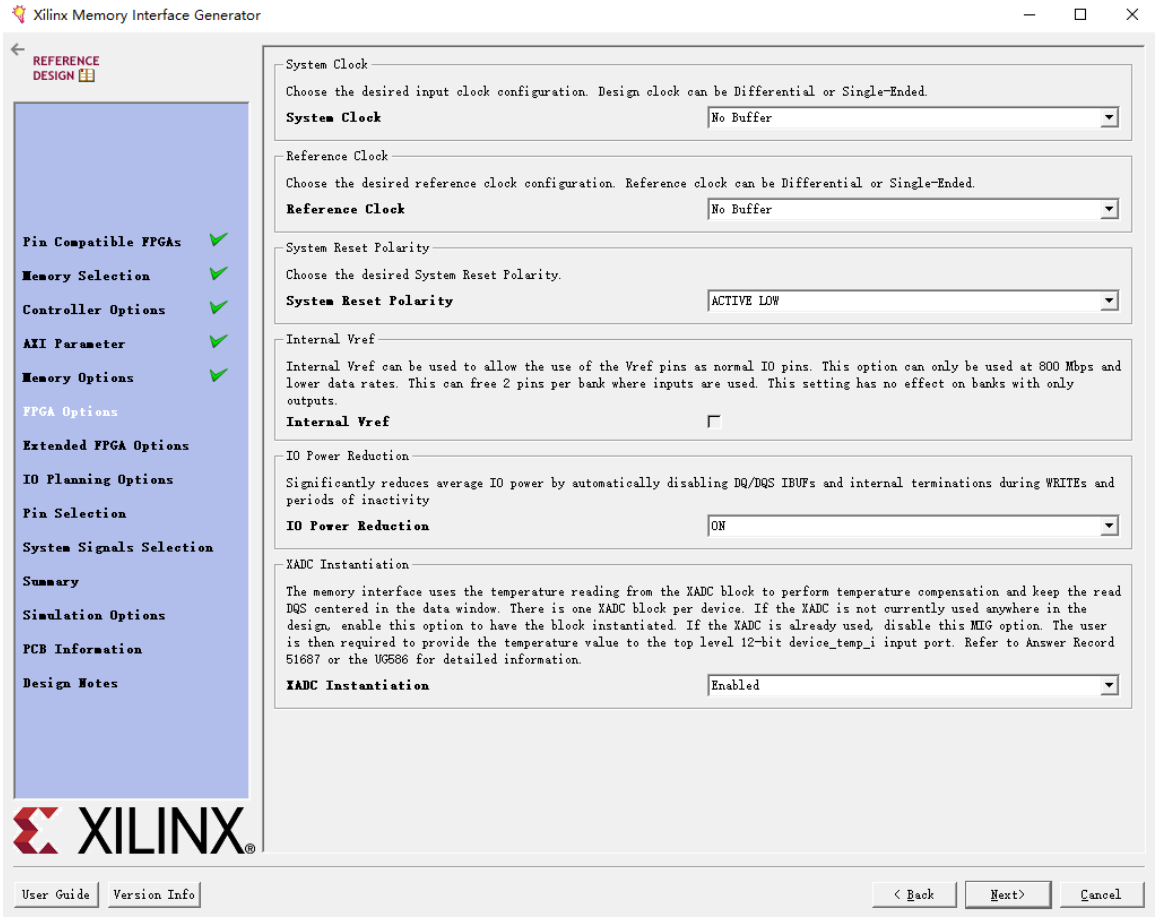
设置 MIG AXI4 最大支持的位宽，对于 4 片 DDR 最大位宽为 512bit



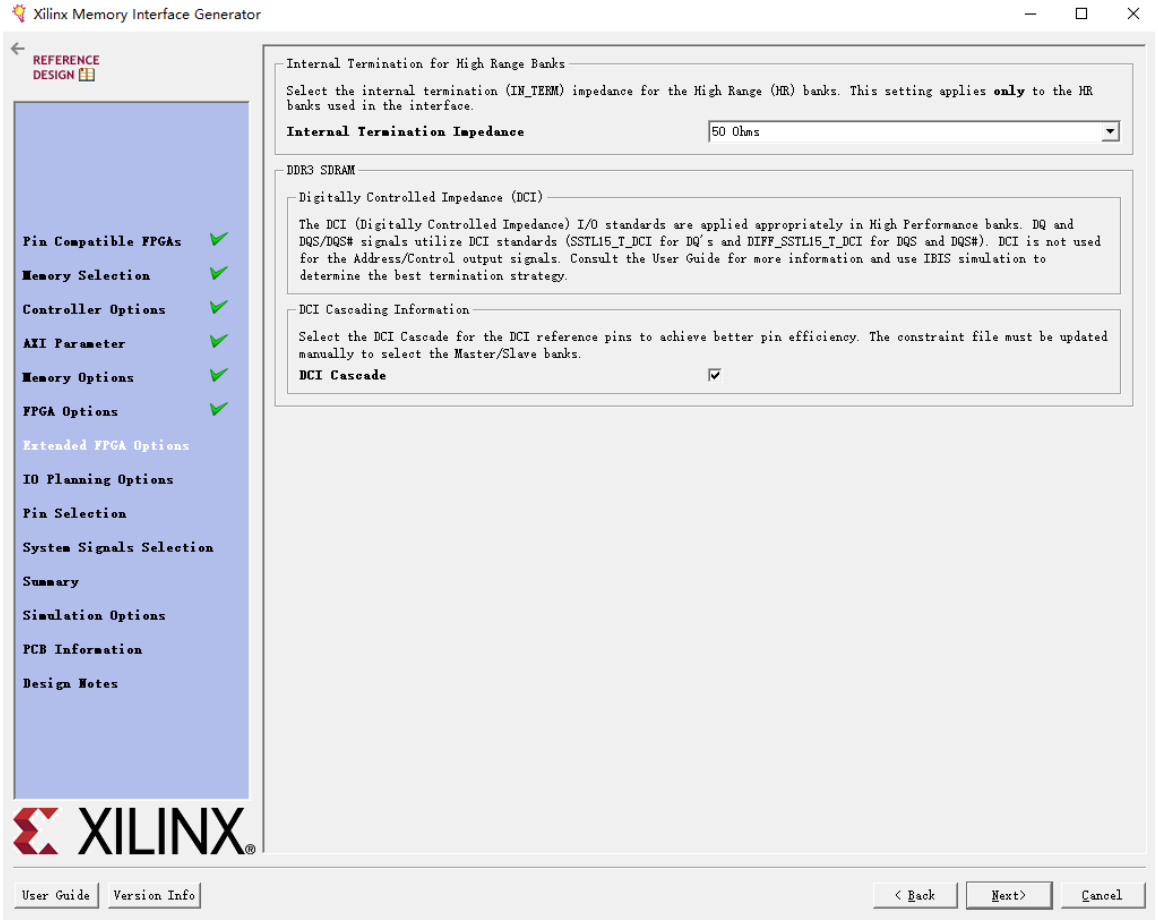
设置输入频率为 200M



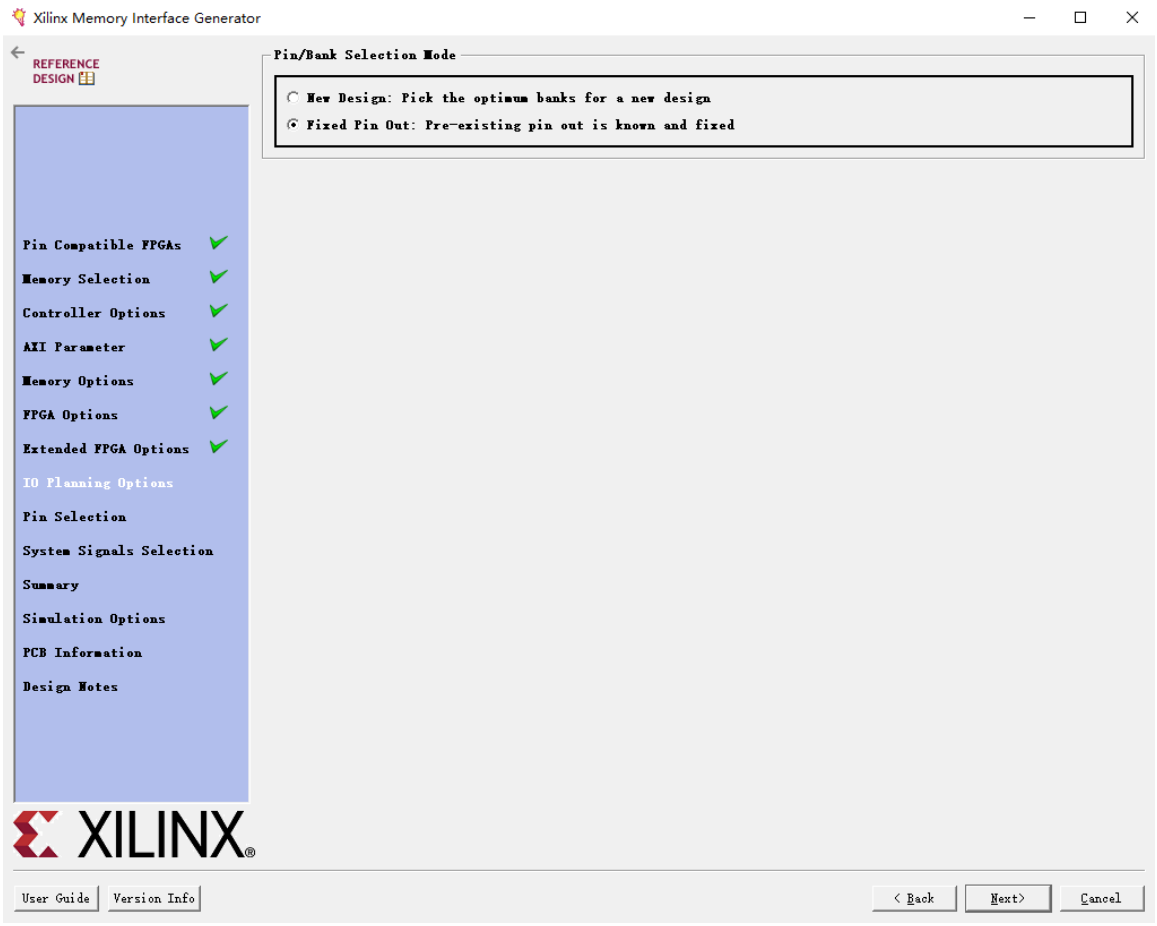
系统和参考时钟时钟选择 no buffer,MIG 低电平复位



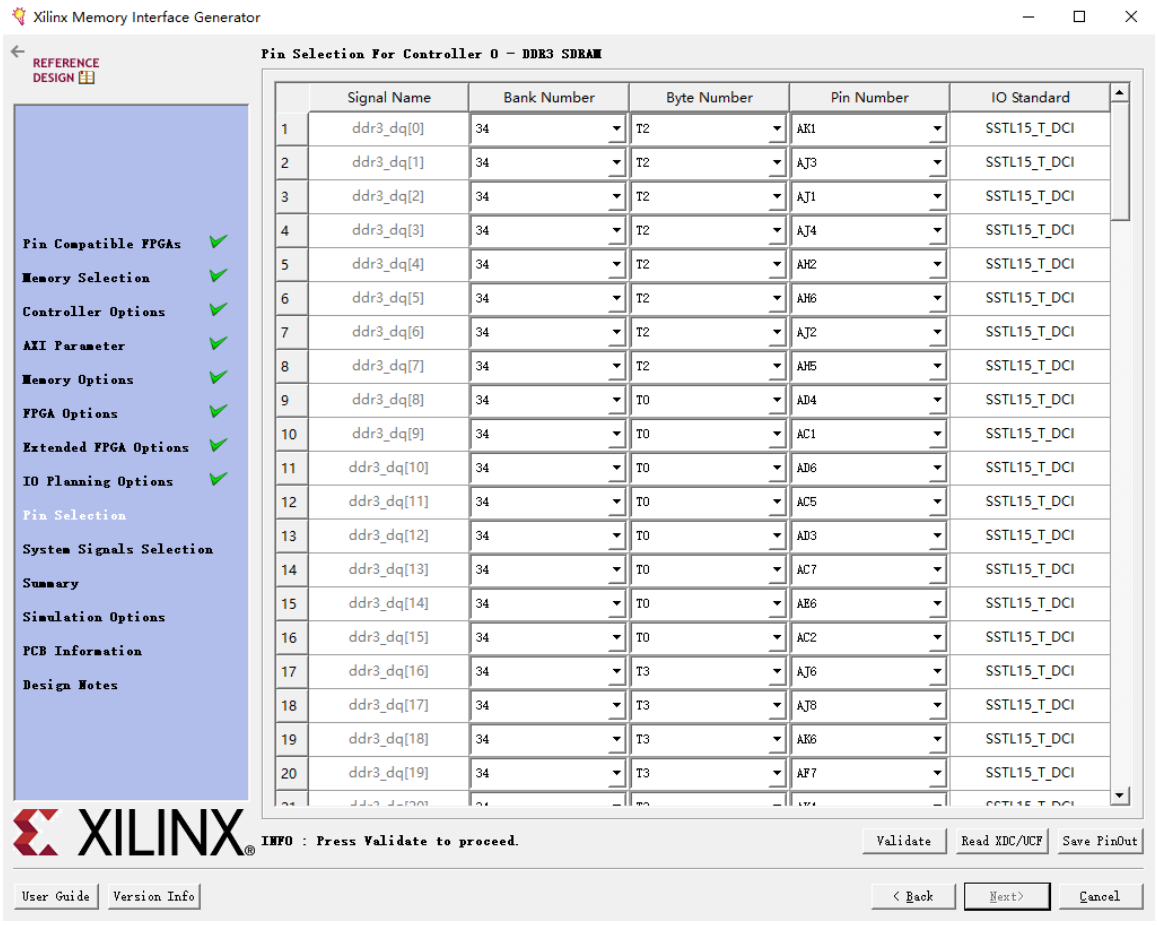
Step9:终端阻抗选择 50hms



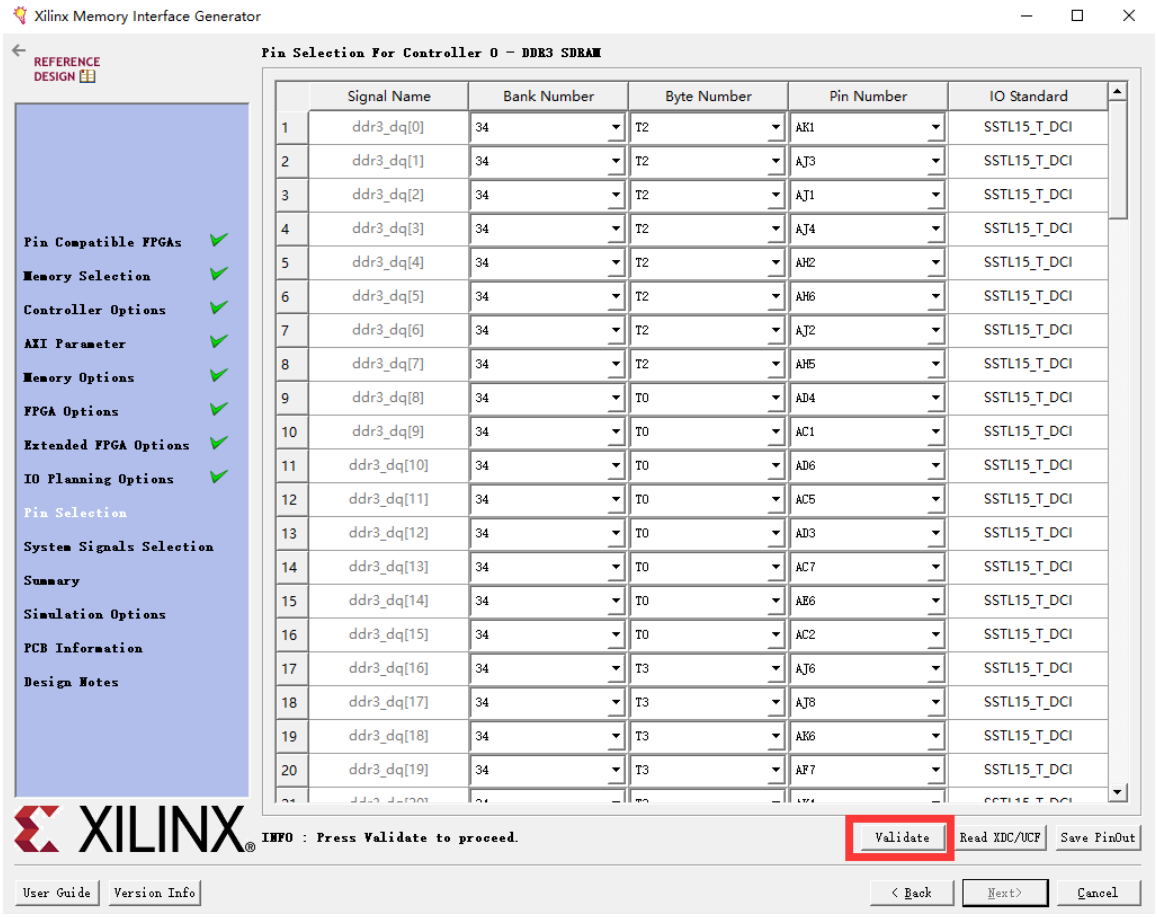
选择 Fixed Pin Out

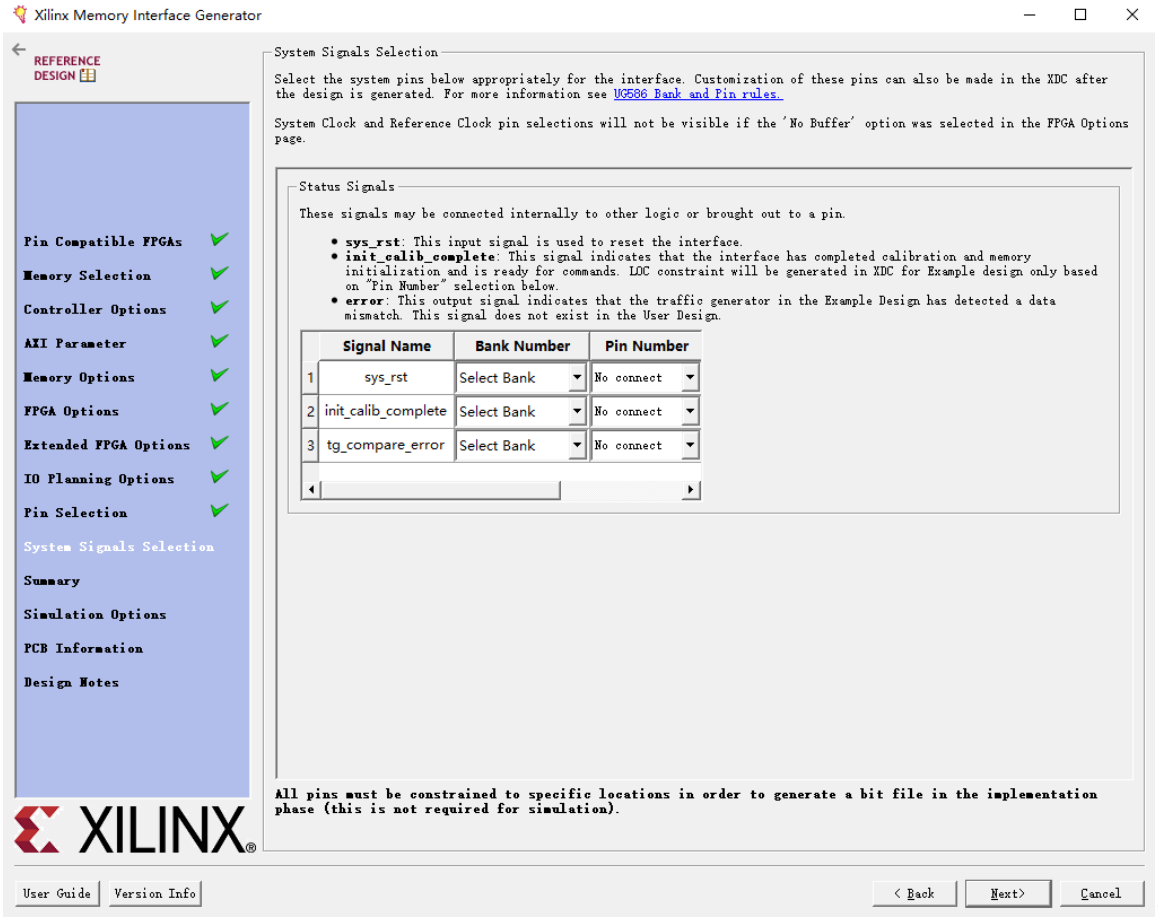


根据原理图手动填写 PIN 脚定义,或者选择 Read XDC/UCF 直接读入 pin 脚定义,本课程下提供了 MA703FA-100T.ucf 的 DDR 配置文件，直接读入既可。



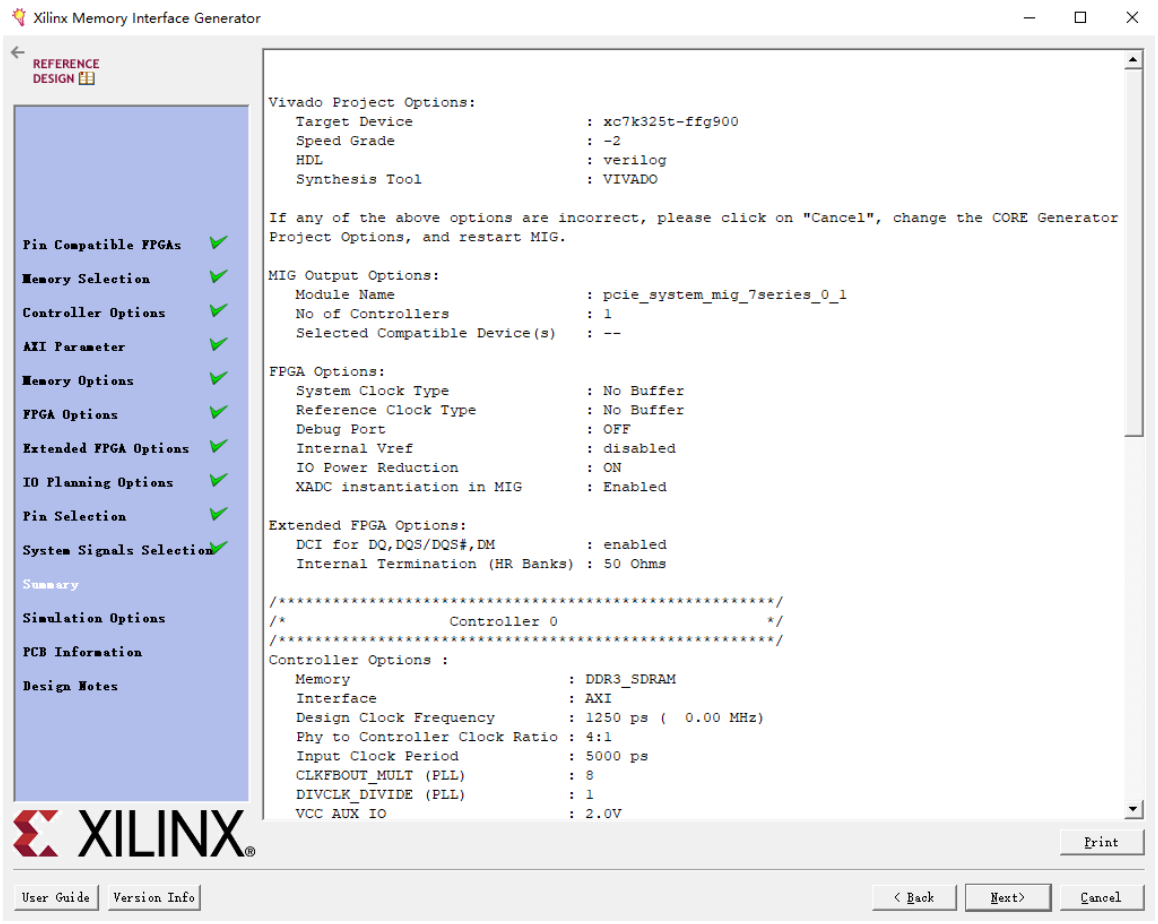
填写完成后，先单击 Validate 再单击 NEXT



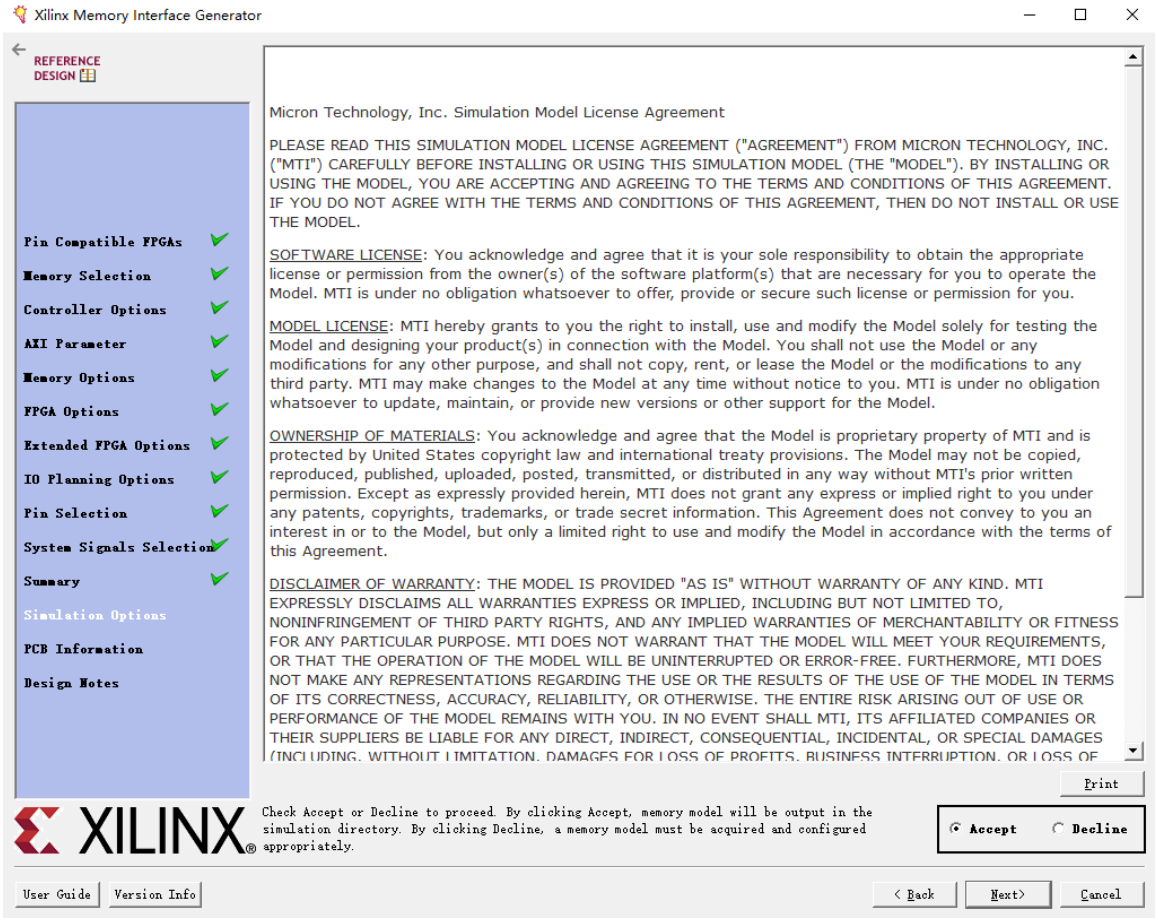


继续单击 NEXT

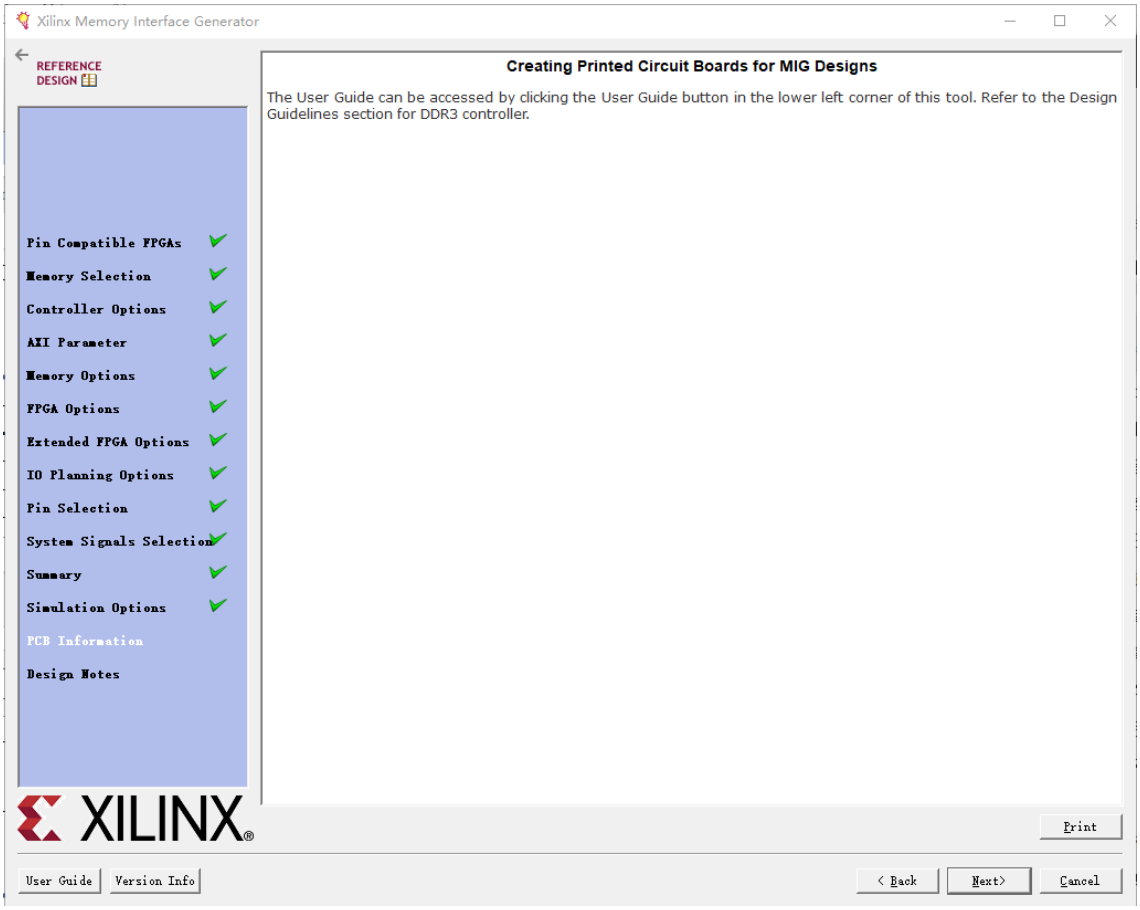
继续单击 NEXT



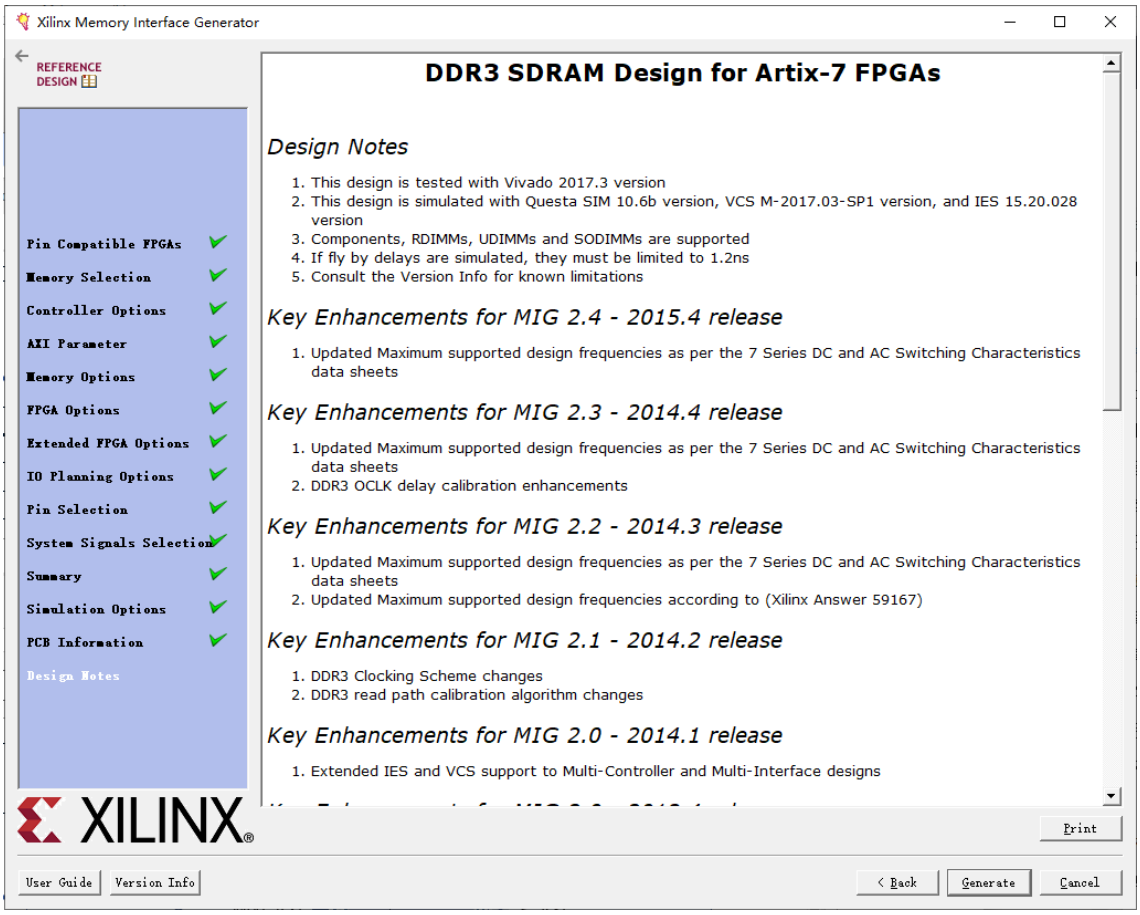
继续单击 NEXT

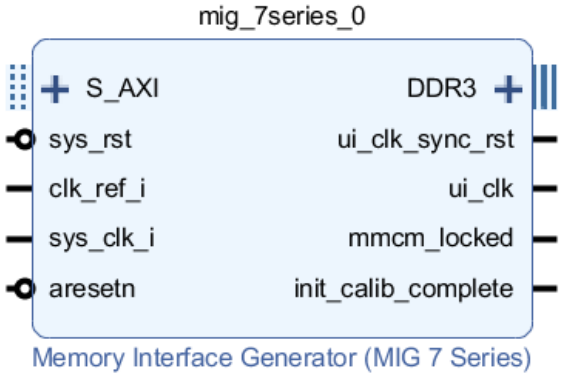


继续单击 NEXT

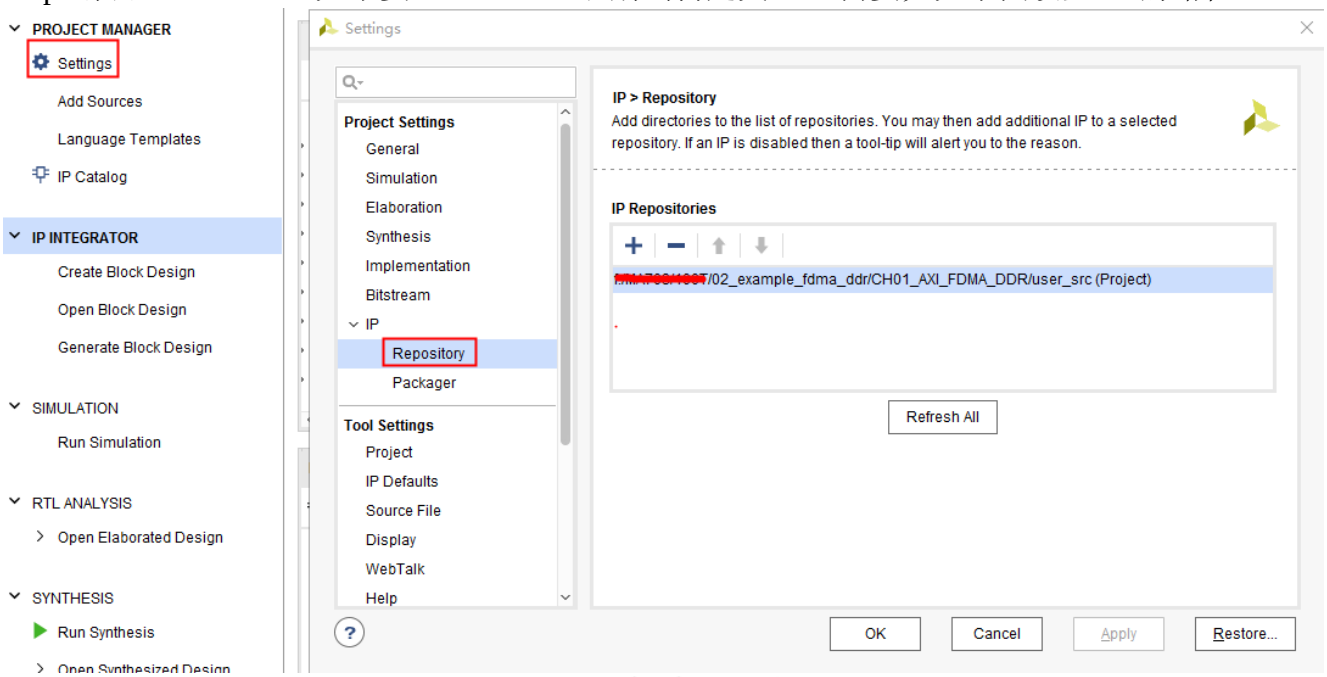


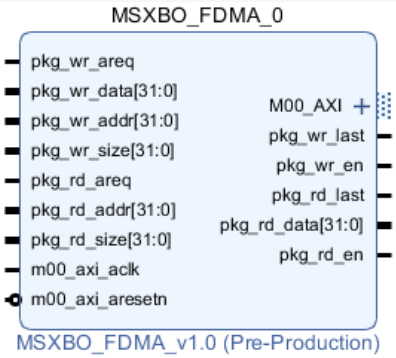
最后单击 Generate，至此 MIG 的配置完成

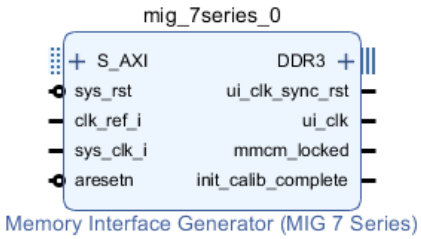




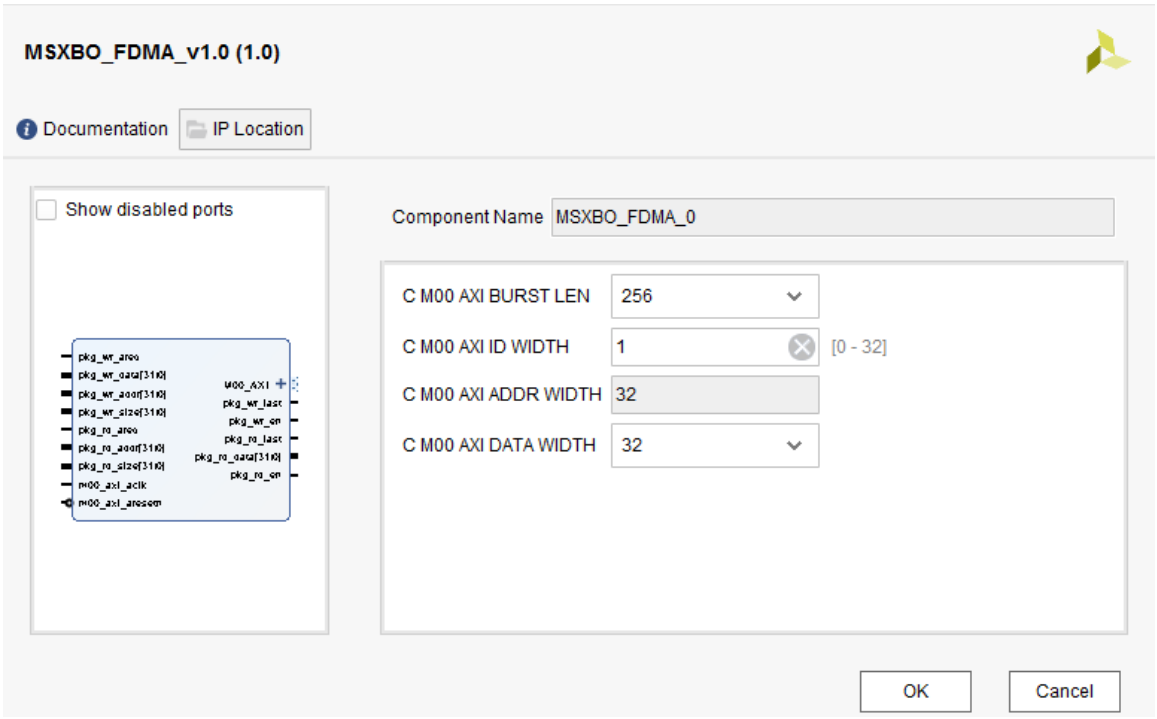
Step3:添加 FDMA IP，如果要让 VIVADO 识别到自定义 IP，需要先如下图设置 IP 的路径



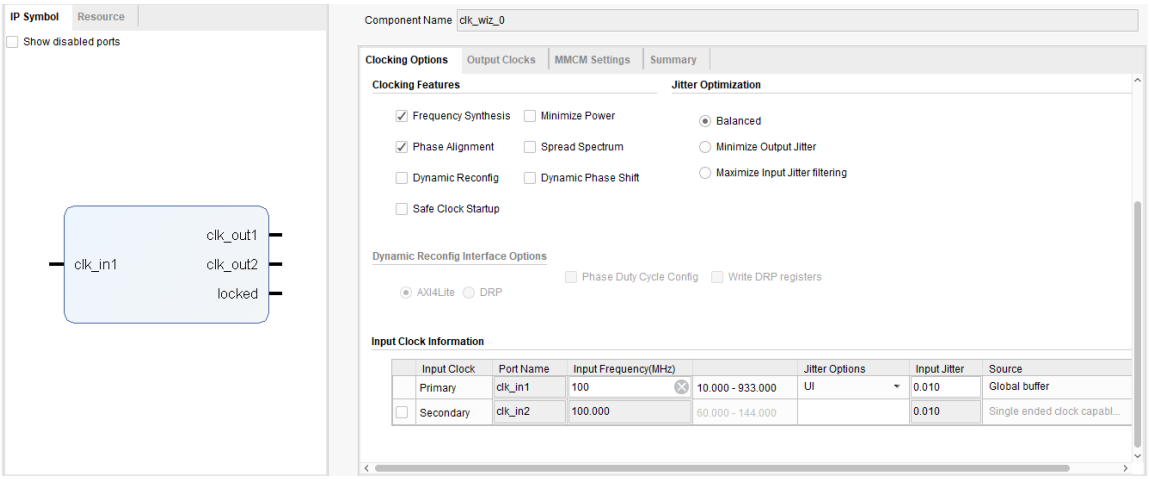




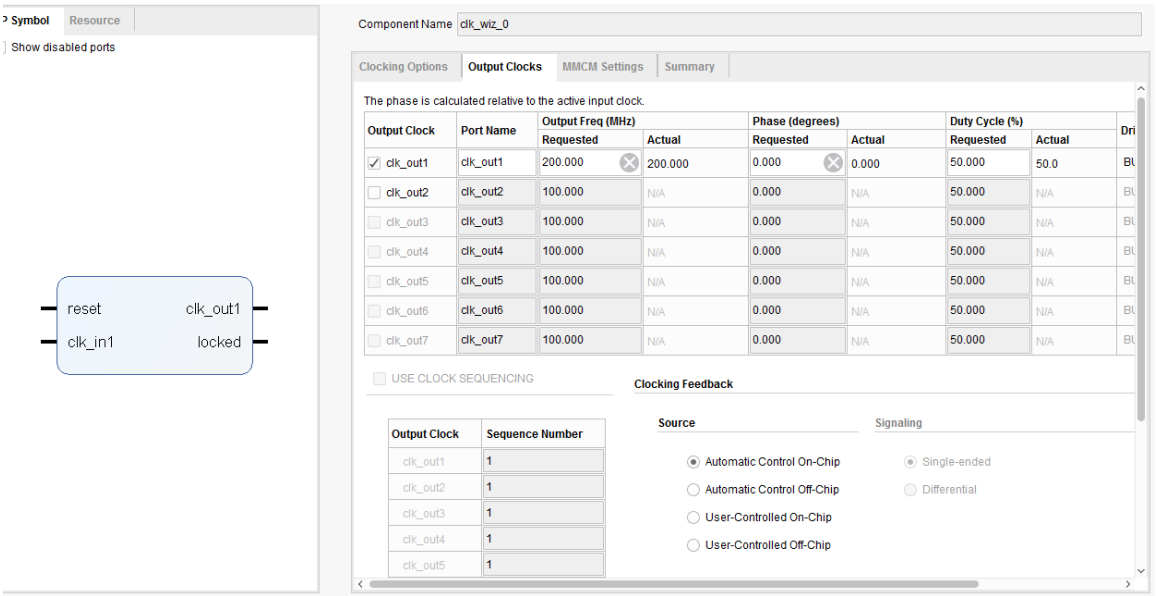
修改 FDMA IP 参数，修改 AXI BURST LEN 为 256，其他参数默认。修改一下参数需要确保一次 AXI BURST 的大小不超过 4KB 比如 当 AXI BURST LEN 为 256 数据位宽为 32bit 那么 256*32=4096bytes 已经是一次标准的 AXI 最大传输的大小了。

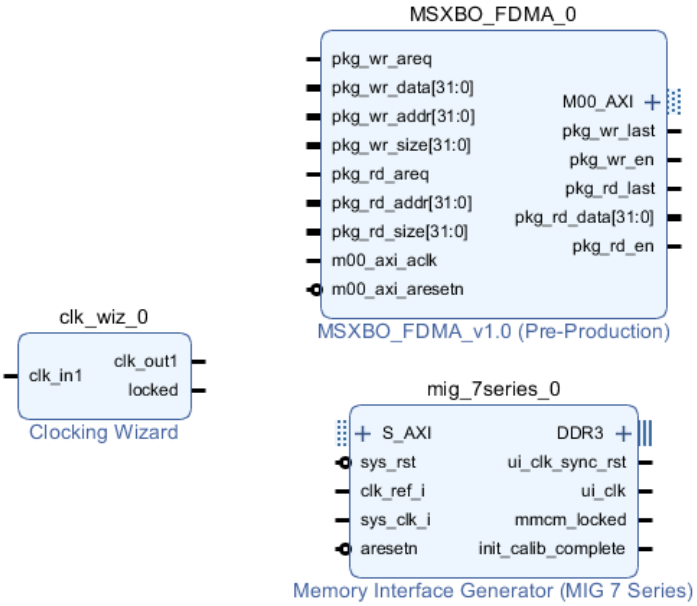


Step4:添加时钟 IP

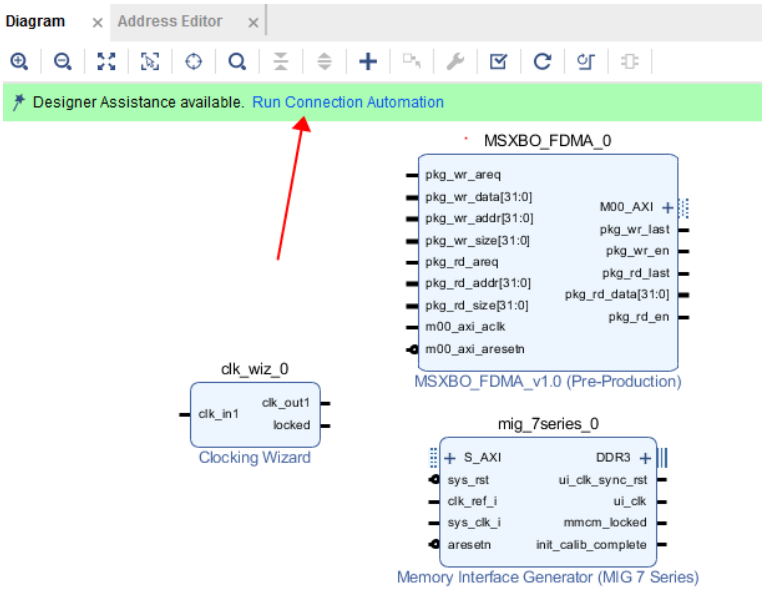


以下配置的 200M 时钟输出用于 MIG 时钟的输入

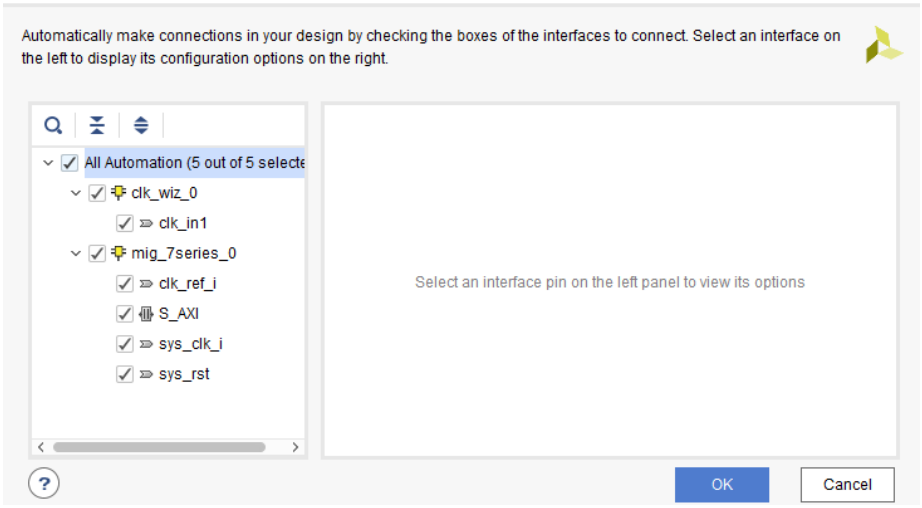


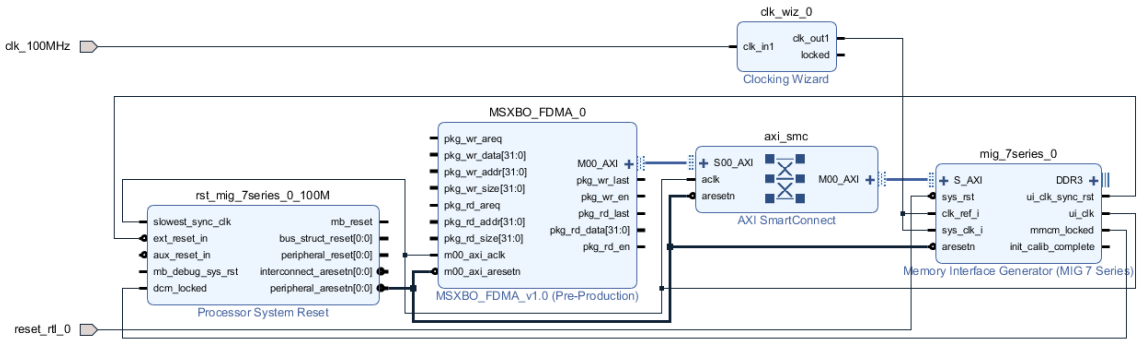


Step4:采用自动连线

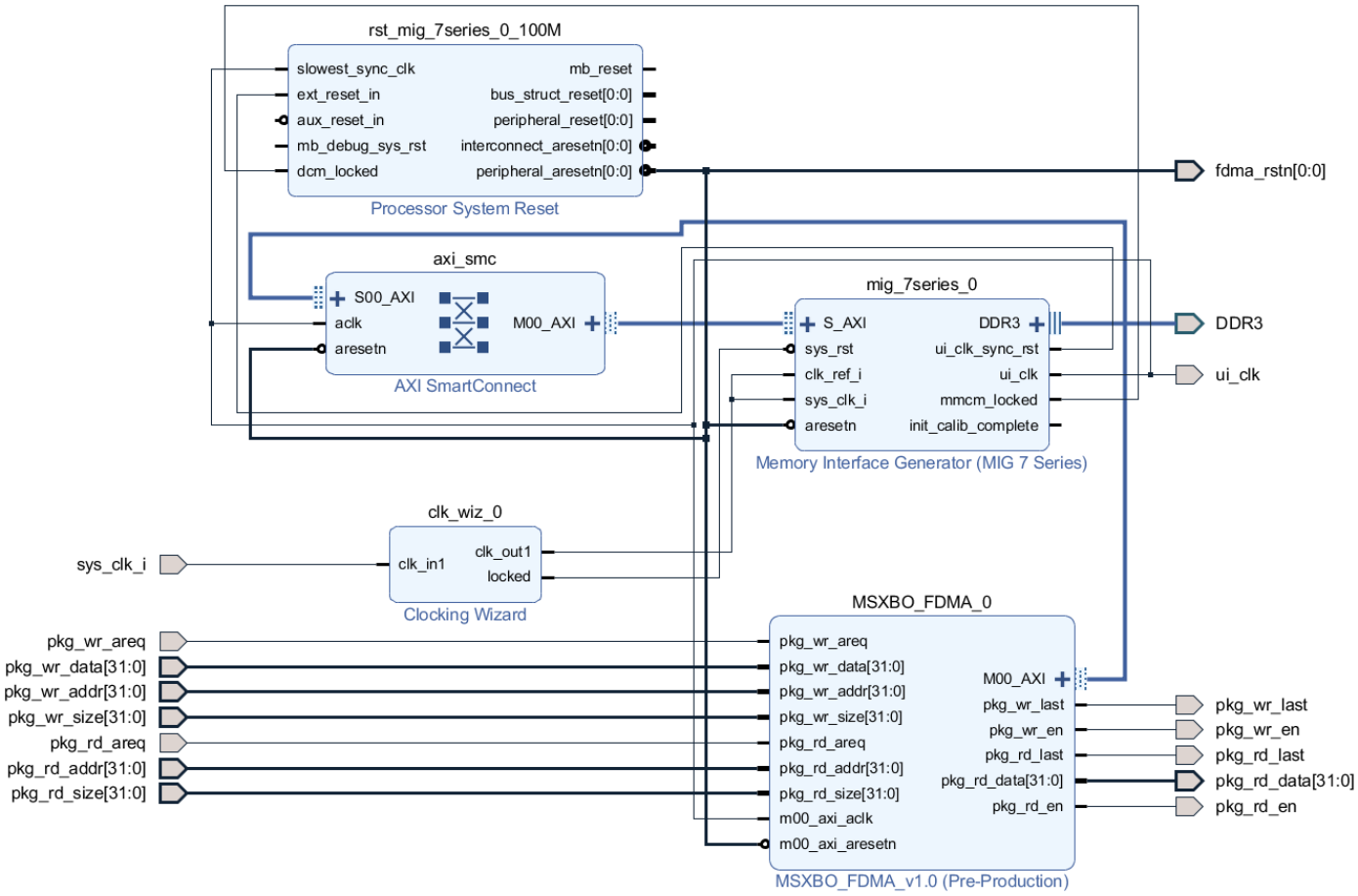


全部勾选默认(也可以有选择勾选，当然也可以不用自动连线，全部手动连线)





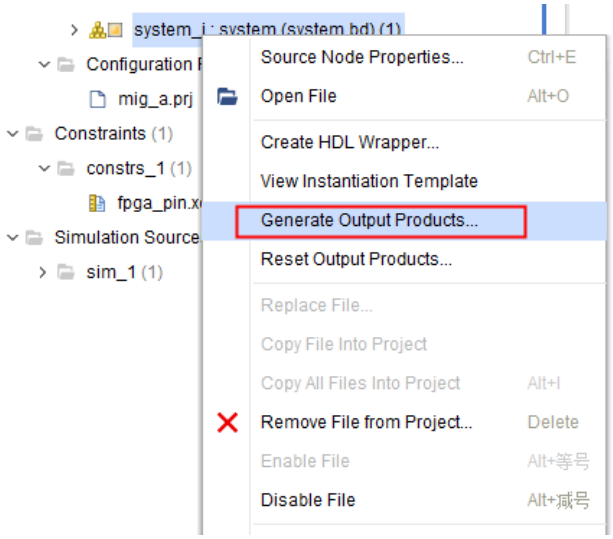
Step5:手动修改连线最终 BD 设计如下



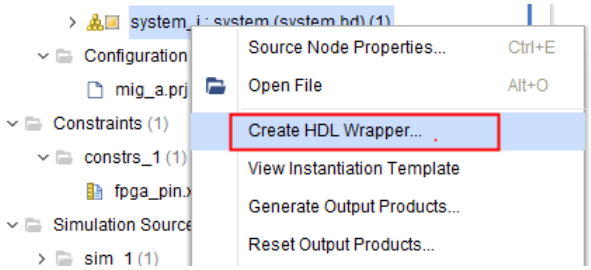
Step6:最后 MIG 地址默认的起始地址为 0x8000_0000,改为 0X0000_0000

1.4 编写 FDMA 测试代码

首先右击 BD 并且单击



其次继续右击 BD 文件，选择 Create HDL Wrapper



最后修改 Wrapper 并且添加测试代码，测试代码如下

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:CZ123 MSXBO www.osrc.cn
// Engineer: tjy
// Create Date: 2019/04/02 12:39:25
// Design Name:
// Module Name: fdma_top
// Project Name: AXI_FDMA
// Target Devices:
// Tool Versions: VIVADO
// Description: test DDR
// Dependencies:
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//////////////////////////////////////////////////////////////////////////////////
module fdma_top(
output [13:0]DDR3_addr,
output [2:0]DDR3_ba,
output DDR3_cas_n,
output [0:0]DDR3_ck_n,
output [0:0]DDR3_ck_p,
output [0:0]DDR3_cke,
output [0:0]DDR3_cs_n,
output [3:0]DDR3_dm,
inout [31:0]DDR3_dq,
```

```
inout [3:0]DDR3_dqs_n,
inout [3:0]DDR3_dqs_p,
output [0:0]DDR3_odt,
output DDR3_ras_n,
output DDR3_reset_n,
output DDR3_we_n,
input sys_clk_i
);

  wire [0:0]ui_rstn;
  wire ui_clk;
  //----------------fmda signals---------------------------------
  wire [31:0] pkg_wr_addr;
(*mark_debug = "true"*)    wire    [31:0] pkg_wr_data;
(*mark_debug = "true"*) (* KEEP = "TRUE" *)    reg                pkg_wr_areq;
(*mark_debug = "true"*)    wire               pkg_wr_en;
(*mark_debug = "true"*)    wire               pkg_wr_last;
  wire [31:0] pkg_wr_size;
  wire [31:0] pkg_rd_addr;
(*mark_debug = "true"*)    wire    [31:0] pkg_rd_data;
(*mark_debug = "true"*) (* KEEP = "TRUE" *) reg                pkg_rd_areq;
(*mark_debug = "true"*)    wire               pkg_rd_en;
(*mark_debug = "true"*)    wire               pkg_rd_last;
  wire [31:0] pkg_rd_size;
//-------------------------------------------------------------
  reg [31:0]pkg_wr_cnt;
(*mark_debug = "true"*) (* KEEP = "TRUE" *) reg [31:0]pkg_rd_cnt;
(*mark_debug = "true"*) (* KEEP = "TRUE" *) reg [1:0] T_S;

  reg [31:0] pkg_addr;

  parameter WRITE1 = 0;
  parameter WRITE2 = 1;
  parameter READ1  = 2;
  parameter READ2  = 3;
  //----------------
  assign pkg_wr_size = 1024;
  assign pkg_rd_size = 1024;

  assign pkg_wr_data = pkg_wr_cnt;
  (*mark_debug = "true"*) wire test_error;
  assign test_error = (pkg_rd_en && (pkg_rd_cnt != pkg_rd_data));

  assign pkg_wr_addr = pkg_addr;
  assign pkg_rd_addr = pkg_addr;

  always @(posedge ui_clk)
  begin
```

```
    if(!ui_rstn)begin
        T_S <=0;
        pkg_wr_areq <= 1'b0;
        pkg_rd_areq <= 1'b0;
        pkg_wr_cnt<=0;
        pkg_rd_cnt<=0;
        pkg_addr<=0;
    end
    else begin
        case(T_S)
        WRITE1:begin
            if(pkg_addr>=32'd536870911) pkg_addr<=0;
            pkg_wr_areq    <= 1'b1;
            T_S <= WRITE2;
        end
        WRITE2:begin
            pkg_wr_areq    <= 1'b0;
            if(pkg_wr_last) begin
                T_S <= READ1;
                pkg_wr_cnt <= 32'd0;
            end
            else if(pkg_wr_en) begin
                pkg_wr_cnt <= pkg_wr_cnt + 1'b1;
            end
        end
        READ1:begin
            pkg_rd_areq <= 1'b1;
            T_S <= READ2;
        end
        READ2:begin
            pkg_rd_areq <= 1'b0;
            if(pkg_rd_last) begin
                T_S <= WRITE1;
                pkg_addr <= pkg_addr + 4096;
                pkg_rd_cnt <= 32'd0;
            end
            else if(pkg_rd_en) begin
                pkg_rd_cnt <= pkg_rd_cnt + 1'b1;
            end
        end
        endcase
    end
end

system system_i
       (.DDR3_addr(DDR3_addr),
        .DDR3_ba(DDR3_ba),
        .DDR3_cas_n(DDR3_cas_n),
```

```
        .DDR3_ck_n(DDR3_ck_n),
        .DDR3_ck_p(DDR3_ck_p),
        .DDR3_cke(DDR3_cke),
        .DDR3_cs_n(DDR3_cs_n),
        .DDR3_dm(DDR3_dm),
        .DDR3_dq(DDR3_dq),
        .DDR3_dqs_n(DDR3_dqs_n),
        .DDR3_dqs_p(DDR3_dqs_p),
        .DDR3_odt(DDR3_odt),
        .DDR3_ras_n(DDR3_ras_n),
        .DDR3_reset_n(DDR3_reset_n),
        .DDR3_we_n(DDR3_we_n),

        .pkg_wr_addr(pkg_wr_addr),
        .pkg_wr_data(pkg_wr_data),
        .pkg_wr_areq(pkg_wr_areq),
        .pkg_wr_en   (pkg_wr_en),
        .pkg_wr_last(pkg_wr_last),
        .pkg_wr_size(pkg_wr_size),

        .pkg_rd_addr(pkg_rd_addr),
        .pkg_rd_data(pkg_rd_data),
        .pkg_rd_areq(pkg_rd_areq),
        .pkg_rd_en   (pkg_rd_en),
        .pkg_rd_last(pkg_rd_last),
        .pkg_rd_size(pkg_rd_size),

        .ui_clk(ui_clk),
        .fdma_rstn(ui_rstn),
        .sys_clk_i(sys_clk_i)
        );

endmodule
```

1.5 测试代码状态机分析

1、WRITE1 状态:为了测试整个 DDR 的存储控件，所以先计算 DDR 大小，536870911=512MB 正好是一片 DDR 的大小。根据之前 BD 里面 FDMA 的参数设置，一次 AXI4 burst 大小为 4096bytes。我们这里每次 parket 传输大小也位置为 4096bytes,那么设置 pkg_wr_size 和 pkg_rd_size 大小为 1024。pkg_wr_size 和 pkg_rd_size 的大小是以 32bit 计算。pkg_wr_size 和 pkg_rd_size 大小的设置需要注意 pkg_wr_size 需要是 FDMA 里面 AXI BURST LEN 的整数倍。

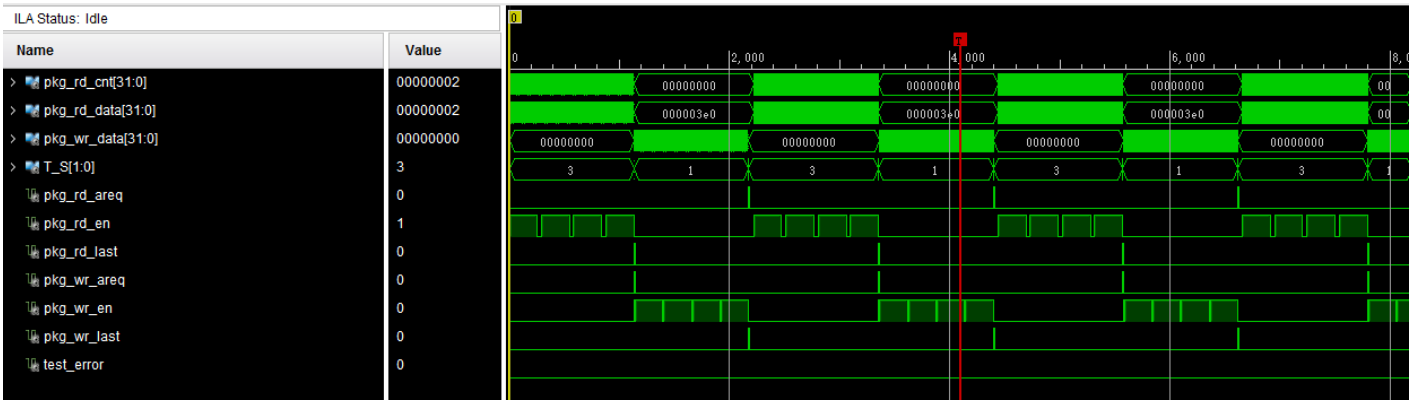
2、WRITE2 状态:WRITE1 通过设置 pkg_wr_areq 为 1 持续 1 个时钟周期，后进入 WRITE2 状态机，并且启动启动一次 FDMA 的 parket 传输。当 pkg_wr_last 为 1 的时候表示一次 parket 传输传输结束。在 WRITE2 状态机中使用了一个计数器，把计数器的值写入到 DDR 中。

3、READ1 状态:在 READ1 状态设置 pkg_rd_areq 为 1 启动一次 FMDA 的 parket read 传输，之后进入 READ2 状态。

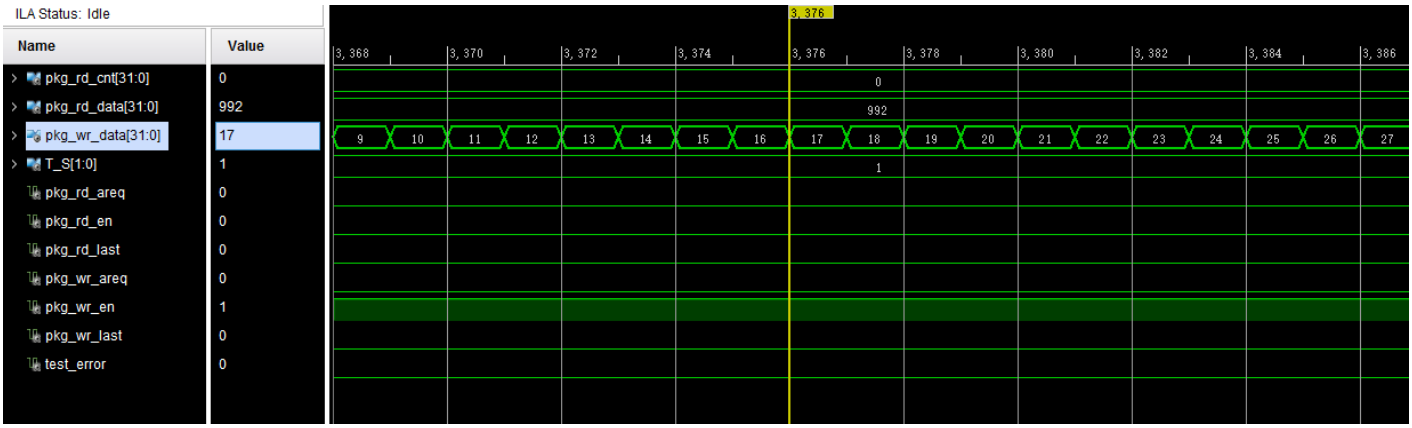
4、READ2 状态:当 pkg_rd_las 为 1 代表一次 read 传输结束，并且地址空间地址增加。在 READ2 状态，通过一个计数器计数并且对比和从 DDR 里面读出的数据，看是否有错误。

```
assign test_error = (pkg_rd_en && (pkg_rd_cnt != pkg_rd_data));
```

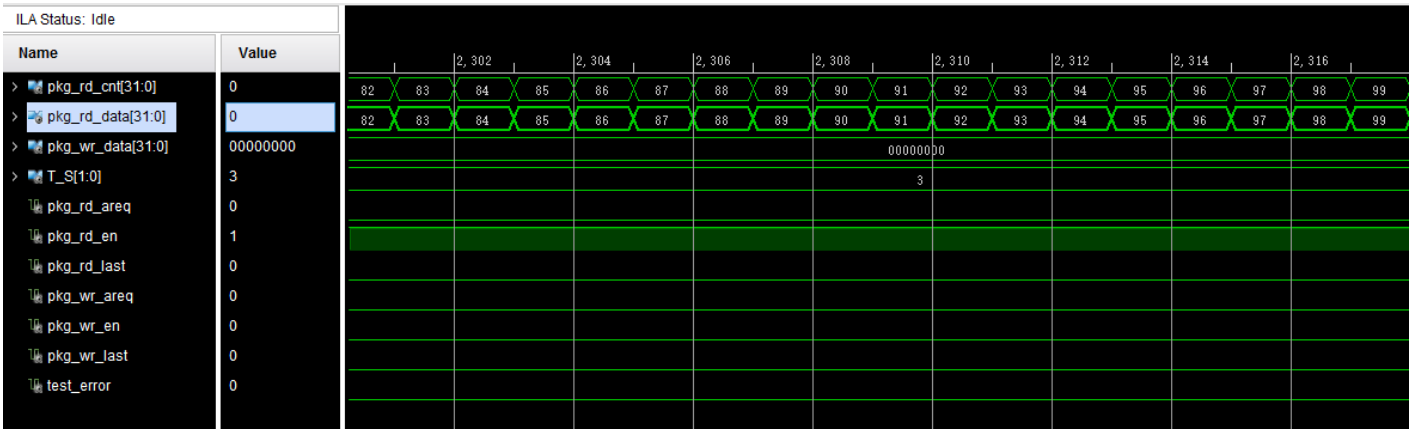
1.6 测试结果



可以看到上图中 error 信号一直为 0 代表数据读写对比没有错误。再看 pkg_wr_en 和 pkg_rd_en 信号，可以看到他们是每次 burst 了 4 次每次的长度是 256。证明我们的 FDMA 工作正常，测试程序工作正常。下面放大数据部分。



上图是写操作往内存里面写数据。



上图是读操作放大后看到的数据分别是计数器和从内存里读取到的数据。

CH02 基于 FDMA 实现多缓存视频构架

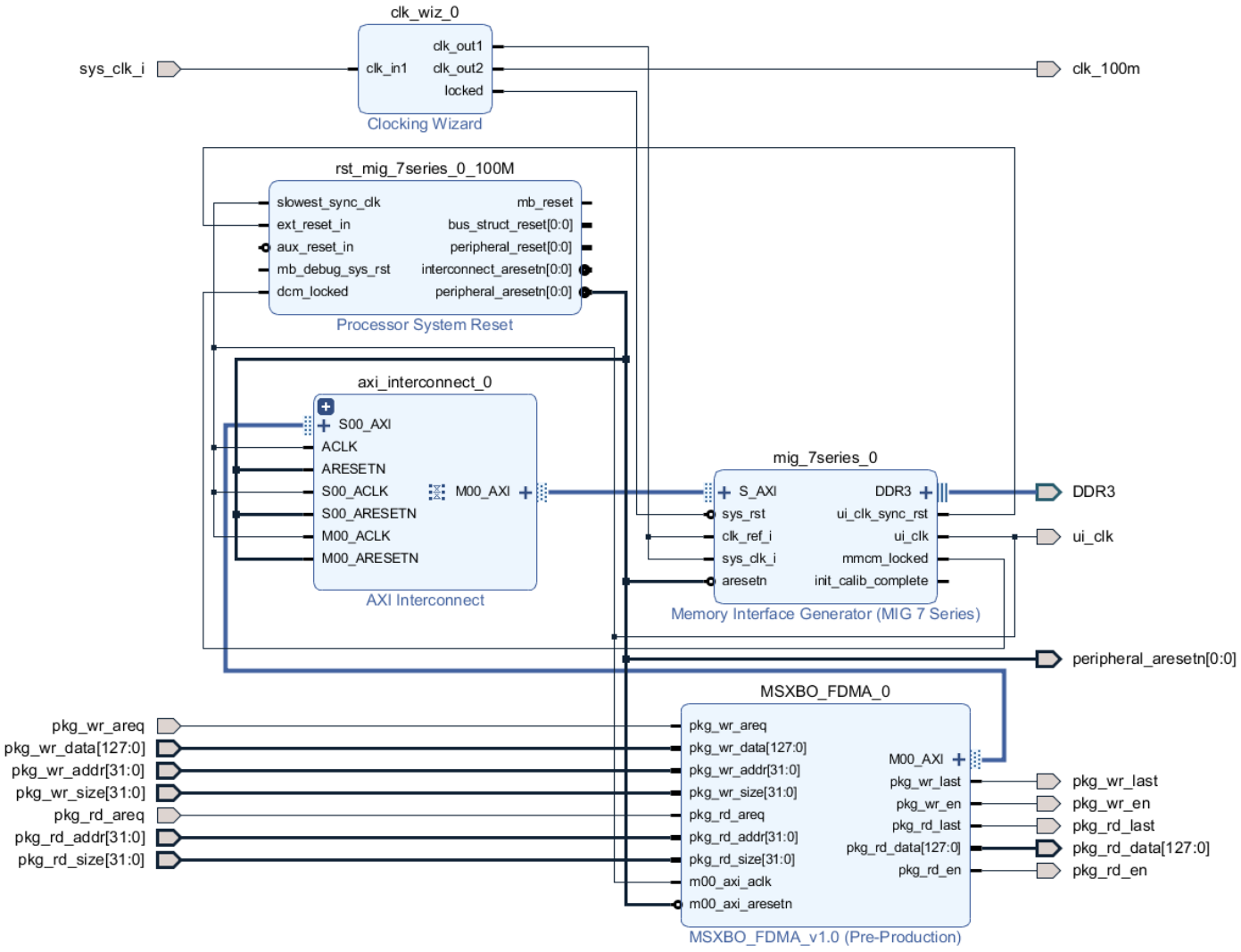
软件版本:VIVADO2017.4
操作系统:WIN10
硬件平台:MK7325

2.1 概述

很多学习 FPGA 编程的人都想走捷径，然而捷径就是一步一个脚印，才能迈向最终的成功，否则即便是你刚开始跑的快一点，但是你还是失败，得不偿失。这是正是所谓磨刀不误砍柴工。对于小编原创的 FDMA 没有经过充分验证就去跑应用，万一那里出了问题，到底是硬件的问题，还是软件的问题，还是 FDMA 的问题，那就难排查了。所以我们还是要做一些准备工作。在这节课中，我们将用 FPGA 代码产生一个 1080P60HZ 的测试图像，然后经过 FDMA 进入 DDR 缓存 3 帧后再从 FDMA 读出来，经过 HDMI 显示器显示。做完这个实验下一节课我们就加入 HDMI 输入视频信号再经过 3 次缓存后输出到 HDMI 显示器。

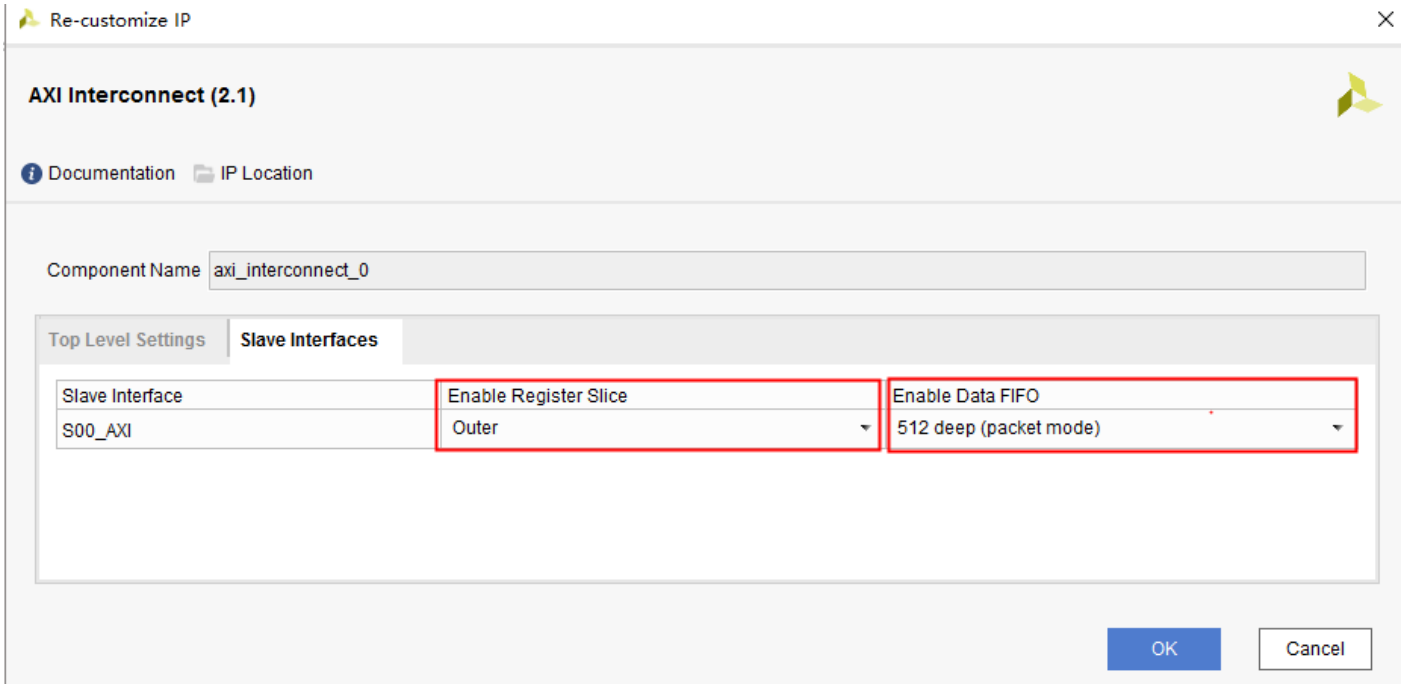
另外从本章开始，不提供详细的 VIVADO 软件使用或者配置步骤，除非一些新 IP 的出现有必要讲解的情况下才说明。

2.2 基于 FDMA 搭建的 BD 工程



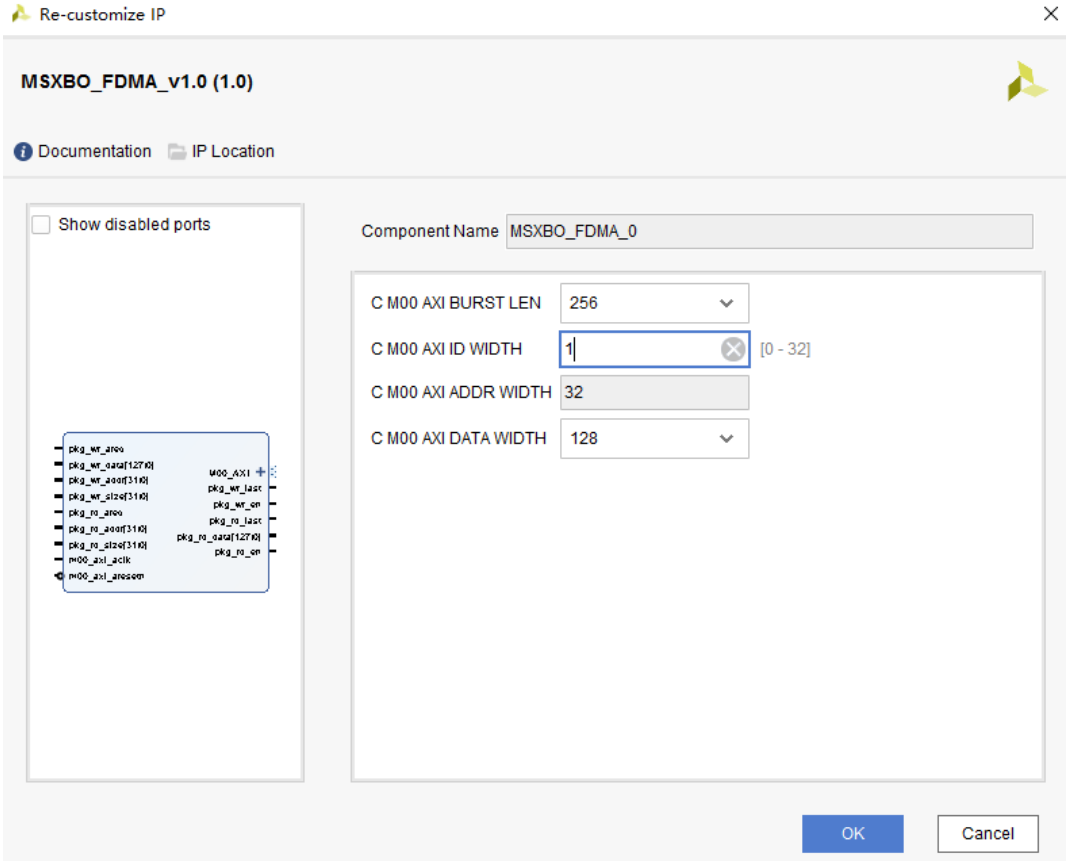
这个BD文件和前面一节课的基本一样。但是有个IP换了，换成了axi_interconnect IP。有人可能会

问，为什么要换这个IP,一开始我也不知道。但是可以肯定的是用上一课的IP无法达到FDMA和MIG之间AXI4的最大带宽。导致1080P视频不能正常传输。所以小编就尝试换了这个IP而且把这个IP里面的几个参数修改了。



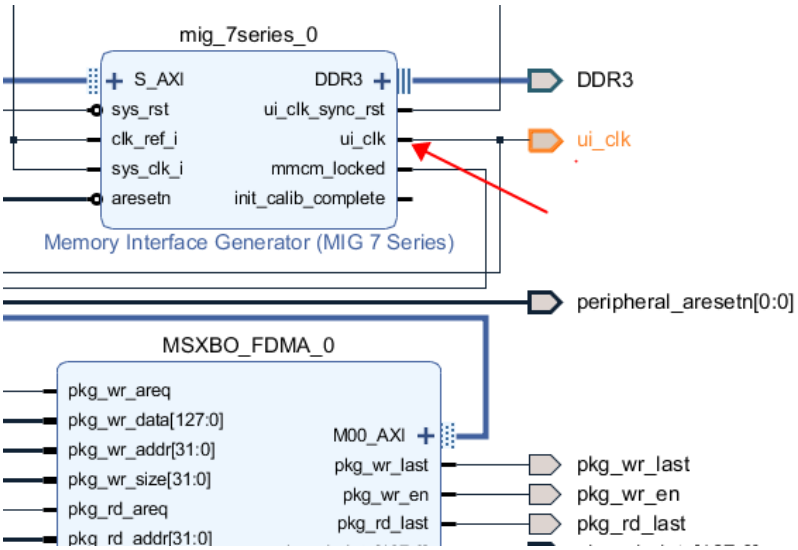
这样修改完成后，带宽就能最大化，在MK7325FA上完美传输1080P@60fps的视频。

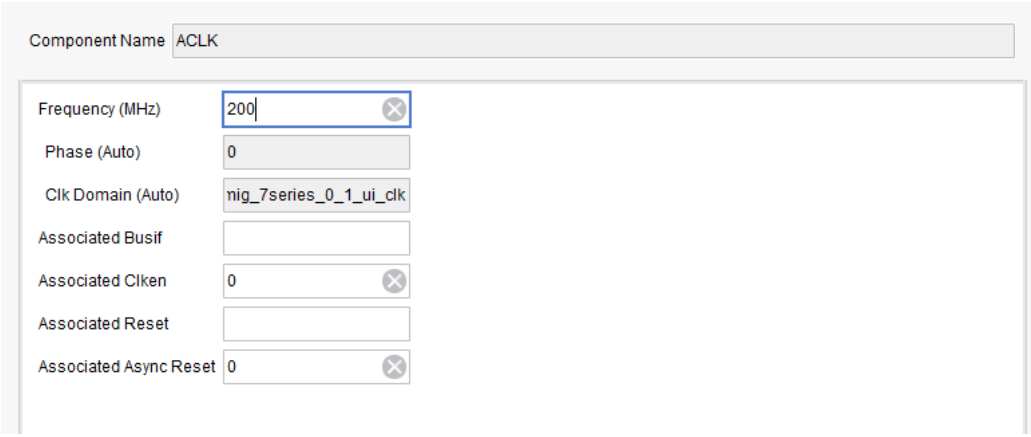
为了最大带宽，我们还要修改FDMA的IP参数如下图。



这里把BURST LEN设置为256 DATA WIDTH设置为128假设时钟100M，如果是设置32bit 那么才只有400MB/S的速度。完全不够1080P@60fps视频带宽要求。

如果设置128bit理论上有1600MB/S的带宽能力(实际不可能达到，效率一般在75% 左右)。如下面图所示。

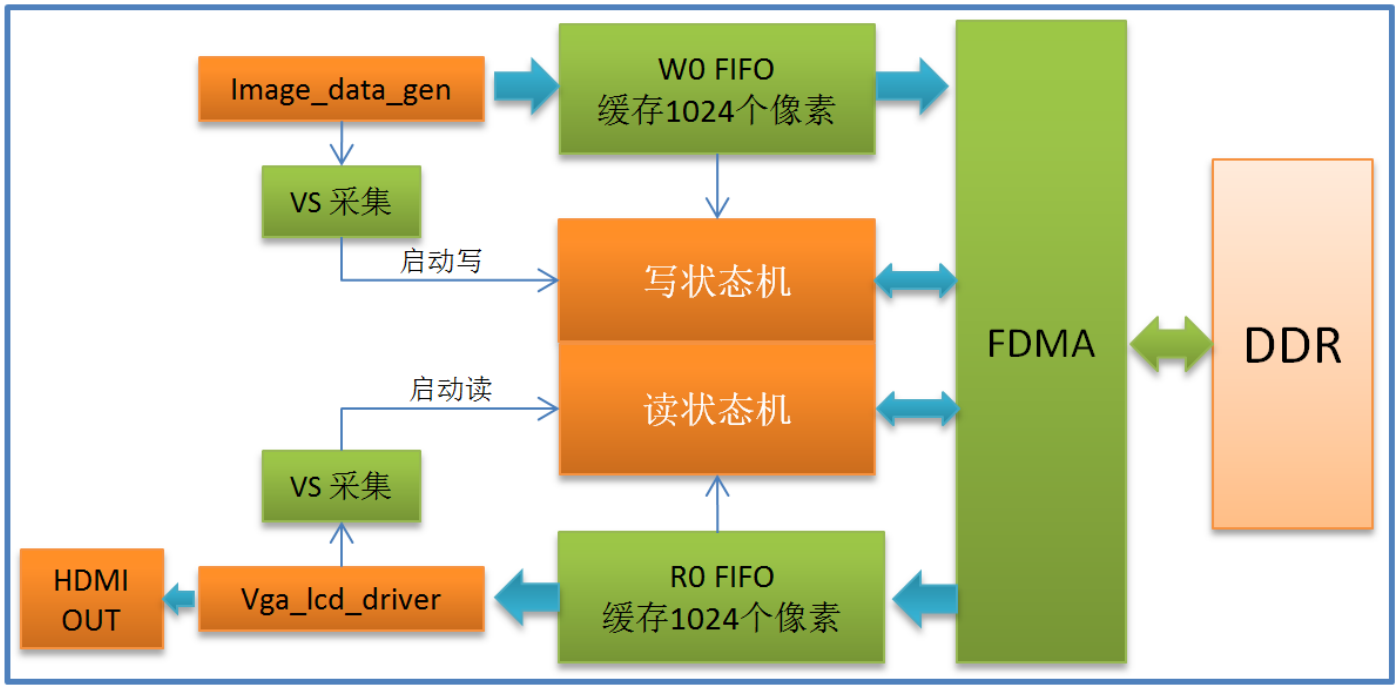




2.3 基于 FDMA 多缓存视频构架 fdma_controller

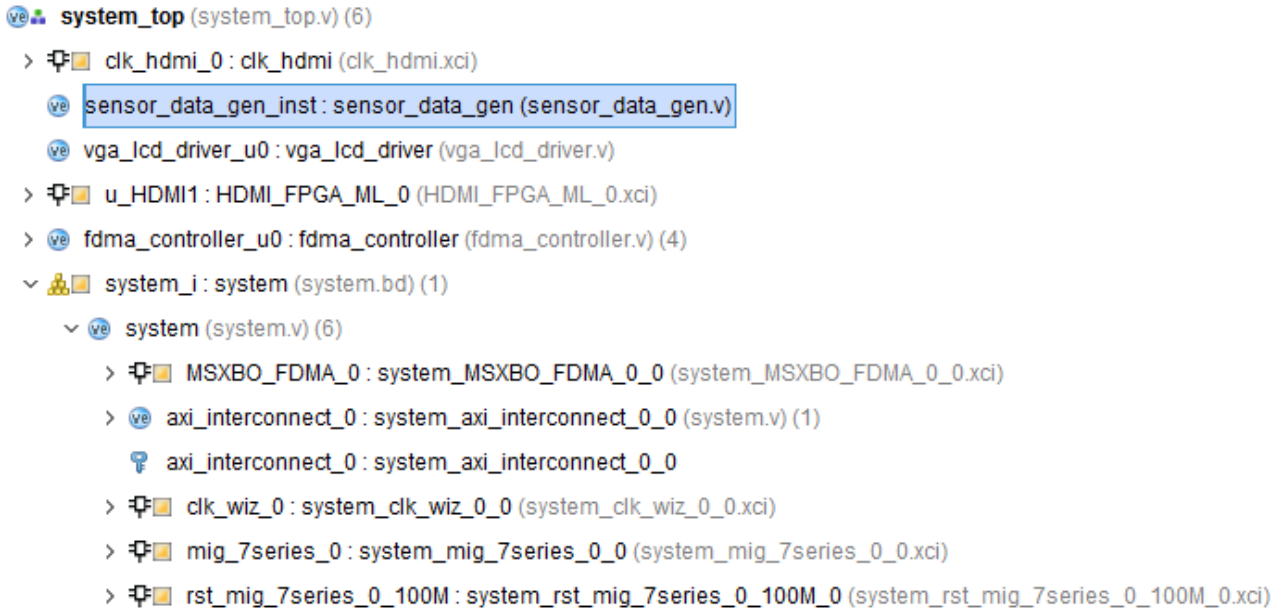
一个优秀的工程，必然有一个优秀的构架。优秀的构架具备通用、简洁、高可靠、易维护等优秀属性。小编见过很多人写的代码，只在一个特定的工程里面可以很好的工作，但是如果要换个环境，几乎要推到重来。这种代码就是比较垃圾的代码。我们在做一件事情的时候一定要注重效率，注重可重复利用性。这就要求我们对现在的使用环境，以及对将来的使用环境有一个认知。然后构建一个比较通用的代码构架。虽然小编也是半路出道，人算不上聪明，而且具备一定的懒惰性。小编喜欢做一些重复性的工作，最喜欢修改个别参数就能实现一些新的功能。基于以上目标，即便是我不是一个优秀的程序员，但是我依然要懂得偷懒，尽最大努力从一个小小的视频缓存构架开始，努力设计好这个简单的构架。那么对于正在阅读小编写的教程的初级程序猿，自然也要时刻考虑如何偷懒，如果你写的一个代码被成千上万，到几百万的人重复利用了，你一定成功了。

对于本课内容，小编绘制了如下框图。可以看 image_data_gen 产生了测试图片，之后进入过 W0 FIFO 进行视频缓存。每次缓存 1024 个像素，就往通过 FDMA 往 DDR 里面搬运数据。另外 VS 信号经过滤波采集后用于启动一次写状态机。同理对于图像的输出部分采用 HDMI 输出，用 Vga_lcd_driver 产生输出的时序。视频经过 R0 FIFO 缓存后输出。R0 FIFO 也是每次缓存 1024 个像素数据。



为了进行图像的多缓存，一般非同步信号至少要满足 3 缓存才能最大减少图片的延迟，撕裂，丢帧问题。本构架只要 DDR 够大，理论上可以进行无限次缓存。

2.4 代码叠层结构



2.5 fdma_controller

当然针对上图还无法完全展示 fdma_controller 控制器的全部功能。下面贴出代码

```
module fdma_controller#
(
parameter    integer    ADDR_OFFSET = 0,
parameter    integer    BUF_SIZE = 3,
parameter    integer    H_CNT = 640,
parameter    integer    V_CNT = 480
```

```
)
(
    input               ui_clk,
    input               ui_rstn,
//sensor input -W0_FIFO----------
    input               W0_FS_i,
    input               W0_wclk_i,
    input               W0_wren_i,
    input  [31:0]   W0_data_i,
//hdmi output -R0_FIFO----------
    input               R0_FS_i,
    input               R0_rclk_i,
    input               R0_rden_i,
    output[31:0]    R0_data_o,
//----------fdma signals write-------
    output  reg     pkg_wr_areq,
    input               pkg_wr_en,
    input               pkg_wr_last,
    output  [31:0]  pkg_wr_addr,
    output  [127:0] pkg_wr_data,
    output  [31:0]  pkg_wr_size,
//----------fdma signals read---------
    output  reg     pkg_rd_areq,
    input               pkg_rd_en,
    input               pkg_rd_last,
    output  [31:0]  pkg_rd_addr,
    input   [127:0] pkg_rd_data,
    output  [31:0]  pkg_rd_size
    );

parameter FBUF_SIZE  = BUF_SIZE  -1'b1;
parameter BURST_SIZE    = 1024*4;// one time 4KB
parameter BURST_TIMES  = H_CNT*V_CNT/1024;// one frame burst times
parameter PKG_SIZE      = 256;
assign pkg_wr_size = PKG_SIZE;
assign pkg_rd_size = PKG_SIZE;

//----------vs 滤波----------
reg   W0_FIFO_Rst;
reg   R0_FIFO_Rst;

wire W0_FS;
wire R0_FS;

reg [6:0] W0_Fbuf;
reg [6:0] R0_Fbuf;

reg W0_s_rdy;
```

```
reg R0_s_rdy;

fs_cap fs_cap_W0(
    .clk_i(ui_clk),
    .rstn_i(ui_rstn),
    .vs_i(W0_FS_i),
    .s_rdy_i(W0_s_rdy),
    .fs_cap_o(W0_FS)
);
fs_cap fs_cap_R0(
    .clk_i(ui_clk),
    .rstn_i(ui_rstn),
    .vs_i(R0_FS_i),
    .s_rdy_i(R0_s_rdy),
    .fs_cap_o(R0_FS)
);
parameter S_IDLE    =    2'd0;
parameter S_RST     =    2'd1;
parameter S_DATA1   =    2'd2;
parameter S_DATA2   =    2'd3;

reg [1    :0]    W_MS;
reg [22:0]    W0_addr;
reg [31 :0]    W0_fcnt;
reg [10    :0] W0_bcnt;
wire [10:0]    W0_rcnt;
reg W0_REQ;
reg [1 :0]    R_MS;
reg [22 :0] R0_addr;
reg [31 :0] R0_fcnt;
reg [10 :0] R0_bcnt;
wire [10:0] R0_wcnt;
reg R0_REQ;

assign pkg_wr_addr = {W0_Fbuf,W0_addr}+ ADDR_OFFSET;
assign pkg_rd_addr = {R0_Fbuf,R0_addr}+ ADDR_OFFSET;
//assign pkg_wr_data = W0_fcnt;
//--------一帧图像写入 DDR----------
  always @(posedge ui_clk) begin
      if(!ui_rstn)begin
            W_MS <= S_IDLE;
            W0_addr <= 21'd0;
            pkg_wr_areq <= 1'd0;
            W0_FIFO_Rst <= 1'b1;
            W0_fcnt <= 0;
            W0_bcnt <= 0;
            W0_s_rdy <= 1'b0;
            W0_Fbuf <= 7'd0;
```

```
    end
    else begin
      case(W_MS)
       S_IDLE:begin
           W0_addr <= 21'd0;
           W0_fcnt <= 0;
           W0_bcnt <= 11'd0;
           W0_s_rdy <= 1'b1;
           if(W0_FS) W_MS <= S_RST;
       end
       S_RST:begin
            W0_s_rdy <= 1'b0;
           if(W0_fcnt > 8'd30 ) W_MS <= S_DATA1;
           W0_FIFO_Rst <= (W0_fcnt < 8'd20);
           W0_fcnt <= W0_fcnt +1'd1;
         end
         S_DATA1:begin
              if(W0_bcnt == BURST_TIMES) begin
                   if(W0_Fbuf == FBUF_SIZE)
                        W0_Fbuf <= 7'd0;
                    else
                        W0_Fbuf <= W0_Fbuf + 1'b1;
                    W_MS <= S_IDLE;
              end
              else if(W0_REQ) begin
                   W0_fcnt <=0;
                   pkg_wr_areq <= 1'b1;
                   W_MS <= S_DATA2;
              end
          end
          S_DATA2:begin
              pkg_wr_areq <= 1'b0;
              if(pkg_wr_last)begin
                   W_MS <= S_DATA1;
                   W0_bcnt <= W0_bcnt + 1'd1;
                   W0_addr <= W0_addr + BURST_SIZE;
              end
          end
       endcase
    end
 end

//--------一帧图像读出 DDR-----------
 always @(posedge ui_clk) begin
    if(!ui_rstn)begin
       R_MS <= S_IDLE;
       R0_addr <= 21'd0;
       pkg_rd_areq <= 1'd0;
```

```
            R0_fcnt <=0;
            R0_bcnt <=0;
            R0_FIFO_Rst <= 1'b1;
            R0_s_rdy <= 1'b0;
            R0_Fbuf <= 7'd0;
        end
        else begin
          case(R_MS)
            S_IDLE:begin
              R0_addr <= 21'd0;
              R0_fcnt <=0;
              R0_bcnt <=0;
              R0_s_rdy <= 1'b1;
              if(R0_FS) R_MS <= S_RST;
            end
            S_RST:begin
              R0_s_rdy <= 1'b0;
              if(R0_fcnt > 8'd30 ) R_MS <= S_DATA1;
              R0_FIFO_Rst <= (R0_fcnt < 8'd20);
              R0_fcnt <= R0_fcnt + 1'd1;
            end
            S_DATA1:begin
                if(R0_bcnt == BURST_TIMES ) begin
                    R_MS <= S_IDLE;
                    if(W0_Fbuf == 7'd0)
                        R0_Fbuf <= FBUF_SIZE;
                     else
                        R0_Fbuf <= W0_Fbuf - 1'b1;
                end
                else if(R0_REQ) begin
                    pkg_rd_areq <= 1'b1;
                    R_MS <= S_DATA2;
                end
             end
             S_DATA2:begin
                pkg_rd_areq <= 1'b0;
                if(pkg_rd_last)begin
                    R_MS <= S_DATA1;
                    R0_bcnt <= R0_bcnt + 1'd1;
                    R0_addr <= R0_addr + BURST_SIZE;
                end
             end
           endcase
        end
    end

  always@(posedge ui_clk)
  begin
```

```
        W0_REQ      <= (W0_rcnt      >= PKG_SIZE);
        R0_REQ      <= (R0_wcnt      <= PKG_SIZE);
  end

W0_FIFO W0_FIFO_0 (
    .rst(W0_FIFO_Rst),    // input wire rst
    .wr_clk(W0_wclk_i),    // input wire wr_clk
    .din(W0_data_i),              // input wire [31 : 0] din
    .wr_en(W0_wren_i),        // input wire wr_en
    .rd_clk(ui_clk),    // input wire rd_clk
    .rd_en(pkg_wr_en),        // input wire rd_en
    .dout(pkg_wr_data),           // output wire [63 : 0] dout
    .rd_data_count(W0_rcnt)    // output wire [10 : 0] wr_data_count
);

  R0_FIFO R0_FIFO_0 (
    .rst(R0_FIFO_Rst),    // input wire rst
    .wr_clk(ui_clk),    // input wire wr_clk
    .din(pkg_rd_data),              // input wire [63 : 0] din
    .wr_en(pkg_rd_en),        // input wire wr_en
    .wr_data_count(R0_wcnt),    // output wire [6 : 0] rd_data_count
    .rd_clk(R0_rclk_i),    // input wire rd_clk
    .rd_en(R0_rden_i),        // input wire rd_en
    .dout(R0_data_o)          // output wire [31 : 0] dout
);

endmodule
```

截取上面完整代码中部分代码如下，可以看到小编通过控制高地址，轻松完成缓存地址切换。

```
assign pkg_wr_addr = {W0_Fbuf,W0_addr}+ ADDR_OFFSET;
assign pkg_rd_addr = {R0_Fbuf,R0_addr}+ ADDR_OFFSET;
```

再来看下顶层接口，这里面 4 个参数分辨率代表内存地址的偏移，帧缓存数量，图像的水平像素 和垂直像素。

```
module fdma_controller#
(
parameter    integer    ADDR_OFFSET = 0,
parameter    integer    BUF_SIZE = 3,
parameter    integer    H_CNT = 640,
parameter    integer    V_CNT = 480
)
```

最后看下我们如何如何调用 fdma_controller，并且设置 1080P 的分辨率的。可以看到小编这里设置的偏移地址为 0，缓存为 3 缓存，分辨率为 1920X1080。就是这么简单。目前 fdma_controller 不修改源码的情况下支持任意是 1024 个像素的整数倍的图像。比如 1280X720/640X800/1024X600，如果不是整数倍那就需要你自己修改 fdma_controller 和 FDMA 的参数了。

```
//--------------fdma image buf controller-----------------------
fdma_controller # (
```

```
.ADDR_OFFSET(0),
.BUF_SIZE(3),
.H_CNT (1920),
.V_CNT (1080)
) fdma_controller_u0
(
        //FDAM signals
        .ui_clk(ui_clk),
        .ui_rstn(ui_rstn),
    //Sensor video
        .W0_FS_i(W0_FS_i),
        .W0_wclk_i(W0_wclk_i),
        .W0_wren_i(W0_wren_i),
        .W0_data_i(W0_data_i),
        //vga/hdmi output -CH6_FIFO
        .R0_FS_i(R0_FS_i),
        .R0_rclk_i(R0_rclk_i),
        .R0_rden_i(R0_rden_i),
        .R0_data_o(R0_data_o),

        .pkg_wr_areq(pkg_wr_areq),
        .pkg_wr_en(pkg_wr_en),
        .pkg_wr_last(pkg_wr_last),
        .pkg_wr_addr(pkg_wr_addr),
        .pkg_wr_data(pkg_wr_data),
        .pkg_wr_size(pkg_wr_size),

        .pkg_rd_areq(pkg_rd_areq),
        .pkg_rd_en(pkg_rd_en),
        .pkg_rd_last(pkg_rd_last),
        .pkg_rd_addr(pkg_rd_addr),
        .pkg_rd_data(pkg_rd_data),
        .pkg_rd_size(pkg_rd_size)
 );
```

其他部分的代码比较简单，只给出代码源码，如果有不清楚的可以给我留言。

2.6 sensor_data_gen

```
module    sensor_data_gen (
    input    clk,
    output [23:0]rgb,
    output de,
    output vsync,
    output hsync
    );

   reg    [23:0] colour=24'd0;
   reg    [11:0] hcounter=12'd0;
```

```
    reg    [11:0] vcounter=12'd0;

// Colours converted using The RGB -> YCbCr converter app found on Google Gadgets
                                        //     Y       Cb      Cr
    `define C_BLACK  24'h000000;    //    16     128    128
    `define C_RED      24'hFF0000;    //    81     90      240
    `define C_GREEN  24'h00FF00;    //    172    42      27
    `define C_BLUE     24'h0000FF;    //    32     240    118
    `define C_WHITE 24'hFFFFFF;    //    234    128    128


//     -- Set the video mode to 1920x1080x60Hz (150MHz pixel clock needed)
    parameter hVisible     = 1920;
    parameter hStartSync = 1920+88;
    parameter hEndSync      = 1920+88+44;
    parameter hMax             = 1920+88+44+148; //2200

    parameter vVisible        = 1080;
    parameter vStartSync    = 1080+4;
    parameter vEndSync        = 1080+4+5;
    parameter vMax              = 1080+4+5+36; //1125
       /**/

//     -- Set the video mode to 1440x900x60Hz (106.47MHz pixel clock needed)
/*     parameter hVisible       = 1440;
    parameter hStartSync = 1440+80;
    parameter hEndSync      = 1440+80+152;
    parameter hMax             = 1440+80+152+232; //1904

    parameter vVisible        = 900;
    parameter vStartSync    = 900+1;
    parameter vEndSync        = 900+1+3;
    parameter vMax              = 900+1+3+28; //932
    */


//     -- Set the video mode to 1280x720x60Hz (75MHz pixel clock needed)
  /*    parameter hVisible       = 1280;
    parameter hStartSync = 1280+72;
    parameter hEndSync      = 1280+72+80;
    parameter hMax             = 1280+72+80+216; //1647

    parameter vVisible        = 720;
    parameter vStartSync    = 720+3;
    parameter vEndSync        = 720+3+5;
    parameter vMax              = 720+3+5+22; //749
     */
```

```
//     -- Set the video mode to 800x600x60Hz (40MHz pixel clock needed)
/*     parameter hVisible       = 800;
      parameter hStartSync = 840; //800+40
      parameter hEndSync      = 968; //800+40+128
      parameter hMax             = 1056; //800+40+128+88

      parameter vVisible        = 600;
      parameter vStartSync    = 601; //600+1
      parameter vEndSync        = 605; //600+1+4
      parameter vMax              = 628; //600+1+4+23



//     -- Set the video mode to 640x480x60Hz (25MHz pixel clock needed)

      parameter hVisible       = 640;
      parameter hStartSync = 656; //640+16
      parameter hEndSync      = 752; //640+16+96
      parameter hMax             = 800; //640+16+96+48

      parameter vVisible        = 480;
      parameter vStartSync    = 490; //480+10
      parameter vEndSync        = 492; //480+10+2
      parameter vMax              = 525; //480+10+2+33
*/
//----------------------------------------
//v_sync counter & generator

always@(posedge clk) begin
if(hcounter < hMax - 12'd1)              //line over
      hcounter <= hcounter + 12'd1;
else
      hcounter <= 12'd0;
end

always@(posedge clk) begin
if(hcounter == hMax - 12'd1) begin
      if(vcounter < vMax - 12'd1)    //frame over
            vcounter <= vcounter + 12'd1;
      else
            vcounter <= 12'd0;
      end
end



 always@(posedge clk) begin
      if (hcounter <= hVisible/5)  begin colour <= `C_RED;  end
```

```
        else if(hcounter <= 2*hVisible/5)  begin colour <= `C_GREEN;  end
        else if(hcounter <= 3*hVisible/5)  begin colour <= `C_BLUE;  end
        else if(hcounter <= 4*hVisible/5)  begin colour <= `C_WHITE;  end
        else    begin colour <= `C_BLACK;   end
    end
/*
    assign r = colour[23:16];
    assign g = colour[15:8];
    assign b = colour[7:0];
    */

reg [7:0]VGA_R_reg;
reg [7:0]VGA_G_reg;
reg [7:0]VGA_B_reg;
reg [10:0] dis_mode;
always @(posedge clk) begin
        if((vcounter == vMax - 12'd1)&&(hcounter == hMax - 12'd1))
                dis_mode <= dis_mode +1'b1;

end

reg[7:0]   grid_data_1;
reg[7:0]   grid_data_2;
always @(posedge clk)                   //格子图像
begin
        if((hcounter[4]==1'b1)^(vcounter[4]==1'b1))
        grid_data_1     <=   8'h00;
        else
        grid_data_1     <=   8'hff;

        if((hcounter[6]==1'b1)^(vcounter[6]==1'b1))
        grid_data_2     <=   8'h00;
        else
        grid_data_2     <=   8'hff;
end

reg[23:0] color_bar;
always @(posedge clk)
begin
        if(hcounter==260)
        color_bar <=    24'hff0000;
        else if(hcounter==420)
        color_bar <=    24'h00ff00;
        else if(hcounter==580)
        color_bar <=    24'h0000ff;
        else if(hcounter==740)
        color_bar <=    24'hff00ff;
        else if(hcounter==900)
```

```
        color_bar <=    24'hffff00;
        else if(hcounter==1060)
        color_bar <=    24'h00ffff;
        else if(hcounter==1220)
        color_bar <=    24'hffffff;
        else if(hcounter==1380)
        color_bar <=    24'h000000;
        else
        color_bar <=    color_bar;
end

always @(posedge clk)
begin
      if(1'b0)
            begin
           VGA_R_reg<=0;
           VGA_G_reg<=0;
           VGA_B_reg<=0;
            end
    else
       case(dis_mode[10:7])
            4'd0:begin
                        VGA_R_reg<=0;                   //LCD 显示彩色条
                        VGA_G_reg<=0;
                        VGA_B_reg<=0;
                end
                4'd1:begin
                        VGA_R_reg<=8'b11111111;                         //LCD 显示全白
                        VGA_G_reg<=8'b11111111;
                        VGA_B_reg<=8'b11111111;
                end
                4'd2:begin
                        VGA_R_reg<=8'b11111111;                        //LCD 显示全红
                        VGA_G_reg<=0;
                        VGA_B_reg<=0;
                  end
               4'd3:begin
                        VGA_R_reg<=0;                                   //LCD 显示全绿
                        VGA_G_reg<=8'b11111111;
                        VGA_B_reg<=0;
                end
                  4'd4:begin
                        VGA_R_reg<=0;                                  //LCD 显示全蓝
                        VGA_G_reg<=0;
                        VGA_B_reg<=8'b11111111;
                end
                  4'd5:begin
                        VGA_R_reg<=grid_data_1;                       // LCD 显示方格 1
```

```
                    VGA_G_reg<=grid_data_1;
                    VGA_B_reg<=grid_data_1;
                end
                4'd6:begin
                    VGA_R_reg<=grid_data_2;                    // LCD 显示方格 2
                    VGA_G_reg<=grid_data_2;
                    VGA_B_reg<=grid_data_2;
                end
                4'd7:begin
                    VGA_R_reg<=hcounter[7:0];                  //LCD 显示水平渐变色
                    VGA_G_reg<=hcounter[7:0];
                    VGA_B_reg<=hcounter[7:0];
                 end
                 4'd8:begin
                    VGA_R_reg<=vcounter[8:1];                  //LCD 显示垂直渐变色
                    VGA_G_reg<=hcounter[8:1];
                    VGA_B_reg<=hcounter[8:1];
                end
                 4'd9:begin
                    VGA_R_reg<=hcounter[7:0];                  //LCD 显示红水平渐变色
                    VGA_G_reg<=0;
                    VGA_B_reg<=0;
                end
                 4'd10:begin
                    VGA_R_reg<=0;                              //LCD 显示绿水平渐变色
                    VGA_G_reg<=hcounter[7:0];
                    VGA_B_reg<=0;
                end
                 4'd11:begin
                    VGA_R_reg<=0;                              //LCD 显示蓝水平渐变色
                    VGA_G_reg<=0;
                    VGA_B_reg<=hcounter[7:0];
                end
                 4'd12:begin
                    VGA_R_reg<=color_bar[23:16];               //LCD 显示彩色条
                    VGA_G_reg<=color_bar[15:8];
                    VGA_B_reg<=color_bar[7:0];
                end
               default:begin
                    VGA_R_reg<=8'b11111111;                    //LCD 显示全白
                    VGA_G_reg<=8'b11111111;
                    VGA_B_reg<=8'b11111111;
                end
            endcase
end

assign hsync = ((hcounter >= (hStartSync - 2'd2))&&(hcounter < (hEndSync - 2'd2)))? 1'b0:1'b1;   //Generate the hSync
Pulses
```

```
assign vsync = ((vcounter >= (vStartSync - 1'b1))&&(vcounter < (vEndSync - 1'b1)))? 1'b0:1'b1; //Generate the vSync Pulses
assign de =       (vcounter >= vVisible || hcounter >= hVisible)  ? 1'b0 : 1'b1;
assign rgb = {VGA_R_reg,VGA_G_reg,VGA_B_reg};

 endmodule
```

2.7 vga_lcd_driver.v

```
module vga_lcd_driver(
      input    clk,
      input    [7:0]    r_i,
      input    [7:0]    g_i,
      input    [7:0]    b_i,
      output [7:0]    r_o,
      output [7:0]    g_o,
      output [7:0]    b_o,
      output de,
      output vsync,
      output hsync
      );

    reg    [11:0] hcounter;
    reg    [11:0] vcounter;

// Colours converted using The RGB

//     -- Set the video mode to 1920x1080x60Hz (150MHz pixel clock needed)
    parameter hVisible     = 1920;
    parameter hStartSync = 1920+88;
    parameter hEndSync      = 1920+88+44;
    parameter hMax             = 1920+88+44+148; //2200

    parameter vVisible        = 1080;
    parameter vStartSync    = 1080+4;
    parameter vEndSync        = 1080+4+5;
    parameter vMax              = 1080+4+5+36; //1125
/*    */

//     -- Set the video mode to 1440x900x60Hz (106.47MHz pixel clock needed)
/*     parameter hVisible       = 1440;
    parameter hStartSync = 1440+80;
    parameter hEndSync      = 1440+80+152;
    parameter hMax             = 1440+80+152+232; //1904

    parameter vVisible        = 900;
    parameter vStartSync    = 900+1;
    parameter vEndSync        = 900+1+3;
```

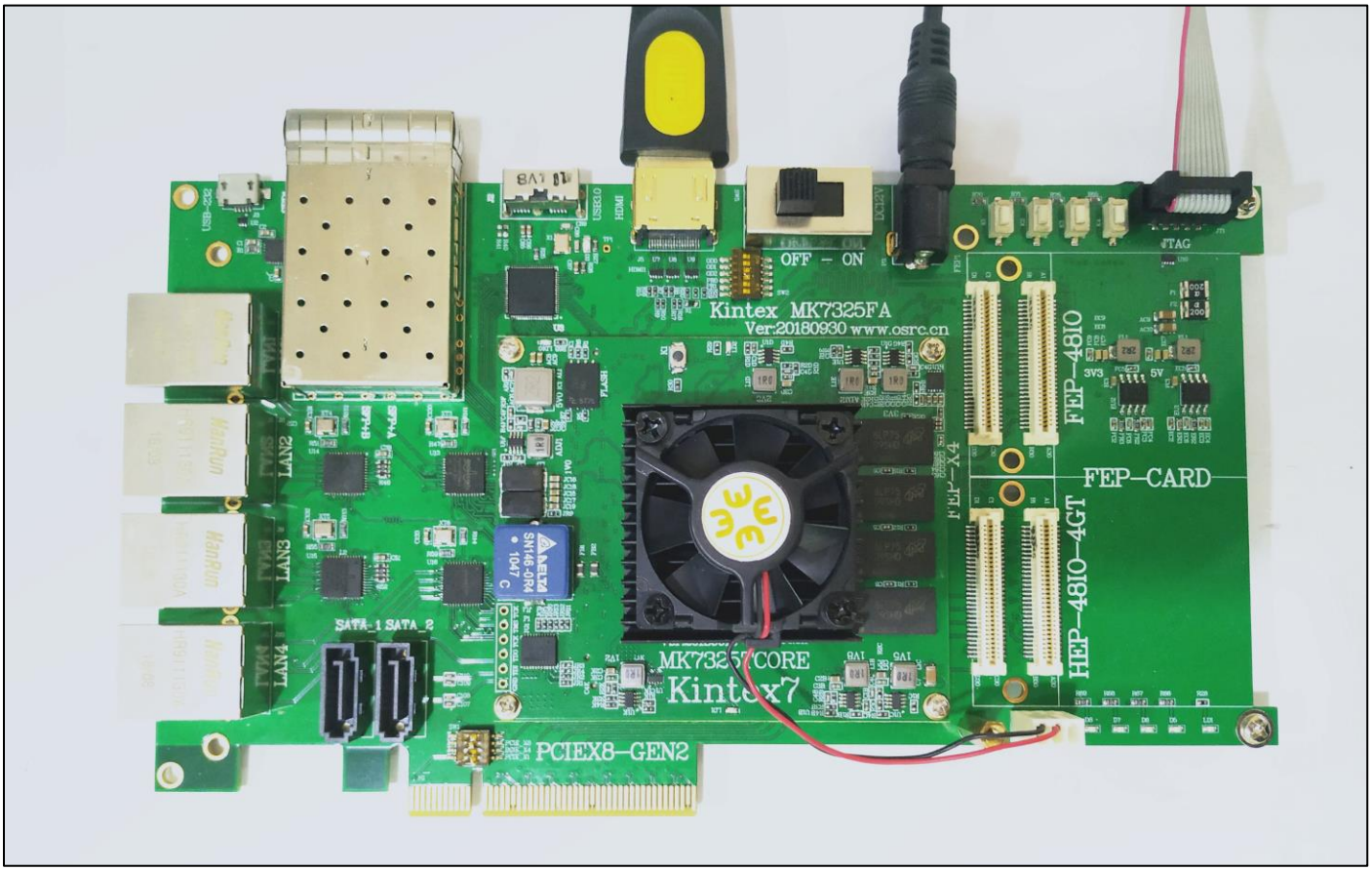
```
    parameter vMax        = 900+1+3+28; //932
    */


//    -- Set the video mode to 1280x720x60Hz (75MHz pixel clock needed)
   /*parameter hVisible    = 1280;
    parameter hStartSync = 1280+72;
    parameter hEndSync    = 1280+72+80;
    parameter hMax        = 1280+72+80+216; //1647

    parameter vVisible    = 720;
    parameter vStartSync  = 720+3;
    parameter vEndSync    = 720+3+5;
    parameter vMax        = 720+3+5+22; //749
     */

//    -- Set the video mode to 800x600x60Hz (40MHz pixel clock needed)
/*    parameter hVisible    = 800;
    parameter hStartSync = 840; //800+40
    parameter hEndSync    = 968; //800+40+128
    parameter hMax        = 1056; //800+40+128+88

    parameter vVisible    = 600;
    parameter vStartSync  = 601; //600+1
    parameter vEndSync    = 605; //600+1+4
    parameter vMax        = 628; //600+1+4+23

//    -- Set the video mode to 640x480x60Hz (25MHz pixel clock needed)

    parameter hVisible    = 640;
    parameter hStartSync = 656; //640+16
    parameter hEndSync    = 752; //640+16+96
    parameter hMax        = 800; //640+16+96+48

    parameter vVisible    = 480;
    parameter vStartSync  = 490; //480+10
    parameter vEndSync    = 492; //480+10+2
    parameter vMax        = 525; //480+10+2+33
*/

//-------------------------------------
//v_sync counter & generator

always@(posedge clk) begin
if(hcounter < hMax - 12'd1)          //line over
    hcounter <= hcounter + 12'd1;
else
    hcounter <= 12'd0;
```

```
end

always@(posedge clk) begin
if(hcounter == hMax - 12'd1) begin
      if(vcounter < vMax - 12'd1)    //frame over
            vcounter <= vcounter + 12'd1;
      else
            vcounter <= 12'd0;
      end
end

assign hsync = ((hcounter >= (hStartSync - 2'd2))&&(hcounter < (hEndSync - 2'd2)))? 1'b0:1'b1;   //Generate the hSync Pulses
assign vsync = ((vcounter >= (vStartSync - 1'b1))&&(vcounter < (vEndSync - 1'b1)))? 1'b0:1'b1; //Generate the vSync Pulses



   assign de = (vcounter >= vVisible || hcounter >= hVisible)  ? 1'b0 : 1'b1;
   assign r_o = r_i;
   assign g_o = g_i;
   assign b_o = b_i;
endmodule
```

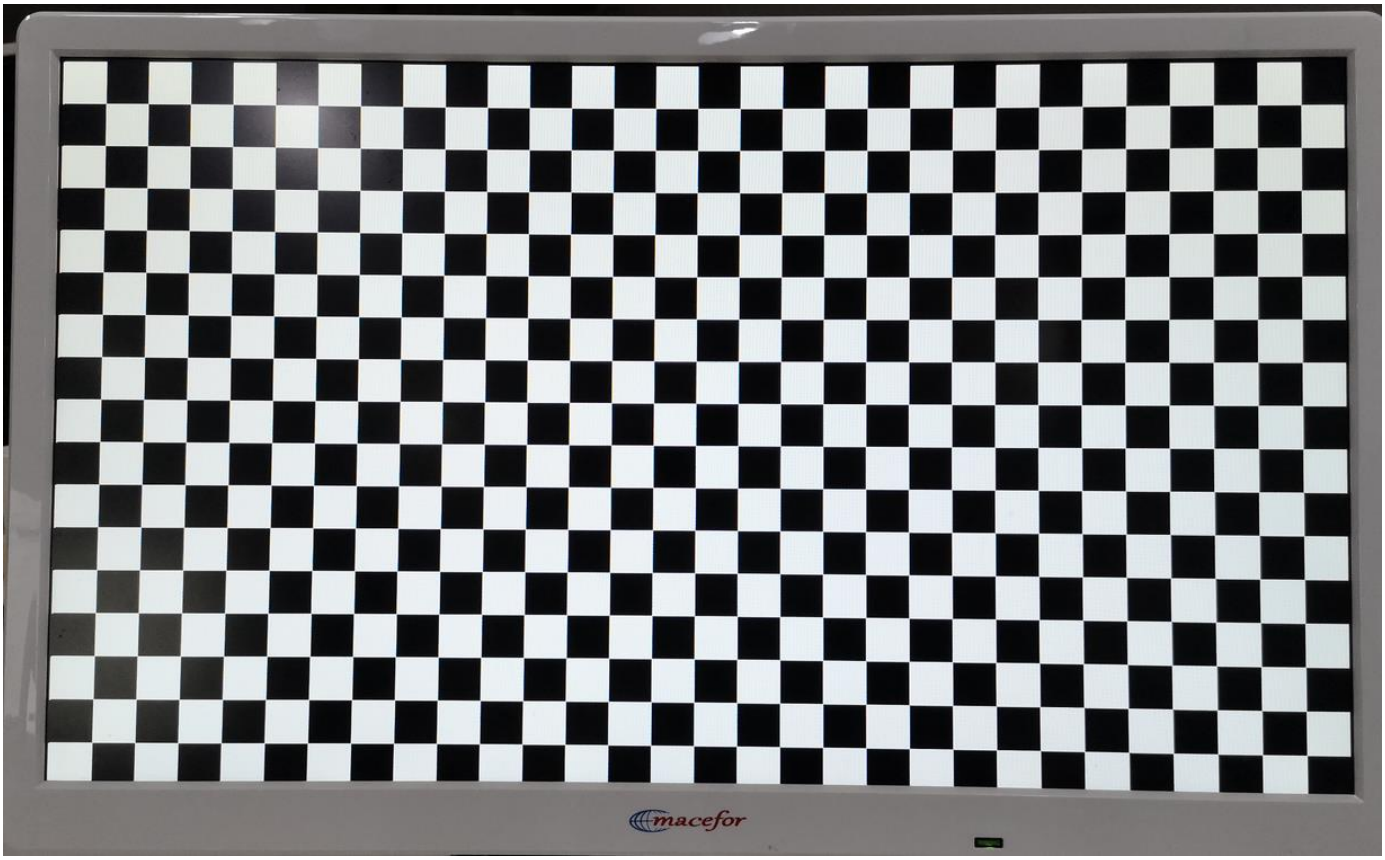
这个模块实际上和 image_data_gen 模块的代码几乎一样，只是这里的 RGB 数据是采用经过 DDR 缓存 后输入进来的 RGB 数据输出出去。

2.8 硬件连线

箭头所指的为 HDMI 输出，接入到显示器。


OFF - ON
Kintex MK7325FA
Ver:20180930 www.osrc.cn
FEP-48IO
FEP-CARD
FEP-48IO-4GT
SATA-1 SATA-2
PCIEX8-GEN2

2.9 测试结果


macefor

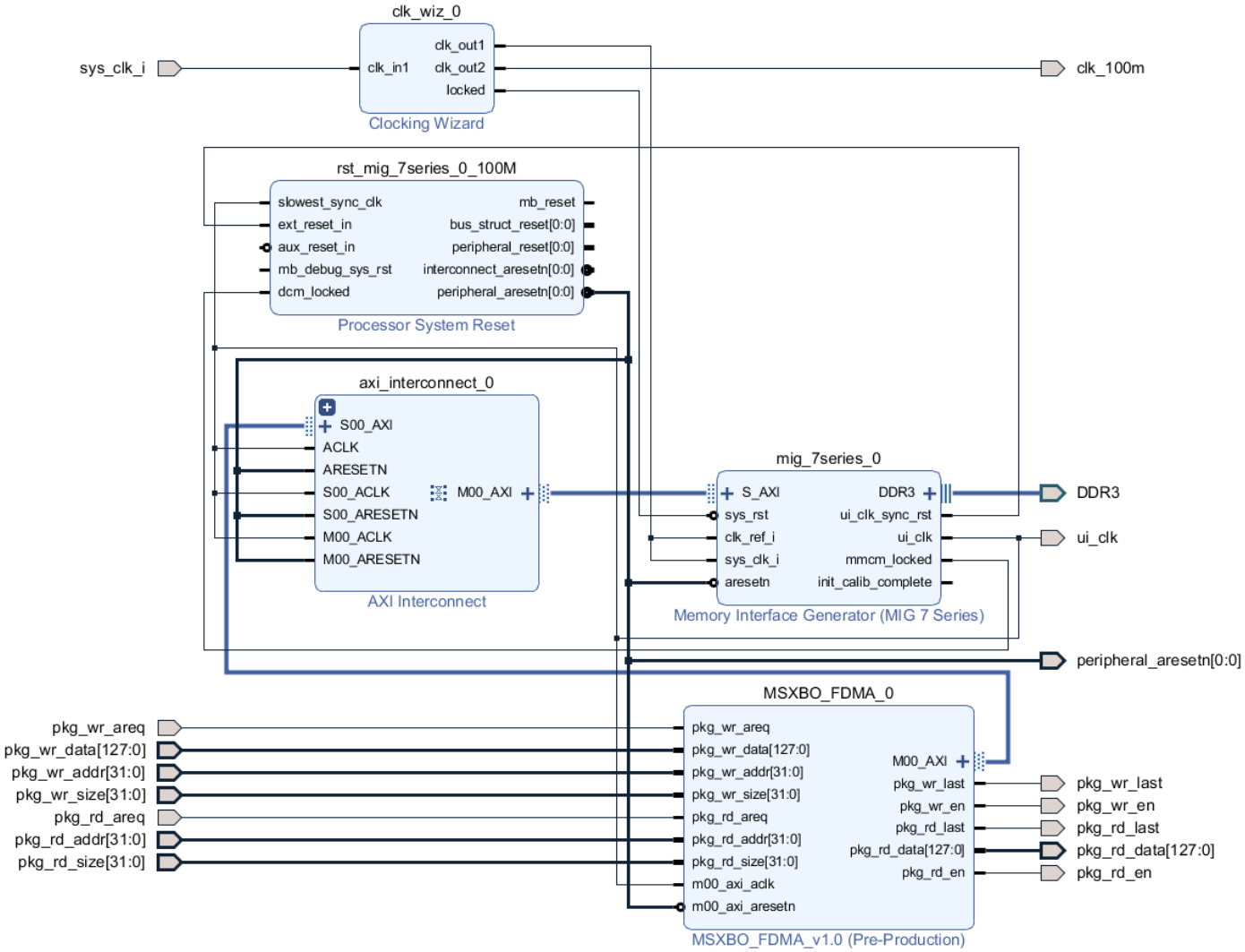
CH03 基于 FDMA 实现 HDMI 视频输入输出

软件版本:VIVADO2017.4
操作系统:WIN10
硬件平台:MK7325

3.1 概述

经过上面实现了 1080P 测试视频的多缓存构架设计，并且正确测试得到了结果。这一节课，我们将通过 HDMI 入门接口采集真实的 1080P 视频，然后经过 fdma_controller 的缓存控制机制，将数据经过 FDMA 缓存到 DDR。

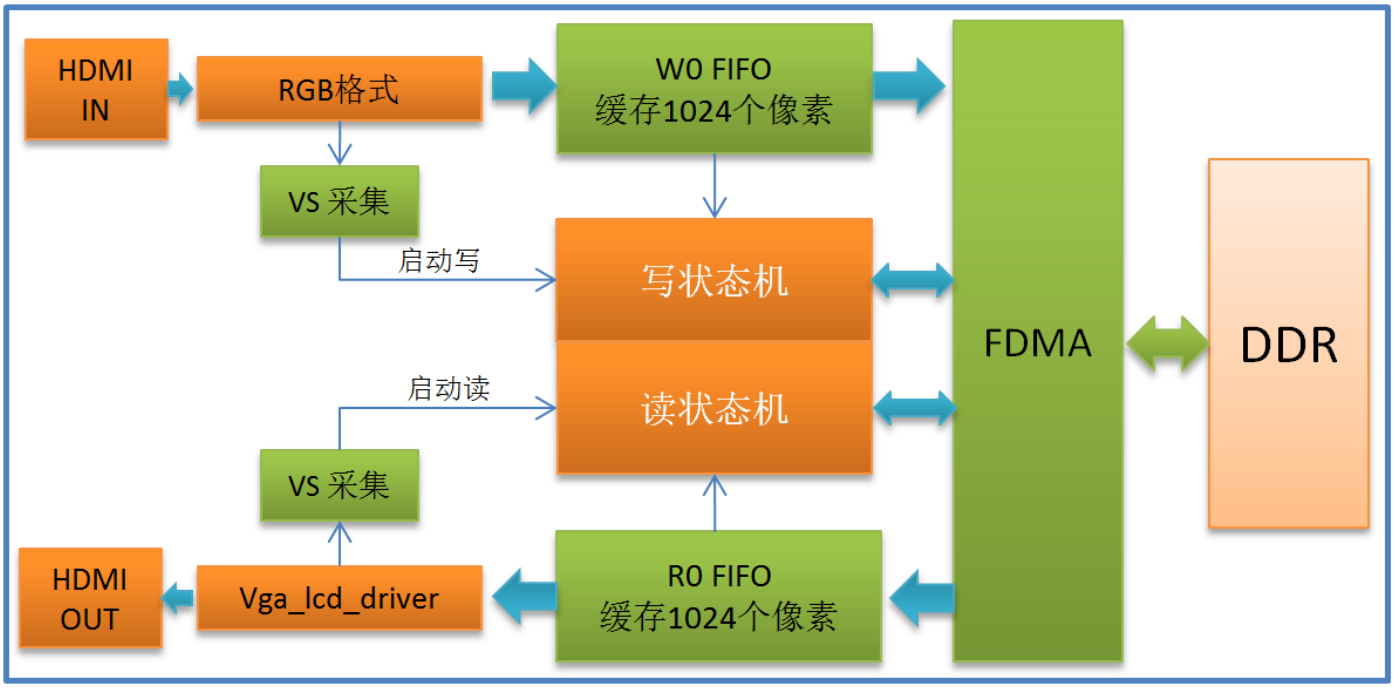
3.2 基于 FDMA 搭建的 BD 工程



这节课的 BD 工程和上一节课一样，所以不再重复叙述。

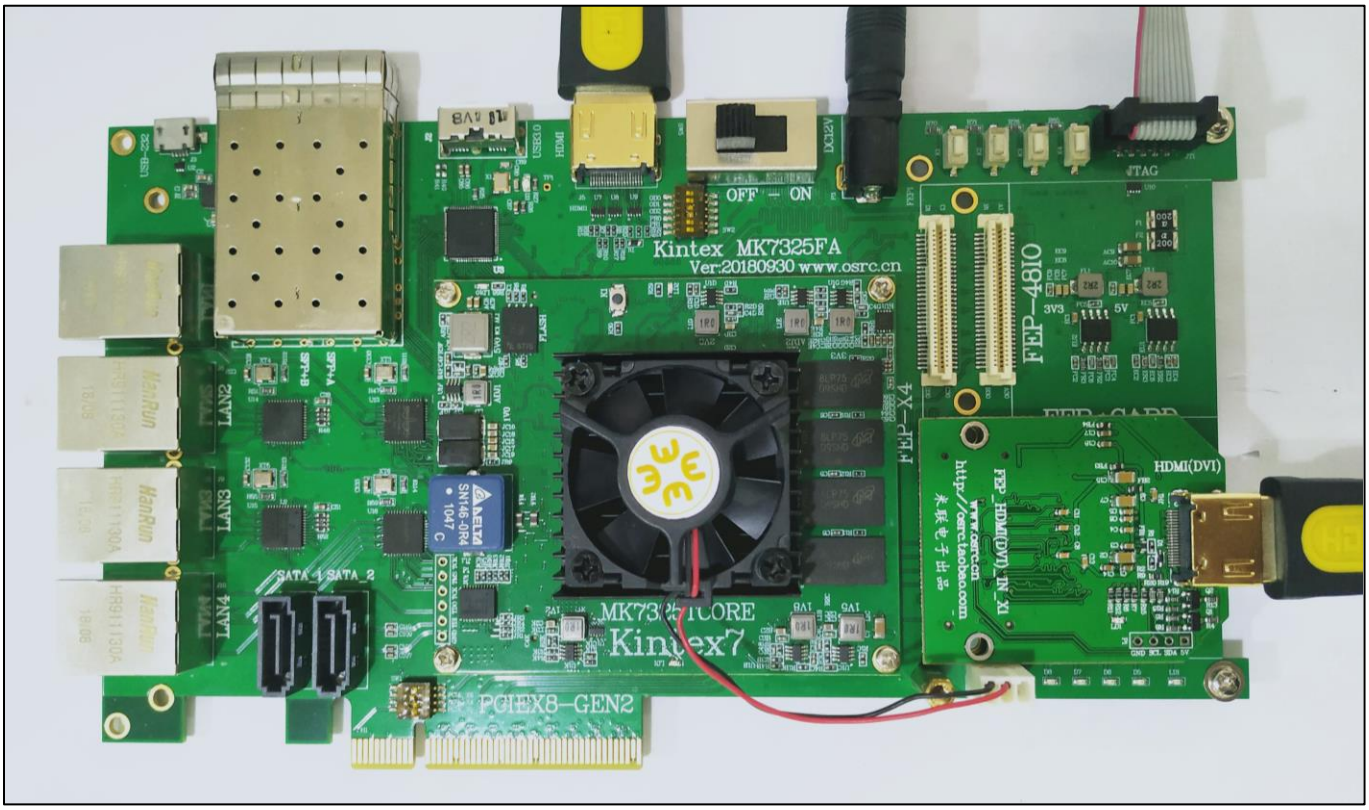
3.3 基于 FDMA 多缓存视频构架 fdma_controller

fdma_controller 的构架也和前面一节课一样，不再重复，只对视频构架中视频输入部分的框图做修改如下图。



3.4 代码叠层结构

3.5 硬件连线


Kintex MK7325FA
Ver:20180930 www.osrc.cn
OFF ON
JTAG
FEP-4810
HDMI(DVI)
SATA_1 SATA_2
PCIEX8-GEN2
Kintex7

3.6 测试结果


12:49
AOC

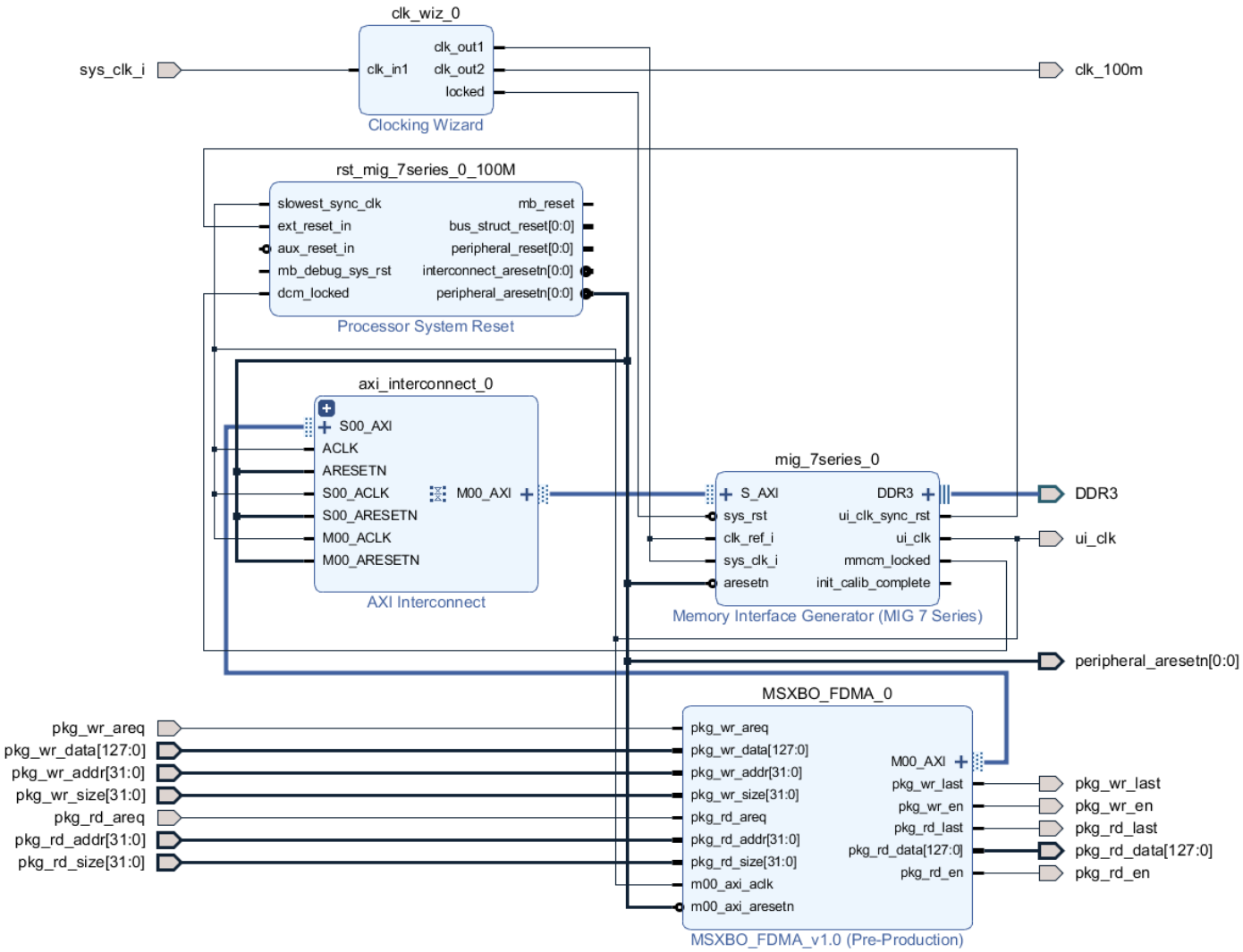
CH04 基于 FDMA 实现 OV5640 摄像头视频采集

软件版本:VIVADO2017.4
操作系统:WIN10
硬件平台:MK7325

4.1 概述

经过上面 CH02 实现了 1080P 测试视频的多缓存构架设计，并且正确测试得到了结果。这一节课，我们将通过 OV5640 摄像头接口采集真实的 720P 视频，然后经过 fdma_controller 的缓存控制机制，将数据经过 FDMA 缓存到 DDR

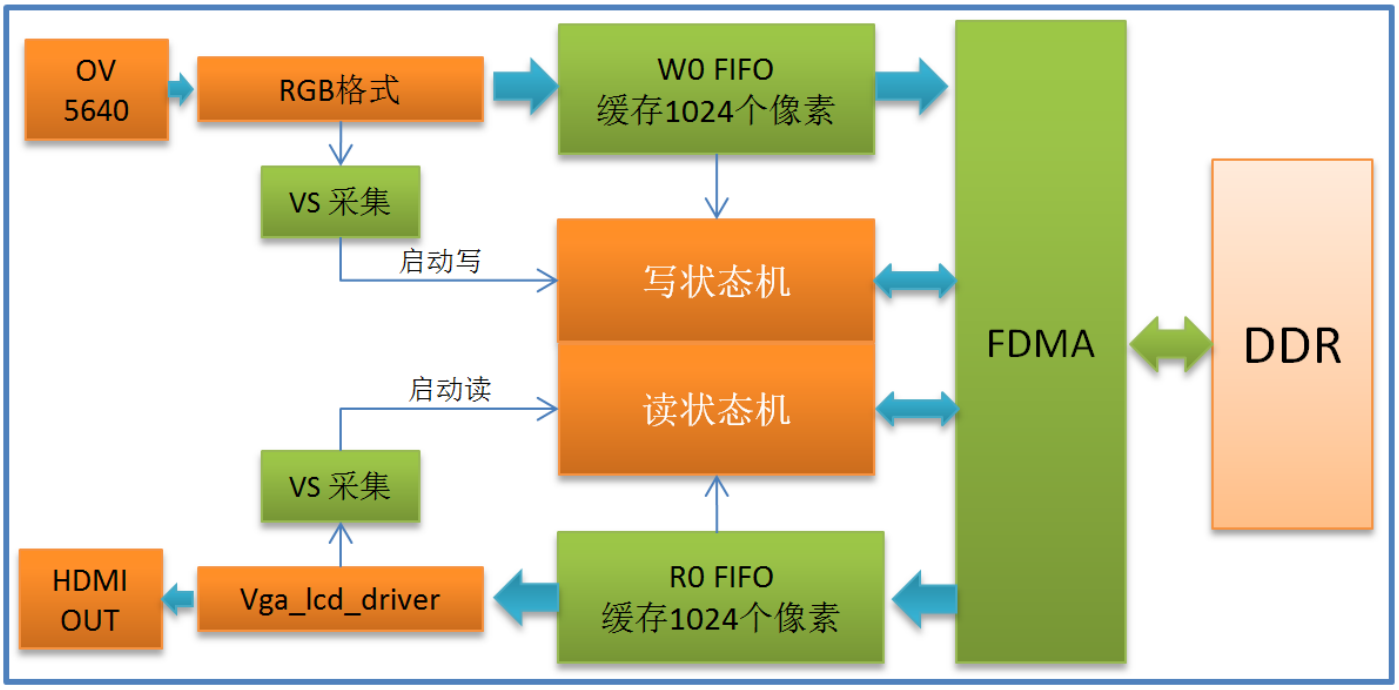
4.2 基于 FDMA 搭建的 BD 工程



这节课的 BD 工程和上一节课一样，所以不再重复叙述

4.3 基于 FDMA 多缓存视频构架 fdma_controller

fdma_controller 的构架也和前面一节课一样，不再重复，只对视频构架中视频输入部分的框图做修改如下图。

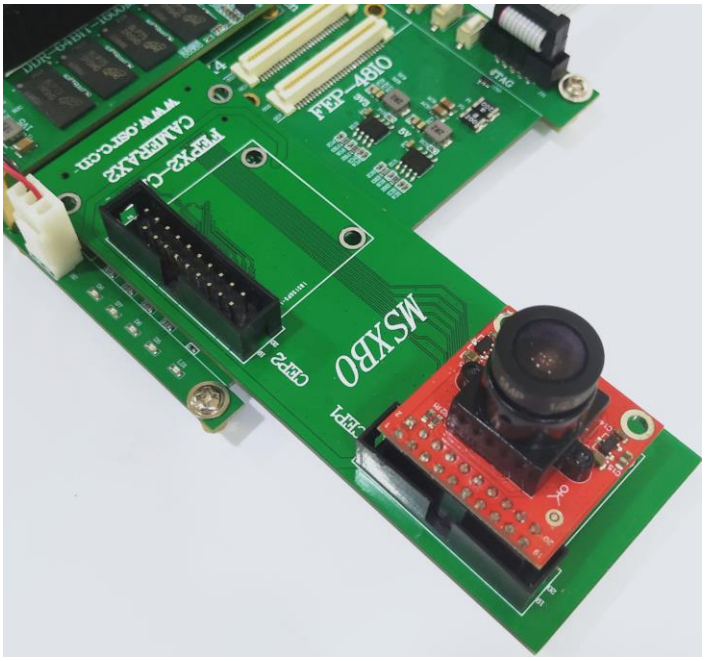


4.4 代码叠层结构

- system_top (system_top.v) (8)
  - clk_hdmi_0 : clk_hdmi (clk_hdmi.xci)
  - MSXBO_OVSensorRGB565_inst : MSXBO_OVSensorRGB565_0 (MSXBO_OVSensorRGB565_0.xci)
  - OV5640IIC_inst : OV5640IIC_0 (OV5640IIC_0.xci)
  - Delay_rst_inst : Delay_rst_0 (Delay_rst_0.xci)
  - vga_lcd_driver_u0 : vga_lcd_driver (vga_lcd_driver.v)
  - u_HDMI1 : HDMI_FPGA_ML_0 (HDMI_FPGA_ML_0.xci)
  - fdma_controller_u0 : fdma_controller (fdma_controller.v) (4)
  - system_i : system (system.bd) (1)
    - system (system.v) (6)
      - MSXBO_FDMA_0 : system_MSXBO_FDMA_0_0 (system_MSXBO_FDMA_0_0.xci)
      - axi_interconnect_0 : system_axi_interconnect_0_0 (system.v) (1)
      - axi_interconnect_0 : system_axi_interconnect_0_0
      - clk_wiz_0 : system_clk_wiz_0_0 (system_clk_wiz_0_0.xci)
      - mig_7series_0 : system_mig_7series_0_0 (system_mig_7series_0_0.xci)
      - rst_mig_7series_0_100M : system_rst_mig_7series_0_100M_0 (system_rst_mig_7series_0_100M_0.xci)

其中新增加了 MSXBO_OVSensorRGB565 为解码程序，OV5640IIC 为摄像头驱动程序。

4.5 摄像头安装



4.6 测试结果